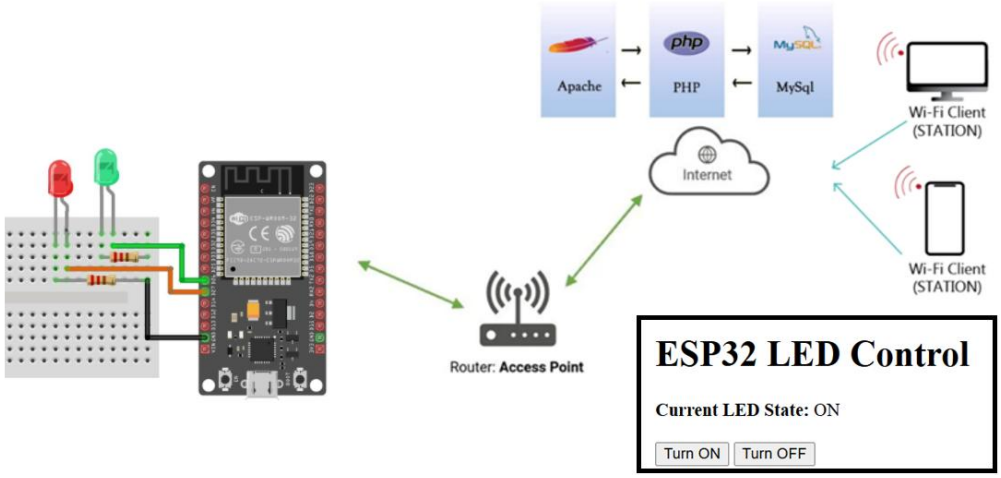
ใบงานที่ 3

การควบคุมการทำงานบอร์ด ESP32 ผ่านบราวเซอร์ และ มัลติทาสกิ้ง (Multitasking)

วัตถุประสงค์

1. เพื่อให้นักศึกษาสามารถพัฒนาโปรแกรมบน ESP32 เพื่อควบคุมอุปกรณ์ (LED) โดยรับข้อมูลสถานะจาก API (เว็บเซิร์ฟเวอร์)
2. เพื่อให้นักศึกษาทราบหลักการของ REST API เบื้องต้น การสื่อสารระหว่าง ESP32 และเซิร์ฟเวอร์
3. เพื่อฝึกการเขียนโปรแกรมเชื่อมต่อฐานข้อมูล MySQL ด้วยภาษา PHP



ส่วนที่ 1: เตรียมฮาร์ดแวร์

1. บอร์ด **ESP32**
2. LED (สีใดก็ได้)
3. ตัวต้านทาน (Resistor) 220Ω – 330Ω
4. สาย Jumper
5. การเชื่อมต่อวงจร (Circuit Connection)
    - ต่อขาขา Anode (ขายาว) ของ LED เข้ากับขา GPIO ของ ESP32 (ตัวอย่างสมมติใช้ขา GPIO 2)
    - ต่อขา Cathode (ขาสั้น) ของ LED เข้ากับตัวต้านทาน แล้วต่อไปยัง GND ของ ESP32

ส่วนที่ 2: เตรียมเว็บเซิร์ฟเวอร์และฐานข้อมูล

ในใบงานนี้จะสมมุติว่าเรามีเว็บเซิร์ฟเวอร์ และได้ติดตั้ง PHP + MySQL พร้อมใช้งานแล้ว

1. **สร้างฐานข้อมูล (Database) ชื่อ** iot_db (ใช้ฐานข้อมูลเดิม)
2. **สร้างตาราง (Table)** ชื่อ **XX_**led_control ภายในฐานข้อมูล iot_db (ปรับชื่อตารางตามชื่อกลุ่ม)

```
CREATE TABLE led_control (
    id INT AUTO_INCREMENT PRIMARY KEY,
    led_state VARCHAR(10) NOT NULL
);
```

- ในตารางนี้เราจะเก็บคอลัมน์ led_state (ค่า “ON” หรือ “OFF”) เพื่อบันทึกสถานะของ LED ที่เราต้องการให้ ESP32 ทำการควบคุม

3.เพิ่มข้อมูลเริ่มต้น

ทำการเพิ่มข้อมูลเริ่มต้นเข้าไปในตารางตามจำนวน LED ที่ต้องการควบคุมและบันทึกหมายเลข ID ของ LED ด้วย

```
INSERT INTO led_control (led_state) VALUES ('OFF');
```

- กำหนดให้สถานะเริ่มต้นเป็น “OFF”

ส่วนที่ 3: สร้างหน้าเว็บ (PHP) และ REST API

เราจะแยกเป็น 2 ไฟล์หลัก ๆ เพื่อความเข้าใจง่าย ได้แก่

1. **ไฟล์สำหรับอัปเดตสถานะ LED** (update_led.php)
2. **ไฟล์สำหรับอ่านสถานะ LED** (get_led_state.php)

3.1 ไฟล์ update_led.php

โค้ดตัวอย่าง (โปรดปรับให้ตรงกับข้อมูลเซิร์ฟเวอร์ของนักศึกษา เช่น host, user, password ของ MySQL)

```
<?php
    // ------------------------------------------------------
    // คำแนะนำ: เก็บค่าการเชื่อมต่อฐานข้อมูลเป็นตัวแปรสภาพแวดล้อม
    // (environment variables) หรือไฟล์ config ที่อยู่นอก web root
    // เพื่อลดความเสี่ยงหากโค้ดถูกเข้าถึงจากภายนอก
    // ------------------------------------------------------
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "iot_db";
```

```
    // สร้างการเชื่อมต่อ
    $conn = new mysqli($servername, $username, $password, $dbname);

    // ตรวจสอบหากมีปัญหาในการเชื่อมต่อ
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    // ตรวจสอบค่า 'state' ที่รับเข้ามา
    if (isset($_REQUEST['state'])) {
      // ดึงค่ามาจาก GET หรือ POST (การรับข้อมูลแบบ $_REQUEST ใน php สามารถรับได้ทั้ง GET และ POST)
        $state = $_REQUEST['state'];

        // กำหนดให้รับได้เฉพาะ "ON" หรือ "OFF" (หากต้องการเพิ่มค่าอื่นก็แก้เงื่อนไขได้)
        if (!in_array($state, ["ON", "OFF"])) {
            echo "Invalid state parameter.";
            exit; // จบการทำงานเพื่อไม่ให้ไปต่อ
        }

        // ใช้ Prepared Statement เพื่อป้องกัน SQL Injection
        $stmt = $conn->prepare("UPDATE led_control SET led_state = ? WHERE id = 1");
        if ($stmt) {
            $stmt->bind_param("s", $state);

            if ($stmt->execute()) {
                echo "OK";
            } else {
                echo "ERROR";
            }

            $stmt->close();
        } else {
            echo "ERROR: Failed to prepare statement.";
        }
    } else {
        echo "No state parameter provided.";
    }

    $conn->close();
?>
```

- เมื่อเราเรียก URL เช่น http://yourserver/update_led.php?state=ON

  จะทำการอัปเดตตาราง led_control ให้มีค่า led_state = "ON"

****ทดสอบ API****

เมื่อเราเรียก URL เช่น http://yourserver/update_led.php?state=ON จะต้องได้สถานะเป็น “OK” และตรวจสอบข้อมูลในตาราง

3.2 ไฟล์ get_led_state.php

```
<?php
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "iot_db";

    // สร้างการเชื่อมต่อ
    $conn = new mysqli($servername, $username, $password, $dbname);

    // ตรวจสอบการเชื่อมต่อ
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    // ใช้ Prepared Statement เช่นกัน แม้ไม่มีตัวแปรภายนอก
    // เพื่อเป็นรูปแบบมาตรฐานที่ปลอดภัย
    $stmt = $conn->prepare("SELECT led_state FROM led_control WHERE id = ?");
    if ($stmt) {
        // ต้องการข้อมูลจาก id=1 เท่านั้น (ปรับค่าตามการออกแบบ)
        $id = 1;
        $stmt->bind_param("i", $id);
        $stmt->execute();

        // ดึงผลลัพธ์จาก stmt
        $result = $stmt->get_result();
        $row = $result->fetch_assoc();

        // สร้างอาร์เรย์สำหรับ JSON response
        $response = array("led_state" => isset($row['led_state']) ? $row['led_state'] : null);

        // กำหนด Content-Type เป็น JSON
        header('Content-Type: application/json');
        echo json_encode($response);

        $stmt->close();
    } else {
        // กรณี prepare statement ไม่สำเร็จ
        echo json_encode(array("error" => "Failed to prepare statement."));
    }

    $conn->close();
?>
```

- เมื่อเรียก URL เช่น http://yourserver/get_led_state.php

ทดสอบ API

- เมื่อเราเรียก URL เช่น http://yourserver/get_led_state.php ได้ข้อมูล JSON ประมาณนี้:

```
{ "led_state": "OFF" }
```

ข้อแนะนำด้านความปลอดภัยเพิ่มเติม

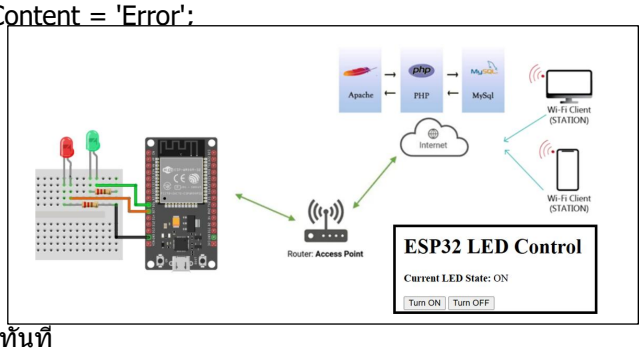
1. แยกเก็บข้อมูลการเชื่อมต่อฐานข้อมูล
    - ควรเก็บ servername, username, password, dbname ไว้ในไฟล์คอนฟิกหรือไฟล์ .env ที่อยู่ นอก โฟลเดอร์เว็บ (web root) แล้วค่อย include ในไฟล์ PHP ทั้งสองแทน
    - ป้องกันกรณีโค้ดถูกเข้าถึงโดยตรง จะได้ไม่เห็น Credentials สำคัญ
2. กรณีต้องการให้ใช้งานเฉพาะผู้ได้รับอนุญาต
    - อาจเพิ่ม Token หรือ API Key เพื่อตรวจสอบก่อนอนุญาตให้เรียก update_led.php เช่น ต้องแนบ header Authorization: Bearer <token> หรือส่งค่า api_key ที่ถูกต้องมากับ request
3. จัดการ Error Logging
    - หากเป็น Production อาจซ่อน detailed error (เช่น mysqli_connect_error()) และใช้การ Log ลงไฟล์แทน เพื่อป้องกันการเปิดเผยโครงสร้างภายในให้ผู้อื่นทราบ
4. Encryption (SSL/TLS)
    - หากเป็นการควบคุมอุปกรณ์ผ่าน Internet จริง ควรติดตั้ง SSL เพื่อให้การสื่อสารเป็น HTTPS ลดความเสี่ยงในการถูกดักฟังข้อมูล
5. Validate Input ฝั่งไคลเอ็นต์
    - นอกจากการ Validate ฝั่งเซิร์ฟเวอร์แล้ว ควรมีการ Validate ฝั่งหน้าเว็บหรือ ESP32 (ฝั่งไคลเอ็นต์) เพื่อเตือนผู้ใช้หรือโปรแกรมเมอร์ก่อนส่งค่า (แม้สุดท้ายยังต้อง Validate ฝั่งเซิร์ฟเวอร์เสมอ)

3.3 หน้าเว็บสำหรับควบคุม (หน้า UI)

สร้างหน้าเว็บ PHP หรือ HTML ง่าย ๆ ให้ผู้ใช้กดเลือก ON/OFF และส่งไปยัง update_led.php


```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>ESP32 LED Control</title>
</head>
<body>
  <h1>ESP32 LED Control</h1>
```

```
  <div>
    <strong>Current LED State:</strong>
    <span id="ledState">Loading...</span>
  </div>
  <br>
  <button onclick="updateState('ON')">Turn ON</button>
  <button onclick="updateState('OFF')">Turn OFF</button>

  <script>
    // ฟังก์ชันดึงสถานะ LED จาก get_led_state.php
    function fetchLEDState() {
      fetch('get_led_state.php')
        .then(response => response.json())
        .then(data => {
          // กำหนด Text ให้กับ span ที่มี id="ledState"
          document.getElementById('ledState').textContent = data.led_state;
        })
        .catch(error => {
          console.error('Error fetching LED state:', error);
          document.getElementById('ledState').textContent = 'Error';
        });
    }

    // ฟังก์ชันสั่งให้ LED ON / OFF
    function updateState(state) {
      fetch(`update_led.php?state=${state}`)
        .then(response => response.text())
        .then(result => {
          console.log('Update result:', result);
          // เมื่ออัปเดตเสร็จ อาจเรียก fetchLEDState() ทันที
          fetchLEDState();
        })
        .catch(error => {
          console.error('Error updating LED state:', error);
        });
    }

    // เมื่อโหลดหน้าเว็บเสร็จ ทำ 2 อย่าง:
    // 1) เรียก fetchLEDState() ครั้งแรกเพื่อแสดงสถานะเริ่มต้น
    // 2) ตั้ง setInterval เรียก fetchLEDState() ทุก 5 วินาที (5000 ms)
    window.onload = function() {
      fetchLEDState();
      setInterval(fetchLEDState, 5000); // 5000 ms = 5 วินาที
    }
  </script>
</body>
</html>
```



อธิบายเพิ่มเติม

1. fetchLEDState()
    - ทำการ fetch('get_led_state.php') ซึ่งจะส่งค่ากลับมาในรูปแบบ JSON
    - จากนั้นจะนำค่าจาก key ชื่อ led_state มาแสดงใน ...
2. updateState(state)
    - เมื่อผู้ใช้กดปุ่ม ON/OFF จะส่ง request ไปยัง update_led.php?state=ON หรือ ...OFF
    - เมื่ออัปเดตเสร็จจะเรียก fetchLEDState() เพื่อดึงสถานะล่าสุดมาแสดงทันที
3. setInterval(fetchLEDState, 5000)
    - ฟังก์ชัน fetchLEDState() จะถูกเรียกทำงานซ้ำทุก ๆ 5 วินาทีโดยอัตโนมัติ
    - หาก LED ถูกเปลี่ยนสถานะจากแหล่งอื่น หรือ ESP32 เขียนค่าเปลี่ยนเองในฐานข้อมูล ก็จะแสดงสถานะใหม่ที่ถูกต้องบนหน้าเว็บ
4. ปรับช่วงเวลา (interval)
    - หากต้องการให้รีเฟรชถี่กว่านี้หรือนานกว่านี้ ให้แก้เลข 5000 เป็นหน่วย มิลลิวินาที ตามความเหมาะสม เช่น ทุก 2 วินาที (2000 ms) หรือทุก 10 วินาที (10000 ms)
5. ความปลอดภัย
    - หากต้องการรับเฉพาะผู้ใช้ที่อนุญาตเท่านั้น อาจเพิ่ม Token หรือ API Key หรือใช้ระบบล็อกอินเพิ่มเติม
    - หากต้องการความปลอดภัยในการส่งข้อมูลจริงจัง ควรใช้ HTTPS (SSL/TLS)

ส่วนที่ 4: เขียนโค้ดบน ESP32 (Arduino IDE หรือ PlatformIO)

ต่อไปเป็นตัวอย่างโค้ดใน Arduino IDE เพื่อให้ ESP32:

1. เชื่อมต่อ Wi-Fi
2. ใช้ HTTP GET ไปขอ JSON จาก get_led_state.php
3. ตรวจสอบว่าค่า led_state ใน JSON เป็น “ON” หรือ “OFF”
4. สั่งเปิด-ปิด LED (GPIO 2)

ตัวอย่างโค้ด (ปรับ SSID, PASSWORD, และ URL ของเซิร์ฟเวอร์)

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

```
// ตั้งค่าขา LED
#define LED_PIN 2

// ตั้งค่า WiFi
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// URL สำหรับดึงสถานะ LED จาก server
String serverName = "http://yourserver/get_led_state.php";

// ตัวจับเวลาเพื่อไม่ให้เรียก API ถี่เกินไป
unsigned long previousMillis = 0;
const long interval = 5000; // ดึงข้อมูลทุก 5 วินาที

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

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW); // เริ่มต้นปิด LED

  // เชื่อมต่อ WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");
}

void loop() {
  unsigned long currentMillis = millis();

  // ถ้าถึงเวลาที่กำหนด (5 วินาทีต่อครั้ง)
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;
      http.begin(serverName); // เริ่มต้นการ request
      int httpResponseCode = http.GET(); // ส่ง GET

      if (httpResponseCode == 200) {
        String response = http.getString();
        Serial.println("Server Response: " + response);

        // เราคาดว่า response เป็น JSON เช่น {"led_state":"ON"} หรือ {"led_state":"OFF"}
        // การ parse JSON เพื่อใช้งานจริงอาจใช้ ArduinoJson
        // แต่ถ้าเป็นตัวอย่างง่าย ๆ สามารถค้นหาคีย์เวิร์ดได้

        if(response.indexOf("\"led_state\":\"ON\"") >= 0) {
          // สั่งเปิด LED
```

```
        digitalWrite(LED_PIN, HIGH);
        Serial.println("LED -> ON");
      }
      else {
        // สั่งปิด LED
        digitalWrite(LED_PIN, LOW);
        Serial.println("LED -> OFF");
      }
    }
    else {
      Serial.print("Error in HTTP request, code: ");
      Serial.println(httpResponseCode);
    }
    http.end(); // ปิดการเชื่อมต่อ
  }
  else {
    Serial.println("WiFi Disconnected");
  }
 }
}
```

ส่วนที่ 5: เพิ่ม LED ตัวที่สอง และปรับปรุงการควบคุม

ในส่วนนี้ ให้นักศึกษาเพิ่ม LED เป็น 2 ดวง (เช่น LED1, LED2) และแก้ไขตารางฐานข้อมูล ตลอดจนโค้ด PHP และโค้ดบน ESP32 ให้รองรับการควบคุม LED สองตัวได้อิสระ ผ่านหน้าเว็บ

****หมายเหตุ**** นศ. สามารถปรับแก้ API เดิม (update_led.php, get_led_state.php) เพื่อให้รองรับการควบคุม LED ที่มากขึ้นได้โดย

ตัวอย่าง การเพิ่มคาพารามิเตอร์ในการควบคุม (update_led.php) เช่น

```
update_led.php?id=1&state =OFF
```

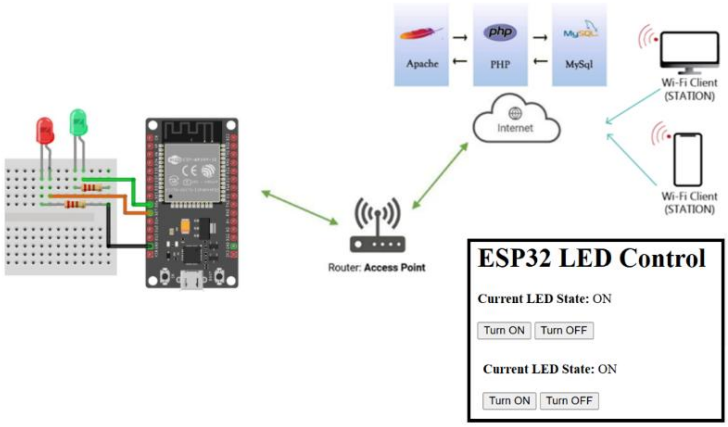
ตัวอย่าง การส่งค่าสถานะ LED1 และ LED 2 (get_led_state.php)

```
// ส่ง JSON เก็บ led1_state, led2_state
      $response = array(
         "led1_state" => $row["led1_state"],
         "led2_state" => $row["led2_state"]
      );
      echo json_encode($response);
```

ตัวอย่าง โค้ด ESP32

```
    String payload = http.getString();
    Serial.println("Server payload: " + payload);
      // ตัวอย่าง JSON: {"led1_state":"ON","led2_state":"OFF"}
      // หากต้องการ Parse จริงจัง ใช้ ArduinoJson library
      // ที่นี่ตัวอย่างง่าย ๆ: ตรวจสอบ substring
      if (payload.indexOf("\"led1_state\":\"ON\"") >= 0) {
        digitalWrite(LED1_PIN, HIGH);
      } else {
```

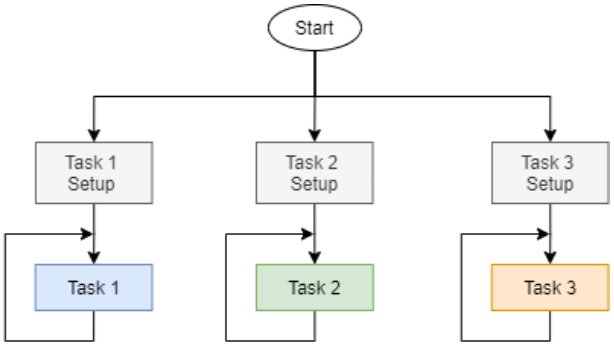
```
    digitalWrite(LED1_PIN, LOW);
  }

  if (payload.indexOf("\"led2_state\":\"ON\"") >= 0) {
    digitalWrite(LED2_PIN, HIGH);
  } else {
    digitalWrite(LED2_PIN, LOW);
  }
```



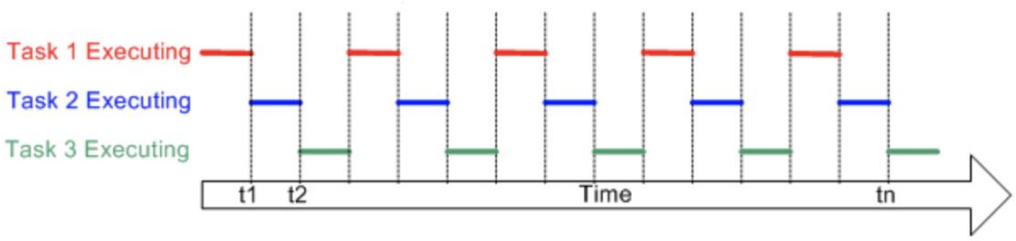
ส่วนที่ 6: มัลติทาสกิ้ง (Multitasking)

มัลติทาสกิ้ง (Multitasking) คือ การที่ระบบคอมพิวเตอร์สามารถบริหารจัดการให้มีการทำงานหลายงาน (Tasks หรือ Processes) เกิดขึ้นพร้อมกันหรือสลับกันในลักษณะที่ผู้ใช้รับรู้เสมือนว่าทุกงานกำลังทำงานพร้อมกัน ในทางเทคนิค ระบบปฏิบัติการ (Operating System) หรือเฟรมเวิร์กที่มีความสามารถในการจัดสรรทรัพยากร (เช่น เวลา CPU, หน่วยความจำ, อุปกรณ์อินพุต/เอาต์พุต) จะทำการ “สลับ” (Context Switch) ระหว่างงานต่าง ๆ อย่างรวดเร็ว จนผู้ใช้หรือโปรแกรมอื่น ๆ เห็นว่าแต่ละงานสามารถทำงาน “พร้อมกัน” ได้

FreeRTOS.h เป็น **Header File** ของระบบปฏิบัติการเรียลไทม์ (Real-Time Operating System - RTOS) สำหรับไมโครคอนโทรลเลอร์ ชื่อ **FreeRTOS** โดยใช้สำหรับสร้างและจัดการ **Multitasking** บนอุปกรณ์ฝังตัว (Embedded Systems) เช่น ESP32, ESP8266 หรือ Arduino (http://www.openrtos.net/RTOS.html)



Basic RTOS Structure



RTOS Execution Timeline

ขั้นตอนการติดตั้ง FreeRTOS ใน Arduino IDE

1. สำหรับ ESP32/ESP8266

FreeRTOS ถูกติดตั้งมาแล้วใน **ESP32 Arduino Core** ไม่ต้องติดตั้งเพิ่ม

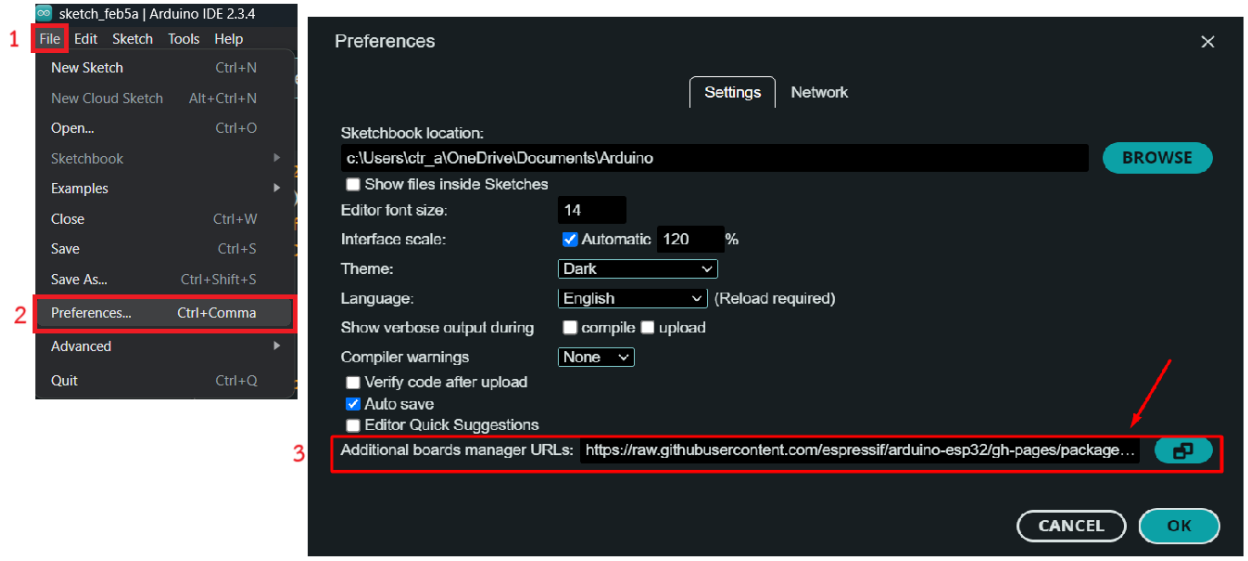
- **ขั้นตอน**:
    1. ติดตั้ง **ESP32 Board Package** ใน Arduino IDE:
        - ไปที่ **File > Preferences**
        - วางลิงก์นี้ใน **Additional Boards Manager URLs**:

ESP32

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

ESP8266

http://arduino.esp8266.com/stable/package_esp8266com_index.json



2. ติดตั้ง **ESP32 Board**:
    - ไปที่ **Tools > Board > Boards Manager**
    - ค้นหา **ESP32** แล้วกด **Install**

****หมายเหตุ***** หากใช้บอร์ด ESP32 กับ Arduino IDE หรือ PlatformIO โดยทั่วไปจะมี FreeRTOS ติดมาอยู่แล้วใน Core ของ ESP32 (เพราะ ESP32 ใช้ FreeRTOS เป็นเคอร์เนลหลัก)

ฟังก์ชันหลักใน FreeRTOS.h

1. การสร้าง Task (งาน)

ใช้ฟังก์ชัน xTaskCreate() เพื่อสร้าง Task ใหม่:

```
xTaskCreate(
  function_name,  // ชื่อฟังก์ชันของ Task
  "Task Name",    // ชื่อ Task (สำหรับดีบัก)
  stack_size,     // ขนาด Stack (หน่วย: bytes สำหรับ ESP32, words สำหรับ AVR)
  parameters,     // พารามิเตอร์ที่ส่งไปให้ Task (ถ้าไม่ใช้ให้ใส่ NULL)
  priority,       // ความสำคัญ (0-24 สำหรับ ESP32, 1-3 สำหรับ AVR)
  task_handle     // ตัวแปรสำหรับอ้างอิง Task (ถ้าไม่ใช้ให้ใส่ NULL)
);
```

ตัวอย่าง:

```
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#define LED1 2

void myTask(void *pvParam) {
  while(1) {
    digitalWrite(LED1, !digitalRead(LED_BUILTIN));
    vTaskDelay(1000 / portTICK_PERIOD_MS); // ดีเลย์ 1 วินาที
  }
}

void setup() {
  xTaskCreate(myTask, "LED Task", 2048, NULL, 1, NULL);
}
```

2. การดีเลย์แบบไม่บล็อกการทำงาน

ใช้ vTaskDelay() แทน delay() เพื่อไม่ให้ Task อื่นถูกบล็อก:

```
vTaskDelay(เวลา / portTICK_PERIOD_MS);
```

- ตัวอย่าง: vTaskDelay(500 / portTICK_PERIOD_MS); = ดีเลย์ 500 มิลลิวินาที

3. การจัดการความสำคัญ (Priority)

- **Priority สูง** = ได้รับ CPU บ่อยกว่า
- **ESP32**: 0 (ต่ำสุด) ถึง 24 (สูงสุด)
- **AVR Arduino**: 1 (ต่ำสุด) ถึง 3 (สูงสุด)

ตัวอย่างโค้ด ESP32 แบบ Multitasking

```
#include <Arduino.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>

void task1(void *pvParam) {
  while(1) {
    printf("Task 1 running\n");
    vTaskDelay(1000 / portTICK_PERIOD_MS); // 1-second delay
  }
}

void task2(void *pvParam) {
  while(1) {
    printf("Task 2 running\n");
    vTaskDelay(500 / portTICK_PERIOD_MS); // 500ms delay
  }
}

void setup() {
  Serial.begin(115200);
  xTaskCreate(task1, "LED Task", 2048, NULL, 1, NULL);
  xTaskCreate(task2, "Serial Task", 2048, NULL, 1, NULL);
}

void loop() {
  // FreeRTOS tasks run independently of loop()
}
```

เปิด **Serial monitor เพื่อดูผลการทำงานของโปรแกรม***

ปัญหาที่พบบ่อยและวิธีแก้ไข

1. **Task ไม่ทำงาน**:
    - ตรวจสอบ **Stack Size** (เพิ่มขนาดถ้าไม่พอ)
    - ตรวจสอบ **Priority** (Task อื่นอาจมีความสำคัญสูงกว่า)
2. **ระบบค้าง**:
    - ****อย่าใช้ delay() ใน Task ให้ใช้ vTaskDelay() แทน****
    - ตรวจสอบ **Deadlock** จาก Semaphore/Mutex

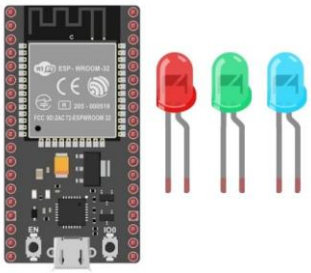
3. **RAM ไม่พอ**:
    - ลดขนาด Stack ของ Task
    - ใช้ **Static Allocation** สำหรับตัวแปร

สรุป

- **FreeRTOS.h** ช่วยจัดการ Multitasking บนไมโครคอนโทรลเลอร์ได้อย่างมีประสิทธิภาพ
- **ESP32/ESP8266** มี FreeRTOS ในตัว ส่วน **Arduino บอร์ดอื่น** ต้องติดตั้ง Library เพิ่ม
- ใช้ xTaskCreate(), vTaskDelay() และ Synchronization Tools (Queue, Semaphore) เพื่อควบคุม Task

ส่วนที่ 7: เพิ่ม LED ตัวที่สาม และปรับปรุงการทำงาน

ให้นักศึกษาต่อ LED ตัวที่ 3 โดยให้ LED ตัวนี้กระพริบ ติด-ดับ ทุกๆ 1 วินาที (1000 ms)



ส่วนที่ 8: การทดสอบและการส่งงาน

1. **ทดสอบหน้าเว็บ**
    - เปิดหน้าเว็บ (เช่น control.html หรือไฟล์ PHP UI ที่สร้าง)
    - กดปุ่ม Turn ON / OFF แล้วตรวจสอบค่าที่ตาราง led_control ในฐานข้อมูล MySQL เพื่อดูว่าค่าถูกอัปเดตถูกต้องหรือไม่
2. **ทดสอบ ESP32**
    - เปิด Serial Monitor ดูข้อความ Log
    - เมื่อกดปุ่มบนหน้าเว็บเพื่อเปลี่ยนเป็น ON หรือ OFF สังเกตว่า ESP32 รับค่าแล้วมีการเปลี่ยนสถานะ LED หรือไม่
    - หลอด LED ตัวที่ 3 ติดดับทุก ๆ 1 วินาที
3. **จับภาพหน้าจอ** เพื่อยืนยันการทำงาน
4. **สรุป** กระบวนการทำงาน ปัญหาที่พบ และวิธีแก้ไข

สิ่งที่ต้องส่ง

1. ไฟล์โค้ด Arduino (ไฟล์ .ino) ที่ใช้งานบน ESP32
2. ไฟล์หน้าเว็บ PHP/HTML และไฟล์ API (update_led.php, get_led_state.php)
3. รูปภาพแสดงการทำงานจริงของระบบ (แสดงให้เห็นว่ากดปุ่ม ON/OFF ที่หน้าเว็บแล้ว LED บน ESP32 เปลี่ยนสถานะตาม)
4. สรุปผลการทดลอง ปัญหาและวิธีการแก้ไข