การพัฒนาไอโอทีเกตเวย์แบบแยกส่วนอิสระรองรับหลายอินเตอร์เฟซ

DEVELOPMENT MODULAR IOT GATEWAY WITH MULTIPLE INTERFACE

โดย

นายขัตติยะ ไกยะราช

นางสาวบัณฑิตา บัวทอง

นางสาวโสภิตา ท่าพริก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2567

การพัฒนาไอโอทีเกตเวย์แบบแยกส่วนอิสระรองรับหลายอินเตอร์เฟซ

DEVELOPMENT MODULAR IOT GATEWAY WITH MULTIPLE INTERFACE

โดย

นายขัตติยะ	ไกยะราช	64010077
นางสาวบัณฑิตา	บัวทอง	64010453
นางสาวโสภิตา	ท่าพริก	64010951

อาจารย์ที่ปรึกษา

ดร. พีระเมศร์ โชติกวีกิจญาดา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2567

ปริญญานิพนธ์ปีการศึกษา 2567

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาไอโอทีเกตเวย์แบบแยกส่วนอิสระรองรับหลายอินเตอร์เฟซ

DEVELOPMENT MODULAR IOT GATEWAY WITH MULTIPLE INTERFACE

ผู้จัดทำ

1. นายขัตติยะ ไกยะราช 64010077
2. นางสาวบัณฑิตา บัวทอง 64010453
3. นางสาวโสภิตา ท่าพริก 64010951

..... **อาจารย์ที่ปรึกษา**

(ดร. พีระเมศร์ โชติกวีกิจญาดา)

กิตติกรรมประกาศ

การดำเนินปริญญานิพนธ์เรื่อง “การพัฒนาไอโอทีเกตเวย์แบบแยกส่วนอิสระรองรับหลายอินเตอร์เฟซ” สามารถดำเนินการจนสำเร็จลุล่วงไปได้ด้วยดี ด้วยความกรุณา และความช่วยเหลือจากอาจารย์ที่ปรึกษา และผู้ที่เกี่ยวข้องในส่วนต่าง ๆ ทางผู้จัดทำขอขอบคุณ ดร. พีระเมศร์ โชติกวีกิจญาดา ที่ได้กรุณาให้คำแนะนำ ความรู้ ชี้แนะแนวคิดและนำแนวทางที่ดีที่เป็นประโยชน์ต่อการแก้ไขปัญหาในการทำงาน รวมถึงสอดแทรกทักษะประสบการณ์ต่าง ๆ ให้กับผู้จัดทำ ทางผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้

ขอขอบคุณคณาจารย์และเจ้าหน้าที่ประจำภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่าน ที่ได้อบรมสั่งสอนประสิทธิ์ประสาทวิชา ความรู้ และประสบการณ์ให้แก่ผู้จัดทำ

สุดท้ายนี้ขอกราบขอบพระคุณบิดา มารดา และครอบครัวที่ให้การสนับสนุน ความรัก ความห่วงใย รวมถึงกำลังใจที่สำคัญเสมอมา และที่สำคัญคือสนับสนุนให้โอกาสทางด้านการศึกษาอันมีค่ายิ่งแก่ผู้จัดทำ

นายขัตติยะ ไกยะราช
นางสาวบัณฑิตา บัวทอง
นางสาวโสภิตา ท่าพริก
ผู้จัดทำ

การพัฒนาไอโอทีเกตเวย์แบบแยกส่วนอิสระรองรับหลายอินเตอร์เฟซ

DEVELOPMENT MODULAR IOT GATEWAY WITH MULTIPLE INTERFACE

โดย นายขัตติยะ ไกยะราช 64010077
นางสาวบัณฑิตา บัวทอง 64010453
นางสาวโสภิตา ท่าพริก 64010951

อาจารย์ที่ปรึกษา ดร. พีระเมศร์ โชติกวีกิจญาดา

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการพัฒนาระบบ IoT ที่สามารถรวบรวมมาตราฐานการสื่อสารต่างๆให้ทำงานร่วมกันได้อย่างอิสระ โดยประกอบไปด้วย ระบบประมวลผลส่วนกลาง (Core Board) ทำหน้าที่เป็น IoT gateway เชื่อมต่อระหว่างอินเตอร์เฟซ Long Range (LoRa), Wireless Fidelity (Wi-Fi) และ Bluetooth Low Energy (BLE) โดยใช้ชิพประมวลผล STM32H563Zi บนสถาปัตยกรรม ARM Cortex M33 มีการออกแบบ slot ไว้รองรับมาตรฐานการสื่อสารอื่น ๆ นอกเหนือจากนั้น มีการพัฒนา web platform สำหรับ monitor ข้อมูลที่ต้องการ

ABSTRACT

This thesis presents the development of an IoT system that integrates various communication standards to work together seamlessly. The Core Board serves as the IoT gateway, connecting LoRa, Wi-Fi, and BLE interfaces, and is powered by the STM32H563Zi microcontroller based on the ARM Cortex M33 architecture. The system is designed with slots to support additional communication standards and includes a web platform for data monitoring.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

จากอดีตจนถึงปัจจุบัน การพัฒนาโปรโตคอลการสื่อสารไร้สาย สำหรับใช้งานร่วมกับ IoT เช่น Zigbee, Z-Wave, BLE (Bluetooth Low Energy), LoRa, NB-IoT, และ MQTT ซึ่งแต่ละชนิด มีลักษณะการใช้งานที่แตกต่างกันไป ตามสภาพแวดล้อมที่ใช้งาน ซึ่งอุปกรณ์ IoT Gateway แบบเดิม ที่ออกแบบมาในรูปแบบคงที่ (Fixed) มักรองรับโปรโตคอลได้เพียงบางชนิดเท่านั้น และขาดความยืดหยุ่นในการปรับเปลี่ยนหรืออัพเกรดเพื่อรองรับโปรโตคอลใหม่ๆ หากผู้ใช้งานต้องการรองรับโปรโตคอลที่พัฒนาใหม่หรือการเปลี่ยนแปลงทางเทคโนโลยีในอนาคต พวกเขาจำเป็นต้องเปลี่ยนอุปกรณ์ Gateway ทั้งหมด ซึ่งไม่เพียงแต่เพิ่มค่าใช้จ่าย แต่ยังทำให้เกิดข้อจำกัดในด้านการปรับปรุงประสิทธิภาพของระบบ

จากปัญหาดังกล่าว ทางผู้จัดทำจึงเริ่มพัฒนา IoT Gateway เพื่อยกระดับมาตรฐาน โดยที่ออกแบบในลักษณะที่เป็นโมดูลาร์ (Modular) ซึ่งสามารถปรับเปลี่ยนหรือเพิ่มโมดูลที่รองรับโปรโตคอลการสื่อสารชนิดต่างๆ ได้อย่างยืดหยุ่น ผู้ใช้งานสามารถเลือกเพิ่มหรือเปลี่ยนโมดูลใหม่ตามโปรโตคอลที่ต้องการใช้งานได้อย่างสะดวก โดยไม่จำเป็นต้องเปลี่ยนอุปกรณ์ Gateway ทั้งชุด ซึ่งจะช่วยลดค่าใช้จ่ายในการอัพเกรดระบบ และเพิ่มความสามารถในการแข่งขันของโซลูชัน IoT ในอนาคต

1.2 วัตถุประสงค์

1) ศึกษาการพัฒนาเฟิร์มแวร์ (Firmware) บนระบบปฏิบัติการเซฟเฟอร์ (Zephyr OS)
2) ศึกษาหลักการทำงานและการเขียนเว็บแอปพลิเคชัน (Web Application) และแสดงผลของอุปกรณ์ผ่านเว็บแอปพลิเคชัน (Web Application)
3) ศึกษาและออกแบบบอร์ดประมวลผลหลัก (Core Board) และออกแบบบอร์ดเสริม (Extension Board)

1.3 ขอบเขตของปริญญานิพนธ์

1) พัฒนาเฟิร์มแวร์สำหรับบอร์ดประมวลผลหลัก (Core Board) ให้สามารถทำงานเชื่อมต่อกับบอร์ดเสริม (Plug In Board) ในลักษณะการใช้งานแบบพร้อมใช้งาน (Plug and play) และเชื่อมต่อกับอินเทอร์เน็ตผ่านบอร์ดเซลลูลาร์บน Zephyr OS บนพื้นฐาน RTOS (Real Time Operation Systems)
2) ออกแบบและพัฒนาเว็บแอปพลิเคชัน (Web Application) ให้เชื่อมโยงกับบอร์ดประมวลผลหลัก (Core Board) เพื่อใช้ในการแสดงผลข้อมูลของเซนเซอร์ผ่านเว็บแอปพลิเคชันได้ (Web Application)
3) ออกแบบบอร์ดประมวลผลหลัก (Core Board) โดยใช้ชิปประมวลผล STM35H563Zi ในการประมวลผลบอร์ดเสริม (Extension Board) โดยมีการนำเซนเซอร์วัดอุณหภูมิ (Temperature Sensor) และเซนเซอร์ตรวจจับฝุ่น (Dust Sensor) มาประมวลผล

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 บลูทูธพลังงานต่ำ (Bluetooth low energy; BLE)

ชื่อ Bluetooth มาจากกษัตริย์แฮรัลด์ บลูทูธ (Harald Blatand) แห่งเดนมาร์ก ซึ่งทรงมีบทบาทสำคัญในการรวมกลุ่มชนที่ขัดแย้งกันในดินแดนของนอร์เวย์ สวีเดน และเดนมาร์กให้เป็นปึกแผ่น ชื่อนี้ได้รับแรงบันดาลใจจาก Jim Kardach แห่ง Intel ในปี 1997 ขณะพัฒนาระบบที่ช่วยให้อุปกรณ์เคลื่อนที่สามารถสื่อสารกับคอมพิวเตอร์ได้ โดย Kardach ได้อ่านนวนิยายเรื่อง “The Long Ships” ซึ่งกล่าวถึงชาวไวกิ้งและกษัตริย์แฮรัลด์ บลูทูธ นอกจากนี้ สัญลักษณ์ Bluetooth ยังมาจากการผสมผสานระหว่างอักษรรูนของชื่อย่อกษัตริย์แฮรัลด์ คือ Hagall (ᚼ) และ Bjarkan (ᛒ) ซึ่งแสดงถึงมรดกทางวัฒนธรรมที่สำคัญ แสดงดังรูปที่ 2.1

รูปที่ 2.1 สัญลักษณ์บลูทูธ

Bluetooth Low Energy เป็นส่วนหนึ่งของมาตรฐาน Bluetooth ที่ดูแลโดย Bluetooth SIG (Special Interest Group) ซึ่งก่อตั้งขึ้นในปี 1998 พลังงานต่ำเปิดตัวครั้งแรกในปี 2011 ในเวอร์ชัน 4.0 ของมาตรฐาน ปัจจุบันมีสมาชิกมากกว่า 30,000 คนใน Bluetooth SIG ที่มีวิสัยทัศน์ร่วมกันว่า "โลกที่เชื่อมต่อกันปราศจากสายไฟ" บลูทูธทำงานโดยใช้ย่านความถี่ ISM 2.4GHz (2.400 – 2483.5MHz) และอิงตามข้อกำหนด IEEE 802.15.1™ – การควบคุมการเข้าถึงสื่อไร้สาย (MAC) และทางกายภาพ (PHY) สำหรับเครือข่ายส่วนบุคคลไร้สาย (WPAN) [1]

2.1.1 คุณสมบัติของ BLE

- การใช้พลังงานที่น้อยลง
- เวลาเชื่อมต่อสั้น ประมาณ 3 ms
- อัตราการส่งข้อมูล 1 Mbps สูงสุด 2 Mbps ด้วย BT5.0
- การเข้ารหัส AES CCM 128 บิต, CRC 24 บิต
- ความไวของเครื่องรับ – 87 dBm – 93 dBm

2.1.2 มาตรฐานของ Bluetooth

วิทยุ Bluetooth Low Energy (LE) ได้รับการออกแบบมาสำหรับการทำงานที่ใช้พลังงานต่ำมาก การส่งข้อมูลมากกว่า 40 ช่องสัญญาณในย่านความถี่ ISM ที่ไม่มีใบอนุญาต 2.4GHz วิทยุ Bluetooth LE ช่วยให้นักพัฒนามีความยืดหยุ่นอย่างมากในการสร้างผลิตภัณฑ์ที่ตรงตามความต้องการการเชื่อมต่อเฉพาะของตลาด Bluetooth LE รองรับโทโพโลยีการสื่อสารที่หลากหลาย โดยขยายจากจุดต่อจุดไปสู่การออกอากาศ ทำให้เทคโนโลยี Bluetooth สามารถรองรับการสร้างเครือข่ายอุปกรณ์ขนาดใหญ่ที่เชื่อถือได้ แม้ว่าในตอนแรกจะเป็นที่รู้จักในด้านความสามารถในการสื่อสารของอุปกรณ์ แต่ปัจจุบัน Bluetooth LE ยังถูกนำมาใช้กันอย่างแพร่หลายในฐานะเทคโนโลยีการระบุตำแหน่งอุปกรณ์เพื่อตอบสนองความต้องการที่เพิ่มขึ้นสำหรับบริการระบุตำแหน่งในร่มที่มีความแม่นยำสูง ตอนนี้ Bluetooth LE มีคุณสมบัติที่ช่วยให้อุปกรณ์เครื่องหนึ่งสามารถกำหนดการมีอยู่ ระยะทาง และทิศทางของอุปกรณ์อื่นได้ แสดงดังตารางที่ 2.1 [2]

1. Bluetooth Classic แบบคลาสสิคที่รองรับ Data Rate สองแบบ ได้แก่ Basic Rate (BR) และ Enhance Data Rate (ER) โดยเวอร์ชั่นที่เป็นมาตรฐานนี้ คือ Bluetooth Version 1 – 3

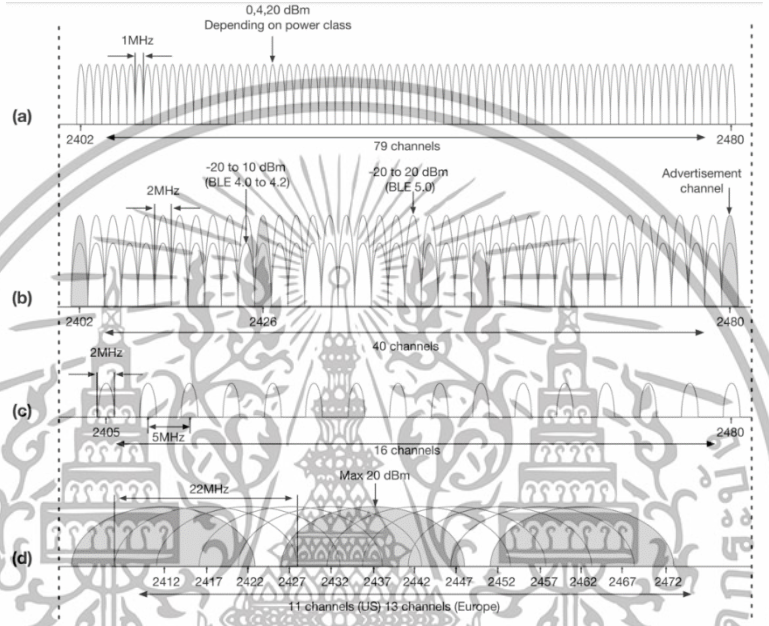
2. Bluetooth Low Energy (LE) เป็นบลูทูธแบบกินพลังงานต่ำ เหมาะกับการใช้ในอุปกรณ์ที่มี Battery ในตัว เพื่อความประหยัดพลังงานนั่นเอง ซึ่ง Bluetooth LE อาจจะไม่สามารถรับส่งข้อมูลจำนวนมาก แต่สามารถรองรับข้อมูลเสียง Audio ที่มีคุณภาพสูง และยืดหยุ่นกว่าแบบ Classic โดยเวอร์ชั่นที่เป็นมาตรฐานนี้ คือ Bluetooth Version 4 – 5

ตารางที่ 2.1 เปรียบเทียบความแตกต่างที่สำคัญระหว่าง Bluetooth แบบดั้งเดิมและ BLE

	Bluetooth Low Energy (BLE)	Bluetooth Basic Rate / Enhanced Data Rate
ปรับให้เหมาะสมสำหรับ	การส่งข้อมูลต่อเนื่องสั้น	
ย่านความถี่	ย่านความถี่ ISM 2.4 GHz (ใช้ 2.402 – 2.480 GHz)	ย่านความถี่ ISM 2.4 GHz (ใช้ 2.402 –2.480 GHz)
จำนวนช่อง	40 ช่อง ระยะห่าง 2MHz (3 ช่องโฆษณา / 37 ช่องข้อมูล)	
ความกว้าง/ระยะห่างของช่อง	2 MHz / 2 MHz	
การใช้ช่อง	สเปกตรัมกระจายความถี่กระโดด (FHSS)	สเปกตรัมกระจายความถี่กระโดด (FHSS)
การมอดูเลต	GFSK	
อัตราข้อมูล	LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE รหัส PHY (S=2): 500 Kb/s LE รหัส PHY (S=8): 125 Kb/s	
กำลังส่งสูงสุด	คลาส 1: 100 mW (+20 dBm) คลาส 1.5: 10 mW (+10 dbm) คลาส 2: 2.5 mW (+4 dBm) คลาส 3: 1 mW (0 dBm)	คลาส 1: 100 mW (+20 dBm) คลาส 2: 2.5 mW (+4 dBm) คลาส 3: 1 mW (0 dBm)
รูปแบบการเชื่อมต่อ	Point-to-Point (including piconet) Broadcast Mesh	Point-to-Point (including piconet)

แผนภาพด้านล่าง ภาพ (a) แสดงบลูทูธแบบดั้งเดิมโดยใช้ 79 ช่องสัญญาณที่มีความกว้าง 1MHz ภาพ (b) ไฮไลต์ BLE ด้วยเวอร์ชัน 4.0-4.2 ที่ตัดกับเวอร์ชัน 5.0 ซึ่งแสดงให้เห็นความแตกต่างในระดับพลังงาน BLE ใช้ 40 ช่องสัญญาณ เว้นระยะห่าง 2MHz และกว้าง 2MHz 37 จาก 40 ช่องใช้เพื่อถ่ายโอนข้อมูล โดยสามช่องสงวนไว้เป็น 'ช่องโฆษณา' ช่องสัญญาณพิเศษทั้งสามนี้ได้รับการคัดเลือกอย่างมีกลยุทธ์ในย่านความถี่ 2.4GHz โดยมีความถี่กลาง 2402MHz, 2426MHz และ

2480MHzการเลือกอย่างระมัดระวังมีจุดประสงค์เพื่อลดการรบกวนและหลีกเลี่ยงการทับซ้อนกับช่องสัญญาณ IEEE 802.11 b/g/n ที่ใช้กันทั่วไป รูปภาพ (c) แสดงถึง 16 ช่องสัญญาณที่ใช้กับเครือข่ายที่ใช้ IEEEE 802.15.4 จำนวนมาก เช่น ZigBee สุดท้าย สำหรับการเปรียบเทียบ รูปภาพ (d) แสดงถึงช่องสัญญาณ IEEE 802.11b™ โดยใช้ DSS พร้อมช่องสัญญาณกว้าง 22MHz แสดงดังรูปที่ 2.2



รูปที่ 2.2 แสดงความกว้างของช่องสัญญาณ

รูปแบบการมอดูเลต Gaussian Frequency-Shift Keying (GFSK) เดิมเป็นรูปแบบการมอดูเลตเพียงอย่างเดียวที่กำหนดไว้ เรียกว่าอัตราพื้นฐาน (BR) ที่มีอัตราข้อมูลทันทีที่ 1 Mbit/s FSK มอดูเลตรูปแบบคลื่นโดยตรงที่มีความถี่ต่างกันกับสัญลักษณ์ข้อมูลดิจิทัล โดยเปลี่ยนความถี่ที่จุดเริ่มต้นของแต่ละช่วงเวลาสัญลักษณ์ตามความจำเป็นตามลำดับบิตที่จะส่ง การคีย์การเปลี่ยนความถี่เกาส์เซียน (GFSK) ทำให้การเปลี่ยนระหว่างพัลส์ข้อมูลราบรื่นโดยใช้ฟิลเตอร์ ข้อดีจากการเปลี่ยนที่ราบรื่นยิ่งขึ้นในการกรอง ได้แก่ การลดความกว้างของสเปกตรัมและลดการรบกวนกับช่องสัญญาณใกล้เคียง การแลกเปลี่ยนนั้นต้องแลกกับการแทรกแซงระหว่างสัญลักษณ์ที่เพิ่มขึ้น ต่อมามีการเพิ่มเทคนิคการมอดูเลตโดยใช้ DQPSK และ 8PPSK ซึ่งเรียกว่าอัตราข้อมูลที่เพิ่มขึ้น (EDR) ซึ่งให้อัตราข้อมูลที่เพิ่มขึ้น 2 และ 3 Mbit/s เวอร์ชัน 3.0 ของมาตรฐานแนะนำอัตราข้อมูลสูงสุดที่ 24 Mbit/s

การเปลี่ยนแปลง มาตาฐานของ Bluetooth ในแต่ละเวอร์ชั่น ให้เหมาะสมกับการใช้งานที่หลากหลาย ตั้งแต่การลดการใช้พลังงาน การเพิ่มประสิทธิภาพการเชื่อมต่อ การรองรับปริมาณข้อมูลที่มากขึ้น รวมถึงการเพิ่มขีดความสามารถในการสื่อสารระยะไกลและการรองรับแอปพลิเคชันที่

ต้องการความเร็วในการถ่ายโอนข้อมูลสูง นอกจากนี้ ยังรวมถึงการพัฒนาด้านความปลอดภัยและการเข้ารหัสที่มีประสิทธิภาพมากขึ้น แสดงดังตารางที่ 2.2 [1]

ตารางที่ 2.2 รวบรวมการเปลี่ยนแปลงในเวอร์ชันต่างๆ ภายในมาตรฐาน Bluetooth

เวอร์ชั่นบลูทูธ	ชื่อ	อัตราข้อมูล (Mbps)	หมายเหตุ
1.0	อัตราพื้นฐาน (BR)	1	วางจำหน่าย 1999
1.2			
2.0	อัตราข้อมูลขั้นสูง (EDR)	2 และ 3	วางจำหน่าย 2004
2.1			เปิดตัว 2007การจับคู่แบบง่าย
3.0	ความเร็วสูง (HS)	สูงสุด 24 (เมื่อใช้ Wi-Fi)	วางจำหน่าย 2009
4.0		BT (สูงสุด 24) BLE (1)	เปิดตัว 2010 เพิ่ม LE
4.1		BT (สูงสุด 24) BLE (1)	วางจำหน่าย 2013
4.2		BT (สูงสุด 24) BLE (1)	เปิดตัวการสนับสนุน IPv6 ปี 2014
5.0		BT (สูงสุด 24) BLE(1 & 2)	เปิดตัวปี 2016 กำลังขับที่สูงขึ้นอัตราข้อมูลเพิ่มขึ้นเป็นสองเท่าสำหรับ LE
5.1		BT (สูงสุด 24) BLE(1 & 2)	เปิดตัวปี 2019ปรับปรุง AoA, AoD สำหรับตำแหน่งเพิ่มตาข่าย
5.2		BT (สูงสุด 24) BLE(1 & 2)	เปิดตัวLE Audio ปี 2020
5.3			HS ที่เลิกใช้แล้ว

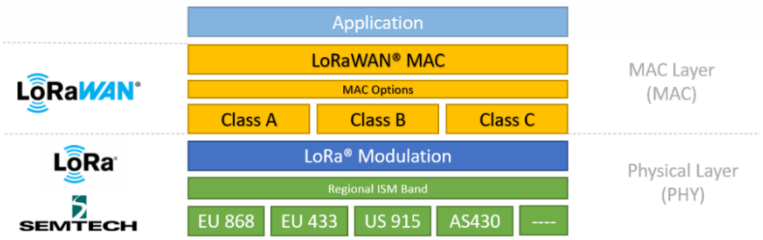
2.1.3 ข้อดีและประโยชน์ของบลูทูธ

ข้อดีและประโยชน์ต่าง ๆ ของบลูทูธ ดังนี้ [3]

1. ไม่ต้องเชื่อมต่ออุปกรณ์หรือไม่ต้องใช้สาย Cable
2. ราคาประหยัด
3. ในการเชื่อมต่อ ไม่ต้อง Set up เพราะเมื่ออุปกรณ์ทั้งสองที่รองรับ Bluetooth ทั้งคู่อยู่ใกล้ ก็สามารถจับสัญญาณอัตโนมัติ และสามารถเชื่อมต่อกันได้ทันที
4. มาตรฐานบลูทูธเป็น Standard ที่นิยมใช้กันเป็นจำนวณมาก ทั้งในอุปกรณ์ทั่วไปจนถึงอุปกรณ์ที่อยู่ในระดับสูง ถ้าหากใช้บลูทูธเหมือนกันก็สามารถเชื่อมต่อกันได้
5. อุปกรณ์บลูทูธจะไม่ถูกรบกวนจากอุปกรณ์ไร้สายอื่น เพราะมีเทคโนโลยี Frequenct Hopping ที่จะปรับเปลี่ยนความถี่ไปเรื่อย ๆ
6. การใช้งานส่วนใหญ่ของ Bluetooth มักจะใช้สำหรับแชร์ข้อมูลกัน ไม่ก็แชร์เสียง Audio สำหรับสื่อสารกันเป็นหลัก
7. สามารถเชื่อมต่ออุปกรณ์ Bluetooth ได้สูงสุด 7- 8 ตัว ในระยะ 10 เมตร เพื่อสร้างเป็น Personal Network

2.2 Long Range (LoRa)

LoRa (Long Range) เป็นเทคนิคการมอดูเลตที่ใช้ในเครือข่าย LoRaWAN สำหรับการสื่อสารไร้สายระยะไกลที่ใช้พลังงานต่ำบนย่านความถี่ต่ำกว่า 1GHz โดย LoRaWAN เป็นโปรโตคอลการควบคุมการเข้าถึงสื่อ (MAC) สำหรับเครือข่ายบริเวณกว้าง (WAN) ที่ใช้การสื่อสารไร้สายระยะไกลที่ใช้พลังงานต่ำเพื่อเชื่อมต่ออุปกรณ์ Internet of Things (IoT) ใช้เทคนิคการมอดูเลต LoRa ซึ่งช่วยให้สามารถสื่อสารระยะไกลได้ที่อัตราข้อมูลต่ำ และได้รับการออกแบบมาเพื่อใช้ในย่านความถี่ sub-1GHz ที่ไม่มีใบอนุญาต โดยทั่วไปแล้ว เครือข่าย LoRaWAN จะประกอบด้วยเกตเวย์ซึ่งเชื่อมต่อกับอินเทอร์เน็ต และอุปกรณ์ปลายทาง ซึ่งเป็นอุปกรณ์ IoT ที่สื่อสารกับเกตเวย์ แสดงดังรูปที่ 2.3



รูปที่ 2.3 โปรโตคอลของ LoRa

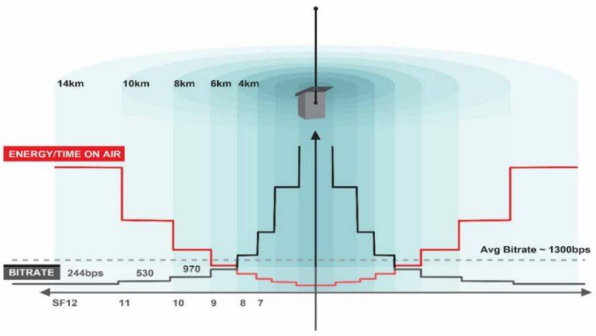
โปรโตคอล LoRaWAN ใช้โทโพโลยีแบบดาว (Star Topology) และรองรับทั้งอุปกรณ์เคลื่อนที่และอุปกรณ์ที่อยู่ประจำที่ โปรโตคอลนี้มีการจัดเตรียมกลไกสำหรับการสื่อสารที่ปลอดภัย เช่น การเปิดใช้งานแบบ Over-the-Air ที่ปลอดภัยและการเข้ารหัสแบบ End-to-End โปรโตคอลนี้ช่วยให้สามารถติดตั้งเครือข่าย IoT ที่มีขนาดใหญ่และต้นทุนต่ำได้ นอกจากนี้ยังเป็นมาตรฐานแบบเปิด (Open Standard) ที่ได้รับการดูแลและกำหนดมาตรฐานโดย LoRa Alliance ซึ่งเป็นสมาคมไม่แสวงหาผลกำไรที่ประกอบไปด้วยบริษัท องค์กร และบุคคลต่าง ๆ

ชั้น Media Access Control (MAC) ใน LoRaWAN ทำหน้าที่บริหารจัดการการเข้าถึงคลื่นความถี่วิทยุที่ใช้ร่วมกัน และประสานการสื่อสารระหว่างอุปกรณ์ในเครือข่าย โดยชั้นนี้มีหน้าที่จัดการกลไกการเปิดใช้งานอุปกรณ์ การสื่อสารที่ปลอดภัย รวมถึงการส่งและรับข้อความ

คุณสมบัติสำคัญอย่างหนึ่งของชั้น MAC ใน LoRaWAN คือการใช้กลไก Time - Division Multiple Access (TDMA) ซึ่งแบ่งคลื่นความถี่วิทยุออกเป็นช่องเวลาต่าง ๆ ที่จัดสรรให้กับอุปกรณ์แต่ละตัว กลไกนี้ช่วยให้การใช้คลื่นความถี่ที่มีอยู่อย่างจำกัดเป็นไปอย่างมีประสิทธิภาพ และลดโอกาสการชนกันของสัญญาณระหว่างการส่งข้อมูล

2.2.1 Spreading Factor

ใน LoRaWAN ค่า Spreading Factor (SF) เป็นพารามิเตอร์ที่ควบคุมการแลกเปลี่ยนระหว่างอัตราการส่งข้อมูล (Data Rate) และระยะทางในการสื่อสาร โดยค่า SF จะใช้ในการกระจายข้อมูลไปยังย่านความถี่ที่กว้างขึ้น ซึ่งจะช่วยเพิ่มอัตราส่วนสัญญาณต่อสัญญาณรบกวน (Signal-to-Noise Ratio: SNR) และทำให้การสื่อสารมีความทนทานต่อสัญญาณรบกวนและการแทรกแซงได้ดียิ่งขึ้น ค่า Spreading Factor มีความสัมพันธ์ผกผันกับอัตราการส่งข้อมูล หมายความว่าเมื่อค่า SF เพิ่มขึ้น อัตราการส่งข้อมูลจะลดลง และในทางกลับกัน ค่า SF จะอยู่ในช่วงระหว่าง 7 ถึง 12 โดยค่า SF ที่ต่ำกว่าจะบ่งบอกถึงอัตราการส่งข้อมูลที่สูงขึ้น แต่มีระยะการส่งข้อมูลที่สั้นกว่า ในขณะที่ค่า SF ที่สูงขึ้นจะบ่งบอกถึงอัตราการส่งข้อมูลที่ต่ำกว่า แต่มีระยะการส่งข้อมูลที่ไกลขึ้น เมื่อค่า SF ต่ำลง สัญญาณจะถูกกระจายไปในแถบความถี่ที่กว้างขึ้น ทำให้ตัวรับสามารถตรวจจับสัญญาณได้ด้วยอัตราส่วนสัญญาณต่อสัญญาณรบกวนที่สูงขึ้น ซึ่งช่วยให้อัตราการส่งข้อมูลเพิ่มขึ้น ในทางกลับกัน เมื่อค่า SF สูงขึ้น สัญญาณจะถูกกระจายไปในแถบความถี่ที่แคบลง ซึ่งส่งผลให้อัตราการส่งข้อมูลลดลง แต่จะช่วยเพิ่มระยะทางในการสื่อสาร แสดงดังรูปที่ 2.4

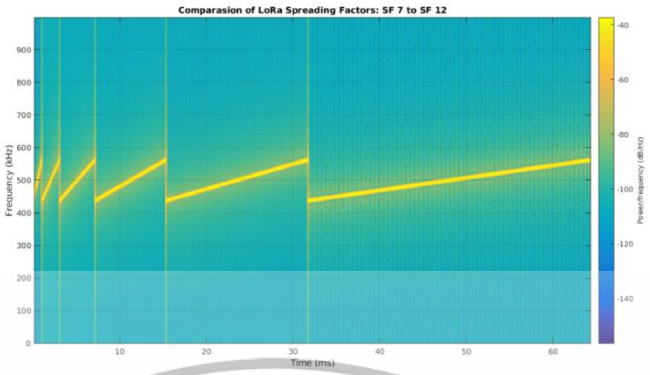


รูปที่ 2.4 แสดง Spreading Factor

ในเครือข่าย LoRaWAN อุปกรณ์ทั้งหมดจะใช้ย่านความถี่เดียวกัน ค่า Spreading Factor (SF) ถูกใช้เพื่อแยกการทำงานของอุปกรณ์แต่ละตัว ทำให้สามารถทำงานในย่านความถี่เดียวกันได้โดยไม่รบกวนกัน ค่า SF สามารถปรับได้แบบไดนามิกด้วยอัลกอริทึม Adaptive Data Rate (ADR) ซึ่งช่วยให้อุปกรณ์สามารถปรับอัตราการส่งข้อมูลและกำลังส่งสัญญาณตามสภาพแวดล้อมของคลื่นวิทยุและความต้องการของแอปพลิเคชัน การปรับนี้ช่วยให้เกิดการแลกเปลี่ยนที่เหมาะสมระหว่างระยะทางและปริมาณข้อมูลที่ส่งผ่าน

2.2.2 การมอดูเลต (Modulate) ของ LoRa

LoRa ใช้การมอดูเลตแบบกระจายสเปกตรัม (Spread-Spectrum Modulation) ซึ่งหมายถึงการกระจายข้อมูลไปในย่านความถี่ที่กว้าง ช่วยให้มีอัตราส่วนสัญญาณต่อสัญญาณรบกวน (SNR) ที่สูงขึ้น และมีความทนทานต่อสัญญาณรบกวนได้ดียิ่งขึ้น LoRa ใช้เทคนิค Chirp Spread Spectrum (CSS) ซึ่งความถี่ของสัญญาณจะเปลี่ยนไปตามเวลา (หรือเรียกว่า "Chirp") วิธีนี้ช่วยให้สามารถใช้แบนด์วิดท์ของสัญญาณที่กว้างขึ้นและมีอัตราการส่งข้อมูลที่สูงกว่าเทคนิคการมอดูเลตแบบ Frequency-Shift Keying (FSK) แบบดั้งเดิม เทคนิคการมอดูเลตของ LoRa ใช้อัตราการส่งข้อมูลที่ต่ำ โดยปกติอยู่ระหว่าง 0.3 kbps ถึง 50 kbps ซึ่งช่วยลดการใช้พลังงานและเพิ่มระยะทางในการสื่อสารได้ สัญญาณ LoRa สามารถเดินทางในระยะไกลด้วยพลังงานต่ำ โดยสามารถครอบคลุมระยะทางได้สูงสุดถึง 15 กิโลเมตรในพื้นที่เปิดโล่ง และระหว่าง 2-5 กิโลเมตรในเขตเมือง นอกจากนี้ LoRa ยังใช้เทคนิค Adaptive Data Rate (ADR) ซึ่งช่วยให้อุปกรณ์สามารถปรับอัตราการส่งข้อมูลและกำลังส่งตามสภาพแวดล้อมของคลื่นวิทยุและความต้องการของแอปพลิเคชัน ซึ่งช่วยให้เกิดการปรับที่เหมาะสมระหว่างระยะทางและปริมาณข้อมูลที่ส่งผ่าน เทคนิคการมอดูเลต LoRa เหมาะอย่างยิ่งสำหรับการใช้งาน IoT ที่ต้องการการสื่อสารระยะไกลและใช้พลังงานต่ำ เช่น การวัดแสงอัจฉริยะ การติดตามสินทรัพย์ และเครือข่ายเซ็นเซอร์ แสดงดังรูปที่ 2.5

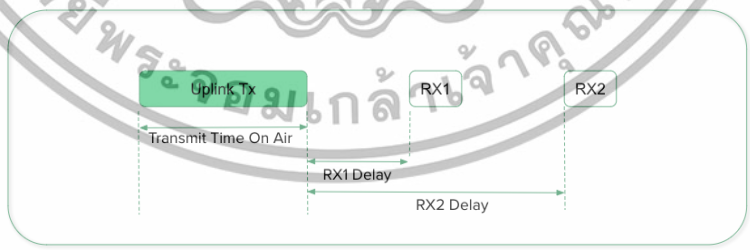


รูปที่ 2.5 การเปรียบเทียบปัจจัยการแพร่กระจายของ LoRa SF7 ถึง SF12

2.2.3 Device Classes

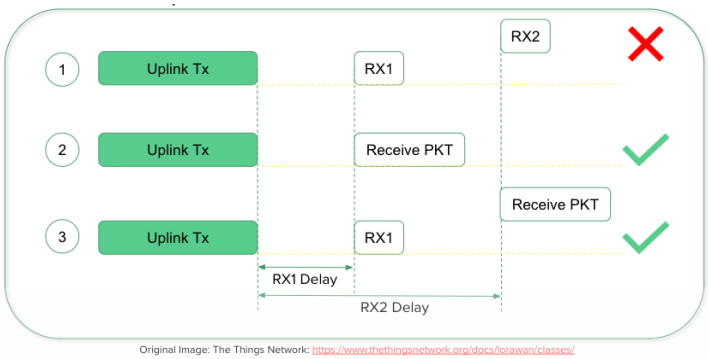
ชั้น MAC ของ LoRaWAN ยังประกอบด้วยอุปกรณ์ประเภท Class A, B และ C ซึ่งกำหนดพฤติกรรมการสื่อสารระหว่างอุปกรณ์กับเกตเวย์ ช่วยให้อุปกรณ์สามารถใช้พลังงานจากแบตเตอรี่และมีรอบการทำงานที่ต่ำ (Low Duty - Cycle)

Class A เป็นประเภทที่ใช้พลังงานอย่างมีประสิทธิภาพมากที่สุด เหมาะสำหรับการใช้งานที่ขับเคลื่อนด้วยแบตเตอรี่ โดยอุปกรณ์เหล่านี้จะมีรอบการทำงานที่ต่ำ หมายถึงการอยู่ในโหมดสลีปที่ใช้พลังงานต่ำเกือบตลอดเวลา และจะตื่นขึ้นมาเพียงชั่วคราวเพื่อฟังข้อความขาเข้าหรือส่งข้อมูลของตัวเองเท่านั้น อุปกรณ์ Class A จะมีหน้าต่างการรับข้อมูล 2 ครั้ง ได้แก่ ครั้งแรกหลังจากสิ้นสุดการส่งข้อมูล และครั้งที่สองหลังจากนั้น 8 วินาที วิธีนี้ช่วยให้อุปกรณ์ประหยัดพลังงานได้มากขึ้นในขณะที่ยังคงสามารถรับข้อความขาเข้าจากเกตเวย์ได้ แสดงดังรูปที่ 2.6



รูปที่ 2.6 Class A ของ LoRaWAN

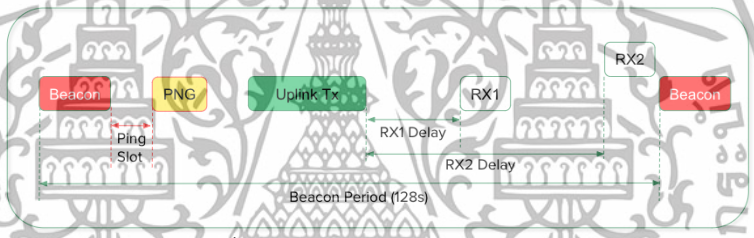
Class B เป็นประเภทที่มีรอบการทำงานที่สูงกว่าอุปกรณ์ประเภท Class A เล็กน้อย เนื่องจากอุปกรณ์เหล่านี้จะตื่นขึ้นเป็นระยะเพื่อฟังข้อความขาเข้า แม้ในขณะที่ไม่ได้ทำการส่งข้อมูลก็ตาม สิ่งนี้ช่วยให้การสื่อสารแบบสองทิศทางมีประสิทธิภาพมากขึ้น และรองรับการใช้งานเช่น การอัปเดตเฟิร์มแวร์แบบ Over-the-Air (OTA) แสดงดังรูปที่ 2.7



Original Image: The Things Network: https://www.thethingsnetwork.org/docs/lorawan/classes/

รูปที่ 2.7 Class B ของ LoRaWAN

Class C มีรอบการทำงานที่สูงที่สุด เนื่องจากอุปกรณ์เหล่านี้จะเปิดรับฟังข้อความขาเข้าตลอดเวลาเมื่อไม่ได้ทำการส่งข้อมูล ซึ่งช่วยให้การสื่อสารเกิดขึ้นได้อย่างรวดเร็วที่สุด แต่ก็ส่งผลให้มีการใช้พลังงานสูงที่สุดด้วย อุปกรณ์ประเภทนี้เหมาะสำหรับการใช้งานที่ต้องการให้สามารถเข้าถึงและตอบสนองได้ตลอดเวลา แสดงดังรูปที่ 2.8



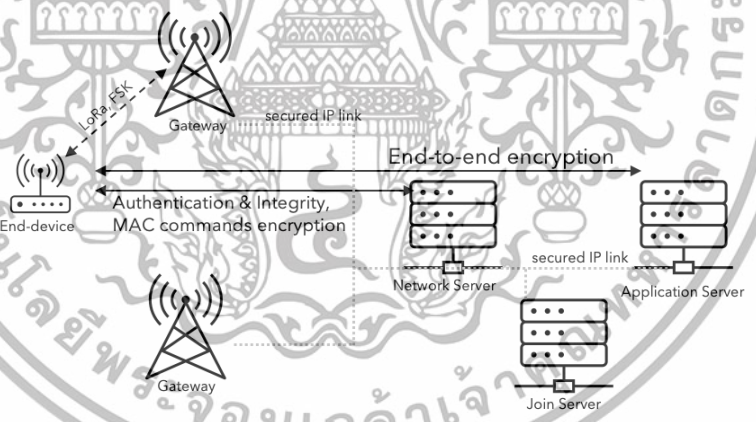
รูปที่ 2.8 Class C ของ LoRaWAN

2.2.4 การเปิดใช้งานอุปกรณ์

การเปิดใช้งานแบบ Over-the-Air Activation (OTAA) เป็นกลไกใน LoRaWAN ที่ช่วยให้อุปกรณ์สามารถเข้าร่วมเครือข่ายได้อย่างปลอดภัยและรับข้อมูลรับรองที่จำเป็นสำหรับการสื่อสารกับเครือข่าย โดยใช้การเข้ารหัสแบบ End-to-End ซึ่งช่วยป้องกันการเข้าถึงหรือดัดแปลงข้อมูลโดยไม่ได้รับอนุญาต OTAA เป็นวิธีที่แนะนำเมื่อเทียบกับ ABP (Authentication by Personalization) ซึ่งควรใช้เฉพาะในการทดสอบเบื้องต้นในห้องทดลองเท่านั้น เมื่ออุปกรณ์ใหม่ถูกเปิดใช้งานและต้องการเข้าร่วมเครือข่าย มันจะใช้กระบวนการ OTAA ในการขอสิทธิ์เข้าถึง โดยอุปกรณ์จะสร้างรหัสประจำตัวที่ไม่ซ้ำกัน (DevEUI) และรหัสเข้ารหัสเฉพาะอุปกรณ์ (AppKey) แล้วส่งข้อความคำขอเข้าร่วม (Join Request) ไปยังเซิร์ฟเวอร์เครือข่าย จากนั้นเซิร์ฟเวอร์เครือข่ายจะตรวจสอบตัวตนของอุปกรณ์ และหากได้รับการอนุญาต จะสร้างคีย์เซสชันเครือข่ายเฉพาะ (AppSKey) และที่อยู่เครือข่าย (DevAddr) สำหรับอุปกรณ์นั้น จากนั้นเซิร์ฟเวอร์เครือข่ายจะส่ง AppSKey และ DevAddr กลับไปยังอุปกรณ์ โดยเข้ารหัสด้วย AppKey ของอุปกรณ์ อุปกรณ์สามารถใช้ AppSKey และ DevAddr เพื่อสื่อสารกับเครือข่ายได้อย่างปลอดภัย และสามารถส่งและรับข้อมูลได้

2.2.5 การเพิ่มอุปกรณ์

LoRaWAN คีย์การเข้าร่วม (Join Key) เป็นคีย์เข้ารหัสเฉพาะที่ใช้ในการรักษาความปลอดภัยสำหรับกระบวนการเปิดใช้งานแบบ Over-the-Air Activation (OTAA) โดยคีย์นี้จะถูกแบ่งปันระหว่างอุปกรณ์และเซิร์ฟเวอร์เครือข่าย และใช้ในการเข้ารหัสและถอดรหัสข้อความคำขอเข้าร่วม (Join Request) และข้อความตอบรับเข้าร่วม (Join Accept) อุปกรณ์จะสร้างรหัสประจำตัวอุปกรณ์เฉพาะ (DevEUI) และรหัสเข้ารหัสเฉพาะอุปกรณ์ (AppKey) แล้วส่งรหัสเหล่านี้ในข้อความคำขอเข้าร่วมไปยังเซิร์ฟเวอร์เครือข่าย เซิร์ฟเวอร์เครือข่ายจะตรวจสอบตัวตนของอุปกรณ์ และหากอุปกรณ์ได้รับการอนุญาต จะสร้างคีย์เซสชันเครือข่ายเฉพาะ (AppSKey) และที่อยู่เครือข่าย (DevAddr) สำหรับอุปกรณ์ จากนั้นเซิร์ฟเวอร์เครือข่ายจะส่ง AppSKey และ DevAddr กลับไปยังอุปกรณ์ โดยเข้ารหัสด้วย AppKey ของอุปกรณ์ คีย์การเข้าร่วม (Join Key) ถูกใช้เพื่อให้แน่ใจว่าเฉพาะอุปกรณ์ที่ได้รับอนุญาตเท่านั้นที่สามารถเข้าร่วมเครือข่ายได้ และการสื่อสารระหว่างอุปกรณ์กับเครือข่ายได้รับการปกป้องด้วยการเข้ารหัส โดยคีย์การเข้าร่วมนี้มักจะถูกกำหนดไว้ในอุปกรณ์ก่อนการจัดส่ง แต่ก็สามารถกำหนดภายหลังได้ผ่านการกำหนดค่าแบบ Over-the-Air Provisioning (OTAP) แสดงดังรูปที่ 2.9 [4]



รูปที่ 2.9 การเข้ารหัสของ LoRaWAN

2.2.6 ข้อดีของ LoRaWAN

1. LoRaWAN ช่วยให้ทราบสถานะและควบคุมองค์ประกอบต่างๆ เช่น เซ็นเซอร์และแอคทูเอเตอร์ด้วยวิธีง่ายๆ

2. สามารถกระจายออกไปและครอบคลุมระยะทางหลายร้อยเมตรหรือหลายกิโลเมตรระหว่างโหนดและเกตเวย์

3. มีการใช้พลังงานต่ำ ซึ่งช่วยให้มีอิสระที่ดีขึ้นสำหรับอุปกรณ์ที่มีแบตเตอรี่หรือแบตเตอรี่

2.3 STM32H563Zi

STM32H562xx และ STM32H563xx เป็นไมโครคอนโทรลเลอร์ประสิทธิภาพสูงในตระกูลSTM32H5 แกนประมวลผล Arm® Cortex®-M33 32-bit RISC ซึ่งสามารถทำงานที่ความถี่สูงสุดถึง 250 MHz Cortex®-M33 core มาพร้อมกับหน่วยประมวลผลลอยตัวแบบความแม่นยำเดี่ยว (FPU) ที่รองรับคำสั่งการประมวลผลข้อมูลแบบความแม่นยำเดี่ยวของ Arm® ทั้งหมดและชนิดข้อมูลทั้งหมด โดยมีชุดคำสั่ง DSH (การประมวลผลสัญญาณดิจิทัล) และหน่วยป้องกันความจำ (MPU) ที่ช่วยเพิ่มความปลอดภัยให้กับแอปพลิเคชัน อุปกรณ์นี้มีหน่วยความเร็วสูงภายใน (หน่วยความจำแฟรชคู่สูงสุด 2 MB และ SRAM 640 KB) คอลโทรลเลอร์หน่วยความจำภายนอกที่ยืดหยุ่น (FMC) สำหรับอุปกรณ์ที่มีแพ็คเกจ 100 ขาหรือมากกว่านั้นหน่วยความจำ OCTOSPI (อย่างน้อยหนึ่ง Quad-SPI ใช้ได้ในทุกแพ็คเกจ) และการเชื่อมต่อ I/O และอุปกรณ์ต่อพ่วงที่เชื่อมต่อกับบัส APB สามบัส, AHB สามบัส, และบัสมัลติ-AHB 32-bit

อุปกรณ์เหล่านี้มีโครงสร้างความปลอดภัยที่สอดคล้องกับข้อกำหนดของสถาปัตยกรรมความปลอดภัยที่เชื่อถือได้ (TBSA) ของ Arm® มีคุณสมบัติในการอัปเดตเฟิร์มแวร์อย่างปลอดภัย นอกจากนี้อุปกรณ์ยังมีการติดตั้งเฟิร์มแวร์อย่างปลอดภัยที่ช่วยให้ผู้ใช้สามารถรักษาความปลอดภัยในการจัดหาซอฟต์แวร์ในระหว่างการผลิต การจัดการวงจรที่ยืดหยุ่นได้รับการรองรับด้วยระดับการป้องกันหลายระดับและการตรวจสอบการดีบักที่ปลอดภัย การแยกเฟิร์มแวร์ด้วยฮาร์ดแวร์ได้รับการรองรับผ่านอุปกรณ์ต่อพ่วงที่สามารถรักษาความปลอดภัยได้ หน่วยความจำ , I/O และการกำหนดค่าที่มีสิทธิพิเศษของอุปกรณ์ต่อพ่วงและหน่วยความจำ

ไมโครคอนโทรลเลอร์นี้มีหลายกลไกการป้องกันสำหรับหน่วยความจำแฟลชภายในและ SRAM: การป้องกันการอ่านออก, การป้องกันการเขียน, การรักษาความปลอดภัย และการซ่อนพื้นที่ป้องกัน ยังมีอุปกรณ์ต่อพ่วงเฉพาะที่เพิ่มความปลอดภัย: เครื่องเร่งความเร็วฮาร์ดแวร์ HASH และเครื่องกำเนิดหมายเลขสุ่มแบบจริง มีการตรวจจับการรบกวนที่ทำงานอยู่และการป้องกันการโจมตีจากการรบกวนชั่วคราวและการโจมตีจากสภาพแวดล้อม โดยใช้การตรวจสอบภายในหลายตัว การสร้างข้อมูลลับในกรณีที่ถูกโจมตี สิ่งนี้ช่วยให้เป็นไปตามข้อกำหนดของ PCI สำหรับแอปพลิเคชันจุดขาย มีอุปกรณ์เหล่านี้มี ADC 12-bit ที่รวดเร็วสองตัว ช่องสัญญาณ DAC สองช่อง บัฟเฟอร์แรงดันไฟฟ้าอ้างอิงภายใน RTC พลังงานต่ำ ตัวนับเวลาทั่วไป 32-bit สองตัว ตัวนับเวลา PWM 16-bit สองตัวที่ออกแบบมาสำหรับการควบคุมมอเตอร์ ตัวนับเวลาทั่วไป 16-bit แปดตัว ตัวนับเวลาพื้นฐาน 16-bit สองตัว และตัวนับเวลาพลังงานต่ำ 16-bit หกตัว มีอุปกรณ์เหล่านี้ยังมีอินเทอร์เฟซการสื่อสารมาตรฐานและขั้นสูง ได้แก่ I2C สี่ตัว I3C หนึ่งตัว SPI หกตัว I2S สามตัว USART หกตัว UART หกตัว และ UART พลังงานต่ำหนึ่งตัว SAI สองตัว อินเทอร์เฟซกล้องดิจิทัล (DCMI) สูงสุดสองตัว SDMMC

สูงสุดสองตัว FDCAN สูงสุดสองตัว USB full-speed หนึ่งตัว คอนโทรลเลอร์ USB Type-C®/USB หนึ่งตัว และอินเทอร์เฟซ Ethernet (มีเฉพาะใน STM32H563xx)

รองรับการจ่ายพลังงานอิสระ: อินพุตจ่ายไฟอิสระสำหรับ ADC, DAC อินพุตจ่ายไฟ 3.3 V ที่ทุ่มเทสำหรับ USB และอินพุตจ่ายไฟที่ทุ่มเทสำหรับบาง GPIO และ SDMMC มีอินพุต VBAT เพื่อเชื่อมต่อแบตเตอรี่สำรองเพื่อรักษาฟังก์ชันการทำงานของ RTC และสำรองรีจิสเตอร์ 32-bit 32 ตัว และ SRAM ขนาด 4 Kbyte แสดงดังรูปที่ 2.10 [5]

รูปที่ 2.10 STM32H563Zi

2.4 Local Network (LoRa Board) LoRa E5 Module

ในการออกแบบใช้ Module ในการสื่อสารเป็น LoRa-E5 module ถูกออกแบบมาตามมาตรฐานอุตสาหกรรม รวมถึงรองรับอุณหภูมิการทำงานที่กว้าง ตั้งแต่ -40°C ถึง +85°C ทำให้เหมาะสมอย่างยิ่งสำหรับผลิตภัณฑ์ IoT ในอุตสาหกรรม STM32H563Zi ของ ST เป็น SoC ที่มีการผสมผสานระหว่างชิป LoRa® RF และ MCU ในชิ้นเดียว นอกจากนี้ โมดูลนี้ยังฝังไว้ด้วย Arm® Cortex®-M4 MCU ที่ใช้พลังงานต่ำมากและ LoRa SX126X เพื่อรองรับโหมด (G)FSK และ LoRa โดยในโหมด LoRa® สามารถใช้แบนด์วิธ 62.5 kHz, 125 kHz, 250 kHz, และ 500 kHz ทำให้เหมาะสำหรับการออกแบบ IoT nodes ที่รองรับ EU868 และ US915 แสดงดังรูปที่ 2.11

LoRa-E5 Mini เป็นบอร์ดพัฒนาแบบขนาดเล็กที่เหมาะสำหรับแอปพลิเคชันที่ต้องการการทดสอบอย่างรวดเร็ว โดยมีช่องสัญญาณสื่อสารใน LoRa-E5 รวมถึง UART, ADC, SPI และ I2C และมีพลังงาน RF output สูงถึง +20.8 dBm ที่ 3.3 V [5]

2.4.1 คุณสมบัติของ LoRa-E5 module

1. การใช้พลังงานต่ำเป็นพิเศษ: กระแสไฟเพียง 2.1 µA (โหมด VOR)
2. ขนาดกะทัดรัด: 12 มม. x 12 มม. x 2.5 มม. 28 พิน SMT
3. ประสิทธิภาพสูง: TXOP=22 dBm ที่ 868/915 MHz; ความไว -136.5 dBm สำหรับ SF12 พร้อม 125 kHz BW

4. การใช้งานทางไกล: 158 dB
5. การเชื่อมต่อไร้สาย: โปรโตคอล LoRaWAN แบบฝัง, คำสั่ง AT, รองรับแผนความถี่
6. LoRaWAN ทั่วโลก
7. ความเข้ากันได้ทั่วโลก: ช่วงความถี่กว้าง EU868, US915, AU915, AS923, KR920, IN865
8. ความยืดหยุ่นที่ยอดเยี่ยมสำหรับผู้ใช้ที่ต้องการพัฒนาซอฟต์แวร์บน MCU ของโมดูล GPIO อื่น ๆ ของ MCU สามารถจัดการได้อย่างง่ายดาย รวมถึง UART, I^2C, ADC และอื่น ๆ อินเทอร์เฟซ GPIO ที่หลากหลายเหล่านี้มีประโยชน์สำหรับผู้ใช้ที่ต้องการขยายอุปกรณ์ต่อพ่วง
9. ได้รับการรับรองจาก FCC และ CE

รูปที่ 2.11 LoRa-E5-HF

2.5 Local Network (BLE Board) HC-05 Bluetooth Module

HC-05 เป็นโมดูลบลูทูธที่ออกแบบมาเพื่อสร้างการสื่อสารข้อมูลไร้สายระยะสั้นระหว่างไมโครคอนโทรลเลอร์หรือระบบสองตัว โมดูลทำงานบนโปรโตคอลการสื่อสาร Bluetooth 2.0 และสามารถทำหน้าที่เป็นอุปกรณ์รองเท่านั้น นี่เป็นวิธีที่ถูกที่สุดสำหรับการส่งข้อมูลแบบไร้สายและมีความยืดหยุ่นมากกว่าเมื่อเทียบกับวิธีอื่น ๆ และยังสามารถส่งไฟล์ด้วยความเร็วสูงสุด 2.1Mb/s แสดงดังรูปที่ 2.12

HC-05 ใช้เทคนิคการแพร่กระจายสเปกตรัมแบบกระโดดความถี่ (FHSS) เพื่อหลีกเลี่ยงการรบกวนกับอุปกรณ์อื่น ๆ และเพื่อให้มีการส่งสัญญาณแบบฟูลดูเพล็กซ์ อุปกรณ์ทำงานบนช่วงความถี่ตั้งแต่ 2.402 GHz ถึง 2.480GHz [6]

2.5.1 HC-05 คุณสมบัติและลักษณะทางไฟฟ้า

1. โปรโตคอลบลูทูธ : มาตรฐานโปรโตคอล Bluetooth V2.0
2. ระดับพลังงาน : Class2(+6dBm)
3. ความถี่ : 2.40 GHz—2.48 GHz, ย่านความถี่ ISM

4. ความไวของตัวรับ : - 85 dBm
5. โปรโตคอล USB : USB v1.1/2.0
6. โหมดมอดูเลต : การคีย์เปลี่ยนความถี่เกาส์
7. คุณลักษณะด้านความปลอดภัย : การรับรองความถูกต้องและการเข้ารหัส
8. ช่วงแรงดันไฟฟ้าที่ใช้งาน : + 3.3V ถึง + 6V
9. ช่วงอุณหภูมิในการทำงาน : -20°C ถึง +55°C
10. กระแสไฟที่ใช้งาน: 40mA

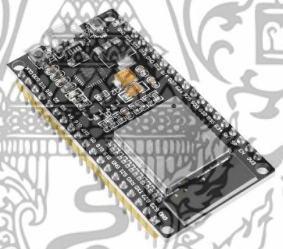
2.5.2 ข้อดีของโมดูลบลูทูธ

1. HC-05 เป็นตัวเลือกที่ดีที่สุดเมื่อต้องการการสื่อสารไร้สายทางไกล โมดูลนี้ใช้สำหรับการสื่อสารไร้สายน้อยกว่า 100 เมตร
2. โมดูลนี้ง่ายต่อการเชื่อมต่อและสื่อสาร
3. โมดูลนี้เป็นหนึ่งในโซลูชั่นที่ถูกที่สุดสำหรับการสื่อสารไร้สายทุกประเภทที่มีอยู่ในตลาด
4. โมดูลใช้พลังงานในการทำงานน้อยกว่ามาก และสามารถใช้กับระบบเคลื่อนที่ที่ใช้แบตเตอรี่
5. โมดูลสามารถเชื่อมต่อกับคอนโทรลเลอร์หรือโปรเซสเซอร์เกือบทั้งหมดเนื่องจากใช้อินเทอร์เฟซ UART

รูปที่ 2.12 HC-05 Bluetooth Module

2.6 Esp 32 Wroom – 32

ESP32-WROOM-32 เป็นโมดูล Wi-Fi + Bluetooth® + Bluetooth LE MCU ทั่วไปที่ทรงพลังซึ่งกำหนดเป้าหมายการใช้งานที่หลากหลาย ตั้งแต่เครือข่ายเซ็นเซอร์ที่ใช้พลังงานต่ำไปจนถึงงานที่มีความต้องการมากที่สุด เช่น การเข้ารหัสเสียง การสตรีมเพลง และการถอดรหัส MP3 แกนหลักของโมดูลนี้คือชิป ESP32-D0WDQ6* ชิปที่ฝังอยู่ได้รับการออกแบบมาให้ปรับขนาดและปรับเปลี่ยนได้ มีคอร์ CPU สองคอร์ที่สามารถควบคุมแยกกันได้ และความถี่สัญญาณนาฬิกาของ CPU สามารถปรับได้ตั้งแต่ 80 MHz ถึง 240 MHz ชิปยังมีโปรเซสเซอร์ร่วมที่ใช้พลังงานต่ำซึ่งสามารถใช้แทน CPU เพื่อประหยัดพลังงานในขณะที่ทำงานที่ไม่ต้องใช้พลังการประมวลผลมากนัก เช่น การตรวจสอบอุปกรณ์ต่อพ่วง ESP32 รวมชุดอุปกรณ์ต่อพ่วงที่หลากหลาย ตั้งแต่เซ็นเซอร์สัมผัสแบบ capacitive, อินเทอร์เฟซการ์ด SD, อีเธอร์เน็ต, SPI ความเร็วสูง, UART, I2S และ I2C โมดูลรองรับอัตราข้อมูลสูงถึง 150 Mbps และกำลังขับ 20 dBm ที่เสาอากาศเพื่อให้แน่ใจว่ามีช่วงทางกายภาพที่กว้างที่สุด ด้วยเหตุนี้โมดูลจึงนำเสนอข้อกำหนดชั้นนำของอุตสาหกรรมและประสิทธิภาพที่ดีที่สุดสำหรับการรวมระบบอิเล็กทรอนิกส์ ดังรูปที่ 2.13 [24]



รูปที่ 2.13 ESP32-WROOM-32

ตารางที่ 2.3 แสดงข้อมูลจำเพาะของ ESP32-WROOM-32

หมวดหมู่	รายการ	รายละเอียด
การรับรอง	การรับรอง Wi-Fi	Wi-Fi Alliance
	การรับรอง Bluetooth	BQB
	การรับรองด้านสิ่งแวดล้อม	RoHS/REACH
การทดสอบ	ความน่าเชื่อถือ	HTOL/HTSL/uHAST/TCT/ESD

ตารางที่ 2.3 (ต่อ) แสดงข้อมูลจำเพาะของ ESP32-WROOM-32

หมวดหมู่	รายการ	รายละเอียด
Wi-Fi	โปรโตคอล	802.11 b/g/n (802.11n ความเร็วสูงสุด 150 Mbps)
	ช่วงความถี่กลางของช่องสัญญาณ	2412 ~ 2484 MHz
Bluetooth	โปรโตคอล	Bluetooth v4.2 BR/EDR และ Bluetooth LE
	วิทยุ	ตัวรับ NZIF ที่ความไว -97 dBm
		เครื่องส่ง Class-1, Class-2 และ Class-3 พร้อม AFH
ฮาร์ดแวร์	อินเตอร์เฟซของโมดูล	SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, Pulse counter, GPIO, Capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWA®), รองรับมาตรฐาน ISO11898-1 (CAN Specification 2.0)
	คริสตัลในตัว	คริสตัล 40 MHz
	หน่วยความจำแฟลช SPI ในตัว	4 MB
	แรงดันไฟฟ้า/การจ่ายไฟ	3.0 V ~ 3.6 V
	กระแสไฟเฉลี่ย	80 mA
	กระแสไฟฟ้าขั้นต่ำที่จ่ายจากแหล่งจ่ายไฟ	500 mA
	อุณหภูมิที่แนะนำสำหรับการทำงาน	-40 °C ~ +85 °C

2.7 การพัฒนา Firmware บนชิพประมวลผล STM32H563Zi

2.7.1 คุณสมบัติ STM32H562xx และ STM32H563xx

STM32H562xx และ STM32H563xx เป็นไมโครคอนโทรลเลอร์ (MCU) ในตระกูล STM32H5 จาก STMicroelectronics ซึ่งเป็นชิปประสิทธิภาพสูงที่ออกแบบมาเพื่อตอบสนองความต้องการของแอปพลิเคชันที่หลากหลาย โดยเฉพาะอย่างยิ่งงานที่ต้องการประสิทธิภาพในการประมวลผลที่สูง เช่น อุตสาหกรรม, การควบคุมมอเตอร์, การเชื่อมต่อเครือข่าย และอินเทอร์เฟซผู้ใช้ ซึ่งสามารถระบุ และ แบ่งประเภทคุณสมบัติได้ซึ่งสรุปได้ แสดงดังตาราง 2.4

ตารางที่ 2.4 ประเภทคุณสมบัติของไมโครคอนโทรลเลอร์ (MCU) STM32H562xx และ STM32H563xx

หัวข้อ	คุณสมบัติ
หน่วยประมวลผลหลัก	Arm® Cortex®-M33 CPU with TrustZone®, FPU, frequency up to 250 MHz, MPU, 375 DMIPS (Dhrystone 2.1)
หน่วยความจำ	1.Flash memory มีขนาด 2 Mbyte 2.RAM มีขนาด 640 Kbyte ถูกแบ่งเป็น 3 Partition คือ SRAM1 มีขนาด 256 Kbytes SRAM2 มีขนาด 64 Kbytes มาพร้อมกันกับ ECC SRAM3 มีขนาด 320 Kbytes 3.รองรับ SD/SDIO/MMC
ระบบสัญญาณนาฬิกา	1. สัญญาณนาฬิกาภายในชิพ (Internal oscillators) - 64 MHz สำหรับ HIS - 48 MHz สำหรับ HSI48 (สัญญาณนาฬิกาแบบคงที่) - 4 MHz สำหรับ CSI - 32 kHz สำหรับ LSI 2. สัญญาณนาฬิกาภายนอก (External oscillators) - 4-50 MHz สำหรับ HSE (สามารถเลือกใช้ Crystal oscillator ได้ตามขอบเขตที่รองรับ) - 32.768 kHz สำหรับ LSE

ตารางที่ 2.4 (ต่อ) ประเภทคุณสมบัติของไมโครคอนโทรลเลอร์ (MCU) STM32H562xx และSTM32H563xx

หัวข้อ	คุณสมบัติ
GPIO (General-purpose input/output)	1.รองรับขา GPIO 140 ขา (Pin)
ฟังก์ชันทางเลือก (Althernate Function)	1.รองรับตั้งแต่ AF0-AF7
อุปกรณ์ต่อพ่วง (Peripheral)	1. UART - UART รองรับ 6 ตัวแบ่งได้ดังนี้ UART4, UART5, UART7, UART8, UART9, UART12 - USART รองรับ 6 ตัวแบ่งได้ดังนี้ USART1, USART2, USART3, USART6, USART10, USART11 - LPUART1 รองรับ 1 ตัว 2. I2C -I2C รองรับ 4 ตัวแบ่งได้ดังนี้ I2C1, I2C2, I2C3, I2C4 3.I3C มีประสิทธิภาพสูงกว่า I2C 4.SPI - SPI รองรับ 6 ตัวแบ่งได้ดังนี้ SPI1, SPI2, SPI3 รองรับ Full feature set instances SPI4, SPI5, SPI6 รองรับ Limited feature set instance 5.FDCAN 6.SDMMC 7.USB Full Speed 8.ETH
พอร์ทสำหรับผู้พัฒนา (Debug port)	1.SWJ - DP (Serial-wire/JTAG Debug port) 2.SWD (Serial wire Debug)

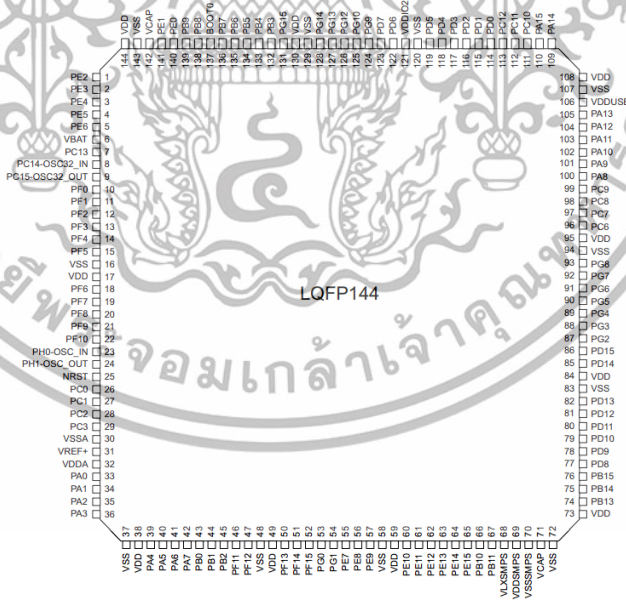
2.7.2 ฟังก์ชันทางเลือก (Alternate Function)

เป็นทางเลือกสำหรับผู้ใช้งาน ที่ต้องการเลือกใช้อินเตอร์เฟซเพิ่มเติมที่นอกเหนือจากการใช้งานปกติ เช่นที่พิน PA1 รองรับการสื่อสารแบบ UART (TX) แต่ต้องการเปลี่ยนเป็นการสื่อสารรูปแบบอื่นที่เป็น SPI ก็สามารถทำได้ โดยผู้ใช้สามารถศึกษาการใช้งานเพิ่มเติมได้จากคู่มือการใช้งานที่เขียนโดย STM32 ซึ่งมีตัวอย่างการศึกษาแสดงดังรูปที่ 2.14 โดย Pin การใช้งานจะมีลักษณะ แสดงดังรูปที่ 2.15

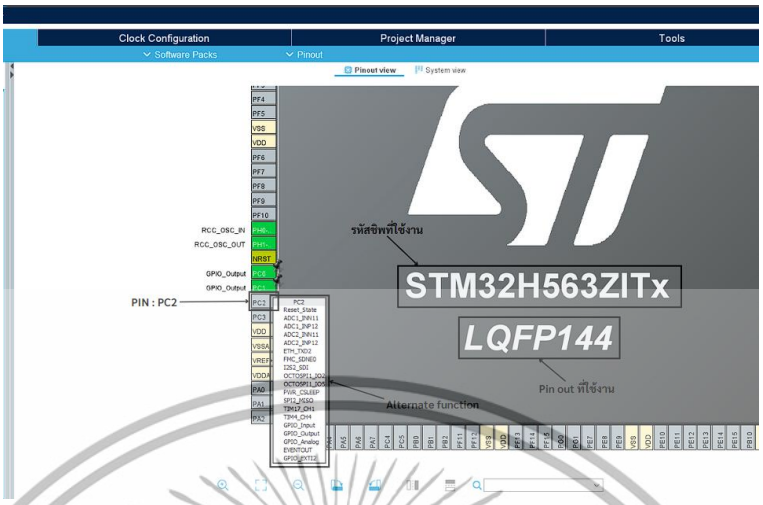
การเปลี่ยน Alternate function ของชิพ STM32 สามารถทำได้ผ่านโปรแกรม STM32CubeIDE โดยในโปรแกรมจะใช้ลักษณะการเปลี่ยนแบบหน้าต่างอินเตอร์เฟซ (GUI) ซึ่งในโปรแกรม เมื่อกดที่ Pin จะปรากฏอุปกรณ์ต่อพ่วง ที่ Pin นั้นๆสามารถเปลี่ยนได้ แสดงดังรูปที่ 2.16 [7]

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
	SYS	LPTIM1/ TIM1/2/16/17	LPTIM3/ PDM_SAI1/ TIM3/4/5/12/15	I2C1/LPTIM2/3/ LPUART1/ OCTOSPI/TIM1/8	CEC/DCMI/ I2C1/2/3/4/ LPTIM1/2/5/PI1/ I2S1/TIM15/ USART1	CEC/I3C1/LPTIM1/ SPI1/I2S1/SPI2/I2S2/ SPI3/I2S3/SPI5/6	I2C4/OCTOSPI/ SAI1/SPI3/I2S3/ SPI4/UART4/12/ USART10/ USB_PD	SDMMC1/SPI2/ I2S2/SPI3/I2S3/ SPI6/UART7/8/12 /USART1/2/3/6/ 10/11
PA0	-	TIM2_CH1	TIM5_CH1	TIM8_ETR	TIM15_BKIN	SPI6_NSS	SPI6_RDY	USART2_CTS/ USART2_NSS
PA1	-	TIM2_CH2	TIM5_CH2		TIM15_CH1N	LPTIM1_IN1	OCTOSPI1_DQS	USART2_RTS/ USART2_DE
PA2	-	TIM2_CH3	TIM5_CH3	-	TIM15_CH1	LPTIM1_IN2		USART2_TX
PA3	-	TIM2_CH4	TIM5_CH4	OCTOSPI1_CLK	TIM15_CH2	SPI2_NSS/I2S2_WS	SAI1_SD_B	USART2_RX

รูปที่ 2.14 ตัวอย่างตารางฟังก์ชันเสริม (Alternate Function)



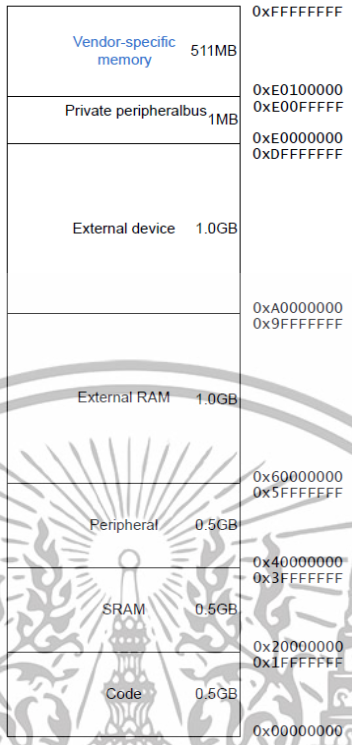
รูปที่ 2.15 ภาพรวมของชิพ STM32 ในรหัส LQFP144



รูปที่ 2.16 Pinout & Configuration ของชิพ STM32H563ZITx

2.7.3 Processer Memory map

ภายในระบบคอมพิวเตอร์ มีข้อมูลมากมาย ที่ต้องใช้พื้นที่ความจำในการเก็บข้อมูลดังกล่าว ภายในชิพ ซึ่งจะเป็นโครงสร้าง ที่จะระบุพื้นที่จัดเก็บด้วยที่อยู่ (Address) โดยแบ่งเป็นหมวดหมู่ และ Memory Map เป็นการจัดการหน่วยความจำในโปรเซสเซอร์ให้มีระเบียบ เพื่อให้โปรเซสเซอร์รู้ว่าจะต้องไปเก็บหรือดึงข้อมูลจากตำแหน่งไหนในหน่วยความจำ ซึ่งในแต่ละผู้ผลิต จะมีการลำดับจัดเก็บข้อมูลที่ไม่เหมือนกัน ตัวอย่างเช่น แสดงรูปที่ 2.17 ซึ่งเป็นชิพจากทาง STMicroelectronics โดยทั่วไปแล้วการพัฒนาเฟิร์มแวร์ของผู้พัฒนามีความจำเป็นที่จะต้องทราบการจัดการของหน่วยความจำ เพื่อใช้เก็บข้อมูลในส่วนของโค้ดโปรแกรม ข้อมูล ตัวแปร หรือจะเป็นพื้นที่ของอุปกรณ์ต่อพ่วง (Peripheral) เช่นพอร์ต GPIO, Real-Time Clock, หน่วยความจำภายนอก ซึ่งสามารถแบ่งประเภทการจัดเก็บได้ตาม แสดงดังตารางที่ 2.5



รูปที่ 2.17 Memory map ของ STM32H563Zi

ตารางที่ 2.5 ประเภทการจัดเก็บข้อมูลภายในชิพ STM32H563Zi

ชื่อหัวข้อการจัดเก็บ	ที่อยู่	ประเภทหน่วยความจำ	ขนาด (GB)	คุณสมบัติการจัดเก็บ
Code	0x00000000 – 0x1FFFFFFF	Normal	0.5 GB	พื้นที่สำหรับเก็บโปรแกรมคำสั่ง
SRAM	0x20000000 - 0x3FFFFFFF	Normal	0.5 GB	หน่วยความจำสำหรับเก็บข้อมูลชั่วคราว เหมาะสำหรับการเก็บข้อมูลที่ต้องการความเร็วในการแลกเปลี่ยน

ตารางที่ 2.5 (ต่อ) ประเภทการจัดเก็บข้อมูลภายในชิพ STM32H563Zi

ชื่อหัวข้อการจัดเก็บ	ที่อยู่	ประเภทหน่วยความจำ	ขนาด (GB)	คุณสมบัติการจัดเก็บ
Peripheral	0x40000000 - 0x5FFFFFFF	Device, nGnRE (ไม่อนุญาตให้หน่วยความจำอ่าน-เขียนซ้ำไปซ้ำมา)	0.5 GB	หน่วยความจำที่ใช้ควบคุมและเก็บข้อมูลอุปกรณ์ต่อพ่วง เช่น UART, I2C , SPI แบบ SoC (Systems on chitp)
External RAM	0x60000000 - 0x9FFFFFFF	Normal	1 GB	หน่วยความจำสำหรับเก็บข้อมูลชั่วคราว เหมาะสำหรับการเก็บข้อมูลที่ต้องการขนาดเก็บข้อมูล
External Device	0xA0000000 - 0xDFFFFFFF	Device, nGnRnE	1 GB	เป็นหน่วยความจำที่ใช้งานสำหรับอุปกรณ์ต่อพ่วงภายนอก (External device memory)
Private Peripheral Bus	0xE0000000 - 0xE003FFFF	Device, nGnRnE	1 MB	เป็นหน่วยความจำสำหรับการควบคุมอุปกรณ์ในตัวชิพ เช่น NVIC, SCS, MPU, SAU, และ ITM
Vendor_SYS	0xE0100000 - 0xFFFFFFFF	Device, nGnRnE	511 MB	พื้นที่หน่วยความจำที่ขึ้นอยู่กับผู้ผลิตชิพ (Vendor specific)

2.7.4 โหมดการบูต (Boot Mode)

การบูตโปรแกรมเมื่อเริ่มการทำงาน หรือเริ่มการอัปโหลดโปรแกรม ตัวไมโครคอนโทรลเลอร์จะเข้าสู่โหมดการบูต โดยตัวชิปเมื่อมีแรงดันไฟฟ้าเข้ามาที่ขา BOOT0 จะทำให้ชิปเข้าสู่ที่อยู่การบูตที่กำหนดไว้ โดยจะทำหน้าที่เป็นตัวกลางในการอัปเดตเฟิร์มแวร์ของชิปผ่านพอร์ตการสื่อสารต่างๆ เช่น UART หรือ USB หรือ Debug interfaceโหมดการบูตของ STM32H563ZI จะแบ่งออกเป็น 2 ประเภท ได้แก่

1. TrustZone Disabled

เมื่อโหมด TrustZone ถูกปิดการทำงาน ไมโครคอนโทรลเลอร์จะบูตตามปกติโดยไม่แบ่งแยกพื้นที่ความปลอดภัย (Secure และ Non-secure) โดยจะใช้ที่อยู่คือ 0x0800 0000

2. TrustZone Enabled

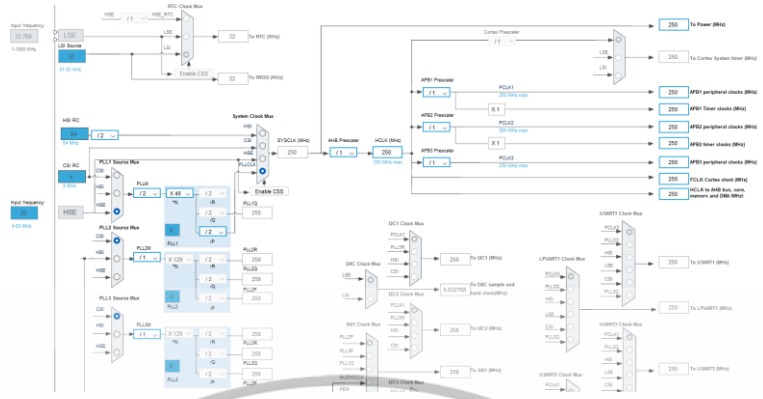
เมื่อโหมด TrustZone ถูกเปิดการทำงาน ไมโครคอนโทรลเลอร์จะบูตโดยแบ่งพื้นที่ความปลอดภัยออกเป็นสองโซน (Secure และ Non-secure) ซึ่งจะทำให้สามารถควบคุมสิทธิ์การเข้าถึงทรัพยากรต่าง ๆ ได้ โดยจะใช้ที่อยู่คือ 0x0C00 0000

ในกรณีที่ต้องการบูตเข้าสู่โหมดบูตโหลดเดอร์ (Bootloader Mode) เพื่อนำไปสู่การอัปเดตเฟิร์มแวร์ใหม่ จะต้องตั้งค่าพิน BOOT0 ให้เป็น HIGH เพื่อเข้าสู่ System Memory ซึ่งจะทำให้สามารถอัปเดตเฟิร์มแวร์ได้ผ่านพอร์ตการสื่อสารที่รองรับ [8]

2.7.5 ตัวควบคุมสัญญาณนาฬิกาและการรีเซ็ต (Reset and clock controller)

RCC หรือ Reset and Clock Controller มีหน้าที่ควบคุมการทำงานของนาฬิกาและรีเซ็ตระบบในไมโครคอนโทรลเลอร์ โดยกระจายสัญญาณนาฬิกา ปรับความเร็ว และเลือกแหล่งกำเนิดสัญญาณที่เหมาะสม สามารถตั้งค่าได้ง่ายผ่านโปรแกรม STM32CubeIDE โดยลักษณะ แสดงดังรูปที่ 2.18 และสามารถแบ่งองค์ประกอบของสัญญาณนาฬิกา และ องค์ประกอบการตั้งค่าสัญญาณนาฬิกาได้ แสดงดังตารางที่ 2.6 และ 2.7

การตั้งค่าสัญญาณนาฬิกาภายในไมโครคอนโทรลเลอร์ STM32 สามารถเลือกแหล่งสัญญาณจาก CSI, HIS(ภายใน) หรือ HSE(ภายนอก) ได้ โดย HSE สามารถใช้ Crystal Oscillator ระหว่าง 4-50 MHz โดยทั่วไปแนะนำที่ 25 MHz ใน STM32CubeIDE มีปุ่ม Resolve Clock Issues ซึ่งจะช่วยคำนวณความถี่ที่เหมาะสม ทำให้นักพัฒนาไม่ต้องคำนวณเอง [9]



รูปที่ 2.18 Clock Configuration บน STM32CubeIDE

ตารางที่ 2.6 ชนิดสัญญาณนาฬิกาที่ใช้ STM32H563Zi

ประเภทสัญญาณนาฬิกา	ชนิด	ความถี่	ประเภทการใช้งาน
Input Frequency Source	Internal SoC	HSI (High-Speed Internal)	64 MHz	แหล่งความถี่ภายใน (Internal Oscillator) เป็นค่าเริ่มต้นเมื่อเปิดใช้งานไมโครคอนโทรลเลอร์
		HSI48	48 MHz	สัญญาณนาฬิกาที่ความที่คงที่
		CSI (Clock Security System Internal Oscillator)	4 MHz	สัญญาณนาฬิกาที่มีความเที่ยงตรงสูงและมีการตรวจสอบความถูกต้องของสัญญาณนาฬิกา
		LSI (Low-Speed Internal Oscillator)	32 kHz	สัญญาณนาฬิกาภายในที่มีความถี่ต่ำ (Low-Speed)

ตารางที่ 2.7 องค์ประกอบการตั้งค่าสัญญาณนาฬิกา

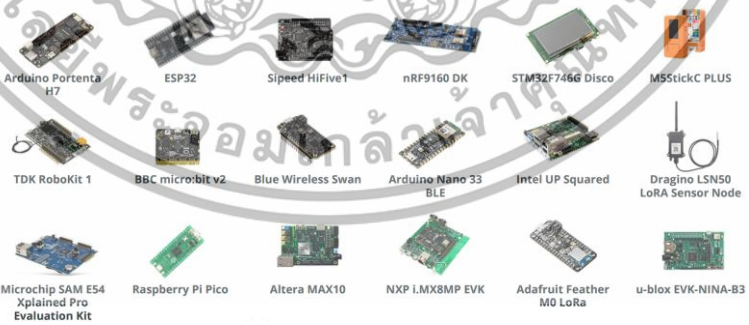
องค์ประกอบสัญญาณนาฬิกา	**ประเภท**	**การทำงาน**
PLL Configuration	PLL Source	ตัวเลือกของสัญญาณต้นทาง เช่น HSE หรือ HSI สามารถเลือกได้
	PLLM (ตัวหารสัญญาณนาฬิกา)	ปรับลดความถี่ก่อนเข้าสู่วงจร PLL
	PLLN (ตัวคูณสัญญาณนาฬิกา)	ขยายความถี่ที่ต้องการ
	PLLP, PLLQ, PLLR	ตัวหารหลังการขยายความถี่แล้ว เพื่อนำสัญญาณไปใช้ในส่วนต่าง ๆ ของไมโครคอนโทรลเลอร์
System Clock (SYSCLK)	SYSCLK	สัญญาณนาฬิกาหลักที่ใช้กำหนดความเร็วการทำงานของโปรเซสเซอร์ (CPU)
AHB และ APB Prescaler	AHB (Advanced High-performance Bus)	โปรโตคอลบัสประสิทธิภาพสูงที่ออกแบบโดย ARM สำหรับการเชื่อมต่ออุปกรณ์ต่อพ่วงความเร็วสูงและหน่วยความจำในการออกแบบระบบบนชิป (SoC) [10]
	APB (Advanced Peripheral Bus)	โปรโตคอลบัสต้นทุนต่ำที่ออกแบบโดย ARM สำหรับการเชื่อมต่ออุปกรณ์ต่อพ่วงความเร็วต่ำในการออกแบบระบบบนชิป (SoC) [10]
Peripheral Clocks		สัญญาณความถี่จาก SYSCLK จะถูกกระจายไปยัง Peripherals ต่าง ๆ เช่น USART, GPIO, I2C, ADC

2.8 ระบบปฏิบัติการ Zephyr

2.8.1 Zephyr

Zephyr แสดงดังรูปที่ 2.19 คือ Real-Time Operating System (RTOS) ที่ออกแบบมาโดยเฉพาะสำหรับอุปกรณ์ฝังตัวขนาดเล็กมีความสามารถในการจัดการทรัพยากรที่ใช้ภายในชิพเช่น CPU ,หน่วยความจำ , อุปกรณ์ต่อพวง , การจัดการอินเทอร์เน็ต, และความสามารถในการจัดการ RTOS (Real-Time Operation systems) ถูกพัฒนาโดย Linux Foundation ที่เป็น Open source และใช้งานได้ฟรี และมีข้อดีคือมีขนาดโค้ดที่เล็กสามารถใช้งานกับไมโครคอนโทรลเลอร์ที่มีพื้นที่จำกัดได้ดี รองรับเซ็นเซอร์ที่หลากหลาย, บอร์ดสำเร็จรูปอื่นๆ แสดงดังรูปที่ 2.20 และชิพประมวลผลที่รองรับ แสดงดังรูปที่ 2.21 [11]

รูปที่ 2.19 ระบบปฏิบัติการ Zephyr



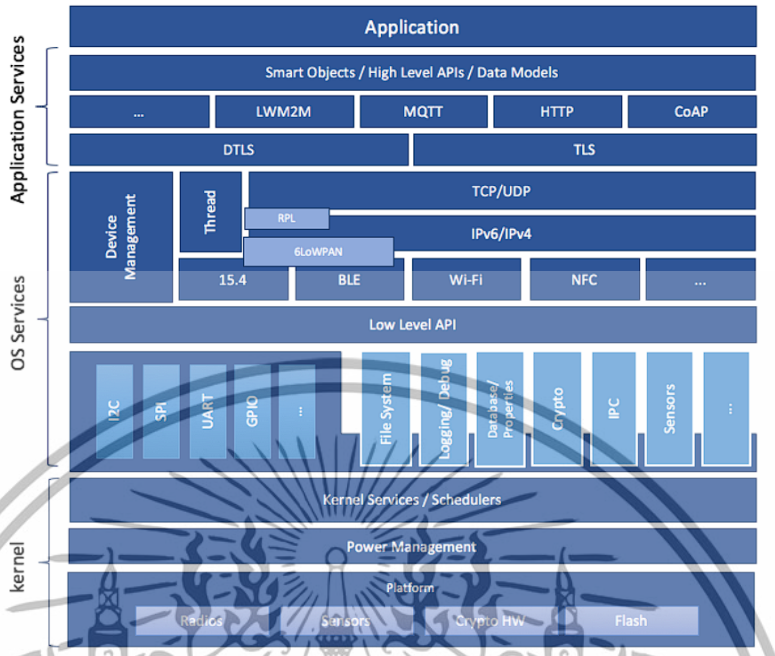
รูปที่ 2.20 บอร์ดไมโครคอนโทรเลอร์ที่รองรับการทำงานบน Zephyr

รูปที่ 2.21 ชิพประมวลผลที่รองรับกับระบบปฏิบัติการ Zephyr

2.8.2 โครงสร้างของ Zephyr OS

ภายในโครงสร้างมีส่วนประกอบหลักๆ แสดงดังรูปที่ 2.22 ซึ่งมีอยู่ 5 ส่วนคือ

1. 3rd Party Libraries เป็นส่วนเสริมของไลบารีอื่นๆ ที่ผู้ใช้งานสามารถเลือกใช้งานได้ เช่น MQTT สำหรับการสื่อสารแบบ publish-subscribe
2. Application Services รวมบริการการจัดการอุปกรณ์เซ็นเซอร์ หรือการเชื่อมต่อคลาวด์
3. OS Services ทำหน้าที่จัดการอุปกรณ์ต่อพ่วง (Peripheral) ซึ่งเป็นการจัดการถัดลงมาจาก Low-level API
4. Kernal ใช้จัดการ Memory Management, Data processing
5. HAL (Hardware Abstraction Layer) เป็นส่วนติดต่อกับผู้ใช้งาน โดยผู้ใช้งานสามารถเรียกใช้งานได้ผ่านฟังก์ชัน และ มาโคร ที่ Zephyr ได้กำหนดไว้และสามารถศึกษาเพิ่มเติมได้จาก Zephyr documentation [12]



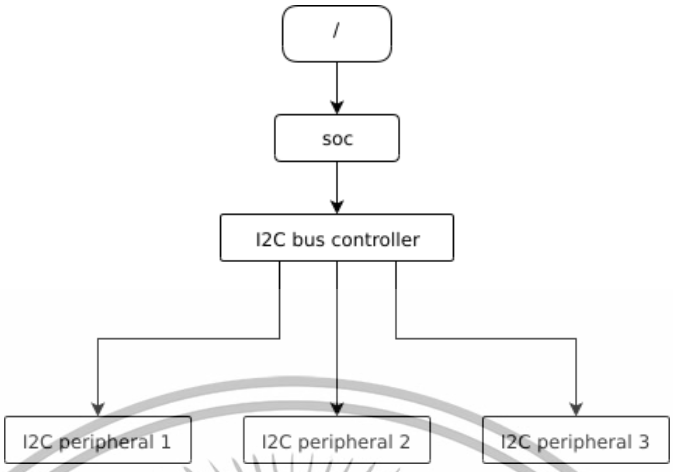
รูปที่ 2.22 โครงสร้างสถาปัตยกรรมของ Zephyr

2.8.3 Devicetree

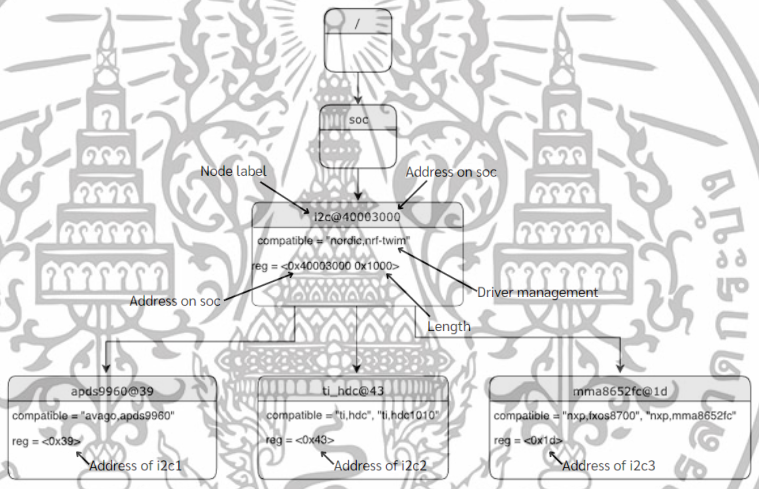
ภาษาที่ใช้อธิบายฮาร์ดแวร์ของไมโครโพรเซสเซอร์และใช้ใน Zephyr OS คือ Devicetree เป็นโครงสร้างแบบต้นไม้ที่มีโหนด (Node) เป็นอุปกรณ์ชนิดต่างๆที่ใช้งานในงาน หรือระบบบนชิพ (Systems on chip, SOC) ที่ใช้อธิบายและระบุที่อยู่ของ Node ต่างๆที่เป็น GPIO หรือ Interface อื่นๆที่ใช้งานอยู่แล้วบนชิพ รวมทั้งการระบุตำแหน่งที่อยู่การอัปโปรแกรม ซึ่งบน Zephyr มีลักษณะ แสดงดังรูปที่ 2.23

จากรูปที่ 2.23 สามารถอธิบายได้ว่า โหนดรูท (/) เป็นโหนดใหญ่สุดที่อยู่ในอุปกรณ์ภายในโหนดรูทที่ชื่อว่า soc มี I2C bus controller และแบ่งได้ 3 ตัวคือ I2C1,I2C2,I2C3 ซึ่งภายใน I2C จะมี Properties ที่ใช้อธิบายที่อยู่บนชิพของทั้ง 3 ตัว ไดร์เวอร์ที่ใช้จัดการ และ ชื่อโหนด (Node Lable) เพื่อเรียกใช้ในโปรแกรม Zephyr ผ่าน Macro ที่มีให้อยู่แล้ว หรือในอีกตัวอย่าง แสดงดังรูปที่ 2.24

ภายในโหนดของ I2C มีอุปกรณ์ต่อพ่วง 3 ตัวในที่นี้คือเซ็นเซอร์ที่ใช้ i2c เป็น interface การสื่อสารซึ่งใช้ I2C1,I2C2,I2C3 ตามลำดับและภายใต้โหนดที่กล่าวมามี Properties ที่บอกที่อยู่ของแต่ละ Peripheral และไดร์เวอร์ที่ใช้จัดการเซ็นเซอร์ต่างๆ โดยการทำงานของ Zephyr จะทำการหาไดร์เวอร์ตามที่ผู้ใช้ได้ตั้งค่าไว้ เพราะฉะนั้นการใช้งานเซ็นเซอร์ต่างๆ ต้องมีการศึกษาเพิ่มเติมจากคู่มือการใช้งานเสมอ เนื่องจากส่วนใหญ่แล้วไดร์เวอร์จะขึ้นต้นด้วยชื่อของผู้ผลิตและเวอร์ชั่นเฟิร์มแวร์ของอุปกรณ์นั้นๆ และตัวอย่างการเขียนโค้ดเพื่ออธิบายฮาร์ดแวร์ดังกล่าว แสดงดังรูปที่ 2.25 [13]



รูปที่ 2.23 โครงสร้างการทำงานของ Devicetree



รูปที่ 2.24 ตัวอย่างการระบุฮาร์ดแวร์ใน Devicetree

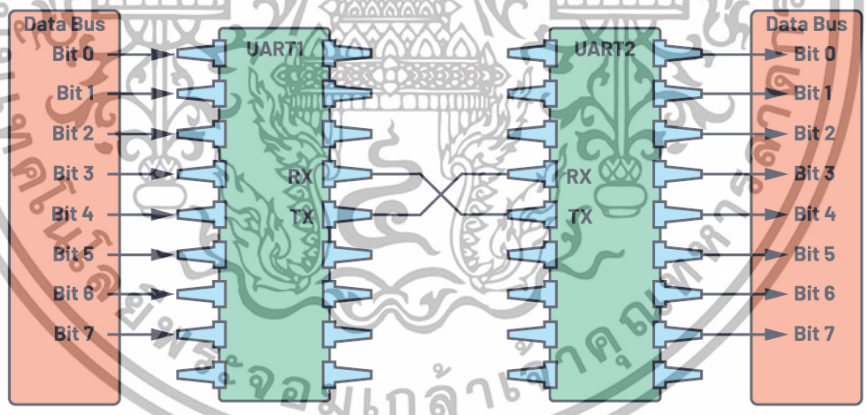
```
/dts-v1/;

/ {
        soc {
                i2c@40003000 {
                        compatible = "nordic,nrf-twim";
                        reg = <0x40003000 0x1000>;

                        apds9960@39 {
                                compatible = "avago,apds9960";
                                reg = <0x39>;
                        };
                        ti_hdc@43 {
                                compatible = "ti,hdc", "ti,hdc1010";
                                reg = <0x43>;
                        };
                        mma8652fc@1d {
                                compatible = "nxp,fxos8700", "nxp,mma8652fc";
                                reg = <0x1d>;
                        };
                };
        };
};
```

รูปที่ 2.25 ตัวอย่างโค้ด Devicetree

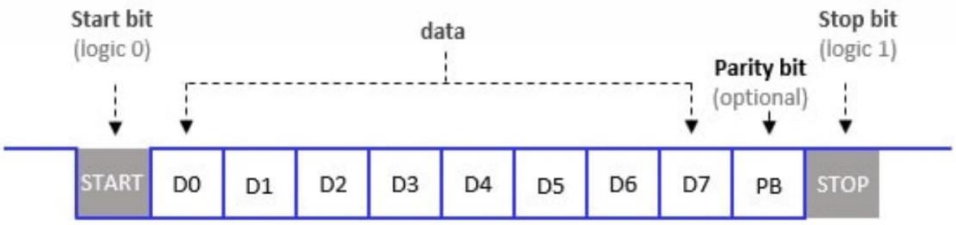
2.9 มาตรฐานการส่งข้อมูลแบบ UART

2.9.1 UART คืออะไร

Universal Asynchronous Receiver Transmitter คือการสื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส สัญญาณนาฬิกาที่เป็นอิสระต่อกัน โดยผู้ใช้สามารถกำหนดรูปแบบที่ใช้ในการรับส่งข้อมูล และกำหนดอัตราเร็วในการรับส่งข้อมูลในหน่วยบิตต่อวินาทีหรือบอดเรท (Baud rate) ให้แก่อุปกรณ์ที่จะติดต่อสื่อสารกันให้มีค่าตรงกันจึงจะสามารถสื่อสารกันได้ สายสัญญาณที่ใช้สาย 2 เส้น คือ Tx ในการส่งข้อมูล และ Rx ในการรับข้อมูล UART transmitting หลักการทำงานคือ Data bus จะส่งข้อมูลให้ UART 1 แบบขนานเมื่อ UART 1 ได้รับข้อมูลก็จะทำการแพ็กข้อมูล โดยเพิ่ม start bits, parity bits และ stop bits รวมเป็นแพ็กเกจข้อมูล จากนั้น ข้อมูลก็จะถูกส่งออกทางขา (Tx) ในรูปแบบอนุกรม UART 2 ก็จะรับข้อมูลเข้ามาทางขา (Rx) ข้อมูลที่ได้รับมาจะนำ start bits, parity bits และ stop bits ออก และจัดเก็บข้อมูลเข้าสู่ Data bus ต่อไปซึ่งการส่งข้อมูลแบบ UART จะทำการส่งข้อมูล LSB ไปก่อนแล้วตามด้วยข้อมูล MSB การต่ออุปกรณ์แบบแสดงดังรูปที่ 2.26 และมีเฟรมการสื่อสาร แสดงดังรูปที่ 2.27 [14]



รูปที่ 2.26 การส่งข้อมูลแบบ UART



รูปที่ 2.27 เฟรมการสื่อสารแบบ UART

2.9.1 วิธีการส่งข้อมูลแบบ UART

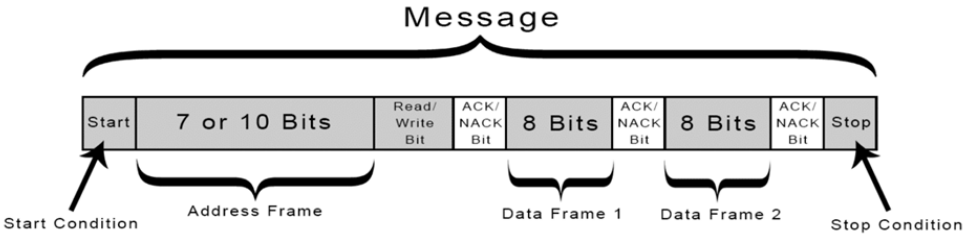
1. Data bus ส่งข้อมูลไปยัง UART transmitting ในรูปแบบขนาน
2. UART transmitting เพิ่ม start bits, parity bits และ stop bits เข้าไปใน Data frame
3. หลังจากที่แพ็กข้อมูลเสร็จก็จะส่งออกไปให้ UART receiving ในรูปแบบอนุกรมซึ่งจะตรวจสอบสัญญาณข้อมูลใน Baud rate ที่กำหนดล่วงหน้า
4. UART receiving จะกรอง start bits, parity bits และ stop bits ออก
5. UART receiving จะทำการเปลี่ยนรูปแบบข้อมูลในรูปแบบอนุกรมในให้อยู่ในรูปแบบขนานและส่งไปยัง Data bus 3

2.10 มาตรฐานการสื่อสาร I2C

การสื่อสาร I2C (Inter-Integrated Circuit) เป็นโปรโตคอลที่มีบทบาทสำคัญในวงการอิเล็กทรอนิกส์ เนื่องจากใช้การสื่อสารผ่านสายเพียงสองเส้น ทำให้เป็นทางเลือกที่ประหยัดและเหมาะสำหรับการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์พ่วงต่าง ๆ ไม่ว่าจะเป็นตัวแปลงข้อมูลดิจิตอลและอนาล็อก หรือเซนเซอร์ชนิดต่าง ๆ

2.10.1 หลักการทำงานของ I2C

I2C เป็นการสื่อสารแบบอนุกรมที่ใช้เส้นสัญญาณเพียงสองเส้น ได้แก่ เส้นสัญญาณข้อมูลอนุกรม (SDA) และเส้นสัญญาณนาฬิกาอนุกรม (SCL) เส้นสัญญาณทั้งสองเส้นนี้เชื่อมต่อกับอุปกรณ์ทุกตัวที่อยู่ในบัสเดียวกัน โดยเส้น SDA ใช้สำหรับการรับส่งข้อมูล และเส้น SCL ใช้สำหรับการซิงโครไนซ์หรือควบคุมจังหวะการส่งข้อมูล ด้วยวิธีการนี้ ไมโครคอนโทรลเลอร์หรือคอนโทรลเลอร์ (Controller) สามารถสื่อสารกับอุปกรณ์พ่วง (Target) หลายตัวในบัสได้โดยใช้เพียงสองเส้นสัญญาณเท่านั้นในการเริ่มต้นสื่อสาร คอนโทรลเลอร์จะสร้างสัญญาณเริ่มต้น (START Condition) โดยดึงเส้น SDA ลงต่ำ แล้วตามด้วยการดึงเส้น SCL ลงต่ำ การกระทำนี้เป็นสัญญาณว่าอุปกรณ์อื่น ๆ ในบัสควรรอให้การสื่อสารเสร็จสิ้น ก่อนที่จะปล่อยให้มีการสื่อสารเพิ่มเติม เมื่อสิ้นสุดการสื่อสารแล้ว คอนโทรลเลอร์จะปล่อยเส้น SCL ขึ้นก่อน และตามด้วยการปล่อยเส้น SDA ขึ้นซึ่งเป็นสัญญาณสิ้นสุดการสื่อสาร (STOP Condition) โดยมีลักษณะการสื่อสารแสดงดังรูปที่ 2.28 [30]



รูปที่ 2.28 Frame Format การสื่อสาร I2C

2.10.2 โหมดความเร็วของ I2C

I2C รองรับหลายโหมดความเร็วเพื่อให้เหมาะกับการใช้งานที่หลากหลาย ตั้งแต่โหมดพื้นฐาน (Standard Mode) ซึ่งส่งข้อมูลได้สูงสุดที่ความเร็ว 100 กิโลบิตต่อวินาที (kbps) ไปจนถึง Ultra-Fast Mode ที่สามารถส่งข้อมูลได้สูงถึง 5 เมกะบิตต่อวินาที (Mbps) ในบางโหมดความเร็วสูง เช่น High-Speed Mode อาจต้องการรหัสการเริ่มต้นพิเศษเพื่อให้ระบบรองรับการสื่อสารที่ความเร็วสูงนี้ได้ โดยแต่ละโหมดการทำงานจะมีความแตกต่างกันในด้านสัญญาณไฟฟ้าและวิธีการตั้งค่าในฮาร์ดแวร์

2.11 โพรโทคอล MQTT

MQTT (Message Queuing Telemetry Transport) เป็นโปรโตคอลที่ออกแบบมาเพื่อลดการใช้แบนด์วิดท์และความต้องการของทรัพยากรในระบบที่เชื่อมต่อระหว่างเครื่องจักร (Machine-to-Machine หรือ M2M) และการสื่อสารของอุปกรณ์ IoT (Internet of Things) ด้วยการใช้โค้ดที่น้อยและการใช้เครือข่ายที่มีขนาดเล็ก

2.11.1 ประวัติและวัตถุประสงค์

MQTT ถูกพัฒนาโดย Dr. Andy Stanford-Clark และ Arlen Nipper ในปี 1999 โดยมีวัตถุประสงค์เริ่มต้นเพื่อให้การติดตามและตรวจสอบอุปกรณ์ในอุตสาหกรรมน้ำมันและก๊าซสามารถส่งข้อมูลไปยังเซิร์ฟเวอร์ที่ห่างไกลได้ อุปกรณ์เหล่านี้มักถูกใช้ในสถานที่ที่ยากต่อการเชื่อมต่อด้วยสายสัญญาณ, การเชื่อมต่อแบบไร้สาย, หรือการส่งข้อมูลผ่านคลื่นวิทยุ ซึ่งในขณะนั้นการใช้เทคโนโลยีดาวเทียมเป็นทางเลือกเดียวที่มีอยู่ แต่มีค่าใช้จ่ายสูงและคิดค่าบริการตามปริมาณข้อมูลที่ใช้

2.11.2 การทำงานและข้อดี

MQTT ถูกออกแบบมาให้ทำงานในสภาพแวดล้อมที่มีข้อจำกัดด้านแบนด์วิดท์และการใช้ทรัพยากรของ CPU ด้วยความเบาและมีประสิทธิภาพ จึงเป็นทางเลือกที่ดีสำหรับการเชื่อมต่อไร้สายที่

อาจมีความล่าช้าหรือเชื่อมต่อไม่เสถียร โดยไม่สนใจประเภทของข้อมูลใน payload นอกจากนี้ยังเหมาะสำหรับการเชื่อมต่ออุปกรณ์ที่มีโค้ดขนาดเล็ก

โปรโตคอล MQTT ทำงานบนโปรโตคอลอินเทอร์เน็ตเช่น Transmission Control Protocol (TCP/IP) ที่ช่วยให้การสื่อสารแบบสองทางเป็นไปอย่างมีระเบียบและราบรื่น ส่วน MQTT-SN v1.2 หรือ MQTT สำหรับเครือข่ายเซ็นเซอร์ (MQTT-S) เป็นเวอร์ชันที่ออกแบบมาเพื่อใช้กับระบบฝังตัวที่ใช้เครือข่ายที่ไม่ใช่ TCP/IP

2.11.3 วิธีการทำงาน

MQTT เป็นโปรโตคอลแบบ Publisher/Subscriber โดยมี Broker คอยจัดการการสื่อสาร ซึ่งหมายความว่าไม่มีการเชื่อมต่อโดยตรงระหว่างผู้ส่งและผู้รับ ข้อความจะถูกส่งในรูปแบบ 1-to-1 หรือ 1-to-n และผู้รับที่สนใจข้อความที่เกี่ยวข้องจะรับข้อมูลตามหัวข้อที่กำหนด

2.11.4 การใช้งาน

MQTT ใช้กันอย่างแพร่หลายในการเชื่อมต่ออุปกรณ์ในหลายอุตสาหกรรม เช่น ยานยนต์, อุตสาหกรรม, การสื่อสาร, และการใช้งานในพื้นที่ที่มีการควบคุมหรือการตรวจสอบที่ห่างไกล เช่น อุตสาหกรรมการผลิตน้ำมันและก๊าซ

ในปัจจุบัน MQTT เป็นโปรโตคอลที่ได้รับการยอมรับจาก OASIS (Organization for the Advancement of Structured Information Standards) สำหรับการสื่อสาร IoT และ IIoT (Industrial Internet of Things) เป็นโปรโตคอลที่เปิดเผยและเหมาะสำหรับการเชื่อมต่ออุปกรณ์ในระยะไกลด้วยโค้ดและทรัพยากรเครือข่ายที่น้อยที่สุด

2.11.5 การทำงานของโพรโทคอล MQTT

MQTT ทำงานโดยใช้รูปแบบการสื่อสารแบบ Publish/Subscribe (pub/sub) ซึ่งช่วยเพิ่มประสิทธิภาพการใช้แบนด์วิดท์ และแตกต่างจากสถาปัตยกรรมการสื่อสารแบบดั้งเดิมที่มีการโต้ตอบระหว่างลูกค้ากับจุดสิ้นสุดโดยตรง ในระบบ pub/sub ลูกค้าผู้ส่งข้อมูล (publisher) จะแยกจากลูกค้าที่รับข้อมูล (subscriber) โดยการสื่อสารระหว่างพวกเขาจะถูกจัดการโดยบุคคลที่สามที่เรียกว่า Broker

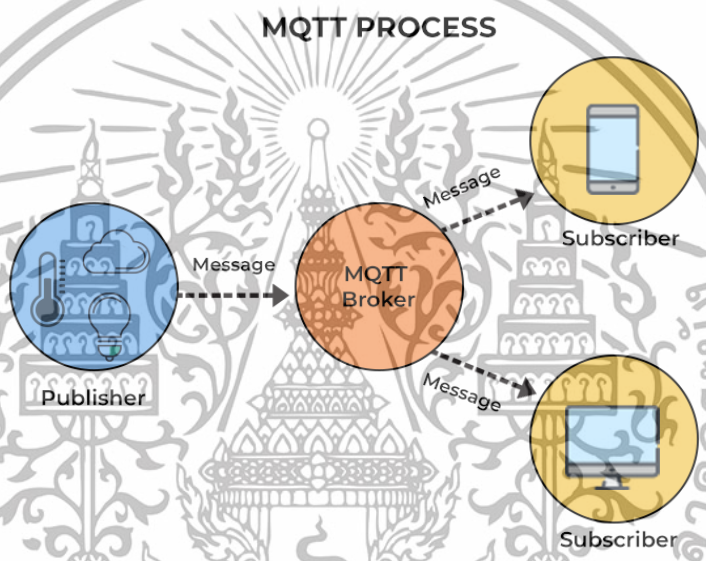
1. Publisher และ Subscriber เป็นสองประเภทของผู้ใช้งานใน MQTT ซึ่งขึ้นอยู่กับว่าผู้ใช้งานส่งข้อมูลหรือรับข้อมูล ผู้ใช้งานอาจทำหน้าที่ทั้งสองอย่างได้ในอุปกรณ์เดียว

- Publisher คืออุปกรณ์ (หรือไคลเอนต์) ที่ต้องการส่งข้อมูลไปยังเซิร์ฟเวอร์หรือ Broker

- Subscriber คืออุปกรณ์ที่สมัครรับข้อมูลตามหัวข้อ (topic) ที่สนใจ โดยอาจกรองตามหัวข้อ เนื้อหา หรือประเภทข้อมูล

2. Broker คือซิร์ฟเวอร์ส่วนกลางที่ทำหน้าที่เป็นตัวกลางในการกรองและส่งข้อมูลไปยังผู้ใช้งานที่เกี่ยวข้อง โดยไม่ต้องให้ Publisher และ Subscriber รู้จักกันหรือต้องการการตั้งค่าและการซิงโครไนซ์ระหว่างกันโดยตรง

MQTT เป็นโครงสร้างที่เหมาะสมสำหรับการส่งข้อมูลแบบ หนึ่งไปยังหลายคน (one-to-many) โดยสามารถขยายจำนวนผู้ใช้ได้โดยไม่เพิ่มภาระให้กับ Publisher แสดงดังรูปที่ 2.29 [15]



รูปที่ 2.29 ตัวอย่างการทำงานของโพรโทคอล MQTT

2.11.6 คุณภาพของการบริการ (Quality of Service - QoS)

MQTT มี ระดับ QoS 3 ระดับเพื่อช่วยควบคุมการรับส่งข้อมูลและเพิ่มความน่าเชื่อถือของข้อมูลดังนี้ [16]

1. QoS 0: เป็นการส่งข้อมูลขั้นต่ำสุด ข้อความจะถูกส่งไปยังผู้ใช้งานเพียงครั้งเดียวโดยไม่มีการยืนยันการรับข้อมูล วิธีนี้เรียกว่า “Fire and Forget” หรือ “ส่งเพียงครั้งเดียวเท่านั้น” (At most once delivery) ไม่มีการเก็บข้อมูลไว้หากผู้ใช้งานตัดการเชื่อมต่อ

2. QoS 1: ข้อความจะถูกส่งและรอการตอบรับจากผู้ใช้งาน หากไม่มีการตอบรับข้อความจะถูกส่งซ้ำ วิธีนี้เรียกว่า “ส่งอย่างน้อยหนึ่งครั้ง” (At least once delivery) ข้อความอาจถูกส่งซ้ำได้หลายครั้งหากไม่มีการยืนยัน

3. QoS 2: มีการใช้ขั้นตอนยืนยันสี่ขั้นตอนเพื่อให้แน่ใจว่าข้อมูลถูกส่งไปเพียงครั้งเดียว วิธีนี้เรียกว่า “ส่งอย่างแม่นยำหนึ่งครั้ง” (Exactly once delivery) เหมาะสำหรับการสื่อสารที่ไม่เสถียรแต่ไม่ต้องกังวลเกี่ยวกับทรัพยากร

QoS 1 และ QoS 2 จะเก็บข้อความสำหรับผู้ใช้งานที่ตัดการเชื่อมต่อ เมื่อผู้ใช้กลับมาเชื่อมต่อ ข้อความเหล่านี้จะถูกส่งใหม่ในระดับ QoS ที่เหมาะสม

2.11.7 ประเภทของข้อความในโพรโทคอล MQTT

MQTT มีข้อความมาตรฐานทั้งหมด 14 ประเภท ซึ่งใช้ในการเชื่อมต่อและตัดการเชื่อมต่อผู้ใช้งานกับ Broker, ส่งข้อมูล, ยืนยันการรับข้อมูล และรักษาการเชื่อมต่อระหว่างผู้ใช้งานกับเซิร์ฟเวอร์ โดย MQTT ใช้โปรโตคอล TCP ในการส่งข้อมูล อย่างไรก็ตาม ข้อความหลักที่ใช้บ่อยมี 3 ประเภท ได้แก่:

1. Connect (การเชื่อมต่อ)

ข้อความนี้ใช้สำหรับส่งคำขอเชื่อมต่อจากผู้ใช้งานไปยัง Broker เนื่องจาก MQTT ออกแบบมาให้ทำงานบนอุปกรณ์ IoT ที่มีทรัพยากรจำกัด การใช้ SSL/TLS อาจไม่สามารถทำได้หรือไม่จำเป็นในบางกรณี ผู้ใช้งานสามารถยืนยันตัวตนผ่านข้อมูลเข้าสู่ระบบแบบข้อความธรรมดา (cleartext) ซึ่งจะถูกส่งไปยังเซิร์ฟเวอร์เป็นส่วนหนึ่งของแพ็กเกจ CONNECT นอกจากนี้ บาง Broker อาจอนุญาตให้ไคลเอนต์ไม่ระบุตัวตน โดยเฉพาะเมื่อให้บริการผ่านอินเทอร์เน็ต ซึ่งในกรณีนี้ข้อมูลเข้าสู่ระบบจะถูกเว้นว่างไว้

2. Publish (การส่งข้อมูล)

ข้อความนี้ใช้เมื่อผู้ใช้งานต้องการส่งข้อมูลไปยัง Broker โดยข้อมูลที่ส่งไปอาจเป็นอะไรก็ได้ เช่น สถานะเปิด/ปิด หรือข้อมูลจากเซ็นเซอร์ เช่น อุณหภูมิหรือความดัน ข้อมูลนี้จะถูกส่งไปยัง Broker ในกรณีที่หัวข้อที่เกี่ยวข้องยังไม่มีอยู่ การออกแบบในรูปแบบ Pub/Sub ช่วยให้ผู้ส่งข้อมูล (Publisher) และผู้รับข้อมูล (Subscriber) ไม่จำเป็นต้องติดต่อกันโดยตรง เพราะ Broker จะเป็นผู้จัดการการเชื่อมต่อ รับข้อความ และแจกจ่ายไปยัง Subscriber ที่สนใจหัวข้อนั้นๆ

3. Subscribe (การสมัครรับข้อมูล)

ข้อความนี้ใช้เมื่อผู้ใช้งานต้องการรับข้อมูลจาก Broker โดยการสมัครรับหัวข้อที่ต้องการ ผู้ใช้งานสามารถสมัครรับทีละหัวข้อหรือใช้ Wildcards เพื่อสมัครรับข้อมูลจากหลายๆ หัวข้อได้ในครั้งเดียว เมื่อผู้ใช้งานส่งคำสั่ง SUBSCRIBE ไปยัง Broker ผู้ใช้งานจะได้รับการตอบกลับด้วยข้อความ SUBACK และหากมีข้อความที่เก็บไว้ในหัวข้อนั้น ผู้ใช้งานใหม่ที่สมัครรับจะได้รับข้อความดังกล่าวทันที [15]

2.11.8 ความสำคัญของ MQTT ใน IoT

MQTT เป็นโปรโตคอลที่สำคัญสำหรับ IoT ด้วยเหตุผลหลักๆ ดังนี้ [16]

1. โปรโตคอลที่มีน้ำหนักเบา คือ MQTT เป็นโปรโตคอลที่มีน้ำหนักเบา ซึ่งทำให้มันเหมาะกับอุปกรณ์ IoT ขนาดเล็กและราคาถูกที่มีพลังงานต่ำ เมื่อเปรียบเทียบกับ HTTP ซึ่งมีหัวขนาดใหญ่ถึงประมาณ 8000 ไบต์ ส่วนหัวของ MQTT มีขนาดเพียง 2 ไบต์ และใช้โค้ดไม่กี่บรรทัด ซึ่งเหมาะสำหรับอุปกรณ์ที่มีความต้องการด้านหน่วยความจำและพลังงานต่ำ

2. ประหยัดพลังงาน คือ ตอนที่ IBM พัฒนา MQTT มันถูกออกแบบมาเพื่อใช้งานในสภาพแวดล้อมที่ไม่มีไฟฟ้า เช่น พื้นที่ห่างไกล ดังนั้นจึงต้องใช้พลังงานต่ำที่สุดเมื่อเปรียบเทียบกับโปรโตคอลอื่นๆ MQTT ใช้พลังงานน้อยกว่าการเชื่อมต่อ 3G ถึง 170 เท่า และน้อยกว่า Wi-Fi ถึง 47 เท่า ทำให้สามารถทำงานได้ยาวนานหลายปีด้วยแบตเตอรี่เดียว

3. รองรับข้อมูลทุกประเภท คือ MQTT สามารถส่งข้อมูลได้ทุกรูปแบบ ไม่ว่าจะเป็นข้อมูลแบบไบนารีหรือข้อความ ไม่สำคัญว่าข้อมูลจะเป็นแบบไหน ขอแค่ฝ่ายที่รับรู้วิธีตีความข้อมูลนั้นๆ

4. ความยืดหยุ่นในเรื่องเวลา คือ อุปกรณ์ที่เชื่อมต่อสามารถส่งข้อมูลได้โดยไม่ต้องกังวลเกี่ยวกับสถานะของเซิร์ฟเวอร์ที่สมัครรับข้อมูล เซิร์ฟเวอร์สามารถเชื่อมต่อและรับข้อมูลเมื่อมันพร้อม

5. เชื่อถือได้และขยายตัวได้ดี คือ MQTT เชื่อถือได้เพราะมีการตั้งค่าคุณภาพการบริการ (QoS) ที่รับประกันการส่งข้อความ และรูปแบบการเผยแพร่และสมัครรับข้อมูลช่วยให้สามารถขยายตัวได้ดี เพราะไม่จำเป็นต้องมีการสื่อสารโดยตรงระหว่างผู้เผยแพร่และผู้รับข้อมูล

6. การออกแบบที่แยกจากกัน คือ MQTT มีการออกแบบที่แยกอุปกรณ์และเซิร์ฟเวอร์ที่สมัครรับข้อมูลออกจากกัน ซึ่งทำให้ระบบการสื่อสารมีความแข็งแกร่งและมีประสิทธิภาพ

2.12 แอปพลิเคชันหน้าเดียว (Single-Page Application; SPA)

แอปพลิเคชันหน้าเดียว (Single Page Application หรือ SPA) หมายถึงเว็บแอปพลิเคชันหรือเว็บไซต์ที่ทำงานอยู่บนหน้าเว็บเพียงหน้าเดียว โดยมีจุดมุ่งหมายเพื่อให้ผู้ใช้ได้รับประสบการณ์การใช้งานที่ลื่นไหล คล้ายกับการใช้แอปพลิเคชันบนเดสก์ท็อป วิธีการนี้ทำได้โดยการส่งเฉพาะข้อมูลที่ต้องแสดงผล แทนที่จะต้องโหลดหน้าใหม่ทุกครั้งที่มีการโต้ตอบกับผู้ใช้ SPAs มักใช้เทคโนโลยี AJAX (Asynchronous JavaScript and XML) และบางครั้งก็ใช้ WebSockets เพื่อให้สามารถอัปเดตและโต้ตอบกับเนื้อหาได้แบบไดนามิกภายในหน้าเดียวกัน

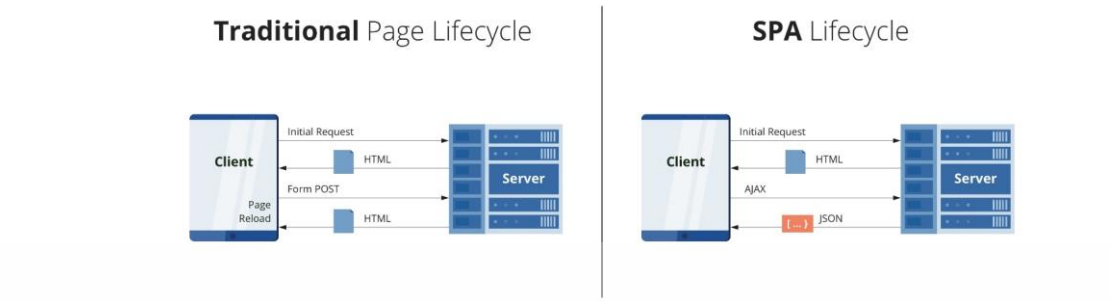
SPAs จะโหลดไฟล์ HTML แค่ไฟล์เดียว และเขียนเนื้อหาในหน้านั้นใหม่แบบไดนามิกตามการโต้ตอบของผู้ใช้ โดยการโต้ตอบทั้งหมดหลังจากนั้นจะใช้ JavaScript และการเรียกข้อมูลด้วย AJAX จากเซิร์ฟเวอร์ที่ทำหน้าที่จัดการข้อมูลและตรรกะของระบบ วิธีนี้แตกต่างจากการทำงานแบบหลายหน้า (Multi-Page) ที่ทุกครั้งที่มีการเปลี่ยนแปลงในอินเทอร์เฟซผู้ใช้ (UI) จะทำให้ต้องโหลดหน้าใหม่ทั้งหมดจากเซิร์ฟเวอร์แสดงดังรูปที่ 2.30 [17]

รูปที่ 2.30 แอปพลิเคชันหน้าเดียว (Single-Page Application - SPA)

ในแอปพลิเคชันหน้าเดียว (Single Page Application หรือ SPA) เมื่อเบราว์เซอร์ส่งคำขอครั้งแรกไปยังเซิร์ฟเวอร์ เซิร์ฟเวอร์จะส่งไฟล์ index.html กลับมา นั่นคือครั้งเดียวที่ไฟล์ HTML จะถูกส่งจากเซิร์ฟเวอร์ โดยในไฟล์ HTML นั้นจะมีแท็กสคริปต์สำหรับไฟล์ .js ที่จะเข้ามาควบคุมหน้า index.html หลังจากนั้น ทุกครั้งที่มีการร้องขอข้อมูลใหม่ จะเป็นการส่งข้อมูลในรูปแบบ JSON กลับมาแทน HTML ซึ่งแอปพลิเคชันจะใช้ข้อมูล JSON เหล่านี้เพื่ออัปเดตเนื้อหาในหน้าเว็บแบบไดนามิก โดยที่หน้าเว็บจะไม่ถูกรีเฟรชเลย

ใน SPA หน้าที่ของการแปลงข้อมูลให้กลายเป็น HTML นั้นจะเป็นของฝั่งผู้ใช้ (Client) ไม่ใช่เซิร์ฟเวอร์ กล่าวคือ เมื่อแอปพลิเคชันเริ่มทำงาน ส่วนใหญ่เฟรมเวิร์ก SPA สมัยใหม่จะมีระบบเทมเพลตที่รันอยู่ในเบราว์เซอร์เพื่อสร้าง HTML ให้กับหน้าเว็บ

หากเปรียบเทียบกับแอปพลิเคชันเว็บแบบดั้งเดิม ทุกครั้งที่มีการเรียกใช้เซิร์ฟเวอร์ แอปพลิเคชันจะให้เซิร์ฟเวอร์เรนเดอร์ HTML ใหม่ทั้งหน้า แล้วส่งกลับไปยังฝั่งผู้ใช้ ซึ่งทำให้เกิดการรีเฟรชหน้าเว็บขึ้น ในกรณีนี้ เบราว์เซอร์ทำหน้าที่เป็นผู้ใช้ (Client) ที่รับหน้าที่แสดงผล แสดงดังรูปที่ 2.31 [18]



รูปที่ 2.31 เปรียบเทียบกับแอปพลิเคชันเว็บแบบดั้งเดิมกับแอปพลิเคชันหน้าเดียว

2.12.1 ข้อดีของแอปพลิเคชันหน้าเดียว

ประโยชน์ของแอปพลิเคชันหน้าเดียว (Single-Page Application หรือ SPA) ที่สำคัญมีดังนี้ [19]

1. ความเร็วและการตอบสนองที่ดีขึ้น คือ SPA โหลดเฉพาะข้อมูลที่จำเป็นตามการกระทำของผู้ใช้แทนที่จะโหลดทั้งหน้าใหม่ทั้งหมด จึงทำให้เวลาโหลดเร็วขึ้น

2. เสถียรภาพที่มากขึ้น คือ รวมถึงความสามารถในการแคชและการใช้แบนด์วิธที่น้อยลง ทำให้หน้าเว็บยังสามารถแสดงผลได้แม้การเชื่อมต่ออินเทอร์เน็ตไม่เสถียร

3. ประสบการณ์ผู้ใช้ที่ดีขึ้น (UX) คือ เนื่องจากหน้าเว็บโหลดได้เร็วและตอบสนองต่อการโต้ตอบของผู้ใช้ได้ดี ทำให้ผู้ใช้มีส่วนร่วมมากขึ้นและได้รับประสบการณ์ที่ดียิ่งขึ้นในการใช้งานเนื้อหาต่าง ๆ

4. การพัฒนาที่เร็วขึ้น คือ SPA สามารถใช้ API ที่หลากหลาย ซึ่งช่วยให้ทีมพัฒนาทำงานแยกกันระหว่างส่วนหน้าบ้าน (front-end), หลังบ้าน (back-end) และการเชื่อมโยงข้อมูลได้อย่างต่อเนื่อง

5. การดีบักที่ง่ายขึ้น คือ โค้ดของ SPA มักถูกจัดเป็นโมดูล ทำให้การทดสอบและแก้ไขปัญหาทำได้ง่ายขึ้น รวมถึงช่วยให้ทีมพัฒนาสามารถทำงานแยกกันในส่วนต่าง ๆ ได้พร้อมกัน

6. รองรับการใช้งานข้ามแพลตฟอร์ม คือ SPA ใช้โค้ดชุดเดียว ทำให้สามารถออกแบบให้ทำงานได้บนทุกแพลตฟอร์มหรือเบราว์เซอร์ ผู้ใช้จึงสามารถใช้งานได้อย่างราบรื่นเมื่อต้องเปลี่ยนอุปกรณ์

7. รองรับการใช้งานบนมือถือได้ดี คือ โค้ดสามารถนำกลับมาใช้ซ้ำในการออกแบบแอปบนมือถือและหน้าเว็บที่ตอบสนองต่อการแสดงผลบนอุปกรณ์มือถือได้ดีเท่ากับบนคอมพิวเตอร์เดสก์ท็อป ช่วยให้ผู้ใช้มือถือได้รับประสบการณ์ที่ไม่ด้อยไปกว่าผู้ใช้บนเดสก์ท็อป

2.13 การพัฒนาเว็บแอปพลิเคชัน (Web Development)

Web Development คือ กระบวนการในการสร้างและพัฒนาเว็บไซต์ โดยอาศัยภาษาโปรแกรมและเครื่องมือเฉพาะสำหรับการออกแบบ จัดการเนื้อหา และฟังก์ชันต่าง ๆ ของเว็บไซต์ กระบวนการนี้ครอบคลุมตั้งแต่การวางแผนโครงสร้าง การออกแบบหน้าเว็บ การพัฒนาเชิงเทคนิค ไปจนถึงการดูแลรักษาและปรับปรุงเว็บไซต์หลังจากเปิดใช้งานแล้ว การพัฒนาเว็บไซต์มีบทบาทสำคัญในยุคดิจิทัล ซึ่งช่วยสร้างความน่าเชื่อถือและการรับรู้แบรนด์ของธุรกิจ อีกทั้งยังทำให้ผู้ใช้งานเข้าถึงข้อมูลและโต้ตอบกับระบบได้อย่างมีประสิทธิภาพมากขึ้น

ปัจจุบัน Web Development แบ่งออกเป็น 3 ประเภทหลัก คือ Front-end Development, Back-end Development, และ Full-stack Development ซึ่งแต่ละประเภทมีหน้าที่และบทบาทที่แตกต่างกัน โดยมุ่งเน้นในการพัฒนาเว็บไซต์ให้มีประสิทธิภาพสูงสุด

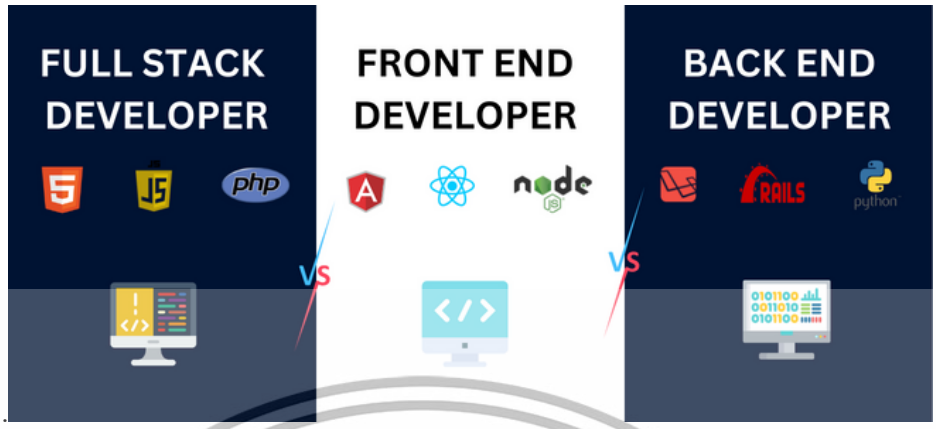
2.13.1 ประเภทของการพัฒนาเว็บแอปพลิเคชัน

การพัฒนาเว็บไซต์สามารถทำได้หลายวิธี ขึ้นอยู่กับวัตถุประสงค์ของการใช้งาน โดยสามารถแบ่งประเภท 3 ประเภทได้ดังนี้ แสดงดังรูปที่ 2.32

1. Front-end Development เป็นการพัฒนาระบบส่วนหน้าบ้าน ซึ่งเป็นส่วนที่ผู้ใช้สามารถมองเห็นและโต้ตอบได้ เช่น หน้าแรกของเว็บไซต์ เนื้อหาต่าง ๆ รูปภาพ หรือแบนเนอร์ ภาษาที่ใช้ทำงานหลักในส่วนนี้ ได้แก่ HTML, CSS และ JavaScript

2. Back-end Development เป็นการพัฒนาระบบส่วนหลังบ้าน ซึ่งผู้ใช้ไม่สามารถมองเห็นหรือเข้าถึงได้โดยตรง ส่วนนี้มีหน้าที่จัดการฐานข้อมูลและโครงสร้างพื้นฐานของเว็บไซต์ ภาษาที่นิยมใช้ในการพัฒนาส่วนนี้ ได้แก่ PHP, Ruby, และ Python

3. Full-stack Development เป็นกการพัฒนาระบบแบบครบวงจรทั้งส่วนหน้าบ้านและหลังบ้าน นักพัฒนา Full-stack ต้องมีความรู้ทั้งในภาษาที่ใช้ใน Front-end และ Back-end เช่น HTML, CSS, JavaScript, PHP, Ruby, และ Python ตำแหน่งนี้มีความต้องการสูง เนื่องจากสามารถจัดการงานทั้งสองส่วนได้อย่างครอบคลุม



รูปที่ 2.32 ตัวอย่างภาษาที่ใช้ในแต่ละประเภท

2.13.2 จุดเด่นของแต่ละประเภท

จุดเด่นของการพัฒนาเว็บแอปพลิเคชันแต่ละประเภทดังนี้ [20]

1. Front-end Development โครงสร้างเว็บไซต์จะถูกสร้างจาก HTML ซึ่งเป็นพื้นฐานของทุกเว็บไซต์ โดยเน้นการสร้างประสบการณ์ที่ดีให้กับผู้ใช้
2. Back-end Development เน้นการทำงานเกี่ยวกับเทคโนโลยีด้านหลังของเว็บไซต์ เช่น การจัดการฐานข้อมูล ซึ่งต้องใช้ทักษะเฉพาะทางมากกว่า
3. Full-stack Development ครอบคลุมการทำงานทั้งสองส่วน นักพัฒนา Full-stack มีความสามารถในการใช้เครื่องมือและภาษาหลักของทั้ง Front-end และ Back-end

2.13.3 ส่วนของBackend

2.13.4.1 แพลตฟอร์มที่ใช้

Node.js คือสภาพแวดล้อมการทำงานบนเซิร์ฟเวอร์ที่เป็นโอเพ่นซอร์ส ซึ่งใช้ JavaScript ในฝั่งเซิร์ฟเวอร์ โดยแอปพลิเคชันที่เขียนด้วย Node.js จะทำงานในกระบวนการเดียวโดยไม่ต้องสร้างเธรดใหม่สำหรับแต่ละคำขอ นอกจากนี้ Node.js ยังมีระบบ I/O แบบ asynchronous เป็นส่วนหนึ่งของไลบรารีมาตรฐาน ซึ่งช่วยป้องกันการบล็อกโค้ด JavaScript โดยทั่วไปแล้วไลบรารีใน Node.js จะพัฒนาด้วยแนวคิดที่ไม่บล็อก ทำให้พฤติกรรมการบล็อกเกิดขึ้นน้อยมาก

Node.js แสดงดังรูปที่ 2.33 ถูกพัฒนาขึ้นโดย Ryan Dahl ในปี 2009 และในปัจจุบันเวอร์ชันล่าสุดคือ v22.4.1 ความสามารถข้ามแพลตฟอร์มของ Node.js ทำให้สามารถใช้งานได้ง่ายบน Windows, Linux, Unix, macOS และอีกหลายระบบ

ข้อได้เปรียบของ Node.js คือ นักพัฒนา frontend หลายล้านคนที่คุ้นเคยกับ JavaScript ในการเขียนโค้ดฝั่งเบราว์เซอร์ สามารถนำทักษะที่มีมาพัฒนาโค้ดฝั่งเซิร์ฟเวอร์ได้ โดยไม่จำเป็นต้องเรียนรู้ภาษาใหม่ ส่งผลให้ Node.js กลายเป็นตัวเลือกยอดนิยมสำหรับการพัฒนา RESTful APIs, microservices และเว็บแอปพลิเคชัน [21]

รูปที่ 2.33 สัญลักษณ์ของ Node.js

คุณสมบัติเด่นของ Node.js ดังนี้

1.JavaScript ครอบคลุมทั้งระบบ: Node.js ช่วยให้นักพัฒนาสามารถใช้ JavaScript ได้ทั้งในฝั่ง front-end และ back-end ทำให้การพัฒนาโค้ดสอดคล้องกันและลดการสลับบริบทในการทำงาน

2.โมเดลการทำงานแบบ asynchronous: Node.js ใช้รูปแบบการทำงานที่ไม่บล็อก (asynchronous I/O) ซึ่งสามารถจัดการคำขอหลาย ๆ คำขอพร้อมกันได้โดยไม่ขัดจังหวะการทำงานอื่น ๆ ส่งผลให้แอปพลิเคชันทำงานได้รวดเร็วและมีประสิทธิภาพ

3.การประมวลผลที่รวดเร็ว: Node.js ใช้ V8 engine ของ Google ที่สามารถคอมไพล์และรันโค้ด JavaScript ได้อย่างรวดเร็ว จึงเหมาะสำหรับการพัฒนาแอปพลิเคชันเรียลไทม์และ microservices

4.ชุมชนที่มีขนาดใหญ่และมีความเคลื่อนไหวตลอดเวลา: Node.js มีชุมชนนักพัฒนาที่ใหญ่และหลากหลาย ซึ่งมีไลบรารีและเครื่องมือมากมายคอยช่วยเสริมการพัฒนา รวมถึงแหล่งข้อมูลและบทเรียนจำนวนมากที่จะช่วยให้การเรียนรู้เป็นไปได้ง่ายขึ้น

5.ความสามารถในการขยายตัวที่ดี: Node.js เป็นแพลตฟอร์มที่มีน้ำหนักเบาและขยายตัวได้ง่าย เหมาะสำหรับการพัฒนาแอปพลิเคชันเรียลไทม์, RESTful APIs และ microservices

6.รองรับหลายแพลตฟอร์ม: Node.js สามารถทำงานได้บน Windows, Linux, Unix, macOS และอื่น ๆ ซึ่งทำให้นักพัฒนาสามารถเขียนโค้ดครั้งเดียวแล้วนำไปใช้ได้ในหลายระบบ

2.13.4 ส่วนของ Frontend

2.13.4.1 แพลตฟอร์มที่ใช้

React JS เป็นไลบรารี JavaScript ที่ได้รับความนิยมอย่างสูงสำหรับการพัฒนาเว็บแอปพลิเคชัน โดยมีการดูแลและพัฒนาอย่างต่อเนื่องจาก Meta และชุมชนนักพัฒนาทั่วโลก มีชื่อเสียงในเรื่องความเร็ว ความสามารถในการขยายตัว และความเรียบง่าย ซึ่งช่วยให้สามารถเปลี่ยนข้อมูลได้โดยไม่ต้องรีโหลดหน้าใหม่ แสดงดังรูปที่ 2.34 [22]

รูปที่ 2.34 สัญลักษณ์ของ React.js

คุณสมบัติเด่นของ React ดังนี้

1.สถาปัตยกรรมแบบคอมโพเนนต์: React มีโครงสร้างที่เน้นการสร้างคอมโพเนนต์ ทำให้สามารถสร้างคอมโพเนนต์ UI ที่สามารถนำกลับมาใช้ใหม่ได้ ช่วยให้การจัดการและบำรุงรักษาเว็บแอปพลิเคชันที่ซับซ้อนได้ง่ายขึ้น

2.การสร้างแอปพลิเคชันแบบหน้าเดียว (SPA): React เหมาะสำหรับการสร้างแอปพลิเคชันที่โหลดเนื้อหาบนหน้าเดียว ซึ่งทำให้ผู้ใช้สามารถทำงานได้อย่างราบรื่นโดยไม่ต้องโหลดหน้าใหม่ทุกครั้ง

บทที่ 3

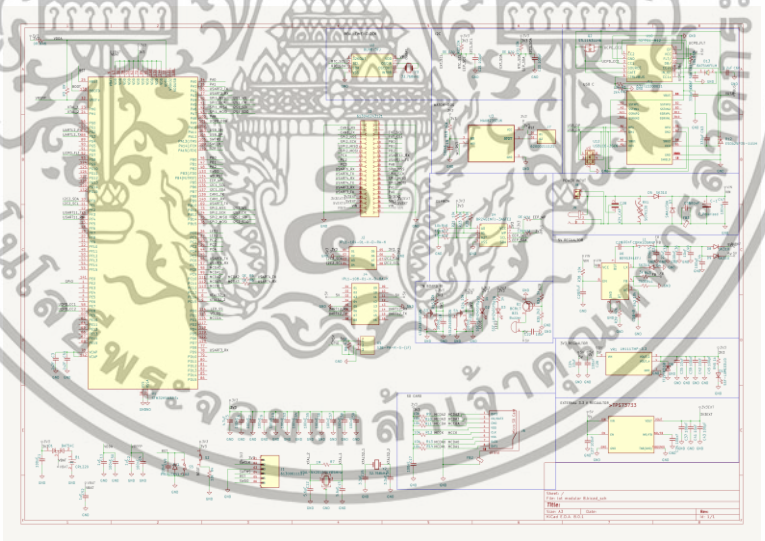
การออกแบบและการจัดทำปริญญานิพนธ์

3.1 การออกแบบ

3.1.1 การออกแบบอุปกรณ์

3.1.1.1 การออกแบบ Core Board

Schematic การทำงานภายใน Core Board อุปกรณ์ต่างๆ และ การเชื่อมต่อของอุปกรณ์แต่ละอุปกรณ์เข้าหากัน โดยในแต่ละอุปกรณ์ต้องคำนึงถึงความเหมาะสมของการใช้งาน ข้อจำกัด รวมถึงการเข้ากันได้ของอุปกรณ์ การสื่อสารในแต่ละอุปกรณ์ใช้การสื่อสารแบบใดเพื่อให้การทำงานมีประสิทธิภาพสูงสุด โดยในการออกแบบการเชื่อมต่อขาของ STM32H563Zi ต้องศึกษาการสื่อสารโดยสามารถดูได้จากคู่มือการใช้งานและเลือกการติดต่อสื่อสารผ่าน Alternate Functions โดยจะต้องดูจากความเข้ากันได้ของอุปกรณ์ที่ต้องการใช้ โดย Core Board ของระบบ IoT แสดงดังรูปที่ 3.1

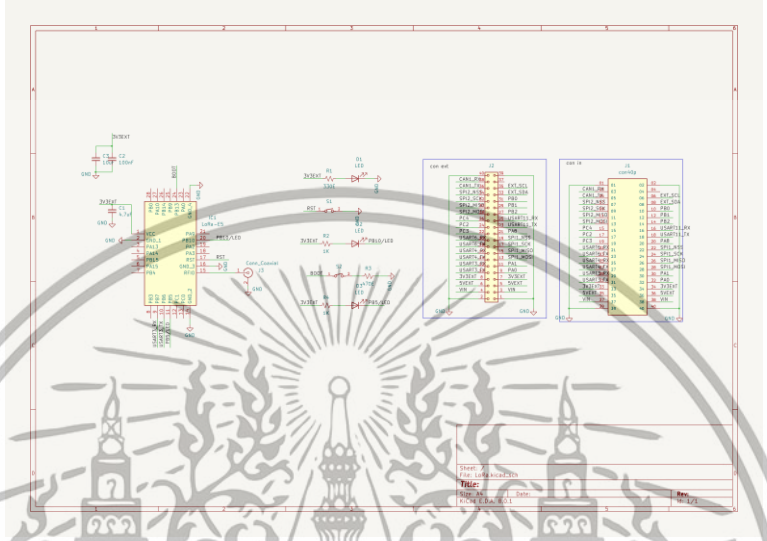


รูปที่ 3.1 การออกแบบ Schematic ของ Core Board

3.1.1.2 Local Network (LoRa Board)

Schematic การทำงานภายในฝั่ง Local Network (LoRa) การเชื่อมต่อเชื่อมต่อเข้ากับ Core Board แบบ Plug-in และจะถูกสั่งการทำงานผ่าน Core Board โดยจะต้องคำนึงถึงความเหมาะสมในการใช้งาน ข้อจำกัด รวมถึงความเข้ากันได้ของอุปกรณ์ เพื่อให้การสื่อสารในแต่ละอุปกรณ์ให้สื่อสารกันได้เพื่อการทำงานที่มีประสิทธิภาพสูงสุด โดยในการออกแบบการเชื่อมต่อ

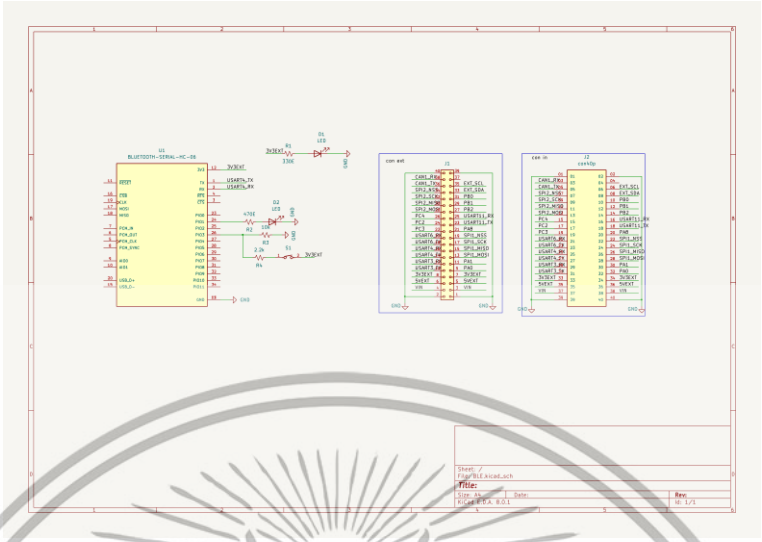
ขากับ LoRa Module ต้องศึกษาจากคู่มือการใช้งานและเลือกการติดต่อสื่อสารกับ Core Board โดยต้องพิจารณาจาก Alternate Functions โดยจะต้องดูจากความเข้ากันได้ของอุปกรณ์ที่ต้องการใช้ว่าใช้การสื่อสารแบบใด โดยในการรออกแบบการทำงานจะเป็น แสดงดังรูปที่ 3.2



รูปที่ 3.2 Schematic การทำงานภายในฝั่ง Local Network (LoRa)

3.1.1.3 Local Network (BLE Board)

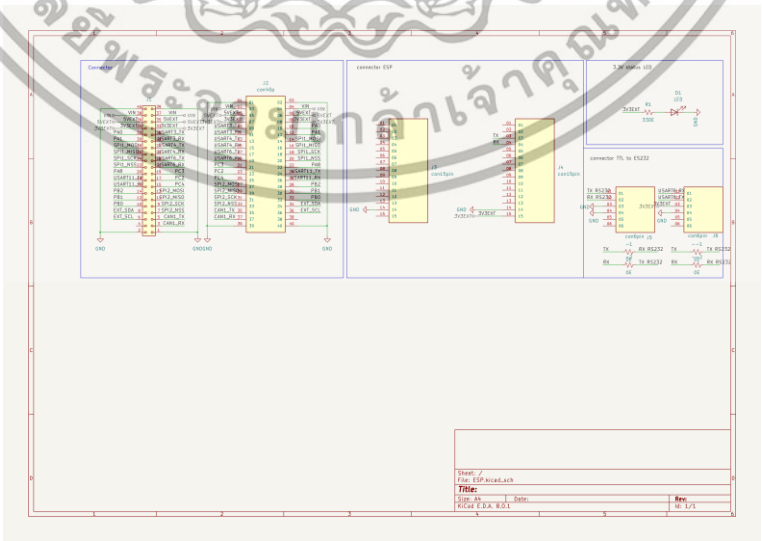
Schematic การทำงานภายในฝั่ง Local Network (BLE) การเชื่อมต่อเชื่อมต่อเข้ากับ Core Board แบบ Plug-in และจะถูกสั่งการทำงานผ่าน Core Board โดยจะต้องคำนึงถึงความเหมาะสมในการใช้งาน ข้อจำกัด รวมถึงความเข้ากันได้ของอุปกรณ์ เพื่อให้การสื่อสารในแต่ละอุปกรณ์ให้สื่อสารกันได้เพื่อการทำงานที่มีประสิทธิภาพสูงสุด โดยในการออกแบบการเชื่อมต่อขากับ BLE Module ต้องศึกษาจากคู่มือการใช้งานและเลือกการติดต่อสื่อสารกับ Core Board โดยต้องพิจารณาจาก Alternate Functions โดยจะต้องดูจากความเข้ากันได้ของอุปกรณ์ที่ต้องการใช้ว่าใช้การสื่อสารแบบใด โดยในการรออกแบบการทำงานจะเป็น แสดงดังรูปที่ 3.3



รูปที่ 3.3 Schematic การทำงานภายในฝั่ง Local Network (BLE)

3.1.1.4 การออกแบบ Public Network (ESP Board)

Schematic การทำงานภายในฝั่ง Public Network (ESP) การเชื่อมต่อเชื่อมต่อเข้ากับ Core Board แบบ Plug-in และจะถูกสั่งการทำงานผ่าน Core Board โดยจะต้องคำนึงถึงความเหมาะสมในการใช้งาน ข้อจำกัด รวมถึงความเข้ากันได้ของอุปกรณ์ เพื่อให้การสื่อสารในแต่ละอุปกรณ์ให้สื่อสารกันได้เพื่อการทำงานที่มีประสิทธิภาพสูงสุด โดยในการออกแบบการเชื่อมต่อขากับ ESP Module ต้องศึกษาจากคู่มือการใช้งานและเลือกการติดต่อสื่อสารกับ Core Board โดยต้องพิจารณาจาก Alternate Functions โดยจะต้องดูจากความเข้ากันได้ของอุปกรณ์ที่ต้องการใช้ว่าใช้การสื่อสารแบบใด โดยในการออกแบบการทำงานจะเป็น แสดงดังรูปที่ 3.4

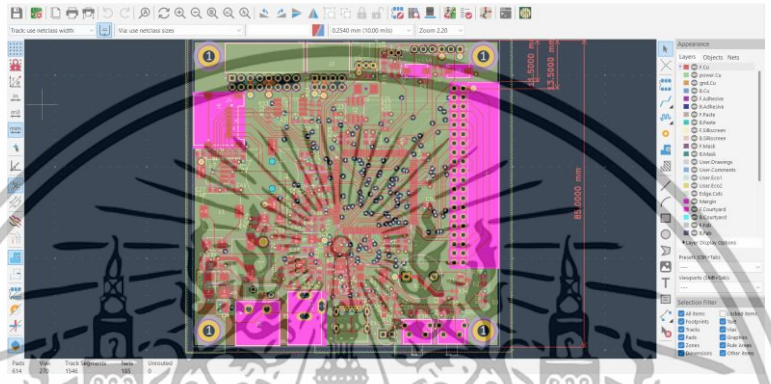


รูปที่ 3.4 Schematic การทำงานภายในฝั่ง Public Network (ESP)

3.1.2 การวางอุปกรณ์

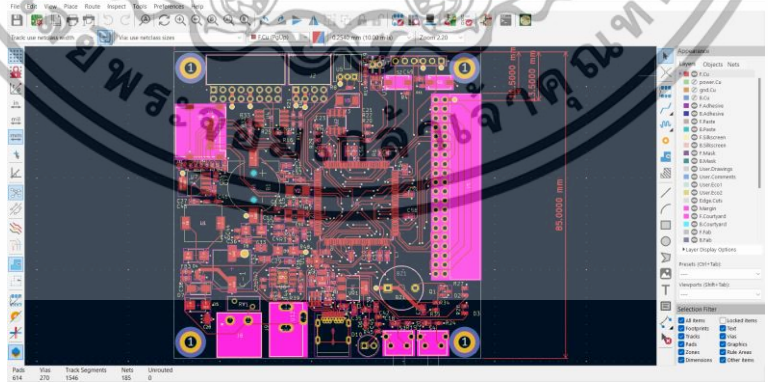
3.1.2.1 การวางอุปกรณ์ใน Core Board

ในการออกแบบ PCB ของ Core Board ซึ่งในการออกแบบได้มีการใช้โปรแกรม Kicad โดยในการออกแบบได้มีการออกแบบให้ PCB เป็น 4 Layer เพื่อสร้างความเสถียรและประสิทธิภาพสูงสุดของการทำงานในวงจร แสดงดังรูปที่ 3.5



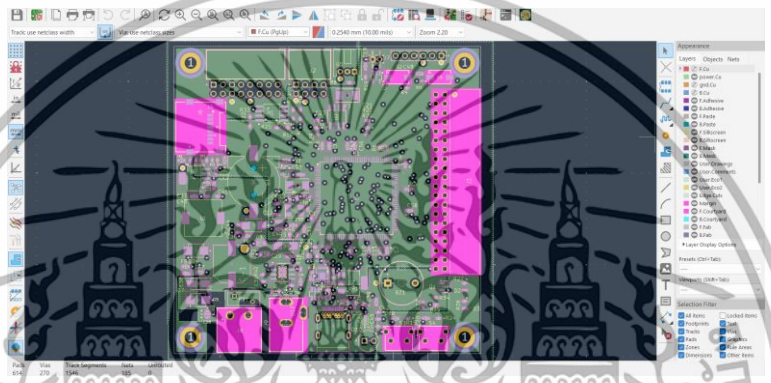
รูปที่ 3.5 การออกแบบ PCB ของ Core Board

Layer ที่ 1 (Top Layer) ใช้สำหรับการติดตั้งอุปกรณ์ Surface Mount ไมโครคอลโทรลเลอร์ STM32H563Zi และเชื่อมต่อสัญญาณหลัก เพื่อความง่ายในการบัดกรี โดยในการวางอุปกรณ์ในแต่ละอุปกรณ์ต้องคำนึงถึงขนาดของอุปกรณ์ที่ใช้ซึ่งจะต้องอ้างอิงจากคู่มือการใช้งาน จากนั้นต้องดำเนินการหา Footprint และ Library ที่ตรงตามคู่มือการใช้งานและตรงตามอุปกรณ์ตามที่ต้องการ แสดงดังรูปที่ 3.6



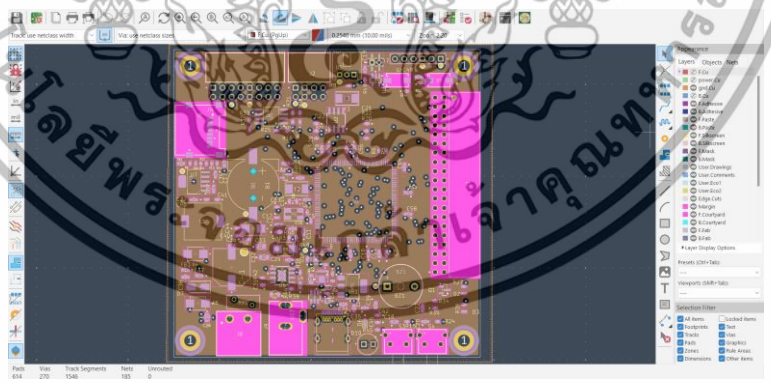
รูปที่ 3.6 ภาพแสดง Layer ที่ 1 ของ Core Board

Layer ที่ 2 (Power Plane) เชื่อมต่อแหล่งจ่ายไฟให้กับอุปกรณ์และวงจร เพื่อรักษาแรงดันไฟฟ้าและลดสัญญาณรบกวน โดยในการออกแบบในชั้นที่มีแรงดันไฟฟ้าที่จ่ายแรงดันไฟฟ้าให้แก่อุปกรณ์คือ 3.3 V และ 5 V โดยจะต้องมีการแยกพื้นที่ของแรงดันที่ป้อนให้แก่อุปกรณ์ให้ตรงกับพื้นที่ที่ป้อนแรงดันไฟฟ้า โดยในการวางอุปกรณ์ควรที่จะวางอุปกรณ์ที่ต้องการแรงดันไฟฟ้าเดียวกันในพื้นที่ที่ใกล้เคียงกัน เพื่อความเสถียรในการจ่ายแรงดันไฟฟ้าในวงจร จากรูปจะเห็นได้ว่ามีการแบ่งพื้นที่ของการป้อนแรงดันไฟฟ้าเป็น 2 ส่วน ได้แก่ พื้นที่ที่ป้อนแรงดันไฟฟ้า 5 V และ พื้นที่ที่ป้อนแรงดันไฟฟ้า 3.3.V แสดงดังรูปที่ 3.7



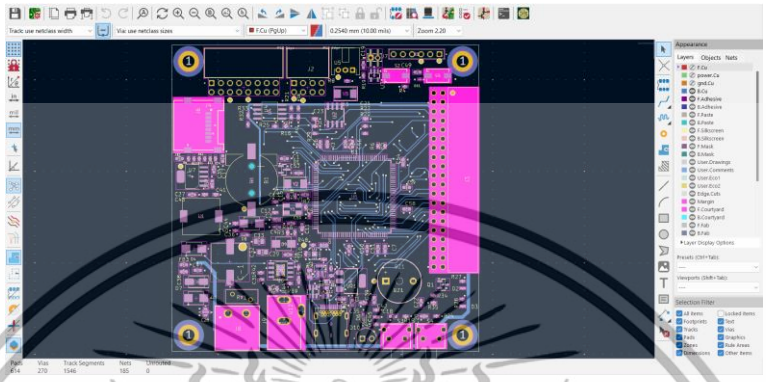
รูปที่ 3.7 ภาพแสดง Layer ที่ 2 ของ Core Board

Layer ที่ 3 (Ground Plane) เพื่อเสริมความเสถียรของสัญญาณความเร็วสูง โดยในวงจรจะต้องมีการใช้ Ground เดียวกันทั้งวงจร แสดงดังรูปที่ 3.8



รูปที่ 3.8 ภาพแสดง Layer ที่ 3 ของ Core Board

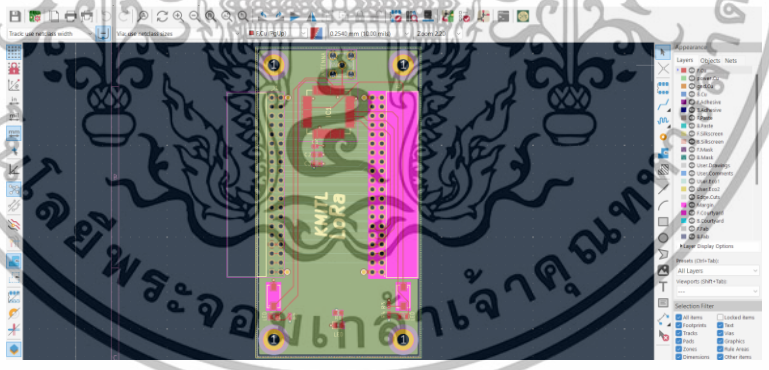
Layer ที่ 4 (Bottom Layer) ใช้สำหรับเชื่อมต่อสัญญาณเพิ่มเติมจาก Layer ที่ 1 (Top Layer) ในการเชื่อมต่อสัญญาณระหว่าง Layer ต้องมีการทำให้เส้นสัญญาณเชื่อมต่อกันระหว่าง Layer แสดงดังรูปที่ 3.9



รูปที่ 3.9 ภาพแสดง Layer ที่ 4 ของ Core Board

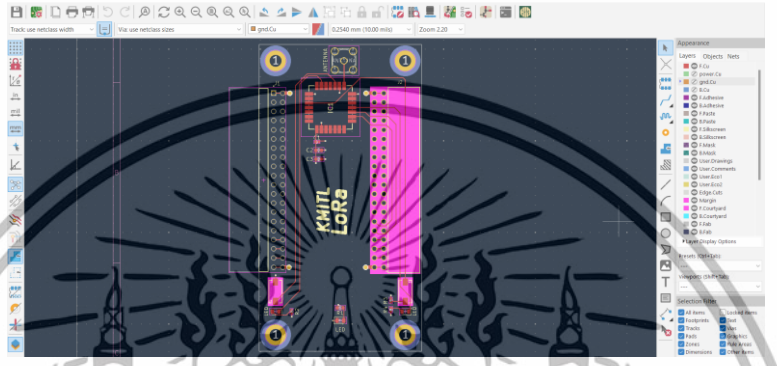
3.1.2.2 การวางอุปกรณ์ใน Local Network (LoRa Board)

ในการออกแบบ PCB ของ Local Network (LoRa) ในการออกแบบได้มีการใช้โปรแกรม Kicad โดยในการออกแบบได้มีการออกแบบเป็น PCB เป็น 4 Layer เพื่อสร้างความเสถียรและเกิดประสิทธิภาพสูงสุดของการทำงานในวงจร แสดงดังรูปที่ 3.10



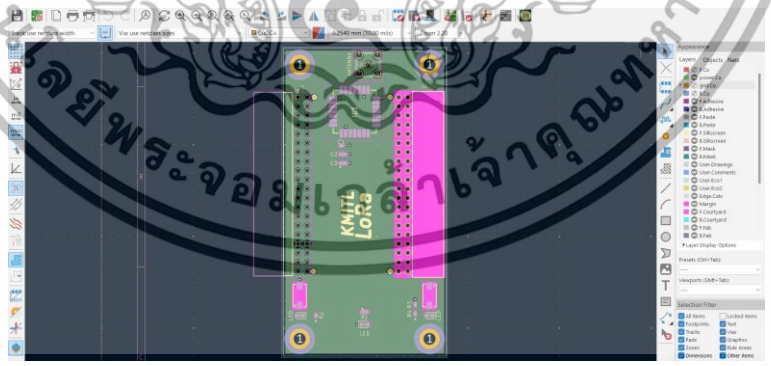
รูปที่ 3.10 การออกแบบ PCB ของ Local Network (LoRa)

Layer ที่ 1 (Top Layer) ใช้สำหรับการติดตั้งอุปกรณ์ Surface Mount ของ LoRa Module และเชื่อมต่อสัญญาณหลัก เพื่อความง่ายในการบัดกรี โดยในการวางอุปกรณ์ในแต่ละอุปกรณ์ต้องคำนึงถึงขนาดของอุปกรณ์ที่ใช้ ซึ่งจะต้องอ้างอิงจากคู่มือการใช้งาน จากนั้นต้องดำเนินการหา Footprint และ ไลบรารี ที่ตรงตามคู่มือการใช้งานและตรงตามอุปกรณ์ที่ต้องการ แสดงดังรูปที่ 3.11



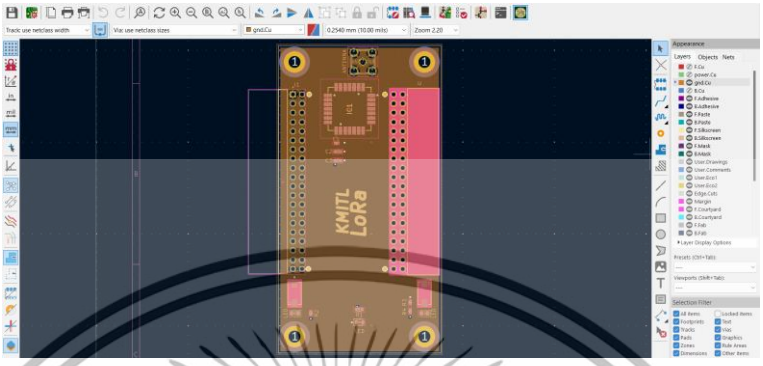
รูปที่ 3.11 ภาพแสดง Layer ที่ 1 ของ Local Network (LoRa)

Layer ที่ 2 (Power Plane) เชื่อมต่อแหล่งจ่ายไฟให้กับอุปกรณ์และวงจร เพื่อรักษาแรงดันไฟฟ้าและลดสัญญาณรบกวน โดยในการออกแบบในชั้นที่มีแรงดันไฟฟ้าที่จ่ายแรงดันไฟฟ้าให้แก่อุปกรณ์คือ 3.3 V External Regulator ที่เชื่อมต่อมาจาก Core Board โดยจะต้องแยกพื้นที่ที่จะป้อนแรงดันไฟฟ้าให้กับวงจร เพื่อความเสถียรในการทำงาน โดยจะเชื่อมต่อ Layer ที่ 2 เข้ากับ Layer ที่ 1 เข้ากับบริเวณที่ต้องการจ่ายแรงดันไฟฟ้าให้กับจุดนั้นๆ แสดงดังรูปที่ 3.12



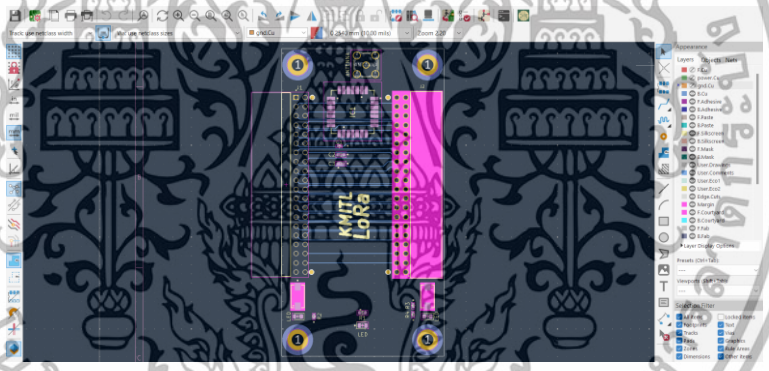
รูปที่ 3.12 ภาพแสดง Layer ที่ 2 ของ Local Network (LoRa)

Layer ที่ 3 (Ground Plane) เพื่อเสริมความเสถียรของสัญญาณความเร็วสูง โดยในวงจรจะต้องมีการใช้ Ground เดียวกันทั้งวงจร แสดงดังรูปที่ 3.13



รูปที่ 3.13 ภาพแสดง Layer ที่ 3 ของ Local Network (LoRa)

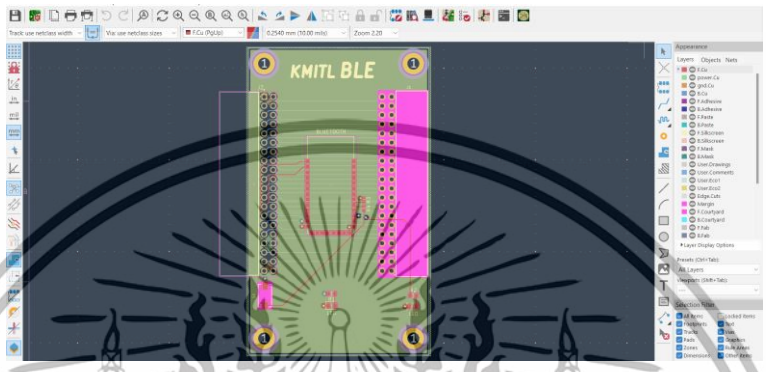
Layer ที่ 4 (Bottom Layer) ใช้สำหรับเชื่อมต่อสัญญาณจาก Connectors ที่เชื่อมต่อมาจาก Core Board และเชื่อมต่อไปยังบอร์ดถัดไป แสดงดังรูปที่ 3.14



รูปที่ 3.14 ภาพแสดง Layer ที่ 4 ของ Local Network (LoRa)

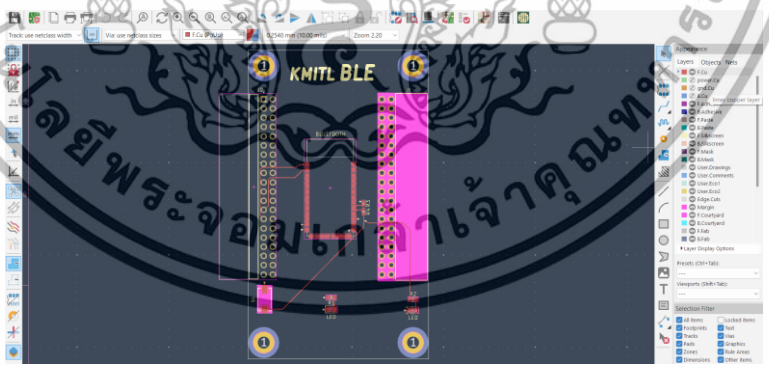
3.1.2.3 การวางอุปกรณ์ใน Local Network (BLE Board)

ในการออกแบบ PCB ของ Local Network (LoRa) ในการออกแบบได้มีการใช้โปรแกรม Kicad โดยในการออกแบบได้มีการออกแบบเป็น PCB เป็น 4 Layer เพื่อสร้างความเสถียรและเกิดประสิทธิภาพสูงสุดของการทำงานในวงจร แสดงดังรูปที่ 3.15



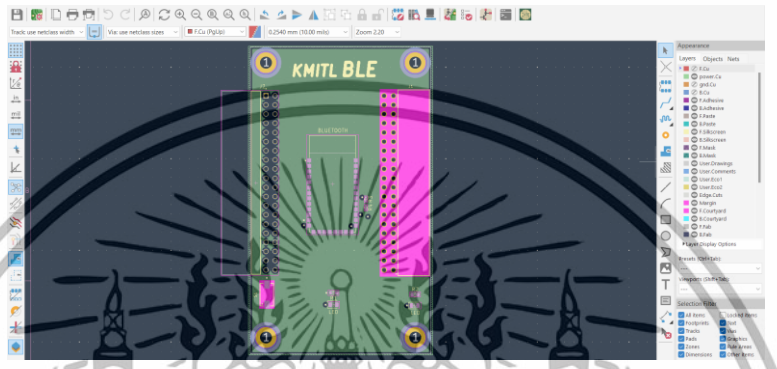
รูปที่ 3.15 การออกแบบ PCB ของ Local Network (BLE)

Layer ที่ 1 (Top Layer) ใช้สำหรับการติดตั้งอุปกรณ์ Surface Mount ของ HC- 06 Bluetooth Module และเชื่อมต่อสัญญาณหลัก เพื่อความง่ายในการบัดกรี โดยในการวางอุปกรณ์ในแต่ละอุปกรณ์ต้องคำนึงถึงขนาดของอุปกรณ์ที่ใช้ ซึ่งจะต้องอ้างอิงจากคู่มือการใช้งาน จากนั้นต้องดำเนินการหา Footprint และ Library ที่ตรงตามคู่มือการใช้งานและตรงตามอุปกรณ์ที่ต้องการ แสดงดังรูปที่ 3.16



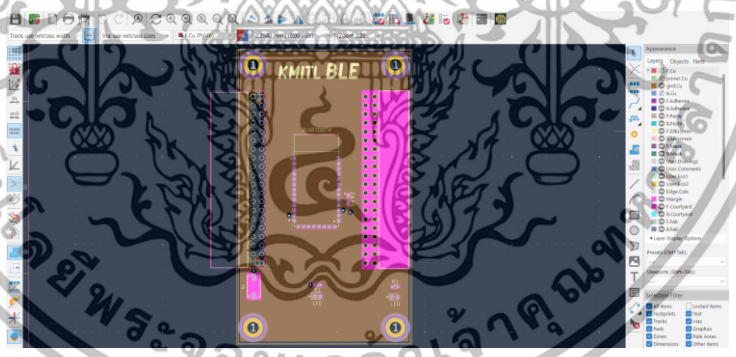
รูปที่ 3.16 ภาพแสดง Layer ที่ 1 ของ Local Network (BLE)

Layer ที่ 2 (Power Plane) เชื่อมต่อแหล่งจ่ายไฟให้กับอุปกรณ์และวงจร เพื่อรักษาแรงดันไฟฟ้าและลดสัญญาณรบกวน โดยในการออกแบบในชั้นที่มีแรงดันไฟฟ้าที่จ่ายแรงดันไฟฟ้าให้แก่อุปกรณ์คือ 3.3 V External Regulator ที่เชื่อมต่อมาจาก Core Board โดยจะต้องแยกพื้นที่ที่จะป้อนแรงดันไฟฟ้าให้กับวงจร เพื่อความเสถียรในการทำงาน โดยจะเชื่อมต่อ Layer ที่ 2 เข้ากับ Layer ที่ 1 เข้ากับบริเวณที่ต้องการจ่ายแรงดันไฟฟ้าให้กับจุดนั้นๆ แสดงดังรูปที่ 3.17



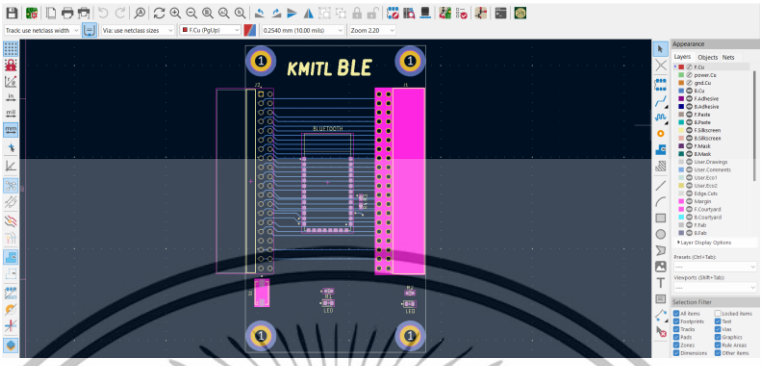
รูปที่ 3.17 ภาพแสดง Layer ที่ 2 ของ Local Network (BLE)

Layer ที่ 3 (Ground Plane) เพื่อเสริมความเสถียรของสัญญาณความเร็วสูง โดยในวงจรจะต้องมีการใช้ Ground เดียวกันทั้งวงจร แสดงดังรูปที่ 3.18



รูปที่ 3.18 ภาพแสดง Layer ที่ 3 ของ Local Network (BLE)

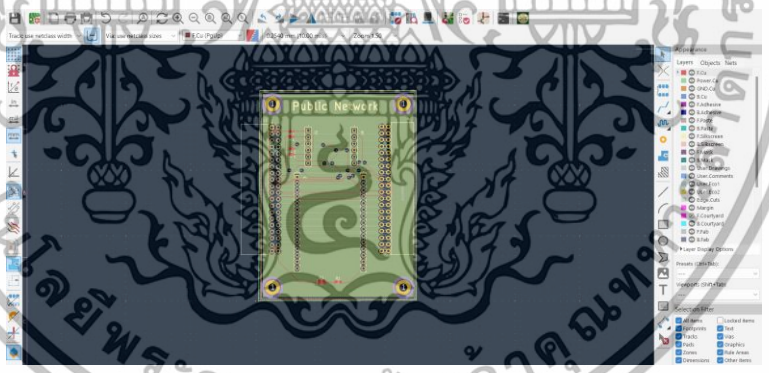
Layer ที่ 4 (Bottom Layer) ใช้สำหรับเชื่อมต่อสัญญาณจาก Connectors ที่เชื่อมต่อมาจาก Core Board และเชื่อมต่อไปยังบอร์ดถัดไป แสดงดังรูปที่ 3.19



รูปที่ 3.19 ภาพแสดง Layer ที่ 4 ของ Local Network (BLE)

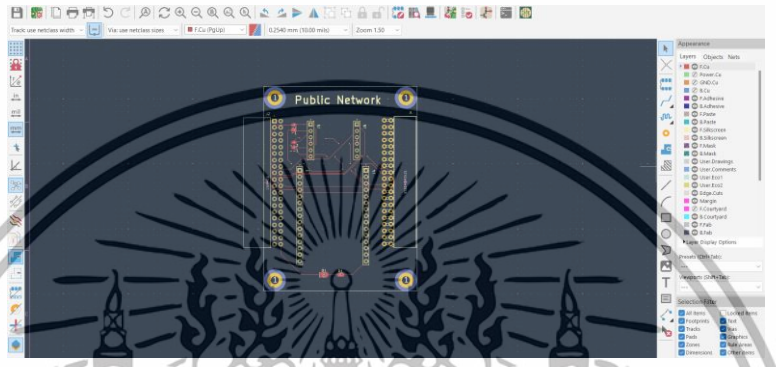
3.1.2.4 การวางอุปกรณ์ใน Public Network (ESP Board)

ในการออกแบบ PCB ของ Public Network (ESP Board) ในการออกแบบได้มีการใช้โปรแกรม Kicad โดยในการออกแบบได้มีการออกแบบเป็น PCB เป็น 4 Layer เพื่อสร้างความเสถียรและเกิดประสิทธิภาพสูงสุดของการทำงานในวงจร ดังรูปที่ 3.20



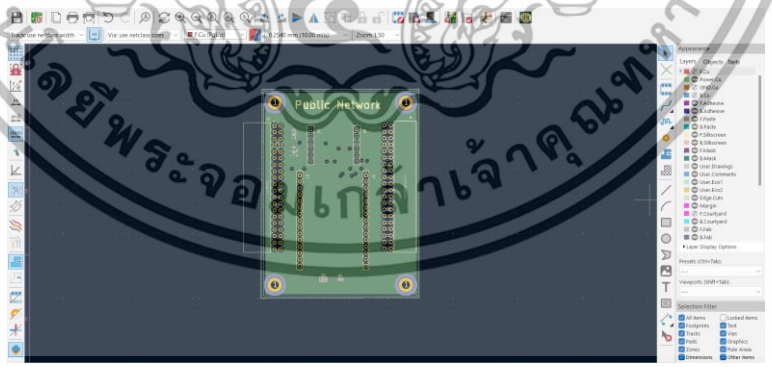
รูปที่ 3.20 การออกแบบ PCB ของ Public Network (ESP Board)

Layer ที่ 1 (Top Layer) ใช้สำหรับการติดตั้งอุปกรณ์ Surface Mount ของ ESP Module และเชื่อมต่อสัญญาณหลัก เพื่อความง่ายในการบัดกรี โดยในการวางอุปกรณ์ในแต่ละอุปกรณ์ต้องคำนึงถึงขนาดของอุปกรณ์ที่ใช้ ซึ่งจะต้องอ้างอิงจากคู่มือการใช้งาน จากนั้นต้องดำเนินการหา Footprint และ ไลบรารี ที่ตรงตามคู่เมื่อการใช้งานและตรงตามอุปกรณ์ที่ต้องการ ดังรูปที่ 3.21



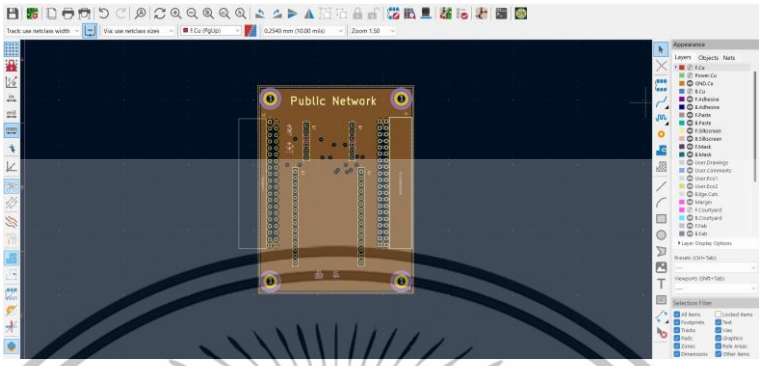
รูปที่ 3.21 ภาพแสดง Layer ที่ 1 ของ Public Network (ESP Board)

Layer ที่ 2 (Power Plane) เชื่อมต่อแหล่งจ่ายไฟให้กับอุปกรณ์และวงจร เพื่อรักษาแรงดันไฟฟ้าและลดสัญญาณรบกวน โดยในการออกแบบในชั้นที่มีแรงดันไฟฟ้าที่จ่ายแรงดันไฟฟ้าให้แก่อุปกรณ์คือ 3.3 V External Regulator ที่เชื่อมต่อมาจาก Core Board โดยจะต้องแยกพื้นที่ที่จะป้อนแรงดันไฟฟ้าให้กับวงจร เพื่อความเสถียรในการทำงาน โดยจะเชื่อมต่อ Layer ที่ 2 เข้ากับ Layer ที่ 1 เข้ากับบริเวณที่ต้องการจ่ายแรงดันไฟฟ้าให้กับจุดนั้นๆ ดังรูปที่ 3.22



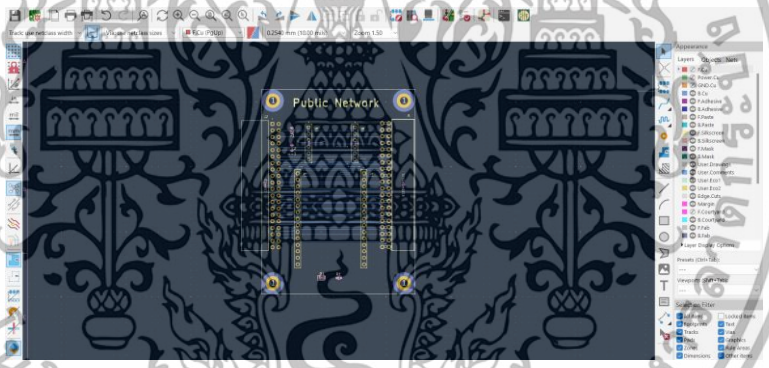
รูปที่ 3.22 ภาพแสดง Layer ที่ 2 ของ Public Network (ESP Board)

Layer ที่ 3 (Ground Plane) เพื่อเสริมความเสถียรของสัญญาณความเร็วสูง โดยในวงจรจะต้องมีการใช้ Ground เดียวกันทั้งวงจร ดังรูปที่ 3.23



รูปที่ 3.23 ภาพแสดง Layer ที่ 3 ของ Public Network (ESP Board)

Layer ที่ 4 (Bottom Layer) ใช้สำหรับเชื่อมต่อสัญญาณจาก Connectors ที่เชื่อมต่อมาจาก Core Board และเชื่อมต่อไปยังบอร์ดถัดไป ดังรูปที่ 3.24



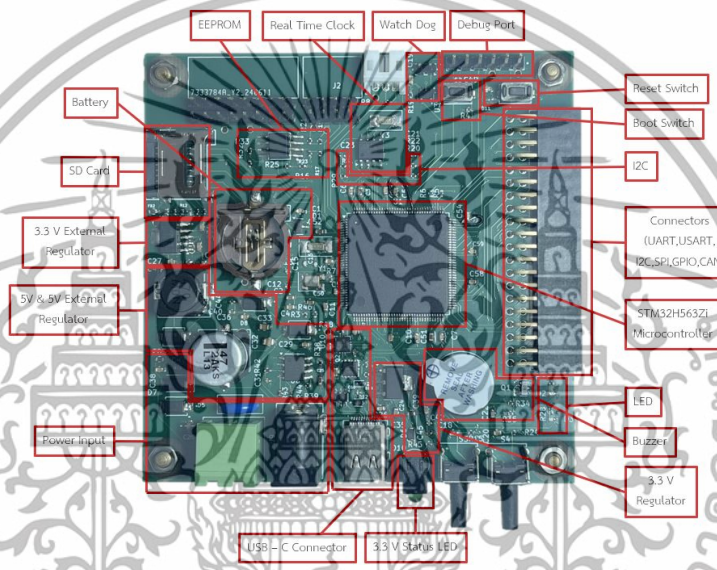
รูปที่ 3.24 ภาพแสดง Layer ที่ 4 ของ Public Network (ESP Board)

3.1.3 การติดตั้งอุปกรณ์

3.1.3.1 การติดตั้งอุปกรณ์ บน Core Board

ในการติดตั้งอุปกรณ์ Core Board ของระบบ IoT นี้ประกอบด้วยองค์ประกอบหลักที่สำคัญหลายอย่างเพื่อให้มั่นใจถึงประสิทธิภาพและความเสถียรในการทำงาน เริ่มจาก Power Input ที่รับแรงดันไฟฟ้า 12 V และแปลงแรงดันเป็น 5V Regulator, 5V External Regulator, 3.3V Regulator, 3.3 V External Regulator เพื่อจ่ายไฟให้กับชิพและอุปกรณ์อื่นๆ บนบอร์ด รวมถึงบอร์ด Plug-in ที่ต่อเข้ามาเพิ่มเติม การทำงานของบอร์ดถูกตรวจสอบผ่าน LED Status 3.3 V ที่แสดงสถานการณ์ทำงานและมี LED การแจ้งเตือนต่างๆ Real Time Clock (RTC) พร้อมแบตเตอรี่สำรองช่วยให้สามารถบันทึกเวลาและวันที่ต่อเนื่องแม้เมื่อบอร์ดถูกปิด นอกจากนี้ยังมี SD Card สำหรับเพิ่มพื้นที่จัดเก็บข้อมูลและ USB-C Connector สำหรับเชื่อมต่ออุปกรณ์ภายนอก เช่น

การอัปโหลดเฟิร์มแวร์ การ Debug การตรวจสอบและแก้ไขปัญหาผ่าน Debug Port ทำได้สะดวก ในกรณีที่ระบบค้างวงจร Watchdog จะทำการรีเซ็ตบอร์ดอัตโนมัติ Reset Switch และ Boot Switch ใช้ในการรีสตาร์ทระบบและเลือกโหมดการ Boot ตามต้องการ โดย Buzzer สามารถแจ้งเตือนตามที่ตั้งโปรแกรมไว้ สำหรับการเชื่อมต่อกับบอร์ด Plug-in และอุปกรณ์อื่นๆ Core Board มี Connectors ที่รองรับโปรโตคอลการสื่อสารหลากหลาย เช่น UART, USART, I2C, SPI, GPIO, และ CAN ซึ่งทั้งหมดนี้ช่วยให้ Core Board สามารถรองรับการขยายตัวและใช้งานในสภาพแวดล้อม IoT ได้อย่างมีประสิทธิภาพและยืดหยุ่น แสดงดังรูปที่ 3.25

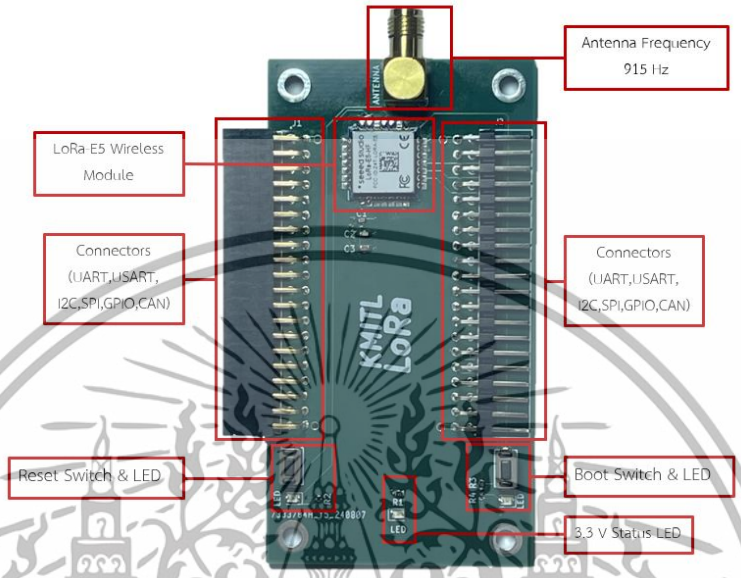


รูปที่ 3.25 ภาพแสดงส่วนประกอบต่าง ๆ ใน Core Board

3.1.3.2 การติดตั้งอุปกรณ์ บน Local Network (LoRa Board)

ในการติดตั้งอุปกรณ์การทำงานภายในฝั่ง Local Network (LoRa) นั้นได้ประกอบด้วยส่วนประกอบหลักที่สำคัญหลายอย่าง เริ่มจากการใช้ Connectors ที่รองรับการสื่อสารผ่านโปรโตคอลที่หลากหลาย ไม่ว่าจะเป็น UART, USART, I2C, SPI, GPIO, และ CAN ซึ่งทั้งหมดนี้มีบทบาทสำคัญในการสื่อสารกับ Core Board และสามารถเชื่อมต่อไปยังบอร์ด Plug-in อื่นๆ ได้อย่างมีประสิทธิภาพ ระบบนี้รับแรงดันไฟฟ้าจาก 3.3 V External Regulator ที่ถูกเชื่อมต่อจาก Core Board เพื่อจ่ายไฟฟ้าให้กับชิพและอุปกรณ์อื่นๆ บนบอร์ด นอกจากนี้ยังมีไฟแสดงสถานะที่ช่วยบ่งชี้การทำงานของบอร์ดเมื่อมีแรงดันไฟฟ้าจ่ายเข้าสู่ระบบ Reset Switch และ Boot Switch ถูกออกแบบมาเพื่อให้ผู้ใช้งานสามารถรีสตาร์ทระบบหรือเลือกโหมดการ Boot ได้ตามความต้องการ อีกทั้งยังมีการใช้ Module LoRa-E5 สำหรับการสื่อสารแบบไร้สาย ซึ่งทำงานร่วมกับสายอากาศที่รองรับการส่งข้อมูลในความถี่ 915 MHz ของ LoRa ทำให้ระบบนี้เหมาะสมอย่างยิ่งสำหรับการพัฒนาระบบ

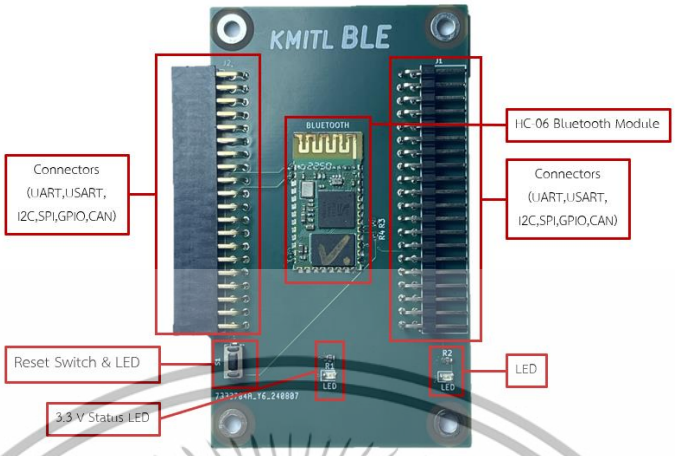
IoT ที่มีประสิทธิภาพสูงและสามารถทำงานได้อย่างราบรื่นในหลากหลายสถานการณ์ แสดงดังรูปที่ 3.26



รูปที่ 3.26 ภาพแสดงส่วนประกอบต่าง ๆ ใน Local Network (LoRa)

3.1.3.3 การติดตั้งอุปกรณ์ บน Local Network (BLE Board)

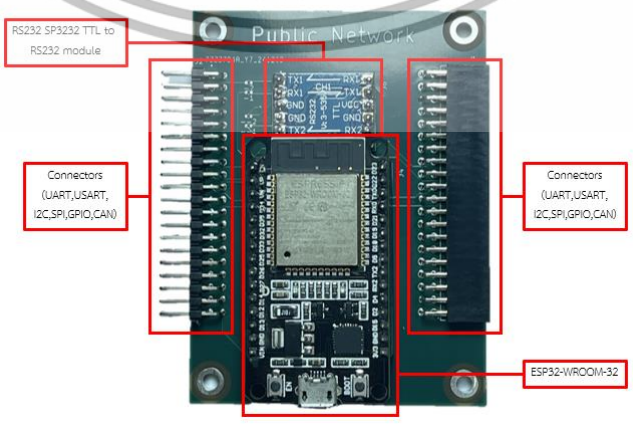
ในการติดตั้งอุปกรณ์การทำงานภายในฝั่ง Local Network (BLE) นั้นได้ประกอบด้วยส่วนประกอบหลักที่สำคัญหลายอย่าง เริ่มจากการใช้ Connectors ที่รองรับการสื่อสารผ่านโปรโตคอลที่หลากหลาย ไม่ว่าจะเป็น UART, USART, I2C, SPI, GPIO, และ CAN ซึ่งทั้งหมดนี้มีบทบาทสำคัญในการสื่อสารกับ Core Board และสามารถเชื่อมต่อไปยังบอร์ด Plug-in อื่นๆ ได้อย่างมีประสิทธิภาพ ระบบนี้รับแรงดันไฟฟ้าจาก 3.3 V External Regulator ที่ถูกเชื่อมต่อจาก Core Board เพื่อจ่ายไฟฟ้าให้กับชิพและอุปกรณ์อื่นๆ บนบอร์ด นอกจากนี้ยังมีไฟแสดงสถานะที่ช่วยบ่งชี้การทำงานของบอร์ดเมื่อมีแรงดันไฟฟ้าจ่ายเข้าสู่ระบบ Reset Switch ถูกออกแบบมาเพื่อให้ผู้ใช้งานสามารถรีสตาร์ทระบบหรือเลือกโหมดการ Boot ได้ตามความต้องการ อีกทั้งยังมีการใช้ HC- 06 Bluetooth Module ที่รองรับการทำงานบนโปรโตคอลการสื่อสาร Bluetooth 2.0 ทำให้ระบบนี้เหมาะสมอย่างยิ่งสำหรับการพัฒนาระบบ IoT ที่มีประสิทธิภาพสูงและสามารถทำงานได้อย่างราบรื่นในหลากหลายสถานการณ์ แสดงดังรูปที่ 3.27



รูปที่ 3.27 ภาพแสดงส่วนประกอบต่าง ๆ ใน Local Network (BLE)

3.1.3.4 การติดตั้งอุปกรณ์ บน Public Network (ESP Board)

ในการออกแบบการทำงานภายในฝั่ง Public Network (ESP) นั้นได้ประกอบด้วยส่วนประกอบหลักที่สำคัญหลายอย่าง เริ่มจากการใช้ Connectors ที่รองรับการสื่อสารผ่านโปรโตคอลที่หลากหลาย ไม่ว่าจะเป็น UART, USART, I2C, SPI, GPIO, และ CAN ซึ่งทั้งหมดนี้มีบทบาทสำคัญในการสื่อสารกับ Core Board และสามารถเชื่อมต่อไปยังบอร์ด Plug-in อื่นๆ ได้อย่างมีประสิทธิภาพ ระบบนี้รับแรงดันไฟฟ้าจาก 3.3 V External Regulator ที่ถูกเชื่อมต่อจาก Core Board เพื่อจ่ายไฟฟ้าให้กับชิพและอุปกรณ์อื่นๆ บนบอร์ด นอกจากนี้ยังมีไฟแสดงสถานะที่ช่วยบ่งชี้การทำงานของบอร์ดเมื่อมีแรงดันไฟฟ้าจ่ายเข้าสู่ระบบ อีกทั้งยังมีการใช้ ESP Module สำหรับการสื่อสารกับเครือข่ายสาธารณะ ซึ่งทำงานที่มีทั้ง Bluetooth และ Wi-Fi ในตัวเอง ทำให้ระบบนี้เหมาะสมอย่างยิ่งสำหรับการพัฒนาระบบ IoT ที่มีประสิทธิภาพสูงและสามารถทำงานได้อย่างราบรื่นในหลากหลายสถานการณ์ ดังรูปที่ 3.28



รูปที่ 3.28 ภาพแสดงส่วนประกอบต่างๆใน Public Network (ESP Board)

3.1.4 การเตรียมสภาพแวดล้อมสำหรับการพัฒนาเฟิร์มแวร์

สภาพแวดล้อม (environment) ที่ใช้บน Zephyr os เพื่อพัฒนาเฟิร์มแวร์จะมี CMake เวอร์ชั่น3.20.5 ,Python เวอร์ชั่น 3.10 , Devicetree compiler เวอร์ชั่น 1.4.6 โดยการติดตั้งจะทำผ่าน Chocolatey เป็นเครื่องมือสำหรับการจัดการแพ็กเกจบน Windows ที่ช่วยให้การติดตั้งโปรแกรมหรือ dependencies ที่จำเป็นสำหรับการพัฒนาโปรเจกต์เป็นเรื่องง่ายและสะดวก โดยทำหน้าที่คล้ายกับ apt บน Ubuntu หรือ Homebrew บน macOS

3.1.4.1 การติดตั้ง Library ที่จำเป็น

1.เปิดโปรแกรม Windows power shell โดยเปิดในโหมดของ Administrator

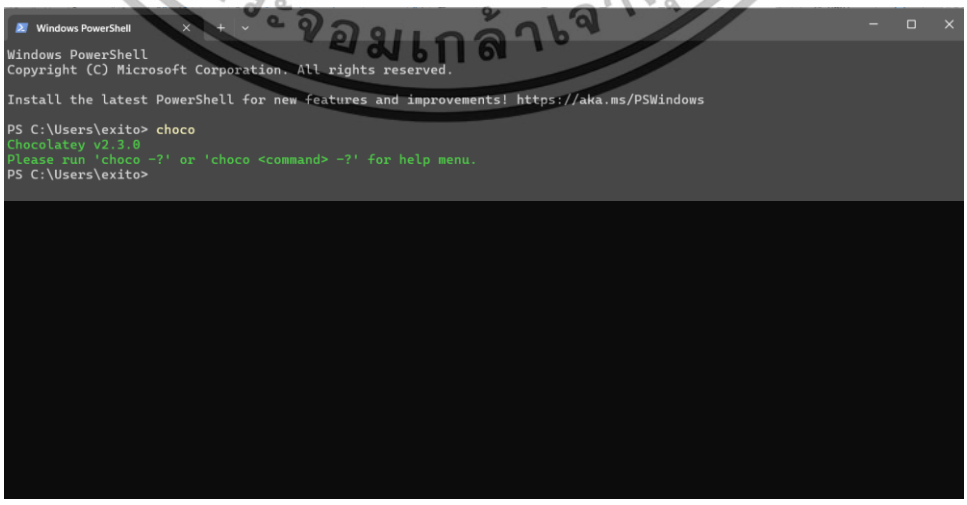
2.ตรวจสอบว่า Execution Policy ไม่ได้ถูกจำกัดการติดตั้งจาก Outsource โดยใช้คำสั่ง >>Get-ExecutionPolicy เมื่อพิมพ์คำสั่งแล้วได้รับคำตอบกลับว่า Restricted ให้ใช้คำสั่ง Set-ExecutionPolicy Bypass -Scope Process -Force เพื่อเปิดการติดตั้งจาก outsource และสามารถตรวจสอบได้อีกครั้งจากคำสั่ง Execution Policy เพื่อดูว่าเป็น AllSigned หรือไม่

3.ติดตั้ง Chocolatey จากคำสั่ง Set-ExecutionPolicy Bypass -Scope Process-Force;[System.Net.WebClient]::DownloadString('https://community.chocolatey.org/install.ps1') | iex

3.ตรวจสอบการติดตั้ง choco จากคำสั่ง choco แสดงดังรูปที่ 3.29

4.ติดตั้งสภาพแวดล้อมจากคำสั่ง

- chocoinstallcmake--installargs'ADD_CMAKE_TO_PATH=System'
- choco install ninja gperf python311 git dtc-msys2 wget7zip



รูปที่ 3.29 การตรวจสอบการติดตั้ง chocolatay

3.1.4.2 การติดตั้ง Zephyr

การติดตั้ง Zephyr จะทำผ่าน West tools ซึ่งเป็นเครื่องมือบรรทัดคำสั่ง (command-line tool) ที่ถูกพัฒนาขึ้นมาโดยเฉพาะเพื่อใช้กับ Zephyr RTOS ซึ่งเป็นระบบปฏิบัติการแบบเรียลไทม์ (Real-Time Operating System) ที่ได้รับความนิยมอย่างมากในการพัฒนาอุปกรณ์ IoT (Internet of Things) โดยมีขั้นตอนการติดตั้งดังต่อไปนี้

1.สร้าง Directory สำหรับติดตั้ง Zephyr

2.เปิด Terminal ในโหมดของ regular user เพื่อสร้าง Virtual environment ซึ่งหมายถึง การติดตั้ง dependencies ที่เฉพาะเจาะจงสำหรับแต่ละโปรเจกต์ ป้องกันไม่ให้เกิดปัญหาความขัดแย้งของเวอร์ชันของ libraries ระหว่างโปรเจกต์ต่าง ๆ โดยทุกอย่างจะถูกติดตั้งและจัดเก็บไว้ในโฟลเดอร์ของ virtual environment นั้น ๆ โดยไม่จำเป็นต้องเชื่อมโยงกับระบบหลัก (Add to system)

3.พิมพ์คำสั่ง zephyrproject\.venv\Scripts\activate.bat ช่วยให้มั่นใจว่า dependencies จะถูกติดตั้งและใช้งานเฉพาะในสภาพแวดล้อมของโปรเจกต์นั้น ๆ ไม่ปนกับระบบภายนอก

4.ทำการติดตั้ง west ด้วยคำสั่ง pip install west

5.กลับไปที่ Directory สำหรับติดตั้ง และพิมพ์คำสั่ง west init เพื่อรับ Zephyr Source code หลังจากติดตั้งเสร็จให้ทำการพิมพ์คำสั่ง west update เพื่ออัพเดทตัว Zephyr ดังกล่าว

6.พิมพ์คำสั่ง west zephyr-export เพื่อ Export Zephyr CMake package ซึ่งจะช่วยให้ CMake โหลดโค้ดที่จำเป็นสำหรับการสร้างโปรเจกต์ Zephyr ได้โดยอัตโนมัติ

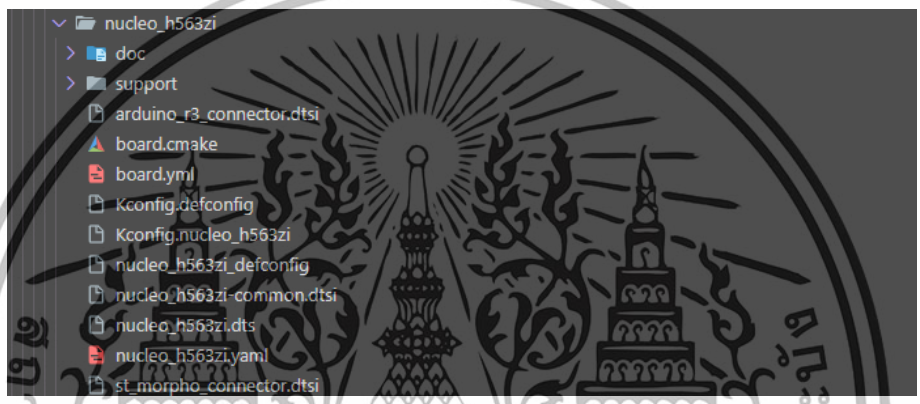
7.ติดตั้ง Python dependencies เพิ่มเติมที่ถูกประกาศไว้ในไฟล์ requirements.txt โดยใช้คำสั่ง pip install -r %HOMEPATH%\zephyrproject\zephyr\scripts\requirements.txt คำสั่งนี้จะติดตั้ง dependencies ที่ระบุในไฟล์ requirements.txt เพื่อให้โครงการทำงานได้สมบูรณ์

3.1.5 การโปรแกรมผ่านโปรโตคอลสื่อสารอย่าง UART

3.1.5.1 การตั้งค่าการสื่อสารของ UART ใน Devicetree

ภายในไฟล์การทำงานของ Zephyr จะมี Directory สำหรับไฟล์ Devicetree ของไมโครคอนโทรเลอร์ที่ใช้งานอยู่แล้วซึ่งสามารถเรียกใช้งานได้จากที่อยู่ zephyr/board/st/nucleo_h563zi ซึ่งจะพบไฟล์ดังกล่าวแสดงดังรูปที่ 3.30 โดยจะเข้าไปแก้ไขความเร็วการสื่อสารและพินการใช้งานได้ในไฟล์ nucleo_h563zi-common.dts และภายในไฟล์จะ

พบโค้ดแสดงดังรูปที่ 3.31 ซึ่งในไฟล์ดังกล่าวจะทำการเปิดโหนด usart3 ซึ่งภายในโครงสร้างของ Zephyr ได้สร้างโหนดการใช้งานอินเตอร์เฟซดังกล่าวเอาไว้แล้วโดยภายในโหนดจะกำหนดค่าที่อยู่ของ Peripheral ดังกล่าวไว้แสดงดังรูปที่ 3.32 ได้กำหนดที่อยู่คือ @40004800 และใช้ไดร์เวอร์ (compatible) ของ stm32-uart ซึ่งเป็นไดร์เวอร์ที่ทาง Zephyr รองรับการใช้งานไว้อยู่แล้ว แต่สถานะ (status) ยังปิดการใช้งานอยู่ ต้องทำการเปิดใช้งานหรือเปิดโหนดในไฟล์ nucleo_h563zi-common.dts ด้วยวิธีการอ้างอิงไปที่โหนดเดิมของ Zephyr ที่ได้สร้างไว้ด้วยสัญลักษณ์ & ซึ่งสามารถกำหนดความเร็วในการสื่อสาร(current-speed) และพิน(pinctrl-0) ที่ใช้ได้แสดงดังรูปที่ 3.33



รูปที่ 3.30 โครงสร้างไฟล์ Devicetree ที่ใช้สำหรับชิพ STM32H563ZI

```
/*
 * Copyright (c) 2021 Linaro Limited
 * Copyright (c) 2023 Thomas Stranger
 *
 * SPDX-License-Identifier: Apache-2.0
 */

#include <st/h5/stm32h563Xi.dtsi>
#include <st/h5/stm32h563zitx-pinctrl.dtsi>
#include "st_morpho_connector.dtsi"
#include <zephyr/dt-bindings/input/input-event-codes.h>

/ {
    leds: leds {
        compatible = "gpio-leds";
        led1_test: led_1 {
            gpios = <&gpioc 0 GPIO_ACTIVE_HIGH>;
            label = "User LD1";
        };
        led2_test: led_2 {
            gpios = <&gpioc 1 GPIO_ACTIVE_HIGH>;
            label = "User LD2";
        };
        buzzer_test: led_3 {
            gpios = <&gpioa 15 GPIO_ACTIVE_HIGH>;
            label = "User LD3";
        };
    };

    gpio_keys {
        compatible = "gpio-keys";
        user_button: button {
            label = "User";
            gpios = <&gpioc 13 GPIO_ACTIVE_HIGH>;
            zephyr,code = <INPUT_KEY_0>;
        };
    };
```

รูปที่ 3.31 โค้ดตัวอย่างของ Devicetree

```
usart3: serial@40004800 {
    compatible = "st,stm32-usart", "st,stm32-uart";
    reg = <0x40004800 0x400>;
    clocks = <&rcc STM32_CLOCK_BUS_APB1 0x00040000>;
    resets = <&rctl STM32_RESET(APB1L, 18U)>;
    interrupts = <60 0>;
    status = "disabled";
};
```

รูปที่ 3.32 โหนด usart3 และการตั้งค่าที่อยู่สัญญาณนาฬิกาที่ใช้

```
&usart3 { /*LoRa node*/
    pinctrl-0 = <&usart3_tx_pb10 &usart3_rx_pd9>;
    pinctrl-names = "default";
    current-speed = <9600>;
    status = "okay";
};
```

รูปที่ 3.33 การตั้งค่าการสื่อสารแบบ UART บน Devicetree

3.1.5.2 การโปรแกรมส่งข้อมูลแบบ UART

ภายในโฟลเดอร์หรือพื้นที่ทำงานสำหรับพัฒนาโปรแกรมจะต้องสร้างองค์ประกอบที่ต้องใช้อยู่ทั้งหมด 3 อย่าง ดังนี้

1.โฟลเดอร์ src สำหรับเก็บไฟล์โค้ดหลักการทำงาน

2.ไฟล์ CMakeLists.txt เพื่อสามารถคอมไพล์โค้ดเข้ากันกับ Zephyr ซึ่งภายในไฟล์สามารถเลือกไฟล์โค้ดที่จะอัปโหลดโปรแกรมได้ แสดงดังรูปที่ 3.34

3. ไฟล์ prj.conf เป็นไฟล์กำหนดค่าการคอมไพล์ของโปรเจกต์ โดยระบุค่าพารามิเตอร์หรือการตั้งค่าต่าง ๆ ที่ต้องการเปิดหรือปิด เช่น การเปิดใช้ไลบรารี โมดูล หรือฟีเจอร์เฉพาะของ RTOS นั้น เพื่อควบคุมการทำงานของเฟิร์มแวร์ตามความต้องการของโปรเจกต์ เมื่อสร้างองค์ประกอบครบ สามารถสร้างไฟล์ main.c เพื่อเป็นพื้นที่ทำงานสำหรับพัฒนาโปรแกรม โดยในไฟล์ดังกล่าวจะใช้มาโครที่ชื่อว่า #DT_NODELABEL() สำหรับดึง Parameter ที่ได้กำหนดค่าไว้ในไฟล์ devicetree-common.dts มาใช้ในโค้ด และเนื่องจาก Core board ไม่มีหน้า Serial terminal ทำให้ไม่สามารถแสดงผลหรือผลการอ่านข้อมูลที่รับเข้ามาได้ แก้ไขโดยการใช้มาโครที่ชื่อว่า LOG_MODULE_REGISTER() เพื่อแสดงผลข้อความหรือผลลัพธ์การทำงานผ่าน interface SWD Segger ได้ ซึ่งจะดูผลลัพธ์ผ่าน โปรแกรม J-Link RTT Viwer และการส่งข้อมูลแบบ UART ใช้ฟังก์ชั่นที่ชื่อว่า uart_poll_out() และใช้คำสั่ง uart_irq_callback_user_data_set() เพื่ออ่านข้อมูลที่เข้ามา

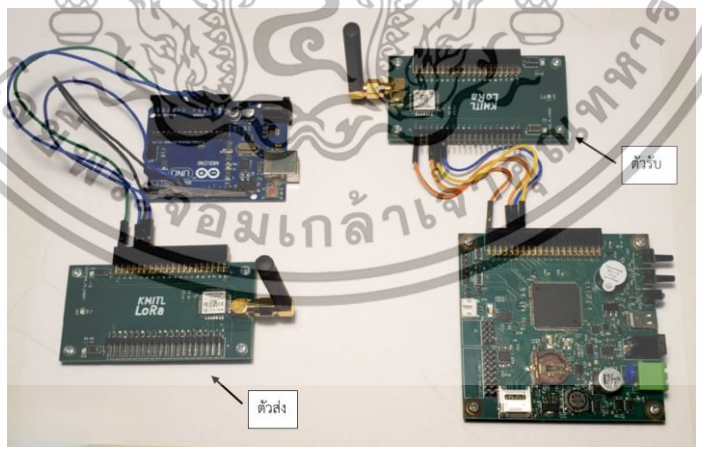
ในทันทีโดยจะทำการ interrupt ทันทีเมื่อมีข้อมูลวิ่งเข้ามา โดยฟังก์ชันที่สองเป็น build in function ที่มีให้ของ Zephyr อยู่แล้ว

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(firmware_iot_v1)

target_sources(app PRIVATE src/main.c)
```

รูปที่ 3.34 โค้ดตัวอย่างที่ใช้ในไฟล์ CMakeLists.txt

3.1.6 การออกแบบโปรแกรม รับส่งข้อมูลผ่านโปรโตคอล LoRa

โมดูล LoRa E5 HF เป็นอุปกรณ์ที่ถูกผลิตออกมาในลักษณะของ End node หรือ ทำหน้าที่เป็นตัวส่งข้อมูลมาที่ Gateway ค่าตั้งต้นจากโรงงานจะใช้งานในโหมด LWABP ซึ่งจะสุ่มความถี่การใช้งานใน Band ที่เลือกใช้ เช่น Band AU915 จะมีช่องสัญญาณทั้งหมด 72 ช่อง การส่งข้อมูล จะสุ่มความถี่การส่งเพื่อป้องกันการชนกันหรือเหตุการณ์ที่จะทำให้เกิดความเสียหายของข้อมูลได้ ซึ่งจะมีโหมดการทำงานอยู่ 3 รูปแบบคือ 1. TEST 2. LWOTAA 3. LWABP โดยในการออกแบบใช้ LoRa E5 HF สองโมดูลในการรับส่งข้อมูลซึ่งกันและกัน เพราะฉะนั้น จึงใช้โหมด TEST เพื่อทำให้โมดูลทั้ง 2 ตัวสื่อสารกันได้ ซึ่งจะทำการแบ่งการออกแบบเป็น 2 ฝั่งคือฝั่งรับ (Recever) และ ฝั่งส่ง (Transmit) ซึ่งมีภาพรวม แสดงดังรูปที่ 3.35



รูปที่ 3.35 การทดสอบการสื่อสารของโปรโตคอล LoRa

3.1.6.1 การตั้งค่า LoRa ฝั่งส่ง

เชื่อมต่อโมดูล LoRa เข้ากับบอร์ด Arduino uno ผ่านพินการสื่อสาร UART และทำการส่งคำสั่ง AT COMMAND เพื่อตั้งค่าอุปกรณ์ให้พร้อมสำหรับการส่งข้อมูล โดยมีขั้นตอนดังต่อไปนี้ โดยโมดูลจะมีการตอบกลับ แสดงดังรูปที่ 3.36

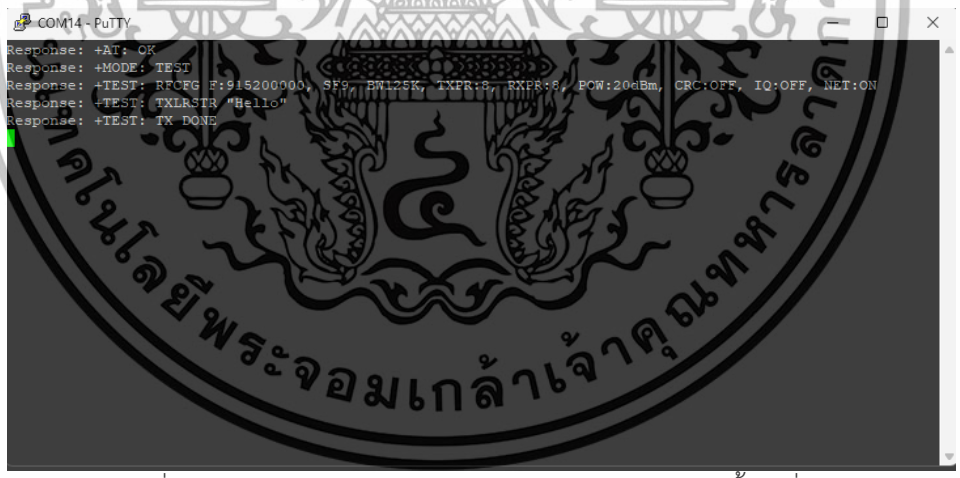
1.ส่งคำสั่ง AT เพื่อยืนยันการใช้งานว่าอุปกรณ์สามารถสื่อสารได้ หากได้รับข้อความตอบกลับว่า +AT: OK หมายความว่าอุปกรณ์สามารถใช้งานได้

2.ส่งคำสั่ง AT+MODE=TEST/n เพื่อตั้งค่าอุปกรณ์ให้เข้าสู่โหมด TEST โดยอุปกรณ์จะตอบกลับมาว่า +MODE: TEST

3.ส่งคำสั่ง AT+TEST=RFCFG,915.200,SF9,125,8,8,20,OFF,OFF,ON โดยคำสั่งดังกล่าวคือการตั้งค่า ให้อุปกรณ์ใช้ความถี่การสื่อสาร ,SF, Bandwidth, ตามที่ต้องการสามารถอธิบายได้ดังนี้

AT+TEST=RFCFG,<Frequency>,<SF>,<Bandwidth>,<tx_power>,<rx_power>,<กำลังสัญญาณ>,<การเปิดใช้งาน CRC>,<การตั้งค่า In-phase Quadrature >,<การตั้งค่าเครือข่าย>

4.ส่งคำสั่ง AT+TEST=TXLRSTR,<message> เพื่อส่งข้อมูล หากส่งข้อมูลสำเร็จโมดูลจะตอบกลับว่า +TEST: TX DONE



รูปที่ 3.36 การตอบกลับของโมดูล LoRa E5 HF ในการตั้งค่าฝั่งส่ง

3.1.6.2 การตั้งค่า LoRa ฝั่งรับ

เชื่อมต่อโมดูล LoRa เข้ากับบอร์ด Arduino uno ผ่านพินการสื่อสาร UART และทำการส่งคำสั่ง AT COMMAND เพื่อตั้งค่าอุปกรณ์ให้พร้อมสำหรับการส่งข้อมูล ซึ่งการโปรแกรมฝั่งรับไม่สามารถพิมพ์คำสั่งผ่านหน้า Terminal เพื่อตั้งค่าอุปกรณ์ได้เนื่องจาก Core board ไม่มีพอร์ทสำหรับ Flash โปรแกรมประเภท Serial จึงไม่สามารถทำได้ โดยผู้จัดทำได้โปรแกรมการส่งข้อมูลไว้แล้วโดยจะมีขั้นตอนการส่งต่อไปนี้

1.ส่งคำสั่ง AT เพื่อยืนยันการใช้งานว่าอุปกรณ์สามารถสื่อสารได้ หากได้รับข้อความตอบกลับว่า +AT: OK หมายความว่าอุปกรณ์สามารถใช้งานได้

2.ส่งคำสั่ง AT+MODE=TEST/n เพื่อตั้งค่าอุปกรณ์ให้เข้าสู่โหมด TEST โดยอุปกรณ์จะตอบกลับมาว่า +MODE: TEST

3.ส่งคำสั่ง AT+TEST=RFCFG,915.200,SF9,125,8,8,20,OFF,OFF,ON โดยความถี่ที่ใช้จะต้องตรงกับความถี่ที่ฝั่งส่งใช้

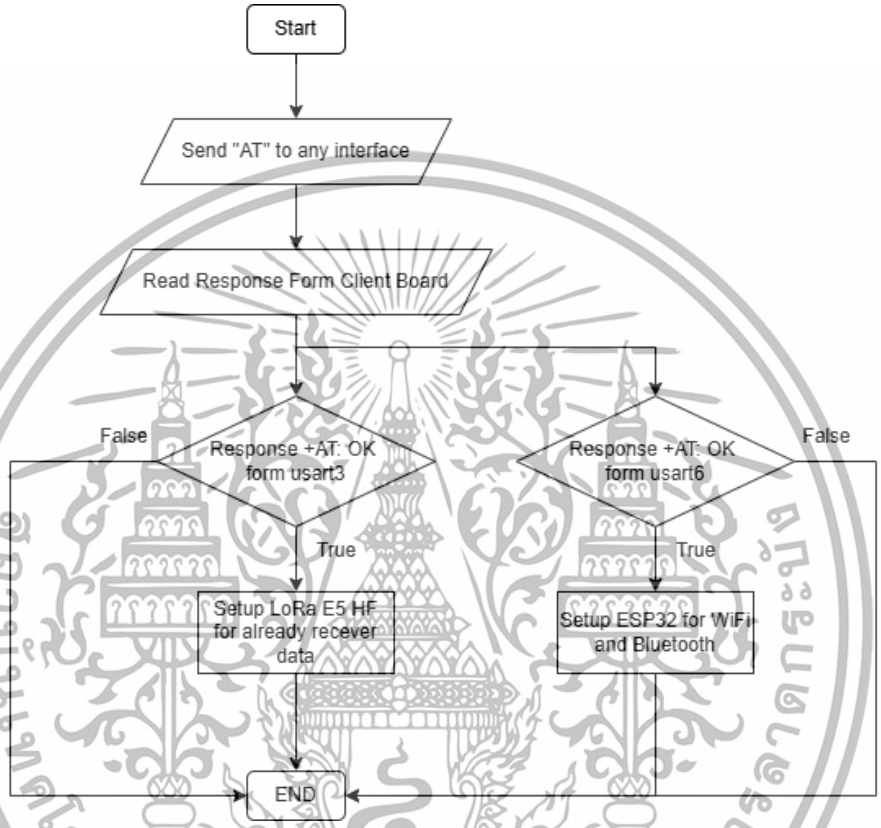
4.ส่งคำสั่ง AT+TEST=RXLRPKT เพื่อให้โมดูลทำหน้าที่รับข้อมูลที่ถูกส่งมาจากฝั่งส่ง โดยเมื่อรับข้อมูลได้ โมดูลจะสรุปการสื่อสารมาให้ คือ ค่า RSSI,SNR,LEN แสดงดังรูปที่ 3.37

รูปที่ 3.37 ผลลัพธ์การสื่อสารระหว่าง LoRa ฝั่งส่งและฝั่งรับ

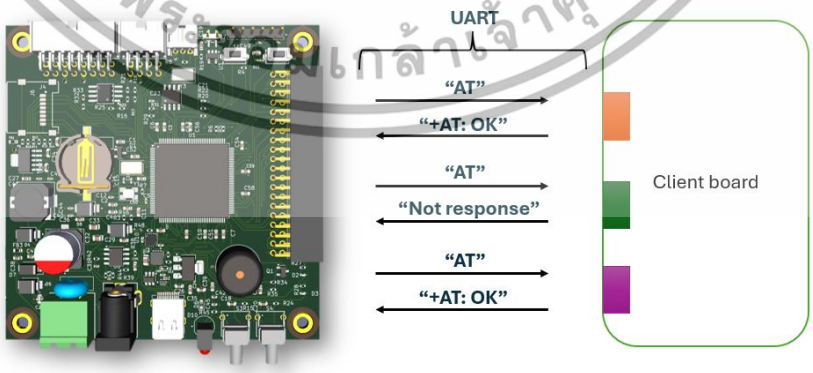
3.1.7 ออกแบบระบบตรวจจับบอร์ดเสริม (Client Board) หรือ โปรโตคอลที่ใช้งานบอร์ดหลัก (Core board)

ระบบการตรวจสอบ ใช้วิธีการส่ง “AT” ไปที่อินเตอร์เฟซต่างๆของบอร์ดเสริมที่ได้กำหนดไว้ในบอร์ดหลักโดย USART3 จะถูกกำหนดให้เป็นขาการสื่อสารระหว่าง บอร์ดหลัก และ LoRa , UART4 จะถูกกำหนดให้เป็นขาการสื่อสารระหว่าง บอร์ดหลัก และ BLE ที่จะทำหน้าที่รับค่า Bluetooth และเชื่อมต่อ WiFi เพื่อส่งออกข้อมูลที่รับเข้ามาไปที่ Broker ผ่านโปรโตคอล MQTT โดยบอร์ดเสริมทั้ง 2 บอร์ดนี้รองรับการทำงานของ AT COMMAND ลักษณะการใช้งานคือ คำสั่งพื้นฐานอย่าง “AT” ใช้เพื่อตรวจสอบการใช้งานของอุปกรณ์ดังกล่าว โดยหาก เป็น LoRa E5 HF หากพร้อม

ใช้งานจะตอบกลับด้วย +AT: OK นั้นหมายถึงพร้อมใช้งาน หรือ ESP32 จะตอบกลับด้วย OK เพื่อบอกผู้ใช้งานว่าพร้อมใช้งานและตั้งค่า (Setup) ต่อไปตามการใช้งานของผู้ใช้งาน โดยอธิบายเป็นแผนผังการทำงานแสดงดังรูปที่ 3.38 และ ลักษณะการทำงานของ AT COMMAND แสดงดังรูปที่ 3.39



รูปที่ 3.38 แผนผังการทำงานระบบตรวจสอบอุปกรณ์เสริมที่ใช้บนบอร์ดหลัก



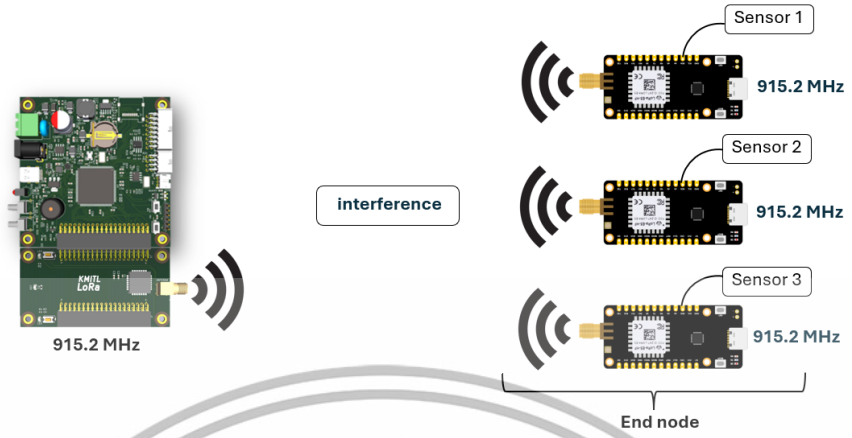
รูปที่ 3.39 ตัวอย่างการทำงานของ AT COMMAND ระหว่างบอร์ดหลักและบอร์ดเสริม

3.1.8 การออกแบบระบบการจัดการข้อมูลจำนวนมากใน LoRa E5 HF

LoRa E5 HF คือโมดูลที่ผลิตออกมาเพื่อใช้งานในลักษณะของ End node หรือสำหรับส่งข้อมูลมาที่ Gateway โดยโหมดการทำงานที่ใช้งานร่วมกันกับเกตุเวย์อื่นๆคือโหมด LWABP เป็นโหมดการเปิดใช้งานอุปกรณ์ LoRaWAN ที่อาศัยการกำหนดข้อมูลสำคัญล่วงหน้าในอุปกรณ์ อุปกรณ์ที่ตั้งค่าในโหมดนี้จะพร้อมใช้งานกับเครือข่าย LoRaWAN ได้ทันทีหลังจากเริ่มต้นทำงาน โดยไม่ต้องผ่านกระบวนการแลกเปลี่ยนข้อมูลเพื่อจับคู่คีย์ (Key Exchange) กับเครือข่ายเหมือนใน OTAA (Over-the-Air Activation) ซึ่งใน LoRaWAN ใช้การ Chirp Spread Spectrum (CSS) ซึ่งเป็นเทคนิคการมอดูเลชันที่เกี่ยวข้องกับการเปลี่ยนแปลงความถี่อย่างต่อเนื่องในลักษณะของ "chirp" เพื่อส่งข้อมูล เพื่อป้องกันการเกิดการทับซ้อนของสัญญาณ (Interference) ในกรณีที่มีหลาย End node ที่ใช้งานใน Local Network

ภายในชิ้นงานนี้ เลือกใช้ชิพ LoRa E5 HF มาประยุกต์ใช้งานเป็น Gateway หรือเป็นตัวรับ (Rx) เนื่องจากราคาที่ย่อมเยา โดยจากการทำ พบว่าเนื่องจากชิพดังกล่าว ไม่มี Firmware ที่ไว้ใช้จัดการข้อมูลจำนวนมาก ที่เข้ามาในโมดูลดังกล่าว และโหมดการทำงาน LWABP ที่จะใช้เพื่อสื่อสารระหว่างตัวส่ง (Tx) LoRa E5 HF และตัวรับ (Rx) ซึ่งเป็นชิพเดียวกัน จะไม่สามารถใช่งานได้ เนื่องด้วยปัญหาดังกล่าว แก้ไขได้โดยใช้โหมดการทำงาน TEST ที่มีอยู่แล้วในชิพดังกล่าว พบว่าสามารถทำให้ชิพทั้งสองตัว สามารถสื่อสารกันได้ แต่พบข้อจำกัดคือในโหมดดังกล่าว จำกัดความถี่การใช้งานกล่าวคือภายในชิพได้จำกัดการใช้งานความถี่ ได้เพียงความถี่เดียวเท่านั้น ซึ่งการใช้งานหากมี End node หลายตัวอาจเกิด interference ได้แสดงดังรูปที่ 3.40

ภายหลังได้เลือกใช้วิธีการเข้าถึงข้อมูลด้วยความถี่คล้ายกับ FDMA หรือ Frequency division แสดงดังรูปที่ 3.41 โดยกำหนดการใช้งาน End node ให้ใช้ความถี่ที่แตกต่างกันในการส่งข้อมูล เนื่องจากชิพ LoRa E5 HF ในโหมด TEST สามารถกำหนดค่าความถี่ได้จากคำสั่ง AT+TEST=RFCFG

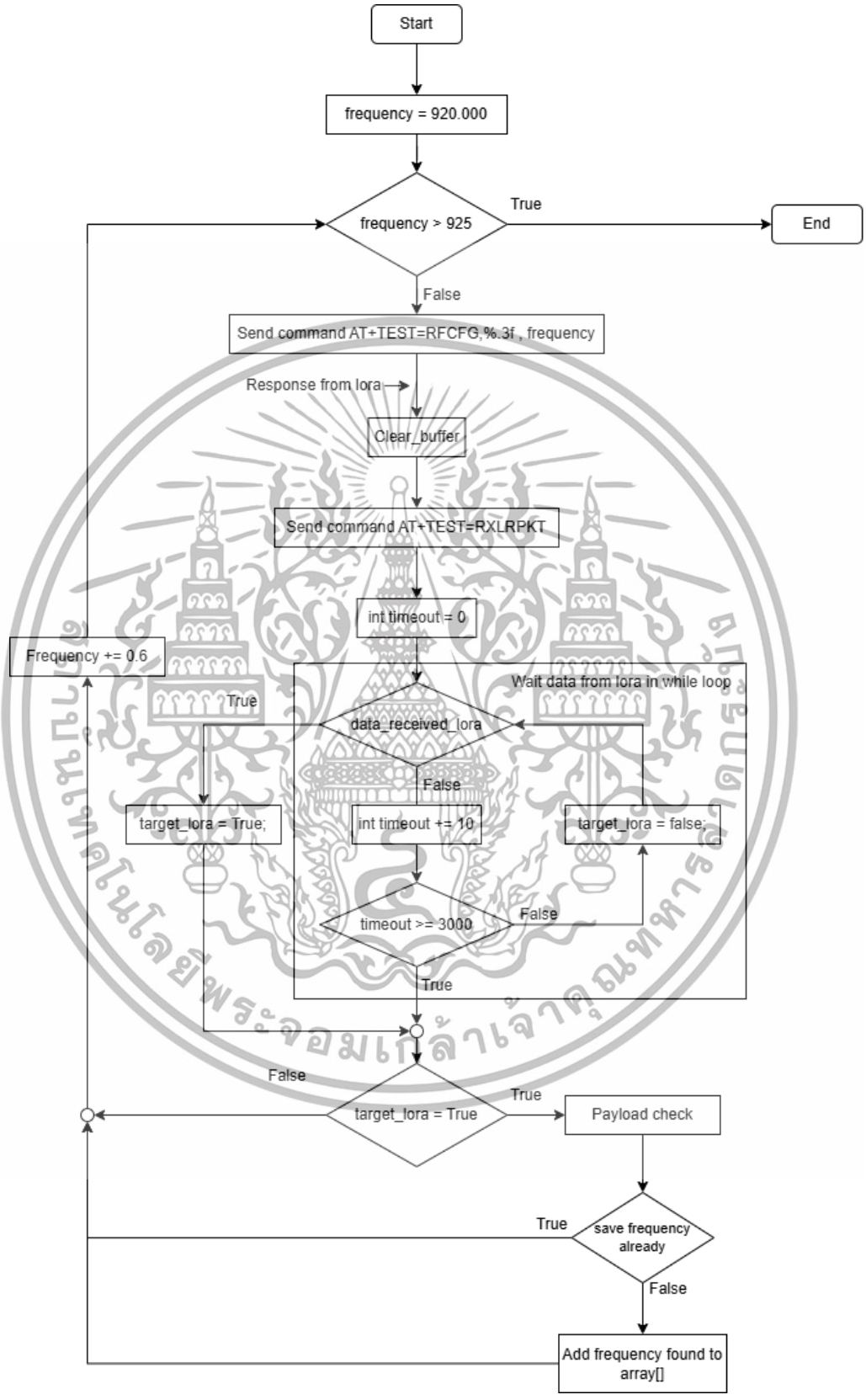


รูปที่ 3.40 การทำงานของ End node ในโหมด TEST ทั้งฝั่งส่งและรับ

รูปที่ 3.41 การทำงานของ End node ในโหมด TEST ทั้งฝั่งส่งและรับในความถี่ที่แตกต่างกัน

3.1.9 ออกแบบระบบตรวจสอบ End node ที่ใช้งานใน LoRa

การออกแบบใช้วิธีการ สลับความถี่การตรวจสอบ โดยจะเริ่มต้นที่ความถี่ 920 MHz จากนั้นความถี่ถัดไปอยู่ที่ 920.6 MHz ใช้ระยะห่างความถี่ที่ 0.6 MHz จนครบที่ความถี่ 925 MHz โดยในฝั่งส่งจะแนบข้อความ “NODE” มาที่ต้นข้อความ และ ฝั่งรับหรือ Gateway จะประมวลผลข้อมูลโดยเมื่อปรับไปที่ความถี่ดังกล่าว หากมีข้อมูลเข้ามาตรวจสอบจากข้อความ “NODE” หากพบจะทำการบันทึกความถี่ดังกล่าวไว้ใน list และสลับไปที่ความถี่ถัดไปที่ระยะห่าง 0.6 MHz จนครบ จากนั้นจะทำการสรุปว่ามี End node ในการใช้งานเท่าไหร่ และใช้ list ดังกล่าวในการรับข้อมูลที่ความถี่นั้นๆ มาประมาลผล และ ส่งออกข้อมูลผ่าน ESP32 บน MQTT Protocol ไปที่ Broker ที่ได้กำหนดไว้ และมีแผนผังการทำงานแสดงดังรูปที่ 3.42



รูปที่ 3.42 แผนผังการทำงานค้นหาจำนวน Endnode ที่ใช้งานบน LoRa

3.1.10 การใช้งาน AT COMMAND บน ESP32 เพื่อสร้างการเชื่อมต่อ WiFi

3.1.10.1 หลักการทำงานของ ESP-AT

ESP-AT ถูกพัฒนาโดย Espressif เพื่อการใช้งานเชื่อมต่อ เทคโนโลยีไร้สายได้โดยเพียงส่งคำสั่งผ่านพินการเชื่อมต่อแบบ UART I2C หรือ SPI โดย AT-COMMAND บน ESP32 จะประพฤติตนเป็น Slave เพื่อรอรับคำสั่งจาก Microcontroller ที่ทำหน้าที่เป็น Host และประโยชน์การใช้งานคือ 1. เชื่อมต่อเครือข่ายไร้สายได้อย่างง่าย 2. เชื่อมต่อกับคลาวด์แพลตฟอร์มได้ง่ายเช่น MQTT Broker เป็นต้น ลักษณะการใช้งานแสดงดังรูปที่ 3.43 โดย AT-COMMAND บน ESP32 มีฟังก์ชันการทำงานที่หลากหลายดังนี้

ฟังก์ชันการทำงานของ AT-COMMAND บน ESP32

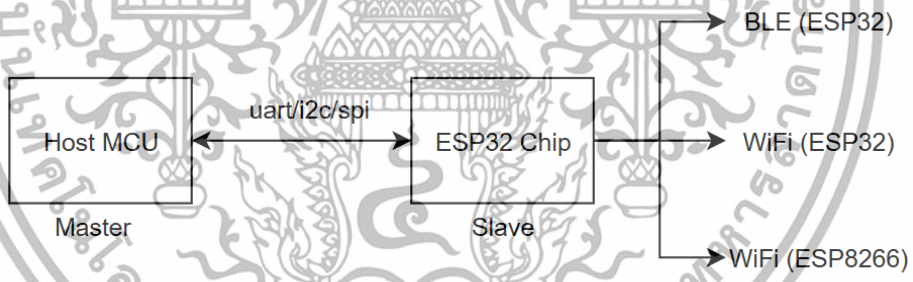
1.คำสั่ง WiFi

2.คำสั่ง TCP/IP

3.คำสั่ง BLE / Bluetooth Classic

4.คำสั่ง MQTT

5.คำสั่ง Ethernet



รูปที่ 3.43 การทำงานภาพรวมของ ESP-AT

3.1.10.2 ฮาร์ดแวร์ที่รองรับการใช้งาน ESP-AT

ฮาร์ดแวร์ที่รองรับ ESP-AT ส่วนใหญ่จะเป็นอุปกรณ์ในตระกูล Espressif ESP32 และ ESP8266 รวมถึงโมดูลและบอร์ดพัฒนา (development boards) ที่ใช้ชิปจาก Espressif ซึ่งรองรับการสื่อสารผ่าน AT Commands ได้ ซึ่งสามารถสรุปได้ดังตารางที่ 3.1

ตารางที่ 3.1 ความเข้ากันได้ของเวอร์ชั่น AT-ESP กับชิพจาก Espressif

Chip	Wireless	AT Firmware
ESP32-C6	Wi-Fi 6 + BLE 5.0	V.4.0.0.0
ESP32-C3	Wi-Fi 4 + BLE 5.0	V3.3.0.0
ESP32-C2	Wi-Fi 4 (or BLE 5.0)	V.3.3.0.0
ESP32	Wi-Fi 4 + BLE V4.2 (+BT)	V3.4.0.0D
ESP32-S2	Wi-Fi 4	V3.4.0.0

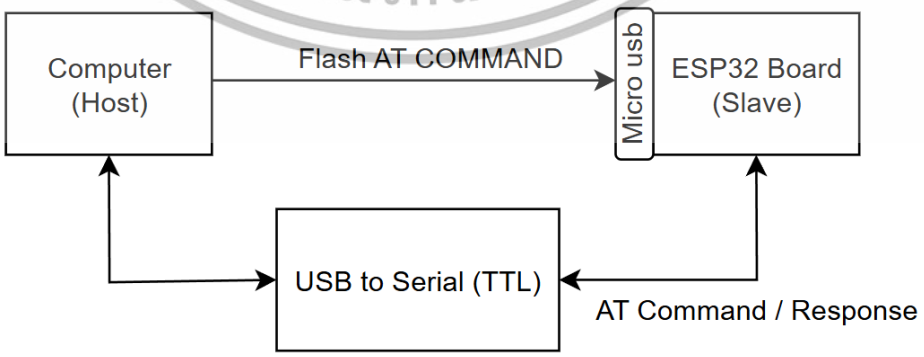
3.1.10.3 การใช้งาน ESP-AT บน ESP32

AT Firmware ทำหน้าที่เปิดการทำงานของ ESP32 ในโหมด AT Command ซึ่งช่วยให้ ESP32 สามารถทำงานเป็นอุปกรณ์ Slave ที่ตอบสนองต่อคำสั่งที่ส่งจาก Host MCU ผ่านการเชื่อมต่อ UART (Serial Communication) คำสั่ง AT เป็นมาตรฐานสำหรับการควบคุมโมดูลสื่อสาร หลังจาก Flash AT Firmware ลงบน ESP32 จะทำให้ ESP32 ทำงานได้เหมือนโมดูลสื่อสารสำเร็จรูป เช่น ESP8266 หรือโมดูล Wi-Fi อื่น ๆ โดยไม่ต้องพัฒนาโปรแกรมหรือเขียนโค้ดเพิ่มเติม เพียงแค่ส่งคำสั่ง AT ผ่าน UART ก็สามารถควบคุมฟังก์ชันต่าง ๆ ได้ และในปริญญานิพนธ์นี้ ผู้พัฒนาได้เลือกใช้ บอร์ด ESP32 และเนื่องจากเป็นบอร์ดสำเร็จรูป Firmware บนอุปกรณ์ดังกล่าวจะไม่ Support AT COMMAND เพราะค่าตั้งต้นจากโรงงานจะเป็น Firmware ที่ใช้งานกับ Arduino IDE เท่านั้น เพื่อการใช้งาน AT COMMAND จำเป็นต้อง Flash Firmware AT COMMAND เพื่อใช้งาน โดยมีขั้นตอนดังนี้

ขั้นตอนการ Flash AT COMMAND บน บอร์ด ESP32

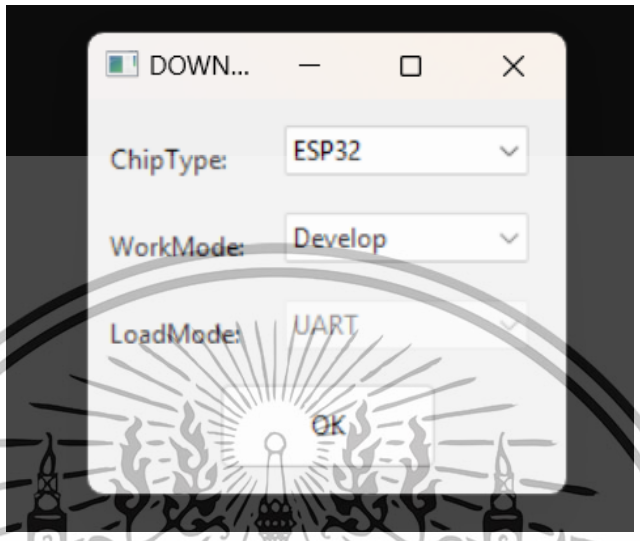
1.การเชื่อมต่อฮาร์ดแวร์

เชื่อมต่อแสดงดังรูปที่ 3.44 เพื่อเริ่มการ Flash AT COMMAND



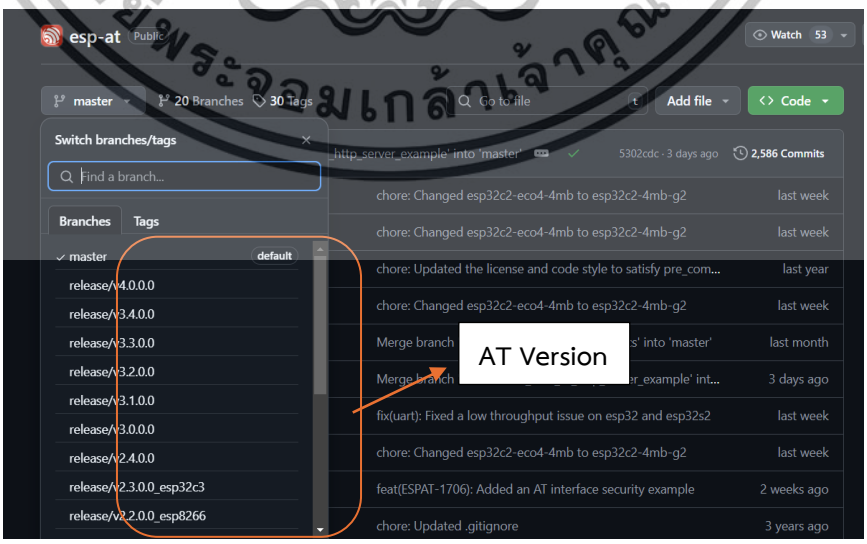
รูปที่ 3.44 การเชื่อมต่อเพื่อ Flash AT COMMAND จาก Host

2.ดาวน์โหลดโปรแกรม Flash Download Tools for Windows เพื่อ Flash AT Firmware ลงบนบอร์ด ESP32 เมื่อเปิดโปรแกรมให้เลือกโหมดแสดงดังรูปที่ 3.45

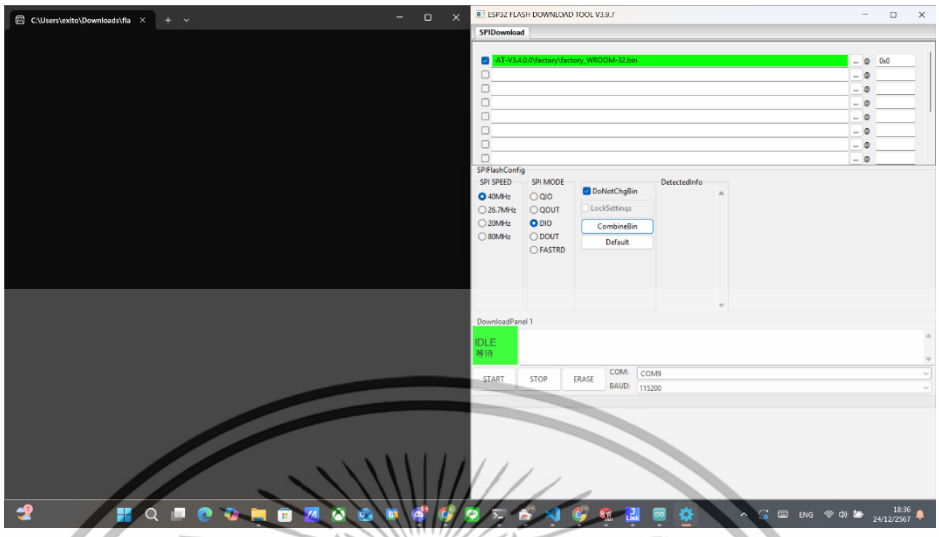


รูปที่ 3.45 โหมดการ Flash AT Fimware ลงบนบอร์ด ESP32

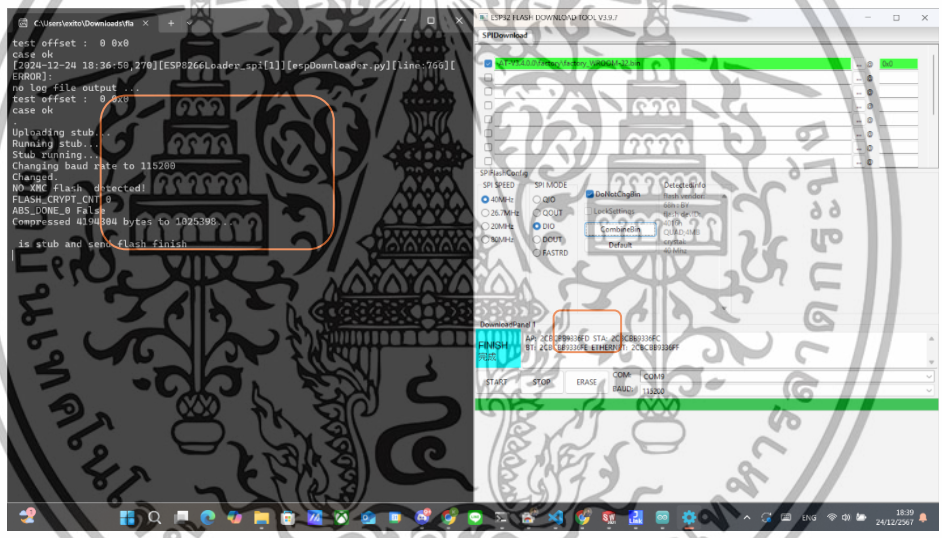
3.ดาวน์โหลดไฟล์ ESP-AT ได้จาก https://github.com/espressif/esp-at จากนั้นกดเลือกที่ master และเลือก Version ให้ตรงตามตามตารางที่ 3.1 โดยเลือก Version ได้แสดงดังรูปที่ 3.46 จากนั้นเลือกไฟล์ factory_WROOM-32.bin เพื่อ Flash Firmware ลงบน ESP32 โดยเลือก SPI Speed ที่ 40 MHz และ SPI Mode ที่ DIO และเลือก DoNotChgBin จากนั้นเลือก COM ให้ตรงกับ ESP32 เลือก BAUD ที่ 115200 Bps จากนั้นกด Start เพื่อเริ่มการ Flash แสดงดังรูปที่ 3.47 โดยหาก Flash เสร็จเรียบร้อยจะปรากฏข้อมูลดังแสดงรูปที่ 3.48



รูปที่ 3.46 การเลือก Version ให้ตรงตามบอร์ดที่ใช้งานเพื่อ Flash AT-Firmware



รูปที่ 3.47 การตั้งค่าเพื่อ Flash AT-Firmware



รูปที่ 3.48 การตั้งค่าเพื่อ Flash AT-Firmware

3.1.11 การออกแบบโปรแกรมเชื่อมต่อ WiFi และ Broker ผ่าน AT COMMAND บน Core board

โปรแกรมเชื่อมต่อ Wi-Fi จะเริ่มต้นจากตรวจสอบการเชื่อมต่อบอร์ด ESP32 เมื่อเชื่อมต่อเข้ากับ Coreboard จากวิธีการส่ง AT ไปที่พินการเชื่อมต่อของ ESP32 หากตอบกลับด้วย OK จะเริ่มทำการเชื่อมต่อ WiFi ด้วย AT COMMAND โดยสามารถสรุปขั้นตอนการเชื่อมต่อ และผลการตอบสนองได้ได้แสดงดังตารางที่ 3.2 และมีการทำงาน ขั้นตอนดังกล่าวทำบน USB Serial เพื่อกำหนดการเชื่อมต่อไว้ล่วงหน้า เนื่องจากบอร์ด ESP32 มี ROM ภายในอยู่แล้ว ทำให้เมื่อไม่จ่ายพลังงานให้กับบอร์ดดังกล่าว บอร์ดจะยังสามารถจดจำการเชื่อมต่อได้

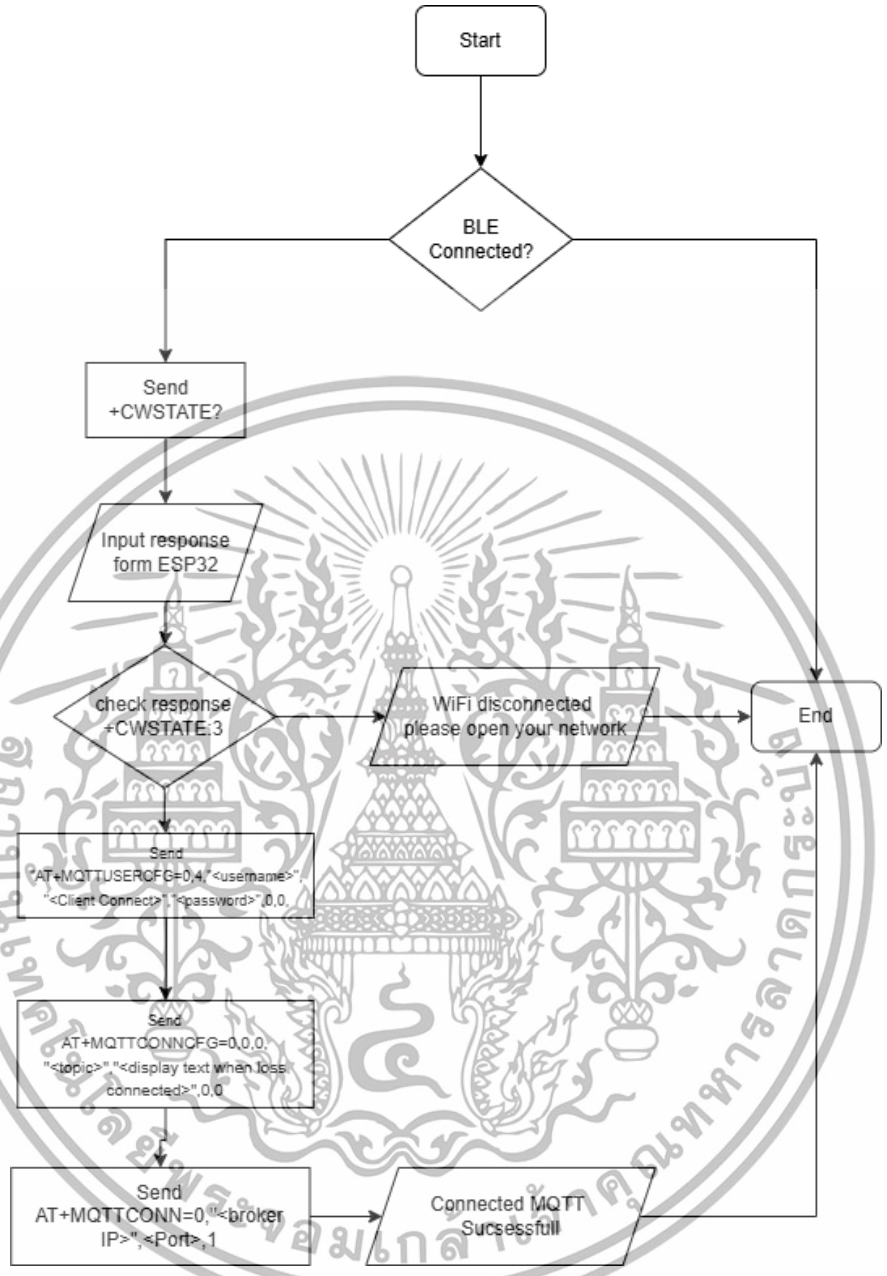
ตารางที่ 3.2 ขั้นตอนการเชื่อมต่อ WiFi

ขั้นตอน	คำสั่งที่ใช้ส่ง	ข้อมูลตอบสนองของ ESP32	คำอธิบาย
1	AT+CWINIT?	AT+CWINIT=<init>	เพื่อตรวจสอบสถานะของ Wi-Fi Driver - ถูกเปิดใช้งาน, 1 - ปิดใช้งาน,0
2	AT+CWMODE=<init>	OK	เพื่อกำหนดโหมดการทำงานของ Wi-Fi โดยค่า init คือ - STA Mode , 1 เพื่อทำให้ ESP32 เป็น Client เพื่อเชื่อมต่อ WiFi - SoftAP,2, เพื่อใช้งานในโหมด AP (Access point) - 3, ทำทั้ง 2 Mode
3	AT+CWLAP	+CWLAP:(3,"kattiya's Galaxy S10",-49,"72:cd:70:41:0b:22",1,-1,-1,4,4,7,0) +CWLAP:(0,"KMITL-WIFI",-67,"00:2e:c7:8f:ef:61",11,-1,-1,0,0,7,0)	แสกน AP Rounter เพื่อเชื่อมต่อ WiFi
4	AT+CWJAP=<ssid>,<password>	WIFI CONNECTED WIFI GOT IP	เพื่อเชื่อมต่อ WiFi ที่ต้องการ โดย - SSID ชื่อ Wi-Fi - Password รหัสผ่าน

ตารางที่ 3.2 (ต่อ) ขั้นตอนการเชื่อมต่อ WiFi

ขั้นตอน	คำสั่งที่ใช้ส่ง	ข้อมูลตอบสนองของ ESP32	คำอธิบาย
5	AT+MQTTUSERCFG= <LinkID>,<Scheme>, <ClientID>,<Username>, <Password>	OK	ใช้สำหรับการเชื่อมต่อ Broker ที่ใช้งาน โดย LinkID คือ ลำดับการใช้งาน Scheme คือ รูปแบบการเชื่อมต่อ 0 คือ TCP 1 คือ TLS 2 คือ MQTT over WebSocket 3 คือ MQTT over Secure WebSocket 4 คือ MQTT over SSL (TLS)
6	AT+MQTTCONNCFG <LinkID>,<keepalive>, <disable_clean_session>, <"lwt_topic">,<"lwt_msg">, <lwt_qos>,<lwt_retain>	OK	ใช้สำหรับการเชื่อมต่อไปที่ Topic ที่ใช้งานบน Broker

จากการตั้งค่า ESP32 ให้เชื่อมต่อ WiFi จากนั้นภายใน Coreboard จะทำการตรวจสอบการเชื่อมต่อ ของ Wi-Fi ทุกครั้ง หากเชื่อมต่อจะเริ่มการตั้งค่า เชื่อมต่อ Broker ที่ได้ตั้งค่าเอาไว้แล้วล่วงหน้า โดยมีแผนผังการทำงานแสดงดังรูปที่ 3.49

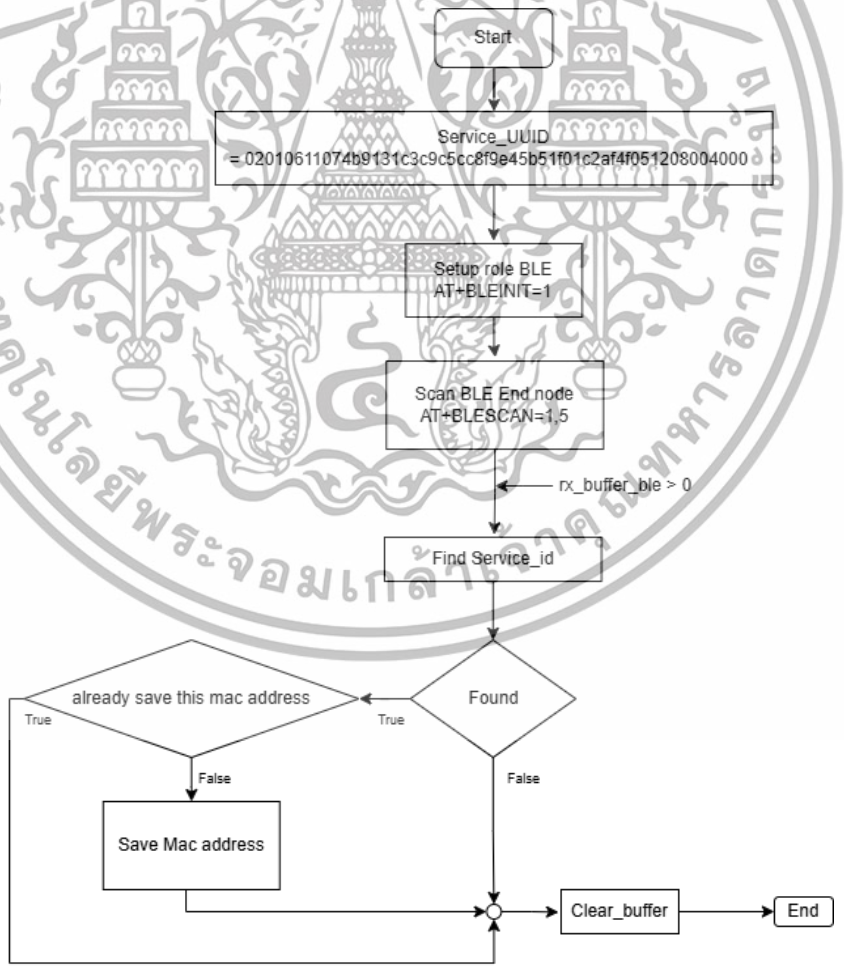


รูปที่ 3.49 แผนผังการเชื่อมต่อ Broker ใน Coreboard

3.1.12 การออกแบบระบบเพื่อค้นหาอุปกรณ์ BLE End node และ รองรับการใช้งาน BLE บน Coreboard

โปรแกรมเริ่มต้นจากการตรวจสอบโปรแกรม ตรวจสอบบอร์ดเสริม BLE&ESP32 ที่เชื่อมต่อ หากบอร์ดเสริม BLE&ESP32 เชื่อมต่ออยู่ ใน Firmware ดังกล่าวจะทำหน้าที่ ส่งคำสั่ง AT+RST เพื่อรีเซ็ตการเชื่อมต่อเดิมที่เคยเชื่อมต่ออยู่ ป้องกันการเกิด เชื่อมต่อซ้ำ จากนั้น รอการตอบกลับจากบอร์ดเสริม เพื่อยืนยันการรีเซ็ต เมื่อบอร์ดเสริมตอบกลับในข้อความ “OK” จะเริ่มส่งคำสั่ง

AT+BLEINIT =1 โดยเป็นการตั้งค่าให้ บอร์ดเสริม มีสถานะเป็น Client เพื่อรอการค้นหาอุปกรณ์ BLE End node ที่อยู่รอบๆ โดยใช้คำสั่ง AT+BLESCAN=<mode>,<duration> จากนั้น นำบัฟเฟอร์ ที่ได้จากการค้นหาอุปกรณ์ BLE มาทำการแยกอุปกรณ์ BLE End node ออกจากอุปกรณ์อื่นที่พบเจอ โดยค้นหาจาก Service_UUID : 4fafc201-1fb5-459e-8fcc-c5c9c331914b ซึ่งอุปกรณ์ BLE End node จะต้องถูกตั้งค่าให้มี Service_UUID ที่เหมือนกัน เพื่อการแยกอุปกรณ์ ออกจาก BLE End node อื่นๆที่ไม่เกี่ยวข้องในการค้นหา เมื่อเจออุปกรณ์ จะทำการบันทึก Mac address โดยเก็บไว้ใน Array ที่มีชื่อว่า target_macs[] หลังจากนั้นนำ Array ดังกล่าว ไปสร้างการเชื่อมต่อจากบอร์ดหลักไปที่ BLE End node ดังกล่าว เมื่อเชื่อมต่อเสร็จเรียบร้อย ระบบจะทำการค้นหา CHARACTERISTIC_UUID ที่อยู่ใน BLE เพื่อขอรับข้อมูลที่ถูกส่งมาจาก BLE โดยมีแผนผังการทำงานแสดงดังรูปที่ 3.50



รูปที่ 3.50 แผนผังการทำงานของ BLE

3.1.13 การใช้งาน Broker เพื่อรับข้อความที่มาจาก Core board

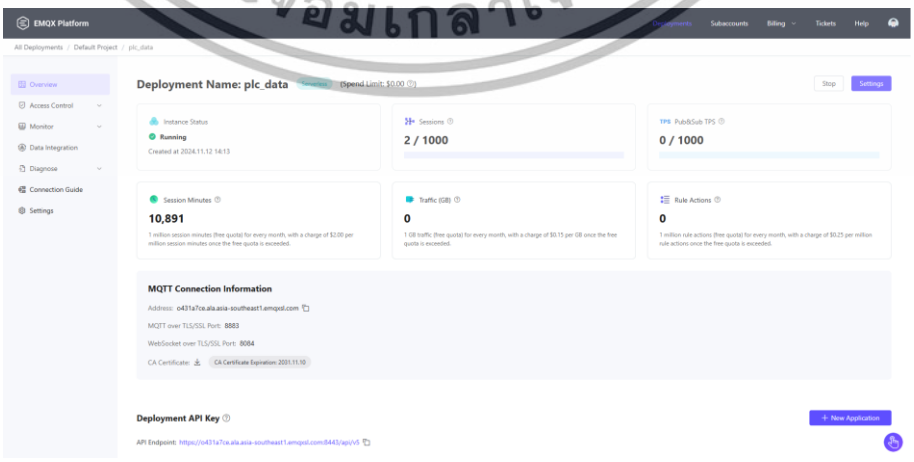
EMQX เป็น MQTT Broker (Message Queuing Telemetry Transport Broker) แบบ Open Source ที่ถูกพัฒนาใช้งานกับ การสื่อสารระหว่างอุปกรณ์ IoT (Internet of Things) รองรับการส่งข้อความแบบ Pub/Sub (Publish/Subscribe) โดยมีขั้นตอนการใช้งานดังต่อไปนี้

ขั้นตอนการใช้งาน EMQX

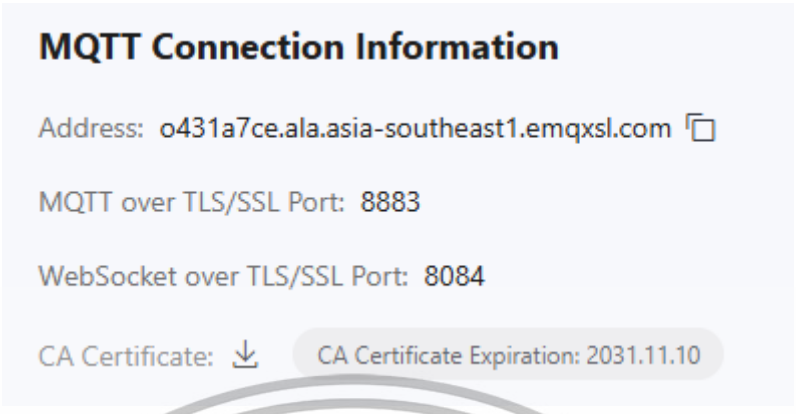
1. สร้างบัญชีการใช้งานได้จากเว็บไซต์ https://accounts.emqx.com/signup
2. สร้าง Serverless เพื่อใช้งานตัว Broker MQTT แสดงดังรูปที่ 3.51 โดยเลือก Region เป็น Asia-Pacific และกด Deploy

รูปที่ 3.51 แผนการใช้งาน Serverless บน EMQX

3. เมื่อเข้ามาที่ Server ที่สร้างเสร็จแล้วจะพบหน้า Dashboard เพื่อดูสถานะของ Server และ Session ที่ใช้งานอยู่แสดงดังรูปที่ 3.52 และจะพบค่า IP ที่ Server ใช้ เพื่อนำค่าดังกล่าวไปเชื่อมต่อจาก Core board แสดงดังรูปที่ 3.53



รูปที่ 3.52 หน้า Dashboard บน EMQX

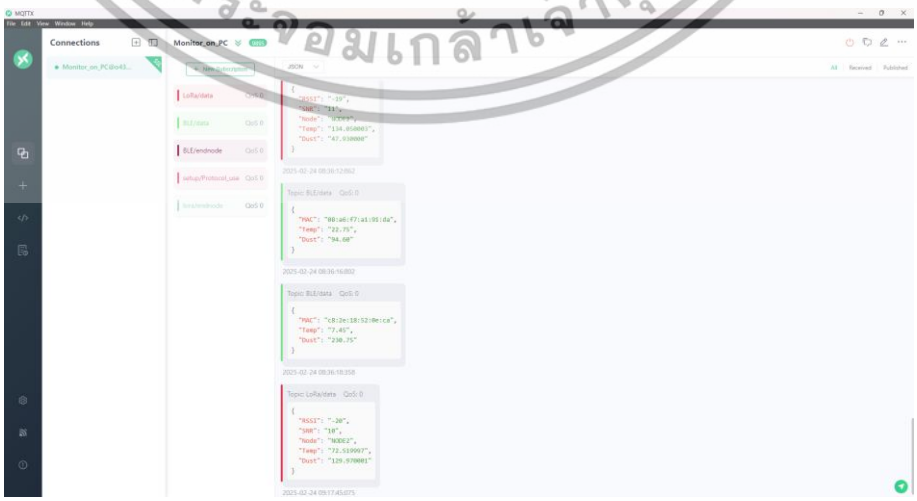


รูปที่ 3.53 MQTT Connection Information

4. สร้างบัญชีการเข้าใช้งาน Server ดังกล่าวได้จาก หัวข้อ Access control และ Authentication แสดงดังรูปที่ 3.54

รูปที่ 3.54 สร้างบัญชีการเข้าใช้งาน Server

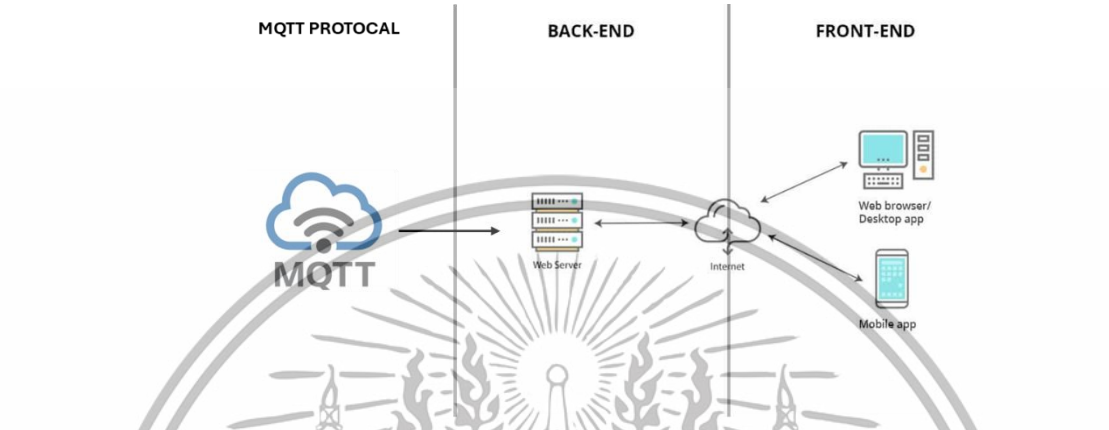
5. ภายในการใช้งาน EMQX มีโปรแกรม MQTTX สำหรับเฝ้าระวังข้อมูลที่เข้ามาใน Server โดยโปรแกรมมีลักษณะแสดงดังรูปที่ 3.55



รูปที่ 3.55 หน้าต่างเฝ้าระวังข้อมูลที่ถูกส่งเข้ามาใน Server

3.1.14 การออกแบบเว็บแสดงผล

การออกแบบหน้าเว็บสำหรับโปรโตคอล MQTT ตามภาพนี้มีรายละเอียดดังนี้ แสดงดังรูปที่ 3.56



รูปที่ 3.56 โครงสร้างเว็บแสดงผล

3.1.14.1 Front-End

Web Browser/Desktop App และ Mobile App : เป็นส่วนติดต่อผู้ใช้ที่ทำให้ผู้ใช้สามารถส่งหรือรับข้อมูลผ่านโปรโตคอล MQTT โดยอาจใช้ JavaScript Libraries เช่น MQTT.js หรือการเชื่อมต่อผ่าน WebSocket เพื่อสื่อสารกับ Back-End ให้ผู้ใช้สามารถสมัคร (subscribe) และส่ง (publish) ข้อความไปยังหัวข้อ (topics) ของ MQTT รวมถึงแสดงข้อมูลแบบเรียลไทม์ที่ได้รับจากอุปกรณ์ IoT หรือระบบต่าง ๆ ที่เชื่อมต่อกับโปรโตคอลนี้

3.1.14.2 Back-End

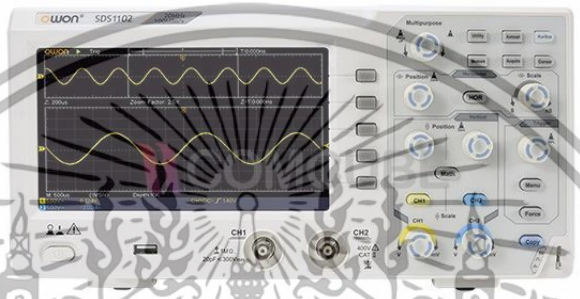
Web Server : เป็นตัวกลางที่รับส่งข้อมูลระหว่าง Front-End กับ MQTT Broker โดยอาจเชื่อมต่อกับ MQTT Broker ผ่าน WebSocket หรือการเชื่อมต่อ TCP/IP ทำการแปลงข้อมูล MQTT ให้เป็นรูปแบบที่ Front-End สามารถเข้าใจได้ (เช่น JSON) และยังอาจจัดการเกี่ยวกับการตรวจสอบสิทธิ์ผู้ใช้และการรักษาความปลอดภัยในการสื่อสารข้อมูล

MQTT Broker : ทำหน้าที่เป็นตัวกลางในการรับส่งข้อมูลระหว่างอุปกรณ์หรือเซิร์ฟเวอร์หลาย ๆ ตัว โดยรับข้อมูลจากผู้ส่ง (publishers) และส่งต่อข้อมูลไปยังผู้รับ (subscribers) ที่ได้สมัครหัวข้อไว้

3.2 เครื่องมือที่ใช้ในการทดลอง

3.2.1 เครื่องออสซิลโลสโคป (Oscilloscope)

เครื่องออสซอลโลสโคปเป็นเครื่องมือที่ใช้วัดองค์ประกอบของสัญญาณต่าง ๆ ให้ออกมาเป็นรูปร่างของสัญญาณไฟฟ้า เช่น ลักษณะรูปร่างของสัญญาณ, การวัดขนาดของสัญญาณ, การวัดความถี่ของสัญญาณ รวมถึงการวัดกระแสไฟฟ้า แสดงดังรูปที่ 3.57



รูปที่ 3.57 เครื่องออสซิลโลสโคป (Oscilloscope)

3.2.2 เครื่องจ่ายไฟ (Power Supply)

เครื่องจ่ายไฟเป็นอุปกรณ์ที่จ่ายแรงดันไฟฟ้าให้กับ Board ต่างๆ ซึ่งจะเป็นการแปลงแรงดันไฟฟ้าจากรูปแบบหนึ่งให้เป็นรูปแบบหนึ่ง ซึ่งสามารถควบคุมแรงดันไฟฟ้าที่จ่ายออกมาได้ แสดงดังรูปที่ 3.58

รูปที่ 3.58 เครื่องจ่ายไฟ (Power Supply)

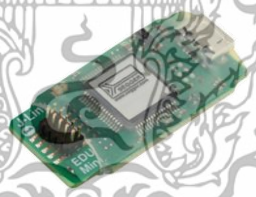
3.2.3 เครื่องวัดสเปกตรัม (Spectrum Analyzer)

เครื่องวัดสเปกตรัมเป็นอุปกรณ์ที่สามารถวัดสัญญาณในความถี่สูง โดยสามารถแสดงค่าความถี่และค่าความแรงของสัญญาณในแต่ละความถี่ แสดงดังรูปที่ 3.59

รูปที่ 3.59 เครื่องวัดสเปกตรัม (Spectrum Analyzer)

3.2.4 J-Link EDU Mini - Debug Probes

J-Link EDU Mini เป็น debug probe ขนาดเล็กจาก SEGGER ที่ใช้สำหรับการดีบักและโปรแกรมไมโครคอนโทรลเลอร์ รองรับโปรโตคอล SWD (Serial Wire Debug) และ JTAG เหมาะสำหรับการพัฒนาโปรแกรม แสดงดังรูปที่ 3.60



รูปที่ 3.60 J-Link EDU Mini

3.2.5 โปรแกรม KiCad

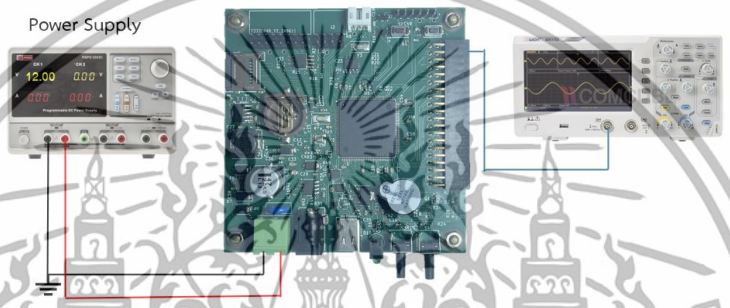
เป็นโปรแกรมสำหรับการออกแบบวงจรอิเล็กทรอนิกส์ภายใน Board ทำหน้าที่ออกแบบวงจร (Schematic) และออกแบบ PCB (Printed Circuit Board) ที่มีประสิทธิภาพสูง แสดงดังรูปที่ 3.61

รูปที่ 3.61 โปรแกรม KiCad

3.3 การจัดเก็บผลการทดลอง

3.3.1 การวัดการทำงานของวงจรรับแรงดันไฟฟ้า 12 V

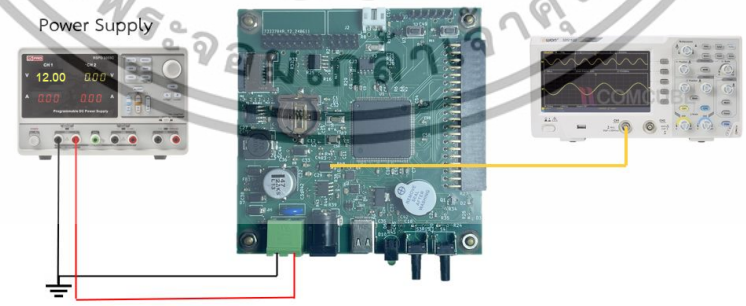
ทำการวัดการทำงานของวงจรที่ทำการรับแรงดันไฟฟ้าจากการป้อนแรงดันไฟฟ้าเข้าสู่ Board เพื่อป้อนแรงดันไฟฟ้าให้กับวงจรอื่น ๆ ภายใน Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.62



รูปที่ 3.62 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 12 V จาก Core Board

3.3.2 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 5 V ที่ใช้ภายใน Core Board

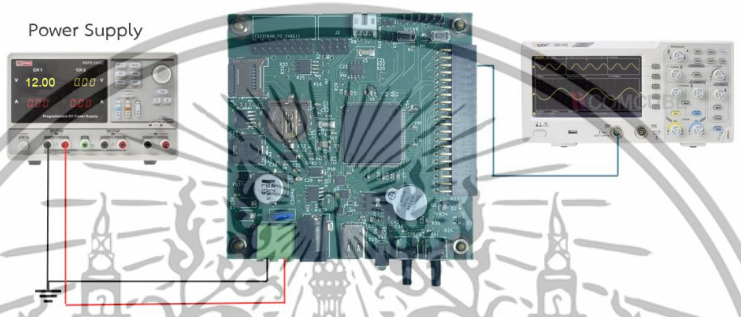
ทำการวัดการทำงานของวงจรที่ทำการรับแรงดันไฟฟ้าจากวงจรจ่ายแรงดันไฟฟ้า 12 V นำมาแปลงเป็นแรงดันไฟฟ้า 5 V เพื่อใช้ในการป้อนแรงดันไฟฟ้า 5 V ภายใน Core Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.63



รูปที่ 3.63 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 5 V ที่ใช้ใน Core Board

3.3.3 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 5 VEXT ที่ใช้จ่ายให้ Client Board

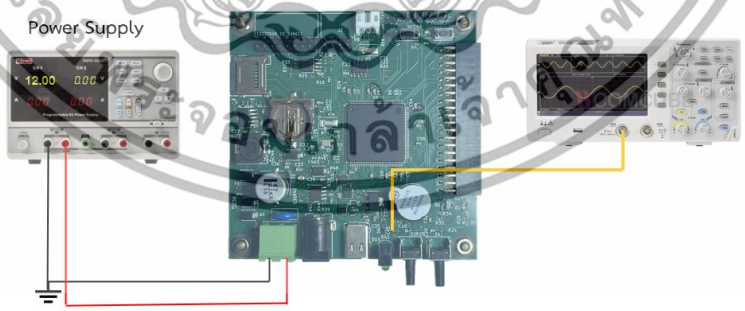
ทำการวัดการทำงานของวงจรที่ทำการรับแรงดันไฟฟ้าจากวงจรจ่ายแรงดันไฟฟ้า 12 V นำมาแปลงเป็นแรงดันไฟฟ้า 5 VEXT เพื่อใช้ในการป้อนแรงดันไฟฟ้า 5 VEXT ใช้จ่ายให้ Client Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.64



รูปที่ 3.64 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 5 VEXT ที่ใช้จ่ายให้ Client Board

3.3.4 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 3.3 V ที่ใช้ภายใน Core Board

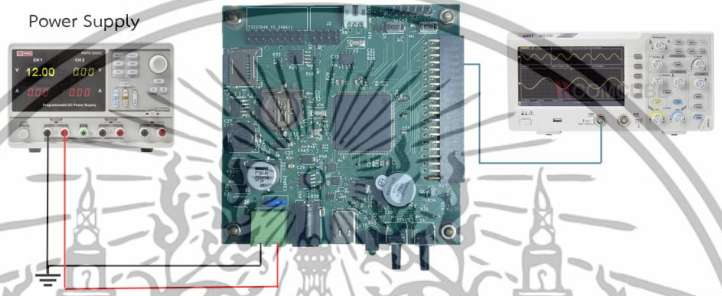
ทำการวัดการทำงานของวงจรที่ทำการรับแรงดันไฟฟ้าจากวงจรจ่ายแรงดันไฟฟ้า 5 V นำมาแปลงเป็นแรงดันไฟฟ้า 3.3 V เพื่อใช้ในการป้อนแรงดันไฟฟ้า 3.3 V ภายใน Core Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.65



รูปที่ 3.65 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 3.3 V ที่ใช้ใน Core Board

3.3.5 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 3.3 VEXT ที่ใช้จ่ายให้ Client Board

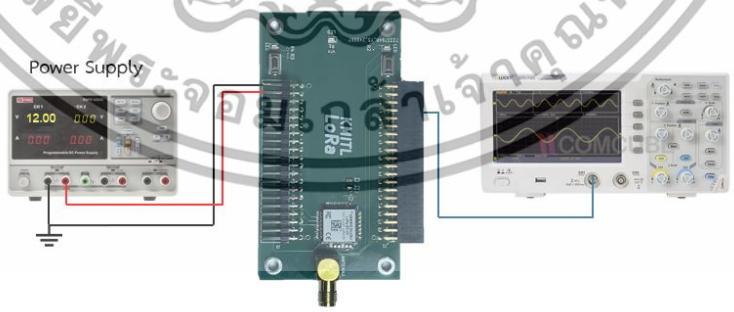
ทำการวัดการทำงานของวงจรที่ทำการรับแรงดันไฟฟ้าจากวงจรจ่ายแรงดันไฟฟ้า 5 V นำมาแปลงเป็นแรงดันไฟฟ้า 3.3 VEXT เพื่อใช้ในการป้อนแรงดันไฟฟ้า 3.3 VEXT ใช้จ่ายให้ Client Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.66



รูปที่ 3.66 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 3.3 VEXT ที่ใช้จ่ายให้ Client Board

3.3.6 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 3.3 VEXT ที่ใช้ใน Client Board Local Network (LoRa Board)

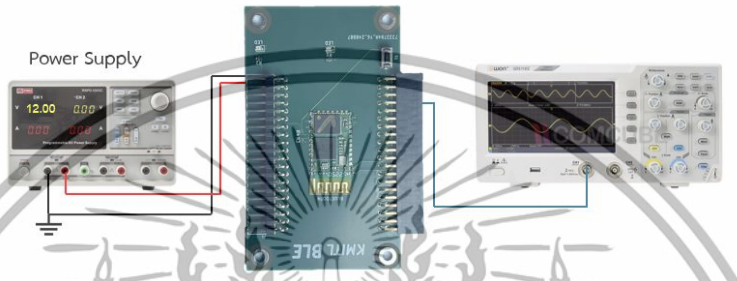
ทำการวัดการทำงานของวงจรรับแรงดันไฟฟ้าจาก Core Board นำมาใช้ในการป้อนแรงดันไฟฟ้าภายใน Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.67



รูปที่ 3.67 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 3.3 VEXT ที่รับมาจาก Core Board

3.3.7 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 3.3 VEXT ที่ใช้ใน Client Board Local Network (BLE Board)

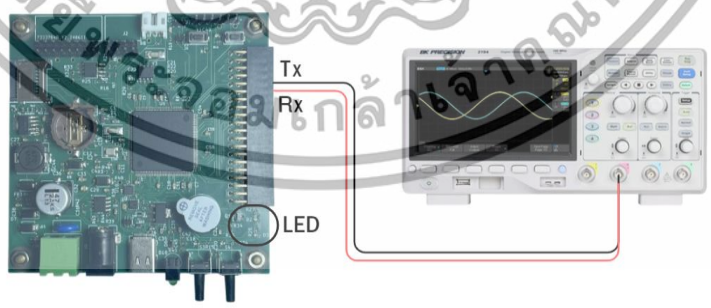
ทำการวัดการทำงานของวงจรรับแรงดันไฟฟ้าจาก Core Board นำมาใช้ในการป้อนแรงดันไฟฟ้าภายใน Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ ดังรูปที่ 3.68



รูปที่ 3.68 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 3.3 VEXT ที่รับมาจาก Core Board

3.3.8 ทดสอบการทำงานของ Coreboard และการสื่อสารแบบ UART ผ่านพินอินเตอร์เฟซ

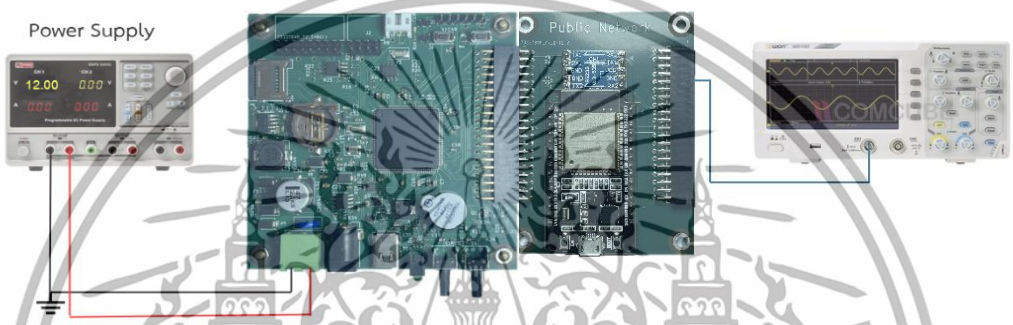
ทดสอบการส่งข้อมูลผ่าน Coreboard ว่าสามารถส่งข้อมูลในโปรโตคอลการสื่อสารแบบ UART และสามารถควบคุมพิน input/output ได้ปกติหรือไม่ ด้วยการสร้างโปรแกรมไฟกระพริบบนบอร์ดเพื่อทดสอบว่าบอร์ดสามาถใช้งานได้ และ สร้างโปรแกรมส่งข้อมูลแบบ UART โดยให้ข้อมูลเป็นคำว่า “Project” และสังเกตผลการทดลองจากเครื่องออสซิลโลสโคป จากนั้นนำมาวิเคราะห์ว่าผลลัพธ์ที่ได้เป็นข้อมูลเดียวกันกับที่ส่งหรือไม่ โดยทดลองแสดงดังรูปที่ 3.69



รูปที่ 3.69 วิธีการทดสอบการสื่อสารแบบ UART ผ่านพินอินเตอร์เฟซ

3.3.9 การวัดการทำงานของวงจรจ่ายแรงดันไฟฟ้า 3.3 VEXT ที่ใช้ใน Client Board Public Network (Communication Board)

ทำการวัดการทำงานของวงจรรับแรงดันไฟฟ้าจาก Core Board นำมาใช้ในการป้อนแรงดันไฟฟ้าภายใน Board โดยวัดจากเครื่องออสซิลโลสโคป และทำการอ่านค่าว่าแรงดันไฟฟ้าที่ออกมานั้นถูกต้องตามที่ต้องการจากหน้าจอออสซิลโลสโคป และมีการติดตั้งอุปกรณ์ แสดงดังรูปที่ 3.70



รูปที่ 3.70 การติดตั้งอุปกรณ์ในการวัดแรงดันไฟฟ้า 3.3 VEXT ที่รับมาจาก Core Board

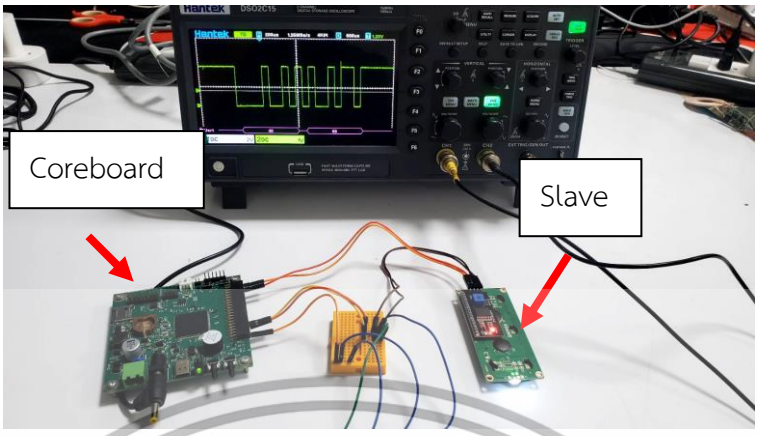
3.3.10 ทดสอบการทำงานของ Alternate function ของ Coreboard

ทดสอบการส่งข้อมูลผ่านพิน PB6 และ PB7 โดยทดสอบการสื่อสารผ่านโปรโตคอล UART และ I2C ซึ่งได้ถูกกำหนด Althernate function ไว้ใน Datasheet ไว้เรียบร้อยแล้วโดยพิน PB6 ใช้ AF6 และ AF7 ในการสื่อสาร I2C1_SCL และ USART1_TX และพิน PB7 ใช้ I2C_SDA และ USART1_RX ตามลำดับ แสดงดังรูปที่ 3.71 จากนั้นใช้เครื่องมือ Osilloscope เพื่อจับสัญญาณการส่งข้อมูลและตรวจสอบความถูกต้องว่าเป็นไปตาม Format frame หรือไม่ โดยมีการทดลองแสดงดังรูปที่ 3.72

Table 15. Alternate function AF0 to AF7(1) (continued)

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
		SYS	LPTIM1/ TIM1/2/16/17	LPTIM3/ PDM_SAI1/ TIM3/4/5/12/15	I3C1/LPTIM2/3/ LPUART1/ OCTOSPI/TIM1/8	CEC/DCMI/ I2C1/2/3/4/ LPTIM1/2/SPI1/ I2S1/TIM15/ **USART1**	CEC/I3C1/LPTIM1/ SPI1/I2S1/SPI2/I2S2/ SPI3/I2S3/SPI4/5/6	I2C4/OCTOSPI/ SAI1/SPI3/I2S3/ SPI4/UART4/12/ USART10/ **USB_PD**	SDMMC1/SPI2/ I2S2/SPI3/I2S3/ SPI6/UART7/8/12 /USART1/2/3/6/ 10/11
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N	-	-	OCTOSPI1_IO1	USART11_CK
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N	-	-	OCTOSPI1_IO0	-
	PB2	RTC_OUT2	-	SAI1_D1	TIM8_CH4N	SPI1_RDY	LPTIM1_CH1	SAI1_SD_A	SPI3_MOSI/ I2S3_SDO
	PB3	JTDO/TRACE SWO	TIM2_CH2	-	-	I2C2_SDA	SPI1_SCK/I2S1_CK	SPI3_SCK/I2S3_ CK	UART12_CTS/ UART12_NSS
	PB4	NJTRST	TIM16_BKIN	TIM3_CH1	OCTOSPI1_CLK	LPTIM1_CH2	SPI1_MISO/I2S1_SDI	SPI3_MISO/I2S3 _SDI	SPI2_NSS/ I2S2_WS
	PB5	-	TIM17_BKIN	TIM3_CH2	OCTOSPI1_NCLK	I2C1_SMBA	SPI1_MOSI/I2S1_SDO	I2C4_SMBA	SPI3_MOSI/ I2S3_SDO
	PB6	-	TIM16_CH1N	TIM4_CH1	I3C1_SCL	I2C1_SCL	HDMI_CEC	I2C4_SCL	USART1_TX
	PB7	-	TIM17_CH1N	TIM4_CH2	I3C1_SDA	I2C1_SDA	-	I2C4_SDA	USART1_RX
	PB8	-	TIM16_CH1	TIM4_CH3	I3C1_SCL	I2C1_SCL	SPI4_RDY	I2C4_SCL	SDMMC1_CKIN
	PB9	-	TIM17_CH1	TIM4_CH4	I3C1_SDA	I2C1_SDA	SPI2_NSS/I2S2_WS	I2C4_SDA	SDMMC1_CDIR

รูปที่ 3.71 Althernate function ของพินที่ใช้ทดสอบ



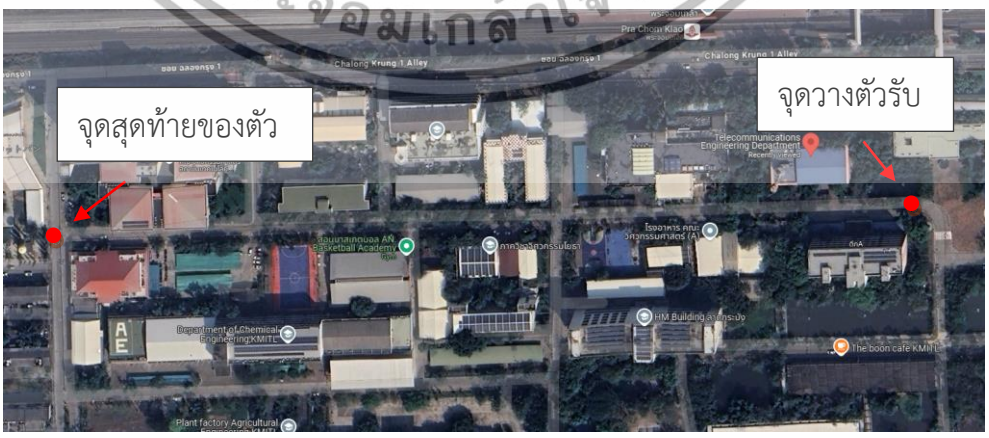
รูปที่ 3.72 การทดลอง Alternate function ของ Coreboard

3.3.11 ทดสอบความถี่ที่ใช้งานสำหรับ LoRa ในโหมดของ LWABP

วัดค่า Bandwidth ที่ใช้งานเมื่อ LoRa ใช้งานจนครบทั้ง 72 ช่องความถี่ ด้วยการส่งข้อมูลทุก 1 วินาที โดยในไฟล์คู่มือการใช้งานได้ระบุเอาไว้ว่าแต่ละช่องความถี่ใช้ Bandwidth อยู่ที่ 125 KHz โดยสังเกต Carrier จากหน้าจอสเปคตรัมอนาไลเซอร์ และ เปิดโหมด Max hold เพื่อสังเกต Bandwidth ที่ใช้ทั้งหมดของ LoRa

3.3.12 ทดสอบการส่งข้อมูลในระยะต่างๆของ LoRa

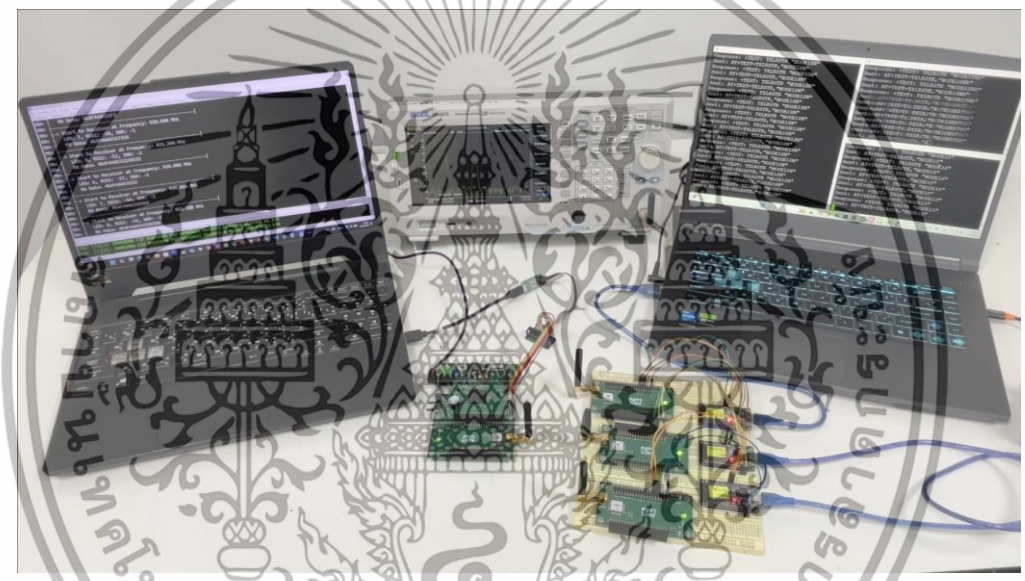
การทดสอบทำในลักษณะ "Line of Sight" และใช้สถานที่ทดสอบจากจุดเริ่มต้นที่ อาคารหอประชุมรวม จนถึง ตึกเรียนรวม 12 ชั้น คณะวิศวกรรมศาสตร์ แสดงดังรูปที่ 3.73 โดยการทดลองจะวางตัวรับข้อมูลไว้ที่บริเวณอาคารหอประชุมรวม และ ตัวส่งจะเพิ่มระยะทางการสื่อสารครั้งละ 50 เมตร ในการทดสอบที่ระยะต่างๆ ทำการส่งข้อมูลจากตัวส่งและฝั่งรับจะรับข้อมูลและแสดงผลค่า RSSI จากนั้นบันทึกข้อมูลระยะละ 10 ครั้ง แล้วนำไปวิเคราะห์ข้อมูลต่อไป



รูปที่ 3.73 สถานที่ทดสอบและระยะทดสอบการสื่อสารของ LoRa

3.3.13 ทดสอบหาค่า Spacing Frequency เพื่อใช้ใน LoRa

การหาระยะห่างความถี่ที่ใช้งาน โดยหาค่าที่เหมาะสมของระยะดังกล่าว เนื่องจากในไทย กสทช ได้กำหนดย่านความถี่การใช้งาน LoRa ในไทยไว้ที่ 920-925 เมกะเฮิรตซ์ (MHz) โดยระยะห่างความถี่ดังกล่าวจะเป็นตัวบอกว่าในระบบจะสามารถมี End node ของ LoRa ได้กี่ตัว โดยทำการทดลองส่งข้อมูลที่ 3 ความถี่ในระยะห่างความถี่ ต่างๆ จากนั้นบันทึกผลที่ได้ในไฟล์ LOG จากนั้นใช้โปรแกรม Mathlab เพื่อตรวจสอบข้อมูลที่ผิดพลาดไปในระยะต่างได้ โดยในการทดลองทดลอง ให้ส่งข้อมูล 5000 ชุด โดยแต่ละชุดจะส่งข้อมูลว่า “NODE1” “NODE2” “NODE3” จากนั้นตรวจสอบความถูกต้องของข้อมูล โดยออกแบบการทดลองแสดงดังรูปที่ 3.74



รูปที่ 3.74 การทดลองหาค่าระยะห่างความถี่ที่เหมาะสมใน LoRa

บทที่ 4

ผลการทดลอง

4.1 ผลการทดสอบการทำงานของอุปกรณ์

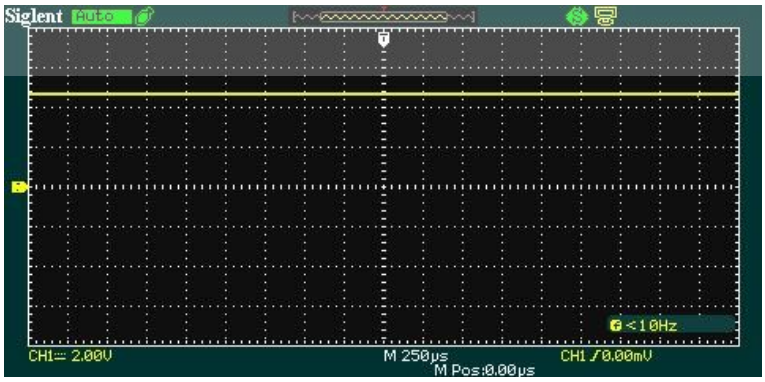
4.1.1 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้า 12 V ภายใน Core Board

จากรูปที่ 4.1 แสดงถึงการวัดผลทดสอบการทำงานวงจรรับแรงดันไฟฟ้า ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่จ่ายแรงดันไฟฟ้าเข้าสู่ Core Board โดยป้อนแรงดันไฟฟ้าที่ 12 V และวัดแรงดันไฟฟ้าได้ 12 V หลังจากผ่านวงจรแปลงแรงดันไฟฟ้า จากการอ่านค่าจากเครื่องออสซิลโลสโคป

รูปที่ 4.1 ผลการทดสอบการทำงานของวงจรรับแรงดันไฟฟ้า 12 V ภายใน Core Board

4.1.2 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้า 5 V ภายใน Core Board

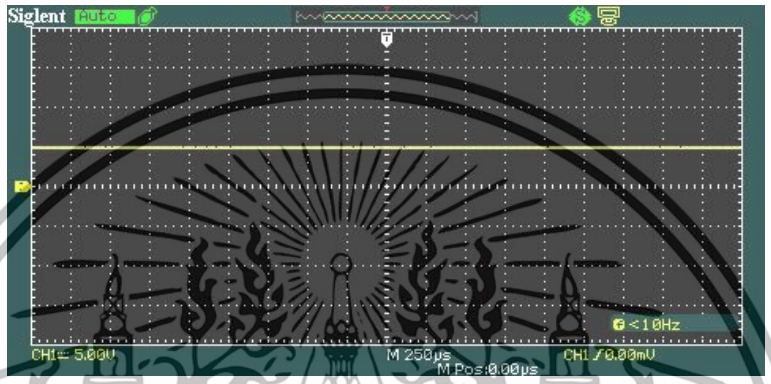
จากรูปที่ 4.2 แสดงถึงการวัดผลทดสอบการทำงานวงจรรับแรงดันไฟฟ้า ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่ผ่านวงจรแปลงแรงดันไฟฟ้าจาก 12 V ให้มีแรงดันไฟฟ้าอยู่ที่ 5 V ที่ใช้ภายใน Core Board จากการอ่านค่าจากเครื่องออสซิลโลสโคป



รูปที่ 4.2 ผลการทดสอบการทำงานของวงจรรับแรงดันไฟฟ้า 5 V ภายใน Core Board

4.1.3 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้า 5 VEXT ที่ใช้จ่ายให้ Client Board

จากรูปที่ 4.3 แสดงถึงการวัดผลทดสอบการทำงานวงจรรับแรงดันไฟฟ้า ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่ผ่านวงจรแปลงแรงดันไฟฟ้าจาก 12 V ให้มีแรงดันไฟฟ้าอยู่ที่ 5 VEXT ที่ที่ใช้จ่ายให้ Client Board จากการอ่านค่าจากเครื่องออสซิลโลสโคป



รูปที่ 4.3 ผลการทดสอบการทำงานของวงจรรับแรงกันไฟฟ้า 5 VEXT ภายใน Core Board

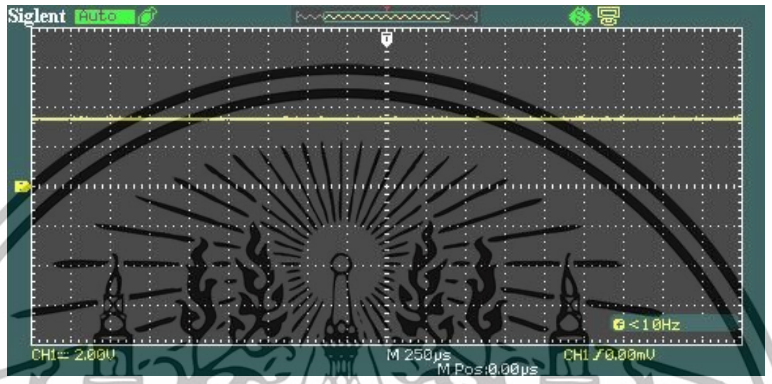
4.1.4 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้า 3.3 V ภายใน Core Board

จากรูปที่ 4.4 แสดงถึงการวัดผลทดสอบการทำงานวงจรรับแรงดันไฟฟ้า ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่ผ่านวงจรแปลงแรงดันไฟฟ้าจาก 5 V ให้มีแรงดันไฟฟ้าอยู่ที่ 3.3 V ที่ใช้ภายใน Core Board จากการอ่านค่าจากเครื่องออสซิลโลสโคป

รูปที่ 4.4 ผลการทดสอบการทำงานของวงจรรับแรงกันไฟฟ้า 3.3 V ภายใน Core Board

4.1.5 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้า 3.3 VEXT ที่ใช้จ่ายให้ Client Board

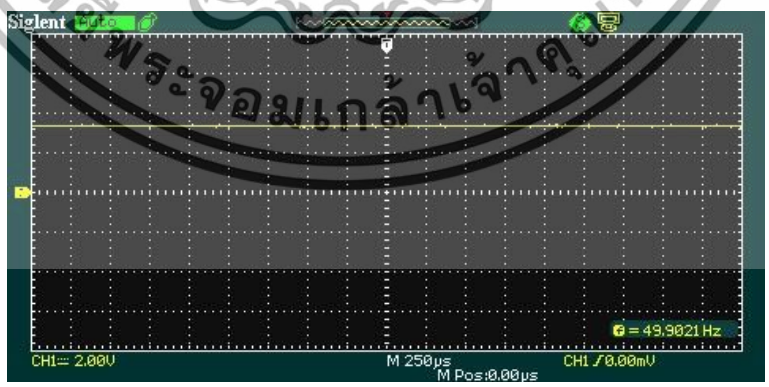
จากรูปที่ 4.5 แสดงถึงการวัดผลทดสอบการทำงานวงจรรับแรงดันไฟฟ้า ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่ผ่านวงจรแปลงแรงดันไฟฟ้าจาก 5 V ให้มีแรงดันไฟฟ้าอยู่ที่ 3.3 VEXT ที่ใช้จ่ายให้ Client Board จากการอ่านค่าจากเครื่องออสซิลโลสโคป



รูปที่ 4.5 ผลการทดสอบการทำงานของวงจรรับแรงกันไฟฟ้า 3.3 VEXT ภายใน Core Board

4.1.6 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้าจาก Core Board มาใช้ใน Client Board Local Network (BLE Board)

จากรูปที่ 4.6 แสดงถึงการวัดผลทดสอบการทำงานกับรับแรงดันไฟฟ้าจาก Core Board มาใช้ใน Client Board Local Network (BLE Board) ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่วัดจาก Connector ที่ใช้จ่ายให้ Client Board จากการอ่านค่าจากเครื่องออสซิลโลสโคป



รูปที่ 4.6 ผลการทดสอบการทำงานในการรับแรงดันไฟฟ้าจาก Local Network (BLE Board)

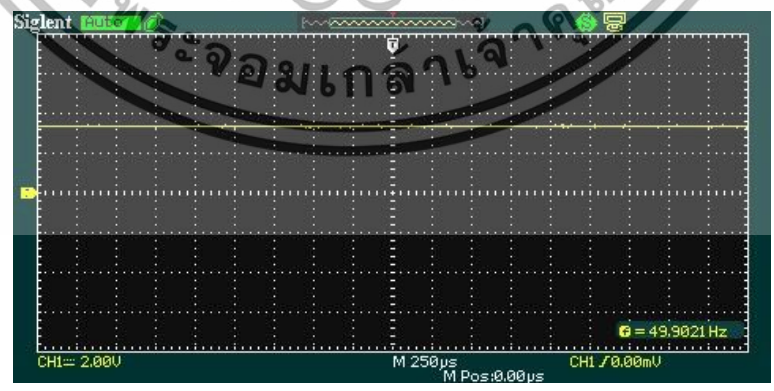
4.1.7 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้าจาก Core Board มาใช้ใน Client Board Local Network (LoRa Board)

จากรูปที่ 4.7 แสดงถึงการวัดผลทดสอบการทำงานกับรับแรงดันไฟฟ้าจาก Core Board มาใช้ใน Client Board Local Network (LoRa Board) ด้วยเครื่องออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่วัดจาก Connector ที่ใช้จ่ายให้ Client Board จากการอ่านค่าจากเครื่องออสซิลโลสโคป

รูปที่ 4.7 ผลการทดสอบการทำงานในการรับแรงดันไฟฟ้าจาก Local Network (LoRa Board)

4.1.8 ผลการทดสอบการทำงานวงจรรับแรงดันไฟฟ้าจาก Core Board มาใช้ใน Client Board Public Network (Communication Board)

จากรูปที่ 4.8 แสดงถึงการวัดผลทดสอบการทำงานกับรับแรงดันไฟฟ้าจาก Core Board มาใช้ใน Client Board Public Network (Communication Board) ด้วยออสซิลโลสโคป จากการวัดแรงดันไฟฟ้าที่วัดจาก Connector ที่ใช้จ่ายให้ Client Board จากการอ่านค่าจากออสซิลโลสโคป

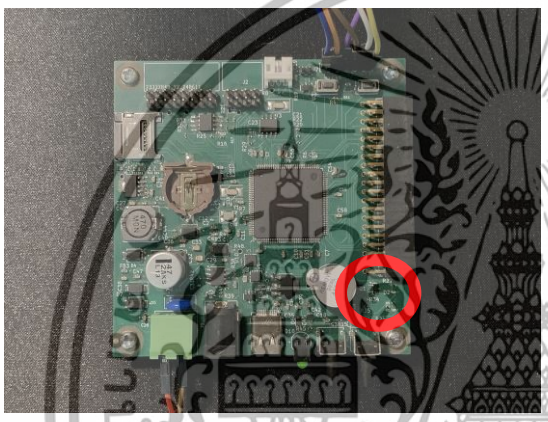
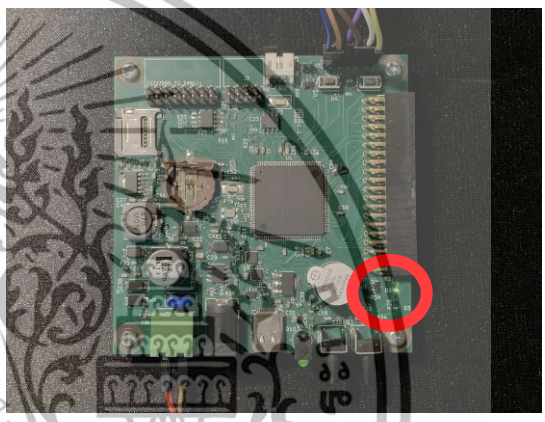


รูปที่ 4.8 ผลการทดสอบการทำงานในการรับแรงดันไฟฟ้าจาก Public Network (Communication Board)

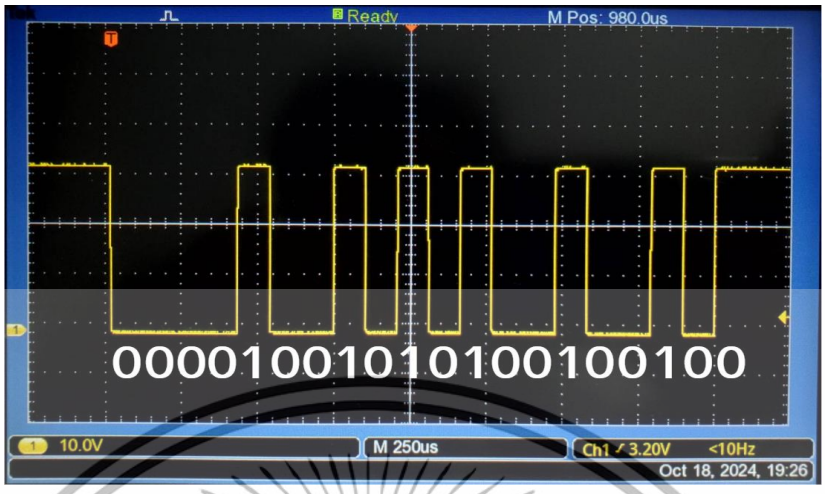
4.2 ผลการทดลองการทำงานของ Coreboard และการสื่อสารแบบ UART ผ่านพินอินเตอร์เฟซ

การทดสอบ Coreboard โดยทดลองควบคุมหลอดไฟ led ที่ถูกติดตั้งบนบอร์ดพบว่ามีการทำงานที่ปกติสามารถใช้งาน input/output ได้ปกติแสดงดังตารางที่ 4.1

ตารางที่ 4.1 ผลการทดสอบ Core Board

หลอดไฟ D2 ดับ	หลอดไฟ D2 ติด
	

การทดสอบการสื่อสารแบบ UART ผ่านพินอินเตอร์เฟซทดสอบอุปกรณ์ Core board โดยส่งจากขา usart3_Tx (PB10) และ usart3_Rx (PD9) โดยเชื่อมเข้ากับคอมพิวเตอร์ผ่าน SEGGER J-LINK EDU MINI หรือหน้าอินเตอร์เฟซ SWD แล้วใช้ west เพื่ออัปโหลดโค้ดในการเขียนคำสั่งส่งข้อมูล ซึ่งทดลองการส่งข้อมูลแบบฐาน 16 และทำการตรวจจับสัญญาณที่ส่งด้วออสซิลโลสโคป เพื่อตรวจสอบว่าข้อมูลที่ส่งมีความถูกต้อง โดยสัญญาณที่รับได้จะขึ้นที่หน้าจอแสดงผลของเครื่องออสซิลโลสโคป โดยทำการอ่านค่าสัญญาณที่ขึ้นบนจอแสดงผล ซึ่งสัญญาณที่แสดงบนจอจะเป็นข้อมูลเลขฐาน 2 แสดงดังรูปที่ 4.9



รูปที่ 4.9 สัญญาณข้อมูลการสื่อสารแบบ UART

จากรูปที่ 4.9 สามารถอ่านข้อมูลได้เป็นเลขฐาน 2 ดังนี้ 00001001010100100101 เมื่อตัดบิต Start และ บิต Stop ออกจาก 00001001010100100101 จะเหลือข้อมูลเป็น 0001001010010010 หลังจากนั้นแบ่งข้อมูลเป็น 2 ชุด ชุดละ 8 บิตจะได้ข้อมูลชุดที่ 1 คือ 00010010 และชุดที่ 2 คือ 10010010 จากนั้นสลับบิตในชุดของตนเองเนื่องจากการส่งข้อมูลของ UART จะทำการส่งข้อมูล MSB ไปก่อน หากต้องการอ่านข้อมูล ต้องสลับบิตก่อนจะได้ผลลัพธ์เป็น ชุดที่ 1 คือ 01001000 และชุดที่ 2 คือ 01001001 จากนั้นแปลงเลขฐานจากฐาน 2 เป็นฐาน 10 และนำค่าที่ได้ไปเทียบกับตารางรหัส ASCII เพื่อดูว่าเป็นตัวอักษรอะไร โดยจะได้ข้อมูลออกมาเป็น ชุดที่ 1 คือ 72 และชุดที่ 2 คือ 73 ซึ่งคือตัวอักษร H และ I ตามลำดับซึ่งตรงกับข้อมูลที่ส่งมา

4.3 ทดสอบการทำงานของ Alternate function ของ Coreboard

การทดสอบ Althernate function โดยใช้อุปกรณ์ Oscilloscope เพื่อตรวจสอบสัญญาณการสื่อสารในโปรโตคอล UART และ I2C ว่ามีการทำงานที่ถูกต้องไหม โดยความเร็วที่ใช้ส่งข้อมูลของ UART อยู่ที่ 9600 bps และ I2C ใช้ความเร็วอยู่ที่ I2C_BITRATE_FAST หรือที่ความเร็ว 400 Kbps ในการทดลองการสื่อสารของ I2C จำเป็นที่จะต้องใช้อุปกรณ์ Slave ที่สื่อสาร I2C ได้ในการทดลองเนื่องจากในโปรโตคอลดังกล่าว หากไม่มีอุปกรณ์ Slave ที่สื่อสาร I2C ได้ จะทำให้ไม่มีสัญญาณการสื่อสาร หรือไม่มีการส่งข้อมูลเกิดขึ้น ซึ่งในการทดลองใช้อุปกรณ์ 1602 LCD Module ในการเชื่อมต่อไปที่ Coreboard โดยสรุปผลการทดการทดลองได้เป็น

4.3.2 การทดลองส่งข้อมูลแบบ UART

จากการทดลองเพื่อศึกษาการส่งข้อมูลผ่านโปรโตคอล UART พบว่าข้อมูลบิตที่ได้รับคือ 00001001010100101101 ซึ่งจัดเก็บในรูปแบบเฟรมข้อมูลของ UART โดยในเฟรมข้อมูลนี้จะมีบิตเริ่มต้น (Start Bit) และบิตสิ้นสุด (Stop Bit) ที่ตำแหน่งบิตที่ 1, 10, 11 และ 20 ตามลำดับ เพื่อที่จะวิเคราะห์ข้อมูล จึงได้ทำการตัดบิตเหล่านี้ออก ทำให้เหลือเฉพาะข้อมูลที่แท้จริง ซึ่งได้เป็นบิต 0001001010010110 เมื่อตัดบิตเริ่มต้นและบิตสิ้นสุดออกแล้วจึงจัดการแบ่งข้อมูลที่เหลือออกเป็นสองชุด ชุดละ 8 บิต โดยข้อมูลที่ได้คือชุดแรก 00010010 และชุดที่สอง 10010110 หลังจากนั้นเพื่อให้สามารถอ่านค่าบิตเหล่านี้ได้อย่างถูกต้อง จึงได้ทำการสลับตำแหน่งบิตของข้อมูลแต่ละชุด ผลลัพธ์ที่ได้คือ ชุดข้อมูลแรกหลังการสลับบิตมีค่าเป็น 01001000 ซึ่งเท่ากับ 0x48 และชุดข้อมูลที่สองมีค่าเป็น 01101001 ซึ่งเท่ากับ 0x69 เมื่อทำการเทียบข้อมูลดังกล่าวกับตารางรหัส ASCII พบว่าข้อมูล 0x48 และ 0x69 ตรงกับตัวอักษร H และ i ตามลำดับ ซึ่งสามารถสรุปได้ว่าการส่งข้อมูลในครั้งนี้ประกอบด้วยคำว่า "Hi" แสดงดังรูปที่ 4.10

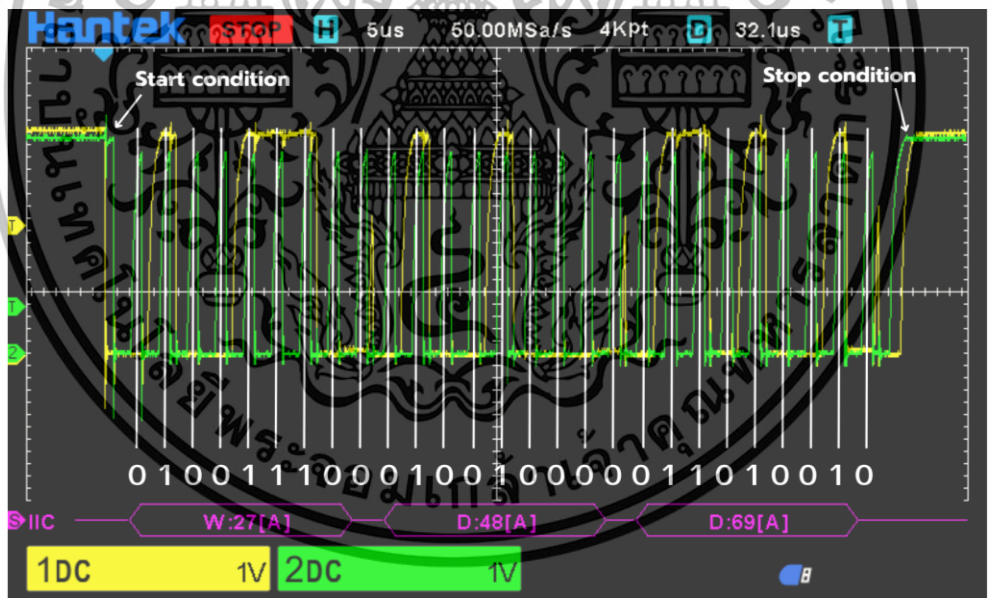
รูปที่ 4.10 ผลการทดลองส่งข้อมูลแบบ UART ผ่าน Althernate function ที่ AF7

4.3.3 การทดสอบส่งข้อมูลแบบ I2C

เมื่อทำการอ่านบิตข้อมูลพบว่าได้ข้อมูล 01001110001001000011010010 ซึ่งเมื่อตีความข้อมูลนี้ในรูปแบบ Frame format ของ I2C จะพบว่าใน 7 บิตแรกคือ 0100111 ซึ่งแสดงถึงที่อยู่ของ Slave ที่รับข้อมูล โดยมีค่าเป็น 0x27 ในระบบเลขฐาน 16 และบิตที่ 8 คือ เขียน/อ่าน โดยเป็นบิต 0 นั้นหมายถึง เขียน ส่วนบิตที่ 9 เป็นบิต ACK ที่มีค่าเป็น 0 ซึ่งเป็นการตอบกลับจาก Slave ไปยัง Coreboard ว่าได้รับข้อมูลเรียบร้อยแล้ว

ถัดไปในบิตที่ 10 ถึงบิตที่ 17 จะเป็นข้อมูลของตัวอักษรตัวแรก คือ 001001000 ซึ่งมีค่าเป็น 0x48 ตามระบบเลขฐาน 16 จากนั้นในบิตที่ 18 จะเป็นบิต ACK ที่มีค่า 0 อีกครั้ง ซึ่งเป็นการตอบรับว่าข้อมูลได้ถูกส่งและได้รับอย่างสมบูรณ์ ต่อมาในบิตที่ 19 ถึงบิตที่ 26 แสดงถึงข้อมูลตัวอักษรตัวที่สอง ซึ่งมีค่าเป็น 01101001 หรือ 0x69 ในเลขฐาน 16 และในบิตที่ 27 เป็นบิต ACK ที่มีค่า 0 เช่นกัน เป็นการยืนยันการรับข้อมูลจาก Slave ไปยัง Master

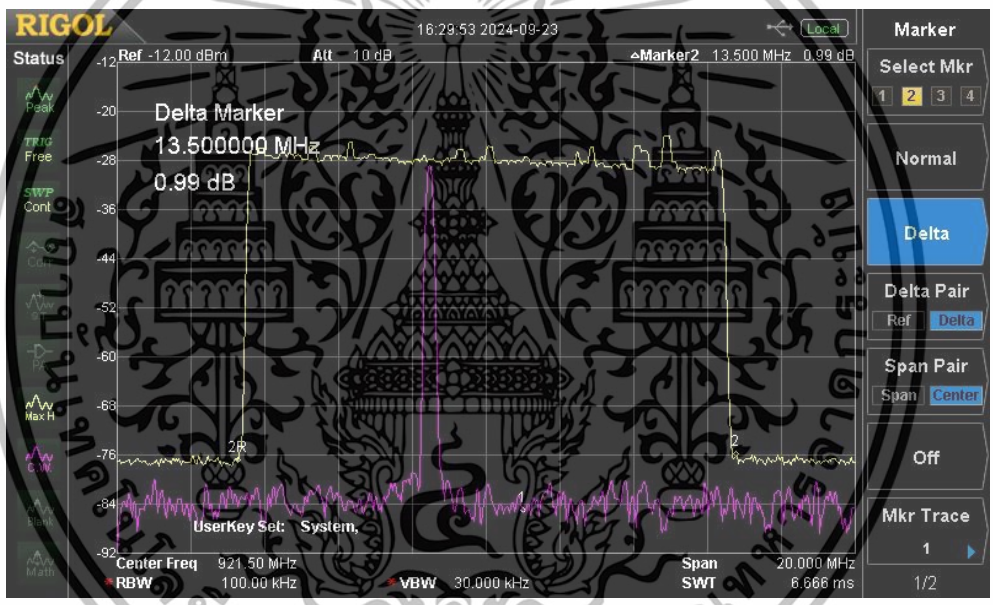
โดยสรุป ข้อมูลทั้งสองชุดคือ 0x48 และ 0x69 เมื่อนำไปเทียบกับตาราง ASCII จะพบว่าข้อมูลเหล่านี้แทนตัวอักษร H และ i ซึ่งสื่อถึงการส่งข้อมูลตัวอักษรคำว่า “Hi” แสดงดังรูปที่ 4.11



รูปที่ 4.11 ผลการทดลองส่งข้อมูลแบบ I2C ผ่าน Althernate function ที่ AF4

4.4 ทดสอบความถี่ที่ใช้งานสำหรับ LoRa ในโหมดของ LWABP

การทดสอบใช้เครื่องมือ Spectrum analyzer เพื่อดูคุณภาพสัญญาณที่ได้จากบอร์ด และความถี่ที่ใช้ในการสื่อสารโดยในบอร์ด LoRa พบว่าใช้ความถี่ที่ในช่วง 915.2 MHz – 922.3 MHz โดยจะแบ่งการใช้งานออกเป็น 72 ช่องสัญญาณ โดยในแต่ละช่องสัญญาณจะใช้ Data rate อยู่ที่ Spreading Factor = 12 และมี Bandwidth = 125 KHz โดยจากรูปที่ 4.12 พบว่าเมื่อทดลองให้ LoRa ส่งข้อมูลซ้ำๆ ตัวบอร์ดจะเลือกใช้ความถี่ที่ไม่ซ้ำกันในแต่ละรอบ โดยจะเลือกใช้จากช่องสัญญาณที่ใช้งานได้ เนื่องจากตัวบอร์ดทำงานในโหมดของ LWABP (LoRaWAN Activation by Personalization) เพื่อป้องกันการชนกันของความถี่ในกรณีที่มีการส่งข้อมูลพร้อมกันมาที่ Core Board และจะพบว่าช่องสัญญาณทั้งหมด ใช้ขนาด Bandwidth = 13.50 MHz



รูปที่ 4.12 สัญญาณที่ได้ Carrier ที่มาจากบอร์ดจากการทดสอบบอร์ด LoRa Board Plugin

4.5 ทดสอบการส่งข้อมูลในระยะต่างๆของ LoRa

ในการทดสอบทดลองการส่งข้อมูลพบว่า LoRa E5 HF ที่ใช้ความถี่ที่ 915.2 MHz มีระยะการส่งที่สูงกว่า 500 เมตร แต่ด้วยความจำกัดของพื้นที่ทดสอบ จึงไม่สามารถทดสอบระยะสูงสุดที่ LoRa สื่อสารได้ โดยกราฟข้อมูลผลการทดลองเป็นไปตามดังตารางที่ 4.2 และนำข้อมูลที่ได้ มาพล็อตกราฟได้ดังรูปที่ 4.13

ตารางที่ 4.2 ผลลัพธ์ในการทดสอบค่า RSSI ที่ระยะต่างๆ

Distance (metre)	RSSI (dbm)
0	-14
0	-13
0	-15
0	-12
0	-13
0	-13
0	-15
0	-12
0	-15
0	-14
50	-70
50	-72
50	-70
50	-71
50	-75
50	-75
50	-72
50	-70
50	-72
50	-70
100	-87
100	-85
100	-85
100	-84
100	-84
100	-83

ตารางที่ 4.2 (ต่อ) ผลลัพธ์ในการทดสอบค่า RSSI ที่ระยะต่างๆ

Distance (metre)	RSSI (dbm)
100	-87
100	-88
100	-87
100	-88
150	-90
150	-92
150	-89
150	-90
150	-93
150	-92
150	-89
150	-88
150	-89
150	-90
200	-93
200	-93
200	-93
200	-94
200	-96
200	-97
200	-96
200	-93
200	-93
200	-94
250	-94

ตารางที่ 4.2 (ต่อ) ผลลัพธ์ในการทดสอบค่า RSSI ที่ระยะต่างๆ

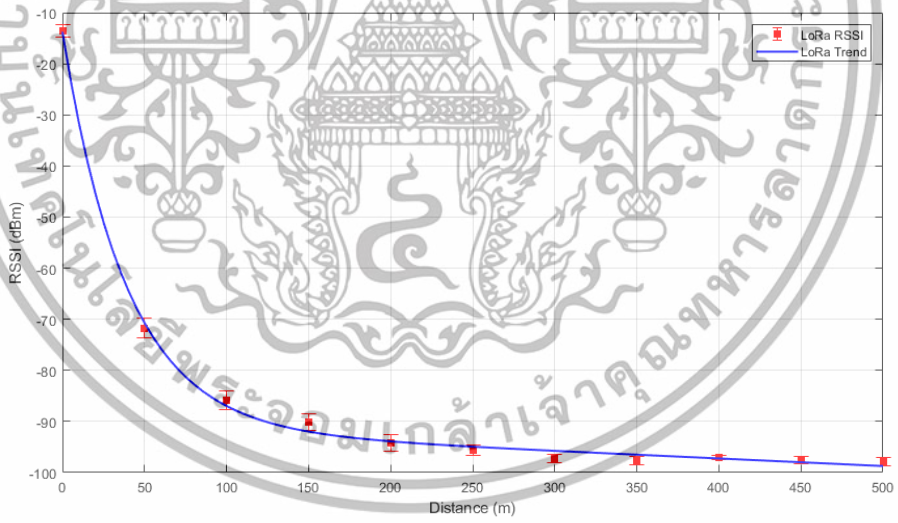
Distance (metre)	RSSI (dbm)
250	-95
250	-95
250	-95
250	-96
250	-96
250	-96
250	-95
250	-96
250	-96
250	-98
300	-96
300	-98
300	-97
300	-98
300	-97
300	-98
300	-98
300	-96
300	-97
300	-97
300	-98
350	-98
350	-98
350	-98
350	-97
350	-98

ตารางที่ 4.2 (ต่อ) ผลลัพธ์ในการทดสอบค่า RSSI ที่ระยะต่างๆ

Distance (metre)	RSSI (dbm)
350	-97
350	-96
350	-98
350	-98
350	-99
350	-98
400	-97
400	-97
400	-97
400	-96
400	-97
400	-97
400	-97
400	-98
400	-97
400	-97
400	-97
450	-97
450	-98
450	-97
450	-97
450	-98
450	-99
450	-97
450	-97
450	-97

ตารางที่ 4.2 (ต่อ) ผลลัพธ์ในการทดสอบค่า RSSI ที่ระยะต่างๆ

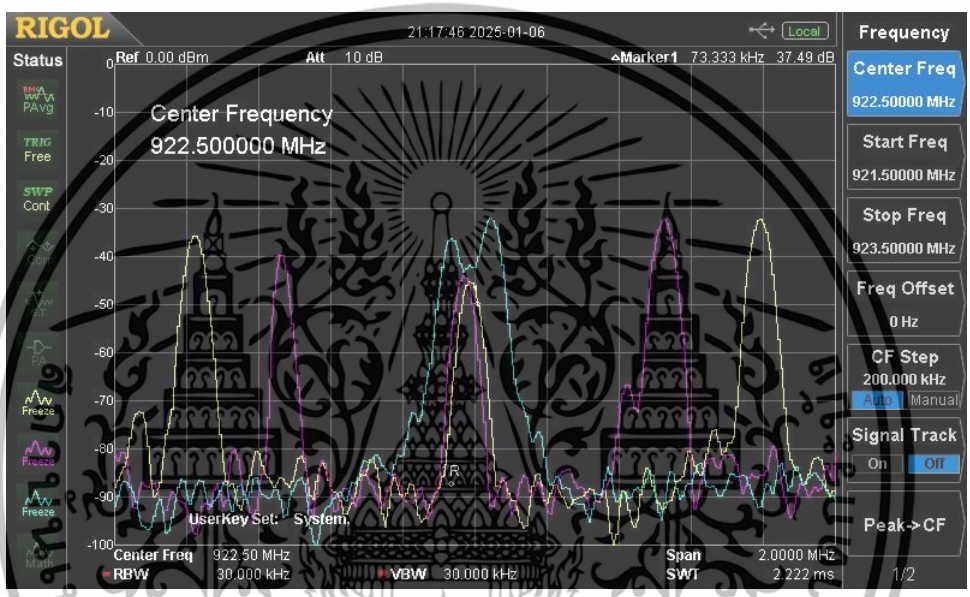
Distance (metre)	RSSI (dbm)
450	-98
500	-98
500	-98
500	-97
500	-97
500	-98
500	-99
500	-97
500	-99
500	-97
500	-98



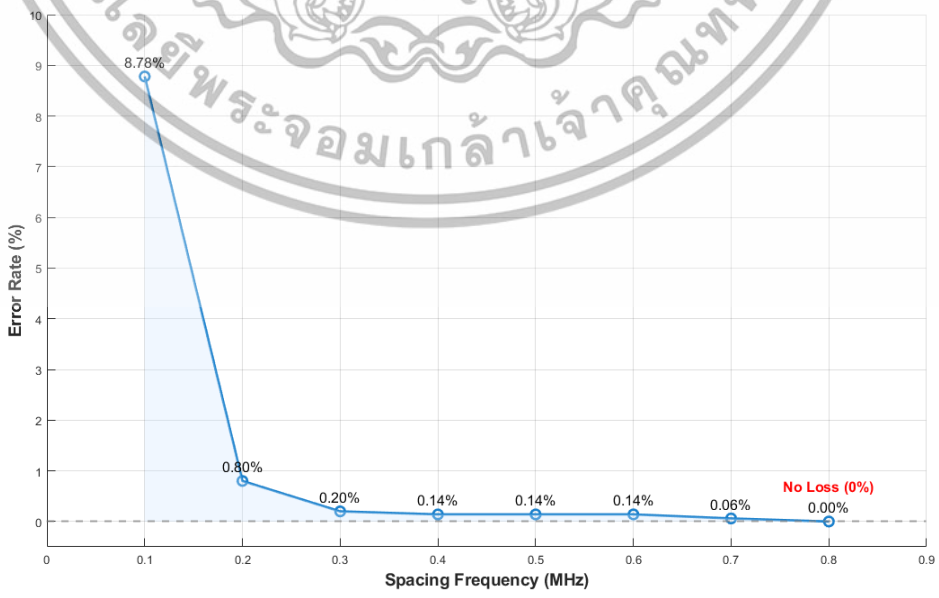
รูปที่ 4.13 กราฟแสดงความสัมพันธ์ระหว่างระยะทางและค่า RSSI ของ LoRa E5 HF

4.6 ผลทดสอบหาค่า Spacing Frequency เพื่อใช้ใน LoRa

จากการทดลองหาระยะห่างความถี่ที่เหมาะสมสำหรับการสื่อสารของ LoRa พบว่า เมื่อระยะห่างความถี่อยู่ที่ 1 MHz รูปสัญญาณในโดเมนความถี่ จะเห็นได้ว่า มีลักษณะที่ Bandwidth ทับกัน แสดงดังรูปที่ 4.14 และหลังจากเพิ่มระยะห่างความถี่ Bandwidth มีลักษณะห่างออกจากกันทำให้ข้อมูลที่รับได้ มีความถูกต้องมากขึ้น แสดงดังรูปที่ 4.15 แต่ต้องแลกมากับจำนวน End node ที่ใช้ได้บน LoRa



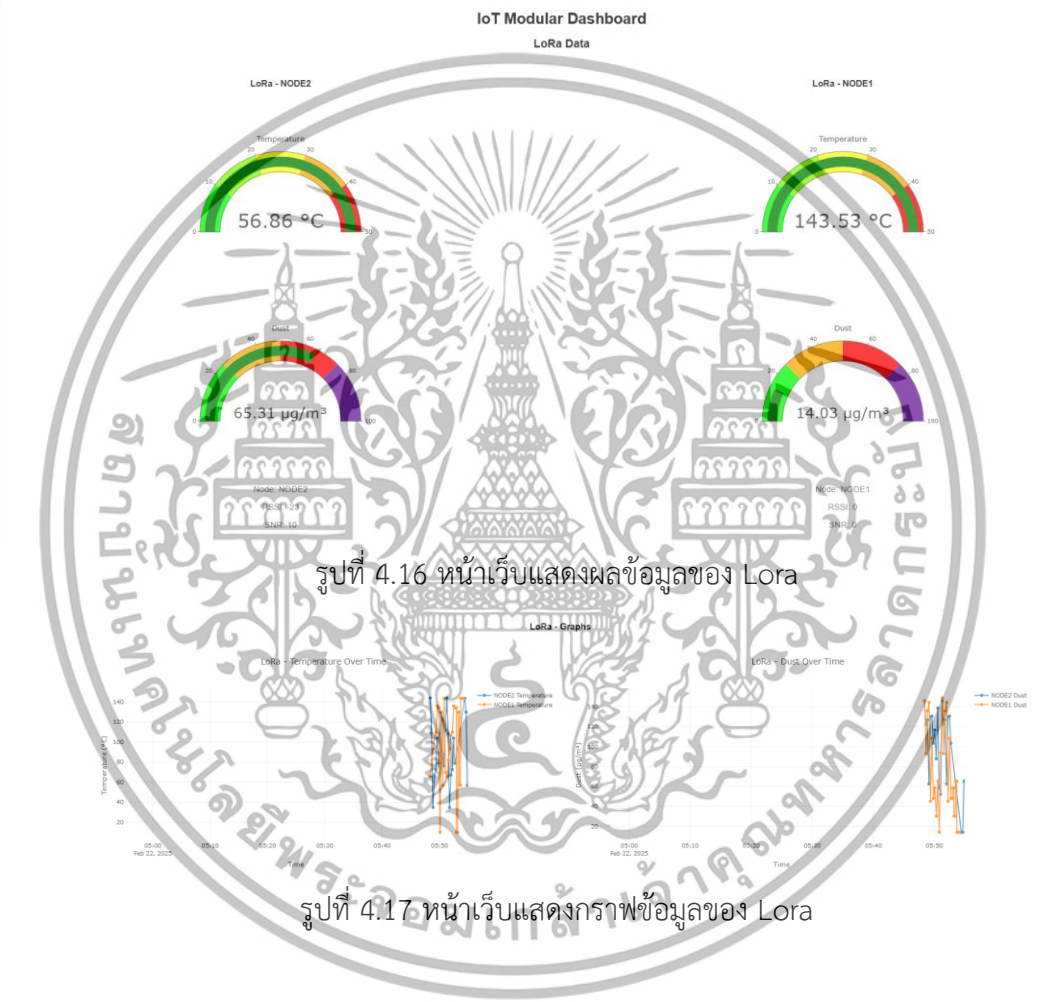
รูปที่ 4.14 ระยะห่างความถี่ในโดเมนความถี่ที่ใช้งานบน LoRa



รูปที่ 4.15 กราฟความสัมพันธ์ระหว่างระยะห่างความถี่ และ ความถูกต้องของข้อมูลที่รับได้บน LoRa

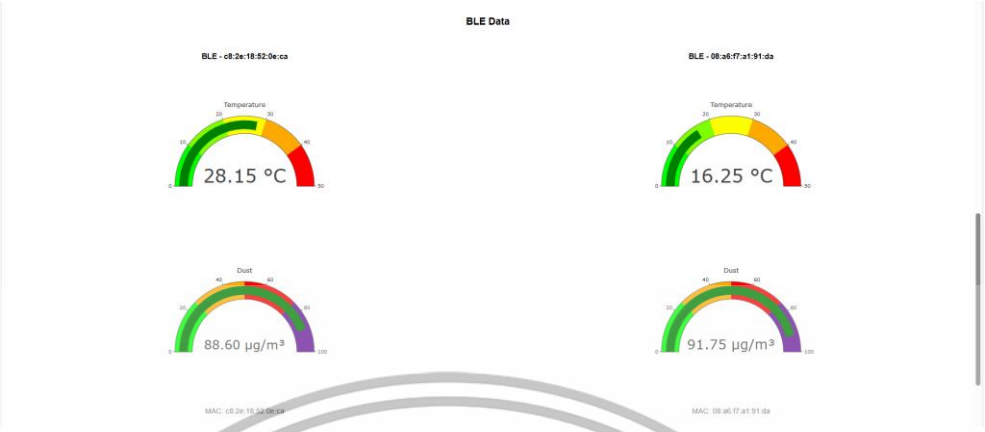
4.7 ผลการออกแบบหน้าเว็บแสดงผล

หน้าเว็บแสดงผล IoT Modular Dashboard นี้มีไว้เพื่อให้ผู้ใช้ติดตามข้อมูลจากเซนเซอร์หลายตัวพร้อมกัน ซึ่งเหมาะสำหรับการตรวจสอบระบบ IoT เช่น การตรวจวัดอุณหภูมิ, ความชื้น, ความดัน หรือข้อมูลอื่น ๆ ที่ต้องการการติดตามแบบต่อเนื่อง โดยหน้าจอจะทำให้ผู้ใช้เข้าใจการทำงานของระบบได้อย่างง่าย



รูปที่ 4.16 หน้าเว็บแสดงผลข้อมูลของ Lora

รูปที่ 4.17 หน้าเว็บแสดงกราฟข้อมูลของ Lora



รูปที่ 4.18 หน้าเว็บแสดงผลข้อมูลของ BLE

รูปที่ 4.19 หน้าเว็บแสดงกราฟข้อมูลของ BLE

จากรูปที่ 4.16-4.19 แสดงการออกแบบ IoT Modular Dashboard สำหรับการแสดงข้อมูลเซนเซอร์ ซึ่งเป็นส่วนหนึ่งของระบบ IoT ที่รวบรวมข้อมูลจากเซนเซอร์หลายตัวและแสดงผลในรูปแบบที่เข้าใจง่าย โดยสามารถอธิบายการออกแบบได้ดังนี้ :

4.7.1 Frontend

เป็นหน้าจอของ IoT Modular Dashboard ซึ่งถูกออกแบบมาเพื่อแสดงข้อมูลเรียลไทม์ของตัวเซนเซอร์หลักสองประเภท ได้แก่ อุณหภูมิ (Temperature) และฝุ่น (Dust) โดยจะแบ่งข้อมูลออกเป็น 2 ชุด คือ 1.Lora ดังรูปที่ 9 2.BLE ดังรูปที่ 10 ข้อมูลของ Lora และ BLE จะมีทั้งหมด 4 โหนด ที่ทำการเชื่อมต่ออยู่กับระบบ โดยมีส่วนประกอบสำคัญและรายละเอียดดังนี้ :

- ส่วนบนสุด : ส่วนบนสุดของหน้าเว็บจะแสดงชื่อของแอปพลิเคชัน (“IoT Modular Dashboard”) เพื่อบ่งบอกว่าเป็นหน้าควบคุมและแสดงข้อมูล IoT
- Temperature Gauge : จะแสดงค่าอุณหภูมิที่วัดได้ในหน่วยองศาเซลเซียส (°C) ผ่านรูปแบบเกจวัดวงกลมที่มีการไล่สีจากเขียวไปจนถึงแดง บ่งบอกระดับอุณหภูมิที่เพิ่มขึ้น
- Dust Gauge : จะแสดงค่าฝุ่นละอองในหน่วยไมโครกรัมต่อลูกบาศก์เมตร (µg/m³) ด้วยรูปแบบเกจวัดคล้ายกัน หากค่าสูงขึ้นก็จะมีการไล่สีเป็นเฉดสีเขียวไปจนถึงม่วง เพื่อสะท้อนระดับความเข้มข้นของฝุ่น

- ค่าคุณภาพสัญญาณ (SNR) , ความแรงของสัญญาณ (RSSI) และ Node ของเซนเซอร์ : ใต้เกจวัดของ Lora จะมีการแสดงค่า SNR , RSSI และที่อยู่ของเซนเซอร์ (Node) เพื่อให้สามารถตรวจสอบแหล่งที่มาของข้อมูล รวมถึงคุณภาพของสัญญาณที่ได้รับ

- MAC Address : จะมีเฉพาะในส่วนของ BLE เป็น หมายเลขเฉพาะของฮาร์ดแวร์เครือข่ายที่ใช้ระบุอุปกรณ์ในเครือข่ายที่เชื่อมต่อกับเครือข่ายผ่าน Ethernet หรือ Wi-Fi

- กราฟแสดงข้อมูลเรียลไทม์ (Real-Time Graph) : Real-Time Temperature Monitoring ใต้ข้อมูล Temperature Gauge จะแสดงกราฟเส้นที่พล็อตข้อมูลอุณหภูมิที่ได้รับมาในแต่ละช่วงเวลาแบบเรียลไทม์ ผู้ใช้สามารถสังเกตการเปลี่ยนแปลงของอุณหภูมิในช่วงเวลาต่าง ๆ ได้อย่างชัดเจน Real-Time Dust Monitoring ในทำนองเดียวกัน ใต้ Dust Gauge จะมีกราฟแสดงค่าฝุ่นละอองในหน่วย μg/m³ แบบเรียลไทม์ ทำให้สามารถติดตามแนวโน้มระดับฝุ่นในช่วงเวลาหนึ่งได้

4.7.2 Backend

ได้ทำการนำค่าเซนเซอร์ที่ส่งจาก core board ผ่านเข้า Broker EMQX หลังจากนั้นทำการดึงข้อมูลจาก Broker ผ่าน Backend ของระบบ IoT Modular Dashboard ดังรูปที่ 4.20 ทำหน้าที่เป็นตัวกลางในการเชื่อมต่อและประมวลผลข้อมูลจากเซ็นเซอร์ IoT ที่ส่งผ่านโปรโตคอล MQTT ก่อนที่จะส่งข้อมูลไปยัง Frontend ผ่าน WebSocket เพื่อนำเสนอในรูปแบบของเกจวัด (Gauge) และกราฟเรียลไทม์ (Real-Time Graph) ในหน้าเว็บ โดย Backend จะรับผิดชอบด้านการเชื่อมต่อกับ MQTT Broker, การ Subscribe หัวข้อ (Topics) ของเซนเซอร์, การแปลงข้อมูล, และการส่งต่อข้อมูลไปยังผู้ใช้ที่เข้าถึงผ่านเบราว์เซอร์

รูปที่ 4.20 ตัวอย่างข้อมูลที่ส่งมาจาก Broker ผ่าน Backend

4.8 ผลการเปรียบเทียบค่าคุณสมบัติของบอร์ดที่ออกแบบ

4.8.1 ผลการเปรียบเทียบค่าคุณสมบัติของ LoRa Gateway ที่ออกแบบกับ LoRa Gateway ที่มีอยู่ทั่วไป

จากตารางที่ 4.3 แสดงการเปรียบเทียบระหว่าง LoRa Gateway ที่ใช้ในปริญญานิพนธ์นี้และ LoRa Gateway ที่มีอยู่ทั่วไป โดยในช่องสุดท้ายจะเป็นการแสดงข้อมูลของ IoT Modular (LoRa Gateway) ซึ่งพัฒนาในปริญญานิพนธ์นี้ โดยเปรียบเทียบกับ Dragino LG01-P LoRa Gateway และ Dragino LG01-P ที่มีอยู่ทั่วไป เพื่อให้เห็นถึงความแตกต่างในด้านสเปคและฟังก์ชันการทำงานที่ได้รับการปรับปรุงและพัฒนาขึ้นในปริญญานิพนธ์นี้ [25] [26]

ตารางที่ 4.3 เปรียบเทียบระหว่าง LoRa Gateway ที่ใช้ในปริญญานิพนธ์นี้และ LoRa Gateway ที่มีอยู่ทั่วไป

	Dragino LG01-P LoRa Gateway	Dragino LG01-P	IoT Modular (LoRa Gateway)
Processor	400 MHz	400 MHz	250 MHz
MCU	ATMega328P	ATMega328P	Arm Cortex-M33
Flash	32 KB	32 KB	2 MB
Dynamic Range	127 dB	-	158 dB
LoRa Chip	SX1272	SX2176/78	SX126X
Frequency Range	868 MHz and 915 MHz	433 MHz / 868 MHz / 915 MHz	920 – 925 MHz With 9 End node Channel
RJ45 Port	2(WAN and LAN)	2(WAN and LAN)	-
WiFi	IEEE 802.11 b/g/n	IEEE 802.11 b/g/n	IEEE 802.11 b/g/n
Power Input	12V DC	12V DC	12V Jack 3.5 DC

4.8.2 ผลการเปรียบเทียบค่าคุณสมบัติของ BLE Gateway ที่ออกแบบกับ BLE Gateway ที่มีอยู่ทั่วไป

จากตารางที่ 4.4 แสดงการเปรียบเทียบระหว่าง BLE Gateway ที่ใช้ในปริญญานิพนธ์นี้ และ BLE Gateway ที่มีอยู่ในตลาดทั่วไป โดยในช่องสุดท้ายจะเป็นการแสดงข้อมูลของ IoT Modular (BLE Gateway) ซึ่งพัฒนาในปริญญานิพนธ์นี้ เปรียบเทียบกับ DSGW-094 Nordic Bluetooth Cellular Gateway และ DSGW-092 ESP32 Gateway ที่มีอยู่ในตลาดทั่วไป เพื่อให้เห็นถึงความแตกต่างในด้านสเปคและฟังก์ชันการทำงานที่ได้รับการปรับปรุงและพัฒนาขึ้นในปริญญานิพนธ์นี้ [27] [28] [29]

ตารางที่ 4.4 เปรียบเทียบระหว่าง BLE Gateway ที่ใช้ในปริญญานิพนธ์นี้และ BLE Gateway ที่มีอยู่ทั่วไป

	DSGW-094 Nordic Bluetooth Cellular Gateway	DSGW-092 ESP32 Gateway	IoT Modular (BLE Gateway)
Processor	Nordic nRF52840	ESP32	Arm Cortex-M33
RAM	256 KB	520 KB	
Flash	1 MB	4 MB	2 MB
Systems	FreeRTOS	FreeRTOS	Zephyr (RTOS)
Tx Power	8 dBm	19.5 dBm	158 dB
Range Communication	150 Meter	150 Meter	60 Meter
Frequency	2.4 GHz	2.4 GHz	2.4 GHz
Bandwidth	2 MHz	2 MHz	2 MHz
WiFi	IEEE 802.11 b/g/n	IEEE 802.11 b/g/n	IEEE 802.11 b/g/n
Power Input	5V/2A type-C	5V/2A type-C	12V Jack 3.5 DC

4.9 ผลการทดลองวัดกำลังส่งสัญญาณจากโมดูล LoRa E5 HF

ในการทดลองนี้ได้ทำการวัดกำลังส่งสัญญาณจากโมดูล LoRa E5 HF โดยใช้เครื่องมือ Spectrum Analyzer เพื่อประเมินคุณภาพของสัญญาณที่ส่งออกจากโมดูลดังกล่าว ในการวัดนั้นได้ใช้สายเชื่อมต่อเสาสัญญาณหัวต่อ SMA Type ซึ่งเชื่อมต่อกับโมดูล และปลายสายเชื่อมต่อกับช่องเชื่อมต่อสัญญาณบนเครื่อง Spectrum Analyzer เพื่อวัดกำลังของสัญญาณที่ส่งออกมา การควบคุมกำลังส่งสัญญาณของโมดูล LoRa E5 HF ถูกทำผ่าน AT Command ด้วยคำสั่ง AT+TEST=RFCFG,<frequency>,<sf>,<bandwidth>,<txpr>,<rxpr>,<pow>,<crc>,<iq>,<net> โดยผู้ทดลองได้กำหนดค่าต่างๆ ดังนี้

- frequency = 922 MHz
- sf (Spreading Factor) = 9
- bandwidth = 125 KHz

ในส่วนของพารามิเตอร์ txpr ที่กำหนดกำลังส่งในค่าต่างๆ ตามที่โมดูลสามารถรองรับได้ โดยวัดกำลังส่งที่ได้จากการตั้งค่าดังกล่าวเพื่อตรวจสอบคุณภาพและประสิทธิภาพของการส่งสัญญาณผ่านเครื่องมือ Spectrum Analyzer โดยผลการทดลองสามารถสรุปได้ดังตารางที่ 4.5

ตารางที่ 4.5 ผลการทดลองวัดกำลังส่งสัญญาณจากโมดูล LoRa E5 HF

กำลังส่งที่กำหนด (dBm)	กำลังส่งที่ได้จากการวัด (dBm)
4	1.45
5	2.28
6	3.31
7	4.42
8	5.41
9	6.43
10	7.34
11	8.24
12	9.13
13	10.12
14	10.97
15	11.81
16	12.98
17	14.24

ตารางที่ 4.5 (ต่อ) ผลการทดลองวัดกำลังส่งสัญญาณจากโมดูล LoRa E5 HF

กำลังส่งที่กำหนด (dBm)	กำลังส่งที่ได้จากการวัด (dBm)
18	15.51
19	16.73
20	17.87
21	18.93
22	19.71

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

ปริญญานิพนธ์นี้นำเสนอการออกแบบและพัฒนาเฟิร์มแวร์โดยใช้ Zephyr OS ซึ่งเป็นระบบปฏิบัติการแบบเปิดที่ใช้งานได้ฟรีและมีคุณสมบัติ RTOS (Real-Time Operating System) อย่างเต็มรูปแบบ โดยเฟิร์มแวร์นี้ถูกพัฒนาขึ้นให้เป็นไปตามมาตรฐานอุตสาหกรรมสำหรับการพัฒนาเฟิร์มแวร์สำหรับไมโครคอนโทรลเลอร์ การทำงานของ Gateway บน Core board สามารถทำงานได้ตรงตามคุณสมบัติหลักของ Core board ที่ควรจะเป็นได้ เช่น การจัดข้อมูลที่มาจาก End node หลายตัว การประมวลผลข้อมูลที่เข้ามาใน Core board ก่อนที่จะส่งค่าไปที่ MQTT Broker เพื่อนำไปใช้งานอื่นได้ เช่น การทำ Website Monitoring หรือการนำข้อมูลไปวิเคราะห์ แบบทันที และ Core board สามารถรองรับอุปกรณ์อื่นๆได้ เนื่องด้วย Interface 40 pin เหลือช่องสำหรับการสื่อสารอื่นๆ อีกได้ในอนาคต

5.2 ข้อเสนอแนะ

ในปริญญานิพนธ์นี้การออกแบบ Coreboard ควรเพิ่มส่วน UART Debug เพื่อช่วยในการพัฒนาเฟิร์มแวร์และตรวจสอบสถานะการทำงานของโปรแกรมในไมโครคอนโทรลเลอร์ โดยสามารถตรวจสอบการทำงานผ่าน SWD โดยใช้เครื่องมือ Segger EDU Mini หรืออุปกรณ์ที่รองรับการตรวจสอบสถานะได้ การใช้โมดูล LoRa E5 HF ซึ่งรองรับการรับส่งข้อมูลในระยะไกลเป็นตัวเลือกที่ดีสำหรับการทำงานในลักษณะ End node โดยไม่จำเป็นต้องใช้เครือข่ายประเภทอื่น ซึ่งช่วยเพิ่มความยืดหยุ่นในการใช้งานในสภาพแวดล้อมที่ข้อจำกัดในการเชื่อมต่อเครือข่ายมีความสูง

การเลือกใช้ Bluetooth 2.0 (HC-05) ควรพิจารณาการใช้ AT COMMAND ในการควบคุมการเชื่อมต่อและตรวจสอบ RSSI ซึ่งเป็นค่าที่ใช้วัดความแรงของสัญญาณ เพื่อให้สามารถตรวจสอบสถานะการเชื่อมต่อได้แม่นยำและปรับปรุงประสิทธิภาพในการรับส่งข้อมูลระหว่างอุปกรณ์ต่างๆ การตรวจสอบ RSSI จะช่วยให้การประเมินความเสถียรของสัญญาณสามารถทำได้แม่นยำและลดปัญหาที่อาจเกิดจากการเชื่อมต่อที่ไม่เสถียร ในส่วนของการพัฒนาเฟิร์มแวร์บน STM32H563Z ควรมีการจัดการหน่วยความจำที่เหมาะสม โดยเฉพาะในส่วนของ Array ที่ใช้ในการเก็บข้อมูลใน Buffer หากพื้นที่ใน Buffer ไม่เพียงพอ อาจเกิดปัญหา Buffer Overflow ซึ่งจะทำให้โปรแกรมทำงานผิดพลาดได้ ดังนั้น ควรใช้ Null-Terminated String ในการควบคุมการจัดสรรพื้นที่ใน Memory เพื่อให้การจัดการข้อมูลใน Buffer เป็นไปอย่างถูกต้องและไม่เกิดข้อผิดพลาด เช่น การใช้

คำสั่ง strlen() เพื่อควบคุมความยาวของข้อมูลใน Buffer ให้เหมาะสมและลดความเสี่ยงในการเกิดข้อผิดพลาดในการทำงานของเฟิร์มแวร์

การวัดกำลังส่งสัญญาณของบอร์ดเสริม BLE ที่มีเสาสัญญาณแบบ Built-in จะทำให้ไม่สามารถใช้เครื่อง Spectrum Analyzer เพื่อวัดกำลังส่งได้โดยตรง เนื่องจากไม่สามารถเชื่อมต่อกับเสาสัญญาณภายนอกได้ วิธีการที่สามารถใช้แทนได้คือ การใช้เครื่องมือที่รองรับการวัดสัญญาณ BLE โดยเฉพาะ เช่น BLE Signal Analyzer ซึ่งสามารถวัดกำลังส่งของสัญญาณ BLE โดยไม่ต้องเชื่อมต่อกับเสาสัญญาณภายนอก หรืออาจใช้ฟังก์ชันใน ESP32 เช่น RSSI (Received Signal Strength Indicator) ซึ่งช่วยให้สามารถประเมินกำลังสัญญาณที่ได้รับจาก BLE Slave ได้อย่างแม่นยำ นอกจากนี้ การทดสอบในสภาพแวดล้อมที่ควบคุม เช่น การตั้งระยะห่างระหว่าง BLE Slave และ ESP32 ก็สามารถช่วยให้การประเมินกำลังส่งจาก BLE Slave ได้ดีขึ้น อีกทางเลือกหนึ่งคือการใช้โมดูล BLE ที่สามารถติดตั้งเสาสัญญาณภายนอกได้ เพื่อให้สามารถใช้เครื่อง Spectrum Analyzer ได้อย่างมีประสิทธิภาพและแม่นยำยิ่งขึ้น

นอกจากนี้ การทดสอบโปรแกรมใน ภาษาคอมพิวเตอร์ C หรือ C++ จะช่วยให้สามารถตรวจสอบสถานะการทำงานของโปรแกรมได้ดีขึ้น และยังสามารถตรวจสอบการจัดการ Memory ได้อย่างมีประสิทธิภาพ เพื่อให้มั่นใจว่าโปรแกรมทำงานได้ตามที่ต้องการและไม่เกิดข้อผิดพลาดจากการจัดการข้อมูลใน Memory

บรรณานุกรม

[1] Joshua F. Ensworth; Matthew S. Reynolds. “BLE-Backscatter: Ultralow-Power IoT Nodes Compatible With Bluetooth 4.0 Low Energy (BLE) Smartphones and Tablets.”
https://ieeexplore.ieee.org/document/7898833

[2] Bluetooth. “Bluetooth Technology Overview.”
https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

[3] Unnote. “Bluetooth คือ อะไร มีหลักการทำงานอย่างไร และวิธีใช้ที่ปลอดภัย.”
https://wifivitae.com/2020/05/01/ble-overview/

[4] Ilsun You; Soonhyun Kwon; Gaurav Choudhary; Vishal Sharma; Jung Taek Seo. “An Enhanced LoRaWAN Security Protocol for Privacy Preservation in IoT with a Case Study on a Smart Factory-Enabled Parking System.”
https://www.researchgate.net/publication/325575837_An_Enhanced_LoRaWAN_Security_Protocol_for_Privacy_Preservation_in_IoT_with_a_Case_Study_on_a_Smart_Factory-Enabled_Parking_System [5] STMicroelectronics. (2024). “STM32H562xx and STM32H563xx Datasheet - Arm® Cortex®-M33 32-bit MCU +TrustZone® + FPU, 375 DMIPS, 250 MHz, 2-Mbyte flash, 640-Kbyte RAM, math accelerators (DS14258 Rev 3).”
https://www.st.com/resource/en/datasheet/stm32h562ag.pdf

[6] digikey. “LoRa-E5 Series Module, Boards, and Dev Kit.”
https://www.digikey.co.th/th/product-highlight/s/seeed/lora-e5-series

[7] STMicroelectronics. “STM32H562xx and STM32H563xx - Arm Cortex-M33 32-bit MCU +TrustZone + FPU, 375 DMIPS, 250 MHz, 2-Mbyte flash, 640-Kbyte RAM, math accelerators (DS14258 Rev 3).”
https://www.st.com/resource/en/datasheet/stm32h562ag.pdf

[8] STMicroelectronics. “STMicroelectronics. (2024, May). STM32 Cortex-M33 MCUs Programming Manual (PM0264 Rev 4).”
https://www.st.com/resource/en/programming_manual/pm0264-stm32-cortexm33-mcus-programming-manual-stmicroelectronics.pdf

บรรณานุกรม (ต่อ)

[9] Petersen, L. “Part 1: Introduction to the STM32 Microcontroller Clock System. STMicroelectronics Community.” https://community.st.com/t5/stm32-mcus/part-1-introduction-to-the-stm32-microcontroller-clock-system/ta-p/605369

[10] S Rangeetha; S Pandithurai; A Prassath; A Reegan Cyril Raj. “Design and Implementation of AHB to APB Bridge and AHB to UART Communication for SoC Integration.” https://ieeexplore.ieee.org/document/10692006

[11] Zephyr Project. “About the Zephyr Project.” https://zephyrproject.org/learn-about/

[12] Zephyr Project. “Zephyr Security Overview.” https://docs.zephyrproject.org/latest/security/security-overview.html.

[13] Zephyr Project. “Syntax and structure — Zephyr Project Documentation.” https://docs.zephyrproject.org/latest/build/dts/intro-syntax-structure.html.

[14] Ashok Kumar Gupta; Ashish Raman; Naveen Kumar; Ravi Ranjan. “Design and Implementation of High-Speed Universal Asynchronous Receiver and Transmitter (UART).” https://ieeexplore.ieee.org/document/9070856

[15] Chiradeep BasuMallick.“What Is MQTT (MQ Telemetry Transport)? Working, Types, Importance, and Applications.” https://www.spiceworks.com/tech/iot/articles/what-is-mqtt/

[16] Vidushi Gupta.“WHAT IS MQTT PROTOCOL FOR IOT DEVICES?.” https://psiborg.in/advantages-of-using-mqtt-for-iot-devices/

[17] Ramotion. “Understanding The Single-Page Application Architecture.” https://www.ramotion.com/blog/single-page-application-architecture/

[18] Saurabh Dashora. “How Single Page Web Applications Actually Work.” https://dzone.com/articles/how-single-page-web-applications-actually-work

บรรณานุกรม (ต่อ)

[19] Adobe Experience Cloud Team. “Single-page applications (SPAs) — what they are and how they work.” https://business.adobe.com/blog/basics/learn-the-benefits-of-single-page-apps-spa

[20] Pirush Prechathavanich. “มาทำความรู้จัก Web Development คืออะไร มีความสำคัญอย่างไร?.” https://www.criclabs.co/post/what-is-web-development

[21] souravsharma098. “Node.js Tutorial.” https://www.geeksforgeeks.org/nodejs/

[22] shobhit__sharma. “React Tutorial.” https://www.geeksforgeeks.org/react-tutorial/?ref=outind

[23] starkyy. “What are the advantages of React.js ?.” https://www.geeksforgeeks.org/react-tutorial/?ref=outind

[24] Espressif Systems (Shanghai) Co., Ltd. “ESP32WROOM32 Datasheet.” https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[25] Dragino Technology Co., Limited. “Open Source LoRa WiFi Gateway LG01.” https://www.dragino.com/downloads/downloads/datasheet/EN/Datasheet_LG01.pdf

[26] DusunIoT. “LoRa-E5 LoRa Wireless Module - Powered by STM32WE5 Datasheet V1.0.” https://files.seeedstudio.com/products/317990687/res/LoRa-E5+module+datasheet_V1.0.pdf

[27] DusunIoT. “DSGW-094 Nordic Bluetooth Cellular (LTE M/NB IoT) Transparent Gateway.” https://www.dusuniot.com/product/dsgw-094-nordic-bluetooth-cellular-gateway/

[28] DusunIoT. “Spec: DSGW-092 ESP32 Gateway – BLE/ZigBee to Wi-Fi/LTE/Ethernet.” https://www.dusuniot.com/product-specification/dsgw-092-esp32-ble-gateway/

บรรณานุกรม (ต่อ)

[29] Espressif Systems (Shanghai) Co., Ltd. “ESP32Series Datasheet Version4.8.” https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[30] R. Shantha Selva Kumari; C. Gayathri. “Interfacing of MEMS motion sensor with FPGA using I2C protocol.” https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

ภาคผนวก ก

การพัฒนาเฟิร์มแวร์

- **โค้ด main.c ใช้งานบน Core board**

```
#include "uart_com.h"
#include "protocol_check.h"
#include <zephyr/drivers/gpio.h>
#include <zephyr/drivers/watchdog.h>
#include <zephyr/sys/printk.h>
#include <string.h>
#include <led.h>
#include <led1.h>
#include <buzzer.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <zephyr/debug/thread_analyzer.h>

/* UART device configuration */
#define UART_DEVICE_NODE DT_NODELABEL(usart3) /* Lora */
#define UART_DEVICE_NODE1 DT_NODELABEL(uart4) /* BLE & ESP32*/

/* UART devices */
static const struct device *const lora_uart =
DEVICE_DT_GET(UART_DEVICE_NODE);
static const struct device *const ble_uart =
DEVICE_DT_GET(UART_DEVICE_NODE1);
/* Check devices */
int connected_count = 0; // Counter for connected devices
#define MAX_DEVICES 50                    /*Define max device in BLE*/
char filtered_mac_addresses[MAX_DEVICES][18]; /*Store mac address
                                          "XX:XX:XX:XX:XX:XX" is 17 character + \0
                                           null terminal 17+1 = 18 */
int filtered_count = 0;                   /*ตัวนับจำนวนอุปกรณ์ที่ผ่านการกรอง*/
```

```
/*Store data*/
double frequency_found[9]; /*j*/
char rx_lora[20][50];      /*x*/
int j = 0;
int x = 0;

/*for status device*/
const char *lora_status;
const char *ble_status;

bool wifi_state = false;
const char *status_publish = "OK";

int rssi, snr;
#define RX_BUFFER_SIZE 2048 // กำหนด buffer ที่ใหญ่พอ
char rx_data[RX_BUFFER_SIZE] = {0};

char ascii_result[100]; // Buffer สำหรับเก็บผลลัพธ์ ASCII

char keyword[] = "4E4F44453"; // Keyword ที่ต้องการเช็ค
bool target_lora;

char node_id[20];   // เก็บ Node ID
double temperature; // ค่าอุณหภูมิ
double dust;        // ค่าฝุ่น

#define MAX_MESSAGES 10                 // กำหนดขนาดสูงสุดของอาร์เรย์ (ต้อง
>= j)
char lora_messages[MAX_MESSAGES][256]; // อาร์เรย์เก็บข้อมูล LoRa MQTT

float Tempble = 0.0, Dustble = 0.0;
```

```
int rssi_lora;
int snr_lora;

char extracted_data[100]; // Buffer สำหรับเก็บผลลัพธ์

/*BLE Parameter*/
#define TARGET_PAYLOAD_BLE
"02010611074b9131c3c9c5cc8f9e45b51f01c2af4f051208004000" /*target detect BLE*/
#define MAX_DEVICES_BLE 50
/*For ble device*/
#define CLEANED_BUFFER_BLE_SIZE 12000
/*Array size for cleaned BLE*/
#define BUFFER_SIZE_STORE_MAC_ADDRESS 2048
char target_macs[MAX_DEVICES][18] = {0};          /*Array เก็บ MAC
Address*/
char cleaned_buffer[CLEANED_BUFFER_BLE_SIZE] = {0}; /*Store Buffer BLE
After cleaned*/
int cleaned_index = 0;
int target_ble_count = 0; /*Count number target_ble have*/
#define MAX_MESSAGES_BLE 10
char ble_messages[MAX_MESSAGES_BLE][256]; // อาร์เรย์เก็บข้อมูล BLE

/*lora parameter*/
int rssi_lora_rx = 0;
int snr_lora_rx = 0;
char rx_lora_for_publish[100];
char rx_buffer_lora_after_rec[256];
char lora_messages_for_publish[MAX_MESSAGES_BLE][256]; // อาร์เรย์เก็บ
ข้อมูล BLE
```

```
#define MAX_LINES 10       // กำหนดจำนวนบรรทัดสูงสุดที่สามารถจัดเก็บได้
#define MAX_LINE_LENGTH 50 // กำหนดความยาวสูงสุดของแต่ละบรรทัด
#define MAX_ROUNDS 10      // กำหนดจำนวนรอบสูงสุด
#define MAX_LENGTH 256

void find_target_in_ble_scan(const char *cleaned_buffer)
{
    char *line = strtok((char *)cleaned_buffer, "\n"); // แยกบรรทัด
    while (line != NULL)
    {
        if (strstr(line, TARGET_PAYLOAD_BLE) != NULL)
        {
            char mac[18];
            if (sscanf(line, "+BLESCAN:\"%17[^\"]\",", mac) == 1)
            {
                if (!is_mac_address_duplicate(mac))
                {
                    if (target_ble_count < MAX_DEVICES)
                    {
                        strcpy(target_macs[target_ble_count], mac);
                        target_ble_count++;
                    }
                }
            }
        }
        line = strtok(NULL, "\n");
    }
}

int is_mac_address_duplicate(const char *mac)
{
```

```
    for (int i = 0; i < target_ble_count; i++)
    {
        if (strcmp(target_macs[i], mac) == 0)
        {
            return 1; // เจอซ้ำ
        }
    }
    return 0; // ไม่ซ้ำ
}

void parse_ble_data(const char *rx_buffer_ble, float *Tempble, float
*Dustble)
{
    char *temp_ptr, *dust_ptr;
    char buffer[100];

    strncpy(buffer, rx_buffer_ble, sizeof(buffer));
    buffer[sizeof(buffer) - 1] = '\0';

    // ค้นหาคำว่า "Temp="
    temp_ptr = strstr(buffer, "Temp=");
    if (temp_ptr)
    {
        *Tempble = atof(temp_ptr + 5);
    }

    // ค้นหาคำว่า "Dust="
    dust_ptr = strstr(buffer, "Dust=");
    if (dust_ptr)
    {
        *Dustble = atof(dust_ptr + 5);
```

```
    }
}

void clear_buffer()
{
    memset(rx_buffer_lora, 0, MSG_SIZE);
    memset(rx_buffer_ble, 0, MSG_SIZE);
    rx_buffer_index_lora = 0;
    rx_buffer_index_ble = 0;
    data_received_lora = false;
    data_received_ble = false;
}

void clear_buffer_lora()
{
    memset(rx_buffer_lora, 0, MSG_SIZE);
    rx_buffer_index_lora = 0;
    data_received_lora = false;
}

void clear_buffer_ble()
{
    memset(rx_buffer_ble, 0, MSG_SIZE);
    rx_buffer_index_ble = 0;
    data_received_ble = false;
}

/*for check data in array rx_lora*/
int is_rx_data_in_lora(const char *rx_data)
{
    // printk("Rx before scan %s\n", rx_data);
```

```
    for (int i = 0; i < x; i++)
    {
        if (strcmp(rx_lora[i], rx_data) == 0)
        {
            return 1;
        }
    }
    return 0;
}

int check_rx_keyword(const char *rx_buffer_lora, const char *keyword)
{
    char *rx_start = strstr(rx_buffer_lora, "RX \""); // ค้นหา RX "
    if (rx_start)
    {
        rx_start += 4;

        if (strlen(rx_start) < 9)
        {
            target_lora = false;
        }

        if (strncmp(rx_start, keyword, 9) == 0)
        {
            target_lora = true;
        }
    }
}

int main(void)
{
```

```
    led_init();
    led1_init();
    buzzer_init();

    /* Enable UART interrupts */
    uart_irq_rx_enable(lora_uart);
    uart_irq_rx_enable(ble_uart);

    /* Set UART callbacks */
    uart_irq_callback_user_data_set(lora_uart, uart_callback_lora, NULL);
    uart_irq_callback_user_data_set(ble_uart, uart_callback_ble, NULL);

    /* Check devices */
    printk("-> Start to scan protocol to use\n");
    printk("+--------------------+--------------+\n");
    printk("| Device Name        | Status       |\n"); // แก้ไขหัวตารางให้กว้างขึ้น
    printk("+--------------------+--------------+\n");

    bool lora_connected = check_device(lora_uart, "Lora", &connected_count);
    k_msleep(2000);
    bool ble_connected = check_device(ble_uart, "BLE&ESP32", &connected_count);
    k_msleep(500);

    printk("+----------------------+----------------+\n");

    if (connected_count > 0)
    {
        printk("| Summary of Device is %d device\n", connected_count);
```

```
}
else
{
    printk("| No devices are connected.\n");
    printk("| Please insert protocol to use.\n");
}

/*lora setup*/
if (lora_connected)
{
    uart_send_command(lora_uart, "AT+MODE=TEST");
    while (!data_received_lora)
    {
      k_msleep(10);
    }
    /*open for debug*/
    // printk("Response lora is %s", rx_buffer_lora);
    clear_buffer();

    printk("\n");
    printk("-> LoRa Endnode Scan (LoRa Connected)\n");
    printk("+----------------------+---------------+\n");
    printk("| Frequency (MHz)      | Status        |\n");
    printk("+----------------------+---------------+\n");

    for (double frequency = 920.000; frequency <= 925.000; frequency
+= 0.600)
    {

        int retry_count = 0;
        while (retry_count < 3)
```

```
{
    clear_buffer();
    char command[128];
    snprintf(command, sizeof(command),
"AT+TEST=RFCFG,%.3f,SF9,125,8,8,20,OFF,OFF,ON\r\n", frequency);
    uart_send_command(lora_uart, command);
    while (!data_received_lora)
    {
        k_msleep(10);
    }

    /*open for debug what frequency to use*/
    // printk("Receiver at %s\n", rx_buffer_lora);

    clear_buffer();

    k_msleep(5000);

    uart_send_command(lora_uart, "AT+TEST=RXLRPKT");
    while (!data_received_lora)
    {
        k_msleep(10);
    }
    clear_buffer();
    k_msleep(2000);

    int timeout = 0;
    while (!data_received_lora)
    {
        if (timeout >= 3000)
        {
```

```
            target_lora = false;
            break;
        }
        target_lora = false;
        k_msleep(10);
        timeout += 10;
    }

    /*open for debug data*/
    // printk("%s\n", rx_buffer_lora);

    check_rx_keyword(rx_buffer_lora, keyword);

    /*open for debug*/
    // printk("%d\n", target_lora);

    /*old version*/
    // const char *substring = "4E4F44453"; /*find node keyword*/
    // if (strstr(rx_buffer_lora, substring) != NULL)
    if (target_lora)
    {
        target_lora = false;
        /*data process*/
        char *newline_pos = strstr(rx_buffer_lora, "\n");
        if (newline_pos != NULL)
        {
            *newline_pos = '\0';

            char *line1 = rx_buffer_lora;
            char *line2 = newline_pos + 1;
```

```
                int len, rssi, snr;
                char rx[100];

                sscanf(line1, "+TEST: LEN:%d, RSSI:%d, SNR:%d", &len,
&rssi, &snr);
                sscanf(line2, "+TEST: RX \"%10[^\"]\"", rx);

                /*response check*/
                if (!is_rx_data_in_lora(rx))
                {
                    strncpy(rx_lora[x], rx, sizeof(rx_lora[x]) - 1);
                    rx_lora[x][sizeof(rx_lora[x]) - 1] = '\0';
                    x++;

                    printk("| %f MHz      | Found        |\n", frequency);
                    frequency_found[j] = frequency;
                    j++;

                    break;
                }
                else
                {
                    retry_count++;
                    continue;
                }
            }
        }
        else
        {
            printk("| %f MHz      | not found    |\n", frequency);
            break;
```

```
            }
        }
        clear_buffer();

        uart_send_command(lora_uart, "AT+MODE=TEST");

        /*RESET LORA*/
        while (!data_received_lora)
        {
            k_msleep(10);
        }
        k_msleep(2000);
    }
    printk("+----------------------+-------------+\n");
    printk("| Found %d End node\n", j);
    printk("\n");

    /*open for debug frequency found*/
    // for (int i = 0; i < j; i++)
    // {
    //     printk("%f ", frequency_found[i]); /*rx_lora[i] for show rx_data *x
, frequency_found[i] for shows frequency *j */
    // }
}

if (ble_connected)
{
    printk("\n");
    printk("-> BLE Endnode Scan (BLE Connected)\n");
    printk("+----------------------+----------------+\n");
    printk("| Setup Process        | Status         |\n");
```

```
printk("+----------------------+----------------+\n");

printk("| Disconnect Prv. BLE   |");
while (1)
{
    uart_send_command(ble_uart, "AT+RST");
    k_msleep(1000); /*Waiting data*/
    // printk("%s", rx_buffer_ble); /*open for debug*/
    if (strstr(rx_buffer_ble, "OK") != NULL)
    {
        printk(" Pass           |\n");
        break;
    }
    else
    {
        continue;
    }
}

clear_buffer();
k_msleep(5000); /*at least*/

printk("| Setup Role BLE      |");
while (1)
{
    uart_send_command(ble_uart, "AT+BLEINIT=1");
    k_msleep(1000);
    // printk("%s", rx_buffer_ble); /*open for debug*/
    if (strstr(rx_buffer_ble, "ready") != NULL)
    {
        printk(" Pass           |\n");
```

```
            break;
        }
        else
        {
            continue;
        }
    }

    clear_buffer();
    k_msleep(2000);

    printk("+--------------------+--------------+\n");
    printk("| Scan BLE Mac address  | Status       |\n");
    printk("+--------------------+--------------+\n");

    printk("| Scaning end node BLE  |");
    while (1)
    {
        clear_buffer();
        uart_send_command(ble_uart, "AT+BLEINIT=1");
        k_msleep(1000);
        clear_buffer();
        uart_send_command(ble_uart, "AT+BLESCAN=1,5");
        k_msleep(5000);

        // printk("%s", rx_buffer_ble); /*open for debug*/
        if (strstr(rx_buffer_ble, "ERROR") != NULL || strstr(rx_buffer_ble,
"busy p...") != NULL)
        {
            continue;
        }
```

```
    else
    {
        printk(" Pass            |\n");
        break;
    }
}
// printk("\n");
// printk("%s\n", rx_buffer_ble);
// printk("\n");
/*Version 3*/
char *line_ble_scan = strtok(rx_buffer_ble, "\n");
while (line_ble_scan != NULL)
{
    if (strstr(line_ble_scan, "+BLESCAN:") != NULL)
    { // ตรวจสอบว่ามี "+BLESCAN:"
        int len = strlen(line_ble_scan);
        int quote_count = 0;
        int comma_count = 0;

        for (int i = 0; i < len; i++)
        {
            if (line_ble_scan[i] == '"')
                quote_count++;
            if (line_ble_scan[i] == ',')
                comma_count++;
        }

        if (quote_count == 2 && comma_count >= 3)
        {
            if (cleaned_index + len < CLEANED_BUFFER_BLE_SIZE - 2)
            {
```

```
                strcat(cleaned_buffer, line_ble_scan);
                strcat(cleaned_buffer, "\n");
                cleaned_index += len + 1;
            }
        }
    }
    line_ble_scan = strtok(NULL, "\n");
}

/*open for debug data filter*/
// printk("%s", cleaned_buffer);

find_target_in_ble_scan(cleaned_buffer); /*function finding target BLE*/

printk("+--------------------+--------------+\n");
for (int i = 0; i < target_ble_count; i++)
{
    printk("| %s                  |\n", target_macs[i]);
    // printk("Device %d: %s\n", i + 1, filtered_mac_addresses[i]);
}
// printk("+--------------------+--------------+\n");
// printk("| Found %d End node\n", target_ble_count);

k_msleep(1000);
clear_buffer();

printk("+----------------------+---------------+\n");
printk("| Setup Rx BLE         | Status        |\n");
printk("+----------------------+---------------+\n");
printk("| Connecting to BLE    |");
```

```
for (int i = 0; i < target_ble_count; i++)
{
    char command_connect_ble[50];
    snprintf(command_connect_ble, sizeof(command_connect_ble),
"AT+BLECONN=%d,\"%s\",0,10", i, target_macs[i]); // แก้ไขรูปแบบคำสั่ง

    while (1)
    {
        k_msleep(1000);
        uart_send_command(ble_uart, command_connect_ble);
        k_msleep(2000);
        // printk("%s", rx_buffer_ble); /*open for debug*/
        if (strstr(rx_buffer_ble, "OK") != NULL)
        {
            printk(" Pass          |\n");
            break;
        }
        else
        {
            printk(" Try agian     |\n");
            continue;
        }
    }
}

k_msleep(1000);
clear_buffer();

printk("| GATT Discover service |");
for (int i = 0; i < target_ble_count; i++)
{
```

```
                    char command_ble_gatt[50];
                    snprintf(command_ble_gatt, sizeof(command_ble_gatt),
"AT+BLEGATTCPRIMSRV=%d", i); // แก้ไขรูปแบบคำสั่ง
                    while (1)
                    {
                        uart_send_command(ble_uart, command_ble_gatt);
                        k_msleep(1000);
                        // printk("Response: %s\n", rx_buffer_ble); // Debugging
response
                        if (strstr(rx_buffer_ble, "OK") != NULL)
                        {
                            clear_buffer();
                            break;
                        }
                        else
                        {
                            clear_buffer();
                            continue;
                        }
                    }
                }
                printk(" Pass          |\n");

                k_msleep(1000);
                clear_buffer();

                printk("| GATT Discover Charac. |");
                for (int i = 0; i < target_ble_count; i++)
                {
                    char command_discover_characteristic[50];
```

```
                snprintf(command_discover_characteristic,
sizeof(command_discover_characteristic), "AT+BLEGATTCCHAR=%d,3", i); // แก้ไขรูปแบบ
คำสั่ง

                while (1)
                {
                    uart_send_command(ble_uart,
command_discover_characteristic);
                    // printk("Response: %s\n", rx_buffer_ble); // Debugging
response
                    if (strstr(rx_buffer_ble, "OK") != NULL)
                    {
                        clear_buffer();
                        break;
                    }
                    else
                    {
                        clear_buffer();
                        continue;
                    }
                }
            }
            printk(" Pass          |\n");

            k_msleep(1000);
            clear_buffer();

            /*For setup WLAN*/
            printk("+---------------------+---------------+\n");
            printk("| Setup WiFi Connected  | Status        |\n");
            printk("+---------------------+---------------+\n");
```

```
printk("| Set Role WiFi          |");
while (1)
{
    uart_send_command(ble_uart, "AT+CWMODE=1");
    k_msleep(1000);
    // printk("Response: %s\n", rx_buffer_ble); // Debugging response
    if (strstr(rx_buffer_ble, "OK") != NULL)
    {
        clear_buffer();
        break;
    }
    else
    {
        clear_buffer();
        continue;
    }
}
printk(" Pass          |\n");

k_msleep(1000);
clear_buffer();

uart_send_command(ble_uart, "AT+CWSTATE?");
k_msleep(1000);
/*open for debug*/
// printk("%s", rx_buffer_ble);
const char *substring = "+CWSTATE:3";  /*WiFi disconnected*/
const char *substring2 = "+CWSTATE:4"; /* WiFi disconnected (อีก
สถานะ) */

const char *substring3 = "busy p...";  /* WiFi disconnected (อีกสถานะ)
*/
```

```
                const char *substring4 = "ERROR";      /* WiFi disconnected (อีกสถานะ)
*/
                if (strstr(rx_buffer_ble, substring) != NULL || strstr(rx_buffer_ble,
substring2) != NULL || strstr(rx_buffer_ble, substring3) != NULL || strstr(rx_buffer_ble,
substring4) != NULL)
                {
                    printk("| WiFi Disconnected     | open network   |\n");
                    while (1)
                    {
                        clear_buffer();
                        k_msleep(2000);

                        uart_send_command(ble_uart,
"AT+CWJAP=\"Ohmaomsinwifi\",\"123456789\"");
                        k_msleep(5000);
                        // printk("%s", rx_buffer_ble);

                        const char *substring = "+CWSTATE:3";  /*WiFi disconnected*/
                        const char *substring2 = "+CWSTATE:4"; /* WiFi disconnected
(อีกสถานะ) */
                        const char *substring3 = "busy p..."; /* WiFi disconnected (อีก
สถานะ) */
                        const char *substring4 = "ERROR";      /* WiFi disconnected (อีก
สถานะ) */
                        const char *substring5 = "+CWJAP:3";   /* WiFi disconnected
(อีกสถานะ) */
                        const char *substring6 = "OK";         /* WiFi disconnected (อีก
สถานะ) */
                        if (strstr(rx_buffer_ble, substring) != NULL || strstr(rx_buffer_ble,
substring2) != NULL || strstr(rx_buffer_ble, substring3) != NULL || strstr(rx_buffer_ble,
substring4) != NULL || strstr(rx_buffer_ble, substring5) != NULL)
```

```
        {
            printk("| WiFi Disconnected     | open network   |\n");
            wifi_state = false;
            buzzer_on();
            k_msleep(500);
            buzzer_off();
            k_msleep(500);
            buzzer_on();
            k_msleep(500);
            buzzer_off();
        }
        else if (strstr(rx_buffer_ble, substring6) != NULL)
        {
            printk("| WiFi Connected        | Pass          |\n");
            wifi_state = true;
            break;
        }
    }
}
else
{
    printk("| WiFi Connected        | Pass          |\n");
}

k_msleep(1000);
clear_buffer();

printk("+----------------------+----------------+\n");
printk("| Setup Broker          | Status         |\n");
printk("+----------------------+----------------+\n");
printk("| Username setup        |");
```

```
                while (1)
                {
                    uart_send_command(ble_uart,
"AT+MQTTUSERCFG=0,4,\"userpb\",\"user1\",\"8888888888\",0,0,\"\"");
                    k_msleep(1000);
                    // printk("Response: %s\n", rx_buffer_ble); // Debugging response
                    if (strstr(rx_buffer_ble, "OK") != NULL)
                    {
                        clear_buffer();
                        break;
                    }
                    else
                    {
                        clear_buffer();
                        continue;
                    }
                }
                printk(" Pass          |\n");

                k_msleep(1000);
                clear_buffer();

                uart_send_command(ble_uart,
"AT+MQTTCONNCFG=0,0,0,\"LoRa/data\",\",0,0"); /*topic*/
                k_msleep(500);
                uart_send_command(ble_uart,
"AT+MQTTCONNCFG=0,0,0,\"BLE/data\",\",0,0"); /*topic*/
                k_msleep(500);
                uart_send_command(ble_uart,
"AT+MQTTCONNCFG=0,0,0,\"lora/endnode\",\",0,0"); /*topic*/
                k_msleep(500);
```

```
                uart_send_command(ble_uart,
"AT+MQTTCONNCFG=0,0,0,\"BLE/endnode\",\",0,0"); /*topic*/
                k_msleep(500);
                uart_send_command(ble_uart,
"AT+MQTTCONNCFG=0,0,0,\"setup/Protocol_use\",\",0,0"); /*topic*/
                k_msleep(500);
                printk("| Connected to Topic    | Pass            |\n");

                k_msleep(1000);
                clear_buffer();

                printk("| Connected to Broker   |");
                while (1)
                {
                    uart_send_command(ble_uart,
"AT+MQTTCONN=0,\"o431a7ce.ala.asia-southeast1.emqxsl.com\",8883,1"); /*topic*/
                    k_msleep(5000);
                    /*open for debug respone*/
                    // printk("%s\n", rx_buffer_ble);
                    if (strstr(rx_buffer_ble, "OK") != NULL)
                    {
                        clear_buffer();
                        break;
                    }
                    else
                    {
                        clear_buffer();
                        continue;
                    }
                }
                printk(" Pass           |\n");
```

```
}
printk("+----------------------+----------------+\n");
k_msleep(1000);
clear_buffer();

if (lora_connected)
{
    lora_status = "True";
}
else
{
    lora_status = "False";
}
if (ble_connected)
{
    ble_status = "True";
}
else
{
    ble_status = "False";
}
printk("\n");
printk("-> Publish status to broker\n");
printk("+----------------------+----------------+\n");
printk("| Topic                | Status         |\n");
printk("+----------------------+----------------+\n");

while (1)
{
    while (1)
```

```
{
    printk("| setup/Protocol          |");
    char mqtt_command[256]; // Buffer for publish device
    snprintf(mqtt_command, sizeof(mqtt_command),

"AT+MQTTPUB=0,\"setup/Protocol_use\",\"{\\\"LoRa\\\":\\\"%s\\\"\\,\\\"BLE\\\":\\\"%s\\\
"\\}\",0,0",
                lora_status, ble_status);
    uart_send_command(ble_uart, mqtt_command);
    k_msleep(3000);
    if (strstr(rx_buffer_ble, status_publish) != NULL)
    {
      printk(" Pass          |\n");
      clear_buffer();
      break;
    }
    else
    {
      printk(" Try agian      |\n");
      clear_buffer();
      continue;
    }
    k_msleep(1000);
}

k_msleep(3000);

while (1)
{
    printk("| BLE/endnode             |");
    char mqtt_command[256]; // Buffer for publish device
```

```
                snprintf(mqtt_command, sizeof(mqtt_command),

"AT+MQTTPUB=0,\"BLE/endnode\",\"{\\\"Node\\\":\\\"%d\\\"\\}\",0,0",
                        target_ble_count);
                uart_send_command(ble_uart, mqtt_command);
                k_msleep(3000);
                if (strstr(rx_buffer_ble, status_publish) != NULL)
                {
                    printk(" Pass          |\n");
                    clear_buffer();
                    break;
                }
                else
                {
                    printk(" Try agian     |\n");
                    clear_buffer();
                    continue;
                }
                k_msleep(1000);
            }
            k_msleep(3000);

            while (1)
            {
                printk("| lora/endnode          |");
                char mqtt_command[256]; // Buffer for publish device
                snprintf(mqtt_command, sizeof(mqtt_command),

"AT+MQTTPUB=0,\"lora/endnode\",\"{\\\"Node\\\":\\\"%d\\\"\\}\",0,0",
                        j);
```

```
            uart_send_command(ble_uart, mqtt_command);
            k_msleep(3000);
            if (strstr(rx_buffer_ble, status_publish) != NULL)
            {
                printk(" Pass            |\n");
                clear_buffer();
                break;
            }
            else
            {
                printk(" Try agian       |\n");
                clear_buffer();
                continue;
            }
            k_msleep(1000);
        }
        break;
    }

    int round = 0;
    int round_broker = 0;
    printk("\n");
    printk("-> Publish data to broker\n");
    printk("+----------------------+------------------+------------+\n");
    printk("| Topic                | Endnode          | Status     |\n");
    printk("+----------------------+------------------+------------+\n");
    while (1)
    {
        // printk("Size buffer lora %d\n", sizeof(rx_buffer_lora));
        // printk("Size buffer BLE %d\n", sizeof(rx_buffer_ble));
        /*LoRa rx data*/
```

```
                printk("----------------------------------------------------------------------------------------
----------------------------------------\n");
                if (lora_connected)
                {
                    for (int k = 0; k < j; k++)
                    {
                        char command_LORA_Publish[128];
                        snprintf(command_LORA_Publish,
sizeof(command_LORA_Publish),
"AT+TEST=RFCFG,%.3f,SF9,125,8,8,20,OFF,OFF,ON\r\n", frequency_found[k]);

                        uart_send_command(lora_uart, command_LORA_Publish);
                        k_msleep(500);

                        while (!data_received_lora)
                        {
                            k_msleep(10);
                        }

                        clear_buffer_lora();
                        k_msleep(200);

                        uart_send_command(lora_uart, "AT+TEST=RXLRPKT");
                        k_msleep(200);
                        clear_buffer_lora();
                        k_msleep(1000);

                        int retry_count = 0;
                        while (retry_count < 10)
                        {
                            if (rx_buffer_lora == NULL || strlen(rx_buffer_lora) == 0)
```

```
                {
                    printk("Error: No data received. Retrying (%d/10)...\n",
retry_count + 1);
                    k_msleep(1000);
                    retry_count++;
                    continue;
                }

                if (strlen(rx_buffer_lora) > 200)
                {
                    printk("\nBuffer after clear usage: %d/%d bytes\n",
strlen(rx_buffer_lora), MSG_SIZE);
                    break;
                }
                else
                {
                    printk("\nTry again (Attempt %d)\n", retry_count + 1);
                    k_msleep(1000);
                    retry_count++;
                }
            }
            if (retry_count == 10)
            {
                printk("Error: Unable to receive valid data. Skipping this
round.\n");
                continue;
            }

            /* Rx data processing */
            char buffer_copy_rx[256];
```

```
snprintf(buffer_copy_rx, sizeof(buffer_copy_rx), "%s",
rx_buffer_lora);

char *lines[MAX_LINES] = {NULL}; // array สำหรับเก็บ pointer ไป
ยังแต่ละบรรทัด
int line_count = 0;

char *token = strtok(buffer_copy_rx, "\n");
while (token != NULL && line_count < MAX_LINES)
{
    size_t token_length = strlen(token);
    lines[line_count] = (char *)malloc(token_length + 1);
    if (lines[line_count] == NULL)
    {
        printk("Error: Memory allocation failed!\n");
        break; // หยุด loop ถ้า malloc ล้มเหลว
    }
    snprintf(lines[line_count], token_length + 1, "%s", token);
    line_count++;
    token = strtok(NULL, "\n");
}

char line_lora_rx_buffer1[MAX_LENGTH] = {0};
char line_lora_rx_buffer2[MAX_LENGTH] = {0};
int found_len = 0;

for (int i = 0; i < line_count; i++)
{
    if (strstr(lines[i], "+TEST: LEN"))
    {
```

```
                snprintf(line_lora_rx_buffer1, sizeof(line_lora_rx_buffer1),
"%s", lines[i]);
                found_len = 1;
            }
            else if (found_len && strstr(lines[i], "+TEST: RX"))
            {
                snprintf(line_lora_rx_buffer2, sizeof(line_lora_rx_buffer2),
"%s", lines[i]);
                break;
            }
        }

        int len_lora_rx_buffer = 0;
        int rssi_lora_rx_buffer = 0;
        int snr_lora_rx_buffer = 0;
        char rx_data_lora_rx_buffer[MAX_LENGTH] = {0};
        char ascii_data[256];
        int len_rx_buffer_befor_ascii = 0;
        char node_id_rx_lora[10];
        float temp_lora_rx_buffer, dust_lora_rx_buffer;

        if (sscanf(line_lora_rx_buffer1, "+TEST: LEN:%d, RSSI:%d,
SNR:%d",
                   &len_lora_rx_buffer, &rssi_lora_rx_buffer,
&snr_lora_rx_buffer) != 3)
        {
            printk("Error: Invalid format for LEN, RSSI, SNR\n");
            continue;
        }
```

```
                    if (sscanf(line_lora_rx_buffer2, "+TEST: RX \"%[^\"]\"",
rx_data_lora_rx_buffer) != 1)
                    {
                        printk("Error: Invalid format for RX buffer\n");
                        continue;
                    }

                    len_rx_buffer_befor_ascii = strlen(rx_data_lora_rx_buffer);
                    if (len_rx_buffer_befor_ascii % 2 != 0)
                    {
                        printk("Error: RX data length is not even. Skipping.\n");
                        continue;
                    }

                    for (int i = 0, j = 0; i < len_rx_buffer_befor_ascii; i += 2, j++)
                    {
                        ascii_data[j] = (rx_data_lora_rx_buffer[i] -
(rx_data_lora_rx_buffer[i] >= 'A' ? 'A' - 10 : '0')) * 16 +
                                        (rx_data_lora_rx_buffer[i + 1] -
(rx_data_lora_rx_buffer[i + 1] >= 'A' ? 'A' - 10 : '0'));
                    }
                    ascii_data[len_rx_buffer_befor_ascii / 2] = '\0';

                    if (sscanf(ascii_data, "%s Temp=%f Dust=%f", node_id_rx_lora,
&temp_lora_rx_buffer, &dust_lora_rx_buffer) != 3)
                    {
                        printk("Error: Unable to extract Node ID, Temp, and Dust\n");
                        continue;
                    }
```

```
                printk("\nRSSI: %d, SNR: %d, Node ID: %s, Temp: %f, Dust:
%f\n",
                       rssi_lora_rx_buffer, snr_lora_rx_buffer, node_id_rx_lora,
temp_lora_rx_buffer, dust_lora_rx_buffer);

                snprintf(lora_messages_for_publish[k],
sizeof(lora_messages_for_publish[k]),

"AT+MQTTPUB=0,\"LoRa/data\",\"{\\\"RSSI\\\":\\\"%d\\\"\\,"
                        "\\\"SNR\\\":\\\"%d\\\"\\,"
                        "\\\"Node\\\":\\\"%s\\\"\\,"
                        "\\\"Temp\\\":\\\"%f\\\"\\,"
                        "\\\"Dust\\\":\\\"%f\\\"\\}\",0,0",
                        rssi_lora_rx_buffer, snr_lora_rx_buffer, node_id_rx_lora,
temp_lora_rx_buffer, dust_lora_rx_buffer);

                for (int i = 0; i < line_count; i++)
                {
                if (lines[i] != NULL)
                {
                    free(lines[i]);
                lines[i] = NULL;
                }
                }


                while (1)
                {
                    printk("\nReset module\n");
                    uart_send_command(lora_uart, "AT+MODE=TEST");
                    k_msleep(500);
                    if (strstr(rx_buffer_lora, "+MODE: TEST") != NULL)
```

```
            {
                clear_buffer_lora();
                break;
            }
            clear_buffer_lora();
        }

        clear_buffer_lora();
        round++;
        k_msleep(500);
    }

    printk("Start to send data lora\n");
    for (int i = 0; i < j && i < MAX_MESSAGES_BLE; i++)
    {
        printk("| LoRa/data          | %f      |", frequency_found[i]);
        while (1)
        {
            buzzer_off();
            uart_send_command(ble_uart,
lora_messages_for_publish[i]);
            k_msleep(1000);
            // printk("%s", rx_buffer_ble); /*open for debug*/
            if (strstr(rx_buffer_ble, status_publish) != NULL)
            {
                round_broker++;
                printk(" Done       |");
                printk(" Round %d\n", round_broker);
                clear_buffer();
                break;
            }
```

```
                else
                {
                    // buzzer_on();
                    k_sleep(K_MSEC(100));
                    printk(" Try agian  |\n");
                    printk("\n");
                    printk("Resent module\n");
                    printk("\n");
                    clear_buffer();
                    continue;
                }
            }
            k_msleep(500);
        }
    }
    printk("-------------------------------------------------------------------------------
-------------------------------\n");
    k_msleep(500);
    if (ble_connected)
    {
     /*BLE Rx data*/
     for (int i = 0; i < target_ble_count; i++)
     {
        k_msleep(500);
        char command_BLE_Publish[50];
        snprintf(command_BLE_Publish,
sizeof(command_BLE_Publish), "AT+BLEGATTCRD=%d,3,1", i); // แก้ไขรูปแบบคำสั่ง
        // printk("Size buffer command_BLE_Publish %d\n",
sizeof(command_BLE_Publish));
        uart_send_command(ble_uart, command_BLE_Publish);
```

```
k_msleep(500);
// printk("%s\n", rx_buffer_ble); //Debugging response
parse_ble_data(rx_buffer_ble, &Tempble, &Dustble);
memset(ble_messages[i], 0, sizeof(ble_messages[i]));
snprintf(ble_messages[i], sizeof(ble_messages[i]),
        "AT+MQTTPUB=0,\"BLE/data\",\"{\\\"MAC\\\":\\\"%s\\\"\\,"
        "\\\"Temp\\\":\\\"%.2f\\\"\\,"
        "\\\"Dust\\\":\\\"%.2f\\\"\\}\",0,0",
        target_macs[i], Tempble, Dustble);

// printk("Mac address %s\n", filtered_mac_addresses[i]);
// printk("Tempble: %.2f\n", Tempble);
// printk("Dustble: %.2f\n", Dustble);
clear_buffer_ble();
}

/*BLE Send data data*/
for (int i = 0; i < target_ble_count; i++)
{
printk("| BLE/data          | %s |", target_macs[i]);
while (1)
{
    buzzer_off();
    uart_send_command(ble_uart, ble_messages[i]);
    k_msleep(1000);
    // printk("%s", rx_buffer_ble); /*open for debug*/

    if (strstr(rx_buffer_ble, status_publish) != NULL)
    {
        round_broker++;
        printk(" Done       |");
```

```
            printk(" Round %d\n", round_broker);
            clear_buffer();
            break;
        }
        else
        {
            // buzzer_on();
            k_sleep(K_MSEC(100));
            printk(" Try agian  |\n");
            clear_buffer();
            continue;
        }
        }
        k_msleep(500);
    }
    }
    }
    return 0;
}
```

ภาคผนวก ข

เว็บแสดงผล

- **Front end**

```
import React from "react";
import ReactDOM from "react-dom";
import Plot from "react-plotly.js";
import "./styles.css";

class IoTDashboard extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      sensorData: {
        LoRa: [],
        BLE: [],
      },
    };

    this.ws = null;
  }
  componentDidMount() {
    // เปิดการเชื่อมต่อ WebSocket
    this.ws = new WebSocket("ws://localhost:3001");

    this.ws.onopen = () => {
      console.log("Connected to WebSocket Server");
    };

    this.ws.onmessage = (event) => {
```

```
    const message = JSON.parse(event.data);
    console.log("Message received:", message);

    if (Array.isArray(message)) {
      message.forEach((item) => {
        this.handleIncomingData(item.topic, item.data);
      });
    } else {
      const { topic, data } = message;
      this.handleIncomingData(topic, data);
    }
  };

  this.ws.onclose = () => {
    console.log("WebSocket connection closed");
  };
}

componentWillUnmount() {
  if (this.ws) {
    this.ws.onclose = null;
    this.ws.close();
  }
}

handleIncomingData = (topic, data) => {
  if (!topic || !data) {
```

```
    console.warn("Invalid data received:", { topic, data });
    return;
  }

  console.log("Received topic:", topic, "Data:", data);

  const category = topic.startsWith("LoRa")
    ? "LoRa"
    : topic.startsWith("BLE")
    ? "BLE"
    : null;
  if (!category) return;

  this.setState((prevState) => {
    const updatedData = { ...prevState.sensorData[category] };

    let key;
    if (category === "LoRa") {
      key = data.Node;
    } else if (category === "BLE") {
      key = data.MAC;
    }

    if (key) {
      if (!updatedData[key]) {
        updatedData[key] = { latest: null, history: [] };
      }
```

```
const timestamp = new Date(); //  ใช้เวลาปัจจุบัน

console.log(`Updating ${category} - ${key}:`, {
  Temp: parseFloat(data.Temp),
  Dust: parseFloat(data.Dust),
  timestamp: timestamp.toISOString(),
});

//  อัปเดตค่าล่าสุดสำหรับ Gauge
updatedData[key].latest = {
  ...data,
  Temp: parseFloat(data.Temp),
  Dust: parseFloat(data.Dust),
  timestamp: timestamp,
};

// เพิ่มค่าลงใน history สำหรับ Graph
updatedData[key].history.push({
  Temp: parseFloat(data.Temp),
  Dust: parseFloat(data.Dust),
  timestamp: timestamp,
});

// จำกัด history ไม่ให้เก็บเกิน 100 ค่า
if (updatedData[key].history.length > 100) {
  updatedData[key].history.shift();
```

```
      }
    }

    return {
      sensorData: {
        ...prevState.sensorData,
        [category]: updatedData,
      },
    };
  });
};

renderGaugesByType(type) {
  const { sensorData } = this.state;
  const dataList = sensorData[type] || {};

  if (Object.keys(dataList).length === 0) {
    return <p>No data available for {type}</p>;
  }

  //  สลับให้ NODE1 อยู่ซ้าย และ NODE2 อยู่ขวา
  const sortedKeys = Object.keys(dataList).sort((a, b) => {
    if (a === "NODE1" && b === "NODE2") return -1; // NODE1 มา
ก่อน
    if (a === "NODE2" && b === "NODE1") return 1; // NODE2 มาหลัง
    return a.localeCompare(b); // จัดเรียงตามลำดับตัวอักษรปกติ
  });
```

```
return Object.keys(dataList).map((key) => {
  const data = dataList[key]?.latest; //  ใช้ค่า "ล่าสุด" เท่านั้น

  if (!data) return null;

  const { Temp, Dust, RSSI, SNR, Node, MAC } = data;

  return (
    <div key={key} className="sensor-column">
      <h3>
       {type} - {key}
      </h3>
      {Temp !== undefined && (
        <Plot
         data={[
          {
            type: "indicator",
            mode: "gauge+number",
            value: Temp,
            title: { text: "Temperature", font: { size: 16 } },
            number: { valueformat: ".2f", suffix: " °C" },
            gauge: {
              axis: { range: [0, 50] },
              steps: [
                { range: [0, 10], color: "#00FF00" },
                { range: [10, 20], color: "#7FFF00" },
```

```
            { range: [20, 30], color: "#FFFF00" },
            { range: [30, 40], color: "#FFAA00" },
            { range: [40, 50], color: "#FF0000" },
          ],
        },
      },
    ]}
    layout={{ width: 500, height: 400, margin: { t: 20, b: 20 } }}
  />
)}
{Dust !== undefined && (
  <Plot
    data={[
      {
        type: "indicator",
        mode: "gauge+number",
        value: Dust,
        title: { text: "Dust", font: { size: 16 } },
        number: { valueformat: ".2f", suffix: " µg/m³" },
        gauge: {
          axis: { range: [0, 100] },
          steps: [
            { range: [0, 25], color: "#00FF00" },
            { range: [25, 50], color: "#FFAA00" },
            { range: [50, 75], color: "#FF0000" },
            { range: [75, 100], color: "#6a1b9a" },
          ],
```

```
            },
          },
        ]}
        layout={{ width: 500, height: 400, margin: { t: 20, b: 20 } }}
      />
    )}
    <div className="gauge-info">
      {type === "LoRa" ? (
        <>
          <p>Node: {Node || "N/A"}</p>
          <p>RSSI: {RSSI || "N/A"}</p>
          <p>SNR: {SNR || "N/A"}</p>
        </>
      ) : (
        <p>MAC: {MAC || "N/A"}</p>
      )}
    </div>
  </div>
  );
});
}

renderGraphsByType(type) {
  const { sensorData } = this.state;
  const dataList = sensorData[type] || {};

  if (Object.keys(dataList).length === 0) {
```

```
    return <p>No graph data available for {type}</p>;
  }

  const tracesTemp = [];
  const tracesDust = [];

  //  เรียง NODE1 ก่อน NODE2
  const sortedKeys = Object.keys(dataList).sort((a, b) => {
    if (a === "NODE1" && b === "NODE2") return -1;
    if (a === "NODE2" && b === "NODE1") return 1;
    return 0;
  });

  Object.keys(dataList).forEach((key) => {
    const data = dataList[key]?.history || []; //  ใช้ค่า "ทั้งหมด" จาก
history
    if (!data.length) return;

    const timestamps = data.map((item) => new
Date(item.timestamp));
    const tempValues = data.map((item) => parseFloat(item.Temp)
|| 0);
    const dustValues = data.map((item) => parseFloat(item.Dust) ||
0);

    tracesTemp.push({
      x: timestamps,
```

```
    y: tempValues,
    mode: "lines+markers",
    name: `${key} Temperature`,
  });

  tracesDust.push({
    x: timestamps,
    y: dustValues,
    mode: "lines+markers",
    name: `${key} Dust`,
  });
});

return (
  <div className="graph-section">
    <h3>{type} - Graphs</h3>
    <Plot
      data={tracesTemp}
      layout={{
        title: `${type} - Temperature Over Time`,
        xaxis: {
          title: "Time",
          type: "date",
          range: [new Date(Date.now() - 60000 * 60), new Date()], //
แสดงข้อมูลย้อนหลัง 1 ชั่วโมง
        },
        yaxis: { title: "Temperature (°C)" },
```

```
            width: 1000,
            height: 500,
          }}
        />
        <Plot
          data={tracesDust}
          layout={{
            title: `${type} - Dust Over Time`,
            xaxis: {
              title: "Time",
              type: "date",
              range: [new Date(Date.now() - 60000 * 60), new Date()], //
แสดงข้อมูลย้อนหลัง 1 ชั่วโมง
            },
            yaxis: { title: "Dust (µg/m³)" },
            width: 1000,
            height: 500,
          }}
        />
      </div>
    );
  }

  render() {
    return (
      <div className="dashboard">
        <h1>IoT Modular Dashboard</h1>
```

```
                <div className="sensor-section">
                  <h2>LoRa Data</h2>
                  <div className="sensor-columns">
                    {this.renderGaugesByType("LoRa")}
                  </div>
                  <div className="graph-container">
                    {this.renderGraphsByType("LoRa")}
                  </div>
                </div>

                <div className="sensor-section">
                  <h2>BLE Data</h2>
                  <div className="sensor-
columns">{this.renderGaugesByType("BLE")}</div>
                  <div className="graph-container">
                    {this.renderGraphsByType("BLE")}
                  </div>
                </div>
              </div>
            );
          }
        }

        ReactDOM.render(<IoTDashboard />,
document.getElementById("root"));
        export default IoTDashboard;
```

- **Backend**

```
const express = require("express");
const mqtt = require("mqtt");
const WebSocket = require("ws");
const cors = require("cors");

// สร้าง Express app
const app = express();
app.use(cors());

// สร้าง HTTP server สำหรับ Express
const server = require("http").createServer(app);

// สร้าง WebSocket Server เพื่อส่งข้อมูลแบบเรียลไทม์ไปยัง frontend
const wss = new WebSocket.Server({ server });

// ตั้งค่าการเชื่อมต่อกับ MQTT Broker
const mqttOptions = {
  host: "o431a7ce.ala.asia-southeast1.emqxsl.com", // ใช้ hostname
ของ MQTT Broker
  port: 8883, // พอร์ต MQTT Broker (TLS/SSL)
  protocol: "mqtts", // ใช้ MQTT แบบ TLS
  username: "user_2", // Username
  password: "88888888", // Password
};
const mqttClient = mqtt.connect(mqttOptions);
```

```
// ตัวแปรเก็บข้อมูลเซ็นเซอร์ที่ได้รับจาก MQTT
let sensorData = {
  LoRa: [],
  BLE: [],
  endnode: {},
  setup: {},
};

// เมื่อเชื่อมต่อกับ MQTT Broker สำเร็จ
mqttClient.on("connect", () => {
  console.log("Connected to MQTT Broker");

  // Subscribe ครอบคลุมหัวข้อย่อยทั้งหมดด้วย wildcard #
  mqttClient.subscribe("LoRa/data");
  mqttClient.subscribe("BLE/data");
  mqttClient.subscribe("lora/endnode");
  mqttClient.subscribe("BLE/endnode");
  mqttClient.subscribe("setup/Protocol_use");
});

// รับข้อมูลจาก MQTT Broker
mqttClient.on("message", (topic, message) => {
  console.log(`Topic received: ${topic}, Message: ${message}`);
  const payload = JSON.parse(message.toString()); // แปลงข้อความ
MQTT เป็น JSON

  if (topic.startsWith("LoRa/data/")) {
```

```
    // เก็บข้อมูล LoRa แบบแยกตาม subTopic
    sensorData.LoRa.push(payload);
  } else if (topic.startsWith("BLE/data/")) {
    // เก็บข้อมูล BLE แบบแยกตาม subTopic
    sensorData.BLE.push(payload);
  } else if (topic === "lora/endnode" || topic === "BLE/endnode") {
    const nodeType = topic.split("/")[0]; // BLE หรือ LoRa
    sensorData.endnode[nodeType] = payload;
  } else if (topic === "setup/Protocol_use") {
    sensorData.setup["Protocol_use"] = payload;
  } else {
    console.log(`Unknown topic: ${topic}`);
  }

  // ส่งข้อมูลที่จัดระเบียบแล้วไปยัง WebSocket Clients (Frontend)
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(JSON.stringify({ topic, data: payload }));
    }
  });

  console.log(`Processed data on ${topic}:`, payload);
});

// เมื่อมีการเชื่อมต่อ WebSocket (Frontend)
wss.on("connection", (ws) => {
  console.log("WebSocket client connected");
```

```
  // เมื่อเชื่อมต่อ ส่งข้อมูลเซ็นเซอร์ทั้งหมดไปยัง Frontend
  ws.send(JSON.stringify(sensorData));
});

// เริ่ม HTTP server บนพอร์ต 3001
server.listen(3001, () => {
  console.log("Server is running on port 3001");
});
```