การรับส่งข้อมูลระหว่างโปโตคอล Modbus (RS-485) และโปโตคอล MQTT

Transfer data between Modbus (RS-485) Protocol and MQTT Protocol

โดย

นายบุญเต็ม จิกจักร์ รหัสนักศึกษา 633040254-4

นายชัยวัฒน์ มูลตรีศรี รหัสนักศึกษา 643040664-6

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร. ศราวุธ ชัยมูล

ภาควิชาวิศวกรรมไฟฟ้า

หลักสูตร วิศวกรรมระบบอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น

ใบประเมินผลงาน

ชื่อเรื่องภาษาไทย การรับส่งข้อมูลระหว่างโปโตคอล Modbus (RS-485) และโปโตคอล MQTT

ชื่อเรื่องภาษาอังกฤษ Transfer data between Modbus (RS-485) Protocol and MQTT Protocol

ผู้จัดทำ

นายบุญเต็ม จิกจักร์ รหัสนักศึกษา 633040254-4

นายชัยวัฒน์ มูลตรีศรี รหัสนักศึกษา 643040664-6

อาจารย์ที่ปรึกษา

_____

(รองศาสตราจารย์ ดร. ศราวุธ ชัยมูล)

อาจารย์ผู้ร่วมประเมิน

_____

(.....)

_____

(.....)

บทคัดย่อ

รายงานเล่มนี้จัดทำ ขึ้นเพื่อเป็นส่วนหนึ่งของวิชา การฝึกปฏิบัติทางวิชาชีพวิศวกรรมระบบอิเล็กทรอนิกส์4 รหัสวิชา EN243803 โดยมีจุดประสงค์เพื่อให้ได้ศึกษาและทดลองความรู้ที่ได้จากเรื่อง การรับส่งข้อมูลทีได้จาดการอ่านค่าจากเซ็นเซอร์ที่มีโปโตคอลเป็นModbus(RS-485)และส่งข้อมูลไปยังโปรโตคอล MQTT ได้อย่างสมบูรณ์

ผู้จัดทำได้เลือกหัวข้อนี้ในการทำ รายงาน เนื่องมาจากเป็นเรื่องที่น่าสนใจและเป็นหนึ่งของหัวข้อเดี่ยวกับทางบริษัท ไพรมัส จำกัด ซึ่งเป็นหัวข้อที่สามารถใช้งานได้จริงในระดับอุตสาหกรรม และ ต้องขอขอบคุณ รองศาสตราจารย์ ดร. ศราวุธ ชัยมูล ผู้ให้ความรู้และแนวทางการศึกษา และให้ความช่วยเหลือมาโดยตลอดผู้จัดทำ หวังว่ารายงานฉบับนี้จะให้ความรู้ และเป็นประโยชน์แก่ผู้อ่านทุก ๆ ท่าน หากมีข้อเสนอแนะประการใด ผู้จัดทำ

ขอรับไว้ด้วยความขอบพระคุณยิ่ง

นายบุญเต็ม จิกจักร์

นายชัยวัฒน์ มูลตรีศรี

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1

บทนำ

1.1 หลักการและเหตุผล

อุปกรณ์เครื่องและเซ็นเซอร์ในงานอุตสาหกรรมส่วนหนึ่งใช้การรับส่งข้อมูลโดยใช้โปรโตคอลและมีการรับส่งข้อมูลแบบ Modbus (RS-485) ซึ่งการส่งข้อมูลผ่านสายนั้นมีความยุ่งยากในการติดตั้งและการซ่อมบำรุงส่งผลให้มีค่าใช้จ่ายที่สูง เพื่อแก้ปัญหาดังกล่าวเลยมีหลักการที่ว่าจะส่งข้อมูลโดยแบบไร้สาย (Internet of thing, IoT) ขึ้นมา โดยเน้นที่ความสามารถในการเฝ้าสังเกต (Monitor) ได้ตลอดเวลาและมีความแม่นยำ ดังนั้นทางผู้จัดทำได้นำเสนอโครงการนี้เพื่อ จะผสานระหว่าง โปโตคอล Modbus (RS-485) เข้ากับโปโตคอล MQTT

โปรโตคอล Modbus เป็นรูปแบบการสื่อสารข้อมูลดิจิตอลแบบอนุกรมรูปแบบหนึ่ง ในการส่งข้อมูลระหว่างอุปกรณ์อิเล็กทรอนิกส์ ตามรูปที่ โดยอุปกรณ์ที่ต้องการข้อมูล เรียกว่า Modbus Master ส่วนอุปกรณ์ที่ให้ข้อมูลที่ต้องการ เรียกว่า Modbus Slave ซึ่งปัจจุบันการสื่อสารข้อมูลแบบ Modbus ได้รับความนิยมเป็นอย่างมาก เนื่องจากอุปกรณ์เครื่องมือวัดอุตสาหกรรมส่วนใหญ่จะใช้ RS485 แบบ Modbus RTU Protocol เช่น Power Meter, Digital Indicator, I/O Modules, PLC เป็นต้น

1.2 วัตถุประสงค์

1. เพื่อศึกษาการทำงานของโปรโตคอล Modbus และส่งสัญญาณตามมาตรฐาน Modbus (RS-485)
2. เพื่อออกแบบและสร้างอุปกรณ์ที่สื่อสารระหว่างโปรโตคอล Modbus กับโปรโตคอล MQTT
3. เพื่อใช้ไมโครคอลโทรเลอร์อ่านข้อมูลจากเซ็นเซอร์ที่มีรูปแบบการรับส่งข้อมูลแบบ Modbus (RS-485)และสามารถเลือกข้อมูลที่ต้องการอ่านได้
4. เพื่อสามารถใช้ Raspberry Pi กำหนดรูปแบบการเลือกอ่านข้อมูลจากเซ็นเซอร์ได้
5. เพื่อใช้ Raspberry Pi Publish ข้อมูลที่ได้จากการอ่านเซ็นเซอร์ไปยัง Broker ได้
6. เพื่อใช้ Node Red เขียนหน้าเว็บในการกำหนดการอ่านค่าของเซ็นเซอร์และกำหนดหัวข้อในการ Publish ข้อมูลไปยัง Broker บน Raspberry Pi ได้

1.3 ขอบเขตของงาน

1. เลือกใช้โครคอลโทลเลอร์เป็น ESP32
2. MQTT Broker เลือกใช้ Hive MQTT Server (Free Package)
3. MQTT Publisher เลือกใช้ Raspberry Pi 3 Model B+
4. Modbus (RS-485) to UART เลือกใช้ TTL to Modbus (RS-485) level serial UART module

5. เลือกใช้ Node Rad ในการเขียนหน้าเว็บบน Raspberry Pi
6. เลือกใช้ Soil Humidity Sensor PR-3000-H-N01และ XY-MD02 SHT20

1.4 แผนการดำเนินงาน

	รายการ/การดำเนินการ	2566		2567		
		พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.
1	เขียนข้อเสนอหัวข้อ Modbus (RS-485) ถึง โปโตคอล MQTT และออกแบบ					
2	เรียนรู้หลักการทำงาน - โปรโตคอลModbus - โปรโตคอล MQTT - Node Red - Raspberry Pi3					
3	- เขียน code อ่านค่าจากเซ็นเซอร์ -ทดสอบส่งข้อมูลในจาก publisher ไป ที่ broker และอ่านค่าจาก client					
4	-ทดลองส่งข้อมูลจาก Raspberry Pi3 ไปยัง MQTT					
5	บันทึกผลการทดลอง					
6	สรุปและเขียนรายงาน					
7	จัดทำรายงานเพื่อนำเสนอ					

ตารางที่ 1 แสดงแผนกานดำเนินงาน

1.5 แนวทางการดำเนินงาน

1. ใช้ TTL to RS485 level serial UART module อ่านค่าจาก เซ็นเซอร์เพื่อแปลงระดับสัญญาณเป็น ระดับสัญญาณโลจิก ใช้ TTL to Modbus (RS-485) ส่งค่าเข้า Microcontroller (ESP 32)
2. เขียนโค๊ดเพื่อให้ Microcontroller (ESP 32) ส่งค่าข้อมูลจากเซ็นเซอร์ไปยัง Raspberry Pi 3 ผ่าน Serial Uart
3. เขียน Node Red บน Raspberry Pi 3 เพื่ออ่านค่าข้อมูลที่ได้จาก Microcontroller (ESP 32) และส่งข้อมูล Publish ไปส่ง MQTT Broker
4. ใช้คอมพิวเตอร์เพื่อ Subscribe topic ของ MQTT Broker เพื่อรับข้อมูล

1.6 ผลที่คาดว่าจะได้รับ

1. เข้าใจหลักการทำงานของ โปรโตคอล Modbus และสามารถเขียนโค๊ดเพื่อให้ Microcontroller (ESP 32) อ่านค่าจากตัวเซ็นเซอร์และส่งข้อมูลไปยัง Raspberry ได้
2. สามารถออกแบบระบบของการสื่อสารระหว่างโปรโตคอล Modbus กับโปรโตคอล MQTT โดยที่เมื่อมีปัญหาต้องสามารถเข้าปรับปรุงระบบได้
3. สามารถเขียน Node Red บน Raspberry โดยที่เขียนเป็นเว็บและให้ Publish ไปส่ง MQTT Broker ได้

บทที่ 2

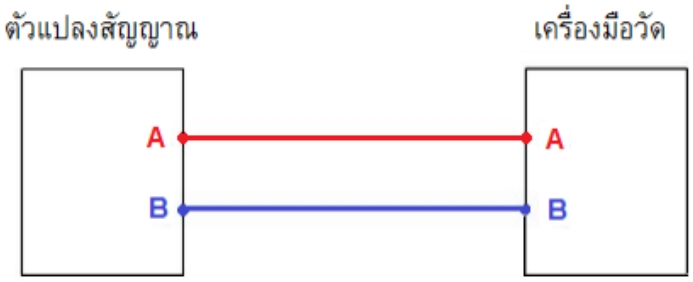
ทฤษฎีที่เกี่ยวข้อง

2.1 RS-485 และ MAX485 Module TTL to RS485

RS485 (ย่อมาจาก Recommended Standard no. 485) คือ มาตรฐานการสื่อสารข้อมูลดิจิตอลแบบอนุกรม (Serial Communication) ซึ่งถูกกำหนดขึ้นเพื่อให้เป็นมาตรฐานในการสื่อสารเพื่อรับ-ส่งข้อมูล ระหว่างอุปกรณ์เครื่องมือวัดกับระบบเครือข่ายคอมพิวเตอร์ (Computer Network) โดยสามารถส่งสัญญาณได้ไกล (สูงสุดถึง 1.2 km.)

หลักการทำงานของ RS485 RS485 เป็นมาตรฐานที่รับ-ส่งข้อมูลในแบบที่เรียกว่า Half Duplex คือ สามารถรับและส่งข้อมูลได้ทีละอย่างเท่านั้น ไม่สามารถทำทั้งสองอย่างได้ในเวลาเดียวกัน ซึ่งในการรับ-ส่งข้อมูลดิจิตอลแบบ RS485 นั้น จะส่งข้อมูลโดยใช้สายไฟเพียงแค่ 2 เส้น คือ A กับ B เป็นตัวบอกค่ารหัสดิจิตอล (Digital Code) โดยจะใช้ความแตกต่างของแรงดันไฟฟ้าระหว่างขั้ว A และ B เป็นตัวบอก ดังนี้

- เมื่อ Va - Vb ได้แรงดันไฟฟ้าน้อยกว่า -200 mV คือสัญญาณดิจิตอลเป็น 1
- เมื่อ Va - Vb ได้แรงดันไฟฟ้ามากกว่า +200 mV คือสัญญาณดิจิตอลเป็น 0

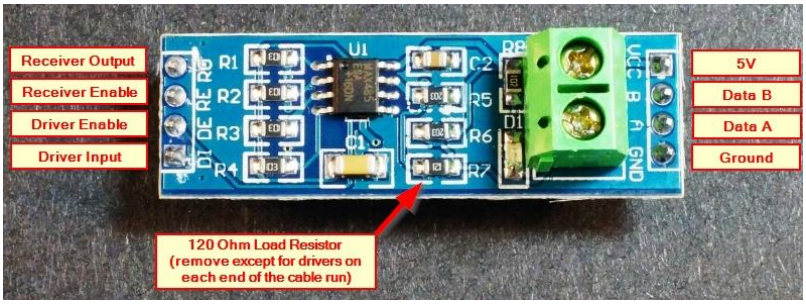


รูปที่ 1 แสดงการเชื่อมต่อ *RS*485 ระหว่างอุปกรณ์เครื่องมือวัด โดยใช้สาย 2 เส้น

โดยหลักการทำงานของ RS485 แบบ Network สามารถเชื่อมต่อการรับ-ส่งข้อมูลแบบเครือข่าย (Network) โดยมีอุปกรณ์ในเครือข่ายได้สูงสุดถึง 32 ตัว

MAX485 Module TTL to RS485

MAX485 module TTL คือโมดูลที่มี IC MAX485 อยู่บนบอร์ดโดยมาพร้อมกับขาต่อสำหรับใช้งานที่ง่ายต่อการเชื่อมต่อกับระบบอื่นๆ และป้องกันการเชื่อมต่อผิดพลาด โดย MAX485 คือ IC ที่ใช้ในการแปลงสัญญาณระหว่าง TTL (Transistor-Transistor Logic) และ RS-485 ตัว MAX485 module นี้จึงสามารถใช้ประยุกต์ในระบบ Arduino หรือระบบบนพื้นฐาน microcontroller อื่น ๆ เพื่อสื่อสารกับอุปกรณ์ที่ใช้มาตรฐาน RS-485 ได้



รูปที่ 2 MAX*485* Module TTL to *RS*485

2.2 Modbus

Modbus คือ โปรโตคอลการสื่อสารแบบเปิด (Open Protocol) ที่ใช้สำหรับการแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์อิเล็กทรอนิกส์ นิยมใช้ในระบบอัตโนมัติทางอุตสาหกรรม ทำงานบนรูปแบบการสื่อสารแบบ Master-Slave

โดย Modbus มี 2 ประเภท

1. Modbus RTU (Remote Terminal Unit): เป็นการสื่อสารแบบอนุกรมผ่านสายสัญญาณ
2. Modbus TCP: เป็นการสื่อสารแบบ TCP/IP ผ่านเครือข่าย Ethernet

โปรโตคอล Modbus มี product data unit (PDU) เมื่อมีการแม็พบัสด้วยโปรโตคอล Modbus จะมีการเพิ่มฟิลด์ข้อมูลบางอย่างลงใน ADU ซึ่ง ADU นี้สร้างโดยไคลเอ็นต์ผู้ใช้

ขนาดของ Modbus PDU สำหรับการสื่อสารแบบอนุกรม (serial communication) ถูกกำหนดเป็น 256 - 1 ไบต์สำหรับที่อยู่เซิร์ฟเวอร์ - 2 ไบต์สำหรับ CRC = 253 ดังนั้นขนาดคือ 253 ไบต์

ดังนั้นขนาดของ ADU สามารถคำนวณได้ดังนี้:

RTU: 256 + 1 ไบต์สำหรับที่อยู่ของอุปกรณ์ +2 ไบต์ สำหรับ CRC = 256

TCP: 256 +7 ไบต์ สำหรับ MBAP = 260

Modbus มี PDU สามประเภทหลัก:

mb_req_pdu = {function-code, request-data}

mb_rsp_pdu = {function-code, response-data}

mb_excep_rsp_pdu = {exception-function-code, request-data

Function Code:

รหัสฟังก์ชัน (Function Code) คือตัวระบุชนิดการดำเนินการที่ต้องการในโปรโตคอล Modbus โดยเข้ารหัสเป็นหนึ่งไบต์ กำหนดช่วงรหัสที่อนุญาต (ทศนิยม) ระหว่าง 1 ถึง 255 และสงวนช่วง 128 ถึง 255 ไว้สำหรับแจ้งข้อผิดพลาดหรือคำตอบพิเศษ รหัสฟังก์ชัน '0' ถือเป็นรหัสที่ไม่ถูกต้อง

Modbus function code	Action	Table name
1	Read coil	Discrete output Coils
2	Read discrete input	Discrete input
3	Read holding Register	Analog output Holding Register
4	Read input Register	Analog Input Register
5	Write to single coil	Discrete output Coil
6	Write to single holding Register	Analog output Holding Register
15	Write to multiple coils	Discrete output Coils
16	Write to multiple holding Registers	Analog output Holding Registers

ตารางที่ 2 ตาราง Function Code

ชนิดของข้อมูลใน Modbus:

Primary table	Object type	Read/Write	Description/comments
Discrete input	Single bit	Read-only	This type of data can be provided by an I/O system.
Coil	Single bit	Read-write	This type of data can be modified by an application program.
Input registers	16-bit word	Read-only	This type of data can be provided by an I/O system.
Holding registers	16-bit word	Read-write	This type of data can be modified by an application program.

Primary table	Coil/register number	Read/Write	Data addresses
Discrete input	10001 - 19999	Read-only	0000 to 270E
Coils	1-9999	Read-write	0000 to 270E
Input registers	30001 - 39999	Read-only	0000 to 270E
Holding registers	40001 - 49999	Read-write	0000 to 270E

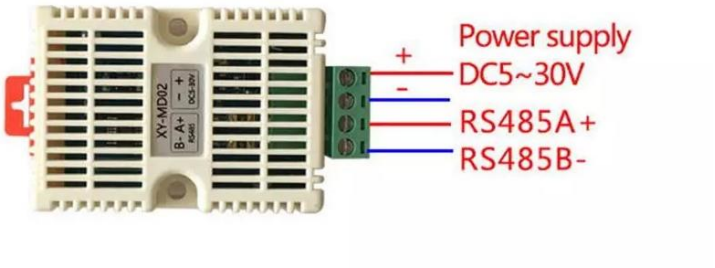
ตารางที่ 3 ตารางชนิดข้อมูลของ Modbus

2.3 เซ็นเซอร์วัดอุณหภูมิและความชื้น XY-MD02 SHT20

XY-MD02 เซ็นเซอร์วัดอุณหภูมิและความชื้นSHT20 Temperature and Humidity Transmitter Detection Sensor Module RS485 มีคุณสมบัติดังนี้

- ใช้วัดอุณหภูมิและความชื้น

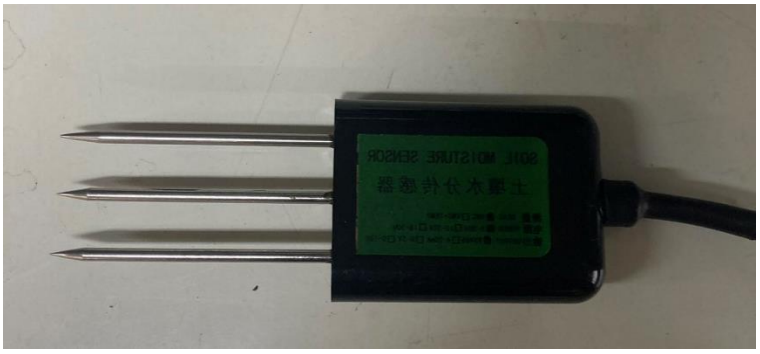
- สื่อสารผ่าน RS485 / Modbus RTU
- วัดอุณหภูมิได้ -40 ถึง 125 องศาเซลเซียล ความละเอียด 0.01 องศาเซลเซียส
- วัดความชื้อได้ 0 ถึง 100%RH ความละเอียด 0.04%RH
- ใช้แรงดันไฟฟ้า 5V ถึง 30V
- ติดตั้งบนรางปีกนก (DIN rail) ขนาดมาตรฐานได้



รูปที่ 3 XY-MD02 SHT20

2.4 เซ็นเซอร์วัดความชื้นในดิน Soil Humidity Sensor PR-3000-H-N01

เซ็นเซอร์วัดความชื้นในดิน PR-3000-H-N01 เป็นเซ็นเซอร์ที่ใช้สำหรับวัดความชื้นในดิน โดยใช้โปรโตคอลการสื่อสาร RS485 มีคุณสมบัติดังนี้



รูปที่ 4 Soil Humidity Sensor PR-3000-H-N01

- วัดค่าความชื้นในดิน
- ใช้โปรโตคอลการสื่อสาร RS485
- เชื่อมต่อกับอุปกรณ์ Arduino, ESP8266, และ ESP32 ได้
- แรงดันไฟฟ้าใช้งาน: 12-24VDC
- กระแสไฟใช้งาน: 35mA
- อุณหภูมิใช้งาน: -40 ถึง +85°C
- ความชื้นใช้งาน: 0-99% RH
- ระดับการป้องกัน: IP68
- ระยะการวัด: 0-100%
- ความแม่นยำ: ±3%
- ช่วงการวัด: 0-100%

2.5 ESP32

ESP32 เป็นชิปไมโครคอนโทรลเลอร์ 32 บิต ที่มี Wi-Fi และ Bluetooth เวอร์ชั่น 4.2 ในตัว ผลิตโดยบริษัท Espressif จากประเทศจีน ถือเป็นรุ่นต่อยอดจากชิป ESP8266

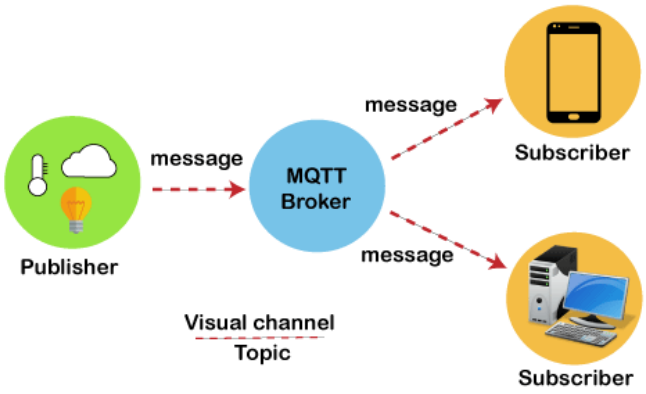
รูปที่ 5 ESP32

คุณสมบัติหลักของ ESP32

- CPU 32 บิต ความเร็วสูงถึง 240 MHz
- หน่วยความจำ RAM 520 KB
- หน่วยความจำ Flash 4 MB
- รองรับ Wi-Fi 802.11 b/g/n
- รองรับ Bluetooth 4.2
- รองรับ GPIO 38 ขา
- รองรับ ADC 12 บิต 2 ช่อง
- รองรับ DAC 8 บิต 2 ช่อง
- รองรับ SPI, I2C, UART, I²S
- รองรับ SD Card
- รองรับ RTC
- รองรับ USB OTG
- รองรับการเข้ารหัส AES / SHA2 / Elliptical Curve Cryptography / RSA-4096

2.6 MQTT

MQTT (Message Queuing Telemetry Transport) เป็นโปรโตคอลสำหรับใช้ส่งข้อความระหว่างอุปกรณ์ โดยใช้โมเดลเน็ตเวิร์คแบบ publish-subscribe ซึ่งจะแตกต่างจากโปรโตคอลอื่นๆโดยส่วนมากที่ใช้โมเดล Server-Client ในการรับส่งข้อมูล ตัวโปรโตคอลรันอยู่บนเทคโนโลยี TCP/IP จึงทำให้การส่งข้อมูลนั้นไม่มีการ loss ระหว่างทาง MQTT ถูกพัฒนาขึ้นมาเพื่อใช้ในการส่งข้อมูลจากที่ห่างไกลซึ่งใช้แบนด์วิธของเน็ตเวิร์คน้อยมาก และมีโครงสร้างตามรูปที่2



รูปที่ 6 MQTT Architecture

MQTT ประกอบไปด้วย

- Broker (Server) คือตัวกลางในการรับข้อมูลจาก Publisher และส่งข้อมูลให้กับ Subscriber
- Clients (Subscriber / Publisher)
- Publisher คือตัวส่งข้อมูลให้กับ Topic ที่อยู่ใน Broker เรียกว่าการ Publish
- Subscriber คือตัวรับข้อมูลจาก Topic ที่อยู่ใน Broker เรียกว่าการ Subscribe
- Topic คือหัวเรื่องที่เราต้องการรับส่งข้อมูล ระหว่าง Publisher กับ Subscriber

MQTT Topics

MQTT Topic เป็น UTF-8 String ในลักษณะเดียวกับ File Path คือสามารถจัดเป็นลำดับชั้นได้ด้วยการขั้นด้วย “/” ตัวอย่างเช่น myhome/floor-one/room-c/temperature ไคลเอนต์สามารถเลือก Publish หรือ Subscribe เฉพาะ Topic หรือ Subscribe หลาย Topic พร้อมๆ กันโดยใช้ Single-Level Wildcard (+) เช่น myhome/floor-one/+/temperature หมายถึงการขอเขียนหรือรับข้อความ temperature จากทุกๆ ห้องของ myhome/floor-one หรือ Multi-Level Wildcard (#) เช่น myhome/floor-one/# หมายถึงการขอเขียนหรือรับข้อความทั้งหมดที่มี Topic ขึ้นต้นด้วย myhome/floor-one เป็นต้น

MQTT Quality of Service (QoS)

ไคลเอนต์จะเป็นผู้กำหนดระดับของบริการส่งและรับข้อความหรือ QoS ที่ตนต้องการในแต่ละ Topic ในแพ็กเกต PUBLISH หรือ SUBSCRIBE และโบรกเกอร์จะตอบสนองด้วย QoS ระดับเดียวกันสำหรับ Topic นั้นๆ

QoS ใน MQTT แบ่งได้เป็น 3 ระดับ คือ

1. อย่างมากหนึ่งครั้ง (At Most Once) แทนด้วยโค้ด 0 **QoS 0** เป็นระดับบริการที่ต่ำที่สุด กล่าวคือ ไม่รับประกันว่าข้อความจะถูกส่งถึงผู้รับใดๆ เลยหรือไม่ หากไคลเอนต์ Publish ข้อความด้วย QoS 0 โบรกเกอร์จะไม่มีการตอบรับใดๆ ว่าได้ Publish ต่อไปให้ผู้รับรายอื่นหรือไม่ หากไม่มีผู้รับข้อความ โบรกเกอร์อาจเก็บข้อความไว้หรือลบทิ้งก็ได้ ขึ้นอยู่กับนโยบายของผู้ให้บริการเซิร์ฟเวอร์ ในทางกลับกันหากไคลเอนต์ที่เป็นผู้รับ Subscribe ไว้ด้วย QoS 0 เมื่อได้รับข้อความจากโบรกเกอร์ ก็ไม่ต้องส่งข้อความตอบรับใดๆ กลับ ทำให้การส่งข้อความแบบนี้รวดเร็วที่สุด เพราะไม่มีโอเวอร์เฮดในการตอบรับ ขณะเดียวกันหากข้อความถูกส่งไม่ถึง ก็ไม่มีทางทราบได้เช่นกัน
2. อย่างน้อยหนึ่งครั้ง (At Least Once) แทนด้วยโค้ด 1 **QoS 1** รับประกันว่าข้อความจะถูกส่งถึงผู้รับอย่างน้อยหนึ่งครั้ง การส่งลักษณะนี้ ผู้ส่งจะเก็บข้อความเอาไว้ จนกว่าจะได้รับแพ็กเกต PUBACK จากผู้รับ ในกรณีไคลเอนต์ขอ Publish ผู้รับข้อความซึ่งเป็นโบรกเกอร์จะต้อง Publish ต่อไปยังไคลเอนต์ที่ Subscribe ไว้อย่างน้อยหนึ่งครั้ง จึงจะสามารถส่งแพ็กเกตตอบรับไปกลับยังผู้ส่ง ในกรณี Subscribe ผู้ส่งซึ่งก็คือโบรกเกอร์จะต้องเก็บข้อความไว้จนกว่าไคลเอนต์ที่ตนส่งข้อความไปให้จะยืนยันตอบรับ ดังนั้นแพ็กเกต PUBACK จึงต้องมีหมายเลขไอดีเดียวกับแพ็กเกต PUBLISH เพื่อให้ผู้ส่งทราบว่าข้อความใดถูกส่งถึงแล้วและสามารถลบออกได้
3. หนึ่งครั้งเท่านั้น (Exactly Once) แทนด้วยโค้ด 2 **QoS 2** รับประกันว่าแต่ละข้อความจะถูกส่งถึงผู้รับเพียงหนึ่งครั้งเท่านั้น เป็นบริการที่ปลอดภัยที่สุดและช้าที่สุดของโพรโทคอล MQTT เนื่องจากผู้รับและผู้ส่งต้องส่งแพ็กเกตควบคุมไปกลับถึงสองรอบ เริ่มต้นด้วยผู้ส่งส่งข้อความไปในแพ็กเกต PUBLISH เมื่อผู้รับได้รับข้อความจะเก็บแพ็กเกตไว้และยืนยันกลับไปยังผู้ส่งด้วยแพ็กเกต PUBREC ผู้ส่งจึงสามารถลบข้อความนั้นของจากหน่วยเก็บข้อมูลของตนได้ และส่งแพ็กเกต PUBREL ไปยังผู้รับเพื่อให้ผู้รับสามารถลบสถานะการส่งข้อความนี้ออกได้ หากผู้รับเป็นไคลเอนต์ปลายทางที่ Subscribe ข้อความเอาไว้ ผู้รับจะส่งแพ็กเกต PUBCOM เพื่อยืนยันว่าข้อความถูกส่งถึงแล้วเรียบร้อยหนึ่งครั้ง หากผู้รับเป็นโบรกเกอร์ ทันทีที่ได้ Publish ข้อความต่อไปยังไคลเอนต์ปลายทางหนึ่งครั้ง จึงจะลบข้อความนั้นออก และปิดเซสชั่นด้วยการส่งแพ็กเกต PUBCOM กลับไปยังผู้ส่ง

2.7 Node-Red

Node-RED เป็นเครื่องมือสำหรับนักพัฒนาโปรแกรมในการเชื่อมต่ออุปกรณ์ฮาร์ดแวร์เข้ากับ APIs (Application Programming Interface) ซึ่งเป็นการพัฒนาโปรแกรมแบบ Flow-Based Programming

ที่มีหน้า UI สำหรับนักพัฒนาให้ใช้งานผ่าน Web Browser ทำให้การเชื่อมต่อเส้นทางการไหลของข้อมูลนั้นเป็นเรื่องง่าย

Node-RED เป็น Flow-Based Programming ทำให้เราแทบจะไม่ต้องเขียน Code ในการพัฒนาโปรแกรมเลย แค่เพียงเลือก Node มาวางแล้วเชื่อมต่อก็สามารถควบคุม I/O ได้ โดย Node-RED จะมี Node ให้เลือกใช้งานอย่างหลากหลาย สามารถสร้างฟังก์ชัน JavaScript ได้โดยใช้ Text Editor ที่มีอยู่ใน Node-RED และยังสามารถบันทึก Function, Templates, Flows เพื่อไปใช้งานกับงานอื่นได้

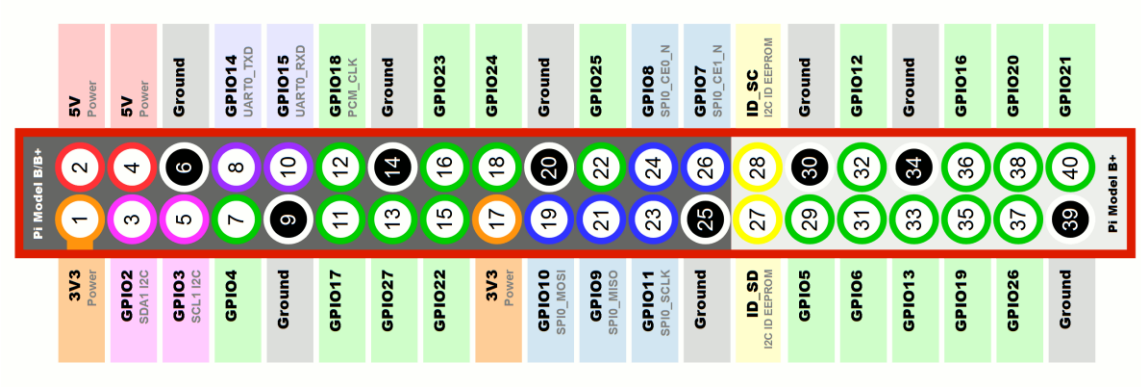
Node-RED ทำงานบน Node.js ทำให้เหมาะสำหรับการใช้งานกับ Raspberry Pi เนื่องจากใช้ทรัพยากรน้อย ขนาดไฟล์ไม่ใหญ่และ Node.js ยังทำหน้าที่เป็นตัวกลางให้ Raspberry Pi สามารถติดต่อกับ Web Browser และอุปกรณ์อื่นๆ ได้

2.8 Raspberry Pi 3 Model B+

Raspberry Pi (ออกเสียงว่า ราส-เบอร์-รี่-พาย) เป็นเครื่องคอมพิวเตอร์ขนาดจิ๋ว ที่มีขนาดเพียงเท่ากับบัตรเครดิต ที่สำคัญคือ ราสเบอร์รี่พายนี้มีราคาที่ถูกมาก เมื่อเทียบกับคอมพิวเตอร์เดสก์ท็อปปกติ คือมีราคาเพียงแค่หนึ่งพันกว่าบาทเท่านั้นเอง!!! แต่เห็นราคาเท่านี้ ทำงานได้เหมือนเครื่องคอมพิวเตอร์ทุกอย่างเลยนะครับ เราสามารถต่อ ราสเบอร์รี่พายนี้เข้ากับจอคอมพิวเตอร์หรือจอทีวีที่รองรับ HDMI หรือถ้าไม่มีพอร์ต HDMI ก็ไม่ต้องกังวล สามารถต่อผ่านสายสัญญาณวิดีโอปกติ (เส้นสีเหลือง) ได้เช่นกัน แต่ความละเอียดอาจจะต่ำกว่า

รูปที่ 7 Raspberry Pi3 B+

Raspberry Pi 3 Model B+ มี GPIO ทั้งหมด 40 Pin และที่ใช้สั่งควบคุมในการทำงานทั้งหมด 17 Pin (ที่เป็นวงกลมสีเขียว) ในการใช้งาน GPIO มีสิ่งที่ต้องพึงระมัดระวังให้มากเป็นพิเศษก็คือเรื่อง ในการต่อสาย เพราะหากมีการต่อผิดพลาด เช่น นำไฟ 5 โวลต์ มาต่อเข้ากับ Port GPIO ที่เป็นวงกลมสีเขียว อาจทำให้เกิดความเสียหายต่อ Raspberry Pi ทำให้ไม่สามารถใช้งานได้อีกต่อไป และไม่สามารถเปลี่ยน หรือ ซ่อมแซมได้ ดังนั้นก่อนต่อสายควรตรวจสอบความถูกต้อง ในการใช้งาน Pin ต่างๆ



รูปที่ 8 GPIO Raspberry Pi3 B+

2.10 HW-694 TTL

HW-694 TTL เป็นโมดูลวงจรอิเล็กทรอนิกส์ที่ใช้สำหรับวัดและแปลงสัญญาณ TTL (Transistor-Transistor Logic) โมดูลนี้ประกอบด้วยวงจรหลายส่วน ดังนี้

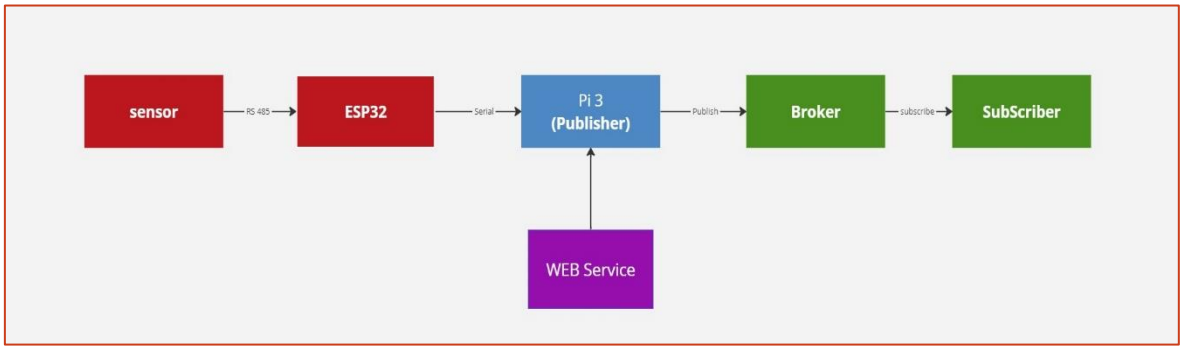
1. วงจรอินพุต: วงจรนี้ทำหน้าที่รับสัญญาณ TTL จากอุปกรณ์ภายนอก สัญญาณ TTL เป็นสัญญาณดิจิทัลที่มีสองระดับแรงดันไฟฟ้า คือ HIGH (ประมาณ 5 โวลต์) และ LOW (ประมาณ 0 โวลต์)
2. วงจรปรับระดับ: วงจรนี้ทำหน้าที่ปรับระดับแรงดันไฟฟ้าของสัญญาณ TTL ให้เข้ากันได้กับวงจรภายในของโมดูล HW-694 TTL
3. วงจรจับเวลา: วงจรนี้ทำหน้าที่วัดระยะเวลาของพัลส์สัญญาณ TTL โมดูล HW-694 TTL สามารถวัดระยะเวลาของพัลส์ได้หลายรูปแบบ เช่น ความกว้างของพัลส์ (Pulse Width) ความถี่ของพัลส์ (Pulse Frequency) และคาบของพัลส์ (Pulse Period)
4. วงจรแปลงสัญญาณ: วงจรนี้ทำหน้าที่แปลงสัญญาณ TTL ที่วัดได้เป็นสัญญาณอนาล็อก สัญญาณอนาล็อกนี้สามารถนำไปแสดงผลบนหน้าจอ LCD หรือส่งไปยังอุปกรณ์อื่นๆ เพื่อประมวลผลต่อไป
5. วงจรเอาต์พุต: วงจรนี้ทำหน้าที่ส่งสัญญาณอนาล็อกที่แปลงได้ไปยังอุปกรณ์ภายนอก

รูปที่ 9 Model HW-*694*

บทที่ 3
การออกแบบและขั้นตอนการทดลอง

3.1 การออกแบบ

ผู้จัดทำได้ทำการออกแบบโดยเริ่มจากการออกแบบ Workflow Diagram เพื่อให้เห็นรูปแบบการทำงานของระบบโดยรวมก่อนที่จะเป็นเป็นส่วนย่อย ๆตามนั้นตอนการทดลองต่อไป ซึ่ง Workflow Diagram เป็นตามรูปด้านล่าง



รูปที่ 10 Workflow Diagram *การทำงานทั้งระบบ*

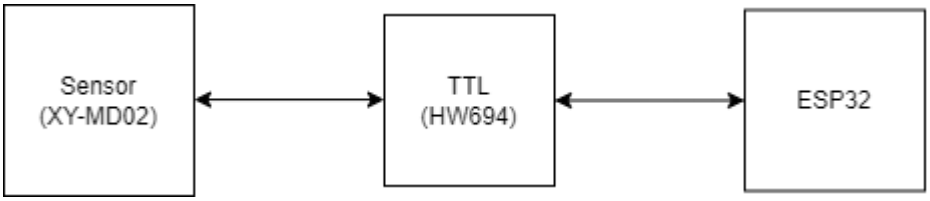
การ Workflow Diagram จะเห็นภาพรวมว่ามีรอยต่อระว่างแต่ละส่วนแบ่งออกเป็น 4 ส่วน ซึ่งทั้ง 5ส่วนนี้ก็จะแบ่งเป็นการทดลองย่อย ๆอีก 4 ส่วน เช่นกัน

3.2 ขั้นตอนในการทดลอง

3.2.1 ทดลองการอ่านค่าจากเซ็นเซอร์ โดยใช้ ESP 32

อุปกรณ์

1. ESP32
2. เซ็นเซอร์ XY-MD02
3. โมดูลแปลง RS384 เป็น TTL (Modbus (RS-485) level serial UART module)
4. สายไฟ
5. แหล่งจ่ายไฟ 5V



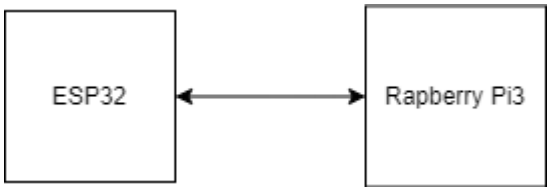
รูปที่ 11 Workflow Diagram ทดลองการอ่านค่าเซ็นเซอร์

ทำการต่อและเขียน code เพื่ออ่านค่าของเซ็นเซอร์โดยที่ในโค๊ด ต้องกำหนดค่า (Slave ID, Function Code,Address,Quantity, Baud rate) ซึ่งผู้จัดทำได้อัพโหลดโค๊ดทดลองส่วนนี้ไว้ที่ Github Download:https://github.com/KPE-Chaiwat/MQTT_ProjectSkill4/blob/main/sent_rx/sent_rx.ino

ซึ่งรูปแบบการส่งข้อมูลนั้นผู้จัดทำได้ทำการส่งข้อมูลแบบชนิด Char

3.2.2 ทดลองการรับส่งข้อมูลระหว่าง ESP 32 กับ Raspberry Pi 3

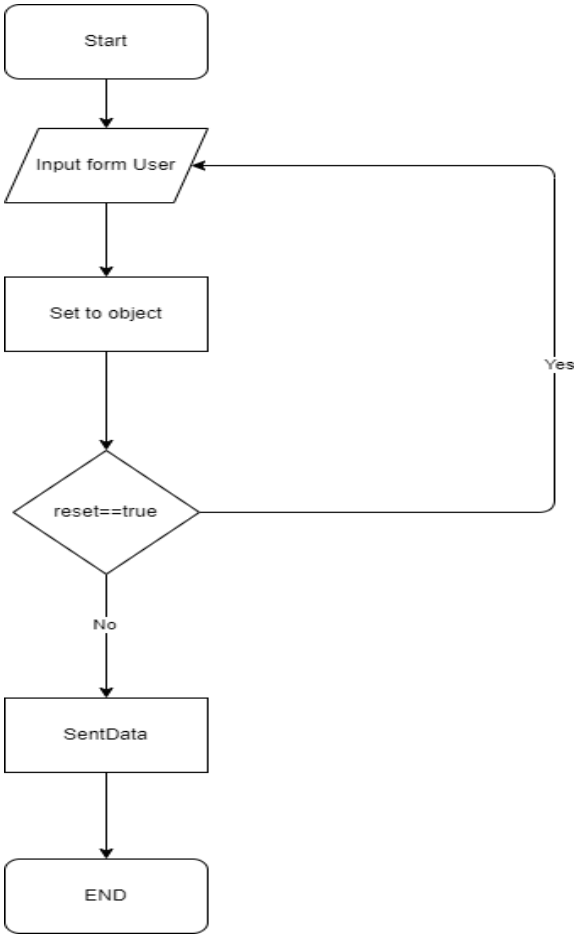
การทดลองนี้เป็นการทดสอบว่าจะสามารถรับ-ส่งข้อมูลจาก ESP32 ไปยัง Raspberry Pi3 ซึ่งการรับ-ส่งข้อมูลนี้จะส่งผ่านสาย Micro USB 2.0 ในการทดลองนี้เป็นไปตามรูป workflow ด้านล่าง



รูปที่ 12 Workflow Diagram *รับ-ส่งข้อมูลจาก* ESP32 *ไปยัง* Raspberry Pi3

3.2.3 ทดลองการเขียนโค๊ดเพื่อและทำหน้าเว็บแล้วส่งค่าไปยัง ESP32

การทำงานคือผู้ที่เข้าเว็บสามารถกำหนดค่าของ (Slave ID, Function Code,Address,Quantity, Baud rate) ได้และการเขียนหน้าเว็บโดยใช้ Node Red ซึ่งใช้ภาษา Java Script หลักการทำงานของหน้าเว็บเป็นไปตามรูป Flowchart

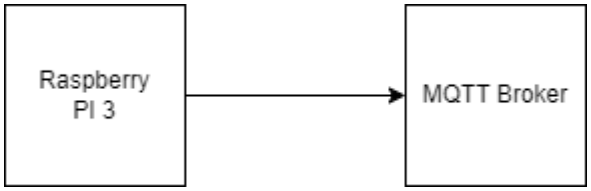


รูปที่ 13 Flowchart *การทำงานของ* Web Config

3.2.4 ทดลองการเขียนโค๊ดเพื่อให้ Raspberry Publish ข้อมูลไปยัง MQTT Broker

ในการทดลองนี้เป็นทดลองที่เขียนโค๊ดต่อจากข้อ 3.2.3 ซึ่งในการทดลองนี้ส่วนที่สำคัญคือการกำหนดค่า และรูปแบบการทำงานเป็นไปตาม Flowchart

1. กำหนด Server : broker.hivemq.com และกำหนด Port: 1883
2. กำหนด QOS เป็น 1 และการ Retain เป็น True
3. กำหนด Topic : ESE/Skill4



รูปที่ 14 Flowchart การ publish

จากการทดลองทั้งต้องทำทุกส่วนมาประกอบเข้ากันให้ได้ตาม รูปที่* Workflow Diagram การทำงานทั้งระบบ เพื่อให้ได้ตามที่ได้ออกแบบไว้

บทที่ 4

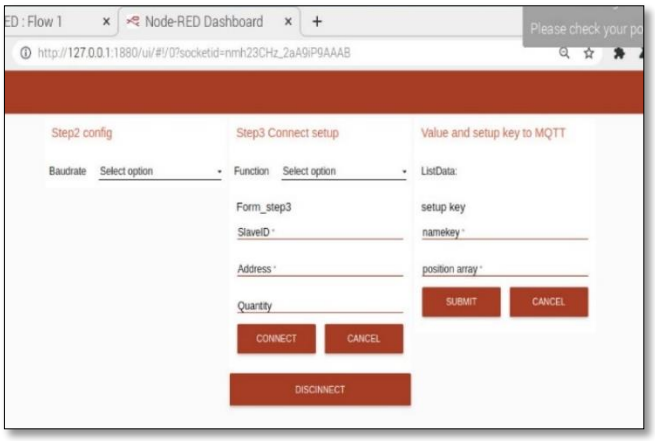
ผลการดำเนินงาน

ผลของการเดินเนินการส่วนแรกคือส่วนของ 3.2.1 รวมไปถึงส่วนของ 3.2.2 เข้าด้วยกัน ในการอ่านค่านั้นได้ค่า Char และกำหนดให้ข้อมูลจากเซ็นเซอร์ส่งมาทุกๆ 1 วินาทีและเขียนโค๊ดบนESP 32 ทำเปลี่ยนรูปส่งข้อมูลในรูปแบบของ String เช่น “[100,200,””,””]” เพื่อนส่งข้อมูลต่อไปยัง Raspberry Pi3 เหตุที่ทำการเปลี่ยนรูปแบบข้อมูลเช่นนี้เพราะว่างานต่อการกรองและง่ายต่อการเขียนโค๊ดในส่วนของ Node Red ผลที่ได้เป็นดังตาราง

เซ็นเซอร์	ผลจากการอ่านค่า
XY-MD02 SHT20	[“”,”256”,”540”]
Soil Humidity Sensor PR-3000-H-N01	[“0”,”2.5”,”255”,”255”]

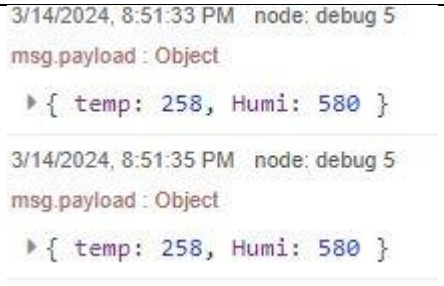
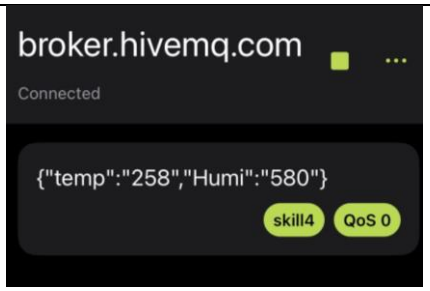
ตารางที่ 4 แสดงผลลัพธ์การอ่านค่าจากเซ็นเซอร์

ในส่วนของการทดลอง 3.2.3 นั้นเป็นการเขียน Node Red และผลที่ได้เป็นดังรูป และสามารถส่งค่าไปยังESP32ได้ซึ่งได้กำหนดรูปแบบได้คือ{“Slave_ID”:””,“Function_Code”:” ”,”Address”:””,”Quantity”:” ”, “Baud rate”:” ” } ซึ่งใน ESP32 สามารถใช้ library ของ JsonDocument เพื่อนำข้อมูลไปกำหนดการอ่านค่าเซ็นเซอร์ต่อไป



รูปที่15 Workflow Diagram *รับ-ส่งข้อมูลจาก* ESP32 *ไปยัง* Raspberry Pi3

จากที่ผู้ใช้งานทำการเลือกค่าที่กำหนด (Config) ค่าตัวแปรในการอ่านค่าต่างๆและนำข้อมูลที่ได้ไปPublish ต่อตามการทดลองที่ 3.2.4 ซึ่งสามารถทดสอบและได้ผลทดลองในการทดลองซึ่งผู้ทดลองได้ทดสอบกับการส่งข้อมูลด้วยเซ็น XY-MD02 SHT20 เท่านั้น ดังนี้

Publish	Subscribe
	

ตารางที่ 4 แสดงผลลัพธ์การอ่านค่าจากเซ็นเซอร์

ในส่วนของการ Subscribe ผู้จัดทำได้ใช้โทรศัพท์ที่ติดตั้งแอฟพิเคชั้น My MQTT ในการเป็นตัว Subscriber ข้อมูลจะเห็นได้ว่าข้อมูลมีค่าที่ตรงกัน

บทที่ 5

สรุป อภิปรายผล

จากที่ได้ศึกษาและทดลองการอ่านค่าของเซ็นเซอร์ที่มีโปโตคอลแบบ Modbus(RS-485) นั้น สามารถนำความรู้ไปประยุกต์การเขียนโค๊ดครั้งเดียวแล้วสามารถเปลี่ยนเซ็นเซอร์แล้วยังสามารถอ่านค่าได้ และออกแบบการเชื่อมต่อระหว่างสองโปรโตคอลคือ Modbus(RS-485) เข้ากับโปรโตคอล MQTT โดยที่ใช้ Raspberry Pi 3 เป็นส่วนที่เชื่อมทั้งสองส่วนเข้าหากัน อีกส่วนที่สำคัญไม่แพ้กันคือ ส่วนของไมโครคอลโทเลอร์(ESP32) จำทำหน้าที่อ่านค่าจากเซ็นเซอร์ถึงแม้ว่าจะเปลี่ยนตัวเองเซ็นไปแล้วก็ตามก็ยังสามารถอ่านค่าได้และการที่ไมโครคอลโทเลอร์ (ESP32) จะอ่านค่าได้นั้นต้องถูกควบคุมโดย Raspberry Pi 3 อีกทีซึ่งเป็นการควบคุมที่ว่าคือการกำหนดค่าของการอ่านโดยผู้ใช้งาน(Config) แล้วผลลัพธ์คือค่าที่ได้จากการอ่านของเซ็นเซอร์ซึ่งจะถูกผู้ใช้งานกำหนดหัวข้อของข้อมูลก่อนที่จะถูกส่งออกไปที่โบกเกอร์(MQTT Broker)หรือที่เรียกว่า Publish

จากโครงงานนี้ผู้จัดทำยังไม่สามารถได้ให้ผู้ใช้งานสามารถกำหนดหัวข้อในการส่งออก (Topic to publish) ซึ่งถ้าทำให้ก็จะสามารถจัดการเรื่องการเพิ่มของขนาดของ MQTT โปรโตคอลได้ซึ่งเป็นข้อดีของโปรโตคอล MQTT และข้อมูลที่ทำการส่งนั้นผู้จัดทำยังไม่สามารถทำการเข้ารหัสตามมาตรฐานการเข้ารหัสของข้อมูลเพื่อความปลอดภัยของข้อมูลได้

อีกทั้งการที่จะเข้าไปกำหนดค่าของการอ่านค่าของเซ็นเซอร์นั้นอุปกรณ์ต้องอยู่ในอินเตอร์เน็ตวงเดียวกันถึงจะสามารถเข้าไปทำหนดการอ่านค่าของเซ็นเซอร์ได้แต่ก็แลกกับการที่สามารถใช้อุปกรณ์อะไรก็ได้ที่สามารถเล่นอินเตอร์เน็ตได้มากำหนดค่าเช่น Tablet Laptop เป็นต้น

เอกสารอ้างอิง

[1] XY-MD02 User Manual https://github.com/banpot408/UNO_XY_MD02

[2] ESP 32 https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[3] MQTT :https://iiot.riverplus.com/mqtt/

[4] https://i-transform.biz/mqtt/

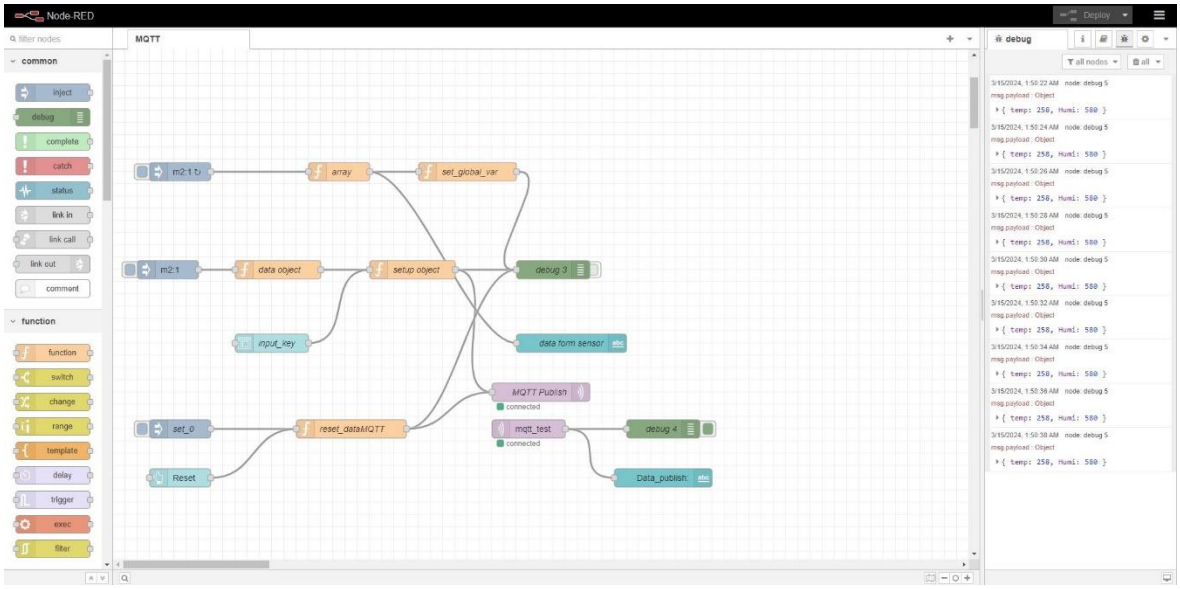
[5] Nodered:https://pantamitsombaddee.blogspot.com/p/node-red-node-red-apis-application.html

ภาพผนวก

```
void loop()
{
  setSlaveID();

  uint8_t result;
  uint16_t data[2];

//  result = node.readInputRegisters(1, 2);
result = node.readInputRegisters(Address, Quanitity);
  if (result == node.ku8MBSuccess)
  {
    //    Serial.print("Temp: ");
    //    Serial.println(node.getResponseBuffer(0) / 10.0f);
    //    Serial.print("Humi: ");
    //    Serial.println(node.getResponseBuffer(1) / 10.0f);
    //    Serial.println();

    for (int i = 0; i < Quanitity; i++) {
      sendMessage[i]= node.getResponseBuffer(i);
    }
 // sent Data
   Serial.println("["+sendMessage[0]+","+sendMessage[1]+","+sendMessage[2]+
","+sendMessage[3]+","+sendMessage[4]+","+sendMessage[5]+","+sendMessage[6]+
","+sendMessage[7]+","+sendMessage[8]+","+sendMessage[9]+","+sendMessage[10]+
"]"+".");
  }
  else
  {
    Serial.print(".\n");//ห้ามแก้
  }
  delay(1000);
}
```

รูปที่ 16 แสดง การเขียนโค๊ดเพื่ออ่านค่าเซ็นเซอร์

เป็นรูปที่แสดงถึงการเขียนโค๊ดเพื่ออ่านค่าและส่งค่าไปยัง Raspberry Pi 3 ซึ่งผู้จัดทำให้จัดทำให้มีรูปแบบการส่งข้อมูลให้เป็นแบบ Arrayมากที่สุดเนื่องจากง่ายต่อการใช้งานต่อนั้นเอง



รูปที่ 17 แสดง การเขียนโค๊ดเพื่ออ่านค่าเซ็นเซอร์

เป็นรูปที่แสดงถึงการเขียน Node red เพื่อรับ-ส่งข้อมูลระหว่าง ESP 32 กับ Raspberry Pi 3 และโค๊ดเดียวกันนี้ก็ยังสามารถPublish ข้อมูลไปยัง MQTT broker (HiveMQ)