วงจรควบคุมควอดคอปเตอร์4ใบพัด และมิเตอร์วัดอุณหภูมิไร้สาย แสดงผลทางคอมพิวเตอร์

Quad copter control circuit and wireless thermometer with computer display

พัทรวุฒ ภูกองชัย

PHATTRAWUT PHUKONGCHAI

พัทธิต มาลีวิลาส

PATTANT MALEEWILAS

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

วงจรควบคุมคอปเตอร์4ใบพัด และมิเตอร์วัดอุณหภูมิไร้สาย แสดงผลทางคอมพิวเตอร์

Quad copter control circuit and wireless thermometer with computer display

โดย

ภัทราวุธ ภูกองชัย

ภัทริศ มาลีวิลาศ

อาจารย์ที่ปรึกษา

อ.โกศล ชวนขยัน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

ปริญญานิพนธ์ปีการศึกษา 2556

สาขา วิศวกรรมอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง วงจรควบคุมคอปเตอร์4ใบพัด และมิเตอร์วัดอุณหภูมิไร้สาย แสดงผลทางคอมพิวเตอร์

Quad copter control circuit and wireless thermometer with computer display

ผู้จัดทำ นายภัทราวุธ ภูกองชัย รหัสประจำตัว 53011220

นายภัทริศ มาลีวิลาศ รหัสประจำตัว 53011223

ปริญญานิพนธ์นี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

.....

(อ.เกศส ชวนขยัน)

อาจารย์ที่ปรึกษา

หัวข้อปริญญานิพนธ์ วงจรควบคุมคอปเตอร์4ใบพัด และมิเตอร์วัดอุณหภูมิไร้สายแสดงผลทางคอมพิวเตอร์

นักศึกษา นายภัทราวุธ ภูกองชัย รหัสประจำตัว 53011220

นายภัทริศ มาลีวิลาศ รหัสประจำตัว 53011223

ปริญญา วิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์

ปีการศึกษา 2556

อาจารย์ที่ปรึกษาปริญญานิพนธ์ อ.โกศล ชวนขยัน

บทคัดย่อ

รายงานฉบับนี้ อธิบายการออกแบบและการสร้างวงจรควบคุมคอปเตอร์4ใบพัด โดยจะควบคุมความเร็วในการหมุนของมอเตอร์ทั้ง4 เพื่อเพิ่มระดับความสูงของคอปเตอร์ และสามารถควบคุมให้เดินหน้า เลี้ยวซ้าย เลี้ยวขวา และถอยหลังได้ โดยในการสั่งงานจะควบคุมผ่านทางคอมพิวเตอร์ ซึ่งใช้โปรแกรม Visual C# ในการสร้างหน้าต่างควบคุม และจะใช้PIC18F4520 ในการเขียนโค้ดคำสั่งในโปรแกรม CCS Compiler และใช้Zigbee ในการรับ - ส่งข้อมูลระหว่างคอมพิวเตอร์กับตัวรับที่ตัวคอปเตอร์ ส่วนวงจรวัดอุณหภูมิไร้สาย ซึ่งจะใช้เซ็นเซอร์วัดอุณหภูมิSHT15 ติดไว้บนตัวคอปเตอร์ และส่งข้อมูลอุณหภูมิที่วัดผ่านทางZigbee กลับมาที่คอมพิวเตอร์เพื่อแสดงผล สามารถวัดได้ทั้งอุณหภูมิและความชื้น ซึ่งสามารถนำไปประยุกต์ใช้เป็นเครื่องวัดอุณหภูมิเคลื่อนที่ได้

Thesis Title	Quad copter control circuit and wireless thermometer with computer display
Student	Phattrawut Phukongchai Student ID 53011220
	Pattarit Maleewilas Student ID 53011223
Degree	Bachelor of Engineering
Program	Electronics Engineering
Year	2013
Thesis Advisor	Mr.Kosol Chaunkayan

Abstract

This report. Describes the design and building control circuit Quad Copter. It controls the rotational speed of the motor to increase the height of the copter. And can control forward , turn left , turn right and backward. The commands are controlled via computer by use Visual C# to create the control window. It uses a PIC18F4520 to writing the code and use Zigbee to receive and transmit data between computers and receiver at the copter. The wireless thermometer circuit which uses temperature sensors "SHT15" stick on the copter. Temperature measurement and transmission of data sent by Zigbee back to the computer to display. Can measure both temperature and humidity. Which can be used as a thermometer moving.

กิตติกรรมประกาศ

โครงงานนี้สำเร็จได้เนื่องจากได้รับคำแนะนำ ความช่วยเหลือจากอาจารย์โกศล ชวนขยัน (อาจารย์ที่ปรึกษา) และอาจารย์พลผดุง ผดุงกุล ที่ให้คำชี้แนะ และให้คำปรึกษาเกี่ยวกับโครงงานชิ้นนี้ในเรื่องต่างๆ ดังนั้นทางคณะผู้จัดทำขอขอบพระคุณท่านเป็นอย่างยิ่ง

ภัทราวุธ ภูกองชัย

ภัทริศ มาลีวิลาศ

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันมีการพัฒนาคอเตอร์ 4 ใบพัดกันอย่างแพร่หลาย ทั้งสามารถถ่ายวิดีโอมุมสูง ใช้สำรวจพื้นที่ หรือแม้กระทั่งในชีวิตประจำวันก็มีการนำคอปเตอร์ 4 ใบพัด มาใช้งาน เช่น ใช้ในการส่งพิซซ่า ส่งของต่างๆ

ผู้จัดทำจึงมีแนวคิดที่จะพัฒนาการควบคุมคอปเตอร์ 4 ใบพัด ที่สามารถวัดอุณหภูมิ และความชื้นที่อยู่ในพื้นที่รัศมีที่เราสามารถควบคุมได้ และสามารถเก็บค่าอุณหภูมิ ความชื้น ตลอดเวลาที่เราต้องการและยังสามารถควบคุมคอปเตอร์ใช้ไปวัดอุณหภูมิ ความชื้น ที่จุดใดก็ได้ที่เราต้องการ โดยในโครงงานได้นำเทคโนโลยีการส่งข้อมูลแบบไร้สาย โดยใช้ Zigbee มาประยุกต์ใช้งานร่วมกับเซ็นเซอร์วัดอุณหภูมิ ความชื้น และส่งข้อมูลที่ได้รับมาแสดงผลในหน้าต่างโปรแกรมทางคอมพิวเตอร์ และสามารถควบคุมทิศทางการบินของคอปเตอร์ ผ่านทางโปรแกรมทางคอมพิวเตอร์

ดังนั้นผู้จัดทำจึงได้จัดทำโครงงานการควบคุมคอปเตอร์ 4 ใบพัด ที่สามารถบอกอุณหภูมิความชื้นขึ้นเพื่อประโยชน์ตามที่กล่าวมาข้างต้น

1.2 วัตถุประสงค์

1) เพื่อศึกษาการทำงานของเทคโนโลยีไร้สาย ZigBee

2) เพื่ออำนวยความสะดวกในการตรวจสอบอุณหภูมิและความชื้นในสถานที่ต่างๆ

3) เพื่อศึกษาการทำงานของเซนเซอร์วัดอุณหภูมิและความชื้นกับไมโครคอลโทรลเลอร์

4) เพื่อศึกษาการเขียน Graphical User interface ด้วยภาษา C#

1.3 ขอบเขตของโครงงาน

ออกแบบและจำลองการควบคุมคอปเตอร์ 4 ใบพัด และเซ็นเซอร์วัดอุณหภูมิและความชื้นโดยให้มีการรับส่งข้อมูลกับส่วนจัดเก็บข้อมูลแบบไร้สายโดยใช้โมดูล ZigBee ในโครงงานประกอบด้วยสามส่วนหลัก คือ ส่วนที่หนึ่งคือส่วนของโปรแกรมควบคุมคอปเตอร์ 4 ใบพัด ส่วนที่สองคือส่วนของเซนเซอร์วัดอุณหภูมิและความชื้น และส่วนที่สามคือส่วนจัดเก็บข้อมูลและแสดงผล โดย

ทั้งสามส่วนนี้จะรับส่งข้อมูลผ่าน ZigBee และทำการสร้างเซนเซอร์วัดอุณหภูมิและความชื้นให้เชื่อมต่อกับโมดูล ZigBee โดยเขียนโปรแกรมควบคุมบนไมโครคอนโทรลเลอร์ และในส่วนจัดเก็บข้อมูลจะทำการเขียนโปรแกรมเพื่อจัดเก็บและแสดงผลผ่านหน้าจอคอมพิวเตอร์โดยใช้โปรแกรม Visual C#

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 เทคโนโลยีการสื่อสารไร้สาย (Broadband Wireless Access Technologies)

ในปัจจุบันมีการพัฒนาเทคโนโลยีหลายแบบสำหรับการเข้าใช้บรอดแบนด์ไร้สายโดยองค์กรมาตรฐานระดับนานาชาติ การพัฒนาเทคโนโลยีมีลักษณะเป็นการปรับปรุงเทคโนโลยีเดิมให้สามารถเข้าใช้บรอดแบนด์ไร้สายได้ หรือเป็นการพัฒนาเทคโนโลยีใหม่สำหรับบรอดแบนด์ไร้สาย เทคโนโลยีบรอดแบนด์นำมาใช้ตามลักษณะของโครงข่าย (Network) ดังนี้

2.1.1 โครงข่ายรอบตัวบุคคล (Personal Area Network: PAN)

เทคโนโลยี WPAN ถูกกำหนดให้อยู่ในมาตรฐาน IEEE 802.15 และมีมาตรฐานย่อย IEEE 802.15.1 ที่เป็นมาตรฐานเฉพาะของเทคโนโลยีบลูทูธ (Bluetooth) ซึ่งในขั้นต้นได้เสนอมาตรฐานนี้โดยร่วมกันของผู้ผลิตจากหลายบริษัท โดยการใช้งานในปัจจุบันนั้น การส่งข้อมูลด้วย Bluetooth สามารถใช้การส่งผ่านสาย (Cable) และมีระยะการส่งถึง 20 เมตร ต่อมามีการเสนอมาตรฐาน IEEE 802.15.3a เป็นมาตรฐานสำหรับระบบ Ultra Wide Band (UWB) ซึ่งเป็นระบบเครือข่ายที่มีอัตราการรับส่งข้อมูลใกล้ (ไม่เกิน 20 เมตร) ซึ่งมาตรฐาน IEEE 802.15.3a ยังไม่มีความสมบูรณ์และมีการพัฒนาอย่างต่อเนื่อง ส่วนมาตรฐาน IEEE 802.15.4 เป็นมาตรฐานระบบ ZigBee ซึ่งเป็นเทคโนโลยีที่มีความซับซ้อนน้อย ออกแบบมาเพื่อเป็นโครงข่ายไร้สายสำหรับอุปกรณ์ควบคุมในโรงงานอุตสาหกรรม

2.1.2 โครงข่ายท้องถิ่น (Local Area Network: LAN)

มาตรฐาน IEEE 802.11 WLAN หรือแลนไร้สาย เป็นที่รู้จักกันทั่วไปในนามเทคโนโลยี Wifi และเป็นที่นิยมกันมาก มาตรฐาน IEEE 802.11 ได้รับการตีพิมพ์ครั้งแรกในปี พ.ศ. 2540 ซึ่งอุปกรณ์ตามมาตรฐานดังกล่าวจะมีความสามารถในการส่งข้อมูลด้วยความเร็ว 1 และ 2 Mbps ด้วยสื่ออินฟราเรด (Infrared) หรือคลื่นวิทยุความถี่ 2.4 GHz และ 5 GHz เนื่องจากมาตรฐาน IEEE 802.11 เวอร์ชันแรกมีประสิทธิภาพค่อนข้างต่ำและไม่มีการรองรับ Quality of service (QoS) ซึ่งเป็นที่ต้องการของตลาด อีกทั้งกลไกการรักษาความปลอดภัยที่ใช้ยังมีช่องโหว่อยู่มาก IEEE จึงได้จัดตั้งคณะทำงาน (Task Group) ขึ้นมาหลายชุดด้วยกันเพื่อทำการปรับปรุงเพิ่มเติมมาตรฐานให้มีศักยภาพ

สูงขึ้น โดยคณะกรรมการกลุ่มที่มีผลงานน่าสนใจและเป็นที่รู้จักกันดีได้แก่ IEEE 802.11a, IEEE 802.11b, IEEE 802.11e, IEEE 802.11g, IEEE 802.11i

2.1.3 เครือข่ายไร้สายครอบคลุมพื้นที่ขนาดกว้าง (Metropolitan Area Network: LAN)

เครือข่ายแมนไร้สาย หรือ WMAN เป็นเครือข่ายไร้สายครอบคลุมพื้นที่ขนาดกว้าง เช่น เมือง ผู้ให้บริการโทรศัพท์ ผู้ให้บริการเครือข่าย T1 หรือ T3 เพื่อเชื่อมโยงไซต์ต่างๆ หรือ แคมปัสขนาดใหญ่ หน่วยงาน IEEE ออกมาตรฐานเทคโนโลยี WMAN ชุดใหม่ ซึ่งมาตรฐาน เทคโนโลยีที่รู้จักกันเป็นอย่างดี คือ IEEE 802.16d หรือ WiMAX ทำงานที่ช่วงความถี่ 2-11 กิโลเฮิรตซ์ (ในอเมริกาจะทำงานที่คลื่นความถี่ 2.5, 3.5 และ 5.8 GHz) ทั้งนี้ในทางทฤษฎีแล้ว WiMAX รองรับความเร็วสูงสุดที่ 70 Mbps ครอบคลุมพื้นที่มากกว่า 50 กิโลเมตร

2.1.4 โครงข่ายที่มีพื้นที่ครอบคลุมในการให้บริการที่กว้างขวาง (Wide Area Network: WAN)

มาตรฐาน IEEE 802.16 เป็นมาตรฐานเทคโนโลยีล่าสุดมาตรฐานหนึ่งที่กำหนดสำหรับ BWA (Broadband Wireless Access) ซึ่งถือว่าเป็นโครงข่ายที่มีพื้นที่ครอบคลุมในการให้บริการที่กว้างขวางพัฒนาขึ้นโดย IEEE ของสหรัฐอเมริกา โดยภาคอุตสาหกรรม ซึ่งประกอบด้วยผู้ผลิตอุปกรณ์ ผู้พัฒนา Chipset และผู้ประกอบการ ได้รวมตัวกันก่อตั้ง WiMAX Forum หรือ Worldwide Interoperability for Microwave Access Forum เป็นองค์กรที่ไม่แสวงหาผลกำไร มีเป้าหมายเพื่อนสนับสนุนและพัฒนาอุปกรณ์เครือข่ายให้มาตรฐานในกลุ่ม IEEE 802.16 เป็นไปในแนวทางเดียวกัน โดยมีวัตถุประสงค์เพื่อให้อุปกรณ์จากผู้ผลิตต่างๆสามารถใช้งานร่วมกันได้ เกิดความกลมกลืนของมาตรฐานและมีราคาเป็นที่ยอมรับของผู้บริโภค ดังนั้น IEEE 802.16 จึงมีชื่อเรียกอีกชื่อหนึ่งว่า WiMAX

BWA เป็นระบบเครือข่ายที่มีขอบเขตคลอบคลุมพื้นที่กว้างขวางมากกว่า WLAN หรือ Wifi โดยแท้จริงแล้วมาตรฐาน IEEE 802.16 BWA แบ่งได้ 2 แบบคือ มาตรฐาน IEEE 802.16-2004 ซึ่งกำหนดใช้ได้ในลักษณะการเข้าถึงแบบอยู่กับที่ (Fixed Wireless Access) เรียกว่า WMAN และมาตรฐาน IEEE 802.16e ซึ่งเป็นการปรับปรุงพัฒนามาจากมาตรฐาน IEEE.16-2004 โดยวัตถุประสงค์ของมาตรฐาน IEEE 802.16e เพื่อให้ใช้งานในขณะที่มีการเคลื่อนที่ และมีประสิทธิภาพการ Roaming ข้ามเซลล์ที่ดีกว่า

อีกหนึ่งมาตรฐานในส่วนของ BWA คือ มาตรฐาน IEEE 802.20 Mobile Broadband Wireless Access (MBWA) ซึ่งกำหนดให้ใช้งานบนพื้นฐานโครงข่าย IP (Internet Protocol) มีการเชื่อมต่อผ่านอากาศแบบแพ็คเก็ต (Packet) แต่อย่างไรก็ตามผู้กำหนดมาตรฐานมีความต้องการให้มาตรฐานนี้เข้าได้กับมาตรฐานอื่นๆที่มีอยู่ด้วยเป็นเทคโนโลยีที่กำหนดสำหรับอุปกรณ์เคลื่อนที่ความเร็วสูง (High Speed Mobile)

2.2 ZigBee

เทคโนโลยี ZigBee เป็นการสื่อสารที่ออกแบบขึ้นสำหรับการสื่อสารในเครือข่ายเซ็นเซอร์แบบไร้สาย (Wireless Sensor Network) โดยเริ่มจากการกำหนดมาตรฐานการรับ-ส่งข้อมูลแบบ IEEE 802.15.4 ที่เน้นการสื่อสารแบบประหยัดพลังงาน ความเร็วการรับส่งข้อมูลต่ำและมีราคาถูก การสื่อสารลักษณะนี้ได้ถูกนำมาใช้สำหรับการสื่อสารระหว่างเครื่องตรวจวัดหรือเซ็นเซอร์ ที่ต้องการสื่อสารแบบไร้สายเพื่อลดความยุ่งยากซับซ้อนสำหรับการติดตั้ง เช่น บริเวณโรงงานหนึ่งๆ อาจจะต้องใช้เซ็นเซอร์ปริมาณมาก และเครื่องรับส่งที่มีราคาถูก และประหยัดพลังงาน โดยการสื่อสารระยะใกล้แบบ ZigBee แตกต่างจากการสื่อสารแบบ บูลทูธ ดังนี้

- มีการเชื่อมต่ออย่างซับซ้อนเพื่อรองรับการเชื่อมต่อสำหรับเครือข่ายขนาดใหญ่

- การใช้งานแบบประหยัดพลังงานเพื่อการใช้งานได้ยาวนาน

- การสื่อสารระยะใกล้ในระยะ 10-1500 เมตร

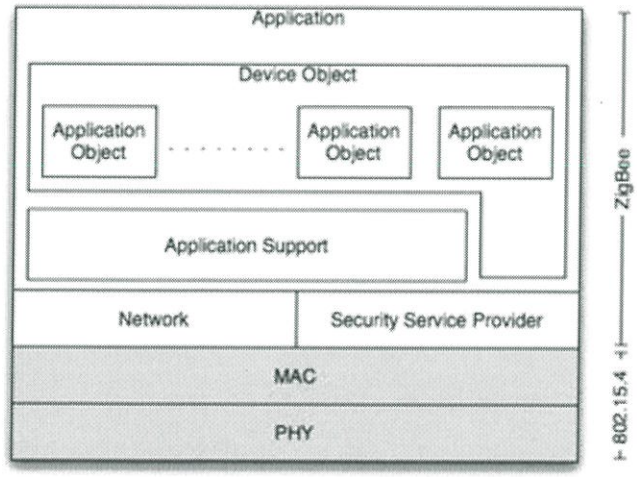
- เหมาะสำหรับการเฝ้าระวัง (Monitor) และการควบคุม (Control) ใช้งานอุตสาหกรรม งานสิ่งแวดล้อม งานก่อสร้างและงานทางการแพทย์

- เน้นการสื่อสารข้อมูลที่ความเร็วประมาณ 125-250 กิโลบิตต่อวินาที (Kbps)

มาตรฐานสากล ZigBee กำหนดโดย ZigBee Alliance เป็น การสื่อสารแบบไร้สาย ที่มีอัตราการรับส่งข้อมูลต่ำ ใช้พลังงานต่ำ ราคาถูก จุดประสงค์ก็เพื่อให้สามารถสร้างระบบที่เรียกว่า Wireless Sensor Network ได้ ซึ่งระบบนี้ จะสามารถทำงาน ในร่ม กลางแจ้ง ทนแดด ทนฝน และอยู่ได้ด้วยแบตเตอรี่ก้อนเล็ก (เช่นถ่าน AA 2 ก้อน) นานเป็นเดือน เป็นปี เหมาะสมใช้งานกับพวก onitoring ต่าง ๆ ZigBee เป็นอุปกรณ์ที่มี Microcontroller และ RF IC อยู่ภายใน ทำหน้าที่เป็นอุปกรณ์ transceiver (อุปกรณ์รับ-ส่งสัญญาณ) แบบ Half Duplex ย่านความที่ 2.4 GHz มีการจัดการโดยใช้พลังงานต่ำ ใช้งานง่าย มี Interface ที่ใช้รับและส่งข้อมูลกับ ZigBee เป็น UART ของ ZigBee ต่อเข้ากับ UART ของไมโครคอนโทรลเลอร์ได้ โดยมากมักสับสน ระหว่าง ZigBee กับ Wifi โดยผู้ที่เริ่มศึกษาจะมีคำถามว่า ZigBee กับ Wifi (หรือ โมดูล 2.4 GHz) นั้น ย่านความถี่เหมือนกัน

จะสามารถสื่อสารกันได้หรือไม่ คำตอบคือ สื่อสารกันไม่ได้ เพราะทางกายภาพ ถึงแม้จะเป็นย่านความถี่เดียวกัน แต่ Protocol ที่ใช้สื่อสารกันนั้นไม่เหมือนกัน

ZigBee สามารถใช้งานได้ตามมาตรฐาน ZigBee ได้โดยที่ไม่ต้องเขียนโปรแกรมสร้างเครือข่าย ZigBee เลย เพราะว่าทางผู้ผลิตได้จัดทำ Firmware ที่จะโหลดเข้าไปในตัว ZigBee ให้สามารถ Set parameter ผ่าน Software interface (X-CTUหรือโปรแกรมที่เขียนขึ้นเอง) ผ่านทาง AT command โดยใช้ Hyper terminal หรือผ่านทางการรับส่งข้อมูลด้วยไมโครคอนโทรลเลอร์ได้อย่างง่ายดาย โดยเมื่อ set ZigBee ให้ทำงานเป็นอุปกรณ์ในเครือข่ายแล้ว จะเรียก ZigBee แต่ละตัวว่าเป็น Node ZigBee นำ Physical Layer และ MAC Layer ของ IEEE 802.15.4 ซึ่งเป็นมาตรฐานการกำหนดการสื่อสารไร้สายแบบ WPAN (Wireless Personal Area Network) มาทำงานใน Layer ที่ต่ำกว่า (2 Layer ล่างสุด) เช่น เรื่องของ ระดับกำลังสัญญาณ , Link Quality , Access control , Security ฯลฯ แต่ใน Layer IEEE 802.15.4 เป็นมาตรฐานของระดับชั้น Physical (PHY) และ Media Access Control (MAC) สำหรับอุปกรณ์ส่งข้อมูลไร้สายที่ต้องการความเร็วในการส่งข้อมูลต่ำ ออกแบบมาเพื่อให้ใช้พลังงานน้อย สำหรับอุปกรณ์ที่ทีราคาไม่สูง การส่งข้อมูลที่มีความเชื่อถือได้เพราะมีความถูกต้องสูง และสามารถใช้ได้ทั้งในเครือข่ายแบบ star และ peer-to-peer มีระยะทางที่สามารถส่งข้อมูลได้ประมาณ 10-75 เมตร ขึ้นอยู่กับกำลังที่ใช้ในการส่งข้อมูลและสภาพแวดล้อมตอนที่ส่งและสามารถเลือกความถี่ที่ใช้ได้สามช่วงเพื่อความสะดวกในการนำไปใช้งาน มีการเข้ารหัสข้อมูลเพื่อความปลอดภัยของข้อมูลที่จะส่งโดยระบบ Advance Encryption System (AES)



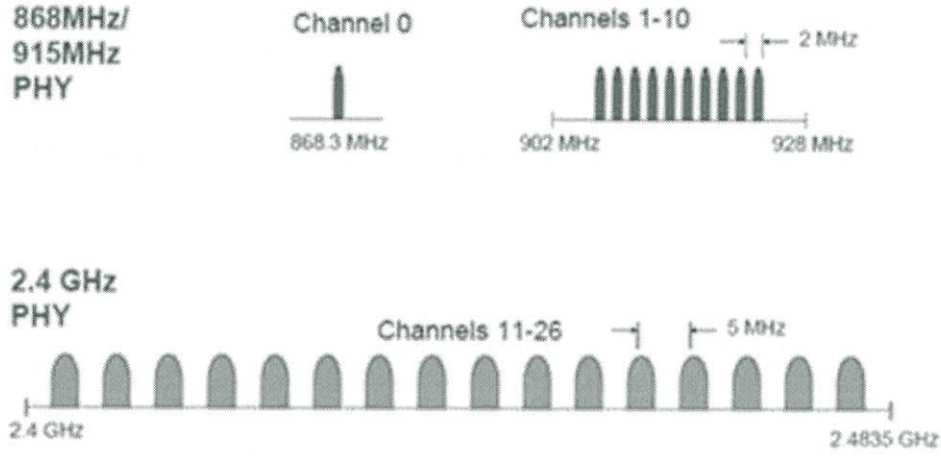
รูปที่ 2.1 ZigBee Stack

จากที่กล่าวมา ZigBee จะสามารถสร้างเป็นเครือข่ายได้เพราะอิงมาตรฐานตาม IEEE 802.15.4 และมีการจัดการในแบบของ ZigBee ใน Layer ถัดไป ทั้งนี้ IEEE 802.15.4 แบ่งชนิดอุปกรณ์ในเครือข่ายออกเป็น 2 ประเภท คือ FFD (Full Function Device) ซึ่งหมายถึงอุปกรณ์ที่

สามารถทำงานได้ทุกอย่างในเครือข่าย และ RFD (Reduce Function Device) ซึ่งหมายถึงอุปกรณ์ที่ถูกลดความสามารถการทำงานในเครือข่าย

Zigbee สามารถกำหนดย่านความถี่ใช้งานตามมาตรฐานไว้ 3 ย่านความถี่ คือ

- ย่านความถี่ 2.4 GHz มี 16 ช่องสัญญาณ อัตรารับส่งข้อมูล 250 Kbps
- ย่านความถี่ 915 GHz มี 10 ช่องสัญญาณ อัตรารับส่งข้อมูล 40 Kbps
- ย่านความถี่ 868 GHz มี 1 ช่องสัญญาณ อัตรารับส่งข้อมูล 20 Kbps



รูปที่ 2.2 แสดงย่านความถี่ของ ZigBee

2.2.1 ลักษณะการทำงานแบบเครือข่ายของ ZigBee

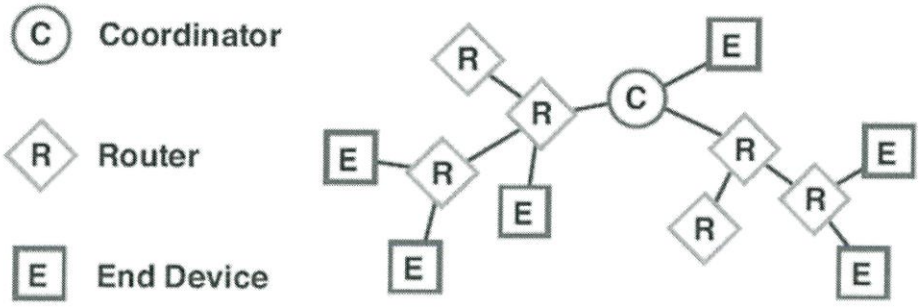
ZigBee ได้แบ่งตามลักษณะการทำงาน 3 แบบ คือ

2.2.1.1 Coordinator มีหน้าที่สร้างการสื่อสาร เชื่อมโยงเครือข่าย ระหว่าง End Device กับ Router หรือ Coordinator กับ Coordinator ด้วยกัน หรือ Coordinator กับ Router กำหนด address ให้กับ device ที่อยู่ในวงเครือข่าย ไม่ให้ซ้ำกัน ดูแลจัดการเรื่องการ Routing เส้นทาง ซึ่งเทียบได้กับ FFD

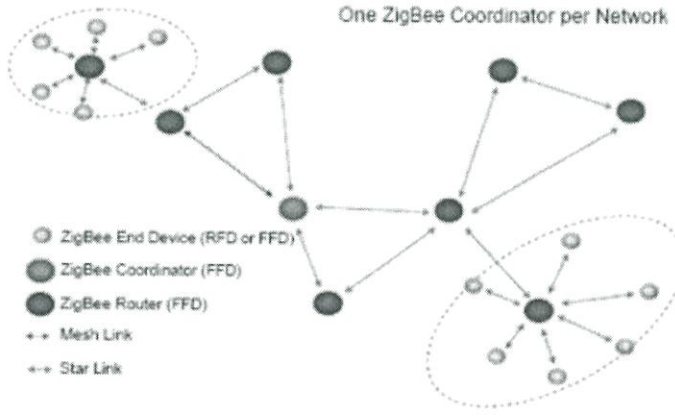
2.2.1.2 End Device เป็นอุปกรณ์ปลายทางสุด ซึ่งจะใช้รับสัญญาณจาก Sensor ที่ปลายทาง โดยที่ใช้พลังงานต่ำในการทำงาน เทียบได้กับ RFD หรือ FFD บางกรณี ขึ้นอยู่กับ sensor ที่ใช้

2.2.1.3 Router มีหน้าที่ รับส่งข้อมูล ในเส้นทางต่าง ๆ ของเครือข่าย ซึ่งเทียบได้กับ FFD

ในการสร้างโครงข่ายไร้สายของ ZigBee นั้นจะประกอบด้วยโหนดจำนวนอย่างน้อยที่สุด 2 ชนิด คือ Coordinator node และ node ลูกข่าย ชนิดใดชนิดหนึ่ง (Router/ End Device) จึงจะสามารถสื่อสารและทำงานในรูปแบบของ PAN (Personal area network) ดังรูปที่ 2.3 และ 2.4



รูปที่ 2.3 network topology



รูปที่ 2.4 network topology

2.2.2 วิธีการเลือกซื้อ Zigbee

2.2.2.1 การเลือก series

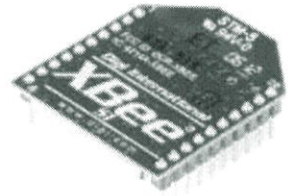
ข้อแตกต่างระหว่าง series1 และ series2 คือ series1 ทำ Mesh Network ไม่ได้ แต่ series2 ทำ Mesh Network ได้ และ series2 จะแบ่งเป็น 2 รุ่นย่อย คือ Znet2.5 และ ZB ซึ่ง ข้อแตกต่างระหว่าง series2 Znet2.5 และ series2 ZB คือ series2 ZB สามารถทำการอัพเดท Firmware ผ่านอากาศได้

2.2.2.2 การเลือกระยะการรับส่งสัญญาณ

ZigBee จะแยกรุ่นสำหรับเรื่องระยะการรับส่งอย่างชัดเจนด้วยคำว่า “PRO” โดยรุ่นระยะสั้น กำลังส่ง 1-2 mW จะมีระยะการรับส่งประมาณ 100-120 เมตร ส่วนรุ่นระยะไกล “PRO” กำลังส่งจะอยู่ในช่วง 50-60 mW โดยมีระยะประมาณ 1500 เมตร

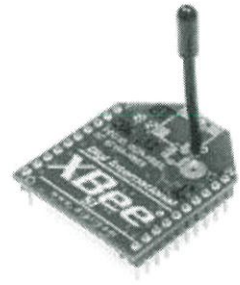
2.2.2.3 การเลือกแบบของสายอากาศ

ระยะที่ ZigBee รับส่งได้จริงตามที่ Spec บอกไว้ จะขึ้นกับสายอากาศ



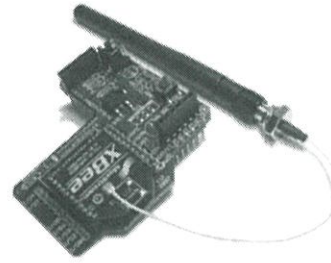
รูปที่ 2.5 Chip Antenna

1) Chip Antenna ดังรูปที่ 2.5 เหมาะกับการใช้งานใน โครงงานที่ต้องการขนาดเล็ก เพราะการใช้สายอากาศแบบนี้มีขนาดเล็ก นำไปใส่กล่องได้ แต่ได้เฉพาะกล่องพลาสติก ไม่สามารถใช้กล่องเหล็กได้ เนื่องจากใส่กล่องเหล็กสัญญาณจะไม่สามารถส่งออกมานอกกล่องเหล็กได้ หากต้องใช้กล่องเหล็ก ควรเลือกใช้สายอากาศที่ต่อออกมานอกกล่องเหล็ก



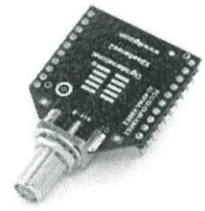
รูปที่ 2.6 Wire Antenna

2) Wire Antenna ดังรูปที่ 2.6 ระยะและความเสถียร จะได้ตาม Spec และด้วยสายอากาศที่ยื่นออกมาลักษณะนี้ บางทีผู้ใช้อาจจะรู้สึกไม่สะดวก ทำให้ใส่กล่องที่ออกแบบมาไม่ได้ แต่ถ้าใช้งานแบบทั่วไป แนะนำสายอากศแบบนี้



รูปที่ 2.7 UFL Antenna

3) UFL Antenna ดังรูปที่ 2.7 ระยะและความเสถียร ดังรูปที่ 2.7จะได้ตามประสิทธิภาพหมาะกับ งานที่ออกแบบใส่ในกล่อง และต้องการให้สายอากาศยื่นออกมานอกกล่อง และเนื่องจากการที่ต้องต่อสาย UFL to SMA ออกมาเพิ่มเติม ตรงจุดนี้จะทำให้เกิดการลดทอนสัญญาณ แต่ก็จะมีการขยายสัญญาณที่สายอากาศ จึงต้องไปพิจารณาอัตราขยายที่สายอากาศต่อ (อัตราขยาย เรียกว่า Gain มีหน่วยเป็น dB หรือ dBi)



รูปที่ 2.8 SMA Antenna

4) SMA Antenna ดังระยะและความเสถียร ดังรูปที่ 2.8 จะได้ตาม Spec ดีที่สุดต่อใช้งานร่วมกับสายอากาศ จะมีการขยายสัญญาณที่สายอากาศ ในการต่อใช้งานจริง การออกแบบใส่กล่อง จะต้องออกแบบให้มีตำแหน่งของ ZigBee ให้ใกล้กับรูเจาะกล่อง

ตารางที่ 2.1 ตารางเปรียบเทียบคุณสมบัติ

Feature	Series1	Series2	Series1 Pro	Series2 Pro
Power Input	3.3V @ 50mA	3.3V @ 40mA	3.3V @ 215mA	3.3V @ 295mA
Max data rate (Air)	250kbps	250kbps	250kbps	250kbps
Power Output	1mW output (+0dBm)	2mW output (+3dBm)	60mW output (+18dBm)	50mW output (+17dBm)
Distance	300ft (100m) range	400ft (120m) range	1 mile (1500m) range	1 mile (1600m) range
Antenna	Wire,Chip,UFL,SMA	Wire,Chip,UFL,SMA	Wire,Chip,UFL,SMA	Wire,Chip,UFL,SMA
Peripheral	6 10-bit ADC input pins 8 digital IO pins	6 10-bit ADC input pins 8 digital IO pins	6 10-bit ADC input pins 8 digital IO pins	6 10-bit ADC input pins 8 digital IO pins
Upgrade Firmware	Local	over-air configuration(ZB)	Local	over-air configuration(ZB)
Network	Point to point and multi-point networks	Point to point / multi-point / Mesh Network	Point to point and multi-point networks	Point to point / multi-point / Mesh Network

2.2.3 โหมดการทำงานของ ZigBee

ZigBee จะสามารถแบ่งช่วงการทำงานได้เป็น 5 แบบ คือ

2.2.3.1 โหมด 1 Idle mode เป็นโหมดที่ไม่ได้มีการรับส่งข้อมูล ตัว ZigBee เตรียมที่จะทำงานในโหมดอื่นๆต่อไปทันทีหากมีเงื่อนไขบางอย่าง

2.2.3.2 โหมด 2 และ 3 Transmit/Receive mode คือช่วงที่ ZigBee มีการรับหรือส่งข้อมูล โดยจะแบ่งลักษณะการทำงานย่อยออกเป็น Direct กับแบบ Indirect , การกำหนด Address ต้นทางปลายทาง , Clear Channel Assessment และการตอบรับ Acknowledgement

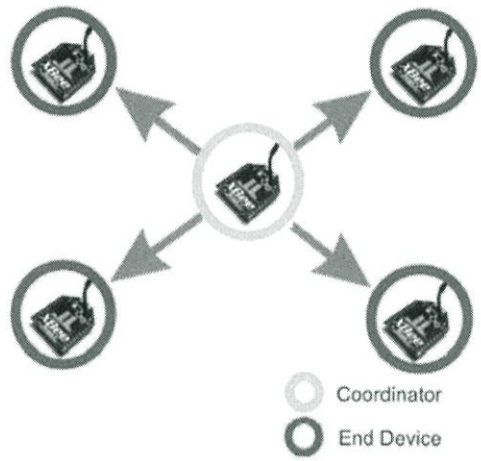
2.2.3.3 โหมด 4 Sleep mode คือช่วงที่ ZigBee อยู่ในสถานการณ์การทำงานพลังงานต่ำที่สุด เมื่อไม่มีการใช้งาน

2.2.3.4 โหมด 5 Command mode คือ เป็นส่วยการปรับ parameter ของ ZigBee ซึ่งจะมีการกำหนด 2 แบบ คือ AT command กับแบบ API command

2.2.4 การตั้งค่า ZigBee

2.2.4.1 Star (Broadcast)

การเชื่อมต่อแบบ Star (รูปที่ 2.9) หรือ แบบ Broadcast เป็นการรับส่งข้อมูลแบบไม่เฉพาะเจาะจง จุดหมายปลายทาง หรือ ZigBee ทุกตัวที่อยู่ในระบบเครือข่ายเดียวกันสามารถ รับข้อมูลทุกข้อมูลได้ทุกตัว การใช้งานแบบ Star จะประกอบไปด้วย ZigBee ที่ทำงานเป็น 2 รูปแบบคือ แบบที่1 เป็น Coordinator ทำหน้าที่ สร้างเครือข่าย และ แบบที่ 2 เป็น End Device ทำหน้าที่เป็นลูกข่าย



รูปที่ 2.9 การเชื่อมต่อแบบ Star

การตั้งค่าที่ตัว Coordinator

- ตั้ง PAN (Personal Area Network) สามารถตั้งได้ตามแต่ผู้ใช้จะกำหนด
- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

DH = 00, DL = FFFF

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น CO เป็นต้น)

การตั้งค่าที่ตัว End Device

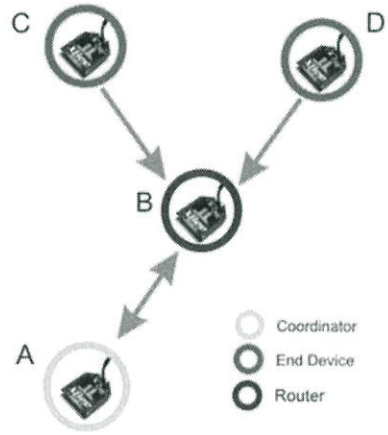
- ตั้ง PAN (Personal Area Network) จะต้องตั้งให้เหมือนกับ Coordinator
- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

DH = 00, DL = 00

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น EN1 เป็นต้น)

2.2.4.2 Cluster Tree

การเชื่อมต่อแบบ Cluster Tree (รูปที่ 2.10) เป็นการรับส่งข้อมูล แบบ ส่งผ่าน เช่น A ต้องการติดต่อ กับ C แต่ C อยู่ไกลจาก A จน A ไม่สามารถ ติดต่อกับ C ได้ แต่พอดีมี B ที่อยู่ระหว่าง A กับ C ดังนั้น Cluster Tree จะใช้ B เป็นเหมือน ตัวกลาง เชื่อมการติดต่อ (Repeater) ระหว่าง A กับ C มีการใช้งาน ZigBee อยู่ 3 ลักษณะด้วยกันคือ Coordinator, End Device, Routers



รูปที่ 2.10 การเชื่อมต่อแบบ Cluster Tree

การตั้งค่าที่ตัว Coordinator

- ตั้ง PAN (Personal Area Network) สามารถตั้งได้ตามแต่ผู้ใช้จะกำหนด
- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

DH = ค่า SH ของ End Device, DL = ค่า SL ของ Coordinator

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น CO เป็นต้น)

การตั้งค่าที่ตัว End Device

- ตั้ง PAN (Personal Area Network) จะต้องตั้งให้เหมือนกับ Coordinator

- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

  DH = ค่า SH ของ Coordinator, DL = ค่า SL ของ End Device

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น EN1,EN2 เป็นต้น)

การตั้งค่าที่ตัว Router

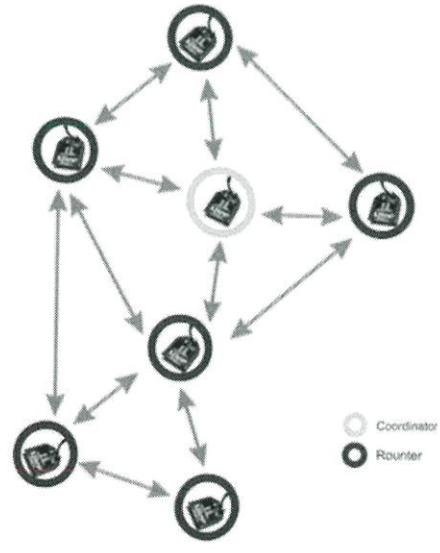
- ตั้ง PAN (Personal Area Network) จะต้องตั้งให้เหมือนกับ Coordinator
- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

DH = 00, DL = 00

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น RO1,RO2 เป็นต้น)

2.2.4.3 Mesh

การเชื่อมต่อเครือข่ายแบบ Mesh (แสดงในรูปที่ 2.11) เป็นโครงข่ายที่มีประสิทธิภาพสูงเนื่องจาก ข้อมูลสามารถส่งไปถึงเป้าหมายได้หลายทาง ทำให้ระบบนี้สามารถรับส่งข้อมูลไปยังจุดหมายปลายทางได้ แม้จะเกิดความเสียหายของระบบในบางส่วนก็ตาม (ขึ้นอยู่กับการออกแบบระบบของผู้ใช้ด้วย) ระบบนี้จึงเป็นระบบที่ได้รับความนิยมเป็นอย่างมาก มีการใช้งาน ZigBee อยู่ 2 ลักษณะด้วยกันคือ Coordinator, Routers



รูปที่ 2.11 การเชื่อต่อมแบบ Mesh

การตั้งค่าที่ตัว Coordinator

- ตั้ง PAN (Personal Area Network) สามารถตั้งได้ตามแต่ผู้ใช้จะกำหนด
- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

DH = 00, DL = FFFF

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น CO เป็นต้น)

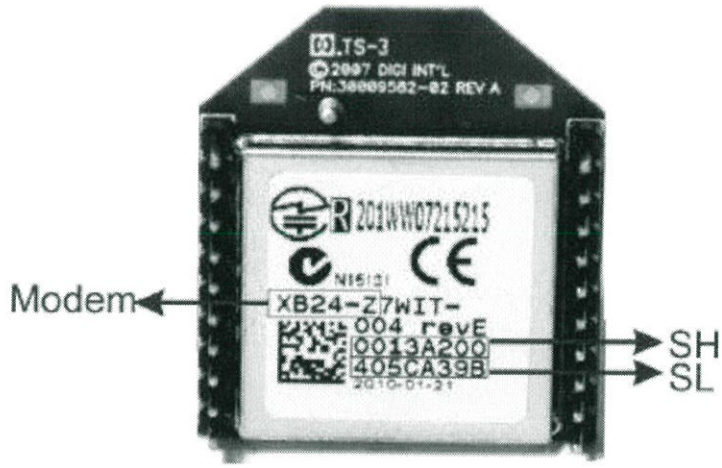
การตั้งค่าที่ตัว Router

- ตั้ง PAN (Personal Area Network) จะต้องตั้งให้เหมือนกับ Coordinator
- กำหนด Destination (จุดหมายที่ต้องการ รับส่งข้อมูลด้วย) โดย ตั้งค่า

DH = 00, DL = 00

- ตั้งชื่อโหนดโดย NI= (ชื่อโหนดที่ต้องการตั้ง เช่น RO1,RO2 เป็นต้น)

การหาค่า SH, SL, MODEM แสดงในรูป 2.12



รูปที่ 2.12 การหาค่า SH, SL, MODEM

MODEM ZigBee XB24-ZB

SH =13A200

SL = 405CA39B

2.2.4.4 การตั้งค่าเพิ่มความปลอดภัยให้กับระบบเครือข่ายและการตั้งค่าอื่นๆ

การตั้งค่าความปลอดภัยของรับบให้กำหนดที่ EE (Encryption Enable) = 1 Encryption Enable ตามด้วยดารตั้งรหัส KY (Link Key) = (ใส่ข้อมูลที่เราต้องการนำไปใช้เข้ารหัส เป็นเลขฐานสิบหก จำนวน 16 byte) โดยทำการตั้งค่านี้กับ ZigBee ในเครือข่ายทุกตัว

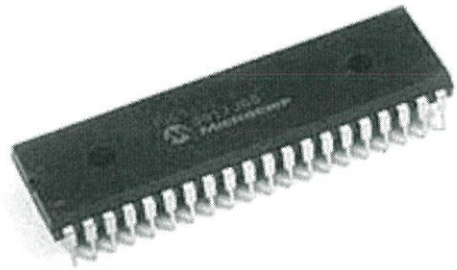
การตั้งค่า Sleep mode สามารถใช้ได้เฉพาะ End device ทำให้สามารถประหยัดพลังงานมากยิ่งขึ้นโดยค่า SM = 4 คือการ Sleep แบบ Cyclic sleep จะทำงานและหยุดทำงานสลับกันไปอยู่ตลอดเวลา และจะตื่นขึ้นมาทำงานเมื่อถูกกระตุ้นผ่านทาง UART หรือได้รับข้อมูลจาก Coordinator หรือโหนดอื่นๆที่ทำการส่งข้อมูลผ่าน สามารถกำหนดรายละเอียดต่างๆได้เพิ่มเติม เช่น ST= หากไม่มีการทำงานนานเท่าไรจึงจะเข้า sleep mode, SP=เวลาในการหลับการตื่นสลับกัน

2.3 ไมโครคอนโทรลเลอร์

PIC คือ ไมโครคอนโทรลเลอร์อีกตระกูลหนึ่งย่อมาจากคำว่า "Peripheral Interface Controller" ซึ่งแนวคิดของไมโครคอนโทรลเลอร์ตระกูลนี้ ก็คือ พยายามรวมเอาทุกอย่างเอาไว้ในตัวของมัน ไม่ว่าจะเป็น Program memory, Ram, EEPROM, Serial, I2C, PWM, A/D ฯลฯ โดยไม่จำเป็นต้องต่ออุปกรณ์เสริมจากภายนอกในตัวของ PIC จะมีฟังก์ชันที่ใช้ในการประมวลผล รวมทั้งหน่วยความจำทำให้มันเหมือนกับ ซีพียู ตัวหนึ่งเลยทีเดียว ความเร็วของไมโครคอนโทรลเลอร์ตระกูล PIC ภาคของความถี่สัญญาณนาฬิกา ปัจจุบันสามารถทำสัญญาณนาฬิกา ได้ที่ 20 MHz ซึ่งทำให้หนึ่งคำสั่งของ PIC ใช้เวลาเพียง 0.25 μS แต่อย่างไรก็ตาม มีบริษัทอื่นได้ซื้อลิขสิทธิ์ PIC จาก ไมโครชิพ และได้สร้างชิพที่มีความเร็วได้มากกว่าเดิมขึ้นไปอีก หน่วยความจำของไมโครคอนโทรลเลอร์ตระกูล PIC ในอดีตหน่วยความจำของ PIC จะมีค่อนข้างน้อย คืออยู่ระหว่าง 512 words ถึง 4K words แต่ในปัจจุบันบริษัท ไมโครชิพซึ่งเป็นเจ้าของ PIC ได้พัฒนาจนทำให้หน่วยความจำของ PIC มีขนาดเป็นหลายสิบกิโลไบต์ และมีแนวโน้มว่าจะขยายได้ใหญ่ขึ้นเรื่อย ๆ ในเรื่องของการนับขนาดของหน่วยความจำของ PIC จะนับไม่เหมือนปกติ โดยที่หนึ่งคำสั่งของ PIC จะมีขนาด 14 บิต ดังนั้นจะเรียกว่า 1 word ของ PIC จะมีขนาด 14 บิต เช่น PIC16F877A ระบุได้ว่ามีหน่วยความจำ 2 กิโลไบต์ ซึ่งหมายถึง 2 Kword ถ้าคำนวณให้เป็นในรูปแบบ 1 ไบต์ = 8 บิต จะได้ว่า 1 × 1,024 × 14 = 14,336 บิตดังนั้นก็คือ 14,336/(8 × 1,024) = 1.75 กิโลไบต์ นั่นเอง สถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูล PIC ตอนนี้มี 3 สายหลักๆ สมัยก่อนมีแค่ 2 คือ ขึ้นต้นด้วย 16xxx, 17xxx และใหม่ล่าสุดคือ 18xxx ถ้าพูดถึงคุณสมบัติที่เหนือกว่าเรียงจากน้อยสุดไปมากสุดก็คือ 16 -> 17 -> 18 คำสั่ง แอสแซมบลี ของ 17 และมี 18 จะมีมากกว่า 16 ทำให้เขียนโปรแกรมได้ง่ายกว่า ราคาก็จะสูงกว่าด้วย แต่ที่เป็นที่นิยมก็คือ ตระกูล 16xxx สรุปแนวคิดสถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูล*PIC**จะยึดถือการออกแบบที่ว่ารวมทุกอย่างไว้ในชิพตัวเดียว โดยไม่

ต้องต่ออุปกรณ์ใด ๆ เพิ่มเติมผลที่ตามมาก็คือแผ่นวงจรจะมีขนาดเล็กและอุปกรณ์ที่ใช้จะไม่มาก บางงานอาจจะใช้แค่ PIC เพียงตัวเดียวโดยไม่ต้องใช้ชิพอื่นมาเพิ่มเติมเลย นี่คือคุณสมบัติพิเศษของไมโครคอนโทรลเลอร์ตระกูล PIC ซึ่งปัจจุบันหลายบริษัทที่ผลิต ไมโครคอนโทรลเลอร์ก็เริ่มจะหันมาเลียนแบบแนวทางนี้ แต่ทุกอย่างย่อมมีข้อดีและข้อเสียเนื่องจากแนวคิดที่จะรวมทุกอย่างไว้ในชิพเดียวทำให้โปรแกรมหน่วยความจำและข้อมูลหน่วยความจำไม่สามารถขยายโดยใช้กับหน่วยความจำภายนอกได้ PIC จึงเหมาะสำหรับงานเล็ก ๆ ไม่ใช่งานใหญ่ ๆ ที่ต้องใช้การคำนวณและหน่วยความจำมาก ๆ

2.3.1 PIC รุ่น PIC18F4520



รูปที่ 2.13 ลักษณะภายนอกของ Microcontroller รุ่น PIC18F4520

2.3.1.1 คุณสมบัติต่างๆของ PIC18F4520

1) ซีพียูเป็นแบบ RISC (Reduced Instruction-Set Computer)

2) มีคำสั่งใช้งาน 35 คำสั่ง

3) ทำตามคำสั่งโดยใช้สัญญาณ 1 ลูก

4) ทำงานด้วยความถี่สัญญาณนาฬิกาตั้งแต่ไฟตรงจนถึง 40 MHz

5) หน่วยความจำโปรแกรม 64 KB

6) หน่วยความจำข้อมูลแรมหรือรีจิสเตอร์ 3968 Bytes

7) หน่วยความจำข้อมูลอีอีพรอม 1024 Bytes

8) มีการตอบสนองสัญญาณอินเตอร์รัปต์

9) มีวงจรเพาเวอร์ออนReset (POR)

10) มีเพาเวอร์อัปไทเมอร์ (PWRT)

11) ออสซิลเลเตอร์สตาร์ตอัปไทเมอร์ (OST)

12) วงจรวอตซ์ด็อกไทเมอร์ (WDT) วงจรออสซิลเลเตอร์ในตัว

13) เลือกป้องกันข้อมูลทั้งในหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลได้

14) สามารถโปรแกรมแรงดัน +5 V

15) แก้ไขตัวโปรแกรมในหน่วยความจำผ่านพอร์ทเพียง 2 ขา ด้วยกระบวนการ ICD (In-Circuit Debugger)

16) ซีพียูสามารถอ่านและเขียนหน่วยความจำโปรแกรมได้

17) ใช้ไฟเลี้ยงที่ใช้งานตั้งแต่ +2 V ถึง +5.5 V

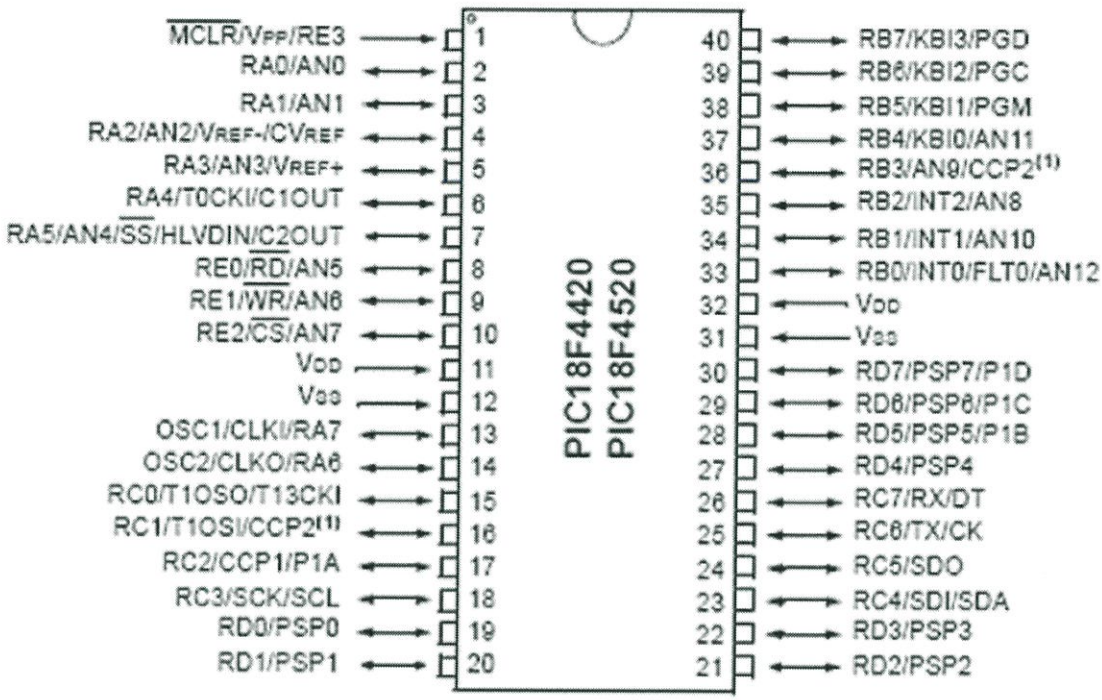
2.3.1.2 คุณสมบัติพิเศษ

1) วงจรแปลงสัญญาณ analog เป็น digital 10 bit สูงสุด 13 ช่อง

2) วงจรเชื่อมต่ออุปกรณ์อนุกรมทั้ง SPI และ I^2C

3) วงจรสื่อสารข้อมูลอนุกรม (USART) พร้อมการตรวจจับแอดเดรส

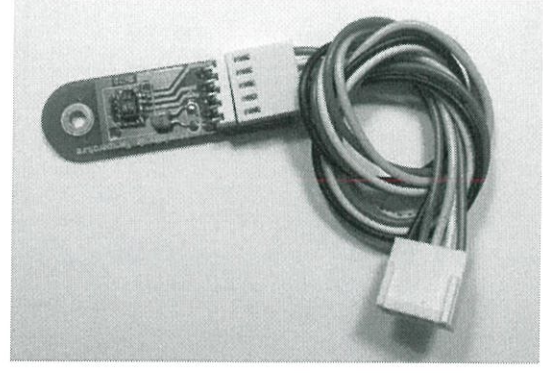
4) วงจรตรวจจับระดับแรงดันไฟเลี้ยง (Brown-out detection) เพื่อResetซีพียู หรือ BOR (Brown-out Reset)



รูปที่ 2.14 ตำแหน่งขาของ Microcontroller รุ่น PIC18F4520

2.4 เซ็นเซอร์วัดอุณหภูมิและความชื้น SHT15

เป็นโมดูลวัดอุณหภูมิและความชื้นสัมพัทธ์มีขนาดเล็กและเพื่อความสะดวกในการใช้งานจึงได้ติดตั้งลงบนแผ่นวงจรพิมพ์และต่อคอนเน็คเตอร์ 5 ขา เพื่อให้สามารถติดตั้งบนแผงวงจรเพื่อทำการทดลองได้ง่าย รวมไปถึงการนำไปประยุกต์ใช้งานจริงด้วย



รูปที่ 2.15 SHT15

2.4.1 คุณสมบัติเบื้องต้น

2.4.1.1 สามารถวัดความชื้นสัมพัทธ์ และอุณหภูมิได้ในเซ็นเซอร์เดียวกัน

2.4.1.2 วัดอุณหภูมิเป็นแบบดิจิตอล

2.4.1.3 สามารถกำหนดความละเอียดของย่านการวัด

2.4.1.4 มีขนาดเล็กและใช้พลังงานต่ำ ทำงานในย่านแรงดันไฟเลี้ยง +2.4V ถึง +5.5V

2.4.1.5 เสถียรภาพในการทำงานสูง

2.4.2 การจัดขาของเซ็นเซอร์

ตารางที่ 2.2 แสดงการจัดขาของเซ็นเซอร์

ขาที่	ชื่อ	รายละเอียด
1	GND	ขากราวน์
2	DATA	ขาข้อมูลแบบ Serial โดยขานี้จะต้องต่อกับตัวต้านทาน(Resistor) pull-up 4.7k-10k
3	SCK	สัญญาณนาฬิกาสำหรับ synchronize กันระหว่าง microcontroller กับ sensor โดยจะอ่านข้อมูลที่ขอบขาขึ้นของสัญญาณ
4	VDD	ขารับไฟเข้า +2.4 – +5.5V
NC	NC	ไม่ได้ใช้งาน

2.4.3 รูปแบบการสื่อสารข้อมูล SHT15

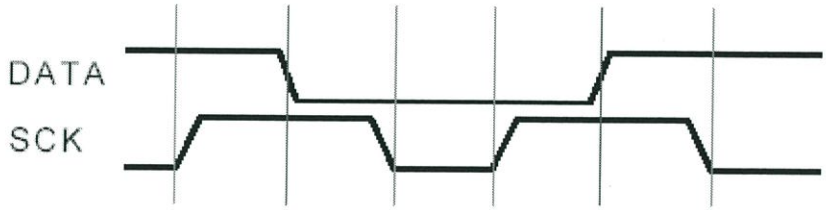
2.4.3.1 การส่งคำสั่ง

ในสภาวะเริ่มต้นก่อนการส่งข้อมูลคำสั่งจากไมโครคอนโทรลเลอร์ไปยัง SHT15 ต้องสร้างรูปแบบสัญญาณกระตุ้นผ่านขา SCKและ DATA เพื่อให้ตรงกับเงื่อนไขที่เรียกว่า Transmission start หรือภาวะเริ่มส่งสัญญาณ นั่นคือขา DATA ต้องถูกทำให้เปลี่ยนเป็นลอจิก “0” นาน 1 ของ

สัญญาณนาฬิกา SCK ดังรูปที่ และ หลังจากนี้ SHT15 จะทราบได้ทันทีว่า ข้อมูลต่อจากนี้คือคำสั่ง แสดงในรูป 2.13

- data เปลี่ยนจาก 1 เป็น 0 ขณะที่ SCK ลูกแรกเป็น 1

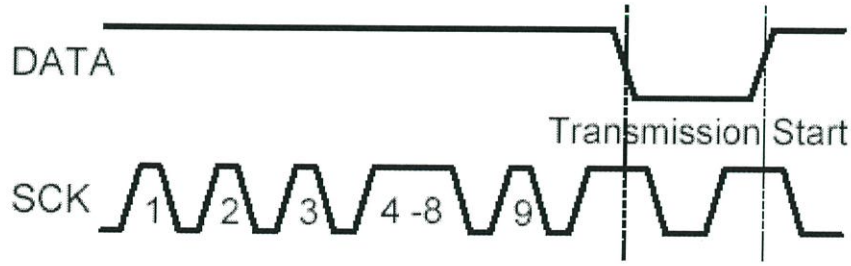
- data เปลี่ยนจาก 0 เป็น 1 ขณะที่ SCH ลูกที่สองเป็น 1



รูปที่ 2.16 Transmission start

2.4.3.2 Command reset sequence

เมื่อต้องมีการเริ่มต้นการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับโมดูล SHT15 ต้องสร้างสัญญาณรีเซ็ตขึ้นก่อน โดยทำให้ขา DATA มีสถานะลอจิก “1” นานเท่ากับช่วงเวลาที่ป้อนสัญญาณนาฬิกาที่ขา SCK 9 ลูกติดต่อกัน แล้วตามด้วยการสร้างภาวะเริ่มต้นการส่งสัญญาณ ดังแสดงในรูป 2.14



รูปที่ 2.17 Command reset sequence

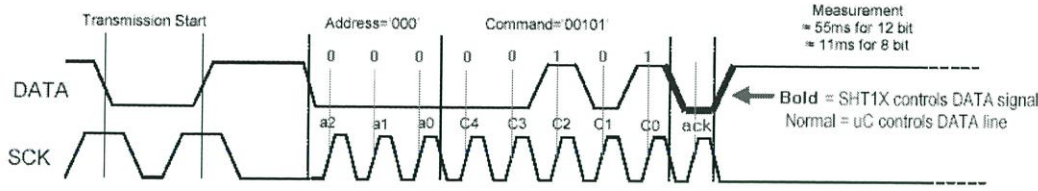
ตารางที่ 2.3 คำสั่งในการทำงานและการอ่านข้อมูลจาก sensor

Command	Code	Description
Reserved	0000x	Reserved
Measure Temperature	00101	Temperature measurement
Measure Humidity	00111	Humidity measurement
Status Register Read	00111	Read access to the status register (see application note)
Status Register Write	00110	Write access to the status register (see application note)
Reserved	0101x-1110x	Reserved
Soft reset	11110	Resets the chip, clear the status register to default values
		Wait 11ms before next command

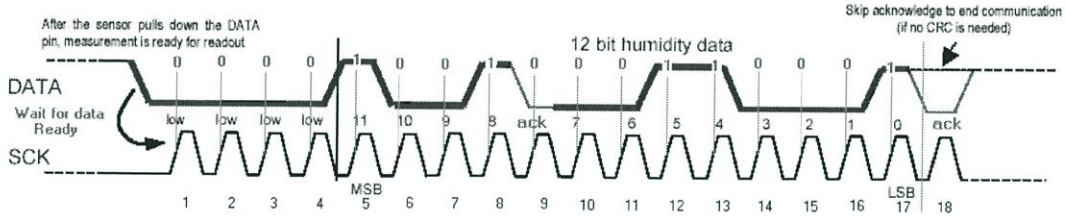
2.4.3.3 Measurement sequence (T and RH)

การอ่านข้อมูลดิบของอุณหภูมิหรือความชื้นสัมพัทน์นั้นทำได้หลังจากสร้างสภาวะเริ่มต้นทีเรียกว่า transmission start แล้วตามด้วยการส่งข้อมูลคำสั่งการอ่านอุณหภูมิหรือความชื้นสัมพัทธ์อย่างใดอย่างหนึ่ง(อุณหภูมิ “0000001” ความชื้น “00000101”)ไปยัง SHT15 โดยเซ็นเซอร์ SHT15 ต้องใช้เวลาในการประมวลผลเพื่อให้ได้ผลลัพธ์ที่ต้องการซึ่งจะใช้เวลา 210/55/11 มิลลิวินาทีสำหรับ 14/12/8 บิท ตามลำดับ แสดงไดอะแกรมเวลาของการอ่านข้อมูลจากโมดูล SHT15 โดยข้อมูลที่ส่งออกมาจากโมดูล SHT15 ประมวลด้วยข้อมูล 2 ไบต์ และสำหรับการตรวจสอบข้อผิดพลาดอีก 1 ไบต์ หรือ CRC checksum โดยไมโครคอนโทรลเลอร์เมื่อรับข้อมูล 1 ไบต์จะต้องส่งสัญญาณรับรู้หรือ Acknowledge ออกมา 1 ลูก (กำหนดให้ขา DATA มีลอจิก“0” บิตนัยสำคัญสูงสุดของข้อมูลจะถูกส่งออกมาก่อนกรณีอ่านค่าแบบ 8 บิต ไบต์แรกจะไม่ถูกใช้งาน การยกเลิกการสื่อสารเมื่อไมโครคอนโทรลเลอร์ส่งสัญญาณรับรู้ หลังจากได้รับข้อมูลบิตสุดท้ายของ CRC แล้ว สำหรับกรณีที่ไม่ต้องการตรวจสอบ CRC การยกเลิกการเชื่อมต่อทำได้โดยการไม่ส่ง

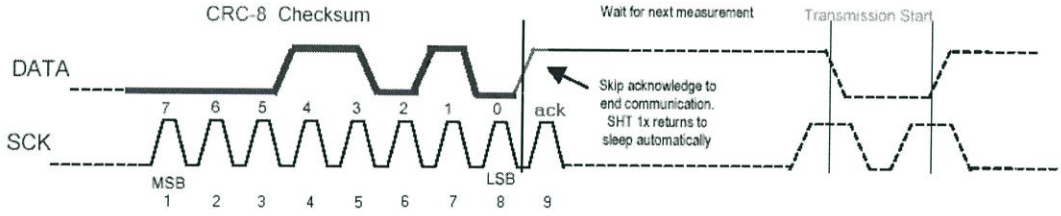
Acknowledge หลังจากรับข้อมูลในไบต์ที่สองแล้ว หลังจากนั้นเพื่อเป็นการประหยัดพลังงาน SHT15 จะเข้าสู่โหมด SLEEP โดยอัตโนมัติ ดังรูป 2.16 และ 2.17



รูปที่ 2.18 ลักษณะสัญญาณในการอ่านข้อมูลจาก Sensor



รูปที่ 2.19 ลักษณะสัญญาณในการอ่านข้อมูลT/H 2 byte จาก Sensor



รูปที่ 2.20 ลักษณะสัญญาณในการอ่านข้อมูล CRC8 1byte จาก Sensor

2.4.3.4 รีจีสเตอร์แสดงสถานะ

สำหรับฟังก์ชันที่ต้องมีการปรับแต่งพิเศษ จะต้องมีการกำหนดรีจิสเตอร์ STATUS โดยรายละเอียดบิตต่างๆของรีจิสเตอร์ STATUS แสดงดังตาราง 2.4

ตารางที่ 2.4 แสดงหน้าที่ของบิตต่างๆในรีจิสเตอร์ STATUS

บิต	การติดต่อ	รายละเอียด	ค่าเริ่มต้น
7	อ่านเท่านั้น	สงวนไว้ไม่ใช้งาน	0
6	อ่านเท่านั้น	ตรวจสอบไฟเลี้ยง "0" ถ้าแรงดันไฟเลี้ยงมากกว่า 2.47V "1" ถ้าแรงดันไฟเลี้ยงน้อยกว่า 2.47V	ไม่มีค่าเริ่มต้น ขึ้นอยู่กับค่าที่ได้จากการตรวจวัด
5	อ่านเท่านั้น	สงวนไว้ไม่ใช้งาน	0
4	อ่านเท่านั้น	สงวนไว้ไม่ใช้งาน	0
3	อ่านเท่านั้น	ไม่ใช้งานบิตนี้ เนื่องจากเป็นบิตตรวจสอบชิปของผู้ผลิต	0
2	อ่าน/เขียน	เปิดปิดตัวทำความร้อน "0" ปิดตัวทำความร้อนภายในเซ็นเซอร์ "1" เปิดตัวทำความร้อนภายในเซ็นเซอร์	0
1	อ่าน/เขียน	การอ่านค่าปรับแต่งความแม่นยำจากหน่วยความจำ	0
0	อ่าน/เขียน	"0" วัดค่าความชื้น 12 บิต , วัดอุณหภูมิ 14 บิต "1" วัดค่าความชื้น 8 บิต , วัดอุณหภูมิ 12 บิต	0

2.4.3.5 การคำนวณค่าอุณหภูมิ

ในการอ่านค่าอุณภูมิจากโมดูล SHT15 สามารถเลือกความละเอียดในการได้อ่านได้ในแบบ 14 บิต หรือ 12 บิต โดยที่ความละเอียด 14 บิต เป็นค่าตั้งต้น โดยการคำนวณค่าข้อมูลจะต้องอ่านข้อมูลดิบจากโมดูล SHT15 เข้ามาก่อน จากนั้นจึงใช้กระบวนการทางคณิตศาสตร์เพื่อให้ได้ค่าอุณหภูมิออกมา โดยคำนวณได้จากสมการ

$$T = d_1 + d_2 \cdot SO_T \qquad (2.1)$$

เมื่อ T คือค่าอุณหภูมิจริง

SO_T คือ serial output Temperature

d_1 คือ ค่าคงที่ที่ขึ้นอยู่กับไฟเลี้ยงที่ป้อนให้กับขา V_{dd} ของ SHT 15

d_2 คือ ค่าคงที่ที่ขึ้นอยู่กับความละเอียดของอุณหภูมิที่ต้องการจาก SHT15

ตารางที่ 2.5 ค่าคงที่ d_1และ d_2

SO_T	$d_2°C$	$d_2°C$
14bit	0.01	0.018
12bit	0.04	0.072

V_{dd}	$d_1°C$	$d_1°C$
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

2.4.3.6 การคำนวณค่าความชื้นสัมพัทธ์

ในการอ่านค่าความชื้นสัมพัทธ์จากเซ็นเซอร์ SHT15 สามารถเลือกความละเอียดในการได้อ่านได้ในแบบ 12 บิต หรือ 8 บิต โดยที่ความละเอียด 12 บิต เป็นค่าตั้งต้น โดยการคำนวณค่าข้อมูลจะต้องอ่านข้อมูลดิบจากโมดูล SHT15 เข้ามาก่อน จากนั้นจึงใช้กระบวนการทางคณิตศาสตร์เพื่อให้ได้ค่าอุณหภูมิออกมา โดยคำนวณได้จากสมการ

$$RH_{true} = (T - 25) \cdot [t_1 + (t_2 \cdot SO_{RH})] + RH_{linear} \quad (2.2)$$

$$RH_{linear} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot SO_{RH}^2 \quad (2.3)$$

เมื่อ RH_{true} คือค่าความชื้นสัมพัทธ์จริง

T คือค่าอุณหภูมิที่คำนวณได้จากสมการ 1

SO_{RH} คือ serial output humidity แบบ linear

t_1, t_2 คือค่าคงที่ที่ขึ้นอยู่กับการชดเชยของอุณหภูมิ

c_1, c_2, c_3 คือค่าคงที่ที่ขึ้นอยู่กับความละเอียดของความชื้นสัมพัทธ์ที่ต้องการจาก SHT15

ตารางที่ 2.6 ค่าคงที่ t_1, t_2 และ c_1, c_2, c_3

SO_{RH}	t_1	t_2
12 bit	0.01	0.00008
8 bit	0.01	0.00128

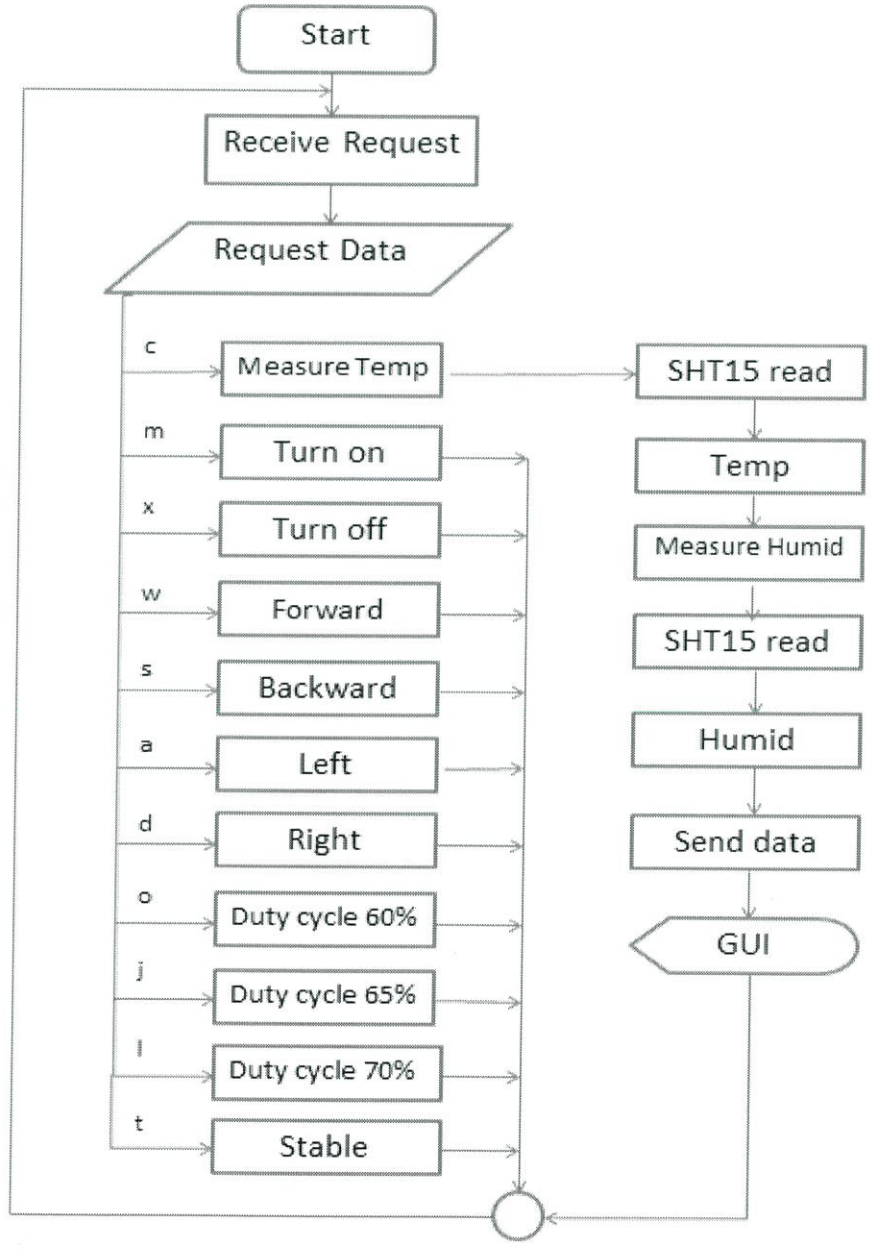
SO_{RH}	c_1	c_2	c_3
12 bit	-4	0.0405	-2.8×10^{-6}
8 bit	-4	0.648	-7.2×10^{-4}

บทที่ 3

การออกแบบและการจัดทำปริญญานิพนธ์

วงจรควบคุมคอปเตอร์ 4 ใบพัด และเซ็นเซอร์วัดอุณหภูมิและความชื้นไร้สายประกอบด้วย เซ็นเซอร์ภาคส่ง ไมโครคอนโทรลเลอร์ และ ZigBee จะรับค่าและส่งไปแสดงผลที่ส่วนแสดงผลซึ่งเชื่อมต่ออยู่กับคอมพิวเตอร์โดยใช้โปรแกรม Visual C#

3.1 Flow Chart



รูปที่ 3.1 Flow Chart

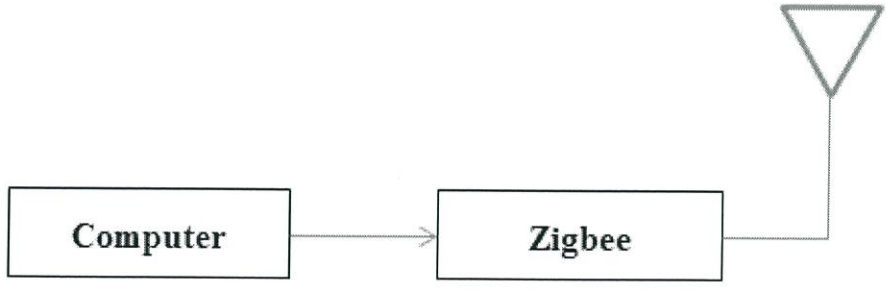
Flow Chart แสดงการทำงานของไมโครคอนโทรเลอร์โดย เมื่อได้รับข้อมูลคำสั่งแล้ว จะทำงานตามโปรแกรมคำสั่งดังนี้

- เมื่อได้รับข้อมูล “ w ” ให้ควบคุมให้คอปเตอร์เคลื่อนที่ไปข้างหน้า
- เมื่อได้รับข้อมูล “ s ” ให้ควบคุมให้คอปเตอร์เคลื่อนที่ไปข้างหลัง
- เมื่อได้รับข้อมูล “ a ” ให้ควบคุมให้คอปเตอร์เคลื่อนที่ไปทางซ้าย
- เมื่อได้รับข้อมูล “ d ” ให้ควบคุมให้คอปเตอร์เคลื่อนที่ไปทางขวา
- เมื่อได้รับข้อมูล “ o ” ให้ควบคุมให้ความแรงของมอเตอร์ที่คอปเตอร์ให้มี Dutycycle=60%
- เมื่อได้รับข้อมูล “ j ” ให้ควบคุมให้ความแรงของมอเตอร์ที่คอปเตอร์ให้มี Dutycycle=65%
- เมื่อได้รับข้อมูล “ l ” ให้ควบคุมให้ความแรงของมอเตอร์ที่คอปเตอร์ให้มี Dutycycle=70%
- เมื่อได้รับข้อมูล “ t ” ให้ควบคุมให้คอปเตอร์บินหยุดนิ่งอยู่กับที่
- เมื่อได้รับข้อมูล “ x ” ให้ควบคุมให้คอปเตอร์หยุดการทำงาน
- เมื่อได้รับข้อมูล “ m ” ให้ควบคุมให้คอปเตอร์เริ่มการทำงาน
- เมื่อได้รับข้อมูล “ c ” ให้สั่งงานให้เซ็นเซอร์ทำงานวัดอุณหภูมิและความชื้น

3.2 การออกแบบ

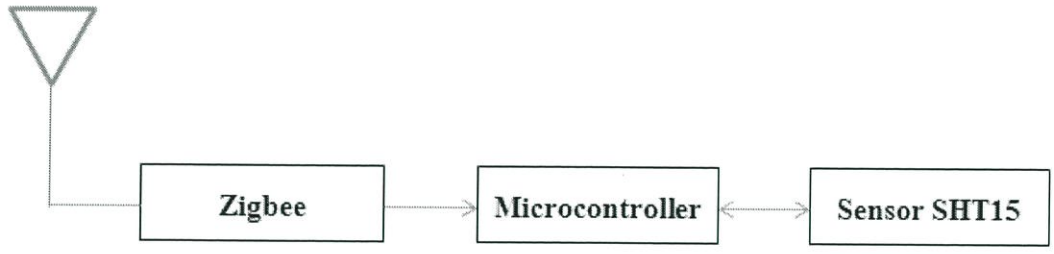
3.2.1 บล็อกไดอะแกรม

การอธิบายบล็อกไดอะแกรมจะแบ่งการอธิบายออกเป็น 2 ส่วนคือ เป็นส่วนของภาคส่ง และภาครับ ซึ่งสามารถอธิบายได้ดังนี้



รูปที่ 3.2 บล็อกไดอะแกรมของภาคส่ง

จากบล็อกไดอะแกรม ที่คอมพิวเตอร์จะใช้โปรแกรม Visual C# ในการสร้างหน้าต่าง Graphical User interface เพื่อเป็นส่วนแสดงผลค่าอุณหภูมิและความชื้นที่วัดได้จากเซ็นเซอร์ และเป็นส่วนที่ใช้สร้างคำสั่งควบคุมการทำงานของคอปเตอร์ 4 ใบพัด จากนั้นส่งสัญญาณคำสั่งที่สร้างขึ้นเข้ามาที่โมดูล Zigbee เพื่อใช้รับ-ส่งข้อมูลติดต่อกับส่วนของภาครับต่อไป



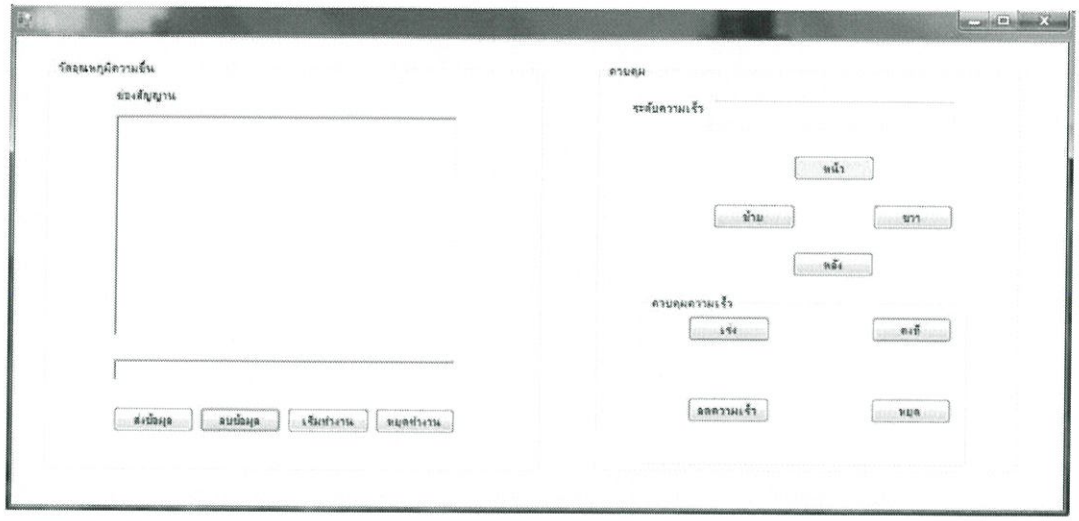
รูปที่ 3.3 บล็อกไดอะแกรมของภาครับ

จากบล็อกไดอะแกรมภาครับ เมื่อ Zigbee รับข้อมูลจากภาคส่งมาแล้ว จะมาส่งเข้ามาที่ Microcontroller โดยใช้ฟังก์ชัน UART ภายในตัว Microcontroller เพื่อใช้ในการติดต่อ รับ-ส่งข้อมูล

เมื่อสั่งให้เซ็นเซอร์ทำงานแล้ว เซ็นเซอร์จะวัดค่าอุณหภูมิ และความชื้น และส่งข้อมูลที่ได้กลับเข้ามาที่ Microcontroller เพื่อคำนวณค่า และส่งข้อมูลออกมาเข้า Zigbee เพื่อส่งมาแสดงผลที่จอคอมพิวเตอร์

3.2.2 การออกแบบวงจรภาคส่ง

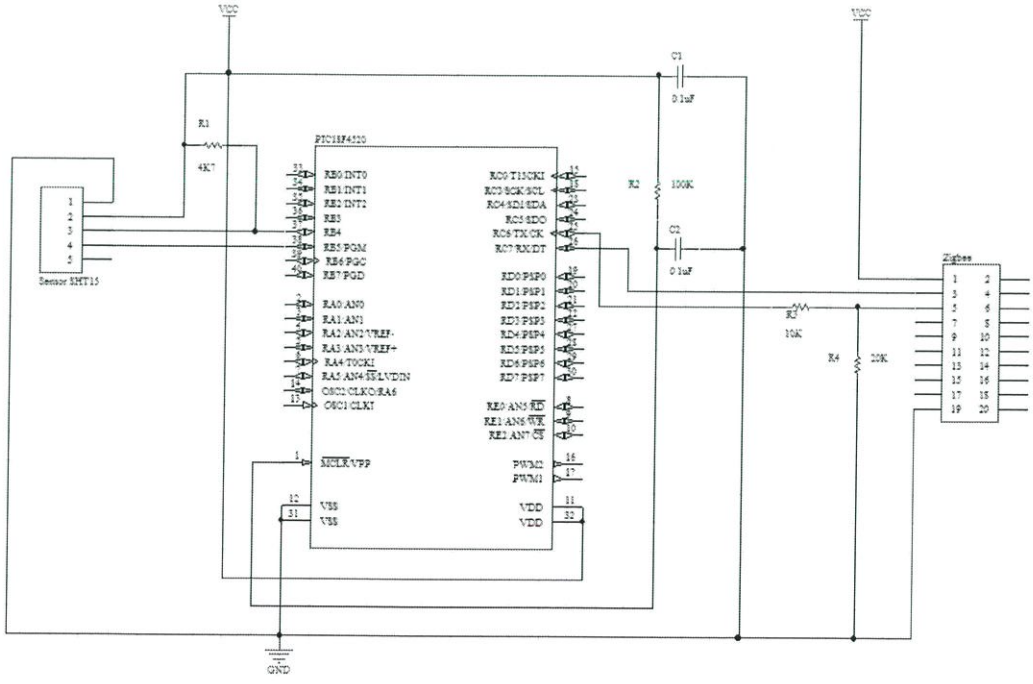
ในส่วนของวงจรภาคส่ง จะใช้ Zigbee ต่อ USB Dongle เข้ากับเครื่องคอมพิวเตอร์ และเขียน Code ในโปรแกรม Visual C# สร้างเป็นหน้าต่างควบคุมการทำงานของคอปเตอร์ และเซ็นเซอร์วัดอุณหภูมิ ความชื้น



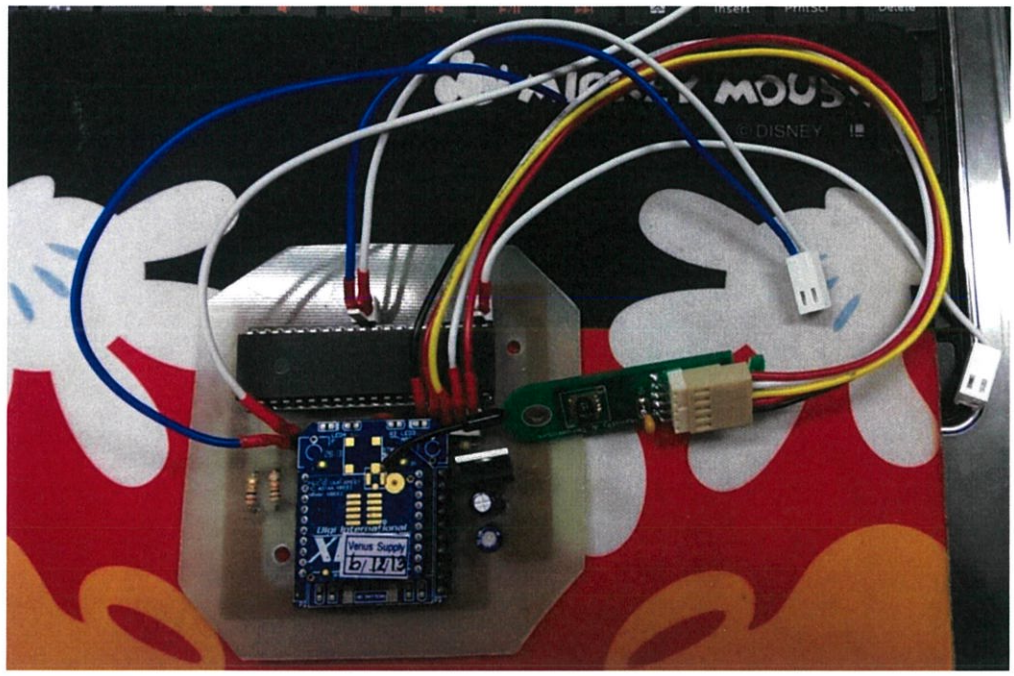
รูปที่ 3.4 รูปหน้าต่างควบคุมในVisual C#

3.2.3 การออกแบบวงจรภาครับ

วงจรในภาครับจะนำข้อมูลที่อ่านได้จากเซนเซอร์ส่งไปเก็บค่าผ่านไมโครคอนโทรลเลอร์ PIC18F4520 หลังจากนั้นจะส่งข้อมูลไปยังโมดูล ZigBee ที่อยู่บนแผงวงจรภาครับ เพื่อส่งค่าไปให้กับภาคส่งที่ติดต่ออยู่กับคอมพิวเตอร์ โดยภาครับจะใช้เซนเซอร์วัดอุณหภูมิและความชื้น SHT 15 รูปวงจรภาครับแสดงในรูป 3.3

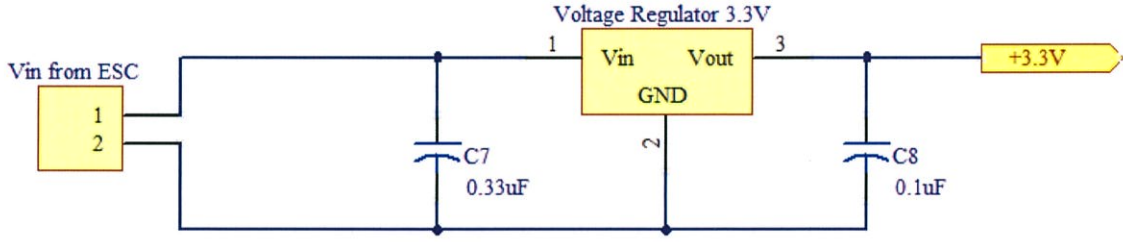


รูปที่ 3.5 วงจรสมบูรณ์ของภาครับสัญญาณ



รูปที่ 3.6 วงจรที่ใช้จริงของภาครับสัญญาณ

3.2.4 วงจรแปลงไฟของแผงวงจรภาครับ



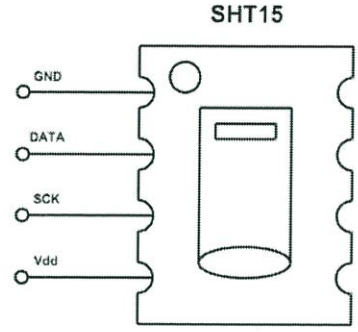
รูปที่ 3.7 วงจรแปลงไฟของภาครับสัญญาณ

การทำงานของวงจรเริ่มจากการเพิ่มแรงดันขนาด 5 โวลต์ จาก ESC ผ่าน IC LM1117T-3.3 เพื่อลดแรงดันจาก 5 โวลต์ ให้เหลือ 3.3 โวลต์ เพื่อจ่ายให้กับโมดูล Zigbee แสดงดังรูป 3.4

3.2.5 ไอซีวัดอุณหภูมิและความชื้น SHT15

เซ็นเซอร์ SHT15 เป็นโมดูลวัดอุณหภูมิและความชื้นสัมพัทธ์ในเซ็นเซอร์เดียวกัน การเชื่อมต่อขาของไอซีวัดอุณหภูมิจะมี 4 ขา ขาแรกเป็นกราวด์ ขาที่สองเป็นขาข้อมูล นำไปต่อกับขา RB4 (ขาที่ 37) ของไมโครคอนโทรลเลอร์ ขาที่ 3 เป็นขา SCK ซึ่งเป็นสัญญาณนาฬิกาสำหรับ

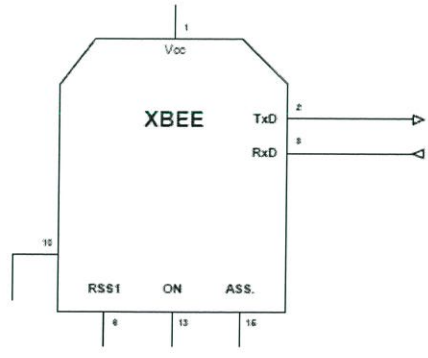
synchronize กันระหว่าง microcontroller กับ sensor โดยต่อกับขา RB5 (ขาที่ 38) ของไมโครคอนโทรลเลอร์ และขาที่ 4 เป็นขาที่รับไฟเลี้ยง ดังรูป 3.5



รูปที่ 3.8 การต่อ IC SHT15 เข้ากับไมโครคอนโทรลเลอร์

3.2.6 วงจรเชื่อมต่อโมดูล ZigBee

โมดูลZigBee จะทำหน้าที่ติดต่อสื่อสารของมูลแบบไร้สายโดยต้องการไฟเลี้ยง 3.3 โวลต์ ต่อเข้าที่ขา 1 (Vcc) และขาส่งข้อมูล (TxD) ต่อเข้ากับขารับข้อมูลของไมโครคอลโทรลเลอร์ที่ขา 26 (RxD) ส่วนขารับข้อมูลของโมดูล ZigBee (RxD) ต่อเข้ากับขาส่งข้อมูลของไมโครคอนโทรลเลอร์ที่ขา 25 (TxD) ดังรูป 3.6



รูปที่ 3.9 การเชื่อมต่อโมดูล ZigBee

3.3 การออกแบบในส่วนของโปรแกรม

การออกแบบในส่วนของโปรแกรมจะเป็นโปรแกรมที่ใช้ในการควบคุมไมโครคอนโทรลเลอร์ให้ติดต่อสื่อสารกับเซนเซอร์ SHT15 ทำการแปลงค่าเป็นข้อมูลดิจิตอล และส่งข้อมูลไปให้โมดูล ZigBee เพื่อส่งสัญญาณไร้สายไปที่ ZigBee ภาคส่ง

การออกแบบโปรแกรมติดต่อสื่อสารกับ IC SHT15 เริ่มต้นด้วยการ Reset การเชื่อมต่อเพื่อให้ SHT15 พร้อมที่จะเริ่มการเชื่อมต่อกับไมโครคอนโทรลเลอร์ ก่อนจะที่จะส่งสัญญาณ Transmission Start เพื่อเริ่มการเชื่อมต่อกับไมโครคอนโทรลเลอร์ จากนั้นก็จะสามารถอ่านอุณหภูมิ-ความชื้นจาก SHT15 ได้ แล้วส่งข้อมูลที่อ่านได้ไปให้ ZigBee เพื่อส่งสัญญาณไร้สายไปให้ ZigBee ภาคส่ง

3.3.1 การคำนวณค่าอุณหภูมิ

ในการอ่านค่าอุณภูมิจากโมดูล SHT15 สามารถเลือกความละเอียดในการได้อ่านได้ในแบบ 14 บิต หรือ 12 บิต โดยที่ความละเอียด 14 บิต เป็นค่าตั้งต้น โดยการคำนวณค่าข้อมูลจะต้องอ่านข้อมูลดิบจากโมดูล SHT15 เข้ามาก่อน จากนั้นจึงใช้กระบวนการทางคณิตศาสตร์เพื่อให้ได้ค่าอุณหภูมิออกมา โดยคำนวณได้จากสมการ

$$T = d_1 + d_2 \cdot SO_T \qquad (3.1)$$

เมื่อ T คือค่าอุณหภูมิจริง

SO_T คือ serial output Temperature

d_1 คือค่าคงที่ที่ขึ้นอยู่กับไฟเลี้ยงที่ป้อนให้กับขา V_{dd} ของ SHT 15

d_2 คือค่าคงที่ที่ขึ้นอยู่กับความละเอียดของอุณหภูมิที่ต้องการจาก SHT15

ตารางที่ 3.1 ค่าคงที่ d_1 และ d_2

SO_T	d_2 °C	d_2 °C
14bit	0.01	0.018
12bit	0.04	0.072

V_{dd}	d_1 °C	d_1 °F
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

3.3.2 การคำนวณค่าความชื้นสัมพัทธ์

ในการอ่านค่าความชื้นสัมพัทธ์จากโมดูล SHT15 สามารถเลือกความละเอียดในการได้อ่านได้ในแบบ 12 บิต หรือ 8 บิต โดยที่ความละเอียด 12 บิต เป็นค่าตั้งต้น โดยการคำนวณค่าข้อมูลจะต้องอ่านข้อมูลดิบจากโมดูล SHT15 เข้ามาก่อน จากนั้นจึงใช้กระบวนการทางคณิตศาสตร์เพื่อให้ได้ค่าอุณหภูมิออกมา โดยคำนวณได้จากสมการ

$$RH_{true} = (T - 25) \cdot [t_1 + (t_2 \cdot SO_{RH})] + RH_{linear} \quad (3.2)$$

$$RH_{linear} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot SO_{RH}^2 \quad (3.3)$$

เมื่อ RH_{true} คือค่าความชื้นสัมพัทธ์จริง

T คือค่าอุณหภูมิที่คำนวณได้จากสมการ (3.1)

SO_{RH} คือ serial output humidity แบบ linear

t_1, t_2 คือค่าคงที่ที่ขึ้นอยู่กับการชดเชยของอุณหภูมิ

c_1, c_2, c_3 คือค่าคงที่ที่ขึ้นอยู่กับความละเอียดของความชื้นสัมพัทธ์ที่ต้องการจาก SHT15

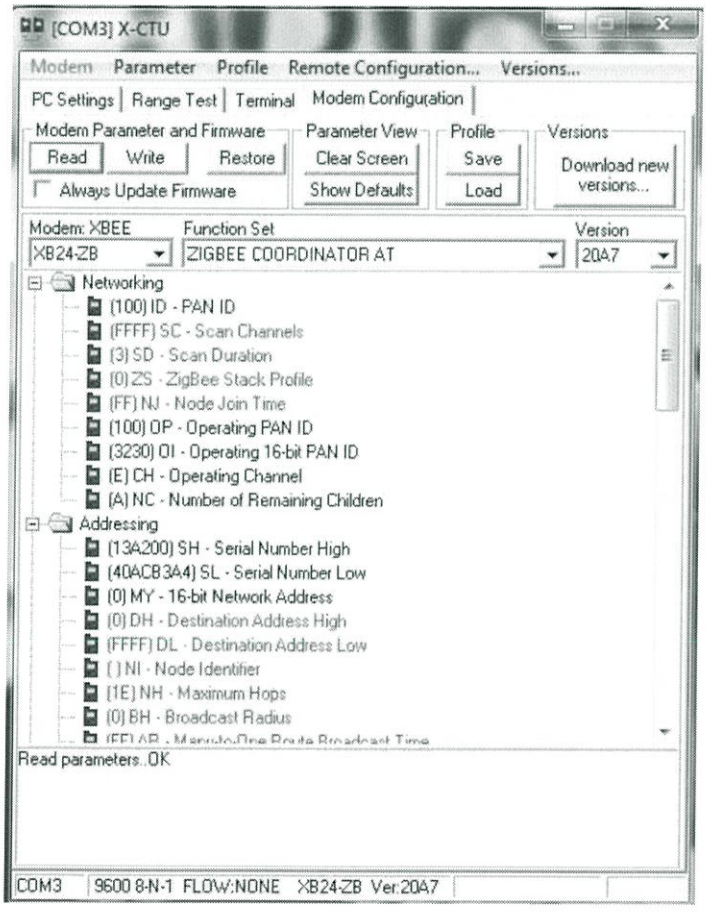
ตารางที่ 3.2 ค่าคงที่ t_1, t_2 และ c_1, c_2, c_3

SO_{RH}	t_1	t_2
12 bit	0.01	0.00008
8 bit	0.01	0.00128

SO_{RH}	c_1	c_2	c_3
12 bit	-4	0.0405	-2.8×10^{-6}
8 bit	-4	0.648	-7.2×10^{-4}

3.3.3 การตั้งค่า Configuration ให้แก่โมดูล XBee

3.3.3.1 การกำหนดพารามิเตอร์ให้กับ Coordinator



รูปที่ 3.10 การกำหนดพารามิเตอร์ต่างๆ ของ ZigBee Coordinator

ทำการกำหนดค่าพารามิเตอร์ให้กับ ZigBee ภาครับ ซึ่งทำงานเป็น Coordinator โดยที่โมดูล XBee

ในการติดต่อสื่อสารระหว่างโมดูล XBee นั้นมีค่าพารามิเตอร์ที่ใช้ในการจัดการเกี่ยวกับเครือข่ายซึ่งมีพารามิเตอรสำคัญๆที่ต้องกำหนดค่าดังนี้

1) CH (Channel) ใช้ในการกำหนดช่องสัญญาณ กำหนดเป็น 15

2) ID (PAN ID) ใช้กำหนดหมายเลขเครือข่าย กำหนดเป็น 1111 ซึ่งจะสามารถส่งข้อมูลไปยังทุกเครือข่ายได้

3) DH (Destination Address High) ใช้กำหนดแอดเดรสของโมดูลตัวรับกำหนด เป็นค่าของ SH

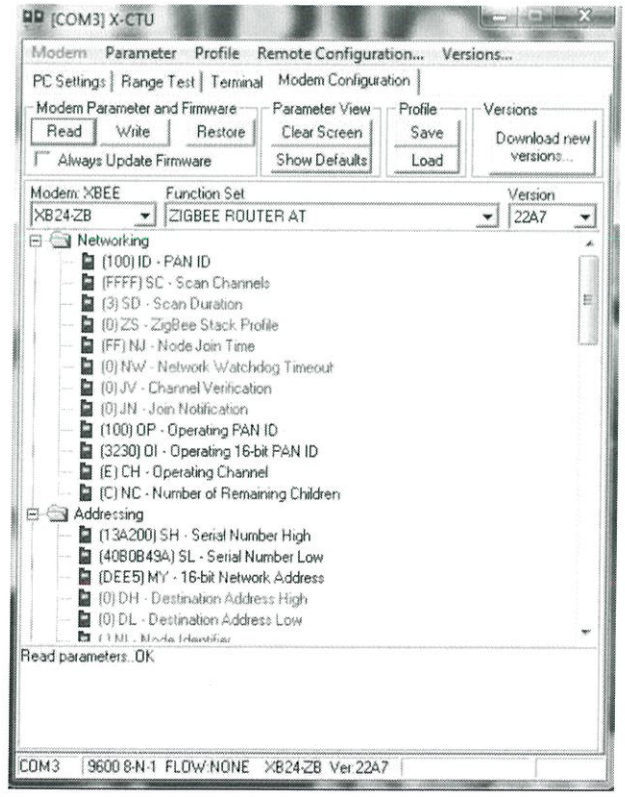
4) DL (Destination Address Low) ใช้กำหนดแอดเดรสของโมดูลตัวส่งกำหนด เป็นค่าของ SL

5) MY (16-bit Source Address) ใช้กำหนดแอดเดรส 16 บิตของแต่ละโมดูลกำหนดเป็น 0xFFE, 0xFFF 2 ค่า เพื่อยกเลิกแอดเดรสนี้ไปใช้รีจิสเตอร์ SH,SL

6) SH/SL (Serial Number High/Low) เป็นรีจิสเตอร์ที่ใช้เก็บค่าหมายเลขเฉพาะ (Serial Number) ของแต่ละโมดูลไม่สามารถเปลี่ยนแปลงได้

7) CE (Coordinator Enable) ใช้กำหนดการทำงานของ ZigBee เป็น Coordinator ซึ่ง active เป็น 1

3.3.3.2 การกำหนดพารามิเตอร์ให้กับ Router



รูปที่ 3.11 การกำหนดพารามิเตอร์ต่างๆ ของ ZigBee Router

สำหรับ ZigBee ที่ภาคส่ง จะทำงานเป็น Router โดยที่พารามิเตอร์ คล้ายคลึงกับZigBeeที่ภาครับดังนี้

3.3.3.3 CH (Channel) กำหนดเป็น15 (เหมือนกันทั้งเครือข่าย)

3.3.3.4 ID (PAN ID) กำหนดเป็น 1111

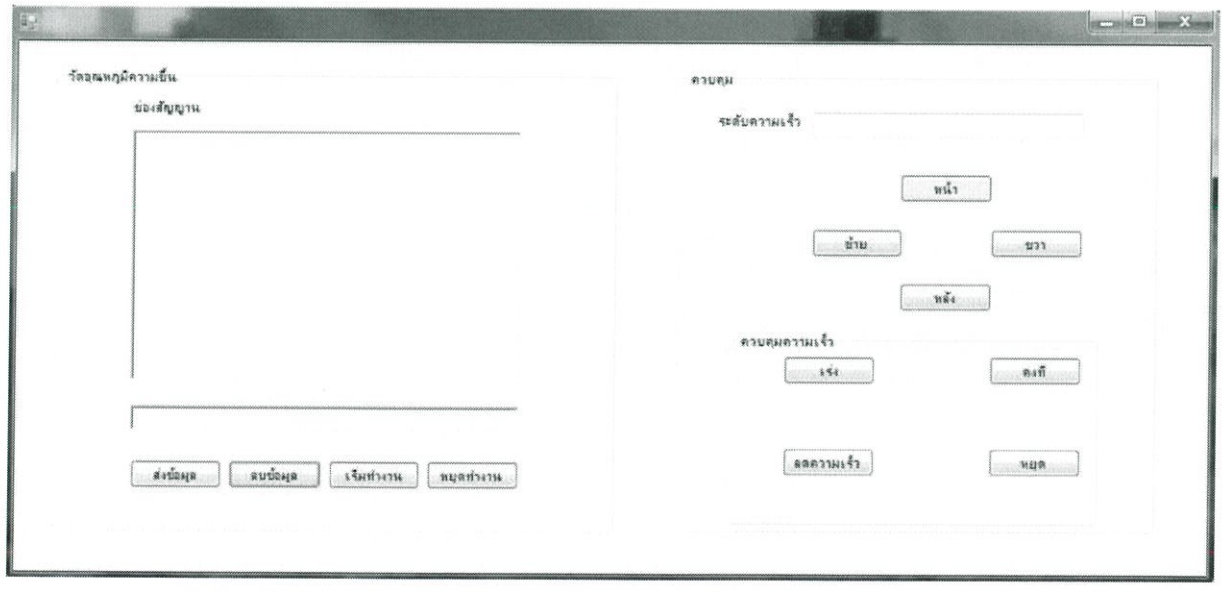
3.3.3.5 DH (Destination Address High) กำหนด 00

3.3.3.6 DL (Destination Address Low) กำหนด 00

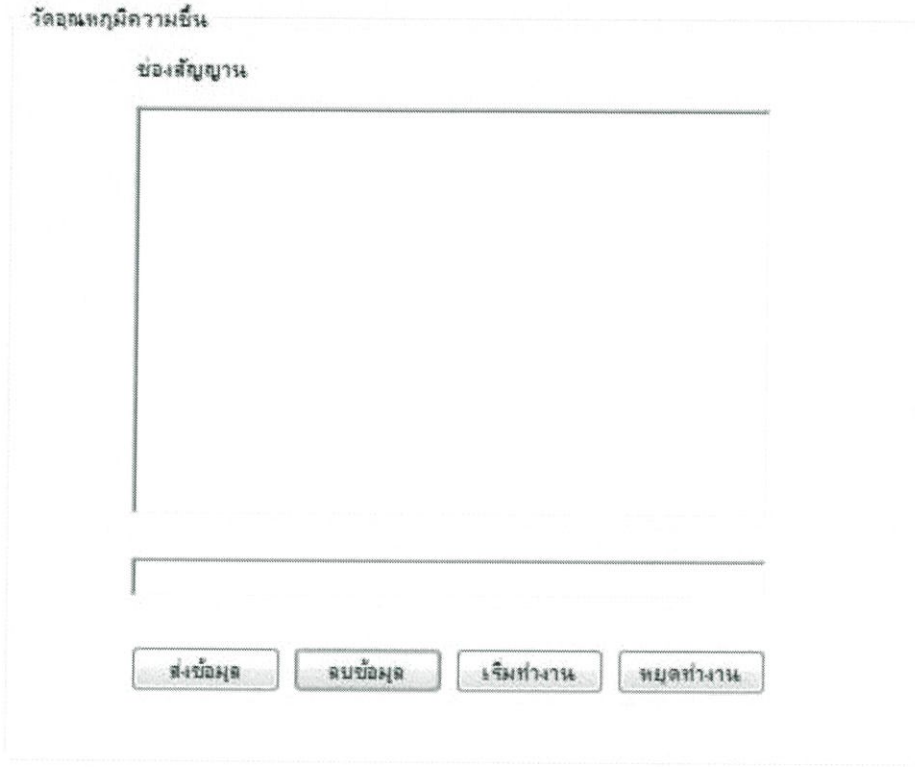
3.3.3.7 MY (16 bit Source Address) กำหนดเป็น 0xFFE, 0xFFE 2 ค่า เพื่อยกเลิก แอดเดรสนี้ไปใช้ริจิตเตอร์

3.4 การออกแบบ Graphical User Interface

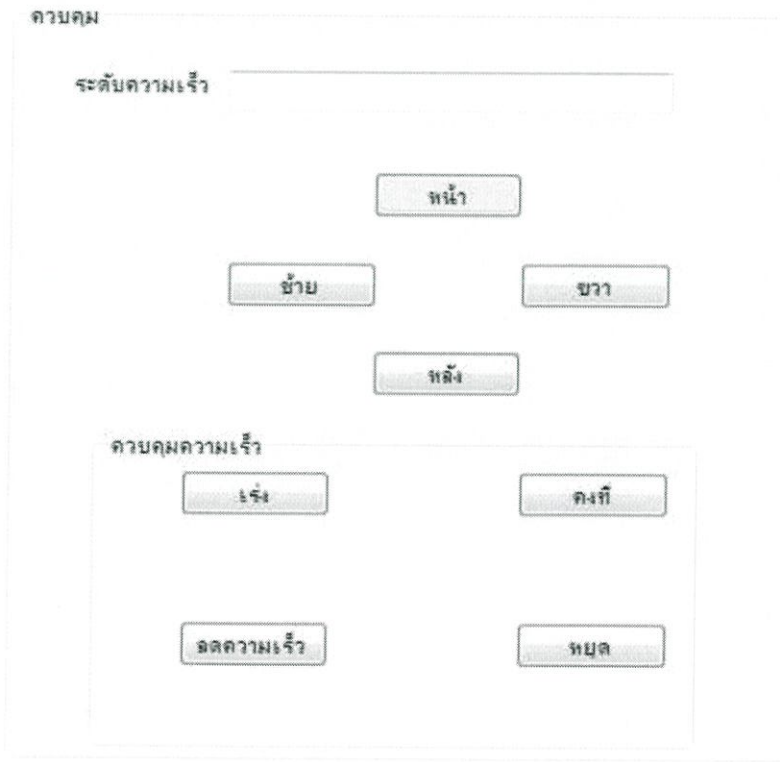
ออกแบบ Graphical User Interface โดยใช้ภาษา C# ผ่านโปรแกรม Microsoft Visual Studio โปรแกรมทำงานโดยการรับค่าผ่าน serial port ซึ่งจะแสดงในส่วน Terminal ของโปรแกรม จากนั้นจะแยกข้อมูลแสดงตามตำแหน่งที่ได้รับมา นอกจะนี้ข้อมูลที่ได้รับมาจะถูกจัดเก็บในฐานข้อมูล โดยสามารถเรียกดูข้อมูลมาแสดง



รูปที่ 3.12 หน้าต่างโปรแกรมที่ออกแบบ



รูปที่ 3.13 ส่วน Terminal ของโปรแกรม



รูปที่ 3.14 ส่วนของการควบคุมคอปเตอร์

3.5 เครื่องมือที่ใช้ในการทดลอง

3.5.1 เครื่องจ่ายไฟ (Power Supply) แหล่งจ่ายไฟหรือที่มักจะเรียกทับศัพท์ว่าเพาว์เวอร์ซัพพลายแสดงได้เป็นส่วนประกอบที่สำคัญส่วนหนึ่งทำหน้าที่แปลงสัญญาณ ไฟฟ้ากระแสสลับจากแหล่งกำเนิดให้เป็นไฟฟ้ากระแสตรงด้วยความต่างศักย์ที่เหมาะสมก่อนนำไปใช้งานต่อไป

3.5.2 มัลติมิเตอร์ (Multimeter) ใช้ในการวัดค่าแรงดัน

3.5.3 เทอร์โมมิเตอร์ (Thermometer) ใช้ในการวัดอุณหภูมิเปรียบเทียบกับข้อมูลอุณหภูมิที่ได้รับ

3.5.4 ออสซิลโลสโคป (Oscilloscope) ออสซิลโลสโคปเป็นเครื่องมือวัดซึ่งจะทำให้เราเห็นรูปร่างของสัญญาณไฟฟ้าโดยแสดงเป็นกราฟของแรงดันบนแกนเวลาที่จอภาพ เหมือนกับเป็นโวลท์มิเตอร์ที่มีฟังชั่นพิเศษแสดงค่าแรงดันที่เปลี่ยนไปตามเวลา และด้วยช่องตารางขนาด 1 เซนติเมตร ทำเราให้สามารถวัด ค่าแรงดันกับเวลาจากจอได้

3.5.5 เครื่องวัดอุณหภูมิและความชื้น HTC-1 ใช้ในการวัดอุณหภูมิและความความชื้นเปรียบเทียบกับข้อมูลที่วัดได้จากเครื่อง ช่วงอุณหภูมิที่สามารถวัดได้: -50~ +70 °C (-58~ +158°F)

ช่วงความชื้อนที่สามารถวัดได้: 10%-99% RH ความละเอียด (Resolution): temperature: 0.1°C(0.1°F), , humidity 1% RH ความแม่นยำ (Accuracy): temperature: +-1 °C (1.8°F), humidity: +-5% RH(40%-80%)

3.5.6 เครื่องวิเคราะห์สเปคตรัม (Spectrum Analyzer) เป็นเครื่องมือที่ใช้เพื่อตรวจสอบองค์ประกอบทางสเปคตรัมของคลื่นไฟฟ้า, คลื่นเสียง หรือคลื่นแสง ซึ่งมักจะวัดค่ากำลังของสเปคตรัม (power spectrum)

3.6 การจัดเก็บผลการทดลอง

3.6.1 การจัดเก็บผลการทดลองวัดสเปกตรัมและการส่งข้อมูลไร้สายผ่านโมดูล ZigBee

3.6.2 การทดลองเพิ่มลดอุณหภูมิ-ความชื้นเพื่อเปรียบเทียบผลระหว่างอุณหภูมิ-ความชื้นและหน้าต่างแสดงผลคอมพิวเตอร์

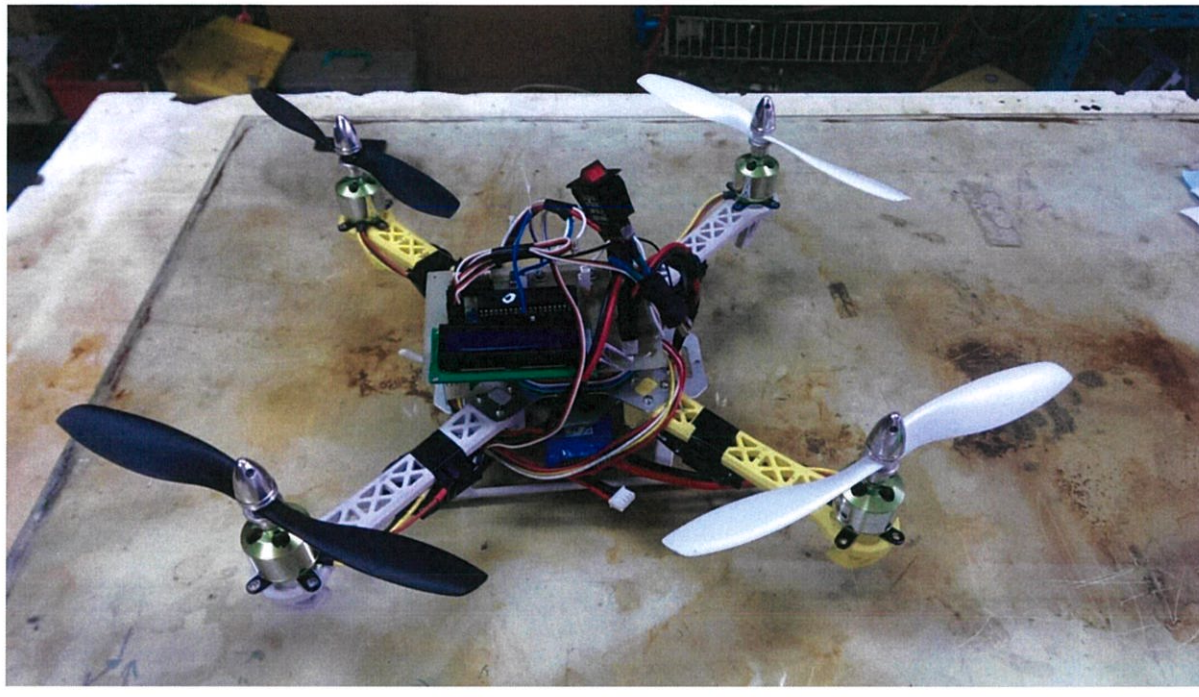
3.6.3 การทดลองเพื่อทดสอบระยะทางในการรับส่งของข้อมูลของ ZigBee

3.6.4 การทดลองการส่งข้อมูลของ SHT15

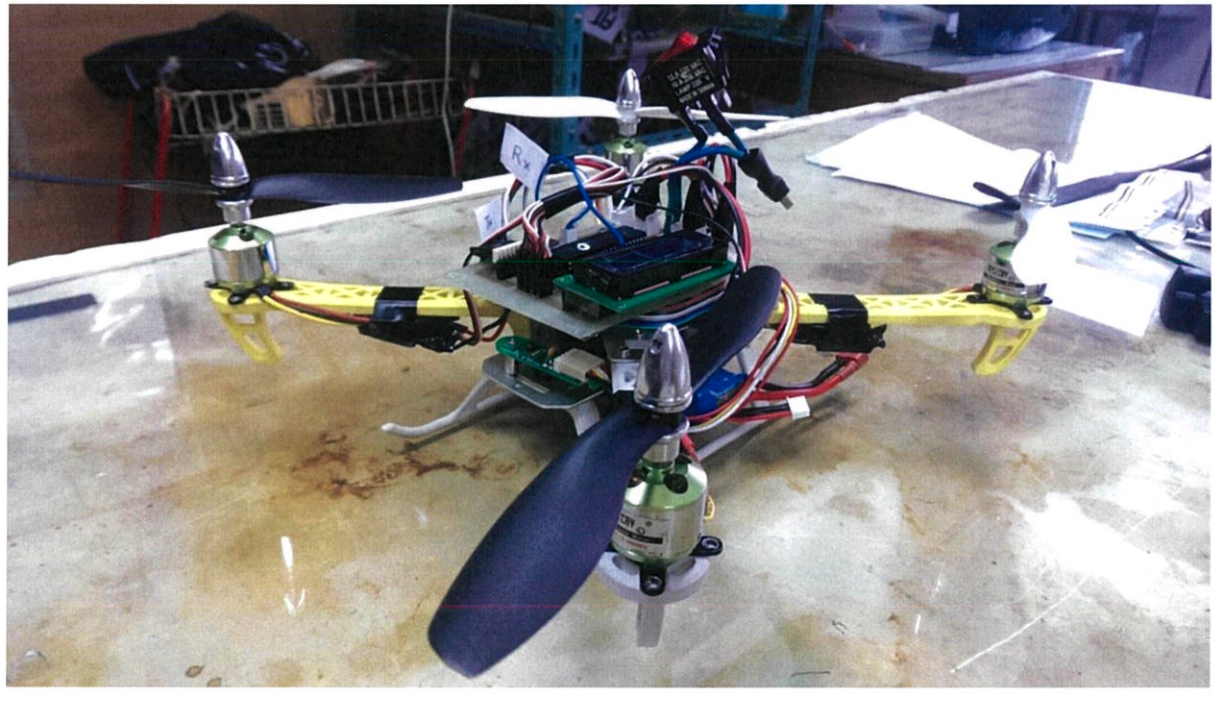
3.6.5 การทดลองส่ง Feedback ควบคุม Heater และ Cooler

3.6.6 การทดลองใช้งาน Database

3.7 ชิ้นงานที่เสร็จสมบูรณ์



รูปที่ 3.15 Quad Copter



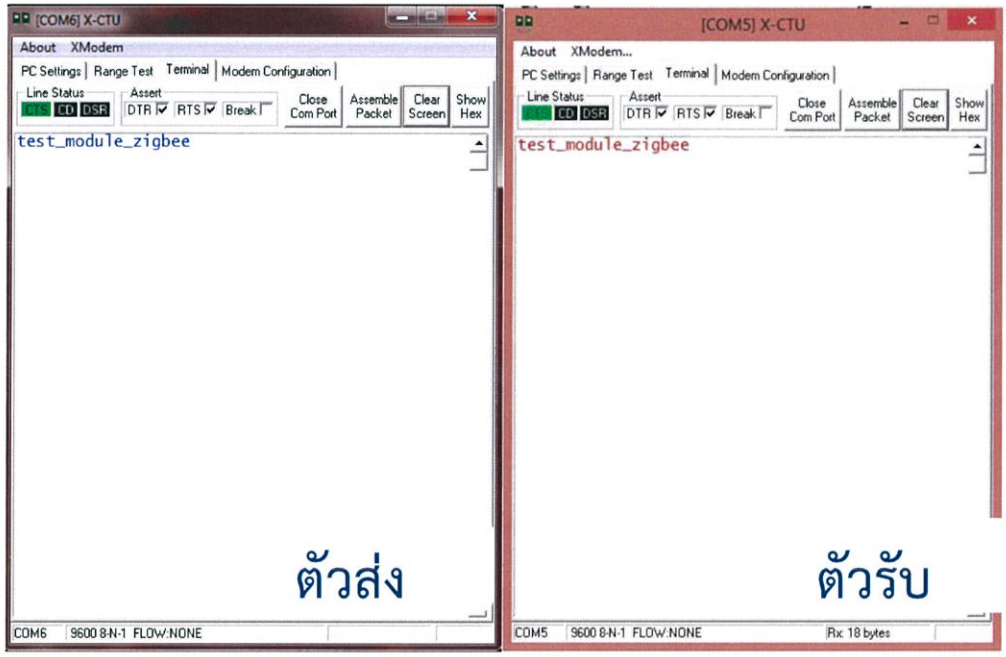
รูปที่ 3.16 Quad Copter

บทที่ 4

ผลการทดลอง

4.1 ทดลองรับ-ส่งข้อมูลไร้สายจากเซนเซอร์

เมื่อทำการ Configuration ZigBee ให้สามารถติดต่อสื่อสารกันได้แล้ว เราจะทำการทดลองโดยพิมพ์ข้อมูลในด้านฝั่งส่ง ข้อมูลจะถูกส่งมายังคอมพิวเตอร์ผ่านการติดต่อสื่อสารระหว่างโมดูล XBee ข้อมูลที่พิมพ์จะถูกแสดงบน X-CTU แบบ Real-Time ดังรูปที่ 4.2



รูปที่ 4.1 แสดงการทดลองรับส่งข้อมูลของ ZigBee

4.2 การทดลองเพิ่ม-ลด อุณหภูมิ-ความชื้นเพื่อเปรียบเทียบผลระหว่างเครื่องอุณหภูมิ-ความชื้นและหน้าต่างแสดงผลคอมพิวเตอร์

ในการทดลองนี้เป็นการทดลองเปรียบเทียบว่าค่าอุณหภูมิที่อ่านได้จากเครื่องวัดอุณหภูมิ-ความชื้นและหน้าต่างแสดงผลคอมพิวเตอร์มีค่าเท่ากันหรือไม่ เพื่อเป็นการสังเกตว่าค่าที่ได้จากการวัดอุณหภูมิมีความแม่นยำมากน้อยเพียงใด

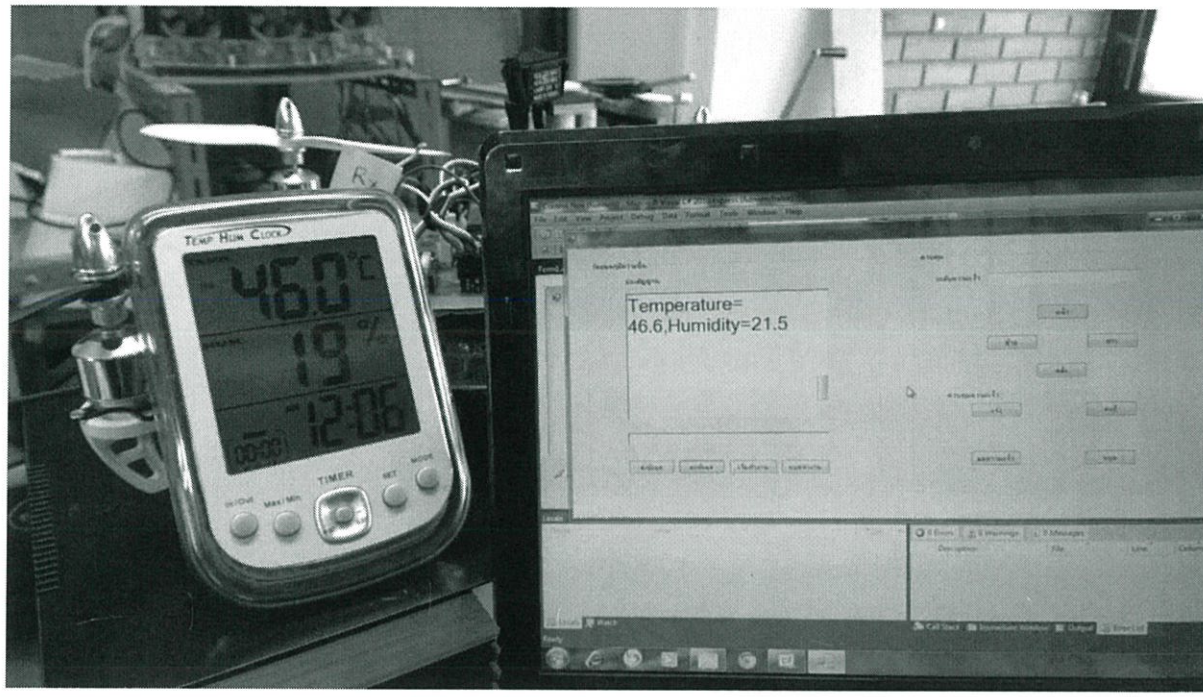
4.2.1 วิธีการทดลอง

นำเครื่องวัดอุณหภูมิ-ความชื้นและวงจรภาคส่งข้อมูลไปไว้ในพื้นที่เดียวกัน และให้วงจรภาคส่งทำการส่งข้อมูลมายังภาครับซึ่งอยู่ห่างกัน ทำการอ่านค่าอุณหภูมิจากเครื่องวัดอุณหภูมิ-ความชื้นและหน้าต่างแสดงผลจากจอคอมพิวเตอร์ว่ามีค่าเท่ากันหรือไม่ เปลี่ยนสถานที่วัดผลเรื่อยๆ และบันทึกผล

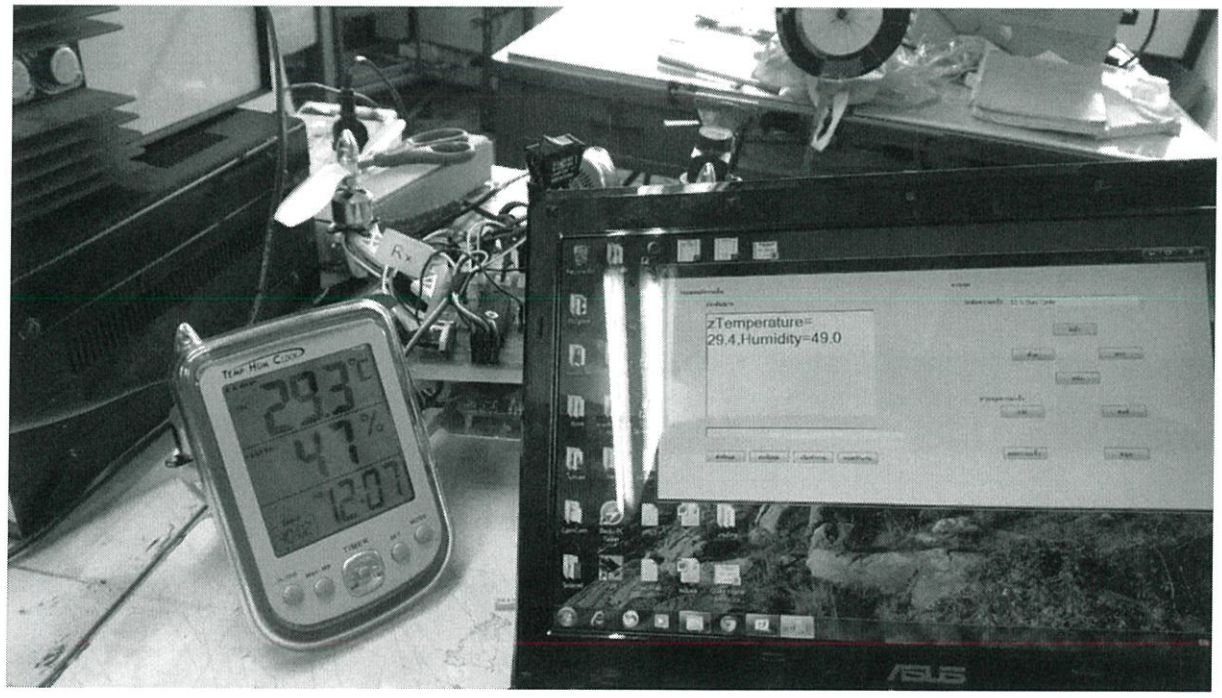
4.2.2 ผลการทดลอง

ตารางที่ 4.1แสดงผลการทดลองระหว่างอุณหภูมิที่วัดได้จากเทียบกับจอคอมพิวเตอร์

เครื่องวัดอุณหภูมิ-ความชื้น		จอคอมพิวเตอร์		% ความผิดพลาด	
อุณหภูมิ	ความชื้น	อุณหภูมิ	ความชื้น	อุณหภูมิ	ความชื้น
7.6	64	7.7	85.3	1.32	33.28
9.2	69	8.3	85.7	9.78	24.20
10.0	72	11.3	73.2	13	1.67
13.0	73	13.0	70.2	0	3.84
18.5	72	18.2	71.0	1.62	1.38
20.3	70	20.8	71.4	2.46	2
22.6	65	23.1	68.2	2.21	4.92
25.9	61	25.6	62.1	1.15	1.80
28.5	73	28.7	74.8	0.70	2.46
33.9	47	35.6	44.5	5.01	5.31
36.5	44	35.9	45.4	1.64	3.40
38.5	40	40.1	37.5	3.90	0.25
48.0	16	49.9	19.0	3.95	18.75
60.0	-	60.1	13.1	0.16	-
72.0	-	71.0	7.5	1.39	-
75.0	-	74.2	7.5	1.06	-
82.0	-	81.0	6.9	1.22	-



รูปที่ 4.2 แสดงการทดลองวัดอุณหภูมิเปรียบเทียบผลระหว่างเครื่องอุณหภูมิ-ความชื้นและหน้าต่าง แสดงผลคอมพิวเตอร์



รูปที่ 4.3 แสดงการทดลองวัดอุณหภูมิเปรียบเทียบผลระหว่างเครื่องอุณหภูมิ-ความชื้นและหน้าต่าง แสดงผลคอมพิวเตอร์

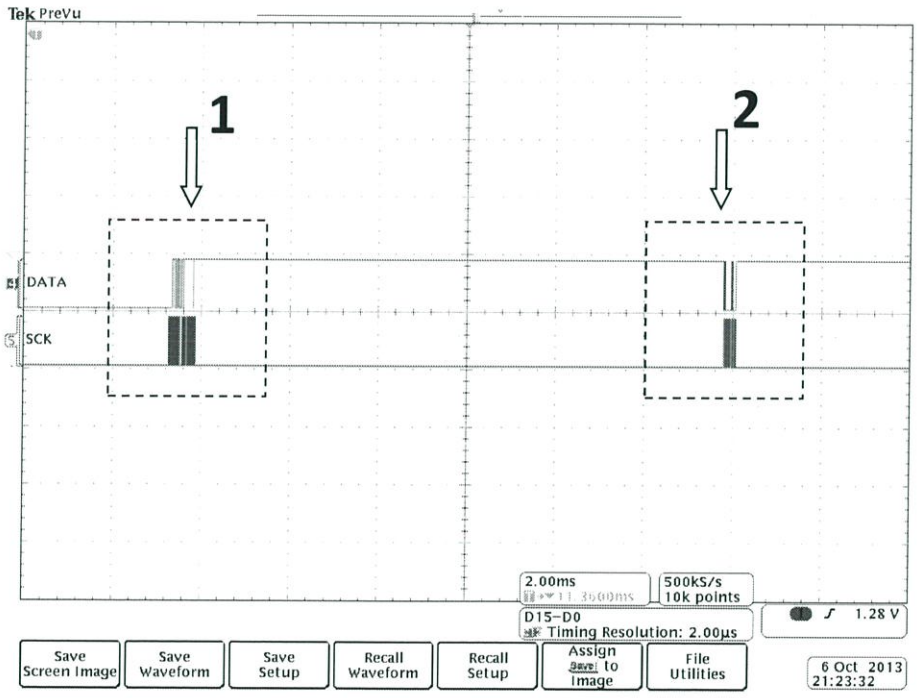
4.3 การทดลองการส่งข้อมูลของ SHT15

ในการทดลองนี้เป็นการทดลองเพื่อหาลักษณะของข้อมูลที่ออกมาจากเซ็นเซอร์ SHT15 และลักษณะของข้อมูลที่ผ่านการประมวลผลจากไมโครคอนโทรเลอร์

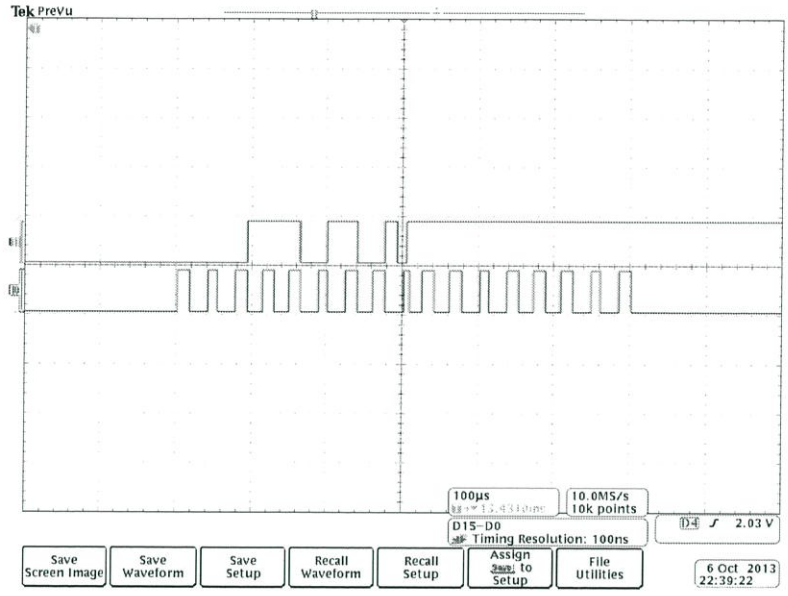
4.3.1 วิธีการทดลอง

การทดลองเพื่อหาลักษณะของข้อมูลที่ออกมาจากเซ็นเซอร์ SHT15 จะวัดที่ขา SCK และขา Data ส่วนลักษณะของ จะวัดที่ขา Tx ของไมโครคอนโทรลเลอร์

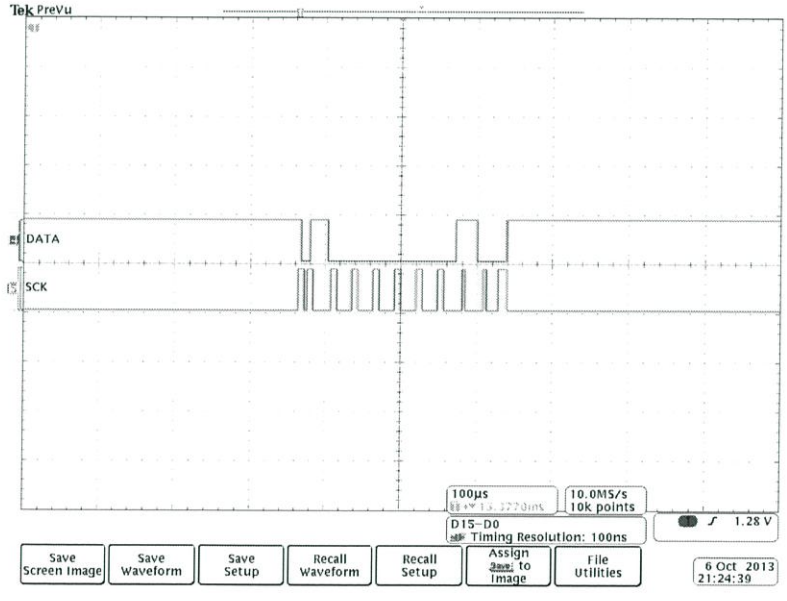
4.3.2 ผลการทดลอง



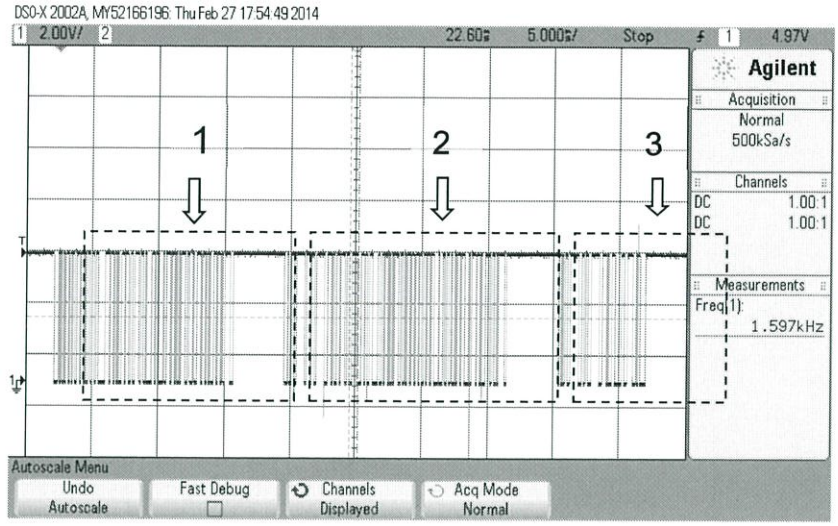
รูปที่ 4.4 ข้อมูลที่วัดจาก SHT15



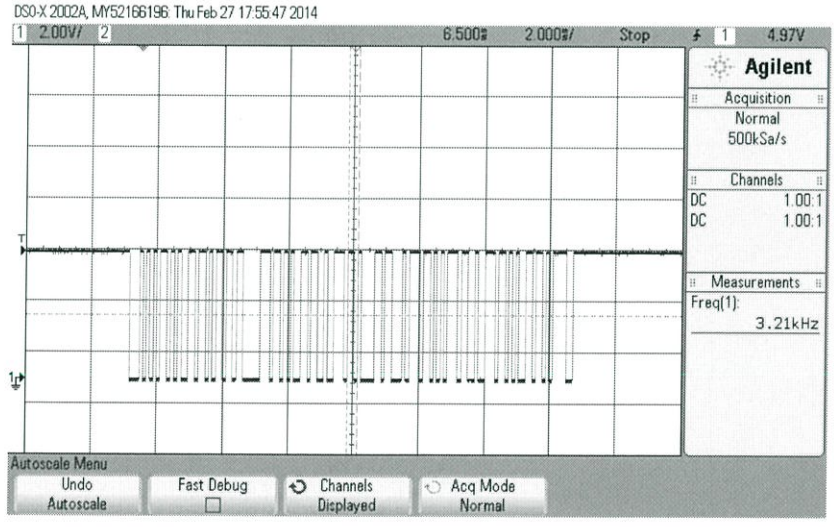
รูปที่ 4.5 ข้อมูลที่วัดจาก SHT15 ช่วงที่ 1 ส่วนที่เป็นอุณหภูมิ



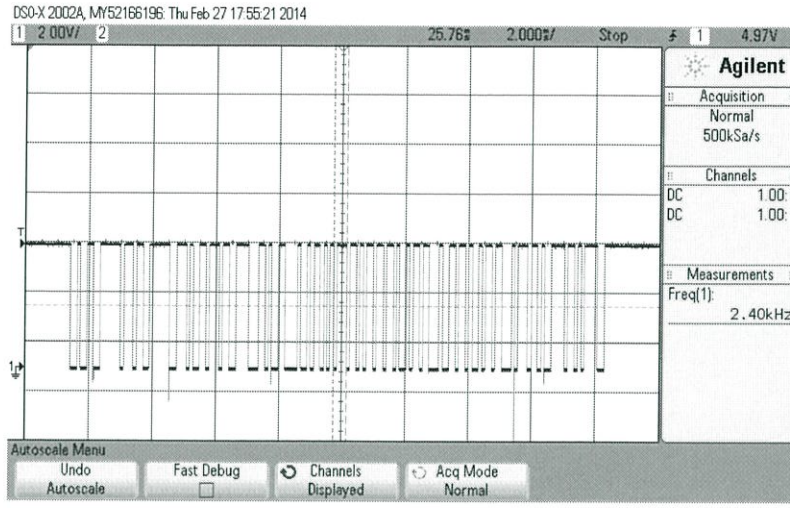
รูปที่ 4.6 ข้อมูลที่วัดจาก SHT15 ช่วงที่ 2 ส่วนที่เป็นความชื้น



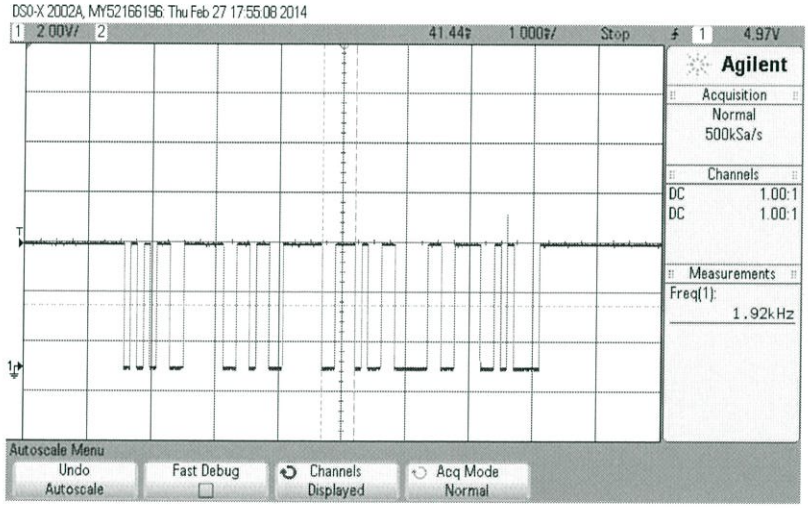
รูปที่ 4.7 ข้อมูลที่ผ่านการประมวลผลจากไมโครคอนโทรลเลอร์ ส่วนออกขา Tx ของไมโครคอมโทรลเลอร์ ไปเข้าขา Rx ของ Zigbee



รูปที่ 4.8 ข้อมูลที่ผ่านการประมวลผลจากไมโครคอนโทรลเลอร์ ช่วงที่ 1



รูปที่ 4.9 ข้อมูลที่ผ่านการประมวลผลจากไมโครคอนโทรลเลอร์ ช่วงที่ 2



รูปที่ 4.10 ข้อมูลที่ผ่านการประมวลผลจากไมโครคอนโทรลเลอร์ ช่วงที่ 3

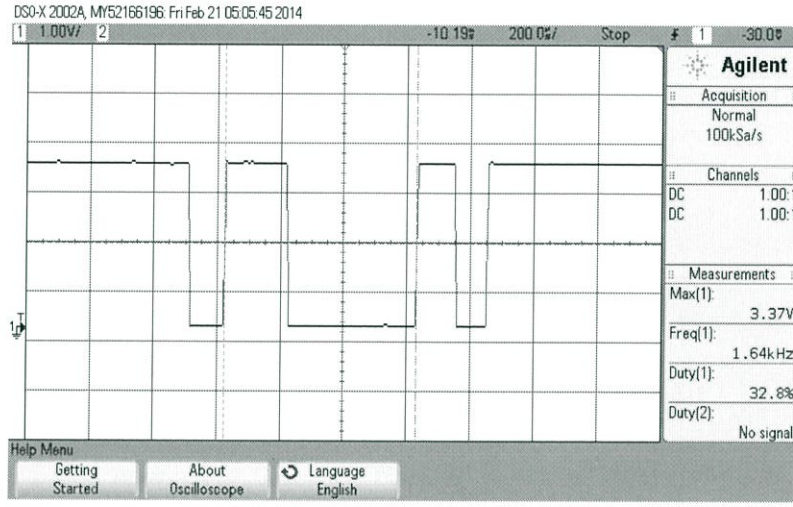
4.4 การทดลองรับ-ส่งข้อมูลระหว่าง Zigbee เพื่อควบคุมการทำงานของคอปเตอร์

ในการทดลองนี้เป็นการทดลองการส่งข้อมูลคำสั่ง เพื่อควบคุมการทำงานของคอปเตอร์ ให้เคลื่อนที่ไปยังทิศทางที่ต้องได้ โดยจะส่งเป็นตัวอักษรออกไป

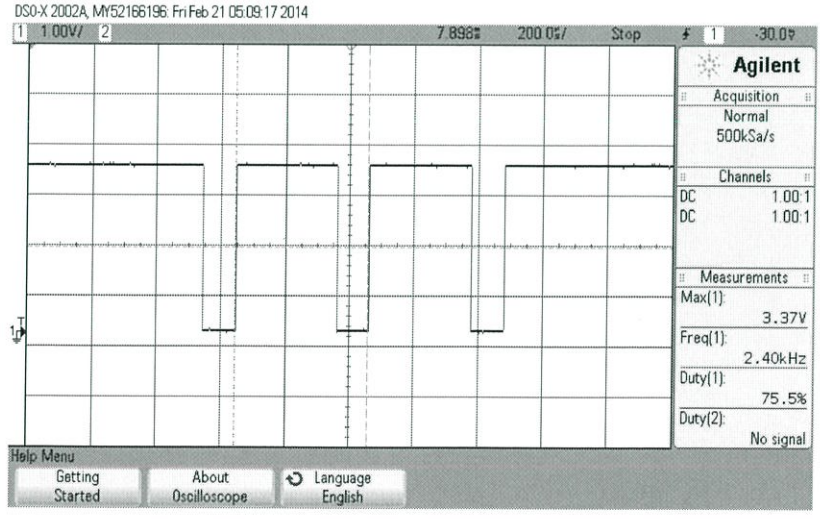
4.4.1 วิธีการทดลอง

การทดลองเพื่อหาลักษณะของข้อมูลที่ออกมาจาก Zigbee จะวัดที่ขา Rx ของไมโครคอนโทรเลอร์

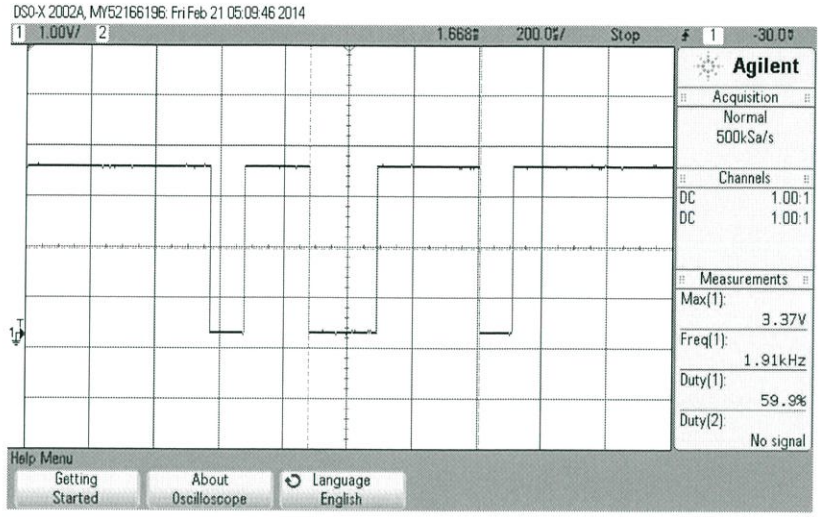
4.4.2 ผลการทดลอง



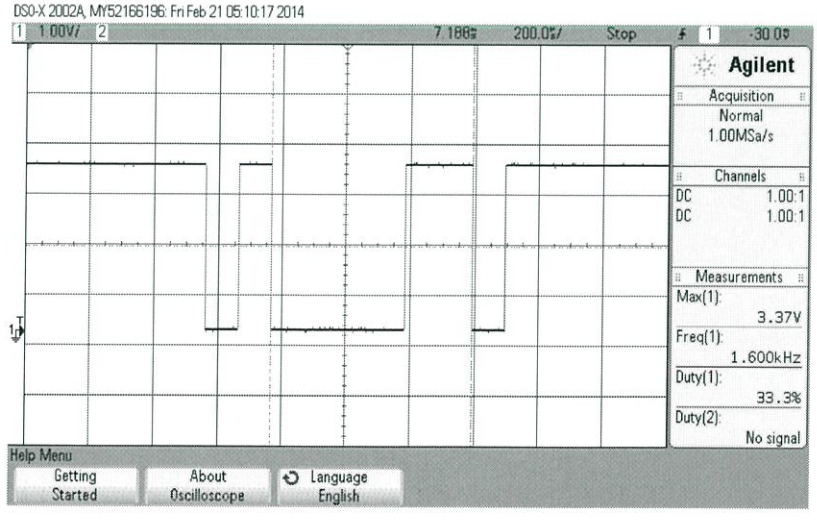
รูปที่ 4.11 ข้อมูลตัวอักษร “C” สั่งให้เซ็นเซอร์วัดอุณหภูมิ และความชื้นทำงาน



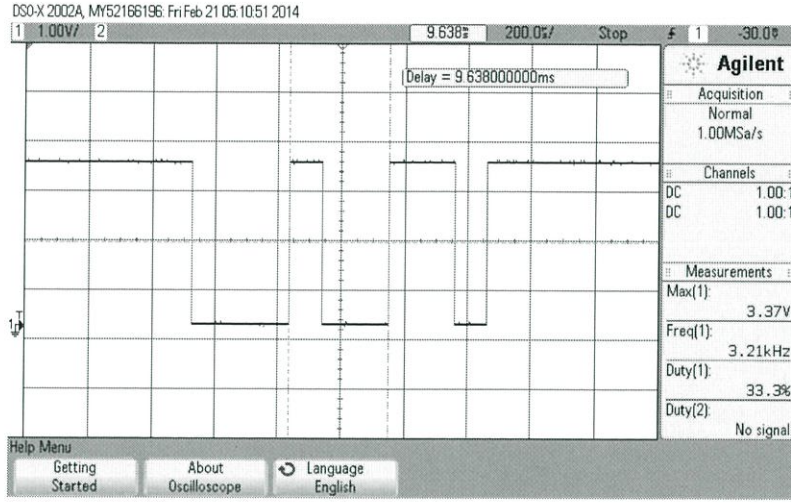
รูปที่ 4.12 ข้อมูลตัวอักษร “w” สั่งให้คอปเตอร์เคลื่อนที่ไปข้างหน้า



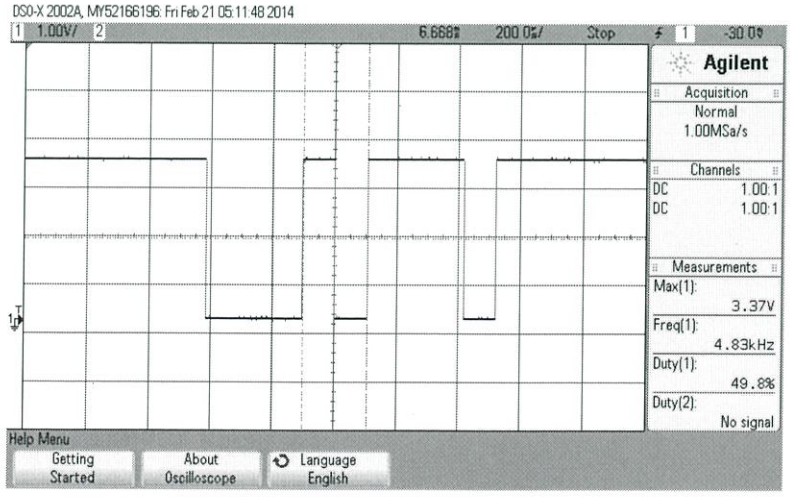
รูปที่ 4.13 ข้อมูลตัวอักษร “s” สั่งให้คอปเตอร์เคลื่อนที่ไปข้างหลัง



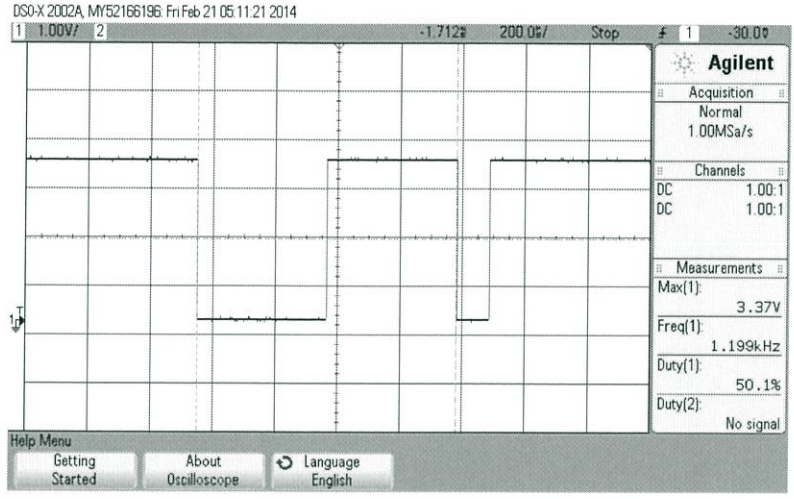
รูปที่ 4.14 ข้อมูลตัวอักษร “a” สั่งให้คอปเตอร์เคลื่อนที่ไปทางซ้าย



รูปที่ 4.15 ข้อมูลตัวอักษร “d” สั่งให้คอปเตอร์เคลื่อนที่ไปทางขวา



รูปที่ 4.16 ข้อมูลตัวอักษร “t” สั่งให้คอปเตอร์บินนิ่งอยู่กับที่



รูปที่ 4.17 ข้อมูลตัวอักษร “x” สั่งให้คอปเตอร์หยุดการทำงาน

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

โครงงานนี้ เป็นการนำ Zigbee มาประยุกต์ใช้ในการส่งข้อมูลคำสั่งไปยัง microcontroller ที่อยู่บนตัวคอปเตอร์ ซึ่งสามารถส่งข้อมูลคำสั่งได้อย่างถูกต้อง และสามารถควบคุมการทำงานของคอปเตอร์ได้ตามที่ต้องการ และนำไปใช้ร่วมกับ SHT-15 เพื่อใช้เป็นเครื่องวัดอุณหภูมิไร้สาย ซึ่งอุณหภูมิและความชื้นที่วัดได้ มีค่าใกล้เคียงกับอุณหภูมิและความชื้นที่วัดได้จากHygrometer โดยจะแสดงผลที่ได้ทางคอมพิวเตอร์ผ่านโปรแกรมVisual C# ที่เป็นทั้งส่วนแสดงผลและสั่งการ โดยการออกแบบ GRAPHICAL USER INTERFACE สามารถพิมพ์ข้อมูลคำสั่งควบคุมคอปเตอร์4ใบพัดผ่านทางคีย์บอร์ดได้ ซึ่งสามารถใช้งานได้เป็นอย่างดี

5.2 ข้อเสนอแนะ

โครงงานนี้ยังสามารถพัฒนาต่อไปได้อีกโดยการเพิ่มจำนวนโหนดภาคส่งให้มากขึ้นเพื่อขยายเครือข่ายการตรวจสอบและวัดค่าจากอุณหภูมิ ความชื้น และยังสามารถพัฒนาโดยการติดตั้งกล้องเพื่อเก็บภาพจากมุมสูงหรือที่ๆเข้าถึงได้ยาก

บรรณานุกรม

[1] Drew Gislason. ZIGBEE WIRELESS NETWORKING: Newnes , 2008.

[2] Shahin Farahani. ZIGBEE WIRELESS NETWORKING AND TRANSCEIVERS: Newnes , 2008.

[3] ประภาพร ช่างไม้. คู่มือเขียนโปรแกรมภาษา C ฉบับผู้เริ่มต้น. พิมพครั้งที่1. นนทบุรี: ไอดีซี อินโฟ ดิสทริบิวเตอร์ เซ็นเตอร์ จำกัด, 2551.

[4] Gerard C. M. Meijer. Smart sensor system: WILEY, 2008.

[5] ทีมงาน Thaimicrotron. "การใช้งาน SHT1x Sensor Probe วัดอุณหภูมิ ความชื้น" http://www.thaimicrotron.com/Sensor/SHT1x.htm

[6] ทีมงาน Thaieasyelec. "Zigbee คืออะไร" http://www.thaieasyelec.com/electronics-in-chapter/what-is-zigbee.html

[7] ทิพวัลย์ คำน้ำนอง. ไมโครคอนโทรลเลอร์และการประยุกต์ใช้งานpic. พิมพ์ครั้งที่ 4 : สำนักพิมพ์ ส.ส.ท.(สมาคมเทคโนโลยี ไทย-ญี่ปุ่น), 2553.

ภาคผนวก ก

PIC Wireless Thermometer

```
#include <18F4520.h>
#fuses INTRC_IO,NOWDT,PUT,BROWNOUT
#use delay(clock=16Mhz)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7,stop=2)
//=======================================
#include <sht_15.c>
char C;
#int_rda
void rs232_isr(void)
{
C = getch();
}

//////////////////////////Main////////////////////////////
void main (void)
{
float temp=0,humi=0;
int16  memhumid=0;
enable_interrupts(GLOBAL);
enable_interrupts(INT_RDA);
C = 0;
sht_init();
```

```
while(1)
{

temp=((measuretemp()*0.01)-39.6); // converttemperature T=d2*temp+d1 14bit
memhumid=measurehumid();
humi=((0.0405*memhumid)-(0.0000028*memhumid*memhumid)-4.0+((temp-
25.0)*(0.01+(0.00008*memhumid))));

if(C!=0)
{
switch(C)
{
case 'C': printf("Temperature=%2.1f,Humidity=%2.1f\n",temp,humi); break;
}
C = 0;
}
}
}
```

ภาคผนวก ข

SHT-15

```
#define sht_data_pin PIN_B4

#define sht_clk_pin PIN_B5

//*****Function to alert SHT15*****

void comstart(void)

{

output_float(sht_data_pin); // data high

output_bit(sht_clk_pin,0); // clk low

delay_us(1);

output_bit(sht_clk_pin, 1); // clk high

delay_us(1);

output_bit(sht_data_pin,0); // data low

delay_us(1);

output_bit(sht_clk_pin, 0); // clk low

delay_us(2);

output_bit(sht_clk_pin, 1); // clk high

delay_us(1);

output_float(sht_data_pin); // data high

delay_us(1);

output_bit(sht_clk_pin, 0); // clk low

}
```

```
//*****Function to write data to SHT15*****
int1 comwrite (int8 iobyte)
{
int8 i, mask = 0x80;
int1 ack;
// shift out command
delay_us(4);
for(i=0;i<8;i++)
{
output_bit(sht_clk_pin, 0);                          // clk low
if((iobyte&mask)>0)output_float(sht_data_pin);       //data high if MSB high
else output_bit(sht_data_pin,0);                     // data low if  MSB low
delay_us(1);
output_bit(sht_clk_pin, 1);                          // clk high
delay_us(1);
mask = mask>>1;                                      // shift to next bit
}
//shift in ack

output_bit(sht_clk_pin,0);                           // clk low
delay_us(1);

ack = input(sht_data_pin);                           // get ack bit
```

```
output_bit(sht_clk_pin, 1);                    // clk high
delay_us(1);
output_bit(sht_clk_pin,0);                     // clk low
return(ack);
}
//*****function to read data from SHT15*****
int16 comread(void)
{
int8 i;
int16 iobyte = 0;
const int16 mask0= 0x0000;
const int16 mask1 = 0x0001;

//shift in MSB data
for(i=0;i<8;i++)
{
iobyte = iobyte <<1;
output_bit(sht_clk_pin, 1); // clk high
delay_us(1);
if(input(sht_data_pin)) iobyte|= mask1;   //shift  in data bit
else iobyte |= mask0;
```

```
 output_bit(sht_clk_pin,0);                    // clk low
 delay_us(1);
 }

 // send ack 0 bit
 output_bit(sht_data_pin,0);                   // data low
 delay_us(1);
 output_bit(sht_clk_pin, 1);                   // clk high
 delay_us(2);
 output_bit(sht_clk_pin,0);                    // clk low
 delay_us(1);
 output_float(sht_data_pin);                   // data high
 delay_us(1);

 //shift in LSB data
 for(i=0;i<8;i++)
 {
 iobyte = iobyte << 1;
 output_bit(sht_clk_pin, 1);                   // clk high
 delay_us(1);

 if(input(sht_data_pin)) iobyte|= mask1;   //shift  in data bit
 else iobyte |= mask0;
```

```
 output_bit(sht_clk_pin,0);                    // clk low
 delay_us(1);
 }

 // send ack 1 bit
 output_float(sht_data_pin);                   // data high
 delay_us(1);
 output_bit(sht_clk_pin, 1);                   // clk high
 delay_us(2);
 output_bit(sht_clk_pin,0);                    // clk low
 return(iobyte);
 }

//*****function to wait for SHT15 reading******
void comwait(void)
{
int16 sht_delay;
output_float(sht_data_pin);                    // data high
output_bit(sht_clk_pin,0);                     // clk low
delay_us(1);
```

```
for(sht_delay=0;sht_delay<30000; sht_delay++)       // wait for max 300ms
{
if(!input(sht_data_pin))break; // if sht_data_pin low, SHT15 ready
delay_us(10);
 }
}

// function to reset SHT15 communication *****
void comreset (void)
{
int8 i;
output_float(sht_data_pin);                          // data high
output_bit(sht_clk_pin,0);                           // clk low
delay_us(2);

for(i=0;i<9;i++)
{
output_bit(sht_clk_pin, 1);                          //toggle clk 9 times
delay_us(2);
output_bit(sht_clk_pin, 0);
delay_us(2);
}
```

```
comstart();
}

// function to soft reset SHT15
void sht_soft_reset(void)
{
comreset();                          //SHT communication reset
comwrite(0x1e);                      //sent SHT15 reset command
delay_ms(15);                        //pause 15 ms
}

//*******function to measure SHT15 temperature *****
int16 measuretemp(void)
{
int1 ack;
int16 iobyte;

comstart();                          // alert SHT15
ack = comwrite(0x03);                // send mesure temp command and
read ack status
```

```
if(ack ==1 ) return;

comwait();                          //wait for SHT15 measurement to
complete

iobyte = comread();                 //read SHT15 temp data

return(iobyte);

}


//****function to measure SHT15 RH*****

int16 measurehumid(void)

{

int1 ack;

int16 iobyte;


comstart();                         // alert SHT15

ack = comwrite(0x05);               // send mesure RH command and read
ack status


if(ack ==1 ) return;


comwait();                          //wait for SHT15 measurement to
complete

iobyte = comread();                 //read SHT15 RH data

return(iobyte);

}
```

```
void sht_init(void)
{
comreset();                    // resetSHT15
}
```

ภาคผนวก ค

โปรแกรม GRAPHICAL USER INTERFACE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {

        public Form1()
        {

            InitializeComponent();
            if (!mySerialPort.IsOpen)
            {
                mySerialPort.Open();
                tbRX.Text = "port opened :) ";
            }
```

```
        else
            tbRX.Text = "port busy :( ";
        this.KeyPreview = true;
        this.KeyPress +=
            new KeyPressEventHandler(Form1_KeyPress);

    }

    private string rxString;
    private void mySerialPort_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
    {
        rxString = mySerialPort.ReadExisting();
        this.Invoke(new EventHandler(displayText));
    }
    private void displayText(object o, EventArgs e)
    {
        tbRX.AppendText(rxString);
    }
    private void bSend_Click(object sender, EventArgs e)
    {
        mySerialPort.Write(tbTX.Text);
```

```
}
private void bClear_Click(object sender, EventArgs e)
{
    tbTX.Clear();
    tbRX.Clear();
}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    mySerialPort.Close();
}
private void button1_Click(object sender, EventArgs e)
{
   mySerialPort.Write("W");

}

private void button2_Click(object sender, EventArgs e)
{
    mySerialPort.Write("B");
}

private void button3_Click(object sender, EventArgs e)
```

```
{
    mySerialPort.Write("R");
}

private void button4_Click(object sender, EventArgs e)
{
    mySerialPort.Write("L");
}

private void button6_Click(object sender, EventArgs e)
{
    mySerialPort.Write("H");
    textBox3.Text = "Speed Hight";
}

private void button7_Click(object sender, EventArgs e)
{
    mySerialPort.Write("M");
    textBox3.Text = "Speed Medium";
}

private void button8_Click(object sender, EventArgs e)
{
```

```
        mySerialPort.Write("S");
        textBox3.Text = "Speed Low";
    }
    private void button10_Click(object sender, EventArgs e)
    {
        mySerialPort.Write("T");
        textBox3.Text = "Stable";
    }
    private void button5_Click(object sender, EventArgs e)
    {
        timer1.Enabled = true;
        mySerialPort.Write("C");
    }

    private void timer1_Tick(object sender, EventArgs e)
    {

        mySerialPort.Write("C");
    }

    private void button9_Click(object sender, EventArgs e)
    {
        timer1.Enabled = false;
```

```
        }

        private void Form1_KeyPress(object sender, KeyPressEventArgs e)
        {
            switch (e.KeyChar) //case :
            {
                case 'w': mySerialPort.Write("W"); e.KeyChar = 'z'; break;
                case 's': mySerialPort.Write("S"); e.KeyChar = 'z'; break;
                case 'a': mySerialPort.Write("A"); e.KeyChar = 'z'; break;
                case 'd': mySerialPort.Write("D"); e.KeyChar = 'z'; break;
                default: break;
            }
        }

    }
}
```

ภาคผนวก ง

Control Copter

```
#include <copter.h>
#include <math.h>
#include <mpu.h>
#use I2C(master,sda=PIN_C4,scl=PIN_C3)
#use rs232(baud=4800,UART2,stop=2)
#include <lcd16x2.c>
//////////////PWM//////////////
unsigned int16 fall1=0;    //time falling ccp1
unsigned int16 fall2=0;    //time falling ccp2
unsigned int16 fall3=0;    //time falling ccp3
unsigned int16 fall4=0;    //time falling ccp4

void set_duty1(float duty)
{
   fall1=((duty*65536)/100);
}
void set_duty2(float duty)
{
   fall2=((duty*65536)/100);
}
void set_duty3(float duty)
{

   fall3=((duty*65536)/100);
}
void set_duty4(float duty)
{

   fall4=((duty*65536)/100);
}

#INT_TIMER1
void rising(void)
{
  if(fall1>0){
  output_high(PIN_C2);}
  if(fall2>0){
  output_high(PIN_C6);}
  if(fall3>0){
  output_high(PIN_C7);}
  if(fall4>0){
```

```
    output_high(PIN_B5);}
}

#INT_CCP2
void CCP2_isr()
{
   output_low(PIN_C2);
}
#INT_CCP3
void CCP3_isr()
{
   output_low(PIN_C6);
}
#INT_CCP4
void CCP4_isr()
{
   output_low(PIN_C7);
}
#INT_CCP5
void CCP5_isr()
{
   output_low(PIN_B5);
}
```

```
//////END  PWM//////////

////////////////////////////////
char C;
#int_rda2
void rs232_isr(void){
C = getch();
}

//////////////////////////////////////
//////////FUNCTION  I2C//////////
int write(unsigned char Control,
unsigned char data)
{
   i2c_start();
   i2c_write(MPU6050_ADDRESS);
   i2c_write(control);
   i2c_write(data);
   i2c_stop();
   return 0;
}
int read(unsigned char control,
unsigned char *data)
```

```
{
   i2c_start();
   i2c_write(MPU6050_ADDRESS);
   i2c_write(control);
   i2c_start();
   i2c_write(MPU6050_ADDRESS |
0x01);
   *data=i2c_read(0);
   i2c_stop();
   return 0;
}
/////END FUNCTION I2C////////
```

```
//////////CONFIG MPU//////////////
void setupMPU()
{
      //Sets sample rate to
1000/1+1 = 500Hz
  write(MPU6050_RA_SMPLRT_DIV,
0x01);
  //Disable FSync, 48Hz DLPF
  write(MPU6050_RA_CONFIG, 0x03);
  //Disable gyro self tests, scale of
500 degrees/s
  write(MPU6050_RA_GYRO_CONFIG,
0b00001000);
  //Disable accel self tests, scale of
+-4g, no DHPF

write(MPU6050_RA_ACCEL_CONFIG,
0b00001000);

  //Freefall threshold of <|0mg|
  write(MPU6050_RA_FF_THR, 0x00);
  //Freefall duration limit of 0
  write(MPU6050_RA_FF_DUR, 0x00);
  //Motion threshold of >0mg
  write(MPU6050_RA_MOT_THR,
0x00);
  //Motion duration of >0s
  write(MPU6050_RA_MOT_DUR,
0x00);
  //Zero motion threshold
  write(MPU6050_RA_ZRMOT_THR,
0x00);
  //Zero motion duration threshold
```

```
  write(MPU6050_RA_ZRMOT_DUR,
0x00);
  //Disable sensor output to FIFO
buffer
  write(MPU6050_RA_FIFO_EN,
0x00);

  //AUX I2C setup
  //Sets AUX I2C to single master
control, plus other config

write(MPU6050_RA_I2C_MST_CTRL,
0x00);
  //Setup AUX I2C slaves

write(MPU6050_RA_I2C_SLV0_ADDR,
0x00);
  write(MPU6050_RA_I2C_SLV0_REG,
0x00);

write(MPU6050_RA_I2C_SLV0_CTRL,
0x00);

write(MPU6050_RA_I2C_SLV1_ADDR,
0x00);
  write(MPU6050_RA_I2C_SLV1_REG,
0x00);

write(MPU6050_RA_I2C_SLV1_CTRL,
0x00);

write(MPU6050_RA_I2C_SLV2_ADDR,
0x00);
  write(MPU6050_RA_I2C_SLV2_REG,
0x00);

write(MPU6050_RA_I2C_SLV2_CTRL,
0x00);

write(MPU6050_RA_I2C_SLV3_ADDR,
0x00);
  write(MPU6050_RA_I2C_SLV3_REG,
0x00);

write(MPU6050_RA_I2C_SLV3_CTRL,
0x00);

write(MPU6050_RA_I2C_SLV4_ADDR,
0x00);
  write(MPU6050_RA_I2C_SLV4_REG,
0x00);
  write(MPU6050_RA_I2C_SLV4_DO,
0x00);
```

```
write(MPU6050_RA_I2C_SLV4_CTRL,
0x00);

   write(MPU6050_RA_I2C_SLV4_DI,
0x00);

   //MPU6050_RA_I2C_MST_STATUS
//Read-only

   //Setup INT pin and AUX I2C pass
through

  write(MPU6050_RA_INT_PIN_CFG,
0x00);

  //Enable data ready interrupt

  write(MPU6050_RA_INT_ENABLE,
0x00);



  //MPU6050_RA_DMP_INT_STATUS
//Read-only

  //MPU6050_RA_INT_STATUS 3A
//Read-only

   //MPU6050_RA_ACCEL_XOUT_H
//Read-only

   //MPU6050_RA_ACCEL_XOUT_L
//Read-only

   //MPU6050_RA_ACCEL_YOUT_H
//Read-only

   //MPU6050_RA_ACCEL_YOUT_L
//Read-only

   //MPU6050_RA_ACCEL_ZOUT_H
//Read-only

   //MPU6050_RA_ACCEL_ZOUT_L
//Read-only

   //MPU6050_RA_TEMP_OUT_H
//Read-only

   //MPU6050_RA_TEMP_OUT_L
//Read-only

   //MPU6050_RA_GYRO_XOUT_H
//Read-only

   //MPU6050_RA_GYRO_XOUT_L
//Read-only

   //MPU6050_RA_GYRO_YOUT_H
//Read-only

   //MPU6050_RA_GYRO_YOUT_L
//Read-only

   //MPU6050_RA_GYRO_ZOUT_H
//Read-only

   //MPU6050_RA_GYRO_ZOUT_L
//Read-only


//MPU6050_RA_EXT_SENS_DATA_00
//Read-only


//MPU6050_RA_EXT_SENS_DATA_01
//Read-only
```

```
//MPU6050_RA_EXT_SENS_DATA_02
//Read-only

//MPU6050_RA_EXT_SENS_DATA_03
//Read-only

//MPU6050_RA_EXT_SENS_DATA_04
//Read-only

//MPU6050_RA_EXT_SENS_DATA_05
//Read-only

//MPU6050_RA_EXT_SENS_DATA_06
//Read-only

//MPU6050_RA_EXT_SENS_DATA_07
//Read-only

//MPU6050_RA_EXT_SENS_DATA_08
//Read-only

//MPU6050_RA_EXT_SENS_DATA_09
//Read-only

//MPU6050_RA_EXT_SENS_DATA_10
//Read-only

//MPU6050_RA_EXT_SENS_DATA_11
//Read-only

//MPU6050_RA_EXT_SENS_DATA_12
//Read-only

//MPU6050_RA_EXT_SENS_DATA_13
//Read-only

//MPU6050_RA_EXT_SENS_DATA_14
//Read-only

//MPU6050_RA_EXT_SENS_DATA_15
//Read-only

//MPU6050_RA_EXT_SENS_DATA_16
//Read-only

//MPU6050_RA_EXT_SENS_DATA_17
//Read-only

//MPU6050_RA_EXT_SENS_DATA_18
//Read-only

//MPU6050_RA_EXT_SENS_DATA_19
//Read-only
```

```
//MPU6050_RA_EXT_SENS_DATA_20
//Read-only

//MPU6050_RA_EXT_SENS_DATA_21
//Read-only

//MPU6050_RA_EXT_SENS_DATA_22
//Read-only

//MPU6050_RA_EXT_SENS_DATA_23
//Read-only

//MPU6050_RA_MOT_DETECT_STAT
US   //Read-only

  //Slave out, dont care
  write(MPU6050_RA_I2C_SLV0_DO,
0x00);
  write(MPU6050_RA_I2C_SLV1_DO,
0x00);
  write(MPU6050_RA_I2C_SLV2_DO,
0x00);
  write(MPU6050_RA_I2C_SLV3_DO,
0x00);
  //More slave config
  write(MPU6050_RA_I2C_MST_DELAY_
CTRL, 0x00);
  //Reset sensor signal paths

write(MPU6050_RA_SIGNAL_PATH_RE
SET, 0x00);
  //Motion detection control

write(MPU6050_RA_MOT_DETECT_C
TRL, 0x00);
  //Disables FIFO, AUX I2C, FIFO and
I2C reset bits to 0
  write(MPU6050_RA_USER_CTRL,
0x00);
  //Sets clock source to gyro
reference w/ PLL
  write(MPU6050_RA_PWR_MGMT_1,
0b00000010);
  //Controls frequency of wakeups
in accel low power mode plus the
sensor standby modes
  write(MPU6050_RA_PWR_MGMT_2,
0x00);
  //MPU6050_RA_BANK_SEL
//Not in datasheet
```

```
//MPU6050_RA_MEM_START_ADDR
//Not in datasheet

   //MPU6050_RA_MEM_R_W
//Not in datasheet

   //MPU6050_RA_DMP_CFG_1
//Not in datasheet

   //MPU6050_RA_DMP_CFG_2
//Not in datasheet

   //MPU6050_RA_FIFO_COUNTH
//Read-only

   //MPU6050_RA_FIFO_COUNTL
//Read-only

  //Data transfer to and from the
FIFO buffer

  write(MPU6050_RA_FIFO_R_W,
0x00);

   //MPU6050_RA_WHO_AM_I
//Read-only, I2C address

}

//////////////////////////////////////////


////////KALMAN////////////

signed int16 ACCEL_XOUT;

signed int16 ACCEL_YOUT;

signed int16 ACCEL_ZOUT;

float ACCEL_FXOUT;

float ACCEL_FYOUT;

float ACCEL_FZOUT;



signed int16 GYRO_XOUT;

signed int16 GYRO_YOUT;

signed int16 GYRO_ZOUT;

float GYRO_FXOUT;

float GYRO_FYOUT;

float GYRO_FZOUT;

float GYRO_OUT;

float Axz=0;

float Ayz=0;

float RxAcc=0;

float RyAcc=0;

float RzAcc=0;

float RxEst=0;

float RyEst=0;

float RzEst=0;

float RzGyro;

#define w 15   //weight KALMAN
```

```
void Get_Accel_Values()
{

int16 ACCEL_XOUT_H=0;
int16 ACCEL_XOUT_L=0;
int16 ACCEL_YOUT_H=0;
int16 ACCEL_YOUT_L=0;
int16 ACCEL_ZOUT_H=0;
int16 ACCEL_ZOUT_L=0;
float r;

read(MPU6050_RA_ACCEL_XOUT_H,
&ACCEL_XOUT_H);

read(MPU6050_RA_ACCEL_XOUT_L,
&ACCEL_XOUT_L);

read(MPU6050_RA_ACCEL_YOUT_H,
&ACCEL_YOUT_H);

read(MPU6050_RA_ACCEL_YOUT_L,
&ACCEL_YOUT_L);

read(MPU6050_RA_ACCEL_ZOUT_H,
&ACCEL_ZOUT_H);

read(MPU6050_RA_ACCEL_ZOUT_L,
&ACCEL_ZOUT_L);

  ACCEL_XOUT =
~(((ACCEL_XOUT_H<<8)|ACCEL_XOU
T_L)-1);

  ACCEL_YOUT =
~(((ACCEL_YOUT_H<<8)|ACCEL_YOU
T_L)-1);

  ACCEL_ZOUT =
~(((ACCEL_ZOUT_H<<8)|ACCEL_ZOU
T_L)-1);

ACCEL_FXOUT=(float)ACCEL_XOUT/
8192;

ACCEL_FYOUT=(float)ACCEL_YOUT/8
192;

ACCEL_FZOUT=(float)ACCEL_ZOUT/
8192;

r=sqrt((ACCEL_FXOUT*ACCEL_FXOU
T)+(ACCEL_FYOUT*ACCEL_FYOUT)+(
ACCEL_FZOUT*ACCEL_FZOUT));

  RxAcc=ACCEL_FXOUT/r;
```

```
   RyAcc=ACCEL_FYOUT/r;
   RzAcc=ACCEL_FZOUT/r;
}
void Get_Gyro_Values()
{

int16 GYRO_XOUT_H=0;
int16 GYRO_XOUT_L=0;
int16 GYRO_YOUT_H=0;
int16 GYRO_YOUT_L=0;
int16 GYRO_ZOUT_H=0;
int16 GYRO_ZOUT_L=0;

read(MPU6050_RA_GYRO_XOUT_H,
&GYRO_XOUT_H);
  read(MPU6050_RA_GYRO_XOUT_L,
&GYRO_XOUT_L);

read(MPU6050_RA_GYRO_YOUT_H,
&GYRO_YOUT_H);
  read(MPU6050_RA_GYRO_YOUT_L,
&GYRO_YOUT_L);

read(MPU6050_RA_GYRO_ZOUT_H,
&GYRO_ZOUT_H);
  read(MPU6050_RA_GYRO_ZOUT_L,
&GYRO_ZOUT_L);

  GYRO_XOUT =
~(((GYRO_XOUT_H<<8)|GYRO_XOUT
_L)-1);
  GYRO_YOUT =
~(((GYRO_YOUT_H<<8)|GYRO_YOUT
_L)-1);
  GYRO_ZOUT =
~(((GYRO_ZOUT_H<<8)|GYRO_ZOUT
_L)-1);

GYRO_FXOUT=(float)GYRO_XOUT/65
.5;

GYRO_FYOUT=(float)GYRO_YOUT/65
.5;

GYRO_FZOUT=(float)GYRO_ZOUT/65
.5;

GYRO_OUT=sqrt((GYRO_FXOUT*GYR
O_FXOUT)+(GYRO_FYOUT*GYRO_FY
```

```
OUT)+(GYRO_FZOUT*GYRO_FZOUT)
);

GYRO_FXOUT=GYRO_FXOUT/GYRO_
OUT;

GYRO_FYOUT=GYRO_FYOUT/GYRO_
OUT;

GYRO_FZOUT=GYRO_FZOUT/GYRO_
OUT;
}

void kalman()
{

float RxGyro,RyGyro;
float r=0;

  Axz=atan2(RxEst,RzEst);
  Ayz=atan2(RyEst,RzEst);
  Axz=Axz+GYRO_FYOUT*0.02;
  Ayz=Ayz+GYRO_FXOUT*0.02;

RxGyro=(sin(Axz))/(sqrt(1+(cos(Axz)*
cos(Axz)*tan(Ayz)*tan(Ayz))));
RyGyro=(sin(Ayz))/(sqrt(1+(cos(Ayz)*
cos(Ayz)*tan(Axz)*tan(AXZ))));
  if(RzEst<0){
    RzGyro=-1*(sqrt(1-
(RxGyro*RxGyro)-(RyGyro*RyGyro)));
    }
  else if(RzEst>0){
    RzGyro=sqrt(1-(RxGyro*RxGyro)-
(RyGyro*RyGyro));
    }
  RxEst=(RxAcc+(RxGyro*w))/(w+1);
  RyEst=(RyAcc+(RyGyro*w))/(w+1);
  RzEst=(RzAcc+(RzGyro*w))/(w+1);

r=sqrt((RxEst*RxEst)+(RyEst*RyEst)+(
RzEst*RzEst));
  RxEst=RxEst/r;
  RyEst=RyEst/r;
  RzEst=RzEst/r;
}
//////////////////////////////////////////

void main()
{
```

```
int de=0;          //delay for kalman
int dela=0;


int force=55;      //duty max of
motor
int16 degreeX=180;
int16 degreeY=180;
int start=24;      //duty start motor
int b=0;
float DAyz=0;
float DAxz=0;


////////////////PID////////////////////
float NAyz=0;
float cy=0;
float erY=0;
float pre_erY=0;       //precious-
error
float integralY=0;     //integral
float dvY=0;           //derivative


float NAxz=0;
float cx=0;
float erX=0;
float pre_erX=0;       //precious-
error
float integralX=0;     //integral
float dvX=0;           //derivative
////////////////////////////////////


/////////////SET PWM//////////////
  set_tris_c(0);
  set_tris_b(0);
  enable_interrupts(GLOBAL);
  enable_interrupts(INT_TIMER1);
  enable_interrupts(INT_CCP2);
  enable_interrupts(INT_CCP3);
  enable_interrupts(INT_CCP4);
  enable_interrupts(INT_CCP5);
  enable_interrupts(INT_RDA2);

setup_timer_1(T1_INTERNAL|T1_DIV
_BY_1);
  set_timer1(0);
  setup_ccp2(CCP_COMPARE_INT);
  setup_ccp3(CCP_COMPARE_INT);
```

```
    setup_ccp4(CCP_COMPARE_INT);
    setup_ccp5(CCP_COMPARE_INT);

    set_duty1(start+0.2);
    CCP_2=fall1;
    set_duty2(start+0.1);
    CCP_3=fall2;
    set_duty3(start-0.1);
    CCP_4=fall3;
    set_duty4(start-0.3);
    CCP_5=fall4;
    delay_ms(9000);

////////////////////////////////////////

C = 'y';

   ///////check MPU/////////////
   unsigned char data;
   lcd_init();
   read(MPU6050_RA_WHO_AM_I,
&data);
   lcd_gotoxy(1,1);
   printf(lcd_putc,"%x ",data);
   delay_ms(200);
   ////////////////////////////////////

   setupMPU();         //config MPU

   ///////////Begin
KALMAN//////////////
   Get_Accel_Values();
   RxAcc=RxEst;
   RyAcc=RyEst;
   RzAcc=RzEst;
   ////////////////////////////////

     while(true)
     {
       if(C!='y'){

              switch(C){
                     case 't':
degreeX=180;degreeY=180; break;

                     case 'a':
degreeX=170;degreeY=180; break;
```

```
                    case 'd':
degreeX=190;degreeY=180; break;
                    case 'w':
degreeY=190;degreeX=180; break;
                    case 's':
degreeY=170;degreeX=180; break;


                    case 'o':
force=60;  break;
                    case 'j': force=65;
break;
                    case 'l':
force=70; break;


                    case 'x': start=24;
                              b=9;
                              break;
                    case 'm': b=0;

        pre_erY=0;

        integralY=0;
                              dvY=0;

        pre_erX=0;
        integralX=0;
                              dvX=0;
                              break;
                    }

                }


        Get_Gyro_Values();
        Get_Accel_Values();
        kalman();


        DAyz=Ayz*57.295;
        DAxz=Axz*57.295;
        if(dela>15)
        {
          lcd_gotoxy(1,1);
          printf(lcd_putc,"%f %ld %c
\n%f %f",cx,degreeX,C,NAxz,NAyz);
          dela=0;
        }
       dela++;
```

```
if((start<force)&&(b==0))
{
 set_duty1(start+0.2);
 CCP_2=fall1;
 set_duty2(start+0.1);
 CCP_3=fall2;
 set_duty3(start-0.1);
 CCP_4=fall3;
 set_duty4(start-0.3);
 CCP_5=fall4;
 start++;
 delay_ms(700);

 }
if((start>=force)&&(b==0))
{

 if(de>=80)
 {
   b=1;
   delay_ms(1000);
 }
 de++;
}

if(b==1)
{
 set_duty1(force+0.1);
 CCP_2=fall1;
 set_duty2(force+0.1);
 CCP_3=fall2;
 set_duty3(force-0.1);
 CCP_4=fall3;
 set_duty4(force-0.3);
 CCP_5=fall4;
 b=2;
}
if(b==2)
{

 ///////set new angle///////
 if(DAyz<0)
 {
   NAyz=DAyz*(-1);
```

```
        }
        else if(DAyz>=0)
        {
           NAyz=360-DAyz;
        }
        if(DAxz<0)
        {
           NAxz=DAxz*(-1);
        }
        else if(DAxz>=0)
        {
           NAxz=360-DAxz;
        }

////////////////////////////////////

/////////////PID/////////////////////
        erX=degreeX-NAxz;
        integralX=integralX+(erX*0.1);
        dvX=(erX-pre_erX)/0.1;

cx=(0.007*erX)+(0.005*integralX)+(0.
04*dvX);
        pre_erX=erX;     //update

        erY=degreeY-NAyz;
        integralY=integralY+(erY*0.1);
        dvY=(erY-pre_erY)/0.1;

cy=(0.007*erY)+(0.005*integralY)+(0.
04*dvY);
        pre_erY=erY;     //update

////////////////////////////////////

          if(cy>=0)//m3
        {
           set_duty1(force+cy+1);
           CCP_2=fall1;
        }
         if(cx>=0)//m2
        {
            set_duty2(force+cx);
            CCP_3=fall2;
        }else if(cy<0)//m4
        {
```

```
            set_duty3(force-cy-0.5);
            CCP_4=fall3;
        }
        else if(cx<0)//m1
        {
            set_duty4(force-cx-0.5);
            CCP_5=fall4;
        }
      }

    }
}
```