การใช้งาน **GPIO** บน **ESP32 - Digital Read** และ **Digital Write**

หนึ่งในฟังก์ชันพื้นฐานที่สำคัญที่สุดของ ESP32 คือ การใช้งาน **GPIO (General Purpose Input/Output)** ซึ่งช่วยให้เราสามารถควบคุมอุปกรณ์ภายนอกหรือรับข้อมูลจากเซ็นเซอร์ได้อย่างง่ายดาย
ในบทเรียนนี้ เราจะเรียนรู้การใช้งาน **Digital Read** และ **Digital Write** บนพิน GPIO ของ ESP32

- **Digital Read** ใช้สำหรับอ่านสถานะของพิน GPIO ที่เชื่อมต่อกับอุปกรณ์ เช่น ปุ่มกด หรือเซ็นเซอร์แบบดิจิทัล

- **Digital Write** ใช้สำหรับควบคุมอุปกรณ์ เช่น หลอดไฟ LED หรือรีเลย์ โดยการส่งสัญญาณดิจิทัล (HIGH หรือ LOW)

วัตถุประสงค์

- เข้าใจพื้นฐานการทำงานของ GPIO บน ESP32
- เรียนรู้การตั้งค่าพินของ ESP32 ให้เป็น **Input** หรือ **Output**
- สามารถใช้งานฟังก์ชัน `digitalRead()` และ `digitalWrite()` ได้อย่างถูกต้อง
- ทดลองใช้งานวงจรและโปรแกรมเพื่อควบคุม LED และอ่านสถานะจากปุ่มกด

GPIO บน **ESP32** คืออะไร?

GPIO (General Purpose Input/Output) บน ESP32 คือชุดของพินที่สามารถตั้งค่าให้เป็นได้ทั้ง **Input** (สำหรับรับค่าจากอุปกรณ์ เช่น เซ็นเซอร์หรือปุ่มกด) หรือ **Output** (สำหรับควบคุมอุปกรณ์ เช่น LED หรือรีเลย์) โดย ESP32 มี GPIO ที่รองรับการทำงานได้หลากหลาย เช่น:
- **Digital Input/Output**: สำหรับการอ่านและเขียนค่าดิจิทัล (0 หรือ 1)
- **Analog Input**: สำหรับการอ่านค่าจากเซ็นเซอร์แบบแอนะล็อก
- **PWM Output**: สำหรับการสร้างสัญญาณ PWM เช่น ควบคุมความสว่างของ LED

วัสดุและอุปกรณ์ที่ต้องใช้

1. **ESP32 Development Board**
    - บอร์ด ESP32 ที่ใช้พัฒนา เช่น DOIT ESP32 Devkit V1

2. **LED (Light Emitting Diode)**
    - สำหรับทดลองการเปิด-ปิดด้วย GPIO

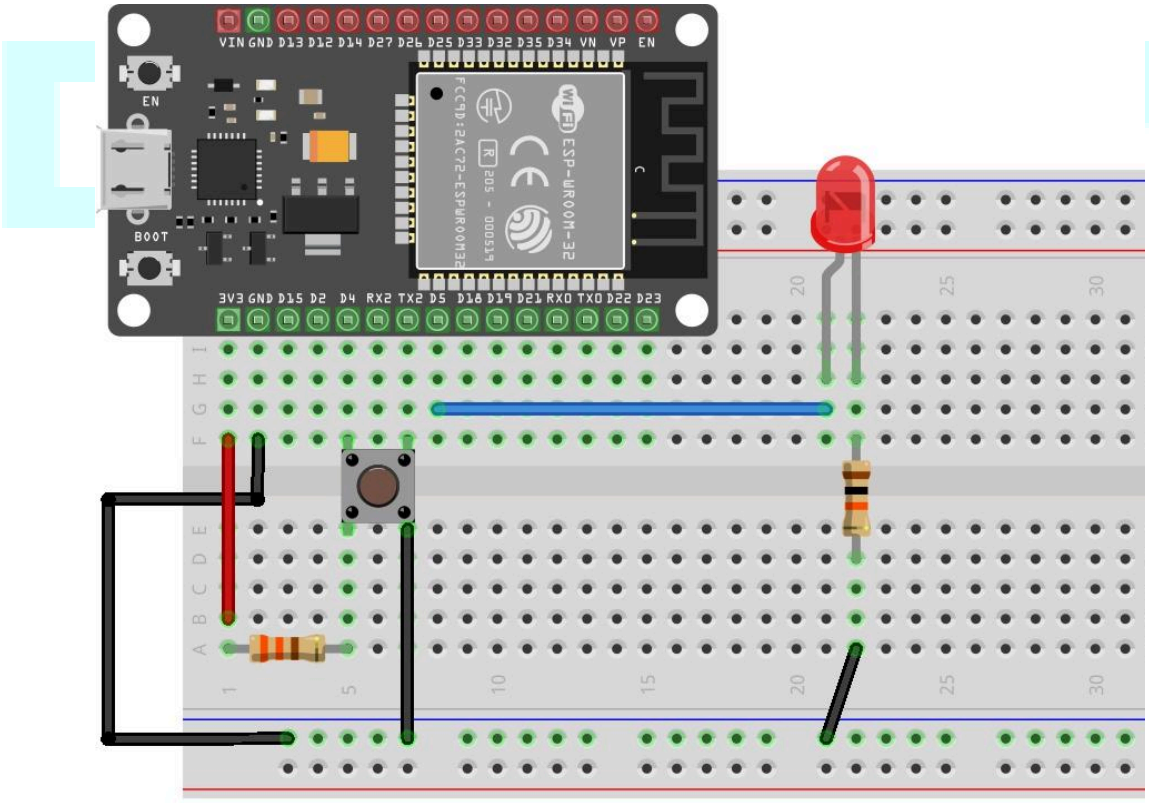
3. ตัวต้านทาน **330 Ω**
    - เพื่อลดกระแสไฟฟ้าที่ไหลผ่าน LED ป้องกันไม่ให้ LED เสียหาย

4. **Push Button** (ปุ่มกด)
    - สำหรับทดลองการอ่านค่าดิจิทัลจาก GPIO

5. ตัวต้านทาน **10 kΩ (Pull-up Resistor)**
    - สำหรับใช้กับปุ่มกด

6. สายไฟ **Jumper**
    - ใช้เชื่อมต่อวงจร

7. บอร์ดทดลอง **(Breadboard)**
    - สำหรับสร้างวงจรทดลอง



พื้นฐานการทำงานของ **GPIO** บน **ESP32**

1. Digital Output (การเขียนค่าดิจิทัล)

- ใช้ฟังก์ชัน `digitalWrite(pin, value)`
- สามารถส่งค่าออกไปยัง GPIO ในรูปแบบ HIGH (1) หรือ LOW (0)
- ตัวอย่าง:
  - HIGH: ทำให้ LED ติด (ส่งแรงดันไฟฟ้า 3.3V)
  - LOW: ทำให้ LED ดับ (ส่งแรงดันไฟฟ้า 0V)

2. Digital Input (การอ่านค่าดิจิทัล)

- ใช้ฟังก์ชัน `digitalRead(pin)`
- อ่านค่าจาก GPIO ที่เชื่อมต่อกับอุปกรณ์ เช่น ปุ่มกด
- ผลลัพธ์ที่ได้จะเป็น HIGH (1) หรือ LOW (0)

3. การตั้งค่า **GPIO**

- ใช้ฟังก์ชัน `pinMode(pin, mode)`
  - `OUTPUT`: สำหรับการเขียนค่าดิจิทัล
  - `INPUT`: สำหรับการอ่านค่าดิจิทัล
  - `INPUT_PULLUP`: เพิ่ม Pull-up Resistor ภายใน (สำหรับปุ่มกด)

การเชื่อมต่อวงจร

1. การเชื่อมต่อ **LED (Digital Output)**

- ต่อขา **GPIO 5** ของ ESP32 เข้ากับขา **Anode (+)** ของ LED
- ขา **Cathode (-)** ของ LED ต่อเข้ากับตัวต้านทาน 330 Ω
- ขาอีกด้านของตัวต้านทานต่อเข้ากับ GND

2. การเชื่อมต่อปุ่มกด **(Digital Input)**

- ต่อขา **GPIO 4** ของ ESP32 เข้ากับขาหนึ่งของปุ่มกด
- ขาอีกด้านของปุ่มกดต่อเข้ากับ GND
- เพิ่ม Pull-up Resistor 10 kΩ ระหว่าง GPIO 4 และ VCC

โค้ดการเปิด-ปิด **LED** (ไฟล์ **LED_BLINK**)

```
#define LED_PIN 5  // กำหนดพิน GPIO ที่เชื่อมต่อกับ LED

void setup() {
```

```
  pinMode(LED_PIN, OUTPUT);   // ตั้งค่าให้พิน LED_PIN เป็น Output
}

void loop() {
  digitalWrite(LED_PIN, HIGH);   // เปิด LED (ส่งสัญญาณ HIGH)
  delay(1000);                   // รอ 1 วินาที
  digitalWrite(LED_PIN, LOW);    // ปิด LED (ส่งสัญญาณ LOW)
  delay(1000);                   // รอ 1 วินาที
}
```

คำอธิบายการทำงานของโค้ด

1. การกำหนดพิน **LED:**
   - ในบรรทัดแรก `#define LED_PIN 5`
     เรากำหนดให้ตัวแปร `LED_PIN` แทนหมายเลข GPIO ที่เชื่อมต่อกับ LED (ในที่นี้คือ GPIO 5) เพื่อให้อ่านและแก้ไขได้ง่ายหากต้องการเปลี่ยนพินในอนาคต

2. การตั้งค่าโหมดของพิน:
   - ในฟังก์ชัน `setup()` เราใช้คำสั่ง `pinMode(LED_PIN, OUTPUT)` เพื่อกำหนดให้พิน GPIO 5 เป็น **Output** หมายความว่าพินนี้จะส่งสัญญาณออกไปควบคุมอุปกรณ์ (ในที่นี้คือ LED)

3. การควบคุม **LED** ในฟังก์ชัน **`loop()`**
   - `digitalWrite(LED_PIN, HIGH)`
     ส่งสัญญาณ **HIGH (1)** ไปยังพิน GPIO 4 ซึ่งทำให้ LED ติดสว่าง
   - `delay(1000)`
     รอเวลา 1 วินาที (1000 มิลลิวินาที) ก่อนดำเนินการคำสั่งถัดไป
   - `digitalWrite(LED_PIN, LOW)`
     ส่งสัญญาณ **LOW (0)** ไปยังพิน GPIO 4 ซึ่งทำให้ LED ดับ
   - `delay(1000)`
     รอเวลาอีก 1 วินาที และวนกลับไปเริ่มต้นที่ `digitalWrite(LED_PIN, HIGH)`

4. การทำงานวนลูป:
   - ฟังก์ชัน `loop()` จะทำงานซ้ำอย่างต่อเนื่อง ส่งผลให้ LED สว่างและดับสลับกันทุก 1 วินาที

โค้ดการอ่านค่าจากปุ่มกด (ไฟล์ **READ_BTN**)

```
#define BUTTON_PIN 4  // กำหนดหมายเลขพินที่เชื่อมต่อกับปุ่มกด

void setup() {
  Serial.begin(115200);          // เริ่มต้น Serial Monitor ที่ความเร็ว
   115200 bps
  pinMode(BUTTON_PIN, INPUT);    // กำหนดพิน BUTTON_PIN เป็นพินสำหรับรับ
   ค่า (Input)
}

void loop() {
  int buttonState = digitalRead(BUTTON_PIN);  // อ่านสถานะของปุ่มกด
   (HIGH หรือ LOW)

  if (buttonState == HIGH) {
    Serial.println("Button Released");  // แสดงข้อความเมื่อปุ่มไม่ได้ถูกกด
  } else {
    Serial.println("Button Pressed");   // แสดงข้อความเมื่อปุ่มถูกกด
  }

  delay(100);  // หน่วงเวลา 100ms เพื่อป้องกันการอ่านค่าที่รวดเร็วเกินไป
}
```

คำอธิบายโค้ด

1. กำหนดพินสำหรับปุ่มกด

```
#define BUTTON_PIN 4  // กำหนดหมายเลขพินที่เชื่อมต่อกับปุ่มกด
```

- ในโค้ดนี้ เรากำหนดให้ **BUTTON_PIN** หมายเลขพิน 4 เป็นพินที่เชื่อมต่อกับปุ่มกด
- คุณสามารถเปลี่ยนหมายเลขพินให้ตรงกับฮาร์ดแวร์ที่ใช้งานได้

2. ตั้งค่าโหมดของพิน (**pinMode**)

```
pinMode(BUTTON_PIN, INPUT);
```

- INPUT: กำหนดพินให้เป็นโหมดรับสัญญาณจากอุปกรณ์ภายนอก (Input) เช่น ปุ่มกดหรือเซ็นเซอร์

3. อ่านสถานะของปุ่ม **(digitalRead)**

```
int buttonState = digitalRead(BUTTON_PIN);
```

- ฟังก์ชัน digitalRead() ใช้สำหรับอ่านสถานะของพินที่ตั้งค่าเป็น Input
- ผลลัพธ์จะเป็น **HIGH** (5V/3.3V) หากปุ่มถูกกด หรือ **LOW** (0V) หากปุ่มไม่ได้ถูกกด

4. ตรวจสอบสถานะของปุ่ม

```
if (buttonState == HIGH) {
    Serial.println("Button Released");   // แสดงข้อความเมื่อปุ่มไม่ได้ถูกกด
  } else {
    Serial.println("Button Pressed");    // แสดงข้อความเมื่อปุ่มถูกกด
  }
```

- ใช้เงื่อนไข if-else เพื่อตรวจสอบว่าปุ่มถูกกดหรือไม่
- หากสถานะเป็น **HIGH** ให้แสดงข้อความว่า "Button Released"
- หากสถานะเป็น **LOW** ให้แสดงข้อความว่า "Button Pressed"

5. หน่วงเวลา **(Delay)**

```
delay(100);
```

- เพิ่มการหน่วงเวลา 100 มิลลิวินาทีในแต่ละรอบของ loop() เพื่อป้องกันการอ่านค่าซ้ำที่เร็วเกินไป

การทำงานของวงจร

- เมื่อปุ่มกดถูกกด (วงจรครบลูป):
  - สถานะพิน BUTTON_PIN จะเปลี่ยนเป็น **HIGH**
  - ข้อความ "Button Pressed" จะถูกส่งไปยัง Serial Monitor
- เมื่อปุ่มกดไม่ได้ถูกกด (วงจรเปิด):
  - สถานะพิน BUTTON_PIN จะเป็น **LOW**
  - ข้อความ "Button Released" จะถูกส่งไปยัง Serial Monitor

โค้ดการควบคุม **LED** ด้วยปุ่มกด **(ไฟล์ BTN_LED)**

```
const int buttonPin = 4;   // พินที่เชื่อมต่อกับปุ่มกด
const int ledPin = 5;      // พินที่เชื่อมต่อกับ LED

int buttonState = 0;       // ตัวแปรเก็บสถานะของปุ่มกด (HIGH/LOW)

void setup() {
     pinMode(buttonPin, INPUT); // ตั้งค่าปุ่มกดเป็น Input
  // pinMode(buttonPin, INPUT_PULLUP); // ตั้งค่าปุ่มกดเป็น Input พร้อมเปิด
   Pull-up Resistor
  pinMode(ledPin, OUTPUT);          // ตั้งค่า LED เป็น Output
}

void loop() {
  buttonState = digitalRead(buttonPin); // อ่านสถานะของปุ่มกด

  if (buttonState == LOW) {             // ตรวจสอบว่าปุ่มถูกกด (LOW)
    digitalWrite(ledPin, HIGH);         // เปิด LED
  } else {
    digitalWrite(ledPin, LOW);          // ปิด LED
  }
}
```

คำอธิบายการทำงานของโค้ด

กำหนดตัวแปร

```
const int buttonPin = 4;
const int ledPin = 5;      // พินที่เชื่อมต่อกับ LED
```

พินที่ 4 ของ ESP32 เชื่อมต่อกับปุ่มกด
พินที่ 5 ของ ESP32 เชื่อมต่อกับ LED

การตั้งค่าพินใน **setup()**

```
pinMode(buttonPin, INPUT);
```

ตั้งค่าปุ่มกดเป็น Input (เปิดใช้งาน **Internal Pull-up Resistor** เพื่อให้สถานะเริ่มต้นของปุ่มกดเป็น HIGH)

- เมื่อกดปุ่ม สัญญาณจะถูกดึงลงเป็น LOW
- หากวงจรปุ่มกดไม่มีตัวต้านทาน Pull-up หรือ Pull-down ภายนอก ควรใช้ INPUT_PULLUP เพื่อใช้งานตัวต้านทาน Pull-up ภายในของ ESP32
- การใช้ Pull-up Resistor ช่วยลดสัญญาณรบกวนและไม่ต้องต่อตัวต้านทานภายนอก

ตั้งค่าพินของ LED เป็น Output เพื่อให้สามารถควบคุมการเปิด-ปิด LED ได้

```
pinMode(ledPin, OUTPUT);
```

การทำงานใน **loop()**

```
buttonState = digitalRead(buttonPin);
```

อ่านสถานะของปุ่มกดและเก็บค่าลงในตัวแปร buttonState

- ถ้าปุ่มไม่ถูกกด จะได้ค่า **HIGH**
- ถ้าปุ่มถูกกด จะได้ค่า **LOW**

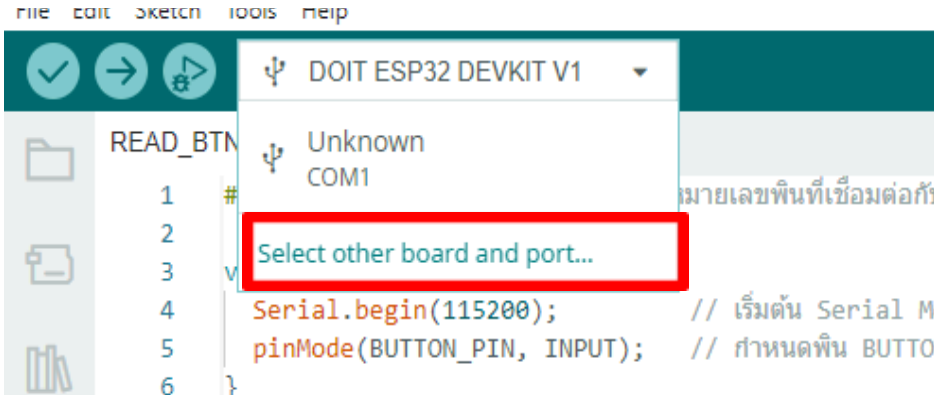
```
if (buttonState == LOW) {              // ตรวจสอบว่าปุ่มถูกกด (LOW)
    digitalWrite(ledPin, HIGH);          // เปิด LED
  } else {
    digitalWrite(ledPin, LOW);           // ปิด LED
  }
```

ตรวจสอบว่าปุ่มถูกกดหรือไม่ (LOW หมายถึงปุ่มถูกกดเนื่องจาก Pull-up Resistor)

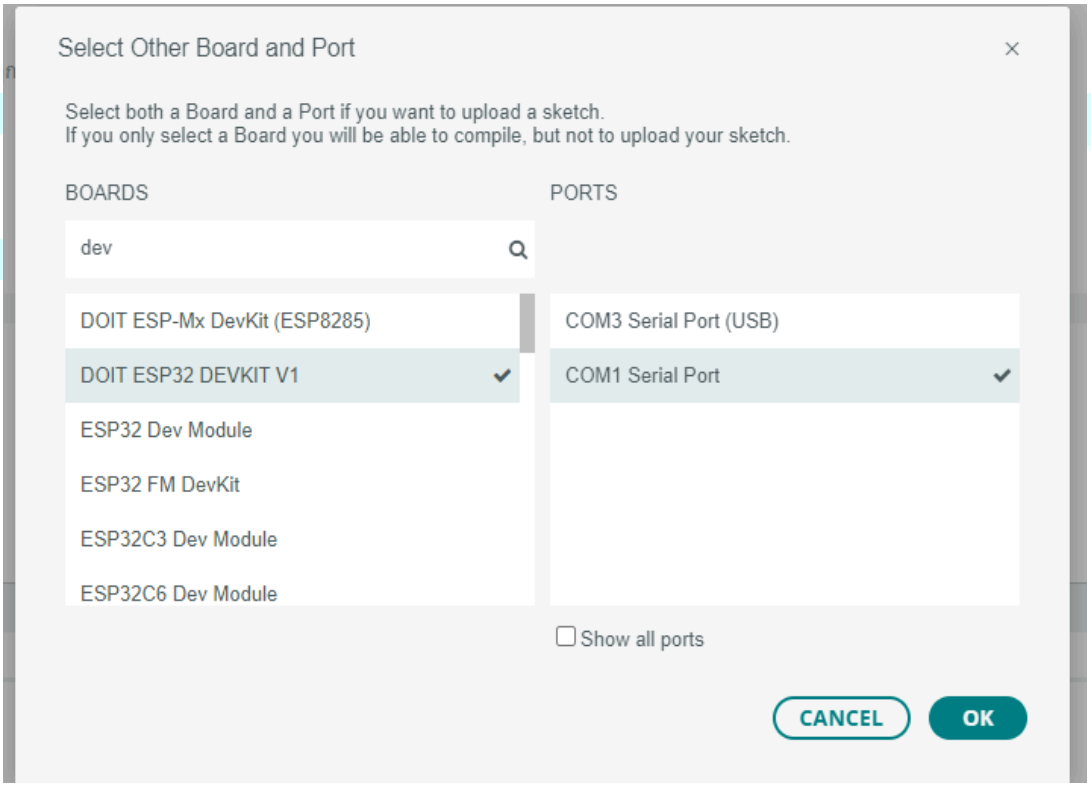
- ถ้าปุ่มถูกกด (LOW): สั่งให้พิน LED เป็น HIGH (เปิดไฟ LED)
- ถ้าปุ่มไม่ถูกกด (HIGH): สั่งให้พิน LED เป็น LOW (ปิดไฟ LED)

การอัปโหลดโค้ด (Sketch)

เลือกบอร์ดของคุณ โดยไปที่ **Tools > Board** หรือคลิกที่เมนูด้านบนแล้วเลือก **"Select other board and port…"**



หน้าต่างใหม่จะปรากฏขึ้นตามตัวอย่างด้านล่าง จากนั้น ให้ค้นหารุ่นบอร์ด **ESP32** ของคุณ ในรายการที่แสดงขึ้นมา



เลือกบอร์ดรุ่นที่คุณใช้งานและพอร์ต COM ที่เชื่อมต่อกับ ESP32 ของคุณ ในตัวอย่างนี้เราใช้ **DOIT ESP32 DEVKIT V1** หลังจากเลือกเสร็จแล้ว ให้คลิก **OK** เพื่อยืนยันการตั้งค่าและดำเนินการต่อ!

คลิกที่ปุ่ม **Upload** ใน Arduino IDE แล้วรอสักครู่เพื่อให้โปรแกรมทำการคอมไพล์ (Compile) และอัปโหลดโค้ดของคุณไปยังบอร์ด ESP32 เมื่ออัปโหลดเสร็จสิ้น

การทดสอบโปรเจกต์ของคุณ

หลังจากอัปโหลดโค้ดเสร็จแล้ว ให้ทดสอบวงจรของคุณ!

- เมื่อคุณกดปุ่ม Pushbutton ไฟ LED ควรสว่างขึ้น
- และเมื่อคุณปล่อยปุ่ม ไฟ LED ควรดับลง