ระบบควบคุมอุปกรณ์ไฟฟ้าผ่าน RS-485

Electical appliances control system via RS-485

โดย

นายปวริศ แขกระโทก

นายสุเทพ ชาติเผือก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ระบบควบคุมอุปกรณ์ไฟฟ้าผ่าน RS-485

Electical appliances control system via RS-485

โดย

นายปวริศ แขกระโทก 47015018

นายสุเทพ ชาติเผือก 47015031

อาจารย์ที่ปรึกษา

รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **ระบบควบคุมอุปกรณ์ไฟฟ้าผ่าน RS-485**

Electronic appliances control system via RS-485

ผู้จัดทำ

1.นาย ปวริศ แขกระโทก 47015018

2.นาย สุเทพ ชาติเผือก 47015031

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี)

ระบบควบคุมอุปกรณ์ไฟฟ้าผ่าน RS-485

Electronic appliances control system via RS-485

ผู้จัดทำ 1. นายปวริศ แขกระโทก 47015018
2. นายสุเทพ ชาติเผือก 47015031

อาจารย์ที่ปรึกษา รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี

บทคัดย่อ

โครงงานนี้เป็นการศึกษาและการนำเอาระบบไมโครคอนโทรลเลอร์มาประยุกต์ใช้งานเพื่อควบคุมอุปกรณ์ไฟฟ้าต่างๆ ภายในอาคาร สำนักงาน หรือห้างสรรพสินค้าโดยใช้คอมพิวเตอร์เป็นตัวสั่งการผ่านพอร์ต RS-485 ส่งผ่านไปยังไมโครคอนโทรลเลอร์เครื่องลูกข่ายโดยแต่ละเครื่องทำหน้าที่ในการควบคุมอุปกรณ์ไฟฟ้าในแต่ละชั้น

Abstract

This project is conducted to study and apply the microcontroller system to control appliances in offices or department stores by using computer to control them via RS-485 port. The system will be sent to microcontroller slave which each slave will control each appliances.

สารบัญ

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาของโครงงาน

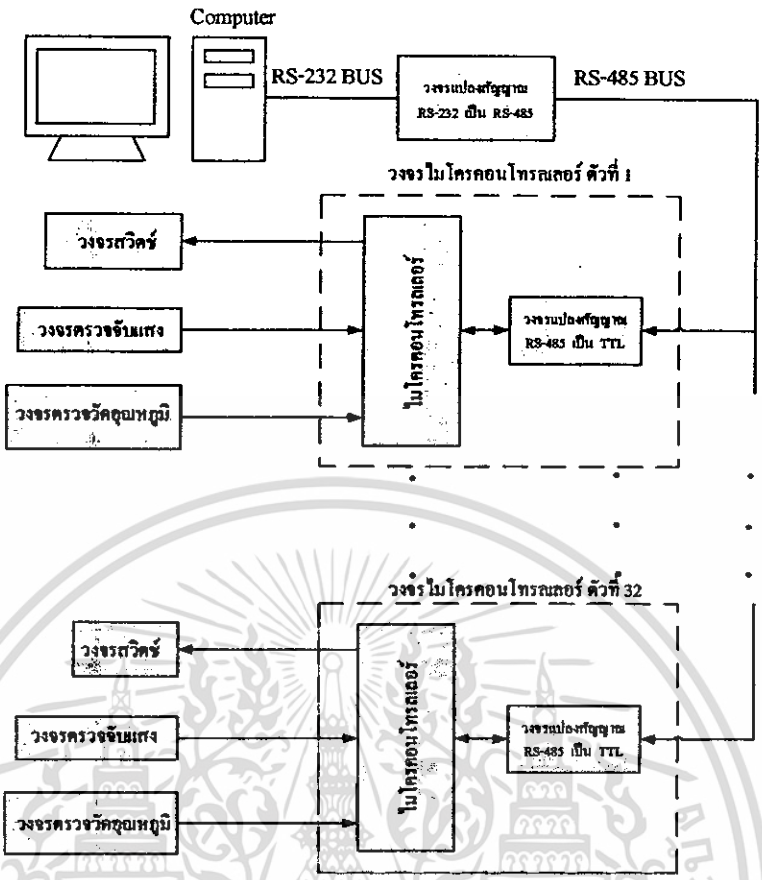
ปัจจุบันเทคโนโลยีได้มีการพัฒนาขึ้นอย่างต่อเนื่องไม่ว่าจะเป็นเทคโนโลยีทางด้านคอมพิวเตอร์ (Computer) ที่ได้มีการพัฒนาซอร์ฟแวร์ต่างๆ ขึ้นมากมาย และเทคโนโลยีด้านไมโครคอนโทรลเลอร์ (Microcontroller) ที่ได้มีการพัฒนาและนำมาใช้งานในการควบคุมอุปกรณ์ต่างๆ เช่น เครื่องจักรภายในโรงงานอุตสาหกรรมต่างๆ และด้วยคุณประโยชน์ต่างๆ เหล่านี้เองที่เป็นแนวทางในการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ร่วมกับคอมพิวเตอร์ จะนำเอาไมโครคอนโทรลเลอร์มาควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร ซึ่งจะติดต่อสื่อสารกันระหว่างคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ผ่านพอร์ตอนุกรม RS-232 แต่เนื่องจากในการออกแบบต้องการรูปแบบการสื่อสารระหว่างคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ที่เป็นแบบ Point to multipoint และมีระยะทางที่ไกลเพราะเนื่องจากภายในอาคารอาจมีมากกว่าหนึ่งชั้น จึงทำให้ต้องใช้ไมโครคอนโทรลเลอร์หลายตัว ด้วยเงื่อนไขนี้เองทำให้ในโครงงานได้เลือกรูปแบบการติดต่อสื่อสารแบบพอร์ตอนุกรมมาตรฐาน RS-485 เนื่องจากมีคุณสมบัติที่เพิ่มมาจากมาตรฐาน RS-232 คือ สามารถติดต่อกับอุปกรณ์ลูกข่ายได้มากสูงสุดถึง 32 ตัว และมีระยะทางในการติดต่อสื่อสารได้ไกลถึง 1200 เมตร จึงเป็นที่มาของปริญญานิพนธ์ฉบับนี้

1.2 วัตถุประสงค์

1. เพื่อศึกษาการรับส่งข้อมูลผ่านพอร์ตอนุกรม
2. เพื่อศึกษาการเชื่อมต่อระหว่างพอร์ต RS-485 กับ ไมโครคอนโทรลเลอร์
3. เพื่อศึกษาการทำงานของวงจรไมโครคอนโทรลเลอร์
4. เพื่อศึกษาการรับส่งข้อมูลผ่านพอร์ตอนุกรมและประมวลผลโดยใช้ภาษาซี

1.3 หลักการทำงาน

คอมพิวเตอร์จะทำหน้าที่รับข้อมูลจากไมโครคอนโทรลเลอร์และส่งข้อมูลผ่านพอร์ตอนุกรมโดยจะมีวงจรแปลงสัญญาณ RS-232 เป็นสัญญาณ RS-485 และที่วงจรไมโครคอนโทรลเลอร์จะแปลงสัญญาณ RS-485 เป็นสัญญาณ ทีทีแอล(TTL) เพื่อส่งผ่านให้ไมโครคอนโทรลเลอร์ โดยไมโครคอนโทรลเลอร์จะนำค่าที่รับได้มาควบคุมอุปกรณ์ไฟฟ้า สำหรับวงจรตรวจจับแสงจะทำหน้าที่ตรวจจับความสว่างของหลอดไฟ คือ ถ้าอยู่ในสถานะมืดเอาต์พุตจะมีค่าเป็น “1” และถ้าอยู่ในสถานะสว่างเอาต์พุตจะมีค่าเป็น “0” จากนั้นจะส่งค่ากลับไปยังไมโครคอนโทรลเลอร์ แล้วจะส่งค่าที่แสดงถึงสถานะของไหลดกลับไปให้ยังคอมพิวเตอร์เพื่อแสดงผลที่หน้าจอ และวงจรตรวจวัดอุณหภูมิจะใช้ IC DS1820 เป็นตัวตรวจวัดอุณหภูมิซึ่งส่งข้อมูลแบบดิจิตอลในระบบบัสหนึ่งสายให้กับไมโครคอนโทรลเลอร์ เพื่อทำการประมวลผลแล้วส่งไปแสดงค่าอุณหภูมิที่หน้าจอคอมพิวเตอร์ ดังแสดงในรูปที่ 1.1



รูปที่ 1.1 แผนภาพแสดงการทำงานโดยรวมของระบบควบคุมอุปกรณ์ไฟฟ้าผ่าน RS-485

1.4 ขั้นตอนการดำเนินงาน

1. แบ่งโครงงานเป็น 4 ส่วน
   1.1 ส่วนฮาร์ดแวร์(Hardware)ที่ใช้ทำการแปลงสัญญาณจากมาตรฐาน RS-232 ให้เป็นสัญญาณมาตรฐาน RS-485
   1.2 ส่วนของฮาร์ดแวร์ไมโครคอนโทรลเลอร์ที่มีการเชื่อมต่อแบบ RS-485
   1.3 ส่วนฮาร์ดแวร์ซึ่งเป็นระบบเซนเซอร์
   1.4 ส่วนโปรแกรมที่ติดต่อระหว่างคอมพิวเตอร์ กับไมโครคอนโทรลเลอร์
2. ออกแบบวงจรและ สร้างวงจรไมโครคอนโทรลเลอร์ วงจรรีเลย์สวิตช์ วงจรแปลงสัญญาณมาตรฐาน RS-485 วงจรเซนเซอร์ และ วงจรตรวจวัดอุณหภูมิ
3. เขียนโฟล์วชาร์ต (Flow Chart) การทำงานของระบบควบคุมอุปกรณ์ไฟฟ้าผ่าน RS-485
4. ดำเนินการเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์โดยใช้ภาษาซี
5. ทดสอบการทำงานร่วมกันทั้งระบบ

บทที่ 2
ทฤษฎีและการสร้าง

2.1 พื้นฐานการสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมในเครื่องคอมพิวเตอร์นั้นจะมีความเร็วในการสื่อสารช้ากว่าแบบขนาน ทั้งนี้ก็เพราะว่าการเคลื่อนย้ายข้อมูลแบบอนุกรมนั้นเป็นการส่งข้อมูลครั้งละ 1 บิต แต่พอร์ตขนานนั้นสามารถส่งข้อมูลได้ครั้งละหลายๆ บิตพร้อมกันส่งผลให้การสื่อสารข้อมูลแบบอนุกรมมีความเร็วต่ำกว่าแบบขนาน

แต่ว่าการส่งข้อมูลแบบอนุกรมนั้นมีข้อที่เหนือกว่าการส่งข้อมูลแบบขนานคือ การสามารถส่งข้อมูลได้ในระยะทางที่ไกลกว่าแบบขนาน อีกทั้งสายสัญญาณที่ใช้ยังมีน้อยกว่าการส่งข้อมูลแบบขนานอีกด้วย การสื่อสารแบบอนุกรมสามารถแบ่งออกเป็น 3 รูปแบบ ดังนี้

1. Simplex สามารถส่งข้อมูลได้อย่างเดียว เป็นการสื่อสารแบบทางเดียว
2. Half-Duplex สามารถส่งข้อมูลไปยังปลายทางและสามารถรับข้อมูลจากปลายทางได้ แต่ไม่สามารถทำการส่งและรับข้อมูลได้ในเวลาเดียวกัน
3. Full-Duplex สามารถรับและส่งข้อมูลได้ในเวลาเดียวกัน

นอกจากนี้แล้วยังสามารถแบ่งประเภทของการสื่อสารแบบอนุกรมตามลักษณะสัญญาณในการส่งได้อีก 2 แบบคือ

- **การสื่อสารแบบซิงโครนัส (Synchronous)** สำหรับการสื่อสารแบบซิงโครนัสนี้จะใช้สัญญาณนาฬิกาควบคุมการรับส่งสัญญาณ เช่น สายคีย์บอร์ดคอมพิวเตอร์ โดยจะมีสายสัญญาณเส้นหนึ่งเป็นสายสัญญาณนาฬิกา ส่วนอีกเส้นหนึ่งเป็นสายของข้อมูล(และมักจะมีสายกราวด์ด้วย)
- **การสื่อสารแบบอะซิงโครนัส (Asynchronous)** สำหรับการสื่อสารแบบอะซิงโครนัสนั้นจะใช้สายข้อมูลเพียงตัวเดียว แต่จะใช้รูปแบบการส่งข้อมูลหรือ Bit Pattern เป็นตัวกำหนดว่าส่วนไหนเป็นส่วนเริ่มต้นข้อมูล, ส่วนไหนเป็นตัวข้อมูล, ส่วนไหนจะเป็นส่วนตรวจสอบความถูกต้องข้อมูลและส่วนไหนเป็นส่วนปิดท้ายของข้อมูล โดยต้องกำหนดให้สัญญาณนาฬิกาเท่ากันทั้งภาคส่งและภาครับ ซึ่งจะมีอุปกรณ์พิเศษที่ชื่อว่า UART หรือ Universal Asynchronous Receiver/Transmitter คอยควบคุมการรับและส่งข้อมูล

2.2 ระบบบัส 1 สาย (1-wire bus)

2.2.1 คุณสมบัติทางเทคนิคของระบบบัส 1 สาย

สายสัญญาณบนระบบบัสนี้เป็นแบบสองทิศทาง แต่ข้อมูลเดินทางได้ทิศทางเดียวภายในช่วงเวลาหนึ่งๆหรือเรียกว่า ฮาล์ฟดูเพล็กซ์ (half duplex) อุปกรณ์บนระบบบัสต้องระบุอย่างชัดเจนว่า ตัวใดเป็นมาสเตอร์ ตัวใดเป็นสเลฟโดยส่วนใหญ่อุปกรณ์มาสเตอร์คือไมโครคอนโทรลเลอร์ ส่วนอุปกรณ์สเลฟได้แก่ ไอซีวัดอุณหภูมิ,หน่วยความจำแรม เป็นต้น ข้อมูลทั้งหมดจะถูกส่งลงบนสายสัญญาณที่มีอยู่เพียงเส้นเดียวนี้ ในระหว่างการทำงานอุปกรณ์มาสเตอร์และสเลฟสามารถเป็นได้ทั้งตัวส่งและตัวรับ ขึ้นอยู่กับเงื่อนไขของการทำงานในขณะนั้น ใน 1 ระบบต้องมีอุปกรณ์มาสเตอร์เพียงตัวเดียว

2.2.2 คุณสมบัติของไทม์สล็อต

อุปกรณ์มาสเตอร์เป็นอุปกรณ์เพียงตัวเดียวบนระบบบัสที่สามารถอินนิเชียลสายสัญญาณได้ โดยกำเนิดจุดเริ่มต้นของไทม์สล็อตด้วยการทำให้สายสัญญาณเป็นลอจิกต่ำช่วงเวลาหนึ่ง จากนั้นทำให้กลับเป็นลอจิกสูง ถ้าหากอุปกรณ์สเลฟต้องการส่งข้อมูลมายังมาสเตอร์ อุปกรณ์สลาฟจะเป็นตัวควบคุมสภาวะของสายสัญญาณต่อไป แต่ถ้าอุปกรณ์มาสเตอร์ต้องการส่งข้อมูลจะสามารถส่งได้เลย

2.2.3 รูปแบบของการสื่อสารข้อมูลแบบ 1 เส้น (1-Wire communication protocol)

ในระบบบัสหนึ่งสาย อุปกรณ์มาสเตอร์สามารถติดต่อกับอุปกรณ์สเลฟได้ครั้งละ 1 ตัวเท่านั้น อุปกรณ์สเลฟแต่ละตัวจึงต้องมีข้อมูลกำหนดแอดเดรสเฉพาะตัว โดยเก็บไว้ในหน่วยความจำรอมภายในอุปกรณ์สเลฟตัวนั้นๆ โดยปกติอุปกรณ์สเลฟจะมีหน่วยความจำขนาด 64 บิตหรือ 8 ไบต์ สำหรับเก็บข้อมูลต่างๆ ที่สำคัญของอุปกรณ์แต่ละตัว ซึ่งประกอบด้วย

1. รหัสของตระกูล จำนวน 8 บิต
2. เลขหมายประจำตัว (serial number) จำนวน 48 บิต
3. รหัสตรวจสอบความผิดพลาด (CRC : Cyclical Redundancy Check) จำนวน 8 บิต

ผู้ใช้งานสามารถอ่านข้อมูลประจำตัวของอุปกรณ์สเลฟได้ด้วยคำสั่งอ่านหน่วยความจำรอม (Read ROM :0x33) ในกรณีที่บนบัสมีอุปกรณ์สเลฟตัวเดียวไม่จำเป็นต้องอ้างแอดเดรสในการติดต่อจึงใช้คำสั่งไม่อ่านรอม (Skip ROM : 0xCC) เพื่อลดเวลาในการติดต่อลง การติดต่อจะเริ่มต้นขึ้นเมื่ออุปกรณ์มาสเตอร์ทำการรีเซตและกำหนดแอดเดรสของอุปกรณ์ที่ทำการติดต่อ ถ้าหากมีอุปกรณ์สเลฟเพียงตัวเดียวสามารถข้ามขั้นตอนการติดต่อกับหน่วยความจำรอมในอุปกรณ์สเลฟได้ จะเรียกวิธีการดังกล่าว การไม่ติดต่อหน่วยความจำรอม หรือ สคิปรอม (Skip ROM : $CC) จากนั้นรอการตอบรับจากอุปกรณ์สเลฟ เมื่อการตอบรับสมบูรณ์ก็จะสามารถเริ่มต้นขั้นตอนการอ่านหรือเขียนข้อมูลได้ต่อไป

2.3 มาตรฐาน RS-232

มาตรฐาน RS-232 เป็นมาตรฐานที่ได้รับการออกแบบมาเพื่อที่จะทำให้อุปกรณ์ต่อพ่วงจากผู้ผลิตต่างกันสามารถทำงานร่วมกันได้มาตรฐานหลายชนิดได้รับการออกแบบขึ้นมา แต่มาตรฐานที่ได้รับความนิยมและใช้กันกว้างขวางมากที่สุดคือ มาตรฐาน RS-232 ซึ่งถูกประกาศใช้ในปี 1969 โดยสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ในยุคแรกๆ การอินเตอร์เฟสแบบ RS-232 ถูกออกแบบสำหรับเชื่อมต่อเทอร์มินอล (DTE: Data Terminal Equipment) กับโมเด็ม (DCE : Data Communication Equipment) ทั้งนี้ก็เพื่อป้องกันไม่ให้เกิดการส่งข้อมูลบนสายเส้นเดียวกัน

มาตรฐาน RS-232 ได้แบ่งอุปกรณ์ออกเป็น 2 ประเภท ซึ่งอุปกรณ์ทั้งสองประเภทนี้ก็คือ

1. อุปกรณ์ DTE (Data Terminal Equipment) เป็นอุปกรณ์สำหรับส่งข้อมูล (เอาต์พุต)
2. อุปกรณ์ DCE (Data Communication Equipment) เป็นอุปกรณ์สำหรับรับข้อมูล (อินพุต)

ตามมาตรฐาน RS-232 แล้วคอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งคอนเน็กเตอร์ที่นิยมใช้กันอยู่จะเป็นชนิด D-Type แบบ 9 ขา และแบบ 25 ขา โดยจะติด

ตั้งอยู่หลังเครื่องคอมพิวเตอร์ ระดับแรงดันจะมีค่าระหว่าง -3 โวลต์ถึง -15 โวลต์ สำหรับลอจิก สูง(High) และลอจิก ต่ำ(Low) จะมีระดับแรงดันระหว่าง +3 โวลต์ ถึง +15 โวลต์ สามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุด 50 ฟุต หรือ 150 เมตร

2.3.1 ลักษณะของคอนเน็กเตอร์แบบ D-Type

หัวต่อแบบ D-Type ที่ใช้ในการสื่อสารแบบอนุกรมของเครื่องคอมพิวเตอร์นั้น จะมีอยู่ 2 ลักษณะ คือ แบบ 9 ขา และแบบ 25 ขา บางครั้งเราจะเรียกว่า DB9 และ DB25 ซึ่งหัวต่อทั้งสองชนิดจะมีลักษณะการทำงานของสัญญาณเหมือนกัน แต่การจัดเรียงไม่เหมือนกัน

D-Type 25 Pin	D-Type 9 Pin	สัญลักษณ์	ชื่อสัญญาณ
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

ตารางที่ 2.1 แผนผังคอนเน็กเตอร์ของ RS-232

2.3.2 รายละเอียดของสายสัญญาณ

สำหรับรายละเอียดของสายสัญญาณนั้นประกอบไปด้วย

Transmit Data : TD ใช้สำหรับส่งข้อมูลอนุกรมออกจากคอมพิวเตอร์

Receive Data : RD ใช้สำหรับส่งข้อมูลอนุกรมเข้ามายังคอมพิวเตอร์

Request To Send : RTS ใช้สำหรับส่งข้อมูลไปยังอุปกรณ์ปลายทางเพื่อร้องขอให้อุปกรณ์ปลายทางส่งข้อมูลกลับมา

Clear To Send : CTS ใช้สำหรับตรวจสอบว่าอุปกรณ์ที่เชื่อมต่อด้วยพร้อมที่จะรับข้อมูลหรือไม่ โดยจะคอยรับสัญญาณ RTS เมื่อทุกอย่างพร้อมก็จะทำการส่งข้อมูลออกทางขา TD

Data Set Ready : DSR ใช้สำหรับตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทางจะใช้คู่กับขา DTR

Signal Ground : SG เป็นกราวด์ของระบบ

Carrier Detect : CD ขานี้จะ Active เมื่อมีการส่งสัญญาณ Carrier จากโมเด็ม

Data Terminal Ready : DTR ใช้สำหรับบอกให้อุปกรณ์ปลายทางรับรู้ว่าต้องการติดต่อด้วยโดยขา DTR นี้ต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทาง

Ring Indicator : RI ขานี้จะ Active เมื่อโมเด็มได้รับสัญญาณเรียกเข้าจากสายโทรศัพท์

2.3.3 องค์ประกอบของการรับส่งข้อมูลแบบอนุกรม

การสื่อสารแบบอนุกรมที่นิยมใช้กับคอมพิวเตอร์นั้น เป็นการสื่อสารข้อมูลแบบอะซิงโครนัสนั่นคือ ต้องใช้สายสัญญาณเส้นเดียวทำหน้าที่ทั้งส่งส่วนที่เป็นข้อมูลและส่วนที่ใช้ควบคุมการส่งข้อมูลดังนั้นข้อมูลที่อ่านได้แต่ละบิตจากการส่งแบบอนุกรมจึงต้องถูกแยกแยะว่าใช้สำหรับวัตถุประสงค์ใดโดยเราสามารถแบ่งได้เป็น 4 ส่วน คือ

1. Start Bit ขนาด 1 บิต
2. บิตข้อมูล (Data Character) ขนาด 7 บิต หรือ 8 บิต
3. Parity Bit ขนาด 1 บิต
4. Stop Bit ขนาด 1 บิต หรือ 2 บิต

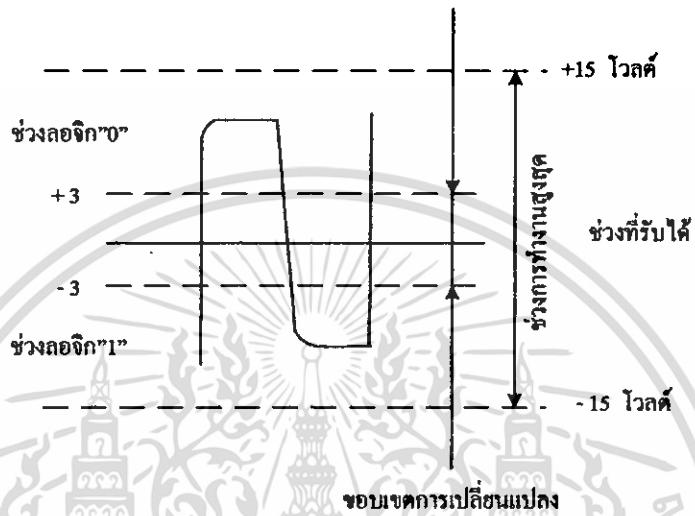
แต่ละตัวอักษรที่ถูกส่งออกไปเป็นกลุ่มจะประกอบไปด้วยบิตเริ่มต้น บิตข้อมูล บิตพาริตี้ (จะมีหรือไม่มีก็ได้) และบิตจบ โดยเราพอจะสรุปหน้าที่ของแต่ละส่วนได้ดังนี้

- Start Bit หรือบิตเริ่มต้น จะใส่ที่จุดเริ่มต้นเสมอ เพื่อเตือนอุปกรณ์ฝ่ายรับว่าข้อมูลกำลังจะมาถึง
- Data Character หรือบิตข้อมูล การส่งบิตข้อมูลจะส่งเป็นกลุ่มๆ โดยทั่วไปจะส่งเป็น 7 บิต หรือ 8 บิต ซึ่งเพียงพอสำหรับการส่ง ASCII Word
- Parity Bit หรือบิตพาริตี้ ใช้ในการตรวจสอบความถูกต้องของข้อมูลที่ส่ง เราจะใส่บิตพาริตี้เข้าไป แต่ทั้งตัวรับและตัวส่งจะต้องรู้กันว่าใช้พาริตี้แบบไหนในการส่งข้อมูลซึ่งหลักการในการกำหนดบิตพาริตี้มีหลายแบบดังนี้
  - พาริตี้คู่ (Even Parity) ค่าของบิตพาริตี้นี้เมื่อรวมกับทุกๆ บิตของข้อมูลแล้ว จะต้องมีจำนวนบิตที่เป็นเลข 0 เป็นเลขคู่ ตัวอย่างเช่น ข้อมูล 1000101 มีเลข 0 ทั้งหมด 4 ตัวดังนั้นบิตพาริตี้จะเป็น 0
  - พาริตี้คี่ (Odd Parity) ค่าของบิตพาริตี้นี้เมื่อรวมกับทุกๆ บิตของข้อมูลแล้ว จะต้องมีจำนวนบิตที่เป็นเลข 1 เป็นเลขคี่ ตัวอย่างเช่น ข้อมูล 1000101 มีเลข 1 ทั้งหมด 3 ตัว ดังนั้นบิตพาริตี้จะเป็น 1
  - ไม่มีพาริตี้ (None) ถ้าตั้งบิตพาริตี้เป็น None ทั้งภาครับและภาคส่งจะไม่มีการตรวจสอบบิตพาริตี้
- Stop Bit หรือบิตจบ เป็นบิตที่ส่งมาปิดท้ายข้อมูล

2.3.4 การส่งข้อมูลแบบไม่สมดุล (Unbalanced line driver)

RS-232 ใช้ระดับสัญญาณในการส่งข้อมูลแบบไม่สมดุล ซึ่งเป็นระบบที่วัดระดับแรงดันเทียบกับสายกราวด์ ยกตัวอย่าง เช่น ในการส่งข้อมูล (Transmission Data : TD) จากอุปกรณ์ (Data Terminal

Equipment : DTE) ผ่านหัวต่อแบบ DB-9 จะส่งสัญญาณข้อมูลทางขา 3 โดยการวัดระดับแรงดันของสัญญาณจะวัดเทียบกับสายกราวด์ (Signal ground) ที่ขา 5 ถ้าสายส่งข้อมูลอยู่ในสถานะ idle ระดับแรงดันจะมีค่าเป็นลบ และเมื่อทำการส่งข้อมูลระดับแรงดันจะเปลี่ยนแปลงในช่วงค่าบวกและค่าลบ โดยมีระดับแรงดันอยู่ในช่วง ± 5 โวลต์ ถึง ± 15 โวลต์ รูปที่ 2.1 แสดงการทำงานที่ตัวรับของ RS-232 โดยจะทำงานในระดับแรงดันช่วง ±3 โวลต์ ถึง ± 12 โวลต์



รูปที่ 2.1 เอาต์พุตของตัวส่งแบบไม่สมดุล

2.3.5 อัตราเร็วในการรับส่งข้อมูลแบบอนุกรม

การที่อุปกรณ์ 2 อย่างจะติดต่อสื่อสารกันได้นั้น จะต้องทำงานด้วยอัตราเร็วเท่ากัน ซึ่งอัตราเร็วในการสื่อสารแบบอะซิงโครนัสคือ ค่าบอดเรต (Baud Rate) มีหน่วยเป็นบิตต่อวินาที ซึ่งค่าอัตราเร็วในการสื่อสารแบบอนุกรมสำหรับมาตรฐาน RS-232 นั้นมีใช้ดังนี้ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที

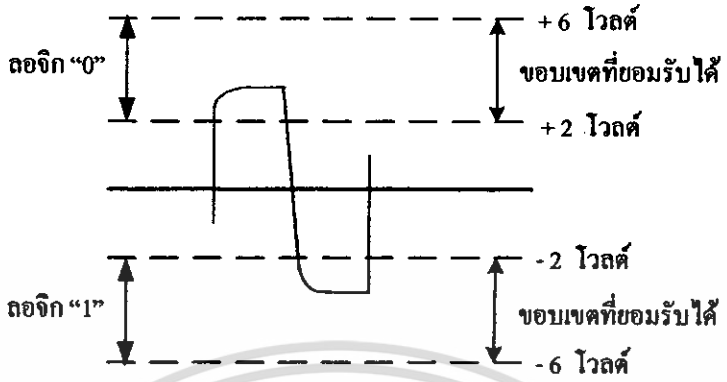
2.4 มาตรฐานการอินเตอร์เฟส RS-485

RS-485 เป็นการส่งข้อมูลในระบบสมดุล (Balanced System) และจากนี้จะแทนมาตรฐาน EIA/TIA-485 ด้วย RS-485

2.4.1 ตัวส่งแบบสมดุล (Balanced line driver)

ระบบสมดุลจะส่งสัญญาณผ่านสาย 2 เส้น รูปที่ 2.2 แสดงรูปแบบและระดับแรงดัน ตัวส่งจะส่งแรงดันช่วง 2-6 โวลต์ ที่เอาต์พุตระบบจะมีการเชื่อมต่อสายกราวด์ 1 เส้น สายกราวด์ในระบบนี้ไม่ได้ใช้ในการส่งสัญญาณหรือหาสถานะลอจิกของข้อมูล แต่สายกราวด์มีความสำคัญคือใช้เป็นจุดอ้างอิงของระบบ การวัดสัญญาณข้อมูลจะถูกวัดเทียบกับสายกราวด์ ในการส่งข้อมูลตัวส่งจะได้รับสัญญาณหนึ่งเรียกว่าสัญญาณควบคุม หรือ Enable signal ซึ่งเชื่อมต่อระหว่างตัวส่งและเทอร์มินอลที่เอาต์พุต ตัวส่งจะส่งสัญญาณควบคุมซึ่งเปรียบเสมือนเป็นเอาต์พุตที่ 3 นอกจากการส่งสถานะลอจิก 1 หรือ 0 ให้กับ

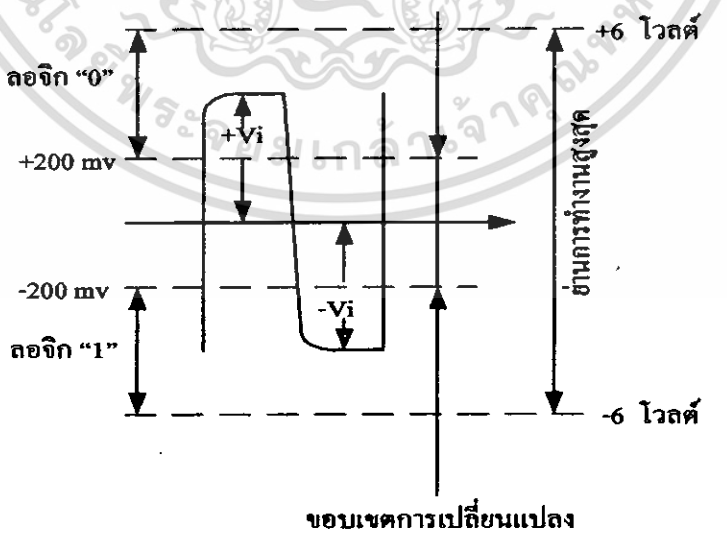
เทอร์มินอล ถ้าสัญญาณควบคุมอยู่ในสถานะ OFF จะหมายถึงตัวส่งตัวนั้นไม่ได้ต่ออยู่กับสายสัญญาณและไม่สามารถส่งสัญญาณข้อมูลได้ หรืออยู่ในสถานะ disable หรือ tri-state



รูปที่ 2.2 เอาต์พุตของตัวส่งแบบสมดุล

2.4.2 ตัวรับแบบสมดุล (Balanced line receiver)

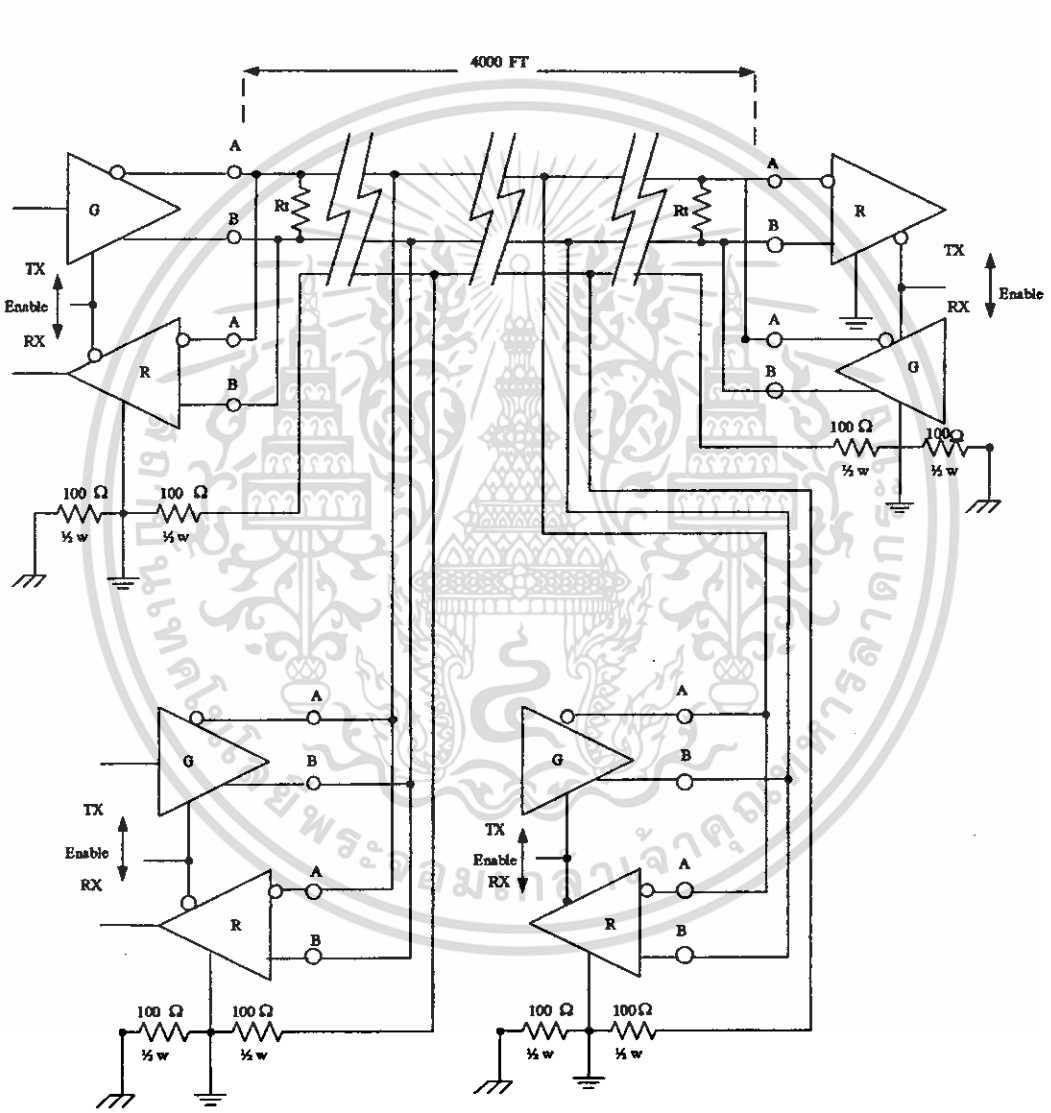
ตัวรับในระบบสมดุลจะถูกเชื่อมต่อด้วยสายส่งสัญญาณข้อมูลและสายกราวด์ โดยจะรับสถานะของสัญญาณได้โดยวัดความแตกต่างของระดับแรงดันที่สายอินพุตของตัวรับ ดังแสดงในรูปที่ 2.3 ถ้าแรงดันมีขนาดมากกว่า +200 mV จะถูกกำหนดให้มีสถานะเป็นลอจิก "0" และถ้ามีขนาดน้อยกว่า -200 มิลลิโวลต์ จะถูกกำหนดให้มีสถานะเป็นลอจิก "1" เนื่องจากอาจเกิดการลดทอนสัญญาณขึ้นได้ในสายส่ง ที่ตัวรับจึงถูกออกแบบให้สามารถรับความแตกต่างระดับแรงดันของสัญญาณได้ในช่วงที่กว้างกว่าตัวส่ง คือ $\pm$ 200 มิลลิโวลต์ ถึง $\pm$ 6 โวลต์ ในขณะที่แรงดันที่ตัวส่งอยู่ในช่วง $\pm$ 2 โวลต์ ถึง $\pm$ 6 โวลต์



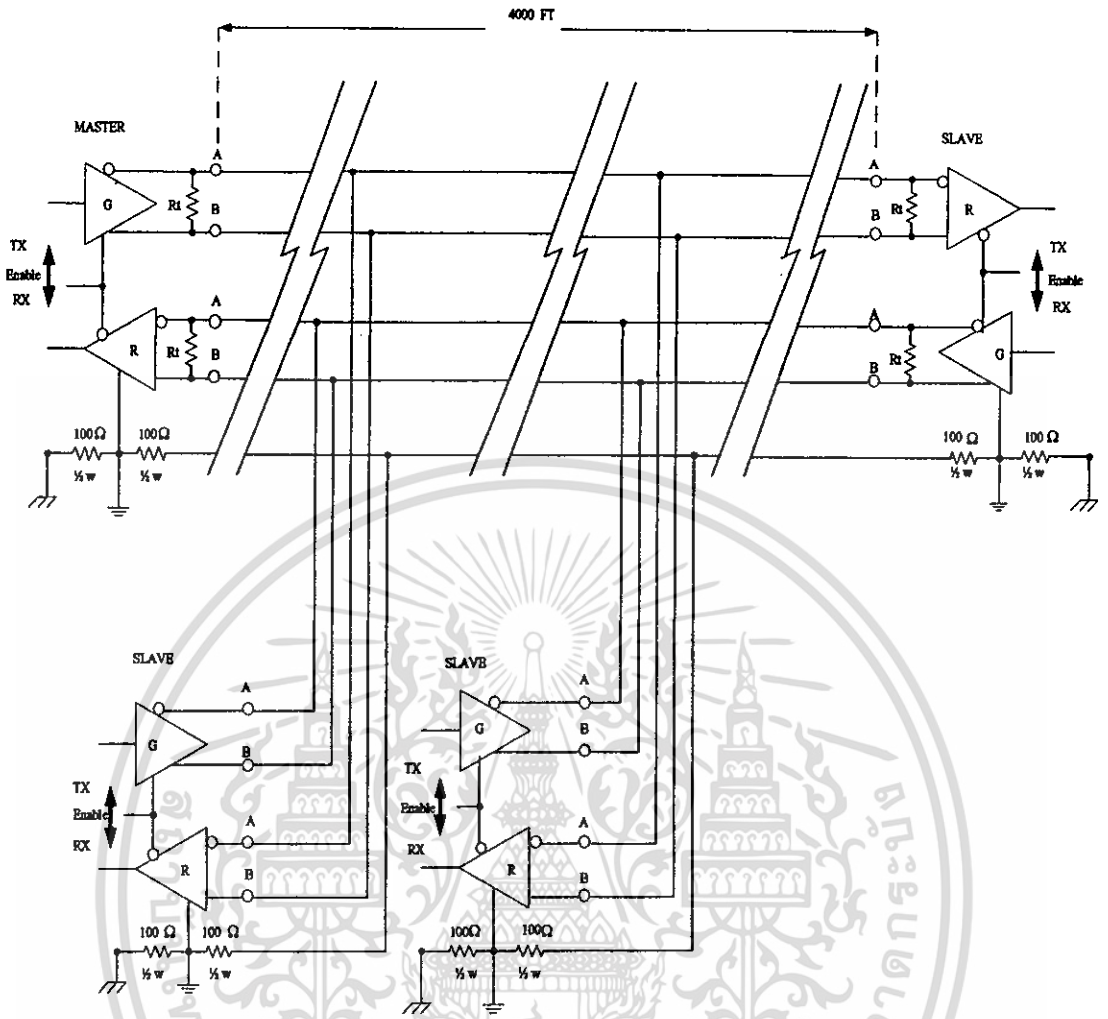
รูปที่ 2.3 อินพุตของตัวรับแบบสมดุล

2.4.3 รูปแบบการเชื่อมต่อของมาตรฐาน RS-485

RS-485 เป็นการส่งข้อมูลในระบบสมดุล สายสัญญาณ 1 คู่สายสามารถติดต่ออุปกรณ์ได้ถึง 32 ตัว คุณสมบัติของอุปกรณ์ในระบบ RS-422 และ RS-485 มีลักษณะคล้ายคลึงกัน ในรูปที่ 2.4 จะแสดงระบบเครือข่ายที่เรียกการต่อในลักษณะนี้ว่า two - wire multi drop จากรูปสังเกตได้ว่ามีการต่อความต้านทานขั้วที่โหนดปลายทั้งสองด้านของสายส่ง แต่ไม่มีการต่อขั้วปลายที่โหนดที่อยู่ระหว่างสายส่งการต่อขั้วปลาย (termination) มักใช้กับระบบที่มีอัตราการส่งข้อมูลสูงและระยะทางยาว



รูปที่ 2.4 ตัวอย่างการเชื่อมต่อของ RS-485 แบบ 2 สาย

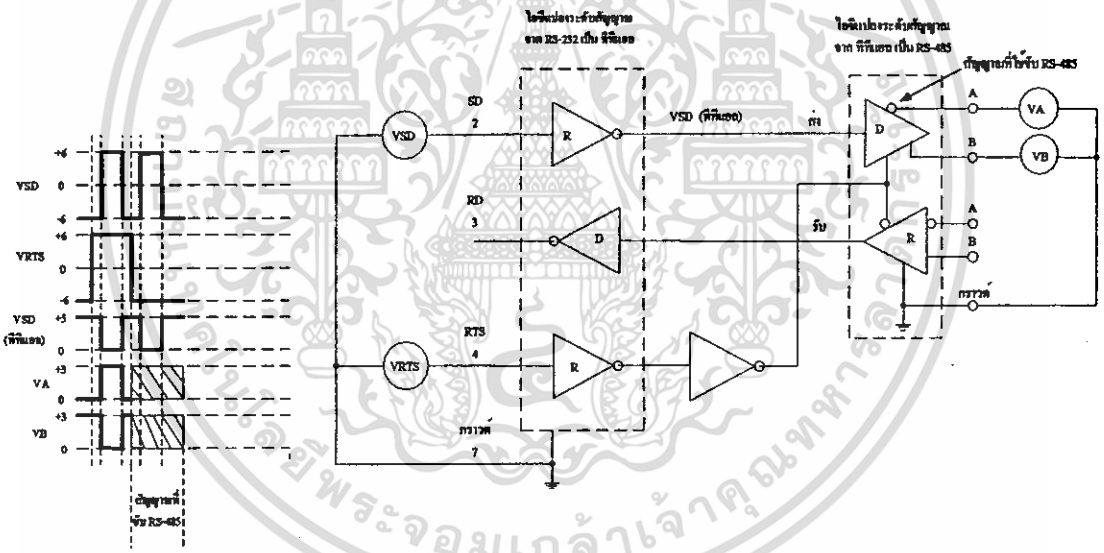


รูปที่ 2.5 ตัวอย่างการเชื่อมต่อของ RS-485 แบบ 4 สาย

รูปที่ 2.5 การเชื่อมต่อระบบ RS-485 แบบ 4 สาย ประกอบด้วยสายสัญญาณ 4 เส้น และสายกราวด์ 1 เส้น มีลักษณะการติดต่อแบบโหนดแม่ (master node) และโหนดลูก (slave node) ตัวส่งของตัวแม่จะต่อถึงตัวรับของตัวลูกทุกตัวผ่านสายสัญญาณ (โดยมากใช้เป็นสายคู่ตีเกลียว) 1 คู่ และตัวส่งของตัวลูกทุกตัวจะต่อเข้ากับตัวรับของตัวแม่ผ่านสายสัญญาณอีก 1 คู่ โดยในการติดต่อนั้นตัวแม่จะสามารถติดต่อกับตัวลูกได้ทุกตัวแต่ตัวลูกทุกตัวจะติดต่อกับตัวแม่ได้เท่านั้น ไม่สามารถติดต่อกับตัวลูกตัวอื่นหรือติดต่อกันเองได้และในการติดต่อจากตัวลูกไปยังตัวแม่นั้นจะทำได้ทีละ 1 ตัว เมื่อมีตัวใดตัวหนึ่งติดต่อหรือทำการส่งข้อมูลอยู่ ตัวลูกตัวอื่นที่เหลือในระบบจะไม่สามารถส่งข้อมูลได้โดยต้องอยู่ในสถานะ disable หรือ tri-state เนื่องจากตัวลูกจะไม่สนใจการติดต่อของตัวลูกอื่นกับตัวแม่ ถือเป็นข้อดีคือทำให้สามารถต่ออุปกรณ์ที่มีโปรโตคอลต่างกันเข้าไว้ในระบบเดียวกันได้

2.4.4 การควบคุม Tri-state ของอุปกรณ์ RS-485 โดยการใช้สัญญาณ RTS (Tri-state control of an RS-485 Device using RTS)

ในระบบ RS-485 เป็นการส่งผ่านข้อมูลผ่านสายส่งชุดเดียวกันโดยผลัดกันส่ง เมื่อมีเครื่องลูกข่ายอยู่หลายตัวที่จะทำการติดต่อสื่อสารกับตัวแม่จะติดต่อได้ทีละตัว เพราะถ้าติดต่อพร้อมกันจะเกิดการชนกันของข้อมูล ฉะนั้นเมื่ออุปกรณ์ตัวใดตัวหนึ่งต้องการส่งข้อมูลจะทำการเชื่อมต่อตัวส่งเข้ากับสายส่ง และจะตัดตัวส่งออกจากสายส่งเมื่อส่งข้อมูลเสร็จ โดยส่วนมากจะใช้อุปกรณ์ RS-485 เพื่อใช้เป็นตัวควบคุมสัญญาณการส่ง หรือเรียกว่า RTS : Request To Send โดยต่อจาก Asynchronous serial port ไปยังขา Enable ของตัวส่งผ่านสาย RTS และอาจกำหนดการทำงานของขา Enable โดยถ้าได้รับสถานะ High หรือลอจิก 1 ตัวส่งต่อเข้ากับสายส่งและทำการส่งข้อมูลได้ ถ้าได้รับสถานะ Low หรือลอจิก 0 ให้ตัวส่งตัดการต่อออกจากสายส่ง หรือเรียกว่า tri-state และยอมให้ตัวส่งอื่นที่ได้รับลอจิก 1 ต่อเข้ากับสายส่งและส่งข้อมูลผ่านสายส่งได้ ดังนั้นในขณะที่มีตัวส่งตัวเดียวได้รับลอจิก 1 และต่อเข้ากับสายส่งตัวส่งอื่นๆ ที่เหลือจะต้องได้รับลอจิก 0



รูปที่ 2.6 แผนภาพแสดงการเปลี่ยนแปลงระดับสัญญาณ RS-232 เป็น RS-485 โดยใช้ RTS ควบคุมการส่งและรับของ RS-485

รูปที่ 2.6 แสดง Timing diagram ของตัวเปลี่ยน RS-232 เป็น RS-485 (RS-232 to RS-485 Converter) จากรูปคลื่น (waveform) แสดงให้เห็นถึงความผิดพลาดเนื่องจากสัญญาณที่ควบคุมการส่งหรือสัญญาณที่ส่งให้กับขา Enable หรือ VRTS ที่มีขนาดแคบกว่าสัญญาณข้อมูล (VSD) มีผลทำให้เกิดการสูญหายของข้อมูลในส่วนหลังนับจากที่สัญญาณ VRTS อยู่ในสถานะ low ดังนั้นต้องมั่นใจว่าสัญญาณควบคุมหรือ VRTS ต้องมีขนาดกว้างกว่าสัญญาณข้อมูล (VSD) หรือกล่าวได้ว่าสัญญาณ RTS จะต้องเป็น

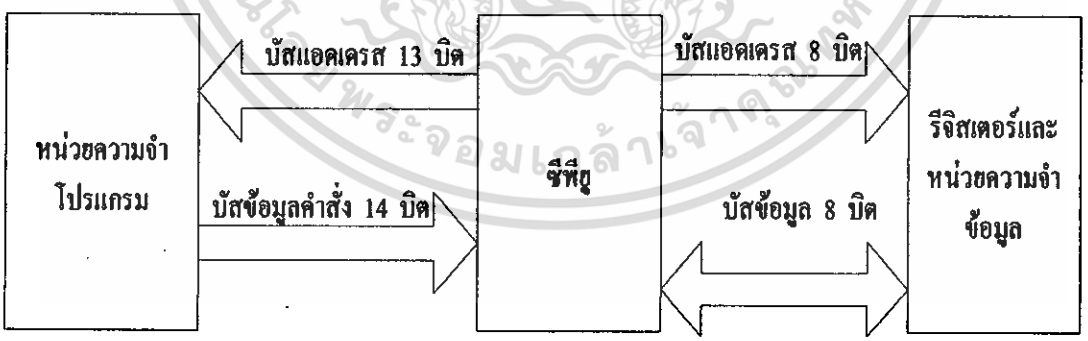
High ก่อนข้อมูลจะถูกส่งและจะต้องเป็น Low หลังจากส่งบิตสุดท้ายแล้ว ซึ่งช่วงเวลาการทำงานดังกล่าวนี้จะถูกทำโดย Software ที่ทำหน้าที่ควบคุมผ่านพอร์ตอนุกรมในการ์ดอนุกรม RS-485

ดังนั้นในการต่ออุปกรณ์หรือการเชื่อมต่อระบบเครือข่าย RS-485 ในลักษณะ 2 สายที่ตัวรับแต่ละตัวจะถูกต่อเข้ากับสายส่งข้อมูลดังในรูปที่ 2.6 จำเป็นต้องศึกษาถึงวิธีการจัดการในส่วนของ Enable function ของแต่ละตัวอุปกรณ์

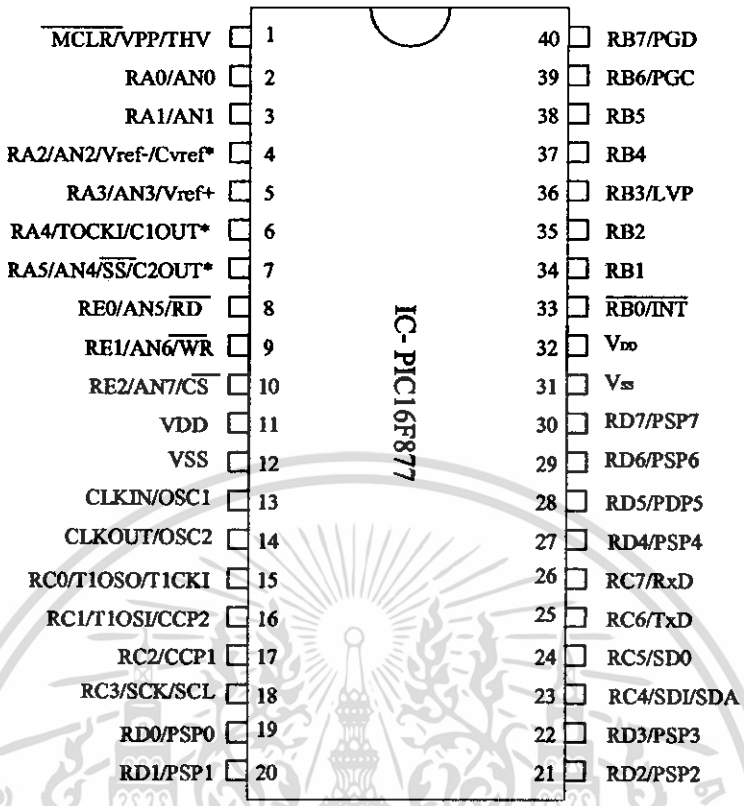
2.5 โครงสร้างทางฮาร์ดแวร์ของไมโครคอนโทรลเลอร์ PIC16F877

ไมโครคอนโทรลเลอร์ตระกูล PIC มีสถาปัตยกรรมแบบฮาร์วาร์ด (Harvard architecture) กล่าวคือ มีการแยกหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลออกจากกัน โดยมีบัสสำหรับติดต่อแยกกันด้วย ดังแสดงในรูปที่ 2.7 จะเห็นว่าซีพียูภายในไมโครคอนโทรลเลอร์จะติดต่อกับหน่วยความจำโปรแกรมด้วยบัสแอดเดรส 13 บิต และบัสข้อมูลหน่วยความจำโปรแกรม 14 บิต ในขณะที่บัสสำหรับติดต่อกับหน่วยความจำข้อมูลและรีจิสเตอร์ภายในเป็นแบบ 8 บิตทั้งบัสแอดเดรสและบัสข้อมูล

นอกจากการจัดสถาปัตยกรรมแบบนี้แล้ว การกระทำคำสั่งทำงานของไมโครคอนโทรลเลอร์ PIC ดิวต์คำสั่งในปัจจุบัน ส่งผลให้ความเร็วในการทำงานของไมโครคอนโทรลเลอร์เพิ่มมากขึ้น นั่นจึงเป็นที่มาของความสามารถในการกระทำคำสั่ง 1 คำสั่งภายในสัญญาณนาฬิกา 1 ลูก (กระบวนการเฟตช์ (fetch) เป็นกระบวนการเรียกคำสั่งออกจากหน่วยความจำโปรแกรมแล้วทำการแปลคำสั่งนั้นให้เป็นรหัสเลขฐานสิบหกเพื่อให้ซีพียูเข้าใจ ส่วนกระบวนการเอ็กซิคิวต์ (execute) เป็นการกระทำคำสั่งให้เกิดผลลัพธ์ตามที่คำสั่งนั้นๆ กำหนด



รูปที่ 2.7 ไดอะแกรมแสดงรูปแบบสถาปัตยกรรมของไมโครคอนโทรลเลอร์แบบฮาร์วาร์ด



รูปที่ 2.8 ตำแหน่งขาต่างๆของไมโครคอนโทรลเลอร์ PIC 16F877

คุณสมบัติทางเทคนิคของ PIC16F877

คุณสมบัติหลัก

- ซีพียูเป็นแบบ RISC (Reduced Instruction-Set Computer) มีคำสั่งใช้งานเพียง 35 คำสั่ง
- สามารถกระทำคำสั่งโดยใช้สัญญาณเพียงหนึ่งลูก ยกเว้นคำสั่งการกระโดด
- ความถี่สัญญาณนาฬิกาสูงสุดถึง 20MHz
- หน่วยความจำโปรแกรม 8 คิโลเวิร์ด สำหรับ PIC16F877
- หน่วยความจำข้อมูลแรมหรือรีจิสเตอร์ 368 ไบต์ สำหรับ PIC16F877
- ขนาดหน่วยความจำข้อมูลอีอีพรอม 256 ไบต์ สำหรับ PIC16F877
- ตอบสนองแหล่งกำเนิดอินเตอร์รัปต์สูงสุดถึง 15 แหล่ง ขึ้นกับเบอร์ของไมโครคอนโทรลเลอร์
- มีสแต็ก 8 ระดับ
- มีวงจรเพาเวอร์ออนรีเซต (POR)
- มีเพาเวอร์อัปไทเมอร์ (PWRT) และออสซิลเลเตอร์สตาร์ทอัปไทเมอร์ (OST)
- มีวงจรวอตช์ด็อกไทเมอร์ (WDT) ที่มีวงจรออสซิลเลเตอร์ในตัว ทำให้มีความน่าเชื่อถือในการทำงานสูงสุด

- เลือกป้องกันข้อมูลทั้งในหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลสามารถเลือกระดับการป้องกันได้
- มีโหมดประหยัดพลังงาน
- สามารถโปรแกรมโดยใช้แรงดัน +5 โวลต์ ได้
- แก้ไขข้อมูลในหน่วยความจำโปรแกรมด้วยกระบวนการ ICD (In-circuit Debugger) ผ่านพอร์ตเพียง 2 ขา
- ซีพียูสามารถอ่านและเขียนหน่วยความจำโปรแกรมได้
- ไฟเลี้ยง +2 โวลต์ ถึง +5.5โวลต์
- กระแสซิงค์และซอร์สของพอร์ต 25 มิลลิโวลต์
- การใช้พลังงานไฟฟ้าในกรณีไม่ขับโหลด

  น้อยกว่า 2 มิลลิแอมป์ ที่ไฟเลี้ยง +5 โวลต์ และสัญญาณนาฬิกา 4 MHz

  20 ไมโครแอมป์ที่ไฟเลี้ยง +3 โวลต์ และสัญญาณนาฬิกา 32 KHz

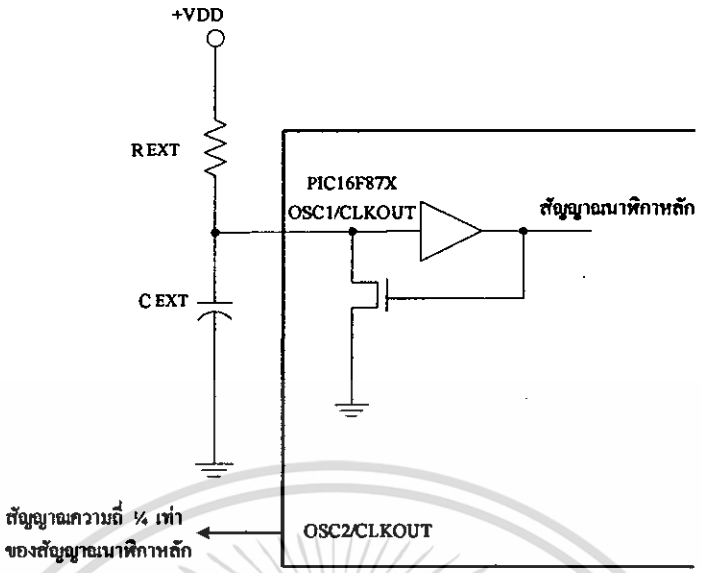
  น้อยกว่า 1 ไมโครแอมป์ ในโหมดประหยัดพลังงานหรือสแตนด์บาย

คุณสมบัติพิเศษ

- ไทเมอร์ 3 ตัว คือ ไทเมอร์ 0 ขนาด 8 บิต มีปรีสเกลเลอร์ขนาด 8 บิตในตัว, ไทเมอร์ 1 ขนาด 16 บิต พร้อมปรีสเกลเลอร์ และไทเมอร์ 2 ขนาด 8 บิต มีปรีสเกลเลอร์, โพสต์สเกลเลอร์ และรีจิสเตอร์คาบเวลา (period register) ขนาด 8 บิตในตัว
- มีโมดูล CCP 2 ชุด โดย

  ส่วนตรวจจับสัญญาณหรือแคปเจอร์ (Capture) มีขนาด 16 บิต ความละเอียดสูงสุด 12.5 นาโนวินาที

  ส่วนเปรียบเทียบสัญญาณ (Compare) มีขนาด 16 บิต ความละเอียดสูงสุด 200 นาโนวินาที
- มีวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล 10 บิต (5 ช่องสำหรับรุ่น 28 ขา และ 8 ช่องสำหรับรุ่น40 ขา)
- วงจรเชื่อมต่ออุปกรณ์อนุกรมทั้ง SPI และบัส I²C
- วงจรสื่อสารข้อมูลอนุกรม (USART) พร้อมการตรวจจับแอดเดรส 9 บิต
- มีวงจรตรวจจับระดับแรงดันไฟเลี้ยง (บราวเอาต์ดีเท็กชั่น : Brown-out detection) เพื่อการรีเซตซีพียู หรือเรียกว่า บราวเอาต์รีเซต (Brown-out reset : BOR)



รูปที่ 2.9 การต่อตัวต้านทานและตัวเก็บประจุที่ขา OSC1 เพื่อกำหนดความถี่สัญญาณนาฬิกาเมื่อไมโครคอนโทรลเลอร์ PIC16F877 ทำงานในโหมด RC

โหมดสัญญาณนาฬิกา

PIC16F877 สามารถเลือกโหมดของสัญญาณนาฬิกาเพื่อกำหนดจังหวะการทำงานได้มากถึง 4 โหมด โดยการกำหนดที่บิต FOSC0 และ FOSC1 ในรีจิสเตอร์ Configuration Word ซึ่งในการทำงานจะต้องเลือกโหมดใดโหมดหนึ่ง ดังมีรายละเอียดต่อไปนี้

1. โหมด LP ใช้กับคริสตอลหรือเซรามิกเรโซเนเตอร์พลังงานต่ำความถี่ 32 kHz – 200 kHz
2. โหมด XT ใช้กับคริสตอลหรือเซรามิกเรโซเนเตอร์มาตรฐานความถี่ 200 kHz – 4 MHz
3. โหมด HS ใช้กับคริสตอลหรือเซรามิกเรโซเนเตอร์ความถี่สูง 4 MHz – 20 MHz
4. โหมด RC (External Resistor/Capacitor) สามารถกำหนดค่าความถี่ได้จากค่าของตัวต้านทานและตัวเก็บประจุที่ต่อภายนอกเข้ากับขา OSC1/CLKIN อย่างไรก็ตามความถี่ของสัญญาณนาฬิกาในโหมดนี้ไม่อาจกำหนดลงไปได้อย่างชัดเจน เนื่องจากต้องพิจารณาถึงองค์ประกอบที่สามารถเปลี่ยนแปลงได้ในขอบเขตที่กว้าง ไม่ว่าจะเป็นค่าของแรงดันไฟเลี้ยง , ค่าของต้านทานและตัวเก็บประจุ ซึ่งต้องรวมไปถึงความผิดพลาดของอุปกรณ์ทั้งสองด้วย อย่างไรก็ตามค่าของตัวต้านทานที่เหมาะสมอยู่ในย่าน 3 kΩ – 100 kΩ ส่วนค่าของตัวเก็บประจุควรมากกว่า 20 pF นอกจากนั้นที่ขา OSC1/CLKOUT จะมีสัญญาณนาฬิกาความถี่ 1/4 เท่า ของความถี่สัญญาณนาฬิกาหลักส่งออกมา

2.6 พอร์ตอินพุตเอาต์พุตของไมโครคอนโทรลเลอร์ PIC16F877

ไมโครคอนโทรลเลอร์ PIC16F877 มีพอร์ตให้ใช้งาน 5 พอร์ต จำนวน 33 บิต ความสามารถของพอร์ตใน PIC16F877 สามารถทำงานได้หลายอย่าง จึงจำเป็นอย่างยิ่งที่ผู้ใช้งานต้องทำความเข้าใจถึงโครงสร้างทางฮาร์ดแวร์และการกำหนดหรือเลือกฟังก์ชั่นการทำงานให้แก่ขาพอร์ตแต่ละขาด้วยกระบวนการทางซอฟต์แวร์ ทั้งนี้เพื่อให้สามารถใช้งานพอร์ตทั้งหมดของ PIC16F877 ได้อย่างมีประสิทธิภาพสูงสุด ต่อไปนี้จะกล่าวถึงภาพรวมของพอร์ตทั้งหมดตั้งแต่พอร์ต A ถึง E โดยจะเน้นไปที่โครงสร้างทางฮาร์ดแวร์และฟังก์ชั่นการทำงานในภาพรวม ความสามารถในการจ่ายกระแสเอาต์พุตของขาพอร์ตที่ไฟเลี้ยง +5 โวลต์ คือ 25 มิลลิแอมป์ ต่อขาทั้งกระแสซิงค์และกระแสซอร์ส ในขณะที่กระแสเอาต์พุตรวมของพอร์ต A,B และ E มีค่าสูงสุด 200 มิลลิแอมป์ ส่วนกระแสเอาต์พุตรวมของพอร์ต C และ D มีค่าสูงสุด 200 มิลลิแอมป์ ดังนั้นในการออกและเพื่อขับโหลดทางเอาต์พุตของขาพอร์ตต้องระวังเรื่องกระแสเอาต์พุตรวมที่ไมโครคอนโทรลเลอร์สามารถขับได้ด้วย

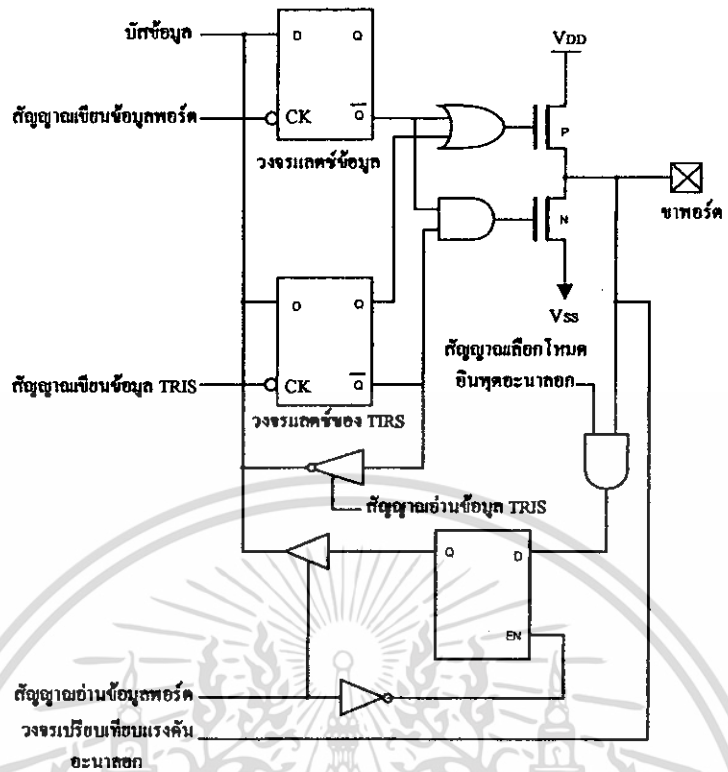
พอร์ต A

มีทั้งสิ้น 6 ช่องหรือ 6 บิต กำหนดชื่อขาเป็น RA0-RA5 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORTA มีแอดเดรสอยู่ที่ 0x05 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต แต่ใช้งานจริงเพียง 6 บิต ที่เหลือ 2 บิตต้องกำหนดให้เป็น "0" ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISA ซึ่งมีแอดเดรสอยู่ที่ 0x85 (แบงก์ 1) มีขนาด 8 บิตและใช้เพียง 6 บิตเช่นกัน 2 บิตบนคือบิต 6 และบิต 7 ต้องกำหนดให้เป็น "0" บิต 0 ของ TRISA ใช้กำหนดทิศทางของขาพอร์ต RA0 ไล่เรียงลำดับจนถึง บิต 5 ของ TRISA ใช้กำหนดทิศทางของขาพอร์ต RA5 หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้นและในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุตให้เขียนข้อมูล "0" ไปยังบิตนั้น

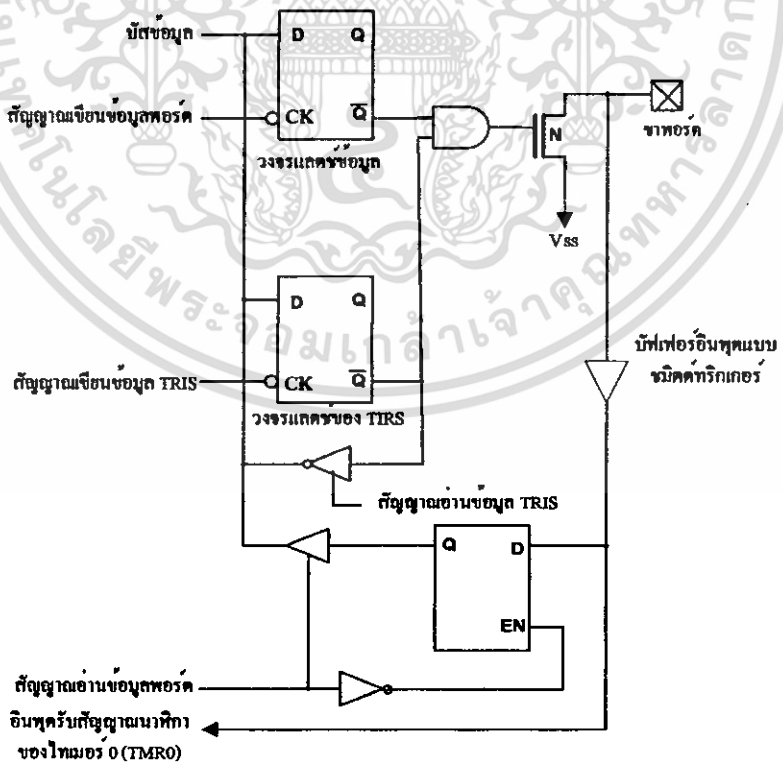
โครงสร้างทางฮาร์ดแวร์

พอร์ต A สามารถทำงานเป็นขาพอร์ตอินพุตเอาต์พุตปกติและเป็นขาอินพุตสัญญาณอะนาลอกสำหรับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลขนาด 10 บิตภายในไมโครคอนโทรลเลอร์โดยขา RA0-RA3 และ RA5 จะมีการทำงานที่เหมือนกัน ส่วน RA4 จะแตกต่างตรงที่ขานี้นอกจากเป็นขาพอร์ตอินพุตเอาต์พุตปกติแล้ว ยังใช้เป็นขาอินพุตสำหรับไทเมอร์ 0 ภายในไมโครคอนโทรลเลอร์ด้วย และขา RA4 นี้ไม่สามารถใช้งานเป็นขาอินพุตรับสัญญาณอะนาลอกได้

ในรูปที่ 2.10 โครงสร้างขา RA0-RA3 ของพอร์ต A จะเห็นได้ว่าที่ขาพอร์ตจะมีแอนด์เกตทำหน้าที่เลือกลักษณะการทำงานของขาพอร์ตเมื่อเป็นขาอินพุต หากเลือกให้ขาพอร์ตนี้เป็นขาพอร์ตอินพุตดิจิตอล สัญญาณเลือกโหมดอินพุตอะนาลอกจะต้องเป็นลอจิก "0" แต่ถ้าหากต้องการกำหนดให้เป็นขาอินพุตสัญญาณอะนาลอก สัญญาณเลือกโหมดต้องเป็นลอจิก "1" สัญญาณอะนาลอกที่ป้อนเข้ามายังขานี้จะผ่านเข้าสู่วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลหรือวงจรเปรียบเทียบแรงดันอะนาลอก (เฉพาะในอนุกรม PIC16F87xA) โดยตรง



รูปที่ 2.10 โครงสร้างขา RA0-RA3 ของพอร์ต A ในไมโครคอนโทรลเลอร์ PIC16F877

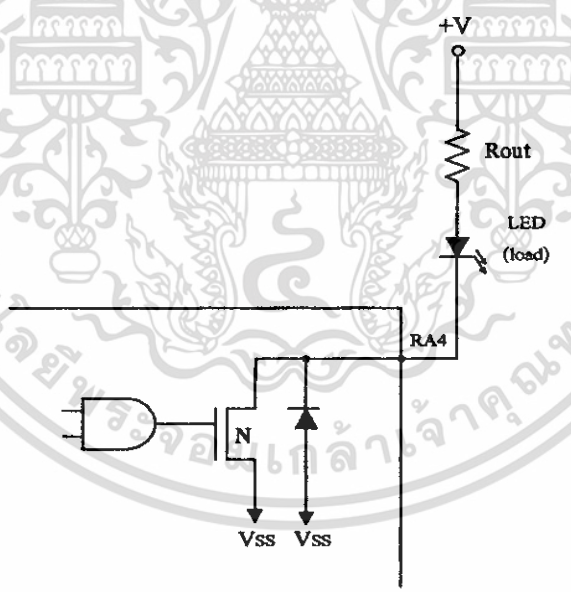


รูปที่ 2.11 โครงสร้างขา RA4 ของพอร์ต A ในไมโครคอนโทรลเลอร์ PIC16F877

เมื่อขาพอร์ต RA0-RA3 ทำงานเป็นขาพอร์ตอินพุตดิจิตอล จะสามารถรับสัญญาณดิจิตอลระดับทีทีแอล (0-5โวลต์) ได้โดยตรง หากทำงานเป็นเอาต์พุตจะสามารถขับโหลดที่ต้องการกระแส 20 มิลลิแอมป์ ได้ หากนำมาขับ LED ต้องต่อตัวต้านทานจำกัดกระแส หรือถ้าใช้ไฟเลี้ยง 3โวลต์ ก็จะขับ LED ได้โดยตรง

ในรูปที่ 2.11 เป็นโครงสร้างของขาพอร์ต RA4 โดยขานี้จะเป็นขาพอร์ตอินพุตเอาต์พุตปกติ และขาอินพุตรับสัญญาณนาฬิกาจากภายนอกของโมดูลไทเมอร์ 0 วงจรอินพุตบัฟเฟอร์ที่ขาพอร์ตนี้เป็นแบบชมิตต์ทริกเกอร์ ทั้งนี้เพื่อจัดการให้สัญญาณอินพุตที่เข้ามามีความสมบูรณ์มากที่สุด และจะต้องต่อตัวต้านทานพูลอัปค่าประมาณ 4.7k-10k ที่ขานี้เสมอเมื่อใช้งานเป็นอินพุต

สำหรับใน PIC16F87xA ขาพอร์ต RA4 ยังใช้เป็นขาเอาต์พุตของโมดูลเปรียบเทียบแรงดันอะนาลอกชุดที่ 1 (C1OUT) ด้วย โครงสร้างภายในจึงเปลี่ยนแปลงจากเดิมไปเล็กน้อย ดังในรูปที่ 2.11 ในกรณีที่ใช้งานเป็นพอร์ตเอาต์พุต วงจรเอาต์พุตจะเป็นแบบเดรนเปิด (open drain) ถ้าหากเทียบกับวงจรทรานซิสเตอร์ก็คือ วงจรคอลเล็กเตอร์เปิด (open collector) นั่นเอง ในการใช้งานจึงต่อตัวต้านทานอนุกรมกับโหลดและไฟเลี้ยงของโหลด ดังวงจรในรูปที่ 2.12 โดยปกติถ้าหากไม่มีความจำเป็นควรกำหนดให้ขา RA4/T0CKI นี้เป็นอินพุตจะดีที่สุด โดยต้องต่อตัวต้านทานพูลอัปเสมอ หรือถ้าหากจำเป็นต้องใช้เป็นเอาต์พุต ก็ต้องไม่ลืมว่า ขานี้เป็นเอาต์พุตแบบเดรนเปิด ต้องต่อขับโหลดในแบบกระแสซิงค์เท่านั้น



รูปที่ 2.12 การต่ออุปกรณ์เข้าที่ขาพอร์ต RA4 เมื่อใช้งานเป็นพอร์ตเอาต์พุต

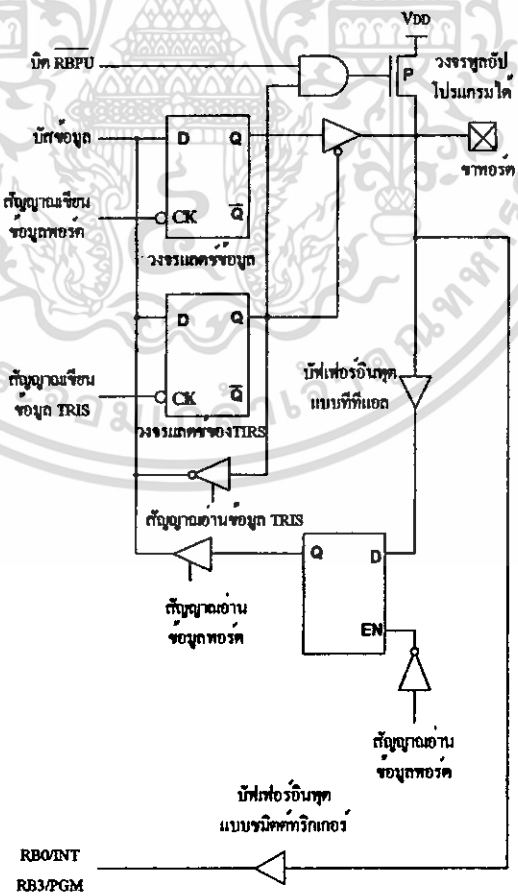
สำหรับใน PIC16F877A ขาพอร์ต RA4 ยังใช้เป็นขาเอาต์พุตของวงจรเปรียบเทียบแรงดันอะนาลอกชุดที่ 1 ด้วย โดยโครงสร้างของเอาต์พุตนี้ก็ยังคงเป็นแบบเดรนเปิดเช่นเดียวกับการใช้ขา RA4 เป็นพอร์ตดิจิตอลเอาต์พุตปกติ

พอร์ต B

มี 8 บิต กำหนดชื่อขาเป็น RB0-RB7 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORTB มีแอดเดรสอยู่ที่ที่ 0x06 (แบงก์ 0) และ 0x106 (แบงก์ 2) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISB ซึ่งมีแอดเดรสอยู่ที่ 0x86 (แบงก์ 1) และ 0x186 (แบงก์ 3) มีขนาด 8 บิต เช่นเดียวกับพอร์ต A บิต 0 ของ TRISA ใช้กำหนดทิศทางของขาพอร์ต RB7 หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้น ในทางตรงกันข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุตให้เขียนข้อมูล "0" ไปยังบิตนั้น

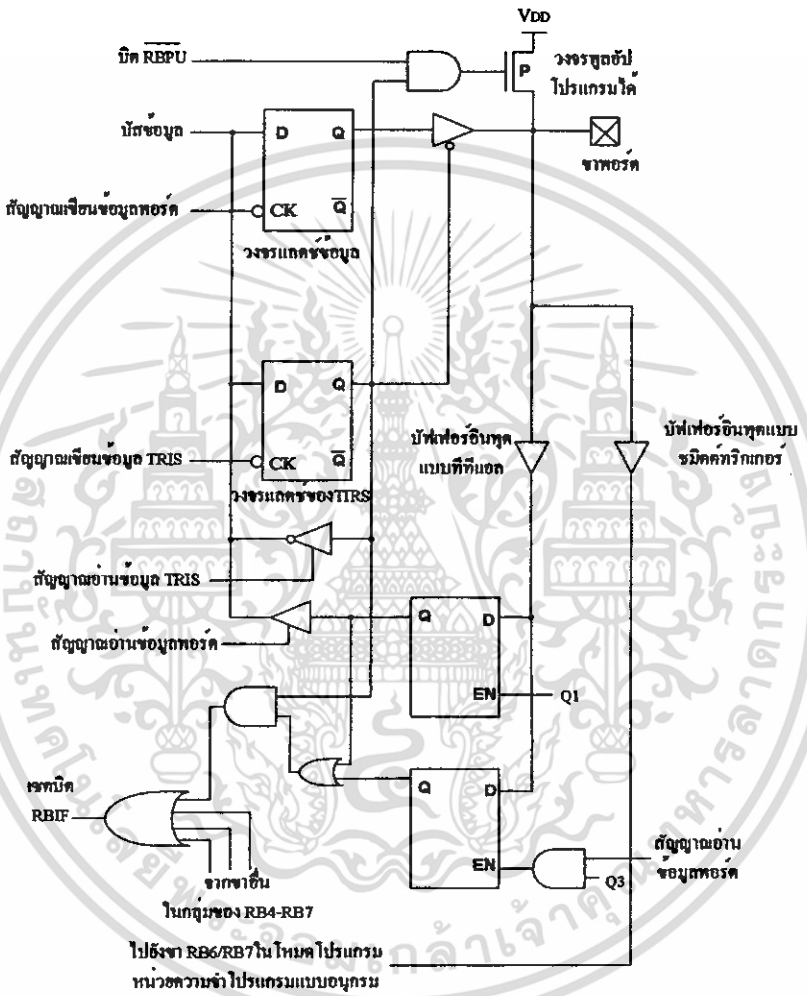
พอร์ต B สามารถใช้งานในลักษณะต่างๆ ได้ 5 แบบคือ

1. เป็นขาพอร์ตอินพุตเอาต์พุตปกติ
2. เป็นขาอินพุตสัญญาณอินเตอร์รัปต์จากภายนอก โดยใช้ขา RB0/INT
3. เป็นขาพอร์ตอินพุตสำหรับรับแรงดันโปรแกรมระดับต่ำ (low voltage programming) โดยใช้ขา RB3/PGM
4. เป็นขาข้อมูลอนุกรมและสัญญาณนาฬิกาอนุกรมสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์ ซึ่งใช้ 2 ขา คือ RB7/PGD และ RB6/PGC
5. ใช้เป็นแหล่งกำเนิดสัญญาณอินเตอร์รัปต์แบบตรวจสอบการเปลี่ยนแปลงข้อมูลหรือระดับสัญญาณที่ขา RB4-RB7



รูปที่ 2.13 โครงสร้างของขาพอร์ต B ในไมโครคอนโทรลเลอร์ PIC16F877 ขา RB0-RB3

จากรูปที่ 2.13 จะเห็นได้ว่าที่ขาพอร์ตจะมีวงจรพูลอัปแบบโปรแกรมได้ต่ออยู่ นั่นคือหากต้องการกำหนดให้เป็นขาอินพุต ต้องทำการเขียนข้อมูล "0" ไปยังบิต RBPU ในรีจิสเตอร์ OPTION_REG เพื่อเอ็นเอเบิลวงจรพูลอัปภายในขาพอร์ต B ในขณะที่หากกำหนดเป็นเอาต์พุต การพูลอัปที่ขาพอร์ต B นี้จะถูกยกเลิกโดยอัตโนมัติ นอกจากนั้นการพูลอัปนี้จะได้รับการยกเลิกเมื่อเกิดเพาเวอร์ออนรีเซตขึ้น ในกรณีที่ใช้ขา RB0/INT เพื่อรับสัญญาณอินเตอร์รัปต์จากภายนอก ซึ่งสัญญาณจะผ่านเข้าไปยังวงจรบัฟเฟอร์แบบชมิตต์ทริกเกอร์เพื่อให้สัญญาณที่ได้มีความแม่นยำและมีเสถียรภาพ



รูปที่ 2.14 โครงสร้างของขาพอร์ต B ในไมโครคอนโทรลเลอร์ PIC16F877 ขา RB4-RB7

ในรูปที่ 2.14โครงสร้างของขาพอร์ต RB4-RB7 โดยขาพอร์ตในกลุ่มนี้มีความสามารถพิเศษพอสมควร โดยสามารถเลือกให้ทำงานเป็นขาพอร์ตอินพุตเอาต์พุตปกติ, ขาอินพุตรับแรงดันสำหรับการโปรแกรม (RB3), ขาสัญญาณสำหรับการโปรแกรม (RB6-RB7) และทำงานเป็นแหล่งกำเนิดสัญญาณอินเตอร์รัปต์ในแบบตรวจสอบการเปลี่ยนแปลงที่ขาพอร์ต RB4-RB7 วงจรอินพุตบัฟเฟอร์ที่ขาพอร์ตนี้มีทั้งแบบทีทีแอลและชมิตต์ทริกเกอร์ ทั้งนี้เพื่อจัดการให้สัญญาณอินพุตที่เข้ามามีความเหมาะสมและสมบูรณ์มากที่สุด และยังคงสามารถรองรับการพูลอัปภายในแบบอัตโนมัติได้ ในกรณีที่มีการเอ็นเอเบิลการตอบสนองอินเตอร์รัปต์แบบตรวจสอบการเปลี่ยนแปลงลอจิกที่พอร์ต RB4-RB7 หากเกิดการ

อินเตอร์รัปต์ขึ้น บิต RBIF (บิต 0 ในรีจิสเตอร์ INTCON) จะเซตและหลังจากตอบสนองการอินเตอร์รัปต์แล้ว ต้องเคลียร์บิต RBIF ด้วยกระบวนการทางซอฟต์แวร์เสมอ

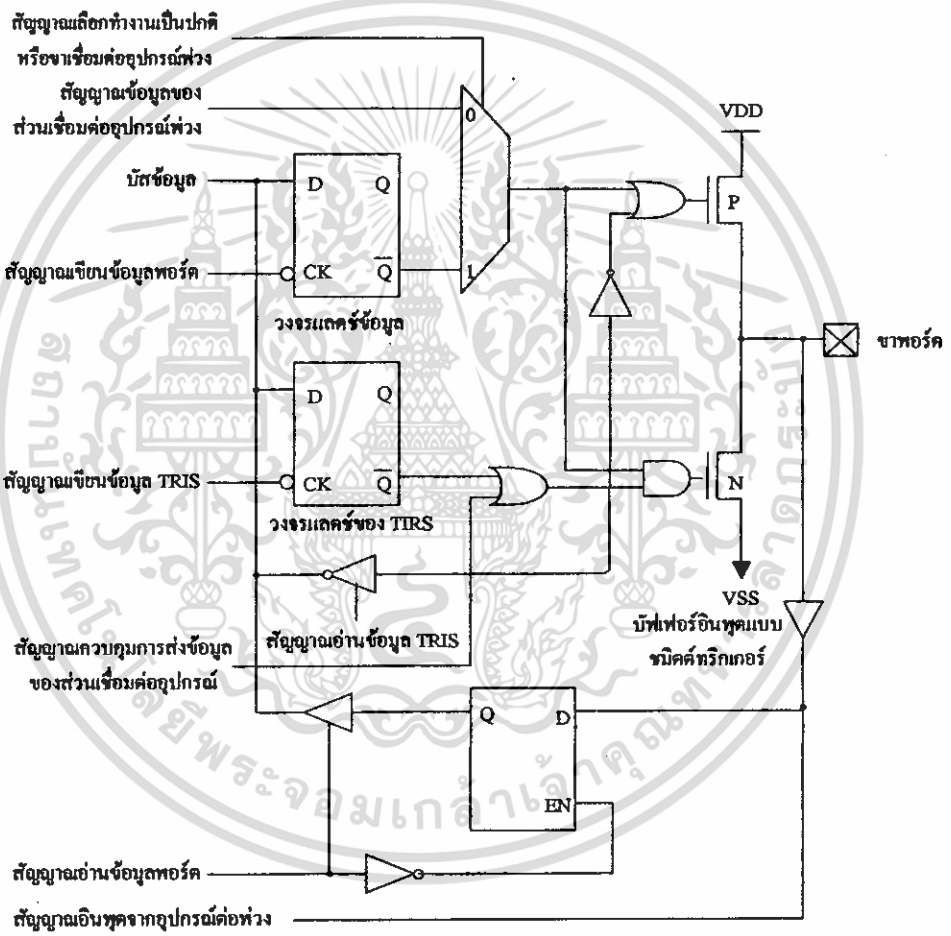
พอร์ต C

มีทั้งสิ้น 8 บิต กำหนดชื่อขาเป็น RC0-RC7 รีจิสเตอร์ที่ใช้เก็บข้อมูลคือ PORTC มีแอดเดรสอยู่ที่ 0x07 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISC มีแอดเดรสอยู่ที่ 0x87 (แบงก์ 1) มีขนาด 8 บิตเช่นเดียวกับพอร์ต A และ B บิต 0 ของ TRISC ใช้กำหนดทิศทางของขาพอร์ต RC0 ไล่เรียงลำดับจนถึง บิต 7 ของ TRISC ใช้กำหนดทิศทางของขาพอร์ต RC7 หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียนข้อมูล "1" ไปยังบิตนั้นและในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุตให้เขียนข้อมูล "0" ไปยังบิตนั้น

ชื่อขา	การทำงาน
RC0/T1OSO/T1CKI	ขาพอร์ตอินพุตเอาต์พุตบิต 0 ของพอร์ต C เอาต์พุตวงจรออสซิลเลเตอร์ของไทเมอร์ 1 (T1OSO) อินพุตรับสัญญาณนาฬิกาของไทเมอร์ 1 (T1CKI)
RC1/T1OSI/CCP2	ขาพอร์ตอินพุตเอาต์พุตบิต 1 ของพอร์ต C อินพุตวงจรออสซิลเลเตอร์ของไทเมอร์ 1 (T1OSI) เอาต์พุตวงจรเปรียบเทียบของโมดูล CCP2
RC2/CCP1	ขาพอร์ตอินพุตเอาต์พุตบิต 2 ของพอร์ต C อินพุตแคปเจอร์หรือวงจรตรวจจับสัญญาณของโมดูล CCP1 เอาต์พุตวงจรเปรียบเทียบของโมดูล CCP1
RC3/SCK/SCL	ขาพอร์ตอินพุตเอาต์พุตบิต 3 ของพอร์ต C ขาสัญญาณนาฬิกาอนุกรมของระบบ SPI (SCK) ขาสัญญาณนาฬิกาอนุกรมของระบบ I²C (SCL)
RC4/SDI/SDA	ขาพอร์ตอินพุตเอาต์พุตบิต 4 ของพอร์ต C ขาข้อมูลอินพุตอนุกรมของระบบ SPI (SDI) ขาข้อมูลอนุกรมของระบบ I²C (SDA)
RC5/SDO	ขาพอร์ตอินพุตเอาต์พุตบิต 5 ของพอร์ต C ขาข้อมูลเอาต์พุตอนุกรมของระบบ SPI (SDO)
RC6/TxD/CK	ขาพอร์ตอินพุตเอาต์พุตบิต 6 ของพอร์ต C ขาส่งข้อมูลพอร์ตอนุกรม USART (TxD) ขาสัญญาณนาฬิกาซิงโครนัส (CK)
RC7/RxD/DT	ขาพอร์ตอินพุตเอาต์พุตบิต 7 ของพอร์ต C ขารับข้อมูลพอร์ตอนุกรม USART (TxD) ขาสัญญาณนาฬิกาซิงโครนัส (DT)

ตารางที่ 2.2 สรุปหน้าที่การทำงานของขาพอร์ต C ในไมโครคอนโทรลเลอร์ PIC16F877

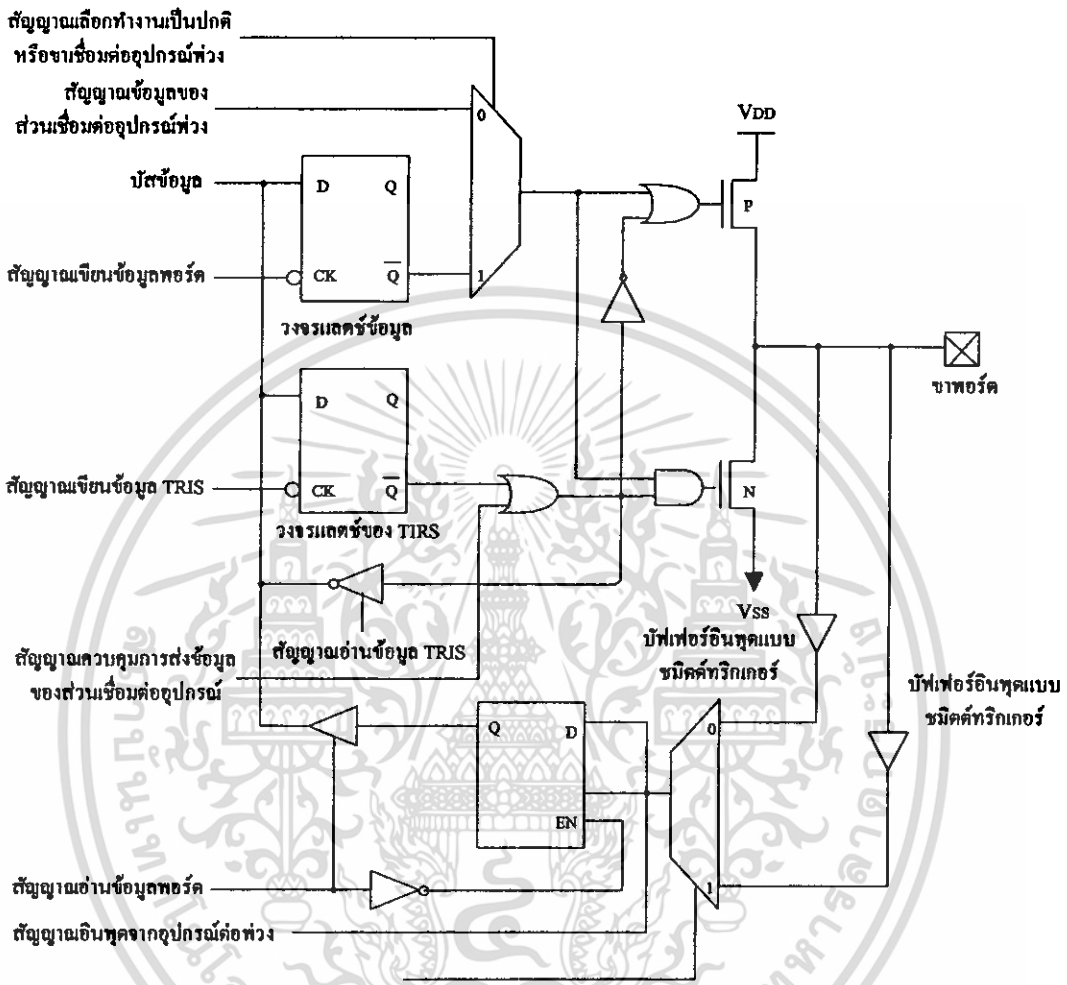
พอร์ต C สามารถใช้งานในลักษณะต่างๆ ได้หลายรูปแบบ และเป็นขาพอร์ตที่มีความสามารถสูงมาก ไม่ว่าจะเป็นขาพอร์ตอินพุตเอาต์พุตปกติ, ขาอินพุตเอาต์พุตออสซิลเลเตอร์ของโมดูลไทเมอร์1, ขาอินพุตสำหรับรับสัญญาณนาฬิกาของโมดูลไทเมอร์1, ขาเชื่อมต่อระบบบัส I²C, ขาเชื่อมต่อแบบ SPI (Serial Peripheral Interface), ขาเชื่อมต่อพอร์ตอนุกรมแบบ USART, ขาอินพุตวงจรแคปเจอร์ (capture) หรือวงจรตรวจจับสัญญาณ ขาเอาต์พุตของวงจรเปรียบเทียบสัญญาณ (compare) และขาเอาต์พุตวงจร PWM (Pulse Width Modulation) หรืออาจกล่าวได้ว่าพอร์ต C เป็นพอร์ตสำหรับเชื่อมต่ออุปกรณ์ภายนอก (peripheral function port) ที่มีความสมบูรณ์แบบมากที่สุด สามารถสรุปหน้าที่และการทำงานที่หลากหลายของขาพอร์ต C ดังในตารางที่ 2.2



รูปที่ 2.15 โครงสร้างขา RC0-RC2, RC5-RC7 ของพอร์ต C ในไมโครคอนโทรลเลอร์ PIC16F877

ในรูปที่ 2.15 แสดงโครงสร้างของพอร์ต C ในบิต RC0-RC2 และ RC5-RC7 จะเห็นได้ว่ามีสัญญาณควบคุมการทำงานของขาพอร์ตมากมาย ทั้งนี้เนื่องจากขาพอร์ต C สามารถทำงานได้หลากหลายนั่นเอง สัญญาณควบคุมที่สำคัญคือ สัญญาณเลือกการทำงานระหว่างเป็นพอร์ตปกติหรือเป็นขาเชื่อมต่ออุปกรณ์พิเศษ (PORT/PERIPERAL Select) และสัญญาณควบคุมการส่งข้อมูลของวงจรเชื่อมต่ออุปกรณ์

(Peripheral Output Enable) สำหรับข้อมูลของวงจรเชื่อมต่ออุปกรณ์ที่ส่งออกและรับเข้ามาจะผ่านทางขาพอร์ตปกติ แต่เมื่อผ่านวงจรสำหรับเลือกสัญญาณข้อมูลแล้ว สายสัญญาณข้อมูลของพอร์ต (data bus) กับข้อมูลของวงจรเชื่อมต่ออุปกรณ์ (peripheral output/peripheral input) จะแยกกัน



รูปที่ 2.16 โครงสร้างขา RC3 และ RC4 ของพอร์ต C ในไมโครคอนโทรลเลอร์ PIC16F877

ในรูปที่ 2.16 เป็นโครงสร้างขาพอร์ต RC3และ RC4 ทั้ง 2 ขานี้มีความพิเศษตรงที่สามารถใช้งานเป็นขาเชื่อมต่ออุปกรณ์อนุกรมแบบซิงโครนัส ซึ่งแบ่งเป็นระบบ SPI และระบบบัส I²C จึงทำให้ต้องเพิ่มสายสัญญาณควบคุมอินพุตเพิ่มเข้ามาอีก 1 เส้น เพื่อเลือกสัญญาณอินพุตระหว่าง SPI และบัส วงจรอินพุตบัฟเฟอร์ของขาพอร์ต C นี้เป็นแบบชมิตต์ทริกเกอร์ทั้งหมด ทั้งนี้เพื่อจัดการให้สัญญาณอินพุตที่เข้ามามีความเหมาะสมและสมบูรณ์มากที่สุด และยังคงสามารถรองรับการพูลอัปภายในแบบอัตโนมัติ

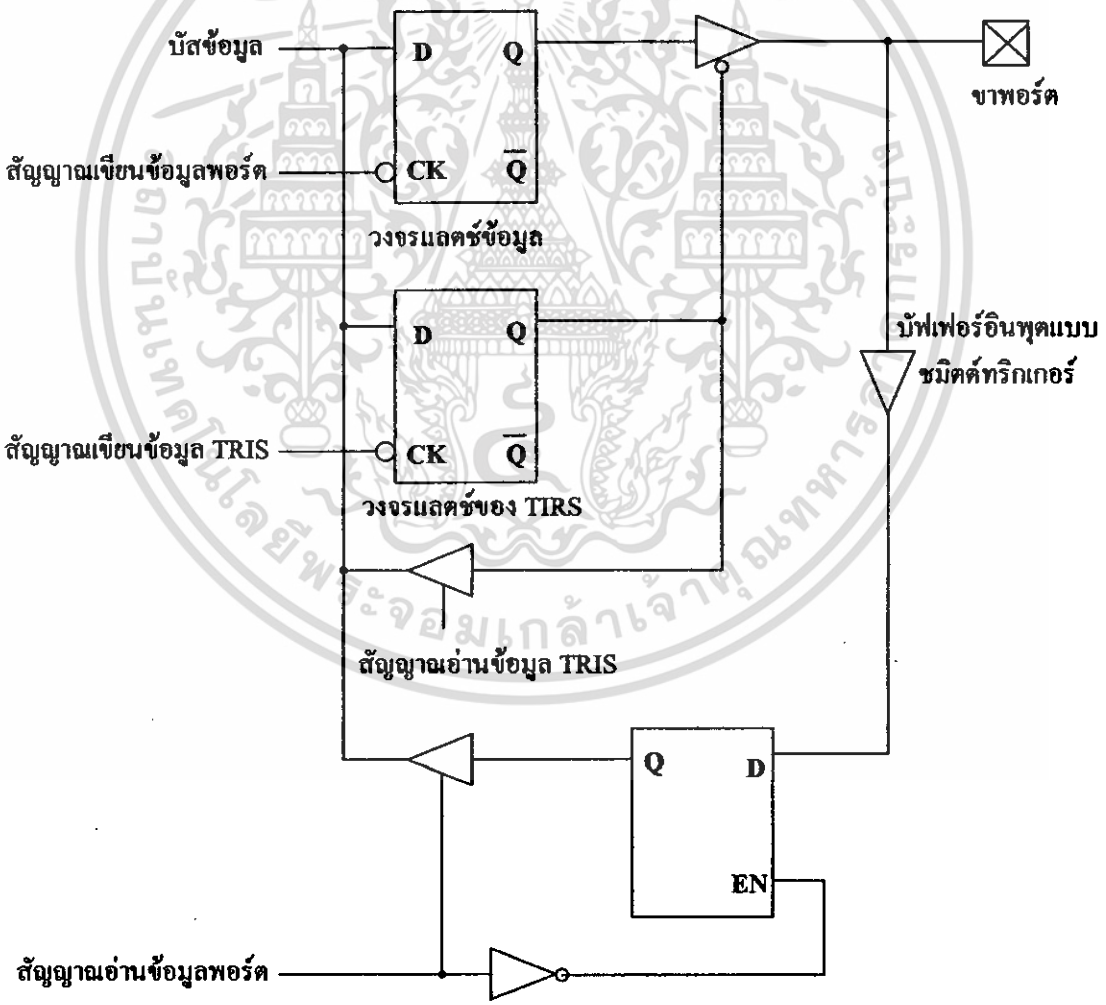
พอร์ต D

มี 8 บิต กำหนดชื่อขาเป็น RD0-RD7 รีจิสเตอร์ที่ใช้เก็บข้อมูลคือ PORTD มีแอดเดรสอยู่ที่ 0x08 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISD มีแอดเดรสอยู่ที่ 0x88 (แบงก์ 1) มีขนาด 8 บิต หากต้องการกำหนดให้ขาพอร์ตในบิตใดเป็นอินพุตต้องเขียน

ข้อมูล "1" ไปยังบิตนั้นและในทางตรงข้ามหากต้องการกำหนดให้เป็นขาเอาต์พุตให้เขียนข้อมูล "0" ไปยังบิตนั้น สำหรับพอร์ต D นี้ จะมีเฉพาะในไมโครคอนโทรลเลอร์ PIC รุ่น 40 ขาเท่านั้น สำหรับในอนุกรม PIC16F87x จะมีในเบอร์ PIC16F871, PIC16F874(A) และ PIC16F877(A)

พอร์ต D สามารถใช้งานได้ 2 โหมดคือ เป็นขาพอร์ตอินพุตเอาต์พุตปกติและเป็นส่วนขยายพอร์ตแบบขนาน (Parallel Slave Port : PSP) สำหรับเชื่อมต่อกับอุปกรณ์ภายนอกที่มีการจัดระบบบัสแบบไมโครโปรเซสเซอร์คือ มีสายข้อมูล 8 เส้น สายสัญญาณควบคุม 3 เส้นคือ สายสัญญาณควบคุมการอ่าน (RD : Read), เขียน (WR : Write) และเลือกอุปกรณ์ (CS : Chip Select)

รูปที่ 2.17 ไดอะแกรมของขาพอร์ต D ซึ่งมีโครงสร้างเหมือนกันทุกบิต เมื่อทำงานในโหมดพอร์ตอินพุตเอาต์พุตปกติ วงจรอินพุตจะเป็นแบบชมิตต์ทริกเกอร์ แต่เมื่อทำงานในโหมดขยายพอร์ตแบบขนานหรือ PSP วงจรอินพุตจะเปลี่ยนเป็นแบบทีทีแอล การเลือกโหมดทำงานของพอร์ต D นี้ขึ้นกับบิต PSP MODE (บิต 4 ในรีจิสเตอร์ TRISE) ถ้าเป็น "0" เป็นการกำหนดให้พอร์ต D เป็นพอร์ตปกติ และถ้าเป็น "1" พอร์ต D จะทำงานในโหมด PSP

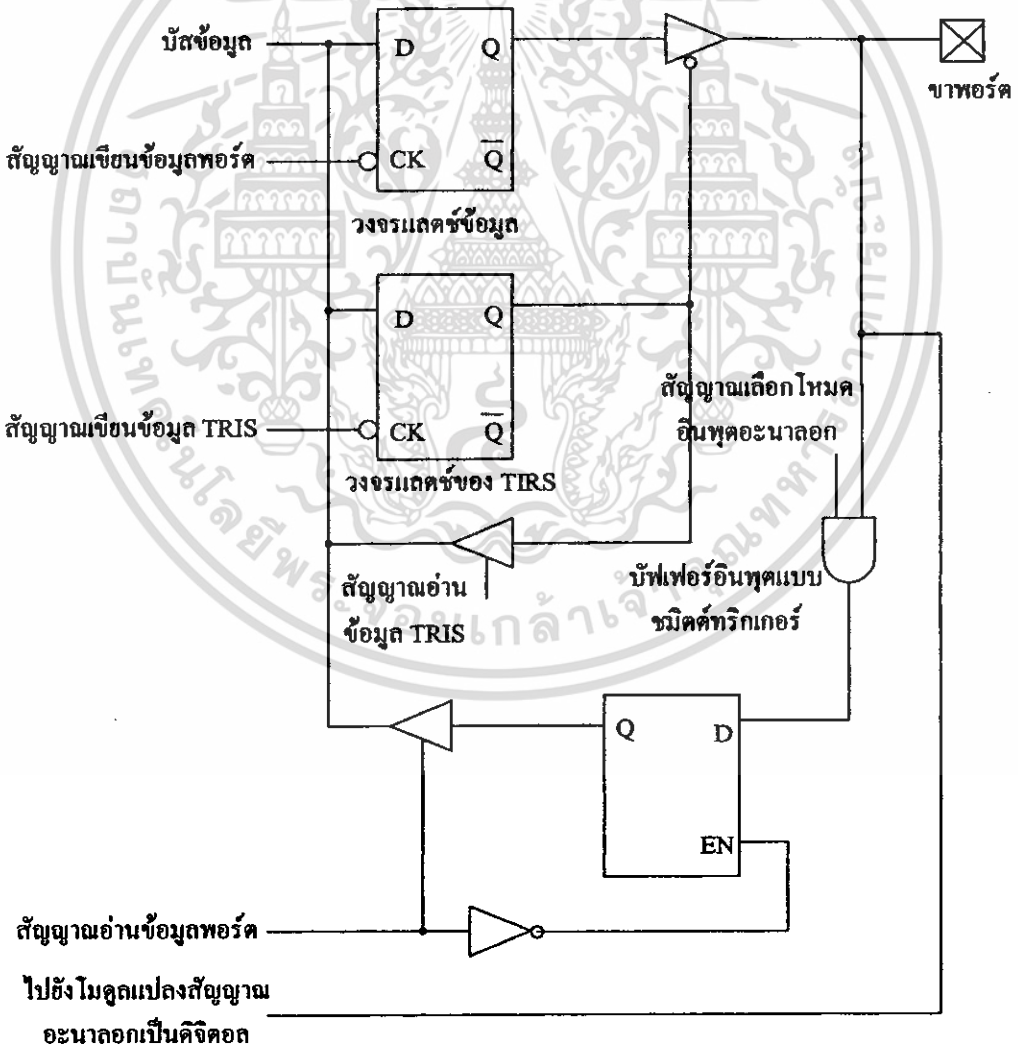


รูปที่ 2.17 โครงสร้างของพอร์ต D ในไมโครคอนโทรลเลอร์ PIC16F877

พอร์ต E

มี 3 บิต กำหนดชื่อขาเป็น RE0-RE2 รีจิสเตอร์ที่ใช้ในการเก็บข้อมูลคือ PORTE มีแอดเดรสอยู่ที่ 0x09 (แบงก์ 0) เป็นรีจิสเตอร์ขนาด 8 บิต แต่ใช้งานเพียง 3 บิตล่างคือ บิต 0 – บิต 2 เท่านั้น ที่เหลือกำหนดให้เป็น "0" ส่วนการกำหนดทิศทางของพอร์ตนี้กระทำผ่านรีจิสเตอร์ TRISE ซึ่งมีแอดเดรสอยู่ที่ 0x89 (แบงก์ 1) มีขนาด 8 บิต โดยใช้สามบิตล่างในการกำหนดทิศทางของพอร์ต E ส่วนที่เหลือใช้ควบคุมการทำงานในโหมด PSP ของพอร์ต D

พอร์ต E สามารถใช้งานเป็นพอร์ตอินพุตเอาต์พุตปกติ, ขาอินพุตอะนาลอกของโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล และขาควบคุมการติดต่อกับอุปกรณ์ภายนอกแบบ PSP ทั้งนี้ขึ้นอยู่กับการกำหนดข้อมูลของรีจิสเตอร์ที่ใช้ควบคุมการทำงานของพอร์ตนี้ เช่นเดียวกับพอร์ต D พอร์ต E จะมีเฉพาะในไมโครคอนโทรลเลอร์ PIC รุ่น 40 ขาเท่านั้น สำหรับในอนุกรม PIC 16F87x จะมีในเบอร์ PIC16F871, PIC16F874(A) และ PIC16F877(A)



รูปที่ 2.18 โครงสร้างของพอร์ต E ในไมโครคอนโทรลเลอร์ PIC16F877

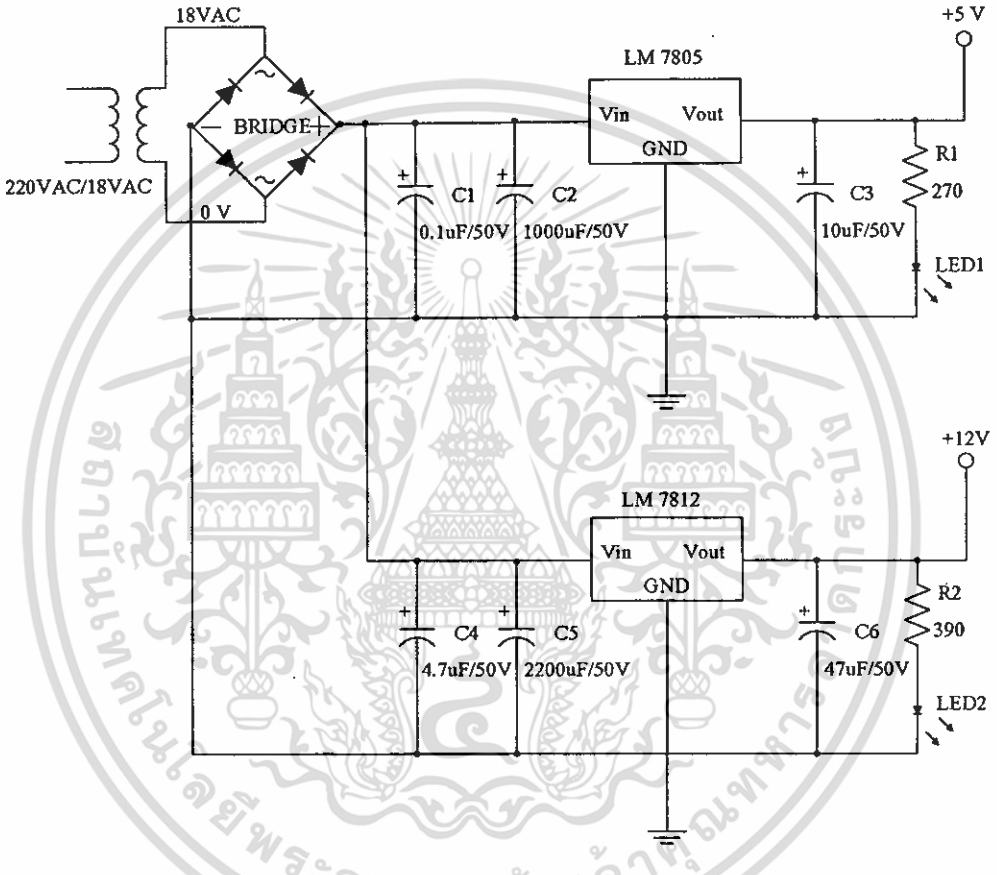
บทที่ 3
การคำนวณและการสร้าง

การคำนวณและการสร้างส่วนของฮาร์ดแวร์

3.1 วงจรกำเนิดแรงดัน

ในส่วนของวงจรกำเนิดแรงดันจะแบ่งออกเป็นสองส่วนดังนี้

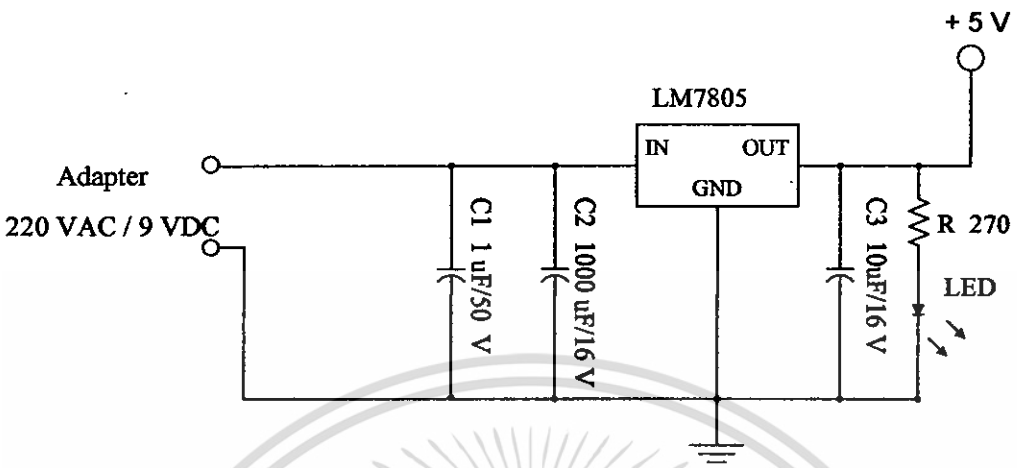
3.1.1 วงจรกำเนิดแรงดันของวงจรไมโครคอนโทรลเลอร์



รูปที่ 3.1 วงจรกำเนิดแรงดันของวงจรไมโครคอนโทรลเลอร์

การทำงานของวงจรเริ่มจากการรับแรงดันไฟฟ้ากระแสสลับ 220โวลต์เข้าที่ขดลวดทรานฟรอมเมอร์ ซึ่งมีการลดระดับแรงดันไฟฟ้าให้เหลือแรงดันไฟกระแสสลับ 18 โวลต์แล้วผ่านวงจรบริดจ์ (Bridge) เพื่อทำหน้าที่แปลงจากแรงดันไฟฟ้ากระแสสลับให้เป็นแรงดันไฟฟ้ากระแสตรง โดยมีตัวเก็บประจุเป็นตัวกรองแรงดันไฟให้คงที่ซึ่งต่ออยู่กับ LM7805 ที่ทำหน้าที่ปรับระดับแรงดันให้ได้ +5 โวลต์ หลังจากนั้นจะมีตัวเก็บประจุเป็นตัวกรองแรงดันไฟให้คงที่อีกครั้ง จากนั้นเอาต์พุตจะได้แรงดันไฟ +5 โวลต์ ส่วนของแรงดันไฟ +12 โวลต์นั้นมีลักษณะการทำงานเหมือนกับแรงดันไฟ +5 โวลต์ต่างกันที่ไอซีเร็กกูเลท (IC Regulate) จะใช้เบอร์ LM7812 เอาต์พุตทำให้ได้แรงดันไฟ +12 โวลต์

3.1.2 วงจรกำเนิดแรงดันของวงจรแปลงสัญญาณ RS-485

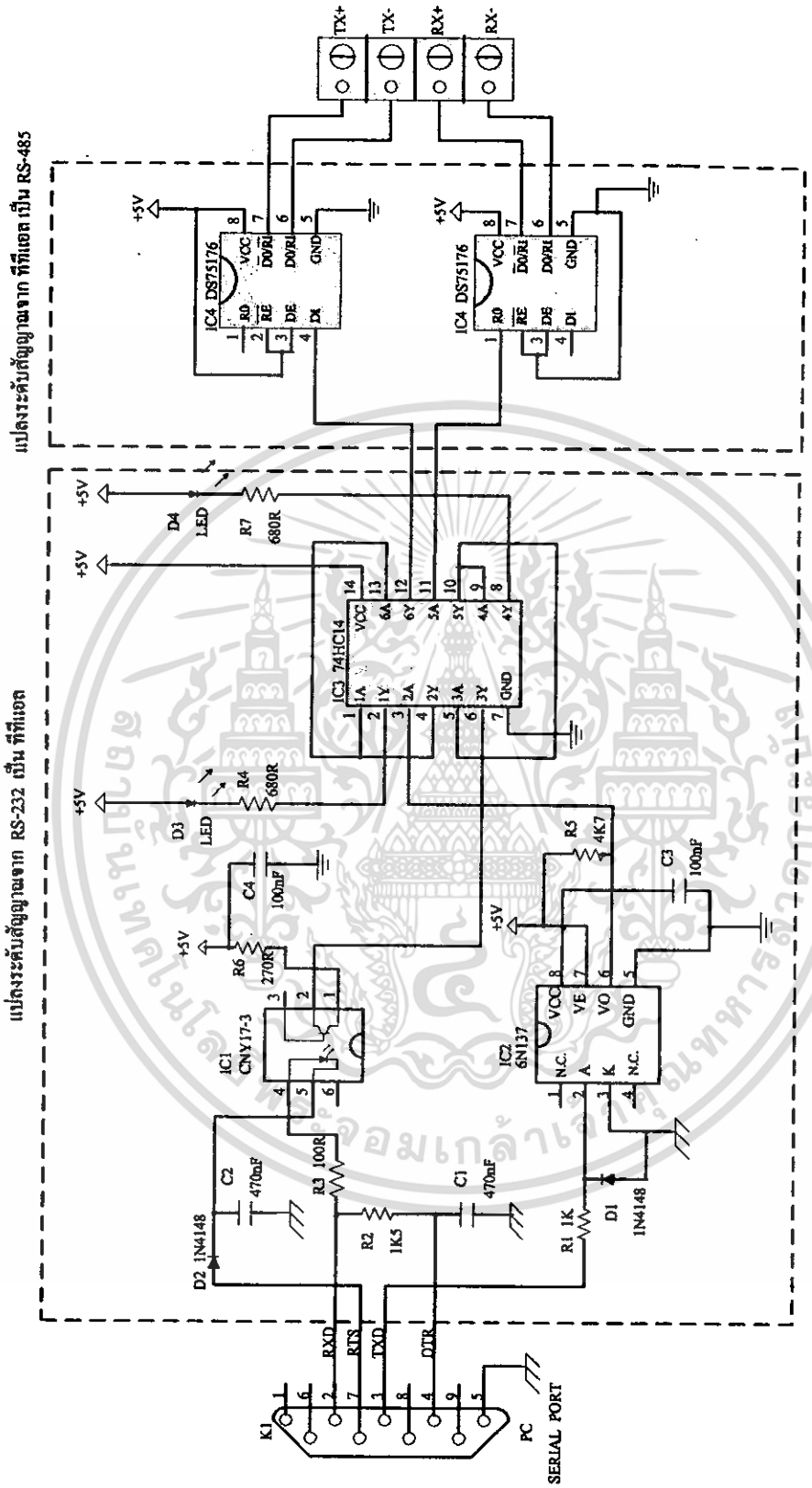


รูปที่ 3.2 วงจรกำเนิดแรงดันของวงจรแปลงสัญญาณ RS-485

การทำงานของวงจรเริ่มจาก มีการรับแรงดันไฟฟ้ากระแสสลับ 220 โวลต์ เข้าที่อแดปเตอร์ (Adapter) ซึ่งทำหน้าที่ในการแปลงระดับแรงดันไฟฟ้าให้เหลือแรงดันไฟฟ้ากระแสตรง +9 โวลต์ จากนั้นจะผ่านตัวเก็บประจุซึ่งจะทำหน้าที่เป็นตัวกรองแรงดันไฟให้คงที่แล้วส่งต่อไปยัง LM7805 เพื่อทำการปรับลดระดับแรงดันไฟฟ้าให้เหลือ +5 โวลต์ หลังจากนั้นจะมีตัวเก็บประจุเป็นตัวกรองแรงดันไฟให้คงที่อีกครั้ง แล้วจึงส่งออกเอาต์พุต

3.2 วงจรแปลงสัญญาณ RS-232 เป็นสัญญาณ RS-485

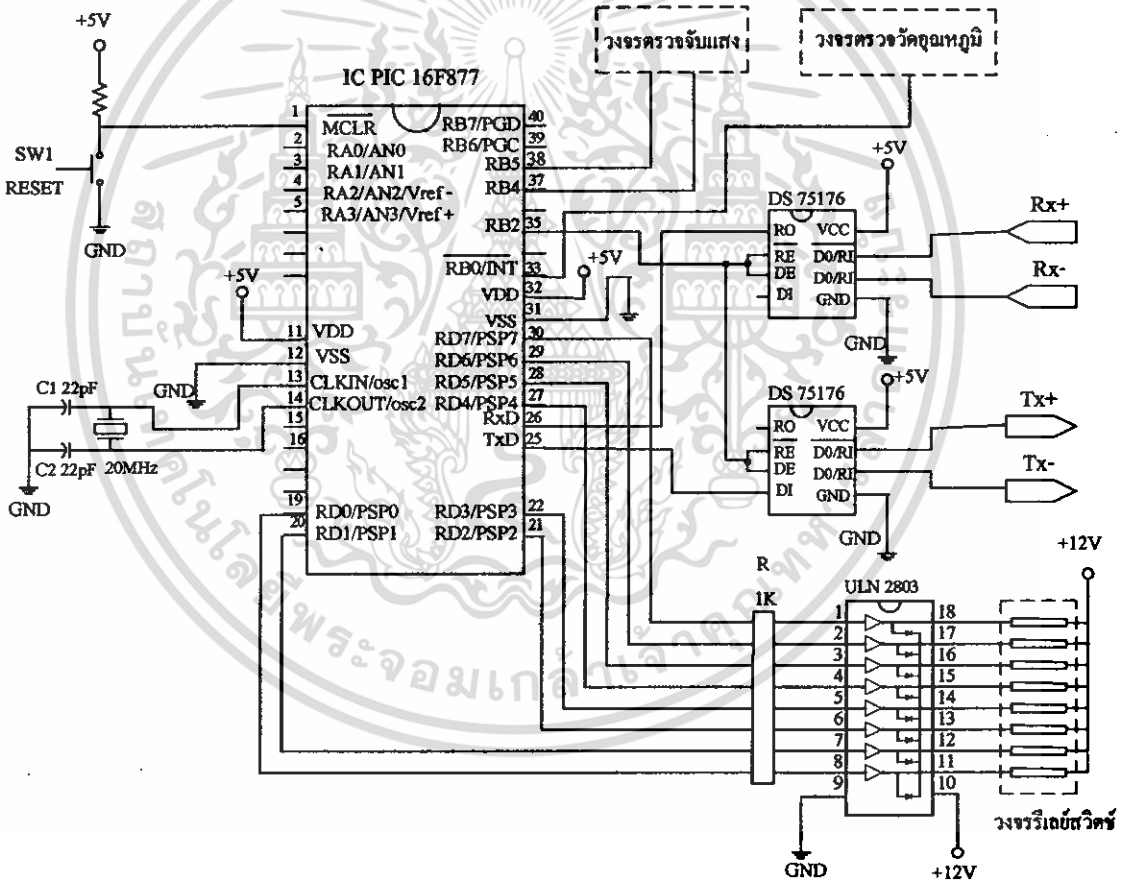
ในวงจรแปลงสัญญาณ RS-232 เป็นสัญญาณ RS-485 จะเป็นวงจรแบบแยกกราวด์จากคอมพิวเตอร์ออกจากกราวด์ของวงจร ทั้งนี้เพื่อป้องกันสัญญาณรบกวนจากคอมพิวเตอร์ การทำงานของวงจรจะแบ่งออกเป็น 2 ส่วน โดยในส่วนแรกจะเป็นการแปลงสัญญาณ RS-232 เป็น ทีทีแอล ก่อน จากนั้นส่วนที่สองก็จะทำการแปลงจากสัญญาณ ทีทีแอล ให้เป็นสัญญาณ RS-485 ซึ่งส่วนนี้จะใช้ไอซี DS-75176 เป็นตัวทำหน้าที่ในการแปลงสัญญาณจากทีทีแอลให้เป็นสัญญาณ RS-485 โดยจะใช้ด้วยกันทั้งหมด สองตัว ตัวแรกจะทำหน้าที่ในการแปลงสัญญาณทางด้านฝั่งส่งเพียงอย่างเดียวและตัวที่สองจะทำหน้าที่ในการแปลงสัญญาณทางด้านฝั่งรับเพียงอย่างเดียว สัญญาณ RS-485 ที่ได้ทางด้านเอาต์พุตนั้น จะเป็นการต่อแบบ 4 เส้น คือด้านส่งจะมีสองเส้นได้แก่ Tx+ และ Tx- ทางด้านรับจะมีอีกสองเส้นคือ Rx+ และ Rx-



รูปที่ 3.3 วงจรแปลงสัญญาณ RS-232 เป็นสัญญาณ RS-485

3.3 วงจรไมโครคอนโทรลเลอร์

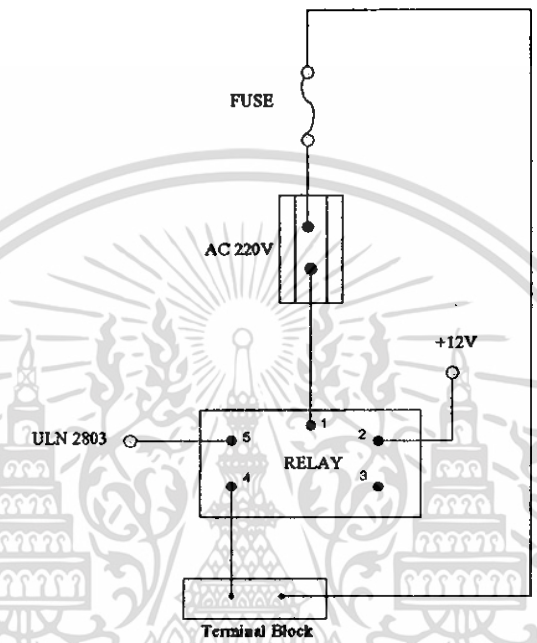
วงจรไมโครคอนโทรลเลอร์ เป็นวงจรที่ใช้ควบคุมการทำงานของวงจรในแต่ละส่วน ดังรูปที่ 3.4 จะต่ออยู่กับวงจรแปลงสัญญาณ RS-485 เพื่อที่จะแปลงสัญญาณจาก RS-485 เป็นสัญญาณ ทีทีแอล โดยจะรับข้อมูลเข้ามาทางด้านขารับ ค่าที่รับเข้ามาจะถูกนำมาตรวจสอบ และเปรียบเทียบกับค่าที่ตั้งไว้ หรือค่าที่กำหนดให้กับโหลดตัวนั้นๆ ถ้าค่าที่ส่งมาตรงกับค่าของโหลดตัวใด ไอซีไมโครคอนโทรลเลอร์ก็จะทำการกลับสถานะบิตแล้วนำสถานะนั้นส่งไปยังไอซี ULN2803 ซึ่งเป็นไอซีไดรเวอร์ ภายในจะบรรจุอินเวอร์เตอร์เกตแบบคอลเลกเตอร์เปิด จำนวน 7 ตัว โดยสามารถใช้กับแรงดันสูงสุดถึง 50 โวลต์และให้กระแสเอาต์พุตในแต่ละขาได้ถึง 500 มิลลิแอมป์ (นอกจากนี้ยังมีไดโอดป้องกันไว้ที่ทุกๆขาของเอาต์พุต ทำให้สามารถต่อกับโหลดที่เป็นขดลวดได้)จะทำหน้าที่ขับรีเลย์ให้ทำงานและทำให้หลอดสว่างในแต่ละช่อง



รูปที่ 3.4 วงจรไมโครคอนโทรลเลอร์ PIC 16F877

3.4 วงจรรีเลย์สวิตช์

เป็นวงจรที่ทำหน้าที่ควบคุมการเปิดและปิดของอุปกรณ์ไฟฟ้าต่างๆ โดยเมื่อได้รับแรงดัน +5 โวลต์จากไมโครคอนโทรลเลอร์เข้าที่ขา 5 ของรีเลย์ทำให้ขดลวดที่อยู่ภายในรีเลย์เกิดการเหนี่ยวนำ ทำให้สวิตช์ปิด จึงส่งผลให้ที่เทอร์มินอลมีแรงดันไฟสลับ 220 โวลต์ ซึ่งที่เทอร์มินอลนั้นจะต่ออยู่กับโหลด ซึ่งในที่นี้คือหลอดไฟและอุปกรณ์ไฟฟ้าต่างๆ



รูปที่ 3.5 วงจรรีเลย์สวิตช์

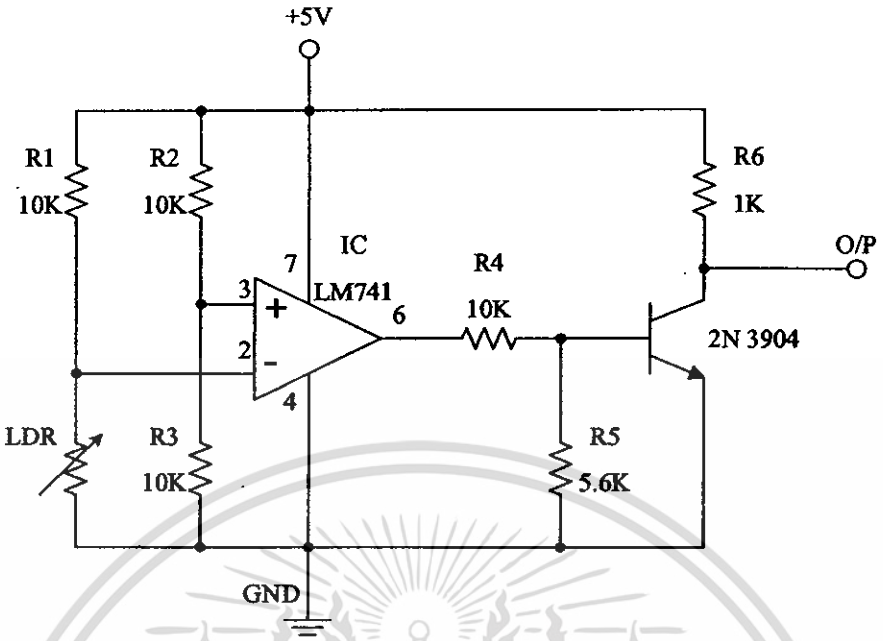
3.5 วงจรตรวจจับแสง

เนื่องจากในการที่เราใช้คอมพิวเตอร์สั่งให้หลอดไฟฟ้าเปิดหรือปิดในแต่ละครั้งนั้น เราจะไม่สามารถทราบได้เลยว่า หลอดไฟฟ้านั้นทำงานตามที่เราสั่งการหรือไม่ ดังนั้นเราจึงได้ทำการออกแบบวงจรตรวจจับแสงขึ้นมา เพื่อที่เราจะได้ทราบถึงสถานะ การทำงานของหลอดไฟฟ้านั้น หรือแม้กระทั้งใช้ตรวจสอบความสว่างภายในห้องว่าสมควรที่จะเปิดหรือปิดไฟ

ในวงจรนี้เราใช้ตัวตรวจจับทางแสงที่เรียกว่า Light Dependent Resistance (LDR) กล่าวคือ ความต้านทานของมันจะเปลี่ยนแปลงไปตามความเข้มของแสง คือถ้าความเข้มของแสงมาก ความต้านทานของตัวมันจะต่ำ และในทางกลับกันถ้าความเข้มของแสงมีค่าน้อย ความต้านทานก็จะสูง เมื่อทำการทดลองวัดค่าความต้านทานของ LDR ในช่วงเวลาตามตารางที่ 3.1

ช่วงเวลาในการวัดค่าความต้านทาน	ค่าความต้านทาน
กลางวัน	20kΩ
กลางคืน	550kΩ

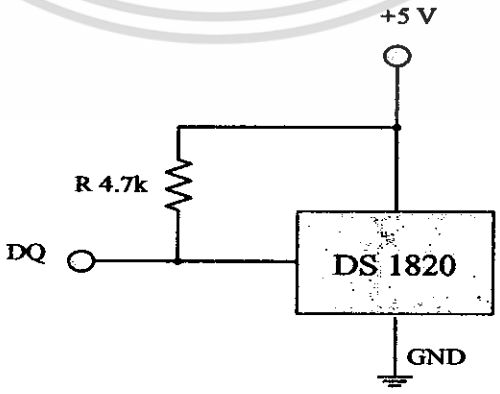
ตารางที่ 3.1 ค่าความต้านทานของ LDR



รูปที่ 3.6 วงจรตรวจจับแสง

รูปที่ 3.6 เป็นวงจรตรวจจับแสงโดยใช้ ไอซี LM741 ต่อเป็นวงจรเปรียบเทียบแรงดัน (Voltage Comparater) การทำงานของวงจรคือในสภาวะที่มีแสงน้อยหรือในช่วงเวลากลางคืนนั้น ความต้านทานของ LDR จะมีค่ามากทำให้ขา 2(inverter) ของไอซี LM741 มีค่ามากกว่าศักดาที่ขา 3 (noninverter) ทำให้ศักดาที่ขา 6 มีลอจิกเป็น 0 ทำให้ทรานซิสเตอร์ไม่ทำงาน จึงได้เอาต์พุทเท่ากับ 5 โวลต์ และในทางกลับกันเมื่อมีแสงสว่างมากหรือในช่วงเวลากลางวัน ค่าความต้านทานของ LDR จะมีค่าลดลงจนถึงระดับที่ทำให้ศักดาที่ขา 2 น้อยกว่า ขา 3 ทำให้ศักดาที่ขา 6 มีลอจิกเป็น 1 ดังนั้นทรานซิสเตอร์ก็จะทำงาน ทำให้ได้เอาต์พุต 0 โวลต์

3.6 วงจรตรวจวัดอุณหภูมิ



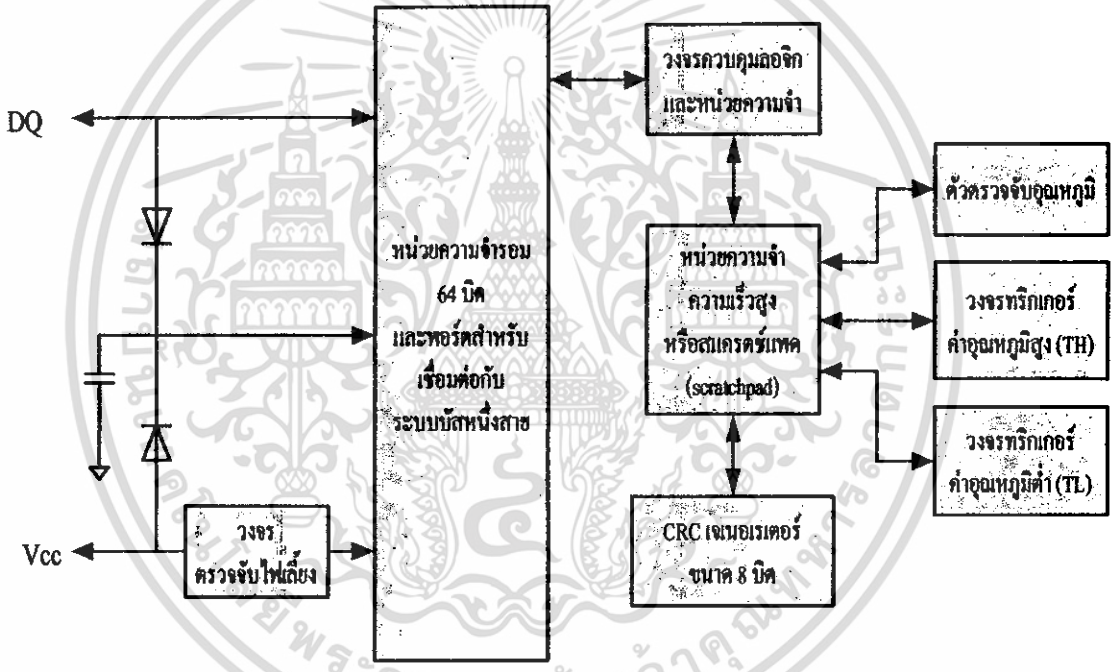
รูปที่ 3.7 วงจรตรวจวัดอุณหภูมิ

วงจรตรวจวัดอุณหภูมินี้ใช้ไอซี DS1820 ซึ่งเป็นไอซีวัดอุณหภูมิแบบดิจิตอลที่ใช้การติดต่อแบบระบบบัส 1 สายมีโครงสร้างและไดอะแกรมการทำงานดังแสดงในรูปที่ 3.8 ซึ่งในการติดต่อกับไอซี DS 1820 มีคำสั่งอยู่ 3 คำสั่งที่ต้องใช้งานคือ

1. คำสั่งสคิปรอม(Skip ROM) คือคำสั่งที่ไม่ต้องการติดต่อกับหน่วยความจำรอม เมื่อมีการต่อกับไอซี DS 1820 เพียงตัวเดียว จึงไม่จำเป็นต้องกำหนดแอดเดรสให้กับไอซี โดยการส่งค่า 0xCC ให้กับบัส

2. คำสั่งแปลงอุณหภูมิ(Convert T) โดยต้องรอการแปลงอุณหภูมิ อย่างน้อย 200 มิลลิวินาที เพื่อนำค่าที่แปลงมาเก็บไว้ในสแครตช์แพดก่อนที่จะอ่านค่าอุณหภูมิมาใช้งานได้โดยการส่งค่า 0x44 ให้กับบัส

3. คำสั่งอ่านข้อมูลจากสแครตช์แพด(Read Scratchpad) เมื่อส่งคำสั่งอ่านค่าแล้ว DS 1820 จะส่งข้อมูลค่าอุณหภูมิกลับมาให้ทั้งหมด 9 ไบต์ โดยการส่งค่า 0xBE ให้กับบัส



รูปที่ 3.8 โครงสร้างและไดอะแกรมการทำงานของไอซี DS 1820

	ไบต์
ข้อมูลอุณหภูมิไบต์ต่ำ (TL)	0
ข้อมูลอุณหภูมิไบต์สูง	1
ข้อมูลอุณหภูมิค่าสูง	2
ข้อมูลอุณหภูมิค่าต่ำ (TL)	3
สำรองไว้	4
สำรองไว้	5
รีจีสเตอร์เก็บค่าการนับ	6
รีจีสเตอร์เก็บค่าการนับต่อ °C	7
CRC	8

รูปที่ 3.9 การจัดสรรพื้นที่ของสแครตช์แพดใน DS1820

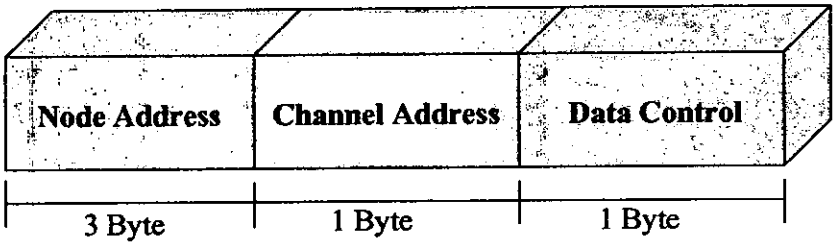
หัวใจสำคัญอยู่ที่ตัวตรวจจับอุณหภูมิและหน่วยความจำความเร็วสูง ที่เรียกว่าสแครตช์แพด (scratchpad) ซึ่งมีขนาด 9 ไบต์ เมื่อ DS1820 วัดอุณหภูมิได้จะนำค่าที่วัดได้นี้เก็บไว้ในสแครตช์แพดที่ไบต์ 0 และ 1 ทั้งนี้เนื่องจากไอซี DS1820 สามารถให้ข้อมูลอุณหภูมิได้ละเอียดถึง 16 บิต เมื่อนำมาแปลงเป็นข้อมูลเลขฐานสิบจึงสามารถแสดงความละเอียดของอุณหภูมิได้ถึง 0.5 องศาเซลเซียสและ 0.9 องศาฟาเรนไฮต์ โดยมีย่านวัดอุณหภูมิตั้งแต่ -55 ถึง +125 องศาเซลเซียส หรือ -67 ถึง +257 องศาฟาเรนไฮต์ โดยค่าขององศาฟาเรนไฮต์ต้องใช้การแปลงหน่วยช่วย DS1820 ใช้เวลาแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลประมาณ 200 มิลลิวินาที สามารถกำหนดขอบเขตของอุณหภูมิที่วัดได้และให้แจ้งเตือนเมื่อค่าของอุณหภูมิสูงขึ้นหรือลดต่ำลงถึงค่าที่กำหนด โดยค่าอุณหภูมิที่กำหนดนี้จะเก็บไว้ที่สแครตช์แพดในไบต์ที่ 2 และ 3

<u>การคำนวณและการสร้างส่วนของโปรแกรมควบคุมการทำงาน</u>

3.7 การออกแบบส่วนของโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์

3.7.1 รูปแบบการส่งข้อมูล

ในการติดต่อสื่อสารกันระหว่าง คอมพิวเตอร์กับไมโครคอนโทรลเลอร์ ในระบบนี้ได้มีการกำหนดรูปแบบของการส่งข้อมูลไว้ดังนี้

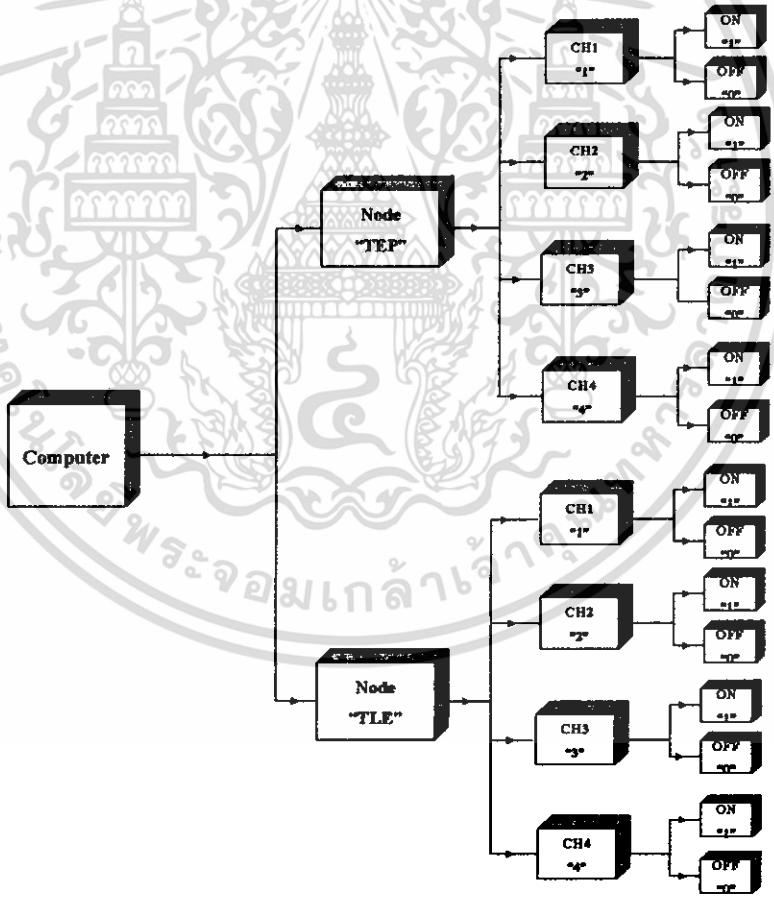


รูปที่ 3.10 รูปแบบเฟรมข้อมูล

Node address : ใน 3 ไบต์แรกนี้จะถูกส่งไปเพื่อกำหนดโหนดว่าต้องการที่จะติดต่อกับโหนดใด หรือกล่าวได้ว่าเป็นการเลือกชั้นว่าต้องการที่จะควบคุมอุปกรณ์ไฟฟ้าในชั้นใดนั่นเอง

Channel address : ในไบต์นี้จะถูกส่งไปเพื่อทำการเลือกพอร์ตที่ต้องการควบคุมหรือกล่าวคือเป็นการกำหนดว่าจะควบคุมอุปกรณ์ไฟฟ้าในส่วนใด

Data control : ในไบต์แรกนี้จะถูกส่งไปเพื่อกำหนดสถานะการทำงานว่าต้องการจะเปิดหรือปิด



รูปที่ 3.11 รูปแบบที่ใช้ในการทดลองส่งข้อมูล

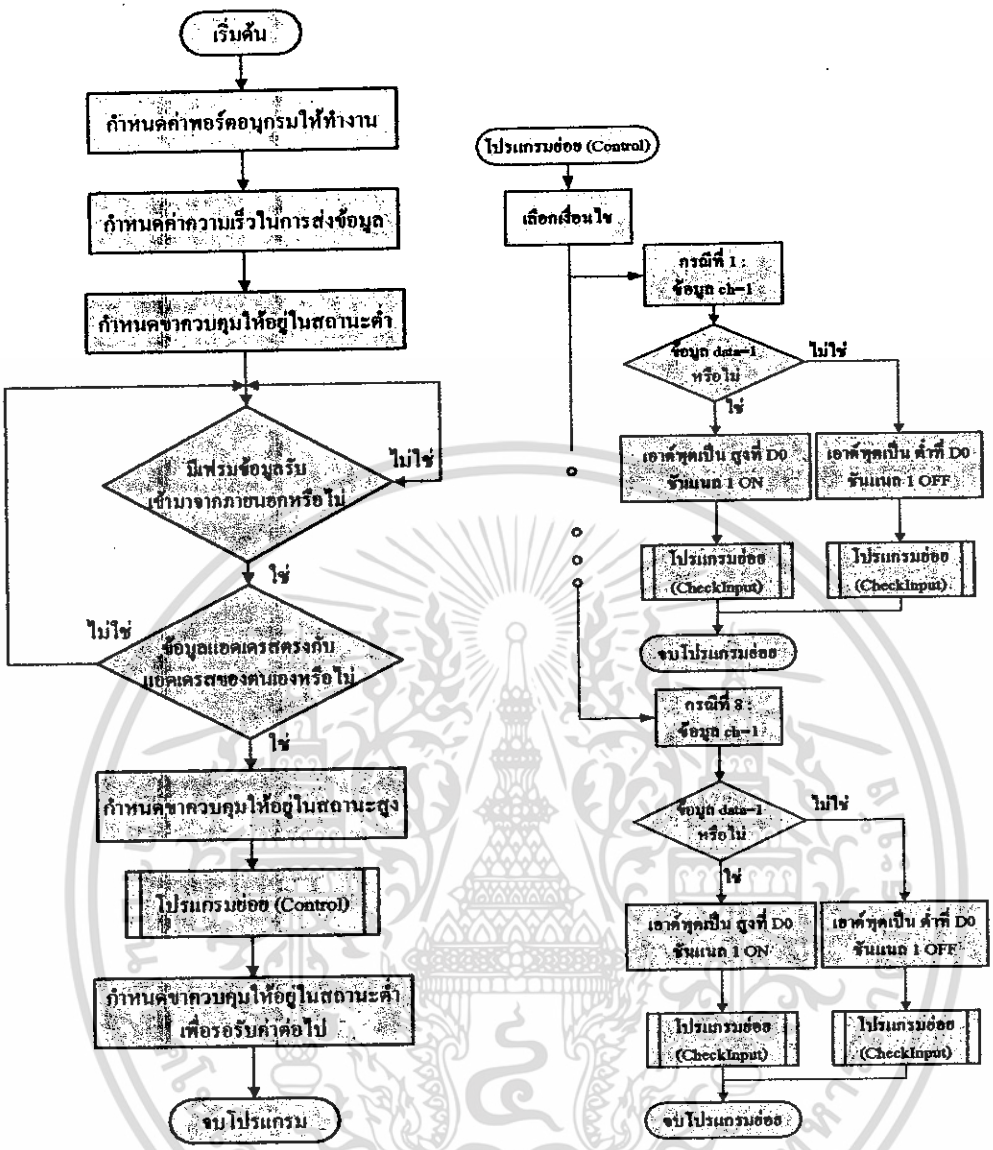
ในการติดต่อสื่อสารกันระหว่าง คอมพิวเตอร์กับไมโครคอนโทรลเลอร์ จะมีการกำหนดค่าที่ใช้ในการควบคุมอุปกรณ์ต่างๆไว้แน่นอนเพื่อที่ให้ง่ายสำหรับการเขียนซอฟแวร์ที่ใช้ในการควบคุมอุปกรณ์ต่างๆ โดยจะมีค่าดังตัวอย่างในตารางที่ 3.2

ข้อมูลที่รับเข้ามา	การทำงานของข้อมูลที่รับเข้ามา		
	โหนด	แชนแนล	สถานะ
TEP11	TEP	1	เปิด
TEP 10	TEP	1	ปิด
TEP 21	TEP	2	เปิด
TEP 20	TEP	2	ปิด
TEP 31	TEP	3	เปิด
TEP 30	TEP	3	ปิด
TEP 41	TEP	4	เปิด
TEP 40	TEP	4	ปิด
TLE11	TLE	1	เปิด
TLE 10	TLE	1	ปิด
TLE 21	TLE	2	เปิด
TLE 20	TLE	2	ปิด
TLE 31	TLE	3	เปิด
TLE 30	TLE	3	ปิด
TLE 41	TLE	4	เปิด
TLE 40	TLE	4	ปิด

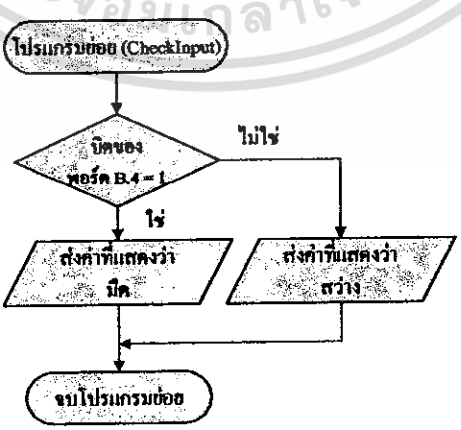
ตารางที่ 3.2 ตัวอย่างการกำหนดค่าของข้อมูลที่ไมโครคอนโทรลเลอร์ใช้ในการควบคุม

3.7.2 การออกแบบโปรแกรมไมโครคอนโทรลเลอร์

ในการกำหนดเงื่อนไขต่างๆให้กับไมโครคอนโทรลเลอร์นั้น จะอ้างอิงจากตารางที่ 3.2 ซึ่งจะกำหนดข้อมูลที่ใช้ในการควบคุมอุปกรณ์ต่างๆไว้ โดยไมโครคอนโทรลเลอร์จะมีขั้นตอนในการเปรียบเทียบค่าต่างๆดังแสดงในโฟล์วชาร์ตในรูปที่ 3.12



รูปที่ 3.12 ไฟล์วชาร์ตการทำงานของไมโครคอนโทรลเลอร์

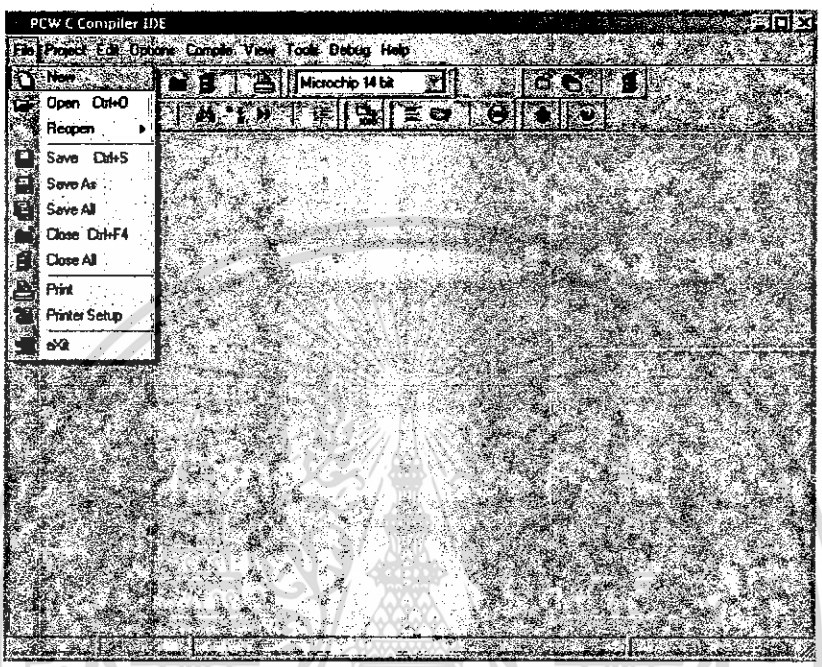


รูปที่ 3.13 ไฟล์วชาร์ตการทำงานของฟังก์ชั่นย่อยในการตรวจจับแสง

3.7.3 ขั้นตอนการคอมไพล์

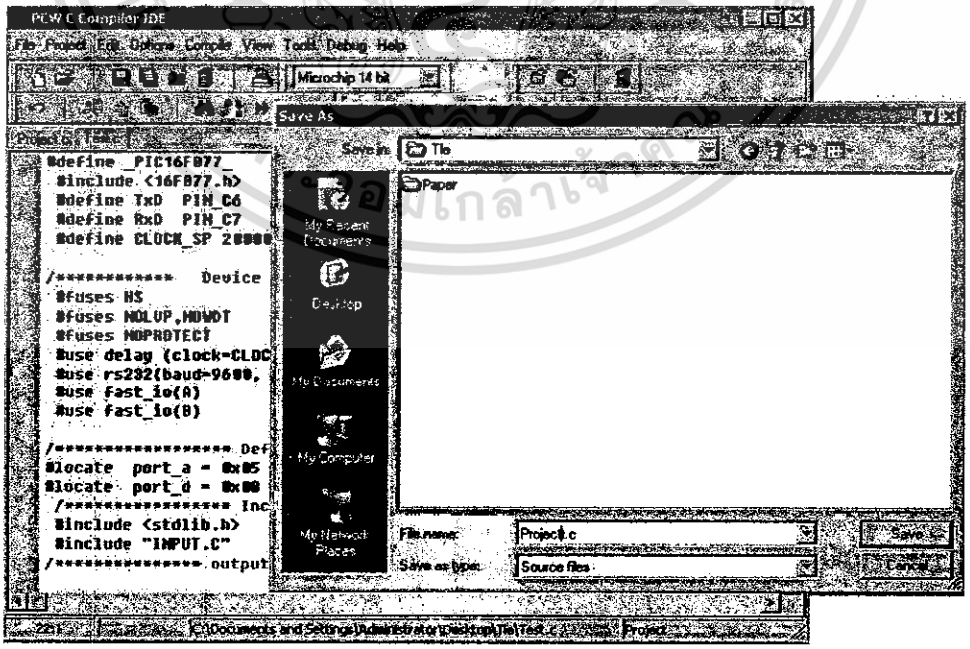
ขั้นตอนการสร้างส่วนนี้จะใช้โปรแกรมภาษา C ในการเขียนคำสั่งเพื่อควบคุมการทำงานของไมโครคอนโทรลเลอร์ โดยใช้โปรแกรม CCS C Compiler เพื่อทำการคอมไพล์ โดยมีขั้นตอนการคอมไพล์ดังนี้

1) เลือกที่ file new เพื่อทำการสร้าง file ขึ้นมาใหม่ไว้สำหรับเขียนโปรแกรม



รูปที่ 3.14 การสร้างไฟล์ใหม่

2) เขียนโปรแกรมลงในหน้าต่าง จากนั้นทำการบันทึกโปรแกรมไว้



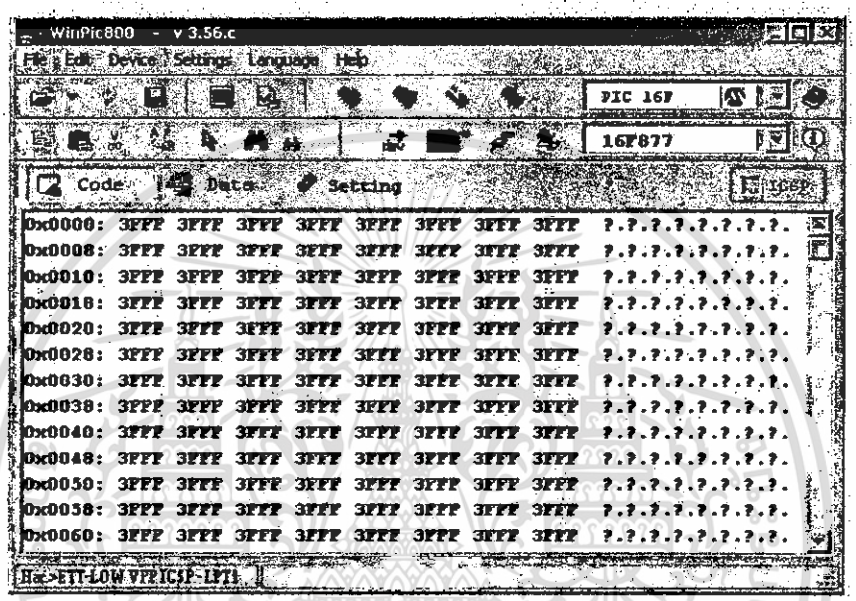
รูปที่ 3.15 การเขียนและบันทึกโปรแกรม

3) คลิกไปที่โปรแกรมเมนู Compile (หรือกด F9) เพื่อทำการคอมไพล์โปรแกรมเมื่อคอมไพล์เสร็จแล้วจะได้ไฟล์ใหม่เป็นไฟล์ .HEX

3.7.4 ขั้นตอนการบันทึกโปรแกรมลงใน ไมโครคอนโทรลเลอร์

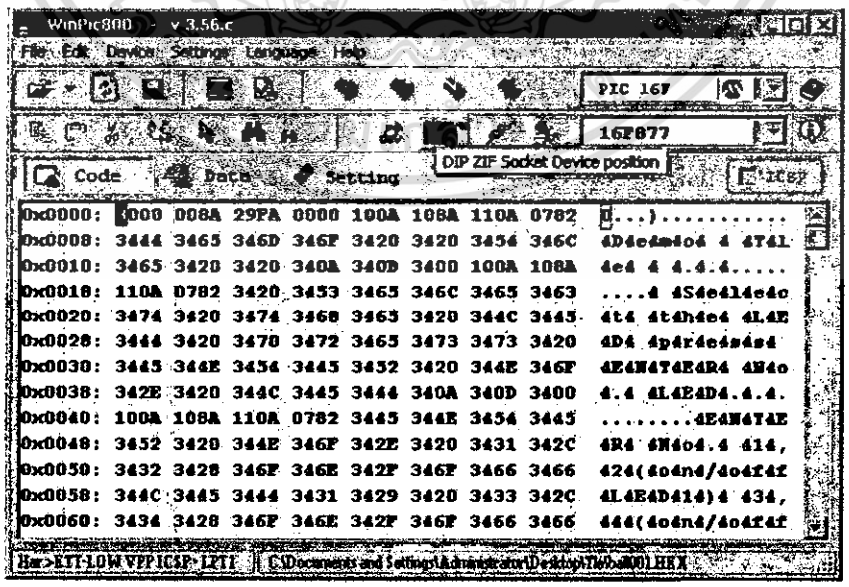
ขั้นตอนต่อไปจะเป็นการบันทึกโปรแกรมที่ทำการคอมไพล์แล้ว ลงไปใน ไมโครคอนโทรลเลอร์ โดยในขั้นตอนนี้ จะต้องมีอุปกรณ์ที่ใช้ในการบันทึกโดยเลือกใช้ บอร์ด CP-PIC877 V1.0 R1 และใช้โปรแกรม WINPIC 800 เป็นโปรแกรมที่ช่วยในการบันทึก โดยมีขั้นตอนในการบันทึกดังนี้

1) เมื่อดับเบิ้ลคลิกไปที่ไอคอนของ โปรแกรม winpic800 จะปรากฏหน้าต่างดังรูป



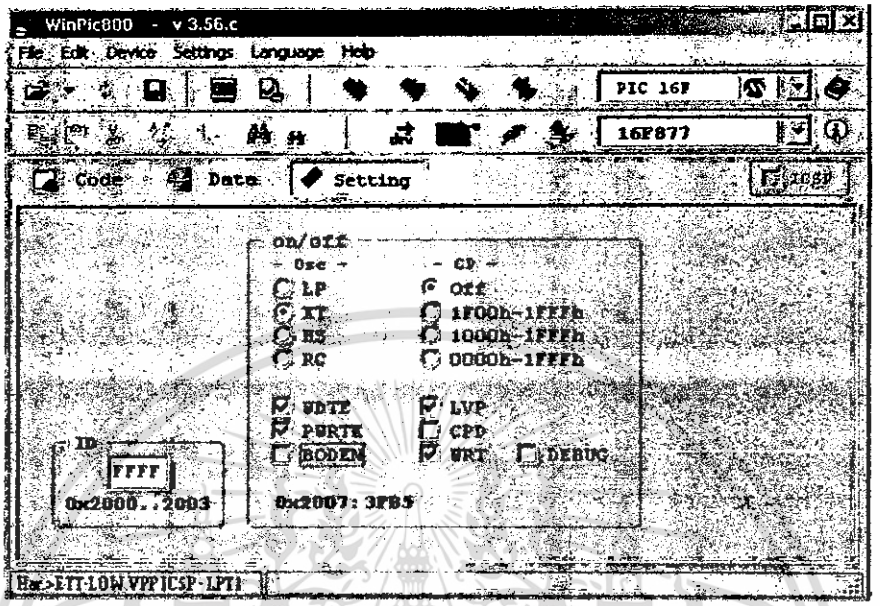
รูปที่ 3.16 โปรแกรมที่ใช้ในการบันทึก

2) ทำการเปิดไฟล์ที่ได้ทำการคอมไพล์ไว้ขึ้นมา



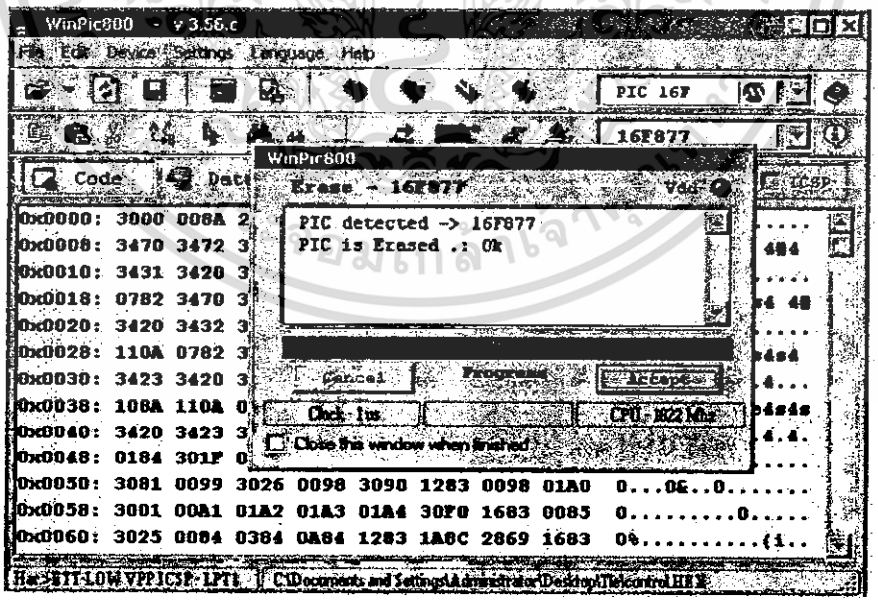
รูปที่ 3.17 การเปิดไฟล์ที่ได้ทำการคอมไพล์ไว้ขึ้นมา

3) คลิกไปที่ setting แล้วทำการกำหนดค่าต่างๆดังรูปที่ 3.18



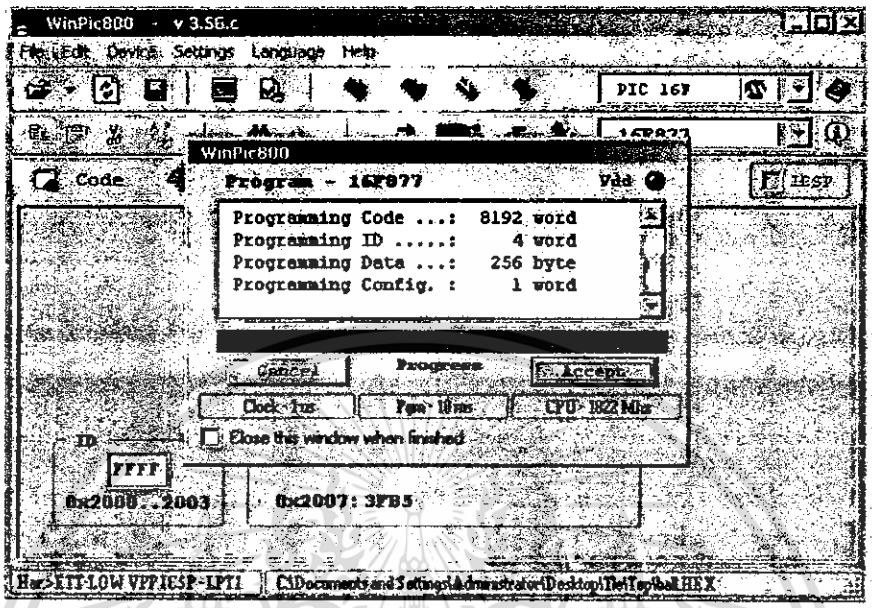
รูปที่ 3.18 การตั้งค่าต่างๆในโปรแกรม

4) คลิกไปที่ Erese เพื่อทำการลบข้อมูลที่มีอยู่ในตัวไมโครคอนโทรลเลอร์ก่อน



รูปที่ 3.19 แสดงการลบข้อมูลที่มีอยู่ในตัวไมโครคอนโทรลเลอร์

5) คลิกไปที่ Program All เพื่อทำการบันทึกข้อมูลลงไปในตัวไมโครคอนโทรลเลอร์



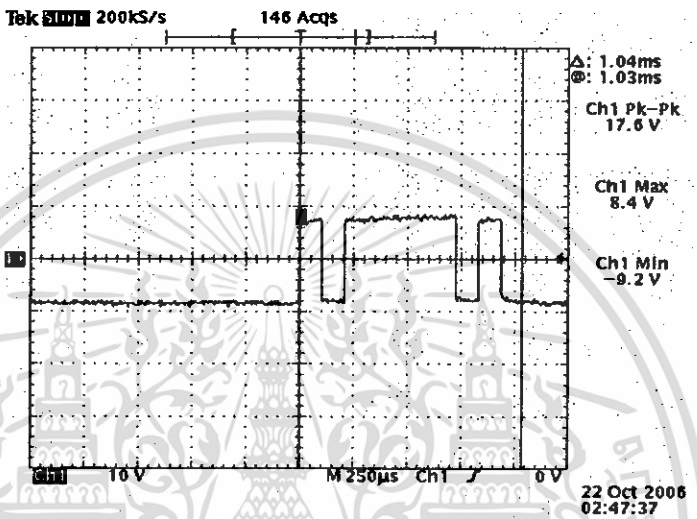
รูปที่ 3.20 การบันทึกข้อมูลลงไปในตัวไมโครคอนโทรลเลอร์

บทที่ 4

การทดลองและผลการทดลอง

4.1 ผลการทดลองเมื่อทำการวัดสัญญาณ RS-232

ขณะทำการส่งข้อมูลสัญลักษณ์ "A" ผ่าน Hyper Terminal และทำการวัดสัญญาณที่ขา 3 (Tx) ของพอร์ตอนุกรม



รูปที่ 4.1 สัญญาณข้อมูลที่ออกจากขา 3 (Tx) ของพอร์ตอนุกรม

จากรูปที่ 4.1 เป็นสัญญาณที่ได้จากการส่งค่า "A" ซึ่งรหัส ASCII ของค่า A = 41H หรือ 0100 0001 B โดยสัญญาณทั้งหมดประกอบด้วย start bit (ลอจิก 0) และ stop bit (ลอจิก 1) อีก 2 บิต รวมเป็น 10 บิต ซึ่งเป็นการส่ง Bit LSB หรือบิตทางขวามือไปก่อน ดังนั้นสัญญาณที่ส่งออกมาจึงเป็นสัญญาณ

ลอจิก 0 1000 0010 1 ตามลำดับ

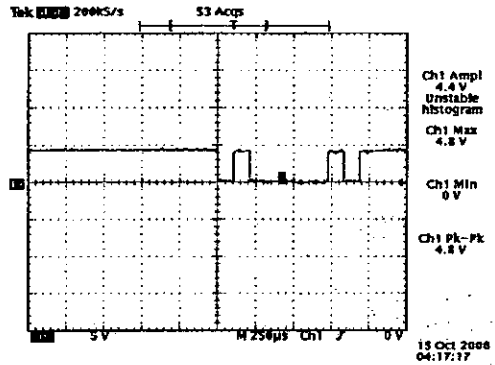
ซึ่งตามมาตรฐาน RS-232 ค่าลอจิก "1" อยู่ในช่วง -3 ถึง -15 โวลต์ และค่าลอจิก "0 " อยู่ในช่วง +3 ถึง +15 โวลต์ ซึ่งในรูปลอจิก "1" มีค่า -9.2 โวลต์ และลอจิก "0" มีค่า +8.4 โวลต์ ซึ่งเป็นไปตามทฤษฎี และจากอัตราเร็วในการส่งข้อมูล (Bit Rate) มีค่า 9600 บิตต่อวินาที ดังนั้น 1 บิตจะมีช่วงคาบเวลาเท่ากับ

$$1/9600 = 1.0416 \times 10^{-4} \text{ วินาที}$$

ดังนั้น สัญญาณที่ขาเอาต์พุตเท่ากับ 10 บิตต่อ 1 สัญลักษณ์ จึงมีช่วงเท่ากับ 1.0416 มิลลิวินาที ซึ่งจากรูปที่ 4.1 มีช่วงคาบเท่ากับ 1.04 มิลิวินาที ซึ่งเป็นไปตามทฤษฎี

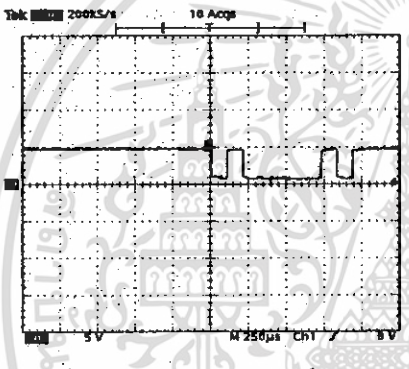
4.2 ผลการทดลองวัดค่าสัญญาณของวงจรอินเตอร์เฟส RS-485 ทางด้านส่ง

- เมื่อทำการวัดสัญญาณอินพุตที่ขา 3 (DE) ของไอซี DS75176 ซึ่งเป็นไอซีที่ใช้แปลงสัญญาณ RS-485 จะได้สัญญาณดังรูปที่ 4.2

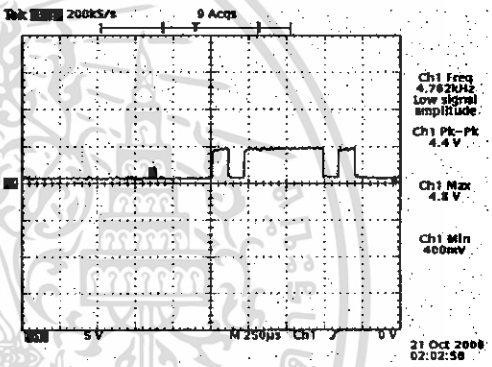


รูปที่ 4.2 สัญญาณอินพุตที่ขา 3 ของไอซี DS75176

- เมื่อทำการวัดสัญญาณที่ขา Tx+ และ Tx- เทียบกับกราวด์ ของวงจรอินเตอร์เฟส RS-485 จะได้ผลดังแสดงในรูปที่ 4.3 และ 4.4

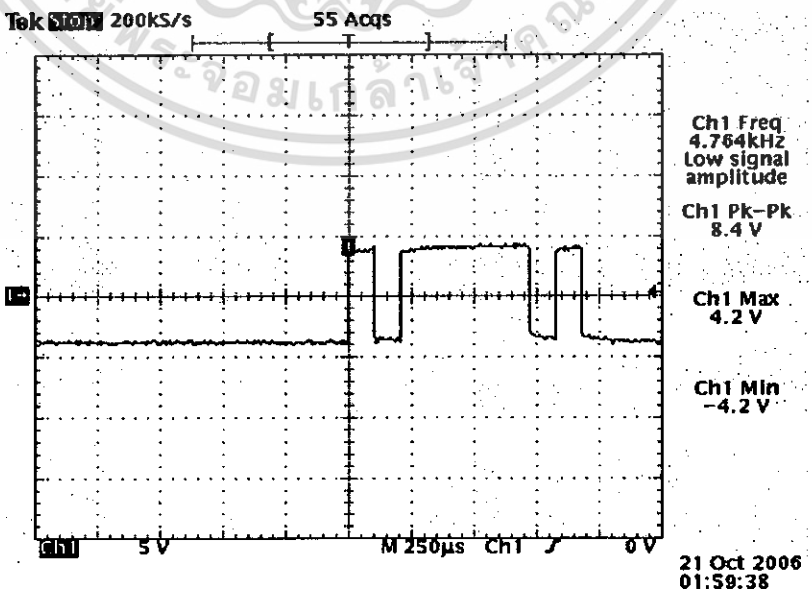


รูปที่ 4.3 สัญญาณเอาต์พุตที่ขา Tx-



รูปที่ 4.4 สัญญาณเอาต์พุตที่ขา Tx+

- เมื่อทำการวัดสัญญาณเอาต์พุตที่ขา T x- เทียบกับขา Tx+ ของวงจรอินเตอร์เฟส RS-485

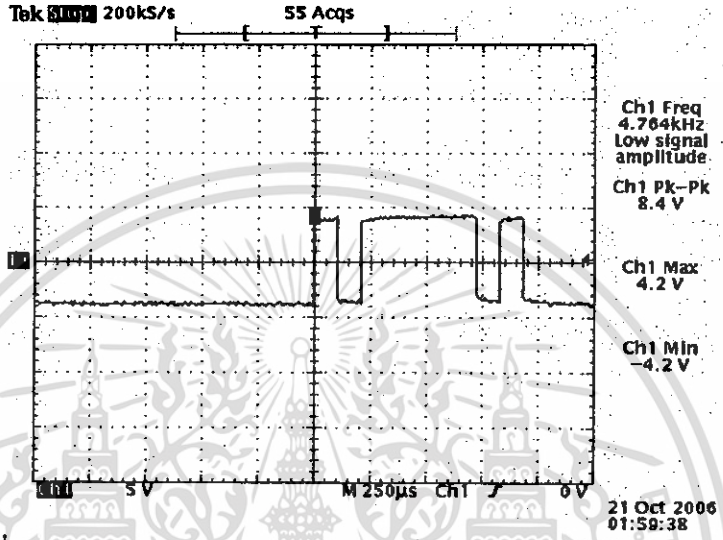


รูปที่ 4.5 สัญญาณข้อมูลสัญลักษณ์ "A" ที่ออกจากเอาต์พุตของวงจรอินเตอร์เฟส RS-485

จากทฤษฎีแล้วระดับแรงดันของตัวส่งจะส่งแรงดันช่วง+2โวลต์ถึง+6โวลต์และ-2โวลต์ถึง-6โวลต์ ทางด้านเอาต์พุตระหว่าง Tx- และ Tx+ ซึ่งจากรูปจะเห็นได้ว่า ระดับแรงดันที่ส่งออกไปอยู่ที่ +4.2โวลต์ และ -4.2โวลต์ ซึ่งอยู่ในช่วงตามที่ทฤษฎีได้กำหนดไว้

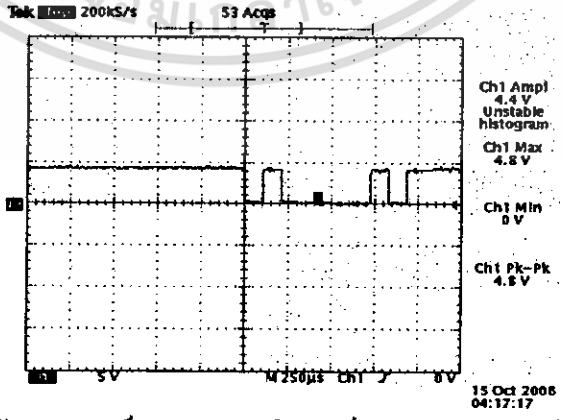
4.3 ผลการทดลองวัดค่าของวงจรอินเตอร์เฟส RS-485 ทางด้านรับ

- เมื่อทำการวัดสัญญาณอินพุตที่วงจรอินเตอร์เฟส RS-485 ทางด้านรับ



รูปที่ 4.6 สัญญาณอินพุตของวงจรอินเตอร์เฟส RS-485 ทางด้านรับ

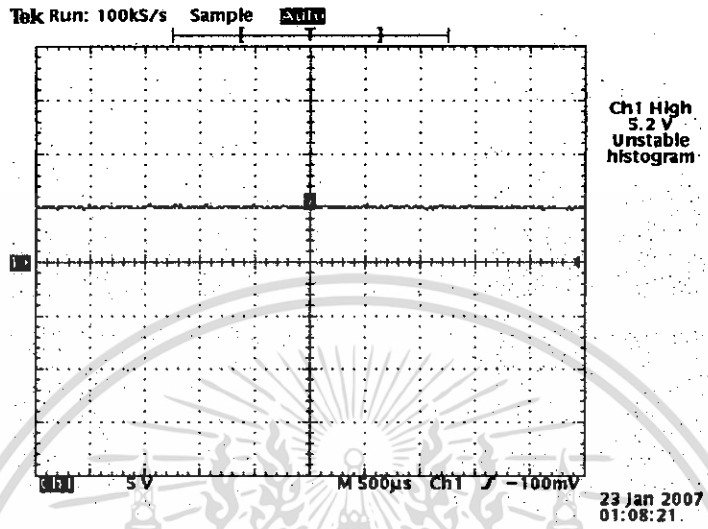
จากทฤษฎีแล้วตัวรับในระบบสมดุลจะรับสถานะของสัญญาณเพื่อวัดความแตกต่างของระดับแรงดัน โดยถ้าแรงดันมีค่าสูงกว่า +200 มิลลิโวลต์ จะถูกกำหนดสถานะลอจิก "0" และถ้ามีขนาดน้อยกว่า -200 มิลลิโวลต์ จะถูกกำหนดสถานะลอจิก "1" ซึ่งจากรูปมีค่าระดับแรงดันที่เป็นไปตามทฤษฎี จึงทำให้ได้สัญญาณเอาต์พุตดังรูปที่ 4.7 โดยสัญญาณที่ได้นี้จะเป็นสัญญาณทีทีแอลที่ใช้ในการควบคุมไมโครคอนโทรลเลอร์



รูปที่ 4.7 สัญญาณเอาต์พุตของวงจรอินเตอร์เฟส RS-485 ทางด้านรับ

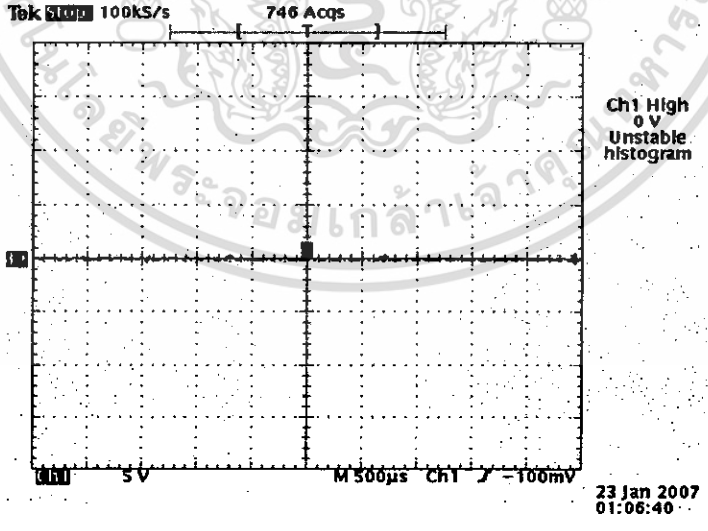
4.4 ผลการทดลองวัดสัญญาณของวงจรตรวจจับแสง

- เมื่อทำการวัดสัญญาณทางด้านเอาต์พุตของวงจรตรวจจับแสงในขณะที่ไม่มีแสงสว่างได้เอาต์พุตเท่ากับ 5 โวลต์ ดังแสดงในรูปที่ 4.8



รูปที่ 4.8 สัญญาณทางด้านเอาต์พุตของวงจรตรวจจับแสงในขณะที่ไม่มีแสงสว่าง

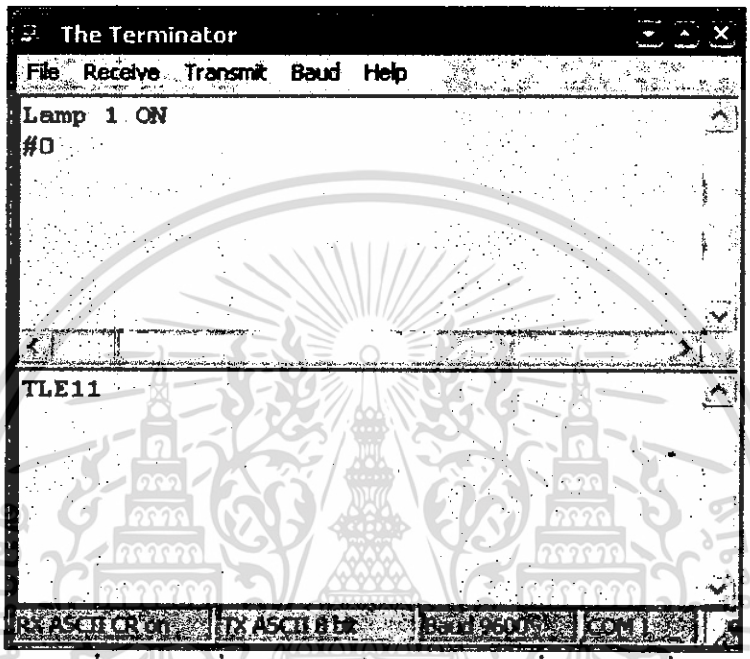
- เมื่อทำการวัดสัญญาณทางด้านเอาต์พุตของวงจรตรวจจับแสงในขณะที่มีแสงสว่าง ได้เอาต์พุตเท่ากับ 0 โวลต์ ดังแสดงในรูปที่ 4.9



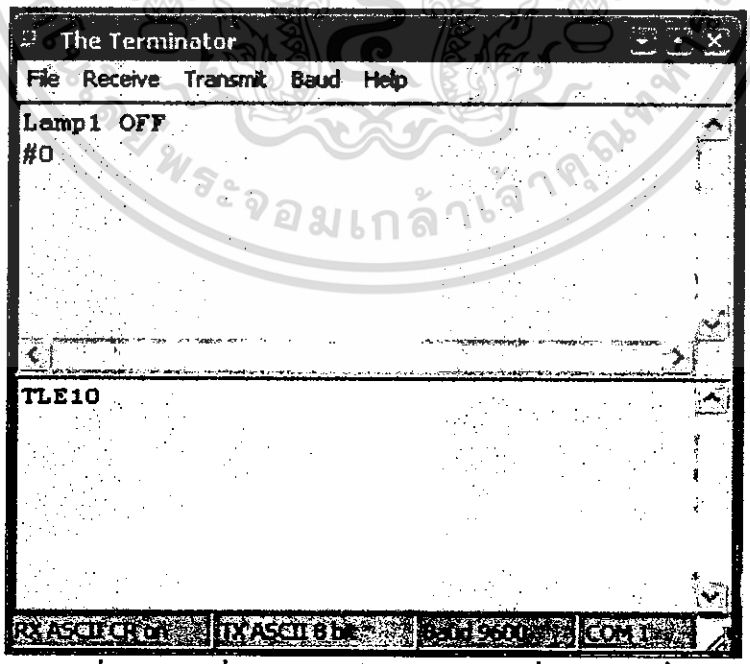
รูปที่ 4.9 สัญญาณทางด้านเอาต์พุตของวงจรตรวจจับแสงในขณะที่มีแสงสว่าง

4.5 ผลการทดลองควบคุมอุปกรณ์ไฟฟ้าผ่านโปรแกรม MiniTerminal

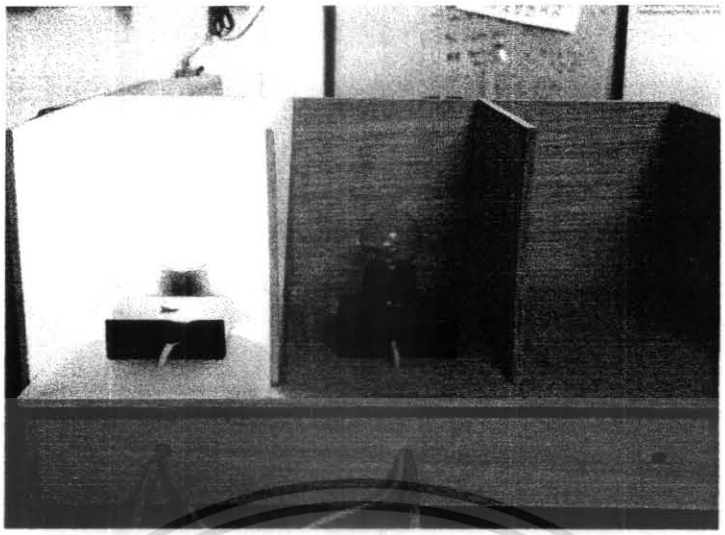
เมื่อเริ่มทำการส่งค่าที่ใช้ในการควบคุมผ่านทางโปรแกรม MiniTerminal ข้อมูลจะถูกส่งไปควบคุมไมโครคอนโทรลเลอร์ผ่านทางพอร์ตอนุกรม และเมื่อไมโครคอนโทรลเลอร์ทำการประมวลผลตามคำสั่งที่ได้รับแล้วจะส่งข้อความเพื่อแสดงสถานะการทำงานของไมโครคอนโทรลเลอร์กลับมายังคอมพิวเตอร์และแสดงผ่านโปรแกรม MiniTerminal ดังรูปที่4.10 และ รูปที่4.11



รูปที่ 4.10 การสั่งเปิดอุปกรณ์ไฟฟ้าในโหนดที่ 1 ชันแนลที่ 1

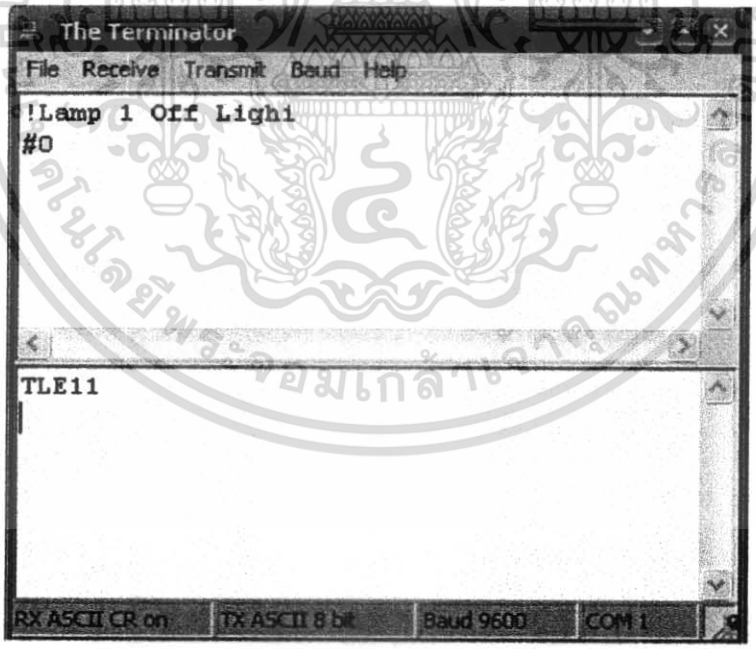


รูปที่ 4.11 การสั่งปิดอุปกรณ์ไฟฟ้าในโหนดที่ 1 ชันแนลที่ 1

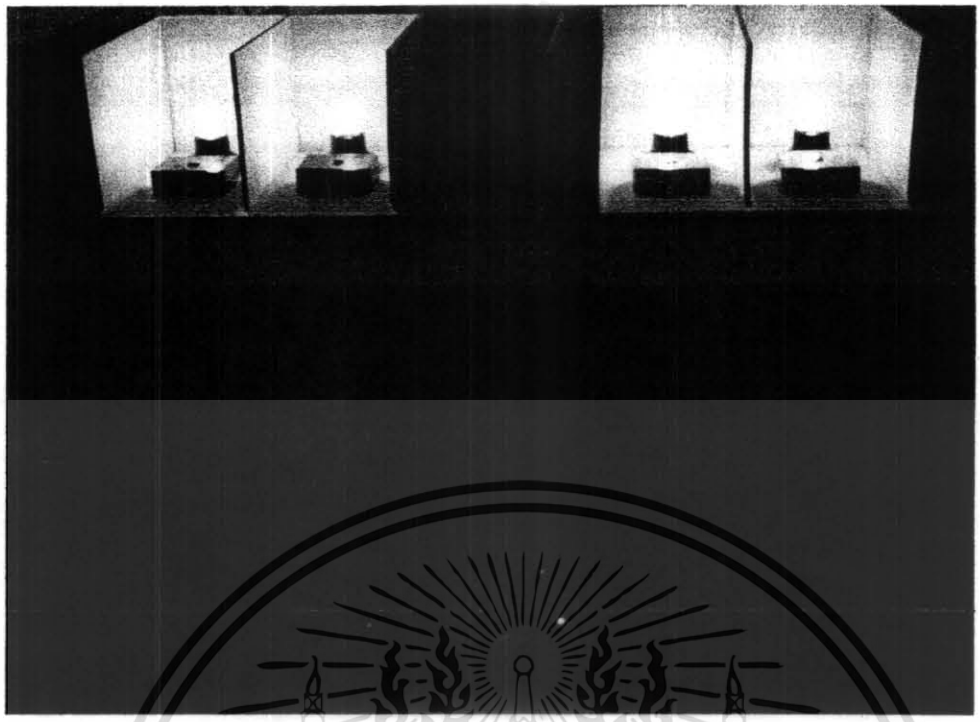


รูปที่ 4.12 การทำงานของหลอดไฟหลังจากสั่งเปิดผ่านทางโปรแกรม MiniTerminal

ในกรณีเมื่อเราสั่งเปิดหลอดไฟฟ้าแล้วปรากฏว่าหลอดไฟฟ้านั้นเกิดขาดหรือใช้งานไม่ได้ ไมโครคอนโทรลเลอร์จะส่งข้อความแจ้งเตือนให้ทราบดังแสดงในรูปที่ 4.13 เพื่อที่จะได้ทำการซ่อมแซมต่อไป



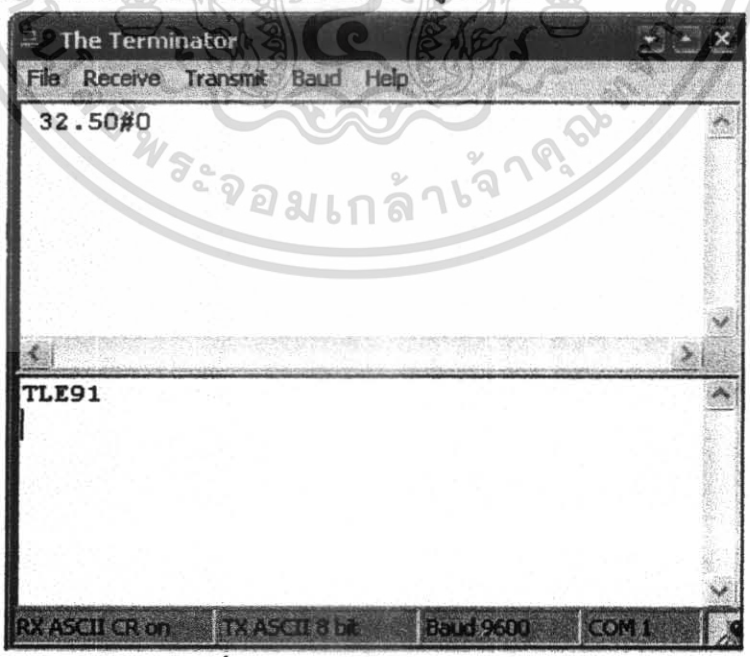
รูปที่ 4.13 การแจ้งเตือนเมื่อหลอดไฟฟ้าใช้การไม่ได้



รูปที่ 4.14 การทำงานของหลอดไฟเมื่อสั่งเปิดอุปกรณ์ไฟฟ้าทั้งหมด

4.6 ผลการทดลองวัดค่าอุณหภูมิผ่านโปรแกรม MiniTerminal

เมื่อต้องการเรียกดูค่าอุณหภูมิ จะต้องส่งคำสั่งเพื่อไปขอเรียกดูค่าจากไมโครคอนโทรลเลอร์ เมื่อไมโครคอนโทรลเลอร์ได้รับแล้วจะทำการเรียกดูค่าอุณหภูมิจากวงจรตรวจวัดอุณหภูมิแล้วส่งกลับมาแสดงผลยังหน้าจอผ่านโปรแกรม MiniTerminal ดังแสดงในรูปที่4.15



รูปที่ 4.15 การเรียกดูค่าอุณหภูมิ

บทที่ 5
บทวิจารณ์และบทสรุป

5.1 สรุปผลการทดลอง

จากที่ได้ทำการออกแบบและทดลองใช้งานพบว่าสามารถควบคุมอุปกรณ์ไฟฟ้าผ่านทางคอมพิวเตอร์ได้ สามารถควบคุมการเปิดและปิด อุปกรณ์ไฟฟ้าได้ และยังตรวจวัดความสว่างของหลอดไฟและค่าอุณหภูมิได้

5.2 ปัญหาที่เกิดขึ้นและแนวทางการแก้ไข

- ปัญหาที่เกิดขึ้น
  1. ในการรับส่งข้อมูลทุกครั้งจะต้องมีการกำหนดสถานะให้กับขาที่ใช้ควบคุมการรับ ส่งข้อมูลของไอซี 75176 ซึ่งเมื่อทำการกำหนดให้อยู่ในสถานะที่พร้อมส่ง แล้วทำการส่งข้อมูลออกไป ปรากฏว่าข้อมูลทางด้านรับเกิดการผิดพลาดไปอยู่บ่อยครั้ง
  2. ในการเลือกดูค่าอุณหภูมิทุกครั้งจะต้องมีการส่งคำสั่งไปเลือกดูทุกครั้ง ทำให้ค่าอุณหภูมิที่แสดงบนหน้าจอคอมพิวเตอร์ไม่ใช่ค่าอุณหภูมิในเวลาปัจจุบันเสมอ
  3. เมื่อมีการใช้งานที่ถี่มากเช่น มีการสั่งเปิด ปิดอุปกรณ์ไฟฟ้าในแต่ละโหนดบ่อยๆและเร็วๆ อาจทำให้วงจรที่ใช้ควบคุมอุปกรณ์ไฟฟ้าเกิดการแฮงค์ขึ้นได้
- แนวทางการแก้ไข
  1. เมื่อกำหนดสถานะให้กับขาควบคุมการรับ ส่ง ของไอซี 75176 แล้ว จะต้องมีการดีเลย์ก่อนชั่วขณะหนึ่ง แล้วจึงค่อยทำการรับส่งข้อมูล
  2. อาจต้องมีการพัฒนาโปรแกรมเพิ่มเติม เพื่อให้สามารถแสดงค่าอุณหภูมิตลอดเวลาในหน้าจอคอมพิวเตอร์
  3. อาจพัฒนาโปรแกรมให้มีเสถียรภาพมากยิ่งขึ้น

5.3 ข้อเสนอแนะ

ในการนำไปใช้งานจริงอาจสร้างช่องต่อเอาต์พุตเพิ่มเติมได้อีก เพราะเนื่องจากโครงงานนี้เป็นเพียงการทดลอง จึงใช้เอาต์พุตเพียง 8 ช่องเท่านั้น และอาจนำไปประยุกต์ใช้งานร่วมกับเซนเซอร์อื่นๆได้อีก เช่น เซนเซอร์ตรวจจับควัน เป็นต้น

ภาคผนวก

โปรแกรมไมโครคอนโทรลเลอร์ในโหนดที่ 1

```
 #include <16877F.h>
 #define TxD  PIN_C6
 #define RxD  PIN_C7
 #define CLOCK_SP 4000000
/*************** Device Specification ***************/
 #fuses HS                  //Oscillator mode HS
 #fuses NOLVP,NOWDT         //No Low Voltage Program, No Watchdog timer
 #fuses NOPROTECT           //Code no protection
 #use delay (clock=CLOCK_SP)
 #use rs232(baud=9600, xmit=TxD, rcv=RxD)
 #include <stdlib.h>
 #include "INPUT.C"

 #include <touch.c>          //Module Function of Temperature

/************************ Port ********************/
 #locate  port_b = 0x06
 #locate  port_d = 0x08

/****************************** Delay ******************************/
void Delay(int16 n)
 {     do
      {n--;}
       while(n>0);
 }

/************************ Show Tempereture ************************/
void ReadTemp_DS1820(void) {
   byte i;
   byte buffer[9];
   byte temp_msb,temp_lsb,temp_half;
   if(touch_present()) {                          // get present (reset)
      touch_write_byte(0xCC);                     // Skip ROM
      touch_write_byte (0x44);                    // Start Conversion
      delay_ms(1000);                             // delay
      touch_present();                            // get present (reset)
      touch_write_byte(0xCC);                     // Skip ROM
      touch_write_byte (0xBE);                    // Read Scratch Pad
      for(i=0; i<9;i++)                           // read 9bytes
         buffer[i] = touch_read_byte();
         temp_lsb = buffer[0];
         temp_msb = buffer[1];
   }
     output_high(PIN_B2);
     Delay(1000);
     printf ("%c%3.2f C", (buffer[1])?'-':' ',(float)buffer[0]/2);
     Delay(1000);
     output_low(PIN_B/2);
     Delay(1000);
}
```

```
/************************** Control_Logic  ***********************/

void Control(char ch,data)
{
 set_tris_d(0x00);                          //SET PIN D0-D7  is  Output
switch(ch)
{
case '1':                                   // Channel 1
   {   if(data=='1')                              //Lamp_1 "ON"
         { output_high(PIN_D0);
           Delay(1000);
         if(input(PIN_B4))                  //Check Logic input
          { output_high(PIN_B2)             //Set PIN Control = "1"
            Delay(1000);
            printf("!Lamp1 off light ");
            Delay(1000);
            output_low(PIN_B2);             //Set PIN Control = "0"
            Delay(1000);
          }
          else
          { output_high(PIN_B2);
            Delay(1000);
            printf("Lamp1 ON");              // Channel 1 Status of  ON
            Delay(1000);
            output_low(PIN_B2);
            Delay(1000);
          }
                }
 else                                       //Lamp_1 "OFF"
         { output_low(PIN_D0);
           Delay(1000);
         if(input(PIN_B4))
          { output_high(PIN_B2);
            Delay(1000);
            printf("Lamp1 OFF");             // Channel 1 Status of  OFF
            Delay(1000);
            output_low(PIN_B2);
            Delay(1000);
          }
          else
         { output_high(PIN_B2);
           Delay(1000);
           printf("Lamp1 ON");               // Channel  1Status of  ON
           Delay(1000);
           output_low(PIN_B2);
           Delay(1000);
           }
            }
             }
    break;
case '2':                                   // Channel 2
   {    if(data=='2')                       //Lamp_2  "ON"
       { output_high(PIN_D1);
         Delay(1000);
                if(input(PIN_B5))
                 { output_high(PIN_B2);
                   Delay(1000);
                   printf("!Lamp2 off light ");
                   Delay(1000);
                   output_low(PIN_B2);
                   Delay(1000);
```

```
                    }
                    else
                     { output_high(PIN_B2);
                       Delay(1000);
                       printf("Lamp2 ON ");  // Channel 2 Status of  ON
                       Delay(1000);
                       output_low(PIN_B2);
                       Delay(1000);
                      }
                    }
          else                                    //Lamp_2 "OFF"
            { output_low(PIN_D1);
              Delay(1000);
                    if(input(PIN_B5))
                     { output_high(PIN_B2);
                       Delay(1000);
                       printf("Lamp2 OFF ");     // Channel 2 Status of
OFF
                       Delay(1000);
                       output_low(PIN_B2);
                       Delay(1000);
                      }
                    else
                     { output_high(PIN_B2);
                       Delay(1000);
                       printf("Lamp2 ON ");    // Channel 2 Status of  ON
                       Delay(1000);
                       output_low(PIN_B2);
                       Delay(1000);
                      }
                   }
                 }
                  break;
case '3':                                       // Channel 3
   {    if(data=='1')
          { output_high(PIN_D2);                // Channel 3  ON
              Delay(1000);
              output_high(PIN_B2);
              Delay(1000);
              printf("CH3 ON");                 // Channel 3 Status of ON
              Delay(1000);
              output_low(PIN_B2);
              Delay(1000);
           }
         else{
              output_low(PIN_D2);               // Channel 3 OFF
              Delay(1000);
              output_high(PIN_B2);
              Delay(1000);
              printf("CH3 OFF");               // Channel 3 Status of OFF
              Delay(1000);
              output_low(PIN_B2);
              Delay(1000);
          }
    }
              break;

case '4':                                       // Channel 4
   {    if(data=='1')
            { output_high(PIN_D3);              // Channel 4 ON
              Delay(1000);
```

```
                    output_high(PIN_B2);
                    Delay(1000);
                    printf("CH4 ON");          // Channel 4 Status of ON
                    Delay(1000);
                    output_low(PIN_B2);
                    Delay(1000);
                  }
        else      {
                    output_low(PIN_D3);        // Channel 4 OFF
                    Delay(1000);
                    output_high(PIN_B2);
                    Delay(1000);
                    printf("CH4 OFF");        // Channel 4 Status of OFF
                    Delay(1000);
                    output_low(PIN_B2);
                    Delay(1000);
                    }
             }
           break;
case '5':                                      // Channel 5
   {  if(data=='1')
                  { output_high(PIN_D4);      // Channel 5 ON
                    Delay(1000);
                    output_high(PIN_B2);
                    Delay(1000);
                    printf("CH5 ON");          // Channel 5 Status of ON
                    Delay(1000);
                    output_low(PIN_B2);
                    Delay(1000);
                  }
        else      {
                    output_low(PIN_D4);       // Channel 5 OFF
                    Delay(1000);
                    output_high(PIN_B2);
                    Delay(1000);
                    printf("CH5 OFF");         // Channel 5 Status of OFF
                    Delay(1000);
                    output_low(PIN_B2);
                    Delay(1000);
                    }
             }
        break;

case '6':                                      // Channel 6
   {    if(data=='1')
                  { output_high(PIN_D5);      // Channel 6 ON
                    Delay(1000);
                    output_high(PIN_B2);
                    Delay(1000);
                    printf("CH6 ON");          // Channel 6 Status of ON
                    Delay(1000);
                    output_low(PIN_B2);
                    Delay(1000);
                  }
        else      {
                    output_low(PIN_D5);        // Channel 6 OFF
                    Delay(1000);
                    output_high(PIN_B2);
                    Delay(1000);
                    printf("CH6 OFF ");       // Channel 6 Status of OFF
                    Delay(1000);
```

```
                output_low(PIN_B2);
                Delay(1000);
              }
          }
        break;
case '7':                                       // Channel 7
   {    if(data=='1')
              { output_high(PIN_D6);       // Channel 7 ON
                Delay(1000);
                output_high(PIN_B2);
                Delay(1000);
                printf("CH7 ON ")    // Channel 7 Status of ON
                Delay(1000);
                output_low(PIN_B2);
                Delay(1000);
              }
       else   {
                output_low(PIN_D6);       // Channel 7 OFF
                Delay(1000);
                output_high(PIN_B2);
                Delay(1000);
                printf("CH7 OFF ");      // Channel 7 Status of OFF
                Delay(1000);
                output_low(PIN_B2);
                Delay(1000);
                }
         }
       break;
case '8':                                       // Channel 8
   {      if(data=='1')
              { output_high(PIN_D7);       // Channel 8 ON
                Delay(1000);
                output_high(PIN_B2);
                Delay(1000);
                printf("CH8 ON ");    // Channel 8 Status of ON
                Delay(1000);
                output_low(PIN_B2);
                Delay(1000);
              }
       else   {
                output_low(PIN_D7);       // Channel 8 OFF
                Delay(1000);
                output_high(PIN_B2);
                Delay(1000);
                printf("CH8 OFF ");    // Channel 8 Status of OFF
                Delay(1000);
                output_low(PIN_B2);
                Delay(1000);
                }
         }
       break;
case '9':                                  // Temperature
       {
          if(data=='1'){
          output_low(PIN_B0);
          Delay(1000);
          ReadTemp_DS1820();   //Go to Function Show Temperature
          Delay(1000);
          output_low(PIN_B2);
          Delay(1000);
            }
```

```
                }
              break;
             }
          }
/***************** Receive & CheckAddress **********************/
#INT_RDA                                  // Rx from RS-232
void Rx_data(void)
    {         char adds,ch,data;
               adds=getc();
               if(adds=='T')              // check byte 1 for 'T'
      {
            adds=getc();
            if(adds=='L')                 // check byte 2 for 'L'
     {
            adds=getc();
            if(adds=='E')                 // check byte 3 for 'L'
      {
            ch=getc();
             if(ch!='A')                  // check byte 4 for select Channel
        {
             data=getc();
             if(data!='A')                // check byte 5 for Select output
         {
                  output_high(PIN_B2);
                  Delay(1000);
                  Control(ch,data);
                  Delay(1000);
                  output_low(PIN_B2);
                  Delay(1000);
                  }
                }                         //data
              }                           //ch
           }                              //Check 'T'
         }                                //Check 'L'
     }                                    //Check 'E'
/********************** Main Program  ************************/
    void main(void)
    { set_tris_b(0xF0); //SET PIN B0-B3 is Output,PIN B4-B7 is Input
       port_d=0x00;
       output_low(PIN_B2);                // SET Rx Data
       enable_interrupts(GLOBAL);
       enable_interrupts(INT_RDA);
       output_low(PIN_B2);
       delay(1000);
while(TRUE); {  }
               }
```

หนังสืออ้างอิง

1. ประจิน พลังสันติกุล , " เรียนรู้และใช้งาน CCS C คอมไพเลอร์เขียนโปรแกรมภาษา C ควบคุมไมโครคอนโทรลเลอร์ PIC 16F877 " , 2521
2. ณัฏฐพล วงศ์สุนทรชัย , ชัยวัฒน์ ลิ้มพรจิตรวิไล , "เรียนรู้และปฏิบัติการ ไมโครคอนโทรลเลอร์ PIC 16F877 " , 2521