ระบบการชั่งน้ำหนักสินค้าผ่านการทำงานของระบบเมชเน็ตเวิร์คในโกดัง

Weighting goods system through mesh networking operation in warehouse

โดย

นายศิวัช อินทรศวร

นายสุทธวีร์ วิญญูรัตน์

นายเอกบุรุษ ณ น่าน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

ระบบการชั่งน้ำหนักสินค้าผ่านการทำงานของระบบเมชเน็ตเวิร์คในโกดัง

Weighting goods system through mesh networking operation in warehouses

โดย

นายศิวัช	อินทรศวร	58011226
นายสุทธวีร์	วิญญูรัตน์	58011344
นายเอกบุรุษ	ณ น่าน	58011468

อาจารย์ที่ปรึกษา

ผศ.ดร.กฤษณ์ วงรุจิระ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2561

ปริญญานิพนธ์ปีการศึกษา 2561

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบการชั่งน้ำหนักสินค้าผ่านการทำงานของระบบเมชเน็ตเวิร์คในโกดัง

Weighting goods system through mesh networking operation in warehouses

ผู้จัดทำ

1. นายศิวัช อินทรศวร 58011226
2. นายสุทธวีร์ วิญญูรัตน์ 58011344
3. นายเอกบุรุษ ณ น่าน 58011468

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. กฤษณ์ วงรุจิระ)

กิตติกรรมประกาศ

โครงงานฉบับนี้จะไม่สามารถสำเร็จลุล่วงได้เลยหากปราศจากคำแนะนำ คำสั่งสอน ความเอาใจใส่ จากอาจารย์ที่ปรึกษาซึ่งก็คือ ผศ.ดร. กฤษณ์ วงศ์รุจิระ อาจารย์ท่านนี้ได้คอยติดตามความคืบหน้า ชี้แนะแนวทางการแก้ไขปัญหาที่ขึ้นระหว่างการทำงานและยังช่วยในการแนะนำแนวทางการคิดแก้ไขปัญหา วิธีการดำเนินงาน อีกทั้งยังแนะแนวทางการปรับปรุงโครงงานให้ไปแนวทางที่ดีขึ้น

ขอขอบพระคุณอาจารย์ เพื่อนๆจากภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่คอยให้ความรู้ ความช่วยเหลือ คำแนะนำ ตลอดจนให้กำลังใจให้ข้าพเจ้าเสมอมา และที่สำคัญที่สุดข้าพเจ้าขอขอบพระคุณบิดา มารดา ครอบครัวของข้าพเจ้า ที่คอยดูแลเอาใจใส่ พร้อมคอยสนับสนุนทางด้านการศึกษาแก่เด็กคนหนึ่งมาโดยตลอด รวมไปถึงคุณครูและอาจารย์ทุกท่านที่ประสิทธิ์ประสาทวิชาให้แก่ข้าพเจ้าจนมาเป็นข้าพเจ้าในทุกวันนี้

ท้ายที่สุดทางผู้จัดทำคาดหวังว่าโครงงานเล่มนี้จะมีประโยชน์แก่ทุกท่านที่สนใจและได้เข้ามาศึกษา ไม่มากก็น้อยและถ้าโครงงานเล่มนี้เกิดข้อบกพร่อง หรือความไม่สมบูรณ์ในส่วนใดส่วนนึงของเนื้อหา ทางผู้จัดทำก็กราบขออภัยมา ณ ที่นี้ด้วย อีกทั้งพร้อมรับทุกคำแนะนำเพื่อเป็นแนวทางในการพัฒนาโครงงานอื่นๆ ในโอกาสต่อไป

นายศิวัช อินทรศวร
นายสุทธวีร์ วิญญูรัตน์
นายเอกบุรุษ ณ น่าน
ผู้จัดทำ

ระบบการชั่งน้ำหนักสินค้าผ่านการทำงานของระบบเมชเน็ตเวิร์คในโกดัง

Weighting goods system through mesh networking operation in warehouses

โดย นายศิวัช อินทรศวร 58011226
นายสุทธวีร์ วิญญูรัตน์ 58011344
นายเอกบุรุษ ณ น่าน 58011468

อาจารย์ที่ปรึกษา ผศ.ดร.กฤษณ์ วงรุจิระ

บทคัดย่อ

โครงงานนี้การออกแบบและสร้างระบบการนับจำนวนสต็อกสินค้าในโกดัง ซึ่งในปัจจุบันบริษัทและกิจการยังคงใช้แรงงานคนในการนับจำนวนสต็อคสินค้าเป็นส่วนใหญ่ โดยบริษัทและกิจการดังกล่าวต้องนับจำนวนสต็อคสินค้ากันตลอด เช่น สัปดาห์ละครั้งหรือเดือนละครั้งบ้าง ซึ่งบ่อยครั้งนั้นการนับไม่ละเอียดถี่ถ้วนพอ ทำให้เกิดปัญหาด้านการสำรองสินค้า การกระจายสินค้า ดังนั้นในโครงงานนี้จึงได้ออกแบบระบบการนับสต็อคสินค้าในโกดังเพื่อแก้ปัญหาข้างต้น โดยใช้ ESP32 ซึ่งเป็น ไอซีไมโครคอนโทรเลอร์ (MICROCONTROLLER) ที่รองรับการเชื่อมต่อ WIFI ต่อกับ LOAD CELL โดยใช้ FIREBASE เป็น SERVER ถ้าโรงงานหรือโกดังมีขนาดใหญ่การส่งข้อมูลของแต่ละ NODE ที่อยู่ไกลกันจะไม่สามารถส่งข้อมูลขึ้น FIREBASE SERVER ได้ทำให้สิ้นเปลือง ACCESS POINT หรือ ROUTER ที่ต้องใช้ติดตั้งจำนวนมาก การใช้ MESH NETWORK จะทำให้ NODE แต่ละ NODE เป็น CLIENT เชื่อมต่อถึงกันหมดส่งข้อมูลค่าน้ำหนักไปที่ NODE SERVER เพื่อส่งค่าน้ำหนักที่วัดได้ไปที่ NODE GATEWAY ก่อนส่งค่าไปยัง FIREBASE SERVER และแสดงผลไปที่หน้า WEB โดยทุกครั้งที่ส่งจะบอกวันและเวลาที่ส่งไปด้วย ทำให้ระบบการเช็คสต็อกสินค้าตามโกดังหรือโรงงานไม่ต้องใช้คนอีกต่อไป ทุกอย่างทำผ่านหน้าเว็บ และทำให้ข้อมูลต่างๆถูกจัดเก็บอย่างเป็นระบบ

ABSTRACT

This project design and build a stock counting system in warehouses. At present, the company still employ workers to count the stock of goods. The company and the business must count the stock of goods throughout, such as once a week or once a month. Frequently, the count is not thorough enough so it cause backup problems and distribution of goods so in this project, we designed a warehouse stock counting system to solve the above problem using ESP32, a microcontroller that supports WIFI connection to load cell using Firebase server. If the factories or warehouses have the large area, data transmission of each remote node will not be able to send data up to the firebase server. It can cause a lot of waste of access points or routers that require a lot of installation.Using Mesh network allows each node to be Client node and connet be connect with other Client nodes then send the wieght data to the Server node and finally send all the data in Server node which come from all Client nodes to the Gateway node before send the data to the Firebase server and display on web page Every time you send the data, it will tell the date and time that makes the stock inspection system of products in warehouses or factories no longer use people. Everything is done via the web page and makes various data stored systematically

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันมีหลายๆกิจการ ในเวลาสิ้นสัปดาห์หรือสิ้นเดือนทุกครั้งก็จะมีการตรวจนับสต็อกสินค้าอย่างละเอียดถี่ถ้วน ทั้งนี้ก็เพื่อเช็คสินค้าที่มีอยู่ให้ตรงความจริง ไม่ว่ารูปแบบการบันทึกข้อมูลเป็นการจดบันทึกในสมุดหรือบันทึกในระบบคอมพิวเตอร์ ก็มักจะมีคำถามว่า “ทำไมเมื่อเช็คสต็อกแล้วจำนวนถึงไม่ถูกต้อง” ซึ่งก็สามารถเกิดได้จากหลายๆอย่างในกระบวนการทำงาน เช่น การบันทึกข้อมูลที่ไม่ละเอียด ไม่จัดเก็บสินค้าหรือชิ้นงานให้เป็นระเบียบ การบันทึกข้อมูลไม่ถูกต้อง อีกทั้งยังเปลืองแรงงานที่ต้องมาตรวงนับสินค้า โครงงานนี้เป็นการนำเทคโนโลยี INTERNET OF THINK (IOT) มาประยุกต์ใช้งานโดยสร้างระบบวัดน้ำหนักแบบอัตโนมัติสำหรับของในโรงงาน โกดังหรือแม้แต่ห้องเก็บของขนาดเล็กโดยเพื่อค่าลดค่าใช้จ่ายและเวลาที่ใช้ในการนับสต็อกสินค้าและเพิ่มประสิทธิภาพทั้งทางด้านการผลิตและการจัดเก็บสินค้า

1.2 วัตถุประสงค์

1.) ศึกษาการทำงานของ Arduino board,ESP32 board,HX711,Load cell และโปรแกรม Arduino

2.) วัดค่าน้ำหนักของสินค้าในโกดังผ่าน ESP32 ที่ต่อร่วมผ่าน HX711 และ Load cell

3.) ทำให้ ESP32 แต่ละตัวสามารถเชื่อมต่อกันแบบ Mesh topology ได้

4.) ส่งค่าสถานะต่างๆของสินค้าขึ้น Web ได้โดยใช้ Firebase เป็นเซิฟเวอร์

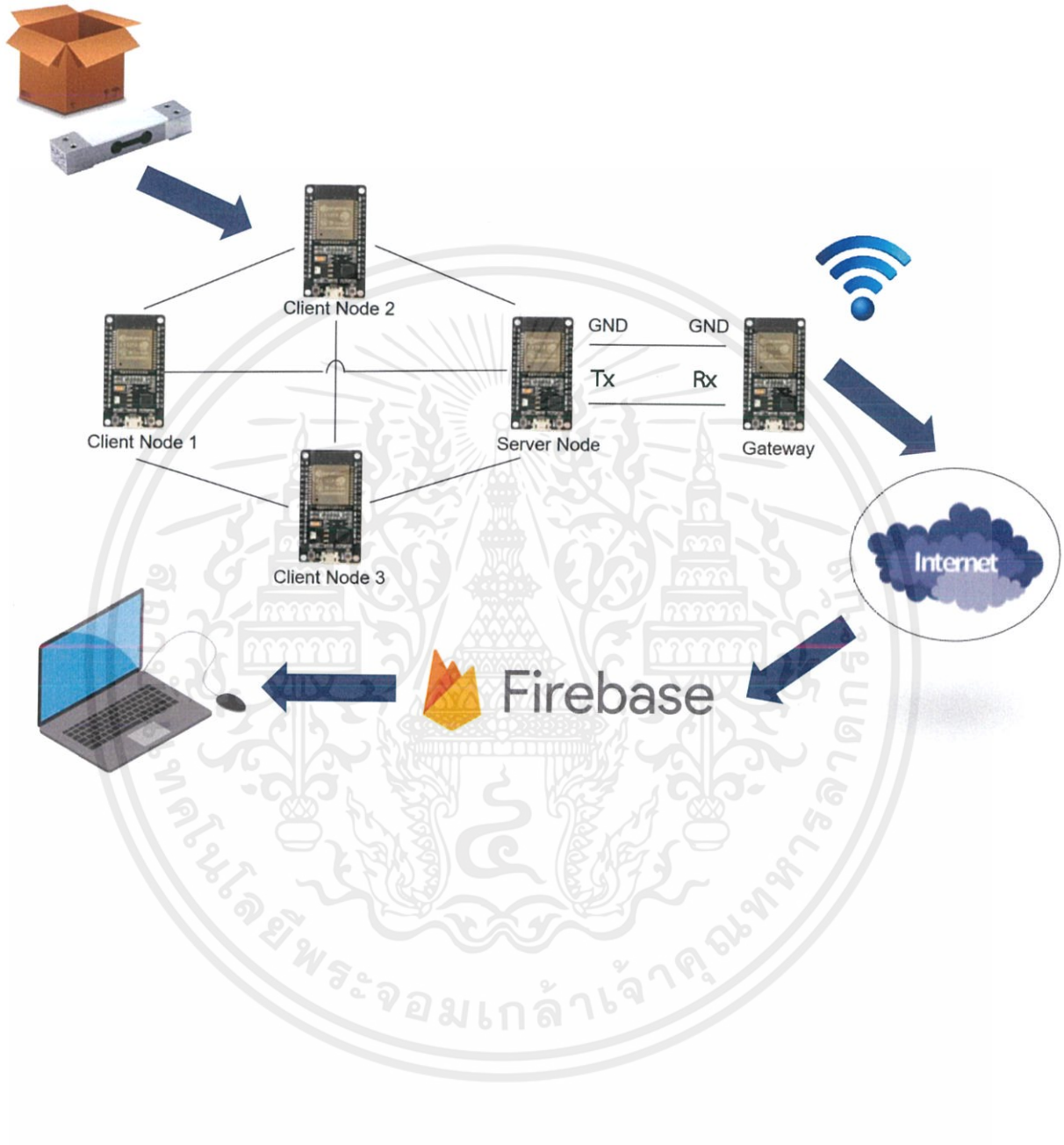
1.3 ขอบเขตของปริญญานิพนธ์

1.) วัดค่าน้ำหนักได้จาก Load cell ที่ต่อผ่าน HX711 ไปสู่ ESP32 ได้แม่นยำอย่างมีนัยสำคัญ

2.) สามารถ ทำให้ ESP32 แต่ละตัวสามารถเชื่อมต่อกันแบบ Mesh topology

3.) สามารถส่งค่าสถานะต่างๆของสินค้าขึ้น Web ได้โดยใช้ Firebase เป็นเซิฟเวอร์ได้

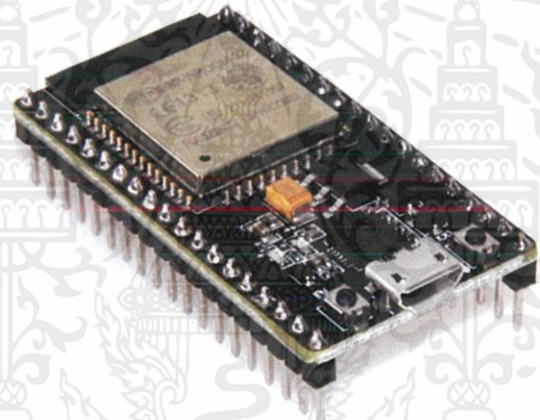
บล็อคไดอะแกรมของโครงงานที่นำเสนอ



บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 ESP32

ก่อนที่ ESP32 จะได้ถือกำเนิดขึ้น ได้มีไอซีไมโครคอนโทรลเลอร์ที่มี WiFi ในตัว และทำราคาได้ถูกมาก ๆ ในขณะนั้น (เพียง $5 หรือประมาณ 200 บาท) ออกมาปฏิวัติโลกของระบบสมองกลฝังตัว นั่นก็คือไอซีเบอร์ ESP8266 ที่ผลิตโดยบริษัท Espressif จากประเทศจีน ในช่วงเริ่มแรก ไอซี ESP8266 สามารถทำงานได้โดยใช้การสื่อสารผ่าน UART เท่านั้น และพูดคุยสั่งงานผ่าน AT command ไม่สามารถอัพเดท หรือแก้ไขเฟิร์มแวร์ด้านในได้ แต่ต่อมาไม่นานบริษัท Espressifก็ได้ออกไอซีเวอร์ชั่นใหม่มา ในครั้งนี้สามารถที่จะอัพเดทเฟิร์มแวร์ได้ และเราสามารถลงไปเขียนเฟิร์มแวร์เองได้ โดยในขณะนั้น การเขียนเฟิร์มแวร์จะใช้ภาษา C เพียงอย่างเดียว และใช้ ESP8266 SDK เป็นชุดซอฟแวร์พัฒนา ด้วยความยากของการใช้งานภาษา C เพียงอย่างเดียว ทำให้ไม่ได้รับความนิยมเรื่องการพัฒนาเฟิร์มแวร์เองมากนัก แสดงดังรูปที่ 2.1

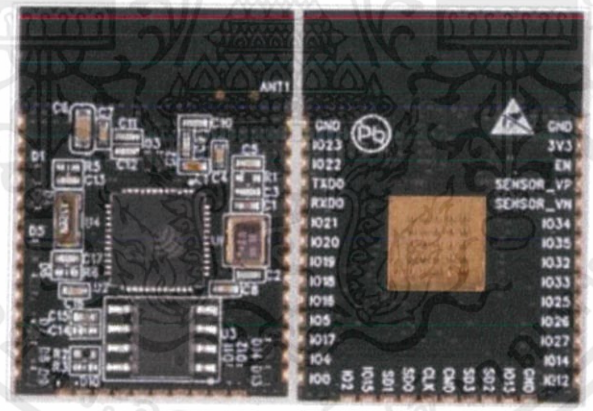


รูปที่ 2.1 ESP32 [1]

หลังจากนั้นมาประมาณ 1 ปี ผู้ผลิตบอร์ด NodeMCU ได้พอร์ตตัว Runtime ภาษา Lua มาลงใน ESP8266 ทำให้ตัว ESP8266 สามารถเขียนโปรแกรมสั่งงานตรง ๆ ได้ง่ายขึ้นมาก รวมทั้งมีเสถียรภาพเพิ่มขึ้น และในขณะนี้เอง บอร์ด NodeMCU เป็นบอร์ดพัฒนา ESP8266 สำเร็จรูปเพียงบอร์ดเดียวในตลาด ที่มาพร้อมกับ USB to UART ทำให้ให้สามารถอัพโหลดเฟิร์มแวร์เข้า ESP8266 ได้ผ่าน USB โดยตรง นอกจากนี้ผู้พัฒนาบอร์ด NodeMCU ได้คิดค้นวงจรการเข้าโหมดอัพโหลดโปรแกรมอัตโนมัติ และตั้งชื่อว่า nodemcu ซึ่งภายหลังบอร์ดพัฒนาทุกรุ่น จะใช้วงจรแบบ nodemcu ในการเข้าโหมดอัพโหลดโปรแกรมอัตโนมัติ และด้วยเหตุผลที่บอร์ด NodeMCU เป็นบอร์ดพัฒนา ESP8266 บอร์ดแรกในท้องตลาด ทำให้ได้รับความนิยมมาก และหลังจากบริษัทในจีนต่าง ๆ ได้ลอกวงจร และลายปริ้นของ NodeMCU มาทำขายเองในราคาที่ถูก แล้วใช้ชื่อเดิมคือ NodeMCU จึงทำให้บอร์ด NodeMCU ได้รับความนิยมมากจนถึงปัจจุบัน

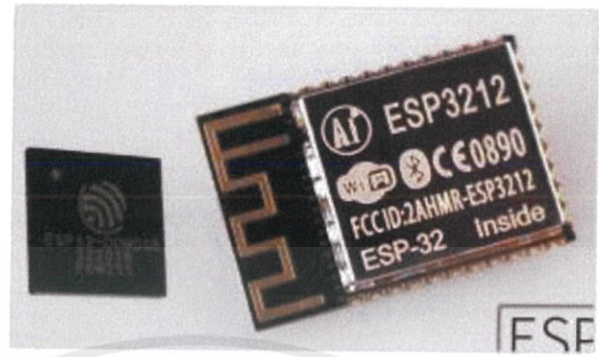
หลังจากตัว Runtime ภาษา Lua ได้ถูกพอร์ตมาลง ESP8266 ได้ประมาณ 2 – 4 เดือน ทางชุมชนพัฒนา ESP8266 ที่ชื่อ ESP8266 Community Forum (www.esp8266.com) ได้ออกชุดไลบารี่ และคอมไพล์เลอร์สำหรับใช้กับโปรแกรม Arduino IDE มาในชื่อ Arduino core for ESP8266 WiFi chip ทำให้การพัฒนาเฟิร์มแวร์ของ ESP8266 นั้นง่ายขึ้นมาก ๆ โดยใช้การเขียนโปรแกรมแบบ Arduino ดังนั้นคนที่มีพื้นฐานการเขียนโปรแกรมลงบอร์ด Arduino เป็นอยู่แล้ว จึงมาเขียนเฟิร์มแวร์ลง ESP8266 โดยใช้โปรแกรม Arduino ได้ไม่ยาก และนอกจากนี้ ไลบารี่ต่าง ๆ ที่ใช้งานได้กับบอร์ด Arduino ยังสามารถนำมาใช้งานกับ ESP8266 ได้เลยทำให้ESP8266 ได้รับความนิยมสูงมากมาจนถึงขณะนี้

ด้วยความสำเร็จอย่างถึงที่สุดของไอซี ESP8266 ทำให้บริษัท Espressif ออกไอซีรุ่นถัดไปมา ในช่วงแรกใช้ชื่อว่า ESP31B แสดงได้ดังรูปที่ 2.2 เปิดให้ร้านค้าใหญ่ๆอย่าง Adafruit SparkFun และผู้สนใจบางส่วนได้ทดสอบโดยในขณะนั้นได้มีการพัฒนาชุดซอฟแวร์ ESP32_RTOS_SDK ไปพร้อมๆกับการพัฒนาไอซี ESP31B ทำให้มีคนนำชุด ESP32_RTOS_SDK ไปพัฒนาลงโปรแกรม Arduino รอก่อนไอซีตัวจริงจะออก ในชื่อ Arduino core for ESP31B WiFi chip แต่หลังจากนั้นไม่นาน บริษัท Espressif ได้ยกเลิกการใช้ชุดซอฟแวร์พัฒนา ESP32_RTOS_SDK แล้วไปสร้างชุดพัฒนาใหม่ที่ชื่อ ESP-IDF แทน (แต่เมื่อไปเจาะลึก จะพบว่าภายในแทบจะลอก ESP32_RTOS_SDK มาทั้งหมด) จากนั้นจึงออกไอซี ESP32 ออกมาเป็นครั้งแรก



รูปที่ 2.2 ESP31B [1]

ด้วยในอดีตที่ไอซี ESP8266 ได้ท้าไว้ดีมาก จึงส่งผลให้ ESP32 ได้รับความสนใจอย่างมาก จนผลิตไม่ทันต่อความต้องการ โดยในช่วงแรก บริษัท Espressif ได้ให้ข่าวว่า จะผลิต ESP32 แบบโมดูลออกมาเพียงอย่างเดียว ในชื่อ ESP-WROOM-32 หลังจากนั้นไม่นาน บริษัท Ai-Thinker ได้ร่วมมือกับ Seeedstudio ผลิตโมดูลESP3212 ขึ้นมา โดยมีสถานะเป็นพรีออเดอร์ แต่เมื่อถึงกำหนดส่งมอบ บริษัท Seeedstudio ได้เลื่อนการส่งมอบออกไป ด้วยปัญหาด้านการออกแบบลายวงจรของตัวโมดูลเอง ทาง Ai-Thinker จึงได้ยกเลิกการผลิตESP3212 แล้วหันไปผลิต ESP32S แทน โดยลายวงจรเหมือนกับ ESP-WROOM-32 ทุกประการ แล้วจึงเริ่มส่งมอบสินค้าได้



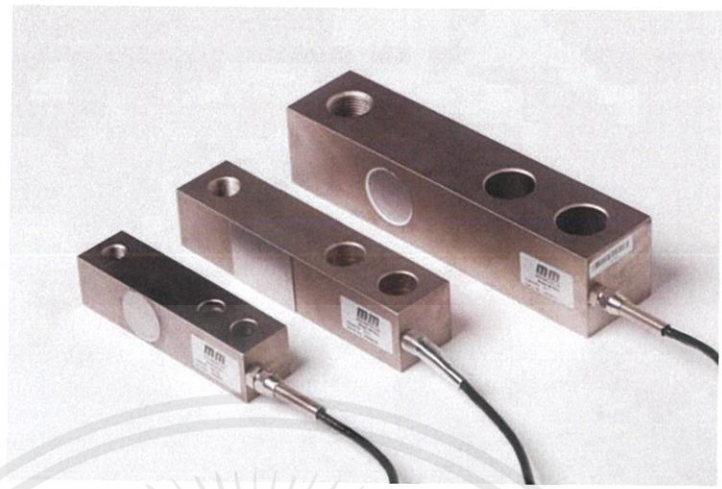
รูปที่ 2.3 หน้าตาของโมดูล ESP3212 [1]

หลังจากสินค้า ESP32S ได้เริ่มส่งมอบ ทางทีมผู้พัฒนา Arduino core for ESP8266 WiFi chip ได้ถูกบริษัท Espressif ซื้อตัวมาทั้งหมด แล้วจ้างให้พัฒนาชุดไลบารี่และคอมไพล์เลอร์สำหรับ Arduino ในชื่อ Arduino core for ESP32 WiFi chip ทำให้การพัฒนาเป็นไปด้วยความรวดเร็วมากขึ้น ภายหลังผู้พัฒนา Arduino core for ESP31B WiFi chip ก็ถูกดึงตัวให้มาร่วมทีมพัฒนา Arduino core for ESP32 WiFi chip ด้วยเช่นเดียวกัน

การพัฒนา Arduino core for ESP32 WiFi chip จะทำไปควบคู่กับการพัฒนา ESP-DF โดยที่ ESPIDF จะเป็นแกนหลัก เมื่อมีการเพิ่มฟีเจอร์ใหม่ ๆ ให้ ESP-IDF แล้ว จึงจะมีการเพิ่มใน Arduino core for ESP32 WiFi chip โดยที่ ESP-IDF รองรับการเขียนโปรแกรมแบบ Arduino เช่นเดียวกัน และรองรับทุกไลบารี่ที่ใช้ได้สำหรับ Arduino เพียงแต่ ESP-IDF ไม่มีโปรแกรม Editor โดยเฉพาะเท่านั้นเอง

2.2 Load cell

Load cell คืออุปกรณ์ที่ใช้ในการเปลี่ยนจากแรงหรือน้ำหนักที่กระทำต่อตัวโหลดเซลล์เป็นสัญญาณทางไฟฟ้า ทางเราสามารถนำสัญญาณทางไฟฟ้านี้ไปจ่ายเข้าจอแสดงผล Display แสดงค่าเป็นน้ำหนักหรือแรงที่กระทำให้คนเห็นได้ โหลดเซลล์ถูกสร้างมาจาก Strain Gauge ที่จัดเรียงวงจรในรูปแบบวงจรวิจสโตนบริดจ์ (Wheatstone Bridge) ซึ่งสามารถแปลงค่าแรงกด หรือแรงดึง ให้เป็นสัญญาณไฟฟ้า แสดงได้ดังรูปที่ 2.4



รูปที่ 2.4 Load cell [2]

Load cell สามารถเอาไปประยุกต์ทำเครื่องชั่งตวงในอุตสาหกรรมได้ (วัดแรงกด Compression) หรือใช้ทดสอบวัสดุ (วัดแรงดึง Tensile) ได้อีกด้วย การทดสอบความแข็งแรงของชิ้นงาน การทดสอบการเข้ารูปชิ้นงาน (Press fit) ใช้สำหรับงานทางด้านวัสดุ โลหะ ทดสอบโลหะ ชิ้นส่วนรถยนต์ วิศวกรรมโยธา ทดสอบคอนกรีต ทดสอบไม้ ฯลฯ ซึ่งมีความจำเป็นอย่างมากงานภาคอุตสาหกรรม

2.2.1 ประเภทของโหลดเซลล์ สามารถแบ่งได้เป็น 5 ประเภท

2.1.1 โหลดเซลล์แบบสเตรนเกจ (Strain Gauge Load Cell)

สเตรนเกจ คือเครื่องมือที่ใช้ในการตรวจวัดแรงดึงเครียด (Strain) ของวัตถุส่วนใหญ่สเตรนเกจจะทำจากลวดโลหะขนาดเล็กขดเป็นรูปร่างต่างๆอยู่บนแผ่นฉนวนนอกจากนั้นยังมีสเตรนเกจแบบอุปกรณ์กึ่งตัวนำด้วยซึ่งมีความไวสูงกว่าและขนาดเล็กกว่าแบบลวดโลหะแต่ก็มีราคาแพงกว่าเช่นกัน ถึงแม้ว่าโหลดเซลล์ชนิดสเตรนเกจจะเป็นชนิดที่มักจะถูกใช้งานก็ตาม แต่ก็มีโหลดเซลล์ชนิดอื่นเช่นกัน ในการประยุกต์ใช้งานเชิงอุตสาหกรรม ไฮดรอลิคนับได้ว่าได้รับความนิยมเป็นอันดับที่สองและถูกใช้ประโยชน์เพื่อแก้ปัญหาที่มาจากโหลดเซลล์ชนิดสเตรนเกจ ตัวอย่างเช่น ไฮดรอลิคโหลดเซลล์สามารถทนทานต่อความต่างศักย์ฉับพลันได้(ฟ้าผ่า) จึงนับว่ามีประสิทธิภาพการใช้งานในสภาวะนอกตัวอาคารได้ดีกว่า โดยโหลดเซลล์แบบสเตรนเกจก็แบ่งเป็นอีก 2 ประเภทใหญ่ คือ โหลดเซลล์แบบใช้แรงกด ออกแบบมาเพื่อใช้แรงกดลงบนตัวโหลดเซลล์ และ โหลดเซลล์แบบใช้แรงดึง ออกแบบมาเพื่อใช้แรงดึงตัวโหลดเซลล์ออกจากกัน

1.) โหลดเซลล์แบบใช้แรงกด มีชื่อเรียกตามรูปร่างและการใช้งาน ได้แก่

- Single End Shear Beam นิยมใช้ในการชั่งน้ำหนักในถังน้ำหนักตั้งแต่ 250 กิโลกรัม ถึง 10 ตัน เช่น การชั่งน้ำหนักหิน,ทรายในถังก่อนปล่อยลงไปผสมกับซิเมนต์และน้ำในแพลนคอนกรีต เป็นต้น โหลดเซลล์ประเภทนี้ใช้งานโดยยึดปลายด้านหนึ่งเข้ากับฐานและนำถังวางลงบนปลายอีกด้านหนึ่งแสดงได้ดังรูปที่ 2.5

รูปที่ 2.5 Single End Shear Beam [2]

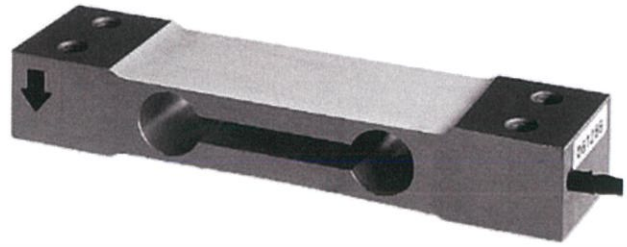
- Double End Shear Beam

โหลดเซลล์ประเภทนี้เหมือนกับนำ Single End Shear Beam จำนวน 2 ตัวมารวมกัน ซึ่งทำให้มีจำนวนสเตรนเกจมากขึ้น ทำให้ได้ความละเอียดมากขึ้นการติดตั้งเป็นการยึดปลายทั้งสองข้างด้วยสกรูติดกับฐาน แล้วนำถังมาวางตรงกลาง โดยมีลูกบอลและเบ้ายึดตัวถังและโหลดเซลล์ เพื่อให้ถังสามารถขยับได้แต่ไม่ หลุดหล่นไป โหลดเซลล์ประเภทนี้นิยมใช้ในงานชั่งที่มีน้ำหนักมาก เช่น ถังหรือไซโล ที่มีขนาดใหญ่ ตั้งแต่ 10 ตัน ถึง 50 ตัน โดยจะติดตั้งไว้ที่ขาของถังหรือไซโล แสดงได้ดังรูปที่ 2.6

รูปที่ 2.6 Double End Shear Beam [2]

- Single point

ออกแบบมาสำหรับงานขนาดเล็ก น้ำหนักตั้งแต่ 2 กิโลกรัม ถึง 800 กิโลกรัม ใช้งานโดยยึดโหลดเซลล์เข้าที่จุดศูนย์กลาง แสดงได้ดังรูปที่ 2.7



รูปที่ 2.7 Single point loadcell [3]

- Bending Beam

โหลดเซลล์ประเภทนี้มีโครงสร้างคล้ายสปริง ทำงานโดยการแปลงแรงบิดที่กดที่ปลายด้านหนึ่ง ซึ่งจะให้สัญญาณได้ดีที่ขนาดแรงกดไม่มาก ตั้งแต่ 25 กิโลกรัม ถึง 500 กิโลกรัม แสดงได้ดังรูปที่ 2.8

รูปที่ 2.8 Bending Beam loadcell [2]

- Pancake

ชื่อเรียกก็มาจากรูปทรงกลมของโหลดเซลล์ประเภทนี้ Pancake เป็นโหลดเซลล์ที่ใช้ได้ทั้งกับแรงกดและแรงดึง ขนาดตั้งแต่ 500 กิโลกรัม ถึง 500 ตัน เป็นโหลดเซลล์ที่มีความแม่นยำสูง โดยค่า Linearity และ Hysteresis อยู่ในระดับ 0.05% เนื่องจากมีจำนวนสเตนนเกจมากกว่าโหลดเซลล์ชนิดอื่น แสดงได้ดังรูปที่ 2.9

รูปที่ 2.9 Pancake loadcell [3]

- Canister

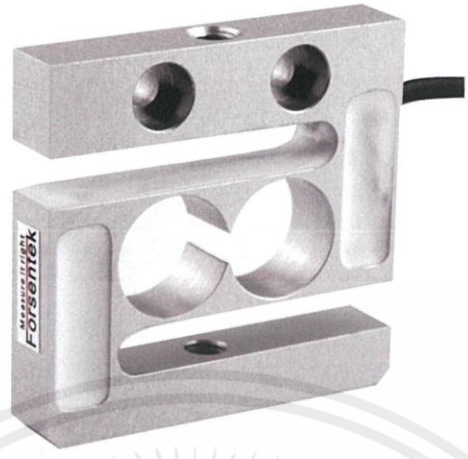
โหลดเซลล์รูปทรงกระบอก เหมือนกระป๋อง ใช้รับแรงกด มีความแม่นยำสูง โดยค่า Linearity และ Hysteresis อยู่ในระดับ 0.05% จึงนิยมใช้ทำเครื่องชั่งทั่วไปที่ต้องการความแม่นยำสูงๆ ไปจนถึงเครื่องชั่งรถบรรทุก มีขนาดตั้งแต่ 200 กิโลกรัม ถึง 20 ตัน แสดงดังรูปที่ 2.10

รูปที่ 2.10 Canister loadcell [2]

2.) โหลดเซลล์แบบใช้แรงดึง

มีชื่อเรียกตามรูปร่างและการใช้งาน ได้แก่

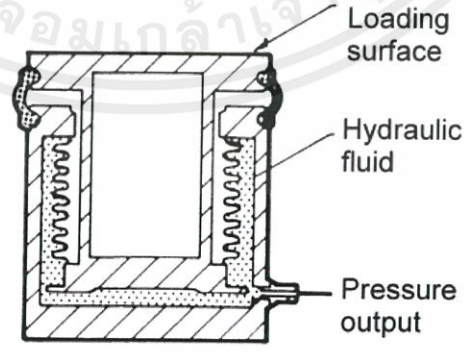
- S beam โหลดเซลล์ประเภทนี้ใช้งานโดยแขวนถังที่ต้องการชั่งที่ด้านล่าง มีขนาดตั้งแต่ 2 กิโลกรัม ถึง 5 ตัน แสดงได้ดังรูปที่ 2.11



รูปที่ 2.11 S beam loadcell [2]

3.) โหลดเซลล์แบบไฮดรอลิค (Hydraulic Load Cell)

โหลดเซลล์แบบไฮดรอลิค วัดน้ำหนักจากการเปลี่ยนแปลงความดันของของเหลวภายในระบบเมื่อมีแรงมากระทำที่แท่นรับน้ำหนักในโหลดเซลล์แบบไฮดรอลิคที่มีแผ่นไดอะแฟรมม้วน แรงจะถูกส่งผ่านลูกสูบเป็นผลให้ของเหลวภายในช่องแผ่นไดอะแฟรมถูกกดอัด การวัดแรงที่เกิดขึ้นสามารถวัดได้จากความดันของของเหลว ความสัมพันธ์ระหว่างแรงกระทำกับแรงดันของของเหลวนี้ มีลักษณะเป็นแบบเชิงเส้น โดยไม่ขึ้นกับอุณหภูมิและปริมาณของของเหลวในกระบอกสูบ ถ้าโหลดเซลล์แบบนี้ได้รับการติดตั้งหรือสอบเทียบที่เหมาะสม ความแม่นยำในการวัดควรจะอยู่ที่ 0.25% ของช่วงการวัดเต็มสเกลหรือดีกว่านั้น ระดับความแม่นยำนี้เป็นที่ยอมรับได้ในงานอุตสาหกรรมทั่วไป เนื่องจากเครื่องมือวัดแบบนี้ไม่ต้องใช้ไฟฟ้า จึงเหมาะที่จะใช้ในพื้นที่ที่อันตราย ข้อเสียของโหลดเซลล์แบบไดอะแฟรมนี้ คือสามารถรับแรงสูงสุดได้ไม่เกิน 1000 psig ในงานที่ต้องการวัดแรงดันสูงจะต้องใช้โหลดเซลล์แบบที่มีไดอะแฟรมทำด้วยโลหะซึ่งสามารถรับน้ำหนักได้ถึง 10,000,000 ปอนด์ แสดงได้ดังรูปที่ 2.12



รูปที่ 2.12 Hydraulic Load Cell [4]

โดยทั่วไปแล้วโหลดเซลล์แบบไฮดรอลิคมักจะใช้ในการวัดน้ำหนักถังเก็บวัสดุหรือแทงก์ สำหรับกรณีที่ต้องการความแม่นยำสูงสุด การวัดควรจะใช้โหลดเซลล์หลายตัวมาวัดที่จุดรองรับแต่ละตำแหน่ง เนื่องจากการกำหนดระนาบจะต้องใช้จุด 3 จุด ดังนั้นการวัดน้ำหนักวัตถุที่มีขนาดใหญ่จึงควรจะใช้โหลดเซลล์ 3 ตัวเพื่อวัดน้ำหนัก ณ จุดรองรับวัตถุทั้ง 3 จุด น้ำหนักของวัตถุจะหาได้จากผลรวมของค่าน้ำหนักที่อ่านได้จากโหลดเซลล์ทั้งสามนั่นเอง

4.) โหลดเซลล์แบบนิวแมติก (Pneumatic Load Cell)

โหลดเซลล์แบบนิวแมติก ทำงานโดยใช้หลักการสมดุลแรงเช่นเดียวกับแบบไฮดรอลิค โหลดเซลล์แบบนี้มีความแม่นยำกว่าแบบไฮดรอลิค เพราะมีการใช้ช่องว่างหลายช่องในการหน่วงความดันของของเหลวและลดการสั่นสะเทือน โหลดเซลล์แบบนิวแมติกนี้มักจะใช้วัดสิ่งของที่มีน้ำหนักไม่มากนักในงานอุตสาหกรรมที่ต้องการความสะอาดและความปลอดภัยสูง จุดเด่นของโหลดเซลล์แบบนี้คือสามารถทนแรงกระแทกได้สูงและไม่ไวต่อการเปลี่ยนแปลงของอุณหภูมิ นอกจากนี้ระบบนิวเมติกไม่ใช้ของเหลวในเครื่องมือวัดเหมือนระบบไฮดรอลิคทำให้มั่นใจได้ว่าจะไม่มีของเหลวมาปนเปื้อนสิ่งที่ต้องการจะวัดถ้าไดอะแฟรมมีการแตกร้าว อย่างไรก็ตาม โหลดเซลล์แบบนี้มีข้อเสียคือความเร็วในการตอบสนองต่ำและต้องใช้งานในสภาวะแวดล้อมที่สะอาดปลอดความชื้นและจะต้องมีการควบคุมอากาศหรือไนโตรเจนภายในเครื่องให้เหมาะสม

5.) โหลดเซลล์แบบไพโซรีซิสทีฟ (Piezoresistive)

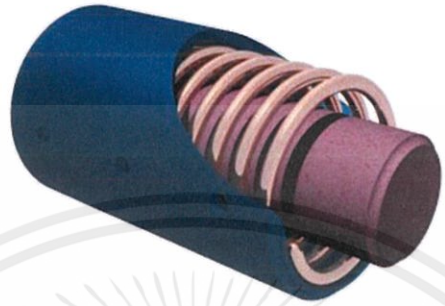
ไพโซรีซิสทีฟมีการทำงานเหมือนกับเกจวัดความเครียด แต่ไพโซรีซิสทีฟสามารถผลิตสัญญาณออกมาได้ในระดับสูงจึงเหมาะสำหรับเครื่องชั่งน้ำหนักที่ไม่ซับซ้อนในการวัด เนื่องจากสามารถต่อเข้าโดยตรงกับส่วนแสดงผล อย่างไรก็ตามเครื่องมือวัดลักษระนี้ได้รับความนิยมลดลงเรื่อยๆ เพราะตัวขายสัญญาณที่มีคุณภาพดีนั้นมีราคาถูกลง นอกจากนี้ไพโซรีซิสทีฟยังมีข้อเสียคือความสัมพันธ์ระกว่างสัญญาณที่ออกกับน้ำหนักที่วัดมีลักษณะไม่เป็นเชิงเส้น

อุปกรณ์ที่ใช้หลักการการเหนี่ยวนำสนามแม่เหล็ก อุปกรณ์ลักษณะนี้จะตรวจวัดการเคลื่อนที่ของแกนแม่เหล็ก โดยการวัดการเหนี่ยวนำของขดลวดแม่เหล็กไฟฟ้าที่เปลี่ยนไป ในที่นี้การเคลื่อนที่ของแกนเหล็กจะแปรผันโดยตรงกับน้ำหนักที่วัดนั่นเอง แสดงได้ดังรูปที่ 2.13

รูปที่ 2.13 Piezoresistive Loadcell [3]

6.) โหลดเซลล์แบบแมกเนโตสเตร็กทีฟ (Magmetostrictive)การทำงานของเซนเซอร์นี้ขึ้นอยู่กับการเปลี่ยนแปลงในการแผ่สัญญาณแม่เหล็กของแม่เหล็กถาวรที่อยู่ภายใต้แรงที่มากระทำ

แรงทำให้เกิดการผิดรูปของสนามแม่เหล็กและจะทำให้เกิดสัญญาณที่เป็นสัดส่วนโดยตรงต่อแรงที่มากระทำ อุปกรณ์ตรวจวัดนี้มีความทนทานมากและยังคงมีใช้อยู่มากโดยเฉพาะอย่างยิ่งในอุตสาหกรรมที่มีการรีดโลหะแผ่น แสดงได้ดังรูปที่ 2.14



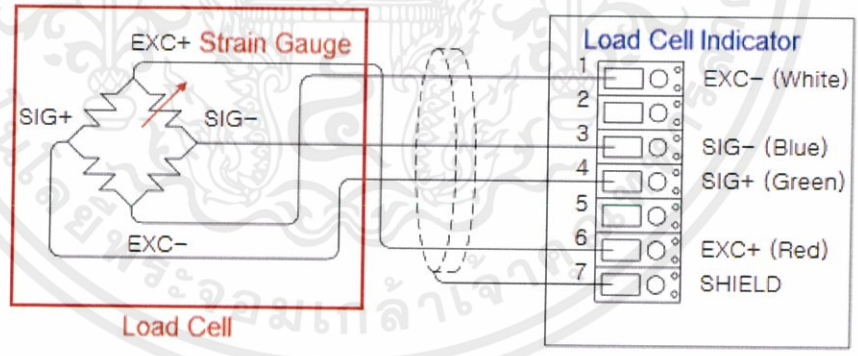
รูปที่ 2.14 Magmetostrictive Loadcell [3]

2.2.2 หลักการทำงานของโหลดเซลล์

หลักการทำงานของ Weighing Indicator, Load Cell Indicator หรือ Strain Gauge Amplifier มีการทำงานแบ่งออกเป็น 2 ส่วน

1.) วงจรภายนอก

จะเป็นส่วนของตัวเซ็นเซอร์ ที่ใช้ต่อกับตัว Weighing Indicator จะประกอบไปด้วย Strain gauge และตัวความต้านทาน ที่ต่ออยู่ในวงจรบริดจ์แบบไม่สมดุล แสดงได้ดังรูปที่ 2.15



รูปที่ 2.15 ภาพวงจรภายนอก (Load Cell) ที่ต่อกับ Load Cell Indicator [5]

- วงจรภายใน จะเป็นส่วนที่อยู่กับตัว Load Cell Indicator จะประกอบไปด้วยส่วนต่างๆ มากมาย โดยแบ่งเป็น

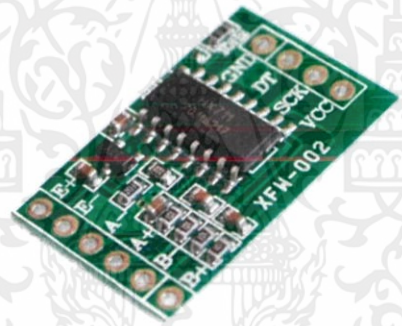
a. ส่วนของอินพุต ที่ทำหน้าที่รับสัญญาณจากโหลดเซลล์ เพื่อทำการแปลงค่าเป็นสัญญาณ Digital หรือ A/D ความละเอียดสูง

b. ส่วนต่อมาเป็นการประมวลผลค่าดิจิตอลที่ได้จากการทำ A/D ซึ่งจะนำสัญญาณที่ได้มาเตรียมข้อมูลเพื่อที่จะแสดงผล หรือกระทำตามลอจิก Logic เงื่อนไขต่างๆ ที่ผู้ใช้งานได้สร้างเอาไว้

c. ส่วนสุดท้ายเป็นภาคเอาท์พุตที่จะคอยส่งต่อสัญญาณที่ได้จากการประมวลผลแล้ว ไม่ว่าจะเป็นหน้าจอตัวเลขที่แสดงค่าสัญญาณอนาล็อกเอาท์พุตที่คอยส่งออกไปใช้งานต่อในอุปกรณ์หรือตัวควบคุมอื่นๆ เช่น PLC หน้าคอนแทครีเลย์ที่ใช้ควบคุมอุปกรณ์ต่างๆ ให้เป็นไปตามเงื่อนไขที่ได้ตั้งค่าไว้ เช่น ควบคุม Solenoid Valve, Cylinder, Alarm

2.3 HX711

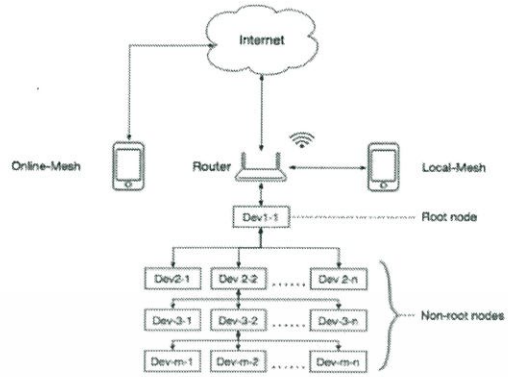
โมดูล HX711 เป็นโมดูลสำหรับขยายสัญญาณจาก Load Cell เซ็นเซอร์วัดน้ำหนัก ซึ่งปกติมาค่าน้อยมากๆ ตัวโมดูลนี้จะขยายสัญญาณ ออกเป็นสัญญาณดิจิตอล 24bit I2C ทำให้สามารถนำ Arduino NodeMCU Raspberry Pi หรือ MCU อื่นๆมาอ่านค่าน้ำหนักได้ง่ายๆ สายเชื่อมต่อเข้า MCU มี 2 เส้นและไฟเลี้ยง 2 เส้น สามารถใช้ไฟเลี้ยงได้ตั้งแต่ 2.6-5.5v และอีกด้านของโมดูลสามารถต่อกับ Load Cell ได้เลย แสดงได้ดังรูปที่ 2.16



รูปที่ 2.16 HX711 [6]

2.4 การทำงานแบบ Mesh network

Mesh Network เนี่ยมันก็คือโครงสร้างของ Network แบบหนึ่งที่มีลักษณะเป็น Node ซึ่งเชื่อมต่อถึงกันหมดและทำงานร่วมกันในการส่งผ่านข้อมูลจาก node นึงไปยังอีก node นึง ซึ่งการทำงานในการส่งผ่านข้อมูลตรงนี้ที่ไม่ขึ้นตรงต่อกันแบบตายตัว นั่นหมายความว่าโครงสร้างของ mesh นั้นสามารถปรับเปลี่ยนได้ตลอดเวลา คุยกันตลอด เช่นถ้าจะส่งข้อมูลจากจุด A ไปจุด B จะไปเส้นทางไหน ถ้าเส้นทางจาก A –>B–>C–>D แล้วถ้า Node B เกิดล่ม จะ route ข้อมูลไปที่ Node ไหนต่อเพื่อให้ถึงปลายทาง (Self-Organize / Self-Configure) ซึ่งนั่นหมายความว่าเราไม่จำเป็นที่จะต้อง config routing และสามารถที่จะขยายพื้นที่การใช้งานออกไปได้ไกลมากขึ้นตาม node ปลายทางที่เชื่อมถึงกันด้วย การทำงานแบบ mesh network สามารถแสดงได้ดังรูปที่ 2.17



รูปที่ 2.17 การทำงานแบบ mesh network [7]

Mesh Network นั้นทำให้การเชื่อมต่อแบบ wireless นั้นทำได้ง่ายขึ้นและมีประสิทธิภาพมากขึ้น โดยใช้อุปกรณ์ที่มีอยู่ในทุกวันนี้มาใช้งาน ซึ่งโดยปกติแล้ว การให้บริการ wireless ในปัจจุบันนั้นทำได้โดยการใช้ Access Point เชื่อมต่อด้วย infrastructure แบบสาย (wired) เพื่อให้บริการ แต่ใน wireless mesh network นั้น การเชื่อมต่อนั้นถูกขยายไปด้วย Access Point เอง ไม่ว่าจะเป็นจำนวนน้อยๆหรือหลายๆร้อยตัวก็ตาม โดยแต่ละ Access Point นั้นเราอาจจะเรียกได้ว่าเป็น node ซึ่งแต่ละโหนดนั้นก็จะทำการสื่อสารกับ node ที่อยู่ใกล้เคียงอัตโนมัติ เพื่อกระจายเน็ตเวิร์คออกไปให้ไกลที่สุด

โดย node ใน mesh network จะมีตัวส่งสัญญาวิทยุ ที่อาจจะเรียกได้ว่าทำงานเหมือนกัน wireless router ในปัจจุบันนี้ node นั้นก็ส่งข้อมูลในมาตรฐาน 802.11a,b,g เพื่อส่งข้อมูลกับผู้ใช้งานและส่งข้อมูลกับอุปกรณ์ตัวอื่น Nodes นั้นจะถูกโปรแกรมมาให้ทำการสื่อสาร และโต้ตอบกับอุปกรณ์อื่นๆ ในเน็ตเวิร์คขนาดใหญ่ โดย node นั้นจะทำการเลือกเส้นทางอัตโนมัติว่าเส้นทางไหนที่จะส่งข้อมูลได้เร็วที่สุด โดยเราจะเรียกวิธีการนี้ว่า dynamic routingข้อดีที่สุดของ mesh network อย่างหนึ่ง เมื่อเราเปรียบเทียบกับ เน็ตเวิร์คแบบสายและการใช้ Access Point แบบเดิมคือมันทำงานแบบไร้สายทั้งหมด แทนที่จะทำการเชื่อมแต่ละ access point เข้าด้วยสายแต่ละจุด ในระบบ mesh network เราก็จะใช้สายเพียงเส้นเดียวเท่านั้นเพื่อเชื่อมต่อกับเน็ตเวิร์คหลัก และแน่นอนว่าการเข้าถึงอินเตอร์เน็ตทั้งหมดของทุก nodes ก็จะต้องแชร์กัน

2.4.1 ข้อดีของ Wireless Mesh network

- ใช้ระบบสายน้อยลงนั่นหมายถึงต้นทุนน้อยลงโดยเฉพาะเมื่อต้องการวางระบบเน็ตเวิร์คที่ครอบคลุมเนื้อที่กว้าง

- การใช้จำนวน nodes ที่เพิ่มขึ้นหมายถึง ความเร็วในการส่งข้อมูลภายในเพิ่มขึ้น

- สะดวกโดยเฉพาะในที่ที่การวางสายบางครั้งเป็นเรื่องลำบากเช่นในสนามกีฬาขนาดใหญ่ โรงงาน การคมนาคม

- เหมาะอย่างยิ่งสำหรับบางสถานที่ที่ในบางครั้งสัญญาณจะโดน block เช่น ในสวนสนุกเมื่อบางครั้ง เครื่องเล่นวิ่งผ่าน สัญญาณขาดหายบางช่วง แต่ถ้าเป็น mesh network การสื่อสารจะใช้ node อื่นๆ ข้างเคียงส่งข้อมูลแทนได้

- Mesh network นั้นเรียกได้ว่าเป็นเน็ตเวิร์คที่ “ปรับตัวเอง” คือ รู้จัก node ใหม่อัตโนมัติ โดยไม่ต้องอาศัย admin

- Mesh network ซ่อมแซมตัวเองได้กรณีที่ node บางตัวไม่สามารถทำงานได้ การค้นหา และเปลี่ยนเส้นทางไปยัง node อื่นๆ ข้างเคียงเกิดขึ้นโดยอัตโนมัติ

- ติดตั้งง่าย ขยายง่าย

2.5 Firebase

Firebase คือ Platform ที่รวบรวมเครื่องมือต่าง ๆ สำหรับการจัดการในส่วนของ Backend หรือ Server side ซึ่งทำให้สามารถ Build Mobile Application ได้อย่างมีประสิทธิภาพ และยังลดเวลาและค่าใช้จ่ายของการทำ Server side หรือการวิเคราะห์ข้อมูลให้อีกด้วย โดยมีทั้งเครื่องมือที่ฟรี และเครื่องมีที่มีค่าใช้จ่าย (สำหรับการ Scale) Firebase มีบริการหลายอย่างมาก ๆ จะขอยกตัวอย่างบางส่วน โดยแบ่งหัวข้อของ Firebase ดังนี้

2.5.1 Build better apps

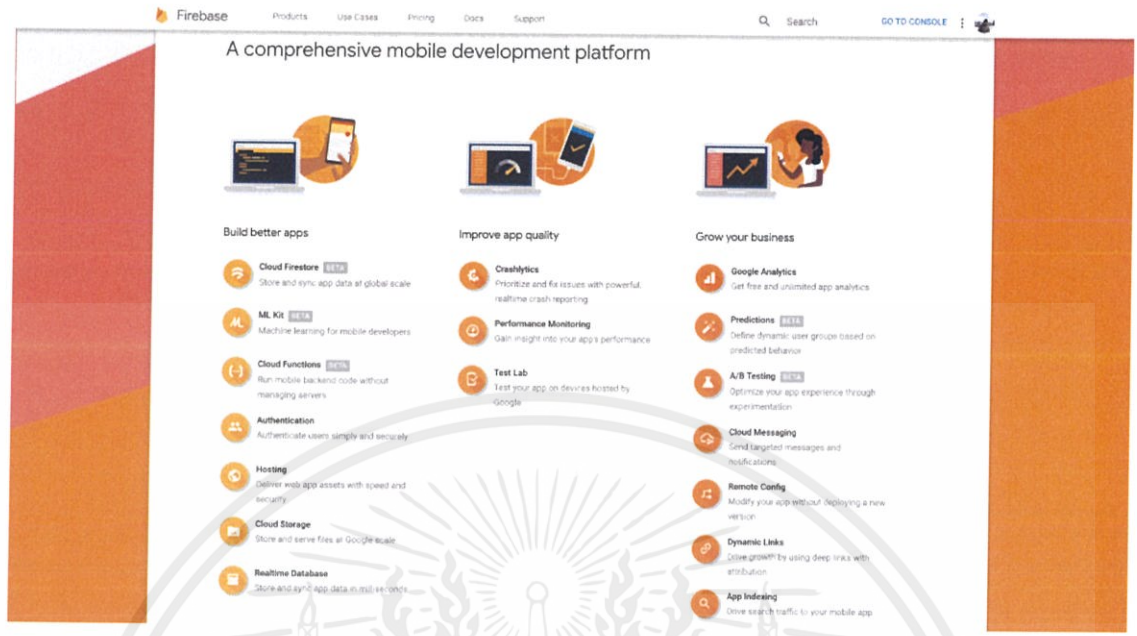
Cloud Firestore คือ บริการทางด้าน Database ที่เป็นลักษณะเป็น NoSQLโดยนำข้อดีของ Realtime Database ของ Firebase เช่นกัน มาต่อยอดอีกด้วย Authentication ชื่อก็บอกอยู่แล้ว ใช่ครับ คือบริการที่จัดการ Auth ให้เรา ซึ่งคลอบคลุมมาก ๆ ทั้ง email-password, phone ไปจนถึง facebook, twitter, github สำหรับการ Login อีกด้วย Hosting คือ hosting สำหรับ single-page web app, landing page website ซึ่งจัดการการ Deploy ให้ และในส่วนของ Custom Domain (ไม่ฟรี) ก็มี การติดตั้ง SSL ให้ด้วย

2.5.2 Improve app quality

Crashlytics ช่วยจัดการ Issue ต่าง ๆ และสามารถตรวจจับ Crash ได้ว่าเกิดขึ้นที่การทำงานไหนใน Mobile App แต่เดิมเริ่มต้นพัฒนาจากทีมงานของ Fabric ซึ่งมีผู้ใช้จำนวนมาก Performance Monitoring สรรพคุณตามชื่อเช่นกัน โดยผู้พัฒนาสามารถ ทราบถึง Performance ของ Code และ Network

2.5.3 Grow your business

Google Analytics คือ ตัวที่เก็บข้อมูลสถิติ พฤติกรรมของ User ที่ใช้งาน Mobile App (Web ก็ใช้ได้นะ) โดยสามารถแบ่งพฤติกรรมให้เราดูได้อย่างละเอียด Remote Config คือ ส่วนที่จัดการรูปแบบของ Mobile App ในเรื่องของ หน้าตา เช่น หากเราต้องการเปลี่ยนภาพ Background ในหน้า Main เราก็สามารถ เปลี่ยนได้ที่ Remote Config นี้ได้เลย ไม่ต้องไปแก้ที่ Code ของ Mobile App Cloud Messaging คือ ตัวที่จะทำให้ Mobile App ของเรารับ Notification ได้โดยส่ง Message ไปหาได้ทุก Platform ทั้ง iOS และ Android รวมไปถึง Web จากที่กล่าวมาทั้งหมดนี้สามารถแสดงได้ดังรูปที่ 2.18



รูปที่ 2.18 Firebase

2.6 HTML

HTML คือ ภาษาหลักที่ใช้ในการเขียนเว็บเพจ โดยใช้ Tag ในการกำหนดการแสดงผล HTML ย่อมาจากคำว่า Hypertext Markup Language โดย Hypertext หมายถึง ข้อความที่เชื่อมต่อกันผ่านลิ้งค์ (Hyperlink) Markup language หมายถึงภาษาที่ใช้ Tag ในการกำหนดการแสดงผลสิ่งต่างๆที่แสดงอยู่บนเว็บเพจ ดังนั้น HTML จึงหมายถึง ภาษาที่ใช้ Tag ในการกำหนดการแสดงผลเว็บเพจที่ต่างก็เชื่อมถึงกันใน Hyperspace ผ่าน Hyperlink นั่นเอง

ความเป็นมาของ HTML เริ่มขึ้นเมื่อปี 1980 เมื่อ Tim Berners Lee เสนอต้นแบบสำหรับนักวิจัยใน CERN เพื่อแลกเปลี่ยนเอกสาร ข้อมูลด้านการวิจัย โดยใช้ชื่อว่า Enquire ในปี 1990 เค้าได้เขียนโปรแกรมบราวเซอร์ และทดลองรันบนเซิฟเวอร์ที่เค้าพัฒนาขึ้น HTML ได้รับการรู้จักจาก HTML Tag ซึ่งมีอยู่ 18 Tag ในปี 1991

HTML ถูกพัฒนาจาก SGML และ Tim ก็คิดเสมือนว่า HTML เป็นโปรแกรมย่อยของ SGML อยู่ในตอนนั้น ต่อมาในปี 1996 เพื่อกำหนดมาตรฐานให้ตรงกัน W3C World Wide Web Consortium จึงเป็นผู้กำหนดสเปกทั้งหมดของ HTML และปี 1999 HTML 4.01 ก็ถือกำเนิดขึ้นโ ด ย มี HTML 5 ซึ่งเป็น Web Hypertext Application ถูกพัฒนาต่อมาในปี 2004 นอกจากนี้ยังมีการพัฒนาไปเป็น XHTML ซึ่ง คือ Extended HTMLซึ่งมีความสามารถและมาตรฐานที่รัดกุมกว่าอีกด้วย

2.7 Java

Java หรือ Java programming language คือภาษาโปรแกรมเชิงวัตถุ พัฒนาโดย เจมส์ กอสลิง และวิศวกรคนอื่นๆ ที่บริษัท ซัน ไมโครซิสเต็มส์ ภาษานี้มีจุดประสงค์เพื่อใช้แทนภาษาซีพลัสพลัส C++ โดยรูปแบบที่เพิ่มเติมขึ้นคล้ายกับภาษาอ็อบเจกต์ทีฟซี (Objective-C) แต่เดิมภาษานี้เรียกว่าภาษาโอ๊ก (Oak) ซึ่งตั้งชื่อตามต้นโอ๊กใกล้ที่ทำงานของ เจมส์ กอสลิง แล้วภายหลังจึงเปลี่ยนไปใช้ชื่อ "จาวา" ซึ่งเป็นชื่อกาแฟแทน จุดเด่นของภาษา Java อยู่ที่ผู้เขียนโปรแกรมสามารถใช้หลักการของ Object-Oriented Programming มาพัฒนาโปรแกรมของตนด้วย Java ได้

ภาษา Java เป็นภาษาสำหรับเขียนโปรแกรมที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ (OOP : Object-Oriented Programming) โปรแกรมที่เขียนขึ้นถูกสร้างภายในคลาส ดังนั้นคลาสคือที่เก็บเมทอด (Method) หรือพฤติกรรม (Behavior) ซึ่งมีสถานะ (State) และรูปพรรณ (Identity) ประจำพฤติกรรม (Behavior)

2.8 JavaScript

JavaScript คือ ภาษาคอมพิวเตอร์สำหรับการเขียนโปรแกรมบนระบบอินเทอร์เน็ต ที่กำลังได้รับความนิยมอย่างสูง Java JavaScript เป็น ภาษาสคริปต์เชิงวัตถุ (ที่เรียกกันว่า "สคริปต์" (script) ซึ่งในการสร้างและพัฒนาเว็บไซต์ (ใช่ร่วมกับ HTML) เพื่อให้เว็บไซต์ของเราดูมีการเคลื่อนไหว สามารถตอบสนองผู้ใช้งานได้มากขึ้น ซึ่งมีวิธีการทำงานในลักษณะ "แปลความและดำเนินงานไปทีละคำสั่ง" (interpret) หรือเรียกว่า Object Oriented Programming ที่มีเป้าหมายในการ ออกแบบและพัฒนาโปรแกรมในระบบอินเทอร์เน็ต สำหรับผู้เขียนด้วยภาษา HTML สามารถทำงานข้ามแพลตฟอร์มได้ โดยทำงานร่วมกับ ภาษา HTML และภาษา Java ได้ทั้งทางฝั่งไคลเอนต์ (Client) และทางฝั่งเซิฟเวอร์ (Server)

JavaScript ถูกพัฒนาขึ้นโดยเน็ตสเคปคอมมิวนิเคชันส์(Netscape Communications Corporation) โดยใช้ชื่อว่า Live Script ออกมาพร้อมกับ Netscape Navigator2.0 เพื่อใช้สร้างเว็บเพจโดยติดต่อกับเซิฟเวอร์แบบ Live Wire ต่อมาเน็ตสเคปจึงได้ร่วมมือกับ บริษัทซันไมโครซิสเต็มส์ ปรับปรุงระบบของบราวเซอร์เพื่อให้สามารถติดต่อใช้งานกับภาษาจาวาได้ และได้ปรับปรุง LiveScript ใหม่เมื่อ ปี 2538 แล้วตั้งชื่อใหม่ว่า JavaScript JavaScript สามารถทำให้ การสร้างเว็บเพจ มีลูกเล่นต่าง ๆ มากมาย และยังสามารถโต้ตอบกับผู้ใช้ได้อย่างทันที เช่น การใช้เมาส์คลิก หรือ การกรอกข้อความในฟอร์ม เป็นต้น

เนื่องจาก JavaScript ช่วยให้ผู้พัฒนา สามารถสร้างเว็บเพจได้ตรงกับความต้องการ และมีความน่าสนใจมากขึ้น ประกอบกับเป็นภาษาเปิด ที่ใครก็สามารถนำไปใช้ได้ ดังนั้นจึงได้รับความนิยมเป็นอย่างสูง มีการใช้งานอย่างกว้างขวาง รวมทั้งได้ถูกกำหนดให้เป็นมาตรฐานโดย ECMA การทำงานของ JavaScript จะต้องมีการแปลความคำสั่ง ซึ่งขั้นตอนนี้จะถูกจัดการโดยบราวเซอร์ (เรียกว่าเป็น client-side script) ดังนั้น JavaScript จึงสามารถทำงานได้ เฉพาะบนบราวเซอร์ที่สนับสนุน ซึ่งปัจจุบันบราวเซอร์เกือบทั้งหมดก็สนับสนุน JavaScript แล้ว อย่างไรก็ดี สิ่งที่ต้องระวังคือ JavaScript มีการพัฒนาเป็นเวอร์ชั่นใหม่ๆออกมาด้วย (ปัจจุบันคือรุ่น 1.5) ดังนั้น ถ้านำโค้ดของเวอร์ชั่นใหม่ ไปรันบนบราวเซอร์รุ่นเก่าที่ยังไม่สนับสนุน ก็อาจจะทำให้เกิด error ได้

2.8.1 ประโยชน์ของ JavaScript

1) JavaScript ทำให้สามารถใช้เขียนโปรแกรมแบบง่ายๆได้โดยไม่ต้องพึ่งภาษาอื่น

2) JavaScript มีคำสั่งที่ตอบสนองกับผู้ใช้งาน เช่นเมื่อผู้ใช้คลิกที่ปุ่ม หรือ Checkbox ก็สามารถสั่งให้เปิดหน้าใหม่ได้ ทำให้เว็บไซต์ของเรามีปฏิสัมพันธ์กับผู้ใช้งานมากขึ้น นี่คือข้อดีของ JavaScript เลยก็ว่าได้ที่ทำให้เว็บไซต์ดังๆทั้งหลายเช่น Google Map ต่างหันมาใช้

3) JavaScript สามารถเขียนหรือเปลี่ยนแปลง HTML Element ได้ นั่น คือสามารถเปลี่ยนแปลงรูปแบบการแสดงผลของเว็บไซต์ได้หรือหน้าแสดงเนื้อหาสามารถซ่อนหรือ แสดงเนื้อหาได้แบบง่ายๆนั่นเอง

4) JavaScript สามารถใช้ตรวจสอบข้อมูลได้ สังเกตว่าเมื่อเรากรอก ข้อมูลบางเว็บไซต์ เช่น Email เมื่อเรากรอกข้อมูลผิดจะมีหน้าต่างฟ้องขึ้นมาว่าเรากรอกผิด หรือลืม กรอกอะไรบางอย่าง เป็นต้น

5) JavaScript สามารถใช้ในการตรวจสอบผู้ใช้ได้ เช่น ตรวจสอบว่าผู้ใช้ ใช้ web browser อะไร

6) JavaScript สร้าง Cookies ได้ (เก็บข้อมูลของผู้ใช้ในคอมพิวเตอร์ของผู้ใช้เอง)

2.8.2 ข้อดีและข้อเสียของจาวาสคริปต์ (JavaScript)

การทำงานของ JavaScript เกิดขึ้นบนบราวเซอร์ (เรียกว่าเป็น client-side script) ดังนั้นไม่ว่าจะใช้เซิฟเวอร์อะไร หรือที่ไหน ก็ยังคงสามารถใช้ JavaScript ในเว็บเพจได้ ต่างกับ ภาษาสคริปต์อื่น เช่น Pert, PHP หรือ ASP ซึ่งต้องแปลความและทำงานที่ตัวเครื่องเซิฟเวอร์ (เรียกว่า server-side script) ดังนั้นจึงต้องใช้บนเซิรฟเวอร์ ที่สนับสนุนภาษาเหล่านี้เท่านั้น อย่างไรก็ดี จากลักษณะดังกล่าวก็ทำให้ JavaScript มีข้อจำกัด คือไม่สามารถรับและส่งข้อมูลต่างๆกับเซิฟเวอร์โดยตรง เช่น การอ่านไฟล์จากเซิฟเวอร์ เพื่อนำมาแสดงบนเว็บเพจ หรือรับข้อมูล จากผู้ชม เพื่อนำไปเก็บบนเซิฟเวอร์ เป็นต้น ดังนั้นงานลักษณะนี้ จึงยังคงต้องอาศัยภาษา server-side script อยู่ (ความจริง JavaScript ที่ทำงานบนเซิร์ฟเวอร์ก็มี ซึ่งต้องอาศัยเซิร์ฟเวอร์ที่ สนับสนุนโดยเฉพาะเช่นกัน แต่ไม่เป็นที่นิยมนัก)

2.9 Variance (σ^2)

คือ ค่าความแปรปรวนของข้อมูล ซึ่งค่าที่ได้เมื่อนำมาถอดแสควรูทแล้วก็คือค่าเบี่ยงเบนมาตรฐาน หรือ Standard Deviation (σ) นั่นเอง การใช้ค่าความแปรปรวนของข้อมูลส่วนใหญ่จะนำไปใช้ในการวัดค่าการกระจายของข้อมูล ยกตัวอย่างเช่น ผมต้องการรู้ว่า ระหว่างลูกค้าเก่า ลูกค้าใหม่ โปรโมชันที่ออกมร มีผลต่อการซื้อสินค้า จากตัวอย่างข้อมูลจากทั้งสองกลุ่มหรือไม่ ยกตัวอย่าง

มูลค่าการซื้อสินค้าของลูกค้าเก่า 100, 200, 300, 400, 800 = 1,800 บาท

มูลค่าการซื้อสินค้าของลูกค้าใหม่ 400, 550, 350, 300, 260 = 1,860 บาท

หากใช้ค่าเฉลี่ยเลขคณิตจะพบว่าช่วงเวลาในการส่งเมล์นั้นไม่มีความแตกต่างกันเลย ค่าเฉลี่ยที่หาได้คือ 1,800/5 = 360 บาท สำหรับลูกค้าเก่า และ 1,860/5 = 372 บาท ซึ่งอาจจะดูได้

ว่า โปรโมชันที่ออกมานั้นไม่ค่อยมีผลกับการซื้อสินค้าของลูกค้าเก่าหรือลูกค้าใหม่เท่าไหร่ แต่หากเรามองดูที่ค่าของความเบี่ยงเบนแล้วจะแตกต่างกันมาก

การหา Variance นั้นสามารถหาได้ตามขั้นตอนต่อไปนี้ (เราจะเริ่มจากกลุ่มลูกค้าเก่า)

$$\sigma^2 = \frac{\sum(X-\mu)^2}{N}$$

- หาค่าเฉลี่ยเลขคณิตก่อน (Mean) = (100+200+300+400+800)/5 = 360

-จากนั้นคำนวณหาค่าของตัวเลขแต่ละตัวจากค่า Mean จากข้อมูลของลูกค้าเก่า เมื่อนำเลขแต่ละตัวไปลบกับค่า Mean แล้วก็จะได้ 100–360, 200–360, 300–360, 400–360, 800–360 ได้เท่ากับ (-260, -160, -60, 40, 440)

-นำค่าที่ได้มายกกำลังสองเพื่อเปลี่ยนให้ค่าลบกลายเป็นค่าบวกจะได้ (67,600 ; 25,600 ; 3,600 ; 1,600 ; 193,600)

- หาค่าเฉลี่ยของตัวเลขที่ได้ นั่นก็คือค่า Variance นั่นเอง 58,400

ใช้วิธีเดียวกันในการหาค่า Variance ของกลุ่มลูกค้าใหม่จะได้เท่ากับ 10,136 นั่นหมายความว่า ค่าความแปรปรวน Variance ของกลุ่มลูกค้าเก่า (58,400) มีการกระจายของข้อมูลมากกว่าของกลุ่มลูกค้าใหม่ (10,136) แสดงให้เห็นว่า โปรโมชันที่ออกมานั้นมีผลต่อการตัดสินใจซื้อสินค้าของกลุ่มลูกค้าเก่ามากกว่า สังเกตุว่าตัวเลขนั้นมีค่ามากเกินไป เราจึงนิยมใช้ ค่าเบี่ยงเบนมาตรฐาน (Standard deviation, SD, σ) ซึ่งหาได้จากการนำ ค่าของ Varience มาถอดสแควรูท ได้เป็น 241.66 และ 100.68 ตามลำดับ หากการนำข้อมูลจากกลุ่มตัวอย่างแค่บางส่วน (Sample data) ในการหาค่า Mean ของ Variance นั้นจะใช้ N-1 แตกต่างจากหากคำนวนจากข้อมูลทั้งหมด (Population) ที่ใช้ N คือจำนวนทั้งหมดในการหาค่า Mean นั่นเองครับ และสัญลักษณ์ของข้อมูลจะเปลี่ยนเป็น s2 สำหรับ Sample Variance และ s สำหรับ Sample Standard Deviation (SSD) ตามลำดับ

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - x_{avg})^2}{n-1}$$

2.10 ฮิสโตแกรม (Histogram)

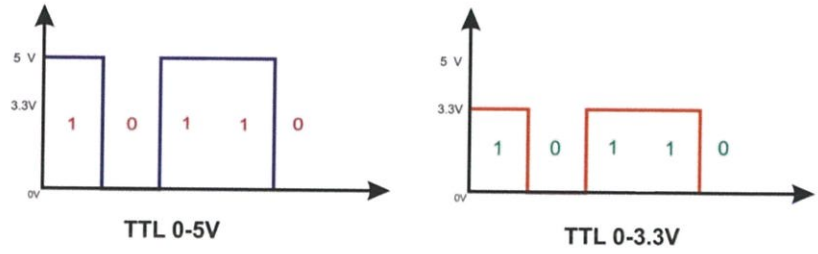
ฮิสโตแกรม (Histogram) คือกราฟแท่งแบบเฉพาะที่แสดงความสัมพันธ์ระหว่างข้อมูลเป็นหมวดหมู่ที่เรียกว่าชั้นข้อมูลกับความถี่ของข้อมูล เพื่อดูการกระจายของข้อมูล ลักษณะของข้อมูลที่เป็นหมวดหมู่จะเรียงลำดับจากน้อยไปหามากโดยจำนวนหมวดหมู่ของข้อมูลจะจัดตามความเหมาะสม โดยแกนตั้งจะเป็นตัวเลขแสดง “ความถี่” และแกนนอนจะเป็นข้อมูลคุณสมบัติของสิ่งที่เราสนใจ แท่งกราฟแต่ละแท่งจะมีความกว้างเท่ากันซึ่งเท่ากับกว้างของชั้นข้อมูล ส่วนความสูงของกราฟแต่ละแท่งนั้นจะสูงเท่ากับจำนวนความถี่ของแต่ละชั้นข้อมูล

ประโยชน์สำคัญของการใช้ฮิสโตแกรม (Histogram) คือการใช้เพื่อวิเคราะห์ความถี่ของข้อมูลแล้วตัดสินใจว่าการแจกแจงหรือการกระจายข้อมูลแบบใดมีผลต่อผลิตภัณฑ์ไปในทิศทางที่ดีหรือไม่และยังสามารถใช้ในการเปรียบเทียบข้อมูลจากการผลิตก่อนและหลังการปรับปรุงและนำมาใช้วิเคราะห์หาความสามารถของกระบวนการผลิต (Process capability) ได้อีกด้วย ในอุตสาหกรรมเรามักจะมีการใช้เครื่องมือ ฮิสโตแกรม (Histogram) ในการวิเคราะห์ข้อมูลของผลิตภัณฑ์อยู่เป็นประจำเพื่อวิเคราะห์ความสามารถของกระบวนการว่าเป็นไปตามแผนที่วางไว้หรือไม่ การวิเคราะห์ ฮิสโตแกรม (Histogram) ที่ลึกซึ้งจะช่วยให้เราเข้าใจธรรมชาติของผลิตภัณฑ์และกระบวนการ นอกจากนี้ยังจะช่วยให้เราหาแนวทางการวิเคราะห์ข้อมูลและแนวทางการปรับปรุงงานที่เหมาะสมในขั้นตอนต่อไปได้อย่างมีประสิทธิภาพมากขึ้น

ฮิสโตแกรม (Histogram) เป็นแผนภูมิแท่งที่บอกถึงความถี่ที่เกิดขึ้นในแต่ละอันตรภาคชั้น โดยแต่ละแท่งจะวางเรียงติดกัน แกนนอนจะกำกับด้วยค่าขอบบนและขอบล่างของชั้นนั้น หรือใช้ค่ากลาง (Midpoint) แกนตั้งเป็นค่าความถี่ในอันตรภาคชั้น ดังนั้นความสูงของแต่ละแท่งจะขึ้นอยู่กับความถี่นั่นเองฮิสโตแกรม (Histogram) ยังใช้ในการเปรียบเทียบกับขีดจำกัดข้อกำหนดเฉพาะของกระบวนการผลิตว่าเป็นอย่างไร หากลักษณะของฮิสโตแกรม (Histogram) เป็นรูประฆังคว่ำ สมมาตร คือมีการแจกแจงแบบปกติหรือใกล้เคียงกับแบบปกติมากที่สุด และฐานของฮิสโตแกรม (Histogram) อยู่ภายในขีดจำกัดข้อกำหนดเฉพาะด้านบน และขีดจำกัดข้อกำหนดเฉพาะด้านล่าง แสดงว่ากระบวนการผลิตดำเนินไปด้วยดี ไม่ต้องทำการแก้ไข แต่ถ้าในกรณีที่ฐานของฮิสโตแกรม (Histogram) หลุดออกจากขีดจำกัดข้อกำหนดเฉพาะด้านบนและขีดจำกัดข้อกำหนดเฉพาะด้านล่าง หรือในกรณีที่ฐานของฮิสโตแกรม (Histogram) กว้างเกินไปจะต้องปรับให้ค่าความผันแปรหรือส่วนเบี่ยงเบนมาตรฐานของข้อมูลการผลิตต่ำลง เพื่อให้ฐานของฮิสโตแกรม (Histogram) นั้นแคบลงอยู่ภายในขีดจำกัดข้อกำหนดเฉพาะด้านบนและด้านล่าง ข้อมูลที่ถูกเลือกมาจากกระบวนการผลิตในแต่ละวัน ไม่ว่าจะมาจากสายงานการผลิตหรือมาจากสินค้าที่บรรจุเป็นลอทย่อมจะมีความแตกต่างกันในค่าคุณสมบัติที่วัดได้ เช่น อาจจะเก็บข้อมูลมาด้วยการวัดค่าความยาวของผลิตภัณฑ์ การนับจำนวนวันที่ขาดงานของพนักงาน การวัดเส้นผ่าศูนย์กลางของผลิตภัณฑ์ หรือการชั่งน้ำหนักของผลิตภัณฑ์ เป็นต้น ค่าที่เราวัดได้เหล่านี้สามารถนำมาเสนอในลักษณะของรายงาน กราฟ หรือแผนภูมิควบคุม

2.11 TTL (Transistor-Transistor Logic)

TTL เป็นระดับแรงดันที่ถูกกำหนดขึ้นในยุคแรกๆเพื่อใช้ระหว่าง Transistor กับ Transistor ภายในวงจรรวม (IC) ดังนั้น TTL จะใช้ระดับแรงดัน อยู่ที่ 0 – 5 V แต่ในปัจจุบันมีอุปกรณ์หลายเบอร์ที่ทำงานในช่วง 0 – 3.3 V (เรียกแรงดันระดับนี้ว่า LVTTL) ซึ่งผู้ใช้ควรตรวจสอบจาก Datasheet ของอุปกรณ์ที่ใช้เสียก่อนว่าเป็นระดับแรงดันแบบใด เพราะหากใช้ผิดประเภทจะทำให้อุปกรณ์เสียหาย แสดงได้ดังรูปที่ 2.19 TTL



รูปที่ 2.19 TTL [8]

2.12 UART

ย่อมาจากคำว่า Universal Asynchronous Receiver Transmitter หมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัส (Asynchronous) ซึ่งเป็นส่วนหนึ่งในการสื่อสารอนุกรมแบบ Asynchronous แท้จริงแล้วการสื่อสารแบบอนุกรมจะแบ่งเป็น 2 แบบ คือ

2.10.1 การสื่อสารอนุกรมแบบ Synchronous

เป็นรูปแบบที่ใช้วิธีส่งข้อมูล โดยใช้สัญญาณ Clock มาเป็นตัวกำหนดจังหวะ การรับส่งข้อมูล การส่งข้อมูลแบบนี้ เป็นการรับส่งที่ค่อนค่างมีคุณภาพ และส่งได้ที่ความเร็วสูง มีโอกาสที่ข้อมูลจะสูญหายระหว่างการส่งน้อย ตัวอย่างการส่งข้อมูลลักษณะนี้เช่น I2C, I2S, SPI ข้อเสียของการส่งข้อมูลแบบนี้คือต้องใช้สายสัญญาณมาก เพราะว่าต้องส่ง Clock ไปด้วย แสดงได้ดังรูปที่ 2.20

รูปที่ 2.20 การสื่อสารอนุกรมแบบ Synchronous [8]

2.20.2 การสื่อสารอนุกรมแบบ Asynchronous

เป็นการส่งข้อมูลที่ไม่ต้องใช้สัญญาณ Clock มาเป็นตัวกำหนดจังหวะการรับส่งข้อมูลแต่ใช้วิธีกำหนด รูปแบบ Format การรับส่งข้อมูลขึ้นมาแทน และ อาศัยการกำหนด ความเร็วของการรับและ ส่ง ที่เท่ากันทั้งฝั่งรับและฝั่งส่ง ข้อดีของการใช้ Asynchronous คือสามารถสื่อสารแบบ Full Duplex รับ และ ส่งได้ในเวลาเดียวกัน แต่ Asynchronous มีโอกาสที่ข้อมูลจะสูญหายขณะรับส่งข้อมูล หรือ รับส่งข้อมูลผิดพลาดได้มากกว่าแบบ Synchronous

สรุปกล่าวคือ UART (Universal Asynchronous Receiver Transmitter) หมายถึงรูปแบบการส่งข้อมูล ที่ถูกกำหนดขึ้นมาเพื่อใช้รับส่งข้อมูลแบบ Asynchronous โดยมีรูปแบบดังรูปที่ 2.21 การรับส่งข้อมูลแบบ Asynchronous

รูปที่ 2.21 การรับส่งข้อมูลแบบ Asynchronous [8]

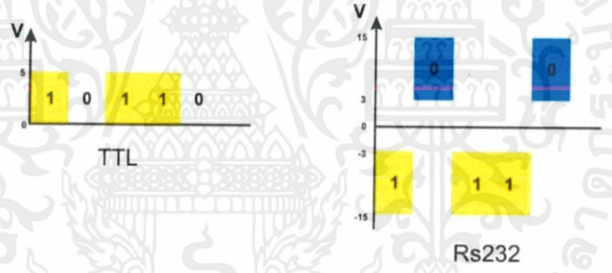
2.13 RS232 (Recommended Standard 232)

RS232 คือ มาตรฐานการเชื่อมต่อข้อมูลแบบ Serial ใช้เพื่อเพิ่มระยะทางในการส่งข้อมูล แบบ Serial ให้สามารถส่งได้ระยะทางที่มากขึ้น โดยมีการเปลี่ยนระดับแรงดัน ของ Logic จากเดิมที่จะอยู่ในช่วง 0-5 V หรือ 0-3.3 V เป็นช่วง -15 ถึง 15 V โดยมีรายละเอียดดังนี้

Logic 0 ของ RS232 จะอยู่ในช่วง 3 ถึง 15V

Logic 1 ของ RS232 จะอยู่ในช่วง -3 ถึง -15V

จากรูปที่ 2.22 จะเห็นได้อย่างชัดเจนครับ ว่าทั้ง 2 อย่าง ส่ง Data เหมือนกันครับ แต่ระดับแรงดันที่ใช้ต่างกันมาก หากอุปกรณ์เป็น TTL แล้ว ไปต่อกับ RS232 ก็จะเกิดความเสียหายได้



รูปที่ 2.22 เปรียบเทียบการส่ง Data ระหว่าง TTL กับ RS232 [8]

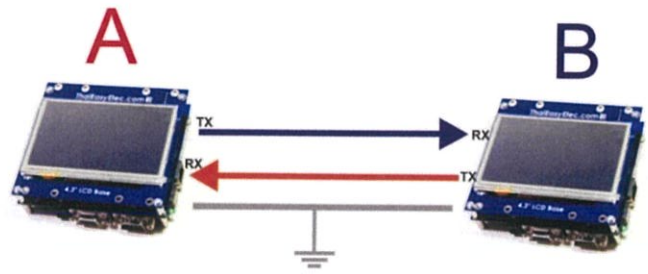
2.14 แนวคิดวิธีเชื่อมต่อระหว่างอุปกรณ์

จากรูปที่ 2.23 แสดงถึงการเชื่อมต่อ ระหว่างบอร์ด 2 ตัว เพื่อส่ง Data หากันระหว่างบอร์ดด้วย UART

1) A ส่งข้อมูล ออกไปทางขา Tx ไปยัง B ซึ่งเป็นฝั่งรับ เพราะฉะนั้น ต้องต่อสายสัญญาณจากขา Tx ของ A ไปยังขา Rx ของ B

2) B ส่งข้อมูล ออกไปทางขา Tx ไปยัง A ซึ่งเป็นฝั่งรับ เพราะฉะนั้น ต้องต่อสายสัญญาณจากขา Tx ของ B ไปยังขา Rx ของ A

3) ต้องต่อ GND ของทั้ง A และ B ร่วมกันเพื่อทำให้ระดับแรงดันของทั้ง 2 บอร์ดมีจุดอ้างอิงเดียวกัน



รูปที่ 2.23 แสดงการเชื่อมต่อระหว่างบอร์ด 2 ตัวด้วย UART [8]

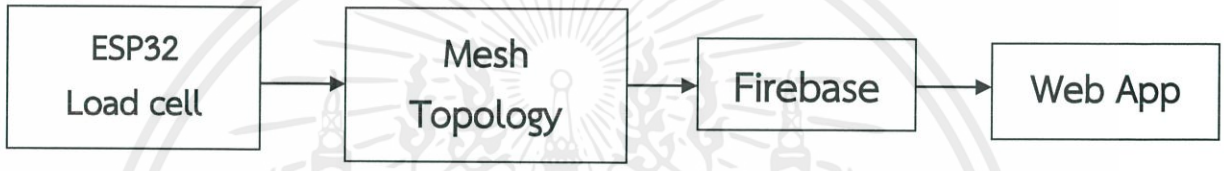
บทที่ 3

การออกแบบและการจัดทำปริญญานิพนธ์

3.1 การออกแบบ

3.1.1 หลักการทำงาน

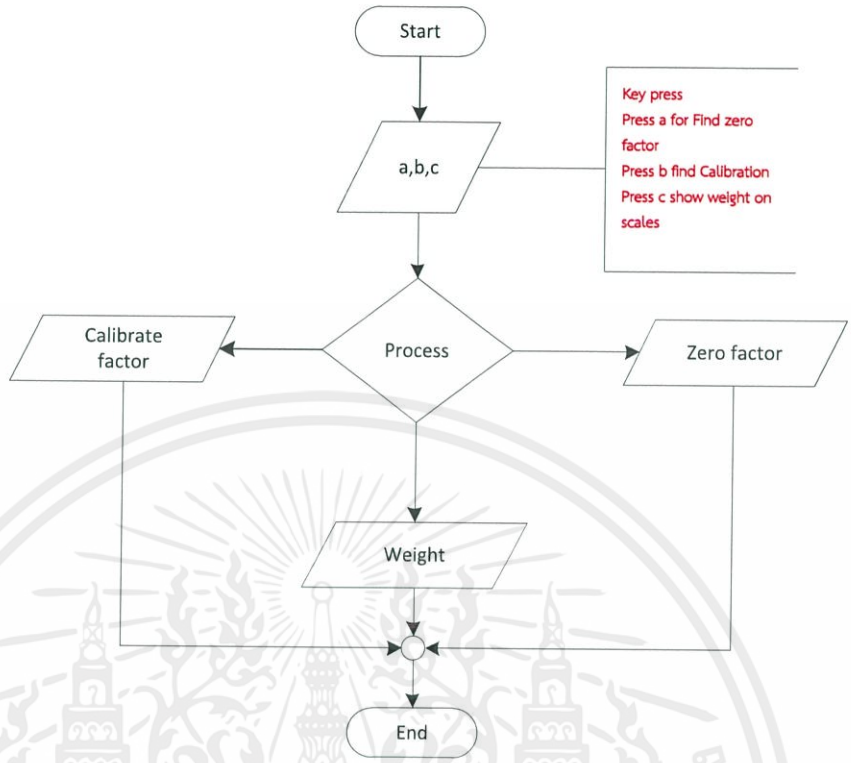
หลักการทำงานเริ่มต้นจากผู้ใช้(User) เริ่มต้นใช้งาน ESP32 โดยเชื่อมต่อกันผ่าน HX711 ซึ่งเป็นตัวแปลงค่าจาก analog เป็น digital เมื่อรันโค้ดเพื่อชั่งน้ำหนักน้ำหนักที่วัดได้จะส่งค่าไปยัง firebase server โดยทุกครั้งที่ส่งค่าไปจะบอกวันและเวลา จากนั้นจะเอาข้อมูลจาก firebase มาใช้เขียนแอฟ มีการ authentication ด้วย email เพื่อความปลอดภัยในการใช้ข้อมูล



รูปที่ 3.1 บล็อคไดอะแกรมของโครงงานที่นำเสนอ

3.1.2 การ Calibrate น้ำหนักก่อนชั่ง

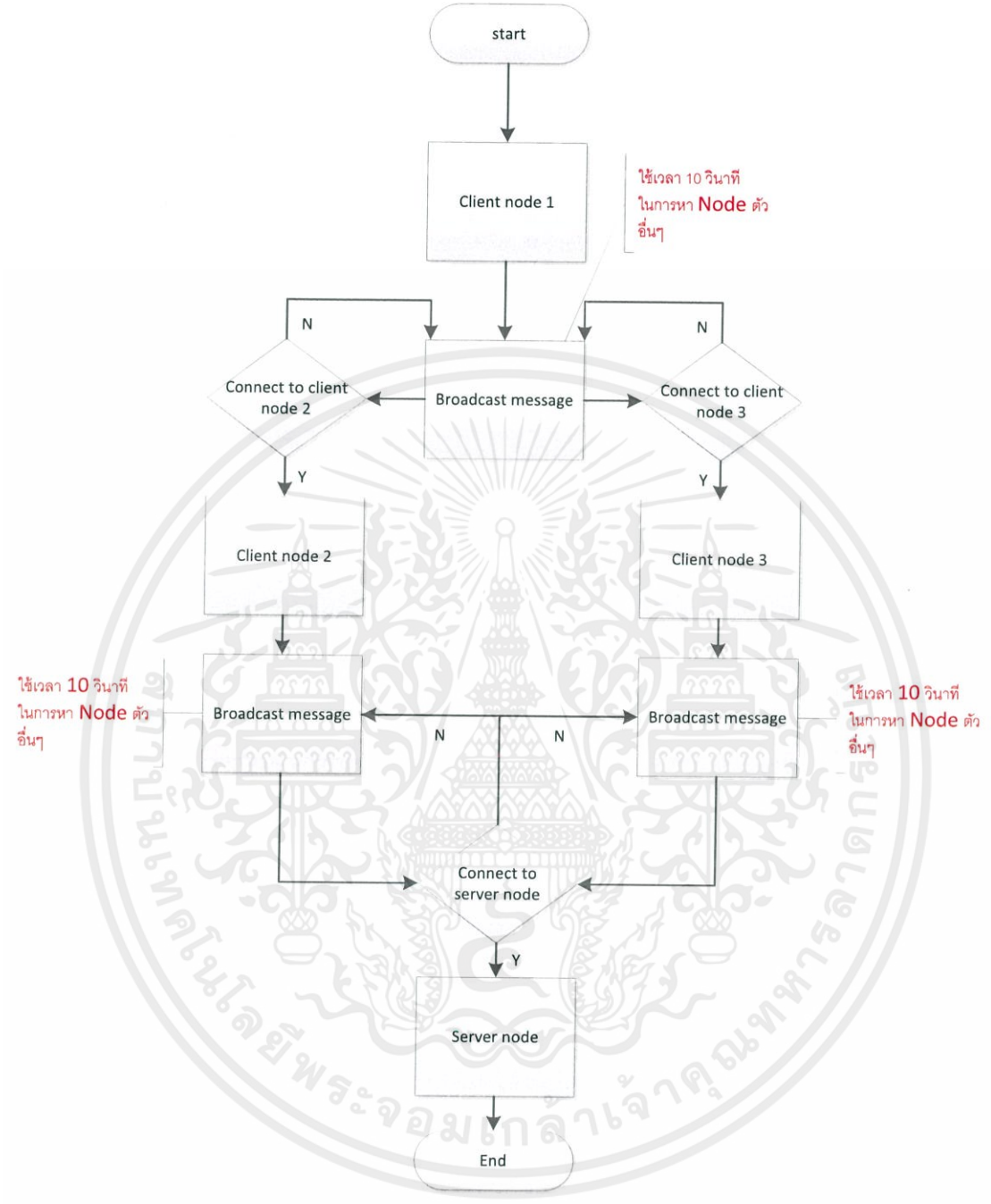
ส่วนของโปรแกรมการ calibrate น้ำหนักก่อนชั่งจะแบ่งการทำงานออกเป็นสามส่วนหลักคือ ส่วนหาค่า zero factor หาค่า calibrate factor และทดสอบน้ำหนักที่ชั่งได้ว่าค่าคลาดเคลื่อนหรือเท่ากับน้ำหนักจริงหรือไม่



รูปที่ 3.2 ผังงานรวมการ calibrate น้ำหนักก่อนชั่ง

3.1.3 การทำงานของ Client node

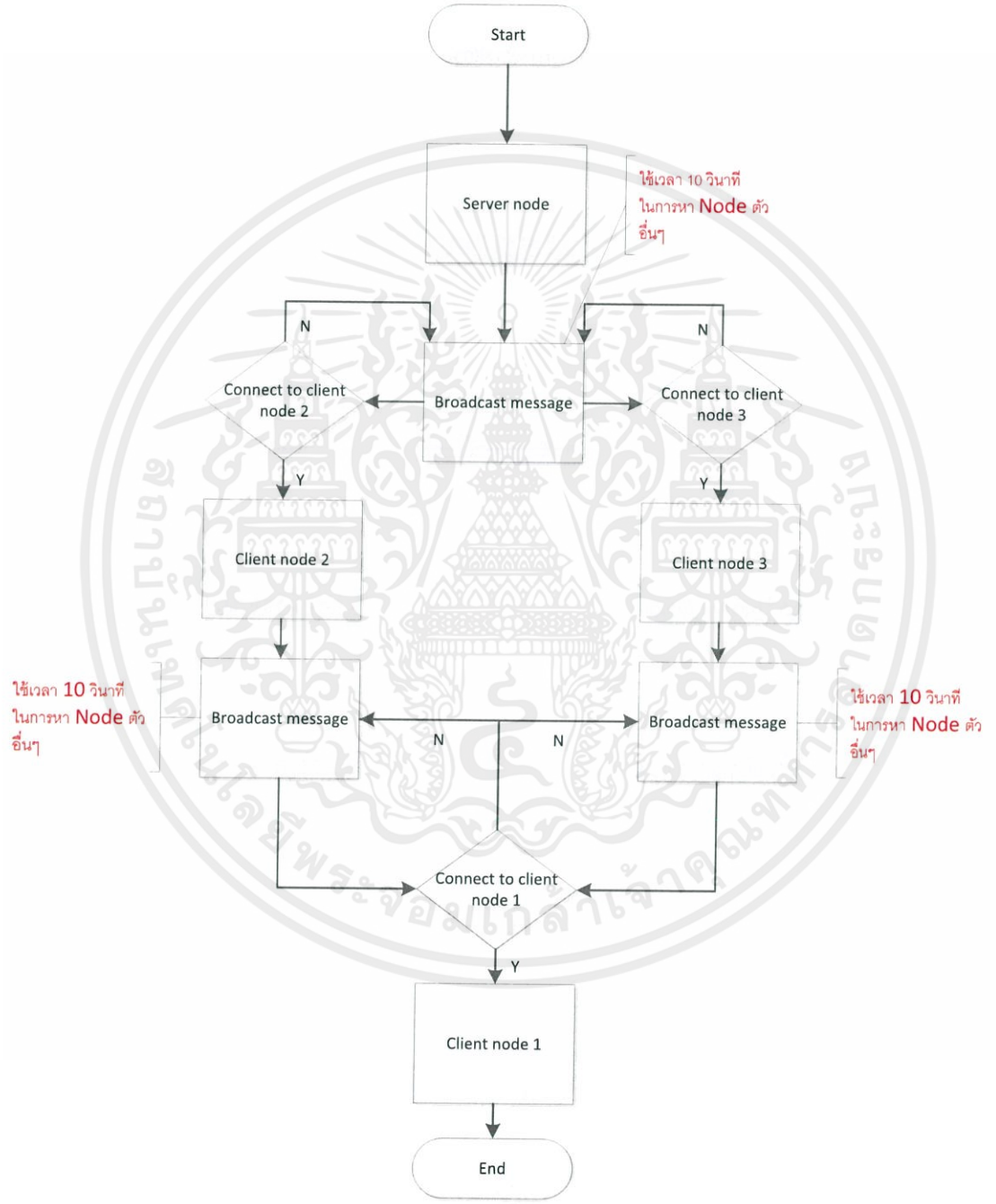
Client node จะทำการ Broadcast message ในเครือ Mesh network เดียวกัน เพื่อที่จะทำการเชื่อมต่อ Client node ตัวอื่นๆและ Server node ทั้งนี้ Client node แต่ละตัวสามารถที่จะแสดงค่าน้ำหนักที่ชั่งได้ ณ ขณะนั้น จากนั้นจะส่งข้อมูลทั้งหมดไปยัง Server node



รูปที่ 3.3 ผังงานการทำงานของ client node

3.1.4 การทำงานของ Server node

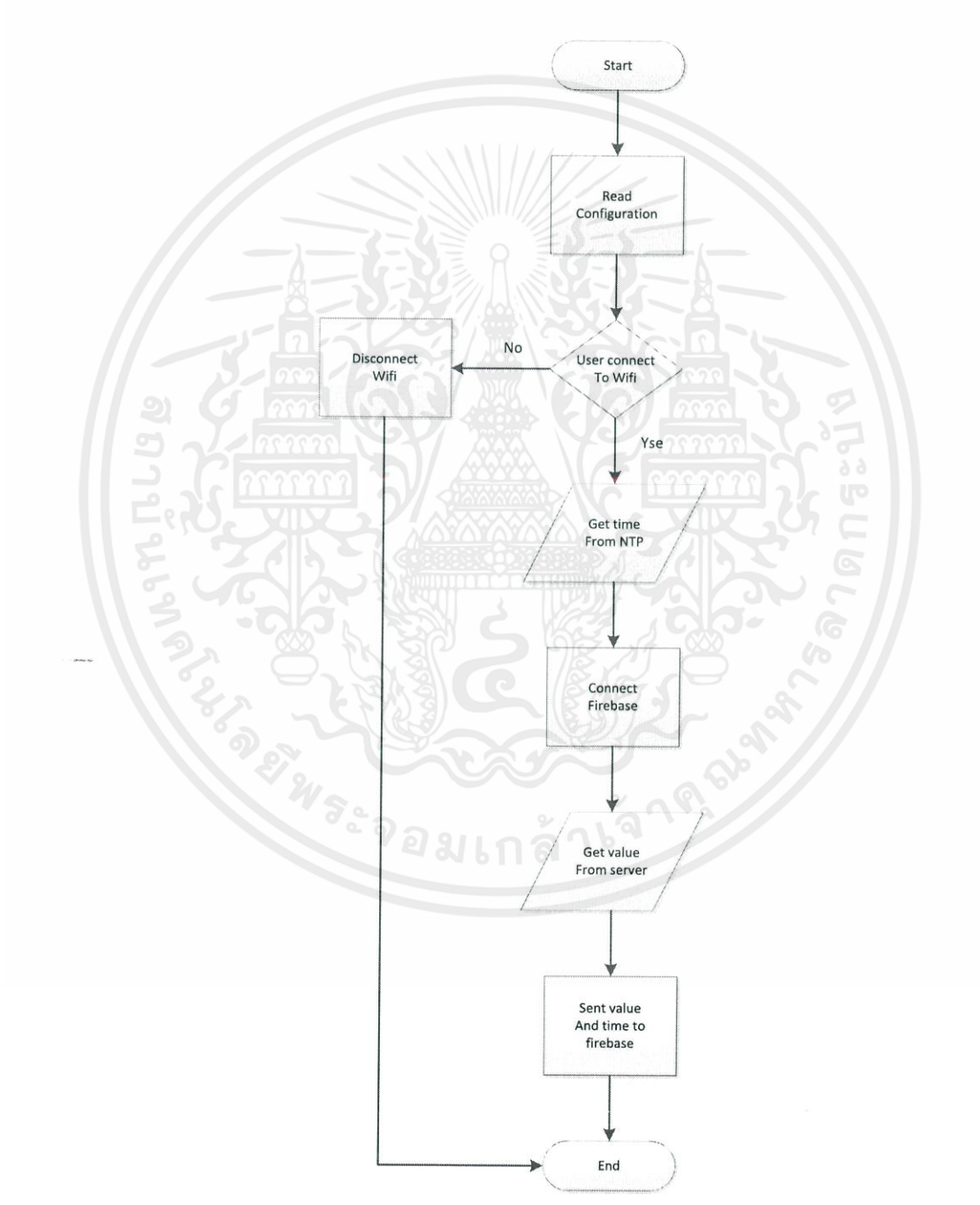
Server node จะทำการ Broadcast message ในเครือ Mesh network เดียวกัน เพื่อที่จะบ่งบอกว่าเป็น Server node เมื่อ Client node ตรวจพบเจอ Server node แล้ว Client node จะทำการส่งค่าน้ำหนักที่ชั่งได้ ณ ขณะนั้น ไปยัง Server node



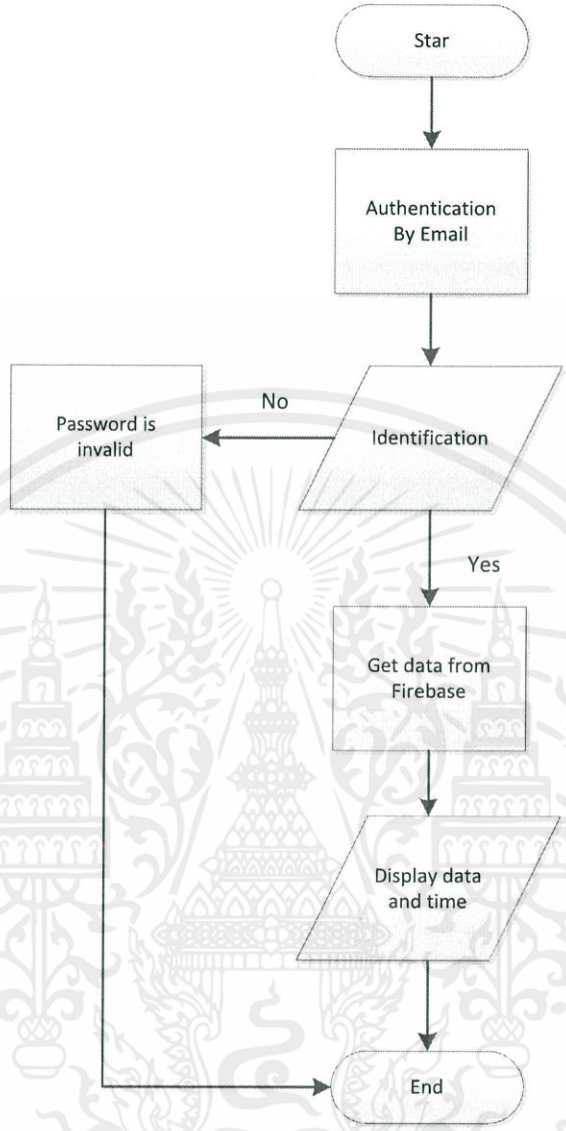
รูปที่ 3.4 ผังงานการทำงานของ server

3.1.4 การนำข้อมูลขึ้น Firebase โดยผ่าน ESP32 ที่เป็น Gateway Node

ส่วนของ Gateway Node จะมีการรับข้อมูลจาก Server Node ผ่านทาง Serial Port และ Gateway Node จะทำหน้า Bridge ระหว่าง Mesh Network กับ Network ภายนอกแล้วส่งข้อมูลต่อไปยัง Firebase Server และนำข้อมูลจาก Firebase Server แสดงโชว์ค่าน้ำหนักบน Web โดยการเข้าใช้งานเว็บต้องมีการยืนยันตัวตนผ่าน email ก่อนที่จะมีการเข้าใช้งานระบบ ข้อมูลที่แสดงค่าน้ำหนักจะโชว์ค่าน้ำหนักของสินค้า ณ เวลานั้นๆ โดยใช้ HTML sync กับ Firebase แบบ real time



รูปที่ 3.5 ผังงานส่วนของ Gateway Node ส่งข้อมูลขึ้น Firebase Server



รูปที่ 3.6 ผังงานส่วนของหน้าเว็บ

3.2 เครื่องมือที่ใช้ในการทดลอง

3.2.1 ESP32

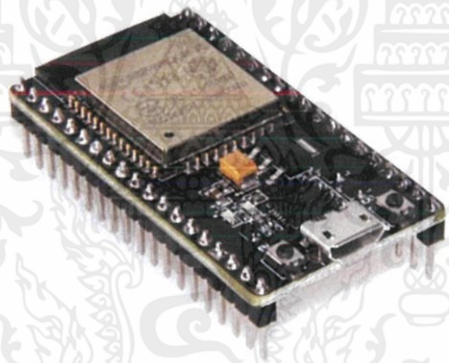
ESP32 เป็นชื่อของไอซีไมโครคอนโทรลเลอร์ที่รองรับการเชื่อมต่อ WiFi และ Bluetooth 4.2 BLE ในตัว ผลิตโดยบริษัท Espressif จากประเทศจีน โดยราคา ณ ที่เขียนบทความอยู่นี้ มีราคาไม่เกิน 500 บาท (บอร์ดพัฒนาสำเร็จรูป) โดยตัวไอซี ESP32 มีสเปคโดยละเอียด ดังนี้

- ซีพียูใช้สถาปัตยกรรม Tensilica LX6 แบบ 2 แกนสมอง สัญญาณนาฬิกา 240MHz
- มีแรมในตัว 512KB
- รองรับการเชื่อมต่อรอมภายนอกสูงสุด 16MB
- มาพร้อมกับ WiFi มาตรฐาน 802.11 b/g/n รองรับการใช้งานทั้งในโหมด Station softAP และ Wi-Fi direct

- มีบลูทูธในตัว รองรับการใช้งานในโหมด 2.0 และโหมด 4.0 BLE
- ช้แรงดันไฟฟ้าในการทำงาน 2.6V ถึง 3V
- ทำงานได้ที่อุณหภูมิ -40°C ถึง 125°C

ขาใช้งานต่าง ๆ ของ ESP32 รองรับการเชื่อมต่อบัสต่าง ๆ ดังนี้

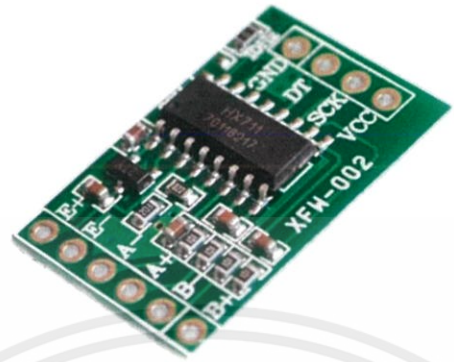
-มี GPIO จำนวน 32 ช่อง
-รองรับ UART จำนวน 3 ช่อง
-รองรับ SPI จำนวน 3 ช่อง
-รองรับ I2C จำนวน 2 ช่อง
-รองรับ ADC จำนวน 12 ช่อง
-รองรับ DAC จำนวน 2 ช่อง
-รองรับ I2S จำนวน 2 ช่อง
-รองรับ PWM / Timer ทุกช่อง
-รองรับการเชื่อมต่อกับ SD-Card



รูปที่ 3.7 ESP32 [1]

3.2.2 HX711

โมดูล HX711 เป็นโมดูลสำหรับขยายสัญญาณจาก Load Cell เซ็นเซอร์วัดน้ำหนัก ซึ่งปกติมาค่าน้อยมากๆ ตัวโมดูลนี้จะขยายสัญญาณ ออกเป็นสัญญาณดิจิตอล 24bit I2C ทำให้สามารถนำ Arduino NodeMCU Raspberry Pi หรือ MCU อื่นๆมาอ่านค่าน้ำหนักได้ง่ายๆ สายเชื่อมต่อเข้า MCU มี 2 เส้นและไฟเลี้ยง 2 เส้น สามารถใช้ไฟเลี้ยงได้ตั้งแต่ 2.6-5.5v และอีกด้านของโมดูลสามารถต่อกับ LoadCell ได้เลย



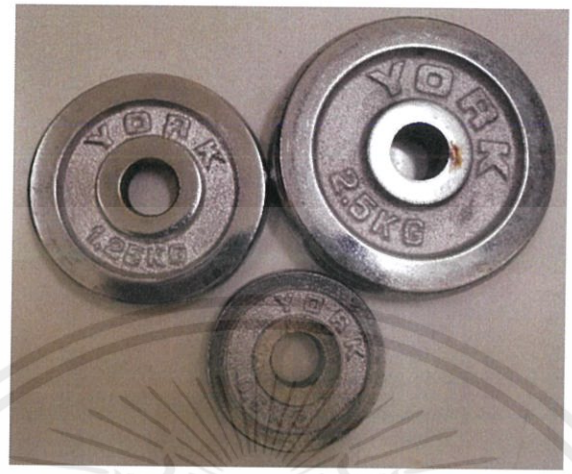
รูปที่ 3.8 HX711 [6]

3.2.3 เครื่องชั่งน้ำหนัก

Sensor สำหรับตรวจวัดน้ำหนัก แรงกระทำทางกล หรือปริมาณของ Load ที่ต้องการทราบค่า โดยใช้ Strain Gauge มาติดตั้งในบริเวณที่มีการเปลี่ยนแปลงรูปทรงของ Load Cell เมื่อมีแรงมากระทำกับตัว Load Cell จะทำให้ Strain Gauge ที่ติดอยู่ในบริเวณที่มีการเปลี่ยนรูปทรง ยืด หรือ หด ตัว ทำให้ค่าความต้านทานที่ตัว Strain Gauge เปลี่ยนไป

รูปที่ 3.9 เครื่องชั่งน้ำหนัก

3.2.4 แผ่นเหล็กน้ำหนัก



รูปที่ 3.10 แผ่นเหล็กน้ำหนัก 0.5kg,1.25kg,2.5kg

3.3 การจัดเก็บผลการทดลอง

ในการทดลองประสิทธิภาพการทำงานของเครื่องชั่งน้ำหนักสามารถตรวจวัดประสิทธิภาพได้ด้วยการเก็บผลการทดลองดังนี้

3.3.1 การ Calibrate Load Cell ก่อนชั่ง

การ Calibrate นั้นต้องใช้ Load หรือน้ำหนักที่ทราบค่าอยู่แล้วโดยการทดลองนี้จะใช้แผ่นเหล็กดัมเบลขนาด 1.25 กิโลกรัม เพื่อหาค่า Zero Factor หรือ ตอนชั่งโดยไม่มีแผ่นเหล็ก และ Calibrate Factor หรือ ตอนชั่งแบบมีแผ่นเหล็ก

3.3.2 การทดลองการวัดน้ำหนัก Client-Server

การวัดค่าน้ำหนักโดยใช้ Zero Factor และ Calibrate Factor จากการ Calibrate Load Cell มาใช้ทดสอบประสิทธิภาพการวัดน้ำหนักได้โดยทดลองชั่งน้ำหนักทั้งหมด 100 ครั้ง และพิจารณาความผิดพลาดว่าความผิดพลาดมีแนวโน้ม หรือความสัมพันธ์ระหว่าง load cell และพื้นผิดที่ชั่งน้ำหนักอย่างไร ซึ่งการทดลองแต่ละครั้งเกิดค่าความคลาดเคลื่อนประมาณ 0.2 กิโลกรัม ในที่นี้ให้ ESP32 ที่ทำหน้าที่ในการเป็นตัวชั่งน้ำหนักให้เป็น Client node ทำการส่งค่าน้ำหนัก ณ ขณะนั้นไปยัง ESP32 ที่ทำหน้าที่เป็น Server node จากนั้น Server node จะส่งค่าน้ำหนักทั้งหมดที่มาจาก Client node หลายๆตัว ไปยัง ESP32 ที่ทำหน้าที่เป็น Gateway node

3.3.3 การนำข้อมูลขึ้น Firebase โดยผ่าน ESP32 ที่เป็น Gateway Node

ลักษณะการทำงานของ Mesh Network นั้นมักจะเป็นการใช้งานแบบ local network จะมีอีก Node หนึ่งขึ้นมาเพื่อเป็นสะพาน (Bridge) และใช้เป็นประตู (Gateway) ในการ forward ข้อมูลหรือใช้ในการสื่อสารกับ Network ภายนอก เพื่อนำข้อมูลน้ำหนักและหมายเลขของ Node Client พร้อมทั้งวันที่และเวลาขึ้น Firebase Server เช่น "mcu-t1":"2.60" , "mcu-t2":"1.33" ,

"mcu-t3" : "0.00" เพื่อไปใช้กับ platform ของ Web โดยต้องลง Firebase-tools และ login เลือก function ที่เราต้องการใช้ของ firebase หลังจากนั้น copy script และ config ในรูปแบบของ json object หลังจากนั้นเชื่อมต่อ database กับหน้าเว็บแบบ realtime

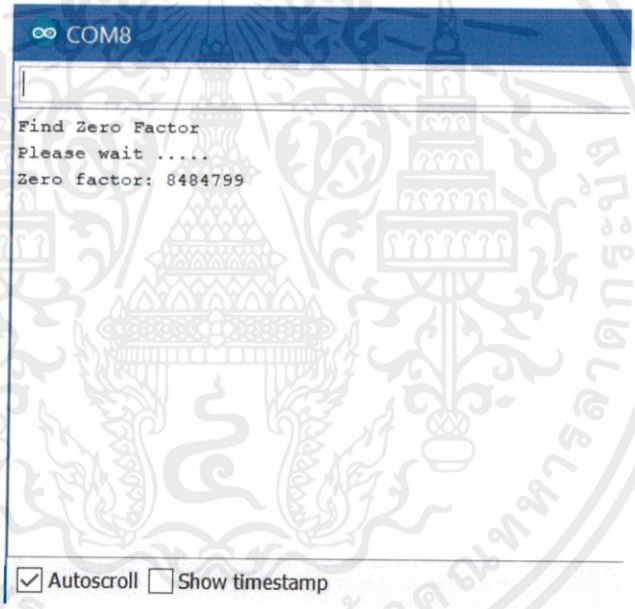
บทที่ 4

ผลการทดลอง

4.1 ผลการทดสอบการ Calibrate Load Cell

4.1.1 ผลการทดลองส่วนของการเก็บค่า Zero Facrtor

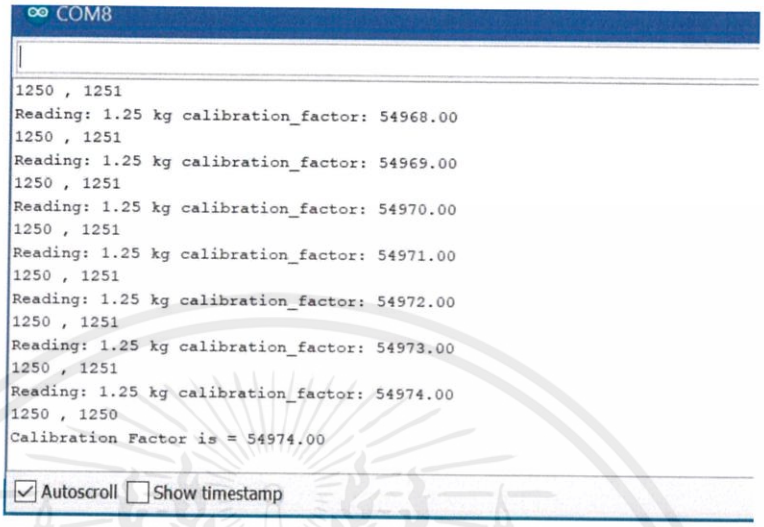
ในการทดสอบการ Calibrate Load Cell จะทำการทดสอบโดยการนำ Load หรือน้ำหนักที่ทราบค่าอยู่แล้ว โดยในการทดสอบนี้จะใช้ เหล็กแผ่น ดัมเบลขนาด 1.25 กิโลกรัม เพื่อเปรียบเทียบกันระหว่างน้ำหนักของทั้งสองขนาด โดยในขั้นตอนแรก ติดตั้ง Load Cell เข้ากับแผ่นชั่งน้ำหนัก เชื่อมต่อ Load Cell และ HX711 เข้ากับบอร์ด Arduino โดยขั้นตอนแรก โดยส่งตัว ‘a’ ไปเพื่อหาค่า Zero Factor หรือค่าตอนที่ไม่มีน้ำหนักอยู่เลย จดบันทึกไว้ โดยผลลัพธ์เป็นไปดังรูปที่ 4.1



รูปที่ 4.1 ผลการทดลองการเก็บค่า Zero Factor ของน้ำหนัก 1.25 กิโลกรัม

4.1.2 ผลการทดลองส่วนของการเก็บค่า Calibrate_Factor

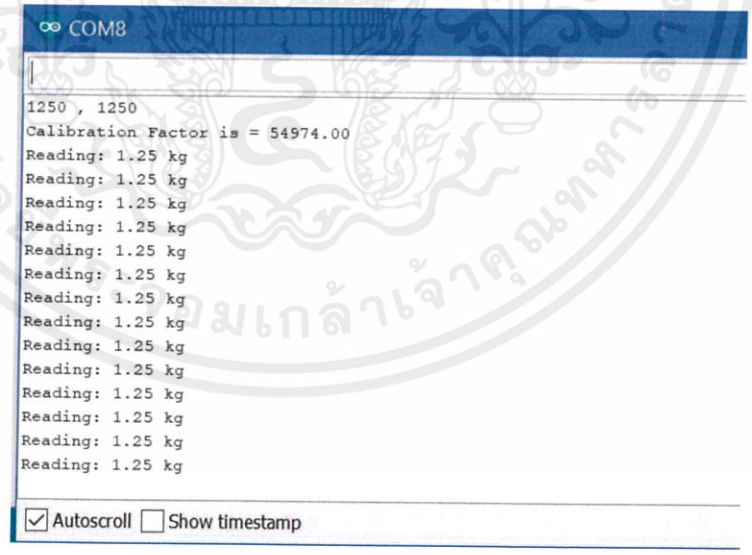
ผลการทดลองการหาค่า Calibrate_Factor โดยโดยส่งตัว ‘b’ และนำเหล็กแผ่นดัมเบลมาชั่งน้ำหนักทั้ง 1.25 และ 2.50 กิโลกรัมโดยผลลัพธ์เป็นไปดังรูปที่ 4.2



รูปที่ 4.2 ผลการทดลองการเก็บค่า Calibrate Factor ของน้ำหนัก 1.25 กิโลกรัม

4.1.3 ผลการทดลองการวัดน้ำหนัก

ผลการทดลองการวัดน้ำหนักก่อนวัดน้ำหนักจริง โดยโดยส่งตัว 'c' โดยนำแผ่นดัมเบลมาชั่งน้ำหนักทั้ง 1.25 กิโลกรัมชั่งไว้เหมือนเดิม โดยผลลัพธ์เป็นไปดังรูปที่ 4.3

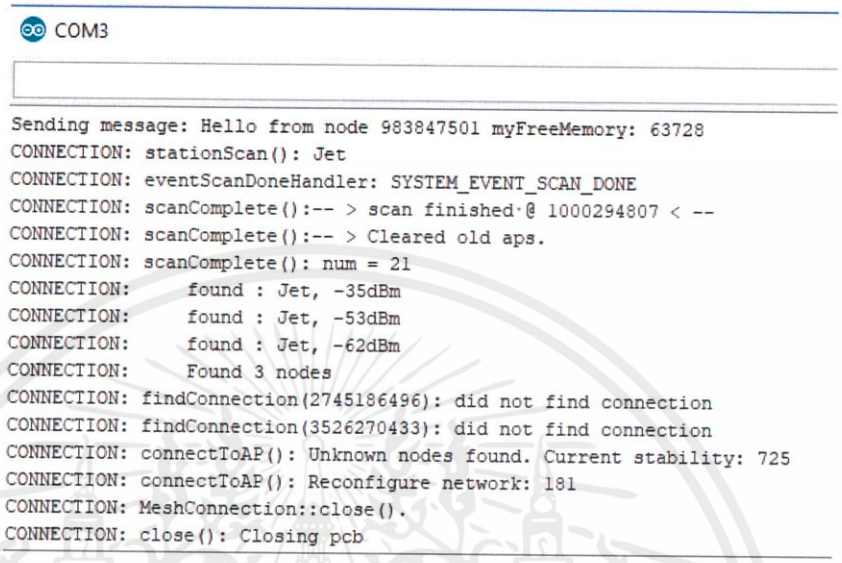


รูปที่ 4.3 ผลการทดลองวัดค่าของน้ำหนัก 1.25 กิโลกรัม

4.2 ผลการทำ Mesh network ของ ESP32

ผู้จัดทำได้วางแผนที่จะนำ ESP32 หลายๆตัวมาทำเป็น Mesh Network ในตอนแรกได้ทำการ upload โปรแกรม Painless Mesh ลงไปใน ESP32 ตัวแรกและเชื่อมต่อ WIFI จะเห็นได้ว่า

ESP32 ตัวแรกได้ทำการปล่อย WIFI ชื่อว่า Jet พร้อมกับบอกหมายเลข AP คือ 983847501 และ memory ที่เหลืออยู่ แสดงดังรูปที่ 4.4 และ 4.5



รูปที่ 4.4 Serial monitor ของ ESP32 ตัวแรก

รูปที่ 4.5 ESP32 หลังจาก upload โปรแกรมสำหรับ Mesh Network

ทำการ upload โปรแกรม Painless Mesh ลงไปใน ESP32 อีก 2 ตัว พบว่า ESP32 ตัวแรกได้ตรวจพบ ESP32 อีก 2 ตัวซึ่งมี AP คือ 983847501 และ 2382783801 แสดงได้ดังรูปที่ 4.6 และรูปที่ 4.7

```
CONNECTION:     Found 2 nodes
CONNECTION: connectToAP(): No unknown nodes found scan rate set to normal
Sending message: Hello from node 2385987433 myFreeMemory: 210564
startHere: Received from 983847501 msg=Hello from node 983847501 myFreeMemory: 208284
startHere: Received from 2382783801 msg=Hello from node 2382783801 myFreeMemory: 212860
startHere: Received from 983847501 msg=Hello from node 983847501 myFreeMemory: 208200
Sending message: Hello from node 2385987433 myFreeMemory: 212344
startHere: Received from 2382783801 msg=Hello from node 2382783801 myFreeMemory: 212860
startHere: Received from 983847501 msg=Hello from node 983847501 myFreeMemory: 206528
startHere: Received from 983847501 msg=Hello from node 983847501 myFreeMemory: 208032
Sending message: Hello from node 2385987433 myFreeMemory: 212344
startHere: Received from 2382783801 msg=Hello from node 2382783801 myFreeMemory: 212860
```

รูปที่ 4.6 แสดงถึงการสื่อสารของ ESP32 ตัวอื่นๆ

รูปที่ 4.7 การสื่อสารของ ESP32 ทั้ง 3 ตัวหลังจาก upload โปรแกรมสำหรับ Mesh Network

4.3 ผลจากการทำ Client/Server ของ ESP32

ผู้จัดทำได้ทำการให้ ESP32 แต่ละตัวทำงานในลักษณะที่เป็น client และ server โดยการ upload โปรแกรมให้ ESP32 ที่มี ID คือ 2385987433 เป็นตัว server node และ ESP32 อีก 2 ตัวเป็น client node เช่น logServer Receive from 2382783801 คือ Client node มี ID คือ 2382783801 ดังรูปที่ 4.8

```
logServer: Received from 2382783801 msg={"topic":"sensor","value":12}
logServer: Received from 983847501 msg={"topic":"sensor","value":47}
{"topic":"logServer","nodeId":2385987433}
logServer: Received from 2382783801 msg={"topic":"sensor","value":149}
logServer: Received from 983847501 msg={"topic":"sensor","value":43}
{"topic":"logServer","nodeId":2385987433}
logServer: Received from 2382783801 msg={"topic":"sensor","value":65}
logServer: Received from 983847501 msg={"topic":"sensor","value":148}
{"topic":"logServer","nodeId":2385987433}
CONNECTION: stationScan(): whateverYouLike
```

รูปที่ 4.8 Mesh network เมื่อปิด ESP32 ตัวที่ 2

หลังจากได้ทดลองปิดตัวที่ 2 แล้ว ที่นี้ได้ลองเปิดตัวที่ 2 อีกครั้งหนึ่ง พร้อมกับสังเกตการทำงานบนตัว server พบว่าตัว server สามารถเชื่อมตัวกับตัวที่ 2 และตัวที่ 3 ได้ แสดงให้เห็นว่า ESP32 3 ตัวนี้เชื่อมต่อกันในแบบ mesh network เช่น ตอนแรก server node พบ client node ตัวที่ 3 คือ ID 2382783801 หลังจากเปิด client ตัวที่ตัวที่สองซึ่งมีหมายเลข 983847501 แสดงดังรูปที่ 4.9

```
S_TIME: handleTimeSync(): Response sent {"type":2,"t0":2518352559,"t1":2518378704,"t2":2518378960}
S_TIME: handleTimeSync(): timeSyncStatus with 2382783801 completed
S_TIME: handleTimeSync(): ----------------------------------
S_TIME: handleTimeSync(): Response sent {"type":2,"t0":2518593833,"t1":2518592988,"t2":2518593239}
S_TIME: handleTimeSync(): timeSyncStatus with 2382783801 completed
S_TIME: handleTimeSync(): ----------------------------------
S_TIME: handleTimeSync(): Response sent {"type":2,"t0":2518799788,"t1":2518807061,"t2":2518807313}
S_TIME: handleTimeSync(): timeSyncStatus with 2382783801 completed
S_TIME: handleTimeSync(): ----------------------------------
logServer: Received from 2382783801 msg={"topic":"sensor","value":169}
{"topic":"logServer","nodeId":2385987433}
logServer: Received from 983847501 msg={"topic":"sensor","value":139}
logServer: Received from 2382783801 msg={"topic":"sensor","value":148}
logServer: Received from 983847501 msg={"topic":"sensor","value":54}
{"topic":"logServer","nodeId":2385987433}
```

รูปที่ 4.9 ทดลองเชื่อมต่อ ESP32 ตัวที่ 2 อีกครั้งหนึ่ง

4.4 ผลของการทำงานของ Client node กับการชั่งน้ำหนัก

ผู้จัดทำได้ทำการนำโปรแกรมในส่วนของการชั่งน้ำหนักมารวมกับโปรแกรมในส่วนของการสื่อสารแบบ Mesh network เพื่อที่จะให้ Client node สามารถที่ชั่งน้ำหนักของสิ่งของและสามารถส่งค่าน้ำหนักแบบ real time ไปยัง node ตัวอื่นๆซึ่งจะสื่อสารกันแบบ Mesh network ได้ดังรูปที่ 4.10

```
22:45:13.973 -> logServer detected!!!
22:45:13.973 -> Handled from 3218191645 msg={"topic":"logServer","nodeId":3218191645}
logClient: Received from 3218191645 msg={"topic":"logServer","nodeId":3218191645}
22:45:14.896 -> logServer detected!!!
22:45:14.896 -> Handled from 3218191645 msg={"topic":"logServer","nodeId":3218191645}
{"1":"mcu-t3B","2":983847501,"3":"-0.06"}
CONNECTION: eventScanDoneHandler: SYSTEM_EVENT_SCAN_DONE
CONNECTION: scanComplete():-- > scan finished @ 169916625 < --
22:45:16.357 -> CONNECTION: scanComplete():-- > Cleared old aps.
22:45:16.357 -> CONNECTION: scanComplete(): num = 23
22:45:16.357 -> CONNECTION:     found : PREM , -49dBm
22:45:16.357 -> CONNECTION:     found : PREM , -55dBm
22:45:16.357 -> CONNECTION:     found : PREM , -66dBm
22:45:16.357 -> CONNECTION:     Found 3 nodes
CONNECTION: findConnection(3209007053): did not find connection
22:45:16.461 -> CONNECTION: findConnection(2385987433): did not find connection
{"1":"mcu-t3B","2":983847501,"3":"-0.06"}
CONNECTION: connectToAP(): Best AP is 3209007053<---
22:45:16.636 -> CONNECTION: connectToAP(): Trying to connect, scan rate set to 4*normal
{"1":"mcu-t3B","2":983847501,"3":"-0.06"}
{"1":"mcu-t3B","2":983847501,"3":"-0.18"}
{"1":"mcu-t3B","2":983847501,"3":"-0.61"}
{"1":"mcu-t3B","2":983847501,"3":"-0.70"}
{"1":"mcu-t3B","2":983847501,"3":"-0.70"}
{"1":"mcu-t3B","2":983847501,"3":"-0.70"}
CONNECTION: onDisconnect():
22:45:25.879 -> CONNECTION: onDisconnect(): dropping 3218191645 now= 179434968
22:45:25.879 -> CONNECTION: MeshConnection::close().
```

รูปที่ 4.10 Serial monitor ของ Client node

จากรูปที่ 4.10 จะเห็นได้ว่าเมื่อเปิด Serial monitor ของ Client node จะแสดงค่าน้ำหนัก ณ ขณะนั้นออกมา ซึ่งก็คือ 0 kg พร้อมกับสามารถที่จะสื่อสารกับ Server node ที่มี ID คือ 3218191645 ได้ และยังสามารถรู้ได้ว่ามี node ที่ตรวจพบอีก 3 node

4.5 ผลของการทำงานของ Server node กับการชั่งน้ำหนัก

ในกรณีของ Server node จะไม่มีการชั่งน้ำหนัก แต่จะรับค่าน้ำหนักจาก Client node หลายๆตัว ซึ่งส่งข้อมูลแบบ Mesh network แบบ real time จากรูปจะเห็นว่า server node พบ client node 3 node คือ ID 983847501,ID 3209007053 และ ID 2385987433 และ server node คือ ID 3218191645 ดังรูปที่ 4.11

```
CONNECTION: eventScanDoneHandler: SYSTEM_EVENT_SCAN_DONE
19:04:39.420 -> CONNECTION: scanComplete():-- > scan finished @ 1152507991 < --
19:04:39.420 -> CONNECTION: scanComplete():-- > Cleared old aps.
19:04:39.454 -> CONNECTION: scanComplete(): num = 42
19:04:39.454 -> CONNECTION:     found : PREM , -51dBm
19:04:39.454 -> CONNECTION:     found : PREM , -62dBm
19:04:39.454 -> CONNECTION:     found : PREM , -67dBm
19:04:39.454 -> CONNECTION:     Found 3 nodes
19:04:39.454 -> CONNECTION: findConnection(983847501): did not find connection
19:04:39.454 -> CONNECTION: findConnection(3209007053): did not find connection
19:04:39.454 -> CONNECTION: findConnection(2385987433): did not find connection
19:04:39.454 -> CONNECTION: connectToAP(): Best AP is 983847501<---
19:04:39.489 -> CONNECTION: connectToAP(): Trying to connect, scan rate set to 4*normal
19:04:39.489 -> {"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
```

รูปที่ 4.11 Serial monitor ของ Server Node เมื่อพบ Client Node 3ตัว

```
{"topic":"logServer","nodeId":3218191645}
logServer: Received from 3209007053 msg={"1":"mcu-t2B","2":3209007053,"3":"-0.02"}{"1":"mcu-t2B","2":3209007053,"3":"-0.02"}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
{"topic":"logServer","nodeId":3218191645}
19:07:07.467 -> logServer: Received from 983847501 msg={"1":"mcu-t3B","2":983847501,"3":"-0.12"}{"1":"mcu-t3B","2":983847501,"3":"-0.12"}
{"topic":"logServer","nodeId":3218191645}
19:07:07.567 -> logServer: Received from 3209007053 msg={"1":"mcu-t2B","2":3209007053,"3":"0.03"}{"1":"mcu-t2B","2":3209007053,"3":"0.03"}
{"topic":"logServer","nodeId":3218191645}
```

รูปที่ 4.12 Serial monitor ของ Server node แสดงค่าน้ำหนักของแต่ละ Node

จากรูปที่ 4.12 จะเห็นได้ว่า เมื่อเปิด Serial monitor ของ Server node ตัว Server node สามารถที่จะตรวจพบ Client node ได้ 3 ตัว และแสดงค่าน้ำหนักได้ว่า ณ ขณะนั้น Client node แต่ละตัวมีค่าน้ำหนักเท่าใด เช่น node mcu ตัวที่ 3 ซึ่งมี ID เป็น 983847501 มีค่าน้ำหนักที่ชั่ง ณ ขณะนั้น คือ -0.12 kg และ node mcu ตัวที่ 2 มี ID เป็น 3209007053 และมี server node คือ ID คือ 3218191645

4.6 ผลการทดสอบและวิเคราะห์ผลการสอบ

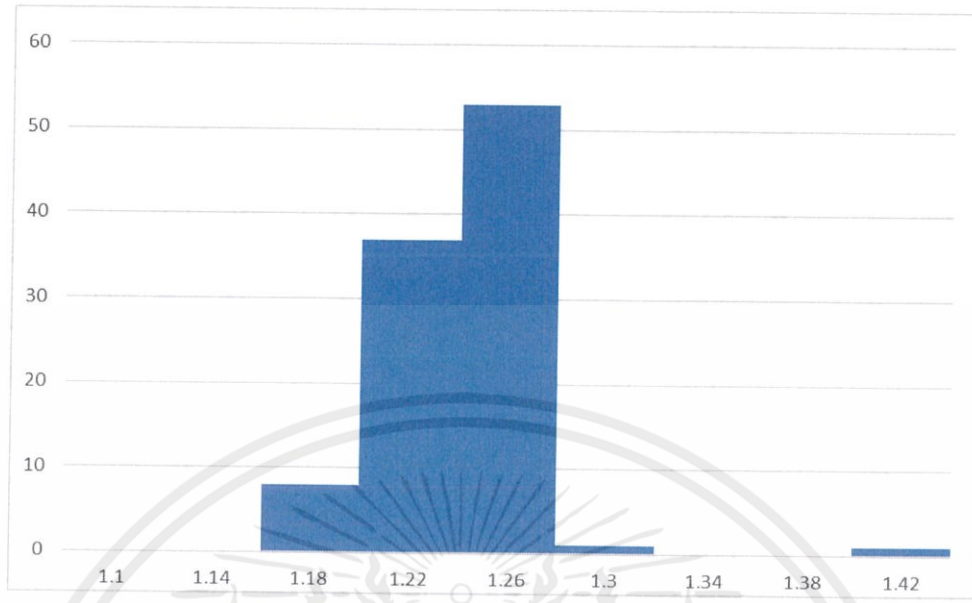
จากผลการทดลองจะพบว่ามีค่าผิดพลาดที่เกิดขึ้นในการทำงานของเครื่องชั่งน้ำหนักโดยวิเคราะห์จากค่า Variance หรือ ค่าความแปรปรวนของข้อมูล โดยข้อมูลที่นำมาวิเคราะห์เป็นตัวอย่างแค่บางส่วน (Sample data) โดยทดลองชั่งแผ่นเหล็ก ดัมเบลทั้งหมด 100 ครั้ง ในการหาค่า Mean ของ Variance นั้นจะใช้ N-1 แตกต่างจากหากคำนวนจากข้อมูลทั้งหมด (Population) ที่ใช้ N คือจำนวนทั้งหมดในการหาค่า Mean จะใช้สมการ

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - x_{avg})^2}{n-1}$$

โดยหาค่าสูงสุดเท่ากับ 1.39 และค่าต่ำสุดเท่ากับ 1.16 และค่าน้ำหนักเฉลี่ยของน้ำหนักทั้งหมดเท่ากับ 1.2296 กิโลกรัม รวมถึงค่าความแปรปรวนเท่ากับ 0.000921051 จะพบว่าค่าการกระจายของข้อมูลมีน้อยมาก และพล็อตกราฟฮิสโตแกรม หาความสัมพันธ์ระหว่างข้อมูลเพื่อกระจายความถี่ของข้อมูล ซึ่งข้อมูลจะเป็นหมวดหมู่โดยจะเรียงลำดับจากน้อยไปหามาก แกนตั้งจะเป็นตัวเลขที่แสดง “ความถี่” และแกนนอนเป็นข้อมูลของน้ำหนัก แท่งกราฟแต่ละแท่งมีความกว้างเท่ากันซึ่งจะเท่ากับความกว้างของชั้นข้อมูล ส่วนความสูงของกราฟแต่ละแท่งนั้นจะสูงเท่ากับความถี่ของแต่ละชั้นข้อมูล แผนภูมิฮิสโตแกรมนี้แสดงให้เห็นถึงความเบี่ยงเบนของข้อมูลว่ามีลักษณะการกระจายตัวของข้อมูลเป็นแบบระฆังคว่ำหรือไม่ หรือมีความเบี่ยงเบนไปทางบวกหรือลบซึ่งกำหนดการแบ่งอันตรภาคชั้นเพิ่มขึ้นชั้นละ 0.04 จะได้จำนวนความถี่ในแต่ละครั้งที่เหมือนกัน ซึ้งสามารถแสดงได้ดังตารางที่ 4.1 และรูปที่ 4.8

ตารางที่ 4.1 ค่าอันตรภาคชั้นและจำนวนความถี่ของแต่ละน้ำหนักที่วัดได้

อันตรภาคชั้น	ความถี่
1.1	0
1.14	0
1.18	8
1.22	37
1.26	53
1.30	1
1.34	0
1.38	0
1.42	1



รูปที่ 4.13 กราฟฮิสโตแกรมแสดงความสัมพันธ์ระหว่างน้ำหนักที่วัดได้กับจำนวนความถี่

จากการทดลองข้างต้น ผู้จัดทำได้ลองวัดระยะการเชื่อมต่อของ ESP32 ทั้ง 3 ตัว ตัว พร้อมกับวัดความแรงสัญญาณที่ ESP32 ได้ปล่อยค่าออกมา ทั้งแบบในตัวอาคารกับนอกอาคารดังรูปที่ 4.15 และ รูปที่ 4.16 ผู้จัดทำได้นำ ESP32 แต่ละตัวมาวางแบบ chain network เพื่อที่ให้ง่ายต่อการวัดระยะทางของการสื่อสารของ ESP32 แต่ละตัว โดยรูปแบบของ chain netwok สามารถแสดงได้ดังรูปที่ 4.14 แบบในตัวอาคาร ขั้นแรกได้ทำการวัดระยะในตัวอาคาร 5 ครั้ง จาก ESP32 ที่เป็น Server node กับ ESP32 ที่เป็น Client node ตัวที่ 1 ตามตารางที่ 4.2

ตารางที่ 4.2 วัดระยะและความแรงสัญญาณในตัวอาคารระหว่าง Server Node กับ Client Node ตัวที่ 1

การวัดระยะในตัวอาคารในขั้นแรก	ระยะห่างจาก Server node กับ Client node ตัวที่ 1	ความแรงของสัญญาณ
1	23.11 เมตร	-88 dB
2	25.68 เมตร	-90 dB
3	23.89 เมตร	-91 dB
4	24.67 เมตร	-88 dB
5	27.23 เมตร	-92 dB

จากนั้นได้ทำการหาค่าเฉลี่ยระยะทางจากตัว Server Node กับ Client Node ตัวที่ 1 ในอาคารได้ 24.91 เมตร ในขั้นที่สองได้ทำการวัดระยะในตัวอาคาร 5 ครั้ง จาก ESP32 ที่เป็น Client node ตัวที่ 1 กับ Client node ตัวที่ 2 ตามตารางที่ 4.3

ตารางที่ 4.3 วัดระยะและความแรงสัญญาณในตัวอาคารระหว่าง Client Node ตัวที่ 1 กับ ตัวที่ 2

การวัดระยะในตัวอาคารในขั้นที่สอง	ระยะห่างจาก Client node ตัวที่ 1 กับตัวที่ 2	ความแรงสัญญาณ
1	23.12 เมตร	-92
2	23.45 เมตร	-89
3	22.14 เมตร	-91
4	23.87 เมตร	-89
5	22.69 เมตร	-89

จากนั้นได้ทำการหาค่าเฉลี่ยระยะทางจากตัว Client Node ตัวที่ 1 กับ Client Node ตัวที่ 2 ในอาคารได้ 23.05 เมตร แบบนอกอาคาร ขั้นแรกได้ทำการแบบนอกตัวอาคาร 5 ครั้ง จาก ESP32 ที่เป็น Server node กับ ESP32 ที่เป็น Client node ตัวที่ 1 ตามตารางที่ 4.4

ตารางที่ 4.4 วัดระยะและความแรงสัญญาณนอกตัวอาคารระหว่าง Server Node กับ Client Node ตัวที่ 1

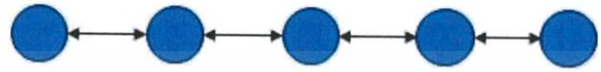
การวัดระยะนอกตัวอาคารในขั้นแรก	ระยะห่างจาก Server node กับ Client node ตัวที่ 1	ความแรงของสัญญาณ
1	82..51 เมตร	-92 dB
2	80.12 เมตร	-93 dB
3	81.45 เมตร	-90 dB
4	82.43 เมตร	-94 dB
5	81.89 เมตร	-88 dB

จากนั้นได้ทำการหาค่าเฉลี่ยระยะทางจากตัว Server Node กับ Client Nodeตัวที่ 1 นอกอาคารได้ 81.68 เมตร ในขั้นที่สองได้ทำการวัดระยะนอกตัวอาคาร 5 ครั้ง จาก ESP32 ที่เป็น Client Node ตัวที่ 1 กับ Client Node ตัวที่ 2 ตามตารางที่ 4.5

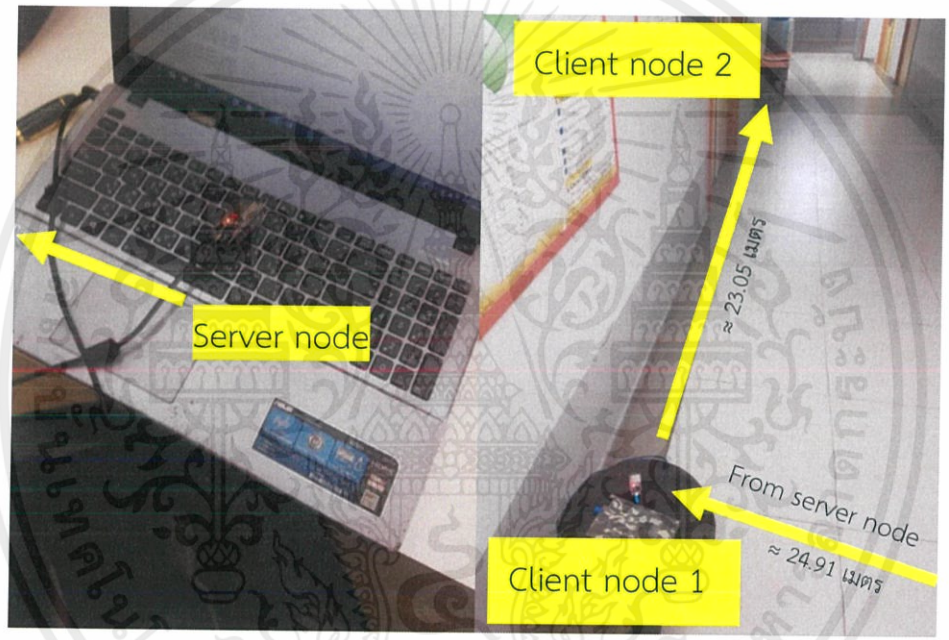
ตารางที่ 4.5 วัดระยะและความแรงสัญญาณนอกตัวอาคารระหว่าง Client Node ตัวที่ 1 กับ ตัวที่ 2

การวัดระยะนอกตัวอาคารในขั้นสอง	ระยะห่างจาก Client node ตัวที่ 1 กับตัวที่ 2	ความแรงของสัญญาณ
1	80.56 เมตร	-91 dB
2	81.47 เมตร	-90 dB
3	80.45 เมตร	-92 dB
4	79.81 เมตร	-91 dB
5	80.21 เมตร	-92 dB

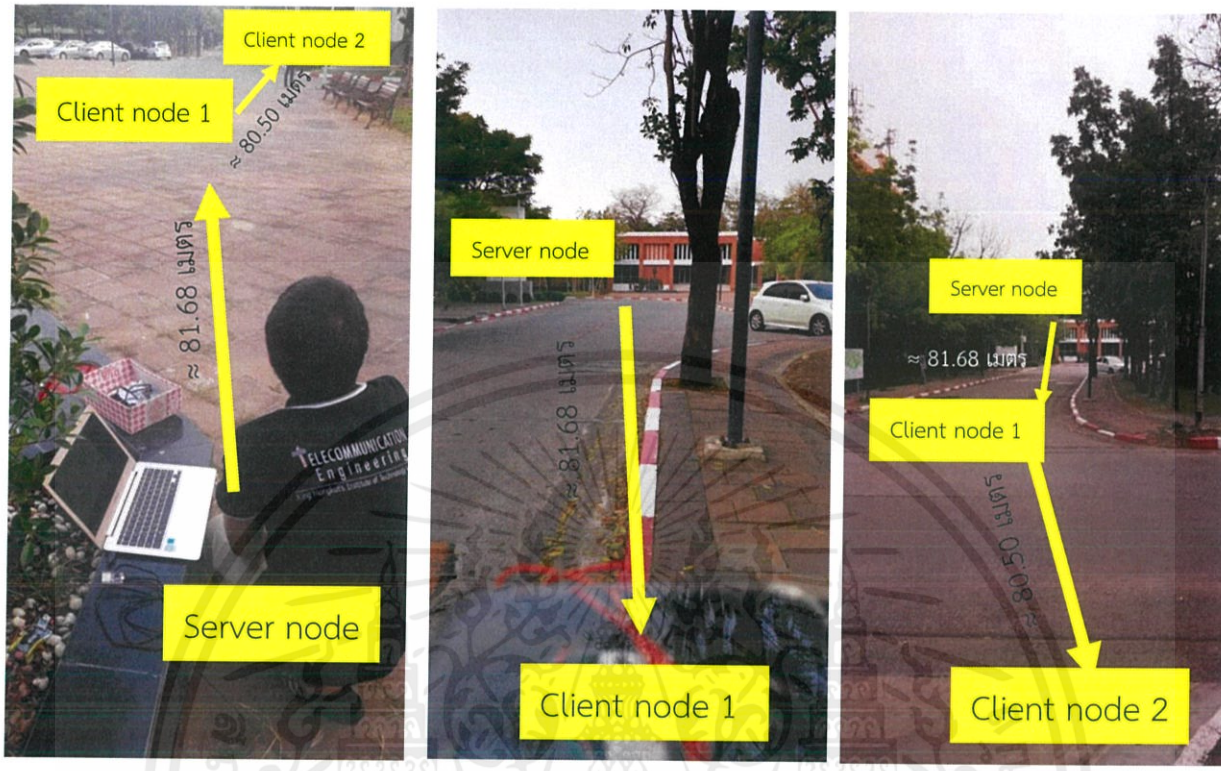
จากนั้นทำการหาค่าเฉลี่ยระยะทางจากตัว Client Node ตัวที่ 1 กับ Client Node ตัวที่ 2 นอกตัวอาคารได้ 80.5 เมตร



รูปที่ 4.14 การสื่อสารแบบ chain network



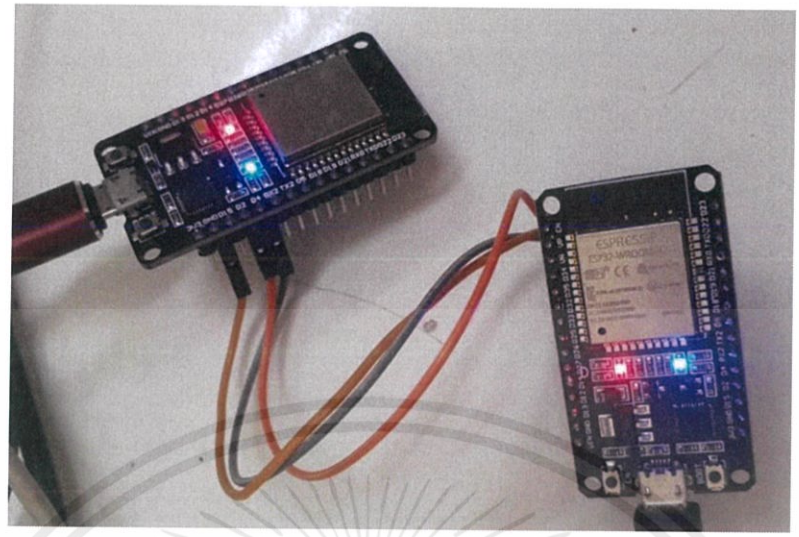
รูปที่ 4.15 วัดระยะแบบ indoor ภายในอาคาร



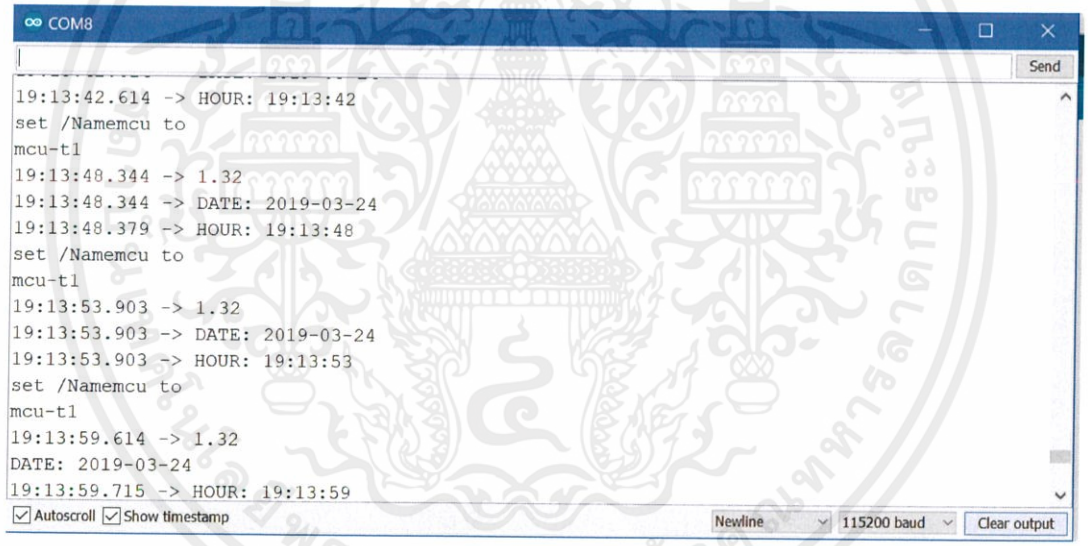
รูปที่ 4.16 วัดระยะแบบ outdoor ภายนอกอาคาร

4.7 ผลการทดสอบการทำงานของ Gateway (Bridge Communication)

ทำการทดสอบการส่งข้อมูลผ่าน Serial Port ระหว่าง Server Node และ Gateway โดยต่อขา Rx และ Tx และต่อ GND ในแต่ละครั้งของการวัดน้ำหนักจะค่าส่งน้ำหนัก Node ID เวลาและวันที่ไปยัง Firebase Server ซึ่งเป็นฐานการให้บริการข้อมูลแบบเรียลไทม์ ซึ่งคุณสมบัติของ ESP32 ที่รองรับการเชื่อมต่อ WiFi ได้อีกทั้งยังมีคุณบัติมัลติทาสกิ้ง (Multitasking) คือการทำงานของโปรแกรม 2 โปรแกรมพร้อมกัน โดยค่าที่ส่งไปนั้นจะส่งวันและเวลาส่งไปพร้อมๆกัน

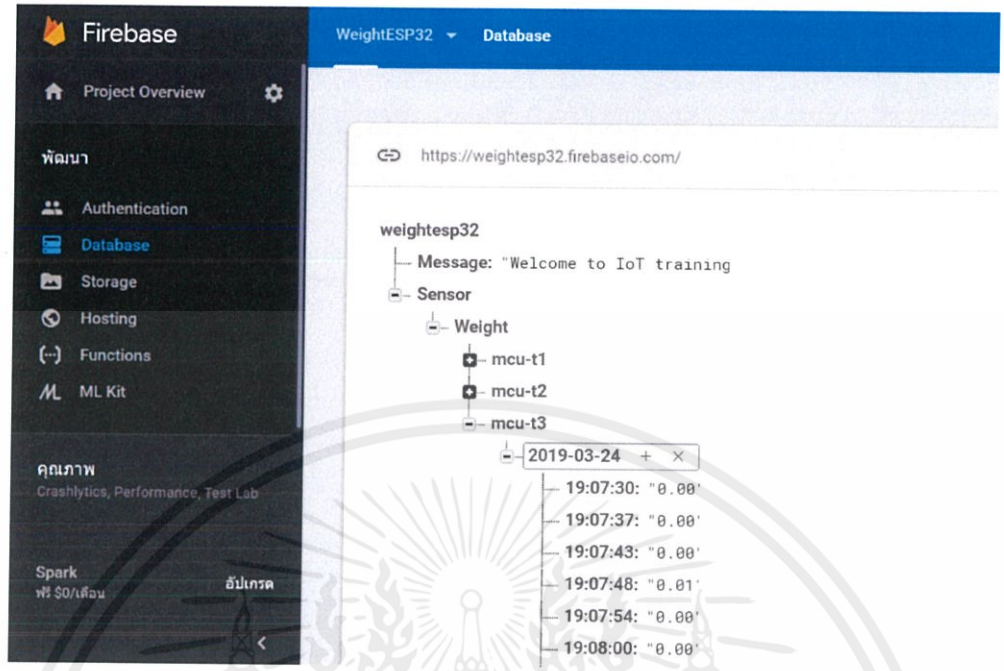


รูปที่ 4.17 การเชื่อมต่อระหว่าง ESP32 2 ตัว



รูปที่ 4.18 ผลการทดสอบวัดของน้ำหนักของแต่ละโหนด พร้อมบอกวันและเวลา

จากรูปที่ 4.17 เมื่อทำการเชื่อมต่อแล้ว ได้ทำการรันโปรแกรม เมื่อเปิด Serial monitor ของ Gateway node จะเห็นได้ว่าได้รับค่าน้ำหนักจาก Server node เช่น mcu-t1 ส่งค่าน้ำหนักที่ชั่งได้ ณ ขณะนั้นเวลา 19.13 นาฬิกาในวันที่ 24 มีนาคม 2019 คือ 1.32 kg ดังรูปที่ 4.18



รูปที่ 4.19 แสดงน้ำหนัก ณ วันและเวลานั้นๆ

การแสดงค่าน้ำหนักที่แสดงเวลาโดยใช้ไลบรารี่ NTPClient คือ networking protocol ที่ใช้สำหรับ sync time ของ server ทุกเครื่องใน network ให้ตรงกัน ผ่าน packet-switch ที่ Serial Monitor และรูปที่ 4.19 เป็นการแสดงค่าข้อมูลที่ถูกส่งมาเก็บไว้ใน Firebase โดยใช้คำสั่งผ่านทางฟังกชั่น

จากนั้นใช้ Authentication และ realtime database สำหรับใช้งาน Firebase กับ Web โดยเลือกสร้างโปรเจคที่ Firebase และลง firebase-tools และ login ลงในเครื่องเราก่อน โดยในระบบการ login เข้าใช้งานต้องมีการ verify ด้วย Email ซึ่งใช้ feature ของ firebase ดังในรูปที่ 4.20

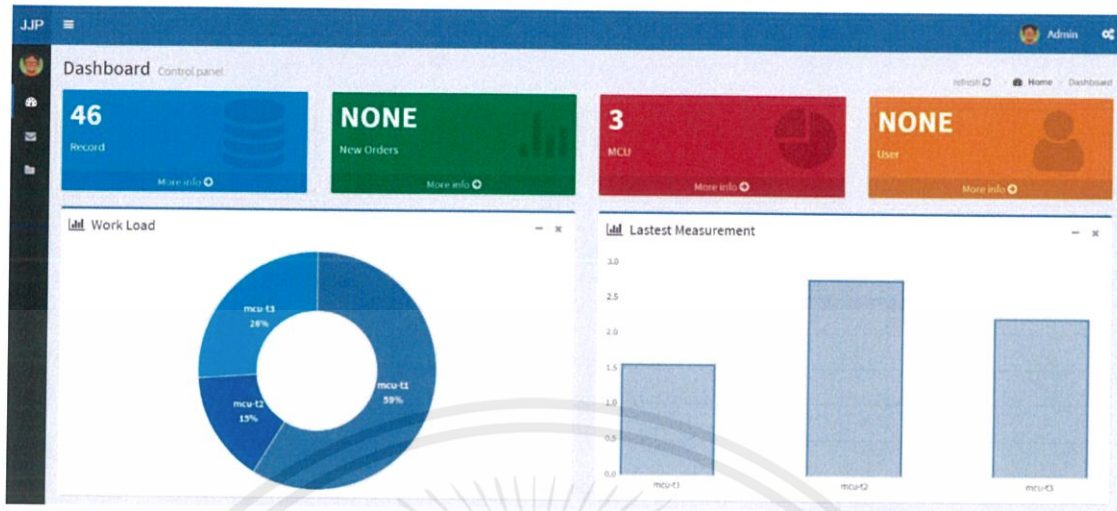
รูปที่ 4.20 หน้า Web loginไปที่

รูปที่ 4.21 หน้าเว็บแสดงค่าน้ำหนัก วันและเวลา

จากรูปที่ 4.21 แสดงหน้า Web ซึ่งเป็น feature ของ Firebase ซึ่งใช้ realtime database ส่วนนี้จะเป็น cloud database แบบ NoSQL ให้เราสร้าง database ไว้ใช้งาน ข้อมูลจะ syncs แบบ realtime และยังสามารถใช้งานแบบ Offline ได้ด้วย การ syncs ข้อมูลนั้นจะใช้ script และ config ในรูปแบบ json object ของ firebase มาใช้ได้เลย

```
{  "Message" : "Welcome to IoT training.",
        "Sensor" :
                {    "Weight" :
                        {       "mcu-t1" :
                                { "2019-03-24" :
                                        {  "19:12:02" : "1.32", "19:12:08" : "1.32", }
                                },
                                "mcu-t2" :
                                {  "2019-03-24" :
                                        {       "19:06:51" : "-0.01", "19:06:56" : "-0.02", }
                                },
                                "mcu-t3" :
                                {       "2019-03-24" :
                                        {       "19:07:30" : "0.00", "19:07:37" : "0.00", }
                                }
                        }
                }
}
```

รูปที่ 4.22 ข้อมูล JSON ของ firebase



รูปที่ 4.23 หน้าเว็บแสดงน้ำหนักเป็นกราฟแท่ง

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

โครงงานนี้เป็นการสร้างเครื่องชั่งน้ำหนัก โดยใช้ ESP32 ซึ่งเป็นไอซีไมโครคอนโทรเลอร์ที่รองรับการเชื่อมต่อ Wifi ไม่เพียงแค่ให้ ESP32 เป็นเครื่องชั่งน้ำหนักเท่านั้น แต่ยังให้ ESP32 แต่ละตัวสื่อสารกันแบบ Mesh Network กล่าวคือ ให้ ESP32 ที่เป็นเครื่องชั่งน้ำหนักทำหน้าที่เป็น Client node จะทำการตรวจหาว่าในเครือ Mesh network เดียวกันนี้ว่า ESP32 หมายเลข ID ใดเป็น Server node โดยที่ Client node จะทำการ boradcast message ไปเรื่อยๆเพื่อที่จะตรวจหา Server node ในขณะเดียวกัน Server node จะทำการ Broadcast message ออกไปเช่นกัน เพื่อที่จะบ่งบอก Client node ในเครือ Mesh network เดียวกันว่าเป็น Server node เมื่อพบแล้วจะทำการส่งค่าน้ำหนักที่ชั่งได้ ณ ขณะนั้นไปยัง Server node ผู้จัดทำได้ทำการวัดระยะเฉลี่ยของการสื่อสารระหว่าง Node กับ Node ว่ามีระยะเท่าใดทั้งในตัวอาคารและนอกอาคาร พบว่าในตัวอาคารมีระยะเฉลี่ยของการสื่อสารระหว่าง Node ด้วยกันคือ และภายนอกอาคารมีระยะเฉลี่ยของการสื่อสารนระหว่าง Node ด้วยกันคือ เห็นได้ว่าระยะเฉลี่ยการสื่อสาร ระหว่าง Node ด้วยกันแบบภายนอกอาคารมีระยะเฉลี่ยที่มากกว่าแบบภายในอาคาร ทั้งนี้เกิดจากไม่มีสิ่งอุปสรรคหรือสัญญาณรบกวนไปรบกวนการสื่อสารของ ESP32 หลังจากที่ Server node ได้รับค่าน้ำหนักจาก Client node ทุกตัวแบบ real time แล้ว จะทำการส่งข้อมูลทั้งหมดไปยัง Gateway node แบบ Bridge communication ซึ่งเชื่อมต่อกับ Internet ไว้ ในการสร้างเครื่องชั่งน้ำหนัก ได้ให้ ESP 32 ต่อกับ Load cell โดยมี HX711 เป็นตัวแปลงสัญญาณอนาลอกเป็นดิจิตอลผลการทดสอบการชั่งน้ำหนักเพื่อดูความคลาดเคลื่อนของน้ำหนักที่ชั่งได้

จากการทดสอบชั่งน้ำหนักขนาด 1.25 กิโลกรัม ทั้งหมด 100 ครั้งโดยจะหยิบแผ่นเหล็กดัมเบลออกทุกๆการส่งค่า 5-6 ครั้งให้น้ำหนักเป็น 0 แล้ววางชั่งใหม่ พบว่ามีความคลาดเคลื่อนประมาณ 0.20 กิโลกรัม และเอาข้อมูลแต่ครั้งมาหาค่าความแปรปรวนของข้อมูล มีค่าอยู่ที่ 0.000921051 ค่านี้บ่งบอกถึงการกระจายตัวของข้อมูลที่น้อยมาก และนำเสนอข้อมูลในรูปของกราฟฮิสโตแกรมเพื่อหาความสัมพันธ์การกระจายความถี่ของข้อมูล พบว่ารูปกราฟฮิสโตแกรมที่มีลักษณะเบ้ขวา (Positively skewed histogram)คือยอดกราฟไม่ได้อยู่ตรงกลาง แต่จะเอนไปทางซ้ายมือค่าเฉลี่ยของข้อมูลมีค่าค่อนข้างต่ำ

5.2 ข้อเสนอแนะ

จากการทดลอบการทำงานของเครื่องชั่งน้ำหนักพบว่ามีความคลาดเคลื่อนในการชั่งน้ำหนักแต่ละครั้ง เพื่อให้เกิดความแม่นยำต้อง Calibrate ทุกครั้งก่อนชั่ง ซึ่งทำให้สิ้นเปลืองเวลา ดังนั้นเพื่อให้เครื่องชั่งน้ำหนักมีประสิทธิภาพมากขึ้น เวลาชั่งน้ำหนักให้มีการ Calibrate น้ำหนักโดยอัตโนมัติ และปรับปรุงค่าความคลาดเคลื่อนของน้ำหนักที่วัดได้ ปรับขนาด Load cell ให้สามารถรองรับน้ำหนักได้มากขึ้นเพื่อการใช้งานจริงในอนาคตและในการสื่อสารของ ESP32 ของแต่ละตัว และ

การคิดคำนวณอัตราการส่งข้อมูล ต้องดูให้เหมาะสมกับ application ด้วยเพราะ esp32 มีข้อจำกัดเรื่องของการประมวลผลและหน่วยความจำ esp32 อาจต้องทำงานหนักขึ้น

ระบบ Mesh Network ถึงจะเป็นระบบที่ค่อนข้าง stable แต่ก็มีโอกาสทำงานผิดพลาดได้ ฉะนั้นข้อมูลที่ส่งผ่านไปก็อาจจะหายไปได้บ้างในบางโอกาส ฉะนั้นก็ขึ้นอยู่กับ logic ของ application ที่เราพัฒนาว่าจะรับมืออย่างไรและควรทำในพื้นที่เปิดโล่งหรือมีสิ่งกีดขวางให้น้อยที่สุดเพื่อที่จะให้ระยะของการสื่อสารแต่ละตัวไม่ถูกรบกวนและมีระยะการเชื่อมต่อที่ไม่น้อยจนเกินไป

บรรณานุกรม

[1] บริษัท วีนัส ซัพพลาย จำกัด 66/3 ถ.เทศบาลรังสรรค์เหนือ แขวงลาดยาว เขตจตุจักร กทม. 10900. "วิธีการใช้งาน Load Cell กับ HX711 Amplifier Module.". (CD-ROM). https://www.thaieasyelec.com/article-wiki/review-product-article/how-to-use-load-cell-and-hx711-amplifier-module.html.

[2] Andreas Spiess. "Connected Cat Feeder Using a Strain Gauge and an ESP32." https://www.instructables.com/id/Connected-Cat-Feeder-Using-a-Strain-Gauge-and-an-E/#discuss

[3] Joe D.S. "ESP8266 / ESP32 & Mesh Network ตอนที่ 1: Introduction & Painlessmesh." http://meetjoeblog.com/2018/03/25/esp8266-esp32-mesh-network-ep1/.

[4] TridentTD. "(ตอนที่ 2) เชื่อมต่อ ESP32 กับ Firebase แบบไม่ต้องพึ่งไลบรารี่ภายนอก." https://tridenttd.wordpress.com/2018/03/17/

ภาคผนวก ก

โปรแกรมการ Calibrate น้ำหนัก
สำหรับบอร์ด ESP32 ด้วยภาษา C

```
#include "HX711.h"
#include <Wire.h>
#include"soc/rtc.h"

#define DOUT  26
#define CLK   25

#define DEC_POINT  2
#define STABLE  1

float offset=0;
float calibration_factor = 100;
float real_weight = 2.50;//kg

HX711 scale(DOUT, CLK);

unsigned char state=0;
long  FindZeroFactor();
float get_units_kg();
void  ReadWeight();
void  FindCalibrationFactor();

void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.println("Auto Calibrate Program");
  Serial.println("Send 'a' to Find Zero Factor (Please Remove all weight from scale)");
  Serial.println("Send 'b' to Find Calibration Factor (Please insert know the weight on the scales)");
  Serial.println("Send 'c' Show weight on the scales");
  rtc_clk_cpu_freq_set(RTC_CPU_FREQ_80M);
}
```

```
void loop()
{
  if(Serial.available())
  {
    char temp = Serial.read();
    if(temp=='a')
      state=1;
    if(temp=='b')
      state=2;
    if(temp=='c')
      state=3;
  }

  switch(state)
  {
    case 0:
    break;
    case 1:
     FindZeroFactor();
   //   ReadWeight();
      state=0;
    break;
    case 2:
     FindCalibrationFactor();
     state=0;
    break;
    case 3:
      ReadWeight();
      delay(200);
    break;
    case 4:

    break;

  }
}
```

```
long FindZeroFactor()
{
    Serial.println("Find Zero Factor");
    Serial.println("Please wait .....");
    scale.set_scale();
    scale.tare();
    long zero_factor = scale.read_average(20);
    Serial.print("Zero factor: ");
    Serial.println(zero_factor);
    return(zero_factor);
}

void FindCalibrationFactor()
{
  unsigned char flag_stable=0;
  unsigned int decpoint=1;
  for(unsigned char i=0;i<DEC_POINT+1;i++ )
    decpoint = decpoint*10;
  while(1)
  {
    scale.set_scale(calibration_factor); //Adjust to this calibration factor
    Serial.print("Reading: ");
    float read_weight = get_units_kg();
    String data = String(read_weight, DEC_POINT);
    Serial.print(data);
    Serial.print(" kg");
    Serial.print(" calibration_factor: ");
    Serial.print(calibration_factor);
    Serial.println();
    long r_weight      = (real_weight*decpoint);
    long int_read_weight = read_weight*decpoint;
    Serial.print(r_weight);
    Serial.print(" , ");
    Serial.println(int_read_weight);
    long x;
```

```
        if(r_weight == int_read_weight)
        {
          flag_stable++;
          if(flag_stable>=STABLE)
          {
            Serial.print("Calibration Factor is = ");
            Serial.println(calibration_factor);
            break;
          }
         }
        if(r_weight > int_read_weight)
          {
            x = r_weight - int_read_weight;
            if(x > 100)
              calibration_factor -= 1000;
            else if(x > 100)
              calibration_factor -= 10;
            else
              calibration_factor -= 1;
            flag_stable=0;
          }
          if(r_weight < int_read_weight)
          {
            x =  int_read_weight-r_weight;
            if(x > 100)
              calibration_factor += 1000;
            else if(x > 10)
              calibration_factor += 10;
            else
              calibration_factor += 1;
            flag_stable=0;
          }
    }
}

float get_units_kg()
```

```
{
  return(scale.get_units()*0.453592);
}
void ReadWeight()
{
  scale.set_scale(calibration_factor);
  Serial.print("Reading: ");
  String data = String(get_units_kg()+offset, DEC_POINT);
  Serial.print(data);
  Serial.println(" kg");
}
```

ภาคผนวก ข

โปรแกรมส่วนของ Client
สำหรับบอร์ด ESP32 ด้วยภาษา C

```
#include <WiFi.h>
#include "painlessMesh.h"
#include <HX711.h>
#include"soc/rtc.h"
float calibration_factor = 51135.00;
#define zero_factor 8239331
#define DOUT  26
#define CLK   25
#define DEC_POINT  2
String data1 ;
float offset = 0;
float get_units_kg();

#define   MESH_PREFIX     "PREM "
#define   MESH_PASSWORD   "0869634090"
#define   MESH_PORT       5555

HX711 scale(DOUT, CLK);
Scheduler     userScheduler;
painlessMesh  mesh;
void receivedCallback( uint32_t from, String &msg );
size_t logServerId = 0;
// Send message to the logServer every 5 seconds
Task myLoggingTask(1500, TASK_FOREVER, []() {

#if ARDUINOJSON_VERSION_MAJOR==6
  DynamicJsonDocument jsonBuffer;
  JsonObject msg = jsonBuffer.to<JsonObject>();
#else
  DynamicJsonBuffer jsonBuffer;
  JsonObject& msg = jsonBuffer.createObject();
#endif
```

```
    msg["1"] = "mcu-t1B";  //change for identify for the node that send
data mcu-t1 to mcu-t3
    msg["2"] = mesh.getNodeId();
    msg["3"] = String(data1);
    String str;
    msg.printTo(str);
   #if ARDUINOJSON_VERSION_MAJOR==6
    serializeJson(msg, str);
   #else
    msg.printTo(str);
   #endif
    if (logServerId == 0) // If we don't know the logServer yet
      mesh.sendBroadcast(str);
    else
      mesh.sendSingle(logServerId, str);

    // log to serial
   #if ARDUINOJSON_VERSION_MAJOR==6
    serializeJson(msg, Serial);
   #else
    msg.printTo(Serial);
   #endif
    Serial.printf("\n");
   });

   void setup()
   {
    Serial.begin(115200);
    Serial.println("Load Cell");
    scale.set_scale(calibration_factor);
    scale.set_offset(zero_factor);
    rtc_clk_cpu_freq_set(RTC_CPU_FREQ_80M);
```

```
        mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION );  // set before init() so that you can see startup messages

        mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT, WIFI_AP_STA, 6 );
        mesh.onReceive(&receivedCallback);

        // Add the task to the your scheduler
        userScheduler.addTask(myLoggingTask);
        myLoggingTask.enable();

      }
      void loop()
      {
        //Serial.print("Reading: ");
        data1 = String(get_units_kg() + offset, DEC_POINT);
        //Serial.print(data1);
        //Serial.println(" kg");
        userScheduler.execute(); // it will run mesh scheduler as well
        mesh.update();
      }
      float get_units_kg()
      {
        return (scale.get_units() * 0.453592);

      }

      void receivedCallback( uint32_t from, String &msg ) {
        Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
```

```
  // Saving logServer
#if ARDUINOJSON_VERSION_MAJOR==6
  DynamicJsonDocument jsonBuffer;
  DeserializationError error = deserializeJson(jsonBuffer, msg);
  JsonObject root = jsonBuffer.as<JsonObject>();
#else
  DynamicJsonBuffer jsonBuffer;
  JsonObject& root = jsonBuffer.parseObject(msg);
#endif
  if (root.containsKey("topic")) {
    if (String("logServer").equals(root["topic"].as<String>())) {
      // check for on: true or false
      logServerId = root["nodeId"];
      Serial.printf("logServer detected!!!\n");
    }
    Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());
  }
}
```

ภาคผนวก ค

โปรแกรมส่วนของ Server
สำหรับบอร์ด ESP32 ด้วยภาษา C

```
#include <HardwareSerial.h>
HardwareSerial MySerial(1);
//************************************************************
// this is a simple example that uses the painlessMesh library to
// setup a single node (this node) as a logging node
// The logClient example shows how to configure the other nodes
// to log to this server
//************************************************************
#include "painlessMesh.h"
#define   MESH_PREFIX     "PREM "
#define   MESH_PASSWORD   "0869634090"
#define   MESH_PORT       5555
Scheduler     userScheduler; // to control your personal task
painlessMesh  mesh;
// Prototype
void receivedCallback( uint32_t from, String &msg );
// Send my ID every 10 seconds to inform others
Task logServerTask(100, TASK_FOREVER, []() {
#if ARDUINOJSON_VERSION_MAJOR==6
      DynamicJsonDocument jsonBuffer;
      JsonObject msg = jsonBuffer.to<JsonObject>();
#else
      DynamicJsonBuffer jsonBuffer;
      JsonObject& msg = jsonBuffer.createObject();
#endif
   msg["topic"] = "logServer";
   msg["nodeId"] = mesh.getNodeId();

   String str;
#if ARDUINOJSON_VERSION_MAJOR==6
   serializeJson(msg, str);
#else
   msg.printTo(str);
#endif
   mesh.sendBroadcast(str);
```

```
    // log to serial
#if ARDUINOJSON_VERSION_MAJOR==6
    serializeJson(msg, Serial);
#else
    msg.printTo(Serial);
#endif
    Serial.printf("\n");
});

void setup() {
    Serial.begin(115200);
   MySerial.begin(9600,SERIAL_8N1, 4, 2);
   //mesh.setDebugMsgTypes( ERROR | MESH_STATUS | CONNECTION | SYNC
| COMMUNICATION | GENERAL | MSG_TYPES | REMOTE | DEBUG ); // all types on
   //mesh.setDebugMsgTypes( ERROR | CONNECTION | SYNC | S_TIME );  // set
before init() so that you can see startup messages
    mesh.setDebugMsgTypes( ERROR | CONNECTION | S_TIME );  // set before
init() so that you can see startup messages

   mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT,
WIFI_AP_STA, 6 );
   mesh.onReceive(&receivedCallback);

   mesh.onNewConnection([](size_t nodeId) {
     Serial.printf("New Connection %u\n", nodeId);
   });
  mesh.onDroppedConnection([](size_t nodeId) {
     Serial.printf("Dropped Connection %u\n", nodeId);
   });
   // Add the task to the your scheduler
   userScheduler.addTask(logServerTask);
   logServerTask.enable();
}
void loop() {
   userScheduler.execute(); // it will run mesh scheduler as well
   mesh.update();
```

```
}
void receivedCallback( uint32_t from, String &msg ) {
Serial.printf("logServer: Received from %u msg=%s\n", from, msg.c_str());
 MySerial.printf(msg.c_str()); // Print received data in Mesh Network to
Software Serial Port
}
```

ภาคผนวก ง

โปรแกรมส่วนของ Gateway สำหรับบอร์ด ESP32 ด้วยภาษา C

```
#include <HardwareSerial.h>
#include <WiFi.h>
#include <IOXhop_FirebaseESP32.h>
#include <time.h>
#include"soc/rtc.h"
#include <NTPClient.h>
#include <WiFiUdp.h>

/*
#################################################################
*/

//Firebase config here
#define FIREBASE_HOST "https://weightesp32.firebaseio.com/"
#define FIREBASE_AUTH "MFKWpxCkCBrnmUFCDViEaX6JbFk6jaLC1yQYJMtM"
#define WIFI_SSID "Joe"
#define WIFI_PASSWORD "0955435910"

/*
#################################################################
*/
HardwareSerial MySerial(1);
char value;
String mcu;
String b;
String namemcu;
String nameweight;

// Define NTP Client to get time
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);
// Variables to save date and time
String formattedDate;
String dayStamp;
String timeStamp;
```

```
void setup() {
  MySerial.begin(9600, SERIAL_8N1, 4, 2);
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("connecting");

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());

  timeClient.begin();
  timeClient.setTimeOffset(25200);

  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  // set string value
  Firebase.setString("Message", "Welcome to IoT training.");
  // handle error
  if (Firebase.failed()) {
    Serial.print("setting /message failed:");
    Serial.println(Firebase.error());
    return;
  }

}

void READ()
{
  int index = 0;
```

```
  while (MySerial.available()) {
    value = MySerial.read();
    Serial.print(value);
    //Serial.println(value);
    if (index == 0 && value != '{') {
      Serial.println("Cannot find the data header.");
      break;
    }

    if (index >= 6 && index <= 11 ) {
      mcu = mcu + value;
      namemcu = mcu ;
    }
    if (index >= 34 && index <= 40 && value != '\"' && value != '}' && value !=
'{'  ) {
      b = b + value;
      //Serial.println(b);
      nameweight = b ;

    } else if (index == 41 ) {
      mcu = "";
      b = "" ;
      break;
    }
    index++;
  }
  while (MySerial.available())
    MySerial.read();
}

void loop() {
  Serial.println(namemcu);
  Serial.println(nameweight);

  READ();
```

```
            while (!timeClient.update()) {
              timeClient.forceUpdate();
            }
            formattedDate = timeClient.getFormattedDate();
            int splitT = formattedDate.indexOf("T");
            dayStamp = formattedDate.substring(0, splitT);
            Serial.print("DATE: ");
            Serial.println(dayStamp);
            timeStamp = formattedDate.substring(splitT + 1, formattedDate.length() -
1);
            Serial.print("HOUR: ");
            Serial.println(timeStamp);
            Firebase.setString("Sensor/Weight/" + namemcu + "/" + dayStamp + "/" +
timeStamp, nameweight);
            if (Firebase.failed()) {
              Serial.println("set /Namemcu failed:");
              Serial.println(Firebase.error());
              return;
            }
            Serial.print("set /Namemcu to ");
            Serial.println(Firebase.getString("namemcu"));
            delay(1000);
          }
```