ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัส ทีซีพี/ไอพี

PROTOTYPE OF REMOTE INPUT/OUTPUT MODULE USING MODBUS TCP/IP

ณัฐพงษ์ อยู่สุข
รุ่งเกียรติ พัทธนันทพงศ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอัตโนมัติ
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2560

ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัส ทีซีพี/ไอพี

PROTOTYPE OF REMOTE INPUT/OUTPUT MODULE USING MODBUS TCP/IP

ณัฐพงษ์ อยู่สุข
รุ่งเกียรติ พัทธนันทพงศ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอัตโนมัติ
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2560

PROTOTYPE OF REMOTE INPUT/OUTPUT MODULE USING MODBUS TCP/IP

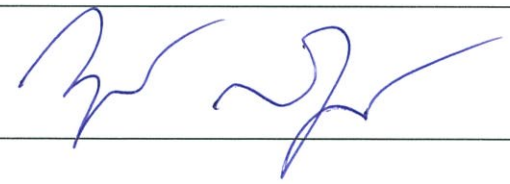
NATTAPONG YOOSUK
RUNGKIAT PHATTHANANTAPONG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF ENGINEERING IN AUTOMATION ENGINEERING
FACULTY OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2017

ปริญญานิพนธ์ปีการศึกษา 2560
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองปริญญานิพนธ์

.....

หัวข้อปริญญานิพนธ์	ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัส TCP/IP		
นักศึกษาผู้จัดทำ	นายณัฐพงษ์	อยู่สุข	**รหัสนักศึกษา** 57010436
	นายรุ่งเกียรติ	พัทธนันทพงศ์	**รหัสนักศึกษา** 57011068
ปริญญา	วิศวกรรมศาสตรบัณฑิต		
สาขาวิชา	วิศวกรรมอัตโนมัติ		
ปีการศึกษา	2560		

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
ผศ.ดร. กฤษณ์ เสมอพิทักษ์	

หัวข้อปริญญานิพนธ์	ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัส Prototype of Remote Input/Output Module Using Modbus TCP/IP		
นักศึกษาผู้จัดทำ	นายณัฐพงษ์	อยู่สุข	รหัสนักศึกษา 57010436
นักศึกษาผู้จัดทำ	นายรุ่งเกียรติ	พัทธนันทพงศ์	รหัสนักศึกษา 57011068
อาจารย์ที่ปรึกษา	ผศ.ดร. กฤษณ์	เสมอพิทักษ์	
ปีการศึกษา	2560		

บทคัดย่อ

โครงงานนี้นำเสนอเทคนิคการสร้างชุดต้นแบบอินพุทเอาท์พุทโดยใช้ไมโครคอนโทรลเลอร์ในการทำชุดรีโมทอินพุทเอาท์พุทผ่านโปรโตคอลมอดบัสทีซีพีไอพีมาสเตอร์ จุดประสงค์ของชุดต้นแบบโดยใช้ไมโครคอนโทรลเลอร์ ESP32 WeMos LOLIN32 รองรับดิจิตอลอินพุท 8 ช่องและดิจิตอลเอาท์พุท 8 ช่อง รวมทั้งอะนาล็อกอินพุท 2 ช่องและอะนาล็อกเอาท์พุท 2 ช่อง โปรแกรมมอดสแกนใช้เพื่อทดสอบความถูกต้องในการอ่านและเขียนข้อมูลของชุดรีโมทอินพุทเอาท์พุทที่นำเสนอ เพื่อที่จะตรวจสอบประสิทธิภาพการทำงานได้ใช้มอดบัสสเลฟ 2 แบบที่แตกต่างกัน คือ พีแอลซีย่ี่ห้อซีเมนที่ต่อกับตัวแสดงผล HMI และหน้าจอแสดงผลบนซอฟต์แวร์วันเดอร์แวร์ที่สามารถติดต่อกับผู้ใช้งาน จากผลการทดลองยืนยันได้ว่า ชุดต้นแบบสามารถทำงานได้อย่างถูกต้อง ทั้งในฟังก์ชั่นอ่านและเขียน

Thesis Title	Prototype of Remote Input/Output Module Using Modbus TCP/IP
Authors	Mr.Nattapong Yoosuk
	Mr.Rungkiat Phatthanantapong
Thesis Advisor	Asst.Prof. Dr. Krit Smerpitak

ABSTRACT

This thesis presents a technique for implementing a prototype of microcontroller-based remote input/output (I/O) module to perform as a Modbus TCP/IP master. The proposed prototype utilizing the microcontroller modeled ESP32 WeMos LOLIN32 supports 8 digital inputs and outputs as well as 2 analog inputs and outputs. The ModScan software was used to test the correction of read and write data of the proposed remote I/O module. In order to verify the performance, the implemented prototype was connected with 2 different Modbus TCP/IP slaves which are Siemens S7 Programmable Logic Controller (PLC) connected with a Human Machine Interface (HMI) panel and HMI station running the Wonderware InTouch software. Experimentally test results confirm that the proposed prototype can perform correctly in both read and write functions.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความกรุณาจากคณาจารย์ สาขาวิศวกรรมอัตโนมัติ คณะวิศวกรรมศาสตร์ ที่ได้เสียสละเวลาอันมีค่าคอยเป็นที่ปรึกษาและให้คำแนะนำอย่างดี อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือ ที่เป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้ และให้ความช่วยเหลือ ทั้งในด้านทุนทรัพย์ แรงงาน และกำลังใจ

นอกจากนี้ผู้วิจัยขอขอบคุณ พ่อ แม่ อันเป็นที่รักที่สนับสนุน และเป็นแรงบันดาลใจในการทำงานปริญญานิพนธ์ฉบับนี้เสมอมา คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ(ต่อ)

ภาพที่ หน้า

สารบัญภาพ(ต่อ)

ภาพที่ หน้า

สารบัญภาพ(ต่อ)

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาของปริญญานิพนธ์

สำหรับการใช้งานตัวควบคุมที่เรียกว่า พีแอลซี (Programmable Logic Controller : PLC) [1] เชิงอุตสาหกรรม ในอดีตนั้นจะมีการเชื่อมต่อผ่านทางการ์ดอินพุทและเอาท์พุทภายในตัวของเครื่องพีแอลซี โดยมีรูปแบบการเชื่อมต่อเป็นแบบจุดต่อจุด (Point to Point connection) ทำให้สายการเชื่อมต่อจะมีปริมาณสายที่ขึ้นอยู่กับจำนวนอินพุทและเอาท์พุทที่เชื่อมต่อ รวมไปถึงค่าใช้จ่ายสำหรับการเชื่อมต่อสายของฝั่งอินพุทและเอาท์พุทในปริมาณที่ค่อนข้างสูงอันเนื่องมาจากระยะทางระหว่างพื้นที่ปฏิบัติงานและเครื่องพีแอลซีที่ติดตั้งในห้องควบคุมระยะทางไกล จึงทำให้ค่าใช้จ่ายสูงขึ้นตามระยะทางและจำนวนอินพุทเอาท์พุทที่ใช้ ต่อมามีการพัฒนาเพื่อลดปัญหาดังกล่าวโดยจัดทำเป็น รีโมทอินพุท/เอาท์พุท (Remote I/O) [2] เพื่อจัดการแก้ปัญหาในเรื่องของสายสำหรับการเชื่อมต่อ และนำเทคโนโลยีในเรื่องของมอดบัสทีซีพี/ไอพี (Modbus TCP/IP) [3] มาประยุกต์ใช้งานร่วมด้วย

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์ได้จัดทำขึ้นมาโดยมีวัตถุประสงค์เพื่อสร้างชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี เพื่อเป็นแนวทางหนึ่งในการลดความยุ่งยากในเรื่องของการติดตั้งและลดค่าใช้จ่ายของสาย โดยชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยสื่อสารผ่านโปรโตคอล Modbus TCP/IP ในการรับและส่งค่าอินพุท/เอาท์พุทรีจิสเตอร์ [4] ของอุปกรณ์ต่าง ๆ ที่ต้องการ โดยชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพีนี้จะทำสร้างตัวเองเป็น Master ในการรับและส่งค่า และใช้อุปกรณ์ที่สามารถสื่อสารผ่าน Modbus TCP/IP เป็น Slave อาทิเช่น PLC [5] และ WONDERWARE [6] อีกทั้งยังสามารถเปลี่ยนแปลงเครือข่ายผ่านทางโปรโตคอลเฮทที่ทีพี (HyperText Transfer Protocol: HTTP) [7] ได้ เพื่อให้ผู้ที่สนใจมีความเข้าใจรวมไปถึงสามารถนำไปพัฒนาต่อยอดได้

1.3 ขอบเขตของปริญญานิพนธ์

1. ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพีนี้สามารถรับดิจิตอลอินพุท (Digital Input : DI) และส่งดิจิตอลเอาท์พุท (Digital Output : DO) ได้อย่างละ 8 ช่อง และสามารถรับอะนาล็อกอินพุท (Analog Input : AI) และส่งอะนาล็อกเอาท์พุท (Analog Output : AO) ได้อย่างละ 2 ช่อง

2. ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทสื่อสารข้อมูลผ่านโปรโตคอล MODBUS TCP/IP

3. ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทมีจอแสดงผลแอลซีดีเพื่อดูสถานะการเชื่อมต่อเน็ตเวิร์ค

4. ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทสามารถเปลี่ยนไอพีแอดเดรสผ่านทางโปรโตคอล HTTP

1.4 ขั้นตอนการศึกษา

1. ศึกษาการทำงานของ Remote INPUT/OUTPUT
2. ศึกษาทฤษฎีการสื่อสารของโปรโตคอล MODBUS TCP/IP
3. ศึกษาและเขียนโปรแกรมควบคุมด้วยไมโครคอนโทรลเลอร์ ESP32 WeMos LOLIN32
4. ออกแบบโครงร่างของชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี
5. ออกแบบและเขียนโปรแกรมไมโครคอนโทรลเลอร์ ESP32
6. จัดทำชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี
7. ทดสอบการทำงานของชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี
8. ปรับปรุงและแก้ไขข้อบกพร่องของชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี
9. สรุปผลและเขียนรายงานฉบับสมบูรณ์

ตารางที่ 1.1 แผนการดำเนินงาน

ลำดับ	แผนการดำเนินงาน	สัปดาห์ที่ 1-4	สัปดาห์ที่ 5-8	สัปดาห์ที่ 9-13	สัปดาห์ที่ 14-17	สัปดาห์ที่ 18-21
1	ศึกษาการทำงานของ Remote Input/Output					
2	ศึกษาทฤษฎีการสื่อสารของโปรโตคอล MODBUS TCP/IP					
3	.ศึกษาและเขียนโปรแกรมควบคุมด้วยไมโครคอนโทรลเลอร์ ESP32 WeMos LOLIN32					
4	ออกแบบโครงร่างของชุดต้นแบบ					
5	ออกแบบและเขียนโปรแกรมไมโครคอนโทรลเลอร์สำหรับรับส่งค่าข้อมูลจาก MODBUS TCP/IP					
6	จัดทำชุดต้นแบบ					
7	ทดสอบการทำงานของชุดต้นแบบ					
8	ปรับปรุงและแก้ไขข้อบกพร่องของชุดต้นแบบโมดูล					
9	สรุปผลและเขียนรายงานฉบับสมบูรณ์					

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี สามารถรับค่าดิจิตอลอินพุทและอะนาล็อกอินพุทเพื่อแสดงค่าที่ต้องการ และสามารถส่งค่าดิจิตอลเอาท์พุทและอะนาล็อกเอาท์พุทเพื่อใช้ควบคุมอุปกรณ์ต่าง ๆ ได้ตามที่ต้องการ

2. ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพีช่วยลดความยุ่งยากในเรื่องการติดตั้งและลดค่าใช้จ่ายของสายไฟหรือสายสัญญาณ

3. นำมาใช้ในการเรียนการสอนหรือการทดลองในห้องปฏิบัติการ

1.6 รายละเอียดของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้จัดทำทั้งหมด 5 บท โดยแต่ละบทมีรายละเอียดดังนี้

บทที่ 1 บทนำ อธิบายถึงความเป็นมา ความสำคัญ วัตถุประสงค์ ขอบเขต ขั้นตอนการศึกษา และประโยชน์ที่คาดว่าจะได้รับจากปริญญานิพนธ์

บทที่ 2 ทฤษฎีและหลักการที่เกี่ยวข้อง ได้แก่ พื้นฐานการติดต่อสื่อสารข้อมูลผ่านโปรโตคอลมอดบัสทีซีพีไอพี การตั้งค่าและการใช้งานของไมโครคอนโทรลเลอร์ ESP32 และซอฟต์แวร์ควบคุมไมโครคอนโทรลเลอร์ Arduino [7]

บทที่ 3 ขั้นตอนการดำเนินงาน การออกแบบส่วนฮาร์ดแวร์และซอฟต์แวร์ รวมทั้งการเขียนโค้ดการทำงานของไมโครคอนโทรลเลอร์ ESP32

บทที่ 4 ผลการดำเนินงาน ในบทนี้อธิบายถึงการทดสอบ และผลการทดสอบกับโปรแกรม MODSCAN PLC และ WONDERWARE InTouch

บทที่ 5 สรุปผลการดำเนินงาน ในบทนี้อธิบายถึงบทสรุปของปริญญานิพนธ์ และปัญหาที่เกิดขึ้นระหว่างทำการทดลอง

บทที่ 2

หลักการที่เกี่ยวข้อง

2.1 กล่าวนำ

ในสมัยอดีตการใช้งานพีแอลซีในงานอุตสาหกรรมมีการใช้งานที่ยุ่งยากและมีค่าใช้จ่ายของสายไฟและสายสัญญาณที่สูง ต่อมามีการแก้ปัญหาดังกล่าวโดยการใช้อุปกรณ์ควบคุมระยะไกลมาแทนที่เพื่อลดปัญหาดังกล่าว และเพิ่มประสิทธิภาพโดยใช้โปรโตคอลมอดบัส ซึ่งสามารถควบคุมได้ระยะไกลโดยการเชื่อมต่อผ่านอินเตอร์เน็ต

ในบทนี้กล่าวถึงหลักการที่เกี่ยวข้องกับอุปกรณ์ควบคุมระยะไกลที่ใช้โปรโตคอลมอดบัส โดยกล่าวถึง หลักการของพีแอลซี อุปกรณ์ควบคุมระยะไกล โปรโตคอลมอดบัส โปรแกรมที่สามารถเชื่อมต่อกับโปรโตคอลมอดบัส บอร์ดที่สามารถพัฒนาให้เป็นตัวรับ-ส่งข้อมูลระยะไกล และโปรแกรมสำหรับเขียนไมโครคอนโทรลเลอร์ จากหลักการที่กล่าวมาสามารถใช้สร้างชุดต้นแบบโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพี โดยมีการออกแบบส่วนประกอบต่าง ๆ ภายในของชุดต้นแบบ ทั้งส่วนที่เป็นฮาร์ดแวร์และส่วนที่เป็นซอฟต์แวร์ เพื่อให้ชุดต้นแบบสามารถทำงานได้อย่างถูกต้อง

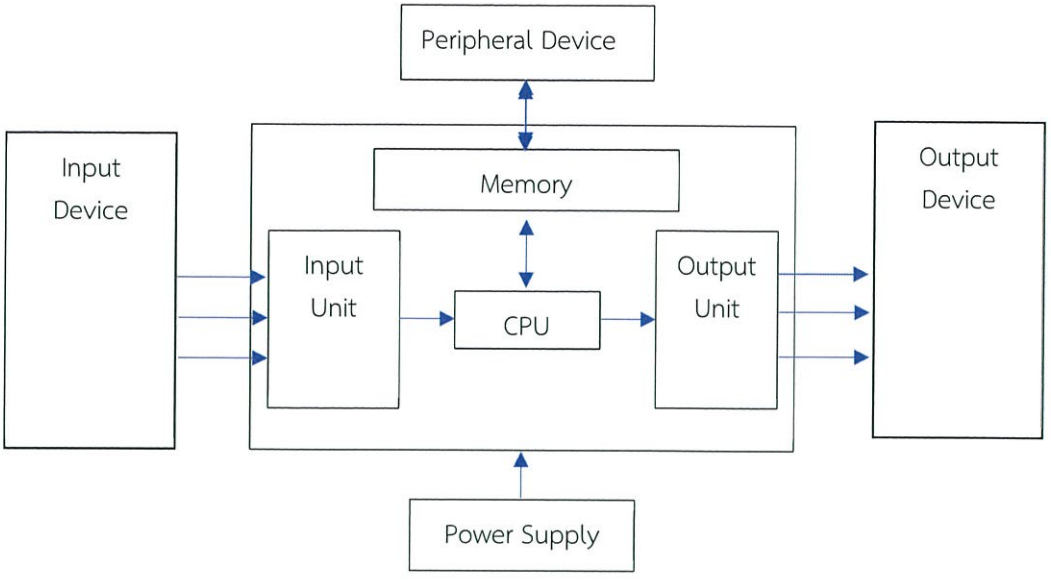
2.2 พีแอลซี (Programmable Logic Controller : PLC)

2.2.1 ความหมายของพีแอลซี

พีแอลซี เป็นอุปกรณ์ควบคุมการทำงานของเครื่องจักรหรือกระบวนการทำงานต่าง ๆ โดยภายในมี Microprocessor เป็นมันสมองสั่งการที่สำคัญและมีส่วนที่เป็นอินพุทและเอาท์พุทที่สามารถต่อออกไปใช้งานได้ทันที โดยตัวตรวจวัดหรือสวิตช์ต่าง ๆ จะต่อเข้ากับอินพุท ส่วนเอาท์พุทจะใช้ต่อออกไปควบคุมการทำงานของอุปกรณ์หรือเครื่องจักรที่เป็นเป้าหมาย พีแอลซีสามารถเขียนโปรแกรมควบคุมได้โดยการป้อนเป็นโปรแกรมคำสั่งเข้าไปในพีแอลซี นอกจากนี้ยังสามารถใช้งานร่วมกับอุปกรณ์อื่น ๆ เช่น เครื่องอ่านบาร์โค้ด (Barcode Reader) เครื่องพิมพ์ (Printer) อุปกรณ์แสดงผล ซึ่งในปัจจุบันนอกจากเครื่องพีแอลซีจะใช้งานแบบเดี่ยว (Stand alone) แล้วยังสามารถต่อพีแอลซีหลาย ๆ ตัวเข้าด้วยกันเป็นเครือข่าย (Network) เพื่อควบคุมการทำงานของระบบให้มีประสิทธิภาพมากยิ่งขึ้น จะเห็นได้ว่าการใช้งานพีแอลซีมีความยืดหยุ่นมาก ดังนั้นในโรงงานอุตสาหกรรมต่าง ๆ จึงใช้พีแอลซีในการควบคุมกระบวนการหรือเครื่องจักร

2.2.2 โครงสร้างของพีแอลซี

พีแอลซีเป็นอุปกรณ์คอมพิวเตอร์สำหรับใช้ในงานอุตสาหกรรม พีแอลซีประกอบด้วยหน่วยประมวลผลกลาง หน่วยรับข้อมูล หน่วยส่งข้อมูล หน่วยความจำ ส่วนอุปกรณ์ต่อพ่วง และแหล่งจ่ายไฟ ดังภาพที่ 2.1



ภาพที่ 2.1 โครงสร้างของพีแอลซี

จากภาพที่ 2.1 สามารถแบ่งโครงสร้างพีแอลซีได้เป็น 6 ส่วน คือ

1. ส่วนที่เป็นหน่วยประมวลผลกลาง (Control Processing Unit : CPU)
2. ส่วนที่เป็นอินพุท (Input Unit)
3. ส่วนที่เป็นเอาท์พุท (Output Unit)
4. ส่วนของหน่วยความจำ (Memory)
5. ส่วนอุปกรณ์ต่อพ่วง (Peripherals Units)
6. แหล่งจ่ายไฟ (Power Supply)

โครงสร้างของพีแอลซีสามารถอธิบายเป็นส่วน ๆ ได้ดังนี้

1. ส่วนที่เป็นหน่วยประมวลผลกลาง (Control Processing Unit : CPU)

CPU เป็นส่วนมันสมองของระบบ ภายใน CPU จะประกอบไปด้วยวงจร Logic Gate ชนิดต่าง ๆ หลายชนิด และมี Microprocessor-based ใช้สำหรับจำลองอุปกรณ์จำพวกรีเลย์ (Relay) เคาน์เตอร์ (Counter) ไทเมอร์ (Timer) และซีเควนเซอร์ (Sequencers) และประมวลผลตามโปรแกรมที่บันทึกไว้

CPU จะอ่านข้อมูลอินพุทจากอุปกรณ์ให้สัญญาณ (Sensing Device) ต่าง ๆ จากนั้นจะปฏิบัติการและเก็บข้อมูลโดยใช้โปรแกรมจากหน่วยความจำ และส่งข้อมูลที่เหมาะสมถูกต้องไปยังอุปกรณ์ควบคุม (Control Device)

การประมวลผลโปรแกรมโดย CPU ทำได้โดยรับข้อมูลจากหน่วยอินพุท และส่งข้อมูลสุดท้ายที่ได้จากการประมวลผลไปยังหน่วยเอาท์พุท เรียกว่า การสแกน (Scan) ซึ่งใช้เวลาจำนวนหนึ่ง ที่เรียกว่า เวลาสแกน (Scan Time) เวลาในการสแกนแต่ละรอบใช้เวลาประมาณ 1 ถึง 100 msec. (0.001 - 0.1 วินาที) ทั้งนี้ขึ้นอยู่กับข้อมูลและความยาวของโปรแกรม หรือจำนวนอินพุทเอาท์พุทหรือจำนวนอุปกรณ์ที่ต่อจากพีแอลซี เช่น เครื่องพิมพ์ จอภาพ เป็นต้น อุปกรณ์เหล่านี้จะทำให้เวลาในการสแกนยาวนานขึ้น การเริ่มต้นการสแกนเริ่มจากอ่านสภาวะของอุปกรณ์จากหน่วย

อินพุทมาเก็บไว้ในหน่วยความจำ (Memory) เสร็จแล้วจะทำการปฏิบัติการตามโปรแกรมที่เขียนไว้ทีละคำสั่งจากหน่วยความจำนั้นจนสิ้นสุด แล้วส่งผลการคำนวณไปที่หน่วยเอาท์พุท

2. ส่วนที่เป็นอินพุท (Input unit)

ส่วนที่เป็นอินพุทจะใช้รับสัญญาณอินพุทจากภายนอกที่เป็นสวิตช์และตัวตรวจจับชนิดต่าง ๆ โดยมีวงจรแยกสัญญาณ (Isolate) ซึ่งอาศัยอุปกรณ์ประเภทโฟโตทรานซิสเตอร์ทำหน้าที่แยกสัญญาณอินพุทจากภายนอกกับ CPU เป็นการป้องกันไม่ให้ CPU เสียหายเมื่ออินพุทมีสัญญาณมากเกิน สัญญาณอินพุทจากอุปกรณ์ต่าง ๆ ที่เข้ามาจะถูกส่วนที่เป็นอินพุทเปลี่ยนให้เป็นสัญญาณที่มีระดับที่เหมาะสมกับพีแอลซี ไม่ว่าจะเป็น AC หรือ DC เพื่อส่งให้ CPU

อุปกรณ์ที่ใช้เป็นสัญญาณอินพุท ได้แก่ พรอกซิมิตี้สวิตช์ ลิมิตสวิตช์ ไทเมอร์ โฟโตอิเล็กริกสวิตช์ เอนโค้ดเดอร์ เคาน์เตอร์

3. ส่วนที่เป็นเอาท์พุท (Output Unit)

ในส่วนของเอาท์พุท จะทำหน้าที่รับค่าสภาวะที่ได้จากการประมวลผลของ CPU แล้วนำค่าเหล่านี้ไปควบคุมอุปกรณ์ทำงาน เช่น รีเลย์ โซลีนอยด์ หรือหลอดไฟ เป็นต้น นอกจากนั้นแล้ว ยังทำหน้าที่แยกสัญญาณของ CPU ออกจากอุปกรณ์เอาท์พุท โดยปกติเอาท์พุทนี้จะมีความสามารถขับโหลดด้วยกระแสไฟฟ้าประมาณ 1 - 2 แอมแปร์ แต่ถ้าโหลดต้องการกระแสไฟฟ้ามากกว่านี้ จะต้องต่อเข้ากับอุปกรณ์ขับอื่นเพื่อขยายให้รับกระแสไฟฟ้ามากขึ้น เช่น คอนแทคเตอร์ เป็นต้น

อุปกรณ์ที่ใช้เป็นเอาท์พุท ได้แก่ รีเลย์ มอเตอร์ไฟฟ้า โซลินอยด์ หลอดไฟ ขดลวดความร้อน หรือคอนแทคเตอร์

4. ส่วนของหน่วยความจำ (Memory)

ส่วนของหน่วยความจำทำหน้าที่เก็บรักษาโปรแกรมและข้อมูลที่ใช้ในการทำงานของพีแอลซี โดยแบ่งประเภทของหน่วยความจำดังนี้ ROM (Read Only Memory) RAM (Random Access Memory) PROM (Programmable Read Only Memory) EPROM (Erasable Programmable Read Only Memory)

ROM ถูกออกแบบมาให้เก็บโปรแกรมอย่างถาวร ไม่สามารถที่เปลี่ยนแปลงตัวโปรแกรมที่อยู่ใน ROM ที่เขียนครั้งแรกได้ ภายใต้สภาวะปกติ ROM มีความสามารถในการต้านทานการสูญเสียข้อมูลจากสัญญาณรบกวนทางไฟฟ้า

RAM บางครั้งอาจจะเรียกว่า หน่วยความจำแบบ อ่านเขียน (Read/Write) หรือ R/W ถูกออกแบบเพื่อให้ข้อมูลสามารถเขียนลงไปในพื้นที่หน่วยความจำได้ และสามารถอ่านหน่วยความจำที่ถูกเขียนแล้วได้ แต่ข้อมูลและโปรแกรมที่เขียนใน RAM จะไม่สามารถอยู่ได้เมื่อขาดกระแสไฟฟ้า ดังนั้น RAM จึงเป็นหน่วยความจำแบบชั่วคราว

PROM เป็นหน่วยความจำแบบ ROM ชนิดพิเศษ สามารถที่จะโปรแกรมได้ภายในตัว เมื่อนำ PROM มาใช้ จะนำมาใช้เก็บข้อมูลสำรองบางชนิดแบบถาวร ถึงแม้ว่าจะมีข้อได้เปรียบจากการที่เขียนโปรแกรมใหม่ได้ แต่ก็มีข้อเสียคือต้องการอุปกรณ์การเขียนโปรแกรมชนิดพิเศษ ดังนั้นเมื่อ PROM ถูกบรรจุข้อมูลหรือโปรแกรมครั้งแรกแล้ว เมื่อทำการลบโปรแกรมหรือเขียนโปรแกรมใหม่จึงทำได้ยาก

EPROM เป็นหน่วยความจำแบบ ROM ชนิดพิเศษ สามารถที่จะโปรแกรมได้โดยใช้สัญญาณไฟฟ้า เมื่อนำ EPROM มาใช้จะนำมาใช้เก็บข้อมูลสำรองบางชนิดแบบถาวรได้ และมีข้อดี

คือไม่ต้องการอุปกรณ์การเขียนโปรแกรมชนิดพิเศษ สามารถเขียนได้ด้วยโปรแกรม ดังนั้นจึงนิยมใช้ในปัจจุบัน เช่น แฟลชไดร์

5. ส่วนอุปกรณ์ต่อพ่วง (Peripherals Units)

ส่วนอุปกรณ์ต่อพ่วงเป็นหน่วยที่ใช้เพิ่มความสามารถให้กับตัวพีแอลซี เช่น เมื่อต้องการเพิ่มจำนวนช่องรับสัญญาณอินพุทหรือช่องส่งสัญญาณเอาท์พุท สามารถนำอุปกรณ์ต่อพ่วง เช่น อุปกรณ์รีโมทไอโอ (Remote I/O) หรือต้องการควบคุมและแสดงผลผ่านทางหน้าจอทัชสกรีน (Touch Screen) หรือใช้เชื่อมต่อกันเป็นระบบ

6. แหล่งจ่ายไฟ (Power Supply)

แหล่งจ่ายไฟทำหน้าที่จ่ายพลังงานและรักษาระดับแรงดันไฟฟ้ากระแสตรงให้กับพีแอลซี

2.2.3 ชนิดของพีแอลซี

การแบ่งชนิดของพีแอลซีจะแบ่งตามบรรจุภัณฑ์หรือลักษณะโครงสร้างภายนอก แบ่งเป็น 4 ชนิด ดังนี้

1. พีแอลซีบอร์ด (PLC Board)
2. พีแอลซีกะทัดรัด (PLC Compact)
3. พีแอลซีแยกส่วน (PLC Modular)
4. ซอฟต์แวร์พีแอลซี (Software PLC)

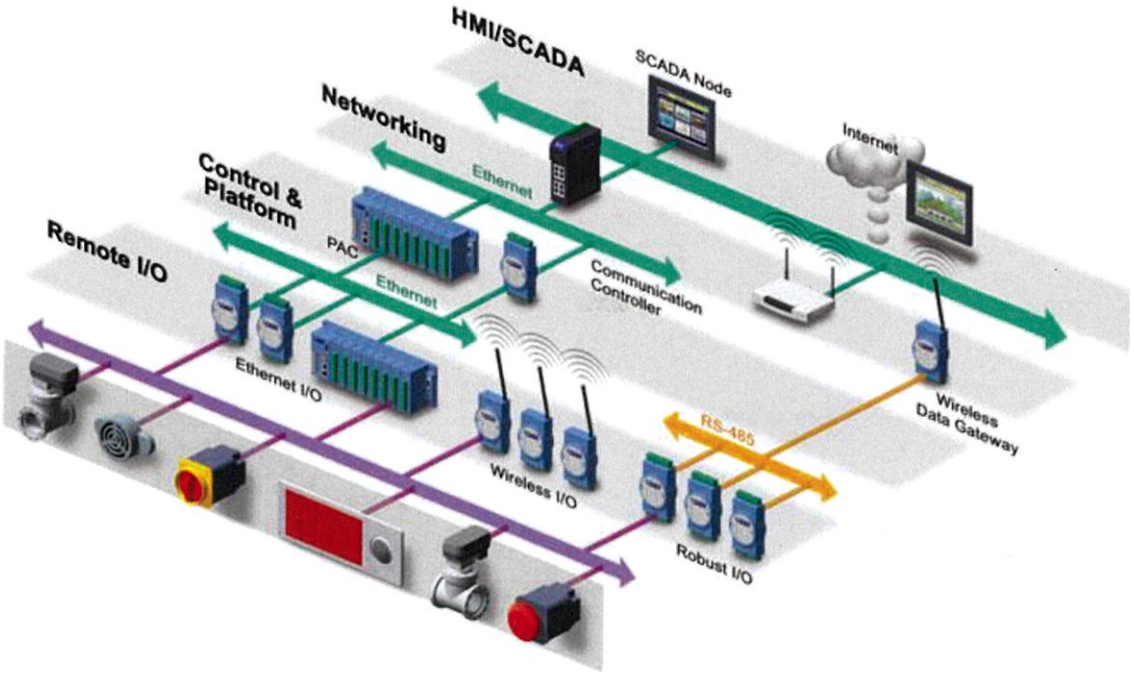
2.3 อุปกรณ์รับ-ส่งอินพุทเอาท์พุทระยะไกล (Remote INPUT/OUTPUT Module)

อุปกรณ์รับ-ส่งอินพุทเอาท์พุทระยะไกลคือ อุปกรณ์สำหรับ รับ-ส่ง ข้อมูลสัญญาณ analog หรือ ข้อมูล digital โดยจะรับคำสั่งมาจากคอมพิวเตอร์ หรืออุปกรณ์ควบคุมต่าง ๆ เช่น พีแอลซี DCS (Distributed Control System) โดยผ่านทางช่องสื่อสาร เช่น RS-232 RS-422 หรือ RS-485 ภายใต้ MODBUS (ASCII RTU) Ethernet (Profinet MODBUS TCP/IP)

ระบบ SCADA (Supervisory Control and Data Acquisition) หรือพีแอลซี ที่ทำหน้าที่เป็นระบบควบคุมและแสดงผลในระยะไกล อุปกรณ์รับส่งข้อมูลระยะไกลหรือ RTU (Remote Terminal Unit) จึงเป็นส่วนสำคัญในการรับค่าอินพุทและสั่งงานเอาท์พุทรวมถึงการแสดงผล

จากภาพที่ 2.2 ด้านล่างเป็นลักษณะการเชื่อมต่ออุปกรณ์รับ-ส่งอินพุทเอาท์พุทระยะไกล จะเห็นได้ว่า อุปกรณ์รับ-ส่งอินพุทเอาท์พุทระยะไกลเป็นอุปกรณ์ที่คอยรับและส่งข้อมูลจาก อุปกรณ์รับรู้ต่าง ๆ ที่อยู่บนพื้นที่ปฏิบัติงาน เช่น สวิตซ์ ทรานสมิสเตอร์ เป็นต้น เพื่อส่งข้อมูลไปยังส่วนควบคุมโดยผ่านโปรโตคอลต่าง ๆ ตามความสามารถและลักษณะการใช้งานของอุปกรณ์รับ-ส่งอินพุทเอาท์พุทระยะไกลแต่ละแบบ

การกำหนดให้สถานีสนาม (Field) ทำหน้าที่รับส่งข้อมูลในกระบวนการไปยังศูนย์ควบคุม ซึ่งมีระยะทางไกล ดังนั้นการเลือกการเชื่อมต่อหรือโปรโตคอลในการติดต่อกับ RTU จึงเป็นปัจจัยในการตัดสินใจเลือกใช้อุปกรณ์รับส่งอินพุทเอาท์พุทระยะไกล



ภาพที่ 2.2 การเชื่อมต่ออุปกรณ์รับ-ส่งอินพุทเอาท์พุทระยะไกล

2.4 โปรโตคอลมอดบัส (Modbus Protocol)

โปรโตคอลมอดบัส เป็นโปรโตคอลเพื่อสื่อสารข้อมูลอินพุทเอาท์พุทและรีจีสเตอร์ที่สามารถเชื่อมต่อกับตัวควบคุม เช่น พีแอลซี SCADA และ อื่น ๆ ซึ่งถูกคิดค้นโดย Modicon (ปัจจุบันคือบริษัท Schneider Electric) มีการสื่อสารข้อมูลในลักษณะ Master/Slave ซึ่งเป็นการสื่อสารจากอุปกรณ์แม่ (Master) เครื่องเดียว ไปยังอุปกรณ์ลูก (Slave) ได้หลาย ๆ เครื่อง โปรโตคอลมอดบัสเป็นที่ยอมรับกันอย่างกว้างขวางในการติดต่อสื่อสารที่เป็นแบบ Network Protocol เนื่องมาจากโปรโตคอลมอดบัสใช้การสื่อสารแบบ RS422 , RS485 และ Ethernet โปรโตคอลมอดบัสเป็นระบบเปิด ไม่มีค่าใช้จ่าย เชื่อมต่อและพัฒนาง่าย พร้อมทั้งยังสามารถนำโปรโตคอลนี้ไปใช้งานในอุปกรณ์อื่น ๆ เช่น Digital Power Meter , Remote INPUT/OUTPUT , พีแอลซี เป็นต้น นอกจากนี้มอดบัส ยังสามารถรองรับและใช้งานร่วมกับซอฟต์แวร์ที่มีใช้ในอุตสาหกรรม เช่น WONDERWARE การรับส่งข้อมูลในโปรโตคอลมอดบัส มีรีจิสเตอร์ที่สำคัญ 4 รีจิสเตอร์ ตามตารางที่ 2.1

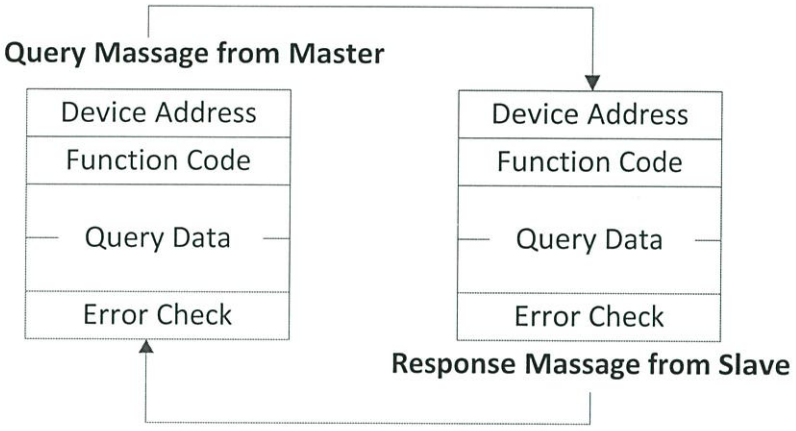
ตารางที่ 2.1 รีจิสเตอร์ของ Modbus

Register type	Use as	Access	Data type
Coil	Digital Output	Read/Write	Bit
Holding Register	Analog Output	Read/Write	Word
Input Status	Digital Input	Read Only	Bit
Input Register	Analog Input	Read Only	Word

2.4.1 โปรโตคอล MODBUS RTU

โปรโตคอล MODBUS RTU เป็นการสื่อสารข้อมูลในลักษณะ Master/Slave ด้วยการสื่อสารแบบ RS422 หรือ RS485 โดยสามารถกำหนดหมายเลขอุปกรณ์ได้สูงสุด 255 เครื่อง โดยมีลักษณะการส่งข้อมูลแบบเลขฐานสอง

โปรโตคอล MODBUS มีรูปแบบในการส่ง Query Message จาก Master ไปยัง Slave ตัวอื่น ๆ โดยมี Device Address ระบุตำแหน่งของ Slave มี Function Code ที่ใช้กำหนดวัตถุประสงค์ของการส่งข้อมูลให้กับ Slave มี Query Data ข้อมูลต่าง ๆ ที่จะทำการส่งและมี Error Check ที่สร้างจากข้อมูลที่ต้องการส่งเพื่อใช้ตรวจสอบข้อผิดพลาดในการส่ง ส่วนรูปแบบของ Response Message จาก Slave ที่ส่งกลับมายัง Master จะมี Device Address ของ Slave ที่ตอบกลับ มี Function Code ที่ยืนยันการทำงานตามคำสั่งของ Master มี Response Data ข้อมูลต่าง ๆ ที่ต้องส่งกลับไปและมี Error Check ที่สร้างจากข้อมูลที่ต้องการส่งเพื่อใช้ตรวจสอบข้อผิดพลาดในการส่ง ดังภาพที่ 2.4



ภาพที่ 2.3 Master-Slave Query-Response Cycle

เฟรมข้อมูลในโหมด RTU แสดงดังภาพที่ 2.3 ประกอบไปด้วยข้อมูลตำแหน่งแอดเดรส 8 บิต หมายเลขฟังก์ชัน 8 บิต ข้อมูลที่ทำการรับ-ส่งจำนวนมากสุดไม่เกิน 252 ไบต์ และรหัสตรวจสอบความถูกต้องของข้อมูลแบบ CRC (Cyclical Redundancy Checking) ขนาด 16 บิต

ADDRESS	FUNCTION	DATA	CRC
8 BITS	8 BITS	8 BITS	16 BITS

ภาพที่ 2.4 เฟรมข้อมูลของ MODBUS RTU

จากภาพที่ 2.4 เฟรมข้อมูลของ MODBUS RTU ประกอบด้วย Address, Function Data, CRC โดยตำแหน่งแรกเป็นแอดเดรสของ Slave ที่สามารถกำหนดอยู่ในช่วง 0-255 (ฐานสิบ)

Slave แต่ละตัวจะถูกกำหนดตำแหน่งไว้ในช่วง 1-255 เนื่องจากตำแหน่งที่ศูนย์ (Address 0) ถูกใช้สำหรับ broadcast query ตัว Master สามารถระบุตำแหน่งของ Slave ที่ต้องการสื่อสารได้โดยใส่ตำแหน่งของ Slave ตัวนั้น ๆ ไปในตำแหน่งแอดเดรส เมื่อ Slave ส่งการตอบสนอง (Response) กลับมาจะทำการใส่ตำแหน่งของตัวมันเองลงในตำแหน่งแอดเดรสของการตอบสนองเพื่อให้ Master รู้ว่าการตอบสนองที่ได้รับนั้นเป็นของ Slave ตัวใด

ตำแหน่งฟังก์ชัน สามารถกำหนดได้อยู่ในช่วง 1-255 (ฐานสิบ) โดยเป็นตัวบอกให้ Slave ทราบว่าคำสั่งที่ส่งมานั้นเป็นชนิดใดและต้องการให้ทำอะไร เมื่อ Slave ทำการตอบสนองต่อคำสั่งที่สั่งจาก Master แล้วจะใช้ Function code เป็นตัวบอกสถานการณ์การทำงานว่าเป็นปรกติหรือเกิดความผิดพลาดขึ้นแล้วจึงแจ้งให้ Master ทราบโดยถ้าการทำงานเป็นปรกติจะส่ง Function code ตัวเดิมกลับมาเพียงอย่างเดียว แต่ถ้าเกิดความผิดพลาดจะส่ง Function code ตัวเดิมพร้อมกับ most-significant bit ที่ถูกเซตให้เป็นลอจิก 1 กลับมา Master ก็จะรู้ความผิดพลาดที่เกิดขึ้น

ตารางที่ 2.2 Function code พื้นฐาน

Function Code	Action	Table Name
01 (01 hex)	Read	Discrete Output Coils
05 (05 hex)	Write single	Discrete Output Coil
15 (0F hex)	Write multiple	Discrete Output Coils
02 (02 hex)	Read	Discrete Input Contacts
04 (04 hex)	Read	Analog Input Registers
03 (03 hex)	Read	Analog Output Holding Registers
06 (06 hex)	Write single	Analog Output Holding Register
16 (10 hex)	Write multiple	Analog Output Holding Registers

ตำแหน่ง CRC นี้เป็นค่าที่คำนวณมาจากข้อมูลทุกไบต์ ไม่รวมบิต Start , Stop และ Parity Check โดยที่ตัว Slave ตัวที่ส่งข้อมูลออกมาจะสร้างรหัส CRC แล้วส่งตามท้ายไบต์ข้อมูลออกมา หลังจากนั้นเมื่อ Master ได้รับเฟรมข้อมูลและถอดข้อมูลออกจากเฟรมแล้วจะทำการคำนวณค่า CRC ตามสูตรเดียวกับ Slave เพื่อทำการเปรียบเทียบค่า CRC ทั้ง 2 ค่าว่าตรงกันหรือไม่ หากไม่ตรงกันแสดงว่าเกิดความผิดพลาดในการรับ-ส่งข้อมูลในโหมด RTU

การเก็บข้อมูลในรูปแบบ MODBUS ข้อมูลต่าง ๆ ของอุปกรณ์ที่เป็น Slave มี 2 ลักษณะที่มีคุณลักษณะต่างกัน คือ Discrete และ Register โดยลักษณะข้อมูลแบบ Discrete เป็นค่าแบบเปิดและปิด ส่วนอีกข้อมูลจะเก็บค่ารีจิสเตอร์ หรือค่าตัวเลข ซึ่ง Discrete และ Register ต่างก็มีลักษณะการเข้าถึงข้อมูลได้ 2 แบบ คือ อ่านได้อย่างเดียว (Read-only) และแบบอ่านและเขียนข้อมูลลงไปได้ (Read-write) โดยแต่ละรูปแบบจะมีจำนวนข้อมูล 9999 ตัว

Coil และ Contact ซึ่งเป็น Discrete จะใช้ข้อมูลแบบบิต (Bit) เพื่อเก็บข้อมูลของแต่ละตัว โดยแต่ละตัวจะถูกระบุตำแหน่งตั้งแต่ 0000 ถึง 270E ซึ่งเป็นเลขฐานสิบหก (แปลงเป็นฐานสิบคือ 0 ถึง 9998)

Register แต่ละตัวใช้พื้นที่ 16 bits = 2 bytes = 1 word จะสามารถใช้เก็บข้อมูลตัวเลขและมีตำแหน่งตั้งแต่ 0000 ถึง 270E เช่นกัน

Coil / Register Number จะเป็นเพียง Location name และไม่ปรากฏอยู่ในข้อมูลที่ถูกรับส่ง แต่ตำแหน่งข้อมูล (Data Address) จะถูกระบุอยู่ในข้อมูล ยกตัวอย่างเช่น Register ตัวแรกของ Analog Input Registers คือ 30001 จะมีตำแหน่งข้อมูลคือ 0000 เมื่อนำตัวเลข 30001 ลบด้วย 0000 จะได้ค่าที่เรียกว่า Offset คือ 30001 โดยแต่ละช่วงจะมีค่า Offset คือ 00001 10000 30001 และ 40001 ตามลำดับแสดงรายละเอียดตามตารางที่ 2.3

ตารางที่ 2.3 ตำแหน่งของ Coil / Register

Data Addresses	Coil/Register Numbers	Type	Table Name
0000 to 270E	1-9999	Read-Write	Discrete Output Coils
0000 to 270E	10001-19999	Read-Only	Discrete Input Contacts
0000 to 270E	30001-39999	Read-Only	Analog Input Registers
0000 to 270E	40001-49999	Read-Write	Analog Output Holding

จากตารางที่ 2.3 การเก็บข้อมูลของโปรโตคอลมอดบัสมี 4 ประเภท คือ Coil, Input Status, Input Register, Holding Register

1. Coil ถูกใช้งานสำหรับสั่งสถานะให้เปิดหรือปิดของ Discrete Output (DO) ในฟิลด์หรือเพื่อแก้ไขโหมดหรือสถานะของอุปกรณ์ Slave ข้อมูลของ Coil สามารถเขียนและอ่านได้ Coil สามารถใช้ตำแหน่งได้ตั้งแต่ 1-9999

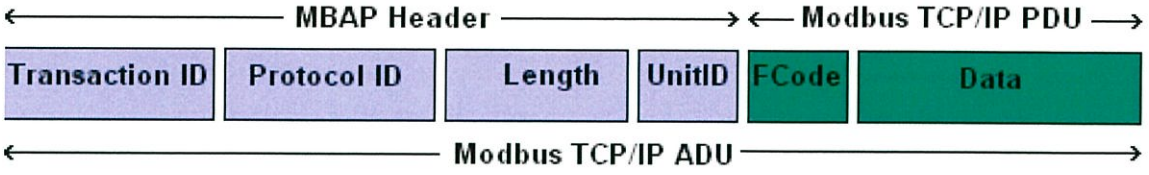
2. Input Status ถูกใช้งานเพื่อการอ่านค่า On/Off สถานะของ Discrete input (DI) จากฟิลด์หรือสถานะของอุปกรณ์ Slave โดย Input Status สามารถอ่านได้อย่างเดียว สามารถใช้ตำแหน่งตั้งแต่ 10001 – 19999

3. Input Register ถูกใช้งานเพื่อการอ่านค่า Analog Input (AI) จากฟิลด์หรือข้อมูลจากอุปกรณ์ Slave ข้อมูลของ Input Register มีความยาว 16 bit สามารถอ่านได้อย่างเดียว ใช้ตำแหน่งตั้งแต่ 30001 – 39999 ข้อมูลชนิด floating หรือ double floating สามารถเป็นตัวควบคุมเมื่อมีการกำหนด 2 ตำแหน่ง

4. Holding Register ถูกใช้งานกับค่าของ Analog Output (AO) ในฟิลด์หรือการตั้งค่าข้อมูลของ Slave ข้อมูลของ Holding Register มีความยาว16 bit สามารถเขียนและอ่านได้ใช้ตำแหน่งตั้งแต่ 40001 – 49999 ข้อมูลชนิด floating หรือ double floating สามารถเป็นตัวควบคุมเมื่อมีการกำหนด 2 ตำแหน่ง

2.4.2 โปรโตคอล MODBUS TCP/IP

Modbus TCP/IP คือ โปรโตคอล Modbus RTU ที่เชื่อมต่อผ่าน Internet Network โครงสร้าง Message ของ Modbus คือ Application Protocol ที่จะถูกส่งผ่านไปพร้อมกับ TCP/IP (Transmission Control Protocol และ Internet Protocol) ซึ่งเป็นตัวกลางที่ใช้ในการส่ง Message ของ Modbus TCP/IP ในทางปฏิบัติ Modbus TCP จะฝัง Modbus RTU data frame ร่วมไปกับ TCP frame โดยไม่ต้องใช้ Modbus checksum แต่จะใช้ Checksum ของ TCP แทน

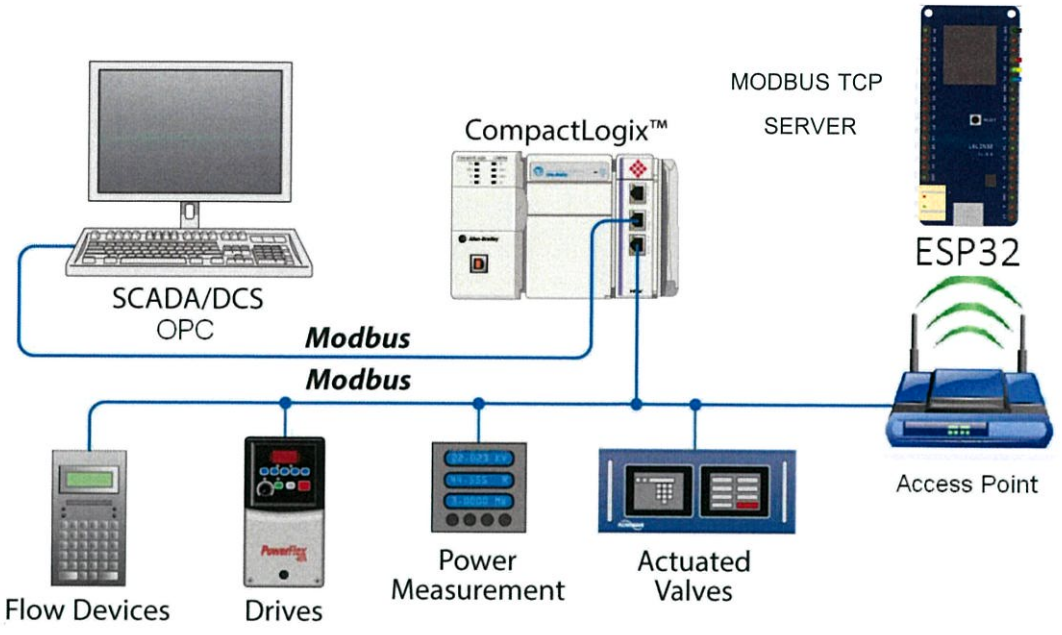


ภาพที่ 2.5 รูปแบบการส่งข้อมูลของ Modbus TCP/IP

ตารางที่ 2.4 แสดงความหมายของ Field Message

Name	Length	Function
Transaction identifier	2	For synchronization between server & client
Protocol identifier	2	Zero for Modbus/TCP
Length field	2	Number of remaining bytes in this frame
Unit identifier	1	Slave address (255 if not used)
Function code	1	Function codes as in other variants
Data bytes or command	N	Data as response or commands

จากภาพที่ 2.5 Modbus Application Protocol (MBAP) จะประกอบด้วยข้อมูล 7 bytes โดยมีรายละเอียดดังนี้ Transaction Identifier (2 bytes) ใช้จับคู่การแลกเปลี่ยนข้อมูลเมื่อมี Message หลาย ๆ ชุดถูกส่งออกมาด้วย TCP เดียวกันด้วย Client ตัวใดตัวหนึ่ง โดยไม่ต้องรอลำดับการ Response Protocol Identifier (2 bytes) มีค่าเป็น 0 เสมอ Length (2 bytes) เป็นการระบุจำนวน byte ที่รวมจำนวน byte ของ Unit identifier, Function code และ Data bytes or command (1 byte) เป็นการระบุ ID ของ server ที่อยู่ในระบบสื่อสารอาจตั้งเป็น 00 ถึง FF ก็ได้



ภาพที่ 2.6 การเชื่อมต่อ Modbus TCP/IP

2.5 อุปกรณ์และซอฟต์แวร์ที่สามารถอ่านค่าผ่านทางโปรโตคอลมอดบัส

อุปกรณ์และซอฟต์แวร์ที่สามารถอ่านค่าผ่านทางโปรโตคอลมอดบัสเป็นอุปกรณ์หรือซอฟต์แวร์ที่ใช้งานในอุตสาหกรรม เช่น โปรแกรม WONDERWARE InTouch โปรแกรม ModScan พีแอลซี โดยอุปกรณ์และซอฟต์แวร์ที่จะกล่าวถึงต่อไปนี้ เป็นอุปกรณ์และซอฟต์แวร์ที่สามารถอ่านค่าผ่านทางโปรโตคอลมอดบัส ที่ได้นำมาใช้ในการรับ-ส่งค่าสำหรับชุดโมดูลต้นแบบ

2.5.1 โปรแกรม WONDERWARE InTouch

WONDERWARE InTouch คือ ซอฟต์แวร์ที่เชื่อมต่อระหว่างมนุษย์กับเครื่องจักร (Human Machine Interface : HMI) โดยการแสดงผลและควบคุมด้วยกราฟิกเพื่อให้ผู้สร้างแอพพลิเคชันเพิ่มประสิทธิภาพการควบคุม การแสดงผล การจัดเก็บข้อมูล เอกสารรายงาน การวิเคราะห์ข้อมูล

โปรแกรม WONDERWARE InTouch สามารถเชื่อมต่อกับโปรโตคอลมอดบัสผ่านทาง OPC (OLE for Process Control) ที่มีชื่อว่า SMC (System Management Console)

ภาพที่ 2.7 WONDERWARE InTouch Software

ในการสร้างหน้าจอแสดงผลบน WONDERWARE จำเป็นต้องมี Tag Name เพื่อทำให้กราฟิกสามารถแสดงผลได้ตามต้องการจาก INPUT/OUTPUT Tag Name ที่แสดงตามตารางด้านล่าง

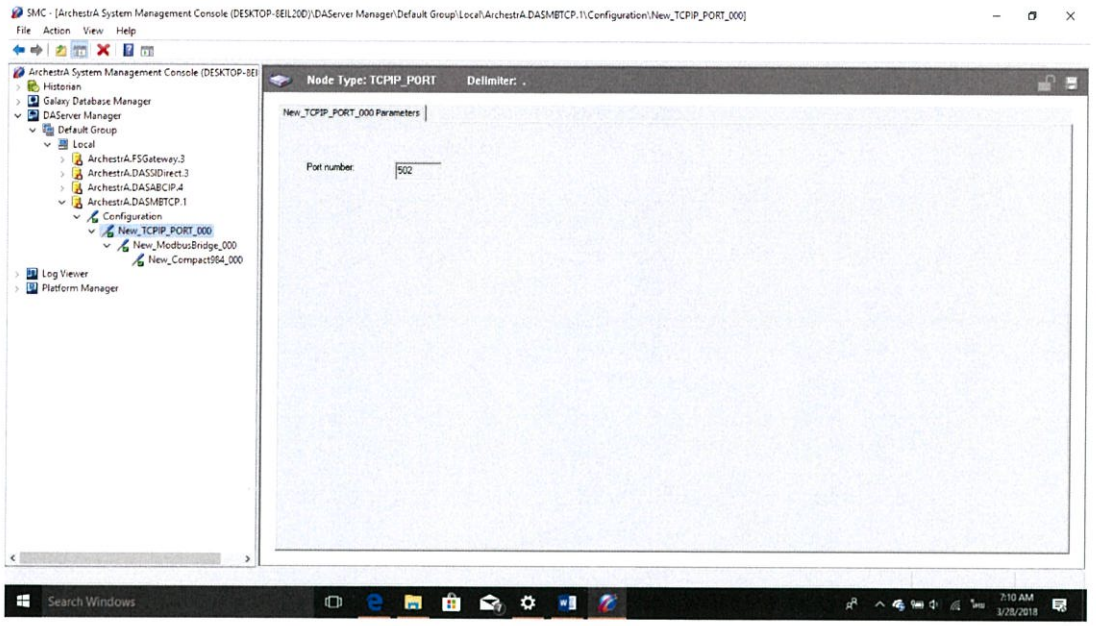
ตารางที่ 2.5 INPUT/OUTPUT Tag Name

INPUT/OUTPUT Tag Name				
No.	Tag Name	Data Type	Access Name	Address
1	DI1	INPUT/OUTPUT Discrete	Module (DASMBTCP)	10001
2	DI2	INPUT/OUTPUT Discrete		10002
3	DI3	INPUT/OUTPUT Discrete		10003
4	DI4	INPUT/OUTPUT Discrete		10004
5	DI5	INPUT/OUTPUT Discrete		10005
6	DI6	INPUT/OUTPUT Discrete		10006
7	DI7	INPUT/OUTPUT Discrete		10007
8	DI8	INPUT/OUTPUT Discrete		10008
9	AI1	INPUT/OUTPUT Real		30001 f
10	AI2	INPUT/OUTPUT Real		30002 f
11	DO1	INPUT/OUTPUT Discrete		00001
12	DO2	INPUT/OUTPUT Discrete		00002
13	DO3	INPUT/OUTPUT Discrete		00003
14	DO4	INPUT/OUTPUT Discrete		00004
15	DO5	INPUT/OUTPUT Discrete		00005
16	DO6	INPUT/OUTPUT Discrete		00006
17	DO7	INPUT/OUTPUT Discrete		00007
18	DO8	INPUT/OUTPUT Discrete		00008
19	AO1	INPUT/OUTPUT Real		40001 f
20	AO2	INPUT/OUTPUT Real		40001 f

ในการเชื่อมต่อกับชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพีเพื่อนอ่านค่าขึ้นมาใช้งาน สามารถทำได้โดยใช้ซอฟต์แวร์ OPC ที่มีชื่อว่า SMC ซึ่งเชื่อมต่อกันผ่าน Modbus TCP/IP โดยใช้ Access Name (DASMBTCP) โดยมีขั้นตอนดังนี้

A) สร้าง TCPIP_PORT

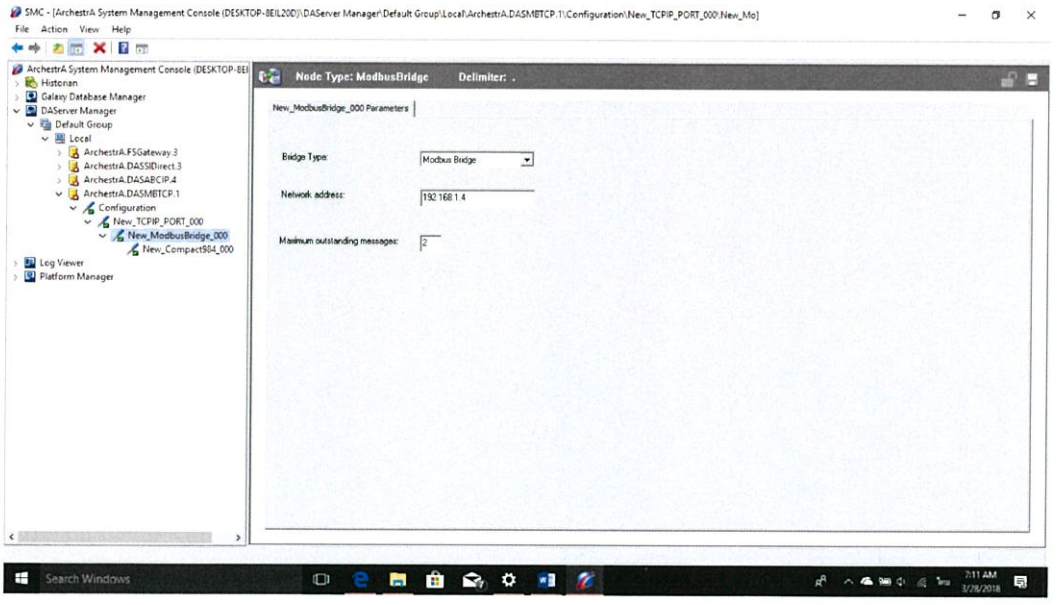
เป็นการกำหนดช่องทางการเชื่อมต่อพอร์ตของเครือข่าย Ethernet



ภาพที่ 2.8 การสร้าง TCPIP_PORT

B) ตั้งค่า ModbusBridge

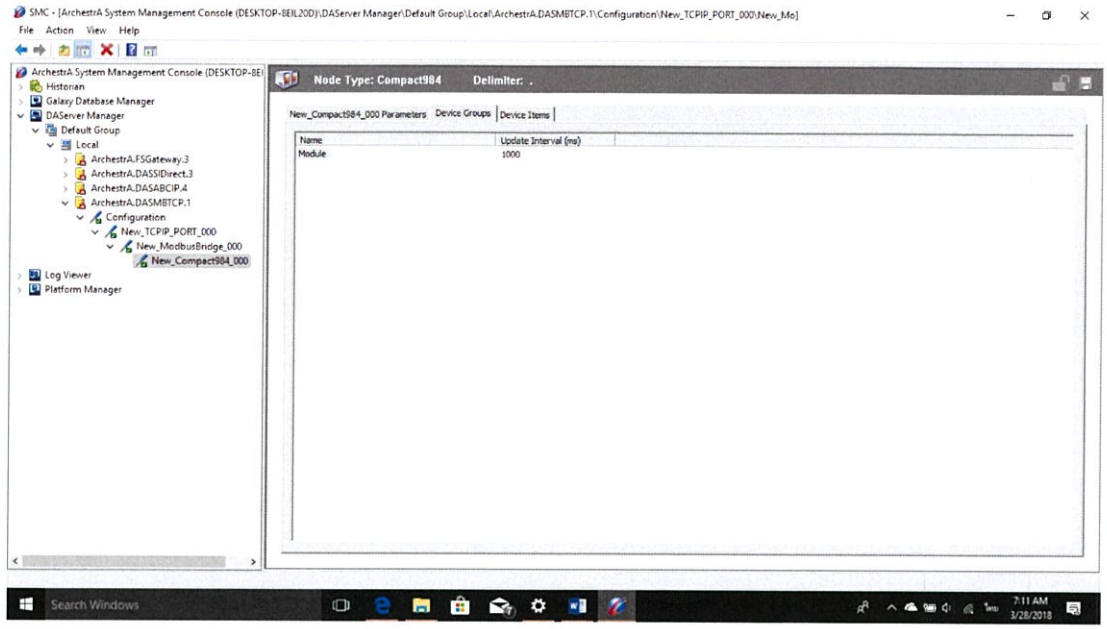
เป็นการกำหนดการเชื่อมต่อ Modbus TCP/IP โดยจะกำหนด IP ให้ตรงกับชุดชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพีเพื่อรับค่าอินพุทและเอาท์พุทที่ถูกส่งมา



ภาพที่ 2.9 การตั้งค่า ModbusBridge

C) สร้าง Device Group

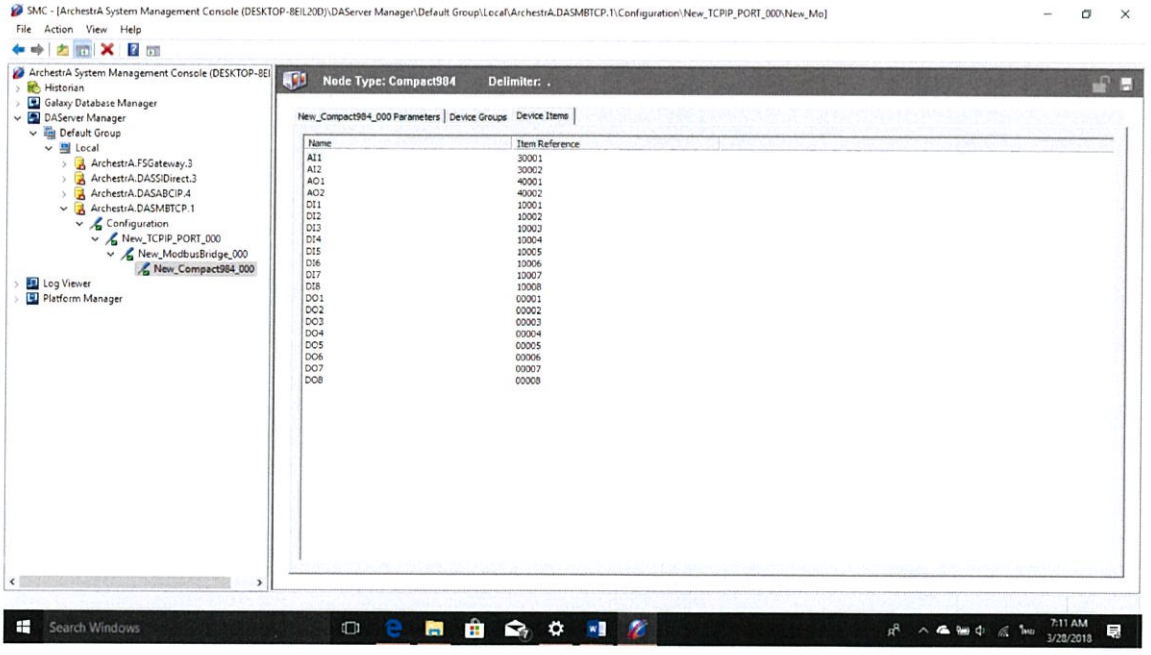
เป็นการกำหนดชื่อกลุ่มของ Device และความเร็วในการส่งข้อมูล



ภาพที่ 2.10 การสร้าง Device Group

D) สร้าง Device Item

เป็นการกำหนด Address ที่เราต้องการจะรับค่าที่แสดงบน WONDERWARE ซึ่งจะใช้ Address ของ Modbus Register เป็นตัวกำหนดค่าของ Tag Name

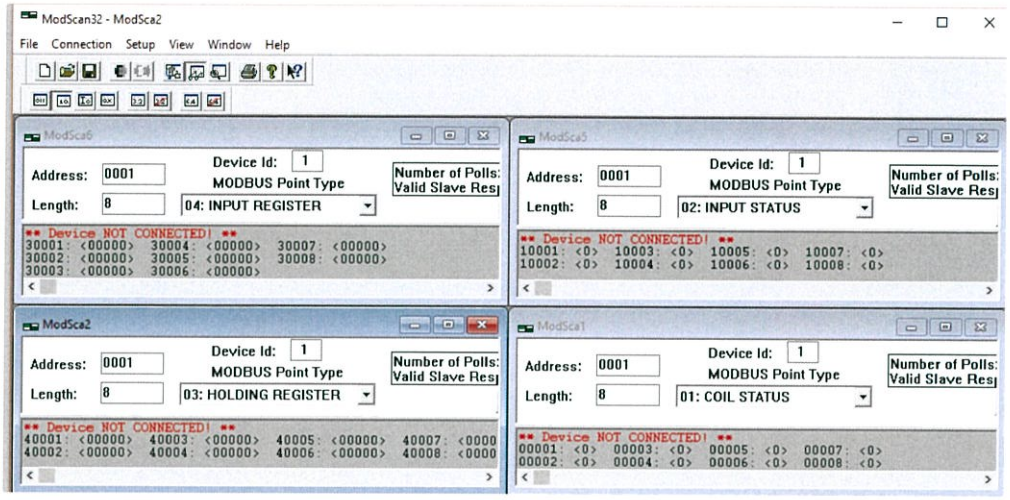


ภาพที่ 2.11 การสร้าง Device Item

2.5.2 โปรแกรม MODSCAN

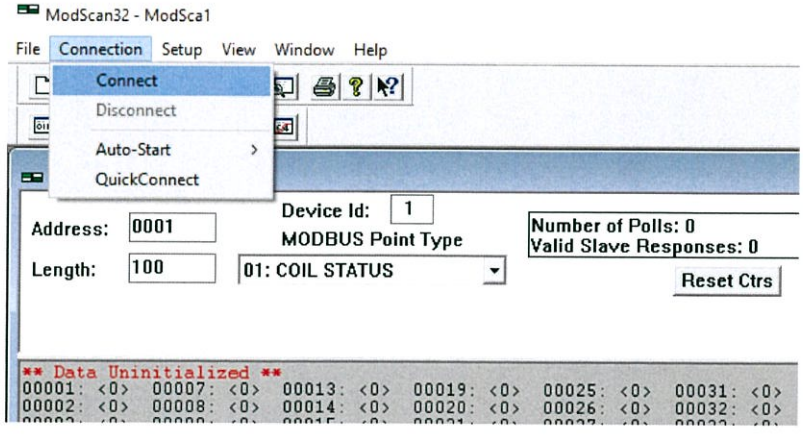
โปรแกรม MODSCAN เป็นเป็นแกรมที่ใช้สื่อสารกับอุปกรณ์ที่ใช้บน Protocol MODBUS โดยสามารถใช้ได้ทั้ง MODBUS RTU MODBUS ASCII และ MODBUS TCP/IP โปรแกรม MODSCAN สามารถอ่าน-เขียนรีจิสเตอร์ข้อมูลและยังสามารถแสดงค่าเฟรมการรับ-ส่งข้อมูล

การใช้โปรแกรม MODSCAN มักจะใช้เพื่อทดสอบอุปกรณ์ที่ใช้ MODBUS Protocol และสามารถดึงค่าจากเครื่องมือวัด หรือ พีแอลซี ขึ้นมาดูค่ารีจิสเตอร์ข้อมูลต่าง ๆ โดยผ่านทางโปรแกรม MODSCAN ได้



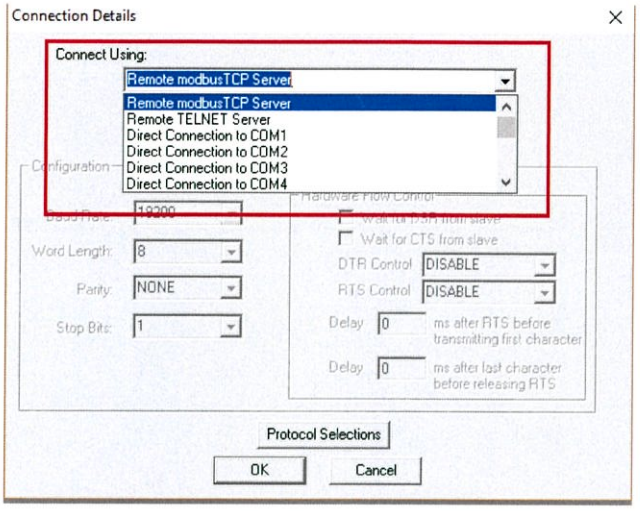
ภาพที่ 2.12 หน้าต่างโปรแกรม MODSCAN

การเชื่อมต่อโปรแกรมสื่อสาร ModScan32 กับ RTU ที่ใช้โปรโตคอลมอดบัสทีซีพีไอพี สามารถเชื่อมต่อได้โดยการเปิดโปรแกรม ModScan32 แล้วคลิกที่ Connection เลือก Connect ดังภาพที่ 2.13

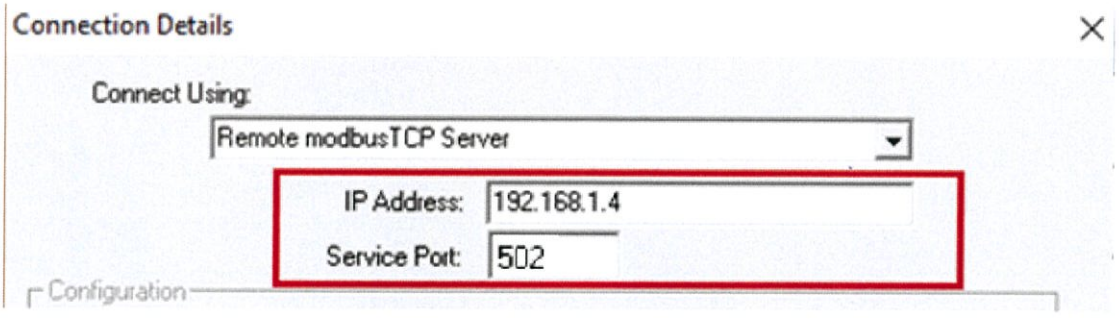


ภาพที่ 2.13 การ Connect ผ่านโปรแกรม ModScan32

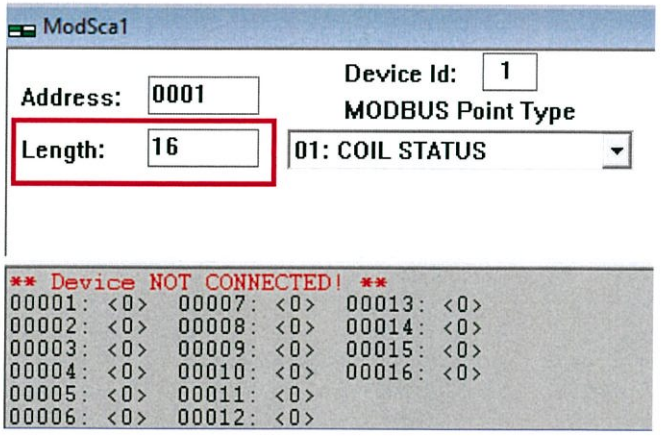
บนหน้าต่าง Connect Using เลือก Remote Modbus TCP Server ดังภาพที่ 2.14 และระบุ IP Address ที่ต้องการตรวจสอบ กำหนดให้ IP Address ของโมดูลนี้ = 192.168.1.4 จากนั้นใส่ Port ที่ต้องการ ดังภาพที่ 2.15 สามารถเลือกจำนวน Length ที่ต้องการตรวจสอบ ดังภาพที่ 2.16



ภาพที่ 2.14 การเลือกการใช้งานการเชื่อมต่อที่ต้องการกับรูปแบบการสื่อสาร



ภาพที่ 2.15 การกำหนด IP Address และ Service Port ที่ต้องการตรวจสอบ (502)



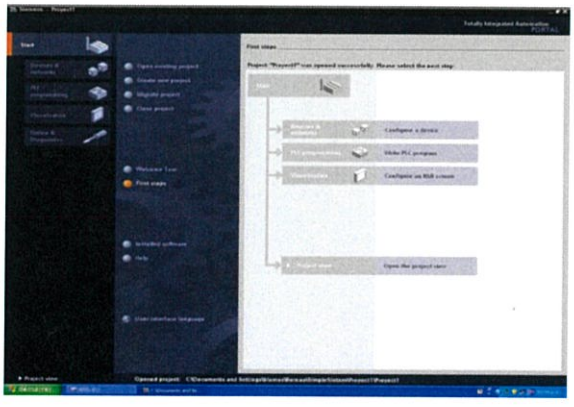
ภาพที่ 2.16 จำนวนแอดเดรสที่ต้องการแสดงผล

2.5.3 พีแอลซี ยี่ห้อ Siemens รุ่น S7-1200

S7-1200 เป็นพีแอลซีรุ่นเล็กที่เหมาะกับผู้ที่จะเริ่มต้นเรียนรู้การใช้งานพีแอลซีเบื้องต้นและพัฒนาต่อยอดไปถึงรุ่นใหญ่ได้ต่อไป เนื่องจากทาง Siemens ได้ใช้ software ใน platform เดียวกัน แต่แยกการตั้งค่าของพีแอลซีแต่ละรุ่นเอาไว้

ภาพที่ 2.17 ตัวอย่าง พีแอลซี รุ่น S7-1200

การใช้งาน S7-1200 จะต้องใช้โปรแกรม STEP7 Basic (TIA Portal) แต่หากจะใช้งาน S7-1500 นั้น จะใช้โปรแกรม STEP7 Professional (TIA Portal) ซึ่งทั้ง 2 แบบ ก็ยังคงใช้คำสั่งพื้นฐานเหมือนกันทั้งคู่ เพียงแต่เพิ่มความสามารถของการใช้งานอื่น ๆ เช่น งาน motion หรือ technology object ต่าง ๆ



ภาพที่ 2.18 หน้าต่างโปรแกรม TIA Portal

การเขียนโปรแกรม PLC โดยใช้ซอฟต์แวร์ TIA Portal V13 ทางผู้จัดทำได้ออกแบบไว้ให้รับค่าอินพุท และส่งค่าเอาท์พุทออกมาแบบ 1 ต่อ 1 ทั้งสัญญาณดิจิตอลและสัญญาณอะนาล็อก โดยการเขียนโปรแกรมประกอบด้วย

1. INPUT/OUTPUT List ของโปรแกรม

เนื่องจากจำเป็นที่จะต้องมีข้อมูลที่แน่ชัด จึงจำเป็นต้องกำหนด INPUT/OUTPUT List ซึ่งเป็นข้อมูลอ้างอิงสำหรับการเขียนโปรแกรม เพื่อนำสัญญาณจาก PLC ไปใช้

เชื่อมต่อกับชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพี มีรายละเอียดดังตารางที่ 2.6

ตารางที่ 2.6 INPUT/OUTPUT list บนชุดต้นแบบ

INPUT/OUTPUT List								
No.	Tag Name	Description	Data Type	PLC Address	DI	DO	AI	AO
1	DI1	ดิจิตอลอินพุท ช่องที่ 1	Boolean	%M118.0	1			
2	DI2	ดิจิตอลอินพุท ช่องที่ 2	Boolean	%M118.1	1			
3	DI3	ดิจิตอลอินพุท ช่องที่ 3	Boolean	%M118.2	1			
4	DI4	ดิจิตอลอินพุท ช่องที่ 4	Boolean	%M118.3	1			
5	DI5	ดิจิตอลอินพุท ช่องที่ 5	Boolean	%M118.4	1			
6	DI6	ดิจิตอลอินพุท ช่องที่ 6	Boolean	%M118.5	1			
7	DI7	ดิจิตอลอินพุท ช่องที่ 7	Boolean	%M118.6	1			
8	DI8	ดิจิตอลอินพุท ช่องที่ 8	Boolean	%M118.7	1			
9	AI1	อะนาล็อกอินพุท ช่องที่ 1	Int	%MD120			1	
10	AI2	อะนาล็อกอินพุท ช่องที่ 2	Int	%MD122			1	
11	DO1	ดิจิตอลเอาท์พุท ช่องที่ 1	Boolean	%M119.0		1		
12	DO2	ดิจิตอลเอาท์พุท ช่องที่ 2	Boolean	%M119.1		1		
13	DO3	ดิจิตอลเอาท์พุท ช่องที่ 3	Boolean	%M119.2		1		
14	DO4	ดิจิตอลเอาท์พุท ช่องที่ 4	Boolean	%M119.3		1		
15	DO5	ดิจิตอลเอาท์พุท ช่องที่ 5	Boolean	%M119.4		1		
16	DO6	ดิจิตอลเอาท์พุท ช่องที่ 6	Boolean	%M119.5		1		
17	DO7	ดิจิตอลเอาท์พุท ช่องที่ 7	Boolean	%M119.6		1		
18	DO8	ดิจิตอลเอาท์พุท ช่องที่ 8	Boolean	%M119.7		1		
19	AO1	อะนาล็อกเอาท์พุท ช่องที่ 1	Int	%MD124				1
20	AO2	อะนาล็อกเอาท์พุท ช่องที่ 2	Int	%MD126				1

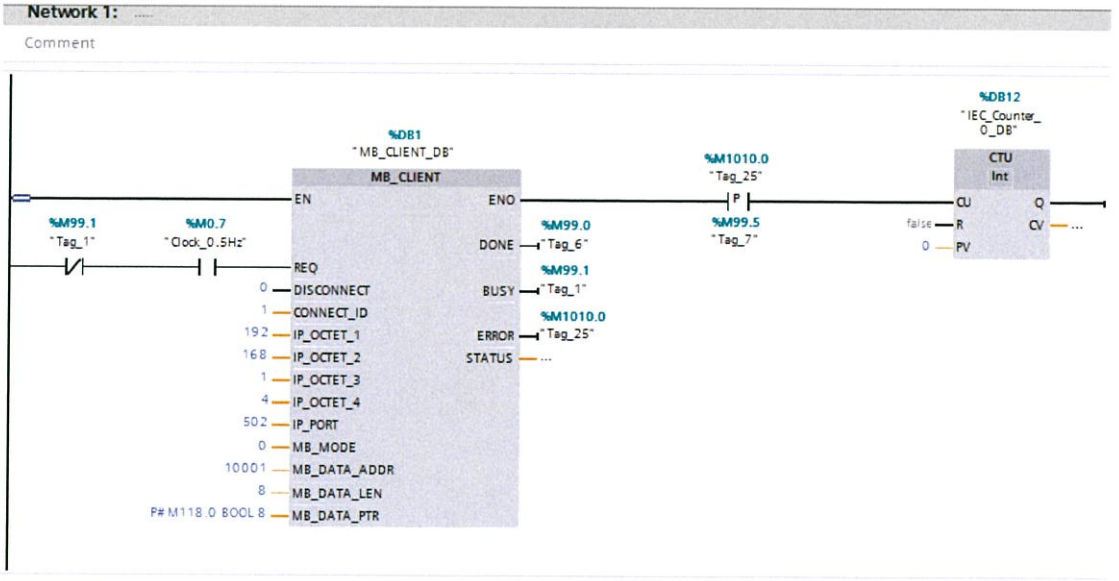
DI(1)	Default tag table	Bool	%M118.0		☑	☑
DI(2)	Default tag table	Bool	%M118.1		☑	☑
DI(3)	Default tag table	Bool	%M118.2		☑	☑
DI(4)	Default tag table	Bool	%M118.3		☑	☑
DI(5)	Default tag table	Bool	%M118.4		☑	☑
DI(6)	Default tag table	Bool	%M118.5		☑	☑
DI(7)	Default tag table	Bool	%M118.6		☑	☑
DO(1)	Default tag table	Bool	%M119.0		☑	☑
DO(2)	Default tag table	Bool	%M119.1		☑	☑
DO(3)	Default tag table	Bool	%M119.2		☑	☑
DO(4)	Default tag table	Bool	%M119.3		☑	☑
DO(5)	Default tag table	Bool	%M119.4		☑	☑
DO(6)	Default tag table	Bool	%M119.5		☑	☑
DO(7)	Default tag table	Bool	%M119.6		☑	☑
AI(1)	Default tag table	Real	%MD120		☑	☑
AI(2)	Default tag table	Real	%MD122		☑	☑
AO(1)	Default tag table	Real	%MD124		☑	☑
AO(2)	Default tag table	Real	%MD126		☑	☑

ภาพที่ 2.19 Tag name ทั้งหมดในโปรแกรม TIA Portal V13

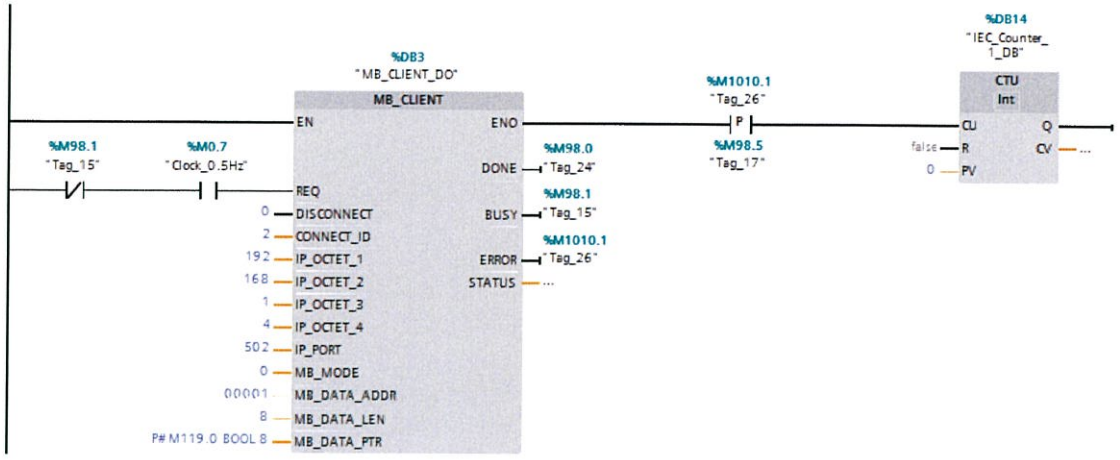
2. โปรแกรม TIA Portal V.13

การเขียนและออกแบบโปรแกรมของ TIA Portal V.13 จะเขียนในรูปแบบของ Ladder Diagram โดยการเรียกใช้ Block ที่มีให้ในตัวโปรแกรม TIA Portal V.13 โดยจะมีรายละเอียดของโปรแกรมดังต่อไปนี้

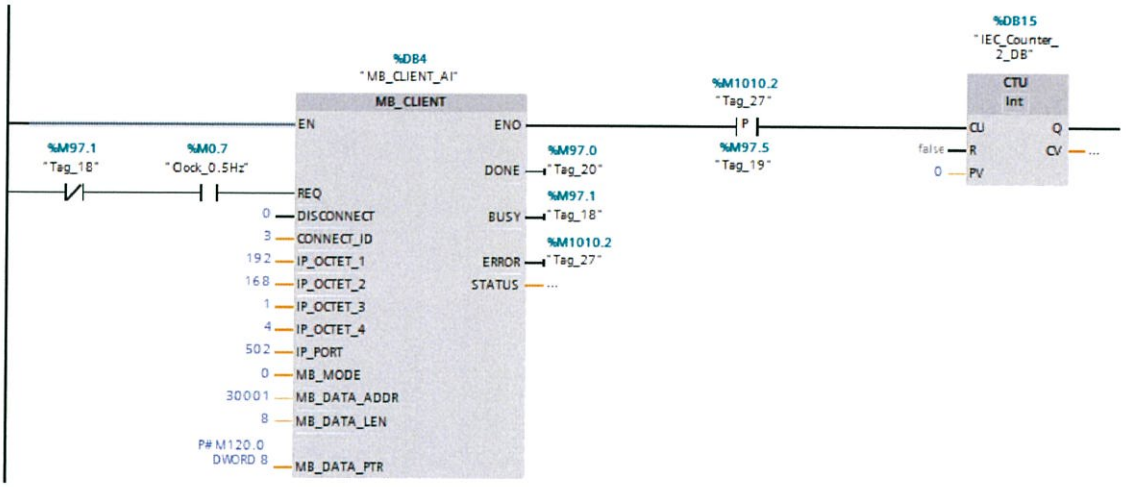
Main Block [OB1] จะทำหน้าที่หลักในการทำงานของฟังก์ชันของโปรแกรมทั้งหมด โดยใน OB1 จะประกอบด้วย Block ใช้งานที่ชื่อว่า MB_Client สำหรับการรับส่งค่าที่เชื่อมเข้ามาของดิจิตอลอินพุทเอาท์พุท และอะนาล็อกเอาท์พุทผ่านมอดบัสทีซีพีไอพี



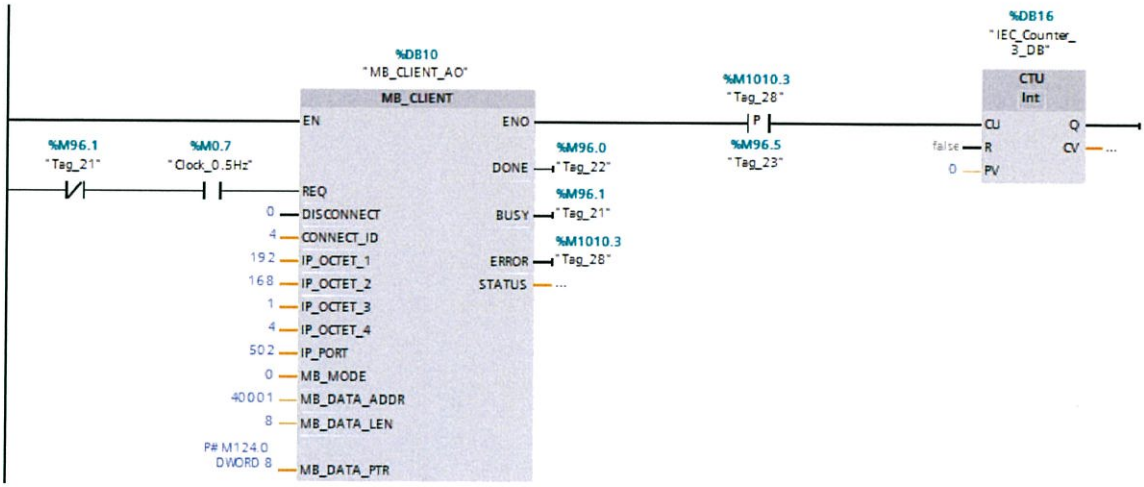
ภาพที่ 2.20 Ladder DI ของ Main Block [OB1]



ภาพที่ 2.21 Ladder DO ของ Main Block [OB1]



ภาพที่ 2.22 Ladder AI ของ Main Block [OB1]



ภาพที่ 2.23 Ladder AO ของ Main Block [OB1]

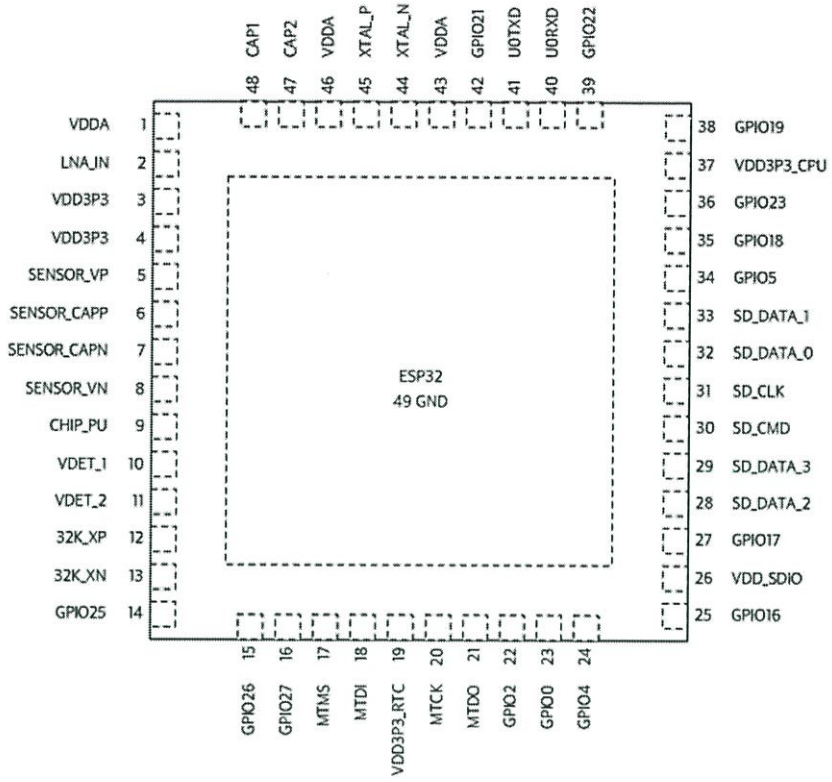
2.6 โปรโตคอลเฮททีทีพี (HyperText Transfer Protocol: HTTP)

HTTP คือ โปรโตคอลในระดับชั้นโปรแกรมประยุกต์เพื่อการแจกจ่ายและการทำงานร่วมกันกับสารสนเทศของสื่อผสม ใช้สำหรับการรับทรัพยากรที่เชื่อมโยงกับภายนอก ซึ่งนำไปสู่การจัดตั้งเวิลด์ไวด์เว็บ การพัฒนา HTTP เป็นการทำงานร่วมกันของเวิลด์ไวด์เว็บคอนซอร์เทียม (W3C) และคณะทำงานเฉพาะกิจด้านวิศวกรรมอินเทอร์เน็ต (IETF) ซึ่งมีผลงานเด่นในการเผยแพร่เอกสารขอความเห็น (RFC) หลายชุด เอกสารที่สำคัญที่สุดคือ RFC 2616 ได้กำหนด HTTP/1.1 ซึ่งเป็นรุ่นที่ใช้กันอย่างกว้างขวางในปัจจุบัน

HTTP เป็นมาตรฐานในการร้องขอและการตอบรับระหว่างเครื่องลูกข่ายกับเครื่องแม่ข่าย ซึ่งเครื่องลูกข่ายคือผู้ใช้ปลายทาง (end-user) และเครื่องแม่ข่ายคือเว็บไซต์ เครื่องลูกข่ายจะสร้างการร้องขอ HTTP ผ่านทางเว็บเบราว์เซอร์ ที่จัดว่าเป็น ตัวแทนผู้ใช้ (user agent) ส่วนเครื่องแม่ข่ายที่ตอบรับ ซึ่งเก็บบันทึกหรือสร้าง ทรัพยากร (resource) อย่างเช่นไฟล์ HTML หรือรูปภาพ จะเรียกว่าเครื่องให้บริการต้นทาง (origin server) ในระหว่างตัวแทนผู้ใช้กับเครื่องให้บริการต้นทางอาจมีสื่อกลางหลายชนิด อาทิพร็อกซี เกตเวย์ และทุนเนล HTTP ไม่ได้จำกัดว่าจะต้องใช้ชุดเกณฑ์วิธีอินเทอร์เน็ต (TCP/IP) เท่านั้น แม้ว่าจะเป็นการใช้งานที่นิยมมากที่สุดบนอินเทอร์เน็ตก็ตาม โดยแท้จริงแล้ว HTTP สามารถนำไปใช้ได้บนโปรโตคอลอินเทอร์เน็ตอื่น ๆ หรือบนเครือข่ายอื่นก็ได้ HTTP คาดหวังเพียงแค่การสื่อสารที่เชื่อถือได้นั่นคือโปรโตคอลที่มีการรับรองเช่นนั้นก็สามารถใช้งานได้ตามปกติ เครื่องลูกข่าย HTTP จะเป็นผู้เริ่มสร้างการร้องขอก่อน โดยเปิดการเชื่อมต่อด้วยเกณฑ์วิธีควบคุมการขนส่งข้อมูล (TCP) ไปยังพอร์ตเฉพาะของเครื่องแม่ข่าย (พอร์ต 80 เป็นค่าเริ่มต้น) เครื่องแม่ข่าย HTTP เปิดรอรับอยู่ที่พอร์ตนั้น จะเปิดรอให้เครื่องลูกข่ายส่งข้อความร้องขอเข้ามา เมื่อได้รับการร้องขอแล้ว เครื่องแม่ข่ายจะตอบรับด้วยข้อความสถานะอันหนึ่ง ตัวอย่างเช่น "HTTP/1.1 200 OK" ตามด้วยเนื้อหาของมันเองส่งไปด้วย เนื้อหานั้นอาจเป็นแฟ้มข้อมูลที่ร้องขอ ข้อความแสดงข้อผิดพลาด หรือข้อมูลอย่างอื่นเป็นต้น

2.7 ไมโครคอนโทรลเลอร์ ESP32

2.7.1 ไมโครคอนโทรลเลอร์รุ่น ESP32



ภาพที่ 2.24 ขา Pinout ของไอซี ESP32

ESP32 เป็นชื่อของไอซีไมโครคอนโทรลเลอร์ที่รองรับการเชื่อมต่อ WiFi และ Bluetooth 4.2 BLE ในตัว ผลิตโดยบริษัท Espressif โดย ESP32 มีคุณสมบัติโดยละเอียด ดังนี้

- ซีพียูใช้สถาปัตยกรรม Tensilica LX6 แบบ 2 แกนสมอง สัญญาณนาฬิกา 240 MHz
- มีแรมในตัว 512 KB
- รองรับการเชื่อมต่อรอมภายนอกสูงสุด 16 MB
- มาพร้อมกับ WiFi มาตรฐาน 802.11 b/g/n รองรับการใช้งานทั้งในโหมด Station AP และ WiFi direct
- มีบลูทูธในตัว รองรับการใช้งานในโหมด 2.0 และโหมด 4.0 BLE
- ใช้แรงดันไฟฟ้าในการทำงาน 2.6 V ถึง 3 V
- ทำงานได้ที่อุณหภูมิ -40°C ถึง 125°C

นอกจากนี้ ESP32 ยังมีเซ็นเซอร์ต่าง ๆ มาในตัวด้วย ดังนี้

- วงจรกรองสัญญาณรบกวนในวงจรขยายสัญญาณ
- เซ็นเซอร์แม่เหล็ก

- เซ็นเซอร์สัมผัส (Capacitive touch) รองรับ 10 ช่อง
- รองรับการเชื่อมต่อคลิสตอล 32.768 kHz สำหรับใช้กับส่วนวงจรนับเวลาโดยเฉพาะ

ขาใช้งานต่าง ๆ ของ ESP32 รองรับการเชื่อมต่อบัสต่าง ๆ ดังนี้

- มี GPIO จำนวน 32 ช่อง
- รองรับ UART จำนวน 3 ช่อง
- รองรับ SPI จำนวน 3 ช่อง
- รองรับ I2C จำนวน 2 ช่อง
- รองรับ ADC จำนวน 12 ช่อง
- รองรับ DAC จำนวน 2 ช่อง
- รองรับ I2S จำนวน 2 ช่อง
- รองรับ PWM / Timer ทุกช่อง
- รองรับการเชื่อมต่อกับ SD-Card

นอกจากนี้ ESP32 ยังรองรับฟังก์ชันเกี่ยวกับความปลอดภัยต่าง ๆ ดังนี้

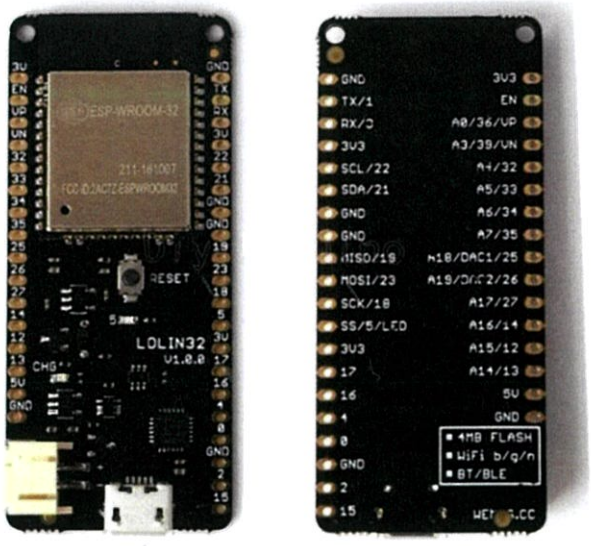
- รองรับการเข้ารหัส WiFi แบบ WEP และ WPA / WPA2 PSK / Enterprise
- มีวงจรเข้ารหัส AES / SHA2 / Elliptical Curve Cryptography / RSA-4096 ในตัว

ในด้านประสิทธิภาพการใช้งาน ตัว ESP32 สามารถทำงานได้โดย

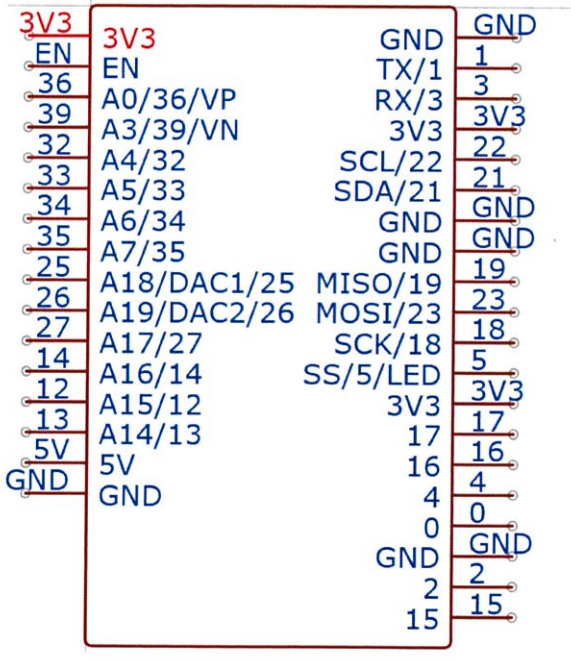
- รับ – ส่ง ข้อมูลได้ความเร็วสูงสุดที่ 150 Mbps เมื่อเชื่อมต่อแบบ 11n HT40 ได้ความเร็วสูงสุด 72 Mbps เมื่อเชื่อมต่อแบบ 11n HT20 ได้ความเร็วสูงสุดที่ 54 Mbps เมื่อเชื่อมต่อแบบ 11g และได้ความเร็วสูงสุดที่ 11 Mbps เมื่อเชื่อมต่อแบบ 11b
- เมื่อใช้การเชื่อมต่อผ่านโปรโตคอล UDP จะสามารถรับ – ส่งข้อมูลได้ที่ความเร็ว 135 Mbps
- ในโหมด Sleep ใช้กระแสไฟฟ้าเพียง 2.5 uA

2.7.2 บอร์ดควบคุมรุ่น WeMos LOLLIN ESP32

บอร์ดควบคุมรุ่น WeMos LOLIN ESP32 เป็นบอร์ดจากผู้ผลิต WeMos ที่มาพร้อมกับ WiFi IEEE 802.11 b/g/n และยังมี Bluetooth 4.2 เพิ่มเข้ามา นอกจากเรื่องราคาที่ถูก ตัวบอร์ดยังรองรับการชาร์จ ใช้พลังงานไฟฟ้าจากแบตเตอรี่ลิโพผ่านคอนเนคเตอร์แบบ JST โดยบอร์ดควบคุมรุ่น WeMos LOLLIN ESP32 มีความแตกต่างจากทุกบอร์ดตรงที่ใช้ชิปไอซีแปลง USB เป็น UART เบอร์ CP2104 จากบริษัท Silicon Labs ที่มีขนาดเล็กกว่าชิปไอซี CP2102 และตามลักษณะเด่นของ WeMos คือไม่มีปุ่มควบคุม GPIO0 ทำให้ไม่สามารถกดเข้าโหมดอัพโหลดโปรแกรมด้วยตนเองได้ บอร์ด WEMOS LOLIN ESP32 ใช้โมดูล ESP-WROOM-32 มีรอม 4MB (32Mbit) ใช้ไอซีเรกกูเลเตอร์แบบ LDO หลอด LED แสดงสถานะการชาร์จแบตเตอรี่ และหลอด LED เชื่อมต่อกับ GPIO5 สวิตช์กดติดปล่อยดับเชื่อมต่อกับขา CHP_PU ของโมดูล ESP-WROOM-32 ใช้สำหรับรีเซ็ตบอร์ด ชิปไอซีจัดการการชาร์จแบตเตอรี่ถูกตั้งไว้ให้จ่ายกระแสชาร์จที่ 500mA ใช้พลังงานไฟฟ้าและสื่อสารผ่านพอร์ต MicroUSB มีขาต่อใช้งานทั้งหมด 40 ขา น้ำหนัก 5.8 กรัม เมื่อนำไปเสียบลงโพรโต้บอร์ดจะเหลือช่องให้ใช้งานด้านละ 1 ช่อง



ภาพที่ 2.25 บอร์ดควบคุมรุ่น WeMos LOLIN ESP32



ภาพที่ 2.26 ขาใช้งานของบอร์ดพัฒนา WeMos LOLIN ESP32

2.8 โปรแกรม Arduino

2.8.1 โปรแกรม Arduino IDE

ภาพที่ 2.27 โปรแกรม Arduino IDE

ซอฟต์แวร์ที่ใช้ในการพัฒนางานสำหรับบอร์ด Arduino นั่นคือโปรแกรมที่เรียกว่า Arduino IDE ใช้ในการเขียนโปรแกรมและคอมไพล์ลงบอร์ด IDE ย่อมาจาก (Integrated Development Environment) คือ ส่วนเสริมของระบบการพัฒนาหรือตัวช่วยต่าง ๆ ที่จะคอยช่วยเหลือผู้พัฒนา หรือช่วยเหลือคนที่พัฒนา Application เพื่อเสริมให้เกิดความรวดเร็ว ถูกต้องแม่นยำ ตรวจสอบระบบที่จัดทำได้ ทำให้การพัฒนางานต่าง ๆ เร็วมากขึ้น

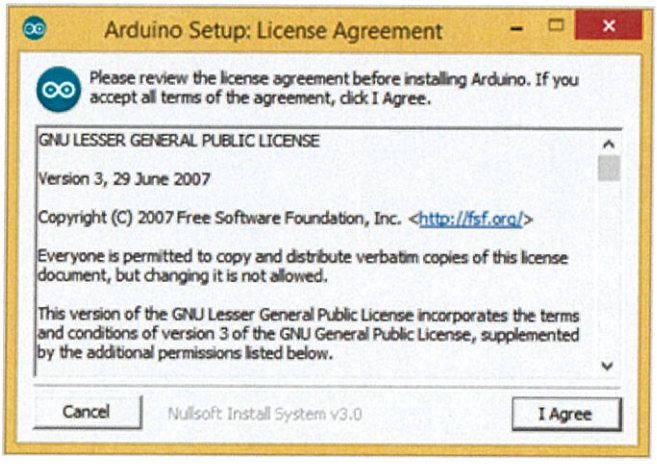
Arduino แต่เดิมเป็นแพลตฟอร์มที่ใช้ในการพัฒนาเฟิร์มแวร์ให้กับบอร์ด Arduino เท่านั้น แต่ภายหลังกลุ่มผู้พัฒนาโปรแกรม Arduino IDE ได้เริ่มรองรับการติดตั้งชุดพัฒนาเฟิร์มแวร์ให้กับบอร์ดอื่น ๆ ด้วย ทำให้บอร์ดอื่น ๆ ที่รองรับการเขียนโปรแกรมด้วยภาษา C/C++ สามารถเข้ามาใช้โปรแกรม Arduino IDE ในการพัฒนาได้ นอกจากข้อดีของโปรแกรม Arduino IDE แล้ว ชุดไลบารี่ต่าง ๆ ที่ทำมารองรับกับแพลตฟอร์ม Arduino ก็จะสามารถนำมาใช้งานกับบอร์ด อื่น ๆ ได้ด้วย นอกจากนี้แพลตฟอร์ม Arduino ยังมีชุมชนขนาดใหญ่ ทั้งในประเทศไทยเอง และในต่างประเทศทำให้สามารถค้นหาข้อมูลได้ง่าย

2.8.2 การติดตั้งโปรแกรม Arduino IDE

การติดตั้ง Arduino IDE เข้าไปดาวน์โหลดที่เว็บไซต์ https://www.arduino.cc/ เลือกเมนู SOFTWARE เลื่อนไปที่หัวข้อ Download the Arduino IDE จากนั้นเลือกดาวน์โหลดตาม OS ที่ต้องการ ดังภาพที่ 2.28

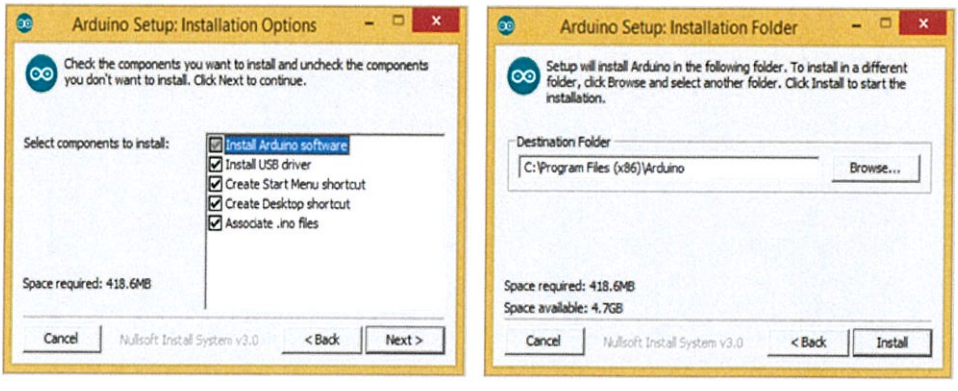
ภาพที่ 2.28 การเลือกติดตั้งร่วมกับ OS ของคอมพิวเตอร์

เมื่อดาวน์โหลดเสร็จแล้ว ให้เปิดไฟล์ติดตั้งโปรแกรมขึ้นมา จะพบหน้ารายละเอียดข้อตกลงการใช้งานโปรแกรม ให้กด I Agree ดังภาพที่ 2.29



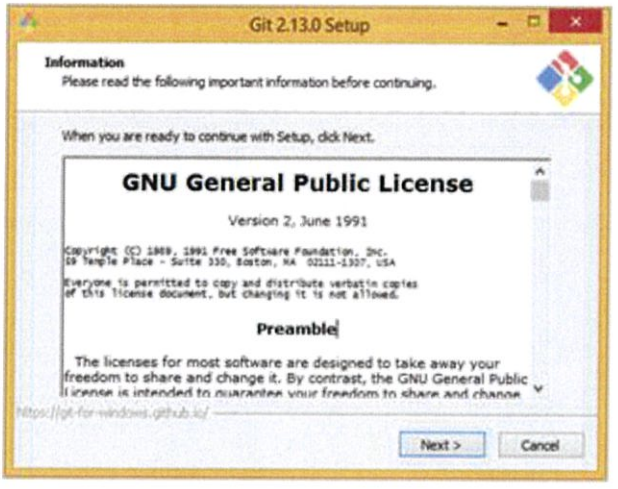
ภาพที่ 2.29 การยอมรับข้อตกลงของโปรแกรม Arduino

หน้านี้จะให้เลือกว่าจะติดตั้งอะไรบ้าง ให้กดที่ NEXT เลือกโฟลเดอร์ที่จะเก็บไฟล์โปรแกรม ให้กด Install ดังภาพที่ 2.30



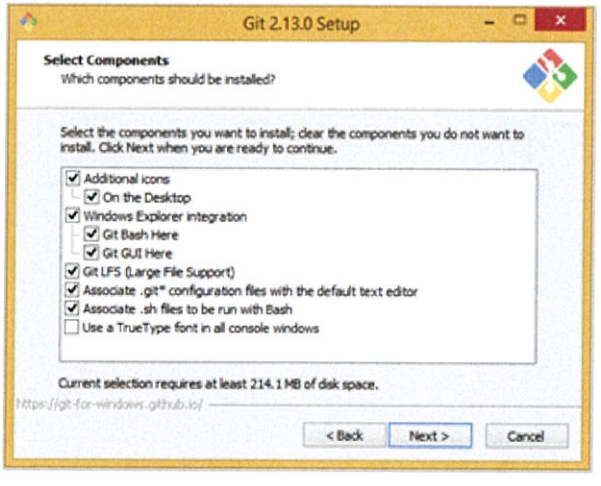
ภาพที่ 2.30 การเลือกส่วนเพิ่มเติมของโปรแกรม Arduino

เมื่อติดตั้งเสร็จแล้ว จะขึ้นคำว่า Completed ดังภาพ ให้กดปุ่ม Close เสร็จแล้วให้ติดตั้ง Arduino core for ESP32 ก่อนอื่นต้องติดตั้งโปรแกรม Git ให้เข้าไปที่หน้า https://gitscm.com/download/win จะมีไฟล์ขึ้นมาให้ดาวน์โหลดอัตโนมัติเมื่อเปิดไฟล์จะมีรายละเอียดเงื่อนไขการใช้งานโปรแกรมขึ้นมา ให้กด NEXT ดังภาพที่ 2.31



ภาพที่ 2.31 ติดตั้งโปรแกรม Git

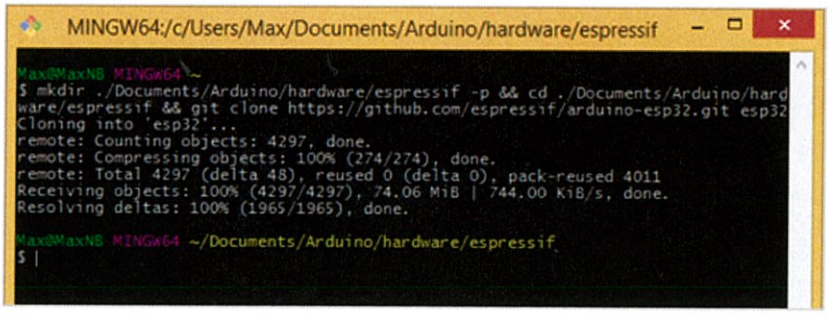
ในหน้านี้ให้เลือกตามภาพให้ครบ จากนั้นกด NEXT ดังภาพที่ 2.32



ภาพที่ 2.32 การเลือกส่วนเพิ่มเติมการติดตั้งของ Git

จากนั้นให้กด NEXT ไปเรื่อย ๆ ในหน้าสุดท้าย ให้กด Install รอติดตั้งโปรแกรมซักครู่ เมื่อติดตั้งเสร็จเรียบร้อยแล้ว ให้เลือก Launch Git Bash แล้วกด Finish จะมีหน้าต่าง Bash ขึ้นมา ให้พิมพ์ชุดคำสั่งต่อไปนี้ (พิมพ์ติดกัน ไม่เว้นบรรทัด) ดังภาพที่ 2.33 จากนั้นกด Enter ได้ดังภาพที่ 2.34

ภาพที่ 2.33 ชุดคำสั่งติดตั้ง Git

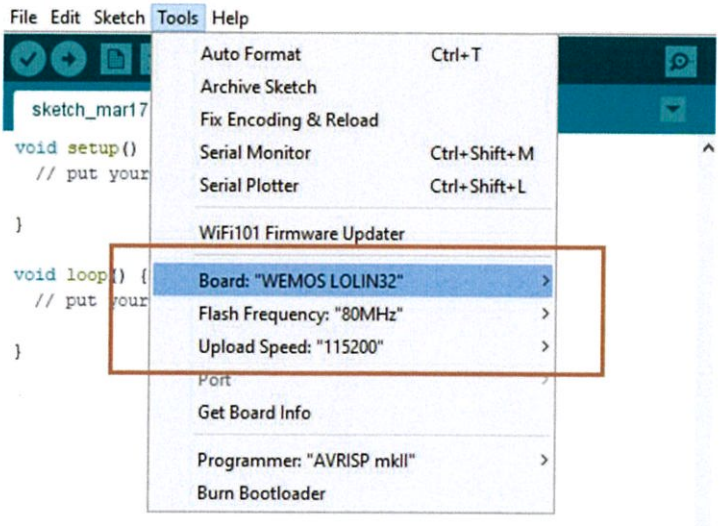


ภาพที่ 2.34 หน้าต่างเมื่อลงโปรแกรมสำเร็จ

เข้าไปที่ Documents\Arduino\hardware\espressif\esp32\tools ดับเบิลคลิกเปิดไฟล์ get.exe ขึ้นมา โปรแกรมจะดาวน์โหลดซอฟต์แวร์ต่าง ๆ รอจนกระทั่งหน้าต่างปิดไป

2.8.3 การใช้งานเบื้องต้นของ ESP32 กับ Arduino

สำหรับการใช้งานเบื้องต้นของไมโครคอนโทรลเลอร์ ESP32 รุ่น WeMos LOLIN32 จะกล่าวถึงการอ่านเขียนส่วนของอินพุทและเอาท์พุทของไมโครคอนโทรลเลอร์ ESP32 โดยใช้โปรแกรม Arduino IDE เป็นตัวเขียนโปรแกรมและดาวน์โหลดเข้ากับบอร์ด WeMos LOLIN32 โดยมีขั้นตอนการเลือกบอร์ดที่ใช้งานร่วมกับโปรแกรม Arduino IDE ดังภาพที่ 2.35



ภาพที่ 2.35 การเลือก Board ที่ใช้งานและปรับ Baud Rate

การใช้ฟังก์ชั่นในการกำหนดขาใช้งาน การอ่านค่าดิจิตอลอินพุท เขียนค่าดิจิตอลเอาท์ อ่านค่าอะนาล็อกอินพุท เขียนค่าอะนาล็อกเอาท์พุท เบื้องต้นดังนี้

1. การกำหนดขาที่ใช้งาน

ในการใช้งานขา GPIO จำเป็นต้องมีการควบคุมหรือบอกให้ขา GPIO นั้น ๆ ได้รู้ว่าต้องการจะให้ใช้รับค่าดิจิตอลเข้ามา หรือจะให้เขียนค่าออกไป ซึ่งจะใช้ฟังก์ชั่น pinMode() ในการกำหนดฟังก์ชั่น pinMode() มีรูปแบบการใช้งานดังนี้

void pinMode(**int** pin, **int** mode);

จะเห็นได้ว่าฟังก์ชันมีค่าพารามิเตอร์ 2 ตัว คือ

(**int**) pin – กำหนดหมายเลขขา GPIO ที่ต้องการควบคุม
(**int**) mode – โหมดที่ต้องการกำหนดให้ขา GPIO ซึ่งสามารถเป็นได้ดังนี้
INPUT – กำหนดให้ขา GPIO มีสถานะเป็นอินพุท รอรับค่าเข้ามา
OUTPUT – กำหนดให้ขา GPIO มีสถานะเป็นเอาท์พุท รอเขียนค่าออกไป
INPUT_PULLUP – กำหนดให้ขา GPIO เรียกใช้วงจร Pull-up ภายใน

2. การอ่านค่าดิจิตอลอินพุท

การอ่านค่าดิจิตอลอินพุท คือ การใช้งานขา GPIO ในการอ่านค่า สภาวะแบบดิจิตอล ในการใช้งานขา GPIO แบบดิจิตอลจำเป็นต้องมีการอ่านค่าจากอุปกรณ์ที่ต่อเข้ามาภายในไมโครคอนโทรลเลอร์ ซึ่งจะใช้ฟังก์ชัน digitalRead() ในการอ่านค่าอินพุท

int digitalRead(**int** pin);

มีค่าพารามิเตอร์ตัวเดียว คือ

(**int**) pin – กำหนดหมายเลขขา GPIO ที่ต้องการอ่านค่า

3. การเขียนค่าเอาท์พุทแบบดิจิตอล

การเขียนค่าเอาท์พุทใช้ในการควบคุมอุปกรณ์แบบ 2 สถานะ คือ เปิดและปิดเท่านั้น โดยการควบคุมจะใช้ขา GPIO ในการควบคุมการจ่ายลอจิก ซึ่งขา GPIO จะรองรับการจ่ายกระแสสูงสุดเพียง 12mA เท่านั้น หากต้องการควบคุมอุปกรณ์ที่ใช้กระแสไฟฟ้ามาก จำเป็นต้องใช้วงจร หรืออุปกรณ์เช่น รีเลย์ ในการช่วยขับ ซึ่งการเขียนค่าเอาท์พุทแบบดิจิตอลจะใช้ฟังก์ชัน digitalWrite() มีรูปแบบการใช้งานดังนี้

void digitalWrite(**int** pin, **int** value);

มีค่าพารามิเตอร์ 2 ตัว คือ

(**int**) pin – กำหนดหมายเลขขา GPIO ที่ต้องการเขียนค่า
(**int**) value – ค่าที่ต้องการเขียนออกไป สามารถเป็นได้ดังนี้
HIGH – เขียนค่าลอจิก 1 หรือสถานะ HIGH หรือแรงดัน 3V
LOW – เขียนค่าลอจิก 0 หรือสถานะ LOW หรือแรงดัน 0V

4. การอ่านค่าอะนาล็อกอินพุท

การอ่านค่าอะนาล็อกอินพุทใช้งานผ่านขา ADC (Analog to Digital Converter) ใช้ในการวัดแรงดันไฟฟ้า สำหรับ ESP32 สามารถใช้งาน ADC ได้ที่ความละเอียด 12 บิต หรือค่าที่ได้จะอยู่ในช่วง 0 – 4095 โดยค่าแรงดันที่วัดได้สูงสุดจะเรียกว่า แรงดันอ้างอิง สมมุติกำหนดให้แรงดันอ้างอิงเท่ากับ 3.3 V หากวัดแรงดันได้ 3.3 V จะอ่านค่าจาก ADC ได้ 4095 การอ่านค่า ADC จะใช้ฟังก์ชัน analogRead()

int analogRead(**int** ch);

มีพารามิเตอร์อยู่ตัวเดียว คือ
(**int**) ch – หมายเลขช่อง ADC ที่ต้องการจะวัด หรือใส่หมายเลขขา GPIO และมีการตอบกลับมาเป็นค่าที่อ่านได้ ซึ่งจะมีค่า 0 – 4095

5. การเขียนค่าอะนาล็อกเอาท์พุท

การเขียนค่าอะนาล็อกเอาท์พุทต้องใช้งานผ่านขา DAC (Digital to Analog Converter) ใช้กำหนดค่าแรงดันเอาท์พุทเพื่อนำไปควบคุมอุปกรณ์ที่ใช้แรงดันไฟฟ้าในการทำงาน ใน ESP32 มีให้ใช้งานจำนวน 2 ขา คือขา GPIO25 และ GPIO26 สามารถใช้งานได้ความละเอียด 8 บิต แรงดันเอาท์พุทสูงสุด 3.3 V ฟังก์ชันที่ใช้คือ dacWrite() โดยมีรูปแบบการใช้งานดังนี้

void dacWrite(**int** pin, **int** value);

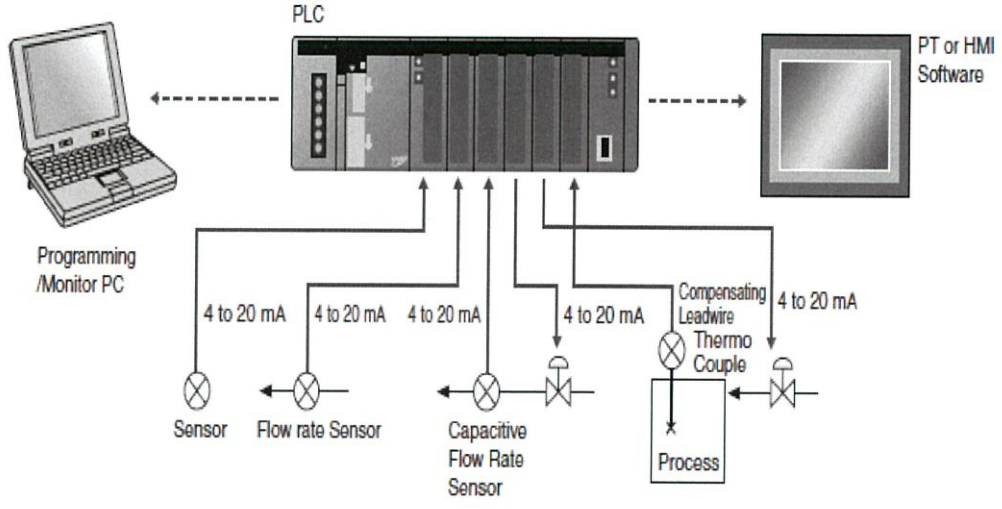
มีรายละเอียดของพารามิเตอร์ คือ

(**int**) pin – ขาที่ใช้งาน 2 ขา คือ GPIO25 (DAC1) และ GPIO26 (DAC2)
(**int**) value – ค่าที่ต้องการเขียนออกไป กำหนดได้ 0 ถึง 255

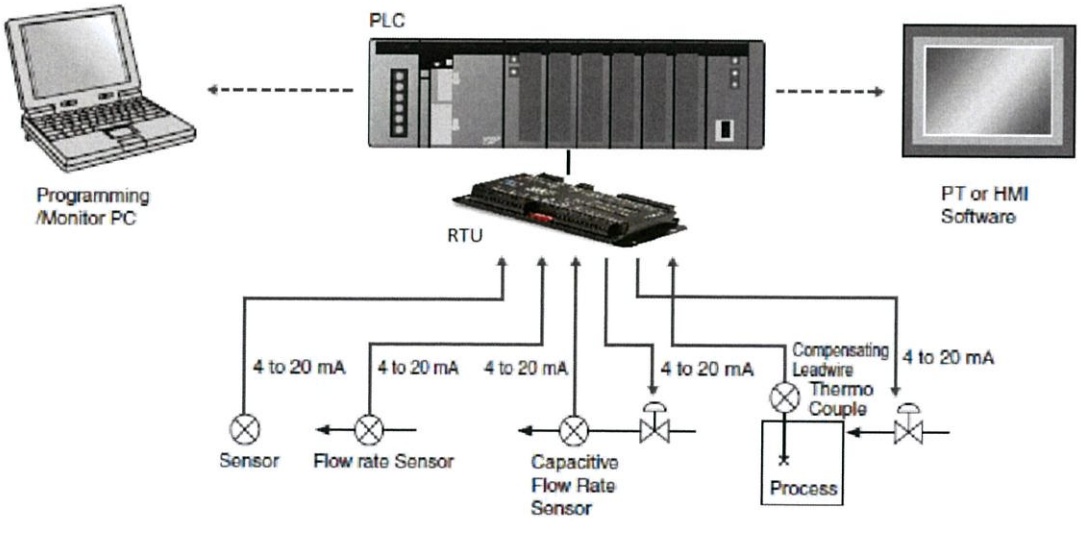
บทที่ 3
ชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทที่นำเสนอ

3.1 กล่าวนำ

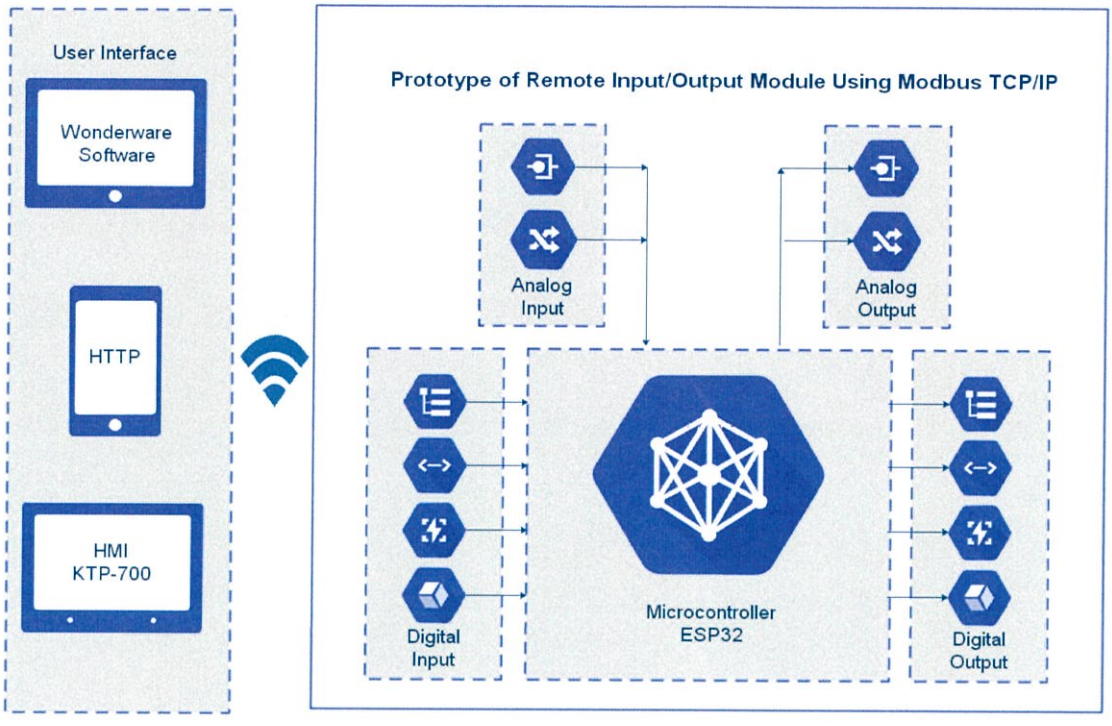
ในอดีตชุดทำงานของ PLC จะประกอบไปด้วยการ์ด I/O ในรูปแบบที่ติดอยู่กับตัว PLC จึงทำให้เกิดปัญหาต่อมาในเรื่องของการใช้งานและค่าใช้จ่ายของสายในการติดตั้งสามารถดูได้จากตัวอย่างการเชื่อมต่อ PLC ในรูปแบบ Direct Wiring จากภาพที่ 3.1 ด้วยปัญหาดังกล่าวจึงได้นำ RTU ดังภาพที่ 3.2 มาช่วยแก้ปัญหานี้ ในโครงงานนี้จึงได้ออกแบบชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพี โดยชุดต้นแบบจะประกอบไปด้วย 3 ส่วนหลัก คือ ส่วนประกอบของฮาร์ดแวร์ โปรแกรมของชุดต้นแบบทางซอฟต์แวร์ และส่วนแสดงผล



ภาพที่ 3.1 ตัวอย่างการใช้งาน PLC รูปแบบ Direct Wiring ในสมัยอดีต



ภาพที่ 3.2 ตัวอย่างการใช้งาน PLC ร่วมกับ RTU



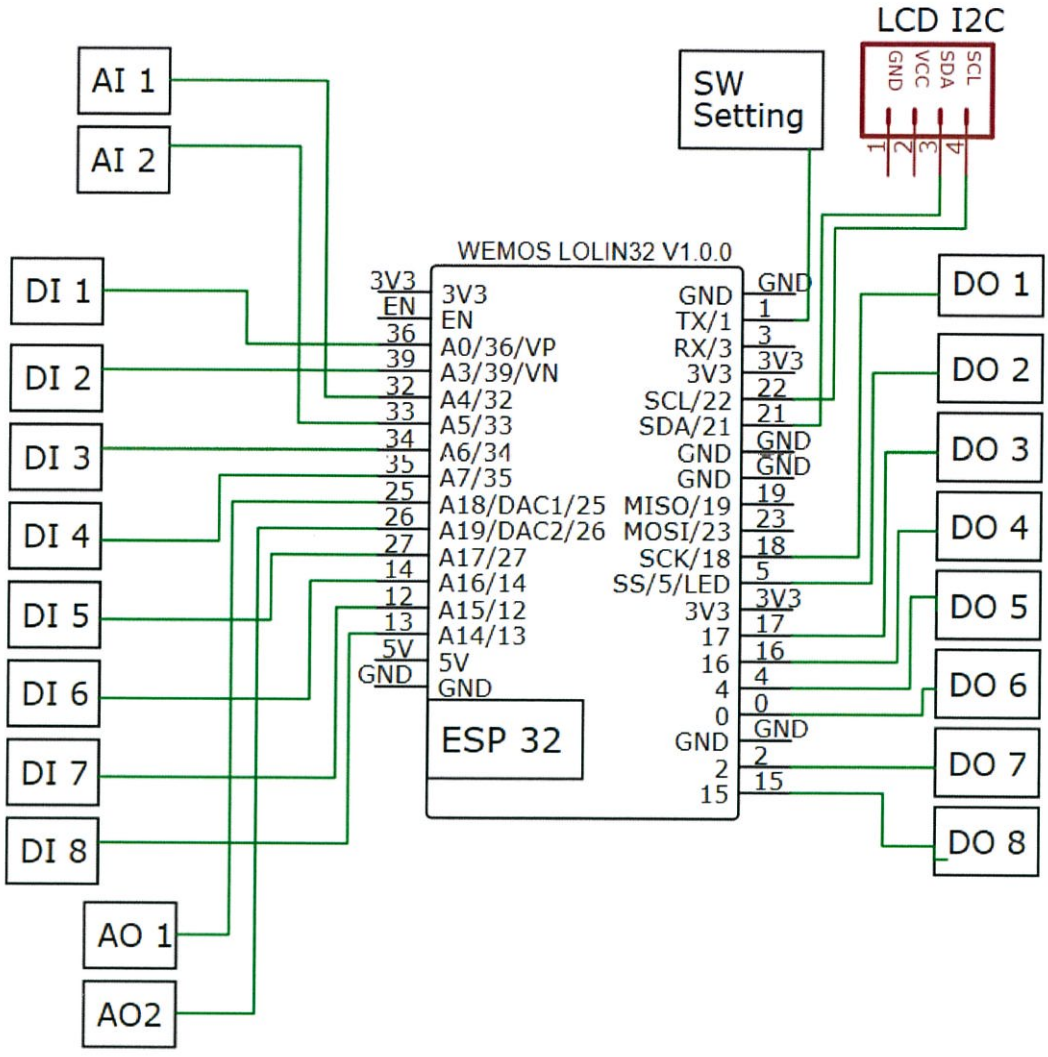
ภาพที่ 3.3 ภาพรวมโครงสร้างของชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพี

3.2 ส่วนประกอบของชุดต้นแบบทางฮาร์ดแวร์

ในส่วนของฮาร์ดแวร์มีการแบ่งออกเป็น 5 ส่วน ได้แก่ บอร์ด ESP32 วงจรดิจิตอลอินพุท วงจรอะนาล็อกอินพุท วงจรดิจิตอลเอาท์พุท และวงจรอะนาล็อกเอาท์พุท สามารถอธิบายได้ดังนี้

3.2.1 บอร์ด ESP32

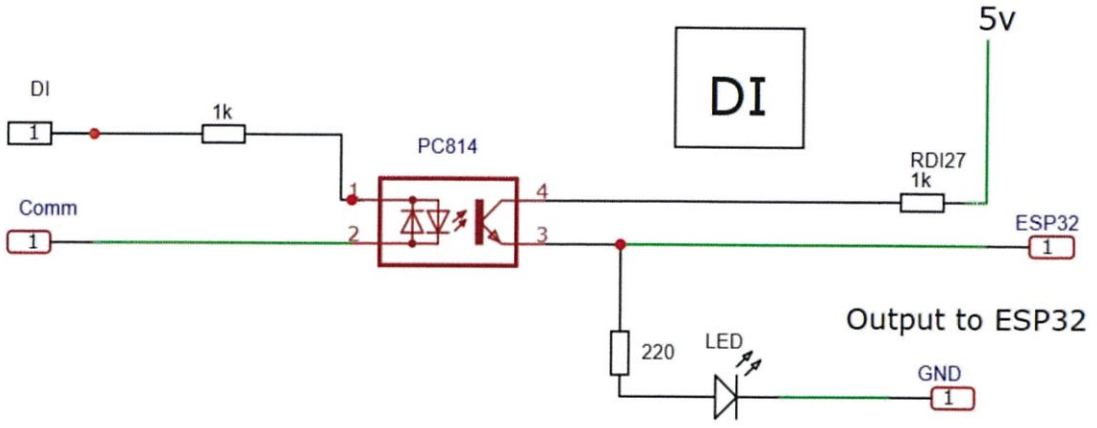
สำหรับบอร์ด ESP32 มีหน้าที่ควบคุมการรับส่งข้อมูลที่มาจากโปรโตคอลมอดบัสทีซีพีไอพีผ่านทางสัญญาณไร้สายแบบ WiFi อ่านค่าข้อมูลจากวงจรดิจิตอลและอะนาล็อกอินพุท เขียนค่าข้อมูลไปยังดิจิตอลและอะนาล็อกเอาท์พุท ผ่านทางขาของไมโครคอนโทรลเลอร์ ESP32 และมีจอแสดงผล LCD เพื่อแสดงสถานะการเชื่อมต่อ การตั้งค่าเน็ตเวิร์ค



ภาพที่ 3.4 ภาพรวมการเชื่อมต่อของแต่ละขาของบอร์ด ESP32

3.2.2 วงจรดิจิตอลอินพุท

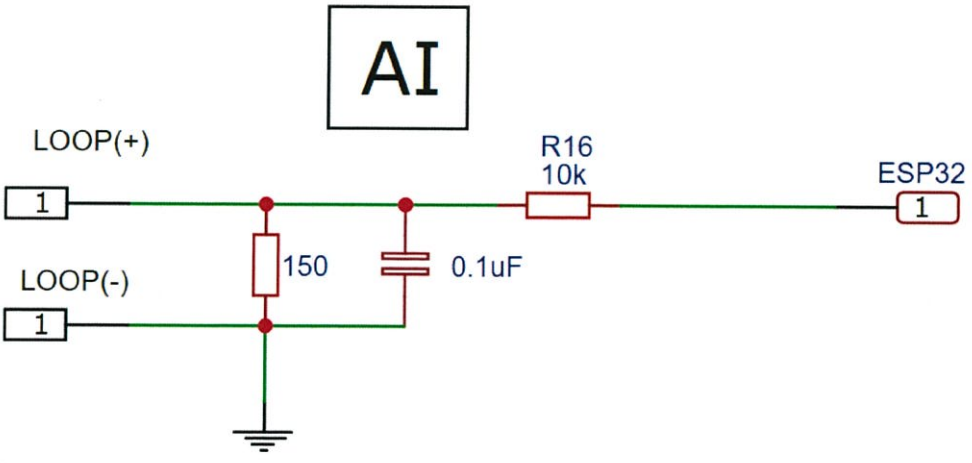
สำหรับวงจรดิจิตอลอินพุทจะใช้วงจร Optocoupler แบบ AC โดยรับค่าสัญญาณอินพุทเข้ามาทำงานที่ขา 1 ของ Optocoupler และมีตัวต้านทาน 1k โอห์มอนุกรมกับขาอินพุท เพื่อจำกัดกระแสที่ไหลเข้า Optocoupler เพื่อป้องกันไม่ให้เกิดอันตรายต่อวงจร เมื่อมีสัญญาณดิจิตอลอินพุทที่มีค่าสัญญาณลอจิก 1 เข้าที่ขาอินพุท เอาท์พุทของ Optocoupler ที่เป็นทรานซิสเตอร์เอาท์จะจ่ายสัญญาณลอจิก 1 ไปยังหลอดแอลอีดีสีเขียวและส่งไปที่ขา ESP32 ดังภาพที่ 3.5 ซึ่งวงจรดิจิตอลอินพุทมีทั้งหมด 8 ช่องสัญญาณ และแบ่งเป็น 2 คอมมอน คอมมอนละ 4 ช่องสัญญาณ ทำให้วงจรดิจิตอลอินพุทสามารถรับสัญญาณที่เป็น Sink หรือสัญญาณที่เป็น Source ได้



ภาพที่ 3.5 วงจรดิจิตอลอินพุทในชุดต้นแบบ

3.2.3. วงจรอะนาล็อกอินพุท

วงจรอะนาล็อกอินพุทเป็นวงจรที่รับกระแส 0-22 mA มาจากภายนอกส่งไปยัง ESP32 โดยมีตัวต้านทาน 150 โอห์ม ทำหน้าที่เปลี่ยนจากกระแสไฟฟ้า 0-22 mA เป็น 0-3.3 V ค่ากระแสที่เป็นมาตรฐานในการรับส่งในอุตสาหกรรมอยู่ที่ 4-20 mA แต่ในบางกรณีค่าที่ส่งอาจมีการคลาดเคลื่อน ดังนั้นวงจรที่ออกแบบไว้ให้สามารถรับได้สูงสุด 22 mA ส่วนถัดมามีตัวเก็บประจุ 0.1 uF และความต้านทาน 10 k โอห์ม ซึ่งช่วยปรับให้แรงดันที่ถูกเปลี่ยนจากกระแสเข้ามาในวงจรมีความเสถียร แสดงดังภาพที่ 3.6

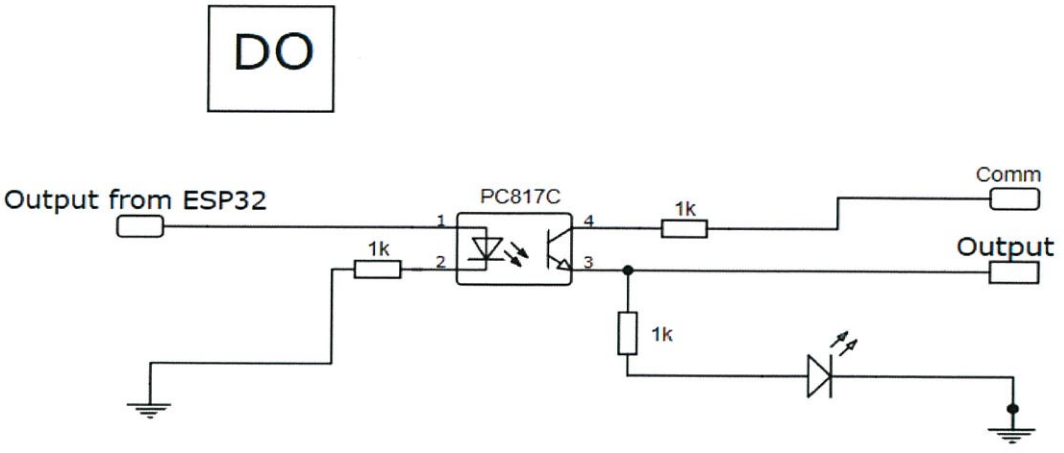


ภาพที่ 3.6 วงจรอะนาล็อกอินพุทในชุดต้นแบบ

3.2.4 วงจรดิจิตอลเอาท์พุท

วงจรดิจิตอลเอาท์พุททำหน้าที่ส่งสัญญาณจาก ESP32 ไปยังขาเอาท์พุทของชุดต้นแบบ โดยมี Optocoupler แยกสัญญาณเอาท์พุทของ ESP32 กับขาเอาท์พุทของชุดต้นแบบและที่ขาเอาท์พุทเทอร์มินอลมีหลอดแอลอีดีสีแดงในชุดต้นแบบใช้แสดงผลสถานะเปิดหรือปิด ที่เอาท์พุทเทอร์มินอลมีตัวต้านทาน 1 k โอห์ม เพื่อป้องกันกระแสเกิน โดยวงจรดิจิตอลเอาท์พุทมีทั้งหมด 8

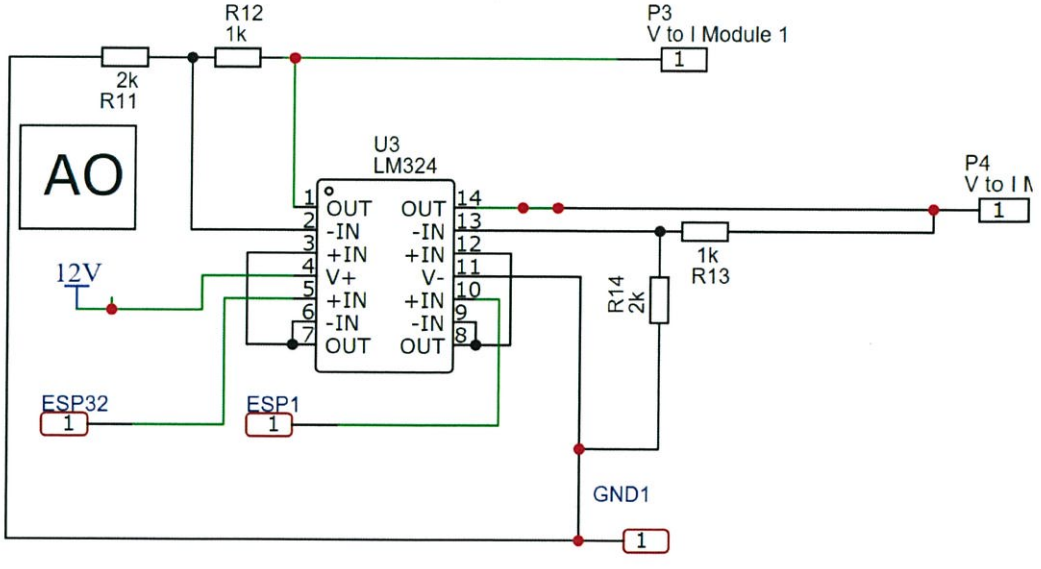
ช่องสัญญาณ และแบ่งเป็น 2 คอมมอน คอมมอนละ 4 ช่องสัญญาณ ทำให้วงจรดิจิตอลเอาท์พุท สามารถจ่ายแรงดันไฟฟ้าที่แตกต่างกันได้ ดังภาพที่ 3.7



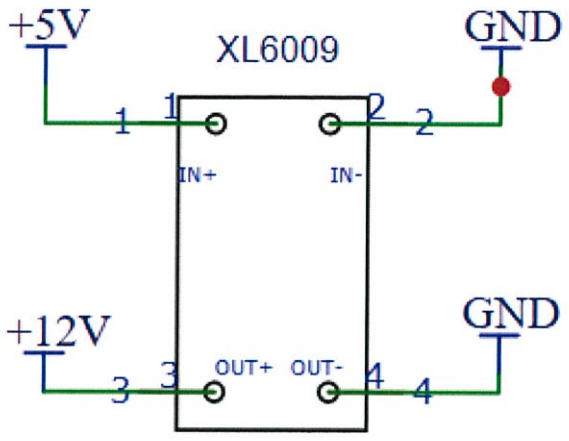
ภาพที่ 3.7 วงจรดิจิตอลเอาท์พุทในชุดต้นแบบ

3.2.5 วงจรอะนาล็อกเอาท์พุท

วงจรอะนาล็อกเอาท์พุทเป็นวงจรที่รับอินพุทมาจาก ESP32 จากขา DAC (Digital Analog Convertor) แต่เนื่องจากขา DAC ของ ESP32 มีแรงดันมากสุด 3.3 V แต่ในวงจรต้องการแรงดันมากสุดที่ 5 V จึงใช้ออปแอมป์เบอร์ LM324 เพื่อขยายแรงดันที่เข้ามา โดยไฟเลี้ยงออปแอมป์ ใช้ไฟ 12 V ที่ได้จากโมดูลสวิชชิ่งเรกูเรเตอร์แบบสเต็ปอัพ สัญญาณเอาท์พุทของออปแอมป์ไปแปลงเป็นสัญญาณอะนาล็อกผ่านทางโมดูลแปลงสัญญาณ 0 - 5 V เป็น 0 – 20 mA (DC To 0-20 mA) เพื่อให้ได้เอาท์พุทเป็นสัญญาณอะนาล็อกตามที่ต้องการ



ภาพที่ 3.8 วงจรอะนาล็อกเอาท์พุทในชุดต้นแบบ



ภาพที่ 3.9 วงจรสวิชชิ่งเรกูเรเตอร์สเต็ปอัพ 5 V to 12 V ในชุดต้นแบบ

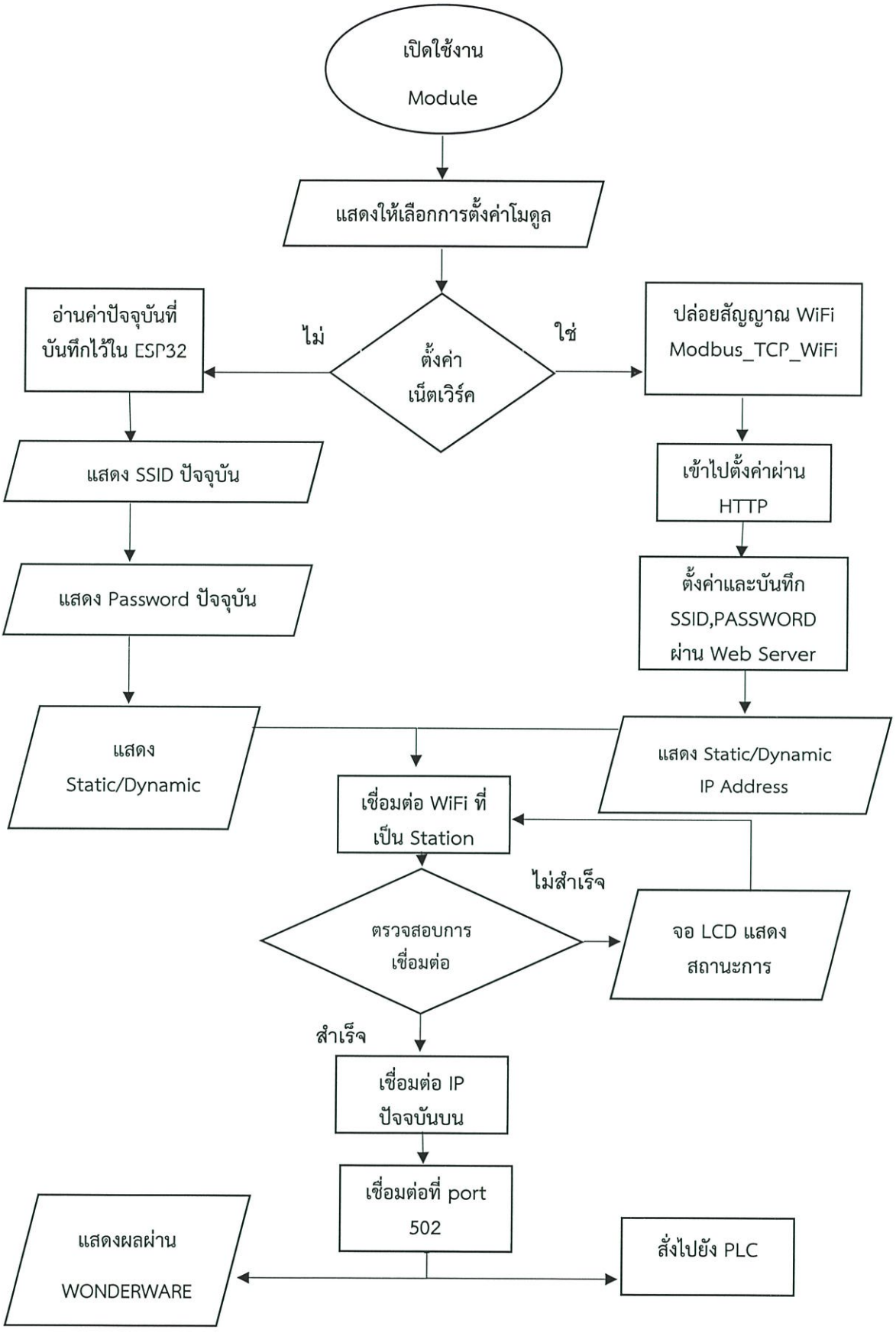
3.3 โปรแกรมของชุดต้นแบบทางซอฟต์แวร์

สำหรับซอฟต์แวร์ที่ใช้ร่วมกับชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพีนี้ คือ ซอฟต์แวร์ Arduino ModScan32 TIA Portal และ WONDERWARE InTouch โดยซอฟต์แวร์ Arduino จะใช้เขียนโค้ดสำหรับไมโครคอนโทรลเลอร์ ESP32 ซอฟต์แวร์ ModScan32 ใช้ตรวจสอบการเชื่อมต่อและแสดงผลอินพุทและเอาท์พุทของโมดูลต้นแบบ ซอฟต์แวร์ TIA Portal สาหรับเขียนโปรแกรมพีแอลซี ยี่ห้อ Siemens รุ่น S7-1200 เพื่อใช้ทดสอบการติดต่อกับชุดต้นแบบ และซอฟต์แวร์ WONDERWARE InTouch ใช้สำหรับแสดงผลผ่านทางหน้าจอคอมพิวเตอร์

3.3.1 โปรแกรมควบคุมไมโครคอนโทรลเลอร์ Arduino

ในการใช้งานไมโครคอนโทรลเลอร์เพื่อควบคุมอุปกรณ์อิเล็กทรอนิกส์มีขั้นตอนการใช้งานดังภาพที่ 3.10 นิยมใช้โปรแกรม Arduino IDE ซึ่งเป็น Open-Source Platform โดยมีจุดประสงค์เพื่อควบคุมอุปกรณ์อิเล็กทรอนิกส์ แต่เนื่องจากบทที่แล้วได้กล่าวถึงขั้นตอนการใช้งานเบื้องต้นไปแล้ว ดังนั้นในบทนี้จึงขอเสนอขั้นตอนการทำงานที่เกี่ยวข้องกับโครงงานนี้

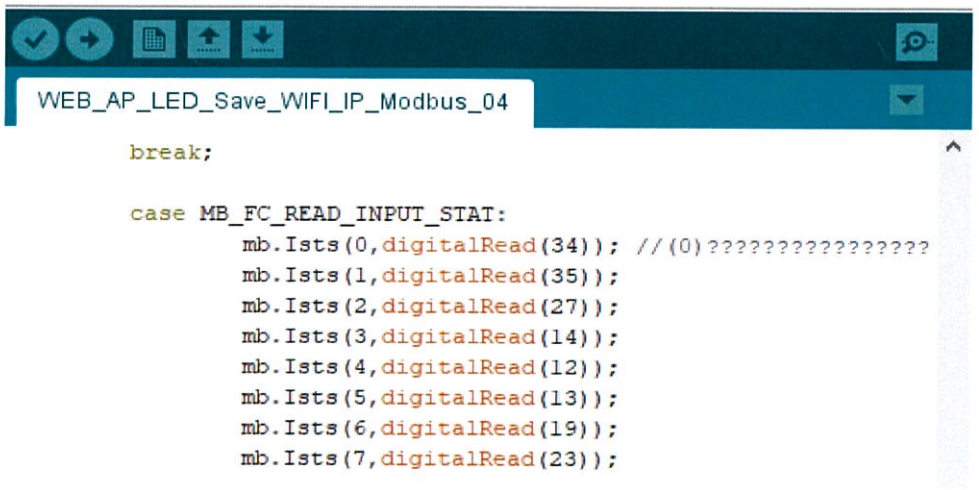
เมื่อเริ่มต้นเปิดชุดต้นแบบจะแสดงผลสถานะของไอพีปัจจุบัน ถ้าต้องการตั้งค่าเน็ตเวิร์คให้กดปุ่ม Setting ค้าง จนกระทั่งหน้าจอ LCD แสดงคำว่า SW_Setting จากนั้นใช้อุปกรณ์ที่สามารถเชื่อมต่ออินเทอร์เน็ตเชื่อมต่อ WiFi ที่ชื่อ MODBUS_TCP_WIFI แล้วใช้เบราเซอร์เปิดหน้าเว็ปเพจ 192.168.4.1 จะเข้าสู่หน้าต่างการตั้งค่า SSID PASSWORD และ IP_ADDRESS แล้วกดบันทึกเมื่อสิ้นสุดการตั้งค่า เมื่อต้องการให้ชุดต้นแบบทำตามค่าที่บันทึกไว้ ขณะเปิดเครื่องไม่ต้องกดปุ่ม Setting โปรแกรมจะอ่านค่าที่บันทึกไว้นำมาใช้สำหรับเชื่อมต่อกับเน็ตเวิร์คที่ต้องการ เมื่อเชื่อมต่อเน็ตเวิร์คได้แล้ว จอแสดงผล LCD จะแสดงผลว่าเชื่อมต่อสำเร็จแล้ว ในส่วนต่อไปโปรแกรมจะเข้าสู่ โหมดการรับส่งข้อมูลด้วยมอดบัสทีซีพีไอพีที่พอร์ตหมายเลข 502 เมื่อมีตัวควบคุม เช่น พีแอลซี WONDERWARE InTouch หรือ ModScan เชื่อมต่อมายังชุดต้นแบบจอแสดงผล LCD จะแสดงผลว่ามีการเชื่อมต่อกับตัวควบคุมดังกล่าวและเมื่อตัวควบคุมยกเลิกการเชื่อมต่อ จอแสดงผล LCD จะแสดงผลว่าตัวควบคุมยกเลิกการเชื่อมต่อแล้ว ตัวชุดต้นแบบยังคงรอการเชื่อมต่ออีกครั้ง



ภาพที่ 3.10 แผนภาพการทำงานโปรแกรม ESP32

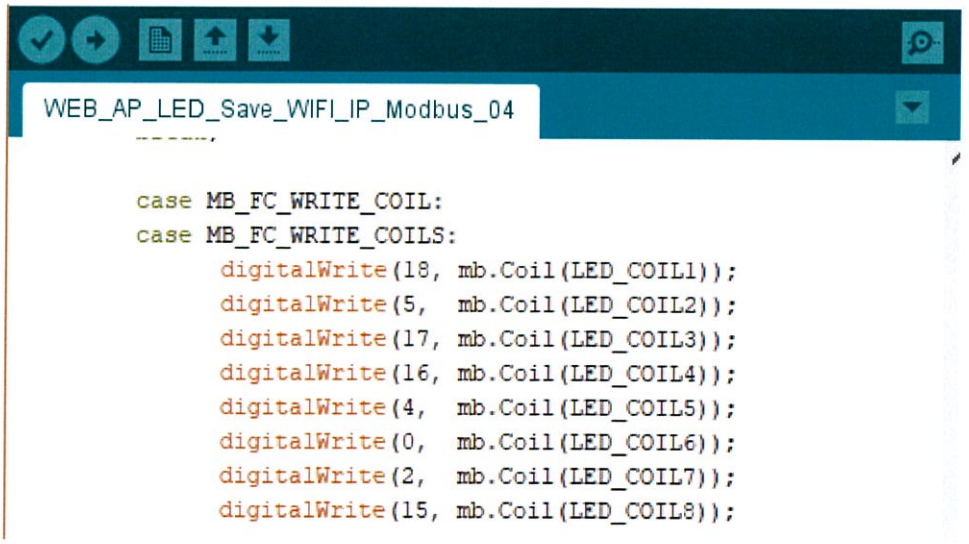
3.3.2 การกำหนดขาอินพุทและเอาท์พุท

ในการอ่านค่าอินพุทจะใช้คำสั่ง digitalRead(Pin Input) ค่าที่ได้จะเก็บไว้ในอินพุทรีติสเตอร์ของมอดบัส (mb.Ists(Input Contact,Value)) จากภาพที่ 3.11 มีการอ่านค่าทั้งหมด 8 ช่อง ช่องแรกสุดอยู่ที่ตำแหน่งแอดเดรส 10001 จะตรงกับอินพุทของ ESP32 ที่ขา 34 และแอดเดรสต่อมาจะอ่านค่าตามภาพด้านล่างจนกระทั่งครบ 8 ช่อง



ภาพที่ 3.11 โค้ดการอ่านค่าอินพุท

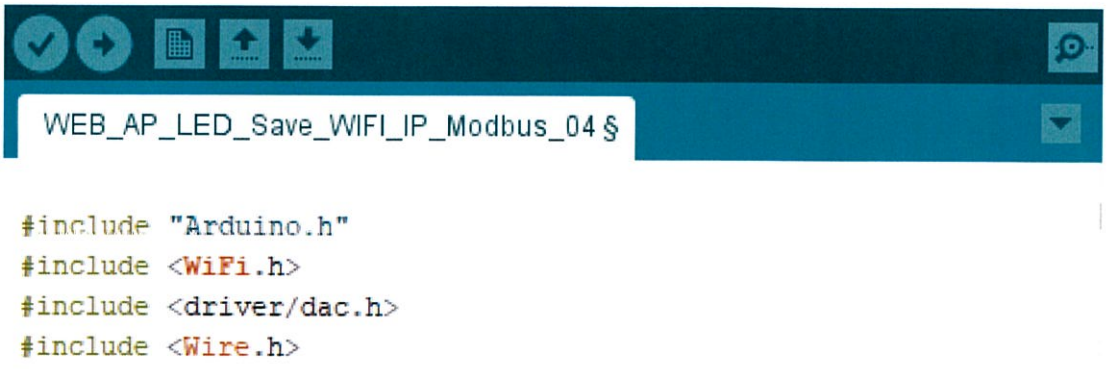
ในการเขียนค่าเอาท์พุทจะใช้คำสั่ง digitalWrite(Pin Output , Value) ค่าที่ต้องการเขียนไปที่เอาท์พุทจะมาจากค่าของคอยล์รีจิสเตอร์มอดบัส (mb.Coil (LED_Coil)) จากภาพที่ 3.12 มีการเขียนค่าทั้งหมด 8 ช่อง ช่องแรกสุดอยู่ที่ตำแหน่งแอดเดรส 00001 จะตรงกับเอาท์พุทของ ESP32 ที่ขา 18 และแอดเดรสต่อมาจะเขียนค่าตามภาพด้านล่างจนกระทั่งครบ 8 ช่อง



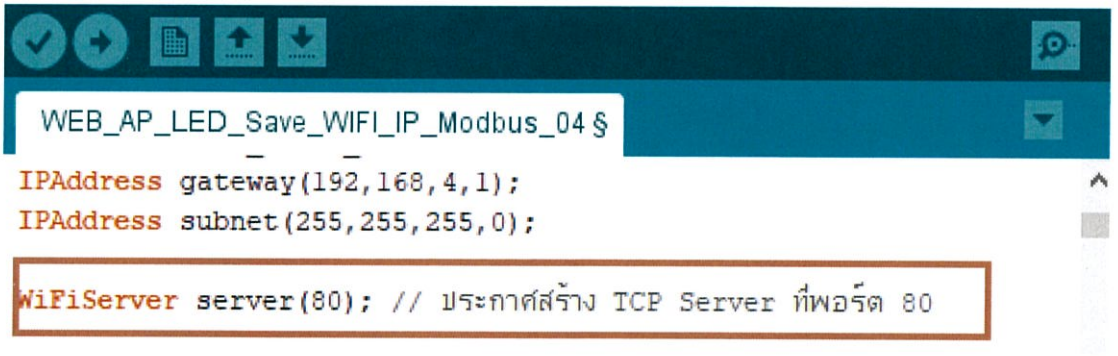
ภาพที่ 3.12 โค้ดการเขียนค่าเอาท์พุท

3.3.3 ฟังก์ชันที่เกี่ยวข้องกับการใช้งานโปรโตคอล TCP

การสร้าง TCP Server จะต้องเรียกใช้ไลบารี่ WiFi.h ทุกครั้ง ดังภาพที่ 3.13 และการใช้งานจะต้องสร้างออปเจคของคลาส WiFi Server ดังภาพที่ 3.14 ขึ้นมานอกฟังก์ชันย่อย มีรูปแบบดังนี้



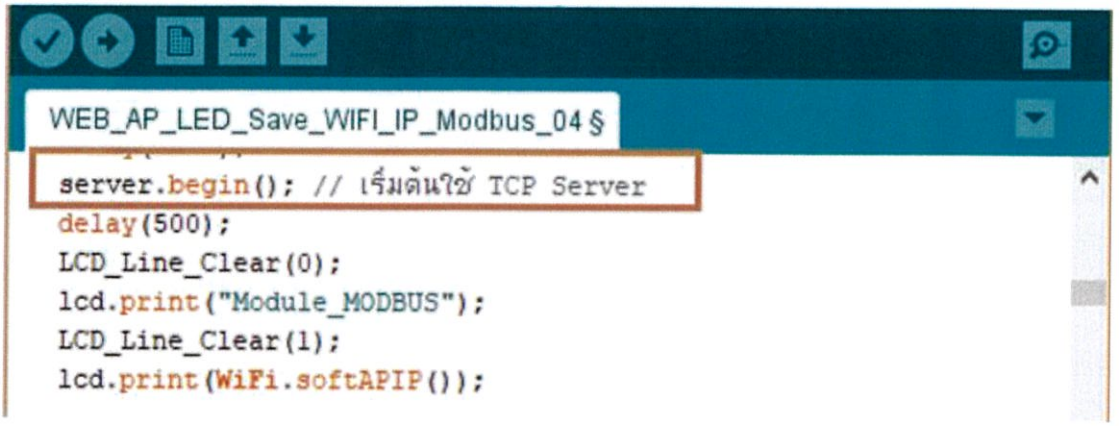
ภาพที่ 3.13 โค้ดการเปิดใช้งานฟังก์ชัน WiFi



ภาพที่ 3.14 โค้ดการกำหนดฟังก์ชัน TCP Server

3.3.4 เริ่มต้นสร้าง TCP Server

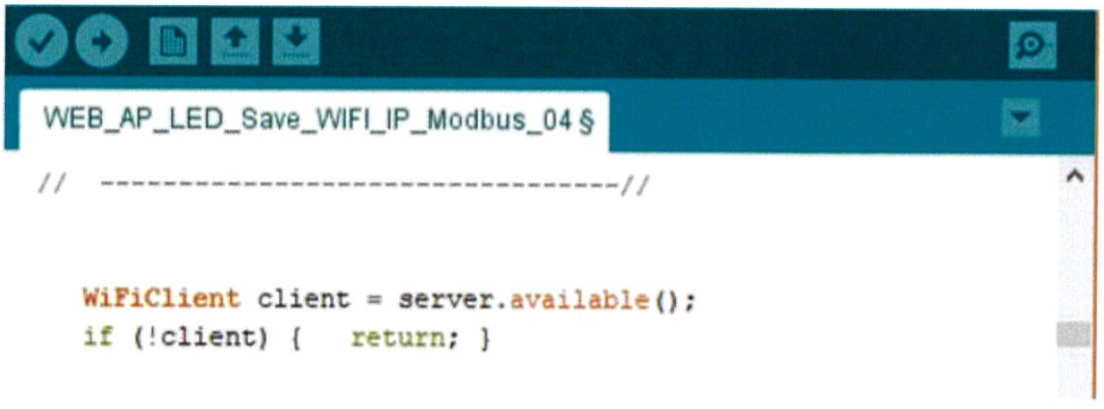
จะใช้ฟังก์ชันย่อย .begin() มีรูปแบบการใช้งานดังนี้ void WiFiServer::begin(); ไม่มีค่าพารามิเตอร์และไม่มีการตอบกลับฟังก์ชันย่อย .begin() จะต้องนำไปไว้ในฟังก์ชัน setup ทุกครั้ง เพื่อให้เริ่มสร้าง TCP Server ขึ้นมาใช้งาน ดังภาพที่ 3.15



ภาพที่ 3.15 โค้ดการเปิดใช้งานฟังก์ชัน TCP Server

3.3.5 ตรวจสอบการเชื่อมต่อเข้ามา

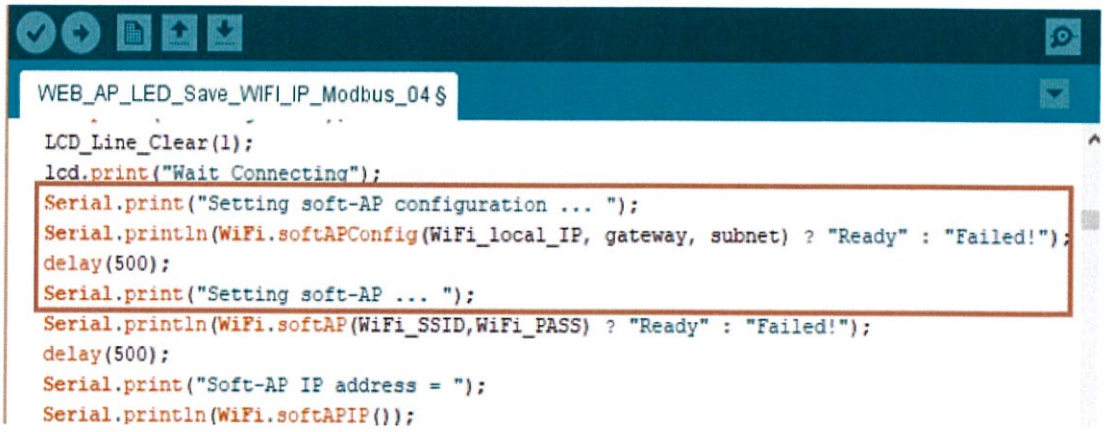
หลังจากใช้ฟังก์ชันย่อย .begin() ไปแล้ว จะต้องเริ่มตรวจสอบว่ามี Client เข้ามาเชื่อมต่อหรือไม่ใช้ฟังก์ชันย่อย .available() ซึ่งมีรูปแบบการใช้งาน ดังภาพที่ 3.16



ภาพที่ 3.16 โค้ดการตรวจสอบการเชื่อมต่อของ WiFi

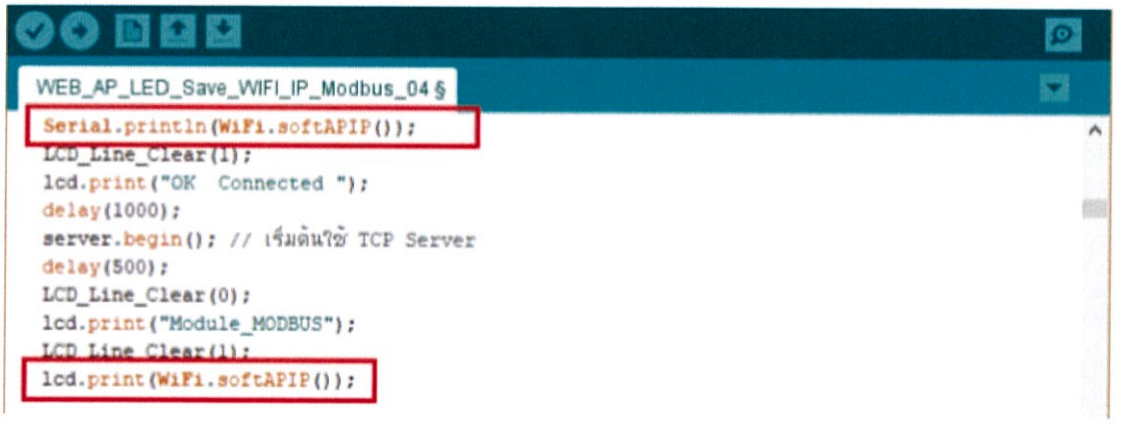
3.3.6 การ Configure โมดูลผ่าน WiFi ในโหมด AP

WiFi ถือเป็นหัวใจสำคัญและเป็นจุดเด่นของ ESP32 เลยก็ว่าได้ ด้วยการรวมส่วนของ WiFi มาในชิปพร้อมกับไมโครคอนโทรลเลอร์ทำให้ประหยัดพื้นที่โดยรวมของระบบได้ นอกจากนี้การใช้งาน WiFi ใน ESP32 ยังง่ายกว่าชิป WiFi ตัวอื่นมาก โดยมีคุณสมบัติพิเศษคือสามารถเลือกโหมดการใช้งาน WiFi ได้ 3 โหมด คือ โหมด AP (Access Point) โหมด STA (Station) และโหมด AP + STA ซึ่งทั้ง 3 โหมดจะมีการใช้งานที่แตกต่างกันเล็กน้อย การเลือกโหมดใช้งานจะเลือกตามลักษณะงานที่นำไปใช้เป็นหลัก แต่เนื่องจากชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพีนี้ใช้งานทั้งในรูปแบบ AP และ STA จึงมีการตั้งค่าการใช้งานดังภาพต่อไปนี้



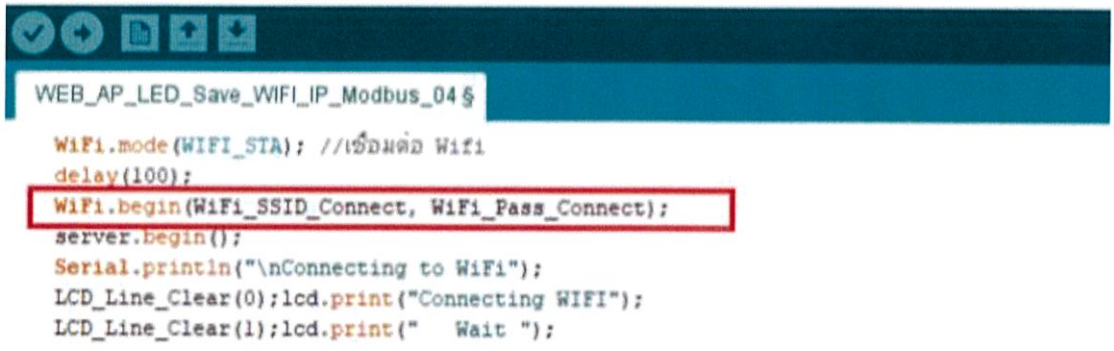
ภาพที่ 3.17 การกำหนดค่า IP ในโหมด AP

เมื่อมีการเชื่อมต่อในโหมด AP สามารถทำงานได้แล้วจะแสดงผล IP Address ผ่านทาง Serial Monitor และ LCD ดังภาพที่ 3.18

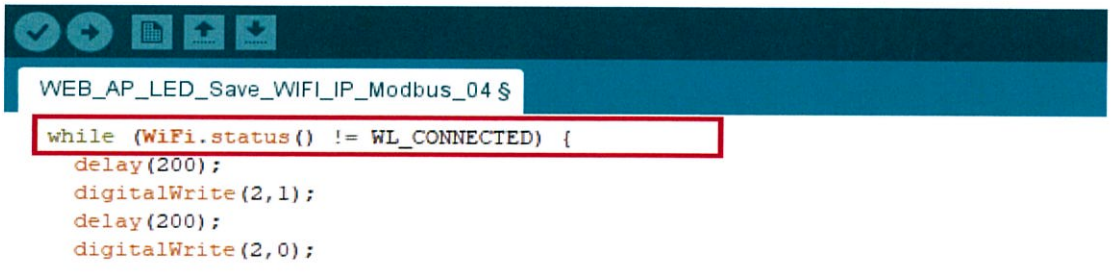


ภาพที่ 3.18 โค้ดการแสดงผล IP Address ผ่านทาง Serial Monitor และ LCD

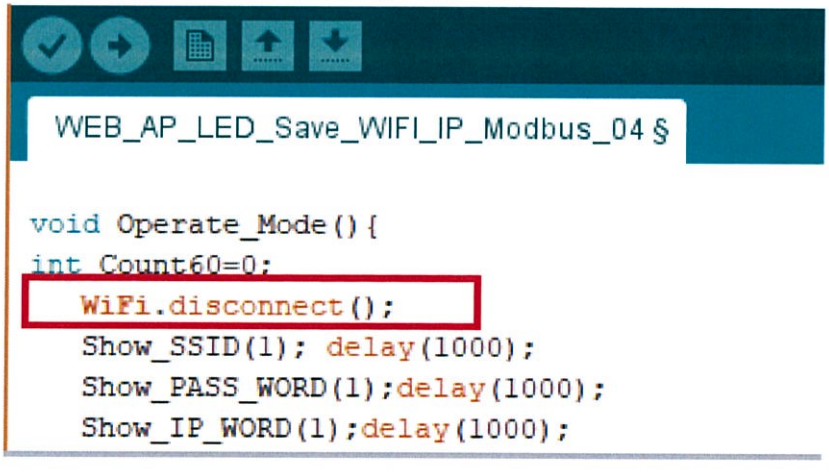
เมื่อผ่านขั้นตอนการเชื่อมต่อในโหมด AP และได้รีเซ็ทชุดต้นแบบแล้ว ชุดต้นแบบจะเข้าสู่โหมด Station โดยจะเชื่อมต่อกับเน็ตเวิร์ค หรือ Access Point ตามที่บันทึกไว้และในส่วน Password จะถูกอ่านเพื่อเชื่อมต่อกับเน็ตเวิร์คที่มีรหัสผ่านและเมื่อกำหนด IP Address แบบ Static ไว้ ชุดต้นแบบจะส่งค่า Static IP ไปยัง Access Point เพื่อกำหนด IP Address ของชุดต้นแบบให้ตรงกับค่าที่บันทึกไว้ หรือขณะที่กำหนด IP Address แบบ Dynamic ชุดต้นแบบจะรอ IP Address จาก Access Point



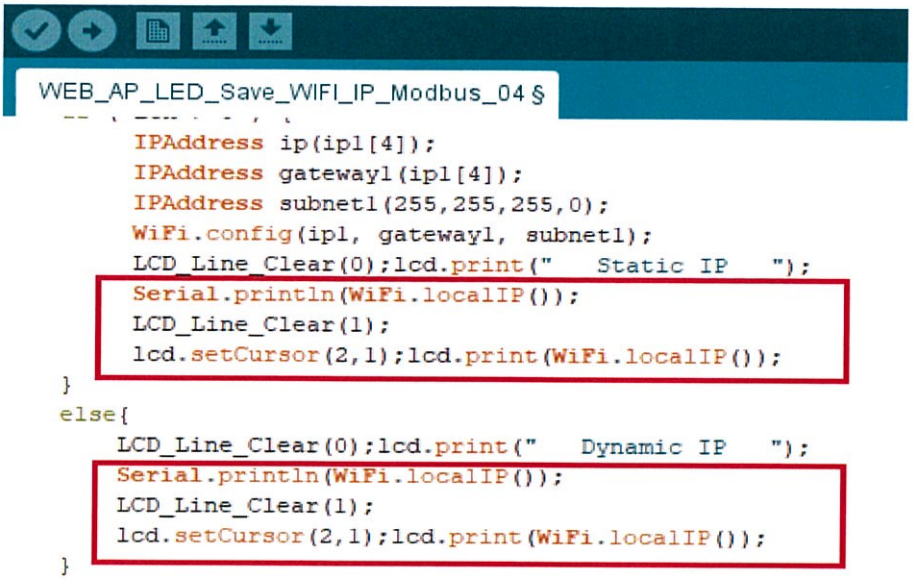
ภาพที่ 3.19 โค้ดการเชื่อมต่อกับ Station



ภาพที่ 3.20 โค้ดตรวจสอบสถานะการเชื่อมต่อกับ Access Point

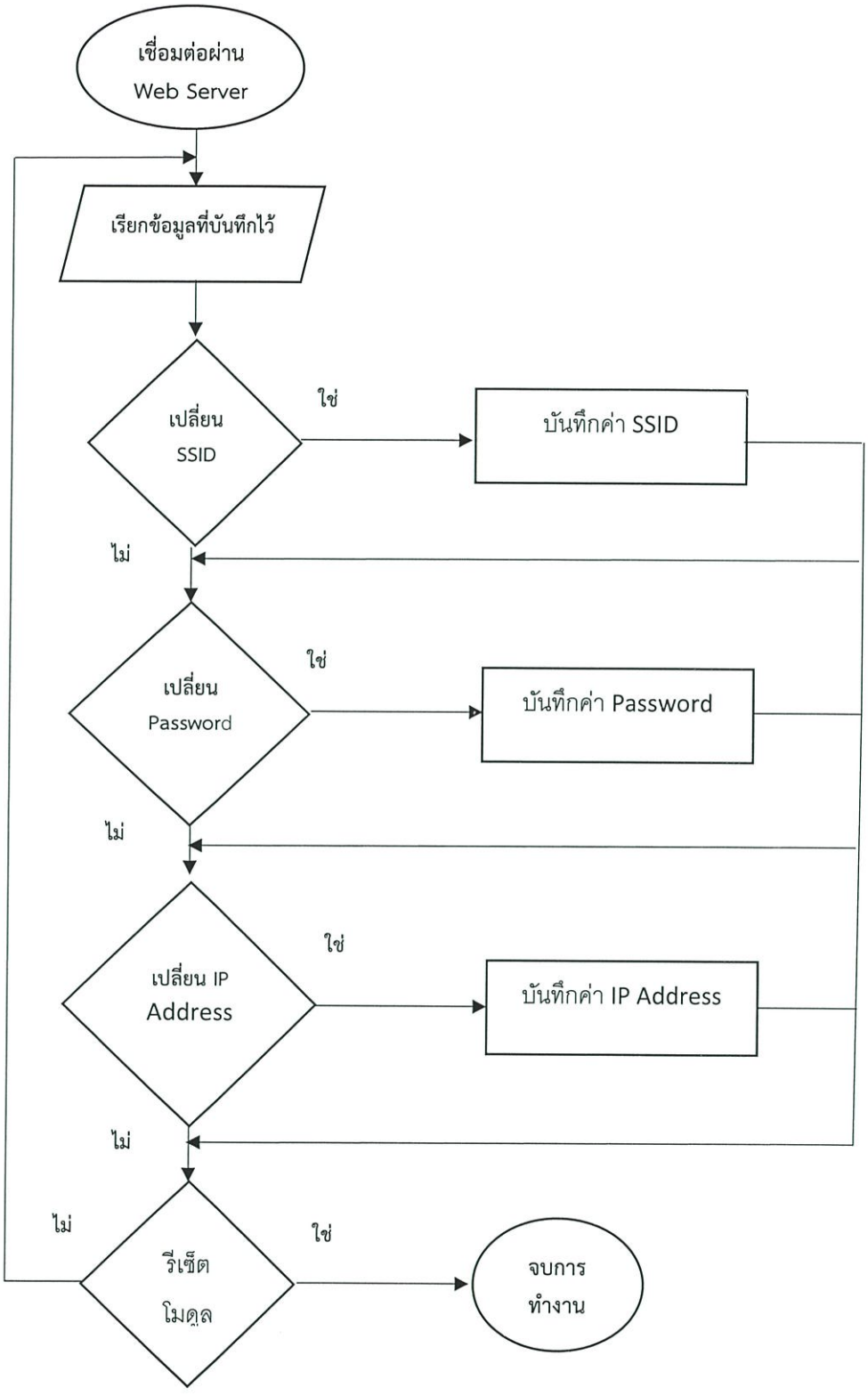


ภาพที่ 3.21 โค้ดการเชื่อมต่อกับ Access Point ใหม่อีกครั้ง



ภาพที่ 3.22 โค้ดการเรียกดูหมายเลข IP ที่ได้จาก Access Point

3.3.7 การออกแบบขั้นตอนการตั้งค่าผ่านทาง HTTP



ภาพที่ 3.23 ลำดับการใช้งานการตั้งค่าผ่านทาง HTTP

ในหัวข้อที่ผ่านมาทางผู้จัดทำได้แสดงโค้ดการเชื่อมต่อและแสดงผลผ่านทาง LCD ในบทนี้จะเป็นการแสดงการตั้งค่าและแสดงผลข้อมูลที่ต้องการผ่านทาง HTTP โดยใช้โค้ดในซอฟต์แวร์ Arduino เพื่อให้ง่ายต่อผู้ที่สนใจสามารถเรียนรู้ผ่านทางภาพด้านล่างดังต่อไปนี้

```
//  SSID Setting ////////////////////////////////////////
   if ( request.indexOf("*W*") > 0 ){
        line = request.substring(request.indexOf("=")+1,request.indexOf("HTTP"));
        unsigned int Len = line.length() ;
        EEPROM.write(0,Len);
        EEPROM.commit();
        for (i = 0 ; i< Len ; i ++){
           EEPROM.write(i+5,line[i]);
           EEPROM.commit();
        }
        Show_SSID(0);
        Serial.println(line);
   }
```

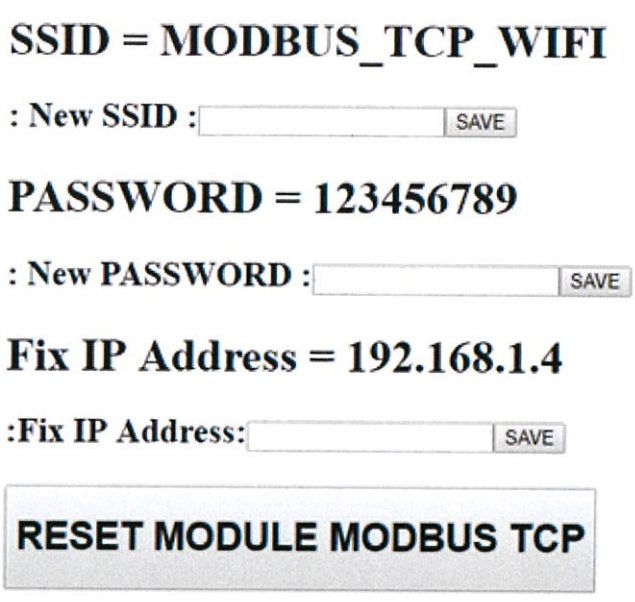
ภาพที่ 3.24 โค้ดคำสั่งสำหรับ HTTP เพื่อตั้งค่า SSID

```
//  PASSWORD Setting ////////////////////////////////////////
   if ( request.indexOf("*P*") > 0 ){
        line = request.substring(request.indexOf("=")+1,request.indexOf("HTTP"));
        unsigned int Len = line.length() ;
        EEPROM.write(3,Len);
        EEPROM.commit();
        for (i = 0 ; i< Len ; i ++){
           EEPROM.write(i+25,line[i]);
           EEPROM.commit();
        }
        Show_PASS_WORD(0);
        Serial.println(line);
   }
```

ภาพที่ 3.25 โค้ดคำสั่งสำหรับ HTTP เพื่อตั้งค่า Password

```
//  IP Address Setting ////////////////////////////////////////
   if ( request.indexOf("*I*") > 0 ){
        line = request.substring(request.indexOf("=")+1,request.indexOf("HTTP"));
        unsigned int Len = line.length() ;
        EEPROM.write(4,Len);
        EEPROM.commit();
        for (i = 0 ; i< Len ; i ++){
           EEPROM.write(i+50,line[i]);
           EEPROM.commit();
        }
        Show_IP_WORD(0);
        Serial.println(line);
```

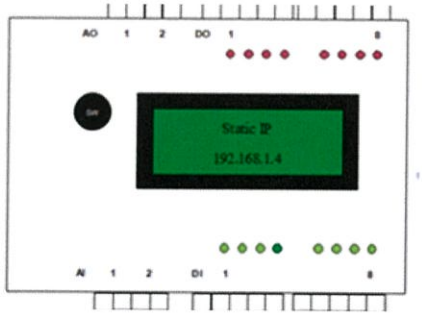
ภาพที่ 3.26 โค้ดคำสั่งสำหรับ HTTP เพื่อตั้งค่า IP Address



ภาพที่ 3.27 หน้าจอการตั้งค่าปัจจุบันผ่าน HTTP

3.4 ส่วนแสดงผลบนชุดต้นแบบ

ส่วนของการแสดงผลผ่านจอ LCD จะใช้แสดงผลสถานการณ์เชื่อมต่อและการตั้งค่าเน็ตเวิร์ค โดยตำแหน่งการวาง LCD บนชุดต้นแบบ แสดงดังภาพที่ 3.28



ภาพที่ 3.28 ตำแหน่ง LCD บนชุดต้นแบบ

โปรแกรมส่วนของการแสดงผลผ่านจอ LCD จะแสดงส่วนของการเชื่อมต่อ IP ของชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพีกับเร้าเตอร์ หรือโทรศัพท์มือถือที่มีการเชื่อมต่อ โดยจะแสดงผลตามคำสั่งต่อไปนี้ในโปรแกรม Arduino

```
WEB_AP_LED_Save_WIFI_IP_Modbus_04 §
  LCD_Line_Clear(0);
// ----------------------------------------//
  lcd.print("Push SW Setting");
```

ภาพที่ 3.29 โค้ดคำสั่งสำหรับ LCD เพื่อตั้งค่าผ่านทางสวิตช์ภายนอก

```
WEB_AP_LED_Save_WIFI_IP_Modbus_04 §
  if ( Mode == 0 ){
      LCD_Line_Clear(0);
      lcd.print("SSID for Setting");
      LCD_Line_Clear(1);
      lcd.print(WiFi_SSID_Connect);
  }
  if ( Mode == 1 ){
      LCD_Line_Clear(0);
      lcd.print("SSID Operate ");
      LCD_Line_Clear(1);
      lcd.print(WiFi_SSID_Connect);
  }
  if ( Mode == 2 ){
     Serial.println(WiFi_SSID_Connect);
  }
```

ภาพที่ 3.30 โค้ดคำสั่งสำหรับ LCD เพื่อตั้งค่า SSID

```
WEB_AP_LED_Save_WIFI_IP_Modbus_04 §
    if ( Mode == 0 ){
      LCD_Line_Clear(0);
      lcd.print("PassWord Setting");
      LCD_Line_Clear(1);
      lcd.print(WiFi_Pass_Connect);
    }
    if ( Mode == 1 ){
      LCD_Line_Clear(0);
      lcd.print("PassWord Operate");
      LCD_Line_Clear(1);
      lcd.print(WiFi_Pass_Connect);
    }
    if ( Mode == 2 ){
     Serial.println(WiFi_Pass_Connect);
    }
}
```

ภาพที่ 3.31 โค้ดคำสั่งสำหรับ LCD เพื่อแสดง Password

WEB_AP_LED_Save_WIFI_IP_Modbus_04 §

```
    if ( Mode == 0 ){
      LCD_Line_Clear(0);
      lcd.print("IP Setting");
      LCD_Line_Clear(1);
      lcd.print(WiFi_IP_Connect);
    }
    if ( Mode == 1 ){
      LCD_Line_Clear(0);
      lcd.print("IP Operate");
      LCD_Line_Clear(1);
      lcd.print(WiFi_IP_Connect);
    }
    if ( Mode == 2 ){
      Serial.println(WiFi_IP_Connect);
    }
}
```

ภาพที่ 3.32 โค้ดคำสั่งสำหรับ LCD เพื่อตั้งค่า IP Address

WEB_AP_LED_Save_WIFI_IP_Modbus_04 §

```
int Loop = 1;
  LCD_Line_Clear(0);
  lcd.print("Setting Wifi");
  LCD_Line_Clear(1);
  lcd.print("Wait Connecting");
  Serial.print("Setting soft-AP configuration ...
  Serial.println(WiFi.softAPConfig(WiFi_local_IP,
  delay(500);
  Serial.print("Setting soft-AP ... ");
  Serial.println(WiFi.softAP(WiFi_SSID,WiFi_PASS)
  delay(500);
  Serial.print("Soft-AP IP address = ");
  Serial.println(WiFi.softAPIP());
  LCD_Line_Clear(1);
  lcd.print("OK  Connected ");
  delay(1000);
```

ภาพที่ 3.33 โค้ดคำสั่งสำหรับ LCD เพื่อตรวจสอบการเชื่อมต่อ Access Point

WEB_AP_LED_Save_WIFI_IP_Modbus_04 §

```
  delay(500);
  LCD_Line_Clear(0);
  lcd.print("Module_MODBUS");
  LCD_Line_Clear(1);
  lcd.print(WiFi.softAPIP());
```

ภาพที่ 3.34 โค้ดคำสั่งสำหรับ LCD เพื่อแสดงผลการเชื่อมต่อ Access Point

WEB_AP_LED_Save_WIFI_IP_Modbus_04 §

```
    case 0:
          LCD_Line_Clear(0);
          lcd.print("Setting Wifi");
          LCD_Line_Clear(1);
          lcd.print("Module_MODBUS_TCP");
    break;
    case 1:
          LCD_Line_Clear(0);
          lcd.print("1. SSID");
          Show_SSID ( 1);
    break;
    case 2:
          LCD_Line_Clear(0);
          lcd.print("2. PASSWORD");
          Show_PASS_WORD(1);
    break;
    case 3:
          LCD_Line_Clear(0);
          lcd.print("3. IP ADDRESS");
          Show_IP_WORD(1);
    break;
  }
}
```

ภาพที่ 3.35 โค้ดคำสั่งสำหรับ LCD เพื่อตั้งค่าเชื่อมต่อ WIFI Station

WEB_AP_LED_Save_WIFI_IP_Modbus_04 §

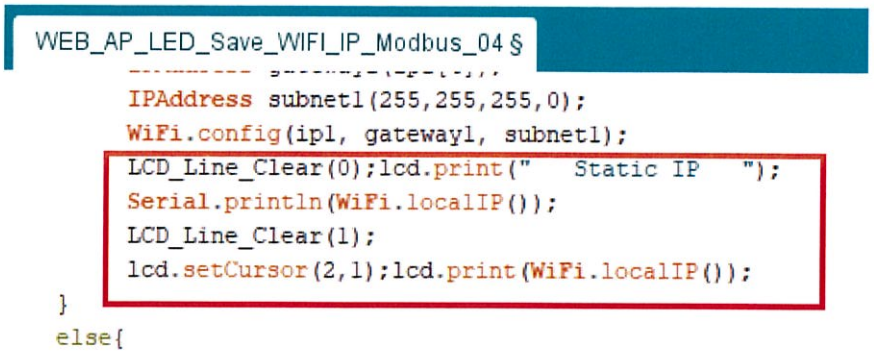
```
server.begin();
Serial.println("\nConnecting to WiFi");
LCD_Line_Clear(0);lcd.print("Connecting WIFI");
LCD_Line_Clear(1);lcd.print("   Wait ");

digitalWrite(2, 0);
while (WiFi.status() != WL_CONNECTED) {
```

ภาพที่ 3.36 โค้ดคำสั่งสำหรับ LCD เพื่อแสดงผลการเชื่อมต่อ Station

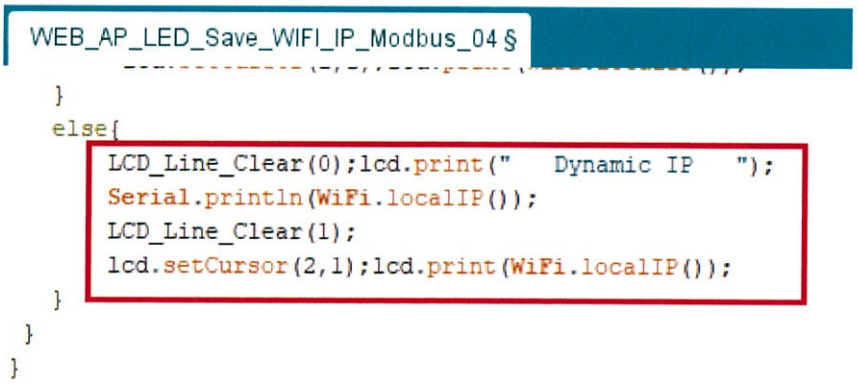
WEB_AP_LED_Save_WIFI_IP_Modbus_04 §

```
digitalWrite(2, !digitalRead(2));
Count60++;
lcd.setCursor(14,1);
lcd.print(Count60);
if ( Count60 >= 60 ) {
  LCD_Line_Clear(0);lcd.print(" Reset Module ");
  LCD_Line_Clear(1);lcd.print("   Wait  ");
  delay(1000);
  ESP.restart(); }
}
```

ภาพที่ 3.37 โค้ดคำสั่งสำหรับ LCD เพื่อตรวจสอบการเชื่อมต่อของ Station



ภาพที่ 3.38 โค้ดคำสั่งสำหรับ LCD เพื่อแสดง Static IP



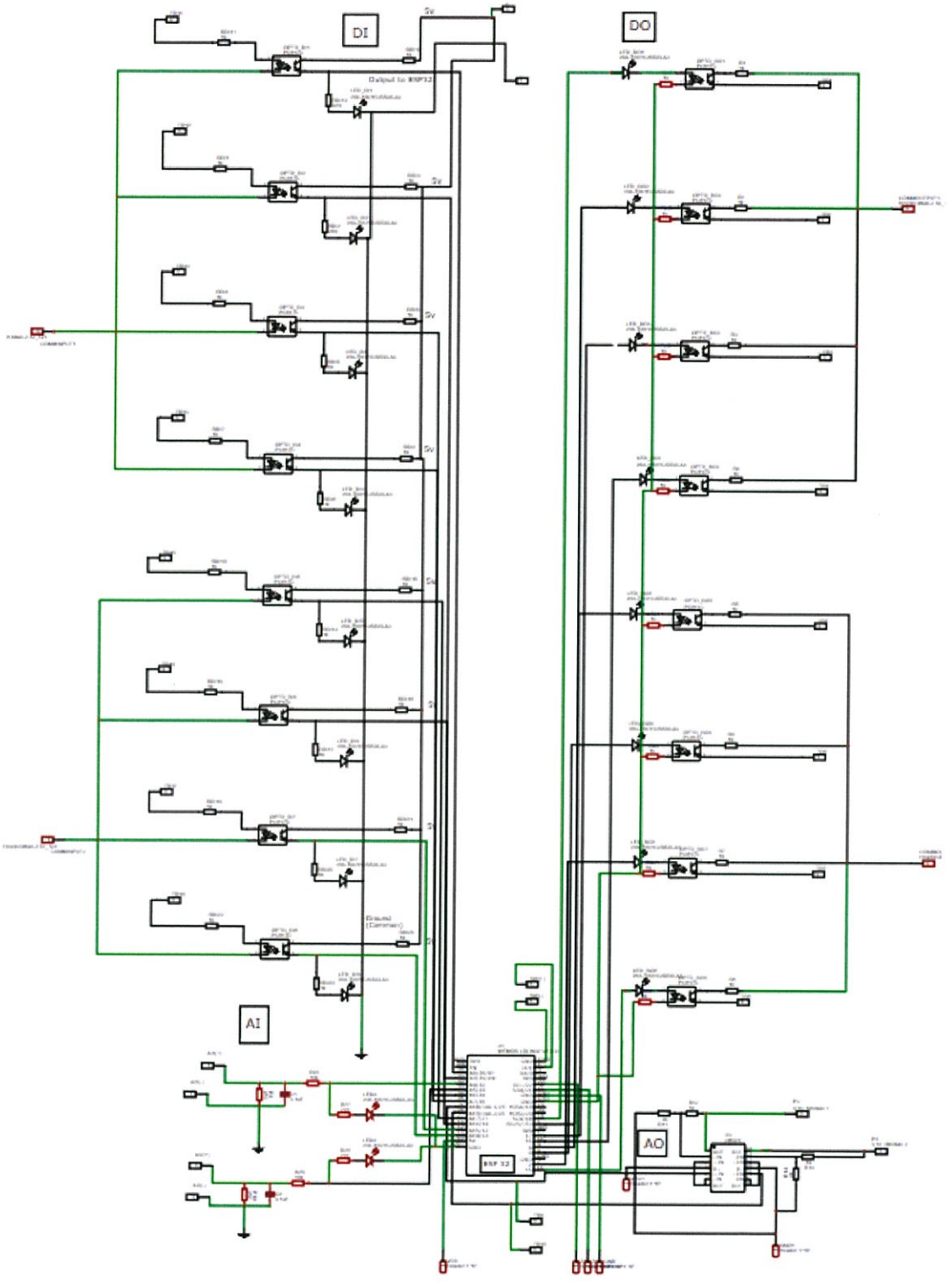
ภาพที่ 3.39 โค้ดคำสั่งสำหรับ LCD เพื่อแสดง Dynamic IP

3.5 โครงสร้างโดยรวมของชุดต้นแบบ

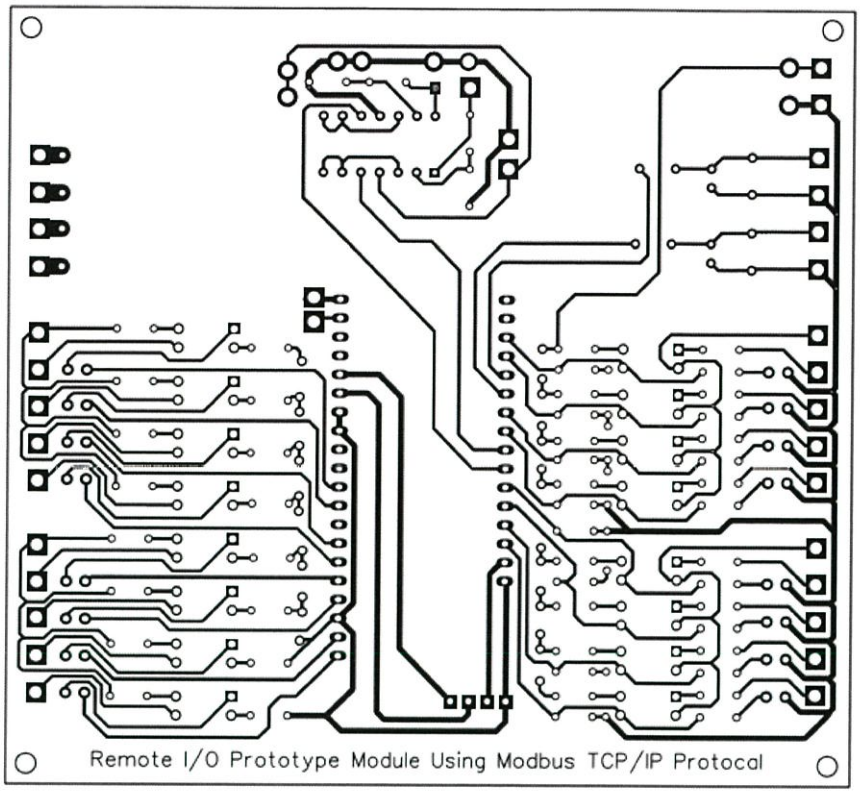
ชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพีไอพีประกอบไปด้วยส่วนต่าง ๆ ที่ได้กล่าวไปข้างต้น ดังนั้นในหัวข้อนี้จะรวมส่วนประกอบต่าง ๆ รวมไปถึงคุณสมบัติของชุดต้นแบบของโมดูลอินพุทเอาท์พุท

3.5.1 วงจรอิเล็กทรอนิกส์ภายในชุดต้นแบบ

วงจรอิเล็กทรอนิกส์ภายในชุดต้นแบบประกอบไปด้วย วงจรดิจิตอลอินพุท วงจรดิจิตอลเอาท์พุท วงจรอะนาล็อกอินพุท วงจรอะนาล็อกเอาท์พุท วงจรสวิทชิ่งเรกูเรเตอร์แบบสเต็ปอัพมาประกอบเข้าด้วยกัน ดังภาพที่ 3.40 และแสดงลายวงจรในรูปแบบ PCB ดังภาพที่ 3.41



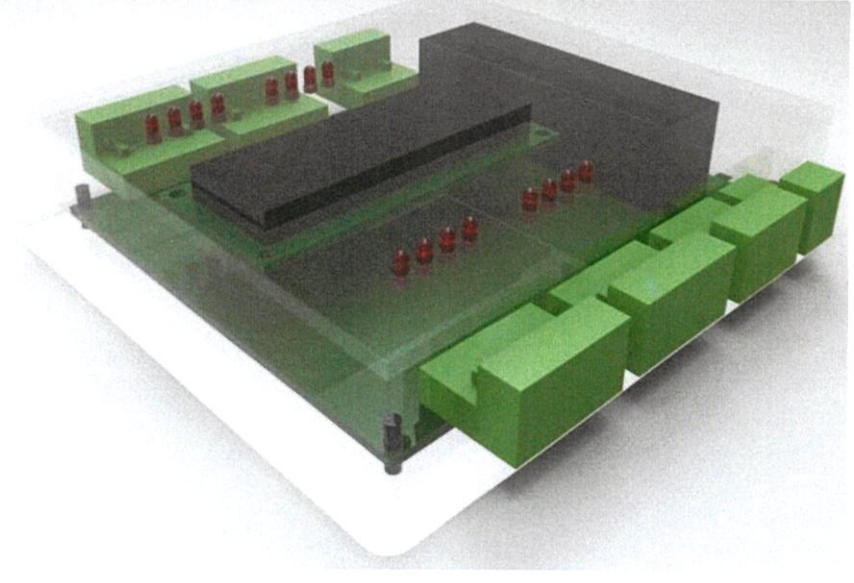
ภาพที่ 3.40 ภาพรวมวงจรอิเล็กทรอนิกส์ของชุดต้นแบบโมดูลอินพุทเอาท์พุท



ภาพที่ 3.41 ลายทองแดงของชุดต้นแบบของโมดูลอินพุทเอาท์พุท

3.5.2 การวางอุปกรณ์บนชุดต้นแบบของโมดูลอินพุทเอาท์พุท

การวางอุปกรณ์บนชุดต้นแบบได้ถูกออกแบบและจัดทำมาในรูปแบบสามมิติ ดังภาพที่ 3.42 การออกแบบมีจุดประสงค์เพื่อให้ง่ายและสะดวกต่อการใช้งาน โดยทางฝั่งด้านซ้ายเป็นด้านของอินพุทรวมไปถึงช่องสำหรับแหล่งจ่ายพลังงาน ส่วนฝั่งด้านขวาเป็นด้านของเอาท์พุท

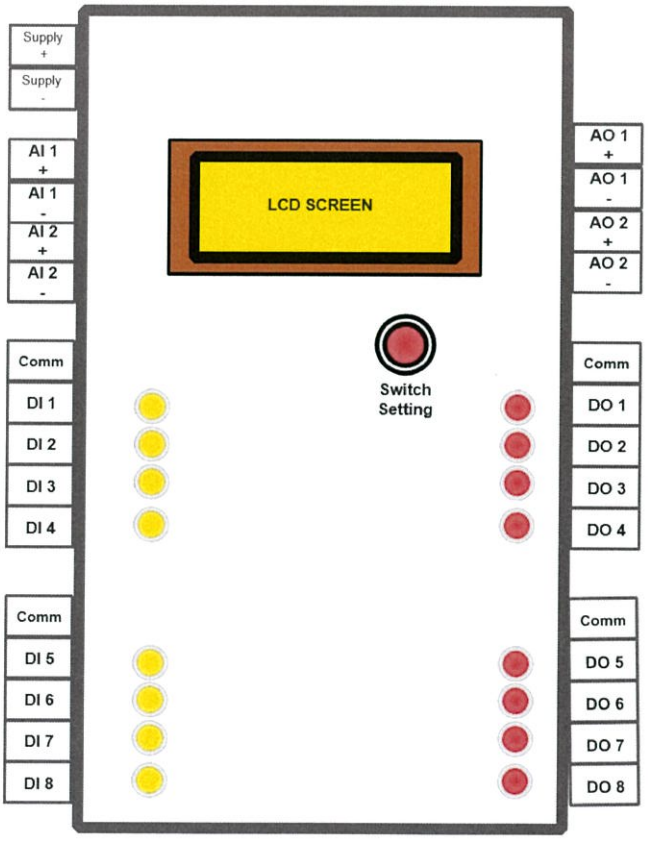


ภาพที่ 3.42 ตำแหน่งของอุปกรณ์ต่าง ๆ บนชุดต้นแบบในรูปแบบสามมิติ

3.5.3 คุณสมบัติภายในของชุดต้นแบบโมดูลอินพุทเอาท์พุท

เนื่องจากชุดต้นแบบโมดูลอินพุทเอาท์พุทมีคุณสมบัติภายในที่สามารถนำไปใช้งานได้หลากหลายตามการใช้งาน โดยมีคุณสมบัติดังนี้

- ดิจิตอลอินพุทและเอาท์พุทจำนวน 8 ช่อง
- อะนาล็อกอินพุทและเอาท์พุทจำนวน 2 ช่อง
- รองรับแหล่งจ่ายพลังงานได้สูงสุด 12 โวลต์
- อะนาล็อกอินพุทรับกระแสได้ 0-22 mA
- อะนาล็อกเอาท์พุทจ่ายกระแสได้ 0-20 mA
- ดิจิตอลอินพุททำงานที่ 3.3 V ถึง 24 V
- ดิจิตอลเอาท์พุททำงานที่ 3.3 V ถึง 24 V
- หลอดแอลอีดีแสดงสถานะดิจิตอล
- จอ LCD แสดงสถานการณ์เชื่อมต่อ
- รีเซ็ทตัวเองเมื่อไม่มีการเชื่อมต่อ
- มีสวิตซ์สำหรับตั้งค่าชุดต้นแบบ
- ตั้งค่าเน็ตเวิร์คผ่านโทรศัพท์มือถือ

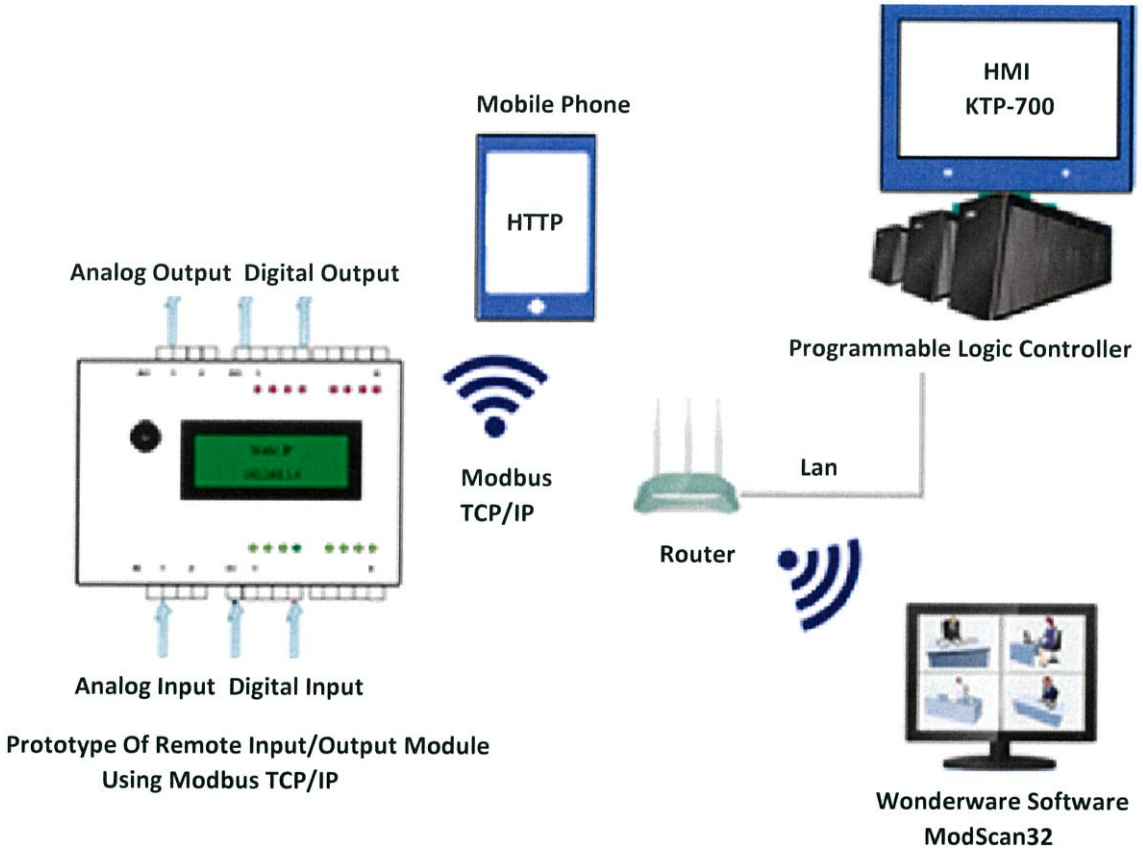


ภาพที่ 3.43 แผนผังและตำแหน่งอุปกรณ์บนชุดต้น

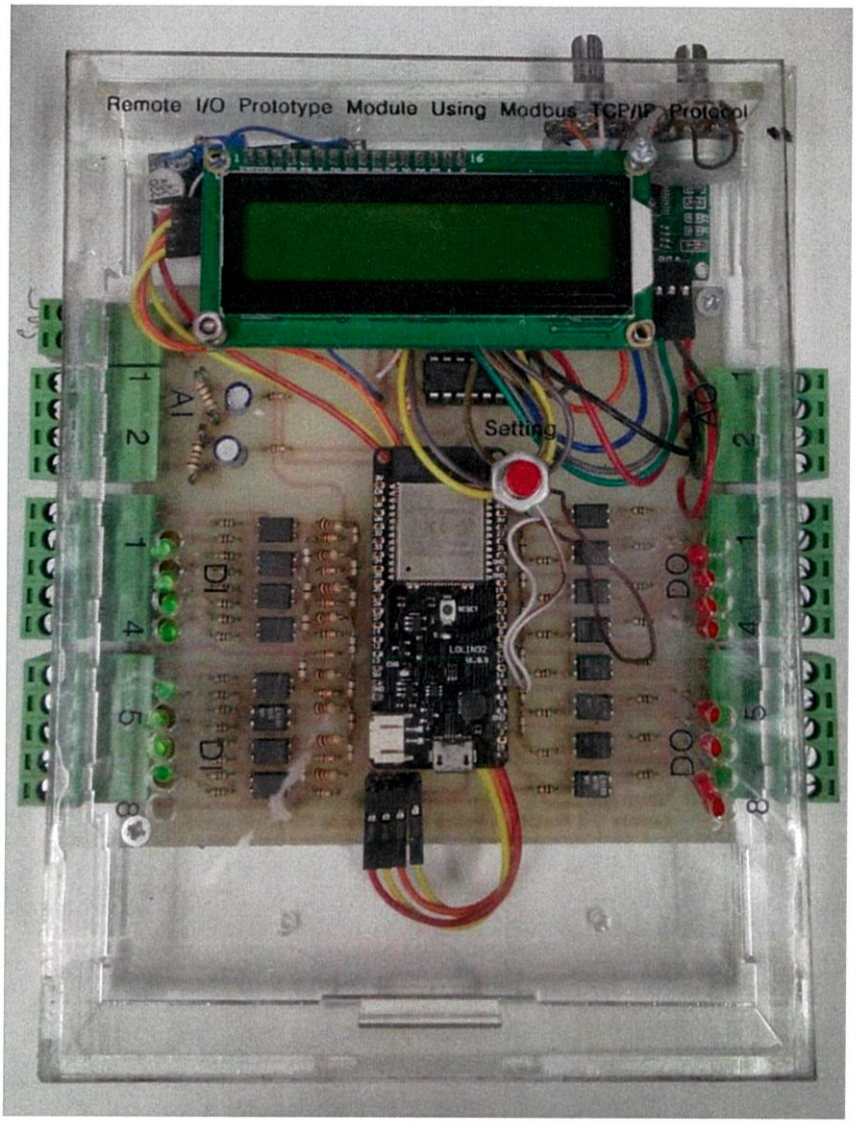
บทที่ 4
ผลการดำเนินงาน

4.1 กล่าวนำ

จากบทที่ 3 ได้มีการกล่าวถึงขั้นตอนการดำเนินงานต่าง ๆ เพื่อทำการสร้างชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัส สำหรับบทนี้จะกล่าวถึงการทดสอบชุดต้นแบบ ในโหมดตั้งค่าเน็ตเวิร์ค ทดสอบในโหมดการทำงานปกติ ทดสอบร่วมกับโปรแกรม ModScan โปรแกรม TIA Portal และ WONDERWARE InTouch ทั้งในส่วนของฮาร์ดแวร์ ส่วนของการรับส่งข้อมูล และส่วนของหน้าจอแสดงผล ซึ่งในแต่ละส่วนจะมีความเกี่ยวข้องและเชื่อมโยงกัน สำหรับการทดลองชุดต้นแบบ มีการทดลองตามโครงสร้างการเชื่อมต่อดังภาพที่ 4.1



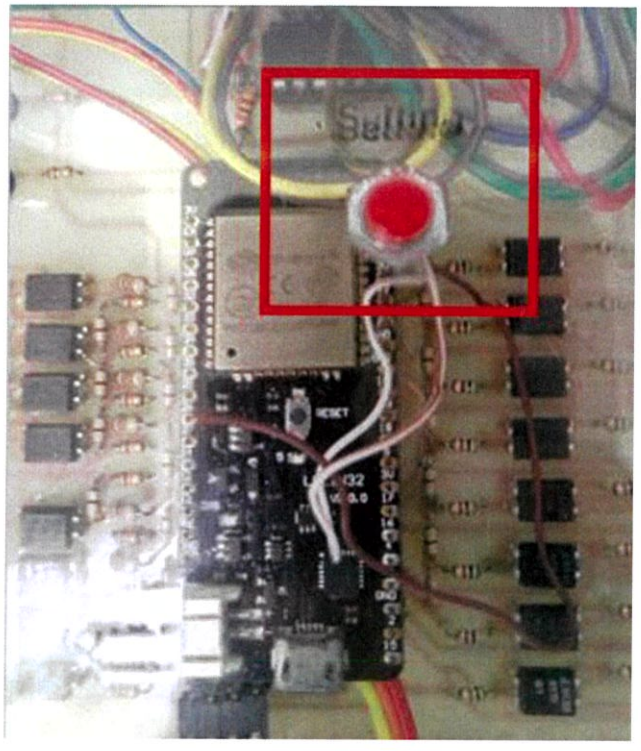
ภาพที่ 4.1 โครงสร้างการเชื่อมต่อชุดต้นแบบกับอุปกรณ์ทดลองและโปรแกรมทดสอบ



ภาพที่ 4.2 ชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัส

4.2 การตั้งค่าเน็ตเวิร์คผ่าน HTTP

สำหรับการเริ่มต้นตั้งค่าโดยการจ่ายไฟเข้าชุดต้นแบบและกดปุ่ม Setting ดังภาพที่ 4.3 และเมื่อกดปุ่มแล้ว จะแสดงการ Setting ผ่านทางหน้าจอ LCD ตามภาพที่ 4.4 - 4.8 แล้วจึงไปตั้งค่าเน็ตเวิร์คผ่าน HTTP โดยการนำอุปกรณ์ที่สามารถเชื่อมต่อ WiFi ได้ เชื่อมต่อกับ WiFi ที่มีชื่อว่า MODBUS_TCP_WIFI ดังภาพที่ 4.9 แล้วเข้าไปตั้งค่าเน็ตเวิร์คที่ต้องการผ่านหน้า Web Server สำหรับการตั้งค่าจะมีการระบุ SSID Password และการตั้งค่า IP Address ที่ต้องการ ดังภาพที่ 4.10

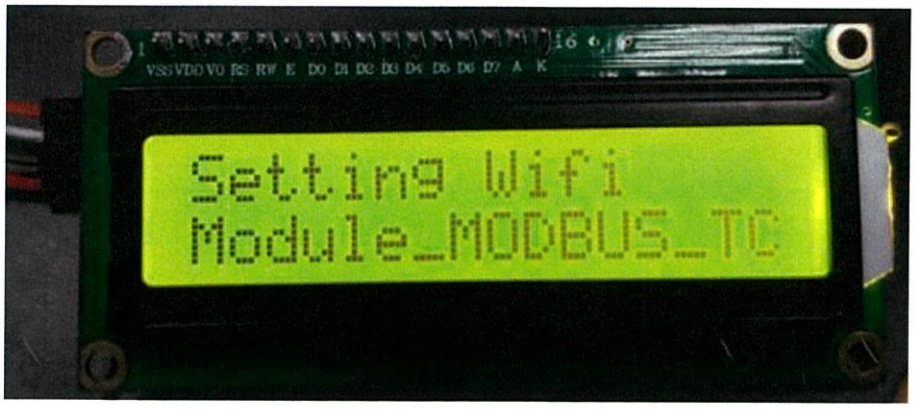


ภาพที่ 4.3 ปุ่ม Setting บนชุดต้นแบบของโมดูลอินพุทเอาท์พุท

เมื่อเปิดเครื่องครั้งแรกหน้าจอจะแสดงข้อความให้กด Setting ดังภาพที่ 4.4

ภาพที่ 4.4 หน้าจอ LCD แสดงการใช้งานตั้งค่าผ่านทางสวิตช์ภายนอก

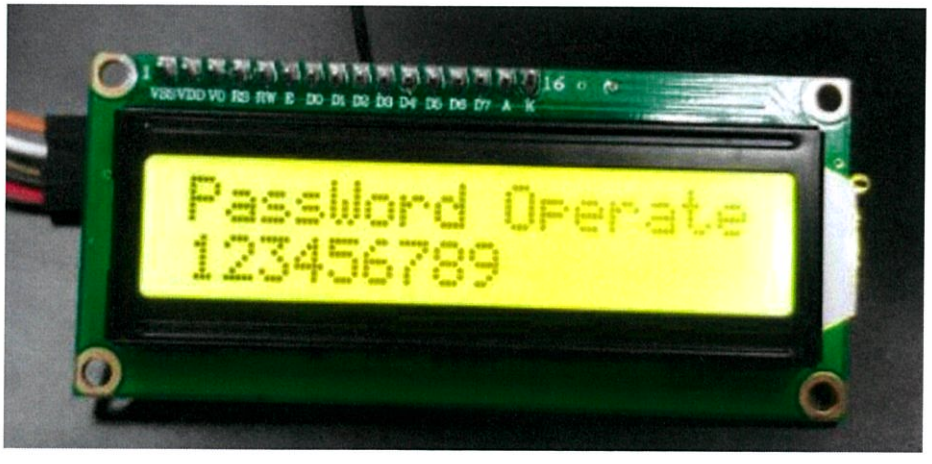
เมื่อกดปุ่ม Setting หน้าจอ LCD แสดงชื่อชุดต้นแบบที่มีชื่อว่า Module_MODBUS_TCP เพื่อเข้าสู่โหมดการตั้งค่าเน็ตเวิร์ค ดังภาพที่ 4.5



ภาพที่ 4.5 หน้าจอ LCD แสดงการตั้งค่า WiFI

แสดงชื่อของสัญญาณ WiFi ที่กระจายออกจากชุดต้นแบบ แสดง Password และแสดง IP Address ดังภาพที่ 4.6 – 4.8

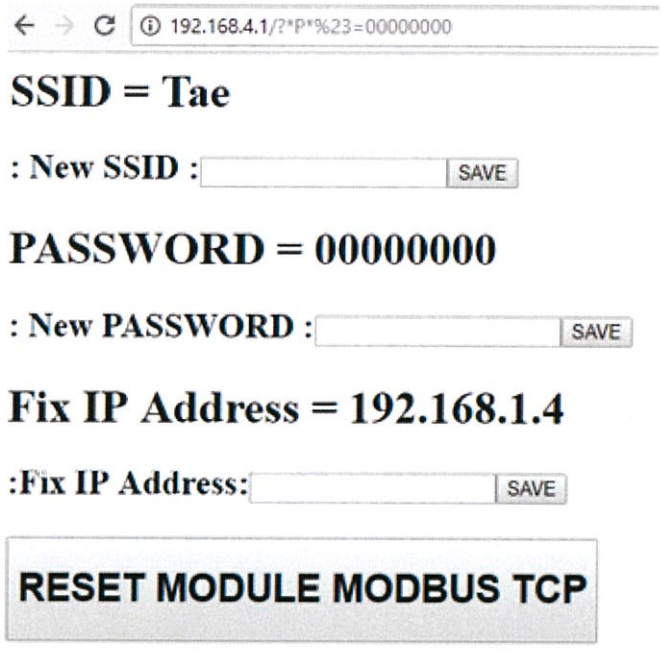
ภาพที่ 4.6 หน้าจอ LCD แสดง SSID



ภาพที่ 4.7 หน้าจอ LCD แสดง Password

ภาพที่ 4.8 หน้าจอ LCD แสดง IP Address

ภาพที่ 4.9 การเลือก WiFi ชื่อ MODBUS_TCP_WIFI เพื่อตั้งค่าโดยผ่านโทรศัพท์มือถือ



ภาพที่ 4.10 หน้าต่างแสดงการตั้งค่า SSID Password และ IP Address ผ่านทาง HTTP

จากภาพที่ 4.10 สามารถอธิบายได้แต่ละบรรทัดดังนี้

1. SSID คือ ชื่อของ WiFi ที่ต้องการจะเชื่อมต่อผ่านเน็ตเวิร์ค
2. Password คือ รหัสผ่านของ WiFi ที่ต้องการจะเชื่อมต่อผ่านเน็ตเวิร์ค
3. IP Address คือ ตำแหน่งของ WiFi ที่ต้องการเชื่อมต่อ
   แบ่งออกเป็น 2 รูปแบบ คือ
   - Static IP คือ ตำแหน่งที่ต้องการเชื่อมต่อ สามารถกรอก IP ที่ต้องการเชื่อมต่อได้
   - Dynamic IP คือ ตำแหน่ง WiFi เดิมที่ใช้งาน โดยไม่จำเป็นต้องกรอก IP หากต้องการใช้ตำแหน่งแบบ Dynamic

4.2 ผลการทดลองส่วนฮาร์ดแวร์

4.2.1 การทดสอบการรับค่าดิจิตอลอินพุท

การทดลองต่อไปนี้จะแสดงการรับค่าดิจิตอลอินพุทเข้ามาทางช่องดิจิตอลอินพุทซึ่งได้ทำการทดลองในแต่ละช่องจำนวน 4 ครั้งของดิจิตอลอินพุท (DI) ดังตารางต่อไปนี้

ตารางที่ 4.1 การทดสอบการรับค่าดิจิตอลอินพุท

	DI 1	DI 2	DI 3	DI 4	DI 5	DI 6	DI 7	DI 8
ครั้งที่ 1	0	1	0	1	0	1	0	1
ครั้งที่ 2	1	0	1	0	1	0	1	0
ครั้งที่ 3	1	1	1	1	1	1	1	1
ครั้งที่ 4	0	0	0	0	0	0	0	0

จากตารางที่ 4.1 เป็นการรับค่าดิจิตอลอินพุทเข้ามาทางช่องดิจิตอลอินพุทในแต่ละช่องโดย หมายเลข 0 คือ สถานะ LOW และหมายเลข 1 คือ สถานะ HIGH และหลอดไฟแอลอีดีแสดงไฟสีเขียวเมื่อมีสถานะ HIGH โดยได้ผลการทดลองเป็นไปตามที่ต้องการ

4.2.2 การทดสอบการส่งค่าดิจิตอลเอาท์พุท

การทดลองต่อไปนี้จะแสดงการส่งค่าดิจิตอลเอาท์พุทจากโปรแกรม ModScan ซึ่งได้ทำการทดลองในแต่ละช่องจำนวน 4 ครั้งของดิจิตอลเอาท์พุท (DO) ดังตารางต่อไปนี้

ตารางที่ 4.2 การทดสอบการจ่ายดิจิตอลเอาท์พุท

	DO 1	DO 2	DO 3	DO4	DO 5	DO 6	DO 7	DO 8
ครั้งที่ 1	0	1	0	1	0	1	0	1
ครั้งที่ 2	1	0	1	0	1	0	1	0
ครั้งที่ 3	1	1	1	1	1	1	1	1
ครั้งที่ 4	0	0	0	0	0	0	0	0

จากตารางที่ 4.2 เป็นการส่งค่าดิจิตอลเอาท์พุทออกไปทางช่องดิจิตอลเอาท์พุทจากการสั่งผ่าน ModScan ในแต่ละช่องโดย หมายเลข 0 คือ สถานะ LOW และหมายเลข 1 คือ สถานะ HIGH และหลอดไฟแอลอีดีแสดงไฟสีแดงเมื่อมีสถานะ HIGH โดยได้ผลการทดลองเป็นไปตามที่ต้องการ

4.2.3 การทดสอบการรับค่าอะนาล็อกอินพุท

การทดลองทางด้านของอะนาล็อกอินพุทเป็นการทดลองในการรับค่ากระแสไฟฟ้าจากเครื่องจ่ายกระแสหรือมิเตอร์ เป็นจำนวน 6 ครั้ง ดังตารางต่อไปนี้

ตารางที่ 4.3 การทดสอบการรับค่าอะนาล็อกอินพุท

	AI 1	AI 2
ครั้งที่ 1	20 mA	0 mA
ครั้งที่ 2	16 mA	4 mA
ครั้งที่ 3	12 mA	8 mA
ครั้งที่ 4	8 mA	12 mA
ครั้งที่ 5	4 mA	16 mA
ครั้งที่ 6	0 mA	20 mA

จากภาพที่ 4.3 เป็นการทดสอบการรับค่าอะนาล็อกเข้ามาทางช่องอะนาล็อกอินพุทในแต่ละขา โดยการรับกระแสไฟจากแหล่งจ่าย

4.2.4 การทดสอบการส่งค่าอะนาล็อกอินพุท

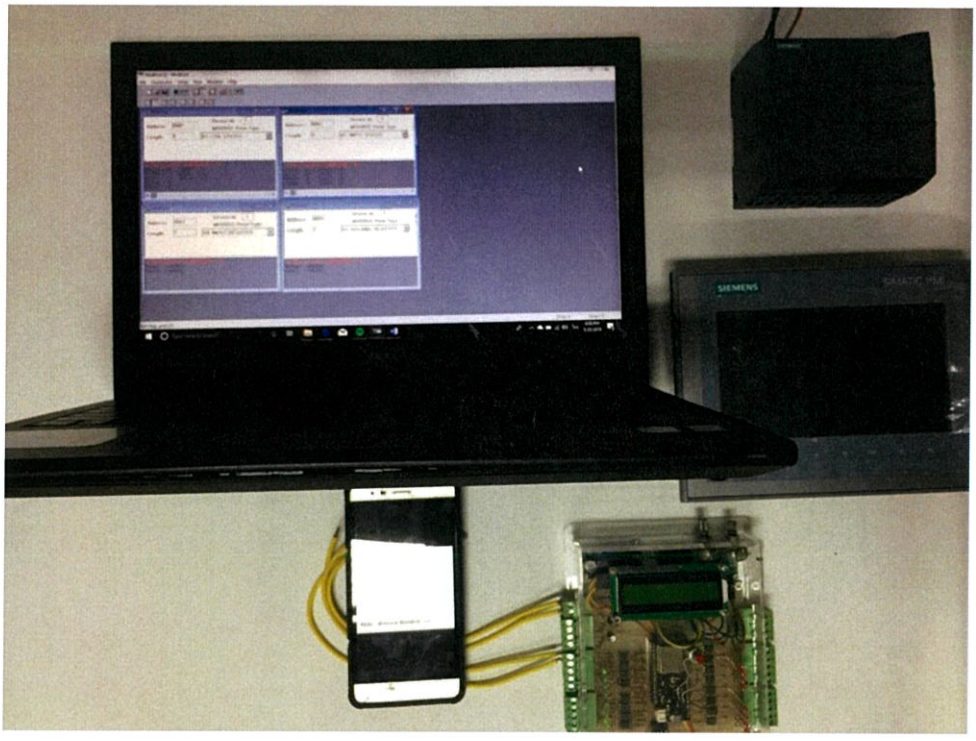
การทดลองทางด้านของอะนาล็อกเอาท์พุทเป็นการทดลองในการส่งค่าอะนาล็อกเอาท์พุทผ่านโปรแกรม ModScan เป็นจำนวน 6 ครั้ง ดังตารางต่อไปนี้

ตารางที่ 4.4 การทดสอบการส่งค่าอะนาล็อกเอาท์พุท

	AO 1	AO 2
ครั้งที่ 1	0 mA	20 mA
ครั้งที่ 2	4 mA	16 mA
ครั้งที่ 3	8 mA	12 mA
ครั้งที่ 4	12 mA	8 mA
ครั้งที่ 5	16 mA	4 mA
ครั้งที่ 6	20 mA	0 mA

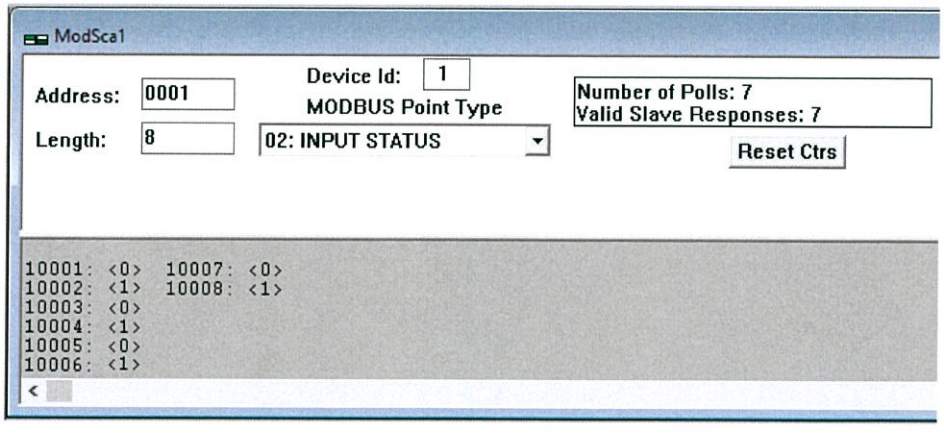
จากตารางที่ 4.4 เป็นการทดสอบการส่งค่าอะนาล็อกเอาท์พุทผ่านโปรแกรม ModScan เข้ามาทางช่องอะนาล็อกเอาท์พุทในแต่ละช่อง

4.3 ผลการทดลองส่วนของการรับส่งข้อมูลโดยใช้โปรแกรม ModScan



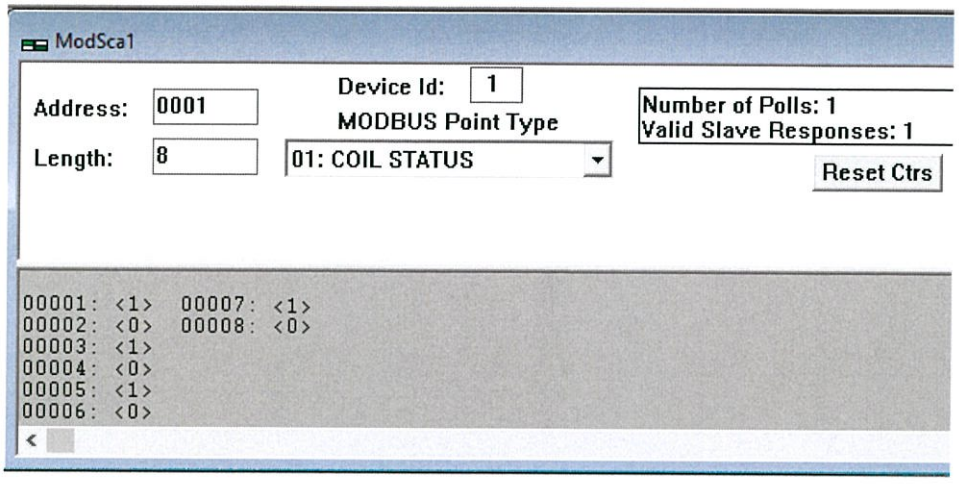
ภาพที่ 4.11 โครงสร้างของอุปกรณ์การเชื่อมต่อโดยรวมที่ใช้ในการทดสอบชุดต้นแบบ

สำหรับผลการทดลองส่วนของการรับและส่งข้อมูลโดยใช้โปรแกรม ModScan โดยการรับหรือส่งค่าได้ตามตารางในหัวข้อ 4.2 และแสดงส่วนของผลการทดลองดังนี้



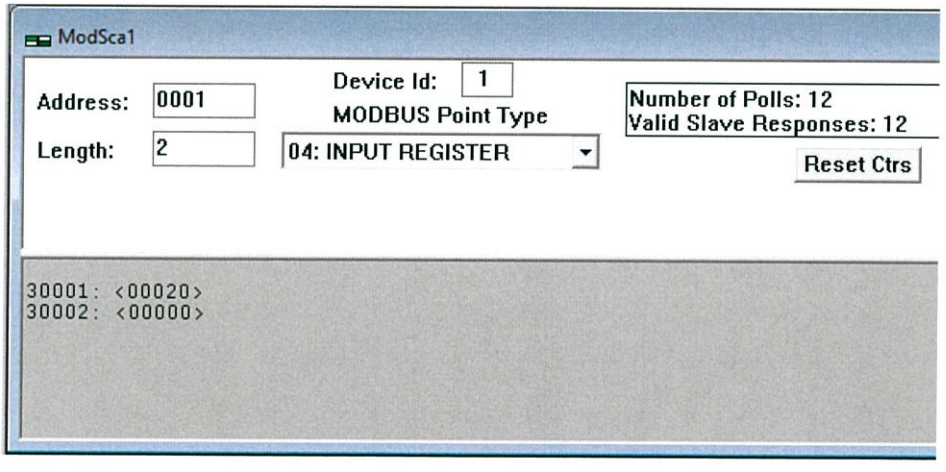
ภาพที่ 4.12 ภาพการอ่านค่าดิจิตอลอินพุทด้วยโปรแกรม ModScan

จากตารางที่ 4.1 เมื่อนำมาตรวจสอบการรับค่าดิจิตอลอินพุทผ่านทาง ModScan จะได้ผลการทดลองดังภาพที่ 4.12



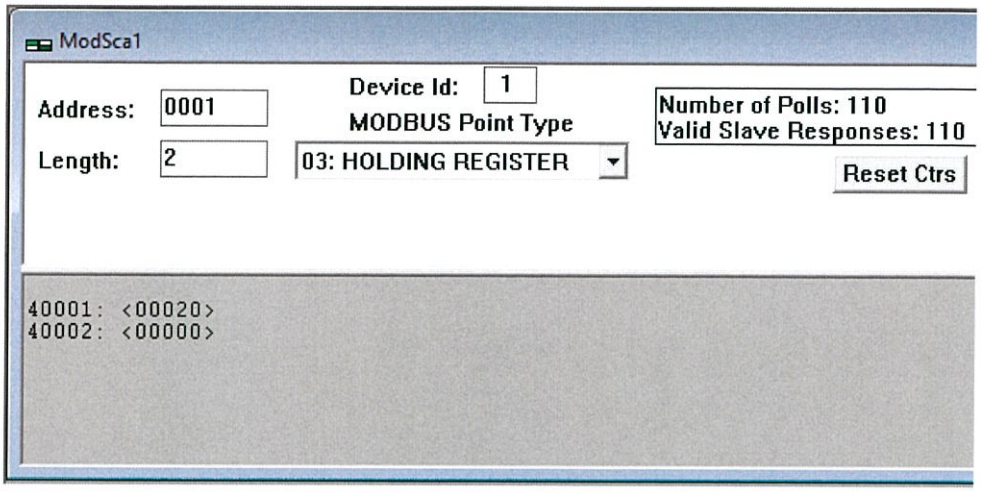
ภาพที่ 4.13 ภาพการส่งค่าดิจิตอลเอาท์พุทด้วยโปรแกรม ModScan

จากภาพที่ 4.13 เมื่อส่งค่าดิจิตอลเอาท์พุทผ่านทาง ModScan จะได้ผลตามตารางการทดสอบที่ 4.2



ภาพที่ 4.14 ภาพการอ่านอะนาล็อกอินพุทด้วยโปรแกรม ModScan

จากตารางที่ 4.3 เมื่อนำมาตรวจสอบการอ่านกระแสอะนาล็อกอินพุทผ่านทาง ModScan จะได้ผลการทดลองดังภาพที่ 4.14

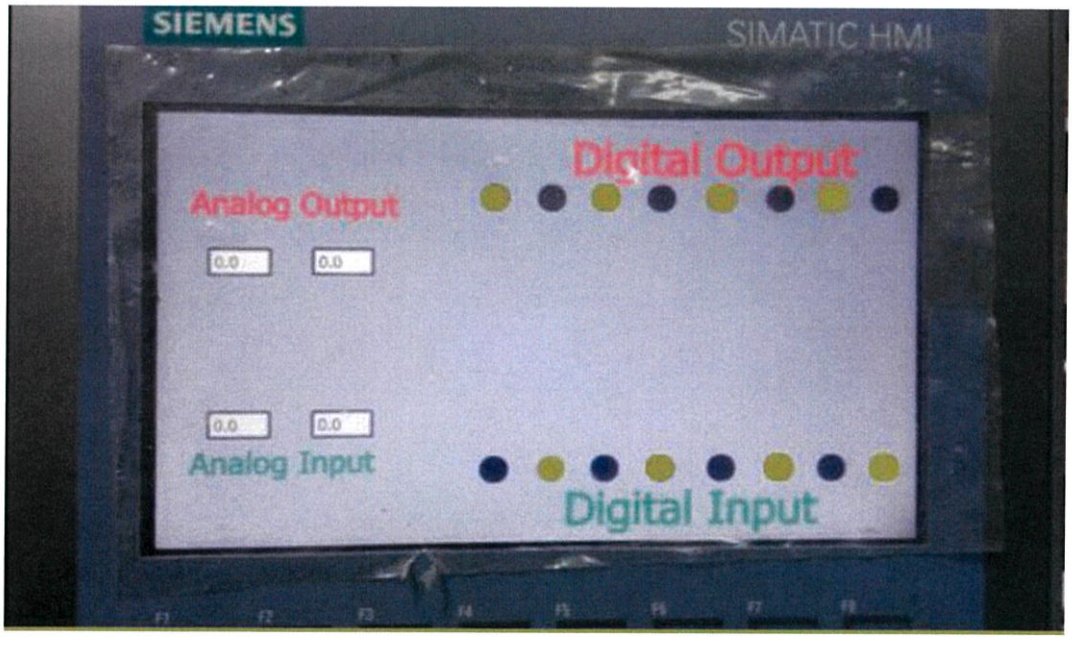


ภาพที่ 4.15 ภาพการส่งค่าอะนาล็อกเอาท์พุทด้วยโปรแกรม ModScan

จากภาพที่ 4.15 เมื่อส่งค่าอะนาล็อกเอาท์พุทผ่านทาง ModScan จะได้ผลตามตารางการทดสอบที่ 4.4

4.4 การทดลองการเชื่อมต่อกับ PLC แสดงผลผ่านหน้าจอ HMI

สำหรับผลการทดลองในการเชื่อมต่อกับ PLC จะสามารถแสดงผลได้ผ่านทางหน้าจอ HMI ยี่ห้อ Siemens รุ่น KTP-700 ซึ่งผู้จัดทำได้ออกแบบหน้าจอให้คล้ายคลึงกับชุดต้นแบบเพื่อให้ง่ายต่อสังเกตุและทำความเข้าใจ

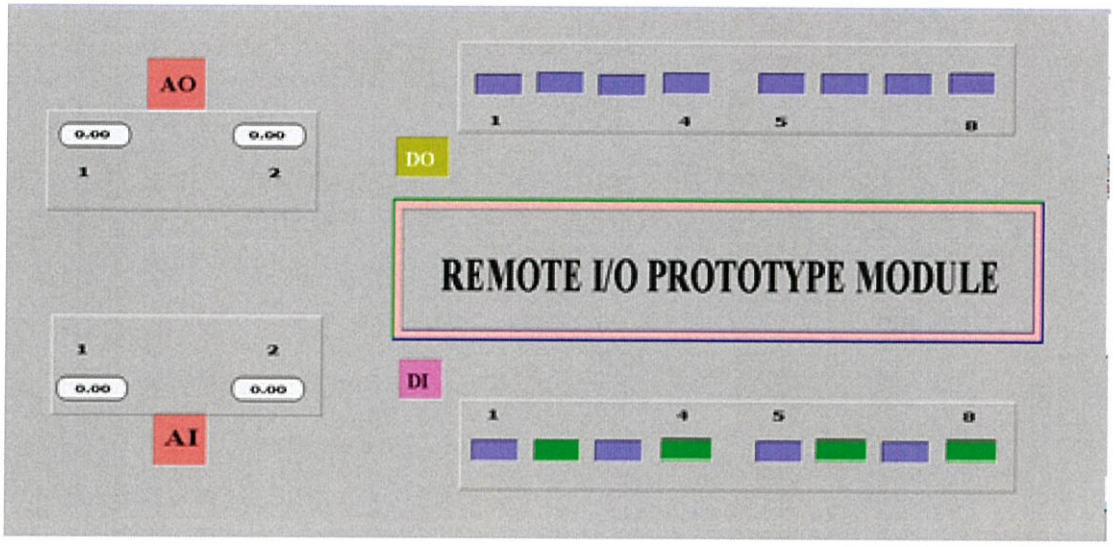


ภาพที่ 4.16 การแสดงผลการเชื่อมต่อกับ PLC ผ่านหน้าจอ HMI ยี่ห้อ Siemens รุ่น KTP-700

จากหัวข้อ 4.2 เมื่อนำมาทดสอบการเชื่อมต่อ PLC และแสดงผลผ่านหน้าจอ HMI จะได้ผลดังภาพที่ 4.16

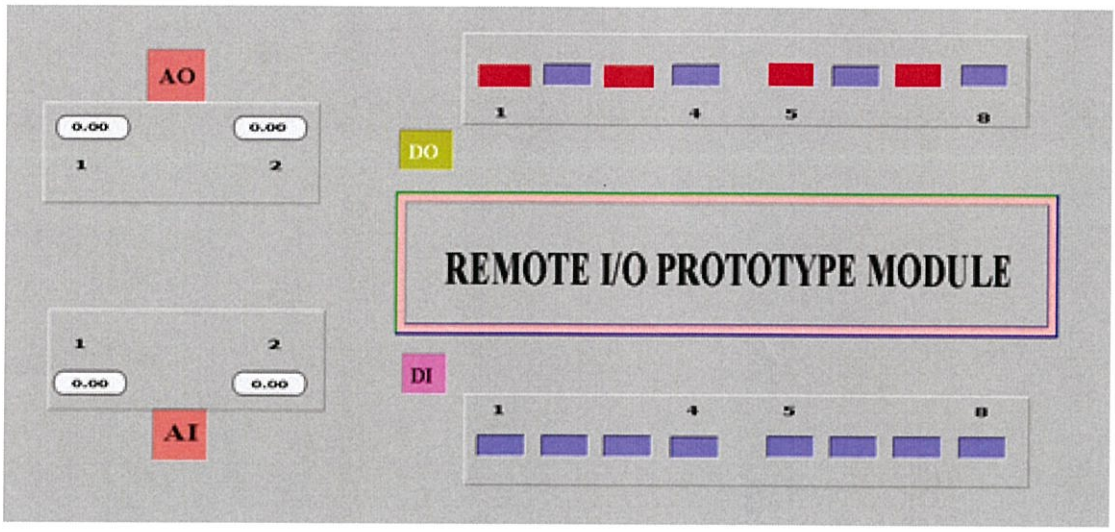
4.5 การแสดงผลผ่านทางหน้าจอของซอฟต์แวร์ WONDERWARE

สำหรับการทดลองการเชื่อมต่อผ่านซอฟต์แวร์ WONDERWARE ผู้จัดทำได้ออกแบบหน้าต่างแสดงผลให้มีรูปแบบคล้ายคลึงกับกล่องชุดต้นแบบของโมดูลอินพุทเอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี เพื่อให้ง่ายต่อการสังเกตและจัดการ



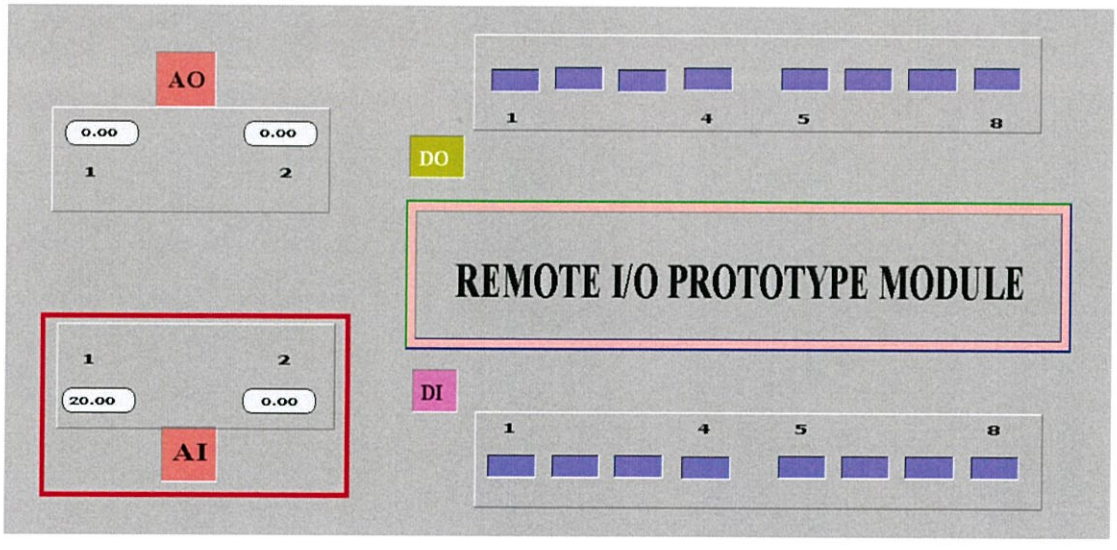
ภาพที่ 4.17 ภาพการรับค่าดิจิตอลอินพุท

ผลการทดลองดังภาพที่ 4.17 แสดงผลการทดลองของตารางที่ 4.1 ของฝั่งดิจิตอลอินพุทบนชุดต้นแบบ และแสดงผ่านซอฟต์แวร์ โดยในช่องดิจิตอลอินพุทมีสถานะเริ่มต้นเป็นค่า 0 และไฟแสดงสถานะเริ่มต้นเป็นสีเขียว ซึ่งค่าที่รับเข้ามาเป็น 1 จึงแสดงเป็นไฟสีเขียว และสามารถแสดงบนซอฟต์แวร์ WONDERWARE InTouch



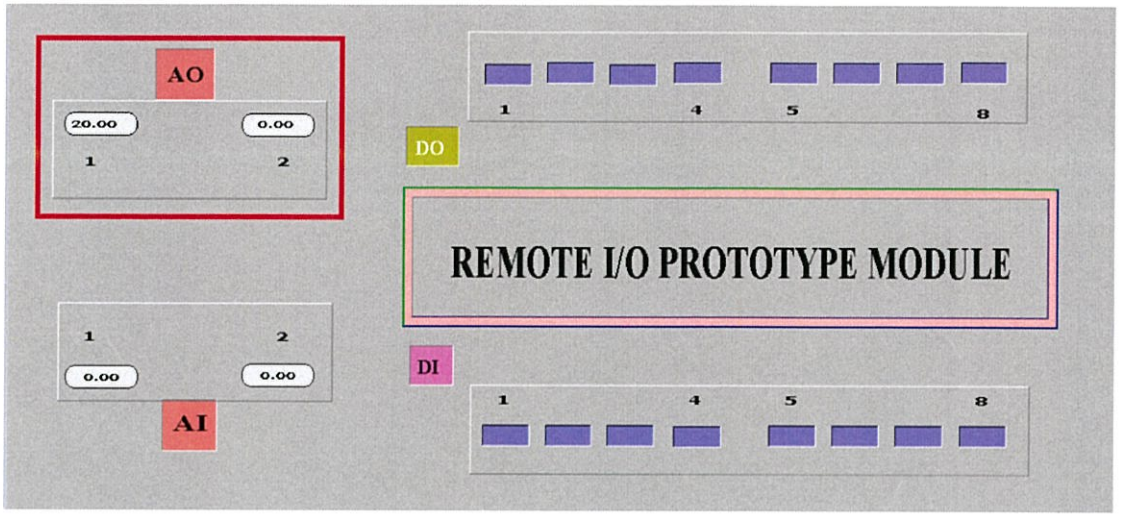
ภาพที่ 4.18 ภาพการส่งค่าดิจิตอลเอาท์พุท

ภาพที่ 4.18 แสดงผลการทดลองของตารางที่ 4.2 แสดงผ่านซอฟต์แวร์ โดยในช่องดิจิตอลเอาท์พุทมีสถานะเริ่มต้นเป็นค่า 0 และไฟแสดงสถานะเริ่มต้นเป็นสีฟ้า ซึ่งค่าที่ส่งออกไปเป็น 1 จึงแสดงเป็นไฟสีแดง และสามารถแสดงบนซอฟต์แวร์ WONDERWARE InTouch



ภาพที่ 4.19 ภาพการรับค่าอะนาล็อกอินพุท

ผลการทดลองดังภาพที่ 4.19 เป็นการแสดงการรับค่าอะนาล็อกอินพุทจากการทดลองของตารางที่ 4.3 ซึ่งแสดงบน WONDERWARE InTouch โดยช่องอะนาล็อกอินพุทตัวที่ 1 แสดงผลเป็น 20 mA ตรงตามค่าที่จ่ายเข้ามา



ภาพที่ 4.20 ภาพการส่งค่าอะนาล็อกเอาท์พุท

ผลการทดลองดังภาพที่ 4.20 เป็นการแสดงการส่งค่าอะนาล็อกเอาท์พุทจากการทดลองของตารางที่ 4.4 ซึ่งแสดงบน WONDERWARE InTouch โดยช่องอะนาล็อกอินพุทตัวที่ 1 แสดงผลเป็น 20 mA ตรงตามค่าที่จ่ายเข้ามา

4.5 ผลการทดลองรวมของอุปกรณ์และซอฟต์แวร์ที่เชื่อมต่อกับชุดต้นแบบ

ผลการทดลองในส่วนโปรแกรม ModScan การเชื่อมต่อกับ PLC แสดงผลผ่านหน้าจอ HMI และ WONDERWARE InTouch ได้ทำการทดลองตามหัวข้อที่ 4.2 เป็นจำนวนทั้งสิ้น 6 ครั้ง ทั้งในฝั่งของ ดิจิตอลอินพุท (DI) ดิจิตอลเอาท์พุท (DO) อะนาล็อกอินพุท (AI) และอะนาล็อกเอาท์พุท (AO) แสดงดังตารางต่อไปนี้

ตารางที่ 4.5 ผลการทดลองรวมในส่วนของอุปกรณ์และซอฟต์แวร์ที่เชื่อมต่อกับชุดต้นแบบ

ครั้งที่	ModScan						หน้าจอ HMI						WONDERWARE					
	DI	DO	AI 1	AI 2	AO 1	AO 2	DI	DO	AI 1	AI 2	AO 1	AO 2	DI	DO	AI 1	AI 2	AO 1	AO 2
1	✓	✓	20	0	0	20	✓	✓	20	0	0	20	✓	✓	20	0	0	20
2	✓	✓	16	4	4	16	✓	✓	16	4	4	16	✓	✓	16	4	4	16
3	✓	✓	12	8	8	12	✓	✓	12	8	8	12	✓	✓	12	8	8	12
4	✓	✓	8	12	12	8	✓	✓	8	12	12	8	✓	✓	8	12	12	8
5	-	-	4	16	16	4	-	-	4	16	16	4	-	-	4	16	16	4
6	-	-	0	20	20	0	-	-	0	20	20	0	-	-	0	20	20	0

บทที่ 5
สรุปผล ปัญหา และข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

จากการดำเนินงานในเรื่องของการสร้างชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพีสามารถรับส่งข้อมูลผ่านโปรแกรม ModScan แสดงการเชื่อมต่อกับ PLC ผ่านหน้าจอ HMI และแสดงการทำงานของอุปกรณ์ผ่านโปรแกรม WONDERWARE โดยส่วนของการแสดงผลจะมีค่าที่ต้องการของโมดูล เช่น ดิจิตอลอินพุท ดิจิตอลเอาท์พุท อะนาล็อกอินพุท และอะนาล็อกเอาท์พุท ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพีนี้ ได้ทำการสร้างส่วนการรับส่งข้อมูลโดยได้ทดสอบฟังก์ชันต่าง ๆ ด้วยโปรแกรม ModScan และสร้างส่วนของการแสดงผลด้วยโปรแกรม Wonderware โดยหลังจากนี้สามารถนำส่วนการรับส่งข้อมูลและส่วนแสดงผลไปทดสอบกับระบบจริงที่ต้องการจะติดตามผล และวิเคราะห์การทำงานของอุปกรณ์รับรู้หรืออุปกรณ์ควบคุม เพื่อนำปรับปรุงให้เป็นไปตามความต้องการของผู้ปฏิบัติงาน

5.2 ปัญหาและวิธีการแก้ไขปัญหา

5.2.1 ปัญหาที่พบ

1. มีข้อจำกัดของการเชื่อมต่อในการใช้งานโปรแกรม ModScan

การใช้งานโปรแกรม ModScan สามารถเชื่อมต่อได้เพียงทีละเครื่องระหว่าง PLC และ ชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพี

2. เครือข่าย WiFi มีความเสถียรต่ำ

เนื่องจากเครือข่าย WiFi ไม่มีความเสถียรเพียงพอจึงทำให้การส่งข้อมูลผ่านโปรโตคอลมอดบัสทีซีพี/ไอพีเกิดความไม่ต่อเนื่อง

5.2.2 วิธีการแก้ไขปัญหา

1. โปรแกรม ModScan เป็นตัวทดลองจึงไม่สามารถเชื่อมต่อได้หลายเครื่อง อันเนื่องมาจากข้อตกลงของโปรแกรม จำเป็นต้องติดตั้งโปรแกรมเวอร์ชั่นเต็ม

2. เลือกใช้เครือข่าย WiFi ที่มีความเสถียรในการเชื่อมต่อ

5.3 ข้อเสนอแนะ

การเชื่อมต่อระหว่างชุดต้นแบบของโมดูลอินพุท/เอาท์พุทแบบรีโมทโดยใช้โปรโตคอลมอดบัสทีซีพี/ไอพีที่ติดตั้งไว้บริเวณพื้นที่ปฏิบัติงานกับ PLC ที่ติดตั้งไว้ที่ห้องควบคุม โดยผ่านโปรโตคอลมอดบัสทีซีพี/ไอพีนั้น มีระยะการเชื่อมต่อที่จำกัดโดยขึ้นอยู่กับระยะทางการส่งสัญญาณของ WiFi ดังนั้นในงานที่มีระยะทางระหว่างพื้นที่ปฏิบัติงานกับห้องควบคุมอยู่ห่างกันเกินกว่าที่สัญญาณ WiFi ครอบคลุมถึง จึงทำให้ไม่สามารถเชื่อมต่อกันได้ ในกรณีที่มีระยะทางห่างกันไกลมากเกินไป อาจจำเป็นต้องใช้อุปกรณ์ในการแปลงสัญญาณจาก WiFi เป็น RS 485 เพื่อเพิ่มระยะทางในการเชื่อมต่อ

เอกสารอ้างอิง

[1] PLC

ที่มา : https://www.keyence.co.th/products/controls/index.jsp

[2] Remote I/O

ที่มา : www.9engineer.com › Webboard › PLC & ระบบคอนโทร

[3] Modbus TCP/IP

ที่มา : ttps://www.rtaautomation.com › Technologies

[4] Input/Output Registor

ที่มา : https://encyclopedia2.thefreedictionary.com/input%2Foutput+register

[5] Wonderware Software

ที่มา : https://www.wonderware.com

[6] HTTP

ที่มา : https://developer.mozilla.org/en-US/docs/Web/HTTP

[7] Arduino

ที่มา : https://www.arduino.cc/en/main/software

[8] TIA Software

ที่มา : https://www.siemens.com/...software/...software/tia-portal/software.html

[9] Microcontroller ESP32

ที่มา : https://www.ioxhop.com/article/63/esp32-เบื้องต้น-บทที่-2-บอร์ดพัฒนา-esp32

ภาคผนวก

```
#include "Arduino.h"
#include <WiFi.h>
#include <driver/dac.h>
#include <Wire.h>
#include "EEPROM.h"
int addr = 0;
#define EEPROM_SIZE 100
#define  SW_Setting   0
#define  LED_ON_Board 16
static bool Connect,Print_NoConnect,Print_Connect;
char Update_LCD;
unsigned int Show_Normal;
bool One_T = 0;
void  setup2();
void  Show_LCD_Mode();
void  Setting_Config_WiFi();
void  Operate_Mode();
int   MODE_LCD = 0;
int   MODE_LCD_Check = 0;
bool  SW1_Positive_Men = 0,SW1_Negtive_Men = 1;
bool  SW1_Positive = 0,    SW1_Negtive = 0;
void  Update_Mode();
void Show_Value_Sensor(int Ch);
//////////////////////////////////////////////
hw_timer_t * timer = NULL;
void IRAM_ATTR onTimer() {
   timerAlarmDisable(timer);
   Update_Mode();
   timerAlarmEnable(timer);
}
////// LCD 20 *2 /////////////////
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x3f, 16, 2);
char buf[20],bu2[20],Bu;
char datestring[20];
```

```
IPAddress WiFi_local_IP(192,168,4,1);
IPAddress gateway(192,168,4,1);
IPAddress subnet(255,255,255,0);
WiFiServer server(80); //  TCP Server 80
char WiFi_SSID[] = "Module_MODBUS" ;
char WiFi_PASS[] = "123456789" ;
char WiFi_SSID_Connect[] = "                " ;
char WiFi_Pass_Connect[] = "                " ;
char WiFi_IP_Connect[]   = "               " ;

void SW_Read() {
    SW1_Positive = 0;SW1_Negtive = 0;
    if ( digitalRead(SW_Setting) == 0 && SW1_Positive_Men == 0 ) {
        SW1_Positive = 1;
        SW1_Positive_Men = 1;
    }
    if ( digitalRead(SW_Setting) == 1 && SW1_Negtive_Men == 1 ) {
        SW1_Negtive = 1;
        SW1_Negtive_Men = 0;
    }
    SW1_Positive_Men = !digitalRead(SW_Setting);
    SW1_Negtive_Men  =  digitalRead(SW_Setting);
}
void LCD_Line_Clear(byte Line) {
  lcd.setCursor(0,Line);
  lcd.print("                ");
  lcd.setCursor(0,Line);
}

void Show_SSID(int Mode) {  // Mode = 0 Show SSID Setting , Mode = SSID Operate
unsigned int i;
byte Len;
  EEPROM.begin(EEPROM_SIZE);
  memset(WiFi_SSID_Connect, 0, sizeof(WiFi_SSID_Connect));
  Len = EEPROM.read(0);
```

```
    if ( Len > 20 ) { Len = 20; }
    for (i = 0 ; i < Len - 1 ; i ++) {
        Bu = EEPROM.read(i+5);
        if ( Bu != 0x20 ) {
            WiFi_SSID_Connect[i] = Bu;
        }
        else{
            WiFi_SSID_Connect[i] = 0;
        }
    }
    if ( Mode == 0 ) {
        LCD_Line_Clear(0);
        lcd.print("SSID for Setting");
        LCD_Line_Clear(1);
        lcd.print(WiFi_SSID_Connect);
    }
    if ( Mode == 1 ) {
        LCD_Line_Clear(0);
        lcd.print("SSID Operate ");
        LCD_Line_Clear(1);
        lcd.print(WiFi_SSID_Connect);
    }
    if ( Mode == 2 ) {
      Serial.println(WiFi_SSID_Connect);
    }
}
void Show_PASS_WORD(int Mode) {
unsigned int i;
byte Len;
   Len = EEPROM.read(3);
   memset(WiFi_Pass_Connect, 0, sizeof(WiFi_Pass_Connect));
   if ( Len > 20 ) { Len = 20; }
   for (i = 0 ; i< Len - 1 ; i ++) {
        Bu = EEPROM.read(i+25);
        if ( Bu != 0x20 ) {
```

```
            WiFi_Pass_Connect[i] = Bu;
          }
          else{
            WiFi_Pass_Connect[i] = 0;
        }
    }
      if ( Mode == 0 ) {
       LCD_Line_Clear(0);
       lcd.print("PassWord Setting");
       LCD_Line_Clear(1);
       lcd.print(WiFi_Pass_Connect);
    }
      if ( Mode == 1 ) {
       LCD_Line_Clear(0);
       lcd.print("PassWord Operate");
       LCD_Line_Clear(1);
       lcd.print(WiFi_Pass_Connect);
    }
    if ( Mode == 2 ) {
      Serial.println(WiFi_Pass_Connect);
    }
}
void Show_IP_WORD(int Mode) {
unsigned int i;
byte Len;
   Len = EEPROM.read(4);
   memset(WiFi_IP_Connect, 0, sizeof(WiFi_IP_Connect));
   if ( Len > 20 ) { Len = 20; }
   for (i = 0 ; i< Len - 1 ; i ++) {
       Bu = EEPROM.read(i+50);
       if ( Bu != 0x20 ) {
           WiFi_IP_Connect[i] = Bu;
       }
       else{
           WiFi_IP_Connect[i] = 0;
```

```
      }
    }
      if ( Mode == 0 ) {
       LCD_Line_Clear(0);
       lcd.print("IP Setting");
       LCD_Line_Clear(1);
       lcd.print(WiFi_IP_Connect);
    }
      if ( Mode == 1 ) {
       LCD_Line_Clear(0);
       lcd.print("IP Operate");
       LCD_Line_Clear(1);
       lcd.print(WiFi_IP_Connect);
    }
    if ( Mode == 2 ) {
      Serial.println(WiFi_IP_Connect);
    }
}
void Setting_Mode() {
int Loop = 1;
  LCD_Line_Clear(0);
  lcd.print("Setting Wifi");
  LCD_Line_Clear(1);
  lcd.print("Wait Connecting");
  Serial.print("Setting soft-AP configuration ... ");
  Serial.println(WiFi.softAPConfig(WiFi_local_IP, gateway, subnet) ? "Ready" : "Failed!");
  delay(500);
  Serial.print("Setting soft-AP ... ");
  Serial.println(WiFi.softAP(WiFi_SSID,WiFi_PASS) ? "Ready" : "Failed!");
  delay(500);
  Serial.print("Soft-AP IP address = ");
  Serial.println(WiFi.softAPIP());
  LCD_Line_Clear(1);
  lcd.print("OK  Connected ");
  delay(1000);
```

```
  server.begin(); //TCP Server
  delay(500);
  LCD_Line_Clear(0);
  lcd.print("Module_MODBUS");
  LCD_Line_Clear(1);
  lcd.print(WiFi.softAPIP());
  while( Loop == 1)
 {
    Setting_Config_WiFi();
  }
}

void setup()
{
  Serial.begin(115200);
  pinMode(18,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(17,OUTPUT);
  pinMode(16,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(0,OUTPUT);
  pinMode(2,OUTPUT);
  pinMode(15,OUTPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);
  pinMode(36, INPUT);
  pinMode(39, INPUT);
  pinMode(34, INPUT);
  pinMode(35, INPUT);
  pinMode(27, INPUT);
  pinMode(14, INPUT);
  pinMode(12, INPUT);
  pinMode(13, INPUT);
  pinMode(SW_Setting, INPUT);
  lcd.begin();
```

```
  lcd.backlight();
  LCD_Line_Clear(1);
  LCD_Line_Clear(0);
  lcd.print("Push SW Setting");
long Tim;
 Tim =  millis();
while ( Tim+2000 >  millis() ) {
    if ( digitalRead(SW_Setting) == 0 ) {
        Setting_Mode();
    }
}
  Operate_Mode();
  setup2();
}
long Time_100mSec = 0,Time_1Sec = 0;
bool Positive_100mSec = 0 ,Positive_1Sec = 0;

void Update_Times_100mSec_1Sec() {
    Positive_100mSec = 0 ;
    Positive_1Sec  = 0 ;
    if (  millis() >= Time_100mSec+100 ) {
        Time_100mSec = millis();
        Positive_100mSec = 1;
    }
    if (  millis() >= Time_1Sec+1000 ) {
        Time_1Sec = millis();
        Positive_1Sec = 1;
    }
}

void Setting_Config_WiFi() {
    int i ;
    static int Time_Blacklight = 0;
    Update_Times_100mSec_1Sec();
    if ( Positive_1Sec == 1 )
```

```
  {
        Time_Blacklight += Positive_1Sec;
        if ( Time_Blacklight >= 10 ) {
            Time_Blacklight = 0;
            lcd.noBacklight();
        }
    }
       SW_Read();
       if ( SW1_Positive == 1 ) {
            MODE_LCD += SW1_Positive ;
            lcd.backlight();
            Time_Blacklight = 0;  // Start 0-10 Sec
            if ( MODE_LCD >= 4 ) {
                MODE_LCD = 0;
            }
            Show_LCD_Mode();
    WiFiClient client = server.available();
    if (!client) {   return; }
    lcd.backlight();
    Time_Blacklight = 0;  // Start 0-10 Sec
    Serial.println("New client");
    String request = "";
    request = client.readStringUntil('\r');
    Serial.println(request);
    if ( request.indexOf("Re_et_Module") > 0 )  { ESP.restart();
}
    String line = "";
    if ( request.indexOf("*W*") > 0 ) {
        line = request.substring(request.indexOf("=")+1,request.indexOf("HTTP"));
        unsigned int Len = line.length() ;
        EEPROM.write(0,Len);
        EEPROM.commit();
        for (i = 0 ; i< Len ; i ++) {
          EEPROM.write(i+5,line[i]);
          EEPROM.commit();
```

```
        }
        Show_SSID(0);
        Serial.println(line);
    }
    if ( request.indexOf("*P*") > 0 ) {
        line = request.substring(request.indexOf("=")+1,request.indexOf("HTTP"));
        unsigned int Len = line.length() ;
        EEPROM.write(3,Len);
        EEPROM.commit();
        for (i = 0 ; i< Len ; i ++) {
          EEPROM.write(i+25,line[i]);
          EEPROM.commit();
        }
        Show_PASS_WORD(0);
        Serial.println(line);
    }
    if ( request.indexOf("*I*") > 0 ) {
        line = request.substring(request.indexOf("=")+1,request.indexOf("HTTP"));
        unsigned int Len = line.length() ;
        EEPROM.write(4,Len);
        EEPROM.commit();
        for (i = 0 ; i< Len ; i ++) {
          EEPROM.write(i+50,line[i]);
          EEPROM.commit();
        }
        Show_IP_WORD(0);
        Serial.println(line);
    }
        client.flush();
        client.print("<h1>SSID = ");Show_SSID(5); line = WiFi_SSID_Connect;
client.print(WiFi_SSID_Connect); client.print("</h1>");
        client.println("<h2><form method=get>:   New SSID   :<input type=text size=20
name=*W*#><input type=submit value=SAVE></form><h2>");
        client.print("<h1>PASSWORD = ");Show_PASS_WORD(5); line = WiFi_Pass_Connect;
client.print(line); client.print("</h1>");
```

```
        client.println("<h2><form method=get>: New PASSWORD :<input type=text size=20
 name=*P*#><input type=submit value=SAVE></form><h2>");
        client.print("<h1>Fix IP Address = ");Show_IP_WORD(5);line = WiFi_IP_Connect;
 client.print(line); client.print("</h1>");
        client.println("<h2><form method=get>:Fix IP Address:<input type=text size=20
 name=*I*#><input type=submit value=SAVE></form><h2>");
        client.println("<a href=\"/Re_et_Module\"><button><h1>   RESET MODULE MODBUS TCP
 <h1></button></a></p>");
        client.stop();
}
void Show_LCD_Mode() {
  switch ( MODE_LCD ) {
    case 0:
        LCD_Line_Clear(0);
        lcd.print("Setting Wifi");
        LCD_Line_Clear(1);
        lcd.print("Module_MODBUS_TCP");
    break;
    case 1:
        LCD_Line_Clear(0);
        lcd.print("1. SSID");
        Show_SSID ( 1);
    break;
    case 2:
        LCD_Line_Clear(0);
        lcd.print("2. PASSWORD");
        Show_PASS_WORD(1);
    break;
    case 3:
        LCD_Line_Clear(0);
        lcd.print("3. IP ADDRESS");
        Show_IP_WORD(1);
    break;
  }
}
```

```
void Operate_Mode() {
int Count60=0;
   WiFi.disconnect();
   Show_SSID(1); delay(1000);
   Show_PASS_WORD(1);delay(1000);
   Show_IP_WORD(1);delay(1000);
   WiFi.mode(WIFI_STA); // Wifi Station
   delay(100);
   WiFi.begin(WiFi_SSID_Connect, WiFi_Pass_Connect);
   server.begin();
   Serial.println("\nConnecting to WiFi");
   LCD_Line_Clear(0);lcd.print("Connecting WIFI");
   LCD_Line_Clear(1);lcd.print("   Wait ");
   while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Count60++;
    lcd.setCursor(14,1);
    lcd.print(Count60);
    if ( Count60 >= 60 ) {
      LCD_Line_Clear(0);lcd.print(" Reset Module ");
      LCD_Line_Clear(1);lcd.print("   Wait  ");
      delay(1000);
      ESP.restart(); }
  }
if ( WiFi_IP_Connect != "" ) {
  uint8_t ip1[4];
  sscanf(WiFi_IP_Connect, "%u.%u.%u.%u", &ip1[0], &ip1[1], &ip1[2], &ip1[3]);
  Serial.println(WiFi_IP_Connect);
  byte Len;   Len = EEPROM.read(4);
  if ( Len > 9 ) {
       IPAddress ip(ip1[4]);
       IPAddress gateway1(ip1[4]);
       IPAddress subnet1(255,255,255,0);
       WiFi.config(ip1, gateway1, subnet1);
```

```
        LCD_Line_Clear(0);lcd.print("   Static IP   ");
        Serial.println(WiFi.localIP());
        LCD_Line_Clear(1);
        lcd.setCursor(2,1);lcd.print(WiFi.localIP());
    }
    else{
        LCD_Line_Clear(0);lcd.print("   Dynamic IP   ");
        Serial.println(WiFi.localIP());
        LCD_Line_Clear(1);
        lcd.setCursor(2,1);lcd.print(WiFi.localIP());
    }
 }
}
//////////////////////////////////////////
#define MODBUSIP_PORT       502
#define MODBUSIP_MAXFRAME   200
#define MODBUSIP_TIMEOUT    10
#define MAX_REGS    32
#define MAX_FRAME   128
WiFiServer server_MODBUS(MODBUSIP_PORT);
typedef unsigned int u_int;
#define countof(a) (sizeof(a) / sizeof(a[0]))
int Flag_Sec1 = 100;
int Flag_Sec2 = 0;
enum {
    MB_FC_READ_COILS       = 0x01, // Read Coils (Output) Status 0xxxx
    MB_FC_READ_INPUT_STAT  = 0x02, // Read Input Status (Discrete Inputs) 1xxxx
    MB_FC_READ_REGS        = 0x03, // Read Holding Registers 4xxxx
    MB_FC_READ_INPUT_REGS  = 0x04, // Read Input Registers 3xxxx
    MB_FC_WRITE_COIL       = 0x05, // Write Single Coil (Output) 0xxxx
    MB_FC_WRITE_REG        = 0x06, // Preset Single Register 4xxxx
    MB_FC_WRITE_COILS      = 0x0F, // Write Multiple Coils (Outputs) 0xxxx
    MB_FC_WRITE_REGS       = 0x10, // Write block of contiguous registers 4xxxx
};
enum {
```

```
    MB_EX_ILLEGAL_FUNCTION = 0x01, // Function Code not Supported
    MB_EX_ILLEGAL_ADDRESS  = 0x02, // Output Address not exists
    MB_EX_ILLEGAL_VALUE    = 0x03, // Output Value not in Range
    MB_EX_SLAVE_FAILURE    = 0x04, // Slave Deive Fails to process request
};
enum {
    MB_REPLY_OFF    = 0x01,
    MB_REPLY_ECHO   = 0x02,
    MB_REPLY_NORMAL = 0x03,
};
typedef struct TRegister {
    word address;
    word value;
    struct TRegister* next;
}
TRegister;
class Modbus
{
   private:
       TRegister *_regs_head;
       TRegister *_regs_last;
       void readRegisters(word startreg, word numregs);
       void writeSingleRegister(word reg, word value);
       void writeMultipleRegisters(byte* frame,word startreg, word numoutputs, byte bytecount);
       void exceptionResponse(byte fcode, byte excode);
          void readCoils(word startreg, word numregs);
          void readInputStatus(word startreg, word numregs);
          void readInputRegisters(word startreg, word numregs);
          void writeSingleCoil(word reg, word status);
          void writeMultipleCoils(byte* frame,word startreg, word numoutputs, byte bytecount);
       TRegister* searchRegister(word addr);
       void addReg(word address, word value = 0);
       bool Reg(word address, word value);
       word Reg(word address);
   protected:
```

```
        byte *_frame;
        byte  _len;
        byte  _reply;
        void receivePDU(byte* frame);
    public:
        Modbus();
        void addHreg(word offset, word value = 0);
        bool Hreg(word offset, word value);
        word Hreg(word offset);
            void addCoil(word offset, bool value = false);
            void addIsts(word offset, bool value = false);
            void addIreg(word offset, word value = 0);
            bool Coil(word offset, bool value);
            bool Ists(word offset, bool value);
            bool Ireg(word offset, word value);
            bool Coil(word offset);
            bool Ists(word offset);
            word Ireg(word offset);
};
class ModbusIP : public Modbus {
    private:
        byte _MBAP[8];
    public:
        ModbusIP();
        void config(const char* ssid, const char* password);
        void task();
};
////////////////////////////////////////////////////////////////////////////////////////////////////
Modbus::Modbus() {
    _regs_head = 0;
    _regs_last = 0;
}
TRegister* Modbus::searchRegister(word address) {
    TRegister *reg = _regs_head;
    if(reg == 0) return(0);
```

```
    do {
        if (reg->address == address) return(reg);
        reg = reg->next;
  } while(reg);
  return(0);
}
void Modbus::addReg(word address, word value) {
  TRegister *newreg;
  newreg = (TRegister *) malloc(sizeof(TRegister));
  newreg->address = address;
  newreg->value   = value;
  newreg->next    = 0;
  if(_regs_head == 0) {
      _regs_head = newreg;
      _regs_last = _regs_head;
    }
else {
      _regs_last->next = newreg;
      _regs_last = newreg;
    }
}
bool Modbus::Reg(word address, word value) {
    TRegister *reg;
    reg = this->searchRegister(address);
    if (reg) {
        reg->value = value;
        return true;
    } else
        return false;
}
word Modbus::Reg(word address) {
    TRegister *reg;
    reg = this->searchRegister(address);
    if(reg)
        return(reg->value);
```

```
    else
        return(0);
}
void Modbus::addHreg(word offset, word value) {
    this->addReg(offset + 40001, value);
}

bool Modbus::Hreg(word offset, word value) {
    return Reg(offset + 40001, value);
}
word Modbus::Hreg(word offset) {
    return Reg(offset + 40001);
}
    void Modbus::addCoil(word offset, bool value) {
        this->addReg(offset + 1, value?0xFF00:0x0000);
    }
    void Modbus::addIsts(word offset, bool value) {
        this->addReg(offset + 10001, value?0xFF00:0x0000);
    }
    void Modbus::addIreg(word offset, word value) {
        this->addReg(offset + 30001, value);
    }
    bool Modbus::Coil(word offset, bool value) {
        return Reg(offset + 1, value?0xFF00:0x0000);
    }
    bool Modbus::Ists(word offset, bool value) {
        return Reg(offset + 10001, value?0xFF00:0x0000);
    }
    bool Modbus::Ireg(word offset, word value) {
        return Reg(offset + 30001, value);
    }
    bool Modbus::Coil(word offset) {
        if (Reg(offset + 1) == 0xFF00) {
            return true;
        } else return false;
```

```
}
bool Modbus::Ists(word offset) {
    if (Reg(offset + 10001) == 0xFF00) {
        return true;
    } else return false;
}
word Modbus::Ireg(word offset) {
    return Reg(offset + 30001);
}
void Modbus::receivePDU(byte* frame) {
    byte fcode  = frame[0];
    word field1 = (word)frame[1] << 8 | (word)frame[2];
    word field2 = (word)frame[3] << 8 | (word)frame[4];
    switch (fcode) {
        case MB_FC_WRITE_REG:
            this->writeSingleRegister(field1, field2);
        break;
        case MB_FC_READ_REGS:
            this->readRegisters(field1, field2);
        break;
        case MB_FC_WRITE_REGS:
            this->writeMultipleRegisters(frame,field1, field2, frame[5]);
        break;
        case MB_FC_READ_COILS:
            this->readCoils(field1, field2);
        break;
        case MB_FC_READ_INPUT_STAT:
            this->readInputStatus(field1, field2);
        break;
        case MB_FC_READ_INPUT_REGS:
            this->readInputRegisters(field1, field2);
        break;
        case MB_FC_WRITE_COIL:
            this->writeSingleCoil(field1, field2);
        break;
```

```
        case MB_FC_WRITE_COILS:
            this->writeMultipleCoils(frame,field1, field2, frame[5]);
        break;
        default:
            this->exceptionResponse(fcode, MB_EX_ILLEGAL_FUNCTION);
    }
}
void Modbus::exceptionResponse(byte fcode, byte excode) {
    free(_frame);
    _len = 2;
    _frame = (byte *) malloc(_len);
    _frame[0] = fcode + 0x80;
    _frame[1] = excode;
    _reply = MB_REPLY_NORMAL;
}
void Modbus::readRegisters(word startreg, word numregs) {
    if (numregs < 0x0001 || numregs > 0x007D) {
        this->exceptionResponse(MB_FC_READ_REGS, MB_EX_ILLEGAL_VALUE);
        return;
    }
    if (!this->searchRegister(startreg + 40001)) {
        this->exceptionResponse(MB_FC_READ_REGS, MB_EX_ILLEGAL_ADDRESS);
        return;
    }
    free(_frame);
  _len = 0;
  _len = 2 + numregs * 2;
    _frame = (byte *) malloc(_len);
    if (!_frame) {
        this->exceptionResponse(MB_FC_READ_REGS, MB_EX_SLAVE_FAILURE);
        return;
    }
    _frame[0] = MB_FC_READ_REGS;
    _frame[1] = _len - 2;   //byte count
    word val;
```

```
    word i = 0;
  while(numregs--) {
        val = this->Hreg(startreg + i);
        _frame[2 + i * 2]  = val >> 8;
        _frame[3 + i * 2] = val & 0xFF;
        i++;
  }
    _reply = MB_REPLY_NORMAL;
}
void Modbus::writeSingleRegister(word reg, word value) {
    if (!this->Hreg(reg, value)) {
        this->exceptionResponse(MB_FC_WRITE_REG, MB_EX_ILLEGAL_ADDRESS);
        return;
    }
    if (this->Hreg(reg) != value) {
        this->exceptionResponse(MB_FC_WRITE_REG, MB_EX_SLAVE_FAILURE);
        return;
    }
    _reply = MB_REPLY_ECHO;
}
void Modbus::writeMultipleRegisters(byte* frame,word startreg, word numoutputs, byte
bytecount) {
    if (numoutputs < 0x0001 || numoutputs > 0x007B || bytecount != 2 * numoutputs) {
        this->exceptionResponse(MB_FC_WRITE_REGS, MB_EX_ILLEGAL_VALUE);
        return;
    }
    for (int k = 0; k < numoutputs; k++) {
        if (!this->searchRegister(startreg + 40001 + k)) {
            this->exceptionResponse(MB_FC_WRITE_REGS, MB_EX_ILLEGAL_ADDRESS);
            return;
        }
    }
    free(_frame);
  _len = 5;
    _frame = (byte *) malloc(_len);
```

```
    if (!_frame) {
        this->exceptionResponse(MB_FC_WRITE_REGS, MB_EX_SLAVE_FAILURE);
        return;
    }
    _frame[0] = MB_FC_WRITE_REGS;
    _frame[1] = startreg >> 8;
    _frame[2] = startreg & 0x00FF;
    _frame[3] = numoutputs >> 8;
    _frame[4] = numoutputs & 0x00FF;
    word val;
    word i = 0;
  while(numoutputs--) {
        val = (word)frame[6+i*2] << 8 | (word)frame[7+i*2];
        this->Hreg(startreg + i, val);
        i++;
  }
    _reply = MB_REPLY_NORMAL;
}
void Modbus::readCoils(word startreg, word numregs) {
    if (numregs < 0x0001 || numregs > 0x07D0) {
        this->exceptionResponse(MB_FC_READ_COILS, MB_EX_ILLEGAL_VALUE);
        return;
    }
    if (!this->searchRegister(startreg + 1)) {
        this->exceptionResponse(MB_FC_READ_COILS, MB_EX_ILLEGAL_ADDRESS);
        return;
    }
    free(_frame);
  _len = 0;
  _len = 2 + numregs/8;
  if (numregs%8) _len++; //Add 1 to the message length for the partial byte.
    _frame = (byte *) malloc(_len);
    if (!_frame) {
        this->exceptionResponse(MB_FC_READ_COILS, MB_EX_SLAVE_FAILURE);
        return;
```

```
    }
    _frame[0] = MB_FC_READ_COILS;
    _frame[1] = _len - 2; //byte count (_len - function code and byte count)
    byte bitn = 0;
    word totregs = numregs;
    word i;
    i = (totregs - numregs) / 8;
  while (numregs--) {
    if (this->Coil(startreg))
     { bitSet(_frame[2+i], bitn);}
    else
     { bitClear(_frame[2+i], bitn);}
      i = (totregs - numregs) / 8;
    bitn++;
    if (bitn == 8) { bitn = 0;_frame[2+i] = 0; }
    startreg++;
  }
    _reply = MB_REPLY_NORMAL;
}
void Modbus::readInputStatus(word startreg, word numregs) {
    if (numregs < 0x0001 || numregs > 0x07D0) {
        this->exceptionResponse(MB_FC_READ_INPUT_STAT, MB_EX_ILLEGAL_VALUE);
        return;
    }
    if (!this->searchRegister(startreg + 10001)) {
        this->exceptionResponse(MB_FC_READ_COILS, MB_EX_ILLEGAL_ADDRESS);
        return;
    }
    //Clean frame buffer
    free(_frame);
  _len = 0;
  _len = 2 + numregs/8;
  if (numregs%8) _len++; //Add 1 to the message length for the partial byte.
    _frame = (byte *) malloc(_len);
    if (!_frame) {
```

```
        this->exceptionResponse(MB_FC_READ_INPUT_STAT, MB_EX_SLAVE_FAILURE);
        return;
    }
    _frame[0] = MB_FC_READ_INPUT_STAT;
    _frame[1] = _len - 2;
    byte bitn = 0;
    word totregs = numregs;
    word i;
    i = (totregs - numregs) / 8;
  while (numregs--) {
    if (this->lsts(startreg))
     { bitSet(_frame[2+i], bitn); }
    else
     { bitClear(_frame[2+i], bitn); }
     i = (totregs - numregs) / 8;
    bitn++;
    if (bitn == 8) { bitn = 0; }
    startreg++;
  }
    _reply = MB_REPLY_NORMAL;
}
void Modbus::readInputRegisters(word startreg, word numregs) {
    if (numregs < 0x0001 || numregs > 0x007D) {
        this->exceptionResponse(MB_FC_READ_INPUT_REGS, MB_EX_ILLEGAL_VALUE);
        return;
    }
    if (!this->searchRegister(startreg + 30001)) {
        this->exceptionResponse(MB_FC_READ_COILS, MB_EX_ILLEGAL_ADDRESS);
        return;
    }
    free(_frame);
  _len = 0;
  _len = 2 + numregs * 2;
    _frame = (byte *) malloc(_len);
    if (!_frame) {
```

```
        this->exceptionResponse(MB_FC_READ_INPUT_REGS, MB_EX_SLAVE_FAILURE);
        return;
    }
    _frame[0] = MB_FC_READ_INPUT_REGS;
    _frame[1] = _len - 2;
    word val;
    word i = 0;
  while(numregs--) {
        val = this->Ireg(startreg + i);
        _frame[2 + i * 2]  = val >> 8;
        _frame[3 + i * 2] = val & 0xFF;
        i++;
  }
    _reply = MB_REPLY_NORMAL;
}
void Modbus::writeSingleCoil(word reg, word status) {
    if (status != 0xFF00 && status != 0x0000) {
        this->exceptionResponse(MB_FC_WRITE_COIL, MB_EX_ILLEGAL_VALUE);
        return;
    }
    if (!this->Coil(reg, (bool)status)) {
        this->exceptionResponse(MB_FC_WRITE_COIL, MB_EX_ILLEGAL_ADDRESS);
        return;
    }
    if (this->Coil(reg) != (bool)status) {
        this->exceptionResponse(MB_FC_WRITE_COIL, MB_EX_SLAVE_FAILURE);
        return;
    }
    _reply = MB_REPLY_ECHO;
}
void Modbus::writeMultipleCoils(byte* frame,word startreg, word numoutputs, byte bytecount) {
    word bytecount_calc = numoutputs / 8;
    if (numoutputs%8) bytecount_calc++;
    if (numoutputs < 0x0001 || numoutputs > 0x07B0 || bytecount != bytecount_calc) {
        this->exceptionResponse(MB_FC_WRITE_COILS, MB_EX_ILLEGAL_VALUE);
```

```
            return;
        }
        for (int k = 0; k < numoutputs; k++) {
            if (!this->searchRegister(startreg + 1 + k)) {
                this->exceptionResponse(MB_FC_WRITE_COILS, MB_EX_ILLEGAL_ADDRESS);
                return;
            }
        }
        free( frame);
      _len = 5;
        _frame = (byte *) malloc(_len);
        if (!_frame) {
            this->exceptionResponse(MB_FC_WRITE_COILS, MB_EX_SLAVE_FAILURE);
            return;
        }
        _frame[0] = MB_FC_WRITE_COILS;
        _frame[1] = startreg >> 8;
        _frame[2] = startreg & 0x00FF;
        _frame[3] = numoutputs >> 8;
        _frame[4] = numoutputs & 0x00FF;
        byte bitn = 0;
        word totoutputs = numoutputs;
        word i;
      while (numoutputs--) {
            i = (totoutputs - numoutputs) / 8;
            this->Coil(startreg, bitRead(frame[6+i], bitn));
            bitn++;
            if (bitn == 8) bitn = 0;
            startreg++;
      }
        _reply = MB_REPLY_NORMAL;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
ModbusIP::ModbusIP() {
}
```

```
void ModbusIP::config(const char* ssid, const char* password) {
  WiFi.begin(ssid, password);
  server_MODBUS.begin();
}
////////////////////////////////////////////////////////////////////////////////////////////////////
//Modbus Registers Offsets (0-9999)
const int LED_COIL1 = 0;
const int LED_COIL2 = 1;
const int LED_COIL3 = 2;
const int LED_COIL4 = 3;
const int LED_COIL5 = 4;
const int LED_COIL6 = 5;
const int LED_COIL7 = 6;
const int LED_COIL8 = 7;
const int TEST_HREG = 1;
const int ledPin = 5; //GPIO_05
int CountC = 0;
ModbusIP mb;
void Re_WIFI_Connect() {
  mb.config("KMITL-WiFi", "");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(WiFi.localIP());
  delay(1000);
}
void setup2() {
  pinMode(18,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(17,OUTPUT);
  pinMode(16,OUTPUT);
  pinMode(4,OUTPUT);
```

```
 pinMode(0,OUTPUT);
 pinMode(2,OUTPUT);
 pinMode(15,OUTPUT);
 pinMode(A4, INPUT);
 pinMode(A5, INPUT);
 pinMode(36, INPUT);
 pinMode(39, INPUT);
 pinMode(34, INPUT);
 pinMode(35, INPUT);
 pinMode(27, INPUT);
 pinMode(14, INPUT);
 pinMode(12, INPUT);
 pinMode(13, INPUT);
 pinMode(SW_Setting, INPUT);
 mb.addCoil(LED_COIL1);
 mb.addCoil(LED_COIL2);
 mb.addCoil(LED_COIL3);
 mb.addCoil(LED_COIL4);
 mb.addCoil(LED_COIL5);
 mb.addCoil(LED_COIL6);
 mb.addCoil(LED_COIL7);
 mb.addCoil(LED_COIL8);
 mb.addHreg(0, 0x0);
 mb.addHreg(1, 0x0);
 mb.addHreg(2, 0x0);
 mb.addHreg(3, 0x0);
 mb.addHreg(4, 0x0);
 mb.addHreg(5, 0x0);
 mb.addHreg(6, 0x0);
 mb.addHreg(7, 0x0);
 mb.addIreg(0);
 mb.addIreg(1);
 mb.addIreg(2);
 mb.addIreg(3);
 mb.addIreg(4);
```

```
  mb.addIreg(5);
  mb.addIsts(0);
  mb.addIsts(1);
  mb.addIsts(2);
  mb.addIsts(3);
  mb.addIsts(4);
  mb.addIsts(5);
  mb.addIsts(6);
  mb.addIsts(7);
  server_MODBUS.begin();
}
void Write_Device(byte* frame) {
    byte fcode  = frame[0];
    switch (fcode) {
        case MB_FC_WRITE_REG:
        case MB_FC_WRITE_REGS:
            dacWrite(DAC1 , mb.Hreg(0));
                dacWrite(DAC2 , mb.Hreg(1));
            snprintf_P(datestring,
            countof(datestring),
            PSTR("%6d:%6d"),
            mb.Hreg(0),
            mb.Hreg(1));
            lcd.setCursor(0, 0);
            lcd.print(datestring);
                    break;
        case MB_FC_WRITE_COIL:
        case MB_FC_WRITE_COILS:
            digitalWrite(18, mb.Coil(LED_COIL1));
            digitalWrite(5,  mb.Coil(LED_COIL2));
            digitalWrite(17, mb.Coil(LED_COIL3));
            digitalWrite(16, mb.Coil(LED_COIL4));
            digitalWrite(4,  mb.Coil(LED_COIL5));
            digitalWrite(0,  mb.Coil(LED_COIL6));
            digitalWrite(2,  mb.Coil(LED_COIL7));
```

```
            digitalWrite(15, mb.Coil(LED_COIL8));
            snprintf_P(datestring,
            countof(datestring),
            PSTR("%1d:%1d:%1d:%1d:%1d:%1d:%1d:%1d"),
            mb.Coil(LED_COIL1),
            mb.Coil(LED_COIL2),
            mb.Coil(LED_COIL3),
            mb.Coil(LED_COIL4),
            mb.Coil(LED_COIL5),
            mb.Coil(LED_COIL6),
            mb.Coil(LED_COIL7),
            mb.Coil(LED_COIL8));

            lcd.setCursor(0, 0);
            lcd.print(datestring);

            snprintf_P(datestring,
            countof(datestring),
            PSTR("%1d:%1d:%1d:%1d:%1d:%1d:%1d:%1d"),
            digitalRead(18),
            digitalRead(5),
            digitalRead(17),
            digitalRead(16),
            digitalRead(4),
            digitalRead(0),
            digitalRead(2),
            digitalRead(15));

            lcd.setCursor(0, 1);
            lcd.print(datestring);
        break;
    }
}
void Read_Device(byte* frame) {
    byte fcode  = frame[0];
```

```
switch (fcode) {
  case MB_FC_READ_REGS:
  break;
  case MB_FC_READ_COILS:
  break;
  case MB_FC_READ_INPUT_STAT:
        mb.Ists(0,digitalRead(36));
        mb.Ists(1,digitalRead(39));
        mb.Ists(2,digitalRead(34));
        mb.Ists(3,digitalRead(35));
        mb.Ists(4,digitalRead(27));
        mb.Ists(5,digitalRead(14));
        mb.Ists(6,digitalRead(12));
        mb.Ists(7,digitalRead(13));
        snprintf_P(datestring,
        countof(datestring),
        PSTR("%1d:%1d:%1d:%1d:%1d:%1d:%1d:%1d"),
        digitalRead(36),
        digitalRead(39),
        digitalRead(34),
        digitalRead(35),
        digitalRead(27),
        digitalRead(14),
        digitalRead(12),
        digitalRead(13));
        lcd.setCursor(0, 0);
        lcd.print(datestring);
 break;
 case MB_FC_READ_INPUT_REGS:
        int Sensor1 = analogRead(A4);
        float Volt1 =  (Sensor1 *3.3 )/ 4095;
        float Current1 = (Volt1 * 22) / 3.3 ;
        int Sensor2 = analogRead(A5);
        float Volt2 =  (Sensor2 * 3.3 )/ 4095;
        float Current2 = (Volt2 * 22) / 3.3 ;
```

```
                typedef union{
                  float number;
                  uint16_t bytes[2];
                } FLOATUNION_t;
                FLOATUNION_t  Convert_F;
                Convert_F.number = Current1;
                mb.Ireg(1, Convert_F.bytes[1]);
                mb.Ireg(0, Convert_F.bytes[0]);
                Convert_F.number = Current2;
                mb.Ireg(3, Convert_F.bytes[1]);
                mb.Ireg(2, Convert_F.bytes[0]);
                mb.Ireg(4,Sensor1);
                mb.Ireg(5,Sensor2);
        break;
    }
}
//////////////////////////////////////////////////////////
bool Check_WIFI() {
  if (WiFi.status() != WL_CONNECTED ) {
      delay(1000);
      if (WiFi.status() == WL_CONNECTED) {goto Run1; }
      delay(1000);
      LCD_Line_Clear(0);lcd.print("   Station Off  ");
      delay(1000);
      LCD_Line_Clear(0);lcd.print(" Reset Module  ");
      LCD_Line_Clear(1);lcd.print("     Wait      ");
      delay(1000);
      ESP.restart();
      return(0);
  }
Run1:
 return(1);
}
void ModbusIP::task() {
WiFiClient client = server_MODBUS.available();
```

```
int raw_len = 0;

if (Check_WIFI() == 0 ) {  return; }
while (client.connected()) {
   if (Check_WIFI() == 0 ) {   return;  }
   Connect = 0 ;
   raw_len = 0;
   if (client) {
   if (client.connected()) {
      for (int x = 0; x < 10; x++) { // Time to have data available
      if (client.available()) {
         while (client.available() > raw_len) {  //Computes data length
            raw_len = client.available();
        }
        break;
      }
    }
  }
  if (raw_len > 7) {
    for (int i=0; i<7; i++) _MBAP[i] = client.read(); //Get MBAP
    _len = _MBAP[4] << 8 | _MBAP[5];
    _len--; // Do not count with last byte from MBAP
    if (_MBAP[2] !=0 || _MBAP[3] !=0) return;   //Not a MODBUSIP packet
    if (_len > MODBUSIP_MAXFRAME) return;      //Length is over MODBUSIP_MAXFRAME
    _frame = (byte*) malloc(_len);
    raw_len = raw_len - 7;
    for (int i=0; i< raw_len; i++)  _frame[i] = client.read(); //Get Modbus PDU
    Read_Device(_frame);
    this->receivePDU(_frame);
    if (Print_Connect == 0 ) {
       Print_Connect = 1;
    }
    Connect = 1;Print_NoConnect = 0;
    client.flush();
    if (_reply != MB_REPLY_OFF) {
```

```
            _MBAP[4] = (_len+1) >> 8;     //_len+1 for last byte from MBAP
            _MBAP[5] = (_len+1) & 0x00FF;
        size_t send_len = (unsigned int)_len + 7;
        uint8_t sbuf[send_len];
        for (int i=0; i<7; i++)      sbuf[i] = _MBAP[i];
        for (int i=0; i<_len; i++)  sbuf[i+7] = _frame[i];
        Write_Device(_frame);
        client.write(sbuf, send_len);//digitalWrite(12,digitalRead(12)^1);
      }
     free(_frame);
      _len = 0;
    }
  }
}
if (Connect == 0 && Print_NoConnect == 0 ) {
    Print_NoConnect = 1;
    Print_Connect = 0;
}
}
void Show_Value_Sensor(int Ch) {
       switch ( Ch ) {
         case 0:
            if ( Print_NoConnect == 1 ) { One_T = 1;
                lcd.setCursor(0,0);lcd.print("S 502 Disconnect");
            }
            if ( Print_Connect == 1 ) {
                lcd.setCursor(0,0);lcd.print("S 502 Connected ");
                if ( One_T == 1 ) { Show_Normal = 50; One_T = 0;}
            }
             lcd.setCursor(0,1);lcd.print(" Press SW Read  ");
        break;
        case 1:
            lcd.setCursor(0,0);lcd.print("1. SSID          ");
            lcd.setCursor(0,1);Show_SSID (2);
        break;
```

```
        case 2:
            lcd.setCursor(0,0);lcd.print("2. PASSWORD     ");
            lcd.setCursor(0,1);Show_PASS_WORD(2);
        break;
       case 3:
            lcd.setCursor(0,0);lcd.print("3. IP ADDRESS   ");
            LCD_Line_Clear(1);
            lcd.setCursor(1,1);lcd.print(WiFi.localIP());
        break;
       }
}
void Update_Mode() {
  Update_Times_100mSec_1Sec();  // Positive_100mSec = 0 ;Positive_1Sec  = 0 ;
  SW1_Positive = 0;
  if ( Positive_100mSec == 1 ) {
     SW_Read();
     if ( Show_Normal != 0 ) {
        if ( Show_Normal == 1) { lcd.noBacklight(); } //lcd.noBacklight();lcd.backlight();
        Show_Normal--;
     }
  }
  if ( Positive_1Sec == 1 ) {
      Show_Value_Sensor(Update_LCD);
  }
  if (SW1_Positive == 1 ) {
       lcd.backlight();
       Show_Normal = 50;
       Update_LCD++;if ( Update_LCD > 4 ) {Update_LCD =0; }
  }
}
void loop() {
   mb.task();
}
```