เครื่องวัดอุณหภูมิเอนกประสงค์

โดย

นายรณชัย อุ่นใจ รหัสนักศึกษา B5008384

นายวิริยะ ชินสมุทร รหัสนักศึกษา B5026500

รายงานนี้เป็นส่วนหนึ่งของการศึกษาวิชา 427499 โครงงานวิศวกรรมโทรคมนาคม

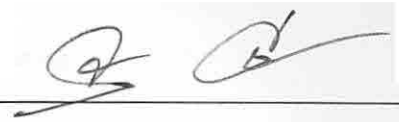
หลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม หลักสูตรปรับปรุง พ.ศ. 2546

สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

ประจำภาคการศึกษาที่ 1 ปีการศึกษา 2553

เครื่องวัดอุณหภูมิเอนกประสงค์

คณะกรรมการสอบโครงงาน



(ผู้ช่วยศาสตราจารย์ เรืออากาศเอก ดร. ประโยชน์ คำสวัสดิ์)
กรรมการ/อาจารย์ที่ปรึกษาโครงงาน

(ผู้ช่วยศาสตราจารย์ ดร. พีระพงษ์ อุฑารสกุล)
กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร. มนต์ทิพย์ภา อุฑารสกุล)
กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ชุติมา พรหมมาก)
กรรมการ

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับรายงานโครงงานฉบับนี้ เป็นส่วนหนึ่งของการศึกษาระดับปริญญาตรี สาขาวิชาวิศวกรรมโทรคมนาคม วิชา 427499 โครงงานวิศวกรรมโทรคมนาคม ประจำปีการศึกษา 2553

โครงงาน เครื่องวัดอุณหภูมิเอนกประสงค์
ผู้ดำเนินงาน 1.นายรณชัย อุ่นใจ รหัสประจำตัว B5008384
2.นายวิริยะ ชินสมุทร รหัสประจำตัว B5026500
อาจารย์ที่ปรึกษา ผศ.ร.อ. ดร.ประโยชน์ คำสวัสดิ์
สาขาวิชา วิศวกรรมโทรคมนาคม
ภาคการศึกษา 1/2553

บทคัดย่อ

(Abstract)

การจัดการและเก็บข้อมูลของอุณหภูมิโดยใช้ ไอซี DS1820 ในการวัดอุณหภูมิ ซึ่งได้มีการเขียนโปรแกรมในการควบคุมการทำงานของ DS1820 ตามที่ต้องการและได้จัดเก็บข้อมูลลงใน Multimedia Cards (MMC) พร้อมทั้งกับส่งข้อมูลที่ได้โดยผ่านทางระบบการสื่อสารแบบไร้สายไปเก็บไว้ยังคอมพิวเตอร์ เพื่อเป็นข้อมูลสำรองในกรณีที่การเก็บข้อมูลลงใน MMC เกิดข้อผิดพลาดขึ้น การเก็บข้อมูลด้วยการใช้วิธีนี้เป็นอีกทางเลือกหนึ่งในการใช้เทคโนโลยีมาช่วยในการจัดเก็บข้อมูล เพื่อให้เกิดประสิทธิภาพและความแม่นยำในการเก็บข้อมูลมากยิ่งขึ้น ทั้งยังทำให้มีความสะดวกรวดเร็วในการวัดอุณหภูมิและจัดเก็บข้อมูล ซึ่งสามารถนำไปใช้กับงานทางด้านเกษตรกรรม เพื่อใช้เก็บข้อมูลอุณหภูมินำไปวิเคราะห์ได้ ทางด้านอุตสาหกรรมที่ต้องการดูอุณหภูมิตลอด และสามารถดูอุณหภูมิย้อนหลัง เพื่อนำมาวิเคราะห์ได้

กิตติกรรมประกาศ

(Acknowledgement)

การจัดทำโครงงานเรื่อง เครื่องวัดอุณหภูมิเอนกประสงค์นี้ได้ประสบความสำเร็จด้วยดีเนื่องจากได้รับความอนุเคราะห์ในการให้คำปรึกษาในด้านต่างๆ ในระหว่างการดำเนินการจากบุคคลหลายท่านที่ได้ให้ความช่วยเหลือและให้คำปรึกษา รวมทั้งข้อเสนอแนะที่เป็นประโยชน์ในการทำโครงงานครั้งนี้ ซึ่งบุคคลเหล่านี้ประกอบไปด้วย

ผศ.ร.อ. ดร.ประโยชน์ คำสวัสดิ์ (อาจารย์ที่ปรึกษาที่ปรึกษาโครงงาน)

นายปัญญา หันตุลา (นักศึกษาปริญญาโท สาขาวิชาวิศวกรรมโทรคมนาคม)

ข้าพเจ้าใคร่ขอขอบพระคุณผู้ที่มีส่วนเกี่ยวข้องทุกท่านที่มีส่วนร่วมในการให้ข้อมูลและเป็นที่ปรึกษาในการทำรายงานฉบับนี้จนเสร็จสมบูรณ์ ตลอดจนให้การดูแลและให้ความเข้าใจเกี่ยวกับพื้นฐานการใช้งานโปรแกรม ซึ่งข้าพเจ้าขอขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้ด้วย

นายรณชัย อุ่นใจ

นายวิริยะ ชินสมุทร

คณะผู้จัดทำ

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมา

ในปัจจุบันโลกของเราเกิดการเปลี่ยนแปลงไปจากเดิมซึ่งสิ่งที่เห็นได้อย่างชัดเจน คือ การเปลี่ยนแปลงทางด้านภูมิอากาศและสภาวะแวดล้อม หากเราต้องการเก็บสถิติความเปลี่ยนแปลงที่เกิดขึ้นในแต่ละวัน คงเป็นการยากหากต้องใช้มนุษย์ในการบันทึกข้อมูลซึ่งจะต้องกระทำซ้ำๆ อยู่ตลอดเวลา นอกจากจะสิ้นเปลืองบุคลากรแล้วอาจเกิดความคลาดเคลื่อนได้เนื่องจากการทำงานติดต่อกันเป็นเวลานาน และยังทำให้บุคลากรเสียสุขภาพได้เนื่องจากในบางครั้งที่จำเป็นต้องวัดอุณหภูมิที่มีความถี่มากๆ ซึ่งก็จะทำให้บุคลากรมีประสิทธิภาพในการทำงานลดน้อยลง ซึ่งคงจะดีกว่าหากมีเครื่องมือที่อำนวยความสะดวกมาทำหน้าที่ในการเก็บข้อมูลแทนบุคลากร ซึ่งมีความแม่นยำสูงกว่า เพราะการเก็บค่าสถิติเป็นสิ่งที่สำคัญต่อการประเมินค่าต่างๆ หากเกิดความผิดพลาดขึ้นในขณะที่ทำการบันทึกหรือลืมเก็บข้อมูลในบางช่วง ซึ่งในกรณีดังกล่าวจะทำให้ผลลัพธ์ที่ได้เกิดความคลาดเคลื่อนหรือผิดพลาดขึ้นได้ และจะนำค่าที่ได้ไปใช้ประโยชน์ได้ไม่เต็มประสิทธิภาพ

โครงงานนี้ผู้จัดทำได้นำเครื่องวัดอุณหภูมิเอนกประสงค์มาแทนบุคลากร เพื่อเป็นการไม่สิ้นเปลืองบุคลากร และเป็นการเล็งเห็นถึงสุขภาพของบุคลากรที่มาทำงาน ณ จุดนี้เป็นเวลานานๆ รวมทั้งยังเป็นการเพิ่มประสิทธิภาพให้กับการเก็บข้อมูลที่ถูกต้องและแม่นยำมากขึ้น เพื่อที่จะได้นำไปใช้ทางด้านการเก็บสถิติเพื่อที่จะนำไปใช้ประโยชน์อย่างอื่นต่อไป

1.2 วัตถุประสงค์

1. เพื่อเพิ่มประสิทธิภาพและความแม่นยำของข้อมูลมากขึ้น
2. เพื่อลดความยุ่งยากในการวัดอุณหภูมิและการจัดเก็บข้อมูลที่ได้
3. เพื่อที่จะสามารถนำข้อมูลย้อนหลังไปประยุกต์ใช้งานในด้านต่างๆ
4. เพื่อเป็นแนวทางในการเลือกใช้วิธีการจัดการและการเก็บข้อมูลอีกทางหนึ่ง

1.3 ขอบเขตงาน

1. ใช้ไอซี DS1820 ในการวัดอุณหภูมิ ซึ่งวัดได้ในช่วง -55 องศาเซลเซียส ถึง +125 องศาเซลเซียส
2. ใช้ไมโคลคอนโทรลเลอร์ตระกูล AVR สามารถเลือกใช้ได้ 2 เบอร์ ATMEGA64 และ ATMEGA128
3. วัดอุณหภูมิโดยกำหนดเวลาแล้วเก็บค่าที่ได้ลง Secure digital card หรือ Multimedia cards และส่งข้อมูลที่ได้ไปยังคอมพิวเตอร์โดยผ่าน ระบบการสื่อสารแบบไร้สาย โดยจัดเก็บเป็นไฟล์ .txt

1.4 ขั้นตอนการดำเนินงาน

1. ศึกษาหัวเรื่องที่ต้องการศึกษานำมาปรึกษา และทำการเลือกหัวข้อที่ต้องการศึกษา เรื่อง เครื่องวัดอุณหภูมิเอนกประสงค์
2. ศึกษาข้อมูลนำมาประกอบกับความรู้ทางทฤษฎีที่เกี่ยวข้องกับหัวข้อ หาข้อมูลเพิ่มเติมเกี่ยวกับหัวข้อที่ต้องการศึกษา
3. ทำการจัดซื้ออุปกรณ์อิเล็กทรอนิกส์ที่ใช้ในการศึกษา และศึกษาอุปกรณ์อิเล็กทรอนิกส์
4. ศึกษาภาษาซีและเขียนโปรแกรมในการควบคุมการทำงาน โดยใช้ภาษาซีในการเขียน
5. ตรวจสอบโปรแกรมและอุปกรณ์อิเล็กทรอนิกส์ตลอดจนการเตรียมเอกสารจัดทำเอกสาร และนำเสนอโครงงาน

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ข้อมูลที่ได้มามีประสิทธิภาพ และความแม่นยำมากขึ้น
2. เพิ่มความสะดวกสบายในการวัดอุณหภูมิ และการจัดเก็บข้อมูล
3. นำข้อมูลไปวิเคราะห์เชิงสถิติด้วยโปรแกรม Excel หรือโปรแกรมอื่นๆ ที่สามารถนำเข้าไฟล์ .txt ได้
4. จะได้รับประโยชน์จากเครื่องวัดอุณหภูมิเอนกประสงค์ และระบบการสื่อสารแบบไร้สาย ทั้งยังสามารถนำไปประยุกต์ใช้ต่ออย่างอื่นได้

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 บทนำ

ในบทนี้เราจะกล่าวถึง ไมโครคอนโทรลเลอร์ คุณสมบัติของ MCU (Multipoint Control Unit) เบอร์ ATmega64 คุณสมบัติและข้อมูลด้านเทคนิคของเซนเซอร์วัดอุณหภูมิ DS18S20 การใช้งาน RTC (Real time clock) ด้วย DS1307 ชุดเชื่อมต่อหน่วยความจำ SD/MMC CARD และจอแสดงผลแบบ LCD (Liquid crystal display)

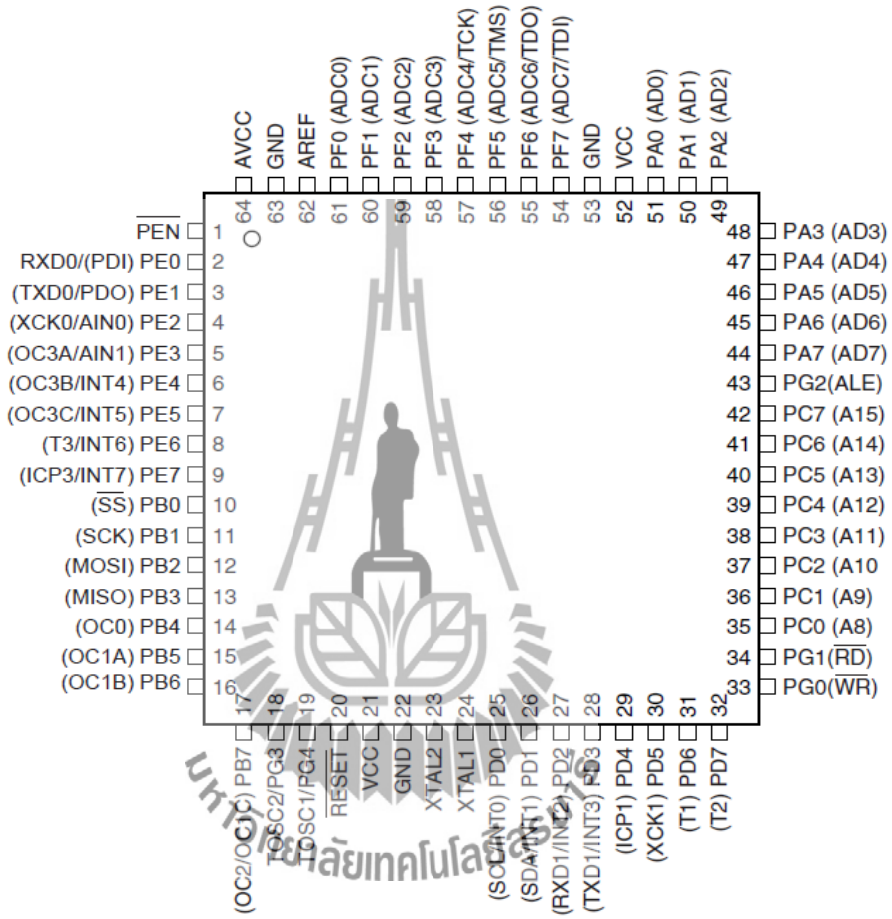
2.2 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ (Microcontroller) มาจากคำ 2 คำ คำหนึ่งคือ ไมโคร (Micro) หมายถึงขนาดเล็ก และคำว่า คอนโทรลเลอร์ (controller) หมายถึงตัวควบคุมหรืออุปกรณ์ควบคุม ดังนั้น ไมโครคอนโทรลเลอร์ จึงหมายถึงอุปกรณ์ควบคุมขนาดเล็ก แต่ในตัวอุปกรณ์ควบคุมขนาดเล็กนี้ ได้บรรจุความสามารถที่คล้ายคลึงกับระบบคอมพิวเตอร์ ที่คนโดยส่วนใหญ่คุ้นเคย กล่าวคือ ภายในไมโครคอนโทรลเลอร์ได้รวมเอาซีพียู หน่วยความจำ และพอร์ต ซึ่งเป็นส่วนประกอบหลักสำคัญของระบบคอมพิวเตอร์เข้าไว้ด้วยกัน โดยทำการบรรจุเข้าไว้ในตัวถังเดียวกัน

ในโครงงานนี้เป็นบอร์ดไมโครคอนโทรลเลอร์ในตระกูล AVR ของบริษัท Atmel ซึ่งบอร์ดนี้เลือกใช้ MCU เบอร์ ATmega64 ขนาด 64 Pin โดยการออกแบบโครงสร้างของบอร์ดนั้นจะเน้นเรื่องการจัดวางบอร์ดให้มีขนาดเล็กเพื่อให้ง่ายต่อการนำไปประยุกต์ใช้งาน โดยได้นำ MCU มาจัดวงจรร่วมกับอุปกรณ์พื้นฐานที่จำเป็น และจัดขาออกมาใช้งานภายนอก ซึ่งการจัดเรียงขาสัญญาณจะทำการจัดเรียงอย่างเป็นระเบียบเพื่อให้สามารถต่อใช้งานได้โดยสะดวก โดยที่ตัวบอร์ดจะใช้ไฟเลี้ยง +5V

2.3 คุณสมบัติของ MCU เบอร์ATmega64

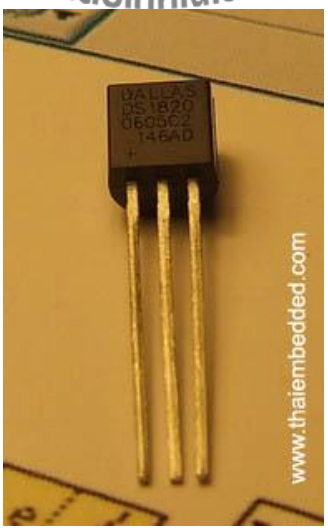
TQFP/MLF



รูปที่ 2.1 Pinout ATmega64

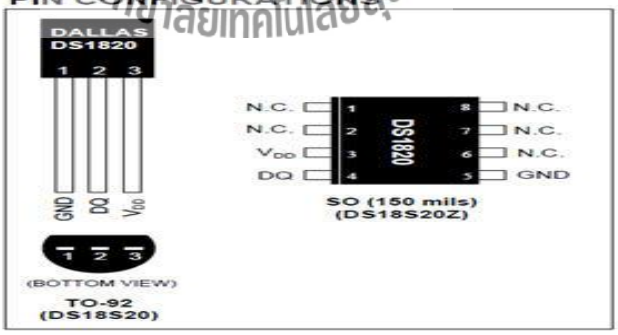
1. ความเร็วสัญญาณนาฬิกา Crystal 16 MHz
2. รองรับการโปรแกรมแบบ SPI และ JTAG (ต้องใช้ร่วมกับบอร์ด ET-AVR START KIT V1.0)
3. Power supply ใช้แรงดันไฟฟ้า 4.5 V - 5.5 V
4. ภายใน MCU มีหน่วยความจำโปรแกรมแบบ Flash ขนาด 64 KB หน่วยความจำข้อมูล RAM ขนาด 4 KB หน่วยความจำข้อมูลถาวรแบบ EEPROM ขนาด 2 KB สามารถลบและเขียนซ้ำได้กว่า 100,000 ครั้ง
5. จำนวน I/O สูงสุดถึง 53 I/O Pins ซึ่งขาสัญญาณ I/O จะมีการใช้งานร่วมกันของ Function อื่น ๆ อีกดังนี้
   5.1 SPI จำนวน 1 ช่อง , I2C จำนวน 1 ช่อง , 10-Bit ADC จำนวน 8 ช่อง
   5.2 Programmable Serial USARTs จำนวน 2 ช่อง
   5.3 Timers/Counters 8-Bit จำนวน 2 ช่อง , Timers/Counters 16-Bit จำนวน 2 ช่อง , 8-Bit PWM 2 ช่อง , Watchdog timer , Real time counter
6. ทนอุณหภูมิใช้งานระหว่าง -40 °C ถึง +85°C (ถ้าใช้งานที่อุณหภูมิ 85°C จะสามารถใช้งานได้ถึง 20 ปี และถ้าใช้งานที่อุณหภูมิ 25°C จะสามารถใช้งานได้ถึง 100 ปี)

2.4 คุณสมบัติและข้อมูลด้านเทคนิคของเซนเซอร์วัดอุณหภูมิ DS18S20



รูปที่ 2.2 IC DS1820

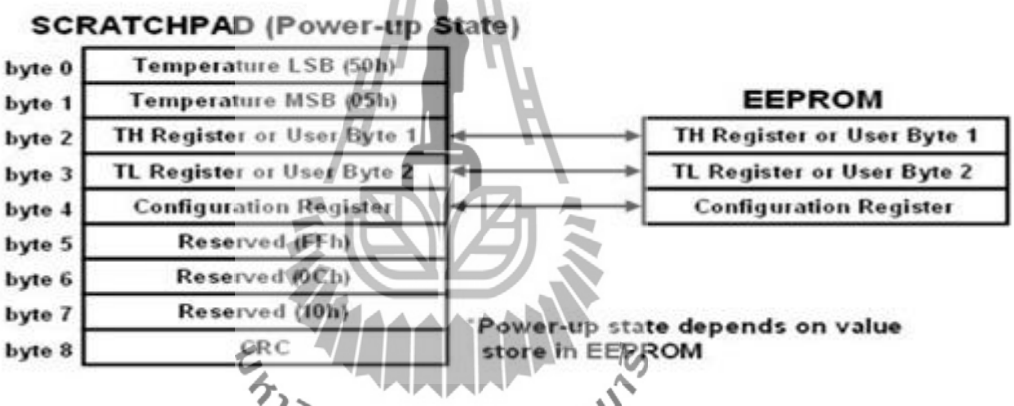
ไอซี DS1820 เป็นไอซีที่มีระบบการสื่อสารข้อมูลอนุกรมแบบหนึ่งสายซึ่งถือได้ว่าเป็นระบบที่มีความชาญฉลาด และใช้จำนวนสายสัญญาณเพียง 1 เส้นเท่านั้น โดยไม่ต้องมีสายสัญญาณนาฬิกามาควบคุมจังหวะการถ่ายทอดข้อมูลเหมือนกับระบบสื่อสารข้อมูลอนุกรมในแบบอื่น สายข้อมูลจะทำหน้าที่เสมือนเป็นสายสัญญาณนาฬิกาในตัว ส่วนค่าของข้อมูลจะพิจารณาจากลักษณะของรูปสัญญาณที่ปรากฏบนสายสัญญาณในแต่ละช่องของเวลาซึ่งเรียกว่า ไทม์สล็อต (Time slot) โดยคาบเวลาต่ำสุดและสูงสุดของสถานะต่าง ๆ ในการสื่อสารข้อมูลในแต่ละไทม์สล็อตมีการกำหนดขอบเขตไว้อย่างชัดเจนการถ่ายทอดข้อมูลจะเกิดขึ้นในแต่ละไทม์สล็อตนั้น รูปแบบการถ่ายทอด ข้อมูลจะเป็นแบบอะซิงโครนัสในระดับบิต ไม่มีการกำหนดความยาวของข้อมูลเป็นระดับไบต์ระบบสื่อสารแบบนี้เหมาะที่จะใช้ในการสื่อสารข้อมูลระหว่างไอซีแผงวงจรเดียวกัน เหตุผลที่เลือก DS18S20 เนื่องจากสิ่งที่เราต้องการในการวัดอุณหภูมินี้เราต้องการระบบบัสที่สามารถเดินสายได้ในระยะไกลซึ่งเหมาะสำหรับระบบบัสแบบ 1-Wire ดูจะเหมาะกว่าในแง่ของความสะดวกในการใช้งานระยะห่างของแต่ละจุดที่วัดความแม่นยำของค่าที่วัดได้ใช้ไอซีเบอร์ DS18S20 ซึ่งมีระดับความผิดพลาดที่ 0.5 องศาเซลเซียส DS18S20 เป็น IC วัดอุณหภูมิแบบดิจิตอลของ Dallas semiconductor สามารถวัดอุณหภูมิเป็นหน่วยองศา C ในช่วง -55°C ถึง 125°C ที่ความละเอียด 9-12 บิต และมีความแม่นยำอยู่ที่ 0.5°C ในช่วง -10°C ถึง 85°C ในกรณีที่เป็นตัวถังแบบ TO-92 นั้นจะมีโครงสร้าง และขาดังแสดงในรูปที่ 2.3



PIN	SYMBOL	Description
1	GND	Ground
2	DQ	Data Input/ Output pin
3	Vdd	Optional Vdd pin

รูปที่ 2.3 โครงสร้าง และขาของ DS18B20 ตัวถังแบบ TO-92

การสื่อสารและควบคุม DS18S20 นั้นสามารถทำได้โดยใช้บัสข้อมูลแบบ 1-wire ของ Dallas semiconductor ซึ่งใช้สายสัญญาณเพียงแค่เส้นเดียวเท่านั้น ภายใน DS18S20 แต่ละตัวมีโค๊ดประจำตัวขนาด 64 บิต ทำให้สามารถใช้งาน DS18S20 หลายตัวทำงานบนบัสแบบ 1 wire พร้อมกันได้ นอกจากนี้ DS18S20 ยังสามารถทำงานในโหมดพาราสิต (Parasite Power Mode) ซึ่งเป็นการทำงานโดยไม่ใช้ไฟเลี้ยง แต่ใช้พลังงานจากสายสัญญาณ 1-wire ซึ่งมีประโยชน์มากสำหรับการวัดอุณหภูมิระยะไกล หรือในการใช้งานในที่ๆ มีเนื้อที่จำกัด โครงสร้างรีจิสเตอร์ภายในของ DS18S20 มีลักษณะดังแสดงในรูปที่ 2.4 จะเห็นได้ว่าประกอบไปด้วย SRAM Scratchpad ขนาด 9 ไบต์ และ EEPROM ขนาด 3 ไบต์ ซึ่งใช้เก็บค่าอุณหภูมิสูงสุด (TH) ต่ำสุด (TL) สำหรับเปรียบเทียบการเกิดสัญญาณเตือน และรีจิสเตอร์ควบคุม (Configuration register)



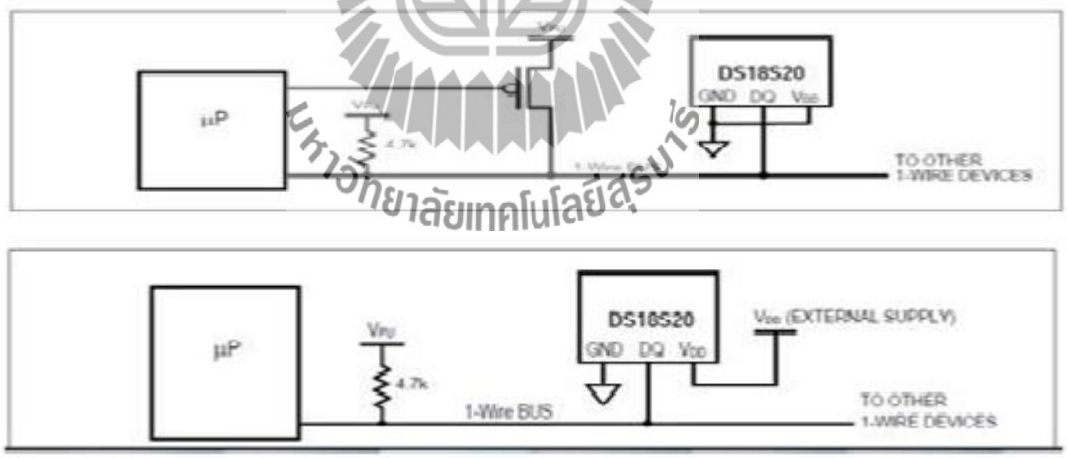
รูปที่ 2.4 โครงสร้างรีจิสเตอร์ภายใน DS18S20

ข้อมูลอุณหภูมิที่วัดได้จะถูกเก็บอยู่ในรีจิสเตอร์ Temperature ซึ่งมีขนาด 16 บิต ดังแสดงในรูปที่ 2.5 ถ้าข้อมูลอุณหภูมิเป็นบวก S จะเป็น “1” แต่ถ้าข้อมูลอุณหภูมิเป็นลบ S จะเป็น “0” ในกรณีที่ DS18S20 ทำงานในโหมดความละเอียด 12 บิต บิตทุกบิตในรีจิสเตอร์ Temperature จะถูกใช้หมด แต่ในกรณีที่ทำงานในโหมด 9-11 บิต บิตล่าง (บิต 0 – บิต 2) จะไม่ถูกใช้งาน ซึ่งในการกำหนดโหมดความละเอียดการทำงานของ DS18S20 นั้นสามารถกำหนดได้ที่รีจิสเตอร์ Configuration ซึ่งโดยปกติเริ่มต้น DS18S20 จะทำงานในโหมด 12 บิต

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
MS Byte	S	S	S	S	S	2^6	2^5	2^4

รูปที่ 2.5 โครงสร้างภายในรีจิสเตอร์ Temperature LSB และ MSB

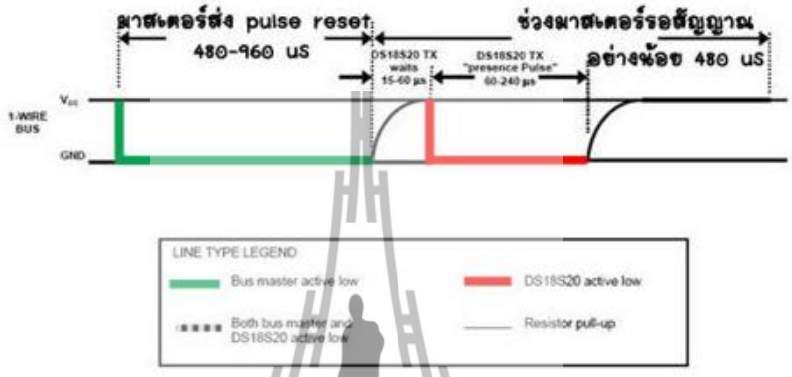
การสื่อสารแบบ 1-wire เป็นระบบบัสข้อมูลแบบ Half-duplex นั้นคือสามารถสื่อสารได้ 2 ทิศทาง แต่ไม่สามารถรับ และส่งข้อมูลพร้อมกันในช่วงเวลาเดียวกันได้ ระบบบัสมีการทำงานเป็นแบบ Master/Slave โดยอุปกรณ์ Master จะเป็นตัวควบคุมสถานะ และจังหวะการรับส่งของบัสข้อมูล ในขณะที่อุปกรณ์ Slave จะทำงานตามการควบคุมของอุปกรณ์ Master เท่านั้น ในการใช้งานบัสแบบ 1 wire นี้ สายสัญญาณข้อมูล DQ จะต้องมีสภาวะปกติที่ลอจิกสูง สามารถทำได้โดยการต่อตัวต้านทานประมาณ 5 กิโลโอห์ม พูลอัปไว้กับไฟเลี้ยง หรือในกรณีที่ใช้บัสแบบ 1 wire ต่อร่วมกับอุปกรณ์ DS18S20 หลายตัว ก็สามารถทำได้ดังแสดงในรูปที่ 2.6



รูปที่ 2.6 การต่อใช้งาน DS18S20

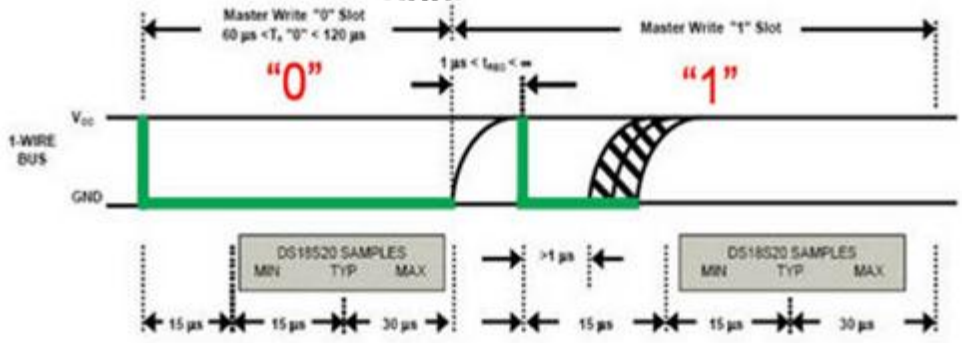
2.4.1 หลักการทำงานของเซนเซอร์วัดอุณหภูมิ DS18S20

ในกระบวนการเริ่มต้นการสื่อสารแบบ 1-wire ทั้งหมดนั้น อุปกรณ์ Master ต้องขอเริ่มการสื่อสารด้วยการสร้าง Reset pulse ก่อน เมื่ออุปกรณ์ Slave ได้รับ Reset pulse ก็จะสร้าง Presence pulse เพื่อตอบรับการขอเริ่มการสื่อสารนั้น ซึ่งมีรายละเอียดของช่วงเวลาต่าง ๆ ดังแสดงในรูปที่ 2.7



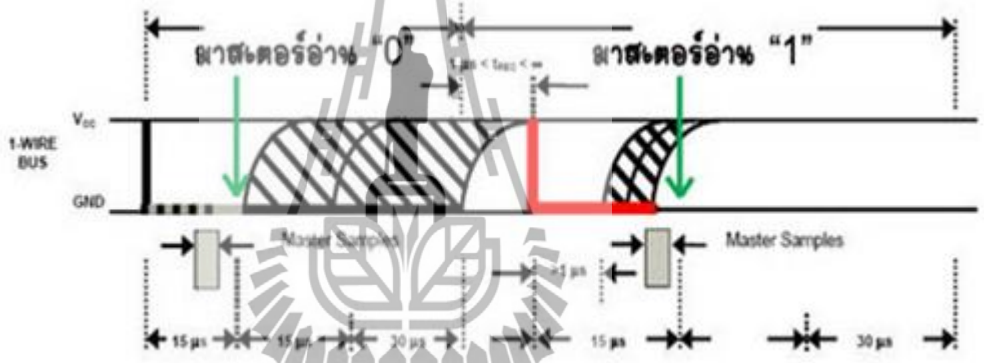
รูปที่ 2.7 การเริ่มการติดต่อสื่อสารแบบ 1-wire ด้วย Reset pulse และ Presence pulse

ในการเขียนข้อมูลแบ่งออกเป็น 2 ชนิดคือการเขียนข้อมูล “0” และการเขียนข้อมูล “1” ดังแสดงในรูปที่ 2.8 การเขียนข้อมูลลง DS18S20 ต้องใช้ช่วงเวลาของไทม์สล๊อตอย่างต่ำ 60 usec และต้องมีช่วงเวลาระหว่างไทม์สล๊อตอย่างต่ำ 1 usec



รูปที่ 2.8 การเขียนข้อมูลจาก Master ลง DS18S20

การเขียนข้อมูลทั้ง 2 ชนิด เริ่มแรกอุปกรณ์ Master ต้องดึงสัญญาณบนบัส 1-wire ลงมาให้อยู่ในสถานะลอจิกต่ำก่อน ในกรณีที่ต้องการเขียนข้อมูล “0” ลงใน DS18S20 อุปกรณ์ Master ต้องดึงสัญญาณบนบัสให้เป็นลอจิกต่ำต่อ จนกว่าจะครบช่วงเวลาไทม์สล๊อต (อย่างต่ำ 60 usec) ส่วนในกรณีที่ต้องการเขียนข้อมูล “1” ลง DS18S20 อุปกรณ์ Master ต้องปล่อยบัส เพื่อให้บัสกลับไปอยู่ในสถานะลอจิกสูงก่อนการ Sampling ของ DS18S20 ซึ่งจะอยู่ในช่วง 15 usec-60 usec หลังจากที่อุปกรณ์ Master ดึงสัญญาณบัส 1-wire ลงมาในการอ่านค่าภายใน SRAM ของ DS18S20 สามารถทำได้ก็ต่อเมื่ออุปกรณ์ Master ได้เขียนข้อมูลเพื่อขอทำการอ่านค่าใน SRAM (Read Scratchpad) ซึ่งมีค่าเป็น 0xBE ลงไปที่ DS18S20 เสียก่อน จากนั้นจึงเริ่มอ่านข้อมูลจากบัส 1-wire โดยไทม์สล๊อตของการอ่านต้องมีช่วงเวลาอย่างต่ำ 60 usec และต้องมีช่วงเวลาระหว่างไทม์สล๊อตอย่างต่ำ 1 usec ดังแสดงในรูปที่ 2.8

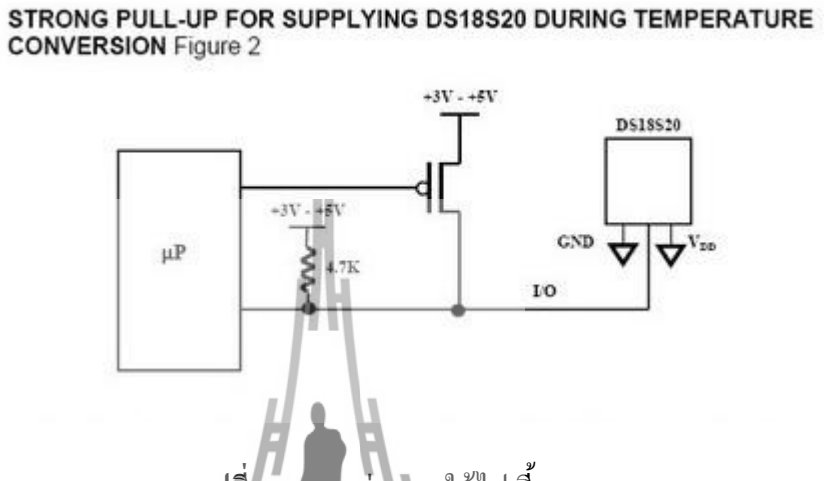


รูปที่ 2.9 การอ่านข้อมูลจาก DS18S20

การอ่านข้อมูลจากบัส 1-wire เริ่มแรกอุปกรณ์ Master จะต้องดึงบัส 1-wire ลงให้อยู่ในสถานะลอจิกต่ำเป็นช่วงเวลาอย่างน้อย 1 usec จากนั้นจึงค่อยปล่อยบัส ในกรณีที่ DS18S20 ส่งข้อมูล “0” DS18S20 จะดึงบัสให้เป็นลอจิกต่ำจนจนกว่าจะสิ้นสุดไทม์สล๊อตถึงจึงจะปล่อยบัสให้กลับไปอยู่ในสถานะลอจิกสูง ส่วนในกรณีที่ DS18S20 ส่งข้อมูล “1” DS18S20 จะปล่อยบัสให้อยู่ในสภานะลอจิกสูงตลอด ในการ Sample เพื่อรับข้อมูลจาก DS18S20 ควรทำภายใน 15 usec หลังจากจุดเริ่มของไทม์สล๊อตดังแสดงในรูปที่ 2.9

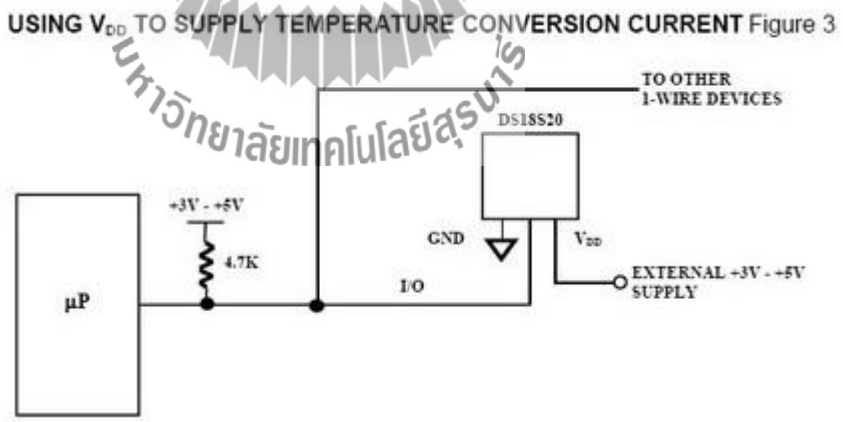
2.4.2 การต่อใช้งาน DS18S20 จะมีอยู่ 2 วิธีด้วยกัน

1. ใช้ไฟเลี้ยงจาก R Pull-up (PARASITE POWER) วิธีนี้ขา VDD จะต้องต่อลง GND ทำให้ต่อสายเพียง 2 เส้นเท่านั้น ดังแสดงในรูปที่ 2.10



รูปที่ 2.10 การต่อแบบใช้ไฟเลี้ยง R Pull-up

2. ต่อไฟเลี้ยงให้กับขา VDD (External power supply) วิธีนี้จะเป็นที่นิยมใช้กันมากกว่า ดังแสดงในรูปที่ 2.11



รูปที่ 2.11 การต่อแบบจ่ายไฟเลี้ยงให้กับขา VDD

ค่าอุณหภูมิที่อ่านได้จาก DS1820 จะมีความละเอียดสเตปละ 0.5°C ขนาด 9 บิต

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	LSB
MSb			(unit = °C)				LSb	
S	S	S	S	S	S	S	S	MSB

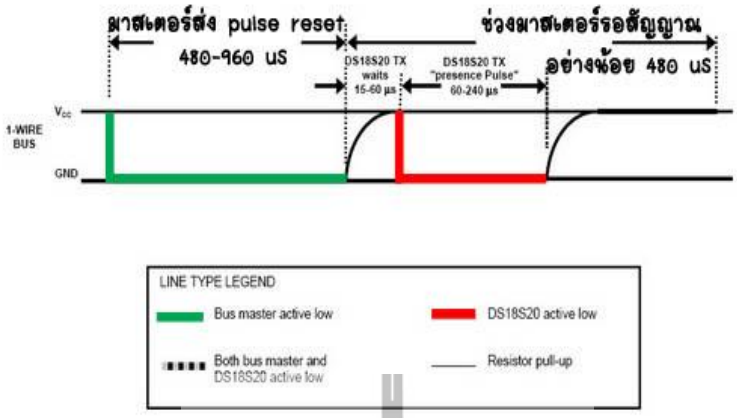
TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+85°C	0000 0101 0101 0000	0550h*
+125°C	0000 0000 1111 1010	00FAh
+25.0°C	0000 0000 0011 0010	0032h
+0.5°C	0000 0000 0000 0001	0001h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1111	FFFFh
-25.0°C	1111 1111 1100 1110	FFCEh
-55°C	1111 1111 1001 0010	FF92h

*The power on reset register value is +85°C

รูปที่ 2.12 ตารางความสัมพันธ์ระหว่างอุณหภูมิกับค่าที่อ่านได้

2.4.3 การทำงานของ Alarm TH, TL

หลังจากที่ DS1820 ได้ทำการแปลงอุณหภูมิออกมาเป็นตัวเลขแล้ว ค่าอุณหภูมิก็จะถูกนำไปเปรียบเทียบกับ TH, TL ค่าอุณหภูมิมีขนาด 9 บิต ส่วนค่า TH, TL มีขนาด 8 บิต แล้วจะเปรียบเทียบกัน โดยมันก็จะตัดบิต LSB ของ DS1820 ทิ้งไป หลังจากเปรียบเทียบแล้วค่าอุณหภูมิมากกว่า TH หรือน้อยกว่า TL ค่า Alarm Flag ก็จะถูกเซต เมื่อใดก็ตามที่ Alarm Flag ถูกเซตอยู่ มันก็จะแสดงตัวออกมาให้รู้ในช่วงของคำสั่ง Search command เราจึงเข้าไปอ่าน DS1820 ตัวนั้นได้ทันที โดยไม่ต้องไล่อ่าน DS1820 ทีละตัว

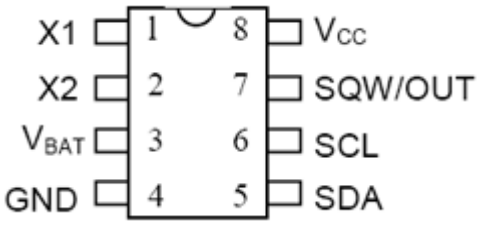


รูปที่ 2.13 Timming การรีเซต และการตอบกลับ (Presence Pulse) ของ DS1820

2.5 การใช้งาน RTC (Real Time Clock) ด้วย DS1307

ระบบฐานเวลา เป็นสิ่งสำคัญที่สามารถนำไปใช้ในอุปกรณ์อีเล็กโทรนิกส์ได้หลากหลาย ภายในไมโครคอนโทรเลอร์เองก็มีไทเมอร์เพื่อใช้ในการจับเวลา หรือนำไปใช้เป็นฐานเวลาจริงได้เช่นกัน แต่เนื่องจากไมโครคอนโทรเลอร์สามารถทำงานได้ต่อเมื่อมีไฟเลี้ยงเท่านั้น ดังนั้นการใช้ไทเมอร์ของไมโครคอนโทรลเลอร์ สร้างฐานเวลาจริงจึงไม่ เหมาะสมในบางแอปพลิเคชัน

DS1307 เป็น IC ฐานเวลาของดัลลัสเซมิคอนดักเตอร์ (Dallas semiconductor) มีบัสรับส่งข้อมูลแบบ I2C ซึ่งเป็นแบบ 2 wire สามารถสื่อสารได้ 2 ทิศทาง (bi-direction bus) ฐานเวลาของ DS1307 นั้นสามารถ เก็บข้อมูล วินาที, นาที, ชั่วโมง, วัน, วันที่, เดือน และปี ได้ ระบบเวลาสามารถทำงานโหมดรูปแบบ 24 ชั่วโมง หรือ 12 ชั่วโมง AM/PM ก็ได้ ภายมีระบบตรวจจับแหล่งจ่ายไฟ โดยถ้าแหล่งจ่ายไฟหลักถูกตัดไป DS1307 สามารถสวิตซ์ไปใช้ไฟจากแบตเตอรี่ และทำงานต่อไป โดยที่ยังสามารถรักษาข้อมูลไว้ได้ โครงสร้างมีขาทั้งหมด 8 ขาดังแสดงในรูปที่ 2.14 และมีรายละเอียดการทำงานของขาต่าง ๆ ดังนี้



รูปที่ 2.14 ตำแหน่งขาไอซี RTC DS1307

VCC: ใช้ต่อไฟเลี้ยง +5V

GND: ใช้ต่อกราวด์

VBAT: ใช้ต่อกับแบตเตอรี่ 3V เพื่อรักษาการทำงาน ในกรณีที่ไม่มีไฟเลี้ยงจ่าย

SDA: ขารับส่งข้อมูลด้วยระบบบัส I^2C

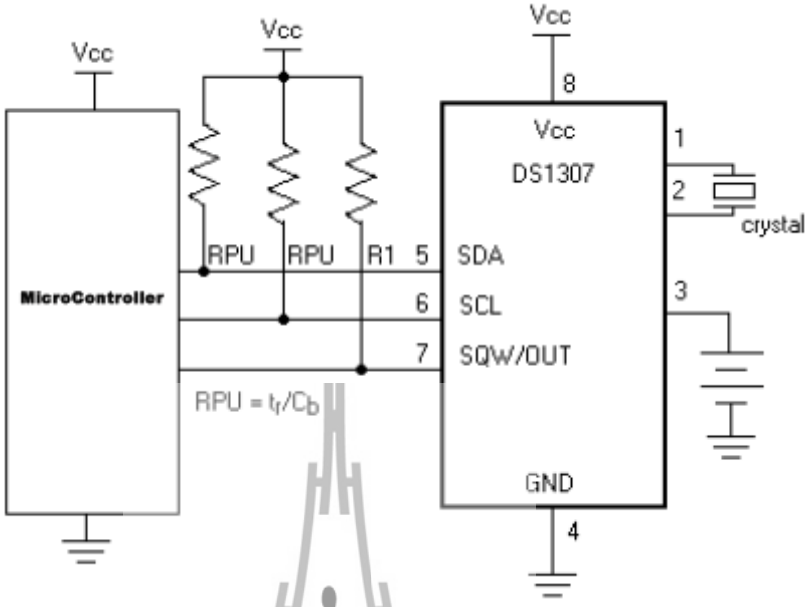
SCL: ขาสัญญาณนาฬิกาสำหรับการรับส่งข้อมูลด้วยระบบบัส I^2C

SQW/OUT: ขาเอาต์พุตสัญญาณ Square wave สามารถเลือกความถี่ได้

X1, X2: ใช้ต่อกับคริสตอลความถี่มาตรฐาน 32.768 kHz เพื่อสร้างฐานเวลาจริงให้กับ IC

ระบบบัสข้อมูลแบบ I^2C (Inter-IC Communication) ได้ถูกพัฒนาขึ้นโดยบริษัทฟิลิปส์ (Phillips) การรับส่งข้อมูลใช้สายสัญญาณเพียงแค่ 2 เส้น คือสายสัญญาณข้อมูล SDA (Serial Data line) และสายสัญญาณนาฬิกา SCL (Serial Clock line) มีการทำงานเป็นแบบ Master, Slave โดยอุปกรณ์ที่ทำหน้าที่เป็น Master (ไมโครคอนโทรลเลอร์) จะควบคุมการรับส่งข้อมูล และควบคุมสัญญาณนาฬิกาบน SCL ส่วนอุปกรณ์ Slave (DS1307) นั้นจะทำงานภายใต้การควบคุมของอุปกรณ์ Master

การต่อใช้งานร่วมกับไมโครคอนโทรลเลอร์ด้วยระบบบัส I^2C นั้นสามารถทำได้โดยต่อตัวต้านทาน Pull up ดังแสดงในรูปที่ 2.15 ในกรณีที่ต้องการต่อร่วมกับอุปกรณ์ Slave หลายตัว ก็สามารถทำได้โดยต่ออุปกรณ์ Slave ขนานกันไป การติดต่อสื่อสารระหว่างอุปกรณ์ Master กับ Slave แต่ละตัวนั้น จะถูกแยกโดย Address ของอุปกรณ์ Slave ซึ่งจะถูกส่งจากอุปกรณ์ Master ไปยังอุปกรณ์ Slave ก่อนเริ่มการรับส่งข้อมูล



รูปที่ 2.15 การเชื่อมต่อ DS1307 เข้ากับไมโครคอนโทรลเลอร์ด้วยระบบบัสแบบ I^2C

2.5.1การรับส่งข้อมูลแบบ I^2C

การรับส่งข้อมูลแบบ I^2C นั้นมีข้อกำหนดอยู่ 2 ประการด้วยกันคือ

1. การรับส่งข้อมูลจะเริ่มขึ้นได้เมื่อบัสมีสถานะว่างเท่านั้น
2. ในช่วงที่ทำการรับส่งข้อมูลอยู่ สายสัญญาณ SDA ต้องไม่เปลี่ยนสถานะในช่วงที่ SCL มีสถานะเป็นลอจิก “1” ถ้า SDA มีการเปลี่ยนสถานะในช่วงที่ SCL เป็นลอจิก “1” จะถือว่าเป็นสัญญาณควบคุมการรับส่งข้อมูล

2.5.2 สถานะของการรับส่งข้อมูลแบบ I^2C

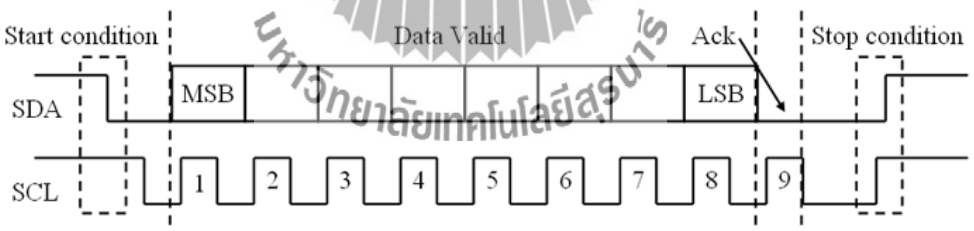
สถานะของการรับส่งข้อมูลแบบ I^2C สามารถแบ่งออกได้เป็น 5 สถานะด้วยกันดังแสดงในรูปที่ 2.16 **และมีรายละเอียดดังนี้**

1. สถานะว่าง (Bus not busy): สัญญาณ SDA และ SCL มีระดับสัญญาณเป็น High

2. เริ่มส่งข้อมูล (Start data transfer): มีการเปลี่ยนระดับสัญญาณของ SDA จาก High เป็น Low ในขณะที่ SCL มีระดับสัญญาณเป็น High ค้างไว้
3. หยุดส่งข้อมูล (Stop data transfer): มีการเปลี่ยนระดับสัญญาณของ SDA จาก Low เป็น High ในขณะที่ SCL มีระดับสัญญาณเป็น High ค้างไว้
4. รับส่งข้อมูล (Data valid): มีการรับส่งข้อมูลผ่านสายสัญญาณ SDA โดยข้อมูลแต่ละบิตจะถูกส่งในช่วงที่ SCL มีระดับเป็น High โดยในช่วงที่ SCL มีสถานะเป็น High อยู่นั้น SDA จะต้องไม่เกิดการเปลี่ยนระดับสัญญาณ

SDA จะเปลี่ยนระดับของสัญญาณ ในช่วงที่ SCL มีระดับสัญญาณเป็น Low เท่านั้น ตามมาตรฐานการส่งข้อมูล แบบ I^2C นี้สามารถส่งข้อมูลด้วยความถี่สัญญาณนาฬิกาสูงสุด 100 kHz ที่โหมดการทำงานธรรมดา และ 400 kHz ที่โหมดการทำงานแบบเร็ว แต่สำหรับ DS1307 สามารถทำงานได้ในโหมดธรรมดาเท่านั้น

ตอบรับ (Acknowledge): เกิดขึ้นหลังจากที่มีการรับส่งข้อมูลครบแล้ว โดยอุปกรณ์ Master ต้องสร้างสัญญาณ Clock บน SCL เพิ่มอีกลูก อุปกรณ์ที่เป็นตัวรับข้อมูลจะดึงระดับสัญญาณบน SDA ให้เป็น Low เพื่อให้ตัวส่งรับรู้ว่าตัวรับได้รับข้อมูลครบแล้ว

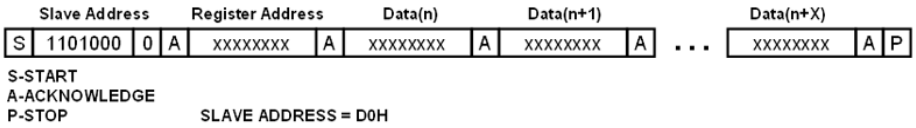


รูปที่ 2.16 การรับส่งข้อมูลผ่านบัส I^2C

ในการรับส่งข้อมูลผ่านบัส I^2C อุปกรณ์ Master จะเป็นผู้สร้างสัญญาณ Clock บน SDA และเป็นตัวควบคุมสถานะ Start และ Stop เพื่อควบคุมการรับส่งข้อมูลทั้งหมด

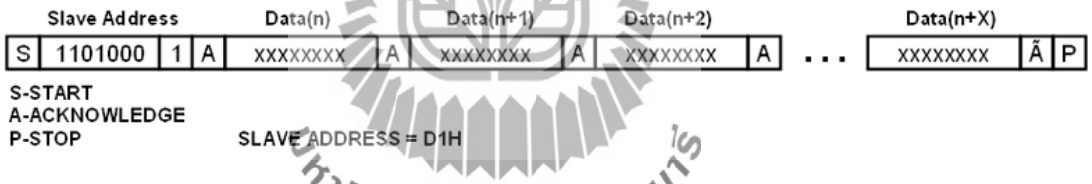
การส่งข้อมูลไปยังอุปกรณ์ DS1307 ดังแสดงในรูปที่ 2.17 ไมโครคอนโทรลเลอร์ต้องสร้างสภาวะ Start ก่อน จากนั้นต้องส่ง Address ของ DS1307 ขนาด 7 บิตซึ่งมีค่าเป็น 1101000 และตามด้วยบิตระบุทิศทางของข้อมูล ในกรณีที่เป็นการเขียนข้อมูลลง DS1307 จะต้องเป็น “0” จากนั้นไมโครคอนโทรเลอร์จะต้องส่งตำแหน่ง Address ภายในรีจิสเตอร์ของ DS1307 ที่ต้องการเขียน

ข้อมูลลง แล้วจึงค่อยเขียนข้อมูลลง โดยในการส่งข้อมูลแต่ละไบต์จะต้องรอบิต Ack จาก DS1307 ทุกไบต์ เมื่อส่งจนครบแล้ว ถึงจะสร้างสภาวะ Stop เพื่อกลับสู่สถานะว่าง



รูปที่ 2.17 การเขียนข้อมูลอุปกรณ์ Slave ผ่านบัส I^2C

การรับข้อมูลจากอุปกรณ์ Slave ดังแสดงในรูปที่ 2.18 เริ่มแรกไมโครคอนโทรลเลอร์ต้องสร้างสภาวะ Start ก่อน จากนั้นต้องส่ง Address ของ DS1307 ขนาด 7 บิตซึ่งมีค่าเป็น 1101000 และตามด้วยบิตระ บุทิศทางของข้อมูล ในกรณีที่เป็นการอ่านข้อมูลจาก DS1307 จะต้องเป็น “1” จากนั้นจึงค่อยรับข้อมูลจากอุปกรณ์ Slave ทีละไบต์ โดยตำแหน่งที่อ่านเข้ามาจะขึ้นอยู่กับตำแหน่งรีจิสเตอร์พอยท์เตอร์ ซึ่งจะเป็นตำแหน่งท้ายสุดที่ได้ทำการเขียนข้อมูลไว้ เมื่ออ่านข้อมูลครบแต่ละไบต์อุปกรณ์ Master ต้องส่ง Acknowledge บิตกลับไปให้อุปกรณ์ Slave ด้วย ในกรณีที่เป็นไบต์สุดท้าย อุปกรณ์ Master ต้องส่ง “not acknowledge” กลับไป



รูปที่ 2.18 การอ่านข้อมูลจากอุปกรณ์ Slave ผ่านบัส I^2C

ภายใน DS1307 มีรีจิสเตอร์ภายในใช้เก็บข้อมูลเวลาขนาด 7 ไบต์ 00H-06H ดังแสดงในรูปที่ 2.19 ข้อมูลค่าเวลา และวันที่จะถูกเก็บอยู่ในรูปของเลขฐาน 10 สามารถเลือกได้ว่าให้ทำงานแบบ 12 ชั่วโมง หรือ 24 ชั่วโมง โดยกำหนดที่บิตที่ 6 ที่แอดเดรส 02H โดยถ้าเป็น “1” จะเป็นการทำงานในโหมด 12 ชั่วโมง และเมื่อเลือกแบบ 12 ชั่วโมง ที่บิต 5 ในแอดเดรส 02H นั้นจะใช้แสดงค่า AM/PM โดยถ้าบิตนี้เป็น “1” จะเป็น PM ในกรณีที่แสดงแบบ 24 ชั่วโมง บิตนี้จะใช้ในการแสดงค่าของหลักสิบในของหน่วยชั่วโมงด้วย

	BIT7							BIT0	
00H	CH	10 SECONDS			SECONDS				00-59
	0	10 MINUTES			MINUTES				00-59
	0	12/24	10 HR A/P	10 HR	HOURS				01-12 00-23
	0	0	0	0	0	DAY			1-7
	0	0	10 DATE		DATE				
	0	0	0	10 MONTH	MONTH				01-12
	10 YEAR				YEAR				00-99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

รูปที่ 2.19 รีจิสเตอร์ภายในไอซีฐานเวลา DS1307

ที่แอดเดรส 07H เป็นรีจิสเตอร์ควบคุมการทำงานของ SQW/OUT โดยมีรายละเอียดดังนี้

OUT (Out control): ใช้ควบคุมเอาต์พุต

SQWE (Square Wave Enable): ใช้ควบคุมออสซิลเลเตอร์ภายใน DS1307 โดยถ้าบิตนี้เป็น “1” จะเป็นการเปิดออสซิลเลเตอร์

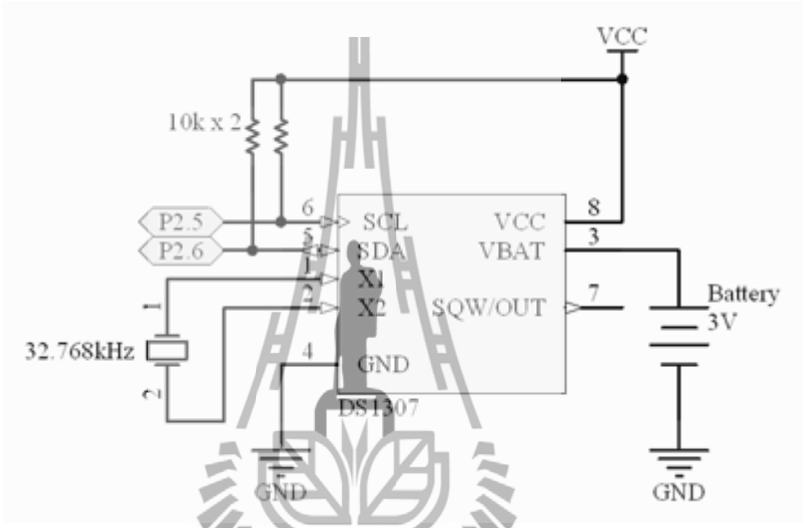
RS (Rate Select): ใช้ควบคุมความถี่ของ Square Wave เมื่อเปิดการทำงานของออสซิเลเตอร์ โดยสามารถปรับเปลี่ยนความถี่ได้ 4 ความถี่ด้วยกันดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 การควบคุมความถี่ออสซิลเลเตอร์ด้วยการเซตบิต RS1, RS0

RS1	RS0	SQW OUTPUT FREQUENCY
0	0	1 Hz
0	1	4.096 kHz
1	0	8.192 kHz
1	1	32.768 kHz

ในโครงงานนี้ได้ต่อ DS1307 กับไมโครคอนโทรลเลอร์ P89V51RD2 โดยใช้พอร์ต P2.5 และ P2.6 ของไมโครคอนโทรเลอร์เป็นบัส I^2C ต่อกับ SCL และ SDA ของ DS1307 ดังแสดงในรูปที่ 2.20 ส่วนขาสัญญาณ SQW/OUT นั้นไม่ได้ใช้สร้างสัญญาณให้ไมโครคอนโทรลเลอร์ แต่ใช้การวนลูปคอยตรวจสอบค่าภายในรีจิสเตอร์ของ DS1307 แทน

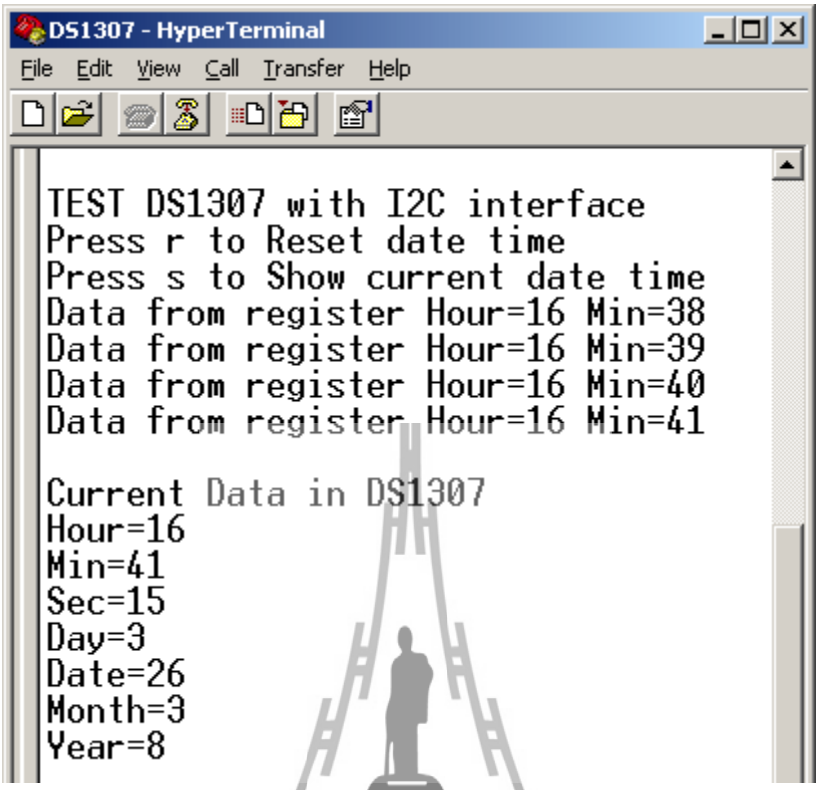
ในการควบคุมการทำงานของโปรแกรม และแสดงผล ได้ใช้โปรแกรม Hyper terminal เป็นโปรแกรมติดต่อผ่านพอร์ตอนุกรมด้วยอัตราข้อมูล 9600 bps ดังแสดงในรูปที่ 2.20



รูปที่ 2.20 วงจรใช้งาน DS1307

ในส่วนโปรแกรมที่ใช้ในการทดสอบ ได้เขียนให้ไมโครคอนโทรลเลอร์คอยตรวจสอบพอร์ตอนุกรม และรีจิสเตอร์ภายใน DS1307 ในกรณีที่มีข้อมูล “r” เข้ามาทางพอร์ตอนุกรม ไมโครคอนโทรลเลอร์จะเขียนข้อมูลเวลา , วันที่, เดือน และปี ที่เก็บอยู่ใน Flash memory ลงในรีจิสเตอร์ของ DS1307 รวมทั้งตั้งให้ DS1307 ทำงานในโหมด 24 ชั่วโมง ส่วนในกรณีที่มีข้อมูล “s” เข้ามาทางพอร์ตอนุกรม ไมโครคอนโทรลเลอร์จะแสดงข้อมูลเวลา , วันที่, เดือน และปี ออกมาทางพอร์ตอนุกรมดังแสดงในรูปที่ 8

ไมโครคอนโทรลเลอร์จะคอยวนตรวจสอบรีจิสเตอร์ภายใน DS1307 ที่ตำแหน่ง 01H ซึ่งใช้เก็บค่าเวลาหน่วยนาที เมื่อค่าภายในรีจิสเตอร์นี้เปลี่ยนไป ไมโครคอนโทรลเลอร์จะอ่านข้อมูลเวลา ชั่วโมง และนาที ภายในรีจิสเตอร์ของ DS1307 ขณะนั้นออกมา และส่งข้อมูลนั้นออกมาแสดงผลทางพอร์ตอนุกรมดังแสดงในรูปที่ 2.21

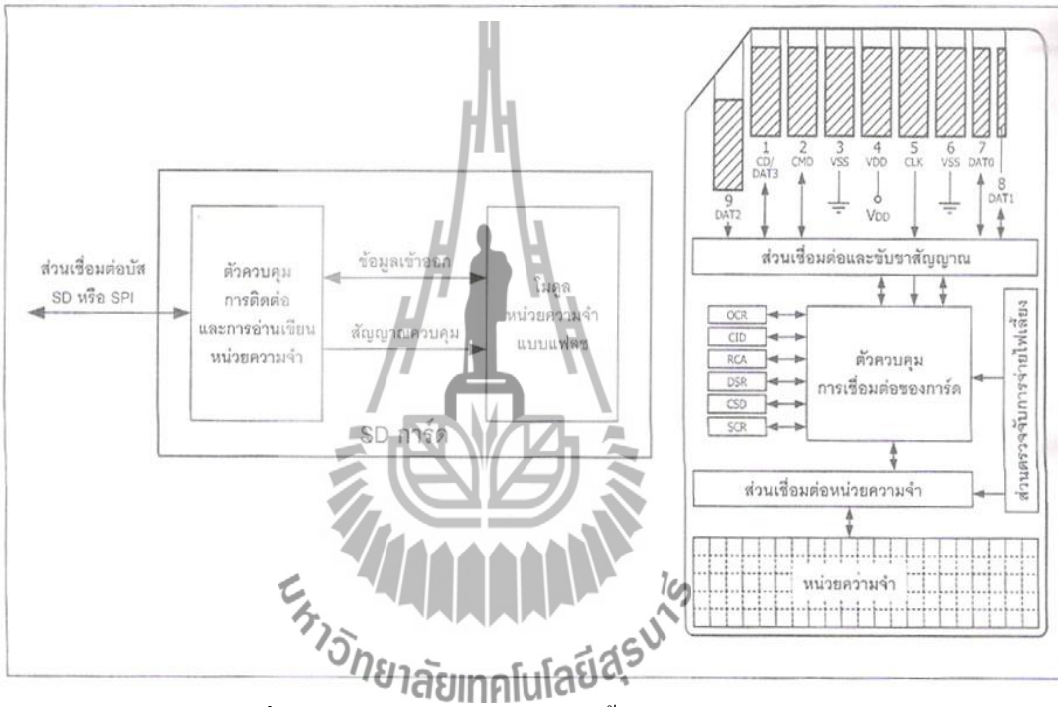


รูปที่ 2.21 การทดสอบการใช้งาน DS1307 ผ่านบัส I^2C

2.6 ชุดเชื่อมต่อหน่วยความจำ SD/MMC CARD

2.6.1 ความรู้เบื้องต้นเกี่ยวกับ SD การ์ด

SD การ์ดเป็นหน่วยความจำแบบเขียนและลบใหม่ได้แบบหนึ่งที่ใช้เทคโนโลยีหน่วยความจำแบบแฟลช (Secure digital card) มีลักษณะการทำงานและการติดต่อคล้ายกับการ์ดหน่วยความจำแบบ MMC (Multimedia card) หากแต่ใน SD การ์ดได้บรรจุส่วนการรักษาข้อมูลเข้าไปเพิ่มเติม ในรูปที่ 2.22 แสดงไดอะแกรมการทำงานของ SD การ์ด จะเห็นว่า มีส่วนประกอบ 2 ส่วนคือโมดูลหน่วยความจำแบบแฟลช และตัวควบคุม การติดต่อกับ SD หรือบัส SPI



รูปที่ 2.22 ไดอะแกรมการทำงานเบื้องต้นของ SD การ์ด

2.6.2 คุณสมบัติเด่นของ SD การ์ด

SD การ์ดเกิดขึ้นจากความร่วมมือของ 3 บริษัทคือ Matsushita Electric Industrial (MEI), SanDisk Corporation (SanDisk) และ Toshiba Corporation (Toshiba) มีการกำหนดคุณสมบัติต่างๆรวมถึงมาตรฐานการติดต่อที่ชัดเจนภายใต้การกำกับดูแลโดย SD Card Association (www.sdcard.org)

ในปัจจุบัน SD การ์ดได้รับความนิยมสูงมากโดยเฉพาะในอุปกรณ์สารสนเทศสมัยใหม่ ไม่ว่าจะเป็นกล้องดิจิตอล โทรศัพท์เคลื่อนที่ เครื่องเล่น MP3 เป็นต้น ทั้งนี้เนื่องจาก SD การ์ดได้รับการออกแบบให้มีความโดดเด่นในทุกด้านที่หน่วยความจำชั้นดีพึงมี 5 ประการ ดังนี้

คุณสมบัติทางเทคนิคที่สำคัญของ SD การ์ด

1. สามารถเก็บข้อมูลได้ถึง 8 GB (ในขณะจัดทำเอกสารนี้)
2. รองรับการติดต่อแบบหนึ่งสายสัญญาณ และแบบ 4 สายสัญญาณ รวมทั้งแบบบัส SPI
3. สามารถป้องกันการคัดลอกข้อมูลลิขสิทธิ์ได้
4. สามารถลบ-เขียนใหม่ในแต่ละเซกเตอร์ได้ 100,000 ครั้ง
5. สามารถเก็บรักษาข้อมูลได้นานมากกว่า 10 ปี

2.6.3 ระบบบัสที่ใช้ติดต่อกับ SD การ์ด

การติดต่อกับ SD การ์ดสามารถกระทำได้ 2 วิธีคือ

1.ผ่านทางบัส SD

2.บัส SPI

ขาสัญญาณของ SD การ์ด

ขาสัญญาณมาตรฐานของ SD การ์ดมีทั้งสิ้น 9 ขา โดยมีลักษณะเป็นหน้าสัมผัสโลหะ ดังแสดงในรูปที่ 2.22 ส่วนการกำหนดชื่อ และหน้าที่ของขาสัญญาณจะแตกต่างกันตามรูปแบบของการติดต่อดังสรุปได้ในตารางที่ 2.2 และ 2.3 โดยในตารางที่ 2.3 เป็นการจัดขาเมื่อติดต่อ SD การ์ดด้วยบัส SD ส่วนตารางที่ 2.4 เป็นการจัดขาเมื่อทำงานผ่านบัส SPI

ตารางที่ 2.2 สรุปข้อมูลสำคัญของการติดต่อกับ SD การ์ดทั้งแบบบัส SD และ SPI

การติดต่อ SD การ์ดด้วยบัส SD	**การติดต่อ SD การ์ดด้วยบัส SPI**
ใช้สายสัญญาณ 6 เส้น สัญญาณนาฬิกา สัญญาณคำสั่ง (Command) สัญญาณข้อมูล 4 สาย	ใช้สายสัญญาณอนุกรม 3 เส้น สัญญาณนาฬิกา สัญญาณข้อมูลเข้า(DI) สัญญาณข้อมูลออก(DO) สัญญาณเลือกการ์ดCS
มีการป้องกันความผิดพลาดในการถ่ายทอดข้อมูล	สามารถเลือกหรือไม่เลือกการป้องกันความผิดพลาดในการถ่ายทอดข้อมูล
สามารถถ่ายทอดข้อมูลได้ทั้งแบบล็อกเดี่ยวหรือหลายบล็อก	สามารถถ่ายทอดข้อมูลได้ทั้งแบบบล็อกเดี่ยวหรือหลายบล็อก

ตารางที่ 2.3 เป็นการจัดขาเมื่อติดต่อ SD การ์ดด้วยบัส SD

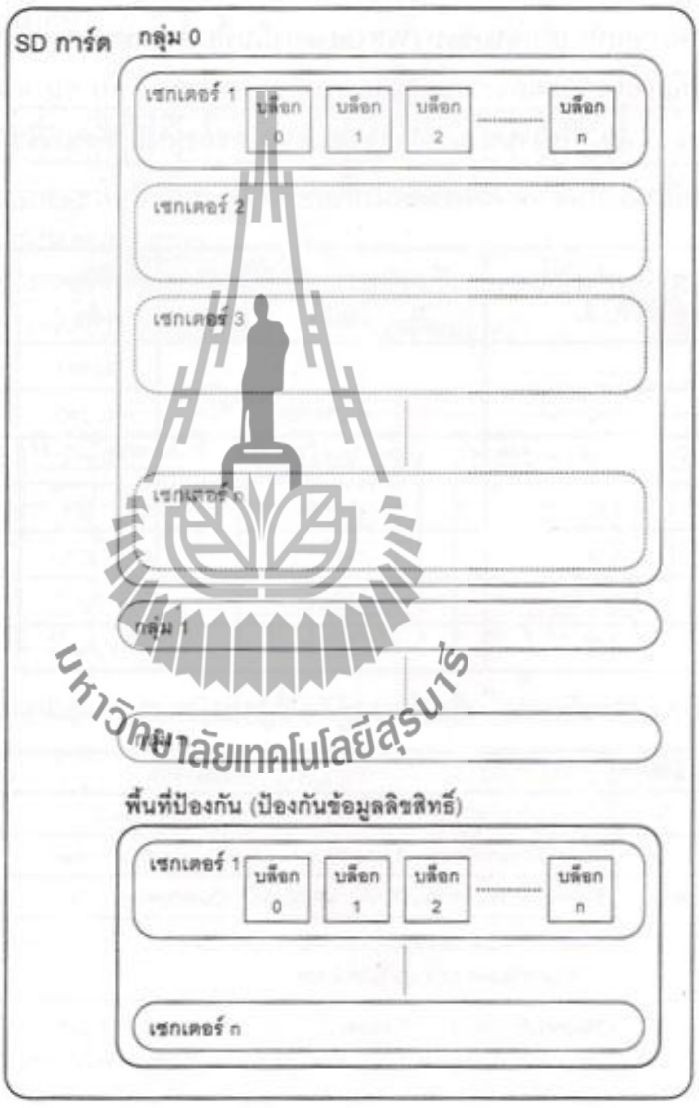
หมายเลขขา	ชื่อขาสัญญาณ	ชนิด	คำอธิบาย
1	CD/DAT3	อินพุต/เอาต์พุต	ตรวจสอบการ์ด/สายข้อมูลบิต 3
2	CMD	อินพุต/เอาต์พุต	สัญญาณคำสั่ง/ตรวจสอบการตอบสนอง
3	Vss	สายแหล่งจ่ายไฟ	กราวด์
4	VDD	สายแหล่งจ่ายไฟ	ไฟเลี้ยง
5	CLK	อินพุต	สัญญาณนาฬิกา
6	Vss	สายแหล่งจ่ายไฟ	กราวด์
7	DAT0	อินพุต/เอาต์พุต	สายข้อมูลบิต0
8	DAT1	อินพุต/เอาต์พุต	สายข้อมูลบิต1
9	DAT2	อินพุต/เอาต์พุต	สายข้อมูลบิต2

ตารางที่ 2.4 เป็นการจัดขาเมื่อติดต่อ SD การ์ดด้วยบัส SPI

หมายเลขขา	ชื่อขาสัญญาณ	ชนิด	คำอธิบาย
1	CS	อินพุต	สัญญาณเลือกติดต่อ(ลอจิก “0”)
2	DI	อินพุต	สัญญาณข้อมูลเข้าจากโฮสต์
3	Vss1	สายแหล่งจ่ายไฟ	กราวด์
4	VDD	สายแหล่งจ่ายไฟ	ไฟเลี้ยง
5	CLK	อินพุต	สัญญาณนาฬิกา
6	Vss2	สายแหล่งจ่ายไฟ	กราวด์
7	DO0	เอาต์พุต	สัญญาณข้อมูลส่งออกจากการ์ด
8	RSV	อินพุต	สำรองไว้
9	RSV	อินพุต	สำรองไว้

2.6.4 การจัดแบ่งพื้นที่ของ SD การ์ด

หน่วยที่เล็กสุดของการถ่ายทอดข้อมูลใน SD การ์ดคือ 1 ไบต์ (byte) ส่วนการถ่ายทอดข้อมูลจริงนั้น ควรกระทำในลักษณะบล็อกข้อมูล โดยสามารถกำหนดขนาดของบล็อกได้ โดยในแต่ละบล็อกสามารถบรรจุข้อมูลได้หลายๆ ไบต์ แต่โดยปกติแล้วมักจะเลือกใช้ที่บล็อกละ 512 ไบต์ ทั้งนี้เพื่อให้สอดคล้องกับระบบ FAT (File Allocation Table) หรือตารางสำหรับจัดวางแฟ้มข้อมูลซึ่งใช้ในระบบคอมพิวเตอร์



รูปที่ 2.23 การจัดแบ่งพื้นที่ของ SD การ์ด

จากรูปที่ 2.23 มีการจัดสรรเป็น 3 ส่วนหลักคือ บล็อกข้อมูล เป็นกลุ่มของข้อมูลที่ได้รับการกำหนดขนาดจากผู้ใช้งาน และนำไปใช้ในคำสั่ง และเขียนบล็อกข้อมูล สำหรับการกำหนดและตรวจสอบขนาดของบล็อกข้อมูลสามารถกระทำได้ที่รีจิสเตอร์

เซกเตอร์ เป็นหน่วยของพื้นที่ข้อมูลใน SD การ์ดที่สัมพันธ์กับคำสั่งลบ ใน 1 เซกเตอร์มีหลายบล็อกข้อมูล โดยได้รับการกำหนดมาตายตัวจากผู้ผลิต ผู้ใช้งานสามารถตรวจสอบขนาดของเซกเตอร์ได้จากรีจิสเตอร์

กลุ่มป้องกันการเขียน (WP Group) เป็นพื้นที่ของหน่วยความจำที่ได้รับการจัดแบ่งให้น้อยที่สุดใช้เพื่อบรรจุลิขสิทธิ์ที่ไม่ต้องการให้เกิดการเขียนทับ ขนาดของพื้นที่จะ ได้รับการกำหนดมาตายตัวเช่นกันผู้ใช้งานสามารถตรวจสอบขนาดของพื้นที่ได้จากรีจิสเตอร์ CSD

2.6.5 รีจีสเตอร์ของ SD การ์ด

มีทั้งหมด 6 ตัว โดยเป็นรีจิสเตอร์หลักที่ใช้ 4 ตัว,รีจิสเตอร์พิเศษ 1 ตัว และรีจิสเตอร์เสริมอีก 1 ตัว ดังแสดงในตารางที่ 2.5

ตารางที่ 2.5 การแสดงรีจิสเตอร์ใน SD การ์ด

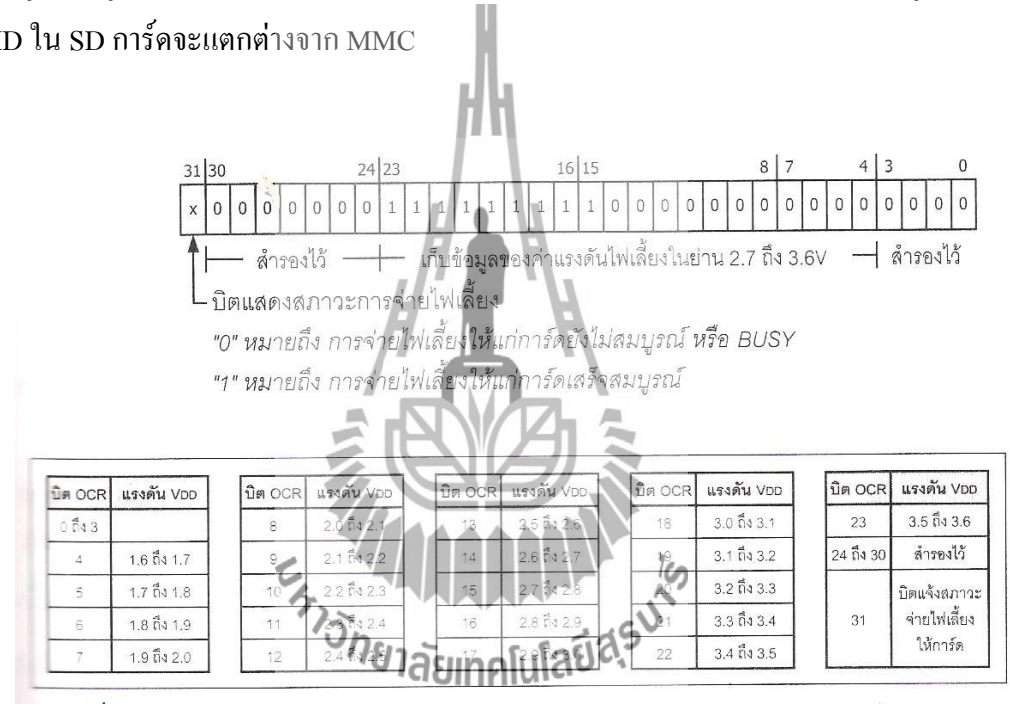
ชื่อริจิสเตอร์	ขนาด	รายละเอียด
OCR	32 บิต	รีจิสเตอร์เก็บสภาวะการทำงาน(Operation Condition Register)
CID	128 บิต	รีจิสเตอร์เก็บค่ารหัสเฉพาะตัวของ SD การ์ด (Card Identification number)
CSD	128 บิต	รีจิสเตอร์เก็บข้อมูลคุณสมบัติเฉพาะของ SD การ์ด (Card Specific Data)
RCA	16 บิต	รีจิสเตอร์กำหนดค่าแอดเดรสแบบสัมพัทธ์ (Relative Card Address) สามารถกำหนดได้จากโฮสต์ไม่ใช่ในกรณีติดต่อ SD การ์ดในโหมด SPI
SCR	64 บิต	รีจิสเตอร์เก็บค่าคุณสมบัติพิเศษของ SD การ์ด(SD Configuration Register) เป็นรีจิสเตอร์พิเศษ ไม่มีใช้ใน MMC (เนื่องจาก MMC มีการติดต่อคล้ายกับ SD การ์ดมาก ดังนั้นใน MMC จะมีรีจิสเตอร์ 4 ตัว ให้ใช้งาน)
DSR	16 บิต	รีจิสเตอร์เสริมสำหรับเก็บค่าคุณสมบัติของไดรเวอร์ทางเอาต์พุต (Driver Stage Register) - ใช้กับSDIO การ์ด

➔รีจิสเตอร์ OCR (Operating condition register)

เป็นรีจิสเตอร์เก็บข้อมูลของค่าแรงดันไฟเลี้ยงของ SD การ์ด สำหรับตรวจสอบแรงดันของ SD การ์ด ปกติแรงดันไฟเลี้ยงของ SD การ์ดอยู่ในช่วง 2.7 V ถึง 3.6 V ดังนั้นค่าของรีจิสเตอร์ OCR ควรเท่ากับ

➔รีจิสเตอร์ CID (Card identification register)

เป็นรีจิสเตอร์ที่มีความยาว 16 ไบต์ ใช้ในการเก็บข้อมูลเฉพาะของ SD การ์ด ซึ่งกำหนดมาจากผู้ผลิต ผู้ใช้งานไม่สามารถทำการเปลี่ยนแปลงได้ โดยค่าและความหมายของข้อมูลในรีจิสเตอร์ CID ใน SD การ์ดจะแตกต่างจาก MMC



บิต OCR	แรงดัน VDD
0 ถึง 3	
4	1.6 ถึง 1.7
5	1.7 ถึง 1.8
6	1.8 ถึง 1.9
7	1.9 ถึง 2.0

บิต OCR	แรงดัน VDD
8	2.0 ถึง 2.1
9	2.1 ถึง 2.2
10	2.2 ถึง 2.3
11	2.3 ถึง 2.4
12	2.4 ถึง 2.5

บิต OCR	แรงดัน VDD
13	2.5 ถึง 2.6
14	2.6 ถึง 2.7
15	2.7 ถึง 2.8
16	2.8 ถึง 2.9
17	2.9 ถึง 3.0

บิต OCR	แรงดัน VDD
18	3.0 ถึง 3.1
19	3.1 ถึง 3.2
20	3.2 ถึง 3.3
21	3.3 ถึง 3.4
22	3.4 ถึง 3.5

บิต OCR	แรงดัน VDD
23	3.5 ถึง 3.6
24 ถึง 30	สำรองไว้
31	บิตแจ้งสภาวะจ่ายไฟเลี้ยงให้การ์ด

รูปที่ 2.24 ความสัมพันธ์ของบิตข้อมูลในรีจิสเตอร์ OCR กับแรงดันของ SD การ์ด

➔รีจิสเตอร์ CSD (Card specific data)

เป็นรีจิสเตอร์ขนาด 16 ไบต์ (128 บิต) ที่ใช้เก็บข้อมูลคุณสมบัติเฉพาะของ SD การ์ด ซึ่งมีรายละเอียดค่อนข้างมากเพราะรีจิสเตอร์นี้บรรจุข้อมูลเกี่ยวกับความจุ,อัตราเร็วในการถ่ายทอดข้อมูล , แรงดันและกระแสไฟฟ้า ในขณะอ่านและเขียนข้อมูล,รูปแบบของไฟล์,การป้องกันข้อมูล, การลบและข้อมูลเกี่ยวกับการเขียนข้อมูลลงใน SD การ์ด สำหรับในการทดลองนี้เลือกใช้ 2 ข้อมูล คือ C_SIZE (บิต 73: 62) และ C_SIZE_MUL(บิต 49:47) เพื่อนำมาคำนวณหาความจุของ SD การ์ด ที่ติดต่อด้วย

ส่วนข้อมูลอื่นๆ เพิ่มเติมของรีจิสเตอร์ตัวนี้สามารถอ่านได้จากไฟล์ดาต้าชี ทของ SD การ์ดในซีดีรอมที่จัดมาพร้อมกับบอร์ด JX-2148

➔รีจิสเตอร์ RCA (Relative card address)

เป็นรีจิสเตอร์ขนาด 16 บิตใช้เก็บค่าแอดเดรสของหน่วยความจำแบบสัมพันธ์ ซึ่งทางโฮสต์ (หมายถึง คอมพิวเตอร์หรือไมโครคอนโทรลเลอร์) สามารถเลือกกำหนดได้ อย่างไรก็ตาม หากเลือกการติดต่อ SD การ์ดแบบ SPI จะไม่สามารถติดต่อกับรีจิสเตอร์ตัวนี้ได้

➔รีจิสเตอร์ SCR (SD Configuration register)

เป็นรีจิสเตอร์ขนาด 64 บิต ที่ใช้เก็บค่าคุณสมบัติพิเศษของ SD การ์ด ที่เพิ่มเติมนอกเหนือไปจากที่เก็บในรีจิสเตอร์ CSD ซึ่งข้อมูลทั้งหมดนี้ได้รับการกำหนดมาจากผู้ผลิตเช่นกัน มีทั้งสิ้น 5 ข้อมูล คือ ข้อมูลเวอร์ชันของ SCR (บิต 63:60 รวม 4 บิต),ข้อมูลเวอร์ชันของคุณสมบัติทางกายภาพของ SD การ์ด (บิต 59:56 รวม 4 บิต ใช้จริงบิตเดียว), ข้อมูลสถานะของข้อมูลหลังจากการลบ (1 บิตคือ บิต 55), ข้อมูลกำหนดระดับการป้องกัน(บิต 54 : 52 รวม 3 บิต),ข้อมูลแจ้งการรับรองขนาดของข้อมูลที่ทำการถ่ายทอดได้ของ SD การ์ด(บิต 47:32) และสำรองสำหรับใช้เฉพาะผู้ผลิตอีก 32 บิต (บิต 31: 0)

➔รีจิสเตอร์ DSR (Drive stage register)

เป็นรีจิสเตอร์ขนาด 16 บิต สำหรับเก็บค่าคุณสมบัติของไดร์เวอร์ทางเอาต์พุตของ SDIO การ์ดจะมีความแตกต่างกันไปในอุปกรณ์เอาต์พุตแต่ละตัว

ดังนั้น รีจิสเตอร์หลักๆที่ใช้จะมี 3 ถึง 4ตัวคือ OCR, CID, CSD และ RCA สำหรับการทดสอบเบื้องต้นจะใช้เพียง 2 ตัวคือ CID และ CSD

รีจิสเตอร์แสดงสถานะการทำงานของ SD การ์ดมี 2 ตัว คือ Card status และ SD_Status โดย Card status มีขนาด 32 บิตใช้แสดงสถานะการทำงานปกติ มีการทำงานเหมือนกับของ MMC การ์ด SD_Status มีขนาด 512 บิต สามารถแสดงสถานะการทำงานพิเศษที่เพิ่มเติมไปจาก Card status โดยข้อมูลสถานะจะถูกส่งส่งลงไปบนสายนำสัญญาณ DAT พร้อมกับรหัสตรวจสอบ CRC 16 บิต

รีจิสเตอร์ทั้งสองตัวนี้มีการจำแนกชนิดของสถานะการทำงานออกเป็น 4 แบบ และสามารถเคลียร์บิตได้ด้วยเงื่อนไขที่แตกต่างกันอีก 3 เงื่อนไข สามารถสรุปได้ดังนี้

• ชนิดของของสถานะการทำงาน

E - บิตแจ้งความผิดพลาด

S - บิตแจ้งสถานะ

R - บิตแจ้งการตรวจจับและเซตเมื่อได้รับการตอบสนองคำสั่ง

X - บิตแจ้งการตรวจจับและเซตในขณะที่กำลังกระทำคำสั่ง หากต้องการอ่านบิตนี้ ซีพียูจะต้องส่งคำสั่งอ่านสถานะมายัง SD การ์ดก่อน

• เงื่อนไขในการเคลียร์บิตแจ้งสถานะ

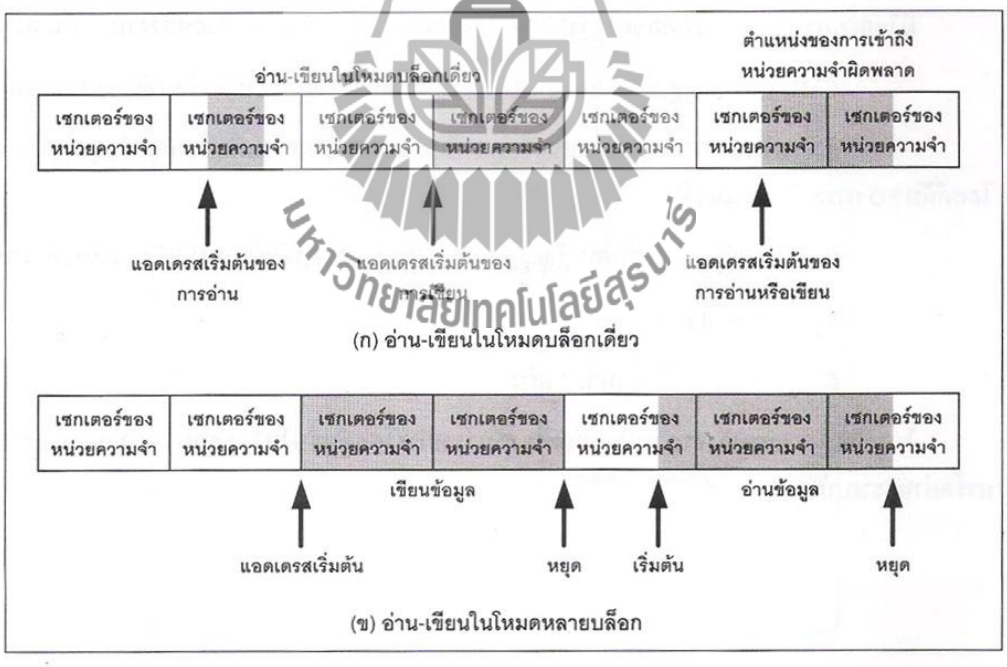
A - เคลียร์ด้วยกระบวนการทำงานตามปกติ

B - เคลียร์เนื่องจากผลของคำสั่งก่อนหน้า ดังนั้นบิตสถานะจะเคลียร์หลังจากทำงานผ่านไป 1 คำสั่ง หรือเป็นการสั่งเคลียร์บิตสถานะโดยตรง

C - เคลียร์ด้วยการอ่าน

2.6.6 กระบวนการอ่าน-เขียน SD การ์ด

SD การ์ดมีกระบวนการอ่าน-เขียนข้อมูล 2 โหมด ดังแสดงในรูปที่ 2. 25 โดยมีอัตราการถ่ายทอดข้อมูล 25 เมกะบิตต่อวินาทีในกรณีใช้สายเดียว (ติดต่อแบบบัส SPI) และ 100 เมกะบิตต่อวินาทีในกรณีใช้สายข้อมูล 4 เส้น (ติดต่อแบบบัส SD)



รูปที่ 2.25 กระบวนการอ่าน-เขียนข้อมูลของ SD การ์ด

2.6.7 การติดต่อกับ SD การ์ด

โฮสต์หรือคอมพิวเตอร์หรือไมโครคอนโทรลเลอร์สามารถติดต่อกับ SD การ์ดได้ 2 วิธี คือ ผ่านบัส SD และบัส SPI โดยใช้สายสัญญาณที่แตกต่างกันดังแสดงรายละเอียดในตารางที่ 2.6

ตารางที่ 2.6 แสดงสายสัญญาณของการติดต่อกับSD การ์ดทั้งแบบผ่านบัส SD และ SPI

ขา	การติดแบบบัส SD			การติดแบบบัส SPI		
	ชื่อขา	**ชนิดวงจร**	**คำอธิบาย**	**ชื่อขา**	**ชนิดวงจร**	**คำอธิบาย**
1	CD/DAT3	I/O และ พุชพูล	ตรวจสอบการมีอยู่ของการ์ด/สายข้อมูล DAT3	CS	อินพุต	สัญญาณเลือกกราวด์ (Chip select)
2	CMD	พุชพูล	สัญญาณคำสั่ง สัญญาณตอบสนอง	DI	อินพุต	สายสัญญาณข้อมูลเข้า
3	VSS1	อินพุต กราวด์	กราวด์	VSS	อินพุต กราวด์	กราวด์
4	VDD	อินพุต ไฟเลี้ยง	ไฟเลี้ยง	VDD	อินพุต ไฟเลี้ยง	ไฟเลี้ยง
5	CLK	อินพุต	สัญญาณนาฬิกา	SCK	อินพุต	สัญญาณนาฬิกา
6	VSS2	อินพุต กราวด์	กราวด์	VSS2	อินพุต กราวด์	กราวด์
7	DAT0	I/O และ พุชพูล	สายข้อมูล DAT0	D0	อินพุต พุชพูล	สายสัญญาณข้อมูลออก
8	DAT1	I/O และ พุชพูล	สายข้อมูล DAT1	RESERVE		สำรองไว้
9	DAT2	I/O และ พุชพูล	สายข้อมูล DAT2	RESERVE		สำรองไว้

2.6.7 .1 การติดต่อ SD การ์ดผ่านบัส SD

มีไดอะแกรมแสดงการติดต่อตามรูปที่ 2.26 ใช้สายสัญญาณ 6 เส้น และสายพลังงาน 3 เส้น

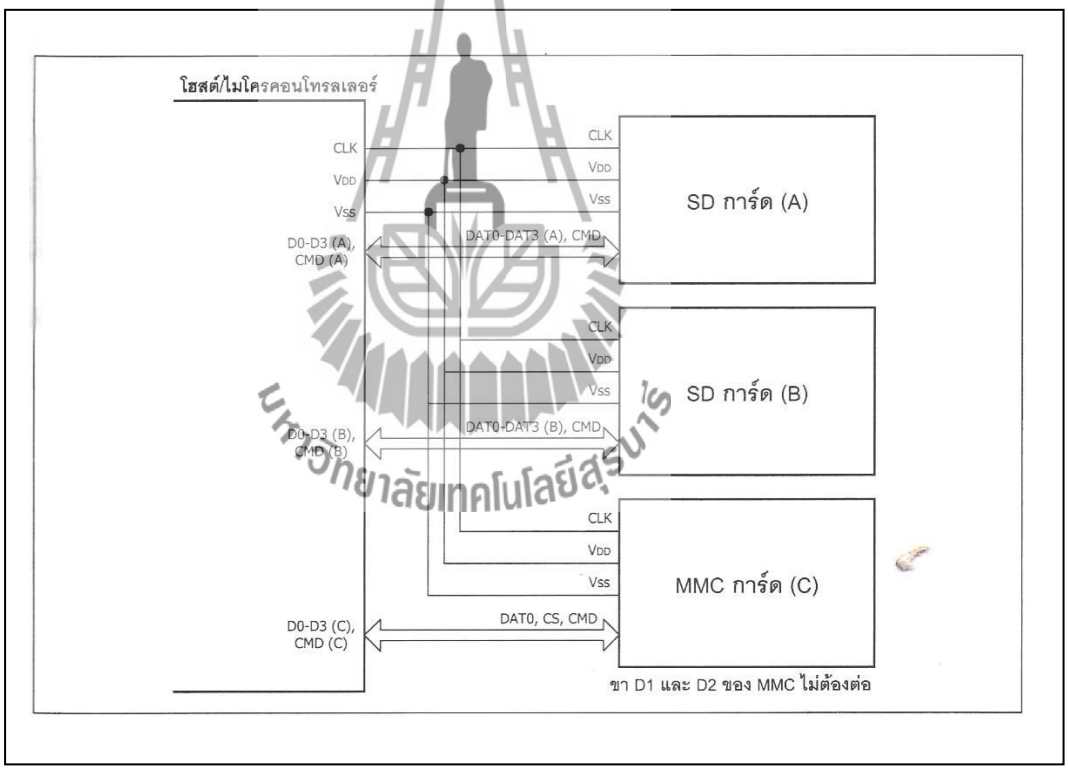
•CMD : สายสัญญาณคำสั่ง เป็นสัญญาณ 2 ทิศทางติดต่อระหว่างโฮสต์กับ SD การ์ด

•DAT0 ถึง DAT3 : สายสัญญาณข้อมูลเป็นสัญญาณ 2 ทิศทางเพื่อถ่ายทอดข้อมูลระหว่างโฮสต์กับ SD การ์ดมีทั้งสิ้น 4 เส้น

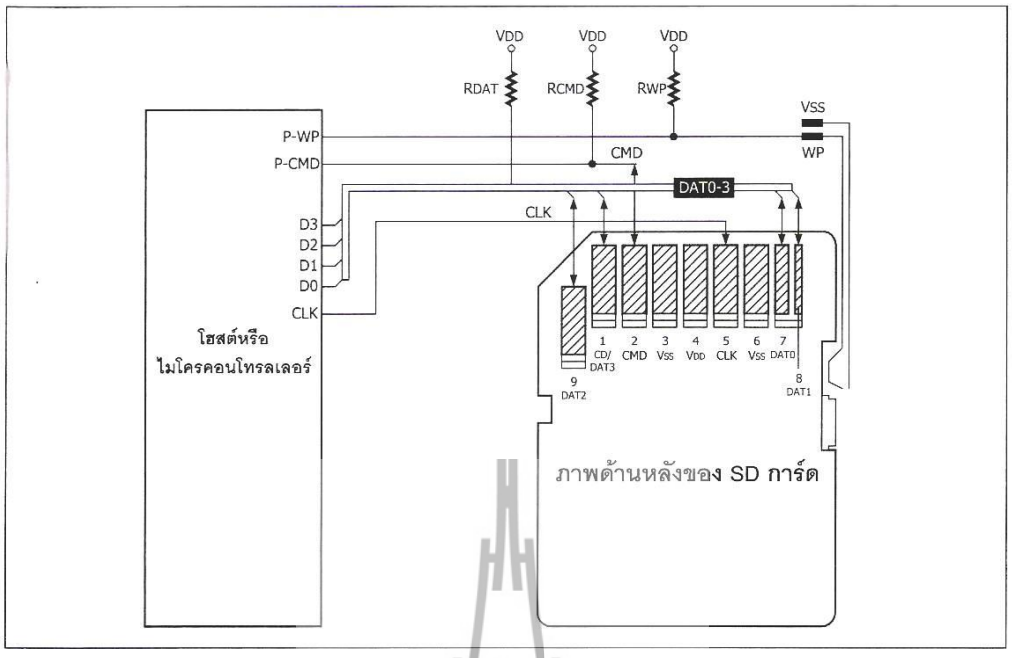
•CLK : สายสัญญาณนาฬิกา สัญญาณนี้จะส่งออกจากโฮสต์เพื่อกำหนดจังหวะการทำงาน

•VDD : สายไฟเลี้ยง

•GND : สายกราวด์ (ปกติมี 2 เส้น)



รูปที่ 2.26 ไดอะแกรมการติดต่อกับ SD การ์ดผ่านบัส SD



รูปที่ 2.27 วงจรการเชื่อมต่อเบื้องต้นระหว่างโฮสต์หรือไมโครคอนโทรลเลอร์กับ SD การ์ดผ่านระบบบัส SD

2.6.7 .2 การติดต่อกับ SD การ์ดผ่านบัส SPI

กลุ่มข้อมูลที่ใช้ในการติดต่อบนระบบบัส SPI หรือเรียกว่า SPI message ประกอบด้วย คำสั่ง (command), การตอบสนอง (response) และบล็อกข้อมูล (data-block) การสื่อสารระหว่างโฮสต์หรือ ไมโครคอนโทรลเลอร์กับ SD การ์ดจะได้รับการกำหนดจังหวะจากโฮสต์ โดยโฮสต์จะเริ่มต้นการติดต่อด้วยการทำให้สายสัญญาณ CS เป็นลอจิก “0”

การตอบสนองของ SD การ์ดในการติดต่อผ่านบัส SPI มีหลักเกณฑ์ดังนี้

(1) SD การ์ดที่ถูกเลือกให้ติดต่อจะต้องตอบสนองต่อทุกคำสั่งเสมอ

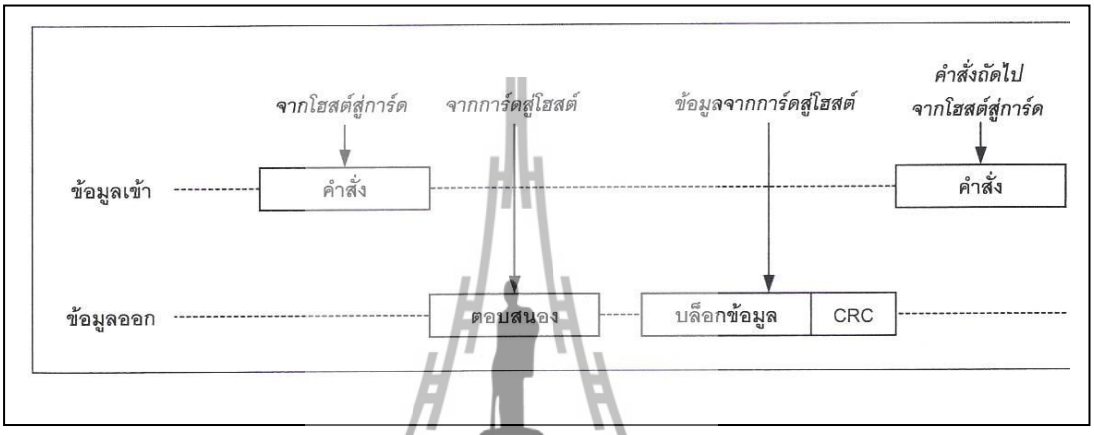
(2) ข้อมูลตอบสนองจะใช้ขนาด 8 หรือ 16 บิต

(3) เมื่อ SD การ์ดประสบปัญหาในการกู้คืนข้อมูล SD การ์ดจะแจ้งกลับด้วยข้อมูลตอบสนองผิดพลาด (error response) แทนที่จะเป็นบล็อกข้อมูล โดยมีค่าเวลาไทม์เอาต์ที่มากกว่าการติดต่อผ่านบัส SD

มีการตอบสนองคำสั่งเมื่อทุกๆบล็อกข้อมูลถูกส่งไปยัง SD การ์ดในระหว่างการเขียน จะมีการตอบสนองด้วยสัญลักษณ์พิเศษ (special data response token) บล็อกของข้อมูลอาจมีขนาดใหญ่เท่ากับ 1 บล็อกข้อมูลปกติ หรือเล็กเพียง 1 ไบต์ได้

2.6.7.3 การอ่านข้อมูลในโหมด SPI

การอ่านข้อมูลในโหมด SD การ์ดในโหมดการติดต่อแบบ SPI นี้ สามารถอ่านได้ทั้งแบบบล็อกเดี่ยวและหลายบล็อก คำสั่งที่ใช้คือ CMD17 สำหรับบล็อกเดี่ยว และ CMD18 สำหรับหลายบล็อก เมื่อ SD การ์ดได้รับคำสั่งร้องขอเพื่ออ่านข้อมูลแล้ว มันจะส่งรหัสตอบสนองต่อด้วยบล็อกข้อมูลที่มีความยาวตามที่กำหนดจากคำสั่ง CMD16 (SET_BLOCK_LENGTH) ปิดท้ายด้วยรหัส CRC ดังแสดงในรูปที่ 2.28



รูปที่ 2.28 กระบวนการอ่านข้อมูลแบบบล็อกเดี่ยวจาก SD การ์ด

สำหรับรหัส CRC 16 บิตนั้นจะถูกกำหนดด้วยสมการโพลิโนมียลตามมาตรฐาน CCITT ดังนี้

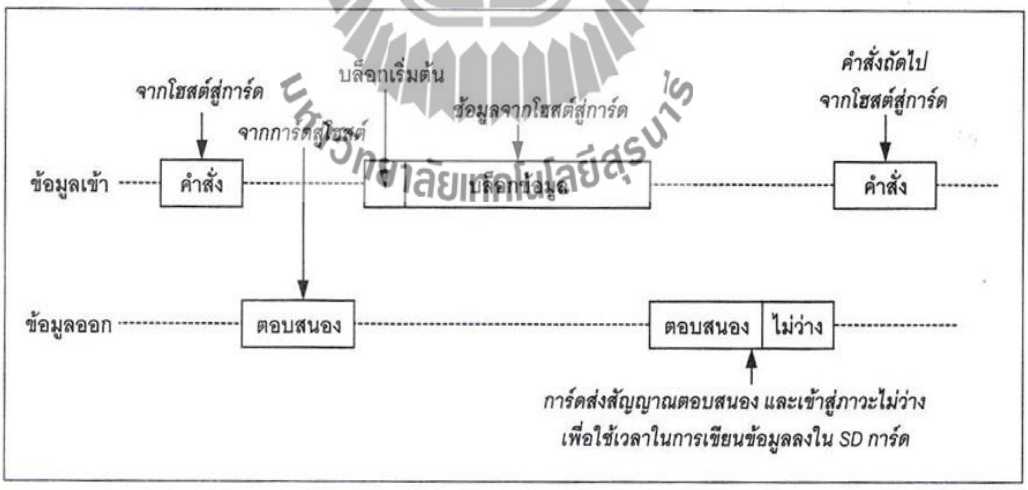
$$X^{16}+X^{12}+X^{5}+1$$

ความยาวของบล็อกข้อมูลสูงสุดคือ 512 ไบต์ กำหนดโดย REAL_BL_LEN หนึ่งพารามิเตอร์ของรีจิสเตอร์ CSD มีขอบเขตในการกำหนดค่าได้ตั้งแต่ 1 จนถึงค่าของ REAL_BL_LEN แต่โดยส่วนใหญ่แล้ว มักเลือกที่จะอ่านบล็อกข้อมูลความยาว 512 ไบต์ เพื่อความเร็วและต่อเนื่องในการทำงาน แอดเดรสเริ่มต้นของการอ่านสามารถกำหนดที่แอดเดรสใดๆ ก็ได้ภายในขอบเขตของ SD การ์ดใบนั้นๆที่ทำการอ่าน ในกรณีที่เกิดความผิดพลาดในกระบวนการอ่านข้อมูลขึ้น SD การ์ดจะไม่ส่งข้อมูลใดๆออกมาแต่จะส่งรหัสแจ้งความผิดพลาดกลับมายังโฮสต์แทน และยกเลิกกระบวนการติดต่อเพื่ออ่านข้อมูลในทันที ในกรณีที่มีการอ่านข้อมูล ในแต่ละบล็อกจะมีรหัส CRC 16 บิตปิดท้ายเสมอ เพื่อช่วยแยกข้อมูลให้ชัดเจน รวมทั้งช่วยในการตรวจสอบว่าข้อมูลที่ถูกอ่านออกไปถูกต้องสมบูรณ์ดังแสดงในรูปที่ **2.**18 และเมื่อต้องการหยุดอ่านข้อมูลต้องมีการส่งรหัสคำสั่งแจ้งแก่ SD การ์ดด้วย นั่นคือรหัสคำสั่ง CMD12 (Stop transmission command) หรือคำสั่งหยุดการส่งข้อมูลของ SD การ์ด

2.6.7.4 การเขียนข้อมูลในโหมด SPI

การเขียนข้อมูลไปยัง SD การ์ดในโหมดการติดต่อแบบ SPI นี้ สามารถเขียนได้ทั้งแบบบล็อกเดี่ยวและหลายบล็อก คำสั่งที่ใช้คือ CMD24 สำหรับบล็อกเดี่ยว และ CMD25 สำหรับบล็อก เมื่อ SD การ์ด ได้รับคำสั่งร้องขอเพื่อเขียนข้อมูลแล้ว มันจะส่งรหัสตอบสนอง จากนั้นจะรอบบล็อกข้อมูลจากโฮสต์ ความยาวของบล็อกข้อมูลในกรณีเขียนนี้ต้องใช้ 512 ไบต์ เพื่อช่วยลดความผิดพลาดในการเขียนข้อมูลในครั้งถัดไป ในรูปที่ 2. 29 แสดงจังหวะการเขียนข้อมูลลงใน SD การ์ดแบบบล็อกเดี่ยว

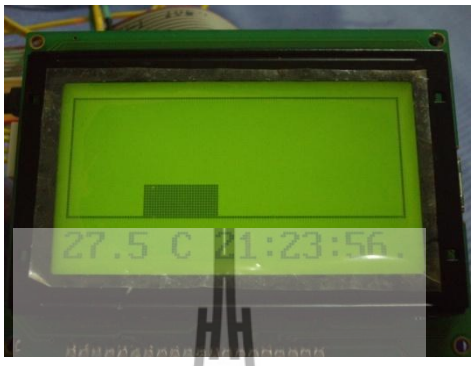
ในทุกๆบล็อกข้อมูลที่จะนำมาเขียนลงใน SD การ์ดต้องเริ่มต้นด้วยบล็อกเริ่มต้น (start block) มีความยาว 1 ไบต์เมื่อข้อมูลถูกส่งออกมายัง SD การ์ดจะส่งสัญญาณตอบสนองตามด้วยสถานะไม่ว่าง (busy) เพื่อใช้เวลาไปตรวจสอบว่า ในขณะนั้นการ์ดยังมีพื้นที่เหลือพอสำหรับเขียนข้อมูลใหม่ลงไปหรือไม่ ถ้ามีพอก็จะเขียนข้อมูล และปรับปรุงความจุของการ์ดที่เหลือหลังจากเขียนข้อมูลใหม่แล้ว ในจังหวะที่เกิดสถานะไม่ว่างนั้น ที่สายสัญญาณข้อมูลออก (หรือข้อมูลเอาต์พุต) จะได้รับการกำหนดให้เป็นลอจิกต่ำจนกว่าจะเสร็จสิ้นกระบวนการสถานะของสายสัญญาณจะกลับมาเป็นลอจิกสูง



รูปที่ 2.29 จังหวะการเขียนข้อมูลลงใน SD การ์ดแบบบล็อกเดี่ยว

หลังจากที่การเขียนข้อมูลเสร็จสิ้นลง โฮสต์ควรตรวจสอบผลการทำงานด้วย โดยส่งรหัสคำสั่ง CMD13 (SEND_STATUS) ไปยัง SD การ์ดโดยจะเน้นการตรวจสอบรหัส CRC และบิตแจ้งเตือนความผิดพลาดจากการเขียนข้อมูลลงไปในหน่วยความจำ

2.7 จอแสดงผล

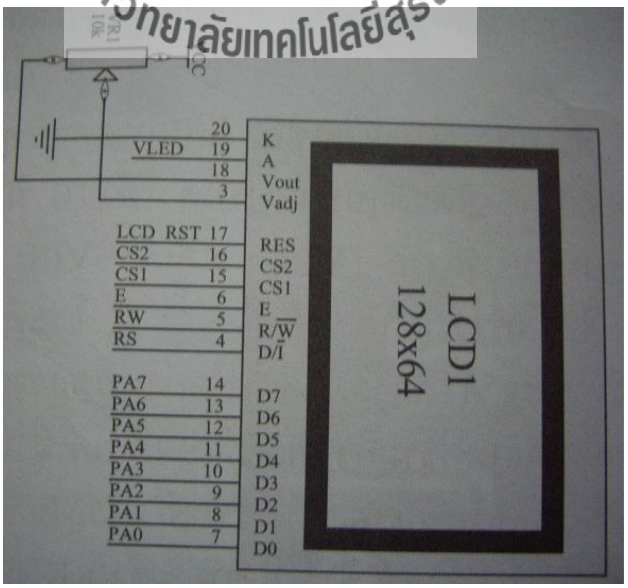


รูปที่ 2.30 จอแสดงผล

การติดต่อกับ LCD Module
มีอยู่ด้วยกัน 2 แบบ

1. แบบ 8 บิต
2. แบบ 4 บิต

ซึ่งในโครงงานนี้ซึ่งจะใช้การต่อ LCD Module แบบ 8 บิต ซึ่งจะแสดงการต่อดังในรูปที่ 2.31



รูปที่ 2.31 แสดงการต่อ LCD Module แบบ 8 บิต

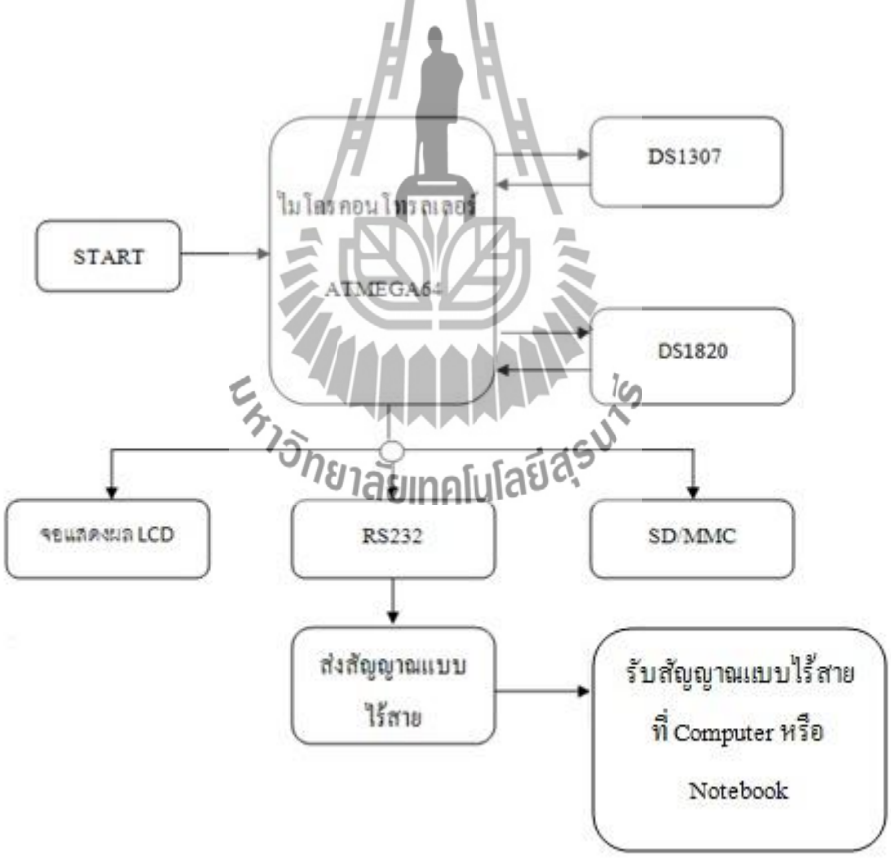
บทที่ 3
หลักการทำงานและการออกแบบ

3.1 บทนำ

ในบทนี้จะกล่าวถึงการทำงาน การใช้โปรแกรม Winavr การใช้คำสั่งในการติดต่อบอร์ดเบื้องต้น การแสดงผลผ่าน LCD (Liquid crystal display) การสื่อสารแบบอนุกรมผ่านวงจร UART (Universal asynchronous receiver transmitter) การสื่อสารแบบไร้สาย และเวลาการชาร์ตแบตเตอรี่

3.2 หลักการทำงานของเครื่องวัดอุณหภูมิอเนกประสงค์

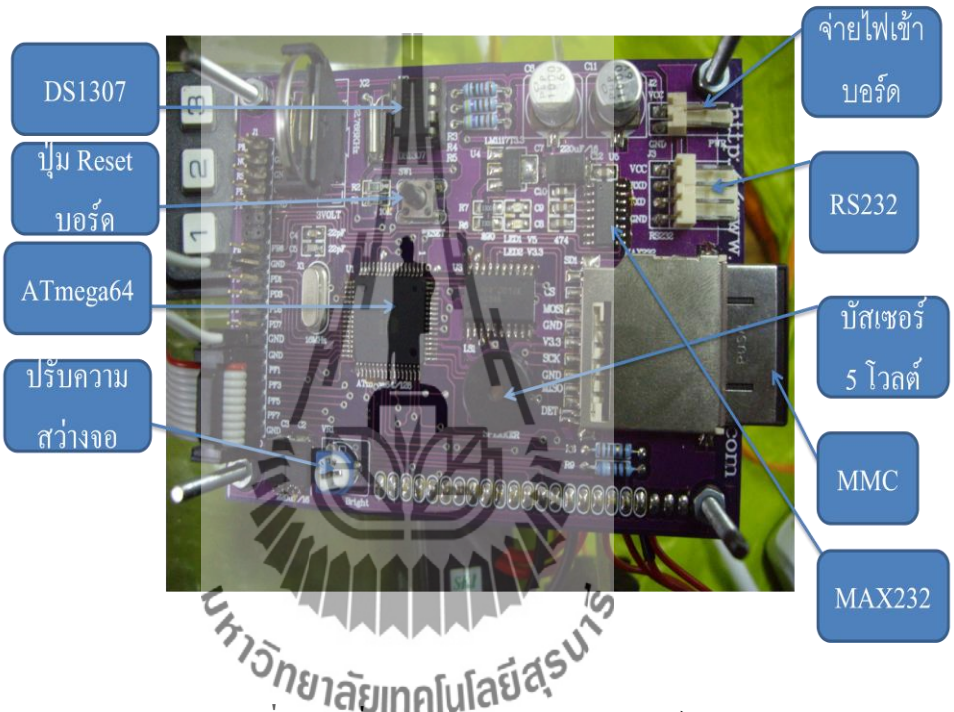
3.2.1 ระบบของเครื่องวัดอุณหภูมิเอนกประสงค์



รูปที่ 3.1 แผนภาพเครื่องวัดอุณหภูมิเอนกประสงค์

ระบบของเครื่องวัดอุณหภูมิเอนกประสงค์ แยกเป็นส่วนหลักๆ ดังนี้คือ ส่วนของไมโครคอนโทรลเลอร์เบอร์ ATMEGA64 ส่วนของไอซีเบอร์ DS1820 และ DS1307 ส่วนของการแสดงผล ที่แสดงที่จอแสดงผล LCD แสดงผลที่คอมพิวเตอร์โดยผ่านการสื่อสารแบบไร้สาย และส่วนของการบันทึกข้อมูลลง SD/MMC ดังได้แสดงของระบบในรูปที่ 3.1

3.2.2 การทำงานของวงจร

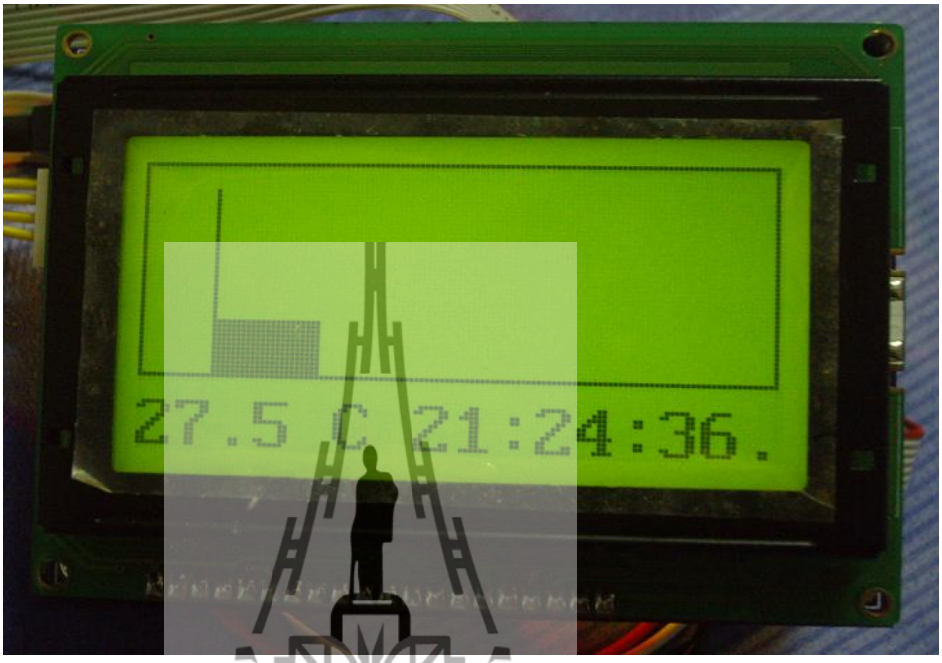


รูปที่ 3.2 เครื่องวัดอุณหภูมิเอนกประสงค์

เริ่มต้นจากการป้อนไฟกระแสตรงขนาด 5 โวลต์ผ่าน CON2 จากนั้นจะมีตัวเก็บประจุ C2, C3, C6 และ C11 ทำหน้าที่กรองแรงดันให้เรียบมากขึ้น โดยสถานะของแรงดันจะถูกแสดงด้วย LED1

หัวใจหลักของโครงงานนี้เป็นไมโครคอนโทรลเลอร์ตระกูล AVR สามารถเลือกใช้ได้ 2 เบอร์ คือ ATMEGA64 และ ATMEGA128 เนื่องจากโครงสร้างภายนอกของทั้งสองเบอร์เหมือนกัน ซึ่งจะต่างกันก็เพียงคุณสมบัติของหน่วยความจำ Flash และหน่วยความจำ EEprom ภายในเท่านั้น กล่าวคือ ATMEGA64 มีหน่วยความจำ Flash ขนาด 64 กิโลไบต์และหน่วยความจำ EEprom ขนาด 2 กิโลไบต์ ในส่วนเบอร์ ATMEGA128 มีหน่วยความจำ Flash ขนาด 128 กิโลไบต์และหน่วยความจำ EEprom ขนาด 4 กิโลไบต์แต่หน่วยความจำ SRAM ทั้งสองเบอร์มีขนาดเท่ากันคือ 4

กิโลไบต์ ซึ่งหน่วยความจำดังกล่าวจะใช้สำหรับการประมวลผลและควบคุมการทำงานในส่วนต่างๆ อาทิเช่น การแสดงผลบนหน้าจอแอลซีดีและนำข้อมูลไปเก็บไว้ที่หน่วยความจำ SD/MMC เป็นต้น



รูปที่ 3.3 จอกราฟิกแอลซีดีขนาด 128x64 พิกเซล

การแสดงผลในโครงงานนี้จะใช้จอกราฟิกแอลซีดีขนาด 128x64 พิกเซล (แนวแกนนอน 128 พิกเซลและแนวแกนตั้ง 64 พิกเซล) โดยใช้พอร์ต A เชื่อมต่อกับ D0-D7 เพื่อส่งข้อมูลการแสดงผลและใช้พอร์ต C ในการควบคุมการทำงานของแอลซีดี และในส่วนควบคุมการเปิดปิดไฟพื้นหลัง (Black Light) จะใช้ขา PC1 ซึ่งจะเชื่อมต่อผ่าน Q2 อีกทีหนึ่ง

PC5 และ PC6 เชื่อมต่อกับขา Chip Select ใช้สำหรับเลือกคอนโทรลเลอร์ภายในของแอลซีดีเพื่อควบคุมการทำงานของแอลซีดี เนื่องจากภายในแอลซีดีชนิด 128x64 ที่ใช้ในโครงงานนี้จะมีคอนโทรลเลอร์เพื่อควบคุมการแสดงผลแนวแกนนอนอยู่ 2 ตัว โดยแต่ละตัวจะถูกแบ่งให้มีการแสดงผล 64 พิกเซล ดังนั้นเมื่อรวมทั้งสองตัวเข้าด้วยกันแกนนอนจะสามารถแสดงผลได้ 128 พิกเซล และในส่วนของการแสดงผลแนวแกนตั้งไม่จำเป็นต้องเพิ่มคอนโทรลเลอร์เนื่องจากมีขนาด 64 พิกเซลอยู่แล้ว สำหรับการรีเซ็ตการทำงานของแอลซีดีจะใช้ขา PC7

สุดท้ายในส่วนของแอลซีดี คือ การควบคุมความสว่างของพิกเซล จะใช้ตัวต้านทานปรับค่าได้ 10 กิโลโอห์ม (VR1) ต่อเข้ากับขาไฟบวก (VCC) และขา VEE (ขา 18 ของแอลซีดี) ที่มี

สภาวะเป็นแรงดันไฟลบ 9 โวลต์ ส่วนขากลางจะต่อเข้ากับขา 3 ของแอลซีดีเพื่อกำหนดแรงดันอ้างอิงและจะทำหน้าที่ควบคุมความสว่างของพิกเซลบนหน้าจอ

การเชื่อมต่อกับหน่วยความจำ SD/MMC ในส่วนนี้จะใช้แรงดันขนาด 3.3 โวลต์ ซึ่งจะแตกต่างจาก IC1 ที่มีระดับแรงดันอยู่ที่ 5 โวลต์ ด้วยเหตุนี้การเชื่อมต่อระหว่าง IC1 กับการ์ด SD/MMC จึงไม่สามารถทำได้โดยตรงเพราะจะทำให้เกิดความเสียหายกับ SD/MMC ได้ การแก้ไขจุดนี้สามารถทำได้โดยใช้ IC3 (74LVC245) มาทำหน้าที่เป็นบัฟเฟอร์เพื่อลดทอนแรงดันจาก 5 โวลต์ให้เหลือ 3.3 โวลต์เสียก่อน ในส่วนของภาคจ่ายไฟ 3.3 โวลต์จะใช้ IC4 เบอร์ LM1117T3.3 เพื่อทำหน้าที่ลดทอนแรงดันจาก 5 โวลต์ให้เหลือขนาด 3.3 โวลต์และจะมี LED2 แสดงสถานะของแรงดัน

ขาที่จำเป็นสำหรับการเชื่อมต่อระหว่าง IC1 กับการ์ด SD/MMC มีดังนี้คือ

- ขา PB0 (MMC_CS) เป็นขา Chip Select ใช้เมื่อต้องการติดต่อกับ SD/MMC
- ขา PB1 (MMC_SCK) เป็นขาสัญญาณนาฬิกาในโหมด SP1 สำหรับ SD/MMC
- ขา PB2 (MMC_MOSI) เป็นขาเพื่อส่งข้อมูลไปยัง SD/MMC
- ขา PB3 (MISO) เป็นขาเพื่อส่งข้อมูลไปยัง SD/MMC
- ขา PB4 (MMC_DETECT) เป็นขาเพื่อคอยตรวจสอบสถานะของสวิตช์ขนาดเล็กที่อยู่ภายในซ็อกเก็ต SD/MMC

DS1307 เป็นไอซีสร้างฐานเวลาจริง (Real Time Clock) มีมาตรฐานการเชื่อมต่อแบบ I2Cในส่วนของ IC1 จะใช้ขา PD0 ที่ภายในมีคุณสมบัติเป็น SCL ใช้สำหรับเป็นสัญญาณนาฬิกา และขา PD1 ที่มีคุณสมบัติ SDA เป็นขาข้อมูล สืบเนื่องจาก DS1307 สามารถกำหนดให้สร้างสัญญาณความถี่ 1 เฮิร์ตซ์ (Hz) ออกมาจากขาที่ 7 (SQ) ได้ ดังนั้นผู้เขียนจึงนำคุณสมบัตินี้มาใช้ประโยชน์เพื่อเป็นสัญญาณการอินเตอร์รัปต์โดยต่อเข้ากับขา PE4 (INT4) ซึ่งเมื่อเกิดการอินเตอร์รัปต์ขึ้น IC1 จะทำการอ่านข้อมูลจาก DS1307 ไปแสดงผลบนหน้าจอแอลซีดี

การเชื่อมต่อกับระหว่างบอร์ดกับคอมพิวเตอร์จะใช้ IC5 (MAX232) ทำหน้าที่เปลี่ยนแรงดันแบบ TTL เป็น RS-232 เพื่อป้อนให้กับพอร์ตสื่อสารของเครื่องคอมพิวเตอร์ สำหรับคอนเน็กเตอร์สำหรับการเชื่อมต่อเข้ากับคอมพิวเตอร์จะมีจำนวน 4 ขา (VCC, RxD, TxD, และ GND) ซึ่งการเชื่อมต่อผู้ใช้จะต้องเรียกตำแหน่งขาให้ถูกต้องการเชื่อมต่อจึงจะสมบูรณ์ หรือจะให้ง่ายกว่านั้นผู้ใช้สามารถนำสายที่เป็นมาตราฐานของบริษัท ETT มาใช้ได้เลย

คอนเน็กเตอร์ CON1 ขนาด 40 ขาเชื่อมต่อกับขาพอร์ตของ IC1 ซึ่งขา 1-10 จะเตรียมไว้สำหรับการโปรกแรมแบบ ISP โดยขา 32 ถึง 40 จะเชื่อมต่อกับพอร์ต F ในการใช้งานจะถูกนำไปเชื่อมต่อเข้ากับคีย์บอร์ดขนาด 4x3 สำหรับการตังค่าการใช้งานต่าง นอกจากนั้นแล้วขา CON1 ที่

เหลือจะถูกเชื่อมต่อกับพอร์ต PB5, PB6, PB7 และพอร์ต D ซึ่งขาต่างๆดังกล่าวผู้ใช้สามารถนำไปเชื่อมต่อกับอุปกรณ์ภายนอกเพื่อสร้างแอพพลิเคชั่นอื่นๆได้ตามต้องการ

DS1820 เป็นไอซีตรวจสอบอุณหภูมิ โดยที่ขาสัญญาณ DQ จะเชื่อมต่อเข้ากับขา PB7 และขาสัญญาณจะต้องต่อตัวต้านทาน 4.7 กิโลโอห์มพลูอัพ (pull-up) ด้วย ข้อควรระวังในการใช้งาน DS1820 ไม่ควรนำไปวัดในสภาวะที่อุณหภูมิสูงถึง 100 องศาเซลเซียส ถึงแม้ว่าคุณสมบัติจะสามารถวัดอุณภูมิสูงถึง 125 องศาเซลเซียส เนื่องจาก DS1820 อาจจะเกิดความเสียหายได้จากการวัดอุณหภูมิที่มีความร้อนสูงแบบต่อเนื่อง

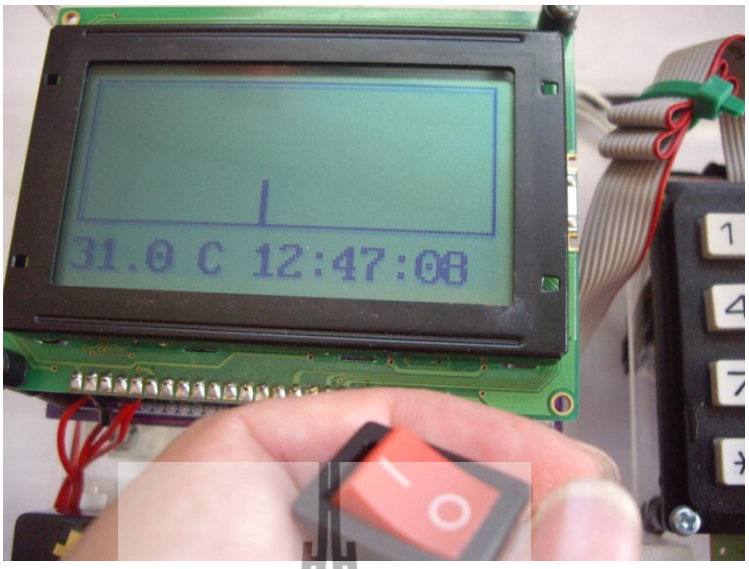
สุดท้ายเป็นวงจรควบคุมการทำงานของบัซเซอร์ เนื่องจากกระแสที่ขาเอาต์พุตของ IC1 ไม่เพียงพอต่อการขับบัซเซอร์เปล่งเสียงได้ ดังนั้นจะต้องทำการขยายกระแสโดยใช้ทรานซิสเตอร์ Q1 เบอร์ MMBT3904

รายละเอียดดูจากภาคผนวก รูปที่ 2.38

3.2.3 การใช้งานเครื่องวัดอุณหภูมิเอนกประสงค์

1. ทำการเปิดสวิตช์เพื่อทำการจ่ายไฟให้กับวงจร

รูปที่ 3.4 ก่อนเปิดเครื่องวัดอุณหภูมิเอนกประสงค์

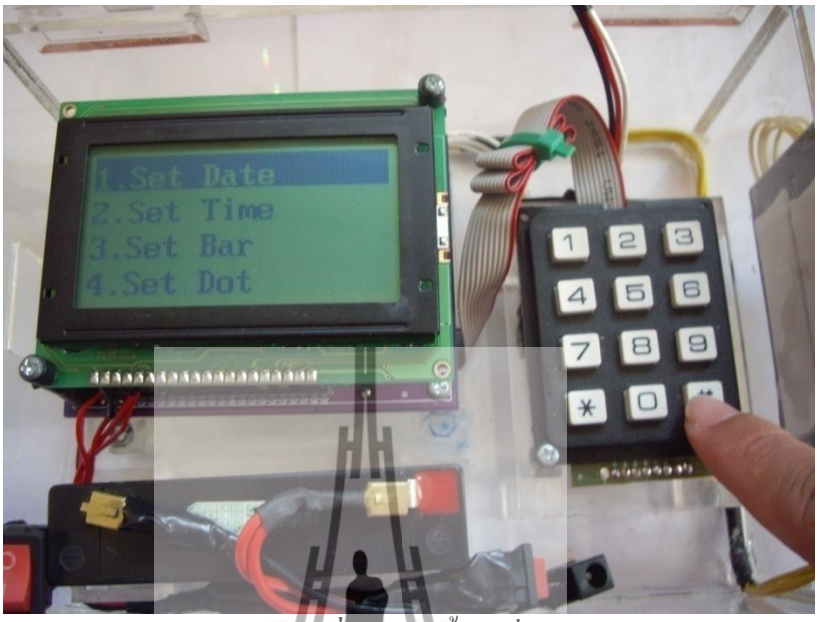


รูปที่ 3.5 เมื่อทำการกดสวิตช์เปิดเครื่องวัดอุณหภูมิเอนกประสงค์

2. เมื่อใช้ครั้งแรกให้ทำการตั้งวันที่ และเวลา เพื่อให้ตรงกับปัจจุบัน โดยทำการกดเลข 0 บนคีย์บอร์ด เครื่องจะถามรหัสผ่าน ให้ทำการกรอกรหัส 1449 แล้วกด # เพื่อตกลง

รูปที่ 3.6 การใส่รหัสผ่าน

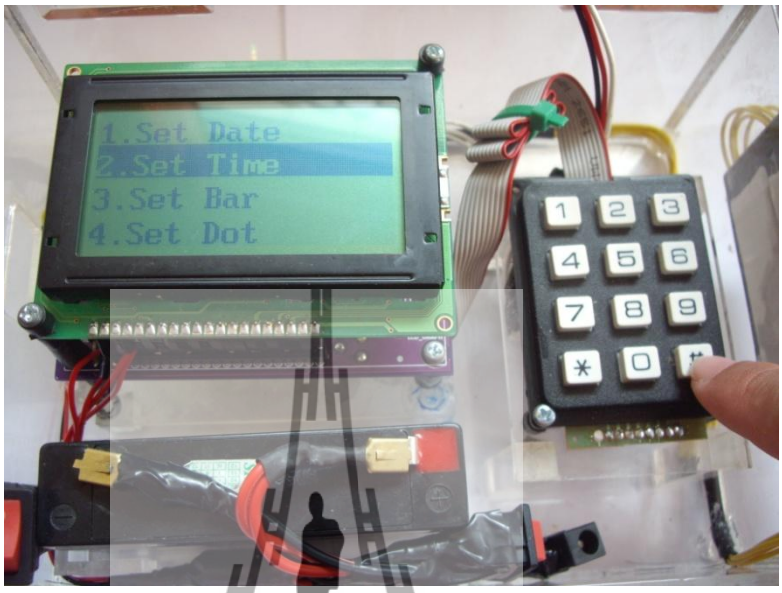
2.1 เลือกอันแลกเพื่อทำการตั้งค่าวันที่ เมื่อใส่วันที่แล้วให้ตกลงโดยกดปุ่ม #



รูปที่ 3.7 การตั้งวันที่

รูปที่ 3.8 การตั้งวันที่ (ต่อ)

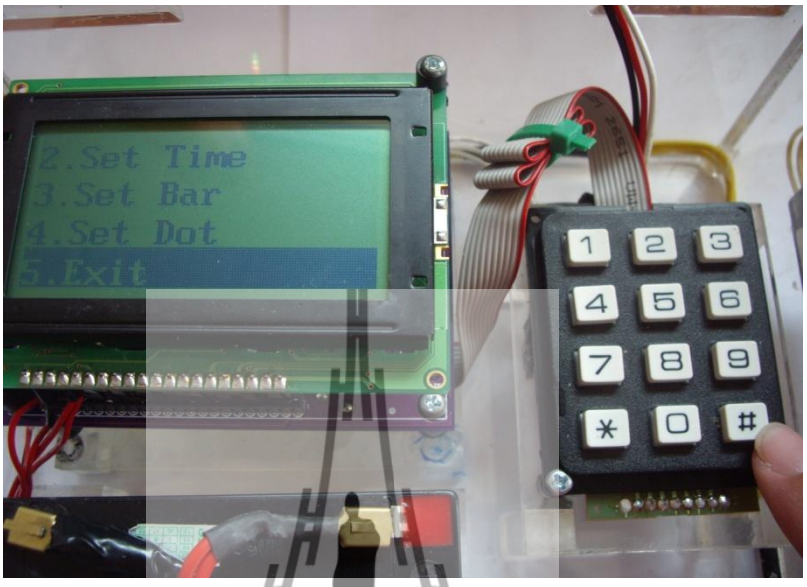
2.2 เลือกอันที่สองเพื่อทำการตั้งเวลา โดยเวลาแสดงเป็นแบบ 24 ชั่วโมง เมื่อใส่เวลาเสร็จแล้วให้ตกลงโดยกดปุ่ม #



รูปที่ 3.9 การตั้งเวลา

รูปที่ 3.10 การตั้งเวลา (ต่อ)

3. เมื่อทำการตั้งค่าเสร็จแล้วให้เลือก Exit โดยกดเลข 7 และ 4 เพื่อทำการขึ้นลง เมื่อเลือกแล้วทำการตกลง โดยกดปุ่ม #



รูปที่ 3.11 ออกจากการตั้งค่า

4. เมื่อไม่ต้องการที่จะดูหน้าจอให้ทำการปิดสวิตช์ เพื่อเป็นการประหยัดแบตตารี

รูปที่ 3.12 ปิดสวิตช์จอ LCD

3.2.4 การเชื่อมต่อการสื่อสารแบบไร้สาย

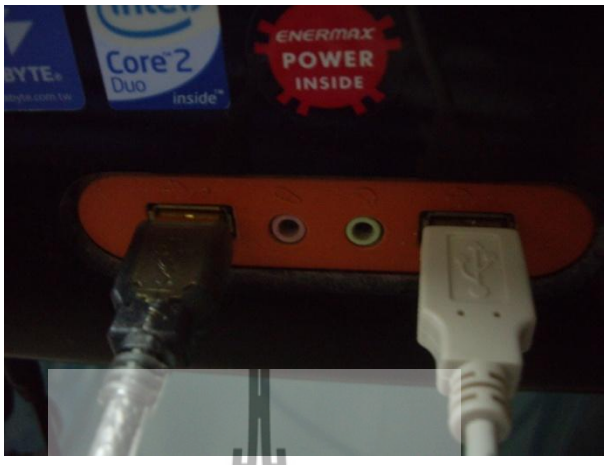
1. ทำการเสียบสาย RS232 และเสียบแหล่งจ่ายไฟโดยใช้ USB (Universal serial bus)

รูปที่ 3.13 การเสียบสาย RS232และแหล่งจ่ายไฟ

2. ทำการแปลงหัวเชื่อมต่อจาก RS232 มาเป็น USB

รูปที่ 3.14 การแปลงหัวเชื่อมต่อจาก RS232 มาเป็น USB

3. ทำการเสียบสาย USB ทั้ง 2 ที่ Computer หรือ Notebook เป็นการติดตั้งพร้อมใช้งาน



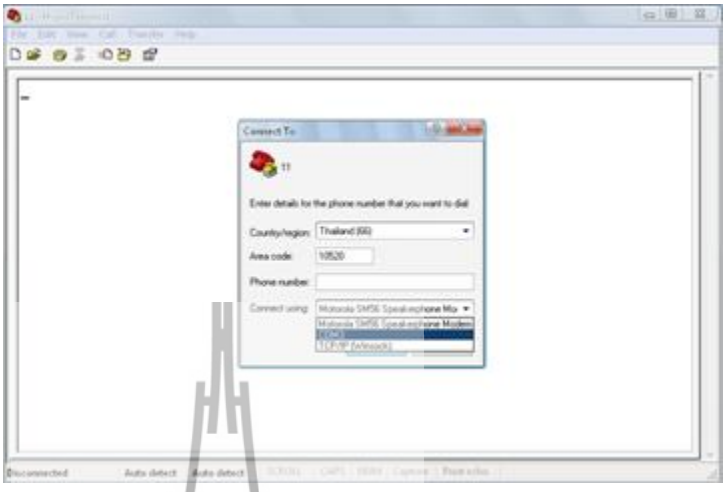
รูปที่ 3.15 การเสียบสาย USB ทั้ง 2 ที่ Computer หรือ Notebook

3.2.5 การ Capture Text

1. เปิดโปรแกรม Hyperterminal ขึ้นมาและตั้งชื่อไฟล์ในการเข้าใช้งาน

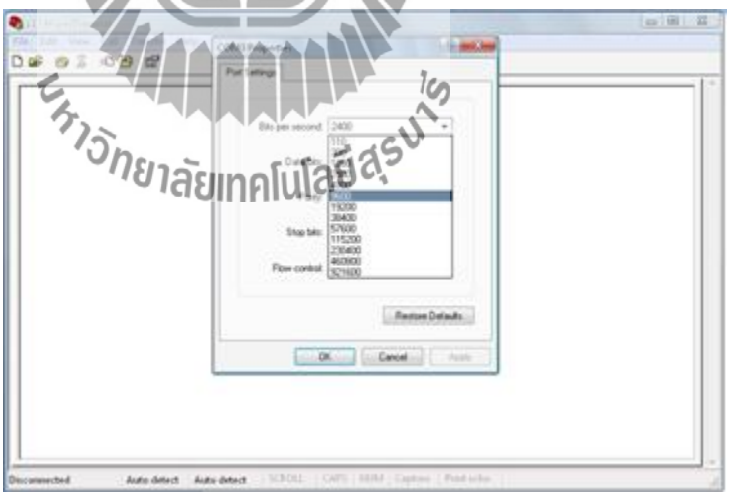
รูปที่ 3.16 การตั้งชื่อไฟล์ในการเข้าใช้งาน

2. ทำการเชื่อมต่อไปยัง Port ของโน๊ตบุ๊คที่ใช้ส่งข้อมูล



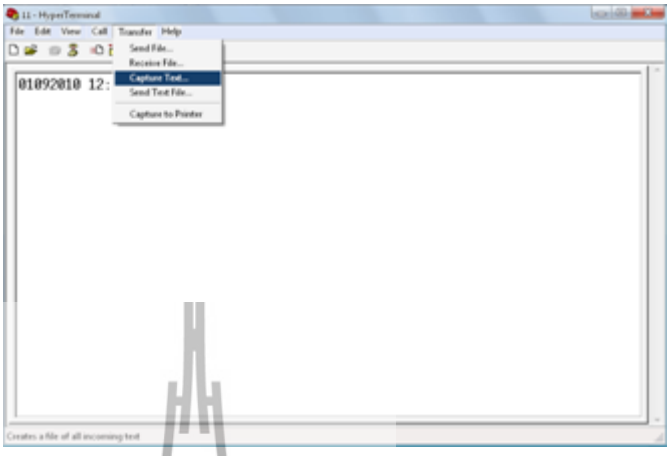
รูปที่ 3.17 การเชื่อมต่อ port ของโน๊ตบุ๊ค

3. กำหนดค่า Bit per second ที่ 9600



รูปที่ 3.18 การกำหนดค่า Bit per second

4. Hyperterminal จะแสดงค่าข้อมูล แล้วไปคลิกที่ Transfer >> Capture Text



รูปที่ 3.19 การ Capture Text

5. ตั้งชื่อไฟล์ที่จะใช้ในการบันทึก แล้วกด save

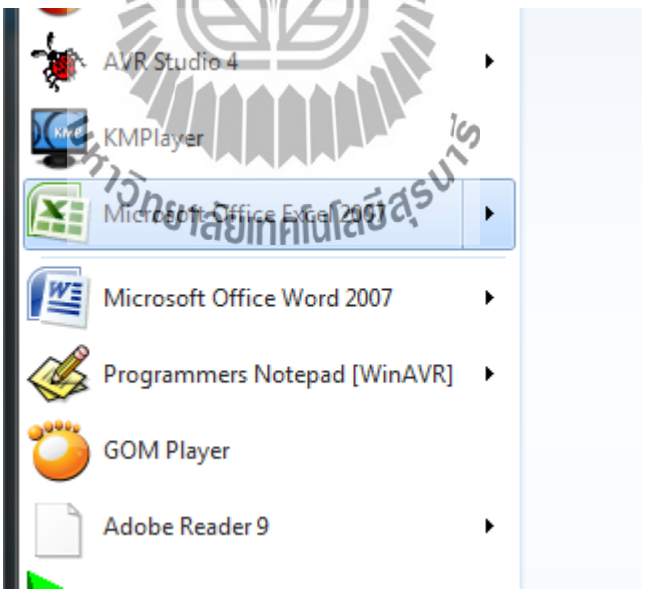
รูปที่ 3.20 การตั้งชื่อไฟล์ในการ Capture Text

6. กดปุ่ม start เพื่อเป็นการเริ่มเก็บข้อมูล

รูปที่ 3.21 การเริ่มเก็บข้อมูล

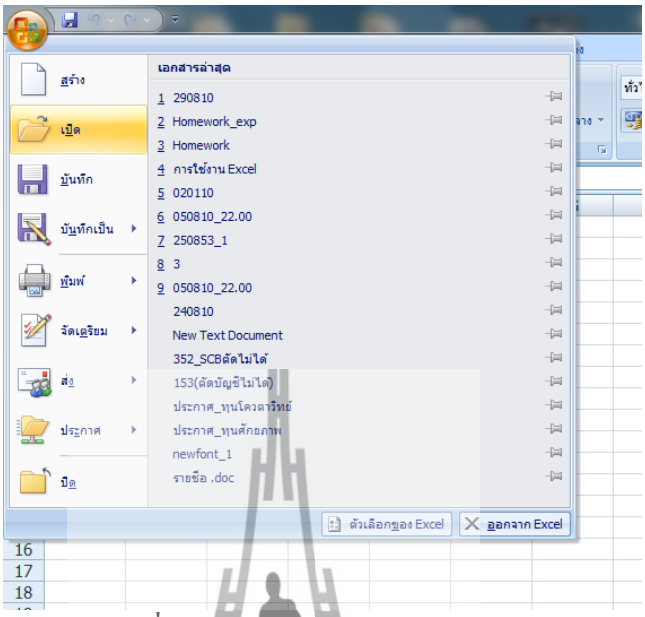
3.2.6 การนำเข้าไฟล์ .TXT โดยใช้โปรแกรม Microsoft office excel 2007

1. ทำการเปิดโปรแกรม Microsoft office excel 2007 (หรือใช้เวอร์ชั่นอื่น)



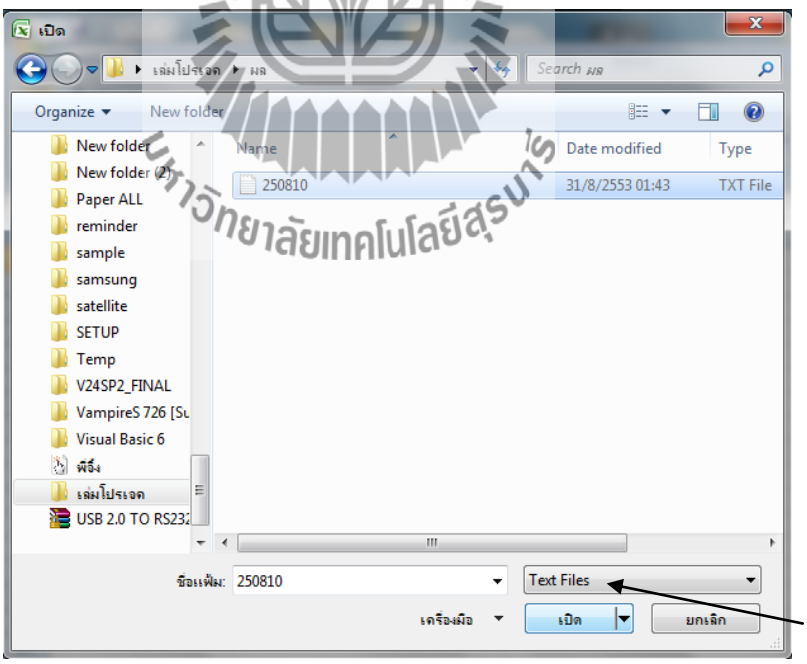
รูปที่ 3.22 เปิดโปรแกรม Microsoft office excel 2007

2. ทำการเปิดไฟล์โดยการเอาเมาส์ชี้ที่ปุ่ม Office แล้วเลือกเปิด



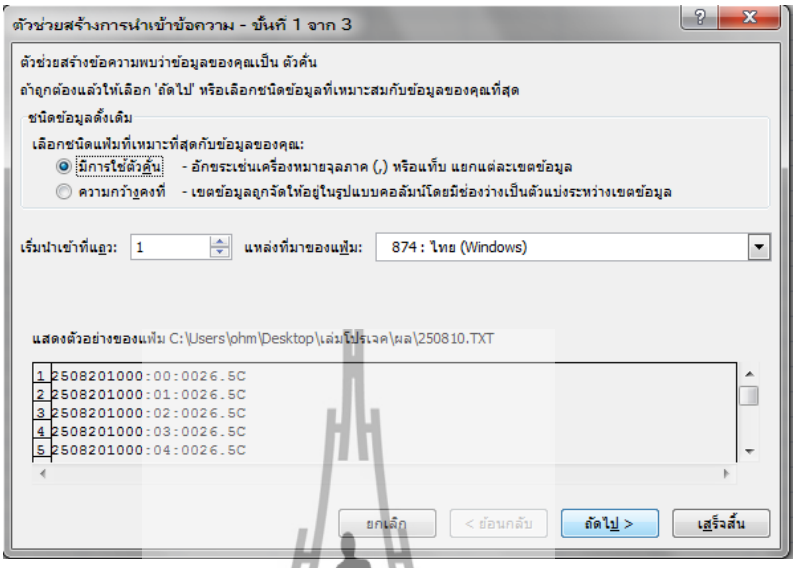
รูปที่ 3.23 การเปิดนำไฟล์ข้อมูลเข้ามา

3. เลือกที่อยู่ที่เราเก็บข้อมูลไว้และเนื่องจากข้อมูลเราเป็นไฟล์ TXT ต้องทำการเลือกนามสกุล ไฟล์เป็นไฟล์ TXT ก่อนแล้วเลือกไฟล์ข้อมูลจากนั้นคลิกเปิด



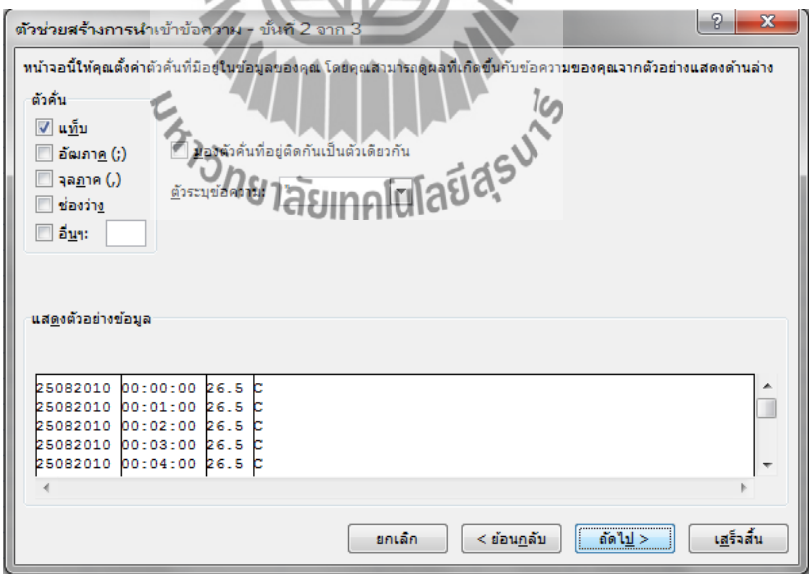
รูปที่ 3.24 การเลือกไฟล์ข้อมูล

4. ให้คลิกถัดไป



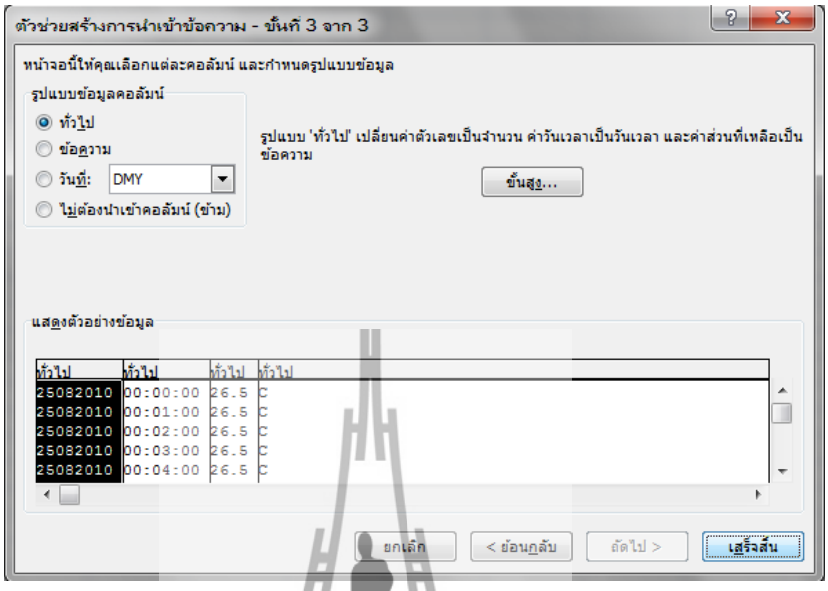
รูปที่ 3.25 หน้าต่างตัวช่วยสร้างการนำเข้าข้อความ ขั้นตอนที่ 1 จาก 3

5. คลิกถัดไป



รูปที่ 3.26 หน้าต่างตัวช่วยสร้างการนำเข้าข้อความ ขั้นที่ 2 จาก 3

6. คลิกเสร็จสิ้น



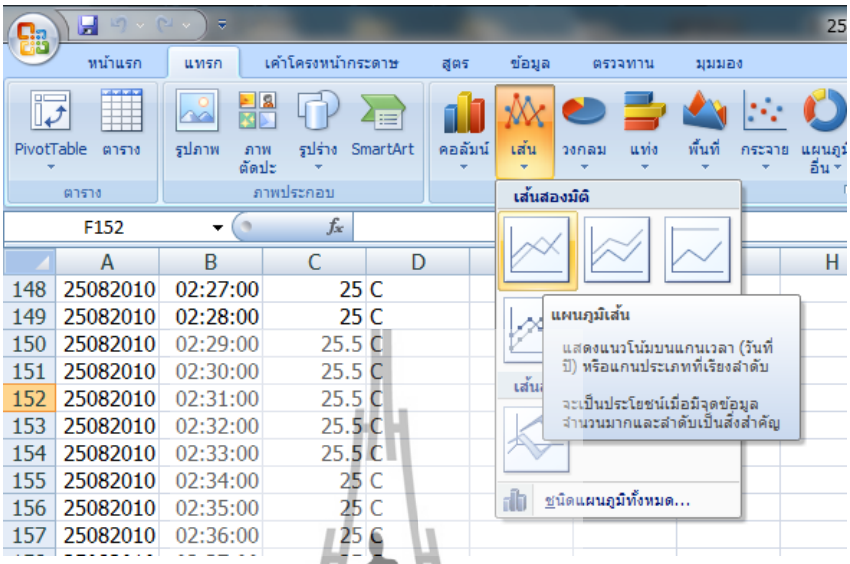
รูปที่ 3.27 หน้าต่างตัวช่วยสร้างการนำเข้าข้อความ ขั้นที่ 3จาก 3

7. จะได้ข้อมูลมา 4 แถว โดยจะมีวันที่ เวลา อุณหภูมิ และหน่วยองศาเซลเซียส

	A	B	C	D	E
148	25082010	02:27:00	25	C	
149	25082010	02:28:00	25	C	
150	25082010	02:29:00	25.5	C	
151	25082010	02:30:00	25.5	C	
152	25082010	02:31:00	25.5	C	
153	25082010	02:32:00	25.5	C	
154	25082010	02:33:00	25.5	C	
155	25082010	02:34:00	25	C	
156	25082010	02:35:00	25	C	
157	25082010	02:36:00	25	C	
158	25082010	02:37:00	25	C	
159	25082010	02:38:00	25.5	C	
160	25082010	02:39:00	25.5	C	
161	25082010	02:40:00	25.5	C	
162	25082010	02:41:00	25	C	
163	25082010	02:42:00	25	C	
164	25082010	02:43:00	25.5	C	

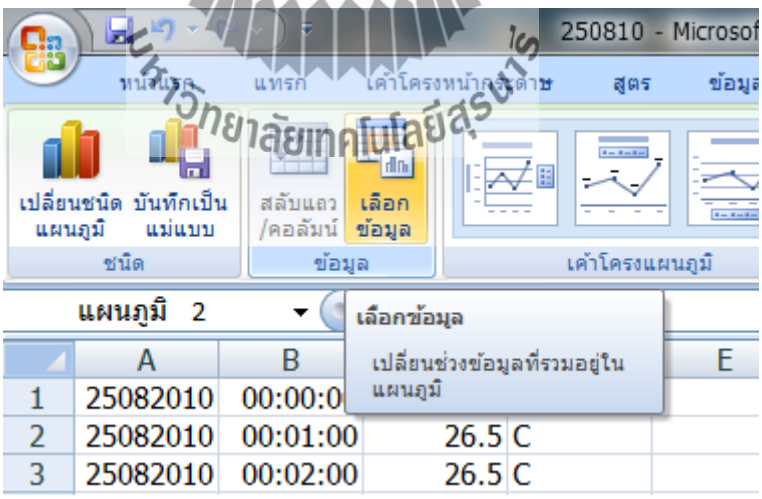
รูปที่ 3.28 ข้อมูลที่ได้จากการนำเข้าไฟล์ TXT

8. เลือกคำสั่งแทรกแล้วเลือกกราฟแบบเส้น



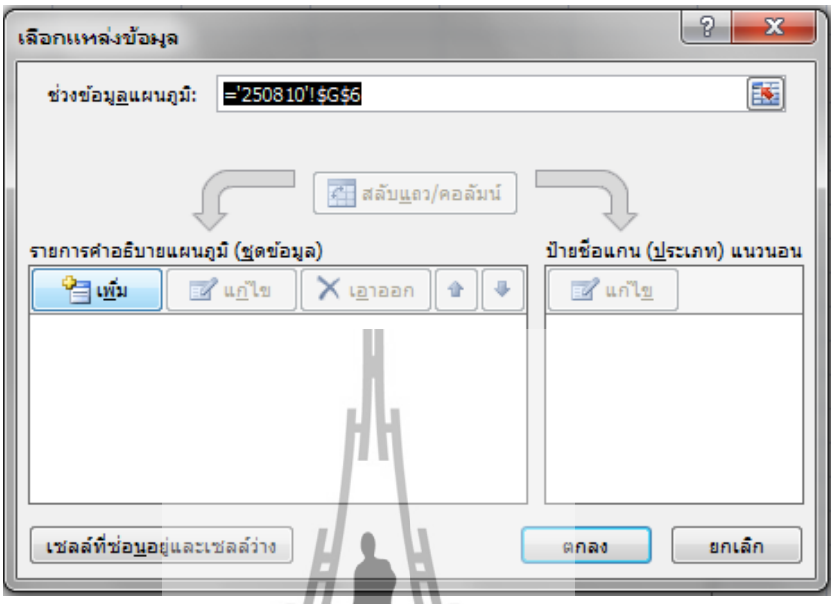
รูปที่ 3.29 การเลือกกราฟที่ใช้งาน

9. คลิกที่เลือกข้อมูล



รูปที่ 3.30 การเลือกข้อมูล

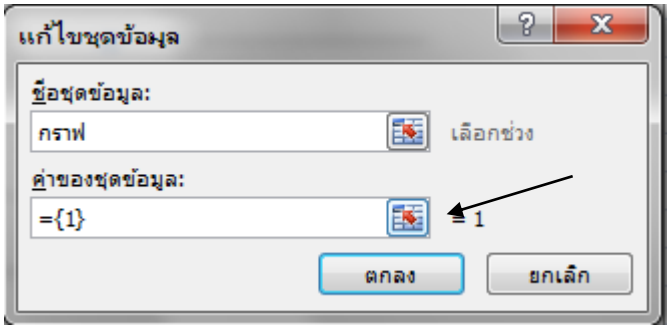
10. คลิกที่เพิ่ม



รูปที่ 3.31 การเลือกข้อมูล (ต่อ)

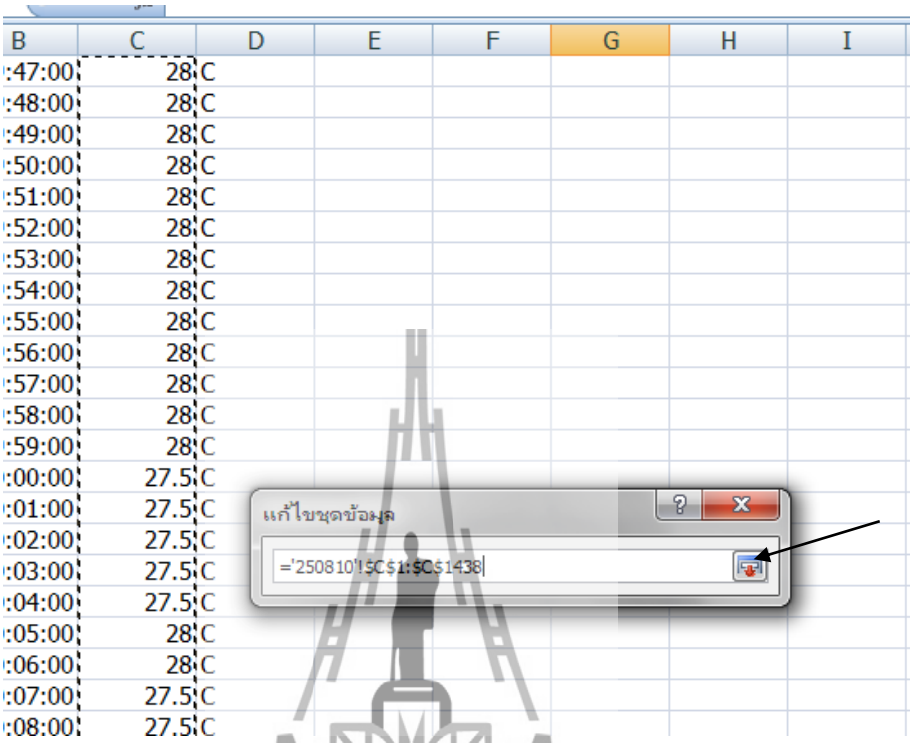
11. ช่องชื่อชุดข้อมูล ให้ใส่ชื่อที่เราต้องการ

ช่องค่าของชุดข้อมูล ให้คลิกที่ลูกศรชี้เพื่อทำการเลือกที่ข้อมูลที่ต้องการ



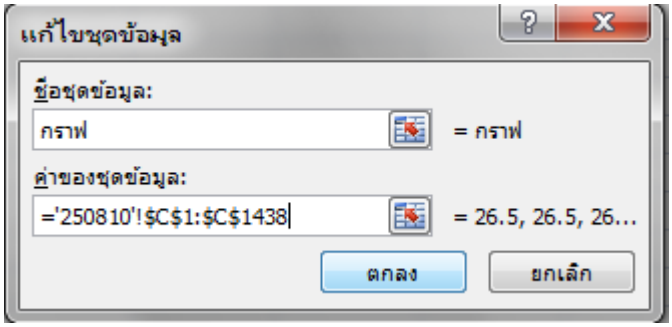
รูปที่ 3.32 การเลือกข้อมูลอุณหภูมิ

12. ทำกรอบคุมข้อมูลที่เป็นอุณหภูมิที่เราต้องการจากนั้นให้คลิกที่ลูกศรชี้



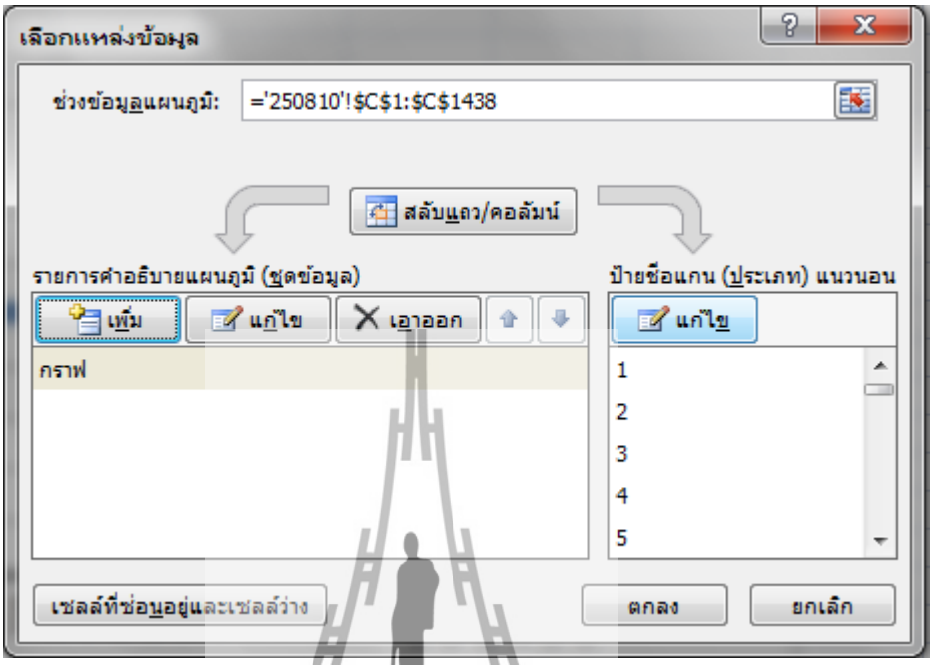
รูปที่ 3.33 การเลือกข้อมูลอุณหภูมิ (ต่อ)

13. คลิกตกลง



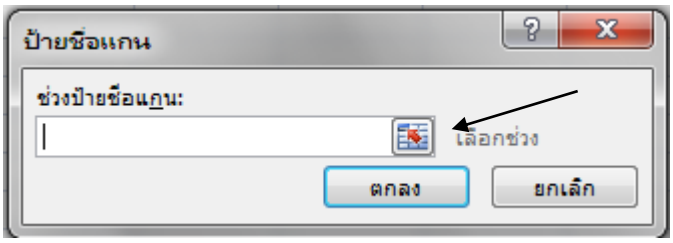
รูปที่ 3.34 การเลือกข้อมูลอุณหภูมิ (ต่อ)

14. คลิกที่แก้ไขข้อมูลเพื่อทำการเลือกข้อมูลที่เป็นเวลา



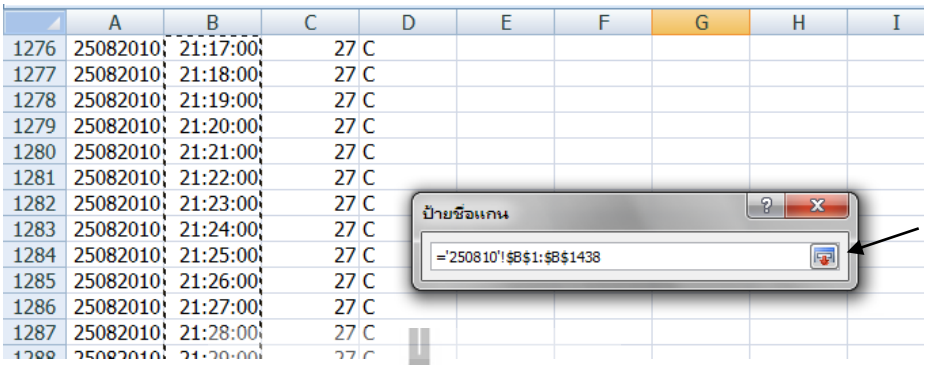
รูปที่ 3.35 การเลือกข้อมูลเวลา

15. คลิกที่ลูกศรชี้



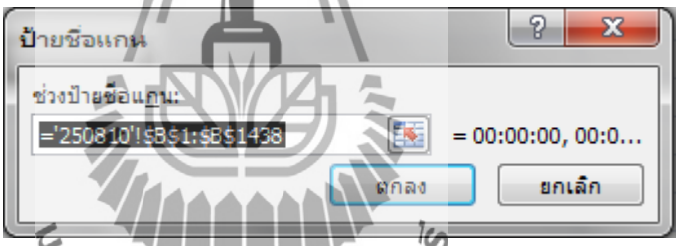
รูปที่ 3.36 การเลือกข้อมูลเวลา (ต่อ)

16. ทำการเลือกข้อมูลในส่วนที่เป็นเวลาแล้วคลิกที่ลูกศรชี้



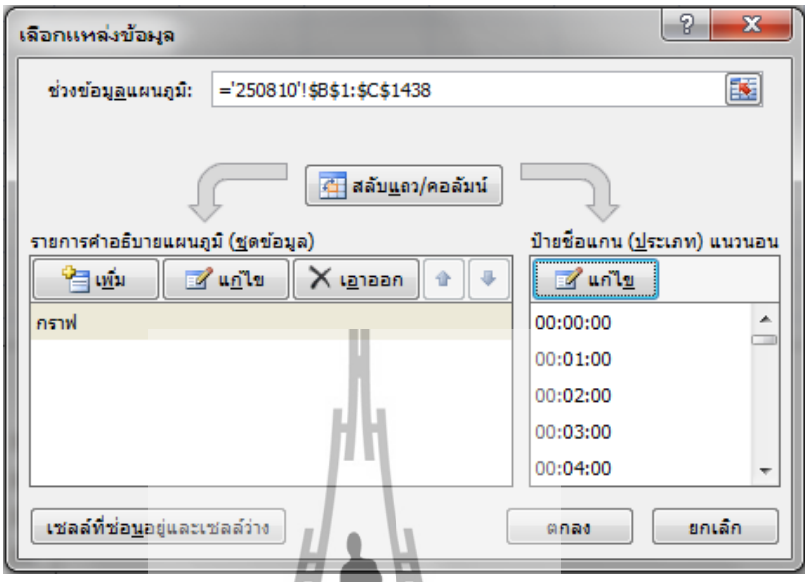
รูปที่ 3.37 การเลือกข้อมูลเวลา (ต่อ)

17. คลิกตกลง



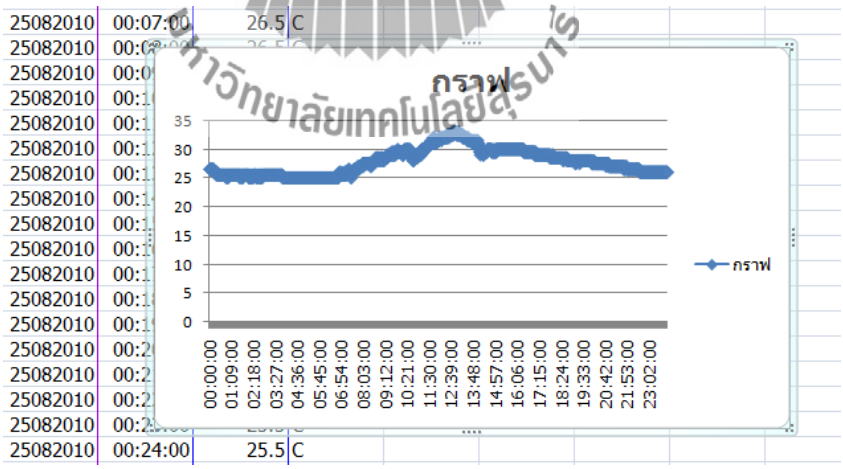
รูปที่ 3.38 การเลือกข้อมูลเวลา (ต่อ)

18. คลิกตกลง



รูปที่ 3.39 เสร็จสิ้นในการเลือกข้อมูล

19. กราฟที่ได้จากข้อมูล



รูปที่ 3.40 กราฟข้อมูลที่ได้จากการเลือกข้อมูล

3.2.7 การทดสอบการทำงาน

หลังจากที่ผ่านการทดสอบการทำงานเบื้องต้นเรียบร้อยแล้ว ที่หน้าจอจะปรากฏดังรูปที่ 3.2 ซึ่งการแสดงผลจะประกอบไปด้วย ค่าอุณหภูมิที่อ่านได้จาก DS1820 เป็นค่าอุณหภูมิในหน่วยองศาเซลเซียส การแสดงผลในส่วนใหญ่ของแอลซีดีจะเป็นการพล็อตค่าอุณหภูมิและส่วนที่เหลือ (ด้านล่างของจอ) จะเป็นการแสดงค่าอุณหภูมิแบบตัวเลขและเวลา(ชั่วโมง, นาทีและวินาที)

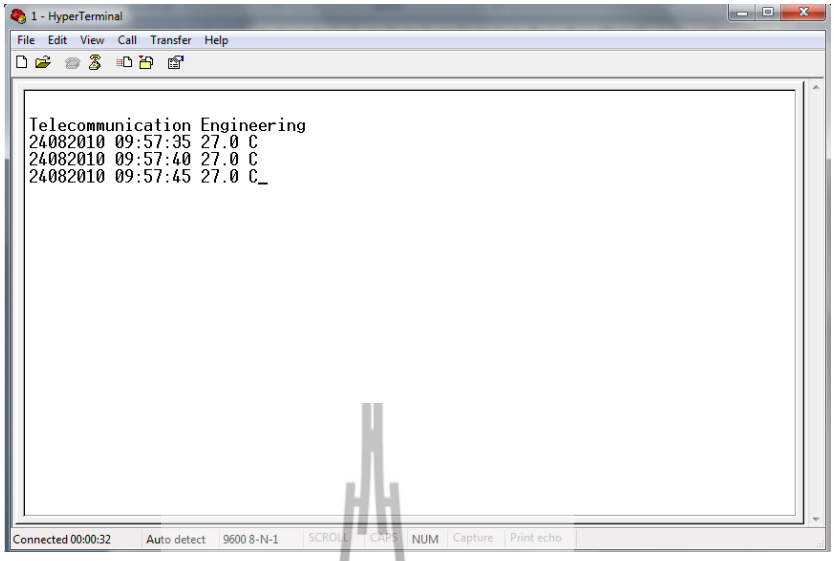
การตั้งค่าแสดงผลต่างๆสามารถทำได้ในโหมดการตั้งค่า โดยการกดปุ่ม "0" บนคีย์บอร์ดขนาด 4x3 แล้วใส่รหัสผ่าน "1449" แล้วกดปุ่ม "#" เพื่อเป็นการยืนยัน หากใส่รหัสผ่านไม่ถูกต้องที่หน้าจอจะไม่สามารถเข้าโหมดการตั้งค่าได้ แต่ถ้ารหัสผ่านถูกต้องที่หน้าจอจะปรากฏเป็นเมนูข้อความ (ดังรูปที่ 10) การแสดงผลในเมนูข้อความจะเป็นแถบบาร์หากต้องการเลื่อนแถบบาร์ขึ้นให้กดปุ่มเลข "1" และหากต้องการเลื่อนแถบบาร์ลงให้กดปุ่มเลข "7" หากต้องการเข้าไปยังส่วนย่อยของเมนูใดๆ ให้เลื่อนแถบบาร์มายังเมนูย่อยนั้นแล้วกดเครื่องหมาย "#" เพื่อเป็นการยืนยันการเข้าโปรแกรมนั้นๆสำหรับการตั้งค่าต่างๆมีดังนี้

การตั้งค่า วัน /เดือน/ปี (Set Date) ซอฟแวร์จะมีการป้องกันการป้อนค่าที่ไม่ถูกต้องอาทิเช่น โดยปกติแล้วเดือนเมษายนมี 30 วัน ดังนั้นวันที่ถูกต้องจะต้องมีค่าระหว่าง 1-30 รวมถึงการคำนวณสำหรับเดือนกุมภาพันธ์ กล่าวคือเดือนกุมภาพันธ์ในปีที่หารด้วย 4 ลงตัวจะมี 29 วัน ซึ่งการป้อนข้อมูลจะต้องทำให้ถูกต้อง หากข้อมูลถูกต้องค่าใหม่ที่ผ่านการตั้งค่าจะถูกอัพเดทไปยัง DS1307

การตั้งค่าเวลา (Set Time) ในส่วนนี้ก็มีการป้องกันการป้อนค่าที่ไม่ถูกต้องด้วย คือ การป้อนค่าชั่วโมงจะต้องอยู่ระหว่าง 00-23 และในการป้อนค่านาทีและวินาทีจะมีค่าอยู่ระหว่าง 00-59 หากการป้อนค่าต่างๆไม่ถูกต้องจะมีข้อความ "Can't Update" ฟ้องที่หน้าจอเพื่อแสดงให้ผู้ใช้ทราบว่าเกิดความผิดพลาดเกี่ยวกับป้อนค่า

ในส่วนเมนูการตั้งค่าที่ 3 และ 4 เป็นการเลือกรูปแบบของการแสดงผล ระหว่างการแสดงผลแบบเชิงเส้นทึบหรือแบบจุด (ดังรูปที่ 11 และ 12 ตามลำดับ) ส่วนสุดท้ายเป็นการออกจากโหมดการตั้งค่า (Exit) ในเมนูนี้หากกดเครื่องหมาย "#" จะเป็นการออกจากโหมดตั้งค่าไปยังหน้าจอการทำงานปกติ

การทำงานของไฟพื้นหลังจะปิดหลังจากที่ไม่มีการกดสวิตช์จากผู้ใช้ 1 นาที กล่าวคือ หากมีการกดคีย์บอร์ดใดๆๆไฟพื้นหลังจะติดสว่าง หลังจากนั้นหากไม่มีการปุ่มครบกำหนด 1 นาที ไฟพื้นหลังจะดับลงไป ในกรณีนี้เพื่อเป็นการยืดชั่วโมงการทำงานของไฟพื้นหลังให้มีช่วงการทำงานให้นานขึ้น ซึ่งไฟพื้นหลังนั้นไม่มีความจำเป็นที่ต้องติดตลอดเวลา (แต่ถ้าต้องการให้ไฟพื้นหลังติดตลอดเวลาก็สามารถแก้ไขที่ฮาร์ดแวร์หรือที่ซอสโค้ดได้ตามความสะดวกของผู้ใช้)

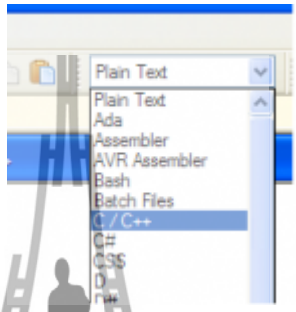


รูปที่ 3.41 ข้อมูลที่ส่งผ่าน พอร์ต RS232

3.3 เขียนโปรแกรมด้วย Programmer Notepad กับ WinAVR

การเขียนโปรแกรมภาษา C ร่วมกับใช้ AVR-GCC Compiler. Programmer Notepad เป็น Freeware ที่สามารถดาวน์โหลดมาใช้งานโดยไม่ต้องมีค่าใช้จ่ายใดๆ

1. ขั้นตอนแรกให้ทำการเปิดโปรแกรมแล้วเลือก Plain text เป็น C / C++



รูปที่ 3.42 การเลือกภาษาในการเขียน

เขียนโปรแกรมด้านล่างนี้ลงใน editor

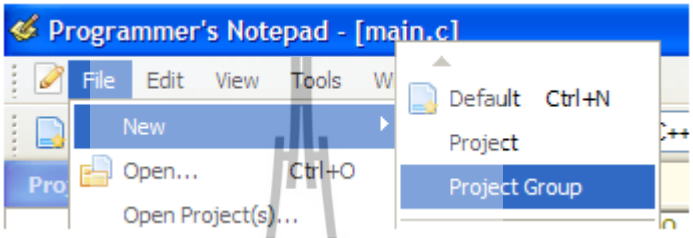
```
#include "avr/io.h"
#include "util/delay.h"

int main (void){
/* set PORTB for output*/
DDRB = 0xFF;
while (1) {
/* set PORTB.6 high */
PORTB = 0x20;
_delay_ms(1000);
/* set PORTB.6 low */
PORTB = 0x04;
_delay_ms(1000);
```

```
}
return 1;
}
```

2.จากนั้น Save ไฟล์ชื่อว่า main.c

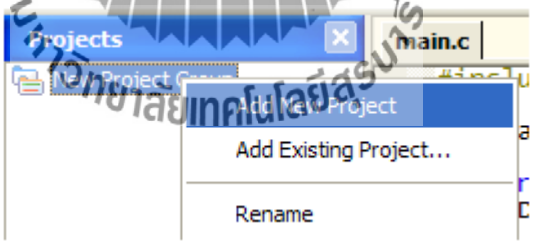
แล้วสร้าง Project Group ใหม่โดยไปที่ File > New > Project Group



รูปที่ 3.43 การสร้างกลุ่มโปรเจค

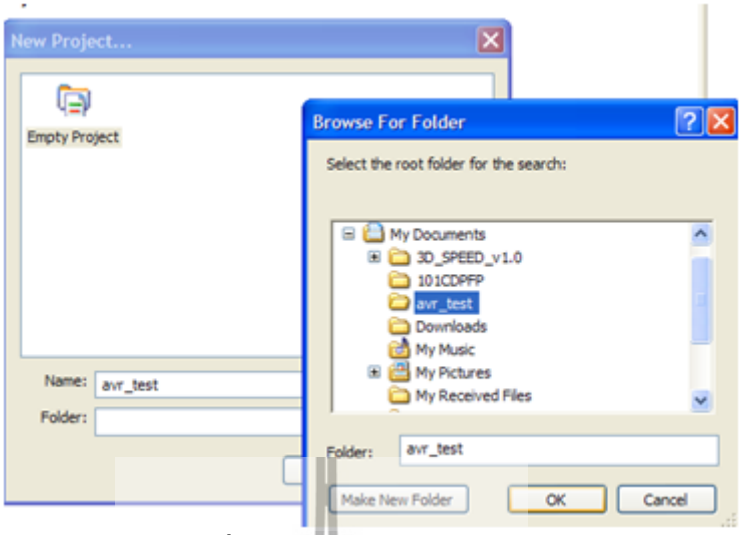
สร้าง Project Group

หลังจากที่สร้าง Group แล้ว ให้ Add New Project ใหม่โดยคลิกขวาที่ New Project Group> Add New Project > OK ตามรูปภาพด้านล่างนี้



รูปที่ 3.44 การสร้างโปรเจค

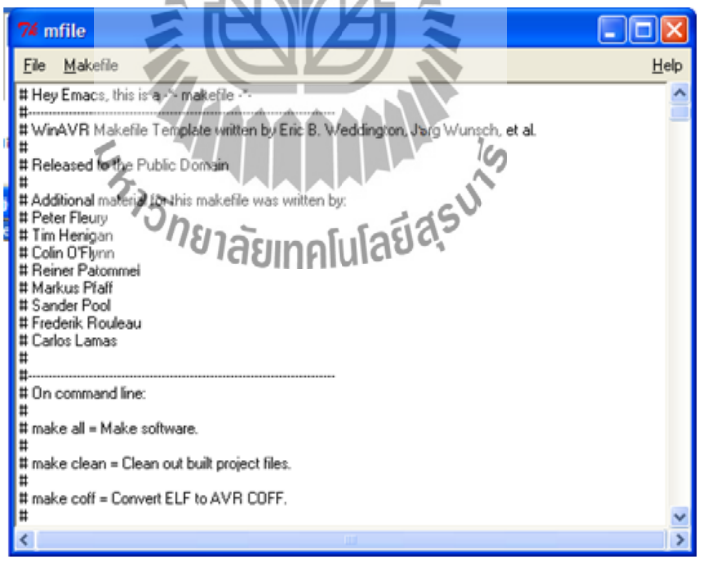
New Project Group > Add New Project



รูปที่ 3.45 เลือกที่เก็บไฟล์โปรเจค

ตั้งชื่อแล้วเลือก Folder ที่สร้างไว้ก่อนหน้านั้น

หลังจากนั้น Save Project Group จากนี้ให้สร้างไฟล์ Make โดยไปที่ WinAVR-20090313 > MFile [WinAVR] เมื่อคลิกแล้วจะปรากฏหน้าต่างดังภาพด้านล่างนี้



รูปที่ 3.46 หน้าต่าง Mfile

จากนั้นไปที่ เมนู Makefile > Enable Editng of Makefile จากนั้นให้ปรับแก้ไขดังนี้

Main file name: main

MCU Type: Atmega128

Output Format: ihex

Optimization level: S

Debug Format: ELF/DWARF-2 (AVR Studio 4.11+)

C standard level : gnu99

Programmer : avrisp

Port : com1

จากนั้น Save ไฟล์ไปยังทำแหน่ง Path ที่ได้สร้าง Project Group

จากนั้น คลิกขวาที่ avr_test > Add Files และทำการเลือกไฟล์ main.c เข้ามาใน Project และอีก 1 ไฟล์ที่ขาดไม่ได้ก็คือ ไฟล์ Makefile ก็ให้เพิ่มเข้ามาเช่นกัน จากนั้นทำการแก้ไข Makefile ให้เปลี่ยน F_CPU = 16000000 ใหม่เสียก่อน (อ้างอิงตาม XTALที่บอร์ดทดลอง) ผลที่ได้ตามรู้ด้านล่างนี้

รูปที่ 3.47 การ Save ไฟล์ไปยังทำแหน่ง Path ที่ได้สร้าง Project Group

คอมไพล์โปรแกรม ก่อนอื่นทำการ Clean ก่อนโดยเลือก Tools > [WinAVR] Make Clean ผลการ Make Clean ดังนี้

> “make.exe” clean

——— begin ———

Cleaning project:

rm -f main.hex

rm -f main.eep

rm -f main.cof

rm -f main.elf

rm -f main.map

rm -f main.sym

rm -f main.lss

rm -f ./main.o

rm -f ./main.lst

rm -f main.s

rm -f main.d

rm -f main.i

rm -rf .dep

——— end ———

> Process Exit Code: 0

> Time Taken: 00:01

จากนั้นทำการ Makefile เพื่อ คอมไพล์ไฟล์ main.c โดยเลือก Tools > [WinAVR] Make All ผลจากการคอมไพล์

> “make.exe” all

——— begin ———

avr-gcc (WinAVR 20090313) 4.3.2

Copyright (C) 2008 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO

warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiling C: main.c

avr-gcc -c -mmcu=atmega128 -I. -gdwarf-2 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=./main.lst -std=gnu99 -MMD -MP -MF .dep/main.o.d main.c -o main.o

Linking: main.elf

avr-gcc -mmcu=atmega128 -I. -gdwarf-2 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=main.o -std=gnu99 -MMD -MP -MF .dep/main.elf.d main.o –output main.elf -Wl,-Map=main.map,–cref -lm

Creating load file for Flash: main.hex

avr-objcopy -O ihex -R .eeprom -R .fuse -R .lock main.elf main.hex

Creating load file for EEPROM: main.eep

avr-objcopy -j .eeprom –set-section-flags=.eeprom=”alloc,load” \

–change-section-lma .eeprom=0 –no-change-warnings -O ihex main.elf main.eep || exit 0

Creating Extended Listing: main.lss

avr-objdump -h -S -z main.elf > main.lss

Creating Symbol Table: main.sym

avr-nm -n main.elf > main.sym

Size after:

AVR Memory Usage

————————-

Device: atmega128

Program: 240 bytes (0.2% Full)

(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)

(.data + .bss + .noinit)

——— end ———

> Process Exit Code: 0

> Time Taken: 00:00

ให้สังเกตจุดนี้หากมีความผิดพลาดใดๆ จะมีการรายงานผล ขณะนี้ไม่มี error ใดๆ เกิดขึ้น นั้นแสดงว่า Compiler สามารถแปลโปรแกรมที่เราเขียน และได้สามารถ hex ไฟล์ให้เรียบร้อยแล้ว hex ไฟล์ที่ได้จะอยู่ที่ Folder avr_test ชื่อว่า main.hex ซึ่งสามารถเปิดดูรายละเอียดภายในได้เช่นกัน สามารถใช้โปรแกรมนี้เปิดได้

ข้อมูล main.hex แสดงดังนี้

[code lang="c"]

:100000000C9446000C945D000C945D000C945D0013

:100010000C945D000C945D000C945D000C945D00EC

:100020000C945D000C945D000C945D000C945D00DC

:100030000C945D000C945D000C945D000C945D00CC

:100040000C945D000C945D000C945D000C945D00BC

:100050000C945D000C945D000C945D000C945D00AC

:100060000C945D000C945D000C945D000C945D009C

:100070000C945D000C945D000C945D000C945D008C

:100080000C945D000C945D000C945D0011241FBE67

:10009000CFEFD0E1DEBFCDBF11E0A0E0B1E0E0EFF7

:1000A000F0E000E00BBF02C007900D92A030B10756

:1000B000D9F70E945F000C9476000C9400008FEF3B

:1000C00087BB50E220E931E044E058BB80E197E291

:1000D000F9013197F1F70197D9F748BB80E197E231

:1000E000F9013197F1F70197D9F7EFCFF894FFCFE6

:00000001FF

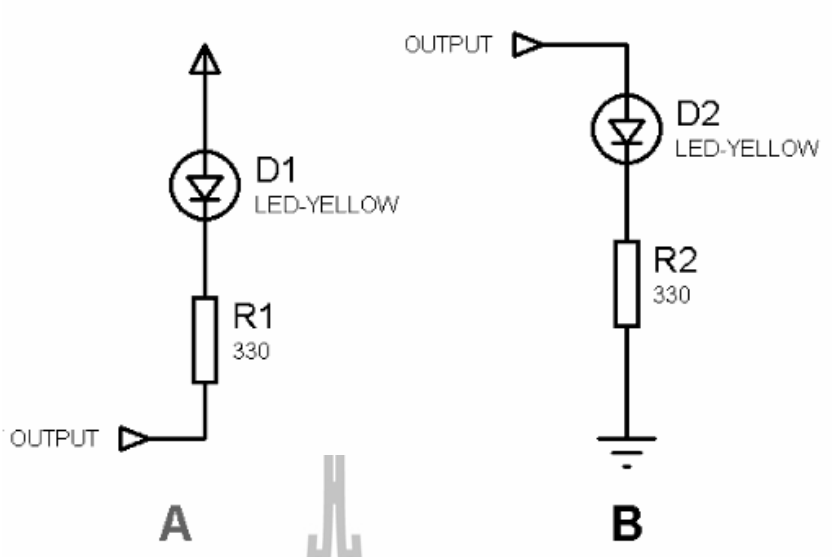
[/code]

3.3.1 การเขียนโปรแกรมแสดงผลบนอุปกรณ์แอลอีดี (Programming output on LED device)

ทฤษฎีพื้นฐาน

3.3.1.1 การเอาท์พุตออกอุปกรณ์แอลอีดี

การเอาท์พุตออกอุปกรณ์แอลอีดี (LED) ให้พิจารณาการต่อของวงจรแอลอีดี โดยทั่วไปจะต่อได้ 2 แบบคือ แบบ Inverting Output (รูปที่ 3.19 A) และแบบ Non-Inverting Output (รูปที่ 3.19 B) การต่อวงจรตามรูปที่ 3.19 B เมื่อป้อนออกตรรกะ 0 ให้หลอดแอลอีดีจะดับ และเมื่อป้อนออกตรรกะ 1 ให้หลอดแอลอีดีจะติดสว่าง เรียกได้ว่า การป้อนออกแบบตรง (Non-Inverting Output) แต่ในทางกลับกันเมื่อต่อวงจรตามรูปที่ 3.19 A การทำงานจะเปลี่ยนแปลงกลับจากรูปที่ 3.19 B นั้นคือ เมื่อป้อนออกตรรกะ 1 หลอดแอลอีดีจะสลับเป็นดับและเมื่อป้อนตรรกะ 0 หลอดแอลอีดีจากดับสลับเป็นติดสว่างขึ้นมา เรียกได้ว่า การป้อนออกแบบพลิกกลับ (Inverting Output) ซึ่งวิธีการป้อนออกแบบนี้เป็นที่นิยมมากกว่าการป้อนออกแบบตรง เนื่องวงจรไอซีทั่วไปสามารถขับกระแสขาเข้า (Sink Current)ได้ดีกว่าการขับกระแสขาออก (Source) อย่างไรก็ตามเมื่อต้องการที่จะขับกระแสหลอดแอลอีดีนั้นจำเป็นต้องสังเกตวงจรก่อนว่ามีการต่อในลักษณะใด



รูปที่ 3.48 ตัวอย่างวงจรแอลอีดีแบบ Inverting Output (A) และ Non-Inverting Output (B)

3.3.1.2 การควบคุมด้วยไมโครคอนโทรลเลอร์ AVR

การเขียนโปรแกรมติดต่ออินพุตเอาท์พุตของไมโครคอนโทรลเลอร์ AVR

ในการเขียนโปรแกรมติดต่อไมโครคอนโทรลเลอร์ จะต้องมีการ Include แฟ้ม Header ที่เกี่ยวข้องมาใช้งาน กรณีนี้ถ้าต้องการควบคุมการใช้งานพอร์ทขาต่าง ๆ ของไมโครคอลโทรลเลอร์ AVR จะต้อง include ไฟล์ Header “avr/io.h” ซึ่งมีการนิยามตัวแปร ฟังก์ชั่น และแมโครเกี่ยวกับไมโครคอลโทรลเลอร์ AVR รุ่นต่างๆไว้รูปแบบคำสั่งสามารถเขียนได้ดังนี้คือ

```
#include <avr/io.h>
```

กำหนดหน้าที่และค่าการทำงานเริ่มต้นของไมโครคอลโทรลเลอร์ AVR เพื่อใช้งานขาต่างๆ ทำได้โดยการตั้งค่ากลุ่มรีจิสเตอร์ (Registers) ของแต่ละพอร์ตขนาด 8 บิต ตามรายการดังนี้

1. รีจีสเตอร์ DDRx (Port Data Direction Register) ทำหน้าที่กำหนดทิศทางของสัญญาณ ของขาไมโครคอนโทรลเลอร์ให้เป็นการส่งออก (output) หรือรับเข้า (input) หากกำหนดเป็นลอจิก 0 จะ

เป็นอินพุตและหากกำหนดเป็นลอจิก 1 จะเป็นเอาท์พุต ตัวอย่าง กำหนดพอร์ต B ขาที่ 0-3 เป็นอินพุต และ ขาที่ 4-7 เป็นเอาท์พุต

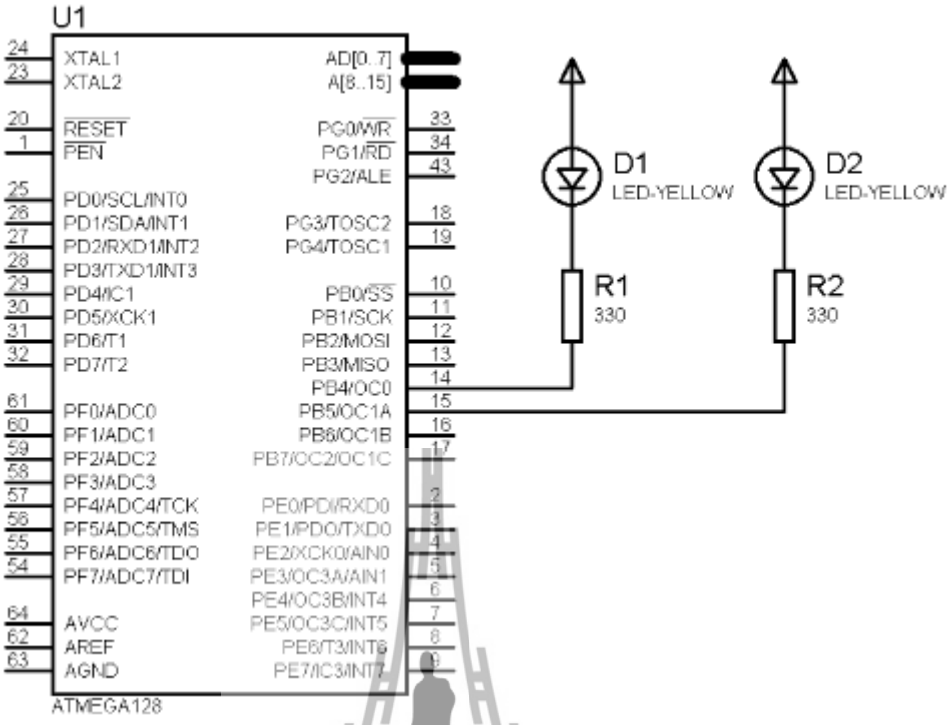
```
DDRB = 0xF0; // 1111 0000
```

2. รีจิสเตอร์ PORTx (Port x Data Register) ทำหน้าที่เก็บค่าที่จะส่งออก (output) แล้วส่งออกไปยังขาไมโครคอนโทรลเลอร์ หรือกำหนดการดึงขึ้น (pull up) ภายในด้วยการตั้งค่าบิตเป็น 1 เมื่อขามีทิศทางการรับเข้า (input) ตัวอย่าง การส่งค่าลอจิก 1 ออกที่พอร์ต B ขาที่ 6

```
DDRB = 0xF0; // 1111 0000
PORTB = 0x40; // 0100 0000
```

3. รีจิสเตอร์ PINx (Port x Input Pins Address) ทำหน้าที่รับค่า (input) สัญญาณที่เปลี่ยนแปลงบนขานั้น ซึ่งสามารถรับได้แม้มีทิศทางส่งออกตัวอย่าง การรับค่าที่พอร์ต B ขาที่ 3

```
DDRB = 0xF0; // 1111 0000
if ((PINB & 0x08) == 0) // 0000 1000
{
}
```



รูปที่ 3.49 ตัวอย่างการต่อแอลอีดีกับไมโครคอนโทลเลอร์ AVR

3.3.1.3 การเขียนโปรแกรมรับค่าจากสวิตช์โดยใช้อินเตอร์รับภายนอก (Programming input from switch device with external interrupt)

ทฤษฎีพื้นฐาน

1. การอินเตอร์รับ (Interrupt)

การอินเตอร์รับ คือ กระบวนการขัดจังหวะการทำงานของโปรแกรมปกติหรือโปรแกรมหลักที่กำลังทำงานอยู่ เพื่อให้เปลี่ยนมาทำงานในส่วนของโปรแกรมที่ได้กำหนดไว้ในอินเตอร์รับกระบวนการนี้จะช่วยให้ลดความเสี่ยงการเกิดข้อผิดพลาด จากการตรวจสอบเงื่อนไขในโปรแกรมหลักโดยกำหนดหน้าที่การตรวจสอบนี้ให้กับอินเตอร์รับแทน ประเภทของอินเตอร์รับในไมโครคอนโทรลเลอร์ จะแบ่งออกเป็น 2 ประเภทคือ

1. อินเตอร์รับภายนอก (External Interrupt) เป็นการตรวจสอบสัญญาณที่รับมาจากภายนอกตัวไมโครคอนโทรลเลอร์

2. อินเตอร์รับภายใน (Internal Interrupt) เป็นการตรวจสอบสัญญาณที่แหล่งกำเนิดสัญญาณเกิดจากวงจรภายในไมโครคอนโทรลเลอร์เอง

อินเตอร์รับภายนอก (External Interrupt)

การอินเตอร์รับภายนอกเป็นการตรวจสอบสัญญาณที่รับจากภายนอกตัวไมโครคอนโทรลเลอร์ ที่ขาของพอร์ตต่างๆ ตามที่ไมโครคอนโทรลเลอร์นั้นๆ อนุญาต (โดยดูได้จาก Data sheet ของไมโครคอนโทรลเลอร์นั้นๆ) เช่น Atmega64 จะมีขาที่เป็นอินเตอร์รับภายนอกได้คือ PD0 (INT0), PD1 (INT1), PD2 (INT2), PD3 (INT3), PE4 (INT4), PE5 (INT5), PE6 (INT6) และ PE7 (INT7) โดยสามารถ

กำหนดรูปแบบของสัญญาณการเกิดอินเตอร์รับภายนอกได้หลายรูปแบบดังนี้

1. ขณะที่ระดับสัญญาณเป็นลอจิกต่ำ
2. ขณะที่ระดับสัญญาณเปลี่ยนแปลง
3. ขณะที่ขอบขาลงของสัญญาณ (Falling edge)
4. ขณะที่ขอบขาขึ้นของสัญญาณ (Rising edge)

การควบคุมด้วยไมโครคอนโทรลเลอร์ AVR

1. การเขียนโปรแกรมใช้อินเตอร์รับภายนอกของไมโครคอนโทรลเลอร์ AVR

ในการเขียนโปรแกรมใช้งานอินเตอร์รับจะต้อง include ไฟล์ “avr/interrupt.h” และการตั้งค่ารูปแบบของอินเตอร์รับภายนอกสำหรับ Atmega64 สามารถกำหนดได้จากรีจีสเตอร์ EICRA, EICRB, EIMSK โดยหน้าที่ของแต่ละรีจีสเตอร์คือ

1. รีจีสเตอร์ EICRA จะกำหนดรูปแบบของสัญญาณการเกิดอินเตอร์รับ INT0 – INT3
2. รีจีสเตอร์ EICRB จะกำหนดรูปแบบของสัญญาณการเกิดอินเตอร์รับ INT4 – INT7
3. รีจีสเตอร์ EIMSK จะกำหนดเปิด/ปิดการใช้อินเตอร์รับ INT0 – INT7

โดยก่อนการเข้าไปกำหนดค่ารีจีสเตอร์ของอินเตอร์รับ จะต้องเรียกคำสั่งหยุดการทำงานของอินเตอร์รับทั้งหมดก่อน (รวมถึงอินเตอร์รับภายในด้วย) เพื่อไม่ให้เกิดการซ้อนกันของคำสั่ง โดยใช้คำสั่ง cli เพื่อหยุดอินเตอร์รับและ sei เพื่อเริ่มทำงานอินเตอร์รับ

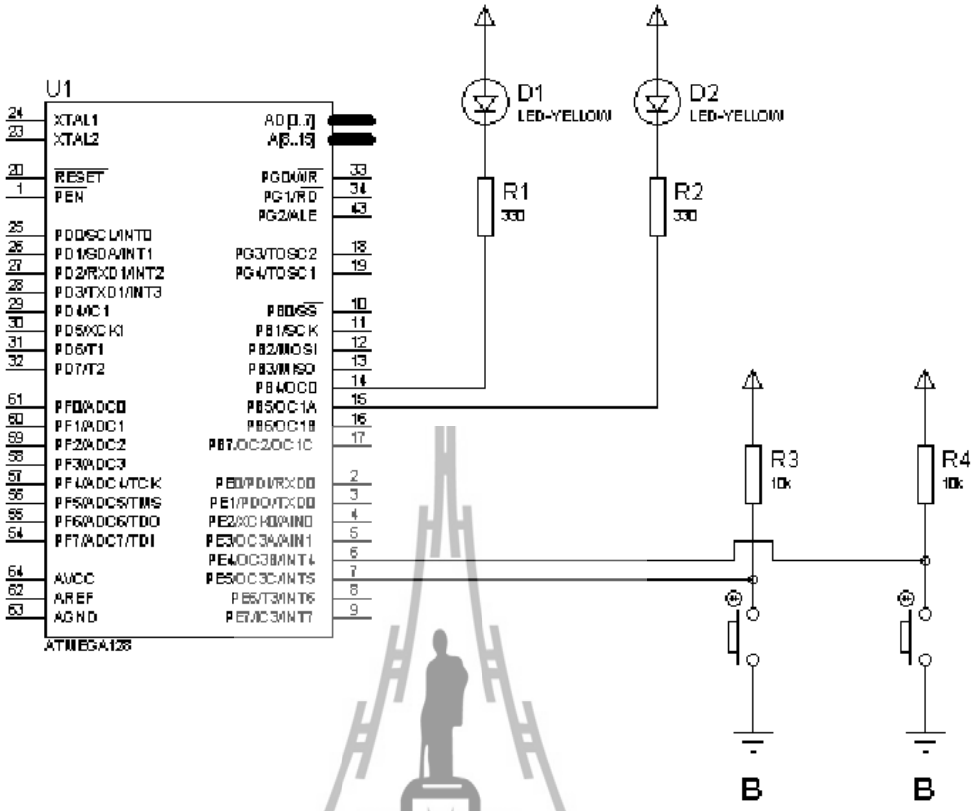
ตัวอย่าง การกำหนดค่าอินเตอร์รับ INT3 (PD3) และ INT4 (PE4)

```
#include <avr/io.h>
#include <avr/interrupt.h>
cli();
EICRA = _BV(ISC30)|_BV(ISC31); // INT3 is rising edge
EICRB = _BV(ISC40)|_BV(ISC41); // INT4 is rising edge
EIMSK = _BV(INT3)|_BV(INT4); // INT3, INT4 is enable
sei();
```

เมื่อเกิดการทำงานของอินเตอร์รับขึ้น โปรแกรมจะทำการกระโดดไปทำงานที่บรรทัดของอินเตอร์รับเวกเตอร์ของอินเตอร์รับนั้นๆ ซึ่งสำหรับอินเตอร์รับภายนอกนั้นอินเตอร์รับเวกเตอร์คือ

INTx_vect และสามารถเรียกใช้งานได้ดังตัวอย่างต่อไปนี้

```
ISR(INT4_vect)
{
//...
}
```

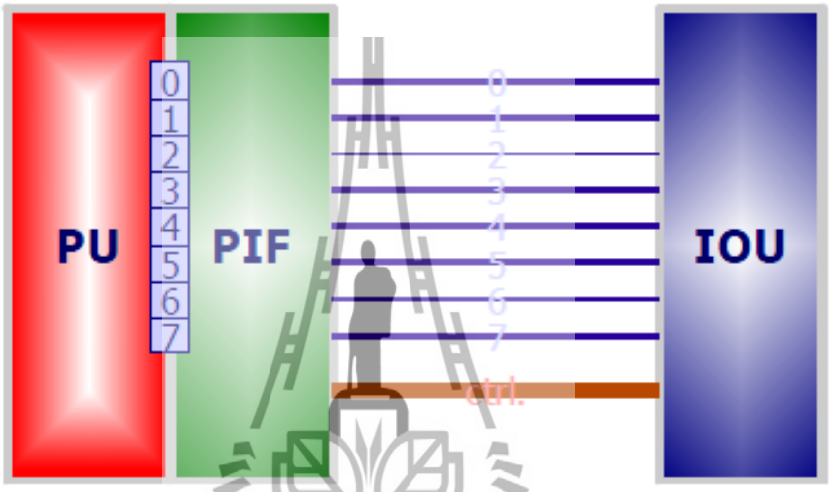


รูปที่ 3.50 ตัวอย่างการต่อสวิตช์กับไมโครคอนโทลเลอร์ AVR

3.3.1.4 การเขียนสื่อสารข้อมูลอนุกรมผ่านทาง UART (Programming serial communication by UART)

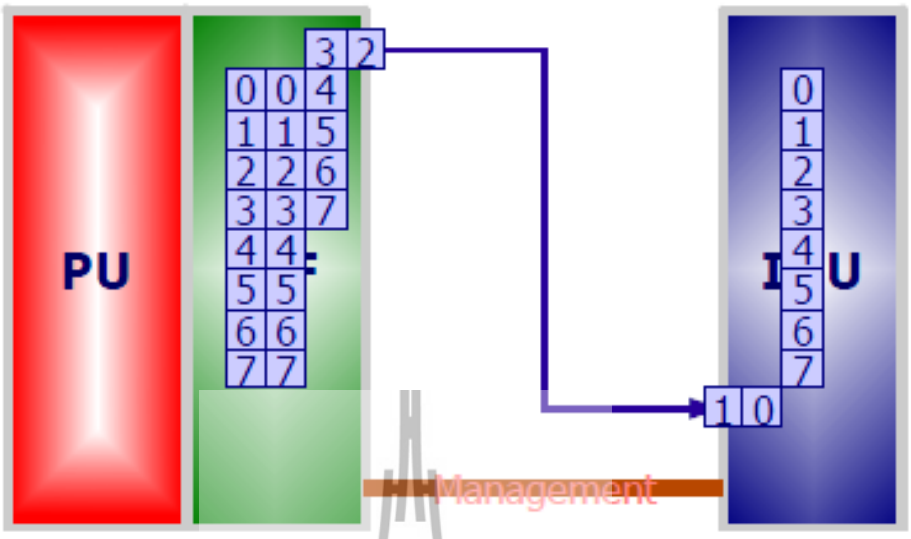
ทฤษฎีพื้นฐาน

1. การสื่อสารข้อมูลแบบอนุกรม (Serial communication) จากการเรียนรู้ที่ผ่านมาการรับส่งข้อมูลผ่านทางพอร์ตจะเป็นการสื่อสารแบบขนาน เพราะเวลาส่งข้อมูลขนาด 1 ไบต์ต้องใช้ขาสัญญาณในการส่งเท่ากับจำนวนบิตทั้งหมดคือ 8 ขา



รูปที่ 3.51 ตัวอย่างการสื่อสารข้อมูลแบบขนาน

เพื่อที่จะลดจำนวนขาในการสื่อสาร จึงต้องมีการสื่อสารแบบอนุกรมเข้ามาช่วย ถึงแม้ว่าการสื่อสารแบบอนุกรมจะใช้ขาสัญญาณที่น้อยกว่าและ สามารถส่งได้ไกลกว่า การสื่อสารแบบอนุกรมก็ยังมีข้อด้อยเรื่องความเร็วในการสื่อสารที่น้อยกว่าแบบขนาน และการเพิ่มเติมวงจรเพื่อแปลงการสื่อสารแบบอนุกรมให้เป็นแบบขนาน ดังรูป



รูปที่ 3.52 ตัวอย่างการสื่อสารข้อมูลแบบอนุกรม

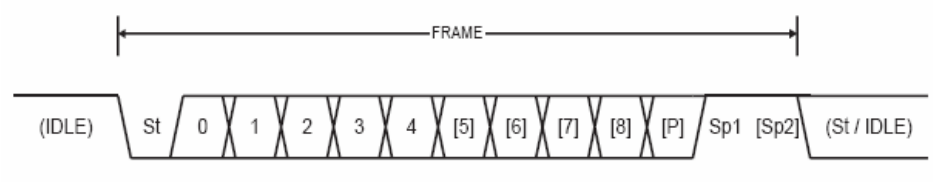
รูปแบบการสื่อสารแบบอนุกรมจะมีอยู่ 3 แบบคือ

- การสื่อสารแบบทิศทางเดียว (simplex) ซึ่งสามารถรับหรือส่งได้เพียงอย่างเดียวเท่านั้น
- การสื่อสารแบบสองทางครึ่งอัตรา (half duplex) ซึ่งสามารถรับและส่งได้แต่ไม่พร้อมกัน
- การสื่อสารแบบสองทางเต็มอัตรา (full duplex) ซึ่งสามารถรับและส่งได้ในเวลาเดียวกัน

2. การสื่อสารข้อมูลแบบ UART

การสื่อสารข้อมูลแบบ UART เป็นการสื่อสารอนุกรมในรูปแบบ full duplex ซึ่งใช้ขา Rx ในการรับข้อมูลและ ใช้ขา Tx ในการส่งข้อมูล การสื่อสารจะเป็นแบบ Asynchronous ดังนั้นจึงไม่มีขาสัญญาณในการ Synchronize ข้อมูล ทำให้ฝ่ายรับและฝ่ายส่งจำเป็นที่จะต้องรู้รูปแบบของข้อมูลและ ความเร็วในการสื่อสาร การใช้งาน UART จะเอาไปประยุกต์ใช้กับมาตรฐานการสื่อสารอนุกรมแบบ RS232 หรือ RS485 เป็นต้น

รูปแบบของข้อมูล (Frame format) ในการสื่อสารของ UART จะมีรูปแบบดังนี้



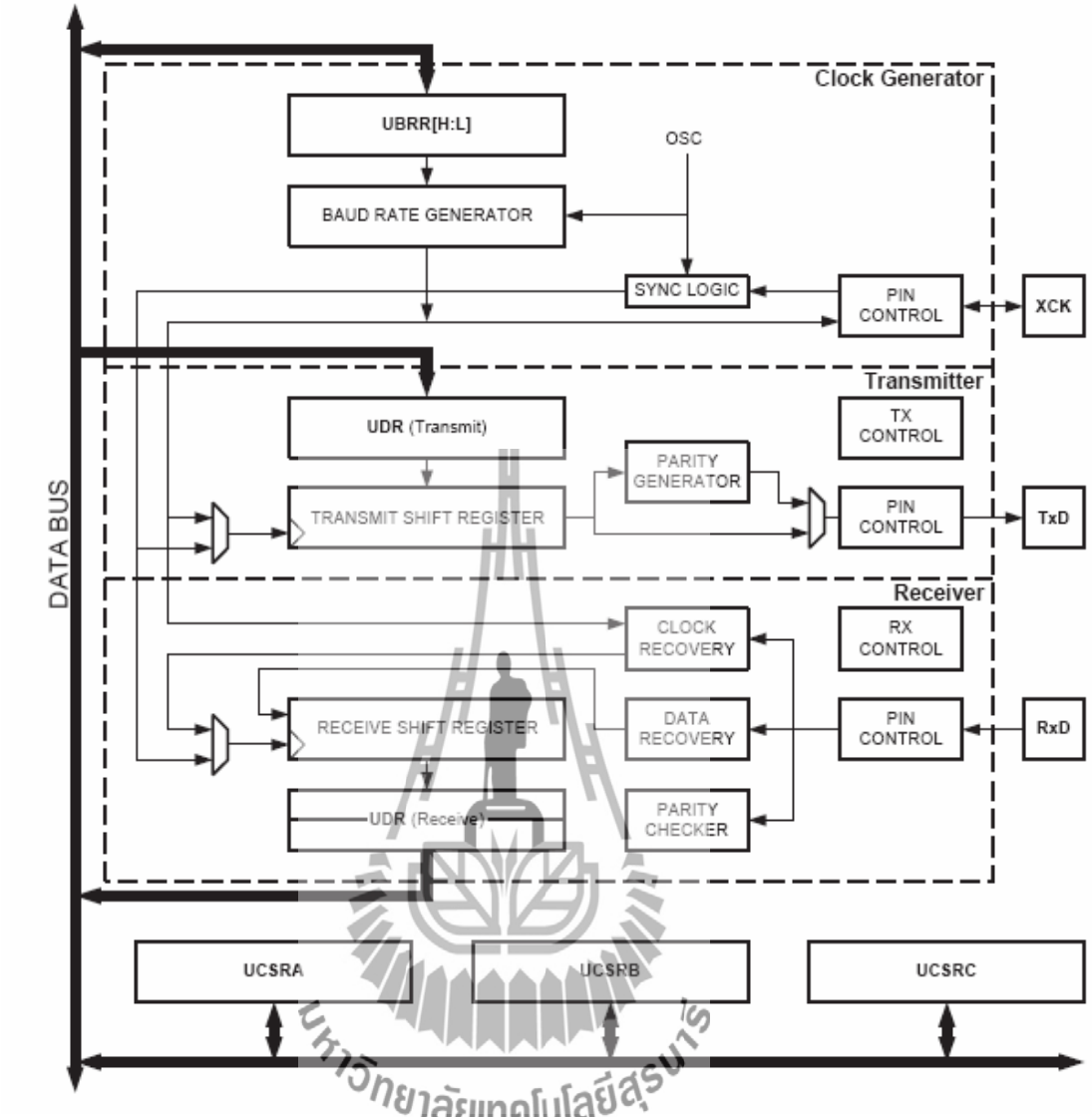
รูปที่ 3.53 Frame format

- IDLE เป็นสภาวะของระดับสัญญาณที่ไม่มีการรับหรือส่งข้อมูล ปกติมีค่าเป็น high
- St (Start bit) เป็นบิตเริ่มต้นของข้อมูล ปกติมีค่าเป็น low
- N (1 – 8) เป็นค่าข้อมูลของแต่ละบิต
- P (Parity) เป็นบิตในการตรวจสอบความถูกต้องของข้อมูลโดยการนับจำนวนบิตที่มีค่าเป็น 1 ว่าเป็นจำนวนคู่หรือคี่
- Sp (Stop bit) เป็นบิตสิ้นสุดข้อมูลซึ่งอาจจะมี 1 หรือ 2 บิตขึ้นอยู่กับการกำหนด ค่าปกติจะมีค่าเป็น high

การควบคุมด้วยไมโครคอนโทรลเลอร์ AVR

1. การเขียนโปรแกรมควบคุม UART ของไมโครคอนโทรลเลอร์ AVR

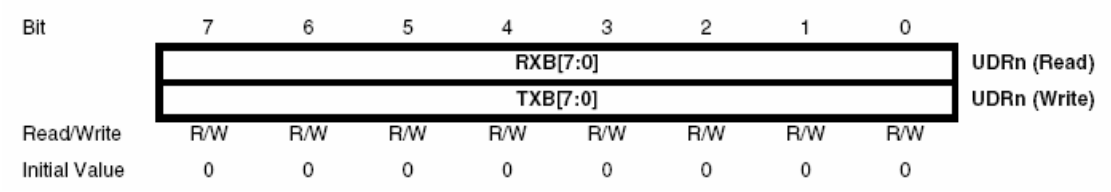
โครงสร้างการทำงานของ UART ภายใน Atmega1281 จะมีลักษณะ ดังรูป



รูปที่ 3.54 โครงสร้างการทำงานของ UART

จากรูปด้านบนจะเห็นได้ว่า UART ของ ATmega1281 จะรองรับการทำงานแบบ Synchronous เพิ่มเติมด้วยหรือเรียกว่า USART โดยจะมีขา XCK เพื่อใช้ในการ Synchronize ข้อมูล แต่ในที่นี้จะกล่าวถึงเฉพาะการทำงานแบบ Asynchronous เท่านั้น รีจิสเตอร์ที่เกี่ยวข้องกับการทำงานของ UART จะมีอยู่ 5 รีจิสเตอร์ด้วยกันคือ UCSRnA, UCSRnB, UCSRnC, UDRnและ UBRRn ซึ่งรีจิสเตอร์แต่ละตัวมีหน้าที่ต่างๆ ดังนี้

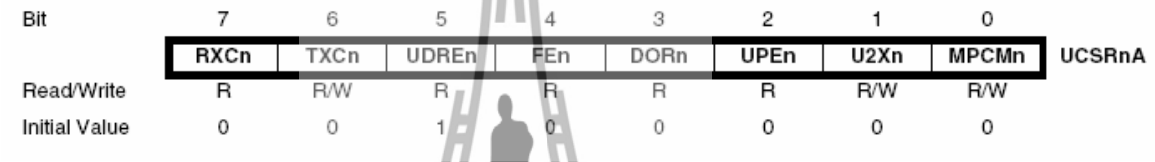
- UDRn (UART I/O Data register) เป็นรีจิสเตอร์ที่ใช้รับและส่งข้อมูล



Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

รูปที่ 3.55 UDRn Register

- **UCSRnA** (UART Control and status register a) เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานและตรวจสอบสถานะต่างๆ ของ UART ซึ่งมีรายละเอียดของบิตต่างๆ ดังนี้



Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

รูปที่ 3.56 UCSRnA Register

1. RXCn เป็นบิตบอกสถานะของการรับข้อมูล ซึ่งจะมีค่าเป็น 1 เมื่อได้รับข้อมูลเรียบร้อยแล้ว
2. TXCn เป็นบิตบอกสถานะของการส่งข้อมูล จะมีค่าเป็น 1 เมื่อส่งข้อมูลเสร็จเรียบร้อยแล้ว ซึ่งสามารถเคลียร์ค่าให้เป็นลอจิก 0 ได้โดยการเขียน ลอจิก 1 ไปที่บิตนี้
3. UDREn เป็นบิตบอกสถานะของรีจิสเตอร์ UDRn ว่าพร้อมที่จะรับข้อมูลใหม่เพื่อไปส่งได้หรือไม่ ค่าบิตจะเป็นลอจิก 1 เมื่อพร้อมที่จะส่งข้อมูลได้
4. FEn เป็นบิตบอกสถานะของการรับข้อมูลที่ผิดพลาดซึ่งเกิดจาก Stop bit มีค่าเป็นลอจิก 0
5. DORn เป็นบิตบอกสถานะของการส่งการได้รับข้อมูลมาใหม่โดยที่ยังไม่ได้อ่านข้อมูลเก่าออกไปจากรีจิสเตอร์ UDRn
6. UPEn เป็นบิตบอกความผิดพลาดเมื่อตรวจสอบพาริตี้แล้วไม่ถูกต้อง
7. U2Xn เป็นบิตที่ใช้ในการความคุมความเร็วในการสื่อสารให้เพิ่มขึ้น 2 เท่าถ้ามีการกำหนดค่าเป็นลอจิก 1
8. MPCMn เป็นบิตที่ใช้ในการกำหนดให้เป็นโหมดการสื่อสารแบบหลายหน่วยประมวลผล

- **UCSRnB** (UART Control and status register b) เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานและตรวจสอบสถานะต่างๆ ของ UART ซึ่งมีรายละเอียดของบิตต่างๆ ดังนี้

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

รูปที่ 3.57 UCSRnB Register

1. RXCIEn เป็นบิตที่กำหนดให้มีการเกิดอินเตอร์รัพท์ เมื่อได้รับข้อมูลเรียบร้อยแล้ว
2. TXCIEn เป็นบิตที่กำหนดให้มีการเกิดอินเตอร์รัพท์ เมื่อส่งข้อมูลเสร็จเรียบร้อยแล้ว
3. UDRIEn เป็นบิตที่กำหนดให้มีการเกิดอินเตอร์รัพท์ เมื่อรีจิสเตอร์ UDRn พร้อมส่งข้อมูล
4. RXENn เป็นบิตที่กำหนดให้ UART สามารถรับข้อมูลได้
5. TXENn เป็นบิตที่กำหนดให้ UART สามารถส่งข้อมูลได้
6. UCSZn2 เป็นบิตที่ใช้กำหนดจำนวนข้อมูลที่จะสื่อสารใน 1 ครั้ง
7. RXB8n เป็นบิตข้อมูลที่ได้รับเพิ่มเติมในกรณีที่ขนาดข้อมูลเกิน 8 บิต
8. TXB8n เป็นบิตข้อมูลที่ใช้ส่งเพิ่มเติมในกรณีที่ขนาดข้อมูลเกิน 8 บิต

- **UCSRnC** (UART Control and status register C) เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานและตรวจสอบสถานะต่างๆ ของ UART ซึ่งมีรายละเอียดของบิตต่างๆ ดังนี้

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

รูปที่ 3.58 UCSRnC Register

1. UMSELn1, UMSELn0 เป็นบิตในการเลือกโหมดการสื่อสารของ UART ซึ่งมีทั้งหมด 4 โหมดตามค่าต่างๆ ดังนี้

ตาราง การกำหนดโหมดในการสื่อสาร

UMSELn1	UMSELn0	โหมด
0	0	Asynchronous
0	1	Synchronous
1	0	Reserved
1	1	Master SPI

รูปที่ 3.59 การกำหนดโหมดในการสื่อสาร

2. UPMn1, UPMn0 เป็นบิตในการเลือกรูปแบบพาริตี้เพื่อตรวจสอบข้อมูล ซึ่งจะมีค่าต่างๆ ดังนี้

ตาราง การกำหนดพาริตี้

UPMn1	UPMn0	โหมด
0	0	Disable
0	1	Reserved
1	0	Even parity
1	1	Odd parity

รูปที่ 3.60 การกำหนดพาริตี้

3. USBSn เป็นบิตในการกำหนดจำนวน Stop bit ถ้าเป็นลอจิก 0 จะมี 1 บิต ถ้าเป็นลอจิก 1 จะมี 2 บิต
4. UCSZn1, UCSZn0 รวมถึง UCSZn2 ที่อยู่ในรีจิสเตอร์ UCSRnB จะทำหน้าที่ในการกำหนดจำนวนบิตของข้อมูลที่สื่อสารแต่ละครั้งดังตาราง

ตารางการกำหนดขนาดข้อมูล

UCSZn2	UCSZn1	UCSZn0	ขนาดข้อมูล
0	0	0	5 บิต
0	0	1	6 บิต
0	1	0	7 บิต
0	1	1	8 บิต
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9 บิต

รูปที่ 3.61 การกำหนดขนาดข้อมูล

5. UCPOLn เป็นบิตที่ใช้กำหนดความสัมพันธ์ระหว่างข้อมูลและสัญญาณนาฬิกา ซึ่งใช้ในโหมด Synchronous เท่านั้น

- **UBRRnL, UBRRnH** (UART Baud rate register) เป็นรีจิสเตอร์ที่ใช้กำหนดความเร็วในการสื่อสารซึ่งค่าที่กำหนดต้องมีความสัมพันธ์กับสัญญาณนาฬิกาที่ป้อนให้ชิปทำงาน

ตารางการคำนวณค่า UBRR จาก Baud Rate

โหมด	สูตรคำนวณ
Asynchronous (U2Xn = 0)	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous (U2Xn = 1)	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$

รูปที่ 3.62 การคำนวณค่า UBRR จาก Baud rate

3.4 รายการอุปกรณ์ของเครื่องวัดอุณหภูมิเอนกประสงค์

ตัวต้านทาน ขนาด ¼ วัตต์ บวกลบ 5%

1. 10 โอห์ม 1 ตัว
2. 10 โอห์ม (1206) 1 ตัว
3. 220 โอห์ม (1206) 1 ตัว
4. 470 โอห์ม 1 ตัว
5. 680 โอห์ม 1 ตัว
6. 1 กิโลโอห์ม (1206) 1 ตัว
7. 4.7 กิโลโอห์ม 2 ตัว
8. 10 กิโลโอห์ม (1206) 4 ตัว
9. 10 กิโลโอห์ม 1 ตัว
10. 10 กิโลโอห์มแบบเกือกม้าปรับค่าได้ 1 ตัว

ตัวเก็บประจุ

1. 220 ไมโครฟารัด 16 โวลต์ 6 ตัว
2. 0.47 ไมโครฟารัด 4 ตัว
3. 10 พิโคฟารัด 2 ตัว
4. 22 พิโคฟารัด 2 ตัว

อุปกรณ์สารกึ่งตัวนำ

1. ATMEGA64 หรือ ATMEGA128 1 ตัว
2. DS1820 1 ตัว
3. DS1307 1 ตัว
4. 74LVC245 1 ตัว
5. LM1117T33 1 ตัว
6. MAX232 1 ตัว
7. MMBT3904 1 ตัว
8. B772 1 ตัว
9. LED สีแดง,สีเขียว 2 ตัว

อื่นๆ

1. กราฟิกแอลซีดีขนาด 128x64 1 ตัว
2. คริสตอล 16 MHz 1 ตัว
3. คริสตอล 32.768 KHz 1 ตัว
4. คอนเน็กเตอร์ 40 ขา 1 ตัว
5. คอนเน็กเตอร์ 20 ขา 1 ตัว
6. คอนเน็กเตอร์ขางอ 4 ขา 1 ตัว
7. คอนเน็กเตอร์ขางอ 2 ขา 1 ตัว
8. คีย์บอร์ดขนาด 4x3 1 ตัว
9. สวิตช์กดติดปล่อยดับ 1 ตัว
10. บัสเซอร์ 5 โวลต์ 1 ตัว
11. แบตตารี 3 โวลต์พร้อมรังถ่าน 1 ชุด

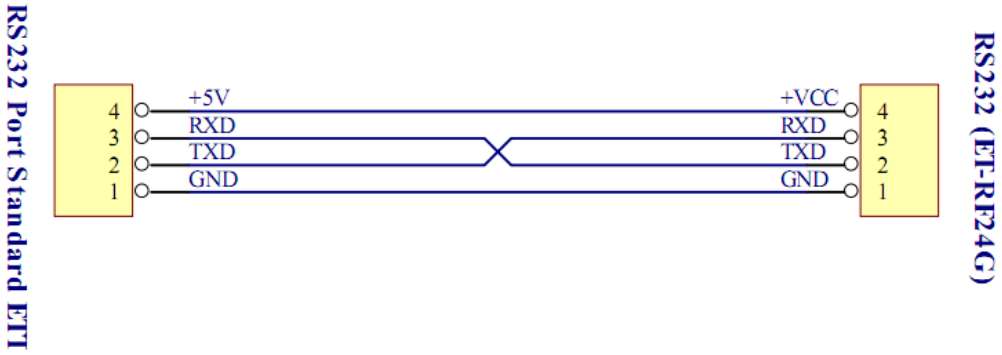
รายการอุปกรณ์อื่นๆ ที่เกี่ยวข้อง

1. ชุด รับ-ส่ง ข้อมูล RS232 แบบไร้สาย 1 ชุด
2. แบตตารี 6 โวลต์ 1 แอมป์ 1 ก้อน

3.5 ชุดรับ-ส่ง ข้อมูล RS232 แบบไร้สาย

รูปที่ 3.63 ชุด รับ-ส่ง ข้อมูล RS232 แบบไร้สาย รุ่น ET-RF24G V1.0

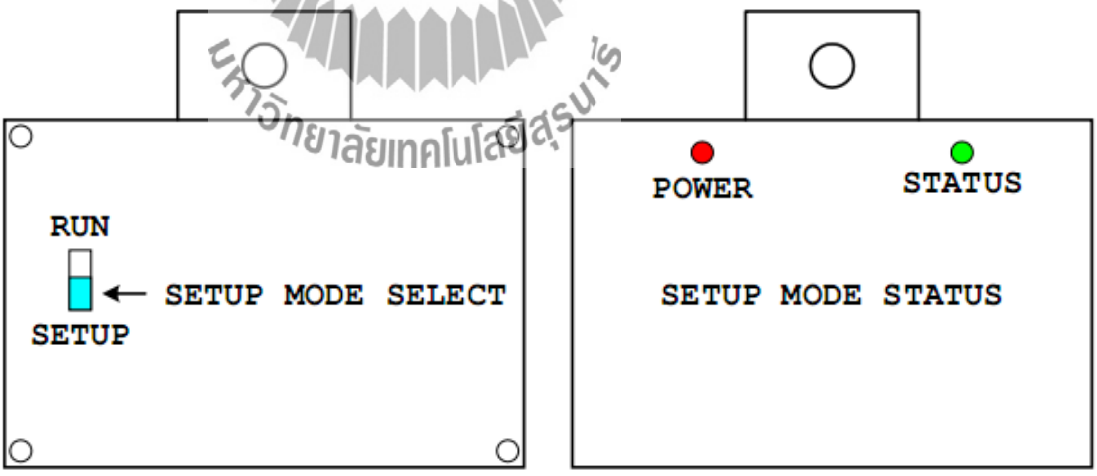
ชุด รับ-ส่ง ข้อมูล รุ่น ET-RF24G V1.0 เป็นชุด Signal converter สำหรับใช้แปลงสัญญาณระหว่าง RS232 และ RF-Wireless โดยในโหมดการทำงานของการส่งข้อมูล (Transmitter) จะทำหน้าที่รอรับข้อมูลจากพอร์ตสื่อสารอนุกรม RS232 จากขา RX แล้วแปลงสัญญาณความถี่ (GFSK :) ส่งออกไปในอากาศ และในโหมดรับ (Receiver) ชุด ET-RF24G V1.0 จะทำหน้าที่คอยตรวจจับข้อมูลที่อยู่ในรูปของสัญญาณความถี่ (GFSK) จากด้าน RF เพื่อแปลงกลับเป็นข้อมูลแบบ RS232 ส่งออกไปทางขา TX ได้ด้วย



รูปที่ 3.64 การต่อสายสัญญาณ RS232 เพื่อใช้แหล่งจ่ายจากบอร์ดไมโครฯ

3.5.1 การตั้งค่าการใช้งานเครื่อง ET-RF24G V1.0

การใช้งานในโหมด Setup mode ซึ่งเป็นโหมดใช้กำหนดค่า Configuration ต่างๆ ในการตั้งค่าต่างๆ นั้นจะกระทำร่วมกับโปรแกรม "ET_RF24G_V1.EXE" เมื่อเข้าสู่โหมด Setup แล้ว จะสังเกตุเห็นหลอดไฟแสดงสถานะการทำงาน หรือ LED STATUS ติดสว่างค้างอยู่ตลอดเวลา แต่เมื่อมีการสั่งอ่านหรือเขียนข้อมูลกับบอร์ด สถานการณ์ทำงานของ LED STATUS จึงจะกระพริบตามจังหวะของการส่งข้อมูล แต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะติดค้างอยู่ตลอดเวลา

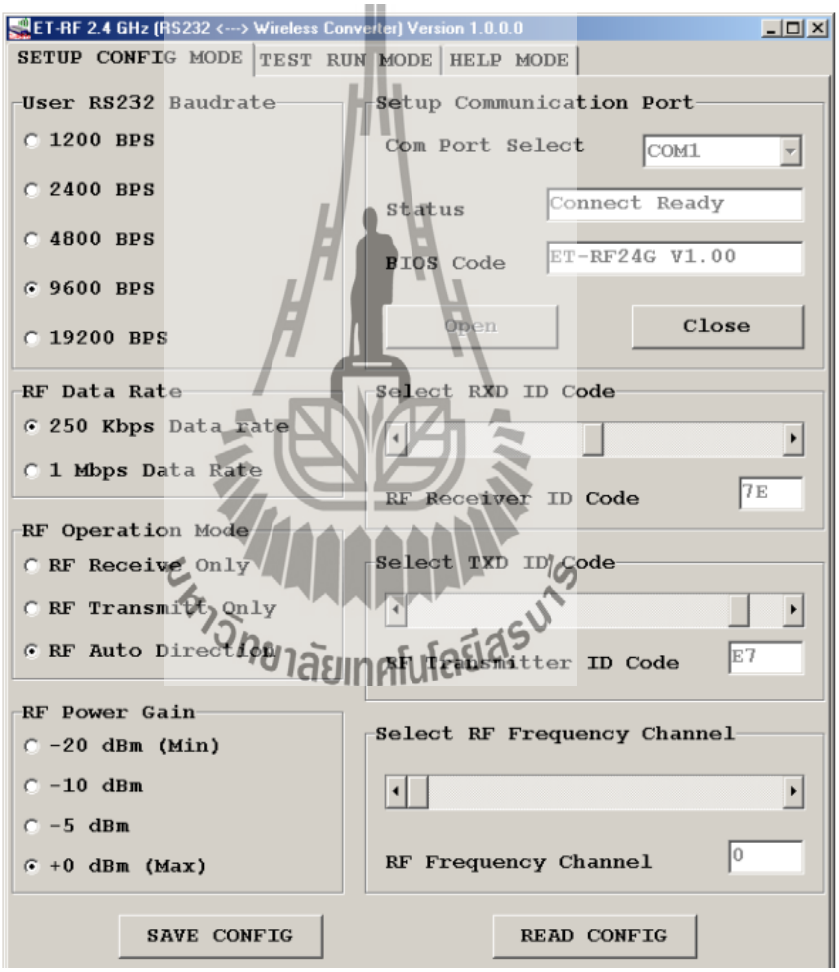


รูปที่ 3.65การเลือกโหมดการทำงาน สำหรับกำหนดค่า Configuration (Setup mode)

การกำหนดค่านั้นจะต้องกระทำขณะที่ตัวเครื่องทำงานอยู่ใน Setup mode เท่านั้น โดยค่าต่างๆนั้นจะถูกใช้สำหรับเป็นเงื่อนไขในการทำงานของเครื่องในขณะที่อยู่ใน Run mode ดังนั้นก่อน

การเริ่มต้นใช้งานเครื่องในครั้งแรกควรจะต้องทำการกำหนดค่าของ Configuration ต่างๆ โดยเมื่อกำหนดค่าตัวเลือกต่างๆของ Configuration เรียบร้อยแล้ว ก็สามารถเปลี่ยนโหมดการทำงานของตัวเครื่องกลับเป็น Run mode พร้อมกับการปิดไฟที่จ่ายให้กับตัวเครื่องชั่วขณะหนึ่ง จากนั้นจึงเริ่มต้นจ่ายไฟให้กับตัวเครื่องใหม่ ก็จะสามารถใช้งานได้ตามค่า Configuration ที่กำหนดไว้แล้วได้ทันที

3.5.2 คุณสมบัติของ Configuration



รูปที่ 3.66 รูปแบบโปรแกรมที่ใช้สำหรับกำหนดค่า Configuration

User RS232 Baudrate ใช้สำหรับกำหนดค่าความเร็วในการรับส่งข้อมูลทางด้าน RS232 ของตัวเครื่องในขณะที่ทำงานอยู่ใน Run mode ซึ่งสามารถกำหนดได้ 5 ค่าคือ

1. 1200 BPS
2. 2400 BPS
3. 4800 BPS
4. 9600 BPS
5. 19200 BPS

RF Data rate ใช้สำหรับกำหนดความเร็วในการรับส่งข้อมูลทางด้าน RF ของเครื่อง ซึ่งจะต้องกำหนดให้เครื่องทุกๆ ตัว ที่จะนำมาใช้ติดต่อสื่อสารกัน มีค่าอัตราความเร็วในการรับส่งข้อมูลด้าน RF หรือ RF Data Rate นี้มีค่าเท่ากันหมด ถ้ากำหนดค่าความเร็วต่างกันจะไม่สามารถส่งข้อมูลกันได้ โดยค่า RF Data rate สามารถกำหนดได้ 2 ค่าคือ

1. 250 Kbps
2. 1 Mbps

RF Operation mode ใช้สำหรับกำหนดโหมดของเครื่อง ซึ่งสามารถกำหนดหน้าที่การทำงานได้ 3 แบบ คือ

1. **RF Receive only** เป็นการกำหนดให้เครื่องทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RF เพื่อเปลี่ยนข้อมูลแบบ RS232 และส่งออกไปทางด้านขา TX ของ RS232 ตลอดเวลา
2. **RF Transmit only** เป็นการกำหนดให้เครื่องทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RS232 จากขา RX เพื่อเปลี่ยนเป็นข้อมูลแบบ GFSK และส่งออกไปทางด้าน RF ตลอดเวลา
3. **RF Auto direction** เป็นการกำหนดโหมดการทำงานแบบ Half duplex 2 ทิศทาง ซึ่งสามารถสลับโหมดการทำงานระหว่างการรับและส่งข้อมูลได้เองโดยอัตโนมัติ

RF Power gain เป็นการกำหนดกำลังส่งของวงจร RF Power ที่ใช้ในการส่งข้อมูล โดยค่า +0dBm เป็นค่ากำลังส่งสูงสุด ส่วน -20dBm เป็นค่ากำลังส่งต่ำสุด โดยสามารถกำหนดได้ 4 ระดับคือ

1. -20dBm (กำลังส่งต่ำสุด)
2. -10dBm
3. -5dBm
4. +0dBm (กำลังส่งสูงสุด)

RXD ID Code เป็นรหัส ID Code ของเครื่องในโหมดของการรับส่งข้อมูลจาก RF โดยเมื่อเครื่องด้านส่งจะทำการส่งข้อมูลออกไปทาง RF นั้นจะมีการระบุหมายเลข ID Code ของด้านรับรวมไปกับชุดข้อมูลด้วยเสมอ ส่วนทางด้านรับเมื่อรับข้อมูลจากด้าน RF ได้จะทำการเปรียบเทียบรหัส ID Code ที่รวมมากับข้อมูลว่าตรงกับรหัสของ RXD ID Code ที่กำหนดไว้ ถ้าถูกต้องจะทำการแยกเฉพาะส่วนของข้อมูลที่รับได้เพื่อเปลี่ยนเป็นข้อมูลแบบ RS232 แล้วส่งออกไปทางด้าน TX ของ RS232 โดยค่า RXD ID Code นั้นสามารถกำหนดได้ 256 ค่าในรูปแบบของเลขฐานสิบหก (00H-FFH)

TXD ID Code เป็นรหัส ID Code ปลายทางที่ส่งข้อมูลไปหา โดยที่เครื่องที่ถูกกำหนดให้เป็นฝ่ายส่งข้อมูลนั้น จะนำข้อมูลที่รับได้จาก RS232 ไปเข้ารหัสรวมกับ TXD ID Code แล้วส่งออกทางด้าน RF โดยค่า TXD ID Code นั้นสามารถกำหนดได้ 256 ค่าในรูปแบบของเลขฐานสิบหก (00H-FFH)

RF Frequency channel เป็นการกำหนดค่าของช่องความถี่ที่จะใช้ในการรับส่งข้อมูลกัน โดยสามารถเลือกได้ทั้งหมด 125 ช่อง (0-124) โดยทั้งฝ่ายรับและฝ่ายส่งต้องเลือกช่องความถี่เดียวกัน ถึงจะสามารถติดต่อกันได้

3.6 การชาร์ตแบตตารี่

จากการทดลองได้ทำการชาร์ตไฟที่แรงดันต่างกัน เวลาที่ใช้จะต่างกัน ดังแสดงในตาราง 3.1

ตารางที่ 3.1 แสดงเวลาในการชาร์ตแบตตารี่ที่ใช้แรงดันต่างกัน

แรงดันที่ใช้ (V)	กระแสที่ใช้ (A)	เวลาในการชาร์ต (ชั่วโมง)
8	1	4
10	1	3

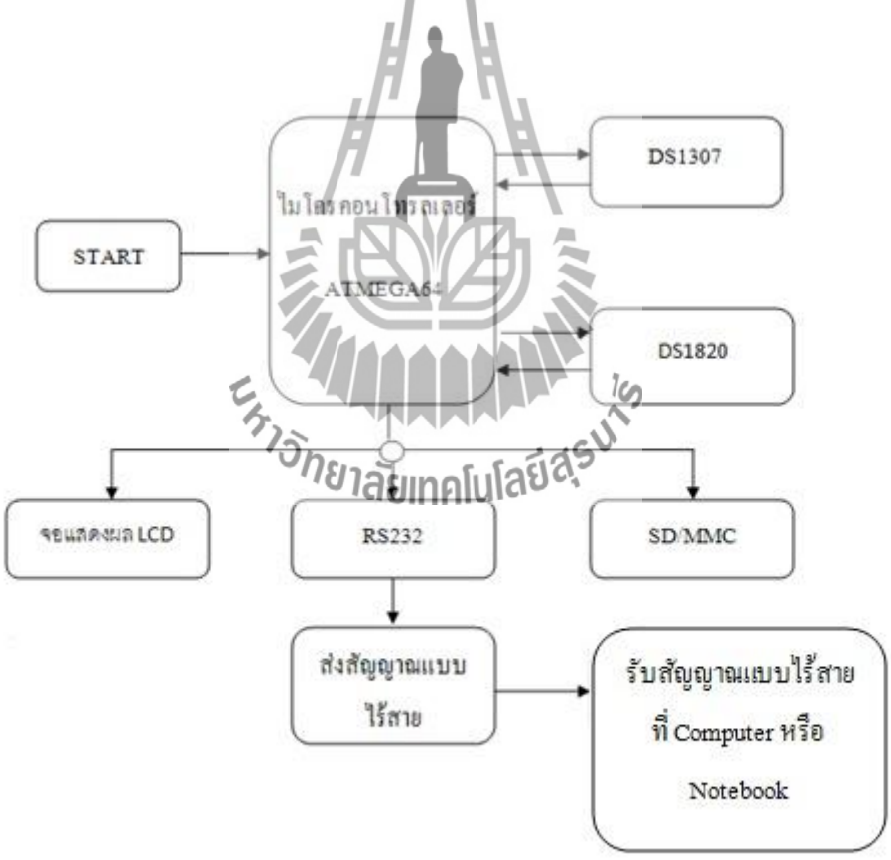
บทที่ 3
หลักการทำงานและการออกแบบ

3.1 บทนำ

ในบทนี้จะกล่าวถึงการทำงาน การใช้โปรแกรม Winavr การใช้คำสั่งในการติดต่อบอร์ดเบื้องต้น การแสดงผลผ่าน LCD (Liquid crystal display) การสื่อสารแบบอนุกรมผ่านวงจร UART (Universal asynchronous receiver transmitter) การสื่อสารแบบไร้สาย และเวลาการชาร์ตแบตเตอรี่

3.2 หลักการทำงานของเครื่องวัดอุณหภูมิอเนกประสงค์

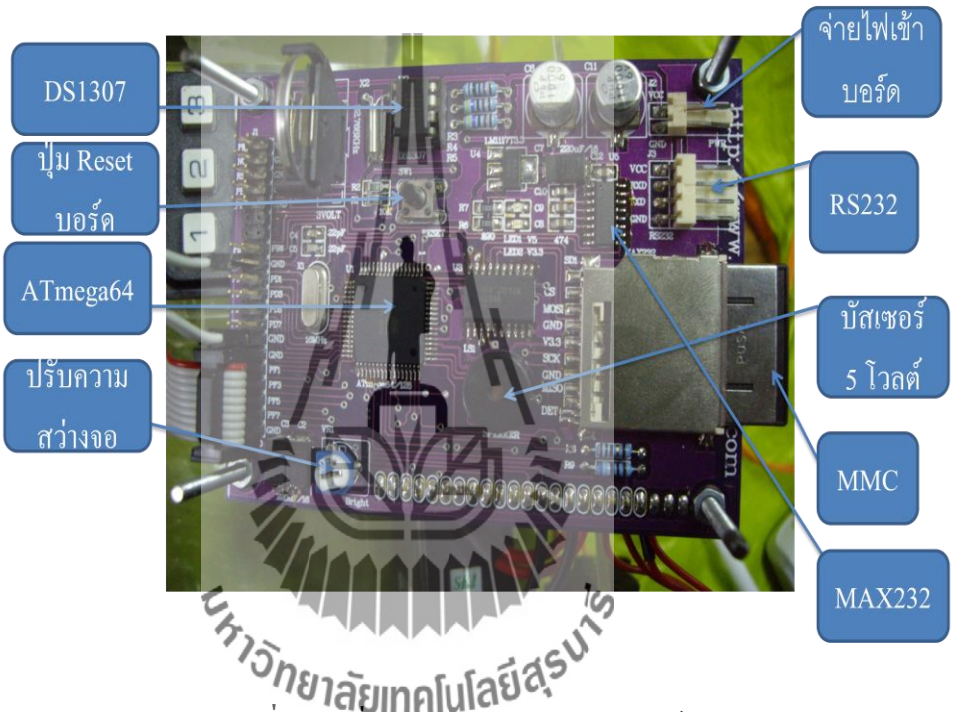
3.2.1 ระบบของเครื่องวัดอุณหภูมิเอนกประสงค์



รูปที่ 3.1 แผนภาพเครื่องวัดอุณหภูมิเอนกประสงค์

ระบบของเครื่องวัดอุณหภูมิเอนกประสงค์ แยกเป็นส่วนหลักๆ ดังนี้คือ ส่วนของไมโครคอนโทรลเลอร์เบอร์ ATMEGA64 ส่วนของไอซีเบอร์ DS1820 และ DS1307 ส่วนของการแสดงผล ที่แสดงที่จอแสดงผล LCD แสดงผลที่คอมพิวเตอร์โดยผ่านการสื่อสารแบบไร้สาย และส่วนของการบันทึกข้อมูลลง SD/MMC ดังได้แสดงของระบบในรูปที่ 3.1

3.2.2 การทำงานของวงจร

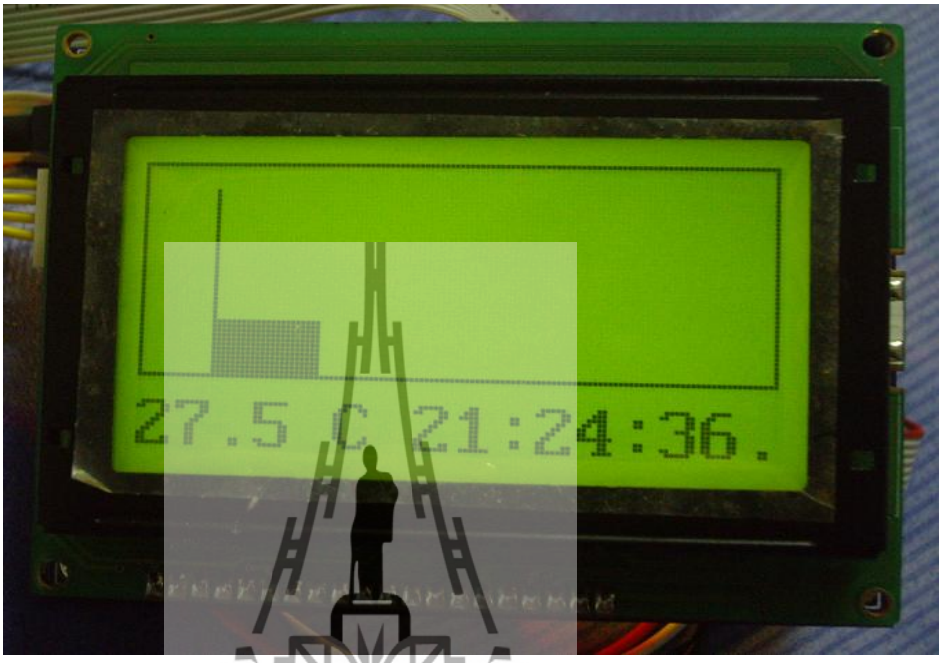


รูปที่ 3.2 เครื่องวัดอุณหภูมิเอนกประสงค์

เริ่มต้นจากการป้อนไฟกระแสตรงขนาด 5 โวลต์ผ่าน CON2 จากนั้นจะมีตัวเก็บประจุ C2, C3, C6 และ C11 ทำหน้าที่กรองแรงดันให้เรียบมากขึ้น โดยสถานะของแรงดันจะถูกแสดงด้วย LED1

หัวใจหลักของโครงงานนี้เป็นไมโครคอนโทรลเลอร์ตระกูล AVR สามารถเลือกใช้ได้ 2 เบอร์ คือ ATMEGA64 และ ATMEGA128 เนื่องจากโครงสร้างภายนอกของทั้งสองเบอร์เหมือนกัน ซึ่งจะต่างกันก็เพียงคุณสมบัติของหน่วยความจำ Flash และหน่วยความจำ EEprom ภายในเท่านั้น กล่าวคือ ATMEGA64 มีหน่วยความจำ Flash ขนาด 64 กิโลไบต์และหน่วยความจำ EEprom ขนาด 2 กิโลไบต์ ในส่วนเบอร์ ATMEGA128 มีหน่วยความจำ Flash ขนาด 128 กิโลไบต์และหน่วยความจำ EEprom ขนาด 4 กิโลไบต์แต่หน่วยความจำ SRAM ทั้งสองเบอร์มีขนาดเท่ากันคือ 4

กิโลไบต์ ซึ่งหน่วยความจำดังกล่าวจะใช้สำหรับการประมวลผลและควบคุมการทำงานในส่วนต่างๆ อาทิเช่น การแสดงผลบนหน้าจอแอลซีดีและนำข้อมูลไปเก็บไว้ที่หน่วยความจำ SD/MMC เป็นต้น



รูปที่ 3.3 จอกราฟิกแอลซีดีขนาด 128x64 พิกเซล

การแสดงผลในโครงงานนี้จะใช้จอกราฟิกแอลซีดีขนาด 128x64 พิกเซล (แนวแกนนอน 128 พิกเซลและแนวแกนตั้ง 64 พิกเซล) โดยใช้พอร์ต A เชื่อมต่อกับ D0-D7 เพื่อส่งข้อมูลการแสดงผลและใช้พอร์ต C ในการควบคุมการทำงานของแอลซีดี และในส่วนควบคุมการเปิดปิดไฟพื้นหลัง (Black Light) จะใช้ขา PC1 ซึ่งจะเชื่อมต่อผ่าน Q2 อีกทีหนึ่ง

PC5 และ PC6 เชื่อมต่อกับขา Chip Select ใช้สำหรับเลือกคอนโทรลเลอร์ภายในของแอลซีดีเพื่อควบคุมการทำงานของแอลซีดี เนื่องจากภายในแอลซีดีชนิด 128x64 ที่ใช้ในโครงงานนี้จะมีคอนโทรลเลอร์เพื่อควบคุมการแสดงผลแนวแกนนอนอยู่ 2 ตัว โดยแต่ละตัวจะถูกแบ่งให้มีการแสดงผล 64 พิกเซล ดังนั้นเมื่อรวมทั้งสองตัวเข้าด้วยกันแกนนอนจะสามารถแสดงผลได้ 128 พิกเซล และในส่วนของการแสดงผลแนวแกนตั้งไม่จำเป็นต้องเพิ่มคอนโทรลเลอร์เนื่องจากมีขนาด 64 พิกเซลอยู่แล้ว สำหรับการรีเซ็ตการทำงานของแอลซีดีจะใช้ขา PC7

สุดท้ายในส่วนของแอลซีดี คือ การควบคุมความสว่างของพิกเซล จะใช้ตัวต้านทานปรับค่าได้ 10 กิโลโอห์ม (VR1) ต่อเข้ากับขาไฟบวก (VCC) และขา VEE (ขา 18 ของแอลซีดี) ที่มี

สภาวะเป็นแรงดันไฟลบ 9 โวลต์ ส่วนขากลางจะต่อเข้ากับขา 3 ของแอลซีดีเพื่อกำหนดแรงดันอ้างอิงและจะทำหน้าที่ควบคุมความสว่างของพิกเซลบนหน้าจอ

การเชื่อมต่อกับหน่วยความจำ SD/MMC ในส่วนนี้จะใช้แรงดันขนาด 3.3 โวลต์ ซึ่งจะแตกต่างจาก IC1 ที่มีระดับแรงดันอยู่ที่ 5 โวลต์ ด้วยเหตุนี้การเชื่อมต่อระหว่าง IC1 กับการ์ด SD/MMC จึงไม่สามารถทำได้โดยตรงเพราะจะทำให้เกิดความเสียหายกับ SD/MMC ได้ การแก้ไขจุดนี้สามารถทำได้โดยใช้ IC3 (74LVC245) มาทำหน้าที่เป็นบัฟเฟอร์เพื่อลดทอนแรงดันจาก 5 โวลต์ให้เหลือ 3.3 โวลต์เสียก่อน ในส่วนของภาคจ่ายไฟ 3.3 โวลต์จะใช้ IC4 เบอร์ LM1117T3.3 เพื่อทำหน้าที่ลดทอนแรงดันจาก 5 โวลต์ให้เหลือขนาด 3.3 โวลต์และจะมี LED2 แสดงสถานะของแรงดัน

ขาที่จำเป็นสำหรับการเชื่อมต่อระหว่าง IC1 กับการ์ด SD/MMC มีดังนี้คือ

- ขา PB0 (MMC_CS) เป็นขา Chip Select ใช้เมื่อต้องการติดต่อกับ SD/MMC
- ขา PB1 (MMC_SCK) เป็นขาสัญญาณนาฬิกาในโหมด SP1 สำหรับ SD/MMC
- ขา PB2 (MMC_MOSI) เป็นขาเพื่อส่งข้อมูลไปยัง SD/MMC
- ขา PB3 (MISO) เป็นขาเพื่อส่งข้อมูลไปยัง SD/MMC
- ขา PB4 (MMC_DETECT) เป็นขาเพื่อคอยตรวจสอบสถานะของสวิตช์ขนาดเล็กที่อยู่ภายในซ็อกเก็ต SD/MMC

DS1307 เป็นไอซีสร้างฐานเวลาจริง (Real Time Clock) มีมาตรฐานการเชื่อมต่อแบบ I2Cในส่วนของ IC1 จะใช้ขา PD0 ที่ภายในมีคุณสมบัติเป็น SCL ใช้สำหรับเป็นสัญญาณนาฬิกา และขา PD1 ที่มีคุณสมบัติ SDA เป็นขาข้อมูล สืบเนื่องจาก DS1307 สามารถกำหนดให้สร้างสัญญาณความถี่ 1 เฮิร์ตซ์ (Hz) ออกมาจากขาที่ 7 (SQ) ได้ ดังนั้นผู้เขียนจึงนำคุณสมบัตินี้มาใช้ประโยชน์เพื่อเป็นสัญญาณการอินเตอร์รัปต์โดยต่อเข้ากับขา PE4 (INT4) ซึ่งเมื่อเกิดการอินเตอร์รัปต์ขึ้น IC1 จะทำการอ่านข้อมูลจาก DS1307 ไปแสดงผลบนหน้าจอแอลซีดี

การเชื่อมต่อกับระหว่างบอร์ดกับคอมพิวเตอร์จะใช้ IC5 (MAX232) ทำหน้าที่เปลี่ยนแรงดันแบบ TTL เป็น RS-232 เพื่อป้อนให้กับพอร์ตสื่อสารของเครื่องคอมพิวเตอร์ สำหรับคอนเน็กเตอร์สำหรับการเชื่อมต่อเข้ากับคอมพิวเตอร์จะมีจำนวน 4 ขา (VCC, RxD, TxD, และ GND) ซึ่งการเชื่อมต่อผู้ใช้จะต้องเรียกตำแหน่งขาให้ถูกต้องการเชื่อมต่อจึงจะสมบูรณ์ หรือจะให้ง่ายกว่านั้นผู้ใช้สามารถนำสายที่เป็นมาตราฐานของบริษัท ETT มาใช้ได้เลย

คอนเน็กเตอร์ CON1 ขนาด 40 ขาเชื่อมต่อกับขาพอร์ตของ IC1 ซึ่งขา 1-10 จะเตรียมไว้สำหรับการโปรกแรมแบบ ISP โดยขา 32 ถึง 40 จะเชื่อมต่อกับพอร์ต F ในการใช้งานจะถูกนำไปเชื่อมต่อเข้ากับคีย์บอร์ดขนาด 4x3 สำหรับการตังค่าการใช้งานต่าง นอกจากนั้นแล้วขา CON1 ที่

เหลือจะถูกเชื่อมต่อกับพอร์ต PB5, PB6, PB7 และพอร์ต D ซึ่งขาต่างๆดังกล่าวผู้ใช้สามารถนำไปเชื่อมต่อกับอุปกรณ์ภายนอกเพื่อสร้างแอพพลิเคชั่นอื่นๆได้ตามต้องการ

DS1820 เป็นไอซีตรวจสอบอุณหภูมิ โดยที่ขาสัญญาณ DQ จะเชื่อมต่อเข้ากับขา PB7 และขาสัญญาณจะต้องต่อตัวต้านทาน 4.7 กิโลโอห์มพลูอัพ (pull-up) ด้วย ข้อควรระวังในการใช้งาน DS1820 ไม่ควรนำไปวัดในสภาวะที่อุณหภูมิสูงถึง 100 องศาเซลเซียส ถึงแม้ว่าคุณสมบัติจะสามารถวัดอุณภูมิสูงถึง 125 องศาเซลเซียส เนื่องจาก DS1820 อาจจะเกิดความเสียหายได้จากการวัดอุณหภูมิที่มีความร้อนสูงแบบต่อเนื่อง

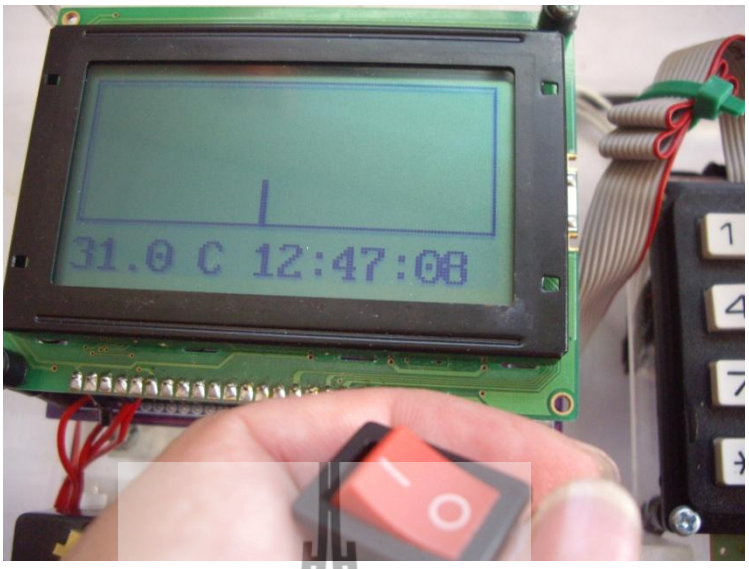
สุดท้ายเป็นวงจรควบคุมการทำงานของบัซเซอร์ เนื่องจากกระแสที่ขาเอาต์พุตของ IC1 ไม่เพียงพอต่อการขับบัซเซอร์เปล่งเสียงได้ ดังนั้นจะต้องทำการขยายกระแสโดยใช้ทรานซิสเตอร์ Q1 เบอร์ MMBT3904

รายละเอียดดูจากภาคผนวก รูปที่ 2.38

3.2.3 การใช้งานเครื่องวัดอุณหภูมิเอนกประสงค์

1. ทำการเปิดสวิตช์เพื่อทำการจ่ายไฟให้กับวงจร

รูปที่ 3.4 ก่อนเปิดเครื่องวัดอุณหภูมิเอนกประสงค์

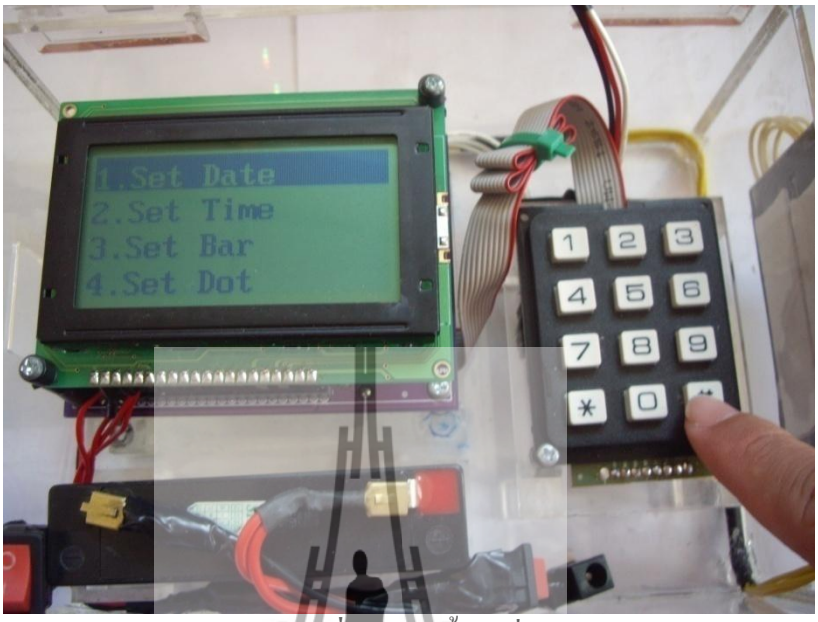


รูปที่ 3.5 เมื่อทำการกดสวิตช์เปิดเครื่องวัดอุณหภูมิเอนกประสงค์

2. เมื่อใช้ครั้งแรกให้ทำการตั้งวันที่ และเวลา เพื่อให้ตรงกับปัจจุบัน โดยทำการกดเลข 0 บนคีย์บอร์ด เครื่องจะถามรหัสผ่าน ให้ทำการกรอกรหัส 1449 แล้วกด # เพื่อตกลง

รูปที่ 3.6 การใส่รหัสผ่าน

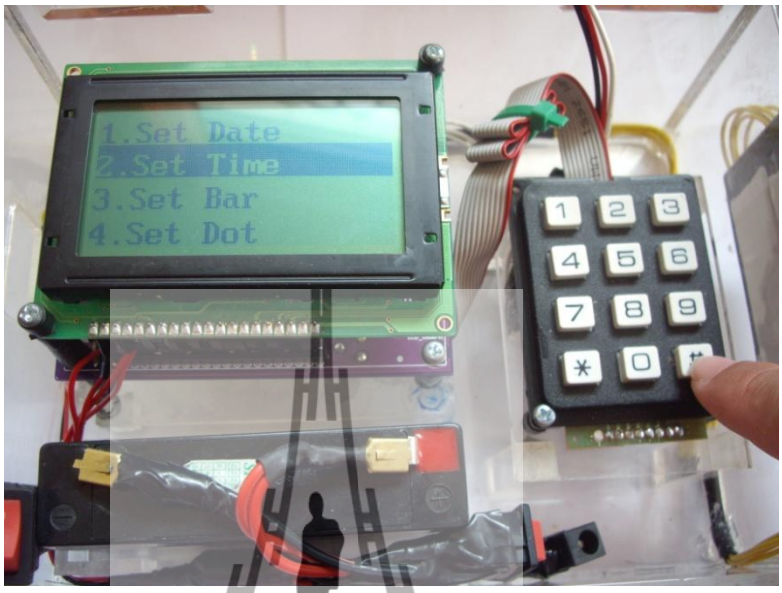
2.1 เลือกอันแลกเพื่อทำการตั้งค่าวันที่ เมื่อใส่วันที่แล้วให้ตกลงโดยกดปุ่ม #



รูปที่ 3.7 การตั้งวันที่

รูปที่ 3.8 การตั้งวันที่ (ต่อ)

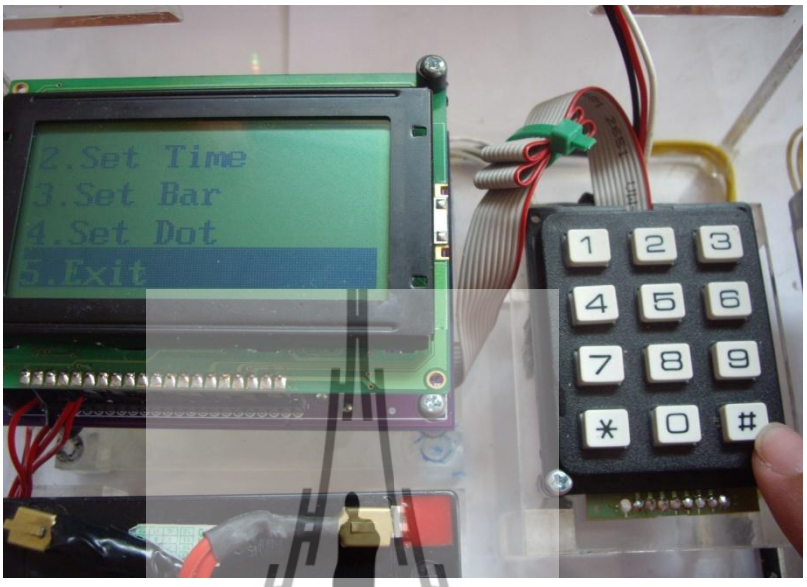
2.2 เลือกอันที่สองเพื่อทำการตั้งเวลา โดยเวลาแสดงเป็นแบบ 24 ชั่วโมง เมื่อใส่เวลาเสร็จแล้วให้ตกลงโดยกดปุ่ม #



รูปที่ 3.9 การตั้งเวลา

รูปที่ 3.10 การตั้งเวลา (ต่อ)

3. เมื่อทำการตั้งค่าเสร็จแล้วให้เลือก Exit โดยกดเลข 7 และ 4 เพื่อทำการขึ้นลง เมื่อเลือกแล้วทำการตกลง โดยกดปุ่ม #



รูปที่ 3.11 ออกจากการตั้งค่า

4. เมื่อไม่ต้องการที่จะดูหน้าจอให้ทำการปิดสวิตช์ เพื่อเป็นการประหยัดแบตตารี

รูปที่ 3.12 ปิดสวิตช์จอ LCD

3.2.4 การเชื่อมต่อการสื่อสารแบบไร้สาย

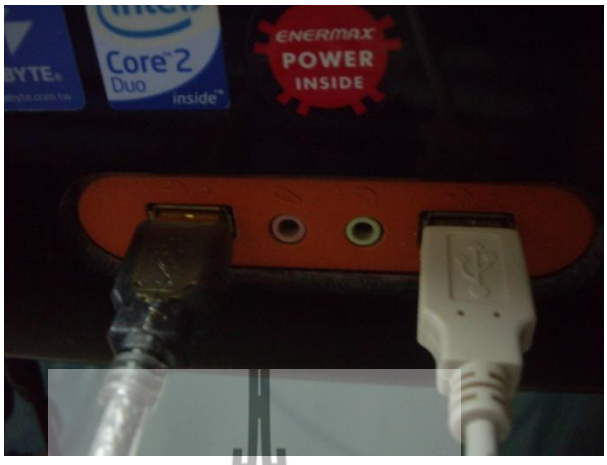
1. ทำการเสียบสาย RS232 และเสียบแหล่งจ่ายไฟโดยใช้ USB (Universal serial bus)

รูปที่ 3.13 การเสียบสาย RS232และแหล่งจ่ายไฟ

2. ทำการแปลงหัวเชื่อมต่อจาก RS232 มาเป็น USB

รูปที่ 3.14 การแปลงหัวเชื่อมต่อจาก RS232 มาเป็น USB

3. ทำการเสียบสาย USB ทั้ง 2 ที่ Computer หรือ Notebook เป็นการติดตั้งพร้อมใช้งาน



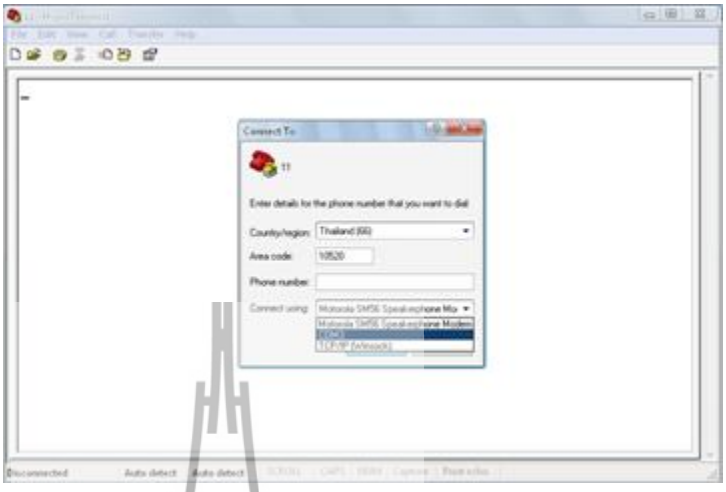
รูปที่ 3.15 การเสียบสาย USB ทั้ง 2 ที่ Computer หรือ Notebook

3.2.5 การ Capture Text

1. เปิดโปรแกรม Hyperterminal ขึ้นมาและตั้งชื่อไฟล์ในการเข้าใช้งาน

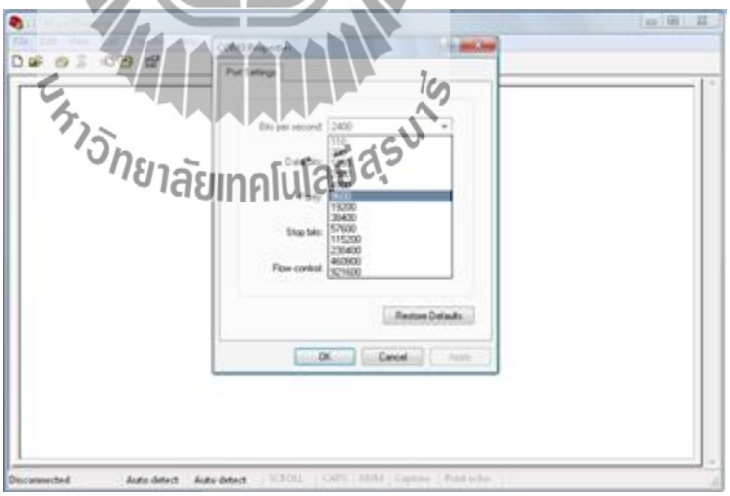
รูปที่ 3.16 การตั้งชื่อไฟล์ในการเข้าใช้งาน

2. ทำการเชื่อมต่อไปยัง Port ของโน๊ตบุ๊คที่ใช้ส่งข้อมูล



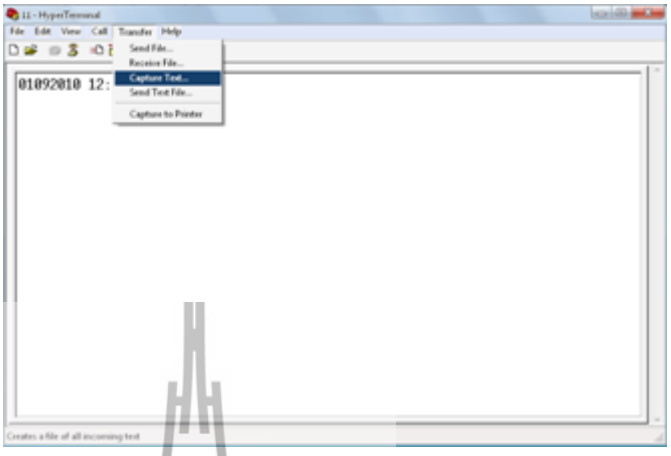
รูปที่ 3.17 การเชื่อมต่อ port ของโน๊ตบุ๊ค

3. กำหนดค่า Bit per second ที่ 9600



รูปที่ 3.18 การกำหนดค่า Bit per second

4. Hyperterminal จะแสดงค่าข้อมูล แล้วไปคลิกที่ Transfer >> Capture Text



รูปที่ 3.19 การ Capture Text

5. ตั้งชื่อไฟล์ที่จะใช้ในการบันทึก แล้วกด save

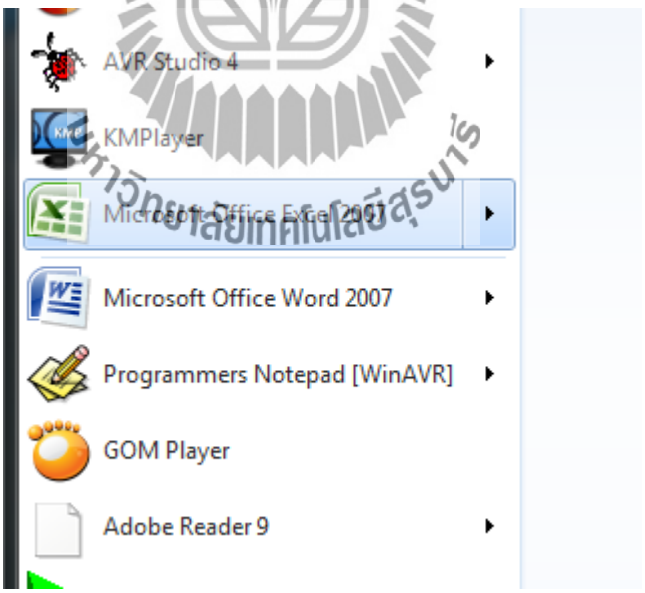
รูปที่ 3.20 การตั้งชื่อไฟล์ในการ Capture Text

6. กดปุ่ม start เพื่อเป็นการเริ่มเก็บข้อมูล

รูปที่ 3.21 การเริ่มเก็บข้อมูล

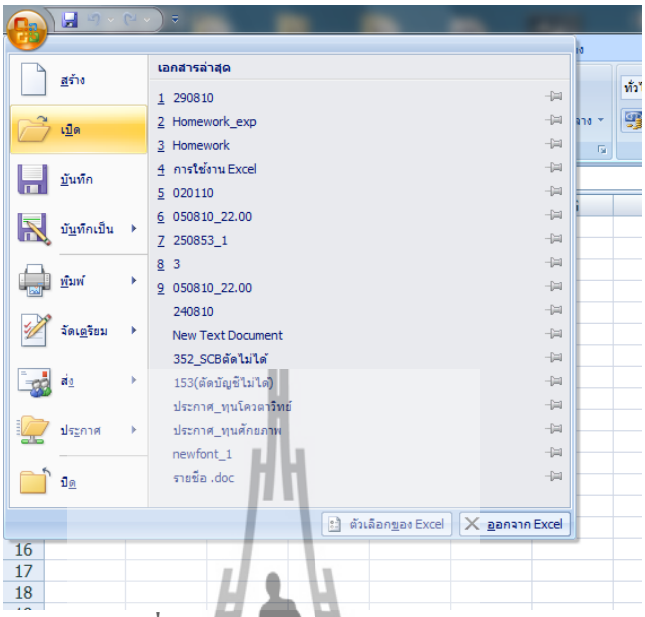
3.2.6 การนำเข้าไฟล์ .TXT โดยใช้โปรแกรม Microsoft office excel 2007

1. ทำการเปิดโปรแกรม Microsoft office excel 2007 (หรือใช้เวอร์ชั่นอื่น)



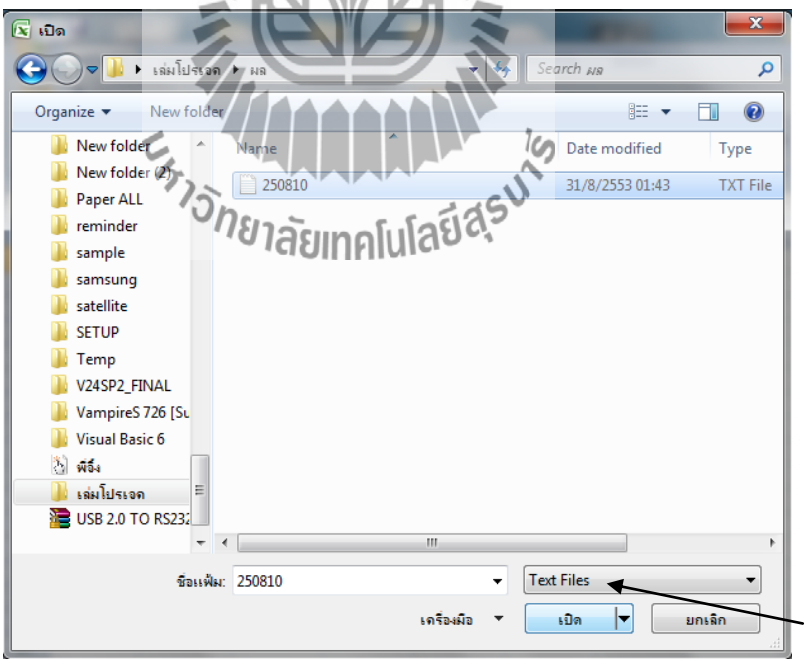
รูปที่ 3.22 เปิดโปรแกรม Microsoft office excel 2007

2. ทำการเปิดไฟล์โดยการเอาเมาส์ชี้ที่ปุ่ม Office แล้วเลือกเปิด



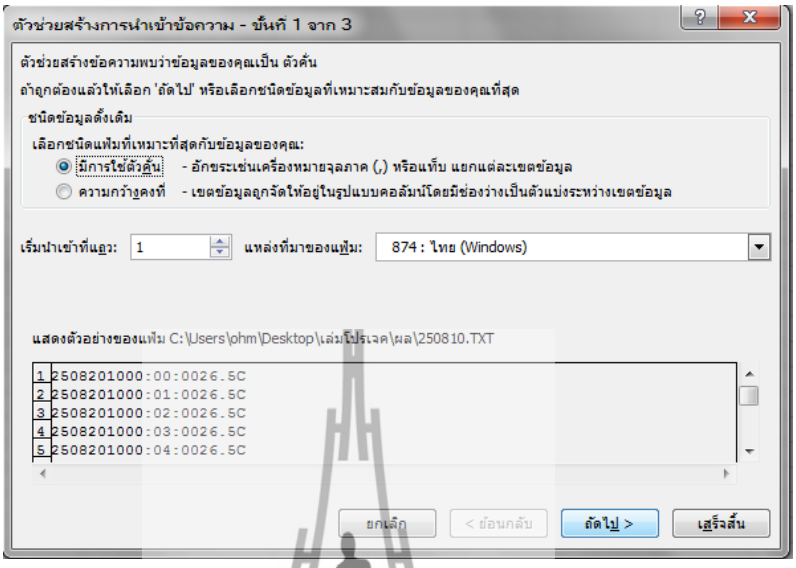
รูปที่ 3.23 การเปิดนำไฟล์ข้อมูลเข้ามา

3. เลือกที่อยู่ที่เราเก็บข้อมูลไว้และเนื่องจากข้อมูลเราเป็นไฟล์ TXT ต้องทำการเลือกนามสกุล ไฟล์เป็นไฟล์ TXT ก่อนแล้วเลือกไฟล์ข้อมูลจากนั้นคลิกเปิด



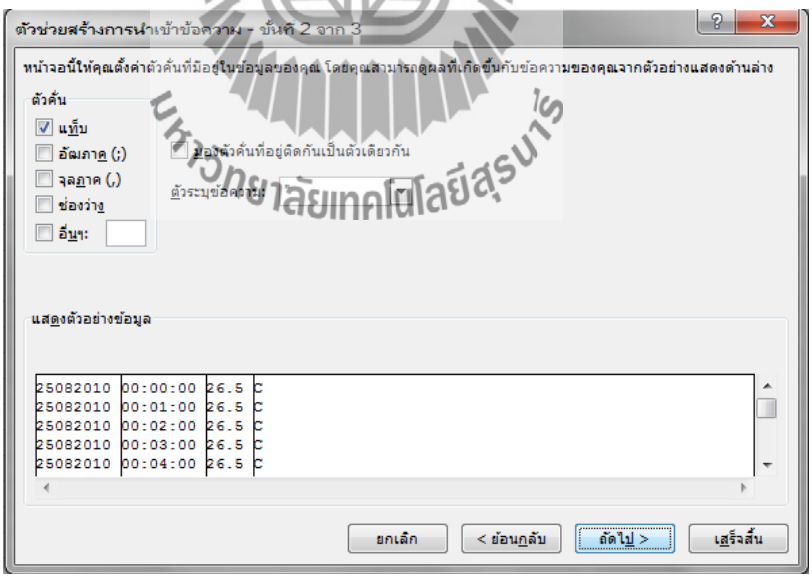
รูปที่ 3.24 การเลือกไฟล์ข้อมูล

4. ให้คลิกถัดไป



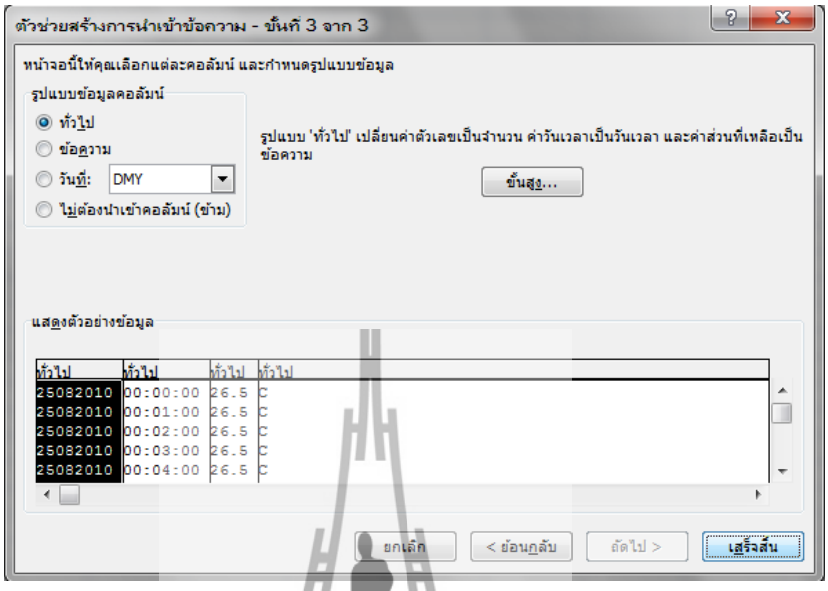
รูปที่ 3.25 หน้าต่างตัวช่วยสร้างการนำเข้าข้อความ ขั้นตอนที่ 1 จาก 3

5. คลิกถัดไป



รูปที่ 3.26 หน้าต่างตัวช่วยสร้างการนำเข้าข้อความ ขั้นที่ 2 จาก 3

6. คลิกเสร็จสิ้น



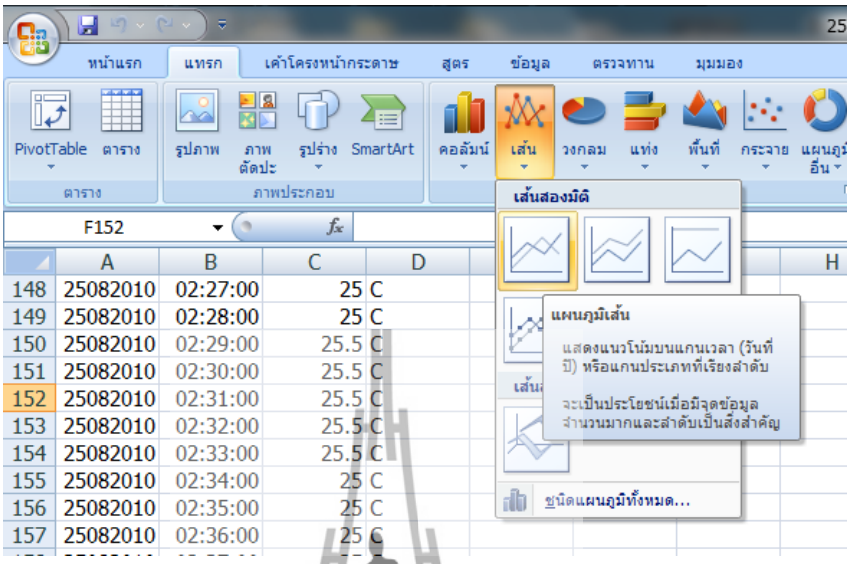
รูปที่ 3.27 หน้าต่างตัวช่วยสร้างการนำเข้าข้อความ ขั้นที่ 3จาก 3

7. จะได้ข้อมูลมา 4 แถว โดยจะมีวันที่ เวลา อุณหภูมิ และหน่วยองศาเซลเซียส

	A	B	C	D	E
148	25082010	02:27:00	25	C	
149	25082010	02:28:00	25	C	
150	25082010	02:29:00	25.5	C	
151	25082010	02:30:00	25.5	C	
152	25082010	02:31:00	25.5	C	
153	25082010	02:32:00	25.5	C	
154	25082010	02:33:00	25.5	C	
155	25082010	02:34:00	25	C	
156	25082010	02:35:00	25	C	
157	25082010	02:36:00	25	C	
158	25082010	02:37:00	25	C	
159	25082010	02:38:00	25.5	C	
160	25082010	02:39:00	25.5	C	
161	25082010	02:40:00	25.5	C	
162	25082010	02:41:00	25	C	
163	25082010	02:42:00	25	C	
164	25082010	02:43:00	25.5	C	

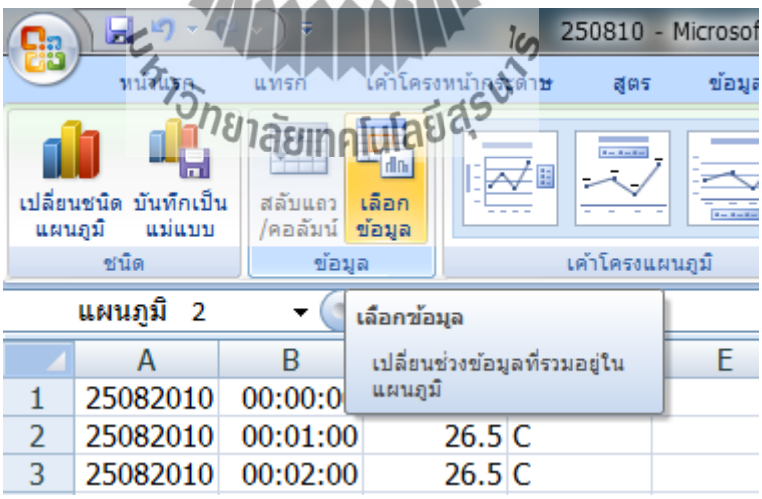
รูปที่ 3.28 ข้อมูลที่ได้จากการนำเข้าไฟล์ TXT

8. เลือกคำสั่งแทรกแล้วเลือกกราฟแบบเส้น



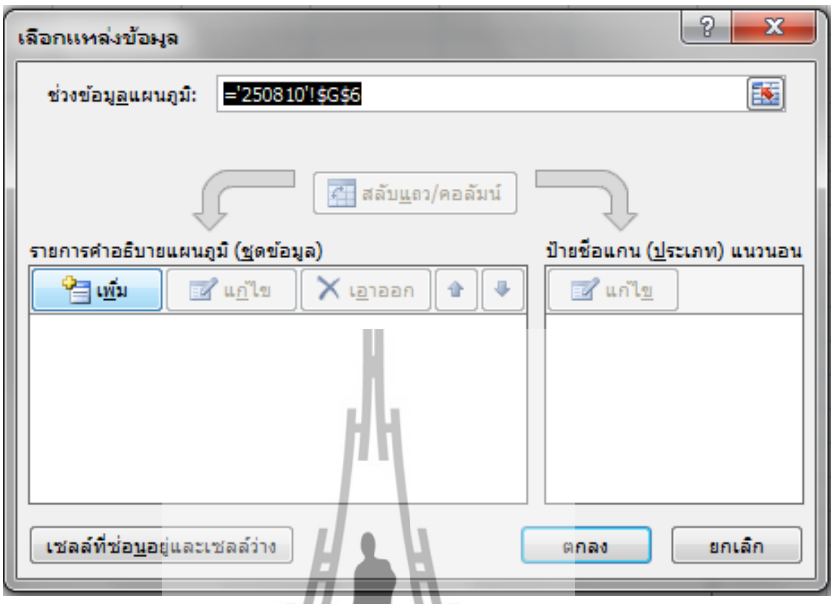
รูปที่ 3.29 การเลือกกราฟที่ใช้งาน

9. คลิกที่เลือกข้อมูล



รูปที่ 3.30 การเลือกข้อมูล

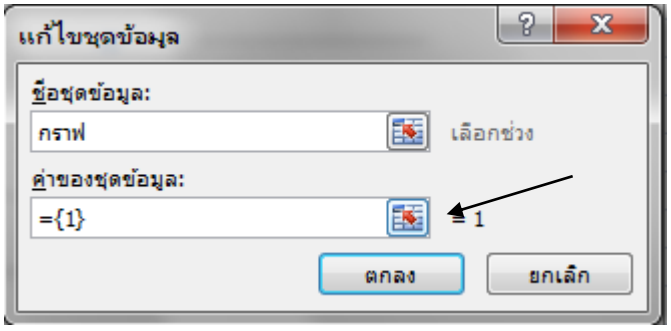
10. คลิกที่เพิ่ม



รูปที่ 3.31 การเลือกข้อมูล (ต่อ)

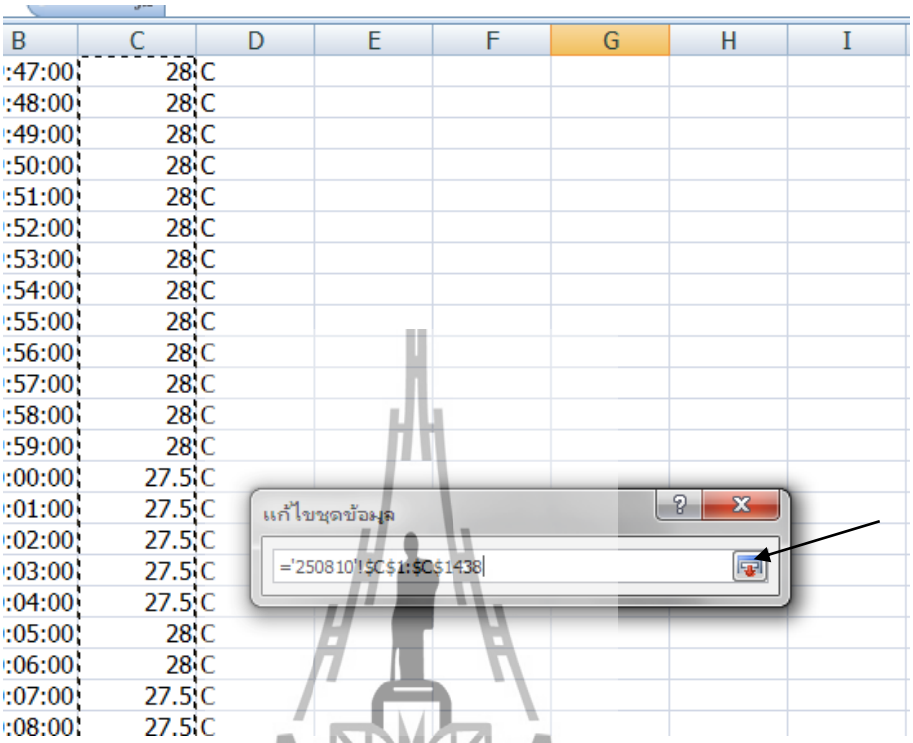
11. ช่องชื่อชุดข้อมูล ให้ใส่ชื่อที่เราต้องการ

ช่องค่าของชุดข้อมูล ให้คลิกที่ลูกศรชี้เพื่อทำการเลือกที่ข้อมูลที่ต้องการ



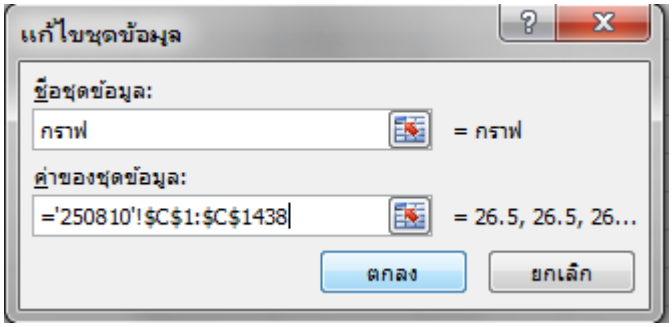
รูปที่ 3.32 การเลือกข้อมูลอุณหภูมิ

12. ทำกรอบคุมข้อมูลที่เป็นอุณหภูมิที่เราต้องการจากนั้นให้คลิกที่ลูกศรชี้



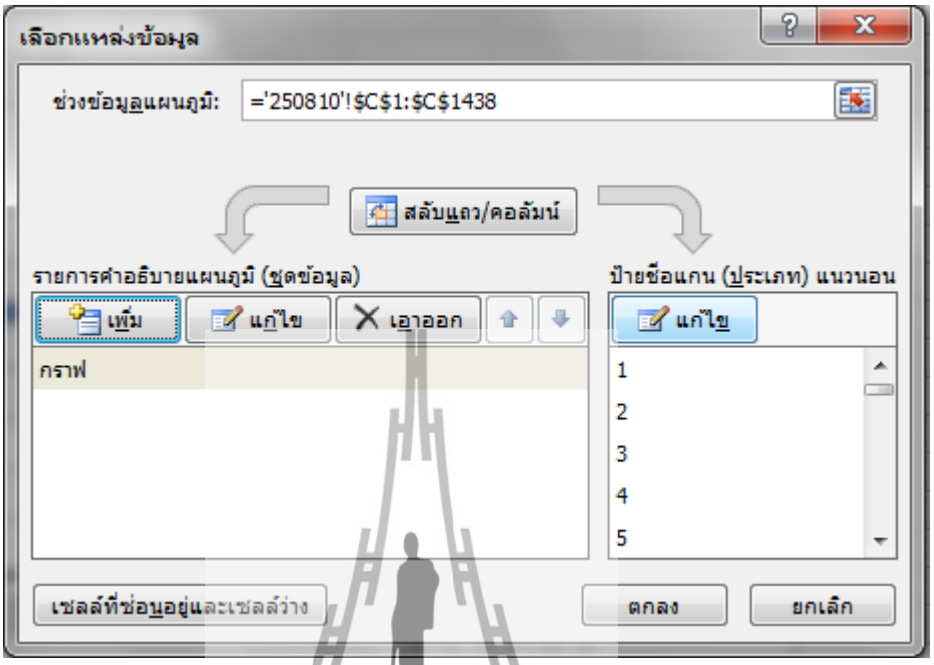
รูปที่ 3.33 การเลือกข้อมูลอุณหภูมิ (ต่อ)

13. คลิกตกลง



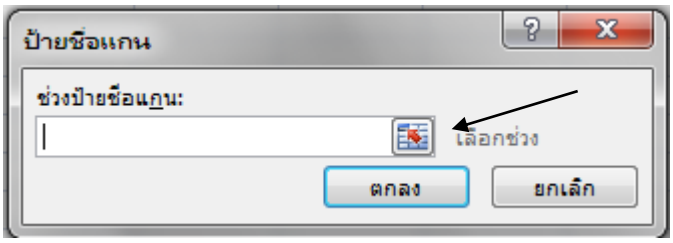
รูปที่ 3.34 การเลือกข้อมูลอุณหภูมิ (ต่อ)

14. คลิกที่แก้ไขข้อมูลเพื่อทำการเลือกข้อมูลที่เป็นเวลา



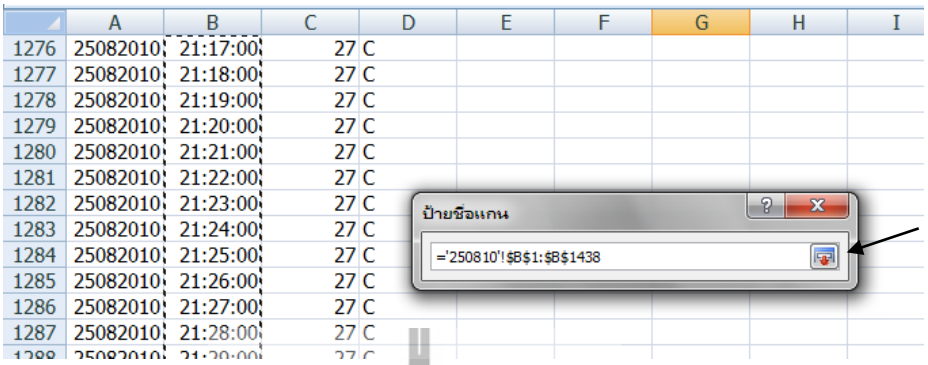
รูปที่ 3.35 การเลือกข้อมูลเวลา

15. คลิกที่ลูกศรชี้



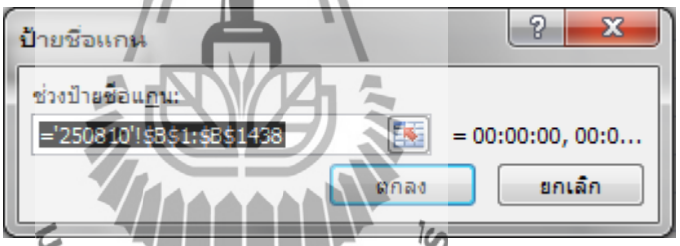
รูปที่ 3.36 การเลือกข้อมูลเวลา (ต่อ)

16. ทำการเลือกข้อมูลในส่วนที่เป็นเวลาแล้วคลิกที่ลูกศรชี้



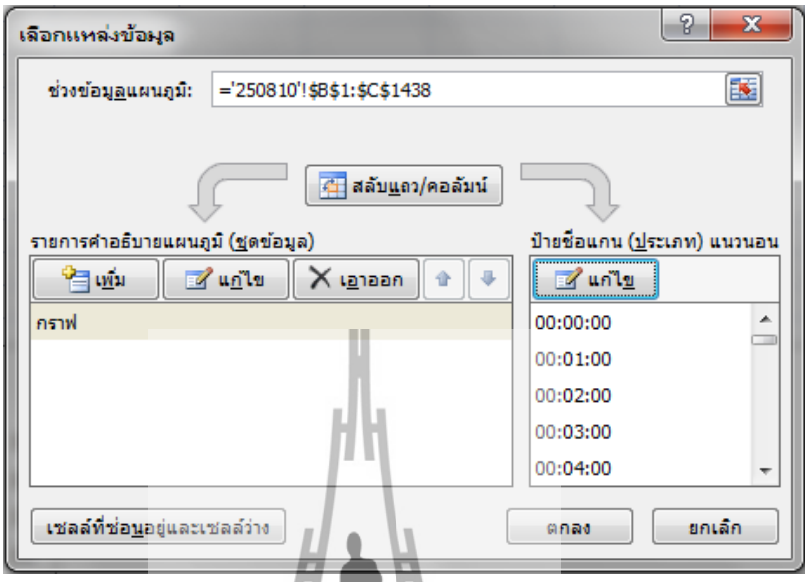
รูปที่ 3.37 การเลือกข้อมูลเวลา (ต่อ)

17. คลิกตกลง



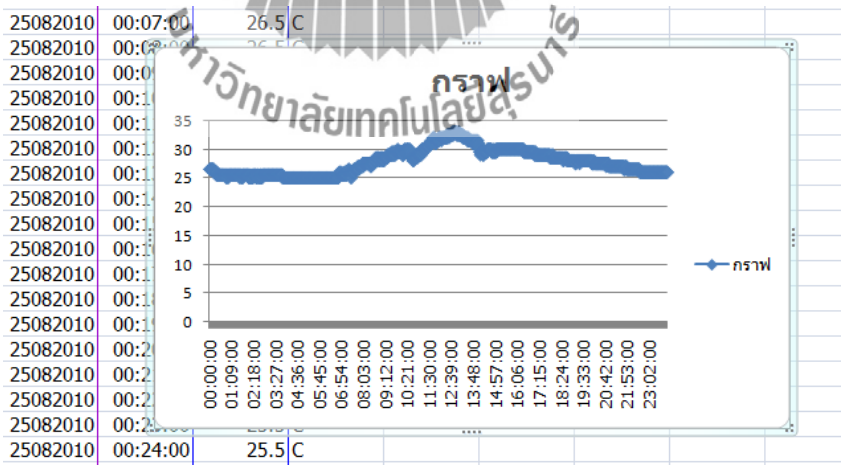
รูปที่ 3.38 การเลือกข้อมูลเวลา (ต่อ)

18. คลิกตกลง



รูปที่ 3.39 เสร็จสิ้นในการเลือกข้อมูล

19. กราฟที่ได้จากข้อมูล



รูปที่ 3.40 กราฟข้อมูลที่ได้จากการเลือกข้อมูล

3.2.7 การทดสอบการทำงาน

หลังจากที่ผ่านการทดสอบการทำงานเบื้องต้นเรียบร้อยแล้ว ที่หน้าจอจะปรากฏดังรูปที่ 3.2 ซึ่งการแสดงผลจะประกอบไปด้วย ค่าอุณหภูมิที่อ่านได้จาก DS1820 เป็นค่าอุณหภูมิในหน่วยองศาเซลเซียส การแสดงผลในส่วนใหญ่ของแอลซีดีจะเป็นการพล็อตค่าอุณหภูมิและส่วนที่เหลือ (ด้านล่างของจอ) จะเป็นการแสดงค่าอุณหภูมิแบบตัวเลขและเวลา(ชั่วโมง, นาทีและวินาที)

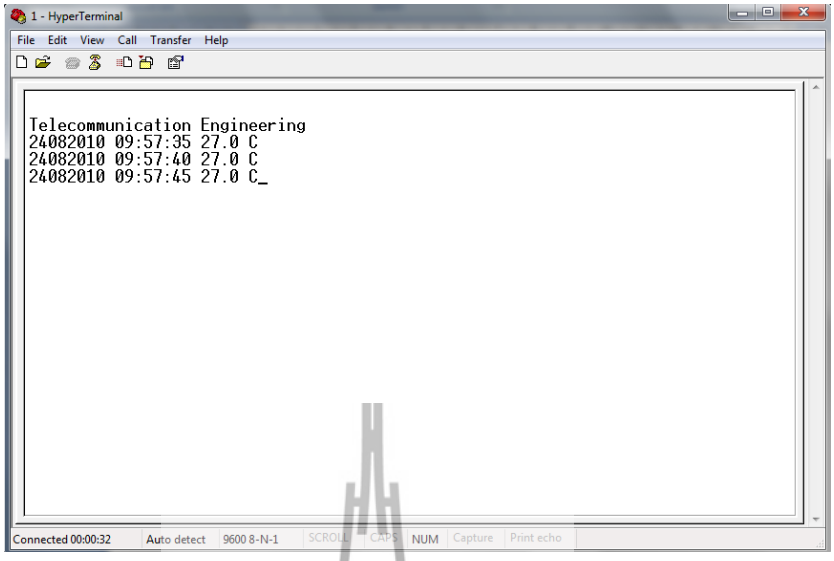
การตั้งค่าแสดงผลต่างๆสามารถทำได้ในโหมดการตั้งค่า โดยการกดปุ่ม "0" บนคีย์บอร์ดขนาด 4x3 แล้วใส่รหัสผ่าน "1449" แล้วกดปุ่ม "#" เพื่อเป็นการยืนยัน หากใส่รหัสผ่านไม่ถูกต้องที่หน้าจอจะไม่สามารถเข้าโหมดการตั้งค่าได้ แต่ถ้ารหัสผ่านถูกต้องที่หน้าจอจะปรากฏเป็นเมนูข้อความ (ดังรูปที่ 10) การแสดงผลในเมนูข้อความจะเป็นแถบบาร์หากต้องการเลื่อนแถบบาร์ขึ้นให้กดปุ่มเลข "1" และหากต้องการเลื่อนแถบบาร์ลงให้กดปุ่มเลข "7" หากต้องการเข้าไปยังส่วนย่อยของเมนูใดๆ ให้เลื่อนแถบบาร์มายังเมนูย่อยนั้นแล้วกดเครื่องหมาย "#" เพื่อเป็นการยืนยันการเข้าโปรแกรมนั้นๆสำหรับการตั้งค่าต่างๆมีดังนี้

การตั้งค่า วัน /เดือน/ปี (Set Date) ซอฟแวร์จะมีการป้องกันการป้อนค่าที่ไม่ถูกต้องอาทิเช่น โดยปกติแล้วเดือนเมษายนมี 30 วัน ดังนั้นวันที่ถูกต้องจะต้องมีค่าระหว่าง 1-30 รวมถึงการคำนวณสำหรับเดือนกุมภาพันธ์ กล่าวคือเดือนกุมภาพันธ์ในปีที่หารด้วย 4 ลงตัวจะมี 29 วัน ซึ่งการป้อนข้อมูลจะต้องทำให้ถูกต้อง หากข้อมูลถูกต้องค่าใหม่ที่ผ่านการตั้งค่าจะถูกอัพเดทไปยัง DS1307

การตั้งค่าเวลา (Set Time) ในส่วนนี้ก็มีการป้องกันการป้อนค่าที่ไม่ถูกต้องด้วย คือ การป้อนค่าชั่วโมงจะต้องอยู่ระหว่าง 00-23 และในการป้อนค่านาทีและวินาทีจะมีค่าอยู่ระหว่าง 00-59 หากการป้อนค่าต่างๆไม่ถูกต้องจะมีข้อความ "Can't Update" ฟ้องที่หน้าจอเพื่อแสดงให้ผู้ใช้ทราบว่าเกิดความผิดพลาดเกี่ยวกับป้อนค่า

ในส่วนเมนูการตั้งค่าที่ 3 และ 4 เป็นการเลือกรูปแบบของการแสดงผล ระหว่างการแสดงผลแบบเชิงเส้นทึบหรือแบบจุด (ดังรูปที่ 11 และ 12 ตามลำดับ) ส่วนสุดท้ายเป็นการออกจากโหมดการตั้งค่า (Exit) ในเมนูนี้หากกดเครื่องหมาย "#" จะเป็นการออกจากโหมดตั้งค่าไปยังหน้าจอการทำงานปกติ

การทำงานของไฟพื้นหลังจะปิดหลังจากที่ไม่มีการกดสวิตช์จากผู้ใช้ 1 นาที กล่าวคือ หากมีการกดคีย์บอร์ดใดๆๆไฟพื้นหลังจะติดสว่าง หลังจากนั้นหากไม่มีการปุ่มครบกำหนด 1 นาที ไฟพื้นหลังจะดับลงไป ในกรณีนี้เพื่อเป็นการยืดชั่วโมงการทำงานของไฟพื้นหลังให้มีช่วงการทำงานให้นานขึ้น ซึ่งไฟพื้นหลังนั้นไม่มีความจำเป็นที่ต้องติดตลอดเวลา (แต่ถ้าต้องการให้ไฟพื้นหลังติดตลอดเวลาก็สามารถแก้ไขที่ฮาร์ดแวร์หรือที่ซอสโค้ดได้ตามความสะดวกของผู้ใช้)

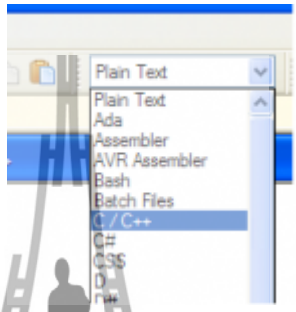


รูปที่ 3.41 ข้อมูลที่ส่งผ่าน พอร์ต RS232

3.3 เขียนโปรแกรมด้วย Programmer Notepad กับ WinAVR

การเขียนโปรแกรมภาษา C ร่วมกับใช้ AVR-GCC Compiler. Programmer Notepad เป็น Freeware ที่สามารถดาวน์โหลดมาใช้งานโดยไม่ต้องมีค่าใช้จ่ายใดๆ

1. ขั้นตอนแรกให้ทำการเปิดโปรแกรมแล้วเลือก Plain text เป็น C / C++



รูปที่ 3.42 การเลือกภาษาในการเขียน

เขียนโปรแกรมด้านล่างนี้ลงใน editor

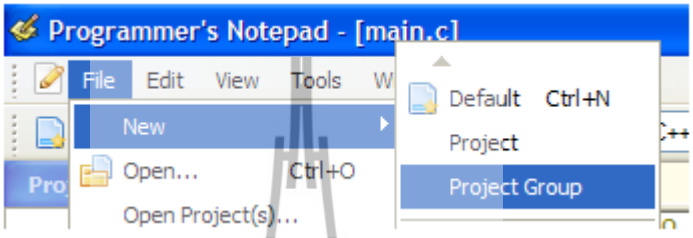
```
#include "avr/io.h"
#include "util/delay.h"

int main (void){
/* set PORTB for output*/
DDRB = 0xFF;
while (1) {
/* set PORTB.6 high */
PORTB = 0x20;
_delay_ms(1000);
/* set PORTB.6 low */
PORTB = 0x04;
_delay_ms(1000);
```

```
}
return 1;
}
```

2.จากนั้น Save ไฟล์ชื่อว่า main.c

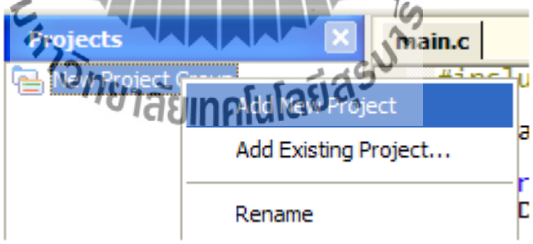
แล้วสร้าง Project Group ใหม่โดยไปที่ File > New > Project Group



รูปที่ 3.43 การสร้างกลุ่มโปรเจค

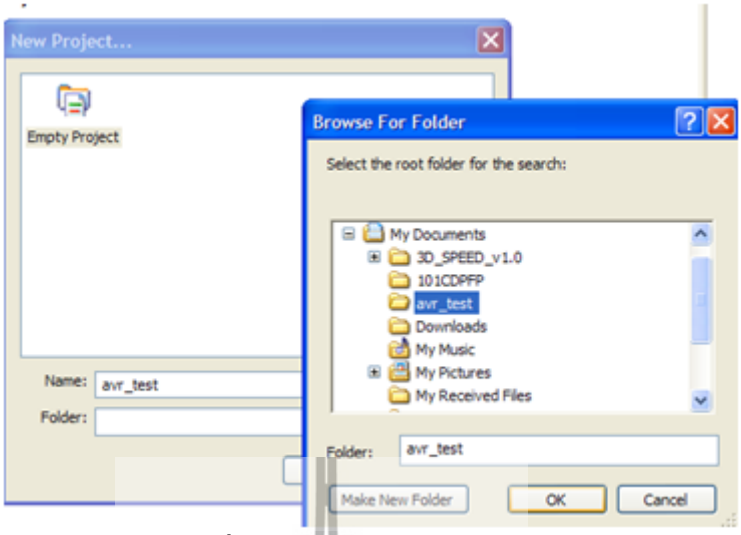
สร้าง Project Group

หลังจากที่สร้าง Group แล้ว ให้ Add New Project ใหม่โดยคลิกขวาที่ New Project Group> Add New Project > OK ตามรูปภาพด้านล่างนี้



รูปที่ 3.44 การสร้างโปรเจค

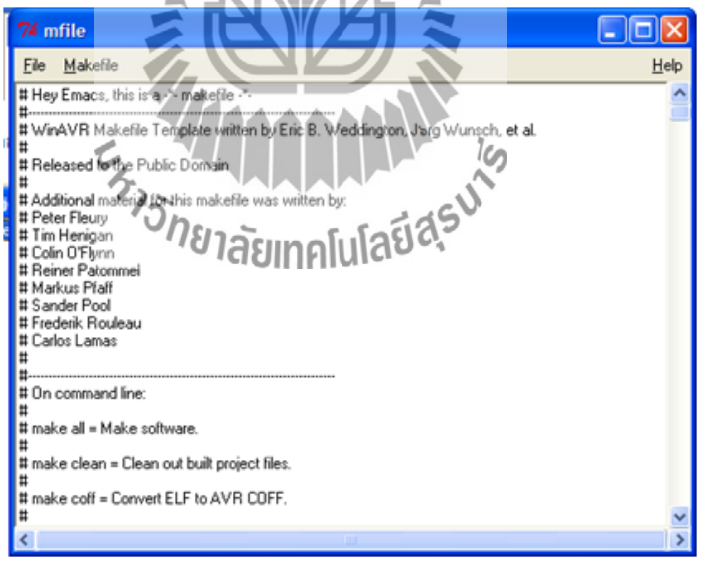
New Project Group > Add New Project



รูปที่ 3.45 เลือกที่เก็บไฟล์โปรเจค

ตั้งชื่อแล้วเลือก Folder ที่สร้างไว้ก่อนหน้านั้น

หลังจากนั้น Save Project Group จากนี้ให้สร้างไฟล์ Make โดยไปที่ WinAVR-20090313 > MFile [WinAVR] เมื่อคลิกแล้วจะปรากฏหน้าต่างดังภาพด้านล่างนี้



รูปที่ 3.46 หน้าต่าง Mfile

จากนั้นไปที่ เมนู Makefile > Enable Editng of Makefile จากนั้นให้ปรับแก้ไขดังนี้

Main file name: main

MCU Type: Atmega128

Output Format: ihex

Optimization level: S

Debug Format: ELF/DWARF-2 (AVR Studio 4.11+)

C standard level : gnu99

Programmer : avrisp

Port : com1

จากนั้น Save ไฟล์ไปยังทำแหน่ง Path ที่ได้สร้าง Project Group

จากนั้น คลิกขวาที่ avr_test > Add Files และทำการเลือกไฟล์ main.c เข้ามาใน Project และอีก 1 ไฟล์ที่ขาดไม่ได้ก็คือ ไฟล์ Makefile ก็ให้เพิ่มเข้ามาเช่นกัน จากนั้นทำการแก้ไข Makefile ให้เปลี่ยน F_CPU = 16000000 ใหม่เสียก่อน (อ้างอิงตาม XTALที่บอร์ดทดลอง) ผลที่ได้ตามรู้ด้านล่างนี้

รูปที่ 3.47 การ Save ไฟล์ไปยังทำแหน่ง Path ที่ได้สร้าง Project Group

คอมไพล์โปรแกรม ก่อนอื่นทำการ Clean ก่อนโดยเลือก Tools > [WinAVR] Make Clean ผลการ Make Clean ดังนี้

> “make.exe” clean

——— begin ———

Cleaning project:

rm -f main.hex

rm -f main.eep

rm -f main.cof

rm -f main.elf

rm -f main.map

rm -f main.sym

rm -f main.lss

rm -f ./main.o

rm -f ./main.lst

rm -f main.s

rm -f main.d

rm -f main.i

rm -rf .dep

——— end ———

> Process Exit Code: 0

> Time Taken: 00:01

จากนั้นทำการ Makefile เพื่อ คอมไพล์ไฟล์ main.c โดยเลือก Tools > [WinAVR] Make All ผลจากการคอมไพล์

> “make.exe” all

——— begin ———

avr-gcc (WinAVR 20090313) 4.3.2

Copyright (C) 2008 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO

warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiling C: main.c

avr-gcc -c -mmcu=atmega128 -I. -gdwarf-2 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=./main.lst -std=gnu99 -MMD -MP -MF .dep/main.o.d main.c -o main.o

Linking: main.elf

avr-gcc -mmcu=atmega128 -I. -gdwarf-2 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=main.o -std=gnu99 -MMD -MP -MF .dep/main.elf.d main.o –output main.elf -Wl,-Map=main.map,–cref -lm

Creating load file for Flash: main.hex

avr-objcopy -O ihex -R .eeprom -R .fuse -R .lock main.elf main.hex

Creating load file for EEPROM: main.eep

avr-objcopy -j .eeprom –set-section-flags=.eeprom=”alloc,load” \

–change-section-lma .eeprom=0 –no-change-warnings -O ihex main.elf main.eep || exit 0

Creating Extended Listing: main.lss

avr-objdump -h -S -z main.elf > main.lss

Creating Symbol Table: main.sym

avr-nm -n main.elf > main.sym

Size after:

AVR Memory Usage

————————-

Device: atmega128

Program: 240 bytes (0.2% Full)

(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)

(.data + .bss + .noinit)

——— end ———

> Process Exit Code: 0

> Time Taken: 00:00

ให้สังเกตจุดนี้หากมีความผิดพลาดใดๆ จะมีการรายงานผล ขณะนี้ไม่มี error ใดๆ เกิดขึ้น นั้นแสดงว่า Compiler สามารถแปลโปรแกรมที่เราเขียน และได้สามารถ hex ไฟล์ให้เรียบร้อยแล้ว hex ไฟล์ที่ได้จะอยู่ที่ Folder avr_test ชื่อว่า main.hex ซึ่งสามารถเปิดดูรายละเอียดภายในได้เช่นกัน สามารถใช้โปรแกรมนี้เปิดได้

ข้อมูล main.hex แสดงดังนี้

[code lang="c"]

:100000000C9446000C945D000C945D000C945D0013

:100010000C945D000C945D000C945D000C945D00EC

:100020000C945D000C945D000C945D000C945D00DC

:100030000C945D000C945D000C945D000C945D00CC

:100040000C945D000C945D000C945D000C945D00BC

:100050000C945D000C945D000C945D000C945D00AC

:100060000C945D000C945D000C945D000C945D009C

:100070000C945D000C945D000C945D000C945D008C

:100080000C945D000C945D000C945D0011241FBE67

:10009000CFEFD0E1DEBFCDBF11E0A0E0B1E0E0EFF7

:1000A000F0E000E00BBF02C007900D92A030B10756

:1000B000D9F70E945F000C9476000C9400008FEF3B

:1000C00087BB50E220E931E044E058BB80E197E291

:1000D000F9013197F1F70197D9F748BB80E197E231

:1000E000F9013197F1F70197D9F7EFCFF894FFCFE6

:00000001FF

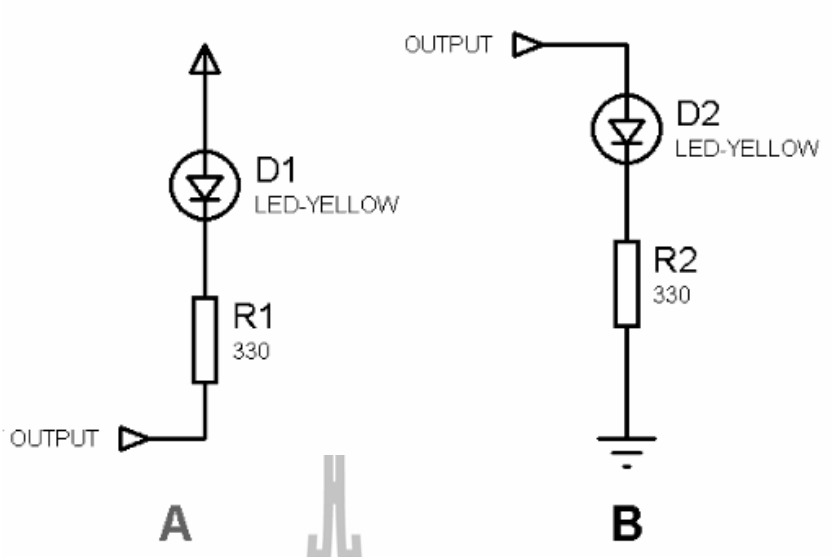
[/code]

3.3.1 การเขียนโปรแกรมแสดงผลบนอุปกรณ์แอลอีดี (Programming output on LED device)

ทฤษฎีพื้นฐาน

3.3.1.1 การเอาท์พุตออกอุปกรณ์แอลอีดี

การเอาท์พุตออกอุปกรณ์แอลอีดี (LED) ให้พิจารณาการต่อของวงจรแอลอีดี โดยทั่วไปจะต่อได้ 2 แบบคือ แบบ Inverting Output (รูปที่ 3.19 A) และแบบ Non-Inverting Output (รูปที่ 3.19 B) การต่อวงจรตามรูปที่ 3.19 B เมื่อป้อนออกตรรกะ 0 ให้หลอดแอลอีดีจะดับ และเมื่อป้อนออกตรรกะ 1 ให้หลอดแอลอีดีจะติดสว่าง เรียกได้ว่า การป้อนออกแบบตรง (Non-Inverting Output) แต่ในทางกลับกันเมื่อต่อวงจรตามรูปที่ 3.19 A การทำงานจะเปลี่ยนแปลงกลับจากรูปที่ 3.19 B นั้นคือ เมื่อป้อนออกตรรกะ 1 หลอดแอลอีดีจะสลับเป็นดับและเมื่อป้อนตรรกะ 0 หลอดแอลอีดีจากดับสลับเป็นติดสว่างขึ้นมา เรียกได้ว่า การป้อนออกแบบพลิกกลับ (Inverting Output) ซึ่งวิธีการป้อนออกแบบนี้เป็นที่นิยมมากกว่าการป้อนออกแบบตรง เนื่องวงจรไอซีทั่วไปสามารถขับกระแสขาเข้า (Sink Current)ได้ดีกว่าการขับกระแสขาออก (Source) อย่างไรก็ตามเมื่อต้องการที่จะขับกระแสหลอดแอลอีดีนั้นจำเป็นต้องสังเกตวงจรก่อนว่ามีการต่อในลักษณะใด



รูปที่ 3.48 ตัวอย่างวงจรแอลอีดีแบบ Inverting Output (A) และ Non-Inverting Output (B)

3.3.1.2 การควบคุมด้วยไมโครคอนโทรลเลอร์ AVR

การเขียนโปรแกรมติดต่ออินพุตเอาท์พุตของไมโครคอนโทรลเลอร์ AVR

ในการเขียนโปรแกรมติดต่อไมโครคอนโทรลเลอร์ จะต้องมีการ Include แฟ้ม Header ที่เกี่ยวข้องมาใช้งาน กรณีนี้ถ้าต้องการควบคุมการใช้งานพอร์ทขาต่าง ๆ ของไมโครคอลโทรลเลอร์ AVR จะต้อง include ไฟล์ Header “avr/io.h” ซึ่งมีการนิยามตัวแปร ฟังก์ชั่น และแมโครเกี่ยวกับไมโครคอลโทรลเลอร์ AVR รุ่นต่างๆไว้รูปแบบคำสั่งสามารถเขียนได้ดังนี้คือ

```
#include <avr/io.h>
```

กำหนดหน้าที่และค่าการทำงานเริ่มต้นของไมโครคอลโทรลเลอร์ AVR เพื่อใช้งานขาต่างๆ ทำได้โดยการตั้งค่ากลุ่มรีจิสเตอร์ (Registers) ของแต่ละพอร์ตขนาด 8 บิต ตามรายการดังนี้

1. รีจีสเตอร์ DDRx (Port Data Direction Register) ทำหน้าที่กำหนดทิศทางของสัญญาณ ของขาไมโครคอนโทรลเลอร์ให้เป็นการส่งออก (output) หรือรับเข้า (input) หากกำหนดเป็นลอจิก 0 จะ

เป็นอินพุตและหากกำหนดเป็นลอจิก 1 จะเป็นเอาท์พุต ตัวอย่าง กำหนดพอร์ต B ขาที่ 0-3 เป็นอินพุต และ ขาที่ 4-7 เป็นเอาท์พุต

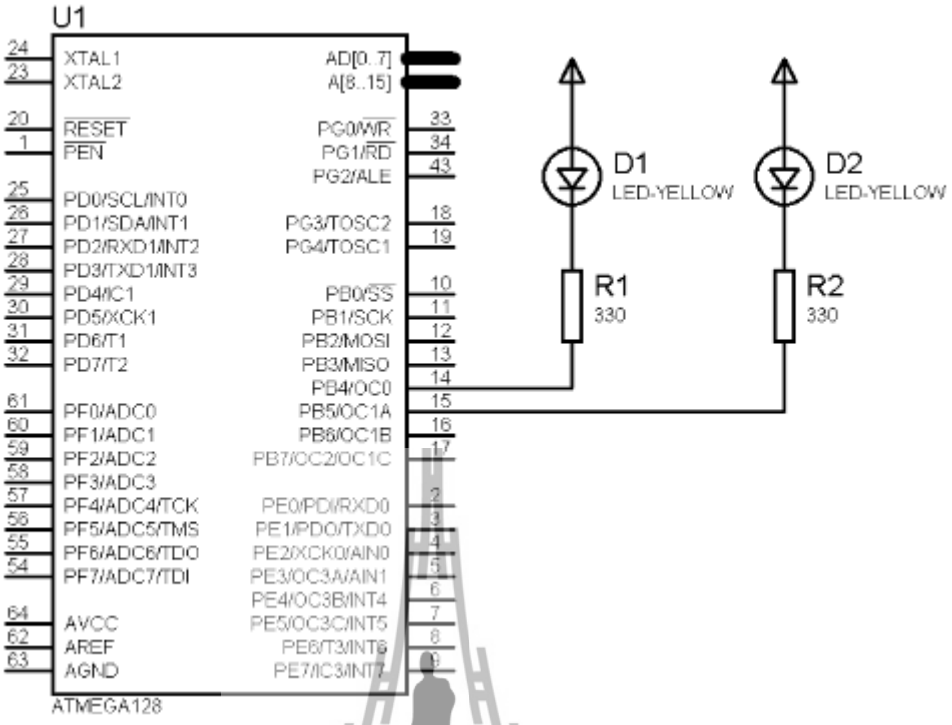
```
DDRB = 0xF0; // 1111 0000
```

2. รีจิสเตอร์ PORTx (Port x Data Register) ทำหน้าที่เก็บค่าที่จะส่งออก (output) แล้วส่งออกไปยังขาไมโครคอนโทรลเลอร์ หรือกำหนดการดึงขึ้น (pull up) ภายในด้วยการตั้งค่าบิตเป็น 1 เมื่อขามีทิศทางการรับเข้า (input) ตัวอย่าง การส่งค่าลอจิก 1 ออกที่พอร์ต B ขาที่ 6

```
DDRB = 0xF0; // 1111 0000
PORTB = 0x40; // 0100 0000
```

3. รีจิสเตอร์ PINx (Port x Input Pins Address) ทำหน้าที่รับค่า (input) สัญญาณที่เปลี่ยนแปลงบนขานั้น ซึ่งสามารถรับได้แม้มีทิศทางส่งออกตัวอย่าง การรับค่าที่พอร์ต B ขาที่ 3

```
DDRB = 0xF0; // 1111 0000
if ((PINB & 0x08) == 0) // 0000 1000
{
}
```



รูปที่ 3.49 ตัวอย่างการต่อแอลอีดีกับไมโครคอนโทลเลอร์ AVR

3.3.1.3 การเขียนโปรแกรมรับค่าจากสวิตช์โดยใช้อินเตอร์รับภายนอก (Programming input from switch device with external interrupt)

ทฤษฎีพื้นฐาน

1. การอินเตอร์รับ (Interrupt)

การอินเตอร์รับ คือ กระบวนการขัดจังหวะการทำงานของโปรแกรมปกติหรือโปรแกรมหลักที่กำลังทำงานอยู่ เพื่อให้เปลี่ยนมาทำงานในส่วนของโปรแกรมที่ได้กำหนดไว้ในอินเตอร์รับกระบวนการนี้จะช่วยให้ลดความเสี่ยงการเกิดข้อผิดพลาด จากการตรวจสอบเงื่อนไขในโปรแกรมหลักโดยกำหนดหน้าที่การตรวจสอบนี้ให้กับอินเตอร์รับแทน ประเภทของอินเตอร์รับในไมโครคอนโทรลเลอร์ จะแบ่งออกเป็น 2 ประเภทคือ

1. อินเตอร์รับภายนอก (External Interrupt) เป็นการตรวจสอบสัญญาณที่รับมาจากภายนอกตัวไมโครคอนโทรลเลอร์

2. อินเตอร์รับภายใน (Internal Interrupt) เป็นการตรวจสอบสัญญาณที่แหล่งกำเนิดสัญญาณเกิดจากวงจรภายในไมโครคอนโทรลเลอร์เอง

อินเตอร์รับภายนอก (External Interrupt)

การอินเตอร์รับภายนอกเป็นการตรวจสอบสัญญาณที่รับจากภายนอกตัวไมโครคอนโทรลเลอร์ ที่ขาของพอร์ตต่างๆ ตามที่ไมโครคอนโทรลเลอร์นั้นๆ อนุญาต (โดยดูได้จาก Data sheet ของไมโครคอนโทรลเลอร์นั้นๆ) เช่น Atmega64 จะมีขาที่เป็นอินเตอร์รับภายนอกได้คือ PD0 (INT0), PD1 (INT1), PD2 (INT2), PD3 (INT3), PE4 (INT4), PE5 (INT5), PE6 (INT6) และ PE7 (INT7) โดยสามารถ

กำหนดรูปแบบของสัญญาณการเกิดอินเตอร์รับภายนอกได้หลายรูปแบบดังนี้

1. ขณะที่ระดับสัญญาณเป็นลอจิกต่ำ
2. ขณะที่ระดับสัญญาณเปลี่ยนแปลง
3. ขณะที่ขอบขาลงของสัญญาณ (Falling edge)
4. ขณะที่ขอบขาขึ้นของสัญญาณ (Rising edge)

การควบคุมด้วยไมโครคอนโทรลเลอร์ AVR

1. การเขียนโปรแกรมใช้อินเตอร์รับภายนอกของไมโครคอนโทรลเลอร์ AVR

ในการเขียนโปรแกรมใช้งานอินเตอร์รับจะต้อง include ไฟล์ “avr/interrupt.h” และการตั้งค่ารูปแบบของอินเตอร์รับภายนอกสำหรับ Atmega64 สามารถกำหนดได้จากรีจีสเตอร์ EICRA, EICRB, EIMSK โดยหน้าที่ของแต่ละรีจีสเตอร์คือ

1. รีจีสเตอร์ EICRA จะกำหนดรูปแบบของสัญญาณการเกิดอินเตอร์รับ INT0 – INT3
2. รีจีสเตอร์ EICRB จะกำหนดรูปแบบของสัญญาณการเกิดอินเตอร์รับ INT4 – INT7
3. รีจีสเตอร์ EIMSK จะกำหนดเปิด/ปิดการใช้อินเตอร์รับ INT0 – INT7

โดยก่อนการเข้าไปกำหนดค่ารีจีสเตอร์ของอินเตอร์รับ จะต้องเรียกคำสั่งหยุดการทำงานของอินเตอร์รับทั้งหมดก่อน (รวมถึงอินเตอร์รับภายในด้วย) เพื่อไม่ให้เกิดการซ้อนกันของคำสั่ง โดยใช้คำสั่ง cli เพื่อหยุดอินเตอร์รับและ sei เพื่อเริ่มทำงานอินเตอร์รับ

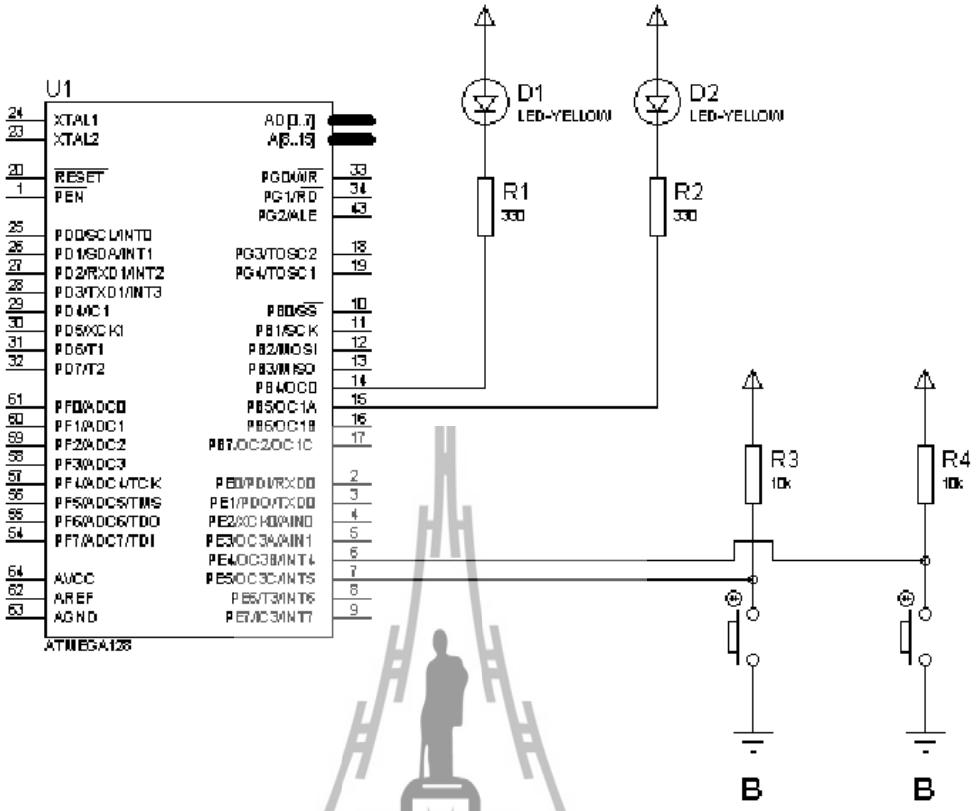
ตัวอย่าง การกำหนดค่าอินเตอร์รับ INT3 (PD3) และ INT4 (PE4)

```
#include <avr/io.h>
#include <avr/interrupt.h>
cli();
EICRA = _BV(ISC30)|_BV(ISC31); // INT3 is rising edge
EICRB = _BV(ISC40)|_BV(ISC41); // INT4 is rising edge
EIMSK = _BV(INT3)|_BV(INT4); // INT3, INT4 is enable
sei();
```

เมื่อเกิดการทำงานของอินเตอร์รับขึ้น โปรแกรมจะทำการกระโดดไปทำงานที่บรรทัดของอินเตอร์รับเวกเตอร์ของอินเตอร์รับนั้นๆ ซึ่งสำหรับอินเตอร์รับภายนอกนั้นอินเตอร์รับเวกเตอร์คือ

INTx_vect และสามารถเรียกใช้งานได้ดังตัวอย่างต่อไปนี้

```
ISR(INT4_vect)
{
//...
}
```

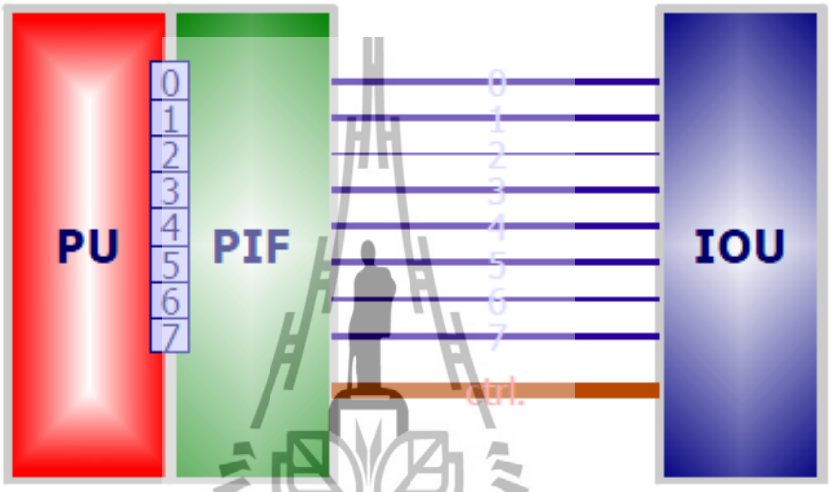


รูปที่ 3.50 ตัวอย่างการต่อสวิตช์กับไมโครคอนโทลเลอร์ AVR

3.3.1.4 การเขียนสื่อสารข้อมูลอนุกรมผ่านทาง UART (Programming serial communication by UART)

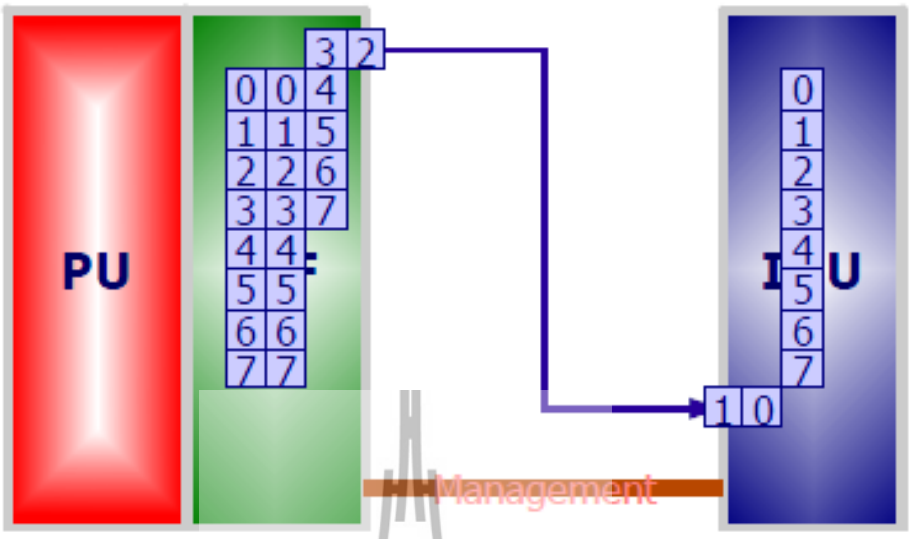
ทฤษฎีพื้นฐาน

1. การสื่อสารข้อมูลแบบอนุกรม (Serial communication) จากการเรียนรู้ที่ผ่านมาการรับส่งข้อมูลผ่านทางพอร์ตจะเป็นการสื่อสารแบบขนาน เพราะเวลาส่งข้อมูลขนาด 1 ไบต์ต้องใช้ขาสัญญาณในการส่งเท่ากับจำนวนบิตทั้งหมดคือ 8 ขา



รูปที่ 3.51 ตัวอย่างการสื่อสารข้อมูลแบบขนาน

เพื่อที่จะลดจำนวนขาในการสื่อสาร จึงต้องมีการสื่อสารแบบอนุกรมเข้ามาช่วย ถึงแม้ว่าการสื่อสารแบบอนุกรมจะใช้ขาสัญญาณที่น้อยกว่าและ สามารถส่งได้ไกลกว่า การสื่อสารแบบอนุกรมก็ยังมีข้อด้อยเรื่องความเร็วในการสื่อสารที่น้อยกว่าแบบขนาน และการเพิ่มเติมวงจรเพื่อแปลงการสื่อสารแบบอนุกรมให้เป็นแบบขนาน ดังรูป



รูปที่ 3.52 ตัวอย่างการสื่อสารข้อมูลแบบอนุกรม

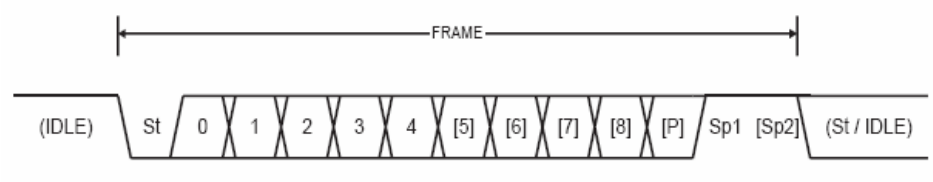
รูปแบบการสื่อสารแบบอนุกรมจะมีอยู่ 3 แบบคือ

- การสื่อสารแบบทิศทางเดียว (simplex) ซึ่งสามารถรับหรือส่งได้เพียงอย่างเดียวเท่านั้น
- การสื่อสารแบบสองทางครึ่งอัตรา (half duplex) ซึ่งสามารถรับและส่งได้แต่ไม่พร้อมกัน
- การสื่อสารแบบสองทางเต็มอัตรา (full duplex) ซึ่งสามารถรับและส่งได้ในเวลาเดียวกัน

2. การสื่อสารข้อมูลแบบ UART

การสื่อสารข้อมูลแบบ UART เป็นการสื่อสารอนุกรมในรูปแบบ full duplex ซึ่งใช้ขา Rx ในการรับข้อมูลและ ใช้ขา Tx ในการส่งข้อมูล การสื่อสารจะเป็นแบบ Asynchronous ดังนั้นจึงไม่มีขาสัญญาณในการ Synchronize ข้อมูล ทำให้ฝ่ายรับและฝ่ายส่งจำเป็นที่จะต้องรู้รูปแบบของข้อมูลและ ความเร็วในการสื่อสาร การใช้งาน UART จะเอาไปประยุกต์ใช้กับมาตรฐานการสื่อสารอนุกรมแบบ RS232 หรือ RS485 เป็นต้น

รูปแบบของข้อมูล (Frame format) ในการสื่อสารของ UART จะมีรูปแบบดังนี้



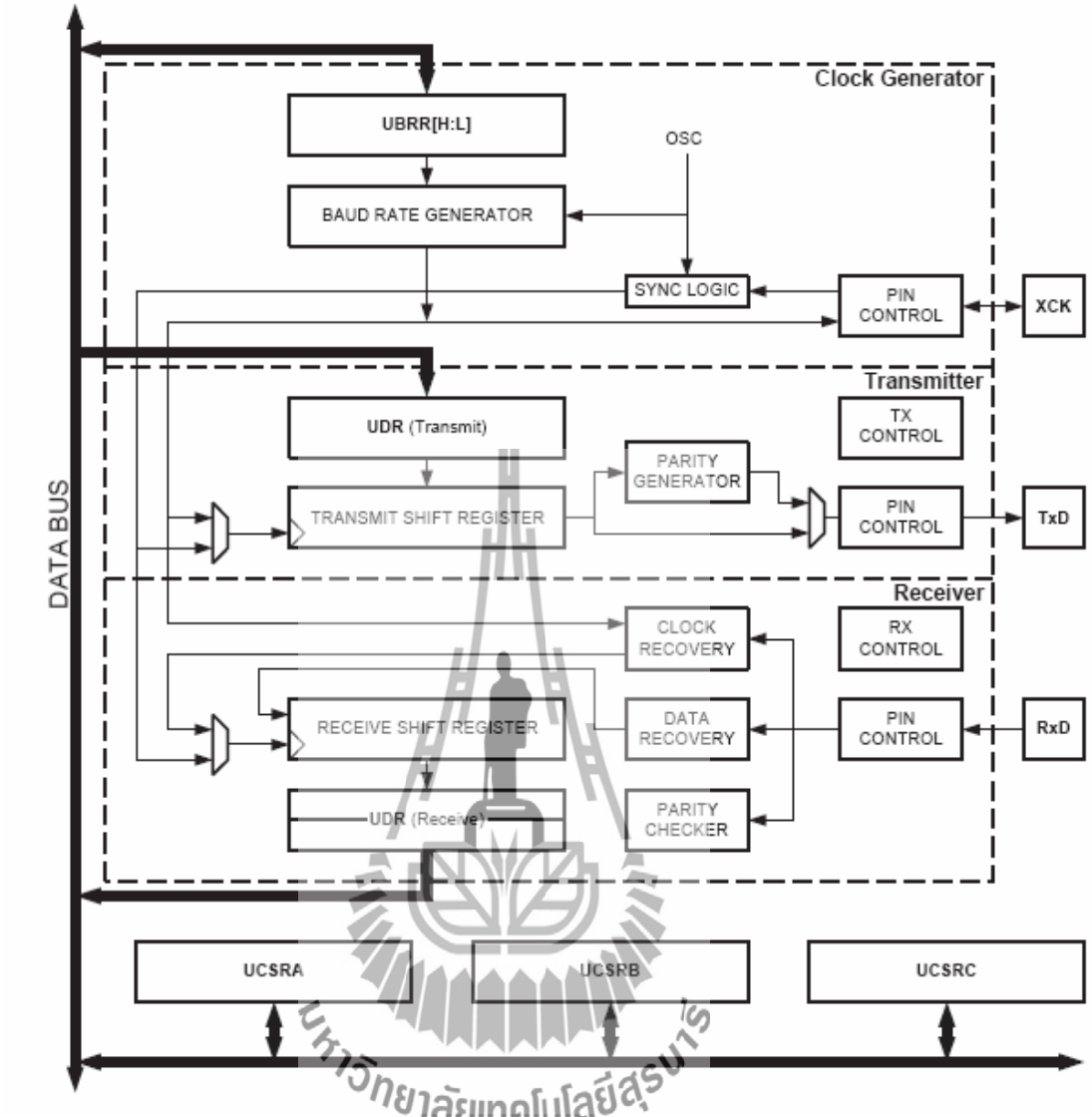
รูปที่ 3.53 Frame format

- IDLE เป็นสภาวะของระดับสัญญาณที่ไม่มีการรับหรือส่งข้อมูล ปกติมีค่าเป็น high
- St (Start bit) เป็นบิตเริ่มต้นของข้อมูล ปกติมีค่าเป็น low
- N (1 – 8) เป็นค่าข้อมูลของแต่ละบิต
- P (Parity) เป็นบิตในการตรวจสอบความถูกต้องของข้อมูลโดยการนับจำนวนบิตที่มีค่าเป็น 1 ว่าเป็นจำนวนคู่หรือคี่
- Sp (Stop bit) เป็นบิตสิ้นสุดข้อมูลซึ่งอาจจะมี 1 หรือ 2 บิตขึ้นอยู่กับการกำหนด ค่าปกติจะมีค่าเป็น high

การควบคุมด้วยไมโครคอนโทรลเลอร์ AVR

1. การเขียนโปรแกรมควบคุม UART ของไมโครคอนโทรลเลอร์ AVR

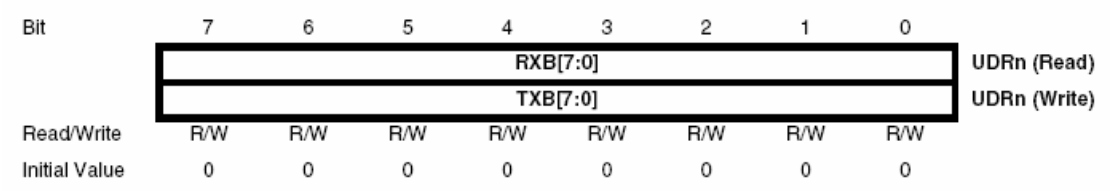
โครงสร้างการทำงานของ UART ภายใน Atmega1281 จะมีลักษณะ ดังรูป



รูปที่ 3.54 โครงสร้างการทำงานของ UART

จากรูปด้านบนจะเห็นได้ว่า UART ของ ATmega1281 จะรองรับการทำงานแบบ Synchronous เพิ่มเติมด้วยหรือเรียกว่า USART โดยจะมีขา XCK เพื่อใช้ในการ Synchronize ข้อมูล แต่ในที่นี้จะกล่าวถึงเฉพาะการทำงานแบบ Asynchronous เท่านั้น รีจิสเตอร์ที่เกี่ยวข้องกับการทำงานของ UART จะมีอยู่ 5 รีจิสเตอร์ด้วยกันคือ UCSRnA, UCSRnB, UCSRnC, UDRnและ UBRRn ซึ่งรีจิสเตอร์แต่ละตัวมีหน้าที่ต่างๆ ดังนี้

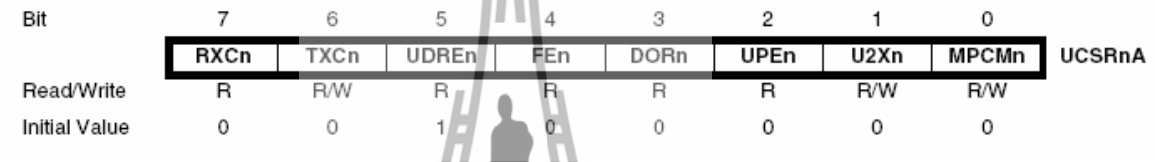
- UDRn (UART I/O Data register) เป็นรีจิสเตอร์ที่ใช้รับและส่งข้อมูล



Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

รูปที่ 3.55 UDRn Register

- **UCSRnA** (UART Control and status register a) เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานและตรวจสอบสถานะต่างๆ ของ UART ซึ่งมีรายละเอียดของบิตต่างๆ ดังนี้



Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

รูปที่ 3.56 UCSRnA Register

1. RXCn เป็นบิตบอกสถานะของการรับข้อมูล ซึ่งจะมีค่าเป็น 1 เมื่อได้รับข้อมูลเรียบร้อยแล้ว
2. TXCn เป็นบิตบอกสถานะของการส่งข้อมูล จะมีค่าเป็น 1 เมื่อส่งข้อมูลเสร็จเรียบร้อยแล้ว ซึ่งสามารถเคลียร์ค่าให้เป็นลอจิก 0 ได้โดยการเขียน ลอจิก 1 ไปที่บิตนี้
3. UDREn เป็นบิตบอกสถานะของรีจิสเตอร์ UDRn ว่าพร้อมที่จะรับข้อมูลใหม่เพื่อไปส่งได้หรือไม่ ค่าบิตจะเป็นลอจิก 1 เมื่อพร้อมที่จะส่งข้อมูลได้
4. FEn เป็นบิตบอกสถานะของการรับข้อมูลที่ผิดพลาดซึ่งเกิดจาก Stop bit มีค่าเป็นลอจิก 0
5. DORn เป็นบิตบอกสถานะของการส่งการได้รับข้อมูลมาใหม่โดยที่ยังไม่ได้อ่านข้อมูลเก่าออกไปจากรีจิสเตอร์ UDRn
6. UPEn เป็นบิตบอกความผิดพลาดเมื่อตรวจสอบพาริตี้แล้วไม่ถูกต้อง
7. U2Xn เป็นบิตที่ใช้ในการความคุมความเร็วในการสื่อสารให้เพิ่มขึ้น 2 เท่าถ้ามีการกำหนดค่าเป็นลอจิก 1
8. MPCMn เป็นบิตที่ใช้ในการกำหนดให้เป็นโหมดการสื่อสารแบบหลายหน่วยประมวลผล

- **UCSRnB** (UART Control and status register b) เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานและตรวจสอบสถานะต่างๆ ของ UART ซึ่งมีรายละเอียดของบิตต่างๆ ดังนี้

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

รูปที่ 3.57 UCSRnB Register

1. RXCIEn เป็นบิตที่กำหนดให้มีการเกิดอินเตอร์รัพท์ เมื่อได้รับข้อมูลเรียบร้อยแล้ว
2. TXCIEn เป็นบิตที่กำหนดให้มีการเกิดอินเตอร์รัพท์ เมื่อส่งข้อมูลเสร็จเรียบร้อยแล้ว
3. UDRIEn เป็นบิตที่กำหนดให้มีการเกิดอินเตอร์รัพท์ เมื่อรีจิสเตอร์ UDRn พร้อมส่งข้อมูล
4. RXENn เป็นบิตที่กำหนดให้ UART สามารถรับข้อมูลได้
5. TXENn เป็นบิตที่กำหนดให้ UART สามารถส่งข้อมูลได้
6. UCSZn2 เป็นบิตที่ใช้กำหนดจำนวนข้อมูลที่จะสื่อสารใน 1 ครั้ง
7. RXB8n เป็นบิตข้อมูลที่ได้รับเพิ่มเติมในกรณีที่ขนาดข้อมูลเกิน 8 บิต
8. TXB8n เป็นบิตข้อมูลที่ใช้ส่งเพิ่มเติมในกรณีที่ขนาดข้อมูลเกิน 8 บิต

- **UCSRnC** (UART Control and status register C) เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานและตรวจสอบสถานะต่างๆ ของ UART ซึ่งมีรายละเอียดของบิตต่างๆ ดังนี้

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

รูปที่ 3.58 UCSRnC Register

1. UMSELn1, UMSELn0 เป็นบิตในการเลือกโหมดการสื่อสารของ UART ซึ่งมีทั้งหมด 4 โหมดตามค่าต่างๆ ดังนี้

ตาราง การกำหนดโหมดในการสื่อสาร

UMSELn1	UMSELn0	โหมด
0	0	Asynchronous
0	1	Synchronous
1	0	Reserved
1	1	Master SPI

รูปที่ 3.59 การกำหนดโหมดในการสื่อสาร

2. UPMn1, UPMn0 เป็นบิตในการเลือกรูปแบบพาริตี้เพื่อตรวจสอบข้อมูล ซึ่งจะมีค่าต่างๆ ดังนี้

ตาราง การกำหนดพาริตี้

UPMn1	UPMn0	โหมด
0	0	Disable
0	1	Reserved
1	0	Even parity
1	1	Odd parity

รูปที่ 3.60 การกำหนดพาริตี้

3. USBSn เป็นบิตในการกำหนดจำนวน Stop bit ถ้าเป็นลอจิก 0 จะมี 1 บิต ถ้าเป็นลอจิก 1 จะมี 2 บิต
4. UCSZn1, UCSZn0 รวมถึง UCSZn2 ที่อยู่ในรีจิสเตอร์ UCSRnB จะทำหน้าที่ในการกำหนดจำนวนบิตของข้อมูลที่สื่อสารแต่ละครั้งดังตาราง

ตารางการกำหนดขนาดข้อมูล

UCSZn2	UCSZn1	UCSZn0	ขนาดข้อมูล
0	0	0	5 บิต
0	0	1	6 บิต
0	1	0	7 บิต
0	1	1	8 บิต
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9 บิต

รูปที่ 3.61 การกำหนดขนาดข้อมูล

5. UCPOLn เป็นบิตที่ใช้กำหนดความสัมพันธ์ระหว่างข้อมูลและสัญญาณนาฬิกา ซึ่งใช้ในโหมด Synchronous เท่านั้น

- **UBRRnL, UBRRnH** (UART Baud rate register) เป็นรีจิสเตอร์ที่ใช้กำหนดความเร็วในการสื่อสารซึ่งค่าที่กำหนดต้องมีความสัมพันธ์กับสัญญาณนาฬิกาที่ป้อนให้ชิปทำงาน

ตารางการคำนวณค่า UBRR จาก Baud Rate

โหมด	สูตรคำนวณ
Asynchronous (U2Xn = 0)	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous (U2Xn = 1)	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$

รูปที่ 3.62 การคำนวณค่า UBRR จาก Baud rate

3.4 รายการอุปกรณ์ของเครื่องวัดอุณหภูมิเอนกประสงค์

ตัวต้านทาน ขนาด ¼ วัตต์ บวกลบ 5%

1. 10 โอห์ม 1 ตัว
2. 10 โอห์ม (1206) 1 ตัว
3. 220 โอห์ม (1206) 1 ตัว
4. 470 โอห์ม 1 ตัว
5. 680 โอห์ม 1 ตัว
6. 1 กิโลโอห์ม (1206) 1 ตัว
7. 4.7 กิโลโอห์ม 2 ตัว
8. 10 กิโลโอห์ม (1206) 4 ตัว
9. 10 กิโลโอห์ม 1 ตัว
10. 10 กิโลโอห์มแบบเกือกม้าปรับค่าได้ 1 ตัว

ตัวเก็บประจุ

1. 220 ไมโครฟารัด 16 โวลต์ 6 ตัว
2. 0.47 ไมโครฟารัด 4 ตัว
3. 10 พิโคฟารัด 2 ตัว
4. 22 พิโคฟารัด 2 ตัว

อุปกรณ์สารกึ่งตัวนำ

1. ATMEGA64 หรือ ATMEGA128 1 ตัว
2. DS1820 1 ตัว
3. DS1307 1 ตัว
4. 74LVC245 1 ตัว
5. LM1117T33 1 ตัว
6. MAX232 1 ตัว
7. MMBT3904 1 ตัว
8. B772 1 ตัว
9. LED สีแดง,สีเขียว 2 ตัว

อื่นๆ

1. กราฟิกแอลซีดีขนาด 128x64 1 ตัว
2. คริสตอล 16 MHz 1 ตัว
3. คริสตอล 32.768 KHz 1 ตัว
4. คอนเน็กเตอร์ 40 ขา 1 ตัว
5. คอนเน็กเตอร์ 20 ขา 1 ตัว
6. คอนเน็กเตอร์ขางอ 4 ขา 1 ตัว
7. คอนเน็กเตอร์ขางอ 2 ขา 1 ตัว
8. คีย์บอร์ดขนาด 4x3 1 ตัว
9. สวิตช์กดติดปล่อยดับ 1 ตัว
10. บัสเซอร์ 5 โวลต์ 1 ตัว
11. แบตตารี 3 โวลต์พร้อมรังถ่าน 1 ชุด

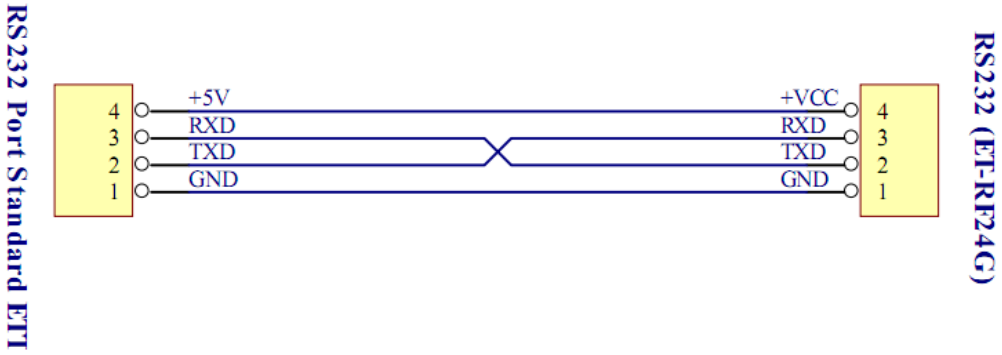
รายการอุปกรณ์อื่นๆ ที่เกี่ยวข้อง

1. ชุด รับ-ส่ง ข้อมูล RS232 แบบไร้สาย 1 ชุด
2. แบตตารี 6 โวลต์ 1 แอมป์ 1 ก้อน

3.5 ชุดรับ-ส่ง ข้อมูล RS232 แบบไร้สาย

รูปที่ 3.63 ชุด รับ-ส่ง ข้อมูล RS232 แบบไร้สาย รุ่น ET-RF24G V1.0

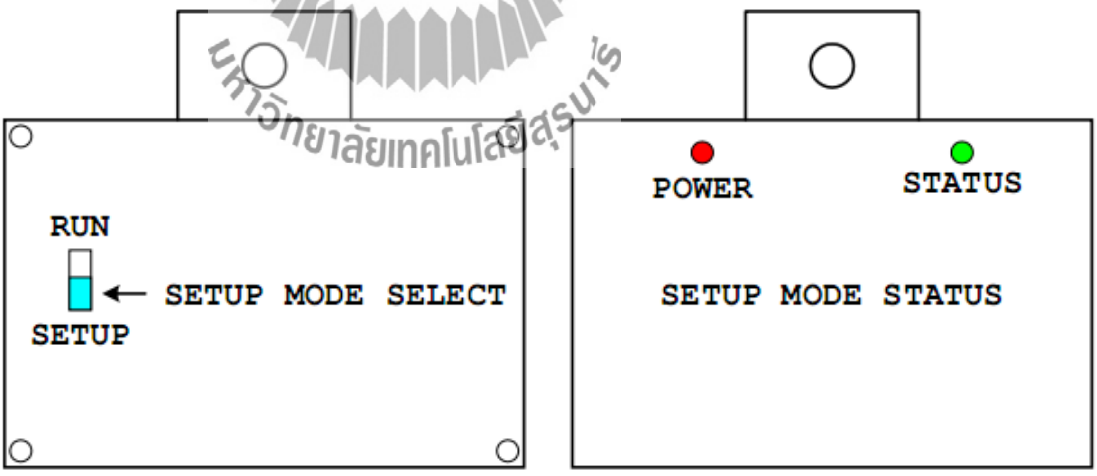
ชุด รับ-ส่ง ข้อมูล รุ่น ET-RF24G V1.0 เป็นชุด Signal converter สำหรับใช้แปลงสัญญาณระหว่าง RS232 และ RF-Wireless โดยในโหมดการทำงานของการส่งข้อมูล (Transmitter) จะทำหน้าที่รอรับข้อมูลจากพอร์ตสื่อสารอนุกรม RS232 จากขา RX แล้วแปลงสัญญาณความถี่ (GFSK :) ส่งออกไปในอากาศ และในโหมดรับ (Receiver) ชุด ET-RF24G V1.0 จะทำหน้าที่คอยตรวจจับข้อมูลที่อยู่ในรูปของสัญญาณความถี่ (GFSK) จากด้าน RF เพื่อแปลงกลับเป็นข้อมูลแบบ RS232 ส่งออกไปทางขา TX ได้ด้วย



รูปที่ 3.64 การต่อสายสัญญาณ RS232 เพื่อใช้แหล่งจ่ายจากบอร์ดไมโครฯ

3.5.1 การตั้งค่าการใช้งานเครื่อง ET-RF24G V1.0

การใช้งานในโหมด Setup mode ซึ่งเป็นโหมดใช้กำหนดค่า Configuration ต่างๆ ในการตั้งค่าต่างๆ นั้นจะกระทำร่วมกับโปรแกรม "ET_RF24G_V1.EXE" เมื่อเข้าสู่โหมด Setup แล้ว จะสังเกตุเห็นหลอดไฟแสดงสถานะการทำงาน หรือ LED STATUS ติดสว่างค้างอยู่ตลอดเวลา แต่เมื่อมีการสั่งอ่านหรือเขียนข้อมูลกับบอร์ด สถานการณ์ทำงานของ LED STATUS จึงจะกระพริบตามจังหวะของการส่งข้อมูล แต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะติดค้างอยู่ตลอดเวลา

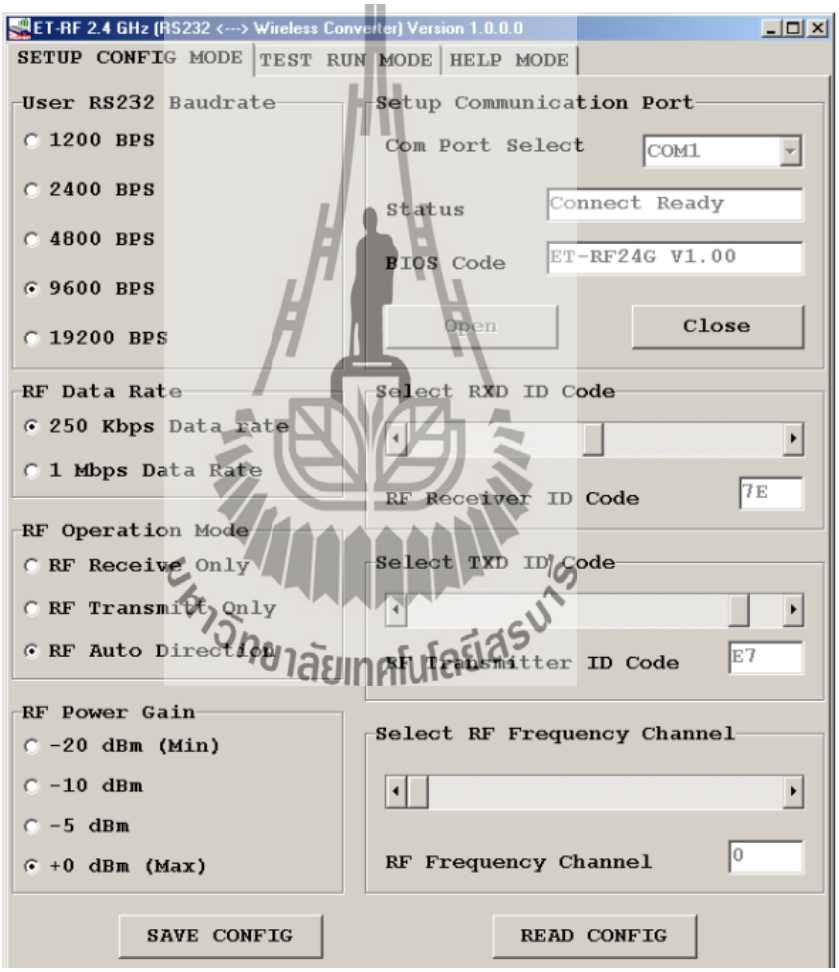


รูปที่ 3.65การเลือกโหมดการทำงาน สำหรับกำหนดค่า Configuration (Setup mode)

การกำหนดค่านั้นจะต้องกระทำขณะที่ตัวเครื่องทำงานอยู่ใน Setup mode เท่านั้น โดยค่าต่างๆนั้นจะถูกใช้สำหรับเป็นเงื่อนไขในการทำงานของเครื่องในขณะที่อยู่ใน Run mode ดังนั้นก่อน

การเริ่มต้นใช้งานเครื่องในครั้งแรกควรจะต้องทำการกำหนดค่าของ Configuration ต่างๆ โดยเมื่อกำหนดค่าตัวเลือกต่างๆของ Configuration เรียบร้อยแล้ว ก็สามารถเปลี่ยนโหมดการทำงานของตัวเครื่องกลับเป็น Run mode พร้อมกับการปิดไฟที่จ่ายให้กับตัวเครื่องชั่วขณะหนึ่ง จากนั้นจึงเริ่มต้นจ่ายไฟให้กับตัวเครื่องใหม่ ก็จะสามารถใช้งานได้ตามค่า Configuration ที่กำหนดไว้แล้วได้ทันที

3.5.2 คุณสมบัติของ Configuration



รูปที่ 3.66 รูปแบบโปรแกรมที่ใช้สำหรับกำหนดค่า Configuration

User RS232 Baudrate ใช้สำหรับกำหนดค่าความเร็วในการรับส่งข้อมูลทางด้าน RS232 ของตัวเครื่องในขณะที่ทำงานอยู่ใน Run mode ซึ่งสามารถกำหนดได้ 5 ค่าคือ

1. 1200 BPS
2. 2400 BPS
3. 4800 BPS
4. 9600 BPS
5. 19200 BPS

RF Data rate ใช้สำหรับกำหนดความเร็วในการรับส่งข้อมูลทางด้าน RF ของเครื่อง ซึ่งจะต้องกำหนดให้เครื่องทุกๆ ตัว ที่จะนำมาใช้ติดต่อสื่อสารกัน มีค่าอัตราความเร็วในการรับส่งข้อมูลด้าน RF หรือ RF Data Rate นี้มีค่าเท่ากันหมด ถ้ากำหนดค่าความเร็วต่างกันจะไม่สามารถส่งข้อมูลกันได้ โดยค่า RF Data rate สามารถกำหนดได้ 2 ค่าคือ

1. 250 Kbps
2. 1 Mbps

RF Operation mode ใช้สำหรับกำหนดโหมดของเครื่อง ซึ่งสามารถกำหนดหน้าที่การทำงานได้ 3 แบบ คือ

1. **RF Receive only** เป็นการกำหนดให้เครื่องทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RF เพื่อเปลี่ยนข้อมูลแบบ RS232 และส่งออกไปทางด้านขา TX ของ RS232 ตลอดเวลา
2. **RF Transmit only** เป็นการกำหนดให้เครื่องทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RS232 จากขา RX เพื่อเปลี่ยนเป็นข้อมูลแบบ GFSK และส่งออกไปทางด้าน RF ตลอดเวลา
3. **RF Auto direction** เป็นการกำหนดโหมดการทำงานแบบ Half duplex 2 ทิศทาง ซึ่งสามารถสลับโหมดการทำงานระหว่างการรับและส่งข้อมูลได้เองโดยอัตโนมัติ

RF Power gain เป็นการกำหนดกำลังส่งของวงจร RF Power ที่ใช้ในการส่งข้อมูล โดยค่า +0dBm เป็นค่ากำลังส่งสูงสุด ส่วน -20dBm เป็นค่ากำลังส่งต่ำสุด โดยสามารถกำหนดได้ 4 ระดับคือ

1. -20dBm (กำลังส่งต่ำสุด)
2. -10dBm
3. -5dBm
4. +0dBm (กำลังส่งสูงสุด)

RXD ID Code เป็นรหัส ID Code ของเครื่องในโหมดของการรับส่งข้อมูลจาก RF โดยเมื่อเครื่องด้านส่งจะทำการส่งข้อมูลออกไปทาง RF นั้นจะมีการระบุหมายเลข ID Code ของด้านรับรวมไปกับชุดข้อมูลด้วยเสมอ ส่วนทางด้านรับเมื่อรับข้อมูลจากด้าน RF ได้จะทำการเปรียบเทียบรหัส ID Code ที่รวมมากับข้อมูลว่าตรงกับรหัสของ RXD ID Code ที่กำหนดไว้ ถ้าถูกต้องจะทำการแยกเฉพาะส่วนของข้อมูลที่รับได้เพื่อเปลี่ยนเป็นข้อมูลแบบ RS232 แล้วส่งออกไปทางด้าน TX ของ RS232 โดยค่า RXD ID Code นั้นสามารถกำหนดได้ 256 ค่าในรูปแบบของเลขฐานสิบหก (00H-FFH)

TXD ID Code เป็นรหัส ID Code ปลายทางที่ส่งข้อมูลไปหา โดยที่เครื่องที่ถูกกำหนดให้เป็นฝ่ายส่งข้อมูลนั้น จะนำข้อมูลที่รับได้จาก RS232 ไปเข้ารหัสรวมกับ TXD ID Code แล้วส่งออกทางด้าน RF โดยค่า TXD ID Code นั้นสามารถกำหนดได้ 256 ค่าในรูปแบบของเลขฐานสิบหก (00H-FFH)

RF Frequency channel เป็นการกำหนดค่าของช่องความถี่ที่จะใช้ในการรับส่งข้อมูลกัน โดยสามารถเลือกได้ทั้งหมด 125 ช่อง (0-124) โดยทั้งฝ่ายรับและฝ่ายส่งต้องเลือกช่องความถี่เดียวกัน ถึงจะสามารถติดต่อกันได้

3.6 การชาร์ตแบตตารี่

จากการทดลองได้ทำการชาร์ตไฟที่แรงดันต่างกัน เวลาที่ใช้จะต่างกัน ดังแสดงในตาราง 3.1

ตารางที่ 3.1 แสดงเวลาในการชาร์ตแบตตารี่ที่ใช้แรงดันต่างกัน

แรงดันที่ใช้ (V)	กระแสที่ใช้ (A)	เวลาในการชาร์ต (ชั่วโมง)
8	1	4
10	1	3

บทที่4

ผลการทดลอง

4.1 บทนำ

ในบทนี้เราจะกล่าวถึง การทดสอบเก็บข้อมูลค่าอุณหภูมิ การนำไปวิเคราะห์เชิงสถิติ และการหาระยะทางที่ใช้ในการส่งสัญญาณแบบไร้สาย

4.2 การทดลองตอนที่1 การทดสอบเก็บข้อมูลค่าอุณหภูมิและการนำไปวิเคราะห์เชิงสถิติ

การทดลองนี้เป็นการทดสอบเก็บค่าอุณหภูมิ แล้วบันทึกผลการเก็บข้อมูลลงใน SD/MMC Card โดยเป็นการทดสอบในช่วงเวลาที่ต่างกันเพื่อเปรียบเทียบค่าอุณหภูมิแล้วนำไปวิเคราะห์เชิงสถิติ

4.2.1 วัตถุประสงค์

1. เพื่อเป็นการทดสอบเก็บค่าอุณหภูมิบันทึกลงใน SD/MMC Card
2. เพื่อเป็นการนำค่าอุณหภูมิที่วัดได้ไปวิเคราะห์เชิงสถิติได้
3. เพื่อศึกษาช่วงเวลาในการเก็บค่าอุณหภูมิ

4.2.2 ขั้นตอนการทดลอง

1. นำเครื่องวัดอุณหภูมิเอนกประสงค์ไปวางไว้ตรงจุดที่เราต้องการทำการวัดอุณหภูมิ โดยการทดลองนี้ได้วางไว้ตรงบริเวณหลังห้อง หอพักชายสุรนิเวศ 12

รูปที่ 4.1 อุปกรณ์ที่ใช้ในการทดลอง

2. ทำการต่อขั้วบวกและขั้วลบของแบตตารี่เข้ากับบอร์ดวัดอุณหภูมิเอนกประสงค์

รูปที่ 4.2 การวัดค่าอุณหภูมิ

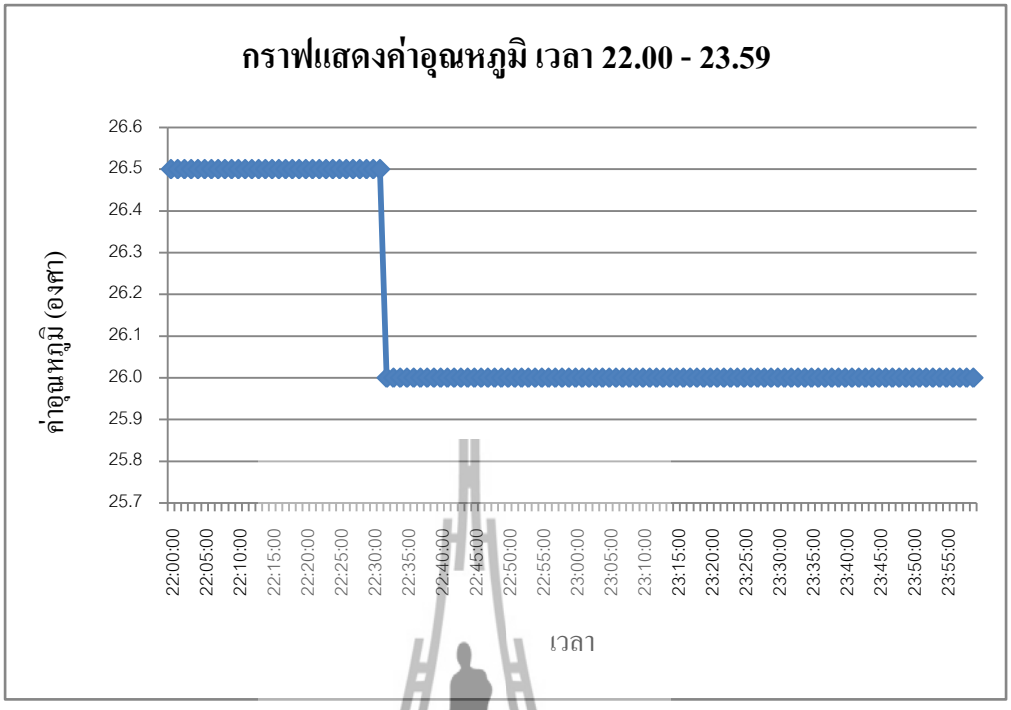
3. ทำการเก็บข้อมูลเป็นเวลา 2 ชั่วโมง , 3 ชั่วโมง และ 4 ชั่วโมง ตามลำดับ
4. เมื่อได้ข้อมูลตามเวลาที่เราต้องการ นำเมมเมอร์รี่การ์ดมาเสียบต่อเข้ากับคอมพิวเตอร์ เพื่อทำการดึงข้อมูลมาทำกราฟ และบันทึกค่าเป็นตารางเพื่อใช้เป็นข้อมูลสถิติ

- **ตารางที่ 4.1** การทดสอบเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 22.00 น. ถึง 23.59 น. เป็นเวลา (2 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ12

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
22:00:00	26.5	22:30:00	26.5	23:00:00	26.0	23:30:00	26.0
22:01:00	26.5	22:31:00	26.5	23:01:00	26.0	23:31:00	26.0
22:02:00	26.5	22:32:00	26.0	23:02:00	26.0	23:32:00	26.0
22:03:00	26.5	22:33:00	26.0	23:03:00	26.0	23:33:00	26.0
22:04:00	26.5	22:34:00	26.0	23:04:00	26.0	23:34:00	26.0
22:05:00	26.5	22:35:00	26.0	23:05:00	26.0	23:35:00	26.0
22:06:00	26.5	22:36:00	26.0	23:06:00	26.0	23:36:00	26.0
22:07:00	26.5	22:37:00	26.0	23:07:00	26.0	23:37:00	26.0
22:08:00	26.5	22:38:00	26.0	23:08:00	26.0	23:38:00	26.0
22:09:00	26.5	22:39:00	26.0	23:09:00	26.0	23:39:00	26.0
22:10:00	26.5	22:40:00	26.0	23:10:00	26.0	23:40:00	26.0
22:11:00	26.5	22:41:00	26.0	23:11:00	26.0	23:41:00	26.0
22:12:00	26.5	22:42:00	26.0	23:12:00	26.0	23:42:00	26.0
22:13:00	26.5	22:43:00	26.0	23:13:00	26.0	23:43:00	26.0
22:14:00	26.5	22:44:00	26.0	23:14:00	26.0	23:44:00	26.0
22:15:00	26.5	22:45:00	26.0	23:15:00	26.0	23:45:00	26.0

- **ตารางที่ 4.1** การทดสอบเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 22.00 น. ถึง 23.59 น. เป็นเวลา (2 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12 (ต่อ)

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
22:16:00	26.5	22:46:00	26.0	23:16:00	26.0	23:46:00	26.0
22:17:00	26.5	22:47:00	26.0	23:17:00	26.0	23:47:00	26.0
22:18:00	26.5	22:48:00	26.0	23:18:00	26.0	23:48:00	26.0
22:19:00	26.5	22:49:00	26.0	23:19:00	26.0	23:49:00	26.0
22:20:00	26.5	22:50:00	26.0	23:20:00	26.0	23:50:00	26.0
22:21:00	26.5	22:51:00	26.0	23:21:00	26.0	23:51:00	26.0
22:22:00	26.5	22:52:00	26.0	23:22:00	26.0	23:52:00	26.0
22:23:00	26.5	22:53:00	26.0	23:23:00	26.0	23:53:00	26.0
22:24:00	26.5	22:54:00	26.0	23:24:00	26.0	23:54:00	26.0
22:25:00	26.5	22:55:00	26.0	23:25:00	26.0	23:55:00	26.0
22:26:00	26.5	22:56:00	26.0	23:26:00	26.0	23:56:00	26.0
22:27:00	26.5	22:57:00	26.0	23:27:00	26.0	23:57:00	26.0
22:28:00	26.5	22:58:00	26.0	23:28:00	26.0	23:58:00	26.0
22:29:00	26.5	22:59:00	26.0	23:29:00	26.0	23:59:00	26.0



รูปที่ 4.3 กราฟแสดงผลการวัดอุณหภูมิช่วงเวลา 22.00 น. – 23.59 น. ของวันที่ 25 สิงหาคม พ.ศ. 2553

- **ตารางที่ 4.2** การทดสอบเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 14.00 น. ถึง 16.59 น. เป็นเวลา (3 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ12

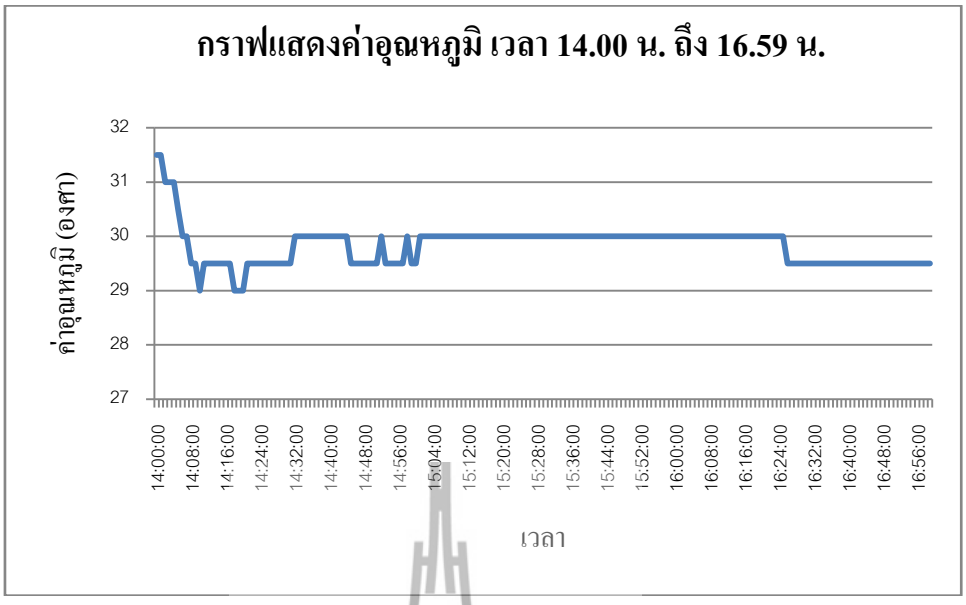
เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
14:00:00	31.5	14:45:00	29.5	15:30:00	30.0	16:15:00	30.0
14:01:00	31.5	14:46:00	29.5	15:31:00	30.0	16:16:00	30.0
14:02:00	31.0	14:47:00	29.5	15:32:00	30.0	16:17:00	30.0
14:03:00	31.0	14:48:00	29.5	15:33:00	30.0	16:18:00	30.0
14:04:00	31.0	14:49:00	29.5	15:34:00	30.0	16:19:00	30.0
14:05:00	30.5	14:50:00	29.5	15:35:00	30.0	16:20:00	30.0

- **ตารางที่ 4.2** การทดสอบเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 14.00 น. ถึง 16.59 น. เป็นเวลา (3 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12 (ต่อ)

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
14:06:00	30.0	14:51:00	29.5	15:36:00	30.0	16:21:00	30.0
14:07:00	30.0	14:52:00	30.0	15:37:00	30.0	16:22:00	30.0
14:08:00	29.5	14:53:00	29.5	15:38:00	30.0	16:23:00	30.0
14:09:00	29.5	14:54:00	29.5	15:39:00	30.0	16:24:00	30.0
14:10:00	29.0	14:55:00	29.5	15:40:00	30.0	16:25:00	30.0
14:11:00	29.5	14:56:00	29.5	15:41:00	30.0	16:26:00	29.5
14:12:00	29.5	14:57:00	29.5	15:42:00	30.0	16:27:00	29.5
14:13:00	29.5	14:58:00	30.0	15:43:00	30.0	16:28:00	29.5
14:14:00	29.5	14:59:00	29.5	15:44:00	30.0	16:29:00	29.5
14:15:00	29.5	15:00:00	29.5	15:45:00	30.0	16:30:00	29.5
14:16:00	29.5	15:01:00	30.0	15:46:00	30.0	16:31:00	29.5
14:17:00	29.5	15:02:00	30.0	15:47:00	30.0	16:32:00	29.5
14:18:00	29.0	15:03:00	30.0	15:48:00	30.0	16:33:00	29.5
14:19:00	29.0	15:04:00	30.0	15:49:00	30.0	16:34:00	29.5
14:20:00	29.0	15:05:00	30.0	15:50:00	30.0	16:35:00	29.5
14:21:00	29.5	15:06:00	30.0	15:51:00	30.0	16:36:00	29.5
14:22:00	29.5	15:07:00	30.0	15:52:00	30.0	16:37:00	29.5
14:23:00	29.5	15:08:00	30.0	15:53:00	30.0	16:38:00	29.5
14:24:00	29.5	15:09:00	30.0	15:54:00	30.0	16:39:00	29.5
14:25:00	29.5	15:10:00	30.0	15:55:00	30.0	16:40:00	29.5

- **ตารางที่ 4.2** การทดสอบเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 14.00 น. ถึง 16.59 น. เป็นเวลา (3 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12 (ต่อ)

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
14:26:00	29.5	15:11:00	30.0	15:56:00	30.0	16:41:00	29.5
14:27:00	29.5	15:12:00	30.0	15:57:00	30.0	16:42:00	29.5
14:28:00	29.5	15:13:00	30.0	15:58:00	30.0	16:43:00	29.5
14:29:00	29.5	15:14:00	30.0	15:59:00	30.0	16:44:00	29.5
14:30:00	29.5	15:15:00	30.0	16:00:00	30.0	16:45:00	29.5
14:31:00	29.5	15:16:00	30.0	16:01:00	30.0	16:46:00	29.5
14:32:00	30.0	15:17:00	30.0	16:02:00	30.0	16:47:00	29.5
14:33:00	30.0	15:18:00	30.0	16:03:00	30.0	16:48:00	29.5
14:34:00	30.0	15:19:00	30.0	16:04:00	30.0	16:49:00	29.5
14:35:00	30.0	15:20:00	30.0	16:05:00	30.0	16:50:00	29.5
14:36:00	30.0	15:21:00	30.0	16:06:00	30.0	16:51:00	29.5
14:37:00	30.0	15:22:00	30.0	16:07:00	30.0	16:52:00	29.5
14:38:00	30.0	15:23:00	30.0	16:08:00	30.0	16:53:00	29.5
14:39:00	30.0	15:24:00	30.0	16:09:00	30.0	16:54:00	29.5
14:40:00	30.0	15:25:00	30.0	16:10:00	30.0	16:55:00	29.5
14:41:00	30.0	15:26:00	30.0	16:11:00	30.0	16:56:00	29.5
14:42:00	30.0	15:27:00	30.0	16:12:00	30.0	16:57:00	29.5
14:43:00	30.0	15:28:00	30.0	16:13:00	30.0	16:58:00	29.5
14:44:00	30.0	15:29:00	30.0	16:14:00	30.0	16:59:00	29.5



รูปที่ 4.4 กราฟแสดงผลการวัดอุณหภูมิช่วงเวลา 14.00 น. – 16.59 น. ของวันที่ 25 สิงหาคม พ.ศ. 2553

- **ตารางที่ 4.3** การทดสอบเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 08.00 น. ถึง 11.59 น. เป็นเวลา (4 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
08:00	27.5	09:00	28.5	10:00	29.5	11:00	29.5
08:01	27.5	09:01	28.5	10:01	29.5	11:01	29.5
08:02	27.0	09:02	28.5	10:02	29.5	11:02	29.5
08:03	27.5	09:03	28.0	10:03	29.5	11:03	29.0
08:04	27.5	09:04	28.0	10:04	29.5	11:04	29.0
08:05	27.5	09:05	28.0	10:05	29.0	11:05	29.5
08:06	27.5	09:06	28.5	10:06	29.0	11:06	30.0
08:07	27.5	09:07	28.0	10:07	29.0	11:07	30.0

- **ตารางที่ 4.3** การเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 08.00 น. ถึง 11.59 น. เป็นเวลา (4 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12 (ต่อ)

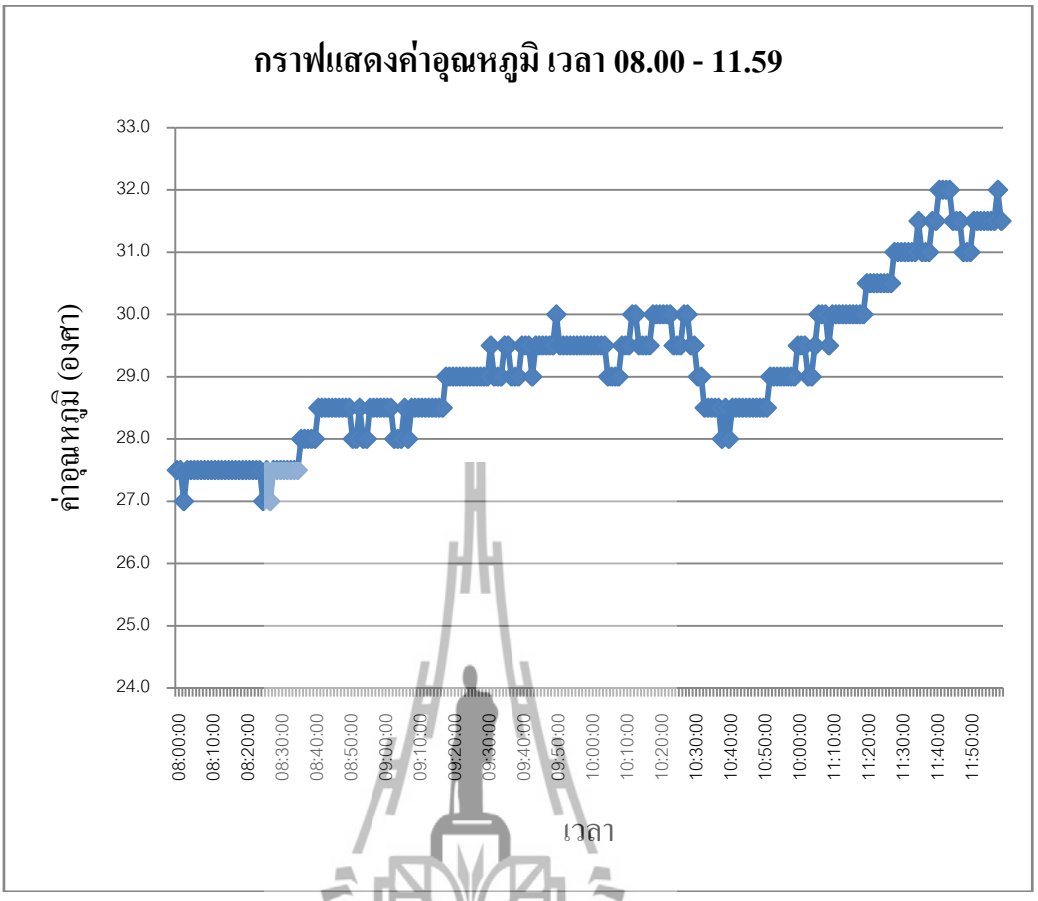
เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
08:08	27.5	09:08	28.5	10:08	29.0	11:08	30.0
08:09	27.5	09:09	28.5	10:09	29.5	11:09	29.5
08:10	27.5	09:10	28.5	10:10	29.5	11:10	30.0
08:11	27.5	09:11	28.5	10:11	29.5	11:11	30.0
08:12	27.5	09:12	28.5	10:12	30.0	11:12	30.0
08:13	27.5	09:13	28.5	10:13	30.0	11:13	30.0
08:14	27.5	09:14	28.5	10:14	29.5	11:14	30.0
08:15	27.5	09:15	28.5	10:15	29.5	11:15	30.0
08:16	27.5	09:16	28.5	10:16	29.5	11:16	30.0
08:17	27.5	09:17	28.5	10:17	29.5	11:17	30.0
08:18	27.5	09:18	29.0	10:18	30.0	11:18	30.0
08:19	27.5	09:19	29.0	10:19	30.0	11:19	30.0
08:20	27.5	09:20	29.0	10:20	30.0	11:20	30.5
08:21	27.5	09:21	29.0	10:21	30.0	11:21	30.5
08:22	27.5	09:22	29.0	10:22	30.0	11:22	30.5
08:23	27.5	09:23	29.0	10:23	30.0	11:23	30.5
08:24	27.5	09:24	29.0	10:24	29.5	11:24	30.5
08:25	27.0	09:25	29.0	10:25	29.5	11:25	30.5
08:26	27.5	09:26	29.0	10:26	29.5	11:26	30.5
08:27	27.0	09:27	29.0	10:27	30.0	11:27	30.5

- **ตารางที่ 4.3** การเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 08.00 น. ถึง 11.59 น. เป็นเวลา (4 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12 (ต่อ)

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
08:28	27.5	09:28	29.0	10:28	30.0	11:28	31.0
08:29	27.5	09:29	29.0	10:29	29.5	11:29	31.0
08:30	27.5	09:30	29.0	10:30	29.5	11:30	31.0
08:31	27.5	09:31	29.5	10:31	29.0	11:31	31.0
08:32	27.5	09:32	29.0	10:32	29.0	11:32	31.0
08:33	27.5	09:33	29.0	10:33	28.5	11:33	31.0
08:34	27.5	09:34	29.0	10:34	28.5	11:34	31.0
08:35	27.5	09:35	29.5	10:35	28.5	11:35	31.5
08:36	28.0	09:36	29.5	10:36	28.5	11:36	31.0
08:37	28.0	09:37	29.0	10:37	28.5	11:37	31.0
08:38	28.0	09:38	29.0	10:38	28.0	11:38	31.0
08:39	28.0	09:39	29.0	10:39	28.5	11:39	31.5
08:40	28.0	09:40	29.5	10:40	28.0	11:40	31.5
08:41	28.5	09:41	29.5	10:41	28.5	11:41	32.0
08:42	28.5	09:42	29.5	10:42	28.5	11:42	32.0
08:43	28.5	09:43	29.0	10:43	28.5	11:43	32.0
08:44	28.5	09:44	29.5	10:44	28.5	11:44	32.0
08:45	28.5	09:45	29.5	10:45	28.5	11:45	31.5
08:46	28.5	09:46	29.5	10:46	28.5	11:46	31.5
08:47	28.5	09:47	29.5	10:47	28.5	11:47	31.5
08:48	28.5	09:48	29.5	10:48	28.5	11:48	31.0
08:49	28.5	09:49	29.5	10:49	28.5	11:49	31.0

- **ตารางที่ 4.3** การเก็บค่าอุณหภูมิ ณ วันที่ 25 สิงหาคม พ.ศ. 2553 เวลา 08.00 น. ถึง 11.59 น. เป็นเวลา (4 ชั่วโมง) สถานที่ หลังห้องพัก หอพักสุรนิเวศ 12 (ต่อ)

เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)	เวลา	ค่าอุณหภูมิ (°C)
08:50	28.5	09:50	30.0	10:50	28.5	11:50	31.0
08:51	28.0	09:51	29.5	10:51	28.5	11:51	31.5
08:52	28.0	09:52	29.5	10:52	29.0	11:52	31.5
08:53	28.5	09:53	29.5	10:53	29.0	11:53	31.5
08:54	28.0	09:54	29.5	10:54	29.0	11:54	31.5
08:55	28.0	09:55	29.5	10:55	29.0	11:55	31.5
08:56	28.5	09:56	29.5	10:56	29.0	11:56	31.5
08:57	28.5	09:57	29.5	10:57	29.0	11:57	31.5
08:58	28.5	09:58	29.5	10:58	29.0	11:58	32.0
08:59	28.5	09:59	29.5	10:59	29.0	11:59	31.5



รูปที่ 4.5 กราฟแสดงผลการวัดอุณหภูมิช่วงเวลา 08.00 น. – 11.59 น. ของวันที่ 25 สิงหาคม พ.ศ. 2553

4.2.3 วิเคราะห์ผลการทดลองตอนที่ 1

จากการทดลองในวันที่ 25 สิงหาคม พ.ศ.2553 ได้ทำการเก็บข้อมูลแบ่งเป็น 3 ช่วง คือช่วง 2 ชั่วโมง 3 ชั่วโมง 4 ชั่วโมง ตามลำดับ โดยทำการเก็บข้อมูลทุกๆ 1 นาที เพื่อความละเอียดของข้อมูลที่พอเหมาะสำหรับเราต้องการ

จากการวัดค่าอุณหภูมิในช่วงเวลา 08.00 น. ถึง 11.59 น.เป็นเวลา 4 ชั่วโมง ช่วงเวลา 08.00 น. ค่าอุณหภูมิอยู่ที่ 27.5 องศาเซลเซียส แต่เมื่อถึงเวลา 11.59 น. อุณหภูมิสูงขึ้นถึง 31.5 องศาเซลเซียส ซึ่งในช่วงนี้เราจะรู้ได้ว่า ในตอนเช้าอากาศจะเย็นสบาย แต่เมื่อถึงช่วงเที่ยง อากาศเริ่มร้อนขึ้นถึง 4 องศาเซลเซียส

จากการวัดค่าอุณหภูมิในช่วงเวลา 14.00 น. ถึง 16.59 น. รวมเป็นเวลา 3 ชั่วโมง จากผลการทดลองที่ได้ ในช่วงเวลาดังกล่าว 14.00 น. ค่าอุณหภูมิ อยู่ที่ 31.5 องศาเซลเซียส เมื่อเวลา 16.59 น.

ค่าอุณหภูมิมีค่าลดลงไป 2 องศาเซลเซียส คือค่า 29.5 องศาเซลเซียส ซึ่งแสดงได้ว่าในเวลาที่ค่ำลงจะมีอากาศที่เย็นลง แต่ในเวลาดังกล่าว ค่าอุณหภูมิลดลงไป 2 องศาเซลเซียส

จากการวัดค่าอุณหภูมิในช่วงเวลา 22.00 น. ถึง 23.59 น. รวมเป็นเวลา 2 ชั่วโมง จากผลการทดลองที่ได้ ในช่วงเวลาดังกล่าว 22.00 น. ถึง 22.59 น. ค่าอุณหภูมิ อยู่ที่ 26.5 องศาเซลเซียส เมื่อเวลา 23.00 น. ถึง 23.59 น. ค่าอุณหภูมิมีค่าลดลงไป 0.5 องศาเซลเซียส คือค่า 26.0 องศาเซลเซียส ซึ่งแสดงได้ว่าในเวลาที่ดึกขึ้น ใกล้เที่ยงคืนมากขึ้น จะมีอากาศที่เย็นลง แต่ในเวลาดังกล่าว ค่าอุณหภูมิลดลงไป 0.5 องศาเซลเซียส อาจทำให้เราไม่รู้สึกถึงอากาศที่เปลี่ยนแปลงไปได้

การวิเคราะห์พื้นที่ความจุของ SD/MMC การ์ดที่ใช้ในการเก็บข้อมูล โดยในที่นี้จะอ้างอิงในการเก็บบันทึกข้อมูลลงใน SD/MMC การ์ด ทุกๆ 1 นาที โดยภายใน 1 วัน จะใช้พื้นที่ในการเก็บข้อมูลประมาณ 40 kbyte จะได้จำนวนวันการใช้งานตามตารางที่ 4.4

- **ตารางที่ 4.4** ตารางแสดงจำนวนวันที่ใช้งานของ SD/MMC การ์ด ตามพื้นที่ความจุของ SD/MMC การ์ด

ความจุของ SD/MMC การ์ด	จำนวนวัน (วัน)	จำนวนปี (ปี)
1 Gbyte	26214	71
2 Gbyte	52428	143
4 Gbyte	104857	287
8 Gbyte	209715	574

4.2.4 สรุปผลการทดลองตอนที่ 1

จากการทดลองในช่วง 4 ชั่วโมง คือช่วงเช้าถึงเที่ยงค่าอุณหภูมิจะเพิ่มขึ้น แต่เมื่อช่วงบ่ายถึงช่วงเย็นค่าอุณหภูมิมีค่าลดลงถึง 29.5 องศาเซลเซียส จนถึงเวลาช่วงค่ำ เริ่มจาก 22.00 น. ซึ่งอุณหภูมิมีค่าลดลงไปจากเดิมถึง 26.0 องศาเซลเซียส ซึ่งอากาศในวันนี้ ช่วงเช้ากับ กับช่วงเวลาค่ำๆ อากาศจะเย็นสบาย แต่ในช่วงเที่ยงอากาศจะร้อน

จากตารางที่ 4.4 สามารถสรุปได้ดังนี้ คือ ถ้าพื้นที่ความจุของ SD/MMC การ์ดมากจะสามารถเก็บบันทึกข้อมูลได้มากขึ้น

4.3 การทดลองตอนที่ 2 การหาระยะทางที่ใช้ในการส่งสัญญาณแบบไร้สาย

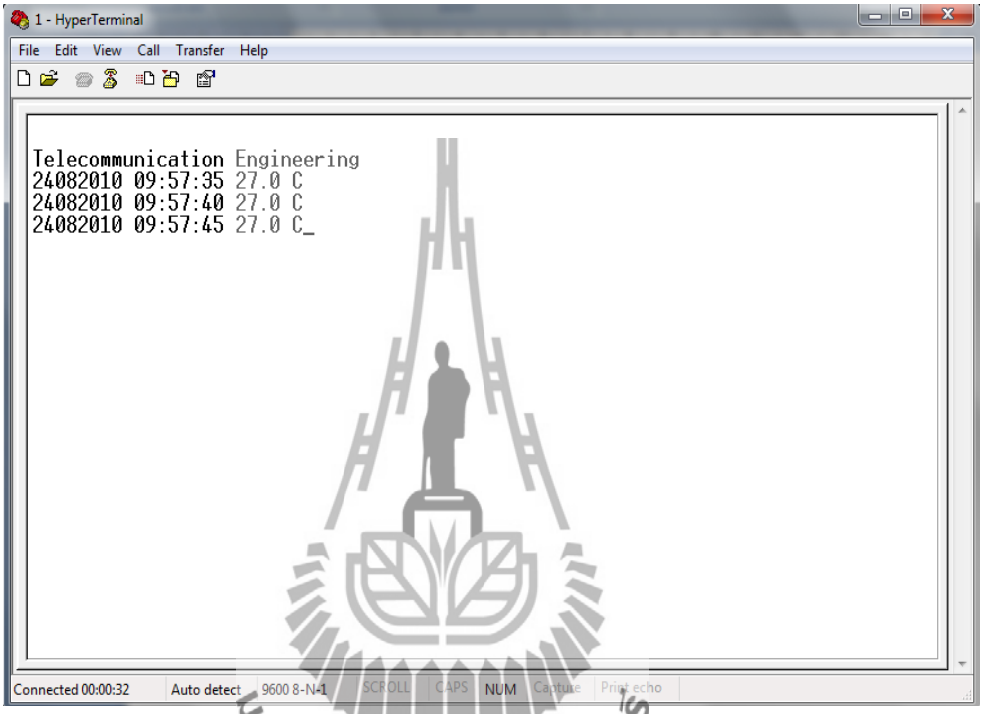
4.3.1 วัตถุประสงค์

เพื่อเป็นการหาระยะทางที่ใช้ในการส่งสัญญาณแบบไร้สาย

4.3.2 ขั้นตอนการทดลอง

1. นำอุปกรณ์เครื่องวัดอุณหภูมิเอนกประสงค์ไปวางไว้ตรงจุดที่เราต้องการทำการวัดสัญญาณ โดยการทดลองนี้ได้ไปทดสอบวัดสัญญาณบริเวณสนามฟุตบอลสนาม 1 และภายในอาคารหอพักสุรนิเวศ 12

2. เปิดโปรแกรม Hyperterminal เพื่อดูความถูกต้องของข้อมูล
3. เดินออกจากเครื่องวัดอุณหภูมิไปเป็นระยะทางทุกๆ 10 เมตร แล้วดูที่โปรแกรม Hyperterminal ว่าข้อมูลเกิดการผิดพลาดหรือไม่ ทำไปเรื่อยๆ จนกว่าข้อมูลจะเกิดการผิดพลาด แล้วหยุดทำการทดลอง



รูปที่ 4.6 โปรแกรม Hyperterminal

ตารางที่ 4.5 ระยะทางที่ใช้ในการส่งสัญญาณแบบไร้สาย

สถานที่	ระยะทาง (เมตร)
สนามฟุตบอลสนาม 1	150
หอพักสุรนิเวศ 12	60

4.3.3 วิเคราะห์ผลการทดลองตอนที่ 2

จากการทดลองจะพบว่า เมื่อเราได้เดินออกห่างจากเครื่องวัดอุณหภูมิเอนกประสงค์ในบริเวณที่เป็นที่โล่ง สัญญาณแบบไร้สายจะสามารถส่งได้ในระยะประมาณ 150 เมตร ส่วนในบริเวณที่เป็นอาคารจะสามารถส่งได้ประมาณ 60 เมตร อาจเป็นเพราะภายในอาคารมีสิ่งกีดขวางทำให้ไม่สามารถส่งสัญญาณไร้สายได้ในระยะไกลๆ

4.3.4 สรุปผลการทดลองตอนที่ 2

ในการส่งสัญญาณแบบไร้สายนั้น ในบริเวณที่เป็นที่โล่งจะสามารถส่งสัญญาณได้ไกลกว่าบริเวณที่เป็นอาคาร

บทที่ 5
สรุปผลการทดสอบและข้อเสนอแนะ

5.1 บทนำ

เนื้อหาในบทนี้จะเป็นการสรุปผลที่ได้จากการทดสอบทั้งหมด ว่าอุณหภูมิในช่วงหนึ่งวันเป็นอย่างไร และสามารถนำผลการทดสอบไปใช้ในด้านต่างๆได้ รวมไปถึงการศึกษาผลกระทบของการการเก็บค่าอุณหภูมิ

5.2 สรุปผลการทดสอบ

โครงงานนี้ได้ทำการศึกษา ถึงการเก็บค่าอุณหภูมิลง SD/MMC การ์ดและการส่งข้อมูลทางการสื่อสารแบบไร้สาย เพื่อให้ทราบถึงประโยชน์ของการเก็บบันทึกข้อมูลโดยใช้เครื่องมือในการช่วยเก็บบันทึก แทนการเก็บบันทึกด้วยตัวบุคคล อีกทั้งยังช่วยประหยัดบุคลากรในการเก็บบันทึกค่าอุณหภูมิ

จากการทดลอง การทดสอบการเก็บค่าอุณหภูมิ จะพบว่าช่วงเวลาที่ร้อนที่สุดคือช่วงเวลาประมาณ 11.30-11.59 น. เพราะว่าช่วงเวลานี้จะมีความเข้มแสงมาก

จากการทดสอบนี้ได้ผลสรุปว่า ได้ทราบถึงค่าอุณหภูมิของในแต่ละวัน เพื่อนำไปใช้ในด้านต่างๆ เช่น ด้านการเกษตรกรเพื่อใช้วัดอุณหภูมิของโรงเพาะปลูกพืชหรือโรงเลี้ยงสัตว์ ด้านอุตสาหกรรมเพื่อใช้วัดค่าอุณหภูมิในกระบวนการผลิตที่จำเป็นที่ต้องบันถึกค่าอุณหภูมิ เป็นต้น ซึ่งจะเป็นการทำการแทนบุคคลซึ่งค่าในวันที่เราทำการทดลองวัด ในช่วงเช้ากับช่วงค่ำค่าอุณหภูมิจะต่ำ แต่ในช่วงกลางวันอุณหภูมิจะมีค่าสูง

5.3 ปัญหาและอุปสรรค

ปัญหาและอุปสรรคที่เกิดขึ้นในระหว่างทำการทดสอบสามารถสรุปได้ดังนี้

1. ปัญหาเรื่องอุปกรณ์อิเล็กทรอนิกส์แต่ละตัว เมื่อใช้ระยะเวลาในการทำการทดลองบางครั้งอุปกรณ์บางตัวเกิดความเสียหายทำให้การทดลองคาดเคลื่อน
2. ปัญหาเกี่ยวกับการสื่อสารแบบไร้สาย เมื่อเราส่งข้อมูลที่มีสิ่งกีดขวาง และส่งระยะที่ไกล ทำให้ได้ข้อมูลไม่ครบถ้วน ทำให้นำมาอ้างอิงหรือใช้ประโยชน์ของข้อมูลได้ลำบาก
3. แบตตารีที่นำว่าใช้ มีพลังงานที่น้อย 6 โวลต์ 1.2 แอมแปร์ ทำให้ไม่สามารถใช้เวลาได้นานๆ

5.4 ข้อเสนอแนะ

1. ถ้าจะเพิ่มเวลาในการใช้งาน ควรใช้แบตเตอรี่ที่มีพลังงานมากขึ้น เช่น 12โวลต์ 1.2 แอมแปร์ หรือเพิ่มระบบประจุแบตตารีผ่านเซลล์แสงอาทิตย์
2. เมื่อต้องการเก็บข้อมูลเป็นเวลาที่นานๆ ควรเพิ่มขนาดความจุของเมมเมอร์รี่การ์ด
3. เมื่อต้องการวัดอุณหภูมิที่สูงมากๆ ควรเปลี่ยนเซนเซอร์ในการวัด
4. ถ้าจำเป็นที่ต้องการเก็บข้อมูลด้วยการสื่อสารแบบไร้สายในระยะที่ไกลๆ ควรเปลี่ยนอุปกรณ์ในการสื่อสารแบบไร้สายให้มีคุณภาพมากยิ่งขึ้น

ประวัติผู้เขียน

นายรณชัย อุ่นใจ เกิดเมื่อวันที่ 26 พฤษภาคม พ.ศ. 2531 ภูมิลำเนาอยู่ที่ ตำบลหัวถนน อำเภอนางรอง จังหวัดบุรีรัมย์ สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนนางรองพิทยาคม อำเภอนางรอง จังหวัดบุรีรัมย์ ปัจจุบันเป็นนักศึกษาชั้นปีที่ 4 สาขาวิชาวิศวกรรมโทรคมนาคม สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา

นายวิริยะ ชินสมุทร เกิดเมื่อวันที่ 16 สิงหาคม พ.ศ .2531 ภูมิลำเนาอยู่ที่ แขวงคลองสองต้นนุ่น เขตลาดกระบัง กรุงเทพมหานคร สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนรัตนโกสินทร์สมโภชลาดกระบัง เขตลาดกระบัง กรุงเทพมหานคร ปัจจุบันเป็นนักศึกษาชั้นปีที่ 4 สาขาวิชาวิศวกรรมโทรคมนาคม สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา

บรรณานุกรม

[1] http://www.10logic.com/microcontroller/avr-project/programmer-notepad-winavr.html

[2] http://www.thaiembedded.com/blog/?tag=ds1820

[3] UN-SOUND ความรู้ทางด้านการแปลงโวลต์ [on line] จาก :

http://www.un-sound.com/board/index.php?topic=7817.0

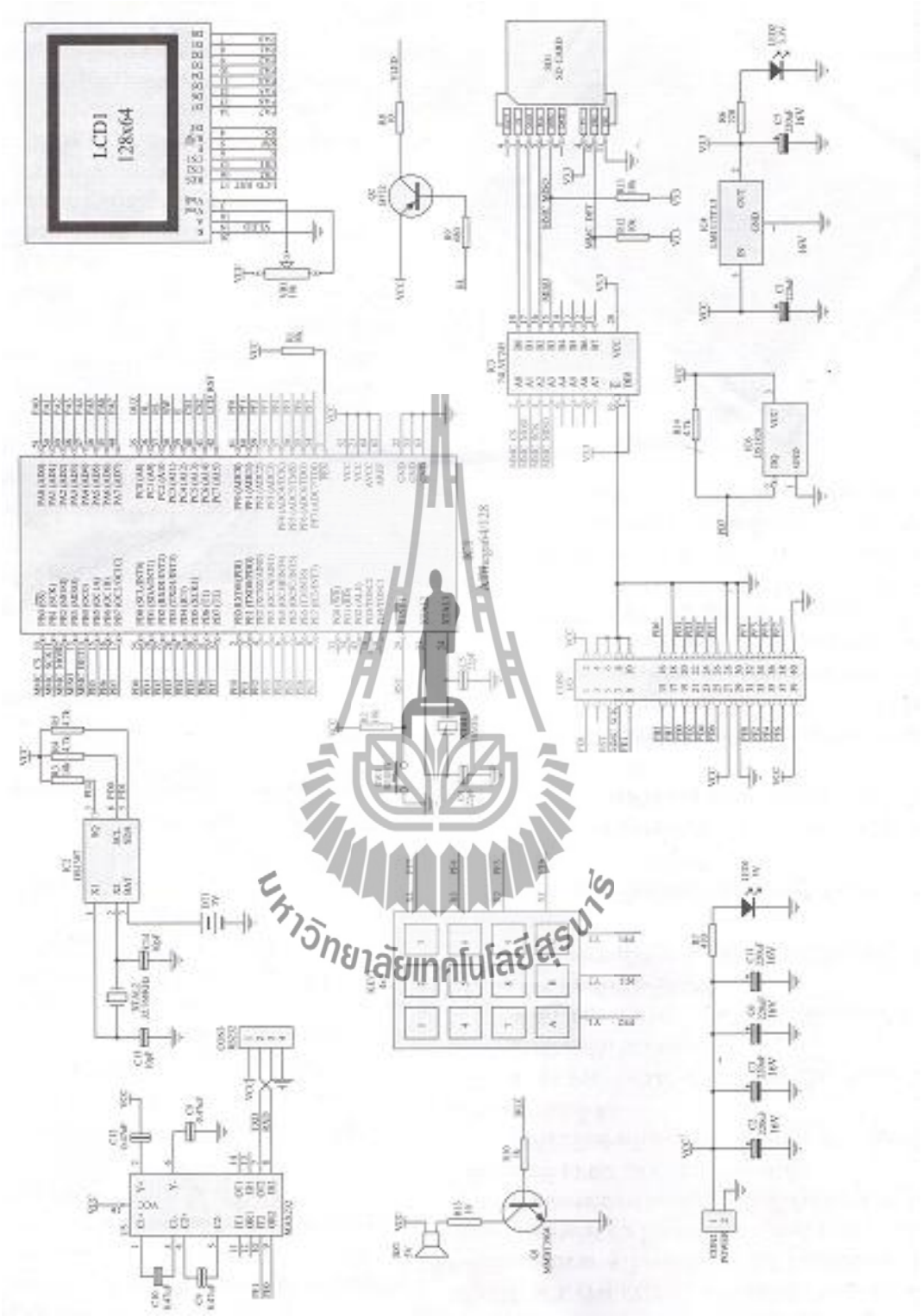
[4]บริษัทอีทีที จำกัด. คู่มือการใช้งานบอร์ด ET -AVR STAMP ATMEGA64 [on line] จาก :

http://www.ett.co.th/download/03AVR/03A08/ET-AVR%20STAMP%20ATmega64.pdf

[5] บริษัทอีทีที จำกัด. คู่มือการใช้งานบอร์ด ET-RF24G V1.0 [on line] จาก :

http://www.ett.co.th/download/12INTERFACE/12A25/MANUAL.rar

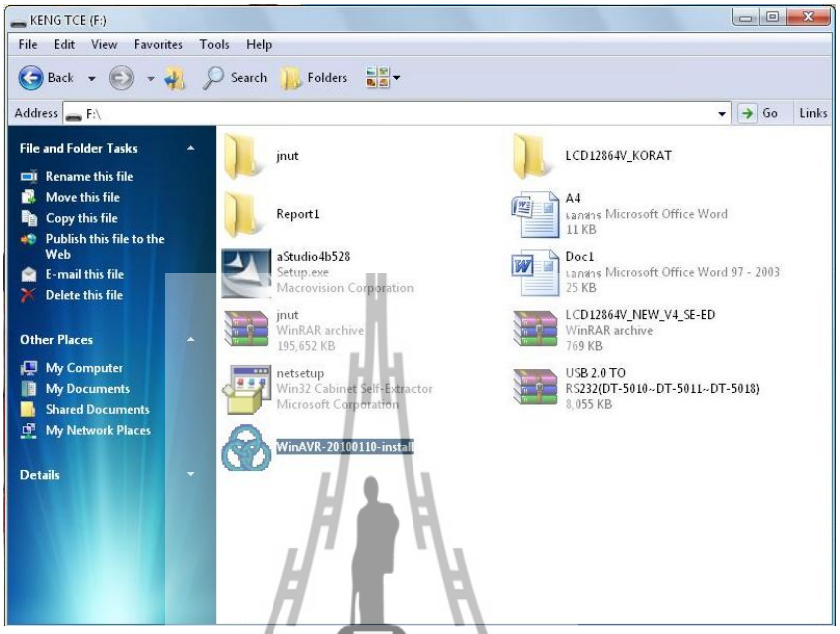
ภาคผนวก ก



รูปที่ ก.1 วงจรสมบูรณ์ของ MICROCONTROLLER ตระกูล AVR Atmega64

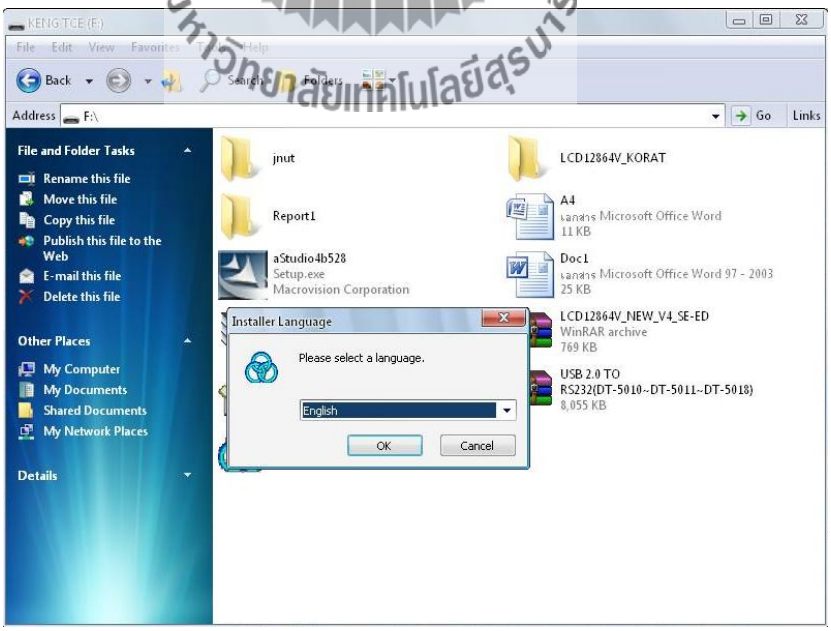
การติดตั้งโปรแกรม WinAVR

1. ดับเบิลคลิกที่ WinAvR-20100110-install.exe



รูปที่ ก.2 เลือกโปรแกรมที่ติดตั้ง

2. เลือกภาษา เสร็จแล้วคลิก OK

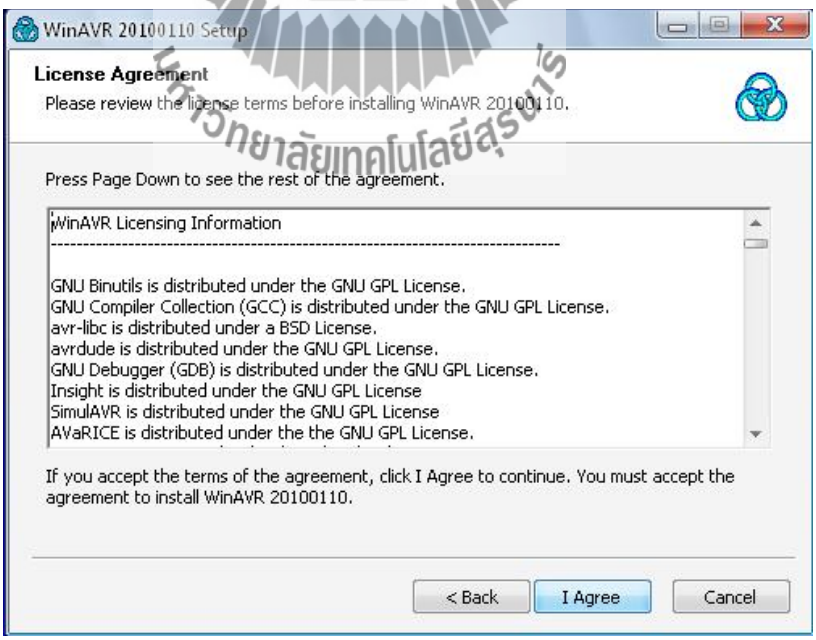


รูปที่ ก.3 เลือกภาษา

3. คลิก Next

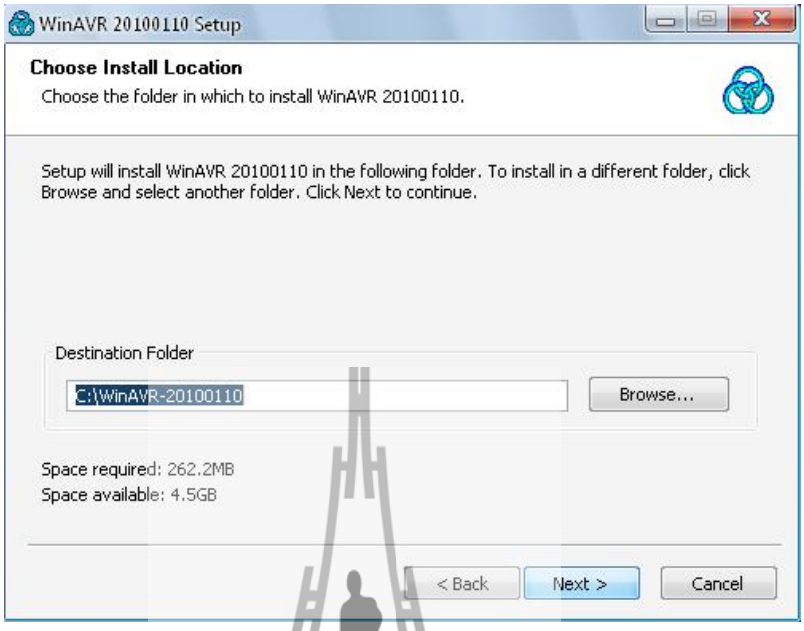
รูปที่ ก.4 หน้าต่าง Welcome to the WinAVR 20100110 setup wizard

4. คลิก I Agree



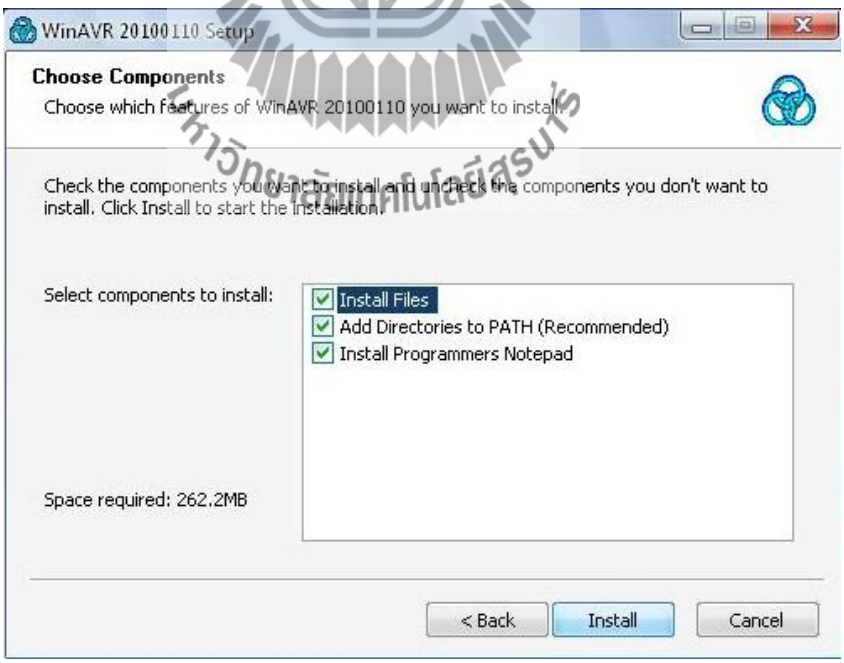
รูปที่ ก.5 หน้าต่าง License Agreement

5. เลือกที่เก็บโปรแกรม แล้วคลิก Next



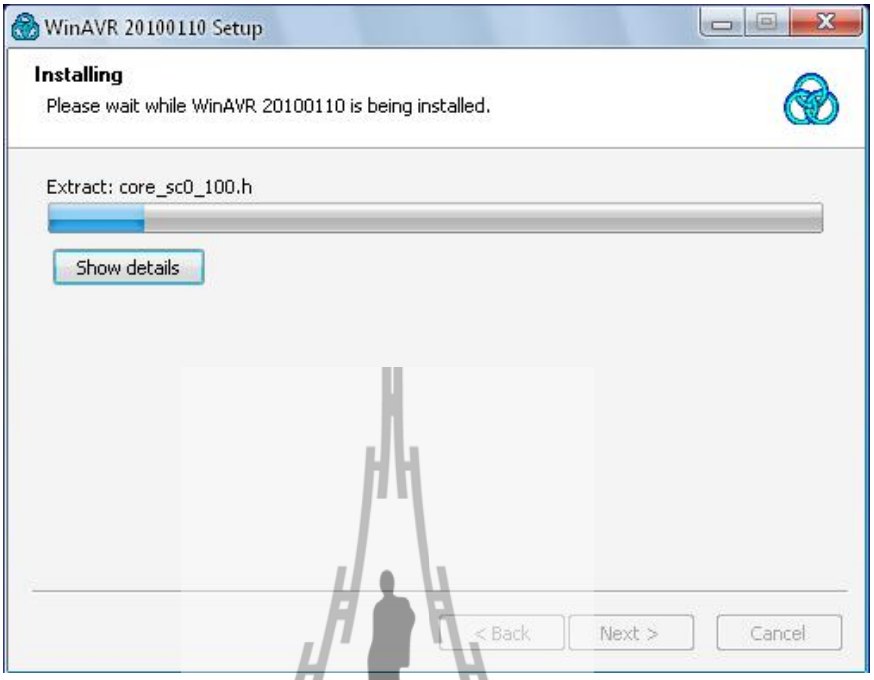
รูปที่ ก.6 หน้าต่าง Choose install location

6. คลิก Install



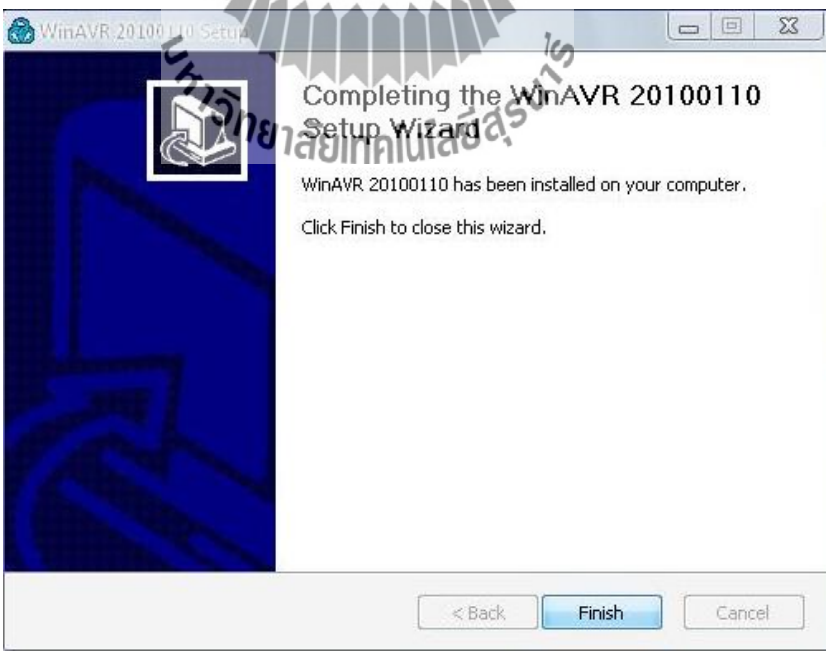
รูปที่ ก.7 หน้าต่าง Choose components

7. รอโหลดโปรแกรม



รูปที่ ก.8 หน้าต่าง Installing

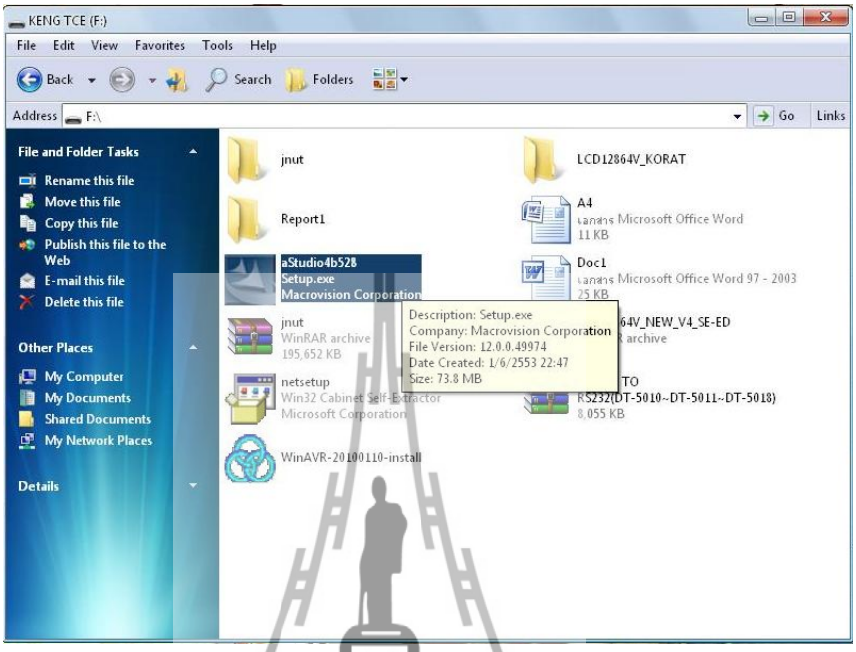
8. คลิก Finish เสร็จสิ้นการลงโปรแกรม WinAVR



รูปที่ ก.9 หน้าต่างแสดงการติดตั้งเสร็จสมบูรณ์

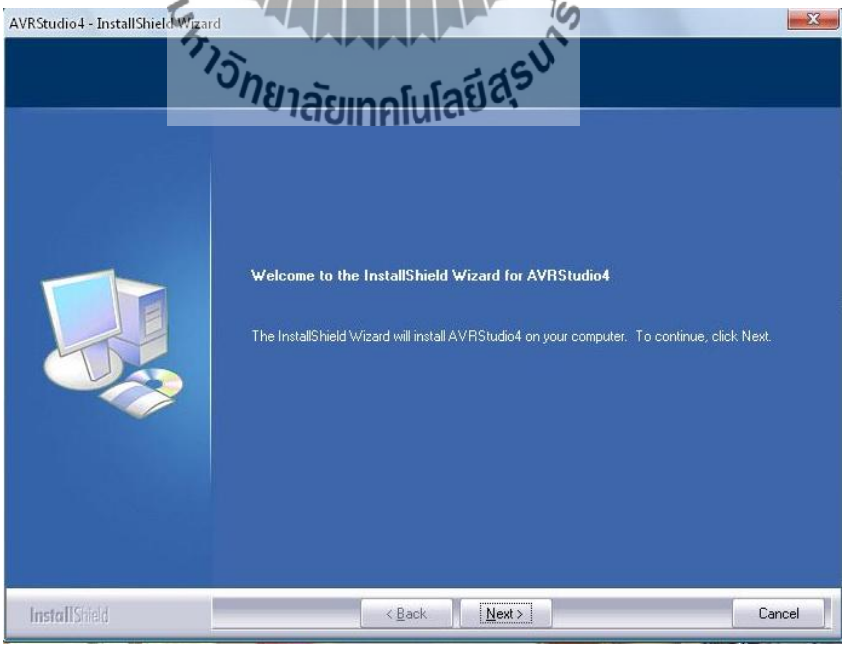
การติดตั้งโปรแกรม AVR Studio 4

1. ดับเบิลคลิกไฟล์ aStudio4b528.exe



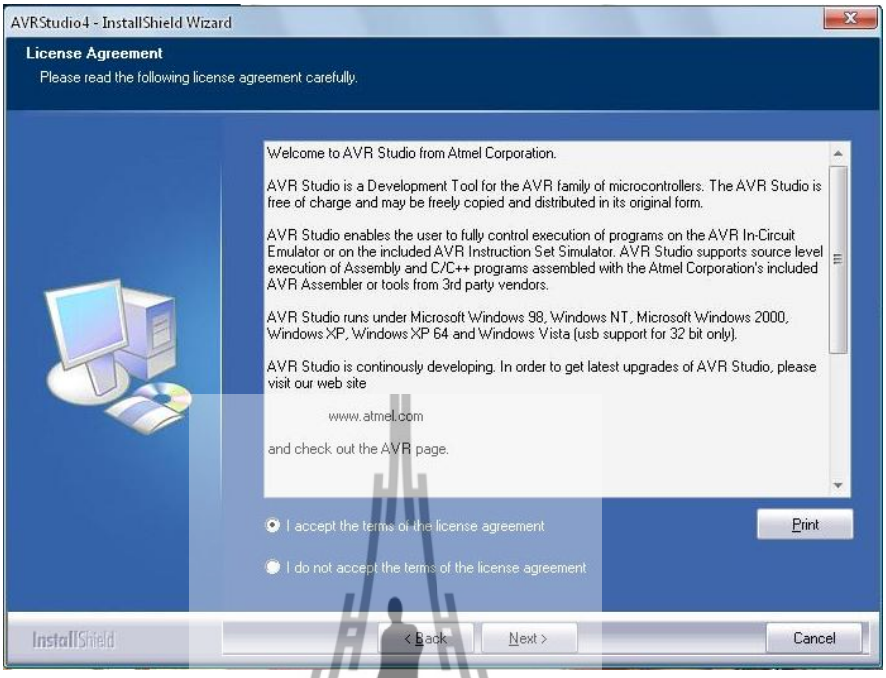
รูปที่ ก.10 เลือกโปรแกรม aStudio4b528

2. คลิก Next



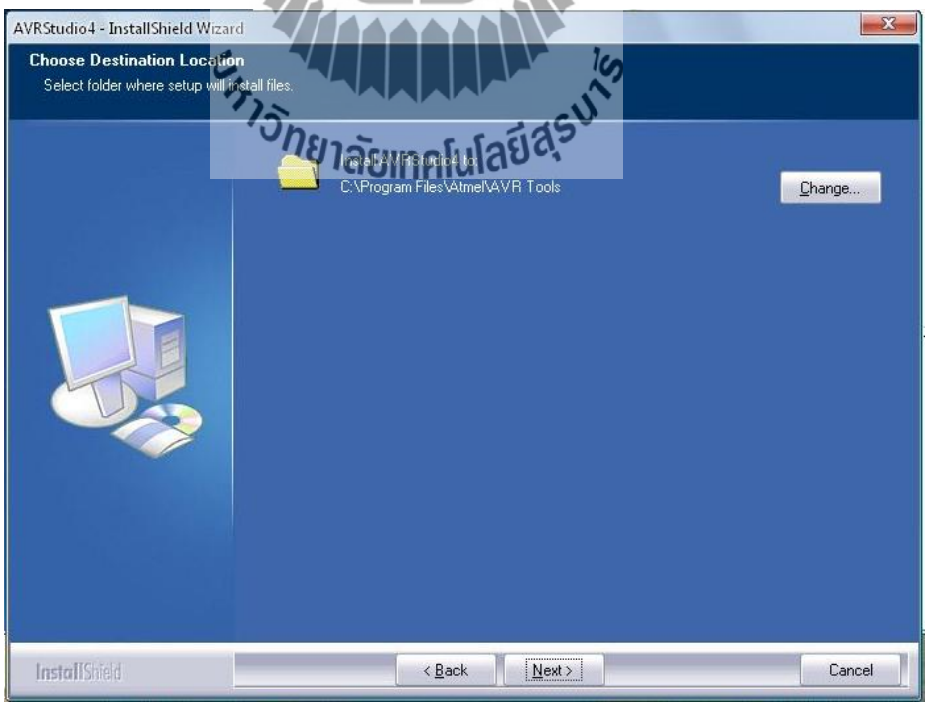
รูปที่ ก.11 หน้าต่าง Welcome to the installshield wizard for AVRStudio4

3. เลือก I accept the of the license agreement แล้วคลิก Next



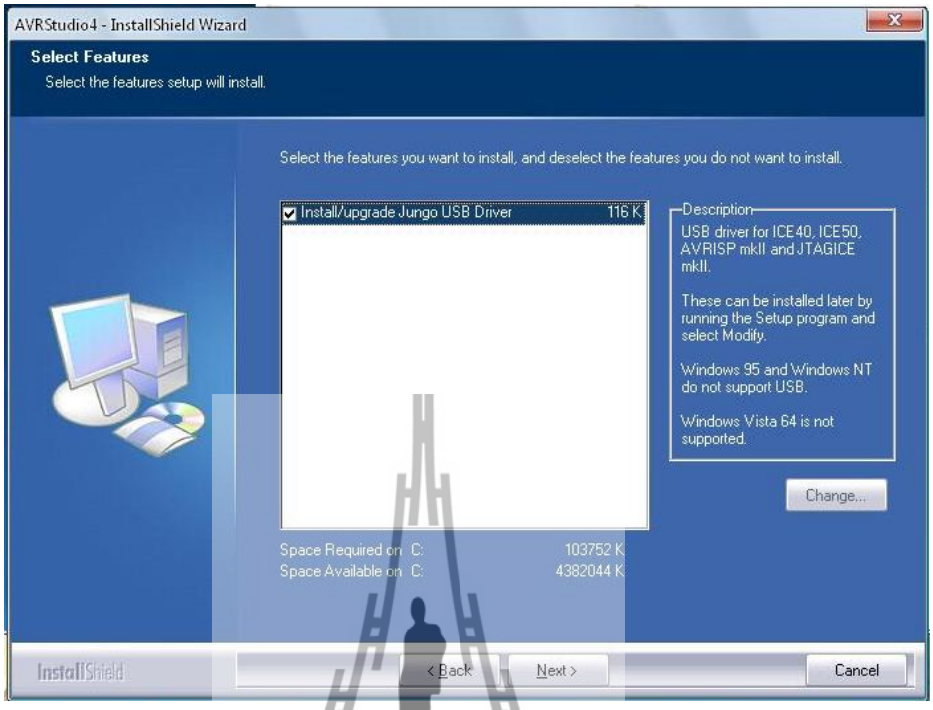
รูปที่ ก.12 หน้าต่าง License agreement

4. เลือกที่จัดเก็บโปรแกรม แล้วคลิก Next



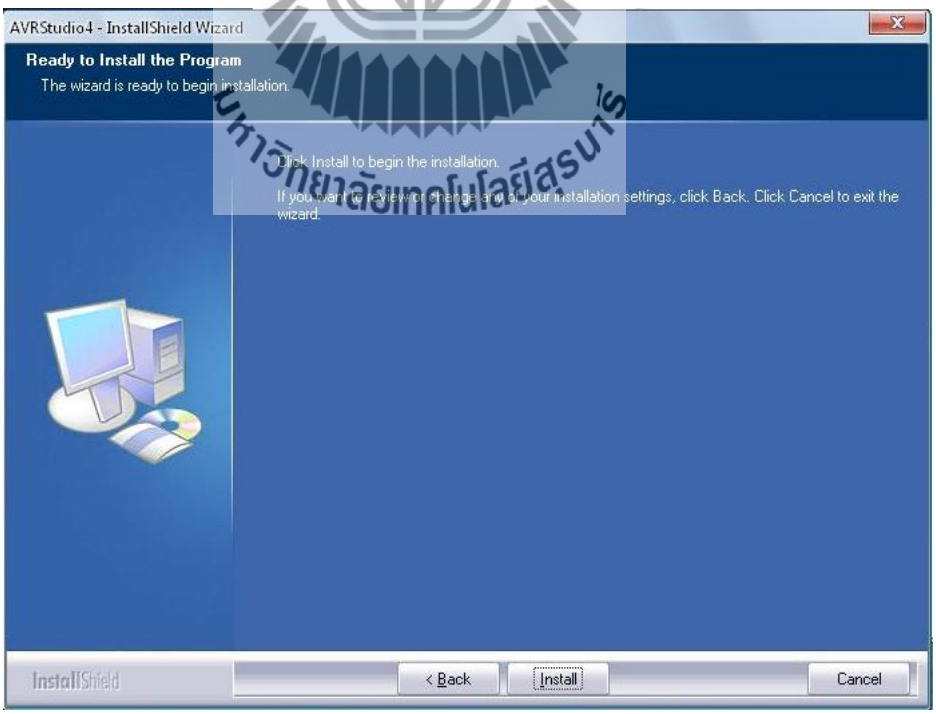
รูปที่ ก.13 หน้าต่าง Choose destination location

5. คลิก Next



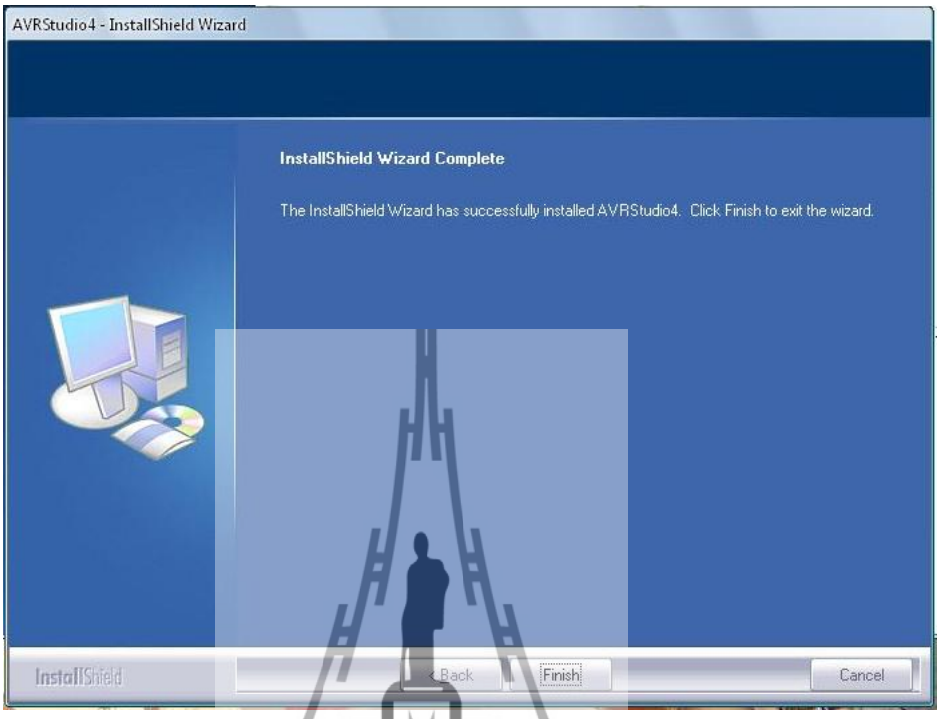
รูปที่ ก.14 หน้าต่าง Select features

6. คลิก Install



รูปที่ ก.15 หน้าต่าง Ready to install the program

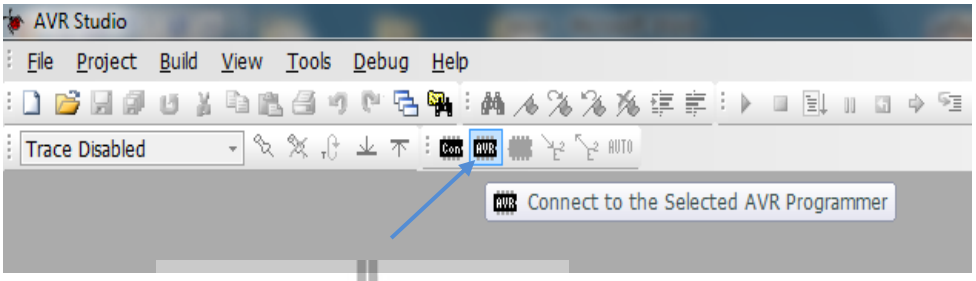
7. คลิก Finish เสร็จขั้นตอนการลงโปรแกรม AVR Studio 4



รูปที่ ก.16 หน้าต่างแสดงการเสร็จสิ้นการลงโปรแกรม AVR Studio 4

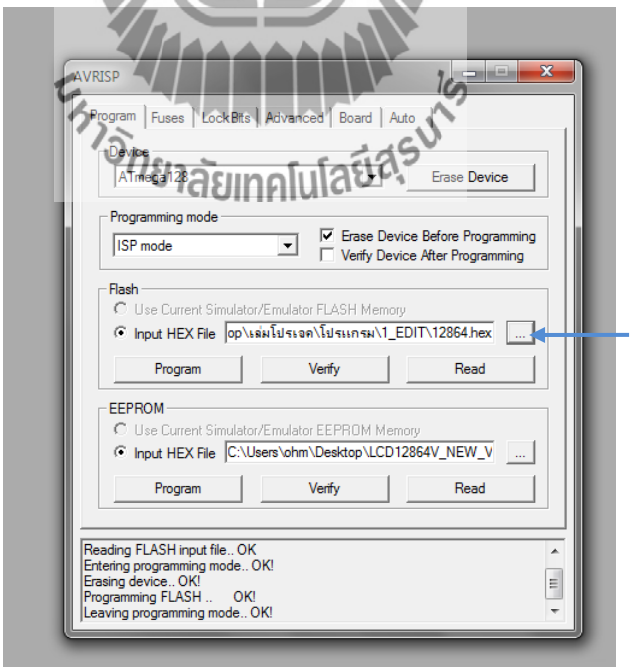
ขั้นตอนการโหลดไฟล์ HEX ลงบอร์ด

1. เปิดโปรแกรม AVR Studio 4 ขึ้นมา
2. คลิก Connect to the selected AVR programmer



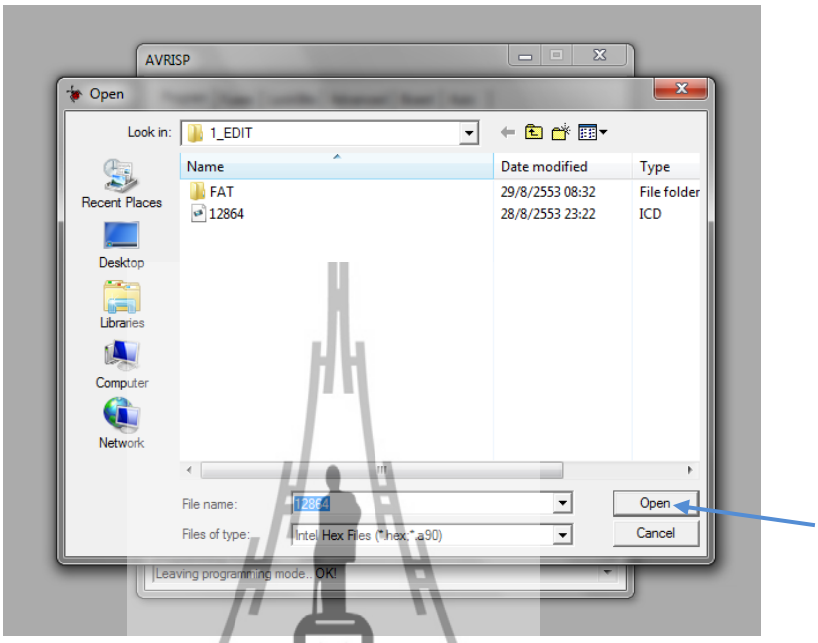
รูปที่ ก.17 การ Connect to the selected AVR programmer

3. แถว Platform เลือก STK500 or AVRISP แถว Port ให้เลือก Port ที่เราใช้ แล้วคลิก Connection
4. เลือกตามลูกศรชี้ เพื่อเปิดไฟล์ HEX



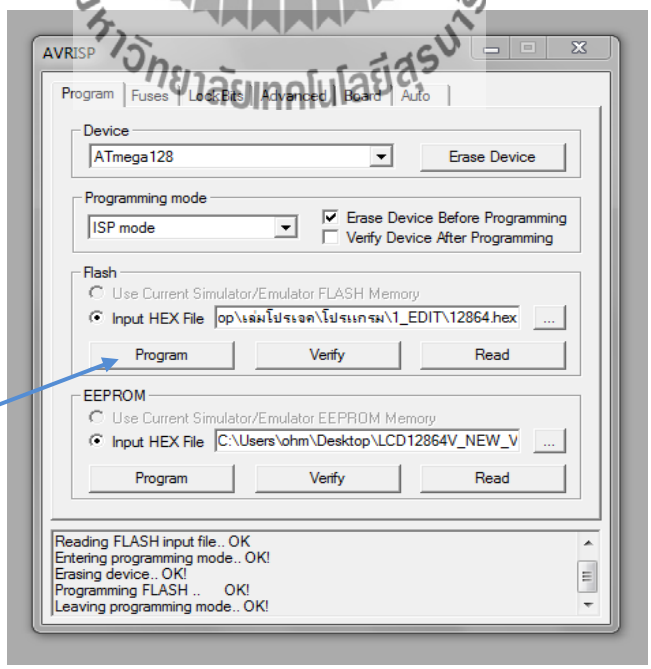
รูปที่ ก.18 วิธีการเปิดไฟล์ HEX

5. เลือกที่เก็บไฟล์ HEX



รูปที่ ก.19 แสดงการเลือกไฟล์ HEX

6. คลิก Program แล้วรอสักครู่ เสร็จการโหลดไฟล์ HEX



รูปที่ ก.20 คำสั่งในการโหลดไฟล์ HEX

ภาคผนวก ข

โค้ดโปรแกรมทั้งหมด

โค้ดโปรแกรมหลัก

```
#include        <avr/io.h>
#include        <avr/interrupt.h>
#include        <stdio.h>
#include        <stdarg.h>
#include        <string.h>
#include        <stdlib.h>
#include        <math.h>
#include        "mydefs.h"
#include        "text.h"
#include        "menu.h"
#include        "ds1820.h"
#include        "key.h"
#include        "lcd.h"
#include        "ds1307.h"
#include    "setup.h"
#include        "FAT/dos.h"
#include        "eeprom.h"
#include        "printf.h"
#include        "serial.h"
#define  DEBUG_JNUT
//-------------------------------------------------------------------------//
//GOBAL Parameter
uint8_t  flag;
uint16_t        Temperature;
```

```
uint8_t  Bar = 0;
uint8_t  CountDown;
//-------------------------------------------------------------------------//
ISR(INT4_vect)
{
        flag    |= DS1307_INT;          //Set Flag interrupt buffer after INT4 interrupt
}
//-------------------------------------------------------------------------//
void help (void)
{
        printf ("\r\n");
        printf ("\r\nTelecommunication Engineering");
        printf ("\r\n");
}
//-------------------------------------------------------------------------//
void delay_ms (unsigned long ndelay)
{
   for(;ndelay >0;ndelay--);
};
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//
uint8_t data[] = {          1,3,5,7,9,25,30,35,40,43,45,43,35,33,39,12,24,
    2,5,5,7,10,10,15,21,22,25,24,23,22,21,20,18                          ,
    17,16,15,14,13,12,11,10,9,8,7,7,7,7,8,8                          ,
    12,33,35,37,27,25,40,21,28,34,46,32,46,49,32,24                          ,

    31,34,5,37,32,32,34,34,5,5,5,6,7,9,12,24                          ,
    41,33,35,37,20,25,40,21,20,3,4,5,6,9,12,24                          ,
    31,3,35,27,28,29,45,21,22,3,44,45,43,47,44,24                          ,
```

```
    11,13,15,17,25,25,40,21,20,34,44,5,6,9,1,2                                    ,
};
//-------------------------------------------------------------------------//
void saveConfig (uint8_t type)
{
        glcd_clearScreen (0);
        glcd_Printf (32,LINE2,"Update",NORMAL,BIG_FONT,DSPL_LEFT);
        eeprom_write_byte (ADDRESS_CONFIG,type);
        beep (LONG);
        holding ();
}
//-------------------------------------------------------------------------//
void config (void)
{
        uint8_t returnKey;
        uint8_t nRet = 1;

        printf ("\r\nConfig");
        while (nRet)
        {
                returnKey = menu (main_menu,5,4);          //3 list and less than 4
 //               returnKey = menu (day_menu,7,4);          //7 = all list 4 = 1 line / page
                if (returnKey != 0)
                {
                        switch (returnKey)
                        {
                                case 1 :  //set date
                                                set_date_time(DATE);
                                                break;
```

```
                    case 2 : // Set Time
                                set_date_time (TIME);
                                break;

                    case 3:  //          Display Bar
                                Bar = BAR;
                                saveConfig (Bar);
                                break;

                    case 4:  //          Display Graph
                                Bar = DOT;
                                saveConfig (Bar);
                                break;

                    case 5:  //          exit
                                nRet = 0;
                                break;
                    default :
                                break;
                }
            }
        }
 //          then returnKey = 0; will be exit to process
}
//-------------------------------------------------------------------------//
uint8_t password (void)
{
        uint32_t        password;
        uint8_t         nRet = 0;
```

```
uint8_t keybuf[20];

memset (keybuf, 0, sizeof keybuf);
glcd_clearScreen (0);
glcd_Printf (20,LINE1,"Telecom
Eng.",NORMAL,BIG_FONT,DSPL_LEFT);
glcd_Printf
(32,LINE2,"Password",NORMAL,BIG_FONT,DSPL_LEFT);
beep (LONG);

if (!get_input_str(keybuf, 6,PASS,40,LINE3))
{
glcd_Printf
(38,32,"Cancel",NORMAL,BIG_FONT,DSPL_LEFT);
}
else
{
password = atol (keybuf);
switch (password)
{
case 1449 :
glcd_Printf
(0,LINE4, "Correct",NORMAL,BIG_FONT,DSPL_LEFT);
nRet = 1;
break;

default : glcd_Printf (0,LINE4, "Not
Correct",NORMAL,BIG_FONT,DSPL_LEFT);
break;
```

```
                                        }
                                }
                        beep (LONG);
                        holding ();
                return nRet;
}
//------------------------------------------------------------------------//
//------------------------------------------------------------------------//
int main(void)
{
//      char            lcdtmp[32];
        uint32_ttimmer;
        uint8_t         line,locate,dat,old,len;
        uint8_t  kbd;
        uint8_t  nCounter=0;
        uint8_t  rs232[50];

        DDRA  = 0xff;
        PORTA = 0xFF;                   //all output
        DDRC  = 0xff;
        PORTC = 0xff;                   //all output

        DDR_INT4        &= ~_BV(INT4);
        PORT_INT4       |=  _BV(INT4);

#ifdef MMC_CARD_SPI
        uint8_t  result = F_ERROR;
        char            card_remove = 0; //status card
        char            card_flag = 0;
        char            filename[13];
```

```
        char            f_id;                   //For file
        int                     lenght;
#endif



        init_key ();

#ifdef  RTC_1307
                TWI_init ();
                EICRB  = (1 << ISC41) | (0 << ISC40);    // INT4 falling edge interrupt
                enable_rtc ();
#endif
        ser_init ();
        sei ();
        beep (LONG);
        beep (LONG);
        Buzzer_off ();
#ifdef MMC_CARD_SPI
        MMC_IO_Init();
#endif
        help ();
#ifdef MMC_CARD_SPI
        if (!mmc_card_present ())
        {
                printf ("\r\nCard Ready....");


                if(GetDriveInformation()!= F_OK) // get drive parameters
                {
                        printf ("\n\rCard Fail");
                }
```

```
                else
                {
                        printf ("\r\nCard OK\n");
                        card_remove = 1;
                        card_flag = 1;
                }
        }
        else
        {
                printf ("\r\nNo Card in Slot\n");
                card_remove = 0;
        }
#endif
        glcd ();

//      glcd_Printf (0,0, "Arampee junyai",NORMAL,BIG_FONT,DSPL_LEFT);
        flag = 0;

        check_DS1307 (0); // 0 none update
        begin ();
        g_draw_rectangle (0,0,127,48);
        line = BcdToBin (DS1307_Read(ADDR_SEC));
        line++;    // not zero
        locate = 1;
        old = 0;

        Bar = eeprom_read_byte (ADDRESS_CONFIG);
        if ((Bar != BAR) && (Bar != DOT))
        {
                Bar = BAR;
```

```
                eeprom_write_byte (ADDRESS_CONFIG,Bar);
        }
        Temperature = read_temp ();                    //Reject 1 time
#ifdef  TEST_KEY
        printf ("\r\nTEST Key");
        uint8_t  ch;
        while (1)
        {
                ch = kbd_getc ();
                if (ch != 0)
                        printf ("%c",ch);
        }
#endif
        CountDown = TIMECOUNT;

        flag |= FLAG_COUNTDOWN;

        while (1)
        {
                        kbd = kbd_getc ();
                        if (kbd != 0)
                        {
                                disable_rtc ();
                                if (!(flag & FLAG_COUNTDOWN))
                                {
                                        Blacklight_on ();
                                }
                                if (kbd == '0')
                                {
                                        if (password ())
```

```
                    {
                        printf ("\r\nPassword Correct");
                        config ();
                    }
                    else
                        printf ("\r\nPassword not Correct");
                    begin ();
                }
                flag |= FLAG_COUNTDOWN;
                CountDown = TIMECOUNT;
                enable_rtc ();
            }
            if (flag & DS1307_INT)
            {
                timmer = display_rtc (nCounter++);
                Temperature = read_temp ();
                sprintf(rs232,"\r\n%02d%02d20%02d %02d:%02d:%02d
%02d.%01d C",
        RTC.date,RTC.month,RTC.year,
                                                    RTC.hour,RTC.min,RTC.sec,
                                                    Temperature >> 4,
(Temperature << 12)/6553);  // 0.5C
                if (RTC.sec == 0)
                printf ("%s",rs232);
    //              Write to card here
                memset (filename + 0, 0, sizeof filename);
```

```
                                    sprintf
(filename,"%02d%02d%02d",RTC.date,RTC.month,RTC.year);
                                    strcpy (filename + 6, ".txt");
   //                                 printf ("\r\n%s",filename);
                                    if (!mmc_card_present ())
                                    {
    //                                    Card Ready in Socket
    //                                    if card_remove still == 1... don't care for check
                                        if (card_remove == 0)
                                        {
                                            if(GetDriveInformation()!=F_OK) // get drive
parameters
                                            {
                                                printf ("\r\nCard Fail");
                                                card_flag = 0;
     //                                          card Fail
                                            }
                                            else
                                            {  // Card OK
                                                card_remove = 1;
                                                card_flag = 1;
      //                                                Card ok
                                            }
                                        }

                                        if (RTC.sec == 0)
                                        {
                                            if ((card_remove == 1) &&
        (card_flag==1))
                                            {
```

```
//                                                  printf ("\r\nWrite
card...%s",filename);
                                                  lenght = strlen(rs232);

                                                  f_id = Fopen(filename,F_WRITE);
                                                  if(f_id >= 0)
                                                  {
                                                        do
                                                        {
                                                        if(Fwrite(rs232,lenght,f_id)!=
1)

                                                              result = F_ERROR;
                                                        } while (result == F_OK );
                                                        Fclose(f_id);

                                                  }
                                                  }
                                                  }
                                                  else
                                                  {
                                                        card_remove = 0;

                                                  }

                                                  flag  &= ~DS1307_INT;          //set flag RTC = 0;

                                                  if (line > 125)     line = 1;
```

```
                    g_clear_vertical_line (line,1,46);
                    dat = (Temperature >> 4)/2;
                    len = dat;
//                    dat = data[locate];
//
                    if (Bar == BAR)
                            if (RTC.sec == 0)
                            g_under_vertical_line(line, 47, len);
        //Bar
                    else    // Bar == 0x02;
                            if (RTC.sec == 0)
                            g_dot_line (line, 47, len);
        //DOT
                if (line < 125)
                        g_clear_vertical_line (line + 1,1,46);
                if (RTC.sec == 0)
                line++;
                if (locate++ > 127)
                        locate = 1;
//                check flag counDown
                if (flag & FLAG_COUNTDOWN)
                {
                        if (CountDown-- < 1)
                        {
                                flag &= ~FLAG_COUNTDOWN;
                                Blacklight_off ();
                        }
                }
```

```
                        }
        }
        while (1);
}
//---------------------------------------------------------------------------//
//---------------------------------------------------------------------------//
```

2.โค้ดควบคุมเวลา

```
#ifndef  DS1307_C_
#define         DS1307_C_

#include        <avr/io.h>
#include        <avr/interrupt.h>
#include        <util/twi.h>
#include        <stdio.h>
#include        <string.h>

#include        "serial.h"
#include        "printf.h"

#include        "ds1307.h"
#include        "lcd.h"
#include        "mydefs.h"

extern uint8_t     flag;
extern uint16_t Temperature;
//---------------------------------------------------------------------------//
uint8_t TWI_Start (void)
{
```

```
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTA);

    while (!(TWCR & (1<<TWINT)));

    switch (TW_STATUS)
          {
       case      TW_START:
       case      TW_REP_START:
                                return 1;

       case      TW_MT_ARB_LOST:
                 default:
                                return 0;
    }
}
//-------------------------------------------------------------------------//
void TWI_Stop (void)
{
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
}
//-------------------------------------------------------------------------//
uint8_t TWI_Read(uint8_t ack_bit)
{
    if (ack_bit)
       TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
          else
       TWCR = (1<<TWINT)|(1<<TWEN);

    while (!(TWCR & (1<<TWINT)));
```

```
    switch (TW_STATUS)
            {
                    case TW_MR_DATA_ACK:
        case TW_MR_DATA_NACK:
                                    break;

        case TW_MR_ARB_LOST:
        default:
                                    return 0;
    }
    return(TWDR);
}
//-------------------------------------------------------------------------------//
uint8_t TWI_Write(uint8_t uc_data,uint8_t ack_bit)
{
    TWDR = uc_data;

    if (ack_bit)
        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
    else
        TWCR = (1<<TWINT)|(1<<TWEN);

    while (!(TWCR & (1<<TWINT)));

    switch (TW_STATUS) {
        case        TW_MT_SLA_ACK:
        case        TW_MT_SLA_NACK:
                                    return 1;

        case        TW_MR_SLA_ACK:
```

```
                case    TW_MR_SLA_NACK:
                                return 2;

        case        TW_MT_DATA_ACK:

        case        TW_MT_DATA_NACK:
                                return 3;

        case        TW_MT_ARB_LOST:
        default:
                                return 0;
    }
}
//-------------------------------------------------------------------------------//
void TWI_init(void)
{
 //          initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1
 //          has prescaler (mega128 & newer)
           TWSR &= ~0x03;
           TWBR = 74;
}
//-------------------------------------------------------------------------------//
uint8_t DS1307_Read(uint8_t ctl)
{
   uint8_t dat;

   TWI_Start();
   TWI_Write(TWI_SLA+TW_WRITE,1);
   TWI_Write(ctl,0);
```

```
   TWI_Start();
   TWI_Write(TWI_SLA+TW_READ,1);
   dat = TWI_Read(0);

   TWI_Stop();

   return dat;
}
//-----------------------------------------------------------------------------//
uint8_t BinToBcd(uint8_t val)
{
  return ((val / 10) *16 + val % 10);
}
//-----------------------------------------------------------------------------//
uint8_t BcdToBin(uint8_t val)
{
        val = (val >> 4) * 10 + (val & 0x0f);
        return val;
}
//-----------------------------------------------------------------------------//
void DS1307_Write(uint8_t ctl,uint8_t dat)
{
   TWI_Start();

   TWI_Write(TWI_SLA+TW_WRITE,1);
   TWI_Write(ctl,1);
   TWI_Write(dat,1);

   TWI_Stop();
}
```

```
//-------------------------------------------------------------------------//
void Read_RTC(void)
{
        RTC.sec   = BcdToBin (DS1307_Read(ADDR_SEC));
        RTC.min   = BcdToBin (DS1307_Read(ADDR_MIN));
        RTC.hour  = BcdToBin (DS1307_Read(ADDR_HOUR));

        RTC.day           = BcdToBin (DS1307_Read(ADDR_DAY));

        RTC.date  = BcdToBin (DS1307_Read(ADDR_DATE));
        RTC.month = BcdToBin (DS1307_Read(ADDR_MONTH));
        RTC.year  = BcdToBin (DS1307_Read(ADDR_YEAR));
}
//-------------------------------------------------------------------------//
void update_RTC(void)
{
  // Set Time
  DS1307_Write(ADDR_SEC, BinToBcd  (RTC.sec));
        DS1307_Write(ADDR_MIN,     BinToBcd  (RTC.min));
        DS1307_Write(ADDR_HOUR,    BinToBcd  (RTC.hour));

        DS1307_Write(ADDR_DAY,     BinToBcd  (RTC.day));
  // Set Date
  DS1307_Write(ADDR_DATE,          BinToBcd  (RTC.date));
        DS1307_Write(ADDR_MONTH,BinToBcd  (RTC.month));
        DS1307_Write(ADDR_YEAR,    BinToBcd  (RTC.year));
}
//-------------------------------------------------------------------------//
void DS1307_interrupt_config (void)
{
```

```
        DS1307_Write(ADDR_CONFIG,0x90);                 // 1Hz interrrupt
}
//-------------------------------------------------------------------------//
//Disable RTC interrupt
void disable_rtc (void)
{
        EIMSK &= ~_BV(INT4);
}
//-------------------------------------------------------------------------//
//Enable RTC interrupt
void enable_rtc (void)
{
        EIMSK |= _BV(INT4);
}
//-------------------------------------------------------------------------//
void check_DS1307 (uint8_t update)
{
        if ((BcdToBin (DS1307_Read(ADDR_SEC)) == 0x50) || update)
        {
#ifdef  DS1307_DEBUG
                printf ("\r\nNew update");
#endif

                RTC.sec = RTC.min = RTC.hour = 0;
                RTC.day   = 5;   //Friday
                RTC.date  = 01;
                RTC.month = 01;
                RTC.year  = 10;

        update_RTC ();
```

```
        }
        DS1307_interrupt_config ();
}
//-------------------------------------------------------------------------//
uint32_t display_rtc (uint8_t nCount)
 {
        char       lcdbuf[SIZEOF_LINE + 1];
        uint16_t th,tl;



        uint32_t tValue = 0;

        Read_RTC ();



        memset (lcdbuf, 0, sizeof lcdbuf);
        th = Temperature >> 4;
        tl = (Temperature << 12)/6553;
        if (!(RTC.sec == 1 || RTC.sec == 2 || RTC.sec == 3))
        {
                sprintf(lcdbuf,"%2d.%01d C %02d:%02d:%02d"   ,th,tl,

                 RTC.hour,RTC.min,RTC.sec);
        }
        else
        {
                sprintf(lcdbuf,"%2d.%01d C %02d/%02d/%02d"   ,th,tl,

                 RTC.date,RTC.month,RTC.year);
```

```
        }
        glcd_Printf(0,48, "                ",NORMAL,BIG_FONT,DSPL_LEFT);
        glcd_Printf(0,48, lcdbuf,NORMAL,BIG_FONT,DSPL_LEFT);


        tValue =    ((uint32_t)(RTC.hour) * HOUR          +
   (                                uint16_t)(RTC.min)  * MIN      +  RTC.sec);
        return  tValue;
};
#endif  // DS1307_C_
```

3. โค้ดอุณหภูมิ

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdarg.h>
#include <string.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include "mydefs.h"
#include "ds1820.h"
#include "serial.h"
#include "printf.h"

//just to reduce this delay_us function for DS1820
//j.nut game!
//---------------------------------------------------------------------------//
```

```
static inline void delayloop(uint16_t count)
{
        asm volatile (  "cp  %A0,__zero_reg__ \n\t"   \
              "cpc %B0,__zero_reg__ \n\t"  \
              "breq L_Exit_%=       \n\t"  \
              "L_LOOP_%=:           \n\t"  \
              "sbiw %0,1            \n\t"  \
              "brne L_LOOP_%=       \n\t"  \
              "L_Exit_%=:           \n\t"  \
              : "=w" (count)
    : "0"  (                                    count)
            );
}

#define DELAY_US_CONV(us) ((uint16_t)((((us)*1000L)/(1000000000/F_CPU))-1)/4))
#define delay_us(us)        delayloop(DELAY_US_CONV(us))
//-------------------------------------------------------------------------//
uint8_t DS1820_reset(void)
{
        uint8_t err;
        DS1820_OUT &= ~_BV(DS1820_PIN);
        DS1820DDR |=  _BV(DS1820_PIN);
        delay_us(480);                          // 480 us
        cli();
        DS1820DDR &=  ~_BV(DS1820_PIN);
        delay_us(66);
        err = DS1820_IN & (1<<DS1820_PIN);   // no presence detect
        sei();
        delay_us(480);
```

```
        if((DS1820_IN & (1<<DS1820_PIN)) == 0) //        short circuit
                err = 1;


 return err;
}
//---------------------------------------------------------------------//
uint8_t DS1820_bit_io (uint8_t ack)
{
        cli();
        DS1820DDR |= _BV(DS1820_PIN);
        delay_us(1);
        if(ack)
                DS1820DDR &= ~_BV(DS1820_PIN);
        delay_us(15);
        if((DS1820_IN & (1<<DS1820_PIN)) == 0)
                ack = 0;
        delay_us(60);
        DS1820DDR &= ~_BV(DS1820_PIN);
        sei();
 return ack;
}
//---------------------------------------------------------------------//
uint8_t DS1820_byte_wr (uint8_t b)
{
        uint8_t i = 8, j;
        do
        {
                j = DS1820_bit_io(b&1);
                        b >>= 1;
                if(j)
```

```
                b |= 0x80;
        } while(--i);

 return b;
}
//--------------------------------------------------------------------//
uint16_t DS1820_byte_rd (void)
{
 return DS1820_byte_wr(0xFF);
}
//--------------------------------------------------------------------//
uint8_t DS1820_rom_search (uint8_t diff, uint8_t *id)
{
        uint8_t i, j, next_diff;
        int b;
        if(DS1820_reset())
                return PRESENCE_ERR;                    // error, no device found

        DS1820_byte_wr(SEARCH_ROM);                     // ROM search command
        next_diff = LAST_DEVICE;                        // unchanged on last device
        i = 64; // 8 * 8 = 64 bit              // 8 bytes
        do
        {
                j = 8;                                  // 8 bits
                do
                {
                        b = DS1820_bit_io(1);           // read bit
                        if(DS1820_bit_io(1))
                        {                               // read complement bit
                                if(b)                           // 11
```

```
                    return DATA_ERR;    // data error
                }
                else
                {
                    if(!b)
                    {                                   // 00 = 2 devices
                        if( diff > i || ((*id & 1) && diff != i) )
                        {
                            b = 1;                      // now
1
                            next_diff = i;              // next pass 0
                        }
                    }
                }
                DS1820_bit_io(b);                       // write bit
  *                 id >>= 1;
                if(b)                                   // store bit
   *                    id |= 0x80;
                i--;
            } while(--j);
   id++;                                    // next byte
  } while(i);


  return next_diff;                                 // to continue search
}
//------------------------------------------------------------------------//
void DS1820_command (uint8_t command, uint8_t *id)
{
   uint8_t i;
```

```
	DS1820_reset();

	if(id)
	{
		DS1820_byte_wr(MATCH_ROM);		// to a single device
		i = 8;
		do
		{
			DS1820_byte_wr(*id);
			id++;
		} while( --i );
	}
	else
		DS1820_byte_wr(SKIP_ROM);		// to all devices

	DS1820_byte_wr(command);
}
//------------------------------------------------------------------//
void DS1820_start (void)
{
	if (DS1820_IN & 1<< DS1820_PIN)
	{
		DS1820_command(CONVERT_T, NULL);
		DS1820_OUT |= _BV(DS1820_PIN);
		DS1820DDR  |= _BV(DS1820_PIN);			//power on
	}
	else
		printf ("\r\nShort Circuit !");
}
//------------------------------------------------------------------//
```

```
uint16_t read_temp (void)
{
        uint8_t id[8],diff;
//        char buf[30];
        uint16_t temp = 0;
        DS1820_start ();
        for(diff = SEARCH_FIRST; diff != LAST_DEVICE;)
        {
                diff = DS1820_rom_search( diff, id );

                if(diff == PRESENCE_ERR)
                {
                        printf ("\r\nNo Sensor found");
                        break;
                }
                if(diff == DATA_ERR)
                {
                        printf ("\r\nBus Error");
                        break;
                }

                if(id[0] == 0x28 || id[0] == 0x10)
                {          // temperature sensor

                        DS1820_byte_wr( READ );                         // read command
                        temp = DS1820_byte_rd();                                // low byte
                        temp |= (uint16_t)DS1820_byte_rd() << 8;   // high byte

                        if(id[0] == 0x10)     // 9 -                                > 12 bit
                                temp <<= 3;
```

```
                }
        }
        return temp;
}
```

4. โค้ด LCD

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include "serial.h"
#include "printf.h"
#include "mydefs.h"
#include "lcd.h"
//---------------------------------------------------------------------------//
void glcd_StrobeEnable (void)
{
        LCD_CMD_PORT |= _BV(EN);                        // EN high
        _delay_us (5);
        LCD_CMD_PORT &= ~_BV(EN);
        _delay_us (5);
}
```

```
//-------------------------------------------------------------------------//
void glcd_Busy (void)
{
        LCD_DATA_DIR = 0x00;                                    //all input
        LCD_CMD_PORT |= _BV(RW);
        LCD_CMD_PORT &= ~_BV(DI);                               //DI = low;
        LCD_CMD_PORT |= _BV(EN);
        while ((LCD_DATA_PIN & 0x80) != 0x00)
                _delay_us (5);
        LCD_CMD_PORT &= ~_BV(EN);                               //EN = low;
   LCD_DATA_DIR = 0xff;
}
//-------------------------------------------------------------------------//
void glcd_WriteCommand (uint8_t cmd)
{
        glcd_Busy ();
        LCD_CMD_PORT &= ~_BV(DI);
   LCD_CMD_PORT &= ~_BV(RW);
        _delay_us (2);
        LCD_DATA_OUT = cmd;
        glcd_StrobeEnable ();
}
//-------------------------------------------------------------------------//
void glcd_WriteData (uint8_t dat)
{
        glcd_Busy ();
        LCD_CMD_PORT |= _BV(DI);
        LCD_CMD_PORT &= ~_BV(RW);                               // R/W = 0
        _delay_us (5);
        LCD_DATA_OUT = dat;
```

```
        glcd_StrobeEnable ();
}
//-------------------------------------------------------------------------//
uint8_t glcd_ReadByte (void)
{
        uint8_t dat;
                glcd_Busy ();
                LCD_DATA_OUT = 0xff;                                    //pullup
                LCD_DATA_DIR = 0x00;
                LCD_CMD_PORT |= _BV(DI) | _BV(RW) | _BV(EN);//DI & R_W & EN =
high
                _delay_us(5);
                dat = LCD_DATA_PIN;
                LCD_CMD_PORT &= ~_BV(EN);
                _delay_us(5);

        LCD_DATA_DIR = 0xff;                                    //output
   return dat;
}
//-------------------------------------------------------------------------//
void glcd_SelectChip(uint8_t page)
{
        if (!page)
        {
                LCD_CMD_PORT &= ~_BV(CSEL2);            // deselect chip 2
                LCD_CMD_PORT |= _BV(CSEL1);             // select chip 1
        }
        else
        {
                LCD_CMD_PORT &= ~_BV(CSEL1);            // deselect chip 1
```

```
                LCD_CMD_PORT |= _BV(CSEL2);                    // select chip 2
        }
        _delay_us (5);
}
//-------------------------------------------------------------------------//
void glcd_SetPage (uint8_t page)
{
        glcd_WriteCommand(GLCD_SET_PAGE + page);
}
//-------------------------------------------------------------------------//
void glcd_SetAddress (uint8_t address)
{
        glcd_WriteCommand(GLCD_SET_ADDRESS + address);
}
//-------------------------------------------------------------------------//
void glcd_gotoxy (uint8_t x, uint8_t y)
{
        uint8_t chip = 0,
                        tmp;
        if (x > 127) x = 0;
        if (y > 63)  y = 0;
                lcd.x = x; lcd.y = y;
        if ((x > 63) && (x < 128))    // 64 --- 127
        {
                chip = 1;
                x -= 64;
        }
        tmp = y/8;
        glcd_SelectChip (chip);                    // Left and Right on screen
        glcd_SetAddress (x);                       // Culume          0 - 127
```

```
        glcd_SetPage(tmp);                                  // Line        0-7
}
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//
void glcd_Init(void)
{
//page 0 0-63
        glcd_SelectChip (0);
        glcd_WriteCommand(GLCD_ON);
        glcd_WriteCommand(GLCD_DISP_START0);
//page 1 64-128
        glcd_SelectChip (1);
        glcd_WriteCommand(GLCD_ON);
        glcd_WriteCommand(GLCD_DISP_START0);
}
//-------------------------------------------------------------------------//
void glcd_SetDot (uint8_t x, uint8_t y)
{
        uint8_t dat = 0;                                    // comment just to clear

                glcd_gotoxy (x,y);
 //                comment  Oct 26,2009
 //                I don't undertand why just to read 2 time, But it work
                dat = glcd_ReadByte();                      //Read The Current location
                dat = glcd_ReadByte();                      //Read data on LCD
                dat |= (0x01 << (y % 8));
                glcd_gotoxy (x,y);
        glcd_WriteData (dat);
}
//-------------------------------------------------------------------------//
```

```
void glcd_ClearDot (uint8_t x, uint8_t y)
{
	uint8_t dat = 0;				// comment just to clear

		glcd_gotoxy (x,y);
 //		  comment  Oct 26,2009
 //		  I don't undertand why just to read 2 time, But it work
		glcd_ReadByte();				//Read The Current location
		dat = glcd_ReadByte();		//Read data on LCD
		dat &= ~(0x01 << (y % 8));	//Clear bit in Byte
		glcd_gotoxy (x,y);
	glcd_WriteData (dat);			//Write to LCD again
}
//-------------------------------------------------------------------------//
void glcd_clearScreen(uint8_t dat)
{
	uint8_t i, j;

	for(j = 0; j < 8; j++)
	{
		for (i = 0; i < 128;i++)
		{
			glcd_gotoxy(i,j*8);
			glcd_WriteData(dat);
		}
	}
}
//-------------------------------------------------------------------------//
void glcd_NewLine(uint8_t fontHeight, uint8_t offset)
{
```

```
        if (lcd.y + fontHeight < 64)
                glcd_gotoxy(offset, lcd.y + fontHeight);
        else
                glcd_gotoxy(offset, 0);
}
//-------------------------------------------------------------------------//
void glcd_PutChar(char c, struct font font)
{
        uint16_t index;
        uint8_t pages, tmp, i, j, xPos, yPos, xtmp;

        if(c == '\n')
                glcd_NewLine(font.height, 0);
        if(c < 32)
//              ignore escape characters
                return;

        xPos = lcd.x;                                   // save old coordinates
        yPos = lcd.y;
        c -= 32;

        pages = font.height/8;                                  // calculate pages

 //                      Small Font = 0;

 //                      BIG FONT        = 1 ->upper;
        if(font.height%8 != 0)
                pages++;
        index = c * font.width * pages;                 // get the needed array index
        for(j=0; j < pages; j++)
```

```
        {
                xtmp = xPos;
                for(i=j; i< font.width * pages; i += pages)
                {
                        tmp = pgm_read_byte(font.size + index + i);
                        if (lcd.inv)        // NORMAL = 0, INV = 1;
                                tmp =~ tmp;                     // Convert Data
                        glcd_gotoxy (xtmp++,lcd.y);
                        glcd_WriteData(tmp);                    // write Character-Data
                }
                glcd_gotoxy(xPos, lcd.y + 8);
        }
        glcd_gotoxy(lcd.x + font.width, yPos);                  // go to the upper right corner
}
//-------------------------------------------------------------------------//
void glcd_PutString(char *string, struct font font)
{
        uint8_t startx=lcd.x, i=0;

        char c = string[0];

        while(c != 0)
        {
                if(c == '\f')
                        glcd_clearScreen(0);
                if(c == '\n')
                        glcd_NewLine(font.height, startx);
                else
                {
                        if (lcd.number)
```

```
                {
                        if ((c >= '0') && (c <= '9'))
                        {

                                if ((c == '0') && (!lcd.flag) && lcd.lenght-- != 1)
                                        c = ' ';
                                else
                                        lcd.flag = 1;
                        }
                }
                glcd_PutChar(c, font);
        }
        c = string[++i];
    }
}
//-------------------------------------------------------------------------//
void display_image (uint8_t x, uint8_t y, const char *dat)
{
    uint8_t pages, tmp, i, j, xPos, yPos, xtmp;
    uint8_t index = 0;
    xPos = lcd.x;                                   // save old coordinates
    yPos = lcd.y;

    pages = 2;                                      // calculate pages

 //                 Small Font = 0;
    glcd_gotoxy (x,y);
    xPos = lcd.x;                                   // save old coordinates
    yPos = lcd.y;
    for(j = 0; j < pages; j++)
```

```
        {
                xtmp = xPos;
                for(i = 0; i < 24; i++)
                {
                        tmp = pgm_read_byte(&dat[index]);
                        glcd_WriteData(tmp);                    // write Character-Data
                        index++;
                }
                glcd_gotoxy(xPos, lcd.y + 8);
        }
}
//-------------------------------------------------------------------------//
void glcd_Printf (      uint8_t x, uint8_t y, char *string, uint8_t inv,
                                                uint8_t type,uint8_t aligh)
{
                uint8_t len;
                char    tmplcd[32];
                struct font     largeFont,
                                                smallFont;

                lcd.inv                 = inv;
                lcd.number                      = 0;

                largeFont.width  = FONT8X16_WIDTH;
                largeFont.height        = FONT8X16_HEIGHT;
                largeFont.size          = Font8x16;

                smallFont.width = FONT6X8_WIDTH;
                smallFont.height        = FONT6X8_HEIGHT;
                smallFont.size          = Font6x8;
```

```
        glcd_gotoxy (x,y);
        len = strlen (string);
        memset (tmplcd, 0,sizeof tmplcd);
        memcpy (tmplcd,string,len);
        lcd.flag = 1;
        if (type == SMALL_FONT)  //small font
        {
                glcd_PutString (tmplcd, smallFont);
        }
        else //big font
        {
                if (type == BIG_FONT)
                {
                        glcd_PutString (tmplcd, largeFont);
                }
                else                    //NUMBER
                {
                        lcd.number = 1; lcd.flag = 0;
                        if (type == NUMBER_SMALL)
                                glcd_PutString (tmplcd, smallFont);
                        else // NUMBER_BIG
                                glcd_PutString (tmplcd, largeFont);
                }
        }
}
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//

void glcd_Circle  (uint8_t cx, uint8_t cy, uint8_t radius)
```

```
{
        int x, y, xchange, ychange, radiusError;
        x = radius;
        y = 0;
        xchange = 1 - 2 * radius;
        ychange = 1;
        radiusError = 0;

        while(x >= y)
        {
                glcd_SetDot(cx+x, cy+y);
                glcd_SetDot(cx-x, cy+y);
                glcd_SetDot(cx-x, cy-y);
                glcd_SetDot(cx+x, cy-y);
                glcd_SetDot(cx+y, cy+x);
                glcd_SetDot(cx-y, cy+x);
                glcd_SetDot(cx-y, cy-x);
                glcd_SetDot(cx+y, cy-x);
                y++;
                radiusError += ychange;
                ychange += 2;
                if ( 2*radiusError + xchange > 0 )
                {
                        x--;
                        radiusError += xchange;
                        xchange += 2;
                }
        }

}
```

```
//-------------------------------------------------------------------------//
void g_draw_horizontal_line(uint8_t x, uint8_t y, uint8_t length)
{
        uint8_t i;
        for (i = x; i <= x +length; i++)
                glcd_SetDot(i, y);
}
//-------------------------------------------------------------------------//
void g_draw_vertical_line(uint8_t x, uint8_t y, uint8_t length)
{
        uint8_t i;
        for (i = y; i < y+length; i++)
                glcd_SetDot(x, i);
}
//-------------------------------------------------------------------------//
void g_under_vertical_line(uint8_t x, uint8_t y, uint8_t length)
{
        uint8_t i;

        for (i = y ;length > 0; i--)
        {
                glcd_SetDot(x, i);
                 length--;
        }
}//-------------------------------------------------------------------------//
void g_dot_line(uint8_t x, uint8_t y, uint8_t length)
{

        glcd_SetDot(x, y - length);
```

```
}
//-------------------------------------------------------------------------//
void g_clear_vertical_line(uint8_t x, uint8_t y, uint8_t length)
{
        uint8_t i;
        for (i = y; i < y + length; i++)
                glcd_ClearDot(x, i);
}
//-------------------------------------------------------------------------//
void g_draw_rectangle(uint8_t x, uint8_t y, uint8_t width, uint8_t height)
{
        width--;
        height--;
        g_draw_horizontal_line(x, y, width);
        g_draw_vertical_line(x, y, height);
        g_draw_vertical_line(x+width, y, height);
        g_draw_horizontal_line(x, y + height, width);
}
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//
void glcd (void)
{
//      uint8_t t;

        LCD_CMD_PORT |= _BV(RST);                       // select chip 1

        glcd_Init ();

        Blacklight_on();
```

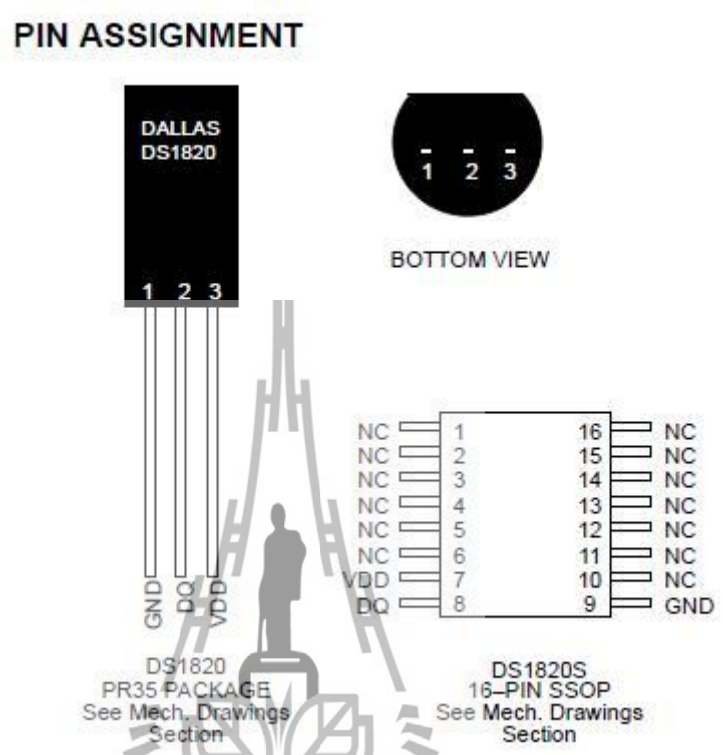
```
        glcd_clearScreen(0);

}
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//
void holding (void)
{
        uint8_t hold;

        for (hold = 0; hold < 50; hold++)
                _delay_ms (500);
}
//-------------------------------------------------------------------------//
void begin (void)
{
        glcd_clearScreen (0);
        g_draw_rectangle (0,0,127,48);
}
//-------------------------------------------------------------------------//
```

ภาคผนวก ค

DS1820



PIN DESCRIPTION

รูปที่ ก.1 แสดงขาการใช้งานของ DS1820

GND – Ground

DQ – Data In/Out

VDD – Optional VDD

NC – No Connect

FEATURES

• Unique 1–WireTM interface requires only one port pin for communication

• Multidrop capability simplifies distributed temperature sensing applications

• Requires no external components

• Can be powered from data line

• Zero standby power required

• Measures temperatures from –55°C to +125°C in 0.5°C increments. Fahrenheit equivalent is –67°F to +257°F in 0.9°F increments

• Temperature is read as a 9–bit digital value.

• Converts temperature to digital word in 200 ms (typ.)

• User–definable, nonvolatile temperature alarm settings

• Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)

• Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

DESCRIPTION

The DS1820 Digital Thermometer provides 9–bit temperature readings which indicate the temperature of the device.

Information is sent to/from the DS1820 over a 1–Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS1820. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS1820 contains a unique silicon serial number, multiple DS1820s can exist on the same 1–Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls,

sensing temperatures inside buildings, equipment or machinery, and in process monitoring

OVERVIEW

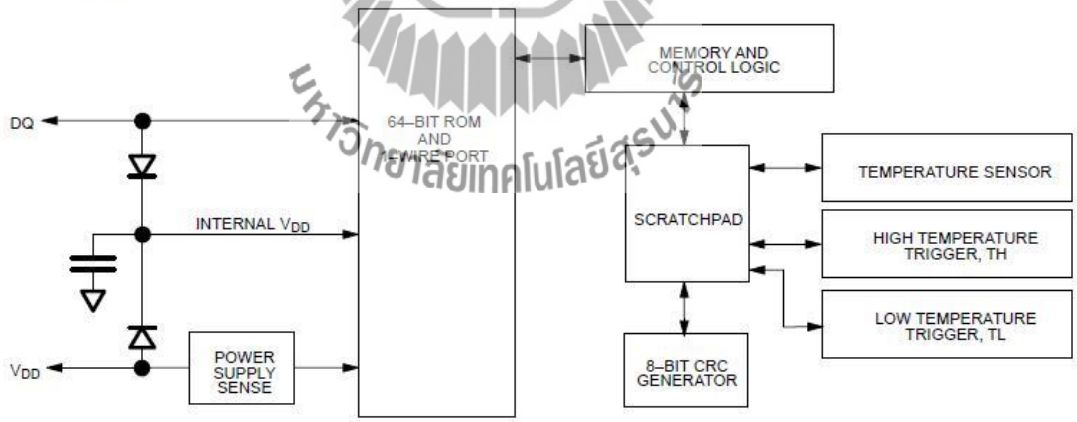
The block diagram of Figure 1 shows the major components of the DS1820. The DS1820 has three main data components: 1) 64–bit lasered ROM, 2) temperature sensor, and 3) nonvolatile temperature alarm triggers TH and TL. The device derives its power from the 1–Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1–Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS1820 may also be powered from an external 5 volts supply.

Communication to the DS1820 is via a 1–Wire port. With the 1–Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64–bit lasered ROM portion of each device and can single out a specific device if many are present on the 1–Wire line as well as indicate to the Bus Master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master

may then provide any one of the six memory and control function commands.

One control function command instructs the DS1820 to perform a temperature measurement. The result of this measurement will be placed in the DS1820's scratchpad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of one byte EEPROM each. If the alarm search command is not applied to the DS1820, these registers may be used as general purpose user memory. Writing TH and TL is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.

DS1820 BLOCK DIAGRAM Figure 1



รูปที่ ค.2 DS1820 Block diagram

PARASITE POWER

The block diagram (Figure 1) shows the parasite powered circuitry. This circuitry "steals" power whenever the

I/O or VDD pins are high. I/O will provide sufficient power as long as the specified timing and voltage requirements are met (see the section titled “1–Wire Bus System”). The advantages of parasite power are two–fold: 1) by parasiting off this pin, no local power source is needed for remote sensing of temperature, and 2) the ROM may be read in absence of normal power.

In order for the DS1820 to be able to perform accurate temperature conversions, sufficient power must be provided over the I/O line when a temperature conversion is taking place. Since the operating current of the DS1820 is up to 1 mA, the I/O line will not have sufficient drive due to the 5K pull–up resistor. This problem is particularly acute if several DS1820’s are on the same I/O and attempting to convert simultaneously. There are two ways to assure that the DS1820 has sufficient supply current during its active conversion cycle.

The first is to provide a strong pull–up on the I/O line whenever temperature conversions or copies to the E2 memory are taking place. This may be accomplished by using a MOSFET to pull the I/O line directly to the power supply as shown in Figure 2. The I/O line must be switched over to the strong pull–up within 10 ms maximum after issuing any protocol that involves copying to the E2 memory or initiates temperature conversions. When using the parasite power mode, the VDD pin must be tied to ground.

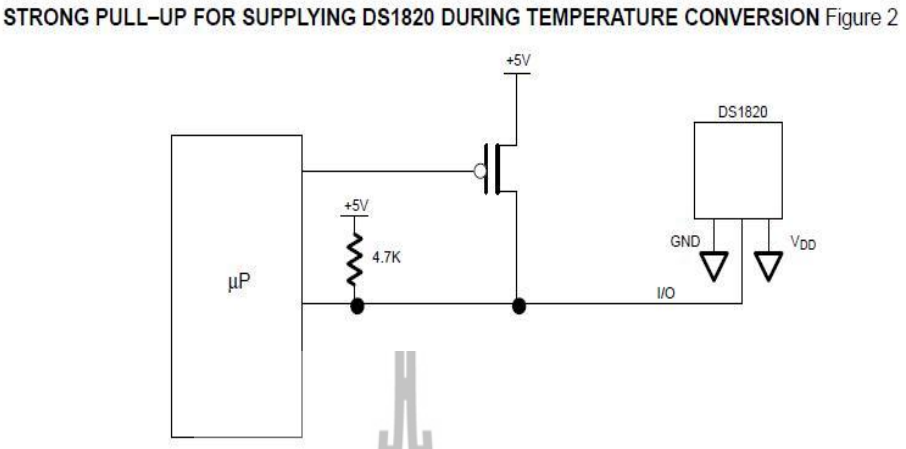
Another method of supplying current to the DS1820 is through the use of an external power supply tied to the VDD pin, as shown in Figure 3. The advantage to this is

that the strong pull–up is not required on the I/O line, and the bus master need not be tied up holding that line high during temperature conversions. This allows other data traffic on the 1–Wire bus during the conversion time. In addition, any number of DS1820's may be placed on the 1–Wire bus, and if they all use external power, they may all simultaneously perform temperature conversions by issuing the Skip ROM command and then issuing the Convert T command. Note that as long as the external power supply is active, the GND pin may not be floating.

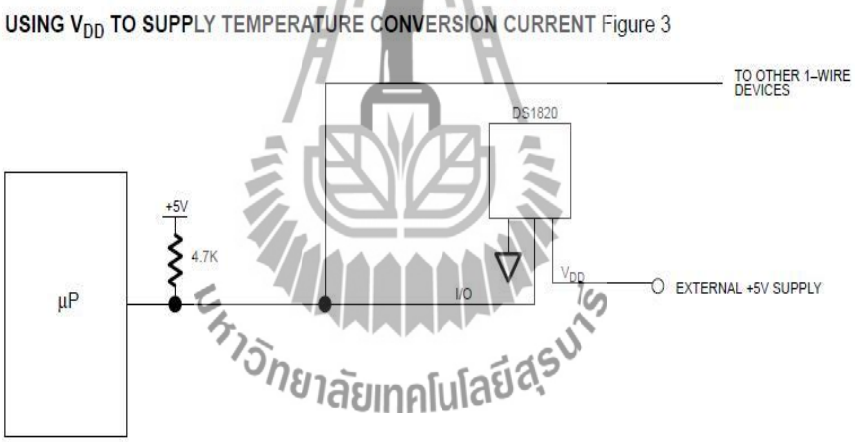
The use of parasite power is not recommended above 100°C, since it may not be able to sustain communications given the higher leakage currents the DS1820 exhibits at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that VDD be applied to the DS1820.

For situations where the bus master does not know whether the DS1820's on the bus are parasite powered or supplied with external VDD, a provision is made in the DS1820 to signal the power supply scheme used. The bus master can determine if any DS1820's are on the bus which require the strong pull–up by sending a Skip ROM protocol, then issuing the read power supply command. After this command is issued, the master then issues read time slots. The DS1820 will send back "0" on the 1–Wire bus if it is parasite powered; it will send back a "1" if it is powered from the VDD pin. If the master receives a "0", it knows that it must supply the strong pull–up on the I/O line during temperature conversions. See "Memory Command Functions" section for more

detail on this command protocol.

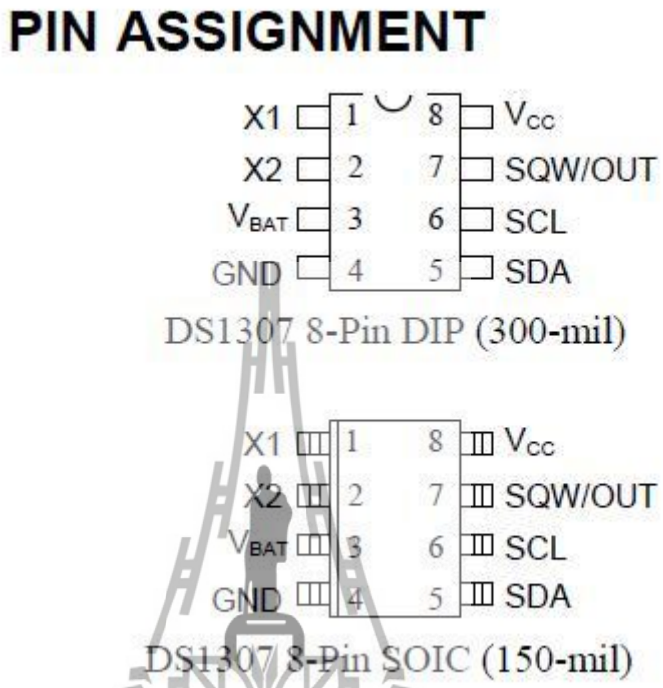


รูปที่ ค.3 การต่อไฟภายในจากขา Pull-up



รูปที่ ค.4 การต่อไฟจากแหล่งจ่ายภายนอก

DS1307



รูปที่ ค.5 แสดงขาในการทำงานของ DS1307

PIN DESCRIPTION

VCC - Primary Power Supply

X1, X2 - 32.768kHz Crystal Connection

VBAT - +3V Battery Input

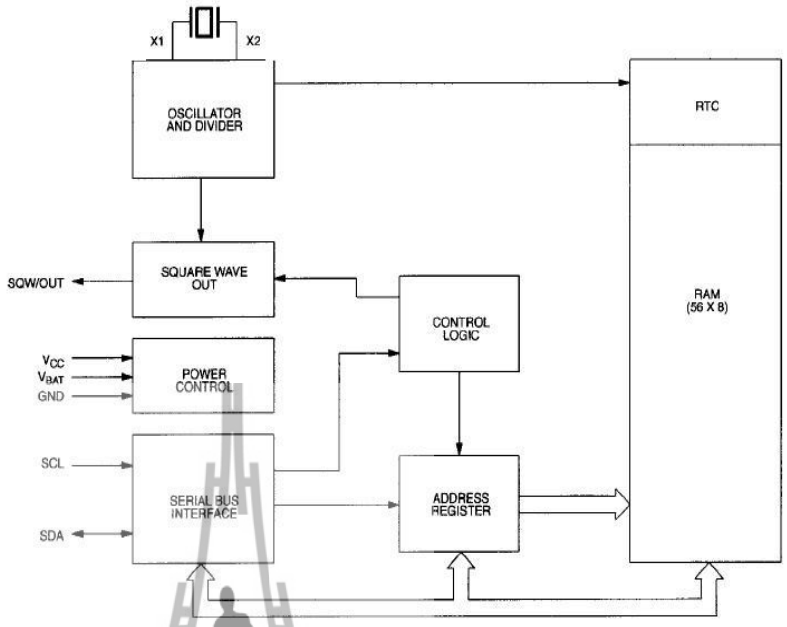
GND - Ground

SDA - Serial Data

SCL - Serial Clock

SQW/OUT - Square Wave/Output

DS1307 BLOCK DIAGRAM Figure 1



รูปที่ ค.6 DS1307 Block diagram

FEATURES

_ Real-time clock (RTC) counts seconds,
minutes, hours, date of the month, month, day
of the week, and year with leap-year
compensation valid up to 2100

_ 56-byte, battery-backed, nonvolatile (NV)
RAM for data storage

_ Two-wire serial interface

_ Programmable squarewave output signal

_ Automatic power-fail detect and switch
circuitry

_ Consumes less than 500nA in battery backup
mode with oscillator running

_ Optional industrial temperature range:

-40°C to +85°C

_ Available in 8-pin DIP or SOIC

_ Underwriters Laboratory (UL) recognized

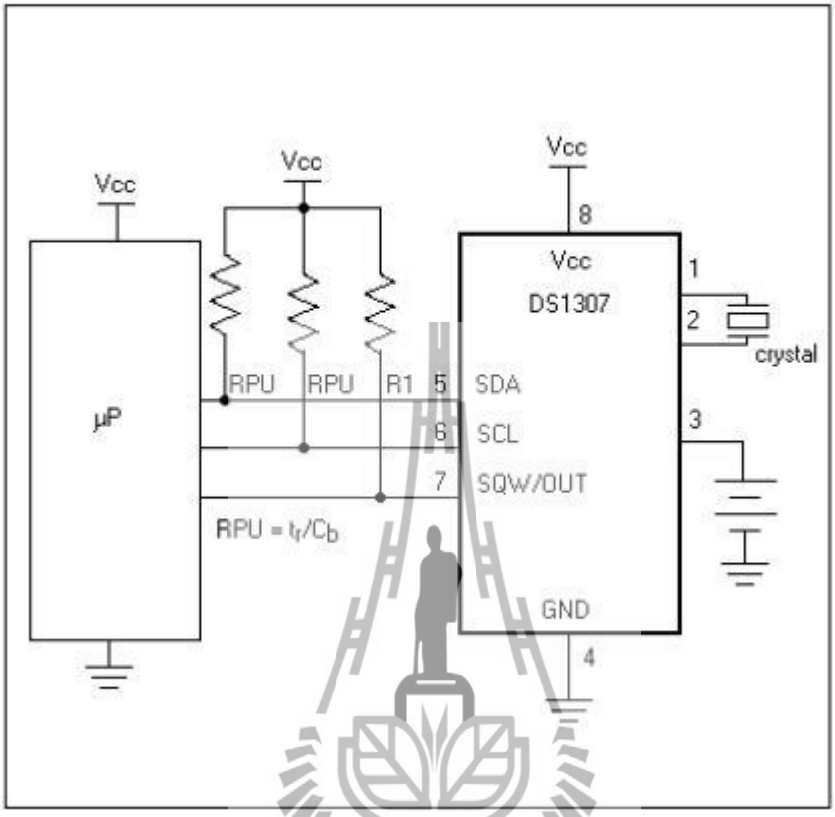
DESCRIPTION

The DS1307 Serial Real-Time Clock is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus.

The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year.

The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit that detects power failures and automatically switches to the battery supply.

TYPICAL OPERATING CIRCUIT



รูปที่ ค.7 วงจรชนิด Operating