	แผนการจัดการเรียนรู้	หน่วยที่2
	รหัสวิชา 21901-2007 วิชา เทคโนโลยีระบบสมองกลฝังตัวและไอโอที	สอนครั้งที่ 3 - 7
	ชื่อหน่วยการเรียนรู้ อุปกรณ์แสดงผลผลลัพธ์ (Display Device)	ทฤษฎี 4 ชม. ปฏิบัติ 16 ชม.
ชื่อเรื่อง/งาน แสดงผลข้อมูลจาก ESP32 ด้วยอุปกรณ์แสดงผล		

1. ผลลัพธ์การเรียนรู้ระดับหน่วยการเรียนรู้

- 1.1 สามารถอธิบายหลักการทำงานและประเภทของอุปกรณ์แสดงผลได้ถูกต้อง
- 1.2 สามารถเชื่อมต่อและเขียนโปรแกรมควบคุม 7-Segment Display ทั้งแบบธรรมดาและแบบ I2C ได้
- 1.3 สามารถเชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 1.4 สามารถประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้อย่างเหมาะสม
- 1.5 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ

2. อ้างอิงมาตรฐาน/เชื่อมโยงกลุ่มอาชีพ

-

3. สมรรถนะประจำหน่วย

- 3.1 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ
- 3.2 ทักษะการเขียนโปรแกรมควบคุมอุปกรณ์แสดงผล
- 3.3 ทักษะการใช้โปรโตคอลการสื่อสาร I2C
- 3.4 ทักษะการแก้ไขปัญหาและ Debugging

4. จุดประสงค์เชิงพฤติกรรม

- 4.1 อธิบายหลักการทำงานของ 7-Segment Display แบบธรรมดาและแบบ I2C ได้
- 4.2 อธิบายโครงสร้างและการทำงานของ LCD 16x2 แบบ I2C ได้
- 4.3 เชื่อมต่อวงจร 7-Segment Display แบบธรรมดากับ ESP32 ได้ถูกต้อง
- 4.4 เขียนโปรแกรมแสดงผลตัวเลขบน 7-Segment ได้
- 4.5 เชื่อมต่อและใช้งาน 7-Segment Display แบบ I2C 4 หลักได้
- 4.6 เชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 4.7 ประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้

5. สารการเรียนรู้

5.1 อุปกรณ์แสดงผล 7-Segment Display

- โครงสร้างและหลักการทำงานของ 7-Segment
- การต่อวงจรแบบ Common Cathode และ Common Anode
- การแสดงตัวเลข 0-9 และตัวอักษรพื้นฐาน
- การควบคุมด้วย GPIO ของ ESP326. **กิจกรรมการเรียนรู้**

5.2 การใช้งาน 7-Segment แบบ I2C

- หลักการทำงานของโปรโตคอล I2C
- การเชื่อมต่อ 7-Segment Display 4 หลักแบบ I2C
- ข้อดีของการใช้ I2C (ประหยัด GPIO pins)
- Library สำหรับควบคุม 7-Segment I2C

5.3 LCD 16x2 แบบ I2C

- โครงสร้างและหลักการทำงานของ LCD
- การเชื่อมต่อ LCD แบบ I2C Module
- คำสั่งพื้นฐานในการแสดงผล
- การจัดรูปแบบการแสดงผล

5.4 การรับสัญญาณจากสวิตช์

- การต่อวงจร Pull-up และ Pull-down
- การอ่านค่าสัญญาณดิจิทัลจากสวิตช์
- เทคนิค Debouncing
- การใช้ Interrupt กับสวิตช์

5.5 การประยุกต์ใช้งาน

- ระบบนับขึ้น-ลง
- การควบคุมการกระพริบ
- การจัดการหลายอุปกรณ์แสดงผล

ชั้นสอน (ทฤษฎี 4 ชม.)

เนื้อหา: 7-Segment Display แบบธรรมดา

โครงสร้างของ 7-Segment (a-g segments + decimal point) ความแตกต่างระหว่าง Common Cathode และ Common Anode วงจรการต่อใช้งานกับ ESP32 ตารางรหัสการแสดงผลตัวเลข 0-9 ตัวอย่างโค้ดพื้นฐาน 2C Protocol และ LCD 16x2 โปรโตคอล I2C และการทำงาน 7-Segment I2C และ LCD

กิจกรรมที่ 1: 7-Segment แบบธรรมดาแสดงผลนับตัวเลข

ศึกษาวงจรและต่อวงจร

เขียนโปรแกรมพื้นฐาน

ทดสอบและแก้ไข

กิจกรรมที่ 2: 7-Segment แบบธรรมดาพร้อมสวิตช์ควบคุม

ศึกษาวงจรและต่อวงจร

เขียนโปรแกรมพื้นฐาน

ทดสอบและแก้ไข

กิจกรรมที่ 3: 7-Segment I2C 4 หลัก + สวิตช์ควบคุม

ศึกษาวงจรและต่อวงจร

เขียนโปรแกรมพื้นฐาน

ทดสอบและแก้ไข

กิจกรรมที่ 4: LCD 16x2 I2C แสดงผลข้อมูล

ศึกษาวงจรและต่อวงจร

เขียนโปรแกรมพื้นฐาน

ทดสอบและแก้ไข

7. สื่อและแหล่งการเรียนรู้

เอกสารประกอบการสอน เรื่อง "อุปกรณ์แสดงผลผลลัพธ์ (Display Device)"

สไลด์นำเสนอ

Video สาธิตการต่อวงจรและการเขียนโปรแกรม

ใบงาน/ใบความรู้

8. หลักฐานการเรียนรู้

8.1 หลักฐานความรู้

ใบงานทฤษฎี เรื่อง แสดงผลข้อมูลจาก ESP32 ด้วยอุปกรณ์แสดงผล

แบบทดสอบก่อนเรียน-หลังเรียน

สมุดบันทึกการเรียนรู้ (Learning Log)

ใบความรู้ที่มีการจดบันทึกและสรุปประเด็นสำคัญ

8.2 หลักฐานการปฏิบัติงาน

ไฟล์โค้ดโปรแกรม (.ino) ที่สามารถทำงานได้ถูกต้อง

Video/ภาพถ่ายการทำงานตามเงื่อนไขที่กำหนด

9. การวัดและประเมินผล

9.1 เกณฑ์การปฏิบัติงาน

ด้านความรู้ (40%)

อธิบายหลักการทำงาน ได้ถูกต้อง ครบถ้วน

ด้านทักษะ (40%)

ต่อวงจรถูกต้องตามหลักการ ไม่เกิดไฟช็อต

เขียนโปรแกรมควบคุมได้อย่างถูกต้อง มีโครงสร้างเป็นระบบ

ด้านเจตคติและกิจนิสัย (20%)

มีความรับผิดชอบต่อการใช้อุปกรณ์และส่งงานตรงเวลา

9.2 วิธีการประเมิน

สังเกตพฤติกรรมการปฏิบัติงานระหว่างเรียน

ประเมินจากชิ้นงานและไฟล์โค้ดโปรแกรม

ประเมินจากรายงานผลการปฏิบัติงาน

9.3 เครื่องมือประเมิน

แบบประเมินทักษะการปฏิบัติงาน

Check List การส่งงาน

แบบประเมินการนำเสนอผลงาน (Presentation Rubric)

10. บันทึกผลหลังการจัดการเรียนรู้

10.1 ผลการจัดการเรียนรู้ที่เกิดขึ้นกับผู้เรียน

ผู้เรียนส่วนใหญ่เข้าใจหลักการทำงานของคำสั่งพื้นฐานและสามารถประยุกต์ใช้ได้

ผู้เรียนแสดงความสนใจและกระตือรือร้นในการทดลองสร้างรูปแบบการทำงานใหม่ๆ

10.2 ปัญหา อุปสรรคที่พบ

ผู้เรียนบางคนมีพื้นฐานด้านการเขียนโปรแกรมน้อย ทำให้เข้าใจช้ากว่าเพื่อน

เวลาปฏิบัติไม่เพียงพอสำหรับผู้เรียนที่ต้องการทดลองเพิ่มเติม

ผู้เรียนบางคนไม่ส่งรายงานตรงเวลา

10.3 การแก้ไขปัญหา

1) ผลการแก้ไขปัญหาที่ส่งผลลัพธ์ที่ดีต่อผู้เรียน

จัดเตรียม Video Tutorial ให้ผู้เรียนสามารถทบทวนได้ด้วยตนเอง


เพิ่มเวลาห้องปฏิบัติการเปิดเพื่อให้ผู้เรียนมาฝึกฝนเพิ่มเติมนอกเวลาเรียน

เน้นย้ำการตรวจสอบวงจรก่อนเปิดไฟทุกครั้ง

2) แนวทางแก้ปัญหาในครั้งต่อไป

พัฒนาสื่อการเรียนรู้ออนไลน์เพิ่มเติม เช่น Interactive Simulation

กำหนดไทม์ไลน์การส่งงานที่ชัดเจนและติดตามอย่างใกล้ชิด

	ใบความรู้ ที่ 2	หน่วยที่2
	รหัสวิชา 21901-2007 วิชา เทคโนโลยีระบบสมองกลฝังตัวและไอโอที	สอนครั้งที่ 3 - 7
	ชื่อหน่วยการเรียนรู้ อุปกรณ์แสดงผลผลลัพธ์ (Display Device)	ทฤษฎี 4 ชม. ปฏิบัติ 16 ชม.
ชื่อเรื่อง/งาน แสดงผลข้อมูลจาก ESP32 ด้วยอุปกรณ์แสดงผล		

1. ผลลัพธ์การเรียนรู้ระดับหน่วยการเรียนรู้

- 1.1 สามารถอธิบายหลักการทำงานและประเภทของอุปกรณ์แสดงผลได้ถูกต้อง
- 1.2 สามารถเชื่อมต่อและเขียนโปรแกรมควบคุม 7-Segment Display ทั้งแบบธรรมดาและแบบ I2C ได้
- 1.3 สามารถเชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 1.4 สามารถประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้อย่างเหมาะสม
- 1.5 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ

2. อ้างอิงมาตรฐาน/เชื่อมโยงกลุ่มอาชีพ

-

3. สมรรถนะประจำหน่วย

- 3.1 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ
- 3.2 ทักษะการเขียนโปรแกรมควบคุมอุปกรณ์แสดงผล
- 3.3 ทักษะการใช้โปรโตคอลการสื่อสาร I2C
- 3.4 ทักษะการแก้ไขปัญหาและ Debugging

4. จุดประสงค์เชิงพฤติกรรม

- 4.1 อธิบายหลักการทำงานของ 7-Segment Display แบบธรรมดาและแบบ I2C ได้
- 4.2 อธิบายโครงสร้างและการทำงานของ LCD 16x2 แบบ I2C ได้
- 4.3 เชื่อมต่อวงจร 7-Segment Display แบบธรรมดากับ ESP32 ได้ถูกต้อง
- 4.4 เขียนโปรแกรมแสดงผลตัวเลขบน 7-Segment ได้
- 4.5 เชื่อมต่อและใช้งาน 7-Segment Display แบบ I2C 4 หลักได้
- 4.6 เชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 4.7 ประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้

5. เนื้อหาสาระ

5.1 อุปกรณ์แสดงผล 7-Segment Display

โครงสร้างและหลักการทำงาน

7-Segment Display เป็นอุปกรณ์แสดงผลตัวเลขที่นิยมใช้กันอย่างแพร่หลายในอุปกรณ์อิเล็กทรอนิกส์ต่างๆ เช่น นาฬิกาดิจิทัล เครื่องชั่งน้ำหนัก เครื่องวัดอุณหภูมิ และอุปกรณ์ที่ต้องการแสดงผลตัวเลข โดย 7-Segment Display ประกอบไปด้วยหลอด LED จำนวน 7 ดวง ที่จัดเรียงในลักษณะรูปตัวเลข 8 โดยแต่ละส่วนของ LED เรียกว่า "Segment" และมีการตั้งชื่อด้วยตัวอักษร a ถึง g ตามตำแหน่งของแต่ละส่วน นอกจากนี้ยังมี decimal point หรือจุดทศนิยมที่อยู่มุมล่างขวาของจอแสดงผลอีกด้วย

การทำงานของ 7-Segment Display อยู่บนหลักการของการเปิด-ปิดหลอด LED แต่ละดวง เมื่อเราต้องการแสดงตัวเลขใดตัวเลขหนึ่ง เราจะควบคุมให้หลอด LED บางดวงติด และบางดวงดับ ตามรูปแบบที่กำหนดไว้ ตัวอย่างเช่น หากต้องการแสดงตัวเลข "0" เราจะต้องเปิดหลอด LED ทุกดวง ยกเว้น segment ตรงกลาง (segment g) หรือหากต้องการแสดงตัวเลข "1" เราจะเปิดเฉพาะ segment b และ c เท่านั้น

7-Segment Display มีอยู่ 2 ประเภทหลักๆ ตามลักษณะการต่อวงจรภายใน คือ แบบ Common Cathode (CC) และแบบ Common Anode (CA) ในแบบ Common Cathode นั้น ขาลบ (Cathode) ของหลอด LED ทุกดวงจะถูกต่อรวมเข้าด้วยกัน และต่อเข้ากับกราวด์ (GND) การควบคุมการเปิด-ปิด LED จะทำได้โดยการส่งสัญญาณ HIGH (3.3V หรือ 5V) ไปยัง segment ที่ต้องการให้ติด และส่งสัญญาณ LOW (0V) ไปยัง segment ที่ต้องการให้ดับ ส่วนแบบ Common Anode นั้นตรงกันข้าม โดยขาบวก (Anode) ของหลอด LED ทุกดวงจะถูกต่อรวมกัน และต่อเข้ากับแหล่งจ่ายไฟบวก การควบคุมจะต้องส่งสัญญาณ LOW เพื่อให้ LED ติด และส่งสัญญาณ HIGH เพื่อให้ LED ดับ

สำหรับการใช้งานกับไมโครคอนโทรลเลอร์อย่าง ESP32 นั้น แบบ Common Cathode จะสะดวกกว่า เพราะ Logic การทำงานเป็นแบบ Active High คือ ส่ง 1 เพื่อเปิด ส่ง 0 เพื่อปิด ซึ่งตรงไปตรงมาและเข้าใจง่ายกว่า อย่างไรก็ตาม ทั้งสองแบบสามารถใช้งานได้ดี เพียงแต่ต้องปรับ Logic ในโปรแกรมให้เหมาะสม

การเลือกใช้ 7-Segment Display ควรพิจารณาถึงขนาดของตัวเลขที่ต้องการแสดง ความสว่างของหลอด LED และกระแสไฟฟ้าที่ใช้งาน โดยทั่วไปแล้ว 7-Segment Display แต่ละ segment จะใช้กระแสประมาณ 10-20 mA ดังนั้นหากเปิด LED ทุกดวง (แสดงตัวเลข 8) จะใช้กระแสรวมประมาณ 140 mA ซึ่งต้องแน่ใจว่าแหล่งจ่ายไฟรองรับได้

การเชื่อมต่อกับ ESP32

การเชื่อมต่อ 7-Segment Display เข้ากับ ESP32 จำเป็นต้องใช้ตัวต้านทานจำกัดกระแสสำหรับแต่ละ segment เพื่อป้องกันไม่ให้กระแสไฟฟ้าที่ไหลผ่านหลอด LED มากเกินไปจนทำให้เสียหาย การคำนวณค่าตัวต้านทานที่เหมาะสมใช้สูตร $R = (V_{cc} - V_f) / I_f$ โดยที่ V_{cc} คือแรงดันไฟฟ้าจากขา GPIO ของ ESP32 ซึ่งมีค่า 3.3V, V_f คือแรงดันตกคร่อมหลอด LED (Forward Voltage) ซึ่งโดยทั่วไปมีค่าประมาณ 2.0V, และ I_f คือกระแสไฟฟ้าที่ต้องการให้ไหลผ่านหลอด LED ซึ่งเราอาจเลือกใช้ 10 mA เพื่อความปลอดภัย

จากการคำนวณ $R = (3.3 - 2.0) / 0.01 = 130\Omega$ อย่างไรก็ตาม ในทางปฏิบัติเรามักเลือกใช้ตัวต้านทานขนาด 220Ω ซึ่งเป็นค่ามาตรฐานที่หาง่าย และให้ความปลอดภัยสูงกว่า แม้ว่าหลอด LED จะสว่างน้อยลงเล็กน้อย แต่ก็ยังเพียงพอสำหรับการมองเห็นได้ชัดเจน

ในการต่อวงจร เราจะต้องต่อตัวต้านทาน 220Ω แยกให้กับแต่ละ segment ทั้ง 7 ตัว (a, b, c, d, e, f, g) และอาจรวมถึง decimal point ด้วย แล้วต่อปลายอีกด้านหนึ่งของตัวต้านทานเข้ากับขา GPIO ของ ESP32 ตัวอย่างเช่น segment a ต่อกับ GPIO 12, segment b ต่อกับ GPIO 13, segment c ต่อกับ GPIO 14 และไปเรื่อยๆ ส่วนขา common (COM) ของ 7-Segment Display แบบ Common Cathode จะต่อเข้ากับ GND ของ ESP32

การเลือกขา GPIO นั้นควรหลีกเลี่ยงขาที่มีหน้าที่พิเศษของ ESP32 เช่น GPIO 0, GPIO 2, GPIO 15 ที่ใช้ในการ boot หรือ GPIO 6-11 ที่ต่อกับ Flash Memory ภายใน ควรเลือกใช้ขาที่เป็น General Purpose และไม่มีการใช้งานอื่นๆ เพื่อหลีกเลี่ยงปัญหาในการทำงาน

การต่อวงจรบน Breadboard ควรใช้สายจัมเปอร์ที่มีสีต่างกัน เพื่อให้สามารถตรวจสอบและแก้ไขได้ง่าย อาจใช้สายสีแดงสำหรับสัญญาณ และสายสีดำสำหรับ GND นอกจากนี้ควรตรวจสอบขา Pin ของ 7-Segment Display จาก Datasheet ให้ถูกต้อง เพราะแต่ละรุ่นอาจมีการจัดเรียงขาที่แตกต่างกัน

การเขียนโปรแกรมควบคุม 7-Segment Display

การเขียนโปรแกรมควบคุม 7-Segment Display ด้วย ESP32 นั้นเริ่มต้นจากการกำหนดขา GPIO ที่ใช้เชื่อมต่อกับแต่ละ segment ในโปรแกรม เราจะสร้างตัวแปร array เพื่อเก็บหมายเลขขา GPIO เหล่านี้ เช่น `int segmentPins[] = {12, 13, 14, 27, 26, 25, 33, 32};` ซึ่งแทนค่า segment a ถึง DP ตามลำดับ จากนั้นในฟังก์ชัน `setup()` เราจะกำหนดให้ขาเหล่านี้ทำงานเป็น OUTPUT โดยใช้คำสั่ง `pinMode(segmentPins[i], OUTPUT);` ในลูป

ขั้นตอนสำคัญคือการสร้างตารางรหัสสำหรับแสดงตัวเลข 0-9 โดยเราจะใช้ข้อมูลแบบ binary หรือ hexadecimal เพื่อกำหนดว่าแต่ละตัวเลขจะต้องเปิด-ปิด segment ใดบ้าง ตัวอย่างเช่น ตัวเลข 0 ต้องเปิด segment a, b, c, d, e, f และปิด segment g ซึ่งเราอาจเขียนเป็น binary `0b01111111` หรือ hexadecimal `0x7F` สำหรับ Common Cathode

เราจะสร้างฟังก์ชันสำหรับแสดงผลตัวเลข เช่น `void displayDigit(int digit)` ที่รับค่าตัวเลข 0-9 เป็น parameter แล้วดึงข้อมูลรหัสจากตารางที่เตรียมไว้ จากนั้นวนลูปเขียนค่า HIGH หรือ LOW ไปยังแต่ละขา GPIO ตามรหัสที่กำหนด ตัวอย่างเช่น `digitalWrite(segmentPins[i], bitRead(segments[digit], i));` ซึ่งจะอ่านแต่ละ bit จากรหัสและส่งออกไปยัง segment ที่สอดคล้องกัน

ในฟังก์ชัน `loop()` เราสามารถสร้างการนับตัวเลขจาก 0-9 โดยใช้ตัวแปร counter และเพิ่มค่าทีละ 1 พร้อมเรียกฟังก์ชัน `displayDigit(counter)` เพื่อแสดงผล จากนั้นใช้ `delay(1000)` เพื่อหน่วงเวลา 1 วินาที และเมื่อ counter ถึง 9 ก็ให้รีเซ็ตกลับเป็น 0 ด้วยคำสั่ง `if (counter > 9) counter = 0;`

การเขียนโปรแกรมที่ดีควรแยกโค้ดเป็นฟังก์ชันย่อยๆ เพื่อให้อ่านและดูแลรักษาได้ง่าย เช่น แยกฟังก์ชันการแสดงผล ฟังก์ชันอ่านค่า input และฟังก์ชันควบคุม Logic หลัก นอกจากนี้ควรใส่ comment อธิบายโค้ดเพื่อให้ผู้อื่นหรือตัวเองในอนาคตสามารถเข้าใจได้ง่าย

5.2 โพรโทคอลการสื่อสาร I2C (Inter-Integrated Circuit)

หลักการทำงานของ I2C

I2C หรือ Inter-Integrated Circuit เป็นโพรโทคอลการสื่อสารแบบอนุกรม (Serial Communication) ที่พัฒนาโดยบริษัท Philips (ปัจจุบันคือ NXP Semiconductors) เพื่อใช้สำหรับการสื่อสารระหว่าง Integrated Circuit ต่างๆ บนแผงวงจรเดียวกัน โดยมีจุดเด่นคือใช้สายสัญญาณเพียง 2 เส้นเท่านั้น แต่สามารถเชื่อมต่ออุปกรณ์หลายตัวเข้าด้วยกันได้ ทำให้ประหยัดขา GPIO ของไมโครคอนโทรลเลอร์ และลดความซับซ้อนของการต่อวงจร

สายสัญญาณทั้ง 2 เส้นของ I2C ประกอบด้วย SCL (Serial Clock Line) ที่ใช้สำหรับสัญญาณนาฬิกาเพื่อซิงโครไนซ์การส่งข้อมูล และ SDA (Serial Data Line) ที่ใช้สำหรับส่งข้อมูลจริง ทั้งสองสายนี้จะต้องมี Pull-up Resistor ต่อขึ้นไปยังแหล่งจ่ายไฟบวก (โดยทั่วไปใช้ $4.7k\Omega$ หรือ $10k\Omega$) เพื่อให้สัญญาณอยู่ในสถานะ HIGH เมื่อไม่มีการส่งข้อมูล

ในระบบ I2C จะมีอุปกรณ์ 2 ประเภท คือ Master และ Slave โดย Master เป็นผู้ควบคุมการสื่อสาร สร้างสัญญาณนาฬิกา และเริ่มต้น-สิ้นสุดการสื่อสาร ส่วน Slave เป็นอุปกรณ์ที่ถูกควบคุม รอรับคำสั่งจาก Master และตอบสนองตามที่ได้รับคำสั่ง ในระบบหนึ่งสามารถมี Master ได้หลายตัว (Multi-master) และมี Slave ได้มากถึง 128 ตัว (ในทางทฤษฎี) ขึ้นอยู่กับ Address space ที่มี 7 bit

การสื่อสารของ I2C เริ่มต้นจาก Master ส่งสัญญาณ Start Condition โดยการดึง SDA จาก HIGH ลงเป็น LOW ขณะที่ SCL ยังคงเป็น HIGH จากนั้น Master จะส่ง Address ของ Slave ที่ต้องการสื่อสารด้วย 7 bit และตามด้วย bit ที่ 8 เพื่อระบุว่าจะเป็นการ Read หรือ Write (0 = Write, 1 = Read) Slave ที่มี Address ตรงกับที่ถูกเรียกจะตอบกลับด้วยสัญญาณ ACK (Acknowledge) โดยดึง SDA ลงเป็น LOW ในช่วง Clock pulse ถัดไป

หลังจากนั้นจะมีการส่งข้อมูลเป็น byte (8 bit) โดยส่งทีละ bit จาก Most Significant Bit (MSB) ไปยัง Least Significant Bit (LSB) หลังจากส่งครบ 1 byte ผู้รับจะส่ง ACK กลับเพื่อยืนยันว่าได้รับข้อมูลแล้ว กระบวนการนี้จะทำซ้ำไปเรื่อยๆ จนกว่าจะส่งข้อมูลครบตามต้องการ สุดท้าย Master จะส่งสัญญาณ Stop Condition โดยดึง SDA จาก LOW ขึ้นเป็น HIGH ขณะที่ SCL เป็น HIGH เพื่อสิ้นสุดการสื่อสาร

ความเร็วในการสื่อสารของ I2C มีหลายระดับ โดยที่นิยมใช้กันคือ Standard Mode ที่ความเร็ว 100 kHz, Fast Mode ที่ 400 kHz และ Fast Mode Plus ที่ 1 MHz สำหรับ ESP32 นั้นรองรับความเร็วได้หลายระดับ และสามารถกำหนดได้ในโปรแกรม โดยความเร็วที่เหมาะสมขึ้นอยู่กับระยะทางของสายสัญญาณ จำนวนอุปกรณ์ที่ต่อ และความต้องการของแอปพลิเคชัน

ข้อดีและข้อจำกัดของ I2C

ข้อดีหลักของ I2C คือความประหยัดในการใช้ขา GPIO เนื่องจากใช้สายเพียง 2 เส้น แต่สามารถเชื่อมต่ออุปกรณ์หลายตัวได้ ตัวอย่างเช่น หากเราต้องการต่อ LCD, RTC, EEPROM และเซนเซอร์อุณหภูมิเข้ากับ ESP32 หากใช้การต่อแบบธรรมดา อาจต้องใช้ GPIO มากกว่า 20 ขา แต่หากใช้ I2C ใช้เพียง 2 ขาเท่านั้น นี่เป็นข้อได้เปรียบอย่างมากสำหรับโปรเจกต์ที่มีการใช้อุปกรณ์หลายชนิด

ข้อดีอีกประการคือโครงสร้างของ I2C ที่ชัดเจนและเป็นมาตรฐาน ทำให้อุปกรณ์จากผู้ผลิตต่างๆ สามารถทำงานร่วมกันได้โดยไม่มีปัญหา นอกจากนี้ยังมี Library สำเร็จรูปมากมายที่รองรับการสื่อสาร I2C ทำให้การพัฒนาโปรแกรมง่ายและรวดเร็วขึ้น นักพัฒนาไม่ต้องกังวลเกี่ยวกับรายละเอียดปลีกย่อยของโปรโตคอล เพียงแค่เรียกใช้ฟังก์ชันที่มีอยู่แล้วในระบบ

อย่างไรก็ตาม I2C ก็มีข้อจำกัดบางประการที่ควรทราบ ข้อแรกคือความเร็วในการสื่อสารที่ต่ำกว่าโปรโตคอลอื่นๆ เช่น SPI ซึ่งอาจไม่เหมาะสำหรับแอปพลิเคชันที่ต้องการถ่ายโอนข้อมูลจำนวนมากในเวลาสั้นๆ เช่น การแสดงผลกราฟิกความละเอียดสูง ข้อจำกัดที่สองคือระยะทางของสายสัญญาณที่จำกัด โดยทั่วไปแนะนำให้ใช้สายไม่เกิน 1 เมตร เพราะหากยาวเกินไปจะเกิดปัญหาเรื่องความจุของสาย (Capacitance) ที่ทำให้สัญญาณบิดเบี้ยว

ปัญหาที่พบบ่อยในการใช้งาน I2C คือการชนกันของ Address เมื่อมีอุปกรณ์หลายตัวที่มี Address เดียวกัน ซึ่งอาจเกิดขึ้นได้หากเราต้องการใช้งานเซนเซอร์ชนิดเดียวกันหลายตัวในระบบเดียว วิธีแก้ไขอาจเป็นการเลือกใช้โมดูลที่มีการปรับ Address ได้ (บางโมดูลมี jumper หรือ solder pad สำหรับเปลี่ยน Address) หรือใช้ I2C Multiplexer เพื่อแยกบัสออกเป็นหลายช่องทาง

การ Debug ปัญหาการสื่อสารของ I2C สามารถทำได้โดยการใช้โปรแกรม I2C Scanner ซึ่งจะสแกนหา Address ของอุปกรณ์ทั้งหมดที่ต่ออยู่บน I2C bus และแสดงผลทาง Serial Monitor ถ้าโปรแกรมไม่พบ Address ของอุปกรณ์ แสดงว่าอาจมีปัญหาเรื่องการต่อวงจร Pull-up resistor หรืออุปกรณ์เสียหาย

5.3 7-Segment Display แบบ I2C 4 หลัก

โครงสร้างและหลักการทำงาน

7-Segment Display แบบ I2C 4 หลักเป็นโมดูลที่รวม 7-Segment Display จำนวน 4 หลักเข้าด้วยกัน พร้อมวงจรควบคุมแบบ I2C ทำให้สามารถแสดงผลตัวเลข 4 หลัก (0000-9999) ได้โดยใช้สายสัญญาณเพียง 2 เส้น โมดูลที่นิยมใช้กันมากคือแบบที่ใช้ชิป TM1637 ซึ่งเป็น LED Driver IC ที่ออกแบบมาสำหรับควบคุม 7-Segment Display โดยเฉพาะ

TM1637 มีความสามารถในการควบคุม LED ได้สูงสุด 6 หลัก แต่ในโมดูลทั่วไปจะใช้ 4 หลัก เนื่องจากเหมาะสำหรับการแสดงผลตัวเลขทั่วไป เช่น เวลา (12:34), อุณหภูมิ (25.6°C), หรือค่าตัวเลขต่างๆ ชิป TM1637 มีหน่วยความจำภายใน (Display RAM) สำหรับเก็บข้อมูลที่จะแสดงผล ทำให้ไมโครคอนโทรลเลอร์ไม่ต้องส่งข้อมูลซ้ำๆ อยู่ตลอดเวลา เพียงแค่ส่งข้อมูลครั้งเดียวเมื่อต้องการเปลี่ยนค่าที่แสดงผล

การทำงานภายในของ TM1637 ใช้เทคนิคที่เรียกว่า Multiplexing คือการเปิด-ปิดแต่ละหลักของ 7-Segment อย่างรวดเร็วสลับกันไป โดยในแต่ละช่วงเวลาจะมีเพียง 1 หลักเท่านั้นที่ถูกเปิด แต่เนื่องจาก

ความเร็วในการสลับสูงมาก (หลายร้อย Hz) ทำให้ดวงตามนุษย์มองเห็นเป็นการแสดงผลพร้อมกันทุกหลัก เทคนิคนี้ช่วยลดการใช้กระแสไฟฟ้า เพราะไม่ต้องเปิด LED ทุกหลักพร้อมกัน

โมดูล 7-Segment แบบ I2C มักมีขาเชื่อมต่อ 4 ขา คือ VCC (แหล่งจ่ายไฟ 3.3V หรือ 5V), GND (กราวด์), CLK (Clock สำหรับ I2C) และ DIO (Data Input/Output สำหรับ I2C) โมดูลบางรุ่นมี Pull-up resistor ติดมาให้แล้วภายในโมดูล ทำให้ไม่ต้องต่อตัวต้านทานเพิ่มเติม แต่ถ้าต่อระยะไกลหรือมีอุปกรณ์หลายตัว อาจจำเป็นต้องเพิ่ม Pull-up resistor ภายนอก

ข้อดีของการใช้โมดูลนี้คือความสะดวกในการใช้งาน ประหยัดขา GPIO และมี Library สำเร็จรูปให้ใช้งาน เช่น TM1637Display Library ซึ่งมีฟังก์ชันครอบคลุมสำหรับการแสดงผล การควบคุมความสว่าง และการจัดการ colon (เครื่องหมาย : สำหรับแสดงเวลา) นอกจากนี้โมดูลยังมีความเสถียรสูง เพราะวงจรถูกออกแบบมาเป็นพิเศษสำหรับการแสดงผล ไม่ต้องกังวลเรื่องการกระพริบหรือความสว่างไม่สม่ำเสมอ

การเชื่อมต่อโมดูล 7-Segment I2C เข้ากับ ESP32 เป็นเรื่องง่าย เพียงแค่ต่อขา VCC เข้ากับแหล่งจ่ายไฟ 3.3V หรือ 5V (ตรวจสอบจาก Datasheet ของโมดูล) ขา GND เข้ากับ GND ของ ESP32 ขา CLK เข้ากับขา GPIO ที่กำหนดให้เป็น SCL (เช่น GPIO 22) และขา DIO เข้ากับขา GPIO ที่กำหนดให้เป็น SDA (เช่น GPIO 21) ควรใช้สายที่มีคุณภาพดีและไม่ยาวเกินไป เพื่อลดปัญหาสัญญาณรบกวนก่อนเริ่มเขียนโปรแกรม ควรติดตั้ง TM1637Display Library ผ่าน Arduino IDE Library Manager โดยค้นหาคำว่า "TM1637" แล้วเลือกติดตั้ง Library ที่มีชื่อ "TM1637Display by Avishay Orpaz" หรืออาจใช้ Library อื่นๆ ที่รองรับชิป TM1637 ได้เช่นกัน การติดตั้ง Library ทำให้เราสามารถเรียกใช้ฟังก์ชันต่างๆ ได้โดยไม่ต้องเขียนโค้ดควบคุมโปรโตคอลการสื่อสารเองในโปรแกรม เราจะเริ่มต้นด้วยการ include Library และสร้าง object ของ TM1637Display โดยระบุหมายเลขขา CLK และ DIO ที่ใช้ เช่น `TM1637Display display(CLK_PIN, DIO_PIN);` จากนั้นในฟังก์ชัน `setup()` เราจะเรียกคำสั่ง `display.setBrightness(brightness);` เพื่อกำหนดความสว่างของหน้าจอ โดยค่า `brightness` มีช่วงตั้งแต่ 0 (มืดที่สุด) ถึง 7 (สว่างที่สุด) และเรียก `display.clear();` เพื่อล้างหน้าจอการแสดงผลตัวเลขสามารถทำได้ง่ายๆ ด้วยคำสั่ง `display.showNumberDec(number);` โดย `number` เป็นจำนวนเต็มที่ต้องการแสดง ตัวอย่างเช่น `display.showNumberDec(1234);` จะแสดงตัวเลข 1234 บนหน้าจอ หากตัวเลขมีหลักน้อยกว่า 4 หลัก ระบบจะเติม leading zero ให้โดยอัตโนมัติ หรือเราสามารถกำหนดให้ไม่แสดง leading zero ด้วยการส่ง parameter เพิ่มเติมสำหรับการแสดงเวลา เช่น 12:34 เราสามารถใช้คำสั่ง `display.showNumberDecEx(1234, 0b01000000, true);` โดย parameter ตัวที่สองคือ bitmask ที่กำหนดว่าจะเปิด colon (เครื่องหมาย :) หรือไม่ ค่า `0b01000000` หมายถึงเปิด colon ตรงกลาง และ parameter ตัวที่สามคือการกำหนดว่าจะแสดง leading zero หรือไม่ วิธีนี้เหมาะสำหรับการสร้างนาฬิกา ดิจิทัลหรือการจับเวลา

5.4 LCD 16x2 แบบ I2C

โครงสร้างและหลักการทำงาน

LCD (Liquid Crystal Display) 16x2 เป็นจอแสดงผลแบบ Character Display ที่สามารถแสดงผลได้ 16 ตัวอักษร (columns) ใน 2 แถว (rows) รวมทั้งหมด 32 ตัวอักษร LCD ประเภทนี้ใช้หลักการทำงานของ Liquid Crystal ที่สามารถควบคุมให้โปร่งแสงหรือทึบแสงได้โดยการใช้สัญญาณไฟฟ้า เมื่อรวมกับ backlight (แสงพื้นหลัง) ทำให้สามารถมองเห็นตัวอักษรได้ชัดเจน

LCD 16x2 มาตรฐานจะใช้ Controller Chip HD44780 หรือ compatible chips ซึ่งเป็น standard ที่แพร่หลายมาก ชิพนี้มีหน่วยความจำภายใน (DDRAM - Display Data RAM) สำหรับเก็บข้อมูลตัวอักษรที่จะแสดงผล และ CGRAM (Character Generator RAM) สำหรับเก็บ custom characters ที่ผู้ใช้สร้างขึ้นเอง LCD สามารถแสดงผลได้ทั้งตัวอักษร A-Z, a-z, ตัวเลข 0-9, สัญลักษณ์พิเศษต่างๆ และตัวอักษรภาษาญี่ปุ่น (Katakana)

การต่อใช้งาน LCD 16x2 โดยตรงต้องใช้ขา GPIO มากถึง 6-10 ขา (ขึ้นอยู่กับ mode ที่ใช้) ซึ่งค่อนข้างสิ้นเปลือง ดังนั้นจึงมีการพัฒนา I2C Adapter Module ที่แปลงสัญญาณ I2C เป็นสัญญาณควบคุม LCD ทำให้สามารถใช้สายเพียง 2 เส้นในการควบคุม LCD ได้ โมดูล I2C adapter ที่นิยมใช้กันคือแบบที่ใช้ชิพ PCF8574 ซึ่งเป็น I2C I/O Expander ที่มี GPIO 8 ขา

PCF8574 จะรับข้อมูลจาก I2C bus แล้วส่งออกเป็นสัญญาณ parallel ไปยังขาต่างๆ ของ LCD โมดูล I2C adapter จะมี potentiometer สำหรับปรับ contrast ของ LCD และ jumper สำหรับเปิด-ปิด backlight ในบางรุ่น นอกจากนี้โมดูลยังมี address jumpers ที่สามารถปรับเปลี่ยน I2C address ได้ โดยทั่วไป default address คือ 0x27 หรือ 0x3F ขึ้นอยู่กับผู้ผลิต

การเชื่อมต่อและการเขียนโปรแกรม

การเชื่อมต่อ LCD 16x2 แบบ I2C เข้ากับ ESP32 ทำได้โดยต่อขา VCC เข้ากับ 5V (LCD ส่วนใหญ่ใช้ 5V แม้ว่า ESP32 จะเป็น 3.3V ก็ตาม เพราะโมดูล I2C มี level shifter ในตัว), GND เข้ากับ GND, SDA เข้ากับ GPIO 21 และ SCL เข้ากับ GPIO 22 ของ ESP32 ก่อนใช้งานควรปรับ potentiometer เพื่อให้ตัวอักษรมองเห็นได้ชัดเจน โดยหมุนไปมาจนกว่าจะเห็นตัวอักษรชัดที่สุด

ขั้นตอนแรกในการเขียนโปรแกรมคือการติดตั้ง LiquidCrystal_I2C Library โดยเปิด Arduino IDE ไปที่ Sketch > Include Library > Manage Libraries แล้วค้นหา "LiquidCrystal I2C" และติดตั้ง Library ที่ชื่อ "LiquidCrystal I2C by Frank de Brabander" หรือ Library อื่นที่รองรับ PCF8574 หากยังไม่แน่ใจว่า LCD ของเรามี address อะไร ให้ใช้โปรแกรม I2C Scanner เพื่อตรวจสอบก่อน

ในโปรแกรม เราจะ include Library และสร้าง object ของ LCD โดยระบุ address, จำนวน columns และ rows เช่น LiquidCrystal_I2C lcd(0x27, 16, 2); หมายถึง LCD ที่มี address 0x27 แสดงผลได้ 16 ตัวอักษร 2 แถว ในฟังก์ชัน setup() เราจะเรียก lcd.init(); เพื่อเริ่มต้นการทำงานของ LCD และ lcd.backlight(); เพื่อเปิด backlight ทำให้มองเห็นได้ชัดเจนขึ้น

การแสดงผลข้อความทำได้โดยใช้คำสั่ง lcd.print("Hello World"); ซึ่งจะแสดงข้อความที่ตำแหน่ง cursor ปัจจุบัน หากต้องการกำหนดตำแหน่งที่จะแสดงผล ใช้คำสั่ง lcd.setCursor(col, row); โดย col คือ

หมายเลขคอลัมน์ (0-15) และ row คือหมายเลขแถว (0-1) ตัวอย่างเช่น `lcd.setCursor(0, 0);` จะย้าย cursor ไปที่มุมซ้ายบนของหน้าจอ

การล้างหน้าจอทำได้ด้วยคำสั่ง `lcd.clear();` ซึ่งจะลบข้อความทั้งหมดและย้าย cursor กลับไปที่ตำแหน่ง (0,0) หากต้องการแค่ย้าย cursor กลับไปตำแหน่งเริ่มต้นโดยไม่ลบข้อความ ใช้คำสั่ง `lcd.home();` สำหรับการแสดงผลตัวเลข เราสามารถใช้ `lcd.print(number);` โดยตรง Library จะแปลงตัวเลขเป็นข้อความให้โดยอัตโนมัติ

ฟังก์ชันที่มีประโยชน์อื่นๆ ได้แก่ `lcd.cursor();` สำหรับแสดง cursor แบบขีดเส้นใต้, `lcd.noCursor();` สำหรับซ่อน cursor, `lcd.blink();` สำหรับให้ cursor กระพริบ, `lcd.noBlink();` สำหรับหยุดการกระพริบ, `lcd.display();` สำหรับเปิดการแสดงผล และ `lcd.noDisplay();` สำหรับปิดการแสดงผล (แต่ข้อมูลยังคงอยู่ใน memory)

การสร้าง Custom Characters

ความสามารถพิเศษของ LCD 16x2 คือการสร้าง custom characters หรือตัวอักษรพิเศษที่เราออกแบบเอง LCD สามารถเก็บ custom characters ได้สูงสุด 8 ตัว (ตำแหน่ง 0-7) แต่ละตัวอักษรมีขนาด 5x8 pixels เราสามารถออกแบบสัญลักษณ์ต่างๆ เช่น หัวใจ, ลูกศร, กราฟ, ไอคอนพิเศษ หรือตัวอักษรภาษาไทย

การสร้าง custom character เริ่มต้นจากการออกแบบ pixel pattern ซึ่งสามารถใช้กระดาษตาราง หรือ online tools เช่น LCD Custom Character Generator จากนั้นแปลง pattern เป็นข้อมูล byte array

ในฟังก์ชัน `setup()` เราจะเรียกคำสั่ง `lcd.createChar(0, heart);` เพื่อบันทึก custom character ลงใน CGRAM ที่ตำแหน่ง 0 จากนั้นเมื่อต้องการแสดงผล ใช้คำสั่ง `lcd.write(0);` เพื่อแสดง custom character ที่ตำแหน่ง 0 เราสามารถสร้าง custom characters หลายตัวและเรียกใช้สลับกันได้ตามต้องการ

การใช้ custom characters ทำให้สามารถสร้าง user interface ที่น่าสนใจขึ้น เช่น แถบ progress bar โดยใช้ custom characters ที่มีความเต็มต่างกันไปในแต่ละตัว, กราฟแท่ง, สัญลักษณ์อุณหภูมิ, หรือ ไอคอนแบตเตอรี่ ความคิดสร้างสรรค์ในการออกแบบ custom characters สามารถยกระดับความสวยงามและประโยชน์ใช้สอยของ LCD ได้อย่างมาก

การรับสัญญาณจากสวิตช์ (Push Button)

หลักการทำงานของสวิตช์กด

Push Button Switch หรือสวิตช์กดเป็นอุปกรณ์ input ที่เรียบง่ายแต่สำคัญมาก ใช้สำหรับรับคำสั่งจากผู้ใช้ สวิตช์กดทำงานโดยการเปิด-ปิดวงจรไฟฟ้า เมื่อกดจะทำให้วงจรเชื่อมต่อกัน (Closed) และเมื่อปล่อยจะทำให้วงจรขาดจากกัน (Open) สวิตช์มีหลายประเภท เช่น Momentary Switch ที่เมื่อปล่อยจะกลับสู่สถานะเดิม และ Latching Switch ที่จะค้างอยู่ในสถานะที่กดไว้

สำหรับการใช้งานกับไมโครคอนโทรลเลอร์ เรามักใช้ Momentary Normally Open (NO) Switch ซึ่งในสถานะปกติวงจรจะเปิดอยู่ (ไม่มีการเชื่อมต่อ) และเมื่อกดจึงจะมีการเชื่อมต่อ สวิตช์ประเภทนี้มักมี 4 ขา แต่ที่ใช้งานจริงมีเพียง 2 ขา โดยอีก 2 ขาจะต่อเชื่อมกันอยู่แล้วภายใน สำหรับความแข็งแรงของโครงสร้าง

การต่อสวิตช์เข้ากับไมโครคอนโทรลเลอร์ต้องใช้ตัวต้านทาน Pull-up หรือ Pull-down เพื่อกำหนดสถานะของสัญญาณเมื่อสวิตช์ไม่ได้ถูกกด หากไม่มีตัวต้านทาน สัญญาณจะอยู่ในสถานะ "Floating" ที่ไม่แน่นอน อาจอ่านค่าได้เป็น HIGH หรือ LOW สลับกันไปมาตามสัญญาณรบกวน

วงจร Pull-down: ต่อตัวต้านทาน ($10k\Omega$) จากขา GPIO ลงสู่ GND และต่อสวิตช์จากขา GPIO ขึ้นไปยัง VCC เมื่อสวิตช์ไม่ถูกกด สัญญาณจะถูก pull ลงมาเป็น LOW (0V) ผ่านตัวต้านทาน เมื่อกดสวิตช์ สัญญาณจะเป็น HIGH (3.3V) โดยตรงจาก VCC

วงจร Pull-up: ต่อตัวต้านทาน ($10k\Omega$) จากขา GPIO ขึ้นไปยัง VCC และต่อสวิตช์จากขา GPIO ลงสู่ GND เมื่อสวิตช์ไม่ถูกกด สัญญาณจะถูก pull ขึ้นเป็น HIGH (3.3V) ผ่านตัวต้านทาน เมื่อกดสวิตช์ สัญญาณจะเป็น LOW (0V) โดยตรงจาก GND

ESP32 มี Internal Pull-up และ Pull-down Resistors อยู่แล้วภายในชิป สามารถเปิดใช้งานได้ผ่านโปรแกรมด้วยคำสั่ง `pinMode(pin, INPUT_PULLUP);` หรือ `pinMode(pin, INPUT_PULLDOWN);` ทำให้ไม่จำเป็นต้องต่อตัวต้านทานภายนอกในหลายกรณี อย่างไรก็ตาม การใช้ Internal Pull-up เป็นที่นิยมมากกว่าเพราะเสถียรกว่า

ปัญหา Switch Bouncing และวิธีแก้ไข

Switch Bouncing เป็นปัญหาที่เกิดขึ้นกับสวิตช์เชิงกลทุกชนิด เมื่อเรากดหรือปล่อยสวิตช์ ส่วนสัมผัสโลหะภายในจะสั่นไหวเล็กน้อย ทำให้เกิดการเปิด-ปิดวงจรหลายครั้งอย่างรวดเร็วในช่วงเวลาสั้นๆ (ประมาณ 10-100 มิลลิวินาที) สำหรับมนุษย์ถือว่าเป็นการกดครั้งเดียว แต่สำหรับไมโครคอนโทรลเลอร์ที่ทำงานเร็วมาก อาจตรวจจับได้เป็นการกดหลายครั้ง

ตัวอย่างปัญหาที่เกิดจาก Switch Bouncing เช่น เราต้องการให้กดสวิตช์แล้ว counter เพิ่มขึ้น 1 ครั้ง แต่เนื่องจาก bouncing ทำให้ counter เพิ่มขึ้น 3-5 ครั้งในการกดเพียงครั้งเดียว หรือในโปรแกรมที่มีการตรวจจับ edge (การเปลี่ยนจาก LOW เป็น HIGH หรือกลับกัน) อาจตรวจจับ edge ได้หลายครั้งในการกดเพียงครั้งเดียว

วิธีแก้ไขปัญหา Bouncing มี 2 แนวทางหลัก คือ Hardware Debouncing และ Software Debouncing

Hardware Debouncing: ใช้ตัวเก็บประจุ (Capacitor) ต่อขนานกับสวิตช์เพื่อทำให้สัญญาณนุ่มนวลขึ้น หรือใช้ IC เฉพาะทาง เช่น 74HC14 Schmitt Trigger แต่วิธีนี้เพิ่มต้นทุนและความซับซ้อนของวงจร จึงไม่ค่อยนิยมในโปรเจกต์ทั่วไป

Software Debouncing: เป็นวิธีที่นิยมมากกว่า มีหลายเทคนิค เช่น:

1. **Simple Delay:** หลังจากตรวจจับการเปลี่ยนแปลงสถานะของสวิตช์ ให้หน่วงเวลา 50-100

มิลลิวินาที จากนั้นอ่านค่าสวิตช์อีกครั้งเพื่อยืนยัน ถ้าสถานะยังคงเหมือนเดิม แสดงว่าเป็นการกดจริง

```

if (digitalRead(buttonPin) == LOW) {
    delay(50); // debounce delay
    if (digitalRead(buttonPin) == LOW) {
        // confirmed button press
    }
}

```

2. **State Change Detection:** จำสถานะก่อนหน้าไว้ และตรวจจับเฉพาะเมื่อมีการเปลี่ยนสถานะจาก HIGH เป็น LOW หรือกลับกัน พร้อมกับใช้ delay สั้นๆ

```

int buttonState;
int lastButtonState = HIGH;

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

    if (buttonState != lastButtonState) {
        delay(50); // debounce
        buttonState = digitalRead(buttonPin);

        if (buttonState == LOW) {
            // button pressed
        }
    }

    lastButtonState = buttonState;
}

```

3. **Millis() Method:** ใช้ฟังก์ชัน millis() ตรวจสอบเวลาที่ผ่านไปนับจากการกดครั้งล่าสุด เพื่อเพิกเฉยการเปลี่ยนแปลงที่เกิดขึ้นเร็วเกินไป วิธีนี้ไม่ทำให้โปรแกรมหยุดทำงานเหมือน delay()

```
unsigned long lastDebounceTime = 0;  
unsigned long debounceDelay = 50;
```

```
void loop() {  
  int reading = digitalRead(buttonPin);  
  
  if (reading != lastButtonState) {  
    lastDebounceTime = millis();  
  }  
  
  if ((millis() - lastDebounceTime) > debounceDelay) {  
    if (reading != buttonState) {  
      buttonState = reading;  
      if (buttonState == LOW) {  
        // button pressed  
      }  
    }  
  }  
  
  lastButtonState = reading;  
}
```

สร้างตารางการแสดงผลชุดตัวเลขบน 7-Segment

```
Week3.ino
1
2 // a, b, c, d, e, f, g
3 int segPins[7] = {4, 16, 17, 5, 18, 19, 21};
4 // กำหนดตำแหน่งขาของ Segment a-g ที่ต่อกับ ESP32
5 // a → GPIO 4
6 // b → GPIO 16
7 // c → GPIO 17
8 // d → GPIO 5
9 // e → GPIO 18
10 // f → GPIO 19
11 // g → GPIO 21
12 byte numberPatterns[10][7] = {
13 // a, b, c, d, e, f, g
14 {1, 1, 1, 1, 1, 1, 0}, // 0 เปิด a b c d e f
15 {0, 1, 1, 0, 0, 0, 0}, // 1 เปิด b c
16 {1, 1, 0, 1, 1, 0, 1}, // 2 เปิด a b d e g
17 {1, 1, 1, 1, 0, 0, 1}, // 3 เปิด a b c d g
18 {0, 1, 1, 0, 0, 1, 1}, // 4 เปิด b c f g
19 {1, 0, 1, 1, 0, 1, 1}, // 5 เปิด a c d f g
20 {1, 0, 1, 1, 1, 1, 1}, // 6 เปิด a c d e f g
21 {1, 1, 1, 0, 0, 0, 0}, // 7 เปิด a b c
22 {1, 1, 1, 1, 1, 1, 1}, // 8 เปิดทุก segment
23 {1, 1, 1, 1, 0, 1, 1} // 9 เปิด a b c d f g
24 // ตารางรูปแบบการแสดงผลตัวเลข 0-9 แต่ละบรรทัดแทนเลข 1 ตัว
25 };
26
```

สร้างฟังก์ชันแสดงตัวเลขตามที่ส่งเข้ามา

```
// ฟังก์ชันแสดงตัวเลขตามที่ส่งเข้ามา
// num = 0-9
void showNumber(int num) {
    for (int i = 0; i < 7; i++) {
        // เขียนค่า HIGH/LOW ไปที่แต่ละ segment
        digitalWrite(segPins[i], numberPatterns[num][i]);
    }
}
```

กำหนดรูปแบบโหมดการทำงาน

```
void setup() {
    Serial.begin(115200);
    // ตั้งค่า GPIO ทุกตัวเป็น OUTPUT
    for (int i = 0; i < 7; i++) {
        pinMode(segPins[i], OUTPUT);
    }
}
```

สร้างฟังก์ชันการแสดงผลวนซ้ำ

```
void loop() {
    // วนแสดงเลข 0-9 ทีละ 1 วินาที
    for (int n = 0; n <= 9; n++) {
        showNumber(n); // เรียกฟังก์ชันแสดงตัวเลข n
        delay(1000);    // หน่วงเวลา 1 วินาทีก่อนเปลี่ยนเป็นเลขถัดไป
    }
}
```


สร้างปุ่มกดเพิ่มรูปแบบการทำงาน

```
5 // ----- ปุ่มกด -----  
6 #define SW1 36  
7 #define SW2 39  
8 #define SW3 34  
9
```

สร้างตัวแปรควบคุมทำงาน

```
// -----  
int currentNum = 0; // เลขที่แสดงอยู่  
bool countingUp = false; // โหมดนับขึ้น  
bool countingDown = false; // โหมดนับลง  
bool paused = false; // หยุด/เล่น  
unsigned long lastUpdate = 0; // เวลาอัปเดตล่าสุด
```

เพิ่มกำหนดรูปแบบโหมดการทำงานของสวิทช์

```
void setup() {  
  Serial.begin(115200);  
  // ตั้งค่า LED 7 หลอดเป็น OUTPUT  
  for (int i = 0; i < NUM_LEDS; i++) {  
    pinMode(leds[i], OUTPUT);  
  }  
  // ตั้งค่า 7-Segment เป็น OUTPUT  
  for (int i = 0; i < 7; i++) {  
    pinMode(segPins[i], OUTPUT);  
  }  
  
  // ตั้งค่าปุ่มกดเป็น INPUT  
  pinMode(SW1, INPUT);  
  pinMode(SW2, INPUT);  
  pinMode(SW3, INPUT);  
  
  // เริ่มต้นแสดงเลข 0  
  showNumber(currentNum);  
  showNumberLEDs(currentNum);  
}
```

สร้างเงื่อนไขเมื่อมีการกดสวิทช์

```
void loop() {  
  // อ่านค่าปุ่ม (1 = กด, 0 = ไม่กด)  
  if (digitalRead(SW1) == 1) {  
    countingUp = true;  
    countingDown = false;  
    paused = false;  
    delay(200); // กันตั่ง  
  }  
  if (digitalRead(SW2) == 1) {  
    countingDown = true;  
    countingUp = false;  
    paused = false;  
    delay(200); // กันตั่ง  
  }  
  if (digitalRead(SW3) == 1) {  
    paused = !paused; // toggle หยุด/เล่น  
    delay(200); // กันตั่ง  
  }  
}
```

สร้างรูปแบบการเล่นแบบกระพริบเมื่อมีการหยุดตัวเลข

```
// ถ้าหยุด → กระพริบเลข
if (paused) {
    static bool blinkState = false;
    if (millis() - lastUpdate > 500) { // กระพริบทุก 0.5 วิ
        blinkState = !blinkState;
        if (blinkState) {
            showNumber(currentNum);
            showNumberLEDs(currentNum);
        } else {
            // ปิดหมด
            for (int i = 0; i < 7; i++) {
                digitalWrite(segPins[i], LOW);
                digitalWrite(leds[i], LOW);
            }
        }
        lastUpdate = millis();
    }
    return; // ออกไปไม่ต้องนับต่อ
}
```

ใช้งานฟังก์ชัน millis() แทน delay

```
// ถ้าไม่วิ่ง → อัปเดตเลขทุก 1 วิ
if (millis() - lastUpdate > 1000) {
    if (countingUp) {
        currentNum++;
        if (currentNum > 9) currentNum = 0;
    }
    if (countingDown) {
        currentNum--;
        if (currentNum < 0) currentNum = 9;
    }

    // แสดงเลข
    showNumber(currentNum);
    showNumberLEDs(currentNum);

    lastUpdate = millis();
}
}
```

7-Segment I2C

เรียกใช้งานไลบรารีสำหรับสื่อสาร I2C

```
#include <Wire.h> // ไลบรารีสำหรับสื่อสาร I2C
#include <Adafruit_GFX.h> // ไลบรารีกราฟิกพื้นฐาน
#include <Adafruit_LEDBackpack.h> // ไลบรารีสำหรับโมดูล 7-seg HT16K33

// ----- 7-Segment แบบขารวมดา (direct pin) -----
// กำหนดพิน segment a,b,c,d,e,f,g
int segPins[7] = {4, 16, 17, 5, 18, 19, 21};

// ----- ตารางรูปแบบตัวเลข 0-9 -----
byte numberPatterns[10][7] = {
    {1, 1, 1, 1, 1, 1, 0}, // 0
    {0, 1, 1, 0, 0, 0, 0}, // 1
    {1, 1, 0, 1, 1, 0, 1}, // 2
    {1, 1, 1, 1, 0, 0, 1}, // 3
    {0, 1, 1, 0, 0, 1, 1}, // 4
    {1, 0, 1, 1, 0, 1, 1}, // 5
    {1, 0, 1, 1, 1, 1, 1}, // 6
    {1, 1, 1, 0, 0, 0, 0}, // 7
    {1, 1, 1, 1, 1, 1, 1}, // 8
    {1, 1, 1, 1, 0, 1, 1} // 9
};
```

เรียกใช้งาน 7-Segment I2C HT16K33

```
// ----- 7-Segment I2C HT16K33 -----  
Adafruit_7segment matrix = Adafruit_7segment();  
// ----- ตัวแปรควบคุม -----
```

กำหนดตัวแปรควบคุม ตัวแปรควบคุม

```
// ----- ตัวแปรควบคุม -----  
int currentNum = 0; // เลขที่รับจาก Serial  
String inputString; // buffer สำหรับข้อความที่พิมพ์เข้ามา
```

สร้างฟังก์ชันแสดงเลข

```
// ----- ฟังก์ชันแสดงเลข -----  
void showNumberDirect(int num) {  
    if (num < 0 || num > 9) {  
        // ถ้าไม่ใช่ 0-9 ให้ปิด segment  
        for (int i = 0; i < 7; i++) digitalWrite(segPins[i], LOW);  
        return;  
    }  
    for (int i = 0; i < 7; i++) {  
        digitalWrite(segPins[i], numberPatterns[num][i]);  
    }  
}
```

สร้างฟังก์ชัน showNumberI2C

```
void showNumberI2C(int num) {  
    matrix.clear();  
    matrix.print(num); // แสดงตัวเลข (0-9999)  
    matrix.writeDisplay();  
}
```

สร้างฟังก์ชันกำหนดรูปแบบการทำงานของพินเชื่อมต่อ

```
// ----- setup -----  
void setup() {  
    Serial.begin(115200);  
    while (!Serial); // รอจนกว่า Serial พร้อม  
    Serial.println("กรุณาพิมพ์ตัวเลข 0-9999 แล้วกด Enter");  
  
    // ตั้งค่า 7-Segment ขาธรรมดาเป็น OUTPUT  
    for (int i = 0; i < 7; i++) {  
        pinMode(segPins[i], OUTPUT);  
    }  
  
    // เริ่มต้นใช้งาน I2C สำหรับ HT16K33  
    Wire.begin(22, 23); // กำหนด SDA=22, SCL=23  
    matrix.begin(0x70); // ที่อยู่ I2C ปกติคือ 0x70  
  
    // เริ่มต้นแสดงเลข 0  
    showNumberI2C(0);  
    showNumberDirect(0);  
}
```

สร้างฟังก์ชันวนซ้ำในการทำงาน

```
58 // ----- loop -----
59 void loop() {
60 // ถ้ามีข้อมูลเข้ามาจาก Serial
61 if (Serial.available() > 0) {
62   inputString = Serial.readStringUntil('\n'); // อ่านจนกด Enter
63   inputString.trim(); // ลบช่องว่าง
64
65   if (inputString.length() > 0) {
66     int num = inputString.toInt(); // แปลงเป็น int
67
68     if (num >= 0 && num <= 9999) {
69       currentNum = num;
70
71       // แสดงบน 7-seg I2C
72       showNumberI2C(currentNum);
73
74       // แยกหลักร้อย (หลักที่ 3)
75       int digit3 = (currentNum / 100) % 10;
76
77       // แสดงบน 7-seg ขาธรรมดา
78       showNumberDirect(digit3);
79
80       // Debug บน Serial Monitor
81       Serial.print("รับค่า: ");
82       Serial.print(currentNum);
83       Serial.print(" -> หลักร้อย = ");
84       Serial.println(digit3);
85       Serial.println("กรุณาพิมพ์ตัวเลข 0-9999 แล้วกด Enter");
86     } else {
87       Serial.println("กรุณาพิมพ์ตัวเลข 0-9999 เท่านั้น");
88     }
89   }
90 }
91 }
```

LCD 16*2 I2C

เรียกใช้งานไลบรารีสำหรับสื่อสาร I2C 16*2 I2C

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// --- ใช้ Hardware I2C บน ESP32 ---
// SDA = GPIO 22, SCL = GPIO 23
LiquidCrystal_I2C lcd(0x27, 16, 2); // ที่อยู่ 0x27 หรือ 0x3F แล้วแต่โมดูล
```

สร้างฟังก์ชันกำหนดรูปแบบการทำงานของพินเชื่อมต่อ

```
8 void setup() {
9   Serial.begin(115200);
10
11   // เริ่มต้นใช้งาน I2C
12   Wire.begin(22, 23);
13   lcd.begin(16, 2);
14   lcd.backlight();
15
16   // แสดงข้อความต้อนรับ
17   lcd.setCursor(0, 0);
18   lcd.print("Hello ESP32!");
19   lcd.setCursor(0, 1);
20   lcd.print("LCD I2C Ready");
21
22   Serial.println("LCD I2C พร้อมทำงาน (SDA=22, SCL=23)");
23 }
```

สร้างฟังก์ชันวนซ้ำในการทำงาน

```
4  
5 void loop() {  
6   // แสดงค่าเวลา millis() ที่ LCD  
7   lcd.clear();  
8   lcd.setCursor(0, 0);  
9   lcd.print("Running Time:");  
0   lcd.setCursor(0, 1);  
1   lcd.print(millis() / 1000);  
2   lcd.print(" sec");  
3  
4   delay(2000);  
5 }
```

6. แบบฝึกหัด/แบบทดสอบ


ใบงาน ที่ 2 ใบงานที่ 3 และ ใบงานที่ 4

7. เอกสารอ้างอิง (ขึ้นหน้าใหม่)

-

8. ภาคผนวก (เฉลยแบบฝึกหัด เฉลยแบบทดสอบ ฯ)

-

	ใบงาน ที่ 2	หน่วยที่2
	รหัสวิชา 21901-2007 วิชา เทคโนโลยีระบบสมองกลฝังตัวและไอโอที	สอนครั้งที่ 3 - 7
	ชื่อหน่วยการเรียนรู้ อุปกรณ์แสดงผลผลลัพธ์ (Display Device)	ทฤษฎี 4 ชม. ปฏิบัติ 16 ชม.
ชื่อเรื่อง/งาน แสดงผลข้อมูลจาก ESP32 ด้วยอุปกรณ์แสดงผล		

1. ผลลัพธ์การเรียนรู้ระดับหน่วยการเรียนรู้

- 1.1 สามารถอธิบายหลักการทำงานและประเภทของอุปกรณ์แสดงผลได้ถูกต้อง
- 1.2 สามารถเชื่อมต่อและเขียนโปรแกรมควบคุม 7-Segment Display ทั้งแบบธรรมดาและแบบ I2C ได้
- 1.3 สามารถเชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 1.4 สามารถประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้อย่างเหมาะสม
- 1.5 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ

2. อ้างอิงมาตรฐาน/เชื่อมโยงกลุ่มอาชีพ

-

3. สมรรถนะประจำหน่วย

- 3.1 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ
- 3.2 ทักษะการเขียนโปรแกรมควบคุมอุปกรณ์แสดงผล
- 3.3 ทักษะการใช้โปรโตคอลการสื่อสาร I2C
- 3.4 ทักษะการแก้ไขปัญหาและ Debugging

4. จุดประสงค์เชิงพฤติกรรม

- 4.1 อธิบายหลักการทำงานของ 7-Segment Display แบบธรรมดาและแบบ I2C ได้
- 4.2 อธิบายโครงสร้างและการทำงานของ LCD 16x2 แบบ I2C ได้
- 4.3 เชื่อมต่อวงจร 7-Segment Display แบบธรรมดากับ ESP32 ได้ถูกต้อง
- 4.4 เขียนโปรแกรมแสดงผลตัวเลขบน 7-Segment ได้
- 4.5 เชื่อมต่อและใช้งาน 7-Segment Display แบบ I2C 4 หลักได้
- 4.6 เชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 4.7 ประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้

5. เครื่องมือ วัสดุ และอุปกรณ์

- 5.1 Arduino IDE 2..3.6
- 5.2 ชุดฝึกปฏิบัติ

6. คำแนะนำ/ข้อควรระวัง

-

7. ขั้นตอนการปฏิบัติงาน

7.1 ต่อวงจร 7-Segment 1 หลักเข้ากับ ESP32

7.3 เพิ่มสวิตช์ ในการสร้างเงื่อนไขการทำงาน

7.4 เขียนโปรแกรมให้ 7-Segment 1 หลักทำงานตามลำดับที่กำหนด

7.5 ทดสอบและปรับแก้โปรแกรม

8. สรุปและวิจารณ์ผล

-

9. การประเมินผล

1. แบบประเมินสมรรถนะงานภาคปฏิบัติ (6 คะแนน)

ความถูกต้องของการต่อวงจร

ความถูกต้องของโปรแกรม

ความสมบูรณ์ของผลลัพธ์


2. แบบสังเกตพฤติกรรมลักษณะนิสัยการทำงาน (4 คะแนน)

ตรงต่อเวลา

ความร่วมมือและวินัย

10. เอกสารอ้างอิง /เอกสารค้นคว้าเพิ่มเติม

-

	ใบงาน ที่ 3	หน่วยที่2
	รหัสวิชา 21901-2007 วิชา เทคโนโลยีระบบสมองกลฝังตัวและไอโอที	สอนครั้งที่ 3 - 7
	ชื่อหน่วยการเรียนรู้ อุปกรณ์แสดงผลผลลัพธ์ (Display Device)	ทฤษฎี 4 ชม. ปฏิบัติ 16 ชม.
ชื่อเรื่อง/งาน แสดงผลข้อมูลจาก ESP32 ด้วยอุปกรณ์แสดงผล		

1. ผลลัพธ์การเรียนรู้ระดับหน่วยการเรียนรู้

- 1.1 สามารถอธิบายหลักการทำงานและประเภทของอุปกรณ์แสดงผลได้ถูกต้อง
- 1.2 สามารถเชื่อมต่อและเขียนโปรแกรมควบคุม 7-Segment Display ทั้งแบบธรรมดาและแบบ I2C ได้
- 1.3 สามารถเชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 1.4 สามารถประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้อย่างเหมาะสม
- 1.5 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ

2. อ้างอิงมาตรฐาน/เชื่อมโยงกลุ่มอาชีพ

-

3. สมรรถนะประจำหน่วย

- 3.1 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ
- 3.2 ทักษะการเขียนโปรแกรมควบคุมอุปกรณ์แสดงผล
- 3.3 ทักษะการใช้โปรโตคอลการสื่อสาร I2C
- 3.4 ทักษะการแก้ไขปัญหาและ Debugging

4. จุดประสงค์เชิงพฤติกรรม

- 4.1 อธิบายหลักการทำงานของ 7-Segment Display แบบธรรมดาและแบบ I2C ได้
- 4.2 อธิบายโครงสร้างและการทำงานของ LCD 16x2 แบบ I2C ได้
- 4.3 เชื่อมต่อวงจร 7-Segment Display แบบธรรมดากับ ESP32 ได้ถูกต้อง
- 4.4 เขียนโปรแกรมแสดงผลตัวเลขบน 7-Segment ได้
- 4.5 เชื่อมต่อและใช้งาน 7-Segment Display แบบ I2C 4 หลักได้
- 4.6 เชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 4.7 ประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้

5. เครื่องมือ วัสดุ และอุปกรณ์

- 5.1 Arduino IDE 2..3.6
- 5.2 ชุดฝึกปฏิบัติ

6. คำแนะนำ/ข้อควรระวัง

-

7. ขั้นตอนการปฏิบัติงาน

7.1 ต่อวงจร 7-Segment 4 หลัก รูปแบบ I2C เข้ากับ ESP32

7.3 เพิ่มสวิตช์ ในการสร้างเงื่อนไขการทำงาน

7.4 เขียนโปรแกรมให้ 7-Segment 4 หลัก รูปแบบ I2C ทำงานตามลำดับที่กำหนด

7.5 ทดสอบและปรับแก้โปรแกรม

8. สรุปและวิจารณ์ผล

-

9. การประเมินผล

1. แบบประเมินสมรรถนะงานภาคปฏิบัติ (6 คะแนน)

ความถูกต้องของการต่อวงจร

ความถูกต้องของโปรแกรม

ความสมบูรณ์ของผลลัพธ์


2. แบบสังเกตพฤติกรรมลักษณะนิสัยการทำงาน (4 คะแนน)

ตรงต่อเวลา

ความร่วมมือและวินัย

10. เอกสารอ้างอิง /เอกสารค้นคว้าเพิ่มเติม

-

	ใบงาน ที่ 4	หน่วยที่2
	รหัสวิชา 21901-2007 วิชา เทคโนโลยีระบบสมองกลฝังตัวและไอโอที	สอนครั้งที่ 3 - 7
	ชื่อหน่วยการเรียนรู้ อุปกรณ์แสดงผลผลลัพธ์ (Display Device)	ทฤษฎี 4 ชม. ปฏิบัติ 16 ชม.
ชื่อเรื่อง/งาน แสดงผลข้อมูลจาก ESP32 ด้วยอุปกรณ์แสดงผล		

1. ผลลัพธ์การเรียนรู้ระดับหน่วยการเรียนรู้

- 1.1 สามารถอธิบายหลักการทำงานและประเภทของอุปกรณ์แสดงผลได้ถูกต้อง
- 1.2 สามารถเชื่อมต่อและเขียนโปรแกรมควบคุม 7-Segment Display ทั้งแบบธรรมดาและแบบ I2C ได้
- 1.3 สามารถเชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 1.4 สามารถประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้อย่างเหมาะสม
- 1.5 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ

2. อ้างอิงมาตรฐาน/เชื่อมโยงกลุ่มอาชีพ

-

3. สมรรถนะประจำหน่วย

- 3.1 มีทักษะการแก้ไขปัญหาและพัฒนาโปรแกรมอย่างเป็นระบบ
- 3.2 ทักษะการเขียนโปรแกรมควบคุมอุปกรณ์แสดงผล
- 3.3 ทักษะการใช้โปรโตคอลการสื่อสาร I2C
- 3.4 ทักษะการแก้ไขปัญหาและ Debugging

4. จุดประสงค์เชิงพฤติกรรม

- 4.1 อธิบายหลักการทำงานของ 7-Segment Display แบบธรรมดาและแบบ I2C ได้
- 4.2 อธิบายโครงสร้างและการทำงานของ LCD 16x2 แบบ I2C ได้
- 4.3 เชื่อมต่อวงจร 7-Segment Display แบบธรรมดากับ ESP32 ได้ถูกต้อง
- 4.4 เขียนโปรแกรมแสดงผลตัวเลขบน 7-Segment ได้
- 4.5 เชื่อมต่อและใช้งาน 7-Segment Display แบบ I2C 4 หลักได้
- 4.6 เชื่อมต่อและเขียนโปรแกรมแสดงผลบน LCD 16x2 แบบ I2C ได้
- 4.7 ประยุกต์ใช้สวิตช์ควบคุมการแสดงผลได้

5. เครื่องมือ วัสดุ และอุปกรณ์

- 5.1 Arduino IDE 2..3.6
- 5.2 ชุดฝึกปฏิบัติ

6. คำแนะนำ/ข้อควรระวัง

-

7. ขั้นตอนการปฏิบัติงาน

7.1 ต่อวงจร LCD 16x2 แบบ I2C เข้ากับ ESP32

7.2 เขียนโปรแกรมให้ LCD 16x2 แบบ I2C ทำงานตามลำดับที่กำหนด

7.3 ทดสอบและปรับแก้โปรแกรม

8. สรุปและวิจารณ์ผล

-

9. การประเมินผล

1. แบบประเมินสมรรถนะงานภาคปฏิบัติ (6 คะแนน)

ความถูกต้องของการต่อวงจร

ความถูกต้องของโปรแกรม

ความสมบูรณ์ของผลลัพธ์

2. แบบสังเกตพฤติกรรมลักษณะนิสัยการทำงาน (4 คะแนน)

ตรงต่อเวลา

ความร่วมมือและวินัย

10. เอกสารอ้างอิง /เอกสารค้นคว้าเพิ่มเติม

-