การใช้งาน **PWM** กับ **ESP32**

ในบทนี้ เราจะมาเรียนรู้วิธีปรับความสว่างของ LED โดยใช้ PWM (Pulse-Width Modulation) โดยจะมีวิธีการ 2 แบบที่เราจะอธิบายให้เข้าใจง่าย:

1. การใช้คำสั่ง **analogWrite**
2. การใช้ **LEDC Controller** ของ **ESP32**

Pulse-Width Modulation (PWM)

พิน GPIO ของ ESP32 สามารถส่งแรงดันไฟฟ้าออกมาได้แค่ 0V หรือ 3.3V เท่านั้น และไม่สามารถส่งแรงดันไฟฟ้าระดับกลาง ๆ ออกมาได้ (ยกเว้นพิน DAC บางพิน)

อย่างไรก็ตาม เราสามารถสร้างแรงดันไฟฟ้าระดับกลางแบบ "จำลอง" ได้ด้วยเทคนิคที่เรียกว่า **Pulse-Width Modulation (PWM)** โดยใช้วิธีการเปิด-ปิดสัญญาณไฟฟ้าเป็นช่วงเวลา (pulse)

ในโปรเจกต์นี้ เราจะใช้ PWM เพื่อปรับระดับความสว่างของ LED ให้มีความหลากหลายตามต้องการ

การทำงานของ **PWM** เพื่อปรับความสว่างของ **LED**

หากเราสลับแรงดันไฟฟ้าที่ส่งไปยัง LED ระหว่าง **HIGH (3.3V)** และ **LOW (0V)** ด้วยความเร็วสูงมาก จนตาของเราไม่สามารถแยกแยะความเร็วในการเปิด-ปิดของ LED ได้ ตาของเราจะเห็นเพียงแค่ระดับความสว่างที่แตกต่างกัน

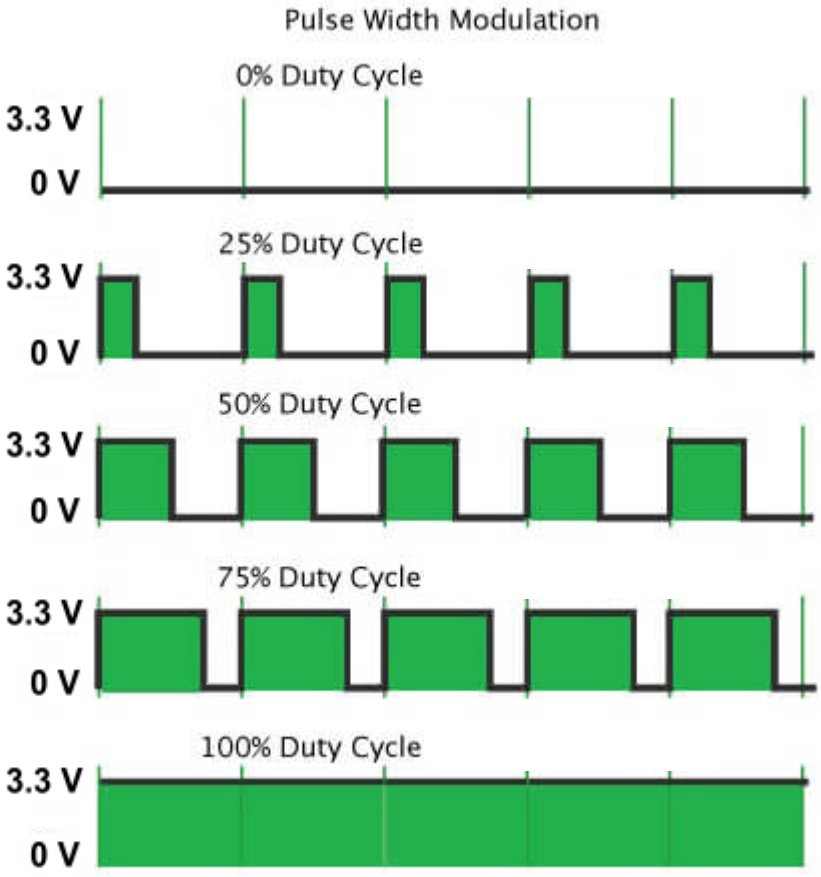
นี่คือหลักการทำงานของ **PWM** (Pulse-Width Modulation) โดยสัญญาณที่ส่งออกมาจะเปลี่ยนระหว่าง **HIGH** และ **LOW** ด้วยความถี่ที่สูงมาก

สิ่งสำคัญใน PWM คือ **Duty Cycle**

- **Duty Cycle** คือ สัดส่วนของเวลาที่ LED อยู่ในสถานะ **HIGH** (เปิด) เมื่อเทียบกับรอบเวลาเต็มหนึ่งรอบ

- หากเวลาที่อยู่ในสถานะ **HIGH** มากกว่า LED จะดูสว่างขึ้น แต่ถ้าเวลาที่อยู่ในสถานะ **LOW** มากกว่า LED จะดูหรี่ลง

ภาพด้านล่างนี้จะแสดงตัวอย่างการทำงานของ PWM เพื่อให้เข้าใจได้ง่ายขึ้น



ตัวอย่างการทำงานของ **Duty Cycle** ใน **PWM**

- **Duty Cycle 50%**: หมายถึง LED จะเปิดในครึ่งหนึ่งของเวลาในแต่ละรอบ ซึ่งจะทำให้ LED มีความสว่างประมาณ 50%
- **Duty Cycle 0%**: หมายถึง LED ปิดตลอดเวลา หรือไม่สว่างเลย
- **Duty Cycle 100%**: หมายถึง LED เปิดตลอดเวลา หรือสว่างเต็มที่

การปรับ **Duty Cycle** นี้เองที่ทำให้เราสามารถสร้างระดับความสว่างของ LED ที่แตกต่างกันได้

ตัวควบคุม **PWM** ใน **ESP32**

ESP32 มีตัวควบคุม PWM สำหรับ LED (LED PWM Controller) ที่สามารถใช้งานได้ตั้งแต่ **6** ถึง **16** ช่องสัญญาณ (ขึ้นอยู่กับรุ่นของ ESP32) ตัวควบคุมนี้สามารถตั้งค่าช่องสัญญาณแต่ละช่องเพื่อสร้างสัญญาณ PWM ที่มีคุณสมบัติต่างกันได้

มีหลายวิธีที่เราสามารถใช้เพื่อสร้างสัญญาณ PWM และได้ผลลัพธ์เดียวกัน เช่น

1. ใช้ฟังก์ชัน **analogWrite()** (เหมือนกับในบอร์ด Arduino)
2. ใช้ฟังก์ชัน **LEDC**

ฟังก์ชัน **analogWrite()**

ฟังก์ชันที่ง่ายที่สุดสำหรับการสร้างสัญญาณ PWM คือ **analogWrite()** ซึ่งต้องใส่พารามิเตอร์ 2 ค่า:

1. **GPIO pin**: พินที่เราต้องการสร้างสัญญาณ PWM
2. **Duty Cycle**: ค่า Duty Cycle (ช่วง 0 ถึง 255)

รูปแบบคำสั่ง:

```
analogWrite(pin, value);
```

ตัวอย่าง

คำสั่งด้านล่างจะสร้างสัญญาณ PWM ที่พิน GPIO 2 และตั้งค่า Duty Cycle ไว้ที่ 150

```
analogWrite(2, 150);
```

การตั้งค่าความถี่ **(Frequency)** และความละเอียด **(Resolution)** ของสัญญาณ **PWM**

เราสามารถกำหนด ความละเอียด **(Resolution)** และ ความถี่ **(Frequency)** ของสัญญาณ PWM ที่สร้างขึ้นบนพินที่เลือกได้ โดยใช้ฟังก์ชันดังนี้:

การตั้งค่าความละเอียด **(Resolution)**

ฟังก์ชัน **analogWriteResolution()** ใช้กำหนดความละเอียดของสัญญาณ PWM

- **Resolution** หมายถึงจำนวนระดับค่า Duty Cycle ที่สามารถตั้งค่าได้ ยิ่งค่าความละเอียดสูงขึ้น การปรับระดับ Duty Cycle ก็จะยิ่งละเอียดขึ้น

รูปแบบคำสั่ง:

```
analogWriteResolution(pin, resolution);
```

การตั้งค่าความถี่ **(Frequency)**

ฟังก์ชัน **analogWriteFrequency()** ใช้กำหนดความถี่ของสัญญาณ PWM

- **Frequency** คือจำนวนครั้งที่สัญญาณ PWM ทำงาน (HIGH และ LOW) ใน 1 วินาที โดยมีหน่วยเป็น Hertz (Hz)

รูปแบบคำสั่ง:

```
analogWriteFrequency(pin, freq);
```

การตั้งค่าพิน **LEDC**

ก่อนที่จะเริ่มใช้งาน คุณต้องกำหนดพินที่ต้องการใช้สำหรับ LEDC ด้วยฟังก์ชันดังนี้:

1. **ledcAttach()**
   ใช้สำหรับเชื่อมต่อพินกับช่องสัญญาณ PWM

2. **ledcAttachChannel()**
   ใช้กำหนดพินและเชื่อมต่อกับช่องสัญญาณ รวมถึงตั้งค่าคุณสมบัติพื้นฐานสำหรับการสร้างสัญญาณ PWM

ฟังก์ชัน **ledcAttach**

ฟังก์ชัน **ledcAttach** ใช้สำหรับตั้งค่าพิน GPIO เพื่อสร้างสัญญาณ PWM โดยระบุค่าความถี่ (**frequency**) และความละเอียด (**resolution**) ของสัญญาณ PWM

```
bool ledcAttach(uint8_t pin, uint32_t freq, uint8_t
resolution);
```

pin: พิน GPIO ที่ต้องการใช้สร้างสัญญาณ PWM

freq: ความถี่ของสัญญาณ PWM (หน่วยเป็น Hz)

resolution: ความละเอียดของ Duty Cycle (หน่วยเป็นบิต เช่น 8, 10, 12 บิต เป็นต้น)

ผลลัพธ์:

- **true**: การตั้งค่าสำเร็จ
- **false**: เกิดข้อผิดพลาด และช่องสัญญาณ LEDC ไม่ถูกตั้งค่า

ฟังก์ชัน **ledcWrite**

หลังจากตั้งค่าพิน LEDC ด้วยฟังก์ชันก่อนหน้าแล้ว (เช่น ledcAttach หรือ ledcAttachChannel), คุณสามารถใช้ฟังก์ชัน **ledcWrite** เพื่อกำหนดค่า Duty Cycle ของสัญญาณ PWM ได้

```
void ledcWrite(uint8_t pin, uint32_t duty);
```

- **pin**: พิน GPIO ที่ต้องการควบคุม

- **duty**: ค่า Duty Cycle ที่ต้องการตั้ง (ขึ้นอยู่กับค่าความละเอียดที่กำหนด เช่น 0-255 สำหรับ 8 บิต หรือ 0-1023 สำหรับ 10 บิต)

การปรับความสว่างของ **LED (Dimming an LED)**

ในส่วนนี้ เราจะมาสร้างตัวอย่างง่าย ๆ เพื่อแสดงการสร้างสัญญาณ PWM ด้วย ESP32 เพื่อปรับความสว่างของ LED โดยจะมี 2 ตัวอย่าง:

1. การใช้ฟังก์ชัน `analogWrite()`
2. การใช้ฟังก์ชัน LEDC

การต่อวงจร

อุปกรณ์ที่ต้องใช้:

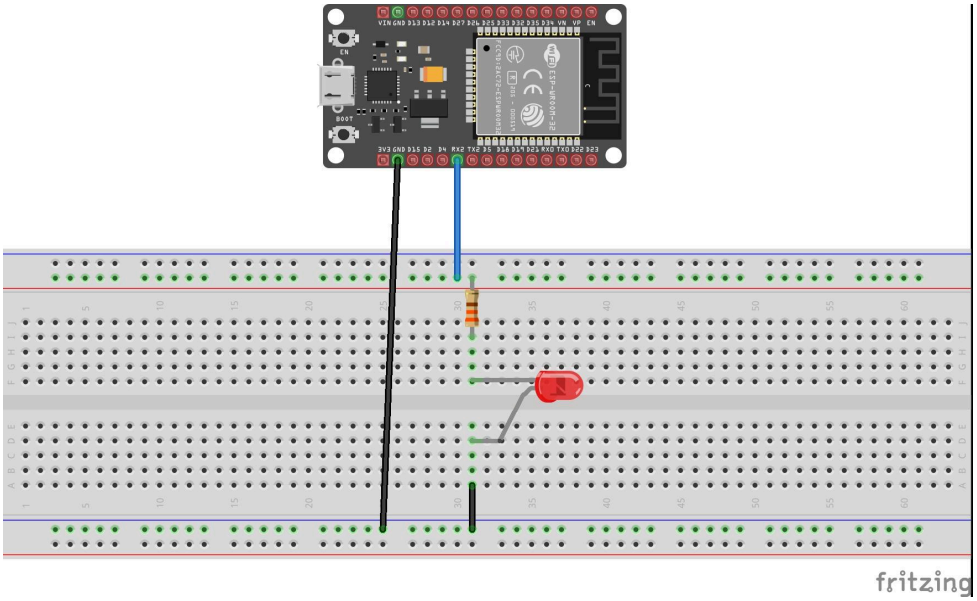
1. **ESP32 DOIT DEVKIT V1 Board**
2. หลอด **LED** ขนาด **5mm**
3. ตัวต้านทาน **330** โอห์ม (หรือค่าที่ใกล้เคียง)
4. **Breadboard**
5. สายจัมเปอร์

การต่อวงจร:

1. ต่อขาของหลอด LED
    - ขา ขั้วบวก (ขาที่ยาวกว่า) ต่อเข้ากับ ตัวต้านทาน **330** โอห์ม
    - ขา ขั้วลบ (ขาสั้นกว่า) ต่อเข้ากับ **GND (Ground)** ของ ESP32

2. ต่อขั้วของตัวต้านทาน
    - ต่อปลายตัวต้านทานอีกด้านเข้ากับ **GPIO 16** ของ ESP32

3. ตรวจสอบการเชื่อมต่อให้ถูกต้องตาม แผนผังวงจร



แผนผังการต่อวงจร **(Schematic Diagram):**

- **GPIO 16** → ตัวต้านทาน → ขั้วบวกของ LED
- ขั้วลบของ LED → GND

ข้อควรรู้:

- พิน GPIO ที่รองรับ เอาต์พุต บน ESP32 สามารถใช้เป็น PWM ได้ทุกพิน
- เลือกพินที่เหมาะสมกับการออกแบบวงจรของคุณ

ตัวอย่างโค้ด **ESP32 PWM** ด้วยฟังก์ชัน **analogWrite** (ไฟล์ **PWM_analogWrite**)

เปิด **Arduino IDE** แล้วคัดลอกโค้ดต่อไปนี้ลงในโปรเจกต์ของคุณ โค้ดนี้จะทำให้ LED ค่อย ๆ สว่างขึ้นและมืดลงตามลำดับด้วยการปรับค่า Duty Cycle ผ่านฟังก์ชัน `analogWrite()`:

```
const int ledPin = 16; // กำหนดพิน GPIO ที่ต่อกับ LED
```

```
void setup() {
  pinMode(ledPin, OUTPUT); // ตั้งค่าให้ GPIO เป็นเอาต์พุต
}

void loop() {
  // เพิ่มความสว่างของ LED
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    analogWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle
    delay(10); // หน่วงเวลา 10 มิลลิวินาที
  }

  // ลดความสว่างของ LED
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    analogWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle
    delay(10); // หน่วงเวลา 10 มิลลิวินาที
  }
}
```

การอธิบายโค้ดโดยละเอียด

1. การกำหนดค่าตัวแปร

```
const int ledPin = 16; // กำหนดพิน GPIO ที่ต่อกับ LED
```

- **`const`**: ใช้สำหรับสร้างค่าคงที่ที่ไม่สามารถเปลี่ยนแปลงได้ในระหว่างการทำงานของโปรแกรม

- **`int ledPin = 16;`**: ระบุว่าพิน GPIO หมายเลข 16 ของ ESP32 จะถูกใช้สำหรับควบคุม LED

2. ฟังก์ชัน **`setup()`**

```
pinMode(ledPin, OUTPUT); // ตั้งค่าให้ GPIO เป็นเอาต์พุต
```

- **pinMode(ledPin, OUTPUT);**: กำหนดให้ GPIO 16 เป็น เอาต์พุต **(Output)** เพื่อส่งสัญญาณควบคุม LED

- ฟังก์ชัน setup() จะทำงานเพียงครั้งเดียวเมื่อ ESP32 เริ่มต้น

3 ฟังก์ชัน **loop()**

ฟังก์ชันนี้ทำงานวนซ้ำไม่รู้จบ (infinite loop) เพื่อควบคุมความสว่างของ LED

(3.1) เพิ่มความสว่างของ **LED API**

```
// เพิ่มความสว่างของ LED
for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
  analogWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle
  delay(10); // หน่วงเวลา 10 มิลลิวินาที
}
```


- **for loop**:
  - เริ่มต้นด้วยค่า dutyCycle = 0
  - เพิ่มค่า dutyCycle ครั้งละ 1 จนถึงค่า **255**

- **analogWrite(ledPin, dutyCycle);**:
  - ส่งสัญญาณ PWM ไปที่ GPIO 16
  - ค่า **dutyCycle** ที่เพิ่มขึ้นทำให้ LED สว่างขึ้นทีละน้อย

- **delay(10);**: หน่วงเวลา 10 มิลลิวินาทีเพื่อให้การเปลี่ยนแปลงความสว่างดูราบรื่น

(3.2) ลดความสว่างของ **LED**

```
// ลดความสว่างของ LED
for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
  analogWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle
  delay(10); // หน่วงเวลา 10 มิลลิวินาที
}
```

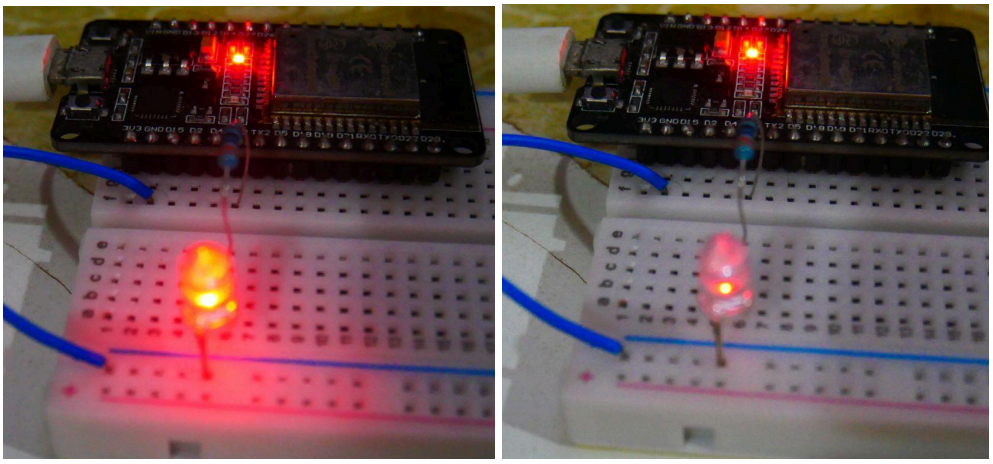
- **for loop**:
  - เริ่มต้นด้วยค่า `dutyCycle = 255`
  - ลดค่า `dutyCycle` ครั้งละ 1 จนถึง **0**

- **`analogWrite(ledPin, dutyCycle);`**:
  - ค่า **`dutyCycle`** ที่ลดลงทำให้ LED ค่อย ๆ มืดลง

- **`delay(10);`**: หน่วงเวลา 10 มิลลิวินาทีเพื่อควบคุมความเร็วของการเปลี่ยนแปลง

4. หลักการทำงานโดยรวม

- การเพิ่มความสว่าง: `dutyCycle` เพิ่มจาก 0 → 255 ทำให้ LED สว่างขึ้น
- การลดความสว่าง: `dutyCycle` ลดจาก 255 → 0 ทำให้ LED มืดลง
- กระบวนการนี้จะวนซ้ำไม่รู้จบในฟังก์ชัน `loop()`



โค้ดตัวอย่าง: ESP32 PWM โดยใช้ LEDC API (ไฟล์ PWM_LEDC)

โค้ดด้านล่างนี้แสดงวิธีการเพิ่มและลดความสว่างของ LED โดยใช้ฟังก์ชัน LEDC ของ ESP32:

```
// กำหนดพิน GPIO ที่ใช้เชื่อมต่อกับ LED
const int ledPin = 16;

// กำหนดคุณสมบัติของสัญญาณ PWM
const int freq = 5000;       // ความถี่ PWM ที่ใช้ (5000 Hz)
const int resolution = 8;    // ความละเอียดของ PWM (8 บิต: ค่า 0-255)

void setup() {
  // ตั้งค่า PWM ให้กับพินที่กำหนด
  ledcAttach(ledPin, freq, resolution);

  // หากต้องการระบุช่องสัญญาณเฉพาะ ให้ใช้ฟังก์ชันนี้แทน
  // ledcAttachChannel(ledPin, freq, resolution, 0);
}

void loop() {
  // เพิ่มความสว่างของ LED
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
```

```
    ledcWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle เพื่อเปลี่ยนความสว่างของ
LED
    delay(15); // หน่วงเวลา 15 มิลลิวินาทีเพื่อทำให้การเปลี่ยนแปลงดูนุ่มนวล
  }

  // ลดความสว่างของ LED
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    ledcWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle เพื่อเปลี่ยนความสว่างของ
LED
    delay(15); // หน่วงเวลา 15 มิลลิวินาทีเพื่อทำให้การเปลี่ยนแปลงดูนุ่มนวล
  }
}
```

การอธิบายการทำงานของโค้ด

1. การตั้งค่าเริ่มต้นใน **`setup()`**

- ฟังก์ชัน `ledcAttach(ledPin, freq, resolution)` ใช้สำหรับกำหนดให้พิน **GPIO16** สร้างสัญญาณ PWM ที่มีความถี่ **5000 Hz** และมีความละเอียด **8** บิต (ค่า Duty Cycle ระหว่าง 0 ถึง 255)

2. การเพิ่มและลดความสว่างใน **`loop()`**

โค้ดในส่วนนี้ทำให้ LED ค่อย ๆ สว่างขึ้นและค่อย ๆ หรี่ลงอย่างต่อเนื่อง โดยแบ่งออกเป็น 2 ส่วนหลัก:

ส่วนที่ **1:** เพิ่มความสว่างของ **LED**

```
// เพิ่มความสว่างของ LED
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    ledcWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle เพื่อเปลี่ยนความสว่างของ
LED
    delay(15); // หน่วงเวลา 15 มิลลิวินาทีเพื่อทำให้การเปลี่ยนแปลงดูนุ่มนวล
  }
```

- เริ่มต้นจาก Duty Cycle = 0 (LED ปิดสนิท)

- เพิ่ม Duty Cycle ทีละ 1 จนถึง 255 (LED สว่างสุด)
- ใช้ฟังก์ชัน `ledcWrite()` เพื่อกำหนดค่าความสว่างของ LED
- หน่วงเวลา **15** มิลลิวินาที เพื่อให้การเปลี่ยนแปลงความสว่างดูนุ่มนวลและต่อเนื่อง

ส่วนที่ **2:** ลดความสว่างของ **LED**

```
  // ลดความสว่างของ LED
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    ledcWrite(ledPin, dutyCycle); // ตั้งค่า Duty Cycle เพื่อเปลี่ยนความสว่างของ
LED
    delay(15); // หน่วงเวลา 15 มิลลิวินาทีเพื่อทำให้การเปลี่ยนแปลงดูนุ่มนวล
  }
```

- เริ่มต้นจาก Duty Cycle = 255 (LED สว่างสุด)
- ลด Duty Cycle ทีละ 1 จนถึง 0 (LED ปิดสนิท)
- ใช้ฟังก์ชัน `ledcWrite()` เพื่อกำหนดค่าความสว่างของ LED
- หน่วงเวลา **15** มิลลิวินาที เช่นเดียวกับส่วนเพิ่มความสว่าง

หลักการทำงานโดยรวม

- LED จะค่อย ๆ สว่างขึ้นและค่อย ๆ หรี่ลงอย่างต่อเนื่องในรูปแบบลูปไม่สิ้นสุด
- กระบวนการนี้เป็นการควบคุมความสว่างของ LED โดยการปรับค่า **Duty Cycle** ของสัญญาณ PWM

การทดสอบโค้ดตัวอย่าง

ขั้นตอนการอัพโหลดโค้ด

1. เชื่อมต่อ **ESP32** กับคอมพิวเตอร์**:**
2. เปิด **Arduino IDE:**
3. ตั้งค่าบอร์ดและพอร์ต**:**
4. อัพโหลดโค้ด**:**

สรุปบทความ

ในบทความนี้ เราได้เรียนรู้เกี่ยวกับการใช้งาน **LED PWM Controller** บน ESP32 ร่วมกับ Arduino IDE เพื่อควบคุมความสว่างของ LED โดยมีการนำเสนอ สองวิธี ที่สามารถให้ผลลัพธ์เหมือนกัน:

1. การใช้คำสั่ง **analogWrite()**
    - วิธีที่ง่ายและรวดเร็วในการสร้างสัญญาณ PWM
    - ใช้ได้กับพิน GPIO ที่รองรับการส่งออก PWM
2. การใช้ฟังก์ชันของ **LEDC API**
    - วิธีที่ยืดหยุ่นและสามารถปรับแต่งสัญญาณ PWM ได้ละเอียดกว่า
    - รองรับการควบคุม PWM หลายช่องพร้อมกัน