เครื่องควบคุมระยะไกลผ่านสายโทรศัพท์

นายเชิดชัย แซ่ตั้ง
นายธนชัย เวชชิรารัตน์

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2539

Long-distance Remote Control Via Public Telephone Network

Mr.Cherdchai Saetung

Mr.Thanachai Wedchirarat

A Special Project Sumitted in Partial Fulfillment of the Requirement for the Degree of Bachelor of Science

Department of Appiled Physics

Faculty of Science

King Mongkut 's Institute of Technology Ladkrabang

1996

หัวข้อโครงงานพิเศษ เครื่องควบคุมระยะไกลผ่านสายโทรศัพท์

โดย นายเชิดชัย แซ่ตั้ง

นายธนชัย เวชชิรารัตน์

ภาควิชา ฟิสิกส์ประยุกต์

อาจารย์ที่ปรึกษา ผศ. วิชิต ศิริโชติ

รศ.ดร. ปรีชา เทียนสมประสงค์

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับโครงงานพิเศษฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

_____ หัวหน้าภาควิชาฟิสิกส์ประยุกต์

(รองศาสตราจารย์ สุรพล รักวิจัย)

คณะกรรมการโครงงานพิเศษ

_____ ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ วิชิต ศิริโชติ)

_____ กรรมการ

(รองศาสตราจารย์ ดร. ปรีชา เทียนสมประสงค์)

_____ กรรมการ

(อ. สุน จ่างประยูร)

_____ กรรมการ

(อ. รัชภาค จิตต์อารีย์)

หัวข้อโครงงานพิเศษ	เครื่องควบคุมระยะไกลผ่านสายโทรศัพท์	
โดย	นายเชิดชัย	แซ่ตั้ง
	นายธนชัย	เวชิรรารัตน์
อาจารย์ที่ปรึกษา	ผศ.วิชิต	ศิริโชติ
	ผศ.ดร.ปรีชา	เทียนสมประสงค์
ปีการศึกษา	2539	

บทคัดย่อ

ระบบควบคุมระยะไกลซึ่งทำหน้าที่เพื่อควบคุมอุปกรณ์เครื่องใช้ไฟฟ้าภายในบ้านผ่านระบบเครือข่ายโทรศัพท์ โครงงานนี้ได้ทำการพัฒนาระบบควบคุมระยะไกลเพื่อควบคุมเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางสายโทรศัพท์ โดยใช้วิธีการเชื่อมต่อระหว่างโมเด็ม 2 เครื่อง ระหว่างคอมพิวเตอร์ตัวแม่(host computter) กับคอนโทรลเลอร์ ทำให้เราสามารถทำการควบคุมอุปกรณ์เครื่องใช้ไฟฟ้าภายในบ้านได้โดยผ่านสายโทรศัพท์ซึ่งจะถูกควบคุมโดย Remote Terminal โดยในส่วนของคอนโทรลเลอร์ทำการควบคุมโดยใช้ชิพคอนโทรลเลอร์ตระกูล PIC เบอร์ PIC16C57 เบสิคแสตมป์ ซึ่งมีหน่วยควบคุมความจำโปรแกรม 256 ไบต์ และมีพอร์ตอินพุตเอาท์พุต 8 ช่อง ทำการโปรแกรมได้โดยใช้โปรแกรมวิชวลเบสิค(โปรแกรมเชื่อมต่อกับผู้ใช้แบบกราฟฟิก) โดยโปรแกรมจะทำการส่งเลขหมายและคำสั่งเปิดหรือปิดที่ความเร็ว 2,400 บิตต่อวินาที ไปยังสถานที่ที่เราจะควบคุม ทำให้เราสามารถควบคุมการปิดและเปิดอุปกรณ์ไฟฟ้าภายในบ้านได้โดยตรงทั้งหมด 8 อุปกรณ์

Special Project Title	Long-distance Remote Control Via Public Telephone Network
Name	Mr.Cherdchai Daetung
	Mr.Thanachai Wedchirarat
Special Project Adviser	Asst.Prof.Wichit Sirichote
	Asst.Prof.Dr.Preecha Tiansomprosong
Department	Applied Physics
Academic	1996

Abstract

A remote controller used for controlling house's appliances via a telephone network has been developed. By using two modems to connect between host computer and the controller, the electrical appliances where-ever the telephone network reach can easily be controlled from the remote terminal. The controller employ a PIC microcontroller chip, PIC16C57 BASIC STAMP,having an internal program of 256 bytes and 8 channels I/O. The remote terminal program was developed using a Graphic User Interface development tool, Visual Basic V3.5. The program sends dialing and on/off command at speed of 2400bps to the remote controller. In addition to direct turn on and off of the eight devices remotely, timer functions are also available.

กิติกรรมประกาศ

โครงงานพิเศษนี้สำเร็จลุล่วงได้ด้วยดี เนื่องด้วยความอนุเคราะห์จากบุคคลหลายฝ่าย ดังนี้

บิดาและมารดา	**คอยให้ความห่วงใยและช่วยเหลือ**
ผศ.วิชิต ศิริโชติ	**ผู้ให้ความรู้และคำแนะนำทุกๆ ด้าน**
ผศ.ดร.ปรีชา เทียนสมประสงค์	**ผู้ให้คำแนะนำ**
คุณยุทธนา สำรวมดี	**ผู้ให้ความช่วยเหลือ**
คุณวันชนะ ทองทั้งสาย	**ผู้เอื้อเฟื้ออุปกรณ์การพิมพ์รายงาน**
คุณพันธ์ศักดิ์ โกวิชาวุฒิคุณ	**ผู้ให้ความบันเทิง**
คุณสุวิทย์ ภักดีบุรี	**ผู้ช่วยในงานด้านฮาร์ดแวร์**
ภาควิชาฟิสิกส์ประยุกต์	**ผู้ให้ความอุปการะ**
คณะกรรมการทุกท่าน	**ที่ตรวจทานรายงานโครงงานพิเศษ**
เพื่อนๆ ทุกคน	**ช่วยเหลือในด้านต่างๆ**

สารบัญ

หน้า

หน้า

สารบัญรูป

หน้า

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1ที่มาของปัญหา

ในปัจจุบันเทคโนโลยีทางด้านการสื่อสารและโทรคมนาคมมีความก้าวหน้ามากและยังได้เข้ามามีบทบาทในชีวิตประจำวันมากยิ่งขึ้นเพื่ออำนวยความสะดวกรวดเร็วในด้านการติดต่อสื่อสาร โดยระบบโทรศัพท์ก็เป็นอีกรูปแบบหนึ่งซึ่งมีความสะดวกรวดเร็ว และง่ายต่อการใช้งานเมื่อเทียบกับการสื่อสารรูปแบบอื่นๆดังนั้นถ้าเรานำคอมพิวเตอร์มาเชื่อมต่อกับระบบโทรศัพท์โดยผ่านโมเด็มซึ่งสามารถส่งข้อมูลในระยะทางไกลได้ ก็จะทำให้การสื่อสารสะดวกมากยิ่งขึ้น และเพิ่มความคล่องตัวในการรับส่งข้อมูลซึ่งประหยัดเวลาอย่างมาก ด้วยแนวความคิดดังกล่าวจึงเป็นที่มาของ **เครื่องควบคุมระยะไกลผ่านสายโทรศัพท์ (Long-distance Remote Control Via Public Telephone Network)** ซึ่งจะเพิ่มความสะดวกสะบายให้กับการทำงานอุปกรณ์ปลายทางระยะไกลได้อย่างมีประสิทธิภาพ

1.2วัตถุประสงค์ในการทำโครงงานพิเศษ

1) เพื่อศึกษาการทำงานของโมเด็ม
2) เพื่อศึกษาการทำงานของไมโครคอนโทรลเลอร์ PIC16C56
3) เพื่อศึกษาและออกแบบระบบควบคุมอุปกณ์เครื่องใช้ไฟฟ้าต่างๆระยะไกลด้วยคอมพิวเตอร์โดยอาศัยการส่งผ่านข้อมูลผ่านโมเด็มเชื่อมต่อกับสายโทรศัพท์
4) เพื่อนำความรู้ทางด้านฟิสิกส์มาประยุกต์สร้างเครื่องมือให้เกิดประโยชน์
5) เรียนรู้ลักษณะการทำงานที่เป็นระบบ และรู้จักวิเคราะห์ปัญหา

1.3ขั้นตอนการดำเนินงาน

1) ศึกษาการทำงานของโมเด็มกับการส่งข้อมูลผ่านโทรศัพท์
2) ศึกษาการทำงานของชิพไมโครคอนโทรลเลอร์ PIC 16C56 (Basic Stamp)
3) เขียนโปรแกรมควบคุมชิพ PIC 16C56 ให้ทำงานรับส่งข้อมูล
4) เขียนโปรแกรม Visaul Basic ควบคุมการทำงานบนคอมพิวเตอร์ PC ระยะไกล
5) ออกแบบวงจร Driver สำหรับควบคุมอุปกรณ์ต่างๆปลายทาง

6) เชื่อมวงจรแต่ละส่วนเข้าด้วยกัน และทดสอบการส่งข้อมูลไปยังปลายทาง

7) ประกอบชิ้นงานให้เสร็จสมบูรณ์ พร้อมที่จะไปใช้งานจริง

1.4ประโยชน์ที่ได้รับ

1) ทำให้เข้าใจการทำงานของโมเด็มกับการส่งผ่านข้อมูล

2) ทำให้เข้าใจการทำงานของระบบไมโครคอนโทรลเลอร์ PIC 16C56 และสามารถนำไปประยุกต์ใช้งานได้อย่างมีประสิทธิภาพ

3) สามารถนำโปรแกรม Visaul Basic ไปประยุกต์ใช้งานด้านต่างๆได้

4) ทำให้รู้จักค้นคว้าหาข้อมูลเพิ่มเติมจากแหล่งต่างๆ เพื่อเป็นแนวทางในการแก้ปัญหา

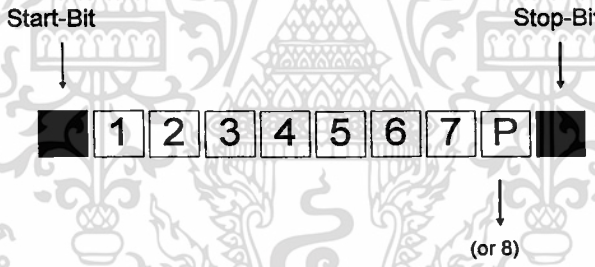
5) รู้จักการทำงานอย่างมีระบบ วิเคราะห์และแก้ไขปัญหาอย่างมีหลักการและเหตุผล

บทที่ 2
ทฤษฎี

2.1 โมเด็ม

2.1.1การรับส่งข้อมูลแบบ อนุกรม

การรับส่งข้อมูลแบบอนุกรมนั้นมีชื่อเรียกว่า Serial Interface หรือ RS-232C การรับส่งข้อมูลแบบนี้จะนำข้อมูลหนึ่งไบต์ส่งออกไปทางสายทีละหนึ่งบิตเรียงกันไปจนครบ 8 บิต จากการที่ส่งข้อมูลเรียงกันไปนี้ จำนวนสายที่ส่งข้อมูลจึงมีเพียง 3 ถึง 5 เส้นเท่านั้น ความซับซ้อนอยู่ตรงที่ทำอย่างไรทางด้านรับจึงจะรู้ว่า ข้อมูลมาถึงเมื่อไร ตรงไหนคือข้อมูลบิตแรก บิตที่สอง ไปจนถึงบิตสุดท้าย เราจึงต้องเพิ่มส่วนเริ่มต้นข้อมูลและส่วนปิดท้ายข้อมูลเข้าไปด้วยเรียกว่า Start Bit และ Stop Bit



รูปที่2.1.1 Start Bit และ Stop Bit จะช่วยให้คอมพิวเตอร์แยกข้อมูลแต่ละตัวออกมาได้อย่างถูกต้อง

คราวนี้ผู้รับหรือคอมพิวเตอร์ที่รับข้อมูลก็จะสามารถแยกแยะสัญญาณที่ได้รับออกมาเป็นข้อมูลได้ถูกต้อง ข้อดีของการส่งข้อมูลแบบอนุกรมก็คือ เหมาะสำหรับส่งข้อมูลระยะทางไกล มากกว่าการส่งข้อมูลแบบขนาน เพราะใช้จำนวนสายน้อยกว่าและระดับแรงดันไฟฟ้าที่ใช้ในการส่งมีค่า +12 โวลต์ กับ -12 โวลต์ ทำให้เราสามารถส่งข้อมูลได้ไกลถึง 35 เมตร โดยไม่ต้องมีอุปกรณ์เพิ่มเติมเข้าช่วยเลย ข้อเสียของการส่งข้อมูลแบบอนุกรมคือ ความเร็วในการส่งข้อมูลจำกัดอยู่ที่ 19,200 บิตต่อวินาทีสูงสุด นับว่าช้ากว่าการส่งข้อมูลแบบขนานอยู่มากทีเดียว นอกจากนี้วงจรฮาร์ดแวร์ที่ใช้ในการรับส่งข้อมูลแบบอนุกรมยังมีราคาแพงกว่าอีกด้วย

การส่งข้อมูลแบบอนุกรมนี้ เราต้องคำนึงถึงรายละเอียดในการส่งข้อมูลมากกว่าการส่งแบบขนานหลายอย่าง เช่น ความเร็วในการรับส่งข้อมูล การตรวจสอบความถูกต้องของข้อมูล จำนวนบิตของข้อมูล ฯลฯ ทั้งหมดนี้ถ้ามีอะไรไม่ตรงกันระหว่างผู้รับและผู้ส่ง การส่งข้อมูลแบบอนุกรมก็จะผิดพลาดหรือรับส่งกันไม่ได้

สายเคเบิลของ RS-232C

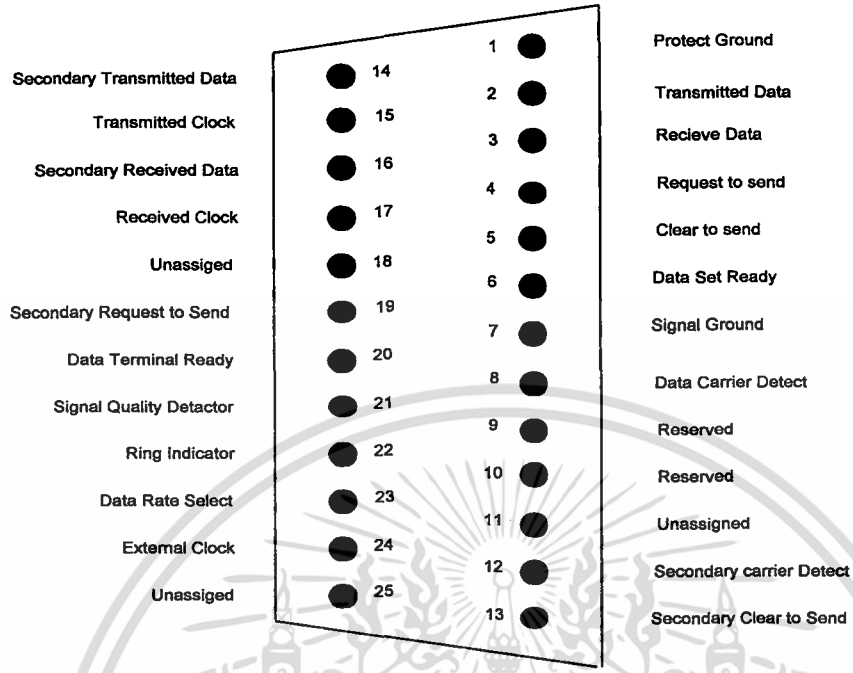
การรับส่งข้อมูลแบบอนุกรมของคอมพิวเตอร์หรือที่เรียกว่า RS-232C นั้น ใช้กันมากในการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ คอมพิวเตอร์กับโมเด็ม คอมพิวเตอร์กับอุปกรณ์ต่อพ่วงแบบต่างๆ เช่น เมาส์ เครื่องวาดภาพ(Plotter) เครื่องพิมพ์บางชนิดที่ใช้พอร์ตอนุกรม รวมทั้งอุปกรณ์วัดสัญญาณต่างๆ ทางวิทยาศาสตร์ก็มักรับส่งข้อมูลกับคอมพิวเตอร์ผ่านทาง RS-232C นี้ การส่งข้อมูลแบบอนุกรมจึงจัดเป็นมาตรฐานที่ใช้กันกว้างขวางวิธีหนึ่ง

มาตรฐานของการรับส่งข้อมูลแบบอนุกรม(RS-232C) นี้ มีการกำหนดขึ้นมาเพื่อให้คอมพิวเตอร์ต่างยี่ห้อกัน หรืออุปกรณ์ต่อพ่วงแต่ละชนิดรับส่งข้อมูลกันได้ เมื่อทำตามมาตรฐานนี้ โดยไม่สนใจว่าอุปกรณ์หรือคอมพิวเตอร์นั้นจะผลิตมาจากที่ใด โดยมีการกำหนดรายละเอียดในการรับส่งข้อมูล เช่น ข้อต่อ(Connector) ที่ใช้เป็นแบบใด มีสัญญาณที่ใช้กี่เส้น แต่ละสัญญาณทำหน้าที่อะไร และใช้ระดับแรงดันไฟฟ้าเท่าไรในการรับส่งข้อมูล ความเร็วในการรับส่งข้อมูลจะเป็นเท่าใดบ้าง ใช้ข้อมูลกี่บิตในการรับส่งข้อมูล ฯลฯ อุปกรณ์หรือคอมพิวเตอร์ก็จะทำตามมาตรฐานนี้ ทำให้สามารถรับส่งข้อมูลได้อย่างไม่มีปัญหา ในตอนนี้เราจะมาดูรายละเอียดกันว่า สัญญาณต่างๆ ของ RS-232C มีอะไรบ้าง และสายเคเบิลที่ใช้เชื่อมต่อส่งข้อมูลกันจะต้องต่อสายอย่างไร

ขาต่างๆของ RS-232C

เริ่มจากตัวข้อต่อ (Connector) ระหว่างสายเคเบิลทั้งสองปลาย จะใช้ข้อต่อแบบ 25Pin รูปร่างหน้าตัดคล้ายตัว “D” มีชื่อเรียกว่า DB-25 ดังแสดงในรูป 2.1.2

กำหนดการใช้งานเอาไว้ทั้งหมด 22 ขา ไม่ได้ใช้ 3 ขา สัญณาณแต่ละขาจะทำหน้าที่ของมันตามที่กำหนดเอาไว้ แต่ปรกติแล้วในการรับส่งข้อมูลทั่วไปเราใช้สัญญาณเพียง 8 ถึง 9 เส้นเท่านั้นก็พอ สัญญาณที่เหลือเราไม่นำมาใช้ เนื่องจากว่าบางเส้นเป็นสัญญาณรับส่งข้อมูล และสัญญาณควบคุมของช่องสัญญาณสำรอง(Secondary Channel) บางเส้นปล่อยว่างไว้ และบางเส้นใช้สำหรับงานพิเศษบางอย่างเท่านั้น



รูป 2.1.2 แสดงข้อต่อแบบ DB-25 และขาต่างๆของมัน

สายเคเบิลที่ใช้รับส่งข้อมูลส่วนมากจึงใช้สายเพียง 8 ถึง 9 เส้นเท่านั้นจากข้อต่อ 25 ขา สัญญาณแต่ละเส้นเรียงตามลำดับดังนี้ ขาที่ 1,2,3,4,5,6,7,8,และ 20 กับ 22 โดยที่ขาที่ 1 (Protective Ground) นั้น มักจะไม่จำเป็นต้องต่อใช้งาน จึงเหลือจำนวนสายที่ใช้เพียง 9 เส้น หน้าที่ของสัญญาณแต่ละเส้นก็คือ

- ขาที่ 1 (Protective Ground) เป็นสายดินของอุปกรณ์
- ขาที่ 2 (Transmitted Data) ใช้สำหรับส่งข้อมูล
- ขาที่ 3 (Received Data) ใช้สำหรับรับข้อมูล
- ขาที่ 4 (Request to Send) เป็นสัญญาณขอทำการส่งข้อมูล
- ขาที่ 5 (Clear to Send) เป็นสัญญาณตอบรับว่าเริ่มส่งข้อมูลได้
- ขาที่ 6 (Data Set Ready) เป็นสัญญาณแสดงว่าตัวรับพร้อมที่จะรับข้อมูลแล้ว
- ขาที่ 7 (Signal Ground) เป็น Ground ของสัญญาณรับส่ง
- ขาที่ 8 (Data Carrier Detect) เป็นตัวบอกว่าทั้งตัวรับและตัวส่งต่อถึงกันเรียบร้อย แล้วพร้อมที่จะทำการรับส่งข้อมูล ในกรณีที่ใช้ต่อกับโมเด็ม ขานี้จะเป็นตัวบอกว่าโมเด็มทั้งสองด้านต่อถึงกันได้แล้วโดยมีสัญญาณ Carrier ส่งถึงกัน

- ขาที่ 20 (Data Terminal Ready) เป็นสัญญาณแสดงว่าตัวส่งพร้อมที่จะส่งข้อมูล
- ขาที่ 22 (Ring Indicator) เป็นขาแสดงแทนกริ่งโทรศัพท์ที่เรียกเข้ามา การเชื่อมต่อบางอย่างก็จะไม่ใช้ขาที่ 22 นี้ในการทำงาน

ส่วนขาอื่นๆที่เหลือนั้น ส่วนมากมีหน้าที่คล้ายกับ 8 ขาแรกที่กล่าวมา และบางเส้นใช้กับงานพิเศษเท่านั้นจึงไม่ขอกล่าวถึง ขาที่เราใช้สำหรับรับส่งข้อมูลของข้อต่อแบบ DB-25 จึงเหลือเพียงขา 2,3,4,5,6,7,8,20 และ 22 ยกเว้นการต่องานบางอย่างถึงจะต่อครบทุกเส้น

DB-25 และ DB-9

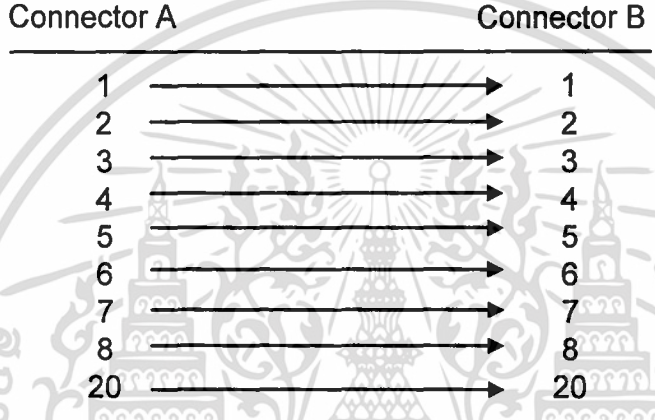
จากการที่ข้อต่อแบบ 25 ขา เราใช้งานจริงเพียง 9 ขาเท่านั้น เครื่องคอมพิวเตอร์รุ่นใหม่ๆ จึงได้ลดข้อต่อลงมาใช้แบบ 9 ขาแทน ซึ่งเราเรียกข้อต่อแบบนี้ว่า DB-9 การใช้ข้อต่อแบบ DB-9 นี้มีข้อดีหลายอย่างคือ ขนาดเล็กกะทัดรัด ราคาของข้อต่อถูกกว่า การต่อสายเคเบิลสะดวกขึ้น และการใช้งานคล่องกว่า DB-25 สัญญาณต่างๆของข้อต่อแบบ DB-9 บางเส้นจะตรงกับที่ใช้ใน DB-25 ดังที่แสดงในตารางเปรียบเทียบ เครื่องคอมพิวเตอร์แบบ IBM และรุ่นใหม่ๆ มักจะใช้ข้อต่อแบบ DB-9 สำหรับรับส่งข้อมูลอนุกรมทั้งนั้น แต่อุปกรณ์ต่อพ่วงส่วนมากยังคงใช้ข้อต่อแบบ DB-25 อยู่เราจึงต้องใช้สายเคเบิลที่เหมาะสมสำหรับทั้งสองด้านในการรับส่งข้อมูล

DB-9 Pin	DB-25 Pin	Assignment/Function
1	8	Carrier detect
2	3	Receive data
3	2	Transmit data
4	20	Data terminal ready
5	7	Signal Ground
6	6	Data set ready
7	4	Request to send
8	5	Clear to send
9	22	Ring Indicator

รูปที่ 2.1.3 การใช้งานรับส่งข้อมูลอนุกรม เราใช้สัญญาณเพียง 9 ขาเท่านั้น

สายเคเบิลของการรับส่งข้อมูลอนุกรมแบ่งออกได้เป็น 2 แบบ คือ สายตรงและสายสลับ ที่ต้องมี 2 สายแบบนี้ก็เพราะว่าการเชื่อมต่อส่งข้อมูลมี 2 กรณี คือ คอมพิวเตอร์ต่อกับคอมพิวเตอร์ และคอมพิวเตอร์ต่อเข้ากับอุปกรณ์ต่างๆ เมื่อเราต่อคอมพิวเตอร์เข้ากับคอมพิวเตอร์เพื่อรับส่งข้อมูลกัน สายสัญญาณรับส่งข้อมูลต้องสลับไขว้

กัน เพื่อให้สัญญาณส่งของตัวแรกไปเข้าสัญญาณรับของตัวที่สอง เราจึงเรียกสายเคเบิลแบบนี้ว่าสายสลับ ส่วนการต่อคอมพิวเตอร์เข้ากับอุปกรณ์ต่อพ่วงนั้น สายสัญญาณของอุปกรณ์ต่อพ่วงเช่น โมเด็มและพล็อตเตอร์ (Plotter) มักจะสลับสัญญาณรอรับไว้ภายในแล้ว สายเคเบิลจากเครื่องคอมพิวเตอร์จึงต่อตรงเข้าแต่ละเส้นของอุปกรณ์ได้เลย เราจึงเรียกสายเคเบิลแบบนี้ว่าสายตรง กรณีที่วงจรของอุปกรณ์ต่อพ่วงไม่ได้สลับสายไว้ภายใน เราจึงต้องใช้สายสลับต่อระหว่างคอมพิวเตอร์กับอุปกรณ์นั้น ไม่จำเป็นต้องใช้สายตรงเสมอไป



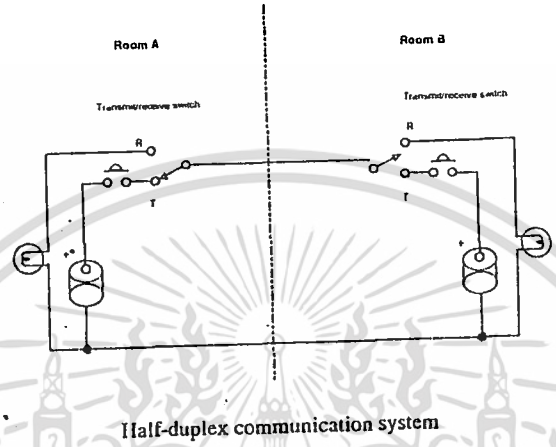
รูป 2.1.4 การต่อสายตรงของ RS-232 อย่างง่าย

การต่อสายเคเบิลแบบสายตรงนั้นไม่ยุ่งยากเท่าใดนัก เนื่องจากสัญญาณแต่ละเส้นตามเบอร์ต่างๆของ DB-25 จะเชื่อมต่อเข้าหากันตรงๆ ทั้ง 8 เส้นหรือ 9 เส้น ตามสัญญาณที่ใช้อย่างที่กล่าวไว้ในตอนต้น เพียงเท่านี้ก็สามารถรับส่งข้อมูลได้ การที่สัญญาณรับส่งข้อมูลและสัญญาณควบคุมต่อเข้าคู่ของมันตรงๆ ทำให้สายเคเบิลแบบนี้ทำขึ้นอย่างง่ายๆได้ โดยใช้ข้อต่อแบบไม่ต้องบัดกรีและสายเคเบิลแบบแผ่น (Ribbon Cable) เท่านั้น

2.1.2 Full duplex และ Half duplex

ส่วนการรับส่งแบบที่สองนี้ เราเรียกว่า การรับส่งแบบ Half Duplex มีคุณสมบัติสามารถรับและส่งข้อมูลได้ แต่ว่าต้องสลับกันส่ง จะส่งพร้อมกันทั้ง 2 ด้านไม่ได้ อุปกรณ์ที่ใช้การติดต่อในแบบ Half Duplex ได้แก่ วิทยุมือถือ(Walkie-talkie) และ INTERCOM เป็นต้น เมื่อฝ่ายใดฝ่ายหนึ่งส่ง อีกฝ่ายหนึ่งก็จะทำหน้าที่รับ จนกระทั่งฝ่ายแรกส่งจบฝ่ายหลังจึงจะกลับเป็นผู้ส่งได้ และฝ่ายส่งในตอนแรกก็จะเป็นผู้รับ สลับกันเช่นนี้เรื่อยไป ทั้ง 2

ฝ่ายจะเป็นผู้ส่งพร้อมกันไม่ได้ เพราะสัญญาณจะชนกันทำให้ฟังไม่รู้เรื่อง การรับส่งในแบบ Half Duplex นั้นทั้ง 2 ด้านสามารถทำหน้าที่รับและส่งได้ตามลำดับ



Half-duplex communication system

รูปที่ 2.1.5 การรับส่งข้อมูลสวนทางกันได้แบบผลัดกันส่งหรือ Half Duplex

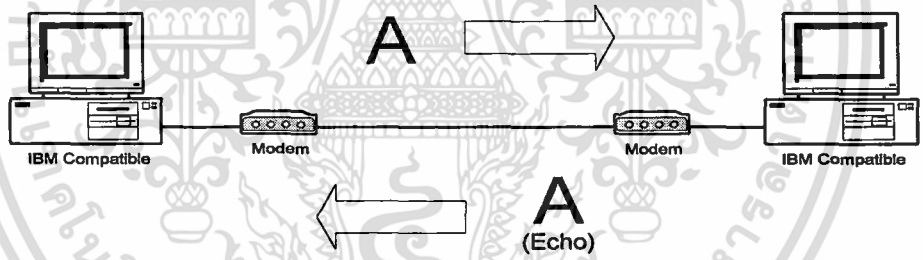
แบบที่ซับซ้อนที่สุด ก็คือ การรับส่งในแบบสวนทางได้พร้อมกัน ซึ่งเรียกว่า Full Duplex การรับส่งแบบนี้ ผู้รับและส่งสามารถรับและส่งพร้อมๆกันในเวลาเดียวได้ ไม่จำเป็นต้องรอให้อีกฝ่ายหนึ่งส่งจบเสียก่อนอย่างใน Half Duplex ตัวอย่างเช่น การพูดโทรศัพท์ของเรา ถึงแม้ปรกติเมื่อผู้หนึ่งพูดอีกฝ่ายจะต้องคอยฟัง แล้วตอบกลับมาเมื่อฝ่ายแรกพูดจบ ซึ่งเป็นลักษณะของการติดต่อแบบ Half Duplex ก็ตามแต่ เราอาจจะพูดพร้อมๆกัน หรือพูดสวนกลับไปได้ทันทีโดยยังคงฟังอยู่เหมือนเดิม ลักษณะเช่นนี้เราเรียกว่าติดต่อกันแบบFull Duplex การติดต่อสื่อสารข้อมูลระหว่างคอมพิวเตอร์ 2 เครื่อง มีใช้ทั้งแบบ Half Duplex และ Full Duplex ขึ้นอยู่กับลักษณะของการเชื่อมต่อ และงานของมัน

<u>ลักษณะของการรับส่ง และการ Echo ของข้อมูล</u>

เมื่อคอมพิวเตอร์รับส่งข้อมูลในแบบ Half Duplex หรือ Full Duplex ก็ตามมันจะต้องใช้การรับส่งให้เหมือนกันทั้ง 2 ด้าน การรับส่งข้อมูลในแบบ Half Duplex เครื่องคอมพิวเตอร์ผลัดกันส่งข้อมูลและมีหน้าที่พิมพ์ข้อความที่ตัวเองส่งออกไปขึ้นแสดงบนจอภาพด้วย หรือมีคุณสมบัติที่เราเรียกว่า "Echo On" นั่นเอง ข้อความไม่ว่าจะส่งออกไปโดย

การพิมพ์จากแป้นพิมพ์หรืออ่านจากแผ่นดิสก์ก็ตาม เครื่องคอมพิวเตอร์ที่ส่งข้อความจะต้องนำข้อความนั้นมาแสดงผลทางจอภาพด้วยตนเอง การรับส่งข้อมูลในแบบ Half Duplex จึงต้องกำหนดให้เครื่องคอมพิวเตอร์หรือเทอร์มินัลที่ใช้ ทำงานในลักษณะ Echo On เสมอ มิฉะนั้นเราจะมองไม่เห็นข้อความที่เราส่งออกไป มองเห็นแต่เฉพาะข้อความที่อีกฝ่ายหนึ่งส่งมาเท่านั้น

เมื่อคอมพิวเตอร์รับส่งข้อมูลในแบบ Full Duplex มันก็สามารถรับและส่งข้อมูลสวนทางกันได้ในเวลาเดียวกัน เครื่องคอมพิวเตอร์ที่รับส่งแบบ Full Duplex จะไม่พิมพ์ข้อความที่ตัวเองส่งออกไปขึ้นแสดงบนจอภาพ แต่จะรอรับข้อความจากอีกฝ่ายหนึ่งส่งกลับมาให้เท่านั้น เราเรียกว่า "Echo Off" ข้อความจากแป้นพิมพ์และจากแผ่นดิสก์ที่ส่งออกไปปรกติเราจะมองไม่เห็นเครื่องคอมพิวเตอร์อีกด้านหนึ่งจะส่งข้อความกลับมาให้ปรากฏบนจอภาพของเราเองเมื่อมีความจำเป็น การรับส่งข้อมูลแบบ Full Duplex จึงต้องกำหนดให้คอมพิวเตอร์ทำงานในลักษณะ Echo Off เสมอ ถ้ากำหนดผิดเราจะเห็นข้อความที่พิมพ์ออกไปกลายเป็นสองตัวซ้อนกันอย่างนี้บนจอภาพ "Liikkee TThhiiss"



รูป 2.1.6 การรับส่งแบบ Full Duplex ผู้รับจะต้องส่งข้อมูลกลับ (Echo) ไปให้ผู้ส่งเสมอ

ในการเชื่อมต่อระหว่างคอมพิวเตอร์นั้น การรับส่งแบบ Half Duplex จะมีประสิทธิภาพต่ำกว่า เนื่องจากต้องผลัดกันส่งข้อมูล แต่ก็มีข้อดี คือ ประหยัดสายส่งข้อมูล เพราะเราสามารถใช้สายเพียงคู่เดียวในการรับส่งข้อมูลแบบ Half Duplex นี้ การรับส่งแบบ Full Duplex จะต้องใช้สายสองคู่ คือ สำหรับส่งข้อมูลหนึ่งคู่ และรับข้อมูลอีกหนึ่งคู่ แยกวงจรรับส่งให้เป็นอิสระออกจากกัน ประสิทธิภาพในการรับส่งข้อมูลจึงสูงกว่าในแบบ Half Duplex ถึง 2 เท่า

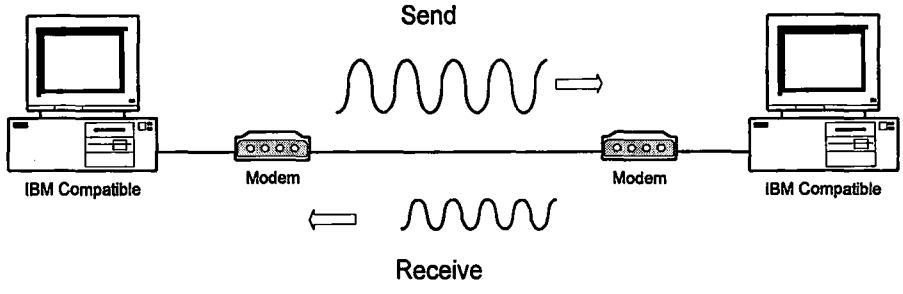
จากการที่ Full Duplex มีข้อดีหลายอย่างในการรับส่งข้อมูล แต่มีข้อเสียตรงที่ต้องใช้สายสองคู่หรือสี่เส้นเพื่อส่งข้อมูล จึงได้มีผู้พยายามพัฒนาลดจำนวนสายส่งลงให้เหลือเพียงคู่เท่ากับที่ใช้ใน Half Duplex ทั้งนี้เนื่องจากค่าใช้จ่ายของสายส่งระยะทางไกลๆมี

ราคาแพง ถ้าต้องใช้สายถึงสองคู่ก็จะทำให้ค่าใช้จ่ายของสายส่งข้อมูลแพงมากจนเกินไป สำหรับการเชื่อมต่อระยะใกล้ๆ เราอาจยอมใช้สายสองคู่รับส่งข้อมูลแยกจากกันได้ แต่การใช้งานระยะทางไกลแล้ว สายคู่เดียวจะประหยัดและสะดวกกว่ามาก โดยเฉพาะอย่างยิ่งในกรณีที่เรารับส่งข้อมูลด้วยความเร็วไม่สูงมากนัก การใช้สายเพียงคู่เดียวส่งข้อมูลในแบบ Full Duplex เป็นสิ่งที่เป็นไปได้ และนับเป็นการใช้งานสายส่งข้อมูลอย่างคุ้มค่าอีกด้วย

คุณสมบัติของสายส่งข้อมูลนั้น จะสามารถส่งสัญญาณไฟฟ้าผ่านไปยังปลายทางได้ โดยมีช่วงความถี่ช่วงหนึ่ง ช่วงความถี่ที่สายส่งข้อมูลส่งผ่านไปถึงปลายทางได้นี้ เราเรียกว่า Bandwidth ของสายส่ง อย่างเช่น สายโทรศัพท์ที่เราใช้กันอยู่ทุกวันนี้ รับส่งความถี่ได้ในช่วง 300Hz ถึง 3,000Hz. ความถี่ที่สูงกว่า และต่ำกว่านี้จะถูกดูดกลืน หรือถูกกำจัดออกไป ทำให้ส่งไปถึงปลายทางไม่ได้

ในการรับส่งข้อมูลต่างๆ ที่ความเร็วไม่สูงนัก เราจะไม่ได้ใช้ความถี่ทั้งหมดของสายในการรับส่งข้อมูล ดังนั้นจึงมีผู้คิดขึ้นมาว่า ความถี่ที่ไม่ได้ใช้ของสายส่งที่เหลือ น่าจะนำมาใช้ให้เป็นประโยชน์ได้ โดยการแบ่งความถี่ของสายส่งออกเป็น 2 ส่วน ส่วนที่หนึ่งใช้สำหรับส่ง และส่วนที่สองใช้สำหรับรับ ข้อมูลเทคนิคอันนี้เรียกว่า Frequency Divisionเช่นด้านส่งใช้ความถี่ 1,200Hz. และด้านรับใช้ความถี่ 2,400Hz.เป็นต้น เพียงเท่านี้เราก็สามารถรับส่งข้อมูลในแบบ Full Duplex ผ่านสายเพียงคู่เดียวได้

ขีดจำกัดของการใช้เทคนิคแบ่งความถี่ของสายส่ง ก็คือ สายส่งจะต้องมีช่วงกว้างของความถี่ (Bandwidth) มากพอที่จะแบ่งออกได้ โดยไม่รบกวนกันระหว่างความถี่รับและความถี่ส่ง ส่วนความเร็วในการส่งข้อมูลของแต่ละด้านสูงสุดนั้น ขึ้นอยู่กับเทคนิคการเข้ารหัสที่ใช้ในการส่งเป็นหลัก ข้อจำกัดอีกอันหนึ่งของการแบ่งความถี่ก็คือจะนำมาใช้กับการรับส่งข้อมูลในแบบดิจิตอล (Digital) ของคอมพิวเตอร์โดยตรงไม่ได้ เนื่องจากสัญญาณข้อมูลข้อมูลคอมพิวเตอร์ดิจิตอลเป็นรูปคลื่นแบบสี่เหลี่ยม ซึ่งประกอบด้วยความถี่หลายๆความถี่รวมกันขึ้นมา เราจึงแบ่งความถี่ของสายส่งเพื่อส่งรูปคลื่นสี่เหลี่ยมจากสัญญาณของคอมพิวเตอร์ไม่ได้ ต้องส่งโดยวิธีทางอ้อม คือ แปลงสัญญาณข้อมูลของคอมพิวเตอร์ที่เป็นรูปสี่เหลี่ยมให้กลายเป็นสัญญาณต่อเนื่องในแบบอนาล็อก (Analog) เสียก่อน ถึงจะรับส่งข้อมูลผ่านสายคู่เดียวในแบบ Full Duplex ได้ เทคนิคนี้เป็นเทคนิคที่โมเด็มใช้รับส่งข้อมูลผ่านสายโทรศัพท์ซึ่งมี 2 สาย และส่งข้อมูลแบบ Full Duplex ที่เราใช้กันอยู่ทุกวันนี้นั่นเอง

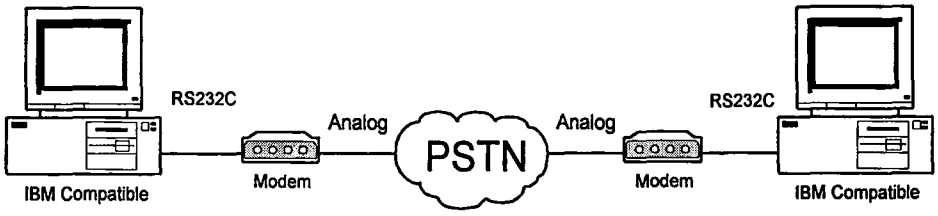


รูปที่ 2.1.7 เมื่อด้านรับและด้านส่งใช้ความถี่คนละความถี่ เราสามารถใช้สายคู่เดียวกันรับส่งแบบ Full Duplex ได้

2.1.3 โมเด็มและหลักการทำงาน

การรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ในระยะทางไม่ไกลมากนัก เราอาจใช้การรับส่งแบบอนุกรม (RS-232C) ซึ่งส่งข้อมูลดิจิตอลของคอมพิวเตอร์ไปตามสายจนถึงผู้รับได้ กรณีนี้เราสามารถรับส่งข้อมูลได้ไกลถึง 35 เมตรตามคุณสมบัติของ RS-232C หรือถ้าสายเคเบิลที่ใช้มีคุณภาพดีอาจส่งได้ไกลถึง 150 เมตรที่มีความเร็ว 9,600 บิตต่อวินาที แต่สำหรับระยะทางที่ไกลมากๆ เช่น หลายสิบกิโลเมตร หลายร้อยกิโลเมตรจนถึงหลายพันกิโลเมตร การส่งข้อมูลแบบดิจิตอลออกไปโดยตรงจะไม่เหมาะสมหลายอย่าง ปัญหาที่สำคัญก็คือ คลื่นรูปสี่เหลี่ยมของสัญญาณดิจิตอล เมื่อส่งไปไกลๆจะเพี้ยนหรือมีรูปร่างผิดไปจากเดิมได้ง่าย ทำให้สายส่งและวงจรรับส่งสัญญาณดิจิตอลต้องถูกออกแบบมาเป็นอย่างดี ราคาของสายส่งสัญญาณแบบดิจิตอลจึงมีราคาแพงกว่าสายส่งสัญญาณแบบอนาล็อกมาก ในทางปฏิบัติเราอาจรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง โดยใช้สัญญาณดิจิตอลผ่านสายส่งได้ ซึ่งทั้งสายส่งและวงจรเชื่อมต่อทั้งหมดเป็นแบบดิจิตอล แต่ค่าใช้จ่ายจะมีราคาแพงมาก จนกระทั่งไม่ค่อยคุ้มที่จะทำเช่นนี้ วิธีหลีกเลี่ยงก็คือ หาทางส่งข้อมูลไปตามสายส่งในแบบอนาล็อกแทน การทำเช่นนี้เราจำเป็นต้องใช้อุปกรณ์เข้าช่วยแปลงสัญญาณในการรับส่งข้อมูลทั้ง 2 ด้าน ซึ่งเป็นที่มาของโมเด็มนั่นเอง

โมเด็ม (MODEM) จะทำหน้าที่ แปลงสัญญาณจากคอมพิวเตอร์ที่ส่งมาทาง RS-232C ให้กลายเป็นสัญญาณอนาล็อกแล้วส่งออกไปตามสายส่ง กระบวนการนี้เราเรียกว่า การ Modulate สัญญาณเมื่อถึงปลายทางโมเด็มก็จะแปลงสัญญาณอนาล็อกที่ได้รับกลับมาเป็นสัญญาณดิจิตอล แล้วส่งให้คอมพิวเตอร์ต่อไปในรูปของสัญญาณดิจิตอลผ่านทาง RS-232C เช่นกัน กระบวนการแปลงสัญญาณกลับนี้ เรียกว่า Demodulation ชื่อของโมเด็มก็ได้จากการทำงานทั้งสองแบบนี้นั่นเอง



รูปที่2.1.8 โมเด็มช่วยให้คอมพิวเตอร์รับส่งข้อมูลผ่านสายโทรศัพท์ได้โดยเปลี่ยนสัญญาณให้เป็นเสียงก่อน

สัญญาณอนาล็อกมีคุณสมบัติเหมาะสมที่จะส่งไปไกลๆมากกว่าสัญญาณแบบดิจิตอล เพราะว่าสัญญาณอนาล็อกจะเพี้ยนหรือมีรูปร่างผิดไปจากเดิมยากกว่า และสูญเสียกำลังในสายส่งน้อยกว่า ทำให้ส่งได้ระยะทางไกลมากขึ้น นอกจากนี้ยังสามารถกรองสัญญาาณรบกวนบางส่วนที่ไม่ต้องการ(Filter) ออกได้อีกด้วย ราคาของสายส่งและอุปกรณ์เชื่อมต่อก็มีราคาถูก เราจึงจำเป็นต้องใช้โมเด็มในการรับส่งข้อมูลคอมพิวเตอร์ระยะทางไกลผ่านสายส่งอนาล็อก

จากการที่โมเด็มแปลงสัญญาณจากคอมพิวเตอร์ให้กลายเป็นสัญญาณอนาล็อกในการรับส่งข้อมูลนี้เอง ถ้าโมเด็มแปลงสัญญาณออกมาอยู่ในรูปของเสียงซึ่งเป็นสัญญาณอนาล็อกแบบหนึ่ง เราก็สามารถรับส่งข้อมูลผ่านทางสายโทรศัพท์ได้ โมเด็มทั่วๆไปที่เราใช้งานจะเป็นโมเด็มที่แปลงสัญญาณจากคอมพิวเตอร์ให้อยู่ในรูปคลื่นเสียงทั้งนั้น มีโมเด็มบางชนิดที่แปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกความถี่สูง แต่โมเด็มแบบนี้มีใช้งานน้อยและส่งข้อมูลโดยใช้สายส่งพิเศษจะส่งผ่านสายโทรศัพท์ธรรมดาไม่ได้ เราจึงเน้นเฉพาะโมเด็มที่ทำงานในย่านความถี่เสียงเท่านั้น ไม่ว่าโมเด็มจะเป็นแบบไหนก็ตาม เมื่อได้รับข้อมูลดิจิตอลจากคอมพิวเตอร์ มันจะเปลี่ยนให้เป็นสัญญาณอนาล็อก จากนั้นก็นำสัญญาณอนาล็อกที่ได้นี้มารวมเข้ากับสัญญาณพาหะ (Carrier Wave) แล้วส่งออกไปทางสายส่งข้อมูล สัญญาณพาหะนี้จะทำหน้าที่พาข้อมูลที่อยู่ในรูปสัญญาณอนาล็อกไปจนถึงปลายทาง

<u>โมเด็มสำหรับ Leased Line และ Dial up line</u>

โมเด็มแบ่งตามการใช้งานได้ 2 แบบ คือ โมเด็มที่ใช้กับสายตรง (Leased Line) และโมเด็มที่ใช้กับสายโทรศัพท์ (Dial Up Line) โมเด็มที่ใช้กับสายตรงหรือสายเช่าจะส่งข้อมูลด้วยความเร็วสูง(9,600 บิตต่อวินาทีหรือสูงกว่า) ผ่านสายไปยังจุดหมายปลายทางตายตัว ซึ่งเป็นการติดต่อในลักษณะจุดต่อจุด (Point to Point) จะต่อไปยังจุดอื่นๆไม่ได้ ส่วนมาก

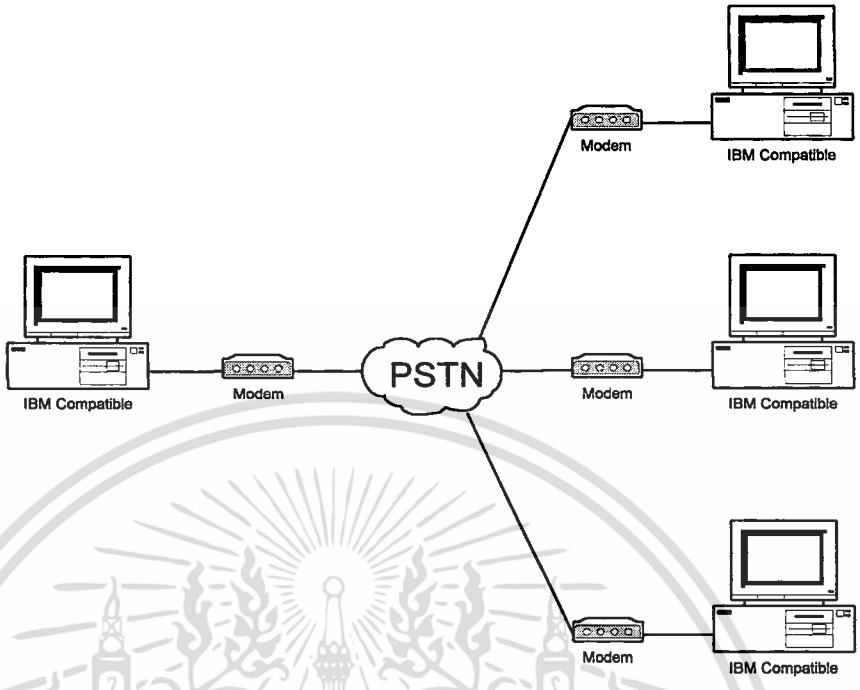
จะเป็นการใช้ติดต่อส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ที่มีข้อมูลส่งไปมาเป็นจำนวนมาก เช่น เครือข่าย ATM ของธนาคาร, เทอร์มินัลของคอมพิวเตอร์ขนาดใหญ่ ซึ่งอาจจะเป็นมินิคอมพิวเตอร์หรือเมนเฟรม เป็นต้น การส่งข้อมูลมักจะส่งเป็นกลุ่มและมีซอฟต์แวร์ควบคุมการรับส่งโดยเฉพาะ ที่เราเรียกว่า การรับส่งแบบ Synchronous นั่นเอง

ข้อดีของโมเด็มแบบที่ใช้กับสายตรง คือ ส่งข้อมูลได้ที่ความเร็วสูง เนื่องจากสายส่งมีคุณภาพเหมาะสมกับงานส่งข้อมูลจำนวนมากระหว่างจุด 2 จุด ข้อเสีย ก็คือ ไม่สามารถเปลี่ยนจุดรับข้อมูลไปที่ต่างๆได้ จึงขาดความคล่องตัว

โมเด็มแบบใช้กับสายโทรศัพท์ จะรับส่งข้อมูลด้วยความเร็วตั้งแต่ 300 บิตต่อวินาทีจนถึง 9,600 บิตต่อวินาที ผ่านเครือข่ายโทรศัพท์ที่เราใช้อยู่ เราสามารถรับส่งข้อมูลไปยังที่ต่างๆได้ตามต้องการ ไม่กำหนดจุดรับข้อมูลตายตัวเหมือนอย่างโมเด็มสายตรง เครื่องคอมพิวเตอร์ส่วนบุคคลและบริการรับส่งข้อมูลต่างๆ จะใช้โมเด็มแบบนี้ในการทำงานเกือบทั้งหมด การรับส่งข้อมูลจะเป็นการรับส่งทีละหนึ่งตัวอักษรไม่ส่งเป็นกลุ่มเรียกว่า รับส่งแบบ Asynchronous ปกติแล้วเรารับส่งข้อมูลผ่านสายโทรศัพท์ด้วยความเร็ว 1,200 ถึง 2,400 บิตต่อวินาทีเท่านั้น

ความเร็วสูงสุดที่เชื่อถือได้คือ 9,600 บิตต่อวินาที ถ้าใช้โมเด็มความเร็วสูงกว่านี้รับส่งข้อมูลผ่านสายโทรศัพท์อาจทำได้ภายในบางจุด แต่ส่วนมากจะมีความผิดพลาดสูง จึงถือความเร็ว 9,600 บิตต่อวินาที เป็นความเร็วสูงสุดสำหรับส่งข้อมูลผ่านสายโทรศัพท์ในปัจจุบัน

ข้อดีของการใช้โมเด็มแบบนี้ ก็คือ มีความคล่องตัวสูง สามารถรับส่งข้อมูลไปยังที่ต่างๆได้ไม่จำกัดและไม่จำเป็นต้องจัดหาวงจรสายตรงมาเป็นพิเศษเพื่อส่งข้อมูล ข้อเสียตรงที่ว่าความเร็วในการส่งข้อมูลต่ำกว่าโมเด็มแบบสายตรง ข้อเสียอีกอันหนึ่งก็คือ ถ้ารับส่งข้อมูลเป็นจำนวนมากไปยังจุดปลายทางจุดเดียวในระยะทางไกล ค่าโทรศัพท์อาจจะแพงกว่าการเช่าวงจรสายตรงมาใช้ก็ได้ อันนี้ขึ้นกับเวลาที่ใช้ส่งข้อมูลเป็นหลัก เราต้องพิจารณาค่าใช้จ่ายเปรียบเทียบเอาเอง



รูปที่ 2.1.9 การรับส่งข้อมูลผ่านสายโทรศัพท์ เราสามารถรับส่งข้อมูลกับจุดต่างๆได้หลายแห่ง โดยหมุนเบอร์ปลายทางผ่านชุมสาย

โมเด็มที่มีขายในท้องตลาดปัจจุบัน บางรุ่นอาจทำงานได้เฉพาะกับสายตรง หรือบางรุ่นอาจทำงานได้กับสายโทรศัพท์เท่านั้น โมเด็มที่ใช้กับสายตรงได้เพียงอย่างเดียว จะนำมาต่อใช้กับสายโทรศัพท์ไม่ได้ และโมเด็มที่ใช้กับสายโทรศัพท์ได้อย่างเดียวจะนำมาใช้งานต่อกับสายตรงไม่ได้เช่นกัน ยกเว้น โมเด็มบางชนิดซึ่งเลือกการทำงานในตัวได้ว่าจะต่อกับสายตรงหรือสายโทรศัพท์ โมเด็มที่ใช้งานได้ทั้งสองอย่างก็จะมีราคาแพงกว่าโมเด็มที่ใช้กับสายโทรศัพท์เพียงอย่างเดียว ถ้าการใช้งานของเราไม่ใช่การต่อคอมพิวเตอร์ส่วนบุคคลเข้ากับเมนเฟรม หรือมินิคอมพิวเตอร์แล้วละก็ การใช้โมเด็มแบบต่อสายโทรศัพท์ได้อย่างเดียว ก็นับว่าเพียงพอแล้วสำหรับรับส่งข้อมูลทั่วไป

<u>โทรศัพท์แบบกดปุ่มและแบบหมุน</u>

โมเด็มทั่วไปมักจะหมุนโทรศัพท์ได้ทั้งแบบกดปุ่มและแบบมือหมุน ที่เราต้องสั่งโมเด็มให้หมุนโทรศัพท์ให้ถูกต้อง ก็เนื่องมาจาก ถ้าโทรศัพท์ที่เราใช้เป็นแบบมือหมุน แต่สั่งให้โมเด็มหมุนโทรศัพท์แบบกดปุ่ม ชุมสายจะไม่รับสัญญาณการหมุนโทรศัพท์ของโมเด็ม ทำให้ติดต่อไปยังปลายทางไม่ได้ และทำนองกลับกันถ้าเราใช้โทรศัพท์แบบกดปุ่มแต่ส่งสัญญาณให้ชุมสายโทรศัพท์แบบมือหมุน ก็จะต่อไปหาปลายทางไม่ได้เช่นกัน ดังนั้นเราจึง

ต้องสั่งให้โมเด็มหมุนโทรศัพท์ให้ตรงตามชนิดของโทรศัพท์ที่เรามีอยู่ วิธีการง่ายๆที่จะรู้ว่าโทรศัพท์ของเราเป็นแบบไหนก็คือ ฟังเสียงขณะที่เราหมุนโทรศัพท์ไปหาเบอร์ปลายทางตามปรกติ ถ้ามีเสียงลักษณะสูงๆต่ำๆละก็ แสดงว่าโทรศัพท์ของเราเป็นแบบกดปุ่ม (Tone Dialing) แต่ถ้าได้ยินเสียงลักษณะเป็นช่วงๆ แสดงว่าโทรศัพท์ของเราเป็นแบบมือหมุน (Pulse Dialing)

2.1.4 Communication Parameter

จากการที่โมเด็มต่อกับคอมพิวเตอร์ผ่านทาง RS-232C และส่งข้อมูลไปให้เครื่องคอมพิวเตอร์อีกเครื่องหนึ่ง ดังนั้นองค์ประกอบในการรับส่งข้อมูล (Communication Parameter) จึงต้องตรงกันตลอดเส้นทางที่รับส่ง องค์ประกอบดังกล่าวคือ ความเร็ว (Speed) จำนวนบิตของข้อมูล (Data Bit) จำนวนบิตเริ่มและบิตจบ (Start,Stop Bit) การตรวจสอบพาริตี้บิต (Parity Bit) และการเลือกใช้ Echo (Duplex) ในการรับส่ง

ความเร็ว (Speed) คือ อัตราการรับส่งข้อมูลเป็นจำนวนบิตต่อวินาที เราต้องใช้ความเร็วเดียวกันตลอดเส้นทางการรับส่งข้อมูล เช่นเดียวกับองค์ประกอบอื่นๆ ทั้งเครื่องคอมพิวเตอร์ที่ส่งข้อมูลออกไปโมเด็มที่ด้านส่ง โมเด็มที่ด้านรับและคอมพิวเตอร์เครื่องรับจะต้องมีความเร็วในการรับส่งข้อมูลเท่ากัน ความเร็วที่ใช้กันทั่วไปมีตั้งแต่ 300 บิตต่อวินาที 1,200 , 2,400 , 4,800 ไปจนถึง 9,600 บิตต่อวินาที

จำนวนบิตของข้อมูล (Data Bit) นั้นคือ ในหนึ่งตัวอักษรจะใช้จำนวนข้อมูลกี่บิตในการส่ง ปกติจะเลือกได้ 2 แบบคือ 7 บิตต่อหนึ่งตัวอักษร หรือ 8 บิตต่อหนึ่งตัวอักษร การรับส่งข้อมูลภาษาไทย เราจำเป็นต้องใช้แบบ 8 บิตต่อตัวอักษรเท่านั้น เนื่องจากภาษาไทยเราใช้รหัสครบทั้ง 8 บิต ถ้าหากส่งไปในแบบ 7 บิตต่อหนึ่งตัวอักษร รหัสภาษาไทยจะถูกตัดบิตที่ 8 ทิ้งไป กลายเป็นตัวภาษาอังกฤษซึ่งใช้งานไม่ได้ จำนวนบิตเริ่มและบิตจบนั้นกำหนดไว้ให้ตัวรับและตัวส่ง แยกออกว่าข้อมูลจะเริ่มต้นเมื่อใดและจบลงเมื่อใด

บิตเริ่มต้น (Start Bit) มักจะใช้ 1 บิตเสมอ ส่วนบิตจบข้อมูล (Stop Bit) จะมี 2 แบบคือ 1 บิต หรือ 2 บิต การใช้งานทั่วไปมักจะใช้ 1 บิตจบเช่นกัน

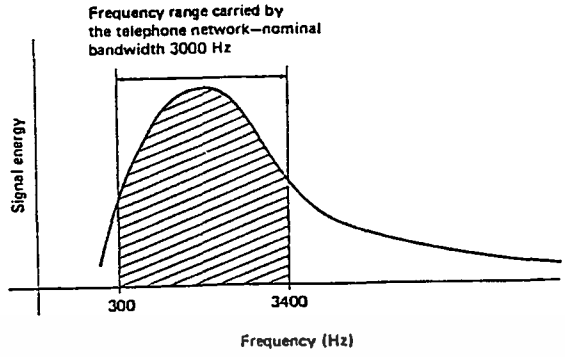
การตรวจสอบพาริตี้บิต (Parity Bit) เป็นการตรวจสอบความถูกต้องของการรับส่งข้อมูลที่ส่งมามีจำนวนข้อมูลที่เป็น "1" เป็นจำนวนคู่ (Even) หรือจำนวนคี่ (Odd) หรือไม่

ต้องตรวจสอบ (None) เช่น ถ้าส่งข้อมูล 11000001 ไปให้ผู้รับและมีการตรวจสอบพาริตี้แบบจำนวนคู่ พาริตี้บิตในกรณีนี้จะมีค่าเป็น "1" เพื่อให้จำนวน "1" ทั้งหมดมีจำนวนเป็นเลขคู่ (Even) ถ้ากำหนดการตรวจสอบพาริตี้เป็นจำนวนคี่ พาริตี้ก็จะเป็น "0" เพื่อให้จำนวน "1" ของข้อมูลทั้งหมดเป็นจำนวนคี่ (Odd) และถ้าไม่มีการตรวจสอบเลย เครื่องส่งก็จะส่งบิตจบ (Stop Bit) ปิดท้ายข้อมูลทันที ไม่มีการส่งพาริตี้บิตไปให้ผู้รับ ส่วนมากถ้ารับส่งข้อมูลแบบ 7 บิต เรามักจะตั้งการตรวจสอบพาริตี้บิตเอาไว้ แต่ถ้ารับส่งข้อมูลแบบ 8 บิต ก็จะไม่มีการตรวจสอบพาริตี้บิต สำหรับการเลือกใช้ Echo ในการรับส่งก็เป็นการเลือกให้สอดคล้องกับ Full Duplex หรือ Half Duplex นั่นเอง ถ้าเป็นการรับส่งแบบ Full Duplex ก็ต้องเลือกใช้ Echo Off และถ้าเป็นแบบ Half Duplex เราก็เลือกใช้ Echo On เมื่อองค์ประกอบนี้ตรงกันการรับส่งข้อมูลก็จะทำได้ถูกต้อง ถ้าส่วนใดส่วนหนึ่งไม่ว่าจะเป็นคอมพิวเตอร์ด้านส่ง โมเด็มด้านส่ง โมเด็มด้านรับหรือคอมพิวเตอร์ด้านรับ เพียงส่วนเดียวใช้องค์ประกอบในการรับส่งข้อมูลผิดไป การรับส่งข้อมูลจะผิดพลาดได้ ทั้งผู้รับและผู้ส่งจึงต้องตกลงกันให้แน่นอนว่า จะใช้องค์ประกอบในการรับส่งข้อมูลแต่ละตัวอย่างไร

2.1.5 <u>การใช้สายโทรศัพท์ส่งข้อมูล</u>

ความสะดวกเมื่อเราใช้สายโทรศัพท์ที่ส่งข้อมูลที่เห็นชัด คือ สามารถส่งข้อมูลไปยังที่ต่างๆได้ตามต้องการ โดยแทบจะไม่ต้องลงทุนเพิ่มเติมเลย นอกจากซื้อโมเด็มมาใช้เท่านั้น อุปกรณ์อื่นๆเป็นสิ่งที่มีอยู่แล้ว เพียงแต่นำมาใช้ให้เป็นประโยชน์มากขึ้น ค่าใช้จ่ายในการรับส่งข้อมูลต่ำกว่าวิธีอื่นๆ ในกรณีที่ส่งข้อมูลไม่มากนัก แต่ขีดจำกัดของระบบโทรศัพท์เป็นสิ่งที่หลีกเลี่ยงไม่ได้ ซึ่งควรจะรู้เอาไว้ประกอบการใช้งาน

ระบบการรับฟังเสียงของคนเราสามารถรับรู้เสียงความถี่ได้ตั้งแต่ 20Hz. จนถึง 20,000Hz. โดยประมาณ ความถี่สูงกว่าและต่ำกว่านี้ เราจะไม่ได้ยินเสียง เมื่อมีการคิดค้นระบบโทรศัพท์ขึ้นมา เทคโนโลยียังไม่สูงมากพอที่จะสร้างระบบโทรศัพท์ให้รับส่งเสียงได้เทียบเท่ากับที่หูคนได้ยิน การออกแบบจึงพิจารณาด้านเทคนิคและราคาที่พอยอมรับได้ในสมัยนั้น ผลที่ออกมาก็คือ ระบบโทรศัพท์สามารถรับส่งสัญญาณเสียงได้ตั้งแต่ความถี่ 300Hz. จนถึงประมาณ 3400Hz. ซึ่งความถี่ช่วงดังกล่าวนี้มากพอสำหรับการพูดคุยของคนทั่วไป ด้วยคุณภาพของเสียงดีพอควร แต่มันไม่มากพอให้เราส่งข้อมูลคอมพิวเตอร์ได้อย่างสะดวกผ่านสายโทรศัพท์ เพราะในการรับส่งข้อมูลทั่วไปสายส่งจะมี Bandwidth ยิ่งสูงเท่าใด ก็จะสามารถรับส่งข้อมูลได้เร็วมากขึ้นเท่านั้น



รูปที่ 2.1.10 สายโทรศัพท์รับส่งสัญญาณได้ในช่วง 300Hz. ถึง 3400Hz. เท่านั้น

ขีดจำกัดต่างๆของเครือข่ายโทรศัพท์

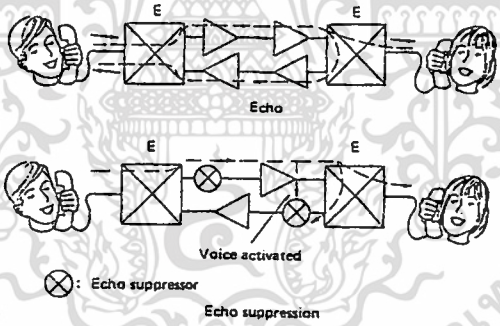
ช่วงความถี่ 300 Hz. ถึง 3,400 Hz. ที่สายโทรศัพท์สามารถรับส่งสัญญาณได้ดี นับว่าเป็นสายส่งที่มี Bandwidth แคบมาก เมื่อเรานำมาใช้รับส่งข้อมูล เพราะว่าตามธรรมดาแล้ว ความเร็วสูงสุดของการรับส่งข้อมูลจะมีค่าต่ำกว่าความถี่สูงสุดของสายส่งเสมอ ถ้าหากไม่ใช้เทคนิคพิเศษเข้าช่วย สายโทรศัพท์ก็จะรับส่งข้อมูลได้ไม่เกิน 3,000 บิตต่อวินาทีเท่านั้น ซึ่งเป็นตัวเลขที่ไม่น่าพอใจเท่าไร แรกเริ่มของการส่งข้อมูลผ่านสายโทรศัพท์ มีความเร็วเพียง 3,000 บิตต่อวินาที และพัฒนาจนรับส่งได้ 1,200 ถึง 2,400 บิตต่อวินาทีตามลำดับ เมื่อเทคโนโลยีก้าวหน้าขึ้นจากการที่ช่วงความถี่ของสัญญาณ(Bandwidth)จำกัดมากในสายโทรศัพท์ ในการรับส่งข้อมูลความเร็วสูงกว่า 2,400 บิตต่อวินาที ผู้ผลิตโมเด็มจำเป็นต้องใช้เทคนิคการเข้ารหัส และการผสมสัญญาณที่ซับซ้อนมากเพื่อแก้ปัญหาดังกล่าว ดังนั้นการรับส่งข้อมูลความเร็วสูงจึงถูกรบกวนง่าย และมีอัตราการผิดผลาดในการรับส่งข้อมูลสูงขึ้นเป็นเงาตามตัว

วงจรสายตรงนับว่าได้เปรียบมากในข้อนี้ เนื่องจากคุณภาพของสายส่งและวงจรมีคุณภาพดีกว่า ทำให้รับส่งข้อมูลความเร็วสูงได้อย่างไม่มีปัญหาด้านขีดจำกัดของช่วงความถี่เหมือนที่เกิดขึ้นที่สายโทรศัพท์

ขีดจำกัดลำดับที่สองของการส่งข้อมูลผ่านสายโทรศัพท์ก็คือ วงจรกำจัดเสียงสะท้อน (Echosuppressor) ของระบบโทรศัพท์ เนื่องจากการส่งสัญญาณไฟฟ้าผ่านสายส่งเป็นระยะทางไกลๆ นั้น สัญญาณที่ส่งออกไปจะมีส่วนหนึ่งสะท้อนกลับมายังผู้ส่ง ซึ่งเรา

เรียกว่าเกิด Echo ขึ้นปรากฏการณ์อันนี้เกิดขึ้นกับคลื่นทุกชนิด ไม่จำกัดว่าจะต้องเป็นสัญญาณไฟฟ้าเท่านั้น เมื่อมันเกิดขึ้นในสายโทรศัพท์เราก็จะได้ยินเสียงตังเองสะท้อนกลับมาในลักษณะของเสียงก้อง คล้ายกับเวลาที่เราพูดอยู่ในถ้ำนั่นเองการเกิดเสียงก้องนี้นอกจากจะทำให้การรับโทรศัพท์มีคุณภาพต่ำลงแล้วมันยังทำให้การสนทนาเป็นไปอย่างลำบากอีกด้วย

ระบบโทรศัพท์จึงใช้ วงจรกำจัดเสียงสะท้อนเข้ามาช่วยแก้ปัญหานี้ การทำงานของมันจะยอมให้สัญญาณจากผู้พูดเดินทางไปถึงผู้รับได้ทางเดียว สัญญาณที่สะท้อนกลับมาจะถูกกั้นเอาไว้ไม่ให้กลับมาหาผู้พูดได้ เมื่อผู้ฟังพูดสวนกลับไปวงจรกำจัดเสียงสะท้อนก็จะสลับทิศทางไปในทางตรงข้าม เพื่อให้สัญญาณจากผู้พูดถูกกั้นไม่ให้สะท้อนกลับมาหาตัวเองอีกเช่นเดียวกัน วิธีการนี้ทำให้เสียงสะท้อนถูกกำจัดออกไปได้ วงจรกำจัดเสียงสะท้อนจะรู้ว่าใครเป็นผู้พูดโดยดูจากความแรงของสัญญาณ ใช้กฎเกณฑ์ที่ว่าด้านที่พูดจะมีสัญญาณแรงกว่าด้านรับเสมอ และการสลับทิศทางของวงจรกำจัดเสียงสะท้อน ทำด้วยความเร็วสูงมากจนเราไม่รู้สึกผิดปกติเมื่อพูดและฟังตามธรรมดา ขณะที่เราสลับกันพูดโทรศัพท์ในการสนทนาทั่วไป วงจรกำจัดเสียงสะท้อนจะสลับทิศทางตลอดเวลา เพื่อป้องกันเสียงสะท้อนไม่ให้เกิดขึ้น



รูปที่ 2.1.11 วงจรป้องกันเสียงสะท้อน (Echo suppressor) จะตัดเสียงผู้พูดไม่ให้สะท้อนกลับมา

<u>วิธีป้องกันเสียงสะท้อนของโทรศัพท์</u>

เมื่อเราส่งข้อมูลผ่านสายโทรศัพท์วงจรกำจัดเสียงสะท้อนจะทำให้การรับส่งข้อมูลมีปัญหามากเนื่องจากการสลับทิศทางของวงจรกำจัดเสียงสะท้อนจะทำให้สัญญาณขาดหายไปในช่วงนั้น ถึงแม้ว่าจะเป็นช่วงเวลาสั้นๆ แต่ก็อาจจะทำให้ข้อมูลบางส่วนหายไปได้ นอกจากนี้การสลับทิศทางของวงจรกำจัดเสียงสะท้อนยังทำให้เสียเวลาในการส่งข้อมูลอีกด้วย เพราะมันต้องใช้เวลาช่วงหนึ่งในการสลับทิศทางเสมอสิ่งเหล่านี้มีผลมากเมื่อเราส่งข้อมูลผ่านสายโทรศัพท์ด้วยความเร็วสูง

โมเด็มในปัจจุบันจะหยุดการทำงานของวงจรกำจัดเสียงสะท้อนได้ โดยส่งคลื่นเสียงความถี่ประมาณ 2,100 Hz. ไปยังชุมสายโทรศัพท์นานประมาณครึ่งวินาที วงจรกำจัดเสียงสะท้อนก็จะหยุดสลับทิศทาง ทำให้การรับส่งข้อมูลเป็นไปได้อย่างไม่มีปัญหา โมเด็มจะต้องหยุดการทำงานของวงจรกำจัดเสียงสะท้อนเสมอ เพราะว่าการรับส่งข้อมูลผ่านสายโทรศัพท์มักจะเป็นแบบส่งสวนทางกันได้ (Full Duplex) โดยใช้เทคนิคแบ่งความถี่เข้าช่วย ซึ่งถ้าหากว่าวงจรกำจัดเสียงสะท้อนยังคงทำงานอยู่ ข้อมูลบางส่วนอาจจะผิดพลาดหรือขาดหายไปได้ในการรับส่งแบบ Half Duplex เราก็ยังจำเป็นจะต้องหยุดการทำงานของวงจรกำจัดเสียงสะท้อนอยู่ดี เนื่องจากว่าสัญญาณของข้อมูลส่งไปตามสายนั้นมักจะทำให้วงจรกำจัดเสียงสะท้อนหลงทางอยู่เสมอ มีผลทำให้ข้อมูลเดินทางไปไม่ถึงผู้รับ เพราะถูกกั้นสัญญาณเอาไว้ วงจรกำจัดเสียงสะท้อนจึงเป็นปัญหาที่โมเด็มจะต้องจัดการให้เรียบร้อยก่อนส่งข้อมูลผ่านสายโทรศัพท์เสมอ

สัญญาณรบกวนในสายโทรศัพท์เป็นปัญหาอีกอันหนึ่งในการรับส่งข้อมูล ปกติสัญญาณรบกวนจะมีอยู่ระดับหนึ่งตลอดเวลาในสาย ซึ่งอาจจะเกิดจากคลื่นแม่เหล็กไฟฟ้าจากที่อื่นถูกเหนี่ยวนำเข้ามาในสาย หรือเกิดจากอุปกรณ์รับส่งของชุมสายโทรศัพท์เองก็ได้ เมื่อสัญญาณรบกวนมีขนาดไม่มากนักเทียบกับสัญญาณข้อมูลที่เราใช้รับส่ง เราก็ยังคงแยกออกว่าส่วนไหนคือข้อมูลและส่วนไหนคือสัญญาณรบกวนแต่ในระยะทางไกลๆ สัญญาณข้อมูลที่ส่งออกไปจะมีกำลังอ่อนลงตามระยะทาง ทำให้ความแตกต่างระหว่างสัญญาณรับส่งและสัญญาณรบกวนน้อยลง การรับข้อมูลจะทำได้ลำบากมากยิ่งขึ้น

โมเด็มที่รับส่งข้อมูลด้วยความเร็วไม่เกิน 2,400 บิต่อวินาที มักไม่ค่อยมีปัญหาเรื่องสัญญาณรบกวน เนื่องจากว่าความเร็วขนาดนี้การผสมสัญญาณในการรับส่งยังไม่ซับซ้อนมากนัก รูปคลื่นที่ส่งออกไปมีความแตกต่างกันชัดเจน เมื่อถูกรบกวนจากสัญญาณอื่นๆ ก็ยังสามารถแยกออกได้ง่ายว่าข้อมูลที่ส่งมาเป็นอย่างไร ปัญหาของสัญญาณรบกวนในสายจะมีผลกับโมเด็มความเร็วสูง เช่น 4,800 หรือ 9,600 บิตต่อวินาทีค่อนข้างมาก เพราะที่ความเร็วสูงจะต้องใช้เทคนิคการผสมสัญญาณซับซ้อนมาก เพื่อส่งข้อมูลไปในสายโทรศัพท์ซึ่งมี Bandwidth 3,000 Hz. เท่านั้น รูปคลื่นของสัญญาณที่ส่งออกไปจึงต้องมีความแม่นยำสูง ผู้รับถึงจะรับข้อมูลได้ถูกต้อง สัญญาณรบกวนจะทำให้รูปคลื่นผิดไปจากเดิมมีผลถึงการรับข้อมูลพลาด โมเด็มความเร็วสูงจำเป็นต้องมีฮาร์ดแวร์พิเศษ มาทำหน้าที่ตัดสัญญาณรบกวนในสายออกโดยเฉพาะ

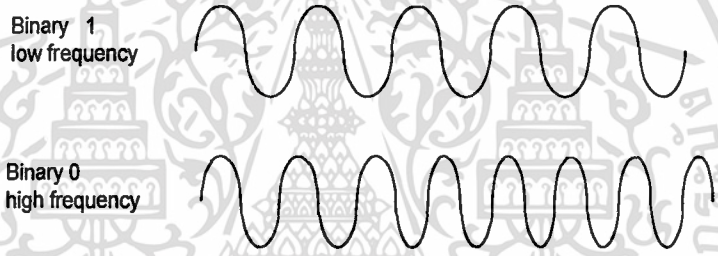
2.1.6 ประเภทของโมเด็ม

เราสามารถแบ่งโมเด็มโดยใช้ความเร็วในการส่งข้อมูลเป็นหลักได้ 3 กลุ่ม คือ

1. โมเด็มความเร็วต่ำ
2. โมเด็มความเร็วปานกลาง
3. โมเด็มความเร็วสูง

โมเด็มความเร็วต่ำ

โมเด็มความเร็วต่ำจะส่งข้อมูลที่ความเร็ว 300 บิตต่อวินาทีจนถึง 600 บิตต่อวินาทีใช้การผสมสัญญาณแบบเปลี่ยนความถี่หรือ FSK (Frequency Shift Keying) ทางด้านรับและด้านส่งจะใช้ความถี่สองความถี่แทนค่าข้อมูล "0" และ "1" ไม่ซ้ำกัน



รูปที่ 2.1.12 สัญญาณ 0 และ 1 จะใช้ความถี่คนละความถี่ในการผสมสัญญาณแบบ FSK

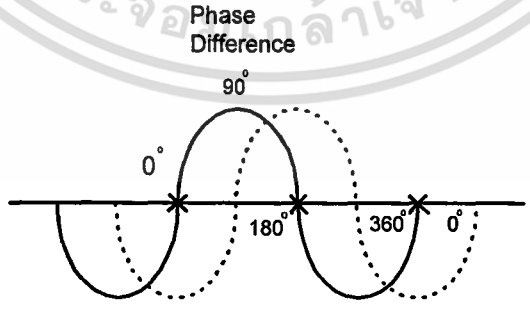
รวมความถี่ที่ใช้ทั้งหมด 4 ความถี่ โดยเมื่อเชื่อมต่อกันแล้ว ฝ่ายเรียกหรือผู้เรียก (Originate) จะใช้ความถี่ของผู้เรียกในการส่งข้อมูล ฝ่ายรับ (Answer) ก็จะใช้ความถี่อีกคู่หนึ่งเพื่อไม่ให้ชนกันสำหรับส่งข้อมูล

เนื่องจากความเร็วในการส่งข้อมูลต่ำ ถึงแม้ว่าจะรับส่งสวนทางกันได้แบบ Full Duplex ก็ตามความเร็ว 300 ถึง 600 บิตต่อวินาทีนี้ จะรับส่งข้อมูลได้เพียง 30 ถึง 80 ตัวอักษรต่อวินาทีเท่านั้นเพราะปกติหนึ่งตัวอักษรจะเท่ากับ 10 บิต โดยประมาณ คือเป็นตัวข้อมูล 8 บิต บิตเริ่มต้นข้อมูลหนึ่งบิต และบิตปิดท้ายข้อมูลอีกหนึ่งบิต รวมทั้งหมดจึงเท่ากับข้อมูล 10 บิต ถ้าหากต้องการส่งข้อมูลขนาด 36 กิโลไบต์ จะต้องใช้เวลาส่งข้อมูลนานถึง 20 นาทีที่ความเร็ว 300 บิตต่อวินาที หรือใช้เวลานาน 10 นาทีที่ความเร็ว 600 บิตต่อวินาที ซึ่งนอกจากจะไม่ทันใจผู้ใช้แล้ว ยังเสียค่าใช้จ่ายในการรับส่งข้อมูลมากอีกด้วย โดยเฉพาะอย่าง

ยิ่งในการส่งข้อมูลทางไกลซึ่งคิดค่าส่งเป็นนาที จึงทำให้ปัจจุบันโมเด็มความเร็วต่ำจึงไม่ค่อยได้รับความนิยมแล้ว

โมเด็มความเร็วปานกลาง

โมเด็มความเร็วปานกลางมีความเร็วในการส่งข้อมูลประมาณ 1,200 ถึง 2,400 บิตต่อวินาที ซึ่งจัดเป็นโมเด็มยอดนิยมในปัจจุบัน เครื่องคอมพิวเตอร์ส่วนบุคคลมักจะใช้โมเด็มความเร็วปานกลางนี้ส่งข้อมูลเกือบทั้งหมดของโมเด็มที่มีใช้กันอยู่ โมเด็มความเร็วปานกลางนี้มีฟังก์ชั่นการทำงานพิเศษมากมาย ช่วยให้ผู้ใช้สะดวกมากยิ่งขึ้นตั้งแต่การหมุนโทรศัพท์อัตโนมัติ การจดจำเบอร์โทรศัพท์ไปจนถึงการปรับจำนวนบิตที่รับส่งข้อมูลอัตโนมัติ ฯลฯ โมเด็มความเร็วปานกลางนี้ ใช้เทคนิคการผสมสัญญาณก้าวหน้ากว่าโมเด็มความเร็วต่ำมาก คือแทนที่จะใช้เปลี่ยนแปลงความถี่ไปมาแสดงข้อมูลของ "0" และ "1" โมเด็มความเร็วปานกลางจะผสมสัญญาณโดยเปลี่ยนแปลงมุมของช่วงคลื่น (Phase) และขนาดของสัญญาณ (Amplitude) ไปพร้อมๆกัน ทำให้ในหนึ่งลูกคลื่นของสัญญาณส่งแทนค่าข้อมูลได้มากกว่าเดิม เช่น โมเด็มความเร็ว 1,200 บิตต่อวินาที หนึ่งลูกคลื่นของสัญญาณจะแทนค่าข้อมูลเท่ากับสองบิต และในโมเด็มความเร็ว 2,400 บิตต่อวินาทีจะส่งข้อมูลสี่บิตผสมเข้าไปในสัญญาณส่งเพียงหนึ่งลูกคลื่นเท่านั้น วิธีนี้ทำให้ข้อมูลส่งได้เร็วขึ้นโดยไม่ต้องเพิ่มความถี่ในการส่ง จากความเร็ว 1,200 หรือ 2,400 บิตต่อวินาที นี้ การส่งข้อมูลขนาด 36 กิโลไบต์ จะใช้เวลาเพียง 5 นาทีที่ควรเร็ว 1,200 บิตต่อวินาที หรือใช้เวลาสองนาทีครึ่งที่ความเร็ว 2,400 บิตต่อวินาที ถึงจะไม่เร็วจัดอย่างที่เราคาดหวังเอาไว้ แต่ความเร็วขนาดนี้ก็นับว่าเพียงพอกับการรับส่งข้อมูลทั่วไป ซึ่งโมเด็มความเร็วปานกลางนี้ทำการรับส่งข้อมูลได้ทั้งแบบ Full duplex และ Half duplex



รูปที่ 2.1.13 แสดงการเปลี่ยนแปลงมุมของช่วงคลื่นหรือ Phase

โมเด็มความเร็วสูง

โมเด็มความเร็วสูงมีความเร็วในการรับส่งข้อมูลตั้งแต่ 4,800 ถึง 9,600 บิตต่อวินาที หรือสูงกว่านั้น การที่โมเด็มสามารถรับส่งข้อมูลได้เร็วมากขนาดนี้ผ่านสายโทรศัพท์ได้ก็เนื่องมาจากใช้ฮารด์แวร์พิเศษเข้าช่วย และมีการผสมสัญญาณที่ซับซ้อนกว่าโมเด็มความเร็วปานกลางมาก ในหนึ่งลูกคลื่นของสัญญาณที่ส่งออกไปอาจมีจำนวนบิตข้อมูลผสมอยู่ถึง 5 บิตก็ได้ และยังสามารถรับส่งในแบบ Full Duplex ได้อีกด้วย ซึ่งเมื่อเทียบกับความสามารถของสายโทรศัพท์ที่ส่งสัญญาณได้สูงสุด 3,400 Hz. แล้วก็นับว่าโมเด็มความเร็วสูงเจาะทะลุขีดจำกัดต่างๆ ได้อย่างเหลือเชื่อ ถ้าเทียบเวลาที่ใช้ส่งข้อมูลขนาด 36 กิโลไบต์เท่าเดิมแล้ว เราจะใช้เวลาส่งข้อมูลเพียงแค่ไม่ถึง 40 วินาทีเท่านั้นที่ความเร็ว 9,600 บิตต่อวินาที ซึ่งจัดว่าเร็วมาก ฟังก์ชั่นการทำงานต่างๆ ของโมเด็มความเร็วสูงก็มีให้ใช้ใกล้เคียงกับโมเด็มความเร็วปานกลางและสามารถลดความเร็วในการรับส่งข้อมูลไปกลายเป็นโมเด็มความเร็วปานกลางก็ได้

โมเด็มความเร็วสูงบางแบบอาจจะรับส่งข้อมูลได้เร็วกว่า 9,600 บิตต่อวินาทีผ่านสายโทรศัพท์แต่ในปัจจุบันเราถือว่าความเร็ว 9,600 บิตต่อวินาทีนี้เป็นความเร็วสูงสุดที่ยอมรับกันโดยมีมาตรฐานรองรับอยู่ ที่ความเร็วสูงกว่านี้มักจะต้องใช้โมเด็มยี่ห้อเดียวกันสำหรับผู้รับและผู้ส่ง เพราะส่วนมากมักจะรับส่งข้อมูลกันไม่ได้ เนื่องจากใช้เทคนิคแตกต่างกันที่ความเร็วสูงกว่า 9,600 บิตต่อวินาที โมเด็มความเร็วสูงยังมีราคาแพง เนื่องจากใช้อุปกรณ์และเทคนิคชั้นสูงส่วนมากโมเด็มความเร็วสูงเหมาะสำหรับงานที่ส่งข้อมูลเป็นจำนวนมากผ่านระยะทางไกลๆ

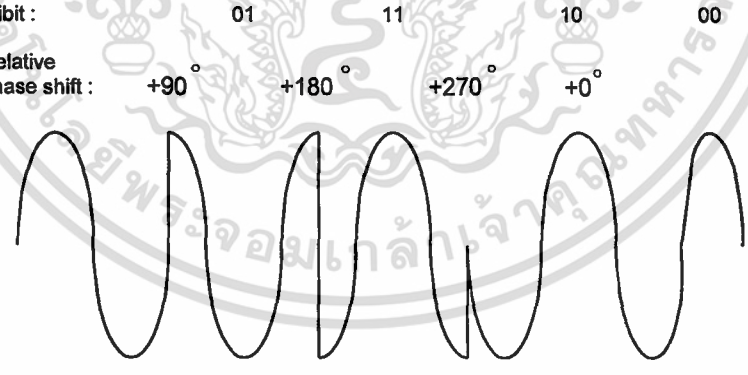
2.1.7 มาตรฐานของโมเด็ม

โมเด็ม จำเป็นต้องมีมาตรฐานเช่นเดียวกับอุปกรณ์อื่นๆ มาตรฐานของโมเด็มจะแบ่งออกเป็นสองส่วน คือ มาตรฐานในส่วนของฮาร์ดแวร์ที่โมเด็มใช้ และอีกส่วนหนึ่งคือมาตรฐานในส่วนของซอฟต์แวร์ที่ควบคุมการทำงานของโมเด็มหรือคำสั่งของโมเด็ม โดยในส่วนนี้จะกล่าวถึงมาตรฐานทางด้านฮาร์ดแวร์ของโมเด็มก่อนโดยมาตรฐานนี้กำหนดขึ้นจากองค์การมาตรฐานสากลหรือ CCITT (International Telephone and Telegraph Consultative Committee) ซึ่งมาตรฐานนี้จะครอบคลุมเกี่ยวกับความเร็วในการรับส่งข้อมูล ความถี่ที่ใช้และเทคนิคการผสมสัญญาณในสาย ฯลฯ ดังนั้นผู้ผลิตรายใหญ่จึงผลิตตามมาตรฐานของ CCITT ทำให้โมเด็มแต่ละยี่ห้อสามารถรับส่งข้อมูลกันได้

Bit rate กับ Baud rate

ความเร็วในการรับส่งข้อมูลของโมเด็มนั้นกำหนดเป็นบิตต่อวินาทีหรือ Bit rate ซึ่งแตกต่างจากอัตราการเปลี่ยนแปลงของสัญญาณไฟฟ้าในสายส่งหรือที่เรียกกันว่า Baud rate

ในสมัยก่อนการรับส่งข้อมูลใช้เทคนิคการผสมสัญญาณแบบง่ายๆ เช่นการเปลี่ยนแปลงความถี่ตามข้อมูล "0" และ "1" ที่ได้รับ อัตราการส่งข้อมูลและอัตราการเปลี่ยนแปลงของสัญญาณในสายส่งมีค่าเท่ากัน เราจึงถือว่าอัตราการเปลี่ยนแปลงของสัญญาณในสายส่ง (Baud rate) คืออัตราการส่งข้อมูลนั่นเอง ต่อมาเทคนิคการผสมสัญญาณซับซ้อนมากขึ้น ทำให้เราสามารถส่งข้อมูลด้วยความเร็วสูงโดยอัตราการเปลี่ยนแปลงในสายยังคงเท่าเดิม ดังนั้นเมื่อเราพูดว่า โมเด็มรับส่งข้อมูลที่ความเร็ว 1,200 Baud เราจะไม่ทราบเลยว่าโมเด็มนั้นรับส่งข้อมูลได้กี่บิตต่อวินาทีเนื่องจากว่าถ้าโมเด็มผสมสัญญาณ 1 บิตต่อหนึ่งลูกคลื่น โมเด็มนั้นจะรับส่งข้อมูลที่ความเร็ว 1,200 บิตต่อวินาทีหรือถ้าโมเด็มผสมสัญญาณ 2 บิตต่อหนึ่งลูกคลื่นที่เปลี่ยนแปลงในสายส่ง โมเด็มจะรับส่งข้อมูลได้เร็วถึง 2,400 บิตต่อวินาที โดยยังคงมีอัตราการเปลี่ยนแปลงในสายส่งเท่ากับ 1,200 Baud เหมือนเดิม เพราะฉะนั้นเราจึงเลิกใช้คำว่า Baud rate สำหรับบอกความเร็วการรับส่งข้อมูลของโมเด็ม มาใช้คำว่า Bit rate หรืออัตราการส่งข้อมูลเป็นบิตต่อวินาทีแทน ซึ่งสื่อความหมายได้เข้าใจตรงกันมากกว่า

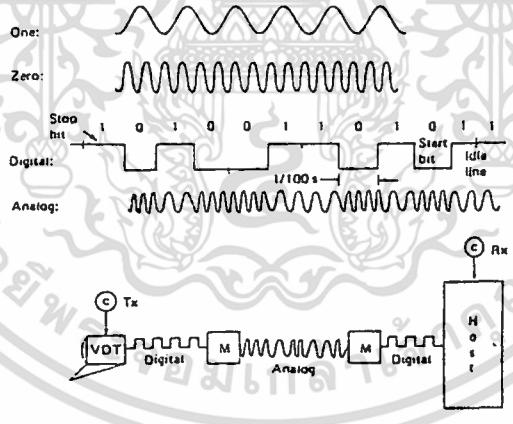


รูป 2.1.14 การผสมสัญญาณ 2 บิตต่อหนึ่งลูกคลื่นทำให้ Bit rate มีค่าสูงกว่า Baud rate

การผสมสัญญาณแบบ FSK และ PSK

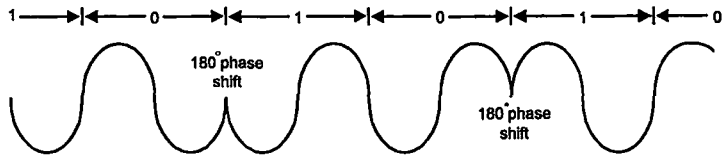
นอกจากมาตรฐานจะกำหนดความเร็วในการรับส่งข้อมูลให้ตรงกันแล้ว ยังกำหนดความถี่ของคลื่นพาหะที่ใช้ว่า ผู้ส่งและผู้รับใช้ความถี่เท่าไร รวมทั้งมาตรฐานของการผสมสัญญาณอีกด้วย มาตรฐานการผสมสัญญาณที่ใช้กันมากในปัจจุบัน คือ Frequency Shift Keying(FSK) , Phase Shift Keying(PSK) และ Quadrature Amplitude Modulation(QAM)

Frequency Shift Keying นั้น เราทราบกันดีแล้วว่าใช้ในโมเด็มความเร็วต่ำ โดยแทน "0" และ "1" ด้วยความถี่ต่างกัน ผู้ส่งจะใช้ความถี่ 2 ความถี่แทน "0" และ "1" ของด้านส่ง ส่วนผู้รับก็ใช้ความถี่อีก 2 ความถี่แทน "0" และ"1" ของด้านรับเช่นกัน ทั้งหมดจึงใช้ความถี่รวม 4 ความถี่ การผสมสัญญาณแบบ FSK มักใช้กับโมเด็มความเร็วประมาณ 300 ถึง 600 บิตต่อวินาที ความเร็วสูงสุดของโมเด็มที่ใช้เทคนิคการผสมสัญญาณแบบนี้อยู่ที่ 1,200 บิตต่อวินาที แต่ตามปกติแล้วโมเด็มในท้องตลาดสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคลที่ใช้การผสมสัญญาณแบบ FSK จะรับส่งข้อมูลด้วยความเร็ว 300 บิตต่อวินาที การผสมสัญญาณแบบ FSK นี้ อัตราการส่งข้อมูล (Bit rate) จะเท่ากับ Baud Rate เสมอดังรูป 2.1.15



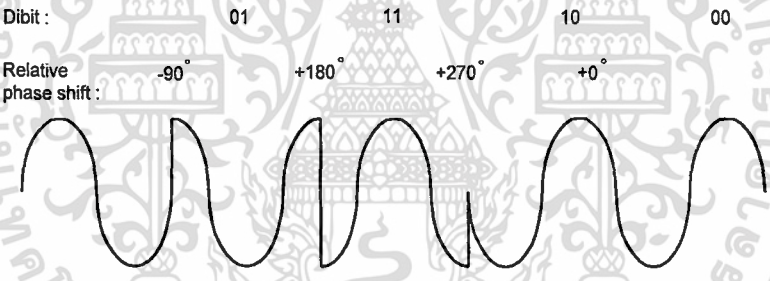
รูปที่ 2.1.15 แสดงการส่งข้อมูลโดยใช้ FSK

สำหรับการผสมสัญญาณแบบ Phase Shift Keying(PSK) นั้น ใช้หลักการที่ว่าแทนข้อมูล "0" และ "1" ด้วยการเปลี่ยนแปลงมุมของช่วงสัญญาณในการส่ง (Phase) เช่น เราอาจกำหนดว่า "0" แทนด้วยมุมต่อเนื่องกันไป และ "1" แทนด้วยมุมเปลี่ยนไปจากเดิม 180 องศา ถ้าเราส่งข้อมูล 0101 รูปร่างคลื่นในสายส่งจะเป็นดังรูปที่ 2.1.16



รูปที่ 2.1.16 การใช้ PSK ผสมสัญญาณแบบหนึ่งบิต

จากหลักการที่เราเปลี่ยน Phase ในสัญญาณส่งนี้ ด้วยมุมทั้ง 360 องศาของคลื่นพาหะ ทำให้เราสามารถแบ่งการเปลี่ยนแปลงของ Phase ให้เล็กลงไปได้อีก เช่น เปลี่ยน Phase ทีละ 90 องศา ซึ่งในหนึ่งลูกคลื่นจะเปลี่ยนได้ถึง 4 สภาวะ และกำหนดให้แต่ละสภาวะแทนด้วยข้อมูลสองบิตได้ จากตารางเราสามารถกำหนด 00,01,10 และ 11 ให้มีการเปลี่ยน Phase ได้ครั้งละ 90 องศา เมื่อเราส่งข้อมูล เช่น 01 11 10 00 รูปร่างของคลื่นในสายส่งจะเป็นดังรูปที่ 2.1.17



รูปที่ 2.1.17 การใช้ PSK ผสมสัญญาณแบบ 2 บิต

ถ้าคลื่นพาหะมีความถี่ 600Hz ข้อมูลที่ส่งออกไปจะมีค่าเท่ากับ 1200บิตต่อวินาที เนื่องจากการเปลี่ยนแปลง Phase หนึ่งครั้งเท่ากับข้อมูลสองบิต

เราจะเห็นว่าการผสมสัญญาณแบบ PSK นี้ อัตราการส่งข้อมูล (Bit Rate) จะมีค่าสูงกว่าอัตราการเปลี่ยนแปลงของสัญญาณในสายส่ง (Baud Rate) ได้ การผสมสัญญาณที่เราใช้กันมาก คือ ใช้ข้อมูล 2 บิต 3 บิต และ 4 บิต ในการเปลี่ยน Phase โดยแบ่งมุมของคลื่นพาหะให้มีการเปลี่ยนแปลง เท่ากับ 90 องศา 45 องศา และ 22.5 องศา ตามลำดับ ความเร็วสูงสุดที่ใช้ PSK คือ 9600 บิตต่อวินาที ซึ่งมีการเปลี่ยนแปลงของมุมครั้งละ 22.5 องศา การเปลี่ยนแปลงมุม 22.5 องศานี้ นับว่าน้อยมาก ทำให้ข้อมูลมักจะผิดพลาดอยู่เสมอ ส่วนมากจึงใช้งาน PSK ที่ความเร็ว 4800 บิตต่อวินาทีแทน และที่ความเร็วสูงกว่านี้เราจะใช้เทคนิคการผสมสัญญาณที่ซับซ้อนขึ้นไปอีก

Quadrature Amplitude Modulation (QAM) เป็นการผสมสัญญาณที่ใช้ทั้งการเปลี่ยน Phase และขนาดของสัญญาณควบคู่กันไป สำหรับใช้กับโมเด็มความเร็วสูง ซึ่งถ้าใช้การเปลี่ยน Phase เพียงอย่างเดียว มุมที่เปลี่ยนแปลงจะมีค่าน้อยเกินไป ทำให้เกิดข้อผิดพลาดได้ง่าย ถ้าเราใช้การเปลี่ยน Phase และขนาดของสัญญาณ ค่าของมุมก็จะอยู่ห่างกันมากขึ้น ปกติที่มีใช้กันอยู่จะมี Phase ต่างกัน 8 Phase และขนาดของสัญญาณต่างกัน 4 ระดับ ใช้แทนข้อมูล 16 สถานะ ซึ่งในหนึ่งลูกคลื่นจะสามารถส่งข้อมูลได้คราวละ 4 บิต การผสมสัญญาณ QAM บางแบบจะใช้ Phase ต่างไปจากนี้ เช่น ใช้ Phase ต่างกัน 12 Phase และขนาดของสัญญาณ 3 ระดับ หรืออาจใช้ Phase ต่างกัน 8 Phase และขนาดของสัญญาณต่างกัน 2 ระดับ ขึ้นอยู่กับการออกแบบ แต่ว่าในมาตรฐานเดียวกัน โมเด็มจะต้องใช้การแบ่ง Phase และระดับสัญญาณเท่ากันเสมอ ความเร็วในการรับส่งข้อมูลของ QAM อยู่ที่ 9600 บิตต่อวินาที โดยใช้ความถี่พาหะ 2400Hz และในหนึ่งลูกคลื่นแทนข้อมูลได้คราวละ 4 บิต เทคนิคการผสมสัญญาณแบบ QAM นี้อาจถูกนำไปใช้กับโมเด็มความเร็วปานกลางก็ได้ เช่น ที่ความเร็ว 2400 บิตต่อวินาที เป็นต้น

นอกจากนี้ยังมีการผสมสัญญาณสำหรับโมเด็มความเร็วสูงอีกหลายแบบ แต่ส่วนมากยังไม่ได้รับการยอมรับให้เป็นมาตรฐานในปัจจุบัน จึงไม่นำมากล่าวถึง การผสมสัญญาณในแบบ QAM มีความซับซ้อนมาก ดังนั้นวงจรทางด้านฮาร์ดแวร์ ไม่ว่าจะเป็นวงจรด้านส่ง วงจรด้านรับ และการแยกสัญญาณต่างๆ ยุ่งยากกว่าที่ใช้กับวงจรเปลี่ยน Phase อย่างเดียว ราคาของโมเด็มที่ใช้เทคนิคแบบQAM จึงสูงกว่าโมเด็มที่ใช้เทคนิค PSK อยู่มาก

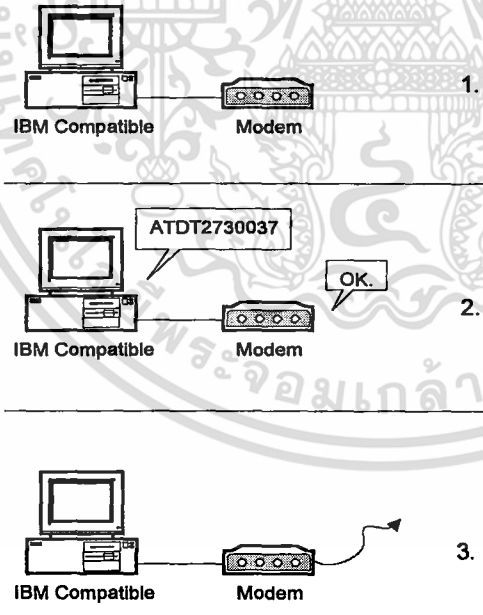
มาตรฐานคำสั่งของโมเด็ม

ตามที่ทราบแล้วว่า โมเด็มในยุคแรกๆเท่านั้น ทำหน้าที่เปลี่ยนสัญญาณข้อมูลของคอมพิวเตอร์เป็นสัญญาณเสียงแล้วส่งออกไปทางสายโทรศัพท์และรับกลับมาเท่านั้น ไม่มี Function พิเศษช่วยการทำงานให้สะดวกขึ้นแต่อย่างใด ต่อมาจึงมีผู้คิดเพิ่มเติมความสามารถให้กับโมเด็มขึ้นให้เราใช้งานง่ายขึ้น และสะดวกรวดเร็วกว่าเดิม โดยการส่งคำสั่งไปให้โมเด็มทำ function ต่างๆ เมื่อมีคำสั่งของโมเด็มเกิดขึ้นก็เป็นเรื่องของมาตรฐานติดตามมาควบคุมให้การสั่งงานโมเด็มเป็นไปอย่างเดียวกันเหมือนอย่างเรื่องของความเร็วในการรับส่งข้อมูลและเทคนิคการผสมสัญญาณ

โมเด็มในยุคแรก การใช้งานต้องอาศัยคนเข้าช่วยเหลือเกือบทุกอย่าง เนื่องจากโมเด็มทำหน้าที่รับส่งสัญญาณ และเปลี่ยนสัญญาณที่ได้รับมาให้เครื่องคอมพิวเตอร์เท่า

นั้นเอง การหมุนโทรศัพท์ และการติดต่อกับอีกด้านหนึ่งเพื่อส่งข้อมูลไปจนถึงการรับโทรศัพท์ สำหรับโมเด็มด้านรับนั้น ต้องใช้คนทำสิ่งเหล่านี้ทั้งหมด โมเด็มแบบนี้จึงใช้งานลำบาก ผู้ใช้ต้องมีความเข้าใจระบบการทำงานของโมเด็มพอสมควร การปรับฟังก์ชั่นการทำงานต่างๆ ของโมเด็มมักจะใช้ผลักสวิทช์เล็กๆ (DIP Switch) บนตัวโมเด็ม ถ้าเราไปดูโมเด็มแบบเก่า จะเห็นสวิทช์ที่ว่านี้เต็มไปหมด นอกจากการปรับสวิทช์จะไม่สะดวกในการใช้งานแล้วยังจำยากด้วยว่า สวิทช์ตัวไหนใช้ทำอะไร และต้องผลักสวิทช์ไปทางไหน การสั่งงานโมเด็มโดยใช้คำสั่งจึงเกิดขึ้น

ปกติโมเด็มจะติดต่อกับเครื่องคอมพิวเตอร์ เพื่อทำหน้าที่รับส่งข้อมูลอยู่แล้ว ดังนั้นขณะที่โมเด็มยังไม่ได้ติดต่อกับปลายทางเพื่อส่งข้อมูล คอมพิวเตอร์สามารถส่งคำสั่งต่างๆให้โมเด็มได้โดยไม่รบกวนการส่งข้อมูลแต่อย่างใด ในขั้นแรกเมื่อเปิดสวิทช์ให้โมเด็มทำงาน สัญญาณที่โมเด็มได้รับจากคอมพิวเตอร์ จะถือว่าเป็นคำสั่งทั้งหมดจนกว่าจะติดต่อกับโมเด็มปลายทางได้ และเมื่อโมเด็มทำการส่งข้อมูลผ่านสายส่ง สัญญาณต่างๆที่คอมพิวเตอร์ส่งให้โมเด็มจะถือว่าเป็นข้อมูลทั้งหมด จนกระทั่งหยุดส่งข้อมูลโดยการเลิกการติดต่อกับปลายทาง หรือวางสายโทรศัพท์นั่นเอง โมเด็มก็จะกลับมาอยู่ในช่วงรับคำสั่งจากคอมพิวเตอร์อีกครั้งหนึ่ง



1. คอมพิวเตอร์เชื่อมต่อกับโมเด็ม
2. สั่งให้โมเด็มหมุนโทรศัพท์
3. โมเด็มส่งสัญญาณไปให้ชุมสาย

รูปที่ 2.1.18 คอมพิวเตอร์สั่งงานโมเด็มโดยใช้ AT command

คำสั่งโมเด็มจะถูกควบคุมการทำงานที่จำเป็นทั้งหมดของโมเด็ม เช่น ตอบรับสัญญาณโทรศัพท์ ที่เรียกเข้ามา เลือกให้ทำงานในแบบ Echo on หรือ Echo off ต่อเข้าสายโทรศัพท์รีเซ็ตโมเด็ม สั่งให้โมเด็มหมุนโทรศัพท์ตามเบอร์ที่กำหนด ปรับพารามิเตอร์ต่างๆของโมเด็มฯลฯ ซึ่งถ้าหากไม่ใช้คำสั่งโมเด็มแล้ว ผู้ใช้จะต้องกำหนดตัวแปรเหล่านั้นด้วยวิธีการผลักสวิทช์บนโมเด็มตามที่ได้กล่าวมาข้างต้น การใช้คำสั่งจึงสะดวกและง่ายต่อการใช้งานมาก ข้อดีอันหนึ่งของการใช้คอมพิวเตอร์ส่งคำสั่งให้โมเด็มก็คือ ซอฟต์แวร์ติดต่อสื่อสาร สามารถปรับตัวแปรต่างๆ ของโมเด็มให้เป็นไปอย่างที่ต้องการ โดยที่ผู้ใช้ไม่ต้องรู้รายละเอียดใดๆเลย โปรแกรมจะจัดการให้เสร็จและติดต่อส่งข้อมูลได้ทันที โปรแกรมคนละโปรแกรมอาจใช้งานโมเด็มไม่เหมือนกัน แต่ละโปรแกรมก็จะปรับโมเด็มให้ทำงานต่างกันได้โดยไม่ต้องแก้ไขส่วนที่เป็นฮาร์ดแวร์ของโมเด็มเลย การใช้งานโมเด็มจึงมีความคล่องตัวมากกว่าการใช้สวิทช์เลือกแบบเก่า ซึ่งถ้ามีการเปลี่ยนแปลงอะไร เราก็ต้องปรับสวิทช์กันทีหนึ่งทุกครั้งไป และอาจเกิดความผิดพลาดได้ง่ายกว่าการใช้คำสั่ง สั่งงานโมเด็ม

ภายในตัวโมเด็มจะมีหน่วยความจำพิเศษสำหรับเก็บตัวแปรในการทำงานแทนที่สวิทช์แบบเก่า หน่วยความจำนี้จะยังคงเก็บค่าต่างๆเอาไว้ด้วย แม้ว่าจะปิดโมเด็ม หรือดึงปลั๊กโมเด็มออกก็ตาม โมเด็มที่ใช้คำสั่งของ Hayes เรียกหน่วยความจำส่วนนี้ว่า S-Register เอาไว้ใช้เก็บพารามิเตอร์ ในการทำงานของโมเด็ม เช่น จำนวนครั้งที่จะตอบรับสัญญาณเรียกเข้า ช่วงเวลาสำหรับรอสัญญาณก่อนหมุนโทรศัพท์ฯลฯ

ซอฟต์แวร์ที่ใช้สื่อสารสามารถเปลี่ยนค่าตัวแปรเหล่านี้ชั่วคราว หรือเปลี่ยนค่าถาวรไปเลยก็ได้ โดยใช้คำสั่งเก็บค่าตัวแปรเอาไว้ หน่วยความจำพิเศษนี้ บางชนิดใช้แบตเตอรี่เล็กๆ คอยจ่ายไฟให้เวลาที่เราปิดโมเด็ม เพื่อป้องกันค่าต่างๆหายไปจากหน่วยความจำ ดังนั้น เมื่อใช้โมเด็มไปนานๆ แบตเตอรี่ดังกล่าวจะหมดลง เราจำเป็นต้องเปลี่ยนอันใหม่ให้ ไม่เช่นนั้นการทำงานของโมเด็มอาจผิดพลาดได้ เนื่องจากค่าของตัวแปรในหน่วยความจำหายไป โมเด็มบางชนิดเก็บค่าตัวแปรในหน่วยความจำแบบที่ไม่ต้องใช้แบตเตอรี่จ่ายไฟสำรองให้โมเด็มแบบนี้เราก็ไม่จำเป็นต้องเปลี่ยนแบตเตอรี่ภายในให้มัน และโมเด็มบางแบบก็ไม่ยอมให้เราเปลี่ยนค่าตัวแปรเหล่านี้อย่างถาวรโดยเก็บค่าต่างๆ เอาไว้ใน ROM (Read Only Memory) การเปลี่ยนค่าตัวแปรจะเป็นไปชั่วคราวเท่านั้น เมื่อเปิดปิดโมเด็มใหม่ ค่าต่างๆจะกลับเหมือนเดิมตามที่ผู้ผลิตกำหนดเอาไว้ใน ROM ของโมเด็มนั่นเอง

ต่อไปจะกล่าวถึงคำสั่งที่มีใช้งานของ Hayes และ V.25 bis ว่าแต่ละคำสั่งทำหน้าที่อะไรและใช้คำสั่งว่าอย่างไร เพื่อให้เข้าใจการทำงานของคำสั่งโมเด็มมากยิ่งขึ้นดังนี้

มาตรฐานคำสั่งโมเด็มของ Hayes

Hayes Command เป็นคำสั่งที่ใช้สั่งงานโมเด็ม มีอีกชื่อหนึ่งเรียกกันว่า AT Command เพราะคำสั่งทุกคำสั่งขึ้นต้นด้วย AT เสมอ เมื่อจบคำสั่งให้ปิดท้ายด้วย CR Carriage Return หรือกดปุ่ม Enter โมเด็มจะรับคำสั่งไปทำงานทันที และตอบคำว่า OK กลับมา คำสั่งเรียงตามตัวอักษร A ถึง Z จะมีดังนี้คือ

- ATA เป็นคำสั่งให้โมเด็มตอบรับสัญญาณโทรศัพท์ที่เรียกเข้ามา เมื่อโมเด็มทำคำสั่งนี้ การติดต่อระหว่างปลายทางทั้งสองข้างจะเริ่มขึ้น และโมเด็มจะอยู่ในช่วงรับส่งข้อมูล ไม่รับคำสั่งจากเครื่องคอมพิวเตอร์ จนกว่าจะเลิกการติดต่อกับปลายทางเสียก่อน

- ATB เป็นคำสั่ง เลือกการทำงานของโมเด็ม ว่าจะใช้มาตรฐานการผสมสัญญาณตาม CCITT หรือมาตรฐานของสหรัฐอเมริกา (Bell Standard) โดยใช้คำสั่ง ATB0 สำหรับมาตรฐาน CCITT และ ATB1 สำหรับมาตรฐานสหัฐอเมริกา ปกติเราจะใช้มาตรฐานตาม CCITT เท่านั้น และโมเด็มมักจะตั้งค่า ATB0 มาจากโรงงานอยู่แล้ว

- ATD ใช้สั่งให้โมเด็มทำการหมุนโทรศัพท์อัตโนมัติ แทนการที่ใช้คนหมุนเอง หรือที่เรียกว่า Audio Dialing คำสั่งนี้มักตามด้วย ชนิดของโทรศัพท์ที่ต่ออยู่ด้วย เช่น ATDT จะเป็นคำสั่งให้โมเด็มหมุนโทรศัพท์แบบกดปุ่ม และ ATDP จะเป็นคำสั่งสำหรับหมุนโทรศัพท์แบบมือหมุนตัวอย่างการใช้งาน เช่น ATDT 2345678 ตามด้วย Return หรือ Enter เป็นตัวจบคำสั่ง

- ATE สั่งให้โมเด็ม Echo ข้อความ กลับไปให้เครื่องคอมพิวเตอร์ โดยใช้คำสั่ง ATE0 สำหรับไม่ให้โมเด็ม Echo และ ATE1 เพื่อให้โมเด็ม Echo ข้อความกลับไปให้เครื่องคอมพิวเตอร์ ในขณะที่โมเด็มรับคำสั่งอยู่ใน Command Mode เมื่อเราใช้โมเด็มรับส่งข้อมูลแบบ Full Duplex จะต้องเลือกใช้ ATE1 เพื่อให้ข้อความที่เราพิมพ์ส่งไปยังโมเด็ม Echoกลับมาให้เราเห็นบนจอภาพ และถ้าเป็น Half Duplex ก็เลือกใช้ ATE0 เพราะเครื่องคอมพิวเตอร์จะพิมพ์ข้อความนั้นบนจอภาพอยู่แล้ว คำสั่ง ATE0 และ ATE1 นี้ จะมีผลเฉพาะก่อนการเชื่อมต่อกับปลายทางเท่านั้น เมื่อเราติดต่อกับปลายทางได้ โมเด็มจะรับส่งข้อมูลผ่านสายโดยไม่มีการ Echo ข้อความใดๆทั้งสิ้น เครื่องคอมพิวเตอร์ต้นทาง และปลายทางจะเป็นผู้รับผิดชอบการ Echo ข้อความตามแบบ Full Duplex และ Half Duplex กันเองจนสิ้นสุดการติดต่อ

- ATH เป็นคำสั่งให้โมเด็มวางสายโทรศัพท์ หรือ ปลดตัวเองออกจากสายโทรศัพท์ และวางหูนั่นเอง ส่วนมากคำสั่งนี้จะใช้เมื่อเลิกติดต่อกับปลายทาง โดยสั่ง ATH

เพื่อวางหูหลังจากส่งข้อความจบแล้ว ก่อนใช้คำสั่งนี้ เราต้องสั่งให้โมเด็มกลับมาอยู่ในสภาวะรับคำสั่งเสียก่อน ซึ่งทำได้โดยส่งเครื่องหมาย + สามตัวไปให้โมเด็ม แต่ละตัวห่างกันประมาณครึ่งวินาที โมเด็มจะถือว่าจบการส่งข้อมูล และรอรับคำสั่งจากเครื่องคอมพิวเตอร์ต่อไป

- ATL ใช้สำหรับเป็นตัวปรับความดังของลำโพงภายในตัวโมเด็ม ซึ่งปรับความดังได้ 3 ระดับนี้ คือ ATL0 และ ATL1 เป็นคำสั่งปรับความดังให้น้อยที่สุด ATL2 จะเป็นระดับความดังปานกลาง และ ATL3 สำหรับปรับความดังของลำโพงมากที่สุด แต่โมเด็มบางแบบจะใช้ปุ่มหมุนปรับความดังของลำโพงภายในแทนก็ได้ ปกติเราก็ไม่ค่อยใช้คำสั่งนี้กันมากนัก

- ATM ใช้สำหรับควบคุมการทำงานของลำโพงภายในโมเด็ม เช่น ATM0 จะสั่งไม่ให้ลำโพงภายในโมเด็มทำงาน คือ เงียบตลอดเวลา ATM1 จะให้ลำโพงมีเสียงเฉพาะตอนที่ยังต่อกับปลายทางไม่ได้เท่านั้น เพื่อให้เราฟังเสียงการหมุนโทรศัพท์ และสัญญาณตอบรับได้ว่ามีอะไรผิดปกติหรือเปล่า ATM2 จะสั่งให้ลำโพงมีเสียงตลอดเวลา แม้ว่าต่อกับปลายทางได้แล้วก็ตาม

- ATO เป็นคำสั่งให้โมเด็มอยู่ในโหมด Online คือ เปลี่ยนจากการรับคำสั่งจากเครื่องคอมพิวเตอร์มาเป็นการรับส่งข้อมูลผ่านสายส่ง เหมือนกับการที่เรากดปุ่ม DATA บนตัวโมเด็มนั่นเอง

- ATS ใช้สำหรับเปลี่ยนแปลงค่าของ S-Register ซึ่งเป็นที่เก็บพารามิเตอร์ในการทำงานต่างๆ Hayes เช่น S0ใช้เก็บจำนวนครั้งของสัญญาณเรียกเข้า ก่อนที่โมเด็มจะทำการรับโทรศัพท์ ถ้าเราต้องการให้โมเด็มรับโทรศัพท์ที่เรียกเข้ามาโดยมีสัญญาณเรียกดัง 4 ครั้ง ก็สั่งว่า ATS0 = 4 เป็นต้น S-Register ของ Hayes แสดงในตารางประกอบว่าตัวไหนมีหน้าที่ทำอะไร นอกจากนี้คำสั่ง ATS ยังใช้เรียกดูค่าของ S-Register ได้อีกด้วยว่าค่าที่เก็บอยู่ในปัจจุบันเป็นเท่าไร โดยใช้คำสั่ง เช่น ATS0? โมเด็มจะนำค่าของ S0 ที่เก็บอยู่ในขณะนั้นมาแสดงให้เราดู

- ATV เป็นคำสั่งบอกให้โมเด็ม แสดงผลลัพธ์ของคำสั่งต่างๆเป็นรหัสตัวเลข (Digit Code) หรือ เป็นตัวอักษรภาษาอังกฤษ ATV0 จะสั่งให้โมเด็มแสดงผลลัพธ์ของคำสั่งกลับมาเป็นตัวเลข และ ATV1 จะให้โมเด็มแสดงผลลัพธ์เป็นตัวอักษร เช่น เมื่อเราสั่งให้โมเด็มหมุนโทรศัพท์ด้วยคำสั่ง ATDT 2345678 <Cr> โมเด็มจะตอบกลับมาว่า OK ในแบบตัวอักษรหรือตอบกลับมาเป็นเลข 0 ในแบบตัวเลข และถ้าโมเด็มต่อกับปลายทางได้ ก็จะตอบ

ว่า Connect 1200 ในกรณีที่ใช้ความเร็ว 1200บิตต่อวินาที หรือ ตอบเป็นรหัสตัวเลข 5 แทน ปกติเราจะใช้โมเด็มตอบผลลัพธ์เป็นตัวอักษร เพื่อให้เข้าใจง่ายว่าขณะนี้โมเด็มมีสภาพเป็นอย่างไร ซอฟต์แวร์ควบคุมโมเด็มทั่วไปก็มักจะให้โมเด็มตอบรับเป็นตัวอักษรเช่นกัน โมเด็มที่เป็นแบบ Hayes Compatible จึงต้องแสดงผลลัพธ์ของคำสั่งต่างๆให้เหมือนกับที่ บริษัท Hayes ใช้ทุกตัวอักษร มิฉะนั้นซอฟต์แวร์ควบคุมโมเด็มจะสั่งงานโมเด็มผิดพลาดได้ หรือสั่งงานไม่ได้เลย เนื่องจากโมเด็มตอบรหัสกลับไปไม่ตรงตามที่บริษัท Hayes กำหนด

- ATX เป็นคำสั่งโมเด็มแสดงผลลัพธ์ในการหมุนโทรศัพท์แบบต่างๆ เช่น ATX0 และ ATX1 จะเป็นคำสั่งให้โมเด็มไม่ตรวจสอบ Dial Tone ก่อนหมุนโทรศัพท์ และไม่รับรู้สัญญาณที่ตอบกลับมาว่าปลายทางไม่ว่าง (Busy Signal) ATX2 จะให้โมเด็มตรวจสอบก่อนว่าสายโทรศัพท์มี Dial Tone ก่อนทำการหมุนโทรศัพท์ แต่ไม่รับรู้สัญญาณปลายทางไม่ว่างเหมือนเดิม ถ้าหากโมเด็มไม่ได้รับ Dial Tone จากชุมสายใน 5 วินาที โมเด็มจะตอบกลับไปยังคอมพิวเตอร์ว่า "No Dialtone" ATX3 จะให้โมเด็มรับรู้สัญญาณปลายทางไม่ว่าง (Busy Signal) และ ATX4 ซึ่งเป็นคำสั่งที่เราใช้เป็นส่วนมาก จะให้โมเด็มตรวจสอบ Dial

- Tone ก่อนที่จะทำการหมุนโทรศัพท์ และรับรู้สัญญาณปลายทางไม่ว่างได้ด้วย คือ เท่ากับ ATX2 รวมกับ ATX3 นั่นเอง

- ATZ เป็นคำสั่งให้รีเซ็ตโมเด็มกลับมาอยู่ในสภาวะที่ผู้ผลิตกำหนด (Default Configuration) เมื่อโมเด็มได้รับคำสั่งนี้ มันจะยกเลิกคำสั่งเปลี่ยนแปลงค่าต่างๆที่ได้รับมาทั้งหมด และดึงค่าพารามิเตอร์ที่ผู้ผลิตกำหนดเอาไว้จากโรงงาน ซึ่งเก็บอยู่ใน ROM (Read Only Memory) ของโมเด็มมาใช้ คำสั่งนี้มีประโยชน์มากในกรณีที่เราต้องการให้ค่าต่างๆกลับคืนมาเหมือนเดิม โดยไม่ต้องคอยจำว่าเปลี่ยนแปลงพารามิเตอร์ใดบ้าง และค่าเดิมของมันเป็นอะไร

ทั้งหมดเป็นตัวอย่างคำสั่งมาตรฐานของโมเด็มตามแบบ Hayes ซึ่งซอฟต์แวร์สื่อสารใช้ในการสั่งงานและควบคุมโมเด็ม ปัจจุบันโมเด็มบางแบบอาจมีคำสั่งเพิ่มเติมมากกว่านี้ เพื่อให้การใช้งานสะดวกขึ้น เราเรียกคำสั่งที่เพิ่มมานี้ว่า Extended Command Set แต่ซอฟต์แวร์สื่อสารทั่วๆไปยังคงใช้เพียงแค่คำสั่งมาตรฐานนี้เท่านั้น เนื่องจากถ้าใช้คำสั่งที่เพิ่มขึ้นมา โมเด็มบางแบบจะรับคำสั่งไม่ได้ โดยเฉพาะโมเด็มราคาถูกทั่วไปจะรับคำสั่งมาตรฐาน ดังนั้น คำสั่งที่เพิ่มขึ้นมาจึงยังไม่มีผู้นิยมใช้กันเท่าใดนัก

มาตรฐานคำสั่งโมเด็มของ CCITT V.25 bis

คำสั่งมาตรฐานของ CCITT ที่ใช้สั่งงานโมเด็มนั้นเรียกว่า V.25 bis Command ซึ่งไม่เกี่ยวกับมาตรฐานของตัวโมเด็มแต่อย่างใด V.25 bis นี้ เป็นคำสั่งที่ใช้สั่งให้โมเด็มทำงานต่างๆเหมือนกับ Hayes Command นั่นเอง เพียงแต่ว่า V.25 bis นี้ใช้กันค่อนข้างน้อย ไม่เหมือนมาตรฐานแบบ Hayes Command ซึ่งคอมพิวเตอร์ส่วนบุคคลสั่งงานโมเด็มแทบทั้งหมด โครงสร้างของคำสั่งจะเป็นคำสั่งตัวอักษรภาษาอังกฤษ ปิดท้ายด้วย CRLF (Carriage Return ตามด้วย Line Feed) เสมอ ถ้าคำสั่งนั้นถูกต้อง โมเด็มจะตอบคำว่า VAL (Valid) แสดงการรับรู้คำสั่งนั้น และถ้าคำสั่งนั้นผิดพลาด หรือโมเด็มไม่รู้จัก ก็จะตอบกลับมาว่า INV (Invalid) แทน คำสั่ง V.25 bis เรียงตามลำดับดังนี้

- EON เป็นคำสั่งให้โมเด็ม Echo ข้อความกลับมายังเครื่องคอมพิวเตอร์ คล้ายกับคำสั่ง ATE1 ของ Hayes Command นั่นเอง การรับส่งข้อมูลแบบ Full Duplex ต้องสั่งให้โมเด็ม Echo ข้อมูลกลับมาเสมอ และการ Echo ข้อมูลนี้ก็ไม่เกี่ยวกับการรับส่งข้อมูล เมื่อมีโมเด็มต่อกับปลายทางได้แล้ว คือ โมเด็มจะ Echo ข้อความเฉพาะใน Command Mode เท่านั้น ในการรับส่งข้อมูล เครื่องคอมพิวเตอร์ทั้งสองด้านจะดูแลเรื่องการ Echo ข้อความระหว่างกันด้วยตัวเองไม่ว่าจะเป็นแบบ Full Duplex หรือ Half Duplex

- EOF เป็นคำสั่ง Echo off หรือเหมือนกับ ATE0 ของ Hayes คำสั่งนี้จะสั่งไม่ให้โมเด็ม Echo ข้อความกลับไปยังคอมพิวเตอร์ ซึ่งการรับส่งข้อมูลในแบบ Half Duplex จะใช้ Echo แบบนี้

- CRN (Call Request with Number Provided) ใช้สำหรับสั่งให้โมเด็มหมุนโทรศัพท์ไปยังเบอร์ที่กำหนด โดยมีตัวกำกับ T สำหรับโทรศัพท์แบบกดปุ่ม และ P สำหรับหมุนโทรศัพท์แบบมือหมุน เหมือนกับคำสั่ง ATDT และ ATDP ของ Hayes ตัวอย่างเช่น CRNT 2345678 จะเป็นคำสั่งให้หมุนโทรศัพท์แบบกดปุ่มไปยังเบอร์ 2345678 ถ้าโทรศัพท์เป็นแบบมือหมุนก็ให้ใช้ตัว P เป็น CRNP 2345678 เมื่อโมเด็มได้รับคำสั่งก็จะตอบคำว่า VAL กลับมาและเริ่มเรียกไปยังเบอร์ที่กำหนด

- CRS (Call Request with Stored Number) เป็นคำสั่งให้โมเด็มหมุนโทรศัพท์ตามเบอร์ที่เก็บเอาไว้ภายในหน่วยความจำของโมเด็ม ซึ่งก่อนใช้คำสั่งนี้ เราก็ต้องทำการเก็บเบอร์โทรศัพท์เอาไว้ในโมเด็มเสียก่อน ถ้าต้องการให้โมเด็มหมุนเบอร์โทรศัพท์เบอร์แรกที่เก็บไว้ ก็ใช้คำสั่งว่า CRST1 หรือ CRSP1 ตามชนิดของโทรศัพท์แบบกดปุ่มหรือแบบมือหมุน

- เครื่องหมาย < ใช้สำหรับหยุดชั่วขณะ เมื่อทำการหมุนโทรศัพท์ เช่น ในกรณีของการใช้โทรศัพท์ภายในตัดออกสายนอก เราต้องโมเด็มรอชั่วขณะก่อนทำการหมุนเบอร์ 7 ตัวตามปกติ ถ้าโทรศัพท์ภายในตัดเบอร์ 9 เพื่อหมุนออกสายนอก เราก็ใช้คำสั่งให้โมเด็มหมุนเบอร์ 9 ก่อน และรอการตัดสายนอก แล้วจึงค่อยหมุนเบอร์ 7 ตัว ดังนี้ CRNT 9 < 2345678 เป็นต้น

- PRN (Telphone Number Storage) ใช้สำหรับเก็บเบอร์โทรศัพท์ที่ใช้บ่อยๆ เอาไว้ภายในหน่วยความจำของโมเด็ม โดยทั่วไปจะเก็บได้ประมาณ 20 เบอร์ แต่ละเบอร์มีความยาวไม่เกิน 20 ตัวเลข เช่นถ้าเราต้องการเก็บเบอร์โทรศัพท์ 2345678 เอาไว้ในหน่วยความจำของโมเด็มตัวแรก ก็ใช้คำสั่ง PRN1;2345678 ซึ่งเครื่องหมาย ; จะเป็นตัวแยกลำดับที่ต้องการเก็บออกจากเบอร์โทรศัพท์

- RLN (Request List of Number in Memory) เป็นคำสั่งที่ใช้เรียกดูเบอร์โทรศัพท์ที่เราเก็บไว้ในโมเด็ม ว่าลำดับที่เท่าไรเก็บเบอร์อะไรไว้ เพื่อจะได้สั่งหมุนเบอร์โทรศัพท์ในลำดับที่ถูกต้องกันความผิดพลาด และเอาไว้ใช้ตรวจสอบ เนื่องจากเบอร์ที่เก็บไว้นานๆ เราอาจลืมไปได้ว่าเก็บเบอร์ของใครไว้ลำดับที่เท่าไร

- CLA (Clear Address) ใช้สำหรับลบ หรือ Clear เบอร์โทรศัพท์ที่โมเด็มเก็บเอาไว้ออกไป ในกรณีที่เลิกใช้เบอร์นั้น หรือ จะลบออกทั้งหมดเลยก็ได้ เช่น CLA1 เป็นคำสั่งลบเบอร์โทรศัพท์ลำดับที่ 1 ออกไป และ CLA** จะลบเบอร์โทรศัพท์ที่เก็บเอาไว้ทั้งหมดออกจากโมเด็ม

- DLN (Last Number Redial) จะเป็นคำสั่งให้โมเด็มหมุนโทรศัพท์ตามเบอร์ที่เพิ่งหมุนไปโดยคำสั่ง CRN ที่สั่งหมุนโทรศัพท์ครั้งหลังสุด คำสั่ง DLN จะสั้นกว่าและสะดวกกว่าสั่ง CRN ใหม่อีกครั้งหนึ่ง

- INC (Incoming Call Response) เป็นสัญญาณแสดงว่าโมเด็มได้รับสัญญาณเรียกเข้า หรือมีผู้โทรเข้ามาหาโมเด็ม โมเด็มจะส่งข้อความ INC มาให้เครื่องคอมพิวเตอร์ เพื่อแสดงว่ามีคนกำลังติดต่อเข้ามา INC นี้ไม่ใช่คำสั่ง แต่เป็นข้อความแสดงการทำงานของโมเด็ม

- ONL (Online Indication) เป็นข้อความที่แสดงว่าโมเด็มเชื่อมต่อกันได้เรียบร้อยแล้ว เหมือนข้อความ Connect ของ Hayes นั่นเอง ONL จึงไม่ใช่คำสั่งเช่นเดียวกับ INC เมื่อโมเด็มทั้งสองด้านเชื่อมต่อกันได้เรียบร้อย ก็จะส่งข้อความ ONL มายังเครื่องคอมพิวเตอร์เพื่อแสดงการทำงาน

- CFI (Call Failure Indication) เป็นข้อความแสดงว่าโมเด็มติดต่อไปยังปลายทางไม่สำเร็จ หลังจากที่หมุนโทรศัพท์ไปยังเบอร์ที่กำหนด ปกติ CFI จะมีตัวอักษรบอกสาเหตุที่ติดต่อไม่ได้ตามมาอีก 2 ตัว เช่น CFIET ตัวอักษร ET จะหมายถึง เบอร์ปลายทางไม่ว่าง CFIDT หมายถึง ไม่มี Dial Tone ในสายโทรศัพท์ CFINT หมายถึง ปลายทางไม่ยอมตอบรับ (No Answer)

- DIC (Ignore Incoming Call) เป็นคำสั่งให้โมเด็มตอบรับสัญญาณเรียกเข้ามา เพื่อจะได้ทำการส่งข้อมูลต่อไป คำสั่งนี้จะเหมือนกับคำสั่ง ATA ของ Hayes

- CSP (Change Modem Signalling Rate) ใช้สำหรับปรับความเร็วของโมเด็ม เช่น CSP1200 เป็นคำสั่งให้โมเด็มทำงานที่ความเร็ว 1200 บิตต่อวินาที เมื่อเราสั่งให้โมเด็มเปลี่ยนความเร็วในการรับส่งข้อมูล เครื่องคอมพิวเตอร์ก็ต้องปรับความเร็วตามให้ตรงกันด้วย

นี่ก็เป็นคำสั่งตามมาตรฐาน V.25 bis ของ CCITT ที่ใช้สั่งงานโมเด็ม เมื่อเปรียบเทียบกับคำสั่งตามาตรฐานของ Hayes แล้ว จะเห็นว่ามาตรฐานของ Hayes ใช้ง่ายและเข้าใจคำสั่งง่ายกว่ามาก เนื่องจากใช้ตัวอักษรสื่อความหมายได้ดีกว่า ทำให้จำคำสั่งได้ง่าย นอกจากนี้ผลลัพธ์ที่โมเด็มตอบกลับมายังคอมพิวเตอร์ตามแบบของ Hayes ยังเข้าใจง่ายกว่าอีกด้วย เพราะว่าใช้ตัวอักษรภาษาอังกฤษตัวเต็ม ผู้ใช้อ่านแล้วเข้าใจทันที เช่น CONNECT, BUSY, NO DIALTONE ซึ่งถ้าเป็นตามแบบ V.25 bis จะเข้าใจยาก คือ ตอบกลับมาเป็น ONL, CFIDT ผู้ใช้ส่วนมากต้องเปิดดูคู่มือว่าตัวอักษรที่โมเด็มตอบกลับมาหมายถึงอะไร อย่างไรก็ตามสำหรับเราผู้ใช้ทั่วไป ก็คงจะใช้เฉพาะมาตรฐานตามคำสั่งของ Hayes แม้ว่าโมเด็มบางแบบจะรับคำสั่งได้ทั้ง Hayes และ V.25 bis ก็ตาม

คำสั่งของโมเด็มนี้จะไม่เกี่ยวกับการเชื่อมต่อระหว่างโมเด็ม 2 ตัวแต่อย่างใด โมเด็มที่ใช้คำสั่งตามแบบ Hayes อาจจะติดต่อกับ โมเด็มที่ใช้คำสั่งตาม CCITT V.25 bis ก็ได้ หรือจะติดต่อเข้ากับโมเด็มแบบที่ไม่มีคำสั่งอะไรเลยในตัว (Manual Modem) ก็ได้เช่นกัน ขอให้โมเด็มทั้งสองด้านใช้ความเร็ว และผสมสัญญาณตรงกัน ตาม V.25 bis ที่กำหนดเป็นใช้ได้

2.2 Basic Stamp

ET-BASIC STAMP เป็นชุดควบคุมที่ใช้ในงานควบคุมต่างๆ ได้โดยง่ายและสะดวกมากโดยเราสามารถใช้ภาษา BASIC เขียนโปรแกรมใช้งานขึ้นมาแล้วนำเข้าสู่บอร์ด ET-BASIC STAMP ทางด้าน PRINT PORT ของเครื่อง PC เขียนลงในหน่วยความจำของบอร์ด ET-BASIC STAMP ซึ่งเป็นหน่วยความจำ EEPROM ขนาด 256 BYTE ทดสอบการทำงาน แล้วนำไปใช้ได้โดยทันทีไม่จำเป็นต้องมา COPY EPROM จึงรวดเร็ว

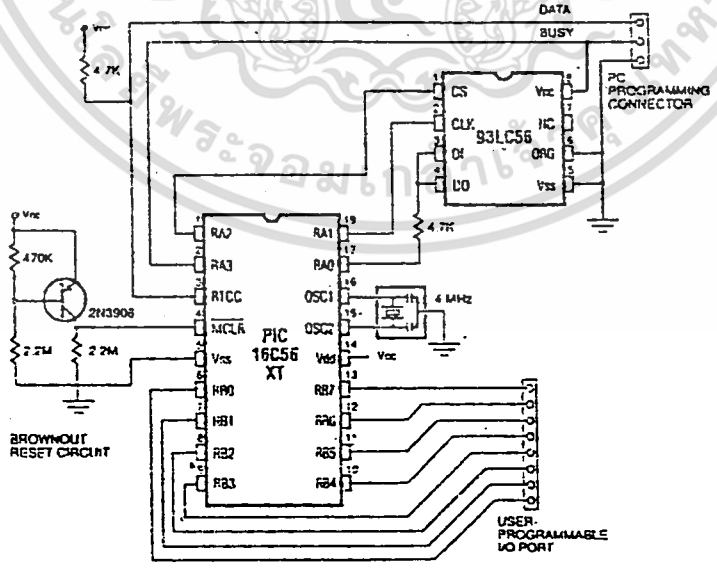
และสะดวกที่สุด ตัวหลักของบอร์ดนี้ก็คือ PIC16C56 CPU แบบ MICRO CONTROLLER ของบริษัท MICROCHIP ซึ่งได้บรรจุตัวโปรแกรม PBASIC ของบริษัท PALALLAX ซึ่งทำงานแบบ INTERPRETER เอาไว้ในตัวทำงานรวมกับตัวเก็บโปรแกรมซึ่งเป็น EEPROM เบอร์ 93LC56 ซึ่งสามารถเก็บข้อมูลได้ 256 BYTE

โดยหนึ่งคำสั่งภาษา BASIC นั้นจะประมาณ 2 ถึง 3 BYTE ซึ่งเราสามารถเขียนคำสั่ง BASIC ได้ 80-100 คำสั่ง ความเร็วในการทำงานของภาษา BASIC นี้ประมาณ 2000 คำสั่ง ต่อวินาที โดยใช้ความถี่ 4 MHZ ในการ RUN ส่วน PORT ก็ได้ใน 8 LINE นี้ เช่น INPUT 3 OUTPUT 5 หรือ INPUT 7 OUTPUT 1 ก็ได้ ซึ่งอาจจะดูน้อยไปแต่ถ้าเทียบกับความง่ายในการใช้งานแล้วก็นับว่าใช้ได้ โดยเฉพาะถ้าเรานำไปใช้ในงานควบคุมขนาดเล็กก็จะเหมาะมาก

ความสามารถของ PORT แต่ละ LINE นั้น สามารถ SINK กระแสได้ 25 MA และ SOURCE กระแสได้ 20 MA แต่ในการใช้งานควรจะรวมทั้ง 8 LINE แล้วไม่ควรเกิน 50 MA <SINK> และ 40 MA <SOURCE>

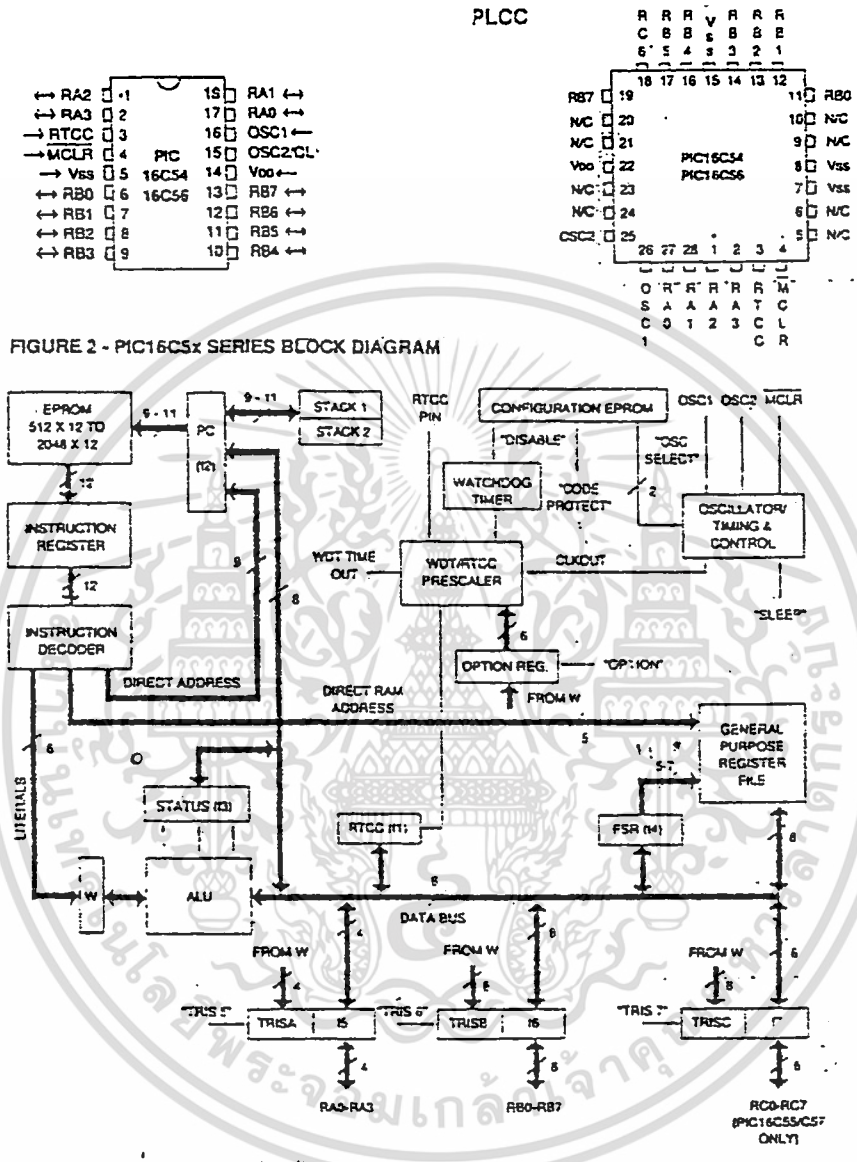
ฮาร์ดแวร์

ฮาร์ดแวร์ ที่เราสามารถสร้างขึ้นนี้ได้พัฒนาเปลี่ยนแปลงวงจรให้สามารถนำมาใช้งานได้ง่ายและสะดวกขึ้น โดยที่รูปจะเป็นวงจร BASIC STAMP ของบริษัท PARALLAX ซึ่งเป็นผู้เขียนและพัฒนานำตัวภาษา BASIC นั้นใส่ในตัว CPU PIC 16C56 นี้



รูปที่ 2.2.1 แสดงวงจร BASIC STAMP

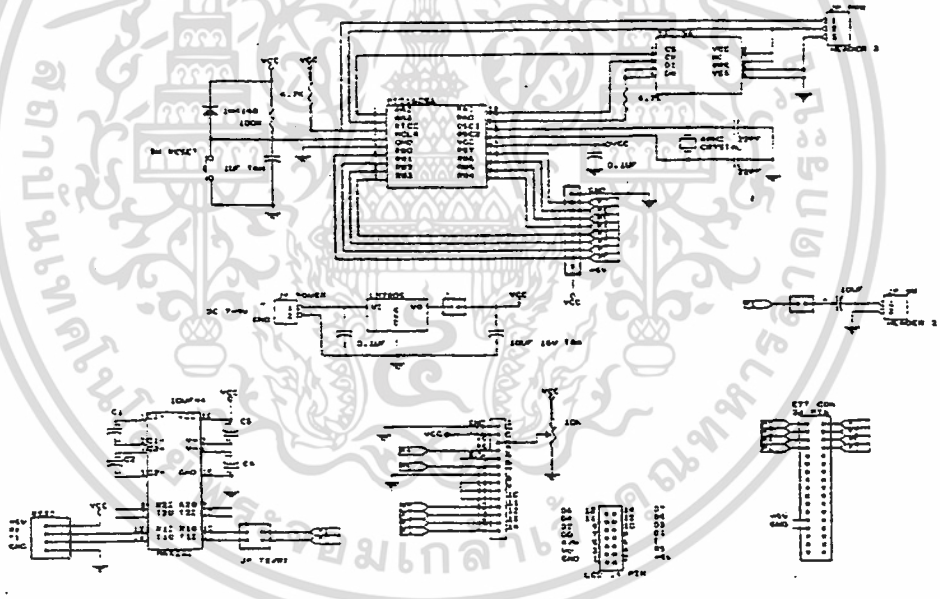
จะเห็นว่าบอร์ดนี้จะใช้ IC เพียง 2 ตัวเท่านั้น คือ PIC 16C56 และ EEPORM 93LC56 โดย CPU 16C56 นั้นจะมี BLOCK DIAGRAM ภายในตามในรูปที่ 2.2.2



รูปที่ 2.2.2 บล็อคไดอะแกรมแสดงการทำงานภายในบอร์ด

การทำงานเราจะต้องใช้ PORT RA0 ถึง RA3 ไปเป็นส่วนติดต่อที่ทำงานกับ EEPROM 93L56 และใช้ขา RTCC (REAL TIME CLOCK COUNTER) ซึ่งเป็นขาอินพุตแบบชมิตทริก-เกอร์ รับข้อมูลจากเครื่อง PC ซึ่งจะส่งข้อมูลที่เราคีย์ เป็นภาษา BASIC และแปลง CODE ส่งทาง PRINTER PORT มายังตัว CPU เราจะมี PORT ใช้งานเป็น INPUT หรือ OUTPUT คือ PORT RB0---RB7

ในส่วนของรูปที่ 2.2.3 จะเป็นวงจร ET-BASIC STAMP ซึ่งจะเพิ่มวงจรบางส่วนเข้าไปจากวงจรเดิมในรูปที่ 1 โดยเพิ่มในส่วนการต่อใช้งานกับ PORT RS232 ได้ ใช้ IC MAX232 และใช้ JUMPER TX/RX เป็นตัวตัดต่อการใช้งานของขา P6(RB6), P7(RB7) นอกจากนี้ยังเพิ่มในส่วนการติดต่อกับ LCD MODULE แบบ CHARACTER TYPE โดยใช้ลักษณะการติดต่อกับ LCD แบบ 4 BIT คือ D4, D5, D6, D7 ต่อกับ P0(RB0),P1(RB1),P2(RB2),P3(RB3) และใช้ P4(RB4) เป็นขาสัญญาณ RS (REGISTOR SELECTION) P5(RB5) เป็นขาสัญญาณ E (ENABLE SIGNAL) ของ LCD MODULE ใช้ JP PRN เป็นขั้วต่อกับเครื่อง PC ใช้พัฒนา BASIC , ขั้ว 34 PIN ใช้ต่อเข้ากับบอร์ด IO ต่าง ETT มีขายอยู่ได้เลยและ JP SK สำหรับต่อลำโพงโดยต่อผ่าน C 10 μF



รูปที่ 2.2.3 แสดงวงจร ET-BASIC STAMP

จะเห็นว่าตัวบอร์ด ET-BASIC STAMP นี้จะสามารถต่อกับ RS232 ได้โดยตรง

2.3 ไมโครคอนโทรลเลอร์ PIC16C56

2.3.1 สถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูล PIC16C5X

PIC16C5X เป็นไมโครคอนโทรลเลอร์แบบซีมอสซึ่งมีหน่วยความจำแบบโปรแกรมได้ครั้งเดียวอยู่ภายใน,มีความเชื่อถือได้สูง,เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ที่มีความสามารถสูงแต่ราคาถูก ผลิตโดยบริษัทไมโครชิปเทคโนโลยี (Microchip Technology) ใช้สถาปัตยกรรมแบบ RISC คำสั่งใช้งานเพียงแค่ 33 คำสั่ง ทุกคำสั่งใช้เวลาในการเอ็กซีคิวส์ 1 ไซเคิล (200 นาโนวินาที) ยกเว้นในกรณีโปรแกรมบรานช์ (Branch) ต้องใช้ถึง 2 ไซเคิล คำสั่งขนาดกว้าง 12 บิต ของ PIC16C5X จะช่วยทำให้ผลของรหัสคำสั่งมีอัตราส่วนเป็น 2:1 เมื่อเทียบกับไมโครคอนโทรลเลอร์ขนาด 8 บิตอื่นๆ มีคำสั่งที่ง่ายต่อการใช้และง่ายต่อการจำจึงช่วยลดเวลาที่ใช้ในการพัฒนา

PIC16C5X เป็นชิปไมโครคอนโทรลเลอร์แบบพิเศษ โดยโครงสร้างของชิปจะช่วยลดราคาของระบบ,ลดความต้องการไฟเลี้ยง,วงจรรีเซ็ตกำลังไฟ(Power on Reset) และวงจร OST (Oscillator Start up-timer) ต้องการแค่วงจรรีเซ็ตภายนอกเชื่อมกับชิปเท่านั้น สามารถต่อวงจรออสซิลเลเตอร์ได้ 4 แบบ ซึ่งจะรวมถึงออสซิลเลเตอร์แบบกินไฟต่ำ และออสซิลเลเตอร์แบบ RC ที่มีราคาถูก มีโหมดสลีปซึ่งจะทำให้กินไฟน้อย มีตัว WDT (Watch-Dod Timer) และการป้องกันการก๊อปปี้รหัสคำสั่ง

เนื่องจาก PIC16C5X นั้น ใช้หน่วยความจำภายในเป็นแบบโปรแกรมได้ครั้งเดียวจึงสามารถทำให้ชิปตระกูลนี้มีราคาถูกและเหมาะสมอย่างยิ่งที่จะใช้ในการผลิตผลิตภัณฑ์จำนวนมาก ซึ่งสามารถนำเอาคุณประโยชน์จากส่วนป้องกันการก๊อปปี้มาใช้เพื่อป้องกันการลอกเลียนแบบซอร์สโค้ดที่อยู่ภายในได้อีกด้วย

2.3.2 คุณสมบัติทั่วๆไป

1. CPU ใช้รูปแบบการทำงานแบบ RISC

- มีคำสั่งที่ต้องเรียนรู้แค่ 33 คำสั่ง
- ทุกคำสั่งใช้เวลาปฏิบัติการ 1 ไซเคิล 200 นาโนวินาที ยกเว้นคำสั่งแบบบรานช์ต้องใช้เวลาถึง 2 ไซเคิล

- ความเร็วในการทำงาน : จากช่วงดีซี - ไซเคิลคำสั่งเท่ากับ 20 เมกะเฮิรตซ์ หรือ จากช่วงดีซี - ไซเคิลคำสั่งเท่ากับ 200 นาโนวินาที
- คำสั่งมีขนาดกว้าง 12 บิต
- บัสข้อมูลกว้าง 8 บิต
- หน่วยความจำเก็บโปรแกรมเป็นอีพรอมขนาด 12 บิต มีความจุ 512 ไบต์ ถึง 2 กิโลไบต์
- รีจิสเตอร์ใช้งานทั่วไปขนาด 8 บิต มีถึง 72 ตัว แล้วแต่เบอร์
- รีจิสเตอร์ฮาร์ดแวร์ที่ทำหน้าที่พิเศษ มีถึง 7 ตัว
- สแต็กฮาร์ดแวร์มีความลึก 2 ระดับ
- มีโหมดการอ้างอิงแอดเดรสของข้อมูลและคำสั่งแบบโดยตรง โดยอ้อม และแบบสัมพัทธ์

2. คุณสมบัติการต่อพ่วง

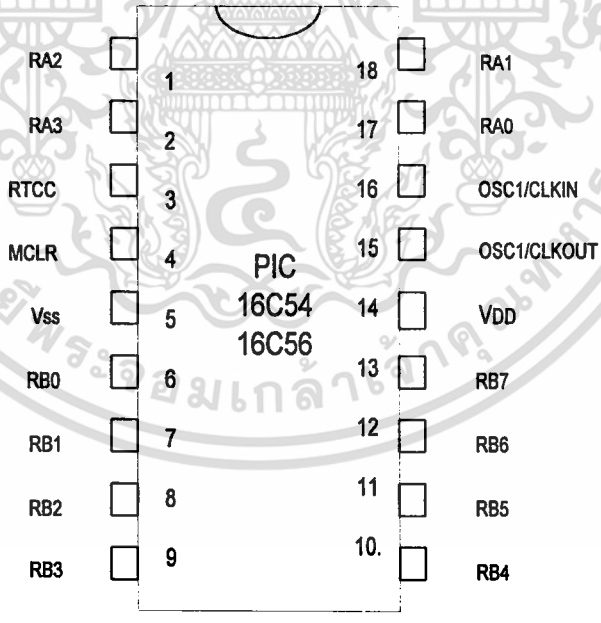
- การควบคุมทิศทางทำโดยการใช้ขา I/O ถึง 20 ขา
- มีตัวนับเวลา/สัญญาณเวลาขนาด 8บิต (RTCC) กับตัวตั้งเวลาที่โปรแกรมได้ขนาด 8 บิต
- มีตัวรีเซตกำลังไฟ (Power on Reset)
- มีตัว OST(Oscillair Start up-Timer)
- มีตัว WDT(Watch Dog Timer) กับออสซิลเลเตอร์แบบ RC เพื่อใช้ในการปฏิบัติงานที่เชื่อถือได้
- มีฟิวส์อีพรอมพิเศษเพื่อป้องกันการเลียนแบบซอร์สโค้ด
- มีโหมด SLEEP ลดการสูญเสียพลังงานเมื่อไม่ได้ใช้งาน
- มีออสซิลเลเตอร์ให้เลือกใช้เพื่อกำหนดไปยังอีพรอมแบบต่างๆ ดังนี้
  1) ออสซิลเลเตอร์แบบ RC ที่มีราคาถูก : RC
  2) คริสตอล/รีโซเนเตอร์มาตรฐาน : XT
  3) คริสตอล/รีโซเนเตอร์ความเร็วสูง : HS
  4) คริสตอลความถี่ต่ำ กินไฟน้อย : LP

3. เทคโนโลยี CMOS

- กินไฟต่ำ ใช้อีพรอมแบบซีมอสที่มีความเร็วสูง
- ออกแบบมาจากข้อมูลทางสถิติ
- ค่าศักดาไฟในช่วงที่ใช้งาน 2.5 โวลต์ ถึง 6.25 โวลต์
- กินไฟต่ำ คือน้อยกว่า 2 มิลลิแอมป์ที่แรงดัน 5 โวลต์ ความถี่ 4 เมกกะเฮิรตซ์
- น้อยกว่า 15 ไมโครแอมป์ แรงดัน 3 โวลต์ ความถี่ 32 กิโลเฮิรตซ์

2.3.3 การใช้งาน

ชิปตระกูล PIC16C5X จะเหมาะแก่การใช้งานตั้งแต่ระบบอัตโนมัติภายในรถยนต์จนถึงการ ควบคุมมอเตอร์ในระยะไกล หน่วยประมวลผลด้านการสื่อสารและอุปกรณ์ที่ใช้ชี้ตำแหน่ง เทคโนโลยีอีพรอมช่วยทำให้ผู้ใช้ทำการโปรแกรมได้อย่างรวดเร็วและสะดวก



รูปที่ 2.3.1 แสดงขนาดและตำแหน่งขาของไมโครคอนโทรลเลอร์ตระกูล PIC16C5X

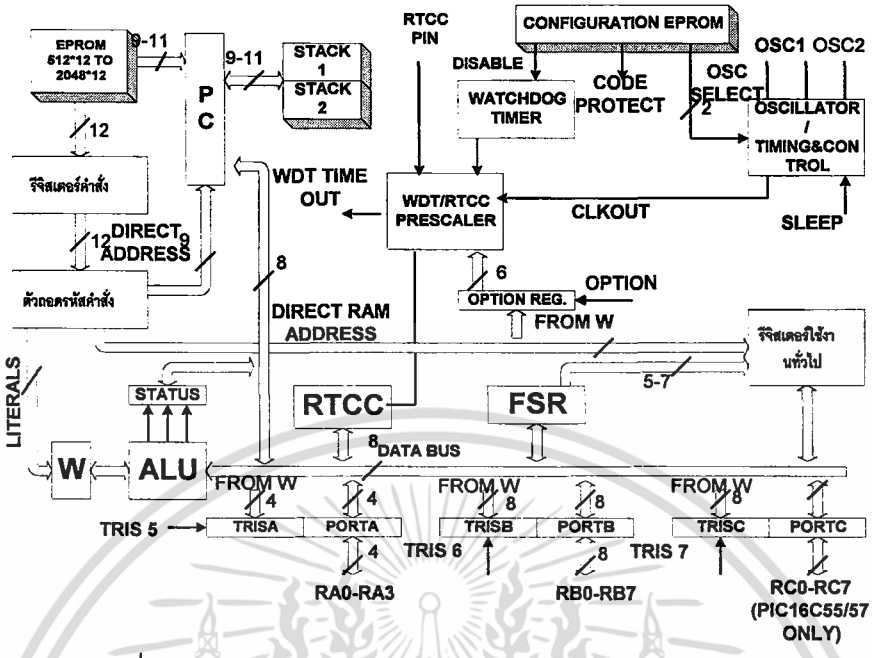
ตำแหน่งขา	*ฟังก์ชั่น*
RA0-RA3	อินพุต/เอาท์พุต พอร์ต A
RB0-RB7	อินพุต/เอาท์พุต พอร์ต B
RC0-RC7	อินพุต/เอาท์พุต พอร์ต C (เฉพาะ 16C57)
RTCC	รีลไทม์คล็อก/ เคาน์เตอร์
MCLR	มาสเตอร์เคลียร์
OSC1/CLKIN	ออสซิลเลเตอร์ (อินพุต)
OSC1/CLKOUT	ออสซิลเลเตอร์ (เอาท์พุต)
VDD	แหล่งจ่ายไฟ
VSS	กราวน์
N/C	ไม่ได้ต่อใช้งาน

ตารางที่ 2.3.1 แสดงหน้าที่ของขาต่างๆของ PIC16C5X

รูปที่ 2.3.1 แสดงขนาดและตำแหน่งขาของไมโครคอนโทรลเลอร์ตระกูล PIC16C5X ส่วนตารางที่ 2.3.1 แสดงหน้าที่ของขาต่างๆ

2.3.4 สถาปัตยกรรมภายใน

สถาปัตยกรรมของ PIC16C5X ใช้แนวความคิดแบบรีจิสเตอร์ไฟล์ ซึ่งจะมีการแยกบัสและหน่วยความจำสำหรับเก็บข้อมูลและเก็บคำสั่ง บัสข้อมูลและหน่วยความจำจะมีขนาดกว้าง 8 บิต ในขณะที่บัสโปรแกรมและหน่วยความจำเก็บโปรแกรมจะมีขนาดกว้าง 12 บิต แนวความคิดนี้ทำให้สามารถใช้ชุดคำสั่งเพียงชุดเดียวในการทำงานเป็นบิต ไบต์ และรีจิสเตอร์ด้วยความเร็วสูง โดยมีการเฟตช์และเอ็กซิคิวส์ คำสั่งแบบซ้อนทับกัน ซึ่งหมายความว่าในขณะที่คำสั่งหนึ่งกำลังถูกเอ็กซิคิวส์ คำสั่งต่อไปจะถูกอ่านจากหน่วยความจำเก็บโปรแกรมมาเก็บไว้ บล็อกไดอะแกรมของไมโครคอนโทรลเลอร์ PIC16C5X แสดงดังรูปที่ 2.3.2



รูปที่ 2.3.2 บล็อกไดอะแกรมของไมโครคอนโทรลเลอร์ PIC16C5X

ไทม์มิ่งไดอะแกรมแสดงลำดับการประมวลผลคำสั่ง

สัญญาณนาฬิกาอินพุต (ป้อนเข้าที่ขา OSC1) จะถูกแบ่งออกเป็น 4 ช่วง เพื่อสร้างเป็นสัญญาณนาฬิกาภายในที่ไม่ทับซ้อนกัน 4 ค่า เรียกว่า Q1 Q2 Q3 Q4 ภายในชิป ค่าโปรแกรมเคาน์เตอร์จะเพิ่มค่าที่เวลา Q1 คำสั่งจะถูกเฟตช์จากหน่วยความจำเก็บโปรแกรมและนำเข้าไปในรีจิสเตอร์คำสั่ง ที่แวลา Q4 มันจะถูกถอดรหัส และเอ็กซิคิวส์ที่เวลา Q1 ถึง Q4

รีจิสเตอร์เก็บข้อมูล

บัสข้อมูลขนาด 8 บิต จะเชื่อมต่อส่วนที่ทำหน้าที่พื้นฐานสองส่วนเข้าด้วยกันคือ รีจิสเตอร์ไฟล์ ที่ประกอบด้วยรีจิสเตอร์ 8 บิต จำนวน 80ตัว รวมถึงพอร์ตอินพุต/เอาท์พุตต่างๆ กับ ALU ขนาด 8 บิต แรมขนาด 32 ไบต์ จะถูกอ้างอิงแอดเดรสได้โดยตรงในขณะที่พื้นที่หน่วยความจำแบงค์ ชนาด 16 ไบต์ต่อหนึ่งแบงค์ จะถูกใช้เพื่อขยายหน่วยความจำให้มีขนาดใหญ่ขึ้นดังแสดงในรูปที่ 2..3.3 ข้อมูลสามารถถูกอ้างอิงตำแหน่งได้โดยตรงหรือโดยอ้อม โดยใช้รีจิสเตอร์ FSR ในทันทีที่มีการอ้างอิงตำแหน่งข้อมูล โดยใช้คำสั่งตามตัวอักษร ข้อมูลจะถูกโหลดจากหน่วยความจำโปรแกรมเข้าไปในรีจิสเตอร์ W

รีจิสเตอร์ ไฟล์แบ่งตามหน้าที่ได้เป็น 2 กลุ่มคือ รีจิสเตอร์ปฏิบัติการ (Operational register) กับ รีจิสเตอร์ใช้งานทั่วไป (General purpose register) รีจิสเตอร์ปฏิบัติ

การประกอบด้วย

1. Real time clock counter
2. Program counter
3. Status register
4. Input/output register
5. File select register
6. General purpose register

นอกจากนี้ รีจิสเตอร์ใช้งานพิเศษ (special purpose register) จะถูกใช้เพื่อควบคุมพอร์ตอินพุต/เอาท์พุต และใช้ในการคำนวณค่า ปรีสเกลเลอร์((prescaler) เพื่อกำหนดค่าการหาร

ALU (Arithmatic/Logic Unit)

ALU ขนาด 8 บิต จะมีรีจิสเตอร์ W ซึ่งใช้เก็บข้อมูลชั่วคราวรวมอยู่ด้วย ALU จะ ทำหน้าที่คำนวณทางคณิตศาสตร์ และ บูลีน ระหว่างข้อมูลที่อยู่ในรีจิสเตอร์ W กับรีจิสเตอร์ไฟล์ใดๆ นอกจากนี้ ALU ยังสามารถปฏิบัติการกับค่า โอเปอร์แรนต์ค่าเดียวที่อยู่ในรีจิสเตอร์ W หรือรีจิสเตอร์ไฟล์ใด ได้ด้วย

หน่วยความจำโปรแกรม (Program memory)

หน่วยความจำโปรแกรมนี้มีขนาด 512 เวิร์ด (1 เวิร์ด มีขนาด 12 บิต) เป็นอีพรอมสามารถอ้างอิงแอดเดรสได้โดยตรง ลำดับคำสั่งของคำสั่งขนาดเล็ก (Micro-instruction) จะถูกควบคุมโดยผ่านโปรแกรมเคาน์เตอร์ที่มีการเพิ่มค่าอย่างอัตโนมัติทุกครั้งที่มีการเอ็กซิคิวส์โปรแกรม ในการปฏิบัติการเพื่อควบคุมโปรแกรม การอ้างอิงแอดเดรสโดยตรงโดยอ้อมและการอ้างอิงแบบสัมพัทธ์ สามารถทำกับบิตทดสอบและคำสั่ง SKIP, CALL, JUMP หรือการคำนวณแอดเดรสเพื่อโหลดลงในโปรแกรมเคาน์เตอร์ นอกจากนี้ในชิพยังมีสแต็กถึง 2 ระดับ เพื่อให้ง่ายต่อการใช้โปรแกรมย่อยแบบโครงข่าย (nesting)

โปรแกรมเคาน์เตอร์

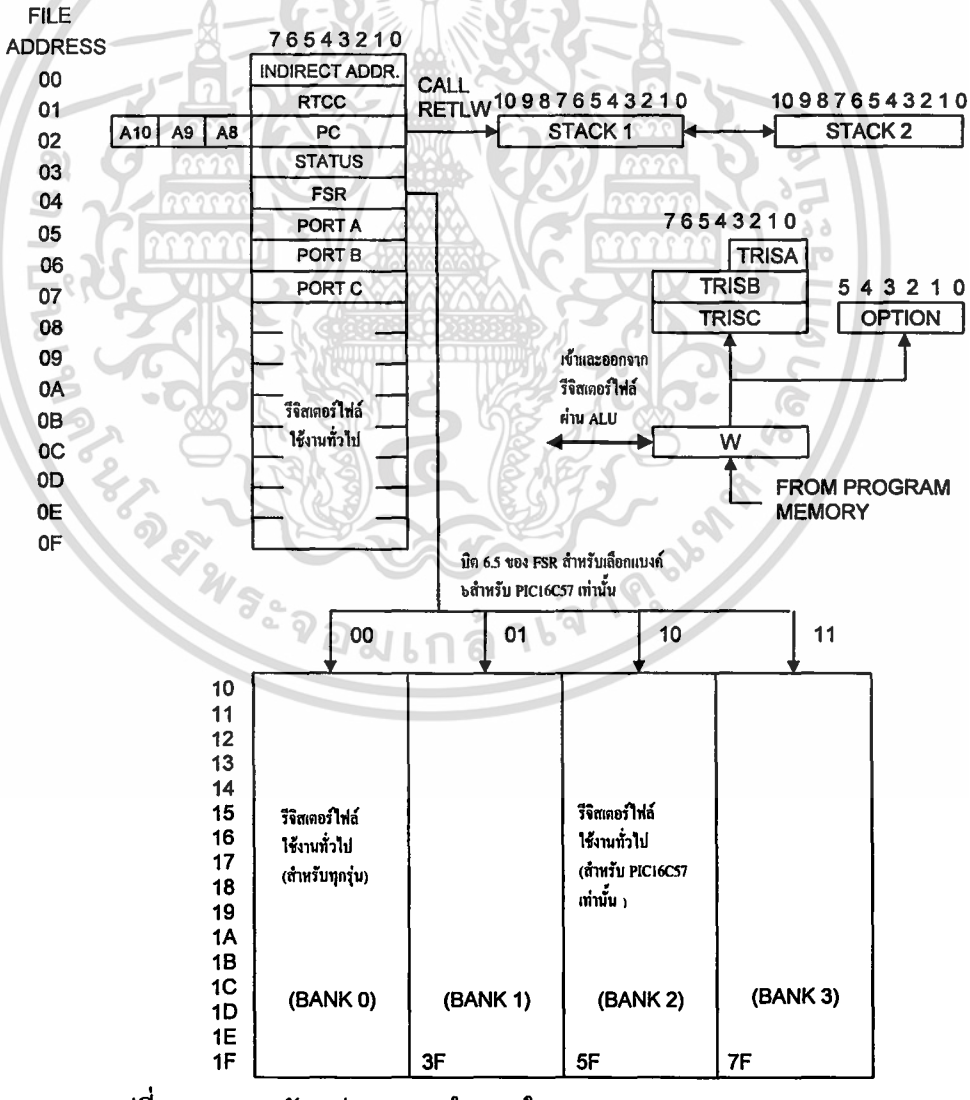
โปรแกรมเคาน์เตอร์จะสร้างแอดเดรสขนาดขนาด 12 บิตเพื่อใช้ชี้ตำแหน่งของเซลต่างๆในอีพรอม 2048 ตำแหน่งที่เก็บคำสั่งของโปรแกรม

ชิปต่างเบอร์กันจมีขนาดของโปรแกรมเคาน์เตอร์และขนาดของสแต็ก(ที่มี 2 ระดับ) แตกต่างกันออกไป

โปรแกรมเคาน์เตอร์จะเซ็ตทุกบิตเป็น 1 ถ้าอยู่ในสถานะรีเซ็ต ในช่วงการเอ็กซิคิวส์ โปรแกรมเคาน์เตอร์จะเพิ่มค่าตัวเองโดยอัตโนมัติในแต่ละคำสั่ง หรือมิฉะนั้น ผลลัพธ์จากการเอ็กซิคิวส์ จะเป็นตัวกำหนดค่าให้กับโปรแกรมเคาน์เตอร์

สแต็ก

ชิปตระกูล PIC16C5X มีสแต็กทางฮาร์ดแวร์ 2 ระดับ ซึ่งสามารถใช้สำหรับเก็บค่าและอ่านค่า (Push/Pop) ข้อมูลได้ ดังรูปที่ 2.3.4



รูปที่ 2.3.3 การจัดหน่วยความจำภายในของ PIC16C5X

คำสั่ง CALL จะดันค่าโปรแกรมเคาน์เตอร์ในขณะนั้น (เพิ่มค่าขึ้น1) เก็บในสแต็กระดับ 1 และสแต็กระดับ 1 จะดันค่าเข้าสแต็กระดับ 2 โดยอัตโนมัติ ถ้ามีการใช้คำสั่ง CALL มากกว่า 2 ครั้ง ค่าที่ได้เก็บในสแต็กล่าสุด 2 ค่าเท่านั้นที่ยังเก็บอยู่

<u>รีจิสเตอร์สถานะ</u>

รีจิสเตอร์นี้จะเก็บค่าสถานะการคำนวณของ ALU ,สถานะการรีเซ็ต และค่าบิตเลือกเพจล่วงหน้าสำหรับหน่วยความจำเก็บโปรแกรมที่ใหญ่กว่า 512 เวิร์ด (PIC16C56, PIC16C57)

บิตตัวทด/ตัวยืม และบิตตัวทด/ตัวยืมแบบดิจิต

บิตตัวทดเป็นค่าตัวทดจากผลการบวก (ADDWF) และเป็นค่าตัวยืม (borrow) จากผลการลบ (SUBWF) คำสั่ง RRF และ RLF มีผลต่อบิตนี้ด้วย บิตตัวทดแบบดิจิตจะมีหน้าที่คล้ายบิตตัวทด เช่น บิตนี้จะใช้เป็นค่าตัวยืมในการลบแบบดิจิต

บิตไทม์เอาท์และบิตแสดงสถานะเพาเวอร์ดาวน์ (T0,PD)

บิต T0 (time out) และ PD (power down) ในรีจิสเตอร์สถานะสามารถใช้ทดสอบเพื่อหาว่าเงื่อนไขการรีเซ็ตที่เกิดขึ้นเป็นเหตุมาจากไทม์เอาท์ที่ WDT ,เงื่อนไขการเปิดไฟเลี้ยง หรือการตื่นจากโหมด SLEEP โดยใช้ตัว WDT หรือการใช้ขา MCLR เหตุการณ์ที่มีผลต่อบิตสถานะเหล่านี้แสดงในตาราง

โหมด	*T0*	*PD*	*หมายเหตุ*
เพาเวอร์อัพ	1	1	ไม่มีผลต่อบิต PD
เกิดไทม์เอาท์จาก WDT	0	x	
คำสั่ง SLEEP	1	0	
คำสั่ง CLR WDT	1	1	

ตารางที่ 2.3.2 การกำหนดค่าบิตไทม์เอาท์และบิตแสดงสถานะเพาเวอร์ดาวน์

รีจิสเตอร์ FSR

บิตที่ 0-4 จะเลือกรีจิสเตอร์ไฟล์ตัวใดตัวหนึ่งจาก 32 ตัว ที่มีในการอ้างอิงแอดเดรสโดยอ้อม บิตที่ 5-7 ของ SFR จะอ่านได้อย่างเดียว และโดยทั่วไปจะอ่านค่าได้เท่ากับ "1"

ถ้าไม่ใช่การอ้างอิงโดยอ้อม รีจิสเตอร์ SFR นี้จะสามารถใช้เป็นรีจิสเตอร์จุดประสงค์ทั่วไปขนาด 5 บิต

รีจิสเตอร์ พอร์ตอินพุต/เอาท์พุต

รีจิสเตอร์ พอร์ตอินพุต/เอาท์พุตสามารถเขียนและอ่านได้ภายใต้การควบคุมของโปรแกรมเหมือนกับรีจิสเตอร์ไฟล์ อย่างไรก็ตาม คำสั่งอ่าน เช่น MOVE PORTB ,W โดยปรกติจะอ่านจากขาอินพุต/เอาท์พุต โดยไม่สนใจว่าขานั้นจะกำหนดให้เป็นขาอินพุต/เอาท์พุต เมื่อเริ่มต้นรีเซ็ตเครื่องใหม่ พอร์ตทั้งหมดจะถูกกำหนดให้เป็นอินพุต (โหมดความต้านทานสูง) และรีจิสเตอร์อินพุต/เอาท์พุตจะถูกเซ็ตค่าเป็น "1" ทั้งหมด

การอินเตอร์เฟสกับพอร์ตอินพุต/เอาท์พุต

พอร์ตทุกพอร์ตสามารถจะกำหนดได้ว่าจะให้เป็นอินพุตหรือเอาท์พุต สำหรับการใช้งานเป็นพอร์ตอินพุต สัญญาณจะไม่ถูกแลตช์เอาไว้ การป้อนสัญญาณอินพุตจะต้องคงอยู่จนกระทั่งถูกอ่านโดยคำสั่งอินพุตจนครบ แต่สำหรับการใช้งานเป็นพอร์ตเอาท์พุตนั้น ข้อมูลจะถูกแลตช์เอาไว้ และจะไม่มีการเปลี่ยนค่าจนกว่าจะมีการเขียนข้อมูลใหม่ส่งมาอีกครั้ง การกำหนดพอร์ตในตำแหน่งขาต่างๆ ให้เป็นเอาท์พุตจะต้องสั่งให้บิตควบคุมทิศทาง (TRISA, TRISB, TRISC)มีค่าเป็น "0" แต่ถ้าต้องการให้เป็นขาอินพุต จะต้องเซ็ตบิตควบคุมให้เป็น "1"

รีจิสเตอร์ใช้งานทั่วไป

- f08h-f1fh : รีจิสเตอร์ใช้งานทั่วไปในหน่วยความจำแบงค์ 0
- f20h-f2fh : ทางกายภาพจะเหมือนกับ f00h-f0fh
- f30h-f3fh : รีจิสเตอร์ใช้งานทั่วไปในหน่วยความจำแบงค์ 1
- f40h-f4fh : ทางกายภาพจะเหมือนกับ f00h-f0fh
- f50h-f5fh : รีจิสเตอร์ใช้งานทั่วไปในหน่วยความจำแบงค์ 2

- f60h-f6fh : ทางกายภาพจะเหมือนกับ f00h-f0fh
- f70h-f7fh : รีจิสเตอร์ใช้งานทั่วไปในหน่วยความจำแบงค์ 3

รีจิสเตอร์หน้าที่พิเศษ

รีจิสเตอร์ W (Working Register) ใช้สำหรับเก็บค่าโอเปอร์แรนต์ที่สองในคำสั่งที่มีสองโอเปอร์แรนต์หรือสนับสนุนการเคลื่อนย้ายภายใน

TRISA เป็นรีจิสเตอร์ควบคุมอินพุต/เอาท์พุตสำหรับพอร์ต A บิต 0-3 เท่านั้นที่ใช้ได้ เพื่อให้สอดคล้องกับพอร์ตอินพุต/เอาท์พุตที่มีขนาดแค่ 4 บิต

TRISB เป็นรีจิสเตอร์ควบคุมอินพุต/เอาท์พุตสำหรับพอร์ต B

TRISC เป็นรีจิสเตอร์ควบคุมอินพุต/เอาท์พุตสำหรับพอร์ต C

รีจิสเตอร์ควบคุมอินพุต/เอาท์พุตจะถูกโหลดด้วยค่าในรีจิสเตอร์ W เมื่อมีการทำคำสั่ง TRISf โดยค่า "1" ที่ใส่ในรีจิสเตอร์ควบคุมจะทำให้ขาอินพุต/เอาท์พุต อยู่ในสถานะความต้านทานสูง ทำหน้าที่เป็นขาอินพุต และค่า "0" ที่ใส่ในรีจิสเตอร์ควบคุมจะทำให้ขาอินพุต/เอาท์พุตที่ถูกเลือกในพอร์ตนั้นมีสถานะเป็นขาเอาท์พุต รีจิสเตอร์เหล่านี้จะเขียนได้อย่างเดียวและจะถูกเซ็ตให้เป็น "1" ทั้งหมดเมื่ออยู่ในสภาวะรีเซ็ต

สภาวะรีเซ็ต

สภาวะรีเซ็ตอาจเกิดจากการที่เริ่มป้อนไฟเลี้ยงให้แกชิป, ป้อนค่าลอจิก "0"เข้าที่ขา MCLR หรือเกิดจากตัว WDT เกิอไทม์เอาท์ ชิปจะอยู่ในสภาวะรีเซ็ตตราบเท่าที่ขา MCLR ยังคงมีค่าลอจิก "0" หรือตัว OST แอกทีฟ

ตัว OST จะแอกทีฟทันทีที่ขา MCLR มีค่าลอจิกเป็น "1" ในสภาวะรีเซ็ต PIC16C5X จะมีสถานะดังนี้

- ออสซิลเลเตอร์จะทำงานหรือเริ่มต้นทำงาน
- ขาของพอร์ตอินพุต/เอาท์พุตทุกขาจะอยู่ในสถานะความต้านทานสูง โดยจะรีเซ็ตให้รีจิสเตอร์ TRIS มีค่าเป็น "1"ทั้งหมด
- โปรแกรมเคาน์เตอร์จะถูกเซ็ตให้เป็น "1" ทั้งหมด
- รีจิเตอร์ออปชั่นจะถูกเซ็ตให้เป็น "1" ทั้งหมด
- ตัว WDT และปรีสเกลเลอร์จะถูกเคลียร์
- 3 บิตบนในรีจิเตอร์สถานะจะเซ็ตเป็น "1" ทั้งหมด

- เฉพาะรีจิสเตอร์ที่ใช้ออสซิลเลเตอร์แบบ RC สัญญาณ CLKOUT ที่ขา OSC2 จะมีค่าเป็นลอจิก “0”

บทที่ 3
หลักการทำงานและการออกแบบ

3.1 คุณสมบัติของเครื่องควบคุมระยะไกลผ่านสายโทรศัพท์

- สามารถควบคุมอุปกรณ์ไฟฟ้าปลายทางได้ โดยที่ผู้ควบคุมนั่งอยู่ที่โต๊ะทำงาน
- ชุดควบคุมเป็นอุปกรณ์ที่ เล็กกระทัดรัด สามารถพกพาไปติดตั้งเองได้ไม่ยุ่งยาก
- สามารถควบคุมการ เปิด/ปิด อุปกรณ์ปลายทางได้โดยตรง หรือ เลือกจากโหมดจากการตั้งเวลา
- อุปกรณ์ไฟฟ้าที่นำมาต่อร่วมนั้น ต้องเป็นอุปกรณ์ไฟฟ้าที่ใช้กระแสสลับ 220 โวลต์
- เครื่องควบคุมระยะไกลติดตั้งที่ใดก็ได้ ภายใต้เครือข่ายโทรศัพท์

3.2 หลักการทำงานของเครื่องควบคุมระยะไกลผ่านสายโทรศัพท์

เครื่องควบคุมระยะไกลผ่านสายโทรศัพท์ ได้ออกแบบขึ้นโดยอาศัยโครงข่ายโทรศัพท์ขององค์การโทรศัพท์เป็นสื่อกลางที่สำคัญ การเชื่อมต่อระบบควบคุมระยะไกลผ่านสายโทรศัพท์ เข้ากับโมเด็ม ความเร็ว 2400 bps เชื่อมต่อกับเครือข่ายโทรศัพท์ เพื่อรับส่งข้อมูลคำสั่งจาก PC ไปยังชุดควบคุมปลายทางระยะไกล

โปรแกรมที่เขียนขึ้นควบคุมการทำงานบน PC เลือกใช้โปรแกรม Microsoft Visual Basic for Windows เหตุผลที่เลือกโปรแกรมนี้คือ Viaual Basic ถือว่าเป็นความก้าวหน้าที่สำคัญที่เกี่ยวข้องกับการเขียนแอพพลิเคชั่นที่ทำงานบน Windows ใน Visual Basic ประกอบไปด้วย เอนจิน (Engine) ที่ทำงานด้วยอีเวนต์และเครื่องมือช่วยออกแบบที่ใช้งานได้ง่าย มีการแสดงหน้าจอทำงานที่สวยงาม และที่สำคัญโปรแกรม Visdual Basic รองรับการสื่อสารการส่งข้อมูลผาน Port อนุกรม เพื่อติดต่อกับ Hardware ภายนอกได้โดยง่าย

โปรแกรมจะถูกสั่งงานจากผู้ใช้ โดย ในขั้นตอนแรก จะสั่งงานให้ Modem ต้นทาง โทรติดต่อกับ Modem ปลายทาง รอสัญญาณตอบรับ ว่า สามารถโทรติดแล้ว

ผู้ควบคุมเลือกคลิกปุ่มที่หน้าจอว่าต้องการ ปิด/เปิด อุปกรณ์ปลายทางตัวใด เมื่อมีการคลิกปุ่มสั่งงาน ข้อมูลจะถูกส่งผ่านสายโทรศัพท์ไปยังปลายทางด้วย Modem มี

ความเร็วในการส่งข้อมูลเท่ากับ 2,400 bps ส่งข้อมูลแบบ Asynchronous

เมื่อข้อมูล ถูกส่งไปยังชุดบอร์ดควบคุมก็จะมีการรับค่าข้อมูลมาตรวจสอบทางขา 6 Chip Pic 16C56 เป็น Microcontroller Basic Stamp ตรวจสอบด้วยโปรแกรมที่เขียนขึ้นแล้วบันทึกไว้ใน EEPROM เบอร์ 93LC56 โปรแกรมจะรอรับค่าว่า Output ตรงกับ Port ใด อุปกรณ์ไฟฟ้าที่ต่ออยู่กับ Port นั้นก็จะทำงาน

สำหรับโปรแกรมควบคุมสามารถสั่งงานได้ 2 ลักษณะ คือ

- เลือกควบคุมปิด/เปิดเอง โดยตรง
- ตั้งเวลาการปิด หรือ เปิด อุปกรณ์ไฟฟ้าโดยอัตโนมัติ

การออกแบบชุดควบคุมระยะไกลผ่านสายโทรศัพท์ อาศัยหลักการการรับส่งข้อมูลผ่านเครือข่ายโทรศัพท์ของ Modem เป็นหัวใจสำคัญ

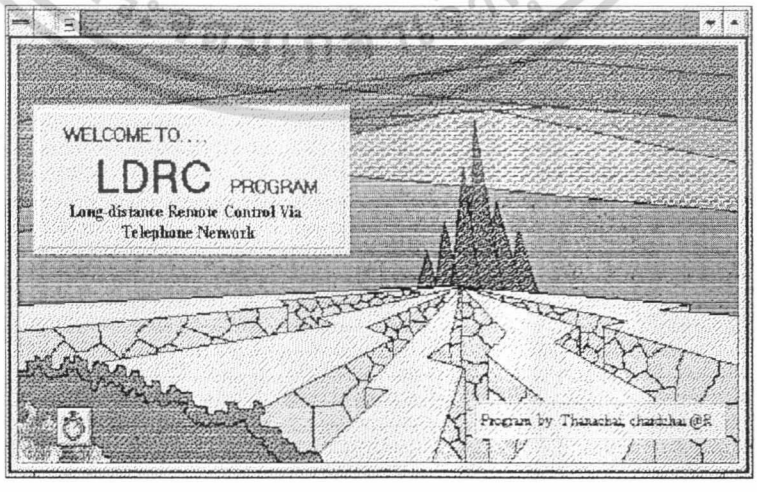
3.3 การออกแบบ

3.3.1โปรแกรม Visual Basic

โปรแกรม LDRC ถูกออกแบบมาเพื่อควบคุมอุปกรณ์ไฟฟ้าปลายทางระยะไกลถูกแบ่งออกเป็น 2 ส่วน

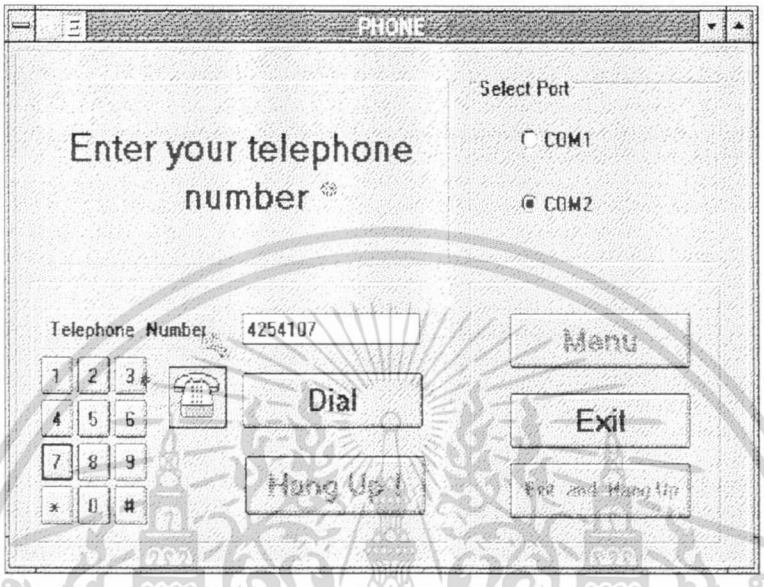
1. ส่วนของการเชื่อมต่อระหว่างโมเด็ม (การโทรติดต่อ)
2. ส่วนของขั้นตอนการเลือกสั่งงาน

เมื่อผู้ใช้เปิดโปรแกรม LDRC ขึ้นมา โปรแกรมจะทำการโหลดภาพ ซึ่งเป็นส่วนแสดงการนำเข้าสู่โปรแกรมนี้ ปรากฏดังรูปที่ 3.1 โดยจะมีระยะเวลาประมาณ 7 วินาที และจะไปยังส่วนของการโทรออก



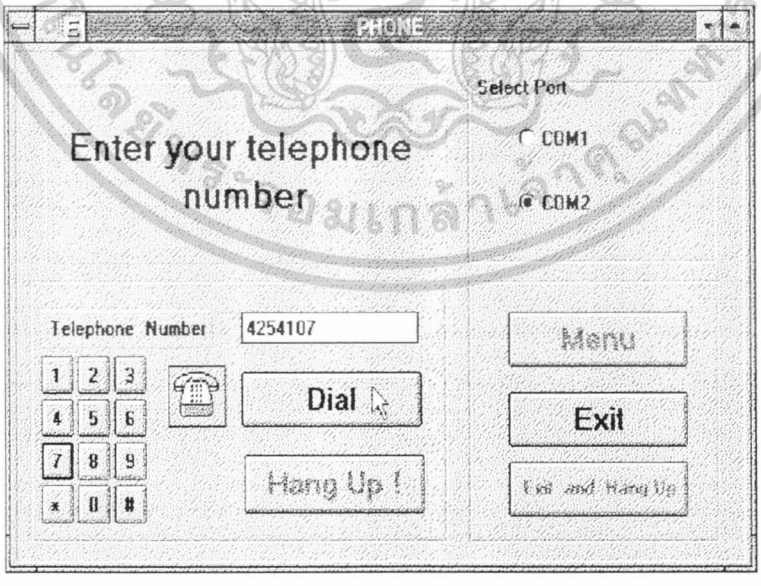
รูปที่ 3.1 แสดงหน้าจอเริ่มต้นของโปรแกรม

หลังจากนั้นโปรแกรมก็จะแสดง ส่วนของการติดต่อระหว่างโมเด็ม ซึ่งก็คือการให้โมเด็มทั้งสองตัวสื่อสารกัน ปรากฏดังภาพที่ 2



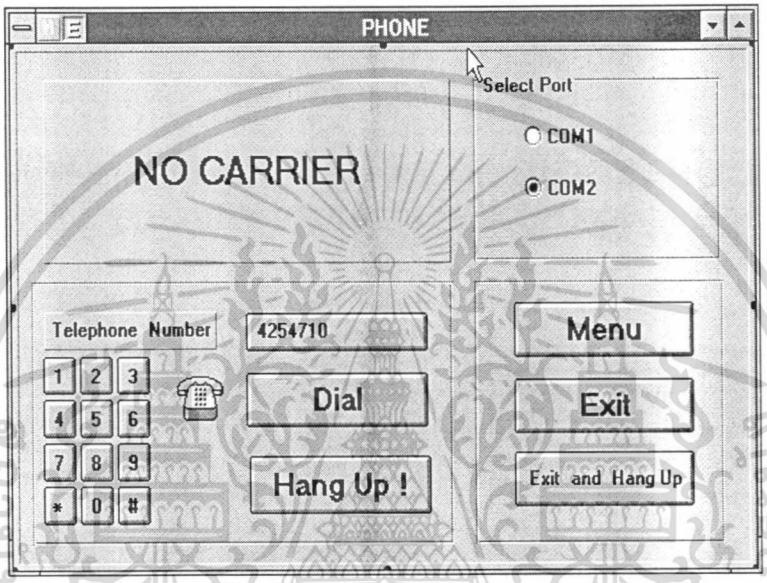
รูปที่ 3.2 แสดงการใส่หมายเลขโทรออก

โปรแกรมจะรอรับหมายเลขโทรศัพท์ปลายทางที่เราต้องการติดต่อสั่งงาน และยังมีการแสดง Option ให้ผู้ใช้เลือกว่าต้องการติดต่อโมเด็มผ่านพอร์ตอนุกรม 1 หรือพอร์ตอนุกรม 2



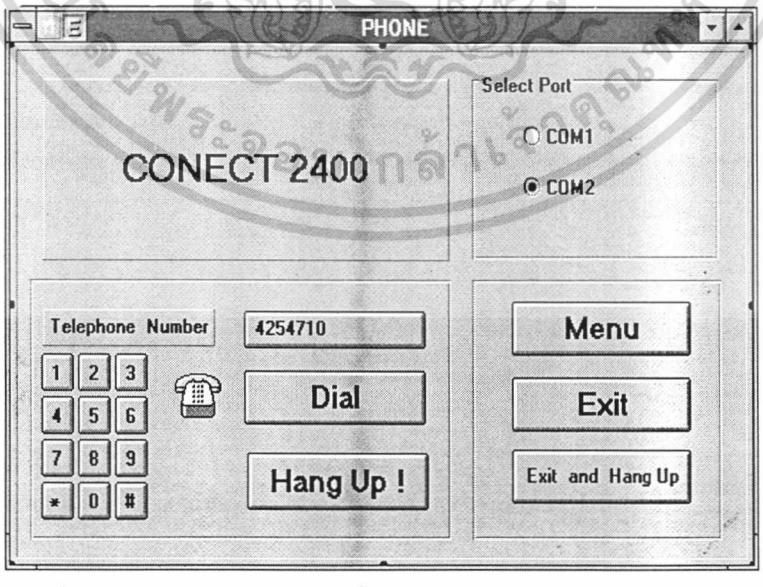
รูปที่ 3.3 แสดงการหมุนโทรศัพท์ไปยังคู่สายปลายทาง

เมื่อเราใส่หมายเลขโทรศัพท์เรียบร้อยแล้วจึงลากเมาส์แล้วคลิกที่ปุ่ม Dial ก็จะเป็นการสั่งให้โมเด็มนั้นติดต่อไปยังปลายทาง โปรแกรมจะทำการสั่งงานให้โมเด็มต้นทางโทรติดต่อกับโมเด็มปลายทาง โดยผู้ใช้ต้องรอสัญญาณการรับสาย 3 ครั้ง หลังจากนั้นโปรแกรมก็จะแสดงผลของการติดต่อว่าติดต่อได้หรือไม่ ถ้าคู่สายปลายทางไม่ว่าง แสดงว่าติดต่อไม่ได้โปรแกรมจะแสดงคำว่า No Carrier ดังรูปที่ 3.4 ดังนั้นผู้ใช้ต้องทำการติดต่ออีกครั้งหนึ่ง



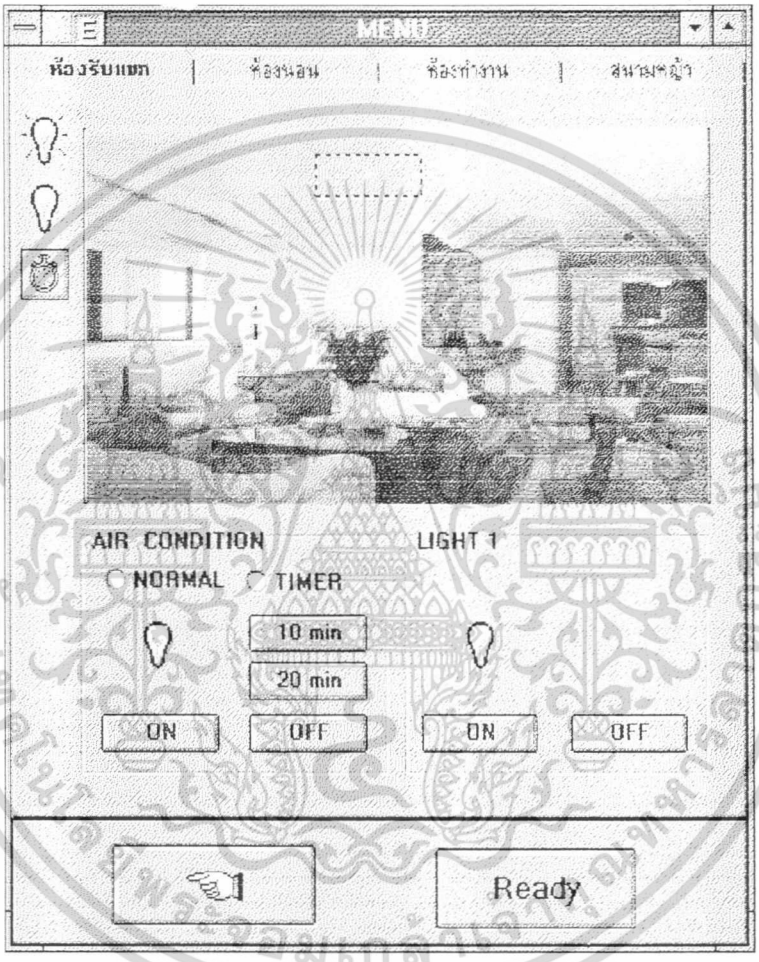
รูปที่ 3.4 แสดงผลการโทรไม่ติด หรือคู่สายปลายทางไม่ว่าง

ถ้าโทรติด โมเด็มต้นทางก็จะมีไฟสว่างที่ CD ปรากฏบนเครื่องโมเด็มต้นทาง และจะมีการแสดงผลที่หน้าจอด้วยคำว่า Connect 2400 ดังรูปที่ 3.5



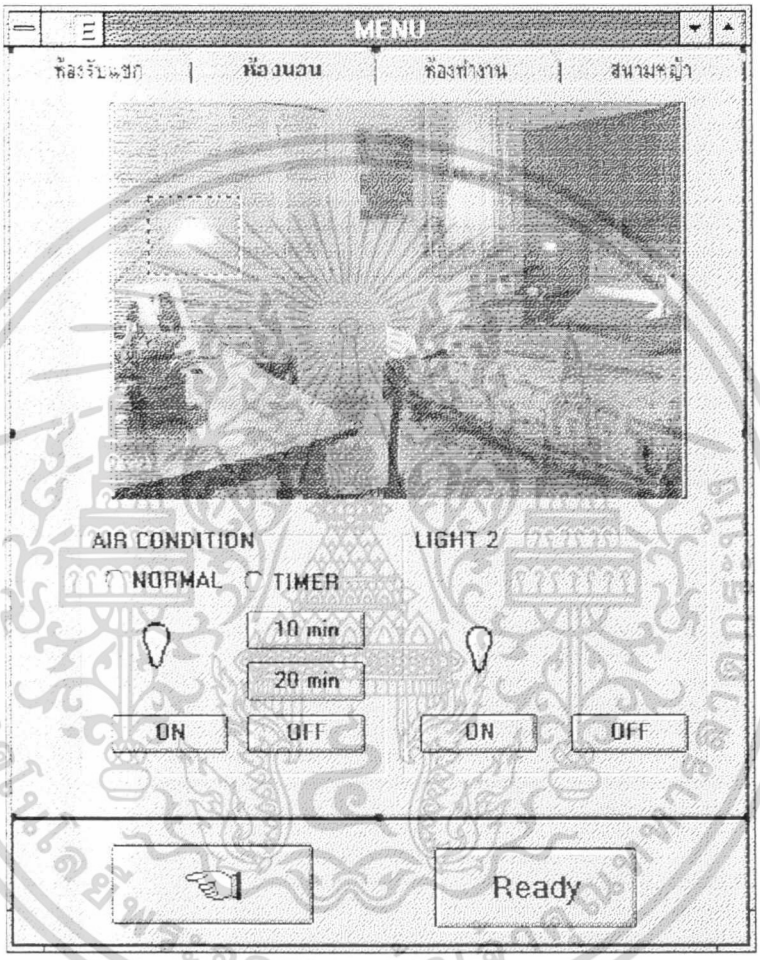
รูปที่ 3.5 แสดงผลการติดต่อ เมื่อโทรติดหรือ คู่สายปลายทางว่าง

คลิกเมาส์ไปที่ปุ่ม MENU ก็จะเป็นการสั่งให้โปรแกรมโหลดส่วนของเมนูให้เลือกสั่งงานขึ้นมา โดยจะมีรูปแบบของแต่ละส่วนดังนี้คือ

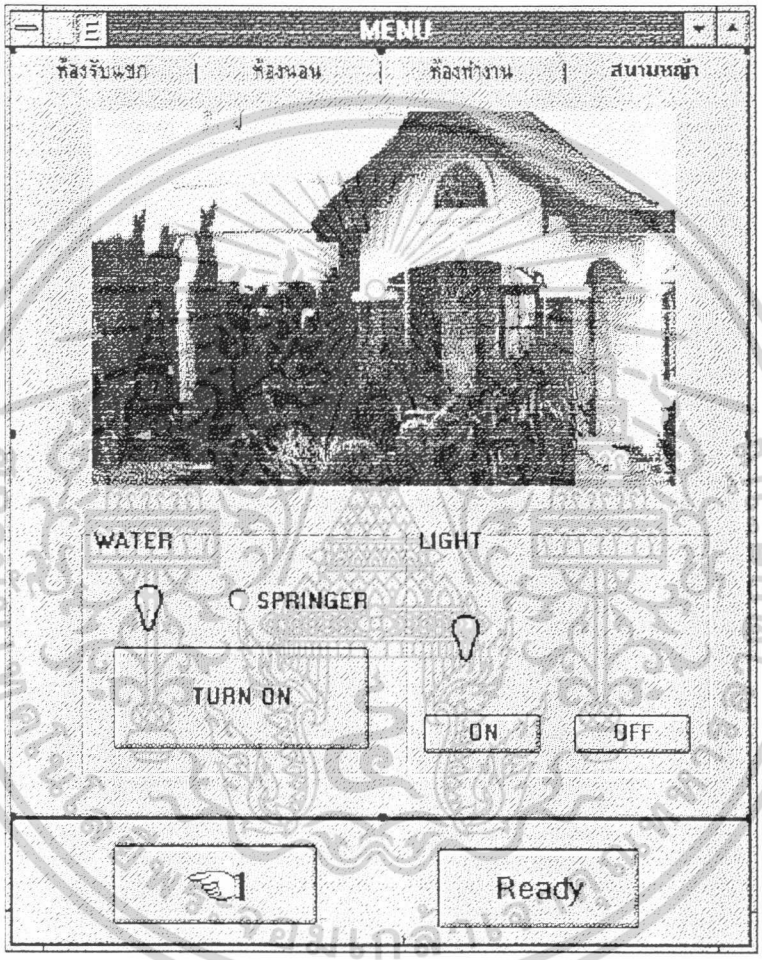


รูปที่ 3.6 แสดงหน้าจอของห้องรับแขก

ผู้ใช้สามารถสั่งงานจากปุ่ม ON และ OFF ทั้งในโหมดของการปิด - เปิด การทำงานหรือการตั้งเวลา

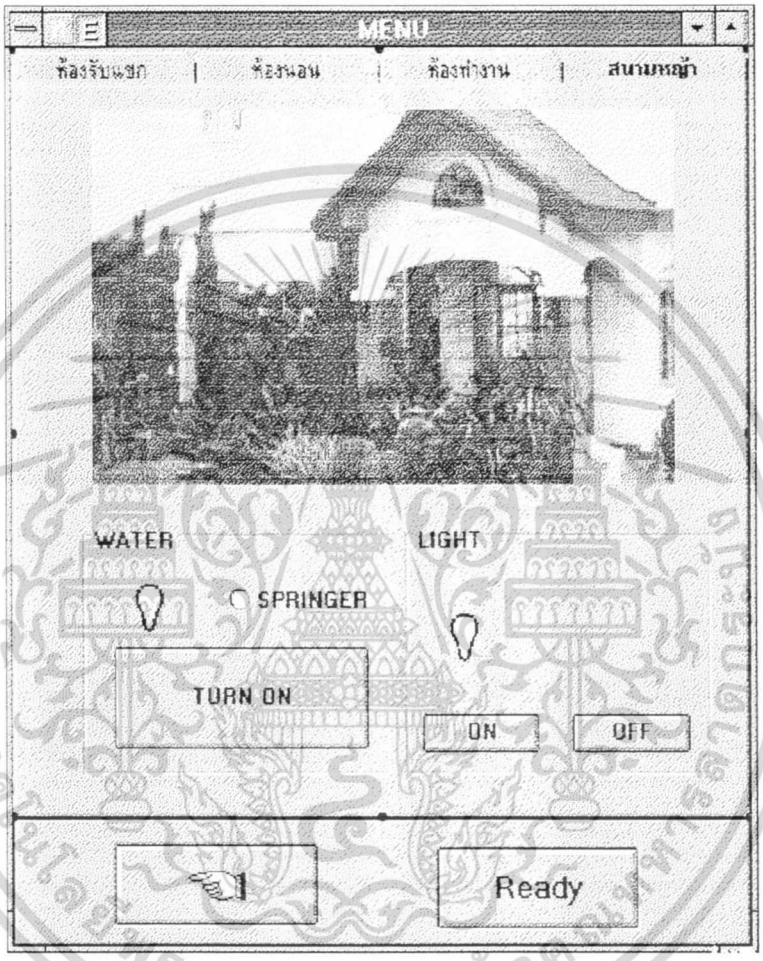


รูปที่ 3.7 แสดงหน้าจอห้องนอน



รูปที่ 3.8 แสดงหน้าจอของห้องทำงาน

ผู้ใช้สามารถควบคุมการปิดเปิดแอร์จากปุ่ม ON และ OFF หรือเลือกคลิกจากรูปแอร์ได้เลย



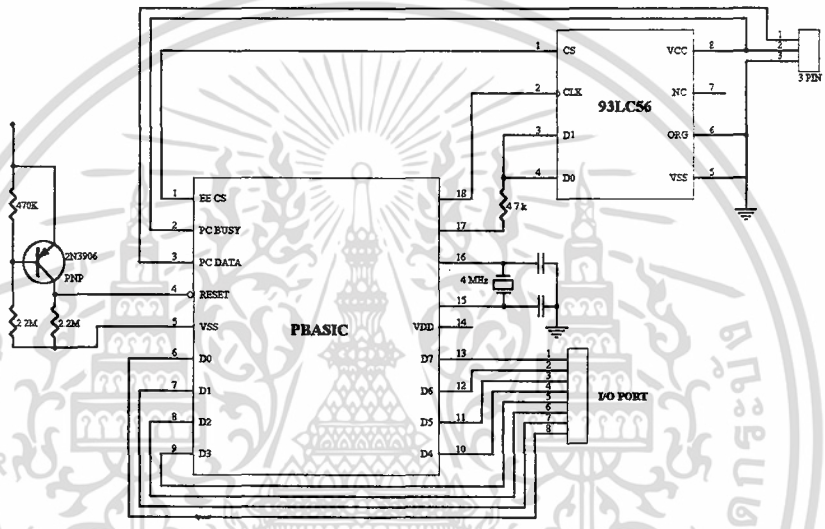
รูปที่ 3.9 แสดงหน้าจอของสนามหญ้า

ผู้ใช้สามารถควบคุมการทำงานของเครื่องรดน้ำต้นไม้ โดยเลือกคลิกที่ปุ่ม Turn on และสามารถควบคุมการปิดเปิด ไฟสนามได้จากปุ่ม ON และ OFF

เมื่อผู้ใช้ต้องการออกจากโปรแกรม ให้คลิกกลับไปยังเมนูแรก และสามารถเลือกออกจากโปรแกรมได้ 2 วิธีคือ

1. ออกจากโปรแกรมพร้อมกับวางหูโทรศัพท์
2. ออกจากโปรแกรม โดยไม่วางหูโทรศัพท์

3.3.2 วงจร Basic Stamp

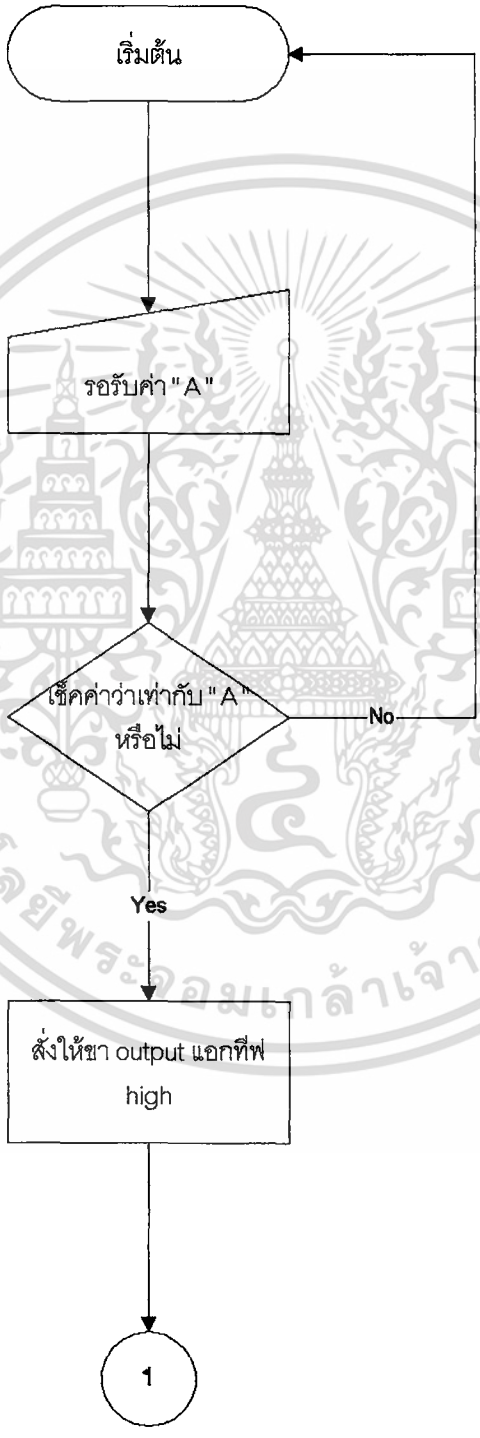


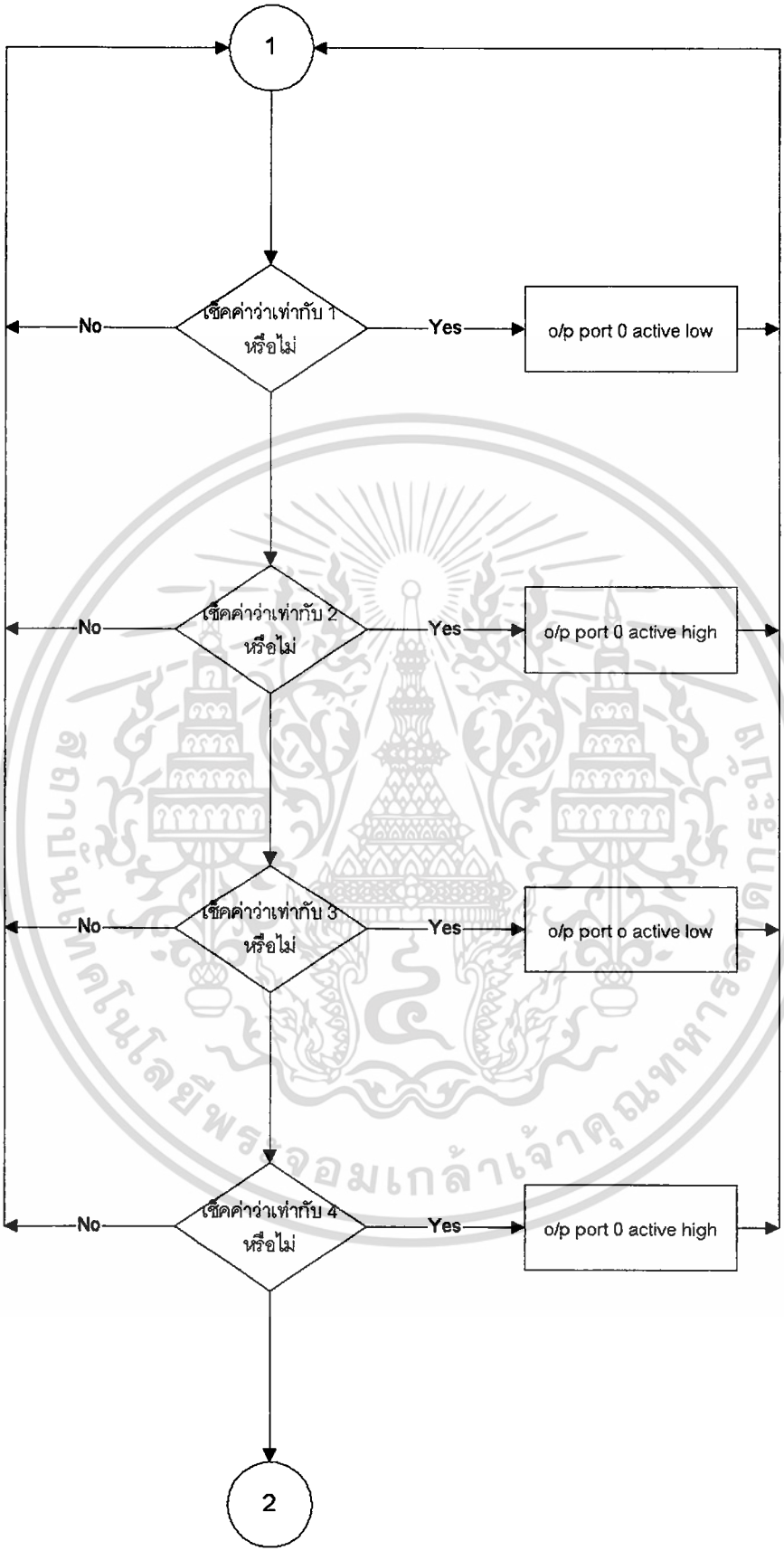
รูปที่ 3.10 แสดงวงจร Basic Stamp

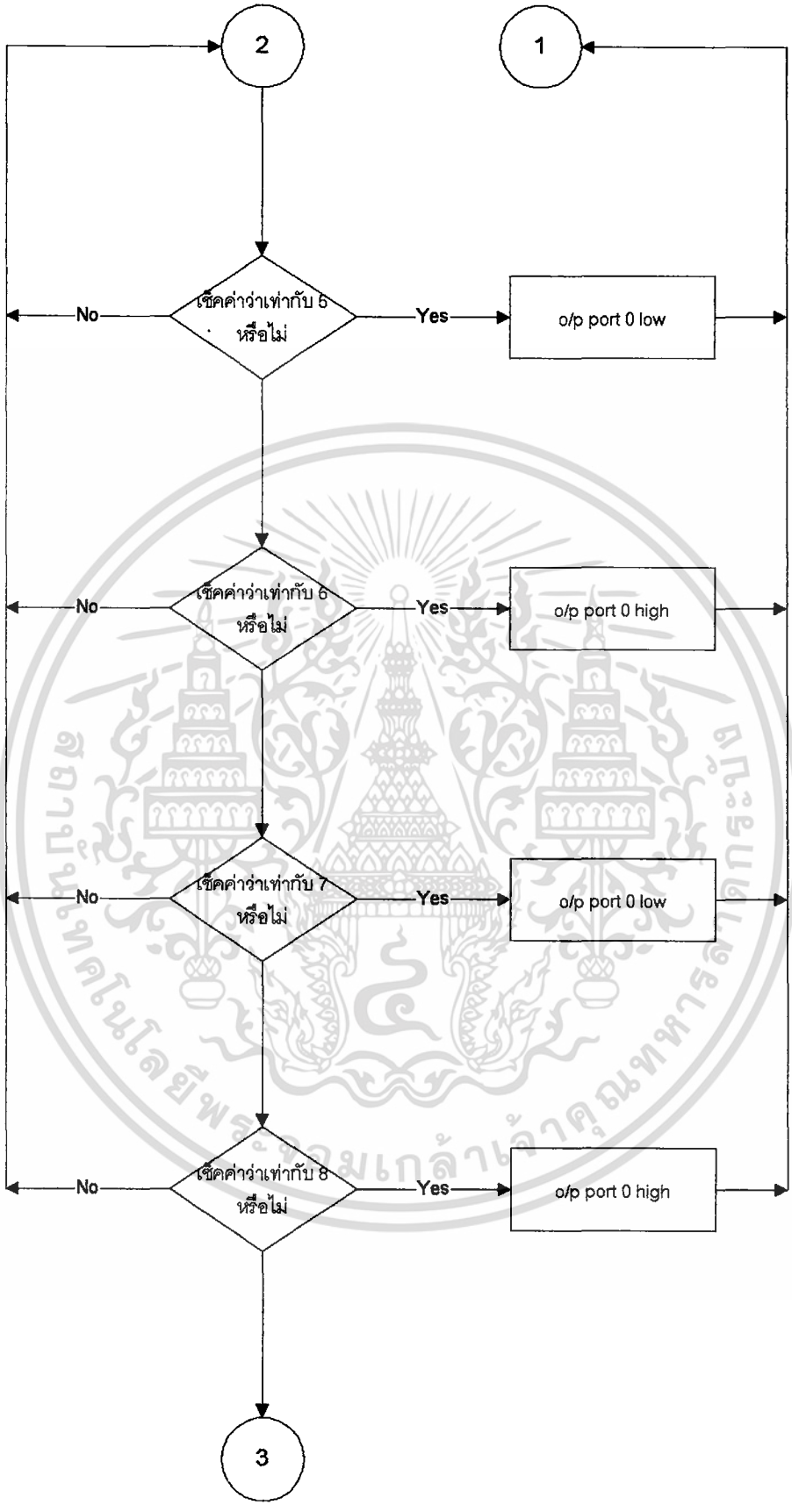
เราเลือกใช้ I/O Port ของ Basic Stamp ในการควบคุมวงจรDriver ที่ต่ออยู่กับอุปกรณ์ไฟฟ้า 220 Vac

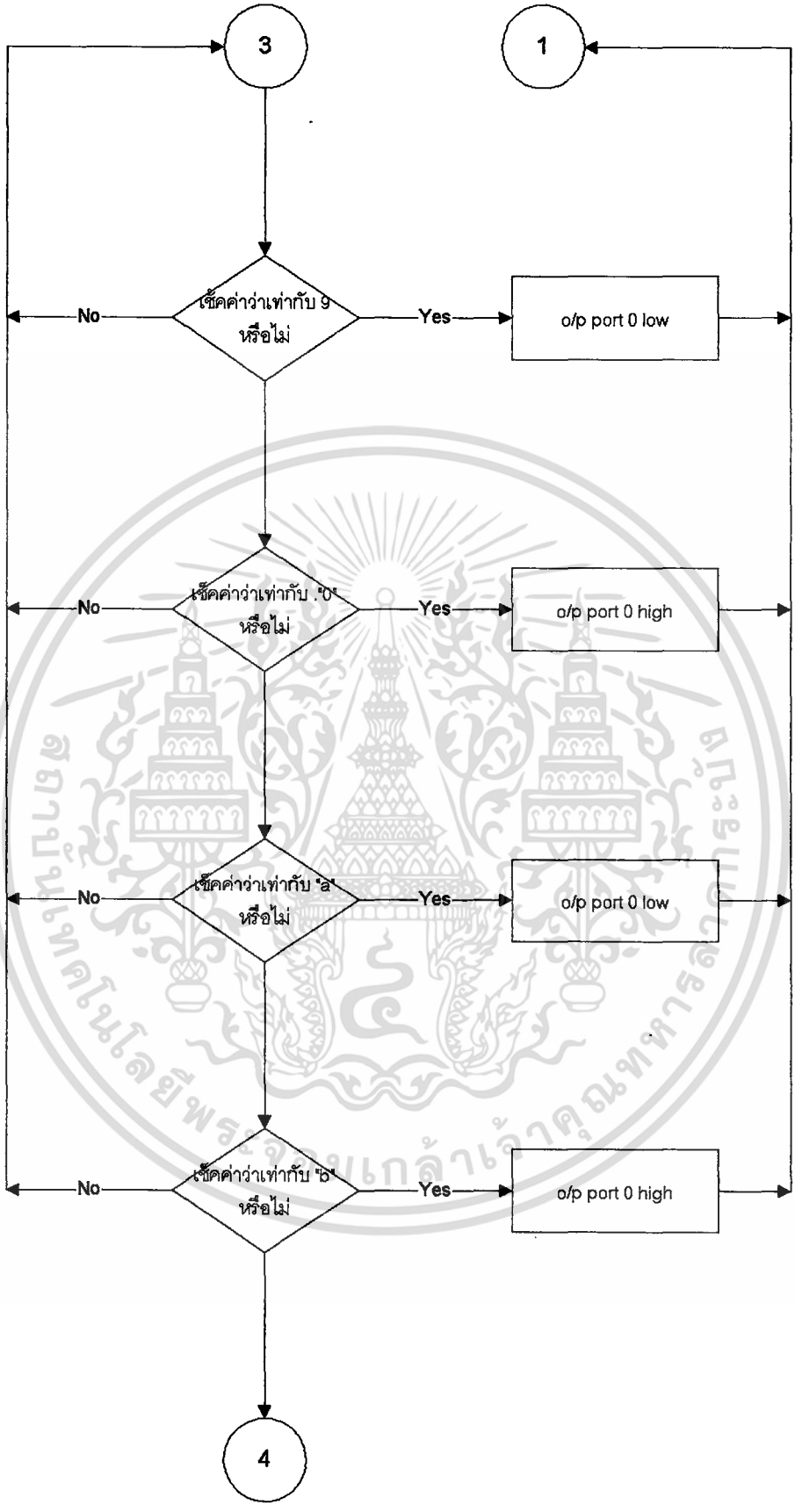
3.3.3 โปรแกรมควบคุม Basic Stamp

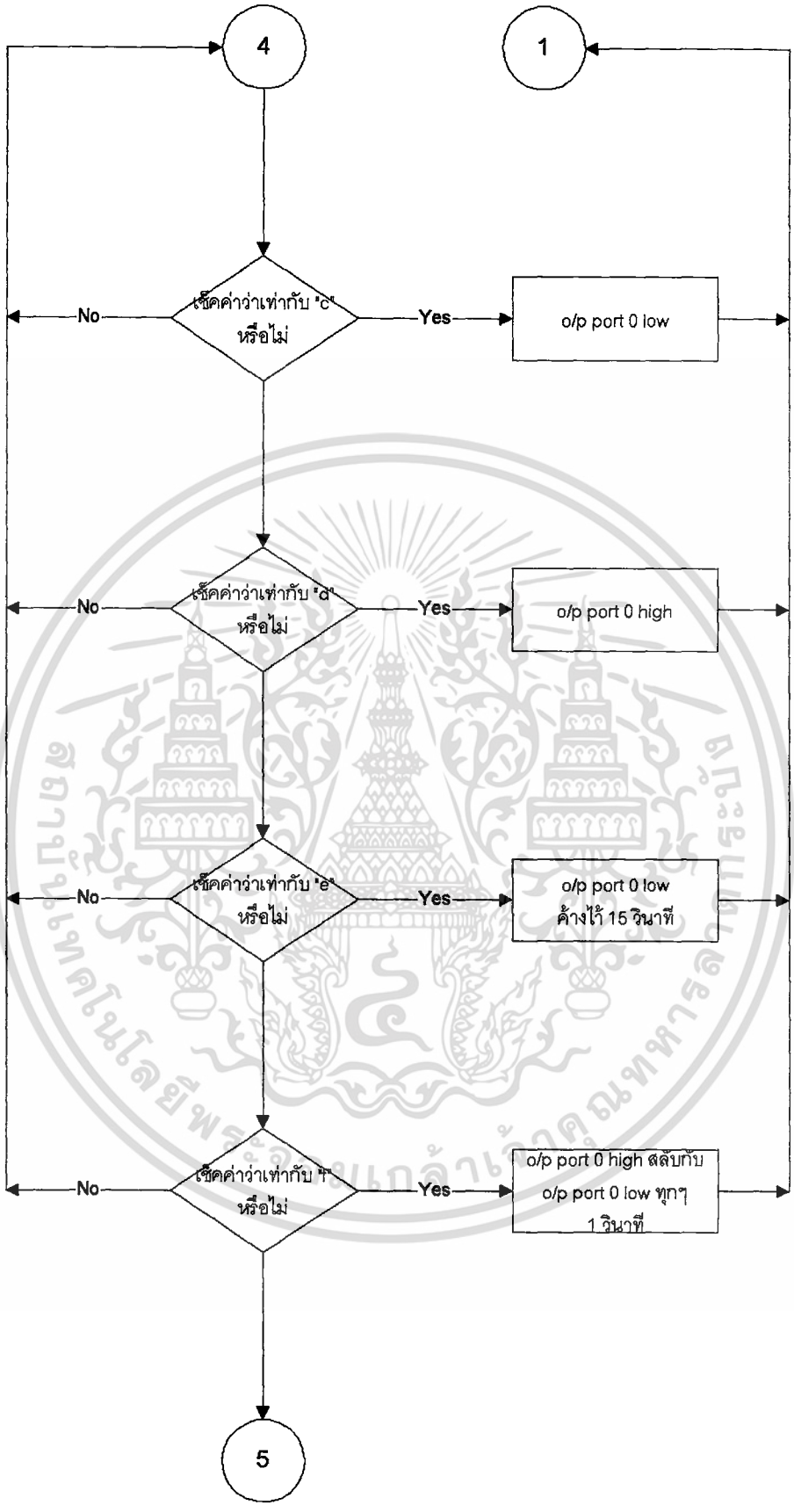
แผนผังแสดงลำดับขั้นตอนการทำงาน

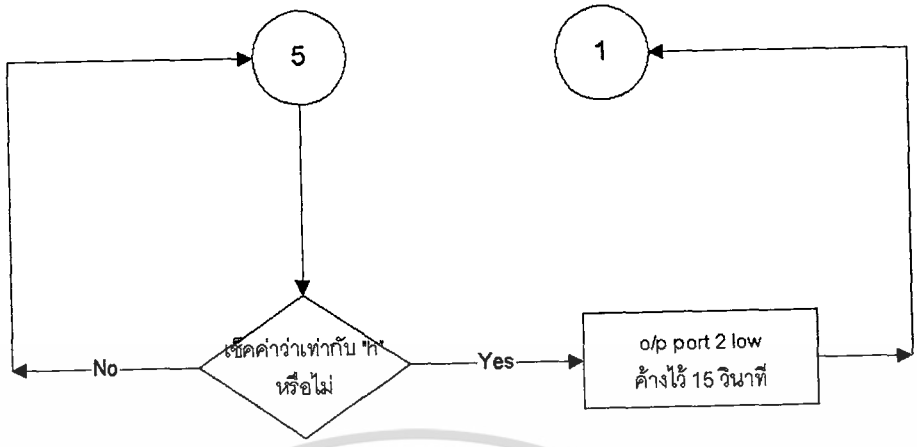












โปรแกรมจะรอรับค่า " A " จากโปรแกรม Visual Basic เมื่อมีการส่งค่าออกไป Basic Stamp จะรอรับค่าที่ ขา 6 ของ I/O port และเช็คว่าเป็น "A" หรือไม่ถ้าไม่ใช่ให้ไปรอรับค่าใหม่ ถ้าใช่ Basic Stamp จะส่งให้ I/O port ทั้งหมด Active High เพื่อเป็นการรีเซ็ตอุปกรณ์ไฟฟ้าให้รอรับคำสั่งต่อไป

หลังจากนั้นโปรแกรมจะมารอรับค่าที่ส่งไปจาก Visual อีกครั้ง โดยค่า แอสกีที่สนใจในการรอรับคือ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 0 , a , b , c , d , e , f , h ถ้าเช็คได้ว่าเป็นตัวใดก็จะกระโดดไปทำงานตามฟังก์ชันนั้นๆ

- ถ้าเป็น "1" ขา O/P Port ที่ 0 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "2" ขา O/P Port ที่ 0 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน
- ถ้าเป็น "3" ขา O/P Port ที่ 1 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "4" ขา O/P Port ที่ 1 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน
- ถ้าเป็น "5" ขา O/P Port ที่ 2 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "6" ขา O/P Port ที่ 2 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน
- ถ้าเป็น "7" ขา O/P Port ที่ 3 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "8" ขา O/P Port ที่ 3 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน
- ถ้าเป็น "9" ขา O/P Port ที่ 4 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "0" ขา O/P Port ที่ 4 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน
- ถ้าเป็น "a" ขา O/P Port ที่ 5 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "b" ขา O/P Port ที่ 5 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน
- ถ้าเป็น "c" ขา O/P Port ที่ 6 Active Low ทำให้อุปกรณ์ไฟฟ้าทำงาน
- ถ้าเป็น "d" ขา O/P Port ที่ 6 Active Hight ทำให้อุปกรณ์ไฟฟ้าไม่ทำงาน

- ถ้าเป็น "e" ขา O/P Port ที่ 0 Active Low ค้างไว้เป็นเวลา 15 นาที และจะเปลี่ยนกลับไป Active High ทำให้อุปกรณ์ไฟฟ้าทำงานในโหมด Timer ตั้งเวลาการทำงานไว้ที่ 15 นาที

- ถ้าเป็น "f" ขา O/P Port ที่ 6 Active Low และ High สลับกันทุกๆ 1 วินาที เป็นเวลา 15 นาที ทำให้อุปกรณ์ไฟฟ้าทำงาน เปิด / ปิด สลับทุก 1 วินาที เป็นเวลา 15 นาที สามารถออกแบบใช้ได้กับเครื่องรดน้ำต้นไม้

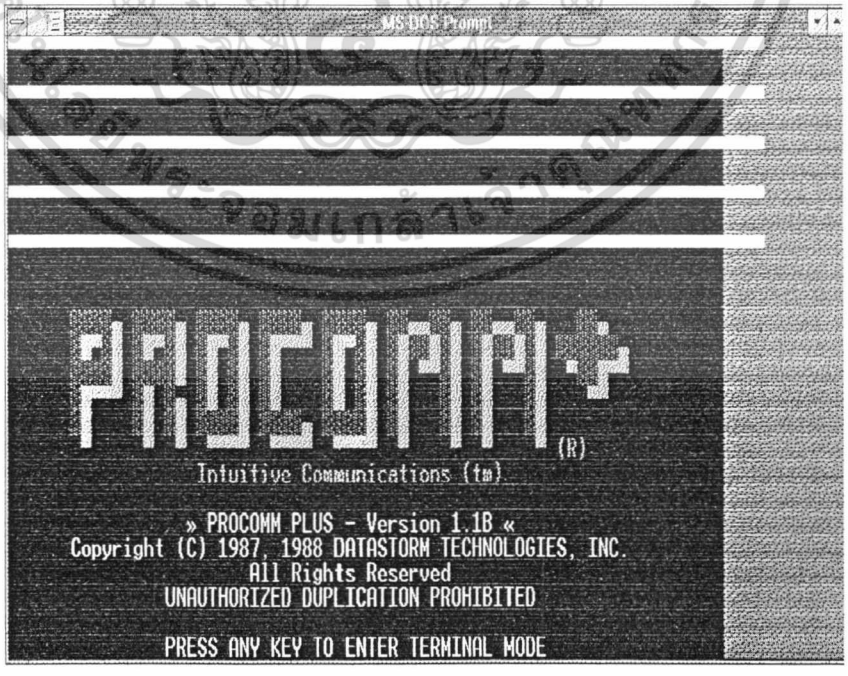
- ถ้าเป็น "h" ขา O/P Port ที่ 6 Active Low ค้างไว้เป็นเวลา 15 นาที และจะเปลี่ยนกลับไป Active High ทำให้อุปกรณ์ไฟฟ้าทำงานในโหมด Timer ตั้งเวลาการทำงานไว้ที่ 15 นาที

ถ้าฟังก์ชันย่อยทำงานแล้ว โปรแกรมจะวนไปรอรับค่าใหม่

3.3.4 กำหนดค่า Profile modem

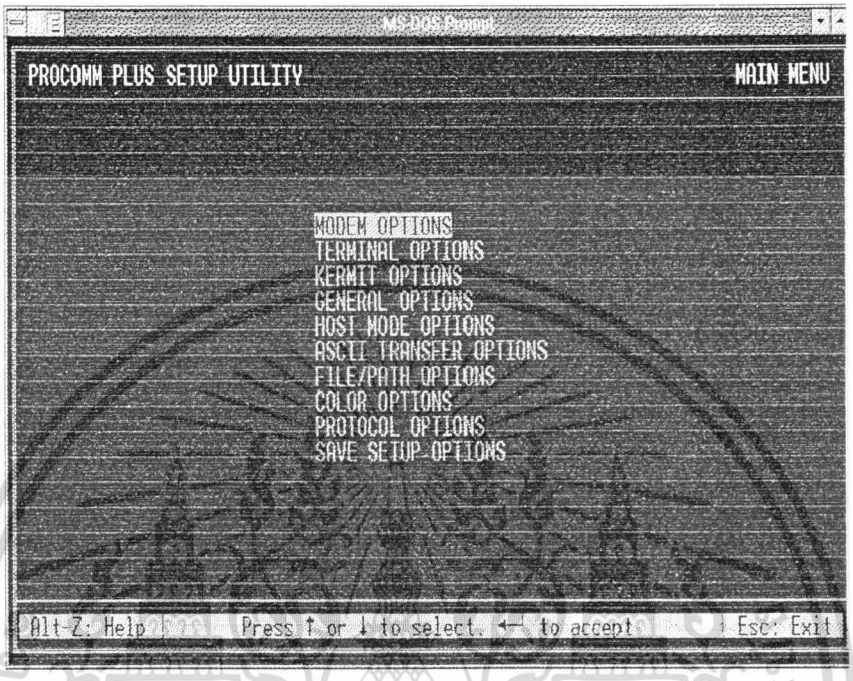
การออกแบบให้โมเด็ม 2 ตัว รับส่งข้อมูล ระหว่างกันได้นั้นจะต้องทำการกำหนดค่า Initialization ของโมเด็มให้สัมพันธ์กัน แบ่งหน้าที่ออกเป็น 2 ส่วน มีขั้นตอนการทำงานดังนี้

1.เลือกใช้โปรแกรม PROCOMM PLUS ซึ่งเป็นโปรแกรมสำเร็จรูป สำหรับการติดต่อสื่อสารทางพอร์ตอนุกรม version ที่เลือกใช้คือ 1.1 ทำงานภายใต้ MS DOS



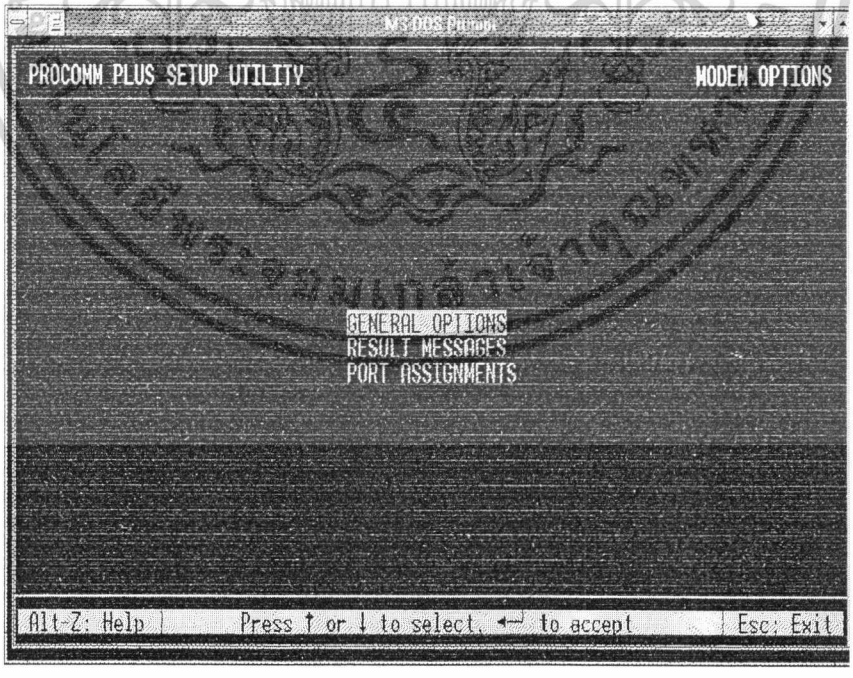
รูปที่ 3.11 แสดงโปรแกรม PROCOMM PLUS version 1.1

2. ใช้คำสั่ง ALT - S จะเข้าสู่การกำหนดค่าต่างๆ เลือกไปที่ MODEM OPTION และกด ENTER



รูปที่ 3.12 แสดงหน้าจอ PROCOMM PLUS SETUP UTILITY

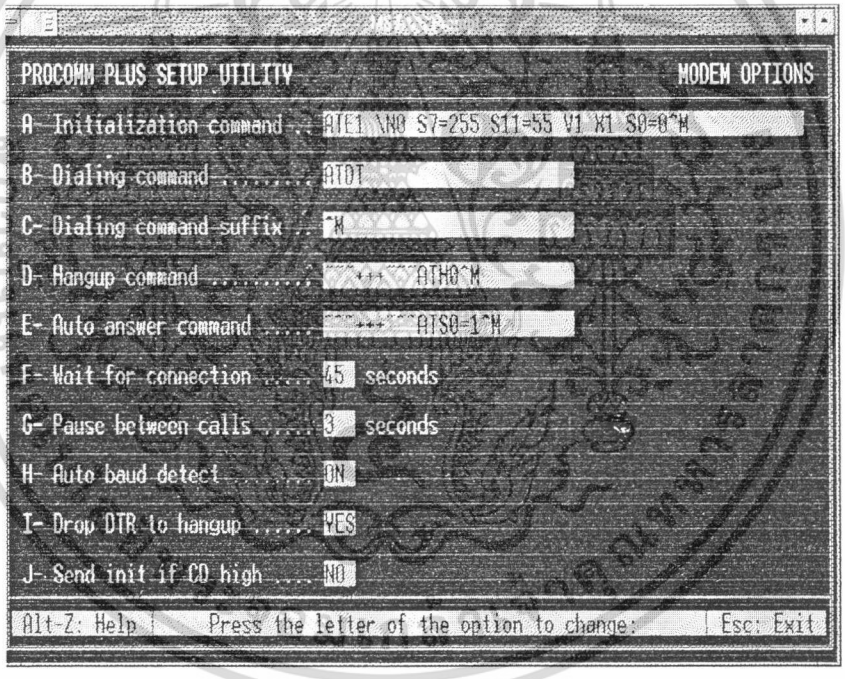
3. เข้ามายัง หน้าจอของ MENU OPTIONS เลือกไปที่ GENERAL OPTIONS



รูปที่ 3.13 แสดงหน้าจอ MENU OPTIONS

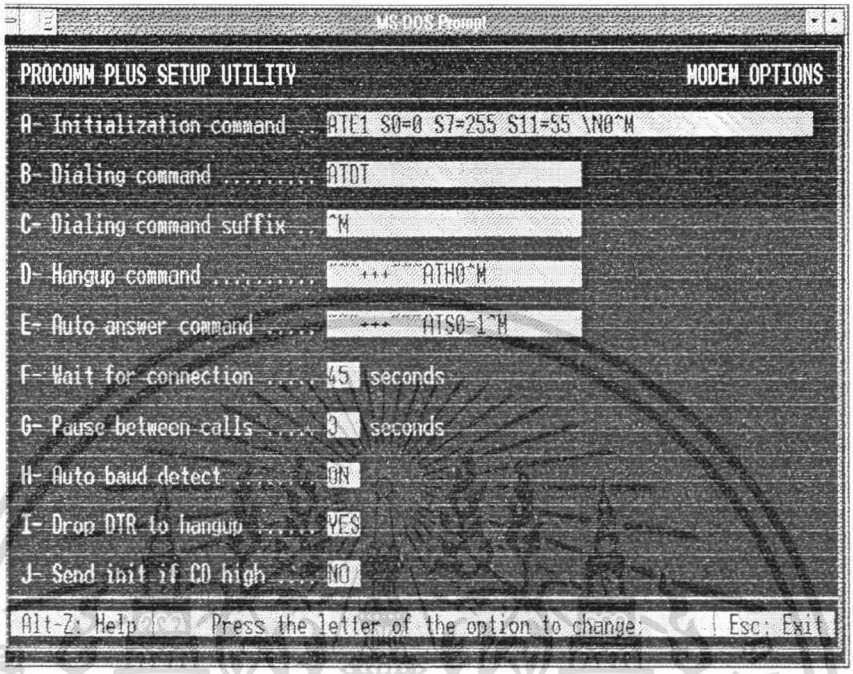
4. เมื่อเข้ามายังหน้าจอนี้ จะแสดงค่าต่างๆ ของการติดต่อสื่อสารของ โมเด็มโดยทั่วๆไป จากรูปประกอบไปด้วย

   - Initialization command : กำหนดค่า register ภายในโมเด็ม
   - Dialing command : คำสั่งการติดต่อของโมเด็ม
   - Hangup command : กำหนดคำสั่งการยกหู
   - Auto answer command : กำหนดคำสั่งการตอบรับ
   - Wait for connection : เวลาสำหรับการถือสายรอ
   - Pause between calls : ระยะเวลาต่อหนึ่งสัญญาณเรียกสาย



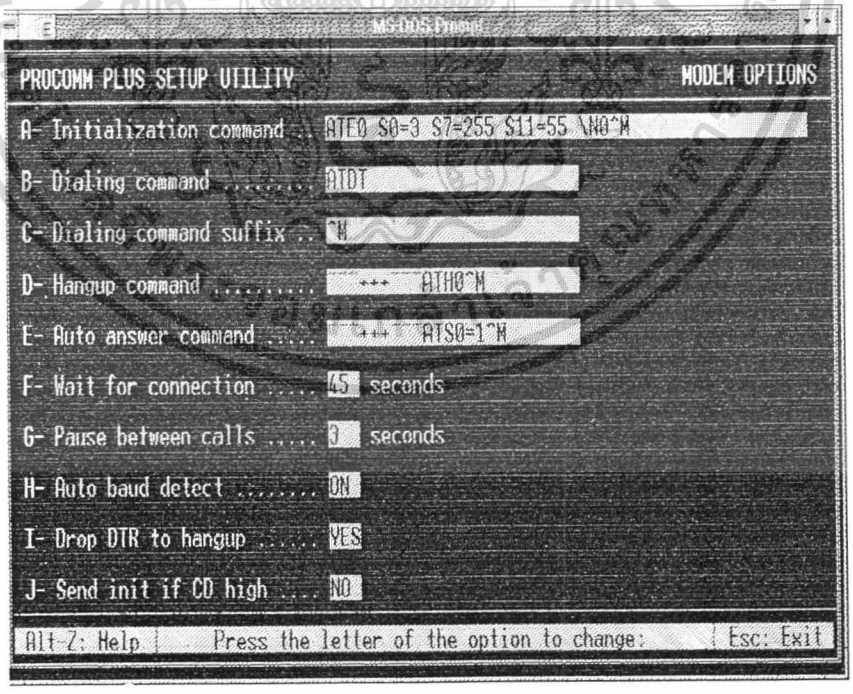
รูปที่ 3.14 แสดงค่าต่างๆของการติดต่อสื่อสารของโมเด็มโดยทั่วๆไป

5. กำหนดให้โมเด็มตัวส่ง หรือตัวที่อยู่ต้นทางมีค่าดังต่อไปนี้



รูปที่ 3.15 แสดงค่า Initialization command ของโมเด็มตัวส่ง

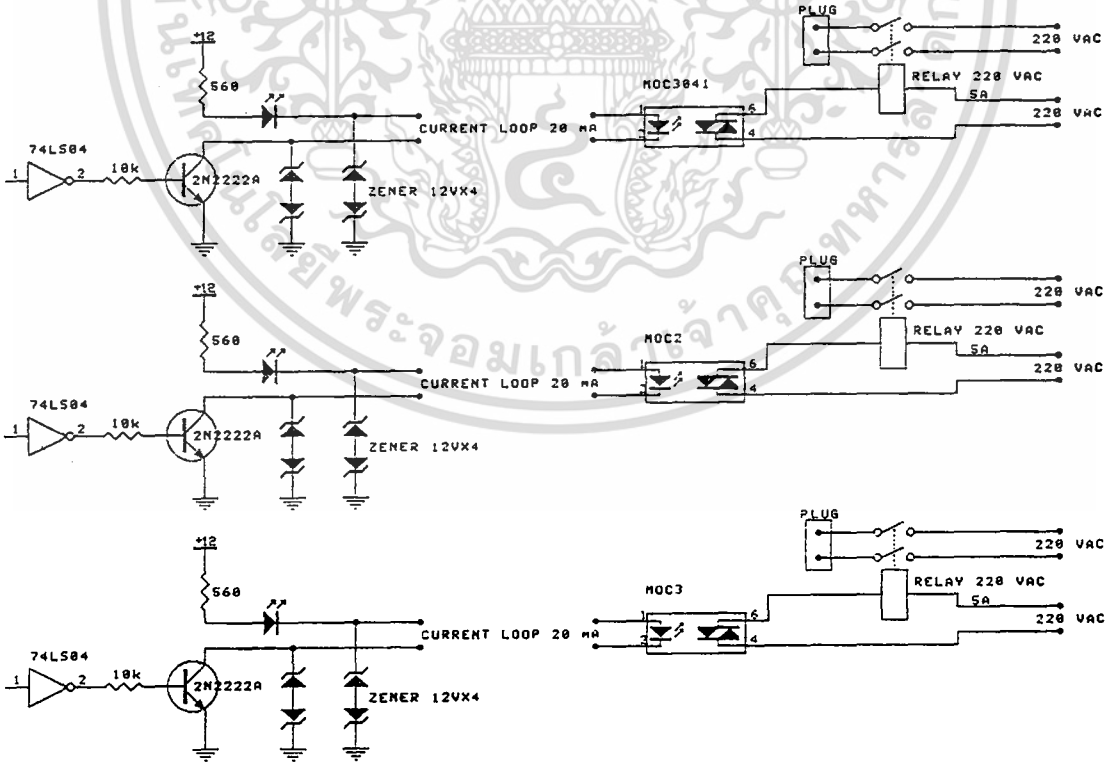
6. กำหนดให้โมเด็มตัวรับ หรือตัวที่อยู่ต้นทางมีค่าดังต่อไปนี้



รูปที่ 3.16 แสดงค่า Initialization command ของโมเด็มตัวรับ

3.3.5 DRIVER CIRCUIT

ในการเชื่อมต่อระหว่างวงจรที่มีแรงดันไฟฟ้าต่ำๆ (Digital circuit) กับวงจรที่มีแรงดันไฟฟ้าสูงๆ (Power circuit) นิยมที่จะใช้ ไอซีประเภท ออปโตไอโซเลเตอร์ ที่มีเอาต์พุตเป็นแบบไตรแอค ในการเชื่อมต่อเพื่อความปลอดภัย ในการออกแบบ เราเลือกใช้ไอซีออปโตไอโซเลทเบอร์ MOC3041 Triac Driver Output ในการเชื่อมต่อ เหตุผลที่เราต้องใช้การเชื่อมต่อในลักษณะดังกล่าวก็เพื่อที่จะควบคุมการปิดเปิดอุปกรณ์ไฟฟ้าด้วยลอจิก 1 จะไม่มีกระแสที่ขาเบสของทรานซิสเตอร์ไหล (ในที่นี้เราใช้ทรานซิสเตอร์ NPN เบอร์ 2N2222A) ดังนั้นจึงไม่มีการดึงกระแสที่ขาคอลเลคเตอร์ ทำให้ไม่เกิดการเปล่งแสงของโฟโตไดโอดภายในออปโตไอโซเลเตอร์ เอาท์พุตไตรแอคจึงอยู่ในสภาวะ OFF แต่ถ้าให้สถานะทางอินพุตของอินเวอร์สเตอร์เป็นลอจิก 0 จะทำให้มีกระแสไหลที่ขาเบสของทรานซิสเตอร์ ดังนั้นจึงมีการดึงกระแสที่ขาคอลเลคเตอร์ ทำให้โฟโตไดโอดทำงานไปทริกให้เอาท์พุตไตรแอคอยู่ในสภาวะ ON ซึ่งจะทำให้รีเลย์ที่อาศัยไฟกระแสสลับ 220 โวลต์ เกิดการเหนียวนำและดึงหน้าสัมผัสให้ต่อกับไฟกระแสสลับ 220โวลต์ จึงทำให้อุปกรณ์ไฟฟ้าเปิด ซึ่งจากการทำงานจะเห็นว่าไตรแอคจะทำหน้าที่เป็นสวิตซ์สำหรับไฟกระแสสลับ 220 โวลต์ (Bidirectional Switch) โดยสามารถควบคุมการปิดเปิด ได้ด้วยไมโครคอนโทรลเลอร์อีกทีหนึ่ง



รูปที่ 3. 15 แสดงวงจรที่ใช้ในการเชื่อมต่อไมโครคอนโทรลเลอร์กับไฟสลับ 220 โวลต์

บทที่ 4
ผลการทดลอง

4.1 ผลการทดลองในส่วนของโปรแกรม Visual Basic

โปรแกรมที่ออกแบบขึ้นมาใช้งานได้ง่าย สามารถส่งข้อมูลออกทางพอร์ตอนุกรมได้ และมีกราฟฟิกที่สวยงาม สามารถใช้งานได้จริง ผลการทดลองโปรแกรมมีความสามารถดังนี้

1. สามารถสั่งงานให้โมเด็มหมุนและโทรเลขหมายได้
2. สั่งงานให้อุปกรณ์ไฟฟ้าทำงานได้
3. ระบบสั่งงานให้โมเด็มยกหูโทรศัพท์ หลังจากการทำงานเสร็จสิ้นได้

4.2 ผลการทดลองในส่วนของโปรแกรม Basic Stamp

โปรแกรมสามารถสั่งงานวงจร driver ให้ทำงาน ON - OFF อุปกรณ์ไฟฟ้า 220 โวลต์ ได้ตามเป้าหมาย

4.3 ผลการทดลองในส่วนของการติดต่อระหว่างโมเด็ม

โมเด็มตัวต้นทาง และโมเด็มปลายทางสามารถรับส่งข้อมูลด้วยกันได้อย่างเหมาะสม

เมื่อสั่งให้มีการหมุนไปยังหมายเลขปลายทาง ต้องรอสัญญาณตอบรับสาย 3 ครั้ง แล้วถ้าโทรติด จะมีไฟขึ้นที่ CD ของตัวโมเด็มต้นทาง แสดงว่าพร้อมรับส่งข้อมูลส่วนที่โมเด็มปลายทาง จะมีการรับสายเองโดยอัตโนมัติ

บทที่ 5
สรุปผลการทดลองและแนวทางแก้ไข

สรุปผลการทดลอง

จากการออกแบบและการใช้งานระบบควบคุมระยะไกลผ่านสายโทรศัพท์ พบว่า เราสามารถควบคุมการเปิด/ปิด อุปกรณ์ไฟฟ้า รวมถึงการตั้งเวลาการใช้งานอุปกรณ์ที่ดยู่ระยะไกลได้

อุปกรณ์ไฟฟ้าต้องเลือกใช้ กับอุปกรณ์ Driver ที่เหมาะสม คือ

1. Driver ที่เป็น Relay ใช้กับหลอดไฟฟลูออเรสเซนต์
2. Driver ที่เป็น Solid State Relay ใช้กับอุปกรณ์ที่เป็น แอร์ พัดลม วิทยุ โทรทัศท์ หลอดไฟฟ้าที่มีไส้หลอด

ปัญหาที่เกิดขึ้นระหว่างขั้นตอนการออกแบบ

1. ไมโครคอนโทรลเลอร์ที่เลือกใช้ในการออกแบบระบบนี้ คือ BASIC STAMP ONE ซึ่งออกแบบมาใช้กับการควบคุมเล็ก ๆ ทำให้เครื่องควบคุมระยะไกลมีขีดจำกัดดังนี้

1.1) ความเร็วในการส่งข้อมูล 2400 b/s นับว่าช้าเกินไป

1.2) สามารถเก็บข้อมูลได้ 256 bit ทำให้เขียนฟังก์ชันการทำงานได้ไม่มาก

วิธีแก้ไข ควรเลือกใช้ไมโครคอนโทรลเลอร์ ที่มีคุณสมบัติที่มากกว่านี้ เช่น BASIC STAMP TWO ส่งข้อมูลด้วยความเร็วสูงถึง 50000 b/s และเก็บข้อมูลได้มากถึง bit ทำาให้สามารถเขียนโปรแกรมการใช้งานได้มากกว่านี้

2. ถ้าอุปกรณ์ปลายทางที่สั่งงานไม่ทำงาน ผู้ใช้ไม่สามารถรับรู้ได้

วิธีแก้ไข ต้องออกแบบระบบ SENSER หรือกล้องจับภาพติดไว้ที่ปลายทาง แล้วส่งภาพกลับมายังผู้ตวบคุม แสดงผลบนหน้าจอว่าขณะนี้อุปกรณ์ปลายทางทำงานตามที่สั่งหรือไม่

สำหรับแนวทางการพัฒนานั้นสามารถทำได้โดย

1)ในส่วนของโปรแกรมวิชวลเบสิคสามารถนำไปพัฒนาให้มีการใช้งานได้ง่ายและสะดวกยิ่งขึ้น

2)สำหรับส่วนตัวควบคุมที่ต้นทางนั้นอาจพัฒนาให้ตัวรับข้อมูลนั้นเปลี่ยนเป็นอุปกรณ์อย่างอื่น เช่น ตัวเซนเซอร์ ฯลฯ แล้วผ่านวงจร

บรรณานุกรม

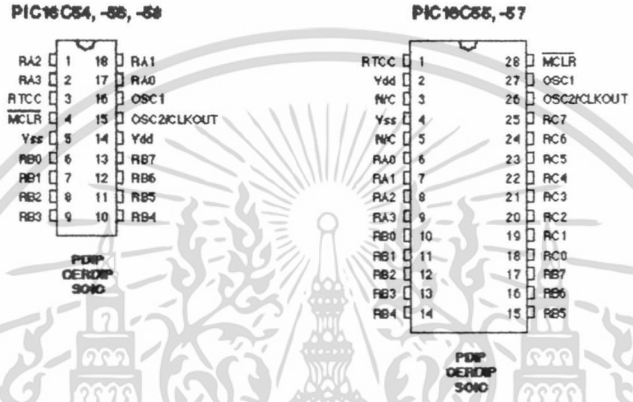
รศ.ยืน ภู่วรวรรณ , **โมเด็ม** , บริษัทซีเอ็ดยูเคชั่น , 2532

Condensed PIC Data Sheets

BLANK PAGE

PIC16C5x Pin-Outs

The following diagrams show the 18-pin and 28-pin PIC16C5x pin-outs:



Pin	Function
RA0 - RA3	I/O Port A
RB0 - RB7	I/O Port B
RC0 - RC7	I/O Port C (only on 28-pin PIC's)
RTCC	Real-time clock/counter input
$\overline{\text{MCLR}}$	Master clear (reset)
OSC1	Oscillator input
OSC2/CLKOUT	Oscillator output (OSC/4)
Vdd	Power supply
Vss	Ground
N/C	No connection

PIC16C5x Devices

PIC16C5x Microcontrollers

The table below shows the various PIC16C5x devices available:

PART #	ERASE	(E)PROM	RAM	I/O	SUPPLY	OSC	FREQ
PIC16C54-RC/P	No	512 x 12	32 x 8	12	3.00 - 6.25 V	RC	DC - 4 MHz
PIC16C54-XT/P	No	512 x 12	32 x 8	12	3.00 - 6.25 V	XTAL	DC - 4 MHz
PIC16C54-HS/P	No	512 x 12	32 x 8	12	4.50 - 5.50 V	XTAL	DC - 20 MHz
PIC16C54-LP/P	No	512 x 12	32 x 8	12	2.50 - 6.25 V	XTAL	DC - 40 kHz
PIC16C54/JW	Yes	512 x 12	32 x 8	12	3.00 - 5.50 V	RC,XTAL	DC - 20 MHz
PIC16C55-RC/P	No	512 x 12	32 x 8	20	3.00 - 6.25 V	RC	DC - 4 MHz
PIC16C55-XT/P	No	512 x 12	32 x 8	20	3.00 - 6.25 V	XTAL	DC - 4 MHz
PIC16C55-HS/P	No	512 x 12	32 x 8	20	4.50 - 5.50 V	XTAL	DC - 20 MHz
PIC16C55-LP/P	No	512 x 12	32 x 8	20	2.50 - 6.25 V	XTAL	DC - 40 kHz
PIC16C55/JW	Yes	512 x 12	32 x 8	20	3.00 - 5.50 V	RC,XTAL	DC - 20 MHz
PIC16C56-RC/P	No	1K x 12	32 x 8	12	3.00 - 6.25 V	RC	DC - 4 MHz
PIC16C56-XT/P	No	1K x 12	32 x 8	12	3.00 - 6.25 V	XTAL	DC - 4 MHz
PIC16C56-HS/P	No	1K x 12	32 x 8	12	4.50 - 5.50 V	XTAL	DC - 20 MHz
PIC16C56-LP/P	No	1K x 12	32 x 8	12	2.50 - 6.25 V	XTAL	DC - 40 kHz
PIC16C56/JW	Yes	1K x 12	32 x 8	12	3.00 - 5.50 V	RC,XTAL	DC - 20 MHz
PIC16C57-RC/P	No	2K x 12	80 x 8	20	3.00 - 6.25 V	RC	DC - 4 MHz
PIC16C57-XT/P	No	2K x 12	80 x 8	20	3.00 - 6.25 V	XTAL	DC - 4 MHz
PIC16C57-HS/P	No	2K x 12	80 x 8	20	4.50 - 5.50 V	XTAL	DC - 20 MHz
PIC16C57-LP/P	No	2K x 12	80 x 8	20	2.50 - 6.25 V	XTAL	DC - 40 kHz
PIC16C57/JW	Yes	2K x 12	80 x 8	20	3.00 - 5.50 V	RC,XTAL	DC - 20 MHz
PIC16LC58A-04/P	No	2K x 12	80 x 8	20	2.50 - 6.25 V	RC,XTAL	DC - 4 MHz
PIC16C58A-04/P	No	2K x 12	80 x 8	20	3.00 - 6.25 V	XTAL	DC - 4 MHz
PIC16C58A-20/P	No	2K x 12	80 x 8	20	3.00 - 6.25 V	XTAL	DC - 20 MHz
PIC16C58A/JW	Yes	2K x 12	80 x 8	20	3.00 - 6.25 V	RC,XTAL	DC - 20 MHz

Peripheral Features

In addition to the obvious features shown above, the PIC's have a number of not-so-obvious features and qualities that are important:

- **Fully static operation,** allowing you to stop the oscillator and then restart where you left off, with all registers intact. This is useful in applications where power conservation is important.
- **Individual I/O pins are programmable as inputs or outputs.**
  Sink current: 25 mA per pin, 50 mA per port
  Source current: 20 mA per pin, 40 mA per port
- **2-level hardware stack.**

Peripheral Features *(continued)*

- **Real-Time Clock/Counter (RTCC).** The RTCC is an 8-bit counter, which can be driven by the RTCC pin or by the PIC's internal instruction clock (OSC/4). If the external pin is used, the counter can be set to increment on low-to-high or high-to-low transitions.

  Normally, the RTCC is driven directly by either source. For higher count values, though, the prescaler can be used to effectively increase the RTCC to 16 bits.

  The RTCC signal source and trigger edge are determined by bits in the Option register (see next section for register descriptions).

  See the Microchip PIC16C5x data sheet for details concerning timing characteristics for the RTCC's external input.

- **Watchdog Timer (WDT).** When enabled, the watchdog timer is used to reset the PIC if the program has "crashed." Normally, a CLR WDT instruction in the main loop of your program would prevent the watchdog timer from ever timing out and resetting the PIC. However, if the program was not executing properly, the watchdog timer would reset the PIC.

  The watchdog timer works from an internal oscillator, allowing it to run even if the PIC's main oscillator has stopped. The watchdog timer's normal time-out period is 18 ms, but can be increased to several seconds by using the post-scaler.

- **Prescaler/Post-scaler.** This 8-bit counter can be assigned to the RTCC (as a prescaler) or the watchdog timer (as a post-scaler). For simplicity, this counter is normally referred to as the "prescaler," even when it's used as a post-scaler.

  When assigned to the RTCC, the prescaler is placed between the RTCC and its clock source. The clock signal which would normally increment the RTCC, increments the prescaler. When the prescaler overflows, the RTCC is incremented. Increment ratios from 1:2 - 1:256 can be used, effectively giving you a 16-bit RTCC.

  When assigned to the watchdog timer, the prescaler is placed between the watchdog timer and the PIC's reset circuit. The watchdog timer signal which would normally reset the PIC,

Peripheral Features *(continued)*

increments the prescaler. When the prescaler overflows, the PIC is reset. Delay ratios from 1:1 - 1:128 can be used, allowing the watchdog period to be set from 18 ms to several seconds.

The prescaler setup is determined by bits in the Option register *(see the next section for register descriptions)*.

- **Code-protect fuse.** This is a special fuse that can be blown during programming; once the fuse is blown, the PIC's EPROM cannot be read.
- **Power saving sleep mode.** For applications where low power consumption is important, the PIC has a sleep mode. This mode is entered by executing a SLEEP instruction, which shuts down the oscillator. I/O pins maintain whatever state they had when the sleep mode was entered.

  To awaken the PIC from sleep mode, a reset must be performed, either from the MCLR pin or from a watchdog timer time-out.

  Once the PIC is running, the "PD" and "TO" bits in the Status register can be read to determine 1) if the PIC was powered up or awakened from sleep, and 2) if the wake-up was caused by an external reset or by the watchdog timer.
- **Programmable oscillator type.** The PIC can run with any of four oscillator types, as shown below:

LP:	Low power crystal	(DC - 40 KHz)
RC:	Resistor & capacitor	(DC - 4 MHz, ± 13-39 percent)
XT:	Crystal or resonator	(100 KHz - 4 MHz)
HS:	High speed crystal	(4 - 20 MHz)

The oscillator type is determined by two EPROM bits, which are normally programmed at the factory. The RC and erasable PIC's, however, can be user-programmed for any of the oscillator types. Keep in mind, though, that RC-type PIC's are only tested for use with RC oscillators.

Internal Architecture

Internally, the PIC is based on a register file concept with separate busses and memories for data and instructions (sometimes called "Harvard architecture"). The data bus and memory (RAM) are 8-bits wide, while the program bus and memory (EPROM) are 12-bits wide. All PIC instructions and their operands fit into a single 12-bit word, resulting in smaller code and faster execution. PIC programs are typically 33-50 percent smaller than programs written for 8-bit processors. And most instructions execute in a single instruction cycle (4 clock cycles); instructions that affect the program counter take an extra instruction cycle, for a total of 8 clock cycles. To further increase speed, the PIC uses overlapping instruction fetch and execution cycles; while one instruction is executed, the following instruction is being read from program memory. Because of its efficiency, the PIC can deliver 5 MIPS execution with a clock frequency of 20 MHz.

PIC16C5x Registers

The following table shows the various PIC16C5x registers; the function of each register is described in the following pages.

Register	Function
00h	Indirect addressing register
01h	Real-time clock/counter (RTCC)
02h	Program counter (PC)
-	Stack registers (2)
03h	Status register
04h	File select register (FSR)
05h	I/O Port A
06h	I/O Port B
07h	I/O Port C
-	TRISA
-	TRISB
-	TRISC
-	W register
-	Option register
08h - 0Fh	General purpose registers
10h - 1Fh	General purpose registers (4 banks in PIC16C57)

PIC16C5x Registers *(continued)*

Quick reference register map:

	BANK 0	BANK 1*	BANK 2*	BANK 3*
	W register	W register	W register	W register
	Stack (2)	Stack (2)	Stack (2)	Stack (2)
	Option register	Option register	Option register	Option register
	TRISA	TRISA	TRISA	TRISA
	TRISB	TRISB	TRISB	TRISB
	TRISC	TRISC	TRISC	TRISC
00h	Indirect addr.	Indirect addr.	Indirect addr.	Indirect addr.
01h	RTCC	RTCC	RTCC	RTCC
02h	PC	PC	PC	PC
03h	STATUS	STATUS	STATUS	STATUS
04h	FSR	FSR	FSR	FSR
05h	PORT A	PORT A	PORT A	PORT A
06h	PORT B	PORT B	PORT B	PORT B
07h	PORT C	PORT C	PORT C	PORT C
08h	8 general purpose registers (RAM)	reads/writes registers 08h - 0Fh	reads/writes registers 08h - 0Fh	reads/writes registers 08h - 0Fh
0Fh				
10h	16 general purpose registers (RAM)	30h 16 general purpose registers (RAM)*	50h 16 general purpose registers (RAM)*	70h 16 general purpose registers (RAM)*
1Fh		3Fh	5Fh	7Fh
	(00h - 1Fh)	(20h - 3Fh)*	(40h - 5Fh)*	(60h - 7Fh)*

* Available on PIC16C57 only.

PIC16C5x Registers *(continued)*

The following text describes the function of each register. For some of the registers, you'll notice the designation "xxh" following the register name. This indicates the address of the register. Registers with no address cannot be addressed directly.

- **Indirect Addressing Register (00h).** This register doesn't actually exist. Naming register 00h in an instruction causes the PIC to read the register *pointed to* by register 04h (file select register). For example, the instruction "`ADD 00h,#05`" will *not* add five to register 00h; instead, it will add five to whatever register is *pointed to* by the address in register 04h.

  If register 00h itself is read through register 04h (04h contains "00h"), 00h will be returned. If register 00h is written to through register 04h, the PIC will execute a NOP.

- **Real-Time Clock/Counter (01h: RTCC).** This is the location of the RTCC (see previous section for a description). Although its contents may change in response to a clock signal, the RTCC register may be read and written just as any other register.

- **Program Counter (02h: PC).** The program counter holds the address for the instruction currently being executed. The program counter and its associated two-level stack are 9-11 bits wide, depending on the EPROM size of the PIC being used.

  Certain instructions affect the program counter, as shown below:

  GOTO (Microchip) and JMP (Parallax) load the lower 9 bits of the program counter. In the PIC16C56 and '57, which have more than 512 words of EPROM, the upper two bits of the program counter are loaded with the page select bits from the status register. The Parallax instruction set includes a convenient instruction, LJMP, which sets the page select bits before executing the jump.

  CALL loads the lower 8 bits of the program counter and clears the ninth bit. The program counter + 1 is pushed into the stack. In the '56 and '57, the upper two bits of the program counter are loaded with the page select bits from the status register. The Parallax instruction set includes LCALL, which

PIC16C5x Registers *(continued)*

sets the page select bits before executing the jump.

RETLW (Microchip) and **RETW** (Parallax) load the program counter with the address most recently pushed on the stack by a CALL instruction.

Instructions which load a computed value into the program counter, such as **JMP PC+W**, load the value into the lower 8 bits. The ninth bit of the program counter is cleared. In the PIC16C56 and '57, the upper two bits of the program counter are loaded with the page select bits from the status register.

It should be noted that because the ninth bit of the program counter is cleared by CALL instructions and computed value instructions (such as JMP PC+W), all subroutine calls and computed jumps must have their destination in the first 256 locations of any page (each page is 512 words).

As you may have noticed when reading the JMP and CALL paragraphs above, the program counter may not be loaded as expected when using a PIC with more than 512 words of EPROM. This is because the upper two bits of the program counter are loaded with the page select bits from the status register. If your program continues into the second page of memory and executes a JMP without having properly set the page select bits, execution may jump to another page (probably not what you want). To avoid this mistake, make sure to set the page select bits for the correct page. Or, use the Parallax "long" instructions, which do this for you (long instructions are LCALL, LJMP, and LSET).

- **Stack.** The stack is a pair of registers which are used for calling and returning from subroutines. The stack is affected by two instructions:

  When a **CALL** is executed, the first stack register is copied into the second register, then the program counter + 1 (the return address) is loaded into the first register. The original contents of the second register are lost. Finally, the program counter is loaded with the subroutine address, at which point execution continues.

PIC16C5x Registers *(continued)*

When a RETLW (Microchip) or RETW (Parallax) is executed, the first stack register is copied into the program counter, then the second register is copied into the first register. Execution continues at the address loaded from the first stack register.

- **Status Register (03h).** This register contains the status of the PIC's arithmetic logic unit (ALU), the reset status, and the page select bits for PIC's with more than 512 words of EPROM.

  The function of each bit in the status register is shown below:

Bit	Function
0	**Carry bit (C).** Set if an addition or subtraction causes an overflow from the most significant bit of the resultant (bit 7). Subtraction is included because it's executed by adding the two's complement. Also used by rotate instructions, which rotate the contents of a register and copy the low or high order bit of the register into the carry bit.
1	**Digit carry bit (DC).** Set if an addition or subtraction causes an overflow from the 4th low order bit (bit 3). Digit carry indicates that more than one hex digit (4 bits) was necessary to accomodate the result.
2	**Zero bit (Z).** Set if the result of an arithmetic or logic operation is zero.
3	**Power-down bit (PD).** Set during power-up or by a CLR WDT (clear watchdog) instruction. Cleared by a SLEEP instruction.
4	**Time-out bit (TO).** Set during power-up, by CLR WDT, or by SLEEP. Cleared by a watchdog time-out.
5-6	**Page select bits (PA0, PA1).** In the '54 and '55, these are unused. In the '56, bit 5 selects program page 0 or 1 (bit 6 is unused). In the '57, both bits select page 0, 1, 2, or 3. Each page is 512 words long.

PIC16C5x Registers *(continued)*

Bit	Function
7	**Unused bit (PA2).** Reserved by Microchip for future use.

The following table shows how various events affect the power-down and time-out bits:

Event	PD	TO
Power-up	1	1
Watchdog time-out	x	0
SLEEP instruction	0	1
CLR WDT instruction	1	1

Lastly, this table shows the status of the power-down and time-out bits after a reset:

Cause of Reset	PD	TO
Watchdog time-out (not during sleep)	1	0
Watchdog time-out (during sleep)	0	0
External reset (not during sleep)	x	x
External reset (during sleep)	0	1
Normal power-up	1	1

- **File Select Register (04h: FSR).** This register serves a dual purpose: it selects the register for indirect addressing, and it selects the current register bank in the PIC16C57.

  In all PIC16C5x devices, bits 0-4 select one of 32 registers in the current bank (only the '57 has more than one bank).

  In the PIC16C57, bits 5-6 select one of four register banks. Each bank has 32 registers. However, reading or writing the lower 16 registers in any bank will access registers 00h - 0Fh (bank 0). Only the upper 16 registers in each bank are unique. This results in one bank of 32 registers, plus three banks of 16 registers, for a total of 80 registers.

PIC16C5x Registers *(continued)*

bank 0. For instance, to load register 30h with #A5h, you would execute the following instructions:

```
MOV   04h,#00100000b    ;Select bank 1
MOV   10h,#A5h          ;Load register 10h in
                        ;bank 1 (register 30h)
                        ;with #A5h
```

- **I/O Port A (05h: RA or Port A).** 4-bit I/O port. This register is used to read and write I/O Port A. This register can be read and written just as any other register. However, read instructions always read the I/O pins, regardless of whether the pins are programmed as inputs or outputs.

  The upper 4 bits are unused and read as 0's.

- **I/O Port B (06h: RB or Port B).** 8-bit I/O port.

- **I/O Port C (07h: RC or Port C).** 8-bit I/O port, only available on 28-pin PIC's. On 18-pin PIC's, this register can be used for storage.

- **TRISA (TRI-State A).** This is the data direction register for Port A. Bits in this register which are set to "1" cause the corresponding bits in Port A to become inputs (the pins go into high impendence mode, allowing them to be driven by an external source). Bits which are cleared to "0" cause the corresponding bits in Port A to become outputs.

  The data direction registers are not directly addressable; to change their contents, you can use either of these instructions:

`TRIS port_fr`	(Microchip) Copies W into the data direction register for *port_fr*, where *port_fr* is 05h-07h.
`MOV !port_fr,#literal`	(Parallax) Copies *literal* into the data direction register for *port_fr*, where *port_fr* is 05h-07h.

PIC16C5x Registers *(continued)*

- **TRISB.** Data direction register for Port B.
- **TRISC.** Data direction register for Port C.
- **W** (working register). The W register holds the second operand in two-operand instructions and is used in internal data transfer; much like the "accumulator" in other processors.
- **Option Register.** This register defines the prescaler ratio, prescaler assignment, RTCC trigger edge, and RTCC signal source. The function of each bit in the option register is shown below:

Bit	Function
0-2	**Prescaler ratio.** These 3 bits determine the prescaler input-to-output ratio. When using the prescaler with the RTCC, the seven possible ratios are 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128, 1:256. When using the prescaler with the watchdog timer, the ratios are 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128. For example, let's say that the prescaler is assigned to the watchdog timer. To increase the watchdog time-out period to 64 times its normal length, the prescaler ratio would be set to 110b; this yields a watchdog period of approx. 1 second (64 x 18 ms).
3	**Prescaler assignment.** This bit determines whether the prescaler is assigned to the RTCC ("0") or to the watchdog timer ("1").
4	**RTCC trigger edge.** This bit determines whether the RTCC increments on a low-to-high ("0") or high-to-low ("1") transition on the RTCC pin.
5	**RTCC signal source.** This bit determines whether the RTCC is driven by the PIC's internal instruction clock ("0") or by the RTCC pin ("1").

- **General purpose registers.** These registers may be used by your program for storage of variables, data, etc.

Stamp Applications electronic version, vol. 1

This document is a collection of the first seven installments of the *Stamp Applications* column published in *Nuts & Volts* magazine starting in March 1995. The column provides hints, tips, and techniques for users of the popular Parallax BASIC Stamp® single-board computer and its kit equivalent, the Counterfeit. The Counterfeit uses a genuine Parallax BASIC (PBASIC) interpreter chip and software licensed from Parallax. It's 100-percent compatible with the original BASIC Stamp.

I prepared this electronic version of *Stamp Applications* in response to many requests from Stamp users on Compuserve and the Internet. I plan to post additional volumes every six issues/months. Don't bother petitioning me to post more frequently; that would defeat one of the purposes of this document, which is to encourage Stamp users to subscribe to *Nuts & Volts*.

Contacting Us

Scott Edwards Electronics
964 Cactus Wren Lane
Sierra Vista, AZ 85635 USA
ph: 520-459-4802; fax 520-459-0623
e-mail: 72037.2612@compuserve.com

Contacting Nuts & Volts magazine

T&L Publications Inc.
430 Princeland Court
Corona, CA 91719
ph: 909-371-8497; fax: 909-371-3052
subscription order line: 800-783-4624

Contacting Parallax

Parallax, Inc.
3805 Atherton Road, #102
Rocklin, CA 95765 USA
ph: 916-624-8333; fax 916-624-8003; BBS: 916-624-7101
e-mail: info@parallaxinc.com; file transfer via Internet: ftp.parallaxinc.com

Topics in Volume 1

Stamp Applications no. 1 (March '95):

New Column Puts the Spotlight on BASIC Stamp Projects, Hints, and Tips

First Look at the New BS1-IC Stamp-on-a-SIP, by Scott Edwards

WELCOME to the first installment of Stamp Applications, a forum for users of BASIC Stamp single-board computers. Every month we'll introduce new ideas in hardware and software for the Stamp, answer questions, and keep up with news about the Stamp and related products.

To kick things off, let's look at the newest member of the Stamp product line, the BS1-IC, also known as "Stamp on a SIP." SIP stands for single-inline package, and it describes the diminuitive new Stamp's printed-circuit board design. The entire Stamp circuit plugs right into a 14-pin SIP socket. This makes it easier to breadboard with the Stamp, or to incorporate it into your own circuit designs.

Figure 1 shows the layout and important dimensions of the BS1-IC. I elected to sketch the unit rather than photograph it because its small size makes it a real challenge to get a decent picture.

The BS1-IC has four advantages over the regular, full-sized Stamp. The first is size. The BS1-IC is small enough to be treated as a component that can be integrated into ultra-small designs.

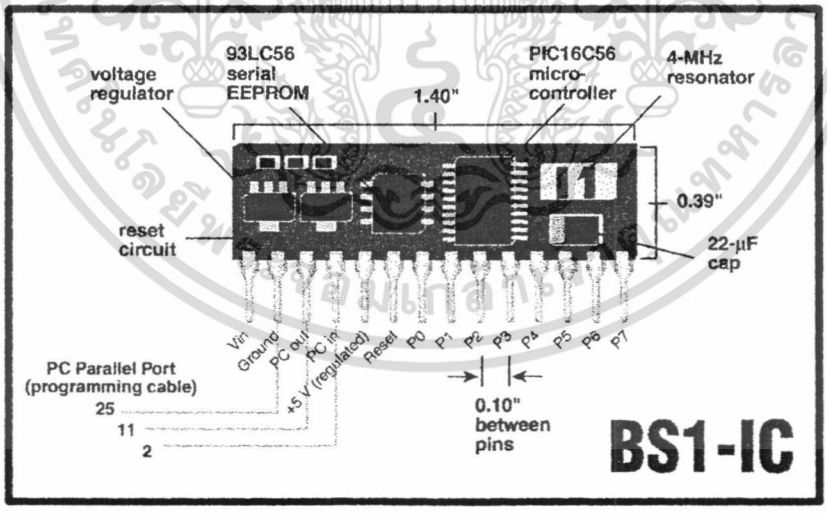


Figure 1. The new BS1-IC.

Stamp Applications electronic version, vol. 1

This document is a collection of the first seven installments of the *Stamp Applications* column published in *Nuts & Volts* magazine starting in March 1995. The column provides hints, tips, and techniques for users of the popular Parallax BASIC Stamp® single-board computer and its kit equivalent, the Counterfeit. The Counterfeit uses a genuine Parallax BASIC (PBASIC) interpreter chip and software licensed from Parallax. It's 100-percent compatible with the original BASIC Stamp.

I prepared this electronic version of *Stamp Applications* in response to many requests from Stamp users on Compuserve and the Internet. I plan to post additional volumes every six issues/months. Don't bother petitioning me to post more frequently; that would defeat one of the purposes of this document, which is to encourage Stamp users to subscribe to *Nuts & Volts*.

Contacting Us

Scott Edwards Electronics
964 Cactus Wren Lane
Sierra Vista, AZ 85635 USA
ph: 520-459-4802; fax 520-459-0623
e-mail: 72037.2612@compuserve.com

Contacting Nuts & Volts magazine

T&L Publications Inc.
430 Princeland Court
Corona, CA 91719
ph: 909-371-8497; fax: 909-371-3052
subscription order line: 800-783-4624

Contacting Parallax

Parallax, Inc.
3805 Atherton Road, #102
Rocklin, CA 95765 USA
ph: 916-624-8333; fax 916-624-8003; BBS: 916-624-7101
e-mail: info@parallaxinc.com; file transfer via Internet: ftp.parallaxinc.com

Topics in Volume 1

Stamp Applications no. 1 (March '95):

New Column Puts the Spotlight on BASIC Stamp Projects, Hints, and Tips

First Look at the New BS1-IC Stamp-on-a-SIP, by Scott Edwards

WELCOME to the first installment of Stamp Applications, a forum for users of BASIC Stamp single-board computers. Every month we'll introduce new ideas in hardware and software for the Stamp, answer questions, and keep up with news about the Stamp and related products.

To kick things off, let's look at the newest member of the Stamp product line, the BS1-IC, also known as "Stamp on a SIP." SIP stands for single-inline package, and it describes the diminuitive new Stamp's printed-circuit board design. The entire Stamp circuit plugs right into a 14-pin SIP socket. This makes it easier to breadboard with the Stamp, or to incorporate it into your own circuit designs.

Figure 1 shows the layout and important dimensions of the BS1-IC. I elected to sketch the unit rather than photograph it because its small size makes it a real challenge to get a decent picture.

The BS1-IC has four advantages over the regular, full-sized Stamp. The first is size. The BS1-IC is small enough to be treated as a component that can be integrated into ultra-small designs.

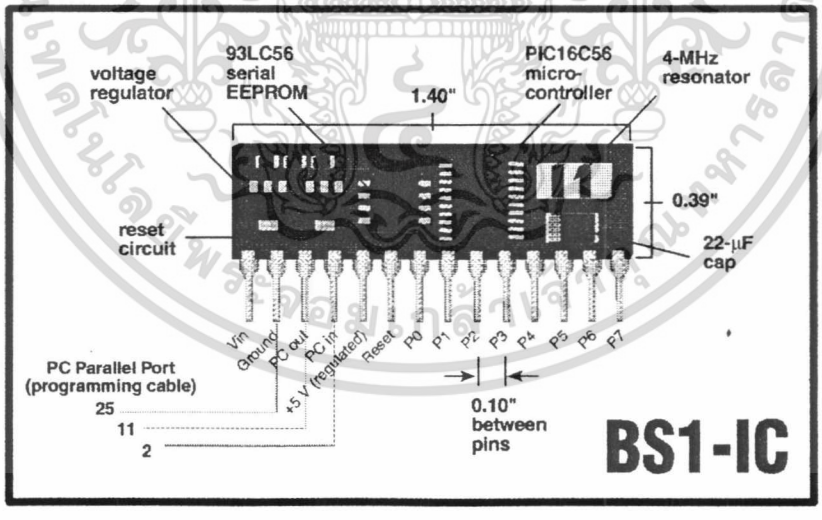


Figure 1. The new BS1-IC.

The second advantage is price. At $29, the BS1-IC has everything the original $39 Stamp had, except for the battery clips and prototyping area. If your power supply is something other than a 9V battery, you won't miss the clips. And if you're not a fan of wire-wrapping (I detest it! Crocheting for engineers!), you'd probably never use the grid-of-holes proto area anyway. The BS1-IC plugs right into socket-type proto boards, like the ones sold by Radio Shack and Jameco, allowing rapid circuit testing. If you want to add the clips and proto grid later, you can buy a "carrier board" for $10 to turn the BS1-IC in a normal-form-factor Stamp.

The third, least tangible benefit of the component-style package is professional appearance. You can integrate a factory-assembled and tested Stamp into a product without having a messy jumble of printed circuit boards under the hood. Put the BS1-IC right on the circuit board with your application-specific components.

The fourth advantage over the original Stamp design is not very glamorous, but nonetheless important: The BS1-IC has a reset input that's accessible to your circuitry. You can add a button between reset and ground. This allows you to reset a runaway program to the beginning. You can also treat the reset connection as an output, using it to reset other logic in your circuit.

In my first application for the BS1-IC, I wasn't too worried about looking good. I just wanted to reduce the jumble of jumper wires that was accumulating as I interfaced Stamps to different projects and test jigs around the shop. My Stamp Stretchers and LCD Serial Backpacks both borrow power from the Stamp and use a standard connector layout: +5V, ground, serial data. I decided to make a bunch of three-wire jumpers with female header sockets at each end. I then designed my own carrier board with eight, three-conductor male headers--one for each input/output pin.

Figure 2 shows the resulting board layout. I present it not as a finished product, but as an example of how simply you can turn Parallax' newest Stamp into Scott's Stamp or Mary's Stamp or Acme Industries' Stamp.

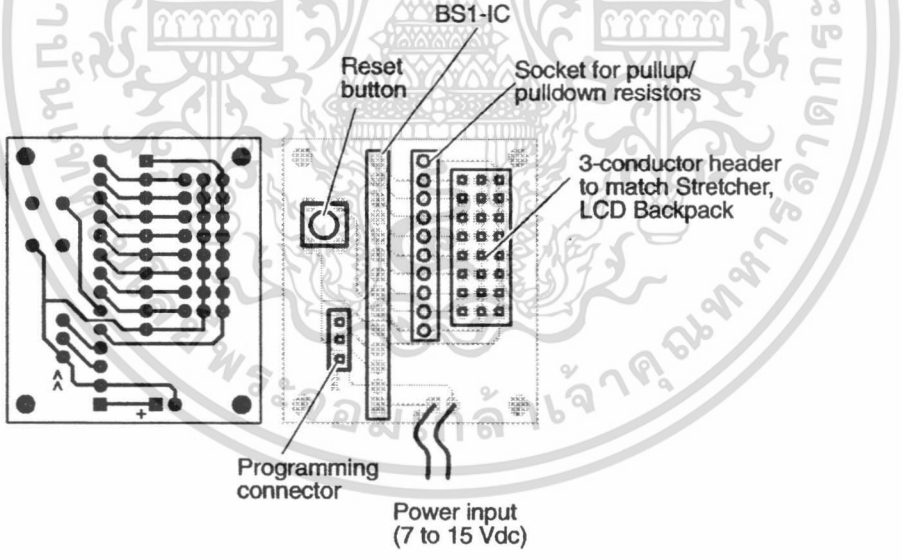


Figure 2. Example of a simple, homemade carrier board for the BS1-IC.

That wraps up this month's installment of Stamp Applications. Future columns will dig into Stamp hardware and software issues, generally presenting at least one schematic and one program listing each month. Initially, I'll take the topics from Parallax's technical support folks and the online venues where the Stamp is discussed. But I'd be grateful for your suggestions and questions. Contact me at:

Scott Edwards Electronics
964 Cactus Wren Lane
Sierra Vista, AZ 85635
phone: 602-459-4802; fax 602-459-0623
Internet: 72037.2612@compuserve.com
Compuserve: 72037,2612

Stamp Applications no. 2 (April '95):

Thermometer-on-a-Chip Simplifies Temperature Measurement

Using the DS1620, by Scott Edwards

DESIGNING a temperature-measurement application for the BASIC Stamp is a lot like voting; you end up selecting the lesser of three evils, shown in figure 1. Cost, calibration, or too many components make these solutions less than ideal.

Now, there's a fourth candidate. It's the Dallas Semiconductor DS1620 digital thermometer/thermostat chip, shown in figure 2. The DS1620 measures temperature in units of 0.5 degrees Centigrade from -55° to +125° C. The DS1620 is calibrated at the factory for exceptional accuracy: ±0.5° C from 0 to +70° C, which, is the Stamp's operating temperature range.

(For fans of the familiar Fahrenheit scale, those °C temperatures convert to: range, -67° to +257° F; resolution, 0.9° F; accuracy, ±0.9° F from 32° to 158° F.)

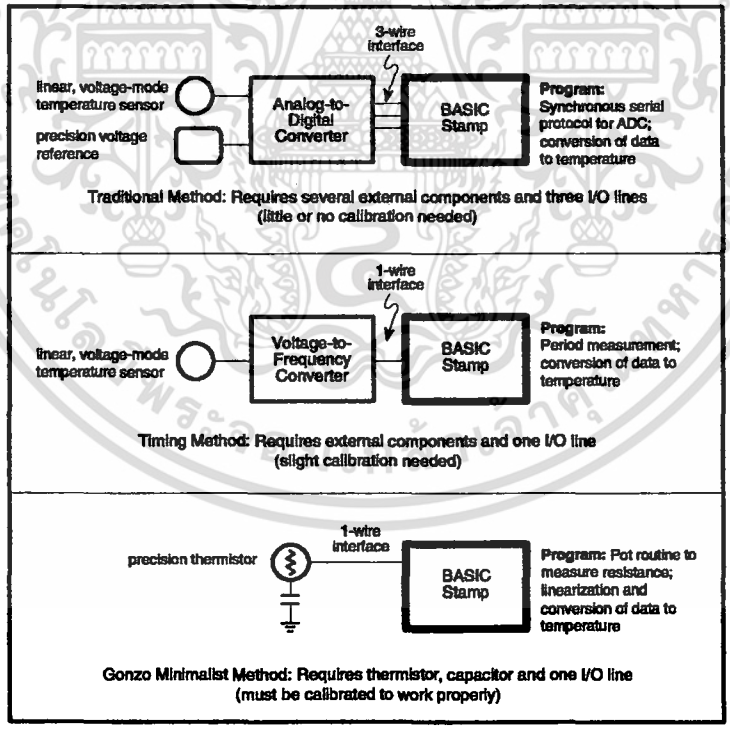


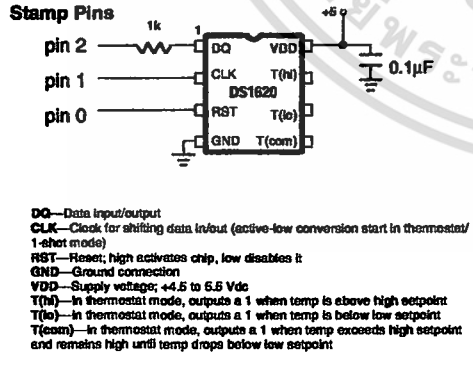
Figure 1. Three methods for measuring temperature with the Stamp.

The chip makes its temperature data available as a 9-bit number conveyed over a three-wire serial interface. The DS1620 can be set to operate continuously, taking one temperature measurement per second, or intermittently, conserving power by measuring only when told to.

The DS1620 can also operate as a standalone thermostat. A temporary connection to a Stamp or other controller establishes the mode of operation and high/low-temperature setpoints. Thereafter, the chip independently controls three outputs: T(high), which goes active at temperatures above the high-temperature setpoint; T(low), active at temperatures below the low setpoint; and T(com), which goes active at temperatures above the high setpoint, and stays active until the temperature drops below the low setpoint.

Let's concentrate on applications using the DS1620 as a Stamp peripheral, as shown in the listing. Later, we'll talk about adapting the code for configuring it as a thermostat.

Using the DS1620 requires sending a command (what Dallas Semi calls a "protocol") to the chip, then listening for a response (if applicable). The code under "DS1620 I/O Subroutines" in the listing shows how this is done. In a typical temperature-measurement application, the program will set the DS1620 to thermometer mode, configure it for continuous conversions, and tell it to start. Thereafter, all the program must do is request a temperature reading, then shift it in, as shown in the listing's *Again* loop.



DQ—Data input/output
CLK—Clock for shifting data in/out (active-low conversion start in thermostat/1-shot mode)
RST—Reset; high activates chip, low disables it
GND—Ground connection
VDD—Supply voltage; +4.5 to 5.5 Vdc
T(hi)—In thermostat mode, outputs a 1 when temp is above high setpoint
T(lo)—In thermostat mode, outputs a 1 when temp is below low setpoint
T(com)—In thermostat mode, outputs a 1 when temp exceeds high setpoint and remains high until temp drops below low setpoint

Figure 2. DS1620 pinout and connection.

The DS1620 delivers temperature data in a nine-bit, two's complement format, shown in the table. Each unit represents 0.5° C, so a reading of 50 translates to +25°C. Negative values are expressed as two's complement numbers. In two's complement, values with a 1 in their leftmost bit position are negative. The leftmost bit is often called the *sign* bit, since a 1 means - and a 0 means +.

To convert a negative two's complement value to a positive number, you must invert it and add 1. If you want to display this value, remember to put a minus sign in front of it.

Rather than mess with two's complement negative numbers, the program converts DS1620 data to an absolute scale called DSabs, with a range of 0 to 360 units of 0.5° C each. The Stamp can perform calculations in this all-positive system, then readily convert the results for display in °C or °F, as shown in the listing.

Going Further. Once you have configured the DS1620, you don't have to reconfigure it unless you want to change a setting. The DS1620 stores its configuration in EEPROM (electrically erasable, programmable read-only memory), which retains data even with the power off. In memory-tight Stamp applications, you might want to run the full program once for configuration, then strip out the configuration stuff to make more room for your final application.

If you want to use the DS1620 in its role as a standalone thermostat, the Stamp can help here, too. The listing includes protocols for putting the DS1620 into thermostat (*NoCPU*) mode, and for reading and writing the temperature setpoints. You could write a Stamp program to accept temperature data serially, convert it to nine-bit, two's complement format, then write it to the DS1620 configuration register. An example program that does this is available from the source listed below.

Be aware of the DS1620's drive limitations in thermostat mode; it sources just 1 mA and sinks 4 mA. This isn't nearly enough to drive a relay—it's barely enough to light an LED. You'll want to buffer this output with a Darlington

Nine-Bit Format for DS1620 Temperature Data

-Temperature-			-DS1620 Data-	
°F	°C	Binary	Hexadecimal	Decimal
+257	+125	0 11111010	00FA	250
+77	+25	0 00110010	0032	50
+32.9	+0.5	0 00000001	0001	1
+32	0	0 00000000	0000	0
+31.1	-0.5	1 11111111	01FF	511
-13	-25	1 11001110	01CE	462
-67	-55	1 10010010	0192	402

Example conversion of a negative temperature:
-25°C = 1 11001110 in binary. The 1 in the leftmost bit indicates that this is a negative number. Invert the lower eight bits and add 1: 11001110 -> 00110001 + 1 = 00110010 = 50. Units are 0.5°C, so divide by 2. Converted result is -25°C.

or MOSFET switch in serious applications.

For Your Convenience

Dallas Semiconductor offers data and samples of the DS1620 at reasonable cost. Call them at 214-450-0448 or contact your distributor. For a copy of the program listing (plus a thermostat programming example and PIC assembly language source code) on disk, a sample DS1620, and DS1620 documentation, you may order the DS1620 App Kit from Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone, 520-459-4802; fax 520-459-0623. Price is $20 postpaid. Visa, Mastercard, American Express, checks, and qualified purchase orders accepted.

```
' Program Listing For Interfacing To DS1620 Digital Thermometer
' Program: DS1620.BAS
' This program interfaces the DS1620 Digital Thermometer to the
' BASIC Stamp. Input and output subroutines can be combined to
' set the '1620 for thermometer or thermostat operation, read
' or write nonvolatile temperature setpoints and configuration
' data.
' ==================== Define Pins and Variables ================
SYMBOL  DQp = pin2          ' Data I/O pin.
SYMBOL  DQn = 2             ' Data I/O pin _number_.
SYMBOL  CLKn = 1            ' Clock pin number.
SYMBOL  RSTn = 0            ' Reset pin number.
SYMBOL  DSout = w0          ' Use bit-addressable byte for DS1620 output.
SYMBOL  DSin = w0           ' "   "   "          word "   "      input.
SYMBOL  clocks = b2         ' Counter for clock pulses.
' ==================== Define DS1620 Constants ==================
' >>> Constants for configuring the DS1620
SYMBOL  Rconfig = $AC       ' Protocol for 'Read Configuration.'
SYMBOL  Wconfig = $0C       ' Protocol for 'Write Configuration.'
SYMBOL  CPU = %10           ' Config bit: serial thermometer mode.
SYMBOL  NoCPU = %00         ' Config bit: standalone thermostat mode.
SYMBOL  OneShot = %01       ' Config bit: one conversion per start request.
SYMBOL  Cont = %00          ' Config bit: continuous conversions after start.
' >>> Constants for serial thermometer applications.
SYMBOL  StartC = $EE        ' Protocol for 'Start Conversion.'
SYMBOL  StopC = $22         ' Protocol for 'Stop Conversion.'
SYMBOL  Rtemp = $AA         ' Protocol for 'Read Temperature.'
' >>> Constants for programming thermostat functions.
SYMBOL  RhiT = $A1          ' Protocol for 'Read High-Temperature Setting.'
SYMBOL  WhiT = $01          ' Protocol for 'Write High-Temperature Setting.'
SYMBOL  RloT = $A2          ' Protocol for 'Read Low-Temperature Setting.'
SYMBOL  WloT = $02          ' Protocol for 'Write Low-Temperature Setting.'
' ==================== Begin Program ============================
' Start by setting initial conditions of I/O lines.
low RSTn                    ' Deactivate the DS1620 for now.
high CLKn                   ' Initially high as shown in DS specs.
pause 100                   ' Wait a bit for things to settle down.

' Now configure the DS1620 for thermometer operation. The
' configuration register is nonvolatile EEPROM. You only need to
' configure the DS1620 once. It will retain those configuration
' settings until you change them--even with power removed. To
' conserve Stamp program memory, you can preconfigure the DS1620,
' then remove the configuration code from your final program.
' (You'll still need to issue a start-conversion command, though.)
let DSout=Wconfig           ' Put write-config command into output byte.
gosub Shout                 ' And send it to the DS1620.
let DSout=CPU+Cont          ' Configure as thermometer, continuous conversion.
gosub Shout                 ' Send to DS1620.
low RSTn                    ' Deactivate '1620.
Pause 50                    ' Wait 50ms for EEPROM programming cycle.
let DSout=StartC            ' Now, start the conversions by
gosub Shout                 ' sending the start protocol to DS1620.
low RSTn                    ' Deactivate '1620.
```

```
' The loop below continuously reads the latest temperature data from
' the DS1620. The '1620 performs one temperature conversion per second.
' If you read it more frequently than that, you'll get the result
' of the most recent conversion. The '1620 data is a 9-bit number
' in units of 0.5 deg. C. See the ConverTemp subroutine below.
Again:
  pause 1000                ' Wait 1 second for conversion to finish.
  let DSout=Rtemp           ' Send the read-temperature opcode.
  gosub Shout
  gosub Shin                ' Get the data.
  low RSTn                  ' Deactivate the DS1620.
  gosub ConverTemp          ' Convert the temperature reading to absolute.
  gosub DisplayF            ' Display in degrees F.
  gosub DisplayC            ' Display in degrees C.
goto Again
' ====================== DS1620 I/O Subroutines ==================
' Subroutine: Shout
' Shift bits out to the DS1620. Sends the lower 8 bits stored in
' DSout (w0). Note that Shout activates the DS1620, since all trans-
' actions begin with the Stamp sending a protocol (command). It does
' not deactivate the DS1620, though, since many transactions either
' send additional data, or receive data after the initial protocol.
' Note that Shout destroys the contents of DSout in the process of
' shifting it. If you need to save this value, copy it to another
' register.
Shout:
high RSTn                   ' Activate DS1620.
output DQn                  ' Set to output to send data to DS1620.
for clocks = 1 to 8         ' Send 8 data bits.
  low CLKn                  ' Data is valid on rising edge of clock.
  let DQp = bit0            ' Set up the data bit.
  high CLKn                 ' Raise clock.
  let DSout=DSout/2         ' Shift next data bit into position.
next                        ' If less than 8 bits sent, loop.
return                      ' Else return.

' Subroutine: Shin
' Shift bits in from the DS1620. Reads 9 bits into the lsbs of DSin
' (w0). Shin is written to get 9 bits because the DS1620's temperature
' readings are 9 bits long. If you use Shin to read the configuration
' register, just ignore the 9th bit. Note that DSin overlaps with DSout.
' If you need to save the value shifted in, copy it to another register
' before the next Shout.
Shin:
input DQn                   ' Get ready for input from DQ.
for clocks = 1 to 9         ' Receive 9 data bits.
  let DSin = DSin/2         ' Shift input right.
  low CLKn                  ' DQ is valid after falling edge of clock.
  let bit8 = DQp            ' Get the data bit.
  high CLKn                 ' Raise the clock.
next                        ' If less than 9 bits received, loop.
return                      ' Else return.
```

```
' ================= Data Conversion/Display Subroutines ===============
' Subroutine: ConverTemp
' The DS1620 has a range of -55 to +125 degrees C in increments of 1/2
' degree. It's awkward to work with negative numbers in the Stamp's
' positive-integer math, so I've made up  a temperature scale called
' DSabs (DS1620 absolute scale) that ranges from 0 (-55 C) to 360 (+125 C).
' Internally, your program can do its math in DSabs, then convert to
' degrees F or C for display.
ConverTemp:
if bit8 = 0 then skip       ' If temp > 0 skip "sign extension" procedure.
  let w0 = w0 | $FE00       ' Make bits 9 through 15 all 1s to make a
                            ' 16-bit two's complement number.
skip:
  let w0 = w0 + 110         ' Add 110 to reading and return.
return
' Subroutine: DisplayF
' Convert the temperature in DSabs to degrees F and display on the
' PC screen using debug.
DisplayF:
let w1 = w0*9/10            ' Convert to degrees F relative to -67.
if w1 < 67 then subzF       ' Handle negative numbers.
  let w1 = w1-67
  Debug #w1, " F",cr
return
subzF:
  let w1 = 67-w1            ' Calculate degrees below 0.
  Debug "-",#w1," F",cr     ' Display with minus sign.
return

' Subroutine: DisplayC
' Convert the temperature in DSabs to degrees C and display on the
' PC screen using debug.
DisplayC:
let w1 = w0/2               ' Convert to degrees C relative to -55.
if w1 < 55 then subzC       ' Handle negative numbers.
  let w1 = w1-55
  Debug #w1, " C",cr
return
subzC:
  let w1 = 55-w1            ' Calculate degrees below 0.
  Debug "-",#w1," C",cr     ' Display with minus sign.
return
```

Stamp Applications no. 3 (May '95):

Adapt a Keypad for Pro-Quality Data Entry; Included Software Lets Stamp "Type" on a PC

Hacking a Commercial Keypad, by Scott Edwards

THIS month's applications make an off-the-shelf numeric keypad for PCs do double duty; first as a serial data-entry terminal for the Stamp, then as a sneaky software method of getting Stamp data into your favorite PC applications, like spreadsheets and word processors.

I've wanted to manufacture a Stamp-compatible keypad, but the cost of starting from scratch is daunting. A decent keypad requires good switches and a sturdy enclosure, both of which are very expensive in the small quantities that the Stamp aftermarket might use. There are occasional surplus bargains, but there's also the risk that the supply will dry up on the very day that Ultra-Mega Industries Inc. places a big order.

I recently found an accessory keypad for laptop computers—an ORTEK MCK-18S/N, made in Taiwan. It communicates with and is powered by a PC's serial port. Included software redirects data from the keypad to the keyboard buffer, so that it functions just like keys on the standard keyboard. The pad has 18 buttons, consisting of the numbers 0 through 9, decimal point, Escape, Num Lock, Enter, and the math operators (/, *, -, +). The keys are quality Alps switches with standard PC-style keycaps. The unit is enclosed in a nice flat-black case.

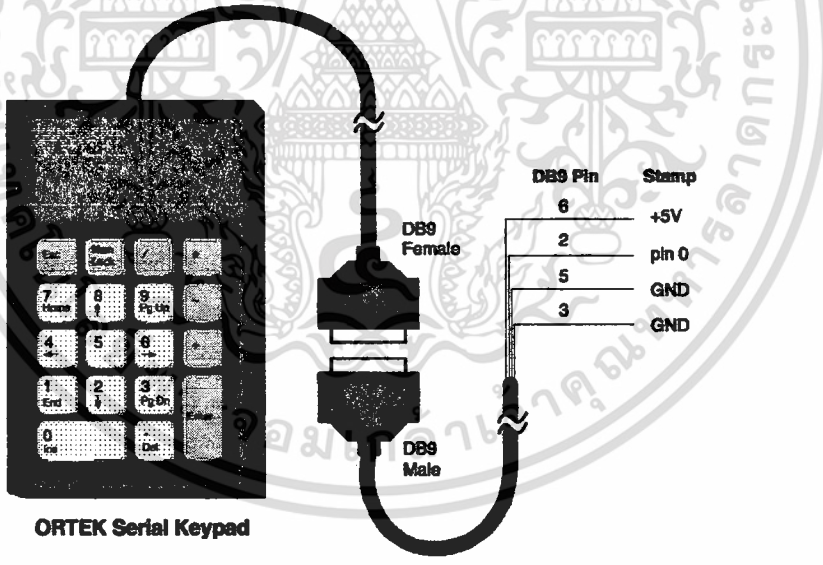


Figure 1. Connecting the serial keypad to the Stamp for data entry requires just one I/O pin plus a few microamps from the Stamp's power supply.

At $49 (see the source list), the keypad isn't cheap, but it's not unreasonably expensive either. So I bought one and proceeded to hack it. Here's what I found:

The heart of the keypad circuitry is a PIC microcontroller, just like the Stamp's own processor. This one runs at a leisurely 38.4 kHz; less than 1/100th the speed of a Stamp. This limits current draw to the 10s of microamps. It also helps the keypad meet FCC limits on radio interference.

When you press one of the pad's keys, its PIC sends a one-byte code (generally a lower-case letter) to the PC via an AT-style DB-9 serial connector. The serial data is sent at 1200 baud, with no parity, 8 data bits, and 1 stop bit (N81). If you hold down a key, the PIC waits a moment, then transmits a string of the same one-byte key code in rapid-fire fashion to simulate the "typeamatic" response of the standard keyboard. When you release a key, the PIC sends a different one-byte code (generally the upper-case version of the key-down code). Table 1 summarizes the codes.

Table 1. Key Codes Used by Unmodified ORTEK Keypad

Key	Key-Down (ASCII)	Key-Up (ASCII)
1	` (96)	@ (64)
2	a (97)	A (65)
3	b (98)	B (66)
4	c (99)	C (67)
5	d (100)	D (68)
6	e (101)	E (69)
7	f (102)	F (70)
8	g (103)	G (71)
9	h (104)	H (72)
Esc	i (105)	I (73)
/	j (106)	J (74)
*	k (107)	K (75)
-	l (108)	L (76)
+	m (109)	M (77)
. (point)	n (110)	N (78)
0 (zero)	o (111)	O (79)
Enter	t (116)	T (84)
Num Lock	y (121)	Y (89)

From an electrical standpoint, the keypad is wired to derive power from and send signals to the PC serial port. Its interface generates bipolarity RS-232 signals from the stolen port output voltages. However, the pad will work just fine with a single-ended 5-volt supply, like that of the Stamp. See table 2 for connections.

Table 2. ORTEK Keypad Wiring

DB-9	Color	PC Function	Stamp
2	orange	receive data	serial in
3	yellow	transmit data	ground
6	white	DSR handshake	+5 volts
5	black	ground	ground

Armed with nothing more than the information above, you could make use of the ORTEK keypad for Stamp data entry. Figure 1 shows the hookup, while listing 1 demonstrates how to make sense of the data. However, I couldn't leave well enough alone.

PC Keypad + PIC Program = "Stamp Pad"

The SERIN command can convert strings of ASCII text like "123" into equivalent numbers without the additional programming effort of listing 1. If only the keypad generated terminal-style output...

Instead of wishing, I picked up a soldering iron and removed the keypad's PIC controller. I probed the remaining keypad circuitry and determined the following:

- Like most keypads, this one is wired as a matrix of row and column connections. When a key is pressed, it shorts one row to one column. The processor looks up the row and column coordinates of the short in a table stored in ROM to translate it to a key code.
- The pad's column connections go to the four bits of PIC port RA, which are configured as inputs. (On the Stamp, RA is used to communicate with the PC and manage the EEPROM. It isn't accessible to the user.) When no key is pressed, all bits of RA are pulled high (reading 1s) by four resistors.
- The pad addresses the row connections with

the outputs of a 74HCT138 eight-channel multiplexer. This chip accepts a three-bit address from the lower bits of PIC port RB, and outputs a low on one of its eight outputs. Another bit of RB is used to activate and deactivate the '138. Bit RB.7 serves as the serial output. The remaining three bits of RB are unused.

Based on these observations, I coded a replacement PIC in assembly language. Thanks to the clever design of the hardware, this was extremely easy to do. Most of the effort went into preparing tables that unscrambled the row/column coordinate system of the pad, which was apparently designed to permit an inexpensive, one-sided circuit-board layout.

In addition to changing the data format with my new program, I added a hardware feature; a key-acknowledgment beeper. If an inexpensive piezo beeper is connected between pin RB.4 of the PIC and circuit ground, the beeper will make a short "bink" sound to acknowledge each key press. For applications that lack a display to confirm the numbers being entered, this is a tremendous help.

The modified keypad acts like a micro terminal that transmits one ASCII character per keypress (as shown in table 3). The characters correspond to the Stamp's internal table of the ASCII character set. That is, the following line of Stamp code will respond to the code sent by pressing the "*" key:

```
IF theKey = "*" THEN Asterisk
```

My keypad PIC program has a strict debounce function that requires keys to be pressed and released in a deliberate manner. Speed isn't compromised, as long as users don't roll from one key to the other. The first key has to be released before the next will register.

You can make your own Stamp Pad. Figure 2 shows details of the modification. To use the modified pad, connect it to the Stamp as shown in listing 1, then use Serin to collect the data. If, for instance, you want a 16-bit number to be stored to variable w1, use:

```
SERIN 0,N1200,#W1
```

Table 3. Key Codes for "Stamp Pad"

Key	Transmitted Text (ASCII value)
1	1 (49)
2	2 (50)
3	3 (51)
4	4 (52)
5	5 (53)
6	6 (54)
7	7 (55)
8	8 (56)
9	9 (57)
Esc	<ESC> (27)
/	/ (47)
*	* (42)
-	- (45)
+	+ (43)
. (point)	. (46)
0 (zero)	0 (48)
Enter	<RETURN> (13)
Num Lock	<SPACE> (32)

The # symbol before the variable name tells the Stamp to interpret up to five keystrokes as a number ranging from 0 to 65,535 (the range of a 16-bit number). The user must press Enter or any other non-numeric key to finish the entry. If you just want to know which key was pressed, use a byte variable without the #:

```
SERIN 0,N1200,B2
```

After the instruction executes, B2 will contain the ASCII value of the key pressed, as shown in table 3. If you have used SERIN before to accept data from a PC or other computer, this stuff is old hat by now.

You can obtain the Stamp Pad PIC from me, either by buying it outright, or by obtaining the source code free with purchase of my PIC Source Book. This is a collection of assembly language routines for the PIC that mimic the functions of the BASIC Stamp, helping users to move their programs into faster, more efficient PIC hardware. See the source box at the end of this column.

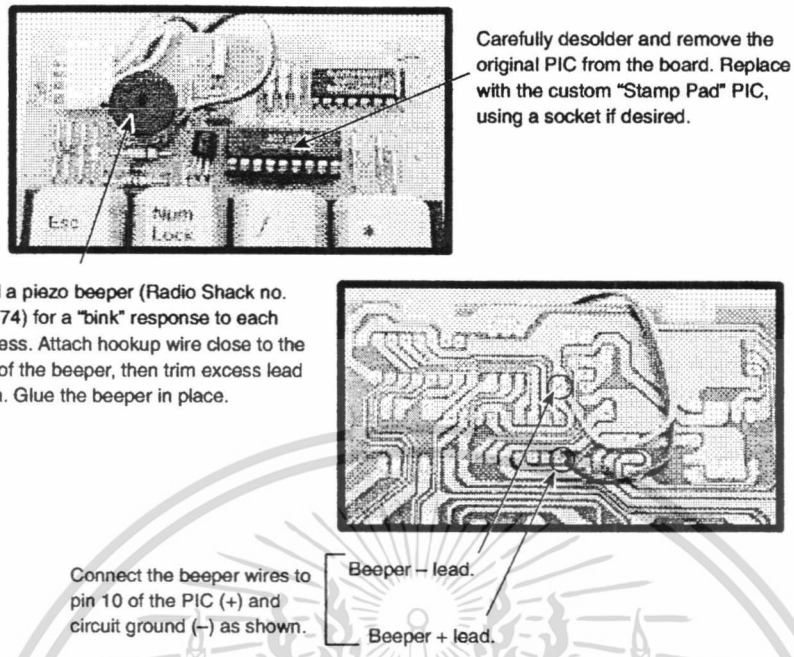


Figure 2. Details of the Stamp Pad modification.

But Wait, There's More!

There's an expression about sausage makers using "every part of the pig except the squeal." In that spirit, I couldn't ignore the software that came with the keypad. It redirects codes from the serial port to the keypad buffer, allowing you to use numeric input from the pad in any DOS or Windows application.

It occurred to me that if the Stamp were programmed to mimic the key codes, and the keypad software were installed, then the Stamp could "type" data at the keyboard. Think about it--Stamp-collected data magically appears in your spreadsheet, database, or word processor document. No fumbling with terminal software or file transfers.

There are general-purpose software utilities and hardware devices sold for this very purpose, with prices from $100 to $500. They accept any kind of serial input, not just numbers. But if numbers are all you need, check out listing 2. To use the program, install the keypad software according to the instructions that come with the pad. Temporarily disable the keypad driver by typing "KEYPAD OFF" at the DOS prompt. Connect a Stamp programmed with listing 2 as shown in table 4, but don't connect power to the Stamp yet. Next, turn on the keypad driver by typing "KEYPAD" at the DOS prompt.

Table 4. Stamp-to-PC Connections for Listing 2

Pin	DB9	DB25
pin 0	2	3
GND	5	7

You're ready to go. For a test drive, boot the BASIC Stamp host program STAMP.EXE, but don't load a program. Reconnect power to the Stamp. A column of numbers will be typed onto the screen. That's the Stamp, communicating with your keyboard buffer through the keypad software. Now you can write data-acquisition programs for the Stamp that can communicate directly with any piece of PC software you own!

By the way, the reason for the somewhat roundabout setup procedure above is to avoid trashing your work. When I wrote the program in listing 2, I already had the keypad driver resident in my PC's memory, and the Stamp connected to the serial port. As soon as I ran the program, it began typing into my just-finished Stamp program listing. I had to quickly

disconnect the Stamp, and erase all of the numbers it had added to the listing.

Conclusion

If you're interested in other keypad solutions for the Stamp, make sure to get Parallax application note no. 3, which shows how to connect a 74C922 16-key pad and an LCD to the Stamp. Looking for an interesting application for our serial keypad? Try Parallax application note no. 6, a stepper-motor driver that can be controlled directly from the Stamp Pad.

Sources

The ORTEK keypad is available from Jameco (part no. 114841) for $49.95, plus applicable shipping and handling charges. To order or request a catalog: Jameco Electronic Components, 1355 Shoreway Drive, Belmont, CA 94002-4100; phone 1-800-831-4242 or 415-592-8097.

The Stamp Pad PIC required to modify the keypad is $10 postpaid from Scott Edwards Electronics. Source code for the Stamp Pad PIC is free with purchase of *The PIC Source Book*, a cookbook of assembly language PIC routines based on the instruction set of the BASIC Stamp. Prototype your ideas with the Stamp, then convert them into fast, professional, one-chip solutions with the help of the cut-and-paste routines from the *Source Book*. Price is $39 postpaid. Order from Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623. Use Visa or Mastercard for phone/fax orders; checks, money orders, or POs from approved institutions also accepted. CODs and foreign orders incur additional charges.

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101.

```
' Listing 1: OR_KEYS.BAS (interpret ORTEK MCK-18S/N keypad codes)
' This program accepts serial data from the ORTEK MCK-18S/N numeric
' keypad and converts it into a 16-bit value in variable w1. Users
' must be careful not to hold keys down, or they will activate the
' pad's autorepeat function, causing entry errors.

' Main program loop.
Loop:
gosub GetKeys                ' Getkeys does all the work.
debug w1                     ' Show the result on the PC screen.
goto Loop                    ' Do it forever.

' Subroutine to receive the serial data, filter out extraneous key-up
' codes, and convert received data bytes to a 16-bit value. GetKeys
' will keep accepting and interpreting digits until the user presses
' a non-numeric key, like <Enter>. The routine takes advantage of the
' fact that the key codes for the numbers 1-9 are sequential;
' subtracting 95 from them leaves you with the number itself.
' Although0 (zero) is out of sequence, an additional IF/THEN
' statement recognizes its code ("o").
GetKeys:
let w1 = 0                   ' Clear w1 to start.
Again:
 Serin 0,N1200,b0            ' Get code from the keypad.
 if bit5 = 0 then Again      ' Bit5 is 0 in key-up codes; ignore em.
 if b0 <> "o" then skip      ' Change 0 code from "o" (111) to 95.
 let b0 = 95
skip:
 if b0 > 104 then done       ' Non-numeric key pressed; we're done.
 let b0 = b0 - 95            ' Otherwise convert to 0-9, multiply
 let w1 = w1 * 10 + b0       ' old total by 10, add new value, and
goto Again                   ' get the next digit.
done:
 return                      ' Done: return to main program.
```

```
' Listing 2: FAKE_PAD.BAS (imitate the ORTEK keypad codes)
' This program mimics the codes produced by the ORTEK keypad,
' allowing a PC running the ORTEK software to receive Stamp data
' to its keyboard buffer. The Stamp "types" directly into programs
' that are incapable of normal serial input. To demonstrate this
' capability, the Stamp will count upward by 1 and type the result
' of each calculation to the PC. Remember that the keypad software
' must be installed on the PC for this to work. Before running
' this program, make sure that this program is saved, since the
' Stamp may begin typing numbers into it, if the keypad software
' is active.

SYMBOL nonZero = bit0       ' Leading-zero suppression flag.
SYMBOL code = b1            ' Key code to send.
SYMBOL decade = w2          ' Power-of-10 divisor for conversion.
SYMBOL count = w3           ' Counter for demo.

' Main program loop.
Loop:
 let w1 = count             ' Transfer value of copy to w1.
 gosub typeData             ' "Type" the data to PC.
 pause 100                  ' Wait briefly
 let count = count + 1      ' Increment counter.
goto Loop                   ' Do it forever.

' Subroutine to convert the value stored in w1 to ORTEK keypad codes.
typeData:
let nonZero = 0             ' Clear flag that indicates first non-0 digit.
let decade = 10000          ' Start with highest digit of w1.
nextDigit:
 let code = w1/decade       ' Get value of current digit.
 let w1 = w1//decade        ' Leave remainder in w1.
 if code=0 AND nonZero=0 AND decade <> 1 then skip3   ' No leading 0s.
 if code=0 then skip1
 let nonZero = 1
 goto skip2
skip1:
 let code = 16              ' Code for 0, minus 95.
skip2:
 let code = code + 95
 serout 0,N1200,(code)      ' Send key-down code of digit.
 let bit13 = 0              ' Clear bit5 of b1 (bit13) for key-up code.
 serout 0,N1200,(code)      ' Send key-up code.
skip3:
 let decade = decade/10     ' Get ready for next lower digit
 if decade > 0 then nextDigit
 serout 0,N1200,("tT")      ' Done. Send <Enter> key.
return
```

Stamp Applications no. 4 (June '95):

High-Precision Measurement Made Easy With New 12-bit Analog-to-Digital Converter

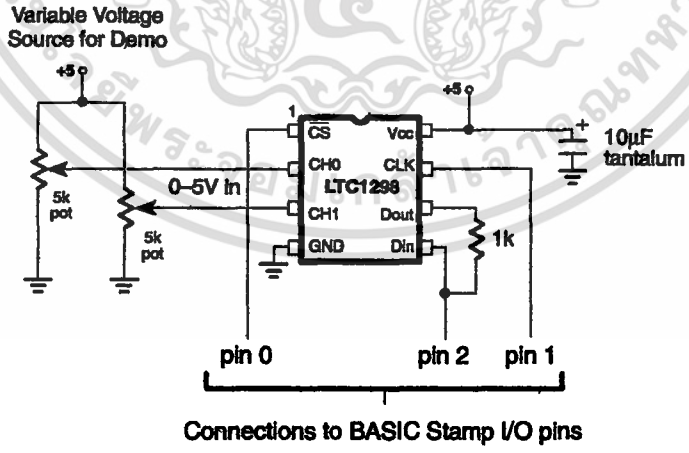
Using the LTC1298, by Scott Edwards

MANY popular applications for the Stamp include analog measurement, either using the built-in Pot (resistance measurement) command or an external analog-to-digital converter (ADC). These measurements are limited to eight-bit resolution, meaning that a 5-volt full-scale measurement would be broken into units of 5/256 = 19.5 millivolts (mV).

That sounds pretty good until you apply it to a real-world sensor. Take the LM34 and LM35 temperature sensors as an example. They output a voltage proportional to the ambient temperature in degrees Fahrenheit (LM34) or Centigrade (LM35). A 1-degree change in temperature causes a 10-mV change in the sensor's output voltage. An eight-bit conversion gives lousy 2-degree resolution. By reducing the ADC's range, or amplifying the sensor signal, you can improve resolution at the expense of more components and a less-general design.

The easy way out is to switch to an ADC with 10- or 12-bit resolution. Until recently, that hasn't been a decision to make lightly, since more bits = more bucks. However, the new LTC1298 12-bit ADC is reasonably priced at less than $10, and gives your Stamp projects two channels of 1.22-millivolt resolution data. It's available in a Stamp-friendly 8-pin DIP, and draws about 250 microamps (uA) of current.

The figure shows how to connect the LTC1298 to the Stamp, and the listing supplies the necessary driver code.



Pinout and connection details for the LTC1298.

If you have used other synchronous serial devices with the Stamp, such as EEPROMs, other ADCs or the DS1620 thermometer described in a previous column, there are no surprises here. We have tied the LTC1298's data input and output together to take advantage of the Stamp's ability to switch data directions on the fly. The resistor limits the current flowing between the Stamp I/O pin and the 1298's data output in case a programming error or other fault causes a "bus conflict." This happens when both pins are in output mode and in opposite states (1 vs. 0). Without the resistor, such a conflict would cause large currents to flow between pins, possibly damaging the Stamp and/or ADC.

You may have noticed that the LTC1298 has no voltage-reference (Vref) pin. The voltage reference is what an ADC compares its analog input voltage to. When the analog voltage is equal to the reference voltage, the ADC outputs its maximum measurement value; 4095 in this case. Smaller input voltages result in proportionally smaller output values. For example, an input of 1/10th the reference voltage would produce an output value of 409.

The LTC1298's voltage reference is internally connected to the power supply at pin 8. This means that a full-scale reading of 4095 will occur when the input voltage is equal to the power-supply voltage, nominally 5 volts. Notice the weasel word "nominally," meaning "in name only." The actual voltage at the +5-volt rail of the full-size (pre-BS1-IC) Stamp with the LM2936 regulator can be 4.9 to 5.1 volts initially, and can vary by 30 mV.

In some applications you'll need a calibration step to compensate for the supply voltage. Suppose the LTC1298 is looking at a source of 2.00 volts. If the supply is 4.90 volts, the LTC1298 will measure (2.00/4.90) * 4095 = 1671. If the supply is at the other extreme, 5.10 volts, the LTC1298 will measure (2.00/5.10) * 4095 = 1606.

How about that 30-millivolt deviation in regulator performance, which cannot be calibrated away? If calibration makes it seem as though the LTC1298 is getting a 5.000-volt reference, a 30-millivolt variation means that the voltage would vary 15 millivolts high or low. Using the 2.00-volt example, the LTC1298 measurements can range from (2.00/4.985) * 4095 = 1643 to (2.00/5.015) * 4095 = 1633.

The bottom line is that the measurements you make with the LTC1298 will be only as good as the stability of your +5-volt supply.

I suppose the reason for leaving off a separate voltage-reference pin was to make room for the chip's second analog input. The LTC1298 can treat its two inputs as either separate ADC channels, or as a single, differential channel. A differential ADC is one that measures the voltage difference between its inputs, rather than the voltage between one input and ground.

A final feature of the LTC1298 is the sample-and-hold capability. At the instant your program requests data, the ADC grabs and stores the input voltage level in an internal capacitor. It measures this stored voltage, not the actual input voltage.

By measuring this voltage snapshot, the LTC1298 avoids the errors that can occur when an ADC tries to measure a changing voltage. Without going into the gory details, most common ADCs are *successive approximation* types. That means that they zero in on a voltage measurement by comparing a guess to the actual voltage, then determining whether the actual is higher or lower. They formulate a new guess and try again. This game becomes very difficult if the voltage is constantly changing!

ADCs that aren't equipped with sample-and-hold circuitry should not be used to measure noisy or fast-changing voltages. The LTC1298 has no such restriction.

The program listing is thoroughly commented, so I won't waste ink by repeating that stuff here. Instead, I want discuss a subtlety that trips up many Stamp users: the difference between an I/O pin *number* and a pin *variable*.

Look at the beginning of the listing, under ADC Interface Pins. We've assigned names (SYMBOLs) to each of the pins that the Stamp uses to talk to the LTC1298. CS is 0, CLK is 1, and DIO (data in/out) is 2. DIO actually has two SYMBOLs: DIO_n (2) and DIO_p (pin2). Why?

PBASIC refers to the I/O pins in one of two ways; as numbers (0 through 7) or as bit

variables with preassigned names (pin0 through pin7). The following commands use pin numbers:

High, Low, Toggle, Input, Output, Reverse, Pot, Pulsin, Pulsout, PWM, Serin, Serout, Sound

Take High for example. The following are valid PBASIC commands:

```
High 3          ' Make pin 3 output high.
High b2         ' Make the pin number (0-7)
                ' contained in b2 output high.
High bit7       ' Make pin 0 high if bit7 = 0;
                ' make pin 1 high if bit7 = 1.
```

You can specify the pin as either a constant like "3" or as a variable like b2 or bit7.

On the other hand, the math and logic instructions look at the pins as bit variables with a value of 0 or 1. Suppose pin 1 is set up as an input, and the following instruction executes:

```
Let b2 = b2 + pin1   ' Add pin 1 to b2.
```

If there's a 0 at pin 1, 0 is added to b2. If there's a 1 at pin 1, 1 is added to b2. This example points out why we sometimes refer to pins by their numbers, and sometimes by their variable names. Would the following instruction have the same result as the one above?

```
Let b2 = b2 + 1      ' Add 1 to b2.
```

Nope. We have to make the distinction between *1* and *pin1*.

Back to the question I originally posed: Why assign two SYMBOLs for the DIO pin? The reason is that this pin is used with High, which requires a pin number, and also with a Let expression, which requires a variable name. The listing's convention for this is to add the ending "_n" for the number and "_p" for the pin-variable name.

The common bug that arises from the two ways of referring to pins looks like this:

```
High pin3     ' Usually a bug!
```

The programmer believes that he or she is making pin 3 output high. PBASIC interprets this line to mean, "if pin3 = 0 then make pin 0 high; if pin3 = 1 then make pin 1 high." Probably not what the programmer had in mind.

Conclusion. The LTC1298 is a Stamp-friendly peripheral for precision sensing applications. It's available from Digi-Key (800-344-4539) for $8.89 in single quantity as part number LTC1298CN8-ND (8-pin DIP) or LTC1298CS8-ND (surface-mount). Be sure to request a data sheet or order the data book (9210B-ND, $9.95) when you order.

For a copy of the program listing (plus a thermostat programming example and PIC assembly language source code) on disk, a sample LTC1298, and LTC1298documentation, you may order the LTC1298App Kit from Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone, 520-459-4802; fax 520-459-0623. Price is $25 postpaid. Visa, Mastercard, American Express, checks, and qualified purchase orders accepted.

```
' Program: LTC1298 (LTC1298 analog-to-digital converter)
' The LTC1298 is a 12-bit, two-channel ADC. Its high resolution, low
' supply current, low cost, and built-in sample/hold feature make it a
' great companion for the Stamp in sensor and data-logging applications.
' With its 12-bit resolution, the LTC1298 can measure tiny changes in
' input voltage; 1.22 millivolts (5-volt reference/4096).
' ============================================================
'                       ADC Interface Pins
' ============================================================
' The 1298 uses a four-pin interface, consisting of chip-select, clock,
' data input, and data output. In this application, we tie the data lines
' together with a 1k resistor and connect the Stamp pin designated DIO
' to the data-in side of the resistor. The resistor limits the current
' flowing between DIO and the 1298's data out in case a programming error
' or other fault causes a "bus conflict." This happens when both pins are
' in output mode and in opposite states (1 vs 0). Without the resistor,
' such a conflict would cause large currents to flow between pins,
' possibly damaging the Stamp and/or ADC.
SYMBOL   CS = 0          ' Chip select; 0 = active.
SYMBOL   CLK = 1         ' Clock to ADC; out on rising, in on falling edge.
SYMBOL   DIO_n = 2       ' Pin _number_ of data input/output.
SYMBOL   DIO_p = pin2    ' Variable_name_ of data input/output.
SYMBOL   ADbits = b1     ' Counter variable for serial bit reception.
SYMBOL   AD = w1         ' 12-bit ADC conversion result.
' ============================================================
'                       ADC Setup Bits
' ============================================================
' The 1298 has two modes. As a single-ended ADC, it measures the
' voltage at one of its inputs with respect to ground. As a differential
' ADC, it measures the difference in voltage between the two inputs.
' The sglDif bit determines the mode; 1 = single-ended, 0 = differential.
' When the 1298 is single-ended, the oddSign bit selects the active input
' channel; 0 = channel 0 (pin 2), 1 = channel 1 (pin 3).
' When the 1298 is differential, the oddSign bit selects the polarity
' between the two inputs; 0 = channel 0 is +, 1 = channel 1 is +.
' The msbf bit determines whether clock cycles _after_ the 12 data bits
' have been sent will send 0s (msbf = 1) or a least-significant-bit-first
' copy of the data (msbf = 0). This program doesn't continue clocking after
' the data has been obtained, so this bit doesn't matter.
' You probably won't need to change the basic mode (single/differential)
' or the format of the post-data bits while the program is running, so
' these are assigned as constants. You probably will want to be able to
' change channels, so oddSign (the channel selector) is a bit variable.
SYMBOL   sglDif = 1      ' Single-ended, two-channel mode.
SYMBOL   msbf = 1        ' Output 0s after data transfer is complete.
SYMBOL   oddSign = bit0  ' Program writes channel # to this bit.
```

```
' ============================================================
'                       Demo Program
' ============================================================
' This program demonstrates the LTC1298 by alternately sampling the two
' input channels and presenting the results on the PC screen using Debug.
high CS                   ' Deactivate the ADC to begin.
Again:                    ' Main loop.
  For oddSign = 0 to 1    ' Toggle between input channels.
    gosub Convert         ' Get data from ADC.
    debug "ch ",#oddSign,":",#AD,cr ' Show the data on PC screen.
    pause 500             ' Wait a half second.
  next                    ' Change input channels.
goto Again                ' Endless loop.

' ============================================================
'                       ADC Subroutine
' ============================================================
' Here's where the conversion occurs. The Stamp first sends the setup
' bits to the 1298, then clocks in one null bit (a dummy bit that always
' reads 0) followed by the conversion data.
Convert:
  low CLK                 ' Low clock--output on rising edge.
  high DIO_n              ' Switch DIO to output high (start bit).
  low CS                  ' Activate the 1298.
  pulsout CLK,5           ' Send start bit.
  let DIO_p = sglDif      ' First setup bit.
  pulsout CLK,5           ' Send bit.
  let DIO_p = oddSign     ' Second setup bit.
  pulsout CLK,5           ' Send bit.
  let DIO_p = msbf        ' Final setup bit.
  pulsout CLK,5           ' Send bit.
  input DIO_n             ' Get ready for input from DIO.
  let AD = 0              ' Clear old ADC result.
  for ADbits = 1 to 13    ' Get null bit + 12 data bits.
    let AD = AD*2+DIO_p   ' Shift AD left, add new data bit.
    pulsout CLK,5         ' Clock next data bit in.
  next                    ' Get next data bit.
  high CS                 ' Turn off the ADC
return                    ' Return to program.
```

Stamp Applications no. 5 (July '95):

Checking Battery Condition and Multiplexing I/O Lines

Two Mini Applications, by Scott Edwards

THIS month's first application was contributed by Guy Marsden of ART TEC, Oakland, California. Guy, a former visual effects specialist for the movies (Star Trek the Motion Picture, Ghostbusters, 2010) makes his living helping artists incorporate electronics into their work. One such work was powered by 12-volt lead-acid battery, and Guy devised a simple, effective way for the Stamp to monitor the battery and sound an alarm at charging time.

Figure 1 and listing 1 show Guy's method. He's using the brightness of the LED as a relative indication of battery voltage. The Stamp reads this brightness as a variable resistance across the photocell. When the photocell resistance exceeds a preset limit, the Stamp sounds an alarm.

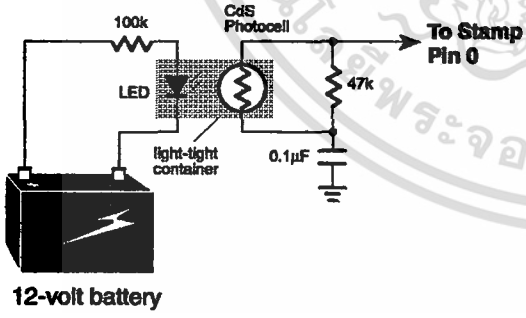


Figure 1. Battery-monitor setup.

Although Guy used a commercially made optoisolator, you should get similar results with a roll-your-own version. Just mount an LED facing a photocell and cover the assembly to block outside light.

This approach can be used more generally to convert a variable voltage into a form that can be read with the Pot command. Just keep a couple of LED characteristics in mind:

• LEDs have a relatively high forward voltage of approximately 1.5 to 2.1 volts. They don't light at all if the input voltage is below their forward voltage. They require a series resistor to make sure that the current through them is reasonable (usually up to 25 mA continuous). To calculate the value of this resistor, use the following formula:

Series Resistor = (Input Voltage–LED Forward Voltage) / LED current

For example, suppose your input is 9 volts. Just guessing, you figure the LED forward voltage at 1.9 volts. And you decide that 10 milliamperes is a safe bet for current. (Remember that most electronic formulas involving current want the value in amperes; 10 mA = 0.01 amperes.) The series resistor should be (9-1.9)/0.01 = 710 ohms. Pick the closest standard value (or the closest value that you have on hand) and you're done.

• An LED's forward voltage decreases with temperature, increasing the current that will pass through it with a given series resistance. More current usually means more light output, but brightness decreases with temperature. To make matters worse, data isn't available for most common LEDs. Even when data is

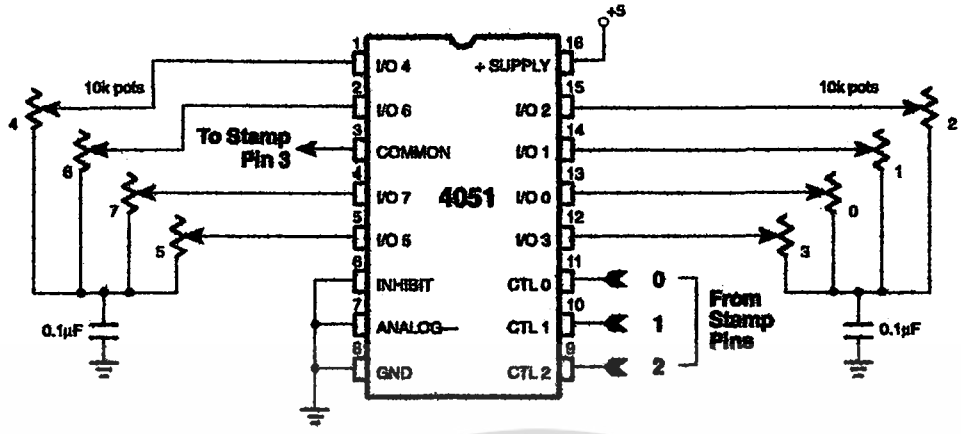


Figure 2. Four Stamp pins read eight pots through a 4051 analog multiplexer.

available, it doesn't usually provide much detail on these effects.

So I'm wondering: Is Guy's circuit usable across a wide range of temperatures, or is it restricted to the great indoors? I started to set up an experiment to find out, then decided that a contest would be more fun.

Here's the deal: Design and construct a Stamp project to test the output of Guy's circuit over a range of at least 32 to 90 degrees F with a steady input voltage. (Temperature control can be as simple as an ice bucket in which the ice is allowed to melt.) Send me a drawing of the circuit, the BASIC program listing, a brief description of your procedure, and a copy of the data generated by it. I'll accept entries until October 31, 1995.

The reader who submits the best entry (my call) will receive the newest version of the LCD Serial Backpack display with 16x1 LCD. This latest model features switchable 2400- or 9600-baud operation, low-voltage reset circuit, improved contrast control, and an easy-to-read 16-character display. All runners-up will receive a coupon good for 10 percent off anything I sell; see the Sources box.

Double Your I/O

The next application is for those of you who can never have enough input/output (I/O) lines. Although I'm demonstrating it with the Pot command, the same method will work with almost any Stamp input or output statment.

Take a look at figure 2 and listing 2. It's based around a 4051 multiplexer/demultiplexer chip. This device works like a digitally controlled rotary switch, shown conceptually in figure 3. Depending on the binary number on its control bits, the chip connects one of eight I/Os to a single common pin. This connection is analog and bidirectional, so you can use it for input, output, or both. The Pot command is a good example of both, since it switches back and forth between input and output as it measures the time required to discharge a capacitor through an unknown resistance.

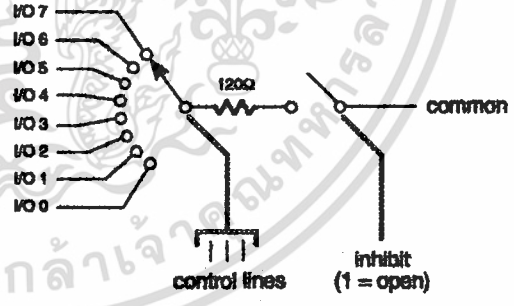


Figure 3. The 4051 multiplexer works like a digitally controlled rotary switch. The three control bits select I/O 0 (000 binary) through 7 (111 binary).

For an investment of four pins—three control bits and one common—you receive a dividend of eight I/Os. Better than Wall Street most days.

There are a few limitations to this trick. The first is evident if you try the pot demonstration. You can never quite get a reading of zero; the

lowest the readings will go is about 8. This is because the 4051's electronic rotary switch isn't perfect--it has a resistance of approximately 120 ohms.

A second limitation is that inputs to the 4051 must never exceed 0.5 volts above the supply voltage or below ground. This means that the 4051 can't be used to directly receive RS-232 serial signals through a series resistor as the Stamp's I/O pins can.

The last restriction is common to both the 4051 and the Stamp, so it really isn't much of a limitation: Current through the 4051 should never exceed 25 mA.

If you like the 4051, but still need more I/O, try the 4067. It provides 16 I/Os for five Stamp pins. Check your favorite data book (such as the CMOS Cookbook by Don Lancaster) for details.

Sources

Enter the LCD Serial Backpack contest and win a new LCD display for your Stamp projects (a $40 value) or 10 percent off Stamp accessories and microcontroller projects seen here in Nuts & Volts. Entries (or questions, suggestions, requests) to: Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623; e-mail (Compuserve) at 72037,2612; or via Internet 72037.2612@compuserve.com.

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101.

```
' Listing 1. BASIC Stamp Battery Monitor
' CHEAP AND SIMPLE LOW BATTERY WARNING
' Guy Marsden, March 1995.  tekart@well.com
'
' Uses a Cds opto coupler with LED input such as: CLM6000.
' Put a 33 or 47K resistor across cell and a .1uF cap to gnd.
' Use a high value resistor in series with the LED wired directly
' to the battery.  I used 100k for a 12volt system.  This value is
' enough to produce a resistance of 20k at nominal battery voltage.
'
' When the voltage drops, the Cds resistance increases.  By setting
' the SCALE of the POT function under a low battery condition, you
' will then have a range to work with.  Determine your setpoint
' using a variable bench supply and a DVM.

symbol Batt = b2
symbol LoBatt = 220

CheckBatt:
        pot 0,76,Batt           ' check battery voltage
        if Batt > LoBatt Alarm      ' if less than established value
goto CheckBatt

Alarm:
        sound 1,100,100         ' beep piezo alarm
        pause 100
goto Alarm
```

```
' Listing 2. Multiplexing Stamp I/O Lines
' Program: MULTIPOT.BAS (Multiple pots using a 4051 multiplexer)
' This program demonstrates how to connect and measure multiple
' pots using a 4051 multiplexer chip. The 4051's control inputs
' (11,10,9) connect to Stamp pins 0, 1, and 2 respectively.
' The common I/O pin (3) of the 4051 goes to Stamp pin 3.
' By writing a value between 0 and 7 to its pins, the Stamp can
' select one of eight variable-resistance inputs through the
' 4051. See the schematic for details.

SYMBOL pot_sel = b2             ' Pot number 0-7 selected through 4051.
SYMBOL pot_val = b3             ' Result of the pot measurement.

Let dirs = %0111                ' Make the lower 3 pins outputs to drive 4051.
Again:
  for pot_sel = 0 to 7          ' For each of the eight pots:
   let pins = pot_sel           ' Write pot number to the 4051.
   pot 3,150,pot_val            ' Perform pot measurement on selected pot.
   debug "pot #",#pot_sel," ",#pot_val,cr  ' Display result.
  next pot_sel                  ' Read the next pot.
  pause 2000                    ' Wait two seconds.
  debug cr                      ' Insert a carriage return on screen.
goto Again                      ' Do it again (endless loop).
```

Stamp Applications no. 6 (August '95):

Silicon Steroids for the Stamp Help Your Projects Heft Big Loads

Using Switching Trasistors, by Scott Edwards

ONE of the outstanding characteristics of the PIC microcontroller used in the BASIC Stamp is its ability to directly drive loads like LEDs through its input/output pins. The Stamp can source (conduct to +5) up to 20 mA and sink (conduct to ground) up to 25 mA. Total current sourced or sunk by all eight pins should not exceed 40 or 50 mA, respectively.

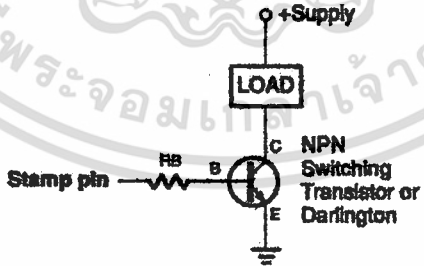
Now I'll grant that 25 mA doesn't sound like a lot of current. But shop around. Other microcontrollers make a big deal out of having a few "high-current" pins capable of sourcing 2 mA and sinking 10.

OK, so the Stamp is a muscle-bound brute by microcontroller standards. It still gets sand kicked in its face by applications that need to drive motors, incandescent bulbs, relays, solenoids, etc. In today's column we're going to pump up the Stamp to new levels of power.

Let's start with the basic transistor switch. To keep matters simple, we will limit our discussion to current-sink capability--switching current to ground.

The tools for the job are simple and easy to obtain; a resistor and an NPN transistor. Figure 1 shows the capabilities of a common 2N2222, a high-gain transistor, and a low-power Darlington transistor.

In figure 1, you can think of the collector (C) and emitter (E) of the transistor as forming a switch to ground. Current at the base turns the switch on. If Stamp pin 0 were connected to this circuit, the instruction High 0 would turn on current to the load; Low 0 would turn it off.



Transistor	RB	Base Current	Load Current	Max +Supply	C-E Voltage Drop	Remarks
2N2222	390Ω	11 mA	100 mA	30 V	0.5 V	Common
ZTX689B	390Ω	11 mA	2 A	12 V	0.5 V	High-gain
ZTX605	3.3k	1 mA	1 A	100 V	1.5 V	Darlington

Figure 1. Simple one-transistor switch boosts the Stamp's current-switching capability.

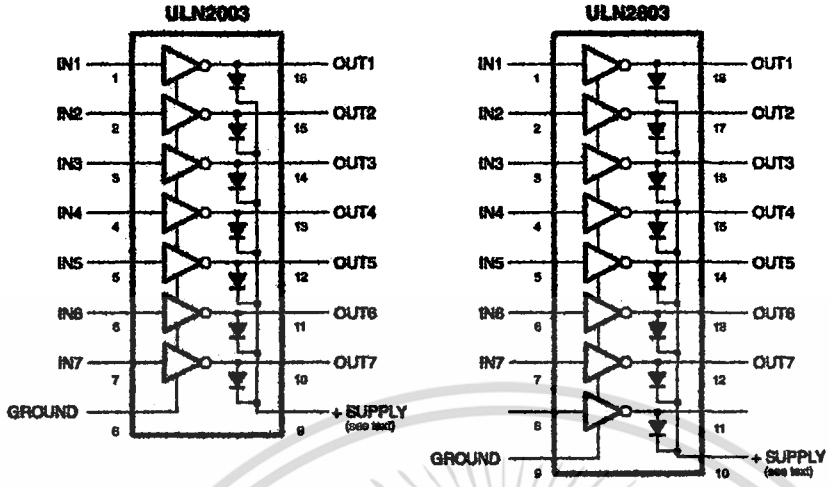


Figure 2. Handy power switches on-a-chip: the ULN 2x03 ICs.

Transistors switches are good, but not perfect. See the column in figure 1 that says C-E Voltage Drop? The voltage across the load will be that much smaller than the supply voltage. For example, if you're driving a flashlight bulb with a 2N2222 transistor and a 6-volt supply, the bulb will actually see only 5.5 volts. The other half volt will be lost, "dropped," across the transistor.

The voltage drop, multiplied by the current drawn by the load, gives you the power in watts wasted by the switching transistor. Where does the wasted power go? It produces heat, sometimes lots of it. For example, the Zetex ZTX689B can conduct as much as 2 amperes of current with a 0.5-volt drop, losing 1 watt to waste heat in the process. When you consider that a small soldering iron has a 15-watt heating element, you can imagine how hot that little transistor can get. Larger switching transistors have metal tabs on their cases for attachment to heat sinks. The large surface area and other properties of the heat sink help spread and dissipate all that waste heat. (A 15-watt soldering iron is very hot; an electric blanket with the same wattage is barely warm.)

All three transistors in the figure-1 table are compatible with Stamp output capabilities, but the third--the Darlington transistor--looks especially good from the Stamp's point of view. It requires only 1 mA to drive a 1-ampere load. But whoa, the C-E voltage drop is terrible at 1.5 volts. Can't we get a Darlington with better C-E specs?

In a word, no. Darlingtons consist of two NPN transistor in the same case wired in a way that multiplies their overall gain (ratio of current in to current controlled). In the process, the Darlington adds one base-to-emitter junction worth of voltage drop, approximately 0.7 volts, to the C-E drop.

Despite this drawback, Darlingtons are so handy for interfacing logic to loads that IC manufacturers offer arrays of seven or eight Darlington switches, complete with appropriate base resistors and protective diodes, in neat IC packages. Figure 2 shows two such units, the ULN2003 and ULN2803. These interface directly to a Stamp pin to drive loads of up to 500 mA per output.

The input pins of the ULN2x03s can connect directly to Stamp I/O pins. They're equivalent to connecting the pin to the left of RB in figure 1. The output pins of the ULN2x03s are equivalent to the collector (C) connection of the transistor switch in figure 1. They provide a switched ground connection for the load.

The ULN2x03s also feature something not shown in figure 1, a series of diodes connected to their outputs. When the devices are used to power inductive loads like relays and motors, these diodes should be connected to the positive

supply that powers the load. When one of the ULN2x03 switches cuts power to the inductive load, the load's magnetic field collapses, generating a nasty negative power spike. The diodes short out this spike, preventing it from damaging the transistor switch.

If you construct your own switches with discrete transistors, you'd do well to copy this protective feature. Just add a common rectifier diode like a 1N4002 with its banded end (cathode; the negative connection when the diode is conducting) to the + connection of the relay or motor. The diode won't interfere with the normal operation of the motor or relay, but it'll snub those spikes!

A real-world example

Francis Rogers of Sun City West, Arizona wrote me to describe an application he'd like to build with the Stamp. He has a PC with barcode software that can read membership cards for his S.C.W. Metals Club. The barcode software generates a code through the PC serial port when it's presented with a valid card. Mr. Rogers would like the Stamp to read this code and energize a relay to unlatch a door.

This fits perfectly with the theme of this month's column. Figure 3 is the schematic. I've made some assumptions about Mr. Rogers' barcode software: that it can be set to output at 2400 baud, and that all valid cards output some common code for the Stamp to recognize.

Thanks to the serial-input (Serin) command's built-in "qualifier" feature, the entire program takes just a few lines of Stamp code:

```
loop:
  low 7          ' Pin 7 low to latch door
                 ' (relay open)
  serin 0,N2400,("OK")  ' Watch serial input
                 ' until "OK" rec'd.
  high 7         ' Pin 7 high to unlatch door
                 ' (relay closed).
  pause 5000     ' Wait 5 seconds.
goto loop        ' Latch door and resume
                 ' watching serial input.
```

Of course, Mr. Rogers will have to substitute the actual password for "OK" in the program above.

That's it for this month. Next time, we'll look at a nifty IC that lets the Stamp transmit and receive DTMF tones (telephone touch tones). In addition to obvious telephone applications, DTMF can be used as a form of low-speed, high-reliability data transfer. Stay tuned!

Sources

For switching transistors and ULN2x03 ICs get a catalog from Jameco Electronic Components, 1355 Shoreway Road, Belmont, CA 94002-4100; phone 1-800-831-4242.

Questions, suggestions or comments about this column? Contact Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623; e-mail (Compuserve) at 72037,2612; or via Internet 72037.2612@compuserve.com.

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101.

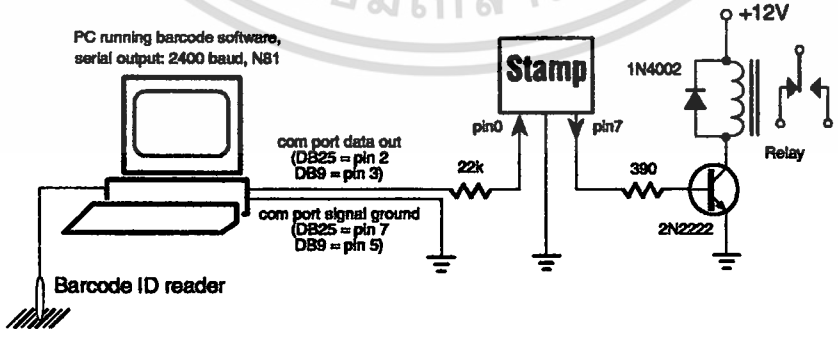


Figure 3. When the Stamp receives the proper code from the PC bar-code reader, it activates a relay to unlock the door.

Stamp Applications no. 7 (September '95):

DTMF "Touch" Tones are Music to the Ears of this Stamp Transmit/Receive Circuit

Working with the CM8880 DTMF Transceiver, by Scott Edwards

ENCODING and decoding DTMF tones—those musical "touch tones" you hear when you dial the phone—is the current fad in electronics projects. There are gadgets that capture and display or store dialed digits, others that dial stored numbers, and still others that use the hardy dual-tone, multifrequency digits for data transmission or remote control.

Stamp users are an independent lot, so we're going to buck the trend of one-trick DTMF projects. In this edition of Stamp Applications, we'll look at a universal DTMF send/receive solution that can serve as the basis for decoders, loggers, dialers, and even data transceivers.

A Quick Review of DTMF Principles

All the recent excitement over the decades-old DTMF signaling method makes the first part of my job easy. I can breeze through the explanation of the signals themselves with confidence that the curious reader who wants more background can find it his or her recent-magazine stack, for example, *DTMF IR Remote Control System, N&V* June '95.

A DTMF signal consists of a mixture of two sinewave tones, often called the *row* and *column* frequencies because of their correspondence to the layout of the phone keypad (figure 1). Engineers at then-mighty Ma Bell chose these tones carefully to ensure that none had a harmonic relationship with the others and that mixing the frequencies would not produce sum or product frequencies that could mimic another valid tone. They specified that the tones be free of distortion, and that the high-group frequencies (the column tones) be slightly louder than the low-group to compensate for the high-frequency rolloff of voice audio systems.

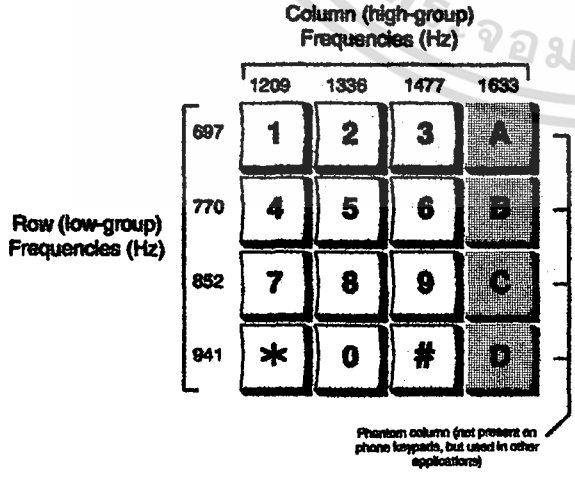


Figure 1. The DTMF tone that's generated when you press a button on the phone keypad consists of a mixture of the row and column frequencies. For example, pressing 9 mixes a column frequency of 1477 Hz with a row frequency of 852 Hz.

Because DTMF was devised in the days before you could hear a pin drop or appreciate Whitney Houston's singing over long distance, it's an exceptionally noise-resistant signaling method. If a properly designed DTMF decoder thinks there's a valid DTMF signal present in some lousy audio, there almost surely is.

DTMF tones can represent one of 16 different states or symbols, as shown in figure 1. That is equivalent to four bits of data, also known as a nibble. Most DTMF decoders can process at least 10 tones per second under the worst of conditions, so DTMF can easily convey 40 bits (5 bytes) of data per second. That's nowhere near the performance of a good communications modem, which can operate nearly 600 times as fast (28,800 bits per second), but it's a lot more robust under noisy line conditions.

Note that the numbers and symbols on the phone keypad don't always match the binary values of the DTMF nibbles. Most notably, the "0" on the keypad is represented in DTMF by a value of 10 (decimal) or 1010 binary. Table 1 summarizes the rest of the four-bit values.

Table 1. DTMF Values, Keypad Symbols

Binary Value	Decimal Value	Keypad Symbol
0000	0	D
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	0
1011	11	*
1100	12	#
1101	13	A
1110	14	B
1111	15	C

A DTMF Transceiver: the CM8880

All you need to generate DTMF tones are two precise, low-distortion sine-wave oscillators and a mixer. To decode DTMF takes just eight tone decoders and a little glue logic. The circuit board to accommodate all this stuff shouldn't be much more than 4 by 6 inches and $50.

Don't like those numbers? Then you will like the California Micro Devices CM8880 Integrated DTMF Transceiver. For $10 or less in single quantity, this chip does everything associated with sending and receiving DTMF tones. The schematic in figure 2 shows a Stamp hookup for both send and receive applications. Listing 1 demonstrates DTMF dialing. Listing 2 shows DTMF reception and display.

Rather than rehash the comments from the program listings, I'd like to show you how to set up and use the CM8880 in your own applications.

Communication with the 8880 takes place over a 4-bit bus, consisting of D0 through D3, with three additional bits selecting modes of operation. Those bits are chip select (CS), read/write (RW) and register select (RS0). Table 2 summarizes the effects of all combinations of CS, RW, and RS0.

To sum up the table, the 8880 is active only when CS is 0. The RW bit determines the data direction; 1 = read (data from 8880 to Stamp) and 0 = write (data from Stamp to 8880). The RS bit determines whether the transaction involves data (DTMF tones) or internal CM8880 functions (instructions or status); 1 = instructions/status and 0 = data.

Once the CM8880 is set up, the Stamp writes 000 to CS, RW, and RS0 to send DTMF; or 010 to read DTMF. Simple as that.

Setting up the CM8880

Before you can use the CM8880, you have to set it up. The device has two control registers, A and B. When you put 001 in bits CS, RW and RS0, the data on D0 through D3 is written to

Table 2. Effects of the CS, RW, and RS0 Bits

CS	RW	RS0	Description
0	0	0	Active: write data (i.e., send DTMF)
0	0	1	Active: write instructions to 8880
0	1	0	Active: read data (i.e., receive DTMF)
0	1	1	Active: read status from 8880
1	0	0	Inactive
1	0	1	Inactive
1	1	0	Inactive
1	1	1	Inactive

Table 3. Functions of Control Register A

Bit	Name	Function
0	Tone Out	0 = tone generator disabled
		1 = tone generator enabled
1	Mode Control	0 = Send and receive DTMF
		1 = Send DTMF, receive call-progress tones
		(DTMF bursts lengthened to 104 ms)
2	Interrupt Enable	0 = Make controller check for DTMF rec'd
		1 = Interrupt controller via pin 13 when DTMF rec'd
3	Register Select	0 = Next instruction write goes to CRA.
		1 = Next instruction write goes to CRB.

Table 4. Functions of Control Register B

Bit	Name	Function
0	Burst	0 = Output DTMF bursts of 52 or 104 ms
		1 = Output DTMF as long as enabled
1	Test	0 = Normal operating mode
		1 = Present test timing bit on pin 13
2	Single/ Dual	0 = Output dual (real DTMF) tones.
		1 = Output separate row or column tones
3	Column/ Row	0 = If above = 1, select row tone.
		1 = If above = 1, select column tone.

control register A (CRA). Table 3 summarizes the functions of the four bits of CRA.

Looking at listing 1, the dialer, we see that 1011 was written to CRA. That works out to (right to left) 1 (tone generator on), 1 (Send DTMF, long tones), 0 (don't generate interrupts), and 1 (next write to CRB). Listing 2 writes 1000 to CRA: 0 (tone-generator off), 0 (receive DTMF), 0 (don't generate interrupts) and 1 (next write to CRB).

Although we don't use it in our applications, the 8880 has a call-progress detection feature, which can be enabled via bit 1 of CRA. Call-progress is a collective term for the dial tone, busy signal, and ringing signal. These tones are all around 400 Hz. The CM8880's call-progress

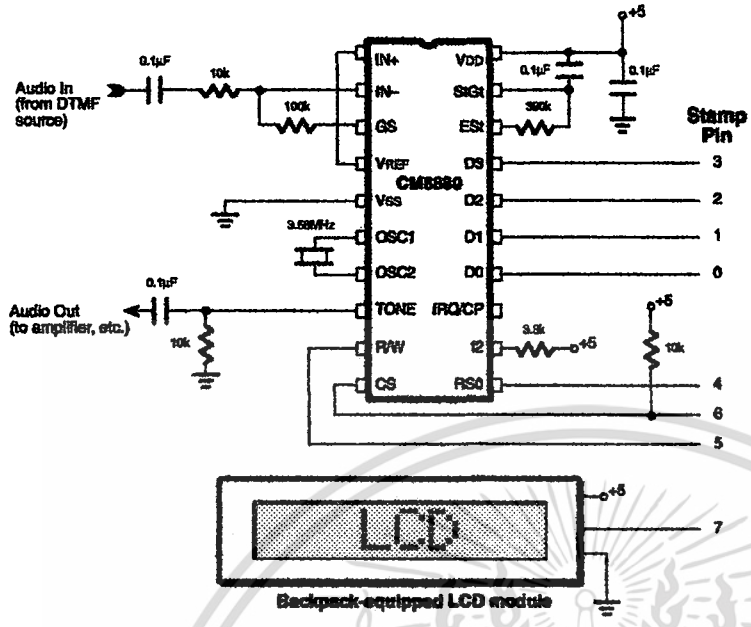


Figure 2. Schematic diagram for DTMF send and receive applications.

filter accepts audio from the normal input and passes frequencies from 300 to 500 Hz to the IRQ/CP pin. You could add hardware to further filter and detect these tones.

Since both listings also set up register B (CRB), take a look at table 4 to see how it works.

Listing 1 (dialer) writes 0s to all bits of CRB, meaning burst mode, normal operation, real DTMF, and (if DTMF weren't selected) row tones. Listing 2 (decoder) also clears CRB. Although CRB has little or nothing to do with DTMF reception, it's wise to always initialize it to a known state.

A Few Hardware Notes

To hear the DTMF tones generated by the CM8880, I connected a small speaker/amplifier (Radio Shack part no. 277-1008C) to the audio output. The amplifier's high gain and the 8880's healthy audio output quickly taught me to set the volume control low! I was able to dial the phone by holding the speaker close to the mouthpiece.

I used two approaches to generate DTMF test tones for the decoder application. The first was to connect an old touch-tone phone to a 9V battery and the audio-input coupling capacitor (0.1µF shown in figure 2) to the positive terminal. The polarity of the connection didn't matter, as phones are designed to tolerate wiring mixups. However, this setup sucked better than 50 mA from the battery.

I found a better DTMF generator at one of my favorite surplus dealers. Marlin P. Jones and Associates (MPJA) has a bunch of dialers that were apparently made for one of those early flash-in-the-pan long-distance services. It's preprogrammed to autodial an access number. If you press the manual-dial button (MAN'L) on the keypad, it works like a normal DTMF keypad. Output is through a headphone speaker built into the case.

MPJA also stocks a couple of other essentials for this circuit. The decoder application uses an LCD equipped with one of my LCD Serial Backpacks. This allows the Stamp to display data on it using a convenient one-wire serial connection. MPJA stocks both the Backpack and a range of LCDs from 1-line by 16 characters to 2x40. See Sources for more information.

DTMF and a glass of red...

It's almost embarrassing to admit that that's all there is to designing DTMF send/receive applications based on the CM8880 and the Stamp. I deliberately skirted issues having to do with direct connection to the phone line, since that's a subject worthy of two (or three!) articles of this size. Your best bet is to see how others have done it successfully, or to purchase a

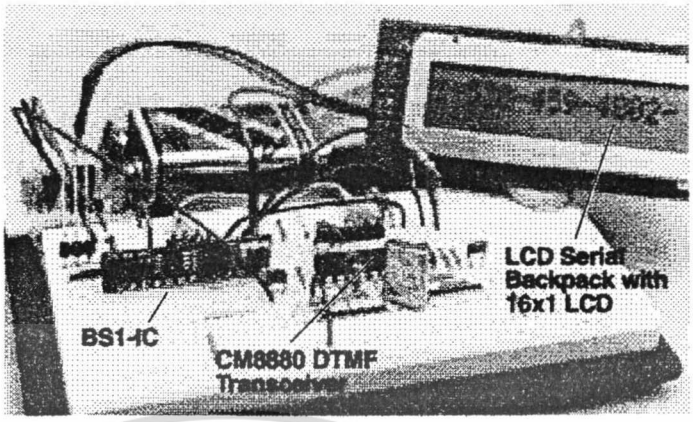


Figure 3. DTMF send/receive setup.

commercially made Data Access Arrangement (DAA) to serve as an interface. One manufacturer of DAAs is Cermetek, Sunnyvale, CA, 408-752-5000.

Don't limit yourself to phone applications. Remember what I said about DTMF serving as a reliable data-transfer method? You could connect CM8880-equipped Stamps to inexpensive two-way radios to create a wireless network for data acquisition over a 1/4-mile radius.

I experimented briefly with this idea, using the DTMF output from the amplified speaker to trigger the voice-activated transmit feature of a Radio Shack headset walkie-talkie (part no 21-406). The voice activation was fast enough that even the first tone got through. This means that you wouldn't need an additional pin to switch between transmit and receive.

This is an application that a lot of Stamp users have been clamoring for. I remember in particular that grape growers need to gather and evaluate "microclimate" data regarding temperature and humidity among their pampered vines, and would kill for a wireless way to obtain it. Here it is: Cheers. (And I prefer red, if you please, a Cabernet Sauvignon or Merlot.)

Return of the Counterfeit

Many of you have called my order line to purchase BASIC Stamps. It's a natural assumption: I'm cheerleading for a great product—I've got to *carry* that product, right?

Unfortunately, I don't. However, the manufacturer (Parallax) makes PBASIC chips available to those who want to build their own controllers based on Stamp technology. I showed how to do just that in my "Counterfeit Stamp" article last spring (*N&V*, May '94).

In response to many requests, I'm making the Counterfeit available again as a complete kit for $29. The design incorporates features I have found handy in developing applications you've seen here: heavy-duty (100+ mA) voltage regulator, socket for pull-up/pull-down resistors, ground and +5V connections paired with every I/O pin, and Turbo options for running the Counterfeit at 8 or 16 MHz—two to four times standard speed. The board is compact, but easy to assemble; see the photo (figure 4).

I have also put together a Counterfeit Development System for $69. This includes everything required to program Counterfeits (or BS1 Stamps, for that matter), including your first Counterfeit kit. See the Sources listing for further information.

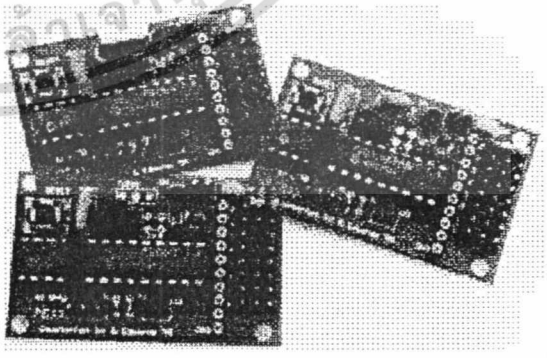


Figure 4. A trio of Stamp-compatible Counterfeit Controllers.

Sources

For more information on the BASIC Stamp, contact Parallax Inc., 3805 Atherton Road no. 102, Rocklin, CA 95765; phone 916-624-8333; fax 916-624-8003; BBS 916-624-7101; e-mail info@parallaxinc.com.

The CM8880 is available from electronics distributors who carry California Micro Devices' product line (CMD phone, 602-961-6000), or from Scott Edwards Electronics (see listing below).

The DTMF box (part no. 5824-EN, $4.95), LCD Serial Backpack (7074-SE, $29), and a variety of LCD displays are available from Marlin P. Jones and Associates, P.O. Box 12685, Lake Park, FL 33403-0685; phone 407-848-8236; fax 407-844-8764.

Scott Edwards Electronics:

Questions, suggestions, or requests for future Stamp Applications to: Scott Edwards Electronics, 964 Cactus Wren Lane, Sierra Vista, AZ 85635; phone 520-459-4802; fax 520-459-0623; e-mail (Compuserve) at 72037,2612; Internet 72037.2612@compuserve.com. Scott also offers the following Stamp-related kit goodies:

The CM8880 is $10; an App Kit containing one CM8880, the Stamp programs shown here (plus PIC assembly language versions) on disk, and complete documentation, is $22.

The Counterfeit controller, a kit alternative to the BASIC Stamp, is $29. Double- and quad-speed options are $2 and $4, respectively. The Counterfeit Development System, required to program Counterfeits (also for programming original BASIC Stamps, like the BS1-IC) is $69 and includes 150-page manual, downloading cable kit, Parallax software, and one Counterfeit controller kit.

Prices are postpaid (express shipping and CODs extra). Visa, Mastercard, American Express accepted for phone/fax orders. POs accepted on approved credit. Personal checks, money orders welcome for orders by mail.

```
' Listing 1. Stamp-Based Autodialer

' Program: DIAL.SRC (Sends a string of DTMF tones via the 8880)
' This program demonstrates how to use the CM8880 as a DTMF tone
' generator. All that's required is to initialize the 8880 properly,
' then write the number of the desired DTMF tone to the 8880's
' 4-bit bus.

' The symbols below are the pin numbers to which the 8880's
' control inputs are connected, and one variable used to read
' digits out of a lookup table.

SYMBOL      RS_p = 4            ' Register-select pin (0=data).
SYMBOL      RW_p = 5            ' Read/Write pin (0=write).
SYMBOL      CS_p = 6            ' Chip-select pin (0=active).
SYMBOL      digit = b2          ' Index of digits to dial.

' This code initializes the 8880 for dialing by writing to its
' internal control registers CRA and CRB. The write occurs when
' CS (pin 6) is taken low, then returned high. See the accompanying
' article for an explanation of the 8880's registers.

let pins = 255          ' All pins high to deselect 8880.
let dirs = 255          ' Set up to write to 8880 (all outputs).
let pins = %00011011    ' Set up  register A, next write to register B.
high CS_p
let pins = %00010000    ' Clear register B; ready to send DTMF.
high CS_p
```

```
' This for/next loop dials the seven digits of my fax number. For
' simplicity, it writes the digit to be dialed directly to the output
' pins. Since valid digits are between 0 and 15, this also takes RS,
' RW, and CS low--perfect for writing data to the 8880. To complete
' the write, the CS line is returned high. The initialization above
' sets the 8880 for tone bursts of 200 ms duration, so we pause
' 250 ms between digits. Note: in the DTMF code as used by the phone
' system, zero is represented by ten (1010 binary) not 0. That's why
' the phone number 459-0623 is coded 4,5,9,10,6,2,3.

for digit = 0 to 6
  lookup digit,(4,5,9,10,6,2,3),pins ' Write current digit to pins.
  high CS_p                  ' Done with write.
  pause 250                  ' Wait to dial next digit.
next digit
end
```

' Listing 2. DTMF Decoder and Display

```
' Program: DTMF_RCV.BAS (Receives and display DTMF tones using the 8880)
' This program demonstrates how to use the 8880 as a DTMF decoder. As
' each new DTMF digit is received, it is displayed on an LCD Serial
' Backpack screen. If no tones are received within a period of time
' set by sp_time, the program prints a space (or other selected character)
' to the LCD to record the delay. When the display reaches the righthand
' edge of the screen, it clears the LCD and starts over at the left edge.

SYMBOL      RS_p = 4          ' Register-select pin (0=data).
SYMBOL      RW_p = 5          ' Read/Write pin (0=write).
SYMBOL      CS_p = 6          ' Chip-select pin (0=active).
SYMBOL      dtmf = b2         ' Received DTMF digit.
SYMBOL      dt_Flag = bit0    ' DTMF-received flag.
SYMBOL      home_Flag = bit1  ' Flag: 0 = cursor at left edge of LCD.
SYMBOL      polls = w2        ' Number of unsuccessful polls of DTMF.
SYMBOL      LCDw = 16         ' Width of LCD screen.
SYMBOL      LCDcol = b3       ' Current column of LCD screen for wrap.
SYMBOL      LCDcls = 1        ' LCD clear-screen command.
SYMBOL      I = 254           ' LCD instruction toggle.
SYMBOL      sp_time = 1000    ' Print space this # of polls w/o DTMF.

' This code initializes the 8880 for receiving by writing to its
' internal control registers CRA and CRB. The write occurs when
' CS (pin 6) is taken low, then returned high.

let pins = %01111111     ' Pin 7 (LCD) low, pins 0 through 6 high.
let dirs = %11111111     ' Set up to write to 8880 (all outputs).
let pins = %00011000     ' Set up register A, next write to register B.
high CS_p
let pins = %00010000     ' Clear register B.
high CS_p
let dirs = %11110000     ' Now make set the 4-bit bus to input.
high RW_p                ' And set RW to "read."
serout 7,n2400,(I,LCDcls,I)   ' Clear the LCD screen.

' In the loop below, the program checks the 8880's status register
' to determine whether a DTMF tone has been received (indicated by
' a '1' in bit 2). If no tone, the program loops back and checks
```

```
' again. If a tone is present, the program switches from status to
' data (RS low) and gets the value (0-15) of the tone. This
' automatically resets the 8880's status flag.
again:
  high RS_p              ' Read status register.
  low CS_p               ' Activate the 8880.
  let dt_flag = pin2     ' Store status bit 2 into flag.
  high CS_p              ' End the read.
if dt_Flag = 1 then skip1      ' If tone detected, continue.
let polls = polls+1            ' Another poll without DTMF tone.
if polls < sp_time then again ' If not time to print a space, poll again.
if LCDcol = LCDw then skip2    ' Don't erase the screen to print spaces.
let dtmf = 16                  ' Tell display routine to print a space.
gosub Display                  ' Print space to LCD.
skip2:
let polls = 0            ' Clear the counter.
goto again               ' Poll some more.
skip1:                   ' Tone detected:
let polls = 0            ' Clear the poll counter.
low RS_p                 ' Get the DTMF data.
low CS_p                 ' Activate 8880.
let dtmf = pins & %00001111    ' Strip off upper 4 bits using AND.
high CS_p                ' Deactivate 8880.
gosub display            ' Display the data.
goto again               ' Do it all again.

Display:
if LCDcol < LCDw then skip3    ' If not at end of LCD, don't clear screen.
serout 7,n2400,(I,LCDcls,I)    ' Clear the LCD screen.
let LCDcol = 0                 ' And reset the column counter.
skip3:                         ' Look up the symbol for the digit.
if LCDcol=0 AND dtmf=16 then ret     ' No spaces at first column.
lookup dtmf,("D1234567890*#ABC-"),dtmf
serout 7,n2400,(dtmf)          ' Write it to the Backpack display.
let LCDcol = LCDcol + 1        ' Increment the column counter.
ret:
return
```