Manual Spidy Detector

คู่มือ Spidy Detector สำหรับนักเรียนและครูหรือผู้ที่สนใจในการเขียนโปรแกรม

จัดทำโดย ศูนย์พัฒนาเทคโนโลยีการศึกษา
มหาวิทยาลัยราชภัฏเชียงราย , 2023

คำนำ

ขอแสดงความยินดีและขอบคุณอย่างสูงที่ท่านได้เลือกเป็นส่วนหนึ่งของการเรียนรู้และพัฒนาตนเองผ่านหนังสือคู่มือนี้ หนังสือคู่มือนี้ถูกสร้างขึ้นเพื่อให้ความรู้และแนวทางในหัวข้อที่ท่านสนใจ เราหวังว่าการอ่านหนังสือคู่มือนี้จะเป็นประโยชน์และนำทางท่านในการเข้าใจและปฏิบัติตามหลักการที่มีอยู่ในหนังสือนี้ หนังสือคู่มือนี้เต็มไปด้วยข้อมูลและข้อแนะนำที่มีคุณค่าในการปฏิบัติงานหรือเรียนรู้ในสาขาที่ท่านสนใจ โดยเราได้รวบรวมความรู้จากผู้เชี่ยวชาญและมีประสบการณ์จริงในสาขานี้เพื่อให้คุณได้รับประโยชน์อย่างเต็มที่จากการอ่านหนังสือคู่มือนี้ เราอาศัยความร่วมมือและความเข้าใจจากผู้เชี่ยวชาญในการเขียนหนังสือคู่มือนี้ เพื่อให้ข้อมูลและข้อแนะนำที่ได้ถูกต้องและเป็นประโยชน์สำหรับคุณอย่างแน่นอน หากท่านมีคำถามหรือข้อสงสัยเกี่ยวกับเนื้อหาใดๆในหนังสือคู่มือนี้ กรุณาอย่าลังเลที่จะติดต่อเราเพื่อขอคำแนะนำเพิ่มเติมหรือคำชี้แนะเพื่อให้คุณได้รับประโยชน์อย่างมาก

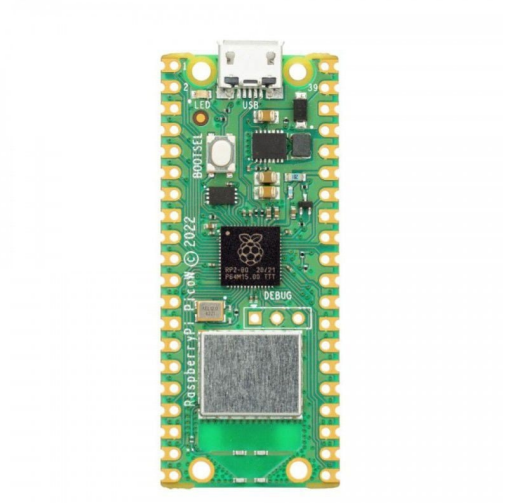
สารบัญ

Micropython คืออะไร ?

Micropython เป็นภาษาโปรแกรมที่เขียนขึ้นบนภาษา Python และออกแบบมาเพื่อใช้ในการโปรแกรมไมโครคอนโทรลเลอร์เบอร์ด โดยเฉพาะอย่างยิ่งในอุปกรณ์เล็กๆ เช่น ไมโครคอนโทรลเลอร์ในเซ็นเซอร์ หุ่นยนต์ขนาดเล็ก หรืออุปกรณ์อิเล็กทรอนิกส์เบื้องต้นอื่น ๆ ที่มีข้อจำกัดในทรัพยากรด้านความจำและประสิทธิภาพพลังงาน

Micropython ใช้ภาษา Python เป็นพื้นฐาน ซึ่งเป็นภาษาโปรแกรมที่เข้าใจง่ายและมีความยืดหยุ่น ดังนั้น การเรียนรู้และใช้ Micropython จะไม่เป็นเรื่องยากสำหรับผู้ที่มีความรู้พื้นฐานใน Python อยู่แล้ว การใช้ Micropython ให้คุณสามารถเขียนโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ได้ตามต้องการ คุณสามารถเขียนโค้ดเพื่ออ่านค่าจากเซ็นเซอร์ ควบคุมการทำงานของมอเตอร์ สื่อสารกับอุปกรณ์อื่น ๆ และจัดการกับการตอบสนองที่ต้องการจากอุปกรณ์ต่าง ๆ ได้อย่างยืดหยุ่น
นอกจากนี้ Micropython ยังมีความสามารถในการเชื่อมต่อกับอินเทอร์เน็ต อ่านและเขียนข้อมูลผ่านหน้าจอ OLED หรือควบคุมอุปกรณ์ผ่านการใช้ ESP8266 เป็นต้น

Micropython เป็นภาษาที่พัฒนาขึ้นโดยดร. ดาม เดเมียร์ จอร์ท (Damien George) ภาษานี้เขียนขึ้นบนพื้นฐานภาษา Python และถูกออกแบบมาเพื่อใช้งานในไมโครคอนโทรลเลอร์เบอร์ด ที่มีข้อจำกัดในทรัพยากรด้านความจำและประสิทธิภาพพลังงาน Micropython ใช้ระบบอินเตอร์เพื่อความสะดวกในการเขียนและประมวลผลโปรแกรม โดยมีตัวแปรและฟังก์ชันที่คล้ายกับภาษา Python และมีความสามารถในการสร้างและควบคุมอุปกรณ์ภายนอก เช่น อิเล็กทรอนิกส์และเซ็นเซอร์ หนึ่งในข้อได้เปรียบของ Micropython คือความสามารถในการรันโค้ดอย่างมีเหตุผลและรวดเร็ว หากต้องการเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์ทำงานเป็นเวลาเร็ว หรือต้องการความประหยัดทรัพยากรในการใช้พลังงาน ก็สามารถใช้ Micropython เพื่อบรรลุเป้าหมายดังกล่าวได้
นอกจากนี้ มีบอร์ดไมโครคอนโทรลเลอร์ที่รองรับ Micropython มากมาย เช่น ESP8266, ESP32, Raspberry Pi Pico เป็นต้น ทำให้ผู้ใช้งานสามารถเริ่มต้นการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์ได้อย่างง่ายและสะดวก

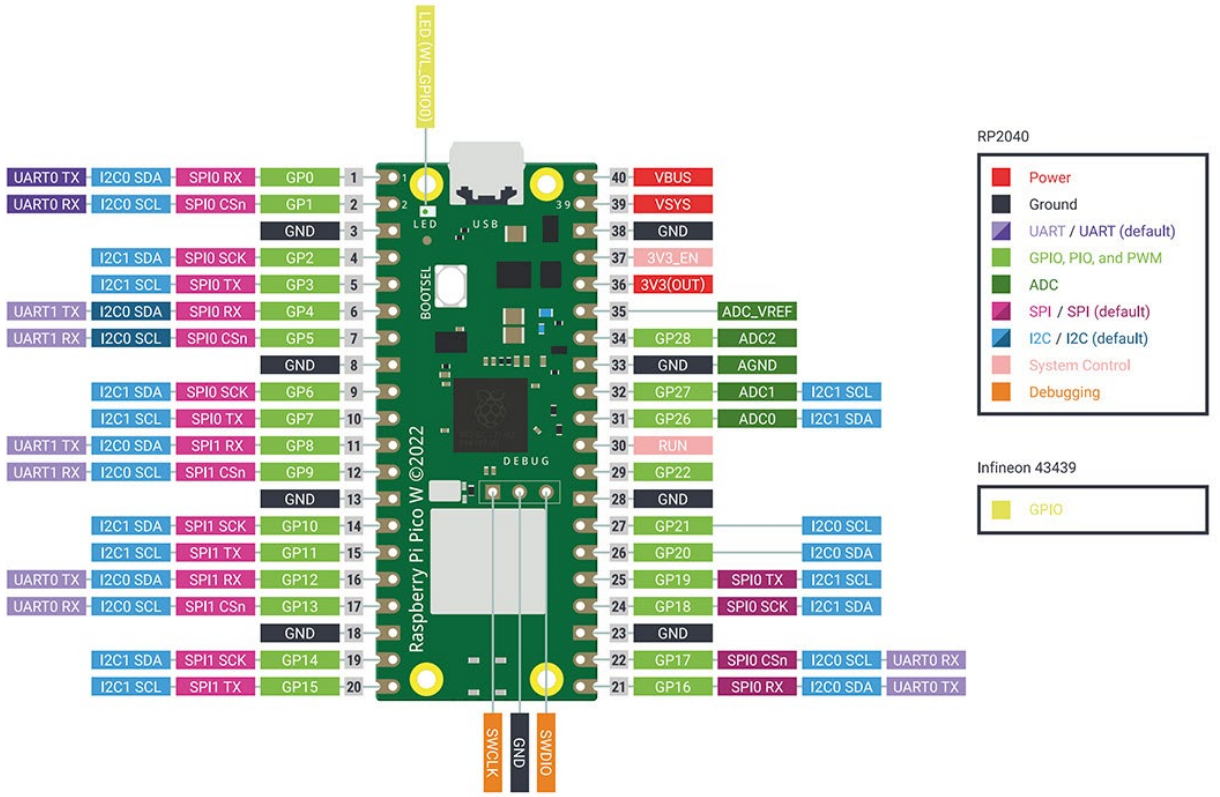


Raspberry Pi Pico W คืออะไร ?

Raspberry Pi Pico W เป็นเวอร์ชันของบอร์ดไมโครคอนโทรลเลอร์ Raspberry Pi Pico ที่เพิ่มความสามารถในการเชื่อมต่อกับเครือข่ายไร้สาย (Wi-Fi) ในตัวอุปกรณ์ โดยมีระบบ Wi-Fi ในรูปแบบของชิป ESP32 ที่ออกแบบมาเพื่อใช้งานในการเชื่อมต่ออินเทอร์เน็ตไร้สาย

Raspberry Pi Pico W มีคุณสมบัติและฟีเจอร์ที่คล้ายกับ Raspberry Pi Pico เดิม ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาดเล็กและมีราคาไม่แพงที่สามารถนำไปใช้ในหลากหลายโครงการได้ คุณสมบัติหลักของ Raspberry Pi Pico W ได้แก่:

1. ไมโครคอนโทรลเลอร์: Raspberry Pi Pico W ใช้ชิป RP2040 เช่นเดียวกับ Raspberry Pi Pico ซึ่งมีส่วนประมวลผลและความจำในตัว มีหน่วยประมวลผลคู่ (Dual-Core) ARM Cortex-M0+ ความเร็ว 133 และ 166 MHz รวมถึงเมมโมรี่ RAM 264KB เพียงพอสำหรับการพัฒนาและการทำงานทั่วไป
2. Wi-Fi: Raspberry Pi Pico W มาพร้อมโมดูล Wi-Fi ESP32 ที่มีความสามารถในการเชื่อมต่อกับเครือข่ายไร้สาย ซึ่งรองรับการเชื่อมต่อ Wi-Fi ที่ความเร็วสูงและมีความเสถียร
3. ขนาดเล็กและตัวควบคุมเสียง: Raspberry Pi Pico W มีขนาดเล็กและเสียงต่ำ ทำให้เหมาะสำหรับการใช้งานในโครงการที่ต้องการความพกพาและตัวควบคุมที่เล็กกะ



Raspberry Pi Pico W เป็นบอร์ดไมโครคอนโทรลเลอร์ที่มีขนาดเล็กเพียง 51 x 21 มิลลิเมตร เข้ากันได้กับบอร์ดอุปกรณ์อื่น ๆ และมีขาเชื่อมต่อ GPIO ขนาด 26 ขา สามารถใช้ในการเชื่อมต่ออุปกรณ์เซ็นเซอร์ อินพุต/เอาต์พุตดิจิตอล อินพุตแอนะล็อก (Analog input) หรืออื่น ๆ ตามความต้องการของโครงการ

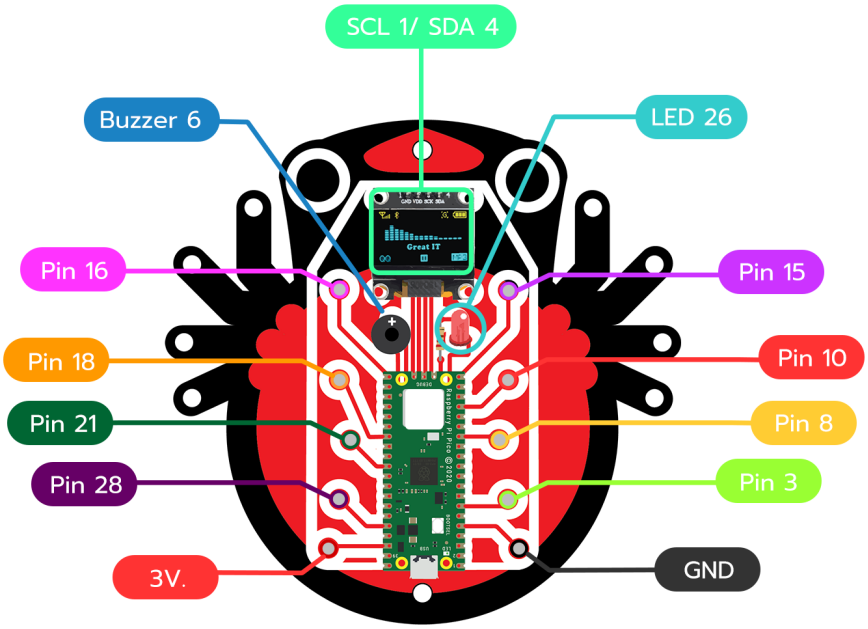
ในส่วนของเครือข่ายไร้สาย Wi-Fi บน Raspberry Pi Pico W มาจากโมดูล ESP32 ซึ่งเป็นชิป Wi-Fi ที่รองรับการเชื่อมต่อ Wi-Fi และการสื่อสารผ่านโปรโตคอลต่าง ๆ เช่น TCP/IP, UDP และ HTTP ทำให้คุณสามารถเชื่อมต่อกับเครือข่ายบ้านหรืออินเทอร์เน็ตได้อย่างง่ายดาย ซึ่งจะเป็นประโยชน์สำหรับโครงการที่ต้องการการสื่อสารแบบไร้สาย

นอกจากนี้ Raspberry Pi Pico W ยังมีฟีเจอร์อื่น ๆ เช่น:

- ตัวควบคุมเสียง: มีไมโครโฟนบอร์ดในตัว สามารถใช้ในการควบคุมเสียงและการสื่อสารผ่านช่องเสียงได้
- ขั้ว USB Type-C: มีขั้วเชื่อมต่อ USB Type-C เพื่อให้คุณสามารถเชื่อมต่อและจ่ายไฟให้กับบอร์ดได้ผ่านทางขั้ว USB
- การเข้ารหัสฮาร์ดแวร์: มีความสามารถในการเข้ารหัสฮาร์ดแวร์เพื่อความปลอดภัยในการใช้งานและป้องกันการถูกแฮ็ก

อุปกรณ์ในกล่อง

สไปดี้



สายไฟ

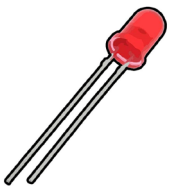
สาย USB

คลิปปากจระเข้

เซ็นเซอร์ แบ่งเป็น Output และ Input

Output

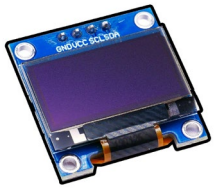
LED



LED คือไดโอดเปล่งแสง ย่อมาจากคำว่า(Light-Emitting Diode) ซึ่งสามารถเปล่งแสงออกมาได้ แสงที่เปล่งออกมาประกอบด้วยคลื่นความถี่เดียวและ เฟสต่อเนื่องกัน ซึ่งต่างกับแสงธรรมดาที่ตาคน

มองเห็น โดย หลอดLED สามารถเปล่งแสงได้เมื่อจ่ายกระแสไฟฟ้าเข้าเพียงเล็กน้อยเท่านั้น และประสิทธิภาพในการให้แสงสว่างก็ยังดีกว่าหลอดไฟขนาดเล็กทั่วๆ ไป

OLED



OLED (Organic Light Emitting Diodes) คือจอภาพที่มีลักษณะคล้ายแผ่นฟิล์ม ซึ่งมีส่วนประกอบเป็นสารอินทรีย์ที่สามารถเปล่งแสงเองได้เมื่อได้รับพลังงานไฟฟ้า เรียกว่ากระบวนการอิเล็คโทรลูมิเนเซนส์ (Electroluminescence) โดยที่ไม่ต้องพึ่งพาแสง Backlight และจะไม่มีการเปล่งแสดงในบริเวณที่เป็นภาพสีดำ ส่งผลให้สีดำนั้นดำสนิท อีกทั้งยังช่วยพลังงานด้วย

Buzzer active

โมดูลเสียง Active Buzzer Module สามารถสร้างเสียงเตือนได้ง่าย โดยการต่อขา I/O ของ Active Buzzer กับขา I/O ของ Arduino หรือไมโครคอนโทรลเลอร์ได้โดยตรง และต่อกับขา VCC ของ Active Buzzer กับไฟเลี้ยง 3.3-5VDC และขา GND กับ GND ในการสั่งให้ Active Buzzer มีเสียง ทำได้โดยการสั่งขาที่ต่อกับ Active Buzzer ให้เป็น LOW

Buzzer Passive

Passive Buzzer คือ buzzer ที่สามารถควบคุมโทนเสียง ด้วยการส่งสัญญาณค่าต่างๆเพื่อสร้างโทนเสียงที่แตกต่างกัน

หมายเหตุ: Active Buzzer คือ buzzer ที่สร้างเสียง peep จากสัญญาณ 2 ระดับ คือเปิด/ปิด เท่านั้น

Traffic light

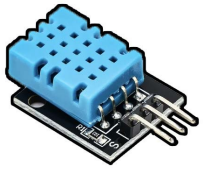
LED ไฟจราจร แสดงผลขนาด 10mm จำนวน 3 ดวง สำหรับทดลองเขียนโปรแกรมควบคุม Output ของ Arduino ทำไฟวิ่งรูปแบบต่าง ๆ ใช้ไฟเลี้ยง 3.3-5V

RGB

โมดูลหลอด LED แบบ RGB ขนาด 5mm สามารถสั่งการโดยใช้ PWM เพื่อผสมสีเป็นแสงสีต่างๆได้

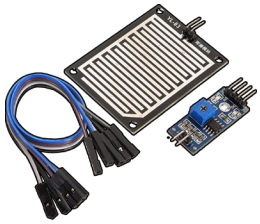
Input

DHT11



DHT11 Digital Temperature and Humidity Sensor เป็นโมดูลวัดอุณหภูมิ (Temperature) และความชื้นสัมพัทธ์ (Humidity) โดยใช้ชิพ DHT11 ให้ Output ออกมาเป็นแบบ Digital Output ใช้ไฟ DC ขนาด 3.5 – 5.5 โวลต์ เหมาะสำหรับวัดอุณหภูมิ (Temperature) ในช่วง 0-60° c และความชื้นสัมพัทธ์ (Humidity) ในช่วง 20-90 % RH.

Rain Detector sensor



เป็นโมดูลวัดความชื้นในอากาศและน้ำฝน โดยค่าที่ได้ออกมาเป็นความต้านทาน (ADC) เมื่ออยู่ในสภาพปกติจะมีความต้านทานสูง ในขณะที่มีความชื้นมากหรือมีปริมาณน้ำฝนในปริมาณมาก ค่าความต้านทานที่ได้จะลดลง สามารถปรับค่าความไวในการตรวจวัดได้ และสามารถให้ Output ได้ทั้ง Analog (ADC) และ Digital

Frame Detector sensor

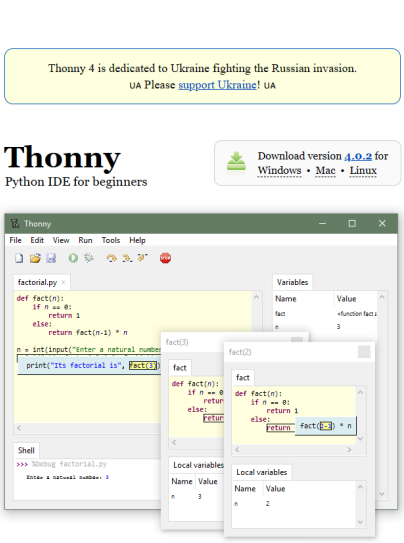
IR Flame Sensor ใช้ตรวจจับเปลวไฟโดยใช้ เซนเซอร์อินฟาเรด ให้สัญญาณเอาต์พุตออกมาทั้งแบบดิจิตอลเมื่อตรวจจับได้ให้สัญญาณเป็น 1 เมื่อตรวจจับไม่พบให้สัญญาณเป็น 0 และแบบสัญญาณอนาล็อก ให้ค่า 0-5V สามารถปรับความไวได้ที่โวลุมบนโมดูล

infrared (IR) sensor

เซ็นเซอร์ตรวจจับวัตถุ IR Infrared เป็นโมดูลเซ็นเซอร์ตรวจจับวัตถุระยะใกล้ มีหลักการทำงานโดยให้หลอด Infrared LED ทำการส่งสัญญาณ เป็นแสงอินฟราเรดออกไปตกกระทบกับวัตถุที่ตรวจพบในระยะ และทำการสะท้อนกลับมายังตัวหลอดโฟโต้ไดโอดที่ทำหน้าที่รับแสงอินฟราเรด โดยจะให้ค่า output ออกมาเป็น Digital signal แต่สำหรับบางโมดูลอาจจะรองรับ output แบบ Analog signal ด้วย ส่วนตัว R ปรับค่านั้น ใช้ในการปรับความไวต่อการตรวจจับแสงอินฟราเรด ซึ่งจะส่งผลต่อระยะในการตรวจพบวัตถุของตัวเซนเซอร์ ตัวโมดูลนี้ก็มีราคาถูก ขนาดเล็ก สะดวกในการนำไปใช้ติดตั้งกับงานจำพวก หุ่นยนต์, Smart car, หุ่นยนต์หลบสิ่งกีดขวาง เป็นต้น

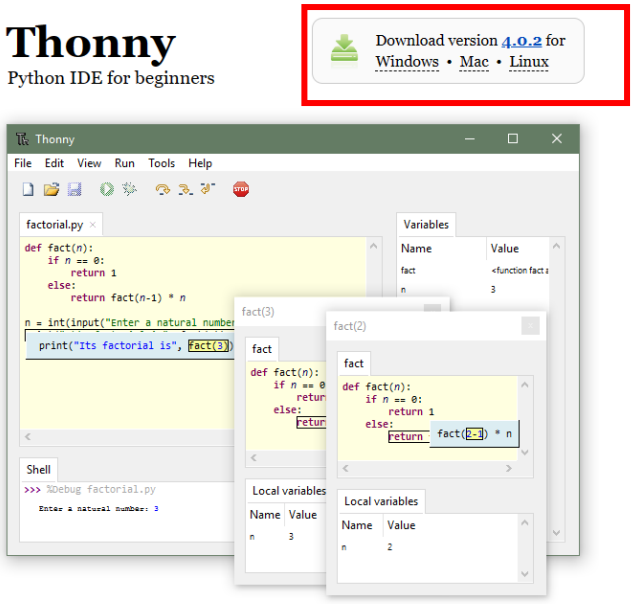
ขั้นตอนดาวน์โหลดและติดตั้งโปรแกรม IDE Thonny

1. เข้าเว็บไซต์ https://thonny.org/



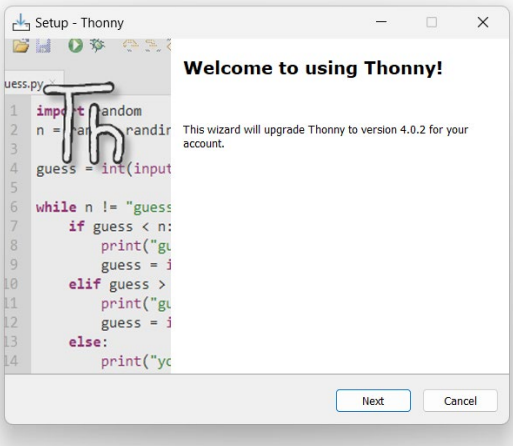
หน้าตาเว็บไซต์ Thonny

2. เลือกดาวน์โหลดตามระบบปฏิบัติการของคอมพิวเตอร์

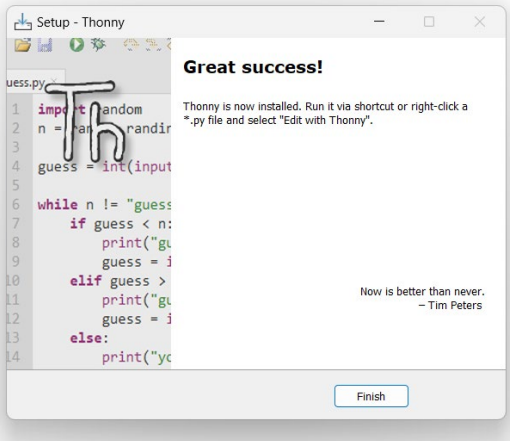


3. จากนั้นเลือกให้ตรงกับ เวอร์ชั้นของระบบปฏิบัติการของคอมพอวเตอร์ ตั้งแต่วินโดว์ 8 ขึ้นมาให้เลือกตามรูป

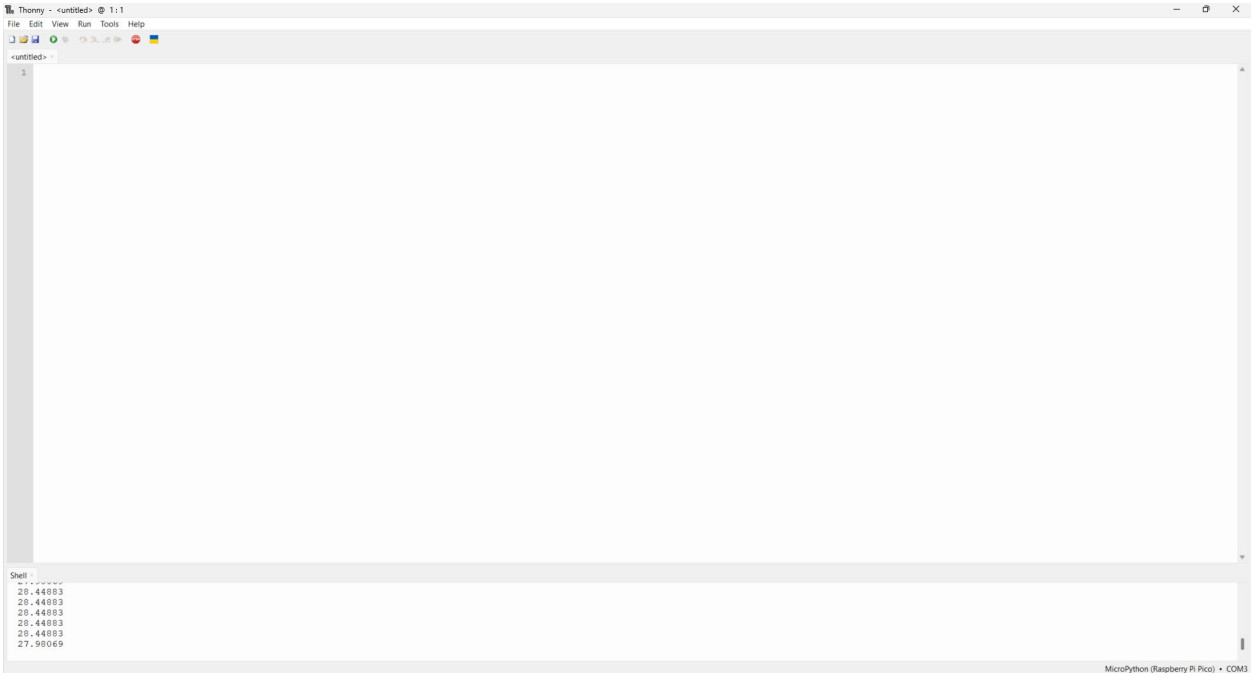
4. คลิกดาวน์โหลดที่ลิ้งค์ ตามรูปด้านบน หลังจากโหลดเสร็จแล้ว ให้คลิกเปิดตัวติดตั้ง



5. ทำการติดตั้ง ให้เสร็จสมบูณ์



หน้าต่างหลังจากติดตั้งเสร็จ



หน้าหลักของโปรแกรม

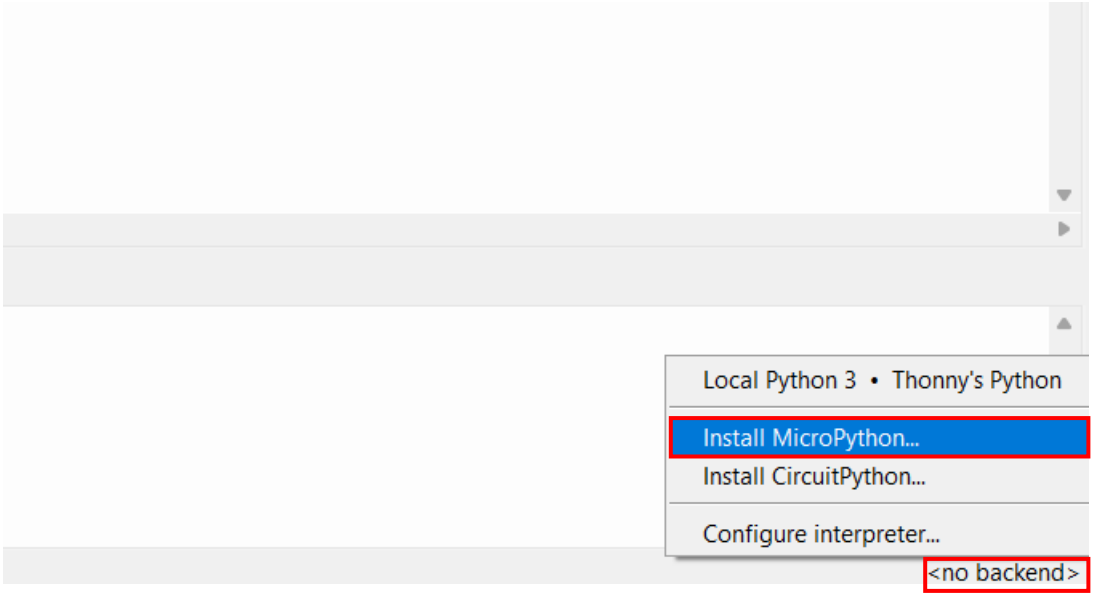
6. หลังจากติดตั้งเสร็จแล้ว เมื่อเปิดโปรแกรมจะเห็นหน้าต่างตามรูปด้านบน

ขั้นตอนการติดตั้ง Micro python ลงบนบอร์ด Raspberry pi pico w

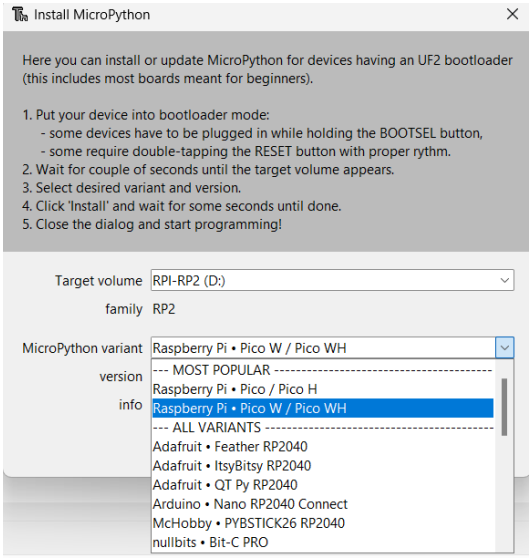
การติดตั้ง Micro python มีอยู่ 2 วิธี วิธีแรกคือการติดตั้งผ่านโปรแกรม Thonny ขั้นตอนที่สอง คือการ Flash ผ่านทาง USB คือการนำไฟล์ Micropython วางลงในโฟร์เดอร์ของตัว Raspberry Pi pico เลย

วิธีที่ 1 การติดตั้ง Micro python ผ่านโปรแกรม Thonny

1. เสียบ USB เข้ากับคอมพิวเตอร์ ด้านของ Pico ให้ กดปุ่มรีเซ็ทค้างไว้จากนั้นค่อยเสียบ(เพื่อรีเซ็ทตัว Pico)
2. หลังจากเสียบ USB แล้วให้กลับมาที่ โปรแกรม Thonny จากนั้น ให้คลิกที่ <no backend > จะแสดงหน้าต่างดังรูป จากนั้นให้เลือก Install Micropython



3. รอสักครู่ จากนั้น ให้เลือกบอร์ดให้ตรงกับที่เราใช้



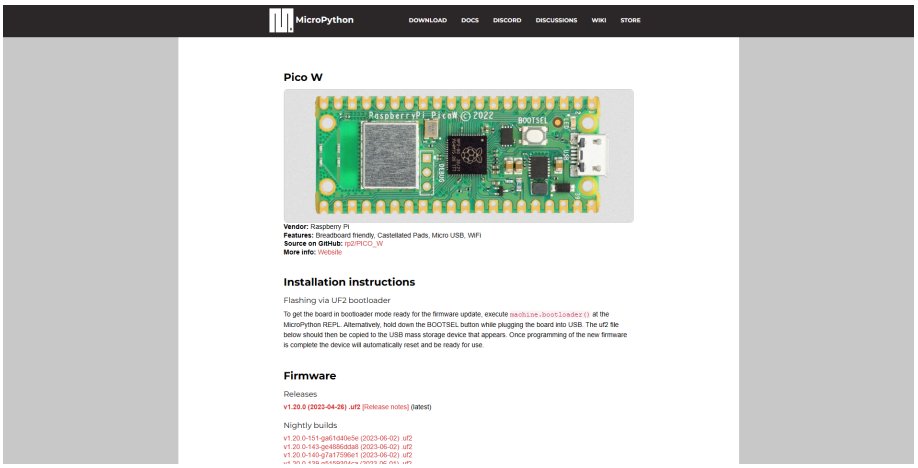
4. กด Install เมือ ติดตั้งเสร็จแล้ว สามารถโค้ดแล้วลองรันได้เลย

```
import machine
import utime

led_onboard = machine.Pin("LED", machine.Pin.OUT)

while True:
    led_onboard.value(1)
    print ('on')
    utime.sleep_ms(1000)

    led_onboard.value(0)
    print('off')
    utime.sleep_ms(1000)
```

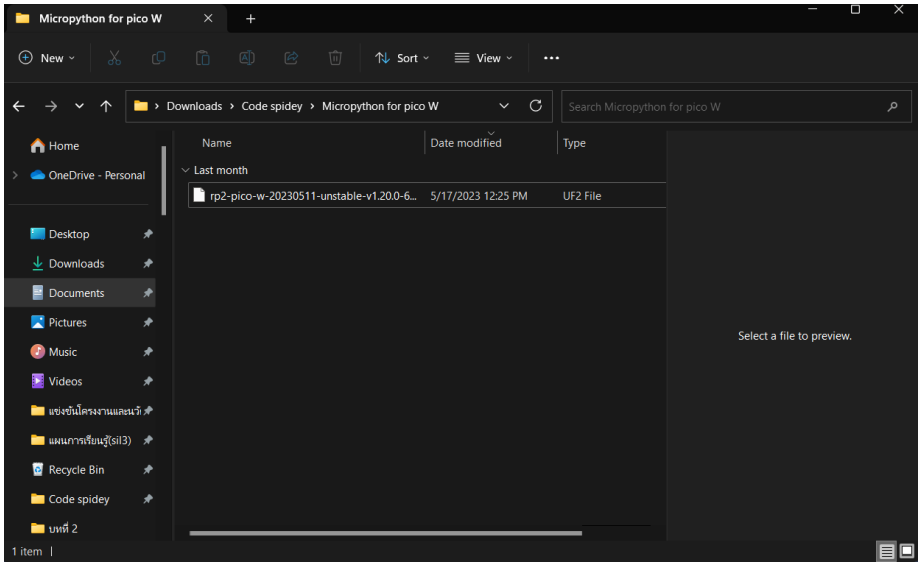
โค้ดสำหรับเทส LED ออนบอร์ดของ Pico w

วิธีที่ 2 การ Flash ผ่าน USB

1.ขั้นตอนที่ 1 เข้าเว็บไซต์ https://micropython.org/download/rp2-pico-w/



2. ดาวน์โหลด Firmware เวอร์ชั้นล่าสุด จะได้ ไฟล์ตามรูปด้านล่างมา

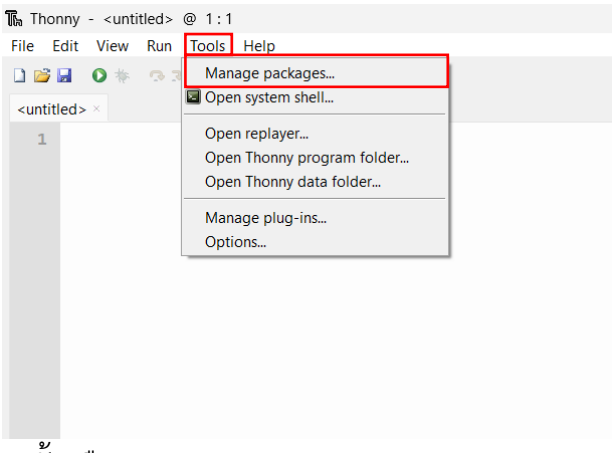


3. ให้ก็อปปี้ไฟล์ Firmware ที่เราดาวน์โหลดมา ไปวางที่โฟร์เดอร์ของ Raspberry pi

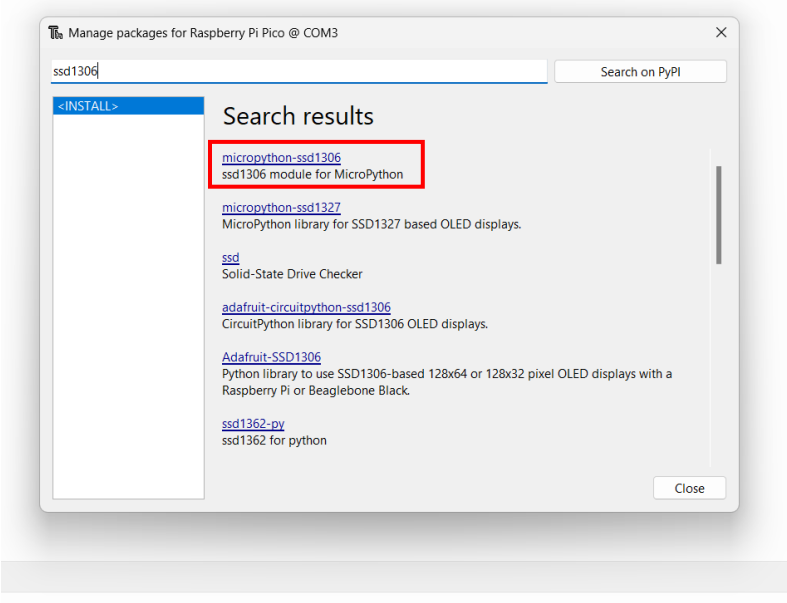
*หากไม่เจอโฟรเดอร์ของ Raspberry pi โดยให้อีกด้านของ USB เสียบกับคอมพิวเตอร์ของเราก่อน จากนั้นให้กดปุ่มรีเซ็ทบน บอร์ด Raspberry pi Pico ค้างไว้จากนั้น เสียบ USB เข้ากับตัวบอร์ด

การติดตั้ง Library ของ จอ OLED

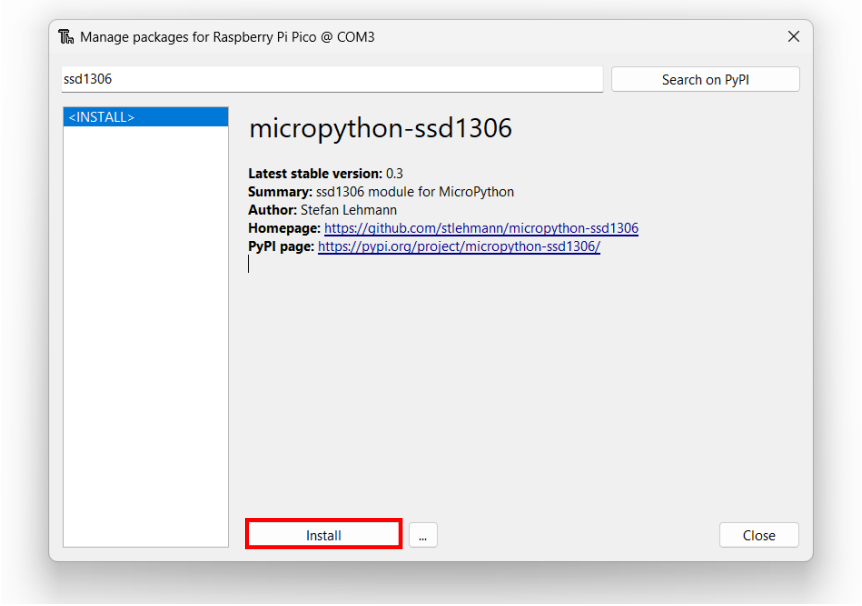
1. เชื่อมต่อ จอ OLED กับ PICO W จากนั้นเชื่อมต่อ PICO W เข้ากับ คอมพิวเตอร์ผ่าน USB



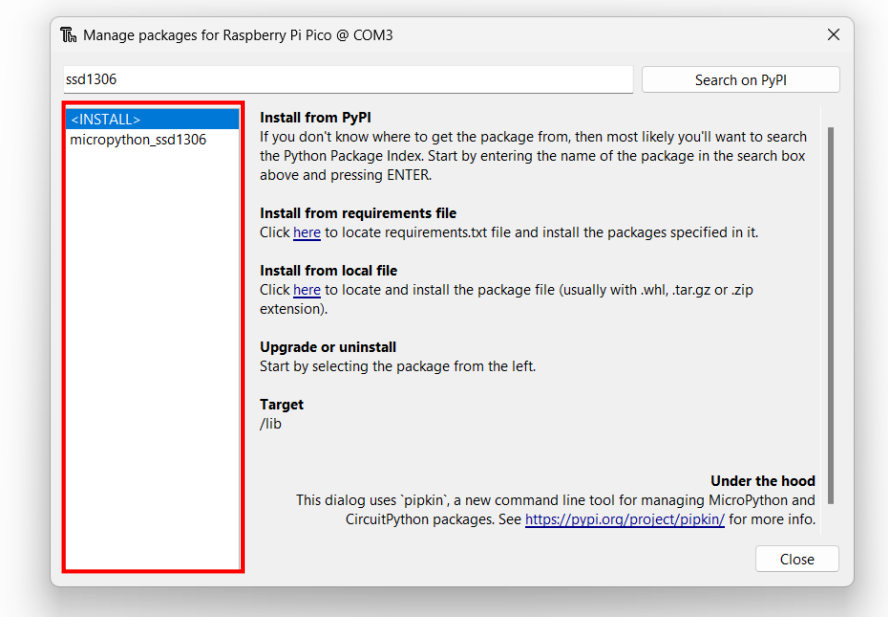
2. คลิกที่ Tools จากนั้นเลือก Manage packegas...



3. จะปรากฏหน้าต่างของ Manage packages ขึ้นมา ให้พิมพ์ ssd1306 ในช่องค้นหา จากนั้น เลือก micropython-ssd1306 ตามรูปด้านบน



4. จากนั้น กด install



5. หลังจาก ติดตั้ง library เสร็จ ต้ว library ที่ได้ติดตั้งไปจะปรากฏในช่องตามรูปด้านบน

Project

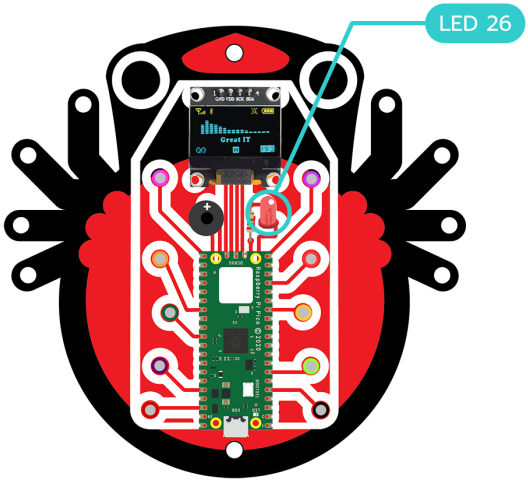
1. LED blink

เตรียมอุปกรณ์

- Spidey

การเชื่อมต่อ

- ยังไม่มีการต่อเซ็นเซอร์แยก เป็นการทดสอบ LED บนบอร์ดของ Raspberry pi pico w



การโค้ด

```
from machine import Pin #ทดสอบLED
import utime

led1 =Pin(26,Pin.OUT)
delay = .40

while True:
    led1.value(1)
    utime.sleep(delay)
    led1.value(0)
    utime.sleep(delay)
```

อธิบายโค้ด

โค้ด Python ที่ใช้เพื่อสร้างการกระพริบ LED โดยใช้โมดูล machine และ utime ใน MicroPython:

1. **from machine import Pin**: นำเข้าโมดูล Pin จากไลบรารี machine เพื่อใช้งานขาของอุปกรณ์ฮาร์ดแวร์
2. **import utime**: นำเข้าโมดูล utime เพื่อใช้งานฟังก์ชัน sleep ในการหน่วงเวลา
3. **led1 = Pin(26, Pin.OUT)**: กำหนดขาของ LED ที่เชื่อมต่อกับขา 26 และตั้งโหมดขาเป็น OUTPUT
4. **delay = 0.40**: กำหนดเวลาหน่วงระหว่างการกระพริบ LED ในหน่วยวินาที (ในที่นี้คือ 0.40 วินาที)
5. **while True:**: เริ่มลูปทำงานตลอดเวลา
6. **led1.value(1)**: กำหนดค่าให้ขา LED เป็น 1 (เปิด LED)
7. **utime.sleep(delay)**: หน่วงเวลาตามค่า delay ที่กำหนด (0.40 วินาที)
8. **led1.value(0)**: กำหนดค่าให้ขา LED เป็น 0 (ปิด LED)
9. **utime.sleep(delay)**: หน่วงเวลาตามค่า delay ที่กำหนด (0.40 วินาที)

โค้ดนี้จะทำให้ LED ที่เชื่อมต่อกับขา 26 กระพริบเปิด-ปิดในรอบนี้ตลอดไป โดยมีเวลาหน่วงระหว่างการกระพริบเป็น 0.40 วินาที

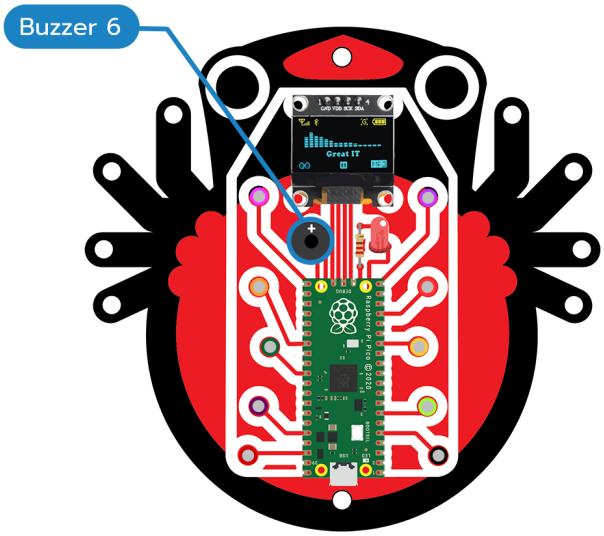
2. Buzzer

เตรียมอุปกรณ์

- Spidey
- Buzzer Active (อยู่บนบอร์ด Spidey แล้ว)

การเชื่อมต่อ

- ยังไม่มีการต่อเซ็นเซอร์แยก เป็นการทดสอบ Buzzer บนบอร์ดของ Spidey



การโค้ด

```
import utime #ทดสอบ buzzer
from machine import Pin

buzzer = Pin(6, Pin.OUT)

while True:
    buzzer.high()
    utime.sleep(1)
    buzzer.low()
    utime.sleep(1)
```

อธิบายโค้ด

โค้ด Python ที่ใช้เพื่อทดสอบ Buzzer โดยใช้โมดูล utime และ machine ใน MicroPython:

1. **import utime**: นำเข้าโมดูล utime เพื่อใช้งานฟังก์ชัน sleep ในการหน่วงเวลา
2. **from machine import Pin**: นำเข้าโมดูล Pin จากไลบรารี machine เพื่อใช้งานขาของอุปกรณ์ฮาร์ดแวร์
3. **buzzer = Pin(6, Pin.OUT)**: กำหนดขาของ Buzzer ที่เชื่อมต่อกับขา 6 และตั้งโหมดขาเป็น OUTPUT
4. **while True:**: เริ่มลูปทำงานตลอดเวลา
5. **buzzer.high()**: กำหนดค่าให้ขา Buzzer เป็นสถานะสูง (เปิด Buzzer)
6. **utime.sleep(1)**: หน่วงเวลาไว้ 1 วินาที
7. **buzzer.low()**: กำหนดค่าให้ขา Buzzer เป็นสถานะต่ำ (ปิด Buzzer)
8. **utime.sleep(1)**: หน่วงเวลาไว้ 1 วินาที

โค้ดนี้จะทำให้ Buzzer ที่เชื่อมต่อกับขา 6 เปิด-ปิดตลอดเวลา โดยมีเวลาหน่วงระหว่างการเปิด-ปิดเป็น 1 วินาที

3. Traffic Light

เตรียมอุปกรณ์

- Spidey
- Traffic light module

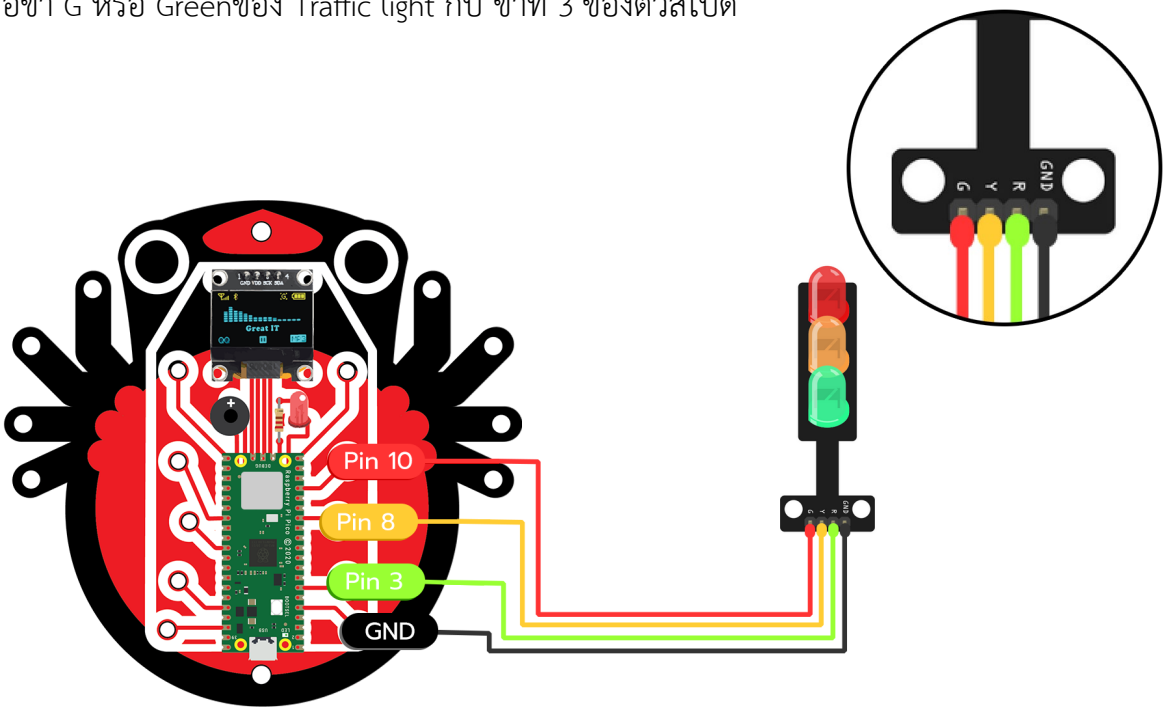
การเชื่อมต่อ

-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ Traffic light กับสไปดี้

-ต่อขา R หรือ Red ของ Traffic light กับ ขาที่ 10 ของตัวสไปดี้

-ต่อขา Y หรือ Yellow ของ Traffic light กับ ขาที่ 8 ของตัวสไปดี้

-ต่อขา G หรือ Greenของ Traffic light กับ ขาที่ 3 ของตัวสไปดี้



การโค้ด

```
from machine import Pin
import utime

led1 =Pin(3,Pin.OUT) #Green
led2 =Pin(8,Pin.OUT) #yellow
led3 =Pin(10,Pin.OUT) #Red

delay = .40
while True:
    led1.value(1)
    utime.sleep(delay)
    led1.value(0)
    utime.sleep(delay)
    led2.value(1)
    utime.sleep(delay)
    led2.value(0)
    utime.sleep(delay)
    led3.value(1)
    utime.sleep(delay)
    led3.value(0)
    utime.sleep(delay)
```

โค้ดดังกล่าวเป็นตัวอย่างการใช้ MicroPython เพื่อสลับการเปิด-ปิด LED สามสี (เขียว เหลือง และแดง) ตามลำดับ

โค้ดทำงานดังนี้:

1. **led1 = Pin(3, Pin.OUT)**, **led2 = Pin(8, Pin.OUT)**, **led3 = Pin(10, Pin.OUT)** สร้างอ็อบเจ็กต์ Pin สำหรับ LED สามสีแต่ละสี โดยกำหนดขา GPIO ที่เชื่อมต่อกับแต่ละ LED
2. **delay = .40** กำหนดค่าเวลาหน่วงระหว่างการสลับสถานะของ LED (ในหน่วยวินาที)
3. ในลูป **while True:** เปิด LED เขียวด้วย **led1.value(1)** และหน่วงเวลาด้วย **utime.sleep(delay)** จากนั้นปิด LED เขียวด้วย **led1.value(0)** และหน่วงเวลาอีกครั้ง
4. ทำขั้นตอนเดียวกันสำหรับการเปิด-ปิด LED เหลืองและ LED แดงตามลำดับ

โค้ดจะทำงานในลูปอย่างต่อเนื่องเพื่อสร้างเอฟเฟกต์การกระพริบของ LED สามสี โดยมีความหน่วงเวลาระหว่างการสลับสถานะของ LED เพื่อความช้าหรือเร็วของการกระพริบที่ต้องการได้

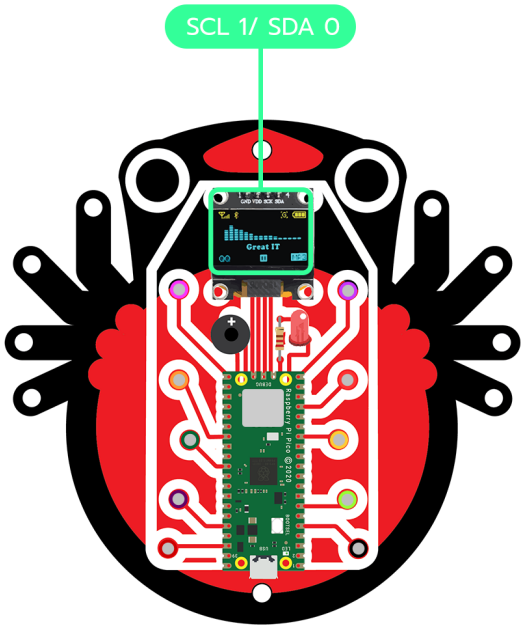
4. OLED

เตรียมอุปกรณ์

- Spidey
- จอ OLED (อยู่บนบอร์ด Spidey แล้ว)

การเชื่อมต่อ

- ยังไม่มีการต่อเซ็นเซอร์แยก เป็นการทดสอบ OLED บนบอร์ดของ Spidey



การโค้ด

*หมายเหตุ ต้องลง library 1306 ก่อนใช้งานจอ OLED

```
from machine import Pin
from machine import I2C
import ssd1306
from time import sleep

i2c=I2C(0, scl=Pin(1), sda=Pin(4))
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
oled.text('Hello world', 20, 40)
oled.show()
```

อธิบายโค้ด

โค้ด Python ที่ใช้เพื่อแสดงข้อความ "Hello world" บนจอ OLED โดยใช้โมดูล machine, I2C, ssd1306, และ time ใน MicroPython:

1. **from machine import Pin**: นำเข้าโมดูล Pin จากไลบรารี machine เพื่อใช้งานขาของอุปกรณ์ฮาร์ดแวร์
2. **from machine import I2C**: นำเข้าโมดูล I2C จากไลบรารี machine เพื่อใช้งานสื่อสารผ่าน I2C
3. **import ssd1306**: นำเข้าโมดูล ssd1306 เพื่อใช้งานจอ OLED
4. **from time import sleep**: นำเข้าฟังก์ชัน sleep จากโมดูล time เพื่อใช้ในการหน่วงเวลา
5. **i2c = I2C(0, scl=Pin(1), sda=Pin(4))**: สร้างอ็อบเจกต์ I2C โดยกำหนดขาควบคุมสัญญาณ SCL เป็นขา 1 และขาควบคุมสัญญาณ SDA เป็นขา 4
6. **oled_width = 128**: กำหนดความกว้างของจอ OLED เป็น 128 พิกเซล
7. **oled_height = 64**: กำหนดความสูงของจอ OLED เป็น 64 พิกเซล
8. **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**: สร้างอ็อบเจกต์ OLED โดยใช้ความกว้างและความสูงของจอ OLED พร้อมกับอ็อบเจกต์ I2C
9. **oled.text('Hello world', 20, 40)**: กำหนดข้อความ "Hello world" ที่ตำแหน่ง x=20, y=40 บนจอ OLED
10. **oled.show()**: แสดงข้อความที่กำหนดบนจอ OLED

โค้ดนี้จะแสดงข้อความ "Hello world" ที่ตำแหน่ง (20, 40) บนจอ OLED ที่มีขนาด 128x64 พิกเซล

5. Internal temperature print in shell

เตรียมอุปกรณ์

- Spidey

การเชื่อมต่อ

- ยังไม่มีการต่อเซ็นเซอร์แยก เป็นการทดสอบ Internal temperature บนบอร์ดของ Raspberry pi pico w

การโค้ด

```
import machine
import utime

sensor_temp = machine.ADC(4)
conversion_factor = 3.3 / (65535)

while True:
    reading = sensor_temp.read_u16() * conversion_factor
    temperature = 27 - (reading - 0.706)/0.001721
    print(temperature)

    utime.sleep(2)
```

อธิบายโค้ด

โค้ดดังกล่าวใช้ MicroPython เพื่ออ่านค่าอุณหภูมิจากเซ็นเซอร์อุณหภูมิแบบแอนะล็อกดิจิตอล (ADC) และคำนวณอุณหภูมิจากค่าที่ได้

1. **sensor_temp = machine.ADC(4)** สร้างอ็อบเจ็กต์ ADC ในขา GPIO 4 เพื่ออ่านค่าแอนะล็อกดิจิตอลจากเซ็นเซอร์อุณหภูมิ

2. **conversion_factor = 3.3 / (65535)** คำนวณตัวปรับค่าสัมประสิทธิ์สำหรับการแปลงค่า ADC เป็นค่าแอนะล็อก
3. ในลูป **while True:** อ่านค่าแอนะล็อกดิจิตอลจากเซ็นเซอร์อุณหภูมิด้วย **reading = sensor_temp.read_u16() * conversion_factor** และคำนวณอุณหภูมิโดยใช้สูตรที่กำหนด จากนั้นพิมพ์ค่าอุณหภูมิออกทางคอนโซลด้วย **print(temperature)**
4. **utime.sleep(2)** หน่วงเวลา 2 วินาทีก่อนที่จะอ่านค่าอุณหภูมิใหม่ในรอบถัดไป

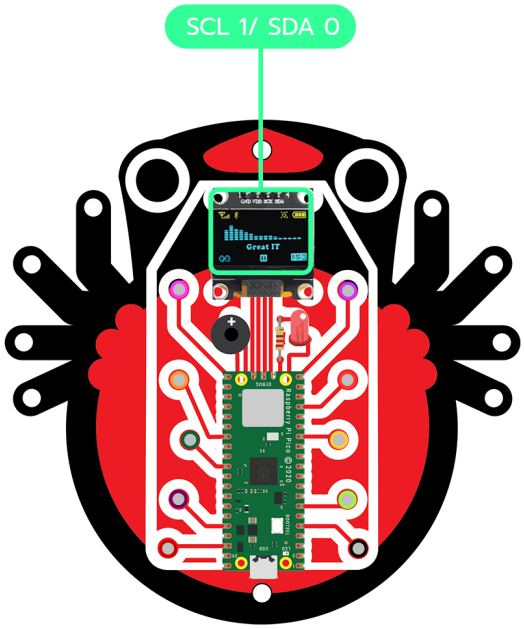
6. Internal temperature and show in OLED

เตรียมอุปกรณ์

- Spidey
- OLED

การเชื่อมต่อ

- ยังไม่มีการต่อเซ็นเซอร์แยก เป็นการทดสอบ Internal temperature บนบอร์ดของ Raspberry pi pico w และ OLED บน บอร์ด Spidey



การโค้ด

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import machine
import utime

sensor_temp = machine.ADC(4)
conversion_factor = 3.3 / (65535)

WIDTH  = 128                                      # ความกว้าง
HEIGHT = 64                                       # ความสูง

i2c = I2C(0, scl=Pin(1), sda=Pin(4), freq=200000)  # Set I2C Pin

oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)      # oled display

while True:
    reading = sensor_temp.read_u16() * conversion_factor
    temperature = 27 - (reading - 0.706)/0.001721
    print(temperature)

    oled.fill(0)   # clear OLED

    oled.text("Temp: ",6,8)   # Text
    oled.text(str(round(temperature,2)),50,8) # ค่าอุณหภูมิ
    oled.text("*C",95,8)
    utime.sleep(2)

    oled.show()
```

อธิบายโค้ด

โค้ด Python ที่ใช้เพื่ออ่านค่าอุณหภูมิจากเซ็นเซอร์และแสดงผลบนจอ OLED โดยใช้โมดูล machine, I2C, ssd1306, และ utime ใน MicroPython:

1. **from machine import Pin, I2C**: นำเข้าโมดูล Pin และ I2C จากไลบรารี machine เพื่อใช้งานขาและการสื่อสารผ่าน I2C
2. **from ssd1306 import SSD1306_I2C**: นำเข้าคลาส SSD1306_I2C จากไลบรารี ssd1306 เพื่อใช้งานจอ OLED
3. **import machine**: นำเข้าโมดูล machine เพื่อใช้งานฟังก์ชัน ADC สำหรับอ่านค่าแอนะล็อกดิจิตอล

4. **import utime**: นำเข้าโมดูล utime เพื่อใช้งานฟังก์ชัน sleep ในการหน่วงเวลา
5. **sensor_temp = machine.ADC(4)**: สร้างอ็อบเจกต์ ADC โดยกำหนดขาของเซ็นเซอร์อุณหภูมิที่เชื่อมต่อกับขา 4
6. **conversion_factor = 3.3 / (65535)**: กำหนดค่าตัวคูณเพื่อแปลงค่าอ่านจากแอนะล็อกดิจิตอลเป็นค่าแอนะล็อกแบบอนาล็อก
7. **WIDTH = 128**: กำหนดความกว้างของจอ OLED เป็น 128 พิกเซล
8. **HEIGHT = 64**: กำหนดความสูงของจอ OLED เป็น 64 พิกเซล
9. **i2c = I2C(0, scl=Pin(1), sda=Pin(4), freq=200000)**: สร้างอ็อบเจกต์ I2C โดยกำหนดขาควบคุมสัญญาณ SCL เป็นขา 1 และขาควบคุมสัญญาณ SDA เป็นขา 4 และความเร็วการสื่อสารเป็น 200000
10. **oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)**: สร้างอ็อบเจกต์ OLED โดยใช้ความกว้างและความสูงของจอ OLED พร้อมกับอ็อบเจกต์ I2C
11. **while True:**: เริ่มลูปทำงานตลอดเวลา
12. **reading = sensor_temp.read_u16() * conversion_factor**: อ่านค่าแอนะล็อกดิจิตอลจากเซ็นเซอร์อุณหภูมิและคำนวณค่าอุณหภูมิโดยใช้ตัวคูณที่กำหนดไว้
13. **temperature = 27 - (reading - 0.706) / 0.001721**: คำนวณค่าอุณหภูมิจากค่าอ่านแอนะล็อก
14. **print(temperature)**: แสดงค่าอุณหภูมิในหน้าจอทางคอนโซล
15. **oled.fill(0)**: เคลียร์จอ OLED
16. **oled.text("Temp: ", 6, 8)**: แสดงข้อความ "Temp: " ที่ตำแหน่ง x=6, y=8 บนจอ OLED
17. **oled.text(str(round(temperature, 2)), 50, 8)**: แสดงค่าอุณหภูมิที่ปัดเศษสองตำแหน่งทศนิยม ที่ตำแหน่ง x=50, y=8 บนจอ OLED
18. **oled.text("*C", 95, 8)**: แสดงสัญลักษณ์องศาเซลเซียสที่ตำแหน่ง x=95, y=8 บนจอ OLED
19. **utime.sleep(2)**: หน่วงเวลาไว้ 2 วินาที
20. **oled.show()**: แสดงผลบนจอ OLED

โค้ดนี้จะอ่านค่าอุณหภูมิจากเซ็นเซอร์และแสดงผลบนจอ OLED ที่มีขนาด 128x64 พิกเซล โดยใช้เวลาหน่วงระหว่างการอัพเดตค่าอุณหภูมิเป็น 2 วินาที

7. DHT11 and show in OLED

เตรียมอุปกรณ์

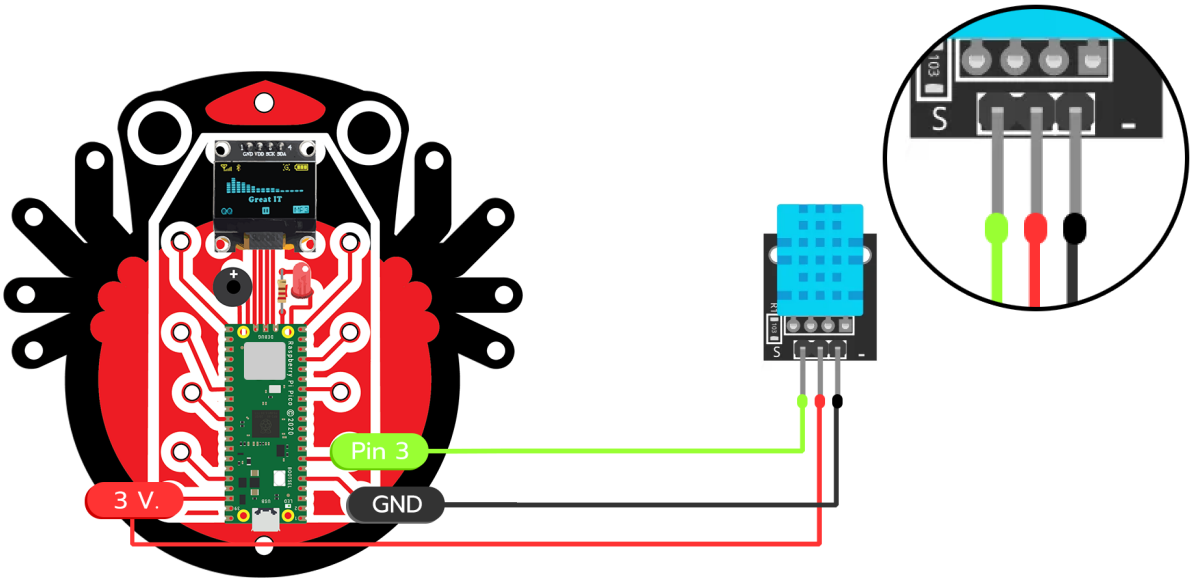
- Spidey
- OLED (บนบอร์ด Spidey)
- DHT11

การเชื่อมต่อ

-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ DHT11 กับสไปดี้ ที่ขา ลบ

-ต่อขา บวก ของ เซนเซอร์เข้ากับ สไปดี้ ที่ขา บวก

-ต่อขา S หรือ signal ของเซนเซอร์เข้ากับ ไปดี้ที่ ขา 3



การโค้ด

```
from machine import Pin
import dht
import time
from machine import I2C
import ssd1306
from time import sleep


while True:
  dhts=dht.DHT11(Pin((3)));dhts.measure();time.sleep(2)
  i2c=I2C(0, scl=Pin(1), sda=Pin(4))
  oled_width = 128
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
  oled.text(('').join([str(x) for x in ['Temperature', ' : ',
dhts.temperature()]])), 0, 30)
  oled.show()
  oled.text(('').join([str(x2) for x2 in ['humidity', ' : ',
dhts.humidity()]])), 0, 45)
  oled.show()
  oled.fill(0)
  time.sleep(1)
```

อธิบายโค้ด

โค้ด Python ที่ใช้เพื่ออ่านค่าอุณหภูมิและความชื้นจากเซ็นเซอร์ DHT11 และแสดงผลบนจอ OLED โดยใช้โมดูล machine, dht, time, I2C, ssd1306 ใน MicroPython:

1. **from machine import Pin**: นำเข้าโมดูล Pin จากไลบรารี machine เพื่อใช้งานขาของอุปกรณ์ฮาร์ดแวร์
2. **import dht**: นำเข้าโมดูล dht เพื่อใช้งานเซ็นเซอร์ DHT11
3. **import time**: นำเข้าโมดูล time เพื่อใช้งานฟังก์ชัน sleep
4. **from machine import I2C**: นำเข้าโมดูล I2C จากไลบรารี machine เพื่อใช้งานสื่อสารผ่าน I2C
5. **import ssd1306**: นำเข้าโมดูล ssd1306 เพื่อใช้งานจอ OLED
6. **while True:**: เริ่มลูปทำงานตลอดเวลา
7. **dhts = dht.DHT11(Pin(3))**: สร้างอ็อบเจกต์ DHT11 โดยกำหนดขาของเซ็นเซอร์ที่เชื่อมต่อกับขา 3
8. **dhts.measure()**: วัดค่าอุณหภูมิและความชื้นจากเซ็นเซอร์ DHT11

9. **time.sleep(2)**: หน่วงเวลา 2 วินาที
10. **i2c = I2C(0, scl=Pin(1), sda=Pin(4))**: สร้างอ็อบเจกต์ I2C โดยกำหนดขาควบคุมสัญญาณ SCL เป็นขา 1 และขาควบคุมสัญญาณ SDA เป็นขา 4
11. **oled_width = 128**: กำหนดความกว้างของจอ OLED เป็น 128 พิกเซล
12. **oled_height = 64**: กำหนดความสูงของจอ OLED เป็น 64 พิกเซล
13. **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**: สร้างอ็อบเจกต์ OLED โดยใช้ความกว้างและความสูงของจอ OLED พร้อมกับอ็อบเจกต์ I2C
14. **oled.text((''.join([str(x) for x in ['Temperature', ' : ', dhts.temperature()]])), 0, 30)**: แสดงข้อความ "Temperature : [ค่าอุณหภูมิ]" ที่ตำแหน่ง x=0, y=30 บนจอ OLED
15. **oled.show()**: แสดงผลบนจอ OLED
16. **oled.text((''.join([str(x2) for x2 in ['humidity', ' : ', dhts.humidity()]])), 0, 45)**: แสดงข้อความ "humidity : [ค่าความชื้น]" ที่ตำแหน่ง x=0, y=45 บนจอ OLED
17. **oled.show()**: แสดงผลบนจอ OLED
18. **oled.fill(0)**: เคลียร์จอ OLED
19. **time.sleep(1)**: หน่วงเวลาไว้ 1 วินาที

โค้ดนี้จะอ่านค่าอุณหภูมิและความชื้นจากเซ็นเซอร์ DHT11 และแสดงผลบนจอ OLED ที่มีขนาด 128x64 พิกเซล โดยแสดงค่าอุณหภูมิที่ตำแหน่ง x=0, y=30 และค่าความชื้นที่ตำแหน่ง x=0, y=45 บนจอ OLED และทำการเคลียร์จอ OLED และหน่วงเวลาไว้ 1 วินาทีก่อนที่จะทำซ้ำการวัดและแสดงผลอีกครั้ง

8. Rain drop sensor if detect buzzer on and show in OLED

เตรียมอุปกรณ์

- Spidey
- OLED (บนบอร์ด Spidey)
- Rain drop sensor
- Buzzer active

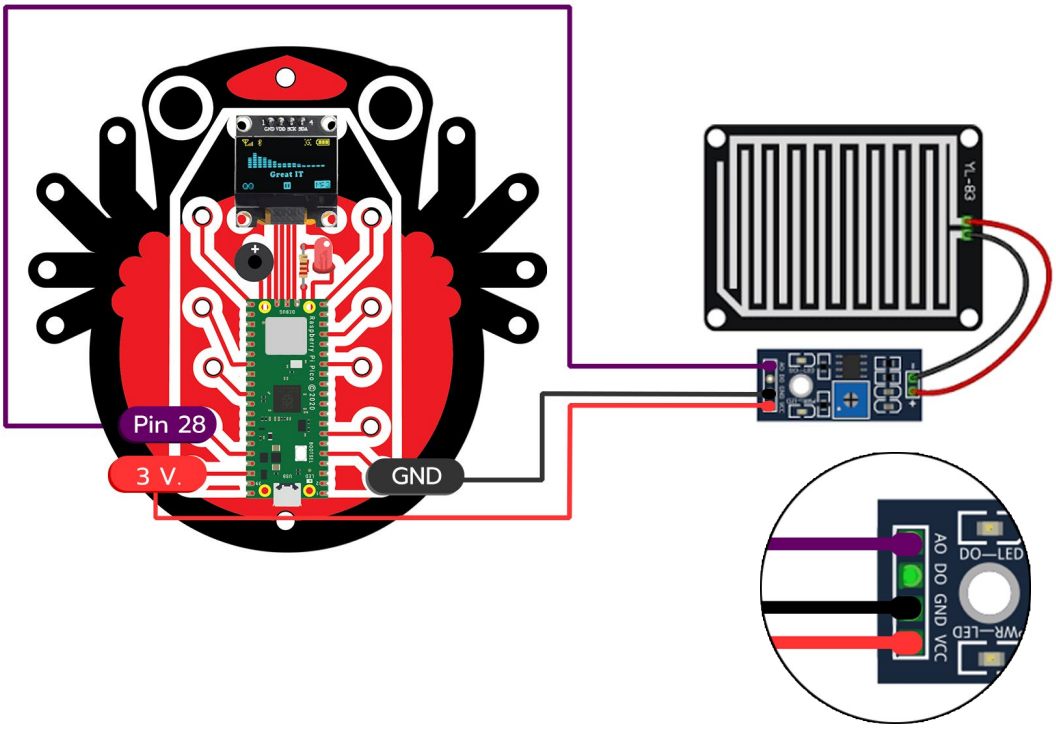
การเชื่อมต่อ

-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ เซนเซอร์วัดฝน กับสไปดี้ ที่ขา ลบ

-ต่อขา บวก ของ เซนเซอร์เข้ากับ สไปดี้ ที่ขา บวก

-เซนเซอร์ วัดฝน สามารถรับข้อมูลได้ทั้ง Analog และ digital

การเชื่อมต่อโดยเลือกใช้ Analog เพราะ ให้ค่าข้อมูลที่ระเอียดกว่า Digital โดยต่อเซนเซอร์ไปที่พินที่ 28 ของสไปดี้



การโค้ด

```
from machine import ADC
from machine import Pin
from machine import I2C
import ssd1306
from time import sleep
import time

adc28=ADC(28) #GPIO28 rain drop sensor

def gpio_set(pin,value):
  if value >= 1:
    Pin(pin, Pin.OUT).on()
  else:
    Pin(pin, Pin.OUT).off()


while True:
  if (adc28.read_u16()) < 30000:
    gpio_set((6), True)
  else:
    gpio_set((6), False)
  i2c=I2C(0, scl=Pin(1), sda=Pin(4))
  oled_width = 128
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
  oled.text(('rain :' + str(adc28.read_u16())), 40, 40)
  oled.show()
  oled.fill(0)
  time.sleep(1)
```

อธิบายโค้ด

เป็นโค้ด Python ที่ใช้สำหรับอ่านค่าจากเซ็นเซอร์ตรวจจับการตกฝนโดยใช้ ADC และแสดงค่าบนหน้าจอ OLED นี่คือการแยกแต่ละบรรทัด:

1. **from machine import ADC, Pin, I2C**: นำเข้าโมดูล ADC, Pin, และ I2C จากไลบรารี machine เพื่อใช้ในการทำงานที่เกี่ยวข้องกับฮาร์ดแวร์
2. **import ssd1306**: นำเข้าโมดูล ssd1306 เพื่อควบคุมหน้าจอ OLED
3. **from time import sleep**: นำเข้าฟังก์ชัน sleep จากโมดูล time เพื่อสร้างความหน่วงในโค้ด

4. **adc28 = ADC(28)**: สร้างอ็อบเจกต์ ADC และกำหนดให้ตัวแปร adc28 เก็บค่านี้ไว้ นี่เป็นการตั้งค่า ADC เพื่ออ่านค่าจากขา GPIO หมายเลข 28 ซึ่งเชื่อมต่อกับเซ็นเซอร์ตรวจจับการตกฝน
5. **def gpio_set(pin, value):**: นิยามฟังก์ชันชื่อ gpio_set ที่รับพารามิเตอร์เป็น pin และ value เพื่อตั้งค่าขาที่ระบุให้เปิดหรือปิดตามค่าที่กำหนด
6. **if value >= 1:**: เช็คว่าค่าที่รับเข้ามาในฟังก์ชัน gpio_set มีค่ามากกว่าหรือเท่ากับ 1 หรือไม่
7. **Pin(pin, Pin.OUT).on()**: ถ้าค่ามากกว่าหรือเท่ากับ 1 ให้ตั้งขาที่ระบุให้เป็นโหมดเอาต์พุตและเปิด
8. **Pin(pin, Pin.OUT).off()**: ถ้าค่าน้อยกว่า 1 ให้ตั้งขาที่ระบุให้เป็นโหมดเอาต์พุตและปิด
9. **while True:**: เริ่มลูปไม่จำกัดจำนวนรอบ
10. **if (adc28.read_u16()) < 30000:**: อ่านค่าแอนะล็อกจากเซ็นเซอร์ตรวจจับการตกฝนโดยใช้ ADC และเช็คว่าค่าน้อยกว่า 30000 หรือไม่
11. **gpio_set((6), True)**: ถ้าค่าน้อยกว่า 30000 ให้เรียกใช้ฟังก์ชัน gpio_set เพื่อเปิดขาหมายเลข 6
12. **gpio_set((6), False)**: ถ้าค่ามากกว่าหรือเท่ากับ 30000 ให้เรียกใช้ฟังก์ชัน gpio_set เพื่อปิดขาหมายเลข 6
13. **i2c = I2C(0, scl=Pin(1), sda=Pin(4))**: สร้างอ็อบเจกต์ I2C เพื่อใช้สื่อสารกับหน้าจอ OLED โดยใช้ขา SCL และ SDA ที่ระบุ
14. **oled_width = 128**: กำหนดความกว้างของหน้าจอ OLED เป็น 128 พิกเซล
15. **oled_height = 64**: กำหนดความสูงของหน้าจอ OLED เป็น 64 พิกเซล
16. **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**: สร้างอ็อบเจกต์ SSD1306_I2C เพื่อควบคุมหน้าจอ OLED ด้วยความกว้างและความสูงที่ระบุ และอ็อบเจกต์ I2C
17. **oled.text(('rain :' + str(adc28.read_u16())), 40, 40)**: กำหนดข้อความที่จะแสดงบนหน้าจอ OLED เป็น 'rain :' ตามด้วยค่าที่อ่านจากเซ็นเซอร์ตรวจจับการตกฝน ข้อความนี้จะแสดงที่ตำแหน่งพิกเซล (40, 40) บนหน้าจอ
18. **oled.show()**: อัปเดตหน้าจอ OLED เพื่อแสดงข้อความ
19. **oled.fill(0)**: เคลียร์หน้าจอ OLED โดยเติมค่าศูนย์ในทุกๆ พิกเซล
20. **time.sleep(1)**: หยุดรันโปรแกรมเป็นเวลา 1 วินาที

โค้ดนี้จะวนลูปอ่านค่าจากเซ็นเซอร์ตรวจจับการตกฝนโดยใช้ ADC และแสดงค่าบนหน้าจอ OLED ซึ่งจะเรียกใช้ฟังก์ชัน gpio_set เพื่อเปิดหรือปิดขาแสดงสถานะตามค่าที่อ่านได้ และจะเริ่มลูปใหม่หลังจากนั้น

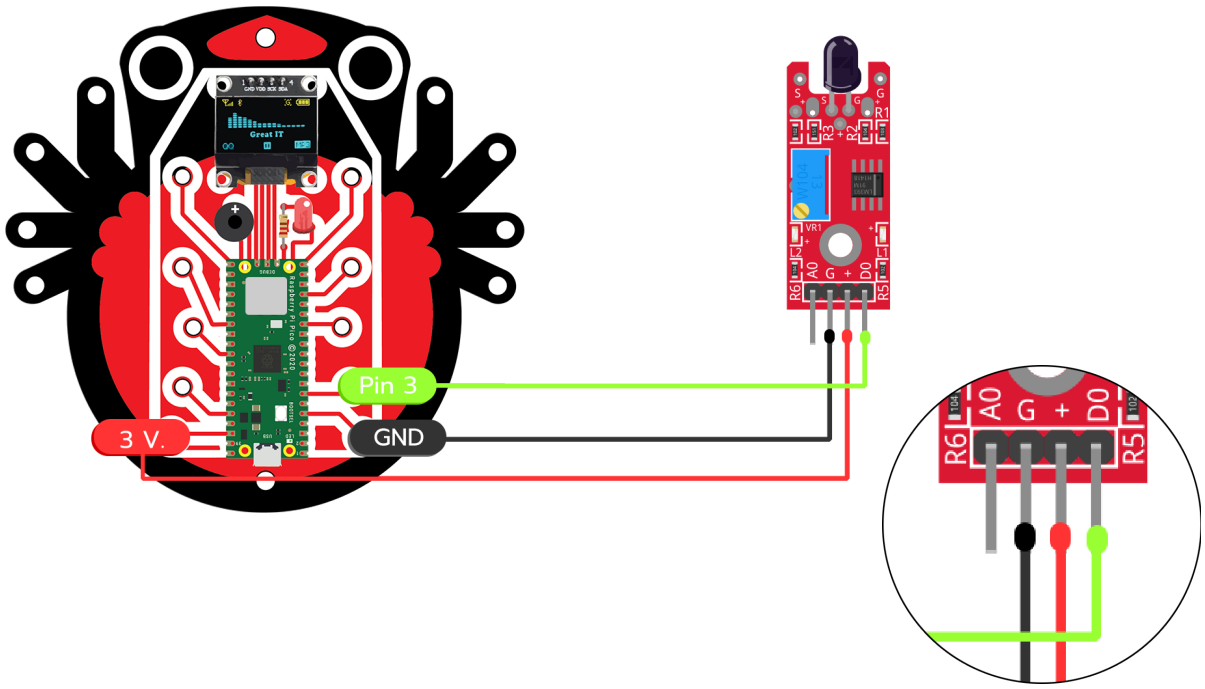
9.frame detector sensor show in OLED

เตรียมอุปกรณ์

- Spidey
- OLED (บนบอร์ด Spidey)
- frame detector sensor

การเชื่อมต่อ

-ตัวเซนเซอร์จะคล้ายกับ เซนเซอร์วัดฝน แต่เซนเซอร์วัดไฟ ตัวนี้เราจะใช้ Digital และปรับค่าความเข้มข้นของแสงที่ตัวเซ้นเซอร์แทน เพื่อให้ง่ายต่อการใช้งาน โดยให้ต่อเซนเซอร์เข้ากับพินที่ 3 ของสไปดี้



การโค้ด

```
from machine import Pin # frame detector sensor
from machine import I2C
import ssd1306
from time import sleep
import time

pIn3=Pin(3, Pin.IN, Pin.PULL_UP)


while True:
  i2c=I2C(0, scl=Pin(1), sda=Pin(4))
  oled_width = 128
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
  oled.text(('').join([str(x) for x in ['frame', ' : ',
pIn3.value()]])), 40, 47)
  oled.show()
  oled.fill(0)
  time.sleep(1)
```

อธิบายโค้ด

1. นำเข้าไลบรารีและคลาสที่ใช้ในโค้ด:
    - **from machine import Pin**: นำเข้าคลาส **Pin** จากไลบรารี **machine** เพื่อใช้สำหรับควบคุมขา GPIO.
    - **from machine import I2C**: นำเข้าคลาส **I2C** จากไลบรารี **machine** เพื่อใช้สำหรับการเชื่อมต่อแบบ I2C.
    - **import ssd1306**: นำเข้าไลบรารี **ssd1306** เพื่อใช้สำหรับควบคุมหน้าจอ OLED.
    - **from time import sleep**: นำเข้าฟังก์ชัน **sleep** จากไลบรารี **time** เพื่อใช้ในการหน่วงเวลา.
2. กำหนดขาและอ็อบเจ็กต์สำหรับเซ็นเซอร์และหน้าจอ OLED:
    - **pIn3 = Pin(3, Pin.IN, Pin.PULL_UP)**: สร้างอ็อบเจ็กต์ **pIn3** สำหรับขา GPIO 3 โดยกำหนดให้เป็นขาแสดงสถานะ (input) และตั้งค่าให้มีการเชื่อมต่อตัวยึด (pull-up) เพื่อใช้ในการตรวจจับเฟรม.

- **i2c = I2C(0, scl=Pin(1), sda=Pin(4))**: สร้างอ็อบเจ็กต์ **i2c** สำหรับการเชื่อมต่อแบบ I2C โดยกำหนดขา SCL เป็น GPIO 1 และขา SDA เป็น GPIO 4.
- **oled_width = 128** และ **oled_height = 64**: กำหนดขนาดความกว้างและความสูงของหน้าจอ OLED.
- **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**: สร้างอ็อบเจ็กต์ **oled** สำหรับควบคุมหน้าจอ OLED ด้วยค่าความกว้างและความสูงที่กำหนด.

3. เริ่มลูปการทำงานอย่างต่อเนื่อง:
    - **while True:**: เริ่มลูปการทำงานอย่างต่อเนื่อง.
4. อ่านค่าสถานะของเฟรมและแสดงผลบนหน้าจอ OLED:
    - **oled.text((''.join([str(x) for x in ['frame', ' : ', pIn3.value()]])), 40, 47)**: สร้างข้อความที่จะแสดงบนหน้าจอ OLED โดยรวมข้อความ "frame : " และค่าสถานะของเฟรมจากการอ่านค่าจากขา GPIO 3.
    - **oled.show()**: แสดงผลบนหน้าจอ OLED.
    - **oled.fill(0)**: เคลียร์หน้าจอ OLED เพื่อเตรียมแสดงผลใหม่ในรอบถัดไป.
    - **time.sleep(1)**: หน่วงเวลา 1 วินาทีก่อนที่จะเริ่มการทำงานในรอบถัดไป.

โค้ดนี้จะทำงานอย่างต่อเนื่องเพื่อตรวจจับเฟรมและแสดงผลสถานะของเฟรมบนหน้าจอ OLED ด้วยการอ่านค่าจากขา GPIO 3 และควบคุมการแสดงผลบนหน้าจอ OLED ผ่าน I2C ตลอดเวลาที่โปรแกรมทำงานอยู่ในลูป while True.

10. frame detector sensor if detect Buzzer Passive on and show in OLED

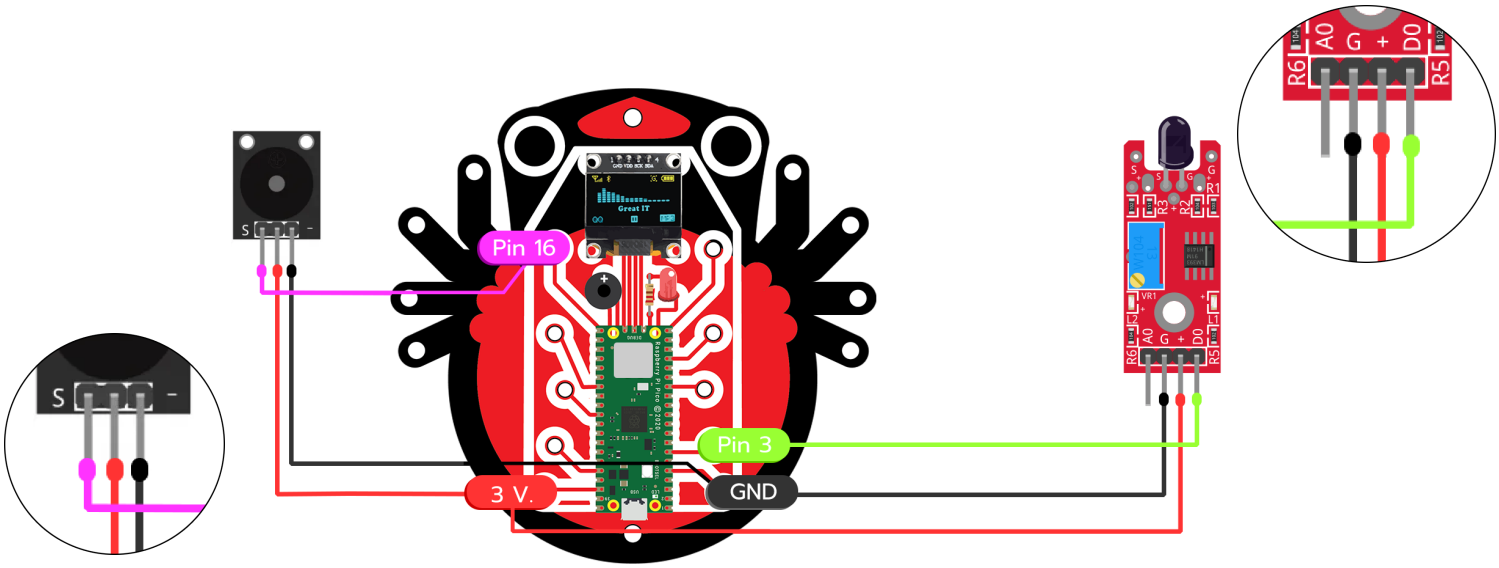
เตรียมอุปกรณ์

- Spidey
- OLED (บนบอร์ด Spidey)
- frame detector sensor
- Buzzer Passive

การเชื่อมต่อ

-ตัวเซนเซอร์จะคล้ายกับ เซนเซอร์วัดฝน แต่เซนเซอร์วัดไฟ ตัวนี้เราจะใช้ Digital และปรับค่าความเข้มข้นของแสงที่ตัวเซ้นเซอร์แทน เพื่อให้ง่ายต่อการใช้งาน โดยให้ต่อเซนเซอร์เข้ากับพินที่ 3 ของสไปดี้

-โปรเจกต์นี้เพิ่ม Buzzer แบบ Passive เข้ามา โดยต่อขาที่ 16 ของสไปดี้



การโค้ด

```
from machine import Pin
from machine import I2C
import ssd1306
from time import sleep
from machine import PWM
import time

pIn3=Pin(3, Pin.IN, Pin.PULL_UP)


while True:
  i2c=I2C(0, scl=Pin(1), sda=Pin(4))
  oled_width = 128
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
  oled.text(('').join([str(x) for x in ['frame', ' : ',
pIn3.value()]])), 40, 47)
  oled.show()
  if (pIn3.value()) == 0:
    pwm15 = PWM(Pin(15))
    pwm15.freq(440)
    pwm15.duty_u16(10000)
    time.sleep(1)
  else:
    pwm15 = PWM(Pin(15))
    pwm15.freq(1000)
    pwm15.duty_u16(0)
  oled.fill(0)
  time.sleep(1)

  time.sleep(1)
```

อธิบายโค้ด

โค้ดนี้เป็นตัวอย่างการใช้เซ็นเซอร์ตรวจจับเฟรม (frame detector sensor) ร่วมกับหน้าจอ OLED และบัซเซอร์ (buzzer) ดังนี้:

1. นำเข้าโมดูลและตั้งค่าตัวแปร:
    - **from machine import Pin**: นำเข้าโมดูล Pin เพื่อใช้ในการควบคุมขา GPIO.

- **from machine import I2C**: นำเข้าโมดูล I2C เพื่อใช้ในการสื่อสารผ่าน I2C.
- **import ssd1306**: นำเข้าโมดูล ssd1306 เพื่อใช้ในการควบคุมหน้าจอ OLED.
- **from time import sleep**: นำเข้าฟังก์ชัน sleep เพื่อใช้ในการหน่วงเวลา.
- **from machine import PWM**: นำเข้าโมดูล PWM เพื่อใช้ในการควบคุมสัญญาณ PWM.
- **import time**: นำเข้าโมดูล time เพื่อใช้ในการหน่วงเวลา.

2. กำหนดขา GPIO ที่เชื่อมต่อกับเซ็นเซอร์ตรวจจับเฟรม:
    - **pIn3=Pin(3, Pin.IN, Pin.PULL_UP)**: กำหนดขา GPIO 3 ให้เป็นขาป้องกันแบบ Pull-up.
3. เริ่มลูปการทำงานอย่างต่อเนื่อง:
    - **while True:**: เริ่มลูปการทำงานอย่างต่อเนื่อง.
4. กำหนดการเชื่อมต่อและตั้งค่าหน้าจอ OLED:
    - **i2c=I2C(0, scl=Pin(1), sda=Pin(4))**: สร้างอ็อบเจ็กต์ I2C เพื่อเชื่อมต่อกับหน้าจอ OLED ที่ขา SCL (GPIO 1) และ SDA (GPIO 4).
    - **oled_width = 128** และ **oled_height = 64**: กำหนดขนาดความกว้างและความสูงของหน้าจอ OLED.
    - **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**: สร้างอ็อบเจ็กต์ OLED ด้วยขนาดความกว้างและความสูงที่กำหนด.
5. แสดงผลสถานะของเฟรมบนหน้าจอ OLED:
    - **oled.text((''.join([str(x) for x in ['frame', ' : ', pIn3.value()]])), 40, 47)**: สร้างข้อความที่แสดงสถานะของเฟรมโดยใช้ค่าจากการอ่านขา GPIO 3 และแสดงผลที่พิกัด (40, 47) บนหน้าจอ OLED.
    - **oled.show()**: แสดงผลที่ออกแบบไว้บนหน้าจอ OLED.
6. ควบคุมเสียงด้วยสัญญาณ PWM ตามสถานะของเฟรม:
    - **if (pIn3.value()) == 0:**: ตรวจสอบว่าสถานะของเฟรมเป็น 0 หรือไม่ (ไม่มีการตรวจจับเฟรม).
        - **pwm15 = PWM(Pin(15))**: สร้างอ็อบเจ็กต์ PWM ที่ขา GPIO 15.
        - **pwm15.freq(440)**: กำหนดความถี่ของสัญญาณ PWM เป็น 440 Hz.
        - **pwm15.duty_u16(10000)**: กำหนดระดับความสว่างสูงสุดของสัญญาณ PWM เป็น 10000.
        - **time.sleep(1)**: หน่วงเวลา 1 วินาที.
    - **else:**: กรณีสถานะของเฟรมไม่เป็น 0 (มีการตรวจจับเฟรม).
        - **pwm15 = PWM(Pin(15))**: สร้างอ็อบเจ็กต์ PWM ที่ขา GPIO 15.

- **pwm15.freq(1000)**: กำหนดความถี่ของสัญญาณ PWM เป็น 1000 Hz.
- **pwm15.duty_u16(0)**: กำหนดระดับความสว่างของสัญญาณ PWM เป็น 0.

7. เคลียร์หน้าจอ OLED และหน่วงเวลา:
    - **oled.fill(0)**: เคลียร์หน้าจอ OLED เป็นสีดำ.
    - **time.sleep(1)**: หน่วงเวลา 1 วินาที.

โค้ดดังกล่าวจะทำงานอย่างต่อเนื่อง โดยตรวจจับสถานะของเฟรมผ่านเซ็นเซอร์ตรวจจับเฟรมที่เชื่อมต่อกับขา GPIO 3 และแสดงผลสถานะบนหน้าจอ OLED พร้อมกับส่งสัญญาณ PWM เพื่อควบคุมเสียงบัซเซอร์ที่ขา GPIO 15 ตามสถานะของเฟรมที่ตรวจจับได้

11. frame detector sensor if detect Buzzer Passive and RGB turn on show in OLED

เตรียมอุปกรณ์

- Spidey
- OLED (บนบอร์ด Spidey)
- frame detector sensor
- Buzzer Passive
- RGB

การเชื่อมต่อ

-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ เซนเซอร์วัดไฟกับสไปดี้ ที่ขา ลบ

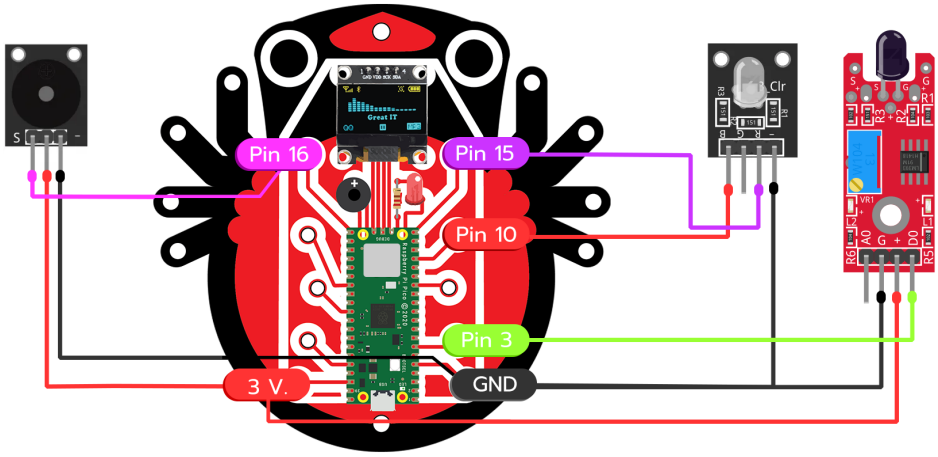
-ต่อขา บวก ของ เซนเซอร์เข้ากับ สไปดี้ ที่ขา บวก

-ตัวเซนเซอร์จะคล้ายกับ เซนเซอร์วัดฝน แต่เซนเซอร์ วัดไฟ ตัวนี้เราจะใช้ Digital และปรับค่าความเข้มข้นของแสงที่ตัวเซ้นเซอร์แทน เพื่อให้ง่ายต่อการใช้งาน โดยให้ต่อเซนเซอร์เข้ากับพินที่ 3 ของสไปดี้

- Buzzer แบบ Passive โดยต่อขาที่ 16 ของสไปดี้

-โดยโปรเจกต์นี้เราเพิ่ม การแจ้งเตือนให้ชัดเจนขึ้นอีกโดยการเพิ่ม RGB เข้ามา และ ขา R หรือ Red ของ RGB ต่อขา 15 และ B ต่อที่ขา 10 ของสไปดี้

*สามารถใช้ Traffic light แทน RGB ได้



การโค้ด

```
from machine import Pin
from machine import I2C
import ssd1306
from time import sleep
from machine import PWM
import time

pIn3=Pin(3, Pin.IN, Pin.PULL_UP)

def gpio_set(pin,value):
  if value >= 1:
    Pin(pin, Pin.OUT).on()
  else:
    Pin(pin, Pin.OUT).off()


while True:
  i2c=I2C(0, scl=Pin(1), sda=Pin(4))
  oled_width = 128
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
  oled.text((''.join([str(x) for x in ['frame', ' : ',
pIn3.value()]])), 40, 47)
  oled.show()
  if (pIn3.value()) == 0:
    pwm15 = PWM(Pin(16)) #buzzer
```

```
    pwm15.freq(1000)
    pwm15.duty_u16(1000)
    gpio_set((10), True)   #RGB
    gpio_set((15), False)  #RGB
    time.sleep(1)
  else:
    pwm15 = PWM(Pin(16)) #buzzer
    pwm15.freq(1000)
    pwm15.duty_u16(0)
    gpio_set((15), False)  #RGB
    gpio_set((10), True)   #RGB
  oled.fill(0)
  time.sleep(1)
```

อธิบายโค้ด

โค้ดดังกล่าวเพิ่มการควบคุมสีของ LED RGB เพิ่มเติม

1. นำเข้าโมดูล **Pin**, **I2C**, **ssd1306**, **sleep**, **PWM** และ **time**.
2. กำหนดขา GPIO 3 เป็นขานำเข้าแบบ Pull-up ซึ่งเป็นขาที่ใช้ตรวจจับสถานะของเฟรม:
    - **pIn3 = Pin(3, Pin.IN, Pin.PULL_UP)**
3. กำหนดฟังก์ชัน **gpio_set()** เพื่อควบคุมสถานะของขา GPIO ให้เป็นสูงหรือต่ำ:
    - **def gpio_set(pin, value):**: ประกาศฟังก์ชัน **gpio_set()** รับพารามิเตอร์ **pin** และ **value**.
    - **if value >= 1:**: ตรวจสอบค่า **value** ว่ามากกว่าหรือเท่ากับ 1 หรือไม่.
        - **Pin(pin, Pin.OUT).on()**: กำหนดขา GPIO ให้เป็นสูง.
    - **else:**: กรณีค่า **value** น้อยกว่า 1.
        - **Pin(pin, Pin.OUT).off()**: กำหนดขา GPIO ให้เป็นต่ำ.
4. เริ่มต้นการทำงานในลูป **while True:**.
5. สร้างอ็อบเจ็กต์ I2C สำหรับการเชื่อมต่อกับหน้าจอ OLED:
    - **i2c = I2C(0, scl=Pin(1), sda=Pin(4))**
6. กำหนดขนาดของหน้าจอ OLED:
    - **oled_width = 128**
    - **oled_height = 64**
7. สร้างอ็อบเจ็กต์ SSD1306_I2C สำหรับการควบคุมหน้าจอ OLED:
    - **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**

8. แสดงผลสถานะของเฟรมบนหน้าจอ OLED:
    - **oled.text((''.join([str(x) for x in ['frame', ' : ', pIn3.value()]])), 40, 47)**: สร้างข้อความที่แสดงสถานะของเฟรมโดยใช้ค่าจากการอ่านขา GPIO 3 และแสดงผลที่พิกัด (40, 47) บนหน้าจอ OLED.
9. แสดงหน้าจอ OLED ที่มีข้อความและสถานะของเฟรม:
    - **oled.show()**
10. ตรวจสอบสถานะของเฟรม:
    - **if (pIn3.value()) == 0:**: ตรวจสอบว่าค่าที่อ่านจากขา GPIO 3 เป็น 0 หรือไม่.
        - **pwm15 = PWM(Pin(15))**: สร้างอ็อบเจ็กต์ PWM สำหรับควบคุมขา GPIO 15 (Buzzer).
        - **pwm15.freq(1000)**: ตั้งค่าความถี่ให้เป็น 1000 Hz.
        - **pwm15.duty_u16(1000)**: ตั้งค่า Duty Cycle เป็น 1000.
        - **gpio_set((13), True)**: เปิด LED RGB โดยกำหนดขา GPIO 13 เป็นสูง (สีแดง).
        - **gpio_set((14), False)**: ปิด LED RGB โดยกำหนดขา GPIO 14 เป็นต่ำ (สีเขียว).
        - **time.sleep(1)**: หน่วงเวลา 1 วินาที.
- **else:**: กรณีค่าที่อ่านจากขา GPIO 3 ไม่ใช่ 0.
    - **pwm15 = PWM(Pin(15))**: สร้างอ็อบเจ็กต์ PWM สำหรับควบคุมขา GPIO 15 (Buzzer).
    - **pwm15.freq(1000)**: ตั้งค่าความถี่ให้เป็น 1000 Hz.
    - **pwm15.duty_u16(0)**: ตั้งค่า Duty Cycle เป็น 0.
    - **gpio_set((13), False)**: ปิด LED RGB โดยกำหนดขา GPIO 13 เป็นต่ำ (สีแดง).
    - **gpio_set((14), True)**: เปิด LED RGB โดยกำหนดขา GPIO 14 เป็นสูง (สีเขียว).
11. เตรียมหน้าจอ OLED สำหรับแสดงข้อมูลถัดไป:
    - **oled.fill(0)**: เตรียมหน้าจอ OLED โดยล้างสีดำทั้งหน้าจอ.
12. หน่วงเวลา 1 วินาที:
    - **time.sleep(1)**

12. Light sensor show in OLED

เตรียมอุปกรณ์

- Spidey
- OLED (บนบอร์ด Spidey)
- Light sensor

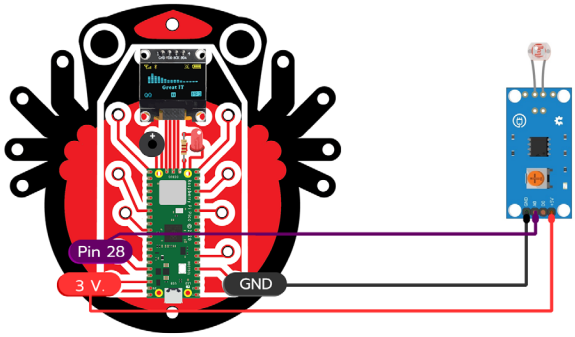
การเชื่อมต่อ

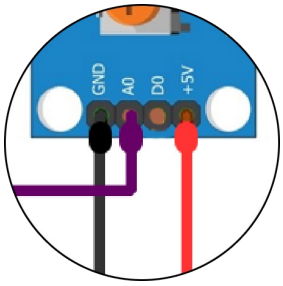
-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ Light sensor กับสไปดี้ ที่ขา ลบ

-ต่อขา บวก ของ เซนเซอร์เข้ากับ สไปดี้ ที่ขา บวก

-โดยการเชื่อมต่อเลือกเป็น analog เพื่อให้เห็นค่าของแสง ได้ละเอียดขึ้น

โดยต่อเซนเซอร์ เข้ากับสไปดี้ที่ ขา 28





การโค้ด

```
from machine import Pin
from machine import I2C
import ssd1306
from time import sleep
from machine import ADC
import time

adc28=ADC(28)


while True:
  i2c=I2C(0, scl=Pin(1), sda=Pin(4))
  oled_width = 128
```

```
  oled_height = 64
  oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
  oled.text((''.join([str(x) for x in ['Light', ' : ',
adc28.read_u16()]])), 40, 47)
  oled.show()
  oled.fill(0)
  time.sleep(1)
```

อธิบายโค้ด

โค้ดดังกล่าวเป็นการวัดค่าแสงและแสดงผลบนหน้าจอ OLED:

1. นำเข้าโมดูล **Pin**, **I2C**, **ssd1306**, **sleep**, **ADC**, และ **time**.
2. กำหนดขา ADC 28 เพื่อใช้ในการวัดค่าแสง:
    - **adc28 = ADC(28)**
3. เริ่มต้นการทำงานในลูป **while True:**.
4. สร้างอ็อบเจ็กต์ I2C สำหรับการเชื่อมต่อกับหน้าจอ OLED:
    - **i2c = I2C(0, scl=Pin(1), sda=Pin(4))**
5. กำหนดขนาดของหน้าจอ OLED:
    - **oled_width = 128**
    - **oled_height = 64**
6. สร้างอ็อบเจ็กต์ SSD1306_I2C สำหรับการควบคุมหน้าจอ OLED:
    - **oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)**
7. แสดงผลค่าแสงบนหน้าจอ OLED:
    - **oled.text((''.join([str(x) for x in ['Light', ' : ', adc28.read_u16()]])), 40, 47)**: สร้างข้อความที่แสดงค่าแสงโดยใช้ค่าที่อ่านจาก ADC 28 และแสดงผลที่พิกัด (40, 47) บนหน้าจอ OLED.
8. แสดงหน้าจอ OLED ที่มีข้อความและค่าแสง:
    - **oled.show()**
9. เตรียมหน้าจอ OLED สำหรับแสดงข้อมูลถัดไป:
    - **oled.fill(0)**: เตรียมหน้าจอ OLED โดยล้างสีดำทั้งหน้าจอ.
10. หน่วงเวลา 1 วินาที:
    - **time.sleep(1)**

13. Passive buzzer

เตรียมอุปกรณ์

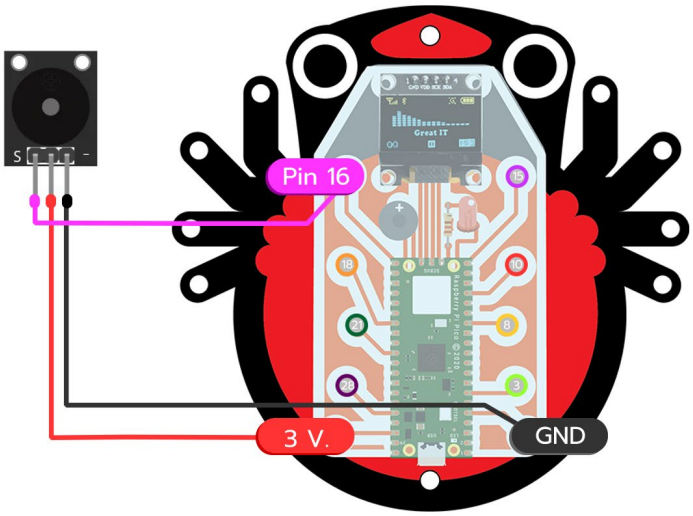
- Spidey
- buzzer passive

การเชื่อมต่อ

-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ Buzzer กับสไปดี้ ที่ขา ลบ

-ต่อขา บวก ของ เซนเซอร์เข้ากับ สไปดี้ ที่ขา บวก

-ต่อขา S ที่ buzzer ที่ขา 16 ของสไปดี้



การโค้ด

```
from machine import Pin, PWM
from time import sleep

buzzerPIN=16
BuzzerObj=PWM(Pin(buzzerPIN))

def
buzzer(buzzerPinObject,frequency,sound_duration,silence_duration
):
    # Set duty cycle to a positive value to emit sound from
```

```
buzzer
    buzzerPinObject.duty_u16(int(65536*0.2))
    # Set frequency
    buzzerPinObject.freq(frequency)
    # wait for sound duration
    sleep(sound_duration)
    # Set duty cycle to zero to stop sound
    buzzerPinObject.duty_u16(int(65536*0))
    # Wait for sound interrumption, if needed
    sleep(silence_duration)


# Play following notes by changing frequency:
#C (DO)
buzzer(BuzzerObj,523,0.5,0.1)

#D (RE)
buzzer(BuzzerObj,587,0.5,0.1)

#E (MI)
buzzer(BuzzerObj,659,0.5,0.1)

#F (FA)
buzzer(BuzzerObj,698,0.5,0.1)

#G (SOL)
buzzer(BuzzerObj,784,0.5,0.1)

#A (LA)
buzzer(BuzzerObj,880,0.5,0.1)

#B (SI)
buzzer(BuzzerObj,987,0.5,0.1)

#Deactivates the buzzer
BuzzerObj.deinit()
```

โค้ดดังกล่าวเป็นตัวอย่างการใช้ MicroPython เพื่อเล่นเสียงบนบัซเซอร์ (Buzzer) โดยกำหนดค่าความถี่ของเสียงแต่ละโน้ตและระยะเวลาการเล่นเสียงและช่วงเวลาเงียบ

โค้ดทำงานดังนี้:

1. **buzzerPIN = 16** กำหนดขา GPIO ที่เชื่อมต่อกับบัซเซอร์
2. **BuzzerObj = PWM(Pin(buzzerPIN))** สร้างอ็อบเจ็กต์ PWM สำหรับการควบคุมบัซเซอร์

3. **def buzzer(buzzerPinObject, frequency, sound_duration, silence_duration):** สร้างฟังก์ชัน buzzer เพื่อกำหนดการเล่นเสียงบนบัซเซอร์
4. ในฟังก์ชัน **buzzer()**:
    - **buzzerPinObject.duty_u16(int(65536*0.2))** กำหนดค่าเกรดของเสียงให้เป็นเสียงเต็มกำลัง
    - **buzzerPinObject.freq(frequency)** กำหนดความถี่ของเสียง
    - **sleep(sound_duration)** หน่วงเวลาเล่นเสียงตามระยะเวลาที่กำหนด
    - **buzzerPinObject.duty_u16(int(65536*0))** หยุดเสียงโดยกำหนดเกรดเสียงเป็นศูนย์
    - **sleep(silence_duration)** หน่วงเวลาเงียบหลังจากเล่นเสียงตามระยะเวลาที่กำหนด
5. เรียกใช้ฟังก์ชัน **buzzer()** เพื่อเล่นโน้ตแต่ละโน้ตที่ต้องการ โดยกำหนดความถี่ของเสียงและระยะเวลาการเล่นเสียงและช่วงเวลาเงียบ
6. **BuzzerObj.deinit()** เลิกใช้งานบัซเซอร์โดยปิดอ็อบเจ็กต์ PWM

14. Passive Buzzer Beethoven

เตรียมอุปกรณ์

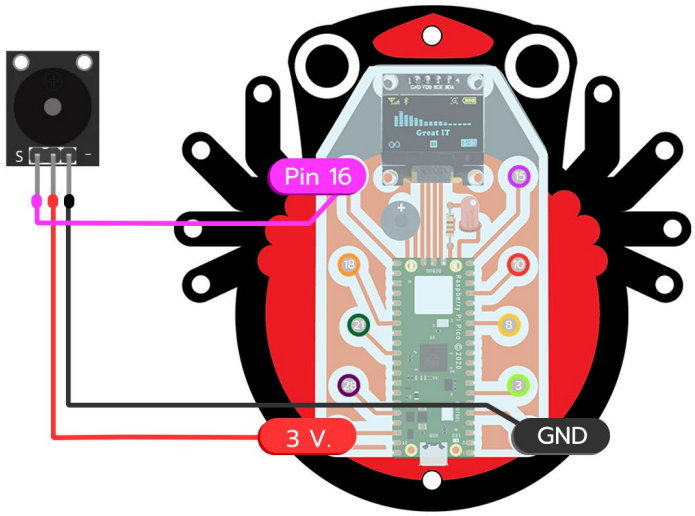
- Spidey
- buzzer passive

การเชื่อมต่อ

-เริ่มต้น ต่อสายขา GND หรือ ขากราวด์ ของ Buzzer กับสไปดี้ ที่ขา ลบ

-ต่อขา บวก ของ เซนเซอร์เข้ากับ สไปดี้ ที่ขา บวก

-ต่อขา S ที่ buzzer ที่ขา 16 ของสไปดี้



การโค้ด

```
from machine import Pin, PWM
from time import sleep

buzzerPIN=16
BuzzerObj = PWM(Pin(buzzerPIN))

def
buzzer(buzzerPinObject,frequency,sound_duration,silence_duration
):
```

```
    # Set duty cycle to a positive value to emit sound from
buzzer
    buzzerPinObject.duty_u16(int(65536*0.1))
    # Set frequency
    buzzerPinObject.freq(frequency)
    # wait for sound duration
    sleep(sound_duration)
    # Set duty cycle to zero to stop sound
    buzzerPinObject.duty_u16(int(65536*0))
    # Wait for sound interrumption, if needed
    sleep(silence_duration)


#set translation table from note to frequency
do5=523
dod5=554
re5=587
red5=622
mi5=659
fa5=698
fad5=739
sol5=784
sold5=830
la5=880
lad5=932
si5=987


buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,red5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,red5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,si5,0.1,0.1)
buzzer(BuzzerObj,re5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,la5,0.5,0.1)

buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,la5,0.1,0.1)
buzzer(BuzzerObj,si5,0.5,0.1)

buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,sold5,0.1,0.1)
buzzer(BuzzerObj,si5,0.1,0.1)
buzzer(BuzzerObj,do5,0.5,0.1)
```

```
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,red5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,red5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,si5,0.1,0.1)
buzzer(BuzzerObj,re5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,la5,0.5,0.1)

buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,la5,0.1,0.1)
buzzer(BuzzerObj,si5,0.5,0.1)

buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,si5,0.1,0.1)
buzzer(BuzzerObj,la5,0.5,0.1)

buzzer(BuzzerObj,si5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,re5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.5,0.1)

buzzer(BuzzerObj,sol5,0.1,0.1)
buzzer(BuzzerObj,fa5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,re5,0.5,0.1)

buzzer(BuzzerObj,fa5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,re5,0.1,0.1)
buzzer(BuzzerObj,do5,0.5,0.1)

buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,re5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,si5,0.5,0.1)

buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,red5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,red5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,si5,0.1,0.1)
```

```
buzzer(BuzzerObj,re5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,la5,0.5,0.1)

buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,la5,0.1,0.1)
buzzer(BuzzerObj,si5,0.5,0.1)

buzzer(BuzzerObj,mi5,0.1,0.1)
buzzer(BuzzerObj,do5,0.1,0.1)
buzzer(BuzzerObj,si5,0.1,0.1)
buzzer(BuzzerObj,la5,0.5,0.1)

#Deactivates the buzzer
BuzzerObj.deinit()
```