

ET-ESP32 LAB V1 (P-ET-A-00725)

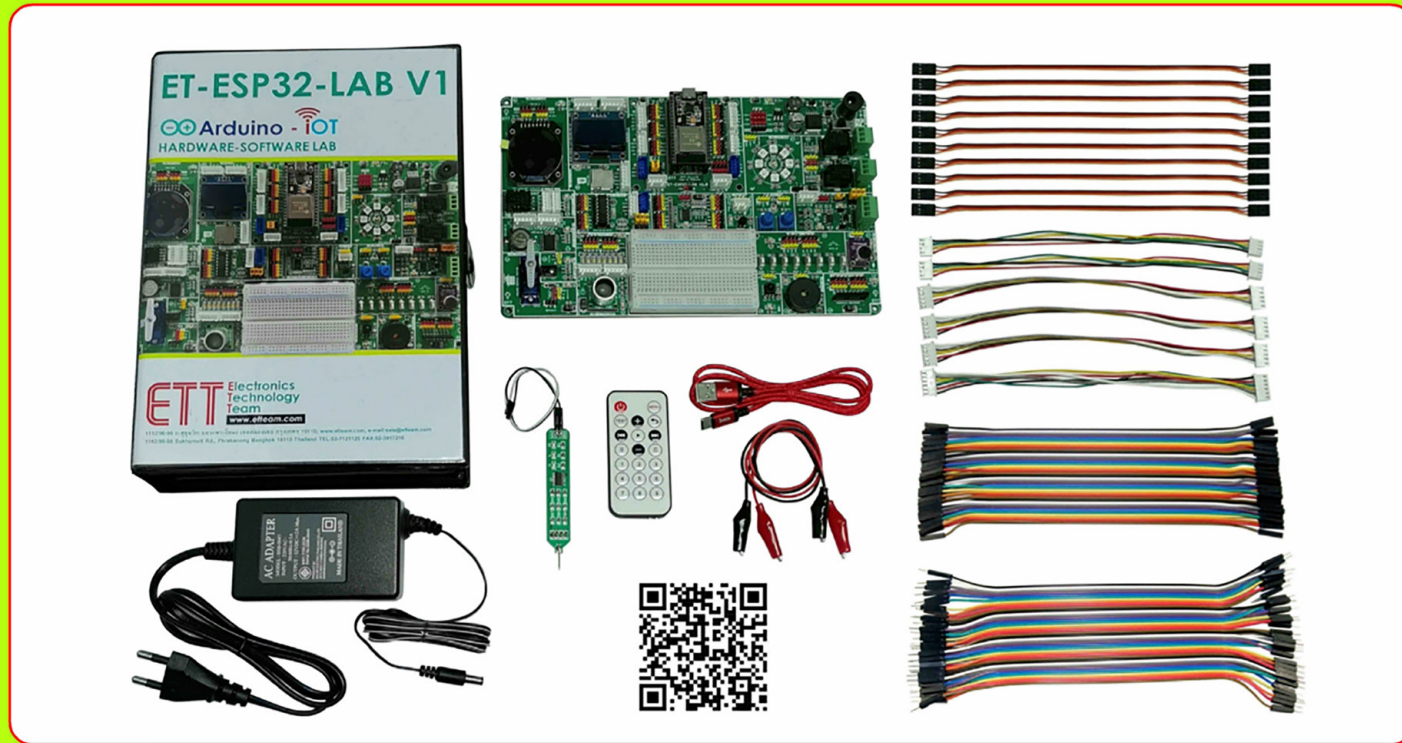
เป็นชุดฝึกที่ออกแบบเพื่อศึกษาเรียนรู้ และเข้าใจการทำงานของ MCU ESP32 ในรูปแบบ ภาษา ซี (C++) ของ Arduino Project ในแบบ OPEN SOURCE และในชุดยังมีอุปกรณ์วงจรต่าง ๆ มากมาย ให้สามารถศึกษาและการทดลอง ระบบทำงาน IOT พร้อมตัวโปรแกรมภาษาซีในการทดลองแสดงวงจร สามารถใช้งานทดลองเรียนรู้ในเรื่องต่าง ๆ ได้มากมาย มีครบจบในชุดเดียว พร้อมกล่องเก็บชุดทดลองไม่ให้ของสูญหาย

- ใช้ MODEL ESP32 รุ่น ESP32-WROOM-32 จากบริษัท ESPRESSIF ที่มี WIFI (802.11 b/g/n/e/i) และ BLUETOOTH 4.2 อยู่พร้อม ใน MODULE, MCU สถาปัตยกรรม Tensilica LX6 DUAL CORE ความเร็วสูงสุด 240 MHz (600 DMIPS) , หน่วยความจำ RAM 520KBYTE, หน่วยความจำแบบ FLASH โปรแกรม 4 MBTE พร้อม PCB ANTENNA

- ใช้ IC USB TO UART เบอร์ CP2102 ของบริษัท SILICON LABS ทำให้ไม่มีปัญหาในการเชื่อมต่อกับ USB PORT บน OS. ต่าง ๆ สามารถ DOWNLOAD โปรแกรมจาก Arduino IDE เข้ามายังบอร์ดได้โดยตรง

- ในการพัฒนาด้วยโปรแกรม Arduino สามารถโปรแกรมได้ทันที ผ่านทาง PORT USB TYPE-C โดยไม่ต้องมีเครื่องโปรแกรมภายนอกใช้งานได้บนระบบปฏิบัติการ Windows XP / 2000 / VISTA / 7 / 8 / 10 / 11/ MAC OSX / LINUX

- ต่อทดลอง WIFI 2.4G ESP32 ในระบบ IOT กับโทรศัพท์มือถือ



บนบอร์ดมี LAB สำหรับเขียนโปรแกรมทดลองอุปกรณ์ต่าง ๆ On Board ดังนี้

- LAB-TFT LCD สี ขนาด 1.28" 240x240 DOT , I2C-3V
- LAB-Micro-SD (SPI-3V)
- LAB-LED RGB จำนวน 8 ตัว (WS2812B)
- LAB-ADC ด้วย VR ปรับค่า 2 วงจร
- LAB-EXPAND IO PORT จำนวน 8 BIT (I2C-3V or 5V)
- LAB-RTC #DS3231 จำนวน 1 วงจร (RTC , # DS3231 , I2C-3V)
- LAB-INPUT SWITCH จำนวน 8 ตัว
- LAB-PIEZO จำนวน 1 วงจร
- LAB-ULTRASONIC จำนวน 1 วงจร วัดระยะทาง (Module I2C-3V)
- WIFI 2.4G ESP32 IOT
- LOGIC-LEVEL 3V to 5V 8BIT
- CONNECT SENSOR EXT. สำหรับต่อบอร์ด I/O อื่น ๆ
- LAB-OLED 1.3" 128X64 DOT , I2C-3V
- LAB-RELAY 5V NO,NC,COM จำนวน 2 วงจร
- LAB-RS485 แบบ AUTO จำนวน 1 วงจร
- LAB-TEMP 1 วงจร (DS18B20 : 1-Wire)
- LAB-T/H ด้วย #DHT20 จำนวน 1 วงจร (Temp & Humidity , I2C-3V)
- LAB-OUTPUT LED จำนวน 8 ตัว
- LAB-ADC MIC. จำนวน 1 วงจร (Module MIC)
- LAB-IR REMOTE ตัวรับ พร้อมตัวส่ง REMOTE IR จำนวน 1 วงจร
- LAB-IR REFLEX จำนวน 1 วงจร (ADC OUT)
- LAB-MICRO SERVO จำนวน 1 วงจร (Micro Servo SG90 , PWM Control)

ปลดล็อกขีดความสามารถของ
ไมโครคอนโทรลเลอร์ ESP32
และก้าวเข้าสู่โลกสมองกลฝังตัว



จัดพิมพ์โดย
สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย
ถนนพญาไท เขตปทุมวัน กรุงเทพฯ 10330
โทร. 0-2218-3269-70
e-mail: cupress@chula.ac.th
www.cupress.chula.ac.th
สรุควินิจฉัยการ สู่สังคม
Knowledge to All



การใช้งานและการทดลอง
ISBN 978-616-636-610-5



ราคา 300.00 บาท



การใช้งานและการทดลอง ESP 32 Arduino

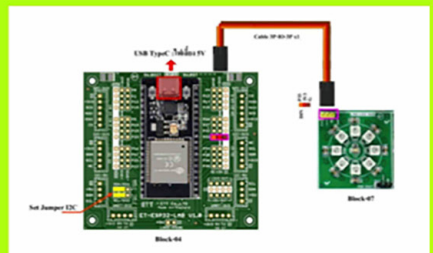
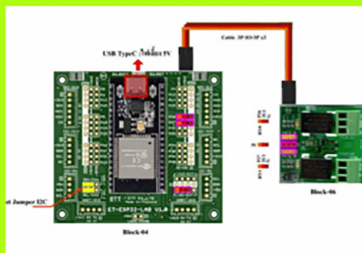
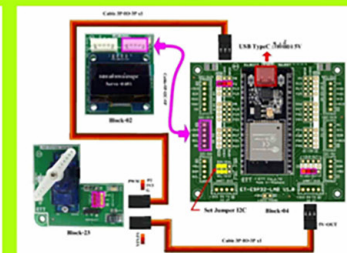
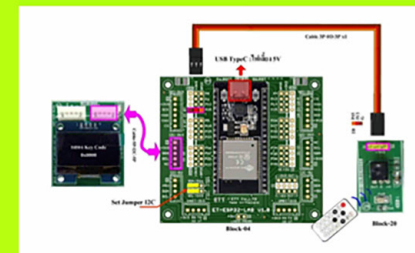
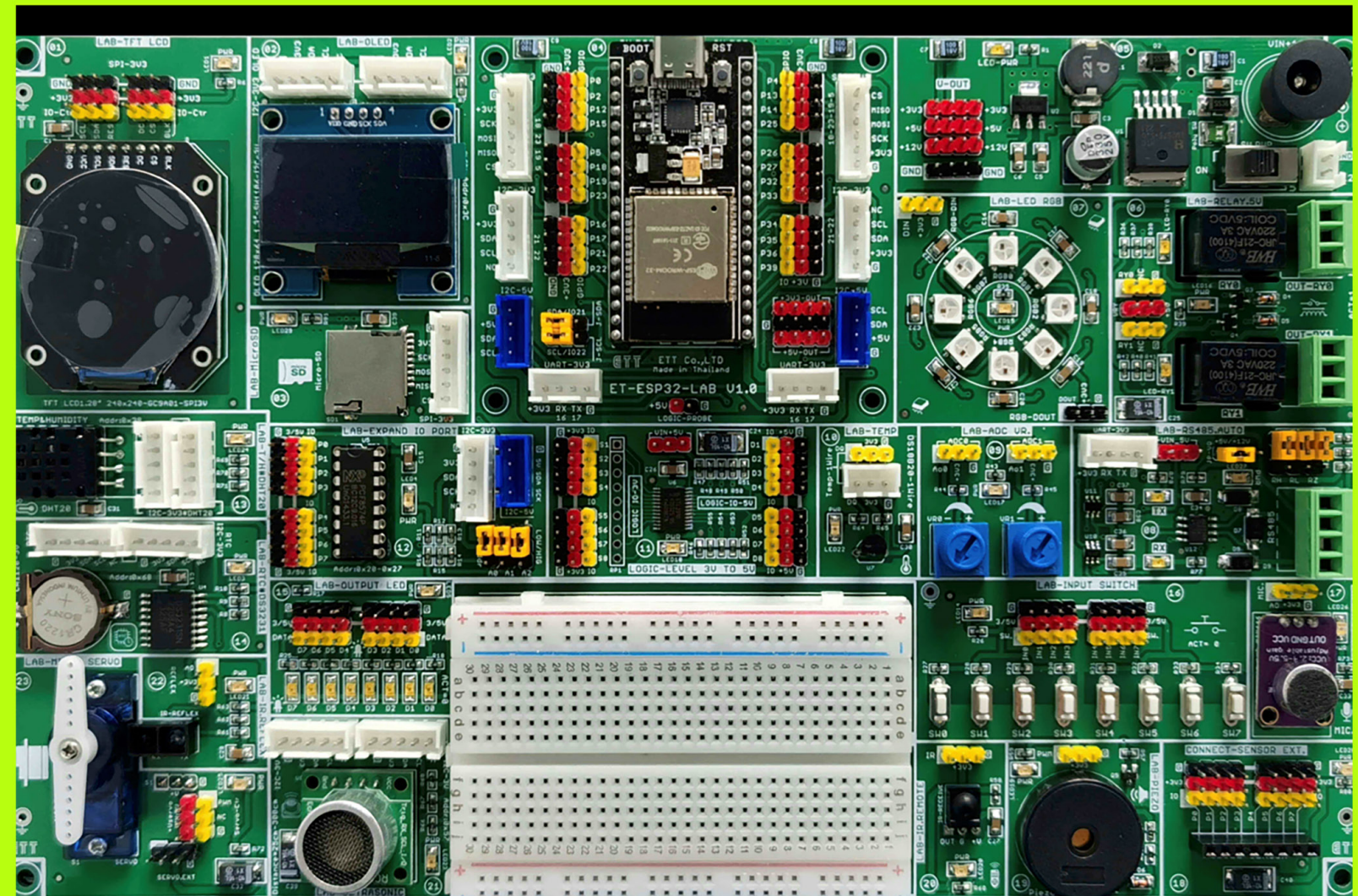
ดนัย เจษฎาฐิติกุล



การใช้งานและการทดลอง ESP32 Arduino

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

๐๐ ดนัย เจษฎาฐิติกุล



- สถาปัตยกรรม ESP32
- ภาพรวมของการทำงาน และชุดคำสั่งที่รองรับการใช้งานกับเฟรมเวิร์คของ Arduino
- GPIO, UART & PWM
- การใช้งานพอร์ตเชื่อมต่อกับอุปกรณ์ภายนอก

- LAB การทดลอง
- OUTPUT LED - LED RGB - RELAY - SERVO - PIEZO - INPUT SW. - IR REMOTE
- ADC/DAC - ADC VR - ADC MIC - IR REFLEX - I²C & SPI BUS - ULTRASONIC
- OLED - RTC - EXPAND I/O PORT - T/H DHT20 - TFT LCD - MICRO SD - TEMP
- WiFi - Blynk

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

การใช้งานและการทดลอง ESP32 Arduino

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

การใช้งานและการทดลอง ESP32 Arduino

दनัย ज्ञेश्वरतिगुल

 **สำนักพิมพ์**
จุฬาลงกรณ์มหาวิทยาลัย
2569

300.-

ชื่อ: ดนัย เจษฎาฐิติกุล, ผู้แต่ง

ชื่อเรื่อง: การใช้งานและการทดลอง ESP32 Arduino / ดนัย เจษฎาฐิติกุล

ข้อมูลการจัดพิมพ์: กรุงเทพฯ: สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย, 2569 | มีดัชนี

หัวเรื่อง: TSHD: อาวุธโน (เครื่องควบคุมแบบโปรแกรม). | เครื่องควบคุมแบบโปรแกรม. |

ไมโครคอนโทรลเลอร์.

เลขหมู่: DDC 006.22--dc23

ISBN 978-616-636-610-5

Q93/12/2569



สรรคณาธิการ สืบค้น
Knowledge to All
www.cupress.chula.ac.th

หนังสือเล่มนี้ผ่านการตรวจคุณภาพวิชาการจากผู้ทรงคุณวุฒิของสำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย

พิมพ์ครั้งที่ 1 จำนวน 20 เล่ม พ.ศ. 2569

จัดทำโดย ดนัย เจษฎาฐิติกุล

สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏเพชรบุรี

บรรณาธิการอำนวยการ : รองศาสตราจารย์ ดร.วิมลวรรณ พิมพ์พันธุ์

กองบรรณาธิการฝ่ายวิชาการ : ศาสตราจารย์กิตติคุณ ดร.ปิยนถ บุนนาค

ศาสตราจารย์ ดร.ปราโมทย์ เดชะอำไพ

ศาสตราจารย์ นายแพทย์ชิษณุ พันธุ์เจริญ

รองศาสตราจารย์ ดร.พิมพ์พันธ์ เดชะคุปต์

ผู้ประสานงาน : เรือนใจ เปรมพรหม

พิสูจน์อักษร : พัชรณธ์ คมมาน

ออกแบบปกและรูปเล่ม : สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย

พิมพ์ที่ สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย [CUP6901-207] โทร. 0-2218-3562-3

www.cupress.chula.ac.th



สงวนลิขสิทธิ์ ห้ามทำซ้ำ



คำนำ

ด้วยเทคโนโลยีปัจจุบันทำให้การผนวกไมโครคอนโทรลเลอร์เข้ากับระบบสื่อสารแบบไร้สายเป็นสิ่งที่ใช้งานได้ง่ายและมีราคาถูก โดยเฉพาะไมโครคอนโทรลเลอร์ตระกูล ESP32 ของบริษัท Espressif Systems ที่ได้ผนวกโมดูลการทำงานทั้งหน่วยประมวลผลแบบ 32 บิต หน่วยความจำภายนอก หน่วยความจำแฟลชรวมภายนอก ระบบสื่อสารไร้สายทั้ง WiFi และ Bluetooth รวมถึงระบบประหยัดพลังงานที่ควบคุมได้ด้วยหน่วยประมวลผลเฉพาะทางต่าง ๆ พร้อมทั้งมีหน่วยแปลงสัญญาณดิจิทัลกับแอนะล็อก ทำให้สามารถนำมาประยุกต์ใช้งานได้หลากหลายทั้งด้านระบบควบคุม สมอกลฝังตัว และ IoT

หนังสือเล่มนี้ได้รวบรวมความรู้เกี่ยวกับไมโครคอนโทรลเลอร์ ESP32 ในส่วนทำงานต่าง ๆ เพื่อเป็นพื้นฐานสำหรับให้ผู้อ่านได้นำไปศึกษาเพิ่มเติม พร้อมทั้งมีตัวอย่างชุดคำสั่งสำหรับเฟรมเวิร์ก Arduino ที่ทางบริษัท Espressif Systems ได้แปลงมาจากการทำงานของชุดพัฒนา ESP-IDF ของตนเพื่อให้ใช้งานกับ Arduino IDE ทั้งนี้ ทางบริษัทที่จัดทำได้ออกแบบชุดทดลองเพื่อให้ผู้อ่านได้ศึกษาการเชื่อมประสานกับอุปกรณ์รอบข้าง ทำให้เมื่อนำเนื้อหาจากหนังสือเล่มนี้ไปประกอบกับชุดปฏิบัติการ ET-ESP32 LAB V1 จะทำให้ผู้อ่านได้เข้าใจหลักการทำงานของวงจร ชุดคำสั่งที่เกี่ยวข้อง และตัวอย่างของโปรแกรมพร้อมกันทั้งด้าน GPIO, PWM, UART, I²C, SPI, 1-Wire, ADC, DAC และ WiFi ทำให้สามารถนำหนังสือเล่มนี้ไปใช้สอนในเนื้อหาเกี่ยวกับวิชาด้านระบบสมอกลฝังตัว ไมโครคอนโทรลเลอร์ ในระดับ ปวช. ปวส. ปริญญาตรี หรือศึกษาด้วยตนเองเพื่อพัฒนาทักษะทางด้านการใช้งานและประยุกต์นำไปใช้งานในสายงานของตนเองต่อไป

สุดท้าย หนังสือเล่มนี้คงสำเร็จเป็นรูปเล่มไม่ได้ถ้าขาดการสนับสนุนจากคุณกอบกิจ เต็มผาติ และ
ผู้ช่วยศาสตราจารย์จารุต์ บุศราทิจ ที่ร่วมกันให้แนวทางและข้อมูลสำหรับการเขียนหนังสือ และครอบครัว
ที่ทำให้กำลังใจตลอดมา ทั้งนี้ ถ้าหนังสือมีข้อผิดพลาดประการใดผู้เขียนขออภัยล่วงหน้าและน้อมรับ
ข้อเสนอแนะ/ข้อเสนอแนะเพื่อนำไปปรับปรุงและให้ดาวน์โหลดส่วนของปรับปรุงต่อไปในอนาคต

दनัย เจษฏาฐิติกุล

1 ตุลาคม 2568



สงวนลิขสิทธิ์ ห้ามทำ



สารบัญ

	หน้า
บทที่ 1 แนะนำ ESP32	1
สถาปัตยกรรม ESP32	2
แรม	10
ภาคจ่ายไฟและการแปลงแรงดัน	15
แฟลชรอม	16
เซ็นเซอร์ภายใน ESP32	22
การขัดจังหวะ	23
ตัวตั้งเวลา	28
WDT	33
โหมดประหยัดพลังงาน	35
การรีเซ็ต	44
สรุป	48
บทที่ 2 GPIO, UART และ PWM	50
แนะนำ GPIO	51
แนะนำ UART	53
แนะนำ PWM	61
สรุป	66
บทที่ 3 LAB-OUTPUT LED	67
แนะนำ LED	68
วงจรและการทำงาน	68
ตัวอย่างโปรแกรม	70
สรุป	74

บทที่ 4 LAB-LED RGB	76
แนะนำ WS2812B	77
ไลบรารี neopixel.....	79
ตัวอย่างโปรแกรม	81
สรุป	82
บทที่ 5 LAB-RELAY.5V	84
แนะนำ โมดูลรีเลย์.....	84
วงจรและการทำงาน	86
ตัวอย่างโปรแกรม	87
สรุป.....	89
บทที่ 6 LAB-MICRO SERVO	90
แนะนำ เซอร์โวมอเตอร์	90
วิธีการคำนวณ	92
วงจรและการทำงาน	93
ไลบรารี ESP32_Servo.....	94
ตัวอย่างโปรแกรม	95
สรุป	98
บทที่ 7 LAB-PIEZO	99
แนะนำ Piezo Buzzer.....	99
วงจรและการทำงาน	103
ไลบรารี toneESP32.....	104
ตัวอย่างโปรแกรม	106
สรุป.....	115
บทที่ 8 LAB-INPUT SWITCH.....	116
แนะนำ LAB-INPUT SWITCH.....	117
วงจรและการทำงาน	117
ตัวอย่างโปรแกรม	120
สรุป	123

บทที่ 9 LAB-IR_REMOTE	125
แนะนำ LAB-IR_REMOTE.....	126
วงจรและการทำงาน	127
ไลบรารี IRremoteESP8266.....	128
ตัวอย่างโปรแกรม	130
สรุป	138
บทที่ 10 ADC/DAC	139
แนะนำ ADC.....	139
แนะนำ DAC.....	142
สรุป	146
บทที่ 11 LAB-ADC VR	148
แนะนำ LAB-ADC VR.....	149
วงจรและการทำงาน	149
ตัวอย่างโปรแกรม	150
สรุป	153
บทที่ 12 LAB-ADC MIC	155
แนะนำ LAB-ADC MIC.....	156
วงจรและการทำงาน	156
ไลบรารี Filter.h.....	157
ไลบรารี FFT	158
ตัวอย่างโปรแกรม	162
สรุป	168
บทที่ 13 LAB-IR.REFLEX	169
แนะนำ LAB-IR.REFLEX.....	169
วงจรและการทำงาน	171
ตัวอย่างโปรแกรม	172
สรุป	174
บทที่ 14 บัส I²C และ SPI	175
แนะนำ I ² C	175
แนะนำ SPI.....	181

สรุป	185
บทที่ 15 LAB-ULTRASONIC	186
แนะนำ เซ็นเซอร์อัลตราโซนิก	186
วิธีการคำนวณหาระยะ	188
วงจรและการทำงาน	188
ไลบรารี RCWL-1X05	190
ตัวอย่างโปรแกรม	192
สรุป	196
บทที่ 16 LAB-OLED	197
แนะนำ OLED	197
วงจรและการทำงาน	198
ไลบรารี Adafruit_GFX.h	199
ไลบรารี Adafruit_SH1106.h	207
ตัวอย่างโปรแกรม	209
สรุป	212
บทที่ 17 LAB-RTC DS3231	213
แนะนำ LAB-RTC DS3231	213
วงจรและการทำงาน	215
ไลบรารี uRTCLib	216
วันและเวลา	219
ตัวอย่างโปรแกรม	220
สรุป	226
บทที่ 18 LAB-EXPAND IO PORT	228
แนะนำ LAB-EXPAND IO PORT	229
วงจรและการทำงาน	229
ไลบรารี PCF8574	232
ตัวอย่างโปรแกรม	233
สรุป	235
บทที่ 19 LAB-T/H DHT20	237
แนะนำ LAB-T/H DHT20	238

วงจรและการทำงาน	238
ไลบรารี DHT20.....	239
ตัวอย่างโปรแกรม	241
สรุป	244
บทที่ 20 LAB-TFT LCD	245
แนะนำ LAB-TFT LCD.....	246
วงจรและการทำงาน	246
ไลบรารี TFT_eSPI.....	247
ตัวอย่างโปรแกรม	256
สรุป	260
บทที่ 21 LAB-MicroSD	261
แนะนำ LAB-MicroSD	261
วงจรและการทำงาน	262
ไลบรารี SD.....	263
ไลบรารี FS	264
ตัวอย่างโปรแกรม	268
สรุป	272
บทที่ 22 LAB-TEMP	273
แนะนำ LAB-TEMP	274
วงจรและการทำงาน	275
ไลบรารี OneWire.h.....	276
ไลบรารี DallasTemperature.h.....	276
ตัวอย่างโปรแกรม	277
สรุป	283
บทที่ 23 WiFi	284
แนะนำ WiFi.....	284
อินเทอร์เน็ตโพรโทคอล.....	285
คำสิ่งที่เกี่ยวกับ WiFi	289
ข้อมูลของการเชื่อมต่อเครือข่าย.....	292
การสแกนจุดเชื่อมต่อ	296

โหมด AP	298
โหมด STA	300
คลาส WiFiServer	301
คลาส WiFiClient	303
ให้บริการเว็บ	306
ตอบสนองจากเว็บ	311
ESP Now	314
สรุป	323
บทที่ 24 Blynk	324
แนะนำ Blynk	324
เริ่มใช้งาน Blynk	325
การสร้างเพิ่มเพลต	327
การเพิ่มอุปกรณ์	329
การสร้างกระแสข้อมูล	330
การสร้างอุปกรณ์บนแอปพลิเคชัน Blynk ในอุปกรณ์พกพา	336
ตัวอย่างโปรแกรม	337
สรุป	346
ดัชนี	347



สารบัญภาพ

	หน้า
ภาพที่ 1-1 ภาควิธีการ ESP32 ของบอร์ด ET-ESP32 LAB V1.0	1
ภาพที่ 1-2 ฟังก์ชันของ ESP32	3
ภาพที่ 1-3 ผลลัพธ์ของตัวอย่างโปรแกรม 1-1	10
ภาพที่ 1-4 ฟังก์ชันความสัมพันธ์ระหว่างหน่วยความจำและหน่วยประมวลผล	11
ภาพที่ 1-5 ผลลัพธ์ของตัวอย่างโปรแกรม 1-2	13
ภาพที่ 1-6 การกำหนดให้ PSRAM ทำงาน	13
ภาพที่ 1-7 ภาควิธีการจ่ายไฟของบอร์ด ET-ESP32 LAB V1.0	16
ภาพที่ 1-8 โมดูล LOGIC-LEVEL 3V TO 5V	16
ภาพที่ 1-9 เมนูเลือกวิธีการบริหารจัดการแฟลชของ ESP32	18
ภาพที่ 1-10 ตัวอย่างผลลัพธ์จากโปรแกรมตัวอย่าง 1-3	21
ภาพที่ 1-11 ตัวอย่างผลลัพธ์จากตัวอย่างโปรแกรม 1-6	28
ภาพที่ 1-12 ตัวอย่างผลลัพธ์จากการทำงานของโปรแกรม 1-7	32
ภาพที่ 1-13 ตัวอย่างผลลัพธ์จากการทำงานของโปรแกรม 1-8	34
ภาพที่ 1-14 ตัวอย่างผลลัพธ์จากโปรแกรม 1-10	39
ภาพที่ 1-15 ตัวอย่างผลลัพธ์จากโปรแกรม 1-11	41
ภาพที่ 1-16 ตัวอย่างผลลัพธ์จากโปรแกรม 1-13	48
ภาพที่ 2-1 ภาควิธีการ ESP32 ของบอร์ด ET-ESP32 LAB V1.0	50
ภาพที่ 2-2 การถ่ายโอนแบบขนานและอนุกรม	53
ภาพที่ 2-3 หน้าจอส่วนของ Serial Monitor ของ Arduino IDE	54
ภาพที่ 2-4 อุปกรณ์แปลง USB/RS232 รุ่น ET-USB RS232 mini	54
ภาพที่ 2-5 ขั้วต่อ UART-3V3 บนบอร์ด ET-ESP32 LAB V1.0	61
ภาพที่ 2-6 กราฟแรงดันเฉลี่ยเมื่อค่ารอบการทำงานเป็นร้อยละ 100	62
ภาพที่ 2-7 กราฟแรงดันเฉลี่ยเมื่อค่ารอบการทำงานเป็นร้อยละ 0	62

ภาพที่ 2-8 กราฟแรงดันเฉลี่ยเมื่อค่ารอบการทำงานเป็นร้อยละ 50	62
ภาพที่ 3-1 ภาพโมดูล LAB-OUTPUT LED	67
ภาพที่ 3-2 (ซ้าย) แสดงหลอดแอลอีดีสีต่าง ๆ (ขวา) สัญลักษณ์ของแอลอีดีและการดูขาของหลอดแอลอีดี	68
ภาพที่ 3-3 วงจรของ LAB-OUTPUT LED.....	69
ภาพที่ 3-4 การสั่งงานวงจร LAB-OUTPUT LED.....	69
ภาพที่ 4-1 ภาพโมดูล LAB-LED RGB.....	76
ภาพที่ 4-2 คาบเวลาของการเก็บลอจิก 0 และ 1 ของ WS2812B.....	77
ภาพที่ 4-3 รูปแบบการเรียงค่าสีสำหรับส่งให้ WS2812B.....	77
ภาพที่ 4-4 วงจรของภาค LAB-LED RGB	78
ภาพที่ 5-1 ภาพโมดูล LAB RELAY.5V.....	85
ภาพที่ 5-2 การเชื่อมต่อของขา NC, COM และ NO ในรีเลย์	85
ภาพที่ 5-3 วงจรของ LAB-RELAY.5V	86
ภาพที่ 6-1 ภาพโมดูลเซอร์โวมอเตอร์ของ ET-ESP32 LAB V1.....	91
ภาพที่ 6-2 ความสัมพันธ์ของคิวต์กับการหมุนของเซอร์โวมอเตอร์.....	91
ภาพที่ 6-3 วงจรของ LAB-MICRO SERVO	93
ภาพที่ 6-4 ไบบรารี ESP32Servo.....	94
ภาพที่ 7-1 ภาพโมดูล LAB-PIEZO ของ ET-ESP32 LAB V1.....	100
ภาพที่ 7-2 (ซ้าย) ความถี่ 1Hz (ขวา) ความถี่ 1Hz แบบดิจิทัล	100
ภาพที่ 7-3 คาบเวลาของคลื่น 1 ลูก.....	101
ภาพที่ 7-4 วงจร LAB-PIEZO	103
ภาพที่ 7-5 ไบบรารี toneESP32	104
ภาพที่ 8-1 สวิตช์แบบกดติดปล่อยดับ ดิปสวิตช์ และคีย์แพด.....	116
ภาพที่ 8-2 ภาพโมดูล LAB-INPUT SWITCH.....	117
ภาพที่ 8-3 วงจรของ LAB-INPUT SWITCH	118
ภาพที่ 8-4 สัญญาณกระแสไฟฟ้าของแรงดัน (ซ้าย) กดแล้วปล่อย (ขวา) กดค้าง	118
ภาพที่ 8-5 วงจรสำหรับสวิตช์กดติดปล่อยดับที่มีการป้องกันการกระเพื่อมของสัญญาณ.....	119
ภาพที่ 9-1 ภาพโมดูล LAB-IR_REMOTE.....	125
ภาพที่ 9-2 ภาพ Remote Control รุ่น ET-IR REMOTE KEY	126
ภาพที่ 9-3 วงจรของ LAB-IR_REMOTE	127

ภาพที่ 9-4	ไลบรารี IRemoteESP8266	129
ภาพที่ 10-1	ขั้นตอนการแปลงสัญญาณแอนะล็อกเป็นดิจิทัล	140
ภาพที่ 10-2	ตัวอย่างกราฟฟันปลา.....	143
ภาพที่ 10-3	ตัวอย่างกราฟสามเหลี่ยม	144
ภาพที่ 10-4	ตัวอย่างกราฟคลื่นไซน์	146
ภาพที่ 11-1	ภาพโมดูล LAB-ADC VR.....	148
ภาพที่ 11-2	วงจรของ LAB-AD VR	149
ภาพที่ 12-1	ภาพโมดูล LAB-ADC MIC.....	155
ภาพที่ 12-2	วงจรของ LAB-AD MIC	157
ภาพที่ 12-3	ไลบรารี arduinoFFT	159
ภาพที่ 12-4	ตัวอย่างผลลัพธ์จากการใช้ arduinoFFT แต่ละคำสั่ง	162
ภาพที่ 13-1	โมดูล LAB-IR.REFLEX.....	170
ภาพที่ 13-2	การสะท้อนของแสงอินฟราเรด	170
ภาพที่ 13-3	วงจรของ LAB-IR.REFLEX.....	171
ภาพที่ 14-1	การต่อเชื่อมไอซีกับบัส I ² C	176
ภาพที่ 14-2	การต่อเชื่อมไอซีกับบัส I ² C	176
ภาพที่ 14-3	ขั้วต่อ I ² C แบบ 3v3 และ 5V ของบอร์ด ET-ESP32 LAB V1.0.....	176
ภาพที่ 14-4	ขั้วต่อ J-SCL/J-SDA ของบอร์ด ET-ESP32 LAB V1.0	177
ภาพที่ 14-5	ขั้วต่อ SPI ของบอร์ด ET-ESP32 LAB V1.0	182
ภาพที่ 15-1	ภาพโมดูลอัลตราโซนิกของ ET-ESP32 LAB V1.....	187
ภาพที่ 15-2	หลักการทำงานของเซ็นเซอร์อัลตราโซนิก.....	187
ภาพที่ 15-3	สัญญาณดิจิทัลที่ขาทรiggerและเอคโคของโมดูลอัลตราโซนิก	188
ภาพที่ 15-4	วงจรของ LAB-ULTRASONIC.....	189
ภาพที่ 15-5	ไลบรารี RCWL-1X05.....	189
ภาพที่ 16-1	โมดูล LAB-OLED	198
ภาพที่ 16-2	วงจรของ LAB-OLED.....	199
ภาพที่ 16-3	ไลบรารี Adafruit_GFX	200
ภาพที่ 16-4	ไลบรารี Adafruit_SH110X.....	208
ภาพที่ 17-1	โมดูล LAB-RTC DS3231	214
ภาพที่ 17-2	ผังบล็อกทำงานของ DS3231.....	214

ภาพที่ 17-3	วงจรของ LAB-RTC DS3231	216
ภาพที่ 17-4	ไลบรารี uRTCLib.....	217
ภาพที่ 18-1	โมดูล LAB-EXPAND IO PORT	228
ภาพที่ 18-2	ผังการทำงานของ PCF8574P	230
ภาพที่ 18-3	วงจรของขา P0 ถึง P7 ของ PCF8574.....	230
ภาพที่ 18-4	วงจรของ LAB-EXPAND IO PORT.....	231
ภาพที่ 19-1	โมดูล LAB-T/H DHT20	237
ภาพที่ 19-2	วงจรของ LAB-T/H DHT20	239
ภาพที่ 20-1	โมดูล LAB-TFT LCD.....	245
ภาพที่ 20-2	รูปแบบสี RGB565.....	246
ภาพที่ 20-3	วงจรของ LAB-TFT LCD.....	247
ภาพที่ 20-4	ไลบรารี TFT_eSPI.....	248
ภาพที่ 20-5	ไฟล์เดอร์ที่เก็บไฟล์ User_Setup_Select.h	248
ภาพที่ 20-6	ค่ามุมของ setRotation().....	250
ภาพที่ 20-7	ค่าของตำแหน่ง DATUM	256
ภาพที่ 21-1	โมดูล LAB-MicroSD	262
ภาพที่ 21-2	ภาพด้านหน้าและหลังของการ์ด SD และ microSD	262
ภาพที่ 21-3	วงจรของ LAB-MicroSD.....	263
ภาพที่ 21-4	ตัวอย่างผลลัพธ์การทำงานของตัวอย่างโปรแกรม 21-1	269
ภาพที่ 21-5	ตัวอย่างผลลัพธ์การทำงานของตัวอย่างโปรแกรม 21-2	271
ภาพที่ 22-1	ภาพโมดูล LAB-TEMP	273
ภาพที่ 22-2	วงจรของ LAB-TEMP	275
ภาพที่ 23-1	ภาพเปรียบเทียบโมเดลแบบ OSI กับ TCP/IP.....	287
ภาพที่ 23-2	การสื่อสารแบบทีซีพี.....	288
ภาพที่ 23-3	การสื่อสารแบบยูดีพี	289
ภาพที่ 23-4	ซ็อกเก็ต	289
ภาพที่ 23-5	ตัวอย่างผลลัพธ์จากโปรแกรม 23-1	295
ภาพที่ 23-6	ตัวอย่างผลลัพธ์เมื่อสแกนหาจุดเชื่อมต่อเครือข่ายอินเทอร์เน็ต.....	300
ภาพที่ 23-7	ตัวอย่างผลลัพธ์จากตัวอย่างโปรแกรม 23-4	301
ภาพที่ 23-8	ตัวอย่างผลลัพธ์จากตัวอย่างโปรแกรม 23-5	306

ภาพที่ 23-9 การสื่อสารระหว่างเครื่องให้บริการและเว็บเบราว์เซอร์.....	307
ภาพที่ 23-10 ตัวอย่างผลลัพธ์บนเว็บเบราว์เซอร์จากตัวอย่างโปรแกรม 23-6.....	311
ภาพที่ 24-1 เว็บไซต์ blynk.io	324
ภาพที่ 24-2 Dashboard ของ Blynk website.....	325
ภาพที่ 24-3 หน้าสมัครบัญชี Blynk.....	326
ภาพที่ 24-4 หน้าเข้าสู่ระบบ Blynk	326
ภาพที่ 24-5 การสร้างเพิ่มเพลต	327
ภาพที่ 24-6 การตั้งค่ารายละเอียดเริ่มต้นของเพิ่มเพลต	328
ภาพที่ 24-7 การแก้ไขเพิ่มเพลต	328
ภาพที่ 24-8 การเพิ่มอุปกรณ์ที่จากเพิ่มเพลต	329
ภาพที่ 24-9 ข้อมูลอุปกรณ์ที่ใช้สำหรับการเชื่อมต่อ ESP32 และ Blynk	329
ภาพที่ 24-10 ขั้นตอนการตั้งค่าเพื่อใช้งานเพิ่มเพลต	330
ภาพที่ 24-11 การสร้างกระแสข้อมูล	330
ภาพที่ 24-12 การตั้งค่ากระแสข้อมูลสำหรับ LED	331
ภาพที่ 24-13 การตั้งค่ากระแสข้อมูลสำหรับ Switch	332
ภาพที่ 24-14 ขั้นตอนการตั้งค่าแผงควบคุม	333
ภาพที่ 24-15 การเลือกใช้งานอุปกรณ์เซ็นเซอร์.....	334
ภาพที่ 24-16 การกำหนดกระแสข้อมูลให้กับ LED	334
ภาพที่ 24-17 การกำหนดกระแสข้อมูลให้กับ Switch	335
ภาพที่ 24-18 การบันทึกการตั้งค่าเพิ่มเพลต	335
ภาพที่ 24-19 หน้าแสดงอุปกรณ์จากเพิ่มเพลตบนอุปกรณ์พกพา	336
ภาพที่ 24-20 แผงควบคุมอุปกรณ์	337
ภาพที่ 24-21 อุปกรณ์เซ็นเซอร์บางส่วนที่สามารถเลือกใช้งานได้	337
ภาพที่ 24-22 ข้อมูลโปรแกรม Arduino IDE	338
ภาพที่ 24-23 การจัดการไลบรารี.....	339
ภาพที่ 24-24 ติดตั้งไลบรารี Blynk.....	339
ภาพที่ 24-25 ติดตั้งไลบรารีเพิ่มเติมที่จำเป็น	340
ภาพที่ 24-26 อัปโหลดโค้ดสำเร็จ.....	341
ภาพที่ 24-27 สถานะออนไลน์ของอุปกรณ์	342
ภาพที่ 24-28 การเลือกอุปกรณ์บน Blynk ที่ต้องการสั่งการ	344

ภาพที่ 24-29 แผงควบคุมอุปกรณ์..... 344

ภาพที่ 24-30 LED ดับเมื่อสวิตช์ที่แผงควบคุมอุปกรณ์บน Blynk ปิดอยู่..... 345

ภาพที่ 24-31 LED ติดเมื่อสวิตช์ที่แผงควบคุมอุปกรณ์บน Blynk เปิดอยู่..... 345



สารบัญตาราง

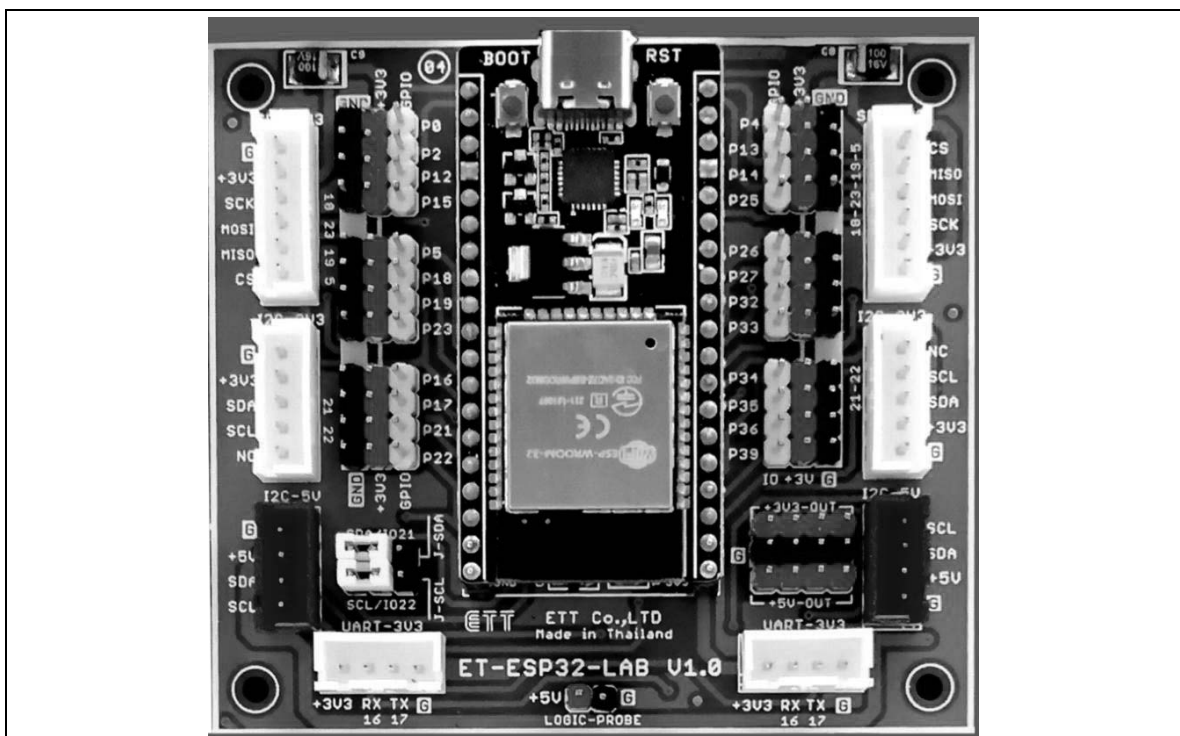
	หน้า
ตารางที่ 1-1 รายการอุปกรณ์ต่อพ่วงของ ESP32 และตำแหน่งหน่วยความจำ	4
ตารางที่ 1-2 การขัดจังหวะ	24
ตารางที่ 1-3 การทำงานในโหมดใช้พลังงานแบบต่าง ๆ ของ ESP32	35
ตารางที่ 7-1 ความถี่ของตัวโน้ตดนตรี	102
ตารางที่ 14-1 การกำหนดโหมดของ SPI.....	184
ตารางที่ 18-1 การกำหนดค่าตำแหน่งของ LAB-EXPAND IO PORT	231
ตารางที่ 24-1 หัวข้อและคำอธิบายในการตั้งค่า	332

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



บทที่ 1 สถาปัตยกรรม ESP32

บทนี้กล่าวถึงสถาปัตยกรรมของไมโครคอนโทรลเลอร์ ESP32 เพื่ออธิบายภาพรวมของการทำงาน และชุดคำสั่งที่รองรับการใช้งานกับเฟรมเวิร์คของ Arduino อันทำให้มองเห็นความสัมพันธ์ของการทำงาน ร่วมกันของหน่วยต่าง ๆ ของ ESP32 ตามผังโมดูล ESP32 ในภาพที่ 1-2 พร้อมทั้งแนะนำส่วนเสริมบน บอร์ด ET-ESP32 LAB v1.0 ที่จัดเตรียมไว้ให้อ่านใช้งาน ดังภาพที่ 1-1 เพื่อให้ใช้งานไมโครคอนโทรลเลอร์ ESP32 ได้สะดวกและเต็มประสิทธิภาพ โดยเนื้อหาเกี่ยวกับ GPIO ถูกแยกไปอธิบายในบทที่ 2, โมดูลแปลงสัญญาณระหว่างดิจิทัลกับแอนะล็อกในบทที่ 10, การเชื่อมต่อกับบัส I²C/SPI ในบทที่ 14, การเชื่อมต่อระบบสื่อสารไร้สายในบทที่ 23 และการใช้งานระบบ IoT ผ่าน Blynk ได้อธิบายขั้นตอนพร้อม ตัวอย่างในบทที่ 24



ภาพที่ 1-1

ภาค ESP32 ของบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ

จากภาพที่ 1-1 แสดงให้เห็นว่าบอร์ด ET-ESP32 LAB V1.0 ภาคไมโครคอนโทรลเลอร์ประกอบด้วยโมดูล Dev-Kit ของ ESP32 ที่มีพอร์ตเชื่อมต่อกับคอมพิวเตอร์ผ่านทางพอร์ตสื่อสารแบบ USB-C และมีการเสียบประกอบบนบอร์ดขยายพอร์ตทำให้สะดวกในการใช้งานทั้งในส่วนของการเชื่อมต่อแต่ละ GPIO ที่มาพร้อมกับหัวต่อของ VCC แรงดันไฟฟ้ากระแสตรง 3.3V (3V3) และขา GND หรือการเชื่อมต่อทั้งพอร์ตสื่อสารอนุกรม UART2 หรือบัสสื่อสารอนุกรมแบบ I²C ทั้งแรงดัน 3.3V และ 5V หรือบัสสื่อสารแบบ SPI พร้อมทั้งหัว +5V/GND สำหรับเชื่อมต่อกับอุปกรณ์เสริม Logic-Probe ทำให้ใช้ตรวจสอบสถานะของขานำออกสัญญาณได้ง่ายขึ้นโดยไม่จำเป็นต้องต่อเชื่อมกับวงจรแอลอีดีภายนอกเพื่อทดสอบการส่งสัญญาณ เป็นต้น

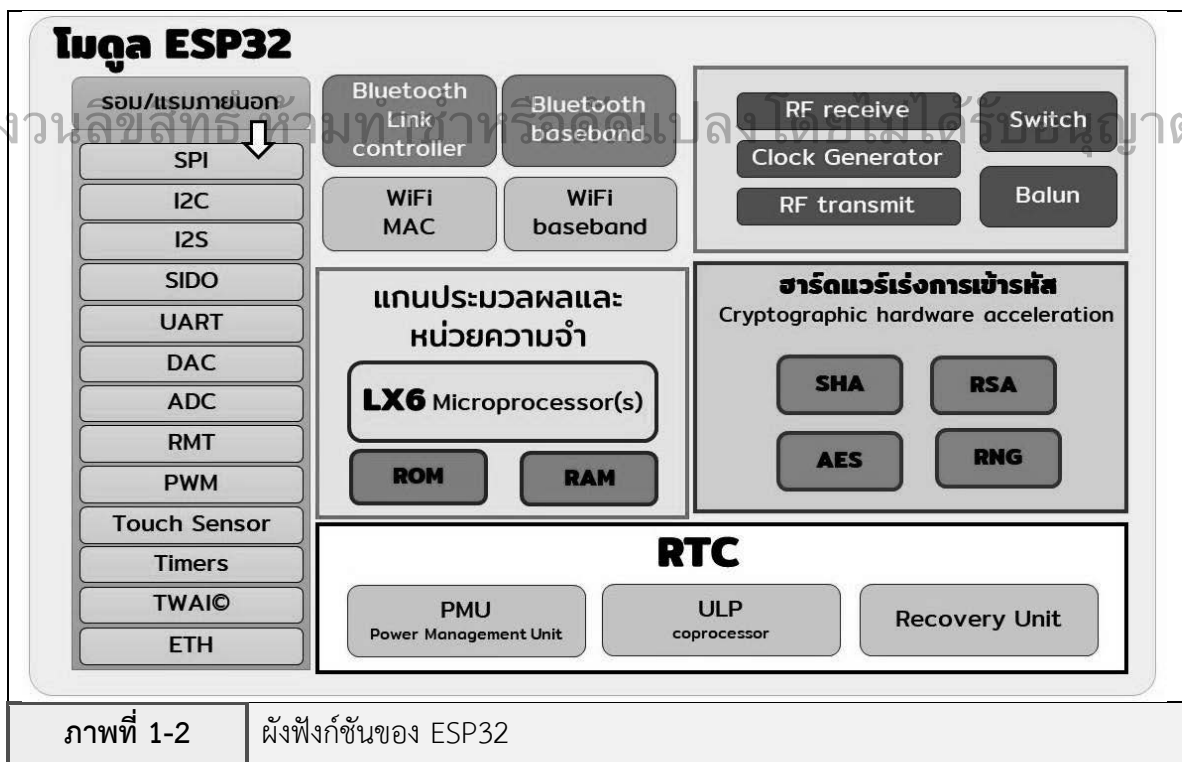
> สถาปัตยกรรม

จากความสำเร็จในการพัฒนาและจำหน่ายโมดูล ESP8266 ซึ่งได้รับความนิยมอย่างแพร่หลายในกลุ่มนักพัฒนาและถูกนำไปประยุกต์ใช้งานในหลากหลายด้าน บริษัท Espressif Systems ได้พัฒนาและนำเสนอไมโครคอนโทรลเลอร์ตระกูล ESP32 ออกสู่ตลาดในหลายรุ่น ได้แก่ ESP32, ESP32-S2, ESP32-C3, ESP32-S3 และ ESP32-S5 โดยไมโครคอนโทรลเลอร์ตระกูลดังกล่าวเป็นการพัฒนาต่อยอดจากสถาปัตยกรรมของ ESP8266 ผ่านการปรับปรุงหน่วยประมวลผลให้มีประสิทธิภาพสูงขึ้นและรองรับการใช้งานที่หลากหลายยิ่งขึ้น

ไมโครคอนโทรลเลอร์ ESP32 ถูกจัดอยู่ในประเภท System on Chips (SoCs) ซึ่งขับเคลื่อนด้วยหน่วยประมวลผลขนาด 32 บิต โดยมีโครงสร้างการทำงานสอดคล้องกับผังฟังก์ชันดังแสดงในภาพที่ 1-2 ภายในติดตั้งหน่วยประมวลผลจำนวน 2 แกน ได้แก่ PRO_CPU และ APP_CPU ซึ่งพัฒนาบนพื้นฐานเทคโนโลยีของบริษัท Tensilica รุ่น Xtensa LX6

การประมวลผลใช้สถาปัตยกรรมฮาร์วาร์ด (Harvard Architecture) ซึ่งแยกโครงสร้างบัสข้อมูลและชุดคำสั่งออกจากกันอย่างอิสระ ส่งผลให้มีหน่วยความจำแยกสำหรับการจัดเก็บข้อมูลและคำสั่งระบบรองรับการขยายหน่วยความจำภายนอกผ่านบัส SPI ที่ออกแบบมาโดยเฉพาะ โดยสามารถเชื่อมต่อหน่วยความจำ RAM ได้สูงสุด 8MB และ ROM ได้สูงสุด 32MB โดยมีการบริหารจัดการใช้หน่วยความจำร่วมกันระหว่างแกนประมวลผลทั้งสอง

สำหรับทรัพยากรภายในชิปประกอบด้วยหน่วยความจำแอสแรม (SRAM) ขนาด 520kB และแรมขนาด 448kB พร้อมช่องสัญญาณเชื่อมต่ออุปกรณ์ภายนอกจำนวน 42 บิต นอกจากนี้ อุปกรณ์ยังถูกออกแบบให้รองรับการใช้งานในสภาพแวดล้อมระดับอุตสาหกรรม โดยสามารถทำงานได้ภายใต้อุณหภูมิระหว่าง -40°C ถึง +125°C



ภาพโดย จารุต บุศราทิจ

โปรแกรมที่พัฒนาสามารถเข้าถึงอุปกรณ์ภาค UART0, UART1, UART2, SPI1, SPI2, SPI3, I2S0, I2S1, SDMMC, EMAC, BT (Bluetooth) และ WiFi ได้โดยตรงจากหน่วยความจำขนาด 328kB ผ่านทางการเข้าถึงโดยตรง หรือ DMA (Direct Memory Access) ยกเว้นอุปกรณ์ SDIO, TWAI®, MotorPWM และ Pulse Counter ที่ไม่สามารถใช้งานได้โดยตรงจากเฟรมเวิร์คของ Arduino เนื่องจากไม่รองรับการทำงานเข้าถึง

แกนประมวลผลของ ESP32

ระบบของ ESP32 ประกอบด้วย 2 ส่วนหลัก คือ Core และ RTC โดยส่วนของ Core หรือแกน จะประกอบไปด้วยการทำงานของโมดูลต่าง ๆ ต่อไปนี้

- CPU ถ้ามีแกนเดียวจะมีเฉพาะ PRO_CPU หากมี 2 แกนจะประกอบด้วย PRO_CPU และ APP_CPU
- Peripheral สำหรับอุปกรณ์ต่อพ่วงของ ESP32 มี 42 รายการ ดังตารางที่ 1-1 โดยอุปกรณ์ต่อพ่วงเหล่านี้สามารถเข้าถึงได้โดยตรงจากแกนของหน่วยประมวลผลแกนใดแกนหนึ่งในแต่ละระยะเวลาหนึ่ง หากมีแกนใดใช้งานอุปกรณ์จะทำให้อีกแกนใช้อุปกรณ์

เดียวกันในเวลาเดียวกันไม่ได้ ยกเว้น PID Controller เท่านั้นที่ยอมให้เข้าถึงได้พร้อมกัน กรณีที่ต้องการสื่อสารกับอุปกรณ์เสียงแบบอนุกรมจะมี I²S จำนวน 2 หน่วยที่ ESP32 เตรียมไว้ จึงสามารถใช้งานเสียงแบบ 2 ช่องสัญญาณ หรือแบบสเตอริโอได้

- Wi-Fi มีความถี่สัญญาณนาฬิกาที่ 2.4GHz โดยรองรับมาตรฐาน IEEE 802.11 b/g/n ความเร็วสูงสุดในการสื่อสารคือ 150Mbps รองรับการยืนยันตัวตน (Authentication) ในแบบ WEP และ WPA/WPA2 สามารถใช้เป็นแบบเครือข่ายแบบเปิดได้ นอกจากนี้รองรับการเข้ารหัสทั้งแบบ WEP, TKIP และ AES บนโพรโทคอลเครือข่าย IPv4 บนบริการประเภท TCP, UDP, HTTP และ FTP
- Bluetooth เป็นโมดูลการสื่อสารแบบไร้สายภายใต้การสื่อสาร Bluetooth รุ่น 4.2 และรองรับ BLE ทำให้ประหยัดพลังงาน
- Digital GPIO สำหรับเชื่อมประสานกับวงจรภายนอกที่แรงดันกระแสตรง 3.3V จำนวน 34 ขา
- มี ADC จำนวน 18 ช่องสัญญาณ รองรับความละเอียดสูงสุด 12 บิต
- DAC ขนาด 8 บิตจำนวน 2 ช่องสัญญาณ
- ความเร็วสัญญาณนาฬิกาสูงสุดที่ 240MHz
- การทำงานของระบบปฏิบัติการของ ESP32 พัฒนามาจาก FreeRTOS จึงเหมาะกับการประมวลผลแบบทันเวลา (Real-time) โดยปกติแกน PRO_CPU หรือ core 0 ถูกใช้เป็นตัวทำงานของระบบปฏิบัติการ และ APP_CPU หรือ core 1 ถูกใช้เป็นตัวทำงานของโปรแกรมที่พัฒนาจาก Arduino

ตารางที่ 1-1 รายการอุปกรณ์ต่อพ่วงของ ESP32 และตำแหน่งหน่วยความจำ

ลำดับ	ตำแหน่งหน่วยความจำ		ขนาด	อุปกรณ์ต่อพ่วง
	Low Address	High Address		
1	0x3FF0_0000	0x3FF0_0FFF	4 kB	DPort Register
2	0x3FF0_1000	0x3FF0_1FFF	4 kB	AES Accelerator
3	0x3FF0_2000	0x3FF0_2FFF	4 kB	RSA Accelerator
4	0x3FF0_3000	0x3FF0_3FFF	4 kB	SHA Accelerator
5	0x3FF0_4000	0x3FF0_4FFF	4 kB	Secure Boot
6	0x3FF1_0000	0x3FF1_3FFF	16kB	Cache MMU Table
7	0x3FF1_F000	0x3FF1_FFFF	4kB	PID Controller
8	0x3FF4_0000	0x3FF4_0FFF	4kB	UART0
9	0x3FF4_2000	0x3FF4_2FFF	4kB	SPI1
10	0x3FF4_3000	0x3FF4_3FFF	4kB	SPI0
11	0x3FF4_4000	0x3FF4_4FFF	4kB	GPIO

ตารางที่ 1-1 รายการอุปกรณ์ต่อพ่วงของ ESP32 และตำแหน่งหน่วยความจำ (ต่อ)

ลำดับ	ตำแหน่งหน่วยความจำ		ขนาด	อุปกรณ์ต่อพ่วง
	Low Address	High Address		
12	0x3FF4_8000	0x3FF4_8FFF	4kB	RTC
13	0x3FF4_9000	0x3FF4_9FFF	4kB	IO MUX
14	0x3FF4_B000	0x3FF4_BFFF	4kB	SDIO Slave (1/3)
15	0x3FF4_C000	0x3FF4_CFFF	4kB	UDMA1
16	0x3FF4_F000	0x3FF4_FFFF	4kB	I2S0
17	0x3FF5_0000	0x3FF5_0FFF	4kB	UART1
18	0x3FF5_3000	0x3FF5_3FFF	4kB	I2C0
19	0x3FF5_4000	0x3FF5_4FFF	4kB	UDMA0
20	0x3FF5_5000	0x3FF5_5FFF	4kB	SDIO Slave (2/3)
21	0x3FF5_6000	0x3FF5_6FFF	4kB	RMT
22	0x3FF5_7000	0x3FF5_7FFF	4kB	PCNT
23	0x3FF5_8000	0x3FF5_8FFF	4kB	SDIO Slave (3/3)
24	0x3FF5_9000	0x3FF5_9FFF	4kB	LED PWM
25	0x3FF5_A000	0x3FF5_AFFF	4kB	eFuse Controller
26	0x3FF5_B000	0x3FF5_BFFF	4kB	Flash Encryption
27	0x3FF5_E000	0x3FF5_EFFF	4kB	MCPWM0
28	0x3FF5_F000	0x3FF5_FFFF	4kB	TIMG0
29	0x3FF6_0000	0x3FF6_0FFF	4kB	TIMG1
30	0x3FF6_4000	0x3FF6_4FFF	4kB	SPI2
31	0x3FF6_5000	0x3FF6_5FFF	4kB	SPI3
32	0x3FF6_6000	0x3FF6_6FFF	4kB	SYSCON
33	0x3FF6_7000	0x3FF6_7FFF	4kB	I2C1
34	0x3FF6_8000	0x3FF6_8FFF	4kB	SDMMC
35	0x3FF6_9000	0x3FF6_AFFF	8kB	EMAC
36	0x3FF6_B000	0x3FF6_BFFF	4kB	TWAI (CAN Bus)
37	0x3FF6_C000	0x3FF6_CFFF	4kB	MCPWM1
38	0x3FF6_D000	0x3FF6_DFFF	4kB	I2S1
39	0x3FF6_E000	0x3FF6_EFFF	4kB	UART2
40	0x3FF6_F000	0x3FF6_FFFF	4kB	MCPWM2

ตารางที่ 1-1 รายการอุปกรณ์ต่อพ่วงของ ESP32 และตำแหน่งหน่วยความจำ (ต่อ)

ลำดับ	ตำแหน่งหน่วยความจำ		ขนาด	อุปกรณ์ต่อพ่วง
	Low Address	High Address		
41	0x3FF7_0000	0x3FF7_0FFF	4kB	MCPWM3
42	0x3FF7_5000	0x3FF7_5FFF	4kB	RNG (Random Generator)

ที่มา: ดัดแปลงจาก Espressif Systems. (2024). *ESP32 Technical Reference Manual*. น. 72-73. สืบค้นเมื่อ มีนาคม 18, 2567, จาก

https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf

ตำแหน่งของหน่วยความจำ

- 0x3FFA_0000 ถึง 0x3FFD_FFFF เป็นหน่วยความจำสำหรับทำ DMA อันเป็นการใช้บัสเดียวกับบัสข้อมูลของ ESP32 เพื่อทำการอ่านหรือเขียนหน่วยความจำภายใน SRAM2
- 0x3FFE_0000 ถึง 0x3FFF_FFFF เป็นหน่วยความจำสำหรับทำ DMA อันเป็นการใช้บัสเดียวกับบัสข้อมูลของ ESP32 เพื่อทำการอ่านหรือเขียนหน่วยความจำภายใน SRAM1
- 0x4000_0000 ถึง 0x4FFF_FFFF ใช้สำหรับอ้างอิงหน่วยความจำที่เก็บชุดคำสั่ง
- 0x5000_0000 เป็นต้นไป ใช้สำหรับเชื่อมต่อกับหน่วยความจำที่เก็บชุดคำสั่งหรือข้อมูล
- 0x3FF8_0000 ถึง 0x3FF8_1FFF เป็นตำแหน่งของหน่วยความจำแรมบนบัสข้อมูลสำหรับเก็บค่าของ RTC FAST
- 0x400C_0000 ถึง 0x400C_1FFF เป็นตำแหน่งของหน่วยความจำแรมบนบัสคำสั่งสำหรับเก็บค่าของ RTC FAST
- 0x5000_0000 ถึง 0x5000_1FFF เป็นหน่วยความจำแรมสำหรับเก็บค่าของ RTC SLOW

ความเร็วการทำงานของหน่วยความจำแรมและรอมภายในโมดูล ESP32 ขึ้นอยู่กับประเภทของสัญญาณนาฬิกาที่เชื่อมต่อเข้ากับโมดูลดังนี้

- รอมและแรม (SRAM) ใช้สัญญาณนาฬิกาจาก CPU_CLK และหน่วยประมวลผลสามารถเข้าถึงได้ด้วยความเร็วเพียง 1 สัญญาณนาฬิกา
- DMA ใช้สัญญาณนาฬิกาจาก APB_CLOCK

แกน RTC

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

RTC ประกอบด้วย 4 หน่วยทำงาน ได้แก่ PMU (Power Management Unit) สำหรับบริหารจัดการพลังงาน, เซ็นเซอร์ (sensor) สำหรับเชื่อมประสานกับเซ็นเซอร์, ULP (Ultra Low Power) สำหรับเป็นหน่วยประมวลผลร่วมที่ทำงานเมื่อ ESP32 อยู่ในโหมดประหยัดพลังงาน และ RTC GPIO สำหรับเชื่อมประสานกับ GPIO พร้อมทั้งมีหน่วยความจำเป็นของตัวเองแยกจากแกนประมวลผลของ ESP32 โดยหน่วยความจำความเร็วสูงเรียกว่า RTC_FAST ใช้สัญญาณนาฬิกาจาก APB_CLOCK และหน่วยความจำที่มีความเร็วต่ำกว่าเรียกว่า RTC_SLOW ที่ใช้สัญญาณนาฬิกาจาก FAST_CLOCK ซึ่งแต่ละประเภทมีความจุประเภทย่อยละ 8kB ในส่วนของหน่วยความจำแบบ RTC_FAST เป็นหน่วยความจำที่สามารถถูกใช้โดย PRO_CPU และหน่วยความจำแบบ RTC_SLOW ถูกเข้าถึงและใช้งานได้จาก APP_CPU เท่านั้น

นอกจากนี้ ULP เป็นตัวประมวลผลร่วมสถาปัตยกรรม 32 บิต ที่มีขนาดของคำสั่งและการอ้างอิงหน่วยความจำแบบ 32 บิต พร้อมกับเรจิสเตอร์ใช้งานทั่วไปขนาด 16 บิตจำนวน 4 ตัว และถูกออกแบบสำหรับชิป XL เพื่อทำงานแบบสเตตแมชชีนแบบง่ายหรือ FSM (Finite State Machine) โดยใช้พลังงานต่ำเมื่อหน่วยประมวลผลหลักของ ESP32 อยู่ในโหมดประหยัดพลังงาน (Deep Sleep Mode) จึงสามารถวัดค่าผลลัพธ์จากโมดูล ADC (Analog to Digital Converter) ที่ทำหน้าที่แปลงแรงดันไฟฟ้ากระแสตรงแบบแอนะล็อกให้เป็นค่าตัวเลขแบบดิจิทัล, เซ็นเซอร์อุณหภูมิ และเซ็นเซอร์ภายนอกผ่านทางวิธีการสื่อสารแบบ I²C โดยไม่ถูกหยุดทำงานไปพร้อมกับ ESP32 นอกจากนี้ หน่วยประมวลผลประเภท ULP สามารถเข้าถึงหน่วยความจำของโมดูล RTC ที่ทำงานช้า หรือ RTC_SLOW_MEM และเรจิสเตอร์ของ RTC อันได้แก่ RTC_CNTL, RTC_IO และ SARADC เมื่ออยู่ในโหมดประหยัดพลังงาน

คำสั่งสำหรับอ่านและเขียนข้อมูลในส่วนของหน่วยความจำสำหรับผู้ใช้งานจากโมดูล RTC คือ rtcUserMemoryRead() และ rtcUserMemoryWrite() โดยผลลัพธ์คือความสำเร็จของการอ่านและเขียนเป็น true หรือ false ซึ่งรูปแบบของคำสั่งมีดังนี้

```
ผลลัพธ์ = ESP.rtcUserMemoryRead( offset, data, size )
```

```
ผลลัพธ์ = ESP.rtcUserMemoryWrite( offset, data, size )
```

โดยที่

offset คือ ค่าตำแหน่งของหน่วยความจำที่ต้องการเข้าถึง

data คือ บัฟเฟอร์หน่วยความจำ

size คือ จำนวนไบนารีของหน่วยความจำ

โมดูล ESP32-S2 และ ESP32-S3 ใช้หน่วยประมวลผลรุ่นใหม่กว่าเป็น LX7 แบบ 1 แกน และ 2 แกนตามลำดับ โดย ESP32-S2 ไม่รองรับ BLE และมีรายละเอียดด้านคุณสมบัติที่เพิ่มเติม พร้อมทั้งเปลี่ยนหน่วยประมวลผลเสริมจาก ULP เป็น ULP ที่รองรับสถาปัตยกรรม RISC-V นอกจากนี้ ในตัว ESP32-S3 มีโมดูลสำหรับรองรับการประมวลผลเครือข่ายประสาทเทียม หรือ NN (Neural Network) พร้อมทั้งกำลังออกรุ่น ESP32-H2 เพื่อตอบสนองการทำงานที่ซับซ้อนขึ้น ส่วนโมดูล ESP32-C3 มีความแตกต่างเนื่องจากใช้หน่วยประมวลผลที่รองรับ RISC-V จึงไม่ได้ใช้ชุดคำสั่งเหมือนกับ LX6 และ LX7 แต่ถูกออกแบบมาทดแทนชิปในตระกูล esp8266 และปัจจุบันมีไมโครคอนโทรลเลอร์รุ่น ESP32-C6 ที่รองรับ WiFi 6 และ BLE 5.0 ออกมาจำหน่าย

ชุดคำสั่งตรวจสอบชิป

การตรวจสอบรายละเอียดของชิปต้องเรียกใช้ไฟล์ส่วนหัว ESP.h เพื่อใช้งานคลาส ESP ที่มีชุดคำสั่งสำหรับอ่านข้อมูลรุ่นของชิป อ่านโมเดลของชิป อ่านจำนวนแกนของชิป อ่านความเร็วของสัญญาณนาฬิกาที่กำลังทำงาน อ่านจำนวนการนับไชนเคิล อ่านรุ่นของเฟิร์มแวร์ (ESP-IDF) และรหัสของอีฟิวส์ (EFuse MAC) เป็นดังนี้ โดยตัวอย่างโปรแกรมสำหรับแสดงรายละเอียดของชิปด้วยชุดคำสั่งดังที่กล่าวมาเป็นดังตัวอย่างโปรแกรม 1-1 และผลลัพธ์ของโปรแกรมเป็นดังภาพที่ 1-3

```
หมายเลข = ESP.getChipRevision()
ชื่อโมเดลของชิป = ESP.getChipModel()
จำนวนแกน = ESP.getChipCores()
ความถี่สัญญาณนาฬิกา = ESP.getCpuFreqMHz()
จำนวนไชนเคิล = uint32_t ESP.getCycleCount()
ชื่อรุ่นของSDK = ESP.getSdkVersion()
หมายเลขฟิวส์ = ESP.getEfuseMac()
```


ตัวอย่างโปรแกรม 1-1 อ่านข้อมูลชิป ESP32

บรรทัด	คำสั่ง
1	#include <ESP.h>
2	void setup () {
3	Serial.begin(115200);
4	}
5	void loop () {
6	Serial.println("\n\n");
7	Serial.print("Chip revision: ");
8	Serial.println(ESP.getChipRevision());
9	Serial.print("Chip model: ");
10	Serial.println(ESP.getChipModel());
11	Serial.print("Chip cores: ");
12	Serial.println(ESP.getChipCores());
13	Serial.print("CPU Frequency (MHz): ");
14	Serial.println(ESP.getCpuFreqMHz());
15	Serial.print("Cycle count: ");
16	Serial.println(ESP.getCycleCount());
17	Serial.print("SDK Version: ");
18	Serial.println(ESP.getSdkVersion());
19	Serial.print("EFuse MAC.: ");
20	Serial.println(ESP.getEfuseMac());
21	delay(5000);
22	}

หมายเหตุ

คำสั่งเกี่ยวกับคลาส Serial สำหรับส่งรับค่าหรือแสดงผลผ่านทางพอร์ตสื่อสาร RS232 อธิบายอยู่ในบทที่ 2

<pre> Chip revision: 1 Chip model: ESP32-D0WDQ6 Chip cores: 2 CPU Frequency (MHz) ...: 240 Cycle count: 2412249900 SDK Version: v4.4.6-dirty EFuse MAC.: 70740253133956 </pre>	
ภาพที่ 1-3	ผลลัพธ์ของตัวอย่างโปรแกรม 1-1

ภาพโดย จารุต บุศราทิจ

> แรม

แรมภายในของ ESP32 มีปริมาณ 520kB โดยแบ่งเป็นส่วนของ **IRAM** (Instruction RAM) ขนาด 200kB สำหรับใช้เก็บชุดคำสั่งที่ต้องนำไปประมวลผล และ **DRAM** (Data RAM) เป็น 160kB จำนวน 2 ชุด สำหรับใช้ในการประมวลผลชุดคำสั่งและเก็บข้อมูลสำหรับใช้ประมวลผลด้วยแกน **PRO_CPU** และ **APP_CPU** โดยการทำงานจะเป็นดังภาพที่ 1-4 หากมีการเข้าถึงหน่วยความจำภายนอก ไม่ว่าจะเป็นกรณีที่ผู้พัฒนาโปรแกรมกำหนดให้มีการใช้งานโดยตรง หรือกรณีที่หน่วยความจำภายในของระบบมีปริมาณไม่เพียงพอจนจำเป็นต้องมีการถ่ายโอนข้อมูลจากหน่วยความจำภายในไปยังหน่วยความจำภายนอก และนำข้อมูลกลับเข้าสู่หน่วยความจำภายในสลับกันไปมา กระบวนการดังกล่าวจะถูกควบคุมและบริหารจัดการโดย **MMU**

ชุดคำสั่งสำหรับอ่านจำนวนหน่วยความจำ (Heap Size) ปริมาณของหน่วยความจำที่เหลืออยู่ ขนาดของหน่วยความจำน้อยสุดตั้งแต่เริ่มต้นระบบ และอ่านจำนวนหน่วยความจำขนาดใหญ่สุดที่สามารถจองใช้งานได้นั้นมีรูปแบบของชุดคำสั่งในคลาส ESP ดังนี้

```

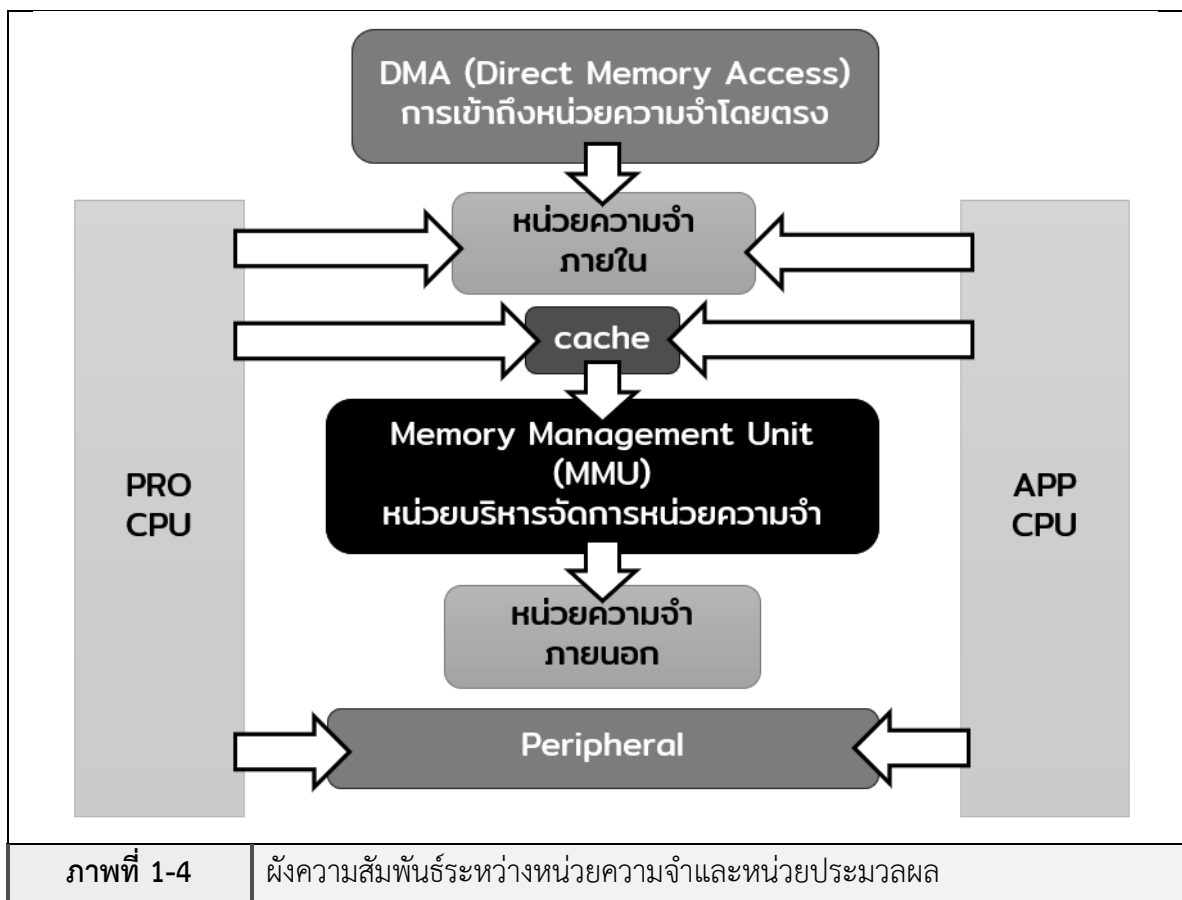
ขนาดหน่วยความจำ = ESP.getHeapSize()
หน่วยความจำที่เหลือ = ESP.getFreeHeap()
ขนาดที่น้อยสุด = ESP.getMinFreeHeap()
ขนาดที่มากที่สุด = ESP.getMaxAllocHeap()

```

นอกจากหน่วยความจำแรมภายในแล้ว ไมโครคอนโทรลเลอร์ ESP32 ถูกออกแบบให้รองรับการขยายหน่วยความจำแรมภายนอกโมดูล ESP32 ที่เชื่อมต่อผ่านบัส SPI จึงสามารถเพิ่มปริมาณ

หน่วยความจำแรมได้สูงสุด 8MB และเรียกหน่วยความจำประเภทนี้ว่า **PSRAM** หรือ SPI RAM โดยคำสั่งสำหรับอ่านปริมาณหน่วยความจำ ปริมาณหน่วยความจำที่เหลือ จำนวนหน่วยความจำที่เหลือน้อยสุด และปริมาณหน่วยความจำสูงสุดที่สามารถจองใช้งานได้ มีรูปแบบดังนี้

```
ขนาดหน่วยความจำ = ESP.getPsramSize()
หน่วยความจำที่เหลือ = ESP.getFreePsram()
ขนาดที่น้อยสุด = ESP.getMinFreePsram()
ขนาดที่มากที่สุด = ESP.getMaxAllocPsram()
```



ภาพโดย จารุต บุศราทิจ

ตัวอย่างโปรแกรมสำหรับการอ่านค่าของหน่วยความจำประเภทแรมมาแสดงผลโดยใช้ชุดคำสั่งจากคลาส ESP เป็นดังตัวอย่างโปรแกรม 1-2 และผลลัพธ์ของการทำงานเป็นตามภาพที่ 1-5

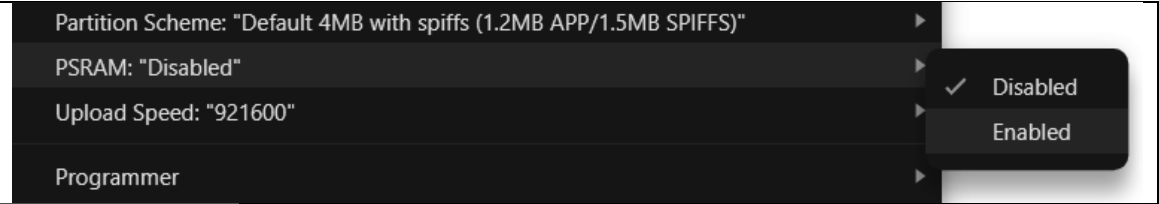
ตัวอย่างโปรแกรม 1-2 แสดงข้อมูลหน่วยความจำ

บรรทัด	คำสั่ง
1	#include <ESP.h>
2	void setup () {
3	Serial.begin(115200);
4	}
5	void loop () {
6	Serial.println("\n\n>>----RAM/HEAP-----<<");
7	Serial.print("Size: ");
8	Serial.println(ESP.getHeapSize());
9	Serial.print("Free: ");
10	Serial.println(ESP.getFreeHeap());
11	Serial.print("Min Free: ");
12	Serial.println(ESP.getMinFreeHeap());
13	Serial.print("Max Alloc: ");
14	Serial.println(ESP.getMaxAllocHeap());
15	Serial.println(">>-----PSRAM-----<<");
16	Serial.print("Size: ");
17	Serial.println(ESP.getPsramSize());
18	Serial.print("Free: ");
19	Serial.println(ESP.getFreePsram());
20	Serial.print("Min Free: ");
21	Serial.println(ESP.getMinFreePsram());
22	Serial.print("Max Alloc: ");
23	Serial.println(ESP.getMaxAllocPsram());
24	delay(5000);
25	}

<pre> >>-----RAM/HEAP-----<< Size: 376720 Free: 351844 Min Free: 346344 Max Alloc ...: 114676 >>-----PSRAM-----<< Size: 0 Free: 0 Min Free: 0 Max Alloc ...: 0 </pre>	
ภาพที่ 1-5	ผลลัพธ์ของตัวอย่างโปรแกรม 1-2

ภาพโดย จารุต บุศราทิจ

การเปิดใช้งานหน่วยความจำ PSRAM จะต้องกำหนดในการตั้งค่าใน Arduino IDE ดังภาพที่ 1-6 ให้ PSRAM: เป็น Enabled

	
ภาพที่ 1-6	การกำหนดให้ PSRAM ทำงาน

ภาพโดย จารุต บุศราทิจ

การใช้งานหน่วยความจำ PSRAM เพื่อจองพื้นที่ หรือจองพื้นที่พร้อมล้างค่าในก่อนหน่วยความจำที่จองได้ให้เป็น 0 ทั้งหมด การปรับขนาดการจองหน่วยความจำของหน่วยความจำที่ถูกจองมาก่อนหน้านี้ และการยกเลิกการจองหน่วยความจำต้องใช้คำสั่ง `ps_malloc()`, `ps_calloc()`, `ps_realloc()` และ `free()` ดังรูปแบบต่อไปนี้

```

ตำแหน่งหน่วยความจำ = ps_malloc( ขนาดของหน่วยความจำ )
ตำแหน่งหน่วยความจำ = ps_calloc( ขนาดของหน่วยความจำ )
ตำแหน่งหน่วยความจำ = ps_realloc( ตำแหน่งหน่วยความจำเดิม,
                                ขนาดของหน่วยความจำ )

free ( ตำแหน่งหน่วยความจำ )

```

ตัวอย่างชุดคำสั่งสำหรับใช้งานหน่วยความจำภายนอกเพื่อจองพื้นที่ขนาด 128 ไบต์ โดยใช้ตัวแปร `m128_ptr` เป็นตัวแปรอ้างอิงสำหรับชี้ไปยังตำแหน่งพื้นที่ในหน่วยความจำ PSRAM หลังจากนั้นจึงสั่งปรับขนาดเป็น 256 ไบต์พร้อมให้ตัวแปร `m256_ptr` เป็นตัวแปรชี้ไปยังตำแหน่งของหน่วยความจำตำแหน่งใหม่ สุดท้ายโค้ดโปรแกรมทำการยกเลิกการใช้หน่วยความจำด้วยคำสั่ง `free()` โดยจุดสังเกตที่ผู้เขียนต้องการให้ผู้อ่านระมัดระวัง คือ ตอนประกาศตัวแปร `m128_ptr` และ `m256_ptr` ควรกำหนดให้มีค่าเป็น `NULL` เพื่อใช้ในการตรวจสอบความสำเร็จในการจองหน่วยความจำ และเมื่อทำการยกเลิกการจองจะต้องกำหนดค่าให้กับตัวแปรเป็น `NULL` เช่นเดียวกัน

```
uint8_t * m128_ptr = NULL;
uint8_t * m256_ptr = NULL;
m128_ptr = (uint8_t*)ps_malloc(128);
if (m128_ptr != NULL) {
    m256_ptr=(uint8_t*)ps_realloc(m128_ptr,256);
    if (m256_ptr != NULL) {
        free(m128_ptr);
        m128_ptr = NULL;
        free(m256_ptr);
        m256_ptr = NULL;
    }
}
```

สำหรับการนำไปประยุกต์กับการจองหน่วยความจำแบบปกติซึ่งจะถูกใช้ในส่วน of DRAM สามารถเปลี่ยนแปลงการเขียนโค้ดได้เป็นดังนี้

```
uint8_t * m128_ptr = NULL;
uint8_t * m256_ptr = NULL;
m128_ptr = (uint8_t*)malloc(128);
if (m128_ptr != NULL) {
    m256_ptr=(uint8_t*)realloc( m128_ptr, 256 );
    if (m256_ptr != NULL) {
        free(m128_ptr);
        m128_ptr = NULL;
        free(m256_ptr);
        m256_ptr = NULL;
    }
}
```

ดังนั้น เพื่อความปลอดภัยและสะดวกในการเขียนโปรแกรมยกเลิกการจองหน่วยความจำ ผู้อ่านสามารถเขียนมาโครสำหรับการทำงานนี้ได้ด้วยชุดคำสั่ง `SAFE_FREE(x)` ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```
#define SAFE_FREE(x) { free(x); x = NULL; }
```

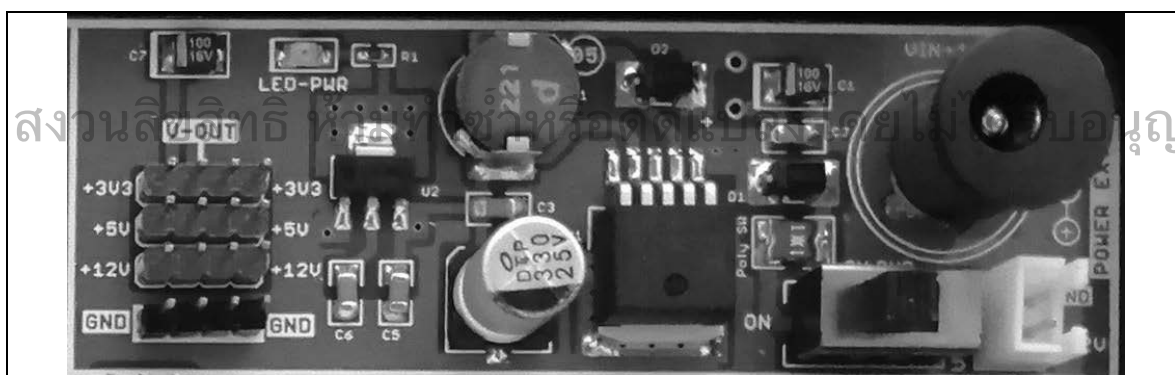
เมื่อนำไปใช้กับการยกเลิกหน่วยความจำสามารถเขียนได้ดังนี้

```
uint8_t * m128_ptr = NULL;
uint8_t * m256_ptr = NULL;
m128_ptr = (uint8_t*)malloc(128);
if (m128_ptr != NULL) {
    m256_ptr=(uint8_t*)realloc( m128_ptr, 256 );
    if (m256_ptr != NULL) {
        SAFE_FREE(m128_ptr);
        SAFE_FREE (m256_ptr);
    }
}
```

> ภาคจ่ายไฟและการแปลงแรงดัน

บนบอร์ด ET-ESP32 LAB V1 มีภาคจ่ายไฟที่แยกจากบอร์ดโมดูล ESP32 เพื่อรองรับการทำงานในกรณีที่มีการต่อพ่วงอุปกรณ์ภายนอกที่ใช้กระแสเยอะกว่าที่บอร์ด ESP32 รองรับได้ โดยภาคจ่ายไฟต้องการแหล่งจ่ายไฟจาก Power DC 12V ทั้งแบบหัวแจ็คที่แกนด้านในเป็น GND และด้านนอกเป็น 12V หรือต่อแบบสายไฟ 2 ขั้ว เพื่อลดทอนเป็นแรงดันกระแสตรง 5V และ 3.3V

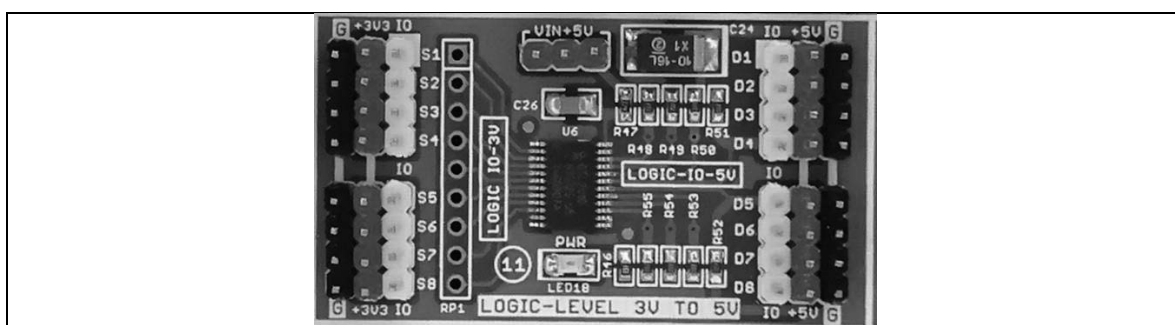
ในส่วนของแหล่งจ่ายไฟมีขั้วต่อแบบ 4 ขา สำหรับเป็น GND, 3.3V, 5V และ 12V ดังภาพที่ 1-7 เพื่อใช้งานกับวงจรที่ต่อเพิ่มหรือต่อเข้ากับวงจรส่วนต่าง ๆ ของ ET-ESP32 LAB V1.0 เพื่อลดภาระโหลดของภาคจ่ายไฟ USB-C ของบอร์ด ESP32 นอกจากนี้ บนบอร์ดมีสวิตช์สำหรับเปิดและปิดการทำงานของภาคจ่ายไฟและหลอดแอลอีดีแสดงสถานะของการทำงานทำให้สะดวกต่อการปิดและเปิดแหล่งจ่ายไฟโดยไม่ต้องถอดแล้วเสียบใหม่เพื่อหยุดการทำงานและเริ่มต้นระบบใหม่



ภาพที่ 1-7

ภาคจ่ายไฟของบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ



ภาพที่ 1-8

โมดูล LOGIC-LEVEL 3V TO 5V

ภาพโดย จารุต บุศราทิจ

ด้วยไมโครคอนโทรลเลอร์ ESP32 ทำงานกับแรงดันไฟฟ้ากระแสตรง 3.3V แต่ในบางกรณีต้องมีการเชื่อมต่อกับอุปกรณ์ภายนอกที่ใช้แรงดันกระแสตรง 5V อันเป็นแรงดันที่เกินกว่าที่ ESP32 ส่งออกได้ หรือรับเข้ามาใช้งานได้ ทำให้ชุดทดลองมีภาคแปลงแรงดันลอจิกไปมาระหว่าง 3.3 กับ 5 โวลต์ผ่านทางโมดูล LOGIC-LEVEL 3V TO 5V ดังภาพที่ 1-8

> แฟลชรอม

แฟลชรอม (FlashROM) ภายนอกขนาด 4MB ที่มากับโมดูล ESP32 ใช้หมายเลข P6, P7, P8, P11, P16 และ P17 ใช้เชื่อมต่อกับวงจรหน่วยความจำแฟลชของไมโครคอนโทรลเลอร์ดังนั้นไม่ควรใช้ขาเหล่านี้กับวงจรภายนอกถ้าชิปที่เลือกใช้มีหน่วยความจำแฟลชรอมภายนอก โดยเฉพาะชิป WROOM32 ใช้ขาเชื่อมต่อกับวงจรของหน่วยความจำแฟลชดังนี้

- P6 ทำหน้าที่ SCK/CLK
- P7 ทำหน้าที่ SDO/SD0
- P8 ทำหน้าที่ SDI/SD1
- P11 ทำหน้าที่ SHD/SD2
- P16 ทำหน้าที่ SWP/SD3
- P17 ทำหน้าที่ CSC/CMD

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ในบางกรณีที่เกิดเหตุการณ์ขนาดของโปรแกรมหรือชุดคำสั่งมีขนาดมากกว่าขนาดของพื้นที่ในส่วนของ IRAM (เก็บได้สูงสุด 200kB) ทำให้ไม่สามารถเก็บชุดคำสั่งในหน่วยความจำได้ ทำให้ไมโครคอนโทรลเลอร์ ESP32 ได้ออกแบบส่วนของตัวระบบบริหารหน่วยความจำ หรือ MMU (Memory Manage Unit) สำหรับดำเนินการจัดเก็บชุดคำสั่งที่เกิน 200kB ลงใน IROM ซึ่งเป็นพื้นที่ของแฟลชรอมที่ถูกเก็บไว้เป็นที่สำรองชุดคำสั่งที่ไม่สามารถเก็บลง IRAM ได้

เช่นเดียวกัน เมื่อมีการใช้หน่วยความจำส่วนของ DRAM ซึ่งเป็นพื้นที่เก็บข้อมูล (ใช้ได้สูงสุด 160kB จาก 320kB) สำหรับประมวลผล โดยมีข้อมูลขนาดมากกว่าพื้นที่สูงสุดที่เก็บได้จะเกิดเหตุการณ์ที่ MMU ทำการนำข้อมูลเหล่านี้เก็บลงในส่วนของ DROM อันเป็นพื้นที่ของแฟลชรอมที่ถูกกันเอาไว้สำหรับเก็บข้อมูล

นอกจากนี้ในการใช้งานแฟลชรอมของ ESP32 นั้นสามารถบริหารจัดการได้หลากหลายแบบ โดยแบ่งส่วนของแฟลชรอมผ่านทางโปรแกรม Arduino IDE ดังตัวอย่างในภาพที่ 1-9 เป็นการเลือกใช้หน่วยความจำแฟลชในรูปแบบ Default 4MB with spiffs (1.2MB APP/ 1.5MB SPIFFS) หมายถึงแบ่งหน่วยความจำแฟลชรอมขนาด 4MB ออกเป็น 2 ส่วน สำหรับเก็บโปรแกรมเป็นพื้นที่ 1.2MB และเป็นพื้นที่หน่วยความจำแฟลชระบบไฟล์ SPIFFS อีก 1.5MB จึงสามารถนำไฟล์ภายนอกไปเก็บไว้ในส่วนนี้ได้ หากเลือก FATFS (FAT File system) จะมีความแตกต่างในเรื่องของโครงสร้างการจัดเก็บเป็นแบบ FAT (File Allocation Table) เหมือนกับระบบปฏิบัติการของบริษัทไมโครซอฟต์



ภาพโดย จารุต บุศราทิจ

ชุดคำสั่งสำหรับอ่านข้อมูลของแฟลชรอม ได้แก่ อ่านขนาดหน่วยความจำของชิป, ความเร็วชิป และโหมดของชิป คือ `getFlashChipSize()`, `getFlashChipSpeed()` และ `getFlashChipMode()` โดยมีรูปแบบของการใช้งานดังนี้

```
ขนาดของแฟลช = ESP.getFlashChipSize()
```

```
ความเร็ว = ESP.getFlashChipSpeed()
```

```
โหมด = ESP.getFlashChipMode()
```

โดยที่ โหมด มีความหมายดังนี้

- FM_QIO หรือ SPI_FREAD_QIO
- FM_QOUT หรือ SPI_FREAD_QUAD
- FM_DIO หรือ SPI_FREAD_DIO
- FM_DOUT หรือ SPI_FREAD_DUAL
- FM_FAST_READ หรือ SPI_FASTRD_MODE
- FM_SLOW_READ หรือไม่ใช่โหมดที่กล่าวมาข้างต้น

กรณีที่ต้องการแปลค่าคืนกลับที่ได้รับมาจากชุดคำสั่ง `getFlashChipSize()`, `getFlashChip Speed()` และ `getFlashChipMode()` สามารถใช้ฟังก์ชันสำหรับแปลความหมายตามรูปแบบของการใช้งานคำสั่ง

แปลค่าของขนาดชิป ค่าความเร็วของชิป และโหมดของชิป ได้แก่ คำสั่ง `magicFlashChipSize()`, `magicFlashChipSpeed()` และ `magicFlashChipMode()` ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ขนาดของแฟลช = `ESP.magicFlashChipSize( ตัวเลข )`

ความเร็ว = `ESP.magicFlashChipSpeed( ตัวเลข )`

โหมด = `ESP.magicFlashChipMode( ตัวเลข )`

ค่าพารามิเตอร์ ตัวเลข ของ `magicFlashChipSize()` มีความหมายหรือค่าที่คืนกลับออกมาเป็นดังนี้

- 1 คืนค่ากลับ 2_MB
- 2 คืนค่ากลับ 4_MB
- 3 คืนค่ากลับ 8_MB
- 4 คืนค่ากลับ 16_MB
- อื่น ๆ จะเกิดความผิดพลาด และคืนค่ากลับเป็น 0

ค่าพารามิเตอร์ ตัวเลข ของ `magicFlashChipSpeed()` มีความหมายหรือค่าที่คืนกลับออกมาเป็นดังนี้

- 0 คืนค่ากลับมาเป็น 40_MHz
- 1 คืนค่ากลับมาเป็น 26_MHz
- 2 คืนค่ากลับมาเป็น 20_MHz
- 15 คืนค่ากลับมาเป็น 80_MHz
- อื่น ๆ คืนค่ากลับเป็น 0

ค่าพารามิเตอร์ ตัวเลข ของ `magicFlashChipMode()` มีความหมายหรือค่าที่คืนกลับออกมาเป็นดังนี้

- ถ้ามีค่ามากกว่า `FM_SLOW_READ` จะคืนค่าเป็น `FM_UNKNOWN`
- กรณีอื่น ๆ จะคืนค่าตาม ตัวเลข ที่ส่งเข้ามา

สำหรับการอ่านขนาดของโปรแกรม ค่ารหัสข้อความที่เข้ารหัสตามขั้นตอนวิธีแบบ MD5 ของโปรแกรม และจำนวนพื้นที่เหลือของโปรแกรมจะใช้คำสั่ง `getSketchSize()`, `getSketchMD5()` และ `getFreeSketchSpace()` ตามรูปแบบของการใช้งานดังนี้

```
ขนาด = ESP.getSketchSize()
```

```
ข้อความรหัส MD5 = ESP.getSketchMD5()
```

```
พื้นที่ที่เหลือ = ESP.getFreeSketchSpace()
```

ชุดคำสั่งสำหรับลบหน่วยความจำ ณ ตำแหน่งเซกเตอร์ที่ต้องการต้องใช้คำสั่ง `flashEraseSector()` และสำหรับการเขียนข้อมูลขนาด `size` ที่ถูกชี้ด้วยตัวแปร `data` ณ ตำแหน่ง `offset` เป็นต้นไปจะต้องใช้คำสั่ง `flashWrite()` และ `flashRead()` โดยให้ ผลการทำงาน กลับมาเป็น `true` หรือ `false` ตามรูปแบบของการใช้งานดังนี้

```
ผลการทำงาน = ESP.flashEraseSector( sector )
```

```
ผลการทำงาน = ESP.flashWrite( offset, data, size )
```

```
ผลการทำงาน = ESP.flashRead( offset, data, size )
```

นอกจากนี้ ผู้เขียนโปรแกรมสามารถสั่งลบพาร์ติชันของหน่วยความจำแฟลชได้ด้วยคำสั่ง `partitionEraseRange()` หรือทำการเขียนและอ่านพาร์ติชันของหน่วยความจำแฟลชด้วยคำสั่ง `partitionWrite()` และ `partitionRead()` โดยระบุพารามิเตอร์เป็นตำแหน่งของพาร์ติชัน (`partition`) ค่าตำแหน่งที่เริ่มต้นเข้าถึง (`offset`) บัฟเฟอร์ของข้อมูล (`data`) และขนาด (`size`) ของพาร์ติชันหรือก่อนข้อมูลตามรูปแบบของการใช้งานดังนี้

```
ผลการทำงาน = ESP.partitionEraseRange(
```

```
    partition, offset, size )
```

```
ผลการทำงาน = ESP.partitionWrite(
```

```
    partition, offset, data, size )
```

```
ผลการทำงาน = ESP.partitionRead(
```

```
    partition, offset, data, size )
```

ตัวอย่างโปรแกรม 1-3 เป็นตัวอย่างการแสดงผลข้อมูลหน่วยความจำแฟลชอันได้แก่ ความจุ ความเร็ว และโหมดการทำงาน และตัวอย่างของผลลัพธ์เป็นดังภาพที่ 1-10

ตัวอย่างโปรแกรม 1-3 อ่านข้อมูลแฟลชรอมภายนอกของ ESP32

บรรทัด	คำสั่ง
1	#include <ESP.h>
2	String sFlashMode[] = {
3	"FM_QIO", "FM_QOUT", "FM_DIO", "FM_DOUT",
4	"FM_FAST_READ", "FM_SLOW_READ", "FM_UNKNOWN" };
5	void setup() {
6	Serial.begin(115200);
7	}
8	void loop() {
9	Serial.print("FlashChipSize (Byte).....: ");
10	Serial.println(ESP.magicFlashChipSize(
11	ESP.getFlashChipSize()));
12	Serial.print("FlashChipSpeed (Hz).....: ");
13	Serial.println(ESP.magicFlashChipSpeed(
14	ESP.getFlashChipSpeed()));
15	Serial.print("FlashChipMode: ");
16	Serial.println(sFlashMode[
17	ESP.magicFlashChipMode(
18	ESP.getFlashChipMode()));
19	Serial.println("-----");
20	delay(5000);
21	}

```
FlashChipSize (Byte).....: 1048576
FlashChipSpeed (Hz).....: 40000000
FlashChipMode .....: FM_QIO
```

ภาพที่ 1-10

ตัวอย่างผลลัพธ์จากโปรแกรมตัวอย่าง 1-3

ภาพโดย จารุต บุศราทิจ

> เซ็นเซอร์ภายใน ESP32

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ไมโครคอนโทรลเลอร์ ESP32 มีเซ็นเซอร์ภายในโมดูล 2 ตัว คือ ตัววัดอุณหภูมิ (Temperature Sensor) และตัววัดความเข้มของสัญญาณแม่เหล็ก (Hall Effect Sensor) โดยคำสั่งสำหรับอ่านค่าจากเซ็นเซอร์อุณหภูมิเพื่อให้คืน ค่าอุณหภูมิ แบบองศาฟาเรนไฮต์กลับมา มีรูปแบบของชุดคำสั่งดังนี้

ค่าอุณหภูมิ = temprature_sens_read()

ตัวอย่างการอ่านข้อมูลจากเซ็นเซอร์อุณหภูมิภายในโมดูล ESP32 ทุก 1 วินาที เพื่อนำออกไปยังพอร์ตสื่อสารอนุกรมที่เชื่อมต่อกับโปรแกรม Arduino IDE เป็นดังตัวอย่างโปรแกรม 1-4

ตัวอย่างโปรแกรม 1-4 การอ่านข้อมูลจากเซ็นเซอร์อุณหภูมิภายในชิป ESP32

บรรทัด	คำสั่ง
1	#ifdef __cplusplus
2	extern "C" {
3	#endif
4	uint8_t temprature_sens_read();
5	#ifdef __cplusplus
6	}
7	#endif
8	void setup() {
9	Serial.begin(115200);
10	}
11	void loop() {
12	Serial.print("Internal temp. = ");
13	Serial.print(temprature_sens_read());
14	Serial.println("F.");
15	delay(1000);
16	}

นอกจากเซ็นเซอร์อุณหภูมิแล้ว ภายในโมดูลของ ESP32 มีเซ็นเซอร์วัดค่าสนามแม่เหล็กที่มีหลักการทำงานของเซ็นเซอร์ คือ เมื่อมีแม่เหล็กขั้วได้เข้ามาใกล้เซ็นเซอร์จะส่งผลให้แรงดันมีค่าสูงขึ้น

จากปกติ ในทางกลับกัน เมื่อขั้วเหนือเข้าใกล้เซ็นเซอร์จะส่งผลให้แรงดันลดลงกว่าปกติ โดยการอ่านค่าจากเซ็นเซอร์ใช้คำสั่ง `hallRead()` ที่ส่งคืน ค่าความเข้มข้นแม่เหล็ก ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ค่าความเข้มข้นแม่เหล็ก = `hallRead()`

ตัวอย่างโปรแกรมสำหรับแสดงค่าจากเซ็นเซอร์แม่เหล็กผ่านทาง Serial Monitor ของ Arduino IDE เป็นดังตัวอย่างโปรแกรม 1-5

ตัวอย่างโปรแกรม 1-5 ตัวอย่างอ่านค่าคลื่นแม่เหล็กจากเซ็นเซอร์ของ ESP32

บรรทัด	คำสั่ง
1	<code>void setup() {</code>
2	<code> Serial.begin(115200);</code>
3	<code>}</code>
4	<code>void loop() {</code>
5	<code> Serial.print("Hall Effect = ");</code>
6	<code> Serial.println(hallRead());</code>
7	<code> delay(1000);</code>
8	<code>}</code>

> การขัดจังหวะ

การขัดจังหวะ หรือ อินเตอร์รัพต์ (Interrupt) เป็นกระบวนการที่ถูกออกแบบมาเพื่อให้ไมโครคอนโทรลเลอร์ใช้ในการทำงานเพื่อรองรับการตอบสนองแบบตรงเวลา เมื่อสัญญาณการขัดจังหวะเกิดขึ้นเมื่อใด สถานะของการทำงานของหน่วยประมวลผลในขณะนั้นจะถูกจัดเก็บสถานะไว้และไปทำงานที่ตอบสนองกับแหล่งการเกิดสัญญาณขัดจังหวะ เมื่อทำส่วนของโปรแกรมที่กำหนดเพื่อตอบสนองการขัดจังหวะจนเสร็จสิ้น หน่วยประมวลผลจึงกลับมาทำงานต่อจากชุดคำสั่งที่ทำค้างไว้

ไมโครคอนโทรลเลอร์ ESP32 แต่ละแกนรองรับการเกิดสัญญาณขัดจังหวะ 32 แหล่งกำเนิดสัญญาณ โดยแหล่งสัญญาณการขัดจังหวะนั้นมีการออกแบบให้มีลำดับความสำคัญและมีแหล่งกำเนิดจาก

2 ประเภท คือ การขัดจังหวะจากฮาร์ดแวร์ (Hardware Interrupts) ที่เกิดจากสัญญาณภายนอกโมดูล ESP32 เช่น การขัดจังหวะจากสวิตช์ภายนอก หรือเซ็นเซอร์แบบสัมผัสจากขาของไมโครคอนโทรลเลอร์ เป็นต้น และการขัดจังหวะจากซอฟต์แวร์ (Software Interrupts) ที่เกิดจากการทำงานภายในของโมดูล ESP32 เช่น จาก WDT หรือตัวตั้งเวลา เป็นต้น โดยการขัดจังหวะที่เกิดต่อหน่วยประมวลผลทั้ง 2 (PRO_CPU และ APP_CPU) เป็นดังตารางที่ 1-2 ที่อธิบายแหล่งการการขัดจังหวะทั้ง 32 ประเภท พร้อมระบุแหล่งการเกิด (Category) ประเภทของการขัดจังหวะ (Type) และระดับความสำคัญ (Priority Level) ของการขัดจังหวะที่เกิดขึ้น

ตารางที่ 1-2 การขัดจังหวะ

ลำดับ	แหล่งการเกิด		ประเภท	ระดับความสำคัญ
0	Peripheral	Level-Triggered	1	0
1	Peripheral	Level-Triggered	1	1
2	Peripheral	Level-Triggered	1	2
3	Peripheral	Level-Triggered	1	3
4	Peripheral	Level-Triggered	1	0
5	Peripheral	Level-Triggered	1	5
6	Internal	Timer.0	1	6
7	Internal	Software	1	7
8	Peripheral	Level-Triggered	1	8
9	Peripheral	Level-Triggered	1	9
10	Peripheral	Edge-Triggered	1	10
11	Internal	Profiling	3	11
12	Peripheral	Level-Triggered	1	12
13	Peripheral	Level-Triggered	1	13
14	Peripheral	NMI	NMI	14
15	Internal	Timer.1	3	15
16	Internal	Timer.2	5	16
17	Peripheral	Level-Triggered	1	17
18	Peripheral	Level-Triggered	1	18
19	Peripheral	Level-Triggered	2	19
20	Peripheral	Level-Triggered	2	20
21	Peripheral	Level-Triggered	2	21
22	Peripheral	Edge-Triggered	3	22
23	Peripheral	Level-Triggered	3	23

ตารางที่ 1-2 การขัดจังหวะ (ต่อ)

ลำดับ	แหล่งการเกิด		ประเภท	ระดับความสำคัญ
24	Peripheral	Level-Triggered	4	24
25	Peripheral	Level-Triggered	4	25
26	Peripheral	Level-Triggered	5	26
27	Peripheral	Level-Triggered	3	27
28	Peripheral	Edge-Triggered	4	28
29	Internal	Software	3	29
30	Peripheral	Edge-Triggered	4	30
31	Peripheral	Level-Triggered	5	31

ที่มา: ดัดแปลงจาก Espressif Systems. (2024). *ESP32 Technical Reference Manual*. น. 178.
สืบค้นเมื่อ มีนาคม 18, 2567, จาก
https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf

การเชื่อมสัญญาณขัดจังหวะจากขาของ ESP32

คำสั่งสำหรับเชื่อมสัญญาณขัดจังหวะของ ESP32 ต้องกำหนด หมายเลขขา ที่ต้องการตรวจสอบ การเกิดสัญญาณขัดจังหวะตาม โหมดการเกิดสัญญาณขัดจังหวะ และเมื่อเป็นจริงตามเงื่อนไขของโหมด ให้ไปทำงานในฟังก์ชันตาม ชื่อฟังก์ชันตอบสนอง ดังรูปแบบต่อไปนี้

attachInterrupt(หมายเลขขา,
ชื่อฟังก์ชันตอบสนอง,
โหมดการเกิดสัญญาณขัดจังหวะ)

โดยที่

โหมดการเกิดสัญญาณขัดจังหวะ มีค่าตามประเภทของการเปลี่ยนแปลงสถานะของแรงดันที่ หมายเลขขา ที่กำหนด อันได้แก่

- **LOW** สร้างสัญญาณขัดจังหวะเมื่อระดับแรงดันเปลี่ยนสถานะเป็น LOW เสร็จสิ้นแล้ว
- **HIGH** สร้างสัญญาณขัดจังหวะเมื่อระดับแรงดันเปลี่ยนสถานะเป็น HIGH เสร็จสิ้นแล้ว
- **CHANGE** สร้างสัญญาณการขัดจังหวะเมื่อเกิดการเปลี่ยนแปลงระดับแรงดันระหว่างการเป็น HIGH หรือ LOW

- FALLING สร้างสัญญาณขัดจังหวะเมื่อเกิดการเปลี่ยนแปลงระดับแรงดันจากสถานะ HIGH มาเป็น LOW

- RISING สร้างสัญญาณการขัดจังหวะเมื่อเกิดการเปลี่ยนแปลงระดับแรงดันจากสถานะ LOW ไปเป็น HIGH

เมื่อต้องการยกเลิกการตรวจสอบการเกิดการขัดจังหวะสามารถกระทำด้วยการสั่ง detachInterrupt() ตามด้วย หมายเลขขา ที่ต้องการยกเลิก ดังรูปแบบของคำสั่งต่อไปนี้

detachInterrupt(หมายเลขขา)

ฟังก์ชันสำหรับรองรับการเกิดการขัดจังหวะ หรือ ISR (Interrupt Service Routine) ต้องมีรูปแบบการประกาศตามโครงสร้างต่อไปนี้

```
void IRAM_ATTR ชื่อฟังก์ชัน( void * arg ) {
    // ชุดคำสั่งที่ทำงานเมื่อเกิดการขัดจังหวะ
}
```

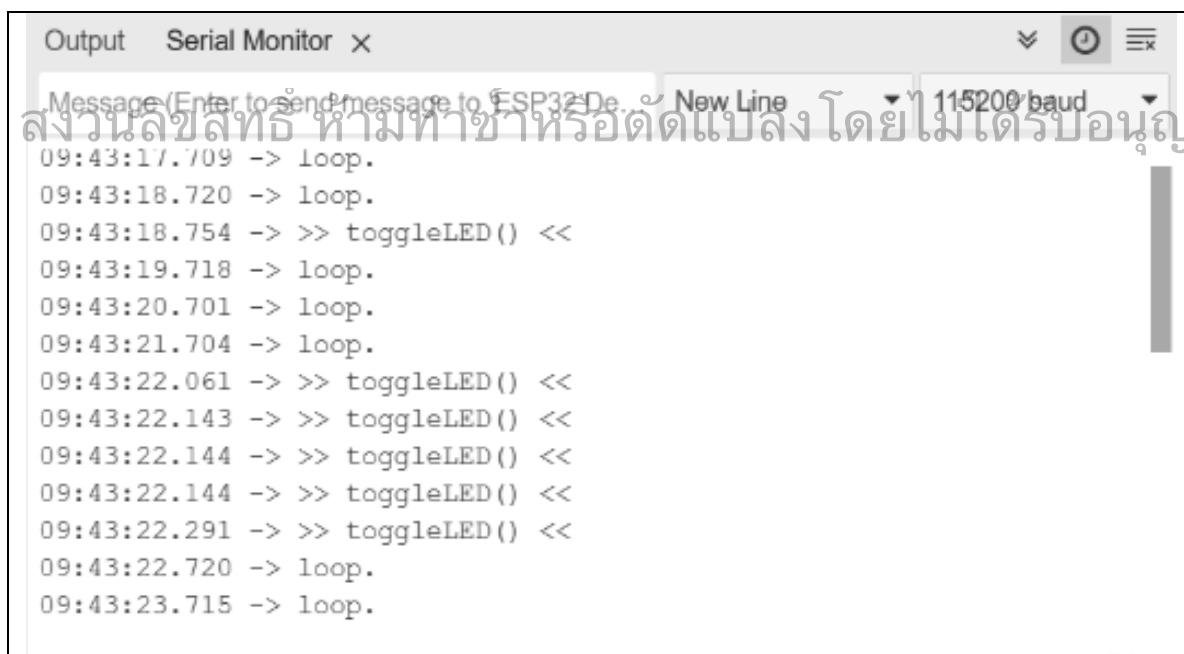
จากภาพที่ 1-11 และโค้ดโปรแกรมตัวอย่าง 1-6 จะพบว่าได้กำหนดให้ขา 34 เชื่อมต่อกับวงจรสวิตช์พร้อมทั้งกำหนดให้ทำงานการขัดจังหวะเป็นประเภท FALLING ดังนั้น เมื่อมีการเปลี่ยนระดับของแรงดันจาก HIGH มาเป็น LOW จะเกิดสัญญาณขัดจังหวะเกิดขึ้น ส่งผลให้ฟังก์ชัน toggleLED() ทำงานดังตัวอย่างในภาพที่ 1-11 เป็นผลลัพธ์ของช่วงเวลา 09:43:22.061, 09:43:22.143, 09:43:22.144, 09:43:22.144 และ 09:43:22.291 ที่เกิดการแสดงข้อความที่เกิดจากฟังก์ชัน toggleLED() อย่างต่อเนื่องโดยระยะห่างของเวลาน้อยกว่า 1 วินาที เป็นต้น การทำงานปกติของโปรแกรมทำการแสดงข้อความ loop. ทุก 1 วินาที แต่ถ้ามีการกดสวิตช์จะเกิดข้อความ >> toggleLED() << แทรกเข้ามาโดยเวลาของการเกิดข้อความนี้จะไม่เป็นช่วงเวลาระยะห่าง 1 วินาที เนื่องจากหน่วยประมวลผลจะออกจากการรอหรือคำสั่งอื่น ๆ เพื่อไปทำงานที่ฟังก์ชัน toggleLED() แล้วดำเนินการตามโค้ดที่เขียนหลังจากนั้นจึงกลับมาทำที่ค้างไว้ก่อนข้ามไปทำที่ toggleLED() อีกครั้ง

หมายเหตุ

- **IRAM_ATTR** คือ การระบุให้สิ่งที่สร้างขึ้นจะถูกโหลดเข้าสู่หน่วยความจำ IRAM อันเป็นหน่วยความจำขนาด 200kB ที่ถูกแบ่งมาจาก SRAM ที่มีขนาด 520kB ของ ESP32 เพื่อลดปัญหาความล่าช้าจากการโหลดชุดคำสั่งนั้นจากแฟลชรอมมาเก็บในแรมเพื่อทำงาน ดังนั้นฟังก์ชันตอบสนองการขัดจังหวะที่ต้องอาศัยการตอบสนองที่รวดเร็วจึงต้องถูกประกาศคุณสมบัติของฟังก์ชันด้วยประเภtno_inline
- **DRAM_ATTR** คือ การระบุให้สิ่งที่สร้างขึ้นจัดเก็บอยู่ในหน่วยความจำ DRAM ที่มีขนาด 320kB แต่ถูกใช้ได้สูงสุดเพียง 160kB และเป็นส่วนที่ถูกใช้เป็นหน่วยความจำในการทำงานของหน่วยประมวลผลของ ESP32

ตัวอย่างโปรแกรม 1-6 ตัวอย่างการขัดจังหวะ

บรรทัด	คำสั่ง
1	#define PIN_SW 34
2	#define PIN_LED 21
3	uint8_t statusLED = 1;
4	void IRAM_ATTR toggleLED() {
5	if (statusLED) statusLED = 0;
6	else statusLED = 1;
7	Serial.println(">> toggleLED() <<");
8	}
9	void setup() {
10	Serial.begin(115200);
11	pinMode(PIN_SW, INPUT_PULLUP);
12	pinMode(PIN_LED, OUTPUT);
13	attachInterrupt(PIN_SW, toggleLED, FALLING);
14	}
15	void loop() {
16	Serial.println("loop.");
17	delay(1000);
18	}



ภาพที่ 1-11

ตัวอย่างผลลัพธ์จากตัวอย่างโปรแกรม 1-6

ภาพโดย จารุต บุศราทิจ

> ตัวตั้งเวลา

ไทมเมอร์ (Timer) หรือตัวตั้งเวลามีหน้าที่ใช้สำหรับให้ไมโครคอนโทรลเลอร์ทำงานตามระยะเวลาที่กำหนด หรือสร้างสัญญาณขัดจังหวะจากการอยู่ในโหมดประหยัดพลังงาน โดยไทมเมอร์ของ ESP32 มีฮาร์ดแวร์ไทมเมอร์จำนวน 2 กลุ่ม ในแต่ละกลุ่มประกอบไปด้วยไทมเมอร์ฮาร์ดแวร์ย่อยอีก 2 ตัว ทำให้ ESP32 รองรับการใช้ไทมเมอร์ได้ทั้งสิ้น 4 ตัว และทั้ง 4 นี้เป็นไทมเมอร์แบบ 64 บิตที่ทำงานแบบ 16 บิต pre-scalers และตัวนับค่าขึ้น/ลงขนาด 64 บิต ที่รองรับการทำงานรีโวลต์ค่าอัตโนมัติ

ใน ESP32 นั้น ตัวไทมเมอร์จะใช้สัญญาณนาฬิกาจาก APB_CLK ซึ่งปกติกำหนดไว้เป็นความถี่ 80MHz หรือ 80000000Hz และถูกแบ่งย่อยความถี่ด้วยค่า Pre-scalers ขนาด 16 บิต โดยสามารถมีค่าได้ตั้งแต่ 2 ถึง 65535 เพื่อสร้างสัญญาณนาฬิกาความถี่ TB_clk และสามารถกำหนดให้เกิดการนับค่าขึ้นหรือลงครั้งละ 1 ค่าตามความถี่ TB_clk ตัวอย่างการคำนวณเมื่อ Pre-scalers เป็น 80 จะได้ TB_clk เป็นดังสมการที่ 1

$$\begin{aligned} \text{TB_clk} &= \frac{80000000}{80} \\ &= 1000000\text{Hz} \end{aligned}$$

สมการที่ 1

คำสั่ง `timerBegin()` เป็นคำสั่งกำหนดค่าการทำงานให้กับไทมเมอร์ที่ต้องการสั่งการ และ `timerEnd()` เป็นการยกเลิกการสั่งการไทมเมอร์ โดยรูปแบบของการทำงานเป็นดังนี้

ตัวชี้ไทมเมอร์ = `timerBegin( หมายเลขไทมเมอร์, ค่าตัวหาร, รูปแบบการนับค่า )`
`timerEnd( ตัวชี้ไทมเมอร์ )`

โดยที่

ตัวชี้ไทมเมอร์ คือ ตัวแปรเก็บตำแหน่งของไทมเมอร์ที่ตั้งค่าการทำงาน แต่ถ้าค่าเป็น NULL หมายความว่า การตั้งค่าไม่สำเร็จทำให้ไม่สามารถใช้ตัวตั้งเวลาที่ต้องการใช้งานได้

หมายเลขไทมเมอร์ คือ ตัวเลขของไทมเมอร์ที่ต้องการใช้ ซึ่งมีค่าอยู่ในช่วง 0 ถึง 3

ค่าตัวหาร คือ ค่าตัวเลขจำนวนเต็มที่เรียกว่า Pre-scaler สำหรับนำไปหารสัญญาณนาฬิกา APB_CLK โดยต้องมีค่าอยู่ในช่วง 0 ถึง 65535 เท่านั้น

รูปแบบการนับค่า คือ รูปแบบของการนับค่า โดย เป็นการนับค่าขึ้นถ้ากำหนดเป็น true และ เป็นการนับลงถ้ากำหนดให้เป็น false

การกำหนดฟังก์ชันที่ต้องถูกเรียกทำงานเมื่อไทมเมอร์นับค่าขึ้นหรือลงจนครบช่วงต้องใช้คำสั่ง `timerAttachInterrupt()` และถ้าต้องการยกเลิกการตอบสนองจากตัวไทมเมอร์จะต้องส่งด้วยคำสั่ง `timerDetachInterrupt()` ตามรูปแบบของการสั่งงานดังนี้

`timerAttachInterrupt( ตัวชี้ไทมเมอร์, ชื่อฟังก์ชัน, โหมดการขัดจังหวะ )`
`timerDetachInterrupt( ตัวชี้ไทมเมอร์ )`

โดยที่

ตัวชี้ไทมเมอร์ คือ ตัวแปรเก็บตำแหน่งของไทมเมอร์ที่ได้จากคำสั่ง `timerBegin()`

ชื่อฟังก์ชัน คือ ชื่อของฟังก์ชันหรือโปรแกรมน้อยๆที่มีรูปแบบการเขียนเหมือนกับโปรแกรมน้อยๆที่ตอบสนองการขัดจังหวะดังที่กล่าวในหัวข้อก่อนหน้านี้

โหมดการขัดจังหวะ คือ รูปแบบของการขัดจังหวะที่ทำการตรวจสอบ โดยมีค่าได้เพียง 2 แบบ คือ true สำหรับการดักสถานะเมื่อเป็น Edge Trigger หรือ false สำหรับดัก Level Trigger

เมื่อสร้างตัวตั้งเวลาพร้อมทั้งกำหนดฟังก์ชันตอบสนองการขัดจังหวะจากตัวตั้งเวลา ขั้นตอนต่อไปเป็นเรื่องการบริหารจัดการทำงานนับค่าตามสัญญาณาฬิกา คือ สั่งให้เริ่มทำงาน หยุดทำงาน และเริ่มทำงานใหม่ ด้วยคำสั่ง `timerStart()`, `timerStop()` และ `timerRestart()` ตามรูปแบบของฟังก์ชันดังนี้

```
timerStart( ตัวชี้ไทมเมอร์ )
timerStop( ตัวชี้ไทมเมอร์ )
timerRestart( ตัวชี้ไทมเมอร์ )
```

โดยที่

ตัวชี้ไทมเมอร์ คือ ตัวแปรเก็บตำแหน่งของไทมเมอร์ที่ได้จากคำสั่ง `timerBegin()` ตัวไทมเมอร์จะยังไม่เริ่มทำงานถ้าไม่สั่ง `timerStart()` และคำสั่ง `timerRestart()` จะเป็นการให้เริ่มทำการนับค่าใหม่พร้อมตั้งค่าตัวนับใหม่

นอกจากการสั่งเริ่ม หยุด และเริ่มต้นใหม่แล้ว ผู้เขียนโปรแกรมสามารถตั้งค่าของเรจิสเตอร์ที่ทำหน้าที่เป็นตัวนับค่าของไทมเมอร์ที่ต้องการใช้งานด้วยคำสั่ง `timerWrite()` หรือตั้งค่าตัวหารด้วยคำสั่ง `timerSetDivider()` ทำให้กำหนดค่า pre-scaler ขึ้นใหม่โดยไม่ต้องยกเลิกไทมเมอร์และสร้างใหม่อีกครั้ง หรืออ่านค่าตัวนับที่เก็บอยู่ในเรจิสเตอร์ออกมาด้วย 4 คำสั่ง คือ `timerRead()` หรือ `timerReadMicros()` หรือ `timerReadMillis()` หรือ `timerReadSeconds()` ทำให้สามารถทราบค่าการนับปัจจุบัน หรืออ่านค่าแบบหน่วยไมโครวินาที หรือ ค่าในหน่วยมิลลิวินาที หรือ ค่าในหน่วยวินาทีตามลำดับดังรูปแบบการใช้งานต่อไปนี้

```
timerWrite( ตัวชี้ไทมเมอร์, ค่าตัวนับ )
timerSetDivider ( ตัวชี้ไทมเมอร์, ค่าตัวหาร )
ผลลัพธ์ = timerRead( ตัวชี้ไทมเมอร์ )
ผลลัพธ์ = timerReadMicros( ตัวชี้ไทมเมอร์ )
ผลลัพธ์ = timerReadMillis( ตัวชี้ไทมเมอร์ )
ผลลัพธ์ = timerReadSeconds( ตัวชี้ไทมเมอร์ )
```

โดยที่

ตัวชี้ไทมเมอร์ คือ ตัวแปรเก็บตำแหน่งของไทมเมอร์ที่ได้จากคำสั่ง timerBegin()
ค่าตัวนับ คือ ค่าตัวเลขที่ต้องการกำหนดให้กับตัวไทมเมอร์ที่กำหนด

ค่าตัวหาร คือ ค่าตัวเลขจำนวนเต็มที่เรียกว่า Pre-scaler สำหรับนำไปหารสัญญาณนาฬิกา APB_CLK โดยต้องมีค่าอยู่ในช่วง 0 ถึง 65535 เท่านั้น

ผลลัพธ์ คือ ค่าที่อ่านได้จากคำสั่งอ่านค่าจากไทมเมอร์ที่กำหนด

สำหรับคำสั่งกลุ่มสุดท้ายเป็นคำสั่งสำหรับควบคุมการทำงานของการทำงานของไทมเมอร์ นับจนเกิดการขัดจังหวะ อันได้แก่ การเปิดการแจ้งเตือน การยกเลิกการแจ้งเตือน การตั้งค่าการแจ้งเตือน อ่านค่าการแจ้งเตือน และอ่านค่าการแจ้งเตือนระดับไมโครวินาที ด้วยคำสั่งด้านการแจ้งเตือน คือ timerAlarmEnable(), timerAlarmDisable(), timerAlarmWrite(), timerAlarmRead() และ คำสั่งสุดท้ายคือ timerAlarmReadMicros() ดังรูปแบบการใช้งานต่อไปนี้

timerAlarmEnable(ตัวชี้ไทมเมอร์)

timerAlarmDisable(ตัวชี้ไทมเมอร์)

timerAlarmWrite (ตัวชี้ไทมเมอร์, ค่าตัวเลข, การโหลดค่าใหม่อัตโนมัติ)

ผลลัพธ์ = **timerAlarmRead(ตัวชี้ไทมเมอร์)**

ผลลัพธ์ = **timerAlarmReadMicros(ตัวชี้ไทมเมอร์)**

โดยที่

ตัวชี้ไทมเมอร์ คือ ตัวแปรเก็บตำแหน่งของไทมเมอร์ที่ได้จากคำสั่ง timerBegin()

ค่าตัวเลข คือ ค่าตัวเลขที่ต้องการกำหนดให้กับตัวแจ้งเตือน

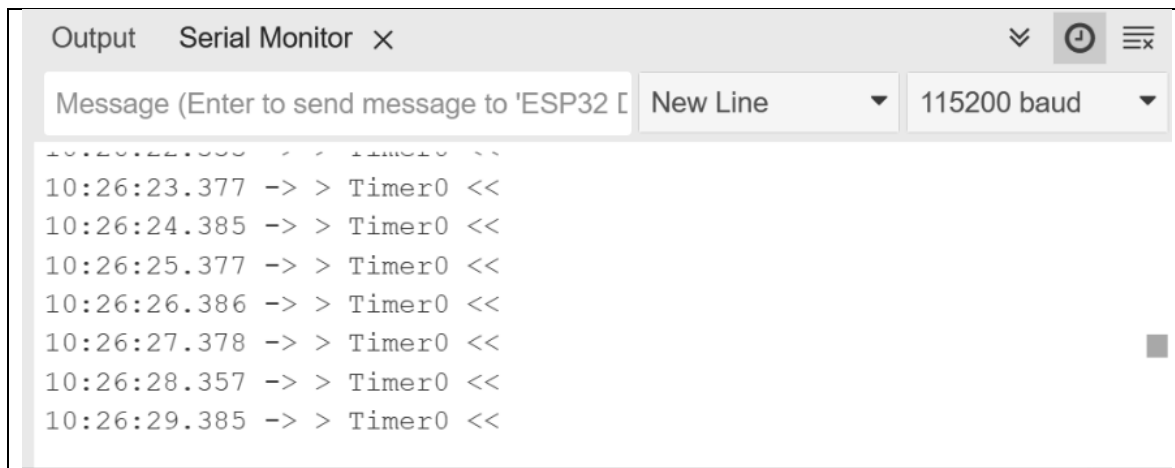
การโหลดค่าใหม่อัตโนมัติ คือ ค่า true หรือ false เพื่อเปิด/ปิดการโหลดค่าใหม่อัตโนมัติ (auto reload)

ผลลัพธ์ คือ ค่าที่อ่านได้จากคำสั่งอ่านค่าจากไทมเมอร์ที่กำหนด

ตัวอย่างโปรแกรม 1-7 เป็นตัวอย่างการใช้ไทมเมอร์เพื่อแจ้งข้อความออกทางพอร์ตสื่อสารอนุกรมผ่านทางโปรแกรม Serial Monitor โดยตัวอย่างผลลัพธ์ของการทำงานในภาพที่ 1-12 จะพบว่าโปรแกรมมีการตั้งเวลาเพื่อส่งข้อความทุกประมาณ 1 วินาที

ตัวอย่างโปรแกรม 1-7 การใช้การขัดจังหวะจากไทมเมอร์

บรรทัด	คำสั่ง
1	hw_timer_t* Timer0Config = NULL;
2	void IRAM_ATTR toggleLED() {
3	Serial.println("> Timer0 <<");
4	}
5	void setup() {
6	Timer0Config = timerBegin(0, 80, true);
7	timerAttachInterrupt(Timer0Config,
8	&toggleLED, true);
9	timerAlarmWrite(Timer0Config, 1000000, true);
10	timerAlarmEnable(Timer0Config);
11	Serial.begin(115200);
12	}
13	void loop() {
14	delay(3000);
15	}



ภาพที่ 1-12

ตัวอย่างผลลัพธ์จากการทำงานของโปรแกรม 1-7

ภาพโดย จารุต์ บุศราทิจ

> WDT

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

WDT (Watch Dog Timer) เป็นนาฬิกาของไมโครคอนโทรลเลอร์ที่ใช้สำหรับรีเซ็ตตนเอง ในกรณีที่ซอฟต์แวร์ที่กำลังทำงานนั้นไม่ส่งสัญญาณตอบสนองกลับมาที่ระบบโดยการเรียกใช้ฟังก์ชัน `wdtFeed()` ทั้งนี้เพื่อป้องกันไม่ใหัระบบเกิดสถานะว่างโดยไม่สามารถทำอะไรได้ เช่น อาการทำงานไม่รู้จบหรือแฮ้ง (Hang) เป็นต้น

โมดูล WDT ของ ESP32 มีหน้าที่รอการตอบสนองจากโปรแกรมใน APP_CPU ผ่านทางคำสั่ง `esp_task_wdt_reset()` ในระยะเวลาหน่วยวินาทีตามที่กำหนดในคำสั่ง `esp_task_wdt_init()` และถูกเริ่มต้นทำงานหลังจากเรียกด้วย `esp_task_wdt_add()` ตามรูปแบบของคำสั่งดังต่อไปนี้

```
esp_task_wdt_init( ค่าเวลา, true )
esp_task_wdt_add( NULL )
esp_task_wdt_reset( )
```

การใช้งานชุดคำสั่งของ WDT จะต้องนำเข้าไฟล์ `esp_task_wdt.h` ด้วยคำสั่ง `#include` ดังนี้

```
#include <esp_task_wdt.h>
```

ขั้นตอนของการใช้ WDT มีดังนี้

- ตั้งค่าการทำงาน ด้วยคำสั่ง `esp_task_wdt_init( เวลาที่ตั้ง, true )`
- เปิดให้ทำงาน ด้วยคำสั่ง `esp_task_wdt_add( NULL )`
- ส่งสัญญาณรีเซ็ตไปให้ WDT ด้วยคำสั่ง `esp_task_wdt_reset()`

ตัวอย่างโปรแกรมสำหรับส่งสัญญาณรีเซ็ตให้ WDT โดยในแต่ละรอบการทำงานของฟังก์ชัน `loop()` ได้เพิ่มค่าการหน่วงเวลาเพิ่มขึ้นครั้งละ 1 วินาที ดังนั้น เมื่อกำหนดระยะเวลาสำหรับการส่งสัญญาณรีเซ็ตให้อยู่ภายใน 5 วินาที จึงทำให้เมื่อโปรแกรมวนรอบจนครั้งที่ 6 จะทำให้เกิดการรีเซ็ตเป็นดังตัวอย่างโปรแกรม 1-8 และตัวอย่างผลลัพธ์ของการทำงานเป็นดังภาพที่ 1-12

ตัวอย่างโปรแกรม 1-8 การใช้งาน WDT รีเซ็ตการทำงาน

บรรทัด	คำสั่ง
1	#include <esp_task_wdt.h>
2	#define WDT_TIMEOUT 5
3	void setup() {
4	Serial.begin(115200);
5	esp_task_wdt_init(WDT_TIMEOUT, true);
6	esp_task_wdt_add(NULL);
7	}
8	int loop_counter = 0;
9	void loop() {
10	esp_task_wdt_reset();
11	loop_counter++;
12	Serial.println(loop_counter);
13	delay(loop_counter*1000);
14	}

```

1
2
3
4
5
E (30103) task_wdt: Task watchdog got triggered. The following tasks did not reset the watchdog in time:
E (30103) task_wdt: - loopTask (CPU 1)
E (30103) task_wdt: Tasks currently running:
E (30103) task_wdt: CPU 0: IDLE
E (30103) task_wdt: CPU 1: IDLE
E (30103) task_wdt: Aborting.

abort() was called at PC 0x400d8609 on core 0

Backtrace: 0x4008340d:0x3ffbe9cc |<-CORRUPTED

ELF file SHA256: c25f9736a99f6db1

Rebooting...
ets Jun  8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0

```

ภาพที่ 1-13

ตัวอย่างผลลัพธ์จากการทำงานของโปรแกรม 1-8

ภาพโดย จารุต บุศราทิจ

จากภาพที่ 1-13 จะพบว่าเมื่อโปรแกรมเริ่มนับ 1, 2, 3, 4 และ 5 หลังจากนั้นจะเข้าสู่ขั้นตอนของ task wdt แจ้งเตือนว่าเกิดการวนรอบซ้ำ (loopTask) ใน CPU 1 หรือ APP_CPU เกินระยะเวลาที่ตั้งไว้ ทำให้เกิดการเรียกคำสั่ง abort() ของ CORE_0 หรือ CPU_0 หรือ PRO_CPU ทำให้เกิดการรีเซ็ตการทำงาน of ระบบและเริ่มการทำงานของโปรแกรมใหม่อีกรอบ ซึ่งจะเกิดเหตุการณ์นี้ซ้ำไปเรื่อย ๆ ตามโปรแกรมที่เขียนไว้

> โหมดประหยัดพลังงาน

ไมโครคอนโทรลเลอร์ ESP32 มีโหมดการใช้พลังงานด้วยกัน 5 โหมด คือ โหมดทำงาน (Active Mode), โหมดปิดการสื่อสาร (Modem Sleep Mode), โหมดใช้พลังงานน้อย (Light Sleep Mode), โหมดประหยัดพลังงาน และโหมดพักการทำงาน (Hibernation Mode) โดยแต่ละโหมดมีข้อกำหนดดังตารางที่ 1-3

ตารางที่ 1-3 การทำงานในโหมดใช้พลังงานแบบต่าง ๆ ของ ESP32

โหมด	Active	Modem	Light	Deep	Hibernation
CPU	ทำงาน	ทำงาน	พัก	ไม่ทำงาน	ไม่ทำงาน
WiFi/BT	ทำงาน	ไม่ทำงาน	ไม่ทำงาน	ไม่ทำงาน	ไม่ทำงาน
RTC	ทำงาน	ทำงาน	ทำงาน	ทำงาน	ไม่ทำงาน
ULP	ทำงาน	ทำงาน	ทำงาน	ทำงาน/ไม่ทำงาน	ไม่ทำงาน
การใช้พลังงาน	240mA	ความเร็วสูง 30mA-50mA ความเร็วปกติ 20mA-25mA ความเร็วน้อย 2mA-4mA	0.8mA	ULP ทำงาน 150uA ULP-เซ็นเซอร์ 100uA RTC 10uA	7uA

ที่มา: ดัดแปลงจาก Espressif Systems. (2024). *ESP32 Technical Reference Manual*. น. 30.
สืบค้นเมื่อ มีนาคม 18, 2567, จาก
https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf

การสั่งให้ไมโครคอนโทรลเลอร์เข้าสู่โหมดประหยัดพลังงานมีรูปแบบของการใช้งานคำสั่งดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```
esp_deep_sleep_start( )
```

เมื่อเข้าสู่โหมดประหยัดพลังงานจะมีขา GPIO กลุ่มหนึ่งที่ยังคงทำงานได้เนื่องจากถูกควบคุมด้วย RTC ดังนี้

- RTC_GPIO0 คือ P36 ที่สามารถนำเข้าข้อมูลได้เพียงอย่างเดียว
- RTC_GPIO3 คือ P39 ที่สามารถนำเข้าข้อมูลได้เพียงอย่างเดียว
- RTC_GPIO4 คือ P34 ที่สามารถนำเข้าข้อมูลได้เพียงอย่างเดียว
- RTC_GPIO5 คือ P35 ที่สามารถนำเข้าข้อมูลได้เพียงอย่างเดียว
- RTC_GPIO6 คือ P25
- RTC_GPIO7 คือ P26
- RTC_GPIO8 คือ P33
- RTC_GPIO9 คือ P32
- RTC_GPIO10 คือ P4
- RTC_GPIO11 คือ P0
- RTC_GPIO12 คือ P2
- RTC_GPIO13 คือ P15
- RTC_GPIO14 คือ P13
- RTC_GPIO15 คือ P12
- RTC_GPIO16 คือ P14
- RTC_GPIO17 คือ P27

การกลับมาจากโหมดประหยัดพลังงานมี 4 วิธี คือ การปลุกด้วยไทมเมอร์ การรับการขัดจังหวะจากภายนอก การรับสัญญาณจากขา Touch และการปลุกด้วย ULP โดยในหนังสือเล่มนี้ไม่ได้อธิบายสำหรับกรณีการใช้ ULP

การกลับจากโหมดประหยัดพลังงานด้วยการขัดจังหวะ

การปลุกจากโหมดประหยัดพลังงานด้วยการขัดจังหวะจากภายนอกจะต้องเลือกการใช้งานว่าเป็น EXT0 หรือ EXT1 ด้วยคำสั่ง `esp_sleep_enable_ext0_wakeup()` หรือ

esp_sleep_enable_ext1_wakeup() มีความแตกต่างกันคือ EXT1 จะใช้ได้กับขาในกลุ่ม RTC แต่ EXT0 ใช้ได้กับขาปกติ ซึ่งรูปแบบของคำสั่งเป็นดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```
esp_sleep_enable_ext0_wakeup( หมายเลขขา, โหมด )
```

```
esp_sleep_enable_ext1_wakeup( กลุ่มของหมายเลขขา,โหมด )
```

โดยที่

หมายเลขขา คือ หมายเลขขาที่ใช้เป็นตัวนำเข้าสู่สัญญาณขัดจังหวะเพื่อปลุกให้ไมโครคอนโทรลเลอร์ ESP32 กลับจากโหมดประหยัดพลังงาน โดยต้องใช้เป็นค่าคงที่ GPIO_NUM_หมายเลขขา เช่น ดักขา P32 จะต้องใช้ชื่อเป็น GPIO_NUM_32

กลุ่มของหมายเลขขา คือ ค่าบิตของขา GPIO ที่ทำงานโดย RTC ที่มีค่าเป็นเลขจำนวนเต็มที่มาจากการคำนวณตามสมการ $2^{\text{หมายเลขขา}}$ เช่นต้องการดักขา P32 อันเป็นขา RTC_GPIO9 จะได้เป็น 4294967296 ในเลขฐาน 10 และเมื่อเป็นเลขฐาน 2 จะได้เป็น

$00010000000000000000000000000000_2$ ซึ่งเขียนเป็นตัวเลขในรูปแบบของภาษา C/C++ ได้เป็น $0b00010000000000000000000000000000$ หรือเมื่อแปลงค่าเป็นเลขฐาน 16 จะได้ค่าเป็น 100000000_{16} และเขียนในรูปแบบตัวเลขฐาน 16 ของภาษา C/C++ ได้เป็น $0x100000000$ เป็นต้น ส่วนการดักหลายขา เช่น ต้องการตรวจสอบขา RTC_GPIO9 กับ RTC_GPIO8 ซึ่งเป็นขา P32 และ P33 จะได้ค่าตัวเลขฐาน 16 ของขาทั้ง 2 เป็น $0x100000000$ กับ $0x100000000$ เมื่อนำมารวมกันจะได้กลุ่มของหมายเลขขาเป็น $0x300000000$ เป็นต้น

โหมด คือ ระดับสัญญาณที่ดักจับ โดย EXT0 จะใช้ค่า 0 หรือ 1 ดังนี้

- 0 สำหรับตรวจการเกิดระดับสัญญาณ LOW
- 1 สำหรับตรวจจับการเกิดระดับสัญญาณ HIGH

แต่ EXT1 ใช้ค่าคงที่ค่าใดค่าหนึ่งดังต่อไปนี้

- ESP_EXT1_WAKEUP_ALL_LOW สำหรับกรณีที่ให้กลับจากโหมดประหยัดพลังงานถ้าทุกขาใน กลุ่มของหมายเลขขา เป็น LOW
- ESP_EXT1_WAKEUP_ANY_HIGH สำหรับกรณีที่มีบางขาของ กลุ่มของหมายเลขขา มีสถานะเป็น HIGH

ตัวอย่างโปรแกรม 1-9 เป็นตัวอย่างของการเข้าสู่โหมดประหยัดพลังงานและปลุกกลับมาด้วยการขัดจังหวะ

ตัวอย่างโปรแกรม 1-9 การกลับจากโหมดประหยัดพลังงานด้วยการขัดจังหวะจากภายนอก

บรรทัด	คำสั่ง
1	void setup() {
2	Serial.begin(115200);
3	esp_sleep_enable_ext0_wakeup(GPIO_NUM_32, 0);
4	Serial.println("Wakeup!");
5	Serial.flush();
6	delay(1000);
7	Serial.println("Start sleep. [wait EXT0/P32]");
8	Serial.flush();
9	esp_deep_sleep_start();
10	}
11	void loop() {
12	}

การกลับจากโหมดประหยัดพลังงานด้วยไทมเมอร์

จากตัวอย่างโปรแกรม 1-10 จะพบว่าการเรียกใช้คำสั่งการตั้งค่าการปลุกด้วยไทมเมอร์ โดยการกำหนดค่าตัวเลขในหน่วยไมโครวินาทีให้กับ `esp_sleep_enable_timer_wakeup()` ตามรูปแบบการใช้งานดังนี้ และตัวอย่างเป็นดังภาพที่ 1-14

`esp_sleep_enable_timer_wakeup( จำนวนเวลา )`

ตัวอย่างโปรแกรม 1-10 การอ่านค่าเซ็นเซอร์และส่งออกไปยัง UART ทุก 10 วินาที โดยใช้โหมดประหยัดพลังงานที่ปลุกมาทำงานด้วยไทมเมอร์

บรรทัด	คำสั่ง
1	<code>#ifdef __cplusplus</code>
2	<code>extern "C" {</code>
3	<code>#endif</code>
4	<code>uint8_t temprature_sens_read();</code>
5	<code>#ifdef __cplusplus</code>
6	<code>}</code>
7	<code>#endif</code>
8	<code>#define USEC_TO_SEC 1000000</code>
9	<code>#define TIME_OUT 10</code>
10	<code>void setup() {</code>
11	<code>Serial.begin(115200);</code>
12	<code>esp_sleep_enable_timer_wakeup(</code>
13	<code>USEC_TO_SEC*TIME_OUT);</code>
14	<code>Serial.print(temprature_sens_read());</code>
15	<code>Serial.println("F");</code>
16	<code>Serial.flush();</code>
17	<code>delay(1000);</code>
18	<code>esp_deep_sleep_start();</code>
19	<code>}</code>
20	<code>void loop() {</code>
21	<code>}</code>

```

115F
ets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
115F

```

ภาพที่ 1-14

ตัวอย่างผลลัพธ์จากโปรแกรม 1-10

ภาพโดย จารุต บุศราทิจ

การใช้หน่วยความจำ RTC

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ในช่วงของการเข้าโหมดประหยัดพลังงานแล้วกลับมาเริ่มต้นทำงานใหม่จะเกิดการบูตระบบใหม่อีกครั้ง ทำให้ตัวแปรต่าง ๆ ที่เขียนจากโปรแกรมซึ่งถูกเก็บใน SRAM สำหรับให้แกหน่วยประมวลผลทำงานนั้นถูกยกเลิกไปด้วย ดังนั้น ถ้าต้องการจัดเก็บข้อมูลเพื่อนำไปใช้กับครั้งถัดไปจะต้องอาศัยหน่วยความจำของ RTC ขนาด 8kB โดยกำหนดแหล่งที่เก็บข้อมูลเอาไว้ด้านหน้าการประกาศตัวแปรด้วย RTC_DATA_ATTR เพื่อให้ระบบทราบว่าตัวแปรนั้นสร้างโดยเก็บไว้ใน RTC ดังรูปแบบต่อไปนี้

RTC_DATA_ATTR ประเภทตัวแปร ชื่อตัวแปร;

ตัวอย่างโปรแกรม 1-11 เป็นการนำตัวอย่างโปรแกรม 1-10 มาใช้แต่เพิ่มการทำงานหาค่าเฉลี่ยของอุณหภูมิ 3 ค่า ซึ่งโดยปกติจะอ่านและแสดงผลแต่ไม่สามารถจำค่าเดิมได้ ด้วยเหตุนี้จึงต้องเปลี่ยนตำแหน่งที่เก็บข้อมูลเป็น RTC_DATA_ATTR ทำให้ได้ผลลัพธ์ดังตัวอย่างในภาพที่ 1-15

ตัวอย่างโปรแกรม 1-11 การอ่านค่าเซ็นเซอร์และแสดงค่าเฉลี่ยทุกการทำงาน 5 ครั้ง

บรรทัด	คำสั่ง
1	<code>#ifdef __cplusplus</code>
2	<code>extern "C" {</code>
3	<code>#endif</code>
4	<code>uint8_t temprature_sens_read();</code>
5	<code>#ifdef __cplusplus</code>
6	<code>}</code>
7	<code>#endif</code>
8	<code>#define USEC_TO_SEC 1000000</code>
9	<code>#define TIME_OUT 10</code>
10	<code>#define MAX_NUM_TEMP 5</code>
11	<code>RTC_DATA_ATTR uint8_t counter=0;</code>
12	<code>RTC_DATA_ATTR uint8_t temps[MAX_NUM_TEMP];</code>
13	<code>void setup() {</code>
14	<code>Serial.begin(115200);</code>
15	<code>esp_sleep_enable_timer_wakeup(</code>
16	<code>USEC_TO_SEC*TIME_OUT);</code>
17	<code>temps[counter] = temprature_sens_read();</code>

ตัวอย่างโปรแกรม 1-11 การอ่านค่าเซ็นเซอร์และแสดงค่าเฉลี่ยทุกการทำงาน 5 ครั้ง (ต่อ)

บรรทัด	คำสั่ง
18	counter++;
19	Serial.print(counter);
20	Serial.print(":");
21	for (int i=0; i<counter; i++) {
22	Serial.print(temps[i]);
23	Serial.print("F ");
24	}
25	Serial.println();
26	if (counter == MAX_NUM_TEMP) {
27	counter = 0;
28	int sum=0;
29	for (int i=0; i<MAX_NUM_TEMP; i++) {
30	sum += temps[i];
31	}
32	Serial.print("Average =");
33	Serial.print(sum/MAX_NUM_TEMP);
34	Serial.println("F");
35	}
36	Serial.flush();
37	delay(1000);
38	esp_deep_sleep_start();
39	}
40	void loop() {}

```

rst:0x5 (DEEPSLEEP_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
5:114F 112F 113F 113F 113F
Average  =113F

```

ภาพที่ 1-15 ตัวอย่างผลลัพธ์จากโปรแกรม 1-11

ภาพโดย จารุต บุศราทิจ

การกลับจากโหมดประหยัดพลังงานด้วยการสัมผัส

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

การปลุกจากโหมดประหยัดพลังงานด้วยการสัมผัสของไมโครคอนโทรลเลอร์เป็นวิธีการขัดจังหวะประเภทหนึ่งแต่แหล่งการเกิดการขัดจังหวะมาจากขาที่ทำหน้าที่รองรับการทำงานเป็นขาสัมผัส หรือ Touch Pin หรือ Touch Pad อันประกอบด้วย

- TOUCH0 หรือ T0 คือขา P4
- TOUCH1 หรือ T1 คือขา P0
- TOUCH2 หรือ T2 คือขา P2
- TOUCH3 หรือ T3 คือขา P15
- TOUCH4 หรือ T4 คือขา P13
- TOUCH5 หรือ T5 คือขา P12
- TOUCH6 หรือ T6 คือขา P14
- TOUCH7 หรือ T7 คือขา P27
- TOUCH8 หรือ T8 คือขา P33
- TOUCH9 หรือ T9 คือขา P32

การตั้งค่าการทำงานระหว่างโหมดประหยัดพลังงานกับขาแบบสัมผัสต้องเรียกใช้คำสั่ง `esp_sleep_enable_touchpad_wakeup()` สำหรับเปิดให้รองรับการปลุกด้วยขาแบบสัมผัส ส่วนการกำหนดขาต้องใช้คำสั่ง `touchAttachInterrupt()` สำหรับระบุขา T0 ถึง T9 ที่ใช้เป็นขาตรวจสอบการสัมผัส และต้องสร้างตัวแปรเก็บสถานะของขาให้เป็นประเภท `touch_t` โดยสามารถอ่านสถานะของขาได้จากคำสั่งสำหรับอ่านสถานะของขาแบบสัมผัสของ ESP32 ชื่อ `esp_sleep_get_touchpad_wakeup_status()` ดังรูปแบบของการใช้งานต่อไปนี้

```
touch_t ชื่อตัวแปร;  
ชื่อตัวแปร= esp_sleep_get_touchpad_wakeup_status();  
esp_sleep_enable_touchpad_wakeup();  
touchAttachInterrupt( หมายเลขขา, ค่าเวลา);
```

โดยที่

หมายเลขฯ คือ ชื่อหมายเลขขาแบบสัมผัสชื่อของค่าคงที่เป็น T0, T1, T2, T3, T4, T5, T6, T7, T8 หรือ T9

ค่าเวลา คือ ค่าระยะเวลาที่นำมาใช้ในการตัดสินใจเกิดการสัมผัสหรือไม่ โดยถ้ามีค่ามากจะมีความเร็วในการตอบสนองมากกว่าค่าน้อย

ตัวอย่างโปรแกรม 1-12 เป็นการเข้าโหมดประหยัดพลังงานพร้อมทั้งตั้งการออกจากโหมดประหยัดพลังงานด้วยการตรวจสอบการสัมผัสขา T3 หรือขาหมายเลข P15 ของไมโครคอนโทรลเลอร์ ESP32

ตัวอย่างโปรแกรม 1-12 การกลับจากโหมดประหยัดพลังงานด้วยการสัมผัสขา T3

บรรทัด	คำสั่ง
1	#define THRESHOLD 100
2	String s[] = {
3	"T0/P4","T1/P0","T2/P2","T3/P15","T4/P13",
4	"T5/P12","T6/P14","T7/P27","T8/P33","T9/P32"};
5	void touch_stat() {
6	touch_pad_t touch_pin =
7	esp_sleep_get_touchpad_wakeup_status();
8	Serial.println(s[touch_pin]);
9	}
10	void setup() {
11	Serial.begin(115200);
12	touch_stat();
13	esp_sleep_enable_touchpad_wakeup();
14	touchSleepWakeUpEnable(T3,THRESHOLD);
15	Serial.flush();
16	delay(1000);
17	esp_deep_sleep_start();
18	}
19	void loop() {
20	}

สาเหตุการกลับจากโหมดประหยัดพลังงาน

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

กรณีที่ต้องการตรวจสอบสาเหตุของการออกจากโหมดประหยัดพลังงานสามารถกระทำได้ด้วย
การเรียกใช้คำสั่ง `esp_sleep_get_wakeup_cause()` เพื่ออ่านค่าคืนกลับประเภท
`esp_sleep_wakeup_cause_t` ตามรูปแบบของคำสั่งดังนี้

```
esp_sleep_wakeup_cause_t ชื่อตัวแปร;  
ชื่อตัวแปร = esp_sleep_get_wakeup_cause();
```

โดยที่ค่าที่ประเภท `esp_sleep_wakeup_cause_t` จะเป็นค่าคงที่ที่บอกละแหล่งหรือสาเหตุที่ทำให้หลุดออกจากโหมดประหยัดพลังงานเป็นค่าใดค่าหนึ่งในรายการต่อไปนี้ แต่ถ้าไม่ใช่ค่าเหล่านี้แสดงว่าเกิดการกลับมาด้วยสาเหตุอื่น ๆ ที่ไม่เกี่ยวข้อง เช่น ระบบแหล่งจ่ายไฟ หรือโดนรีเซ็ตจาก WDT เป็นต้น

- `ESP_SLEEP_WAKEUP_EXT0`
- `ESP_SLEEP_WAKEUP_EXT1`
- `ESP_SLEEP_WAKEUP_TIMER`
- `ESP_SLEEP_WAKEUP_TOUCHPAD`
- `ESP_SLEEP_WAKEUP_ULP`

> การรีเซ็ต

ตัว ESP32 รองรับการรีเซ็ตทำงานของระบบมี 3 ระดับ คือ

- CPU Reset สำหรับการรีเซ็ตเฉพาะเรจิสเตอร์ของ `PRO_CPU` และ `APP_CPU`
- Core Reset สำหรับรีเซ็ตเรจิสเตอร์ดิจิทัลทั้งหมด อันได้แก่ แกนของหน่วยประมวลผล
GPIO และ GPIO แบบดิจิทัล
- System Reset รีเซ็ตทั้งส่วนของ Core และ RTC
 - Chip Power On Reset
 - RWD System Reset เป็นการรีเซ็ตโดยโมดูล Watch Dog Timer

■ Brown Out Reset เป็นการรีเซ็ตจากการมีแรงดันไฟฟ้าต่ำกว่าค่ามาตรฐาน หรือแรงดันกระแสตรงที่ 2.4V การสั่งให้ไมโครคอนโทรลเลอร์ ESP32 รีเซ็ต (reset) ระบบหรือรีบูต (Reboot) หรือรีสตาร์ท (Restart) ระบบให้ใช้งานคำสั่งสำหรับงานดังกล่าวดังนี้

```
ESP.restart()
```

การตรวจสอบหาสาเหตุที่ระบบถูกรีเซ็ตสามารถทำได้ด้วยการตรวจสอบสถานะของการบูตได้จากคำสั่ง `rtc_get_reset_reason()` รูปแบบของคำสั่งการใช้งานคือ และต้องนำเข้าไฟล์ `"esp32/rom/rtc.h"` ก่อนใช้งาน

```
สาเหตุการรีเซ็ต = rtc_get_reset_reason( หมายเลขหน่วยประมวลผล )
```

โดยที่

หมายเลขหน่วยประมวลผล คือ ตัวเลข 0 สำหรับหมายถึง CPU0 หรือ PRO_CPU และ 1 หมายถึง CPU1 หรือ APP_CPU

สาเหตุการรีเซ็ต คือ ค่าของแต่ละสถานะเป็นดังนี้

- 1 หมายถึง POWERON_RESET หรือการรีเซ็ตจากการรีเซ็ตจากการจ่ายไฟจากแหล่งจ่ายไฟเข้ากับบอร์ดไมโครคอนโทรลเลอร์ ESP32
- 3 หมายถึง SW_RESET หรือ มีการเรียกคำสั่งรีเซ็ตจากซอฟต์แวร์
- 4 หมายถึง OWDT_RESET หรือ เกิดจากการรีเซ็ตจาก WDT
- 5 หมายถึง DEEPSLEEP_RESET หรือ เกิดการรีเซ็ตจากการกลับจากโหมดประหยัดพลังงาน
- 7 หมายถึง TG0WDT_SYS_RESET หรือ เกิดจาก TIMER0/WDT ทำการรีเซ็ต
- 8 หมายถึง TG1WDT_SYS_RESET หรือ เกิดจาก TIMER1/WDT ทำการรีเซ็ต
- 9 หมายถึง RTCWDT_SYS_RESET หรือ RTC/WDT ทำการรีเซ็ตแกนประมวลผล
- 10 หมายถึง INTRUSION_RESET หรือ มีการเรียกใช้คำสั่งเกี่ยวกับการทดสอบการรีเซ็ตหน่วยประมวลผล
- 11 หมายถึง TGWDT_CPU_RESET หรือ มีการรีเซ็ตจากไทมเมอร์/WDT
- 12 หมายถึง SW_CPU_RESET หรือซอฟต์แวร์มีการสั่งคำสั่งรีเซ็ตหน่วยประมวลผล

- 13 หมายถึง RTCWDT_CPU_RESET หรือ RTC/WDT สั่งรีเซ็ตหน่วยประมวลผล
- 14 หมายถึง EXT_CPU_RESET หรือ APP_CPU สั่งรีเซ็ต
- 15 หมายถึง RTCWDT_BROWN_OUT_RESET หรือเกิดการรีเซ็ตเพราะแหล่งจ่ายแรงดันไม่เสถียร
- 16 หมายถึง RTCWDT_RTC_RESET หรือ RTC/WDT สั่งรีเซ็ตแกนหน่วยประมวลผลและ RTC
- ตัวเลขอื่น ๆ หมายถึงไม่ทราบสาเหตุของการรีเซ็ต

ตัวอย่างโปรแกรม 1-13 เป็นการแสดงสถานะของการบูตเพื่อรายงานสถานะของการเกิดเหตุของการบูตระบบ โดยมีผลลัพธ์ของการทำงานเป็นดังภาพที่ 1-16 ซึ่งตัวอย่างนี้มีประโยชน์ในการประยุกต์ใช้สำหรับการเขียนโปรแกรมเพื่อตรวจสอบเหตุการณ์ที่เกิดขึ้นกับ ESP32 และส่งกลับมาเพื่อหาสาเหตุของปัญหาต่อไป

ตัวอย่างโปรแกรม 1-13 การแสดงสถานะของการบูต

บรรทัด	คำสั่ง
1	#include <ESP.h>
2	#include "esp32/rom/rtc.h"
3	void print_wakeup_reason() {
4	switch (esp_sleep_get_wakeup_cause()) {
5	case ESP_SLEEP_WAKEUP_EXT0:
6	Serial.println("RTC_IO"); break;
7	case ESP_SLEEP_WAKEUP_EXT1:
8	Serial.println("RTC_CNTL"); break;
9	case ESP_SLEEP_WAKEUP_TIMER:
10	Serial.println("Timer"); break;
11	case ESP_SLEEP_WAKEUP_TOUCHPAD:
12	Serial.println("Touchpad"); break;
13	case ESP_SLEEP_WAKEUP_ULP:
14	Serial.println("ULP"); break;
15	default: Serial.println("???");
16	}
17	}
18	void print_reset_reason(int cpu) {
19	switch (rtc_get_reset_reason(cpu)) {
20	case 1: Serial.println("POWERON_RESET");

ตัวอย่างโปรแกรม 1-13 การแสดงสถานะของการบูต (ต่อ)

บรรทัด	คำสั่ง
21	break;
22	case 3: Serial.println("SW_RESET"); break;
23	case 4: Serial.println("OWDT_RESET"); break;
24	case 5: Serial.println("DEEPSLEEP_RESET");
25	break;
26	case 6: Serial.println("SDIO_RESET"); break;
27	case 7: Serial.println("TG0WDT_SYS_RESET");
28	break;
29	case 8: Serial.println("TG1WDT_SYS_RESET");
30	break;
31	case 9: Serial.println("RTCWDT_SYS_RESET");
32	break;
33	case 10: Serial.println("INTRUSION_RESET");
34	break;
35	case 11: Serial.println("TGWDT_CPU_RESET");
36	break;
37	case 12: Serial.println("SW_CPU_RESET");
38	break;
39	case 13: Serial.println("RTCWDT_CPU_RESET");
40	break;
41	case 14: Serial.println("EXT_CPU_RESET");
42	break;
43	case 15:
44	Serial.println("RTCWDT_BROWN_OUT_RESET");
45	break;
46	case 16: Serial.println("RTCWDT_RTC_RESET");
47	break;
48	default: Serial.println("NO_MEAN");
49	}
50	}
51	void setup() {
52	Serial.begin(115200);
53	Serial.println("----WAKEUP?");
54	print_wakeup_reason();

ตัวอย่างโปรแกรม 1-13 การแสดงสถานะของการบูต (ต่อ)

บรรทัด	คำสั่ง
55	Serial.println("----CPU0/PRO_CPU reset?");
56	print_reset_reason(0);
57	Serial.println("----CPU1/APP_CPU reset?");
58	print_reset_reason(1);
59	Serial.println("reset in 10seconds...");
60	Serial.flush();
61	delay(10000);
62	ESP.restart();
63	}
64	void loop() {
65	}

<pre> ----WAKEUP? ??? ----CPU0/PRO_CPU reset? SW_CPU_RESET ----CPU1/APP_CPU reset? SW_CPU_RESET reset in 10seconds... </pre>
--

ภาพที่ 1-16

ตัวอย่างผลลัพธ์จากโปรแกรม 1-13

ภาพโดย จารุต บุศราทิจ

> สรุป

จากบทนี้จะพบว่า สถาปัตยกรรมของไมโครคอนโทรลเลอร์ ESP32 เป็นประเภทผสานโมดูลการทำงานหลายอย่างเข้าด้วยกัน ได้แก่ หน่วยประมวลผลกลางที่มีหน่วยความจำ ROM/RAM ในตัว ส่วนสื่อสารกับอุปกรณ์ภายนอก ส่วนงานเมื่ออยู่ในโหมดประหยัดพลังงาน หน่วยเชื่อมต่อบนระบบสื่อสารไร้สาย หน่วยแปลงสัญญาณดิจิทัล/แอนะล็อก และรองรับการต่อหน่วยความจำแรมภายนอกผ่านบัส SPI เพื่อขยายปริมาณหน่วยความจำของระบบ โดยผู้เขียนโปรแกรมจะต้องคอยสลับการใช้ข้อมูลด้วยตนเอง

ระหว่างหน่วยความจำแฟลชภายใน ESP32 กับภายนอกเพราะการประมวลผลจะใช้หน่วยความจำภายในในการทำงาน นอกจากนี้ ได้เรียนรู้เรื่องของการขัดจังหวะ โหมดประหยัดพลังงาน ตัวตั้งเวลา การใช้ระบบ และการตรวจสอบสถานะการรีบูตของระบบอันเป็นพื้นฐานและเป็นตัวเสริมการเขียนโปรแกรมเพื่อสั่งงานระบบได้อย่างเหมาะสม เช่น ถ้ามีการรีบูตเนื่องจากแหล่งจ่ายพลังงานไม่เพียงพอต่อการใช้งาน หรือแรงดันตก ผู้ออกแบบระบบจะได้แก้ปัญหาในเรื่องของการออกแบบแหล่งพลังงาน เป็นต้น

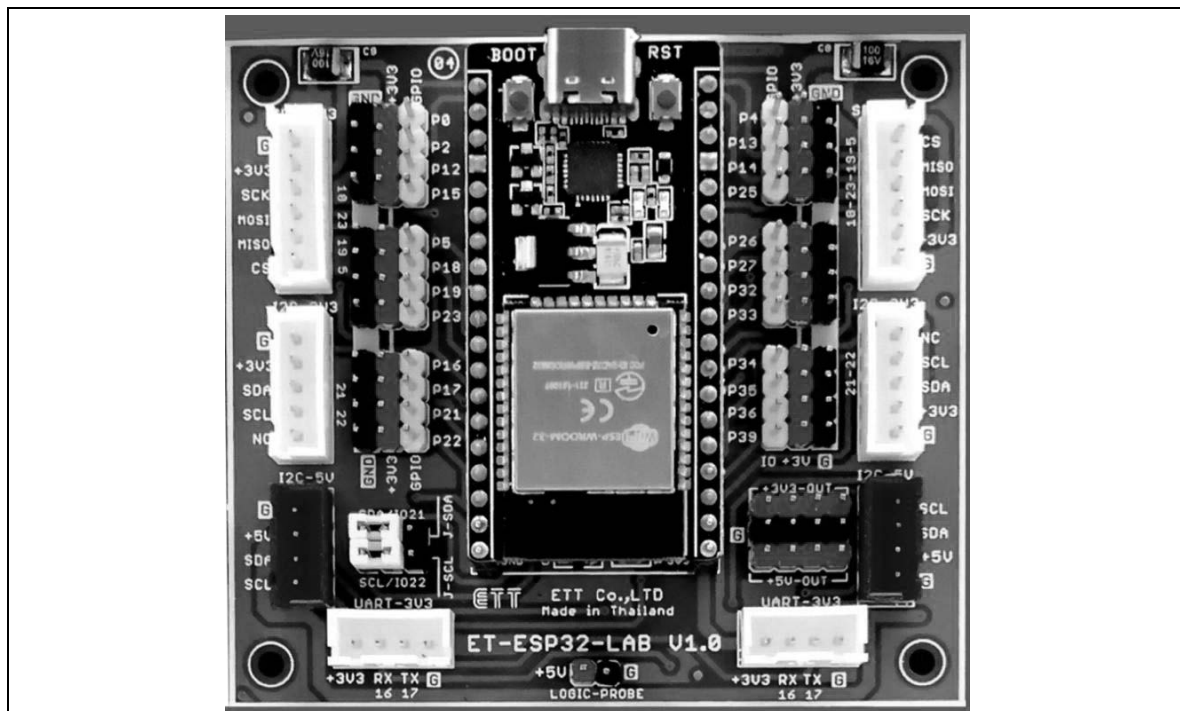
บรรณานุกรม

- Espressif Systems. (2024). *ESP32 Technical Reference Manual*. สืบค้นเมื่อ มีนาคม 18, 2567, จาก https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf
- Espressif Systems. (2024). *ESP32 Series Datasheet*. สืบค้นเมื่อ มีนาคม 18, 2567, จาก https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf



บทที่ 2 GPIO, UART และ PWM

บทนี้กล่าวถึงการใช้งานพอร์ตเชื่อมต่อกับอุปกรณ์ภายนอกไมโครคอนโทรลเลอร์ ESP32 หรือ GPIO (General Purpose I/O) เพื่อกำหนดบทบาทการทำงานของขาแต่ละขา การอ่านสถานะของขา และการกำหนดสถานะของขา อันเสมือนการนำเข้าข้อมูลและการส่งออกข้อมูล การใช้งานพอร์ตสื่อสารอนุกรมที่ทำงานภายใต้คลาส Serial ที่ใช้ในการสื่อสารกับอุปกรณ์หรือแสดงผล/รับค่า และการทำ PWM (Pulse Width Modulate) ส่งข้อมูลดิจิทัล 0 และ 1 เรียงซ้ำกันไปเรื่อย ๆ ตามความถี่ที่กำหนดขึ้นพร้อมสามารถกำหนดอัตราส่วนของสถานะ 0 หรือ 1 ได้ตามต้องการ โดยอธิบายหลักการใช้พร้อมตัวอย่างโปรแกรมเพื่อนำไปประยุกต์ใช้กับอุปกรณ์ทดลองของ ET-ESP32 LAB V1.0 บทที่ 3 ถึง 9 หรือบทอื่น ๆ ที่เกี่ยวข้อง



ภาพที่ 2-1

ภาค ESP32 ของบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ

> แนะนำ GPIO

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

GPIO เป็นหน่วยเชื่อมประสานกับอุปกรณ์ภายนอกเพื่อทำการนำออกสัญญาณ หรือเอาต์พุต (Output) กับนำเข้าสัญญาณ หรืออินพุต (Input) ซึ่งมีการจัดเรียงขา GPIO ที่ต่อเข้ากับบอร์ด ET-ESP32 LAB V1.0 ดังภาพที่ 2-1 โดยแต่ละขาของ ESP32 สามารถขับโหลดหรือส่งกระแสได้ออกได้สูงสุด 40mA ทำให้การขับกระแสออกจากขาของ ESP32 ต้องระมัดระวังกระแสที่ออกจากบอร์ดต้องพอดีกับอุปกรณ์ที่ต่อพ่วงด้วย โดยการสั่งงานให้ขาใดของ ESP32 ทำหน้าที่เป็นแบบใดจะต้องกำหนดผ่านทางคำสั่ง `pinMode()` ตามรูปแบบของการใช้งานดังนี้

```
void pinMode( หมายเลขขา, ประเภทของการเชื่อมต่อ )
```

โดยที่

หมายเลขขา คือ หมายเลขขาของ ESP32 ที่เชื่อมต่อกับอุปกรณ์ภายนอก

ประเภทของการเชื่อมต่อ คือ คำสั่งสำหรับกำหนดการทำงานของขาที่เชื่อมต่อกับอุปกรณ์ภายนอก โดยมีค่าได้ดังนี้

- INPUT สำหรับให้ขาที่ระบุทำหน้าที่นำเข้าสัญญาณหรือเป็นอินพุต
- OUTPUT สำหรับให้ขาที่ระบุทำหน้าที่นำออกสัญญาณหรือเป็นเอาต์พุต
- INPUT_PULLUP สำหรับให้ขาที่ระบุทำหน้าที่นำเข้าสัญญาณหรือเป็นอินพุตโดยที่อุปกรณ์ภายนอกที่เชื่อมต่อกับขามีวงจรพูลอัพที่ทำหน้าสถานะของขานั้นเป็น 1 อยู่ด้วย

ขาสำหรับทำหน้าที่นำเข้าด้วยการสัมผัส หรือ Capacitive Touch Sensors ของ ESP32 เป็นกลุ่มขาที่มีเซ็นเซอร์แบบสัมผัสสำหรับใช้เป็นขา Wake Up เพื่อใช้ในการให้ไมโครคอนโทรลเลอร์กลับจากการอยู่ในโหมดประหยัดพลังงาน หรือเป็นขานำเข้าสัญญาณดิจิทัลนั้นมีด้วยกัน 10 ขา เรียกว่า T0, T1, T2, T3, T4, T5, T6, T7, T8 และ T9 นั้นเป็นขามีหมายเลข P4, P0, P2, P15, P14, P12, P14, P17, P33 และ P32 ของ ESP32 ตามลำดับ

สิ่งที่ต้องระวังคือ สถานะของขาต่อไปนี้ต้องมีสถานะเป็นตามที่บริษัท espressif กำหนดเท่านั้น จึงจะทำให้การบูตหรือการเริ่มต้นทำงานของไมโครคอนโทรลเลอร์ ESP32 ทำงานได้อย่างถูกต้อง

- P1 ต้องเป็น HIGH
- P3 ต้องเป็น HIGH
- P5 ต้องเป็น HIGH ระหว่างบูต
- P6 ถึง P11 เชื่อมต่อกับหน่วยความจำแฟลช ไม่ควรนำไปใช้
- P12 จะต้องเป็น LOW ระหว่างบูต

○ P14 ต้องเป็น HIGH

○ P15 ต้องมีสถานะของขาเป็น HIGH ระหว่างบูต
การนำข้อมูลส่งไปยังขานำออกสัญญาณหรือเอาต์พุตเพื่อส่งระดับสัญญาณดิจิทัลไปให้อุปกรณ์ภายนอกหรือวงจรที่เชื่อมต่อกับขาของ ESP32 จะต้องใช้คำสั่ง digitalWrite() ตามรูปแบบของการใช้งานต่อไปนี้

digitalWrite(หมายเลขขา, ประเภทของระดับแรงดัน)

โดยที่

หมายเลขขา คือ หมายเลขขาของ ESP32 ที่เชื่อมต่อกับอุปกรณ์ภายนอก

ประเภทของระดับแรงดัน คือ ระดับแรงดันที่ต้องการนำออกไปยังขาที่ระบุ โดยค่าที่ส่งออกสามารถระบุเป็นตัวเลข 0 หรือ 1 แทนค่าดิจิทัล หรือใช้เป็นค่าคงที่ใดต่อไปนี้แทนได้

○ HIGH สำหรับแทนค่าแรงดันของสัญญาณดิจิทัล 0

○ LOW สำหรับแทนค่าแรงดันของสัญญาณดิจิทัล 1

และกรณีที่อ่านข้อมูลนำเข้าแบบดิจิทัลจากขาที่เชื่อมต่อกับอุปกรณ์ภายนอกกระทำได้ด้วยการใช้คำสั่ง digitalWrite() ตามรูปแบบของการใช้งานต่อไปนี้

ค่าที่อ่านได้ = digitalRead(หมายเลขขา)

โดยที่

หมายเลขขา คือ หมายเลขขาของ ESP32 ที่เชื่อมต่อกับอุปกรณ์ภายนอก

ค่าที่อ่านได้ คือ ระดับแรงดันหรือสถานะของขาที่ต้องการนำเข้าสัญญาณ โดยสามารถมีสถานะเป็น 0 หรือ 1 อย่างใดอย่างหนึ่ง และสามารถใช้เป็นค่าคงที่ใดต่อไปนี้แทนการมองเป็นตัวเลข 0 หรือ 1 ได้เช่นกัน

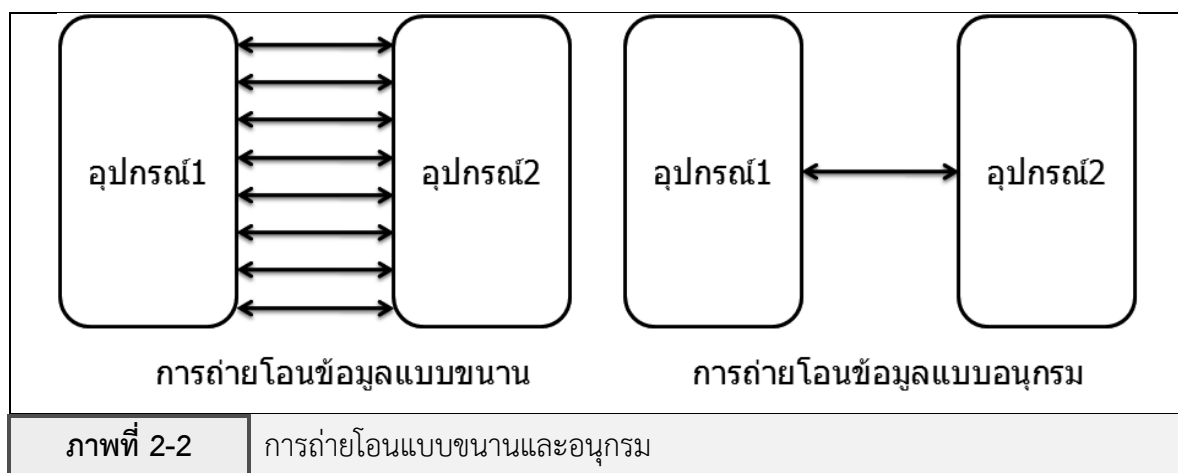
○ HIGH สำหรับแทนค่าแรงดันของสัญญาณดิจิทัล 0

○ LOW สำหรับแทนค่าแรงดันของสัญญาณดิจิทัล 1

> แนะนำ UART

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

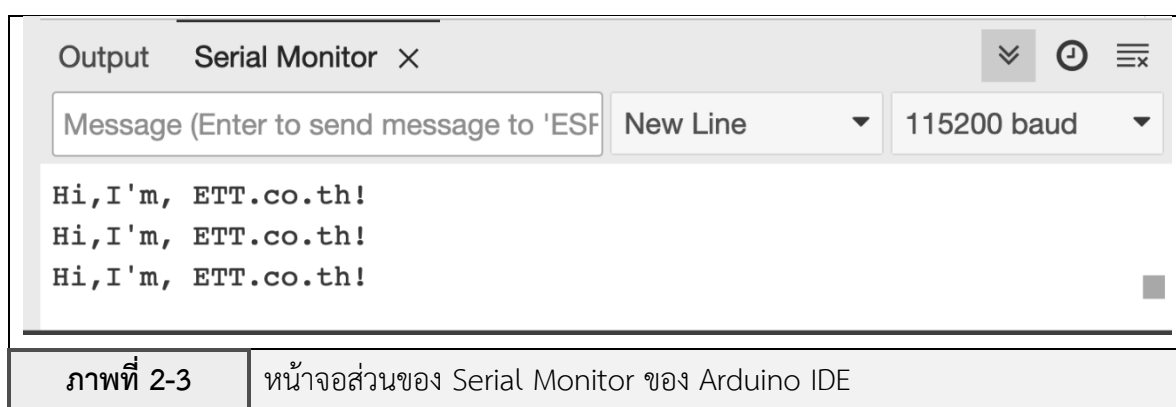
ในระบบสถาปัตยกรรมคอมพิวเตอร์มีการถ่ายโอนข้อมูล 2 วิธี คือ การถ่ายโอนแบบขนาน (Parallel Transfer) และการถ่ายโอนแบบอนุกรม (Serial Transfer) โดยวิธีการทำงานของการถ่ายโอนแบบขนานเป็นการรับ/ส่งข้อมูลครั้งละ 8 บิต ขณะที่การถ่ายโอนแบบอนุกรมนั้นมีวิธีการรับส่งที่แตกต่างกัน คือ รับ/ส่งครั้งละ 1 บิต ด้วยเหตุนี้ วิธีการถ่ายโอนข้อมูลแบบขนานมีความเร็วในการส่งต่อครั้งที่สูงกว่าการถ่ายโอนแบบอนุกรม แต่การถ่ายโอนข้อมูลแบบอนุกรมนั้นมีคุณสมบัติที่ดีด้านการส่งข้อมูลได้โดยใช้สายข้อมูลเพียงเส้นเดียว ดังภาพที่ 2-2



ภาพโดย จารุต บุศราทิจ

บอร์ด ESP32 รองรับการรับส่งข้อมูลแบบยูเออาร์ที UART ที่ทำงานกับแรงดันไฟฟ้ากระแสตรง 3.3V ทำให้สามารถสื่อสารอนุกรม RS232 อันเป็นมาตรฐานแนะนำ 232 (Recommended Standard 232) ของสมาคมผู้ประกอบการอุตสาหกรรมอิเล็กทรอนิกส์แห่งอเมริกาหรือ EIA (Electronics Industry Association) ซึ่งเป็นมาตรฐานการเชื่อมต่อข้อมูลแบบอนุกรม ด้วยสายข้อมูลแบบ 25 ขา หรือ 9 ขา ที่เรียกว่า DB25 และ DB9 (ปัจจุบันไม่ได้ใช้ขาเชื่อมต่อแบบนี้แล้ว) เพื่อทำการสื่อสารแบบอะซิงโครนัส (Asynchronous) ระหว่างอุปกรณ์ โดยที่ ฮาร์ดแวร์ UART0 ของ ESP กำหนดให้ขา P1 และ P3 ทำหน้าที่ REPL UART (สื่อสารกับระบบปฏิบัติการภายในไมโครคอนโทรลเลอร์ ESP32) ขา Tx และ Rx ดังนั้น ถ้ามีการเชื่อมต่อสื่อสารอนุกรมไม่ควรใช้ 2 ขานี้เชื่อมต่อกับวงจรภายนอก ส่วน UART1 ใช้ขา P10 และ P9 และ UART2 ใช้ขา P17 และ P16 เป็นขาทำหน้าที่ TX และ RX กับ TX2 และ RX2 แต่อย่างไรก็ดี เนื่องด้วยขาของ UART1 ถูกใช้ในการเชื่อมต่อกับโมดูลแฟลชรอมภายนอกผ่านทางบัส SPI ทำให้ไม่สามารถนำมาใช้งานได้ ดังนั้น โมดูล ESP32 จึงสามารถใช้พอร์ตสื่อสารอนุกรม UART0 และ UART2 ได้เท่านั้น

อัตราความเร็วในการสื่อสารของ RS232 มีหน่วยเป็น bps (bit per second) หรือบิตต่อวินาที ซึ่งเป็นสัดส่วนของการส่งข้อมูล n บิตในเวลา 1 วินาที ด้วยเหตุนี้ 9600bps จึงหมายถึงใน 1 วินาทีมีการส่งหรือรับข้อมูล 9600 บิต แต่ถ้าฝั่งส่งใช้ 9600bps ฝั่งรับใช้ความเร็ว 19200bps จะเกิดปัญหา คือ เมื่อส่งข้อมูลไป 1 วินาที ทางฝั่งรับจะมองว่าข้อมูลที่ส่งมานั้นมีจำนวน 19200 ข้อมูล ขณะที่ผู้ส่งส่งไปเพียง 9600 ข้อมูลเท่านั้น ด้วยเหตุนี้ ความเร็วในการสื่อสารทั้งฝั่งบอร์ดไมโครคอนโทรลเลอร์และอุปกรณ์ภายนอกจะต้องมีความเร็วที่เท่ากันทั้งฝั่งผู้ส่งและผู้รับ โดยการกำหนดอัตราความเร็วในโปรแกรม Serial Monitor ของ Arduino IDE ที่กำหนดอัตราการรับส่งเป็น 15200bps อันความเร็วมาตรฐานของบอร์ด ดังภาพที่ 2-3

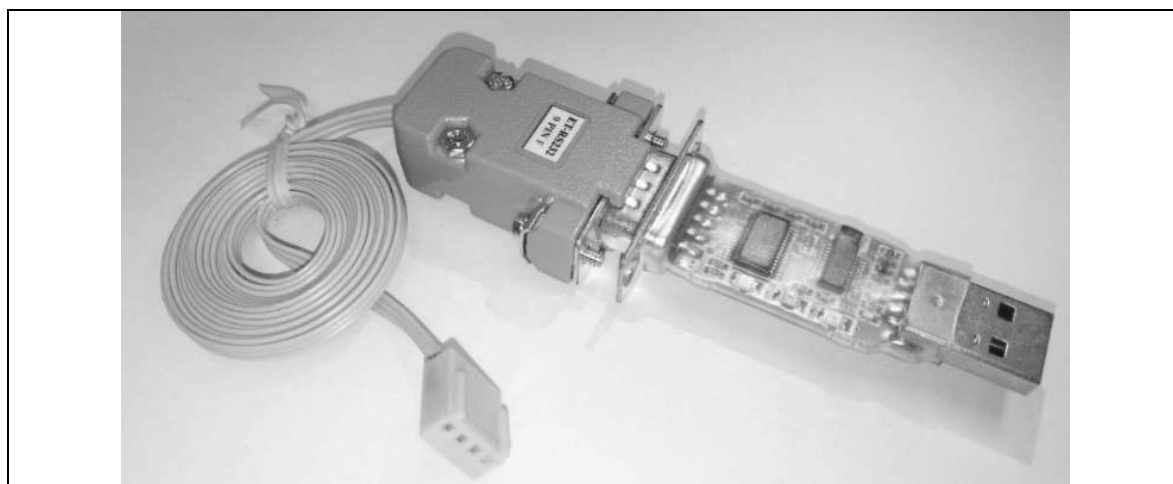


ภาพที่ 2-3

หน้าจอส่วนของ Serial Monitor ของ Arduino IDE

ภาพโดย จารุต บุศราทิจ

แม้ในปัจจุบันเครื่องคอมพิวเตอร์แบบพกพาได้ตัดพอร์ตเชื่อมต่อประเภทนี้ออกไป แต่ผู้ใช้อย่างคงสามารถหาตัวแปลงจากพอร์ตสื่อสารยูเอสบี (USB) ให้เป็น RS232 ได้ เช่น อุปกรณ์ ET-USB RS232 mini ดังภาพที่ 2-4 เป็นต้น



ภาพที่ 2-4

อุปกรณ์แปลง USB/RS232 รุ่น ET-USB RS232 mini

ภาพโดย จารุต บุศราทิจ

คลาส Serial ของเฟรมเวิร์ค Arduino เป็นการใช้พอร์ตสื่อสารอนุกรมของบอร์ดไมโครคอนโทรลเลอร์ที่ใช้สื่อสารกับอุปกรณ์ภายนอก โดยการทำงานของพอร์ตสื่อสารอนุกรมจะมีบัฟเฟอร์สำหรับเก็บข้อมูลเพื่อส่งออกและที่รับเข้า ซึ่งการใช้งานคำสั่งหรือ เมธอด (method) ต่าง ๆ เป็นดังนี้

คำสั่ง `begin()` ของคลาส Serial ใช้สำหรับกำหนดความเร็วในการสื่อสาร หรือการรับ/ส่งระหว่างอุปกรณ์ภายนอกกับบอร์ดไมโครคอนโทรลเลอร์ โดย ค่าความเร็วในการสื่อสาร ได้แก่ 300 1200 2400 4800 9600 19200 38400 57600 และ 115200 bps เป็นต้น

ค่าการทำงานโดยปกติของคำสั่ง `begin()` คือ SERIAL_8N1 อันมีความหมายถึง การสื่อสารผ่านพอร์ตอนุกรมจะมีข้อมูลจำนวน 8 บิต ไม่มีพาริตีบิต (non-parity bit) และมีบิตกำกับการหยุด 1 บิต (1-stop bit) ซึ่งตั้งค่าตรงกับกับอุปกรณ์ภายนอกที่เชื่อมต่อสื่อสารกับบอร์ดไมโครคอนโทรลเลอร์

`Serial.begin( ความเร็วในการสื่อสาร )`

โดยที่

ความเร็วในการสื่อสาร คือ อัตราการสื่อสารในหน่วย bps อันได้แก่ 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 250000, 460800, 500000, 921600, 1000000 และ 2000000

เมื่อสิ้นสุดการสื่อสารควรทำการปิดการเชื่อมต่อสื่อสารด้วยคำสั่ง `end()` ตามรูปแบบต่อไปนี้

`Serial.end( )`

ในบางครั้งบอร์ดไมโครคอนโทรลเลอร์อาจไม่มีการเชื่อมต่อสื่อสารอนุกรมกับอุปกรณ์ภายนอก ด้วยเหตุนี้ ถ้าโปรแกรมที่พัฒนาขึ้นจำเป็นต้องการสื่อสารกับอุปกรณ์ภายนอกจะส่งผลให้เกิดความผิดพลาด แต่ในการทำงานจริงจะไม่สามารถมองเห็นความผิดพลาดของการทำงานประเภทนี้ได้ ด้วยเหตุนี้ ผู้เขียนโปรแกรมจำเป็นต้องตรวจสอบความพร้อมของการสื่อสารก่อนเสมอ คำสั่งสำหรับตรวจสอบพอร์ต หรือ การเชื่อมต่อสื่อสารพอร์ตอนุกรมมีรูปแบบให้ใช้งานดังบรรทัดที่ 3 ของโปรแกรมตัวอย่าง ซึ่งเป็นการวนรอบจนกว่าพอร์ตสื่อสารจะพร้อมใช้งาน ดังตัวอย่างโปรแกรม 2-1

ตัวอย่างโปรแกรม 2-1 โปรแกรมตรวจสอบความพร้อมของพอร์ต UART

บรรทัด	คำสั่ง
1	void setup() {
2	Serial.begin(115200);
3	while (!Serial);
4	Serial.println("Ready!");
5	}
6	void loop() {
7	}

การส่งข้อมูลออก

คำสั่งสำหรับส่งออกข้อมูลด้วยคำสั่ง write() มีรูปแบบการใช้งานดังนี้ โดยความแตกต่างของคำสั่งทั้ง 3 คือ แบบแรกส่งไบต์ของข้อมูลที่มีค่า ข้อมูลตัวเลข ออกไป แบบที่ 2 ทำการส่ง ข้อมูลข้อความ หรือที่เรียกว่าเป็น "สตริง" หรือ "สายอักขระ" ออกไป และแบบที่ 3 เป็นการส่งข้อมูลจากตัวแปรแถวลำดับที่เก็บในตัวแปร buf จำนวน len ไบต์

```
Serial.write( ข้อมูลตัวเลข )
Serial.write( ข้อมูลข้อความ )
Serial.write( buf, len )
```

กรณีที่ต้องการนำออกข้อมูลในลักษณะของสตริงที่เกิดจากการแปลงข้อมูลต่าง ๆ ให้อยู่ในรูปแบบสายอักขระประเภทไบต์ที่จัดเรียงกันในหน่วยความจำและปิดท้ายด้วยอักขระรหัส 0 สำหรับเป็นตัวบ่งบอกให้ทราบว่าสิ้นสุดของข้อมูลสตริงจะต้องใช้คำสั่ง print() หรือ println() ในรูปแบบต่อไปนี้

```
Serial.print( ข้อมูล )
Serial.println( ข้อมูล )
Serial.print( ข้อมูล, ฐาน )
Serial.println( ข้อมูล, ฐาน )
Serial.print( ตัวเลขทศนิยม, จำนวนหลักหลังจุดทศนิยม )
Serial.println( ตัวเลขทศนิยม, จำนวนหลักหลังจุดทศนิยม )
```


คำสั่ง `print()` และ `println()` ใช้สำหรับนำ ข้อมูล ส่งออกไปที่พอร์ตสื่อสารอนุกรมด้วยอัตราความเร็วที่กำหนดจากคำสั่ง `begin()` โดยคำสั่ง `print()` เมื่อนำข้อมูลส่งออกจนครบทุกตัวอักษรที่เก็บในสตริงที่ถูกแปลงจะถือว่าเป็นการจบการทำงาน แต่ `println()` จะส่งรหัสขึ้นบรรทัดใหม่ต่อจากสตริงที่ส่งจนครบ ทำให้เกิดผลเป็นขึ้นบรรทัดใหม่

ผู้เขียนโปรแกรมสามารถกำหนดเลขฐานของตัวเลขได้สำหรับในกรณีที่ ข้อมูล นั้นเป็นจำนวนเต็ม โดยกำหนด ฐาน เป็นตัวเลขประเภทของเลขฐานของข้อมูลที่ต้องการแสดง ซึ่งมีค่าคงที่เป็นดังนี้

- DEC สำหรับเลขฐาน 10
- OCT สำหรับเลขฐาน 8
- HEX สำหรับเลขฐาน 16

สำหรับกรณีที่ใช้กับข้อมูลแบบตัวเลขทศนิยมจะต้องกำหนดจำนวนหลักของตัวเลขหลังจุดทศนิยมได้ด้วยการกำหนดตัวเลขจำนวนเต็มให้กับอาร์กิวเมนต์หรือพารามิเตอร์ จำนวนหลักหลังจุดทศนิยม เช่น ต้องการแสดงเลขทศนิยม 9.87654321 แบบจำนวนหลักหลังจุดทศนิยมเป็น 0, 1, 2, 3 และ 4 จะเขียนโค้ดได้ดังตัวอย่างโปรแกรม 2-2 โดยผลลัพธ์ที่ได้ คือ

```
10
9.9
9.88
9.877
```

ตัวอย่างโปรแกรม 2-2 ตัวอย่างการใช้ `println()` กับตัวเลขทศนิยม

บรรทัด	คำสั่ง
1	<code>void setup() {</code>
2	<code> Serial.begin(115200);</code>
3	<code> while (!Serial);</code>
4	<code>}</code>
5	<code>void loop() {</code>
6	<code> Serial.println(9.87654321, 0);</code>
7	<code> Serial.println(9.87654321, 1);</code>
8	<code> Serial.println(9.87654321, 2);</code>
9	<code> Serial.println(9.87654321, 3);</code>
10	<code> delay(5000)</code>
11	<code>}</code>

คำสั่งสำหรับสั่งให้นำข้อมูลออกจากบัฟเฟอร์ให้หมดโดยไม่รอการรับข้อมูลเข้าด้วยการสั่ง flush() อันมีรูปแบบดังนี้

```
Serial.flush( )
```

การอ่านข้อมูล

การรับข้อมูลนำเข้าจากพอร์ตสื่อสารอนุกรมมีคำสั่งแตกต่างกันตามวัตถุประสงค์ของการใช้งาน ได้แก่ read(), readBytes(), readBytesUntil(), readString() และ readStringUntils() ดังรูปแบบของการใช้ต่อไปนี้

```
ตัวแปร = Serial.read()

Serial.readBytes( ตัวแปร, จำนวนข้อมูล )

Serial.readBytesUntil( character, ตัวแปรแถวลำดับ, จำนวนข้อมูล )

ตัวแปร = Serial.readString();

ตัวแปร = Serial.readStringUntil( terminator )
```

คำสั่ง read() ใช้สำหรับนำข้อมูลจากบัฟเฟอร์ขนาด 1 ไบต์ของพอร์ตสื่อสารอนุกรมมาเก็บในตัวแปร

คำสั่ง readBytes() ใช้สำหรับอ่านข้อมูลจำนวน n ข้อมูล ตามที่กำหนดในพารามิเตอร์ จำนวนข้อมูล มาเก็บในตัวแปร

คำสั่ง readBytesUntil() ใช้สำหรับทำการอ่านจากสตรีมจำนวน n ไบต์ หรือจนกว่าจะพบ character ไปเก็บในตัวแปร ซึ่งเป็นตัวแปรประเภทแถวลำดับ

คำสั่ง readString() ใช้สำหรับอ่านข้อมูลมาเก็บในตัวแปร ที่เป็นตัวแปรประเภทสตริง โดยข้อมูลที่อ่านมาจะสิ้นสุดด้วยรหัส '\0'

คำสั่ง readStringUntil() ใช้สำหรับการอ่านสตริงหรือจนกว่าจะพบอักขระที่กำหนดใน terminator โดยคำสั่งจะไม่นำตัวข้อมูลที่เป็น terminator เก็บลงใน ตัวแปร แต่ใช้เป็นตัวอักขระสำหรับบอกการสิ้นสุดของการรับค่า โดยที่ไม่นำอักขระนี้เก็บลงบัฟเฟอร์

คำสั่งสำหรับแปลงข้อมูลที่อ่านจากบัฟเฟอร์และแปลงประเภทของข้อมูลให้เป็นตัวเลขจำนวนเต็ม และทศนิยมมีรูปแบบดังนี้

```
ผลลัพธ์ = Serial.parseInt()
```

```
ผลลัพธ์ = Serial.parseFloat()
```

และกรณีที่ต้องการอ่านข้อมูลไบต์ถัดไปของบัฟเฟอร์โดยไม่ต้องนำข้อมูลออกจากบัฟเฟอร์ของ ส่วนสื่อสารข้อมูลอนุกรมของ ESP32 มาเก็บใน ผลลัพธ์ นั้นมีรูปแบบของการใช้งานดังนี้

```
ผลลัพธ์ = Serial.seek()
```

การตรวจสอบข้อมูลที่เก็บในบัฟเฟอร์ของพอร์ตสื่อสาร

กรณีที่ต้องการทราบจำนวนข้อมูลที่ถูกเก็บอยู่ในบัฟเฟอร์ของพอร์ตสื่อสารอนุกรมสามารถใช้ คำสั่ง `available()` จะทำการตรวจสอบบัฟเฟอร์ของพอร์ตสื่อสารอนุกรม และรายงานจำนวนไบต์ของ ข้อมูลที่ถูกจัดเก็บกลับมา

```
จำนวนข้อมูล = Serial.available()
```

และในกรณีที่ต้องตรวจสอบจำนวนข้อมูลที่เก็บในบัฟเฟอร์ที่เตรียมนำออกจากพอร์ตสื่อสาร อนุกรมต้องใช้คำสั่งดังรูปแบบต่อไปนี้

```
จำนวนข้อมูล = Serial.availableForWrite()
```

การค้นหาข้อมูล `target` ในบัฟเฟอร์ของพอร์ตสื่อสารอนุกรมสามารถใช้คำสั่ง `find()` ในการ สั่งงาน โดย ผลลัพธ์ ของการทำงานของคำสั่งจะเป็น `TRUE` เมื่อพบ `target` ในบัฟเฟอร์ และเป็น `FALSE` เมื่อไม่พบ `target` ในบัฟเฟอร์

```
ผลลัพธ์ = Serial.find( target )
```

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

สำหรับกรณีที่ต้องการค้นหา **target** จากพอร์ตอนุกรมแบบสตรีมจนกว่าจะพบ **terminal** มีรูปแบบการใช้งานดังนี้ โดยผลลัพธ์ จะมีค่าเป็น TRUE เมื่อพบ **target** และ FALSE เมื่อเกิด time-out

```
ผลลัพธ์ = Serial.find( target, terminal )
```

การตั้งค่าระยะเวลาของการสื่อสาร

ในบางกรณีที่ต้องการควบคุมระยะเวลาการรอการสื่อสาร หรือค่า Time-out อันมีหน่วยเวลาเป็นมิลลิวินาทีเพื่อไม่ให้โปรแกรมเกิดการรอนานเกินไป ผู้เขียนโปรแกรมสามารถตั้งค่าการรอการสื่อสารของพอร์ตสื่อสารอนุกรม โดยปกติตั้งค่าไว้ 1000 มิลลิวินาที ซึ่งค่าช่วงเวลานี้มีผลกับคำสั่งที่เข้าถึงบัฟเฟอร์การสื่อสารอนุกรม อันได้แก่ find(), findUntil(), parseInt(), parseFloat(), readBytes(), readBytesUntil(), readString() และ readStringUntil() ด้วยการระบุ จำนวนมิลลิวินาที สำหรับการรอคอยดังนี้

```
void Serial.setTimeout( จำนวนมิลลิวินาที )
```

การสื่อสารอนุกรมแบบซอฟต์แวร์

บางครั้งจำนวนของพอร์ตสื่อสารมีไม่เพียงพอทำให้ผู้เขียนโปรแกรมต้องเขียนโปรแกรมจำลองการทำงานของการทำงานการสื่อสารแบบพอร์ตอนุกรมขึ้นมาใช้งานเอง แต่ด้วยเฟรมเวิร์คของ Arduino ที่เตรียมไลบรารี **SoftwareSerial** เป็นคลาสสำหรับสื่อสารแบบอนุกรมด้วยไลบรารีที่เป็นซอฟต์แวร์ และมีการใช้หน่วยความจำจากแรมจำนวน 64 ไบต์สำหรับใช้เป็นบัฟเฟอร์ พร้อมทั้งระบุขาสำหรับทำหน้าที่เป็น TX และ RX ด้วยความเร็วสูงสุดในการสื่อสารอยู่ที่ 115200bps โดยการเรียกใช้ต้องนำเข้าไปไฟล์ส่วนหัว **SoftwareSerial.h** ดังรูปแบบการเขียนโค้ด ดังนี้

```
#include <SoftwareSerial.h>
```

คำสั่งสำหรับสร้างวัตถุประเภท SoftwareSerial เพื่อใช้งานต้องกำหนดค่าหน้าที่ขาสำหรับเป็นขา RX และ TX ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

SoftwareSerial ชื่อวัตถุ(ขาสำหรับทำหน้าที่RX, ขาสำหรับทำหน้าที่TX)

ส่วนการใช้งานคำสั่งอื่น ๆ สามารถทำได้เหมือนการใช้ไลบรารีจากคลาส Serial แต่ให้ใช้ชื่อของชื่อวัตถุ แทนชื่อคลาส Serial เท่านั้น และตัวบอร์ด ET-ESP32 LAB V1.0 มีขั้วต่อรองรับการสื่อสารพอร์ตหลักจากช่องเสียบ USB-C ของโมดูล ESP32 และมีส่วนขยาย RX2 และ TX2 เรียกว่า UART-3V3 สำหรับการสื่อสารเพิ่มเติมที่ด้านล่างของโมดูล ดังภาพที่ 2-5

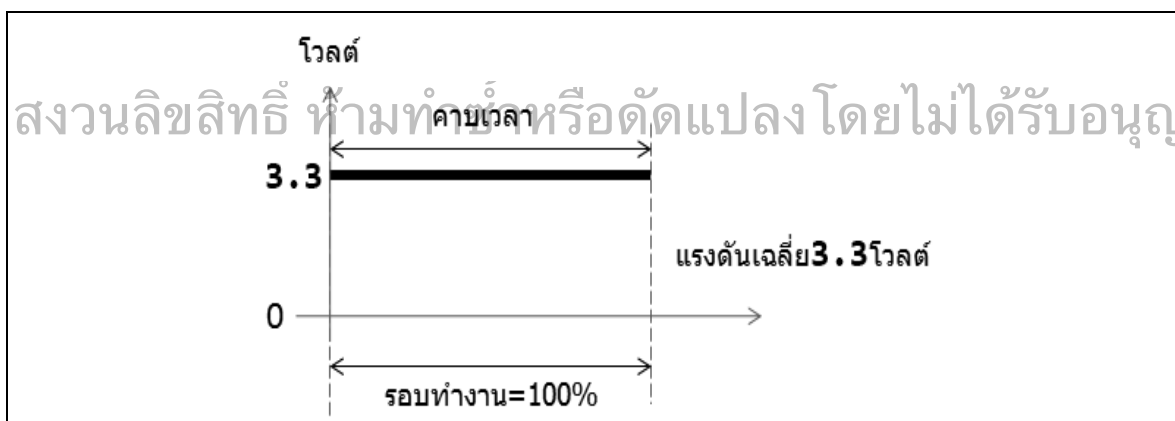


ภาพที่ 2-5 ขั้วต่อ UART-3V3 บนบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ

> แนะนำ PWM

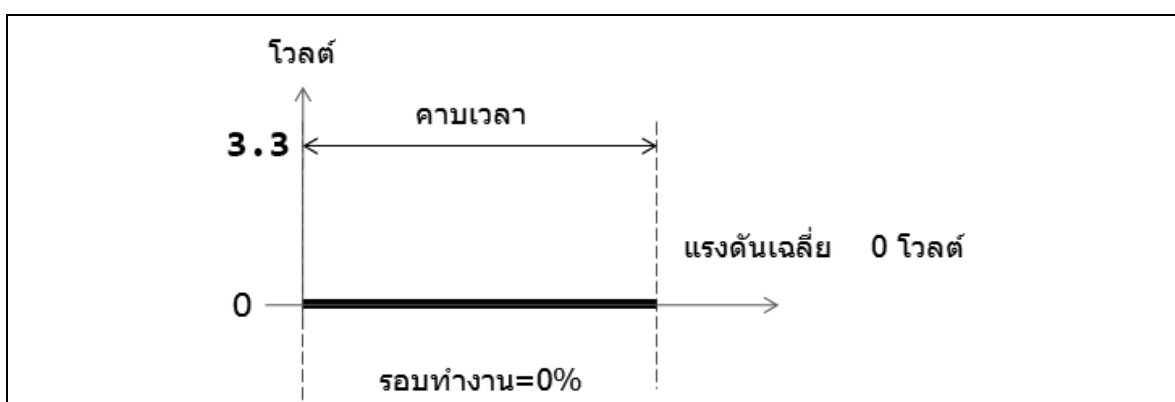
พีดีบีแอลเอ็ม หรือ PWM คือ กระบวนการกำหนดค่าดิวตีไซเคิล หรือค่ารอบการทำงานเพื่อระบุสัดส่วนของแรงดันที่เป็นสถานะ 1 ต่อแรงดันที่เป็นสถานะ 0 ในแต่ละพัลส์ ซึ่งมักถูกนำไปประยุกต์ใช้กับการปรับแรงดันด้วยพัลส์ ตัวอย่างเช่น ถ้ากำหนดค่ารอบการทำงานเป็นร้อยละ 100 ให้แก่ขานำสัญญาณออกของ ESP32 ดังภาพที่ 2-6 จะทำให้แรงดันเฉลี่ยที่ขานำสัญญาณออกมีค่าเป็น 3.3 โวลต์ ถ้ากำหนดค่ารอบการทำงานเป็นร้อยละ 0 ให้แก่ขานำสัญญาณออกของ ESP32 ดังภาพที่ 2-7 จะทำให้แรงดันเฉลี่ยที่ขานำสัญญาณออกมีค่าเป็น 0 โวลต์ และถ้ากำหนดค่ารอบการทำงานเป็นร้อยละ 50 ให้ขานำสัญญาณออกของ ESP32 ดังภาพที่ 2-8 จะทำให้แรงดันเฉลี่ยที่ขานำสัญญาณออกมีค่าเป็น 1.65 โวลต์



ภาพที่ 2-6

กราฟแรงดันเฉลี่ยเมื่อค่ารอบการทำงานเป็นร้อยละ 100

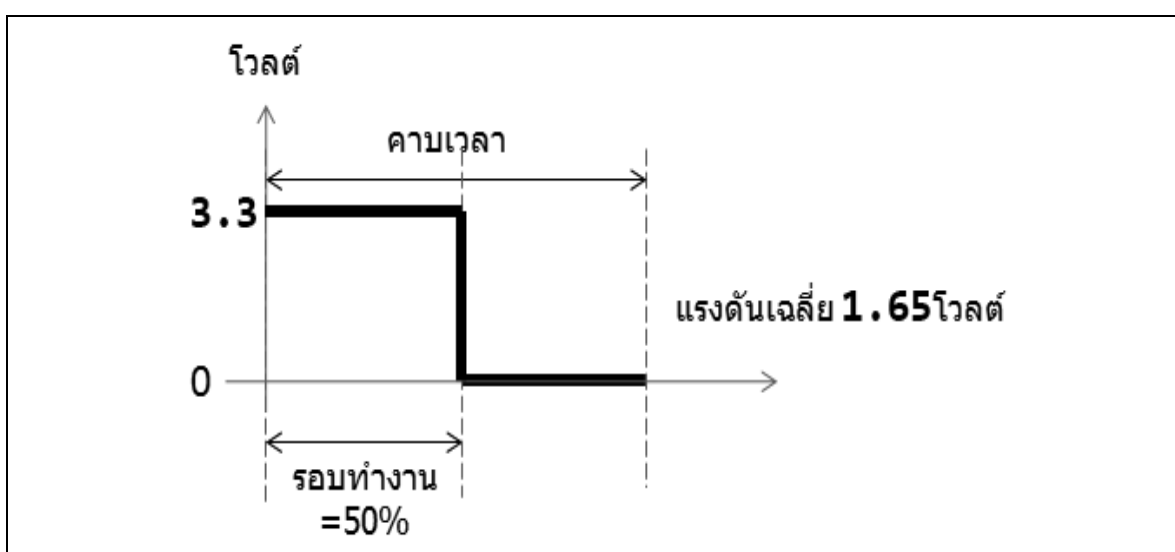
ภาพโดย จารุต บุศราทิจ



ภาพที่ 2-7

กราฟแรงดันเฉลี่ยเมื่อค่ารอบการทำงานเป็นร้อยละ 0

ภาพโดย จารุต บุศราทิจ



ภาพที่ 2-8

กราฟแรงดันเฉลี่ยเมื่อค่ารอบการทำงานเป็นร้อยละ 50

ภาพโดย จารุต บุศราทิจ

โดย ค่าดิจิทัลนำออก ต้องคำนวณจากสัดส่วนของการแปลงค่า เช่น ต้องการนำออก 3.3V ต้องส่งค่า 1023 ถ้าต้องการนำออก 0V ต้องส่งค่า 0 หรือต้องการนำออก 1.65V ต้องส่งค่า 511 เป็นต้น ดังนั้นค่าที่ใช้ในการนำออกจึงเป็นตามสมการที่ 2-1

$$\text{ค่าดิจิทัลนำออก} = \max\left(\left(v_{\text{analog}} \times \frac{1024}{3.3}\right) - 1, 0\right) \quad \text{สมการที่ 2-1}$$

ขาที่ทำหน้าที่ PWM ของไมโครคอนโทรลเลอร์ตระกูล ESP32 สามารถใช้ได้กับทุกขาที่เปิดให้ใช้งาน โดยการกำหนดค่าตัวที่อยู่ในค่า 0 ถึง 1023 และความถี่ที่สามารถกำหนดได้อยู่ในย่าน 1Hz ถึง 40MHz รองรับการสร้าง PWM ได้ 16 ช่องสัญญาณ (channels) โดยมี 8 ช่องสัญญาณที่มีค่าความถี่แตกต่างกันได้ แต่อีก 8 ช่องสัญญาณจะต้องมีค่าความถี่เดียวกัน

การใช้งาน PWM ในเฟรมเวิร์ค Arduino ของไมโครคอนโทรลเลอร์ ESP32 นั้นประกอบด้วย 3 ขั้นตอน คือ

- ตั้งค่าเพื่อกำหนดค่าความถี่และความละเอียดของค่าตัว
- เชื่อมการตั้งค่าเข้ากับขานำออกสัญญาณ
- ส่งค่าตัวไปที่ขานำออกสัญญาณ

จากขั้นตอนทั้ง 3 นี้มีคำสั่งสำหรับใช้งานและรายละเอียดคำสั่งตามหัวข้อต่อไปนี้

การตั้งค่า

วิธีการตั้งค่าให้กับ PWM ต้องกำหนดผ่านคำสั่ง `ledcSetup()` เพื่อกำหนดให้หมายเลขช่องสัญญาณ ที่ต้องการนั้นสร้าง ความถี่ ที่กำหนด และค่า ความละเอียด ของช่วงค่าของค่าตัวที่ใช้เคลดังนี้

`ledcSetup( หมายเลขช่องสัญญาณ, ความถี่, ความละเอียด )`

การตั้งค่านีส่งผลต่อความถี่ที่ส่งให้กับอุปกรณ์ที่ใช้ช่องสัญญาณ ทำให้สามารถประยุกต์กับการกำหนดค่าความถี่เป็นความถี่เดียวกับตัวโน้ตดนตรีจะส่งผลให้เกิดเสียงตามตัวโน้ตนั้น ค่าความละเอียดนั้นมีค่าได้ในช่วง 1 บิต ถึง 16 บิต ขึ้นอยู่กับความต้องการของผู้เขียนโปรแกรมว่าต้องการควบคุมความละเอียดของการกำหนดค่าตัวที่ใช้เคลในระดับใด เช่น ถ้ากำหนดเป็น 8 บิตจะสามารถกำหนดค่าตัวได้เป็น 0 ถึง 255 และถ้ากำหนดเป็น 12 บิตจะสามารถกำหนดช่วงค่าของตัวได้้อยู่ในช่วง 0 ถึง 4095 เป็นต้น

กรณีที่ต้องเปลี่ยนค่าความถี่ของ PWM โดยไม่ต้องการตั้งค่าใหม่จะต้องใช้คำสั่ง `ledcChangeFrequency()` ตามด้วย หมายเลขช่องสัญญาณ, ค่า ความถี่ ใหม่ และค่าความละเอียด ดังนี้

`ledcChangeFrequency( หมายเลขช่องสัญญาณ, ความถี่, ความละเอียด )`

การกำหนดขา

คำสั่งสำหรับกำหนดให้ขาใดทำหน้าที่ PWM ต้องใช้คำสั่ง `ledcAttachPin()` โดยกำหนด หมายเลขขา และ หมายเลขช่องสัญญาณ ที่เลือกใช้งานดังรูปแบบต่อไปนี้

`ledcAttachPin( หมายเลขขา, หมายเลขช่องสัญญาณ )`

การส่งข้อมูลไปที่ขาของ PWM

การส่งค่าดิจิทัลให้กับขาที่กำหนดหน้าที่ไว้ต้องใช้งานผ่านทางคำสั่ง `ledcWrite()` โดยระบุขาที่ต้องการนำออกค่าและค่าดิจิทัลที่ต้องการดังนี้

`ledcWrite( หมายเลขช่องสัญญาณ, ค่าดิจิทัล )`

จากคำสั่งนี้จะพบว่า การกำหนดค่า ค่าดิจิทัล ทำให้ความดังของเสียงนั้นมากหรือน้อยเนื่องจากการส่งช่วงที่เป็น 1 นานจะส่งผลให้แรงดันมากกว่า เมื่อแรงดันมากกว่าจึงผลให้เสียงที่ดังขึ้น นอกจากการนำไปประยุกต์เรื่องของเสียงแล้ว การตั้งค่าความถี่เป็น 50Hz สามารถเปลี่ยนค่าดิจิทัลเพื่อควบคุมตำแหน่งของแขนของเซอร์โวมอเตอร์ได้อีกด้วย

ตัวอย่างโปรแกรม 2-3 การสร้าง PWM แบบ 50Hz ในช่องสัญญาณหมายเลข 0 แบบความละเอียด 12 บิต ตามบรรทัดคำสั่งที่ 4 เพื่อส่งสัญญาณไปยังขา P25 หรือช่องสัญญาณหมายเลข 0 ดังในบรรทัดที่ 5 ด้วยค่าดิจิทัล 50% หรือครึ่งหนึ่งของ 4096 ในบรรทัดที่ 6 สามารถเขียนได้ดังนี้

ตัวอย่างโปรแกรม 2-3 ตัวอย่างการใช้ PWM สร้างความถี่ 50Hz

บรรทัด	คำสั่ง
1	void setup() {
2	pinMode(25, OUTPUT);
3	ledcSetup(0, 50, 12);
4	ledcAttachPin(25, 0);
5	ledcWrite(0, (uint32_t)(4096 / 2));
6	}
7	void loop() {
8	}

สร้างโทนหรือตัวโน้ต

นอกจากนี้ ชุดเฟรมเวิร์คได้เตรียมชุดคำสั่ง ledcWriteTone() สำหรับสร้างความถี่ที่มีค่าตัวตีสไซเคิล 50% และ ledcWriteNote() สำหรับสร้างความถี่เสียงตัวโน้ตดนตรีที่มีค่าออกเตฟ (octave) ตามที่กำหนดได้ดังรูปแบบการใช้งานต่อไปนี้

ผลลัพธ์ = ledcWriteTone(หมายเลขช่องสัญญาณ, ค่าความถี่)

ผลลัพธ์ = ledcWriteNote(หมายเลขช่องสัญญาณ, ชื่อตัวโน้ต, ค่าออกเตฟ)

โดยที่

ผลลัพธ์ เป็นค่าคืนกลับจากการทำงาน หากผิดพลาดจะได้ผลลัพธ์เป็น 0

หมายเลขช่องสัญญาณ เป็นค่าช่องสัญญาณ PWM

ค่าความถี่ เป็นค่าตัวเลขจำนวนเต็มของความถี่ที่ต้องการสร้าง

ชื่อตัวโน้ต เป็นค่าคงที่สำหรับแทนตัวโน้ต C, C#, D, Eb, E, F, F#, G, G#, A, Bb และ B ดังนี้

- NOTE_C
- NOTE_Cs
- NOTE_D
- NOTE_Eb
- NOTE_E
- NOTE_F
- NOTE_Fs

- NOTE_G

- NOTE_Gs

- NOTE_Bb

- NOTE_B

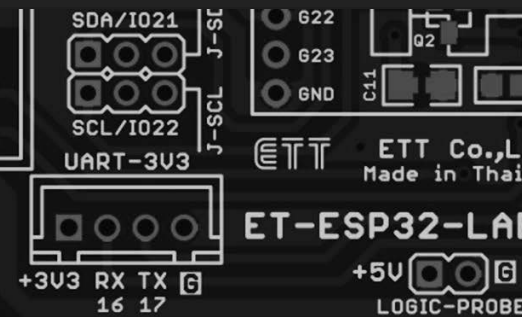
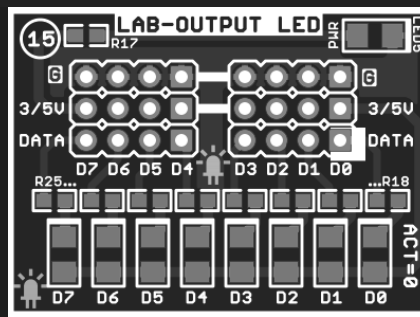
ค่าออกเตฟ เป็นค่าออกเตฟของตัวโน้ต เช่น ถ้าต้องการเล่น C4 ต้องกำหนดให้ค่าพารามิเตอร์นี้เป็น 4 หรือต้องการเล่น C3 จะต้องกำหนดให้เป็น 3 เป็นต้น

> สรุป

การกำหนดการทำงานของขา GPIO จะต้องใช้คำสั่ง `pinMode()` พร้อมระบุหน้าที่นำเข้าหรือส่งออกสัญญาณ ขณะที่การสื่อสารพอร์ตอนุกรมสามารถใช้งานได้จากขั้ว USB-C ของบอร์ด ESP32 เพื่อรับหรือส่งข้อมูลกับอุปกรณ์ภายนอกหรือเครื่องคอมพิวเตอร์ผ่านทาง Serial Monitor ในโปรแกรม Arduino IDE ด้วยการใช้งานผ่านคลาส Serial ของเฟรมเวิร์ค และการควบคุมการส่งสัญญาณแบบ PWM ของ ESP32 สามารถกำหนดค่าดิวตีไซเคิลเพื่อปรับสัดส่วนของการเป็นสถานะ 1 กับ 0 พร้อมทั้งสามารถประยุกต์ใช้คำสั่งเพื่อส่งคลื่นความถี่ของโน้ตดนตรีที่ควบคุมค่าออกเตฟได้อีกด้วย

บรรณานุกรม

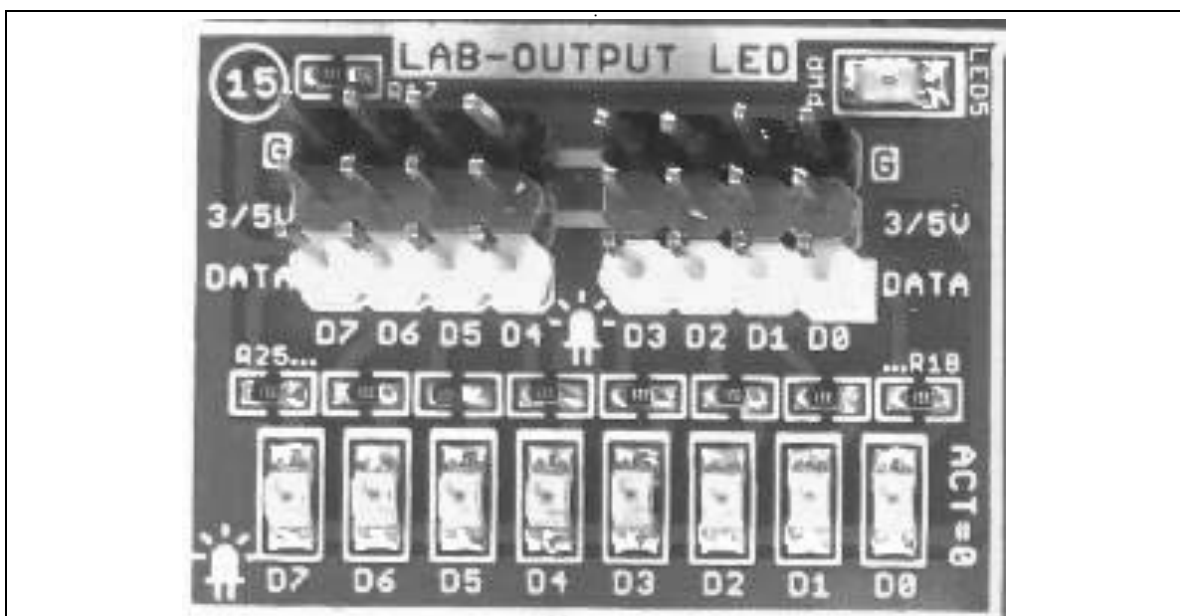
Espressif Systems. (2024). *ESP32 Series Datasheet*. สืบค้นเมื่อ มีนาคม 24, 2567, จาก https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf



บทที่ 3 LAB-OUTPUT LED

ในบทนี้เป็นการเรียนรู้เกี่ยวกับอุปกรณ์แปลงแสงแบบไดโอดที่เรียกว่าหลอดแอลอีดี (LED: Light Emitted Diode) ของภาค LAB-OUTPUT LED ดังภาพที่ 3-1 ซึ่งเป็นอุปกรณ์พื้นฐานที่มีการนำไปประยุกต์ใช้มากมาย เช่น การต่อเป็นไฟวิ่ง การต่อให้เป็นวงกลมหลายวงแล้วสั่งให้แต่ละวงแปลงแสงและดับทีละวง หรือนำไปใช้กับหลอดขนาดใหญ่หลายหลอดเพื่อให้แสงที่สว่างขึ้น เช่น หลอดแอลอีดีของไฟจราจร ไฟท้ายรถ หรือไฟเลี้ยวของรถ เป็นต้น โดยเนื้อหาบทนี้กล่าวถึง หลักการทำงานของหลอดแอลอีดี วิธีการสั่งงาน และตัวอย่างการเขียนโปรแกรมเพื่อใช้งาน

เมื่อจบบทนี้ ผู้อ่านจะเข้าใจถึงขั้นตอนการเขียนโปรแกรม และวิธีการเขียนโปรแกรมเพื่อส่งสัญญาณดิจิทัลเพื่อเปิดหรือปิดหลอดแอลอีดีของ LAB-OUTPUT LED ไปประยุกต์ใช้ต่อไป



ภาพที่ 3-1

ภาพโมดูล LAB-OUTPUT LED

ภาพโดย จารุต บุศราทิจ

> แนะนำ LED

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

โมดูล LAB-OUTPUT LED เป็นโมดูลนำสัญญาณออกหรือเอาต์พุตเป็นหลอดแอลอีดีจำนวน 8 ดวง ที่ถูกเชื่อมต่อเข้ากับขา D0, D1, D2, D3, D4, D5, D6 และ D7 พร้อมมีขั้วสำหรับจ่ายแรงดันไฟฟ้า กระแสตรงและกราวด์ ดังภาพที่ 3-1

หลอดแอลอีดีหรือไดโอดเปล่งแสงเป็นอุปกรณ์อิเล็กทรอนิกส์ประเภทไดโอดที่เปล่งแสงได้ขณะที่นำกระแส มีหลายขนาดและหลากสี ดังภาพที่ 3-2 ภาพซ้าย และใช้สัญลักษณ์ในวงจรเป็นดังภาพที่ 3-2 ด้านขวา

หลอดแอลอีดีจะมีขา 2 ขา เรียกว่า แอโนด (Anode) และแคโทด (Cathode) โดยการใช้งาน หลอดแอลอีดีจะต้องต่อขาแอโนดเข้ากับขั้วบวกและต่อขาแคโทดเข้ากับขั้วลบหรือสายดินหรือกราวนด์

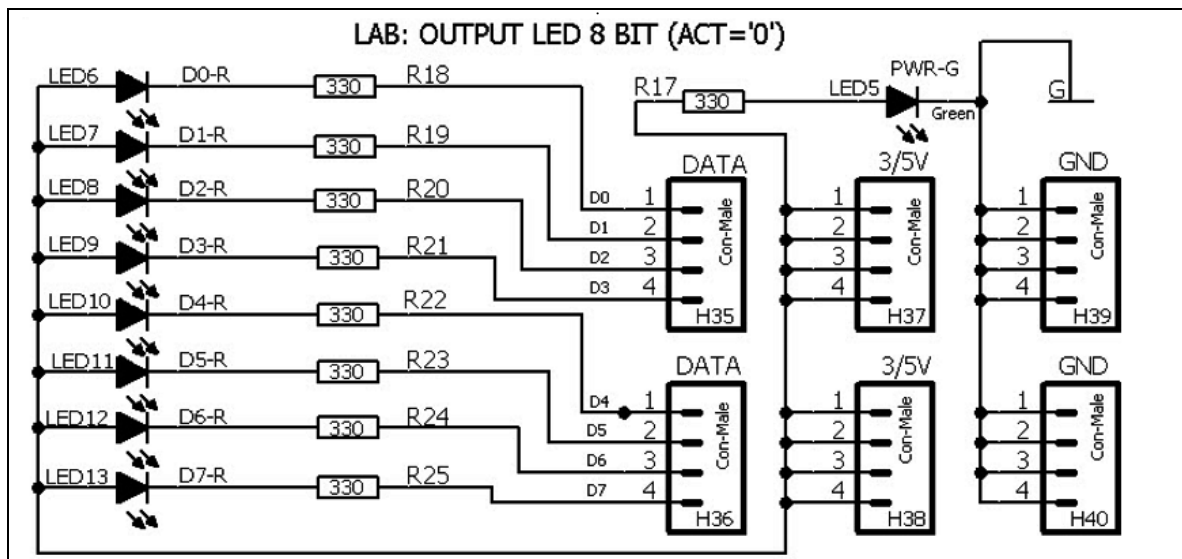


ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

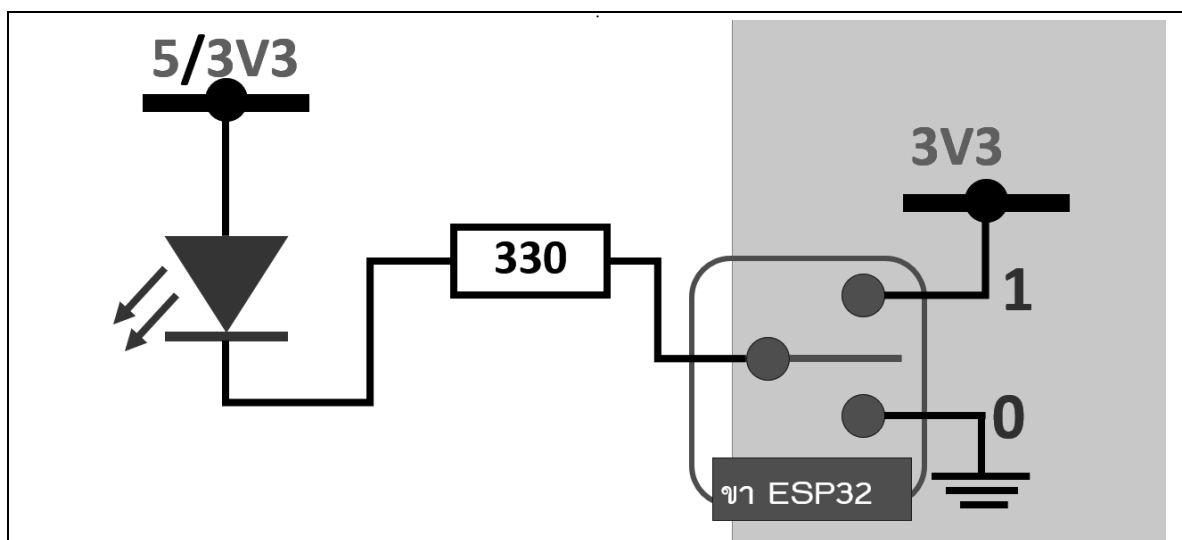
วงจรของ LAB-OUTPUT LED เป็นดังภาพที่ 3-3 โดยจะพบว่าขาแอโนดถูกต่อเข้ากับแหล่งจ่ายไฟ 5/3V3 ทางขาแคโทดเชื่อมต่อกับตัวต้านทาน 330 โอห์มเพื่อลดทอนแรงดันที่ผ่านมาจากหลอดแอลอีดีเพื่อเชื่อมต่อกับไมโครคอนโทรลเลอร์ ESP32 เพื่อเขียนโปรแกรมควบคุมการทำงานของแต่ละหลอดจะเป็นดังภาพที่ 3-4

จากภาพที่ 3-4 จะพบว่า ขาของ ESP32 มีได้ 2 สถานะ คือ 0 และ 1 โดยการมีสถานะ 0 เป็นเสมือนการเลือกเชื่อมต่อวงจรเข้ากับกราวด์ และสถานะ 1 เปรียบเสมือนการเชื่อมต่อกับกับแรงดันไฟฟ้ากระแสตรง 3V3 ด้วยเหตุนี้ เมื่อมีแรงดันจากวงจรของแอลอีดี และกำหนดให้สถานะของขาเป็น 1 ทำให้วงจรเป็นวงจรเปิด หรือแรงดันไม่ไหลจากแหล่งกำเนิดไปลงกราวด์ ส่งผลให้หลอดแอลอีดีดับ และในทางตรงกันข้ามเมื่อกำหนดสถานะของขาเป็น 0 ทำให้ขาเปรียบเสมือนเป็นกราวด์ของวงจรทำให้วงจรเป็นวงจรปิดและแรงดันถูกส่งจากแหล่งกำเนิดไปลงกราวด์ได้สำเร็จส่งผลให้หลอดแอลอีดีสว่าง



ภาพที่ 3-3 วงจรของ LAB-OUTPUT LED

ภาพโดย จารุต บุศราทิจ



ภาพที่ 3-4 การสั่งงานวงจร LAB-OUTPUT LED

ภาพโดย จารุต บุศราทิจ

> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 3-1 เป็นการสั่งงาน LED ทั้ง 8 ดวง ติดเป็นเวลา 1 วินาที และดับเป็นเวลา 1 วินาทีสลับกันไปเรื่อย ๆ โดยทำการเชื่อมต่อขา D0 ของวงจรภาคแอลอีดีเข้ากับขา P0 ของ ESP32 และส่งสัญญาณจากขา P0 เป็น 0 และ 1 สลับกันทุกๆ 1 วินาที

ตัวอย่างโปรแกรม 3-1 LED ติดและดับสลับกัน

บรรทัด	คำสั่ง
1	void setup() {
2	pinMode(0, OUTPUT);
3	}
4	void loop() {
5	digitalWrite(0, HIGH);
6	delay(1000);
7	digitalWrite(0, LOW);
8	delay(1000);
9	}

ตัวอย่างโปรแกรม 3-2 เป็นการสั่งงานให้ LED ติดเพิ่มทีละดวงทุก 500ms จนครบ 8 ดวง และเริ่มต้นใหม่ โดยการสั่งงานจำเป็นต้องสั่งงานแยก LED แต่ละดวงจึงจำเป็นต้องต่อขา LED เข้ากับขาสั่งงานโดยการต่อขาคือ

ขาของ ESP32

ขาของ LAB-OUTPUT LED

P0	D0
P2	D1
P12	D2
P15	D3
P5	D4
P18	D5
P19	D6
P23	D7

โดยตัวแปร LEDs เป็นตัวแปรเก็บหมายเลขขาของ ESP32 ในบรรทัดที่ 5-7 เป็นการกำหนดให้แต่ละขงทำหน้าที่เป็น OUTPUT ส่วนโปรแกรมบรรทัดที่ 11-16 เป็นส่วนของการวนรอบเพื่อสั่งเปิดหรือปิดการส่องสว่างของแอลอีดีโดยหลอดใดมีลำดับน้อยกว่าค่าที่เก็บในตัวแปร counter จะถูกกำหนดให้มีสถานะเป็น LOW

ตัวอย่างโปรแกรม 3-2 สั่งให้ LED ติดเพิ่มทีละดวง

บรรทัด	คำสั่ง
1	int LEDs[] = {0, 2, 12, 15, 5, 18, 19, 23 };
2	int i;
3	int counter = 1;
4	void setup() {
5	for(i = 0; i < 8; i ++){
6	pinMode(LEDs[i], OUTPUT);
7	}
8	}
9	void loop() {
10	for(i=0; i<8; i ++){
11	if(i < counter){
12	digitalWrite(LEDs[i], LOW);
13	}else{
14	digitalWrite(LEDs[i], HIGH);
15	}
16	}
17	Delay(500);
18	counter++;
19	if(counter == 9){
20	counter = 1;
21	}
22	}

ตัวอย่างโปรแกรม 3-3 สั่งงานให้ LED ทั้ง 8 ดวงติดและดับในลักษณะวิ่งทุก 500ms โดยเริ่มต้นจากดวงที่ D0 ติดดวงที่เหลือนดับ จากนั้น ดวงที่ D1 ติด ดวงที่เหลือนดับ จนถึงดวงที่ D7 ติด ดวงที่เหลือนดับ และกลับมานวนที่ D0 ใหม่ไปเรื่อย ๆ การต่อขาจะเหมือนกับตัวอย่างโปรแกรมที่ 3-2

ตัวอย่างโปรแกรม 3-3 สั่งให้แอลอีดี 8 ติดและดับแบบไฟวิ่ง

บรรทัด	คำสั่ง
1	int LEDs[] = {0, 2, 12, 15, 5, 18, 19, 23};
2	int i;
3	int counter = 0;
4	void setup() {
5	for(i = 0; i < 8; i++){
6	pinMode(LEDs[i], OUTPUT);
7	}
8	}
9	void loop() {
10	for(i = 0; i < 8; i++){
11	if(i == counter){
12	digitalWrite(LEDs[i], LOW);
13	} else {
14	digitalWrite(LEDs[i], HIGH);
15	}
16	}
17	counter++;
18	if(counter == 8){
19	counter = 0;
20	}
21	delay(500);
22	}

ตัวอย่างโปรแกรม 3-4 สั่งงานให้ LED ติดและดับในลักษณะวิ่งจากด้านนอกเข้าด้านในทุก 500ms เริ่มจากคู่ D0 และ D7 จากนั้นคู่ D1 และ D6 จนถึงคู่ D3 และ D4 และวนซ้ำไปเรื่อย ๆ โดยการต่อขาเหมือนกับตัวอย่างโปรแกรมที่ 3-3

ตัวอย่างโปรแกรม 3-4 สั่งงานให้ LED วิ่งจากด้านนอกเข้าด้านใน

บรรทัด	คำสั่ง
1	int LEDs[] = {0, 2, 12, 15, 5, 18, 19, 23};
2	int i;
3	int front = 0;
4	int rear = 7;

ตัวอย่างโปรแกรม 3-4 สั่งงานให้ LED วิ่งจากด้านนอกเข้าด้านใน (ต่อ)

บรรทัด	คำสั่ง
5	void setup() {
6	for(i = 0; i < 8; i++){
7	pinMode(LEDs[i], OUTPUT);
8	}
9	}
10	void loop() {
11	for(i = 0; i < 8; i++){
12	digitalWrite(LEDs[i], HIGH);
13	if(i == front i == rear){
14	digitalWrite(LEDs[i], LOW);
15	}
16	}
17	front++;
18	rear--;
19	if(front == 4){
20	front = 0;
21	}
22	if(rear == 3){
23	rear = 7;
24	}
25	delay(500);
26	}

ตัวอย่างโปรแกรม 3-5 สั่งงาน LED ติดและดับในลักษณะวิ่งจากด้านในไปด้านนอกทุก 500ms เริ่มจากคู่ D3 และ D4 จากนั้นคู่ D2 และ D5 จนถึงคู่ D0 และ D7 และวนซ้ำไปเรื่อย ๆ โดยการต่อขาเหมือนกับตัวอย่างโปรแกรมที่ 3-4

ตัวอย่างโปรแกรม 3-5 สั่งงานให้ LED วิ่งจากด้านในไปด้านนอก

บรรทัด	คำสั่ง
1	int LEDs[] = {0, 23, 2, 19, 12, 18, 15, 5};
2	int i;
3	int front = 0;
4	int rear = front + 1;
5	void setup() {
6	for(i = 0; i < 8; i++){
7	pinMode(LEDs[i], OUTPUT);
8	}
9	}
10	void loop() {
11	for(i = 0; i < 8; i++){
12	digitalWrite(LEDs[i], HIGH);
13	if(i == front i == rear){
14	digitalWrite(LEDs[i], LOW);
15	}
16	}
17	front = front + 2;
18	if(front > 7){
19	front = 0;
20	}
21	rear = front + 1;
22	delay(500);
23	}

> สรุป

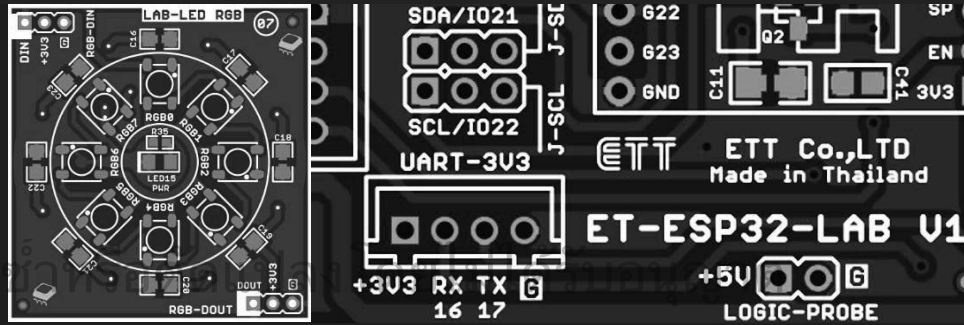
จากบทนี้จะได้ว่า การสั่งงานควบคุมแอลอีดีเป็นการสั่งงานโดยส่งสัญญาณดิจิทัล 0 หรือ 1 เพื่อสั่งให้หลอดแอลอีดีติดสว่างหรือดับด้วยคำสั่ง digitalWrite() โดยต้องกำหนดหน้าที่ของขาที่เชื่อมต่อกับวงจรของแอลอีดีให้ทำหน้าที่แบบ OUTPUT และได้เรียนรู้ถึงวิธีการทำไฟวิ่งอันเป็นแนวทางในการนำไปประยุกต์เพื่อสร้างแบบจำลองการทำงานที่ซับซ้อนยิ่งขึ้นต่อไป

บรรณานุกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ETT. (2024). *MANUAL - คู่มือ ET-ESP32-LAB V1.0*. สืบค้นเมื่อ มีนาคม 25, 2567, จาก

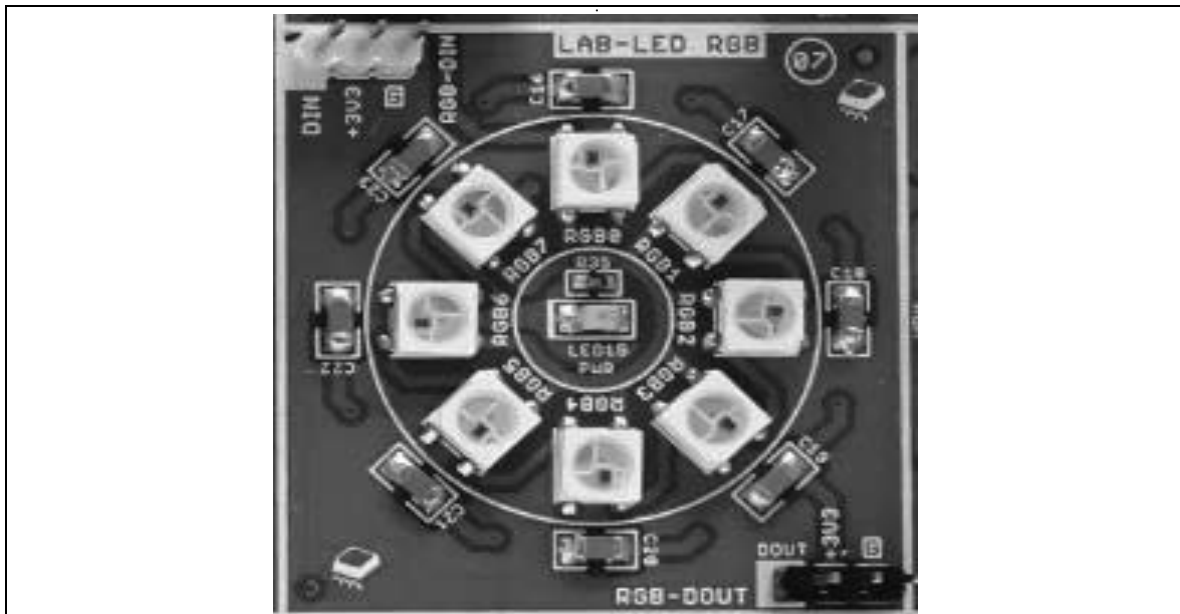
<https://www.etteam.com/prodKitNLab/ET-ESP32-LABV1/th-man-ET-ESP32-LABV1.pdf>



บทที่ 4 LAB-LED RGB

ในบทนี้กล่าวถึงเทคโนโลยีการผลิตแอลอีดีในปัจจุบันสามารถผลิตหลอดแอลอีดีให้แสดงสีแบบ RGB (Red-Green-Blue) ได้ภายในหลอดเดียวกัน ทำให้ผู้พัฒนาสามารถเลือกใช้สีได้หลากหลาย และมีวิธีการเชื่อมต่อใช้งานที่ไม่ซับซ้อน ในบทนี้กล่าวถึงโมดูล WS2812B หรือ NeoPixel ในภาค LAB-LED RGB ดังภาพที่ 4-1 เพื่อทำความเข้าใจไอซีควบคุมการทำงานของแอลอีดีแบบ RGB, การผสมสี, การเข้าถึงตำแหน่งของแอลอีดีแต่ละหลอด และประยุกต์การทำไฟวิ่งในรูปแบบต่าง ๆ เช่น วิ่งไล่ไปที่หลอดไป/กลับ เป็นต้น

เมื่อจบบทนี้ ผู้อ่านจะเข้าใจถึงขั้นตอนการเขียนโปรแกรม และวิธีการเขียนโปรแกรมควบคุมการทำงานของหลอดแอลอีดีสีแบบ RGB อันเป็นพื้นฐานในการประยุกต์ใช้กับจำนวนหลอดที่มากขึ้น หรือใช้กับอุปกรณ์ต่อพ่วงอื่น ๆ



ภาพที่ 4-1

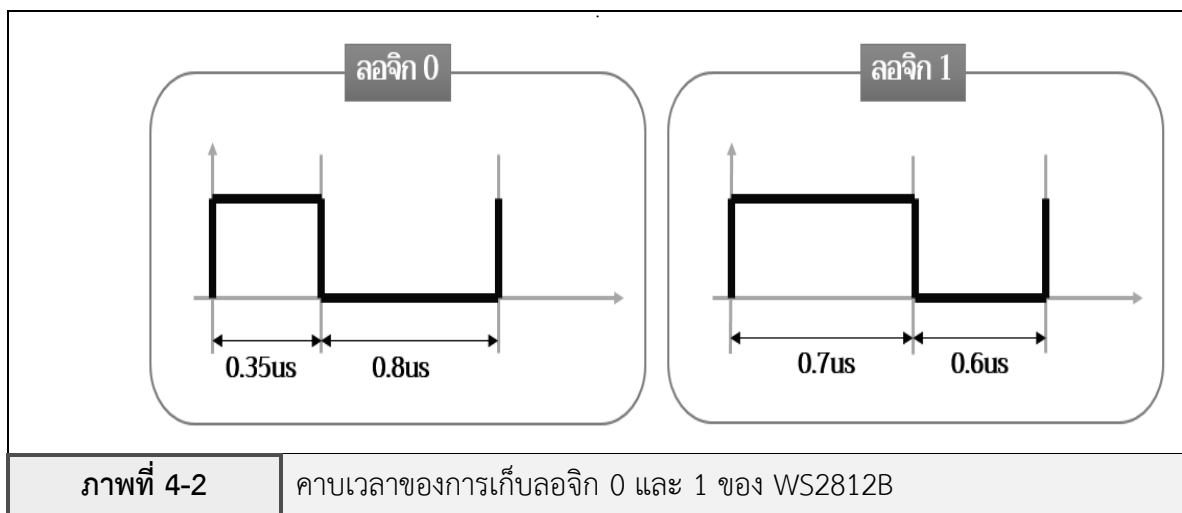
ภาพโมดูล LAB-LED RGB

ภาพโดย จารุต บุศราทิจ

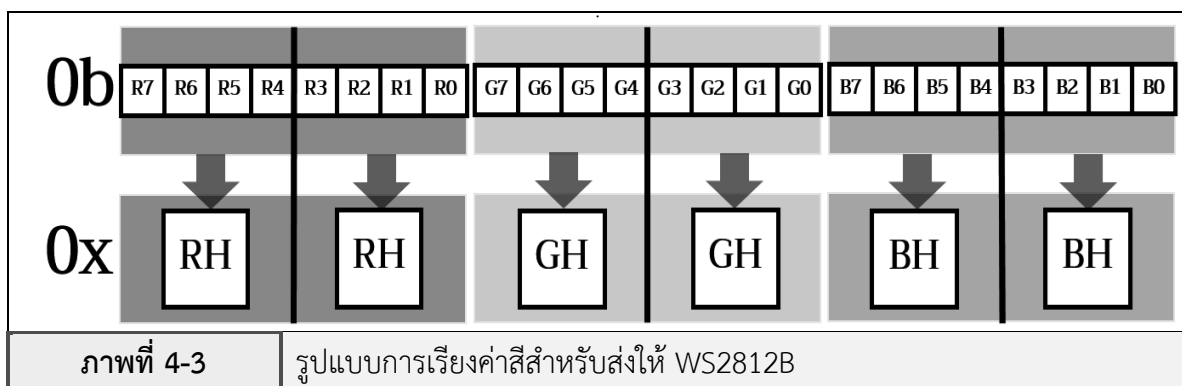
> แนะนำ WS2812B

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ภาค LAB-LED RGB เป็นโมดูลแอลอีดีแบบอาร์จีบีที่เรียกว่า NeoPixel ใช้ไอซี WS2812B จำนวน 8 ดวง ที่ผู้เขียนโปรแกรมสามารถควบคุมหลอดแอลอีดีสีแบบ RGB ทำงานด้วยการควบคุมจากสายเพียง 1 เส้นแบบ NZR ที่ทำงานแบบอนุกรม และค่าที่เป็น 0 หรือ 1 จะถูกกำหนดด้วยคาบเวลาของการมีสถานะของแรงดัน 0 หรือ 1 ในช่วงที่กำหนด ดังภาพที่ 4-2 สีของแอลอีดีแต่ละดวงเกิดได้โดยผสมสีจากแม่สี RGB คือ สีแดง, สีเขียวและสีน้ำเงิน แต่ละสีมีค่าอยู่ในช่วง 0 ถึง 255 หรือเรียกว่าสามารถควบคุมระดับความเข้มของแม่สีได้ 256 ระดับ ทำให้การผสมสีของแม่สีทั้งสามรวมเป็นจำนวนสีได้ $256^3 = 16,777,216$ หรือ 16.7 ล้านสี โดยความถี่ในการสแกนจะไม่ต่ำกว่า 400Hz การส่งข้อมูลมีความเร็วสูงสุดอยู่ที่ 800kbps หรือ 800,000 บิตต่อวินาที (bps) รองรับการทำงานที่แรงดันไฟฟ้ากระแสตรง 3.3V-5V ดังภาพที่ 4-3



ภาพโดย จารุต บุศราทิจ



ภาพโดย จารุต บุศราทิจ



ภาพโดย จารุต บุศราทิจ

ในภาพที่ 4-4 เป็นผังวงจรของภาค LAB-LED RGB บนบอร์ด ET-ESP32 LAB V1 จะพบว่ามีโมดูล WS2812B ต่อพ่วงกันเป็นจำนวน 8 หลอด โดยใช้แหล่งจ่ายไฟกระแสตรงแรงดัน 3V3 มีขา DIN เป็นขาเชื่อมต่อกับขาของไมโครคอนโทรลเลอร์ ESP32 สำหรับส่งสัญญาณควบคุมการทำงานของโมดูล WS2812B และขา DOUT สำหรับใช้เชื่อมต่อเพิ่มเติมโมดูล WS2812B เพิ่มเติมภายนอกเพื่อให้มีจำนวนหลอด RGB เพิ่มขึ้น

แต่สิ่งหนึ่งที่ต้องระมัดระวังในการใช้งานแหล่งจ่ายไฟต้องมีกระแสที่เพียงพอต่อการขับหลอดแอลอีดีของโมดูล WS2812B ที่ต่อพ่วงใช้งาน ดังนั้น ในการเสียบตัวอย่างของบทนี้ผู้อ่านควรต่อแหล่งจ่ายไฟจากอะแดปเตอร์เข้าบอร์ด ET-ESP32 LAB V1 เพิ่มขึ้น

> ไลบรารี neopixel

การใช้งาน WS2812B สามารถใช้ไลบรารี neopixel ของ Adafruit โดยการใช้งานต้องกำหนดหมายเลขขาที่เชื่อมต่อกับ DIN และเรียกใช้ Adafruit_NeoPixel.h ดังนี้

```
#include <Adafruit_NeoPixel.h>
```

การสร้างตัวแปรสำหรับอ้างอิงโมดูลต้องกำหนดพารามิเตอร์หรืออาร์กิวเมนต์ 3 ตัว คือ จำนวนหลอด, ขาของ ESP32 ที่เชื่อมต่อกับ DIN และประเภทของหลอดแสดงผล โดยประเภทของหลอดแสดงผลมีรายการค่าคงที่ดังนี้

- NEO_KHZ800 สำหรับโมดูลที่ทำงาน 800kHz
- NEO_KHZ400 สำหรับโมดูลที่ทำงาน 400kHz
- NEO_GRB สำหรับโมดูลที่จะเรียงข้อมูลสีเป็น GRB หรือ เขียวแดง น้ำเงิน
- NEO_RGB สำหรับโมดูลที่เรียงค่าสีเป็น RGB หรือ แดงเขียวน้ำเงิน
- NEO_RGBW สำหรับโมดูลที่ใช้การกำหนดค่าสีเป็น 4 ไบต์ สำหรับ RGBW

```
Adafruit_NeoPixel ชื่อตัวแปร = Adafruit_NeoPixel(
                                จำนวนหลอด, ขาเชื่อมต่อ,
                                ประเภทของหลอด )
```

การเริ่มต้นทำงานตามค่าที่กำหนดไว้ต้องเรียกใช้คำสั่ง `begin()` ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```
ชื่อตัวแปร.begin()
```

กรณีที่ต้องการกำหนดระดับความสว่างของหลอดจะต้องเรียกคำสั่ง `setBrightness()` ตามด้วยอาร์กิวเมนต์ ระดับความสว่าง ที่ต้องตัวเลขจำนวนเต็มในช่วง 0 ถึง 255 ดังรูปแบบของการใช้งานดังนี้

```
ชื่อตัวแปร.setBrightness( ระดับความสว่าง )
```

การกำหนดค่าสีของหลอดจะต้องสร้างตัวแปรเก็บค่าสีด้วยคำสั่ง `Color()` ที่มีอาร์กิวเมนต์เป็นระดับของสีแดง ระดับของสีเขียว และ ระดับของสีน้ำเงิน หลังจากนั้นเรียกใช้คำสั่ง `setPixelColor()` โดยระบุ หมายเลขหลอด ที่ต้องกำหนดกำหนดสี และ ค่าสี ของหลอดนั้น ดังรูปแบบของคำสั่งต่อไปนี้

```
ค่าสี = ชื่อตัวแปร.Color( ระดับของสีแดง, ระดับของสีเขียว, ระดับของสีน้ำเงิน )
ชื่อตัวแปร.setPixelColor( หมายเลขหลอด, ค่าสี )
```

การสั่งให้มีการอัปเดตค่าของการแสดงไปยังชุดของหลอดแอลอีดีจะต้องเรียกใช้คำสั่ง `show()` ดังนี้

```
ชื่อตัวแปร.show()
```

ตัวอย่างการสร้างตัวแปรเพื่อใช้กับภาค LAB-LED RGB ของบอร์ด ET-ESP32 LAB V1 โดยเชื่อมต่อขา P25 เข้ากับ DIN สามารถสร้างได้ดังนี้

```
#define PIN 25
```

```
Adafruit_NeoPixel ledRGB = Adafruit_NeoPixel(8, PIN, NEO_RGB+NEO_KHZ800)
```


> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 4-1 เป็นการสั่งงานเปิด/ปิด RGB ในลักษณะครึ่งวงกลมโดยมีการเปลี่ยนสีของ RGB ครึ่งละ 2 ดวงเริ่มจากดวงที่ 1 และ 5 เป็นดวงที่ 2 และ 6 จนครบถึงดวงที่ 4 และ 8 จึงเปลี่ยนสีและเพิ่มความสว่างและวนซ้ำในลักษณะเดิมจากจุดเริ่มต้น โดยในบรรทัดที่ 9 – 14 เป็นการตั้งค่าสีที่ต้องการวนซ้ำ บรรทัดที่ 16 – 25 เป็นฟังก์ชันในการไล่สี RGB ตามสีที่กำหนด บรรทัดที่ 39 – 43 เป็นการตั้งค่าความสว่าง การต่อขาของโมดูล RGB ขา DIN กับขาที่ 33 ของ ESP32 และขา +3V3 ของโมดูล RGB เข้ากับขา +3V3 ของโมดูลจ่ายไฟ

ตัวอย่างโปรแกรม 4-1 สั่งงานเปิด/ปิด RGB ในลักษณะครึ่งวงกลม

บรรทัด	คำสั่ง
1	#include <Adafruit_NeoPixel.h>
2	#define RGB 33
3	int allRGB = 8;
4	int brightness = 50;
5	int waitTime = 200;
6	int currentColor = 0;
7	Adafruit_NeoPixel rgbs = Adafruit_NeoPixel(
8	allRGB, RGB, NEO_GRB + NEO_KHZ800);
9	uint32_t colors[] = {
10	rgbs.Color(255, 0, 0),
11	rgbs.Color(0, 255, 0),
12	rgbs.Color(0, 0, 255),
13	rgbs.Color(255, 255, 255),
14	rgbs.Color(0, 0, 0)
15	};
16	void RGB_Spinning(long color, int wait)
17	{
18	for(int rgb=0; rgb < allRGB/2; rgb++)
19	{
20	rgbs.setPixelColor(rgb,color);
21	rgbs.setPixelColor(rgb+4,color);
22	rgbs.show();
23	delay(wait);
24	}

ตัวอย่างโปรแกรม 4-1 สั่งงานเปิด/ปิด RGB ในลักษณะครึ่งวงกลม (ต่อ)

บรรทัด	คำสั่ง
25	}
26	void setup()
27	{
28	rgbs.begin();
29	rgbs.show();
30	rgbs.setBrightness(brightness);
31	}
32	void loop()
33	{
34	RGB_Spinning(colors[currentColor], waitTime);
35	currentColor++;
36	if(currentColor > 4){
37	currentColor = 0;
38	}
39	rgbs.setBrightness(brightness);
40	brightness = brightness + 50;
41	if(brightness > 255){
42	brightness = 50;
43	}
44	}

> สรุป

จากบทนี้จะพบว่าการสั่งงาน neopixel สะดวกง่ายดายเนื่องจากมีไลบรารี ที่เตรียมการให้เรียกใช้โดยผู้เขียนโปรแกรมมีหน้าที่สั่งล้างค่า กำหนดสี และระดับของความสว่างแก่หลอดแอลอีดี สามารถสั่งงานแอลอีดีแบบผสมแม่สีสีแดง เขียว และน้ำเงินได้สีละ 256 ระดับ ทำให้เกิดสีได้มากถึง 16.7 ล้านสี

บรรณานุกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

Adafruit. (2024). *Adafruit NeoPixel Library*. สืบค้นเมื่อ มีนาคม 28, 2567, จาก

https://github.com/adafruit/Adafruit_NeoPixel

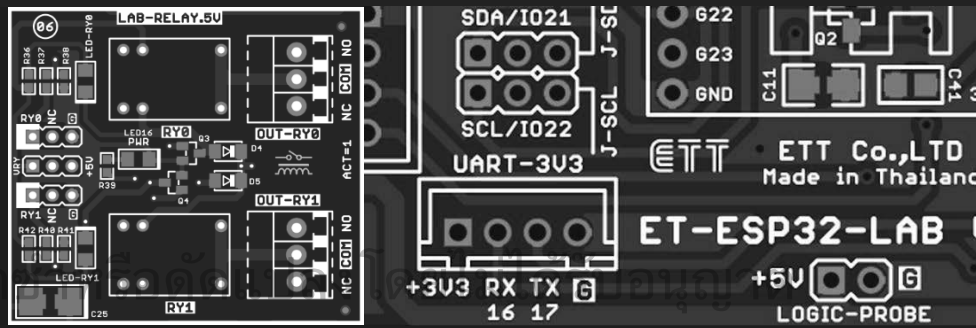
Worldsemi. (2024). *WS2812B Datasheet*. สืบค้นเมื่อ มีนาคม 28, 2567, จาก

<https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf>



บทที่ 5

LAB-RELAY.5V



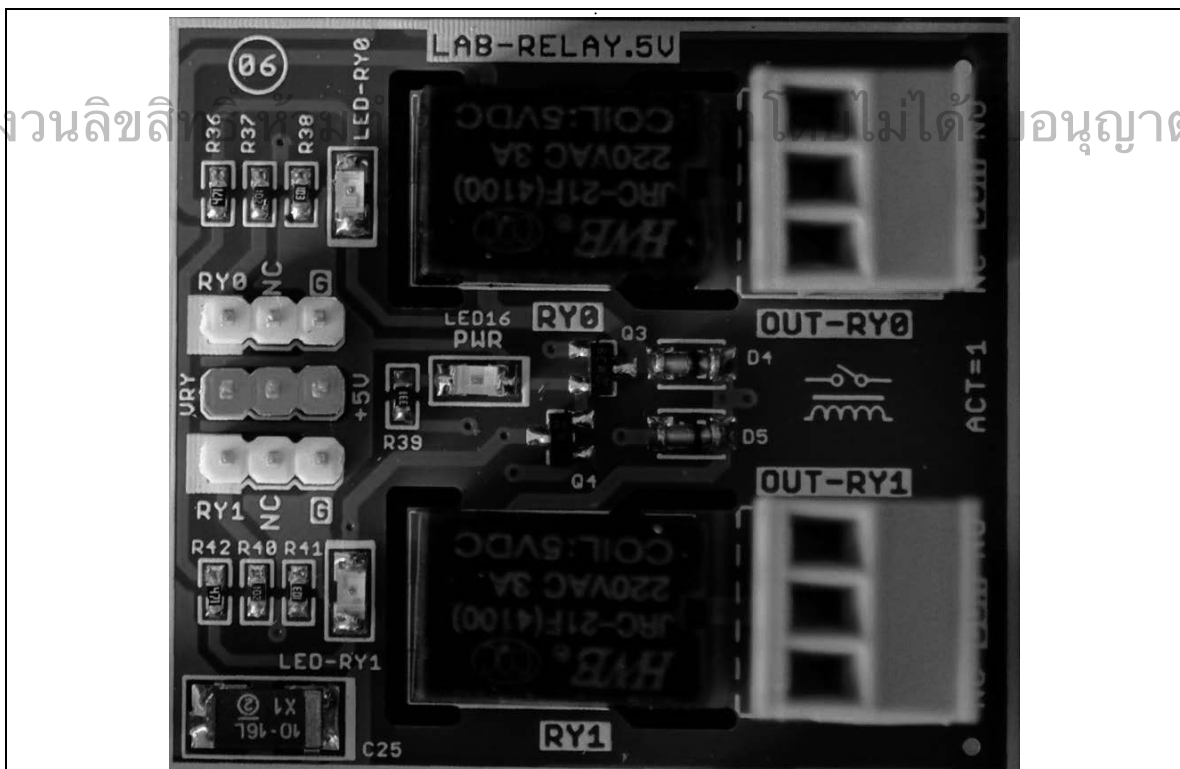
ในบทนี้กล่าวถึงการเขียนโปรแกรมเพื่อเชื่อมประสานส่วนของวงจรรีเลย์บนบอร์ด ET-ESP32 LAB V1 ที่เรียกว่า LAB-RELAY .5V ดังภาพที่ 5-1 อันเป็นโมดูลที่ออกแบบมาเพื่อให้ ESP32 สามารถขับโหลดอุปกรณ์ภายนอกด้วยรีเลย์ (Relay) จำนวน 1 ตัว โดย ส่งสัญญาณดิจิทัลนำออกเป็น 1 หรือ 0 เพื่อควบคุมการทำงานของรีเลย์

เมื่อจบบทนี้ ผู้อ่านจะเข้าใจถึงขั้นตอนการเขียนโปรแกรม และวิธีการเขียนโปรแกรมเพื่อส่งสัญญาณควบคุมรีเลย์ซึ่งเป็นประโยชน์ในการประยุกต์ใช้งานเพื่อให้ ESP32 สามารถเปิดปิดอุปกรณ์ไฟฟ้าภายนอก เช่น หลอดไฟ มอเตอร์ เป็นต้น

> แนะนำ โมดูลรีเลย์

โมดูลรีเลย์เป็นโมดูลสำหรับทดลองการส่งออกสัญญาณดิจิทัลเพื่อควบคุมการทำงานของรีเลย์ขนาดคอยล์ 5VDC ดังภาพที่ 5-2 รองรับการทำงานกับโหลดกระแสได้สูงสุดที่ 3A ใช้กับไฟฟ้ากระแสสลับ 220V และมีขั้วต่อทั้ง NO และ NC มาให้ ดังนั้น ในการทดลองหรือนำไปใช้จริงผู้อ่านต้องระมัดระวังการต่อวงจรเมื่อใช้กับไฟฟ้ากระแสสลับ และต้องตรวจสอบให้แน่ใจว่าโหลดที่นำมาใช้นั้นกินกระแสไม่เกิน 3A ทั้งนี้เพื่อความปลอดภัย และป้องกันความผิดพลาดที่อาจเกิดกับอุปกรณ์และผู้ใช้

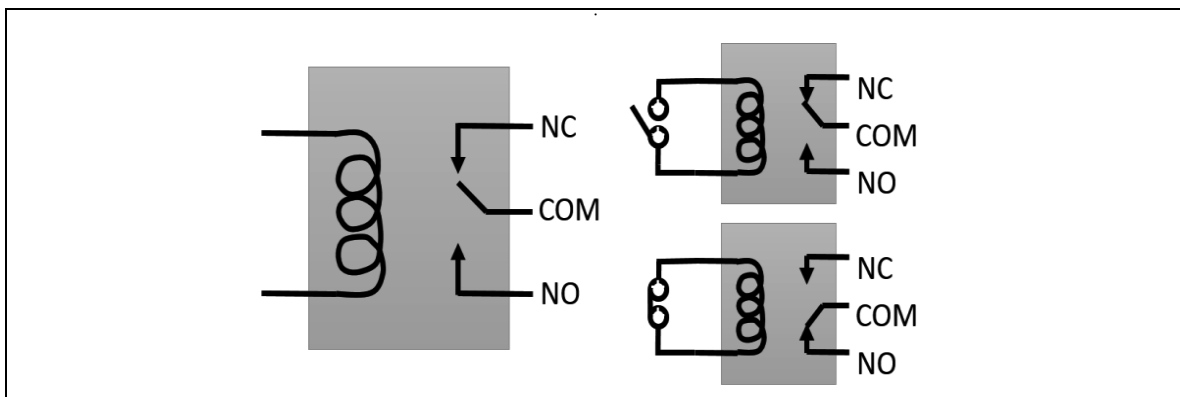
ภาค LAB RELAY.5V ประกอบด้วยรีเลย์จำนวน 2 ตัว โดยแต่ละตัวมีขั้วต่อ 3 ขา สำหรับเชื่อมต่อกับกราวด์ แหล่งไฟฟ้ากระแสตรง และขา IO ที่ต่อเข้ากับการควบคุมเปิด/ปิดรีเลย์ผ่านทางขั้ว RY0 และ RY1 สำหรับเชื่อมต่อกับบอร์ด ESP32 และฝั่งของรีเลย์มีขา COM และ NO สำหรับต่อกับอุปกรณ์ไฟฟ้าภายนอกพร้อมหลอดแอลอีดีแสดงสถานะของรีเลย์



ภาพที่ 5-1

ภาพโมดูล LAB RELAY.5V

ภาพโดย จารุต บุศราทิจ



ภาพที่ 5-2

การเชื่อมต่อของขา NC, COM และ NO ในรีเลย์

ภาพโดย จารุต บุศราทิจ

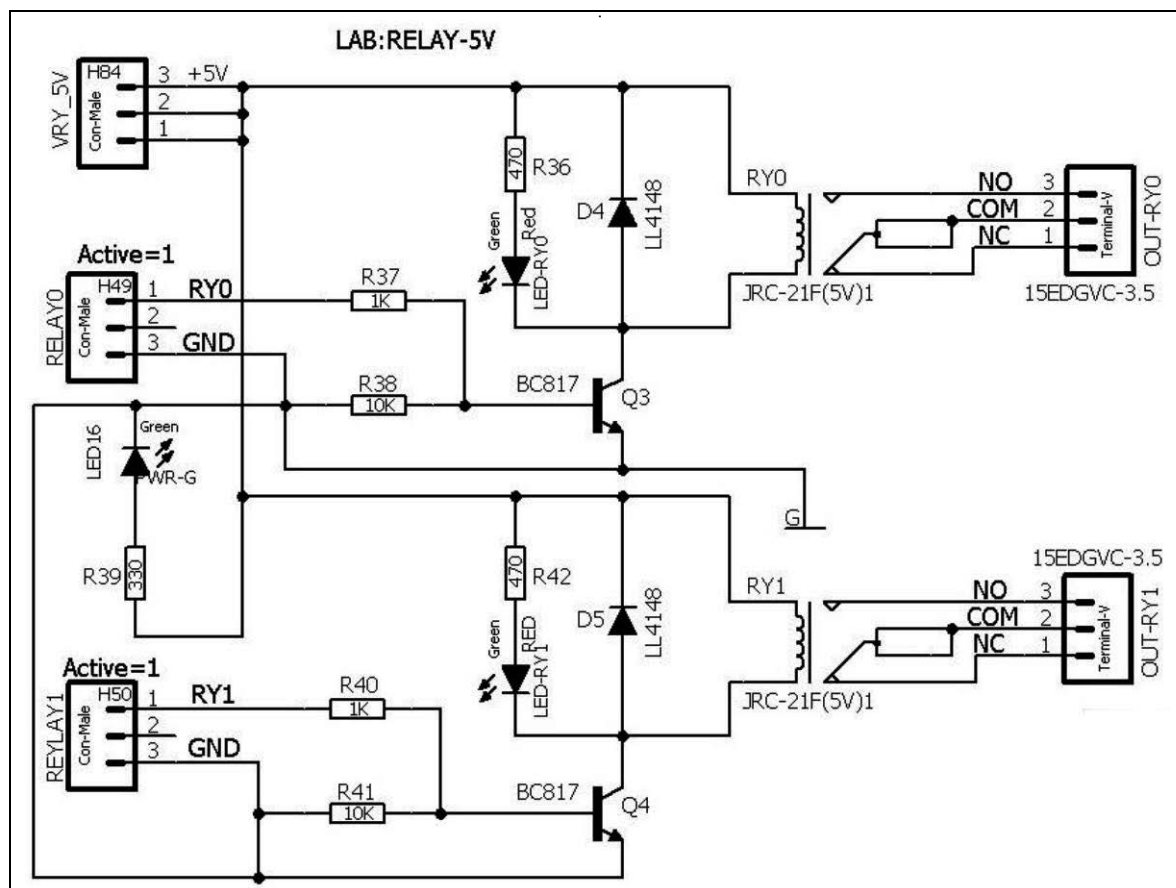
จากภาพ 5-1 จะพบว่าการใช้โมดูล LAB RELAY.5V กับอุปกรณ์ไฟฟ้าภายนอกจะมีขาเชื่อมต่อ 3 ขา คือ NC COM และ NO โดยในการต่อวงจรให้ต่อ COM เข้ากับกราวด์ของวงจรภายนอก และ NO ต่อเข้ากับอุปกรณ์ที่เป็นฝั่งขั้วลบหรือกราวด์ของอุปกรณ์ และแหล่งจ่ายไฟแยกกันใช้กับบอร์ด ESP32 แต่อย่างไรก็ดี เนื่องด้วยรีเลย์แต่ละรุ่นมีคุณสมบัติที่แตกต่างกัน ซึ่งในโมดูลนี้เป็นรีเลย์สำหรับไฟกระแสสลับ (ซึ่งใช้กับไฟฟ้ากระแสตรงได้) แรงดัน 220V มีความคงทนต่อกระแสสูงสุดที่ 3A และต้องใช้แรงดันไฟฟ้า

กระแสตรง 5VDC เป็นตัวขับเคลื่อนการเปิดและปิด สดท้าย สิ่งที่ต้องระวังก่อนทดลองกับไฟฟ้ากระแสสลับหรือไฟบ้านจะต้องเชื่อมต่อวงจรให้ถูกต้องก่อนจ่ายกระแสไฟเข้าวงจร

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

> วงจรและการทำงาน

วงจรของ LAB-RELAY.5V เป็นดังภาพที่ 5-3 จะพบว่าหลอดแอลอีดี LED-RY0 และ LED-RY1 เป็นไฟแสดงสถานการณ์ทำงานของรีเลย์ตัวแรกและตัวที่สอง เมื่อสังเกตจะพบ Q3 และ Q4 เป็นทรานซิสเตอร์เบอร์ BC817 เป็นตัวขับโหลดให้กับขานำเข้าสัญญาณ RY0 และ RY1 ที่มาจาก ESP32



ภาพที่ 5-3

วงจรของ LAB-RELAY.5V

ภาพโดย จารุต บุศราทิจ

> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 5-1 เป็นการสั่งงานเปิด/ปิดด้วยสวิตช์กดติดปล่อยดับโดยมีการเชื่อมต่อรีเลย์กับขาที่ 16 และ สวิตช์ที่ขา 17 จากนั้นตรวจสอบการกดสวิตช์หากมีการกดสวิตช์จะส่งสถานะ LOW เพื่อทำการกลับค่าตัวแปร status ให้เป็นจริงหรือเท็จและสั่งชะลอโปรแกรมเพื่อ debounce เพื่อป้องกันการส่งค่าซ้ำที่รวดเร็วมากเกินไป จากนั้นสั่งเปิดหรือปิดรีเลย์ตามสถานะปัจจุบัน

ตัวอย่างโปรแกรม 5-1 สั่งงานเปิด/ปิดด้วยสวิตช์

บรรทัด	คำสั่ง
1	#define relay1 16
2	#define switch1 17
3	bool status = false;
4	void setup() {
5	pinMode(relay1, OUTPUT);
6	pinMode(switch1, INPUT);
7	}
8	void loop() {
9	if(digitalRead(switch1) == LOW){
10	status = !status;
11	delay(300);
12	}
13	if(status){
14	digitalWrite(relay1, HIGH);
15	}else{
16	digitalWrite(relay1, LOW);
17	}
18	}

ตัวอย่างโปรแกรม 5-2 เป็นการสั่งงานเปิด/ปิดด้วยตัวตั้งเวลา RTC3231 ดังจะกล่าวถึงรายละเอียดอยู่ในบทที่ 17 โดยโปรแกรมตัวอย่างจะสั่งเปิดรีเลย์เป็นเวลา 5 วินาที และปิดเป็นเวลา 1 วินาทีสลับกันไปเรื่อย ๆ การเชื่อมต่อขา SDA ของ RTC เชื่อมต่อขา 21 และ SCL ต่อขา 22 ผ่านทางขั้วต่อ I²C แบบ 3V3 โดยในบรรทัดที่ 6 ถึง 12 เป็นส่วนของการกำหนดค่าวันที่และเวลาเพื่อนำไปตั้งค่าให้กับโมดูล RTC ในบรรทัดที่ 16

ตัวอย่างโปรแกรม 5-2 สักงานเปิด/ปิดด้วยตัวตั้งเวลา

บรรทัด	คำสั่ง
1	#include <uRTCLib.h>
2	#define relay1 16
3	#define OLED_SDA 21
4	#define OLED_SCL 22
5	uRTCLib rtc(0x68,URTCLIB_MODEL_DS3231);
6	int hour = 12; //0-23
7	int minute = 0; //0-59
8	int sec = 0; //0-59
9	int day = 5; // 1(sun) - 7(Sat)
10	int date = 1; // 1-31
11	int month = 2; // 1(Jan)-12(Dec)
12	int year = 24; //0-99 คศ
13	void SetTimeDate ()
14	{
15	URTCLIB_WIRE.begin();
16	rtc.set(sec,minute,hour,day,date,month,year);
17	}
18	void setup () {
19	pinMode(relay1, OUTPUT);
20	SetTimeDate();
21	}
22	void loop () {
23	rtc.refresh() ;
24	if(rtc.second() % 5 != 0){
25	digitalWrite(relay1, HIGH);
26	}else{
27	digitalWrite(relay1, LOW);
28	}
29	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

โมดูล LAB RELAY.5V เป็นโมดูลติดตั้งรีเลย์เพื่อใช้ควบคุมการเปิดหรือปิดอุปกรณ์ไฟฟ้าทั้งที่เป็นไฟฟ้ากระแสตรงหรือกระแสสลับ โดยขับโหลดได้สูงสุด 3A จึงสามารถนำไปประยุกต์ใช้งานกับการเปิด/ปิดหลอดไฟในบ้าน หรืออุปกรณ์ ซึ่งเมื่อใช้ร่วมกับโมดูลนาฬิกาฐานเวลาจริงหรือตัวตั้งเวลาของ ESP32 เพื่อตั้งเวลาของการเปิด/ปิดหลอดไฟ หรือนำไปใช้กับอัลตราโซนิก เพื่อปิดปิดก๊อคน้ำเมื่อมีคนหรือวัตถุเข้าใกล้ในระยะ และประยุกต์กับเซ็นเซอร์อุณหภูมิ เพื่อสั่งเปิด/ปิดตัวทำความเย็น หรือทำความร้อนตามเงื่อนไขที่ต้องการ

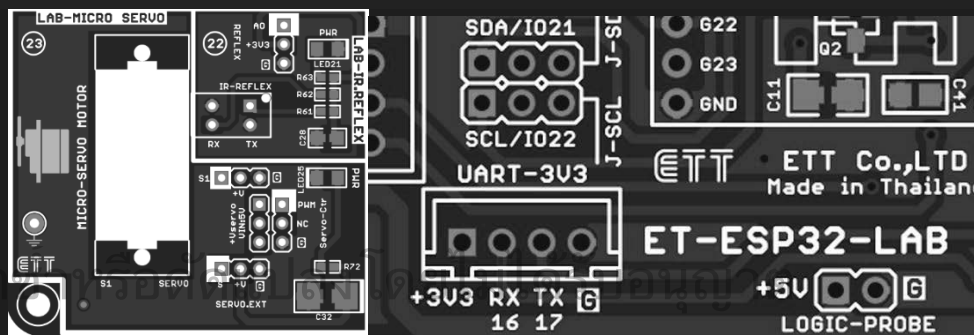
แต่อย่างไรก็ดี การนำไปขับโหลดจะต้องใช้กระแสไม่เกิน 3A และการใช้กับวงจรไฟฟ้ากระแสสลับ จะต้องระมัดระวังอุบัติเหตุจากการลัดวงจรหรือไฟดูดเสมอ

บรรณานุกรม

ETT. (2024). *MANUAL - คู่มือ ET-ESP32-LAB V1.0*. สืบค้นเมื่อ มีนาคม 30, 2567, จาก <https://www.etteam.com/prodKitNLab/ET-ESP32-LABV1/th-man-ET-ESP32-LABV1.pdf>



บทที่ 6



ในบทนี้กล่าวถึงการใช้ LAB-MICRO SERVO ดังภาพที่ 6-1 ขั้วมอเตอร์ที่เรียกว่าเซอร์โวมอเตอร์ (Servo Motor) ซึ่งเป็นอุปกรณ์พื้นฐานที่นำไปประยุกต์กับการควบคุมการเคลื่อนที่ของอุปกรณ์ได้หลากหลายโดยใช้สายสัญญาณควบคุมทิศทางหมุนเพียงเส้นเดียว และกล่าวถึง วิธีการสั่งงานเซอร์โวมอเตอร์ให้หมุนตามเข็มนาฬิกา หมุนทวนเข็มนาฬิกา และการหมุนไปที่ตำแหน่งเริ่มต้น พร้อมตัวอย่างโปรแกรม เพื่อเป็นพื้นฐานในการสั่งงานในระดับที่ซับซ้อนขึ้น เช่น การควบคุมการเคลื่อนที่ของรถที่ใช้เซอร์โวมอเตอร์เป็นตัวขับเคลื่อน การควบคุมข้อต่อของแขนกล เป็นต้น

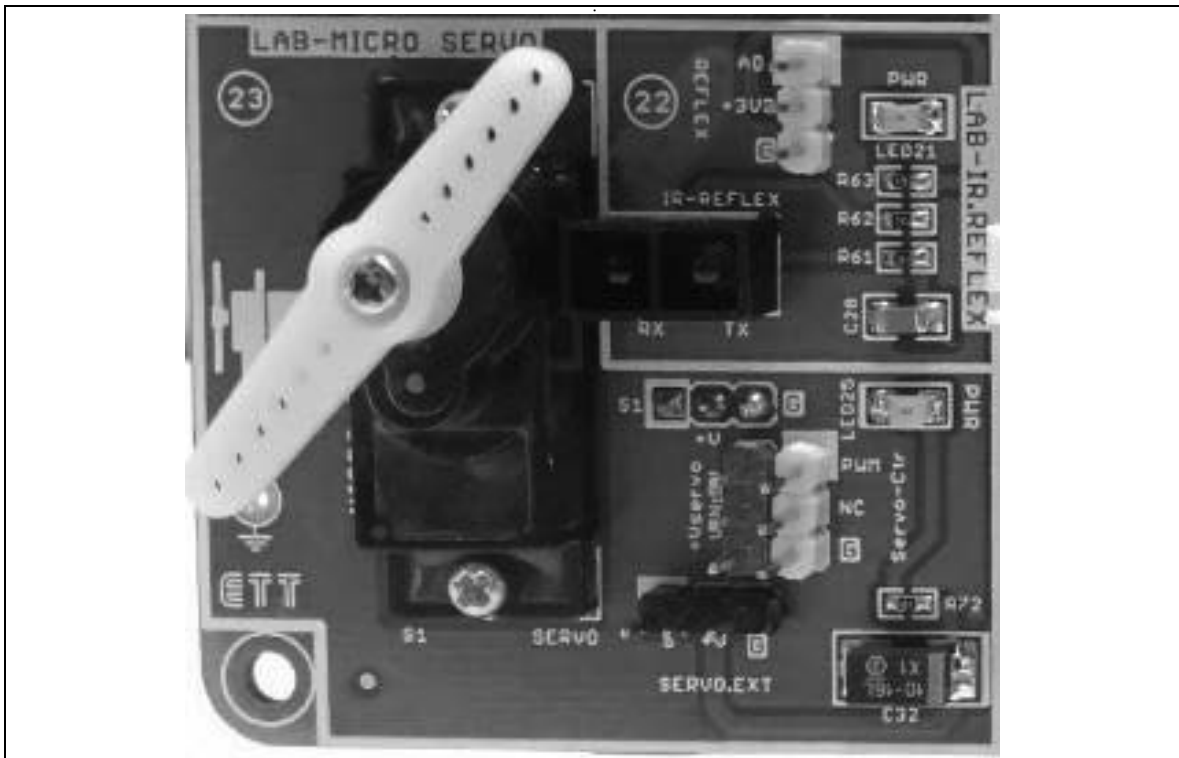
> แนะนำ เซอร์โวมอเตอร์

เซอร์โวมอเตอร์เป็นมอเตอร์ที่มีชุดเฟืองสำหรับทดแรงขับเคลื่อนจึงใช้แรงดันและกระแสไฟน้อย แต่ขับโหลดได้สูง เป็นแอกชูเอเตอร์แบบโรตารี (rotary actuator) หรือแอกชูเอเตอร์แบบเส้นตรง (linear actuator) ที่สามารถถูกควบคุมมุมการหมุน หรือความเร็วในการหมุนได้อย่างแม่นยำ นิยมใช้ในการควบคุมอุปกรณ์หุ่นยนต์หรือเครื่องจักร ตัวอย่างของเซอร์โวมอเตอร์ที่ใช้ในหนังสือเล่มนี้เป็นดังภาพที่ 6-1

การสั่งงานเซอร์โวมอเตอร์แตกต่างจากมอเตอร์ปกติ คือ ต้องส่งเป็น ค่าดิวตี (Duty) หรือช่วงที่มีค่าสัญญาณดิจิทัลเป็น 1 ที่แตกต่างกันเพื่อสั่งงานให้มอเตอร์หมุนมาตรงกลาง หมุนทวนเข็มนาฬิกา หรือหมุนตามเข็มนาฬิกา ดังภาพที่ 6-2

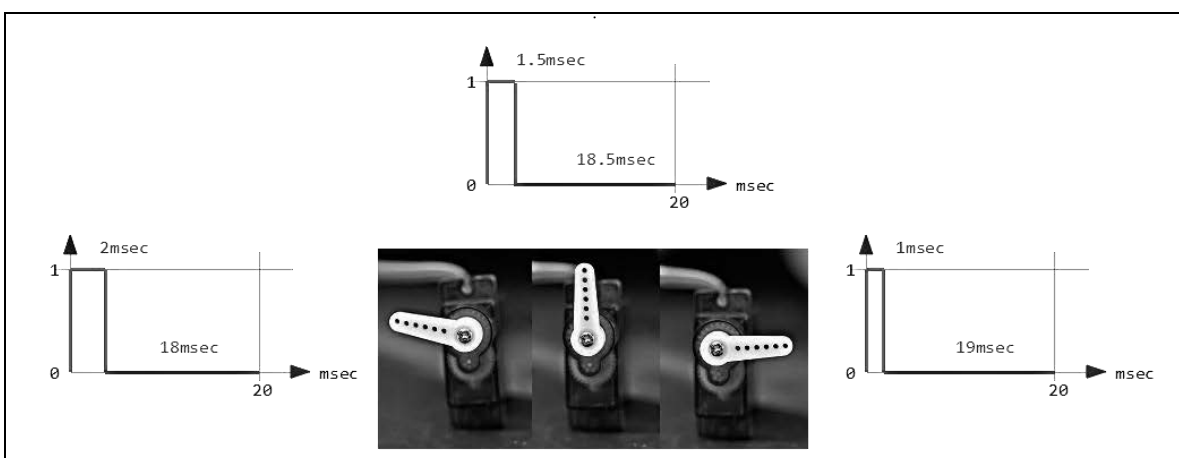
จากภาพที่ 6-2 จะเห็นได้ว่า 1 คาบเวลาของการส่งสัญญาณพัลส์ให้แก่เซอร์โวมอเตอร์ต้องใช้เวลา 20 มิลลิวินาที โดยการส่งสัญญาณดิจิทัล 1 ยาวเป็นเวลา 1 มิลลิวินาที และส่งสัญญาณดิจิทัล 0 อีก 19 มิลลิวินาทีที่มีผลทำให้เซอร์โวมอเตอร์หมุนไปในทิศตามเข็มนาฬิกา การส่งสัญญาณดิจิทัล 1 เป็นเวลา 1.5

มิลลิวินาที ตามด้วยสัญญาณดิจิทัล 0 เป็นเวลา 18.5 มิลลิวินาทีมีผลทำให้เซอร์โวมอเตอร์หมุนกลับมาที่ตรงกลาง และการส่งสัญญาณดิจิทัล 1 เป็นเวลา 2 มิลลิวินาทีตามด้วยสัญญาณดิจิทัล 0 เป็นเวลา 18 มิลลิวินาทีทำให้เซอร์โวมอเตอร์หมุนทวนเข็มนาฬิกา



ภาพที่ 6-1 ภาพโมดูลเซอร์โวมอเตอร์ของ ET-ESP32 LAB V1

ภาพโดย จารุต บุศราทิจ



ภาพที่ 6-2 ความสัมพันธ์ของดิวตีกับการหมุนของเซอร์โวมอเตอร์

ภาพโดย จารุต บุศราทิจ

> วิธีการคำนวณ

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

วิธีการกำหนดสัดส่วนของการส่งสัญญาณดิจิทัล 1 และสัญญาณดิจิทัล 0 ในแต่ละคาบเวลาต้องอาศัยการทำ PWM (กล่าวถึงในบทที่ 2) ซึ่ง ESP32 อาศัยการคำนวณสัดส่วนแรงดันตามความละเอียดของวงจรแปลงสัญญาณแอนะล็อกเป็นดิจิทัลและวงจรแปลงสัญญาณดิจิทัลเป็นแอนะล็อก ซึ่งมีความละเอียดอยู่ที่ 10 บิต (2^{10}) ทำให้สามารถแยกความละเอียดของแรงดันได้ 1024 ค่า หรือ น้อยสุดเป็น 0 และค่ามากที่สุดเป็น 1023 นั้นหมายความว่า การสร้างพัลส์ที่มีสัญญาณดิจิทัล 1 เป็นร้อยละ 50 และสัญญาณดิจิทัล 0 เป็นร้อยละ 50 ทำให้ค่าของแรงดันเหลือครึ่งหนึ่งของค่าสูงสุด นั่นคือ เหลือ 512 และเช่นเดียวกัน เมื่อสัดส่วนของการเกิดสัญญาณดิจิทัล 1 มีค่าเวลาน้อยลงจึงมีผลให้ค่าที่ส่งออกไปยังขา pinx มีค่าที่แตกต่างกันออกไป

ดังนั้น การส่งสัญญาณดิจิทัล 1 เพื่อควบคุมเซอร์โวมอเตอร์จึงมีค่าดังสมการ 6-1, สมการที่ 6-2 และ สมการที่ 6-3

ค่าที่ส่งให้ pinx กรณีหมุนตามเข็มนาฬิกา

$$\begin{aligned} v_{\text{digital}} &= \frac{1}{20} \times \text{Analog}_{\text{resolution}} && \text{สมการที่ 6-1} \\ &= \frac{1}{20} \times 1024 \\ &= 51.2 \end{aligned}$$

ค่าที่ส่งให้ pinx กรณีหมุนไปตรงกลาง

$$\begin{aligned} v_{\text{digital}} &= \frac{1.5}{20} \times 1024 && \text{สมการที่ 6-2} \\ &= 76.8 \end{aligned}$$

ค่าที่ส่งให้ pinx กรณีหมุนทวนเข็มนาฬิกา

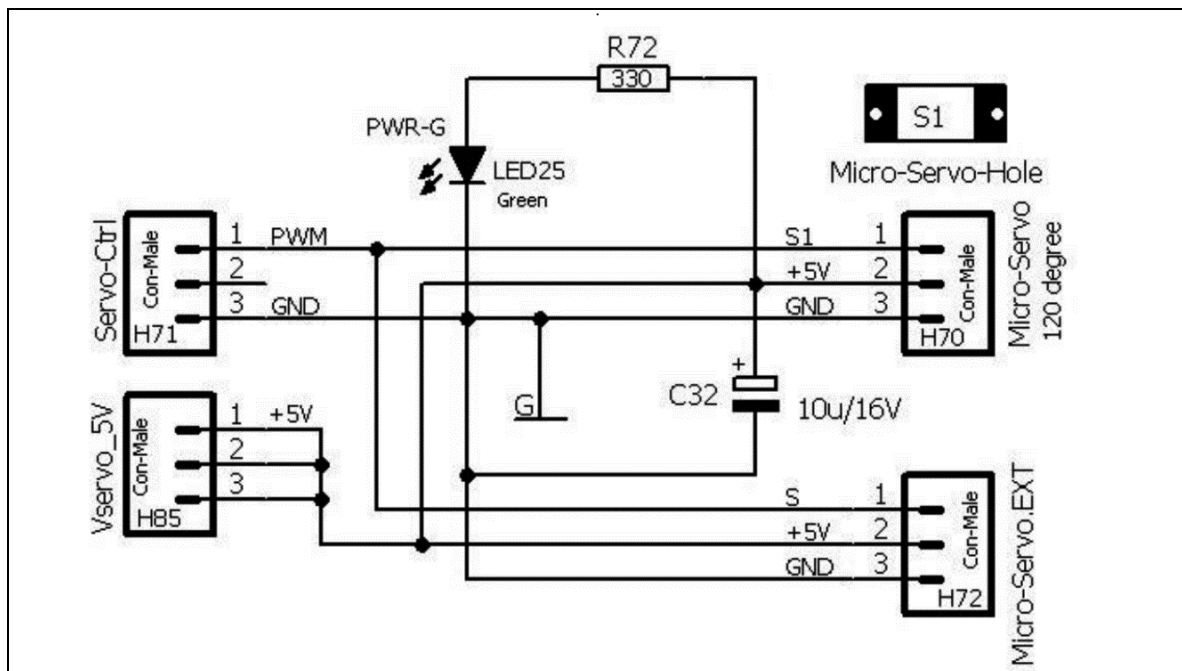
$$\begin{aligned} v_{\text{digital}} &= \frac{2}{20} \times 1024 && \text{สมการที่ 6-3} \\ &= 102.4 \end{aligned}$$

โดย v_{digital} คือ ค่าที่ส่งไปยัง pinx
 $\text{Analog}_{\text{resolution}}$ คือ ค่าสูงสุดของสัญญาณแอนะล็อกที่ระบบกำลังใช้งาน

> วงจรและการทำงาน

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

วงจรของ LAB-MICRO SERVO เป็นดังภาพที่ 6-3 จะพบว่า LED25 เป็นไฟแสดงสถานะพร้อมทำงานของบอร์ด มีขั้วต่อเข้ากับไมโครเซอร์โว H70 ซึ่งหมุนได้ 120 องศา มีขั้วต่อ H71 สำหรับเชื่อมต่อกับขาของ ESP32 เพื่อควบคุมเซอร์โว (ตัวอย่างโปรแกรมของบทนี้ใช้ขา P2) ขาสำหรับจ่ายแรงดัน 5 โวลต์สำหรับขับเคลื่อนมอเตอร์ผ่านทางขั้วต่อ H85 และรองรับการต่อเซอร์โวภายนอกผ่านทางขั้วต่อ H72



ภาพที่ 6-3

วงจรของ LAB-MICRO SERVO

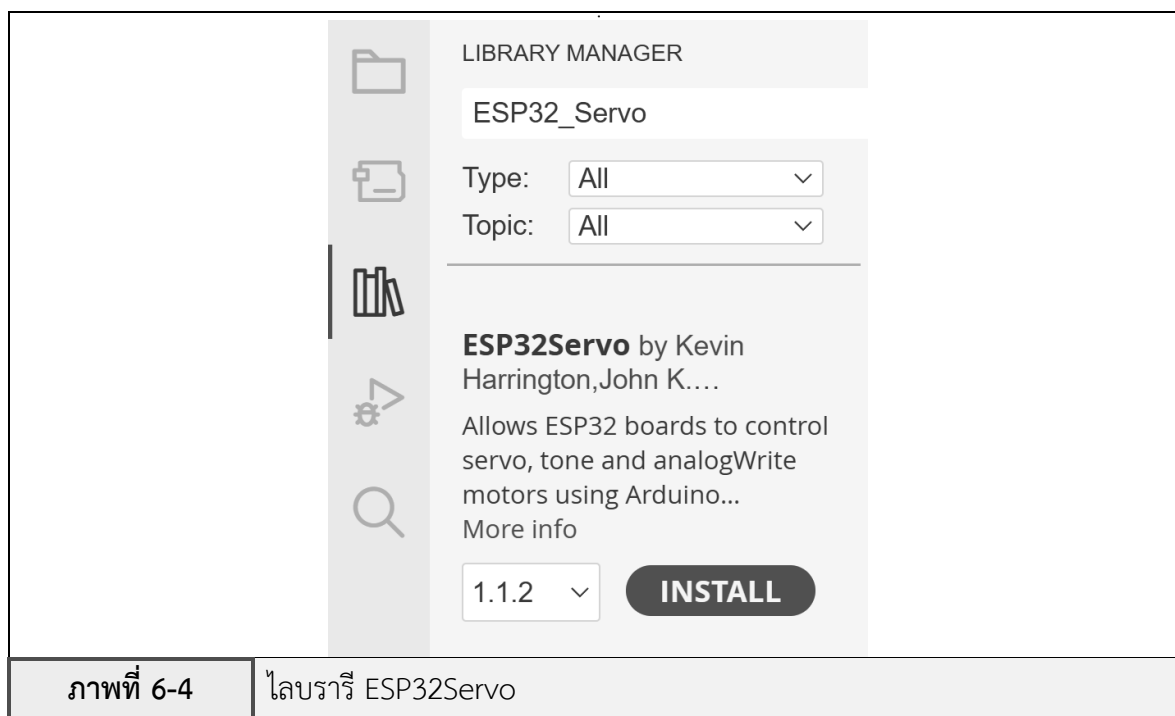
ภาพโดย จารุต บุศราทิจ

การเขียนโปรแกรมเพื่อควบคุมเซอร์โวสามารถเขียนด้วยตนเองผ่านทาง PWM ที่ได้กล่าวถึงในบทที่ 2 หรือใช้ไลบรารีสำเร็จรูปที่หนังสือเล่มนี้เลือกใช้คือ ESP32_Servo

> ไลบรารี ESP32_Servo

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ไลบรารี ESP32_Servo พัฒนาโดย Kevin Harrington และ John K. Bennett ด้วยการใช้ PWM ของ ESP32 ได้สูงสุด 16 ตัว ซึ่งการใช้ไลบรารีต้องติดตั้งผ่านทาง Library Manager ดังภาพที่ 6-4



ภาพที่ 6-4

ไลบรารี ESP32Servo

ภาพโดย จารุต บุศราทิจ

หลังจากติดตั้งไลบรารีจะต้องเรียกใช้ไฟล์ส่วนหัวชื่อ ESP32_Servo.h ดังนี้

```
#include <ESP32Servo.h>
```

การสร้างวัตถุเพื่ออ้างอิงไปยังเซอร์โวดึงต้องประกาศตัวแปรที่เป็นประเภท Servo ดังนี้

```
Server ตัวแปรอ้างอิง;
```

เมื่อได้ตัวแปรอ้างอิงหรือวัตถุที่สร้างจากคลาส Servo แล้ว ขั้นตอนต่อไปคือกำหนดการเชื่อมต่อกับ หมายเลขขา ของ ESP32 ด้วยคำสั่ง attach() ในฟังก์ชัน setup() ตามรูปแบบของการใช้งานต่อไปนี้

```
ตัวแปรอ้างอิง.attach( หมายเลขขา )
```

การสั่งให้แขนของเซอร์โวมอเตอร์ไปยังตำแหน่งมุมที่ต้องการต้องเรียกใช้คำสั่ง `write()` ตามด้วยค่าตำแหน่ง มุมตามรูปแบบของคำสั่งดังนี้

```
ตัวแปรอ้างอิง.write( ตำแหน่ง )
```

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 6-1 เป็นการสั่งงานเซอร์โวมอเตอร์ให้หมุนทวนเข็มนาฬิกา กลับมาจุดเริ่มต้น หมุนตามเข็มนาฬิกา และกลับมาจุดเริ่มต้น หลังจากนั้นวนไปเรื่อย ๆ จนกว่าผู้ใช้จะกดปุ่ม `button_b` เพื่อหยุดหรือให้ทำงานต่อ โดยมีการตั้งค่าตำแหน่งในการหมุน ทิศในการหมุนการหมุน และสถานการณ์หมุนของเซอร์โวมอเตอร์ที่ 5, 6 และ 7 สั่งการหมุนไปตำแหน่งที่ต้องการในบรรทัดที่ 19-29 โดยค่อยๆ เพิ่มหรือลดตำแหน่งเพื่อให้มีการใช้คำสั่ง `delay` เป็นระยะเวลาน้อยที่สุด เนื่องจากอาจมีผลกระทบกับการใช้งานปุ่มกดหรืออุปกรณ์อื่นๆ และมีการตรวจสอบตำแหน่งเพื่อปรับเปลี่ยนทิศทางในการหมุนในบรรทัดที่ 30-34

ตัวอย่างโปรแกรม 6-1 สั่งงานเซอร์โวมอเตอร์ให้หมุนทวนเข็มนาฬิกา	
บรรทัด	คำสั่ง
1	#include <ESP32Servo.h>
2	#define servo 2
3	#define button_b 17
4	Servo servo1;
5	int position = 180;
6	int stage = 1;
7	bool rotate = true;
8	void setup()
9	{
10	pinMode(button_b, INPUT);
11	servo1.attach(servo);
12	}
13	void loop()
14	{

ตัวอย่างโปรแกรม 6-1 สั่งงานเซอร์โวให้หมุนทวนเข็มนาฬิกา (ต่อ)

บรรทัด	คำสั่ง
15	if(digitalRead(button_b) == LOW){
16	rotate = !rotate;
17	delay(300);
18	}
19	if(rotate){
20	if(stage == 1 && position < 180){
21	servo1.write(position);
22	position++;
23	delay(5);
24	}else if(stage == 2 && position > 0){
25	servo1.write(position);
26	position--;
27	delay(5);
28	}
29	}
30	if(position == 180){
31	stage = 2;
32	}else if(position == 0){
33	stage = 1;
34	}
35	if(position == 90){
36	delay(500);
37	}
38	}

ตัวอย่างโปรแกรม 6-2 เป็นการสั่งงานให้เซอร์โวมอเตอร์หมุนตามเข็มนาฬิกาเมื่อระยะห่างที่วัดได้จากเซ็นเซอร์อัลตราโซนิก (กล่าวถึงในบทที่ 15) ต่ำกว่า 30 เซนติเมตร และหมุนทวนเข็มนาฬิกาเมื่อระยะห่างที่วัดได้ของเซ็นเซอร์อัลตราโซนิกมากกว่าหรือเท่ากับ 30 เซนติเมตร โดยมีการตั้งค่าเริ่มต้นให้กับเซ็นเซอร์อัลตราโซนิกในบรรทัดที่ 9-13 วัดระยะห่างในบรรทัดที่ 19-25 และสั่งให้เซอร์โวมอเตอร์หมุนตามทิศที่กำหนดในบรรทัดที่ 26-30

ตัวอย่างโปรแกรม 6-2 ควบคุมการหมุนซ้ายขวาด้วยสวิตช์

บรรทัด	คำสั่ง
1	#include <ESP32Servo.h>
2	#include <RCWL_1X05.h>
3	#define servo 2
4	Servo servo1;
5	RCWL_1X05 ultrasonic1;
6	int position = 90;
7	void setup()
8	{
9	Wire.begin();
10	ultrasonic1.begin();
11	ultrasonic1.setFilter(true);
12	ultrasonic1.setTimeout(50);
13	ultrasonic1.setMode(RCWL_1X05::continuous);
14	servo1.attach(servo);
15	servo1.write(90);
16	}
17	void loop()
18	{
19	float cm ;
20	bool data ;
21	data = ultrasonic1.update();
22	if(data)
23	{
24	cm = ultrasonic1.read();
25	cm = cm/10;
26	if(cm < 30){
27	servo1.write(0);
28	}else{
29	servo1.write(180);
30	}
31	}
32	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

วิธีการสั่งงานเซอร์โวมอเตอร์ที่ติดตั้งกับโมดูล LAB-MICRO SERVO ต้องกำหนดค่าดีวีของพัลส์ที่มีค่าคาบเวลา 20 มิลลิวินาที โดยการกำหนดสัญญาณดิจิทัล 1 เป็นเวลา 1.5 มิลลิวินาทีเป็นการสั่งให้มอเตอร์หมุนไปที่กึ่งกลาง และการให้ค่าที่น้อยกว่าและมากกว่ามีผลต่อการหมุนทวนเข็มนาฬิกาและตามเข็มนาฬิกาที่แตกต่างกันขึ้นอยู่กับรุ่นของเซอร์โว

สิ่งหนึ่งที่ผู้อ่านต้องพึงระวังในการใช้เซอร์โวมอเตอร์ คือ ความผิดพลาดที่เกิดจากมอเตอร์ต้องการกระแสไฟที่มากกว่าที่บอร์ด ESP32 จะขับได้ (40mA) อาจส่งผลทำให้พอร์ตที่เชื่อมต่อได้รับความเสียหายได้ ดังนั้น การใช้งานเซอร์โวมอเตอร์ต้องดูให้มั่นใจว่ามีแรงดันไฟฟ้าที่ถูกต้องและมีกระแสที่เพียงพอต่อการขับมอเตอร์ให้หมุนได้

บรรณานุกรม

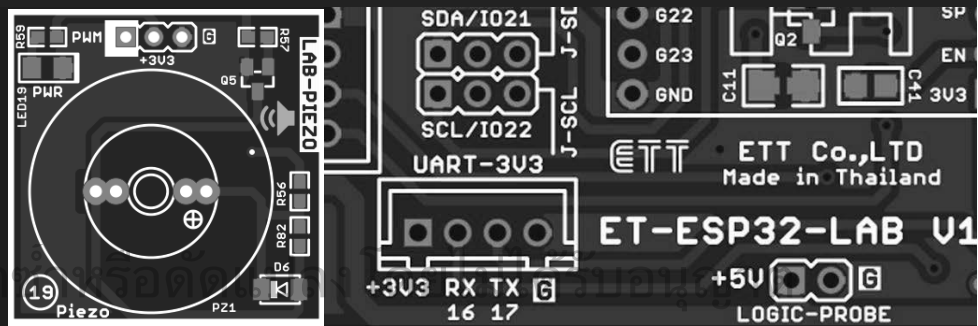
John K. Bennett. (2017). *ESP32Servo*. สืบค้นเมื่อ มีนาคม 31, 2567, จาก

<https://github.com/madhephaestus/ESP32Servo>



บทที่ 7

LAB-PIEZO

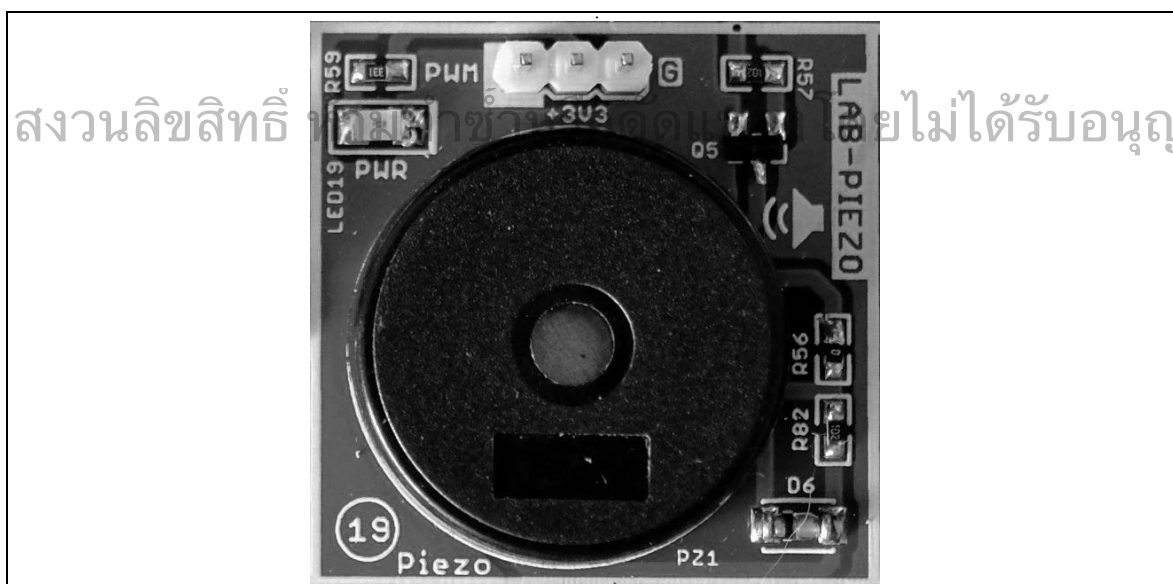


ในบทนี้กล่าวถึงการใช้อุปกรณ์แปลงเสียง หรือลำโพงขนาดเล็กที่เรียกว่าบัสเซอร์แบบเพียโซ (Piezo Buzzer) เพื่อใช้เป็นอุปกรณ์ให้เสียงเตือน หรือเสียงดนตรีเพื่อใช้เป็นสื่อตอบสนอง การทำงานกับผู้ใช้ โดยในบทนี้กล่าวถึงหลักการทำงานของวงจร หลักการสร้างคลื่นเสียง และตัวอย่างการเขียนโปรแกรมโดยใช้บอร์ดไมโครคอนโทรลเลอร์ ESP32 เมื่อจบบทนี้ ผู้อ่านจะเข้าใจถึงขั้นตอนการเขียนโปรแกรม และวิธีการเขียนโปรแกรมเพื่อส่งสัญญาณเสียงไปยังโมดูลลำโพงขนาดเล็ก หรือวงจรเชื่อมต่อกับลำโพงด้วย ESP32 เพื่อนำไปประยุกต์ใช้ต่อไป

> แนะนำ Piezo Buzzer

ภาคลำโพงขนาดเล็กหรือเพียโซบัสเซอร์ (LAB-PIEZO) ของบอร์ด ET-ESP32 LAB V1 เป็นวงจรขับเสียงออกลำโพงดังภาพที่ 7-1 โดยมีขา 3 ขาสำหรับต่อกับบอร์ด ESP32 คือ ขาไฟ (+V) ขากราวด์ (G) และขาสัญญาณ PWM สำหรับส่งสัญญาณให้กับลำโพง โดยตัวลำโพงรองรับความถี่สูงสุด 4KHz และโมดูล ESP32 สามารถใช้โมดูล PWM ในการสร้างคลื่นความถี่ได้ในช่วง 1KHz ถึง 10KHz อันครอบคลุมช่วงความถี่เสียงโน้ตดนตรี

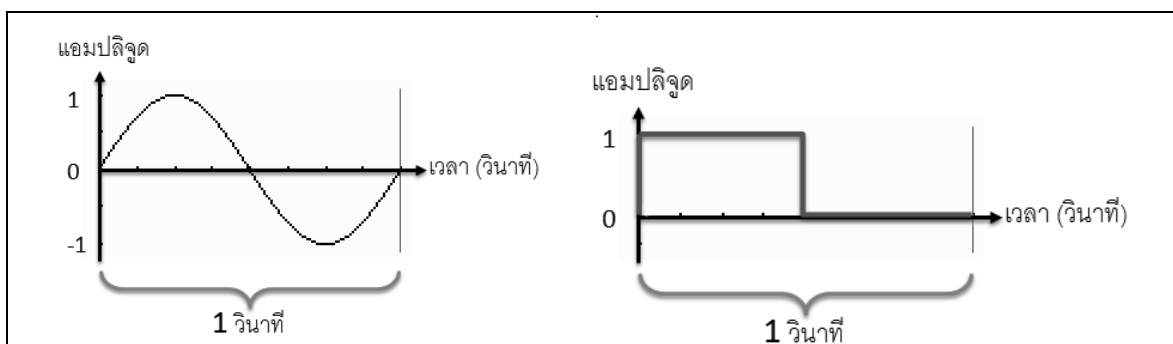
ความถี่เป็นอัตราการเกิดคลื่นต่อหนึ่งหน่วยเวลา โดยปกติใช้หน่วยเป็นเฮิรตซ์ (Hz) ซึ่งเป็นอัตราการเกิดลูกคลื่นต่อ 1 วินาที นั่นหมายความว่า ถ้าคลื่นมีความถี่ 1 เฮิรตซ์แล้วจะมีลูกคลื่นเต็มลูก 1 ลูกคลื่นในเวลา 1 วินาที ดังภาพที่ 7-2 ภาพซ้าย และเมื่อนำคลื่น 1 เฮิรตซ์มาแปลงเป็นสัญญาณดิจิทัลที่มีค่า 0 และ 1 จึงทำให้แอมพลิจูด (Amplitude) ของคลื่นมีค่าเป็น 0 กับ 1 จะได้รูปคลื่นแบบดิจิทัลดังภาพที่ 7-2 ภาพขวา



ภาพที่ 7-1

ภาพโมดูล LAB-PIEZO ของ ET-ESP32 LAB V1

ภาพโดย จารุต บุศราทิจ



ภาพที่ 7-2

(ซ้าย) ความถี่ 1Hz (ขวา) ความถี่ 1Hz แบบดิจิทัล

ภาพโดย จารุต บุศราทิจ

เมื่อเขียนโปรแกรมเพื่อสร้างความถี่ 1Hz สามารถเขียนสั่งงานให้ เกิดลูกคลื่น 1 ลูกจะใช้คาบเวลา 1 วินาที และมีส่วนของที่มีค่าเป็น 1 เป็นระยะเวลา 500 มิลลิวินาที และเป็นส่วนของที่มีค่าเป็น 0 อีก 500 มิลลิวินาที เมื่อนำมาเขียนเป็นสมการที่แสดงความสัมพันธ์ระหว่างความถี่กับคาบเวลาได้ตามสมการที่ 7-1 คือ

$$F = \frac{1}{T}$$

สมการที่ 7-1

โดย F คือ ความถี่ (หน่วยเป็น Hz)

T คือ คาบเวลาของคลื่น 1 ลูก (หน่วยเป็นวินาที)

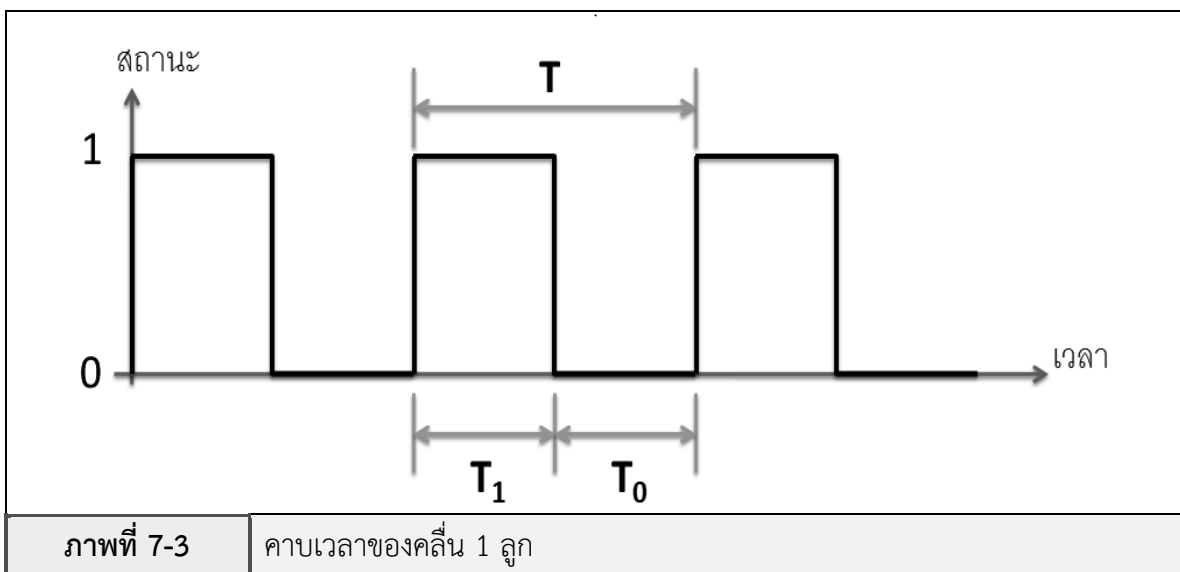
ถ้าลูกคลื่นมีคาบเวลาเป็น 0.001 วินาที หรือ 1 มิลลิวินาที จะทำให้เกิดคลื่นที่มีความถี่เป็น 1,000 เฮิรตซ์ หรือ 1 kHz ตามสมการที่ 7-2 ดังนี้

$$\begin{aligned} F &= \frac{1}{0.001} \text{ Hz} \\ &= 1000 \text{ Hz} \end{aligned} \quad \text{สมการที่ 7-2}$$

เนื่องจากในแต่ละคาบเวลาประกอบไปด้วยช่วงเวลาที่สถานะเป็น 1 และสถานะ 0 ดังภาพที่ 7-3 จะได้ความสัมพันธ์ของช่วงเวลาที่ทั้งสองกับคาบเวลาของการเกิดคลื่น 1 ลูก ดังสมการ 7-3

$$T = T_1 + T_0 \quad \text{สมการที่ 7-3}$$

โดย T คือ คาบเวลาที่เกิดคลื่น 1 ลูก (หน่วยเป็นวินาที)
 T_1 คือ ช่วงเวลาที่มีสถานะเป็น 1 (หน่วยเป็นวินาที)
 T_0 คือ ช่วงเวลาที่มีสถานะเป็น 0 (หน่วยเป็นวินาที)



ภาพโดย จารุต บุศราทิจ

จากที่กล่าวไปแล้ว เมื่อนำมาสร้างเป็นความถี่เสียงของตัวโน้ตดนตรี โด เร มี ฟา ซอล ลา ที หรือ C, D, E, F, G, A และ B จะได้ความถี่เป็น 262, 294, 330, 349, 392, 440 และ 494 Hz โดยมีค่า T_0 และ T_1 เป็นดังตารางที่ 6-1 จากตารางความถี่ของเสียงตัวโน้ต เมื่อนำมาคำนวณหาค่าคาบเวลาของ T_1 และ T_0 ทำได้ดังนี้

จาก

สมมติฐานที่ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

$$T = T_1 + T_0 \quad \text{วินาที}$$

เนื่องจากคาบเวลาของ T_1 และ T_0 มีค่าเท่ากัน จึงทำให้

$$T = 2 \times T_1 \quad \text{วินาที}$$

หรือ

$$T = 2 \times T_0 \quad \text{วินาที}$$

ด้วยเหตุนี้

$$F = \frac{1}{2 \times T_1} \quad \text{เฮิรตซ์} \quad \text{สมการที่ 7-4}$$

เนื่องจากทราบค่าความถี่ และต้องการหาค่าคาบเวลาของ T_1 เพื่อใช้ในการห้วงเวลา จึงได้สมการสำหรับหาค่า T_1 ดังนี้

$$\begin{aligned} T_1 &= \frac{1}{2 \times F} && \text{วินาที} \\ &= \frac{1}{2 \times 262} && \text{วินาที} \\ &= \frac{1}{524} && \text{วินาที} \\ &= 0.0001908396947 && \text{วินาที} \end{aligned}$$

หรือ

$$= 1,908.396947 \quad \text{ไมโครวินาทีวินาที}$$

ตารางที่ 7-1 ความถี่ของตัวโน้ตดนตรี

โน้ตดนตรี		ความถี่	คาบเวลาของ T_0 และ T_1 (ไมโครวินาที)
โด	C	262	1908
เร	D	294	1700
มี	E	330	1515
ฟา	F	349	1432
ซอล	G	392	1275

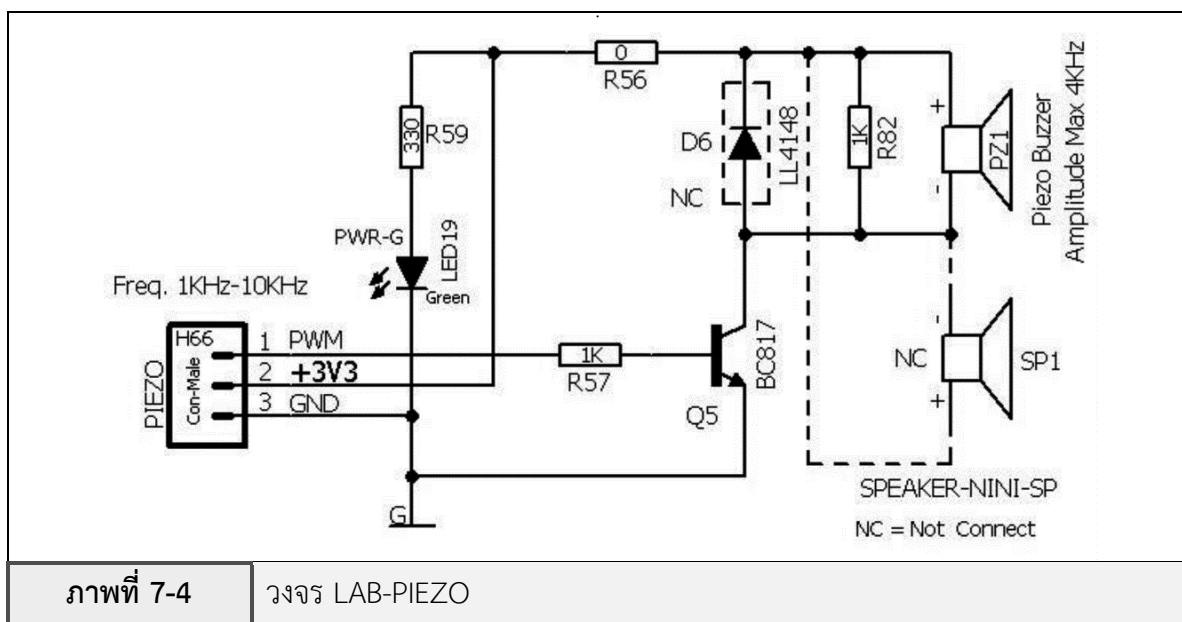
ตารางที่ 7-1 ความถี่ของตัวโน้ตดนตรี (ต่อ)

โน้ตดนตรี		ความถี่	คาบเวลาของ T0 และ T1 (ไมโครวินาที)
ลา	A	440	1136
ที	B	494	1012

ที่มา: ดัดแปลงจาก Peter Harrison. (n.d.). *Introduction to Music and Science*. สืบค้นเมื่อ เมษายน 3, 2567, จาก <https://pmcharrison.github.io/intro-to-music-and-science/pitch.html>

> วงจรและการทำงาน

จากภาพวงจรของ LAB-PIEZO ดังภาพที่ 7-4 หลอด LED19 สีเขียวเป็นไฟแสดงสถานะการทำงานของบอร์ด มีทรานซิสเตอร์ Q5 เบอร์ BC817 เป็นสวิตช์แปลงสัญญาณจากขา PWM เพื่อให้แรงดันที่ผ่านลำโพง PZ1 หรือ SP1 ลงกราวด์ และรองรับการต่อลำโพงขนาดเล็ก SP1 เพื่อใช้งานเพิ่มแทนเปียโซได้เช่นกัน



ภาพที่ 7-4

วงจร LAB-PIEZO

ภาพโดย จารุต บุศราทิจ

การเขียนโปรแกรมเพื่อสั่งงาน LAB-PIEZO สามารถทำได้ทั้งส่งสัญญาณเสียงเองด้วยการสลับค่าแรงดันหรือปิดด้วยความถี่ต่าง ๆ หรือใช้โมดูล PWM ของ ESP32 ช่วยสร้างสัญญาณความถี่ หรือใช้ไลบรารี toneESP32

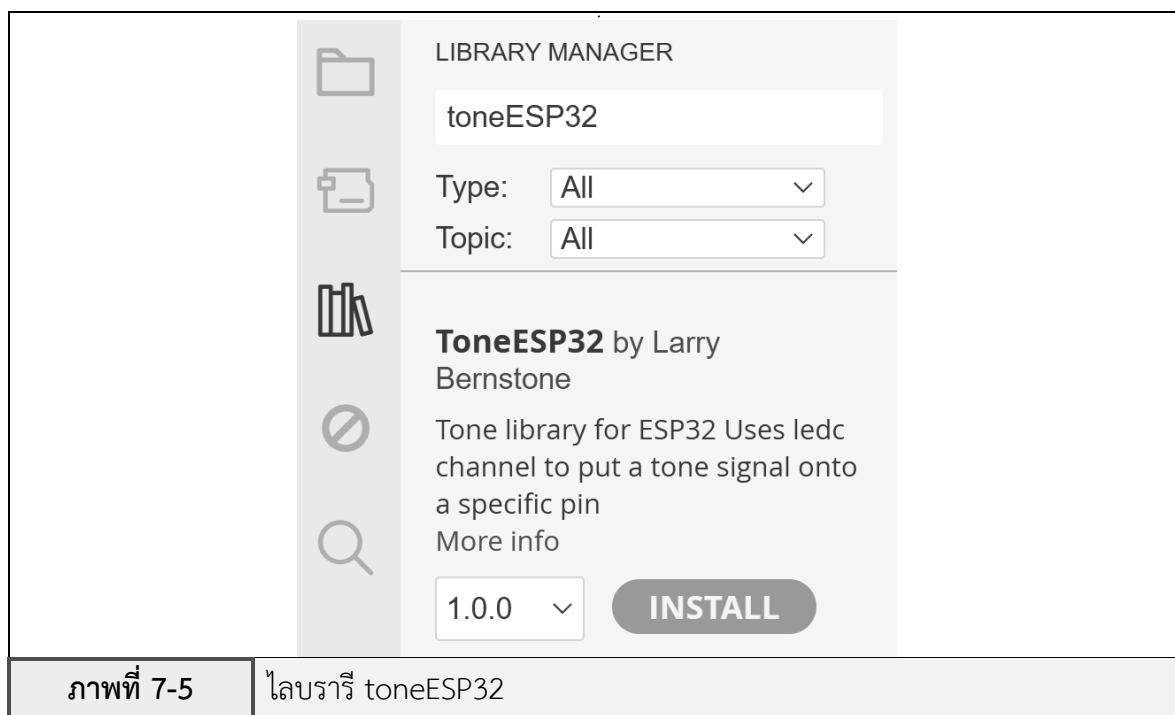
> ไลบรารี toneESP32

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ไลบรารี toneESP32 พัฒนาโดย Larry Bernstein ที่ผู้อ่านสามารถติดตั้งผ่านหน้าต่าง Library Manager ดังภาพที่ 7-5

เมื่อต้องการใช้งานไลบรารีต้องนำเข้าไฟล์ส่วนหัวชื่อ toneESP32.h ดังต่อไปนี้

```
#include <toneESP32.h>
```



ภาพที่ 7-5

ไลบรารี toneESP32

ภาพโดย จารุต บุศราทิจ

หลังจากนำเข้าไฟล์ส่วนหัวเสร็จสิ้น ผู้เขียนต้องสร้างวัตถุที่เป็นตัวแปรสำหรับอ้างอิงไปยังโมดูล LAB-PIEZO โดยสืบทอดจากคลาส toneESP32 พร้อมระบุหมายเลขขา และ ช่องสัญญาณ ดังรูปแบบการใช้งานต่อไปนี้

toneESP32 ชื่อวัตถุ(หมายเลขขา, ช่องสัญญาณ)

การส่งสัญญาณเสียงเมื่อมีตัวแปรอ้างอิงถึงโมดูลเสียงจากการสืบทอดจากคลาส toneESP32 ต้องเรียกใช้คำสั่ง tone() พร้อมค่า ตัวโน้ต และ ระยะเวลา ตามรูปแบบต่อไปนี้

ชื่อวัตถุ.tone(ตัวโน้ต, ระยะเวลา)

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

โดยที่

ตัวโน้ต เป็นค่าคงที่แทนความถี่ของเสียงโน้ตดนตรี อันได้แก่

NOTE_B0	NOTE_C1	NOTE_CS1	NOTE_D1	NOTE_DS1
NOTE_E1	NOTE_F1	NOTE_FS1	NOTE_G1	NOTE_GS1
NOTE_A1	NOTE_AS1	NOTE_B1	NOTE_C2	NOTE_CS2
NOTE_D2	NOTE_DS2	NOTE_E2	NOTE_F2	NOTE_FS2
NOTE_G2	NOTE_GS2	NOTE_A2	NOTE_AS2	NOTE_B2
NOTE_C3	NOTE_CS3	NOTE_D3	NOTE_DS3	NOTE_E3
NOTE_F3	NOTE_FS3	NOTE_G3	NOTE_GS3	NOTE_A3
NOTE_AS3	NOTE_B3	NOTE_C4	NOTE_CS4	NOTE_D4
NOTE_DS4	NOTE_E4	NOTE_F4	NOTE_FS4	NOTE_G4
NOTE_GS4	NOTE_A4	NOTE_AS	NOTE_AS4	NOTE_B4
NOTE_C5	NOTE_CS5	NOTE_D5	NOTE_DS5	NOTE_E5
NOTE_F5	NOTE_FS5	NOTE_G5	NOTE_GSH	NOTE_GS5
NOTE_A5	NOTE_AS5	NOTE_B5	NOTE_C6	NOTE_CS6
NOTE_D6	NOTE_DS6	NOTE_E6	NOTE_F6	NOTE_FS6
NOTE_G6	NOTE_GS6	NOTE_A6	NOTE_AS6	NOTE_B6
NOTE_C7	NOTE_CS7	NOTE_D7	NOTE_DS7	NOTE_E7
NOTE_F7	NOTE_FS7	NOTE_G7	NOTE_GS7	NOTE_A7
NOTE_AS7	NOTE_B7	NOTE_C8	NOTE_CS8	NOTE_D8
NOTE_DS8				

สำหรับการหยุดการส่งสัญญาณเสียงจะต้องเรียก noTone() ด้วยรูปแบบคำสั่งต่อไปนี้

ชื่อวัตถุ.noTone()

> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 7-1 เป็นการสร้างความถี่เสียงแบบดิจิทัลโดยเล่นเสียงโน้ตตั้งแต่ C4-C5 วนซ้ำไปเรื่อย ๆ มีการตั้งค่าเริ่มต้นให้กับ Piezo ในบรรทัดที่ 4 สร้างฟังก์ชันในการเล่นเสียงในบรรทัดที่ 5 – 9 และเล่นเสียงโน้ตตามกำหนดในบรรทัดที่ 14 ถึง 21

ตัวอย่างโปรแกรม 7-1 ตัวอย่างโปรแกรมสร้างความถี่เสียงแบบดิจิทัล

บรรทัด	คำสั่ง
1	#include <ToneESP32.h>
2	#define peizo 12
3	#define ch 0
4	ToneESP32 piezo1(peizo, ch);
5	void beep(int note, int duration) {
6	piezo1.tone(note, duration);
7	delay(duration*0.25);
8	}
9	void setup() {
10	}
11	void loop() {
12	beep(NOTE_C4, 250);
13	beep(NOTE_D4, 250);
14	beep(NOTE_E4, 250);
15	beep(NOTE_F4, 250);
16	beep(NOTE_G4, 250);
17	beep(NOTE_A4, 250);
18	beep(NOTE_B4, 250);
19	beep(NOTE_A5, 1000);
20	delay(1000);
21	}

ตัวอย่างโปรแกรม 7-2 เป็นการใช้คำสั่งสร้างโน้ตดนตรีเรียงกันเป็นเพลง โดยเรียงโน้ตและเล่นเพลงในบรรทัดที่ 14 - 36

ตัวอย่างโปรแกรม 7-2 ตัวอย่างโปรแกรมใช้คำสั่งสร้างโน้ตดนตรี

บรรทัด	คำสั่ง
1	#include <ToneESP32.h>
2	#define peizo 12
3	#define ch 0
4	ToneESP32 piezo1(peizo, ch);
5	void beep (int note, int duration) {
6	piezo1.tone(note, duration);
7	delay(duration*0.25);
8	}
9	void setup () {
10	}
11	void loop () {
12	beep(NOTE_GS5, 240);
13	beep(NOTE_E5, 240);
14	beep(NOTE_GS5, 240);
15	beep(NOTE_CS6, 240);
16	beep(NOTE_A5, 480);
17	delay(480);
18	beep(NOTE_FS5, 240);
19	beep(NOTE_DS5, 240);
20	beep(NOTE_FS5, 240);
21	beep(NOTE_B5, 240);
22	beep(NOTE_GS5, 480);
23	beep(NOTE_FS5, 240);
24	beep(NOTE_E5, 240);
25	delay(480);
26	beep(NOTE_E5, 240);
27	beep(NOTE_CS5, 240);
28	beep(NOTE_FS5, 480);
29	beep(NOTE_CS5, 480);
30	delay(480);
31	beep(NOTE_FS5, 240);
32	beep(NOTE_E5, 240);
33	beep(NOTE_GS5, 480);
34	beep(NOTE_FS5, 480);

ตัวอย่างโปรแกรม 7-2 ตัวอย่างโปรแกรมใช้คำสั่งสร้างโน้ตดนตรี (ต่อ)

บรรทัด	คำสั่ง
35	delay(3000);
36	}

ตัวอย่างโปรแกรม 7-3 เป็นการขับลำโพงด้วยการใช้ PWM เป็นภาคส่งความถี่ตัวโน้ต โดยเพิ่มเติมเรื่องของลักษณะของลูกคลื่นที่สร้างตามค่าที่กำหนดในตัวแปร instrument เป็นค่า 0, 1, 2 หรือ 3 เพื่อใช้ในการตรวจสอบสำหรับสร้างคลื่นสี่เหลี่ยม ฟันเลื่อย สามเหลี่ยม และคลื่นสมการไซน์ โดยโปรแกรมย่อยสำหรับแต่ละประเภท คือ playSquare(), playSaw(), playTriangle() และ playSine() ตามลำดับ

การกด sw1 ส่งผลให้มีการเปลี่ยนแปลงค่าตัวแปร instrument หรือเรียกได้ว่าเป็นปุ่มสำหรับเปลี่ยนรูปแบบเสียง และการกด sw2 เป็นการสั่งให้เล่นเสียงดนตรีตามรูปแบบของลูกคลื่นที่เลือกจาก sw1 โดยจะพบว่า แม้ใช้ค่าความถี่ของตัวโน้ตเดียวกัน แต่เมื่อเปลี่ยนลักษณะของลูกคลื่นจะส่งผลให้เสียงที่ออกมามีความแตกต่างกัน

ตัวอย่างโปรแกรม 7-3 ตัวอย่างโปรแกรม PWM

บรรทัด	คำสั่ง
1	#define sw1 32
2	#define sw2 33
3	#define spk 25
4	#define spkCh 0
5	#define volume 2048 // 50%
6	#define res 12
7	#define octave 4
8	#define time_ms 200
9	note_t notes[] = {
10	NOTE_C, NOTE_D, NOTE_E, NOTE_F,
11	NOTE_G, NOTE_A, NOTE_B
12	};
13	float notes_freq[] = {
14	261.63,293.66,329.63,349.23,392.00,440.00,493.88
15	};
16	int idx = 0;
17	int instruments = 0; // 0-square, 1-saw, 2-triangle, 3-sine

ตัวอย่างโปรแกรม 7-3 ตัวอย่างโปรแกรม PWM (ต่อ)

บรรทัด	คำสั่ง
18	int sinePWMValue(float freq);
19	void setup () {
20	Serial.begin(115200);
21	Wire.begin();
22	pinMode(sw1, INPUT);
23	pinMode(sw2, INPUT);
24	pinMode(spk, OUTPUT);
25	ledcSetup(spkCh, 1, res);
26	ledcAttachPin(spk, spkCh);
27	ledcWrite(spkCh, 0);
28	}
29	void playSine () {
30	ledcChangeFrequency(spkCh,
31	(uint32_t)notes_freq[idx], res);
32	for (int i = 0; i < time_ms; i++) {
33	float angle=sin(2*PI*((float)i/
34	(float)time_ms);
35	int sValue = 2048+2047*angle*
36	(1000000/notes_freq[idx]));
37	ledcWrite(spkCh, sValue);
38	delay(1);
39	}
40	}
41	void playSaw () {
42	ledcChangeFrequency(spkCh,
43	(uint32_t)notes_freq[idx], res);
44	for (int i = 1; i <= time_ms; i++) {
45	ledcWrite(spkCh,
46	(int)(volume*(time_ms/i)));
47	delay(1);
48	}
49	}
50	void playTriangle () {
51	ledcChangeFrequency(spkCh,

ตัวอย่างโปรแกรม 7-3 ตัวอย่างโปรแกรม PWM (ต่อ)

บรรทัด	คำสั่ง
52	(uint32_t)notes_freq[idx], res);
53	for (int i = 1; i <= time_ms / 2; i++) {
54	ledcWrite(spkrCh)(volume*
55	((time_ms/2)/i));
56	delay(1);
57	}
58	for (int i = 1; i <= time_ms / 2; i++) {
59	ledcWrite(spkrCh,volume-
60	(int)(volume*
61	((time_ms/2)/i));
62	delay(1);
63	}
64	}
65	void playSquare() {
66	ledcWriteNote(spkrCh, notes[idx], octave);
67	}
68	void loop() {
69	if (digitalRead(sw1)) { // change instrument
70	instruments++;
71	if (instruments > 3) {
72	instruments = 0;
73	}
74	delay(250);
75	}
76	if (digitalRead(sw2)) { // Play
77	for (idx = 0; idx < 7; idx++) {
78	switch (instruments) {
79	case 0: playSquare();
80	delay(time_ms); break;
81	case 1: playSaw(); break;
82	case 2: playTriangle(); break;
83	case 3: playSine(); break;
84	}
85	}

ตัวอย่างโปรแกรม 7-3 ตัวอย่างโปรแกรม PWM (ต่อ)

บรรทัด	คำสั่ง
86	ledcWrite(spkCh, 0);
87	}
88	}

ตัวอย่างโปรแกรม 7-4 มีการทำงานเหมือนกับตัวอย่างโปรแกรม 7-3 แต่เพิ่มเรื่องการเล่นเป็นเพลง โดยเก็บตัวโน้ตของเพลงไว้ในตัวแปร song() เมื่อลองรันโปรแกรมจะพบว่า การเปลี่ยนลักษณะของคลื่นเสียงทำให้เพลงมีความแตกต่างกันออกไป และสามารถนำแนวคิดนี้ไปสร้างเสียงสังเคราะห์เพื่อเลียนแบบเครื่องดนตรีได้

ตัวอย่างโปรแกรม 7-4 ตัวอย่างโปรแกรม PWM เล่นเสียงเพลง

บรรทัด	คำสั่ง
1	#define sw1 32
2	#define sw2 33
3	#define spk 25
4	#define spkCh 0
5	#define volume 2048 // 50%
6	#define res 12
7	#define octave 4
8	#define time_ms 90
9	#define c 261.63
10	#define d 293.66
11	#define e 329.63
12	#define f 349.23
13	#define g 392.00
14	#define a 440.00
15	#define b 493.88
16	#define s 0.0
17	float song[] = {
18	s,s,s,c,s,c,s,c,s,d,s,e,s,s,e,s,e,s,d,s,c,s,d,s,e,s,c,
19	s,s,s,s,s,s,e,s,e,s,e,s,f,s,g,s,s,g,s,s,f,s,e,s,f,
20	s,g,s,e,s,s,s,s,s,c,s,c,s,c,s,d,s,e,s,s,e,s,s,d,
21	s,c,s,d,s,e,s,c,s,s,s,s,s,c,s,c,s,c,s,d,s,e,s,s,e,
22	s,e,s,d,s,c,s,d,s,e,s,c,s,s,s

ตัวอย่างโปรแกรม 7-4 ตัวอย่างโปรแกรม PWM เล่นเสียงเพลง (ต่อ)

บรรทัด	คำสั่ง
23	};
24	int song_len = 0;
25	int idx = 0;
26	int instruments = 0;
27	int adcValue;
28	void setup() {
29	pinMode(sw1, INPUT);
30	pinMode(sw2, INPUT);
31	pinMode(spk, OUTPUT);
32	song_len = sizeof(song) / sizeof(float);
33	ledcSetup(spkCh, 1, res);
34	ledcAttachPin(spk, spkCh);
35	ledcWrite(spkCh, 0);
36	}
37	void playSine() {
38	if (song[idx] > 0.00) {
39	ledcChangeFrequency(spkCh,
40	(uint32_t)song[idx],res);
41	for (int i = 0; i < time_ms; i++) {
42	float angle=sin(2*PI*
43	((float)i/(float)time_ms);
44	int sValue=volume+(volume-1)*angle*
45	(1000000/song[idx]));
46	ledcWrite(spkCh, sValue);
47	delay(1);
48	}
49	ledcWrite(spkCh, 0);
50	} else {
51	ledcWrite(spkCh, 0);
52	delay(time_ms);
53	}
54	}
55	void playSaw() {
56	if (song[idx] > 0.0) {

ตัวอย่างโปรแกรม 7-4 ตัวอย่างโปรแกรม PWM เล่นเสียงเพลง (ต่อ)

บรรทัด	คำสั่ง
57	ledcChangeFrequency(spCh,
58	(uint32_t)song[idx], res);
59	for (int i = 1; i <= time_ms; i++) {
60	ledcWrite(spkCh,
61	(int)(volume * (time_ms / i));
62	delay(1);
63	}
64	ledcWrite(spkCh, 0);
65	} else {
66	ledcWrite(spkCh, 0);
67	delay(time_ms);
68	}
69	}
70	void playTriangle() {
71	if (song[idx] > 0.0) {
72	ledcChangeFrequency(spCh,
73	(uint32_t)song[idx], res);
74	for (int i = 1; i <= time_ms / 2; i++) {
75	ledcWrite(spCh,(int)(volume*
76	((time_ms/2)/i));
77	delay(1);
78	}
79	float val = (float)(time_ms/2.0);
80	for (int i = 1; i <= time_ms / 2; i++) {
81	ledcWrite(spCh,volume-(int)(volume*(val/i)));
82	delay(1);
83	}
84	ledcWrite(spkCh, 0);
85	} else {
86	ledcWrite(spkCh, 0);
87	delay(time_ms);
88	}
89	}

ตัวอย่างโปรแกรม 7-4 ตัวอย่างโปรแกรม PWM เล่นเสียงเพลง (ต่อ)

บรรทัด	คำสั่ง
90	void playSquare() {
91	if (song[idx] > 0.0) {
92	ledcChangeFrequency(spCh,
93	(uint32_t)song[idx],res);
94	ledcWrite(spCh, volume);
95	delay(time_ms);
96	} else {
97	ledcWrite(spCh, 0);
98	delay(time_ms);
99	}
100	}
101	void loop() {
102	if (digitalRead(sw1)) { // change instrument
103	instruments++;
104	if (instruments > 3) {
105	instruments = 0;
106	}
107	delay(250);
108	}
109	if (digitalRead(sw2)) { // Play
110	for (idx = 0; idx < song_len; idx++) {
111	switch (instruments) {
112	case 0: playSquare(); break;
113	case 1: playSaw(); break;
114	case 2: playTriangle(); break;
115	case 3: playSine(); break;
116	}
117	}
118	}
119	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

การส่งงานเพื่อสร้างเสียงสามารถทำได้โดยการส่งคลื่นเสียงเป็นความถี่ให้แก่โมดูลเปียโซโซนาร์ผ่านทางพอร์ตนำออกสัญญาณดิจิทัล ด้วยคำสั่ง `digital_write()` หรือใช้งานโมดูล PWM สำหรับสร้างความถี่ เพื่อสร้างเอฟเฟกต์เสียงของคลื่นสี่เหลี่ยม สามเหลี่ยม หรือไซน์ และสามารถนำไปประยุกต์ใช้เพื่อสร้างเสียงเตือน หรือเสียงดนตรีบอกสถานะการทำงานของระบบได้

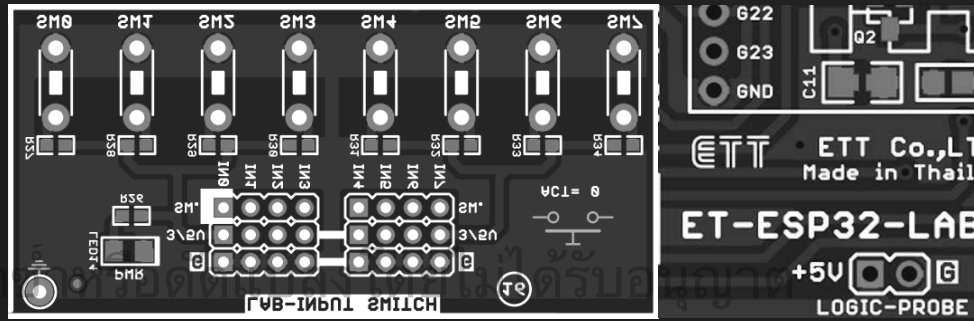
บรรณานุกรม

Larry Bernstone. (2017). *toneESP32*. สืบค้นเมื่อ เมษายน 3, 2567, จาก

<https://github.com/lbernstone/Tone32>

Peter Harrison. (n.d.). *Introduction to Music and Science*. สืบค้นเมื่อ เมษายน 3, 2567, จาก

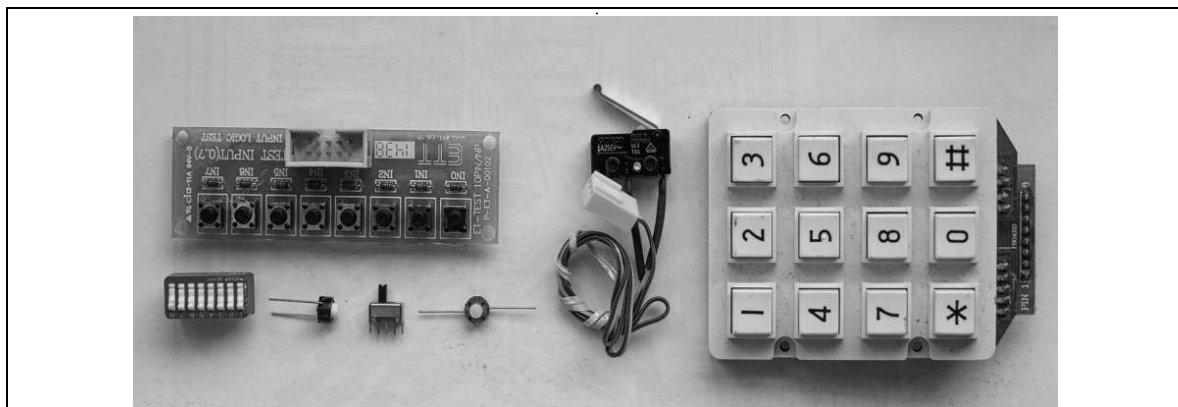
<https://pmcharrison.github.io/intro-to-music-and-science/pitch.html>



บทที่ 8 LAB-INPUT SWITCH

การนำเข้าข้อมูลจากอุปกรณ์ภายนอกประเภทหนึ่งที่ใช้กันมากที่สุด คือ การรับข้อมูลจากสวิตช์ ซึ่งมีหลากหลายแบบ เช่น แบบกดติดปล่อยดับ ดิพสวิตช์ (DIP Switch) และคีย์แพด ดังภาพที่ 8-1 ในบทนี้เป็นการใช้งานภาค LAB-INPUT SWITCH เป็นนำเข้าข้อมูลแบบสวิตช์จำนวน 8 ตัว (ภาพที่ 8-2) โดยตัวอย่างโปรแกรมเป็นการเขียนโปรแกรมเพื่อนำเข้าข้อมูลจาก การกดสวิตช์เพื่อทำงานอย่างใดอย่างหนึ่ง และการนำเข้าข้อมูลจากการกดสวิตช์ด้วยการดีเบาซ์ (debounce)

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรการทำงานของ LAB-INPUT SWITCH ไปประยุกต์ใช้งานเพื่อรองรับการกดสวิตช์เพื่อสั่งงานจากผู้ใช้ หรือ ตรวจสอบการชนของวัตถุที่มากระทบสวิตช์



ภาพที่ 8-1 สวิตช์แบบกดติดปล่อยดับ ดิพสวิตช์ และคีย์แพด

ภาพโดย จารุต บุศราทิจ

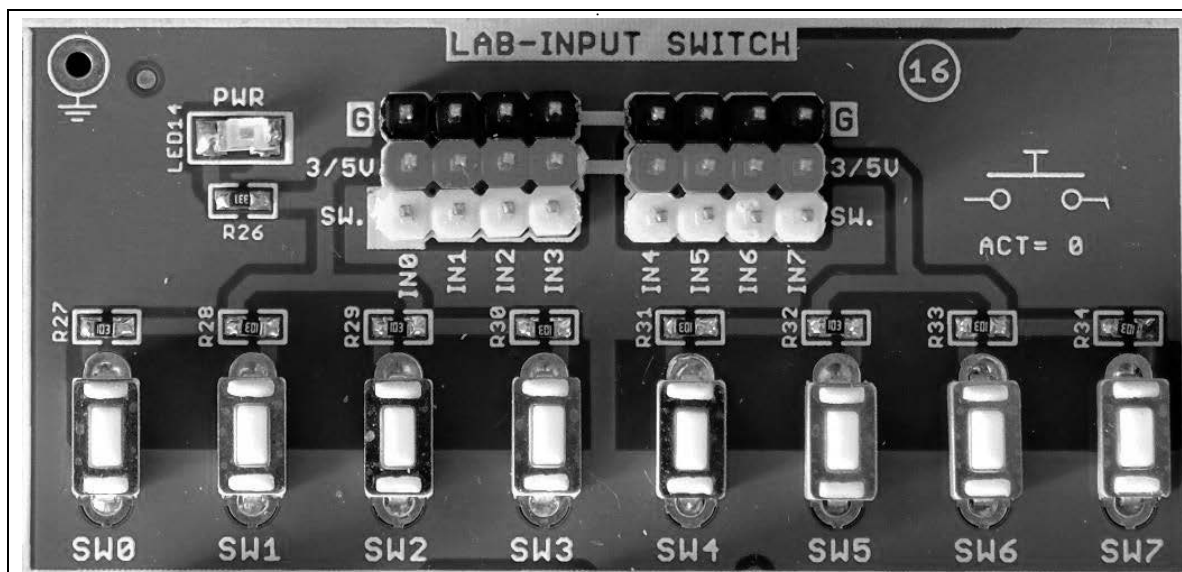
> แนะนำ LAB-INPUT SWITCH

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ภาค LAB-INPUT SWITCH เป็นโมดูลสำหรับทดลองการนำเข้าข้อมูลโดยใช้สวิตช์แบบกดติดปล่อยดับ ดังภาพที่ 8-2 โดยมีขาเชื่อมต่อ 8 ขา สำหรับ IN0, IN1, IN2, IN3, IN4, IN5, IN6 และ IN7 ตามชื่ออ้างอิงของสวิตช์แต่ละตัว พร้อมทั้งมีขั้วต่อ GND สำหรับต่อเข้ากับกราวด์ของระบบ V+ สำหรับต่อแรงดันกระแสตรง 3.3V-5V และ IO เป็นขาส่งข้อมูลกลับไปยังบอร์ด ESP32 โดยให้ผลลัพธ์ 2 กรณี คือ

สถานะเป็น 0 เมื่อมีการกดสวิตช์

สถานะเป็น 1 เมื่อมีการปล่อยสวิตช์



ภาพที่ 8-2

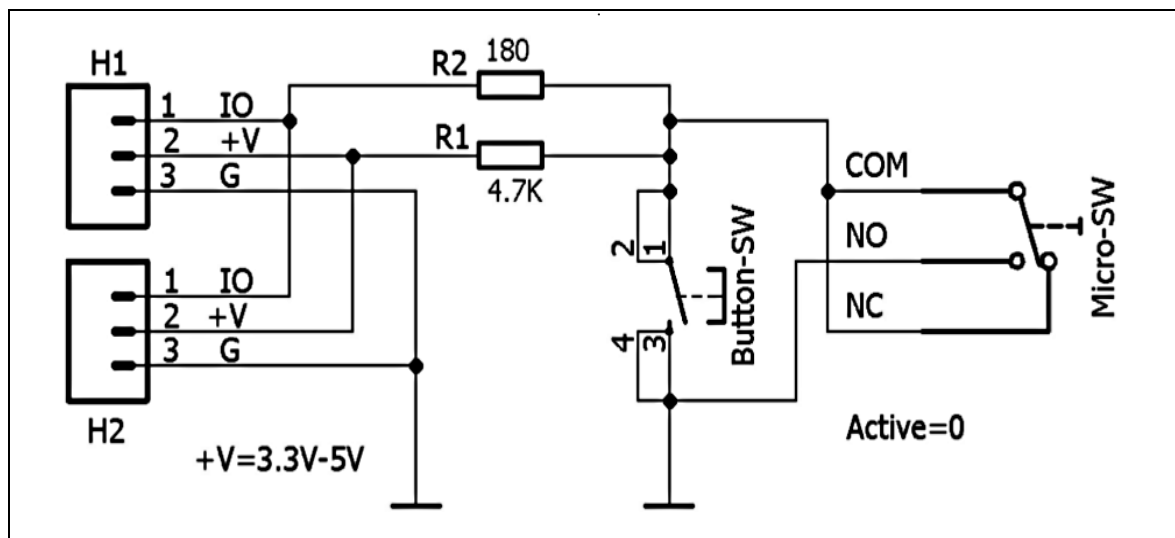
ภาพโมดูล LAB-INPUT SWITCH

ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

วงจรของภาค LAB-INPUT SWITCH เป็นดังภาพที่ 8-3 โดยสวิตช์กดติดปล่อยดับ และไมโครสวิตช์มีคุณลักษณะการใช้งาน คือ เมื่อมีการกดที่ตัวสวิตช์จะมีสถานะเป็นวงจรปิด และเมื่อปล่อยมือจากสวิตช์จะเปลี่ยนสถานะเป็นวงจรเปิด

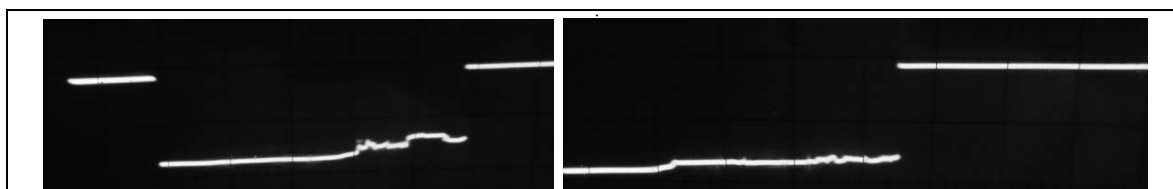
การเชื่อมประสานวงจรสวิตช์เข้ากับขาของไมโครคอนโทรลเลอร์ มีหลักการทำงาน คือ ถ้าสวิตช์ไม่ถูกกดสถานะที่ขาที่เชื่อมต่อ จะเป็น 1 แต่ถ้าสวิตช์ถูกกดจะทำให้สถานะที่ขา Pin1 จะเปลี่ยนจาก 1 เป็น 0 ซึ่งในช่วงของการเปลี่ยนสถานะนี้เองจะเกิดสัญญาณกระเพื่อมของแรงดัน (bounce) ดังภาพที่ 8-4



ภาพที่ 8-3 วงจรของ LAB-INPUT SWITCH

ภาพโดย จารุต บุศราทิจ

เพื่อป้องกันปัญหาสัญญาณที่เกิดจากหน้าสัมผัสของสวิตช์ที่สัมผัสกัน ซึ่งขณะที่สัมผัสนั้นแรงดันยังไม่เปลี่ยนแปลงในทันทีทันใด จึงเกิดการกระเพื่อมของแรงดันไฟฟ้ากระแสตรง โดยที่การกระเพื่อมในบางครั้งมีการกระเพื่อมในแรงดันระดับสูงแล้วลดลงเป็นต่ำและกระเพื่อมกลับขึ้นไปอีกสลับไปมาเป็นระยะเวลาหนึ่ง ส่งผลให้เสมือนมีการกดสวิตช์หลายครั้งทั้งที่กดเพียงครั้งเดียว



ภาพที่ 8-4 สัญญาณกระเพื่อมของแรงดัน (ซ้าย) กดแล้วปล่อย (ขวา) กดค้าง

ภาพโดย จารุต บุศราทิจ

ด้วยเหตุนี้จึงต้องมีวิธีการป้องกันการเกิดสถานการณ์การกระเพื่อม ซึ่งทำได้ 2 ลักษณะ คือ

1. ป้องกันด้วยฮาร์ดแวร์ โดยการเพิ่มบัฟเฟอร์ประเภทชmittริกเกอร์ (schmitt trigger) และตัวเก็บประจุในวงจรสวิตช์ดังภาพที่ 8-5
2. การป้องกันด้วยซอฟต์แวร์ โดยการหน่วงเวลามีขั้นตอนวิธีต่อไปนี้

ถ้ามีการกดสวิตช์ ให้ดำเนินการต่อไปนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

1. หน่วงเวลาเพื่อผ่านช่วงเวลาของการเกิดการกระเพื่อมของแรงดัน

2. ถ้ายังคงกดสวิตช์อยู่ ให้ดำเนินการต่อไปนี้

a. ดำเนินการใด ๆ เพื่อตอบสนองการกดสวิตช์

b. หน่วงเวลาเพื่อรอรับการกดสวิตช์ครั้งต่อไป

จากวงจรในภาพที่ 8-5 ต้องใช้รายการอุปกรณ์ดังต่อไปนี้

1. ตัวต้านทาน R1 ความต้านทาน $10\text{k}\Omega$

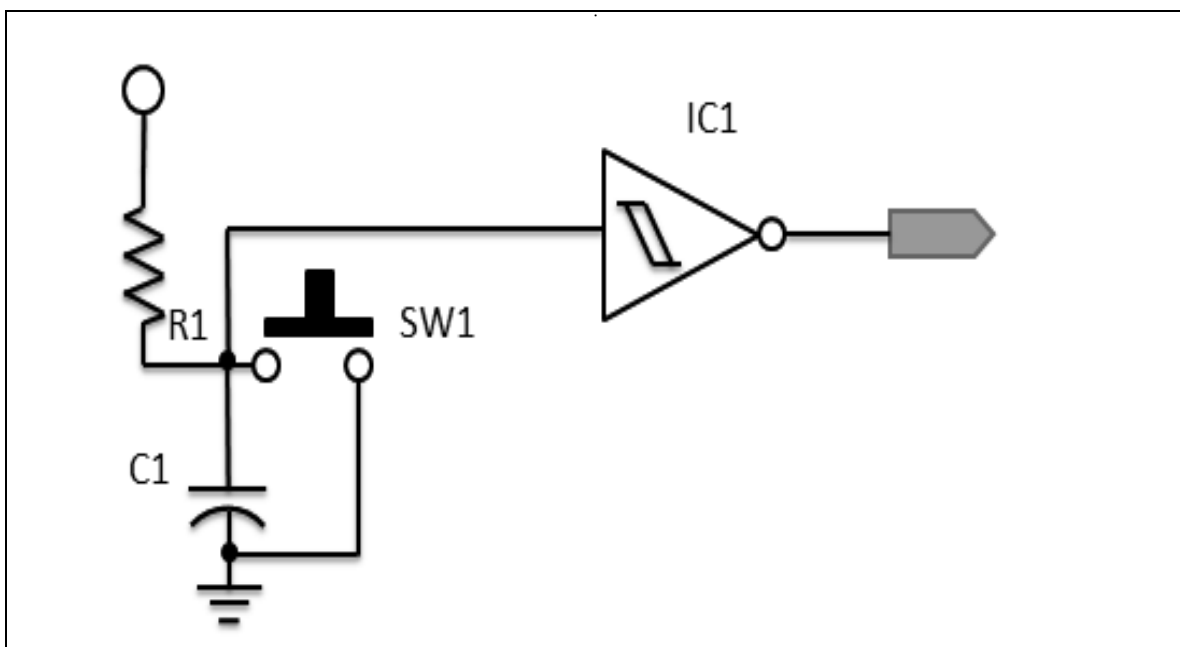
2. สวิตช์ SW1 แบบกดติดปล่อยดับ

3. ตัวเก็บประจุ C1 ความจุ 100 นาโนฟารัด (nF)

4. ไอซี IC1 เบอร์ 74LS14

ข้อดีของการใช้การป้องกันด้วยฮาร์ดแวร์ คือ ทำให้ไม่ต้องดำเนินการป้องกันด้วยวิธีการเขียนโปรแกรม แต่ข้อเสีย คือ ต้องเพิ่มต้นทุนของวงจรป้องกันการกระเพื่อมของแรงดัน

ข้อดีของการใช้การป้องกันด้วยซอฟต์แวร์ คือ ประหยัดต้นทุนในการผลิตฮาร์ดแวร์ แต่ข้อเสีย คือ โปรแกรมมีความซับซ้อนเพิ่มขึ้น



ภาพที่ 8-5

วงจรสำหรับสวิตช์กดติดปล่อยดับที่มีการป้องกันการกระเพื่อมของสัญญาณ

ภาพโดย จารุต บุศราทิจ

> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 8-1 แสดงหมายเลขของปุ่มที่ถูกกดโดยเมื่อผู้ใช้กดปุ่มใดหมายเลขของปุ่มจะถูกแสดงบนจอ OLED โดยใช้อะแดร์เก็บขาของปุ่มในบรรทัดที่ 5 เพื่อให้สามารถเรียกใช้ได้ด้วยการวนลูปเช่น กำหนดโหมดการทำงานของขาในบรรทัดที่ 9-11 และตรวจสอบการกดปุ่มและแสดงหมายเลขปุ่มที่ถูกกดในบรรทัดที่ 24-33 ขาที่ต่อกับปุ่มที่ 0-7 คือ ขา 26 27 32 33 34 35 36 และ 39 ตามลำดับ

ตัวอย่างโปรแกรม 8-1 แสดงหมายเลขของปุ่มที่ถูกกด

บรรทัด	คำสั่ง
1	#include <Adafruit_GFX.h>
2	#include <Adafruit_SH1106.h>
3	#define SDA 21
4	#define SCL 22
5	int btn[] = {26, 27, 32, 33, 34, 35, 36, 39};
6	int i;
7	Adafruit_SH1106 display(SDA, SCL);
8	void setup(){
9	for(i = 0; i < 8; i++){
10	pinMode(btn[i], INPUT);
11	}
12	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
13	display.clearDisplay();
14	display.display();
15	delay(100);
16	display.setTextSize(1);
17	display.setTextColor(WHITE);
18	display.setCursor(0,0);
19	Wire.begin();
20	display.println("Ready!");
21	delay(1000);
22	}
23	void loop(){
24	for(i = 0; i < 8; i++){
25	if(digitalRead(btn[i]) == LOW){
26	display.clearDisplay();

ตัวอย่างโปรแกรม 8-1 แสดงหมายเลขของปุ่มที่ถูกกด (ต่อ)

บรรทัด	คำสั่ง
27	<code>display.setCursor(0,25);</code>
28	<code>display.print("You pressed button ");</code>
39	<code>display.print(i);</code>
30	<code>display.display();</code>
31	<code>break;</code>
32	<code>}</code>
33	<code>}</code>
34	<code>}</code>

ตัวอย่างโปรแกรม 8-2 เป็นการทายตัวเลขที่ถูกสุ่มโดยระบบสุ่มเลขตั้งแต่ 0-7 และให้ผู้ใช้งานทายเลขโดยการกดปุ่มหมายเลขที่ต้องการ โดยสุ่มเลขด้วยฟังก์ชัน `rand()` ซึ่งจะได้ค่าสุ่มของตัวเลขจำนวนเต็มตั้งแต่ 0 จนถึงค่าสูงสุดที่ระบบรับได้ มอดูโล ด้วย 8 เพื่อเปลี่ยนให้เป็นค่าตั้งแต่ 0-7 ในบรรทัดที่ 21 และ 61 และตรวจสอบหมายเลขของปุ่มที่ถูกกดและแสดงผลผลลัพธ์ในบรรทัดที่ 37-59

ตัวอย่างโปรแกรม 8-2 ทายตัวเลขที่ถูกสุ่ม

บรรทัด	คำสั่ง
1	<code>#include <Adafruit_GFX.h></code>
2	<code>#include <Adafruit_SH1106.h></code>
3	<code>#define SDA 21</code>
4	<code>#define SCL 22</code>
5	<code>int btn[] = {26, 27, 32, 33, 34, 35, 36, 39};</code>
6	<code>int randomNumber;</code>
7	<code>int i;</code>
8	<code>Adafruit_SH1106 display (SDA, SCL);</code>
9	<code>void setup(){</code>
10	<code>for(i = 0; i < 8; i++){</code>
11	<code>pinMode(btn[i], INPUT);</code>
12	<code>}</code>
13	<code>display.begin(SH1106_SWITCHCAPVCC, 0x3C);</code>
14	<code>display.clearDisplay();</code>
15	<code>display.display();</code>
16	<code>delay(100);</code>
17	<code>display.setTextSize(1);</code>

ตัวอย่างโปรแกรม 8-2 ทายตัวเลขที่ถูกสุ่ม (ต่อ)

บรรทัด	คำสั่ง
18	display.setTextColor(WHITE);
19	display.setCursor(0,0);
20	Wire.begin();
21	randomNumber = rand() % 8;
22	display.print("Guess the number 0 - 7");
23	delay(1000);
24	}
25	void loop(){
26	if(i == 8){
27	display.clearDisplay();
28	display.setCursor(0,25);
39	display.print("Guess the number 0 - 7");
30	display.display();
31	}
32	for(i = 0; i < 8; i++){
33	if(digitalRead(btn[i]) == LOW){
34	break;
35	}
36	}
37	if(randomNumber == i){
38	display.clearDisplay();
39	display.setCursor(0,25);
40	display.println("Correct!");
41	display.print("Random number = ");
42	display.print(randomNumber);
43	display.println();
44	display.print("Button pressed ");
45	display.print(i);
46	display.display();
47	delay(2000);
48	}else if(i != 8){
49	display.clearDisplay();
50	display.setCursor(0,25);

ตัวอย่างโปรแกรม 8-2 ทายตัวเลขที่ถูกสุ่ม (ต่อ)

บรรทัด	คำสั่ง
51	<code>display.println("Wrong!");</code>
52	<code>display.print("Random number = ");</code>
53	<code>display.print(randomNumber);</code>
54	<code>display.println();</code>
55	<code>display.print("Button pressed ");</code>
56	<code>display.print(i);</code>
57	<code>display.display();</code>
58	<code>delay(2000);</code>
59	<code>}</code>
60	<code>if(i != 8){</code>
61	<code> randomNumber = rand() % 8;</code>
62	<code>}</code>
63	<code>}</code>

> สรุป

ภาค LAB-INPUT SWITCH เป็นโมดูลสวิตช์แบบกดติดปล่อยดับจำนวน 8 ตัว โดยทั้ง 8 ตัวต่อพ่วงเข้าวงจรเดียวกัน การกดตัวใดตัวหนึ่งเสมือนกดอีกตัวหนึ่ง ทั้งนี้เพื่อให้ผู้อ่าน ได้เรียนรู้การใช้สวิตช์ทั้ง 2 ประเภท อันมีหน้าที่เดียวกันแต่สถานการณ์ประยุกต์ใช้แตกต่างกัน เช่น แบบกดปุ่มแดงนั้นเหมาะกับการให้ผู้ใช้งานมองเห็น หรือสังเกตเห็นและกดที่ปุ่ม ขณะที่อีกแบบหนึ่งเหมาะสำหรับการฝังไว้ในตำแหน่งของการตรวจสอบการชนหรือกระแทก เช่น วางไว้ผนังด้านหลังประตู เมื่อมีคนเปิดประตูและประตูสัมผัสกับสวิตช์จึงส่งเสียงเตือน หรือวางสวิตช์ไว้บริเวณกันชนของหุ่นยนต์ เมื่อหุ่นยนต์สัมผัสกับสิ่งแวดล้อมและเกิดการกดสวิตช์หุ่นจะได้ทราบว่าเกิดเหตุการณ์ชนเกิดขึ้น เป็นต้น

การทำงานของโมดูล LAB-INPUT SWITCH ให้สัญญาณนำเข้าเป็นดิจิทัลเป็น 0 เมื่อสวิตช์ถูกกด และให้สัญญาณดิจิทัลเป็น 1 เมื่อสวิตช์ถูกปล่อย และมีขาสำหรับต่อเชื่อมสัญญาณ

บรรณานุกรม

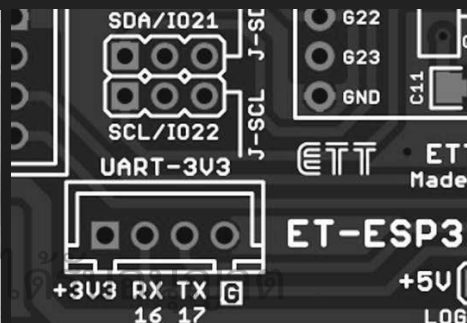
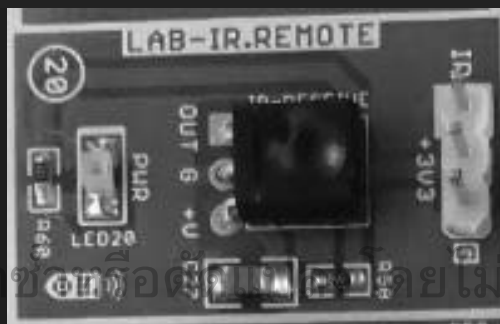
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ETT. (2024). *MANUAL - คู่มือ ET-ESP32-LAB V1.0*. สืบค้นเมื่อ เมษายน 6, 2567, จาก

<https://www.etteam.com/prodKitNLab/ET-ESP32-LABV1/th-man-ET-ESP32-LABV1.pdf>



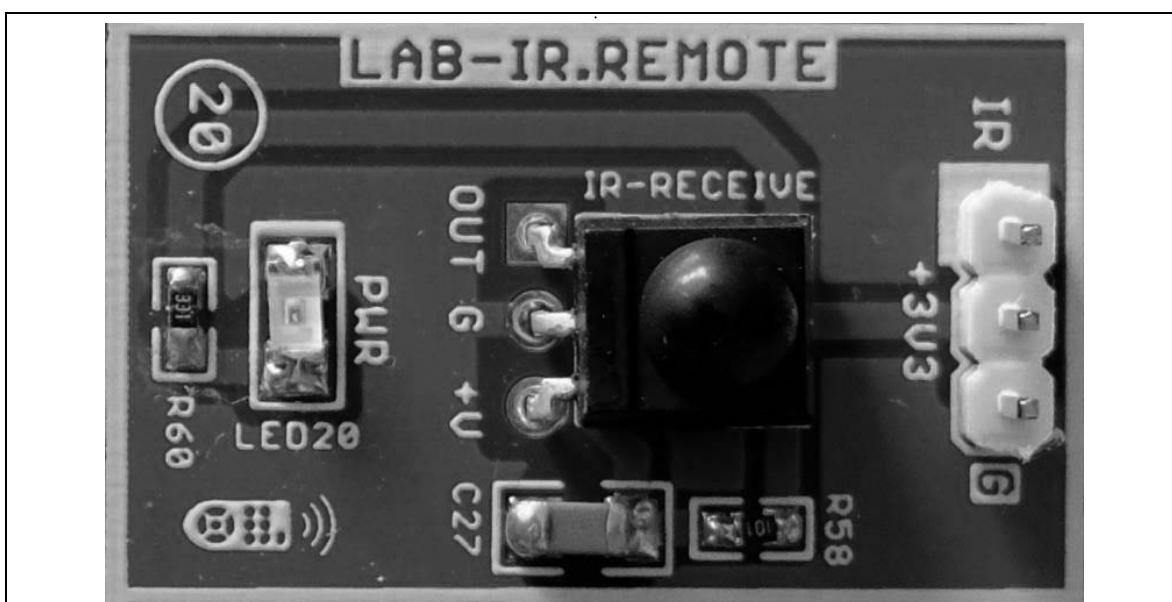
บทที่ 9 LAB-IR_REMOTE



โมดูล LAB-IR_REMOTE ดังภาพที่ 9-1 เป็นโมดูลนำเข้าสัญญาณคลื่นอินฟราเรดที่ใช้กับรีโมท ทรานส์มิทหรืออุปกรณ์ไฟฟ้าต่าง ๆ จากระยะไกล ดังภาพที่ 9-2 ผ่านทางไอซี TSO4838 โดยคำเรียก “รีโมทคอนโทรล” เป็นคำเรียกย่อจากคำว่า “รีโมตคอนโทรลเลอร์” (remote controller)

การส่งงานแต่ละคำสั่งประกอบด้วย 2 ส่วนเรียกว่า รหัส (code) และตัวส่งสัญญาณ (carrier) และฝั่งรับสัญญาณต้องถอดรหัสเพื่อนำไปใช้งานต่อ

ในบทนี้กล่าวถึงคุณสมบัติของ LAB-IR_REMOTE ทั้งฝั่งส่ง (ภาพที่ 9-2) และฝั่งรับ (ภาพที่ 9-1) พร้อมทั้งแนะนำวงจรและการทำงานของไลบรารี/ตัวอย่างโปรแกรมเพื่อให้ ESP32 สามารถรับและถอดรหัสที่ส่งมาจากภาคส่งเพื่อนำไปประยุกต์ใช้งานต่อไป



ภาพที่ 9-1

ภาพโมดูล LAB-IR_REMOTE

ภาพโดย จารุต บุศราทิจ



ภาพที่ 9-2

ภาพ Remote Control รุ่น ET-IR REMOTE KEY

ภาพโดย จารุต บุศราทิจ

> แนะนำ LAB-IR_REMOTE

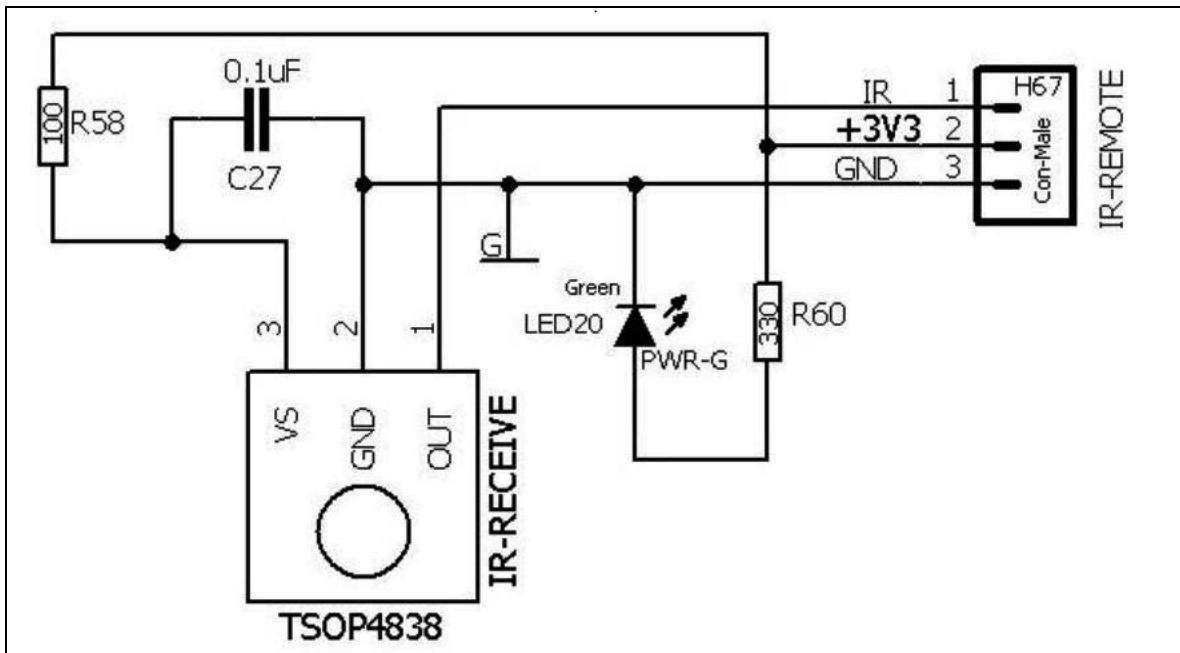
ภาค LAB-IR_REMOTE จากภาพที่ 9-1 เป็นโมดูลสำหรับทดลองรับสัญญาณที่มาจากอุปกรณ์ส่งสัญญาณในภาพที่ 9-2 โดยโมดูลนี้ใช้ TSOP4838 เป็นไอซีแบบ 3 ขาทำหน้าที่เป็นโมดูลภาครับแสงอินฟราเรด (IR) สำหรับใช้กับรีโมตคอนโทรล หรือรีโมตคอนโทรลเลอร์ โดยไอซีนี้มีคุณสมบัติดังนี้

- ทำงานที่ 38kHz
- มีขา 3 ขา สำหรับเป็นขานำออกสัญญาณ ขากราวด์ และขาจ่ายไฟแรงดันกระแสตรง
- ทนทานต่อสัญญาณรบกวน เช่น แสงไฟ แสงจากแอลซีดีของจอทีวี และคลื่นจาก WiFi เป็นต้น
- มีความต้องการกระแสที่ต่ำ 5mA
- ทำงานในอุณหภูมิ -25°C ถึง 85°C
- มีภาครับตรวจจับแสง (Photo detector) และภาคขยายสัญญาณ (preamplifier) ในตัว
- ทำงานที่แรงดันไฟฟ้ากระแสตรง 2.0V ถึง 5.5V
- รองรับรหัสของ NEC, RC5/RC6, Thonson RCA 56kHz, Sharp, Sony และ r-step
- ความเร็วในการรับสัญญาณสูงสุด 950burst/second

> วงจรและการทำงาน

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

วงจรของภาค LAB-IR_REMOTE เป็นดังภาพที่ 9-3 มีแอลอีดีสีเขียว (LED20) บอกสถานะการทำงานของบอร์ด มีขั้วต่อสำหรับเชื่อมต่อแบบ 3 ขั้ว (H67) สำหรับต่อแหล่งจ่ายไฟ กราวด์ และขา IR หรือขา P15 ของ ESP32



ภาพที่ 9-3

วงจรของ LAB-IR_REMOTE

ภาพโดย จารุต บุศราทิจ

เมื่อนำมาใช้กับภาคส่งในภาพที่ 9-2 หรือ ET-IR REMOTE KEY จะได้ว่า รหัสของแต่ละปุ่มกดนั้นเป็นดังนี้

- ปุ่ม Power มีค่าเป็น 0xFFA25D
- ปุ่ม MENU มีค่าเป็น 0xFFE21D
- ปุ่ม TEST มีค่าเป็น 0xFF22DD
- ปุ่ม + มีค่าเป็น 0xFF02FD
- ปุ่ม - มีค่าเป็น 0xFF9867
- ปุ่ม << มีค่าเป็น 0xFFE01F
- ปุ่ม >> มีค่าเป็น 0xFF906F
- ปุ่ม Play มีค่าเป็น 0xFFA857

- ปุ่ม Back มีค่าเป็น 0xFFC23D
- ปุ่ม C มีค่าเป็น 0xFFB04F
- ปุ่ม 0 มีค่าเป็น 0xFF6897
- ปุ่ม 1 มีค่าเป็น 0xFF30CF
- ปุ่ม 2 มีค่าเป็น 0xFF18E7
- ปุ่ม 3 มีค่าเป็น 0xFF7A85
- ปุ่ม 4 มีค่าเป็น 0xFF10EF
- ปุ่ม 5 มีค่าเป็น 0xFF38C7
- ปุ่ม 6 มีค่าเป็น 0xFF5AA5
- ปุ่ม 7 มีค่าเป็น 0xFF42BD
- ปุ่ม 8 มีค่าเป็น 0xFF4AB5
- ปุ่ม 9 มีค่าเป็น 0xFF52AD

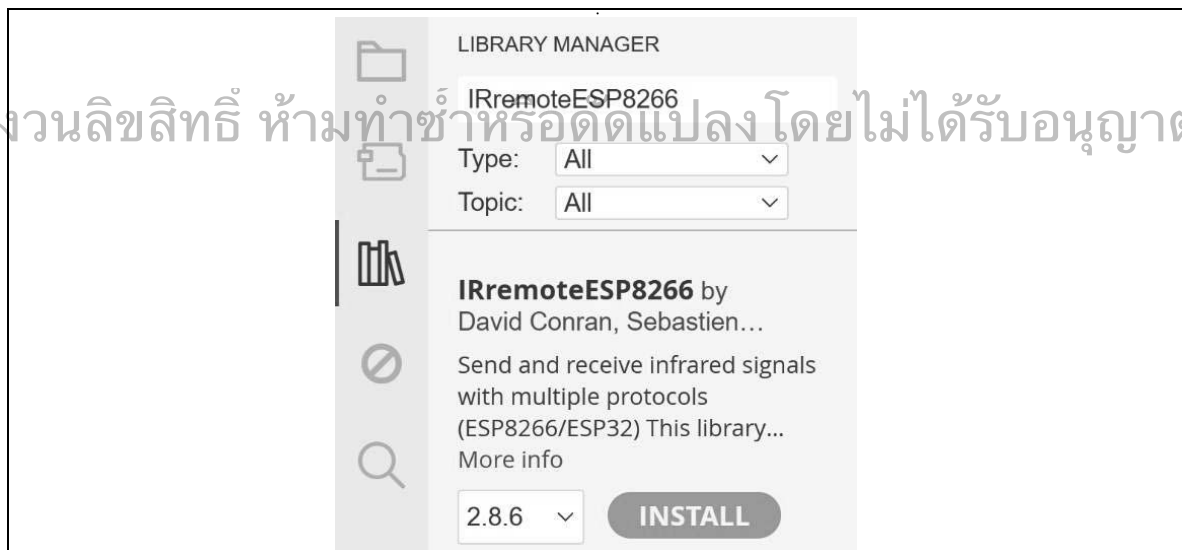
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

> ไลบรารี IRremoteESP8266

ไลบรารี IRremoteESP8266 พัฒนาโดย David Conran, Sebastien Warin, Mark Szabo และ Ken Shirriff เป็นไลบรารีที่ใช้งานกับ IR ย่าน 940nm ทั่วไป เช่น TSOP ตระกูล 17, 22, 24, 36, 38, 44 และ 48 เป็นต้น พร้อมทั้งโค้ดของไลบรารีรองรับการทำงานทั้งไมโครคอนโทรลเลอร์ ESP32 และ ESP8266 ซึ่งการใช้งานต้องติดตั้งเพิ่มเติมจาก Library Manager ดังภาพที่ 9-4

การใช้งานไลบรารีต้องเรียกใช้ไฟล์ส่วนหัว 2 ไฟล์ คือ IRremoteESP8266.h และ IRrecv.h ดังนี้ โดยไฟล์ IRremoteESP8266.h เก็บข้อมูลค่าคงที่ต่าง ๆ และ IRrecv.h เป็นส่วนหัวของคลาสไลบรารีตัวนี้

```
#include <IRremoteESP8266.h>
#include <IRrecv.h>
```

ภาพที่ 9-4 ไลบรารี IRremoteESP8266

ภาพโดย จารุต บุศราทิจ

การใช้งานต้องสร้างวัตถุประเภท IRrecv พร้อมระบุ หมายเลขขา ของ ESP32 ที่เชื่อมต่อกับขั้ว IR ดังรูปแบบของการใช้งานดังนี้

IRrecv ชื่อวัตถุ(หมายเลขขา)

หลังจากนั้นต้องสั่งเริ่มต้นทำงานในฟังก์ชัน setup() ของโปรแกรมที่เขียน ด้วยการเรียกคำสั่ง enableIRIn() ตามรูปแบบการใช้งานต่อไปนี้

ชื่อวัตถุ.enableIRIn()

การอ่านค่าต้องใส่ในส่วนของ loop() เพื่อให้เรียกทำงานทุกวงรอบการทำงาน โดยใช้คำสั่ง decode() ตามด้วยค่าตำแหน่งของ ตัวแปรเก็บรหัส ดังรูปแบบต่อไปนี้

ชื่อวัตถุ.decode(&ตัวแปรเก็บรหัส)

เมื่อได้รับรหัสที่ถูกถอดหรือแปลงจาก decode() ผู้เขียนโปรแกรมสามารถนำค่าจากตัวแปรเก็บรหัส ไปเปรียบเทียบกับค่าของแต่ละปุ่มของ ET-IR REMOTE KEY ที่ได้รับรู้ค่าคงที่ของแต่ละปุ่ม เพื่อใช้เป็นตัวเลือกสำหรับทำงานต่าง ๆ ต่อไป

สำหรับคำสั่ง `resume()` เป็นคำสั่งที่ต้องเรียกใช้งานหลังจากทำการ `decode()` เพื่อให้มีการรับค่าข้อมูลชุดถัดไป ดังรูปแบบของคำสั่งต่อไปนี้

```
ชื่อวัตถุ.resume( )
```

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 9-1 เป็นโปรแกรมเปลี่ยนช่องด้วยการกดรีโมตเป็นเลขช่องที่ต้องการหรือเพิ่มช่องโดยการกดปุ่มบวก ลดช่องโดยการกดปุ่มลบ ไปที่ช่องแรกสุดโดยการกดปุ่มลูกศรซ้าย และไปที่ช่องท้ายสุดโดยการกดปุ่มลูกศรขวา เลขจะถูกแสดงบนหน้าจอ OLED โดยมีการกำหนดค่าคีย์ของปุ่มลูกศรซ้าย ลูกศรขวา บวก ลบ และเลข 0-9 ในบรรทัดที่ 12-20 จากนั้นรับค่าคีย์ของปุ่ม ตรวจสอบการกดปุ่มที่กำหนดและแสดงผลบนจอ OLED โดยมีการตั้งค่านัยเวลาด้วยการตรวจสอบเวลาแทนการใช้ `delay` เพื่อไม่ให้มีผลกระทบต่อการใช้อุปกรณ์อื่น ในบรรทัดที่ 35-66 และต่อขา 15 เข้ากับโมดูล IR ขา และต่อขา 21 และ 22 เข้ากับ SDA และ SCL ของโมดูล OLED ตามลำดับ

ตัวอย่างโปรแกรม 9-1 เปลี่ยนช่องด้วยการกดรีโมต

บรรทัด	คำสั่ง
1	<code>#include <IRremoteESP8266.h></code>
2	<code>#include <IRrecv.h></code>
3	<code>#include <Adafruit_GFX.h></code>
4	<code>#include <Adafruit_SH1106.h></code>
5	<code>#define SDA 21</code>
6	<code>#define SCL 22</code>
7	<code>#define IR 15</code>
8	<code>IRrecv irRecv(IR);</code>
9	<code>decode_results results;</code>
10	<code>int channel = 0;</code>
11	<code>int i;</code>
12	<code>unsigned long keyCode ,last = millis();</code>
13	<code>unsigned long firstChannel = 0xFFE01F;</code>
14	<code>unsigned long lastChannel = 0xFF906F;</code>

ตัวอย่างโปรแกรม 9-1 เปลี่ยนช่องโดยการกดรีโมต (ต่อ)

บรรทัด	คำสั่ง
15	unsigned long plusChannel = 0xFF02FD;
16	unsigned long minusChannel = 0xFF9867;
17	unsigned long numberCodes[] = {
18	0xFF6897,0xFF30CF,0xFF18E7,0xFF7A85,0xFF10EF,
19	0xFF38C7,0xFF5AA5,0xFF42BD,0xFF4AB5,0xFF52AD
20	};
21	Adafruit_SH1106 display(SDA, SCL);
22	void setup(){
23	irRecv.enableIRIn();
24	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
25	display.clearDisplay();
26	display.setTextSize(1);
27	display.setTextColor(WHITE);
28	display.setCursor(0, 0);
29	display.print("Current Channel : ");
30	display.print(channel);
31	display.display();
32	Wire.begin();
33	}
34	void loop(){
35	if(irRecv.decode(&results)){
36	if(millis() - last > 250){
37	keyCode = results.value;
38	if(keyCode == firstChannel){
39	channel = 0;
40	}else if(keyCode == lastChannel){
41	channel = 9;
42	}else if(keyCode == plusChannel){
43	channel += 1;
44	if(channel > 9){
45	channel = 0;
46	}
47	}else if(keyCode == minusChannel){
48	channel -= 1;

ตัวอย่างโปรแกรม 9-1 เปลี่ยนช่องโดยการกดรีโมต (ต่อ)

บรรทัด	คำสั่ง
49	if(channel < 0){
50	channel = 9;
51	}
52	}
53	for(i = 0; i < 10; i++){
54	if(keyCode == numberCodes[i]){
55	channel = i;
56	}
57	}
58	display.clearDisplay();
59	display.setCursor(0, 0);
60	display.print("Current Channel : ");
61	display.print(channel);
62	display.display();
63	}
64	last = millis();
67	irRecv.resume();
66	}
67	delay(50);
68	}

ตัวอย่างโปรแกรม 9-2 เป็นการเล่นเพลงตามปุ่มกดโดยมี 3 เพลงคือ หนูมาลี ข้าง และแมงมุม ลายตัวนั้น ด้วยการกดปุ่มเลข 0-2 ตามลำดับ โดยมีการทำฟังก์ชันเก็บโน้ตของทั้งสามเพลงไว้ในบรรทัดที่ 21-135 จากนั้นรอรับการกดปุ่มที่รีโมตและแปลงโค้ดในบรรทัดที่ 140-148 และทำการตรวจสอบโค้ดของปุ่มและเล่นเพลงตามเลขที่กดในบรรทัดที่ 149-154 และต่อขา 15 เข้ากับโมดูล IR และขา 12 เข้ากับโมดูล Piezo

ตัวอย่างโปรแกรม 9-2 เล่นเพลงตามปุ่มกด

บรรทัด	คำสั่ง
1	#include <IRremoteESP8266.h>
2	#include <IRrecv.h>
3	#include <ToneESP32.h>
4	#define IR 15
5	#define Piezo 12
6	#define CH 0
7	IRrecv irRecv(IR);
8	decode_results results;
9	ToneESP32 piezo(Piezo ,CH);
10	int channel = 0;
11	int i;
12	unsigned long keyCode ,last = millis();
13	unsigned long numberCodes[] = {
14	0xFF6897,0xFF30CF,0xFF18E7,0xFF7A85,0xFF10EF,
15	0xFF38C7,0xFF5AA5,0xFF42BD,0xFF4AB5,0xFF52AD
16	};
17	void beep (int note, int duration){
18	piezo .tone(note, duration);
19	delay(duration*0.25);
20	}
21	void nooMalee () {
22	beep(NOTE_E4, 333);
23	beep(NOTE_D4, 166);
24	beep(NOTE_C4, 250);
25	beep(NOTE_D4, 250);
26	beep(NOTE_E4, 250);
27	beep(NOTE_E4, 250);
28	beep(NOTE_E4, 500);
29	beep(NOTE_D4, 250);
30	beep(NOTE_D4, 250);
31	beep(NOTE_D4, 500);
32	beep(NOTE_E4, 250);
33	beep(NOTE_G4, 250);
34	beep(NOTE_G4, 500);

ตัวอย่างโปรแกรม 9-2 เล่นเพลงตามปุ่มกด (ต่อ)

บรรทัด	คำสั่ง
35	beep(NOTE_E4, 333);
36	beep(NOTE_D4, 166);
37	beep(NOTE_C4, 250);
38	beep(NOTE_D4, 250);
39	beep(NOTE_E4, 250);
40	beep(NOTE_E4, 250);
41	beep(NOTE_E4, 500);
42	beep(NOTE_D4, 250);
43	beep(NOTE_D4, 250);
44	beep(NOTE_E4, 333);
45	beep(NOTE_D4, 333);
46	beep(NOTE_C4, 1000);
47	}
48	void chang(){
49	beep(NOTE_C5, 200);
50	beep(NOTE_C5, 200);
51	beep(NOTE_C5, 200);
52	beep(NOTE_C5, 200);
53	beep(NOTE_C5, 400);
54	beep(NOTE_C5, 200);
55	beep(NOTE_A4, 200);
56	beep(NOTE_G4, 200);
57	beep(NOTE_A4, 200);
58	beep(NOTE_C5, 200);
59	beep(NOTE_F4, 400);
60	beep(NOTE_C5, 200);
61	beep(NOTE_A4, 200);
62	beep(NOTE_G4, 200);
63	beep(NOTE_A4, 200);
64	beep(NOTE_F4, 200);
65	beep(NOTE_G4, 300);
66	beep(NOTE_F4, 200);
67	beep(NOTE_D4, 200);
68	beep(NOTE_F4, 200);

ตัวอย่างโปรแกรม 9-2 เล่นเพลงตามปุ่มกด (ต่อ)

บรรทัด	คำสั่ง
69	beep(NOTE_F4, 200);
70	beep(NOTE_C4, 200);
71	beep(NOTE_D4, 200);
72	beep(NOTE_F4, 400);
73	beep(NOTE_F4, 200);
74	beep(NOTE_D4, 300);
75	beep(NOTE_F4, 200);
76	beep(NOTE_F4, 300);
77	beep(NOTE_C4, 200);
78	beep(NOTE_D4, 300);
79	beep(NOTE_F4, 400);
80	beep(NOTE_C5, 300);
81	beep(NOTE_D4, 300);
82	beep(NOTE_C5, 300);
83	beep(NOTE_A4, 300);
84	beep(NOTE_G4, 400);
85	beep(NOTE_F4, 500);
86	}
87	void thatSpider () {
88	beep(NOTE_C4, 200);
89	beep(NOTE_C4, 200);
90	beep(NOTE_C4, 200);
91	beep(NOTE_D4, 200);
92	beep(NOTE_E4, 300);
93	beep(NOTE_E4, 200);
94	beep(NOTE_E4, 200);
95	beep(NOTE_D4, 200);
96	beep(NOTE_C4, 200);
97	beep(NOTE_D4, 200);
98	beep(NOTE_E4, 200);
99	beep(NOTE_C4, 500);
100	beep(NOTE_D4, 200);
101	beep(NOTE_D4, 200);
102	beep(NOTE_D4, 200);

ตัวอย่างโปรแกรม 9-2 เล่นเพลงตามปุ่มกด (ต่อ)

บรรทัด	คำสั่ง
103	beep(NOTE_F4, 200);
104	beep(NOTE_G4, 300);
105	beep(NOTE_G4, 400);
106	beep(NOTE_F4, 200);
107	beep(NOTE_E4, 200);
108	beep(NOTE_F4, 200);
109	beep(NOTE_G4, 200);
110	beep(NOTE_E4, 400);
111	beep(NOTE_C4, 200);
112	beep(NOTE_C4, 200);
113	beep(NOTE_C4, 200);
114	beep(NOTE_D4, 200);
115	beep(NOTE_E4, 400);
116	beep(NOTE_E4, 200);
117	beep(NOTE_E4, 200);
118	beep(NOTE_D4, 200);
119	beep(NOTE_C4, 200);
120	beep(NOTE_D4, 200);
121	beep(NOTE_E4, 400);
123	beep(NOTE_C4, 200);
124	beep(NOTE_E4, 200);
125	beep(NOTE_E4, 200);
126	beep(NOTE_E4, 200);
127	beep(NOTE_F4, 400);
128	beep(NOTE_G4, 400);
129	beep(NOTE_G4, 200);
130	beep(NOTE_F4, 200);
131	beep(NOTE_E4, 200);
132	beep(NOTE_F4, 200);
133	beep(NOTE_G4, 200);
134	beep(NOTE_E4, 400);
135	}
136	void setup(){
137	irRecv.enableIRIn();

ตัวอย่างโปรแกรม 9-2 เล่นเพลงตามปุ่มกด (ต่อ)

บรรทัด	คำสั่ง
138	}
139	void loop(){
140	if(irRecv.decode(&results)){
141	if(millis() - last > 250){
142	keyCode = results.value;
143	for(i = 0; i < 10; i++){
144	if(keyCode == numberCodes[i]){
145	channel = i;
146	break;
147	}
148	}
149	switch (channel){
150	case 0 : nooMalee(); break;
151	case 1 : chang(); break;
152	case 2 : thatSpider(); break;
153	default : break;
154	}
155	}
156	last = millis();
157	irRecv.resume();
158	}
159	delay(50);
160	}

หมายเหตุ

- เพลง หนูมาลี มาจาก <https://www.arduitronics.com/article/20/arduino-sing-a-song>
- เพลง ช้าง มาจาก <https://www.tinkercad.com/things/2H2v0KUORol> โดย นางสาวจินตนา อาษา
- เพลง แมงมุมลายตัวนั้น โน้ตมาจาก <https://writer.dekd.com/33325/writer/viewlongc.php?id=898225&chapter=7>

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-IR_REMOTE เป็นภาครับสัญญาณอินฟราเรดจากรีโมทคอนโทรลรุ่น ET-IR REMOTE KEY และด้วยการใช้ไลบรารี IRremoteESP8266 ทำให้การเขียนโปรแกรมเพื่อรับข้อมูลรหัสและการถอดรหัสเป็นเรื่องง่ายด้วยการใช้คำสั่ง enableIRin() สำหรับเริ่มทำงาน ทำการถอดรหัสด้วยคำสั่ง decode() และเริ่มทำงานใหม่ด้วยคำสั่ง resume() พร้อมตัวอย่างจากบทนี้จะทำให้เห็นว่าสามารถนำไปปรับปรุงเพื่อสั่งงานให้อุปกรณ์ทำงานอื่น ๆ เช่น เปิด/ปิดน้ำ เปิด/ปิดไฟ ควบคุมทิศทางการเคลื่อนที่ของหุ่นยนต์ เป็นต้น

บรรณานุกรม

VISHAY. (2022). *TSOP4838*. สืบค้นเมื่อ เมษายน 10, 2567, จาก

<https://www.vishay.com/docs/82459/tsop48.pdf>

David Conran, Sebastien Warin, Mark Szabo และ Ken Shirriff. (2024). *IRremoteESP8266*.

สืบค้นเมื่อ เมษายน 10, 2567, จาก

<https://www.arduino.cc/reference/en/libraries/irremotesp8266/>



บทที่ 10 ADC/DAC

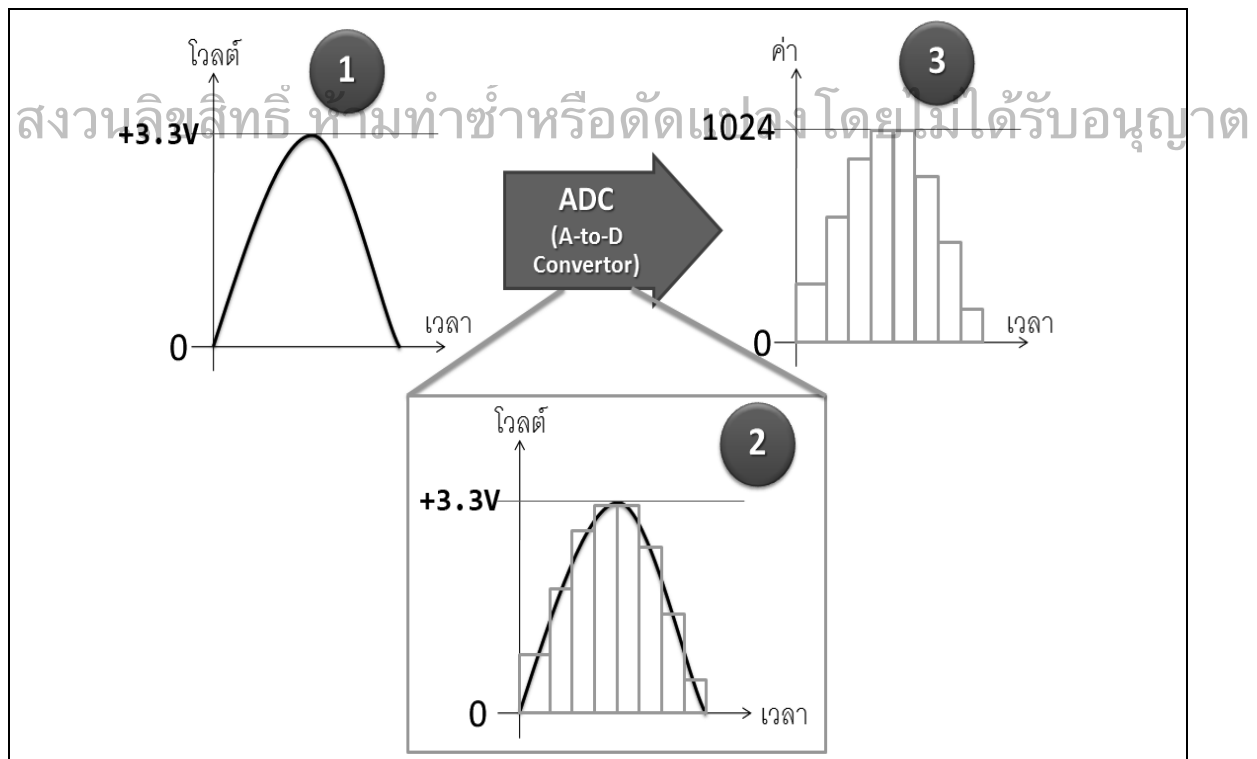
บทนี้กล่าวถึงโมดูลแปลงสัญญาณของไมโครคอนโทรลเลอร์ ESP32 ได้แก่ การแปลงสัญญาณแอนะล็อกเป็นดิจิทัลหรือ ADC และการแปลงสัญญาณดิจิทัลเป็นแอนะล็อกหรือ DAC (Digital-to-Analog convertor) โดยอธิบายถึงหลักการทำงานของทั้ง 2 แบบ พร้อมอธิบายชุดคำสั่งสำหรับใช้งานและตัวอย่างโปรแกรมสำหรับนำไปใช้กับอุปกรณ์ทดลองในบทที่ 11 ถึง 13 ต่อไป

> แนะนำ ADC

ADC เป็นโมดูลนำเข้าสู่สัญญาณแอนะล็อกเพื่อแปลงค่าสัญญาณเป็นค่าตัวเลขจำนวนเต็ม โดยขั้นตอนของการนำเข้าสู่สัญญาณแอนะล็อกเป็นดังภาพที่ 10-1 คือ (1) เมื่อมีสัญญาณแบบแอนะล็อกส่งเข้ามาส่วนตัวแปลงที่เรียกว่าเอดีซี (2) ตัวแปลงจะเปรียบเทียบแรงดันเป็นค่าตัวเลขจำนวนเต็มแบบดิจิทัล และ (3) ตัวแปลงสัญญาณจะส่งข้อมูลที่แปลงเป็นค่าตัวเลขดิจิทัลกลับออกมา

โมดูล ADC ของ ESP32 รองรับแรงดันกระแสตรงในช่วง 0 ถึง 1.1V ทำให้มีปัญหาในการนำไปใช้งานเนื่องจากส่วนอื่นของ ESP32 ทำงานกับแรงดัน 3.3V ด้วยเหตุนี้ ผู้ผลิตจึงได้เพิ่มวงจรภาคลดทอนแรงดันจาก 3.3V ให้เหลือในช่วง 1.1V มากับบอร์ดจึงไม่ต้องกังวลเรื่องของการภาคลดทอนเพื่อใช้งาน แต่อย่างไรก็ดี ผู้อ่านต้องระมัดระวังไม่ให้มีการจ่ายแรงดันที่เกิน 3.3V เข้ากับขาของ ESP32

การนำเข้าสู่สัญญาณแอนะล็อกเป็นการทำงานของตัวแปลงสัญญาณแอนะล็อกเป็นดิจิทัล โดยคุณสมบัติของบอร์ด ESP32 คือ สามารถแปลงสัญญาณแอนะล็อกได้ด้วยความสะดวก 12 บิต หรือ 2^{12} ทำให้สามารถแปลงแรงดันแอนะล็อก 3.3V ได้ 4096 ระดับ คือ ได้ค่าน้อยสุด หรือ 0V ที่ค่าดิจิทัล 0 และค่ามากที่สุด 3.3V ที่ค่าดิจิทัล 4095



ภาพที่ 10-1

ขั้นตอนการแปลงสัญญาณแอนะล็อกเป็นดิจิทัล

ภาพโดย จารุต บุศราทิจ

ขาของ ESP32 ที่ทำหน้าที่ ADC มีด้วยกัน 2 ช่องสัญญาณ คือ ADC1 และ ADC2 โดย ADC1 มีด้วยกัน 8 ขา และ ADC2 มี 10 ขาดังนี้

- ADC1_CH0 คือขา P36 หรือ ขา VP
- ADC1_CH1 คือขา P37
- ADC1_CH2 คือขา P38
- ADC1_CH3 คือขา P39 หรือ ขา VN
- ADC1_CH4 คือขา P32
- ADC1_CH5 คือขา P33
- ADC1_CH6 คือขา P34
- ADC1_CH7 คือขา P35
- ADC2_CH0 คือขา P4
- ADC2_CH1 คือขา P0
- ADC2_CH2 คือขา P2
- ADC2_CH3 คือขา P15
- ADC2_CH4 คือขา P13
- ADC2_CH5 คือขา P12

- ADC2_CH6 คือขา P14
- ADC2_CH7 คือขา P27
- ADC2_CH8 คือขา P25
- ADC2_CH9 คือขา P26

การใช้งานขา P0, P2 และ P15 ของ ADC2 ต้องระมัดระวังในการใช้เนื่องจากถูกใช้งานในหน้าที่พิเศษของ ESP32

คำสั่ง `analogReadResolution()` เป็นคำสั่งสำหรับปรับความละเอียดการแปลงค่าของโมดูล ADC ของ ESP32 มีรูปแบบการใช้งานดังนี้ โดย ความละเอียดของสัญญาณ คือตัวเลขจำนวนบิตในการทำงานโมดูลแปลงสัญญาณมีค่าไม่เกิน 12 และถ้าค่าที่กำหนดไม่ใช่ตัวเลขในช่วง 9 ถึง 12 ตัวระบบจะทำการแปลงค่าให้จำนวนบิตน้อยลงด้วยการเลื่อนบิต นั้นหมายความว่า ความเร็วสูงสุดในการแปลงจึงเป็นแบบ 9 บิต และช้าที่สุดอยู่ที่แบบ 12 บิต แต่ถ้ากำหนดต่ำกว่า 9 บิต จะเป็นการเลื่อนบิตจากการทำงานแบบ 9 บิต เพื่อให้ได้ช่วงค่าที่ลดลง หรือใช้คำสั่ง `analogSetWidth()` สำหรับกำหนดความละเอียดในการแปลงด้วยการกำหนดค่า ความละเอียดของสัญญาณ เป็นตัวเลขในช่วง 9 ถึง 12 ตามรูปแบบของการใช้งานดังนี้

`analogReadResolution( ความละเอียดของสัญญาณ )`

`analogSetWidth( ความละเอียดของสัญญาณ )`

รูปแบบคำสั่งสำหรับอ่านค่า ADC ของ ESP32 ใช้คำสั่ง `analogRead( )` พร้อมระบุหมายเลขขาสำหรับอ่านค่า โดย ESP32 จะสามารถใช้ขา P32, P33, P34, P35, P36 และ P39 เป็นขานำเข้าข้อมูลสัญญาณแอนะล็อก โดยขา P34, P35, P36 และ P39 ทำหน้าที่ได้เพียงนำเข้าสัญญาณได้เพียงอย่างเดียวเท่านั้น แต่ P32 และ P33 สามารถทำงานได้ทั้งเป็นขานำเข้าสัญญาณแอนะล็อก/ดิจิตอล และส่งออกสัญญาณดิจิตอล

ค่าที่อ่านได้ = `analogRead ( หมายเลขขา )`

> แนะนำ DAC

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

DAC เป็นโมดูลส่งออกสัญญาณแอนะล็อกเพื่อให้ขา P25 และ P26 สามารถส่งแรงดันตั้งแต่ 0 ถึง 3.3V ออกแทนค่าดิจิทัลที่เป็นแรงดันแทนค่า 0 และ 1 ทำให้สามารถควบคุมความละเอียดของแรงดันเพื่อส่งให้กับวงจรที่ต่อพ่วง โดยโมดูลมีความละเอียด 8 บิต ทำให้ส่งค่า 0-255 แทนค่าแรงดันในช่วง 0 ถึง 3.3V

คำสั่งปรับความละเอียดของ DAC ของ ESP32 ใช้คำสั่งตามรูปแบบต่อไปนี้ โดยกำหนด ความละเอียดของสัญญาณ เป็นค่าจำนวนเต็มของค่าบิตที่ต้องการใช้งาน ปกติมีค่าเป็น 8

`analogWriteResolution( ความละเอียดของสัญญาณ )`

สำหรับคำสั่งส่งออกข้อมูลขนาด 10 บิตของ ESP32 นั้นสามารถใช้คำสั่งตามรูปแบบต่อไปนี้โดยกำหนดขาเป็น P25 หรือ P26 ดังนี้

`dacWrite ( หมายเลขขา, ค่าส่งออก )`

ตัวอย่างกราฟฟืนปลา

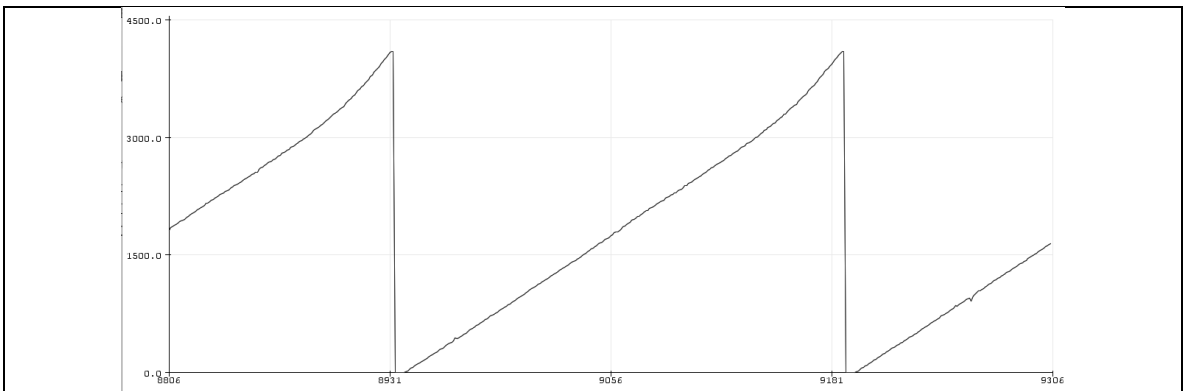
การสร้างกราฟฟืนปลาตามตัวอย่างโปรแกรม 10-1 เป็นการส่งค่า 0 ถึงค่าสูงสุดที่เป็นไปได้ของภาค DAC หลังจากนั้นวนรอบส่งใหม่อีกครั้ง ทำให้ได้กราฟมีลักษณะเป็นฟืนปลาดังภาพที่ 10-2

ตัวอย่างโปรแกรม 10-1 การใช้ DAC สร้างคลื่นแบบฟืนปลา

บรรทัด	คำสั่ง
1	<code>#define pinSpk 26</code>
2	<code>#define pinMic 36</code>
3	<code>int adcValue;</code>
4	<code>void setup() {</code>
5	<code> pinMode(pinSpk, OUTPUT);</code>
6	<code> pinMode(pinMic, INPUT);</code>
7	<code> Serial.begin(115200);</code>
8	<code> analogReadResolution(12);</code>

ตัวอย่างโปรแกรม 10-1 การใช้ DAC สร้างคลื่นแบบฟันปลา (ต่อ)

บรรทัด	คำสั่ง
9	}
10	void loop() {
11	for (int i = 0; i < 255; i++) {
12	dacWrite(pinSpk, i);
13	adcValue = analogRead(pinMic);
14	Serial.println(adcValue);
15	}
16	}



ภาพที่ 10-2

ตัวอย่างกราฟฟันปลา

ภาพโดย จารุต บุศราทิจ

ตัวอย่างกราฟสามเหลี่ยม

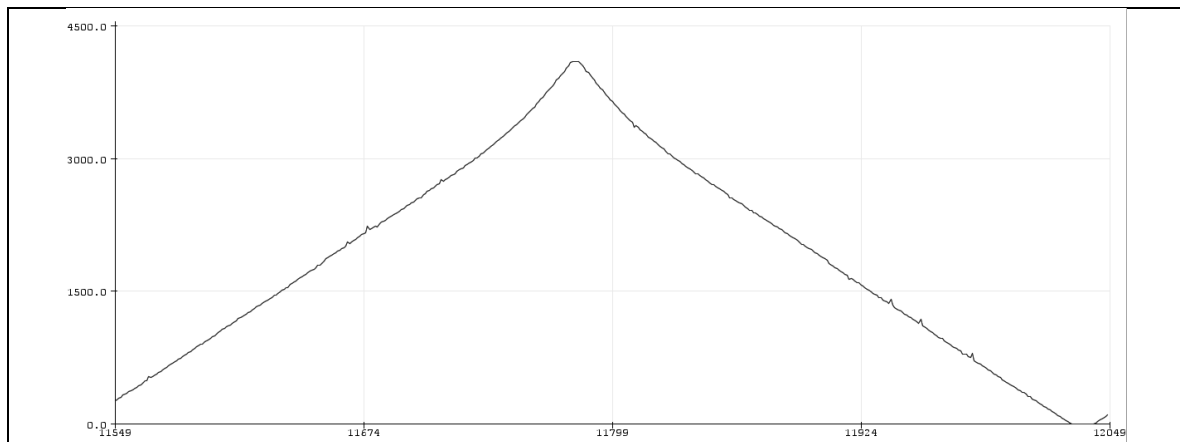
ตัวอย่างโปรแกรม 10-2 กราฟสามเหลี่ยมมีหลักการคล้ายกับฟันปลาด้วยการเพิ่มการวนรอบเพื่อส่งค่ามากที่สุดกลับมาเป็น 0 ทำให้กราฟออกมาดูเป็นสามเหลี่ยมดังตัวอย่างในภาพที่ 10-3

ตัวอย่างโปรแกรม 10-2 การใช้ DAC สร้างกราฟคลื่นสามเหลี่ยม

บรรทัด	คำสั่ง
1	#define pinSpk 26
2	#define pinMic 36
3	int adcValue;
4	void setup() {
5	pinMode(pinSpk, OUTPUT);

ตัวอย่างโปรแกรม 10-2 การใช้ DAC สร้างกราฟคลื่นสามเหลี่ยม (ต่อ)

บรรทัด	คำสั่ง
6	pinMode(pinMic, INPUT);
7	Serial.begin(115200);
8	analogReadResolution(12);
9	}
10	void loop() {
11	for (int i = 0; i < 255; i++) {
12	dacWrite(pinSpk, i);
13	adcValue = analogRead(pinMic);
14	Serial.println(adcValue);
15	}
16	for (int i = 254; i >= 0; i--) {
17	dacWrite(pinSpk, i);
18	adcValue = analogRead(pinMic);
19	Serial.println(adcValue);
20	}
21	}



ภาพที่ 10-3

ตัวอย่างกราฟสามเหลี่ยม

ภาพโดย จารุต บุศราทิจ

ตัวอย่างกราฟคลื่นไซน์

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

การสร้างกราฟคลื่นรูปไซน์ตามตัวอย่างโปรแกรม 10-3 ใช้การกำหนดให้มีการวนรอบเพื่อเพิ่มค่ามุมมองจาก 0 ไปถึง 359 โดยในแต่ละรอบทำสิ่งต่อไปนี้

1. แปลงค่าองศาเป็นเรเดียน
2. คำนวณหาไซน์จากมุมเรเดียน
3. แปลงค่าทศนิยมให้เป็นจำนวนเต็มในช่วงค่าที่ DAC รองรับ
4. อ่านค่าจาก ADC ที่กำหนดให้ทำงานที่ความละเอียด 8 บิต
5. ส่งข้อมูลออกทางพอร์ตอนุกรมให้โปรแกรม Serial Plotter ได้ผลลัพธ์ดังภาพที่ 10-4

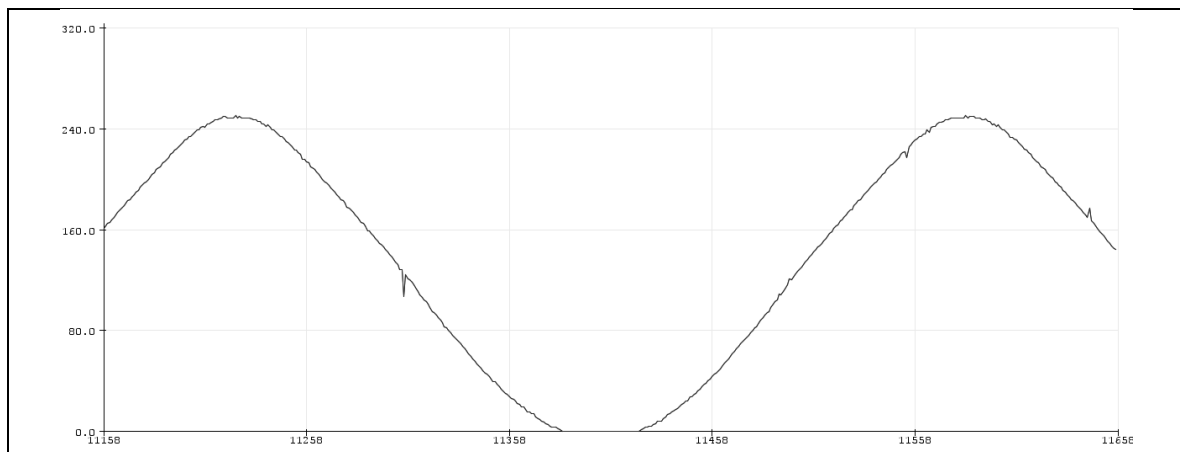
กราฟจาก ADC ของ ESP32 นั้นออกมาไม่เป็นเส้นตรง ทั้งนี้เนื่องจาก ภาควิทยา ADC ของ ESP32 มีความไวต่อสัญญาณรบกวนค่อนข้างมาก ดังนั้น ต้องหาวิธีการชดเชยค่าที่ผิดพลาด และการป้องกันสัญญาณรบกวน

ตัวอย่างโปรแกรม 10-3 ส่ง DAC สร้างกราฟคลื่นไซน์

บรรทัด	คำสั่ง
1	#define pinSpk 26
2	#define pinMic 36
3	int adcValue;
4	void setup() {
5	pinMode(pinSpk, OUTPUT);
6	pinMode(pinMic, INPUT);
7	Serial.begin(115200);
8	analogReadResolution(8);
9	}
10	void loop() {
11	int degree = 0;
12	float radian = 0.0;
13	float sineValue = 0.0;
14	int dValue = 0;
15	for (degree = 0; degree < 360; degree++) {
16	radian=(float)degree*(2.0*3.1415926)/360.0;
17	sineValue = sin(radian);
18	dValue = (int)((1.0 + sineValue) * 125.0);
19	dacWrite(pinSpk, dValue);

ตัวอย่างโปรแกรม 10-3 สร้าง DAC สร้างกราฟคลื่นไซน์ (ต่อ)

บรรทัด	คำสั่ง
20	adcValue = analogRead(pinMic);
21	Serial.println(adcValue);
22	}
23	}



ภาพที่ 10-4

ตัวอย่างกราฟคลื่นไซน์

ภาพโดย จารุต บุศราทิจ

> สรุป

การแปลงสัญญาณระหว่างสัญญาณดิจิทัลและแอนะล็อกนั้นสามารถทำได้ด้วยไมโครคอนโทรลเลอร์ ESP32 ทำให้สามารถลดค่าใช้จ่ายของการเพิ่มวงจรการใช้งาน โดยภาค ADC นั้นรองรับความละเอียดในการแปลงสัญญาณ 12 บิต และสามารถตั้งค่าได้ต่ำสุดที่ 9 บิตเพื่อให้งานของการแปลงสัญญาณดำเนินการรวดเร็วขึ้น ขณะที่ภาค DAC สามารถแปลงสัญญาณที่มีความละเอียด 8 บิตทำให้สามารถควบคุมระดับแรงดันส่งออกได้ 256 ระดับ

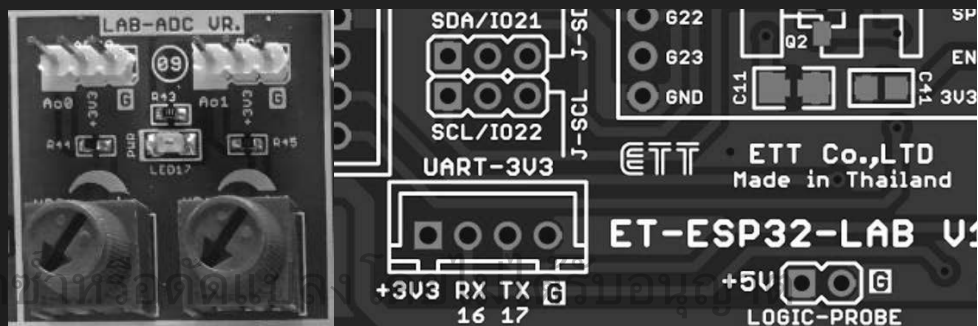
แต่อย่างไรก็ดี ถ้าสังเกตที่ตัวอย่างในภาพที่ 10-2, 10-3 และ 10-4 จะพบว่า ภาคแปลง ADC นั้นมีความผิดพลาดบ้างในบางครั้ง ดังนั้น ถ้าผู้อ่านต้องทำงานกับการแปลงสัญญาณที่มีความละเอียดสูง ความเร็วสูง และมีความแม่นยำสูง ควรเลือกใช้ไอซีเฉพาะทางที่เหมาะสมกับงาน

บรรณานุกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

Espressif System. (2024). *ESP32 Series Datasheet*. สืบค้นเมื่อ เมษายน 11, 2567, จาก

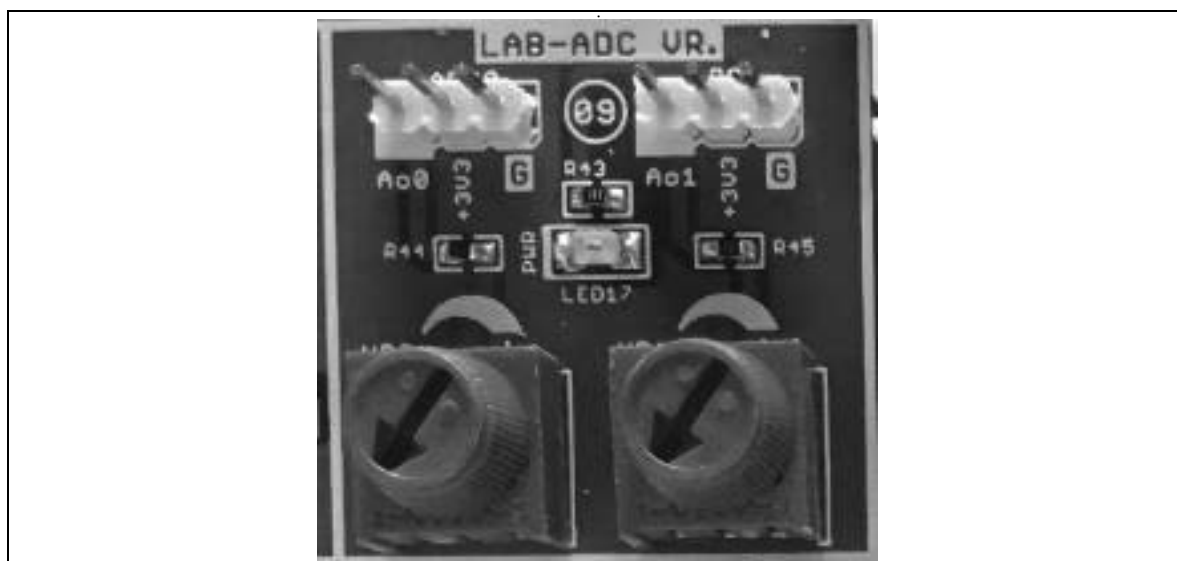
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_e



บทที่ 11 LAB-ADC VR

ในบทที่ 8 ได้เรียนรู้การนำเข้าสู่สัญญาณดิจิทัลจากอุปกรณ์ประเภทสวิตช์เพื่อนำค่า 0 หรือ 1 (LOW หรือ HIGH) ไปตัดสินใจทำงานต่อไป แต่อย่างไรก็ดี สัญญาณแบบแอนะล็อกเป็นสัญญาณประเภทหนึ่งที่พบบ่อย และค่าของสัญญาณไม่ได้เป็นเพียง 1 หรือ 0 ทำให้สามารถเขียนโปรแกรมเพื่อประกอบการตัดสินใจ หรือแปลงช่วงค่าได้หลายระดับมากกว่าแบบดิจิทัล โดยการทำงานต้องอาศัยภาคแปลงสัญญาณแอนะล็อกเป็นดิจิทัลที่ได้กล่าวถึงในบทที่ 10 และในบทนี้เป็นวงจรของ LAB-ADC VR อันเป็นการนำเข้าสู่สัญญาณจากตัวต้านทานแบบปรับค่าได้หรือวอร์ (VR: Variable Resistor)

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรการทำงานของ LAB-ADC VR ดังภาพที่ 11-1 ไปประยุกต์ใช้งานเพื่อรองรับการกดสวิตช์เพื่อส่งงานจากผู้ใช้ เช่น ทำตัววัดระดับแรงดันไฟฟ้ากระแสตรงเพื่อตัดสินใจว่าอยู่ในช่วงค่า HIGH, LOW หรือ UNKNOWN เป็นต้น



ภาพที่ 11-1

ภาพโมดูล LAB-ADC VR

ภาพโดย จารุต บุศราทิจ

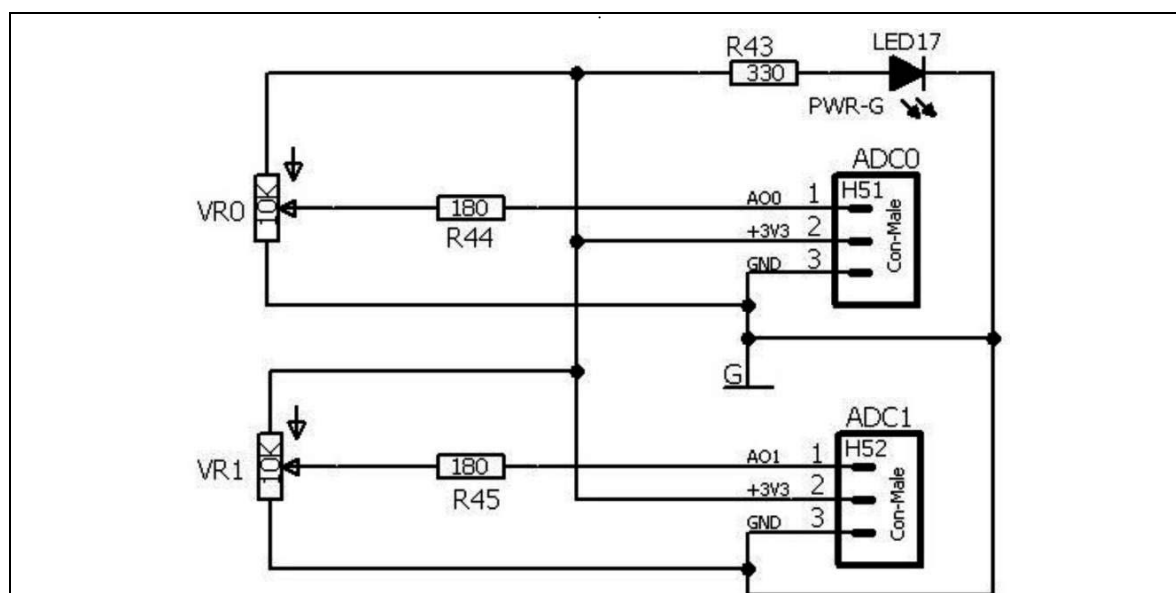
> แนะนำ LAB-ADC VR

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-ADC VR จากภาพที่ 11-1 เป็นโมดูลสำหรับทดลองการนำเข้าข้อมูลแบบแอนะล็อกด้วยการหมุนเพื่อปรับค่าความต้านทาน VR0 และ VR1 โดยมี LED หลอดสีเขียวสำหรับแสดงสถานะทำงานของโมดูล และมีขาสำหรับเชื่อมต่อกับ ESP32 จำนวน 2 ชุดสำหรับ VR0 และ VR1 แต่ละชุดประกอบด้วยขาสัญญาณ 3 ขา ดังนี้

สำหรับ VR0 ประกอบด้วย Ao0 สำหรับส่งแรงดันไฟฟ้ากระแสตรงจาก VR0 ขา +3V3 สำหรับจ่ายแรงดัน 3V3 จากบอร์ด ESP32 และ G สำหรับต่อกับกราวด์

และสำหรับ VR1 ประกอบด้วย Ao1 สำหรับส่งแรงดันไฟฟ้ากระแสตรงจาก VR0 ขา +3V3 สำหรับจ่ายแรงดัน 3V3 จากบอร์ด ESP32 และ G สำหรับต่อกับกราวด์



ภาพที่ 11-2 วงจรของ LAB-AD VR

ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

วงจรของภาค LAB-ADC VR เป็นดังภาพที่ 11-2 โดยตัวต้านทานปรับค่าได้ขนาด 10 โอห์ม (ตัวปรับแบบหมุน) มีจำนวน 2 ตัว เรียกว่า VR0 และ VR1 ที่นำเข้าแรงดัน 3V3 จากขา +3V3 หลังจากนั้นส่งแรงดันกลับไปยังขา Ao0 และ Ao1 พร้อมมีการลดทอนแรงดันด้วยตัวต้านทาน 180 โอห์ม (R44 และ

R45) เพื่อให้โมดูล ADC ของ ESP32 นำไปใช้เป็นแรงดันนำเข้าเพื่อแปลงผลลัพธ์ออกมาเป็นเลขจำนวนเต็ม

การใช้งานต้องเชื่อมต่อกับขา ESP32 ที่รองรับการทำงานของโมดูลแปลงสัญญาณแอนะล็อกเป็นดิจิทัล โดยความละเอียดของค่าที่แปลงขึ้นอยู่กับค่าจำนวนบิตในการแปลงข้อมูล เช่น กำหนดให้ทำงาน 10 บิต จะได้ค่าผลลัพธ์ 1024 ค่า หรือ 0 ถึง 1023 และถ้ากำหนดเป็น 12 บิตจะทำให้ผลลัพธ์ที่กลับออกมามีจำนวน 4096 ค่า หรือเป็นค่าตัวเลขจำนวนเต็มในช่วง 0 ถึง 4095 เป็นต้น

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 11-1 อ่านค่าจาก VR0 และ VR1 แบบความละเอียด 12 บิต เพื่อแสดงค่าที่แปลงได้ และคำนวณกลับเป็นระดับแรงดัน โดยรับค่าของ ADC1 และ ADC2 ในบรรทัดที่ 25 และ 26 จากนั้นแปลงเป็นค่าแรงดันในบรรทัดที่ 28 และ 29 และแสดงผล โดยต่อขา 21 และ 22 เข้ากับ SDA และ SCL ของโมดูล OLED และต่อขา 34 และ 35 เข้ากับโมดูล ADC

ตัวอย่างโปรแกรม 11-1 อ่านค่าจาก VR เพื่อแสดง OLED

บรรทัด	คำสั่ง
1	#include <Adafruit_GFX.h>
2	#include <Adafruit_SH1106.h>
3	#define SDA 21
4	#define SCL 22
5	#define ADC0 34
6	#define ADC1 35
7	Adafruit_SH1106 display(SDA, SCL);
8	float inputVoltage = 3.3;
9	int ADCIn0,ADCIn1, i;
10	float VA0,VA1;
11	void setup(){
12	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
13	display.clearDisplay();
14	display.display();
15	delay(100);

ตัวอย่างโปรแกรม 11-1 อ่านค่าจาก VR เพื่อแสดง OLED (ต่อ)

บรรทัด	คำสั่ง
16	display.setTextSize(1);
17	display.setTextColor(WHITE);
18	display.setCursor(0,0);
19	pinMode(ADC0,INPUT);
20	pinMode(ADC1,INPUT);
21	}
22	void loop(){
23	display.clearDisplay();
24	display.setCursor(0,0);
25	ADCIn0 = analogRead(ADC0);
26	ADCIn1 = analogRead(ADC1);
27	delay(500);
28	VA0 = float((ADCIn0 * inputVoltage) / 4095);
29	VA1 = float((ADCIn1 * inputVoltage) / 4095);
30	display.print("ADC0 Value = ");
31	display.println(ADCIn0);
32	display.print("Voltage of A0 = ");
33	display.print(VA0, 1);
34	display.println(" V");
35	for(i = 0; i < ADCIn0 / 195; i++){
36	display.print("/");
37	}
38	display.println("");
39	display.print("ADC1 Value = ");
40	display.println(ADCIn1);
41	display.print("Voltage of A1 = ");
42	display.print(VA1,1);
43	display.println(" V");
44	for(i = 0; i < ADCIn1 / 195; i++){
45	display.print("/");
46	}
47	display.display();
48	delay(1000);
49	}

ตัวอย่างโปรแกรม 11-2 อ่านค่าจาก VR0 และ VR1 แบบความละเอียด 12 บิต เพื่อแสดงนำมาประมวลผลว่า ระดับแสงดังกล่าวเมื่อนำมาใช้เปรียบเทียบกับเป็นสัญญาณดิจิทัลแบบ TTL คือ HIGH อยู่แรงดันในช่วง 0.0 ถึง 0.8 โวลต์, LOW เป็นแรงดันในช่วง 2.0 ถึง 3.3 โวลต์ และค่าอื่นเป็นค่าที่ระบุไม่ได้หรือ Unknown โดยตรวจสอบแรงดันของโมดูล ADC ทั้งสองตัว หากแรงดันต่ำกว่า 0.8V จะแสดงค่า LOW หากแรงดันสูงกว่า 2.0V จะแสดงค่า HIGH นอกนั้นจะแสดง Unknown ในบรรทัดที่ 33 ถึง 47

ตัวอย่างโปรแกรม 11-2 ตรวจสอบแรงดันแบบดิจิทัล

บรรทัด	คำสั่ง
1	#include <Adafruit_GFX.h>
2	#include <Adafruit_SH1106.h>
3	#define SDA 21
4	#define SCL 22
5	#define ADC0 34
6	#define ADC1 35
7	Adafruit_SH1106 display(SDA, SCL);
8	float inputVoltage = 3.3;
9	float lowBound = 0.8;
10	float highBound = 2.0;
11	int ADCIn0,ADCIn1;
12	float VA0,VA1;
13	void setup(){
14	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
15	display.clearDisplay();
16	display.display();
17	delay(100);
18	display.setTextSize(1);
19	display.setTextColor(WHITE);
20	display.setCursor(0,0);
21	pinMode(ADC0,INPUT);
22	pinMode(ADC1,INPUT);
23	}
24	void loop(){
25	display.clearDisplay();
26	display.setCursor(0,0);
27	ADCIn0 = analogRead(ADC0);

ตัวอย่างโปรแกรม 11-2 ตรวจสอบแรงดันแบบดิจิทัล (ต่อ)

บรรทัด	คำสั่ง
28	ADCIn1 = analogRead(ADC1);
29	delay(500);
30	VA0 = float((ADCIn0 * inputVoltage) / 4095);
31	VA1 = float((ADCIn1 * inputVoltage) / 4095);
32	display.print("TTL of A0 = ");
33	if(VA0 < lowBound){
34	display.println("LOW");
35	}else if(VA0 > highBound){
36	display.println("HIGH");
37	}else{
38	display.println("Unknow");
39	}
40	display.print("TTL of A1 = ");
41	if(VA1 < lowBound){
42	display.println("LOW");
43	}else if(VA1 > highBound){
44	display.println("HIGH");
45	}else{
46	display.println("Unknow");
47	}
48	display.display();
49	delay(1000);
50	}

> สรุป

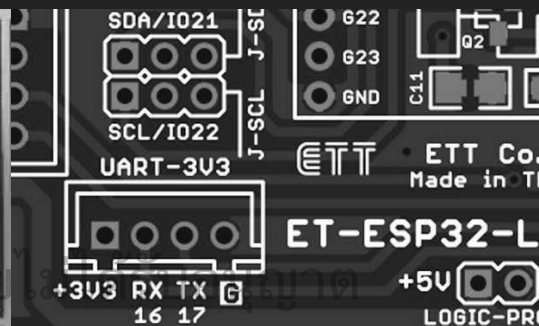
ภาค LAB-ADC VR เป็นการใช้งานตัวต้านทานปรับค่าได้ ทำให้แรงดันที่เล่นผ่านอุปกรณ์ประเภทนี้มีค่าเปลี่ยนแปลงไปตามค่าความต้านทาน และเมื่อนำข้อมูลแรงดันมาใช้งานกับ ESP32 จะต้องทำการแปลงค่าจากแรงดันที่เป็นสัญญาณแบบแอนะล็อกให้เป็นสัญญาณดิจิทัลที่เป็นการแทนค่าแรงดันด้วย

ตัวเลขจำนวนเต็มค่าหนึ่งด้วยการใช้คำสั่ง `analogRead()` หลังจากนั้นนำค่าตัวเลขที่ได้ไปตรวจสอบเงื่อนไขที่กำหนดขึ้นเพื่อให้โปรแกรมทำงานตามข้อกำหนดที่ต้องการต่อไป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

บรรณานุกรม

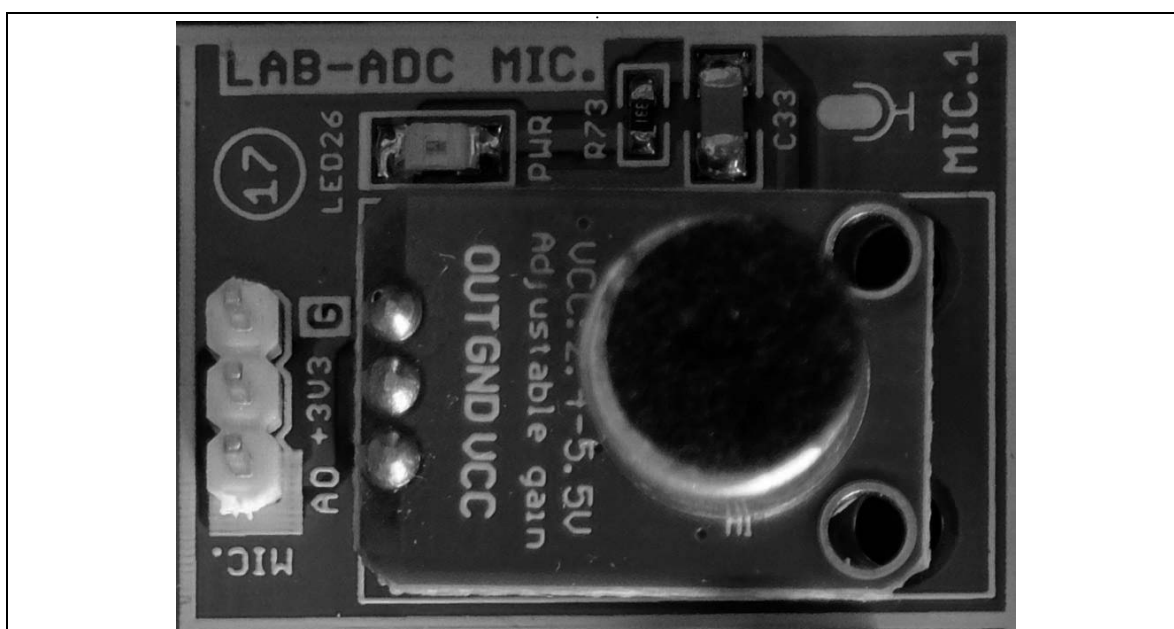
ETT. (2024). *MANUAL - คู่มือ ET-ESP32-LAB V1.0*. สืบค้นเมื่อ เมษายน 18, 2567, จาก <https://www.etteam.com/prodKitNLab/ET-ESP32-LABV1/th-man-ET-ESP32-LABV1.pdf>



บทที่ 12 LAB-ADC MIC

ในบทที่ 11 เป็นตัวอย่างการใช้งานโมดูลแปลงสัญญาณแอนะล็อกเป็นดิจิทัลหรือ ADC ที่ได้กล่าวถึงชุดคำสั่งในบทที่ 10 ซึ่งเป็นการปรับเปลี่ยนแรงดันตามการหมุนที่ตัวต้านทานแบบปรับค่าได้ สำหรับบทนี้เป็นการใช้งาน LAB-ADC MIC ที่เป็นสัญญาณแอนะล็อกที่มาจากโมดูลรับเสียงหรือไมค์ (MIC) ดังภาพที่ 12-1 เพื่อเรียนรู้เพิ่มเติมจากตัวอย่างบทที่ 10 คือ นำเข้าสัญญาณเพื่อผ่านการแปลงฟูริเย (Fourier Transform) อย่างเร็ว (FFT: Fast Fourier Transform) สำหรับแปลงสัญญาณเสียงเป็นความถี่ อันเป็นพื้นฐานเบื้องต้นสำหรับการประมวลผลเสียง

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรและหลักการทำงานของโมดูล LAB-ADC MIC และการใช้งานไลบรารี FFT ไปประยุกต์ใช้งานเพื่อรองรับการประมวลผลเสียง หรือตรวจจับเสียงเพื่อส่งงานต่อไป



ภาพที่ 12-1

ภาพโมดูล LAB-ADC MIC

ภาพโดย จารุต บุศราทิจ

> แนะนำ LAB-ADC MIC

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ภาค LAB-ADC MIC จากภาพที่ 12-1 เป็นโมดูลสำหรับทดลองการนำเข้าสู่ข้อมูลแบบแอนะล็อก ในรูปแบบของเสียงความถี่สูงจากโมดูลเซ็นเซอร์ไมโครโฟนที่มีหน่วยขยายเสียงขนาดเล็กรุ่น MAX4466 ของบริษัทแอนะล็อกดีไวส์ (Analog Device) ทำให้ไม่จำเป็นต้องเพิ่มโมดูลขยายเสียงที่นำเข้ามาจากไมโครโฟนเอง ทั้งนี้เนื่องจากข้อมูลเสียงที่ได้จากไมโครโฟนมีค่าแอมพลิจูดที่ต่ำทำให้มีปัญหาเมื่อนำเข้าไปที่ภาคแปลงสัญญาณแบบแอนะล็อกเป็นดิจิทัล เพราะเมื่ออ่านมาได้ค่าต่ำแล้วนำมาสเกลค่าจะทำให้สัญญาณขาดหายไป ดังนั้น โดยปกติเมื่อใช้การนำเข้ามาจากไมโครโฟนจึงต้องใช้วงจรขยายสัญญาณเข้ามาช่วย เช่น ออปแอมป์ LM386 หรือ LM358 ที่อันเป็นตัวขยายสัญญาณของบริษัท TI เป็นต้น

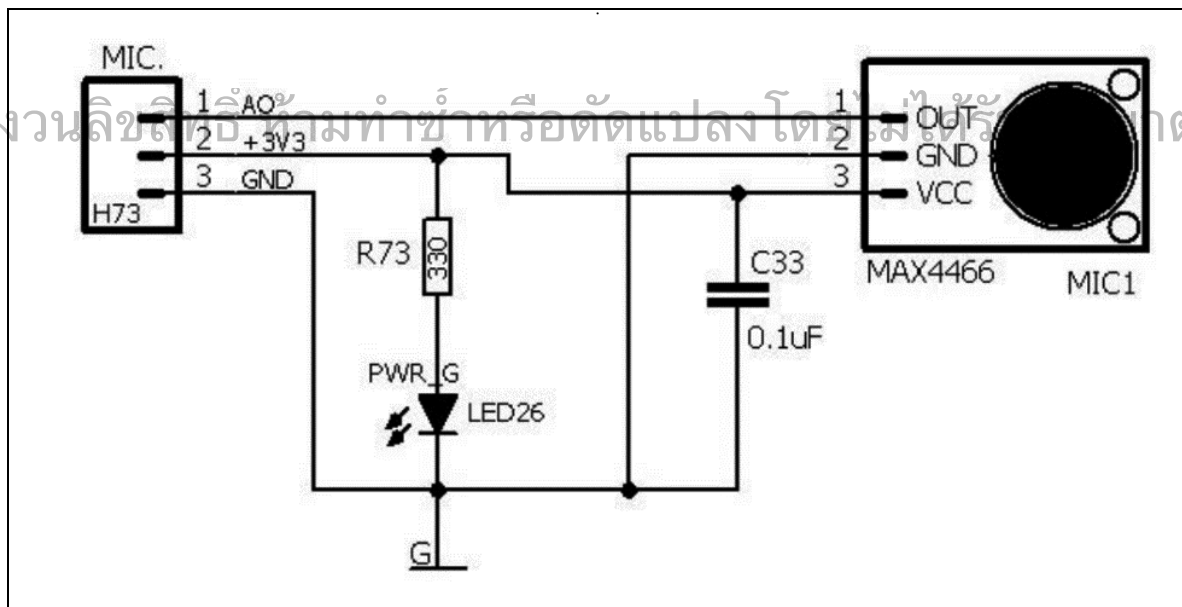
บนบอร์ดภาค LAB-ADC MIC มีขั้วต่อสำหรับใช้งาน 3 ขา คือ ขาต่อแรงดันไฟฟ้ากระแสตรง 3.3V, ขาเชื่อมต่อกราวด์ และขาสัญญาณแอนะล็อก A0

> วงจรและการทำงาน

วงจรของภาค LAB-ADC MIC เป็นดังภาพที่ 12-2 จะพบว่า ขา OUT จาก MAX4466 จะต่อเข้ากับขั้ว A0 สำหรับเป็นนำสัญญาณแอนะล็อกจากไอซีไปยังภาคแปลงสัญญาณแอนะล็อกเป็นดิจิทัล และบนบอร์ดมีหลอดแอลอีดี (LED26) สีเขียวสำหรับแสดงสถานะการทำงานของบอร์ด

ไอซี MAX4466 เป็นโมดูลไมโครโฟนขนาดเล็กที่รวมภาคขยายสัญญาณ และมีราคาถูกที่ผลิตโดยบริษัท Analog Device อันมีคุณสมบัติดังนี้

- ทำงานที่แรงดันกระแสตรงย่าน +2.4V ถึง +5.5V
- ความดังสูงสุด 125dB
- ใช้กระแส 24uA
- Gain Bandwidth 600kHz
- ทำงานในอุณหภูมิ -40°C ถึง 85°C
- โมดูลไมโครโฟนอิเล็กทรอนิกส์ 20 Hz ถึง 20 KHz
- สามารถตัดสัญญาณรบกวน



ภาพที่ 12-2

วงจรของ LAB-AD MIC

ภาพโดย จารุต บุศราทิจ

> ไลบรารี Filter.h

ในบทที่ 1 ได้กล่าวถึงวิธีการอ่านข้อมูลจากสัญญาณแอนะล็อกและทำการแปลงค่าให้เป็นตัวเลขแบบดิจิทัลด้วยคำสั่ง `analogRead()` แต่อย่างไรก็ดี ด้วยความเร็วในการทำงานของการแปลงค่าของ ESP32 อาจทำให้เกิดความผิดพลาดในการแปลงค่าในบางครั้ง จึงต้องแก้ปัญหาด้วยการอ่านค่าข้อมูลมาจำนวนหนึ่งแล้วนำค่าเฉลี่ยของก้อนข้อมูลนั้นมาใช้งาน หรือทำการหาค่าเฉลี่ยของข้อมูลก่อนหน้ากับปัจจุบันเพื่อไม่ให้เกิดค่าการกระเพื่อมของค่ามากเกินไป เป็นต้น

ไลบรารี Filter.h เป็นไลบรารีที่พัฒนาโดย Hideakitai ด้วยการเก็บข้อมูลเข้าโครงสร้างข้อมูลแบบคิว (Queue Data Structure) ที่มีหลักการทำงานแบบ FIFO (First-In-First-Out) หรือสิ่งที่เข้ามาก่อนจะถูกนำออกไปก่อน ด้วยการจัดเก็บแบบลิงค์ลิสต์เดี่ยว (Single Linked List) ทำให้สามารถจัดเก็บข้อมูลที่อ่านได้ตามหน่วยความจำที่มีอยู่ ทั้งนี้ ไลบรารีออกแบบมาเพื่อทำการหาค่าเฉลี่ยของข้อมูลนำเข้า ทำให้แก้ปัญหาเรื่องของสัญญาณที่อ่านมาแล้วมีการเปลี่ยนแปลงไว โดยการใช้งานจะต้องเรียกใช้ไฟล์ส่วนหัว Filter.h ดังนี้

```
#include <Filter.h>
```

ในการใช้งานจะเรียกผ่านทางคลาส Moving_average ที่ผู้ใช้อยู่ระบุ จำนวนข้อมูล ที่ต้องการนำมาใช้ในการหาค่าเฉลี่ย โดยการใช้งานต้องสร้างตัวแปรสำหรับใช้งานดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

Moving_average ชื่อตัวแปร(จำนวนข้อมูล)

การนำเข้า ข้อมูลแอนะล็อก และคืนค่ากลับเป็นค่าเฉลี่ยจากข้อมูลที่มีอยู่ต้องเรียกผ่านคำสั่ง filter() ตามรูปแบบของการใช้งานดังนี้

ค่าเฉลี่ย = ชื่อตัวแปร.filter(ข้อมูลแอนะล็อก)

> ไลบรารี FFT

ฌ็อง บาติสต์ โฌแซ็ฟ ฟูรีเย (Jean Baptiste Joseph Fourier , ค.ศ. 1768–1830) ชาวฝรั่งเศส ได้ค้นพบสมการของการไหลของความร้อน โดยทำการแก้ปัญหาและพิสูจน์สมการให้เห็นว่าฟังก์ชันแบบหลายตัวแปรสามารถขยายออกเป็นอนุกรมของไซน์ (sines) และแนวคิดนี้ได้ถูกพัฒนาต่อในหลายด้าน นั่นหมายความว่าคลื่นเสียงจะประกอบได้ด้วยคลื่นรูปไซน์ที่มีความถี่แตกต่างกันและด้วยเหตุนี้จึงสามารถแยกคลื่นเสียงออกมาเป็นคลื่นรูปไซน์ที่มีความถี่แตกต่างกันหรืออนุกรมฟูรีเยตามสมการที่ 12-1 ต่อไปนี้

$$f(x) = \frac{1}{2} a_0 + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)] \quad \text{สมการ 12-1}$$

โดยที่

a_n และ b_n เป็นค่าจำนวนจริงของขนาดของอนุกรมฟูรีเย

แต่เมื่อนำสมการ 12-1 มาใช้กับข้อมูลแบบไม่ต่อเนื่อง (discrete) เพื่อคำนวณด้วยเครื่องคอมพิวเตอร์เรียกว่าการแปลงฟูรีเยไม่ต่อเนื่อง (DFT: Discrete Fourier Transform) ที่เขียนแทนด้วย $x[n]$ ดังสมการที่ 12-2

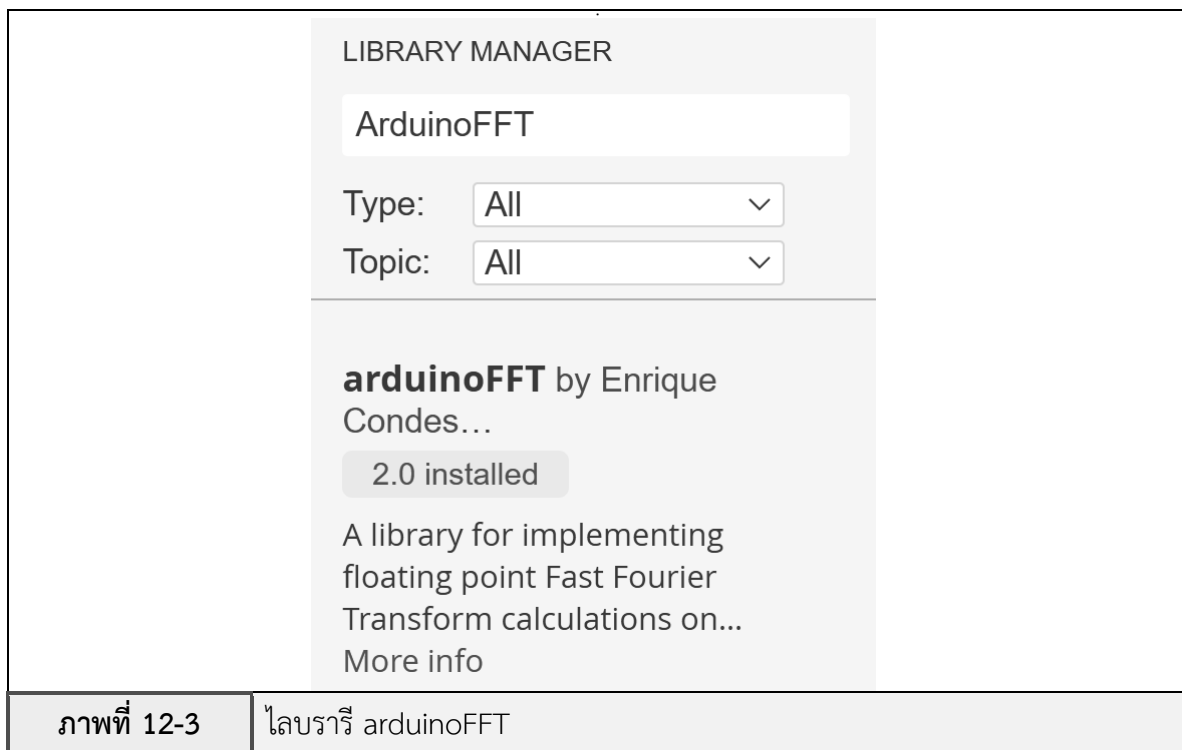
$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{2\pi i n k / N} \quad \text{สมการที่ 12-2}$$

โดยที่

$X[k]$ เป็นค่าขนาดบนโดเมนการแปลง
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

การคำนวณด้วยสมการ 12-2 จะได้ค่าความซับซ้อนของอัลกอริทึมเป็น $O(N^2)$ และมีผู้คิดค้นวิธีการลดความซับซ้อนให้เหลือ $O(N \log N)$ ด้วยวิธีการแปลงฟูรีเยอย่างรวดเร็ว

ไลบรารีสำหรับคำนวณ FFT ที่เลือกใช้ในหนังสือเล่มนี้เป็นไลบรารีที่พัฒนาโดย เอ็นริเก คอนเดส (Enrique Condes) ชื่อว่า arduinoFFT รุ่น 2.0 (รุ่นนี้มีการใช้งานแตกต่างกับรุ่น 1.x) อันมีจุดเด่นในเรื่องการทำงานได้กับหลายไมโครคอนโทรลเลอร์ทำให้เหมาะสมกับการเรียนรู้และนำไปใช้งานกับระบบอื่นโดยไม่ต้องแก้ไขโปรแกรม โดยการติดตั้งจะต้องติดตั้งผ่านทางตัวบริหารจัดการไลบรารีดังภาพที่ 12-3



ภาพที่ 12-3

ไลบรารี arduinoFFT

ภาพโดย จารุต บุศราทิจ

การเรียกใช้ต้องนำเข้าไฟล์ส่วนหัวชื่อ arduinoFFT.h ดังนี้

```
#include <arduinoFFT.h>
```

ก่อนใช้งานไลบรารี arduinoFFT จะต้องเตรียมบัฟเฟอร์สำหรับเก็บข้อมูลค่าจำนวนจริงและค่าจินตภาพเป็นจำนวนตามค่าแอมพลิจูดที่ต้องการดังนี้

```
#define SAMPLES      1024
#define SAMPLING_FREQ 1000
float ตัวแปรเก็บจำนวนจริง[SAMPLES];
float ตัวแปรเก็บจำนวนจินตภาพ[SAMPLES];
```

ขั้นตอนถัดไปคือสร้างวัตถุประเภท `arduinoFFT` ดังรูปแบบต่อไปนี้ โดยระบุตัวแปรเก็บจำนวนจริง ตัวแปรเก็บจำนวนจินตภาพ ค่าแซมเปิล และค่าความถี่ของแซมเปิล

```
ArduinoFFT<float> ตัวแปรFFT = ArduinoFFT<float>(
    ตัวแปรเก็บจำนวนจริง,
    ตัวแปรเก็บจำนวนจินตภาพ,
    ค่าแซมเปิล,
    ค่าความถี่ของแซมเปิล )
```

โดยที่

ค่าแซมเปิล หรือค่าคงที่ `SAMPLES` จะต้องเป็นค่าตัวเลขจำนวนเต็มที่มาจากค่ากำลัง 2 เช่น 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 เป็นต้น

ค่าความถี่ของแซมเปิล หรือค่าคงที่ `SAMPLING_FREQ` หน่วยเป็น Hz โดยต้องมีค่าเป็นครึ่งหนึ่งของความเร็วของภาคแปลงสัญญาณแอนะล็อกเป็นดิจิทัล โดยในตัวอย่างนี้ใช้ความถี่เป็น 40000Hz และความถี่สูงสุดในการวิเคราะห์ของ FFT เป็นครึ่งหนึ่งของค่าความถี่ของแซมเปิล

การทำงานต้องใส่ค่าระดับสัญญาณดิจิทัลที่แปลงได้เก็บลงตัวแปรเก็บจำนวนจริง โดยกำหนดค่าจำนวนจินตภาพเป็น 0 ดังนี้

```
ตัวแปรเก็บจำนวนจริง[i] = analogRead( ขาสัญญาณแอนะล็อก )
ตัวแปรเก็บจำนวนจินตภาพ[i] = 0
```

เมื่อมีข้อมูลในตัวแปรครบตามจำนวนที่กำหนดเป็นค่าแซมเปิล ขั้นตอนถัดไปคือการคำนวณค่าใช้คำสั่ง `dcRemoval()`, `windowing()`, `compute()` และแปลงกลับด้วยคำสั่ง `complexToMagnitude()` เป็นการสิ้นสุดขั้นตอนของการใช้ชุดคำสั่งของ `arduinoFFT`


```

ตัวแปรFFT.dcRemoval()
ตัวแปรFFT.windowing( FFT_WIN_TYP_HAMMING, FFT_FORWARD )
ตัวแปรFFT.compute( FFT_FORWARD )
ตัวแปรFFT.complexToMagnitude()

```

เมื่อทำงานครบทั้ง 4 คำสั่งจะทำให้ค่าที่เก็บใน ตัวแปรเก็บจำนวนจริง เป็นค่าที่ถูกแปลงแล้ว และการอ่านค่าความถี่สูงสุดของสัญญาณหรือค่าความถี่หลักต้องใช้คำสั่ง majorPeak() ตามรูปแบบของการใช้งานต่อไปนี้

```

ย่านความถี่ที่มีความถี่สูงสุด = ตัวแปรFFT.majorPeak()

```

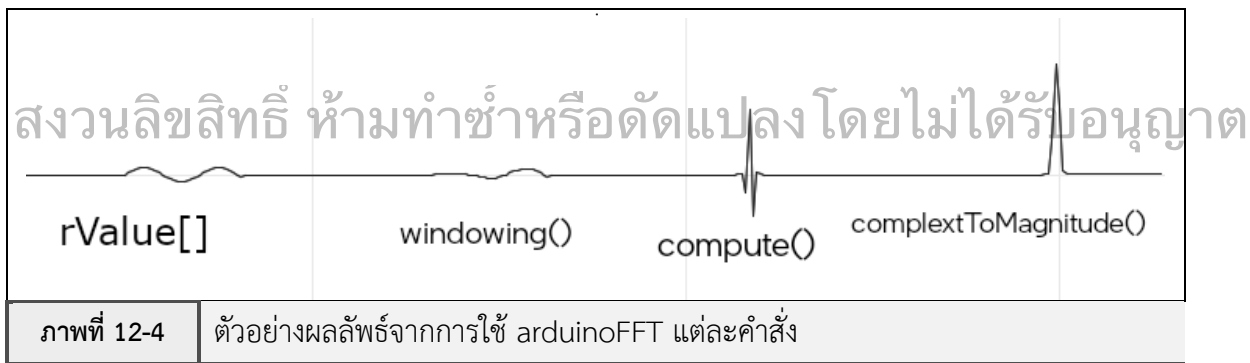
หลังจากการแปลงค่าที่ถูกเก็บในตัวแปรจำนวนจริงเป็นค่าจำนวนประมาณการของความถี่ที่นับได้ ส่วนค่าความถี่ของแต่ละข้อมูลสามารถคำนวณได้จากสมการที่ 12-3

$$\text{ค่าความถี่} = \frac{\text{ลำดับของข้อมูล} \times \text{ค่าความถี่ของแซมเปิล}}{\text{ค่าแซมเปิล}} \quad \text{สมการที่ 12-3}$$

โดยที่

ลำดับของข้อมูล คือ ค่าตัวเลขจำนวนเต็มที่บอกลำดับสมาชิกของ ตัวแปรเก็บจำนวนจริง ดังนั้น จึงมีค่าอยู่ในช่วง [0, ค่าแซมเปิล)

จากชุดคำสั่งของ arduinoFFT จะพบว่า ประกอบด้วยคำสั่งสำหรับประมวลผลข้อมูลดิบหรือสัญญาณที่เก็บใน ตัวแปรเก็บจำนวนจริง จำนวน 3 ชุดคำสั่ง โดยตัวอย่างผลลัพธ์จะเป็นดังภาพที่ 12-4 คือ เมื่อนำข้อมูลดิบที่เก็บใน rValue[] จะต้องนำไปคำนวณค่าด้วยคำสั่ง Windowing() เพื่อได้ข้อมูลสำหรับนำไป Compute() สุดท้ายนำข้อมูลที่คำนวณแปลงเป็นค่าแมกนิจูดด้วยคำสั่ง ComplexToMagnitude() ทำให้มองเห็นว่าความถี่ใดมีค่าเท่าไร โดยในภาพตัวอย่างเป็นการสร้างกราฟรูปไซน์ความถี่ 200Hz เก็บในตัวแปร rValue เพียงความถี่เดียวทำให้เกิดกราฟแสดงการพบความถี่เพียงตำแหน่งเดียว



ภาพโดย จารุต บุศราทิจ

นอกจากเรื่องของลำดับคำสั่งที่ใช้แล้ว สิ่งที่คุณผู้อ่านต้องทำความเข้าใจ คือ ความสัมพันธ์ระหว่างจำนวนแอมป์ที่ใช้ในการสร้างตัวแปรเก็บสัญญาณกับค่าความถี่ เนื่องจากไลบรารีตัวนี้มีข้อกำหนดในการใช้งานให้เก็บข้อมูลใน ตัวแปรเก็บจำนวนจริง เพื่อนำไปคำนวณแล้วนำผลการคำนวณเก็บลง ตัวแปรเก็บจำนวนจริง อีกครั้ง และสุดท้ายในคำสั่ง `complexToMagnitude()` ที่เป็นการสร้างผลลัพธ์ค่าตามลำดับจาก 0 ถึง (ค่าแอมป์-1) จะเป็นที่เก็บจำนวนความถี่ของคลื่นสัญญาณที่พบใน ตัวแปรเก็บจำนวนจริง แต่มีข้อกำหนดว่าจำนวนความถี่ที่เก็บนั้นจะเป็นครึ่งหนึ่งของ ค่าความถี่ของแอมป์ ทำให้ `rValue[0]` เก็บตัวเก็บความถี่ 0Hz และ `rValue[ค่าแอมป์-1]` เก็บค่าความถี่ (ค่าความถี่ของแอมป์/2) Hz โดยค่าอัตราการเพิ่มความถี่แต่ละลำดับสามารถคำนวณได้ดังสมการที่ 12-4

$$\text{อัตราการเพิ่ม} = \text{ลำดับข้อมูล} \times \frac{\text{ค่าแอมป์}}{\text{ค่าความถี่ของแอมป์}} \quad \text{สมการที่ 12-4}$$

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 12-1 เป็นการนำค่าที่อ่านได้จากโมดูลไมโครโฟนมาแสดงผลแบบกราฟด้วย Plotter ของ Arduino IDE โดยใช้เส้นค่าเฉลี่ยเคลื่อนที่ในบรรทัดที่ 7 อ่านข้อมูลจากโมดูลไมโครโฟนในบรรทัดที่ 12 กรองข้อมูลให้มีความถี่สูงมากขึ้นในบรรทัดที่ 13 แปลงข้อมูลที่อ่านได้เป็นค่าแรงดันในบรรทัดที่ 14 และทำการแสดงข้อมูลในระบบเพื่อให้ระบบสามารถนำไปใช้พล็อตกราฟได้โดยมีรูปแบบคือ ชื่อที่1:ค่าที่1,ชื่อที่2:ค่าที่2,...\r\n โดยต้องปิดบรรทัดสุดท้ายที่ส่งข้อมูลด้วยการขึ้นบรรทัดใหม่ด้วย \r\n หรือใช้ฟังก์ชัน `Serial.println()` ก็ได้ ในการต่อการใช้งาน 39 ต่อเข้ากับโมดูลไมโครโฟน

ตัวอย่างโปรแกรม 12-1 แสดงค่าจากไมโครโฟน

บรรทัด	คำสั่ง
1	#include<Filter.h>
2	#define Mic 39
3	int filter = 0;
4	int adc;
5	float volt;
6	float voltInput = 3.3;
7	Moving_average ma(20);
8	void setup(){
9	Serial.begin(115200);
10	}
11	void loop(){
12	adc = analogRead(Mic);
13	filter = ma.filter(adc);
14	volt = float((filter * voltInput) / 4095);
15	Serial.print("ADC:");
16	Serial.print(adc);
17	Serial.print(",");
18	Serial.print("Filter:");
19	Serial.print(filter);
20	Serial.print(", ");
21	Serial.print("Volt:");
22	Serial.println(volt);
23	delay(500);
24	}

ตัวอย่างโปรแกรม 12-2 เป็นการเรียนรู้การใช้ชุดคำสั่ง arduinoFFT เพื่อสร้างผลลัพธ์ดังที่แสดงในภาพที่ 12-4 ด้วยการสร้างคลื่นรูปไซน์ความถี่ 200Hz ที่มีค่าความสูงของคลื่นหรือแอมพลิจูดเป็น 1000 โดยกำหนด ค่าแซมเปิล เป็น 64 ชุดข้อมูล ค่าความถี่ของแซมเปิล เป็น 4000 ครั้งต่อวินาที หลังจากนั้นนำข้อมูลไปประมวลผลด้วย FFT ตามลำดับคำสั่งตามด้วยแสดงผลลัพธ์ของการคำนวณและแสดงค่าความถี่พบสูงสุด ซึ่งได้ผลลัพธ์เป็น 200Hz

ตัวอย่างโปรแกรม 12-2 สร้างคลื่น 200Hz และประมวลผล FFT

บรรทัด	คำสั่ง
1	#include <arduinoFFT.h>
2	#define SAMPLES 64
3	#define SAMPLING_RATE 4000
4	#define SIG_FREQ 200
5	#define SIG_AMP 1000
6	float rValue[SAMPLES];
7	float iValue[SAMPLES];
8	void setup() {
9	Serial.begin(115200);
10	while (!Serial);
11	}
12	void do_sampling() {
13	float ratio= twoPi*SIG_FREQ/SAMPLING_RATE;
14	for (int i = 0; i < SAMPLES; i++) {
15	rValue[i] = (SIG_AMP*sin(i*ratio)+1.0)/
16	2.0;
17	iValue[i] = 0.0;
18	}
19	}
20	void show_rValue() {
21	int last = SAMPLES/2;
22	for (int i=0; i<last; i++) {
23	float hz = (float)i*SAMPLING_RATE/SAMPLES;
24	Serial.print(i);
25	Serial.print(" ");
26	Serial.print(hz);
27	Serial.print("=");
28	Serial.print(rValue[i]);
29	Serial.println(" ");
30	}
31	}
32	void loop() {
33	do_sampling();
34	ArduinoFFT<float> FFT = ArduinoFFT<float>(2π

ตัวอย่างโปรแกรม 12-2 สร้างคลื่น 200Hz และประมวลผล FFT (ต่อ)

บรรทัด	คำสั่ง
35	rValue, iValue,
36	SAMPLES, SAMPLING_RATE);
37	FFT.dcRemoval();
38	FFT.windowing(FFT_WIN_TYP_HAMMING,
39	FFT_FORWARD);
40	FFT.compute(FFT_FORWARD);
41	FFT.complexToMagnitude();
42	show_rValue();
43	Serial.print("max = ");
44	Serial.print((int)(FFT.majorPeak()+0.5));
45	Serial.println("Hz");
46	while(1);
47	}

ตัวอย่างโปรแกรม 12-3 เป็นการนำค่าสัญญาณที่ได้จากไมโครโฟนที่เชื่อมต่อเข้ากับขา P36 ที่ทำหน้าที่แปลงสัญญาณแอนะล็อกเป็นดิจิทัลด้วยคำสั่ง readAnalog() ไปเก็บลงตัวแปร rValue แล้วนำไปผ่านไลบรารี arduinoFFT เพื่อแยกช่วงความถี่ในย่านเสียงเบส (Bass หรือ Low) อยู่ที่ 60Hz ถึง 250Hz และนำค่าจำนวนความถี่ที่คำนวณได้ไปแปลงเป็นตัวอักษร > จำนวน 0 ถึง 4 ตัว ตามระดับของค่าที่นับได้ดังนี้

- ช่วงค่า 0-49 ไม่แสดง
- ช่วงค่า 50-99 แสดง >
- ช่วงค่า 100-149 แสดง >>
- ช่วงค่า 150-199 แสดง >>>
- ช่วงค่า 200 ขึ้นไปแสดง >>>>

ตัวอย่างโปรแกรม 12-3 แสดงระดับจำนวนความถี่ของเสียงในช่วงเบส

บรรทัด	คำสั่ง
1	#include <arduinoFFT.h>
2	#define L1 50
3	#define L2 100
4	#define L3 150
5	#define L4 200
6	#define MIC_PIN 36
7	#define SAMPLES 64
8	#define SAMPLING_RATE 2000 // Hz
9	float rValue[SAMPLES];
10	float iValue[SAMPLES];
11	void setup() {
12	pinMode(MIC_PIN, INPUT);
13	Serial.begin(115200);
14	}
15	void do_sampling() {
16	unsigned int period;
17	unsigned long usec;
18	usec = micros();
19	for (int i = 0; i < SAMPLES; i++) {
20	rValue[i] = (float)analogRead(MIC_PIN);
21	iValue[i] = 0.0f;
22	while (micros() - usec < period);
23	usec += period;
24	}
25	}
26	void do_report() {
27	int last = SAMPLES / 2;
28	for (int i = 0; i < last; i++) {
29	float hz = (float)i*SAMPLING_RATE/SAMPLES;
30	float counter = 0.0;
31	if (hz >= 60) {
32	if (hz <= 250) {
33	counter += rValue[i];
34	}

ตัวอย่างโปรแกรม 12-3 แสดงระดับจำนวนความถี่ของเสียงในช่วงเบส (ต่อ)

บรรทัด	คำสั่ง
35	}
36	if (counter > L4) {
37	Serial.println(">>>>");
38	} else if (counter > L3) {
39	Serial.println(">>>");
40	} else if (counter > L2) {
41	Serial.println(">>");
42	} else if (counter > L1) {
43	Serial.println(">");
44	}
45	}
46	}
47	void loop() {
48	do_sampling();
49	ArduinoFFT<float> FFT = ArduinoFFT<float>(50
51	rValue, iValue, 52
53	SAMPLES, SAMPLING_RATE);
54	FFT.dcRemoval();
55	FFT.windowing(FFT_WIN_TYP_HAMMING, 56
57	FFT_FORWARD);
58	FFT.compute(FFT_FORWARD);
59	FFT.complexToMagnitude();
	do_report();
	delay(1);
	}

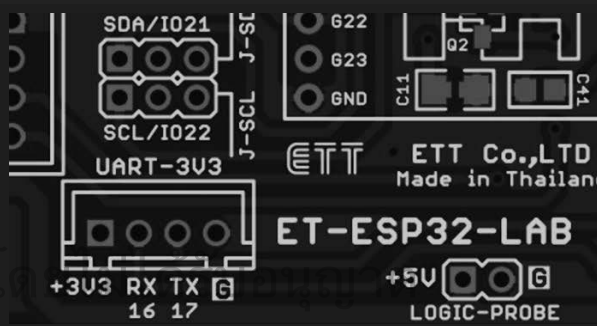
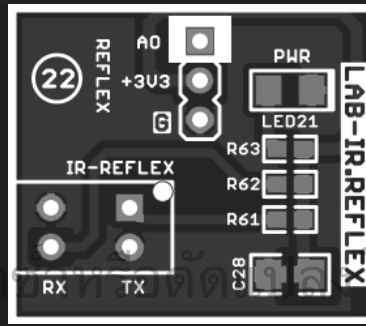
> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-ADC MIC เป็นการใช้งานไมโครโฟนแบบมีความเร็วในการตอบสนองต่อเสียงพร้อมมีตัวขยายเสียงขนาดเล็กอยู่ในตัวทำให้ไม่ต้องทำวงจรขยายเสียงเข้ามาใช้งาน และการนำมาใช้กับ ESP32 จะต้องทำการแปลงค่าจากแรงดันที่เป็นสัญญาณแบบแอนะล็อกให้เป็นสัญญาณดิจิทัลที่เป็นการแทนค่าแรงดันด้วยตัวเลขจำนวนเต็มค่าหนึ่งด้วยการใช้คำสั่ง `analogRead()` และสามารถนำค่าที่ได้ช่วงระยะหนึ่งมาประมวลผลด้วยการแปลงฟูรีเยแบบ FFT เพื่อแยกสัญญาณเสียงออกมาเป็นการนับความถี่ของสัญญาณความถี่คลื่นไซน์ ด้วยวิธีการนี้ทำให้สามารถนำลักษณะของสัญญาณเสียงไปประมวลผลต่อไปได้ เช่น การตัดเสียงรบกวน การผสมช่องเสียง การลดความดังของความถี่เสียงในช่วงที่กำหนด เป็นต้น

บรรณานุกรม

- วิกิพีเดีย. (2565). *โอมแซ่ฟ ฟูรีเย*. สืบค้นเมื่อ เมษายน 23, 2567, จาก [https://th.wikipedia.org/wiki/วิกิพีเดีย. \(2565\). การแปลงฟูรีเย](https://th.wikipedia.org/wiki/วิกิพีเดีย. (2565). การแปลงฟูรีเย). สืบค้นเมื่อ เมษายน 23, 2567, จาก [https://th.wikipedia.org/wiki/Enrique Condes. \(2024\). arduinoFFT 2.0](https://th.wikipedia.org/wiki/Enrique Condes. (2024). arduinoFFT 2.0). สืบค้นเมื่อ เมษายน 23, 2567, จาก <https://github.com/kosme/arduinoFFT>https://en.wikipedia.org/wiki/Fast_Fourier_transform
- hideakitai. (2024). *Filters.h*. สืบค้นเมื่อ เมษายน 23, 2567, จาก <https://www.arduino.cc/reference/en/libraries/filters/>
- MAXIM. (2012). *MAX4466 Datasheet*. สืบค้นเมื่อ เมษายน 23, 2567, จาก <https://www.analog.com/media/en/technical-documentation/data-sheets/MAX4465-MAX4469.pdf>
- Texas Instruments. (2023). *LM386 Datasheet*. สืบค้นเมื่อ เมษายน 23, 2567, จาก <https://www.ti.com/lit/ds/symlink/lm386.pdf>
- Texas Instruments. (2024). *LM358 Datasheet*. สืบค้นเมื่อ เมษายน 23, 2567, จาก <https://www.ti.com/lit/ds/symlink/lm358.pdf>
- WIKIPEDIA. (2024). *Fast Fourier transform*. สืบค้นเมื่อ เมษายน 23, 2567, จาก https://en.wikipedia.org/wiki/Fast_Fourier_transform



บทที่ 13 LAB-IR.REFLEX

เซ็นเซอร์เป็นโมดูลสำคัญของการนำเข้าข้อมูล ในบทนี้กล่าวถึงเซ็นเซอร์แสงในจำพวกอินฟราเรด (IR: Infrared) ที่เรียกว่า รีเฟล็กซ์ (Reflex) อันเป็นโมดูลทดลอง LAB-IR.REFLEX ของบอร์ด ET-ESP LAB V1 โดยอธิบายถึงหลักการทำงานของวงจรพร้อมตัวอย่างโปรแกรม

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรการทำงานของ LAB-IR.REFLEX หรือนำโมดูลไปประยุกต์ใช้เพื่อตรวจการสะท้อนของพื้นผิวที่มีสีที่แตกต่างกัน ประยุกต์ใช้ตรวจจับเส้นขาว/ดำสำหรับใช้เป็นเส้นนำของหุ่นยนต์เดินตามเส้น นำไปพัฒนาเป็นเซ็นเซอร์ของเมาส์ เป็นต้น

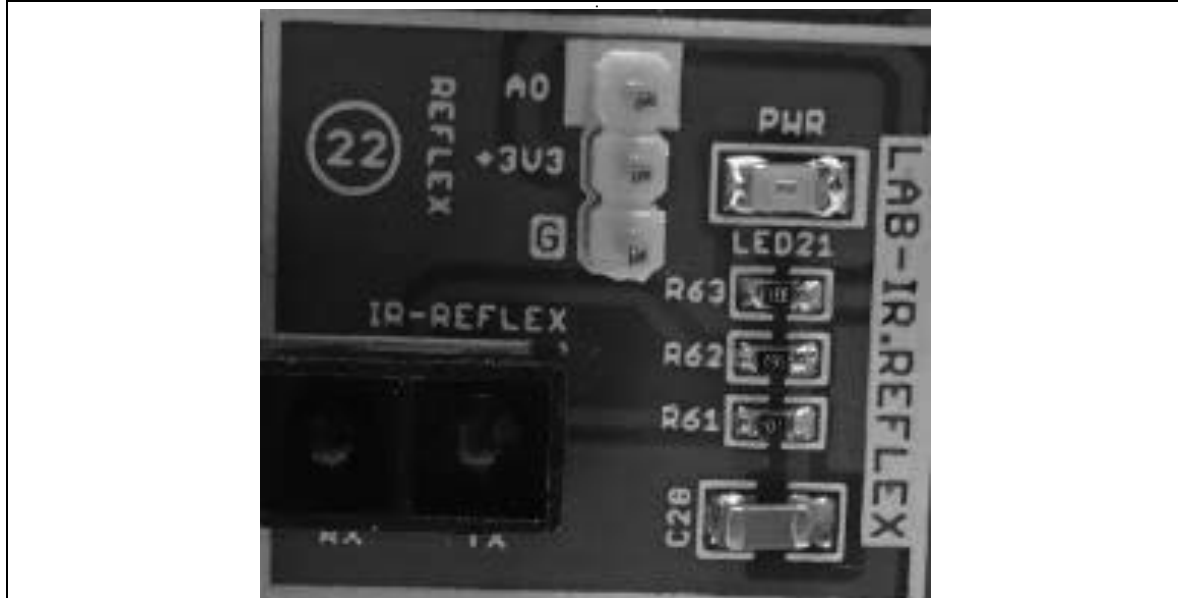
> แนะนำ LAB-IR.REFLEX

ภาค LAB-IR.REFLEX เป็นโมดูลสำหรับทดลองการนำเข้าข้อมูลแบบแอนะล็อกจากเซ็นเซอร์แบบอินฟราเรด ดังภาพที่ 13-1 โดยภาครับและภาคส่งสัญญาณอินฟราเรดอยู่ฝั่งเดียวกัน อาศัยการยิงลำแสงจากภาคส่งเพื่อสะท้อนกับวัตถุที่ต้องการตรวจจับ เมื่อลำแสงชนเข้ากับวัตถุจะสะท้อนกลับมาที่ภาครับ จะได้ค่าของการสะท้อนที่ต่างกันตามสีของวัตถุ โดยสีขาวจะมีการสะท้อนได้ดีที่สุดและสีดำสะท้อนได้น้อยที่สุด

ขาของโมดูล LAB-IR.REFLEX ประกอบด้วย 3 ขา คือ ขากราวด์ (G) ขาแหล่งจ่ายแรงดันไฟฟ้ากระแสตรงที่รองรับแรงดัน +3V3 และขา AO ที่เป็นสัญญาณ แอนะล็อกที่ได้จากเซ็นเซอร์อินฟราเรด ภาครับการทำงานของโมดูลเป็นการอาศัยการสะท้อนของแสงอินฟราเรดที่ส่งจากภาคส่งไปยังพื้นผิวของวัตถุตกกระทบ หลังจากนั้นแสงอินฟราเรดจึงสะท้อนกลับเข้ามาที่ภาครับ โดยความเข้มของแสงอินฟราเรดที่ภาครับแปรผันตามสีของวัตถุนั้นสามารถสะท้อนและดูดซับแสงอินฟราเรดได้ดีเพียงใด ดังภาพที่ 13-2 แต่อย่างไรก็ดี ระยะห่างระหว่างพื้นผิวกับชุดเซ็นเซอร์มีผลต่อการสะท้อน และสภาพแสง

แวดล้อมมีผลต่อการสะท้อนเช่นเดียวกัน ดังนั้นในการประยุกต์ใช้งานต้องมีการตั้งค่าการทำงานเพื่อให้เหมาะสมกับสภาพแสง ณ ขณะนั้นเสมอ

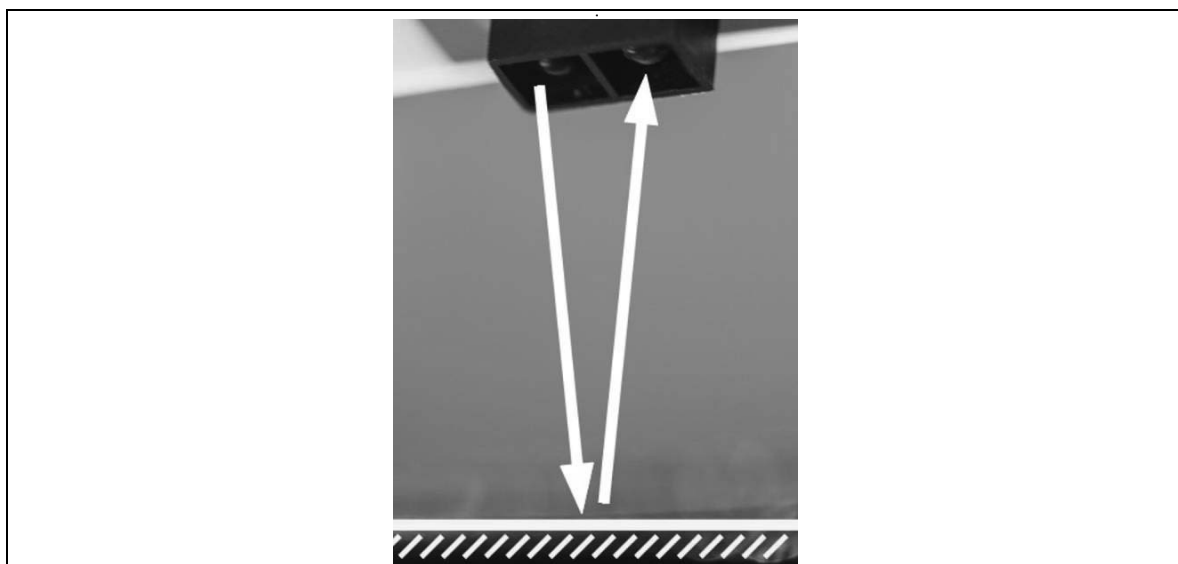
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



ภาพที่ 13-1

โมดูล LAB-IR.REFLEX

ภาพโดย จารุต บุศราทิจ



ภาพที่ 13-2

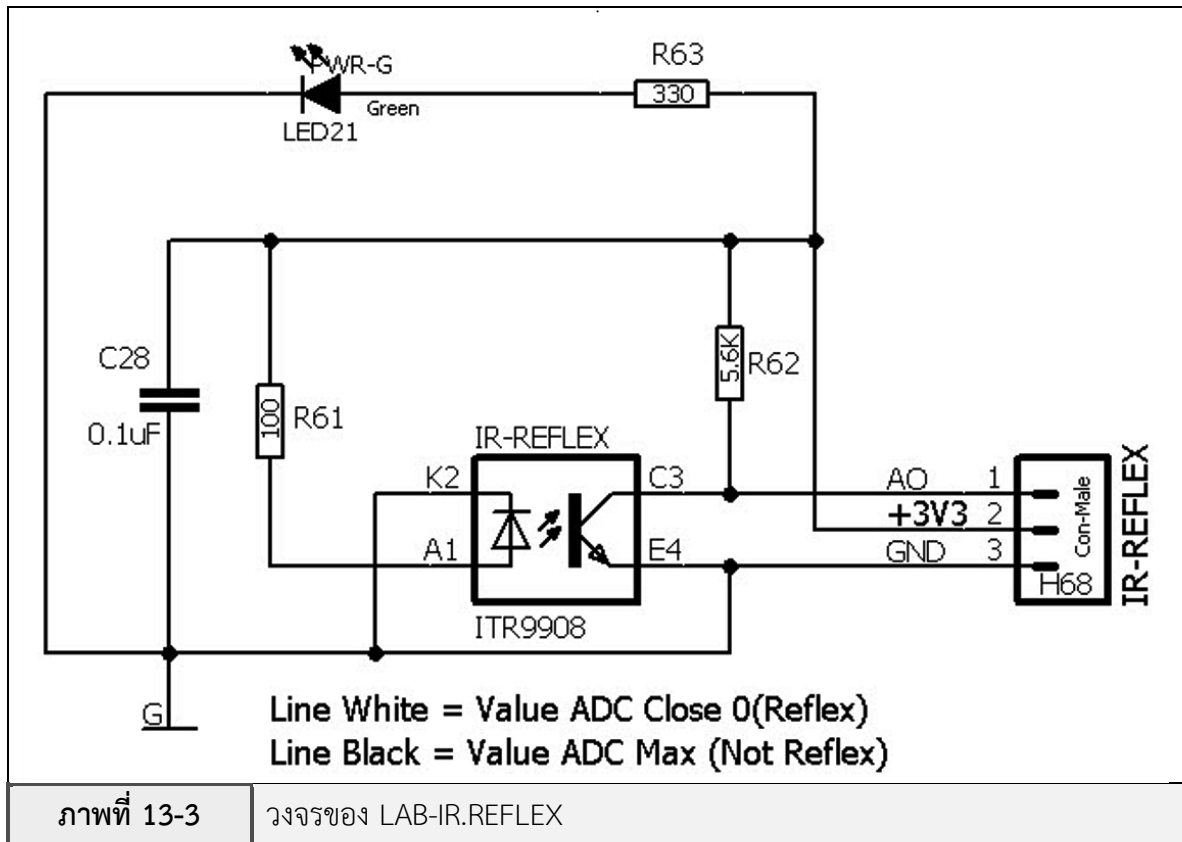
การสะท้อนของแสงอินฟราเรด

ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

วงจรของ LAB-IR.REFLEX ที่ใช้โมดูล ITR-9908 เป็นดังภาพที่ 13-3 จะพบว่าถ้าบอร์ดพร้อมทำงานจะมีสัญญาณไฟของหลอด LED21 สีเขียวติด และการเชื่อมต่อสำหรับตัวอย่างโปรแกรมโดยใช้ขา ADC ของ ESP32 เป็นขานำเข้าสัญญาณจากบอร์ดเซ็นเซอร์ผ่านทางขั้ว A0



ภาพที่ 13-3

วงจรของ LAB-IR.REFLEX

ภาพโดย จารุต บุศราทิจ

โมดูล ITR-9908 มีคุณสมบัติดังนี้

- ทำงานที่อุณหภูมิ -25°C ถึง 85°C
- ใช้คลื่นยาว 940nm
- ทำงานที่แรงดันไฟฟ้ากระแสตรงไม่เกิน 5V

> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 13-1 เป็นตัวอย่างโปรแกรมอ่านค่าความเข้มแสงที่สะท้อนกลับและแสดงบนโมดูล OLED โดยมีการอ่านค่าจากโมดูล IR Reflex เป็นค่าแอนะล็อกในบรรทัดที่ 23 ทำการแปลงเป็นแรงดันในบรรทัดที่ 24 และแสดงค่าทั้งสองบนโมดูล OLED ในบรรทัดที่ 26-31 โดยมีการต่อขา 36 เข้ากับโมดูล IR Reflex

ตัวอย่างโปรแกรม 13-1 อ่านค่าจาก IR Reflex ไปแสดงที่ OLED

บรรทัด	คำสั่ง
1	#include <Adafruit_GFX.h>
2	#include <Adafruit_SH1106.h>
3	#define SDA 21
4	#define SCL 22
5	#define IRF 36
6	Adafruit_SH1106 display(SDA, SCL);
7	int adc;
8	float volt;
9	float voltInput = 3.3;
10	void setup(){
11	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
12	display.clearDisplay();
13	display.display();
14	delay(100);
15	display.setTextSize(1);
16	display.setTextColor(WHITE);
17	display.setCursor(0, 0);
18	pinMode(IRF, INPUT);
19	}
20	void loop(){
21	display.clearDisplay();
22	display.setCursor(0, 0);
23	adc = analogRead(IRF);
24	volt = float((adc * voltInput) / 4095);
25	delay(100);
26	display.print("ADC = ");

ตัวอย่างโปรแกรม 13-1 อ่านค่าจาก IR Reflex ไปแสดงที่ OLED (ต่อ)

บรรทัด	คำสั่ง
27	display.println(adc);
28	display.print("Volt = ");
29	display.print(volt, 1);
30	display.println(" V");
31	display.display();
32	delay(1000);
33	}

ตัวอย่างโปรแกรม 13-2 เป็นตัวอย่างโปรแกรมตรวจสอบพื้นที่ขาวหรือดำโดยตรวจสอบจากค่าแอนะล็อกที่รับได้จากโมดูล IR Reflex หากต่ำกว่า 2048 จะถือว่าเป็นสีดำ หากมากกว่า 2048 ถือว่าเป็นสีขาว โดยค่านี้สามารถปรับแต่งได้ตามความเหมาะสมของแสงในแต่ละพื้นที่ โค้ดมีการอ่านค่าของโมดูล IR Reflex ในบรรทัดที่ 10 และตรวจสอบหากค่าน้อยกว่าขอบที่กำหนดคือ 2048 ให้ไฟของ LED ดับ หากค่ามากกว่า 2048 ให้ LED ติด ในบรรทัดที่ 11-15 มีการต่อขา 0 เข้ากับ LED0 และต่อขา 36 เข้ากับโมดูล IR Reflex

ตัวอย่างโปรแกรม 13-2 ตรวจสอบพื้นที่ขาว/ดำ

บรรทัด	คำสั่ง
1	#define led1 0
2	#define IRF 36
3	int adc;
4	int bound = 2048;
5	void setup(){
6	pinMode(led1, OUTPUT);
7	pinMode(IRF, INPUT);
8	}
9	void loop(){
10	adc = analogRead(IRF);
11	if(adc < bound){
12	digitalWrite(led1, HIGH);
13	}else{
14	digitalWrite(led1, LOW);
15	}
16	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-IR.REFLEX เป็นโมดูลสำหรับทดลองการนำเข้าข้อมูลแบบแอนะล็อกจากเซ็นเซอร์แบบอินฟราเรดที่ติดตั้งให้ภาครับและภาคส่งสัญญาณอินฟราเรดอยู่ฝั่งเดียวกันเพื่ออาศัยหลักของสะท้อนพื้นผิวด้วยการยิงลำแสงจากภาคส่งเพื่อสะท้อนกับวัตถุที่ต้องการตรวจจับ โดยค่าที่ได้กลับมานั้นจะแตกต่างกันไปตามการดูดซับแสงอินฟราเรดของพื้นผิว เช่นพื้นผิวสีสว่างจะสะท้อนได้ดีกว่าพื้นผิวที่มืดทึบ ทำให้พื้นผิวที่เป็นสีดำจึงสะท้อนได้น้อยกว่าพื้นผิวสีขาว เป็นต้น ดังนั้น การเขียนโปรแกรมจึงอาศัยการอ่านค่าจากสัญญาณแอนะล็อกมาตัดสินใจว่าค่าที่อ่านมาควรอยู่ในกลุ่มใด

แต่อย่างไรก็ดี การประยุกต์ใช้การสะท้อนแสงจะมีค่าที่อ่านได้แปรผันตามสภาพของแสงแวดล้อม ดังนั้น ในการใช้งานจริงควรหาวิธีการควบคุมสภาพแสงรอบข้างให้บริเวณเซ็นเซอร์ให้น้อยที่สุด หรือต้องมีการตั้งค่าเงื่อนไขที่เหมาะสมทุกครั้งที่ต้องใช้งาน

บรรณานุกรม

EVERLIGHT. (2009). *ITR-9908 Datasheet*. สืบค้นเมื่อ เมษายน 26, 2567, จาก

https://www.mouser.com/datasheet/2/143/ever_s_a0007431488_1-2493369.pdf



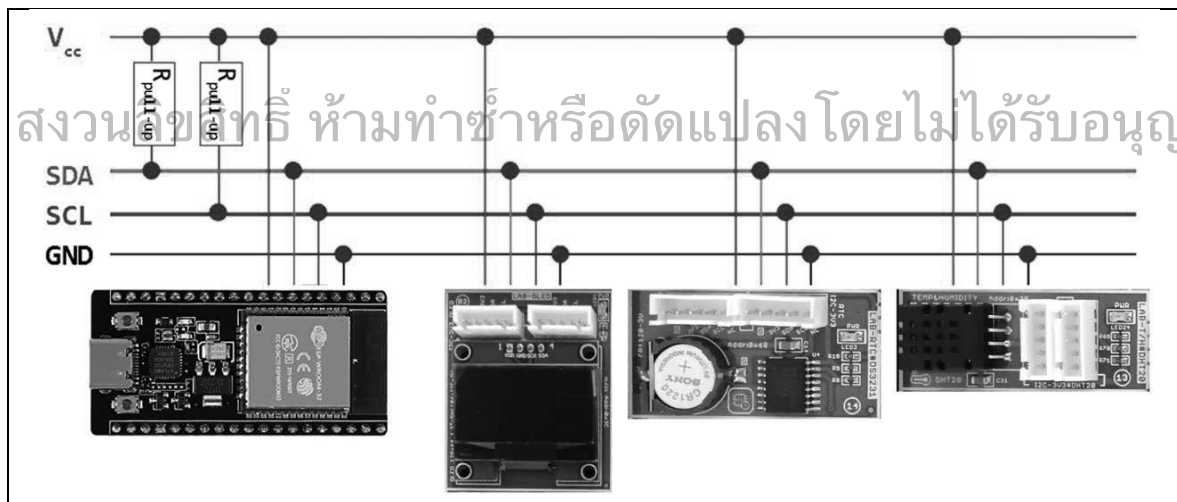
บทที่ 14 บัส I²C และ SPI

บทนี้กล่าวถึงการสื่อสารกับอุปกรณ์ภายนอกด้วยโปรโตคอลแบบ I²C และ SPI โดยทั้ง 2 มาตรฐานการเชื่อมต่อเป็นที่นิยมและมีอุปกรณ์ทั้งเซ็นเซอร์/จอแสดงผลมารองรับการสื่อสารบนระบบทั้งสอง เนื้อหาในบทนี้กล่าวถึงประวัติ รายละเอียด หลักการทำงาน และชุดคำสั่งของ I²C และ SPI บัสพร้อมตัวอย่างใช้งานที่ต้องนำไปใช้งานในบทที่ 15 ถึง 22

> แนะนำ I²C

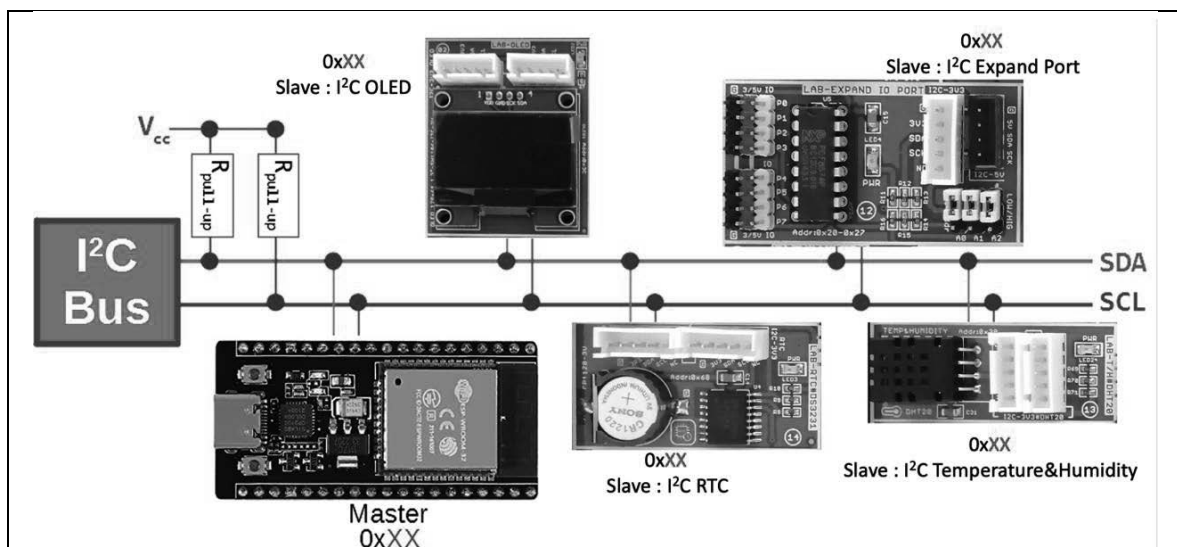
I²C Bus เป็นโปรโตคอลการสื่อสารที่ออกแบบโดยบริษัท Philips สำหรับการสื่อสารระหว่างไอซีโดยใช้สายสัญญาณเพียง 2 เส้น ที่ทำงานแบบ Bi-Directional Line ที่เรียกว่า SDA (Serial Data Line) สำหรับรับ/ส่งข้อมูล และ SCL (Serial Clock Line) สำหรับควบคุมสัญญาณนาฬิกาการรับส่ง เงื่อนไขหลักของการใช้งาน คือ ต้องต่อตัวต้านทานพูลอัพ (Pull Up) กับแรงดัน +5VDC หรือ +3.3V ตลอดเวลา และต้องต่อตัวเก็บประจุไฟฟ้าความจุไม่เกิน 400pF ระหว่างสาย SDA และ SCL ดังภาพที่ 14-1 และ 14-2 โดยความเร็วในการสื่อสารของระบบบัส I²C อยู่ในช่วง 100 kbps ถึง 400kbps และเข้าถึงข้อมูลแบบ 7 หรือ 10 บิต

บนบอร์ด ET-ESP32 LAB V1 มีขั้วต่อเสริมสำหรับบัส I²C ดังภาพที่ 14-3 ซึ่งเป็นขั้วต่อที่มีขา SCL ต่อเข้ากับ P22 ที่เรียกว่าขา SCL1, SDA ต่อเข้ากับ P21 หรือขา SDA1, 5VDC และ GND โดยขา SCL และ SDA ได้ถูกต่อที่ผ่านวงจรปรับระดับแรงดันบนบอร์ดแล้วจึงไม่จำเป็นต้องต่อ R พูลอัพที่ขา SDA/SCL กับอุปกรณ์ I²C ภายนอกบอร์ดอีก



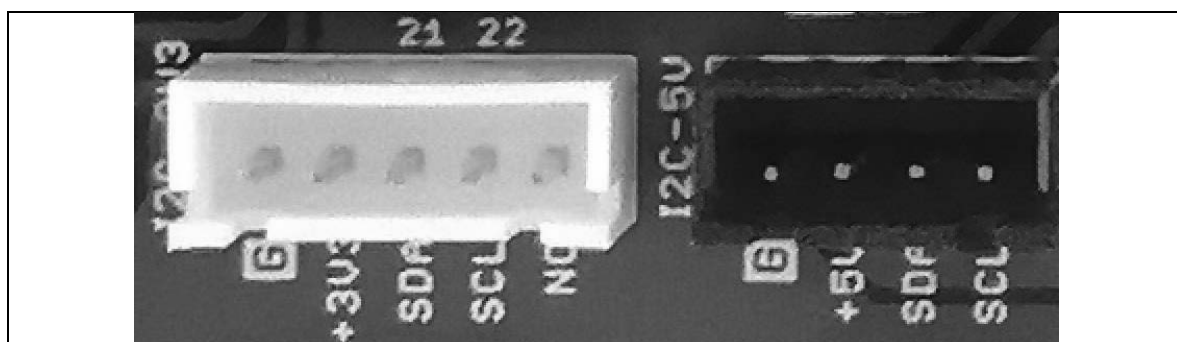
ภาพที่ 14-1 การต่อเชื่อมไอซีกับบัส I²C

ภาพโดย จารุต บุศราทิจ



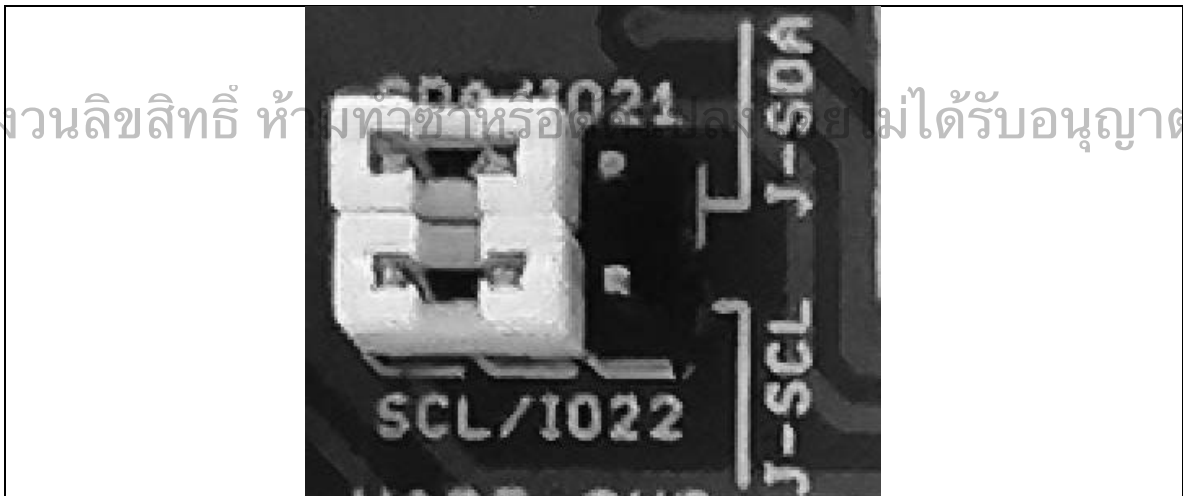
ภาพที่ 14-2 การต่อเชื่อมไอซีกับบัส I²C

ภาพโดย จารุต บุศราทิจ



ภาพที่ 14-3 ขั้วต่อ I²C แบบ 3V3 และ 5V ของบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ



ภาพที่ 14-4

ขั้วต่อ J-SCL/J-SDA ของบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ

นอกจากนี้ ผู้ใช้งานสามารถกำหนดให้ขา P21 และ P22 เชื่อมต่อหรือไม่เชื่อมต่อกับขา SDA/SCL ของการสื่อสารอนุกรมแบบ I²C ได้ด้วยการใช้จัมเปอร์ (jumper) กำหนดการเชื่อมต่อ J-SCL และ J-SDA ตามภาพที่ 14-4

ไมโครคอนโทรลเลอร์ ESP32 รองรับฮาร์ดแวร์ I²C จำนวน 2 ชุดดังนี้ แต่โดยปกติแล้ว ขาในชุดที่ 2 นั้นใช้ร่วมกับการทำ DAC

- I2C id=0
 - SCL เป็นขา P18
 - SDA เป็นขา P19
- I2C id=1
 - SCL เป็นขา P25
 - SDA เป็นขา P26

การใช้บัส I²C ต้องเรียกผ่านไลบรารีที่ถูกเก็บไว้ในไฟล์ส่วนหัว **Wire.h** โดยที่ชุดคำสั่งหรือเมธอดของไลบรารีสำหรับใช้งานต้องเริ่มด้วยการเปิดการทำงานของบัสดังนี้

```
Wire.begin()
```

สำหรับกรณีที่มีการเชื่อมต่อขาไม่ตรงกับการตั้งค่าของฮาร์ดแวร์ เช่น เชื่อมต่อขา SDA เข้ากับ P21 และ SCL เข้ากับขา P22 ของ ESP32 จะต้องระบุหมายเลขขาเฉพาะเจาะจงตามรูปแบบของการใช้งานดังนี้

```
Wire.begin( หมายเลขขาSDA, หมายเลขขาของSCL )
```

จากรูปแบบการใช้งานข้างต้นทำให้โค้ดของโปรแกรมสำหรับกำหนดขา SDA เป็น P21 และขา SCL เป็น P22 เป็นดังนี้

```
Wire.begin( 21, 22 );
```

คำสั่งสำหรับขอเข้าถึงอุปกรณ์ ณ ตำแหน่งของอุปกรณ์ ที่กำหนดเป็นดังนี้

```
Wire.beginTransmission( ตำแหน่งของอุปกรณ์ )
```

การยกเลิกการเข้าถึงอุปกรณ์ที่ขอเข้าถึงไว้มีรูปแบบดังนี้ โดยเมธอดคืนค่า 0 ถ้าการเข้าถึงถูกยกเลิก

```
Wire.endTransmission()
```

เมื่อมีการร้องขอข้อมูลผ่านบัส ระบบจะนำข้อมูลที่ได้มาเก็บในบัฟเฟอร์ ซึ่งคำสั่งที่ทำหน้าที่ตรวจสอบจำนวนข้อมูลในบัฟเฟอร์มีรูปแบบดังนี้

```
จำนวนข้อมูลในบัฟเฟอร์ = Wire.available()
```

วิธีการนำข้อมูลจำนวน 1 ไบต์ออกจากบัฟเฟอร์ใช้คำสั่งดังนี้

```
byteData = Wire.read()
```

กรณีที่ต้องการส่งข้อมูลไปยังอุปกรณ์ผ่านบัสสามารถทำได้ 3 วิธี คือ ส่งไบต์ข้อมูล ส่ง สตริง และ ส่ง ก้อนข้อมูล พร้อมระบุ จำนวนไบต์ของข้อมูล ดังนี้

```
void Wire.write( ไบต์ข้อมูล )
```

```
void Wire.write( สตริง )
```

```
void Wire.write( ก่อนข้อมูล, จำนวนไบต์ของข้อมูล )
```

คำสั่งร้องขออ่านข้อมูลจากอุปกรณ์เพื่อนำมาเก็บในบัฟเฟอร์ต้องระบุตำแหน่งของอุปกรณ์และจำนวนไบต์ที่ต้องการอ่านดังนี้

```
void Wire.requestFrom( ตำแหน่งอุปกรณ์,  
จำนวนไบต์ที่ต้องการอ่าน )
```

การปรับความเร็วในการสื่อสารของบัส I²C ใช้คำสั่งกำหนดค่าดังนี้

```
void Wire.SetClock( ความถี่สัญญาณนาฬิกาในการสื่อสาร )
```

คำสั่งสำหรับกำหนดฟังก์ชันตอบสนองเมื่อได้รับข้อมูลจากอุปกรณ์ภายนอก

```
void Wire.onReceive( ชื่อฟังก์ชัน )
```

กำหนดฟังก์ชันตอบสนองเมื่อมีการร้องขอจากบัส I²C ทำให้เกิดการเรียกใช้ ชื่อฟังก์ชัน อัตโนมัติ มีรูปแบบของคำสั่งดังนี้

```
void Wire.onRequest( ชื่อฟังก์ชัน )
```

ด้วยเนื่องจากบริษัท espressif กำหนดให้ขาเชื่อมต่อบัสสาย SDA เป็นขา P21 และ SCL เป็นขา P22 อาจจะทำให้ไม่ตรงกับขาของวงจรอื่น จึงทำให้ต้องเรียกใช้คำสั่ง setPins() สำหรับกำหนดหมายเลขขาที่ทำหน้าที่ SDA และ SCL ใหม่ดังนี้

```
ผลลัพธ์ = Wire.setPins( หมายเลขขาSDA , หมายเลขขาSCL )
```

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

โดยที่

ผลลัพธ์ เป็นผลจากการตั้งค่า ถ้าคืนค่าเป็น true หมายถึงการตั้งค่าขาดำเนินการสำเร็จ แต่ถ้าผิดพลาดจะได้ผลลัพธ์เป็น false

ตัวอย่างการตรวจสอบอุปกรณ์

ตัวอย่างการตรวจสอบว่ามีอุปกรณ์ I2C_DEVICE หรือไม่ เขียนได้ดังนี้

```
Wire.beginTransmission( I2C_DEVICE );  
if (Wire.endTransmission() == 0) {  
    // พบอุปกรณ์  
}  
else {  
    // ไม่พบอุปกรณ์  
}
```

ตัวอย่างการเขียนข้อมูลไปยังอุปกรณ์

การเขียนข้อมูลไปยังอุปกรณ์มีขั้นตอน คือ (1) เชื่อมต่อ (2) ส่งข้อมูลตามลำดับไบนารี และ (3) ปิดการเชื่อมต่อ ดังโค้ดต่อไปนี้

```
Wire.beginTransmission( I2C_DEVICE );  
Wire.write( ไบต์ข้อมูล );  
Wire.endTransmission();
```

ตัวอย่างการอ่านข้อมูลจากอุปกรณ์

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

การอ่านข้อมูลจากอุปกรณ์ประกอบไปด้วย 3 ขั้นตอนหลัก คือ ขอเข้าถึงตำแหน่งข้อมูลของอุปกรณ์ การอ่านมาเก็บในบัฟเฟอร์ และการเข้าถึงบัฟเฟอร์เพื่อนำข้อมูลออก ดังนี้

ขั้นตอน 1

```
Wire.beginTransmission( I2C_DEVICE );
Wire.write( ตำแหน่งของข้อมูล );
Wire.endTransmission();
```

ขั้นตอน 2

```
Wire.requestFrom( I2C_DEVICE, จำนวนไบต์ที่ต้องการอ่าน );
```

ขั้นตอน 3

```
while (Wire.available( ) ) {
    ข้อมูล = Wire.read( );
}
```

> แนะนำ SPI

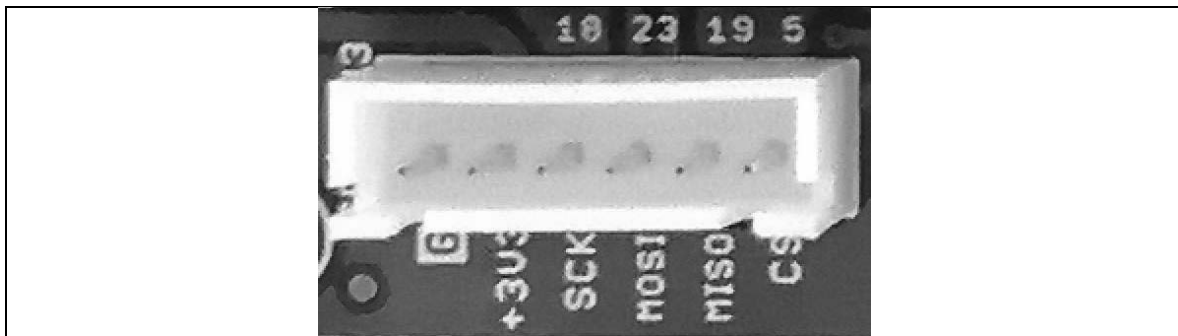
การสื่อสารแบบ SPI (Serial Peripheral Interface) คือ วิธีการสื่อสารข้อมูลอนุกรมแบบซิงโครนัส (Synchronous) ถูกพัฒนาตั้งแต่ ค.ศ. 1980 เพื่อใช้สื่อสารระหว่างอุปกรณ์สมองกลฝังตัวด้วยการกำหนดสายเชื่อมต่อที่ทำหน้าที่เป็นขาให้สัญญาณนาฬิกาการระหว่างอุปกรณ์ และมีขาสำหรับรับและส่ง นอกจากนี้มีการแบ่งประเภทของอุปกรณ์เป็น 2 ประเภท คือ อุปกรณ์สเลฟ (Slave) หรืออุปกรณ์ที่มีหน้าที่รับคำสั่ง และอุปกรณ์มาสเตอร์ (Master) ที่เป็นผู้ควบคุมการทำงานของอุปกรณ์สเลฟ โดยอุปกรณ์มาสเตอร์สามารถมีอุปกรณ์สเลฟภายใต้การดูแลได้หลายชิ้น

ขาสำหรับสื่อสารบัส SPI ที่เชื่อมต่อกับบอร์ด ESP32 คือ ขา SCLK ที่ทำหน้าที่ส่งสัญญาณนาฬิกาจากบอร์ด ESP32 ไปยังอุปกรณ์สเลฟ ขา MOSI ที่ทำหน้าที่ส่งออกข้อมูลไปยังอุปกรณ์สเลฟ และขา MISO ที่ทำหน้าที่นำเข้าข้อมูลจากอุปกรณ์ สเลฟ โดยบอร์ด ET-ESP32 LAB V1.0 มีขาเชื่อมต่อเตรียมไว้ให้ใช้งานดังภาพที่ 14-5

สำหรับชิพ ESP32 มีพอร์ต SPI ให้ใช้งาน 3 ชุด แต่ใช้งานได้โดยผู้ใช้งานจำนวน 2 ชุดดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

- SPI id=1 หรือ HSPi
 - SCLK ใช้ขา P14
 - MISO ใช้ขา P12
 - MOSI ใช้ขา P13
- SPI id=2 หรือ VSPI
 - SCLK ใช้ขา P18
 - MISO ใช้ขา P23
 - MOSI ใช้ขา P19



ภาพที่ 14-5

ขั้วต่อ SPI ของบอร์ด ET-ESP32 LAB V1.0

ภาพโดย จารุต บุศราทิจ

จากภาพที่ 14-5 จะพบว่าบอร์ด ET-ESP32 LAB V1.0 กำหนดการเชื่อมต่อของขาเข้ากับ SPI ID=2 หรือ VSPI โดยใช้ขา P5 เป็นขา CS ดังนี้

- SCK เชื่อมต่อกับ P18
- MOSI เชื่อมต่อกับ P23
- MISO เชื่อมต่อกับ P19
- CS เชื่อมต่อกับ P5

การเรียกใช้งานคลาส SPI ต้องนำเข้าไฟล์ส่วนหัว SPI.h ดังนี้

```
#include <SPI.h>
```

การสั่งเริ่มต้นและสิ้นสุดการทำงานของคลาส SPI ใช้คำสั่ง `begin()` และ `end()` ดังรูปแบบต่อไปนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```
SPI.begin()
```

```
SPI.end()
```

กรณีที่ต้องการตั้งค่าการทำงานของ การสื่อสารผ่านบัส SPI ให้ใช้การตั้งค่าจากคำสั่ง `SPISettings()` เพื่อสร้างวัตถุเก็บค่าการตั้งค่า ตามรูปแบบดังนี้ โดยความเร็วในการส่งขึ้นอยู่กับความเร็วสูงสุดที่ไม่โครคอนโทรลเลอร์แต่ละชนิดกำหนด และมีปัจจัยเรื่องคุณภาพของสายสัญญาณ/ความยาวของสายสัญญาณที่ส่งผลต่อความเร็วด้วยเช่นกัน การเรียงลำดับข้อมูลเป็นการกำหนดวิธีการส่งข้อมูลบิตว่าเป็นแบบ `MSBFIRST` หรือ `LSBFIRST` สุดท้ายโหมดข้อมูลของการสื่อสารของพอร์ต SPI ประกอบด้วย 4 โหมดดังนี้

```
SPI.SPISettings ( ความเร็วสูงสุดในการส่ง, การเรียงลำดับข้อมูล, โหมดข้อมูล )
```

การนำค่า `SPISettings` ไปใช้นั้นมักใช้ในการสั่งเริ่มต้นสื่อสารด้วยคำสั่ง `beginTransaction()` เพื่อนำค่าการตั้งค่านั้นมากำหนดให้บัส SPI เริ่มทำงานตามค่านั้นดังรูปแบบการใช้งานต่อไปนี้

```
SPI.beginTransaction( SPISettings( ... ) )
```

คุณสมบัติของโหมดข้อมูลเป็นตามตารางที่ 14-1 ทำให้เห็นว่าการกำหนดที่แตกต่างกันมีผลต่อเรื่อง Timing Diagram ของ Output Edge และ Data Capture

การสั่งสิ้นสุดรายการสื่อสารใช้คำสั่ง `endTransaction()` ดังนี้ ซึ่งมักเรียกหลังจากสิ้นสุดการเลือกให้อุปกรณ์ปลายทางนั้นทำงาน

```
SPI.endTransaction()
```

ตารางที่ 14-1 การกำหนดโหมดของ SPI

SPI mode	Clock polarity (CPOL/CKP)	Clock phase (CPHA)	Clock edge (CKE/NCPJA)	Output Edge	Data Capture
0	0	0	1	Falling ช่วงเปลี่ยนจาก 1 มาเป็น 0	Rising ช่วงเปลี่ยนจาก 0 มาเป็น 1
1	0	1	0	Rising ช่วงเปลี่ยนจาก 0 มาเป็น 1	Falling ช่วงเปลี่ยนจาก 1 มาเป็น 0
2	1	0	1	Rising ช่วงเปลี่ยนจาก 0 มาเป็น 1	Falling ช่วงเปลี่ยนจาก 1 มาเป็น 0
3	1	1	0	Falling ช่วงเปลี่ยนจาก 1 มาเป็น 0	Rising ช่วงเปลี่ยนจาก 0 มาเป็น 1

ที่มา: ดัดแปลงจาก Espressif Systems. (2024). *ESP32 Technical Reference Manual*. น. 359

สืบค้นเมื่อ เมษายน 28, 2567, จาก

https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf

เมื่อกำหนดค่าเริ่มต้นการสื่อสาร ขั้นตอนต่อไปคือการส่งเพื่ออ่านค่าหรือส่งค่ากับอุปกรณ์ปลายทาง ด้วยคำสั่ง `transfer()` ด้วยหลักการส่งและรอรับข้อมูล ตามรูปแบบต่าง ๆ ต่อไปนี้ ซึ่งสามารถส่งข้อมูลขนาด 1 ไบต์หรือ 2 ไบต์ได้

```
ค่าที่อ่านได้ = SPI.transfer( ค่าที่ส่ง )
ค่าที่อ่านได้16 = SPI.transfer16( ค่าที่ส่ง16 )
SPI.transferr( บัฟเฟอร์, ขนาดของบัฟเฟอร์ )
```

นอกจากตั้งค่าของการสื่อสารผ่านทาง `SPISettings()` แล้ว ผู้เขียนโปรแกรมสามารถตั้งค่าจากฟังก์ชัน `setBitOrder`, `setClockDivider` และ `setDataMode` ได้ตามรูปแบบต่อไปนี้

```
SPI.setBitOrder( การเรียงลำดับข้อมูล )
SPI.setClockDivider( ค่าตัวหารสัญญาณนาฬิกา )
SPI.setDataMode( โหมดข้อมูล )
```


ชื่อของค่าคงที่สำหรับใช้แทนตัวเลขสำหรับเป็นตัวหารความถี่สัญญาณนาฬิกาด้วย 2, 4, 8, 16, 32, 64 และ 128 สำหรับการสื่อสารแบบ SPI ได้แก่ค่าต่อไปนี้

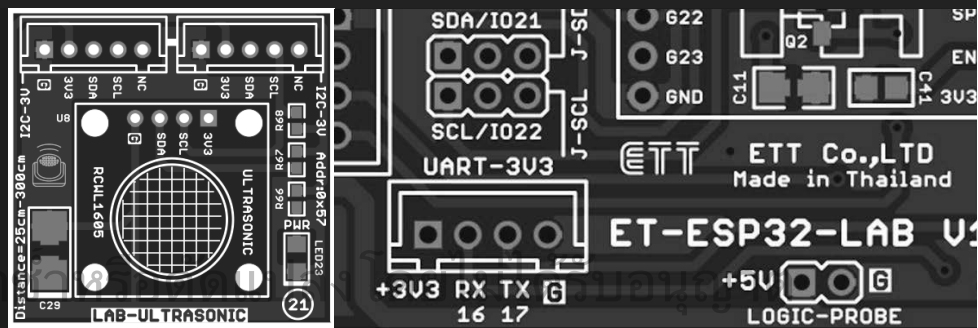
- SPI_CLOCK_DIV2
- SPI_CLOCK_DIV4
- SPI_CLOCK_DIV8
- SPI_CLOCK_DIV16
- SPI_CLOCK_DIV32
- SPI_CLOCK_DIV64
- SPI_CLOCK_DIV128

> สรุป

จากบทนี้จะพบว่าทั้งบัสแบบ I²C และ SPI เน้นการสื่อสารระหว่างอุปกรณ์บนบัส โดยจำนวนสายสัญญาณของทั้ง 2 ประเภทมีความแตกต่างกัน และด้วยการที่ SPI รองรับการมีสายสัญญาณที่แยกระหว่างการส่งและการรับจึงทำให้มีการส่งข้อมูลได้รวดเร็วกว่า แต่ต้องแลกด้วยจำนวนขาที่ต้องถูกใช้มากกว่า และต้องมีขาสำหรับทำหน้าที่ CS หรือเลือกอุปกรณ์ใดให้ทำงาน ขณะที่ I²C ใช้เพียง SDA และ SCL พร้อมทั้งใช้ค่าตำแหน่งของอุปกรณ์เป็นตัวกำหนดการเลือกให้อุปกรณ์ใดทำงานจึงประหยัดจำนวนขามากกว่าแบบ SPI แต่ข้อเสียกลับเป็นเรื่องของความเร็วในการรับส่งที่น้อยกว่าด้วยเช่นกัน

บรรณานุกรม

- Espressif System. (2024). *ESP32 Series Datasheet*. สืบค้นเมื่อ เมษายน 28, 2567, จาก https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- Espressif Systems. (2024). *ESP32 Technical Reference Manual*. สืบค้นเมื่อ เมษายน 28, 2567, จาก https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf



บทที่ 15 LAB-ULTRASONIC

ในบทนี้กล่าวถึงการอ่านสัญญาณดิจิทัลจากเซ็นเซอร์วัดระยะทางด้วยคลื่นเสียงอัลตราโซนิก (Ultrasonics Sensor) ด้วยโมดูล LAB-ULTRASONIC บนบอร์ด ET-ESP32 LAB V1 ที่เชื่อมต่อแบบบัส I²C แรงดัน 3.3V อันเหมาะกับการนำมาประยุกต์ใช้ตรวจสอบระยะทางระหว่างตำแหน่งที่เซ็นเซอร์ติดตั้งกับสิ่งกีดขวางที่ใกล้ที่สุดโดยในบทนี้ได้กล่าวถึงหลักการทำงานของเซ็นเซอร์วัดระยะทาง วงจรการเชื่อมต่อกับโมดูล ESP32 และตัวอย่างการเขียนโปรแกรมเพื่ออ่านค่ามาแสดงผล โดยแปลงค่าที่อ่านเป็นระยะทางในหน่วยเซนติเมตร และสามารถประยุกต์ใช้งานต่อไป เช่น การเตือนการชน การหาวัตถุที่อยู่ใกล้ที่สุด เป็นต้น

> แนะนำ เซ็นเซอร์อัลตราโซนิก

เซ็นเซอร์อัลตราโซนิก เป็นโมดูลเซ็นเซอร์สำหรับวัดระยะทางดังภาพที่ 15-1 โดยใช้คลื่นเสียงที่เรียกว่าอัลตราโซนิก โดยหลักการทำงานของเซ็นเซอร์ประเภทนี้คือ ตัวส่งสัญญาณหรือที่เรียกว่าทริกเกอร์ (Trigger) ทำการส่งคลื่นเสียงออกไป จนเมื่อพบวัตถุ มีผลทำให้คลื่นเสียงสะท้อนกลับมายังภาครับที่เรียกว่าเอคโค (Echo) ดังภาพที่ 15-2 โดยวิธีการคำนวณระยะทางเป็นการใช้ค่าเวลาที่ถูกใช้ไปตั้งแต่เริ่มส่งคลื่นออกทางภาคส่งหรือทริกเกอร์และได้รับคลื่นกลับมาที่ภาครับหรือเอคโค ดังนั้น สามารถเขียนเป็นกราฟของระดับสัญญาณดิจิทัลที่เกิดกับขาทริกเกอร์ และขาเอคโค ได้ดังภาพที่ 15-3

โมดูล LAB-ULTRASONIC เลือกใช้ชุดเซ็นเซอร์อัลตราโซนิกขนาดเล็กรุ่น RCWL-1605 อันมีคุณสมบัติเป็นดังนี้

1. ระยะทำงานใกล้สุด 2.5 เซนติเมตร และไกลสุด 400 เซนติเมตร
2. ต้องการแรงดันไฟฟ้ากระแสตรง 3.3V และต้องการกระแส 2mA ในการทำงาน
3. มีขาเชื่อมต่อกับอุปกรณ์ไมโครคอนโทรลเลอร์ ESP32 ผ่านทางบัส I²C แบบแรงดัน 3V3
4. ทำงานด้วยคลื่นความถี่ 40kHz
5. ความแม่นยำ $\pm 2\%$
6. ความละเอียดในการวัด 1 มิลลิเมตร
7. ทำงานในอุณหภูมิ -10°C ถึง 60°C

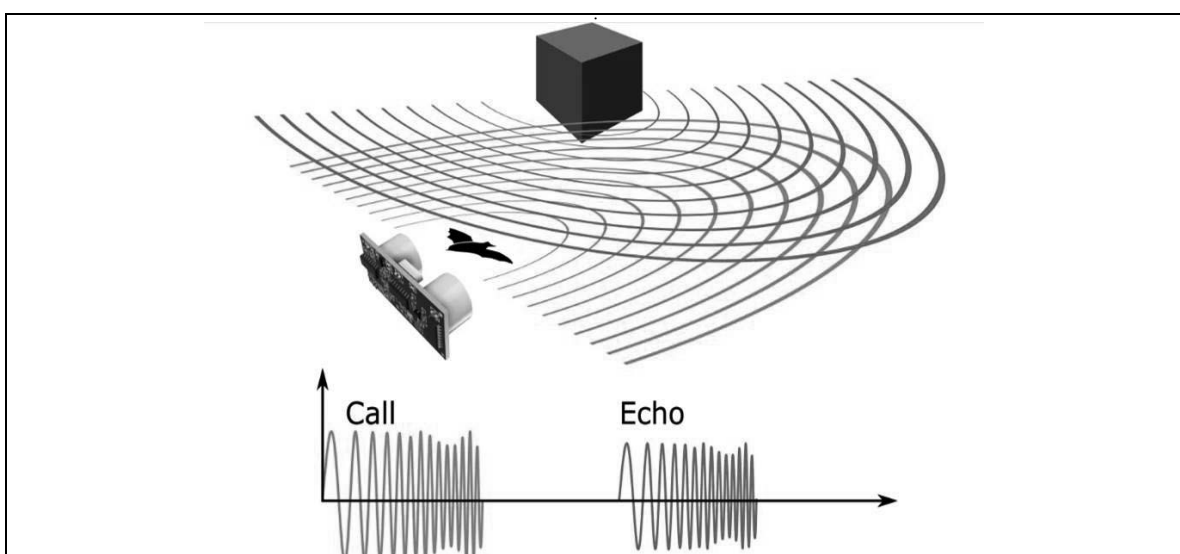
สงวนลิขสิทธิ์ ห้ามนำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



ภาพที่ 15-1

ภาพโมดูลอัลตราโซนิกของ ET-ESP32 LAB V1

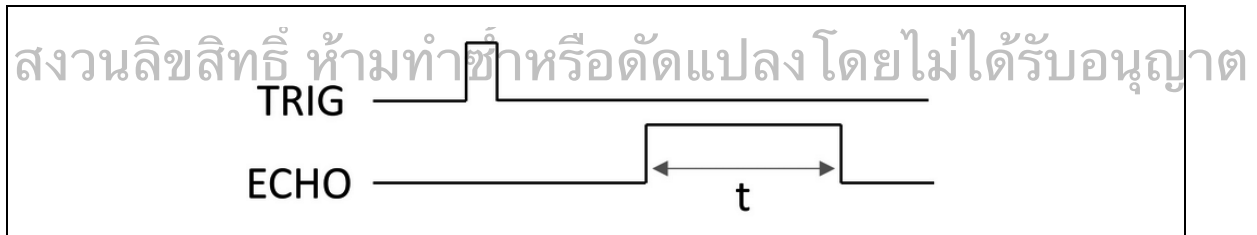
ภาพโดย จารุต บุศราทิจ



ภาพที่ 15-2

หลักการการทำงานของเซ็นเซอร์อัลตราโซนิก

ที่มา Wikipedia, 2024



ภาพที่ 15-3

สัญญาณดิจิทัลที่ขาทรiggerและเอคโคของโมดูลอัลตราโซนิก

ภาพโดย จารุต บุศราทิจ

> วิธีการคำนวณหาระยะ

จากหลักการทำงานของเซ็นเซอร์อัลตราโซนิกจะเห็นได้ว่า การที่เซ็นเซอร์ฝั่งรับรอจนกว่าจะมีคลื่นที่เกิดจากภาคส่งหรือทรiggerส่งออกไปและสะท้อนกลับมานั้นต้องใช้เวลาในการเดินทางของคลื่นเสียงที่มีความเร็วเมื่อเคลื่อนที่ในอากาศเป็น 340 เมตรต่อวินาที ดังสมการที่ 15-1

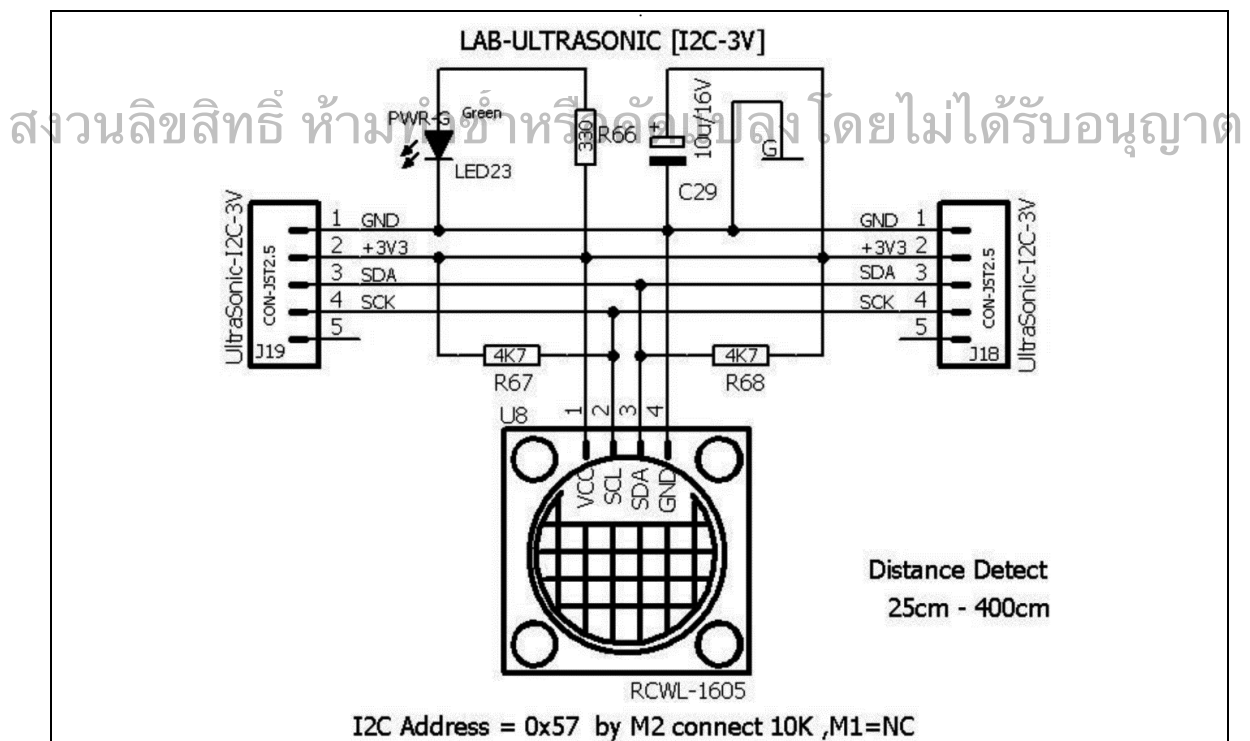
$$\text{ระยะ} = \frac{t}{2} \times 340$$

สมการที่ 15-1

โดย t คือ ค่าเวลาที่เกิดจากการรอตั้งแต่ทรiggerส่งคลื่นเสียงออกไปและเอคโคได้รับเสียงสะท้อนกลับ

> วงจรและการทำงาน

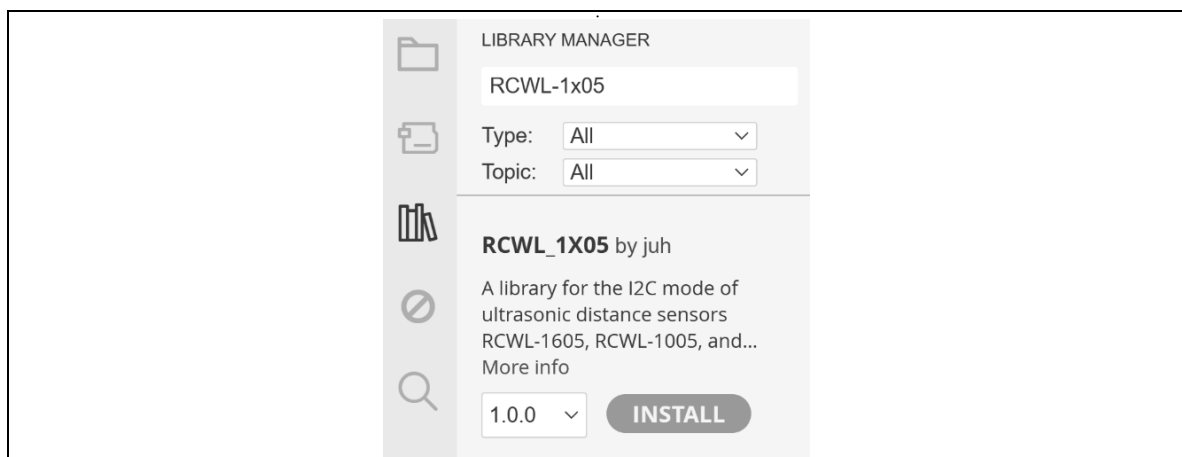
วงจรของโมดูล LAB-ULTRASONIC เป็นดังภาพที่ 15-4 และมีแอลอีดีสีเขียว (LED23) แสดงสถานะการทำงานเมื่อมีการเชื่อมต่อแหล่งจ่ายไฟ พร้อมมี R-Pullup ขนาด 4k7 โอห์มสำหรับขา SDA และ SCL นอกจากนี้ โมดูลเป็นชุดทำงานแบบบัส I²C แบบ 3V3 ทำให้เขียนโปรแกรมมีความสะดวกขึ้นด้วยการเรียกใช้ไลบรารีชื่อ RCWL-1X05 และไม่ต้องเขียนโปรแกรมคำนวณตามสมการ 15-1 โดยค่าตำแหน่งของโมดูลบนบัสคือ 0x57



ภาพที่ 15-4

วงจรของ LAB-ULTRASONIC

ภาพโดย จารุต บุศราทิจ



ภาพที่ 15-5

ไลบรารี RCWL-1X05

ภาพโดย จารุต บุศราทิจ

> ไบรารี RCWL-1X05

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ไลบรารี RCWL-1X05 พัฒนาโดย Juh เมื่อ ค.ศ. 2023 อันต้องติดตั้งเพิ่มเติมจาก Library Manager ดังตัวอย่างในภาพที่ 15-5 และเมื่อติดตั้งไลบรารีเรียบร้อยแล้วให้นำเข้าไฟล์ส่วนหัว 2 ตัว คือ RCWL_1X05.h และ Wire.h ดังนี้

```
#include <RCWL_1X05.h>
#include <Wire.h>
```

การสร้างวัตถุอ้างอิงไปยังเซ็นเซอร์อัลตราโซนิกจะต้องสร้างตัวแปรที่สืบทอดจากคลาส RCWL_1X05 ดังนี้

```
RCWL_1X05 ชื่อวัตถุ;
```

สิ่งที่ต้องระวังคือต้องการกำหนดการตั้งค่าขา SDA และ SCL ให้ถูกต้องในการเรียกการทำงานของคลาส Wire ในฟังก์ชัน Setup()

```
Wire.begin(ขา_SDA, ขา_SCL)
```

การสั่งให้เซ็นเซอร์วัดระยะทางเริ่มต้นทำงานต้องเรียกใช้เมธอด begin() ดังนี้

```
ชื่อวัตถุ.begin()
```

จากรูปแบบของคำสั่งสามารถเขียนโค้ดเพื่อตรวจสอบและรายงานความผิดพลาดจากการไม่พบเซ็นเซอร์บนบัส I²C ได้ดังนี้

```
if (not ชื่อวัตถุ.begin()) {
    Serial.println("Error:RCWL-1605 not found!");
    while(1);
}
```

การกำหนดโหมดทำงานของโมดูล LAB-ULTRASONIC จะต้องใช้คำสั่ง `setMode()` ตามด้วย ชื่อโหมดทำงาน ดังรูปแบบการใช้งานต่อไปนี้

ชื่อวัตถุ.`setMode( ชื่อโหมดทำงาน )`

โดยที่

ชื่อโหมดทำงาน ได้แก่

- `RCWL_1X05::oneShot` เป็นโหมดทำงานปกติของโมดูล หรือมีการส่งสัญญาณและรอรับสัญญาณสะท้อนเพียงครั้งเดียว โดยใช้คำสั่ง `read()`
- `RCWL_1X05::continuous` เป็นโหมดการทำงานแบบต่อเนื่อง หรือ ทำการส่งและรอรับสัญญาณต่อเนื่องไปเรื่อย ๆ โดยใช้คำสั่ง `update()` ก่อนทำการ `read()` ทุกครั้งเพื่อปรับปรุงค่าระยะทาง พร้อมทั้งสามารถกำหนดระยะเวลาเกินเวลาด้วยการตั้งค่าใน `timeout()`
- `RCWL_1X05::triggered` เป็นโหมดที่ต้องส่งเริ่มต้นวัดเองด้วยคำสั่ง `trigger()` และอ่านค่าด้วย `read()` พร้อมทั้งสามารถกำหนดระยะเวลาเกินเวลาด้วยการตั้งค่าใน `timeout()`

การตั้งค่าช่วงเวลาการเกินเวลาหรือ time-out สำหรับป้องกันความผิดพลาดกรณีที่ไม่ได้รับสัญญาณสะท้อนกลับหรือสัญญาณเอคโค ให้ใช้คำสั่ง `setTimeout()` พร้อมระบุจำนวนเวลา ในหน่วยมิลลิวินาที ดังนี้

ชื่อวัตถุ.`setTimeout( จำนวนเวลา )`

กรณีที่ใช้โหมด `triggered` หรือสั่งงานการวัดด้วยตนเองจะต้องเรียก `trigger()` เพื่อเริ่มส่งสัญญาณทรiggerออกจากเซ็นเซอร์ด้วยรูปแบบการใช้งานต่อไปนี้

ชื่อวัตถุ.`trigger()`

โดยปกติแล้วการอ่านค่าจากคำสั่ง `read()` จะได้ข้อมูลที่อ่านได้ขณะนั้น แต่ถ้าผู้เขียนโปรแกรมต้องการค่าเฉลี่ยจากข้อมูล 5 ครั้งสุดท้ายจะต้องกำหนดการทำการกรองให้เป็นจริงด้วยคำสั่ง `setFilter()` และกำหนด สถานะ เป็น `true` แต่ถ้าต้องการรับข้อมูลดิบแบบปัจจุบันจาก `read()` จะต้องกำหนดให้ สถานะเป็น `false` ดังรูปแบบการใช้งานต่อไปนี้

```
ชื่อวัตถุ.setFilter(สถานะ)
```

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

การสั่งให้เซ็นเซอร์ดำเนินการวัดระยะทางใหม่อีกครั้ง ต้องเรียก update() ดังรูปแบบการใช้งานต่อไปนี้

```
ชื่อวัตถุ.update()
```

ค่าระยะทางที่ได้จากการคำนวณแล้วต้องใช้คำสั่ง read() ตามรูปแบบของการใช้งานต่อไปนี้ เพื่อรับค่า ระยะทาง ในหน่วยมิลลิเมตร

```
ระยะทาง=ชื่อวัตถุ.read()
```

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 15-1 เป็นการอ่านข้อมูลระยะทางจากโมดูลอัลตราโซนิกและเล่นเสียงตามระยะห่างของวัตถุ หากวัตถุอยู่ใกล้เสียงจะเล่นถี่มากขึ้น มีการตรวจสอบโมดูลอัลตราโซนิกในบรรทัดที่ 14-16 ตั้งค่าโมดูลอัลตราโซนิกในบรรทัดที่ 20-22 อัปเดตค่าของโมดูลอัลตราโซนิกและนำมาเล่นเสียงในบรรทัดที่ 29-35 โดยต่อขา 12 กับโมดูลเสียง และต่อขา 21 และ 22 กับ SDA, SCL ของโมดูลอัลตราโซนิกตามลำดับ

ตัวอย่างโปรแกรม 15-1 อ่านข้อมูลระยะทางจากโมดูลอัลตราโซนิก

บรรทัด	คำสั่ง
1	#include <RCWL_1X05.h>
2	#include <ToneESP32.h>
3	#define PiezoPin 12
4	#define PiezoCH 0
5	RCWL_1X05 ultrasonic1;
6	ToneESP32 piezo(PiezoPin ,PiezoCH);
7	void beep(int note, int duration){

ตัวอย่างโปรแกรม 15-1 อ่านข้อมูลระยะทางจากโมดูลอัลตราโซนิก (ต่อ)

บรรทัด	คำสั่ง
8	piezo.tone(note, duration);
9	delay(duration * 0.25);
10	}
11	void setup(){
12	Serial.begin(115200);
13	Wire.begin();
14	if(not ultrasonic1.begin()){
15	while (true);
16	} else {
17	delay(1000);
18	}
19	ultrasonic1.setFilter(true);
20	ultrasonic1.setTimeout(50);
21	ultrasonic1.setMode(RCWL_1X05::continuous);
22	delay(2000);
23	}
24	void loop(){
25	delay(50);
26	float cm;
27	bool data ;
28	data = ultrasonic1.update();
29	if(data){
30	cm = ultrasonic1.read();
31	cm = cm/10;
32	Serial.println(cm);
33	beep(NOTE_G5, 240);
34	delay(cm*3);
35	}
36	}

ตัวอย่างโปรแกรม 15-2 เป็นตัวอย่างการอ่านข้อมูลระยะทางจากโมดูลอัลตราโซนิกและแสดงหรือไม่แสดงข้อมูลระยะทางบนจอ OLED ด้วยการกดปุ่ม โดยตั้งค่าโมดูล OLED และ โมดูลอัลตราโซนิก ในบรรทัดที่ 12-35 ตรวจสอบสถานะของปุ่มกดในบรรทัดที่ 44-46 และตรวจสอบสถานะของการแสดง

หรือไม่แสดงและแสดงผลในบรรทัดที่ 48-64 การต่อขาคือ ขา 21 และ 22 ต่อกับ SDA, SCL ของโมดูล OLED และโมดูลอัลตราโซนิก ตามลำดับ และต่อขา 16 กับปุ่มกด

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 15-2 อ่านข้อมูลระยะทางจากโมดูลอัลตราโซนิก

บรรทัด	คำสั่ง
1	#include <RCWL_1X05.h>
2	#include <Adafruit_GFX.h>
3	#include <Adafruit_SH1106.h>
4	#define SDA 21
5	#define SCL 22
6	#define button1 16
7	bool status = true;
8	Adafruit_SH1106 display(SDA, SCL);
9	RCWL_1X05 ultrasonic1;
10	void setup(){
11	pinMode(button1, INPUT);
12	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
13	display.clearDisplay();
14	display.display();
15	delay(100);
16	display.setTextSize(1);
17	display.setTextColor(WHITE);
18	display.setCursor(0,0);
19	Wire.begin();
20	if(not ultrasonic1.begin()){
21	display.println("Sensor not found!");
22	display.display();
23	while (true) {}
24	}
25	else{
26	display.println();
27	display.println("Sensor ready!");
28	display.println();
29	delay(1000);
30	}
31	display.println("Display in Continuous mode\n\n");

ตัวอย่างโปรแกรม 15-2 อ่านข้อมูลระยะทางจากโมดูลอัลตราโซนิก (ต่อ)

บรรทัด	คำสั่ง
32	ultrasonic1.setFilter(true);
33	ultrasonic1.setTimeout(50);
34	ultrasonic1.setMode(RCWL_1X05::continuous);
35	display.display();
36	delay(2000);
37	}
38	void loop(){
39	float cm ;
40	bool data ;
41	int press;
42	press = digitalRead(button1);
43	if(press == LOW){
44	status = !status;
45	delay(300);
46	}
47	if(status){
48	data = ultrasonic1.update();
49	if(data){
50	cm = ultrasonic1.read();
51	cm = cm/10;
52	display.clearDisplay();
53	display.setCursor(0,25);
54	display.print("Distance = ");
55	display.print(cm,1);
56	display.println(" cm.");
57	display.display();
58	}
59	}else{
60	display.clearDisplay();
61	display.setCursor(0,25);
62	display.println("Off");
63	display.display();
64	}
65	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ในบทนี้ได้เรียนรู้ถึงคุณสมบัติ หลักการทำงาน และวิธีการคำนวณระยะทางด้วยหลักของการสะท้อนของเสียงของเซ็นเซอร์วัดระยะทางด้วยอัลตราโซนิก และได้ประยุกต์เขียนโปรแกรมเพื่อสั่งเริ่มต้นและคำนวณหาระยะทางระหว่างเซ็นเซอร์กับวัตถุที่ใกล้ที่สุดที่อยู่ตรงหน้าของวัตถุ ซึ่งสามารถนำไปประยุกต์ใช้เพื่อเตือนการชนสิ่งกีดขวาง หรือทำตัวเมตโรอิเล็กทรอนิกส์ เป็นต้น

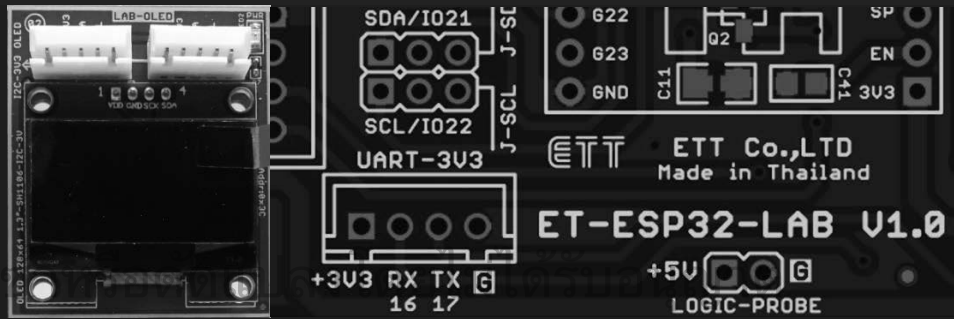
บรรณานุกรม

Juh. (2023). *RCWL_1X05*. สืบค้นเมื่อ เมษายน 30, 2567, จาก

https://github.com/ftjuh/RCWL_1X05

Wikipedia. (2024). *Animal echolocation*. สืบค้นเมื่อ เมษายน 30, 2567, จาก

https://en.wikipedia.org/wiki/Animal_echolocation



บทที่ 16 LAB-OLED

โอแอลอีดี (OLED) หรือ Organic Light-emitting diode เป็นจอแสดงผลด้วยไดโอดเปล่งแสงแบบฟิล์มสารประกอบแบบอินทรีย์ที่ปล่อยแสงตอบสนองต่อกระแสไฟฟ้า และถูกประยุกต์ใช้สร้างจอแสดงผลสำหรับคอมพิวเตอร์ จอโทรทัศน์ และจอร์บบเครื่องเล่นเกมแบบพกพา หรือจอสมาร์ทโฟน เป็นต้น

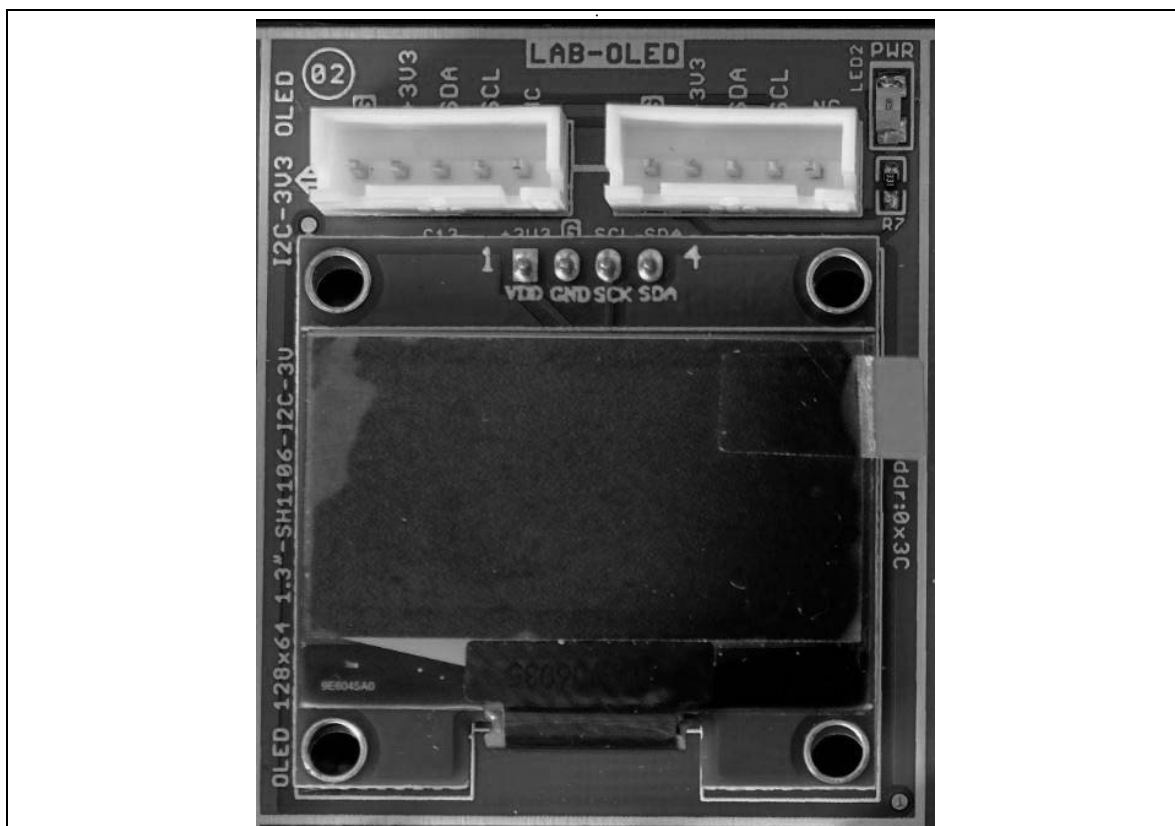
LAB-OLED เป็นโมดูลแสดงผลแบบกราฟิกประเภท OLED ที่ควบคุมผ่านทางบัสแบบ I²C แบบ 3V3 โดยบทนี้อธิบายถึงหลักการทำงานของวงจรเชื่อมต่อของโมดูลและการเชื่อมต่อกับบอร์ดไมโครคอนโทรลเลอร์ ESP32 เพื่อเขียนโปรแกรมสั่งงาน พร้อมตัวอย่างโปรแกรมสำหรับควบคุมด้วยไลบรารี SH1106 ของ Adafruit

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรการทำงานของ LAB-OLED และการใช้งานไลบรารี SH1106 ในการควบคุมการแสดงผลด้วยไลบรารีเพื่อแสดงตัวอักษร การลงพิกเซล (Pixel) การวาดกล่อง เส้นตรง การลงสีและการควบคุมการแสดงผลของโมดูล OLED เป็นต้น

> แนะนำ LAB-OLED

ภาค LAB-OLED เป็นโมดูลสำหรับทดลองการนำออกข้อมูลเพื่อแสดงผลที่ OLED หรือจอแสดงผลกราฟิกความละเอียด 128x64 จุด ที่ใช้หน่วยควบคุมการแสดงผลด้วยไอซี SH1106 ดังภาพที่ 16-1 โดยขาของโมดูล LAB-OLED ประกอบด้วย 4 ขา คือ ขากราวด์ ขาแหล่งจ่ายแรงดันไฟฟ้ากระแสตรงที่รองรับแรงดัน +3V3 และขาสื่อสารที่ทำหน้าที่ SDL และ SDA ตามที่ได้กล่าวไว้ในบทที่ 14

ข้อดีของโมดูล LAB-OLED คือ มีขาสำหรับเชื่อมต่อให้ 2 ชุด ทำให้ผู้ใช้งานสามารถเชื่อมต่อเข้ากับโมดูลอื่นที่เป็นบัส I²C แบบ 3V3 ได้ เช่น ใช้โมดูล OLED กับโมดูลอ่านค่าอุณหภูมิและความชื้น



ภาพที่ 16-1

โมดูล LAB-OLED

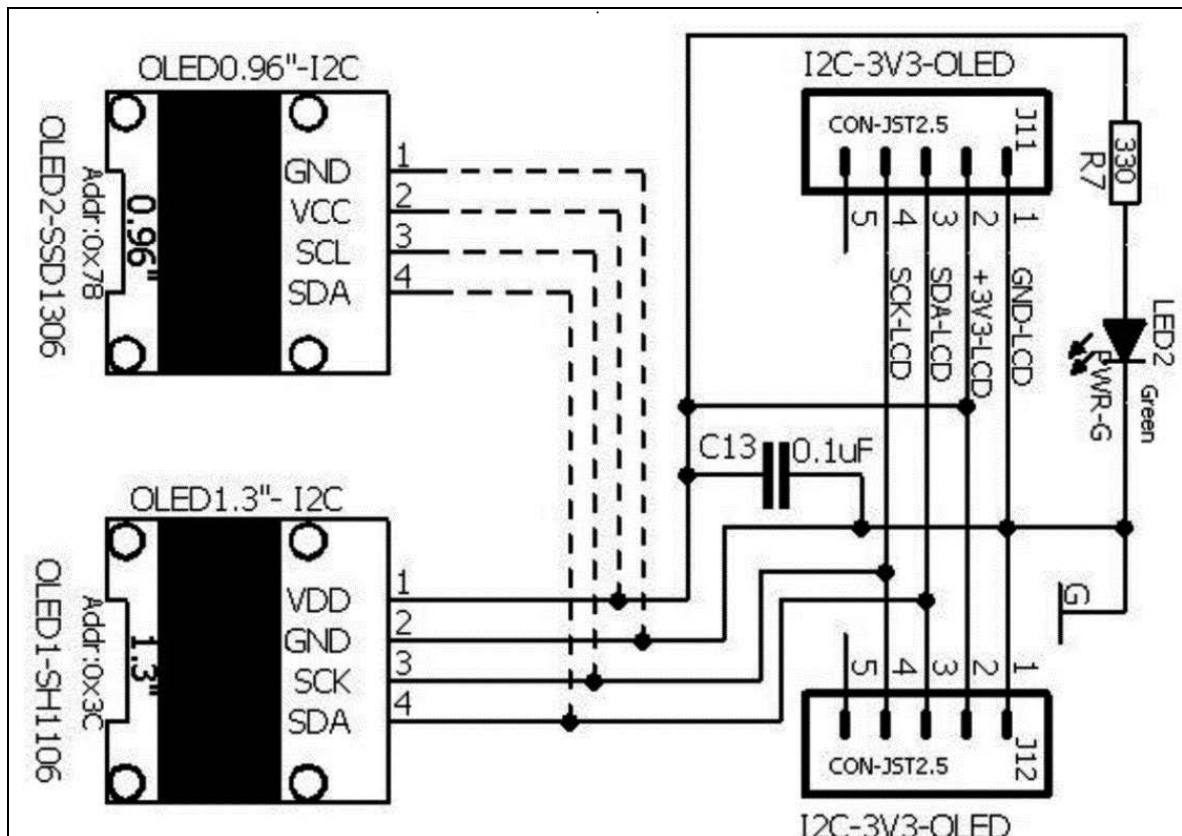
ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

วงจรของ LAB-OLED เป็นดังภาพที่ 16-2 ซึ่งจะพบว่าเป็นการเชื่อมต่อเข้ากับบัส I²C แบบ 3V3 2 ชุดใช้สำหรับเชื่อมต่อกับไมโครคอนโทรลเลอร์ ESP32 มีตัวเก็บประจุ C13 สำหรับบัพเฟอร์แรงดันของแหล่งจ่ายไฟที่มาจากขา +3V3 พร้อมทั้งมี LED2 แบบสีเขียวสำหรับแสดงสถานะของการจ่ายกระแสเข้าบอร์ด

ตัวควบคุม OLED ที่ติดตั้งมากับโมดูลทดลองเป็นรุ่น SH1106 ซึ่งมีคุณสมบัติแตกต่างจาก DS1306 เช่น การวางตำแหน่งของขาที่เรียงไม่เหมือนกัน ขุดคำสั่งไม่เหมือนกัน เป็นต้น ดังนั้น ผู้อ่านต้องตรวจสอบรุ่นของโมดูลให้ติก่อนนำผังวงจรหรือโปรแกรมตัวอย่างไปใช้งาน

ค่าตำแหน่งของอุปกรณ์ OLED บนบัส I²C คือ 0x3C และการใช้งานต้องเชื่อมประสานขา P21 ของ ESP32 เข้ากับขา SDA ของขั้วต่อบัส I²C และขา P22 เข้ากับ SCL



ภาพที่ 16-2

วงจรของ LAB-OLED

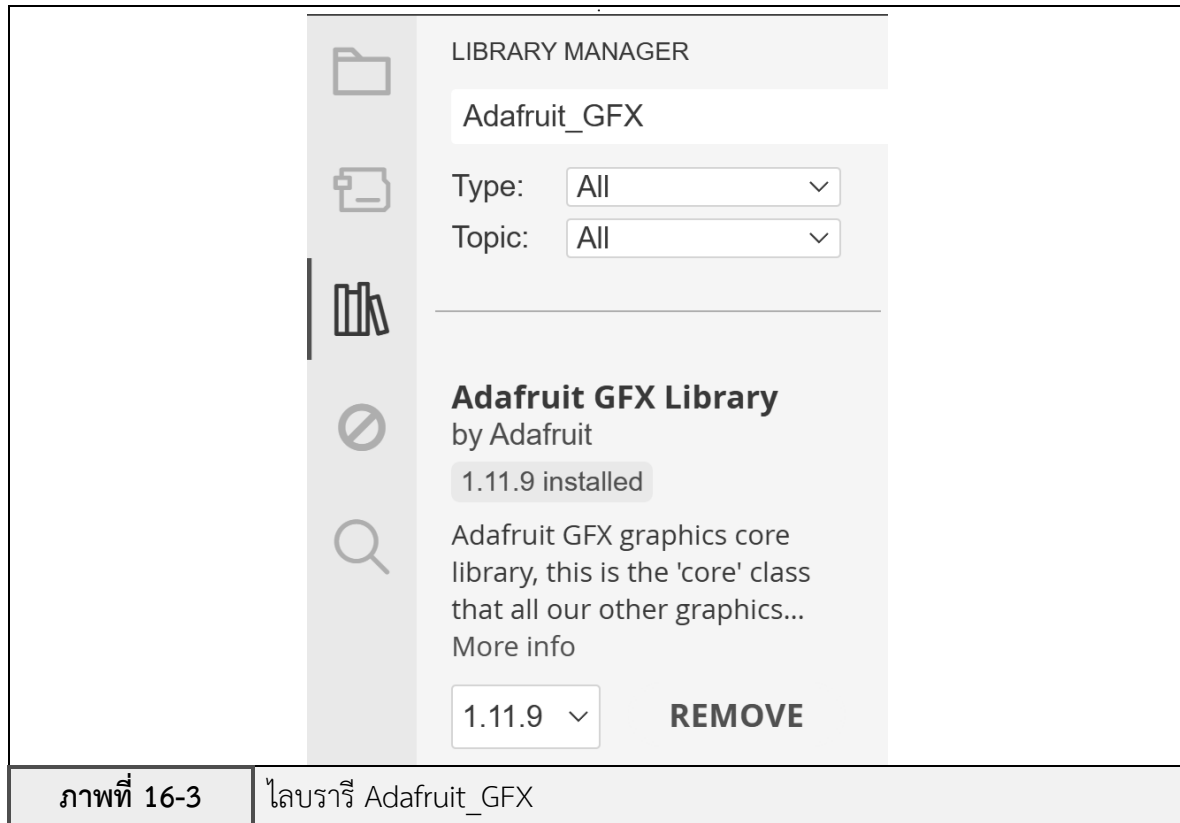
ภาพโดย จารุต บุศราทิจ

> ไลบรารี Adafruit_GFX.h

ไลบรารี Adafruit_GFX เป็นแกนหลักของไลบรารีกราฟิกที่พัฒนาโดยบริษัท Adafruit ที่ออกแบบมาเพื่อบริหารจัดการบัพเฟอร์กราฟิกส์ โดยเมื่อต้องการใช้ในการแสดงผลบนอุปกรณ์แสดงผล ต้องใช้ควบคู่กับไดรเวอร์ของหน่วยแสดงผลตามอุปกรณ์ที่ใช้งาน เช่น SH1106, SSD1306 เป็นต้น ทำให้

อุปกรณ์แสดงผลแบบกราฟฟิกส์ที่มีไดรเวอร์สนับสนุนการทำงานสามารถใช้ชุดคำสั่งของ GFX ร่วมกันได้
จึงช่วยลดระยะเวลาในการเรียนรู้ชุดคำสั่งใหม่ทุกครั้งเมื่อเปลี่ยนอุปกรณ์แสดงผล
การใช้งานไลบรารี Adafruit_GFX ต้องติดตั้งไลบรารีเพิ่มเติมดังตัวอย่างในภาพที่ 16-3 และต้อง
เรียกใช้ไลบรารีดังนี้

```
#include <Adafruit_GFX.h>
```



ภาพที่ 16-3

ไลบรารี Adafruit_GFX

ภาพโดย จารุต บุศราทิจ

หลังจากนั้นสร้างตัวแปรเพื่อใช้อ้างอิงไลบรารี Adafruit_GFX พร้อมกำหนดค่าความกว้างและ
ความสูงของจอแสดงผลดังนี้

Adafruit_GFX ตัวแปรอ้างอิง(ความกว้าง, ความสูง)

คำสั่งลงจุดสี ณ ตำแหน่ง (x, y) คือ drawPixel() อันมีรูปแบบของการใช้งานดังนี้

ตัวแปรอ้างอิง.drawPixel(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, ค่าสี)

สำหรับคำสั่งการวาดแบบมีการสั่งเริ่มต้นการวาด ตามด้วยชุดคำสั่งการวาด และปิดท้ายการสั่งหยุดวาด หรือเรียกว่าชุดคำสั่งแบบรายการ (Transaction API) สามารถกระทำด้วยคำสั่ง `startWrite()` สำหรับเริ่มต้นการวาด, `writePixel()` สำหรับลงจุดสี, `writeFillRect()` สำหรับวาดสี่เหลี่ยมทึบ, `writeFastVLine()` สำหรับการวาดเส้นแนวตั้ง, `writeFastHLine()` สำหรับวาดเส้นแนวนอน, `writeLine()` สำหรับวาดเส้นตรงจาก (x0, y0) ไปยัง (x1, y1) และปิดท้ายคำสั่งด้วย `endWrite()` มีรูปแบบของแต่ละคำสั่งดังนี้

ตัวแปรอ้างอิง.`startWrite()`

ตัวแปรอ้างอิง. `writePixel()` (ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, ค่าสี)

ตัวแปรอ้างอิง.`writeFillRect()` (ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความกว้าง, ค่าความสูง, ค่าสี)

ตัวแปรอ้างอิง.`writeFastVLine()` (ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความสูงของเส้น, ค่าสี)

ตัวแปรอ้างอิง.`writeFastHLine()` (ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความยาวของเส้น, ค่าสี)

ตัวแปรอ้างอิง.`writeLine()` (ค่าตำแหน่งในแกนX0, ค่าตำแหน่งในแกนY0,
ค่าตำแหน่งในแกนX1, ค่าตำแหน่งในแกนY1, ค่าสี)

ตัวแปรอ้างอิง.`endWrite()`

คำสั่งสำหรับควบคุมการแสดงผล ได้แก่ การหมุนจอแสดงผลตามมุมมอง 0, 90, 180 และ 270 องศา ต้องใช้คำสั่ง `setRotation()` ตามด้วยค่า 0, 1, 2 และ 3 สำหรับค่าหมุนมุมทั้ง 4 มุม และการสั่งกลับค่าการแสดงผลกระทำด้วยการกำหนดค่า `true` หรือ `false` เป็นพารามิเตอร์ของคำสั่ง `invertDisplay()` ดังรูปแบบการใช้งานต่อไปนี้

ตัวแปรอ้างอิง.`setRotation()` (ค่ามุมของการหมุน)

ตัวแปรอ้างอิง.`invertDisplay()` (สถานะการกลับค่าแสดงผล)

คำสั่งสำหรับการเปลี่ยนสีของจอแสดงผล คือ `fillScreen()` มีรูปแบบการใช้งานดังนี้

ตัวแปรอ้างอิง.`fillScreen()` (ค่าสี)

คำสั่งสำหรับการวาดเส้นตรงแนวตั้ง เส้นตรงแนวนอน และเส้นตรง คือ คำสั่ง drawFastVLine(), drawFastHLine() และ drawLine() ดังรูปแบบของคำสั่งใช้งานต่อไปนี้

ตัวแปรอ้างอิง.drawFastVLine(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความสูงของเส้น, ค่าสี)

ตัวแปรอ้างอิง.drawFastHLine(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความยาวของเส้น, ค่าสี)

ตัวแปรอ้างอิง.drawLine(ค่าตำแหน่งในแกนX0, ค่าตำแหน่งในแกนY0,
ค่าตำแหน่งในแกนX1, ค่าตำแหน่งในแกนY1, ค่าสี)

ชุดคำสั่งสำหรับวาดสี่เหลี่ยม สี่เหลี่ยมทึบ วงกลม วงกลมทึบ สามเหลี่ยม สามเหลี่ยมทึบ สี่เหลี่ยมมุมมน และสี่เหลี่ยมมุมมนแบบทึบ ได้แก่ คำสั่ง drawRect(), fillRect(), drawCircle(), fillCircle(), drawTriangle(), fillTriangle(), drawRoundRect() และ fillRoundRect()

ตัวแปรอ้างอิง.drawRect(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความกว้าง, ค่าความสูง, ค่าสี)

ตัวแปรอ้างอิง.fillRect(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความกว้าง, ค่าความสูง, ค่าสี)

ตัวแปรอ้างอิง.drawCircle(ค่าจุดกึ่งกลางในแกนX, ค่าจุดกึ่งกลางในแกนY,
ค่ารัศมี, ค่าสี)

ตัวแปรอ้างอิง.fillCircle(ค่ากึ่งกลางในแกนX, ค่ากึ่งกลางในแกนY,
ค่ารัศมี, ค่าสี)

ตัวแปรอ้างอิง.drawTriangle(ค่าตำแหน่งX0, ค่าตำแหน่งY0,
ค่าตำแหน่งX1, ค่าตำแหน่งY1,
ค่าตำแหน่งX2, ค่าตำแหน่งY2, ค่าสี)

ตัวแปรอ้างอิง.fillTriangle(ค่าตำแหน่งX0, ค่าตำแหน่งY0,
ค่าตำแหน่งX1, ค่าตำแหน่งY1,
ค่าตำแหน่งX2, ค่าตำแหน่งY2, ค่าสี)

ตัวแปรอ้างอิง.drawRoundRect(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความกว้าง, ค่าความสูง, ค่ามุม, ค่าสี)

ตัวแปรอ้างอิง.fillRoundRect(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY,
ค่าความกว้าง, ค่าความสูง, ค่ามุม, ค่าสี)

การวาดบิตแมพเป็นการนำข้อมูลจากหน่วยความจำที่เก็บค่าของเม็ดสีของภาพมาใส่ลงบัฟเฟอร์ของจอแสดงผล โดยคำสั่งในกลุ่มนี้ประกอบด้วย drawBitmap(), drawXBitmap(), drawGrayscaleBitmap() และ drawRGBBitmap() ดังรูปแบบการใช้งานต่อไปนี้

ตัวแปรอ้างอิง.drawBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, ค่าความกว้างภาพ, ภาพความสูงภาพ, ค่าสี)

ตัวแปรอ้างอิง.drawBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, ค่าความกว้างภาพ, ภาพความสูงภาพ, ค่าสี, ค่าสีพื้นหลัง)

ตัวแปรอ้างอิง.drawXBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, ค่าความกว้างภาพ, ภาพความสูงภาพ, ค่าสี)

ตัวแปรอ้างอิง.drawGrayscaleBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, ค่าความกว้างภาพ, ภาพความสูงภาพ)

ตัวแปรอ้างอิง.drawGrayscaleBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, บัฟเฟอร์ภาพหน้ากาก, ค่าความกว้างภาพ, ภาพความสูงภาพ)

ตัวแปรอ้างอิง.drawRGBBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, ค่าความกว้างภาพ, ภาพความสูงภาพ)

ตัวแปรอ้างอิง.drawRGBBitmap(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, บัฟเฟอร์ภาพ, บัฟเฟอร์ภาพหน้ากาก, ค่าความกว้างภาพ, ภาพความสูงภาพ)

การวาดตัวอักษรใช้คำสั่ง drawChar() ตามรูปแบบการใช้งานต่อไปนี้

ตัวแปรอ้างอิง.drawChar(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, ค่าตัวอักษร, ค่าสีตัวอักษร,ค่าสีพื้นหลังตัวอักษร, ค่าขนาดตัวอักษร)

ตัวแปรอ้างอิง.drawChar(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY, ค่าตัวอักษร, ค่าสีตัวอักษร,ค่าสีพื้นหลังตัวอักษร, ค่าขนาดตัวอักษรในแกนX, ค่าขนาดตัวอักษรในแกนY)

กรณีที่ต้องการปรับขนาดของข้อความสามารถใช้คำสั่ง setTextSize() ตามรูปแบบการใช้งานดังนี้

ตัวแปรอ้างอิง.setTextSize(ค่าขนาด)

ตัวแปรอ้างอิง.setTextSize(ค่าขนาดในแกนX, ค่าขนาดในแกนY)

รูปแบบตัวอักษรหรือฟอนต์ (Font) สามารถเลือกได้จากตัวแปรในคลาสหรือไลบรารี GFXFont ด้วยคำสั่งใช้งานต่อไปนี้ โดยถ้าไม่กำหนดค่าฟอนต์ให้กับคำสั่งจะหมายถึงเลือกใช้ฟอนต์ปกติที่มีอยู่

ตัวแปรอ้างอิง.setFont(ประเภทของฟอนต์)

ประเภทของฟอนต์สามารถเลือกใช้ได้จากไลบรารีแต่ต้องนำเข้าไฟล์ส่วนของฟอนต์ต่าง ๆ ที่เลือกใช้ อันได้แก่

FreeMono12pt7b.h	FreeSansBoldOblique12pt7b.h
FreeMono18pt7b.h	FreeSansBoldOblique18pt7b.h
FreeMono24pt7b.h	FreeSansBoldOblique24pt7b.h
FreeMono9pt7b.h	FreeSansBoldOblique9pt7b.h
FreeMonoBold12pt7b.h	FreeSansOblique12pt7b.h
FreeMonoBold18pt7b.h	FreeSansOblique18pt7b.h
FreeMonoBold24pt7b.h	FreeSansOblique24pt7b.h
FreeMonoBold9pt7b.h	FreeSansOblique9pt7b.h
FreeMonoBoldOblique12pt7b.h	FreeSerif12pt7b.h
FreeMonoBoldOblique18pt7b.h	FreeSerif18pt7b.h
FreeMonoBoldOblique24pt7b.h	FreeSerif24pt7b.h
FreeMonoBoldOblique9pt7b.h	FreeSerif9pt7b.h
FreeMonoOblique12pt7b.h	FreeSerifBold12pt7b.h
FreeMonoOblique18pt7b.h	FreeSerifBold18pt7b.h
FreeMonoOblique24pt7b.h	FreeSerifBold24pt7b.h
FreeMonoOblique9pt7b.h	FreeSerifBold9pt7b.h
FreeSans12pt7b.h	FreeSerifBoldItalic12pt7b.h
FreeSans18pt7b.h	FreeSerifBoldItalic18pt7b.h
FreeSans24pt7b.h	FreeSerifBoldItalic24pt7b.h
FreeSans9pt7b.h	FreeSerifBoldItalic9pt7b.h
FreeSansBold12pt7b.h	FreeSerifItalic12pt7b.h
FreeSansBold18pt7b.h	FreeSerifItalic18pt7b.h
FreeSansBold24pt7b.h	FreeSerifItalic24pt7b.h
FreeSansBold9pt7b.h	FreeSerifItalic9pt7b.h

โดยจากรายชื่อไฟล์นำเข้าจะพบว่าประกอบด้วยชื่อฟอนต์ รูปแบบหนา (bold) รูปแบบเอียง (Italic) ขนาดฟอนต์ในหน่วยพอยต์ (pt) และจำนวนตัวอักษร (2^{บิต}) ดังนั้น FreeSansBold24pt7b หมายถึง ฟอนต์ชื่อ FreeSans เป็นฟอนต์ตัวหนา ขนาด 24 พอยต์ และมีตัวอักษรจำนวน 2⁷ หรือ 128 ตัวอักษร และเมื่อนำไปใช้ต้องนำเข้าและเพิ่มโค้ดดังนี้

```
#include <FreeSansBold24pt7b.h>
...
ตัวแปรอ้างอิง.setFont( &FreeSansBold24pt7b)
...
```

การอ่านค่ากรอบรอบข้อความที่วาดสามารถใช้งานคำสั่ง getTextBounds() ตามรูปแบบการใช้งานต่อไปนี้ โดยผลลัพธ์ที่ได้จะถูกเก็บในตัวแปร x1, y1, w และ h อันเป็นค่าพิกัดเริ่มต้นมุมซ้ายบนของข้อความ และความกว้าง/ความสูงของกรอบข้อความตามลำดับ

ตัวแปรอ้างอิง.getTextBounds(ข้อความ, ค่าตำแหน่งในแกนX,
ค่าตำแหน่งในแกนY, x1, y1, w, h)

ถ้าต้องการให้มีการวาดข้อความโดยกำหนดให้มีการขึ้นบรรทัดใหม่ให้กับข้อความที่เกินความกว้างของจอแสดงผลต้องเรียกใช้คำสั่ง setTextWrap() พร้อมกำหนดค่า true หรือ false เพื่อเปิดหรือปิดการทำงานดังรูปแบบของการสั่งต่อไปนี้

ตัวแปรอ้างอิง.setTextWrap(สถานะของการตัดขึ้นบรรทัดใหม่)

การกำหนดตำแหน่งเคอร์เซอร์สำหรับใช้เป็นพิกัดมุมซ้ายบนสำหรับแสดงข้อความสามารถกระทำด้วยการเรียกใช้คำสั่ง setCursor() ดังนี้

ตัวแปรอ้างอิง.setCursor(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY)

การกำหนดสีของตัวอักษรต้องใช้คำสั่ง setTextColor() ดังนี้

ตัวแปรอ้างอิง.setTextColor(ค่าสีตัวอักษร)

ตัวแปรอ้างอิง.setTextColor(ค่าสีตัวอักษร, ค่าสีพื้นหลัง)

เมื่อผู้เขียนโปรแกรมต้องการทราบความกว้างและความสูงของจอแสดงผลสามารถใช้คำสั่ง width() และ height() ดังนี้

ค่าความกว้าง = ตัวแปรอ้างอิง.width()

ค่าความสูง = ตัวแปรอ้างอิง.height()

การอ่านค่าการหมุนของจอแสดงผลต้องใช้คำสั่ง getRotation() เพื่อค่าผลลัพธ์ที่ตอบกลับมาเป็น 0, 1, 2 หรือ 3 ที่เป็นค่าแทนมุมของการหมุนที่ได้กล่าวในขั้นตอนของการสั่งหมุนจอแสดงผล โดยรูปแบบคำสั่งของ getRotation() เป็นดังนี้

ค่าการหมุน = ตัวแปรอ้างอิง.getRotation()

สุดท้าย การอ่านค่าตำแหน่งของเคอร์เซอร์ต้องใช้คำสั่ง getCursorX() และ getCursorY() ตามรูปแบบของคำสั่งทั้ง 2 ดังนี้

ค่าตำแหน่งแกนX = ตัวแปรอ้างอิง.getCursorX()

ค่าตำแหน่งแกนY = ตัวแปรอ้างอิง.getCursorY()

นอกจากนี้ Adafruit_GFX ถูกสืบทอดไปเป็นคลาสใหม่อีก 3 คลาส สำหรับประยุกต์ใช้กับ บัฟเฟอร์แบบ 1, 8 และ 16 บิต คือ GFXCanvas1, GFXCanvas8 และ GFXCanvas16 ทำให้สามารถเลือกสร้างบัฟเฟอร์ที่เหมาะสมกับการใช้งาน โดยบัฟเฟอร์ประเภท 1 บิต หรือ GFXCanvas1 จะเหมาะสมกับเป็นบัฟเฟอร์ของโมดูล LAB-OLED เนื่องจากเป็นจอแสดงผลแบบ 1 บิต หรือ เลือกได้ว่าแสดงจุดสีหรือไม่แสดงจุดสี และคำสั่งที่เพิ่มเติมในคลาสทั้ง 3 คือ คำสั่งอ่านค่าสี ณ ตำแหน่ง (x, y) ดังรูปแบบของการใช้งานต่อไปนี้

ค่าสี = ตัวแปรอ้างอิง.getPixel(ค่าตำแหน่งในแกนX, ค่าตำแหน่งในแกนY)

> ไลบรารี Adafruit_SH1106.h

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ไลบรารี Adafruit_SH1106 เป็นไดรเวอร์ที่พัฒนาต่อยอดจากไดรเวอร์ของ SSD1306 ของบริษัท Adafruit โดย Limor Ladyada Fried ที่มีหน้าที่ควบคุมการทำงานของไอซี SH1106 และไดรเวอร์นี้ไม่รองรับคำสั่งเลื่อนหน้าจอ (Scroll command) เหมือน SSD1306 โดยไลบรารีตัวนี้เป็นสืบทอดมาจากคลาส Adafruit_GFX ทำให้สามารถเรียกใช้งานคำสั่งของ GFX ได้จากวัตถุประเภท Adafruit_SH1106 การใช้งานต้องติดตั้งไลบรารี Adafruit_SH110X ดังภาพที่ 16-4 และต้องเรียกใช้ไลบรารีดังนี้

```
#include <Adafruit_SH110X.h>
```

หลังจากนั้นสร้างตัวแปรเพื่อใช้อ้างอิงอุปกรณ์ประเภท Adafruit_SH1106 ดังนี้

Adafruit_SH1106 ตัวแปรอ้างอิง(หมายเลขเขาSDA, หมายเลขเขาSDL)

ขั้นตอนหลังจากสร้างตัวแปรอ้างอิงไปยังอุปกรณ์ คือ การสั่งให้ไลบรารีเริ่มต้นทำงานด้วยคำสั่ง begin() พร้อมกับการกำหนดพารามิเตอร์ของการเริ่มทำงาน 2 ค่า คือ SH1106_SWITCHCAPVCC และตามด้วยค่าตำแหน่งของจอแสดงผลบนบัส I²C ดังนี้

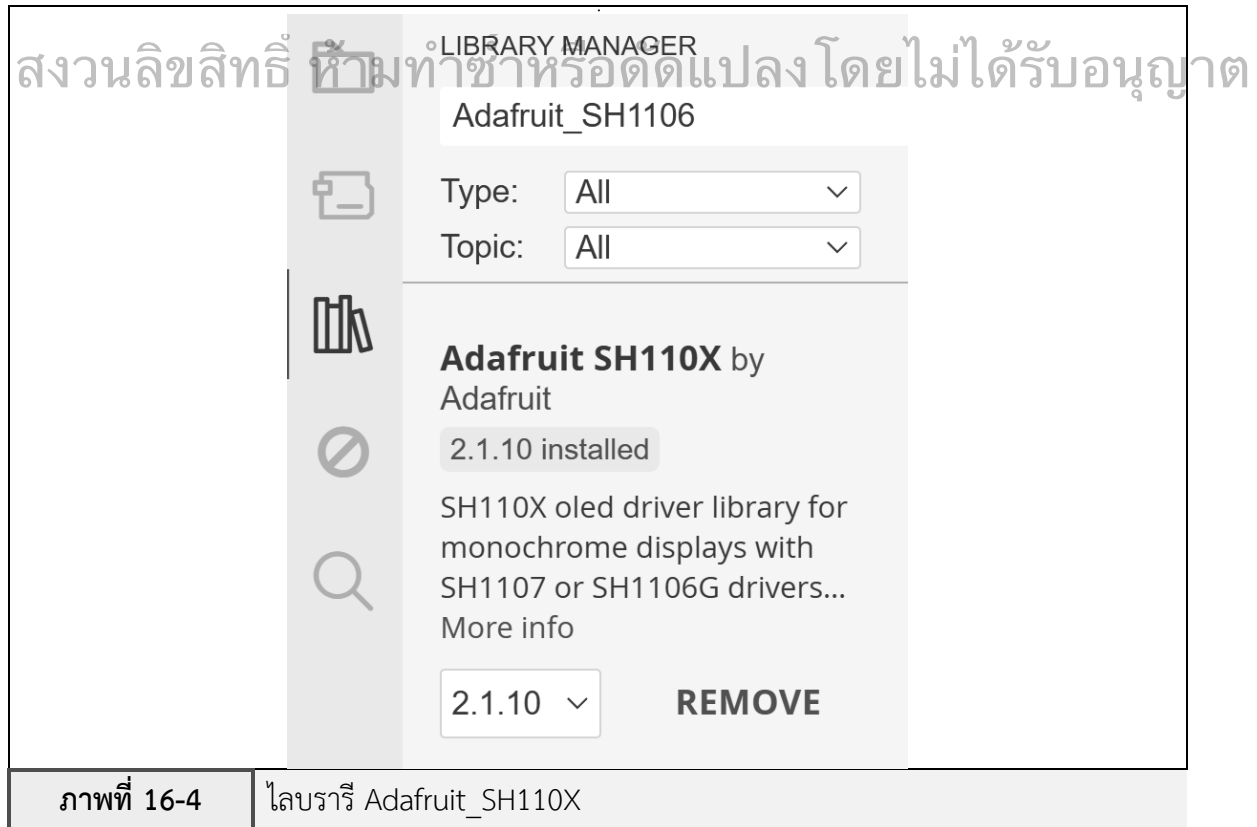
ตัวแปรอ้างอิง.begin(SH1106_SWITCHCAPVCC, 0x3C)

กรณีที่ต้องการสั่งงานจอแสดงผลโดยตรงต้องใช้คำสั่ง SH1106_command() ตามด้วยหมายเลขของคำสั่ง หรือค่าคงที่ของคำสั่งต่อไปนี้

ตัวแปรอ้างอิง.SH1106_command(หมายเลขคำสั่ง)

ความหมายของค่าหมายเลขคำสั่ง คือ

- SH1106_DISPLAYOFF หรือ 0xAE สำหรับสั่งปิดหน้าจอ
- SH1106_DISPLAYON หรือ 0xAF สำหรับสั่งเปิดหน้าจอ



ภาพโดย จารุต บุศราทิจ

แต่ถ้าคำสั่งในกลุ่มที่ต้องส่งพารามิเตอร์จะต้องเรียกใช้คำสั่ง 2 ตัว คือ คำสั่งสำหรับส่งหมายเลขคำสั่ง หรือ SH1106_command() กับคำสั่งส่งพารามิเตอร์ของคำสั่งหรือ SH1106_data() ดังนี้

ตัวแปรอ้างอิง.SH1106_command(หมายเลขคำสั่ง)

ตัวแปรอ้างอิง.SH1106_data(ข้อมูลสำหรับคำสั่ง)

ตัวอย่างค่าหมายเลขคำสั่ง คือ SH1106_SETCONTRAST หรือ 0x81 ตามด้วยส่งข้อมูลแทนค่าความสว่างในช่วง 0-255

การสั่งให้ล้างหน้าจอแสดงผลต้องใช้คำสั่ง clearDisplay() ดังนี้

ตัวแปรอ้างอิง.clearDisplay()

ถ้าต้องการสั่งกลับการแสดงผลจะต้องใช้คำสั่ง `invertDisplay()` ตามด้วยค่า TRUE หรือ FALSE
ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวแปรอ้างอิง `invertDisplay()` (สถานะของการกลับค่า)

และกรณีที่ต้องการแสดงผลต้องเรียกใช้ `display()` เป็นคำสั่งให้นำข้อมูลจากบัฟเฟอร์ของตัวควบคุม OLED ส่งไปยังจอแสดงผล ดังนี้

ตัวแปรอ้างอิง `display()`

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 16-1 เป็นการแสดงข้อความและสีเหลี่ยมผืนผ้าโดยแสดงข้อความ Hello World!! We are ETT!! และวาดสีเหลี่ยมผืนผ้าด้านล่างของหน้าจอโมดูล OLED โดยในตั้งค่าโมดูลในบรรทัดที่ 8-12 แสดงข้อความในบรรทัดที่ 13 และวาดรูปสีเหลี่ยมผืนผ้าในบรรทัดที่ 14-15 มีการต่อขา 21 เข้ากับ SDA และต่อขา 22 เข้ากับ SCL ของโมดูล OLED

ตัวอย่างโปรแกรม 16-1 แสดงข้อความและสีเหลี่ยมผืนผ้า

บรรทัด	คำสั่ง
1	#include <Adafruit_GFX.h>
2	#include <Adafruit_SH1106.h>
3	#define SDA 21
4	#define SCL 22
5	String words = "Hello World! \nWe are ETT!!";
6	Adafruit_SH1106 display(SDA, SCL);
7	void setup(){
8	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
9	display.clearDisplay();
10	display.setTextSize(1);
11	display.setTextColor(WHITE);
12	display.setCursor(0,0);
13	display.println(words);
14	display.drawRect(10, 50, 100, 10, WHITE);
15	display.fillRect(12, 52, 96, 6, WHITE);
16	display.display();
17	}
18	void loop(){
19	}

ตัวอย่างโปรแกรม 16-2 เป็นตัวอย่างแสดงข้อความและแถบความคืบหน้าโดยแสดงข้อความเช่นเดียวกับตัวอย่างโปรแกรม 16-1 พร้อมแสดงแถบความคืบหน้าและตัวอักษรกระพริบด้านล่างของจอโมดูล OLED โดยมีการวนลูปแสดงตัวอักษรด้วยฟังก์ชันวาดตัวอักษรในบรรทัดที่ 22-28 และวาดรูปและปรับปรุงแถบความคืบหน้าในบรรทัดที่ 30-35 การต่อขาเป็นแบบเดียวกับตัวอย่างโปรแกรมที่ 16-1

ตัวอย่างโปรแกรม 16-2 แสดงข้อความและแถบความคืบหน้า

บรรทัด	คำสั่ง
1	#include <Adafruit_GFX.h>
2	#include <Adafruit_SH1106.h>
3	#define SDA 21
4	#define SCL 22
5	int charPos = 10;
6	String words = "Hello World! \nWe are ETT!!";

ตัวอย่างโปรแกรม 16-2 แสดงข้อความและแถบความคืบหน้า (ต่อ)

บรรทัด	คำสั่ง
7	String textToWrite = "Progressing";
8	int progress = 0;
9	bool showText = true;
10	Adafruit_SH1106 dsp(SDA, SCL);
11	void setup(){
12	dsp.begin(SH1106_SWITCHCAPVCC, 0x3C);
13	dsp.clearDisplay();
14	dsp.setTextSize(1);
15	dsp.setTextColor(WHITE);
16	dsp.setCursor(0,0);
17	dsp.display();
18	}
19	void loop() {
20	dsp.clearDisplay();
21	dsp.println(words);
22	if(showText){
23	charPos = 10;
24	for(int i = 0; i < 11; i++){
25	dsp.drawChar(charPos, 40,
26	textToWrite[i], BLACK, WHITE, 1);
27	charPos += 5;
28	}
29	}
30	showText = !showText;
31	dsp.drawRect(10, 50, 100, 10, WHITE);
32	dsp.fillRect(10, 50, progress, 10, WHITE);
33	progress += 10;
34	if(progress > 100){
35	progress = 0;
36	}
37	dsp.setCursor(0,0);
38	dsp.display();
39	delay(1000);
40	}

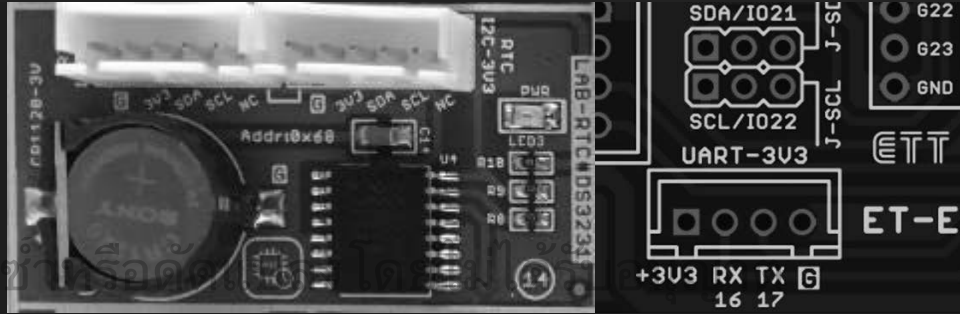
> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-OLED เป็นโมดูลแสดงผล SH1106 ความละเอียด 128x64 จุด อยู่ตำแหน่ง 0x3C ต้องใช้คลาส Adafruit_SH1106 เป็นไดรเวอร์สำหรับควบคุมการแสดงผล และใช้คลาส Adafruit_GFX เป็นคลาสต้นแบบที่ Adafruit_SH1106 สืบทอดมาอีกทีหนึ่ง ทำให้ตัวแปรอ้างอิงที่สร้างขึ้นมีคุณสมบัติในการเรียกใช้คำสั่งแสดงผลของไลบรารี Adafruit_GFX และสามารถนำไปประยุกต์เพื่อวาดหรือแสดงอักษรกับระบบที่พัฒนาขึ้นต่อไป

บรรณานุกรม

- SINO WEALTH. (2015). *SH1106-OLED Datasheet*. สืบค้นเมื่อ พฤษภาคม 2, 2567, จาก <https://www.displayfuture.com/Display/datasheet/controller/SH1106.pdf>
- WIKI PEDIA. (2024). *OLED*. สืบค้นเมื่อ พฤษภาคม 2, 2567, จาก <https://en.wikipedia.org/wiki/OLED>
- Adafruit. (2024). *Adafruit_GFX.h*. สืบค้นเมื่อ พฤษภาคม 2, 2567, จาก <https://learn.adafruit.com/adafruit-gfx-graphics-library/overview>
- Adafruit. (2024). *Adafruit_SH1106*. สืบค้นเมื่อ พฤษภาคม 2, 2567, จาก https://github.com/wonho-maker/Adafruit_SH1106



บทที่ 17 LAB-RTC DS3231

LAB-RTC DS3231 ดังภาพที่ 17-1 เป็นโมดูลนาฬิกาฐานเวลาจริงหรือ RTC (Real Time Clock) ที่ใช้ไอซีเบอร์ DS3231 ของบริษัท Analog Device ทำงานผ่านทางบัส I²C แบบ 3V3 และทำงานได้ถึง ค.ศ. 2100 พร้อมเซ็นเซอร์อุณหภูมิแบบดิจิทัล โดยในบทนี้กล่าวถึงคุณสมบัติ วงจร การใช้งานไลบรารี และตัวอย่างโปรแกรมสำหรับอ่านวัน/เวลา แกะไขวัน/เวลา

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรการทำงานของ LAB-RTC DS3231 และตัวอย่างโปรแกรม ไปประยุกต์ใช้เป็นนาฬิกาของระบบที่นำไปใช้งาน เช่น ระบบตั้งเวลาการรดน้ำ ระบบเปิด/ปิดไฟตามเวลาที่กำหนด เป็นต้น

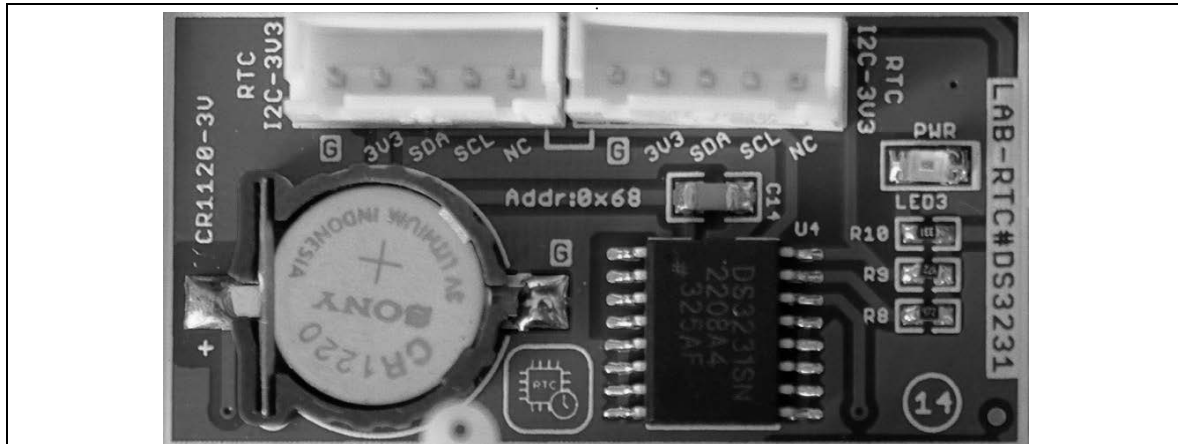
> แนะนำ LAB-RTC DS3231

ภาค LAB-RTC DS3231 เป็นโมดูลสำหรับการใช้งานนาฬิกาฐานเวลาจริงที่ใช้ไอซี DS3231 ของบริษัท Analog Device ที่มีคุณสมบัติดังนี้

- ทำงานที่แรงดันไฟฟ้ากระแสตรง 3.3V
- ทำงานในอุณหภูมิ 0°C ถึง +70°C
- เก็บค่าวินาที นาฬิกา ชั่วโมง
- วันที่ในเดือน เดือน วันในสัปดาห์ และปีแบบ ค.ศ.
- รองรับการทำงานจนถึงปี ค.ศ. 2100
- ความแม่นยำ ±2ppm ในช่วง 0°C ถึง +40°C
- ความแม่นยำ ±3.5ppm ในช่วง -40°C ถึง +85°C

- ความเร็วสูงสุดในการสื่อสารบนบัส I²C อยู่ที่ 400kHz
- รองรับการวัดอุณหภูมิในช่วง -40°C ถึง +85°C
- ความแม่นยำของการวัดอุณหภูมิอยู่ที่ $\pm 3^\circ\text{C}$

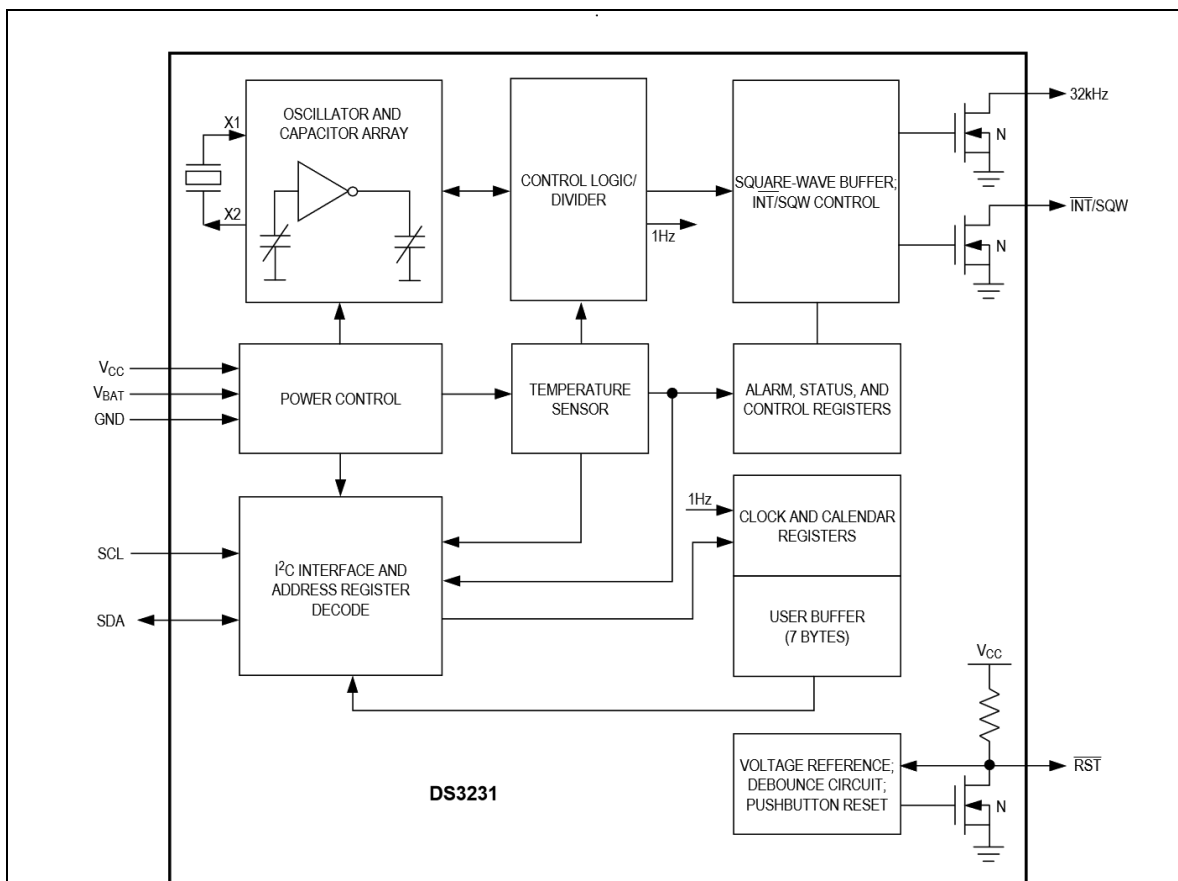
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



ภาพที่ 17-1

โมดูล LAB-RTC DS3231

ภาพโดย จารุต บุศราทิจ



ภาพที่ 17-2

ผังบล็อกทำงานของ DS3231

ภาพโดย จารุต บุศราทิจ

ผังบล็อกทำงานภายในไอซี DS3231 เป็นดังภาพที่ 17-2 จะพบว่า การใช้งานต้องเชื่อมต่อ 8 สายสัญญาณ ได้แก่ แหล่งจ่ายไฟแบบกระแสตรงผ่านทางขา V_{CC} , แหล่งจ่ายไฟจากแบตเตอรี่สำหรับเก็บข้อมูลในชิพผ่านทางขา V_{BAT} , ขากราวด์ที่ขา GND, ขา SCL/SDA สำหรับบัส I²C, ขารีเซ็ตที่ทำงานเมื่อสถานะเป็น LOW, ขาความถี่ 32kHz และขาสถานะการณเกิดการขัดจังหวะตามการตั้งเวลา (INT) หรือใช้เป็นขานำออกสัญญาณแบบสี่เหลี่ยมหรือ Square Wave (SQW)

โดยจากผังบล็อกการทำงานจะพบว่าไม่จำเป็นต้องต่อคริสตัลหรือตัวสร้างสัญญาณนาฬิกาภายนอกให้กับ DS3231 และภายในมีโมดูลวัดอุณหภูมิเพื่อใช้ตรวจสอบความร้อนของไอซี

ภาคควบคุมพลังงานของ DS3231 มีเงื่อนไขของการเลือกแหล่งจ่ายไฟดังนี้

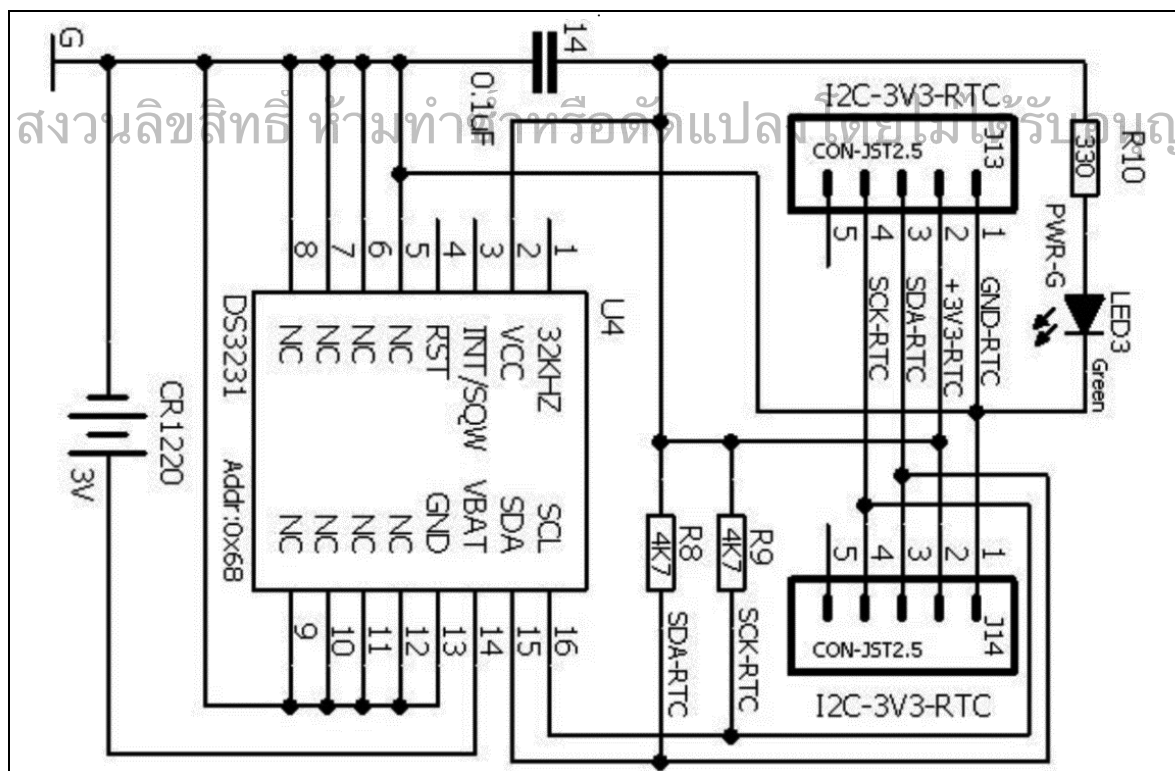
- ถ้า $V_{CC} < V_{PF}$ และ $V_{CC} < V_{BAT}$ จะเลือกแหล่งจ่ายไฟจาก V_{BAT}
- ถ้า $V_{CC} < V_{PF}$ และ $V_{CC} > V_{BAT}$ จะเลือกแหล่งจ่ายไฟจาก V_{CC}
- ถ้า $V_{CC} > V_{PF}$ และ $V_{CC} < V_{BAT}$ จะเลือกแหล่งจ่ายไฟจาก V_{CC}
- ถ้า $V_{CC} > V_{PF}$ และ $V_{CC} > V_{BAT}$ จะเลือกแหล่งจ่ายไฟจาก V_{CC}

> วงจรและการทำงาน

วงจรของ LAB-RTC DS3231 เป็นดังภาพที่ 17-3 จะพบว่าหลอดแอลอีดี LED3 ทำหน้าที่แสดงสถานะของบอร์ดเมื่อมีการจ่ายแรงดันไฟฟ้ากระแสตรงเข้าวงจรทำให้ทราบว่าพร้อมทำงาน โดยการเชื่อมประสานกับบอร์ดของ ESP32 จะใช้พอร์ตเชื่อมต่อผ่านขั้ว I²C แบบ 3V3 มีค่าตำแหน่งของอุปกรณ์เป็น 0x68 และบนบอร์ดมีตัวต้านทาน 4k7 เป็น Pull-up สำหรับสายสัญญาณ SDA และ SCL เพื่อป้องกันสัญญาณอ่อนเมื่อส่งผ่านบัส

นอกจากนี้ บนบอร์ดติดตั้งถังถ่านแบบเหรียญแรงดัน 3V รุ่น CR1220 เพื่อเป็นแบตเตอรี่สำรองของไอซี DS3231 ทำให้เก็บค่าวันที่และเวลาได้ตลอดโดยไม่ต้องตั้งค่าใหม่ทุกครั้งที่ใช้งาน

การเขียนโปรแกรมใช้งานโมดูล LAB-RTC DS3231 สามารถเขียนเองด้วยการส่งชุดคำสั่งและข้อมูลตามเอกสารของไอซี หรือใช้ไลบรารีสำเร็จรูป โดยในหนังสือเล่มนี้เลือกใช้ไลบรารี uRTCLib



ภาพที่ 17-3

วงจรของ LAB-RTC DS3231

ภาพโดย จารุต บุศราทิจ

> ไบรารี uRTCLib

ไลบรารี uRTCLib เขียนโดย Naguissa เพื่อใช้งานกับ DS1307, DS3231 และ DS3232 พร้อมรองรับการทำงานกับไมโครคอนโทรลเลอร์ เช่น AVR, STM32, ESP8266 และ ESP32 เป็นต้น โดยการใช้งานไลบรารีตัวนี้ต้องติดตั้งเพิ่มเติมดังภาพที่ 17-4

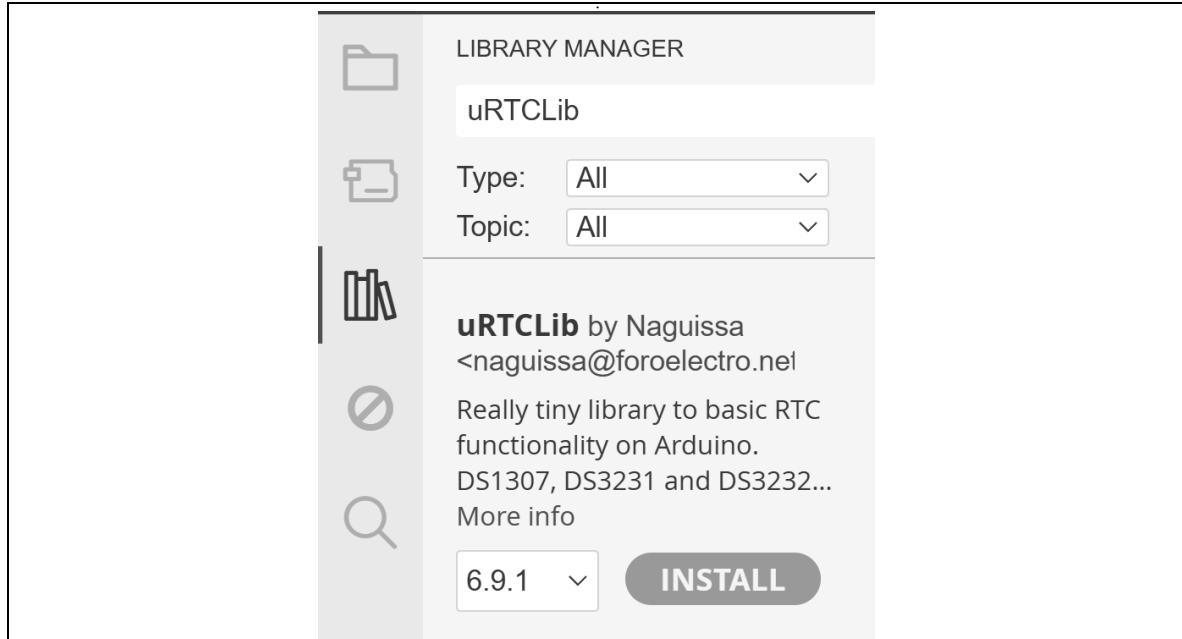
การเริ่มต้นใช้งานต้องนำเข้าไฟล์ส่วนหัวชื่อ uRTCLib.h ดังนี้

```
#include <uRTCLib.h>
```

หลังจากนำเข้าต้องสร้างวัตถุที่เป็นตัวแปรสำหรับอ้างอิงไปยังโมดูล RTC พร้อมทั้งระบุ ค่าตำแหน่งอุปกรณ์ และ ค่ารุ่นอุปกรณ์ โดย ค่าตำแหน่งอุปกรณ์ เป็น 0x68 และ ค่ารุ่นอุปกรณ์ มีค่าเป็น URTCLIB_MODEL_DS3231 ดังนี้

uRTCLib ชื่อวัตถุ(คำตำแหน่งอุปกรณ์, คำรุ่นอุปกรณ์)

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



ภาพที่ 17-4

ไลบรารี uRTCLib

ภาพโดย จารุต บุศราทิจ

การเริ่มต้นให้โมดูล DS3231 ทำงานต้องเรียกใช้คำสั่ง begin() ในฟังก์ชัน setup() ดังนี้

`URTCLIB_WURE.begin()`

ชุดคำสั่งสำหรับอ่านค่าตัวเลขจำนวนเต็มของชั่วโมง นาที และวินาทีที่เก็บไว้ในโมดูล DS3231 คือ hour() minute() และ second() ดังรูปแบบการใช้งานต่อไปนี้

```
ชั่วโมง = ชื่อวัตถุ.hour()
นาที = ชื่อวัตถุ.minute ()
วินาที = ชื่อวัตถุ.second()
```

การอ่านค่าโหมดเวลาต้องใช้คำสั่ง `hourModeAndAmPm()` ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

```
ค่าโหมด = ชื่อวัตถุ.hourModeAndAmPm()
```

โดยที่

ค่าโหมด เป็นค่าที่ได้จากโมดูล RTC อันได้แก่

- 0 หมายถึง โหมด 24 ชม.
- 1 หมายถึง โหมด 12 ชม. แบบ AM
- 2 หมายถึง โหมด 12 ชม. แบบ PM

ชุดคำสั่งสำหรับอ่านวัน เดือน ปี และวันในสัปดาห์ที่เก็บไว้ คือ `day()` `month()` `year()` และ `dayOfWeek()` ดังรูปแบบการใช้งานต่อไปนี้

```
วันปัจจุบัน = ชื่อวัตถุ.day()
```

```
เดือนปัจจุบัน = ชื่อวัตถุ.month()
```

```
ปี ค.ศ. ปัจจุบัน = ชื่อวัตถุ.year()
```

```
วันในสัปดาห์ = ชื่อวัตถุ.dayOfWeek()
```

คำสั่งให้มีการรีเฟรชข้อมูลของ DS3231 ต้องเรียกใช้ `refresh()` และมีคืนค่าผลของการรีเฟรชเป็น `true` หรือ `false` อันเป็นผลของการสั่งรีเฟรชตามรูปแบบคำสั่งต่อไปนี้

```
ผลของการรีเฟรช = ชื่อวัตถุ.refresh()
```

และคำสั่งสำหรับตั้งค่าวัน/เวลากับรูปแบบโหมดเวลาของโมดูล LAB-RTC DS3231 จะใช้คำสั่ง `set()` และ `set_12hour_mode()` โดยแต่ละคำสั่งมีรูปแบบของพารามิเตอร์แตกต่างกัน คือ คำสั่ง `set()` จะต้องกำหนดค่า วินาที, นาที, ชั่วโมง, วันในสัปดาห์, วัน, เดือน และ ปีแบบค.ศ. ส่วนคำสั่ง `set_12hour_mode()` จะต้องระบุรูปแบบของ โหมด เป็นค่าตัวเลขใดตัวเลขหนึ่งดังต่อไปนี้ 0, 1 หรือ 2 อันหมายถึง โหมด 24 ชม., โหมด 12 ชม. แบบ AM และ โหมด 12 ชม. แบบ PM ตามลำดับ

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```

ชื่อวัตถุ.set( วินาที, นาที, ชั่วโมง, วันในสัปดาห์, วัน, เดือน, ปี )
ชื่อวัตถุ.set_12hour_mode( โหมด )

```

> วันและเวลา

ในเฟรมเวิร์กของ Arduino มีโครงสร้างข้อมูลสำหรับเก็บข้อมูลของวันและเวลาชื่อ tm ที่อยู่ในไฟล์ส่วนหัว time.h ดังนี้

```

struct tm {
    int8_t tm_sec;
    int8_t tm_min;
    int8_t tm_hour;
    int8_t tm_mday;
    int8_t tm_wday;
    int8_t tm_mon;
    int16_t tm_year;
    int16_t tm_yday;
    int16_t tm_isdst;
};

```

การสร้างตัวแปรต้องสืบทอดจาก tm ดังนี้

```
struct tm ตัวแปร;
```

การอ่านค่าเวลาจากระบบต้องใช้คำสั่ง localtime() ตามด้วยค่าตำแหน่งของตัวแปรประเภท tm ที่สร้างไว้ ดังรูปแบบการใช้งานต่อไปนี้

```
localtime(&ตัวแปร)
```

การตั้งเวลาเพื่อให้มีความเที่ยงตรงนิยมเรียกใช้บริการ NTP หรือ Network Time Protocol อันเป็นโพรโทคอลสำหรับอ่านค่าวันที่และเวลาบนเครือข่ายทำให้เครื่องในเครือข่ายนั้นตั้งเวลาได้ตรงกันหรือคลาดเคลื่อนน้อยที่สุด นอกจากนี้ด้วยการที่เป็นโพรโทคอลที่ทำงานแบบ UDP จึงมีความรวดเร็วในการสื่อสาร

คำสั่ง configTime() มีหน้าที่สำหรับการอ่านค่าเวลาจากเครื่องให้บริการ NTP โดยต้องระบุพารามิเตอร์ 3 รายการ คือ GMT, Day_{Light} และ NTP_{server} ดังรูปแบบต่อไปนี้

```
configTime( GMT, DayLight, NTPserver )
```

ค่าชื่อเครื่องให้บริการ NTP ที่เป็นแหล่งหลัก คือ time.nist.gov และค่าช่วงเวลาเป็นค่าระบุจำนวนวินาทีจากเวลาของกรีนิช ดังนั้น เมื่อใช้กับประเทศไทยจึงเป็นค่า GMT เป็น 7*3600 หรือ GMT+7 พร้อมทั้งกำหนดค่า Day_{Light} เป็นค่าเดียวกัน คือ 7*3600 โดยรายชื่อเครื่องให้บริการ NTP ภายในประเทศไทย ได้แก่

- 1.th.pool.ntp.org
- asia.pool.ntp.org
- 1.asia.pool.ntp.org
- time.navy.mi.th

เวลามาตรฐานประเทศไทย โดย กรมอุทกศาสตร์ กองทัพเรือ

- time2.navy.mi.th

เวลามาตรฐานประเทศไทย โดย กรมอุทกศาสตร์ กองทัพเรือ

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 17-1 เป็นการตั้งเวลาจากอินเทอร์เน็ตผ่านไวไฟและตั้งค่าเวลาให้กับโมดูล RTC โดยมีการตั้งค่า SSID และรหัสผ่านสำหรับไวไฟในบรรทัดที่ 8-9 ตั้งค่าทดเวลาเพื่อปรับเวลามาตรฐาน GMT+7 ในบรรทัดที่ 11-12 สร้างฟังก์ชันในการปรับปรุงค่าวันที่และเวลาและแสดงผ่านโมดูล OLED ในบรรทัดที่ 20-58 ทำการต่อไวไฟจนสำเร็จและตั้งค่าเวลาให้กับโมดูล RTC ในบรรทัดที่ 68-83

และสร้างฟังก์ชันในการดึงข้อมูลเวลาจากอินเทอร์เน็ตในบรรทัดที่ 98-117 โดยมีการต่อขา SDA และ SCL ของโมดูล OLED และ โมดูล RTC เข้ากับขา 21 และ 22 ตามลำดับ

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 17-1 ตั้งค่าวันที่และเวลาจาก NTP

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	#include <time.h>
3	#include <uRTCLib.h>
4	#include <Adafruit_GFX.h>
5	#include <Adafruit_SH1106.h>
6	#define SDA 21
7	#define SCL 22
8	const char* ssid = "ชื่อ SSID";
9	const char* password = "รหัสผ่าน";
10	const char* ntpServer = "pool.ntp.org";
11	const long gmtoffset_sec = 7 * 3600;
12	const int daylightOffset_sec = 7 * 3600;
13	struct tm timeinfo;
14	String Str_Day[] = {"Sun", "Mon", "Tue", "Wed",
15	"Thu", "Fri", "Sat"};
16	String Str_Month[] = {"Jan", "Feb", "Mar", "Apr",
17	"May", "Jun", "Jul", "Aug",
18	"Sep", "Oct", "Nov", "Dec"};
19	Adafruit_SH1106 display(SDA, SCL);
20	uRTCLib rtc(0x68, URTCLIB_MODEL_DS3231);
21	int hour, minute, sec, day, date, month, year;
22	void updateTime(){
23	hour = rtc.hour();
24	if(hour < 10){
25	display.print("0");
26	}
27	display.print(hour);
28	display.print(":");
29	minute = rtc.minute();
30	if(minute < 10){
31	display.print("0");

ตัวอย่างโปรแกรม 17-1 ตั้งค่าวันที่และเวลาจาก NTP (ต่อ)

บรรทัด	คำสั่ง
32	}
33	display.print(minute);
34	display.print(":");
35	sec = rtc.second();
36	if(sec < 10){
37	display.print("0");
38	}
39	display.print(sec);
40	}
41	void updateDate(){
42	day = rtc.dayOfWeek();
43	display.print(Str_Day[day - 1]);
44	display.print(",");
45	date = rtc.day();
46	if(date < 10){
47	display.print("0");
48	}
49	display.print(date);
50	display.print(" ");
51	month = rtc.month();
52	display.print(Str_Month[month - 1]);
53	display.print(" ");
54	year = rtc.year();
55	display.print("20");
56	if(year < 10){
57	display.print("0");
58	}
59	display.print(year);
60	}
61	void setup(){
62	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
63	display.clearDisplay();
64	display.display();
65	delay(100);

ตัวอย่างโปรแกรม 17-1 ตั้งค่าวันที่และเวลาจาก NTP (ต่อ)

บรรทัด	คำสั่ง
66	display.setTextSize(1);
67	display.setTextColor(WHITE);
68	display.setCursor(0, 0);
69	display.println("Connecting to Wifi");
70	WiFi.begin(ssid, password);
71	while (WiFi.status() != WL_CONNECTED) {
72	delay(500);
73	display.print(".");
74	display.display();
75	}
76	display.clearDisplay();
77	display.setCursor(0, 0);
78	display.print("Connect Success");
79	display.display();
80	delay(3000);
81	configTime(gmtOffset_sec, daylightOffset_sec,
82	ntpServer);
83	URTCLIB_WIRE.begin();
84	printLocalTime();
85	WiFi.disconnect(true);
86	WiFi.mode(WIFI_OFF);
87	}
88	void loop(){
89	rtc.refresh();
90	display.clearDisplay();
91	display.setTextSize(2);
92	display.setCursor(15, 10);
93	updateTime();
94	display.println("");
95	display.println("");
96	display.setTextSize(1);
97	display.setCursor(15, display.getCursorY());
98	updateDate();
99	display.display();

ตัวอย่างโปรแกรม 17-1 ตั้งค่าวันที่และเวลาจาก NTP (ต่อ)

บรรทัด	คำสั่ง
100	}
101	void printLocalTime(){
102	display.clearDisplay();
103	display.setCursor(0, 0);
104	display.println("Retrieving Time from Internet");
105	display.display();
106	if(!getLocalTime(&timeinfo)){
107	display.print(".");
108	display.display();
109	delay(500);
110	}
111	display.clearDisplay();
112	display.setCursor(0, 0);
113	display.print("Retrieve Success");
114	display.display();
115	delay(3000);
116	rtc.set(timeinfo.tm_sec, timeinfo.tm_min,
117	timeinfo.tm_hour, timeinfo.tm_wday + 1,
118	timeinfo.tm_mday, timeinfo.tm_mon,
119	timeinfo.tm_year - 100
120	);
121	}

ตัวอย่างโปรแกรม 17-2 เป็นตัวอย่างการตั้งเวลาเปิดปิดทุก 10 วินาทีโดยใช้วิธีการหารเอาเศษของวินาทีที่ได้จากโมดูล RTC ด้วย 10 หากเศษเป็น 0 จะสั่งให้ไฟติดหรือดับ โดยสร้างฟังก์ชันดึงค่าเวลาจากอินเทอร์เนตและตั้งให้กับโมดูล RTC ในบรรทัดที่ 41-49 ทำการตรวจสอบเวลาหากเวลาหารด้วย 10 ลงตัวและยังไม่เคยสั่ง LED ให้ติดหรือดับจึงจะสั่งให้ LED ติดหรือดับในบรรทัดที่ 31-39 มีการต่อขาใหม่คือขา 4 เข้ากับ LED ดวงที่ 0 ของโมดูล LED

ตัวอย่างโปรแกรม 17-2 เปิดและปิดรีเลย์ทุก 10 วินาทีด้วย RTC

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	#include <time.h>

ตัวอย่างโปรแกรม 17-2 เปิดและปิดรีเลย์ทุก 10 วินาทีด้วย RTC (ต่อ)

บรรทัด	คำสั่ง
3	#include <uRTCLib.h>
4	#define SDA 21
5	#define SCL 22
6	#define led1 4
7	const char* ssid = "ชื่อ SSID";
8	const char* password = "รหัสผ่าน";
9	const char* ntpServer = "pool.ntp.org";
10	const long gmtoffset_sec = 7 * 3600;
11	const int daylightOffset_sec = 7 * 3600;
12	struct tm timeinfo;
13	int sec;
14	uRTCLib rtc(0x68, URTCLIB_MODEL_DS3231);
15	int ledStatus = LOW;
16	void setup(){
17	pinMode(led1, OUTPUT);
18	WiFi.begin(ssid, password);
19	while (WiFi.status() != WL_CONNECTED) {
20	delay(500);
21	}
22	delay(1000);
23	configTime(gmtoffset_sec, daylightOffset_sec,
24	ntpServer);
25	URTCLIB_WIRE.begin();
26	setLocalTime();
27	WiFi.disconnect(true);
28	WiFi.mode(WIFI_OFF);
29	}
30	void loop(){
31	rtc.refresh();
32	if(rtc.second() % 10 == 0 &&
33	sec != rtc.second()){
34	digitalWrite(led1, ledStatus);
35	if(ledStatus == HIGH){
36	ledStatus = LOW;

ตัวอย่างโปรแกรม 17-2 เปิดและปิดรีเลย์ทุก 10 วินาทีด้วย RTC (ต่อ)

บรรทัด	คำสั่ง
37	}else{
38	ledStatus = HIGH;
39	}
40	sec = rtc.second();
41	}
42	}
43	void setLocalTime(){
44	while(!getLocalTime(&timeinfo)){
45	delay(500);
46	}
47	rtc.set(timeinfo.tm_sec, timeinfo.tm_min,
48	timeinfo.tm_hour, timeinfo.tm_wday + 1,
49	timeinfo.tm_mday, timeinfo.tm_mon,
50	timeinfo.tm_year - 100
51	);
52	}

> สรุป

ภาค LAB-RTC DS3231 เป็นโมดูลนาฬิกาฐานเวลาจริงที่ทำงานผ่านทางบัส I2C แบบ 3V3 ใช้ไลบรารี uRTCLib มีแบตเตอรี่แยก ทำให้สามารถเก็บเวลาได้โดยไม่ต้องตั้งค่าเวลาใหม่ทุกครั้ง สามารถนับเวลาได้แม่นยำสูงสุด $\pm 2\text{ppm}$ เก็บค่าได้ทั้งวันที่และเวลา ใช้งานเป็นโมดูลนับเวลาแยกจากบอร์ดหลักได้ ทำให้การทำงานหลายอุปกรณ์สะดวกมากขึ้น เนื่องจากไม่จำเป็นต้องใช้ฟังก์ชัน delay() ในการหน่วงเวลา

บรรณานุกรม

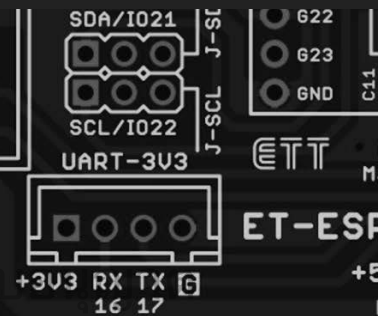
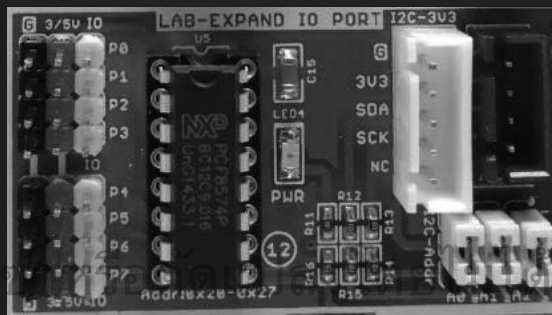
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

Maxim integrated. (2015). *DS3231 Datasheet*. สืบค้นเมื่อ พฤษภาคม 4, 2567, จาก

<https://www.analog.com/media/en/technical-documentation/data-sheets/ds3231.pdf>

Naguissa. (2024). *uRTCLib*. สืบค้นเมื่อ พฤษภาคม 4, 2567, จาก

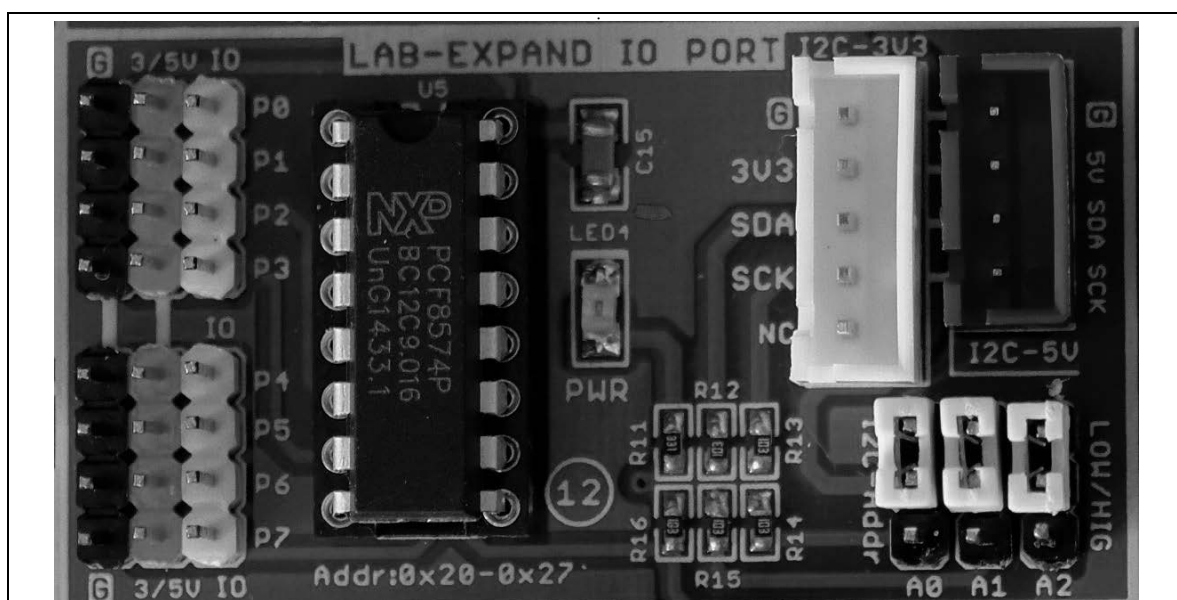
<https://github.com/Naguissa/uRTCLib>



บทที่ 18 LAB-EXPAND IO PORT

LAB-EXPAND IO PORT เป็นโมดูลขยายพอร์ตใช้งานจำนวน 8 บิต ได้แก่ P0, P1, P2, P3, P4, P5, P6 และ P7 ด้วยการใช้งานผ่านบัส I²C แบบ 3V3 โดยสามารถเลือกค่าตำแหน่งได้ ทำให้ไมโครคอนโทรลเลอร์มีจำนวนอุปกรณ์ต่อพ่วงได้มากขึ้น โดยในบทนี้กล่าวถึงคุณสมบัติของ PCF8574P ที่เป็นไอซีขยายพอร์ต วงจรการทำงานเพื่อกำหนดค่าตำแหน่งของอุปกรณ์บนบัสหรือเชื่อมต่อกับไมโครคอนโทรลเลอร์ ESP32 และเขียนโปรแกรมควบคุมการทำงานของพอร์ต P0 ถึง P7 ของ PCF8574P พร้อมตัวอย่างโปรแกรม

เมื่อจบบทนี้ ผู้อ่านสามารถประยุกต์วงจรขยายพอร์ต I/O ผ่านทางบัส I²C ด้วยไอซี PCF8574P เพื่อเพิ่มจำนวนอุปกรณ์ต่อพ่วงของ ESP32 และช่วยให้พอร์ตของ ESP32 ปลอดภัยจากความผิดพลาดที่อาจเกิดจากกระแสหรือแรงดันที่เกินขีดจำกัดที่ ESP32 รองรับได้



ภาพที่ 18-1

โมดูล LAB-EXPAND IO PORT

ภาพโดย จารุต บุศราทิจ

> แนะนำ LAB-EXPAND IO PORT

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ภาค LAB-EXPAND IO PORT ในภาพที่ 18-1 เป็นโมดูลสำหรับขยายพอร์ต I/O จำนวน 8 บิต ได้แก่ P0, P1, P2, P3, P4, P5, P6 และ P7 ด้วยการทำงานไอซี PCF8574P ผ่านการส่งงานจากไมโครคอนโทรลเลอร์ ESP32 ทางบัส I²C และดำเนินการผลการทำงาน เช่น ผลของการอ่านค่าจากขานำเข้าสัญญาณ หรือสถานะของ PCF8574P เป็นต้น โดยในแต่ละบิตมีขั้วต่อไฟและกราวด์ไว้ให้ใช้งานทำให้สะดวกต่อการนำไปพัฒนาต่อ

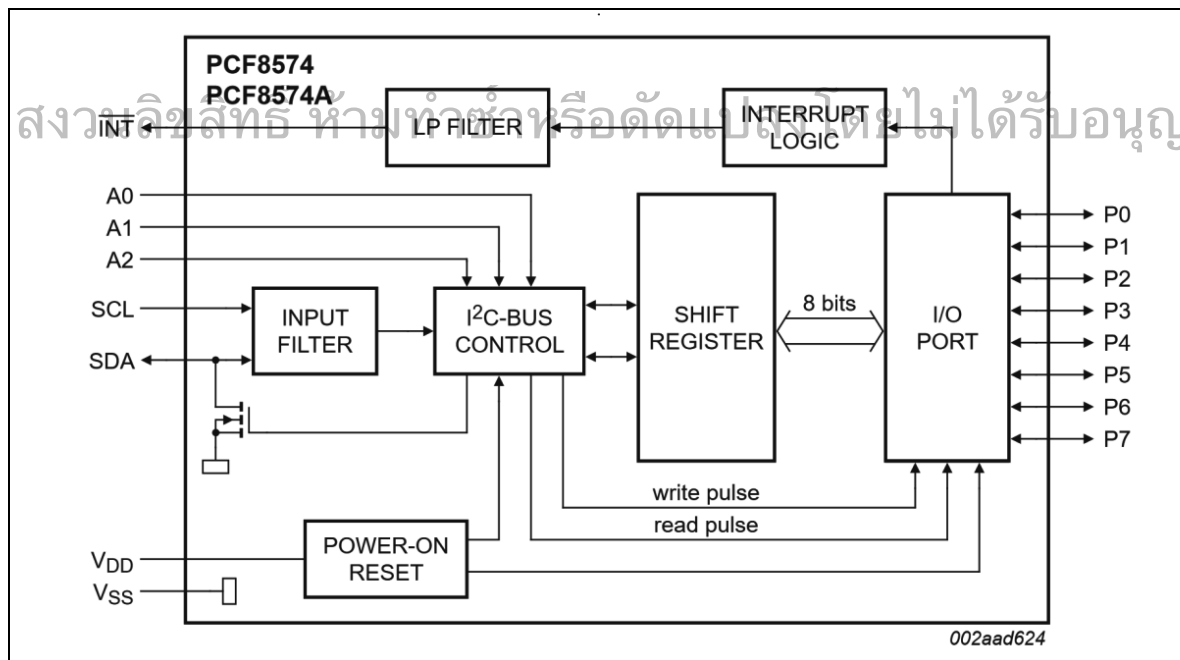
คุณสมบัติของ PCF8574P เป็นดังนี้

- ทำงานที่แรงดัน 2V5 ถึง 6V
- ทำงานที่ความเร็ว 100kHz
- ขยายพอร์ต 8 ขา โดยเมื่อเริ่มต้นทำงานแต่ละพอร์ตทำหน้าที่เป็นขานำเข้าสัญญาณ
- แต่ละขาสามารถขับโหลดแอลอีดีได้โดยตรง
- ทุกขารวมกันสามารถรับกระแสซิงค์ได้ 80mA
- ขณะไม่ได้ทำงานต้องการกระแสไฟ 2.5uA
- ทำงานที่อุณหภูมิ -40°C ถึง 85°C

ผังการทำงานของ PCF8574P เป็นดังภาพที่ 18-2 อันจะพบว่าหลักของการขยายพอร์ตเป็นการใช้ชิพท์เรจิสเตอร์ (Shift Register) สำหรับแปลงบัสแบบอนุกรมให้เป็นแบบขนานที่ถูกเก็บค่าสถานะในบัฟเฟอร์ (I/O Port) ของขาต่าง ๆ และแต่ละขามีวงจรดังภาพที่ 18-3

> วงจรและการทำงาน

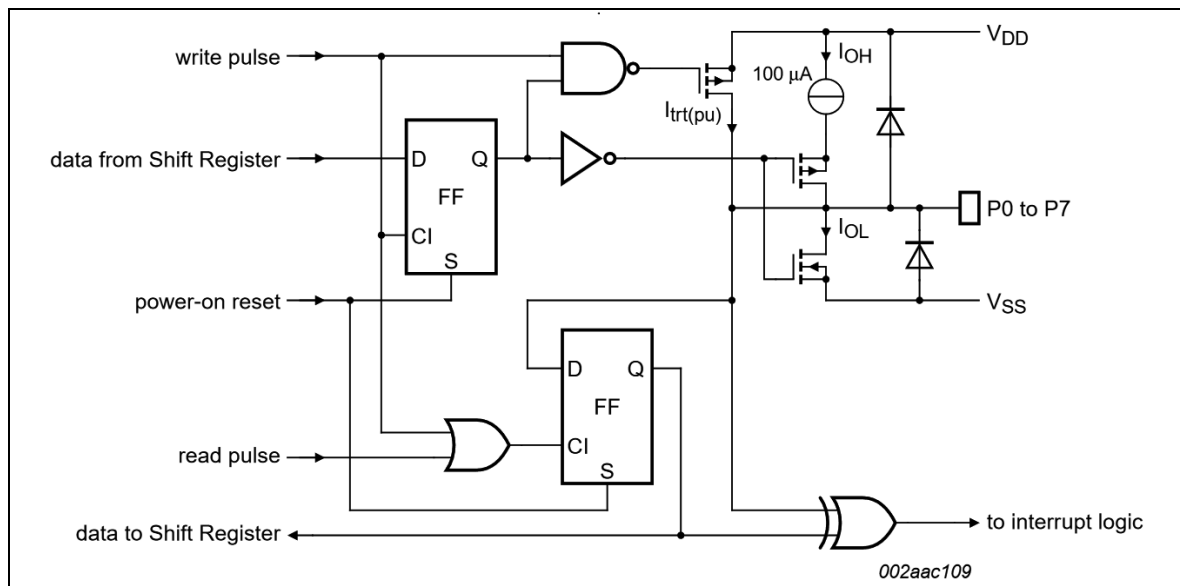
วงจรของ LAB-EXPAND IO PORT เป็นดังภาพที่ 18-4 โดยตำแหน่งของอุปกรณ์บนบัส I²C อยู่ที่ 0x27 หรือ 39 ในเลขฐานสิบ หรือกำหนดเองจากการกำหนดค่าตำแหน่งของอุปกรณ์ LAB-EXPAND IO PORT ทำได้ด้วยการเลือกจัมเปอร์ A0, A1 และ A2 ดังตารางที่ 18-1



ภาพที่ 18-2 แผงการทำงานของ PCF8574P

ที่มา: PCF8574; PCF8574A remote 8-bit I/O expander for I2C-bus with interrupt, Rev.5, 27 May 2013, ภาพที่ 1 หน้าที่ 3

https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf



ภาพที่ 18-3 แผงวงจรของขา P0 ถึง P7 ของ PCF8574

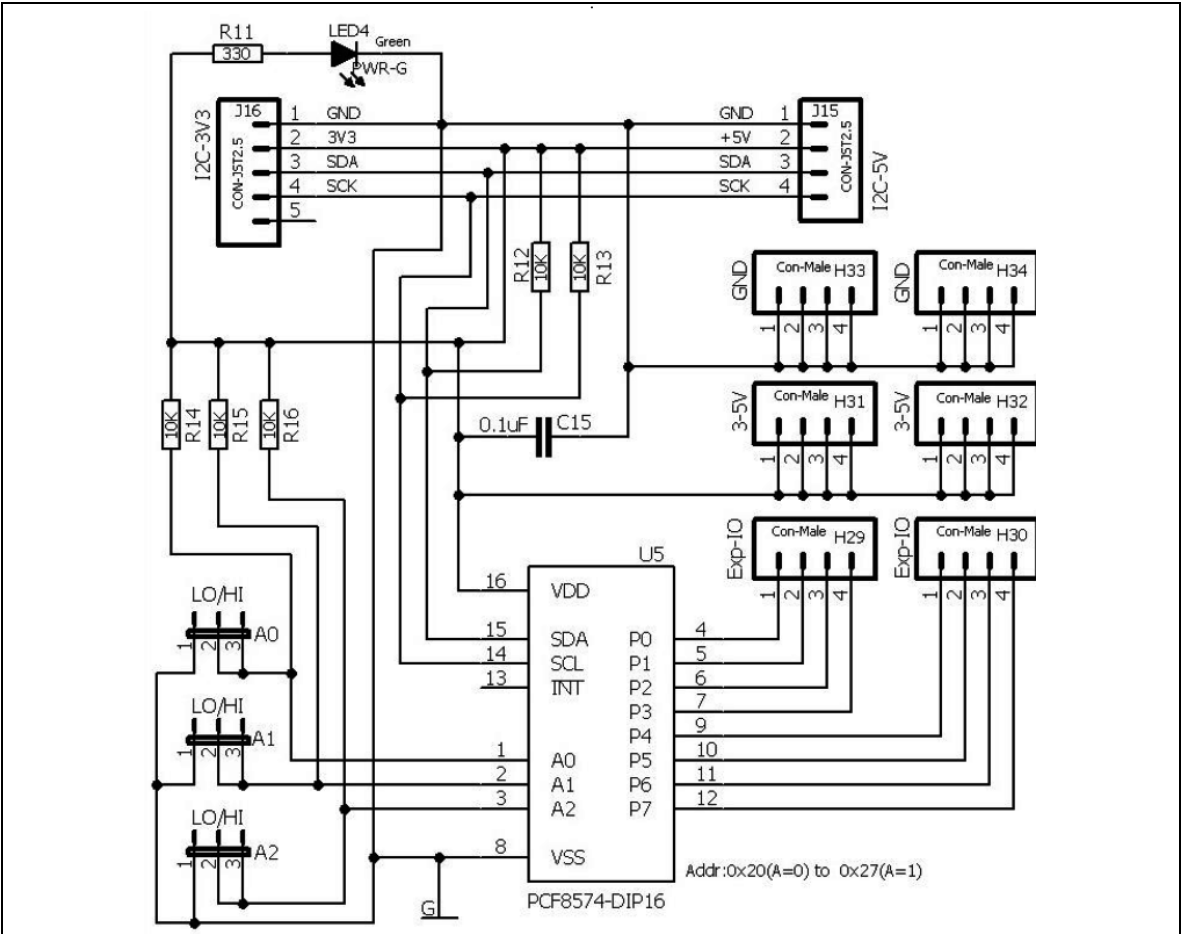
ที่มา: PCF8574; PCF8574A remote 8-bit I/O expander for I2C-bus with interrupt, Rev.5, 27 May 2013, ภาพที่ 2 หน้าที่ 3

https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf

ตารางที่ 18-1 การกำหนดค่าตำแหน่งของ LAB-EXPAND IO PORT

จัมเปอร์			ตำแหน่งพอร์ต
A0	A1	A2	
LO	LO	LO	0x20
HI	LO	LO	0x21
LO	HI	LO	0x22
HI	HI	LO	0x23
LO	LO	HI	0x24
HI	LO	HI	0x25
LO	HI	HI	0x26
HI	HI	HI	0x27

ที่มา: ดัดแปลงจาก ETT. (2024). *MANUAL - คู่มือ ET-ESP32-LAB V1.0*. น. 13. สืบค้นเมื่อ พฤษภาคม 5, 2567, จาก <https://etteam.com/prodKitNLab/ET-ESP32-LABV1/th-man-ET-ESP32-LABV1.pdf>



ภาพที่ 18-4 วงจรของ LAB-EXPAND IO PORT

ภาพโดย จารุต บุศราทิจ

> ไบเบรารี PCF8574

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ไลบรารี PCF8574 พัฒนาโดย MSZ98 โดยต้องติดตั้งไลบรารี PCF8574 และนำเข้าไฟล์ส่วนหัวชื่อ PCF8574.h ดังนี้

```
#include <PCF8574.h>
```

หลังจากนั้นสร้างตัวแปรเพื่อใช้อ้างอิงอุปกรณ์ LAB-EXPAND IO PORT พร้อมทั้งระบุค่าตำแหน่งอุปกรณ์ที่กำหนดจากจัมเปอร์ A0, A1 และ A2 ตามที่กำหนดการตั้งค่าตามตารางที่ 18-1 ดังนี้

PCF8574 ตัวแปรอ้างอิง(ค่าตำแหน่งของอุปกรณ์);

ขั้นตอนหลังจากสร้างตัวแปรอ้างอิงไปยังอุปกรณ์ คือ การสั่งให้ไลบรารีเริ่มต้นทำงานด้วยคำสั่ง begin() ดังนี้

ตัวแปรอ้างอิง.begin()

การกำหนดโหมดของแต่ละขากระทำได้ด้วยการสั่ง pinMode() ตามรูปแบบของการใช้งานต่อไปนี้

ตัวแปรอ้างอิง.pinMode(หมายเลขขา, ประเภทของการเชื่อมต่อ)

การนำออกสัญญาณที่ขาที่ต้องการใช้คำสั่ง digitalWrite() โดยระบุ หมายเลขขา ที่ต้องการกำหนด สถานะของขา เป็น HIGH หรือ LOW ดังนี้

ตัวแปรอ้างอิง.pinMode(หมายเลขขา, สถานะของขา)

การนำเข้าสัญญาณที่ขาที่กำหนดสามารถใช้คำสั่ง digitalRead() ตามรูปแบบของการใช้งานต่อไปนี้

ผลลัพธ์ = ตัวแปรอ้างอิง.digitalRead(หมายเลขขา)

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

โดยค่า ผลลัพธ์ ที่ได้จะเป็นสถานะของ หมายเลขขา ที่สั่งให้มีการอ่านค่า อันได้แก่ HIGH หรือ LOW

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 18-1 เป็นการสั่งงานโมดูล LED ผ่านโมดูล Expand IO โดยสั่งให้ไฟมีการติดและดับในลักษณะของไฟวิ่ง โดยเก็บค่าของขาทั้งหมดที่ต้องใช้ไว้ในตัวแปรอะเรย์ในบรรทัดที่ 3 สั่งโหมดของการใช้งานขาในลักษณะรูปในบรรทัดที่ 8-10 และสั่งให้ไฟดวงที่ต้องการติดและดวงที่ไม่ต้องการให้ดับในบรรทัดที่ 14-20 มีการต่อขา SDA และ SCK ของโมดูล Expand IO เข้ากับขา 21 และ 22 ตามลำดับ และต่อขา P0-P7 ของโมดูล Expand IO เข้ากับ D0 – D7 ของโมดูล LED

ตัวอย่างโปรแกรม 18-1 สั่งงานโมดูล LED ผ่านโมดูล Expand IO

บรรทัด	คำสั่ง
1	#include <PCF8574.h>
2	PCF8574 pcf8574(0x20);
3	int LEDs[] = {P0, P1, P2, P3, P4, P5, P6, P7};
4	int i;
5	int counter = 0;
6	void setup(){
7	for(i = 0; i < 8; i ++){
8	pcf8574.pinMode(LEDs[i], OUTPUT);
9	}
10	pcf8574.begin();
11	}
12	void loop(){
13	for(i = 0; i < 8; i ++){
14	if(i == counter){
15	pcf8574.digitalWrite(LEDs[i], LOW);
16	} else {

ตัวอย่างโปรแกรม 18-1 สักงานโมดูล LED ผ่านโมดูล Expand IO (ต่อ)

บรรทัด	คำสั่ง
17	pcf8574.digitalWrite(LEDs[i], HIGH);
18	}
19	}
20	counter++;
21	if(counter == 8){
22	counter = 0;
23	}
24	delay(500);
25	}

ตัวอย่างโปรแกรม 18-2 เป็นการสั่งงานโมดูล LED ผ่านโมดูล Expand IO และ โมดูลปุ่มกดโดยแสดงไฟติดจากตำแหน่งของปุ่มที่กด โดยใช้อะเรย์เก็บค่าของโมดูล LED ดวงที่ D0-D3 และโมดูลปุ่มกดปุ่มที่ SW0-SW3 ในบรรทัดที่ 3 และ 4 วนลูปตั้งค่าโหมดให้กับ LED และปุ่มกดในบรรทัดที่ 9-14 วนลูปรับค่าการกดของปุ่มทั้ง 4 และเก็บตำแหน่งของปุ่มที่ถูกกดไว้ในตัวแปรในบรรทัดที่ 18-23 และสั่งแสดงไฟติดให้กับ LED จากตำแหน่งของปุ่มที่ถูกกดในบรรทัดที่ 24-30 มีการต่อขา 21 และ 22 เข้ากับขา SDA และ SCK ของโมดูล Expand IO ตามลำดับ ต่อขา P0-P3 ของโมดูล Expand IO เข้ากับขา D0-D3 ของโมดูล LED และต่อขา P4-P7 ของโมดูล Expand IO เข้ากับขา SW0 – SW3 ของโมดูลปุ่มกด

ตัวอย่างโปรแกรม 18-2 ใช้ Expand IO ทั้งแบบ Input และ Output

บรรทัด	คำสั่ง
1	#include <PCF8574.h>
2	PCF8574 pcf8574(0x20);
3	int LEDs[] = {P0, P1, P2, P3};
4	int buttons[] = {P4, P5, P6, P7};
5	int i;
6	int pushedButton = 0;
7	void setup(){
8	for(i = 0; i < 4; i ++){
9	pcf8574.pinMode(LEDs[i], OUTPUT);
10	}
11	for(i = 0; i < 4; i ++){
12	pcf8574.pinMode(buttons[i], INPUT);

ตัวอย่างโปรแกรม 18-2 ใช้ Expand IO ทั้งแบบ Input และ Output (ต่อ)

บรรทัด	คำสั่ง
13	}
14	pcf8574.begin();
15	}
16	void loop(){
17	for(i = 0; i < 4; i ++){
18	if(!pcf8574.digitalRead(buttons[i])){
19	pushedButton = i;
20	break;
21	}
22	}
23	for(i = 0; i < 4; i ++){
24	if(i == pushedButton){
25	pcf8574.digitalWrite(LEDs[i], LOW);
26	}else{
27	pcf8574.digitalWrite(LEDs[i], HIGH);
28	}
29	}
30	delay(100);
31	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-EXPAND IO PORT เป็นโมดูลสำหรับขยายพอร์ตโดยใช้ PCF8574P ใช้ไลบรารี PCF8574 สามารถขยายพอร์ตได้ 8 พอร์ต โดยใช้จัมเปอร์ ในการควบคุมการส่งและแปลงสัญญาณ เพื่อสะดวกต่อการใช้งานกับบอร์ดที่มีพอร์ตน้อยให้สามารถควบคุมอุปกรณ์ได้เพิ่มขึ้น

บรรณานุกรม

NXP. (2013). *PCF8574A Datasheet*. สืบค้นเมื่อ พฤษภาคม 8, 2567, จาก

https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf

MSZ98. (2024). **PCF8574**. สืบค้นเมื่อ พฤษภาคม 8, 2567, จาก

<https://github.com/RobTillaart/PCF8574>

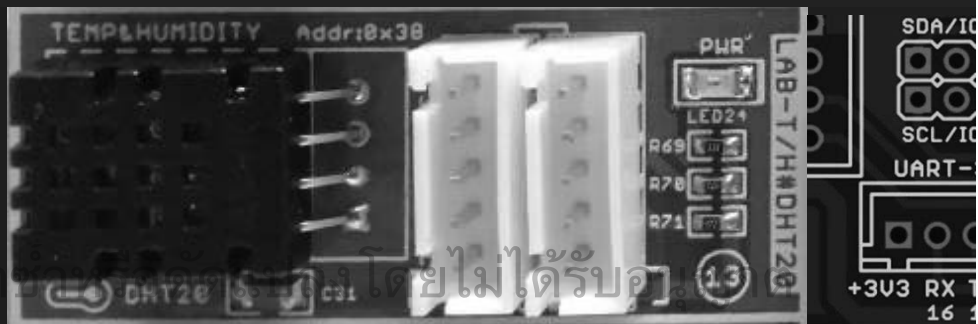
ETT. (2024). *MANUAL - คู่มือ ET-ESP32-LAB V1.0*. น. 13. สืบค้นเมื่อ พฤษภาคม 8, 2567, จาก

<https://etteam.com/prodKitNLab/ET-ESP32-LABV1/th-man-ET-ESP32-LABV1.pdf>



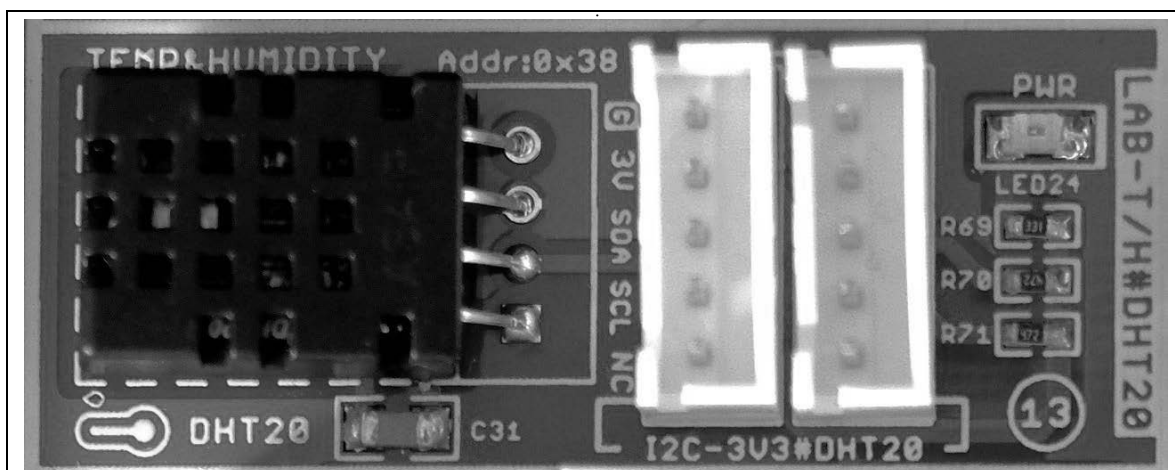
สงวนลิขสิทธิ์ ห้ามทำ

บทที่ 19 LAB-T/H DHT20



เซ็นเซอร์ตระกูล DHT สำหรับอ่านค่าอุณหภูมิและความชื้นที่เป็นที่นิยมใช้เป็นอุปกรณ์พื้นฐานในการเรียนรู้ด้านสมองกลฝังตัวและ IoT มี 2 รุ่น คือ DHT11 และ DHT22 โดยที่โมดูล LAB-T/H DHT20 (ดังภาพที่ 19-1) เลือกใช้เซ็นเซอร์รุ่น DHT20 ที่ปรับปรุงการทำงานมาจากรุ่น DHT11 และรองรับการทำงานผ่านทางบัสสื่อสารแบบ I²C ทำให้สะดวกต่อการเชื่อมประสานกับโมดูลอื่น เช่น เมื่อต้องการสร้างระบบอ่านอุณหภูมิและความชื้นด้วย DHT11 หรือ DHT22 เพื่อแสดงผลบนจอ OLED จะต้องต่อขาเชื่อมต่อสำหรับเซ็นเซอร์เพิ่มอีก 1 ขา แต่เมื่อใช้ DHT20 จะทำให้ประหยัดการเชื่อมต่อขาเพิ่มเติมเนื่องจากใช้การสื่อสารผ่านทางบัส I²C เช่นเดียวกับ OLED เป็นต้น

ในบทนี้กล่าวถึงคุณสมบัติของ DHT20 การเชื่อมประสานกับ ESP32 เพื่อเขียนโปรแกรมผ่านไลบรารี DHT20 ที่พัฒนาโดย Rob Tillaart ดังนั้น เมื่อจบบทนี้ ผู้อ่านสามารถนำไปพัฒนาเป็นต้นแบบของระบบเปิดพัดลมเมื่ออุณหภูมิสูงโดยใช้ร่วมกับโมดูลดีเลย์ หรือระบบระบายรดน้ำแบบฝอยเพื่อควบคุมอุณหภูมิและความชื้นของโรงเรือนให้เหมาะสมกับพืช เป็นต้น



ภาพที่ 19-1

โมดูล LAB-T/H DHT20

ภาพโดย จารุต บุศราทิจ

> แนะนำ LAB-T/H DHT20

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

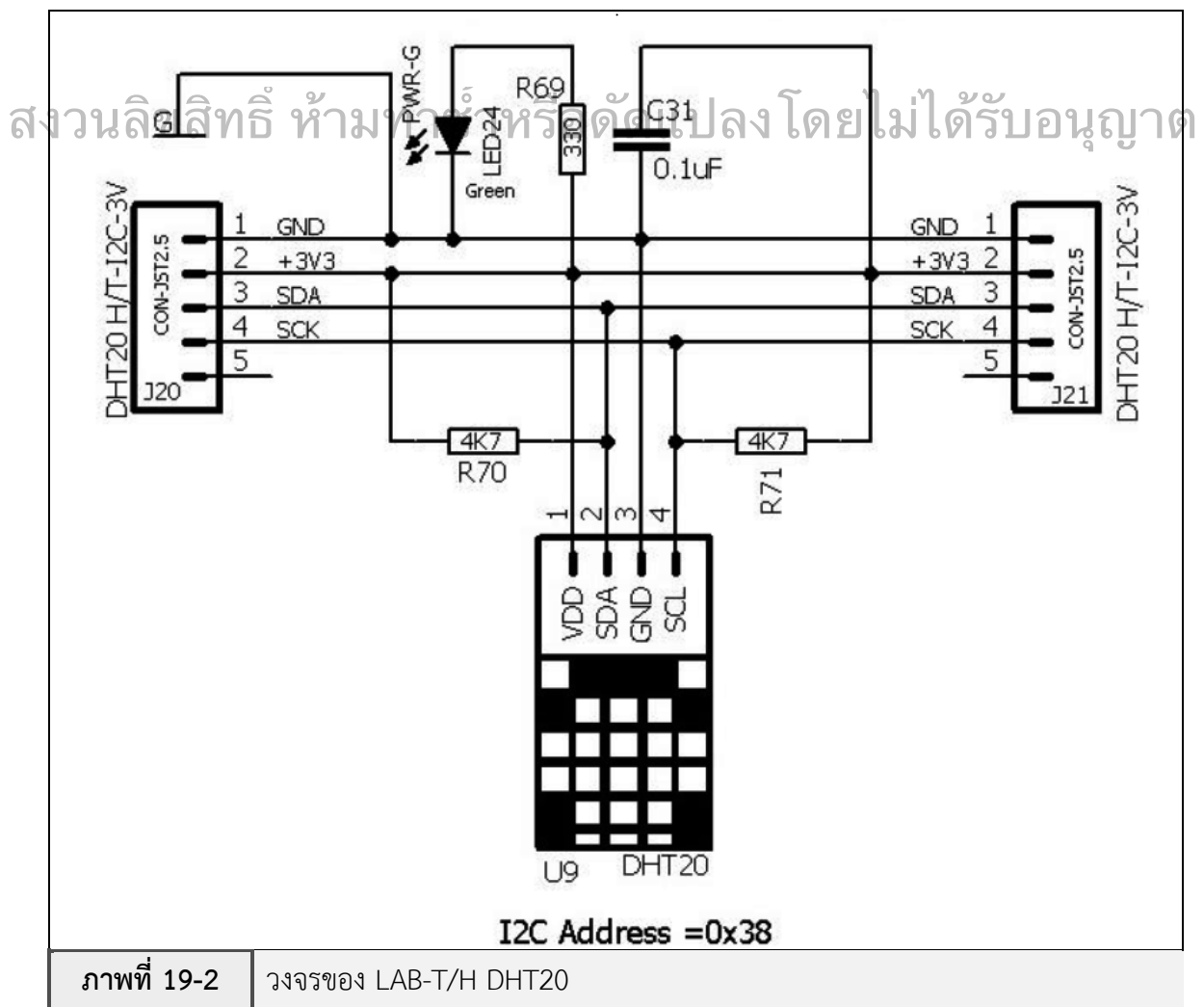
ภาค LAB-T/H DHT20 เป็นโมดูลสำหรับวัดค่าอุณหภูมิและความชื้นโดยมีไอซี DHT20 เป็นตัวทำงานตามคุณสมบัติต่อไปนี้

- ทำงานที่แรงดันไฟฟ้ากระแสตรง 3V3 ถึง 5V5
- ทำงานผ่านบัส I²C
- อุณหภูมิที่วัดได้อยู่ในช่วง -40°C ถึง 80°C
- ความชื้นที่วัดได้อยู่ในย่าน 0 ถึง 100% RH
- ความแม่นยำในการวัดความชื้นอยู่ที่ $\pm 3\%RH$ ที่อุณหภูมิ 25°C
- ความแม่นยำในการวัดอุณหภูมิอยู่ที่ $\pm 0.5^\circ C$
- ความละเอียดในการอ่านค่าอุณหภูมิอยู่ที่ $0.01^\circ C$
- ความละเอียดในการอ่านค่าความชื้นอยู่ที่ $0.024\%RH$
- ต้องการกระแสในการทำงานปกติ 250nA และ 980uA ในช่วงของการวัดค่า
- ระยะเวลาในการตอบสนองในการอ่านค่าความชื้นน้อยกว่า 8 วินาที
- ระยะเวลาในการตอบสนองในการอ่านค่าอุณหภูมิอยู่ในช่วง 5 ถึง 30 วินาที

บนโมดูล LAB-T/H DHT20 มีส่วนเชื่อมต่อสำหรับบัส I2C แบบ 3V3 จำนวน 2 ชุดเพื่ออำนวยความสะดวกในการต่อพ่วงไปยังอุปกรณ์อื่นที่ใช้สื่อสารแบบ I²C โดยแต่ละชุดเชื่อมต่อกับขา 3V3, SDA, SCL และ GND

> วงจรและการทำงาน

วงจรของ LAB-T/H DHT20 เป็นดังภาพที่ 19-2 โดยค่าตำแหน่งของเซ็นเซอร์บนระบบบัส I²C แบบ 3V3 อยู่ที่ 0x38 นอกจากนี้ LED24 เป็นหลอดแอลอีดีสีเขียวที่ทำหน้าที่แสดงสถานะของการจ่ายแรงดันเข้าบอร์ด นั้นหมายความว่า เมื่อเชื่อมต่อกับโมดูลไมโครคอนโทรลเลอร์ ESP32 จะทำให้ขา SDA เชื่อมต่อเข้ากับ P21 และ SCL ต่อกับขา P22



ภาพโดย จารุต บุศราทิจ

> ไบรารี DHT20

ไลบรารี DHT20 จะต้องถูกติดตั้งก่อนใช้งาน และต้องเรียกใช้ไฟล์ส่วนหัว DHT20.h เพื่อให้ตัวแปลภาษารู้จักไลบรารี ดังนี้

```
#include <DHT20.h>
```

เมื่อนำเข้าไลบรารีเสร็จสิ้นต้องสร้างตัวแปรสำหรับใช้อ้างอิงไปยังตัวเซ็นเซอร์ DHT20 โดยระบุข้อมูลการเชื่อมต่อที่สร้างขึ้นจากคลาส Wire.h ดังนี้

```
#include <Wire.h>
```

```
Wire myBus( 21, 22 );
```

```
DHT20 DHT20( &myBus );
```

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

การอ่านข้อมูลจากเซ็นเซอร์ด้วยวิธี Asynchronous มี 4 ขั้นตอน คือ

- ขั้นตอนที่ 1 ร้องขอการอ่านค่าด้วยคำสั่ง requestData()
- ขั้นตอนที่ 2 อ่านค่าข้อมูลดิบด้วยคำสั่ง readData()
- ขั้นตอนที่ 3 ทำการแปลงข้อมูลดิบเป็นข้อมูลอุณหภูมิและความชื้นด้วยคำสั่ง convert()
- ขั้นตอนที่ 4 อ่านค่าอุณหภูมิที่ได้ด้วยคำสั่ง getTemperature() และอ่านค่าความชื้นที่ได้ด้วยคำสั่ง getHumidity()

กรณีทำการอ่านด้วยวิธี Synchronous มี 2 ขั้นตอนด้วยกัน คือ

- ขั้นตอนที่ 1 สั่งอ่านข้อมูลและแปลงข้อมูลด้วยคำสั่ง read()
- ขั้นตอนที่ 2 อ่านค่าอุณหภูมิที่ได้ด้วยคำสั่ง getTemperature() และอ่านค่าความชื้นที่ได้ด้วยคำสั่ง getHumidity()

คำสั่งจากไลบรารี DHT20 มีดังต่อไปนี้

- Bool begin() เป็นคำสั่งสำหรับสั่งเริ่มต้นการทำงานของไลบรารี DHT20
- bool isConnected() เป็นคำสั่งสำหรับตรวจสอบว่ามีการเชื่อมต่อเซ็นเซอร์เข้ากับบัสหรือไม่ ถ้าเชื่อมต่อจะคืนค่าเป็น true และถ้าไม่พบในระบบบัสจะคืนค่าเป็น false
- uint8_t getAddress() เป็นคำสั่งสำหรับคืนค่าตำแหน่งของเซ็นเซอร์ DHT20 โดยในหนังสือเล่มนี้จะคืนค่าเป็น 38₁₆ หรือ 56
- int requestData() เป็นคำสั่งสำหรับร้องขอการเชื่อมต่อกับเซ็นเซอร์ โดยถ้าค่าคืนกลับเป็น 0 หรือ DHT20_OK หมายถึง DHT20 พร้อมให้ข้อมูล แต่ถ้ามีค่าเป็นค่าลบจะหมายถึงเกิดความผิดพลาด
- int readData() เป็นคำสั่งสำหรับอ่านข้อมูลดิบจากบัส I2C หลังจากที่ได้ร้องขอ requestData() และได้ค่าคืนกลับเป็น 0 หรือ DHT20_OK โดยค่าที่ได้นั้นจะเป็นตัวเลขจำนวนเต็ม
- int convert() เป็นคำสั่งแปลงข้อมูลดิบที่อ่านได้จาก readData() ให้เป็นข้อมูลของอุณหภูมิและความชื้น และคืนค่า 0 หรือ DHT20_OK เมื่อทำงานเสร็จ โดยการอ่านค่ามาใช้งานจะเรียกผ่าน getHumidity() หรือ getTemperature()
- int read() สำหรับทำการร้องขอ อ่านค่าข้อมูลดิบ และแปลงข้อมูลดิบเป็นค่าอุณหภูมิและความชื้นในคำสั่งเดียว ถ้าสำเร็จจะคืนกลับเป็น DHT20_OK

- `float getHumidity()` เป็นคำสั่งสำหรับคืนค่าความชื้นปัจจุบันที่ถูกส่งให้อ่านจากเซ็นเซอร์โดยข้อมูลจัดเก็บในรูปแบบของตัวเลขทศนิยม
- `float getTemperature()`; เป็นคำสั่งสำหรับคืนค่าอุณหภูมิปัจจุบันที่ถูกส่งให้อ่านจากเซ็นเซอร์โดยข้อมูลจัดเก็บในรูปแบบของตัวเลขทศนิยม

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 19-1 เป็นการอ่านค่าจากโมดูล DHT20 และนำมาแสดงผ่านโมดูล OLED โดยมีการอ่านค่าสถานะและตรวจสอบหากพบข้อผิดพลาดจะทำการอ่านใหม่ในบรรทัดที่ 23-28 อ่านค่าความชื้นในบรรทัดที่ 29 อ่านค่าอุณหภูมิในบรรทัดที่ 30 ทำการแสดงผลข้อมูลทั้งความชื้นแล้วอุณหภูมิในบรรทัดที่ 32-41 และสร้างกรอบสี่เหลี่ยมเพื่อแสดงความของทั้งความชื้นและอุณหภูมิในบรรทัดที่ 42-45 มีการต่อขา SDA และ SCL ของทั้งโมดูล OLED และโมดูล DHT20 กับขา 21 และ 22 ตามลำดับ

ตัวอย่างโปรแกรม 19-1 แสดงค่าที่อ่านจาก DHT20 ผ่านทาง OLED

บรรทัด	คำสั่ง
1	<code>#include <DHT20.h></code>
2	<code>#include <Adafruit_GFX.h></code>
3	<code>#include <Adafruit_SH1106.h></code>
4	<code>#define SDA 21</code>
5	<code>#define SCL 22</code>
6	<code>float temp, humid;</code>
7	<code>int status;</code>
8	<code>Adafruit_SH1106 display(SDA, SCL);</code>
9	<code>DHT20 dht;</code>
10	<code>void setup(){</code>
11	<code> dht.begin();</code>
12	<code> display.begin(SH1106_SWITCHCAPVCC, 0x3C);</code>
13	<code> display.clearDisplay();</code>
14	<code> display.display();</code>
15	<code> delay(100);</code>
16	<code> display.setTextSize(1);</code>

ตัวอย่างโปรแกรม 19-1 แสดงค่าที่อ่านจาก DHT20 ผ่านทาง OLED (ต่อ)

บรรทัด	คำสั่ง
17	display.setTextColor(WHITE);
18	display.setCursor(0, 0);
19	}
20	void loop(){
21	display.clearDisplay();
22	display.setCursor(0, 0);
23	status = dht.read();
24	if(status != DHT20_OK){
25	display.print("Error");
26	display.display();
27	return;
28	}
29	humid = dht.getHumidity();
30	temp = dht.getTemperature();
31	display.println("");
32	display.print("Humidity = ");
33	display.print(humid, 1);
34	display.print(" %");
35	display.println("");
36	display.println("");
37	display.println("");
38	display.print("Temperature = ");
39	display.print(temp, 1);
40	display.println(" ^C");
41	display.println("");
42	display.drawRect(10, 20, 100, 10, WHITE);
43	display.fillRect(10, 20, humid, 10, WHITE);
44	display.drawRect(10, 50, 100, 10, WHITE);
45	display.fillRect(10, 50, temp, 10, WHITE);
46	display.display();
47	delay(1000);
48	}

ตัวอย่างโปรแกรม 19-2 เป็นตัวอย่างโปรแกรมอ่านค่าจากโมดูล DHT20 ตรวจสอบและเล่นเสียงผ่านโมดูล Piezo โดยตรวจสอบหาอุณหภูมิเกินกำหนดจะเล่นเสียงเตือน โดยมีการสร้างฟังก์ชันสำหรับเล่นเสียงจากตัวโน้ตในบรรทัดที่ 12-15 และตรวจสอบอุณหภูมิหากเกิน 25 องศาให้เล่นเสียงในบรรทัดที่ 29-31 มีการต่อขา SDA และ SCL ของโมดูล DHT20 เข้ากับขา 21 และ 22 ตามลำดับ

ตัวอย่างโปรแกรม 19-2 อ่านค่าจากโมดูล DHT20 ตรวจสอบและเล่นเสียงผ่านโมดูล Piezo

บรรทัด	คำสั่ง
1	#include <DHT20.h>
2	#include <ESP32_Servo.h>
3	#include <ToneESP32.h>
4	#define peizo 12
5	#define SDA 21
6	#define SCL 22
7	#define ch 0
8	float humid, temp;
9	int status;
10	DHT20 dht;
11	ToneESP32 piezo1(peizo, ch);
12	void beep(int note, int duration){
13	piezo1.tone(note, duration);
14	delay(duration*0.25);
15	}
16	void setup(){
17	Wire.begin(SDA, SCL);
18	dht.begin();
19	}
20	void loop(){
21	status = dht.read();
22	if(status != DHT20_OK){
23	delay(100);
24	return;
25	}
26	humid = dht.getHumidity();
27	temp = dht.getTemperature();
28	Serial.println(temp);
29	if(temp > 25){

ตัวอย่างโปรแกรม 19-2 อ่านค่าจากโมดูล DHT20 ตรวจสอบและเล่นเสียงผ่านโมดูล Piezo (ต่อ)

บรรทัด	คำสั่ง
30	beep(NOTE_C4, 250);
31	}
32	delay(1000);
33	}

> สรุป

ภาค LAB-T/H DHT20 เป็นโมดูลที่ปรับปรุงจาก DHT11 ใช้สำหรับวัดค่าความชื้นและอุณหภูมิ รองรับการทำงานผ่านทางบัสสื่อสารแบบ I2C ใช้ไลบรารี DHT20 สามารถวัดอุณหภูมิได้ในช่วง $-40-80^{\circ}\text{C}$ และวัดความชื้นได้ 0-100 % RH แต่ในการใช้งานจำเป็นต้องเว้นช่วงระยะเวลาในการนำเข้าสู่ข้อมูลจากโมดูล DHT20 เพื่อให้การทำงานถูกต้อง เป็นที่นิยมในการประยุกต์ใช้กับการทำสมาร์ทฟาร์ม

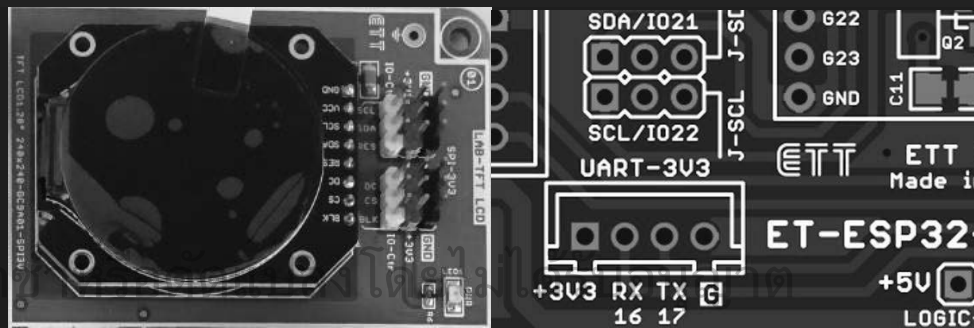
บรรณานุกรม

ASAIR. (2021). *DHT20 Datasheet*. สืบค้นเมื่อ พฤษภาคม 14, 2567, จาก

<https://www.mouser.com/pdfDocs/Productoverview-DFRobot-SEN0497.pdf>

MSZ98. (2024). *DHT20*. สืบค้นเมื่อ พฤษภาคม 14, 2567, จาก

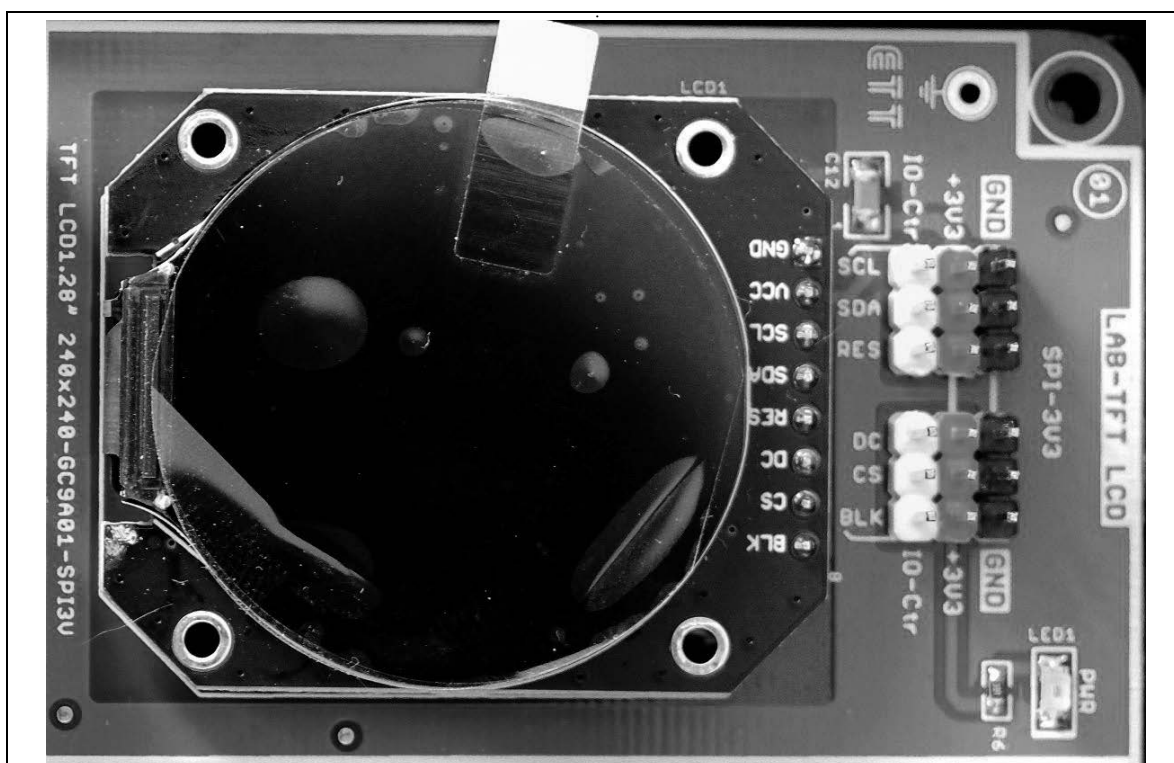
<https://github.com/RobTillaart/DHT20>



บทที่ 20 LAB-TFT LCD

LAB-TFT LCD เป็นโมดูลแสดงผลกราฟิกสีแบบวงกลมที่ใช้อินเทอร์เฟซแบบบัส SPI โดยโมดูลนี้แสดงผลสีแบบ RGB565 ด้วยความละเอียด 240x240 จุดสี หรือพิกเซลในลักษณะของวงกลม ซึ่งในบทนี้กล่าวถึงการต่อใช้งาน การตั้งค่าไลบรารี TFT_eSPI เพื่อให้ ESP32 เชื่อมต่อการทำงานกับโมดูลจอกราฟิกอย่างถูกต้อง และชุดคำสั่งพื้นฐานพร้อมตัวอย่างการใช้งาน

เมื่อจบบทนี้ ผู้อ่านสามารถประยุกต์การแสดงผลการทำงานในแบบกราฟิกสี เช่น สเกลมาตรวัด นาฬิกา หรือกราฟต่าง ๆ เป็นต้น



ภาพที่ 20-1

โมดูล LAB-TFT LCD

ภาพโดย จารุต บุศราทิจ

> แนะนำ LAB-TFT LCD

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ภาค LAB-TFT LCD เป็นโมดูลสำหรับแสดงผลกราฟิกแบบสีที่เกิดจากการผสมระหว่างแม่สีแดง (Red) เขียว (Green) และน้ำเงิน (Blue) ในรูปแบบ RGB565 ทำให้สามารถเก็บค่าสีแต่ละสีด้วยตัวแปรขนาด 16 บิต หรือ 2 ไบต์ ดังภาพที่ 20-2 โดยแต่ละสีมีความละเอียดของค่าสี 5, 6 และ 5 บิตตามลำดับ ทำให้สามารถกำหนดเฉดสีแดงได้ 32 ระดับ (0 ถึง 31) สีเขียวได้ 64 ระดับ (0 ถึง 63) และน้ำเงิน 32 ระดับ (0 ถึง 31) ทำให้สามารถแสดงสีได้ 65536 สี โดยใช้ GC9A01 เป็นไอซีขับเคลื่อนการทำงาน ซึ่งมีคุณสมบัติดังนี้

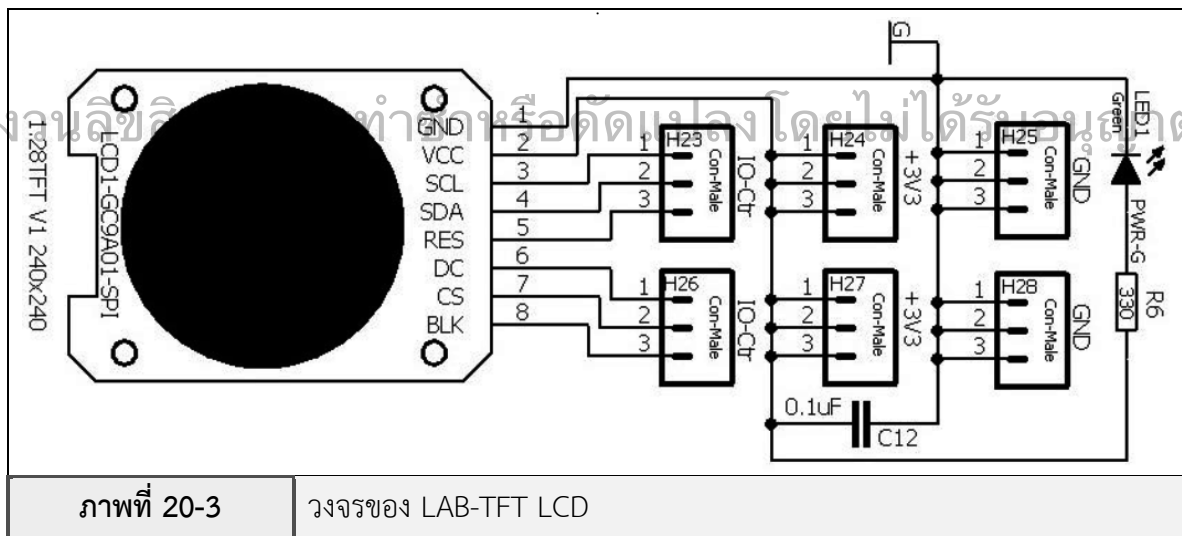
- ทำงานที่แรงดันไฟฟ้ากระแสตรง 3V3 ถึง 5V
- ทำงานที่อุณหภูมิ -40°C ถึง 80°C
- พาเนลเป็นแบบ IPS ทำให้มองเห็นจอสว่างเท่ากันในองศาที่กว้าง
- ความละเอียดในการแสดงผล 240x240 พิกเซล
- หน่วยความจำแรมสำหรับแสดงกราฟิก (GRAM) 129,600 ไบต์

Red สีแดง					Green สีเขียว						Blue สีน้ำเงิน				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5 บิต, 32 ระดับ (0..31)					6 บิต, 64 ระดับ (0..63)						5 บิต, 32 ระดับ (0..31)				
ภาพที่ 20-2					รูปแบบสี RGB565										

ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

วงจรของ LAB-TFT LCD เป็นดังภาพที่ 20-3 จะพบว่าการเชื่อมต่อทำโดยเชื่อมต่อสายเข้ากับ บัส SPI ของ ESP32 จำนวน 6 เส้น และมี LED1 หลอดสีเขียวสำหรับแสดงสถานะการทำงานของโมดูล



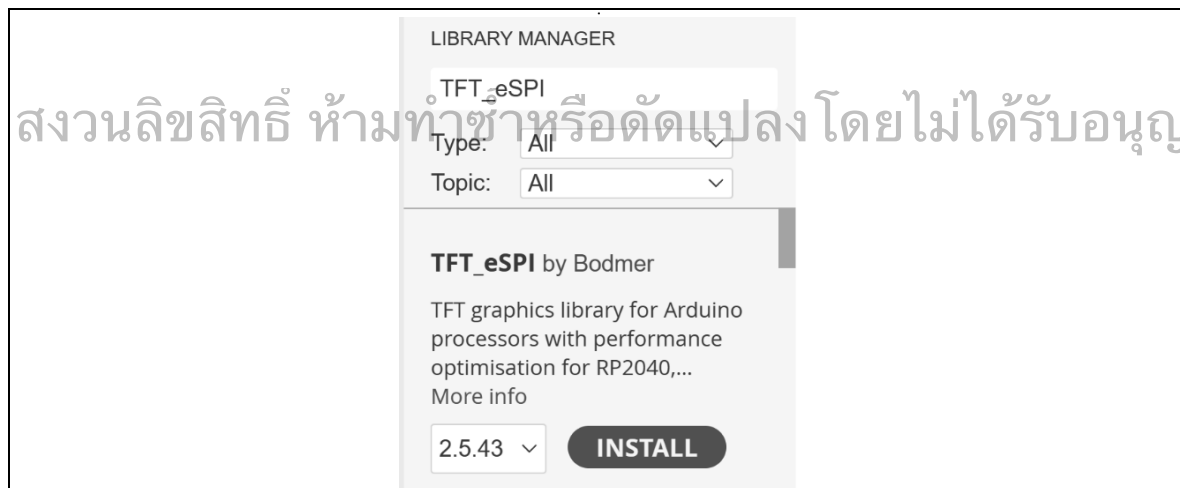
ภาพโดย จารุต บุศราทิจ

การเชื่อมต่อระหว่างโมดูลจอแสดงผลกราฟิกกับ ESP32 เป็นดังนี้

- ขา SCL เชื่อมต่อกับขา P18 มีหน้าที่เป็นขาสัญญาณนาฬิกาในการสื่อสารระหว่างอุปกรณ์กับ ESP32
- ขา SDA เชื่อมต่อกับขา P23 มีหน้าที่ส่งข้อมูลจาก ESP32 ไปยังไอซีควบคุมการแสดงผล
- ขา RES เชื่อมต่อกับขา P13 สำหรับใช้สั่งรีเซ็ตการทำงานของหน่วยแสดงผล
- ขา DC เชื่อมต่อกับขา P14 มีหน้าที่เป็นตัวบอกให้หน่วยควบคุมจอแสดงผลทราบว่าสิ่งที่ส่งมาในสาย SDA เป็นข้อมูลหรือชุดคำสั่ง (Command)
- ขา CS เชื่อมต่อกับขา P25 มีหน้าที่สั่งเลือก/ไม่เลือกให้โมดูลทำงาน
- ขา BL เชื่อมต่อกับขา P4 มีหน้าที่ควบคุมการเปิดหรือปิดแสงหลังของจอแสดงผล

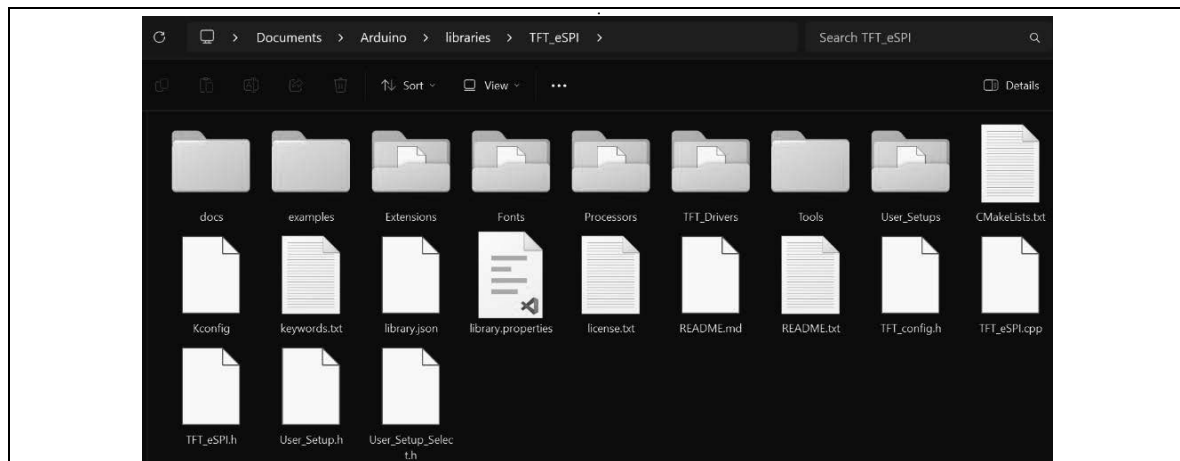
> ไบรารี TFT_eSPI

TFT_eSPI เป็นไลบรารีที่พัฒนาโดย Bodmer มีหน้าที่ใช้สำหรับเชื่อมประสานจอแสดงผลกราฟิกที่รองรับการใช้งานกับหลากหลายรุ่นและรองรับการใช้งาน GC9A01 โดยต้องติดตั้งไลบรารีดังภาพที่ 20-4 หลังจากนั้นต้องเข้าไปตั้งค่าในไฟล์ชื่อ User_Setup_Select.h จากโฟลเดอร์ติดตั้งไลบรารีของ TFT_eSPI ดังตัวอย่างภาพที่ 20-5 จะพบว่าไลบรารีอยู่ในโฟลเดอร์ Documents/Arduino/libraries/TFT_eSPI



ภาพที่ 20-4 ไลบรารี TFT_eSPI

ภาพโดย จารุต บุศราทิจ



ภาพที่ 20-5 โฟลเดอร์ที่เก็บไฟล์ User_Setup_Select.h

ภาพโดย จารุต บุศราทิจ

ในไฟล์ User_Setup_Select.h ได้เรียกใช้งานไฟล์ User_Setup.h ดังตัวอย่างต่อไปนี้

```
#include <User_Setup.h>
```

ด้วยเนื่องจากไลบรารีมีการตั้งค่าให้เลือกใช้จอแสดงผลหลายรุ่น ผู้เขียนโปรแกรมจึงจำเป็นต้องระบุรุ่นที่ถูกต้อง โดยในที่นี้ให้ยกเลิกการเรียกใช้ไฟล์ User_Setup.h ด้วยการใส่เครื่องหมาย // ไว้หน้าบรรทัด ดังนี้

```
// #include <User_Setup.h>
```


หลังจากนั้นให้หาบรรทัดที่มีการเรียกใช้ Setup200_GC9A01.h มีเครื่องหมาย // ใส่ไว้ด้านหน้าบรรทัดดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

```
//#include <User_Setups/Setup200_GC9A01.h>
```

ให้นำเครื่องหมาย // ออกและทำให้บรรทัดคำสั่งมีการเรียกใช้การตั้งค่า GC9A01 สำหรับไมโครคอนโทรลเลอร์ ESP32 ดังนี้

```
#include <User_Setups/Setup200_GC9A01.h>
```

โดยในไฟล์ Setup200_GC9A01.h ให้ทำการแก้ไขข้อมูลในไฟล์ให้มีรายละเอียดเป็นดังต่อไปนี้

```
#define USER_SETUP_ID 200
#define GC9A01_DRIVER

#define TFT_MOSI 23 // SDA
#define TFT_SCLK 18 // SCL
#define TFT_CS 25
#define TFT_DC 14
#define TFT_RST 13 // RES
#define TFT_BL 4

#define LOAD_GLCD
#define LOAD_FONT2
#define LOAD_FONT4
#define LOAD_FONT6
#define LOAD_FONT7
#define LOAD_FONT8
#define LOAD_GFXFF
#define SMOOTH_FONT

#define SPI_FREQUENCY 27000000
#define SPI_READ_FREQUENCY 5000000
```

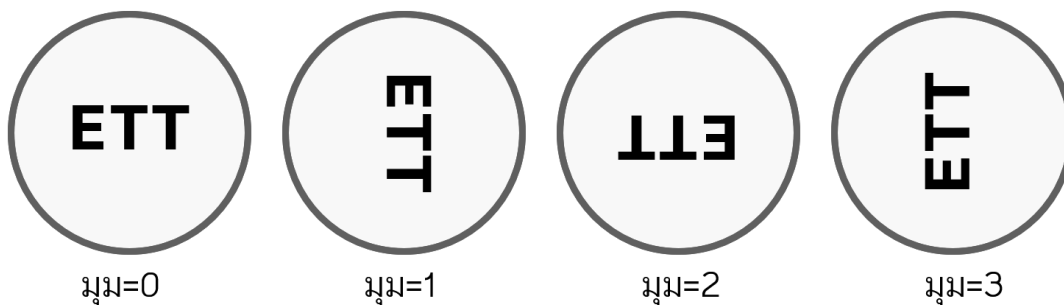
ในการเขียนโปรแกรมจะต้องเรียกใช้ไฟล์ TFT_eSPI.h พร้อมทั้งสร้างตัวแปรวัตถุประเภท TFT_eSPI หรือถ้ามีการใช้ภาพสไปรท์ (Sprite) ต้องสร้างวัตถุประเภท TFT_eSprite ที่ต้องเชื่อมเข้ากับวัตถุประเภท TFT_eSPI ดังตัวอย่างต่อไปนี้

```
#include <TFT_eSPI.h>

TFT_eSPI วัตถุแสดงผล = TFT_eSPI();
TFT_eSprite วัตถุภาพ = TFT_eSprite( &วัตถุแสดงผล );
```

คำสั่งเริ่มทำงานของไลบรารี คือ begin() และต้องกำหนดทิศทางการแสดงผลด้วยการสั่ง setRotation() ตามด้วยพารามิเตอร์ตัวเลขเพื่อระบุ ทิศทางแสดงผล โดย 0 คือ ตั้งตรงปกติ 1 คือ หมุนไปทางขวา 90 องศา 2 คือ หมุนไปทางขวา 180 องศา และ 3 คือ หมุนไปทางขวา 270 องศา (ภาพที่ 20-6) ดังรูปแบบการใช้งานต่อไปนี้

```
วัตถุแสดงผล.begin()
วัตถุแสดงผล.setRotation( ทิศทางแสดงผล )
```



ภาพที่ 20-6

ค่ามุมของ setRotation()

ภาพโดย จารุต บุศราทิจ

สำหรับกรณีที่ต้องการทราบการตั้งค่ามุมสำหรับการแสดงผล ต้องใช้คำสั่ง getRotation() และจะได้ค่าคืนกลับเป็นตัวเลข 0, 1, 2 หรือ 3 โดยรูปแบบของการใช้งานเป็นดังนี้

```
ทิศทางแสดงผล = วัตถุแสดงผล.getRotation()
```

คำสั่ง fillScreen() มีหน้าที่ล้างค่าสีที่แสดงผลของทั้งหน้าจอ โดยค่าสีประกอบด้วย

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

TFT_BLACK TFT_NAVY TFT_DARKGREEN
TFT_DARKCYAN TFT_MAROON TFT_PURPLE
TFT_DARKGREY TFT_OLIVE TFT_BLUE
TFT_GREEN TFT_LIGHTGREY TFT_RED
TFT_CYAN TFT_MAGENTA TFT_YELLOW
TFT_WHITE TFT_ORANGE TFT_GREENYELLOW
TFT_PINK TFT_BROWN TFT_GOLD
TFT_SILVER TFT_SKYBLUE TFT_VIOLET

รูปแบบคำสั่ง fillScreen() เป็นดังนี้

วัตถุแสดงผล.fillScreen(สี)

ถ้าต้องการกำหนดค่าสีด้วยตนเองสามารถสมสีจากแม่สีด้วยคำสั่ง color565() ตามด้วยค่าระดับของแต่ละสีดังนี้

สีผลลัพธ์ = วัตถุแสดงผล.color565(ค่าสีแดง, ค่าสีเขียว, ค่าสีน้ำเงิน)

การลงจุดสีใช้คำสั่ง drawPixel() พร้อมทั้งระบุพิกัด (x, y) ที่ต้องการลงสี และค่าสีที่ต้องการให้แสดงตามรูปแบบต่อไปนี้

วัตถุแสดงผล.drawPixel(x, y, สี)

การวาดตัวอักษร 1 อักษรใช้คำสั่ง drawChar() โดยมีรูปแบบให้ใช้งาน 3 แบบ คือ

- ระบุพิกัด ระบุอักขระ ระบุสีตัวอักษร สีพื้นหลังตัวอักษร และขนาด
- ระบุอักขระแบบยูนิโคด ระบุพิกัด และฟอนต์
- ระบุอักขระแบบยูนิโคด และระบุพิกัด

วัตถุแสดงผล.drawChar(x, y, อักขระ, สีตัวอักษร, สีพื้นหลัง, ขนาดตัวอักษร)

สงวนลิขสิทธิ์ ห้ามนำข้อมูลหรือัดแปลงโดยไม่ได้รับอนุญาต

วัตถุแสดงผล.drawChar(อักขระ_{unicode}, x, y, พื้นหลัง)

วัตถุแสดงผล.drawChar(อักขระ_{unicode}, x, y)

การวาดเส้นตรงสามารถใช้คำสั่ง drawLine() เพื่อลากเส้นที่มีสีตามระบุจาก (xs, ys) ไปยังพิกัด (xe, ye) หรือใช้คำสั่งวาดเส้นตรงในแนวนอนด้วยคำสั่ง drawFastVLine() และแนวตั้งด้วยคำสั่ง drawFastHLine() โดยระบุจุดเริ่มที่ (x, y) ตามด้วยความยาวของเส้น และสีของเส้น ดังนี้

วัตถุแสดงผล.drawLine(xs, ys, xe, ye, สีของเส้น)

วัตถุแสดงผล.drawFastVLine(x, y, ความยาว, สีของเส้น)

วัตถุแสดงผล.drawFastHLine(x, y, ความยาว, สีของเส้น)

กรณีที่ต้องการอ่านค่าสีจากพิกัด (x, y) ต้องใช้คำสั่ง readPixel() และค่าที่คืนกลับเป็นตัวเลขจำนวนเต็มประเภท uint16_t ที่เก็บข้อมูลสีตามรูปแบบ 565

สีผลลัพธ์ = วัตถุแสดงผล.readPixel(x, y)

คำสั่งวาดสี่เหลี่ยมประกอบด้วยคำสั่ง drawRect() และ drawRoundRect() พร้อมระบุพิกัด (x, y) มุมซ้ายบนของสี่เหลี่ยมตามด้วย ความกว้าง ความสูง และสีที่ต้องการวาดและสำหรับการแสดงรูปสี่เหลี่ยมทึบสีต้องใช้คำสั่ง fillRect() และ fillRoundRect() พร้อมระบุพิกัด (x, y) มุมซ้ายบนของสี่เหลี่ยมตามด้วยความกว้าง ความสูง และสีที่ต้องการระบาย หรือถ้าเป็นสี่เหลี่ยมมุมมนจะต้องกำหนดค่ามุมของทั้ง 4 ด้านดังนี้

วัตถุแสดงผล.drawRect(x, y, w, h, สี)

วัตถุแสดงผล.fillRect(x, y, w, h, สี)

วัตถุแสดงผล.drawRoundRect(x, y, w, h, มุม, สี)

วัตถุแสดงผล.fillRoundRect(x, y, w, h, มุม, สี)

กรณีที่ต้องการสีเหลี่ยมที่มีการเกลี่ยสี 2 สีจาก สี₁ ไป สี₂ ต้องเลือกใช้คำสั่ง fillRectVGradient() หรือ fillRectHGradient() สำหรับสร้างการเกลี่ยสีจากบนลงล่าง หรือซ้ายไปขวา โดยระบบพารามิเตอร์ พิกัดเริ่มต้นที่มุมซ้ายบนตามด้วยความกว้างและสูงของสีเหลี่ยม หลังจากนั้นกำหนดค่าสีเริ่มต้นหรือ สี₁ กับ สีปลายทางหรือ สี₂ ตามรูปแบบของคำสั่งต่อไปนี้

```
วัตถุแสดงผล.fillRectVGradient( x, y, w, h, สี1, สี2 )
```

```
วัตถุแสดงผล.fillRectHGradient( x, y, w, h, สี1, สี2 )
```

การวาดวงกลมมี 2 รูปแบบคือ วงกลมไม่ทึบสีทึบกับวงกลมทึบ ด้วยคำสั่ง drawCircle() และ fillCircle() โดยระบุพิกัด (x, y) เป็นจุดกึ่งกลาง ตามด้วยค่ารัศมีและสีของวงกลมดังนี้

```
วัตถุแสดงผล.drawCircle( x, y, รัศมี, สี )
```

```
วัตถุแสดงผล.fillCircle( x, y, รัศมี, สี )
```

วงรีมีคำสั่งวาด 2 รูปแบบเช่นเดียวกันคือวงรีแบบไม่ทึบสีและวงรีแบบทึบสีโดยใช้คำสั่ง drawEllipse() และ fillEllipse() พร้อมบอกจุดกึ่งกลางของวงรีกับค่ารัศมีในแกน x กับ y และค่าสีของวงรี ดังรูปแบบของคำสั่งใช้งานต่อไปนี้

```
วัตถุแสดงผล.drawEllipse( x, y, รัศมีx, รัศมีy, สี )
```

```
วัตถุแสดงผล.fillEllipse( x, y, รัศมีx, รัศมีy, สี )
```

การวาดสามเหลี่ยมด้วยคำสั่ง drawTriangle() และ fillTriangle() มีความแตกต่างจากการวาดรูปทรงอื่น คือ ต้องระบุพิกัดของจุดยอดทั้ง 3 ของสามเหลี่ยม และค่าสีของสามเหลี่ยม ตามรูปแบบใช้งานต่อไปนี้

```
วัตถุแสดงผล.drawTriangle( x1, y1, x2, y2, x3, y3, สี )
```

```
วัตถุแสดงผล.fillTriangle( x1, y1, x2, y2, x3, y3, สี )
```

การกำหนดค่าสีของข้อความที่ต้องการวาดต้องใช้คำสั่ง `setTextColor()` ต้องกำหนดสีตัวอักษร หรือสีพื้นหลังพร้อมระบุสถานะการเติมสีพื้นหลังเป็น `true` เพื่อแสดงสีพื้นหลังแบบนุ่มนวล หรือ `false` สำหรับเทสีพื้นหลังโดยมีต้องเติมค่าสีให้ดูนุ่มนวล ดังรูปแบบต่อไปนี้

```
วัตถุแสดงผล.setColor( สีตัวอักษร )
```

```
วัตถุแสดงผล.setColor( สีตัวอักษร, สีพื้นหลังตัวอักษร, สถานะการเติมพื้นหลัง )
```

สำหรับคำสั่งกำหนดขนาดของตัวอักษรต้องใช้ `setTextSize()` ตามด้วยขนาดของพิกเซลของตัวอักษร

```
วัตถุแสดงผล.setTextSize( ขนาด )
```

ถ้าต้องการให้มีการตัดข้อความขึ้นบรรทัดใหม่โดยอัตโนมัติต้องกำหนดสถานะการตัดคำด้วย `setTextWrap()` ของการตัดคำในแกน X และ Y ด้วยการกำหนดค่าเป็น `true` หรือ `false` ดังรูปแบบการใช้งานต่อไปนี้

```
วัตถุแสดงผล.setTextWrap( แกนx, แกนy )
```

การวาดตัวเลขจำนวนเต็มหรือจำนวนทศนิยมต้องใช้คำสั่ง `drawNumber()` หรือ `drawFloat()` โดยระบุค่าที่ต้องการแสดง พิกัดเริ่มต้นของการแสดง หรือฟอนต์ที่ใช้ ตามรูปแบบของคำสั่งดังนี้

```
วัตถุแสดงผล.drawNumber( ค่าตัวเลขจำนวนเต็ม, x, y )
```

```
วัตถุแสดงผล.drawNumber( ค่าตัวเลขจำนวนเต็ม, x, y, ฟอนต์ )
```

```
วัตถุแสดงผล.drawFloat( ค่าตัวเลขทศนิยม, x, y )
```

```
วัตถุแสดงผล.drawFloat( ค่าตัวเลขทศนิยม, x, y, ฟอนต์ )
```

คำสั่งวาดสายอักขระคือ drawString() ที่สามารถส่งข้อมูลในรูปแบบของ char* หรือ string โดยระบุพิกัดที่ต้องการวาด และพอนต์ด้วยรูปแบบการใช้งานดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

วัตถุแสดงผล.drawString(ค่าสตริง, x, y)

วัตถุแสดงผล.drawString(ค่าสตริง, x, y, พอนต์)

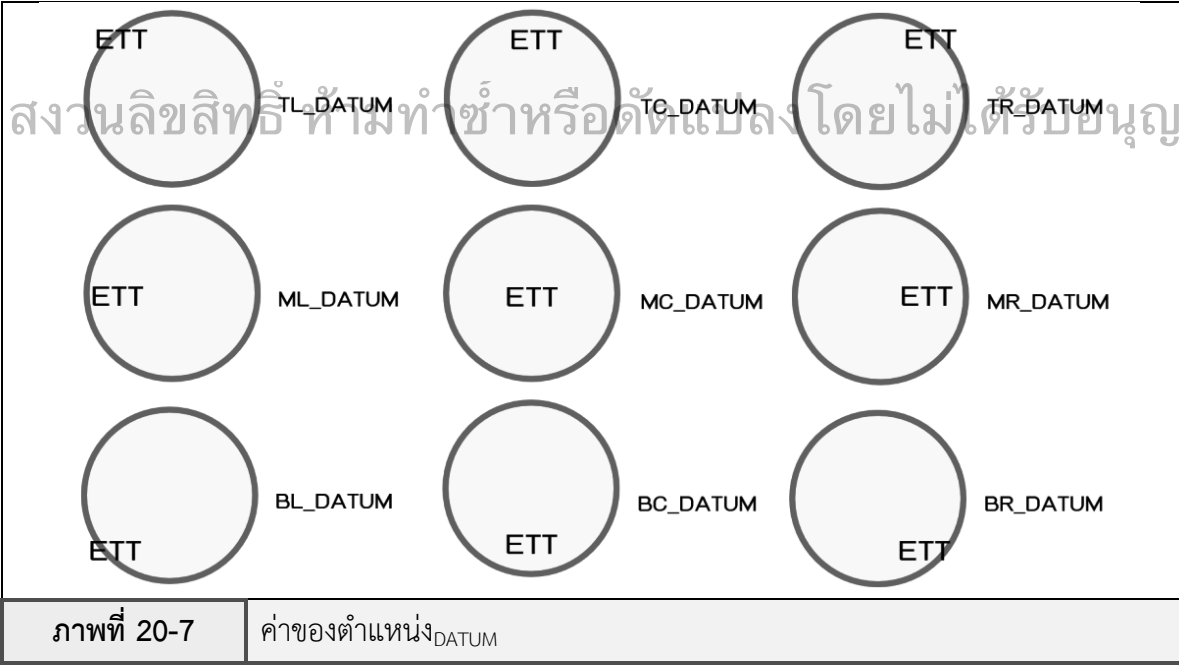
เนืองด้วย drawNumber(), drawFloat() และ drawString() เป็นคำสั่งวาดข้อความแบบชิดซ้าย แต่ถ้าต้องการวาดสตริงแบบระบุให้อยู่กึ่งกลางโดยใช้จุด (x, y) เป็นตำแหน่งกึ่งกลาง หรือชิดขวาโดยใช้พิกัด (x, y) เป็นตำแหน่งขอบขวาของการวาด ต้องใช้คำสั่ง setTextDatum() ดังภาพที่ 20-7 เพื่อระบุเงื่อนไขการวางก่อนเรียก drawString() โดยคำสั่งมีรูปแบบการใช้งานดังนี้

วัตถุแสดงผล.setTextDatum(ตำแหน่ง)

โดยที่

ตำแหน่ง เป็นค่าระบุการวางตำแหน่งของตัวเลขหรือข้อความอันมีค่าดังต่อไปนี้

- TL_DATUM หรือ 0 หมายถึง อยู่ชิดด้านบนของแกน Y และอยู่ชิดซ้ายของแกน X
- TC_DATUM หรือ 1 หมายถึง อยู่ชิดด้านบนของแกน Y และอยู่กึ่งกลางของแกน X
- TR_DATUM หรือ 2 หมายถึง อยู่ชิดด้านบนของแกน Y และอยู่ชิดด้านขวาของแกน X
- ML_DATUM หรือ 3 หมายถึง อยู่กึ่งกลางของแกน Y และอยู่ชิดซ้ายของแกน X
- MC_DATUM หรือ 4 หมายถึง อยู่กึ่งกลางของแกน Y และอยู่กึ่งกลางของแกน X
- MR_DATUM หรือ 5 หมายถึง อยู่กึ่งกลางแกน Y และอยู่ชิดด้านขวาของแกน X
- BL_DATUM หรือ 6 หมายถึง อยู่ด้านล่างของแกน Y และอยู่ชิดซ้ายของแกน X
- BC_DATUM หรือ 7 หมายถึง อยู่ด้านล่างของแกน Y และอยู่กึ่งกลางของแกน X
- BR_DATUM หรือ 8 หมายถึง อยู่ด้านล่างแกน Y และอยู่ชิดด้านขวาของแกน X



ภาพโดย จารุต บุศราทิจ

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 20-1 เป็นการเปลี่ยนสีจอภาพตามค่านำเข้าจากโมดูล ADC โดยเมื่อค่าที่รับจากโมดูล ADC มีการเปลี่ยนแปลงถึงช่วงค่าที่กำหนดจะแสดงสีเขียว สีเหลือง สีส้ม และสีแดง ตามลำดับจากค่าแอนะล็อกที่ได้รับจากโมดูล ADC มากไปน้อยโดยมีการตรวจสอบค่าแอนะล็อกและปรับเปลี่ยนการแสดงผลในบรรทัดที่ 7-15

ตัวอย่างโปรแกรม 20-1 เปลี่ยนสีจอภาพตามค่านำเข้าจากโมดูล ADC	
บรรทัด	คำสั่ง
1	#include <TFT_eSPI.h>
2	#define ADC1 34
3	int adc;
4	TFT_eSPI tft = TFT_eSPI();
5	void setup() {
6	tft.begin();
7	}
8	void loop(){

ตัวอย่างโปรแกรม 20-1 เปลี่ยนสีจอภาพตามค่านำเข้าจากโมดูล ADC (ต่อ)

บรรทัด	คำสั่ง
9	adc = analogRead(ADC1);
10	if(adc > 3000){
11	tft.fillScreen(TFT_RED);
12	}else if(adc > 2000){
13	tft.fillScreen(TFT_ORANGE);
14	}else if(adc > 1000){
15	tft.fillScreen(TFT_YELLOW);
16	}else{
17	tft.fillScreen(TFT_GREEN);
18	}
19	}

ตัวอย่างโปรแกรม 20-2 แสดงเมนูตัวเลือกโดยรับค่าเมนูจากโมดูลปุ่มกดและแสดงหน้าตามเมนูที่เลือก โดยสร้างฟังก์ชันสำหรับหน้าแรกที่มีการตั้งค่าสีและพื้นหลังของตัวอักษร ขนาดของตัวอักษร กำหนดตำแหน่งที่แสดงตัวอักษรในบรรทัดที่ 7-17 ฟังก์ชันสำหรับเมนูที่ program ในบรรทัดที่ 18-25 ฟังก์ชันสำหรับเมนู credit ที่มีการปรับสีและพื้นหลังของตัวอักษรสลับกันเพื่อให้เกิดการกระพริบในบรรทัดที่ 26-38 ทำการรอร์ับสัญญาณจากปุ่มกดเพื่อปรับค่าสำหรับเรียกฟังก์ชันตามเมนูในบรรทัดที่ 49-63 และแสดงเมนูตามค่าที่กำหนดจากปุ่มกดในบรรทัดที่ 64-70 มีการต่อขา SCL SDA RES DC CS และ BLK ของโมดูล TFT LCD เข้ากับขา 18 23 13 14 4 25 ของ ESP32 ตามลำดับ

ตัวอย่างโปรแกรม 20-2 แสดงเมนูตัวเลือก

บรรทัด	คำสั่ง
1	#include <TFT_eSPI.h>
2	#define button1 16
3	#define button2 17
4	#define button3 21
5	TFT_eSPI tft = TFT_eSPI();
6	int choice = 3;
7	void home(){
8	tft.setTextColor(TFT_WHITE, TFT_BLUE);
9	tft.setTextSize(3);
10	tft.setCursor(85,30);
11	tft.println("Menu");

ตัวอย่างโปรแกรม 20-2 แสดงเมนูตัวเลือก (ต่อ)

บรรทัด	คำสั่ง
12	tft.setTextSize(2);
13	tft.setCursor(60, 100);
14	tft.println("1. Program");
15	tft.setCursor(60, 120);
16	tft Credit");
17	}
18	void program(){
19	tft.setTextSize(2);
20	tft.setTextColor(TFT_WHITE, TFT_BLUE);
21	tft.setCursor(45, 100);
22	tft.println("Hello World!!");
23	tft.setCursor(25, 120);
24	tft.println("3. Back to Home");
25	}
26	void credit(){
27	tft.setTextSize(2);
28	tft.setTextColor(TFT_BLUE, TFT_WHITE);
29	tft.setCursor(90, 100);
30	tft.println("ETT!");
31	tft.setCursor(25, 120);
32	tft.println("3. Back to Home");
33	delay(100);
34	tft.setTextColor(TFT_WHITE, TFT_BLUE);
35	tft.setCursor(90, 100);
36	tft.println("ETT!");
37	delay(100);
38	}
39	void setup(void){
40	Serial.begin(115200);
41	pinMode(button1, INPUT);
42	pinMode(button2, INPUT);
43	pinMode(button2, INPUT);
44	tft.begin();
45	tft.fillScreen(TFT_BLUE);

ตัวอย่างโปรแกรม 20-2 แสดงเมนูตัวเลือก (ต่อ)

บรรทัด	คำสั่ง
46	home();
47	}
48	void loop(){
49	if(!digitalRead(button1)){
50	tft.fillScreen(TFT_BLUE);
51	choice = 1;
52	delay(300);
53	}
54	if(!digitalRead(button2)){
55	tft.fillScreen(TFT_BLUE);
56	choice = 2;
57	delay(300);
58	}
59	if(!digitalRead(button3)){
60	tft.fillScreen(TFT_BLUE);
61	choice = 3;
62	delay(300);
63	}
64	if(choice == 1){
65	program();
66	}else if(choice == 2){
67	credit();
68	}else{
69	home();
70	}
71	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-TFT LCD เป็นโมดูลแสดงผลสีแบบ RGB565 ด้วยความละเอียด 240x240 จุดสี หรือ พิกเซลในลักษณะของวงกลม ใช้ไลบรารี TFT_eSPI สามารถแสดงผลได้ทั้งลักษณะภาพและตัวอักษร การใช้คำสั่งแสดงผลเป็นการวาดภาพทับลงในตำแหน่งที่กำหนด โดยไม่ส่งผลกระทบต่อตำแหน่งอื่น ๆ เหมาะสำหรับการแสดงผลข้อมูลในระบบเนื่องจากการทำงานปกติของระบบสมองกลฝังตัวจะไม่มีโมดูลแสดงผลติดตั้งมาให้

บรรณานุกรม

GalaxyCore. (2019). *GC9A01A Datasheet*. สืบค้นเมื่อ พฤษภาคม 17, 2567, จาก

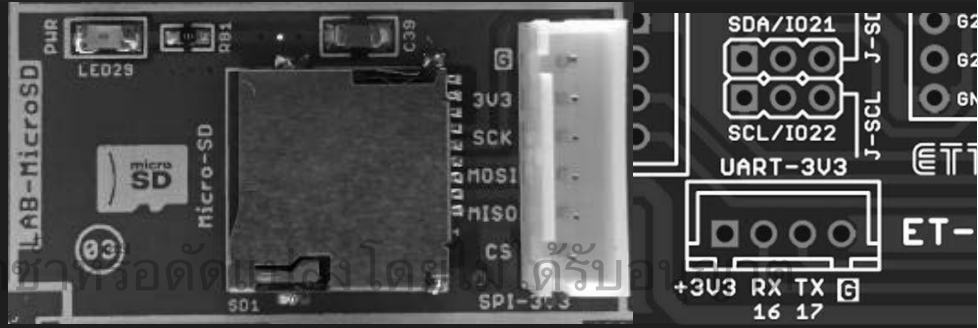
<https://files.waveshare.com/upload/5/5e/GC9A01A.pdf>

Bodmer. (2024). *TFT_eSPI*. สืบค้นเมื่อ พฤษภาคม 17, 2567, จาก

https://github.com/Bodmer/TFT_eSPI



บทที่ 21 LAB-MicroSD



โมดูล LAB-MicroSD เป็นโมดูลสำหรับเชื่อมต่อกับหน่วยความจำภายนอกประเภท microSD ผ่านทางช่องอ่านดังภาพที่ 21-1 ผ่านทางบัส SPI เพื่อเพิ่มช่องทางในการจัดเก็บข้อมูลหรือนำเข้าข้อมูลผ่านไฟล์ในการ์ดแบบ microSD ทำให้สามารถประยุกต์สร้างระบบเก็บล็อกข้อมูล (Data logger) เช่น จัดเก็บข้อมูลอุณหภูมิ ความชื้น เป็นต้น โดยข้อดีของการบันทึกลงการ์ดจะทำให้สามารถเก็บข้อมูลได้แม้ไม่มีไฟเลี้ยงบอร์ดไมโครคอนโทรลเลอร์ และสะดวกต่อการถ่ายโอนข้อมูลมากกว่าการจัดเก็บลงในหน่วยความจำแฟลชภายใน ESP32

เมื่อจบบทนี้ ผู้อ่านสามารถเขียนโปรแกรมเพื่อใช้งานการ์ด microSD ผ่านโมดูล LAB-MicroSD เพื่อทำการตรวจสอบสถานะของการ์ดในช่องเสียบ การสร้างโฟลเดอร์เพื่อเก็บข้อมูล และตัวอย่างการเก็บข้อมูลอุณหภูมิในทุก 5 วินาที

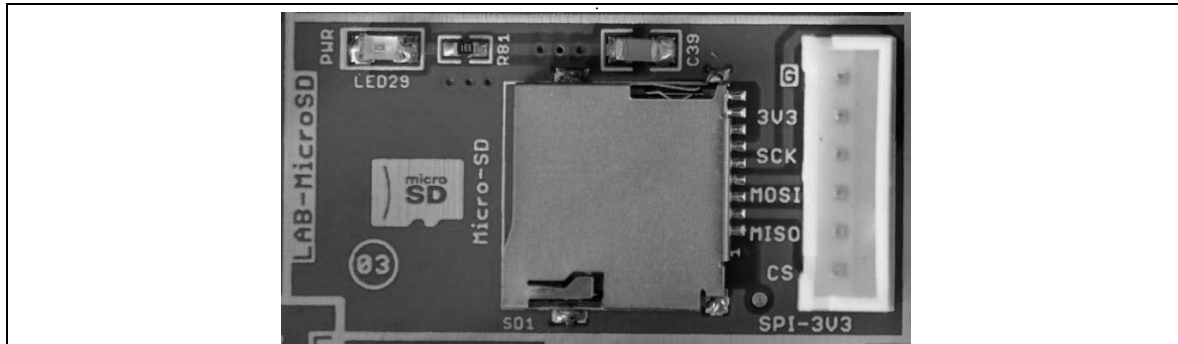
> แนะนำ LAB-MicroSD

ภาค LAB-MicroSD เป็นโมดูลสำหรับเชื่อมประสานกับการ์ดหน่วยความจำภายนอกประเภท microSD ดังภาพที่ 21-2 ผ่านทางบัส SPI แบบขั้วต่อ 6 ขาดังภาพที่ 21-1 ส่วนด้านหลังของการ์ด microSD มีขั้วทองแดงจำนวน 8 ขั้วสำหรับทำหน้าที่ดังนี้ (เรียงจากซ้ายไปขวา)

- DAT2 เป็นขาที่ไม่ได้ใช้
- CD/DAT3 สำหรับทำหน้าที่ขา CS ของบัส SPI
- CMD สำหรับทำหน้าที่ DI หรือ MOSI ของบัส SPI
- VDD สำหรับต่อกับแหล่งจ่ายไฟแรงดันกระแสตรง 3V3
- CLK สำหรับต่อกับขา SCLK ของบัส SPI

- VSS สำหรับต่อเข้ากับกราวด์
- DAT0 สำหรับทำหน้าที่ DO หรือ MISO ของบัล SPI
- DAT1 เป็นขาที่ไม่ได้ใช้

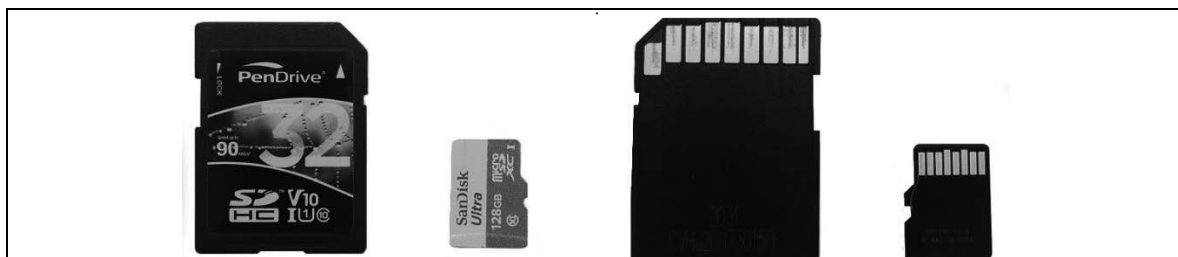
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



ภาพที่ 21-1

โมดูล LAB-MicroSD

ภาพโดย จารุต บุศราทิจ



ภาพที่ 21-2

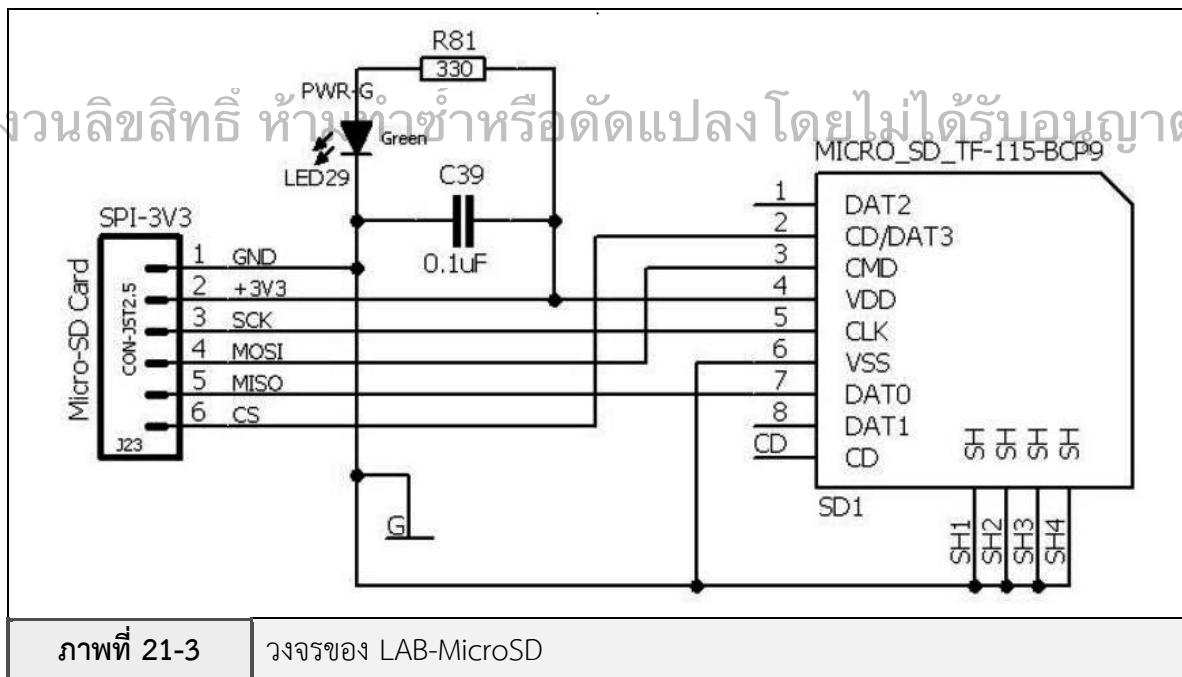
ภาพด้านหน้าและหลังของการ์ด SD และ microSD

ภาพโดย จารุต บุศราทิจ

> วงจรและการทำงาน

วงจรของ LAB-MicroSD เป็นดังภาพที่ 21-3 จะพบว่ามี LED29 สีเขียวบอกสถานะของโมดูลอ่านการ์ด microSD และทำการเชื่อมต่อกับขาของ ESP32 ดังนี้

- ขา CS เชื่อมต่อกับ P5
- ขา MOSI เชื่อมต่อกับ P23
- SCLK เชื่อมต่อกับ P18
- MISO เชื่อมต่อกับ P19



ภาพที่ 21-3

วงจรของ LAB-MicroSD

ภาพโดย จารุต บุศราทิจ

การใช้งานโมดูลเชื่อมประสานกับการ์ด microSD ต้องอาศัยไลบรารี 3 ตัว คือ SPI, FS และ SD โดย SPI เป็นไลบรารีแกนหลักสำหรับเชื่อมประสานกับบัส SPI ตัวไลบรารี FS จัดการเกี่ยวกับไฟล์และไดเรกทอรี ส่วน SD เป็นไลบรารีเชื่อมต่อกับโมดูล LAB-MicroSD เพื่อตรวจสอบสถานะของการเสียบการ์ด ประเภทของการ์ด ขนาดการ์ด และอื่น ๆ ดังจะกล่าวในรายละเอียดของไลบรารี

> ไลบรารี SD

ไลบรารีสำหรับเชื่อมประสานกับการ์ด SD หรือ microSD มี 2 ตัว คือ SD และ SD_MMC โดยตัวแรกเป็นไลบรารีสำหรับการเชื่อมประสานผ่านบัส SPI และแบบหลังเป็นการเชื่อมต่อกับบัส SDMMC ของไมโครคอนโทรลเลอร์ ESP32 โดยในบทนี้ใช้การเชื่อมประสานผ่านบัส SPI จึงเลือกใช้ไลบรารี SD

บัสแบบ SPI ที่ได้กล่าวถึงในบทที่ 1 เป็นการเชื่อมต่อด้วยการใช้สายสัญญาณ 4 เส้น ใช้ขา CS แบบควบคุมด้วยซอฟต์แวร์ และอีก 2 เส้นสำหรับแหล่งจ่ายไฟและกราวด์ พร้อมทั้งมีความเร็วสูงสุดในการสื่อสารอยู่ที่ 80MHz

ไลบรารี SD ถูกพัฒนาโดยบริษัท Espressif ผู้พัฒนาไมโครคอนโทรลเลอร์ ESP32 โดยเป็นคลาสที่สืบทอดมาจากไลบรารี FS อีกทอดหนึ่งทำให้คำสั่งไลบรารีตัวนี้ครอบคลุมการเข้าถึงไฟล์และไดเรกทอรีดังที่ได้กล่าวไว้ก่อนหน้านี้ และสามารถควบคุมหรืออ่านข้อมูลเกี่ยวกับการ์ด microSD ดังคำสั่งต่อไปนี้

- สถานะการเชื่อมต่อของการ์ด = `SD.begin()`
เป็นคำสั่งเริ่มต้นเชื่อมต่อกับโมดูลการ์ด microSD โดยผลลัพธ์ที่ได้จากคำสั่งเป็น
สถานะการเชื่อมต่อของการ์ด จะเป็น true ถ้าพบว่าการ์ด microSD ได้ถูกนำใส่เข้าตัว
อ่านการ์ด และเป็น false เมื่อไลบรารีไม่พบการ์ด microSD หรือไม่พบโมดูล LAB-
MicroSD
- `SD.end()`
เป็นคำสั่งปิดการเชื่อมต่อกับไลบรารี SD
- ประเภทของการ์ด = `SD.cardType()`
เป็นคำสั่งอ่านประเภทของการ์ด โดยมีค่าคืนกลับเป็นค่าคงที่ของประเภทการ์ดได้แก่
CARD_MMC, CARD_SD และ CARD_SDHC
- ขนาดที่ของการ์ด = `SD.cardSize()`
เป็นคำสั่งอ่านความจุของการ์ด microSD เป็นหน่วยจำนวนไบต์
- จำนวนเซกเตอร์ = `SD.numSectors()`
เป็นคำสั่งอ่านจำนวนเซกเตอร์ของการ์ด microSD
- ขนาดของเซกเตอร์ = `SD.sectorSize()`
เป็นคำสั่งอ่านขนาดของเซกเตอร์
- จำนวนไบต์ทั้งหมด = `SD.totalBytes()`
เป็นคำสั่งอ่านจำนวนไบต์ที่เขียนลงการ์ด microSD ได้
- จำนวนไบต์ที่ถูกใช้ไป = `SD.usedBytes()`
เป็นคำสั่งจำนวนไบต์ของการ์ด microSD ที่ถูกใช้งานไปแล้ว

> ไลบรารี FS

ไลบรารี FS ถูกพัฒนาขึ้นโดย Ivan Grokhotkov ตั้งแต่ ค.ศ. 2015 เพื่อใช้สำหรับบริหารจัดการ
กับระบบไฟล์ที่เก็บในหน่วยความจำแบบ SD-Card ที่พัฒนาต่อเนื่องมาจากแกนไลบรารีของไมโคร
คอนโทรลเลอร์ ESP8266 นั้นหมายความว่า ความสามารถของไลบรารีนี้ประกอบด้วย

- การจัดการกับไฟล์ เช่น การอ่าน การเขียน การเข้าถึง การย้ายตำแหน่งตัวชี้ในไฟล์ การ
ตรวจสอบสถานะของไฟล์ ขนาดของไฟล์ การเปิดและปิดไฟล์ การสร้าง การลบ การเปลี่ยน
ชื่อไฟล์ การย้ายตำแหน่งไฟล์ เป็นต้น

- การจัดการกับไดเรกทอรี เช่น การตรวจสอบว่าเป็นไดเรกทอรีหรือไม่ การสร้าง การตรวจสอบว่ามีไดเรกทอรีอยู่หรือไม่ การอ่านพาร การอ่านชื่อ การเข้าถึงไดเรกทอรี การเปลี่ยนชื่อ การลบ เป็นต้น

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

หมายเหตุ

ไฟล์ คือ ที่เก็บข้อมูลแบบข้อความหรือแบบไบนารี และเรียกว่าไบนารีไฟล์ (Binary File) และเท็กซ์ไฟล์ (Text File) ตามลำดับ

ไดเรกทอรี (โฟลเดอร์) คือ ที่เก็บไฟล์หรือไดเรกทอรี

พาร (path) เป็นชื่อเรียกของการเข้าถึงที่อยู่ของไฟล์หรือไดเรกทอรี โดยเรียงลำดับตั้งแต่ไดเรกทอรีราก (root directory) ที่แทนด้วยเครื่องหมาย / และตามด้วยไดเรกทอรีชั้นแรก และตามด้วยไดเรกทอรีชั้นต่อ ๆ มา โดยแต่ละชื่อจะคั่นด้วยเครื่องหมาย / เช่น พารชื่อ /ett/product/esp32 หมายความว่า จากไดเรกทอรีรากจะมีไดเรกทอรีชื่อ ett หลังจากนั้นเมื่อเข้าไปดูใน ett จะพบไดเรกทอรี product และในไดเรกทอรี product จะพบกับไดเรกทอรีชื่อ esp32 เป็นต้น

การใช้ต้องนำเข้าไฟล์ส่วนหัวชื่อ FS.h ด้วยคำสั่งต่อไปนี้ และใช้งานระบบไฟล์ด้วยการเข้าถึงวัตถุชื่อ fs::FS โดยในที่นี้คือ SD หรือการ์ด microSD

```
#include <FS.h>
```

การใช้งานไฟล์จะต้องสร้างวัตถุประเภท File ดังต่อไปนี้ โดยข้อมูลประเภท File นี้อาจจะเป็นไฟล์ข้อมูลหรือไดเรกทอรี

File วัตถุไฟล์;

ชุดคำสั่งสำหรับบริหารจัดการไฟล์และไดเรกทอรีมีดังต่อไปนี้

- File **SD.open**(const char* path, const char* mode = FILE_READ)
เป็นคำสั่งสำหรับเชื่อมต่อกับไฟล์ โดยโหมดของการเชื่อมต่อได้แก่ FILE_READ, FILE_WRITE และ FILE_APPEND สำหรับการอ่าน การเขียนใหม่ลงไฟล์หรือเป็นการเขียนทับทำให้ข้อมูลในไฟล์เดิมหายไปหรือเป็นการเขียนเพื่อสร้างไฟล์ใหม่ และการเพิ่มข้อมูลเข้าไปในไฟล์ที่มีอยู่แล้ว
- สถานะการทำงาน = **SD.exists**(พารของไดเรกทอรี)
เป็นคำสั่งสำหรับตรวจสอบชื่อ พารของไดเรกทอรี ในการ์ด microSD โดยคืนค่า

สถานะการทำงาน เป็น true ถ้าพบ พารของไดเรกทอรี และเป็น false เมื่อไม่พบ

พารของไดเรกทอรี

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

- สถานะการทำงาน = SD.remove(พารของไดเรกทอรี)

เป็นคำสั่งสำหรับลบไฟล์จาก พารของไดเรกทอรี ออกจากการ์ด microSD โดยคืนค่า

สถานะการทำงาน เป็น true ถ้าลบไฟล์สำเร็จ และเป็น false เมื่อล้มเหลว

- สถานะการทำงาน = SD.rename(พารของไดเรกทอรี_{ต้นทาง}, พารของไดเรกทอรี_{ปลายทาง});

เป็นคำสั่งสำหรับเปลี่ยนชื่อของไดเรกทอรีหรือไฟล์จาก พารของไดเรกทอรี_{ต้นทาง} เป็น

พารของไดเรกทอรี_{ปลายทาง} โดยคืนค่า สถานะการทำงาน เป็น true ถ้าเปลี่ยนชื่อ

ไดเรกทอรีหรือไฟล์สำเร็จ และเป็น false เมื่อล้มเหลว

- สถานะการทำงาน = SD.mkdir(พารของไดเรกทอรี)

เป็นคำสั่งสร้างไดเรกทอรีตามชื่อที่กำหนดใน พารของไดเรกทอรี และคืนค่า

สถานะการทำงาน เป็น true ถ้าสร้างไดเรกทอรีสำเร็จ และเป็น false เมื่อล้มเหลว

- สถานะการทำงาน = SD.rmdir(พารของไดเรกทอรี)

เป็นคำสั่งสำหรับลบ พารของไดเรกทอรี ออกจากการ์ด microSD โดยสถานะการ

ทำงาน เป็น true ถ้าการลบประสบความสำเร็จ และเป็น false เมื่อล้มเหลว

ชุดคำสั่งสำหรับใช้งานเกี่ยวกับไฟล์เป็นดังนี้

- พารปัจจุบัน = fs.path()

เป็นคำสั่งสำหรับอ่านชื่อพารปัจจุบัน

- ชื่อของไฟล์ = วัตถุไฟล์.name()

เป็นคำสั่งอ่านชื่อไฟล์ของวัตถุไฟล์

- สถานะของไฟล์ = วัตถุไฟล์.isDirectory()

เป็นคำสั่งตรวจสอบสถานะของไฟล์ว่าเป็นไดเรกทอรีหรือไฟล์ โดยถ้าค่าสถานะของไฟล์

เป็น true หมายถึง เป็นไดเรกทอรี และเป็น false เมื่อเป็นไฟล์

- ขนาดที่เขียนได้ = วัตถุไฟล์.write(ข้อมูล)

สำหรับเขียนข้อมูลขนาด 8 บิตลงระบบไฟล์

- ขนาดที่เขียนได้ = วัตถุไฟล์.write(บัฟเฟอร์, n)

สำหรับเขียน บัฟเฟอร์ ลงไฟล์เป็นจำนวน n ไบต์ และคืนค่าจำนวนไบต์ที่บันทึกได้

กลับมา

- สถานะของไฟล์ = วัตถุไฟล์.available()

ตรวจสอบว่าไฟล์ที่เปิดนั้นมีอยู่ก่อนแล้วหรือไม่ ถ้ามีจะคืนค่า สถานะของไฟล์ เป็น true แต่ถ้าไม่มีจะคืนเป็น false

- ข้อมูลที่อ่านได้ = วัตถุไฟล์.read()

เป็นคำสั่งอ่านข้อมูลจากไฟล์

- วัตถุไฟล์.flush()

สำหรับส่งบันทึกข้อมูลในบัฟเฟอร์ของไลบรารีลงไฟล์

- ขนาดที่อ่านได้ = วัตถุไฟล์.read(บัฟเฟอร์, n);

เป็นคำสั่งอ่านข้อมูลจากไฟล์จำนวน n ไบต์ไปเก็บในตัวแปร บัฟเฟอร์ จองไว้ในแบบ uint8_t พร้อมทั้งคืนค่ากลับเป็นขนาดที่อ่านได้ โดยถ้าขนาดที่อ่านได้ กับ n มีขนาดเท่ากัน หมายความว่า การอ่านครบถ้วนตามที่สั่งงาน

- ขนาดที่อ่านได้ = วัตถุไฟล์.readBytes(บัฟเฟอร์, n)

เป็นคำสั่งอ่านข้อมูลจากไฟล์จำนวน n ไบต์ไปเก็บในตัวแปร บัฟเฟอร์ จองไว้ในแบบ char พร้อมทั้งคืนค่ากลับเป็นขนาดที่อ่านได้ โดยถ้า ขนาดที่อ่านได้ กับ n มีขนาดเท่ากัน หมายความว่า การอ่านครบถ้วนตามที่สั่งงาน

- สถานะ = วัตถุไฟล์.seek(n)

เป็นคำสั่งย้ายตำแหน่งการชี้ข้อมูลในไฟล์ไป n ไบต์ และคืนค่า สถานะ เป็น 1 ถ้าสำเร็จ และเป็น 0 เมื่อล้มเหลว

- ตำแหน่งปัจจุบัน = วัตถุไฟล์.position()

เป็นคำสั่งคืนค่า ตำแหน่งปัจจุบัน ในไฟล์กลับมา

- ขนาดของไฟล์ = วัตถุไฟล์.size()

เป็นคำสั่งสำหรับอ่านขนาดของไฟล์

- สถานะ = วัตถุไฟล์.setBufferSize(n)

เป็นคำสั่งสำหรับปรับเปลี่ยนขนาดบัฟเฟอร์ในหน่วยความจำที่ถูกใช้เป็นที่เก็บข้อมูลชั่วคราวเพื่อนำไปเขียนลงไฟล์ให้มีขนาดเป็น n ไบต์

- วัตถุไฟล์.close()

เป็นคำสั่งสั่งปิดไฟล์

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

> ตัวอย่างโปรแกรม

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรม 21-1 เป็นการตรวจสอบการคงอยู่ของการ์ด microSD ที่เสียบเข้ากับโมดูล LAB-MicroSD ถ้าพบการ์ดจะดำเนินการอ่านรายละเอียดประเภทของการ์ด ความจุของการ์ด จำนวนไบต์ที่สามารถเขียนลงการ์ดได้ และปริมาณพื้นที่ที่ถูกใช้งานไป โดยตัวอย่างผลลัพธ์ของการทำงานเป็นดังภาพที่ 21-4

ตัวอย่างโปรแกรม 21-1 อ่านสถานะและความจุของการ์ด microSD

บรรทัด	คำสั่ง
1	#include <FS.h>
2	#include <SPI.h>
3	#include <SD.h>
4	void setup() {
5	Serial.begin(115200);
6	while (!Serial);
7	if (!SD.begin()) {
8	Serial.println("Error : Card mount failed!");
9	while (1);
10	}
11	show_card_info();
12	}
13	void show_card_info() {
14	uint8_t card_type = SD.cardType();
15	uint32_t mb_base = (1024*1024);
16	if (card_type == CARD_NONE) {
17	Serial.println("Error: Card not attatched!");
18	while (1);
19	}
20	Serial.print("Found : ");
21	switch (card_type) {
22	case CARD_MMC:
23	Serial.print("MMC (");
24	break;
25	case CARD_SD:
26	Serial.print("SD (");

ตัวอย่างโปรแกรม 21-1 อ่านสถานะและความจุของการ์ด microSD (ต่อ)

บรรทัด	คำสั่ง
27	break;
28	case CARD_SDHC:
29	Serial.print("SDHC (");
30	break;
31	default:
32	Serial.print("Unkown! (");
33	}
34	Serial.print(SD.cardSize()/mb_base);
35	Serial.println("MB.");
36	Serial.print("Usage : ");
37	Serial.print(SD.usedBytes()/mb_base);
38	Serial.print("/");
39	Serial.print(SD.totalBytes()/mb_base);
40	Serial.println("MB.");
41	}
42	void loop() {
43	}

ภาพที่ 21-4	ตัวอย่างผลลัพธ์การทำงานของตัวอย่างโปรแกรม 21-1

ภาพโดย จารุต์ บุศราทิจ

ตัวอย่างโปรแกรม 21-2 เป็นการอ่านข้อมูลเซ็นเซอร์อุณหภูมิภายในไมโครคอนโทรลเลอร์ ESP32 เก็บลงไฟล์ชื่อ ett_log.txt ทุก 5 วินาที หลังจากนั้นทำการอ่านข้อมูลจากไฟล์มาแสดงผล โดยในการจัดเก็บจะใช้การเขียนแบบ FILE_APPEND ยกเว้นการเปิดเพื่อเขียนเพิ่มเติมไม่สำเร็จจะทำการเปลี่ยนการเปิดไฟล์เป็นแบบ FILE_WRITE เพื่อสร้างไฟล์ขึ้นมา ส่วนในกระบวนการอ่านต้องเปลี่ยนโหมดของการเปิดไฟล์เป็น FILE_READ สุดท้ายทำการลบไฟล์ที่สร้างขึ้นทิ้งไป ตัวอย่างผลลัพธ์ของการทำงานเป็นดังภาพที่ 21-5

สิ่งสำคัญที่ต้องระมัดระวัง คือ การเขียนข้อมูลลงไฟล์ใช้คำสั่ง write() เพื่อเขียนไบนารีของข้อมูลลงไฟล์ แต่ถ้าใช้คำสั่ง print() หรือ println() จะส่งผลให้มีกรแปลงไบนารีข้อมูลให้กลายเป็นข้อความและเขียนข้อความลงนไฟล์ ทำให้การเขียนโปรแกรมอ่านค่าจะต้องอ่านอีกชะข้อความมาเรียงใหม่เพื่อแปลงให้เป็นตัวเลขอีกครั้งหนึ่ง

ตัวอย่างโปรแกรม 21-2 บันทึกและอ่านอุณหภูมิกับการ์ด microSD

บรรทัด	คำสั่ง
1	#include <FS.h>
2	#include <SPI.h>
3	#include <SD.h>
4	extern "C" {
5	uint8_t temprature_sens_read();
6	}
7	void setup() {
8	Serial.begin(115200);
9	while (!Serial);
10	if (!SD.begin()) {
11	Serial.println("Error : Card mount failed!");
12	while (1);
13	}
14	}
15	void loop() {
16	for (int i=0; i<5; i++) {
17	uint8_t temp_c=(temprature_sens_read()
18	-32)/1.8;
19	File o_file = SD.open("/ett_log.txt",
20	FILE_APPEND);
21	if (!o_file) {
22	o_file = SD.open("/ett_log.txt",
23	FILE_WRITE);
24	}
25	o_file.write(temp_c);
26	o_file.close();
27	delay(5000);
28	}
29	File i_file = SD.open("/ett_log.txt", FILE_READ);

ตัวอย่างโปรแกรม 21-2 บันทึกและอ่านอุณหภูมิกับการ์ด microSD (ต่อ)

บรรทัด	คำสั่ง
30	int i=0;
31	while (i_file.available()) {
32	uint8_t data = i_file.read();
33	Serial.print(++i);
34	Serial.print("=");
35	Serial.print(data);
36	Serial.println("C");
37	}
38	i_file.close();
39	SD.remove("/ett_log.txt");
40	while(1);
41	}

	1=32C 2=33C 3=32C 4=33C 5=32C	
ภาพที่ 21-5	ตัวอย่างผลลัพธ์การทำงานของตัวอย่างโปรแกรม 21-2	

ภาพโดย จารุต บุศราทิจ

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-MicroSD เป็นโมดูลสำหรับเชื่อมประสานกับการ์ดหน่วยความจำภายนอกประเภท microSD ผ่านบัส SPI ใช้ไลบรารี SD และ FS เพื่อเป็นทางเลือกในการเก็บข้อมูลสำรองในกรณีที่ไม่สามารถใช้งานคลาวด์ได้ โดยการเก็บไฟล์เป็นไปตามมาตรฐานของระบบการเก็บไฟล์ของระบบปฏิบัติการทั่วไป ฟังก์ชันการทำงานมีการคืนค่าสถานะในการทำงาน ผู้ใช้ควรเรียนรู้คำสั่งพื้นฐานในการจัดการไฟล์ก่อนใช้งาน

บรรณานุกรม

Ivan Grokhotkov. (2015). *FS.h*. สืบค้นเมื่อ พฤษภาคม 20, 2567, จาก

<https://github.com/espressif/arduino-esp32/blob/master/libraries/FS/src/FS.h>

Espressif System. (2016). *SD.h*. สืบค้นเมื่อ พฤษภาคม 20, 2567, จาก

<https://github.com/espressif/arduino-esp32/blob/master/libraries/SD/src/SD.h>

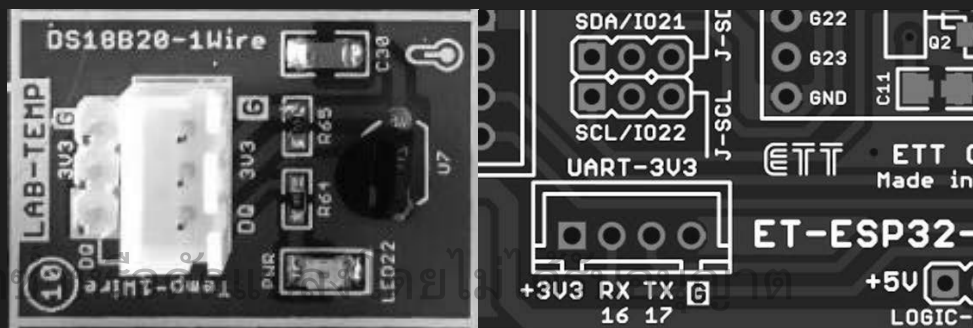
Espressif System. (2024). *SD SPI Host Driver*. สืบค้นเมื่อ พฤษภาคม 20, 2567, จาก

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/sdspi_host.html

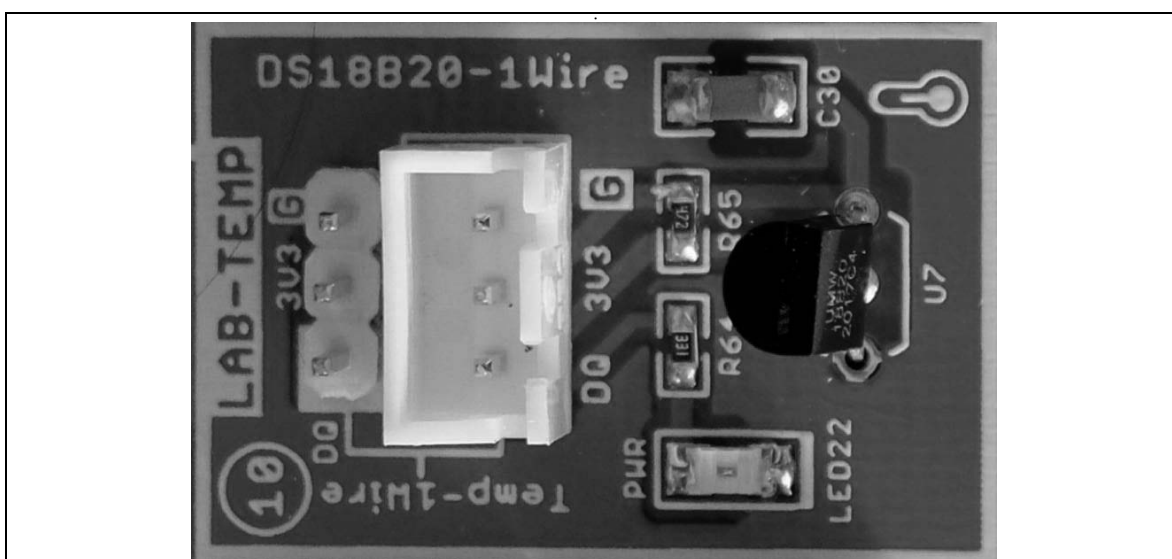


สงวนลิขสิทธิ์ ห้ามทำ

บทที่ 22 LAB-TEMP



ในบทนี้กล่าวถึงโมดูล LAB-TEMP ดังภาพที่ 22-1 อันเป็นการนำเข้าข้อมูลอุณหภูมิจากไอซีขนาดเล็กเบอร์ DS18B20 ของบริษัท Analog Device ที่มีหลักการสื่อสารกับบอร์ดไมโครคอนโทรลเลอร์ผ่านบัสแบบ 1-Wire® หรือใช้สายสัญญาณสื่อสารระหว่างกันเพียงเส้นเดียวแบบ Half-Duplex หรือผลัดกันทำงานระหว่างผู้ส่งและผู้รับ นั้นหมายความว่า ถ้าไมโครคอนโทรลเลอร์ ESP32 กำลังสั่งงาน ตัวเซ็นเซอร์จะเป็นผู้รับคำสั่ง และเมื่อเซ็นเซอร์รับคำสั่งเสร็จสิ้นจะทำการอ่านและแปลงค่าอุณหภูมิเป็นข้อมูลดิจิทัล และเริ่มทำหน้าที่เป็นผู้ส่งข้อมูลให้ไมโครคอนโทรลเลอร์ ด้วยเหตุนี้ ระหว่างที่เซ็นเซอร์ส่งออกข้อมูล ทางฝั่งไมโครคอนโทรลเลอร์จะต้องเป็นผู้รับข้อมูล แต่ถ้าต่างฝ่ายต่างส่งข้อมูลหรือรอรับข้อมูลทั้งคู่จะเกิดเหตุการณ์ไม่สามารถดำเนินการใด ๆ ได้เนื่องจากข้อมูลชนกันจากต่างฝ่ายต่างส่ง และไม่ได้รับข้อมูลใด ๆ เนื่องจากต่างฝ่ายต่างรอข้อมูลจากอีกฝ่าย



ภาพที่ 22-1

ภาพโมดูล LAB-TEMP

ภาพโดย จารุต บุศราทิจ

ในบทนี้จะกล่าวถึงคุณสมบัติของ DS18B20 พร้อมการใช้คำสั่งเพื่อติดต่อสื่อสารกับเซ็นเซอร์ และตัวอย่างโปรแกรมเพื่อเป็นแนวทางสำหรับการนำไปใช้งานต่อไป

เมื่อจบบทนี้ ผู้อ่านสามารถนำวงจรการทำงานของ LAB-TEMP ไปประยุกต์ใช้งานสำหรับวัดอุณหภูมิที่ใช้ขาเชื่อมประสานเพียง 1 ขา และราคาประหยัดเพื่อลดต้นทุนของการพัฒนาระบบตรวจสอบอุณหภูมิ เช่น ความเย็นภายในตู้แช่เย็น โรงเพาะเห็ด หรือ วัดอุณหภูมิภายในน้ำผ่านทางเซ็นเซอร์ที่ออกแบบให้ทนทานต่อน้ำ เป็นต้น

> แนะนำ LAB-TEMP

ภาค LAB-TEMP จากภาพที่ 22-1 เป็นโมดูลสำหรับทดลองการนำเข้าข้อมูลอุณหภูมิจากเซ็นเซอร์ที่ทำงานด้วยไอซี DS18B20 ผ่านทางการสื่อสารในรูปแบบ 1-Wire โดยไอซี DS18B20 เป็นโมดูลวัดอุณหภูมิอันมีคุณสมบัติดังนี้

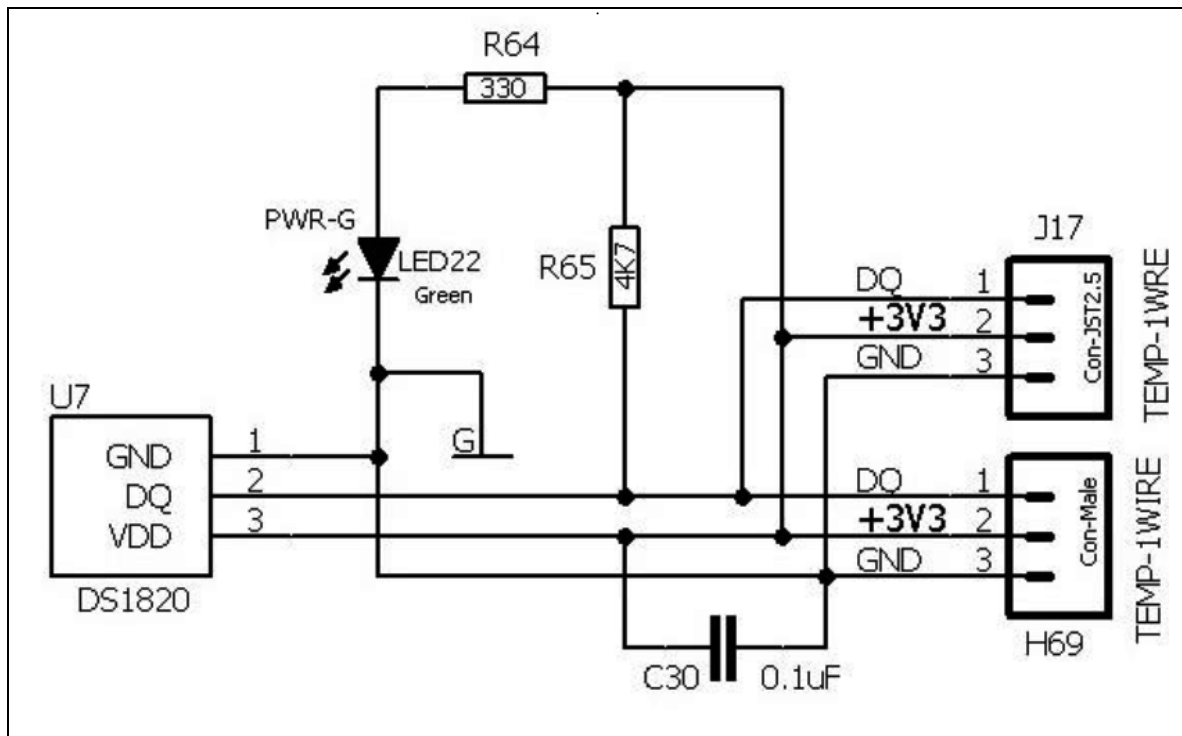
- รองรับการวัดอุณหภูมิในช่วง -55°C ถึง 125°C หรือ -67°F ถึง 257°F
- ในช่วงอุณหภูมิ -10°C ถึง 85°C มีความผิดพลาดหรือค่าคลื่อนในการวัด $\pm 0.5^{\circ}\text{C}$
- ในช่วงอุณหภูมิ -30°C ถึง 100°C มีความผิดพลาดหรือค่าคลื่อนในการวัด $\pm 1^{\circ}\text{C}$
- ในช่วงอุณหภูมิ -55°C ถึง 125°C มีความผิดพลาดหรือค่าคลื่อนในการวัด $\pm 2^{\circ}\text{C}$
- มีรหัสไอซีเป็นตัวเลขขนาด 64 บิต ทำให้สามารถใช้ไอซี DS18B20 หลายตัวในบัสเดียวกันได้
- ตัวถังแบบ TO-92 มีขา 3 ขาสำหรับทำหน้าที่ GND, DQ และ V_{DD} เพื่อต่อกับกราวด์ ต่อกับบัสสื่อสาร และต่อกับแหล่งจ่ายไฟกระแสตรงในช่วง $+3\text{V}$ ถึง $+5\text{V}$

จากภาพที่ 22-1 จะพบว่า บนโมดูล LAB-TEMP มีขั้วขนาด 3 ขาจำนวน 2 ชุดสำหรับใช้เชื่อมต่อกับไมโครคอนโทรลเลอร์ ESP32 คือ DQ สำหรับเชื่อมต่อกับสื่อสารระหว่างเซ็นเซอร์กับไมโครคอนโทรลเลอร์ G สำหรับต่อสายกราวด์ และ 3V สำหรับต่อกับแรงดันไฟฟ้ากระแสตรง 3V3 โดยหลอดแอลอีดี LED22 เป็นไฟแสดงสถานะโมดูล

> วงจรและการทำงาน

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

วงจรของภาค LAB-TEMP เป็นดังภาพที่ 22-2 จะพบว่า J17 และ H69 เป็นขั้วต่อสายเชื่อมต่อแบบ 3 ขาที่แตกต่างกันตามประเภทของหัวเชื่อมต่อ โดยขาที่ 1 ของทั้ง 2 ขั้วต่อเป็นขา DQ ขาที่ 2 เป็น +3V3 สำหรับต่อเข้ากับแรงดันไฟฟ้ากระแสตรง 3V3 และขาที่ 3 เป็นขากาวด์



ภาพที่ 22-2 วงจรของ LAB-TEMP

ภาพโดย จารุต บุศราทิจ

เมื่อมีการเชื่อมต่อขา +3V3 กับ GND อย่างถูกต้องจะทำให้แรงดันไฟฟ้าผ่าน LED22 และลง G ของวงจรที่ต่อร่วมกับขา 1 หรือ GND ของ DS18B20 ส่งผลให้หลอดแอลอีดีสีเขียวสว่าง ทำให้ทราบว่า การต่อวงจรถูกต้อง และแรงดันนี้ถูกส่งเข้าขา 3 หรือ VDD ของ DS18B20 ส่วนขา 2 หรือ DQ ของ DS18B20 ถูกเชื่อมต่อเข้ากับขาที่ 1 ของ J17 และ H69 เพื่อทำไปต่อเข้ากับขาของ ESP32 โดยในบทนี้ทำการเชื่อมเข้ากับขา P0 เพื่อใช้สื่อสารกับเซ็นเซอร์

การเขียนโปรแกรมเพื่อใช้งานโมดูล LAB-TEMP จะต้องเรียกใช้ไลบรารี 2 ตัว คือ OneWire กับ DallasTemperature เพื่อสร้างตัวแปรสำหรับอ้างอิงถึงการสื่อสารผ่านบัสแบบ 1-Wire และตัวแปรสำหรับอ้างอิงถึงเซ็นเซอร์วัดอุณหภูมิ DS18B20

> ไลบรารี OneWire.h

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

การสื่อสารตามรูปแบบสื่อสารผ่านบัสแบบ 1-Wire® ถูกออกแบบโดยบริษัทดัลลัสเซมิคอนดักเตอร์ (Dallas Semiconductor) ปัจจุบันเป็นบริษัท Maxim ที่ทำงานแบบบัสความเร็ว 16.3 kbps โดยมีหลักการทำงานคล้ายกับบัสแบบ I²C ที่กล่าวถึงในบทที่ 1 แต่ไม่มีสายสัญญาณ SCL สำหรับสร้างสัญญาณนาฬิกาในร่วมกันในการสื่อสาร

การใช้ไลบรารีจะต้องนำเข้าไฟล์ส่วนหัวชื่อ OneWire.h และคำสั่งสำหรับกำหนดขาสื่อสารระหว่างไมโครคอนโทรลเลอร์และเซ็นเซอร์ LAB-TEMP คือ

OneWire ตัวแปรสื่อสาร(หมายเลขขา)

เนื่องจากใช้ P0 ในการเชื่อมต่อกับ LAB-TEMP ทำให้คำสั่งที่ได้ คือ

```
#include <OneWire.h>
OneWire ds( 0 );
```

> ไลบรารี DallasTemperature.h

สำหรับการเชื่อมต่อกับเซ็นเซอร์จะต้องใช้ไลบรารี DallasTemperature ที่พัฒนาโดย Miles Burton และต้องติดตั้งเพิ่ม พร้อมทั้งเรียกใช้ไฟล์ส่วนหัวดังนี้

```
#include <DallasTemperature.h>
```

การเชื่อมเซ็นเซอร์เข้ากับบัสสื่อสารต้องใช้คำสั่งดังนี้

DallasTemperature ตัวแปรเซ็นเซอร์(&ตัวแปรสื่อสาร)

ตามด้วยการเปิดการทำงานด้วยคำสั่ง begin() ดังรูปแบบของคำสั่งต่อไปนี้

```
ตัวแปรเซ็นเซอร์.begin()
```

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

เมื่อต้องการอ่านค่าอุณหภูมิจากเซ็นเซอร์จะต้องดำเนินการ 2 ขั้นตอน คือ ร้องขอเข้าถึงเซ็นเซอร์ ด้วยคำสั่ง `requestTemperatures()` และอ่านค่าอุณหภูมิแบบอนาล็อกจากสาย 1-Wire ด้วยคำสั่ง `getTempCByIndex()` ดังนี้

```
ตัวแปรเซ็นเซอร์.requestTemperatures()
```

```
ค่าอุณหภูมิ = ตัวแปรเซ็นเซอร์.getTempCByIndex( ลำดับของเซ็นเซอร์ )
```

> ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรม 22-1 เป็นตัวอย่างการรับค่าอุณหภูมิจากโมดูล DS18B20 และแสดงไอคอนที่โมดูลแสดงผล OLED ตามเงื่อนไขที่กำหนดโดยตรวจสอบค่าอุณหภูมิที่ได้รับ หากอุณหภูมิเย็นแสดงไอคอนหิมะ หากอุณหภูมิปกติแสดงไอคอนหน้าปกติ หากอุณหภูมิร้อนแสดงไอคอนโกรธ โดยมีการกำหนดไอคอนต่างๆ ในบรรทัดที่ 12-51 ทำการรับค่าอุณหภูมิจากโมดูล DS18B20 ในบรรทัดที่ 65-66 ตรวจสอบค่าอุณหภูมิและแสดงไอคอนตามที่กำหนดในบรรทัดที่ 74-80 โดยมีการต่อขา SDA และ SCL ของโมดูล OLED เข้ากับขา 21 และ 22 ของ ESP32 ตามลำดับ และต่อขา DQ ของโมดูล DS18B20 เข้ากับขา P0 ของ ESP32

ตัวอย่างโปรแกรม 22-1 รับค่าอุณหภูมิจากโมดูล DS18B20 และแสดงไอคอนตามเงื่อนไขที่กำหนด

บรรทัด	คำสั่ง
1	<code>#include <OneWire.h></code>
2	<code>#include <DallasTemperature.h></code>
3	<code>#include <Adafruit_GFX.h></code>
4	<code>#include <Adafruit_SH1106.h></code>
5	<code>#define SDA 21</code>
6	<code>#define SCL 22</code>
7	<code>#define tempPin 0</code>

ตัวอย่างโปรแกรม 22-1 รับค่าอุณหภูมิจากโมดูล DS12B20 และแสดงไอคอนตามเงื่อนไขที่กำหนด (ต่อ)

บรรทัด	คำสั่ง
8	Adafruit_SH1106 display(SDA, SCL);
9	OneWire oneWire(tempPin);
10	DallasTemperature DS18B20(&oneWire);
11	float tempC;
12	static const unsigned char PROGMEM degree[] = {
13	0b00000010,
14	0b00000101,
15	0b00000010,
16	0b00000000,
17	0b00000000,
18	0b00000000,
19	0b00000000,
20	0b00000000
21	};
22	static const unsigned char PROGMEM smile[] = {
23	0b01100110,
24	0b10011001,
25	0b00000000,
26	0b00000000,
27	0b10000001,
28	0b01000010,
29	0b00111100,
30	0b00000000
31	};
32	static const unsigned char PROGMEM normal[] = {
33	0b00000000,
34	0b11100111,
35	0b00000000,
36	0b00000000,
37	0b00000000,
38	0b01111110,
39	0b00000000,
40	0b00000000

ตัวอย่างโปรแกรม 22-1 รับค่าอุณหภูมิจากโมดูล DS12B20 และแสดงไอคอนตามเงื่อนไขที่กำหนด (ต่อ)

บรรทัด	คำสั่ง
41	};
42	static const unsigned char PROGMEM angry[] = {
43	0b10100101,
44	0b01000010,
45	0b10100101,
46	0b00000000,
47	0b00000000,
48	0b00111100,
49	0b01000010,
50	0b10000001
51	};
52	void setup () {
53	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
54	display.clearDisplay();
55	display.display();
56	delay(100);
57	display.setTextSize(1);
58	display.setTextColor(WHITE);
59	display.setCursor(0, 0);
60	DS18B20.begin();
61	}
62	void loop () {
63	display.clearDisplay();
64	display.setCursor(0, 0);
65	DS18B20.requestTemperatures();
66	tempC = DS18B20.getTempCByIndex(0);
67	display.println("");
68	display.print("Temp: ");
69	display.print(tempC, 2);
70	display.drawBitmap(display.getCursorX(),
71	display.getCursorY(), degree, 8, 8, WHITE);
72	display.setCursor(display.getCursorX() + 10,
73	display.getCursorY());

ตัวอย่างโปรแกรม 22-1 รับค่าอุณหภูมิจากโมดูล DS12B20 และแสดงไอคอนตามเงื่อนไขที่กำหนด (ต่อ)

บรรทัด	คำสั่ง
74	display.print("C");
75	if(tempC < 25){
76	display.drawBitmap(50, 36, smile, 8, 8, WHITE);
77	}
78	else if(tempC < 30) {
79	display.drawBitmap(50,36,normal, 8, 8, WHITE);
80	}
81	else{
82	display.drawBitmap(50, 36, angry, 8, 8, WHITE);
83	}
84	display.display();
85	delay(500);
86	delay(500);
87	}

ตัวอย่างโปรแกรม 22-2 รับค่าอุณหภูมิจากโมดูล DS18B20 และแปลงเป็นค่าหน่วยอื่นโดยรับค่าเป็นอุณหภูมิเซลเซียสและเปลี่ยนค่าเป็นฟาเรนไฮต์หรือเคลวินเมื่อกดปุ่ม button1 โดยมีการรอรับปุ่มกดเพื่อเปลี่ยนโหมดการแสดงผลในบรรทัดที่ 38-44 เนื่องจากการรับค่าจากโมดูลจำเป็นต้องเว้นช่วงเวลาแต่หากใช้การเว้นช่วงเวลาด้วยฟังก์ชัน delay(1000) อาจทำให้การกดปุ่มจำเป็นต้องรอให้การหน่วงเวลาเสร็จสิ้นด้วย อาจทำให้ต้องกดค้างเป็นเวลานาน จึงใช้ฟังก์ชันในการเรียกเวลาปัจจุบันคือ millis() เพื่อบันทึกเวลาและหาค่าต่างเวลาเพื่อให้โปรแกรมทำงานได้อย่างต่อเนื่องในบรรทัดที่ 12 และ 47 หากเวลาเกินช่วงหน่วงที่กำหนดคือ 1 วินาทีจึงทำการแสดงค่าตามโหมดการแสดงผลในบรรทัดที่ 48-65 มีการต่อขาเพิ่มเติมจากโปรแกรม 19-2 คือต่อขา SW0 ของโมดูลปุ่มกดเข้ากับขา P39 ของ ESP32

ตัวอย่างโปรแกรม 22-2 รับค่าอุณหภูมิจากโมดูล DS18B20 และแปลงเป็นค่าเป็นหน่วยอื่น

บรรทัด	คำสั่ง
1	#include <OneWire.h>
2	#include <DallasTemperature.h>
3	#include <Adafruit_GFX.h>
4	#include <Adafruit_SH1106.h>
5	#define SDA 21

ตัวอย่างโปรแกรม 22-2 รับค่าอุณหภูมิจากโมดูล DS18B20 และแปลงเป็นค่าเป็นหน่วยอื่น (ต่อ)

บรรทัด	คำสั่ง
6	#define SCL 22
7	#define button1 39
8	#define tempPin 0
9	Adafruit_SH1106 display(SDA, SCL);
10	OneWire oneWire(tempPin);
11	DallasTemperature DS18B20(&oneWire);
12	unsigned int now = millis();
13	float temp;
14	int mode = 1;
15	char degreeLetter = 'C';
16	static const unsigned char PROGMEM degree[] = {
17	0b00000010,
18	0b00000101,
19	0b00000010,
20	0b00000000,
21	0b00000000,
22	0b00000000,
23	0b00000000,
24	0b00000000
25	};
26	void setup(){
27	display.begin(SH1106_SWITCHCAPVCC, 0x3C);
28	pinMode(button1, INPUT);
29	display.clearDisplay();
30	display.display();
31	delay(100);
32	display.setTextSize(1);
33	display.setTextColor(WHITE);
34	display.setCursor(0, 0);
35	DS18B20.begin();
36	}
37	void loop(){
38	if(!digitalRead(button1)){
39	mode++;

ตัวอย่างโปรแกรม 22-2 รับค่าอุณหภูมิจากโมดูล DS18B20 และแปลงเป็นค่าเป็นหน่วยอื่น (ต่อ)

บรรทัด	คำสั่ง
40	if(mode > 3){
41	mode = 1;
42	}
43	delay(300);
44	}
45	display.clearDisplay();
46	display.setCursor(0, 0);
47	if(millis() - now > 1000){
48	DS18B20.requestTemperatures();
49	temp = DS18B20.getTempCByIndex(0);
50	now = millis();
51	display.print("Temp : ");
52	degreeLetter = 'C';
53	if(mode == 2){
54	temp = 9 / 5 * temp + 32;
55	degreeLetter = 'F';
56	}else if(mode == 3){
57	temp = temp + 273;
58	degreeLetter = 'K';
59	}
60	display.print(temp, 2);
61	display.drawBitmap(display.getCursorX(),
62	display.getCursorY(),degree,8,8, WHITE);
63	display.setCursor(display.getCursorX() +
64	10,display.getCursorY());
65	display.println(degreeLetter);
66	display.display();
67	}
68	}

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ภาค LAB-TEMP เป็นแนวทางหนึ่งของการอ่านค่าอุณหภูมิโดยใช้สายสัญญาณเพียงเส้นเดียวทำให้ประหยัดสายสัญญาณและมีราคาที่ถูกกว่าโมดูลอุณหภูมิและความชื้นแบบอื่น โดยหลักการเขียนโปรแกรมจะต้องใช้ไลบรารี 2 ตัว คือ ตัวอ้างอิงไปยังบัสแบบ 1-Wire ที่ต้องผูกเข้ากับขาของไมโครคอนโทรลเลอร์ และตัวไลบรารีสำหรับเซ็นเซอร์ DS18B20 ที่ต้องเชื่อมเข้ากับตัวอ้างอิงบัสอีกทอดหนึ่ง นอกจากนี้ ตัวโมดูล DS18B20 รองรับการใช้หลายตัวบนบัสตัวเดียวด้วยการอ้างอิงหมายเลขลำดับเป็นตัวอ้างอิงถึงเซ็นเซอร์แต่ละตัว

บรรณานุกรม

- Maxim integrated. (2019). *DS18B20 Datasheet*. สืบค้นเมื่อ พฤษภาคม 24, 2567, จาก <https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf>
- WIKIPEDIA. (2024). *1-Wire*. สืบค้นเมื่อ พฤษภาคม 24, 2567, จาก <https://en.wikipedia.org/wiki/1-Wire>
- Paul Stoffregen. (2024). *OneWire.h*. สืบค้นเมื่อ พฤษภาคม 24, 2567, จาก https://www.pjrc.com/teensy/td_libs_OneWire.html
- Miles Burton. (2020). *DallasTemperature.h*. สืบค้นเมื่อ พฤษภาคม 24, 2567, จาก <https://github.com/milesburton/Arduino-Temperature-Control-Library>



บทที่ 23 WiFi



บทนี้กล่าวถึงการใช้โมดูล WiFi ของไมโครคอนโทรลเลอร์ ESP32 เพื่อรู้จักหลักการทำงานของระบบเครือข่ายอินเทอร์เน็ต อินเทอร์เน็ตโพรโทคอลอย่างทีซีพี/ยูดีพี การตั้งค่าให้ ESP32 เป็นจุดให้บริการเพื่อให้ลูกข่ายใช้บริการหรือเชื่อมต่อ ESP32 เข้ากับจุดบริการที่มีอยู่แล้วเพื่อให้ ESP32 เข้าใช้เครือข่ายอินเทอร์เน็ต การทำงานเป็นเครื่องให้บริการเว็บ การรับคำสั่งผ่านเว็บ พร้อมทั้งแนะนำชุดคำสั่งที่เกี่ยวข้องเพื่อนำไปประยุกต์ใช้ต่อไป

นอกจากเรื่องของ WiFi แล้ว ในบทนี้ได้อธิบายการใช้งานคุณสมบัติ ESP Now ของไมโครคอนโทรลเลอร์ ESP32 ที่ทำงานแบบไร้สายแต่เป็นโพรโทคอลสื่อสารระหว่าง ESP32 ด้วยกันเอง ทำให้สามารถประยุกต์สร้างเครือข่ายเพื่อการสื่อสารระหว่างกันโดยไม่ต้องให้ทุกโหนดเชื่อมต่ออินเทอร์เน็ตตลอดเวลา เมื่อจบบทนี้ ผู้อ่านจะได้นำตัวอย่างการใช้งานของบทนี้ไปสร้างระบบ IoT หรือระบบสื่อสารไร้สายของตนเอง โดยใช้การแสดงผลเป็นหน้าเว็บ และนำเข้าข้อมูลผ่านทางเว็บแทนการส่งจรรยาเข้าข้อมูลและแสดงผลข้อมูลต่าง ๆ ทำให้ประหยัดต้นทุนในการพัฒนาระบบ

> แนะนำ WiFi

ESP32 เป็นอุปกรณ์ที่มีระบบการเชื่อมต่อสัญญาณ WiFi ตามมาตรฐาน IEEE 802.11 ในตัว ผ่านทางคลื่นวิทยุความถี่ 2.4GHz โดยสามารถทำงานได้ทั้งโหมดการให้ตนเองเป็นจุดให้เข้าใช้งาน (AP: Access Point) หรือโหมด **WIFI_AP** โหมดลูกข่ายที่เชื่อมต่อเข้ากับเครือข่าย WiFi ที่มีอยู่แล้ว หรือทำตัวเองเป็นลูกข่าย หรือสถานะของเครือข่าย (station) หรือโหมด **WIFI_STA** และโหมดที่เป็นทั้งลูกข่ายและเป็นผู้ให้บริการ หรือ **WIFI_AP_STA** ทำให้ตัวเองเชื่อมต่อเครือข่ายอินเทอร์เน็ตและใช้ตนเองเป็น

ทางผ่านให้กับลูกข่ายได้ด้วย หรือที่มักคุ้นกันในโหมด Hot-Spot ซึ่งตัวอย่างมาตรฐานของเครือข่ายไร้สาย ได้แก่

- 802.11a ที่ทำงานบนความถี่ 5GHz มีความเร็วในการสื่อสาร 54Mbps
- 802.11b ที่ทำงานบนความถี่ 2.4GHz มีความเร็วในการสื่อสาร 11Mbps
- 802.11g ที่ทำงานบนความถี่ 2.4GHz มีความเร็วในการสื่อสาร 54Mbps
- 802.11n ที่ทำงานบนความถี่ 2.4GHz หรือ 5GHz มีความเร็วในการสื่อสาร 248Mbps

> อินเทอร์เน็ตโพรโทคอล

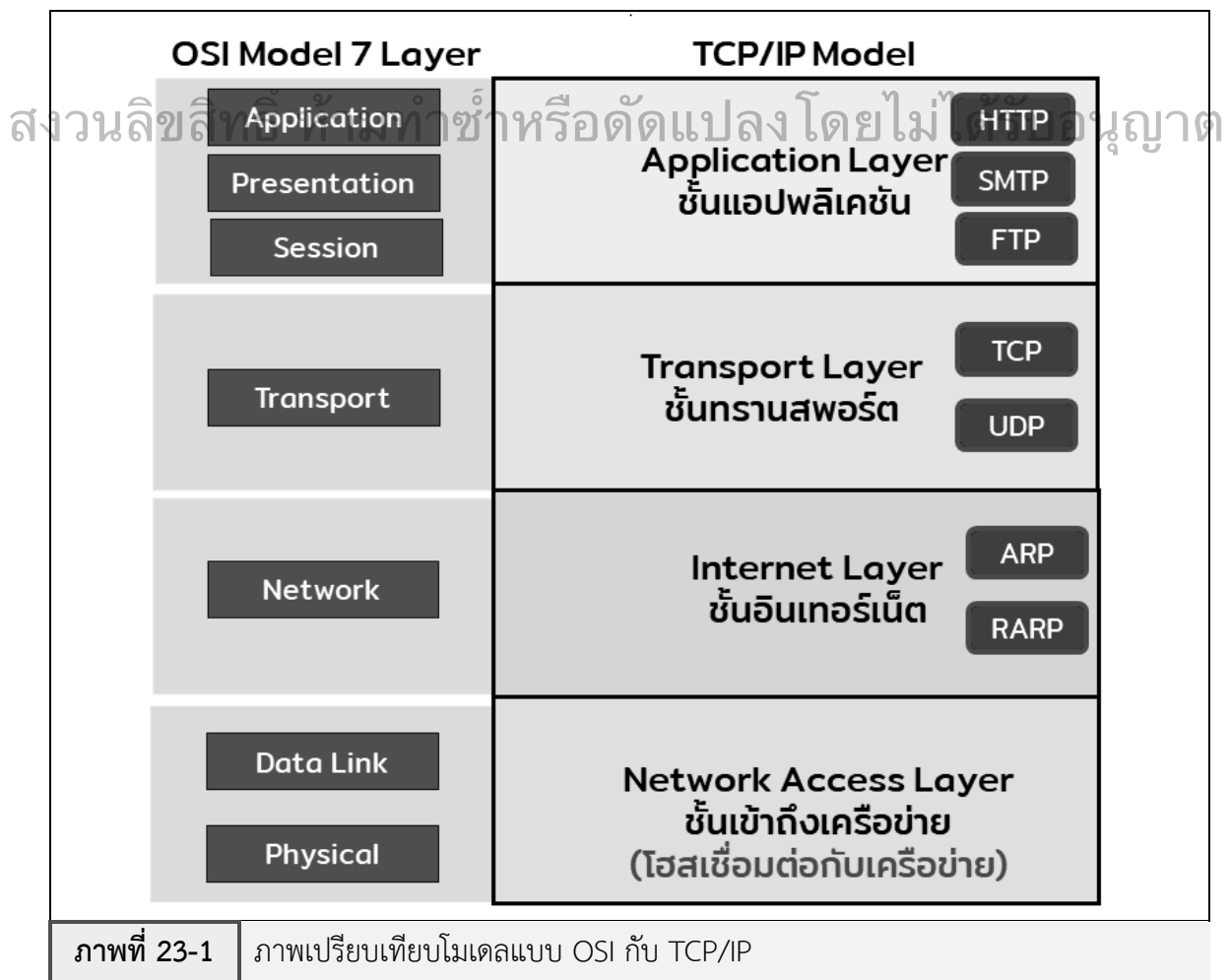
ทีซีพี/ไอพีหรือ TCP/IP (Transmission Control Protocol/Internet Protocol) เป็นโพรโทคอลที่เริ่มมาจากโครงการ DARPA (Defense Advanced Research Projects Agency) เพื่อใช้งานด้านการสื่อสารที่ต้องมีการรับประกันได้ว่าสามารถเชื่อมต่อได้ตลอดเวลาแม้เครือข่ายได้รับความเสียหาย โดยชั้นการทำงานของโพรโทคอลอินเทอร์เน็ตมี 4 ชั้น

- ชั้นที่ 1 หรือชั้นเข้าถึงเครือข่าย (Network Access Layer) ทำหน้าที่รับ/ส่งเฟรมข้อมูลและเข้าถึงการสื่อสารในระดับกายภาพ เช่น Ethernet FDDI X.25 Wireless และ ATM เป็นต้น
- ชั้นที่ 2 หรือชั้นอินเทอร์เน็ต (Internet Layer) ทำหน้าที่กำหนดตำแหน่งการรับ/ส่งโดยอินเทอร์เน็ตโพรโทคอล บริการที่ได้จากลำดับชั้นนี้ได้แก่ ICMP ARP และ RARP
- ชั้นที่ 3 หรือชั้นทรานสปอร์ต (Transport Layer) ทำหน้าที่ส่งผ่านและนำส่งด้วย TCP และ UDP แบบ Host-to-Host โดยบริการที่ได้จากชั้นนี้ได้แก่ ping และ trace route
- ชั้นที่ 4 หรือชั้นแอปพลิเคชัน (Application Layer) เป็นชั้นโปรแกรมประยุกต์ต่าง ๆ เช่น NCS MIME/SMTP telnet DNS http NFS/RPC และ SNMP เป็นต้น

เมื่อนำชั้นของการสื่อสารตามโมเดล TCP/IP มาเปรียบเทียบกับมาตรฐานของชั้นการสื่อสารแบบระบบเปิดหรือ OSI (Open Systems Interconnection) แบบ 7 ชั้น (Layers) เป็นดังภาพที่ 23-1 โดยโมเดลแบบ OSI แต่ละชั้นมีรายละเอียดโดยคร่าวดังนี้^[1]

- ชั้นที่ 1 หรือกายภาพ (Physical) ประกอบด้วย การสื่อสารในระดับกายภาพในระดับบิต เช่น Bluetooth, IEEE 802.3, DSL, SDH เป็นต้น ทำหน้าที่ส่งบิตออกไปยังอุปกรณ์สื่อสารเพื่อส่งผ่านไปยังอุปกรณ์ควบคุมการส่งข้อมูลภายนอก และนำบิตที่รับเข้ามาส่งต่อให้กับชั้นที่ 2

- สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต
- ชั้นที่ 2 หรือเดตาลิงก์ (Data Linked) ประกอบด้วย โพรโทคอลสื่อสาร IEEE 802.3, SLIP, PPP, ISDN, ARP และ ARC โดยใช้ในการรับส่งในระดับเฟรม (Frame) ทำหน้าที่แปลงเฟรมเป็นบิตส่งให้ชั้นที่ 1 และนำบิตรวมเป็นเฟรมเพื่อส่งต่อไปยังชั้นที่ 3
 - ชั้นที่ 3 หรือเครือข่าย หรือ เน็ตเวิร์ก (Network) ประกอบด้วย โพรโทคอล IP, NAT และ ICMP โดยทำการรับส่งในระดับแพ็กเก็ต (Packet) ทำหน้าที่นำแพ็กเก็ตแยกเป็นเฟรมส่งให้ชั้นที่ 2 และนำเฟรมจากชั้นที่ 2 มารวมเป็นแพ็กเก็ตเพื่อส่งต่อไปยังชั้นที่ 4
 - ชั้นที่ 4 หรือทรานสปอร์ต (Transport) ประกอบด้วย โพรโทคอล DCCP (Datagram Congestion Control Protocol), SCTP (Stream Control Transmission Protocol), TCP และ UDP โดยทำงานรับหรือส่งในระดับเซกเมนต์ (segment) หรือเดต้าแกรม (datagram) ทำหน้าที่แปลงเดต้าแกรมเป็นแพ็กเก็ตเพื่อส่งไปให้ชั้นที่ 3 และรวมแพ็กเก็ตจากชั้นที่ 3 ให้เป็นเดต้าแกรมเพื่อส่งต่อไปยังชั้นที่ 5
 - ชั้นที่ 5 หรือเซสชัน (Session) ประกอบด้วย โพรโทคอล NetBios, NFS (Network File Systems), RPC (Remote Procedure Call), SQL (Structured Query Language) เป็นต้น โดยในชั้นนี้มีรูปแบบการสื่อสารข้อมูลตามแต่ละโพรโทคอลใช้งาน และในชั้นนี้ทำการแปลงข้อมูลตามโพรโทคอลเป็นเดต้าแกรมเพื่อส่งไปให้ชั้นที่ 4 และนำเดต้าแกรมที่รับจากชั้นที่ 4 มารวมเป็นชุดข้อมูลตามโพรโทคอลที่ใช้งานเพื่อส่งต่อไปยังชั้นที่ 6
 - ชั้นที่ 6 หรือพรีเซนเตชัน (Presentation) ประกอบด้วย โพรโทคอล NDR และ LPP โดยการทำงานในชั้นนี้ทำหน้าที่เกี่ยวกับการเข้ารหัสหรือถอดรหัสข้อมูลตามโพรโทคอลต่าง ๆ เพื่อนำส่งข้อมูลที่เข้ารหัสไปยังชั้นที่ 5 และถอดรหัสข้อมูลจากชั้นที่ 5 ส่งต่อไปยังชั้นที่ 7
 - ชั้นที่ 7 หรือแอปพลิเคชัน (Application) ประกอบด้วย โปรแกรมประยุกต์ หรือบริการ (Service) ต่าง ๆ รวมถึงโพรโทคอลระดับโปรแกรมประยุกต์ เช่น HTTPS (Hypertext Transfer Protocol Secure), HTTP (Hypertext Transfer Protocol), DNS (Domain Name System), POP (Post Office Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), DHCP (Dynamic Host Configuration Protocol), MIME (Multipurpose Internet Mail Extensions), RTSP (Real-Time Streaming Protocol), TLS (Transport Layer Security Protocol), URL (Universe Resource Locator) และ Telnet (Telnet Remote Protocol) เป็นต้น โดยมีหน้าที่รับหรือส่งข้อมูลตามโพรโทคอลกับโปรแกรม/บริการกับการส่ง/รับข้อมูลไปยังชั้นที่ 6



ภาพโดย จารุต บุศราทิจ

หลักการทำงานของเครือข่ายอินเทอร์เน็ต คือ เป็นเครือข่ายที่ใช้ระบบการสื่อสารแบบแบ่งปันช่องสัญญาณโดยมองการสื่อสารแต่ละการเชื่อมต่อเป็นเซสชันและรองรับการทำงานหลายเซสชันพร้อมกัน ซึ่งข้อมูลที่สื่อสารในแต่ละเซสชันถูกแบ่งเป็นแพ็กเก็ตแต่ละแพ็กเก็ตถูกกำหนดตำแหน่งที่อยู่ของผู้ส่งและผู้รับ หลังจากนั้นนำส่งไปในระบบเครือข่ายในรูปแบบของการส่งผ่าน (forward) ไปยังพอร์ตที่ดีที่สุด โดยอาศัยอุปกรณ์เราเตอร์ (router) เป็นตัวเปลี่ยนเส้นทางการนำส่งแพ็กเก็ตเพื่อให้ไปสู่ปลายทางได้เร็วขึ้น เมื่อแพ็กเก็ตถูกส่งถึงปลายทาง เครื่องรับปลายทางทำหน้าที่รวบรวมแพ็กเก็ตย่อยที่ได้รับที่ส่งจากต้นทางมาประกอบ (reassemble) เป็นแพ็กเก็ตที่สมบูรณ์ แล้วนำส่งขึ้นไปแต่ละชั้นของเครือข่ายต่อไป

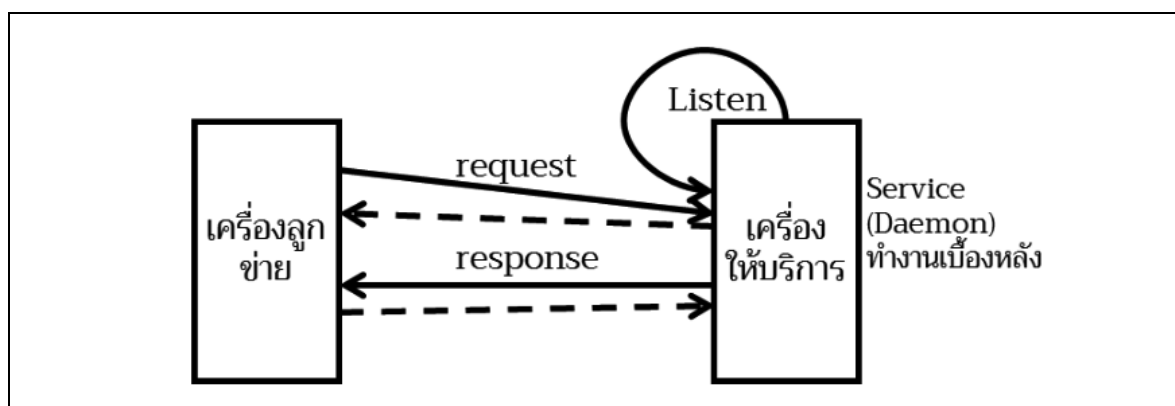
ข้อตกลงร่วมกันของอุปกรณ์บนเครือข่ายอินเทอร์เน็ตคือ ใช้ตำแหน่งไอพี หรือ IP Address เป็นค่าระบุโฮสต์ (Host) ในเครือข่าย โดยแต่ละโฮสต์มีค่าหมายเลขประจำตัวเรียกว่า MAC (Media Access Control) Address ที่ไม่ซ้ำกัน โดยขณะเวลาหนึ่งจะมีเพียง 1 MAC Address ที่ครอบครอง IP Address ไว้

ตัวอย่าง IP Address ของ IP v4 ประกอบด้วยเลขจำนวน 4 ชุดคั่นด้วยเครื่องหมาย . โดยเลขแต่ละชุดเป็นตัวเลขจำนวนเต็มที่มีค่าอยู่ในช่วง 0 ถึง 255 เช่น 12.10.2.1 192.168.1.2 หรือ 10.10.0.1 เป็นต้น นอกจากนี้มีการแบ่งระดับของ IP Address เป็น 3 คลาส คือ คลาส A, B และ C ซึ่งแต่ละคลาสมีจำนวนลูกข่ายที่แตกต่างกัน และใช้ซับเน็ตมาสก์ (Subnet Mask) เป็นตัวเลขแบ่งเครือข่ายเป็นเครือข่ายย่อย

การสื่อสารระหว่างโฮสต์ในเครือข่ายอินเทอร์เน็ตมีการสื่อสาร 2 วิธี เรียกว่า connection Oriented ที่รับรองการรับ/ส่งข้อมูลว่าถึงปลายทางเนื่องจากการให้เครื่องปลายทางตอบกลับการรับข้อมูลเสมอ กับ Connection-less ที่เน้นการรับส่งข้อมูลด้วยความเร็วสูง แต่แลกกับการไม่รับรองการส่งถึงเครื่องปลายทาง โดยการรับ/ส่งแบบ Connection-Oriented เรียกว่าการสื่อสารแบบ TCP (Transmission Control Protocol) ดังภาพที่ 23-2 และการรับ/ส่งแบบ Connection-less เรียกว่า UDP (User Datagram Protocol) ดังภาพที่ 23-3 โดยขั้นตอนการสื่อสารแบบ TCP ระหว่างโฮสต์ที่เป็นเครื่องให้บริการหรือเครื่องเซิร์ฟเวอร์ (Server) และไคลเอนต์ (Client) ประกอบด้วย 3 ขั้นตอน ทำให้มั่นใจได้ว่าข้อมูลส่งถึงกัน ดังนี้

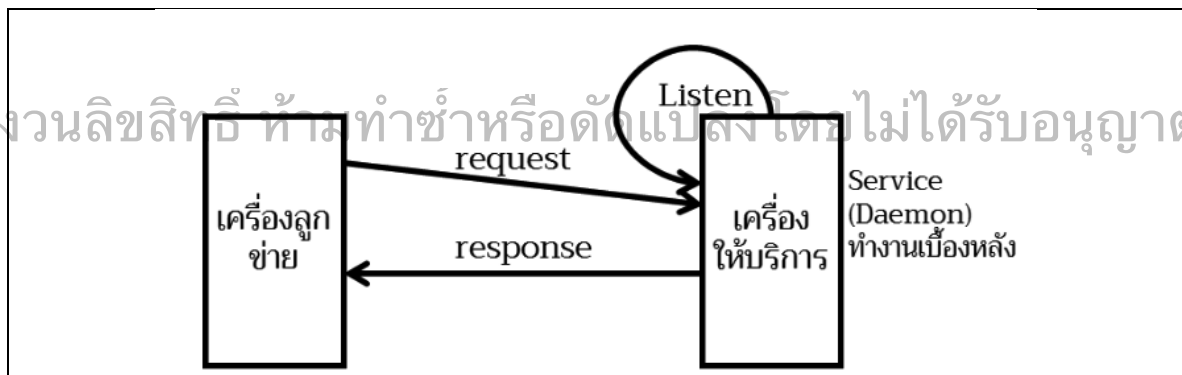
- ขั้นตอนที่ 1 ลูกข่ายส่ง SYN ไปยังเครื่องให้บริการ
- ขั้นตอนที่ 2 เครื่องให้บริการส่ง SYN-ACK ให้ลูกข่าย
- ขั้นตอนที่ 3 เครื่องลูกข่ายส่ง ACK ให้เครื่องบริการ

ในขั้นตอนของการสื่อสารระหว่างเครื่องต้นทางและปลายทางจะต้องอาศัยช่องสื่อสารหรือซ็อกเก็ต (socket) ดังภาพที่ 23-4 ทำหน้าที่เสมือนท่อของการรับ/ส่งข้อมูลระหว่างกัน โดยการขอเปิดใช้ซ็อกเก็ตต้องระบุหมายเลขของซ็อกเก็ตซึ่งเรียกว่าหมายเลขพอร์ต (port number) และตัวเลขหมายเลขพอร์ตนี้เป็นหมายเลขที่บริการบนโพรโทคอล-อินเทอร์เน็ตใช้ในการสื่อสาร เช่น โพรโทคอล HTTP ใช้พอร์ตหมายเลข 80 บริการ FTP ใช้หมายเลขพอร์ต 21 หรือ DNS ใช้หมายเลขพอร์ต 53 เป็นต้น โดยหมายเลขพอร์ตในการสื่อสารแบบ TCP หรือ UDP นั้นสามารถกำหนดหมายเลขเป็นตัวเลขจำนวนเต็มในช่วงค่า 0 ถึง 65535



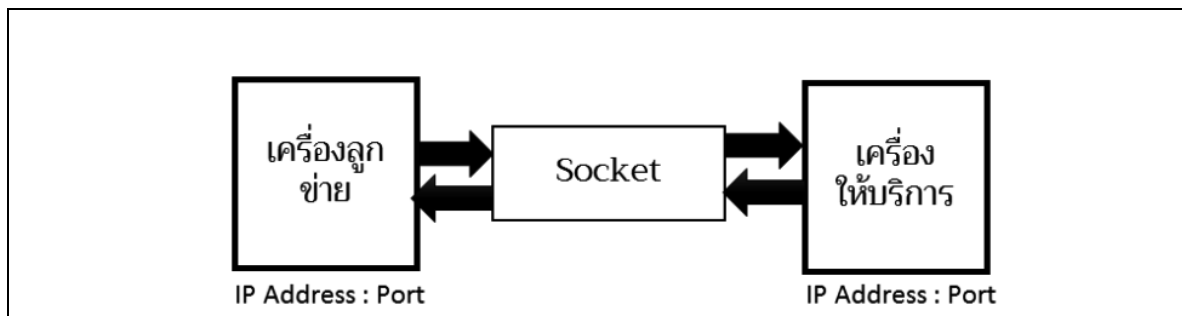
ภาพที่ 23-2 การสื่อสารแบบที่ซีพี

ภาพโดย จารุต์ บุศราทิจ



ภาพที่ 23-3 การสื่อสารแบบยูดีพี

ภาพโดย จารุต บุศราทิจ



ภาพที่ 23-4 ซ็อกเก็ต

ภาพโดย จารุต บุศราทิจ

> คำสั่งที่เกี่ยวข้องกับ WiFi

การเขียนโปรแกรมเพื่อเชื่อมต่อ WiFi ด้วย ESP32 จะต้องนำเข้าไฟล์ส่วนหัว WiFi.h และเตรียมข้อมูลของอุปกรณ์ AP ที่ต้องการเชื่อมต่อ หรือเตรียมตั้งชื่อ ประเภทการเข้ารหัส และรหัสผ่านสำหรับกรณีที่ต้องการให้ ESP เป็นผู้ให้เข้าถึงเครือข่ายอินเทอร์เน็ต ดังตัวอย่างต่อไปนี้

```
#include <WiFi.h>
char SSID[] = "ชื่อAPที่จะเชื่อมต่อหรือสร้าง";
char PASSWD[] = "รหัสผ่านของAP";
```

คำสั่งสำหรับเชื่อมต่อกับเครือข่ายที่มีอยู่แล้วจะใช้ begin() ตามด้วยพารามิเตอร์ชื่อของจุดเชื่อมต่อและรหัสผ่านสำหรับเข้าใช้งาน ดังรูปแบบของการใช้งานต่อไปนี้

```
void WiFi.begin()
```

```
void WiFi.begin(ชื่ออุปกรณ์ที่ต้องการเชื่อมต่อ, รหัสผ่าน)
```

```
void WiFi.begin(ชื่ออุปกรณ์ที่ต้องการเชื่อมต่อ, รหัสผ่าน,  
ช่องสัญญาณ, สถานะการซ่อน, จำนวนลูกข่ายสูงสุด)
```

โดยที่

ชื่ออุปกรณ์ คือ ชื่อ SSID (Service Set Identifier) ที่ต้องการให้ลูกข่ายหรือผู้เข้าใช้อ้างถึงเพื่อ
เข้าใช้

รหัสผ่าน คือ รหัสผ่านที่ตั้งโดยมีความยาวอย่างน้อย 8 ตัวอักษร

ช่องสัญญาณ คือ หมายเลขช่องสัญญาณไร้สาย มีค่าอยู่ในช่วง 1 ถึง 13

สถานะการซ่อน คือ การกำหนดให้แสดงชื่ออุปกรณ์หรือซ่อนชื่ออุปกรณ์ โดยกำหนดค่าเป็น 0
หมายถึง แสดงชื่อ (broadcast) และ 1 หมายถึง ซ่อน (hidden)

จำนวนลูกข่ายสูงสุด คือ จำนวนการเชื่อมต่อสูงสุดจากลูกข่ายซึ่งมีค่าเป็นตัวเลขในช่วง 1 ถึง 4

เมื่อทำการเชื่อมต่อจะต้องตรวจสอบสถานะของการเข้าถึงด้วยคำสั่ง status() เพื่อตรวจสอบค่า
คืนกลับว่าเป็น WL_CONNECTED หรือไม่ ถ้าไม่ใช่หมายถึงไม่สามารถเข้าถึงได้ ดังนั้น ต้องทำการวนรอบ
รอจนกว่าจะเชื่อมต่อสำเร็จ และเมื่อใช้งานเสร็จสิ้นต้องปิดการเชื่อมต่อด้วยคำสั่ง disconnect() เพื่อเป็น
การประหยัดพลังงาน หรือสั่งเชื่อมต่อใหม่อีกครั้งตามค่าการเชื่อมต่อเดิมด้วยคำสั่ง reconnect()
ดังรูปแบบของคำสั่งต่อไปนี้

```
ผลลัพธ์ = WiFi.status( )
```

```
void WiFi.disconnect( )
```

```
void WiFi.reconnect( )
```

โดยที่

ผลลัพธ์ จากคำสั่ง status() เป็นค่าตัวเลขที่หมายถึงค่าคงที่ต่อไปนี้

- 0 หรือ WL_IDLE_STATUS เป็นค่าเริ่มต้นเมื่อ begin() ถูกเรียกใช้
- 1 หรือ WL_NO_SSID_AVAIL เมื่อไม่พบชื่อจุดเชื่อมต่อที่ระบุ
- 2 หรือ WL_SCAN_COMPLETED เมื่อการสแกนเครือข่ายสิ้นสุด
- 3 หรือ WL_CONNECTED เมื่อการเชื่อมต่อสำเร็จ
- 4 หรือ WL_CONNECT_FAILED เมื่อการเชื่อมต่อล้มเหลว
- 5 หรือ WL_CONNECTION_LOST เมื่อสูญเสียการเชื่อมต่อ

- 6 หรือ WL_DISCONNECTED เมื่อปิดการเชื่อมต่อถ้าโมดูลไม่ได้ตั้งค่าโหมดทำงานอยู่ในโหมด STA

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ถ้าต้องการเพียงตรวจสอบการเชื่อมต่อสามารถใช้คำสั่ง `isConnected()` ทดแทนการใช้คำสั่งตรวจสอบสถานะ โดยผลลัพธ์ที่ได้จากคำสั่งจะเป็นค่า 0 หรือ 1 สำหรับแทนการไม่ได้เชื่อมต่อและเชื่อมต่อสำเร็จ ดังนี้

```
ผลลัพธ์ = WiFi.isConnected( )
```

กรณีที่ต้องการให้อุปกรณ์เชื่อมต่ออัตโนมัติเข้ากับ AP ตัวหลังสุดที่เข้าใช้งานต้องใช้คำสั่ง `setAutoConnect()` พร้อมด้วยพารามิเตอร์ การตั้งค่า ให้เป็น `true` หรือ `false` เพื่อเปิดหรือปิดการทำงานเชื่อมต่ออัตโนมัติดังนี้

```
void WiFi.setAutoConnect( การตั้งค่า )
```

การตรวจสอบว่าได้กำหนดตั้งค่าให้อุปกรณ์ทำการเชื่อมต่ออัตโนมัติเอาไว้หรือไม่สามารถใช้คำสั่ง `getAutoConnect()` เป็นคำสั่งสำหรับอ่านค่าสถานะนั้นกลับมา ซึ่งถ้า ผลลัพธ์ มีค่าเป็น `true` หมายถึงได้เปิดให้การเชื่อมต่ออัตโนมัติทำงาน และ `false` หมายถึงปิดการเชื่อมต่ออัตโนมัติไว้

```
ผลลัพธ์ = WiFi.getAutoConnect( )
```

กรณีที่ไดยกเลิกการเชื่อมต่อด้วย `disconnect()` และต้องการเชื่อมต่อใหม่อีกครั้งให้ใช้คำสั่ง `reconnect()` อีกครั้ง แต่ถ้าผู้เขียนโปรแกรมต้องการให้ไมโครคอนโทรลเลอร์ ESP32 ทำการเชื่อมต่ออีกครั้งโดยอัตโนมัติจะต้องเรียกใช้การตั้งค่าการทำงานนี้ผ่านทาง `setAutoReconnect()` ด้วยการกำหนดพารามิเตอร์ การตั้งค่า เป็น `true` และยกเลิกการทำงานอัตโนมัติด้วยการส่งอาร์กิวเมนต์หรือพารามิเตอร์เป็น `false` ตามรูปแบบของคำสั่งดังนี้

```
void WiFi.setAutoReconnect( การตั้งค่า )
```

คำสั่งสำหรับให้รอจนกว่า ESP32 จะถูกเชื่อมต่อกับ AP และคืนค่าสถานะ ผลลัพธ์ กลับมา คือ คำสั่ง `waitForConnectResult()` โดยค่าสถานะจะมีความหมายเหมือนกับสถานะของคำสั่ง `status()` ที่กล่าวมาก่อนหน้านี้

```
ผลลัพธ์ = WiFi.waitForConnectResult( )
```

ข้อกำหนดอย่างหนึ่งของการสื่อสารในเครือข่ายคอมพิวเตอร์ คือ อุปกรณ์เครือข่ายแต่ละชิ้นจะมีหมายเลขประจำตัวของตัวเองที่ไม่ซ้ำกันเพื่อใช้ในการระบุตัวตนของอุปกรณ์ โดยเรียกหมายเลขนี้ว่า MAC Address ซึ่งคำสั่ง `macAddress()` ต้องการพารามิเตอร์เป็นตัวแปรแถวลำดับขนาด 6 ไบต์สำหรับเก็บค่าตำแหน่งแต่ละหลักตามรูปแบบ `xx:xx:xx:xx:xx:xx` ดังนี้

```
UInt8_t ตัวแปร;
void WiFi.macAddress( ตัวแปร )
```

จากที่ได้กล่าวไว้ว่า ESP32 รองรับโหมดการทำงานของ WiFi ด้วยกัน 3 โหมด คือ WIFI_AP, WIFI_STA และ WIFI_AP_STA โดยใช้คำสั่ง `mode()` และกำหนดค่าคงที่ของโหมดใดโหมดหนึ่งเป็นพารามิเตอร์ของคำสั่งดังรูปแบบของการใช้งานดังนี้ ส่วนกรณีที่ต้องการปิดโหมดทำงานสามารถใช้ WIFI_OFF เป็นพารามิเตอร์แทนชื่อโหมด

```
void WiFi.mode( โหมด )
```

> ข้อมูลของการเชื่อมต่อเครือข่าย

กรณีที่เข้าใช้เครือข่ายไร้สายได้สำเร็จ ตัวเครื่องให้บริการจะให้หมายเลขไอพี (IP Address) แก่ อุปกรณ์เพื่อใช้แทนค่าที่อยู่ของอุปกรณ์ คำสั่งสำหรับอ่านค่าหมายเลขไอพีที่ได้จากอุปกรณ์เชื่อมต่ออินเทอร์เน็ต คือ `localIP()` ที่คืนค่าผลลัพธ์ออกมาในรูปแบบข้อมูลประเภท IPAddress มีรูปแบบของการใช้งานดังนี้

IPAddress ตัวแปร;

ตัวแปร = WiFi.localIP()

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

คลาส IPAddress เป็นคลาสสำหรับการจัดการกับโครงสร้างข้อมูลสำหรับ IP Address ด้วยเหตุนี้ ถ้ามีการสร้างวัตถุหรือตัวแปรสำหรับค่าตำแหน่งไอพี หรือ IP Address ให้ใช้คลาสนี้เป็นตัวจัดการ ตัวอย่างการสร้างตัวแปรสำหรับเก็บค่า IPAddress ของอุปกรณ์แต่ละชิ้นบนเครือข่ายอินเทอร์เน็ตที่ต้องประกอบไปด้วยค่าไอพีของตนเอง ค่าไอพีของเกตเวย์ (gateway) และค่าซับเน็ตมาสก์ สามารถเขียนได้ดังนี้

IPAddress ค่าตำแหน่งไอพี(xx, xx, xx, xx);

IPAddress ค่าตำแหน่งไอพีของเกตเวย์(xx, xx, xx, xx);

IPAddress ค่าซับเน็ตมาสก์(xx, xx, xx, xx);

คำสั่งกำหนดให้ ESP32 ทำงานเป็นผู้ให้บริการจุดเชื่อมต่อเพื่อเข้าถึงเครือข่ายอินเทอร์เน็ตหรือทำตนเอง AP จะใช้คำสั่ง softAP() ตามรูปแบบของการใช้งานต่อไปนี้

void WiFi.softAP(ชื่ออุปกรณ์, รหัสผ่าน)

void WiFi.softAP(ชื่ออุปกรณ์, รหัสผ่าน,
ช่องสัญญาณ, สถานะการซ่อน, จำนวนลูกข่ายสูงสุด)

กรณีที่ต้องการเปลี่ยนหรือกำหนดค่าไอพีของอุปกรณ์ใหม่จะต้องใช้คำสั่ง softAPConfig() ตามด้วยพารามิเตอร์ทั้ง 3 ตามที่สร้างไว้ด้านบน ดังรูปแบบของคำสั่ง คือ

ผลลัพธ์ = WiFi.softAPConfig(ค่าตำแหน่งไอพี,
ค่าตำแหน่งไอพีของเกตเวย์, ค่าซับเน็ตมาสก์)

การอ่านค่าตำแหน่งไอพีของ softAP ต้องใช้คำสั่ง softAPIP() ดังนี้

ค่าตำแหน่งไอพี = WiFi.softAPIP ()

ส่วนคำสั่งสำหรับอ่าน ค่าสับเน็ตมาสก์ และ ค่าตำแหน่งไอพีของเกตเวย์สามารถอ่านค่าได้ด้วย คำสั่ง `subnetMask()` และ `gatewayIP()` ตามรูปแบบการใช้งานต่อไปนี้

```
ค่าสับเน็ตมาสก์ = WiFi.subnetMask()  
ค่าตำแหน่งไอพีของเกตเวย์ = WiFi.gatewayIP()
```

และคำสั่ง `printDiag()` เป็นคำสั่งสำหรับแสดงรายการข้อมูลจาก WiFi ดังรายการต่อไปนี้ผ่านทางพอร์ตสื่อสารอนุกรม ทำให้ก่อนใช้คำสั่งนี้จะต้องเปิดการใช้พอร์ตสื่อสารอนุกรมก่อนเสมอ

- Mode
- PHY mode
- Channel
- AP id
- Status
- Auto connect
- SSID
- Passphase
- BSSID set

```
ผลลัพธ์ = WiFi.printDiag( คลาสสื่อสารอนุกรม )
```

ตัวอย่างโปรแกรม 23-1 เป็นการอ่านค่า MAC Address และผลลัพธ์จากคำสั่ง `Diag()` มาแสดงผลที่ Serial Monitor ดังตัวอย่างภาพที่ 23-5

ตัวอย่างโปรแกรม 23-1 แสดง MAC Address และ Diag

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	uint8_t myMACADDR[6];
3	void setup() {
4	Serial.begin(115200);
5	WiFi.begin();
6	delay(1000);
7	WiFi.setAutoConnect(true);
8	WiFi.waitForConnectResult();
9	WiFi.macAddress(myMACADDR);
10	Serial.print("MAC Address : ");
11	Serial.print(myMACADDR [0], HEX);
12	Serial.print(":");
13	Serial.print(myMACADDR [1], HEX);
14	Serial.print(":");
15	Serial.print(myMACADDR [2], HEX);
16	Serial.print(":");
17	Serial.print(myMACADDR [3], HEX);
18	Serial.print(":");
19	Serial.print(myMACADDR [4], HEX);
20	Serial.print(":");
21	Serial.print(myMACADDR [5], HEX);
22	Serial.println();
23	WiFi.printDiag(Serial);
24	}
25	void loop() {
26	}

	MAC Address : 84:CC:A8:7F:56:40 Mode: STA Channel: 1 SSID (0): Passphrase (0): BSSID set: 0
--	--

ภาพที่ 23-5 ตัวอย่างผลลัพธ์จากโปรแกรม 23-1

ภาพโดย จารุต บุศราทิจ

> การสแกนชื่อจุดเชื่อมต่อ

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

คำสั่งสำหรับสแกนหรือสร้างรายชื่อจุดเชื่อมต่อที่มองเห็น คือ `scanNetworks()` ซึ่งคำสั่งคืนค่าเป็นจำนวนจุดเชื่อมต่อที่มองเห็น และรายการชื่อ SSID, RSSI หรือความแรงของสัญญาณ วิธีการเข้ารหัสค่าแม็คแอดเดรสของอุปกรณ์ที่เชื่อมต่อหรือ BSSID และสตริงของ BSSID จะถูกเก็บเอาไว้ในหน่วยความจำของโมดูล WiFi ที่สามารถเข้าถึงได้จากคำสั่ง `SSID()`, `RSSI()`, `encryptionType()`, `BSSID()` และ `BSSIDStr()` ตามลำดับ โดยแต่ละคำสั่งต้องการพารามิเตอร์เป็นตัวเลขที่เป็นลำดับของจุดเชื่อมต่อและค่านี้จะอยู่ในช่วง 0 ถึง n-1 ซึ่งรูปแบบคำสั่งของทั้งหมดเป็นดังนี้

```
จำนวนจุดเชื่อมต่อที่พบ = WiFi.scanNetworks()
ชื่อจุดเชื่อมต่อ = WiFi.SSID( ลำดับ )
ความแรงสัญญาณของจุดเชื่อมต่อ = WiFi.RSSI( ลำดับ )
ประเภทการเข้ารหัสของจุดเชื่อมต่อ = WiFi.encryptionType ( ลำดับ )
แกลวลำดับตัวอักษรแม็คแอดเดรส = WiFi.BSSID( ลำดับ )
สตริงแม็คแอดเดรส = WiFi.BSSIDStr( ลำดับ )
```

โดยที่ ประเภทการเข้ารหัสของจุดเชื่อมต่อ จะเป็นค่าคงที่ตัวใดตัวหนึ่งจากรายการต่อไปนี้ ซึ่งเป็นค่าคงที่ประเภท `wifi_auth_mode_t`

- `WIFI_AUTH_OPEN`
- `WIFI_AUTH_WEP`
- `WIFI_AUTH_WPA_PSK`
- `WIFI_AUTH_WPA2_PSK`
- `WIFI_AUTH_WPA_WPA2_PSK`
- `WIFI_AUTH_WPA2_ENTERPRISE`

ตัวอย่างโปรแกรม 23-2 เป็นการสแกนรายชื่อ และแสดงรายละเอียดของจุดเชื่อมต่อเครือข่ายที่มองเห็น

ตัวอย่างโปรแกรม 23-2 แสดงรายชื่อจุดเชื่อมต่อพร้อมรายละเอียด

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	void showAPs (int networksFound) {
3	Serial.printf("Found %d AP(s)\n",
4	networksFound);
5	for (int i = 0; i < networksFound; i++) {
6	Serial.print("AP No.");
7	Serial.print(i + 1);
8	Serial.print(" : ");
9	Serial.print(WiFi.SSID(i));
10	Serial.print(" Ch:");
11	Serial.print(WiFi.channel(i));
12	Serial.print(" S:");
13	Serial.print(WiFi.RSSI(i));
14	Serial.print("dBm ENC:");
15	switch (WiFi.encryptionType(i)) {
16	case WIFI_AUTH_OPEN:
17	Serial.print("Open"); break;
18	case WIFI_AUTH_WEP:
19	Serial.print("WEP"); break;
20	case WIFI_AUTH_WPA_PSK:
21	Serial.print("WPA/PSK"); break;
22	case WIFI_AUTH_WPA2_PSK:
23	Serial.print("WPA2/PSK"); break;
24	case WIFI_AUTH_WPA_WPA2_PSK:
25	Serial.print("WPA/WPA2/PSK"); break;
26	case WIFI_AUTH_WPA2_ENTERPRISE:
27	Serial.print("WPA2 Enterprise");break;
28	default:
29	Serial.print("Unknown");
30	}
31	Serial.print(" BSSID:");
32	Serial.println(WiFi.BSSIDstr(i));
33	}
34	}

ตัวอย่างโปรแกรม 23-2 แสดงรายชื่อจุดเชื่อมต่อพร้อมรายละเอียด (ต่อ)

บรรทัด	คำสั่ง
35	void setup() {
36	Serial.begin(115200);
37	Serial.println("\n\r\n\r");
38	WiFi.mode(WIFI_STA);
39	WiFi.disconnect();
40	delay(100);
41	int num_aps = WiFi.scanNetworks();
42	showAPs(num_aps);
43	}
44	void loop() {
45	}

> โหมด AP

โหมด AP เป็นการกำหนดให้ ESP32 เป็นอุปกรณ์ให้บริการเว็บ และผู้ใช้ทำการเชื่อมต่อ WiFi มายังอุปกรณ์นี้เพื่ออ่านข้อมูลจากเว็บไซต์ซึ่งมีค่า IP Address เป็น 192.168.4.1 โดยชื่อของ ESSID และรหัสผ่านผู้พัฒนาสามารถกำหนดขึ้นเองได้ จากหลักการนี้จึงนำมาประยุกต์เพื่อสร้างตัวกระจายสัญญาณ WiFi สำหรับเครือข่ายภายในที่รองรับจำนวนโหนดเข้าใช้พร้อมกัน 3 โหนดตามตัวอย่างโปรแกรม 23-3

ตัวอย่างโปรแกรม 23-3 กำหนดให้ SSID เป็น “ETT_AP” รหัสผ่านสำหรับเข้าเชื่อมต่อเป็น “ETT12345678” และกำหนดค่าต่าง ๆ ของ IPAddress เป็น 192.168.4.54 ค่าเกตเวย์เป็น 192.168.4.11 และซับเน็ตเป็น 255.255.255.0 ในตัวแปร myIP, gwIP และ subnet ตามลำดับ หลังจากนั้นกำหนดโหมดเป็น WIFI_AP ในบรรทัดที่ 12 สร้างข้อมูลไอพีในบรรทัดที่ 24 และสร้างจุดเชื่อมต่อตามชื่อ SSID, PASSWD ใช้ช่องสัญญาณที่ 8 ไม่ซ่อนการแสดงชื่อและกำหนดให้เข้าใช้ได้สูงสุด 3 อุปกรณ์ และตัวอย่างของผลลัพธ์การทำงานเป็นดังภาพที่ 23-6

ตัวอย่างโปรแกรม 23-3 เชื่อมต่อกับจุดเชื่อมต่อและแสดง IP Address ที่ได้รับ

บรรทัด	คำสั่ง
1	#include < WiFi.h>
2	#define SSID "ETT_AP"
3	#define PASSWD "ETT12345678"
4	IPAddress myIP(192, 168, 4, 54);
5	IPAddress gwIP(192, 168, 4, 11);
6	IPAddress subnet(255, 255, 255, 0);
7	uint8_t myMACAddr[6];
8	void setup() {
9	Serial.begin(115200);
10	Serial.println("\n\r\n\r");
11	WiFi.begin();
12	WiFi.mode(WIFI_AP);
13	WiFi.macAddress(myMACAddr);
14	WiFi.softAPmacAddress(myMACAddr);
15	Serial.print("MAC Address : ");
16	Serial.println(WiFi.softAPmacAddress());
17	Serial.printf("%x:%x:%x:%x:%x:%x\n\n",
18	myMACAddr[0], myMACAddr[1],
19	myMACAddr[2], myMACAddr[3],
20	myMACAddr[4], myMACAddr[5]);
21	if (WiFi.softAPConfig(myIP, gwIP, subnet)) {
22	if (WiFi.softAP(SSID, PASSWD, 8, false, 3)) {
23	Serial.print("IP Address : ");
24	Serial.println(WiFi.softAPIP());
25	} else {
26	Serial.println("softAP() failed!!!");
27	}
28	} else {
29	Serial.println("softAPConfig() failed!");
30	}
31	}
32	void loop() {
33	}

```
Found 7 AP(s)
AP No.1 : ETT_AP Ch:8 S:-50dBm ENC:WPA2/PSK BSSID:7C:9E:BD:46:5C:59
AP No.2 : 
AP No.3 : 
AP No.4 : F7902_2.4G Ch:1 S:-54dBm ENC:WPA/WPA2/PSK BSSID:F0:9B:80:80:40:24
AP No.5 : 388-Tittle_2.4GHz Ch:6 S:-55dBm ENC:WPA/WPA2/PSK BSSID:34:1E:6B:23:B1:D4
AP No.6 : Meshokwong Ch:1 S:-55dBm ENC:WPA2/PSK BSSID:70:F9:2B:EB:94:39
AP No.7 : Kiatkavin_2.4G Ch:9 S:-55dBm ENC:WPA/WPA2/PSK BSSID:50:0E:84:F2:7F:4D
```

ภาพที่ 23-6

ตัวอย่างผลลัพธ์เมื่อสแกนหาจุดเชื่อมต่อเครือข่ายอินเทอร์เน็ต

ภาพโดย จารุต บุศราทิจ

จากภาพที่ 23-6 จะพบว่าเมื่อรันตัวอย่างโปรแกรม 23-3 เพื่อสแกนรายชื่อจุดเชื่อมต่อเครือข่ายไร้สายมีชื่อ ETT_AP ปรากฏอยู่ และมีการตั้งค่าถูกต้องตามที่เขียนโปรแกรมไว้คือ ใช้ช่องสัญญาณที่ 8 และ BSSID จะตรงกับ MAC Address ของอุปกรณ์

> โหมด STA

โหมด STA หรือ Station เป็นการเชื่อมต่อในกรณีที่ต้องการให้ ESP32 เชื่อมต่อเข้ากับเครือข่าย WiFi ผ่านทางจุดเชื่อมต่อที่มีอยู่แล้ว โดยในตัวอย่างโปรแกรม 23-4 เป็นเชื่อมต่อยังจุดเชื่อมต่อที่สร้างขึ้นจากตัวอย่างโปรแกรมที่ 23-3 หรือผู้อ่านกำหนดชื่อ SSID และ PASSWD ใหม่ตามการตั้งค่าของเครือข่ายไร้สายที่ใช้งาน

ตัวอย่างโปรแกรม 23-4 ทำการเชื่อมต่อไปยังจุดเชื่อมต่อชื่อ ETT_AP หลังจากนั้นทำการแสดง MAC Address ของอุปกรณ์ตามด้วยการรอจนกว่าจะเชื่อมต่อสำเร็จ เพื่อแสดงค่า IP และผลลัพธ์จาก printDiag() ดังจะเห็นตัวอย่างผลลัพธ์ในภาพที่ 23-7

ตัวอย่างโปรแกรม 23-4 เชื่อมต่อกับจุดเชื่อมต่อและแสดง IP Address ที่ได้รับ

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	#define SSID "ETT_AP"
3	#define PASSWD "ETT12345678"
4	uint8_t myMACaddr[6];
5	void setup() {
6	Serial.begin(115200);
7	WiFi.begin(SSID, PASSWD);

ตัวอย่างโปรแกรม 23-4 เชื่อมต่อกับจุดเชื่อมต่อและแสดง IP Address ที่ได้รับ (ต่อ)

บรรทัด	คำสั่ง
8	WiFi.mode(WIFI_STA);
9	WiFi.macAddress(myMACAddr);
10	Serial.printf("MAC Address:%x:%x:%x:%x:%x:%x\n\n",
11	myMACAddr[0], myMACAddr[1], myMACAddr[2],
12	myMACAddr[3], myMACAddr[4], myMACAddr[5]);
13	while (!WiFi.isConnected()) { delay(100); }
14	Serial.print("IP Address : ");
15	Serial.println(WiFi.localIP());
16	WiFi.printDiag(Serial);
17	WiFi.disconnect();
18	}
19	void loop() {
20	}

MAC Address : 84:cc:a8:7f:56:40	
IP Address : 192.168.4.55	
Mode: STA	
Channel: 8	
SSID (6): ETT_AP	
Passphrase (11): ETT12345678	
BSSID set: 0	
ภาพที่ 23-7	ตัวอย่างผลลัพธ์จากตัวอย่างโปรแกรม 23-4

ภาพโดย จารุต บุศราทิจ

> คลาส WiFiServer

คลาส WiFiServer เป็นคลาสสำหรับทำให้ไมโครคอนโทรลเลอร์ทำงานเป็นเครื่องให้บริการ โดยการสร้างวัตถุประเภทเครื่องให้บริการต้องระบุหมายเลขพอร์ตให้บริการในขั้นตอนการสร้าง ดังรูปแบบการใช้งานต่อไปนี้

WiFiServer วัตถุ (หมายเลขพอร์ต)

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

เมื่อได้วัตถุสำหรับอ้างอิงการทำงานของไลบรารีเครื่องให้บริการเป็นที่เรียบร้อย ขั้นตอนต่อไปคือการสร้างวัตถุแบบลูกข่ายที่เตรียมไว้สำหรับให้บริการดังนี้

WiFiClient วัตถุลูกข่าย = วัตถุ.available()

ถ้าวัตถุลูกข่ายไม่ใช่ 0 หมายถึง เครื่องให้บริการที่เราสร้างนั้นพร้อมให้บริการ ผู้เขียนโปรแกรมต้องทำการตรวจสอบว่ามีลูกข่ายใดเข้าเชื่อมต่อมาที่เครื่องให้บริการหรือไม่ ด้วยคำสั่งต่อไปนี้

ผลการตรวจสอบ = วัตถุลูกข่าย.connected()

การอ่าน ข้อมูลคำร้องขอ จากลูกข่ายที่อยู่ในรูปแบบของตัวแปรประเภท String ใช้คำสั่งใช้คำสั่ง readStringUntil() ดังรูปแบบต่อไปนี้

ข้อมูลคำร้องขอ = วัตถุลูกข่าย.readStringUntil(ตัวสิ้นสุดการอ่าน)

การส่ง ข้อมูลส่งกลับ เพื่อตอบสนองให้ลูกข่ายต้องใช้คำสั่ง print() หรือ println() ดังนี้

วัตถุลูกข่าย.print(ข้อมูลส่งกลับ)

วัตถุลูกข่าย.println(ข้อมูลส่งกลับ)

เพื่อความมั่นใจในการส่งข้อมูลหรือรับข้อมูลการ flush() เป็นการทำให้บัฟเฟอร์ว่างด้วยการนำเข้าหรือนำออกให้เสร็จสิ้นก่อนทำงานอื่น ๆ ต่อไป โดยคำสั่ง flush() มีรูปแบบการใช้งานดังนี้

วัตถุลูกข่าย.flush()

สุดท้ายเมื่อไม่มีการเชื่อมต่อหรือสื่อสารกับลูกข่ายให้ยกเลิกการเชื่อมต่อกับลูกข่ายด้วยคำสั่ง stop() ดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

วัตถุลูกข่าย.stop()

> คลาส WiFiClient

คลาส WiFiClient ที่ผู้เขียนโปรแกรมจะต้องสร้างวัตถุจากคลาสนี้เพื่อเชื่อมต่อไปยังเครื่องให้บริการผ่านทาง IP Address และหมายเลขพอร์ตที่ต้องการสื่อสาร ซึ่งการสร้างวัตถุประเภท WiFiClient เขียนคำสั่งได้ดังนี้

WiFiClient ชื่อวัตถุ()

การเริ่มต้นใช้งานวัตถุแบบ WiFiClient ต้องใช้คำสั่ง begin() ด้วยอาร์กิวเมนต์ เครื่องให้บริการ และ พอร์ต ดังรูปแบบต่อไปนี้

ชื่อวัตถุ.begin(เครื่องให้บริการ, พอร์ต)

การขอเชื่อมต่อไปยังเครื่องให้บริการที่หมายเลขไอพีและพอร์ตที่ต้องการนั้นใช้คำสั่ง connect() ตามรูปแบบการใช้งานต่อไปนี้ โดยผลลัพธ์ที่ได้กลับมาเป็น true หรือ false นอกจากนี้ต้องสร้างตัวแปรตำแหน่งไอพีของเครื่องให้บริการจากคลาส IPAddress

IPAddress เครื่องให้บริการ(xx, xx, xx, xx)

ผลของการเชื่อมต่อ = ชื่อวัตถุ.connect (เครื่องให้บริการ, พอร์ต)

กรณีที่เครือข่ายไร้สายที่เชื่อมต่อนั้นมีบริการ DNS อยู่ในระบบ ทำให้สามารถใช้ชื่อ หรือ URL แทนหมายเลขไอพีได้ ดังนั้น รูปแบบของคำสั่งเชื่อมต่อจึงเป็นดังต่อไปนี้

```
ผลของการเชื่อมต่อ = ชื่อวัตถุ.connect ( URL, พอร์ต )
```

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

กรณีที่ต้องการทราบสถานะของการเชื่อมต่อให้เรียกใช้คำสั่ง `connected()` เพื่ออ่านสถานะผลลัพธ์ของการเชื่อมต่อว่าเป็น `true` หรือ `false` ดังรูปแบบของการใช้งานต่อไปนี้

```
ผลของการเชื่อมต่อ = ชื่อวัตถุ.connected( )
```

การส่งข้อมูลไปยังเครื่องให้บริการที่เรียกว่าการร้องขอ (Request) ใช้คำสั่ง `println()` ดังนี้ ซึ่งคำร้องขอนั้นแตกต่างกันไปตามบริการหรือโปรโตคอลที่ต้องการสื่อสาร

```
ชื่อวัตถุ.println( คำร้องขอ )
```

เมื่อต้องการทราบว่าเครื่องให้บริการได้ส่งข้อมูลกลับมาตามการร้องขอหรือไม่ จะต้องใช้คำสั่ง `available()` เป็นคำสั่งตรวจสอบสถานะของการมีข้อมูลส่งกลับจากเครื่องให้บริการดังต่อไปนี้

```
สถานะ = ชื่อวัตถุ.available( )
```

การอ่าน ข้อมูล ที่ถูกตอบกลับจากเครื่องให้บริการด้วยการใช้คำสั่ง `read()` และผลลัพธ์ที่ได้กลับมานั้นจัดเก็บในรูปแบบแถวลำดับของข้อมูลแบบไบนารีดังนี้

```
ข้อมูล = ชื่อวัตถุ.read( )
```

เมื่อสิ้นสุดการเชื่อมต่อจะต้องเรียกใช้คำสั่ง `stop()` ดังนี้

```
ชื่อวัตถุ.stop( )
```

คำสั่งสำหรับอ่านสตริงผลลัพธ์จาก `WiFiClient` จนกว่าจะพบตัวอักษรที่ระบุคือ

ผลลัพธ์ = ชื่อวัตถุ.readStringUntil(ตัวอักษรที่ต้องการ)

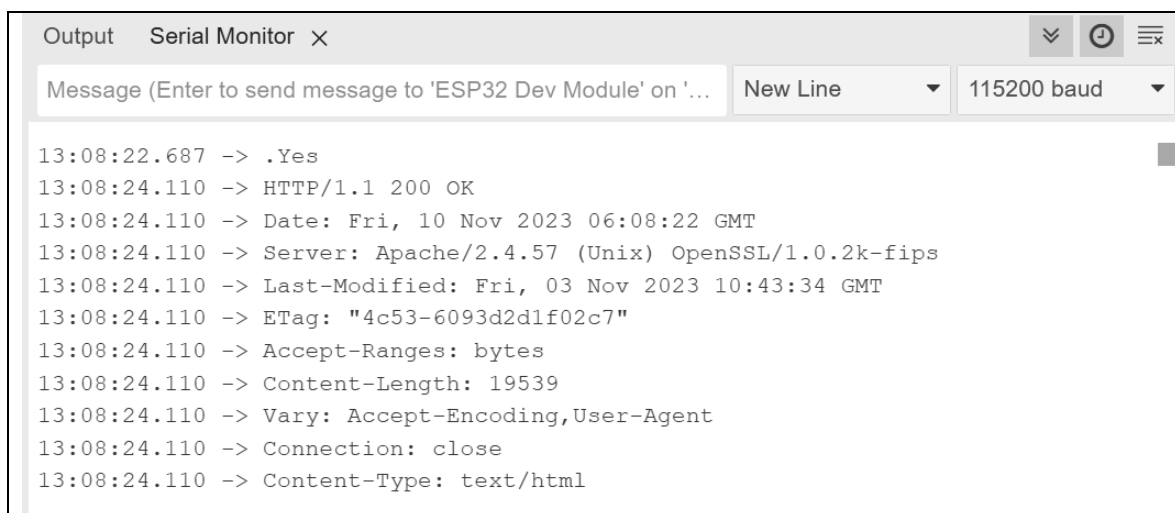
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ตัวอย่างโปรแกรมอ่านข้อมูลหน้าแรกจากเว็บ ETT ด้วยคลาส WiFiClient ที่เชื่อมต่อเข้าจุดเชื่อมต่ออินเทอร์เน็ตชื่อ ETT_AP ที่กำหนดรหัสผ่านเป็น ETT12345678 เป็นดังตัวอย่างโปรแกรม 23-5 และได้ตัวอย่างผลลัพธ์เป็นดังภาพที่ 23-8

ตัวอย่างโปรแกรม 23-5 อ่านข้อมูลหน้าแรกของ ETT	
บรรทัด	คำสั่ง
1	#include < WiFi.h>
2	#define SSID "ETT_AP"
3	#define PASSWD "ETT12345678"
4	WiFiClient client;
5	char url[] = "www.etteam.com";
6	void setup() {
7	Serial.begin(115200);
8	Serial.println("\n\r\n\r");
9	WiFi.mode(WIFI_STA);
10	WiFi.begin(SSID, PASSWD);
11	while (WiFi.status() != WL_CONNECTED) {
12	delay(250);
13	Serial.print(".");
14	}
15	if (client.connect(url, 80)) {
16	Serial.println("Yes");
17	client.print(String("GET ") +
18	"/ HTTP/1.1\r\nHost: " + String(url) +
19	"\r\nConnection: close\r\n\r\n");
20	delay(1000);
21	while (client.available()) {
22	String line = client.readStringUntil('\r');
23	Serial.print(line);
24	}
25	client.stop();
26	} else {
27	Serial.println("OMG!");

ตัวอย่างโปรแกรม 23-5 อ่านข้อมูลหน้าแรกของ ETT (ต่อ)

บรรทัด	คำสั่ง
28	}
29	WiFi.disconnect();
30	}
31	void loop() {
32	}



ภาพที่ 23-8

ตัวอย่างผลลัพธ์จากตัวอย่างโปรแกรม 23-5

ภาพโดย จารุต บุศราทิจ

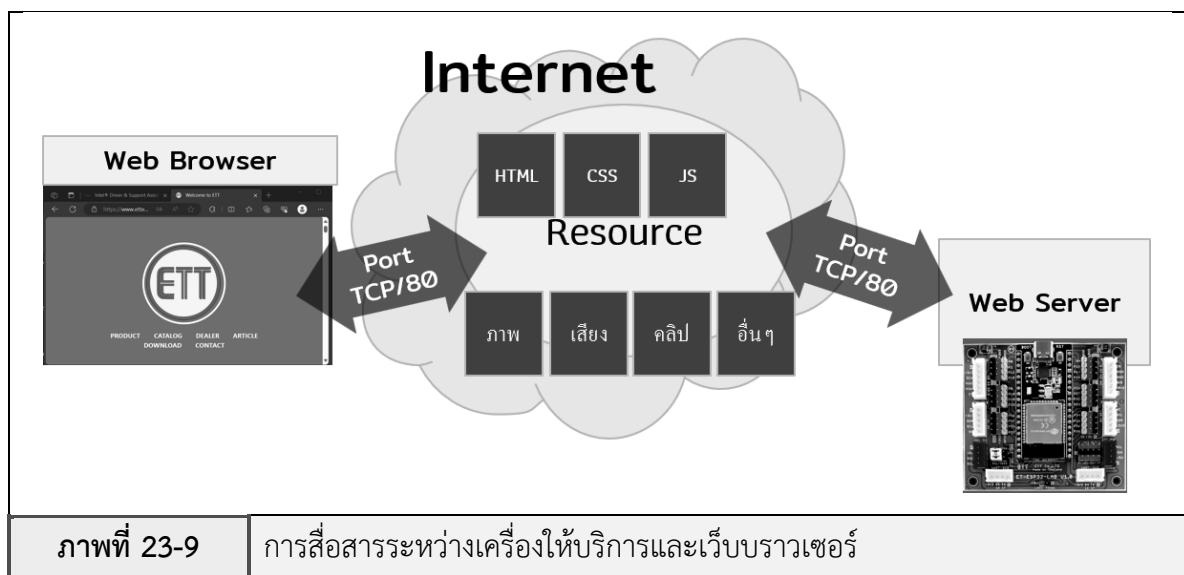
> ให้บริการเว็บ

การให้บริการเว็บเป็นวิธีการสั่งงานให้ไมโครคอนโทรลเลอร์ ESP32 ทำงานตอบสนองกับรูปแบบการสื่อสารหรือโพรโทคอล (Protocol) ประเภท HTTP ทางพอร์ตสื่อสารแบบ TCP หมายเลข 80 โดยการพัฒนาเอชทีทีพีเป็นการทำงานร่วมกันของ W3C และ IETF ซึ่งมีผลงานเด่นในการเผยแพร่เอกสารขอความเห็น (RFC) โดยมีเอกสาร RFC 2616 ที่เผยแพร่เมื่อเดือนมิถุนายน พ.ศ. 2542 เป็นเอกสารที่กำหนดรูปแบบการสื่อสาร HTTP/1.1 ซึ่งเป็นรุ่นที่ใช้กันอย่างกว้างขวางในปัจจุบัน และมีองค์ประกอบของอุปกรณ์ที่ต้องใช้ ดังภาพที่ 23-8

จากภาพที่ 23-8 จะพบว่า เมื่อผู้ใช้เรียกโปรแกรมเว็บเบราว์เซอร์ (Web Browser) จะต้องกำหนดแหล่งที่อยู่ของทรัพยากรที่รู้จักในชื่อของ URL หรือชื่อเว็บไซต์ โดย URL เป็นได้ทั้งรูปแบบของชื่อ หรือ IP

Address (ตัว DNS ทำหน้าที่แปลงข้อมูลชื่อเป็น IP Address ให้เองอัตโนมัติ แต่ในหนังสือเล่มนี้จะใช้หมายเลขไอพีของอุปกรณ์เป็น URL) เมื่อระบบเครือข่ายค้นพบตำแหน่งเป้าหมายตามที่กำหนดใน URL จะทำการเชื่อมต่อผ่านพอร์ตหมายเลข 80 (ถ้าถูกปิดกั้นด้วยระบบความปลอดภัยจะไม่สามารถใช้งาน) พร้อมส่งรหัสร้องขอทรัพยากร (Resource) จากเครื่องให้บริการที่ถูกออกแบบซอฟต์แวร์ให้ทำการรอคอยการร้องขอจากพอร์ตหมายเลข 80 ดังนั้น เมื่อได้รับรหัสร้องขอมาจากเว็บเบราว์เซอร์หรือเครื่องลูกข่าย (ที่อาศัยพอร์ต 80 ในการสื่อสาร) จะทำการตรวจสอบการคงอยู่ของทรัพยากร และส่งรหัสตอบกลับไปให้โดยถ้ารหัสสถานะเป็น "HTTP/1.1 200 OK" หมายถึง พบสิ่งที่ต้องการ และดำเนินการส่งข้อมูลหลังจากส่งรหัสนี้ แต่ถ้าไม่พบจะสามารถส่งรหัสตอบกลับตามลักษณะของปัญหาที่เกิดขึ้น เช่น

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 408 Request Timeout
- 500 Internal Server Error
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout



ภาพที่ 23-9

การสื่อสารระหว่างเครื่องให้บริการและเว็บเบราว์เซอร์

ภาพโดย จารุต บุศราทิจ

จากภาพที่ 23-9 จะพบว่า การให้บริการเว็บเป็นการใช้งานพอร์ตสื่อสารหมายเลข 80 ของโพรโทคอลสื่อสารแบบทีซีพี โดยลูกข่ายจะทำการร้องขอทรัพยากรจากเครื่องให้บริการ ซึ่งสิ่งที่เครื่องให้บริการหรือไมโครคอนโทรลเลอร์ ESP32 จะส่งกลับไปยังเครื่องลูกข่ายคือ รหัสหมายเลข 200 เพื่อบอกว่า

พร้อมให้บริการข้อมูล และตัวข้อมูลที่ต้องอยู่ในรูปแบบของภาษาเอชทีเอ็มแอล หรือ HTML (Hyper Text Markup Language) เพื่อให้เว็บเบราว์เซอร์ทำการเรนเดอร์ (Render) ผลลัพธ์บนหน้าจอของโปรแกรม ทำให้นักพัฒนางานด้านไอโอทีหรือระบบควบคุมประหยัอุปกรณ์แสดงผลเนื่องจากสามารถเข้าดูได้โดยตรงจากเว็บเบราว์เซอร์ โดยการเข้าถึงอุปกรณ์ผ่านเว็บเบราว์เซอร์ต้องระบุหมายเลขไอพีของอุปกรณ์ที่ช่องยูอาร์แอลของเว็บเบราว์เซอร์ และด้วยวิธีการนี้จึงทำให้สามารถใช้เครื่องลูกข่ายเป็นระบบใด ๆ ที่มีเว็บเบราว์เซอร์ใช้งานเท่านั้น

ขั้นตอนของการทำให้ ESP32 ทำหน้าที่เป็นเครื่องให้บริการเว็บเป็นดังนี้

1. เชื่อมต่อเข้าเครือข่ายอินเทอร์เน็ตผ่านทางจุดให้บริการหรือกำหนดให้ตนเองเป็นจุดให้บริการ
2. สร้างตัวแปรสำหรับทำหน้าที่เป็นเครื่องให้บริการจากคลาส WiFiServer
3. ในฟังก์ชัน loop() รอรับการร้องขอจากเครื่องลูกข่ายผ่านทางคลาส WiFiClient
4. อ่านคำร้องขอจากเครื่องลูกข่าย
5. ตอบกลับไปยังลูกข่าย
 - 1) ส่งรหัสตอบกลับ
 - 2) ส่งข้อมูลภาษาเอชทีเอ็มแอล
6. ปิดการเชื่อมต่อด้วย stop()

ตัวอย่างโปรแกรม 23-6 เป็นการแสดงค่าอุณหภูมิและค่าคลื่นแม่เหล็กจากเซ็นเซอร์ภายในตัว ESP32 ในรูปแบบของเว็บ โดยกำหนดให้ไมโครคอนโทรลเลอร์ทำหน้าที่เป็นจุดให้บริการเชื่อมต่อ ดังนั้นผู้อ่านจะต้องเชื่อมต่อเข้ากับ WiFi ของ ESP32 (กำหนดชื่อจุดเชื่อมต่อเป็น ETT_AP และรหัสผ่านเป็น ETT12345678) และผลลัพธ์ที่ได้จะต้องดูจากการเว็บเบราว์เซอร์ที่ระบุหมายเลขไอพีเป็น 192.168.4.1 ดังภาพที่ 23-10

ตัวอย่างโปรแกรม 23-6 แสดงค่าอุณหภูมิและคลื่นแม่เหล็กผ่านเว็บ

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	#define SSID "ETT_AP"
3	#define PASSWD "ETT12345678"
4	WiFiServer server(80);
5	String html;
6	#ifdef __cplusplus
7	extern "C" {
8	#endif
9	uint8_t temprature_sens_read();

ตัวอย่างโปรแกรม 23-6 แสดงค่าอุณหภูมิและคลื่นแม่เหล็กผ่านเว็บ (ต่อ)

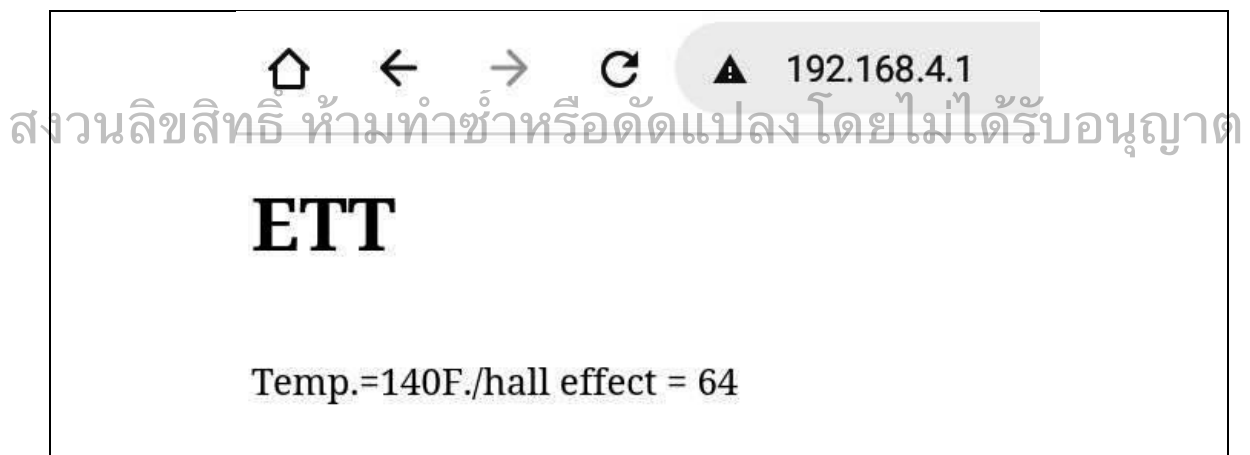
บรรทัด	คำสั่ง
10	<code>#ifdef __cplusplus</code>
11	<code>}</code>
12	<code>#endif</code>
13	<code>void html_page() {</code>
14	<code>String t = String(temperature_sens_read());</code>
15	<code>String m = String(hallRead());</code>
16	<code>html = F("HTTP/1.1 200 OK\r\n"</code>
17	<code> "Content-Type: text/html\r\n"</code>
18	<code> "Connection: close\r\n"</code>
19	<code> "Refresh: 5\r\n\r\n");</code>
20	<code>html += F("<!DOCTYPE HTML>"</code>
21	<code> "<html><head>"</code>
22	<code> "<meta charset='utf-8' "</code>
23	<code> "name='viewport' "</code>
24	<code> "content='width=device-width, "</code>
25	<code> "initial-scale=1'></head><body>");</code>
26	<code>html += F("<h1>ETT</h1>
Temp.=");</code>
27	<code>html += t;</code>
28	<code>html += F("F./hall effect = ");</code>
29	<code>html += m;</code>
30	<code>html += F("</body></html>");</code>
31	<code>}</code>
32	<code>void setup() {</code>
33	<code> Serial.begin(115200);</code>
34	<code> Serial.println("\n\n\n");</code>
35	<code> html.reserve(2048);</code>
36	<code> WiFi.begin();</code>
37	<code> WiFi.mode(WIFI_AP);</code>
38	<code> WiFi.macAddress(myMACaddr);</code>
39	<code> WiFi.softAPmacAddress(myMACaddr);</code>
40	<code> if (!WiFi.softAP(SSID, PASSWD, 8, false, 3)) {</code>
41	<code> Serial.println("SoftAP failed!");</code>
42	<code> while (1);</code>
43	<code>}</code>

ตัวอย่างโปรแกรม 23-6 แสดงค่าอุณหภูมิและคลื่นแม่เหล็กผ่านเว็บ (ต่อ)

บรรทัด	คำสั่ง
44	server.begin();
45	Serial.println("ready!");
46	}
47	void loop() {
48	WiFiClient client = server.available();
49	if (!client) return;
50	String req = client.readStringUntil('\r');
51	if (req.indexOf("GET / HTTP/1.1") == 0) {
52	html_page();
53	client.println(html);
54	client.flush();
55	delay(1);
56	} else {
57	client.println(
58	"HTTP/1.1 503 Service Unavailable");
59	client.println();
60	}
61	client.stop();
62	}

จากตัวอย่างโปรแกรม 23-6 จะพบว่า ฟังก์ชัน `html_page()` เป็นฟังก์ชันที่สร้างเพื่อสร้างผลลัพธ์ที่ส่งกลับไปให้เครื่องรับบริการ โดยบรรทัดที่ 16 ถึง 19 เป็นการสร้างรหัสสำหรับตอบกลับ และบรรทัดที่ 19 กำหนดให้มีการรีเฟรช (refresh) หน้าเว็บทุก 5 วินาที ส่วนบรรทัดที่ 20 ถึง 30 เป็นส่วนของภาษาเอชทีเอ็มแอลที่ส่งกลับไปให้เครื่องรับบริการที่มีการแทรกข้อมูล `t` (บรรทัดที่ 14) และ `m` (บรรทัดที่ 15) ที่อ่านจากเซ็นเซอร์อุณหภูมิและคลื่นแม่เหล็กของ ESP32 พร้อมทั้งเก็บผลลัพธ์ที่สร้างลงตัวแปร `html` ซึ่งเป็นตัวแปรแบบภายนอก ทำให้นำกลับมาใช้ได้โดยไม่ต้องสร้างใหม่ทุกครั้ง เนื่องจากการเปลี่ยนแปลงค่าของตัวแปรนี้จะเกิดจากการเรียกฟังก์ชัน `html_page()` เท่านั้น

บรรทัดที่ 50 ถึง 61 เป็นส่วนของการอ่านค่าข้อมูลจากบัฟเฟอร์ (ตัวแปร `req` จากบรรทัดที่ 50) และทำการตรวจสอบว่าเมื่อใดพบข้อความ "GET / HTTP/1.1" จะดำเนินการสร้างภาษาเอชทีเอ็มแอล แต่ถ้าข้อความร้องขอไม่ใช่ตามที่ตรวจสอบจะรายงานกลับเป็น 503 หลังจากนั้นหยุดการเชื่อมต่อกับลูกค้าที่เชื่อมต่อเข้ามา แต่อย่างไรก็ดี ด้วยการกำหนดให้มีการรีเฟรชตัวเองทุก 5 วินาทีจึงทำให้เว็บเบราว์เซอร์ดำเนินการร้องขอใหม่ตามเวลาดังกล่าว ยกเว้นผู้ใช้ปิดเบราว์เซอร์



ภาพที่ 23-10

ตัวอย่างผลลัพธ์บนเว็บเบราว์เซอร์จากตัวอย่างโปรแกรม 23-6

ภาพโดย จารุต บุศราทิจ

> ตอบสนองจากเว็บ

กรณีที่ต้องตอบสนองจากเว็บจะมีความคล้ายกับตัวอย่างโปรแกรม 23-6 เนื่องจากในบรรทัดที่ 57 มีการใช้คำสั่ง `indexOf( "ข้อความที่ต้องการค้นหา" )` ในการค้นหาข้อความที่เก็บในตัวแปร `req` อันเป็นสตริงที่อ่านจากบัฟเฟอร์การส่งร้องขอมาจากเครื่องลูกข่าย ดังนั้น เมื่อนำมาประยุกต์กับการตอบสนองจะต้องมีการออกแบบรูปแบบของการร้องขอใหม่โดยใช้คำสั่ง `GET` เช่นเดียวกัน แต่เปลี่ยนทรัพยากรที่ต้องการร้องขอ เช่น กำหนดให้เมื่อสั่ง `"GET /reboot HTTP/1.1"` เป็นการรับคำสั่งให้รีเซ็ตบอร์ด ESP32 และที่โค้ดภาษาเอชทีเอ็มแอลในส่วนของการแสดงปุ่มจะใช้แท็ก `<a>` เป็นตัวกำหนดให้เมื่อคลิกที่ปุ่มให้ `href='/reboot'` ดังตัวอย่างโปรแกรม 23-7 เป็นต้น

ตัวอย่างโปรแกรม 23-7 ส่งรีเซ็ตอุปกรณ์ผ่านเว็บ

บรรทัด	คำสั่ง
1	#include <WiFi.h>
2	#define SSID "ETT_AP"
3	#define PASSWD "ETT12345678"
4	WiFiServer server(80);
5	String html;
6	void html_page() {
7	String t = String(temperature_sens_read());
8	String m = String(hallRead());
9	html = F("HTTP/1.1 200 OK\r\n"
10	"Content-Type: text/html\r\n"
11	"Connection: close\r\n"
12	"\r\n");
13	html = F("<!DOCTYPE HTML>"
14	"<html><head>"
15	"<meta charset=\"utf-8\" "
16	"name=\"viewport\" "
17	"content=\"width=device-width,\"
18	"initial-scale=1\">"
19	"<style>"
20	".button { border: none;"
21	"color: white; padding: 20px;"
22	"text-align: center;"
23	"text-decoration: none;"
24	"display: inline-block;"
25	"font-size: 14px;"
26	"margin: 4px 2px;"
27	"cursor: pointer;"
28	"border-radius: 4%;"
29	"width: 100%; height: 100%;"
30	"}"
31	".button1{background-color: #3ABC40;}"
32	"</style></head><body><table>");
33	html += F(" ETT ");
34	html += F(""

ตัวอย่างโปรแกรม 23-7 สร้างเว็บไซต์อุปกรณ์ผ่านเว็บ (ต่อ)

บรรทัด	คำสั่ง
35	"<button class=\"button button1\">"
36	"Reboot</button>";
37	html += F("</body></html>");
38	}
39	void setup () {
40	Serial.begin(115200);
41	Serial.println("\n\n\n");
42	html.reserve(2048);
43	WiFi.begin();
44	WiFi.mode(WIFI_AP);
45	WiFi.macAddress(myMACaddr);
46	WiFi.softAPmacAddress(myMACaddr);
47	if (!WiFi.softAP(SSID, PASSWD, 8, false, 3)) {
48	Serial.println("SoftAP failed!");
49	while (1);
50	}
51	server.begin();
52	Serial.println("ready!");
53	}
54	void loop () {
55	WiFiClient client = server.available();
56	if (!client) return;
57	String req = client.readStringUntil('\r');
58	if (req.indexOf("GET / HTTP/1.1") == 0) {
59	html_page();
60	client.println(html);
61	client.flush();
62	delay(1);
63	}
64	else if (req.indexOf("GET /reboot HTTP/1.1")==0)
65	{
66	ESP.restart();
67	} else {
68	client.println(

ตัวอย่างโปรแกรม 23-7 ส่งรีเซตอุปกรณ์ผ่านเว็บ (ต่อ)

บรรทัด	คำสั่ง
69	"HTTP/1.1 503 Service Unavailable");
70	client.println();
71	}
72	client.stop();
73	}

> ESP Now

ESP-NOW เป็นโปรโตคอลสื่อสารที่บริษัท espressif เป็นผู้ออกแบบเพื่อใช้ในการสื่อสารระหว่างอุปกรณ์ผ่านทางระบบไร้สายหรือ WiFi โดยรองรับการสื่อสารแบบเข้ารหัสและไม่เข้ารหัส รองรับการส่งข้อมูล/รับข้อมูลได้สูงสุดเฟรมละ 250 ไบต์ และรองรับการใช้ callback สำหรับรับการแจ้งเตือนผลของการส่งข้อมูล แต่อย่างไรก็ดี ขีดจำกัดของ ESP-NOW มีด้วยกัน 3 ประเด็น คือ

- เมื่อใช้การเข้ารหัส (Encryption) ในการสื่อสารจะทำงานได้เพียง 10 อุปกรณ์ในโหมด Station และเหลือเพียง 6 อุปกรณ์เมื่อทำงานโหมด SoftAP หรือ SoftAP+Station
- การสื่อสารแบบไม่เข้ารหัสสามารถทำงานร่วมกันได้ไม่เกิน 20 อุปกรณ์
- ข้อมูลในเฟรมของการรับ/ส่งหรือส่วนของข้อมูลหรือเพย์โหลด (Payload) นั้นมีขนาดได้ไม่เกิน 250 ไบต์

สถาปัตยกรรมการสื่อสารของ ESP-NOW กำหนดให้ในแต่ละวงสื่อสารมีโหนดที่เรียกว่า ผู้ส่ง หรือ มาสเตอร์ หรือคอนโทรลเลอร์ (controller) กับผู้รับ หรือสเลฟ หรือเพียร์ (peer) โดยระหว่าง 2 ประเภทสามารถสื่อสารได้ 2 ลักษณะ คือ

- สื่อสารทางเดียว (one-way communication) ที่ให้อุปกรณ์หนึ่งเป็นผู้ส่งและอีกอุปกรณ์เป็นผู้รับ โดยทำงานได้ 3 รูปแบบ คือ
 - 1 ผู้ส่ง และ 1 ผู้รับ (one to one)
 - 1 ผู้ส่ง หลายผู้รับ (one master, multiple slaves)
 - หลายผู้ส่ง และ 1 ผู้รับ (multiple masters, one slave)
- การสื่อสารสองทาง (two-way communication) ที่ให้ทั้งผู้ส่งและผู้รับ สามารถเป็นทั้งผู้ส่งและผู้รับ

ค่าคงที่เหล่านี้เป็นผลที่คืนค่ากลับมาในประเภท `esp_err_t` ซึ่งเป็นประเภทข้อมูลที่คืนกลับจากชุดคำสั่งที่เกี่ยวข้องกับ ESP-NOW

- `ESP_ERR_ESPNOW_NOT_INIT` ผู้เขียนโปรแกรมยังไม่ได้เรียก `esp_now_init()`
- `ESP_ERR_ESPNOW_ARG` อาร์กิวเมนต์ที่ส่งให้คำสั่งมีความผิดพลาด
- `ESP_ERR_ESPNOW_NO_MEM` หน่วยความจำไม่เพียงพอ
- `ESP_ERR_ESPNOW_FULL` จำนวนเพียร์ ในลิสต์เต็ม
- `ESP_ERR_ESPNOW_NOT_FOUND` ไม่พบเพียร์ของ ESP-NOW
- `ESP_ERR_ESPNOW_INTERNAL` เกิดความผิดพลาดภายใน (Internal error)
- `ESP_ERR_ESPNOW_EXIST` มีผู้รับนี้อยู่ก่อนแล้ว
- `ESP_ERR_ESPNOW_IF` เกิดความผิดพลาดที่อินเทอร์เฟซ (Interface error)
- `ESP_OK` คำสั่งทำงานสำเร็จหรือการรับ/ส่งสำเร็จ

รายการค่าคงที่ภายใน ESP-NOW สำหรับใช้อ้างอิงเกี่ยวกับข้อจำนวนของข้อมูล ขนาดคีย์ และจำนวนเพียร์ มีดังนี้

- `ESP_NOW_ETH_ALEN` มีค่าเป็น 6 หมายถึง ขนาดของ MAC Address ของผู้รับ
- `ESP_NOW_KEY_LEN` มีค่าเป็น 16 หมายถึง ขนาดของกุญแจต้นฉบับ (master key) ของผู้รับ
- `ESP_NOW_MAX_TOTAL_PEER_NUM` มีค่าเป็น 20 หมายถึง จำนวนสูงสุดของผู้รับ
- `ESP_NOW_MAX_ENCRYPT_PEER_NUM` มีค่าเป็น 6 หมายถึง จำนวนสูงสุดของผู้รับเมื่อสื่อสารแบบเข้ารหัส
- `ESP_NOW_MAX_DATA_LEN` มีค่าเป็น 250 หมายถึง จำนวนไบต์สูงสุดของข้อมูลที่ใช้ในการสื่อสารต่อ 1 เฟรม

โครงสร้างข้อมูลสำหรับเก็บรายละเอียดของแต่ละเพียร์อยู่ในโครงสร้างข้อมูลชื่อ `esp_now_peer_info` หรือ `esp_now_peer_info_t` ซึ่งประกอบด้วยข้อมูลดังนี้

```
typedef struct esp_now_peer_info {
    uint8_t peer_addr[ESP_NOW_ETH_ALEN];
    uint8_t lmk[ESP_NOW_KEY_LEN];
    uint8_t channel;
    wifi_interface_t ifidx;
    bool encrypt;
    void *priv;
} esp_now_peer_info_t;
```

จากโครงสร้างดังกล่าวจะได้ว่า

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

- `peer_addr` เป็นค่า MAC Address ของผู้รับ ซึ่งถูกใช้ในการทำงานโหมด Station หรือ SoftAP
- `lmk` เป็นค่าขนาดของกุญแจต้นฉบับ ที่ใช้เป็นตัวเข้ารหัสและถอดรหัสข้อมูล
- `channel` คือ หมายเลขช่องสื่อสารระหว่างกัน แต่ถ้ากำหนดเป็น 0 หมายถึงการใช้ช่องสื่อสารตาม Station หรือ SoftAP ที่สื่อสารด้วย สำหรับกรณีอื่น ต้องระบุหมายเลขช่องสัญญาณให้ตรงกับ Station หรือ SoftAP ที่สื่อสาร
- `ifidx` คือ ค่าหมายเลขของส่วนเชื่อมต่อประสานหรืออินเตอร์เฟซ (IF: Interface)
- `encrypt` ถ้าเป็น `true` หมายถึง การสื่อสารกับผู้รับจะถูกเข้ารหัส แต่ถ้าเป็น `false` เป็นการสื่อสารแบบไม่เข้ารหัส
- `priv` เป็นตัวแปรตัวชี้ที่ชี้ไปยังข้อมูลส่วนตัวของผู้รับ

คำสั่งเริ่มต้นทำงาน

```
สถานะ = esp_now_init( )
```

สถานะ เป็นคำสั่งนี้ใช้สำหรับการเริ่มต้นการทำงานของ ESP-NOW และคืนค่า

- `ESP_OK` ถ้าสำเร็จ
- `ESP_ERR_ESPNOW_INTERNAL`

คำสั่งที่ใช้สำหรับยกเลิกการทำงาน ESP-NOW และจะคืนค่ากลับมาเป็น `ESP_OK` หลังทำการยกเลิกเสร็จสิ้น

```
สถานะ = esp_now_deinit( )
```

คำสั่งขอข้อมูลของ ESP-NOW ใช้สำหรับอ่านหมายเลขรุ่นของ ESP-NOW โดยต้องส่งตัวแปรประเภท `uint32_t` และนำค่ารุ่นเก็บในหน่วยความจำที่อ้างถึง หลังจากนั้นคืนกลับค่า สถานะ เป็น `ESP_OK` หรือ `ESP_ERR_ESPNOW_ARG` ถ้ามีปัญหาเรื่องของอาร์กิวเมนต์ที่ส่งให้กับฟังก์ชัน โดยรูปแบบของการใช้งานเป็นดังนี้

```
สถานะ = esp_now_get_version( &version )
```

คำสั่ง callback ที่ใช้ในการกำหนดให้ถูกเรียกใช้เมื่อ ESP-NOW ได้รับข้อมูลจะต้องมีโครงสร้างของการกำหนดฟังก์ชันดังนี้

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

```
typedef void (*esp_now_recv_cb_t)( const uint8_t *mac_addr,
                                   const uint8_t *data, int data_len )
```

รูปแบบของฟังก์ชัน callback ที่จะถูกเรียกให้ทำงานเมื่อเกิดข้อผิดพลาดจากการส่งข้อมูลจะต้องมีโครงสร้างดังนี้

```
typedef void (*esp_now_send_cb_t)( const uint8_t *mac_addr,
                                   esp_now_send_status_t status )
```

คำสั่งสำหรับกำหนดฟังก์ชัน callback เมื่อเกิดเหตุการณ์ register send เป็นดังนี้

```
สถานะ = esp_now_register_send_cb( cb )
```

จากคำสั่งจะได้ว่า cb คือ ชื่อของฟังก์ชัน callback ที่เขียนเอาไว้สำหรับถูกเรียก และ สถานะ เป็นค่าคืนกลับจากคำสั่งมีผลดังรายการต่อไปนี้

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_INTERNAL

คำสั่งสำหรับกำหนดฟังก์ชัน callback ที่ต้องถูกเรียกให้ทำงานเมื่อเกิดการรับข้อมูลมีรูปแบบดังนี้

```
สถานะ = esp_now_register_recv_cb( cb )
```

สถานะ เป็นค่าคืนกลับจากคำสั่ง ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_INTERNAL

คำสั่งสำหรับยกเลิกการลงทะเบียนฟังก์ชัน callback ที่ถูกเรียกหลังจากเกิดการรับข้อมูลมีรูปแบบดังนี้

```
สถานะ = esp_now_unregister_recv_cb( )
```

สถานะ เป็นค่าคืนกลับจากคำสั่ง ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT

คำสั่งสำหรับขอยกเลิกการลงทะเบียนฟังก์ชัน callback สำหรับตอบสนองการส่งข้อมูลมีรูปแบบของคำสั่งดังนี้

```
สถานะ = esp_now_unregister_send_cb( )
```

สถานะ เป็นค่าคืนกลับจากคำสั่ง ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT

คำสั่งสำหรับส่งข้อมูล data ขนาด len ไปให้ผู้รับ ที่มีค่าตำแหน่งเป็น peer_addr มีรูปแบบการใช้งานดังนี้

```
สถานะ = esp_now_send( const uint8_t *peer_addr,
                        const uint8_t *data, size_t len )
```

สถานะ เป็นค่าคืนกลับของการทำงาน ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG
- ESP_ERR_ESPNOW_INTERNAL
- ESP_ERR_ESPNOW_NO_MEM
- ESP_ERR_ESPNOW_NOT_FOUND
- ESP_ERR_ESPNOW_IF

คำสั่งสำหรับเพิ่มผู้รับเข้าในระบบมีรูปแบบของการใช้งานดังนี้

สถานะ = **esp_now_add_peer**(const esp_now_peer_info_t *peer)

สถานะ เป็นค่าคืนกลับของฟังก์ชันเพิ่ม peer ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG
- ESP_ERR_ESPNOW_FULL
- ESP_ERR_ESPNOW_NO_MEM
- ESP_ERR_ESPNOW_EXIST

กรณีที่ต้องการนำออกหรือลบผู้รับที่ไม่ต้องการโดยต้องระบุ peer_addr มีรูปแบบของการใช้งานดังนี้

สถานะ = **esp_now_del_peer**(const uint8_t *peer_addr)

เมื่อทำงานเสร็จจะรายงานผลของการทำงานในตัวแปร สถานะ คือ

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG
- ESP_ERR_ESPNOW_NOT_FOUND

กรณีที่ต้องการเปลี่ยนแปลงข้อมูลของผู้รับ ต้องส่งค่าข้อมูลชุดใหม่ให้กับฟังก์ชันต่อไปนี้

```
สถานะ = esp_now_mod_peer( const esp_now_peer_info_t *peer)
```

สถานะ เป็นค่าคืนกลับจากฟังก์ชัน esp_now_mod_peer() ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG
- ESP_ERR_ESPNOW_FULL

สำหรับการอ่านข้อมูลผู้รับ ที่มีค่าตำแหน่งเป็น peer_addr ต้องเตรียมบัฟเฟอร์สำหรับเก็บ esp_now_peer_info_t และส่งค่าตำแหน่งไปให้กับคำสั่งต่อไปนี้เพื่ออ่านข้อมูลของ peer ที่ระบุมาเก็บในบัฟเฟอร์ที่เตรียมไว้

```
สถานะ = esp_now_get_peer( const uint8_t *peer_addr,
                           esp_now_peer_info_t *peer )
```

สถานะ เป็นค่าผลลัพธ์ของการทำงานเป็นดังนี้

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG
- ESP_ERR_ESPNOW_NOT_FOUND

กรณีที่ทราบหมายเลขของผู้รับที่ต้องการอ่านข้อมูล ผู้เขียนโปรแกรมสามารถทำการอ่านจาก ลิสต์ของผู้รับได้ด้วยคำสั่งต่อไปนี้ เพื่อเข้าถึงทีละรายการ ด้วยวิธีการกำหนดให้ from_head เป็น true หมายถึง การอ่านรายการแรกจากลิสต์ และถ้ากำหนดเป็น false จะเป็นรายการถัดไปเรื่อย ๆ ในทุกครั้งที่เรียกใช้

```
สถานะ = esp_now_fetch_peer( bool from_head,
                             esp_now_peer_info_t *peer )
```

สถานะ เป็นค่าคืนกลับจาก esp_now_fetch_peer() มีดังนี้

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT

- ESP_ERR_ESPNOW_ARG
- ESP_ERR_ESPNOW_NOT_FOUND

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

การตรวจสอบว่ามี peer_addr อยู่ในลิสต์ของผู้รับหรือไม่ โดยได้ สถานะ เป็นผลลัพธ์จากการทำงานเป็น true หรือ false มีรูปแบบของคำสั่งใช้งานดังนี้

```
สถานะ = esp_now_is_peer_exist( const uint8_t *peer_addr )
```

กรณีที่ต้องการทราบจำนวนผู้รับที่เพิ่มเข้ามาในระบบ สามารถส่งนับจำนวนของผู้รับ ด้วยคำสั่ง esp_now_get_peer_num() โดยส่งค่าตำแหน่งของตัวแปรเก็บจำนวน ผู้รับให้กับฟังก์ชันต่อไปนี้

```
สถานะ = esp_now_get_peer_num( esp_now_peer_num_t *num )
```

สถานะ เป็นผลลัพธ์ของการทำงานของฟังก์ชัน หรือค่าคืนกลับจากฟังก์ชันนั้นได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG

คำสั่งสำหรับตั้งค่า PMK (Primary Master Key) ซึ่งเป็นคีย์ของการเข้าและถอดรหัสของการสื่อสารภายในเครือข่ายของ ESP-NOW มีรูปแบบการใช้งานดังนี้

```
สถานะ = esp_now_set_pmk(const uint8_t *pmk)
```

สถานะ เป็นค่าคืนกลับจากการกำหนดค่ารหัสของการเข้าและถอดรหัส ได้แก่

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT
- ESP_ERR_ESPNOW_ARG

การตั้งค่าสำหรับการออกจากโหมดประหยัดพลังงานโดยการกำหนดค่า window ในหน่วยไมโครวินาที ซึ่งอยู่ในช่วง 0 ถึง 65535 เพื่อให้ชิพถูกปลุกในทุก window ไมโครวินาทีนั้นมีรูปแบบการใช้งานดังนี้

```
สถานะ = esp_now_set_wake_window( uint16_t window )
```

ค่าคืนกลับของฟังก์ชันตั้งค้าระยะเวลาในการปลุกให้ชิพทำงานมีดังนี้

- ESP_OK
- ESP_ERR_ESPNOW_NOT_INIT

ขั้นตอนวิธีการเขียนโปรแกรมฝั่ง master

1. เริ่มต้น ESP-NOW ด้วยการให้ผู้ส่งทำงานในโหมดของ STA
2. ทำการสแกนหาผู้รับหรือลูกข่ายด้วยการพิจารณาจากชื่อ SSID ที่ขึ้นต้นด้วยคำว่า 'slave'
3. เมื่อพบลูกข่ายให้เพิ่มลูกข่ายนั้นเข้าระบบเครือข่าย ESP-NOW
4. ร้องขอลงทะเบียนจากลูกข่าย
5. เริ่มต้นการส่งข้อมูลจากเครื่องผู้ส่งไปยังผู้รับ

ขั้นตอนวิธีการเขียนโปรแกรมฝั่งผู้รับ

1. เปิดการทำงาน ESP-NOW ของฝั่งผู้รับ
2. เปลี่ยนชื่อ SSID ของผู้รับให้ขึ้นต้นด้วยคำว่า 'slave'
3. กำหนดโหมดทำงานของ WiFi เป็นโหมด AP
4. ร้องขอลงทะเบียนเพื่อรอรับข้อมูลจากผู้ส่ง
5. เริ่มต้นรับข้อมูลจากเครื่องผู้ส่ง

หมายเหตุ

มีผลเฉพาะเมื่อเปิดโหมดการทำงาน ESP_WIFI_STA_DISCONNECTED_PM_ENABLE การกำหนดค่าทำได้เฉพาะกับโหมด Station และเมื่ออยู่ในสถานะไม่ได้เชื่อมต่อ หากมีการเรียกตั้งค่าหลายโมดูลชิพจะเลือกค่าที่มากที่สุดเป็นค่าที่ถูกนำมาใช้งาน หากมีช่องว่างระหว่างเวลากับค่า window น้อยกว่า 5ms จะหมายถึงชิพนั้นทำงานตลอดเวลา ถ้าไม่มีการกำหนดค่า window จะทำให้ชิพถูกปลุกเมื่อไม่มีการเชื่อมต่อเมื่อใช้ ESP-Now

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ความรู้พื้นฐานเกี่ยวกับเครือข่ายคอมพิวเตอร์ เครือข่ายไร้สาย การพัฒนาเว็บ และไมโครคอนโทรลเลอร์ถูกผนวกเข้าด้วยกันทำให้สามารถใช้การแสดงผลที่เคยเป็นหน้าจอแอลซีดีหรือหลอดแอลอีดี ถูกเปลี่ยนเป็นแสดงผ่านหน้าเว็บ ยิ่งผู้อ่านสามารถเขียน HTML5 ได้คล่องยิ่งเสริมการทำงานของเว็บให้เข้าสู่ความเป็นเว็บแอป (Web Application) ได้มากขึ้น สามารถแสดงผลและตอบสนองกับผู้ใช้ได้มากขึ้น ส่วน ESP Now มาช่วยอุดช่องโหว่เรื่องของการสื่อสารระหว่างกันของ ESP32 โดยไม่ต้องผ่านเครือข่ายอินเทอร์เน็ต ทำให้สามารถประยุกต์สร้างเครือข่ายของเซ็นเซอร์ เช่น เครือข่ายเซ็นเซอร์ในฟาร์มหรือในโรงงาน เป็นต้น ช่วยให้ผู้พัฒนาระบบมีทางเลือกในการลดปริมาณข้อมูลที่ต้องถูกส่งเข้าไปในเครือข่ายอินเทอร์เน็ตขององค์กรด้วยการให้ ESP32 โหนดหลักคอยตรวจสอบหรือติดตามค่าจากโหนดลูก เมื่อพบความผิดพลาดหรือมีสิ่งที่ต้องการส่งกลับเข้าระบบอินเทอร์เน็ตค่อยสลับให้โหนดหลักเชื่อมต่อเข้าเครือข่ายอินเทอร์เน็ตเพื่อดำเนินการต่อไป และเมื่อทำงานเสร็จค่อยกลับเข้ามาโหมด ESP Now อีกครั้ง เป็นต้น

ในบทถัดไปเป็นการประยุกต์ใช้กับระบบของ Blynk เพื่อให้อุปกรณ์สามารถเชื่อมเจ้าอินเทอร์เน็ต และผู้ใช้สามารถสร้างความสัมพันธ์ระหว่างค่าจากอุปกรณ์กับการแสดงผลหรือสถานะของอุปกรณ์กับการตอบสนองจากคอนโทรลต่าง ๆ

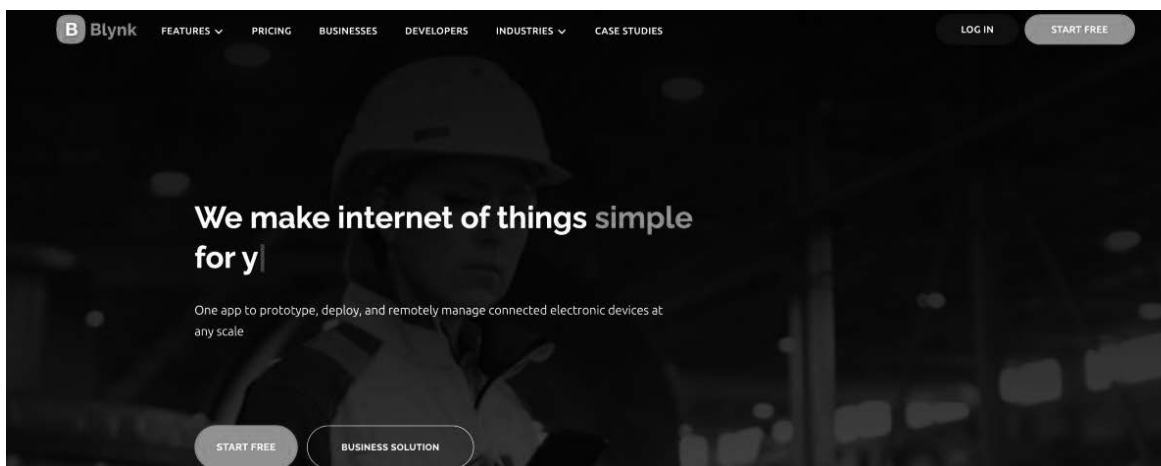
บรรณานุกรม

Espressif System. (2024). *ESP32 Series Datasheet*. สืบค้นเมื่อ พฤษภาคม 28, 2567, จาก https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf



บทที่ 24 Blynk

บทนี้กล่าวถึงแพลตฟอร์ม Blynk ดังภาพที่ 24-1 รวมถึงการใช้งานร่วมกับ ESP32 เพื่ออธิบาย พร้อมทั้งเห็นภาพโดยรวมของวิธีการใช้งานอุปกรณ์ฝังตัวและชุดคำสั่งร่วมกับแพลตฟอร์ม Blynk และการประยุกต์ใช้แพลตฟอร์ม Blynk ในการสื่อสารกับอุปกรณ์สมองกลฝังตัวจากระยะไกล



ภาพที่ 24-1

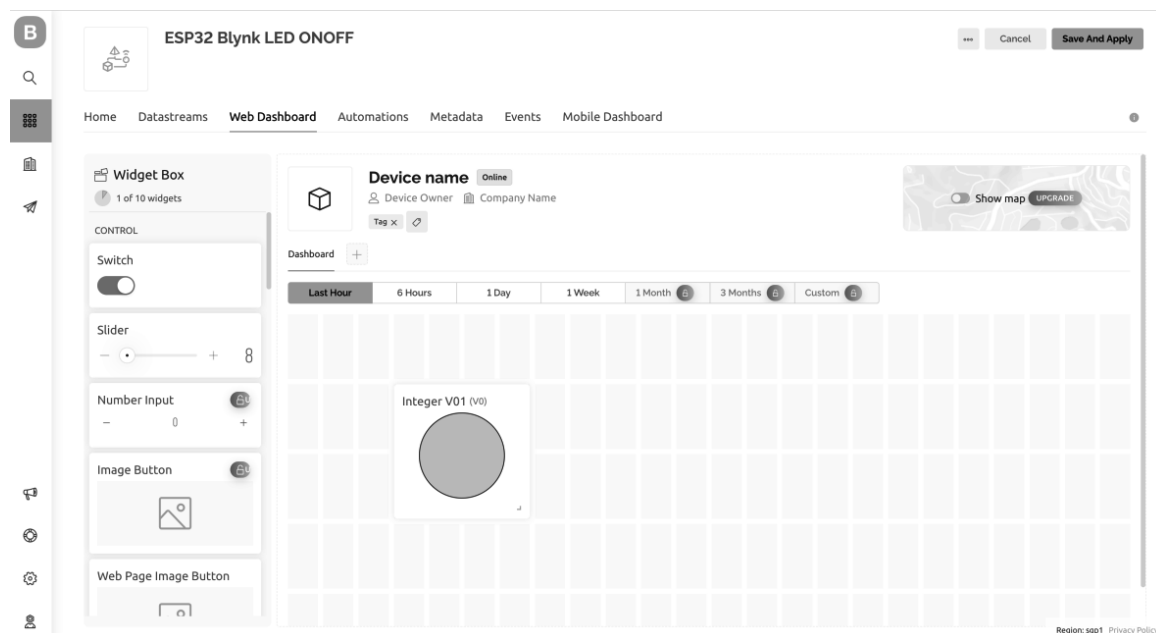
เว็บไซต์ blynk.io

ที่มา: blynk, 2024

> แนะนำ Blynk

ปัจจุบันอินเทอร์เน็ตมีการพัฒนาที่มากขึ้น ทั้งมีความเร็วสูง มีความเสถียรมากขึ้น จึงทำให้สามารถประยุกต์ใช้ร่วมกับระบบสมองกลฝังตัวตามที่ต่าง ๆ ในการรับและส่งข้อมูลเพื่อนำไปต่อยอดใน

การวิเคราะห์ข้อมูล นอกจากนี้ยังสามารถใช้ในการสั่งการสิ่งต่าง ๆ ได้มากขึ้น เช่น การเปิด-ปิดอุปกรณ์ต่าง ๆ แต่อย่างจำเป็นต้องใช้ความรู้ทางเทคนิคและการเขียนโปรแกรมค่อนข้างสูงในการสร้างระบบขึ้นเอง ส่งผลให้มีการพัฒนาแพลตฟอร์มเพื่อช่วยอำนวยความสะดวกในการเชื่อมต่อระบบสมองกลฝังตัวเข้ากับการใช้อินเทอร์เน็ตเกิดขึ้นคือ Blynk ซึ่งนอกจากอำนวยความสะดวกในการเชื่อมต่อระบบสมองกลฝังตัวกับอินเทอร์เน็ตแล้วยังมีการใช้งานที่สะดวก ง่ายต่อการปรับใช้ในสถานการณ์ต่าง ๆ รวมถึงหน้าต่างที่สวยงาม และยังสามารถใช้ได้ทั้งบนคอมพิวเตอร์ในลักษณะเว็บไซต์ดังภาพที่ 24-2 และโทรศัพท์มือถือ โดยมีทั้งฟีเจอร์ที่เสียค่าใช้จ่ายและไม่เสียค่าใช้จ่าย



ภาพที่ 24-2

Dashboard ของ Blynk website

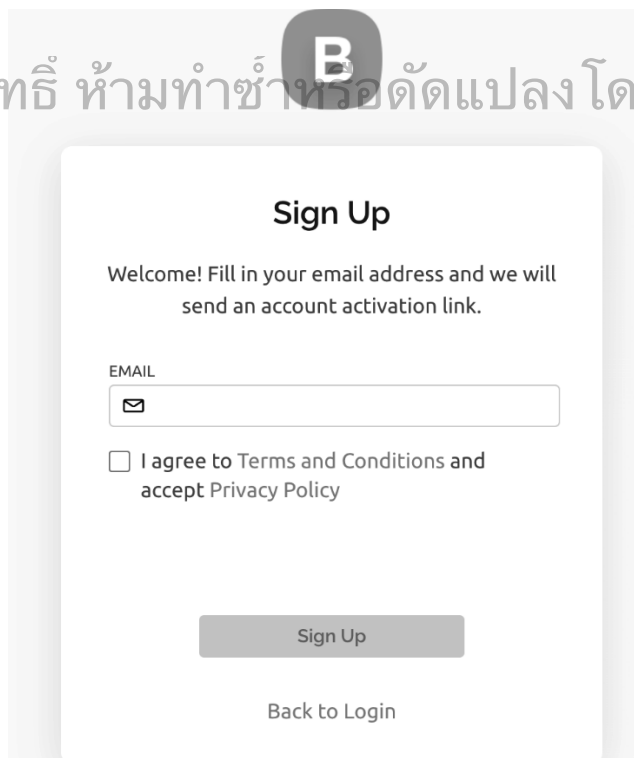
ที่มา: blynk, 2024

> เริ่มใช้งาน Blynk

ในการเริ่มใช้งาน Blynk อันดับแรกจำเป็นต้องมีบัญชีก่อน โดยสามารถสมัครได้ที่ <https://blynk.cloud/dashboard/register> จะมีหน้าต่างดังภาพที่ 24-3

กรอกอีเมลพร้อมทั้งยอมรับเงื่อนไขการใช้งานและกดปุ่ม Sign Up เพื่อรับลิงค์ยืนยันการสมัครที่ส่งให้และต้องทำการยืนยันการสมัครผ่านลิงก์ที่ได้รับภายในเวลาที่กำหนด (1 ชั่วโมง) เพื่อสร้างรหัสผ่านใหม่และสามารถเข้าสู่ระบบได้ที่ <https://blynk.cloud/dashboard/login> ดังภาพที่ 24-4

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

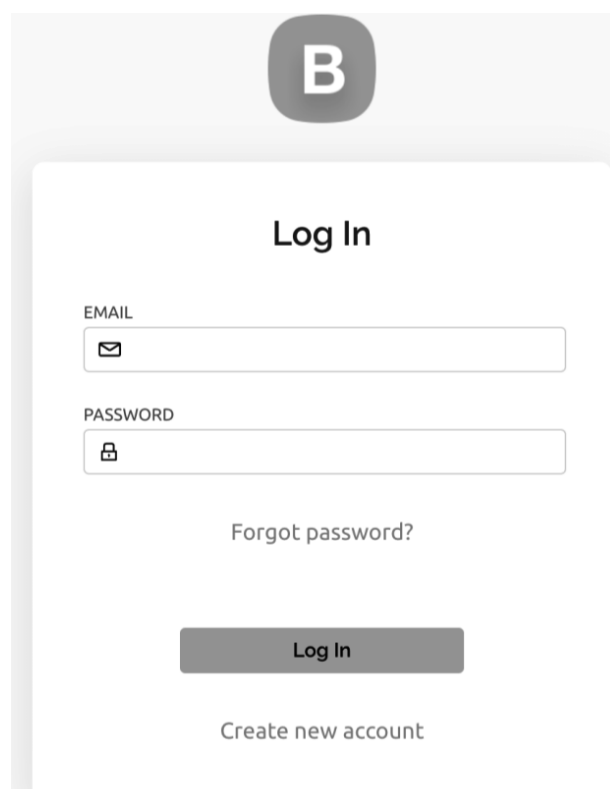


The image shows the Blynk 'Sign Up' screen. At the top is the Blynk logo (a grey circle with a white 'B'). Below it, the title 'Sign Up' is centered. A welcome message reads: 'Welcome! Fill in your email address and we will send an account activation link.' There is a text input field for 'EMAIL' with an envelope icon. Below the field is a checkbox with the text 'I agree to Terms and Conditions and accept Privacy Policy'. At the bottom, there is a grey 'Sign Up' button and a link that says 'Back to Login'.

ภาพที่ 24-3

หน้าสมัครบัญชี Blynk

ที่มา: blynk, 2024



The image shows the Blynk 'Log In' screen. At the top is the Blynk logo (a grey circle with a white 'B'). Below it, the title 'Log In' is centered. There are two text input fields: 'EMAIL' with an envelope icon and 'PASSWORD' with a lock icon. Below the password field is a link that says 'Forgot password?'. At the bottom, there is a grey 'Log In' button and a link that says 'Create new account'.

ภาพที่ 24-4

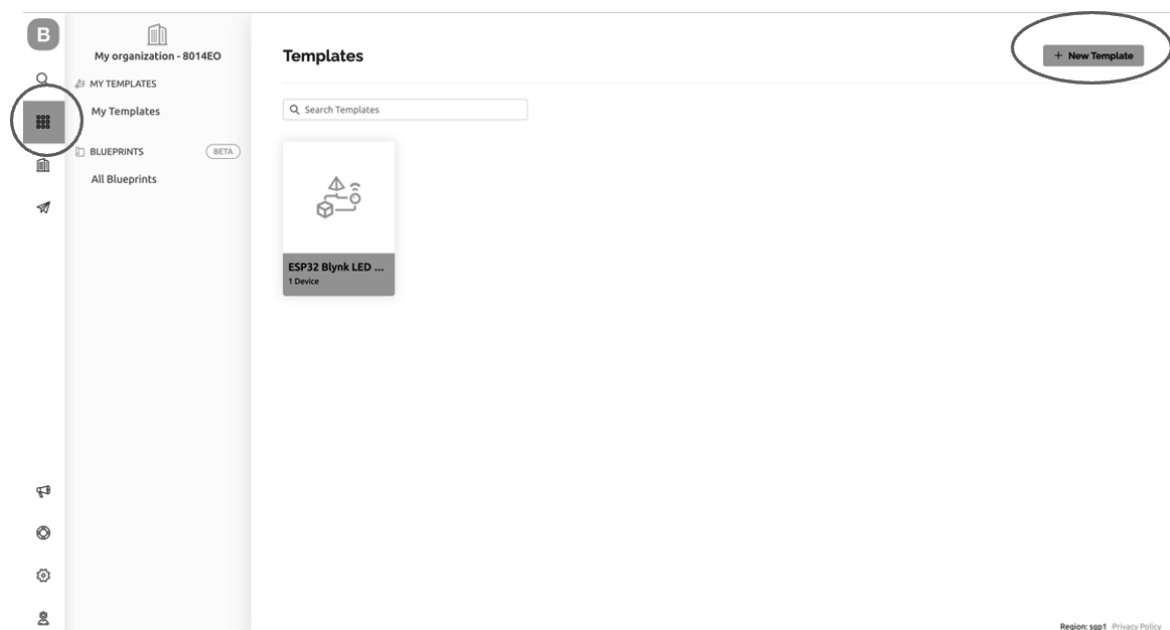
หน้าเข้าสู่ระบบ Blynk

ที่มา: blynk, 2024

> การสร้างเทมเพลต

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

ก่อนที่จะสามารถใช้งานแพลตฟอร์มร่วมกับอุปกรณ์ต่าง ๆ ได้จำเป็นต้องสร้างเทมเพลตหรือตัวต้นแบบสำหรับการสร้างอุปกรณ์ในแพลตฟอร์ม Blynk การตั้งค่าของการรับ-ส่งข้อมูล รวมถึงการจำลองอุปกรณ์เซ็นเซอร์ต่าง ๆ ภายในแพลตฟอร์ม โดยต้องทำการคลิกที่ปุ่ม  เพื่อเข้าสู่หน้าเทมเพลต และคลิกที่ปุ่ม **+ New Template** เพื่อเริ่มสร้างเทมเพลตดังภาพที่ 24-5



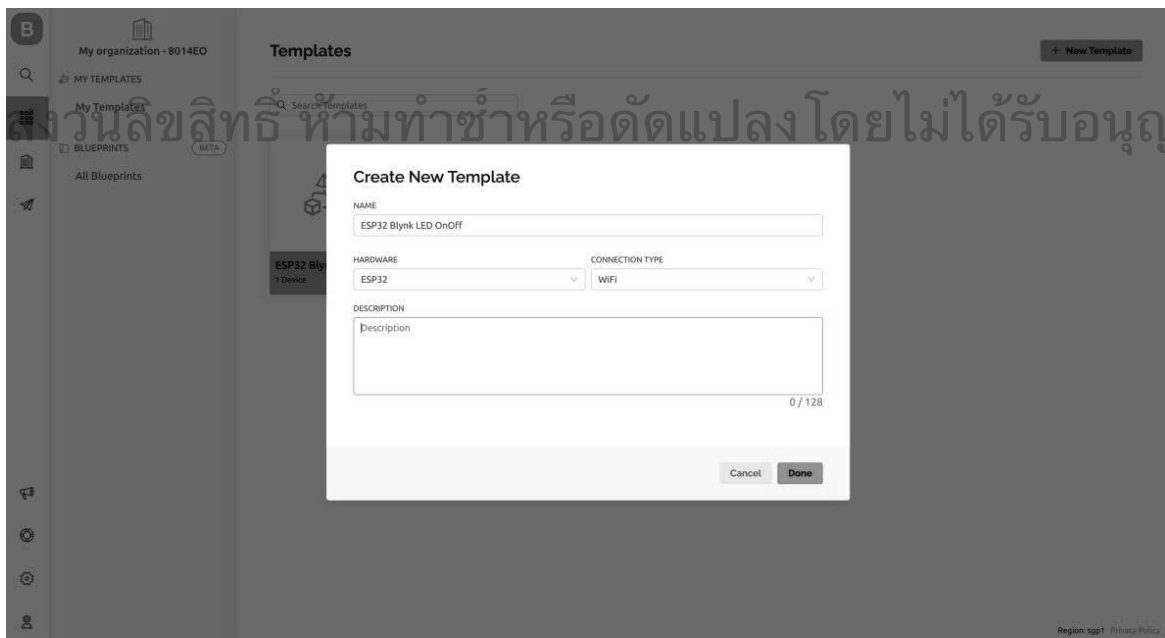
ภาพที่ 24-5

การสร้างเทมเพลต

ที่มา: blynk, 2024

หลังจากนั้นจะปรากฏหน้าต่างการตั้งค่ารายละเอียดของเทมเพลตดังภาพที่ 24-6 ซึ่งต้องกรอกข้อมูลชื่อของเทมเพลต อุปกรณ์ที่ใช้ คือ ESP32 การเชื่อมต่อที่อุปกรณ์คือ WiFi และสุดท้ายคำอธิบายเทมเพลต

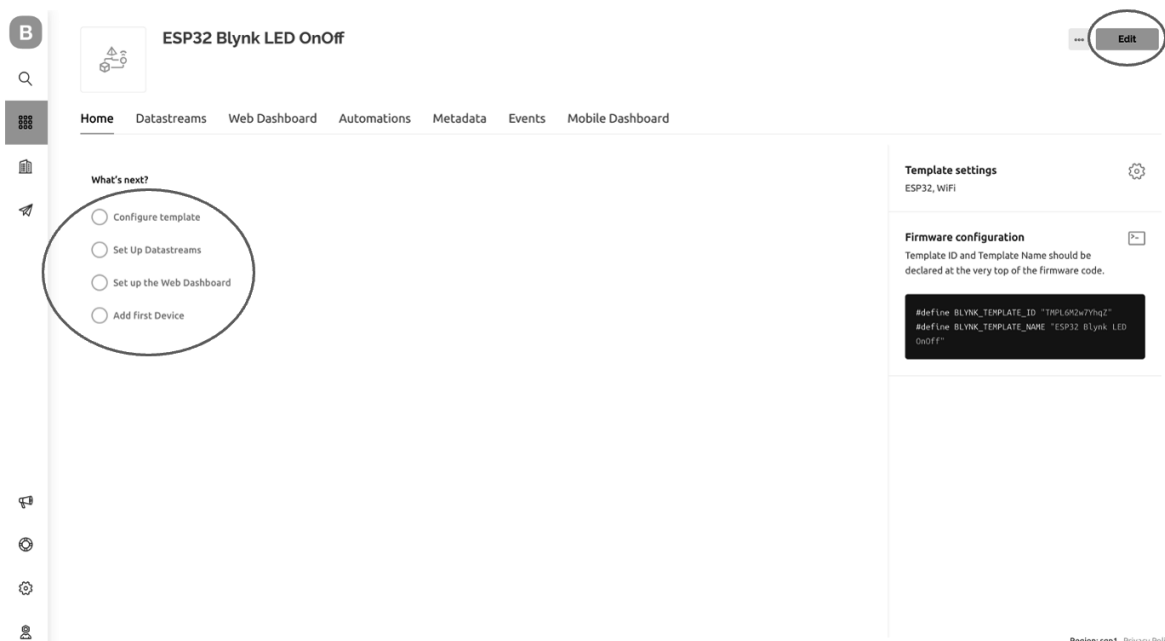
หลังจากสร้างเทมเพลตเสร็จสิ้นจะสามารถคลิกเลือกเทมเพลตที่ต้องการใช้งานได้เพื่อตั้งค่าการใช้งาน เริ่มต้นให้กดที่ปุ่ม **Edit** ด้านขวาบน จะปรากฏหน้าต่างภาพที่ 24-7 สามารถเริ่มทำตามขั้นตอนคือ สร้างอุปกรณ์จำลองจากเทมเพลต กำหนดการใช้งานอุปกรณ์เซ็นเซอร์ต่าง ๆ และเชื่อมต่ออุปกรณ์เซ็นเซอร์เข้ากับกระแสข้อมูล



ภาพที่ 24-6

การตั้งค่ารายละเอียดเริ่มต้นของเทมเพลต

ที่มา: blynk, 2024



ภาพที่ 24-7

การแก้ไขเทมเพลต

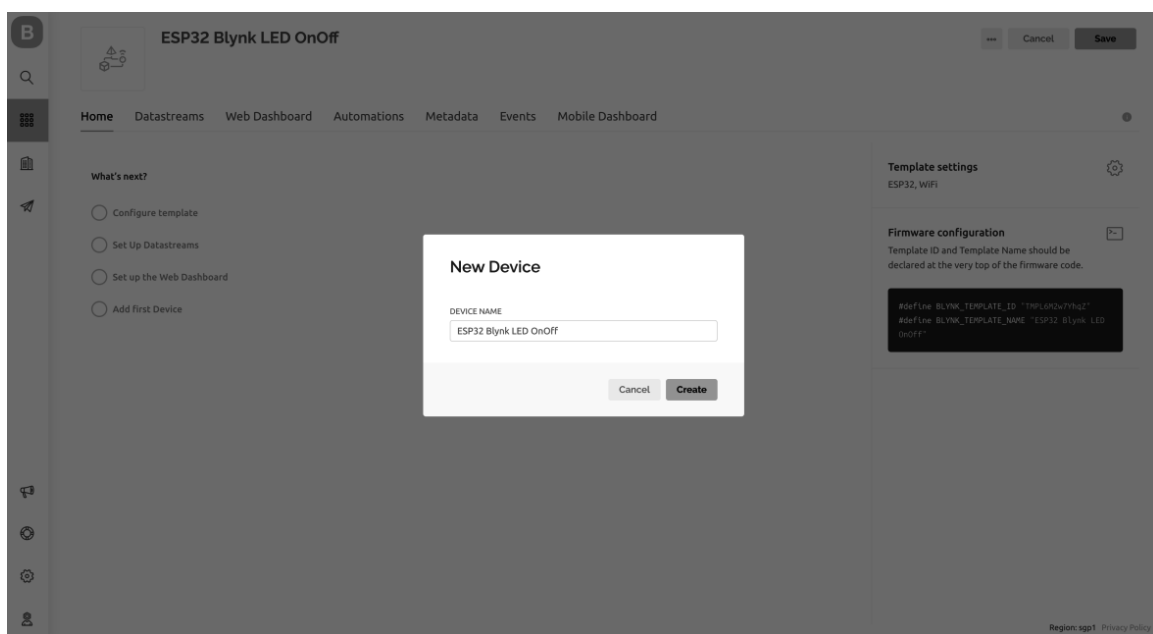
ที่มา: blynk, 2024

> การเพิ่มอุปกรณ์

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

เพิ่มเพลตเป็นเพียงตัวต้นแบบเชิงนามธรรมเท่านั้น ไม่สามารถเชื่อมต่อกับอุปกรณ์จริงได้ จึงจำเป็นต้องจำลองอุปกรณ์ที่อ้างอิงถึงรูปแบบของเพิ่มเพลตโดยกดที่ Add first Device ตั้งชื่ออุปกรณ์และกดที่ปุ่ม **Create** ดังภาพที่ 24-8

หลังจากสร้างอุปกรณ์เสร็จสิ้นจะมีหน้าต่างโค้ดดังภาพที่ 24-9 สำหรับใช้ระบุอุปกรณ์เพื่อให้บอร์ดสามารถเชื่อมต่อกับอุปกรณ์ที่สร้างจาก Blynk ได้ ซึ่งโค้ดนี้ต้องนำไปใช้กับบอร์ดต่อไป



ภาพที่ 24-8

การเพิ่มอุปกรณ์ที่จากเพิ่มเพลต

ที่มา: blynk, 2024

New Device Created! X

```
#define BLYNK_TEMPLATE_ID "TMPL6M2w7YhqZ"
#define BLYNK_TEMPLATE_NAME "ESP32 Blynk LED OnOff"
#define BLYNK_AUTH_TOKEN
"fst0B2m0b_siC82Lgr8aqleuFBqPLZeQ"
```

Template ID, Device Name, and AuthToken should be declared at the very top of the firmware code.

[Documentation](#)
[Copy to clipboard](#)

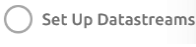
ภาพที่ 24-9

ข้อมูลอุปกรณ์ที่ใช้สำหรับการเชื่อมต่อ ESP32 และ Blynk

ที่มา: blynk, 2024

> การสร้างกระแสข้อมูล

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ในการเชื่อมต่อข้อมูลระหว่าง ESP32 และอุปกรณ์บน Blynk จำเป็นต้องใช้กระแสข้อมูล (Datastreams) เริ่มต้นโดยการกดที่  ในภาพที่ 24-10 หรือแท็บ Datastreams จากนั้นกดที่ปุ่ม  ในภาพที่ 24-11 และเลือก Virtual Pin

What's next?

- ☐ Configure template
- ☐ Set Up Datastreams
- ☐ Set up the Web Dashboard

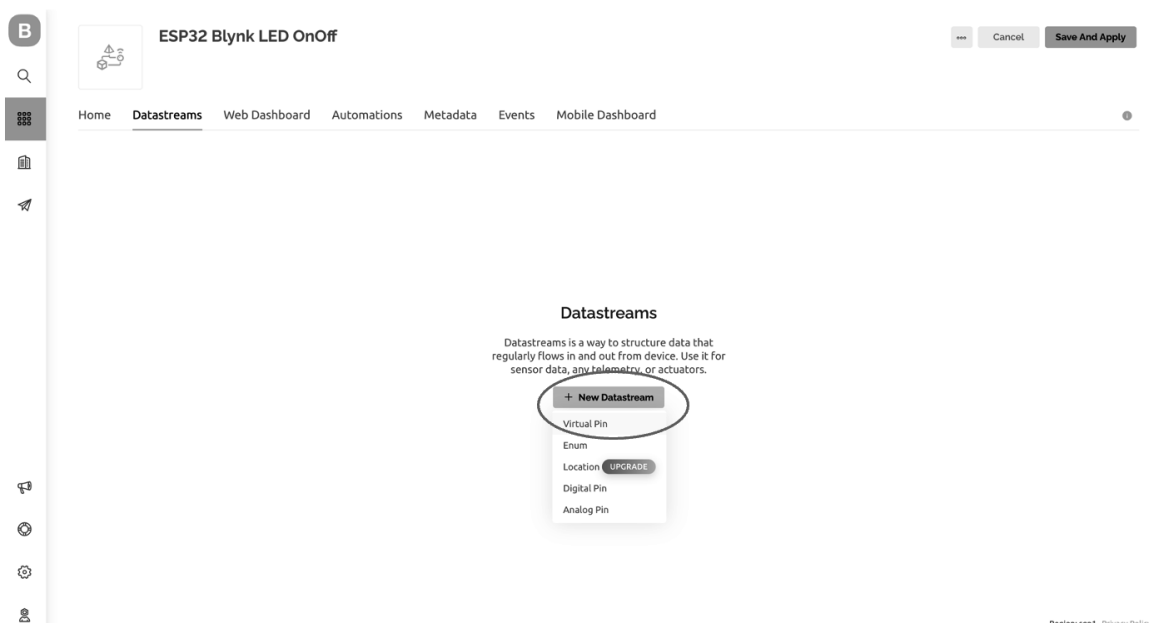
Done:

- ☒ Add first Device

ภาพที่ 24-10

ขั้นตอนการตั้งค่าเพื่อใช้งานเต็มเฟลต

ที่มา: blynk, 2024



ภาพที่ 24-11

การสร้างกระแสข้อมูล

ที่มา: blynk, 2024

เมื่อมีหน้าต่างใหม่ขึ้นนั้นจะสามารถตั้งค่ากระแสข้อมูลได้โดยในตัวอย่างสร้างกระแสข้อมูล 2 ขาคือ LED และ Switch ตั้งค่าด้วยค่าตามรูปภาพที่ 24-12 และ 24-13 โดยมีคำอธิบายตามตารางที่ 24-1 และกดที่ปุ่ม 

Virtual Pin Datastream

NAME	LED		ALIAS	LED	
PIN	V0		DATA TYPE	Integer	
UNITS	None				
MIN	0	MAX	1	DEFAULT VALUE	0

 ADVANCED SETTINGS

Cancel



ภาพที่ 24-12

การตั้งค่ากระแสข้อมูลสำหรับ LED

ที่มา: blynk, 2024

Virtual Pin Datastream

NAME

Switch

ALIAS

Switch

PIN

V1

DATA TYPE

Integer

UNITS

None

MIN

0

MAX

1

DEFAULT VALUE

0

⊕

 ADVANCED SETTINGS

Cancel

Create

ภาพที่ 24-13

การตั้งค่ากระแสข้อมูลสำหรับ Switch

ที่มา: blynk, 2024

ตารางที่ 24-1 หัวข้อและคำอธิบายในการตั้งค่า

ลำดับ	หัวข้อ	คำอธิบาย
1	Name	ชื่อของกระแสข้อมูล
2	ALIAS	ชื่อเรียกอื่นๆ ของกระแสข้อมูล
3	PIN	ตำแหน่งขา
4	DATA TYPE	ประเภทข้อมูลของกระแสข้อมูล
5	UNITS	หน่วยของกระแสข้อมูล
6	MIN	ค่าต่ำที่สุดของกระแสข้อมูล
7	MAX	ค่าสูงที่สุดของกระแสข้อมูล
8	DEFAULT VALUE	ค่าเริ่มต้นของกระแสข้อมูล
9	DECIMALS	จำนวนทศนิยมของกระแสข้อมูล

ที่มา: ดัดแปลงจาก Blynk. (2024). Datastream Settings. สืบค้นเมื่อ พฤษภาคม 28, 2567, จาก <https://docs.blynk.io/en/blynk.console/templates/datastreams/datastreams-common-settings>

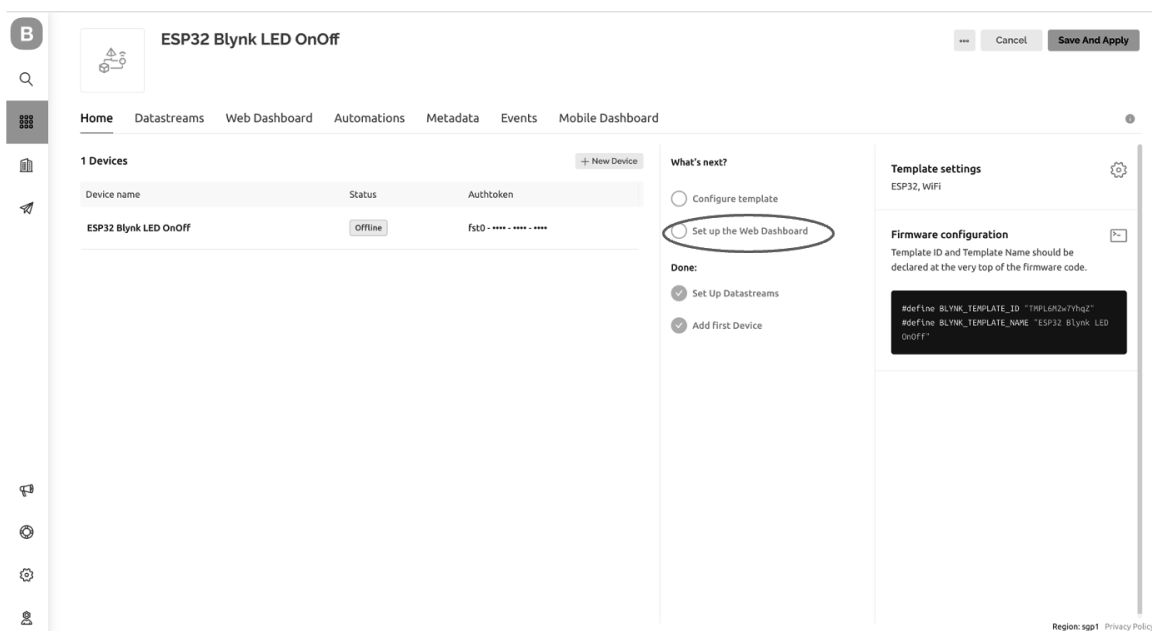
หมายเหตุ

หากเลือกประเภทของกระแสข้อมูลเป็น Double จึงสามารถกำหนดค่า DECIMALS ได้หากเลือกประเภทข้อมูลของกระแสข้อมูลเป็น STRING จะไม่สามารถกำหนดค่า UNITS, MIN, MAX และ DECIMALS ได้

เมื่อมีกระแสข้อมูลแล้วสามารถกำหนดอุปกรณ์เซ็นเซอร์ได้จากที่แท็บ Web Dashboard โดยสามารถกดที่แท็บ Home จากนั้นกดที่  Set up the Web Dashboard ดังภาพที่ 24-14 ซึ่งจะนำไปที่แท็บ Web Dashboard หรือสามารถกดที่แท็บ Web Dashboard โดยตรงก็ได้เช่นกัน

ในแท็บ Web Dashboard ด้านซ้ายคืออุปกรณ์เซ็นเซอร์ที่สามารถนำมาใช้งานได้ และด้านขวาคือการจัดเรียงตำแหน่งอุปกรณ์เซ็นเซอร์ของอุปกรณ์ โดยในตัวอย่างใช้ Switch เพื่อสั่งงาน LED ที่ต่อกับ ESP32 และ LED เพื่อรับการสั่งงานจาก ESP32

เมื่อกำหนดอุปกรณ์เซ็นเซอร์ที่ต้องการใช้เสร็จสิ้นต้องตั้งค่ากระแสข้อมูลให้กับอุปกรณ์แต่ละตัวเพื่อใช้รับ-ส่งข้อมูลจาก Blynk และ ESP32 โดยการตั้งค่ากระแสข้อมูลให้อาเมาส์ซีที่อุปกรณ์ LED จะมีจุดสามจุดขึ้นมา เลื่อนเมาส์ไปชี้จุดสามจุดและกดที่ Settings ดังภาพที่ 24-15



ภาพที่ 24-14

ขั้นตอนการตั้งค่าแผงควบคุม

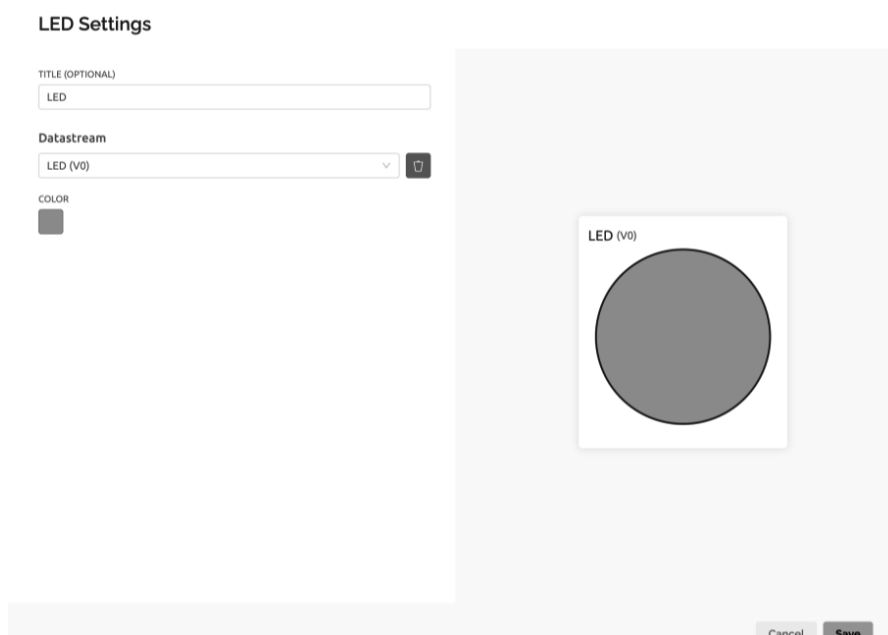
ที่มา: blynk, 2024



ภาพที่ 24-15 การเลือกใช้งานอุปกรณ์เซ็นเซอร์

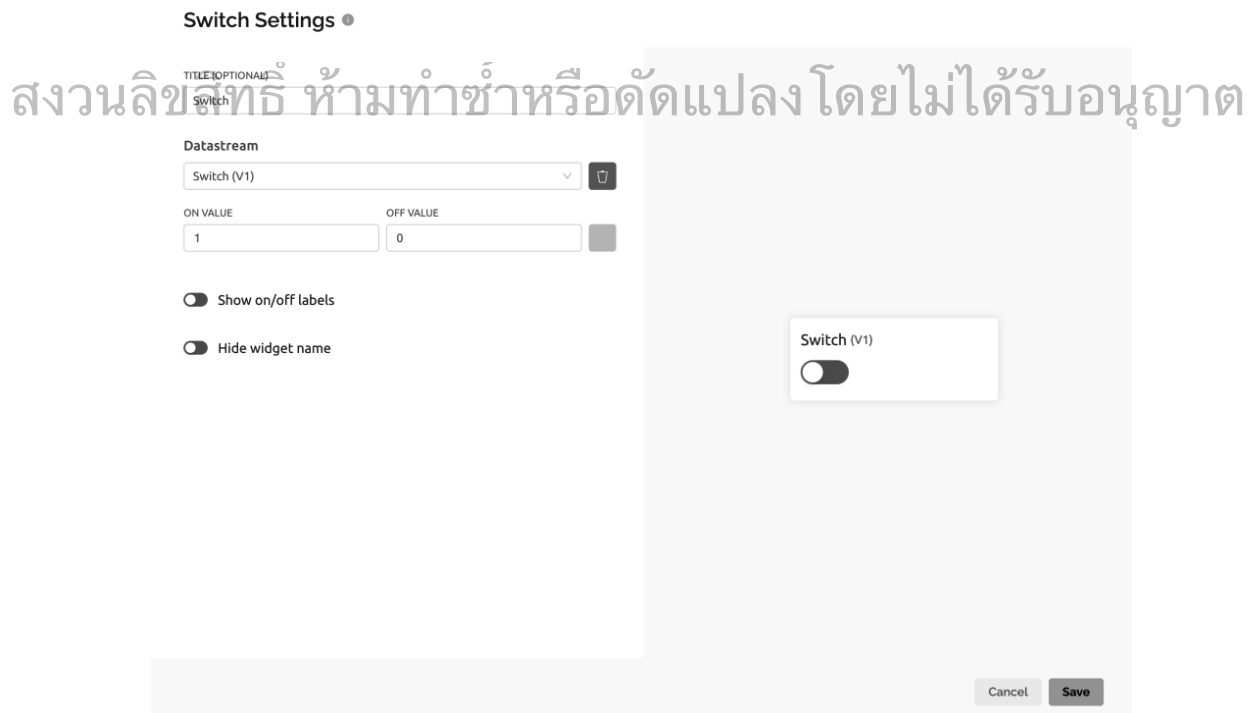
ที่มา: blynk, 2024

ในหน้าต่างใหม่ที่ขึ้นมาให้กำหนด Datastream เป็น LED (V0) ดังภาพที่ 24-16 และกดที่ปุ่ม **Save** สำหรับ Switch ให้ตั้งค่า Datastream เป็น Switch (V1) ดังภาพที่ 24-17 เเท่านี้การตั้งค่าในฝั่ง Blynk ก็เสร็จสิ้นให้ทำการบันทึกทั้งหมดโดยการกดปุ่ม **Save And Apply** ที่ด้านขวาบนของหน้าจอ ดังภาพที่ 24-18



ภาพที่ 24-16 การกำหนดกระแสข้อมูลให้กับ LED

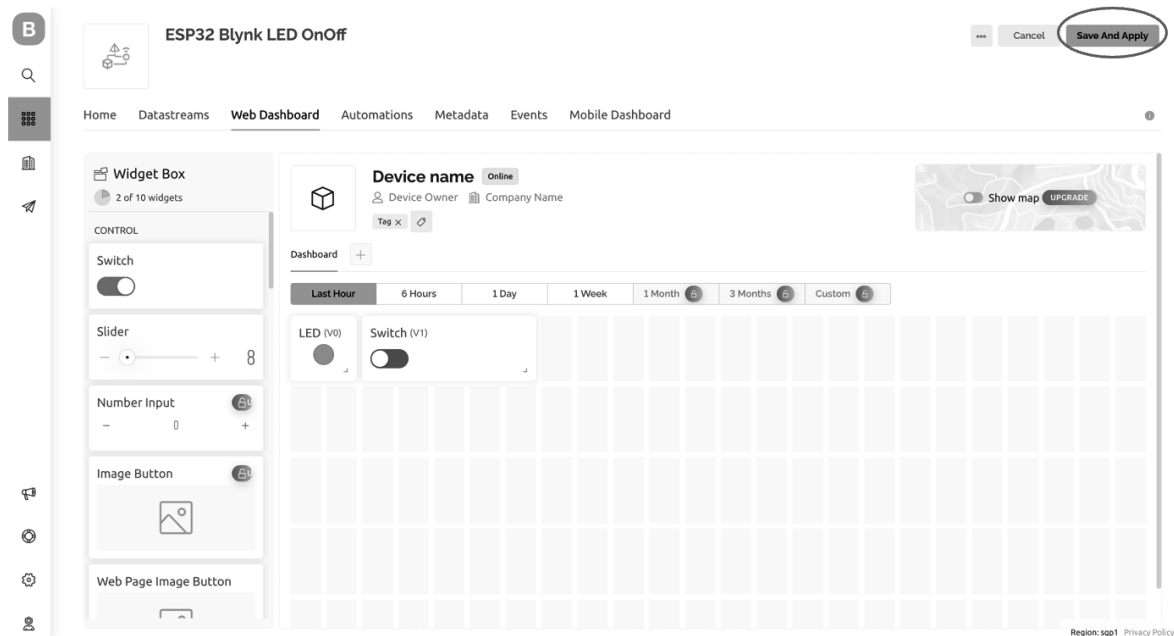
ที่มา: blynk, 2024



ภาพที่ 24-17

การกำหนดกระแสข้อมูลให้กับ Switch

ที่มา: blynk, 2024



ภาพที่ 24-18

การบันทึกการตั้งค่าเพิ่มเติม

ที่มา: blynk, 2024

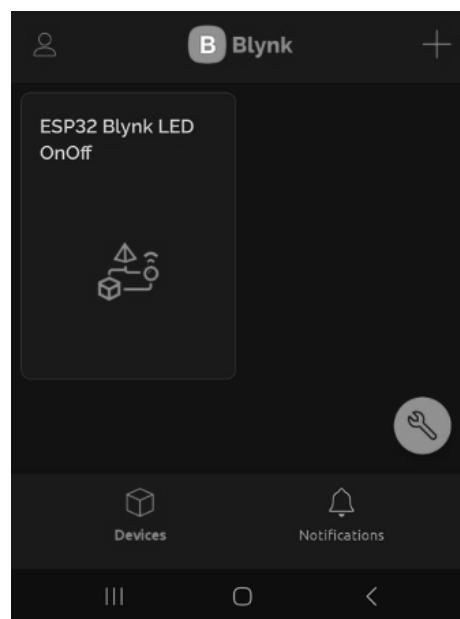
> การสร้างอุปกรณ์บนแอปพลิเคชัน Blynk ในอุปกรณ์พกพา

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

นอกจากการสั่งงานอุปกรณ์ผ่านคอมพิวเตอร์แล้ว เพื่ออำนวยความสะดวกในการใช้งานให้สามารถสั่งการอุปกรณ์ได้ในทุกที่ Blynk จึงมีทั้งเว็บไซต์และแอปพลิเคชัน โดยสามารถดาวน์โหลดแอปพลิเคชันได้ผ่าน Google Play Store สำหรับ Android และ App store สำหรับ iOS

หลังจากโหลดแอปพลิเคชันเสร็จให้ทำการเข้าสู่ระบบด้วยบัญชีเดียวกับที่สมัครข้างต้น หลังจากเข้าสู่ระบบได้เรียบร้อยแล้วจะเห็นเพิ่มเพลตที่ได้สร้างไว้ สามารถเลือกเพิ่มเพลตที่ต้องการได้

จะมีหน้าแสดงอุปกรณ์จากเพิ่มเพลตดังภาพที่ 24-19 ซึ่งกำลังมีการรับและส่งข้อมูลระหว่าง ESP32 และ Blynk เกิดขึ้นเพียงแต่หากไม่ได้มีอุปกรณ์เซ็นเซอร์ใดเชื่อมต่อกับกระแสข้อมูลอยู่ก็จะไม่เห็นการเปลี่ยนแปลงของอุปกรณ์ หากต้องการแก้ไขอุปกรณ์ให้กดที่ปุ่มประแจสีเขียวด้านขวาล่าง ดังภาพที่ 24-20 จากนั้นกดที่เครื่องหมายบวกด้านขวาบน สามารถเลือกอุปกรณ์เซ็นเซอร์ที่ต้องการได้ โดยมีทั้งแบบฟรีและเสียค่าใช้จ่ายดังภาพที่ 24-21 ซึ่งในตัวอย่างมีเพียง LED เท่านั้น

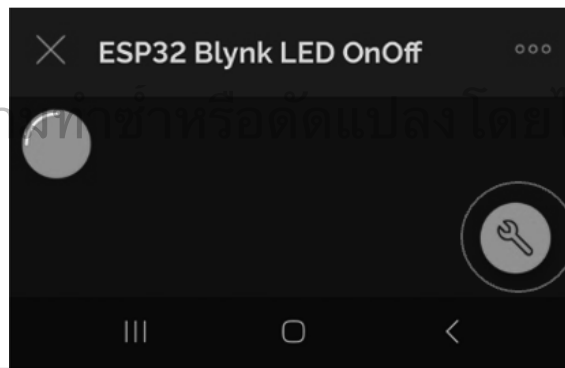


ภาพที่ 24-19

หน้าแสดงอุปกรณ์จากเพิ่มเพลตบนอุปกรณ์พกพา

ที่มา: blynk, 2024

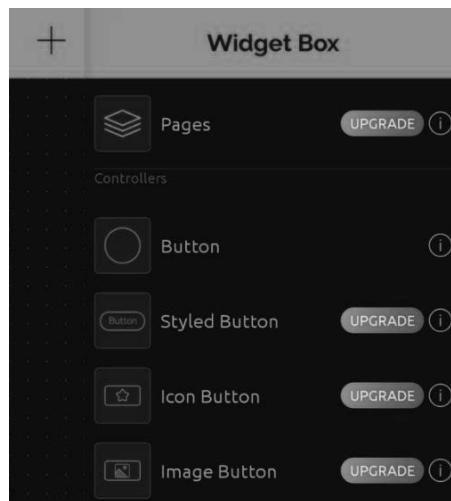
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต



ภาพที่ 24-20

แผงควบคุมอุปกรณ์

ที่มา: blynk, 2024



ภาพที่ 24-21

อุปกรณ์เซ็นเซอร์บางส่วนที่สามารถเลือกใช้งานได้

ที่มา: blynk, 2024

> ตัวอย่างโปรแกรม

สำหรับการเขียนโปรแกรมเชื่อมต่อระหว่าง ESP32 และ Blynk สามารถใช้ Arduino IDE ได้โดยมีรายละเอียดของโปรแกรกดังภาพที่ 24-22

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต



ภาพที่ 24-22

ข้อมูลโปรแกรม Arduino IDE

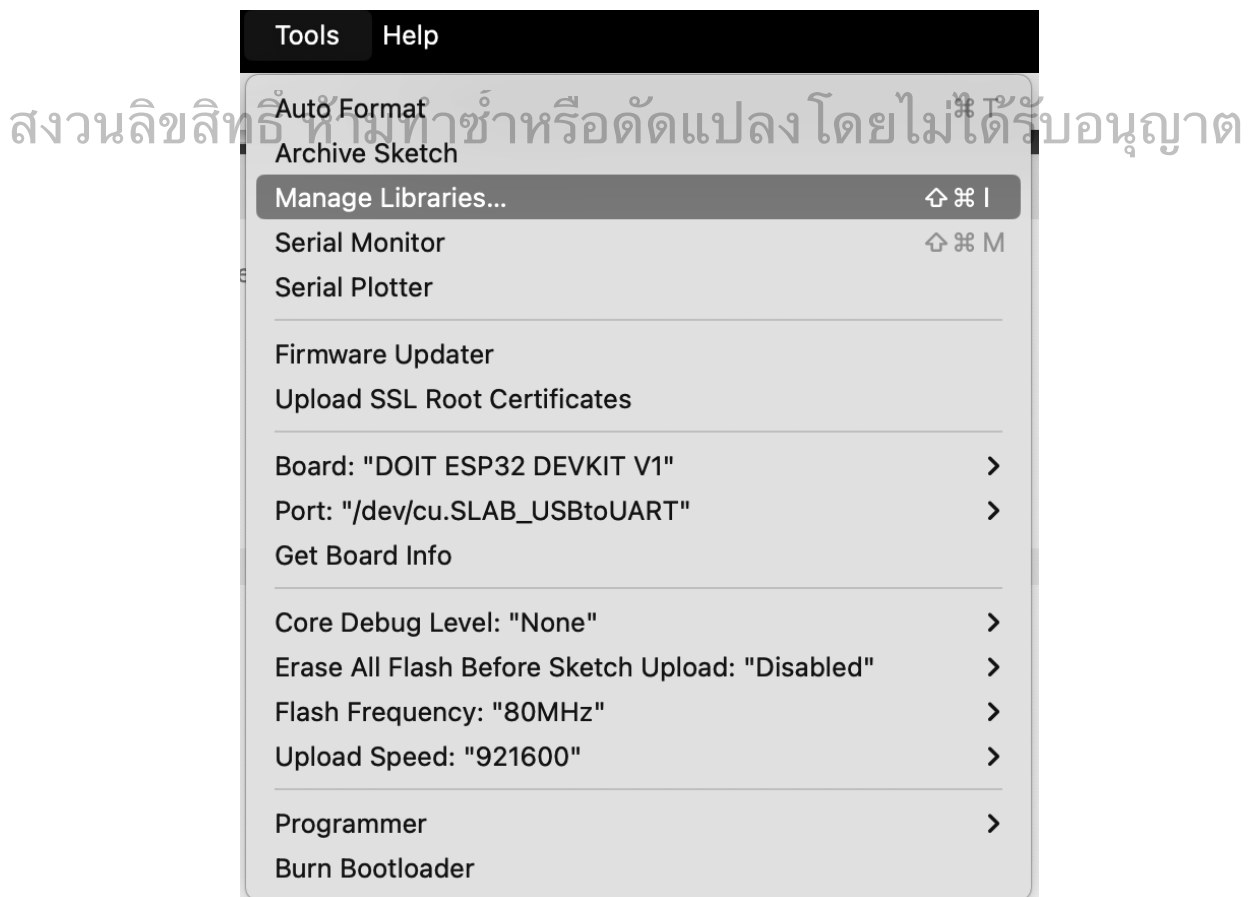
ที่มา: Arduino, 2024

เตรียมพร้อมโปรแกรม

ก่อนการใช้งานจำเป็นต้องมีการลงไลบรารีเพื่อให้โปรแกรมนู้จักกับ Blynk ก่อน โดยคลิกที่เมนู Tools -> Manage Libraries ดังภาพที่ 24-23

พิมพ์ blynk ในช่องค้นหาจะพบไลบรารี Blynk by Volodymyr Shymanskyy โดยสามารถระบุเวอร์ชันได้ซึ่งในตัวอย่างใช้เวอร์ชัน 1.3.2 และคลิกที่ปุ่ม INSTALL ดังภาพที่ 24-24

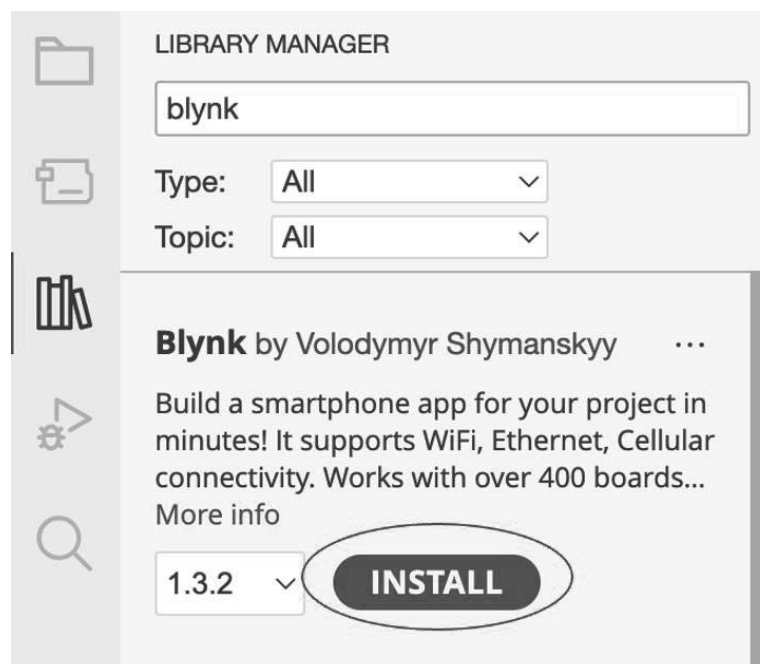
เมื่อมีการติดตั้งไลบรารีที่จำเป็นต้องใช้ไลบรารีอื่น ๆ โปรแกรมจะมีการแจ้งเตือนให้ติดตั้งไลบรารีที่ต้องใช้ทั้งหมดเพื่อให้การทำงานสมบูรณ์ คลิกที่ปุ่ม INSTALL ALL เมื่อติดตั้งเสร็จสิ้นก็สามารถใช้งาน ESP32 ร่วมกับ Blynk ได้แล้ว โดยกำหนด Board เป็น DOIT ESP32 DEVKIT V1 ดังภาพที่ 24-25 และ Port เป็นพอร์ตที่เชื่อมต่อกับ ESP32



ภาพที่ 24-23

การจัดการไลบรารี

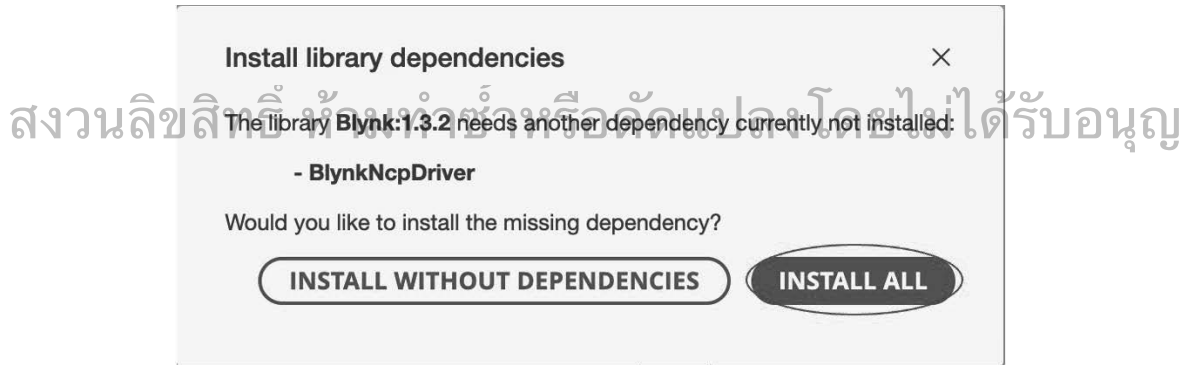
ที่มา: blynk, 2024



ภาพที่ 24-24

ติดตั้งไลบรารี Blynk

ที่มา: blynk, 2024



ภาพที่ 24-25

ติดตั้งไลบรารีเพิ่มเติมที่จำเป็น

ที่มา: blynk, 2024

ตัวอย่างโปรแกรมเริ่มต้น

ไลบรารีที่ได้ติดตั้งมีตัวอย่างโปรแกรมให้ในเบื้องต้นโดยสามารถใช้ได้โดยคลิกที่เมนู File -> Examples -> Blynk -> Boards_WiFi -> ESP32_WiFi_SSL และแก้ไขโค้ดโดยลบการคอมเม้นในบรรทัดที่ 2, 3 และ 4 เพื่อให้มีการกำหนดตัวแปรที่จำเป็นต้องใช้งาน โดยข้อมูลของแต่ละคนจะแตกต่างกัน ซึ่งในตัวอย่างใช้ข้อมูลตามภาพที่ 24-9 จากนั้นตั้งค่า ssid[] ด้วยชื่อไวไฟของแต่ละคน และตั้งค่า pass[] ด้วยรหัสผ่านที่กำหนด หากใครมีอินเทอร์เน็ตของมือถือสามารถใช้อุปกรณ์ไวไฟเพื่อทดลองได้

ตัวอย่างโปรแกรม 24-1 ตัวอย่างเริ่มต้นจากไลบรารี Blynk	
บรรทัด	คำสั่ง
1	#define BLYNK_PRINT Serial
2	#define BLYNK_TEMPLATE_ID "TMPL6M2w7YhqZ"
3	#define BLYNK_TEMPLATE_NAME "ESP32 Blynk LED OnOff"
4	#define BLYNK_AUTH_TOKEN "fst0B2m0b_siC82Lgr8aqleuFBqPLZeQ"
5	#include <WiFi.h>
6	#include <WiFiClientSecure.h>
7	#include <BlynkSimpleEsp32_SSL.h>
8	char ssid[] = "YourNetworkName";
9	char pass[] = "YourPassword";
10	void setup()
11	{
12	Serial.begin(9600);
13	Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
14	}

ตัวอย่างโปรแกรม 24-1 ตัวอย่างเริ่มต้นจากไลบรารี Blynk (ต่อ)

บรรทัด	คำสั่ง
15	void loop()
16	{
17	Blynk.run();
18	}

เมื่อกำหนดการตั้งค่าเสร็จสิ้นสามารถอัปโหลดโค้ดลงบอร์ดของได้ทันที หากการเชื่อมต่อสำเร็จ
 ดังภาพที่ 24-26 ในแท็บ Home บนเว็บไซต์จะแสดงว่าอุปกรณ์มีการออนไลน์ดังภาพที่ 24-27

หมายเหตุ

สามารถดูข้อมูลเพิ่มพลตได้ที่แท็บ Home ของเพิ่มพลต และข้อมูล Auth Token ได้ที่อุปกรณ์ในหน้าเดียวกัน

Output

```
Writing at 0x00096b9b... (62 %)
Writing at 0x0009be9b... (65 %)
Writing at 0x000a1256... (68 %)
Writing at 0x000a656f... (71 %)
Writing at 0x000abc3b... (74 %)
Writing at 0x000b130e... (77 %)
Writing at 0x000b6ac3... (80 %)
Writing at 0x000bc8f1... (82 %)
Writing at 0x000c223b... (85 %)
Writing at 0x000ca7c9... (88 %)
Writing at 0x000d2e11... (91 %)
Writing at 0x000d7f67... (94 %)
Writing at 0x000dda25... (97 %)
Writing at 0x000e32c7... (100 %)
Wrote 875136 bytes (564413 compressed) at 0x00010000 in 8.4 seconds (effective 835.4 kbit/s)...
Hash of data verified.
```

Leaving...

Hard resetting via RTS pin...

ภาพที่ 24-26

อัปโหลดโค้ดสำเร็จ

ที่มา: blynk, 2024

HomeDatastreamsWeb DashboardAutomationsMetadataEventsMobile Dashboard

1 Devices

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

Device name

Status

Auth token

ESP32 Blynk LED OnOff

Online

fst0 - - -

ภาพที่ 24-27

สถานะออนไลน์ของอุปกรณ์

ที่มา: blynk, 2024

ตัวอย่างโปรแกรมรับส่งข้อมูล

การรับส่งข้อมูลระหว่าง Blynk และบอร์ดสามารถใช้ Virtual Pin ซึ่งมีข้อดีคือสามารถใช้กับบอร์ดหลายแบบได้ หรือ Digital Pin หรือ Analog Pin ซึ่งมีข้อดีคือการเขียนโปรแกรมจะน้อยกว่า โดยตัวอย่างนี้เป็นการรับและส่งข้อมูลโดยใช้ Virtual Pin V0 เพื่อส่งข้อมูลจากบอร์ดไปยัง Blynk และ V1 เพื่อรับข้อมูลจาก Switch ที่ได้กำหนดไว้บนอุปกรณ์ของ Blynk

ตัวอย่างโปรแกรม 24-2 ตัวอย่างโปรแกรมรับส่งข้อมูล

บรรทัด	คำสั่ง
1	#define BLYNK_PRINT Serial
2	#define BLYNK_TEMPLATE_ID "TMPL6M2w7YhqZ"
3	#define BLYNK_TEMPLATE_NAME
4	"ESP32 Blynk LED OnOff"
5	#define BLYNK_AUTH_TOKEN
6	"fst0B2m0b_siC82Lgr8aqleuFBqPLZeQ"
7	#include <WiFi.h>
8	#include <WiFiClientSecure.h>
9	#include <BlynkSimpleEsp32_SSL.h>
10	char ssid[] = "YourNetworkName";
11	char pass[] = "YourPassword";
12	void setup()
13	{
14	Serial.begin(9600);
15	Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
16	pinMode(2, OUTPUT);
17	}

342

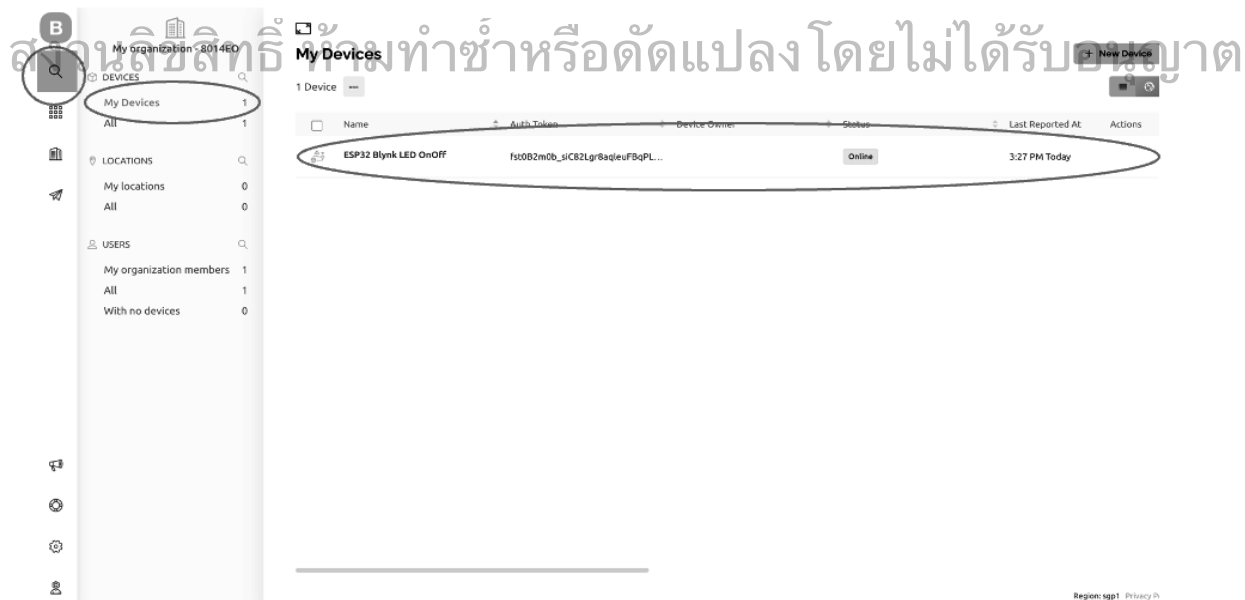
ETT

ตัวอย่างโปรแกรม 24-2 ตัวอย่างโปรแกรมรับส่งข้อมูล (ต่อ)

บรรทัด	คำสั่ง
18	void loop()
19	{
20	Blynk.run();
21	Blynk.virtualWrite(V0, LOW);
22	delay(1000);
23	Blynk.virtualWrite(V0, HIGH);
24	delay(1000);
25	}
26	BLYNK_WRITE(V1)
27	{
28	if(param.asInt() == 1)
29	{
30	digitalWrite(2,HIGH); // Set digital pin 2 HIGH
31	}
32	else
33	{
34	digitalWrite(2,LOW); // Set digital pin 2 LOW
35	}
36	}

โปรแกรมมีการสั่งงานให้ LED ของอุปกรณ์บน Blynk มีการติดและดับสลับกันทุก 1 วินาที และมีการรับการสั่งงานจากอุปกรณ์บน Blynk ด้วยฟังก์ชัน BLYNK_WRITE(V1) โดยที่ V1 คือ Virtual Pin ที่ได้กำหนดให้กับ Switch และฟังก์ชันจะทำงานเมื่อมีการเปลี่ยนค่า ผลการทำงานเป็นดังภาพที่ 24-30 และภาพที่ 24-31

ในการส่งข้อมูลจากอุปกรณ์บน Blynk จะต้องไปที่หน้าแผงควบคุมของอุปกรณ์เท่านั้น ไม่สามารถสั่งงานในหน้าเว็บได้ โดยสามารถเข้าหน้าอุปกรณ์ได้โดยกดที่เมนูค้นหาทางด้านซ้าย เลือก My Devices และเลือกอุปกรณ์ที่ต้องการสั่งการดังภาพที่ 24-28

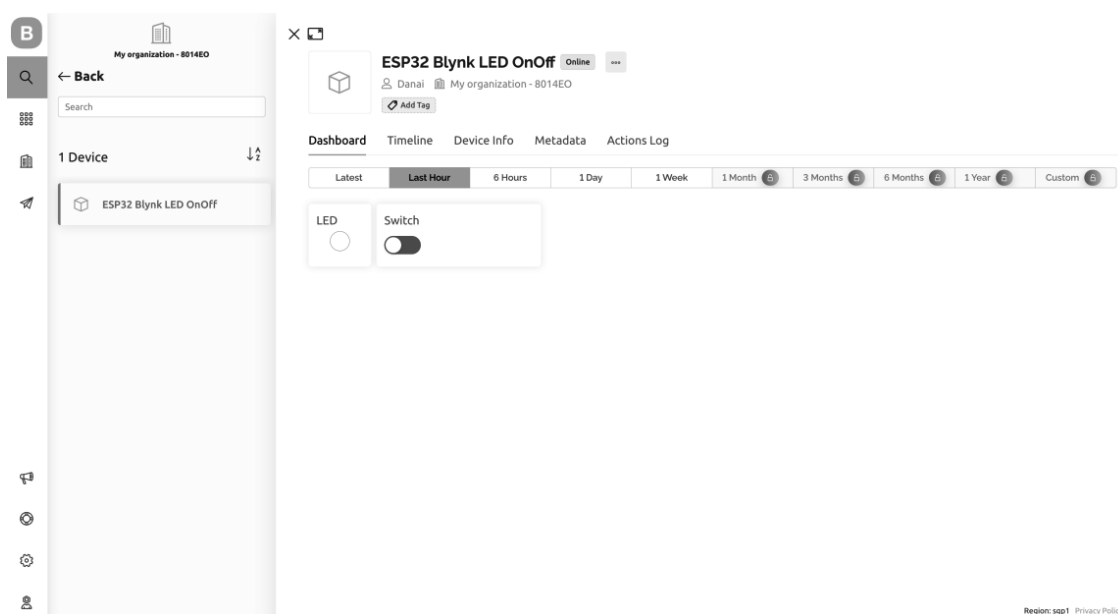


ภาพที่ 24-28

การเลือกอุปกรณ์บน Blynk ที่ต้องการสั่งการ

ที่มา: blynk, 2024

หลังจากเลือกอุปกรณ์ที่ต้องการใช้สั่งงานเชื่อมต่อกับ ESP32 จะมีข้อมูลอุปกรณ์และแผงควบคุมอุปกรณ์แสดงอยู่ดังภาพที่ 24-29 สามารถสั่งงานจากหน้านี้ได้เช่น ลองกดเปิดสวิตช์และดูผลลัพธ์ว่า LED ที่อุปกรณ์มีไฟติดหรือไม่ และเมื่อกดปิดสวิตช์ LED ดับหรือไม่ ดังภาพที่ 24-30 และภาพที่ 24-31



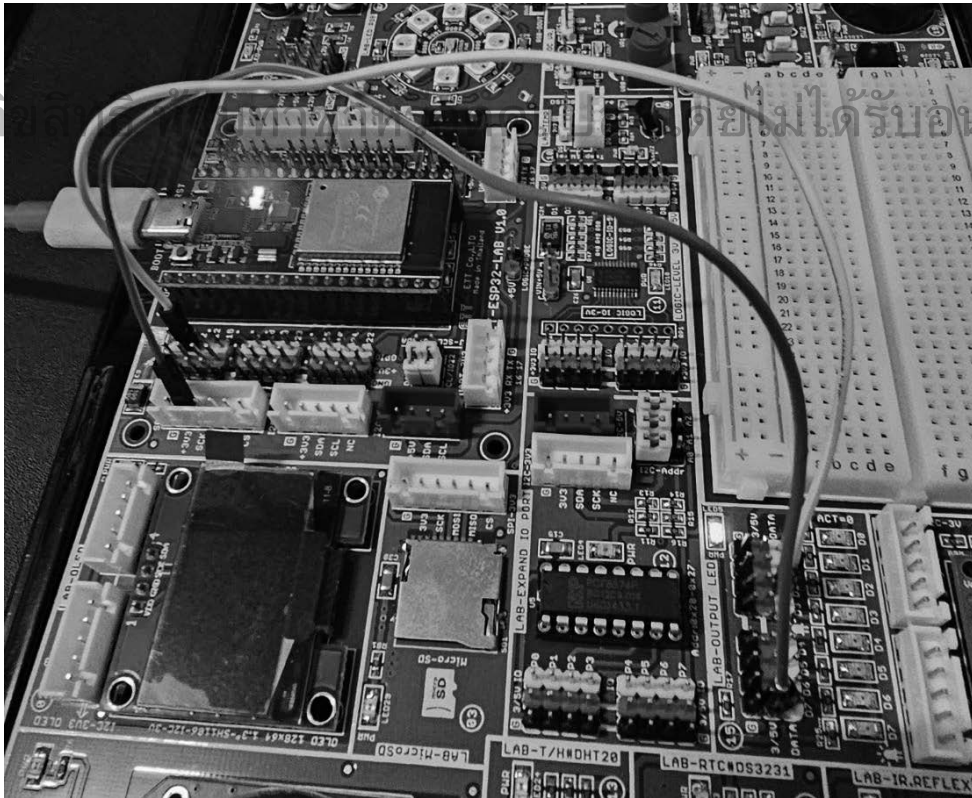
ภาพที่ 24-29

แผงควบคุมอุปกรณ์

ที่มา: blynk, 2024

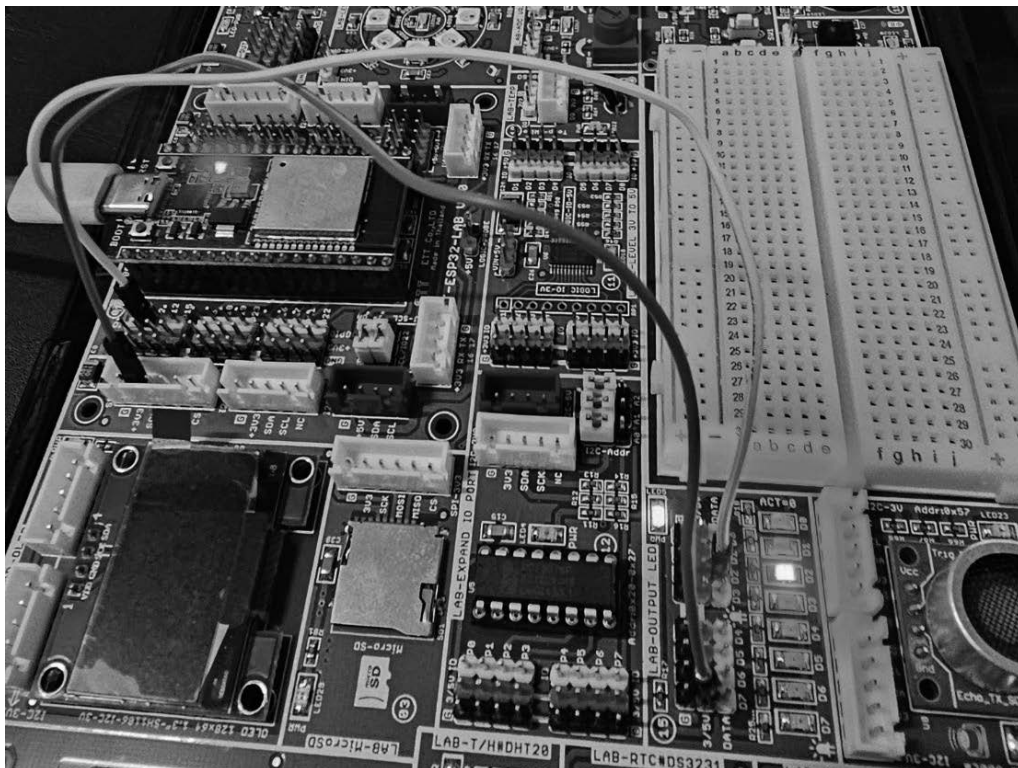
สงวนลิขสิทธิ์

โดยไม่ได้รับอนุญาต



ภาพที่ 24-30 LED ดับเมื่อสวิตช์ที่แผงควบคุมอุปกรณ์บน Blynk ปิดอยู่

ภาพโดย จารุต บุศราทิจ



ภาพที่ 24-31 LED ติดเมื่อสวิตช์ที่แผงควบคุมอุปกรณ์บน Blynk เปิดอยู่

ภาพโดย จารุต บุศราทิจ

> สรุป

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลงโดยไม่ได้รับอนุญาต

ในบทนี้ได้เรียนรู้เกี่ยวกับแพลตฟอร์ม Blynk ที่ช่วยในการรับและส่งข้อมูลระยะไกลได้ ทำให้สามารถสั่งการอุปกรณ์ได้จากระยะไกลด้วยความสะดวกมากขึ้น โดยแพลตฟอร์มสามารถตั้งค่าการแจ้งเตือนและกำหนดกระแสข้อมูลให้กับอุปกรณ์ที่ต้องการได้ มีการจำลองกระแสข้อมูลในรูปแบบมาตรฐาน ทำให้ไม่ยึดติดกับบอร์ดใดบอร์ดหนึ่ง และยังสามารถใช้ได้ทั้งรูปแบบเว็บไซต์และบนโทรศัพท์มือถือแบบสมาร์ทโฟน

บรรณานุกรม

Blynk. (2024). *Blynk Documentation*. สืบค้นเมื่อ พฤษภาคม 30, 2567, จาก

<https://docs.blynk.io/en/>

Blynk. (2024). *Datastream Settings*. สืบค้นเมื่อ พฤษภาคม 30, 2567, จาก

<https://docs.blynk.io/en/blynk.console/templates/datastreams/datastreams-common-settings>

Arduino. (2024). *Arduino IDE*. สืบค้นเมื่อ พฤษภาคม 30, 2567, จาก

<https://docs.arduino.cc/software/ide/>



สงวนลิขสิทธิ์ ห้ามทำซ้ำ



ดัชนี

หน้า

A	
Application Layer	285
Auth Token.....	329
B	
BSSID.....	296
Buzzer	99
Blynk Platform.....	324
D	
Datastream	330
Direct Memory Access.....	10
DS3231	213
E	
ESP32_Servo	94
Embedded System.....	324
ESP-NOW.....	314
F	
FS.h	264
G	
GC9A01.....	247

H

Harvard Architecture	2
HTTP	288

I

IR_REMOTE	25
IR Reflex	169
ITR-9908	174

L

Logic Level Converter	16
-----------------------------	----

M

Memory Management Unit	11
MicroSD Card	261

R

RCWL-1605	189
Relay	84
RS232	54

S

Servo Motor	91
System on Chips	1
SSD1306	207
SPI Modes	184

T

TCP/IP Model	287
Tensilica Xtensa	22
TFT_eSPI	247
TSOP4838	126

V

Virtual Pin	331
Variable Resistor	44

W

Watchdog Timer.....	46
WS2812B.....	77
สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต	
กระแสน้ำ 330	
การจัดการไลบรารี.....	339
การตั้งค่าเครือข่าย.....	289
การถอดรหัสอินฟราเรด.....	128
การทายตัวเลข.....	123
การส่งรหัสตอบกลับ.....	307
จ	
จุดเชื่อมต่อเครือข่าย.....	298
ด	
ตัวต้านทานปรับค่าได้.....	148
ถ	
แถบความถี่.....	202
บ	
บัลลูนและชุดคำสั่ง.....	26
ร	
ระบบสมองกลฝังตัว.....	324
รหัสยืนยันตัวตน.....	329
ส	
สวิตช์แบบกดติดปล่อยดับ.....	116
ห	
หน่วยความจำแบบแฟลช.....	16
ผ	
แผงควบคุมอุปกรณ์.....	333



ประวัติผู้เขียน



อาจารย์ดำนัย เจษฎาฐิติกุล

อาจารย์ประจำสาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏเพชรบุรี

ประวัติการศึกษา

วิทยาศาสตร์มหาบัณฑิต (วท.ม.) สาขาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

วิทยาศาสตร์บัณฑิต (วท.บ.) สาขาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยราชภัฏเพชรบุรี

ประวัติการทำงาน

พ.ศ. 2559-2564 อาจารย์ประจำคณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏเพชรบุรี

พ.ศ. 2565-2567 อาจารย์ประจำสาขาวิชานวัตกรรมดิจิทัล คณะเทคโนโลยีสารสนเทศ
มหาวิทยาลัยราชภัฏเพชรบุรี

พ.ศ. 2567-ปัจจุบัน อาจารย์ประจำสาขาวิชาเทคโนโลยีสารสนเทศ คณะเทคโนโลยีสารสนเทศ
มหาวิทยาลัยราชภัฏเพชรบุรี

ผลงานวิจัย

มีผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่ในวารสารวิชาการในระดับชาติ /นานาชาติ

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต

- 1) มนูญชัย กิตติธนเกษม, อัสนีวัลย์ อินทร์คำ, ศิริพร อ่วมศิริ, และ **ดนัย เจษฎาฐิติกุล**. (2021). เกมเสริมทักษะการเรียนรู้อักษรไทยไตรยางศ์ ด้วยหลักการเขียนโปรแกรมเชิงวัตถุ. The 7th National Conference on Technology and Innovation Management 2021 (NCTIM2021). (หน้า1893-1900).มหาสารคาม: มหาวิทยาลัยราชภัฏมหาสารคาม.
- 2) **ดนัย เจษฎาฐิติกุล**, สุกุม่า อ่วมเจริญ, จารุต บุศราทิจและทัตทอง พรหมณี. (2024). ต้นแบบการจำลองพฤติกรรมการเล่นโคสำหรับศูนย์เรียนรู้การเพิ่มประสิทธิภาพการผลิตสินค้าเกษตร อำเภอชะอำ. การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 47 THE 47TH ELECTRICAL ENGINEERING CONFERENCE (EECON-47) วันที่ 13-15 พฤศจิกายน 2567 ณ โรงแรมเซ็นทาราและคอนเวนชันเซ็นเตอร์ จังหวัดอุดรธานี (อยู่ระหว่างการดำเนินการ)
- 3) สุกัญชลิกา บุญมาธรรม, ศิริพร อ่วมศิริ, พีรศุขย์ บุญมาธรรม, ปราโมทย์ ตงฉิน, **ดนัย เจษฎาฐิติกุล**, กรกรต เจริญผล และ ญาณิศา อัดถาวะระ. (2024). การพัฒนาสื่อแอนิเมชันเพื่อส่งเสริมการท่องเที่ยวเชิงวัฒนธรรมการร่อนทอง ตำบลร่อนทอง อำเภอบางสะพาน จังหวัดประจวบคีรีขันธ์. PBRU SCIENCE JOURNAL. 2(2).
- 4) จารุต บุศราทิจ, **ดนัย เจษฎาฐิติกุล**, สุวัฒน์ เตชะเพชรไพบูลย์ และจุฑาภรณ์ ชาตินฤมาณ" สมาร์ทฟาร์มแบบไอโอทีสำหรับควบคุมคุณภาพฟาร์มนกนางแอ่นกินรัง" ในการประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ NCCIT ครั้งที่ 15 ณ Arnoma Grand Hotel Bangkok Thailand วันที่ 4-5 กรกฎาคม 2562
- 5) จารุต บุศราทิจ, **ดนัย เจษฎาฐิติกุล**, จิรวิทย์ แก้วโกศล "การพัฒนาระบบความเป็นจริงเสริม (AR) สำหรับแนะนำเส้นทางท่องเที่ยวภายในโครงการศึกษาวิจัยและพัฒนาสิ่งแวดล้อมผักเป็ด อันเนื่องมาจากพระราชดำริ รัชการที่ 9 บ้านแหลม จังหวัดเพชรบุรี พ.ศ. 2561" ในการนำเสนอผลงานวิชาการระดับชาติ มหาวิทยาลัยราชภัฏภูเก็ต ครั้งที่ 12 วันที่ 20 ตุลาคม พ.ศ. 2562 "การวิจัยสหวิทยาการบนพื้นที่ ฐานยุค 5.0" หน้า 518-545

สงวนลิขสิทธิ์ ห้ามทำซ้ำหรือดัดแปลง โดยไม่ได้รับอนุญาต



สรรคุณค่าวิชาการ สู่สังคม
Knowledge to All
www.cupress.chula.ac.th