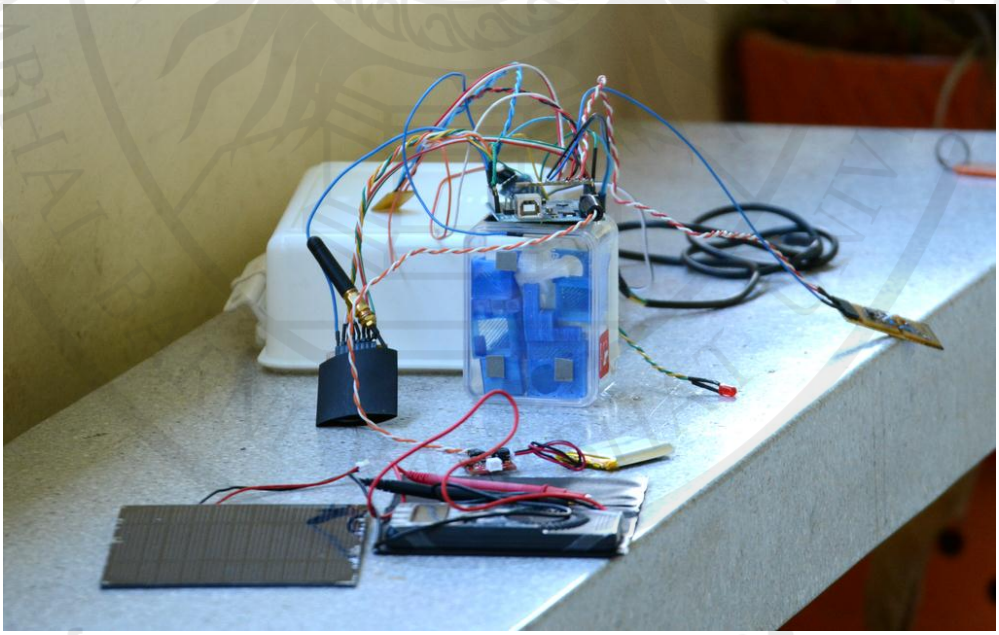
บทที่ 4
ผลการทดลองและวิเคราะห์ผลการทดลอง

ในบทที่ผ่านมาผู้วิจัยได้กล่าวถึงรายละเอียดการวิเคราะห์และออกแบบระบบของงานวิจัยนี้ ซึ่งผู้วิจัยได้กล่าวทั้งรายละเอียดด้านฮาร์ดแวร์และซอฟต์แวร์ดังนั้นเนื้อหาในบทนี้ผู้วิจัยจึงจะกล่าวถึงผลการดำเนินงานตามการออกแบบดังกล่าว นั่นคือผู้วิจัยจะอธิบายเกี่ยวกับรายละเอียดของการอิมพลิเม้นต์ขั้นตอนต่าง ๆ ของระบบ เช่น การชุดคำสั่งที่ใช้ในการสื่อสารด้วยโพรโทคอล SPI และไอสแควร์ซี รวมถึงการจัดการฐานข้อมูลและการแสดงผลข้อมูลที่บันทึกไว้สำหรับการวิเคราะห์และอภิปรายการทำงานของระบบจะอยู่ในส่วนท้ายของบทนี้ รายละเอียดต่างๆมีดังต่อไปนี้

4.1 โปรแกรมควบคุมการทำงานฝั่งลูกข่าย

งานวิจัยนี้ใช้บอร์ด Arduino UNO R2 ในการทำหน้าที่ควบคุมการทำงานของ SHT11, TPS852, LM393, DS18B20 และ CC1100 โดยภาพที่ ภาพที่ 4.1 แสดงตัวอย่างการเชื่อมต่อของอุปกรณ์ต่างๆ ที่ใช้ในการทดลองครั้งนี้



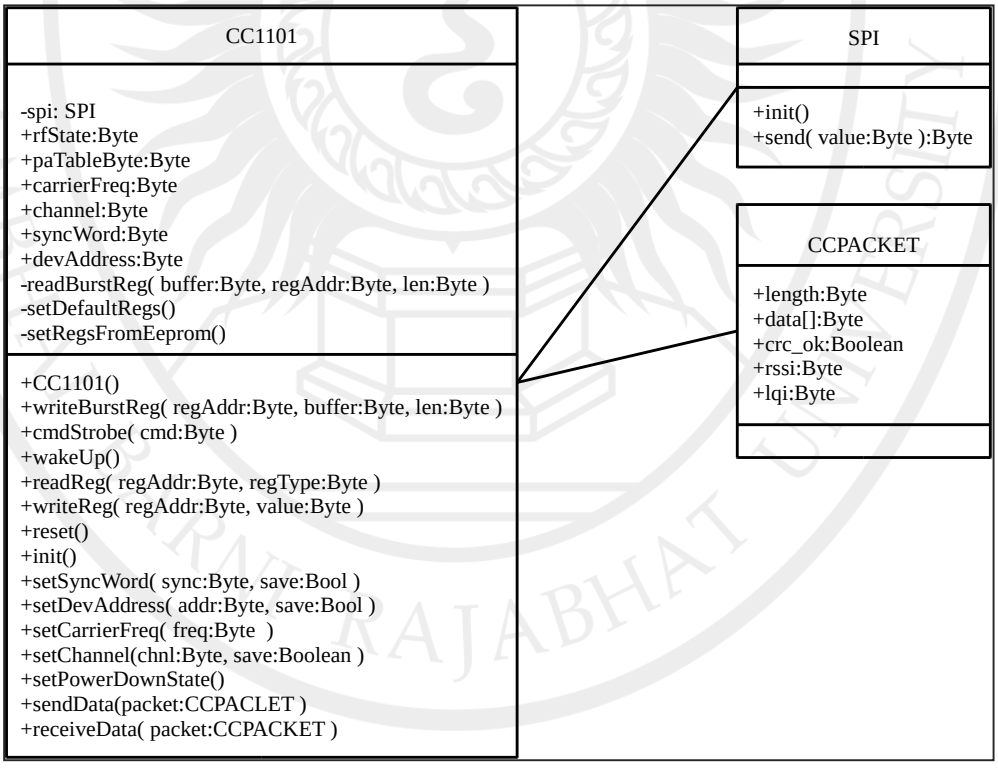
ภาพที่ 4.1 การเชื่อมต่ออุปกรณ์ที่ใช้ในการทดลองครั้งนี้ (ฝั่งลูกข่าย)

4.1.1 การทำงานของไมโครคอนโทลเลอร์

ภาพรวมเกี่ยวกับการออกแบบการสื่อสารระหว่างบอร์ดArduino กับเซนเซอร์ต่าง ๆ นั้น ผู้วิจัยได้อธิบายไว้อย่างคร่าว ๆ ในบทที่สามแล้ว เนื้อหาในส่วนนี้จะเป็นการอธิบายผลการอิมพลิเม้นต์ขั้นตอนการสื่อสารดังกล่าวบนฝั่งลูกข่าย ซึ่งผู้วิจัยได้พัฒนาด้วยการใช้Arduino IDE 1.0.1โดยเนื้อหาจะแยกอธิบายเป็นรายอุปกรณ์และจะยกเฉพาะเมธอดที่สำคัญมาอธิบาย ดังรายละเอียดต่อไปนี้

4.1.1.1 การสื่อสารกับ CC1100 ด้วยโพรโทคอล SPI

ไมโครคอนโทรเลอร์จะสื่อสารกับ CC1100 ด้วยโพรโทคอล SPI โดยงานวิจัยนี้ได้ใช้คลังคำสั่งสำหรับสื่อสารกับ CC1100 ของ panstamp[1] มาใช้ในการทำงานเป็นหลัก คลังคำสั่ง panstamp นี้มีคลาสสำหรับใช้ร่วมกับเซนเซอร์ต่าง ๆ หลายชนิด รวมถึงคลาส CC1101 ซึ่งเป็นตัวรับส่งวิทยุเช่นเดียวกับ CC1100 ที่ใช้ในการทดลองนี้ ซึ่งแม้ว่าจะเป็นคนละรุ่นแต่ ผู้วิจัยได้ทดสอบแล้วว่าสามารถใช้งานทดแทนการได้โดยไม่มีปัญหาแต่อย่างใด ภาพที่ 4.2 Illustrationแสดงคลาสไดอะแกรมของคลาส CC1101 ของ panstamp



ภาพที่ 4.2 คลาสไดอะแกรมของ CC1101

1 http://www.panstamp.com/

จากภาพจะเห็นว่าผู้พัฒนาคลาส CC1101 ได้เตรียมเมธอดที่สำคัญสำหรับการใช้งานตัวรับส่งวิทยุให้เรียบร้อยแล้ว ทั้งเมธอดสำหรับการเริ่มต้นใช้งาน การกำหนดความถี่ รวมไปถึงเมธอดสำหรับการอ่านและเขียนเรจิสเตอร์ จากไดอะแกรมจะเห็นว่าคลาสนี้จัดเก็บข้อมูลที่ผู้ใช้ต้องการสื่อสารและข้อมูลประกอบอื่น ๆ ที่จำเป็นต้องใช้ในการตรวจสอบการสื่อสารครั้งนั้น ๆ ไว้เป็นวัตถุของคลาส CCPACKET (เป็นส่วนหนึ่งของคลังคำสั่ง panstamp เช่นเดียวกัน) นอกจากนั้นแล้วคลาส CC1101 นี้ยังได้เรียกใช้งานคลาส SPI ในการทำหน้าที่ส่งและรับข้อมูลด้วยโพรโทคอล SPI รายละเอียดของการอิมพลิเม้นต์เมธอดที่สำคัญที่เกี่ยวข้องกับงานวิจัยนี้มีดังต่อไปนี้

1) การเตรียมการใช้งาน

การเริ่มต้นใช้งาน CC1101ทำได้ได้ด้วยการเรียกใช้เมธอด init() โดยไม่ต้องมีอาร์กิวเมนต์ เมธอดนี้จะเตรียมความพร้อมการสื่อสารด้วยโพรโทคอล SPI ด้วยการเรียกเมธอด SPI::init() ซึ่งเมธอดนี้จะไปเตรียมขาสัญญาณทั้งสี่ของโพรโทคอล SPI ให้อยู่ในโหมดที่ถูกต้อง นั่นคือจะไปกำหนดให้ขา SS, MOSI และ SCK เป็นขาสำหรับเอ๊าต์พุตและให้ขา MISO เป็นอินพุต หลังจากนั้นจะเขียนข้อมูลลงขา SCK และ MOSI ด้วยค่า HIGHT และ LOW ตามลำดับ เสร็จแล้วจึงกำหนดความเร็วของการสื่อสารด้วยการ กำหนดค่าตัวแปรของไมโครคอนโทรเลอร์ โดยในที่นี้จะกำหนดให้ความเร็วของการสื่อสารมีค่าเป็น clk/4 หลังจากผ่านขั้นตอนนี้แล้วเมธอด init() จะเตรียมขาสัญญาณ GDO0 (ซึ่งเป็นขาของ GPIO CC1100) ให้เป็นขาอินพุตเพื่อใช้เป็นตัวขัดจังหวะการทำงาน (ทั้งนี้งานวิจัยนี้ไม่ได้ใช้การขัดจังหวะการทำงานผ่าน GDO0 นี้แต่อย่างใด) เมื่อเตรียมการต่าง ๆ เหล่านี้เรียบร้อยแล้วก็จะทำการรีเซตการทำงานของCC1100 ด้วยการเรียกเมธอด CC1100::reset () ตามรายละเอียดในชุดคำสั่งที่ 4. 1

```
void CC1101::reset(void)
{
  cc1101_Deselect();
  delayMicroseconds(5);
  cc1101_Select();
  delayMicroseconds(10);
  cc1101_Deselect();
  delayMicroseconds(41);
  cc1101_Select();
  wait_Miso();
```

```
  spi.send(CC1101_SRES);
  wait_Miso();
  cc1101_Deselect();
  setDefaultRegs();
  setReg
```

ชุดคำสั่งที่ 4.1 เมธอดreset ของคลาส CC1101

เมธอด reset()คือเมธอดหลักของคลาส CC1101 ที่จะใช้กำหนดค่าพารามิเตอร์ต่าง ๆ ของอุปกรณ์ให้พร้อมทำงานก่อนเริ่มทำงานจริง โดยไมโครคอนโทรเลอร์จะส่งสัญญาณการเลือก (select) สลับกับไม่เลือก (deselect) อุปกรณ์สองครั้งไปให้อุปกรณ์เพื่อบอกให้อุปกรณ์รู้ว่าผู้ใช้ต้องการรีเซต หลังจากนั้นจึงรอสัญญาณตอบรับจากอุปกรณ์ เมื่อพร้อมทำงานไมโครคอนโทรเลอร์จะส่งคำสั่งสั่งงานรีเซต (ค่าคงที่ CC1101_SRESมีค่าเท่ากับ 0x30) ไปให้อุปกรณ์แล้วรอการตอบรับ เมื่ออุปกรณ์รับทราบคำสั่งและพร้อมทำงานต่อแล้วไมโครคอนโทรเลอร์จึงจะทำการรีเซตอุปกรณ์จริง ๆ ด้วยการเรียกเมธอด setDefaultRegs และ setRegsFromEeprom ตามลำดับ

เมธอด setDefaultRegs จะทำหน้าที่เขียนข้อมูลพารามิเตอร์ต่าง ๆ เช่น ความยาวข้อมูล, รหัสเวลาพร้อมสัมพันธ์ (sync word) และความถี่วิทยุลงไปในเรจิสเตอร์ด้วยการเรียกเมธอด writeRegs ซึ่งพารามิเตอร์ที่ใช้ในงานวิจัยนี้แสดงอยู่ในตารางที่ 4.1 ในทำนองเดียวกันเมธอด setRegsFromEeprom จะเขียนค่าเรจิสเตอร์โดยใช้ข้อมูลจากอีอีพร็อมของ CC1100 โดยเมธอดนี้จะอ่านข้อมูลช่องสัญญาณวิทยุ (RF channel), รหัสเวลาพร้อมสัมพันธ์และเลขที่อยู่จากอีอี พร็อมของ CC1100 หลังจากนั้นแล้วจึงเขียนค่านี้ลงในเรจิสเตอร์ การอ่านข้อมูลจากอีอีพร็อมนี้ทำโดยการ

เรียกใช้เมธอด read ของคลาส EEPROM (ซึ่งเป็นส่วนหนึ่งของคลังคำสั่ง panstamp ด้วยเช่นเดียวกัน)

ตารางที่ 4.1 ค่าพารามิเตอร์ที่ใช้ในการเริ่มใช้งาน CC1100

ชื่อพารามิเตอร์	ค่าที่กำหนด	
CC1101_IOCFG2	0x2E	ขาเอ๊าต์พุตสำหรับGDO2
CC1101_IOCFG1	0x2E	ขาเอ๊าต์พุตสำหรับGDO1
CC1101_IOCFG0	0x06	ขาเอ๊าต์พุตสำหรับ GDO0
CC1101_FIFOTHR	0x07	ค่าขีดแบ่งสำหรับ RX FIFO และ TX FIFO
CC1101_PKTLEN	0x3D	ความยาวของกลุ่มข้อมูล
CC1101_PKTCTRL1	0x06	ตัวควบคุมกลุ่มข้อมูล
CC1101_PKTCTRL0	0x05	ตัวควบคุมกลุ่มข้อมูล
CC1101_FSCTRL1	0x08	
CC1101_FSCTRL0	0x00	
CC1101_MDMCFG4	0xCA	
CC1101_MDMCFG3	0x83	
CC1101_MDMCFG2	0x93	
CC1101_MDMCFG1	0x22	
CC1101_MDMCFG0	0xF8	
CC1101_DEVIATN	0x35	
CC1101_MCSM2	0x07	
CC1101_MCSM1	0x20	
CC1101_MCSM0	0x18	
CC1101_FOCCFG	0x16	
CC1101_BSCFG	0x6C	
CC1101_AGCCTRL2	0x43	
CC1101_AGCCTRL1	0x40	
CC1101_AGCCTRL0	0x91	

ตารางที่ 4.1 ค่าพารามิเตอร์ที่ใช้ในการเริ่มใช้งาน CC1100 ต่อ

CC1101_WOREVT1	0x87	
CC1101_WOREVT0	0x6B	
CC1101_WORCTRL	0xFB	
CC1101_FREND1	0x56	
CC1101_FREND0	0x10	
CC1101_FSCAL3	0xE9	
CC1101_FSCAL2	0x2A	
CC1101_FSCAL1	0x00	
CC1101_FSCAL0	0x1F	
CC1101_RCCTRL1	0x41	
CC1101_RCCTRL0	0x00	
CC1101_FSTEST	0x59	
CC1101_PTEST	0x7F	
CC1101_AGCTEST	0x3F	
CC1101_TEST2	0x81	
CC1101_TEST1	0x35	
CC1101_TEST0	0x09	

2) การอ่านข้อมูลจากเรจิสเตอร์

ไมโครคอนโทรเลอร์สามารถอ่านข้อมูลจากเรจิสเตอร์ได้ด้วยการเรียกเมธอด readReg โดยจะต้องส่งอาร์กิวเมนต์ไปสองค่า ค่าแรกคือเลขที่อยู่ของเรจิสเตอร์ที่ต้องการอ่านข้อมูล และอาร์กิวแมนต์ที่สองคือชนิดของเรจิสเตอร์ โดยชนิดของเรจิสเตอร์นี้จะเป็นไปได้สองชนิดคือ เรจิสเตอร์สำหรับการกำหนดค่า (config register) และเรจิสเตอร์สำหรับอ่านสถานของอุปกรณ์ (status register) ซึ่งจะมีค่าเป็น0x80และ 0x00 ตามลำดับโดยค่าทั้งสองนี้จะผ่านเข้ามาในเมธอดในรูปแบบของชนิดข้อมูลไบต์ ต่อมาจึงรวมข้อมูลทั้งสองไบต์นี้เข้าด้วยกันด้วยตัวดำเนินการตรรกะออร์ (logical OR) หลังจากรวมข้อมูลทั้งสองเข้าด้วยกันแล้วจึงส่งค่าที่ได้ไปให้เมธอด SPI::sendเพื่อส่งไปยัง CC1101 หลังจากนั้นจึงเรียกใช้ SPI::send พร้อมกับส่งอาร์กิวเม้นต์เป็น 0x00 เพื่อทำการอ่านค่าข้อมูลจากเรจิสเตอร์ เสร็จแล้วจึงยกเลิกการเลือกอุปกรณ์ ดังรายละเอียดในชุดคำสั่งที่ 4.2

```
byte CC1101::readReg(byte regAddr, byte regType)
{
  byte addr, val;
  addr = regAddr | regType;
  cc1101_Select();
  wait_Miso();
  spi.send(addr);
val = spi.send(0x00);
  cc1101_Deselect
   return val;
}
```

ชุดคำสั่งที่ 4.2 เมธอดอ่านข้อมูลจากเรจิสเตอร์

3) การเขียนข้อมูลลงเรจิสเตอร์

การเขียนข้อมูลลงเรจิสเตอร์ของ CC1100 ทำได้ด้วยการเรียกเมธอด writeReg โดยเมธอดนี้จะรับอาร์กิวเม้นต์สองค่า ค่าแรกคือเลขที่อยู่ของเรจิสเตอร์ที่ต้องการเขียนค่า และอาร์กิวเมนต์ที่สองคือค่าที่ต้องการเขียน ข้อมูลทั้งสองค่านี้จะส่งมาจากไมโครคอนโทรเลอร์เป็นข้อมูลไบต์ การทำงานของเมธอดจะเริ่มด้วยการเลือกอุปกรณ์แล้วรอให้อุปกรณ์พร้อมทำงาน ซึ่งทำได้โดยการเรียกเมธอด cc1101_select และ wait_Miso ตามลำดับ เมื่อไมโครคอนโทรเลอร์ตรวจสอบพบว่าอุปกรณ์พร้อมทำงานแล้วจึงจะส่งเลขที่อยู่ของเรจิสเตอร์ที่ต้องการบันทึกข้อมูลและค่าที่ต้องการบันทึกด้วยการเรียกเมธอด send สองครั้ง โดยการเรียกครั้งแรกจะเป็นการเรียกโดยส่งเลขที่อยู่ของเรจิสเตอร์เป็นอาร์กิวเม้นต์ และครั้งที่สองจะส่งไบต์ข้อมูลที่ต้องการเขียนเป็นอาร์กิวเม้นต์เมื่อส่งข้อมูลเรียบร้อยแล้วไมโครคอนโทรเลอร์จึงยกเลิกการใช้งาน CC1100 โดยการเรียกเมธอด cc1101_Deselect ดังรายละเอียดในชุดคำสั่งที่ 4.3

```
void CC1101::writeReg(byte regAddr, byte value)
{
  cc1101_Select();
  wait_Miso();
  spi.send(regAddr);
  spi.send(value);
  cc1101_Deselect();
}
```

ชุดคำสั่งที่ 4.3 เมธอดเขียนข้อมูลลงรีจิสเตอร์

4) การส่งคำสั่งงาน

ไมโครคอนโทรเลอร์จะส่งคำสั่งงาน ไปยัง CC1100 ด้วยการเรียกเมธอด cmdStrobe โดยมีอาร์กิวเม้นต์หนึ่งค่า ซึ่งก็คือคำสั่งงาน ที่ต้องการส่ง ซึ่งคำสั่งงาน นี้จะเป็นข้อมูลขนาดหนึ่งไบต์ โดยเมธอดนี้จะทำตามขั้นตอนการส่งคำสั่งงาน ที่ได้อธิบายไว้แล้วในบทที่สาม โดยจะเริ่มต้นการทำงานด้วยส่งสัญญาณการเลือกอุปกรณ์ด้วยการเรียกใช้เมธอด cc1101_Select เพื่อให้อุปกรณ์และไมโครคอนโทรเลอร์รู้ว่าการสื่อสารจะเกิดขึ้นระหว่างอุปกรณ์คู่ใด หลังจากนั้นไมโครคอนโทรเลอร์จะรอสัญญาณจากอุปกรณ์โดยการเรียกเมธอด wait_Miso ซึ่งจะวนรอบรอจนกว่าอุปกรณ์จะพร้อม เมื่อพบว่าอุปกรณ์พร้อมทำงานแล้วไมโครคอนโทรเลอร์จะเรียกเมธอด send ของคลาส SPI เพื่อดำเนินการส่งคำสั่ง และเมื่อสิ้นสุดการส่งข้อมูลแล้วจึงยกเลิกการเลือกอุปกรณ์ด้วยการเรียกเมธอด cc1101_Deselect ดังรายละเอียดในชุดคำสั่งที่ 4.4

```
void CC1101::cmdStrobe(byte cmd)
{
  cc1101_Select();
  wait_Miso();
  spi.send(cmd);
  cc1101_Deselect();
}
```

ชุดคำสั่งที่ 4.4 เมธอดส่งข้อความสั่ง

5) การส่งข้อมูล

เมธอด sendData จะทำหน้าที่ส่งข้อมูลออกไปเป็นสัญญาณวิทยุ โดยเมธอดนี้จะรับอาร์กิวเม้นต์หนึ่งค่าเป็นวัตถุของคลาส CCPACKET เมธอดนี้จะเริ่มต้นส่งข้อมูลด้วยการเปลี่ยนโหมดการทำงานของชิป CC1100 ให้มาอยู่ในโหมดส่งวิทยุ(RX)ซึ่งจะทำด้วยการเรียกใช้เมธอด setRxState หลังจากนั้นจึงวนรอบอ่านข้อมูลสถานะของชิปด้วยเมธอด readStatusReg พร้อมกับกำหนดสถานะที่ต้องการอ่านค่าเป็น MARCSTATE (หมายถึงสถานะของเครื่อง (machine state))เพื่อรอจนกว่าสถานะการทำงานของ CC1100 จะมีค่าเป็น 0x0D หลังจากนั้นจึงรอเป็นเวลาประมาณ 500 ไมโครวินาที เมื่อพร้อมแล้วจึงกำหนดจำนวนไบต์ของข้อมูลที่ต้องการส่ง โดยเรียกใช้เมธอด writeReg โดยอ่านค่าความยาวของแถวลำดับของตัวแปร packet (วัตถุของคลาส CCPACKET)เป็นอาร์กิวเม้นต์ หลังจากนั้นจึงเขียนข้อมูลที่ต้องการส่งไว้บนเรจิสเตอร์ด้วยการเรียกเมธอด writeBurstReg เมื่อเตรียมข้อมูลทุกอย่างพร้อมแล้วจึงสั่งให้ CC1100 สั่งสัญญาณวิทยุออกไป โดยการเปลี่ยนโหมดการทำงานให้เป็นโหมดส่งวิทยุ TX ด้วยการเรียกเมธอด setTxState ระหว่างนี้ไมโครจะรอให้ CC1100 ทำงาน โดยการเรียกเมธอด wait_GDO0_height เพื่อรอให้ CC1100 ส่งไบต์ รหัสเวลาพร้อมสัมพันธ์ ให้เรียบร้อย หลังจากนั้นจึงเรียกเมธอด wait_GDO0_low เพื่อรอให้การส่งทั้งหมดเสร็จสิ้น หลังจากส่งข้อมูลเรียบร้อยแล้วจึงเปลี่ยนสถานะการทำงานของชิปเข้าสู่โหมด เดินเครื่องเปล่า และล้างข้อมูลในบัพเฟอร์ข้อมูลส่ง TX FIFO ด้วยการเรียกเมธอด setIdleState และ flushTxFifo ตามลำดับ ท้ายที่สุดกระบวนการส่งข้อมูลยุติลงด้วยการเปลี่ยนโหมดการทำงานเข้าสู่โหมดรับวิทยุดังรายละเอียดในชุดคำสั่งที่ 4.5

```
// Check that the RX state has been entered
  while (((marcState = readStatusReg(CC1101_MARCSTATE)) & 0x1F) != 0x0D)
  {
    if (marcState == 0x11)        // RX_OVERFLOW
      flushRxFifo();              // flush receive queue
  }
  delayMicroseconds(500);
  // Set data length at the first position of the TX FIFO
  writeReg(CC1101_TXFIFO,  packet.length);
  // Write data into the TX FIFO
  writeBurstReg(CC1101_TXFIFO, packet.data, packet.length);
// CCA enabled: will enter TX state only if the channel is clear
  setTxState();
```

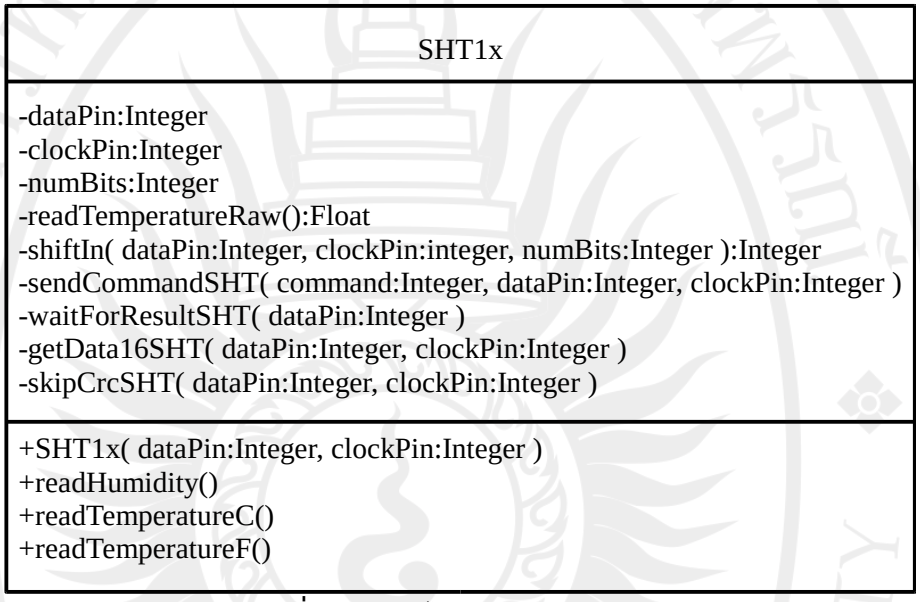
```
// Check that TX state is being entered (state = RXTX_SETTLING)
  marcState = readStatusReg(CC1101_MARCSTATE) & 0x1F;
  if((marcState != 0x13) && (marcState != 0x14) && (marcState != 0x15))
  {
    setIdleState();       // Enter IDLE state
    flushTxFifo();        // Flush Tx FIFO
    setRxState();         // Back to RX state
 // Declare to be in Rx state
    rfState = RFSTATE_RX;
    return false;
  }
// Wait for the sync word to be transmitted
  wait_GDO0_high();
// Wait until the end of the packet transmission
  wait_GDO0_low();
// Check that the TX FIFO is empty
  if((readStatusReg(CC1101_TXBYTES) & 0x7F) == 0)
    res = true;

  setIdleState();       // Enter IDLE state
  flushTxFifo();        // Flush Tx FIFO
// Enter back into RX state
  setRxState();
// Declare to be in Rx state
  rfState = RFSTATE_RX;
return res;
}
```

ชุดคำสั่งที่ 4.5 เมธอดส่งข้อมูลผ่านสัญญาณวิทยุ

4.1.1.2 การสื่อสารกับ SHT11

การสื่อสารระหว่างไมโครคอนโทรเลอร์กับ SHT11 นั้นทำด้วยการใช้คลาส SHT1x2 (Oxer, 2009) ซึ่งเป็นคลังคำสั่งที่ออกแบบเพื่อใช้สื่อสารกับเซนเซอร์ SHT10, SHT11 และ SHT15 ได้ โดยคลาสนี้ใช้ฟังก์ชั่นพื้นฐานของ Arduino เป็นหลักทั้งหมด ดังรายละเอียดในภาพที่ 4.3



ภาพที่ 4.3 คลาสไดอะแกรมของ SHT1x

เมธอด shiftIn เป็นเมธอดหลักของคลาส SHT1X ที่ไมโครคอนโทรเลอร์จะต้องเรียกใช้เพื่อการอ่านข้อมูลจากอุปกรณ์ โดยเมธอดนี้จะมี อาร์กิวเม้นต์สามค่า ประกอบด้วย หมายเลขขาข้อมูล, หมายเลขขานาฬิกา และจำนวนของบิตที่ต้องการอ่าน โดยงานวิจัยนี้กำหนดให้อ่านข้อมูลครั้งละแปดบิตเสมอ การอ่านข้อมูลจากโพรโทคอลนี้จะใช้การวนรอบอ่านข้อมูลจากอุปกรณ์ครั้งละหนึ่งบิตจนครบแปลดบิต ซึ่งไมโครคอนโทรเลอร์และอุปกรณ์จะประสานจังหวะการส่งข้อมูลด้วยการใช้สัญญาณที่ขานาฬิกาเป็นหลัก ขั้นตอนเหล่านี้จะเริ่มด้วยการส่งสัญญาณ HIGH ไปที่ขานาฬิกาของอุปกรณ์เพื่อบอกให้อุปกรณ์ทราบว่าไมโครคอนโทรเลอร์ต้องการอ่านข้อมูลหนึ่งบิต หลังจากนั้นจึงอ่านข้อมูลจากขาข้อมูลด้วยการเรียกใช้ฟังก์ชั่นdigitalWriteหลังจากนั้นแล้วจึงลดสัญญาณที่ขานาฬิกาเป็น LOW เพื่อบอกให้อุปกรณ์รู้ว่าอ่านข้อมูลของบิตนั้นเรียบร้อยแล้ว เสร็จแล้วจึงวนรอบอ่านข้อมูลของบิตอื่นต่อไป อย่างไรก็ตามผู้วิจัยก็พบว่าการอ่านส่งสัญญาณนาฬิกาเป็น HIGH แล้วอ่านข้อมูลกลับมาด้วยฟังก์ชั่น digitalRead นั้นในทันทีนั้นมักจะเกิดปัญหาการประสานจังหวะการทำงาน ผู้วิจัยจึงแก้ไขปัญหานี้ด้วยการให้ไมโครคอนโทรเลอร์หน่วงเวลาระหว่างการทำงานดังกล่าวเล็กน้อย

2 http://www.practicalarduino.com/

ซึ่งในกรณีนี้จะกำหนดให้หน่วงเวลา 10 มิลลิวินาทีก่อนที่จะการอ่านข้อมูลบิตเข้ามา ดังรายละเอียดในชุดคำสั่งที่ 4.6

```
int SHT1x::shiftIn(int _dataPin, int _clockPin, int _numBits)
{
  int ret = 0;
  int i;

  for (i=0; i<_numBits; ++i)
  {
     digitalWrite(_clockPin, HIGH);
     delay(10);
     ret = ret*2 + digitalRead(_dataPin);
     digitalWrite(_clockPin, LOW);
  }

  return(ret);
}
```

ชุดคำสั่งที่ 4.6 เมธอด shiftIn ของคลาส SHT1x

การส่งคำสั่งไปยัง SHT11 จะทำด้วยเมธอด sendCommandSHT ซึ่งจะมีอาร์กิวเม้นต์สามค่า ประกอบด้วย ไบต์คำสั่ง, ขาข้อมูล และขานาฬิกา เมธอดนี้จะเริ่มการทำงานด้วยตัวจัดลำดับ (sequence) ที่เรียกว่าตัวจัดลำดับเริ่มต้นการส่งข้อมูล(Transmission Start sequence)ซึ่งจะเริ่มต้นด้วยการทำให้สัญญาณขาข้อมูลมีค่าเป็น LOW ในขณะที่สัญญาณขานาฬิกามีค่าเป็นHIGH หลังจากนั้นจึงดึงสัญญาณขาข้อมูลให้เป็น LOW สลับกับ HIGH ขั้นตอนสุดท้ายก็คือการดึงสัญญาณที่ขาข้อมูลให้กลับมามีค่าเป็น HIGH อีกครั้งหนึ่ง ดังรายละเอียดในชุดคำสั่งที่ 4.8

```
pinMode(_dataPin,                                    OUTPUT);
 pinMode(_clockPin,                                  OUTPUT);
 digitalWrite(_dataPin,                              HIGH);
 digitalWrite(_clockPin,                             HIGH);
 digitalWrite(_dataPin,                              LOW);
 digitalWrite(_clockPin,                             LOW);
 digitalWrite(_clockPin,                             HIGH);
 digitalWrite(_dataPin,                              HIGH);
 digitalWrite(_clockPin, LOW);
```

ชุดคำสั่งที่ 4.7 ชุดคำสั่ง Arduino สำหรับตัวจัดลำดับเริ่มต้นการส่งข้อมูลของ SHT11

ตารางที่ 4.2 บิตคำสั่งงานของ SHT11
ที่มา: Sensirion, 2010

คำสั่ง	**ค่า**
Reserved	0000x
Measure Temperature	00011
Measure Relative Humidity	00101
Read status Register	00111
Write status Register	00110
Reserved	0101x-1110x
Soft reset,resets the interface, clears the status register to default values.Wait minimum 11 ms before next command	11110

ตารางที่ 4.2 แสดงรายการของบิตคำสั่งแบบต่าง ๆ ของ SHT11ซึ่งจะเห็นว่าประกอบด้วยบิตจำนวนห้าบิต โดยในการอิมพลิเม้นต์จริงนั้น ผู้วิจัยได้ใช้ฟังก์ชั่น shiftOut ซึ่งเป็นฟังก์ชั่นจากคลังคำสั่งของ Arduino เองในการส่งค่าบิตนี้ไปยังอุปกรณ์ โดยฟังก์ชั่น shiftOut นี้จะมีอาร์กิวเม้นต์สี่ค่า ประกอบด้วยหมายเลขขาสัญญาณ, หมายเลขขานาฬิกา, รูปแบบลำดับของบิต และไบต์ข้อมูลที่ต้องการส่ง โดยรูปแบบของลำดับบิตนี้สามารถกำหนดได้ว่าจะให้ส่งบิตสำคัญน้อยสุดก่อน

(LSBFIRST) หรือจะให้ส่งบิตสำคัญมากสุดก่อน (MSBFIRST) โดยในกรณีของการสื่อสารกับ SHT11 นั้นกำหนดให้ส่งบิตน้อยสุดไปก่อน สำหรับค่าที่ต้องส่งไปนั้นจะเป็นแปลงคำสั่งที่ต้องการในตารางให้เป็นข้อมูลหนึ่งไบต์ แล้วจึงเรียกฟังก์ชั่นนี้ ตัวอย่างต่อไปนี้แสดงการส่งคำสั่งวัดอุณหภูมิ โดยจากตารางจะเห็นว่าค่าของคำสั่งนี้มีค่าเป็น 00011 ทั้งนี้เราอาจจะส่งข้อมูลนี้ในรูปแบบเลขฐานสิบก็ได้ (เช่น ส่งค่า 0b00000011 หรือ 3 ก็ได้ทั้ง 2 แบบ)

```
shiftOut(_dataPin, _clockPin, MSBFIRST, 0b00000011);
```

หลังจากส่งคำสั่งไปแล้ว เราจะต้องรอให้อุปกรณ์ตอบสนองต่อคำสั่งสักครู่ โดยหากอุปกรณ์ได้รับคำสั่งเรียบแล้วนั้น อุปกรณ์จะดึงสัญญาณที่ขา DATA ให้มีค่าเป็น LOW กระบวนการนี้ทำได้โดยการเรียกฟังก์ชั่น digitalWrite พร้อมระบุขาสัญญาณให้เป็น HIGH หลังจากนั้นจึงอ่านค่าจากขา DATAแล้วนำมาเปรียบเทียบดูว่าค่าที่อ่านได้มีค่าเป็น LOW หรือไม่ ถ้าหากพบว่าค่าที่อ่านได้ไม่ใช่ LOW ก็จะหมายความว่าเกิดความผิดพลาดในการทำงาน แต่หากเป็น LOW ก็จะตรวจสอบต่อไปว่าสัญญาณขา DATAค่าอ่านได้ค่าต่อไปนั้นมีค่าเป็น HIGH หรือไม่ ถ้าหากยังเป็น LOW เช่นเดิมก็จะแสดงว่ามีความผิดพลาดเกิดขึ้น แต่หากเป็น HIGH ก็แสดงการสื่อสารและคำสั่งที่ส่งไปนั้นได้รับการปฏิบัติเรียบร้อย รายละเอียดของการทำงานดังกล่าวนี้แสดงอยู่ในชุดคำสั่งต่อไปนี้

```
// Verify we get the correct ack
  digitalWrite(_clockPin, HIGH);
  pinMode(_dataPin, INPUT);
  ack = digitalRead(_dataPin);
  if (ack != LOW) {
    // Serial.println("Ack Error 0");
  }
  digitalWrite(_clockPin, LOW);
  ack = digitalRead(_dataPin);
  if (ack != HIGH) {
    // Serial.println("Ack Error 1");
  }
```

ชุดคำสั่งที่ 4.8 การอ่านข้อมูลจาก SHT11

4.1.1.3 การวัดค่าความชื้นสัมพัทธ์และอุณหภูมิ

เมื่อSH11 ได้รับคำสั่งงานวัดอุณหภูมิ (0b00000101) หรือวัดความชื้นสัมพัทธ์ (0b000000011) อุปกรณ์ก็จะทำการวัดข้อมูลดังกล่าวให้ ซึ่งจะใช้เวลาระยะหนึ่งจึงจะเสร็จ ซึ่งในกรณีของการวัดค่าความละเอียดแบบ 8, 12 และ 14 บิตนั้น อุปกรณ์จะใช้เวลาในการทำงานประมาณ 20, 80 และ 320 มิลลิวินาทีตามลำดับ โดยระยะเวลาเหล่านี้อาจจะเปลี่ยนแปลงได้ ขึ้นอยู่กับความเร็วของออสซิลเลเตอร์ (oscillator) ภายในชิปแต่ละรุ่น และเมื่ออุปกรณ์ทำงานเรียบร้อยแล้ว อุปกรณ์ก็จะส่งสัญญาณให้ไมโครคอนโทลเลอร์รับรู้โดยการส่งสัญญาณขา DATA ให้เป็น LOW แล้วเปลี่ยนการทำงานไปสู่โหมดเดินเครื่องเปล่า

ไมโครคอนโทรเลอร์จะต้องรอให้แน่ใจว่าอุปกรณ์ทำการวัดค่าเรียบร้อยแล้วก่อนจะอ่านข้อมูล นั่นคือจะต้องรอจนกว่าสัญญาณขา DATA จะมีค่าเป็น LOW หลังจากนั้นจึงเริ่มต้นอ่านค่าได้ โดยค่าที่วัดได้นี้จะคงอยู่ในบัฟเฟอร์ของ SHT11 จนกว่าจะมีการอ่านค่าออกไป โดยข้อมูลที่ได้จากการวัดจะประกอบด้วยข้อมูลจำนวนสองไบต์และซีอาร์ซี (cyclic redundancy check)สำหรับการตรวจสอบส่วนซ้ำซ้อนอีกหนึ่งไบต์ นอกจากนั้นแล้วไมโครคอนโทรเลอร์จะต้องส่งสัญญาณให้อุปกรณ์รับทราบทุกครั้งที่มีการอ่านไบต์ข้อมูลค่าเหล่านี้ออกไป ซึ่งทำได้ด้วยการดึงสัญญาณขา DATA ให้มีค่าเป็น LOW ทั้งนี้อุปกรณ์จะส่งข้อมูลกลับให้ไมโครคอนโทรเลอร์แบบบิตมากสุดก่อนเสมอ ดังนั้นโปรแกรมเมอร์จึงจะต้องตรวจสอบให้แน่ใจว่าข้อมูลที่ได้รับนั้นมีลำดับของบิตที่ถูกต้องก่อนที่จะนำไปใช้งานต่อไปการอ่านข้อมูลจะสิ้นสุดลงเมื่อไมโครคอนโทรเลอร์อ่านถึงมาบิตสุดท้ายของไบต์ซีอาร์ซี ถ้าหากไมโครคอนโทรเลอร์ไม่ต้องการใช้ข้อมูลซีอาร์ซีแล้ว ก็อาจจะยุติการอ่านข้อมูลได้ด้วยการทำให้ค่า ACK หรือค่าการตอบรับมีค่าเป็น HIGH ค้างไว้ และเมื่ออ่านข้อมูลหมดแล้ว ไมโครคอนโทรเลอร์ก็จะสั่งให้อุปกรณ์เปลี่ยนการทำงานไปสู่โหมดนอนหลับโดยอัตโนมัติ

เมธอด readTemperatureRawจะทำหน้าที่อ่านข้อมูลตามขั้นตอนดังกล่าว โดยเมธอดนี้จะเรียกใช้เมธอดที่ชื่อว่า waitForResultSHT เพื่อรอให้อุปกรณ์ทำการวัดค่าให้เรียบร้อยแล้วเรียกใช้เมธอด skipCrcSHT เพื่อยกเลิกการอ่านข้อมูล CRC จากอุปกรณ์ดังรายละเอียดในชุดคำสั่งที่ 4.9

```
float SHT1x::readTemperatureRaw()
{
  int _val;

  // Command to send to the SHT1x to request Temperature
  int _gTempCmd  = 0b00000011;

  sendCommandSHT(_gTempCmd, _dataPin, _clockPin);
  waitForResultSHT(_dataPin);
  _val = getData16SHT(_dataPin, _clockPin);
  skipCrcSHT(_dataPin, _clockPin);

  return (_val);
}
```

ชุดคำสั่งที่ 4.9 เมธอด readTemeratureRaw ของคลาส SHT1x

โดยเมธอด waitForResultSHT จะทำหน้าที่รอการตอบรับจาก SHT11 ตามหลักการที่ได้กล่าวมาแล้ว ซึ่งผู้วิจัยได้ปรับปรุงคำสั่งของเมธอดให้รอการวัดค่าดังกล่าวด้วยการวนซ้ำเป็นจำนวน 100 รอบ และหากไมโครคอนโทรเลอร์วนซ้ำครบ 100 รอบแล้วยังพบว่าสัญญาณขา DATA ยังมีค่าเป็น HIGH อยู่ (ซึ่งหมายความว่าอุปกรณ์ยังไม่ตอบสนองหรือยังไม่พร้อมทำงานต่อ) ก็จะจบการทำงานโดยแจ้งกลับให้เมธอดหลักทราบว่าเกิดความผิดพลาดในการทำงาน

```
void SHT1x::waitForResultSHT(int _dataPin)
{
  int i;
  int ack;

  pinMode(_dataPin, INPUT);

  for(i= 0; i < 100; ++i)
  {
    delay(10);
    ack = digitalRead(_dataPin);

    if (ack == LOW) {
      break;
    }
  }

  if (ack == HIGH) {
    Serial.println("Ack Error 2");
  }
}
```

ชุดคำสั่งที่ 4.10 เมธอด waitForResultSHT ของคลาส SHT1x

ในทำนองเดียวกัน เมธอด skipCrcSHT จะทำหน้าที่ข้ามการอ่านและตรวจสอบข้อมูลซีอาร์ซีซึ่งทำโดยการส่งสัญญาณ HIGH ไปที่ขาข้อมูลตามด้วยการส่งสัญญาณHIGH และ LOW ไปที่ขานาฬิกาตามลำดับ ดังรายละเอียดในชุดคำสั่งที่ 4.11

```
void SHT1x::skipCrcSHT(int _dataPin, int _clockPin)
{
  // Skip acknowledge to end trans (no CRC)
  pinMode(_dataPin, OUTPUT);
  pinMode(_clockPin, OUTPUT);

  digitalWrite(_dataPin, HIGH);
  digitalWrite(_clockPin, HIGH);
  digitalWrite(_clockPin, LOW);
}
```

ชุดคำสั่งที่ 4.11 เมธอด skipCrcSHT ของคลาส SHT1x

ข้อมูลที่อ่านจากเมธอด readTemperatureRaw ได้นั้นจะเป็นข้อมูลดิบ ซึ่งอาจจะยังไม่มีความหมายตามที่ต้องการ ดังนั้นขั้นตอนต่อไปคือการแปลงข้อมูลดิบนี้ให้อยู่ในรูปที่ถูกต้องดังรายละเอียดต่อไปนี้

ในกรณีของการอ่านค่าความชื้นสัมพัทธ์นั้น ถ้ากำหนดให้ค่าข้อมูลดิบที่อ่านได้จากจากชิปมีค่าเป็น SO_{RH} แล้ว เราจะสามารถคำนวณหาRH_{linear} ซึ่งเป็นความชื้นสัมพัทธ์เชิงเส้น (linear relative humidity) ได้จากสูตรต่อไปนี้

$$RH_{linear} = c_1 + c_2*SO_{RH} + c_3*SO_{RH}^2$$

โดยที่ c1, c2 และ c3 คือค่าคงที่ที่มีค่าขึ้นอยู่กับความละเอียดของการอ่านค่า ดังรายละเอียดที่แสดงในตารางที่ 4.3

ทั้งนี้หากค่าอุณหภูมิ ณ ขณะที่ทำการวัดค่าความชื้นสัมพัทธ์นั้นมีค่ามากกว่า 25 องศาเซลเซียสแล้ว เราจะต้องชดเชย (compensate)ค่า RH_linear ที่คำนวณได้ ให้เป็นค่าความชื้อสัมพัทธ์จริง ดังนี้

$$RH_{true} = (T_{°C}-25)\cdot(t_1 + t_2\ SO_{RH}) + RH_{linear}$$

โดยที่ T คือ อุณหภูมิขณะที่ทำการวัดค่า และ t_1 และ t_2 คือสัมประสิทธิ์ของการชดเชยอุณหภูมิที่มีค่าขึ้นอยู่กับความละเอียดของการอ่านค่า ดังรายละเอียดในตารางที่ 4.4

ตารางที่ 4.3 สัมประสิทธิ์สำหรับการคำนวณความชื้นสัมพัทธ์จาก SHT1x

ที่มา: Sensirion, 2010

SO_{RH}	C_1	C_2	C_3
12 บิต	-2.0468	0.0367	-1.5955E-6
8 บิต	-2.0468	0.5872	-4.0845E-4

ตารางที่ 4.4 สัมประสิทธิ์การชดเชยอุณหภูมิสำหรับการคำนวณความชื้นสัมพัทธ์จาก SHT1x

ที่มา: Sensirion, 2010

SO_{RH}	T_1	T_2
12 บิต	0.01	0.00008
8 บิต	0.01	0.00128

สำหรับเซนเซอร์วัดอุณหภูมิของ SHT1x นั้นมีความเป็นเชิงเส้นมาก (Sensirion, 2010) ดังนั้นเราจึงสามารถแปลงข้อมูลที่อ่านได้จากขาดิจิทัลให้เป็นค่าอุณหภูมิได้จากสูตร

$T = d_1 + d_2 . SO_T$

โดยที่SOt เป็นค่าข้อมูลดิบที่อ่านได้จากขาอุปกรณ์ และ d1 และ d2 เป็นสัมประสิทธิ์ของการแปลงค่า ซึ่งจะมีค่าขึ้นอยู่กับความต่างศักย์และความละเอียดของการวัดค่า ตามรายละเอียดในตารางที่ 4.5 และ 4.6

ตารางที่ 4.2 สัมประสิทธิ์การแปลงอุณหภูมิสำหรับการคำนวณอุณหภูมิจาก SHT1x เมื่อเทียบกับความต่างศักย์อินพุต

ที่มา: Sensirion, 2010

VDD	**d1 (องศาเซลเซียส)**	**d1 (องศาฟาเรนต์ไฮต์)**
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

ตารางที่ 4.6 สัมประสิทธิ์การแปลงอุณหภูมิสำหรับการคำนวณอุณหภูมิจาก SHT1x เมื่อเทียบกับกับความละเอียดของการวัดค่า

ที่มา: Sensirion, 2010

SO_{RH}	d_2 **(องศาเซลเซียส)**	d_2 **(องศาฟาเรนต์ไฮต์)**
14บิต	0.01	0.018
12บิต	0.04	0.072

4.1.1.4 การสื่อสารกับตัวอ่านแสง

ตัววัดความเข้มแสงแวดล้อมที่ใช้ในงานวิจัยนี้ส่งข้อมูลมาให้ไมโครคอนโทรเลอร์แบบแอนะล็อก ดังนั้นโดยหลักการแล้วเราสามารถอ่านข้อมูลได้โดยง่าย ซึ่งโดยทั่วไปนั้นจะทำโดยใช้ฟังก์ชั่น analogRead ของบอร์ด Arduino ในการอ่านข้อมูลจากขาแอละน็อกที่กำหนดไว้ อย่างไรก็ตามผู้วิจัยพบว่า การอ่านข้อมูลแบบแอนะล็อกนั้นมักจะเกิดความคลาดเคลื่อนได้ โดยอย่างยิ่งหากเอดีซียังปรับแรงดังไฟที่ขาอุปกรณ์ไม่เรียบร้อย นอกจากนั้นแล้ว ผู้วิจัยพบว่าเซนเซอร์วัดค่าความเข้มแสงแวดล้อมนี้ยังมีความไว (sensitive) ต่อการเปลี่ยนแปลงความเข้มแสงเป็นอย่างมากด้วย ดังนั้นในการอิมพลิเม้นต์จริง ผู้วิจัยจึงสร้างฟังก์ชั่น readAnalogRaw เพื่อทำหน้าที่อ่านข้อมูลแอนะล็อกโดยมีอาร์กิวเม้นต์เป็นหมายเลขขาแอนะล็อกที่ต้องการใช้เป็นขาอินพุต และได้กำหนดให้ฟังก์ชั่นนี้จะวนอ่านค่าเฉลี่ยจากขาอินพุตที่กำหนดมาเป็นจำนวน 10 รอบ ทั้งนี้เพื่อลดความคลาดเคลื่อนของค่าที่อ่านจากอุปกรณ์ที่มีความไวต่อการเปลี่ยนแปลงเช่นนี้ โดยในแต่ละรอบการทำงานจะเริ่มต้นด้วยการอ่านข้อมูลเปล่าทิ้งไปก่อนหนึ่งครั้งแล้วหยุดรอประมาณ 10 มิลลิวินาที เพื่อรอให้เอดีซี เสถียร หลังจากนั้นจึงทำการอ่านข้อมูลจริง ๆ มาเก็บสะสมไว้แล้วรอ 20 มิลลิวินาที ก่อนจะวนซ้ำ เมื่ออ่านข้อมูลครบ 10 รอบแล้วจะจึงคำนวณหาค่าเฉลี่ยจากค่าที่อ่านได้ทั้ง 10 รอบ ชุดคำสั่งที่ 4.12 ต่อไปนี้แสดงการอิมพลิเม้นต์การทำงานดังกล่าว

```
int readAnalogRaw(int pin) {
  int i;
  int tmp_val = 0;
  for(i = 0; i < 10; ++i) {
    analogRead(pin); // switch the pin to the ADC
    delay(10); // wait for ADC to stabilize
    tmp_val += analogRead(pin); // accumulate the value
    delay(20); // another small delay before next measurement
  }
  return tmp_val = (int)(tmp_val/10.0);
}
```

ชุดคำสั่งที่ 4.12 ฟังชั่นสำหรับการอ่านค่าจากเซนเซอร์แบบแอนะล็อก

4.1.1.5 การสื่อสารกับตัววัดค่าอุณหภูมิดิน

เซนเซอร์วัดอุณหภูมิดินจะสื่อสารกับไมโครด้วยโพรโทคอล 1-Wire โดยงานวิจัยนี้ใช้คลังคำสั่งOneWire[3]เพื่อการอ่านข้อมูลจากโพรโทคอลนี้ และใช้คลังคำสั่ง DallasTemperature[4] ในการอ่านข้อมูลที่วัดได้จากอุปกรณ์

ในขั้นตอนการเตรียมความพร้อมการใช้งานนั้น ผู้วิจัยได้กำหนดให้อุปกรณ์นี้อ่านข้อมูลที่ความละเดียด 12 บิต ซึ่งทำได้โดยการเรียกเมธอด setResolution ดังนี้

```
ds.setResolution(dsAddress, 12);
```

โดยที่ ds คือวัตถุของคลาส DallasTemperatureและ dsAddress คือเลขที่อยู่ของอุปกรณ์ ซึ่งจะได้มาด้วยการเรียกใช้เมธอด ds.getAddress พร้อมส่งตัวแปร dsAddress เป็นอาร์กิวเม้นต์ของเมธอด หลังจากนั้นจึงอ่านค่าอุณหภูมิซึ่งทำได้ด้วยการเรียกเมธอด requestTemperatures ซึ่งจะเป็นการส่งคำสั่งงานอ่านอุณหภูมิไปให้อุปกรณ์ หลังจากนั้นจึงทำการอ่านค่าที่วัดได้ออกมาจากอุปกรณ์ด้วยการเรียกเมธอด getTempC โดยส่งเลขที่อยู่ของอุปกรณ์เป็นอาร์กิวเม้นต์ของฟังก์ชั่น

3 https://www.pjrc.com/teensy/td_libs_OneWire.html

4 https://milesburton.com/Dallas_Temperature_Control_Library

4.1.1.6 การสื่อสารกับตัวอ่านความชื้นดิน

เซนเซอร์สำหรับการวัดความชื้นดินนั้นจะส่งค่าเป็นสัญญาณแอนะล็อก ดังนั้นในงานวิจัยนี้จะกำหนดให้ไมโครคอนโทรเลอร์ใช้ฟังก์ชั่น readAnalogRaw ในการอ่านค่าจากอุปกรณ์เช่นเดียวกับการอ่านค่าจากเซนเซอร์ความเข้มแสงแวดล้อม

4.1.1.7 การเข้ารหัสข้อมูล

ข้อมูลที่อ่านได้จากเซนเซอร์ทั้งหมดนี้จะถูกรวบรวมเข้าด้วยกันแล้วเข้ารหัสเป็นข้อมูลใหม่หนึ่งชุด โดยข้อมูลใหม่นี้จะอยู่ในรูปของแถวลำดับของไบต์ ซึ่งงานวิจัยนี้กำหนดไว้ให้มีจำนวน 13 ไบต์ และได้กำหนดการสำรองไบต์ให้ข้อมูลต่าง ๆ ดังรายละเอียดในตารางที่ 4.7

ตารางที่ 4.7 การเข้ารหัสข้อมูลสำหรับการส่งสัญญาณวิทยุ

ไบต์ที่	**ข้อมูล**
1 - 2	อุณหภูมิดิน
3 - 4	ความชื้นสัมพัทธ์ดิน
5 - 6	อุณหภูมิอากาศ
7 - 8	ความชื้นสัมพัทธ์อากาศ
9 - 10	ความเข้มแสงแวดล้อม
11 - 12	voltage อ้างอิง

และเพื่อความสะดวกในการส่งข้อมูล ผู้วิจัยจึงแปลงข้อมูลที่เป็นทศนิยมให้เป็นรูปจำนวนเต็ม โดยการคูณด้วย 100.0 ซึ่งจะทำให้ค่าที่ส่งได้รับมีความละเอียดถึงทศนิยมลำดับที่สอง ซึ่งเป็นความละเอียดที่เพียงพอต่อการใช้งานในงานวิจัยครั้งนี้ อย่างไรก็ตามเราจำเป็นที่จะต้องแปลงข้อมูลดังกล่าวให้อยู่ในแถวลำดับของไบต์ ในกรณีของบอร์ด Arduino นั้นอาจจะมีความซับซ้อนกว่าการดำเนินการในภาษา C/C++ ซึ่งในงานวิจัยนี้แปลงข้อมูลเหล่านี้ให้เป็นแถวลำดับของไบต์โดยการแคส (cast) ชนิดข้อมูลให้เป็น int (เลขจำนวนเต็ม) หลังจากนั้นจึงแปลงข้อมูล int ให้เป็นตัวชี้ (pointer) ด้วยการแคส เช่น *((int) *byte_array) โดยชุดคำสั่งที่ 4.13 แสดงรายละเอียดของฟังก์ชั่น encodeData ที่กล่าวมานี้

```
// encode air humidity data
  tmp = int(air_humd * 100);
  *((int *)byte_array) = tmp;
  encoded_data.data[7] = byte_array[0];
  encoded_data.data[8] = byte_array[1];

  // encode ambient light intensity data
  tmp = int(light_amb);
  *((int *)byte_array) = tmp;
  encoded_data.data[ 9] = byte_array[0];
  encoded_data.data[10] = byte_array[1];

  // Arduino reference volatage level
  tmp = int(ref_volt);
  *((int *)byte_array) = tmp;
  encoded_data.data[11] = byte_array[0];
  encoded_data.data[12] = byte_array[1];
}
```

ชุดคำสั่งที่ 4.13 ฟังก์ชั่นเข้ารหัสข้อมูล

4.1.1.8 การส่งข้อมูลและการสั่งให้บอร์ด Arduino หลับ

การส่งข้อมูลไปยังเครื่องคอมพิวเตอร์แม่ข่าย จะทำในฟังก์ชั่น loop ของ Arduino โดยจะเริ่มต้นด้วยการอ่านข้อมูลจากเซนเซอร์ต่างๆ แล้วเข้ารหัสด้วยการเรียกใช้ฟังก์ชั่นตามที่ได้อธิบายมาในบทที่สามและส่วนต้นของบทที่สี่นี้ หลังจากเตรียมข้อมูลทั้งหมดพร้อมแล้ว ไมโครคอนโทรเลอร์จะเรียกตัวรับส่งวิทยุให้พร้อมทำงานด้วยการเรียกเมธอด wakeup และเมื่อตัวรับส่งวิทยุพร้อมทำงานแล้วจึงส่งข้อมูลที่เข้ารหัสแล้วไปยังเครื่องแม่ข่ายตามรายละเอียดที่ได้กล่าวมาแล้ว

ในทางปฏิบัติผู้วิจัยพบว่าการส่งข้อมูลไปยังเครื่องแม่ข่ายนั้นมักจะมีปัญหาการสูญหายของข้อมูลเกิดขึ้นบ้างเป็นบางครั้ง ซึ่งปัญหานี้อาจจะเกิดได้จากหลายสาเหตุ โดยเฉพาะอย่างยิ่งสัญญาณรบกวนที่แทรกเข้ามาจากแหล่งกำเนิดอื่น ๆ ภายนอก รวมทั้งความบกพร่องของอุปกรณ์เอง ดังนั้นเพื่อลดปัญหาการสูญหายของข้อมูลงานวิจัยนี้จึงได้กำหนดให้มีการส่งข้อมูลเดียวกันซ้ำ

หลายรอบ โดยในกรณีนี้ได้กำหนดให้การส่งข้อมูลหนึ่งชุดนี้จะถูกส่งไปยังเครื่องแม่ข่ายจำนวนสิบครั้ง และเมื่อส่งข้อมูลเรียบร้อยแล้วไมโครคอนโทรเลอร์ก็จะปรับการทำงานของตัวเองให้ไปอยู่โหมดหลับแบบประหยัดพลังงานที่สุดตามรายละเอียดที่ออกแบบไว้ในบทที่สาม

การสั่งให้บอร์ด Arduino หลับลึกนั้นจะใช้คลังคำสั่ง LowPower5(Moh, 2013) ในการทำงาน ซึ่งงานวิจัยนี้กำหนดให้บอร์ด Arduino หลับในโหมดปิดพลังงาน

POWER_DOWN ซึ่งจะส่งผลให้ตัวไมโครคอนโทรเลอร์นี้ปิดการทำงานของโมดูลเอดีซีและบีโอดีรวมถึงพาร์ตเชื่อมต่อทั้งหมดด้วย โดยเมธอด powerDown ของคลาส LowPower นั้นจะทำหน้าที่สั่งไมโครคอนโทรเลอร์ให้เข้าสู่โหมดหลับลึก โดยเมธอดนี้จะรับอาร์กิวเม้นต์จำนวนสามค่าคือ ระยะเวลาของการหลับ, การสั่งปิดโมดูลเอดีซี และการสั่งปิดโมดูลบีโอดี ตามลำดับซึ่งกระบวนการสั่งงานจริง ๆ นั้นจะทำงานโดยกลุ่มของคลังคำสั่ง sleep, wdt และ power ของ AVR[6] ซึ่งอาจจะถือว่าเป็นคลาสและฟังก์ชั่นระดับล่างสุดสำหรับการโปรแกรมชิป Atmel บนบอร์ด Arduino

ดังที่ได้กล่าวมาแล้วว่าระยะเวลาที่นานที่สุดที่เราสามารถสั่งให้ไมโครคอนโทรเลอร์หลับได้ก็คือแปดวินาที แต่งานวิจัยนี้กำหนดให้ไมโครวัดข้อมูลทุกๆ 10 นาที อย่างไรก็ตามบอร์ด Arduino สามารถหลับลึกตามขั้นตอนข้างบนนี้ได้นานที่สุดเป็นเวลาแปดวินาทีเท่านั้น ดังนั้นการที่จะให้อุปกรณ์หลับได้เป็นเวลา 10นาทีนั้นจะต้องใช้การวนรอบเพื่อสั่งให้หลับให้ครบตามที่กำหนด ซึ่งแม้ว่าอุปกรณ์จะไม่ได้หลับอย่างต่อเนื่อง แต่ก็ยังคงใช้พลังงานน้อยกว่าการไม่หลับเป็นอย่างมาก ชุดคำสั่งต่อไปนี้แสดงการวนรอบเพื่อให้ไมโครหลับได้ 10นาที ชุดคำสั่งที่ 4.14 แสดงวิธีการสั่งให้ไมโครคอนโทรเลอร์หลับเป็นระยะเวลา 10 นาที

```
for(i = 0; i < NUM_SLEEP_ITERATIONS; ++i) {
    LowPower.powerDown(
      SLEEP_8S, // Sleep for 8s
      ADC_OFF, // turn off ADC module
      BOD_OFF  // turn off Brown Out Detector (BOD) module.
    );
    delay(500);
  }
```

ชุดคำสั่งที่ 4.14 ลูปหลักสำหรับใช้สั่งให้บอร์ด Arduino หลับเป็นเวลา 10 นาที โดยในที่นี้กำหนดให้ค่าคงที่ NUM_SLEEP_ITERATIONS มีค่าเป็น 75

5 http://www.rocketscream.com/blog/2011/07/04/lightweight-low-power-arduino-library/

6 http://www.atmel.com/webdoc/AVRLibcReferenceManual

4.1.2 การจัดการค่าบนเครื่องแม่ข่าย

ที่เครื่องคอมพิวเตอร์แม่ข่าย เครื่องหลักนั้นจะมี ชิป CC1100 ทำหน้าที่ฟังสัญญาณวิทยุที่ส่งมาจากเครื่องลูกข่าย โดยงานวิจัยนี้กำหนดให้ทั้งสองฝั่งสื่อสารกันด้วยความถี่ 833เฮิรทซ์ โดยในกรณีที่ตัวรับวิทยุพบว่ามีข้อมูลที่ความถี่นี้เข้ามา เมธอด receiveData ของคลาส CC1101 ก็จะเริ่มต้นทำงาน เมธอดนี้จะเริ่มต้นทำงานด้วยการอ่านข้อมูลสถานะของตัวรับวิทยุจากเรจิสเตอร์หากพบว่า ตัวรับวิทยุอยู่ในสถานะพร้อมสำหรับการอ่านข้อมูลและมีข้อมูลไม่มากเกินเก็บ (overflow) อุปกรณ์ก็จะเริ่มการจัดการข้อมูลสัญญาณวิทยุที่ส่งมา ซึ่งจะเริ่มด้วยการอ่านความยาวของข้อมูลด้วยการเรียกเมธอด readConfigReg พร้อมส่งอาร์กิวเม้นต์ CC1101_RXFIFO เพื่อกำหนดว่าไมโครคอนโทรเลอร์ต้องการอ่านข้อมูลในเรจิสเตอร์ของตัวรับวิทยุ เมื่อตรวจสอบแล้วว่ามีข้อมูลก็จะอ่านข้อมูลจากเรจิสเตอร์เป็นชุดอย่างรวดเร็ว(burst) ด้วยเมธอด readBurstRegs ตามด้วยการอ่านค่า RSSI, CLI และ CRC_OKและเมื่ออ่านข้อมูลมาเก็บเรียบร้อยแล้วไมโครคอนโทรเลอร์ก็จะล้างข้อมูลในบัฟเฟอร์รับข้อมูล (Rx FIFO) แล้วจะสั่งให้อุปกรณ์กลับไปอยู่ในสถานะหลับเพื่อประหยัดพลังงานตามเดิม

ดังรายละเอียดที่แสดงในชุดคำสั่งที่ 4.15

```
byte CC1101::receiveData(CCPACKET * packet)
{
  byte val;
  byte rxBytes = readStatusReg(CC1101_RXBYTES);

  if (rxBytes & 0x7F && !(rxBytes & 0x80))
  {
    packet->length = readConfigReg(CC1101_RXFIFO);
    if (packet->length > CC1101_DATA_LEN)
      packet->length = 0;
    else
    {
      readBurstReg(packet->data, CC1101_RXFIFO, packet->length);
      packet->rssi = readConfigReg(CC1101_RXFIFO);
      val = readConfigReg(CC1101_RXFIFO);
```

```
      packet->lqi = val & 0x7F;
      packet->crc_ok = bitRead(val, 7);
    }
  }
  else
    packet->length = 0;
    setIdleState();
    flushRxFifo();
setRxState();
  }
  return packet->length;
}
```

ชุดคำสั่งที่ 4.15 เมธอด receiveData ของคลาส CC1101

4.1.3 การบันทึกข้อมูล

เมื่อสกัดข้อมูลที่รับมาจากทางสัญญาณวิทยุได้แล้ว ไมโครคอนโทรเลอร์จะส่งข้อมูลนี้ ไปยังเครื่องคอมพิวเตอร์แม่ข่ายเพื่อจะได้ทำการบันทึกข้อมูลลงฐานข้อมูล โดยผู้วิจัยได้กำหนดให้ไมโครคอนโทรเลอร์เขียนข้อมูลทั้งหมดลงไปที่พอร์ตอนุกรมด้วยอัตราการส่งข้อมูลที่ 9,600 บิตต่อวินาที และที่คอมพิวเตอร์แม่ข่ายจะมีโปรแกรมที่พัฒนาด้วย Processingคอยฟังข้อมูลที่พอร์ตอนุกรมที่หมายเลขที่กำหนดไว้ หากพบว่ามีข้อมูลมา ก็จะสกัดสายอักขระจากช่องทางการสื่อสารออกมาเก็บไว้ตามรายละเอียดที่ได้กล่าวไว้

เนื่องจากงานวิจัยนี้ต้องการให้ข้อมูลที่ตรวจวัดได้ทั้งหมดเผยแพร่สู่สาธารณะ ดังนั้นจึงจำเป็นที่จะต้องนำข้อมูลที่บันทึกได้ไปจัดเก็บบนเครื่องแม่ข่ายอีกเครื่องหนึ่งที่มีการเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตอยู่ตลอดเวลา ดังนั้นในกรณีนี้ผู้วิจัยจึงเลือกใช้เครื่องแม่ข่ายของสาขาวิชาฯ ที่ตั้งอยู่ที่มหาวิทยาลัยฯ เป็นคอมพิวเตอร์แม่ข่ายเครื่องที่สอง สำหรับใช้เป็นที่บันทึกข้อมูลและแสดงผลข้อมูลที่ตรวจวัดได้ โปรแกรมที่จัดการสิ่งต่าง ๆ เหล่านี้พัฒนาด้วยภาษา PHP และใช้ระบบจัดการฐานข้อมูล MySQL ในการทำงาน โดยสคริปต์ PHP ที่เครื่องแม่ข่ายเครื่องนี้จะทำหน้าที่รับข้อมูลที่ส่งมาผ่านทางโพรโทคอล HTTP และวิธีการส่งข้อมูลแบบ GET

โดยหลังจากที่ตรวจสอบความสมเหตุสมผลของข้อมูลที่ได้รับแล้ว โปรแกรมนี้จะสกัดข้อมูลที่ส่งมาเป็นสายอักขระ(ข้อมูลที่ส่งมานั้นจะแยกแต่ละฟิลด์ด้วยเครื่องหมาย ",") ด้วยการใช้ฟังก์ชั่น

explode เพื่อแยกข้อมูลในสายอักขระออกจากกันหลังจากนั้นสคริปต์นี้จะสร้างคำสั่ง SQL เพื่อบันทึกข้อมูลทั้งหมดลงไปในฐานข้อมูล ดังชุดคำสั่งที่ 4.16

ทั้งนี้นอกจากจะบันทึกข้อมูลไปยังเครื่องแม่ข่ายที่มหาวิทยาลัยแล้ว ระบบจะสำรองข้อมูลเหล่านี้ไว้ที่เครื่องแม่ข่ายอีกด้วย ทั้งนี้เพื่อป้องการการสูญหายเมื่อการเชื่อมต่ออินเทอร์เน็ตขัดข้อง

```
$strSQL = "INSERT INTO wrsensor57 ";
$strSQL .= "VALUES (";
                $strSQL .= "0,";
                $strSQL .= "'" . $sensor_id . "',";
                $strSQL .= "'" . $rdate[0] . "',";
                $strSQL .= "'" . $rdate[1] . "',";
                $strSQL .= "" . $soil_temp . ",";
                $strSQL .= "" . $soil_humd . ",";
                $strSQL .= "" . $air_temp . ",";
                $strSQL .= "" . $air_humd . ",";
                $strSQL .= "" . $light_amb . ",";
                $strSQL .= "" . $ref_volt . "";
        $strSQL .= ");";

        $conn  = mysql_connect(BB_DB_HOST, BB_USER_NAME, BB_PASSWORD);
        if(!$conn)
        {
                die('DB ERROR');
        }
        $resource = mysql_select_db(BB_DB_NAME, $conn);
        $result = mysql_query($strSQL);
        mysql_close($conn);
return packet->length;}
```

ชุดคำสั่งที่ 4.16 คำสั่ง PHP สำหรับการเขียนข้อมูลที่อ่านได้ลงฐานข้อมูล MySQL

4.2 กล่องบรรจุเซนเซอร์

เมื่อพัฒนาเครื่องต้นแบบเสร็จแล้ว ผู้วิจัยจึงได้ออกแบบกล่องสำหรับบรรจุเซนเซอร์ทั้งหมดนี้ โดยได้พยายามออกแบบกล่องบรรจุนี้ตามแนวคิดของสถานีตรวจอากาศภาคพื้นดินขนาดเล็กที่ใช้แนวคิด สตีเว่นสัน สกรีน (Stevenson screen) ซึ่งเป็นกล่องหรือตู้ที่ใช้บรรจุอุปกรณ์วัดข้อมูลสภาพอากาศที่ใช้กันตามสถานีตรวจอากาศโดยทั่วไป โดยกล่องลักษณะนี้จะมีการออกแบบเพื่อพยายามแยกเซนเซอร์ทั้งหลายให้ออกห่างจากความร้อนที่แผ่ออกมาจากวัสดุรอบข้างให้มากที่สุด โดยเฉพาะความร้อนที่แผ่ออกมาจากพื้นดินและตัวกล่องเอง รวมไปถึงการหลีกเลี่ยงไม่ให้เซนเซอร์รับแสงแดดโดยตรง นอกจากนั้นแล้วกล่องนี้จะต้องมีระบบถ่ายเทอากาศที่ดี เพื่อระบายความร้อนและความชื้นภายในกล่องรวมถึงเพื่อให้อากาศจากภายนอกไหลเวียนเข้ามาได้โดยสะดวก ดังนั้นผู้วิจัยจึงพยายามออกแบบกล่องบรรจุเซนเซอร์นี้ตามแนวคิดดังกล่าว โดยพยายามจะเลือกใช้วัสดุที่หาได้โดยทั่วไปและมีราคาไม่แพงมาสร้างกล่องนี้ ซึ่งผู้วิจัยพบว่าวัสดุที่มีความเหมาสมที่จะใช้สร้างกล่องสำหรับงานวิจัยครั้งนี้คือฝาพลาสติกสำหรับปิดช่องระบายอากาศของเครื่องปรับอากาศในโรงงานหรือเตาไฟฟ้า (หรือท่อแอร์) โดยผู้วิจัยได้ใช้ฝาพลาสติกนี้จำนวนสี่แผ่นมาประกอบกันเป็นด้านข้างของกล่องตรวจอากาศแล้วใช้ชิ้นส่วนที่เหลือของฝามาทำเป็นฝากล่องด้านบนและด้านล่าง หลังจากนั้นจึงใช้ด้ามพลาสติกที่แกะออกมาจากไม้ถูพื้นนำมาทำเป็นเสาเพื่อให้กล่องนี้อยู่สูงจากพื้นดินอย่างน้อยประมาณ 1.50 เมตร โดยอุปกรณ์เหล่านี้หาซื้อได้ตามร้านขายอุปกรณ์ก่อสร้างทั่วไป

ภาพที่ 4.4 แสดงการทดลองติดตั้งกล่องในพื้นที่ทดลอง แผงอุปกรณ์ด้านบนกล่องคือแผงเซลล์แสงอาทิตย์ ขนาด 116x160x1.5 มิลลิเมตร ที่ให้แรงดันไฟ 5.5 โวลต์, กระแส 450 มิลลิแอมป์ และให้กำลังไฟ 2.5 วัตต์ กระแสไฟจากแผงเซลล์แสงอาทิตย์นี้ใช้สำหรับการชาร์จแบตเตอรี่แบบลิเที่ยมโพลิเมอร์ 3.7 โวลต์ 900 มิลลิแอมป์ ซึ่งการชาร์จนี้ทำด้วยแผงวงจร Power Cell LiPo Charger/Booster จาก Sparkfun และสายไฟที่เห็นด้านล่างของกล่องคือสายไฟที่เชื่อมไปยังเซนเซอร์วัดข้อมูลดิน

ภาพที่ 4.4 ตัวอย่างการควบคุมผ่านอุปกรณ์เคลื่อนที่
(เข้าชมวิดีโอได้ที่ https://www.youtube.com/watch?v=ZPgn-HGGlys)

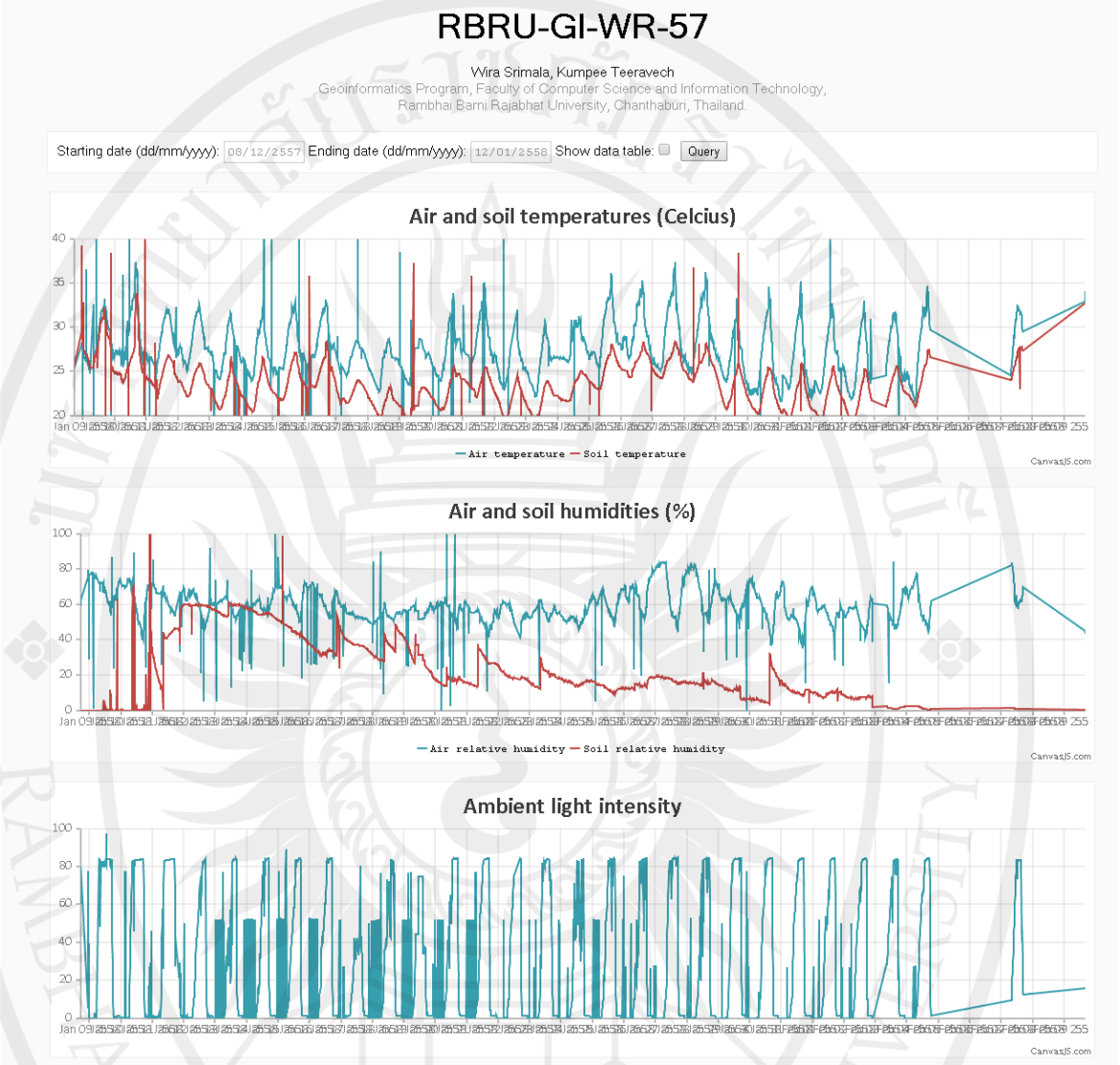
ในทางปฏิบัตินั้น ผู้วิจัยพบปัญหาประการหนึ่งเกี่ยวกับการวัดความชื้นสัมพัทธ์ของดิน นั่นคือการเสื่อมสภาพของเซนเซอร์ ซึ่งอาจจะการเสื่อมสภาพตามปกติของการใช้เซนเซอร์กลุ่มนี้ก็เป็นได้ ทั้งนี้เพราะว่าหน้าสัมผัสของเซนเซอร์ที่เลือกมาใช้นั้นไม่ได้มีการเคลือบสารป้องกันการเสื่อมสภาพจากการสัมผัสความชื้นเป็นเวลานาน ผู้วิจัยพบว่าอุปกรณ์นี้สามารถใช้วัดข้อมูลได้เป็นระยะเวลาประมาณหนึ่งเดือนก่อนที่หน้าสัมผัสจะเสื่อมจนไม่สามารถใช้งานต่อได้อีก ภาพที่ 4.5 แสดงสภาพของเซนเซอร์หลังจากใช้งานหนึ่งเดือน

ภาพที่ 4.5 เซนเซอร์วัดความชื้นสัมพัทธ์ของดินที่เสื่อมสภาพ หลังจากใช้งานประมาณ 30 วัน

4.3 การแสดงผล

การแสดงผลกราฟ ทำโดยใช้คลังคำสั่ง ChartJS[7] ซึ่งเป็นคลังคำสั่งภาษาจาวาสคริปต์ โดยหน้าเว็บแสดงผลนั้น ผู้วิจัยได้ออกแบบให้มีกล่องข้อความให้ผู้ใช้ป้อนช่วงเวลาสำหรับใช้ในการกรองข้อมูลที่ต้องการแสดงและเมื่อผู้ใช้กดปุ่มแสดงผล สคริปต์จะตรวจสอบความสมเหตุสมผลของตัวแปร หลังจากนั้นจึงสร้างใช้ภาษา PHP อ่านข้อมูลจากฐานข้อมูล MySQL ตามเงื่อนไขช่วงเวลาที่ผู้ใช้กำหนด ข้อมูลที่อ่านได้นั้นเก็บอยู่เป็นตัวแปรของภาษา PHP ขั้นตอนต่อไปคือการใช้สคริปต์ ภาษา PHP ในการถ่ายค่าข้อมูลจากภาษา PHP มาเป็นตัวแปรในภาษาจาวาสคริปต์ เพื่อจะได้ใช้ข้อมูลเหล่านี้บรรจุลงไปในแผนภูมิ ดัง ภาพที่ 4.6 แสดงตัวอย่างหน้าต่างเว็บไซต์ที่ใช้แสดงผลข้อมูลที่บันทึกไว้ เส้นสีน้ำเงินแสดงข้อมูลอากาศ เส้นสีแดงแสดงข้อมูลดิน กราฟด้านบนแสดงข้อมูลอุณหภูมิของดินและอากาศกราฟตรงกลางแสดงข้อมูลความชื้นสัมพัทธ์ของดินและอากาศ กราฟด้านล่างสุดแสดงความเข้มแสงแวดล้อม

7 http://www.chartjs.org/



ภาพที่ 4.6 ตัวอย่างการแสดงข้อมูลที่บันทึกไว้ประมาณ 30 วัน (บน) อุณหภูมิ, (กลาง) ความชื้นสัมพัทธ์ และ (ล่าง) ความเข้มแสงแวดล้อม โดยแสดงข้อมูลของอากาศและดินด้วยเส้นสีน้ำเงินและแดงตามลำดับ

ผู้วิจัยพบว่าระบบที่พัฒนานี้ทำงานได้ดี ระบบสามารถวัดและบันทึกค่าต่าง ๆ ได้ตามที่กำหนดไว้ อย่างไรก็ตามผู้วิจัยมักจะพบสัญญาณรบกวนแทรกเข้ามาบ้าง โดยเฉพาะอย่างยิ่งในกรณีของข้อมูลความเข้มแสงแวดล้อมนั้นจะพบมากในช่วงกลางคืน ทั้งนี้อาจจะเป็นเพราะการออกแบบระบบไฟฟ้าของวงจรนั้นยังทำได้ไม่รัดกุมพอ จึงอาจจะทำเกิดการรบกวนกันจากสนามแม่เหล็กไฟฟ้าบนลายวงจรจนไปรบกวนการทำงานของเซนเซอร์ต่าง ๆ ได้

4.4 การควบคุมโซเลนอยด์

ภาพที่ 4.7 ตัวอย่างการควบคุมผ่านอุปกรณ์เคลื่อนที่
(เข้าชมวิดีโอได้ที่ https://www.youtube.com/watch?v=ZPgn-HGGlys)

ภาพที่ 4.7 แสดงตัวอย่างการควบคุมการทำงานของวาล์วเปิดปิดน้ำผ่านอินเตอร์เน็ตโดยใช้อุปกรณ์เคลื่อนที่ ในการศึกษาครั้งนี้ผู้วิจัยใช้สมาร์ทโฟนซัมซุง S3 (Samsung S3) เป็นเครื่องมือในการทดสอบการทำงานและผู้วิจัยใช้เว็บเบาร์เซอร์กูเกิลโครม (Google Chrome) ในการเปิดเว็บเพจพัฒนาด้วยภาษา PHP โดย หน้าต่างหลักของโปรแกรมจะประกอบด้วยปุ่มอินพุตสองปุ่ม ปุ่มหนึ่งทำหน้าเปิดวาล์วน้ำ และอีกปุ่มหนึ่งจะทำหน้าที่ปิดวาล์วน้ำ เมื่อผู้ใช้กดปุ่ม เพจจะส่งข้อมูลกลับมาหาตัวเองก่อนเพื่อให้ตรวจสอบว่าค่าที่กำหนดมานั้นอยู่ในช่วงที่เหมาะสมหรือไม่ โดยงานวิจัยนี้กำหนดให้เว็บเบราเซอร์ส่งข้อมูลไปยังเครื่องแม่ข่ายผ่านโพรโทคอล HTTP ด้วยวิธี GET อย่างไรก็ตามผู้ใช้ทั่วไปสามารถเข้าถึงเว็บเพจนี้ได้ดังนั้นในทางปฏิบัติเราจึงควรมีการตรวจสอบสิทธิของผู้ใช้งานเพื่อป้องกันการสั่งเปิดปิดวาล์วน้ำโดยไม่ได้รับอนุญาต ซึ่งอาจจะทำได้โดยการส่งผ่านชื่อผู้ใช้และรหัสผ่านมาพร้อมกับสถานะของวาล์วน้ำ ทั้งนี้ในกรณีของงานวิจัยนี้จำกัดจำนวนผู้ใช้ที่หนึ่งคนเท่านั้น

เครื่องแม่ข่ายจะรับข้อมูลสถานะของวาล์วน้ำผ่านค่าตัวแปรโกลบอล (Global variable) [$valve_status = $_POST['valve_status'] และเมื่อตรวจความสมเหตุสมผลของตัวแปร [$valve_status (เช่น ไม่เป็นค่าว่างและมีความยาวของข้อความมากกว่าหนึ่งอักขระ) โปรแกรมนี้ก็จะเขียนสถานะของวาล์วตามค่าในตัวแปร $valve_status ลงไว้ในแฟ้มข้อความ ซึ่งในที่นี้กำหนดให้มีชื่อว่า valve_status.txt ซึ่งอาจจะทำได้ด้วยการเรียกใช้ฟังก์ชั่น file_put_contents ของ PHP ดังนี้

```
file_put_contents('valve_status.txt', $valve_status);
```

เครื่องแม่ข่ายที่เชื่อมต่อกับวาล์วน้ำจะมีการร้องขอสถานะของวาล์วน้ำอย่างสม่ำเสมอ ซึ่งในที่นี้กำหนดให้มีการร้องขอทุก ๆ หนึ่งวินาที ซึ่งการร้องขอนี้จะทำโดยโปรแกรมประยุกต์ที่พัฒนาด้วยภาษา Processing โดยได้แยกการทำงานของโปรแกรมนี้ออกเป็นสองขั้นตอนหลัก ๆ ขั้นแรกคือการร้องขอสถานะของวาล์วไปยังเครื่องแม่ข่ายเครื่องที่สอง และขั้นตอนที่สองคือเปิดและปิดวาล์วน้ำตามสถานะที่กำหนด ขั้นตอนแรกนั้นจะทำด้วยการเรียกใช้ฟังก์ชั่น loadString ซึ่งเป็นฟังก์ชั่นที่ออกแบบมาสำหรับการอ่านสายอักขระจากแฟ้มแล้วคืนให้โปรแกรมในรูปแบบแถวลำดับของสายอักขระ โดยมีอาร์กิวเม้นต์เป็นชื่อไฟล์หรือยูอาร์แอล (uniform resource locator (URL)) ที่ต้องการโหลดข้อมูล หลังจากนั้นโปรแกรมก็จะตรวจสอบว่าสถานะที่อ่านมาได้นั้นมีค่าเป็น 0 (หมายถึงปิดวาล์ว) หรือ 1 (หมายถึงเปิดวาล์ว) แล้วจึงส่งค่าไปให้บอร์ด Arduino ผ่านทางพอร์ตอนุกรม ขั้นตอนนี้ทำได้ด้วยการเรียกใช้เมธอด write ของคลาส Serial ของ Processing ตัวอย่างเช่น หากโปรแกรมพบว่าผู้ใช้ต้องการปิดวาล์วน้ำก็จะเรียกเมธอดเป็น serail_port.write (1) และหากโปรแกรมพบว่าผู้ใช้ต้องการปิดวาล์วน้ำก็จะเรียกเมธอดเป็น serail_port.write(0) เป็นต้น เมื่อบอร์ด Arduino ได้รับสัญญาณจากพอร์ตอนุกรมก็จะแปลงค่านี้จากอักขระให้เป็นเลขจำนวนเต็มแล้วส่งสัญญาณออกไปทางขาดิจิทัลที่เชื่อมต่อกับโซเลนอยด์ก็จะทำให้เราสามารถควบคุมการทำงานของโซเลนอยด์ได้ตามต้องการ

ขั้นตอนทั้งหมดนี้ทำอยู่ในฟังก์ชั่น draw() ของProcessing ดังรายละเอียดในชุดคำสั่งต่อไปนี้

```
void draw() {
  int i;
  String msg;
  delay(1000);
  // get current status
  msg = loadStrings("http://127.0.0.1/getValveStatus.php");;
  // extract data
  String d_date = msg[0].substring(0, 10);
  String d_time = msg[0].substring(11, 19);
  String d_status = msg[0].substring(20, 21);
  d_status = trim(d_status);
  valveStatus = int(d_status);
  // Open valve so that the water can flow.
  if(valveStatus == 1) {
    // send message to the target COM port.
    println("open");
    serial_port.write(1);
  } else {
    println("close");
    serial_port.write(0);
  }
}
```

ชุดคำสั่งที่ 4.17 การควบคุมโซเลนอยด์เพื่อเปิดปิดวาล์วน้ำ

ผู้วิจัยพบว่าการเปิดปิดวาล์วน้ำนั้นจะมีการหน่วงเวลาบ้างเล็กน้อย ทั้งนี้เพราะผู้วิจัยกำหนดให้โปรแกรม Processing บนคอมพิวเตอร์แม่ข่ายเครื่องแรก ส่งข้อความร้องขอไปยังแม่ข่ายเครื่องที่สอง (ที่มหาวิทยาลัย) เพื่อตรวจสอบสถานะของวาล์วน้ำ ซึ่งกำหนดให้มีตรวจสอบสถานะนี้ทำทุก ๆ หนึ่งวินาทีนั่นเอง เราอาจจะเพิ่มความถี่ในการตรวจสอบนี้ให้สูงขึ้นเพื่อจะได้ลดเวลาหน่วงดังกล่าวลง แต่ก็อาจจะไปเพิ่มปริมาณการรับส่งข้อมูลโดยไม่จำเป็นได้เช่นเดียวกัน