สถานีถ่ายทอดสัญญาณ
ควบคุมโดยระบบไมโครโปรเซสเซอร์

นายพิชัย ศรีสาริสถ์
นายสุจิต เอื้อปัญญาพร

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

MICROPROCESSOR BASED REPEATER

Mr.Phichai Srisarist

Mr.Sujit Uarpanyaporn

A Special Project Submitted in Partial Fulfillment of the Requirement

For the Degree of Bachelor of Science

Department of Applied Physics

Faculty of Science

King Monkut's Institute of Technology Ladkrabang

1994

หัวข้อโครงงานพิเศษ	สถานีถ่ายทอดสัญญาณควบคุมโดยระบบไมโครโปรเซสเซอร์
โดย	นายพิชัย ศรีสาริสถ์
	นายสุจิต เอื้อปัญญาพร
ภาควิชา	ฟิสิกส์ประยุกต์
อาจารย์ที่ปรึกษา	อาจารย์วิชิต ศิริโชติ

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ได้อนุมัติให้นับรายงานโครงงานพิเศษฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

..... หัวหน้าภาควิชาฟิสิกส์ประยุกต์

(ผศ.ดร.ปรีชา เทียนสมประสงค์)

คณะกรรมการโครงงานพิเศษ

..... กรรมการ

(ผศ.ดร.อารีย์ วิเชียรฉาย)

..... กรรมการ

(ผศ.ดร.เสน่ห์ เอกะวิภาต)

..... กรรมการ

(อ.ดร.ฐิตินัย แก้วแดง)

..... กรรมการ

(อ.ธีรวัฒน์ ประกอบผล)

ลิขสิทธิ์ของภาควิชาฟิสิกส์ประยุกต์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อโครงงานพิเศษ	สถานีถ่ายทอดสัญญาณควบคุมโดยระบบไมโครโปรเซสเซอร์
นักศึกษา	นายพิชัย ศรีสาริสถ์
	นายสุจิต เอื้อปัญญาพร
ภาควิชา	ฟิสิกส์ประยุกต์
อาจารย์ที่ปรึกษา	อาจารย์วิชิต ศิริโชติ
ปีการศึกษา	2537

บทคัดย่อ

สถานีถ่ายถอดสัญญาณควบคุมโดยระบบไมโครโปรเซสเซอร์ เป็นส่วนที่ออกแบบและสร้างขึ้นมาเพื่อช่วยเพิ่มระยะทางการติดต่อสื่อสารแบบดิจิตอลระหว่างสถานีลูกกับสถานีแม่ สถานีถ่ายทอดสัญญาณจะทำหน้าที่หน่วงเวลาในการส่งคำสั่งจากสถานีแม่ และสถานีลูกก็จะทำการอ่านข้อมูลแบบดิจิตอล โดยจะประกอบไปด้วยไมโครโปรเซสเซอร์เบอร์ 8031 และใช้ร่วมกับ NVSRAM ขนาด 8 กิโลไบต์ เรดิโอโมเด็มและเครื่องรับส่งสัญญาณวิทยุ รูปแบบของแพคเก็ตจะประกอบไปด้วยคำสั่งและข้อมูลดิจิตอล ซึ่งจะทำการโมดูเลทสัญญาณโดยเรดิโอโมเด็มแล้วส่งไปยังเครื่องรับส่งสัญญาณวิทยุ โดยที่จะทำการเก็บข้อมูลไว้ก่อนแล้วจึงทำการส่งข้อมูลออกไปเป็นแบบแพคเก็ต ซึ่งมีแอดเดรสของสถานีถ่ายทอดสัญญาณรวมอยู่ด้วย โปรโตคอลที่ได้ออกแบบจะใช้คำสั่งเป็นตัวอักษรตัวเดียว

การติดต่อสื่อสารระหว่างสถานีแม่กับสถานีลูก ซึ่งใช้ร่วมกับสถานีถ่ายทอดสัญญาณ ระยะทางการติดต่อสื่อสารจะขึ้นอยู่กับระบบของสายอากาศที่เลือกใช้ โดยที่โครงงานนี้เลือกใช้ระบบสายอากาศแบบคอลลิเนียร์ 2 สแต็คที่สถานีแม่กับสถานีถ่ายทอดสัญญาณ ส่วนสถานีลูกจะใช้ระบบสายอากาศแบบเทเลสโคปิค ซึ่งทำให้สามารถติดต่อสื่อสารได้ที่ระยะทาง 15 กิโลเมตร

Special Project Title : Microprocessor Based Repeater

Name : Mr.Phichai Srisarist

: Mr.Sujit Uarpanyaporn

Special Project Advisor : Mr.Wichit Sirichote

Department : Applied Physics

Acadamic Year : 2537

Abstract

A Microprocessor Based Repeater, MBR has been developed for increasing the operating range of digital communication between an analog sender and a central station. The main function of repeater was used to relay the telecommand sent by the central station and the digital data read by the analog sender. The repeater consists of a microprocessor i8031 with 8KB NVSRAM, a radio MODEM and a radio transceiver. The incoming packets, both the telecommand and digital data which serially received via a transceiver through the radio modem, were saved and retransmitted as the outgoing packets with a repeater addressing code. The designed protocol for sending the packet was a simple character oriented protocol.

The operating range between the analog sender and central station has been preliminary tested with the repeater, even technically the range depends mostly upon the antenna system. With a homemade 2 stack collinear antenna at the central and repeater station and a telescopic antenna at the analog sender station, the range upto 15 Km can be obtained.

กิตติกรรมประกาศ

โครงงานพิเศษฉบับนี้สำเร็จขึ้นมาได้ด้วยความช่วยเหลือของบุคคลหลายๆ ฝ่ายดังต่อไปนี้

อ.วิชิต ศิริโชติ

ที่ได้กรุณาให้คำปรึกษา และคำแนะนำ ตลอดจนให้ความรู้เกี่ยวกับโครงงานพิเศษ

ขอขอบคุณ

ชุมนุมวิทยุสมัครเล่น คณะวิทยาศาสตร์ พระจอมเกล้า ลาดรกระบัง
เพื่อนๆ นักวิทยุสมัครเล่นทุกท่าน

ที่ได้ให้ความเอื้อเฟื้อทางด้านเครื่องมือสื่อสารที่จำเป็นสำหรับใช้ในการทดลอง

รวมทั้งเพื่อนๆ พี่ๆ น้องๆ ทุกคนที่ได้ให้ความช่วยเหลือมาโดยตลอด

และบุคคลที่สำคัญที่สุดของผู้เขียน คือ บิดาและมารดา ที่ได้ให้กำเนิดและคอยเลี้ยงดูอบรมสั่งสอน จนประสบความสำเร็จในการศึกษา ผู้เขียนขอกราบขอบพระคุณอย่างสูง

พิชัย ศรีสาริสถ์
สุจิต เอื้อปัญญาพร
22 มีนาคม 2538

สารบัญ

สารบัญตาราง

สารบัญรูป

บทที่ 1

บทนำ

จากความเจริญก้าวหน้าทางด้านเทคโนโลยีในปัจจุบัน ทำให้คอมพิวเตอร์เข้ามามีบทบาทมากขึ้น ทำให้ความต้องการในการติดต่อระหว่างคอมพิวเตอร์ด้วยกันเป็นที่ต้องการมากขึ้น โดยการติดต่อที่ใช้มีด้วยกัน 2 แบบคือ การติดต่อสื่อสารแบบอนุกรมกับแบบขนาน ซึ่งการติดต่อสื่อสารแบบอนุกรมนั้นใช้จำนวนสายนำสัญญาณน้อย ทำให้ประหยัด และสะดวกสบายกว่า แต่ให้อัตราเร็วในการส่งน้อยกว่าแบบขนานมาก ในการติดต่อสื่อสารแบบอนุกรม มีอุปกรณ์อย่างหนึ่ง ที่ช่วยเพิ่มประสิทธิภาพของการรับส่งข้อมูลแบบอนุกรม ก็คือ โมเด็ม ซึ่งใช้ในการติดต่อสื่อสารระหว่างคอมพิว เตอร์ด้วยกันผ่านทางสายโทรศัพท์ จะต้องมีการใช้สายนำสัญญาณ ซึ่งได้มีการพัฒนาโมเด็มให้สามารถติดต่อกันโดยผ่านทางคลื่นวิทยุแทน เรียกอุปกรณ์นี้ว่า packet modem

ในการพัฒนาอุปกรณ์นี้ใช้หลักการเดียวกันกับโมเด็ม คือ ทำการเปลี่ยนสัญญาณดิจิตอลให้เป็นสัญญาณเสียงแล้วทำการโมดูเลทเข้ากับคลื่นวิทยุ เพื่อทำการส่งออกอากาศ ซึ่งสามารถนำไปประยุกต์ เพื่อใช้ในการติดต่อกับเครื่องมือวัดที่อยู่ในบริเวณต่างๆ ได้ โดยไม่ต้องใช้สายนำสัญญาณ โดยการติดต่อสื่อสารแบบนี้จะเป็นแบบ Half-Duplex เมื่อใช้ความถี่ในการติดต่อเพียงความถี่เดียว ข้อได้เปรียบของการติดต่อสื่อสารโดยใช้คลื่นวิทยุนี้ สามารถติดต่อกับอุปกรณ์ได้เป็นจำนวนมากและได้ค่อนข้างไกล

มีอยู่บ่อยครั้งที่ระบบสื่อสารไม่สามารถติดต่อกันได้ด้วยเครื่องที่มีกำลังค่อนข้างต่ำ หรือในท้องที่อับสัญญาณ ทำให้ติดต่อสื่อสารกันไม่ได้ จำเป็นจะต้องใช้สถานีที่มีกำลังส่งค่อนข้างสูงและมีเสาอากาศที่สูงช่วยถ่ายทอดข้อความจากเราไปยังผู้อื่น จึงต้องมีการติดตั้งระบบรีพีทเตอร์ (repeater) นับว่ามีความจำเป็นอยู่มาก เพื่อช่วยเพิ่มระยะทางและรัศมีการติดต่อออกไปให้สามารถรับหรือส่งข่าวสารกันได้ไกลยิ่งขึ้น เพราะพื้นที่การใช้งานถ้ามองตามภูมิศาสตร์จะเห็นพื้นที่ต่างๆ มีความสูงต่ำไม่เท่ากัน บางแห่งเป็นภูเขาค่อนข้างสูงจะทำให้ติดต่อสัญญาณกันไม่ได้

<u>วัตถุประสงค์ของโครงงาน</u>

1. เพื่อศึกษาการทำงานของระบบ Digital Repeater
2. เพื่อศึกษาการเชื่อมต่อไมโครคอมพิวเตอร์เข้ากับวิทยุสื่อสาร
3. เพื่อศึกษาการติดต่อและควบคุมบอร์ดไมโครคอนโทรลเลอร์ โดยผ่านทางวิทยุสื่อสาร
4. เพื่อศึกษาการแพร่กระจายคลื่นวิทยุของระบบสายอากาศ
5. เพื่อเป็นโครงการพัฒนาเทคโนโลยีด้าน Digital Telemetry สำหรับการประยุกต์ในด้านการส่งข้อมูลการวัดปริมาณทางฟิสิกส์

<u>ขอบเขตของโครงงาน</u>

1. สามารถสร้าง Packet Radio Modem ที่มีอัตราการรับส่งข้อมูลขนาด 1200 บิตต่อวินาที
2. สามารถรับส่งข้อมูลระหว่างไมโครคอมพิวเตอร์ 2 ชุดได้
3. สามารถรับส่งข้อมูลระหว่างส่วนควบคุมย่อย (Analog Zender) กับไมโครคอมพิวเตอร์ได้
4. สามารถสร้างระบบสายอากาศ สำหรับสถานีควบคุม (Central Work Station) และสถานีถ่ายทอดสัญญาณ (Repeater) ได้

<u>ประโยชน์ที่ได้รับจากโครงงาน</u>

1. นำ packet modem ที่สร้างขึ้นมาใช้ในการติดต่อสื่อสารระหว่างไมโครคอมพิวเตอร์ โดยผ่านทางวิทยุสมัครเล่น

2. นำบอร์ดควบคุมที่สร้างขึ้น มาใช้ในการรับส่งข้อมูลกับไมโครคอมพิวเตอร์ โดยผ่านทางวิทยุสมัครเล่น

<u>ขั้นตอนการดำเนินงาน</u>

1. ทำการศึกษาทฤษฎีและการทำงานของระบบ Digital Repeater
2. ทำการศึกษาทฤษฎีและการทำงานของระบบการสื่อสารข้อมูล ผ่านทางระบบ RS 232
3. ทำการศึกษาทฤษฎีและการทำงานของระบบ Microcontroller 8051
4. ทำการศึกษาทฤษฎีและสร้างระบบสายอากาศ
5. ทำการศึกษาทฤษฎีและการทำงานของระบบ Microprocessor Based Repeater
6. ทดลองและออกแบบวงจรของ Packet Radio Modem 1200 บิตต่อวินาที
7. ทดลองและออกแบบวงจรของ Microcontroller โดยเลือกใช้ชิป 8031
8. เขียนโปรแกรมควบคุมการทำงานของ Microprocessor Based Repeater
9. ทดสอบการทำงานของระบบ และประยุกต์ใช้งาน

บทที่ 2

ทฤษฎี

2.1 ระบบการสื่อสารข้อมูล

เมื่อข้อมูลผ่านการแปลงสัญญาณเป็นข้อมูลทางดิจิตอลแล้วข้อมูลส่วนนี้จะถูกนำมาวิเคราะห์บนไมโครคอมพิวเตอร์ ซึ่งจะต้องมีการส่งผ่านข้อมูลจากบอร์ดมาเก็บไว้ในไมโครคอมพิวเตอร์ การส่งผ่านข้อมูลหรือการถ่ายโอนข้อมูลสามารถกระทำได้ทั้งแบบขนานและอนุกรม โดยที่การถ่ายโอนข้อมูลแบบขนานอาจผ่านทางพอร์ตพรินเตอร์ ซึ่งต้องทำความเข้าใจสัญญาณต่างๆ ที่ใช้ติดต่อกับพรินเตอร์ แต่หากทำการถ่ายโอนข้อมูลแบบอนุกรมสามารถทำการถ่ายโอนได้หลายแบบตามแต่ระยะทาง และความเร็วที่ต้องการ หรืออาจถ่ายโอนข้อมูลด้วยการทำเป็นการ์ดเสียบสล็อตคอมพิวเตอร์ก็ได้ แต่จะมีข้อเสียที่มีสัญญาณรบกวนมากกว่า

ในหัวข้อนี้จะกล่าวรูปแบบการสื่อสารต่างๆ โดยเน้นที่การสื่อสารแบบอนุกรมเป็นหลัก เนื่องจากในโครงงานชิ้นนี้ใช้การสื่อสารแบบอนุกรม

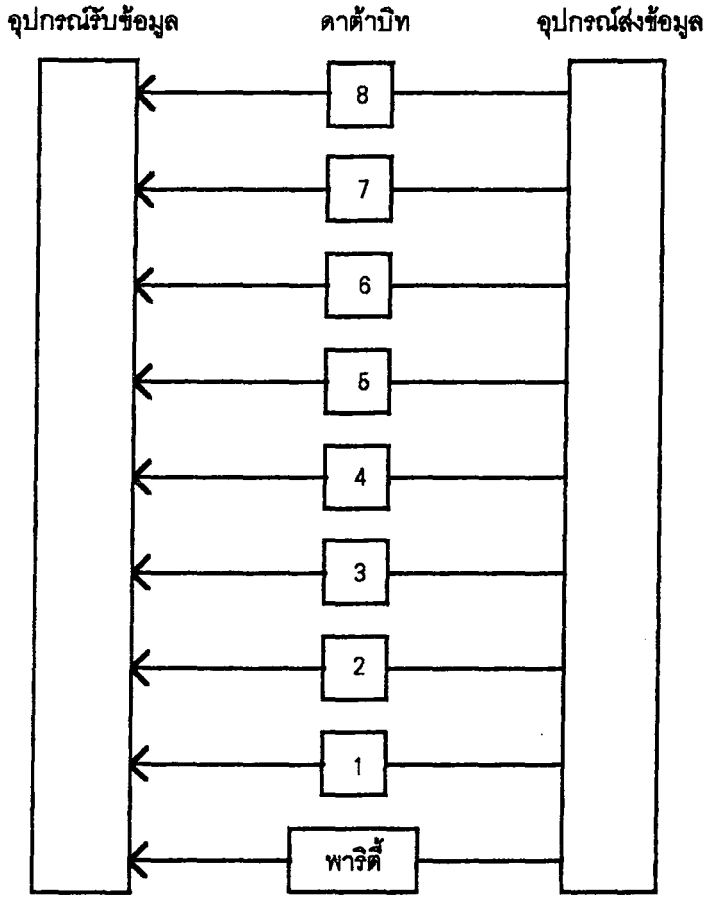
วิธีการถ่ายโอนข้อมูล

ก่อนที่กล่าวถึงการสื่อสารระหว่างเครื่อง ลองมาศึกษาว่าสัญญาณที่ส่งและรับนั้น มีวิธีการอย่างไร ดังนี้

การถ่ายโอนข้อมูลแบบขนาน

ลักษณะการถ่ายโอนข้อมูลแบบขนาน ทำได้โดยการส่งข้อมูลออกมาทีละ 1 ไบท์ คือ 8 บิท จากอุปกรณ์ส่งไปยังอุปกรณ์รับ ตัวกลางระหว่างเครื่อง 2 เครื่อง จะต้องมีช่องทางให้ข้อมูลเดินทางได้อย่างน้อย 8 ช่องทาง โดยมากจะเป็นสายขนานให้กระแสวิ่งผ่านไปมากกว่าจะเป็นอุปกรณ์อย่างอื่น เนื่องจากมีสัญญาณสูญหายไปกับความต้านทานของสาย ระยะทางระหว่างเครื่องจึงไม่ควรเกิน

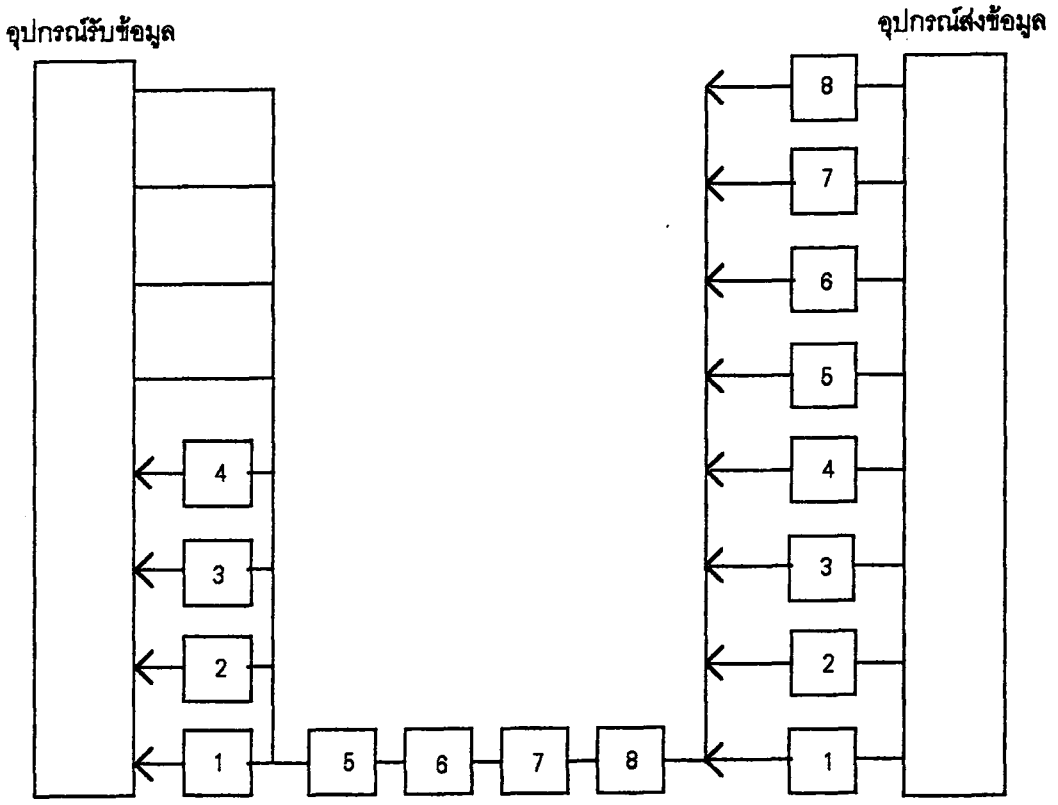
100 ฟุต ปัญหาที่เกิดขึ้นหากระยะทางมากกว่านี้คือ ระดับของกราวด์ทางไฟฟ้าที่จุดรับผิดไปจากจุดส่ง ทำให้เกิดความผิดพลาดในการรับสัญญาณลอจิกทางฝ่ายรับนอกจากสายที่เป็นทางเดินของข้อมูลแล้ว อาจจะมีทางเดินของสัญญาณควบคุมอื่นๆ อีก เป็นต้นว่าบิทที่บอกพาริตี้ของสัญญาณ เพื่อเป็นการตรวจสอบความผิดพลาดของการรับสัญญาณที่ปลายทางหรือสายที่ควบคุมการโต้ตอบ (Hand-shake) ดังที่กล่าวมาแล้ว



รูปที่ 2.1 การส่งข้อมูลแบบขนาน

การถ่ายโอนข้อมูลแบบอนุกรม

ในการถ่ายโอนข้อมูลแบบอนุกรม ข้อมูลถูกส่งออกมาทีละบิท ระหว่างจุดส่งและจุดรับ จะเห็นว่าการส่งข้อมูลแบบนี้จะช้ากว่าแบบขนานที่กล่าวมาแล้วแน่นอน แล้วทำไมต้องส่งแบบนี้ละ คำตอบก็ คือ ตัวกลางการต่อสายเพียงคู่เดียว ค่าใช้จ่ายในสื่อกลางจะต้องถูกกว่าแบบขนานอย่างแน่นอน สำหรับการส่งระยะทางไกลๆ โดยเฉพาะเมื่อเรามีระบบการสื่อสารทางโทรศัพท์ไว้ใช้งานอยู่แล้วย่อมจะเป็นการประหยัดกว่าที่จะทำการติดต่อสื่อสารทีละ 8 ช่อง เพื่อการถ่ายโอนข้อมูลแบบขนานอย่างแน่นอน



รูปที่ 2.2 การส่งข้อมูลแบบอนุกรม

รูปที่ 2.2 แสดงให้เห็นการส่งข้อมูลแบบอนุกรม ข้อมูลจากจุดส่งจะถูกเปลี่ยนให้เป็นอนุกรมเสียก่อนแล้วค่อยทยอยส่งทีละบิทไปยังจุดรับ ณ ที่จุดรับจะต้องมีกลไกในการเปลี่ยนข้อมูลที่ส่งมาทีละบิท ให้เป็นสัญญาณแบบขนานซึ่งลงตัวพอดี นั่นคือ บิท 1 ลงที่บัสข้อมูลเส้นที่ 1 พอดี การที่จะทำให้การแปลงสัญญาณจากอนุกรมทีละบิทให้ลงพอดีนั้นจำเป็นต้องมีกลไกที่เหมาะสม เพื่อป้องกันการผิดพลาดในการรับ กลไกที่ว่านี้แบ่งเป็น 2 แบบ คือ

1. การสื่อสารแบบซิงโครนัส

2. การสื่อสารแบบอะซิงโครนัส

รายละเอียดของทั้งสองแบบจะกล่าวในหัวข้อถัดไป

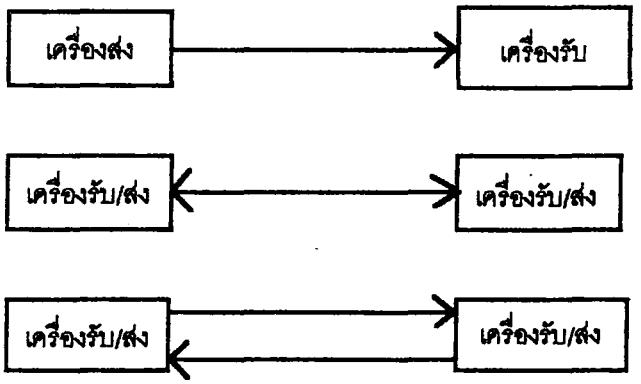
<u>รูปแบบการติดต่อสื่อสารแบบอนุกรม</u>

การติดต่อแบบอนุกรมอาจจะแบ่งตามรูปลักษณะได้ 3 รูปแบบ ตามรูปที่ 2.3

1) แบบซิมเพล็กซ์ **(Simplex)** ข้อมูลส่งได้ในทางเดียวเท่านั้น บางครั้งเรียกว่าการส่งทิศทางเดียว (Unidirectional data bus)

2) แบบฮาล์ฟดูเพล็กซ์ **(Half duplex)** ข้อมูลสามารถส่งได้ทั้งสองสถานี แต่จะต้องผลัดกันส่งและผลัดกันรับ จะส่งและรับพร้อมกันไม่ได้

3) แบบฟลูดูเพล็กซ์ **(Full duplex)** ทั้งสองสถานีสามารถรับและส่งได้ในเวลาเดียวกัน



รูปที่ 2.3 รูปแบบของการติดต่อสื่อสารข้อมูลแบบอนุกรม

การส่งแบบฟลูดูเพล็กซ์และฮาล์ฟดูเพล็กซ์ ไม่ขึ้นอยู่กับจำนวนของสายในการติดต่อ บางครั้งคำว่า ทูไวร์ (two wire) หรือสองเส้น และโฟร์ไวร์ (four wire) หรือ 4 เส้น ใช้ในการบรรยายถึงลักษณะการสื่อสารข้อมูลซึ่งอาจจะทำให้เข้าใจและฮาล์ฟดูเพล็กซ์ สายโทรศัพท์ทั่วไปเป็นแบบ 2 เส้น ส่วนในสายที่เ ป็นแบบเช่า (lease line) นั้นส่วนมากจะเป็น 4 เส้น

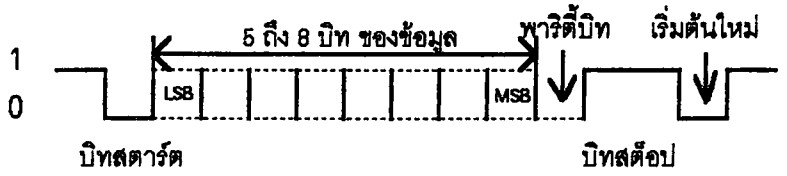
ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม

ความเร็วของการถ่ายโอนข้อมูลแบบอนุกรม หน่วยวัดเป็นบิทต่อวินาที (bps) หน่วยที่บรรยายถึงการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่าบอดเรต (baud rate) หรืออัตราบอดหลายคนยังเข้าใจสับสนระหว่างอัตราบอดและอัตราบิท (bit rate) การเปลี่ยนแปลงของสัญญาณ 1 ครั้ง อาจจะแสดงถึงการส่งข้อมูลแบบอนุกรมมากกว่า 1 บิท (รายละเอียดเกี่ยวกับเรื่องนี้จะได้กล่าวถึงอีกครั้งในเรื่องของโมเด็ม) ถ้าเขียนในรูปของสมการทางคณิตศาสตร์เราก็จะได้

อัตราบิท(bit rate) = อัตราบอด(baud rate) * บิทใน 1 บอด

2.1.1 การสื่อสารแบบอะซิงโครนัส

การส่งแบบอะซิงโครนัสนี้ พัฒนามาจากการส่งโทรพิมพ์ในสมัยก่อน ลักษณะของสัญญาณแสดงไว้ในรูปที่ 2.4 เพื่อเพิ่มกลไกในการรับส่งอย่างถูกต้อง สัญญาณอะซิงโครนัสจะประกอบด้วยบิทเริ่มต้นหรือบิทสตาร์ท (start) และบิทสิ้นสุดหรือบิทสต๊อป (stop bit)



รูปที่ 2.4 ฟอร์แมตการสื่อสารแบบอะซิงโครนัส

ขณะที่สถานะของการส่งเป็นแบบว่าง (Idle) คือ ยังไม่มีสัญญาณส่งออกมา จะมีสัญญาณหรือมีแรงดัน (หรือกระแส) ตลอดเวลา เพื่อความแน่ใจว่าฝ่ายรับยังติดต่ออยู่กับฝ่ายส่ง เมื่อเริ่มจะส่งข้อมูลสัญญาณของอะซิงโครนัสจะเป็น 0 หนึ่งช่วงสัญญาณนาฬิกา บิทนี้เรียกว่า สตาร์ทบิท ตามหลังของ

สตาร์ทบิทก็จะเป็นข้อมูลสำหรับ 1 ตัวอักษร ซึ่งอาจจะมีขนาดตั้งแต่ 5 บิท จนถึง 8 บิท โดยบิทที่มีค่าน้อยที่สุด (LSB) จะถูกส่งออกมาก่อนไล่ไปจนถึงบิทที่มีค่ามากที่สุด (MSB) การเข้ารหัสอักขระนี้ส่วนมากจะนิยมใช้รหัส ASCII แรกเริ่มที่เดียวในงานของโทรพิมพ์เขาใช้รหัส Baudot ซึ่งใช้ 5 บิทในการแทนอักขระ 1 ตัวตามหลังข้อมูลก็จะเป็นพาริตี้บิท ซึ่งอาจจะใช้หรือไม่ใช้ก็ได้พาริตี้บิททำหน้าที่เป็นตัวตรวจสอบความถูกต้องของสัญญาณที่ได้รับพาริตี้บิทอาจจะเป็นแบบ คู่ (Even) หรือแบบคี่(Odd)หมายความว่าถ้าหากเป็นพาริตี้คู่จำนวนบิทที่เป็น 1 ในช่วงบิทข้อมูลกับบิทพาริตี้รวมแล้วจะต้องเป็นจำนวนคู่ ผู้ส่งจะต้องทำหน้าที่ตรวจสอบข้อมูลแล้วใส่พาริตี้บิทเอง ฝ่ายรับเมื่อรับแล้วก็จะต้องตรวจสอบดูว่าเป็นจริงดังสถานการณ์ที่ตั้งเอาไว้หรือไม่ หากผิดพลาดก็หมายความว่าสัญญาณที่รับนั้นผิดพลาดไปจากสถานีส่งส่งออกมา ทั้งนี้ทั้งนั้นจะต้องผิดเป็นจำนวนคี่เท่านั้น คือผิดไป 1 บิท 3 บิท หรือ 5 บิท พร้อมกันจึงจะตรวจสอบได้ว่าผิด มองเห็นง่าย ๆ ว่าถ้าผิดเป็นจำนวนคู่ ผลรวมของจำนวนหนึ่งก็ยังเป็นคู่อยู่ดีทั้งนี้ทั้งนั้นไม่ได้หมายความว่า พาริตี้คี่ (Odd Parity) จะตรวจสอบการผิดพลาดเป็นจำนวนคี่ความจริงแล้วตรวจสอบความผิดพลาดได้เหมือนกับพาริตี้คู่ (Even Parity) แต่แทนที่จะตรวจสอบดูว่าสัญญาณที่รับเข้ามีจำนวนคู่ ก็ตรวจสอบดูว่ามีจำนวนคี่หรือเปล่าอย่างไรก็ตามโอกาสที่จะผิดพลาด 2 บิทพร้อมกันมีน้อยมาก

ย้อนกลับมาดูสัญญาณอะซิงโครนัสใหม่ หลังจากบิทพาริตี้แล้วก็จะต้องมีสต๊อปบิทซึ่งเป็น 1 ความกว้างของสต๊อปบิทอาจจะเป็น 1,1.5 หรือ 2 พัลส์ของสัญญาณนาฬิกา แล้วแต่ผู้รับและผู้ส่งจะตกลงใช้กันเอง การเริ่มใช้พอร์ตอนุกรม (ทางออกอนุกรม) จึงจำเป็นจะต้องตั้งค่าต่างๆ สำหรับเป็นการส่งแบบอนุกรมอันได้แก่

1. ความเร็วในการส่ง
2. ความยาวรหัส 1 อักขระ
3. บิทตรวจสอบ
4. จำนวนสต๊อปบิท

ในการส่งโทรพิมพ์หรือโทรเลขเมื่อก่อนนี้ใช้ความเร็วแค่ 70 บอด และ 110 บอด สำหรับคอมพิวเตอร์ความเร็วในการส่งมีให้เลือกตั้งแต่ 110,200,300,1200,2400,4800,9600 บอด และสูงไปกว่านั้น เนื่องจากมี IC หลายเบอร์ทำหน้าที่รับส่งอะซิงโครนัสให้ใช้ การส่งแบบอนุกรมจึงสะดวกสบายสำหรับคนออกแบบพอร์ตอนุกรม

จะเห็นว่ากลไกในการซิงโครนัสของการสื่อสารอะซิงโครนัส มีลักษณะเป็นไปทีละตัวอักขระ จำนวนพัลส์ของสัญญาณที่ส่งออกยังมีบางส่วน ใช้ในการควบคุมการส่งอยู่อันได้แก่ บิทสตาร์ท บิทสต๊อป และบิทพาริตี ทำให้ความเร็วการส่งอักขระต่อวินาทีน้อยลงไป การส่งสัญญาณด้วยความเร็ว 300 บอด สำหรับการเข้ารหัส 7 บิท ไม่ได้หมายความว่าส่งได้ 300 หารด้วย 7 อักขระต่อวินาที

<u>การสื่อสารแบบอะซิงโครนัสที่มีการแมตช์ความเร็ว</u>

การควบคุมการถ่ายโอนเทคนิค เราได้กล่าวถึงการถ่ายโอนข้อมูลจากแบบอนุกรมจากอุปกรณ์เครื่องหนึ่งไปยังอีกเครื่องหนึ่ง ซึ่งอาจจะเป็นคอมพิวเตอร์หรืออุปกรณ์สื่อสารชนิดอื่น โดยสมมติฐานว่าความเร็วในการเปลี่ยนสัญ-ญาณจากขนานเป็นอนุกรมได้เร็วพอ และฝ่ายรับเปลี่ยนจากอนุกรมเป็นขนานแล้วนำไปแสดงบนจอ พิมพ์ออกที่เครื่องพิมพ์หรือเก็บไว้ในดิสค์ทันทีได้ทันเวลาด้วยความเร็วในแต่ละการทำงานเท่ากันทั้งฝ่ายรับและฝ่ายส่ง ไม่มีการหน่วงเวลาหรือการอินเตอร์รัพต์ระหว่างตัวกลาง อย่างไรก็ตามสมมติฐาน นี้ย่อมไม่เป็นความจริง ฝ่ายส่งทำหน้าที่ส่งอย่างเดียวแต่ฝ่ายรับอาจต้องทำหน้าที่หลายอย่าง เช่น รับ แสดงผล เก็บ พิมพ์ เป็นต้น ความเร็วของฝ่ายรับหากไม่เพียงพอที่จะทำหลายอย่างให้ทันกับฝ่ายส่ง (แน่นอนย่อมขึ้นอยู่กับความเร็วในการส่ง) ก็จำเป็นจะต้องมีกลไกในในการควบคุมความเร็วในการส่งมอยู่หลายรูปแบบซึ่งอาจจะแบ่งออกได้เป็น 2 ลักษณะ คือ ส่งข้อ-มูลบอกการออนออฟการทำงาน (on-off data flow toggle) และหาที่เก็บข้อมูลชั่วคราวหรือสร้างบัฟเฟอร์ (Temporary data storage mechanism)

<u>การควบคุมความเร็วในการรับส่งข้อมูลที่ไม่เท่ากัน</u>

เนื่องจากการใช้ภาษาในระดับสูงเขียนเป็นโปรแกรมสำหรับการควบคุมการทำงานของการถ่ายโอนข้อมูลแบบอนุกรม อาจจะใช้เวลามากกว่าที่จะรับข้อมูลเข้ามาได้ทันกับสถานีส่งข้อมูลมา จำเป็นจะต้องมีวิธีการควบคุมไม่ให้เกิดการสูญหายของข้อมูลที่สถานีส่งมา วิธีการดังกล่าวนี้มีอยู่หลายวิธี คือ

<u>การมีบัฟเฟอร์ในการสื่อสารข้อมูล</u>

บัฟเฟอร์สำหรับการสื่อสารก็คือหน่วยความจำในคอมพิวเตอร์ซึ่งแบ่งแยกออกมาจากหน่วยความจำหลักสำหรับเก็บพักข้อมูลในการติดต่อชั่วคราว บัฟเฟอร์สำหรับการสื่อสารนี้ส่วนมากใช้สำหรับฝ่ายรับเท่านั้น เนื่องจากฝ่ายรับจำเป็นจะต้องตามฝ่ายส่งให้ทัน ถ้าหากฝ่ายรับใช้ภาษาแอส-

แฮมบลี้ควบคุม ถ้ามีความเร็วพออาจไม่จำเป็นต้องใช้บัฟเฟอร์สำหรับการสื่อสารเนื่องจากภาษาแอส-แฮมบลี้มีความเร็วสูง

ข้อมูลที่จัดส่งให้คอมพิวเตอร์ที่เป็นฝ่ายรับ ส่วนมากจะอ่านมาจากแฟ้มที่บันทึกไว้ในดิสค์ หากพิจารณาระหว่างการส่งข้อมูลออก ข้อมูลที่อ่านมาจากดิสค์จะมีลักษณะเป็นกลุ่ม ได้รับการนำมาสู่บัฟเฟอร์ การอ่านแต่ละกลุ่มดำเนินไปจนกระทั่งบัฟเฟอร์เต็ม การอ่านจะหยุดลงจนกระทั่งบัฟเฟอร์ถูกส่งออกไปหมดในลักษณะของเข้าก่อนออกก่อน ข้อมูลก็จะถูกอ่านออกมาใส่ในบัฟเฟอร์ส่งอีกครั้ง โดยปกติบัฟเฟอร์ส่งจะมีขนาด 255 ตัวอักษรหรือประมาณ 3 บรรทัดของ 80 อักษร

บัฟเฟอร์ของฝ่ายรับมีผลกระทบต่อการรับ - ส่งข้อมูลมากกว่าบัฟเฟอร์ส่ง บัฟเฟอร์รับทำหน้าที่เช่นเดียวกับบัฟเฟอร์ส่ง แต่ทิศทางของการไหลของข้อมูลอยู่ในทางตรงกันข้าม ฝ่ายรับข้อมูลเข้ามาเก็บไว้ในบัฟเฟอร์รับก่อนจนกว่าโปรแกรมควบคุมการสื่อสารจะนำข้อมูลออกไปจากบัฟเฟอร์รับเพื่อไปแสดงหรือพิมพ์หรือเก็บไว้ในแฟ้มก็แล้วแต่ ขอเพิ่มเติมอีกนิดว่าในระบบควบคุมการทำงานของไมโครคอมพิวเตอร์อย่างเช่น IBM PC มีกลไกบัฟเฟอร์รับส่งนี้ไว้อยู่ โปรแกรมในระดับสูงจึงเพียงแต่ทำหน้าที่ดึงเอาข้อมูลจากบัฟเฟอร์นี้ไปใช้ เราจะเห็นได้ชัดถึงความจำเป็นในการใช้บัฟเฟอร์เมื่อความเร็วในการส่งสูงเกินกว่า 600 บอด ภาษาในระดับสูง เช่น ภาษาเบสิกไม่สามารถที่จะรับข้อมูลจากพอร์ตอนุกรมได้ทันแน่ๆ ระบบควบคุมการทำงานจึงถูกออกแบบมาเพื่อการสื่อสารข้อมูล โดยการใช้อินเตอร์รัพต์เข้าช่วยเมื่อมีข้อมูลเข้ามาที่พอร์ตอนุกรทมเมื่อไร ระบบควบคุมจะอินเตอร์รัพต์การทำงานเพื่อดึงข้อมูลไปใส่ในบัฟเฟอร์รับทันที เพื่อไม่ให้ข้อมูลที่รับหายไปก่อนเมื่อมีตัวใหม่ส่งมาทับที่พอร์ตอนุกรม

หน้าที่ของโปรแกรมควบคุมการรับส่งก็คือ การอ่านข้อมูลจากบัฟเฟอร์รับไปใช้เมื่อถูกอ่านจากบัฟเฟอร์รับไปแล้ว ตัวที่อ่านออกไปก็จะหายไปจากบัฟเฟอร์ ลองนึกภาพดูจะเห็นว่าฝ่ายหนึ่งคือระบบควบคุมการทำงาน (OS) รับข้อมูลจากพอร์ตอนุกรมใส่บัฟเฟอร์ อีกฝ่ายหนึ่งคือโปรแกรมควบคุมการรับ ส่งดึงข้อมูลออกจากบัฟเฟอร์ เปรียบเสมือนคนหนึ่งตักน้ำใส่ตุ่ม อีกคนหนึ่งตักน้ำออกจากตุ่ม ถ้าฝ่ายที่ ตักออกมีความเร็วมากกว่าตุ่มก็อาจจะมีโอกาสแห้ง ในทางตรงกันข้ามถ้าฝ่ายที่ตักออกช้ากว่าฝ่ายที่ตักเข้าโอกาสที่จะล้นตุ่มก็ย่อมจะมี ในทางสื่อสารเรียกว่าบัฟเฟอร์รับโอเวอร์โฟลว์ (receive buffer overflow) การไหลล้นดังกล่าวทำให้ข้อมูลที่รับหายไป

โปรแกรมที่เขียนด้วยภาษาแอสแซมบลี้สามารถทำงานได้เร็ว และอาจไม่จำเป็นต้องใช้บัฟเฟอร์สำหรับการรับส่งเลยก็ได้ แต่ถ้าหากเขียนด้วยภาษาเบสิกจำเป็นจะต้องมีบัฟเฟอร์อย่างน้อย 1024 ไบท์ สำหรับการสื่อสารที่ความเร็วไม่เกิน 600 บอด

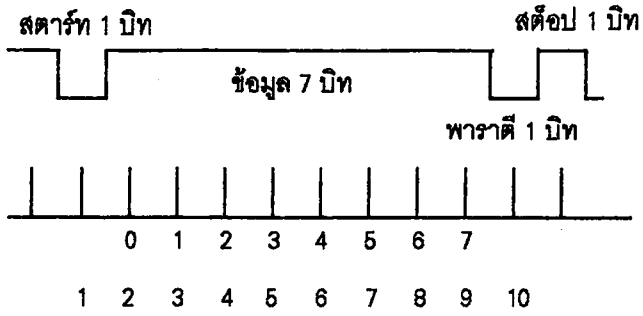
ในเครื่อง IBM PC เราสามารถกำหนดบัฟเฟอร์สำหรับการสื่อสารได้เมื่อเราเรียกใช้ BASIC หรือ BASICA เช่น A > BASICA/C:4096

เป็นการกำหนดบัฟเฟอร์การสื่อสารด้วยขนาด 4 กิโลไบท์ ถ้าหากเราไม่กำหนดค่าบัฟเฟอร์สื่อ สารเอาไว้ DOS จะตั้งค่าให้เท่ากับ 256 ไบท์ โดยการใส่ / C: ทำให้เราสามารถกำหนดบัฟเฟอร์ได้จาก 0 ถึง 32,767 ไบท์ อย่างไรก็ตามภาษาเบสิกใช้หน่วยความจำได้ 64 กิโลไบท์ การที่เรากำหนดค่าบัฟเฟอร์สื่อสารเอาไว้มากย่อมจะมีหน่วยความจำเหลือสำหรับใช้อย่างอื่นน้อยลง เมื่อเข้าสู่ภาษาเบสิกแล้วแรกเริ่มภาษาเบสิกจะบอกจำนวนหน่วยความจำเหลือใช้ทำงานจริง ในขณะที่โหลดโปรแกรมภาษาเบสิกเข้ามา ขนาดของโปรแกรมย่อมจะเข้าไปอาศัยที่ในหน่วยความจำที่เหลือทำให้หน่วยความจำที่จะใช้งานสำหรับเก็บตัวแปรต่างๆ น้อยลงไปอีก เราสามารถใช้คำสั่งโดยตรงสำหรับการตรวจดูขนาดของหน่วยความจำที่ว่างโดยการพิมพ์คำสั่ง PRINT PRE(0)

<u>การควบคุมโดยใช้ XON/XOFF</u>

ถึงแม้ว่าเราจะมีบัฟเฟอร์สำหรับการสื่อสารแล้วก็ตาม ในบางครั้งการถ่ายโอนข้อมูลด้วยความเร็วสูงและด้วยขนาดของแฟ้มที่จะทำการถ่ายโอนมีขนาดใหญ่กว่าบัฟเฟอร์สื่อสาร โอกาสที่ข้อมูลจะหายไปมีอยู่มาก ลองมาคำนวณดูง่ายๆ สมมติว่าเราใช้ความเร็วในการถ่ายโอนข้อมูล 9600 บิทต่อวินาที สต๊อปบิทเป็น 1 บิท ข้อมูล 7 บิท และพาริตีเป็น 1 คู่ ใน 1 ตัวอักษร จะต้องใช้ 11บิท

เพราะฉะนั้นฝ่ายรับจะต้องอ่านข้อมูลจากพอร์ตอนุกรมทุก 110/9600 ได้ผลประมาณ 0.001 วินาที หรือ 1 มิลลิวินาที ถ้าเปรียบเป็นพัลส์ได้ประมาณ 5000 พัลส์นาฬิกา (ความเร็วนาฬิกาของ IBM PC = 4.77 MHz หรือประมาณ 0.2 ไมโครวินาทีต่อหนึ่งพัลส์ 1000/0.2 = 5000) ในเมื่อบัฟเฟอร์ ไม่เพียงพอ ก็จำเป็นต้องควบคุมการรับส่ง โดยการบอกให้ฝ่ายส่งหยุดส่งชั่วคราว (XOFF) จนกว่าฝ่ายรับจะจัดการเอาข้อมูลออกจากบัฟเฟอร์สื่อสารหมดเสียก่อน จึงบอกให้ฝ่ายส่งจัดการส่งต่อไป (XON) ในรหัส ASCII XON มีค่าเท่ากับ 17 XOFF มีค่าเท่ากับ 19 เป็นหน้าที่ของนักเขียนโปรแกรมที่จะต้องจัดการส่ง XOFF ออกไปให้ฝ่ายส่งได้รับรู้ก่อนที่บัฟเฟอร์สื่อสารจะเต็มเสียก่อน



รูปที่ 2.5 แสดงรูปแบบของข้อมูล 1 ตัวอักษร

คอมพิวเตอร์เมนเฟรมส่วนมากจะมีระบบ XON/XOFF ให้ สำหรับการเชื่อมต่อทางด้านความเร็ว (Speed Matching) แต่โปรแกรมสื่อสารที่มีขายสำหรับ IBM PC ไม่มี XON/XOFF ทุกตัว โดยมากโปรแกรม ที่เขียนโดยภาษาแอสแซมบลี้จะมี XON/XOFF ให้ แต่โปรแกรมที่เขียนโดยภาษาเบสิกจะมีเพียงบางโปร แกรมเท่านั้น โปรแกรมที่เขียนโดยภาษาแอสแซมบลี้มีความเร็วพอที่จะคอยตรวจสอบดู XOFF ที่ส่งมาจากฝ่ายรับ แต่ถ้าเป็นภาษาเบสิก ความเร็วอาจจะไม่เพียงพอต่อการตรวจสอบ XOFF ที่ฝ่ายส่ง

<u>การใช้โปรโตคอล</u>

อีกเทคนิคหนึ่งในการควบคุมการรับส่งก็คือ การส่งโปรโตคอล (Protocol Transfer) เทคนิคนี้จำเป็นจะต้องมีเหมือนกันทั้งฝ่ายรับและฝ่ายส่ง โดยการใช้อักขระควบคุมในตารางของ ASCII สำหรับควบคุมการส่งข้อมูลออกมาเป็นกลุ่มที่มีขนาดคงที่

การใช้โปรโตคอลอาจจะใช้อักขระต่อไปนี้ ในการควบคุมการส่งข้อมูลเป็นกลุ่มๆ

ETB End of Transmission block

ETB มีค่าเท่ากับ 23 ในตาราง ASCII เป็นการบอกฝ่ายรับว่าขณะนี้สิ้นสุดการส่งข้อมูลกลุ่มที่หนึ่งแล้ว

ETX End of Text

ETX มีค่าเท่ากับ 03 ในตาราง ASCII เป็นการบอกฝ่ายรับว่าขณะนี้สิ้นสุดการส่งแล้ว

ENQ Enquiry

ENQ มีค่าเท่ากับ 05 ในตาราง ASCII เป็นอักขระที่ส่งมาจากฝ่ายรับขอให้ฝ่ายส่งส่งข้อมูลมา

NAK Negative Acknowledge

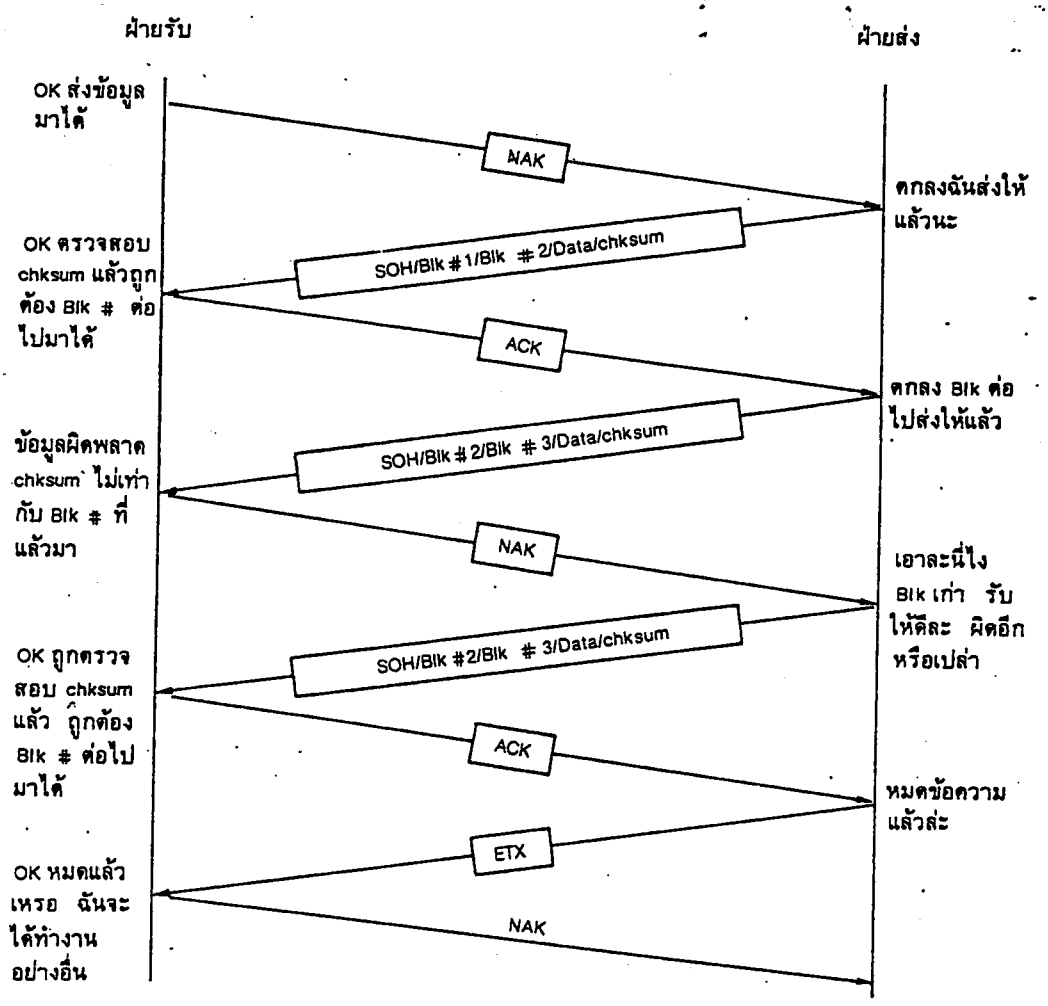
ในตารางรหัส ASCII มีค่าเท่ากับ 021 เป็นการบอกฝ่ายส่งว่าข้อมูลที่ได้รับนั้นผิดพลาด

ACK Acknowledge

เป็นการบอกฝ่ายส่งว่าข้อมูลที่ได้รับนั้นถูกต้องแล้วเป็นรหัส ASCII เท่ากับ 06

โปรโตคอลที่ใช้ใน IBM PC ส่วนมากจะเป็นโปรโตคอลที่มีชื่อว่า XMODEM ลักษณะการทำงานแสดงไว้ในรูปที่ 2.6

ฝ่ายส่งจะยังไม่ส่งข้อมูลจนกว่าจะได้รับ NAK จากฝ่ายรับ ฝ่ายส่งจะส่งข้อมูลออกไปโดยมีรูป แบบเริ่มด้วย SOH ตามด้วยอักขระ 2 ตัว สำหรับบอกกลุ่มของข้อมูลที่ส่งและตามด้วยส่วนเติมเต็ม 1 (1's complement) ของกลุ่มต่อไปที่จะส่ง ต่อจากนั้นก็จะเป็นข้อมูล 128 ไบท์ ตามหลังด้วยการตรวจสอบข้อผิดพลาดโดยวิธีตรวจสอบผลบวก (checksum) การ checksum คำนวณมาจากการบวกค่า ASCII ของข้อมูลที่ส่งออกไปทั้งหมด 128 ไบท์ แล้วหารด้วย 255 เศษที่เหลือก็คือค่า checksum ข้างฝ่ายรับเมื่อแยกเอา SOH และหมายเลขบล็อก (Block Number) ทั้งสองออกไปแล้วก็จะเอาข้อมูลทั้ง 128 ไบท์มารวมกันเพื่อหาค่า checksum เอา checksum ที่หาได้เปรียบเทียบกับค่าที่ได้รับ หากตรงกันก็ถือว่าข้อมูลที่ได้รับถูกต้องจึงส่งสัญญาณ ACK ไปให้ฝ่ายส่งได้รู้ว่าขณะนี้ได้รับข้อมูลไว้ถูกต้องแล้วส่งกลุ่มของข้อมูลต่อไปมาได้ ถ้าหากค่า checksum ไม่ถูกต้อง ฝ่ายรับก็จะส่ง NACK ให้ฝ่ายส่งเพื่อเป็นการบอกให้รู้ว่าข้อมูลที่ได้รับผิดพลาด ช่วยส่งกลุ่มของข้อมูลอันเก่ามาให้ทีเถอะ ฝ่ายส่งก็จะส่งข้อมูลกลุ่มเก่ามาให้ใหม่ การส่งใหม่จะดำเนินไป 9 ครั้ง หากยังคงได้รับแต่สัญญาณ NAK ฝ่ายส่งจะหยุดทำงานแสดงว่าตัวกลางการสื่อสารไม่ดี



รูปที่ 2.6 โปรโตคอล XMODEM

การที่ XMODEM ใช้เลขบอกกลุ่ม (Block Number) 2 ตัว (ตัวหนึ่งบอกกลุ่มที่ส่งขณะนี้ อีกตัวหนึ่งเป็นส่วนเติมเต็ม 1 ของกลุ่มต่อไป) เพื่อความแน่นอนว่ากลุ่มเดียวกันจะไม่ถูกส่งออกไปสองครั้ง ถ้าหากอักขระควบคุมการส่งเกิดสูญหายไประหว่างการส่ง ฝ่ายรับจะตรวจสอบดูว่ากลุ่มของข้อมูลที่ส่งมาเป็นกลุ่มที่ฝ่ายรับต้องการหรือไม่ ถ้าหากกลุ่มเก่าเกิดส่งมาใหม่อีกด้วยความผิดพลาด จาก ACK เป็น NACK ของฝ่ายส่ง ฝ่ายรับก็จะจับข้อมูลที่รับมาโยนทิ้งไปเมื่อทุกอย่างดำเนินไปอย่างเรียบร้อยจนสิ้นสุดแฟ้มที่จะส่ง ฝ่ายส่งก็จะส่ง ETX เป็นการบอกฝ่ายรับว่าหมดข้อความที่จะส่งแล้วละ

XMODEM เหมาะสำหรับ IBM PC เหนือโปรโตคอลชนิดอื่น 3 ข้อ คือ

1. ใช้อักขระควบคุมที่มีอยู่แล้วใน ASCII

2. สามารถจะใช้ภาษาในระดับสูงควบคุมได้ เช่น ภาษาเบสิก ภาษาปาสคาล

3. ต้องการบัฟเฟอร์สื่อสารแค่ 256 ไบท์

4. ระบบบริการข่าวสารด้วยคอมพิวเตอร์ โดยทั่วไปใช้โปรโตคอล XMODEM

ข้อดีของระบบการควบคุมการรับส่งแบบอนุกรมโดยการใช้ระบบการใช้โปรโตคอล ก็คือ

1. โปรโตคอลบางชนิดสามารถเลือกขนาดของกลุ่มข้อมูลได้

2. สามารถส่งข้อมูลที่ไม่ใช่ ASCII ได้ โดยไม่ต้องกลัวว่ารหัสนั้นจะไปทับกับรหัสควบคุมของ ASCII

3. การตรวจสอบโดยวิธี checksum มีความสามารถตรวจสอบความผิดพลาดได้ดีกว่า ใช้บิทพาริตีในอะซิงโครนัสในขณะที่บิทพาริตีสามารถให้ประสิทธิผลได้ 95 เปอร์เซ็นต์แต่ checksum สามารถ ให้ประสิทธิผลถึง 99.5 เปอร์เซ็นต์ หากพบการผิดพลาดด้วยบิทพาริตี ไม่ทำให้เกิดการส่งใหม่เกิดขึ้นแต่ข้อผิดพลาดจาก checksum ทำให้เกิดการส่งข้อมูลมาใหม่

2.1.2 <u>การสื่อสารแบบซิงโครนัส</u>

ข้อแตกต่างระหว่างวงจรส่งข้อมูลอนุกรมแบบซิงโครนัสและอะซิงโครนัสก็คือ ความต่อเนื่องของข้อมูลที่ส่ง ในแบบซิงโครนัสข้อมูลที่ส่งออกมาแบบต่อเนื่อง ไม่มีบิทสตาร์ทหรือบิทสต๊อป หรือแม้กระทั่งบิทพาริตีโปรโตคอลที่ใช้ในการส่งแบบซิงโครนัส จึงแตกต่างไปจากโปรโตคอลแบบอะซิงโครนัส มีโปรโตคอลหลายแบบที่ใช้ในการส่งแบบซิงโครนัสดังจะกล่าวต่อไปนี้

<u>โปรโตคอลไบซิงก์ (Bisyn Protocal)</u>

Bisyn ย่อมาจาก binary synchronous communication เป็นผลผลิตของบริษัท IBM Bisync เป็น โปรโตคอลในระดับอักขระซึ่งหมายความว่าอักขระแต่ละตัวมีขอบเขตที่แน่นอนแต่อักขระไม่มีสตาร์ทบิทหรือสต๊อปบิทเหมือนกับอะซิงโครนัส การซิงโคนัสกระทำกันที่จุดเริ่มต้นของการส่งข้อมูลเลยทีเดียว สถานีส่งจะส่งสัญญาณที่เรียกว่า leading pad character ไปยังสถานีรับก่อนที่จะเริ่มส่งข้อมูลตัวนำอักษร (leading pad character) จะประกอบด้วย 0 และ 1 สลับกันเพื่อให้สถานีรับจัด

สัญญาณนาฬิกาให้ตรงกัน นอกจากนั้นก่อนข้อมูลจะส่งออกมาจะต้องมีอักขระที่เรียกว่า syn ตามหลัง PAD มาก่อน และสถานีส่ง จำเป็นจะต้องบอกความยาวของข้อมูลมาในกลุ่มนี้และเครื่องหมายที่เป็นตัวบอกจุดเริ่มต้นของข้อมูลมาด้วย

อักขระ syn ในโปรโตคอล Bisyn ทำหน้าที่คล้ายกับบิทเริ่มต้นในอะซิงโครนัสซึ่งทำหน้าที่ "ปลุก"สถานีรับให้ตื่นมารับข้อมูล ขณะที่สถานีรับกำลังรอรับสัญญาณจากสถานีส่ง เครื่องรับอยู่ในสถานภาพที่เรียกว่า "Hunt" บิททุกบิทที่ผ่านเข้ามาจะถูกค้นหาอักขระ syn ก่อน เมื่อได้รับอักขระ syn แล้วจึงจะเริ่มต้นนับบิทที่เข้ามาเพื่อจุดเริ่มต้นของสัญญาณ เพื่อป้องกันโอกาสที่จะเกิดความผิดพลาด อักขระ syn จะส่งมา 2 ตัวก่อนที่จะเริ่มส่งข้อมูล

<u>โปรโตคอล SBLC และ HDLC</u>

SDLC ย่อมาจาก Synchronous Data Link Control และ HDLC ย่อมาจาก High Level Data Link Control โปรโตคอลทั้งสองนี้ไม่เหมาะสมสำหรับมือสมัครเล่นเพราะราคาเริ่มต้นแพง แต่ทั้งสองนี้เป็นมาตรฐานโปรโตคอลที่ใช้ในงานธุรกิจสำหรับเครื่องระดับเมนเฟรม อย่างไรก็ตามไมโครคอมพิวเตอร์ยังจำเป็นที่จะต้องติดต่อกับเมนเฟรม จำเป็นอยู่ดีที่ไมโครคอมพิวเตอร์จะต้องใช้โปรโตคอลทั้งสอง จึงขอกล่าวถึงโปรโตคอลทั้งสองไว้ย่อๆ ด้วยกัน ข้อแตกต่างที่เห็นชัดจะบอกเอาไว้ในขณะที่กล่าว

ข้อมูลที่ส่งโดย SDLC/HDLC เรียกว่า Information field และฟิลด์ดังกล่าวก็คือ เลขฐานสองที่วิ่งต่อเนื่องกันมาแบบอนุกรม ขนาดของฟิลด์อาจจะมีตั้งแต่ 0 จนถึงค่าสูงสุดที่หน่วยความจำจะรับได้ สายของข้อมูลดังกล่าวจะอยู่ในรูปของบิทข้อมูล Bisyn เป็นโปรโตคอลแบบเชิงตัวอักษร ซึ่งหมายความว่าไม่มีขอบเขตระหว่างอักษร ถ้าหากข้อมูลเป็นอักขระฝ่ายรับแยกออกเอาเองเมื่อรับข้อมูลทั้งหมดแล้ว

การต่อเนื่องของข้อมูลใน SDLC/HDLC แตกต่างไปจาก Bisyn การหยุดชั่วขณะของการส่งสามารถทำได้ใน Bisyn แต่ SDLC/HDLC ไม่ยอมให้ทำอย่างนั้น ความต่อเนื่องของสายข้อมูลจะต้องต่อเนื่องไปจนสิ้นสุด Information field ถ้าหากมีการหยุดหรือเบรกเกิดขึ้น ก่อนที่จะสิ้นสุด Information field ถือว่ามีความผิดพลาดเกิดขึ้น

ใน SDLC/HDLC สถานีรับส่งแบ่งเป็น 2 ฝ่าย คือ ฝ่ายปฐมภูมิและฝ่ายทุติยภูมิ สถานีปฐมภูมิจะเป็นตัวควบคุมการเชื่อมต่อข้อมูล

รูปแบบของ HDLC แสดงในรูปที่ 2.7

Begining flag	Address	Control	Packet heading	Infomation	Frame check	Ending flag

รูปที่ 2.7 รูปแบบของ HDLC

แฟล็กเริ่มต้น (Beginning flag) เป็นตัวอ้างอิงถึงตำแหน่งเริ่มต้นสำหรับฟิลด์แอดเดรส (Address field) และฟิลด์ควบคุม (control field) แฟล็กเริ่มต้นประกอบด้วย 01011110 ทำนองเดียวกันแฟล็กท้ายขบวน(Ending flag)เป็นตัวบอกว่า 16 บิทที่รับก่อนหน้านี้เป็นการตรวจสอบเฟรม ในขณะเดียวกันถ้าหาก เฟรมที่ส่งต่อเนื่องกัน แฟล็กท้ายขบวน (Ending flag) จะทำหน้าที่ทั้งเป็นแฟล็กเริ่มต้นและแฟล็กท้าย ขบวน จะเห็นว่าฮาร์ดแวร์ของโปรโตคอล SDLC/HDLC จะต้องคอยตรวจสอบรูปแบบของ 01111110 อยู่เสมอ เพราะถือว่าเป็นแฟล็ก เพื่อป้องกันความผิดพลาดในการตีความข้อมูลเป็นแฟล็ก SDLC/HDLC จะจัดการแทรกบิท "0" ลงในข้อมูลที่มี "1" ต่อเนื่องมากกว่า 5 บิท ฝ่ายรับจะต้องคอยแยก "0" ที่แทรกใส่เข้าไปเอาเองด้วยเหตุผลนี้เองราคาของฮาร์ดแวร์จึงแพง

พิจารณาฟิลด์แอดเดรส (Address field) ซึ่งมีอยู่ 8 บิท เป็นการบอกสถานีทุติยภูมิว่าต้องการส่งให้ตัวไหน ฟิลด์ควบคุม(control field)ประกอบไปด้วย 8 บิท ซึ่งจะคอยแยกการส่งข้อความเป็น 3 รูปแบบใหญ่ ๆ คือ

1. ฟอร์แมตการโอนย้ายข้อมูล
2. ฟอร์แมต supervisory
3. ฟอร์แมต non sequence

รูปแบบที่ 1 ใช้ทั่วไปในการส่งข้อมูล ในจำนวน 8 บิทนี้จะมีจำนวนเฟรมที่กำลังส่งและจำนวน เฟรมที่ได้รับอยู่ด้วย

2.2 มาตรฐาน RS 232C

โดยปกติไมโครคอมพิวเตอร์จะมีพอร์ตที่เป็นแบบอนุกรมเรียกชื่อกันว่า RS 232C อยู่ในตัวเองอยู่แล้วหลายเครื่องไม่มีมากับเครื่องอย่างเช่น IBM PC จำเป็นจะต้องมีการ์ดที่เรียกว่าอะซิงโครนัสอะแดปเตอร์ (Asyn chronous Communication Adapter) มาเสียบใส่

พอร์ต RS 232 นี้ทำหน้าที่รับและส่งข้อมูลในแบบอนุกรมเรียกว่า Universal Asynchronous Adapter เหตุที่มีชื่อเรียกว่า RS 232C ก็เนื่องจากสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์ของอเมริกาหรือ EIA ได้กำหนดมาตรฐานของอุปกรณ์การสื่อสารแบบอนุกรมเอาไว้ภายใต้ชื่อว่า RS 232C ความจริงมาตรฐานของการส่งข้อมูลแบบอนุกรมมีหลายมาตรฐานแต่ที่นิยมกันมากที่สุดสำหรับไมโครคอมพิวเตอร์ก็คือ RS 232C

หน้าที่สำคัญของการสื่อสารแบบอะซิงโครนัสก็คือ

รับสัญญาณ

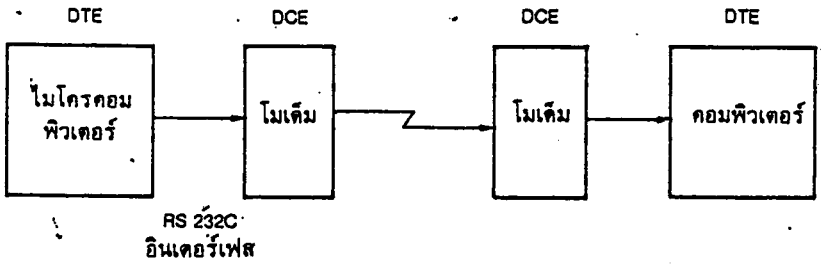
1. เปลี่ยนสัญญาณเข้ามาแบบอนุกรมให้เป็นแบบขนาน
2. ตรวจสอบความผิดพลาดของสัญญาณที่รับ
3. ตัดสต๊อปบิตและพาริตี้บิตออก
4. ส่งสัญญาณให้ซีพียูรู้ว่ารับสัญญาณไว้แล้ว

ส่งสัญญาณ

1. เปลี่ยนสัญญาณแบบขนานจากซีพียูค่อยทยอยส่งออกเป็นแบบอนุกรม
2. เพิ่มสต๊อปบิตและพาริตี้
3. เพิ่มสัญญาณควบคุมโมเดมที่ต่อเชื่อม (ถ้ามี)

มาตรฐาน RS 232C ได้จัดพิมพ์ขึ้นเมื่อ ปี ค.ศ. 1969 โดยสมาคมผู้ผลิตอุปกรณ์เล็กทรอนิกส์แห่งสหรัฐอเมริกา RS ย่อมาจาก Recommened Standard ส่วน 232 เป็นหมายเลขบ่งบอกของมาตรฐานตัวนี้ C เป็นหมายเลขของฉบับสุดท้ายของมาตรฐานตัวนี้ จุดประสงค์ของมาตรฐานตัวนี้ก็เพื่อบรรยายคุณลักษณะของการเชื่อมต่ออุปกรณ์รับส่งข้อมูลปลายทาง (Data Terminal Equipment DTE) กับอุปกรณ์สื่อสารข้อมูล (Data Communication Equipment DCE) สำหรับผู้ใช้ไมโครคอมพิวเตอร์ DTE ก็หมายถึงตัวไมโครคอมพิวเตอร์ และ DCE ก็หมายถึง โมเดม อุปกรณ์อื่นๆ เช่น เครื่องพิมพ์ที่รับ

สัญญาณแบบอนุกรมอาจจะเป็นได้ทั้ง DTE และ DCE ขึ้นอยู่กับผู้ผลิต ข้อแตกต่างของ DTE และ DCE จะเห็นได้จากรูปที่ 4.1 จากรูปนี้เราจะเห็นได้ว่า RS 232C มีส่วนสำคัญอย่างใหญ่หลวงสำหรับการสื่อสารข้อมูลระหว่างไมโครคอมพิวเตอร์

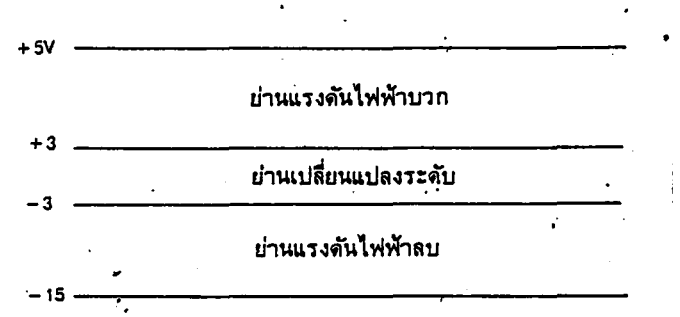


รูปที่ 2.8 การใช้ RS 232C เชื่อมต่ออุปกรณ์

ความจริงอีกประการหนึ่งของ RS 232C ก็คือความเร็วและระยะทางของการเชื่อมต่อ RS 232C สามารถเชื่อมต่อการถ่ายโอนข้อมูลได้จาก 0 - 20,000 บิตต่อวินาที ซึ่งเพียงพอสำหรับไมโครคอมพิวเตอร์ที่มีขนาดอัตราบอด 110 ถึง 9600 บอด ความยาวของสายเชื่อมต่อโดยสัญญาณตามมาตรฐานของ RS 232 จำกัดอยู่แค่ 50 ฟุต ซึ่งเพียงพอสำหรับการสื่อสารไมโครคอมพิวเตอร์กับอุปกรณ์รอบนอก เพื่อเป็นหลักประกันว่าข้อมูลถูกส่งออกไปอย่างถูกต้อง จำเป็นจะต้องมีข้อตกลงกันในเรื่องของสัญญาณที่ใช้ มาตรฐาน RS 232C กำหนดย่านของแรงดันไฟฟ้าในสัญญาณเพื่อสนองจุดประสงค์ข้างบน ดังแสดงในตารางที่ 2.1 และรูปที่ 2.9

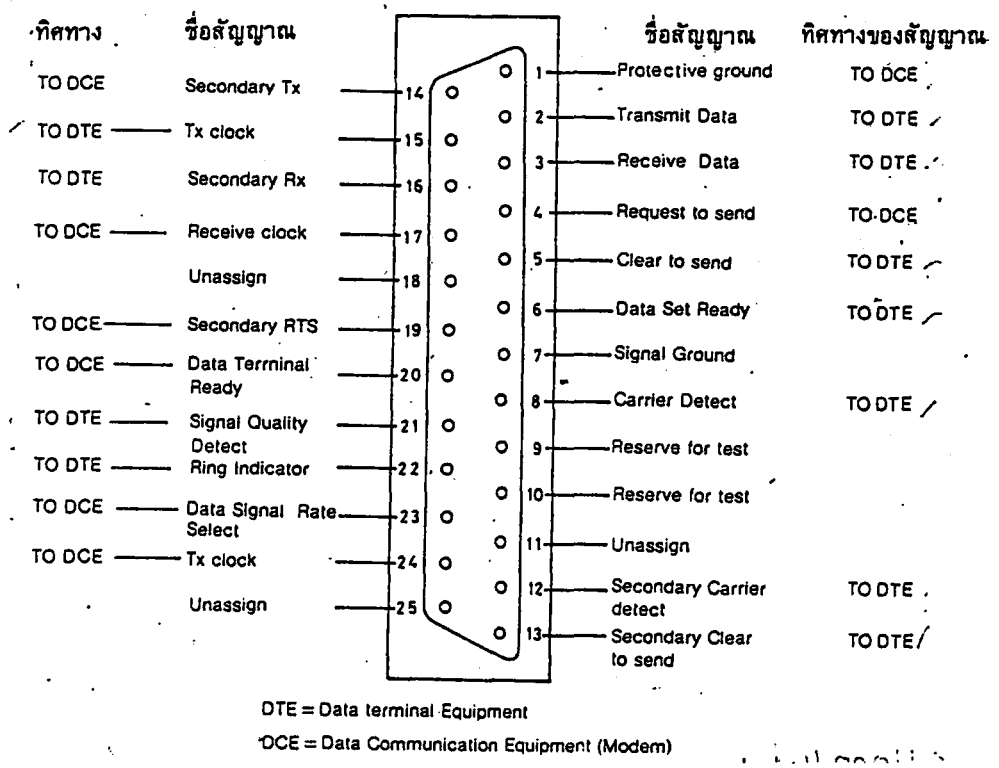
ตารางที่ 2.1

มาตรฐานของการใช้แรงดันไฟฟ้า			
แรงดันไฟฟ้า	สถานภาพลอจิก	สถานภาพของสัญญาณ	ฟังก์ชันในการควบคุม
บวก	0	สเปซ	ออน
ลบ	2	มาร์ค	ออฟ



รูปที่ 2.9 ย่านของแรงดันไฟฟ้าที่ใช้ในสัญญาณ RS 232C

สำหรับไมโครคอมพิวเตอร์บางเครื่อง ใช้แต่สัญญาณลอจิกออกมาเป็นสัญญาณของ RS 232C เลยอย่างเช่น อะซิงโครนัสอะแดปเตอร์ของ IBM PC ในกรณีเช่นนี้ระยะทางของสายที่เชื่อมต่ออาจจะไปได้สั้นกว่า 50 ฟุตดังที่กล่าวเอาไว้เนื่องจากระดับของกราวด์นเปลี่ยนแปลงไป อันเนื่องจากการสูญเสียไปในความต้านทานของสาย ผู้ที่เคยใช้ IBM PC อาจจะเคยประสบกับปัญหานี้มาแล้วว่า ทำไมต่อสัญญาณ RS 232C เกินกว่า 10 ฟุต แล้วใช้งานไม่ได้ แต่อย่างไรก็ตาม RS 232 ของ IBM PC ยังมีโอกาสให้เลือกใช้ 20 มิลลิแอมแปร์ กระแสวนกลับแทนแรงดันไฟฟ้า (20 mA loop จะกล่าวถึงในหัวข้อถัดไป) ในทางฟิสิกส์แล้ว มาตรฐานของ RS 232C กำหนดข้อต่อแบบ DB-25 แต่ละขาของข้อต่อกำหนดไว้ดังในรูปที่ 2.10 อย่างไรก็ตามผู้ผลิตไมโครคอมพิวเตอร์อาจจะต้องใช้ข้อต่อชนิดอื่นที่นอกเหนือไปจาก DB-25 ยกตัวอย่างเช่น Fujitsu F-8 , IBM AT , IBM Jr เป็นต้น ตัวเมียของข้อต่อควรจะอยู่ที่ตัวโมเด็ม ขณะที่ตัวผู้ควรจะอยู่ที่ asynchronous communication adapter หรือที่ตัวไมโครคอมพิวเตอร์เอง อย่างไรก็ตามผู้ผลิตหลายรายการไม่ได้ทำตามกฎที่ว่านี้



รูปที่ 2.10 การกำหนดของข้อต่อ RS 232

สัญญาณต่าง ๆ ถูกมอบหมายให้ทำหน้าที่ดังนี้

Transmit Data (TD ขาที่ 2)

เป็นสัญญาณที่ส่งออกจาก DTE (หรือตัวไมโครคอมพิวเตอร์) ไปยังโมเด็มหรือต่อเข้าโดยตรงกับไมโครคอมพิวเตอร์ตัวอื่น หรือเครื่องพิมพ์ เมื่อไม่มีสัญญาณส่งออกสถานภาพของลอจิกที่ขานี้จะมีค่าเท่ากับ "1" หรือเทียบเท่ากับสต๊อปบิต

Receive Data (RD ขาที่ 3)

เป็นทางของสัญญาณเข้าไปยัง DTE หรือไมโครคอมพิวเตอร์เมื่อไม่มีสัญญาณรับเข้ามา ขานี้จะมีสถานภาพทางลอจิกเป็น "1"

Request To Send (RTS ขาที่ 4)

ใช้สำหรับส่งสัญญาณไปยังโมเด็มหรือเครื่องพิมพ์เป็นการเรียกร้องที่จะส่งสัญญาณมาทางขา 2 สัญญาณนี้ใช้คู่กับ CTS หรือ Clear to send อุปกรณ์รับหากได้รับสัญญาณ RTS จะตรวจสอบตัวเองว่าพร้อมจะรับสัญญาณได้หรือยัง หากพร้อมที่จะรับก็ส่งสัญญาณออกไปที่สาย CTS

Clear To Send (CTS ขาที่ 5)

ดังอธิบายไว้ใน RTS เมื่อสัญญาณนี้อยู่ในสถานะออฟ (negative voltage หรือลอจิก "1") หมายความว่า อุปกรณ์รับกำลังบอกว่าพร้อมที่จะรับข้อมูลแล้ว

Data Set Ready (DSR ขาที่ 6)

เมื่อสัญญาณสายนี้อยู่ในสถานะออน (หรือลอจิก 0) เป็นการบอกไมโคคอมพิวเตอร์หรือฝ่ายส่งว่า โมเด็มต่อเข้ากับสายโทรศัพท์เรียบร้อยแล้วและพร้อมที่จะส่งได้แล้ว โมเด็มที่มีการหมุนหมายเลขอัตโนมัติจะส่งสัญญาณสายนี้ไปบอกให้คอมพิวเตอร์รู้ว่าต่อโทรศัพท์ได้สำเร็จแล้ว

Signal Ground (SG ขาที่ 7)

SG ทำหน้าที่เป็นระดับแรงดันอ้างอิงสำหรับทุก ๆ สายของสัญญาณ จะมีแรงดันเป็น "0" เมื่อเทียบกับสัญญาณตัวอื่น

Carrier Detect (CD ขาที่ 8)

โมเด็มจะส่งสัญญาณที่อยู่ในสถานะออน (ลอจิก"0")ไปบอกไมโครคอมพิวเตอร์ เมื่อได้รับสัญญาณจากโมเด็มของอีกฝ่ายหนึ่ง สัญญาณนี้จะนำไปจุด LED บอกว่าได้รับสัญญาณจากโมเด็มอีกฝ่ายหนึ่งแล้ว ไฟ LED จะอยู่บนหน้าปัดของโมเด็มเอง

Data Terminal Ready (DTR ขาที่ 20)

คอมพิวเตอร์เปิดสัญญาณสายนี้ให้ออน (ลอจิก "0") เมื่อพร้อมที่จะติดต่อกับโมเด็ม โมเด็มส่วนมากจะไม่รายงานสถานภาพของตัวเอง (CD , DSR และ CTS) ให้คอมพิวเตอร์รู้ หากคอมพิวเตอร์ไม่เปิดสัญญาณ DTR

Ring Indicator (RI ขาที่ 22)

สัญญาณนี้ใช้ในโมเด็มที่เป็นระบบตอบได้อัตโนมัติ (Auto - answer) สัญญาณนี้จะออนเมื่อมีสัญญาณกระดิ่งมา และออฟระหว่างเสียงดังของกระดิ่ง

ท่านผู้อ่านอาจจะสับสนระหว่างสถานภาพของลอจิกกับสถานภาพของสัญญาณ โดยปกติเราจะคุ้นเคยอยู่กับความรู้สึกที่ว่า เมื่อแรงดันเป็นบวก หรือสัญญาณออนลอจิกน่าจะเป็น "1" สำหรับสัญญาณต่าง ๆ ที่กล่าวมานี้จะมีลักษณะตรงกันข้าม ทำไมเขากำหนดกฎเกณฑ์ออกมาอย่างนี้ ก็เพราะว่าแต่เดิมนั้นการติดต่อกันทางโทรเลขการทำงานของสัญญาณจะต้องครบวงจรทั้งฝ่ายส่งและฝ่ายรับ เมื่อลอจิกเป็น "0" หรือขณะที่ไม่มีอะไรส่งควรจะมีสัญญาณทางไฟฟ้าครบวงจรอยู่ตลอดเวลา จะได้รู้ว่าวงจรไม่ขาดระหว่างทางตรงไหน ควรจะรู้ว่าวงจรครบอยู่ตลอดเวลาก็โดยการให้ค่าแรงดันที่ฝ่ายส่ง ดังนั้นจึงถือกันว่าสัญญาณไฟบวกใช้เป็นลอจิก "0"

ตารางที่ 2.2 คุณลักษณะโดยย่อของสัญญาณ RS 232C

Driver output logic levels with	15V>O_h>5V
3k to 7k load	-5>O_l>-15 โวลต์
Driver output voltage when open	Vo <25 โวลต์
circuit	
Driver output impedance with Power off'	R_o>300 Ohms
Output short circuit current	I_o < 0.5 A
Driver slew rate	dv/dt < 30 V/s
Receiver input impedance	7k >R_{in}>3k
Receiver input voltage	+15 compatible with
	driver
Receiver output with open	MARK
circuit input	
Receiver output with +3V input	SPACE
Receiver output with -3V input	MARK
+15	LOGIC 0 = SPACE =
+5	CONTROL ON
+5	Noise Margin
+3	
+3	Transition Region
-3	
-3	Noise Margin
-5	
-5	LOGIC 1 = MARK =
-15	CONTROL OFF

2.3 MODEMS

Modems ประกอบด้วย modulator และ demodulator ใน telephone terminalogy จะเรียกว่า data set modem ใช้ในการต่อ digital baseband signals form กับ analog form โดยส่งผ่านสื่อกลาง analog โดยการสื่อสารแบบ analog มี bandwidth 3 kHz ซึ่ง bandwidth ของสัญญาณ analog จะมีความสัมพันธ์กับอัตราการส่งข้อมูล ซึ่งส่วนใหญ่จะใช้ 1200 bit/s ใน 3 kHz analog channel ซึ่งจะมีราคาค่อนข้างถูกและในปัจจุบันนี้ก็สามารถทำได้ถึง 9600 bit/s ใน 3 kHz analog channel โดยใช้เทคนิคการ modulation ที่ซับซ้อน

โมดูเลท (Modulate)

คือ ขบวนการอย่างหนึ่งที่นำเอาคลื่นความถี่สูงค่าหนึ่งที่เรียกว่า สัญญาณพาหะ (Carrier Signal) เข้าไปรวมกับคลื่นความถี่เสียงซึ่งจะทำให้เกิดความถี่ใหม่ขึ้นอีกสองค่า เช่น สมมติให้สัญญาณพาหะมีความถี่ เป็น F_1 และความถี่เสียงเป็น F_2 เมื่อผ่านขบวนการโมดูเลเตอร์แล้ว ความถี่ที่เกิดขึ้นจะเป็น F_1+F_2 , F_1-F_2 และ F_1 อีกหนึ่งค่า ซึ่งจะทำให้เกิดแถบความถี่ที่เรียกว่า Upper Side Band และ Lower Side Band ขึ้นมา

โดยทั่วไปขบวนการโมดูเลทของสัญญาณอนาล็อกจะมีเทคนิคอยู่ 3 แบบ คือ

1. แอมปลิจูดโมดูเลชั่น (Amplitude Modulation) หรือแบบ AM.
2. ฟรีเคนซี่โมดูเลชั่น (Frequency Modulation) หรือแบบ FM.
3. เฟสโมดูเลชั่น (Phase Modulation) หรือแบบ PM.

ดีโมดูเลท (Demodulate)

เป็นขบวนการที่ทำงานตรงข้ามกับขบวนการโมดูเลท คือ ทำการแยกเอาสัญญาณพาหะออก ซึ่งจะทำให้ได้สัญญาณของข้อมูลอย่างแท้จริงเพียงอย่างเดียว จุดประสงค์สำคัญที่จำเป็นต้องมีการโมดูเลทสัญญาณความถี่เสียงนี้ ก็เพื่อทำให้สามารถส่งไปได้ในระยะไกลๆ โดยเกิดการผิดเพี้ยนของสัญญาณน้อยที่สุด

การโมดูเลทสัญญาณดิจิตอลซึ่งมีเพียง 2 ระดับ คือ 0 กับ 1 โดยทั่วไปเทคนิคที่ใช้มีอยู่หลายแบบ คือ

1. ฟรีเคนซี่ชิฟคีย์อิ้ง (Frequency Shift Keying : FSK)
2. เฟสชิฟคีย์อิ้ง (Phase Shift Keying : PSK)
3. แอมปลิจูดชิฟคีย์อิ้ง (Amplitude Shift Keying)

โมเด็ม เป็นอุปกรณ์ที่ทำหน้าที่ร่วมระหว่างขบวนการโมดูเลท และดีโมดูเลท (Modulate/DEModulate)นั่นเอง ซึ่งบางทีก็อาจจะเรียกว่าเป็น "ตัวแปลงสัญญาณ" (Signal Converter) ก็ได้ โดยอุปกรณ์ที่ทำการแปลงสัญญาณดิจิตอลที่ส่งออกมาจากเครื่องคอมพิวเตอร์ให้เป็นสัญญาณเสียง (Voice Signal) เพื่อส่งผ่านไปในสายโทรศัพท์ได้ โดยต้องผ่านขบวนการโมดูเลทก่อน ส่วนทางคอมพิวเตอร์ด้านรับจะมีการต่ออุปกรณ์โมเด็มคั่นไว้ เมื่อโมเด็มด้านรับรับสัญญาณเสียงเข้ามาก็จะทำการแปลงสัญญาณเสียงให้กลับไปเป็นสัญญาณดิจิตอล (โดยใช้ขบวนการดีโมดูเลท) ก่อนที่จะส่งผ่านไปยังเครื่องคอมพิวเตอร์ต่อไป ในการส่งผ่านสัญญาณระหว่างเครื่องคอมพิวเตอร์กับอุปกรณ์รอบข้างเช่น เครื่องพิมพ์ที่ตั้งอยู่ในระยะไกลออกไป ก็จำเป็นอย่างยิ่งที่จะต้องส่งสัญญาณเข้าไปในสายโทรศัพท์ในรูปของสัญญาณอนาล็อก และเมื่อส่งถึงด้านรับก็จะแปลงกลับมาเป็นสัญญาณดิจิตอลเหมือนเดิม

<u>การแบ่งชนิดของโมเด็มตามอัตราส่งข้อมูล</u>

1.อัตราการส่งข้อมูลต่ำ (Low-speed) มีอัตราการส่งข้อมูลไม่เกิน 600 bps

2.อัตราการส่งข้อมูลปานกลาง(Medium-speed)มีอัตราการส่งข้อมูลระหว่าง1200 ถึง 9600 bps

3.อัตราการส่งข้อมูลสูง (High-speed) มีอัตราการส่งข้อมูลมากกว่า 9600 bps

โมเด็มชนิดอัตราการส่งข้อมูลต่ำ

The Bell 103 Modem : เป็นโมเด็มชนิดที่มีอัตราการส่งข้อมูลต่ำที่นิยมใช้กันอย่างแพร่หลาย เป็นผลผลิตของบริษัท Bell-Laboratory สหรัฐอเมริกา นอกจากนี้ก็มีผลิตภัณฑ์ของบริษัทที่ทำเลียนแบบ คือ มีความสามารถเหมือนกับโมเด็ม Bell 103 หรือที่เรียกว่า Bell 103 compatible ซึ่งโมเด็มชนิดนี้จะพบว่ามีการติดตั้งใช้งานในเครื่องคอมพิวเตอร์เมนเฟรมทั่วๆไป และอาจจะมีใช้บ้างในเครื่องไมโครคอมพิวเตอร์บางยี่ห้อสำหรับใน Bell 103 compatibel Modem นั้น คุณสมบัติบางอย่างอาจจะขาดหายไปคือ ไม่คอมแพททิเบิลกับ Bell 103 ทุกประการ สำหรับความสามารถของ Bell 103 อาจจะสรุปได้ว่าประกอบด้วย

1. สามารถทำงานได้ทั้งเป็นออริจิเนติ้ง (Originating) หรือ แอนเซอริ่ง (Answering) หรือ CY lead-Controlled ก็ได้ สำหรับ

Originating หรือ ในระบบโทรศัพท์เรียกว่า เป็นผู้เรียก นั่นเอง

Answering หรือ ในระบบโทรศัพท์เรียกว่า เป็นผู้ตอบรับ

CY lead-Controlled หมายถึง เป็นการทำงานของโมเด็มที่สามารถเปลี่ยนแปลงโหมดของการทำงานเป็นผู้เรียก หรือผู้ตอบรับก็ได้ โดยใช้ซอฟท์แวร์ควบคุมการทำงานผ่าน RS-232 ขา 11

2. จะยกเลิกการเชื่อมต่อ (ตัดสายโทรศัพท์) ทันทีถ้าสัญญาณ incoming carrier หายไป

3. สามารถที่จะสร้างสัญญาณเอาท์พุทให้มีระดับตามที่ต้องการได้ และสามารถปรับให้มีอัตราส่วนของสัญญาณข้อมูลต่อสัญญาณรบกวน(Signal-to-noise ratio)และคุณลักษณะของสัญญาณไฟฟ้าตามต้องการได้

4. สามารถสร้างระดับเสียง (tone) หรือสัญญาณพัลส์ที่ถูกต้องเพื่อใช้ในการเรียก/ต่อโทรศัพท์ไปยังสถานที่ไกลๆได้ ซึ่งสัญญาณดังกล่าวนี้จะต้องสอดคล้องกับสัญญาณที่ใช้ในเครือข่ายโทรศัพท์หรือ PSTN (Package Switch Telephone Network) การทำงานในลักษณะนี้ เรียกว่าเป็นการทำงานแบบเรียก โทรศัพท์อัตโนมัติ (Auto dial)

5. มีความสามารถที่จะใช้ซอฟท์แวร์ควบคุมการบ่งบอกสถานะขณะนั้นๆ เพื่อตอบรับต่อการเรียกเข้ามาว่าขณะนี้โทรศัพท์ไม่ว่าง

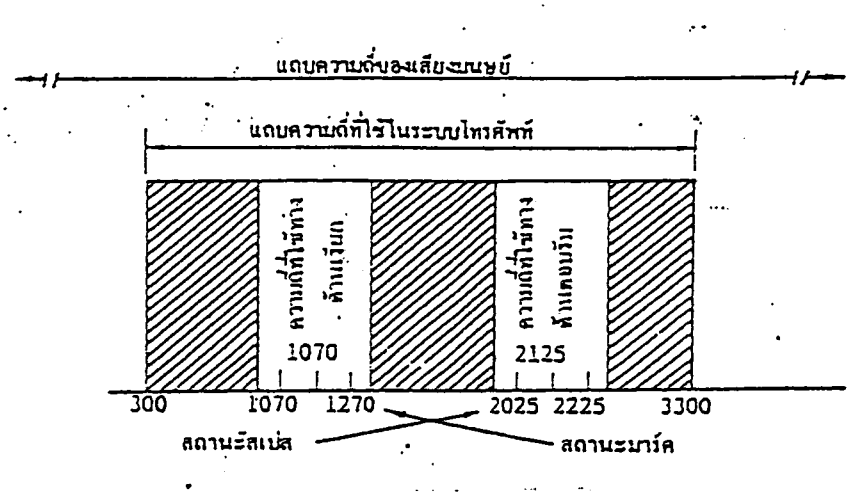
นอกจากคุณลักษณะต่างๆ ดังกล่าวแล้ว Bell-103 โมเด็มยังสามารถใช้งานร่วมกับการทำงานที่เป็นข้อยกเว้นพิเศษของสัญญาณควบคุมตามมาตรฐาน RS-232Cได้อีกด้วย

เทคนิคการโมดูเลชั่น

การกำหนดความถี่ของด้านผู้เรียกกับผู้ตอบรับสัญญาณดิจิตอล ประกอบด้วยสัญญาณ 2 สถานะ คือ 1 และ 0 ดังนั้นระดับโวลเตจหนึ่งแทน 0 อีกระดับโวลเตจหนึ่งก็จะแทนด้วย 1 เช่นกัน จึงได้มีการกำหนดความถี่ของสัญญาณ เพื่อใช้แทนระดับโวลเตจของทางด้านผู้เรียกกับผู้ตอบรับ ซึ่งต้องใช้ต่างๆ กัน 4 ความถี่ เพื่อไม่ให้เกิดการรบกวนของสัญญาณ (ในกรณีที่มีการสื่อสารแบบฟลูดูเพล็กซ์) โดยทางด้านส่งจะต้องใช้ความถี่ของสัญญาณ 2 ค่า เพื่อแทนโวลเตจระดับ 1 กับระดับ 0 และทางด้านรับก็เช่นเดียวกัน แต่ความถี่ที่ใช้จะต่างกัน ในกรณีของ Bell-103 ได้มีการกำหนดความถี่ของสัญญาณที่ใช้ไว้ดังนี้ คือ

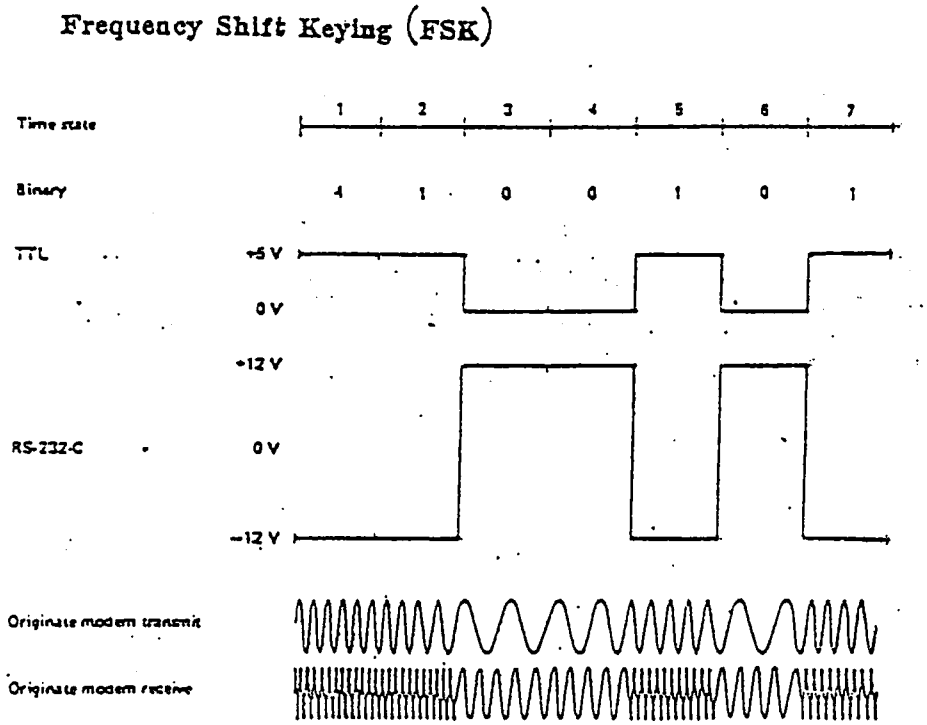
ความถี่ 1070 Hz สำหรับทางด้านส่ง แทนลอจิก 0
ความถี่ 1270 Hz สำหรับทางด้านส่ง แทนลอจิก 1
ความถี่ 2025 Hz สำหรับทางด้านรับ แทนลอจิก 0
ความถี่ 2225 Hz สำหรับทางด้านรับ แทนลอจิก 0

โดยแสดงเป็นแถบความถี่ได้ดังรูป 2.11



รูปที่ 2.11 แสดงแบนด์วิดท์ของความถี่ที่กำหนดให้ใช้ในโมเด็มชนิด Bell-103 โดยใช้งานร่วมกับ PSTN ได้

จากรูป จะเห็นว่าความถี่ที่ระบุมานั้น จะต้องมีความสัมพันธ์กันระหว่างโมเด็มทางด้านส่ง (Originate modem) ที่ทำการส่งสัญญาณด้วยความถี่ๆ หนึ่ง ส่วนโมเด็มทางด้านรับ (Answer modem)ก็จะรับสัญญาณความถี่ค่านั้นเข้ามา แล้วตอบกลับไปด้วยสัญญาณที่ความถี่อีกค่าหนึ่ง ซึ่งเทคนิคในการนำสัญญาณดิจิตอลมาโมดูเลทเข้ากับสัญญาณอนาล็อก แล้วส่งผ่านตัวนำออกไป เราเรียกว่าเทคนิคของ FSK หรือ Frequency Shift Keying



รูปที่ 2.12 แสดงสัญญาณที่เกิดจากการส่งอักษร S ด้วยเทคนิคการโมดูเลท (รวมถึงวิธี FSK ด้วย)

จากรูปแสดงให้เห็นว่าเทคนิค FSK นำมาใช้ในการส่งสัญญาณดิจิตอลได้อย่างไร ในกรณีนี้สมมติว่าเราต้องการจะส่งอักษร S ออกไป ในรหัสแอสกี้ (ASCII) แทนตัวอักษร S ด้วยเลข 53H หรือ 53 ในฐานสิบหก หรือ ในฐานสองแทนด้วย 1010011 นั่นเอง ดังนั้นในบรรทัดแรกแสดงถึงสถานะเวลา (Time State) ของสัญญาณดิจิตอลที่แทนเลข 101011 ขนาด 7 บิท ตามมาตรฐานของรหัสแอสกี้และส่งออกไปแบบอนุกรม

ในบรรทัดต่อมาแสดงถึงสัญญาณดิจิตอลที่แทนรหัสฐานของ 1010011 โดยมีบิทต่ำสุด (Least Significant Bit:LSB) อยู่ทางซ้ายมือ

ในบรรทัดที่ 3 แสดงถึงสัญญาณเอาท์พุทที่ได้จาก TTL (Transistor Logic) เพราะอุปกรณ์จำพวก TTL ใช้กับระดับไฟที่ 5 โวลต์ กับ 0 โวลต์ ดังนั้นจึงแทนลอจิก 1 ด้วยระดับไฟ 5 โวลต์ และลอจิก 0 ด้วยระดับไฟ 0 โวลต์

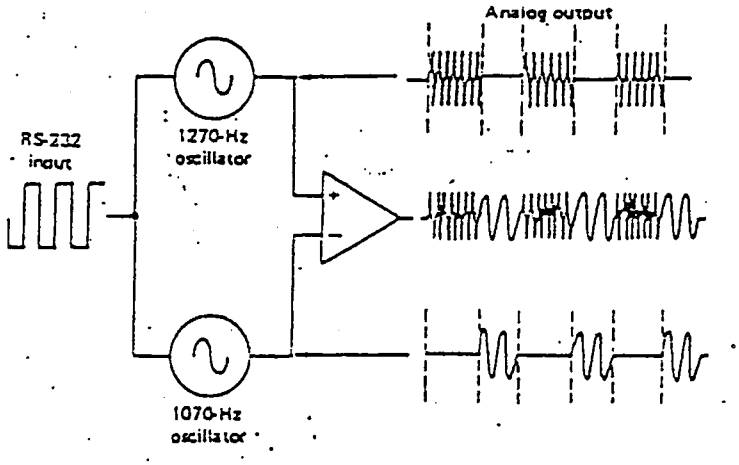
ในบรรทัดที่ 4 แสดงถึงสัญญาณที่ต่อผ่านพอร์ท RS-232C ออกมาโดยใช้กับไฟระดับ +12 V และ -12 V ดังนั้นจึงแทนลอจิก 0 ด้วยระดับไฟ +12 V และลอจิก 1 ด้วยระดับไฟ -12 V

สำหรับในสองบรรทัดสุดท้ายแสดงถึงสัญญาณที่ผ่านการโมดูเลทด้วยเทคนิค FSK ซึ่งส่งผ่านระหว่างโมเด็ม จะสังเกตเห็นว่าในกรณีของเทคนิค FSK นี้ ถ้าระดับลอจิกของสัญญาณมีค่าเป็น 1 ความถี่ของสัญญาณที่ผ่านการโมดูเลทจะสูงขึ้นและถ้าระดับลอจิกของสัญญาณมีค่าเป็น 0 ความถี่ก็จะลดลง ในรูปนี้ความถี่กำหนดเพียงคร่าวๆ เท่านั้น จากที่กล่าวมาแล้วว่าสัญญาณที่ส่ง/รับ ใน Originate Modem และ Answer Modem นั้นจะต่างกัน ในกรณี Originate Modem จะส่งสัญญาณ 2 บิทแรก ที่มีระดับลอจิกเป็น 1 ออกไปด้วยความถี่ของสัญญาณที่ผ่านการโมดูเลทแล้วเท่ากับ 1270 Hz ส่วนในบิทที่ 3 และ 4 แทนลอจิก 0 จะส่งออกไปด้วยความถี่ 1070 Hz และในบิทที่ 5 และ 7 ก็จะถูกส่งออกไปด้วยความถี่ 1270 Hz ส่วนในบิทที่ 6 ก็จะถูกส่งด้วยความถี่ 1070 Hz ตามลำดับ ดังนั้นจึงสรุปได้ว่าเทคนิคของ FSK ก็คือเทคนิคในการโมดูเลทสัญญาณดิจิตอลด้วยวิธีฟรีเคนซี่โมดูเลชั่น (FM) นั่นเอง

แต่ในสถานะที่ Originating Modem ทำหน้าที่รับสัญญาณเข้ามา ความถี่ของสัญญาณที่แทนลอจิก 4 คือ ในบิทที่ 1,2,5 และบิทที่ 7 จะถูกโมดูเลทส่งออกมาด้วยความถี่ 2225 Hz ส่วนในบิทที่ 3,4 และบิทที่ 6 จะถูกโมดูเลทส่งออกมาด้วยความถี่ 2025 Hz ซึ่งจุดสำคัญก็คือ ป้องกันการรบกวนกันระหว่างสัญญาณจาก Originating และ Answering ในสายส่งนั่นเอง

2.3.1 โมเด็มทรานสมิทเตอร์

หลักการในการสร้างสัญญาณต่างๆ ภายในโมเด็ม



รูปที่ 2.13 โครงสร้างในการกำเนิดสัญญาณ FSK

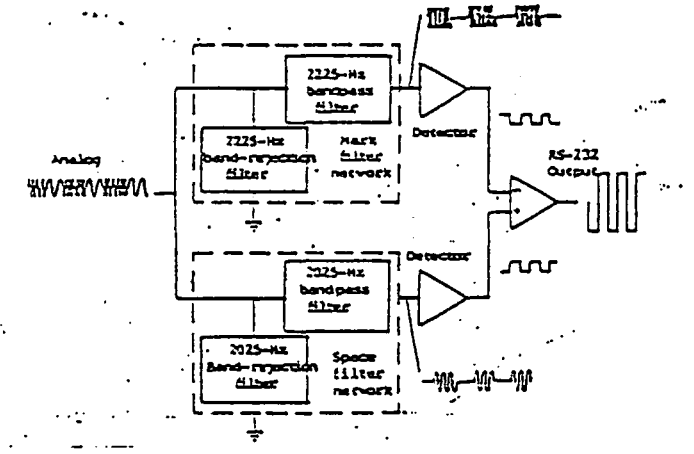
จากรูป จะเห็นว่าในโมเด็มจะประกอบไปด้วยออสซิลเลเตอร์ที่กำเนิดสัญญาณพาหะ (carrier) ด้วยความถี่ 1270 Hz และ 1070 Hz โดยที่ออสซิลเลเตอร์ตัวบนจะทำงานเมื่อสัญญาณที่เข้ามามีระดับโวลเตจเกิน 5 โวลต์ ส่วนออสซิลเลเตอร์ตัวล่างจะทำงานและตัวบนจะหยุดทำงานเมื่อระดับโวลเตจของสัญญาณที่เข้ามาเกินกว่า +5 โวลต์ เมื่อรับเอาท์พุทที่ออกจากพอร์ท RS-232C เข้ามายังอินพุทของออสซิลเลเตอร์ ก็จะสามารถทำให้ออสซิลเลเตอร์ทั้งสองตัวทำงานได้ดังนี้

เมื่อสัญญาณที่ผ่านพอร์ท RS-232C เข้ามามีระดับโวลเตจเท่ากับ -12 โวลต์ ออสซิลเลเตอร์ที่ผลิตสัญญาณพาหะความถี่ 1270 Hz ก็จะทำงาน แต่ออสซิลเลเตอร์ที่ผลิตสัญญาณพาหะความถี่ 1070 Hz หยุดทำงานเราทราบแล้วว่าระดับโวลเตจ -12 โวลต์จะแทนลอจิก 1 และเมื่อผ่านการโมดูเลทแล้วความถี่ 1270 Hz ทางด้านโมเด็มทรานมิสเตอร์ก็จะแทนลอจิก 1 นั่นเอง ดังนั้นจากรูปจะแสดงถึงการแปลงสัญญาณจากพอร์ท RS-232C ที่แทนลอจิก 1 ไปเป็นสัญญาณข้อมูลที่แทนลอจิก 1 เช่นกัน

สำหรับในกรณีที่สัญญาณข้อมูลที่จะส่งมีระดับโวลเตจเป็น +12 โวลต์ ออสซิลเลเตอร์ที่ผลิตความถี่ 1070 Hz ก็จะทำงานแทน และทำการผลิตสัญญาณที่แทนลอจิก 0 ออกมา หลังจากนั้นจึงทำการรวมสัญญาณเอาท์พุทจากออสซิลเลเตอร์ทั้งสองตัวเข้าด้วยกัน โดยผ่านเข้าไปยังอินพุทของอ็อปแอม (Operational Amplifier) ก็จะทำให้ได้สัญญาณที่เอาท์พุทเป็นสัญญาณโมดูเลททันที จากรูปจะเห็นว่าสัญญาณข้อมูลก็ยังมีลอจิกเป็น 1 หรือ 0 เช่นเดิม แต่เปลี่ยนรูปแบบจากระดับโวลเตจไปอยู่ในรูปของความถี่แทนคือถ้าเป็นลอจิก 1 ก็จะมีความถี่ของสัญญาณเป็น 1270 Hz และถ้าเป็นลอจิก 0 จะมีความถี่เป็น 1070 Hz

2.3.2 โมเด็มรีซีฟเวอร์ (Modem Receiver)

เมื่อสัญญาณที่ส่งออกมาจากโมเด็มทางด้านส่งผ่านขบวนการโมดูเลทแล้ว เมื่อมาถึงโมเด็มทางด้านรับสัญญาณเหล่านี้ก็จะผ่านขบวนการดีโมดูเลท แยกเอาสัญญาณพาหะออกจากสัญญาณข้อมูลแล้วผ่านสัญญาณข้อมูลไปใช้งานต่อไป



รูปที่ 2.14 โครงสร้างการทำงานของโมเด็มรีซีฟเวอร์

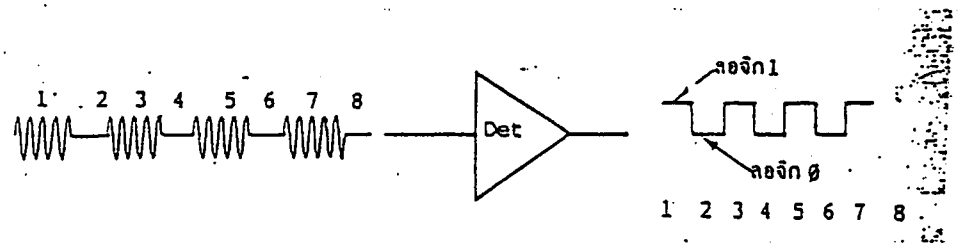
จากรูป จะแสดงถึงวิธีการในการแปลงสัญญาณอนาล็อก (ที่ส่งผ่านสายโทรศัพท์เข้ามา) กลับไปเป็นสัญญาณที่จะส่งต่อผ่านเข้าไปยังพอร์ท RS-232C ต่อไป ซึ่งแสดงถึงบล็อกไดอะแกรมของวงจรในโมเด็มทางด้านรับ จะเห็นว่าญญาณที่ส่งผ่านสายโทรศัพท์แล้วต่อเข้าที่อินพุทของโมเด็มนั้น เป็นสัญญาณที่ประกอบด้วยความถี่สองความถี่ที่ต่างกัน ดังนั้นจึงต้องทำการแยกความถี่ของสัญญาณนี้ออกจากกันก่อน โดยผ่านวงจรฟิลเตอร์ (Filter) สองวงจร ซึ่งหน้าที่ของวงจรฟิลเตอร์หรือวงจรกรองความถี่นี้ก็คือ ทำการแยกสัญญาณที่ประกอบด้วยความถี่หลักๆ ให้ออกจากกัน แล้วส่งต่อไปยังส่วนของวงจรที่ทำงานตอบสนองต่อสัญญาณความถี่นั้นเพียงค่าเดียวต่อไป

จากรูปวงจรส่วนบนประกอบด้วยวงจรฟิลเตอร์สองวงจรแยกกัน โดยทำการกรองเอาความถี่ค่าอื่นๆ ออกไป ยกเว้นสัญญาณความถี่ 2225 Hz และแถบความถี่แคบๆ ที่ใกล้เคียงกับ 2225 Hz ไว้ ซึ่งหน้าที่ของแบนด์รีเจคชั่นฟิลเตอร์ ก็คือมีลักษณะเป็นตัวต้านทานต่อสัญญาณความถี่ 2225 Hz ซึ่งหมายความว่าสัญญาณที่รับเข้ามานั้นถ้ามีความถี่เป็นค่าอื่นๆ นอกจาก 2225 Hz แล้ว ก็จะผ่านฟิลเตอร์ส่วนนี้แล้วไหลลงจุดกราวด์ไป (เอาท์พุทของแบนด์รีเจคชั่นฟิลเตอร์จะต่อลงจุดกราวด์ บางทีจึงเรียกว่า Notch Filter) ส่วนสัญญาณความถี่ 2225 Hz ซึ่งผ่านฟิลเตอร์ส่วนนี้ไม่ได้ ก็จะผ่านเข้าไปยังแบนด์พาสฟิลเตอร์(Bandpass Filter) ต่อไป ซึ่งการทำงานของแบนด์พาสฟิลเตอร์นี้ก็คือจะยอมให้สัญญาณที่มีความถี่ 2225 Hz เท่านั้นที่ผ่านไปได้ ส่วนสัญญาณความถี่อื่นก็จะถูกกันเอาไว้ ดังนั้นวงจรฟิลเตอร์ส่วนบนก็มีการงานเพื่อต้องการกรองเอาสัญญาณความถี่ 2225 Hz ไปผ่านออกที่เอาท์พุทของแบนด์พาสฟิลเตอร์นั่นเอง

ส่วนในกลุ่มวงจรฟิลเตอร์ส่วนล่างนั้น ก็มีการทำงานในลักษณะที่คล้ายคลึงกัน แต่ต่างกันที่มีการทำงานตอบสนองต่นอสัญญาณความถี่ 2025 Hz คือการทำงานผลสุดท้ายก็เพื่อต้องการกรองเอาความถี่อื่นที่ไม่ใช่ 2025 Hz ไว้แล้วปล่อยให้สัญญาณความถี่ 2025 Hz ไปผ่านออกที่เอาท์พุทของแบนด์พาสฟิลเตอร์

เมื่อได้กรองเอาความถี่ที่ต้องการคือความถี่ 2225 Hz (ซึ่งแทนลอจิก 1) และความถี่ 2025 Hz (แทนลอจิก 0) ได้แล้ว ก็จะต่อเอาท์พุทของฟิลเตอร์แบนด์พาสแต่ละตัวเข้ากับวงจรอีกส่วนหนึ่งซึ่งเรียกว่า "ดีเทคเตอร์" (Detector) หน้าที่ของวงจรดีเทคเตอร์ก็คือ การทำดีโมดูเลทสัญญาณเพื่อแยกเอาสัญญาณพาหะและสัญญาณข้อมูลออกจากกัน แล้วจึงผ่านเอาสัญญาณข้อมูลไปใช้ จากในวง

จรดังรูป เมื่อมีสัญญาณรูปไซน์ (Sine wave) ผ่านเข้าไปยังวงจรดีเทคเตอร์ก็จะสร้างระดับแรงดันเอาท์พุทของดีเทคเตอร์ให้มีค่าเป็นศูนย์นั่นคือ การทำงานดังรูป 2.15



รูปที่ 2.15 การทำงานของดีเทคเตอร์

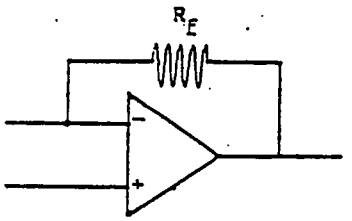
ซึ่งจะหมายความว่าที่ช่วงสัญญาณ 1,3,5 และ 7 ซึ่งมีความถี่ของสัญญาณไซน์เป็น 2225 Hz เมื่อผ่านวงจรดีเทคเตอร์ออกมา ก็จะมีระดับลอจิก 1 ส่วนในช่วงสัญญาณ 2,4,6 และ 8 เมื่อผ่านดีเทคเตอร์ออกมาจะมีระดับลอจิกเป็น 0 จากรูป จะสังเกตเห็นว่าดีเทคเตอร์แต่ละตัวมีการทำงานเพื่อสร้างสัญญาณดิจิตอล โดยมีเฟสของสัญญาณต่างกันคือ วงจรดีเทคเตอร์ตัวบนจะสร้างสัญญาณลอจิก 1 ออกมาเมื่อสัญญาณไซน์ที่อินพุทแทนสถานะ Mark หรือลอจิก 1 หรือมีความถี่ 2225 Hz แต่วงจรดีเทคเตอร์ตัวล่างจะสร้างสัญญาณลอจิก 1 ออกมาเมื่อสัญญาณไซน์ที่อินพุทแทนสถานะ Space หรือลอจิก 0 หรือความถี่ 2025 Hz เป็นต้น

ในส่วนของวงจรภาคต่อไปก็เป็นการรวมเอาสัญญาณดิจิตอลจากเอาท์พุทของดีเทคเตอร์เข้าด้วยกันโดยใช้ออฟแอมป์ (Operational Amplifier) ซึ่งหน้าที่ของออฟแอมป์หรือ op-Amp นี้ ก็คือทำการรวมสัญญาณดิจิตอลจากสองแหล่งเข้าด้วยกัน แล้วเปลี่ยนสัญญาณให้อยู่ในรูปของสัญญาณมาตรฐานที่สามารถส่งผ่านพอร์ท RS-232C (+12 V) ซึ่งวิธีการในการเปลี่ยนสัญญาณดังกล่าวนี้ก็โดยการต่อเอาท์พุทของสเปส (Space) ดีเทคเตอร์ (ดีเทคเตอร์ตัวล่าง) เข้ากับขาบวกหรือขานอนอินเวิทติ้ง

(Non inverting) ของ op-Amp และต่อเอาท์พุทของมาร์ค (Mark) ดีเทคเตอร์ (ดีเทคเตอร์ตัวบน) เข้ากับขาอินเวิทติ้ง หรือขาลบของ op-Amp ซึ่ง op-Amp สร้างสัญญาณเอาท์พุทขึ้นมาโดย

- เมื่อเอาท์พุทของสเปสดีเทคเตอร์ แอคทีฟ, op-Amp จะสร้างระดับแรงดันค่า + ขึ้นที่เอาท์พุท
- เมื่อเอาท์พุทของมาร์คดีเทคเตอร์ แอคทีฟ, op-Amp จะสร้างระดับแรงดันค่า - ขึ้นที่เอาท์พุท

นอกจากนี้อาจจะมีการใช้เทคนิคการฟีดแบค (Feedback) โดยต่อตัวต้านทานฟีดแบคเข้ากับขานอนอินเวิทติ้งของ op-Amp เพื่อทำให้การปรับเกน (Gain) ของระบบดีขึ้นและสามารถสร้างสัญญาณระดับแรงดันจากยอดถึงยอด (peak to peak) เท่ากับ 24 V (จาก -12 V ถึง +12 V) ซึ่งมีผลถึงการต่อสัญญาณข้อมูลไปใช้ส่งผ่านพอร์ท RS-232C ได้ดีขึ้น



รูปที่ 2.16 การต่อตัวต้านทานฟีดแบ็ค

ซึ่งมีผลถึงการต่อสัญญาณข้อมูลไปใช้ส่งผ่านพอร์ท RS 232 ได้ดีขึ้น

<u>โมเด็มชนิดความเร็วปานกลางและความเร็วสูง</u>

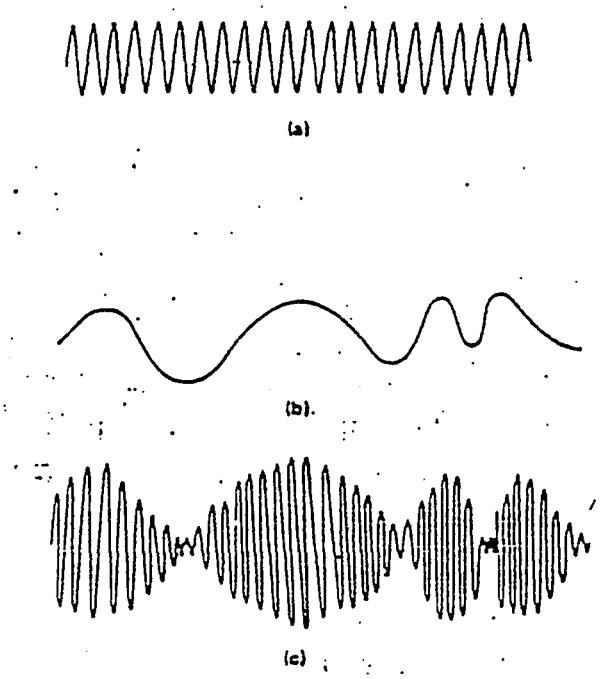
เทคนิคการโมดูเลท

เทคนิคต่างๆ ที่มีการนำมาใช้งานในอุปกรณ์โมเด็ม ก็เพื่อทำให้โมเด็มสามารถส่งข้อมูลด้วยอัตราการส่ง (บิทต่อวินาที) ที่สูงขึ้น สำหรับโมเด็มที่ใช้เทคนิคการโมดูเลทแบบ FSK นั้นจะมีอัตราการส่งข้อมูลสูงสุดเพียง 600 บิทต่อวินาทีเท่านั้น

เทคนิคการโมดูเลทที่ใช้ในโมเด็มชนิดอัตราการส่งข้อมูลปานกลางและสูงนั้น จะมีข้อยุ่งยากขึ้นมาอีกขั้นหนึ่งคือการโมดูเลทแบบ AM หรือแอมปลิจูดโมดูเลชั่น (Amplitude Modulation) และแบบ FM หรือ ฟรีเคนซี่โมดูเลชั่น (Frequency Modulation) ซึ่งเป็นเทคนิคการโมดูเลทที่นิยมใช้กันทั่วๆ ไปในการส่งกระจายคลื่นของสถานีวิทยุกระจายเสียงต่างๆ แต่การนำมาประยุกต์ใช้ในอุปกรณ์โมเด็มนั้นจะต้องใช้ร่วมกับเทคนิคการโมดูเลทแบบอื่นๆ เช่น เฟสโมดูเลชั่น(Phase Modulation) หรือ PM แต่ก่อนอื่นเราจะมาทำความเข้าใจกับวิธีการของ AM และ FM เสียก่อนก่อนที่จะกล่าวถึงรายละเอียดของ PM

แอมปลิจูดโมดูเลชั่น (AM)

เป็นเทคนิคการโมดูเลทสัญญาณเสียงเข้ากับสัญญาณพาหะ(Carrier Signal) ซึ่งเป็นสัญญาณรูปไซน์ (Sine Wave) ที่มีความถี่สูงทำให้สัญญาณที่ผ่านการโมดูเลทแล้วมีลักษณะการเปลี่ยนแปลงของสัญญาณพาหะตามแอมปลิจูด (ความสูงของสัญญาณ) ของสัญญาณเสียง แต่ความถี่ของสัญญาณพาหะยังคงที่ ดังรูป



รูปที่ 2.17 แสดงลักษณะของสัญญาณตามวิธีแอมปลิจูดโมดูเลชั่น

a) สัญญาณพาหะ

b) สัญญาณเสียงที่จะนำเข้าไปโมดูเลท

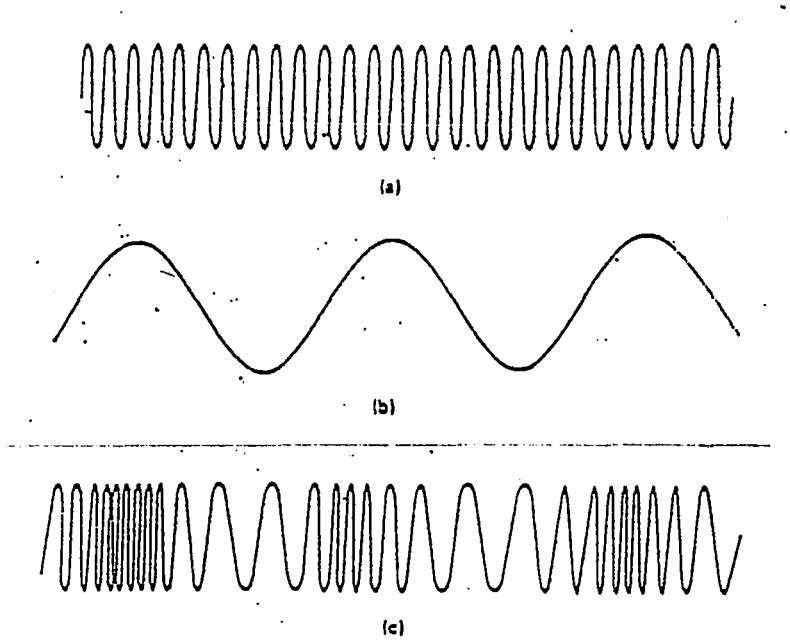
c) สัญญาณที่ผ่านการโมดูเลทแล้ว

จุดสำคัญของการโมดูเลทชนิดนี้อยู่ที่ "แอมปลิจูดเปลี่ยนแปลง แต่ความถี่คงที่"

สำหรับหลักการทำงานของโมเด็มที่ใช้เทคนิคการโมดูเลทแบบ AM นั้น ในส่วนของวงจรทางด้านส่งจะต้องทำการแปลงสัญญาณดิจิตอลไปเป็นสัญญาณอนาล็อกก่อน โดยใช้ D/A converter (Digital convert to Analog) แล้วผ่านขบวนการโมดูเลทส่งออกไป ส่วนในโมเด็มทางด้านรับเมื่อรับสัญญาณเข้ามา ก็จะผ่านขบวนการดีโมดูเลทแยกเอาสัญญาณพาหะออก แล้วจึงส่งผ่านวงจร A/D converter (Analog to Digital Converter) เพื่อแปลงสัญญาณอนาล็อกไปเป็นสัญญาณดิจิตอลแล้วส่งเข้าเทอร์มินัลใช้งานต่อไป

ฟรีเควนซีโมดูเลชั่น (Frequency Modulation : FM)

หลักการของการโมดูเลททางความถี่นี้ จุดสำคัญอยู่ที่ "แอมปลิจูดของสัญญาณจะคงที่แต่ความถี่ของสัญญาณพาหะจะเปลี่ยนแปลงไปตามแอมปลิจูดของสัญญาณที่นำเข้าโมดูเลท(Modulation Signal)" ดังรูป



รูปที่ 2.18 ฟรีเควนซีโมดูเลชั่น

a) สัญญาณพาหะ

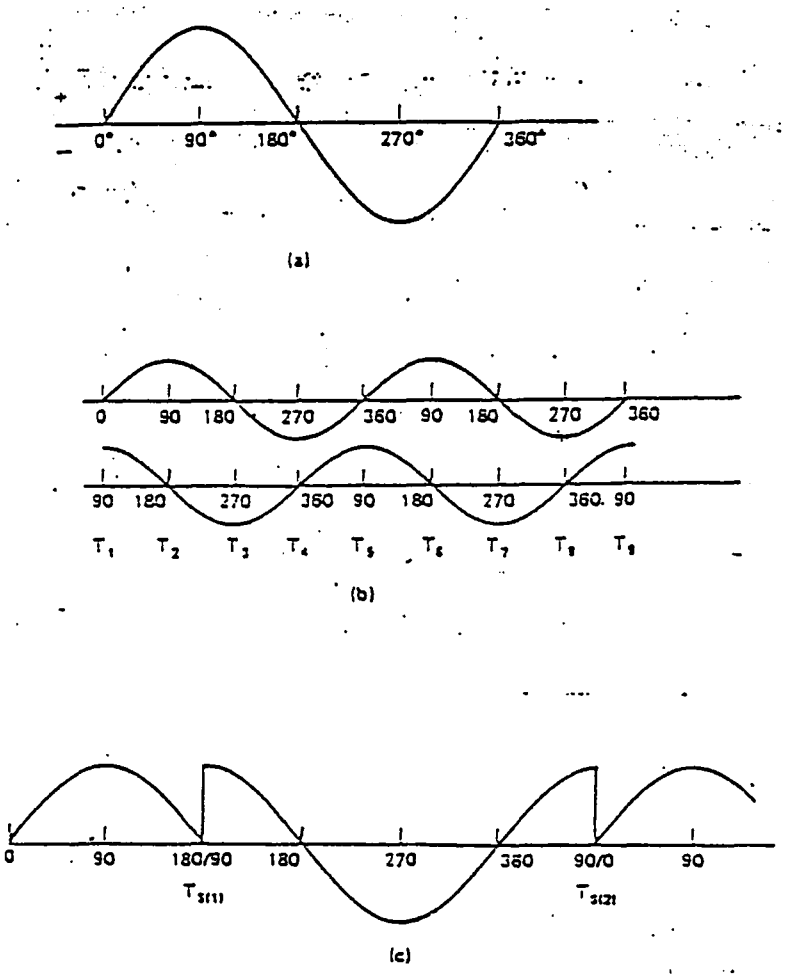
b) สัญญาณอนาล็อกที่นำเข้ามาโมดูเลท

c) สัญญาณที่ผ่านการโมดูเลท

เมื่อพิจารณารูป 2.18 (a) และ2.18 (b) ประกอบกัน ในขณะที่แอมปลิจูดของสัญญาณอนาล็อกมีค่าเป็นบวก ความถี่ของสัญญาณที่ผ่านการโมดูเลทจะสูงขึ้น แต่ในขณะที่แอมปลิจูดของสัญญาณอนาล็อกมีค่าเป็นลบ ความถี่ของสัญญาณที่ผ่านการโมดูเลทจะลดลง

เฟสโมดูเลชั่น (Phase Modulation : PM)

คำว่าเฟส (Phase) ในที่นี้หมายถึงว่า ถ้ามีสัญญาณอนาล็อก 2 สัญญาณเดินทางไป แต่แตกต่างกันด้วยค่าเวลาค่าหนึ่ง จุดๆ เดียวกันของสัญญาณคู่นี้ตลอดเวลาจะพบว่า จุดที่สังเกตบนคลื่นสัญญาณทั้งสองนี้มีมุมที่แตกต่างกันเสมอ ซึ่งมุมผลต่างนี้เราเรียกว่า "มุมเฟส" (Phase Angle)



รูปที่ 2.19 หลักการของมุมเฟสและ PM

a) สัญญาณไซน์ที่เดินทางไป 1 คาบเวลา (Period)

b) สัญญาณไซน์สองสัญญาณที่มีความต่างเฟสกัน 90 องศา

c) สัญญาณที่เฟสเลื่อนไป 90 องศา

จากรูปที่ 2.19(a) แสดงสัญญาณอะนาล็อกให้ 1 คาบเวลาโดยจะแบ่งช่วงเวลาของสัญญาณออกเป็นมุมได้ตั้งแต่ 0 ถึง360 องศา จุดเริ่มต้นของสัญญาณไซน์อยู่ที่ 90 องศา และมีแอมปลิจูดเป็น 0 และในเวลา 1/4 ของคาบเวลา แอมปลิจูดจะมีค่าสูงสุดทางบวกซึ่งตรงกับมุม 90 องศา แล้ว

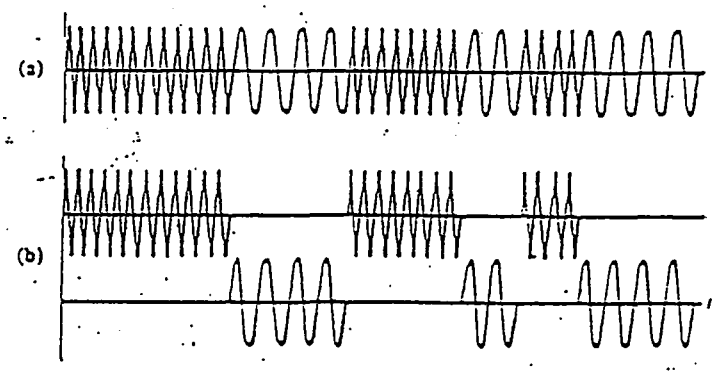
แอมปลิจูดจะลดลงเป็น 0 อีกที่มุม 180 องศา หรือ 1/2 ของคาบเวลา ต่อมาแอมปลิจูดจะมีค่าสูงสุดทางลบที่มุม 270 องศา แล้วกลับมามีแอมปลิจูดเป็น 0 อีกครั้งหนึ่งเมื่อครบ 1 คาบเวลา วนเวียนเป็นวัฏจักรอย่างนี้เรื่อยไป

สำหรับในกรณีมีการเปรียบเทียบสัญญาณไซน์ 2 สัญญาณ เราก็ใช้หลักการพิจารณาอย่างเดียวกันคือ ยึดเอาจุดใดจุดหนึ่งบนสัญญาณนั้นๆ เป็นหลักเช่นในรูปที่ 2.19 (b) จะเห็นว่าสัญญาณคู่นี้มีความต่างเฟสกันอยู่ด้วยมุมเฟสค่าหนึ่งที่เวลา T_1 สัญญาณอันล่างจะมีความต่างเฟสกับสัญญาณอันบนเท่ากับ 90 องศา ที่เวลา T_3 สัญญาณอันล่างมีเฟสอยู่ที่ 270 องศา ในขณะที่สัญญาณอันบนมีเฟสอยู่ที่ 180 องศา นั่นคือมีความต่างเฟสอยู่ 90 องศาสรุปได้ว่า สัญญาณคู่นี้มีความต่างเฟสของสัญญาณอยู่ค่าๆ หนึ่งที่คงที่คือเท่ากับ 90 องศานั้นเอง

เทคนิคปลีกย่อยของเฟสโมดูเลชั่นที่จะกล่าวถึงต่อไปนี้ คือ Phase Shift Keying (PSK) และ Phase Amplitude Modulation (PAM) รวมถึง Frequency Shift Keying (FSK) ซึ่งมีรายละเอียดดังต่อไปนี้

FREQUENCY-SHIFT KEYING (FSK)

หลักการคือ ความถี่ของสัญญาณพาหะจะแปรเปลี่ยนตามสัญญาณดิจิตอลที่เข้ามาคือ ค่าสัญญาณมีระดับลอจิกเป็น 1 ความถี่ของสัญญาณพาหะจะสูงขึ้น แต่ค่าสัญญาณมีระดับลอจิกเป็น0 ความถี่ของสัญญาณพาหะจะลดลงดังรูป โดยเราสามารถแยกข้อแตกต่างของ FSK จาก FM ได้ว่าในขบวน FSK คลื่นพาหะอาจจะมีความถี่ของคลื่นได้มากกว่า 2 ความถี่ แต่ใน FM จะมีความถี่ของคลื่นพาหะได้เพียง 1 ความถี่เท่านั้น



รูปที่ 2.20 แสดงสัญญาณที่ได้จากเทคนิค FSK

จากรูป จะเห็นว่าเราสามารถจะแยกสัญญาณ FSK ออกเป็นสัญญาณ Amplitude Shift Keying (ASK) ได้สองสัญญาณ

PHASE-SHIFT KEYING (PSK)

หลักการคือ เฟสของสัญญาณพาหะจะเปลี่ยนไปตามสัญญาณดิจิตอลที่เข้ามาคือ ถ้าระดับลอจิกของสัญญาณเปลี่ยนจาก 0 เป็น 1 เฟสของสัญญาณก็จะเปลี่ยนไป 90 องศา

จากรูป 2.19 c เฟสของสัญญาณหลังจาก 180 องศา แล้วควรจะเป็น 270 องศา แต่สัญญาณจะกลับไปเป็น 90 องศา เช่น ที่เวลา $T_s(1)$ และ ที่เวลา $T_s(2)$ เฟสของสัญญาณในช่วงต่อไปควรจะเป็น 180 องศา แต่ปรากฏว่าเฟสของสัญญาณกลับมีค่าเป็น 0 องศา เพราะฉะนั้น สรุปได้ว่าการเลื่อนเฟสจะมีค่าเป็น 90 องศา นั่นเอง

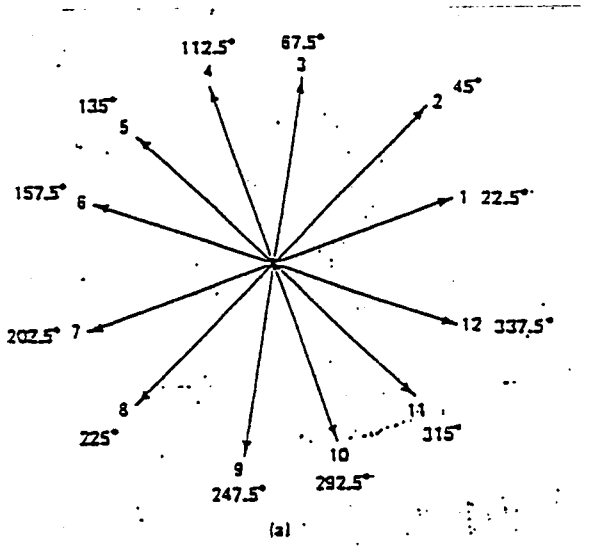
ในลักษณะเช่นนี้ถ้าเราสามารถเลื่อนเฟสของสัญญาณไซน์ออกเป็นค่าต่างๆ เช่น อาจจะต่างกัน 4 ค่าประสิทธิภาพของโมเด็มย่อมจะสูงขึ้น ซึ่งในโมเด็มชนิดที่ใช้เทคนิค PSK ก็ใช้หลักการนี้เช่นกัน เช่น ในโมเด็ม Bell 201 B มีการเลื่อนสัญญาณไปเป็น 45,135,225 และ 315 องศา ซึ่งค่าของเฟสที่เลื่อนไปจะสัมพันธ์กับการวนรอบของเฟสอื่นๆ ด้วย มุมเฟสต่างๆ กันทั้ง 4 ค่านี้จะใช้แทนบิทข้อมูลได้ 4 บิท (แทนที่จะเป็น 1บิท เช่นใน FSK) คือ เป็นค่า 11,01,10 และ 11 ตามลำดับ ซึ่งในการเรียงบิทในลักษณะนี้เรียกว่า dibits ซึ่งมีผลดีคือ เป็นเทคนิคที่ทำให้สามารถเพิ่มอัตราโบดได้สูงยิ่งขึ้น เช่น ถ้าโมเด็มสามารถส่งข้อมูลในอัตรา 600 dibits ต่อ วินาที (600 โบด) ก็จะมีค่าเท่ากับโมเด็มส่งข้อมูลด้วยอัตรา 1200 บิท/วินาที ในปัจจุบันขีดความสามารถของโมเด็มได้ถูกพัฒนาขึ้นอย่างมาก เช่น โมเด็มรุ่น Bell 201 C มีความสามารถในการเลื่อนมุมเฟสได้ต่างๆ กันถึง 8 ค่า ซึ่งในมุมเฟสค่าหนึ่งใช้แทนบิทข้อมูลได้ 3 บิท ในกรณีนี้อัตราการส่งข้อมูลยิ่งสูงขึ้นอย่างมาก สามารถส่งข้อมูลได้ถึง 4800 บิท/วินาที เช่น ในกรณีที่มีอัตราของสัญญาณในสายส่ง(Line Signaling Rate)มีค่าเป็น 1600 โบด โมเด็มจะมีอัตราการส่งข้อมูลได้ถึง 4800 บิท/วินาที ซึ่งผู้อ่านจะสับสนระหว่างอัตราบิทต่อวินาที , อัตราโบด และอัตราสัญญาณในสายส่ง ดังนั้นจึงได้แสดงรายละเอียดได้ในตารางต่อไปนี้

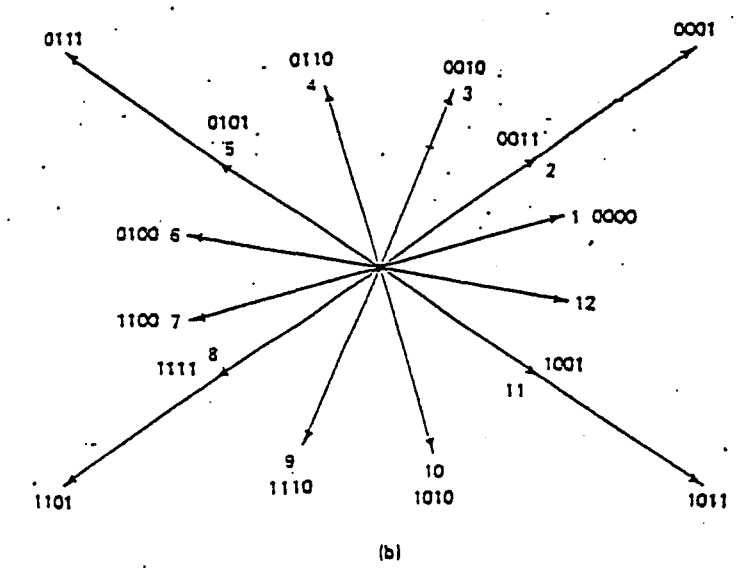
Line Signaling		
Modulation technique	Rate, bauds	Bit rate, bps
FSK	n(300)	n(300)
Dibits	n(1200)	2n(2400)
Tribits	n(1200)	8n(9600)
Quadbits	n(1200)	16n(19,200)

ตารางที่ 2.3

PHASE AMPLITUDE MODULATION (PAM)

เป็นเทคนิคที่เกิดจากการรวมเอาสัญญาณโมดูเลทที่เกิดจากเทคนิค PSK และ AM เข้าด้วยกัน ทำให้ได้อัตราการส่งสูงขึ้น คือ สามารถส่งบิทข้อมูลได้ถึง 16 แบบต่าง ๆ กัน ดังรูป





รูปที่ 2.21 PAM

(a) มุมเฟสต่าง ๆ กัน 12 มุม

(b) การเข้ารหัสขนาด 4 บิท

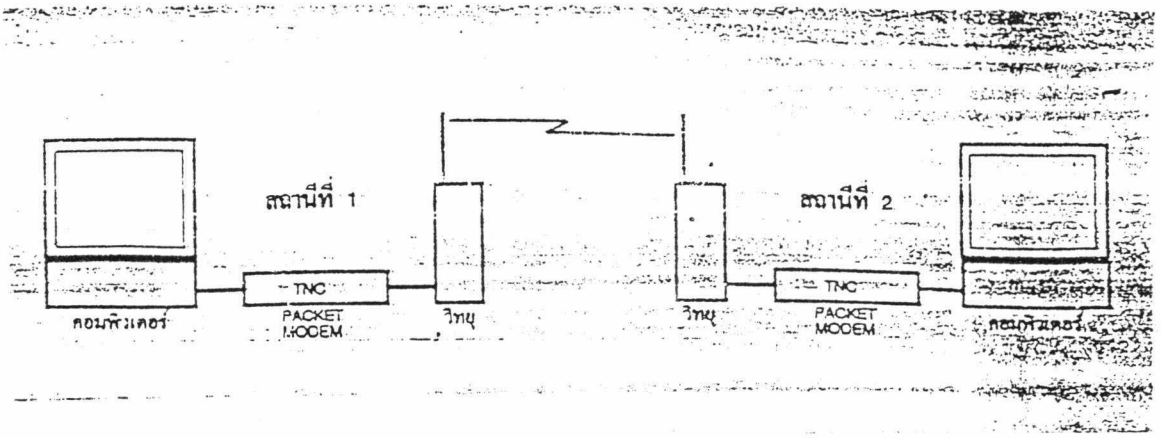
จากรูป 2.21a จะเห็นว่ามีการแบ่งมุมเฟสออกเป็น 12 มุม สำหรับใช้ในการโมดูเลทแบบ PAM และถ้าเราเพิ่มขนาดของแอมปลิจูดให้แก่มุมเฟสทั้ง 4 มุม จากทั้งหมด ทำให้มุมเฟส 4 มุมที่เราเพิ่มแอมปลิจูดเข้าไปนั้นมีแอมปลิจูดถึง 2 ค่า ดังนั้นเกิดมุมเฟสขึ้นอีก 4 ค่า รวมเป็น 16 ค่า ดังรูป 2.21 b ซึ่งมุมเฟสแต่ละค่า ก็ถูกใช้ในการแทนข้อมูลกลุ่มหนึ่ง เพราะฉะนั้นจึงสามารถแทนกลุ่มข้อมูลได้ทั้งหมด 16 แบบต่างๆ กัน ดังนั้นเทคนิควิธีนี้จึงทำให้อัตราการส่งข้อมูลขึ้นถึง 9600 บิทต่อวินาทีส่วนใหญ่โมเด็มประเภทนี้ จะใช้กับการส่งข้อมูลแบบซิงโครนัส

2.4 แพคเกต เรดิโอ

แพคเกต เรดิโอ (packet radio) คือ การรับ/ส่งข่าวสารหรือข้อมูลผ่านวิทยุสื่อสาร โดยการนำข่าวสารข้อมูลที่จะส่งนั้นมาตัดเป็นส่วนย่อยๆแล้วทยอยส่งไปยังผู้รับ โดยเพิ่มเติมข้อมูลบอกว่าข่าวสารนั้นจะส่งไปให้ใครเป็นข้อมูลส่วนใดในจำนวนข้อมูลทั้งหมด รวมทั้งมีการตรวจสอบความถูกต้องของข้อมูลที่รับ/ส่งนั้นอยู่ตลอดเวลา

แพคเกต เรดิโอ ประกอบด้วยอุปกรณ์อะไรบ้าง ?

ในระบบการติดต่อรับ/ส่งข่าวสารด้วยแพคเกต เรดิโอ นั้น ส่วนประกอบที่สำคัญๆ มีดังแผนภาพรูปที่ 2.22 ซึ่งอธิบายได้ดังนี้



รูปที่ 2.22 แสดงแผนภาพการติดต่อรับ/ส่งข่าวสารระหว่างเครื่องคอมพิวเตอร์สองเครื่อง

จากแผนภาพรูปที่ 2.22 มีจุดประสงค์หลัก คือ การติดต่อรับ/ส่งข่าวสารหรือข้อมูลระหว่างเครื่องคอมพิวเตอร์สองเครื่อง ถ้าคุ้นเคยกับการใช้เครื่องคอมพิวเตอร์ส่วนบุคคล (PERSONAL COMPUTER หรือที่เรียกย่อๆ ว่า PC) บ้างแล้วจะพบว่า หากต่อสายเครื่องคอมพิวเตอร์สองเครื่องในระยะใกล้ๆ (ไม่เกิน20ฟุต)เครื่องคอมพิวเตอร์ทั้งสองเครื่องจะถ่ายเทข่าวสารข้อมูลระหว่างกันได้ (DATA TRANSFERING) ทั้งนี้โดยต้องอาศัยโปรแกรมการสื่อสารระหว่างคอมพิวเตอร์ (COMMUNICATION

PROGRAM) เช่น LAPLINK , PCPLUS , TELIX หรือ CROSSTALK เป็นต้น เมื่อต้องการติดต่อรับ/ส่งข่าวสารข้อมูลระหว่างคอมพิวเตอร์สองเครื่องในระยะไกลออกไปกว่านี้ จะต้องอาศัยอุปกรณ์อีกชนิดหนึ่งเพื่อช่วยเพิ่มระยะการติดต่อดังกล่าว อุปกรณ์นั่นคือ MODEM

MODEM

คำว่า MODEM เป็นคำย่อเพื่อสะดวกในการใช้งาน โดยย่อมาจากคำว่า MODULATION AND DEMODULATION ซึ่งลักษณะการทำงานของ MODEM นั้น จะเป็นไปตามชื่อของมัน โดยการนำข่าวสารจากภายนอกที่ต้องการส่ง (ซึ่งจะต้องเป็นสัญญาณไฟฟ้าที่เรียกว่าSERIAL DATA)ไปเปลี่ยนแปลงและผสม (MODULATE) กับคลื่นเสียง แล้วส่งไปยังตัวกลางส่งข้อมูล (TRANSMISSION MEDIA) ซึ่งอาจเป็นสายโทรศัพท์ , วิทยุสื่อสารหรือแม้กระทั่งเส้นใยแสง (OPTICAL FIBER) ก็ได้ ลักษณะนี้จะเป็นการส่งข่าวสารข้อมูล (DATA TRANSMISSION)

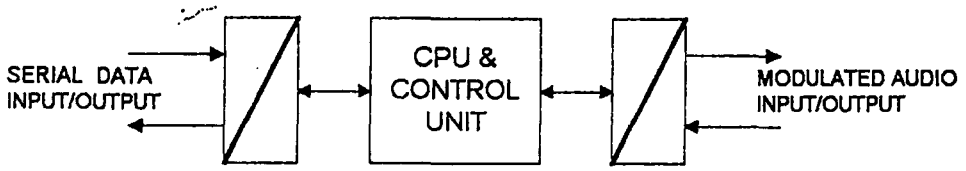
ในส่วนที่ใช้ MODEM เป็นตัวรับสัญญาณ ข่าวสารข้อมูล (DATA RECEIVING) จะใช้ MODEM อีกตัวหนึ่งรับสัญญาณเสียงที่ถูกผสมมาด้วยข่าวสารนั้น แล้วแปลง(DEMODULATE)ข่าวสารออกมาเป็นสัญญาณไฟฟ้า (SERIAL DATA)

จะเห็นได้ว่า MODEM ช่วยยืดระยะการส่งข่าวสารออกไปได้ดีมาก ทั้งนี้ขึ้นอยู่กับชนิดของ MODEM ที่ใช้ (อาจเป็น MODEM ชนิดใช้งานระยะใกล้ MODEM ชนิดใช้งานระยะไกลหรือชนิดอื่นๆ) เพราะสัญญาณไฟฟ้าที่เรียกว่า SERIAL DATA นั้น จะมีความเร็วสูงมาก และมีการเปลี่ยนแปลงลักษณะสัญญาณอย่างรวดเร็ว หากส่งผ่านสายไฟฟ้าหรือสายโทรศัพท์ โดยตรงไปในระยะไกลๆจะมีการสูญเสียและผิดเพี้ยนมาก

PACKET MODEM ที่ใช้ในระบบ PACKET RADIO จัดเป็น MODEM ชนิดหนึ่ง ที่จะถูกนำมาใช้เป็นตัวกลางทำหน้าที่แปลงข่าวสารข้อมูลจากคอมพิวเตอร์ให้เหมาะสมที่จะส่งผ่านไปยังคอมพิวเตอร์อีกเครื่องหนึ่ง โดยอาศัยคลื่นวิทยุจากเครื่องรับ/ส่งวิทยุเป็นตัวกลาง

โครงสร้างของ PACKET MODEM

PACKET MODEM จัดเป็น MODEM ชนิดหนึ่ง ซึ่งใช้วงจรอิเล็กทรอนิคส์ภายในจัดการกับข้อมูลที่ส่งผ่านเข้าและรับผ่านออก ในลักษณะการจัดการข้อมูลที่เรียกว่าเป็น PACKET DATA จากรูปที่ 2.23 จะเห็นว่าตัว CPU & CONTROL UNIT จะจัดการกับ SERIAL DATA INPUT ให้กลายเป็น MODULATED AUDIO OUTPUT และจัดการกับ MODULATED AUDIO INPUT ให้เป็น SERIAL DATA OUTPUT โดยอาศัย PROGRAM ที่ผู้สร้าง PACKET MODEM ได้บรรจุไว้ภายในตัวอุปกรณ์ ซึ่งในปัจจุบันนี้สามารถผลิตให้อยู่ในรูปของชิ้น IC (INTEGRATED CIRCUIT) เพียงชิ้นเดียว (SINGLE CHIP CENTRAL PROCESSING UNIT)



รูปที่ 2.23 แผนภาพแสดงโครงสร้างของ PACKET MODEM

การใช้งาน PACKET RADIO

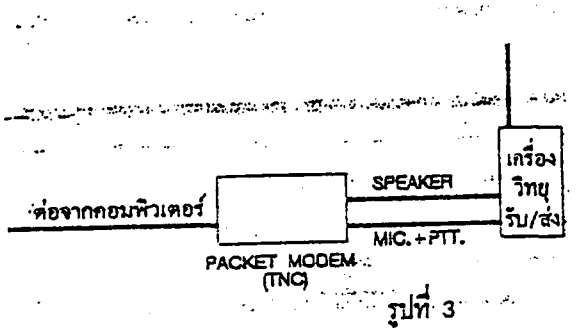
เนื่องจาก PACKET RADIO จะต้องประกอบด้วยเครื่องคอมพิวเตอร์ , PACKET MODEM และเครื่องรับ/ส่งวิทยุพร้อมสายอากาศ จึงควรทราบลักษณะการต่อเชื่อมระหว่างอุปกรณ์ทั้งสามชนิดดังนี้

1.เครื่องคอมพิวเตอร์ควรจะเป็น PC COMPATIBLE คือมีลักษณะการทำงาน (FUNCTIONING)เสมือนเครื่อง IBM PC ประกอบด้วยจอภาพ , ตัวเครื่องและแป้นอักษร เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบควรจะมี SERIAL COMMUNICATION PORT (โดยเฉพาะCOM1) ซึ่งตามปกติเครื่องคอมพิวเตอร์ PC มักจะมีอยู่แล้ว

จากเครื่องคอมพิวเตอร์จะต้องต่อสายสัญญาณข้อมูลเชื่อมกับ PACKET MODEM ซึ่งบนตัว PACKET MODEM ก็จะมี CONNECTOR รอรับอยู่เป็น D-SUB 9-PIN CONNECTOR ซึ่งเป็น CONNECTOR มาตรฐานในงานรับ/ส่งข่าวสารข้อมูลแบบ SERIAL เมื่อใช้ SOFTWARE ที่เหมาะสม จะทำการติดต่อ ควบคุมแลกเปลี่ยนข่าวสารข้อมูลระหว่างคอมพิวเตอร์และ PACKET MODEM ได้ที่ความเร็วต่างๆ กัน

2. PACKET MODEM หัวใจสำคัญของระบบ PACKET RADIO คือ PACKET MODEM ซึ่งมีอยู่หลายชนิดหลายยี่ห้อ และต่างมีความสามารถในการทำงานไม่เหมือนกัน ซึ่งเมื่อเชื่อมต่อกับคอมพิวเตอร์แล้วจะพร้อมที่จะทำงานเปลี่ยนข่าวสารข้อมูลจากคอมพิวเตอร์ไปส่งต่อให้เครื่องวิทยุอีกทีหนึ่ง และรับสัญญาณจากเครื่องวิทยุมาเปลี่ยนกลับเป็นข่าวสารข้อมูลส่งกลับให้เครื่องคอมพิวเตอร์ต่อไป ทั้งนี้โดยการจัดการของ SOFTWARE ในเครื่องคอมพิวเตอร์

ส่วนสำคัญส่วนหนึ่งของตัว PACKET MODEM คือ ส่วนที่ต่อระหว่าง PACKET MODEMกับเครื่องวิทยุ ซึ่งเป็นการต่อสัญญาณเสียงที่ผสมกับข่าวสารแล้ว ตามแผนภูมิรูปที่ 2.24



รูปที่ 3

รูปที่ 2.24 แสดงแผนภาพจุดต่อระหว่างแพคเกตโมเด็มกับเครื่องวิทยุรับ/ส่ง

จากแผนภาพที่ 2.24 จุดต่อระหว่าง PACKET MODEM กับเครื่องวิทยุรับ/ส่งนั้น เป็นการต่อสัญญาณเสียงระหว่างกัน ซึ่งการผสมข่าวสารกับเสียงนี้ เป็นลักษณะระบบที่เรียกว่า FREQUENCY SHIFT KEYING (FSK) หากลองรับฟังเสียงในขณะที่ PACKET MODEM ทำงานรับ/ส่งข้อมูลจะได้ยินเสียงเป็นจังหวะสั้นยาว คล้ายๆกับสัญญาณของเครื่องโทรสาร (FACISIMILE) ที่ได้ยินทางเครื่องโทรศัพท์

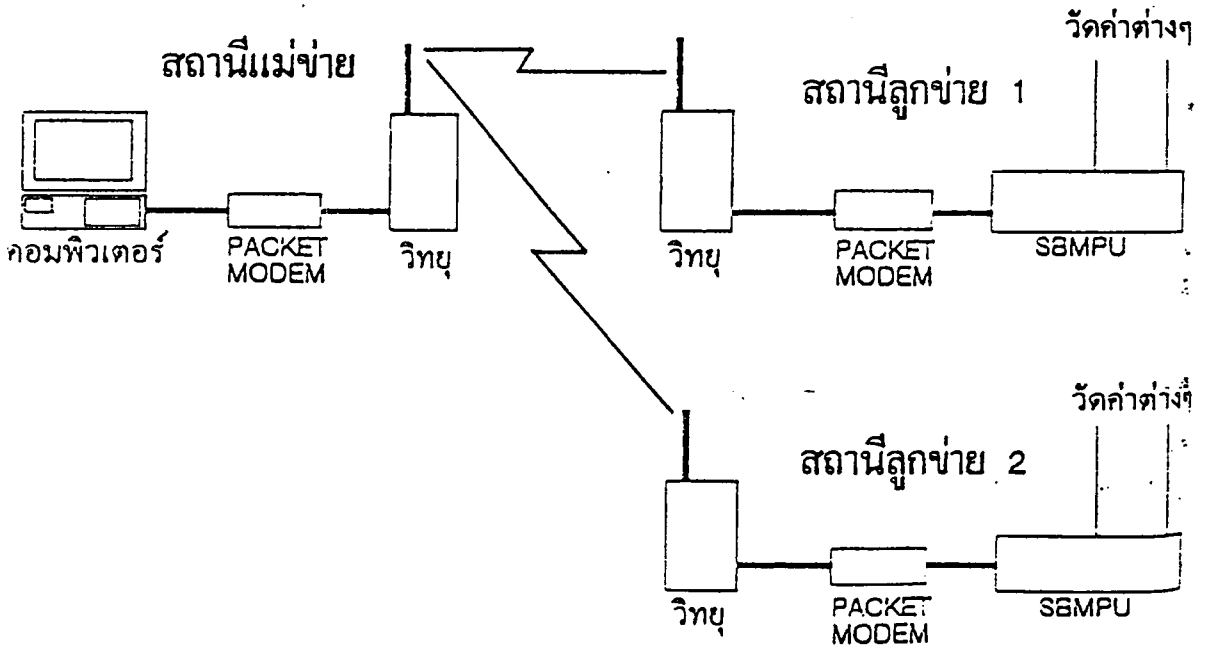
3. เครื่องวิทยุและสายอากาศของ PACKET RADIO นั้นแต่ละสถานีจะติดต่อถึงกันได้ต้องสามารถรับ/ส่งสัญญาณกันได้ดีพอสมควร ซึ่งจะดูได้จากความแรงของสัญญาณที่รับได้จากคู่สถานี และความผิดพลาดของข้อมูลที่ได้ออกมาจากคอมพิวเตอร์

สิ่งที่น่าสังเกตเกี่ยวกับเครื่องวิทยุในระบบ PACKET RADIO คือ การที่ PACKET MODEM จะต่อเชื่อมกับวิทยุแต่ละยี่ห้อนั้นจะไม่เหมือนกัน กล่าวคือ ในวิทยุแต่ละยี่ห้อ แต่ละรุ่นจะมีค่า IMPEDANCE ของ TRANSMITTING KEY ไม่เท่ากัน จึงต้องมีการปรับเปลี่ยนบางส่วนของระบบ ในการต่อเชื่อมระหว่าง PACKET MODEM กับเครื่องวิทยุบ้าง มิฉะนั้นเครื่องวิทยุจะไม่ทำการส่งสัญญาณตามที่ PACKET MODEM สั่งการมา แต่ทั้งนี้ รายละเอียดบอกวิธีการปรับเปลี่ยนเหล่านี้อยู่ในคู่มือของวิทยุแต่ละรุ่นอยู่แล้ว

<u>การใช้ PACKET RADIO กับระบบรายงานข้อมูลอัตโนมัติ</u>

ในระบบการรายงานข้อมูล (DATA ACQUISITION) จากเครื่องวัดต่างๆ สิ่งสำคัญ คือ ข้อมูลต้องไม่ผิดพลาดและจะต้องสามารถส่งข้อมูลได้ต่อเนื่องตลอดเวลา การรับส่งข้อมูลแต่ละครั้งจะต้องใช้เวลาน้อยที่สุด เพื่อประหยัดเวลาของตัวกลางที่ใช้ส่งข้อมูล (TRANSMISSION MEDIUM) ซึ่งจากข้อจำกัดเหล่านี้ ได้นำมาพิจารณาจัดระบบ PACKET RADIO ให้ทำงานสนองตอบวัตถุประสงค์อย่างได้ผลดียิ่ง โดยการประยุกต์ใช้ PACKET MODEM ประกอบกับระบบวิทยุสื่อสารชนิด SIMPLEX SINGLE FREQUENCY ใช้เครื่องคอมพิวเตอร์ขนาดเล็ก (IBM PC COMPATIBLE) ก็เพียงพอที่จะใช้งานรับ/ส่งข้อมูลได้อย่างมีประสิทธิภาพ โดยใช้ทรัพยากรด้านความถี่วิทยุเพียงความถี่เดียวสำหรับทั้งระบบ

หลักการของระบบรายงานข้อมูลอัตโนมัตินั้นจะมีศูนย์กลางที่เป็นการทำงานโดย PROGRAM ที่อยู่บนเครื่องคอมพิวเตอร์ต่อเชื่อมกับ PACKET MODEM และเครื่องวิทยุ ทำการเรียกถามสถานีต่างๆที่อยู่โดยรอบ (สถานีลูกข่าย) ให้ส่งข้อมูลที่ต้องการทีละสถานี มาให้สถานีที่ศูนย์กลางได้ทราบการกำหนด PROGRAM ให้สถานีลูกข่ายตอบรับอย่างไร เมื่อมีการทำงานสนองตอบอย่างไรนั้น สามารถเขียนเป็น PROGRAM สั้น ๆ ติดตั้งไว้บน SINGLE BOARD MICROPROCESSOR UNIT ให้ทำงานแทนเครื่องคอมพิวเตอร์ขนาดใหญ่ได้ดังแผนภาพรูปที่ 2.25



รูปที่ 2.25 แผนภาพแสดงหลักการของระบบรายงานข้อมูลอัตโนมัติ

2.5 ระบบสายอากาศแบบคอลลิเนียร์ (collinear หรือ colinear)

แบบคอลลิเนียร์ คือ แบบที่นำเอาสายอากาศพื้นฐานตั้งแต่ 2 ตัวขึ้นไปมาต่อพ่วงกันให้มีอัตราขยายสูงขึ้น โดยให้สายอากาศทั้งหมดอยู่ในแนวดิ่งเดียวกัน สายอากาศพื้นฐานที่ว่านี้อาจได้แก่ สายอากาศพวกกราวด์เพลน, ไดโพล, $^5/_8\lambda$ เป็นต้น

สายอากาศคอลลิเนียร์แบ่งเป็น 2 แบบย่อย ตามลักษณะการต่อพ่วงกัน แบบแรก คือ **คอลลิเนียร์แบบต่ออนุกรม** (series fed collinear) ซึ่งเป็นการนำเอาสายอากาศพื้นฐานมาต่ออนุกรมกันแบบนี้มีปัญหาตรงที่มีผลของสายนำสัญญาณเข้ามารบกวนการทำงานของสายอากาศทำให้มุมยิงสูงการปรับแต่งก็ยาก และไม่สามารถคำนวณหรือคาดเดาได้ล่วงหน้าว่าอัตราขยายที่เพิ่มขึ้นเมื่อนำสายอากาศมาต่อเพิ่มขึ้นนั้นจะเป็นเท่าไรต้องอาศัยการทดลองเป็นหลัก และส่วนใหญ่แล้วอัตราขยายที่ได้มักจะไม่ค่อยคุ้มกับความสูงที่ต้องเพิ่มขึ้น ตัวอย่างสายอากาศคอลลิเนียร์แบบต่ออนุกรมได้แก่ สายอากาศรอบตัว $^5/_8\lambda$ 2 ชั้น และ $^5/_8\lambda$ 3 ชั้น ซึ่งการจะต่อจำนวนสายอากาศพื้นฐาน $^5/_8\lambda$ เข้าไปอีกเป็น 4 ชั้นหรือมากกว่านั้น จะทำให้ได้อัตราขยายเพิ่มขึ้นน้อยมากจนไม่น่าสนใจที่จะทำและการสร้างก็วุ่นวาย เพราะความยาวของแต่ละชั้นที่เพิ่มขึ้นก็แตกต่างไปจากความยาวของสายอากาศ $^5/_8$ ชั้นเดียว จึงต้องอาศัยการทดลองหาความยาวที่เหมาะสมอีก

แบบที่สองคือ**คอลลิเนียร์แบบต่อขนาน** (parallel fed collinear) เป็นการนำเอาสายอากาศพื้นฐานมาต่อขนานกันหรือที่นิยมเรียกว่านำมาสแต็ค (stack) กันแบบนี้จะลดผลของสายนำสัญญาณที่มีต่อการทำงานของสายอากาศไปได้มาก ปรับแต่งได้ง่ายและสามารถเดาอัตราขยายที่ได้ค่อนข้างแม่นยำ เช่น เมื่อนำสายอากาศพื้นฐาน 2 ต้นมาสแต็คกัน เป็นแบบที่นิยมเรียกว่า 2 สแต็คก็จะได้อัตราขยายกำลังไฟฟ้ารวมเป็นประมาณ 2 เท่า ของต้นเดียว (หรือได้อัตราขยายเพิ่มจากเดิมอีกประมาณ 3 dB) ถ้านำมาทำเป็น 3 สแต็ค ก็จะได้อัตราขยายกำลังไฟฟ้ารวมเป็นประมาณ 3 เท่าของต้นเดียว (หรือเท่ากับอัตราขยายเพิ่มขึ้นจากเดิมอีกประมาณ 4.8 dB) อย่างนี้เป็นต้น และที่สำคัญอีกอย่างหนึ่ง คือ ไม่ว่าจะต่อกี่สแต็คก็ตาม ขนาดและรูปร่างของสายอากาศพื้นฐานยังคงเป็นเช่นเดิม

ทำให้ง่ายแก่การทดลองสร้างขึ้นใช้เอง ตัวอย่างสายอากาศคอลลิเนียร์แบบขนานที่เราคุ้นเคยกันได้แก่ สายอากาศเจโพลที่นิยมนำมา สแต็คกัน 2,4 และ 8 สแต็คอย่างที่เห็นหน่วยราชการใช้กันโดยทั่วไป

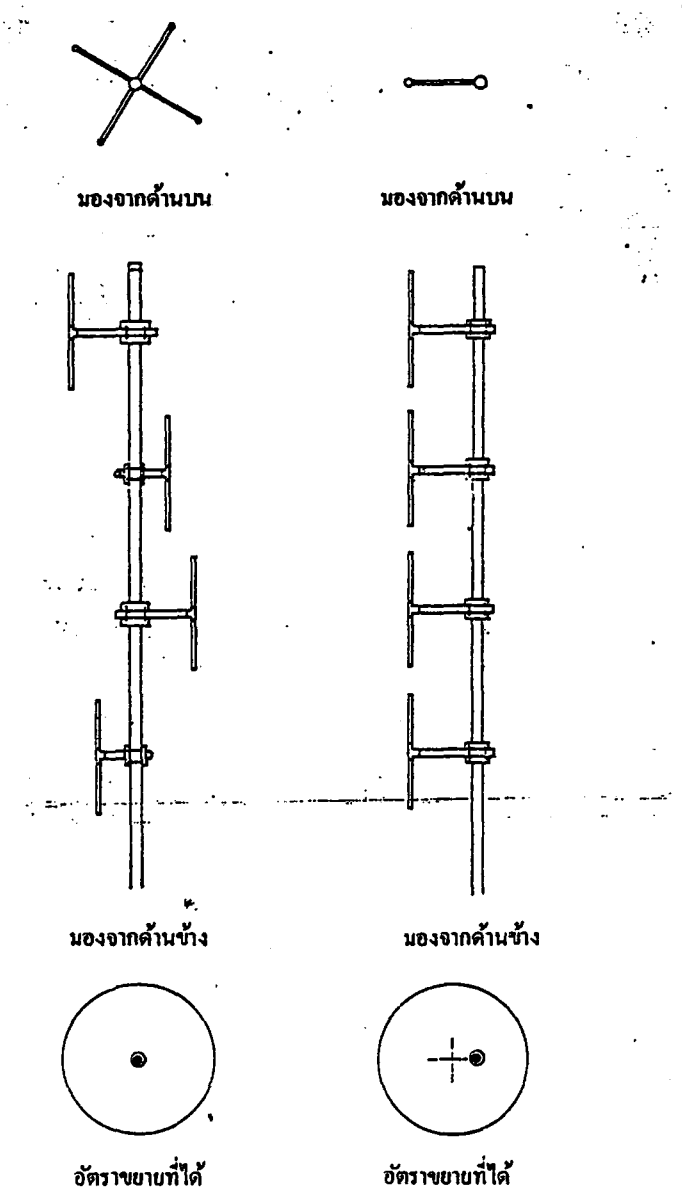
หลังจากพิจารณาข้อดีข้อเสียเช่นนี้ ก็คงเห็นแล้วว่าสายอากาศคอลลิเนียร์แบบขนานนั้นน่าสนใจกว่าแบบอนุกรมและเมื่อพิจารณาจากแบบต่างๆเพิ่มเติมอีก คิดว่าการนำเอาสายอากาศไดโพลมาสร้างเป็นสายอากาศคอลลิเนียร์แบบต่อขนานจะเหมาะสมกว่า

ในแง่ที่ว่าสามารถใช้การคำนวณง่ายๆมาหาระยะต่างๆได้การสร้างง่ายและเมื่อสร้างเสร็จแล้วก็แทบจะไม่ต้องปรับแต่งอีกเลย แม้ว่าแถบความถี่ใช้งาน (bandwidth) จะแคบกว่าการใช้สายอากาศโฟลเดดไดโพลอยู่บ้างก็ตาม แต่ก็ไม่มีปัญหาอะไรในแง่ปฏิบัติ เพราะเรามุ่งเจาะจงใช้งานในย่านความถี่แคบๆ คือระหว่าง 144 ถึง 146 MHz เท่านั้น แม้กระนั้นก็ได้แถบความถี่กว้างไม่น้อยกว่า 8 MHz (คิดที่ VSWR 1.5:1 เป็นเกณฑ์) เมื่อใช้ท่ออะลูมิเนียมขนาดเส้นผ่านศูนย์กลางไม่น้อยกว่า 1 นิ้วมาทำเป็นตัวไดโพล ซึ่งก็นับว่าแถบความถี่กว้างเกินพออยู่แล้ว

ในรูปที่ 2.26 แสดงตัวอย่างวิธีการจัดวางไดโพล 2 แบบที่นิยมกัน แบบแรกเป็นการจัดวางไดโพลให้หันหน้าออกไปรอบตัวจริงๆ อย่างในรูป 2.26 (ก) เป็นแบบ 4 สแต็ค ก็แบ่งมุม (เมื่อมองดูจากด้านบน) 360 องศา ออกเป็น 4 ส่วนเท่าๆกัน ดังนั้นไดโพลจะหันหน้าทำมุมกัน 90 องศา เรียงกันไปรอบเสากลาง วิธีนี้ในตำราทั่วๆ ไปจะระบุว่าให้อัตราขยายในทุกทิศทุกทางเท่ากันประมาณ 6 dB_d อย่างไรก็ตามในทางปฏิบัติจริงๆแล้ว การติดตั้งวิธีนี้จะทำให้สัญญาณที่ไดโพลแต่ละตัวรับได้มีขนาดไม่เท่ากันและมีเฟสไม่ตรงกันทีเดียว จึงมีการหักล้างหรือเสริมกัน แล้วแต่ระยะห่างและทิศของคู่สถานี รูปแบบการกระจาย/รับคลื่นของการจัดวางแบบนี้จึงไม่เป็นวงกลมทีเดียวนัก บางมุมอาจบอดไป (เช่น อัตราขยายบางทิศอาจน้อยกว่าทิศอื่นถึง 6 dB!)

แบบที่สองเป็นแบบที่วางไดโพลอยู่ในแนวดิ่งเดียวกันทั้งหมด การติดตั้งวิธีนี้ ทำให้อัตราขยายด้านหน้า (ด้านที่ไดโพลหันหน้าไปหา)มากกว่าด้านหลังประมาณ 3 dB แต่รูปแบบการกระจาย/รับคลื่นในแนวราบก็ยังคงเป็นวงกลมอยู่ เพียงแต่จุดศูนย์กลางเหลื่อม (offset) ไปทางด้านหน้าดังแสดงในรูป 2.26 (ข) ทำให้อัตราขยายด้านหน้าเพิ่มเป็นประมาณ 9 dB_d แต่ด้านหลังก็ลดลงเล็กน้อย วิธีนี้ทำให้คู่สถานีอยู่ห่างจากไดโพลแต่ละตัวเท่ากัน เฟสของแต่ละสัญญาณจึงตรงกัน ทำให้ไม่มีจุดบอดในบางทิศทาง และก็ได้ข้อเด่นที่ประโยชน์ต่อการใช้งานบางอย่างที่ทราบทิศทางของคู่สถานีอย่างแน่นอน คือ ได้อัตราขยายด้านหน้าเพิ่มขึ้นกว่าแบบ (ก) เกือบ 3 dB ดังนั้น เวลาติดตั้งก็เพียงแต่หันสาย

อากาศไปหาคู่สถานีที่มีสัญญาณอ่อนมาก การติดตั้งวิธีนี้จึงได้รับความนิยมมากและก็เป็นแบบที่เราจะเลือกมาใช้ ณ ที่นี้



(ก) แบบวางรอบตัว (ข) แบบวางให้อยู่แนวเดียวกันทั้งหมด

รูปที่ 2.26 แสดงวิธีการจัดวางไดโพล 2 แบบที่นิยมกันสำหรับ 4 สแต็ค

สำหรับแบบสายอากาศที่นำมาในที่นี้ ดัดแปลงมาจากแบบที่นายเซอร์แมน ซี.คาร์ W9NGT เผยแพร่ลงในวารสาร CQ ของสหรัฐอเมริกา ฉบับประจำเดือนมีนาคม พ.ศ.2514 แบบที่ว่านี้เป็นการนำเอาสายอากาศไดโพลมาสร้างเป็นสายอากาศรอบตัวคอลลิเนียร์แบบต่อขนาน โดยจะสามารถเลือกสแต็คว่าจะทำเป็น 2,3,4,5,6,7 และ 8 ให้เหมาะสมกับอัตราขยายด้านหน้าที่ต้องการซึ่งจะอยู่ในช่วงระหว่างประมาณ 5.2 dB_d จนถึง 11.2 dB_d

ผลของเสากลาง ในการนำเอาสายอากาศไดโพลมาสแต็คกันในแนวดิ่งและยึดติดเข้ากับเสากลางนั้น เราสามารถใช้ผลการทดลองเกี่ยวกับสายอากาศยากิมาใช้งานได้ ทั้งนี้ เพราะเสากลางทำหน้าที่คล้ายรีเฟล็คเตอร์ และไดโพลทำหน้าที่คล้ายไดรเว่นอีลีเมนต์ รูปที่ 2.27 แสดงผลการทดลองดังกล่าว จากกราฟในรูปที่ 2.27 (ก) จะเห็นว่าเมื่อนำเอาไดโพลตัวหนึ่งมาวางห่างจากเสากลางประมาณ 0.22λ (วัดจากเส้นผ่านศูนย์กลางของไดโพลถึงเส้นผ่านศูนย์กลางของเสากลาง)ค่าความต้านทานการแผ่คลื่น (radiation resistance) หรืออิมพีแดนซ์ของสายอากาศที่ความถี่เรโซแนนซ์จะเป็น 50 โอห์ม และเมื่อนำมาวางใกล้เสากลางมากขึ้น ค่าความต้านทานของสายอากาศจะลดลง ในทำนองกลับกันเมื่อเลื่อนให้ห่างจากเสากลางมากขึ้น ค่าความต้านทานนี้ก็จะเพิ่มขึ้น ขณะเดียวกัน ขนาดเส้นผ่านศูนย์กลางของไดโพลจะมีผลกับค่าความต้านทานเพียงเล็กน้อยที่ระยะห่างระหว่าง 0.15λ และ 0.3λ

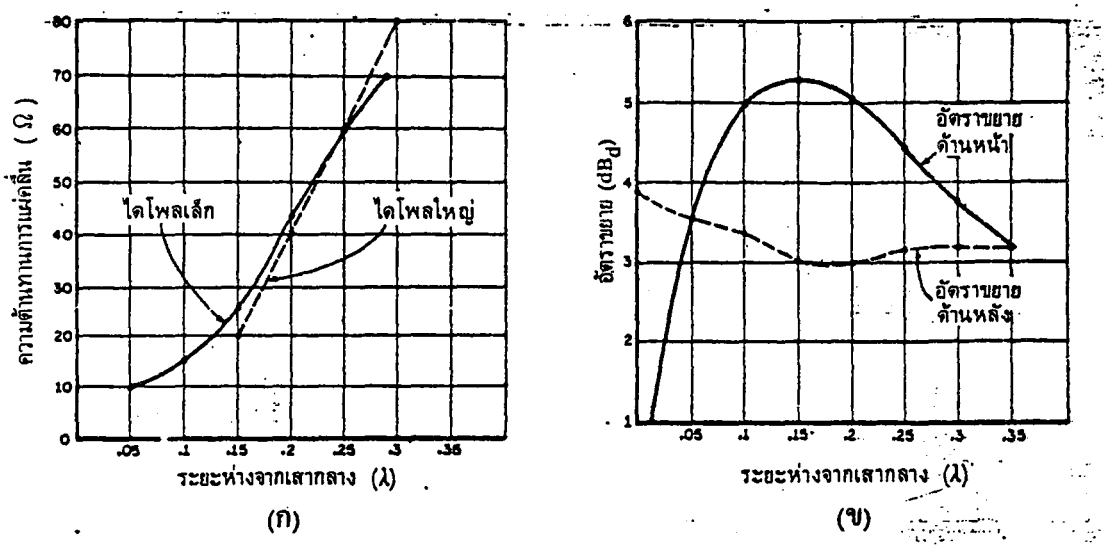
ส่วนกราฟในรูปที่ 2.27(ข)แสดงผลของระยะห่างระหว่างไดโพลและเสากลางที่มีต่ออัตราขยายในทิศด้านหน้าและด้านหลัง ตัวเลขอัตราขยายที่แสดงในกราฟ เป็นการเทียบกับอัตราขยายของไดโพลเมื่อไม่มีเสากลางจะเห็นได้ว่าที่ค่าระยะห่างน้อยกว่า 0.1λ อัตราขยายด้านหน้าจะตกลงอย่างรวดเร็วขณะที่อัตราขยายด้านหลังจะเพิ่มขึ้น และเมื่อย้อนกลับไปดูกราฟในรูปที่ 2.27 (ก) จะเห็นว่าที่ระยะ 0.1λ นั้น ค่าความต้านทานสำหรับไดโพลขนาดเล็กจะลดลงมาเหลือเพียงประมาณ 15 โอห์ม ซึ่งการที่ความต้านทานของสายอากาศต่ำ จะทำให้แถบความถี่ใช้งานของสายอากาศแคบลงกว่าเดิมมาก ดังนั้นในทางปฏิบัติแล้วเราจึงมักจะวางไดโพลให้ห่างจากเสากลางไม่น้อยกว่า 0.1λ (และมักจะไม่ต่ำกว่า 0.15λ สำหรับไดโพลขนาดใหญ่) ยิ่งห่างมากยิ่งดี เพราะแถบความถี่จะกว้างขึ้น แต่ก็ไม่ควรห่างเกินกว่า 0.25λ ซึ่งอัตราขยายด้านหน้าเริ่มตกลงมามาก λ (แลมด้า) คือ ความยาวคลื่นในอากาศของความถี่กลางของช่วงความถี่ที่เราต้องการใช้งานโดยที่ค่า λ หาได้ดังนี้

$$\lambda = \frac{29980}{f(MHz)} \quad \text{เซนติเมตร}$$

โดยที่ f คือ ความถี่กลางในหน่วยเป็น MHz ดังนั้นระยะในหัวข้อนี้ซึ่งเป็นระยะที่อยู่กลางอากาศ จึงสามารถใช้สูตรนี้คำนวณหาระยะห่างได้เลยโดยตรง โดยไม่ต้องมีตัวคูณมาลดตัวอย่าง เช่น ต้องการทราบว่าค่า 0.22λ สำหรับความถี่กลาง 145 MHz จะยาวเท่าไร

$$0.22\lambda = 0.22 * \frac{29980}{145}$$

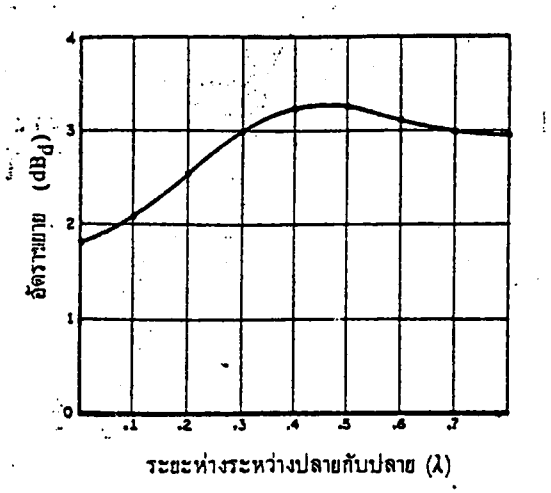
= 45.5 เซนติเมตร



รูปที่ 2.27 แสดงผลการทดลองนำเอาไดโพลตัวเดียวมาวางห่างจากเสากลางที่ระยะต่างๆ (อัตราขยายในรูป (ข) เป็นอัตราขยายสูงสุดที่เป็นไปได้ทางทฤษฎี แต่ในทางปฏิบัติมักจะได้น้อยกว่านี้)

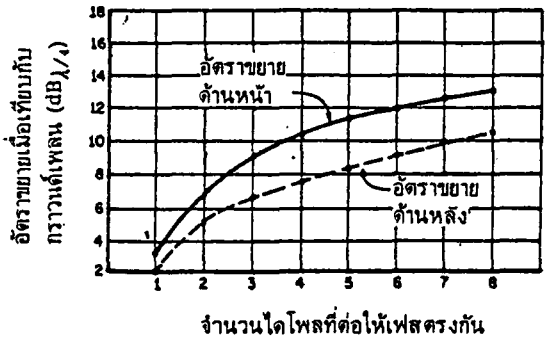
ผลของระยะห่างระหว่างไดโพล ในรูปที่ 2.28เป็นกราฟที่ได้จากการทดลองนำเอาไดโพลมาสแต็คกันในแนวดิ่งเดียวกัน เพื่อดูว่าไดโพลควรจะวางห่างกันเท่าไรดีถ้าไดโพลอยู่ชิดกันมากเกินไป

จะมีผลกระทบถึงกันทางด้านอิมพีแดนซ์ จะเห็นว่าที่ระยะห่างน้อยกว่า 0.3λ ระหว่างปลายของไดโพลตัวหนึ่งกับอีกตัวหนึ่งที่อยู่ติดกัน อัตราขยายโดยรวมของทั้ง 2 ตัว จะน้อยกว่า 3dB คือน้อยกว่า 2 เท่าของอัตราขยายของตัวเดียว และที่ระยะห่างระหว่าง 0.4λ และ 0.5λ จะได้อัตราขยายสูงที่สุด



รูปที่ 2.28 แสดงผลการทดลองนำเอาไดโพลมาสแต็คกัน 2 ตัว ที่ระยะห่างต่างๆ ระหว่างปลายของไดโพลตัวหนึ่งกับอีกตัวหนึ่งที่อยู่ติดกัน

อัตราขยายที่ได้จากการสแต็ค ในรูปที่ 2.29 แสดงอัตราขยายที่ได้เมื่อสแต็คไดโพลในแนวดิ่งเดียวกันโดยให้ไดโพลแต่ละตัวอยู่ห่างจากเสากลาง 0.22λ เพื่อให้มีอิมพีแดนซ์เป็น 50 โอมห์ และปลายของไดโพลอยู่ห่างกัน 0.48λ เพื่อให้ได้อัตราขยายสูงที่สุด อัตราขยายที่แสดงในกราฟนี้เป็นอัตราขยายเมื่อเทียบกับสายอากาศกราวด์นเพลน λ/4 และเป็นอันตราขยายขั้นต่ำที่ควรจะได้ในทางปฏิบัติยกตัวอย่างเช่นเมื่อนำเอาไดโพลมาทำเป็น 8 สแต็คจะได้อัตราขยายด้านหน้าเป็นประมาณ 13 dB และได้อัตราขยายด้านหลังประมาณ 10.5 dB เป็นต้น ในกรณีที่ต้องการแปลงเป็น dB_d ให้เอา 1.8 ไปลบออกจากตัวเลขที่แสดงกราฟรูปที่ 2.29 อย่างเช่น 13 dB จะเท่ากับ 11.2 dB_d เป็นต้น



รูปที่ 2.29 แสดงความสัมพันธ์ระหว่างอัตราขยายที่ได้กับจำนวนไดโพลที่นำมาสแต็คกันให้ห่าง 0.22λ จากเสากลาง และปลายห่างกัน 0.48λ

การหาความยาวของไดโพล ในการสร้างไดโพล ซึ่งเป็นสายอากาศพื้นฐานที่มีความยาวของอีลีเมนต์เป็นครึ่งของความยาวคลื่นและป้อนสัญญาณนำสัญญาณเข้าตรงกลางนั้นเราจะคำนวณหาความยาวทั้งหมด (L) ของไดโพลได้จากสมการต่อไปนี้

$$L = \frac{\lambda}{2} * v_r$$
$$= \frac{29980}{2 * f(MHz)} * v_r$$
$$= \frac{14990}{f(MHz)} * v_r$$

โดยที่ f คือความถี่กลางของย่านความถี่ที่ต้องการในหน่วยเป็น Mhz และ v_r คือค่าความเร็วสัมพัทธ์ของท่ออะลูมิเนียมที่ใช้ทำเป็นตัวไดโพล ถ้าเป็นท่อที่มีขนาดเส้นผ่าศูนย์กลางใหญ่ขึ้น ค่า v_r จะมีค่าน้อยลง ทำให้ความยาวที่ต้องใช้สั้นลง โดยทั่วไปค่า v_r จะมีค่าประมาณ 0.911, 0.903, 0.896

และ 0.882 สำหรับท่ออะลูมิเนียมขนาดเส้นผ่านศูนย์กลางภายนอก ประมาณ 1/2 นิ้ว, 5/8 นิ้ว, 3/4 นิ้ว และ 1 นิ้วตามลำดับเมื่อนำมาใช้กับความถี่กลางที่ 145 MHz ในการสร้างจริงจะได้ผลว่า VSWR ที่ความถี่กลางจะมีค่าน้อยกว่า 13:1 ซึ่งก็นับว่าใช้งานได้จนไม่จำเป็นต้องไปทำอะไรอีก

ตัวอย่างเช่น ถ้าท่านเลือกความถี่กลางเป็น 145 MHz และใช้ท่ออะลูมิเนียมขนาดเส้นผ่านศูนย์กลาง 1.2 เซนติเมตร (ประมาณ 1/2 นิ้ว) จะต้องตัดไดโพลให้ยาวดังนี้

$$L = \frac{14990 * 0.911}{145}$$

$$= 94.18 \quad \text{เซนติเมตร}$$

การต่อสายแมทชิ่ง สมมติว่าต้องการสแต็คไดโพล 2 ตัวที่มีอิมพีแดนซ์ 50 โอห์มให้มีอิมพีแดนซ์รวมเป็น 50 โอห์ม เนื่องจากไดโพลทั้งสองตัวจะต้องมีค่าอิมพีแดนซ์เท่ากัน ดังนั้นก็หมายความว่าไดโพล ทั้งสองจะต้องถูกแปลงให้มีอิมพีแดนซ์เป็น 100 โอห์มเสียก่อนเพื่อว่าเมื่อนำมาขนานกันแล้วอิมพีแดนซ์รวม จึงจะลดลงมาครึ่งหนึ่งเป็น 50 โอห์ม ตามต้องการ

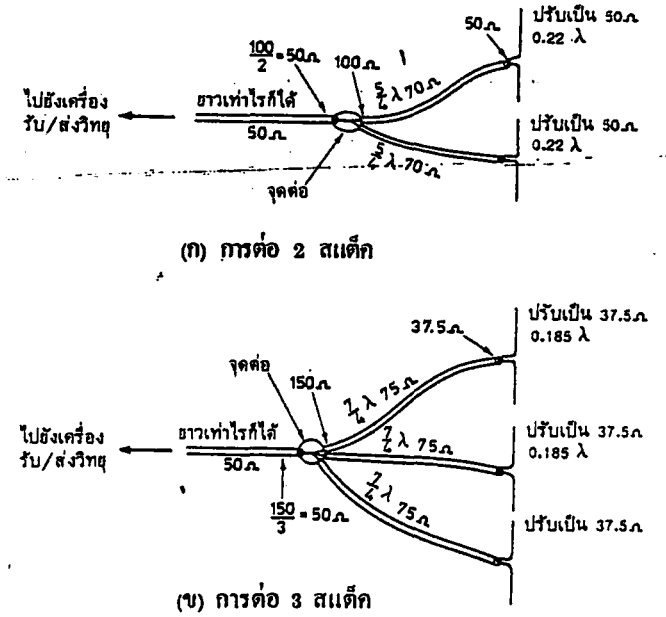
วิธีการแปลงอิมพีแดนซ์ที่ง่ายวิธีหนึ่งคือ ใช้ *สายแมทชิ่ง* (matching section) หรือที่เรียกตามภาษาวิชาการว่า *ควอเตอร์เวฟทรานสฟอร์เมอร์* (quarter-wave transformer) หรือที่นิยมเรียกกันอย่างกว้างๆ ว่า *สายเฟสชิ่ง* (phasing line) วิธีนี้มีหลักการที่พอจะสรุปได้สั้นๆ ดังนี้เมื่อนำเอาสายแมทชิ่งที่มีอิมพีแดนซ์เป็น Z_0 ที่มีความยาวทางไฟฟ้าเป็นจำนวน *เลขคี่* ของ $\lambda/4$ มาต่อเข้ากับอุปกรณ์ (สายอากาศ) ที่มีอิมพีแดนซ์เป็น Z_1 อิมพีแดนซ์ (Z_s) ที่ปรากฏที่ปลายอีกข้างหนึ่งของสายแมทชิ่งมีความสัมพันธ์กับ Z_0และ Z_1 ดังนี้

$$z_0 = \sqrt{z_s z_1}$$

ดังนั้นเมื่อทราบค่าอิมพีแดนซ์ของสายแมทชิ่งที่ใช้และอิมพีแดนซ์ที่ถูกแปลงมาแล้ว ก็จะหาค่าอิมพีแดนซ์ของสายอากาศที่ต้องสร้างขึ้นได้จาก

$$z_1 = \frac{z_0^2}{z_s}$$

จากตัวอย่างที่ยกมาข้างต้น เราจะต้องแปลงอิมพีแดนซ์ของไดโพลแต่ละตัวให้เป็น 100 โอห์มเสียก่อน ซึ่งก็ทำได้โดยนำเอาสายโคแอกเชียลที่มีอิมพีแดนซ์ 70 โอห์ม และมีความยาวทางไฟฟ้าเป็นจำนวนเท่าเลขคี่ของ $\lambda/4$ มาทำเป็นสายแมทชิ่งโดยแทรกเข้าไปที่ตัวไดโพลแต่ละตัวก่อน ดังนั้นเมื่อมองที่ปลายอีกข้างหนึ่งของสายแมทชิ่ง เราก็จะมองเห็นอิมพีแดนซ์เป็น 100 โอห์ม เมื่อนำมาขนานกันก็จะได้อิมพีแดนซ์รวมเป็น 50 โอห์ม ซึ่งก็พร้อมที่จะต่อเข้ากับสายโคแอกเชียล 50 โอห์ม เพื่อไปเข้ากับเครื่องรับ/ส่งวิทยุต่อไป รูปที่ 2.30 (ก) แสดงวิธีการต่อสายแมทชิ่งสำหรับ 2 สแต็คที่ได้อธิบายมานี้ ขอให้สังเกตด้วยว่าสายแมทชิ่ง 70 โอห์มที่ใช้ต้องยาว $5/4\lambda$ เพื่อให้ยาวพอที่จะมาต่อรวมกันได้ (ดูจากหัวข้อ "ความยาวจริงของสายแมทชิ่ง" ประกอบด้วย)



รูปที่ 2.30 แสดงตัวอย่างวิธีการแมทชิ่งไดโพล 2 สแต็คและ 3 สแต็ค

ความยาวจริงของสายแมทชิ่ง ความยาวของสายแมทชิ่งที่ระบุในหัวข้อก่อนเป็นความยาวทางไฟฟ้า ความยาวจริงจะสั้นกว่าความยาวทางไฟฟ้า ทั้งนี้เพราะคลื่นวิทยุจะเดินทางในสายได้ช้ากว่าเดินทางในอากาศ ดังนั้นเมื่อต้องการทราบความยาวที่แท้จริงของสายโคแอกเชียล ก็จะต้องเอาตัวคูณทางความเร็ว (velocity factor เขียนย่อๆ ว่า v) ของสายโคแอกเชียลที่ใช้มาคูณเข้าไป ซึ่งค่าตัวคูณนี้จะเปิดดูได้จากแคตาล็อกของผู้ผลิต หรือดูจากตำราที่เกี่ยวกับวิทยุสมัครเล่นก็ได้

สมมติว่าท่านต้องการหาความยาวจริงของสายโคแอกเชียลเบอร์ RG-11A/U ซึ่งมีค่าตัวคูณทางความเร็ว (v) เป็น 0.66 ต้องการตัดให้ได้ค่าความยางทางไฟฟ้าเป็น 5/4λ จะหาค่าความยาวจริงได้ดังนี้คือ

$$\text{ความยาวจริง} = \frac{5}{4}\lambda * v$$
$$= \frac{5}{4}\lambda * V$$
$$= \frac{5}{4}(\frac{29980}{f}) * v$$
$$= \frac{5}{4} * \frac{29980}{145} * 0.66$$
$$= 170.6 \text{ เซนติเมตร}$$

<u>สรุปหลักการสแต็คสายอากาศไดโพล</u>

จากหลักการเบื้องต้นทั้งหมดที่กล่าวมาแล้วนี้นั้นสามารถสรุปวิธีการสแต็คสายอากาศไดโพลเพื่อนำมาทำเป็นคอลลิเนียร์แบบขนานได้ดังนี้

1. สายอากาศไดโพลแต่ละตัว จะต้องถูกตัดให้เรโซแนนซ์ที่มีความถี่กลางของย่านความถี่ที่ต้องการ

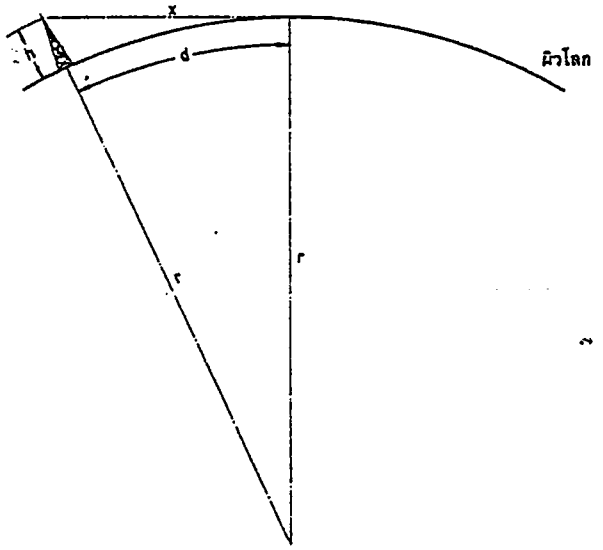
2. สายอากาศไดโพลแต่ละตัว จะต้องถูกนำมาต่อขนานกันในลักษณะที่ให้เรียกสัญญาณออกมามีเฟสตรงกัน หรือที่เรียกทับศัพท์ว่า *อินเฟส* (in-phase) กัน ดังนั้นไดโพลทุกตัวจึงควรถูกติดตั้งในลักษณะเดียวกันหมด ถูกต่อเข้ากับสายนำสัญญาณเดียวกัน นำมาจัดวางเรียงในแนวดิ่งเดียวกัน สายนำสัญญาณที่ต่อจากไดโพลแต่ละตัวมารวมกันจะต้องมีความยาวทางไฟฟ้าเท่ากัน

3. ระบบสายอากาศคอลลิเนียร์ จะต้องมีอิมพีแดนซ์เหมาะสมกับอิมพีแดนซ์ของเครื่องรับ/ส่งวิทยุที่ใช้ ดังนั้นจึงอาจใช้สายโคแอกเชียลมาทำเป็นสายแมทชิ่งเพื่อแปลงอิมพีแดนซ์ให้พอเหมาะกัน ในบางกรณีต้องขยับระยะระหว่างไดโพลและเสากลางให้ไดโพลมีอิมพีแดนซ์ที่เหมาะสมที่ถูกดัดแปลงมาได้สายแมทชิ่งจะต้องถูกตัดตามค่าความถี่กลางที่ต้องการและตามค่าตัวคูณทางความเร็ว (v) ของสายที่ใช้

2.6 การหาระยะสุดสายตาสำหรับคลื่นวิทยุในประเทศไทย

เป็นที่ทราบกันดีว่าการติดต่อสื่อสารในย่าน VHF และสูงกว่านั้น มักใช้การเดินทางแบบคลื่นอวกาศ (space wave) เป็นหลักโดยประกอบด้วยคลื่น 2 ส่วน ส่วนแรกเป็น *คลื่นทางตรง* (direct wave) ซึ่งเดินทางจากสายอากาศส่งไปยังสถานีรับโดยตรง อีกส่วนหนึ่งเป็น *คลื่นสะท้อน* (reflected wave) จากพื้นดิน (รวมทั้งสะท้อนจากวัตถุอื่น เช่น ตึก เป็นต้น) คลื่นทั้งสองส่วนเมื่อมาถึงเครื่องรับก็จะกลายเป็นสัญญาณทั้งหมดที่สถานีรับรับได้

ในการศึกษาเรื่องระยะการติดต่อสื่อสารเบื้องต้นนั้น มักจะละเว้นการลงไปกล่าวถึงคลื่นสะท้อนโดยละเอียดก่อนเพื่อความง่ายโดยระบุว่าเพื่อให้การติดต่อสื่อสารแน่นอน อย่างน้อยสถานีส่งและสถานีรับควรจะมองเห็นกันโดยไม่ถูกส่วนโค้งของผิวโลกบัง ระยะสูงสุดที่สามารถติดต่อกันได้โดย *มองเห็น* กันนี้มีชื่อเรียกว่า *ระยะสุดสายตา* (line of sight หรือ geometric horizon หรือ optical horizon) ซึ่งถูกจำกัดโดยความสูงของสายอากาศและรัศมีความโค้งของผิวโลก



รูปที่ 2.31 แสดงการหาระยะสุดสายตา (d)

ในรูปที่ 2.31 แสดงการหาระยะสุดสายตา ในที่นี้ h คือความสูงของสายอากาศส่งหรือรับ r เป็นรัศมีความโค้งของผิวโลกจากหลักตรีโกณมิติ เราได้ว่า

$$(r+h)^2 = x^2 + r^2$$

ซึ่งกระจายใหม่ได้เป็น

$$x = \sqrt{2rh + h^2}$$

เนื่องจากรัศมีความโค้งของผิวโลก (r) มีค่ามากกว่า h มากๆ (เว้นแต่จะออกไปส่งในอวกาศสูงๆ) ดังนั้น 2rh มีค่ามากกว่า h^2 มากมาย จนสามารถตัดทิ้งค่า h^2 ได้ จึงประมาณได้ว่า

$$x = \sqrt{2rh}$$

และเนื่องจากระยะ x มีค่าโดยประมาณใกล้เคียงกับ d มาก จึงสรุปได้ว่า

$$d = \sqrt{2rh}$$

เป็นความโชคดีของนักวิทยุทั้งหลาย ที่ระยะการติดต่อสามารถไปได้ไกลกว่าระยะสุดสายตาอีกพอสมควรเพราะมีบรรยากาศที่หุ้มห่อโลกมาช่วยหักเหคลื่นก่อนที่จะพุ่งออกนอกโลกให้ค่อยๆ เบนกลับมายังพื้นโลกได้เหตุผลสำหรับเรื่องนี้คือ บรรยากาศหุ้มห่อโลกมีความหนาแน่นต่างกันตามระดับความสูง บริเวณใกล้พื้นโลกจะมีความหนาแน่นมาก ยิ่งสูงขึ้นไปยิ่งมีความหนาแน่นน้อยลง ทำให้ผิวหน้าของคลื่น (wavefront) ด้านที่อยู่ใกล้ผิวโลกเคลื่อนที่ช้ากว่าผิวหน้าของคลื่นด้านที่อยู่ห่างกว่าผิวหน้าของคลื่นจึงค่อยๆ เบี่ยงกลับลงมาหาพื้นโลกได้ นั่นคือเส้นทางเดินของคลื่นย่าน VHF และสูงกว่าจึงเดินทางไม่เป็นเส้นตรงด้วยผลจากบรรยากาศ ระยะทางที่ได้ไกลขึ้นเนื่องจากบรรยากาศนี้มีชื่อเรียกว่า ***ระยะขอบคลื่นวิทยุ***(radio horizon) รัศมีความโค้งของผิวโลก จึงดูเหมือนเพิ่มขึ้นด้วยตัวคูณ k ซึ่ง

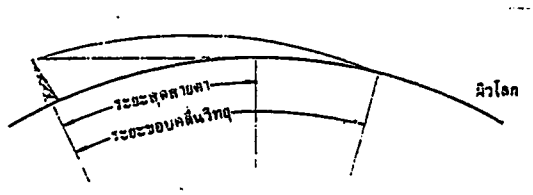
เรียกว่า *สัมประสิทธิ์รัศมีความโค้งของโลกสำหรับการแพร่กระจายคลื่นวิทยุ* นั่นคือ ระยะขอบคลื่นวิทยุ เป็น

$$d = \sqrt{2krh}$$

ในเอกสารต่างๆ ที่เกี่ยวกับวิทยุสมัครเล่นและระบบการสื่อสารเบื้องต้นมักจะบอกว่าค่า k โดยเฉลี่ยสำหรับประเทศที่อยู่ในเขตอบอุ่น (อย่างเช่น อเมริกา , ญี่ปุ่น) มีค่าประมาณ 1.33 และรัศมีความโค้งของผิวโลกมีค่าประมาณ 6,400 กิโลเมตร ดังนั้น ระยะขอบคลื่นวิทยุตามข้อมูลนี้ก็จะมีค่าเป็น

$$d = \sqrt{17h}$$

โดยที่ d มีหน่วยเป็นกิโลเมตร และ h มีหน่วยเป็นเมตร



รูปที่ 2.32 แสดงระยะขอบคลื่นวิทยุสำหรับย่าน VHF หรือสูงกว่า

สำหรับประเทศไทย เคยมีการวิจัยเพื่อหาค่า k อยู่ 2 ครั้ง เป็นงานวิจัยในระดับปริญญาโทของภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

การวิจัยชิ้นแรกมีชื่อว่า "การหารัศมีความโค้งของโลกที่มีต่อการแพร่กระจายคลื่นวิทยุในประเทศไทยโดยใช้ข้อมูลที่วัดได้จากราดิโอซอนด์" ทำวิจัยโดย อำนวยศักดิ์ ทูลศิริ เมื่อปี พ.ศ. 2517 การวิจัยครั้งนั้น ใช้ข้อมูลอุณหภูมิ ความดัน และความชื้นสัมพัทธ์ของบรรยากาศในประเทศไทยที่กรมอุตุนิยมวิทยา กระทรวงคมนาคม ได้เก็บรวบรวมไว้ระหว่างปี 2497-2503 เป็นข้อมูลที่ได้จากสถานีตรวจอากาศที่ เชียงใหม่ กรุงเทพ ฯ และสงขลา ซึ่งเป็นสถานีตรวจอากาศที่ใช้ราดิโอซอนด์วัดข้อมูล วิธีการวัดทำโดยส่งเครื่องราดิโอซอนด์ติดไปกับลูกบอลลูน ซึ่งลอยตัวสูงด้วยอัตรา 300 เมตร

ต่อวินาที แล้วส่งข้อมูลต่างๆ สำหรับแต่ละความสูงกลับลงมายังสถานีตรวจอากาศนั้นๆ โดยใช้ความถี่ 403 MHz

จากข้อมูลที่ได้นำมาคำนวณหาค่าดรรชนีหักเหคลื่นวิทยุของอากาศ (n) ที่ระดับความสูงต่างๆ เหนือพื้นผิวของโลกขึ้นไป จากนั้นจึงนำมาเขียนกราฟกับระดับความสูงเพื่อหาความชัน ซึ่งทำให้สามารถคำนวณหาค่า k ของแต่ละสถานีตรวจอากาศในเดือนต่างๆได้ แล้วนำมาคำนวณหาค่าห้าสิบเปอร์เซ็นต์ของเวลา ซึ่งได้ว่า k = 1.59 (หมายความว่ามี 50% ของเวลาที่มีค่าเป็น 1.59 เวลาที่เหลือให้ค่า k มากหรือน้อยกว่านี้)

ตารางที่ 2.4 แสดงค่าสัมประสิทธิ์รัศมีความโค้งของโลกสำหรับการแพร่กระจายคลื่นวิทยุ จากผลการวิจัย โดย บัณฑิต พฤกษเมธานันท์ เมื่อปี พ.ศ. 2520

	ค่า k จากสถานีตรวจอากาศที่ต่างๆ			
	กรุงเทพฯ	เชียงใหม่	สงขลา	อุบลราชธานี
มกราคม	1.59	1.51	1.64	1.49
กุมภาพันธ์	1.57	1.59	1.72	1.54
มีนาคม	1.67	1.66	1.87	1.57
เมษายน	1.65	1.67	1.86	1.47
พฤษภาคม	1.75	1.73	1.72	1.66
มิถุนายน	1.78	1.78	1.78	1.59
กรกฎาคม	1.76	1.67	1.72	1.56
สิงหาคม	1.71	1.68	1.76	1.51
กันยายน	1.56	1.72	1.52	1.67
ตุลาคม	1.64	1.65	1.67	1.50
พฤศจิกายน	1.59	1.61	1.62	1.56
ธันวาคม	1.57	1.50	1.63	1.54
เฉลี่ยตลอดปี	1.65	1.65	1.71	1.55

การวิจัยอีกชิ้นหนึ่งชื่อ "การหาค่าหักเหของคลื่นวิทยุและค่าสัมประสิทธิ์รัศมีความโค้งของโลกสำหรับการแผ่กระจายคลื่นวิทยุ" ทำวิจัยโดย บัณฑิต พฤกษา เมธานันท์ เมื่อปี พ.ศ.2520 การวิจัยนี้คล้ายกับครั้งก่อน แต่ใช้ข้อมูลในช่วงปี 2509-2517 และได้ข้อมูลจากสถานีตรวจอากาศที่อุบลราชธานีเพิ่มเข้ามาอีกแห่ง ได้ว่าค่าเฉลี่ยตลอดปีที่กรุงเทพฯ เชียงใหม่ สงขลา และอุบลราชธานี เป็น 1.65, 1.65, 1.71 และ 1.55 ตามลำดับ ดังแสดงในตารางที่ 2.5 ซึ่งเมื่อเฉลี่ยค่า k จากทั้งสี่สถานีตรวจอากาศแล้ว จะได้ค่าห้าสิบเปอร์เซ็นต์ของเวลา (ค่าเฉลี่ย) สำหรับประเทศไทยเป็น k = 1.64

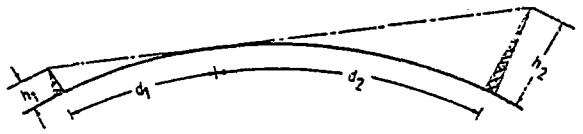
นั่นแสดงว่าประเทศไทยของเราโชคดีกว่าประเทศที่อยู่ในเขตอบอุ่นตรงที่ค่า k เฉลี่ยของเราสูงกว่าทำให้คลื่นวิทยุย่าน VHF หรือสูงกว่าสามารถเดินทางได้ไกลกว่าถ้ายึดถือค่า k = 1.64 เป็นค่าเฉลี่ยสำหรับประเทศไทยตามผลการการวิจัยครั้งหลังนี้ จะได้ระยะขอบคลื่นวิทยุสำหรับประเทศไทยเป็น

$$d = \sqrt{21h}$$

ในกรณีที่เป็นการติดต่อระหว่างสถานีดังรูปที่ 2.33 จะได้ความสัมพันธ์ระหว่างระยะติดต่อสูงสุดและความสูงของสายอากาศทั้งสองดังนี้

$$d = d_1 + d_2 = \sqrt{21h_1} + \sqrt{21h_2}$$

โดยที่ d มีหน่วยเป็นกิโลเมตร และ h_1, h_2 มีหน่วยเป็นเมตร ตัวอย่างเช่น ถ้าสถานีหนึ่งใช้เสาสูง 30 เมตร ต้องการติดต่อไกล 60 กิโลเมตร สายอากาศอีกด้านหนึ่งควรสูงอย่างน้อย 58 เมตร เป็นต้น



รูปที่ 2.33 แสดงการหาระยะทางการติดต่อระหว่าง

บทที่ 3

การออกแบบและการสร้าง

เนื่องจากโครงงานนี้ ประกอบไปด้วยอุปกรณ์ 3 ส่วนด้วยกัน คือ ส่วนของแพคเกตโมเด็ม (packet modem) เพื่อใช้แปลงสัญญาณดิจิตอลให้เป็นสัญญาณเสียง หรือ เพื่อใช้แปลงสัญญาณเสียงให้เป็นสัญญาณดิจิตอลกับส่วนของบอร์ดไมโครโปรเซสเซอร์ ที่มีหน้าที่รับข้อมูลจากสถานีส่ง และส่งข้อมูลนั้นกลับไปให้สถานีรับต่อไปและสุดท้ายคือ ส่วนของสายอากาศทำหน้าที่แพร่กระจายคลื่นวิทยุให้สามารถทำการติดต่อสื่อสารได้ในระยะทางที่ไกลออกไป

3.1 ส่วนของ packet modem

การออกแบบส่วนนี้ได้เลือกใช้ IC สำเร็จรูปเบอร์ TCM 3105 ของบริษัท TEXAS INSTRUMENT ซึ่งเป็น IC single-chip modem โดยมี baud rate เท่ากับ 1200 มาทำหน้าที่ในการแปลงสัญญาณดิจิตอลเป็นสัญญาณเสียง ซึ่งภายในประกอบด้วยส่วน Frequency Shift Keying (FSK) modulator, demodulator

ขาสัญญาณต่าง ๆ ของ IC เบอร์ TCM 3105 มีหน้าที่ต่าง ๆ ดังนี้

ขา 1 - V_{DD} (positive power supply) ต่อกับไฟแรงดันขนาด 5 V

ขา 9 - V_{SS} (negative power supply) ต่อลง ground

ขา 4 - R_XA (receive audio) เป็นขา input เพื่อรับ FSK เข้ามาเพื่อทำการ demodulation

ขา 8 - R_XD (receive data) โดยให้ข้อมูลออกมาเป็นสัญญาณดิจิตอล

ขา 11 - T_XA (transmit audio) เป็นขา output จะให้ output ออกมาเป็นสัญญาณเสียง sine wave

ขา 14 - T_XD (transmit data) เป็นขา output โดยให้ข้อมูลออกมาเป็นสัญญาณดิจิตอล ซึ่งข้อมูลที่ได้ออกมาจากส่วน demodulation

ขา 15 ,16 - osc_1 , osc_2 เป็นขาที่ต่อกับคริสตอล 4.43361 MHz เพื่อกำเนิดความถี่อ้างอิงให้กับวงจร

การออกแบบวงจรเพื่อทำแพคเก็ตโมเด็ม

ในส่วนของตัวโมเด็ม ใช้ไอซีเบอร์ TCM 3105 ทำการต่อวงจรดังรูปที่ 3.1 โดยดิปสวิตช์ (dipswitch) หมายเลข 1 - 8 จะทำการเลือกขาสัญญาณที่จะติดต่อกับพอร์ต (port) อนุกรม RS-232 ให้เหมาะสมกับซอฟแวร์ที่เราเลือกใช้

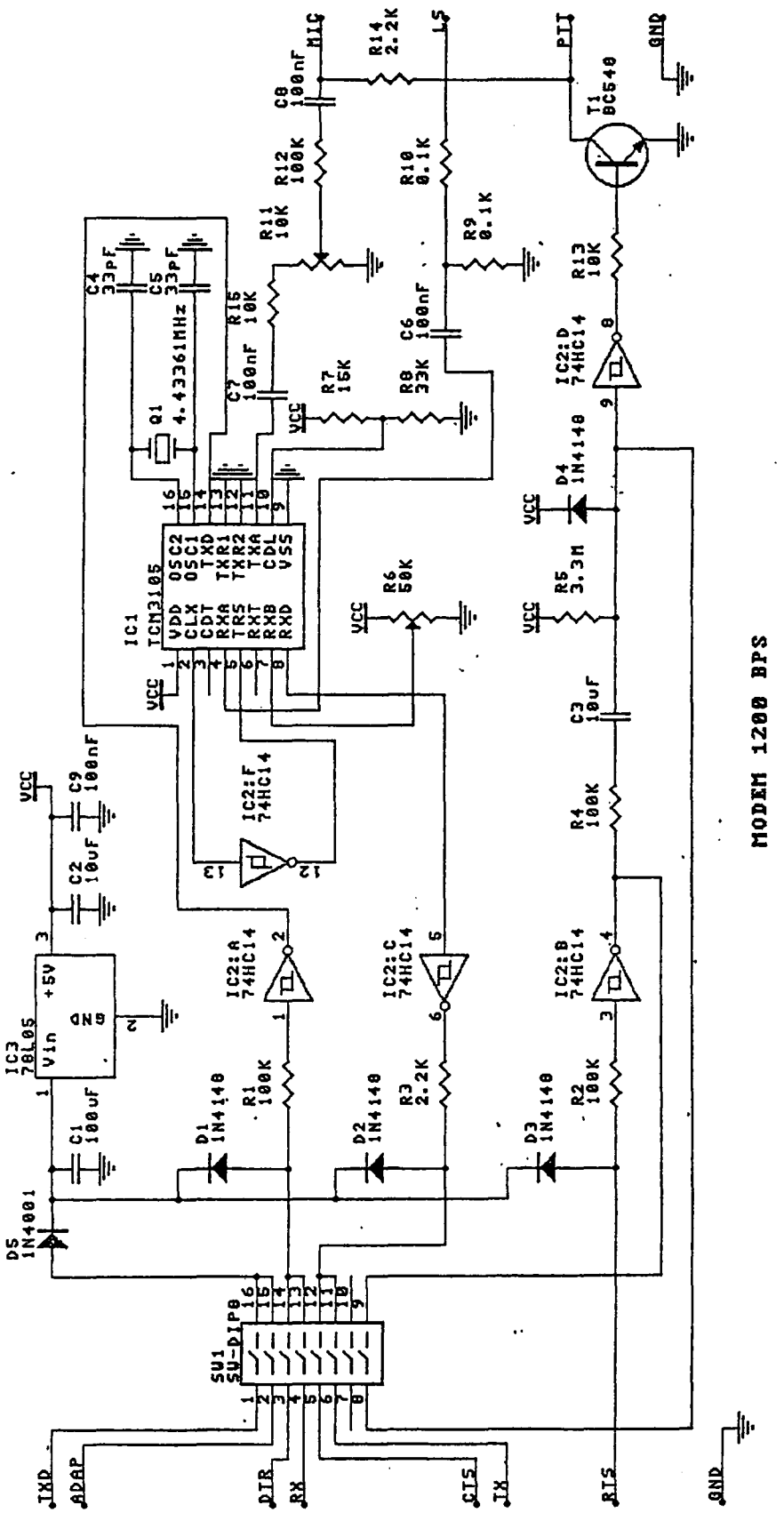
การทำงาน

วงจรสมบูรณ์ของแพคเก็ตโมเด็ม แสดงดังรูป 3.1 ซึ่งประกอบด้วยไอซีโมเด็มแบบชิปเดียว (IC-single chip modem 1200 BPS) เบอร์ TCM 3105 , อินเวอร์เตอร์บัฟเฟอร์ (invertor buffer) , ส่วนควบคุมการกดสวิตส์ส่งสัญญาณ (key-PTT) , ส่วนจ่ายแรงดัน 5 โวลต์ ให้แก่วงจร

IC_1 เป็นชิงเกิลชิพโมเด็มของเท็กซัสอินสทรูเมนท์ (TEXAS INSTRUMENT) ความเร็ว 1200 BPS เบอร์ TCM 3105 ทำงานด้วยคริสตอลความถี่ 4.43361 เมกะเฮิรตซ์ ทำการติดต่อกับส่วนจัดการข้อมูลที่เรียกว่าเป็น แพคเก็ตข้อมูล (Packet Data) ซึ่งประกอบอยู่ภายในตัวซีพียูและคอนโทรลยูนิต (CPU & CONTROL UNIT) โดยจะจัดการกับข้อมูลที่เข้ามาทางพอร์ตอนุกรม ให้กลายเป็นสัญญาณเสียงที่มอดดูเลทแล้ว (Modulate Audio Output) และจัดการกับสัญญาณเสียงมอดดูเลทที่เข้ามา (Modulate Audio Input) ให้กลายเป็นข้อมูลออกมาทางพอร์ตอนุกรม (Serial Data Output) โดยอาศัยโปรแกรมที่ผู้สร้างไอซีได้บรรจุไว้ภายในตัวอุปกรณ์

IC_2 เบอร์ 74HCI4 เป็นอินเวอร์เตอร์บัฟเฟอร์ ในส่วนของการติดต่อกับระบบ RS-232โดยมีดิปสวิตช์ SW1 เป็นตัวเลือกขาสัญญาณที่ใช้ให้เหมาะสมกับซอฟแวร์ที่ใช้งาน IC_3 เป็นไอซีเรกูเลเตอร์ เบอร์ 78LO5 สำหรับควบคุมไฟเลี้ยงของวงจรให้คงที่ 5 โวลต์ T_1 เป็นทรานซิสเตอร์เบอร์ BC 548 เป็นตัวควบคุมการกดสวิตช์ PTT สำหรับควบคุมการออกอากาศของเครื่องส่งวิทยุ ขาที่ 4 ของ IC_1 เป็นขาดิปสวิตช์ R_XA รับสัญญาณเสียงที่เป็นข้อมูลจากขั้วสัญญาณลำโพง (Speaker) ของเครื่องรับวิทยุ ขาที่ 11 ของ IC_1 เป็นขา T_XA (Transmit Audio) ส่งสัญญาณเสียงที่เป็นข้อมูลไปยังขั้วสัญญาณ MIC ของเครื่องส่งวิทยุ

รูปที่ 3.1 แสดงวงจรสมบูรณ์ของแพคเก็ตโมเด็ม 1200 BPS



3.2 ส่วนของบอร์ดไมโครโปรเซสเซอร์

คุณสมบัติของบอร์ด

- ใช้กับไมโครคอนโทรลเลอร์ในตระกูล MCS-51 (8031/ 8032)
- ความถี่สัญญาณนาฬิกา 11.0592 เมกะเฮิรตซ์
- หน่วยความจำอีพรอม 8 ถึง 32 กิโลไบต์ สำหรับโปรแกรม
- หน่วยความจำแรม 8 กิโลไบต์ สำหรับเก็บข้อมูล
- มีพอร์ตเบอร์ 8255 จำนวน 1 พอร์ต สำหรับการต่อไปใช้งานภายนอก
- มีวงจรพอร์ตอนุกรม (RS-232) สำหรับการต่อเข้ากับเครื่องไมโครคอมพิวเตอร์
- มีขั้วต่อสำหรับพอร์ต 1 ของไมโครคอนโทรลเลอร์โดยเฉพาะ
- มีขั้วต่อระบบ (System Bus) ทำให้ขยายระบบได้ง่าย และสามารถใช้กับบอร์ดขยายต่างๆ ที่จะมีขึ้นในอนาคต
- สามารถเลือกเบอร์หน่วยความจำหรือกำหนดคุณสมบัติต่างๆ ของบอร์ดได้ด้วยจัมพ์เปอร์

การทำงาน

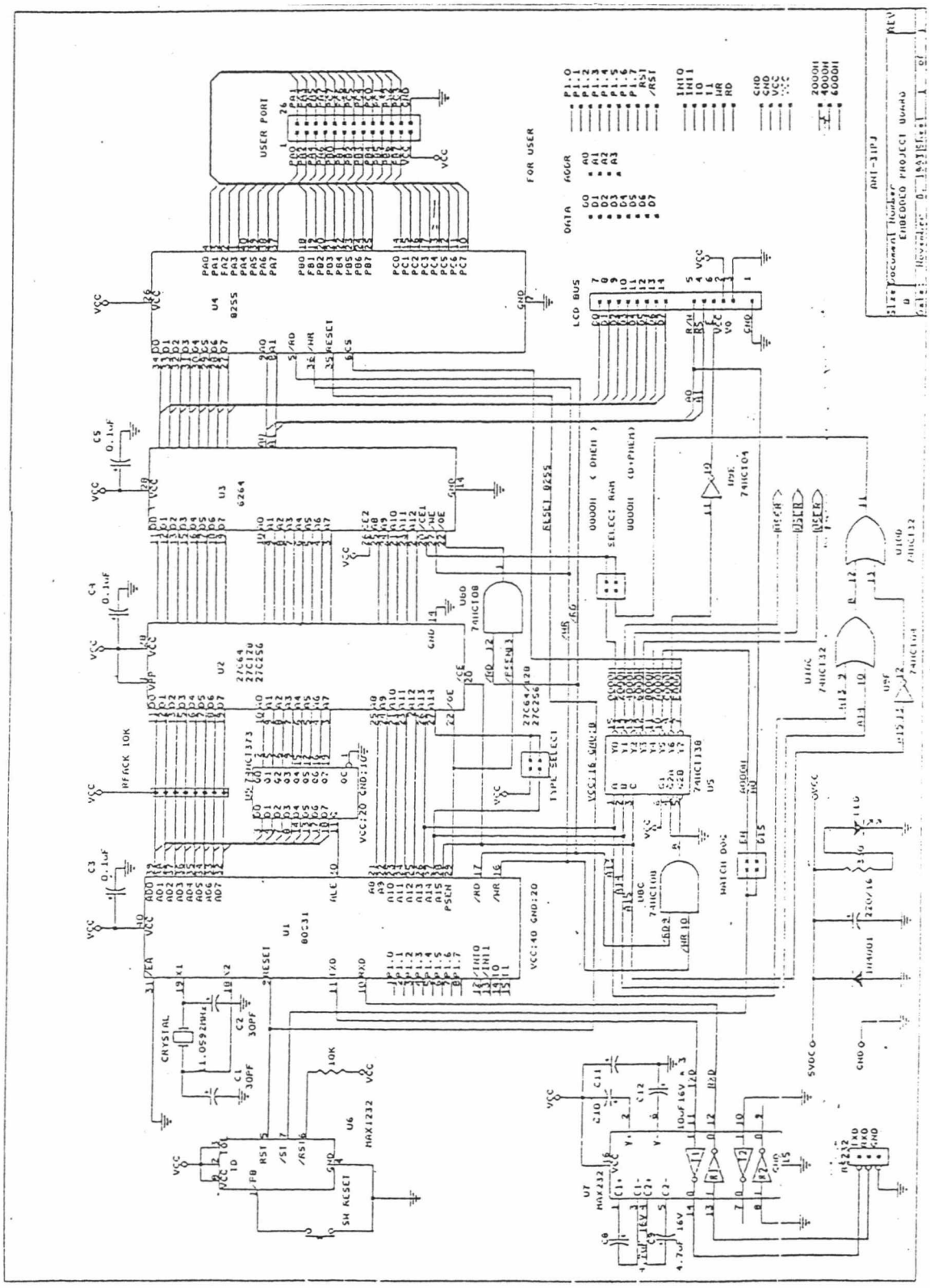
วงจรสมบูรณ์ของคอนโทรลเลอร์บอร์ด แสดงดังรูปที่ 3.2 ประกอบด้วยไมโครคอนโทรลเลอร์ 8031 , หน่วยความจำ (ROM / RAM) , พอร์ตอินพุต / เอาท์พุต , ส่วนถอดรหัสแอดเดรสของหน่วยความจำและพอร์ต , ส่วนแปลงแรงดันสำหรับพอร์ตอนุกรม RS-232

IC_1 เป็นชิพไมโครคอนโทรลเลอร์ในตระกูล MCS-51 ของ INTEL เบอร์8031 ทำงานที่ความถี่ 11.0592 เมกะเฮิรตซ์ ทำการติดต่อกับหน่วยความจำ 3 ส่วน คือ IC_2 เป็นหน่วยความจำภายนอกที่ใช้เก็บโปรแกรม (external program memory) ใช้กับอีพรอมขนาด 8 ถึง 32 กิโลไบต์ เบอร์ 2764, 27128 ส่วนที่สอง IC_3 เป็นหน่วยความจำที่ใช้เก็บข้อมูล (data memory) ใช้กับแรม 8 กิโลไบต์ เบอร์ 6264

ส่วนของพอร์ต นอกจากพอร์ตภายใน (internal port) ของซีพียู คือ port1 รวมทั้งขา INT0 , INT1, T0 และ T1 แล้ว ยังมีพอร์ตภายนอก คือ IC_4 เบอร์ 8255 ให้ใช้งานเพิ่มเติมอีกถึง 24 บิต

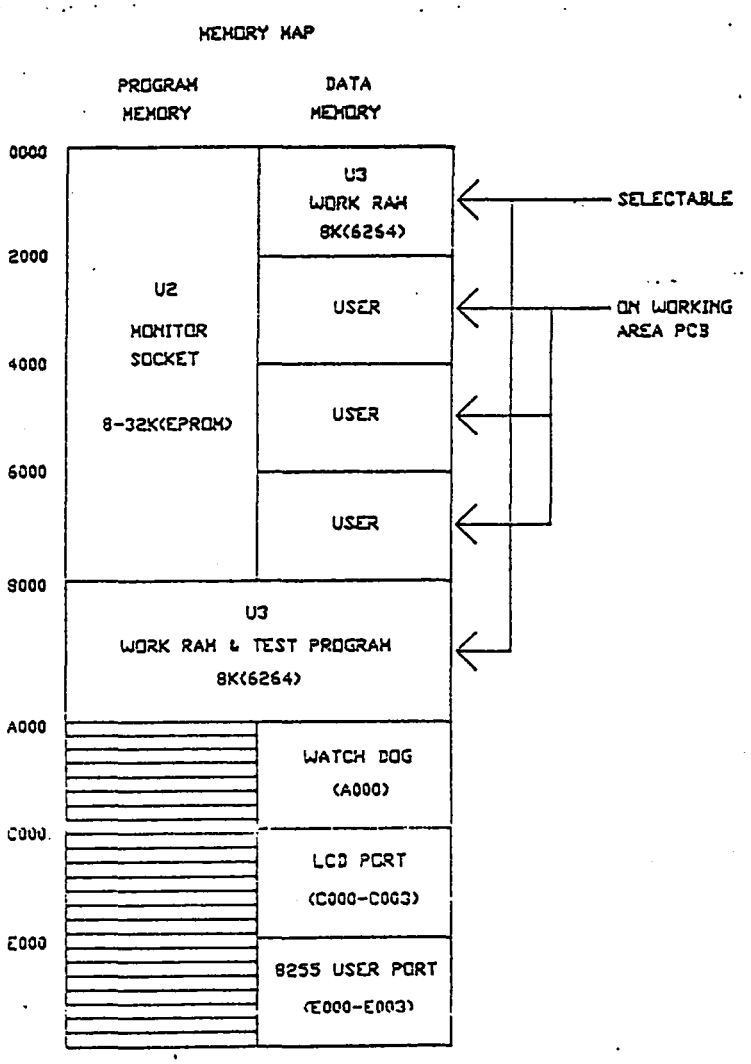
ส่วนถอดรหัสแอดเดรสของหน่วยความจำ สำหรับ IC_2 ใช้สายสัญญาณแอดเดรส A_{15} เพื่อกำหนดให้ IC_2 อยู่ที่แอดเดรส 0000H - 7FFFH ส่วน IC_4 และ IC_5 ใช้สายสัญญาณแอดเดรส A_{13}-A_{15} ร่วมกับ IC_5 จัดให้ IC_3 อยู่ที่แอดเดรส 8000H - 9FFFH และส่วนพอร์ต IC_{10} จัดให้อยู่ที่แอดเดรส E000H - F003H

รูปที่ 3.2 แสดงวงจรสมบูรณ์ของคอนโทรลเลอร์บอร์ด



การจัดหน่วยความจำ

รูปที่ 3.3 แสดงถึงการจัดหน่วยความจำของคอนโทรลเลอร์บอร์ด ซึ่ง IC_2 เป็นส่วนของรอมมอนิเตอร์อยู่ที่แอดเดรส 0000H - 7FFFH IC_3 เป็นแรมอยู่ที่แอดเดรส 8000H - 9FFFH และแอดเดรสช่วง E000H - E003H ถูกใช้สำหรับเป็นแอดเดรสของพอร์ต



รูปที่ 3.3 แสดงตารางการจัดหน่วยความจำและอินพุทเอาท์พุท

การนำไปใช้งานกับระบบ MICROPROCESSOR BASED REPEATER

IC_1 เป็นชิพไมโครคอนโทรลเลอร์เบอร์ 8031

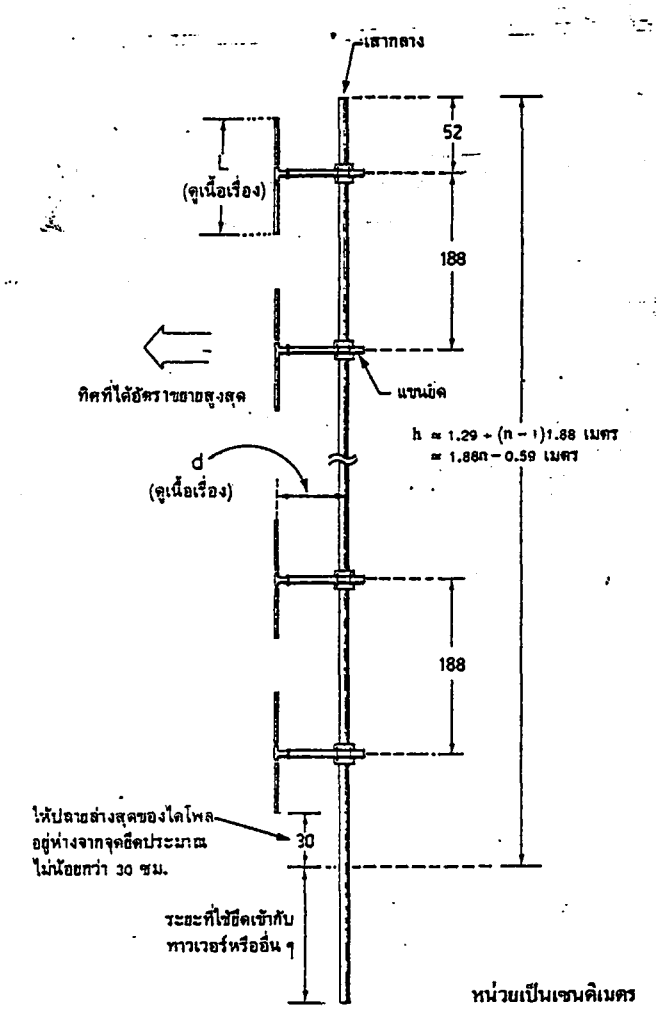
IC_2 เป็นหน่วยความจำภายนอกบรรจุโปรแกรมมอนิเตอร์เบอร์ 2764

IC_3 เป็นหน่วยความจำที่ใช้เก็บข้อมูล (data memory) เบอร์ 6264

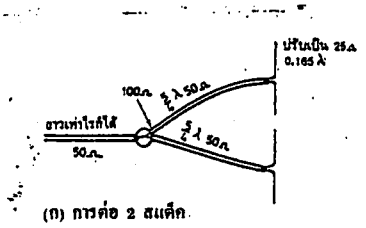
3.3 ส่วนของสายอากาศ

สายอากาศคอลลิเนียร์แบบขนานใช้ไดโพลมาสแต็ค ที่ทดลองสร้างนี้ใช้ลักษณะการติดตั้งดังรูปที่ 3.4 ระยะ 188 เซนติเมตร ซึ่งเป็นระยะห่างระหว่างแขนยึดของแต่ละไดโพล ได้มาจากการกำหนดให้ปลายของไดโพลที่อยู่ติดกันห่างประมาณ 0.46λ และความยาวเฉลี่ยของไดโพล มีค่าประมาณ 0.45λ ดังนั้นระยะห่างระหว่างแขนยึดจึงมีค่าเป็น 0.45λ + 0.46λ ซึ่งเท่ากับประมาณ 0.91λ ในกรณีนี้เลือกความถี่กลางเป็น 145 MHz ก็จะคำนวณค่า 0.91λ ได้เป็นประมาณ 188 เซนติเมตร

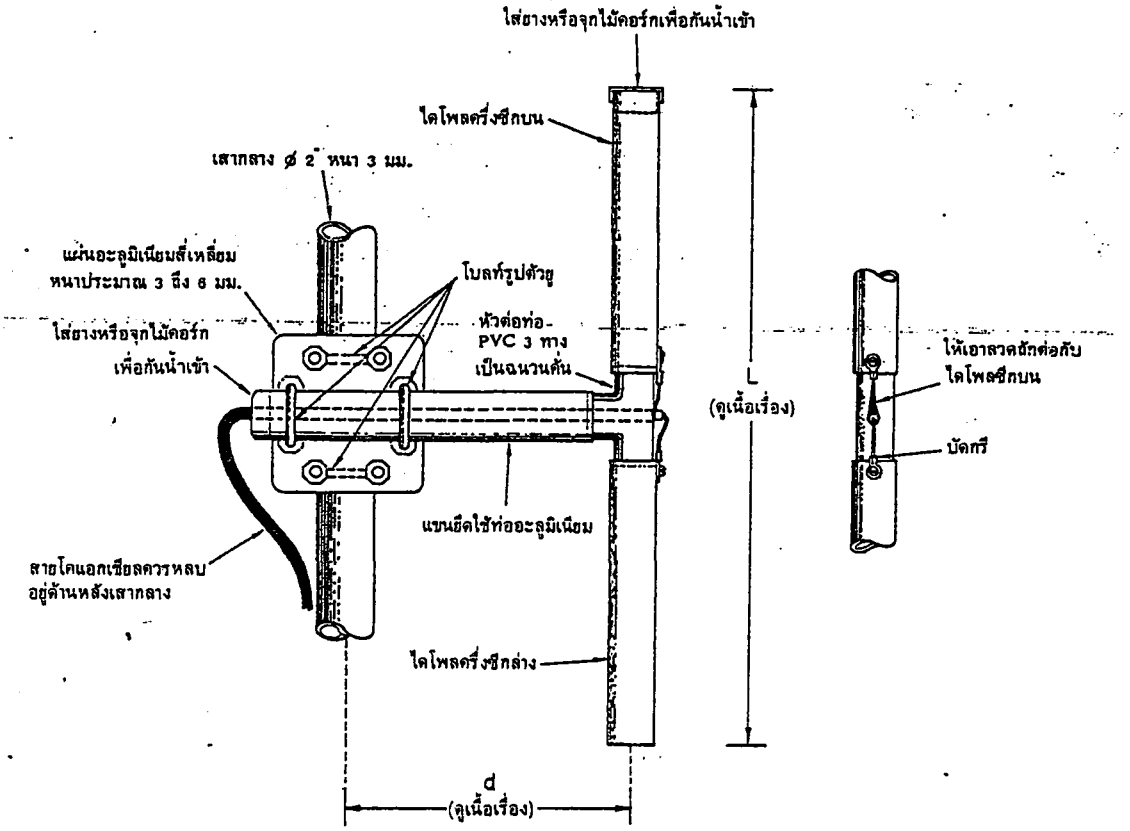
จุดยึดเข้ากับทาวเวอร์ หรือขายคาอยู่ห่างจากปลายล่างของไดโพลตัวล่างสุดประมาณ 30 เซนติเมตร ความยาวของเสากลางส่วนที่สูงจากจุดยึดหรือ h หาได้จากสมการที่อยู่ในรูป 3.4 โดยที่ n คือจำนวนไดโพลที่นำมาสแต็คกัน รูปที่ 3.5 แสดงวิธีการต่อสายแมทชิ่งสำหรับไดโพล 2 แสต็ค โดยใช้สายโคแอกเชียล 50 โอห์ม รูปที่ 3.6 แสดงรายละเอียดการสร้างไดโพลแต่ละตัว ตลอดจนการยึดติดเข้ากับเสากลาง รูปที่ 3.7 แสดงรูปภาพสายอากาศที่ทดลองสร้างขึ้นและติดตั้งอยู่บนยอดตึก 5 ชั้น คณะวิทยาศาสตร์ พระจอมเกล้า ลาดกระบัง



รูปที่ 3.4 แสดงวิธีการนำสายอากาศไดโพลมาสแต็คกันในแนวดิ่งเดียวกันทั้งหมด โดย n ที่แสดงในสมการคือจำนวนไดโพลที่นำมาสแต็คกัน



รูปที่ 3.5 แสดงวิธีการต่อสายแมทชิ่งเพื่อแปลงอิมพิแดนซ์ของระบบสายอากาศ

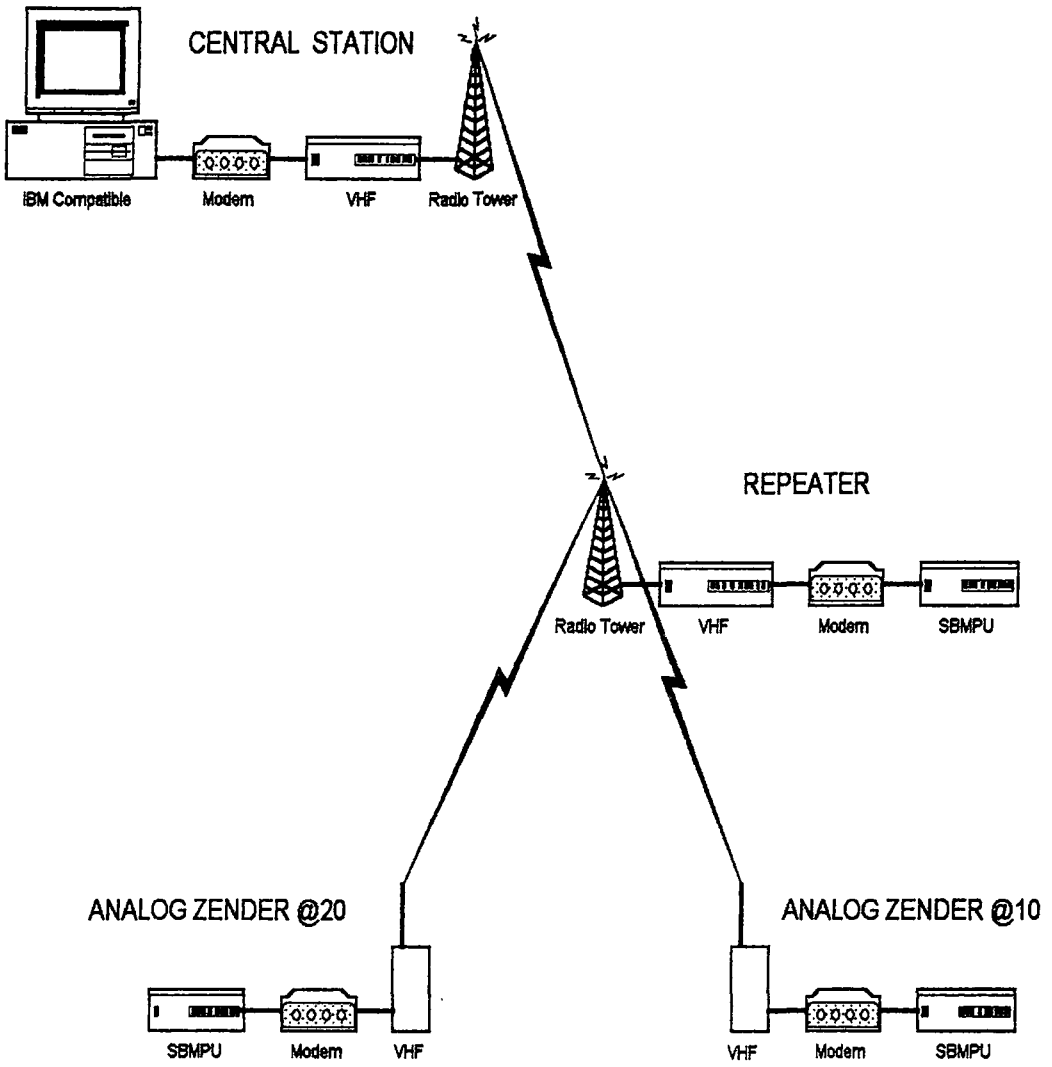


รูปที่ 3.6 แสดงรายละเอียดการสร้างไดโพลแต่ละตัว

รูปที่ 3.7 สายอากาศที่ทดลองสร้างขึ้นและติดตั้งอยู่บนยอดตึก 5 ชั้น
คณะวิทยาศาสตร์ พระจอมเกล้า ลาดกระบัง

2.4 **ส่วนของการทำงานทั้งระบบ**

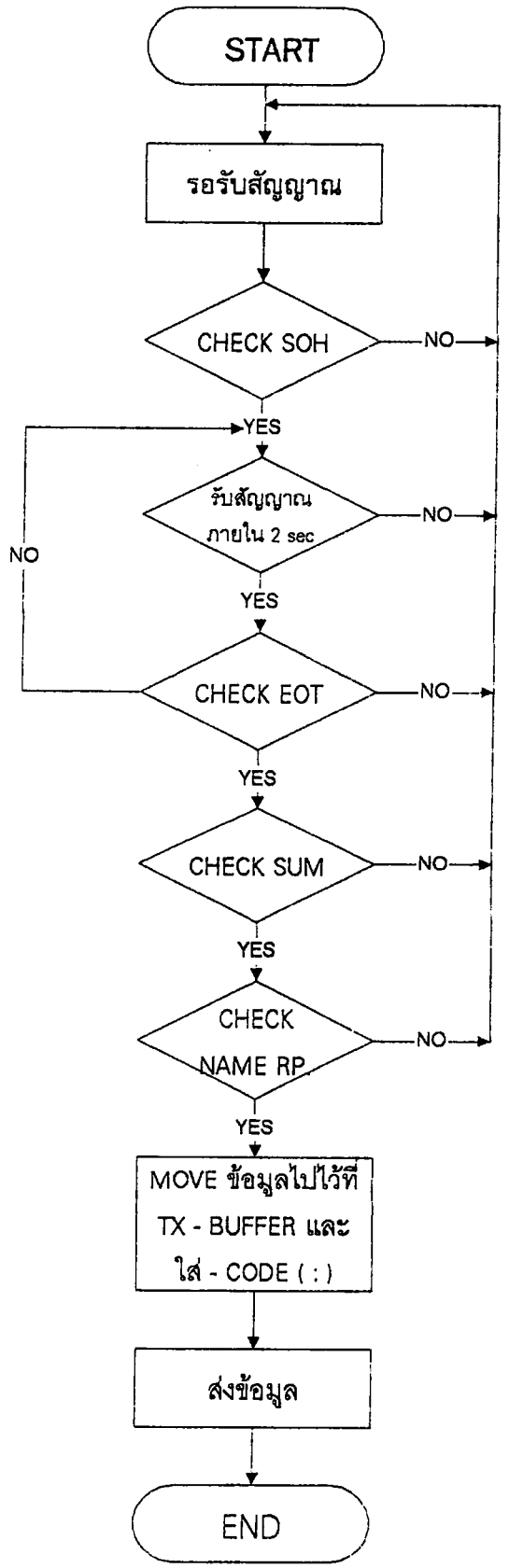
ในระบบการรายงานข้อมูล (DATA ACQUISITION) สำหรับการตรวจวัดคุณภาพน้ำในแหล่งต่างๆ จะทำการวัดค่า pH , Conductivity , Temperature และค่า DO เป็นต้น โดยมีสถานีลูกจะติดตั้งบนทุ่นลอยน้ำและมีหัววัดค่าต่างๆ ติดตั้งในตำแหน่งที่ต้องตรวจวัดคุณภาพของน้ำ โดยในจุดต่างๆก็จะมีสถานีลูกแหล่งน้ำละหนึ่งสถานี สถานีลูกจะทำการบันทึกค่าพารามิเตอร์ดังกล่าวภายในช่วงเวลาที่โปรแกรมได้และบันทึกเก็บไว้ ภายในช่วงเวลาหนึ่งและสถานีแม่ซึ่งเป็นศูนย์กลาง โดยจะมีศูนย์กลางที่เป็นการทำงานโดย PROGRAM ที่อยู่บนเครื่องคอมพิวเตอร์ต่อเชื่อมกับ PACKET MODEM และเครื่องวิทยุ ทำการเรียกสถานีรีพีทเตอร์ แล้วให้สถานีรีพีทเตอร์ทำการเรียกสถานีต่างๆ ที่อยู่โดยรอบให้ส่งข้อมูลที่ต้องการทีละสถานีมาให้สถานีรีพีทเตอร์ แล้วสถานีรีพีทเตอร์ก็จะส่งข้อมูลไปยังสถานีศูนย์กลางได้ทราบ การกำหนด PROGRAM ให้สถานีลูกตอบรับอย่างไร เมื่อมีการทำงานสนองตอบอย่างไรนั้น สามารถเขียนเป็น PROGRAM ติดตั้งไว้บน SINGLE BOARD MICROPROCESSOR UNIT ให้ทำงานได้ดังแผนภาพดังรูป 3.8



รูปที่ 3.8 แสดงแผนภาพการทำงานของระบบสื่อสาร

รูปที่ 3.9 แสดงรูปภาพสถานีตรวจวัด Analog Zender

รูปที่ 3.10 แสดงโฟลว์ชาร์ตการทำงานของโปรแกรม Microprocessor Based Repeater



Packet Format:

Start of Head	Caller Address	Station Address	Repeater Address	Command	Data	Byte Check Sum	End of Text

SOH PPH>EDG,R99:T,"ED"EOT

PPH : Caller Address

EDG : Station Address

R99 : Repeater Address

ED : Byte Check Sum (Two's complement)

SOH : 01H

EOT : 04H

บทที่ 4

การทดลอง

ในการทดลอง จะแบ่งการทดลองออกเป็น 3 ตอนด้วยกัน คือ

4.1 การทดสอบการทำงานของโปรแกรมรีพีทเตอร์และโปรแกรมสื่อสารที่เขียนขึ้น

4.2 การทดสอบการทำงานของแพคเกตโมเด็มและบอร์ดไมโครโปรเซสเซอร์ที่สร้างขึ้น โดยทำการติดต่อสื่อสารและส่งข้อมูลผ่านทางคลื่นวิทยุ

4.3 การทดสอบการทำงานของระบบสายอากาศที่สร้างขึ้น เพื่อทดสอบค่า VSWR (volttage standing wave ratio)

4.4 การทดสอบประสิทธิภาพของการติดต่อสื่อสาร เพื่อทดสอบระยะทางที่เหมาะสมในการติดต่อสื่อสารข้อมูล

ซึ่งแต่ละการทดลองมีรายละเอียดดังต่อไปนี้

4.1 การทดสอบการทำงานของโปรแกรมรีพีทเตอร์และโปรแกรมสื่อสาร

อุปกรณ์ที่ใช้ในการทดลองมีดังต่อไปนี้

1. เครื่องไมโครคอมพิวเตอร์ 1 ชุด
2. บอร์ดไมโครโปรเซสเซอร์ 1 ชุด
3. สายเชื่อมต่อ RS 232 1 ชุด
4. โปรแกรมการสื่อสารข้อมูล 1 ชุด

เนื่องจากในการเชื่อมต่อระหว่างไมโครคอมพิวเตอร์และแพคเกตโมเด็มที่สร้างขึ้นจำเป็นต้องใช้ขาสัญญาณ RTS (Request to Send) และขาสัญญาณ CTS (Clear to Send) ของ RS-232C ในการควบคุมตัวทรานซิสเตอร์ภายในแพคเกตโมเด็ม ที่ทำหน้าที่ปิด/เปิดสัญญาณในการออกอากาศผ่าน

ทางวิทยุรับ/ส่ง จากการทดลองใช้โปรแกรมการสื่อสารข้อมูลสำเร็จรูปทั่วไป เช่น PROCOMM นั้นจะพบว่า ไม่สามารถใช้ขาสัญญาณทั้งสองในการควบคุมได้ ดังนั้นจึงจำเป็นต้องเขียนโปรแกรมการสื่อสารข้อมูลขึ้นมาใหม่ ซึ่งสามารถควบคุมลักษณะสัญญาณที่ขาสัญญาณทั้งสองได้ โดยภาษาที่ใช้ในการเขียนโปรแกรมคือ ภาษาปาสคาล

วิธีการทดลอง

ติดตั้งสายเชื่อมต่อ RS 232 ระหว่างไมโครคอมพิวเตอร์ และบอร์ดไมโครโปรเซสเซอร์จากนั้นใช้โปรแกรมการสื่อสารข้อมูลที่เขียนขึ้นมาใหม่โดยโหลดโปรแกรมให้กับไมโครคอมพิวเตอร์และบอร์ดไมโครโปรเซสเซอร์ ซึ่งจะต้องตั้งพารามิเตอร์ในการรับส่งข้อมูลให้ตรงกัน ใช้อัตราการรับส่งข้อมูลที่ความเร็ว 1200 บิตต่อวินาที ทำการทดสอบโปรแกรมทีละขั้นตอนตามโฟล์วชาร์ตที่เขียนขึ้น

ผลการทดลอง

โปรแกรมที่เขียนขึ้นทำงานได้ดีตามขั้นตอนที่กำหนดไว้ สามารถที่จะติดต่อสื่อสารข้อมูลระหว่างไมโครคอมพิวเตอร์และบอร์ดไมโครโปรเซสเซอร์ได้

4.2 การทดสอบการทำงานของแพคเกตโมเด็มและบอร์ดไมโครโปรเซสเซอร์

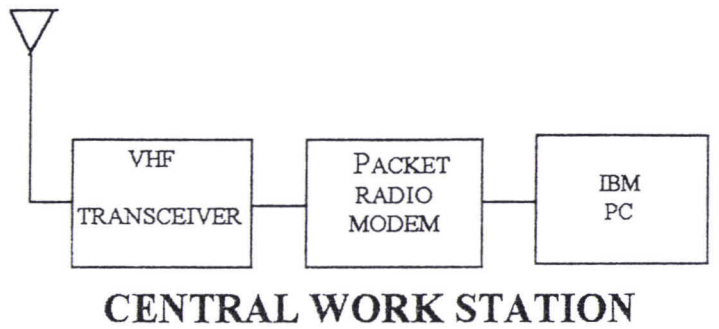
อุปกรณ์ที่ใช้ในการทดลองมีดังต่อไปนี้

1. เครื่องไมโครคอมพิวเตอร์ 1 ชุด
2. แพคเกตโมเด็ม 1 ชุด
3. เครื่องรีพีทเตอร์ 1 ชุด
   - บอร์ดไมโครโปรเซสเซอร์
   - แพคเกตโมเด็ม
4. วิทยุรับ/ส่ง 2 ชุด
5. สายอากาศ 2 ชุด
6. สายนำสัญญาณ 2 ชุด
7. โปรแกรมการสื่อสารข้อมูล 1 ชุด

4.2.1 การทดสอบความเร็วในการรับส่งข้อมูลของแพคเก็ตโมเด็ม

วิธีการทดลอง

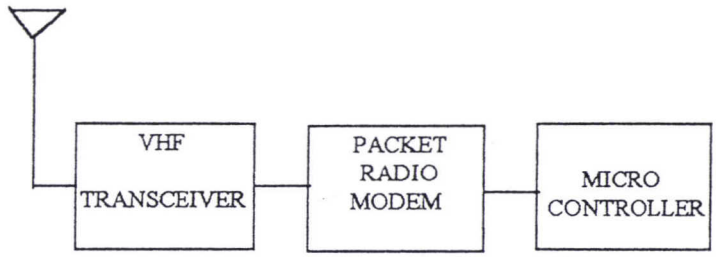
ให้ต่ออุปกรณ์ทางด้านไมโครคอมพิวเตอร์ ดังรูปที่ 4.1 และรูปที่ 4.2



รูปที่ 4.1 แสดงแผนภาพการต่ออุปกรณ์ทางด้านไมโครคอมพิวเตอร์

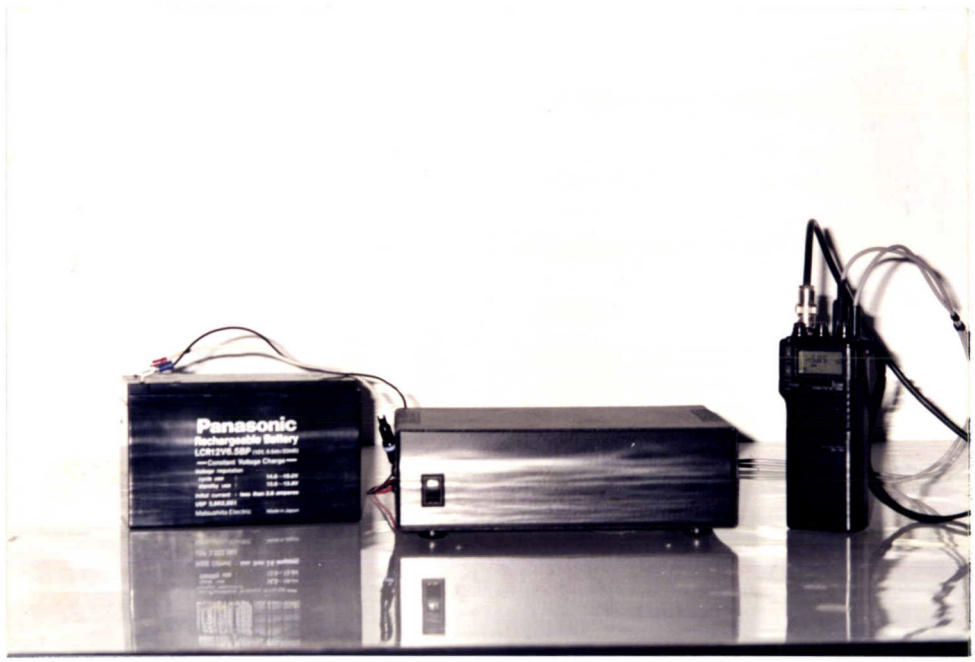
รูปที่ 4.2 แสดงรูปภาพการต่ออุปกรณ์ทางด้านไมโครคอมพิวเตอร์

ให้ต่ออุปกรณ์ทางด้านรีพีทเตอร์ ดังรูปที่ 4.3 และรูปที่ 4.4



REPEATER

รูปที่ 4.3 แสดงแผนภาพการต่ออุปกรณ์ทางด้านรีพีทเตอร์



รูปที่ 4.4 แสดงรูปภาพการต่ออุปกรณ์ทางด้านรีพีทเตอร์

จากนั้นใช้โปรแกรมการสื่อสารข้อมูลที่เขียนขึ้นมาใหม่ โดยโหลดโปรแกรมให้กับไมโครคอมพิวเตอร์และบอร์ดไมโครโปรเซสเซอร์ ซึ่งจะต้องตั้งพารามิเตอร์ในการรับส่งข้อมูลให้ตรงกัน ในที่นี้ทดลองใช้อัตราการรับส่งข้อมูลที่ความเร็ว 1200 บิตต่อวินาที และความเร็ว 2400 บิตต่อวินาที

ผลการทดลอง

สามารถทำการติดต่อสื่อสารระหว่างไมโครคอมพิวเตอร์และรีพีทเตอร์ได้ โดยมีข้อมูลของคำสั่งที่ส่งออกไปจากไมโครคอมพิวเตอร์ ทวนสัญญาณกลับมาจากรีพีทเตอร์ดังรูปที่ 4.5

Data To Receive

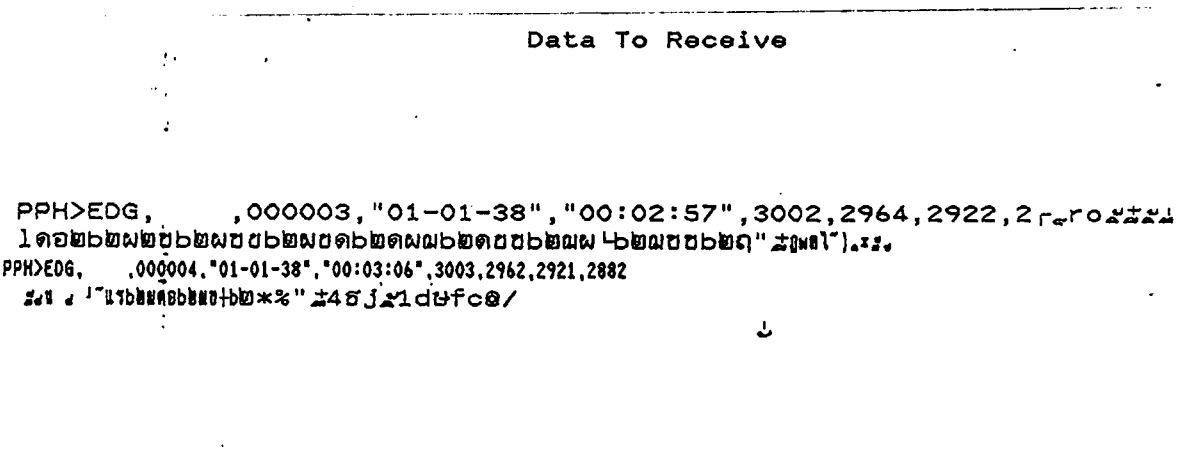
```
PPH>EDG,     ,000009,"01-01-38","00:04:28",2993,2957,2919,2882,2832,2801,272
9,2727,2751,2822,2954,3081,3054,3038,2977,2944,"19"

PPH>EDG,     ,000010,"01-01-38","00:04:36",2993,2953,2915,2877,2828,2796,272
5,2723,2751,2822,2952,3080,3054,3039,2978,2945,"1C"
```

รูปที่ 4.5 แสดงข้อมูลที่รับได้จากเครื่องไมโครคอมพิวเตอร์

ต่อมาทำการเปลี่ยนแปลงอัตราการรับส่งข้อมูลเป็น 2400 บิตต่อวินาที ปรากฏว่าโมเด็มไม่สามารถทำการถอดรหัสข้อมูลได้ จึงไม่มีข้อมูลปรากฏที่จอมอนิเตอร์ของไมโครคอมพิวเตอร์

4.2.2 การทดสอบความถูกต้องของข้อมูลเมื่อมีสัญญาณรบกวน

ในขณะที่มีการติดต่อสื่อสารระหว่างไมโครคอมพิวเตอร์ และรีพีทเตอร์ผ่านทางคลื่นวิทยุ ได้ทดลองใช้เครื่องวิทยุรับส่งอีกเครื่องทำการส่งสัญญาณออกอากาศด้วย ปรากฏว่าข้อมูลที่ปรากฏบนจอมอนิเตอร์ของเครื่องไมโครคอมพิวเตอร์มีการผิดพลาดดังรูปที่ 4.6



รูปที่ 4.6 แสดงข้อมูลที่ได้รับจากเครื่องไมโครคอมพิวเตอร์เมื่อมีสัญญาณรบกวน

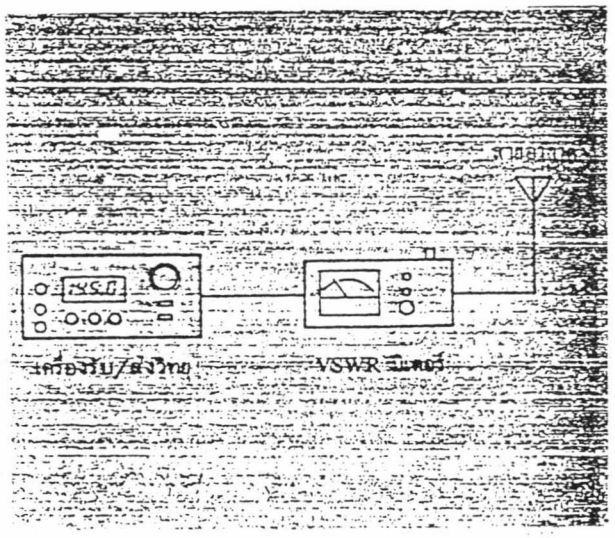
4.3 การทดสอบการทำงานของระบบสายอากาศ

อุปกรณ์ที่ใช้ในการทดลองมีดังต่อไปนี้

1. สายอากาศไดโพล 2 สแต็คที่สร้างขึ้น 1 ชุด
2. สายนำสัญญาณ 1 ชุด
3. วิทยุรับส่ง 1 เครื่อง
4. VSWR METER 1 เครื่อง

วิธีการทดลอง

ให้ต่อุปกรณ์ดังรูปที่ 4.7 จากนั้นทำการวัดค่า VSWR ที่ความถี่ใช้งาน 144-146 MHz.



รูปที่ 4.7 แสดงการต่ออุปกรณ์ในการวัดค่า VSWR

ผลการทดลอง

หลังจากทดลองสร้างสายอากาศคอลลิเนียร์แบบ 2 สแต็ค สำหรับใช้ในย่านความถี่ 144-146 MHz. (ความถี่ย่านความยาวคลื่น 2 เมตร ของนักวิทยุสมัครเล่น) ปรากฏว่าได้ผลเป็นที่น่าพอใจ ค่า VSWR ค่อนข้างคงที่มากตลอดย่านความถี่นี้ ดังตารางที่ 4.1

ความถี่ (MHz)	ค่า VSWR
144.00	1.35
144.25	1.35
144.50	1.30
144.75	1.30
145.00	1.25
145.25	1.20
145.50	1.20
145.75	1.25
146.00	1.30

ตารางที่ 4.1 แสดงค่า VSWR ของสายอากาศที่สร้างขึ้น

4.4 การทดสอบประสิทธิภาพและระยะทางการติดต่อสื่อสาร

อุปกรณ์ที่ใช้ในการทดลองมีดังต่อไปนี้

1. สถานีตรวจวัด Analog Zender 1 ชุด
   - สายอากาศ Slide 1/2 λ
   - แผงวงจรและแพคเก็ตโมเด็ม
   - วิทยุรับ/ส่ง
   - แบตเตอรี่

2. สถานีรีพีทเตอร์ 1 ชุด
   - สายอากาศไดโพล 2 สแต็ค
   - เครื่องรีพีทเตอร์
   - วิทยุรับ/ส่ง
   - สายนำสัญญาณ
   - แบตเตอรี่

3. สถานีไมโครคอมพิวเตอร์ 1 ชุด
   - สายอากาศไดโพล 2 สแต็ค
   - แพคเก็ตโมเด็ม
   - วิทยุรับ/ส่ง
   - สายนำสัญญาณ

วิธีการทดลอง

จัดการทดลองดังรูปที่ 4.8 โดยติดตั้งให้สายอากาศของแต่ละสถานีอยู่สูงจากพื้นดินดังนี้

สถานีไมโครคอมพิวเตอร์	5 เมตร
สถานีรีพีทเตอร์	25 เมตร
สถานีตรวจวัด Analog Zender	0.5 เมตร

ในการทดลอง กำหนดให้สถานีไมโครคอมพิวเตอร์อยู่ห่างจากสถานีรีพีทเตอร์ เป็นระยะทาง 8 กิโลเมตร และสถานีตรวจวัด Analog Zender จะเคลื่อนที่ออกห่างจากสถานีรีพีทเตอร์เป็นระยะทางช่วงละ 5 กิโลเมตร แล้วทำการทดลองติดต่อสื่อสารจากสถานีไมโครคอมพิวเตอร์ไปยังสถานีตรวจวัด Analog Zender โดยติดต่อผ่านสถานีรีพีทเตอร์

ผลการทดลอง

ระยะห่างจาก Repeater (km)	ผลการติดต่อสื่อสาร
5	ติดต่อได้ดี
10	ติดต่อได้ดี
15	ติดต่อได้ดี
20	ติดต่อได้ลำบาก
25	ติดต่อไม่ได้
30	ติดต่อไม่ได้

ตารางที่ 4.2 แสดงผลการติดต่อสื่อสารระหว่างสถานีไมโครคอมพิวเตอร์ สถานีรีพีทเตอร์และสถานีตัววัด Analog Zender

บทที่ 5

สรุปและข้อเสนอแนะ

อุปกรณ์ แพคเกตโมเด็ม คือ อุปกรณ์ที่ใช้สำหรับการติดต่อสื่อสารข้อมูลดิจิตอลผ่านทางสัญญาณวิทยุ ซึ่งให้ข้อดีกว่าการติดต่อสื่อสารในระบบดิจิตอลผ่านทางสายนำสัญญาณ เพราะว่าสามารถติดต่อกับอุปกรณ์อีกตัวหนึ่งที่บริเวณใดก็ได้ แต่อาจมีปัญหาจากสัญญาณรบกวนที่มาจากภายนอกทำให้ข้อมูลเกิดการผิดพลาดและสูญหาย ซึ่งเป็นข้อเสียส่วนหนึ่งของระบบนี้

สำหรับอุปกรณ์ รีพีทเตอร์ที่ได้สร้างขึ้น เพื่อใช้ในการถ่ายทอดสัญญาณระหว่าง 2 สถานีที่ไม่สามารถติดต่อสื่อสารกันได้นั้น ได้ผลเป็นที่น่าพอใจคือ สามารถส่งข้อมูลในการติดต่อได้ถูกต้องและช่วยเพิ่มระยะทางในการติดต่อสื่อสารให้ไกลมากขึ้นโดยระยะทางการติดต่อสื่อสารนั้นจะคำนวณค่าได้จาก สูตรการหาระยะทางการติดต่อสื่อสาร ซึ่งใช้หลักการเส้นขอบคลื่นวิทยุโดยถือว่าไม่มีสิ่งกีดขวางอยู่ในเส้นทางผ่าน ถ้ามีสิ่งกีดขวางระยะการติดต่ออาจใกล้หรือไกลกว่านี้ในทางปฏิบัติ มักจะพบว่าเราสามารถติดต่อได้ไกลเกินระยะเส้นขอบคลื่นวิทยุที่คำนวณได้ โดยไม่ต้องอาศัยสภาพบรรยากาศพิเศษมาช่วย (เช่น เกิด tropospheric ducting,sporadic E เป็นต้น) แต่อาศัยสิ่งที่สามารถควบคุมได้มาช่วยเพิ่มระยะการติดต่อ เช่น ความไวของเครื่องรับ กำลังส่งของเครื่องส่ง อัตราขยายของสายอากาศรับและสายอากาศส่ง การสูญเสียของสายนำสัญญาณเป็นต้น ซึ่งแม้จะไม่สามารถทำการติดต่อได้ 100% ตลอดเวลา แต่ก็อาจเพียงพอสำหรับการใช้งาน เช่น 90%สำหรับการใช้งานทั่วไป และ 50% อาจเพียงพอสำหรับกิจการวิทยุสมัครเล่นเป็นต้น ค่า k นี้จึงมีประโยชน์ในการคำนวณหาความสูงขั้นต่ำของสายอากาศ เพื่อให้สามารถติดต่อสื่อสารได้ไม่น้อยกว่าที่ต้องการและเป็นค่าที่สำคัญที่สุดค่าหนึ่งในการวางแผนเส้นทางการแพร่กระจายคลื่นวิทยุ

ปัญหา

ปัญหาที่เกิดขึ้นในการทดลอง คือ เมื่อมีสัญญาณรบกวนจากภายนอกที่ไม่พึงประสงค์เข้ามาจะทำให้รับข้อมูลจากสถานีที่ทำการติดต่ออยู่ไม่ได้ และในการวางวิทยุสื่อสารนั้นไม่สามารถวางไว้ใกล้กับเครื่องคอมพิวเตอร์ได้ เพราะเครื่องคอมพิวเตอร์จะให้กำเนิดสัญญารรบกวนออกมาในย่านความถี่วิทยุ ทำให้ไม่สามารถรับสัญญาณข้อมูลขนาดอ่อนๆ ได้

ข้อเสนอแนะ

การใช้อุปกรณ์ในการสื่อสารข้อมูลผ่านทางคลื่นวิทยุนั้น ควรเลือกช่องสัญญาณที่ไม่มีการติดต่อกัน (เป็นช่องสัญญาณที่ว่าง) เพื่อหลีกเลี่ยงกับการที่ข้อมูลจะสูญหายจากสัญญาณรบกวนภายนอก และไม่ควรนำอุปกรณ์ในการสื่อสารข้อมูลไปไว้กับอุปกรณ์ที่กำเนิดสัญญาณคลื่นวิทยุซึ่งจะมารบกวนระบบได้ หากต้องการติดต่อสื่อสารให้ได้ระยะทางไกลๆ ควรจะติดตั้งสายอากาศให้มีความสูงจากพื้นดินมากๆ

บรรณานุกรม

1. ชมรมวิทยุสมัครเล่น กฟผ. , "PACKET RADIO การสื่อสารข้อมูลผ่านคลื่นวิทยุ" , CQ Amateur Radio ฉบับที่ 54 : 70-74
2. นพดล สหชัยเสรี , "PACKET RADIO สำหรับคนยากฯ" , CQ Amateur Radio ฉบับที่ 55 : 90-93
3. ชาญยุทธ พฤกษ์คุ้มวงศ์ , "เรียนรู้...การสื่อสารข้อมูล" , เซมิคอนดักเตอร์ อิเลคทรอนิกส์ ฉบับที่ 130 (2536) : 95-97
4. ทนง โชติสรยุทธ์ , "สายอากาศคอลลิเนียร์แบบขนานใช้ไดโพล สำหรับ 144 MHz" , เซมิคอนดักเตอร์ อิเลคทรอนิกส์ ฉบับที่ 56 (2526) : 128-140
5. นิพนธ์ ศิริรัตน์ , "MCS-51 กับการติดต่อสื่อสารทางพอร์ตอนุกรม" , เซมิคอนดักเตอร์ อิเลคทรอนิกส์ ฉบับที่ 120 (2535) : 49-53
6. รศ.ยืน ภู่วรวรรณ , น.ต.ดร.ไพศาล สงวนหมู่ , "การสื่อสารข้อมูลและไมโครคอมพิวเตอร์เนตเวอร์ค" , บริษัท ซีเอ็ดยูเคชั่น จำกัด

ภาคผนวก

- โปรแกรมการสื่อสารข้อมูลที่ใช้กับ Repeater Station

```
;*************************************************************
;                  16 CHANNELS ANALOG SENDER
;
;       DESIGNED  BY   : WICHIT  SIRICHOTE       (HS1HQN)
;                      : 19 MAY 1994
;       Last Update    : SUJIT    UARPANYAPORN   (HS1EDG)
;                      : PHICHAI  SRISARIST      (E21PPH)
;                      : 27 FEB 1995
;*************************************************************
;
;

          CPU      "8051.TBL"
          HOF      "INT8"
;
;-------------------------------------------------------------
;                  MCS-51 INTERNAL REGISTERS
;-------------------------------------------------------------

B:          EQU      0F0H     ;B REGISTER
ACC:        EQU      0E0H     ;ACCUMULATOR
PSW:        EQU      0D0H     ;PROGRAM STATUS WORD
IPC:        EQU      0B8H     ;INTERRUPT PRIORITY
P3:         EQU      0B0H     ;PORT 3  P3.4-P3.7 BOARD ADDRESS
IEC:        EQU      0A8H     ;INTERRUPT ENABLE
P2:         EQU      0A0H     ;PORT 2
SBUF:       EQU       99H     ;SEND BUFFER
SCON:       EQU       98H     ;SERIAL CONTROL
P1:         EQU       90H     ;PORT 1
TH1:        EQU       8DH     ;TIMER 1 HIGH
TH0:        EQU       8CH     ;TIMER 0 HIGH
TL1:        EQU       8BH     ;TIMER 1 LOW
TL0:        EQU       8AH     ;TIMER 0 LOW
TMOD:       EQU       89H     ;TIMER MODE
TCON:       EQU       88H     ;TIMER CONTROL
PCON:       EQU       87H     ;POWER CONTROL REGISTER
DPH:        EQU       83H     ;DATA POINTER HIGH
DPL:        EQU       82H     ;DATA POINTER LOW
SP:         EQU       81H     ;STACK POINTER

P0:         EQU       80H     ;PORT 0 B1-B8, B9-B12,OR,POL 7109

;-------------------------------------------------------------
;               MCS-51 INTERNAL BIT ADDRESSES
;-------------------------------------------------------------

CY:         EQU      0D7H     ;CARRY FLAG
AC:         EQU      0D6H     ;AUXILIARY-CARRY FLAG
F0:         EQU      0D5H     ;USER FLAG 0
RS1:        EQU      0D4H     ;REGISTER SELECT MSB
RS0:        EQU      0D3H     ;REGISTER SELECT LSB
OV:         EQU      0D2H     ;OVERFLOW FLAG
P:          EQU      0D0H     ;PARITY FLAG
PS:         EQU      0BCH     ;PRIORITY SERIAL PORT
PT1:        EQU      0BBH     ;PRIORITY TIMER 1
PX1:        EQU      0BAH     ;PRIORITY EXTERNAL 1
PT0:        EQU      0B9H     ;PRIORITY TIMER 0
```

```
PX0:        EQU     0B8H    ;PRIORITY EXTERNAL 0
EA:         EQU     0AFH    ;ENABLE ALL INTERRUPT
ES:         EQU     0ACH    ;ENABLE SERIAL INTERRUPT
ET1:        EQU     0ABH    ;ENABLE TIMER 1 INTERRUPT
EX1:        EQU     0AAH    ;ENABLE EXTERNAL 1 INTERR
ET0:        EQU     0A9H    ;ENABLE TIMER 0 INTERRUPT
EX0:        EQU     0A8H    ;ENABLE EXTERNAL 0 INTERR
SM0:        EQU     09FH    ;SERIAL MODE 0
SM1:        EQU     09EH    ;SERIAL MODE 1
SM2:        EQU     09DH    ;SERIAL MODE 2
REN:        EQU     09CH    ;SERIAL RECEPTION ENABLE
TB8:        EQU     09BH    ;TRANSMITT BIT 8
RB8:        EQU     09AH    ;RECEIVE BIT 8
TI:         EQU     099H    ;TRANSMIT INTERRUPT FLAG
RI:         EQU     098H    ;RECEIVE INTERRUPT FLAG
TF1:        EQU     08FH    ;TIMER 1 OVERFLOW FLAG
TR1:        EQU     08EH    ;TIMER 1 RUN CONTROL BIT
TF0:        EQU     08DH    ;TIMER 0 OVERFLOW FLAG
TR0:        EQU     08CH    ;TIMER 0 RUN CONTROL BIT
IE1:        EQU     08BH    ;EXT INTERR. 1 EDGE FLAG
IT1:        EQU     08AH    ;EXT INTERR. 1 TYPE FLAG
IE0:        EQU     089H    ;EXT INTERR. 0 EDGE FLAG
IT0:        EQU     088H    ;EXT INTERR. 0 TYPE FLAG

;-------------------------------------------------------------
;                    PORT 1 BIT ADDRESS
;-------------------------------------------------------------

P1.0:       EQU     090H    ;P1.0-P1.3 OUTPUT TO SELECT
                            ;MUX CHANNEL
P1.1:       EQU     091H
P1.2:       EQU     092H    ;PTT key  H=Tx   , L= Rx
P1.3:       EQU     093H    ;
P1.4:       EQU     094H    ;RUN/HOLD 7109
P1.5:       EQU     095H    ;CE/LOAD
P1.6:       EQU     096H    ;HBEN
P1.7:       EQU     097H    ;LBEN

P2.0:       EQU     0A0H    ;19200/9600 BIT/SEC SELECT
                            ;H=19200,L=9600
P2.1:       EQU     0A1H    ;FREE RUNNING
P2.2:       EQU     0A2H
P2.3:       EQU     0A3H    ;POLARITY SELECT
P2.4:       EQU     0A4H    ;MAXIMUM CHANNEL TO BE SCANED 1-16
P2.5:       EQU     0A5H    ;P2.4 - P2.7
P2.6:       EQU     0A6H
P2.7:       EQU     0A7H

;-------------------------------------------------------------
;                      RAM BIT ADDRESS
;-------------------------------------------------------------

KEY:            EQU     0BH
CHECK_SUM:      EQU     0BH     ;CHECK SUM BIT
LED:            EQU     08H     ;INTERRUPT BLINK
XOFF_FLAG:      EQU     0AH
ZEROING:        EQU     0DH
```

```
WARM:           EQU     55H     ;56H, 57H, 58H

STATION_ADDRESS: EQU    59H     ;SOFTWARE LOG STATION ADDRESS
CALLER_ADDRESS:  EQU    5AH

INPUT_BUFFER:    EQU    57H     ;INPUT CONSOLE ASCII 40-45
                                ;(6 CHARACTERS)

;-------------------------------------------------------------
;                       STRING CONSTANT
;-------------------------------------------------------------

SOH:        EQU     01H     ;START OF HEADER
EOT:        EQU     04H     ;END OF TEXT
NAK:        EQU     15h     ;NO ACKNOWLEDGE
ACK:        EQU     06h     ;ACKNOWLEDGE
BS:         EQU     08H
FS:         EQU     0CH
CR:         EQU     0DH
LF:         EQU     0AH
EOS:        EQU     10H
BELL:       EQU     07H
SPACE:      EQU     20H
COMMA:      EQU     2Ch

SIN_BUFFER: EQU     01F00H

;-------------------------------------------------------------
;                       INTERUPT ADDRESS
;-------------------------------------------------------------
                        ORG     0000H
        LJMP        MAIN_TIMER

                        ORG     0003H
        LJMP        READ_7109   ;SERVICE EX0

                        ORG     000BH
        LJMP        SERVICE_T0  ;TIMER COUNTER 0 INTERRUPT
                                ;SET TO HIGHEST PIORITY

;*************************************************************
;                       MAIN PROGRAM
;*************************************************************

PORT_A:         EQU     8000H
PORT_B:         EQU     8001H
PORT_C:         EQU     8002H
CONTROL_8255:   EQU     8003H

COUNT_L:        EQU     00H
COUNT_H:        EQU     00H     ;INTERRUPT EVERY
                                ;11.0592E+6/(12*65536) =
                                ;14.0625 Hz

;-------------------------------------------------------------
MAIN_TIMER:     MOV     R0,#10H
POWER_DELAY:    DJNZ    R0,POWER_DELAY
```

```
        MOV SP,#60H             ;FIRST STACK MUST BE INITIALIZED
;       CLR  P1.4               ; HOLD 7109
        CLR  PTT                ; RX MODE
;       MOV  STATION_ADDRESS,#30H
                                ;INITIALISED STATION ADDRESS TO @30
                                ;EG., WICHIT IS 30 YEARS OLD.

;       MOV  A,P2
;       ANL  A,#0F0H        ;READ ONLY HIGH NIBBLE (P2.4-P2.7)
;       SWAP A
;       INC  A              ;GET CHANNEL MAX
;       MOV CHANNEL_MAX,A

;       MOV   A,WARM
;       CJNE  A,#53H,COLD_BOOT
;       SJMP  WARM_BOOT

COLD_BOOT:
;       MOV   TIME_SEC,#10H
;       MOV   TIME_MIN,#00D
;       MOV   SAMPLE,#00D
;       MOV   SAMPLE+1,#00D
;       MOV   SAMPLE+2,#00D

;       MOV   WARM,#53H
        MOV   SEC/50,#00H
        MOV   SEC,#35H
        MOV   MIN,#24H
        MOV   HOUR,#08H
        MOV   DAY,#31H
        MOV   MONTH,#07H
        MOV   YEAR,#37H

;       MOV   WIND_SPEED,#00H
;       MOV   WIND_SPEED+1,#00H
;       MOV   WIND_COUNTER,#00H
;       MOV   WIND_COUNTER+1,#00H

WARM_BOOT:   SETB  EA           ;ENABLE ALL INTERRUPT
        SETB  EX0               ;ENABLE EXTERNAL INTERRUPT 0
        SETB  IT0               ;FALLING EDGE EXT0
        SETB  ET0               ;ENABLE TIMER0 INTERRUPT
        SETB  PT0           ;REAL-TIME CLOCK 0 HIGHEST PIORITY
        MOV   TMOD,#00000001B;SET MODE 1 TO (16 BIT DIVIDER)

;       MOV   CHANNEL,#00H

        LCALL INIT

        MOV   TL0,#COUNT_L
        MOV   TH0,#COUNT_H
        CLR   READY
        CLR   CHECK_SUM

;       CLR  P1.0           ;SELECT CH0
;       CLR  P1.1
```

```
;       CLR  P1.2
;       CLR  P1.3
;       MOV   MUX,#00H
        SETB  TR0            ;START REALTIME CLOCK


;************************* MAIN2 ****************************
;------------------------------------------------------------
;                          BIG-MAIN
;------------------------------------------------------------
        MOV     DPTR,#NAME_RP
        MOV     R1,#90h
        MOV     R0,#00h
        MOV     R2,#03h
        LCALL   COPY0_EX      ;COPY NAME_RP TO RAM ADDR 9000H

MAIN2:  JNB     RI,$
        MOV     A,SBUF
        CLR     RI
        MOV     DPTR,#8000h
        CJNE    A,#01h,MAIN2      ;CHECK START OF HEAD
MAIN_B: MOVX    @DPTR,A
        CLR     RI
        INC     DPTR
        MOV     R0,#02d           ;SET TIMER 2 SEC.
        LCALL   RECEIVE_TIME
        JB      RI,RX_OK2
        LJMP    MAIN2
RX_OK2: MOV     B,A
        CLR     RI
        MOV     A,DPH
        CJNE    A,#84h,RX_OK3
        MOV     A,DPL
        CJNE    A,#0FFh,RX_OK3
        LJMP    MAIN2
RX OK3: MOV     A,B
        CJNE    A,#04h,MAIN_B     ;check end EOT
        MOVX    @DPTR,A
        MOV     R0,30d
        LCALL   B_DELAY
        CLR     RI
        MOV     7Dh,DPH
        MOV     7Eh,DPL

        MOV     DPTR,#8000H       ;CHECK SUM
        LCALL   CHK_SUM
        MOVX    A,@DPTR
        CJNE    A,05h,MAIN2
        INC     DPTR
        MOVX    A,@DPTR
        CJNE    A,04h,MAIN2

        MOV     DPTR,#8009H
        MOV     R3,#80H
        MOV     R2,#0BH
        MOV     R1,#90H
        MOV     R0,#00H
```

```
          LCALL    COM_B              ;CHECK NAME REPEATER
          CJNE     A,#00H,MAIN2

          MOV      DPH,7DH
          MOV      DPL,7EH

          MOV      R3,DPH
          MOV      R2,DPL
          MOV      DPTR,#8000H
          MOV      R1,#85H
          MOV      R0,#00H
          LCALL    COPY2_EX
          CJNE     A,#00H,MAIN2

          MOV      DPTR,#850CH         ;PUSH CONTROL WITH ":"
          MOV      A,#":"
          MOVX     @DPTR,A

          MOV      DPTR,#8500H         ;PUT CHECKSUM
          LCALL    CHK SUM
          MOV      A,R5
          MOVX     @DPTR,A
          INC      DPTR
          MOV      A,R4
          MOVX     @DPTR,A

          MOV      R0,#20d
          LCALL    B_DELAY
          LCALL    PTT ON
          MOV      DPTR,#8500h
          LCALL    SEND_BLOCK
          LCALL    PTT_OFF
          LJMP     MAIN2
;-------------------------------------------------------------------
MAIN3:    ;LCALL   SEND_PROMPT

MAIN4:    SETB     P1.4                ;RUN 7109 DURING WAIT
          JNB      RI,MAIN4
          CLR      RI
          MOV      A,SBUF
          ORL      A,#00100000B        ;IGNOR UPPER-CASE

COM2:     CJNE    A,#"g",COM3
          LJMP    TEST10              ;RUN ZENDER

COM3:     CJNE    A,#"c",COM4
          LJMP    SET_DATE

COM4:     CJNE    A,#"s",COM5
          LJMP    SEC_DELAY

COM5:     CJNE    A,#"+",COM6
          NOP
COM6:     CJNE    A,#"-",COM7
          NOP

COM7:     CJNE    A,#"l",COM8
```

```
        NOP

COM8:   CJNE    A,#"u",COM9
        NOP

COM9:   CJNE    A,#"m",COM10
        LJMP    MIN_DELAY

COM10:  CJNE    A,#"?",COM11
        LJMP    HELP_COMMAND

COM11:  CJNE    A,#"r",COM12
        LJMP    READ_TIME_SET

COM12:  CJNE    A,#"b",COM13
        LJMP    SET_BAUD

COM13:  CJNE    A,#"d",COM14
        LJMP    DISPLAY_DATE_TIME

COM14:  CJNE    A,#"n",COM15
        LJMP    NEW_FILE

COM15:  CJNE    A,#"t",COM16
        LJMP    TRIGGER SEND

COM16:  CJNE    A,#"p",COM17
        LJMP    RUN

COM17:  CJNE    A,#"e",COM18
        LJMP    SET_TIME

COM18:  CJNE    A,#cr,COM19
        LCALL   SEND
        LJMP    MAIN3

COM19:  CJNE    A,#"a",COM20
        LJMP    LOG_ADDRESS

COM20:  CJNE    A,#".",COM21      ;RESTART ZENDER
        LJMP    MAIN_TIMER

COM21:  CJNE    A,#60H,COM22      ;eg. @ COMMAND 40H OR 20H
        LJMP    GET_ADDRESS

COM22:  LJMP    MAIN3


MENU:   MOV DPTR,#TITLE
        LCALL SEND_STRING
MENU3:  RET

;ENTER_COM ENTER CURRENT TIME AND DAY

ENTER_COM:
        MOV     DPTR,#ENTER_DATE
        LCALL   SEND_STRING
```

```
          LCALL    GET_WORD
          LCALL    GET_RANGE
          MOV      A,UP_START_L
          MOV      MONTH,A
          MOV      A,UP_START_H
          MOV      DAY,A
          MOV      A,UP_STOP_L
          MOV      YEAR,A

          MOV      DPTR,#ENTER_TIME
          LCALL    SEND_STRING
          LCALL    GET_WORD
          LCALL    GET_RANGE
          MOV      A,UP_START_L
          MOV      HOUR,A
          MOV      A,UP_STOP_L
          MOV      MIN,A
          LJMP     MAIN2

;-------------------------------------------------------------
;                        SUB ROUTINE
;-------------------------------------------------------------

;INIT_RS232C INITIALIZED 8051 SERIAL PORT
;=========== BAUD RATE: 19200 bit/sec OR 9600 BIT/SEC
;            DATA LENGTH: 8 BIT
;            STOP BIT: 1 BIT
;            PARITY: NO
;            X-TAL: 11.0592 MHz
;
;           ENTRY : P2.0, IF P1.4 = H THEN BAUD RATE=1200
;                         IF P1.4 = 0 THEN BAUD RATE= 600

INIT:       MOV      TMOD,#00100001B
            MOV      SCON,#01010010B
            SJMP     HIGH_BAUD
            MOV      PCON,#00000000B
            SJMP     LOW_BAUD
HIGH_BAUD:  MOV      PCON,#10000000b     ;DOUBLE BOARD RATE
LOW_BAUD:   MOV      TH1,#0D0h           ;TIMER1 LOAD VALUE
            SETB     TR1                 ;START TIMER1
            RET

;PTT ON-OFF
;==========
PTT_ON:     SETB     PTT
            MOV      R0,#120D            ;DELAY 1 SEC.
            LCALL    B_DELAY
            RET

PTT_OFF:    CLR      PTT
            RET

;SEND     SEND A TO SERIAL PORT
;         CHECK TRANSMITTER BUFFER BEFORE SEND
;         if check_sum bit = 1 then calculate check sum byte
;         ENTRY: A
```

```
;         EXIT: A
;
SEND:     CLR       TI
          MOV       SBUF,A        ;OK BUFFER EMPTY SEND OUT...
          JNB       TI,$
          JNB       CHECK_SUM,NO_CHECK_SUM
          PUSH      ACC
          ADD       A,BCC
          MOV       BCC,A
          POP       ACC

NO_CHECK_SUM:

          RET
;
;RECIVES RECIVE BYTE FROM SERIAL PORT WITH TIME-OUT FUNCTION
;=============================================================
;         ENTRY: NO
;         EXIT: A = ASCII NUMBER OR STRING COMMAND
;                                        A = FF  TIME-OUT

RECIVES:  MOV       R7,#05H
AGAIN:    MOV       R6,#00H          ; TIME-OUT DELAY
GET_RI:   JB        RI,READYS

          DJNZ      R6,GET_RI
          DJNZ      R7,AGAIN
          MOV       A,#0FFH          ; TIME-OUT NO DATA RECEPTED
          RET
READYS:   CLR       RI
          MOV       A,SBUF
          RET


;RECIVE  RECIVE BYTE PUT TO A FROM SERIAL PORT
;=============================================
;         WAIT UNTIL RECIVER BUFFER READY
;         IF RECEIVE CHARACTER "!" RESET SYSTEM BY START
;            EXECUTE AT $0000
;         ENTRY: NO
;         EXIT: A

RECIVE:   JNB       RI,RECIVE
          CLR       RI
          MOV       A,SBUF
          CJNE      A,#"!",NORMAL
          LJMP      MAIN_TIMER

NORMAL:
          RET

;RECEIVE RECEIVE BYTE NO ECHO
;============================

RECEIVE:  JNB       RI,RECEIVE
          CLR       RI
          MOV       A,SBUF
```

```
            RET

;CR_SEND SEND CR TO RS232
;========================
;          ENTRY: NO
;          EXIT:  NO
;
SEND_CR:   MOV      A,#CR
           LCALL    SEND
SEND_LF:   MOV      A,#LF
           LCALL    SEND
           RET

;SEND_COMMA
;==========

SEND COMMA:   MOV     A,#","
              LCALL   SEND
              RET

;SEND_22H
;========

SEND_22H:     MOV     A,#22H
              LCALL   SEND
              RET

;SEND_SPACE
;==========

SEND_SPACE:   MOV     A,#SPACE
              LCALL   SEND
              RET

XOFF:   EQU   13H
XON:    EQU   11H
BREAK:  EQU   00H


;XOFF_SEND SEND XOFF TO RS232
;============================
;       ENTRY: NO
;       EXIT:  NO
;
XOFF_SEND: MOV      A,#XOFF     ;GET XOFF CHARACTER
           LCALL    SEND
           RET

;XON_SEND
;
;
;
XON_SEND:  MOV       A,#XON
           LCALL     SEND
           RET
```

```
;NIBBLE_SHIFT SHIFT FOUR DIGIT BCD NUMBER LEFT A NIBBLE
;=====================================================
;            ENTRY : R0 POINTED TO LSD
;            EXIT  : NO

NIBBLE_SHIFT:  INC   R0
               MOV   A,@R0
               SWAP  A
               ANL   A,#0F0H
               MOV   R3,A
               DEC   R0
               MOV   A,@R0
               SWAP  A
               MOV   R2,A
               ANL   A,#0FH
               ADD   A,R3
               INC   R0
               MOV   @R0,A
               MOV   A,R2
               ANL   A,#0F0H
               DEC   R0
               MOV   @R0,A
               RET

;SHIFT_16BIT SHIFT LEFT 1 BIT ADDRESS $4D,$4E
;============================================
;             $4D  LO BYTE
;             $4E  HI BYTE
;             ENTRY: NO
;             EXIT: Carry Flag

SHIFT_16BIT: CLR    C           ;     {($4E,$4D)} X 2
          MOV    A,5DH
          ADD    A,5DH
          MOV    5DH,A
          MOV    A,5EH
          ADDC   A,5EH
          MOV    5EH,A
          RET                   ; CARRY GONE

;BIN_BCD CONVERT 16 BIT BINARY TO BCD
;====================================
;         SHIFT LEFT 1 BIT FROM MOST SIGNIFICANT BIT TO LEAST
;           SIGNIFICANT
;         BIT , IF CARRY = 1 THEN BCD= (2^N + BCD)
;
;         ENTRY: 16 BIT DATA
;
;
;         5D  4D  $3D = LO BYTE
;         5E  4E  $3E = HI BYTE
;         EXIT: 6 DIGIT BCD
;             46 $36L =      0
;                $36H =     10
;             47 $37L =    100
;                $37H =   1000
;             48 $38L =  10000
```

```
;                   $38H = 100000

BIN_BCD:    MOV   56H,#00       ; CLEAR BCD BEFORE
            MOV      57H,#00
            MOV      58H,#00

            MOV      R1,#00H
            MOV      R7,#16D
BIN_1:      LCALL    SHIFT_16BIT
            JNC      NEXT_BIT

;       OK CARRY = 1 ADD BCD TO 2^N CONSTANT IN TABLE2
            MOV      DPTR,#TABLE2
            CLR      C
            MOV      R0,#56H
            MOV      R6,#03H
BIN 2:      MOV      A,R1
            MOVC     A,@A+DPTR
            ADDC      A,@R0
            DAA
            MOV      @R0,A
            INC      DPTR
            INC      R0
            DJNZ     R6,BIN_2
NEXT_BIT:   INC      R1
            INC      R1
            INC      R1
            DJNZ     R7,BIN_1
            RET

TABLE2:     DFB      68H,27H,03H
            DFB      84H,63H,01H
            DFB      92H,81H,00H
            DFB      96H,40H,00H
            DFB      48H,20H,00H
            DFB      24H,10H,00H
            DFB      12H,05H,00H
            DFB      56H,02H,00H
            DFB      28H,01H,00H
            DFB      64H,00H,00H
            DFB      32H,00H,00H
            DFB      16H,00H,00H
            DFB      08H,00H,00H
            DFB      04H,00H,00H
            DFB      02H,00H,00H
            DFB      01H,00H,00H

;#############################################################
;             TIMER 0 INTERRUPT SERVICE ROUTINE               #
;            enter to this routine every 0.01 SECOND          #
;              update current time and date                   #
;                                                             #
;                                                             #
;#############################################################

SERVICE_T0:
          PUSH     PSW
```

```
          PUSH    ACC
          PUSH    B
          PUSH    DPL
          PUSH    DPH
          MOV     PSW.#00010000B  ;SELECT REGISTER BANK 2

; TIMER 0 SERVICE BODY


          MOV     TH0,#COUNT_H
          MOV     TL0,#COUNT_L
          INC     SEC100
          MOV     A,SEC100
          CJNE    A,#14D,CHECK_SEC
          MOV     SEC100,#00H

          MOV     A,SEC
          ADD     A,#01D
          DAA
          MOV     SEC,A

          MOV     A,TIME_SEC        ;IF USER SET =00 THEN IGNOR
                                    ;SETB READY
          CJNE    A,#00D,CHECK_SEC_TIMER
                                    ;ELS INCREMENT AND CHECK
          SJMP    CHECK_SEC

CHECK_SEC_TIMER:
          MOV     A,TRIG_SEC
          ADD     A,#01D
          DAA
          MOV     TRIG_SEC,A
          CJNE    A,TIME_SEC,CHECK_SEC    ;COMPARE CURRENT
                                          ;TIMER AND USER SET
          MOV     TRIG_SEC,#00D
          SETB    READY


CHECK_SEC:
          MOV     A,SEC
          CJNE    A,#60H,CHECK_MIN

          MOV     SEC,#00D
          MOV     A,MIN
          ADD     A,#01D
          DAA
          MOV     MIN,A


          MOV     A,TIME_MIN      ;INCREMENT TRIGGER MIN TIMER
          CJNE    A,#00D,CHECK_MIN_TIMER
          SJMP    CHECK_MIN

CHECK_MIN_TIMER:
          MOV     A,TRIG_MIN
          ADD     A,#01D
          DAA
```

```
            MOV     TRIG_MIN,A
            CJNE    A,TIME_MIN,CHECK_MIN
            MOV     TRIG_MIN,#00D
            SETB    READY           ;INDICATE READY TO SEND DATA

CHECK MIN:
            MOV     A,MIN
            CJNE    A,#60H,CHECK_HOUR
            MOV     MIN,#00D
            MOV     A,HOUR
            ADD     A,#01D
            DAA
            MOV     HOUR,A

CHECK_HOUR:
            MOV     A,HOUR
            CJNE    A,#24H,CHECK_DAY
            MOV     HOUR,#00D
            MOV     A,DAY
            ADD     A,#01D
            DAA
            MOV     DAY,A

CHECK_DAY:
            MOV     A,DAY
            CJNE    A,#31H,CHECK_MONTH
            MOV     DAY,#01D
            MOV     A,MONTH
            ADD     A,#01D
            DAA
            MOV     MONTH,A

CHECK_MONTH:
            MOV     A,MONTH
            CJNE    A,#12H,CHECK_YEAR
            MOV     MONTH,#01D
            MOV     A,YEAR
            ADD     A,#01D
            DAA
            MOV     YEAR,A

CHECK_YEAR:


            POP     DPH
            POP     DPL
            POP     B
            POP     ACC
            POP     PSW
            RETI


;TIMER_DELAY
;           set bit ready when delay timer is equal set table
;           read from p3.4 - p3.7
;           entry: 4 bit at p3.4 -p3.7
;           exit; ready flag
```

```
;#################################################################
;                  INTERRUPT SERVICE ROUTINE EX0                  #
;               READ DATA FROM ICL7109 12 BIT ADC                 #
;                                                                 #
;                                                                 #
;#################################################################

READ 7109:
          PUSH    PSW
          PUSH    ACC
          PUSH    B
          PUSH    DPL
          PUSH    DPH
          MOV     PSW,#00010000B  ;SELECT REGISTER BANK 2
          CLR     P1.4            ;HOLD 7109

          MOV     R0,#ADC_DATA  ;COMPUTE ADDRESS OF ADC BUFFER

          MOV     A,MUX
          MOV     B,#02H
          MUL     AB
          ADD     A,R0
          MOV     R0,A

          INC     MUX
          MOV     A,MUX
          CJNE    A,CHANNEL_MAX,NEXT_CHANNEL
                        ;CHANGE 10 TO VARIABLE IN CHANNEL_MAX
          MOV     MUX,#00H
          SETB    F0                ;16 CHANNEL FINISHED

NEXT_CHANNEL:

          MOV     C,00H
          MOV     P1.0,C
          MOV     C,01H
          MOV     P1.1,C
          MOV     C,02H
          MOV     P1.2,C
          MOV     C,03H
          MOV     P1.3,C

          CLR     P1.5            ; CE --> 0
          CLR     P1.7            ; LBEN -> 0
          MOV     P0,#0FFH
          NOP
          MOV     A,P0

          MOV     @R0,A
          SETB    P1.7            ; LBEN -> 1
          NOP
          CLR     P1.6            ; HBEN -> 0
          MOV     P0,#0FFH
          NOP
          MOV     A,P0

          INC     R0
```

```
          MOV     @R0,A             ;SAVE TO @R0+1
;         SETB    P1.5              ;RESTORE CE
          SETB    P1.6              ;HBEN -> 1

          POP     DPH
          POP     DPL
          POP     B
          POP     ACC
          POP     PSW
          RETI

;SEND_7109_SERIAL
;         SEND PACKET DIGITAL DATA TO PC VIA SERIAL PORT
;         DESTROY: R0,R3


SEND_7109_SERIAL:
          LCALL   PTT_ON
          MOV     BCC,#00H          ;CLEAR BCC BYTE
          SETB    CHECK_SUM         ;NEEDED FOR SEND ROUTINE TO
                                    ;CALCULATE CHECK SUM

          MOV     A,#SOH            ;START WITH START OF HEAD
          LCALL   SEND
          LCALL   SEND_CR

          MOV     A,#"@"            ;NEXT SEND CONNECTION ADDRESS
          LCALL   SEND
          MOV     A,STATION_ADDRESS
          LCALL   SEND_ASCII
          MOV     A,#">"
          LCALL   SEND
          MOV     A,#">"
          LCALL   SEND
          MOV     A,#"@"
          LCALL   SEND
          MOV     A,CALLER_ADDRESS
          LCALL   SEND_ASCII
          LCALL   SEND_COMMA


          MOV     A,SAMPLE+2        ;NEXT WITH READING
          LCALL   SEND_ASCII
          MOV     A,SAMPLE+1
          LCALL   SEND_ASCII
          MOV     A,SAMPLE
          LCALL   SEND_ASCII

          LCALL   SEND COMMA
          MOV     A,#22H
          LCALL   SEND
          MOV     A,DAY
          LCALL   SEND ASCII
          MOV     A,#"-"
          LCALL   SEND
          MOV     A,MONTH
          LCALL   SEND_ASCII
```

```
            MOV     A,#"-"
            LCALL   SEND
            MOV     A,YEAR
            LCALL   SEND_ASCII
            MOV     A,#22H
            LCALL   SEND

            LCALL   SEND_COMMA
            MOV     A,#22H
            LCALL   SEND
            MOV     A,HOUR
            LCALL   SEND ASCII
            MOV     A,#":"
            LCALL   SEND
            MOV     A,MIN
            LCALL   SEND ASCII
            MOV     A,#":"
            LCALL   SEND
            MOV     A,SEC
            LCALL   SEND_ASCII
            MOV     A,#22H
            LCALL   SEND
            LCALL   SEND_COMMA

 ; NOW SEND DIGITAL DATA FROM CH1 TO CH16

            MOV     R3,CHANNEL_MAX    ;CHANGE 16D TO CHANEL_MAX
            MOV     R0,#ADC_DATA

READ_BUFFER:
            MOV     A,@R0
            INC     R0
            MOV     A,@R0

            JB      P2.3,NO_POL
            LCALL   FIND_POLARITY

NO_POL:     ANL     A,#0FH
            MOV     5EH,A
            DEC     R0
            MOV     A,@R0
            MOV     5DH,A
            PUSH    00H
            LCALL   BIN_BCD
            POP     00H

            JB      P2.3,NO_SEND_POL
            MOV     A,POL
            LCALL   SEND

NO_SEND_POL:
            MOV     A,57H
            LCALL   SEND_ASCII
            MOV     A,56H
            LCALL   SEND_ASCII

            MOV     A,R3
```

```
          CJNE    A,#01H,OLD_LINE
          SJMP    NEW_LINE

OLD_LINE:
          MOV     A,#","
          LCALL   SEND

NEW_LINE:
          INC     R0
          INC     R0
          DJNZ    R3,READ_BUFFER

;NOW BCC IS READY TO SEND MAKE IT TWO COMPLEMENT
          CLR     CHECK_SUM
          LCALL   SEND_COMMA
          MOV     A,#22H
          LCALL   SEND
          MOV     A,BCC
          CPL     A
          INC     A
          LCALL   SEND_ASCII
          MOV     A,#22H
          LCALL   SEND
          MOV     A,#EOT
          LCALL   SEND          ; ENDING FLAG WITH END OF TEXT
          LCALL   PTT_OFF
          RET

;FIND_POLARITY
;         ENTRY: A
;         EXIT:  ASCII + OR - IN POL BYTE

FIND_POLARITY:
              PUSH    ACC
              ANL     A,#00100000B
              CJNE    A,#00H,POSITIVE
              MOV     POL,#"-"
              SJMP    FOUND_POL

POSITIVE:
              MOV     POL,#"+"
FOUND_POL:
              POP     ACC
              RET


              SETB    SEC_FLAG          ; SET FLAG 1 SECOND
              MOV     A,SEC
              ADD     A,#01H
              DAA
              MOV     SEC,A


UP_SEC:       MOV     A,SEC
              CJNE    A,#60H,UP_MIN

              MOV     SEC,#00H
```

```
                MOV     A,MIN
                ADD     A,#01H
                DAA
                MOV     MIN,A

UP_MIN:         MOV     A,MIN
                CJNE    A,#60H,UP_HOUR
                MOV     MIN,#00H
                MOV     A,HOUR
                ADD     A,#01H
                DAA
                MOV     HOUR,A

UP HOUR:        MOV     A,HOUR
                CJNE    A,#24H,UP_DAY
                MOV     HOUR,#00H
                MOV     A,DAY
                ADD     A,#01H
                DAA
                MOV     DAY,A

UP_DAY:         MOV     A,DAY
                PUSH    ACC
                LCALL    GET_DAY_MONTH          ;GET NUMBER OF DAY
                                               ;EACH MONTH
                POP     ACC
                CJNE    A,DAY_MONTH,UP_MONTH
                MOV     DAY,#01H
                MOV     A,MONTH
                ADD     A,#01H
                DAA
                MOV     MONTH,A

UP_MONTH:       MOV     A,MONTH
                CJNE    A,#13H,UP_YEAR
                MOV     MONTH,#01H
                MOV     A,YEAR
                ADD     A,#01H
                DAA
                MOV     YEAR,A

UP_YEAR:        RET


;LATCH_COUNTER  MOVE COUNT FROM WIND_COUNTER TO WIND_SPEED

LATCH_COUNTER:  CLR   TR0
                MOV   A,TL0
                MOV   WIND_SPEED,A
                MOV   A,TH0
                MOV   WIND_SPEED+1,A
                MOV   TL0,#00H                  ; CLEAR COUNTER
                MOV   TH0,#00H
                SETB  TR0
                RET

;GET DAY MONTH
```

```
;     ENTRY  : MONTH
;     EXIT   : DAY_MONTH

GET_DAY_MONTH:  MOV    DPTR,#DAY_TABLE
                MOV    A,MONTH
                LCALL  BCD_BIN
                DEC    A
                MOVC   A,@A+DPTR
                MOV    DAY_MONTH,A
                RET

DAY_TABLE:      DFB    31H      ; JAN
                DFB    28H      ; FEB
                DFB    31H      ; MAR
                DFB    30H      ; APR
                DFB    31H      ; MAY
                DFB    30H      ; JUN
                DFB    31H      ; JUL
                DFB    31H      ; AUG
                DFB    30H      ; SEP
                DFB    31H      ; OCT
                DFB    30H      ; NOV
                DFB    31H      ; DEC


;BCD_SUM ADD TWO'S SIX DIGIT BCD CONTAIN IN THREE BYTE
;            POINTED BY R0 AND R1
;                   FIRST = (FIRST + SECOND)
;            ENTRY: R0 POINT TO LSD FIRST BCD NUMBER
;                   R1 POINT TO LSD SECOND BCD NUMBER
;            EXIT:  RESULTS PLACE TO FIRST BCD NUMBER
;                   C = 1 (OVERFLOW,eg.GREATER THAN 999999)

BCD_SUM:       CLR     C
               MOV     R7,#03H
SUM:           MOV     A,@R0
               ADDC    A,@R1
               DAA
               MOV     @R0,A
               INC     R0
               INC     R1
               DJNZ    R7,SUM
               RET

;BCD ASCII CONVERT BCD NUMBER TO ASCII CODE
;           ENTRY: A (LOW NIBBLE ONLY)
;           EXIT: A

BCD_ASCII:     ANL     A,#0FH          ;GET ONLY LOW NIBBLE
               ADD     A,#30H          ;CONVERT TO ASCII CODE
               RET

;BUFFER_ASCII CONVERT BCD IN BUFFER START $30 - $35
;          ENTRY: BCD NUMBER IN TRANSMIT BUFFER
;          EXIT: ASCII CODE IN TRANSMIT BUFFER
```

```
BUFFER_ASCII: MOV     R7,#06H
              MOV     R0,#30H
NEXT_ASCII:   MOV     A,@R0
              LCALL   BCD_ASCII
              MOV     @R0,A
              INC     R0
              DJNZ    R7,NEXT_ASCII
              RET

;SEND_BUFFER SEND ASCII CODE IN TRANSMIT BUFFER TO
;SERIAL PORT
;          THEN DATA SEPERATOR /32/32
;          ENTRY: ASCII CODE IN TRANSMIT BUFFER
;          EXIT: NO

SEND_BUFFER:  MOV     R7,#06H
              MOV     R0,#35H
NEXT_BYTE:    MOV     A,@R0
              LCALL   SEND        ;SEND OUT TO SERIAL PORT
              DEC      R0
              DJNZ     R7,NEXT_BYTE

              MOV      A,#20H
              LCALL    SEND
              MOV      A,#20H
              LCALL    SEND
              RET

;TRANSFERS MOV 4 BYTE IN DATA BUFFER POINTED BY R0 TO
;DISPLAY BUFFER
;   ENTRY: BCD NUMBER 4 BYTES IN CURRENT PROG & TIME
;   XIT: BCD NUMBER 6 DIGIT IN TRANSMIT BUFFER

TRANSFERS:    MOV      R7,#03
              MOV      R1,#30H
TRANS2:       MOV      A,@R0
              MOV      R6,A
              ANL      A,#0FH
              MOV      @R1,A
              INC      R1
              MOV      A,R6
              SWAP     A
              ANL      A,#0FH
              MOV      @R1,A
              INC      R0
              INC      R1
              DJNZ     R7,TRANS2
              RET


;COMPARE TWO 16 BIT BINARY NUMBER
;         ENTRY: R0 FIRST OPERAND
;                R1 SECOND OPERAND
;         EXIT: C=1 FIRST OPERAND < SECOND OPERAND
;               C=0 FIRST OPERAND >= SECOND OPERAND
```

```
COMPARE:    CLR     C
            MOV     A,@R0
            SUBB    A,@R1
            INC     R0
            INC     R1
            MOV     A,@R0
            SUBB    A,@R1
            RET

;READ_VANE READ WIND DIRECTION FROM PB0-PB3 16 DIRECTIONS
;GRAY CODE

READ_VANE:  MOV     DPTR,#PORT B
            MOVX    A,@DPTR
            ANL     A,#0FH
            MOV     VANE,A
            RET


;SEND_VANE SEND WIND DIRECTION FROM GRAY CODE 0-F
;             ENTRY : A = GRAY CODE REPESENT VANE

SEND VANE:  MOV     DPTR,#VANE TABLE
            MOV     B,#03H
            MUL     AB
            MOV     B,A
            MOVC    A,@A+DPTR
            LCALL   SEND
            INC     B
            MOV     A,B
            MOVC    A,@A+DPTR
            LCALL   SEND
            INC     B
            MOV     A,B
            MOVC    A,@A+DPTR
            LCALL   SEND
            RET

;VANE_TABLE

VANE_TABLE: DFB     "N  "
            DFB     "NNW"
            DFB     "NW "
            DFB     "WNW"
            DFB     "W  "
            DFB     "WSW"
            DFB     "SW "
            DFB     "SSW"
            DFB     "S  "
            DFB     "SSE"
            DFB     "SE "
            DFB     "ESE"
            DFB     "E  "
            DFB     "ENE"
            DFB     "NE "
            DFB     "NNE"
```

```
;SEND_MONTH SEND MONTH TO TERMINAL
;       ENTRY : MONTH

SEND_MONTH:     MOV     A,MONTH
                LCALL   BCD_BIN
                DEC     A
                MOV     DPTR,#MONTH_TABLE
                MOV     B,#03H
                MUL     AB
                MOV     B,A
                MOVC    A,@A+DPTR
                LCALL   SEND
                INC     B
                MOV     A,B
                MOVC    A,@A+DPTR
                LCALL   SEND
                INC     B
                MOV     A,B
                MOVC    A,@A+DPTR
                LCALL   SEND
                RET

MONTH_TABLE:    DFB     "JAN"
                DFB     "FEB"
                DFB     "MAR"
                DFB     "APR"
                DFB     "MAY"
                DFB     "JUN"
                DFB     "JUL"
                DFB     "AUG"
                DFB     "SEP"
                DFB     "OCT"
                DFB     "NOV"
                DFB     "DEC"

;SEND_DATE SEND DAY-MONTH-YEAR TO TERMINAL

SEND_DATE:      MOV     A,DAY
                LCALL   SEND_ASCII
                MOV     A,#"-"
                LCALL   SEND
                LCALL   SEND_MONTH
                MOV     A,#"-"
                LCALL   SEND
                MOV     A,YEAR
                LCALL   SEND_ASCII
                LCALL   SEND_SPACE
                RET
;SEND_RECORD

SEND_RECORD:    LCALL   SEND_CR
                MOV     A,CHANNEL
                LCALL   SEND_ASCII
                LCALL   SEND_SPACE
                MOV     A,POL
                LCALL   SEND
                MOV     A,ADC_BUFFER
```

```
                LCALL   SEND_ASCII
                MOV     A,ADC_BUFFER+1
                LCALL   SEND_ASCII
                RET

send_time:      LCALL   PTT_ON
                MOV     A,#SOH
                LCALL   SEND
                LCALL   SEND_CR
send_time1:     MOV     A,DAY
                LCALL   SEND_ASCII
                MOV     A,#"-"
                LCALL   SEND
                MOV     A,MONTH
                LCALL   SEND_ASCII
                MOV     A,#"-"
                LCALL   SEND
                MOV     A,#25H
                LCALL   SEND_ASCII
                MOV     A,YEAR
                LCALL   SEND_ASCII
                LCALL   SEND_SPACE
                MOV     A,HOUR
                LCALL   SEND_ASCII
                MOV     A,#":"
                LCALL   SEND
                MOV     A,MIN
                LCALL   SEND_ASCII
                MOV     A,#":"
                LCALL   SEND
                MOV     A,SEC
                LCALL   SEND_ASCII
;               MOV     A,#EOT
;               LCALL   SEND
;               LCALL   PTT_OFF
                RET

;SET_TIME ENTER CURRENT TIME

SET_TIME:       ;MOV    DPTR,#TIME_SET
                ;LCALL  SEND_STRING
                 LCALL  GET_BYTE
                 MOV    HOUR,A                  ;SAVE HOUR
                 LCALL  GET_BYTE
                 MOV    MIN,A                   ;SAVE MIN
                 LCALL  GET_BYTE
                 MOV    SEC,A                   ;SAVE SEC
                 LCALL  PTT_ON
                 LCALL  SEND_TIME
                 LCALL  PTT_OFF
                 LJMP   MAIN3

;SET_DATE ENTER CURRENT DATE

SET_DATE:       ;MOV    DPTR,#DATE_SET
                ;LCALL  SEND_STRING
                 LCALL  GET_BYTE
                 MOV    DAY,A
```

```
                LCALL   GET_BYTE
                MOV     MONTH,A
                LCALL   GET_BYTE
                MOV     YEAR,A
                LCALL   PTT_ON
                LCALL   SEND_TIME
                LCALL   PTT_OFF
                LJMP    MAIN3

;CHANGE DATE TIME

CHANGE_DATE_TIME:
;               LCALL   SET_TIME
;               LCALL   SET_DATE
                LJMP    MAIN3

;DISPLAY_DATE_TIME

DISPLAY_DATE_TIME:
                LCALL   PTT_ON
                LCALL   SEND_TIME
                LCALL   PTT_OFF
                LJMP    MAIN3

;UPDATE_SAMPLE INCREMENT SAMPLE BY 1 (000000 - 999999)

UPDATE_SAMPLE:
                CLR     C
                MOV     A,SAMPLE
                ADD     A,#01D
                DAA
                MOV     SAMPLE,A
                MOV     A,SAMPLE+1
                ADDC    A,#00D
                DAA
                MOV     SAMPLE+1,A
                MOV     A,SAMPLE+2
                ADDC    A,#00D
                DAA
                MOV     SAMPLE+2,A
                RET


;SCAN ANALOG INPUT (TEMPERATURE,VANE,WIND SPEED)

SCAN:           MOV     SEC,#00H        ;CLEAR SECOND
                CLR     SEC_FLAG
                SETB    TR0             ;START TIMER

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TEST10:         LCALL   PTT_ON
                MOV     DPTR,#LOG_DECARE
                LCALL   SEND_STRING


RUN0:          ;LCALL   RECIVE          ;WAIT FOR SPACE BAR
```

```
                ;CJNE     A,#" ",RUNO

                 MOV      DPTR,#REPORT_TITLE
                 LCALL    SEND_STRING

                 LCALL    SEND_TIME1
                 LCALL    SEND 22H
                 MOV      DPTR,#INTERVAL
                 LCALL    SEND_STRING
                 LCALL    READ_TIME1
                 LCALL    SEND_22H
                 MOV      DPTR,#CONTENTS
                 LCALL    SEND_STRING

                 LCALL    PTT_OFF


                 MOV      TRIG_SEC,#00H  ;CLEAR TIMER SEC AND MIN
                 MOV      TRIG_MIN,#00H

RUN:             SETB     P1.4
;                JB       P2.1,SOFT_TRIG ;IF P2.1 LOW THEN SEND
                                         ;DATA TO PC CONTINOUSLY
                 JNB      READY,RUN
                 CLR      READY          ;READY FOR SENDING DATA
                                         ;TO REMOTE TERMINAL
                 LCALL    UPDATE_SAMPLE  ;INCREMENT SAMPLE BY ONE
                 SJMP     FREE_RUN

SOFT_TRIG:       JNB      RI,RUN         ;ELS WAIT ADDRESS FROM PC
                 CLR      RI             ;ADDRESS IS IN SBUF
                 CLR      P1.4           ;HOLD 7109

                 MOV      P3,#0FFH
                 MOV      A,P3

                 ANL      A,#0F0H        ;GET DATA ONLY P3.4-P3.7
                 SWAP     A
                 LCALL    BIN_ASCII
                 MOV      B,A
                 MOV      A,SBUF
                 CJNE     A,B,RUN        ;ADDRESS BOARD #0

;IF ADDRESS MATCH SEND DATA VIA SERIAL PORT 19200 BIT/SEC

FREE_RUN:        LCALL    SEND_7109_SERIAL

                 JNB      RI,PAUSE
                 CLR      RI
                 MOV      A,SBUF
                 CJNE     A,#"p",PAUSE   ;CHECK CONTROL BREAK
                                         ;CHARACTER
                 LJMP     MAIN3          ;RECEIVES CONTROL BREAK
PAUSE:
                 LJMP     RUN
```

```
;TRIGGER SEND
TRIGGER_SEND:
             ;LCALL   RECIVE              ;DUMMY READ @
             ;LCALL   GET_BYTE            ;GET ADDRESS
             ;PUSH    ACC
             ;LCALL   RECIVE              ;DUMMY READ >>@
             ;LCALL   RECIVE
             ;LCALL   RECIVE
             ;LCALL   GET BYTE            ;GET WHO CALL
              MOV     CALLER_ADDRESS,#10H
             ;POP     ACC
             ;CJNE    A,STATION_ADDRESS,NO_RESPONSE
              LCALL   SEND_7109_SERIAL
              LCALL   UPDATE_SAMPLE
              LJMP    MAIN3

NO_RESPONSE: MOV      A,#NAK
             LCALL    SEND
             LJMP     MAIN3

;SEC_DELAY GET A BYTE BCD NUMBER PUT TO TRIG_SEC BYTE

SEC_DELAY:   ;MOV     DPTR,#GET_SEC
             ;LCALL   SEND_STRING
              LCALL   GET_BYTE
              PUSH    ACC
              LCALL   SEND_CR
              POP     ACC
              MOV     TIME_SEC,A
              MOV     TIME_MIN,#00D       ;ONLY SECOND USED
              LJMP    MAIN3

;MIN_DELAY GET A BYTE BCD NUMBER PUT TO TRIG_MIN BYTE

MIN_DELAY:   ;MOV     DPTR,#GET_MIN
             ;LCALL   SEND STRING
              LCALL   GET_BYTE
              PUSH    ACC
              LCALL   SEND_CR
              POP     ACC
              MOV     TIME_MIN,A
              MOV     TIME_SEC,#00D        ;ONLY MINUTE USED
              LJMP    MAIN3

;HELP_COMMAND SEND LISTING HELP COMMAND

HELP_COMMAND:
              MOV     DPTR,#HELP
              LCALL   PTT_ON
              LCALL   SEND_STRING
              LCALL   PTT_OFF
              LJMP    MAIN3

;READ TIME SET

READ TIME SET:
              MOV     DPTR,#READ_TIMER
```

```
                LCALL   PTT_ON
                LCALL   SEND_STRING
                LCALL   READ_TIME1
                LCALL   PTT_OFF
                LJMP    MAIN3

READ_TIME1:
                MOV     A,TIME_SEC
                LCALL   SEND_ASCII
                MOV     A,#","
                LCALL   SEND
                MOV     A,TIME_MIN
                LCALL   SEND_ASCII
               ;MOV     A,#EOT
               ;LCALL   SEND
                RET

;SET_BAUD PROGRAMMABLE BAUD RATE

SET_BAUD:       MOV     DPTR,#BAUD
                LCALL   SEND_STRING
                LCALL   RECIVE
                CJNE    A,#"1",BAUD2
                MOV     TH1,#0A0H
                SJMP    BAUD10

BAUD2:          CJNE    A,#"2",BAUD3
                MOV     TH1,#0D0H
                SJMP    BAUD10

BAUD3:          CJNE    A,#"3",BAUD4
                MOV     TH1,#0E8H
                SJMP    BAUD10

BAUD4:          CJNE    A,#"4",BAUD5
                MOV     TH1,#0F4H
                SJMP    BAUD10

BAUD5:          CJNE    A,#"5",BAUD6
                MOV     TH1,#0FAH
                SJMP    BAUD10

BAUD6:          CJNE    A,#"6",BAUD10
                MOV     TH1,#0FDH

BAUD10:         LJMP    MAIN3


;NEW_FILE

NEW_FILE:       MOV     SAMPLE,#01D
                MOV     SAMPLE+1,#00D
                LJMP    TEST10

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
;SEND_PROMPT

SEND_PROMPT: LCALL    SEND_CR
             MOV      A,#"$"
             LCALL    SEND
             MOV      A,#" "
             LCALL    SEND
             RET

;LOG_ADDRESS GET STATION ADDRESS VIA COMMAND \25

LOG_ADDRESS: LCALL    GET_BYTE
             MOV      STATION_ADDRESS,A
             LCALL    SEND_ASCII
             LJMP     MAIN3

;CONNECT WAIT CALL SIGN END THEN ECHO CONNECT MESSAGE
CONNECT:     LCALL    RECIVE
             CJNE     A,#CR,CONNECT
             LCALL    PTT_ON
             MOV      DPTR,#CALL_SIGN
             LCALL    SEND_STRING
             LCALL    PTT_OFF
             LJMP     MAIN3


;GET_ADDRESS SEND CURRENT ADDRESS TO CENTRAL STATION

GET_ADDRESS: LCALL    PTT_ON
             MOV      A,STATION_ADDRESS
             LCALL    SEND_ASCII
             CLR      PTT
             LJMP     MAIN3


;SEND_STRING SEND STRING CONSTANT TO TERMINAL
;            ENTRY: DPTR
;            EXIT:  FOUND EOS
;
SEND_STRING: CLR      A
             MOVC     A,@A+DPTR
             CJNE     A,#EOS,SEND_STRING1
             RET

SEND_STRING1:
             PUSH     DPL
             PUSH     DPH
             LCALL    SEND
             POP      DPH
             POP      DPL
             INC      DPTR
             SJMP     SEND_STRING

;SEND_SP
SEND_SP:     MOV      A,#SPACE
             LCALL    SEND
             MOV      A,#SPACE
```

```
                LCALL   SEND
                RET

;BIN_ASCII CONVERT BIN TO ASCII CODE
;          ENTRY: A
;          EXIT : A
BIN ASCII:
                ANL     A,#0FH
                MOV     R2,A
                CLR     C
                SUBB    A,#0AH
                JNC     ASCII_AF2
                MOV     A,R2
                ADD     A,#30H
                RET
ASCII_AF2:      MOV     A,R2
                ADD     A,#37H
                RET

;BCD_BIN CONVERT 0-12 BCD NUMBER TO BINARY NUMBER
;         ENTRY: A
;         EXIT:  A

BCD_BIN:        CJNE    A,#10H,BIN2
                MOV     A,#0AH
                RET
BIN2:           CJNE    A,#11H,BIN3
                MOV     A,#0BH
                RET
BIN3:           CJNE    A,#12H,BIN4
                MOV     A,#0CH
                RET
BIN4:           RET

;BYTE_ASCII  CONVERT A TO ASCII IN R4, R5

BYTE_ASCII:     MOV     B,A
                LCALL   BIN_ASCII
                MOV     R4,A
                MOV     A,B
                SWAP    A
                LCALL   BIN_ASCII
                MOV     R5,A
                RET

;ASCII_BIN  CONVERTS SINGLE ASCII CHARACTER TO SINGLE NIBBLE
;BINARY
;   ENTRY: A ( 30-39 for 0-9, 41-46 for A-F, 61-66 for a-f)
;   EXIT: A
;   IGNOR LOWER CASE CLR BIT 5 BEFORE

ASCII BIN:      CLR     C
                MOV     R6,A
                SUBB    A,#41H
                JNC     ASCII_AF
                MOV     A,R6
                CLR     C
```

```
                SUBB    A,#30H
                RET

ASCII AF:       CLR     C
                MOV     A,R6
                SUBB    A,#37H
                RET

;SEND_ASCII SEND ASCII IN R5(HI) R4(LO) BYTE TO MONITOR
;               INPUT   : A
;               OUTPUT  : NONE
SEND_ASCII:     LCALL   BYTE_ASCII
                MOV     A,R5
                LCALL   SEND
                MOV     A,R4
                LCALL   SEND
                RET


;COMBINE DATA 2 BYTE TO SINGLE BYTE
;         ENTRY: R4= LOW NIBBLE
;                R5= HIGH NIBBLE
;         EXIT : A
;
COMBINE:        MOV     A,R5
                SWAP    A
                ADD     A,R4
                RET

;GET_BYTE GET DATA FROM SERIAL PORT TWO BYTE SAVE TO
;    R5 AS A HIGH NIBBLE AND R4 AS A LOW NIBBLE
;    ENTRY: NO, SERIAL PORT MUST BE INITIALZED BEFORE CALLING
;    EXIT: A
;
;
GET BYTE:       LCALL   RECIVE ; FIRST READING MUST GO TO R5 !!
                LCALL   ASCII_BIN
                MOV     R5,A
                LCALL   RECIVE
                LCALL   ASCII_BIN
                MOV     R4,A
                LCALL   COMBINE
                RET

;GET_WORD GET ASCII WORD FROM TERMINAL END BY ENTERING CR
;    RECEIVE BACK SPACE FOR DELETE BACKWARD
;    START INPUT BUFFER = 30H

GET WORD:       MOV     R0,#INPUT_BUFFER
WORD0:          LCALL   RECIVE
                CJNE    A,#CR,WORD1         ;END OF STRING
                SJMP    END_OF_STRING
WORD1:          CJNE    A,#BS,WORD2
                SJMP    WORD4
WORD2:          CJNE    A,#FS,WORD3
                INC     R0
                SJMP    WORD0
```

```
WORD3:          MOV     B,A
                CLR     C
                SUBB    A,#41H
                JC      WORD5
                MOV     A,B
                ANL     A,#11011111B    ;CONVERT LOWER TO UPPER
                SJMP    WORD6
WORD5:          MOV     A,B             ;IF A= 0-9 THEN STORE
                                        ;TO BUFFER
WORD6:          MOV     @R0,A
                INC     R0
                SJMP    WORD0
END_OF_STRING:

                MOV     A,#EOS
                MOV     @R0,A            ;SAVE EOS
                RET
WORD4:          DEC     R0
                SJMP    WORD0

;SEND_WORD SEND ASCII WORD IN DATA MEMORY TO  TERMINAL
;          ENTRY: R0 BUFFER POINTER

SEND_WORD:      MOV     A,@R0
                CJNE    A,#EOS,SEND_WORD2
                RET
SEND_WORD2:     LCALL   SEND
                INC     R0
                SJMP    SEND_WORD


;GET RANGE READ ASCII IN INPUT BUFFER CONVERT TO START,
;STOP ADDRESS
;           MOVE TO UP_START_L    $00
;                   UP_START_H    $00
;                   UP_STOP_L     $07
;                   UP_STOP_H     $FF
;
GET_RANGE:      MOV     UP_START_L,#00H
                MOV     UP_START_H,#00H
                MOV     UP_STOP_L,#00H
                MOV     UP_STOP_H,#00H
                MOV     R1,#INPUT_BUFFER

RANGE0:         MOV     A,@R1
                CJNE    A,#EOS,RANGE1
                MOV     DPTR,#DO_NOTING
                LCALL   SEND_STRING
                RET

RANGE1:         CJNE    A,#20H,RANGE2       ;OK GET FIRST ASCII
                SJMP    RANGE3
RANGE2:         LCALL   ASCII_BIN
                MOV     B,A
                MOV     R0,#UP_START_L
                LCALL   NIBBLE_SHIFT
                MOV     A,UP_START_L
```

```
                ADD     A,B
                MOV     UP_START_L,A
                INC     R1
                SJMP    RANGE0

RANGE3:         INC     R1
RANGE4:         MOV     A,@R1
                CJNE    A,#EOS,RANGE5
                RET

RANGE5:         CJNE    A,#20H,RANGE6
                INC     R1
                SJMP    RANGE4
RANGE6:         LCALL   ASCII_BIN
                MOV     B,A
                MOV     R0,#UP_STOP_L
                LCALL   NIBBLE_SHIFT
                MOV     A,UP_STOP_L
                ADD     A,B
                MOV     UP_STOP_L,A
                INC     R1
                SJMP    RANGE4


;SEND_BRK
SEND_BRK:       MOV     A,#"*"
                LCALL   SEND
BRK:            SJMP    BRK


;SEND_WORDS send word in WORD_L AND WORD_H to terminal
;          entry: DATA IN WORD_L AND WORD_H

SEND_WORDS:     MOV     A,WORD_H
                LCALL   SEND_ASCII
                MOV     A,WORD_L
                LCALL   SEND_ASCII
                RET

;-----------------------------------------------------------------
;                        BIG - SUB ROUTINE
;-----------------------------------------------------------------

RECEIVE_TIME:                   ;R0   = Time Sec.
;============                   ;Reg : A,R1
    LOOP1:      MOV     R1,SEC
    LOOP2:      MOV     A,SEC
                JNB     RI,CHECK
                MOV     A,SBUF
                SJMP    OK
    CHECK:      CJNE    A,01h,LOOP3
                SJMP    LOOP2
    LOOP3:     .DJNZ    R0,LOOP1
    OK:         RET

B_TIMER:                        ;R2   = Time Sec.
;============                   ;Reg : R3
```

```
   T_LOOP1:    MOV     R3,SEC
   T_LOOP2:    MOV     A,SEC
               CJNE    A,03h,T_LOOP3
               SJMP    T_LOOP2
   T_LOOP3:    DJNZ    R2,T_LOOP1
               RET

B_DELAY:                    ;delay = (#R0 * 0.01) Sec>
;=======                    ;Input = R0
                            ;Reg. : R0,R1,R2

               MOV      R1,#12h
DELAY_2:       MOV      R2,#0FFh
DELAY_1:       DJNZ     R2,DELAY_1
               DJNZ     R1,DELAY_2
               DJNZ     R0,B DELAY
               RET

COPY0_EX:                   ;COPY ROM TO EX_RAM BY COUNTER
;=======                    ;DPTR  = Start addr.
                            ;R1,R0 = Destination addr.
                            ; B    = Byte Counter
                            ; Reg. : R0,R1,B

               MOV      A,#00h
               MOVC     A,@A+DPTR
               MOV      P2,R1
               MOVX     @R0,A
               INC      DPTR
               INC      R0
               CJNE     R0,#00h,COPY0_IX_INC
               INC      R1
COPY0_IX_INC:  DJNZ     B,COPY0_EX
               RET
COPY1_EX:                   ;COPY EX_RAM TO EX_RAM BY COUNTER
;=======                    ;DPTR  = Start addr.
                            ;R1,R0 = Destination addr.
                            ; B    = Byte Counter
                            ; Reg. : R0,R1,B

               MOVX     A,@DPTR
               MOV      P2,R1
               MOVX     @R0,A
               INC      DPTR
               INC      R0
               CJNE     R0,#00h,COPY1_IX_INC
               INC      R1
COPY1_IX_INC:  DJNZ     B,COPY1_EX
               RET

COPY2_EX:                   ;COPY EX_RAM TO EX_RAM
;=======                    ;DPTR  = Start addr.
                            ;R3,R2 = Final Block addr.
                            ;R1,R0 = Destination addr.
                            ;OutPut : ACC = 0ffH If copy error
                            ; Reg.  : ACC,DPTR,R0,R1,R2,R3
```

```
                PUSH    DPH
                PUSH    DPL
                PUSH    00h
                PUSH    01h
                PUSH    02h
                PUSH    03h
                INC     R2
                CJNE    R2,#00h,COPY2_LOOP
                INC     R3
COPY2_LOOP:     MOVX    A,@DPTR
                MOV     P2,R1
                MOVX    @R0,A
                INC     DPTR
                INC     R0
                CJNE    R0,#00h,COPY2_IX_INC
                INC     R1
COPY2_IX_INC:   MOV     A,DPL
                CJNE    A,02h,COPY2_LOOP
                MOV     A,DPH
                CJNE    A,03h,COPY2_LOOP
                POP     03h
                POP     02h
                POP     01h
                POP     00h
                POP     DPL
                POP     DPH
                LCALL   COM_B
                RET

COM_B:                  ;COMPARE BLOCK
;=====                  ;DPTR  = Start Block addr.
                        ;R3,R2 = Final Block addr.
                        ;R1,R0 = Destination addr.
                        ;OutPut  : ACC = 0FFh If non equal
                        ;Reg.    : ACC,DPTR,R0,R1,R2,R3,R4

                INC     R2
                CJNE    R2,#00h,COMB_LOOP
                INC     R3
COMB_LOOP:      MOVX    A,@DPTR
                MOV     R4,A
                MOV     P2,R1
                MOVX    A,@R0
                CLR     C
                SUBB    A,R4            ;If equal ACC = 00h
                CJNE    A,#00h,COMB NOT
                INC     DPTR
                INC     R0
                CJNE    R0,#00h,COMB_IX_INC
                INC     R1
COMB_IX_INC:    MOV     A,DPL
                CJNE    A,02h,COMB_LOOP
                MOV     A,DPH
                CJNE    A,03h,COMB_LOOP
                MOV     A,#00h
                SJMP    COMB_OUT
COMB_NOT:       MOV     A,#0FFH
```

```
COMB_OUT:       RET

CHK_SUM:                        ;CHECK SUM BLOCK
;=======                       ;INPUT    DPTR  = Start Block addr.
                                ;OutPut   : BCC  = Total.
                                ;         : DPTR = Addr CHECK SUM
                                ;         : R5   = HI BYTE
                                ;         : R4   = LO BYTE
                                ;Reg.     : ACC,DPTR,R0,R2,R3,R4,R5

                MOV     BCC,#00h
                PUSH    DPH
                PUSH    DPL

CHK_LOOP:       MOVX    A,@DPTR
                CJNE    A,#EOT,CHK_ADD
                LJMP    CHK_NEXT

CHK_ADD:        INC     DPTR
                LJMP    CHK_LOOP

CHK_NEXT:       MOV     R0,#04h         ;DATA end addr = DPTR - 5
CHK_DEC:        MOV     A,DPL
                CJNE    A,#00,CHK_OUT
                DEC     DPH
CHK_OUT:        DEC     DPL
                DJNZ    R0,CHK_DEC
                MOV     R3,DPH
                MOV     R2,DPL
                POP     DPL
                POP     DPH
CHK_LOOP2:      MOVX    A,@DPTR
                INC     DPTR
                ADD     A,BCC
                MOV     BCC,A

                MOV     A,DPL
                CJNE    A,02h,CHK LOOP2
                MOV     A,DPH
                CJNE    A,03h,CHK_LOOP2
                MOV     DPH,R3
                MOV     DPL,R2
                INC     DPTR
                MOV     A,BCC
                CPL     A
                INC     A
                LCALL   BYTE_ASCII
                RET

;SEND_Block TO TERMINAL
;               ENTRY: DPTR
;               EXIT : FOUND EOT
;
SEND_BLOCK:     CLR     A
                MOVC    A,@A+DPTR
                CJNE    A,#EOT,SEND_BLOCK1
                LCALL   SEND
```

```
                RET

SEND_BLOCK1:    PUSH    DPL
                PUSH    DPH
                LCALL   SEND
                POP     DPH
                POP     DPL
                INC     DPTR
                SJMP    SEND_BLOCK


;*************************************************************
;                   STRING CONSTANT AREA
;*************************************************************

TITLE:      DFB     SOH,CR,LF
            DFB  "ANALOG ZenDer ZX3.0",CR,LF
            DFB   CR,LF
            DFB  "Telemetry Research Group, TRG",CR,LF
            DFB  "Applied Physics Laboratory, APL",CR,LF
            DFB  "Department of Applied Physics",CR,LF
            DFB  "KING MONGKUT'S INSTITUTE OF TECHNOLOGY "
            DFB  "LATKRABANG",CR,LF,EOT,EOS
PROMPT:     DFB  "$ ",EOS


HELP:       DFB  SOH,CR,LF
            DFB  "e    e070510 < 07:05:10 current time",CR,LF
            DFB  "c    c140737 < 14-07-37 current date",CR,LF
            DFB  "d    dispaly current date & time",CR,LF
            DFB  "r    read time setting in delay timer",CR,LF
            DFB  "g    go to sending routine",cr,lf
            DFB  "s    s10 < delay between sample 10 seconds",
            DFB  CR,LF
            DFB  "m    m30 < delay between sample 30 minutes",
            DFB  CR,LF
            DFB  "b    baud rate select",cr,lf
            DFB  "n    new file & begin reading",cr,lf
            DFB  "t    t@30<<@00 call station #30 from "
            DFB  "station #00",cr,lf
            DFB  "b    stop sending",cr,lf
            DFB  "p    pause toggle",cr,lf
            DFB  "a    a30 log station address to 30",cr,lf
            DFB  "?    command help",cr,lf,EOT,EOS


GET_SEC:    DFB  SOH,CR,LF,"Delay in second [00-99] ? ",EOS
GET_MIN:    DFB  SOH,CR,LF,"Delay in minute [00-99] ? ",EOS

READ_TIMER:DFB  SOH,CR,LF,"Delay between sample"
            DFB  " [second,minute] > ",EOS
BAUD:       DFB  SOH,CR,LF,"1)  300",CR,LF
            DFB             "2)  600",CR,LF
            DFB             "3) 1200",CR,LF
            DFB             "4) 2400",CR,LF
            DFB             "5) 4800",CR,LF
            DFB             "6) 9600",CR,LF,LF
```

```
            DFB             "Select> ",EOT,EOS

DATE_SET:   DFB     CR,LF
TIME_SET:   DFB     CR,LF
LOG_DECARE: DFB     SOH,CR,LF,"Pls open log file then press"
            DFB     " space bar to start reading",EOS
REPORT_TITLE: DFB   CR,LF,22h,"ANALOG ZENDER ZX3.0",22h,CR,LF
              DFB   22H,"Telemetry Research Group",22h,CR,LF
              DFB   22H,"APPLIED PHYSICS DEPARTMENT, KMITL",
              DFB   22H,cr,lf,lf
              DFB   22H,"STARTING DATE & TIME > ",EOS
INTERVAL:     DFB   CR,LF,22H,"SAMPLING INTERVAL [SEC,MIN] >"
              DFB   " ",EOS
CONTENTS:     DFB   CR,LF,22H,"DATA FILE >____________________"
              DFB   "__________",22H,CR,LF,LF
              DFB   22H,"READING",22H,COMMA,22H,"DATE",22H
              DFB   COMMA,22H,"TIME",22H,COMMA
              DFB   22H,"CH1",22H,COMMA
              DFB   22H,"CH2",22H,COMMA
              DFB   22H,"CH3",22H,COMMA
              DFB   22H,"CH4",22H,COMMA
              DFB   22H,"CH5",22H,COMMA
              DFB   22H,"CH6",22H,COMMA
              DFB   22H,"CH7",22H,COMMA
              DFB   22H,"CH8",22H,COMMA
              DFB   22H,"CH9",22H,COMMA
              DFB   22H,"CH10",22H,COMMA
              DFB   22H,"CH11",22H,COMMA
              DFB   22H,"CH12",22H,COMMA
              DFB   22H,"CH13",22H,COMMA
              DFB   22H,"CH14",22H,COMMA
              DFB   22H,"CH15",22H,COMMA
              DFB   22H,"CH16",22H,COMMA
              DFB   22H,"CHECK SUM",22H,CR,LF,EOT,EOS

CALL_SIGN:    DFB   SOH,CR,LF,"HS1HQN",CR,LF,EOT,EOS
ENTER_TIME:   DFB   CR,LF

ENTER_DATE:   DFB   CR,LF
DO_NOTING:    DFB   CR,LF
CHANNELS:     DFB   "CHANNEL"

;-------------------------------------------------------------
                    ORG 1000h

NAME_RP:      DFB       "R99"
;-------------------------------------------------------------
              END
```

- โปรแกรมการสื่อสารข้อมูลที่ใช้กับ Central Work Station

```
Program zender10;     {Have name and check sum check SOH Name}
uses Crt,Dos,Graph;
Const Com1 = 0;
      Com2 = 1;
      Com3 = 2;
      Com4 = 3;

Var Regs      :Registers;
    lead,ch,J,C,S,k,data      : Char;
    out,Initbyte,Com      : Byte;
    Portbase              : Word;
    h,i,Show,l,B,p,r1,r2  : Integer;
    Row1                  : Array[1..76] of Char;
    RxData                : Array[1..255] of Char;
    FILENAME              : String;
    F                     : Text;
    Home,KEYS,Check,Num2   : Char;
    Msg,Msg01,Msg02,Msg03,Msg04    : String;
    ForG,BacG,BCC         : Integer;
    Mode,T                : Byte;
    Color,Size            : Word;
    Mono,VgaMono,TmisMode : Boolean;
    D                     : LongInt;

Procedure InitSerial(Com,Initbyte:Byte);
    Begin
         Regs.ah  := $00;
         Regs.dx  := Com;
         Regs.al  := Initbyte;
         Intr($14,Regs);

    End;

Function SendSerial (Data:Char):Byte;
    Begin
         Regs.ah  := $01;
         Regs.dx  := Com;
         Regs.al  := Ord(Data);
         Intr($14,Regs);
         If (Regs.al And $80) = $80 Then SendSerial := $FF
         Else SendSerial := 0;
    End;

Procedure SetRts(Com:Byte);
Var OldVal :Byte;
Begin
     Case Com of
          Com1 : Begin
                 OldVal := Port[$3FC];
                 OldVal := OldVal Or $02; {set pin RTS}
                 Port[$3FC] := OldVal ;
                 End;

          Com2 : Begin
                 OldVal := Port[$2FC];
                 OldVal := OldVal Or $02; {set pin RTS}
                 Port[$2FC] := OldVal ;
```

```
                End;
          Com3 : Begin
                 OldVal := Port[$3EC];
                 OldVal := OldVal Or $02; {set pin RTS}
                 Port[$3EC] := OldVal ;
                 End;
          Com4 : Begin
                 OldVal := Port[$2EC];
                 OldVal := OldVal Or $02; {set pin RTS}
                 Port[$2EC] := OldVal ;
                 End;
      End;
End;

Procedure ClrRts(com:Byte);
Var OldVal : Byte;
Begin
     Case Com Of
          Com1 : Begin
                 OldVal := Port[$3FC];
                 OldVal := OldVal And $0FD; {clear pin RTS}
                 Port[$3FC] := OldVal ;
                 End;

          Com2 : Begin
                 OldVal := Port[$2FC];
                 OldVal := OldVal And $0FD; {clear pin RTS}
                 Port[$2FC] := OldVal ;
                 End;
          Com3 : Begin
                 OldVal := Port[$3EC];
                 OldVal := OldVal And $0FD; {clear pin RTS}
                 Port[$3EC] := OldVal ;
                 End;
          Com2 : Begin
                 OldVal := Port[$2EC];
                 OldVal := OldVal And $0FD; {clear pin RTS}
                 Port[$2EC] := OldVal ;
                 End;
      End;
End;

Procedure Dec_Ascii(VAR num1:integer);
Begin
     CASE num1 OF
       0 : num2 := '0';
       1 : num2 := '1';
       2 : num2 := '2';
       3 : num2 := '3';
       4 : num2 := '4';
       5 : num2 := '5';
       6 : num2 := '6';
       7 : num2 := '7';
       8 : num2 := '8';
       9 : num2 := '9';
      10 : num2 := 'A';
      11 : num2 := 'B';
```

```
       12 : num2 := 'C';
       13 : num2 := 'D';
       14 : num2 := 'E';
       15 : num2 := 'F';
      End;
End;

Procedure Receive_Time;
Var
    data    : Byte;
    ready   : Byte;
    check   : Byte;
    B_sec   : Integer;
Begin
    Repeat
      Begin
        Regs.ah  := $03;
        Regs.dx  := Com;
        Intr($14,Regs);
        Ready    := Regs.ah;
        Delay(01);
        B_sec    := B_sec + 1;
      End;
    Until  ((Ready AND $01) = $01) OR (B_sec = 1000)
    OR KeyPressed;
    If KeyPressed Then Exit;
    If B_sec = 1000 Then Exit;
    If (Ready AND $01) = $01 Then
        Begin
           Regs.ah  := $02;
           Regs.dx  := Com;
           Intr($14,Regs);
           Check    := Regs.ah;
           Data     := Regs.al;
           If (check ANd $80) = $80  Then  Writeln('ERROR');
           If (Char(data)) <> #01 Then Exit;
           If (Chr(data)) = #01 Then
             Begin
               RxData[1] := (Chr(data));
               i  := 2 ;
               Repeat
                 Begin
                   Repeat
                    Regs.ah  := $03;
                    Regs.dx  := Com;
                    Intr($14,Regs);
                    Ready    := Regs.ah;
                    Delay(01);
                    B_sec    := B_sec + 1;
                   Until  ((Ready AND $01) = $01)
                     OR (B_sec = 500) OR KeyPressed;
                  If KeyPressed Then Exit;
                  If B_sec = 500 then Exit;
                  If (Ready AND $01) = $01 Then
                       Begin
                         Regs.ah  := $02;
                         Regs.dx  := Com;
```

```
                    Intr($14,Regs);
                    Check     := Regs.ah;
                    Data      := Regs.al;
                    If (check ANd $80) = $80
                      Then  Writeln('ERROR');
                    RxData[i] := (Chr(data));
                    i := i+1 ;
                  End;
            End;
          Until ((Chr(data)) = #04) OR (B_sec = 500)
             OR KeyPressed;
          If ((Chr(data)) = #04) Then
              Begin
                    Delay(300);
{clear buffer RX}   Regs.ah  := $03;
                    Regs.dx  := Com;
                    Intr($14,Regs);
                    Ready    := Regs.ah;
                    If (Ready AND $01) = $01 Then
                       Begin
                           Regs.ah  := $02;
                           Regs.dx  := Com;
                           Intr($14,Regs);
                       End;
                    h := 1 ;
{check name}        If (RxData[6]) <> (Row1[2])
                      Then Exit;
                    If (RxData[7]) <> (Row1[3])
                      Then Exit;
                    If (RxData[8]) <> (Row1[4])
                      Then Exit;
{check RP}          If ((RxData[10]) <> #32) AND
                       ((RxData[11]) <> #32) AND
                       ((RxData[12]) <> #32) AND
                       ((RxData[13]) <> #58)
                       Then Exit;
{check sum}         i := i - 5 ;
                    Bcc := 0;
                    p := i ;
                    i := 1 ;
                    Repeat
                       Bcc := Bcc+Ord(RxData[i]);
                       i := i+1;
                    Until p = i;
                    While Bcc > 256 Do
                    Bcc := Bcc-256;
                    Bcc := 256-Bcc;
                    r1 := Bcc Div 16;
                    Dec_Ascii(r1);
                    If RxData[i+1] <> Num2 Then Exit;
                    r2 := Bcc Mod 16;
                    Dec_ascii(r2);
                    If RxData[i+2] <> Num2 Then Exit;

                    i := p + 5 ;
                        Repeat
                          Write(RxData[h]);
```

```
                                        Delay(2);
                                        h := h + 1 ;
                                      Until h = i ;
                                      WriteLn;
                        End;
                End;
        End;
end;

Procedure Tranrece;
Begin
     ClrScr;
     TextBackground(15);
     ClrScr;
     Window(3,2,78,2);
     TextBackground(0);
     ClrScr;
     GotoXY(30,1);Write('Data To Transmit');
     Window(3,4,78,7);
     TextBackground(0);
     ClrScr;
     Window(3,9,78,9);
     TextBackground(0);
     ClrScr;
     GotoXY(30,1);Write('Data To Receive');
     Window(3,11,78,22);
     TextBackground(0);
     ClrScr;
     Check := '~';
   Repeat
      Window(3,11,78,22);
      gotoXY(1,12);
      WriteLn;
        Repeat
            Receive_time;
        Until KeyPressed;
   {  GoToXY(1,9);ClrEol; }
   Lead := Readkey;
   If Lead <> #27 Then
                    Begin
                        Bcc := 0;
                        Window(3,4,78,7);
                        GotoXY(1,4);
                        Begin
                             Write(Lead);
                             Row1[14] := Lead;
                        i := 15;
                        l := 1;
                        D := 1;
                        Repeat
                              Lead := Readkey;
                              Write(Lead);
                              Row1[i] :=Lead;
                              If Lead = #27 Then  Begin
                                      Window(1,1,80,25);
                                      Exit
                                   End;
```

```
                                If Lead = #13 Then
                                   Begin
                                    p := i ;
                                    i := 1 ;
                                    Repeat
                                     Bcc := Bcc+Ord(Row1[i]);
                                     i := i+1;
                                    Until p = i;
                                    While Bcc > 256 Do
                                     Bcc := Bcc-256;
                                     Bcc := 256-Bcc;
                                     r1 := Bcc Div 16;
                                        Dec_Ascii(r1);
                                        Row1[i]   := '"';
                                        Row1[i+1] := Num2;
                                          r2 := Bcc Mod 16;
                                          Dec_ascii(r2);
                                        Row1[i+2] := Num2;
                                        Row1[i+3] := '"';
                                        Row1[i+4] := #04;
                                        i := 76;
                                   End;
                                If Lead = #08 Then
                                   Begin
                                        i := i-1;
                                        If i <= 0 Then i := 1;
                                        ClrEol;
                                        Row1[i] := ' ';
                                        If Row1[1] = ' ' Then
                                           Begin
                                            ClrEol;
                                            D := 1;
                                            i := 0;
                                            Repeat
                                            D := D+1;
                                            If KeyPressed
                                              Then D:=9000;
                                            Until (d = 8000)
                                              Or (D = 9000);
                                           End;
                                        l := l+1;
                                   End;
                                i := i+1;
                Until (i >76) Or (D = 8000);
While D <> 8000 Do
      Begin
           WriteLn;
           SetRts(Com); Delay(1200);
           For i:=1 To 76 Do
               Begin
                    Out := SendSerial(Row1[i]);
                    If out = -1 Then  Begin
                             Lead := #27;
                             Write('Error');
                       End;
                    If Row1[i] = #13 Then i := 76;
                End;
```

```
                    ClrRts(Com);
                    D := 8000;
                End;
            End;
        End;
    window(1,1,80,25);
    Until Lead = #27;
End;

Procedure Menu01;
Begin
     ClrScr;
     WriteLn;
     WriteLn;
     WriteLn('          1)   300,N,8,1');
     WriteLn('          2)  1200,N,8,1');
     WriteLn('          3)  2400,N,8,1');
     WriteLn('          4)  4800,N,8,1');
     WriteLn('          5)  9600,N,8,1');
     WriteLn;
     Write('        Select Baud Rate(1-5):');
     Repeat
          J := Readkey;
          If J = #27 then exit;
     Until J In ['1'..'5'];
     WriteLn(J);
         Case J Of
             '1' : Begin
                    Initbyte:= $43;
                    Msg := 'Baud Rate =  300,N,8,1';
                    Window(3,24,40,24);TextBackground(0);
                    ClrScr; GotoXY(10,1);Write(Msg);
                    End;
             '2' : Begin
                    Initbyte:= $83;
                    Msg := 'Baud Rate = 1200,N,8,1';
                    Window(3,24,40,24);TextBackground(0);
                    ClrScr; GotoXY(10,1);Write(Msg);
                   End;
             '3' : Begin
                    Initbyte:= $A3;
                    Msg := 'Baud Rate = 2400,N,8,1';
                    Window(3,24,40,24);TextBackground(0);
                    ClrScr; GotoXY(10,1);Write(Msg);
                   End;
             '4' : Begin
                    Initbyte:= $C3;
                    Msg := 'Baud Rate = 4800,N,8,1';
                    Window(3,24,40,24);TextBackground(0);
                    ClrScr; GotoXY(10,1);Write(Msg);
                   End;
              '5': Begin
                    Initbyte:= $E3;
                    Msg := 'Baud Rate = 9600,N,8,1';
                    Window(3,24,40,24);TextBackground(0);
                    ClrScr; GotoXY(10,1);Write(Msg);
                   End;
```

```
                End;
            InitSerial(Com,Initbyte);
End;

Procedure Menu02;
Begin
     ClrScr;
     WriteLn;
     WriteLn('            1)    Com1');
     WriteLn;
     WriteLn('            2)    Com2');
     WriteLn;
     WriteLn('            3)    Com3');
     WriteLn;
     WriteLn('            4)    Com4');
     WriteLn;
     Write('          Select port(1-4) :');
     Repeat
           C := Readkey;
           If C = #27 Then Exit;
           Until C In ['1'..'4'];
           WriteLn(C);
           Case C Of
             '1' : Begin
                    Com:= Com1;
                    Msg01:='Serial Port = Com1';
                    Window(40,24,78,24);TextBackground(0);
                    ClrScr;GotoXY(5,1);Write(Msg01);
                   End;
             '2' : Begin
                    Com:= Com2;
                    Msg01:='Serial Port = Com2';
                    Window(40,24,78,24);TextBackground(0);
                    ClrScr;GotoXY(5,1);Write(Msg01);
                   End;
             '3' : Begin
                    Com:= Com3;
                    Msg01:='Serial Port = Com3';
                    Window(40,24,78,24);TextBackground(0);
                    ClrScr;GotoXY(5,1);Write(Msg01);
                   End;
             '4' : Begin
                    Com:= Com4;
                    Msg01:='Serial Port = Com4';
                    Window(40,24,78,24);TextBackground(0);
                    ClrScr;GotoXY(5,1);Write(Msg01);
                   End;
              End;
          InitSerial(com,Initbyte);
     End;

Procedure Menu03;
Begin
     ClrScr;
     WriteLn;
     WriteLn;
     WriteLn('       1)  Transmit And Receive');
```

```
      WriteLn;
      WriteLn;
      WriteLn;
      WriteLn;
      WriteLn;
      Write('       Press " 1 " For Run Program ');
      Repeat
      s:=Readkey;
      If S = #27 Then Exit;
      Until S In ['1'];
      WriteLn(s);
        Case S Of
           '1': TranRece;
        End;
End;

Procedure Menu04;
Begin
      ClrScr;
      WriteLn;
      WriteLn;
      WriteLn('      1)  Set Control  Station Name ');
      Writeln('      2)  Set Target   Station Name ');
      WriteLn('      3)  Set Repeater Station Name ');
      WriteLn;
      WriteLn;
      WriteLn;
      WriteLn;
      Write('  Press " 1-3 " For Set Station Name ');
      Repeat
      s:=Readkey;
      If S = #27 Then Exit;
      Until S In ['1'..'3'];
        Case S Of
         '1' : Begin
                 Window(23,19,60,19);TextBackground(0);
                 ClrScr;Write('      ');
                 Write(Msg02);
                 S:=Readkey;
                 Write(S);
                 Row1[2] := S;
                 S:=Readkey;
                 Write(S);
                 Row1[3] := S;
                 S:=Readkey;
                 Write(S);
                 Row1[4] := S;
               End;
         '2' : Begin
                 Window(23,20,60,20);TextBackground(0);
                 ClrScr;Write('      ');
                 Write(Msg03);
                 S:=Readkey;
                 Write(S);
                 Row1[6] := S;
                 S:=Readkey;
                 Write(S);
```

```
                Row1[7] := S;
                S:=Readkey;
                Write(S);
                Row1[8] := S;
              End;
        '3' : Begin
                Window(23,21,60,21);TextBackground(0);
                ClrScr;Write('      ');
                Write(Msg04);
                S:=Readkey;
                Write(S);
                Row1[10] := S;
                S:=Readkey;
                Write(S);
                Row1[11] := S;
                S:=Readkey;
                Write(S);
                Row1[12] := S;
              End;

      End;
End;
Procedure Menu00;
Begin
  ClrScr;
  TextBackground(15);
  ClrScr;
  Window(3,2,78,2);
  TextBackground(0);
  ClrScr;
  GotoXY(33,1);Write('MAIN MENU');
  Window(23,18,60,22);
  TextBackground(0);
  ClrScr;
  WriteLn;
  Write('      ');Write(Msg02);
  Write(Row1[2]);Write(Row1[3]);WriteLn(Row1[4]);
  Write('      ');Write(Msg03);
  Write(Row1[6]);Write(Row1[7]);WriteLn(Row1[8]);
  Write('      ');Write(Msg04);
  Write(Row1[10]);Write(Row1[11]);WriteLn(Row1[12]);
  Window(3,24,39,24);
  TextBackground(0);
  ClrScr;
  GotoXY(10,1);Write(Msg);
  Window(40,24,78,24);
  TextBackground(0);
  ClrScr;
  GotoXY(5,1);Write(Msg01);
  Window(23,5,60,15);
  TextBackground(0);
  ClrScr;
  WriteLn;
  WriteLn;
  WriteLn('         1) Select Baud Rate');
  WriteLn('         2) Select Port');
  WriteLn('         3) Select Mode');
```

```
  WriteLn('         4) Select Station Name');
  WriteLn('         5) Exit');
  WriteLn;
  WriteLn('         Select Number (1-5) :');
  WriteLn;
    Repeat
    k:=Readkey;
    If k = #27 Then k := '5';
  Until k In ['1'..'5'];
  WriteLn(k);
    Case k Of
       '1' : Begin Menu01;window(1,1,80,25);Clrscr;end;
       '2' : Menu02;
       '3' : Menu03;
       '4' : Menu04;
       '5' : Exit;

    End;
End;

 {main}

Begin
     ClrScr;
    Msg := 'Baud Rate = 1,200,N,8,1';
    Initbyte := $83;
    Msg01 := 'Serial Port = Com1';
    Msg02 := 'Control  Station Name :  ';
    Msg03 := 'Target   Station Name :  ';
    Msg04 := 'Repeater Station Name :  ';
       Row1[1]  := #01;
       Row1[2]  := #69;
       Row1[3]  := #68;
       Row1[4]  := #71;
       Row1[5]  := #62;
       Row1[6]  := #80;
       Row1[7]  := #80;
       Row1[8]  := #72;
       Row1[9]  := #44;
       Row1[10] := #82;
       Row1[11] := #57;
       Row1[12] := #57;
       Row1[13] := #00;
    Com := Com1;
    InitSerial(Com,Initbyte);
    Repeat
      Menu00;
    Until k = '5';
    Window(1,1,80,25);
    ClrScr;
    WriteLn('Thanks For Used Program (Zender10)');
    Write('From KMIT''L Appiled Physics Solidy #10');
    WriteLn(' (1994-1995)');
    WriteLn('');
End.
```

DATA SHEET 80C51-L / 80C31-L

CMOS SINGLE-CHIP 8 BIT 3V-MICROCONTROLLER

- 80C51-L - CMOS SINGLE-SHIP 8-BIT MICROCONTROLLER with factory mask-programmable ROM
- 80C31-L - CMOS SINGLE-CHIP 8-BIT CONTROL-ORIENTED CPU with RAM and I/O
- 80C51-L/C31-L: 0 TO 6 MHz, VCC = 2.7V TO 6V

FEATURES

- POWER CONTROL MODES
- 128 x 8 BIT RAM
- 32 PROGRAMMABLE I/O LINES
- TWO 16-BIT TIMER/COUNTERS
- 64K PROGRAM MEMORY SPACE
- FULLY STATIC DESIGN
- HIGH PERFORMANCE SAJI VI CMOS PROCESS
- BOOLEAN PROCESSOR
- 5 INTERRUPT SOURCES
- PROGRAMMABLE SERIAL PORT
- 64K DATA MEMORY SPACE
- TEMPERATURE RANGE: 0 TO 70°C

DESCRIPTION

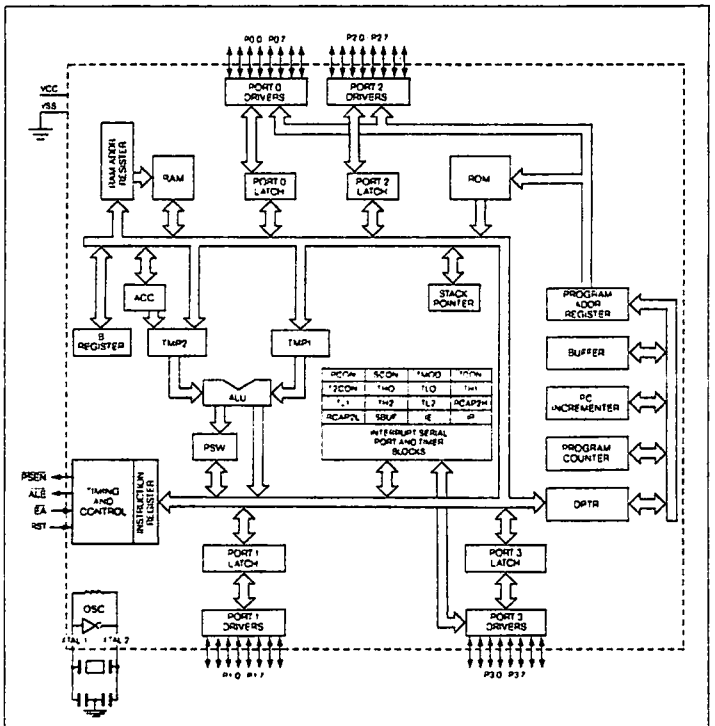


Figure 1. Block Diagram

MHS's 80C51 and 80C31 are high performance CMOS versions of the 8051/8031 NMOS single chip 8 bit μC and is manufactured using a self-aligned silicon gate CMOS process (SAJI VI).

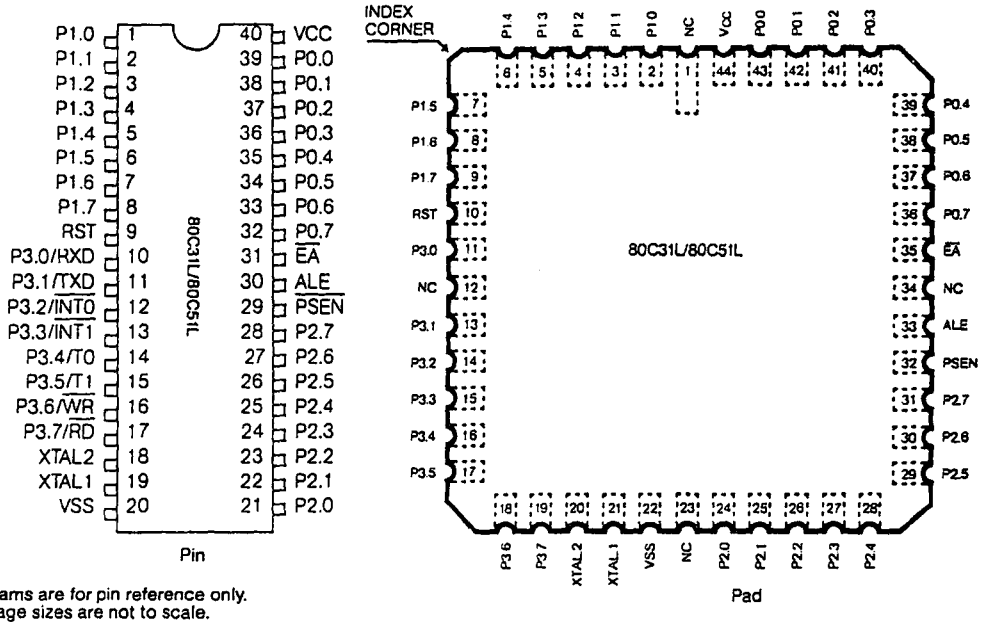
The fully static design of the MHS 80C51/80C31 allows to reduce system power consumption by bringing the clock frequency down to any value, even DC, without loss of data.

The 80C51 retains all the features of the 8051: 4K bytes of ROM; 128 bytes of RAM; 32 I/O lines; two 16 bit timers; a 5-source 2-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuits.

In addition, the 80C51 has two software-selectable modes of reduced activity for further reduction in power consumption. In the Idle Mode the CPU is frozen while the RAM, the timers, the serial port, and the interrupt system continue to function. In the Power Down Mode the RAM is saved and all other functions are inoperative.

The 80C31 is identical to the 80C51 except that it has no on-chip ROM.

Figure 2. Configurations



Diagrams are for pin reference only.
Package sizes are not to scale.

IDLE AND POWER DOWN OPERATION

Figure 3 shows the internal Idle and Power Down clock configuration. As illustrated, Power Down operation stops the oscillator. Idle mode operation allows the interrupt, serial port, and timer blocks to continue to function while the clock to the CPU is gated off.
These special modes are activated by software via the Special Function Register, Its hardware address is 87H. PCON is not bit addressable.

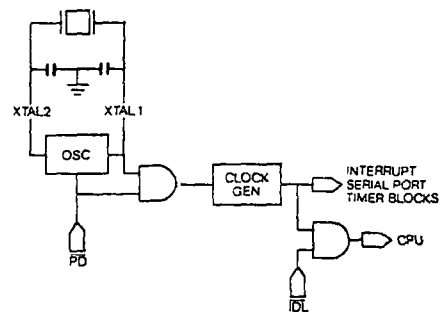


Figure 3. Idle and Power Down Hardware

PCON: Power Control Register

(MSB) (LSB)

SMOD	-	-	-	GF1	GF0	PD	IDL

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.
-	PCON.6	(Reserved)
-	PCON.5	(Reserved)
-	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Ide mode bit. Setting this bit activates idle mode operation.

If 1's are written to PD and IDL at the same time. PD takes precedence. The reset value of PCON is (OXXX0000).

Table 1. Status of the external pins during Idle and Power Down modes

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Port Data	Port Data	Port Data	Port Data
Idle	External	1	1	Floating	Port Data	Address	Port Data
Power Down	Internal	0	0	Port Data	Port Data	Port Data	Port Data
Power Down	External	0	0	Floating	Port Data	Port Data	Port Data

IDLE MODE

The instruction that sets PCON.0 is the last instruction executed before the Idle mode is activated. Once in the Idle mode the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all other registers maintain their data during Idle. *Table 1* describes the status of the external pins during Idle mode.

There are two ways to terminate the Idle mode. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating Idle mode. The interrupt is serviced, and following RETI, the next instruction to be executed will be the one following theinstruction that wrote a 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an enabled interrupt, the service routine can examine the status of the flag bits.

The second way of terminating the Idle mode is with a hardware reset. Since the oscillator is still running, the hardware reset needs to be active for only 2 machine cycles (24 oscillator periods) to complete the reset operation.

POWER DOWN MODE

The instruction that sets PCON.1 is the last executed prior to entering power down. Once in power down, the oscillator is stopped. The contents of the onchip RAM and the Special Function Register is saved during power down mode. A hardware reset is the only way of exiting the power down mode. The hardware reset initiates the Special Function Register *(see Table 1).*

In the Power Down mode. V_{CC} may be lowered to minimize circuit power consumption. Care must be taken to ensure the voltage is not reduced until the power down mode is entered, and that the voltage is restored before the hardware reset is applied which frees the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

Table 1 describes the status of the external pins while in the power down mode. It should be noted that if the power down mode is activated while in external program memory, the port data that is held in the Special Function Register P2 is restored to Port 2. If the data is a 1, the port pin is held high during the power down mode by the strong pullup, T1, shown in *Figure 4.*

STOP CLOCK MODE

Due to static design, the MHS 80C31/C51 clock speed can be reduced until 0 MHz without any data loss in memory or registers. This mode allows step by step utilization, and permits to reduce system power consumption by bringing the clock frequency down to any value. At 0 MHz, the power consumption is the same as in the Power Down Mode.

80C51 I/O PORTS

The I/O port drive of the 80C51 is similar to the 8051. The I/O buffers for Ports 1, 2, and 3 are implemented as shown in *figure 4.*

When the port latch contains a 0, all pFETS in *figure 4* are off while the nFET is turned on. When the port latch makes a 0-to-1 transition, the nFET turns off. The strong pullup pFET, T1, turns on for two oscillator periods, pulling the output high very rapidly. As the output line is drawn high, pFET T3 turns on through the inverter to supply the I_{OH} source current. This inverter and T3 form a latch which holds the 1 and is supported by T2. When Port 2 is used as an address port, for access to external program of data memory, any address bit that contains a 1 will have his strong pullup turned on for the entire duration of the external memory access.

When an I/O pin on Ports 1, 2, or 3 is used as an input, the user should be aware that the external circuit must sink current during the logical 1-to-0 transition. The maximum sink current is specified as ITL under the D.C. Specifications. When the input goes below approximately 2V, T3 turns off to save ICC current. Note, when returning to a logical 1, T2 is the only internal pullup that is on. This will result in a slow rise time if the user's circuit does not force the input line high.

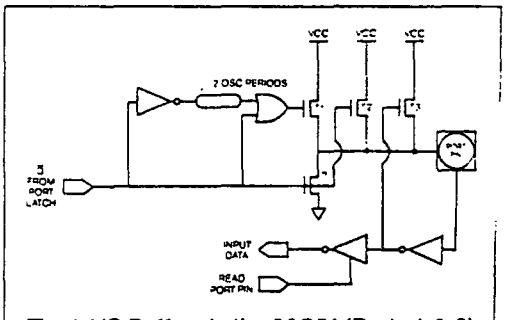


Fig. 4. I/O Buffers in the 80C51 (Ports 1, 2, 3)

80C51 PIN DESCRIPTIONS

V_{SS}

Circuit ground potential

V_{CC}

Supply voltage during normal, Idle, and Power Down operation.

Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have 1's written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1's. Port 0 also outputs the code bytes during program verification in the 80C51. External pullups are required during program verification. Port 0 can sink eight LS TTL inputs.

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. Port 1 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification. In the 80C51, Port 1 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pullups. Port 2 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups. Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @ DPTR). In this application, it uses strong internal pullups when emitting 1's. During accesses to external Data Memory that uses 8-bit addresses (MOVX @ Ri), Port 2 emits the contents of the P2 Special Function Register.

It also receives the high-order address bits and control signals during progam verification in the 80C51. Port 2 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pullups. Port 3 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below.

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

Port 3 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

RST

A high level on this for two machine cycles while the oscillator is running resets the device. An internal pull-down resistor permits Power-On reset using only a capacitor connected to V_{CC}.

ALE

Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated as though for this purpose at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pullup.

$\overline{PSEN}$

Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory, two activations of PSEN are skipped during each access to external Data Memory). PSEN is not activated during fetches from internal Program Memory. PSEN can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pullup.

$\overline{EA}$

When EA is held high, the CPU executes out of internal Program Memory (unless the Program Counter exceeds OFFFH). When EA is held low, the CPU executes only out of external Program Memory. EA must not be floated.

XTAL1

Input to the inverting amplifier that forms the oscillator. Receives the external oscillator signal when an external oscillator is used.

XTAL2

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. This pin should be floated when an external oscillator is used.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output respectively, of an inverting amplifier which is configured for use as an on-chip oscillator, as shown in figure 5. Either a quartz crystal or ceramic resonator may be used.
To drive the device from an external clock source, XTAL1 should be driven while XTAL2 is left unconnected as shown in *figure 6*. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

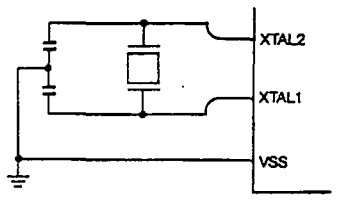


Figure 5. Crystal Oscillator

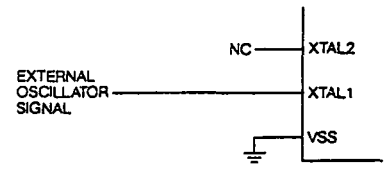


Figure 6. External Drive Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias:
- Commercial ..... 0°C to 70°C
- Industrial ..... -40°C to 85°C

Storage Temperature ..... -65°C to +150°C

Voltage on V_{CC} to V_{SS} ..... -0.5V to +7V

Voltage on Any Pin to V SS..... -0.5V to V_{CC} + 0.5V

Power Dissipation 1W*

*This value is based on the maximum allowable die temperature and the thermal resistance of the package.

*NOTICE:

Stresses at or above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions may affect device reliability.

DC CHARACTERISTICS

TA = -40°C to 85°C; VCc = 2.7V to 6V; VSS = 0V; F = 0 to 6 MHz

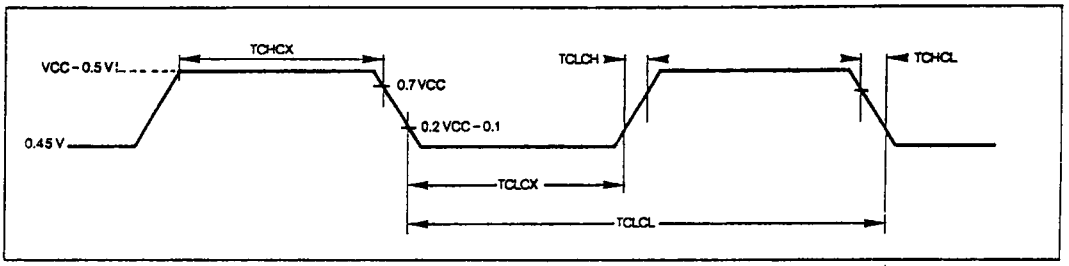
Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	$0.2V_{CC}$ -0.1	V	
VIH	Input High Voltage (Except XTALs and RST)	$0.2V_{CC}$ +0.9	V_{CC} +0.5	V	
VIH1	Input High Voltage to RST for Reset	$0.7V_{CC}$	V_{CC} +0.5	V	
VIH2	Input High Voltage To XTAL 1	$0.7V_{CC}$	V_{CC} +0.5	V	
VPD	Power Down Voltage To V_{CC} in PD Mode	2.0	6.0	V	
VOL	Output Low Voltage (Ports 1, 2, 3)		0.45	V	IOL = 1.6mA (note 1)
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$		0.45	V	IOL = 3.2mA (note 1)
VOH	Output High Voltage Ports 1, 2, 3	$0.9V_{CC}$		V	IOH = -10µa
		2.4		V	IOH = -60µA V_{CC} = 5V ± 10%
VOH1	Output High Voltage (Port 0 in External in External Bus Mode), ALE, $\overline{PSEN}$	$0.9V_{CC}$		V	IOH = -40µA
		2.4		V	IOH = -400µA V_{CC} = 5V ± 10%
IIL	Logical 0 Input Current Ports 1,2,3		-50	µA	Vin = 0.45V
ILI	Input Leakage Current		± 10	µA	0.45 < Vin < V_{CC}
ITL	Logical 1 to 0 Transition Current (Ports 1, 2, 3)		-500	µA	Vin = 2.0V
ICCPD	Power Supply Current (Power Down Mode)	50	10	µA	V_{CC} = 2.0V to 5.5V (note 2)
RRST	RST Pulldown Resistor	50	150	kΩ	
CIO	Capacitance of I/O Buffer		10	pF	f_C = 1MHz, T_A = 25°C

Note 1:

Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V_{OL}S of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading 100 pF), the noise pulse on the ALE line may exceed 0.45V with maxi V_{OL} peak 0.6V. A Schmitt Trigger use is not necessary.

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL 1)

Symbol	Parameter	Variable Clock freq = 0 to 6 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	166		ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS

(T_A = -40°C to 85°C, V_{CC} = 2.7V to 6V, V_{SS} = 0V)
(Load Capacitance for Port 0, ALE, and $\overline{PSEN}$ = 100pf; Load Capacitance for All Other Outputs = 80pf).

EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
TLHLL	ALE Pulse Width	2TCLCL - 40		ns
TAVLL	Address Valid to ALE	TCLCL - 55		ns
TLLAX	Address Hold After ALE	TCLCL - 35		ns
TLLIV	ALE to Valid instr In		4TCLCL - 170	ns
TLLPL	ALE to $\overline{PSEN}$	TCLCL - 25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	3TCLCL - 35		ns
TPLIV	$\overline{PSEN}$ to Valid Instr In		3TCLCL - 220	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		TCLCL - 20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	TCLCL - 8		ns
TAVIV	Address to Valid Instr In		5TCLCL - 220	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		0	ns

See next page for External Data Memory Characteristics.

EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
TRLRH	$\overline{RD}$ Pulse Width	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	6TCLCL-100		ns
TLLAX	Data Address Hold After ALE	TCLCL-35		ns
TRLDV	$\overline{RD}$ to Valid Data In		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns
TRHDZ	Data Float After $\overline{RD}$		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		8TCLCL-150	ns
TAVDV	Address to Valid Data In		9TCLCL-165	ns
TLLWL	ALE to $\overline{WR}$ or $\overline{RD}$	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{WR}$ or $\overline{RD}$	4TCLCL-130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	TCLCL-60		ns
TQVWH	Data Setup to $\overline{WR}$ High	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	TCLCL-50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	TCLCL-40	TCLCL-40	ns

MAXIMUM I_{CC} (mA)

	Operating (Note 3)			Idle (Note 4)		
Freq. V_{CC}	2.7V	5V	6V	2.7V	5V	6V
1 MHz	0.8 mA	1.5 mA	1.8 mA	400 µA	800 µA	1 mA
6 MHz	4 mA	8 mA	10 mA	1.2 mA	3.5 mA	3.8 mA

Note 2:

Power Down I_{CC} is measured will all output pins disconnected; EA = Port 0 = V_{CC}; XTAL2 N.C.; RST = V_{SS}

Note 3:

I_{CC} is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 5 ns, $V_{IL} = V_{SS}$ + 0.5V; V_{IH} V_{CC} - 0.5V; XTAL2 N.C.; EA = RST = Port 0 = V_{CC}. I_{CC} would be slightly higher if a crystal oscillator used.

Note 4:

Idle I_{CC} is measured with all output pins disconnected; XTAL1 driven TCLCH, TCHCL = 5 ns, V_{IL} = V_{SS} + 0.5V; V_{IH} = V_{CC} - 0.5V; XTAL2 N.C.; Port 0 = V_{CC}; EA = RST = V_{SS}.

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has 5 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list all the characters and what they stand for.

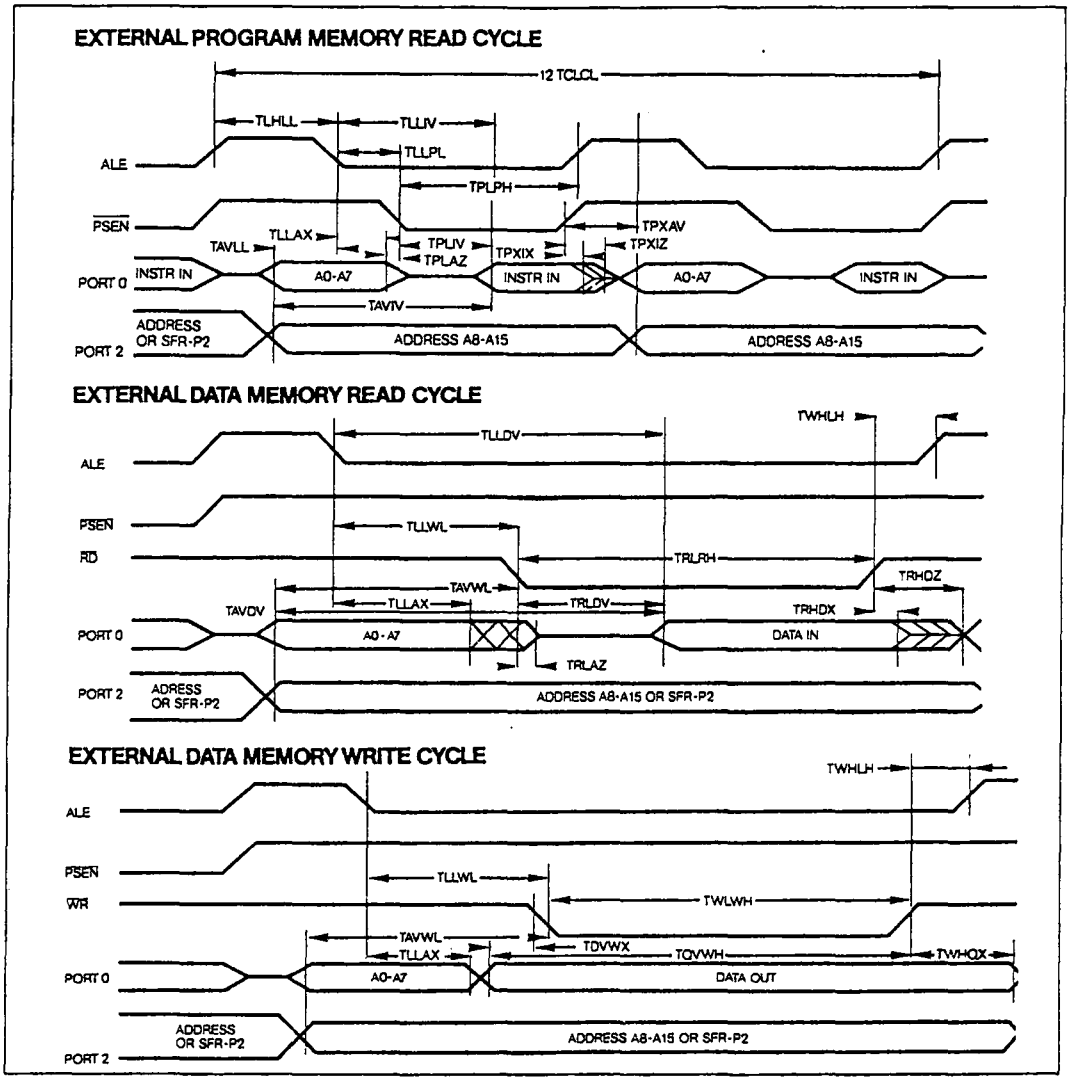
EXAMPLE:

TAVLL = Time for Address Valid to ALE low.
TLLPL = Time for ALE low to $\overline{PSEN}$ low.

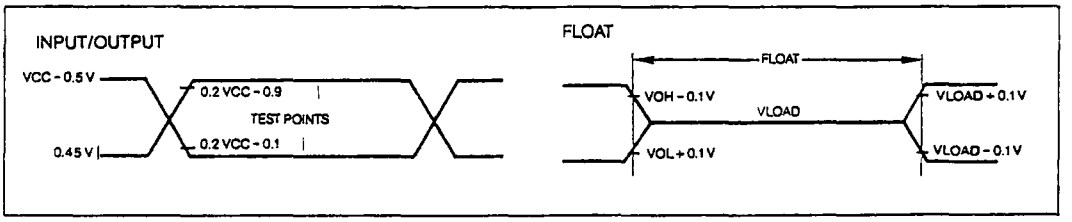
A: Address.
C: Clock.
D: Input data.
H: Logic level HIGH.
I: Instruction (program memory contents).
L: Logic level LOW, or ALE.
P: PSEN
Q: Output data.
R: READ signal.
T: Time.
V: Valid.
W: WRITE signal
X: No longer a valid logic level.
Z: Float.

MHS

AC TIMING DIAGRAMS



AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at VCC – 0.5 for a logic "1" and 0.45V for a logic "0". Timing measurements are made at VIH min for a logic "1" and VIL max for a logic "0". For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs and begins to float when a 100 mV change from the loaded VOH/VOL level occurs. Iol/IoH ≥ ± 20 Ma.

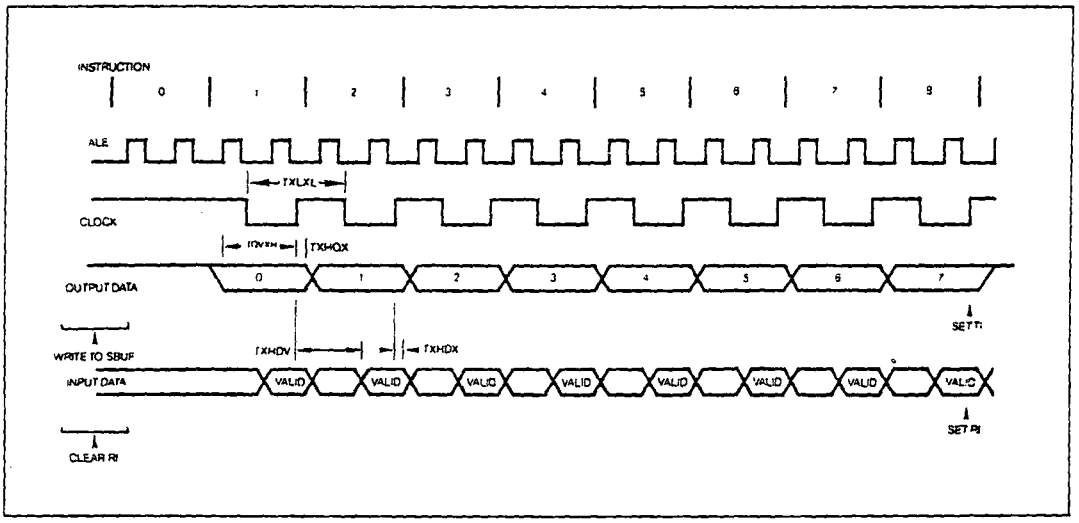
SERIAL PORT TIMING - SHIFT REGISTER MODE

A.C. CHARACTERISTICS:

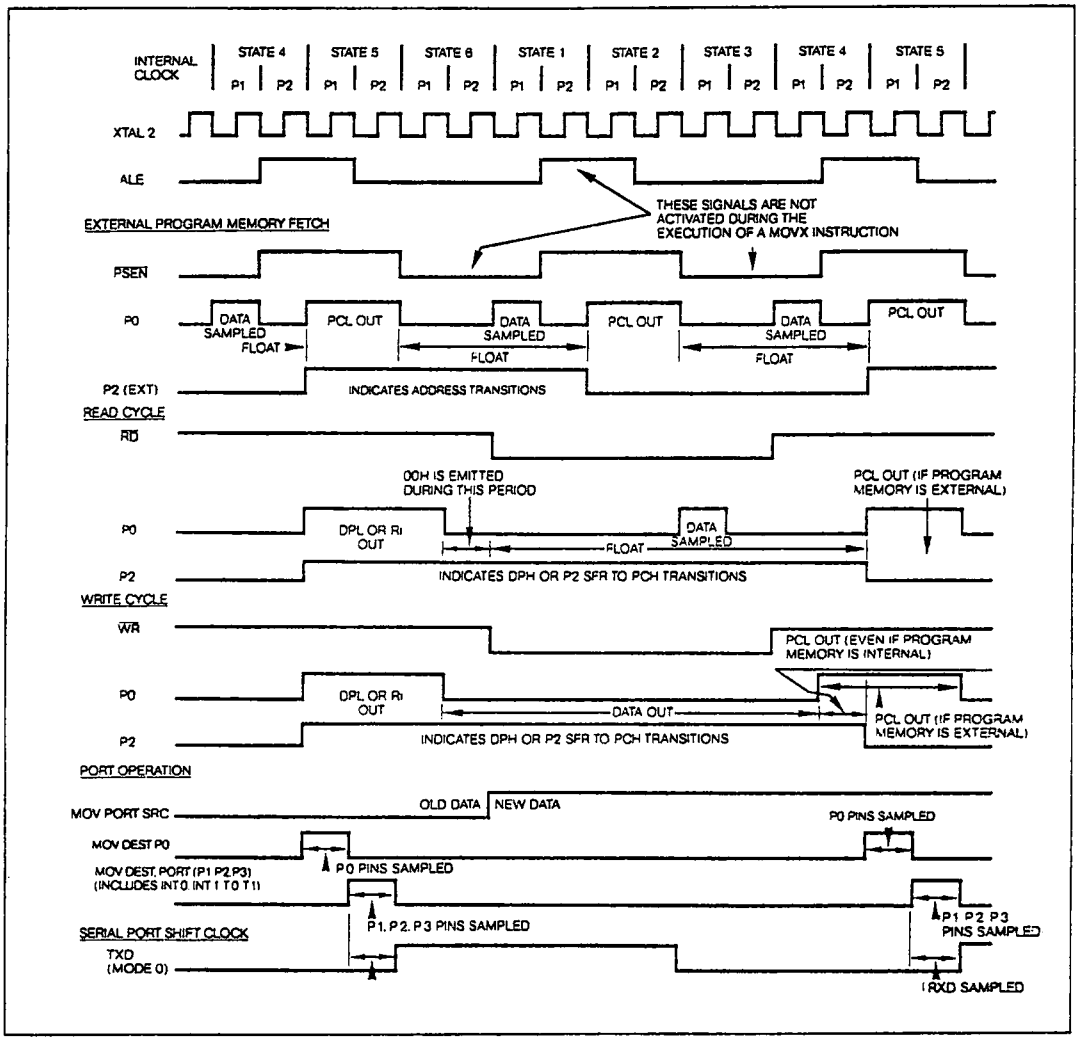
(T_A = 0 °C to 70 °C; V_{SS} = 0V; V_{CC} = 2.7V to 6V; Load Capacitance = 80 pF)

Symbol	Parameter	Min	Max	Units
TXLXL	Serial Port Clock Cycle Time	12TCLCL		µs
TQVXH	Output Data Setup to Clock Rising Edge	10TCLCL-133		ns
TXHQX	Output Data Hold After Clock Rising Edge	2TCLCL-117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		ns
TXHDV	Clock Rising Edge to Input Data Valid		10TLCL-133	ns

SHIFT REGISTER TIMING WAVEFORMS



CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component. Typically though ($T_A = 25°C$ fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed. data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Incriment indirect RAM	1	1
INC	DPTR	Incriment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

Table 1. (Cont.)

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Move immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data 16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte form stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order nibble ind RAM with A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr 11	Absolute Subroutine Call	2	2
LCALL	addr 16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr 11	Absolute Jump	2	2
LJMP	addr 16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2

Table 1. (Cont.)

PROGRAM AND MACHINE CONTROL (cont.)

Mnemonic		Description	Byte	Cyc
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp. immed. to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data. rel	Comp. immed. to ind. & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct. rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn	– Working register R0-R7
direct	– 128 internal RAM locations, any I/O port, control or status register
@Ri	– Indirect internal RAM location addressed by register R0 or R1
#data	– 8-bit constant included in instruction
#data 16	– 16-bit constant included as bytes 2 & 3 of instruction
bit	– 128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr 16	– Destination address for LCALL & LJMP may be anywhere within the 64-k program memory address space
Addr 11	– Destination address for ACALL & AJMP will be within the same 2-k page of program memory as the first byte of the following instruction
rel	– SJMP and all conditional jumps include an 8-bit offset byte. Range is +127–128 bytes relative to first byte of the following instruction.

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr,code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr,code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr,code addr
31	2	ACALL	code addr
32	1	RETI	

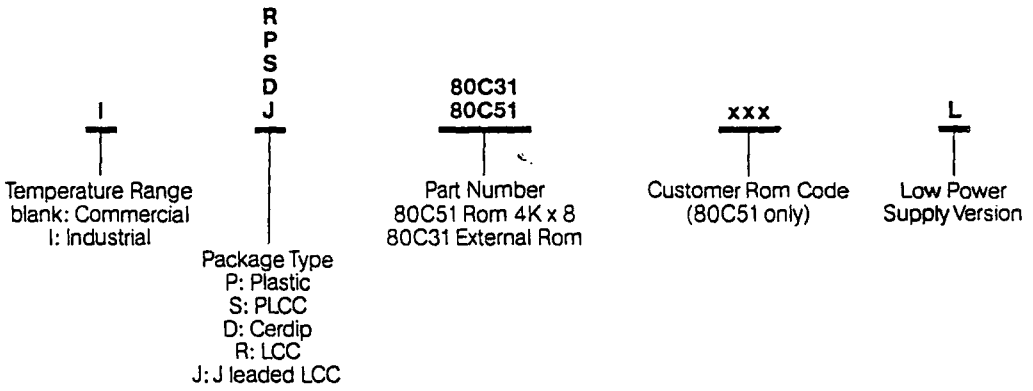
Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A,@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C, bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr,data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

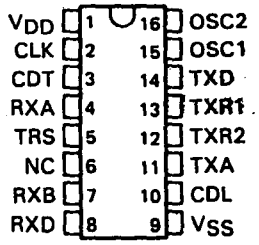
Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A



D2662, NOVEMBER 1985—REVISED APRIL 1986

- Single-Chip Frequency-Shift-Keying (FSK) Modem
- Meets Both Bell 202 and CCITT V23 Specifications
- Transmit Modulation at 75, 150, 600, and 1200 Baud
- Receive Demodulation at 5, 75, 150, 600, and 1200 Baud
- Half-Duplex Operation Up to 1200 Baud Transmit and Receive
- Full-Duplex Operation Up to 1200 Baud Transmit and 150 Baud Receive
- On-Chip Group Delay Equalization and Transmit/Receive Filtering
- Carrier-Detect-Level Adjustment and Carrier-Fail Output
- Single 5-V Power Supply
- Low Power Consumption
- Reliable CMOS Silicon Gate Technology

J DUAL-IN-LINE PACKAGE
(TOP VIEW)



NC—No internal connection

Caution. These devices have limited built-in gate protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

description

The TCM3105 is a single-chip asynchronous Frequency Shift Keying (FSK) voiceband modem that uses silicon gate CMOS technology to implement a switched capacitor architecture. It is pin selectable (TXR1, TXR2, and TRS inputs) for a wide range of transmit/receive baud rates and is compatible with the applicable BELL 202 or CCITT V23 standards. Operation is fully reversible, thereby allowing both forward and backward channels to be used simultaneously.

The transmitter is a programmable frequency synthesizer that provides two output frequencies (on TXA), representing the 'marks' and 'spaces' of the digital signal present on the TXD input.

The receive section is responsible for the demodulation of the analog signal appearing at the RXA input and is based on the principle of frequency-to-voltage conversion. This section contains a group delay equalizer (to correct phase distortion), automatic gain control, carrier detect level adjustment, and bias distortion adjustment, thereby optimizing performance and giving the lowest possible bit error rate.

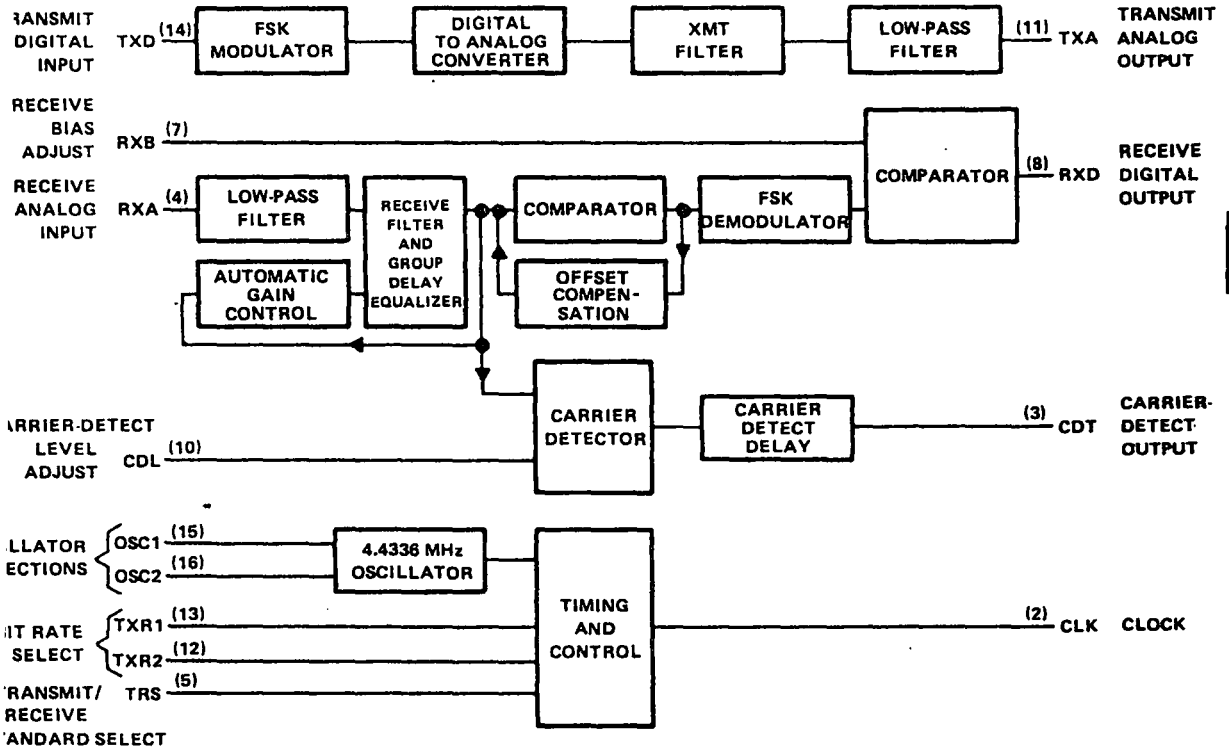
Carrier-detect information is given to the system by means of the carrier-detect circuits, which set a flag on the CDT output if the level of received in-band energy falls below a value set on the CDL input for a specified minimum duration.

The TCM3105JE is characterized for operation from −40°C to 85°C. The TCM3105JL is characterized for operation from 0°C to 70°C.

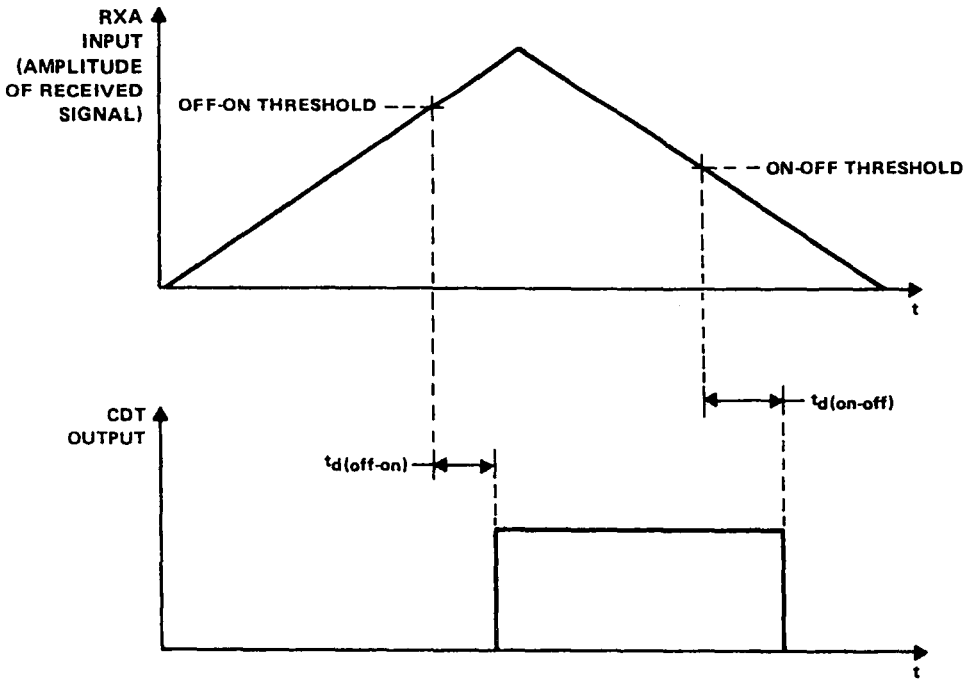
PIN FUNCTIONAL DESCRIPTION

PIN NO.	PIN NAME	DESCRIPTION
1	V_{DD}	Positive supply voltage
2	CLK	Output for a continuous clock signal at 16 times the highest selected (transmit or receive) bit rate
3	CDT	Carrier-Detect Output. A low-level output indicates carrier failure
4	RXA	Receive Analog Input to which the received line signal must be ac coupled
5	TRS	Transmit/Receive Standard Select Input, which, with TXR1 and TXR2, sets the standard bit rates and mark/space frequencies
6	NC	No internal connection
7	RXB	Receive Bias Adjust for external adjustment of the decision threshold of the final comparator to minimize bias distortion
8	RXD	Receiver Digital Output for the demodulated received data in positive logic. The high logic level is a mark and the low logic level is a space.
9	V_{SS}	Most negative supply voltage (normally ground); connected to substrate
10	CDL	Carrier Detect Level Adjust for external adjustment of carrier detect threshold
11	TXA	Transmit Analog Output for the modulated signal, which must be ac coupled
12	TXR2	Bit Rate Select 2 input, which, along with TXR1 and TRS, sets the bit rates and mark/space frequencies
13	TXR1	Bit Rate Select 1 input, which, along with TXR2 and TRS, sets the bit rates and mark/space frequencies
14	TXD	Transmit Digital Input for input data to the transmitter in positive logic. The high logic level is a mark and the low logic level is a space. The data can be accepted at any speed from zero to the selected speed and may be totally asynchronous.
15 16	OSC1 OSC2	Oscillator connections. The crystal (typically 4.4336 MHz) is connected to these pins. If an external clock is used, OSC2 is left open and the clock is connected to OSC1.

ional block diagram



ng diagram



2 Telecommunications Circuits

absolute maximum ratings over free-air operating temperature range (unless otherwise noted)

Supply voltage, V_{DD} (see Note 1) . . . . . −0.3 V to 10 V
Input voltage, V_I (any input) . . . . . −0.3 to V_{DD}
Operating free-air temperature range: TCM3105JE . . . . . −55°C to 85°C
TCM3105JL . . . . . −10°C to 70°C
Storage temperature range . . . . . −55°C to 150°C

NOTE 1: All voltage values are with respect to V_{SS}.

recommended operating conditions

	TCM3105JE			TCM3105JL			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{DD}	4	5	5.5	4	5	5.5	V
High-level input voltage, V_{IH}	2		V_{DD}	2		V_{DD}	V
Low-level input voltage, V_{IL}	0		0.8	0		0.8	V
Analog input level, peak-to-peak (ac coupled)		0.30	0.78		0.30	0.78	V
Clock frequency, f_{clock}	4.4334	4.4336	4.4338	4.4334	4.4336	4.4338	MHz
Analog load impedance at TXA	50			50			kΩ
Operating free-air temperature range, T_A	−40		85	0		70	°C

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

electrical characteristics over recommended ranges of supply voltage and operating free-air temp...

PARAMETER			TEST CONDITIONS		TCM3105JE			TCM3105JL			UNIT
					MIN	TYP†	MAX	MIN	TYP†	MAX	
V_{OH}	High-level output voltage	RXD, CDT, CLK	I_{OH} = −100 μA		2.4		V_{DD}	2.4		V_{DD}	V
V_{OL}	Low-level output voltage	RXD, CDT, CLK	I_{OL} = 1.6 mA		V_{SS}		0.4	V_{SS}		0.4	V
	Analog output voltage level, peak-to-peak	TXA	V_{DD} = 4 V	R_L = 50 kΩ, C_L = 100 pF		1.55			1.55		V
			V_{DD} = 5 V		1.4	1.9	2.3	1.4	1.9	2.3	
			V_{DD} = 5.5 V			2.1			2.1		
	Adjust voltage	RXB	V_{DD} = 5 V		2.3	2.7	3.1	2.3	2.7	3.1	V
		CDL			2.8	3.3	3.9	2.8	3.3	3.9	
	Analog output dc offset	TXA				$V_{DD}/2$			$V_{DD}/2$		V
	Digital input current	TXD, TRS, TXR1, TXR2	V_I = 0 to V_{DD}				±1			±1	μA
	Analog input current	RXA					±15			±15	μA
	Bias input current	RXB, CDL	V_I = 3 V				±150			±150	μA
I_{DD}	Supply current		V_{DD} = 4 V			3	6		3	5	mA
			V_{DD} = 5 V			5	10		5	8	
			V_{DD} = 5.5 V			8	16		8	12	
C_i	Input capacitance, all inputs		f = 1 MHz			10			10		pF
C_o	Output capacitance, all inputs		f = 1 MHz			10			10		pF
	Phase jitter						200			200	μs
	Bias distortion‡						±15%			±15%	
	Carrier detect threshold, off-on§				−45.5		−43	−45.5		−43	dBm
	Carrier detect threshold, on-off§				−48		−45.5	−48		−45.5	dBm
	Carrier detect hysteresis				2.5	2.8		2.5	2.8		dBm

†All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡Bias distortion is the departure from a 50% duty cycle when a series of alternating mark and space tones is received.
§This is the threshold with the CDL input properly adjusted.

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS	TCM3105JE			TCM3105JL			UNIT
			MIN	TYP†	MAX	MIN	TYP†	MAX	
$t_{d(off\text{-}on)}$	Carrier detect off-to-on delay time	RX = 600 or 1200 b/s	12		25	12		25	ms
		RX = 5, 75, or 150 b/s	48		80	48		80	
$t_{d(on\text{-}off)}$	Carrier detect on-to-off delay time	RX = 600 or 1200 b/s	12		20	12		20	ms
		RX = 5, 75, or 150 b/s	48		75	48		75	
	Transmit frequency deviation from assignment (see Table 1)	f_{clock} = 4.4336 MHz		±1			±1		Hz

†All typical values are at V_{CC} = 5 V, T_A = 25°C.

PRINCIPLES OF OPERATION

The TCM3105 FSK modem is made up of four functional circuits. The circuits are the transmitter, the receiver, a carrier detector, and control and timing (See Figure 1).

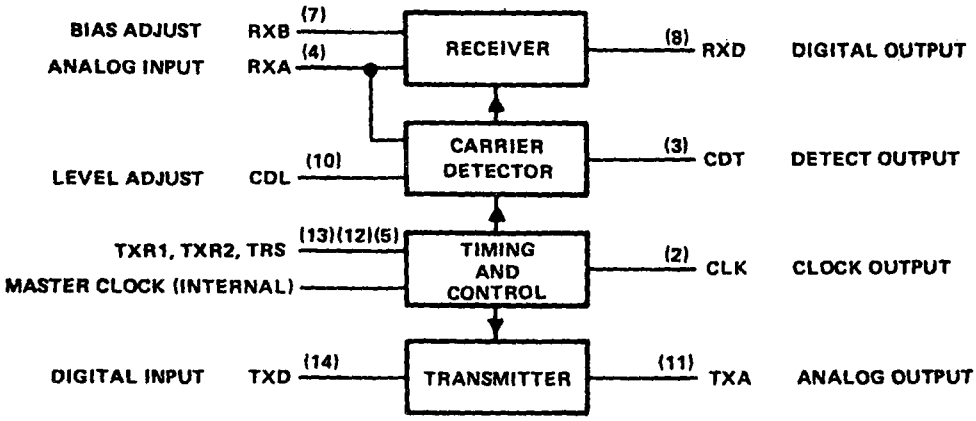


FIGURE 1. TCM3105 SYSTEM PARTITIONING

transmitter

The transmitter comprises a phase coherent FSK modulator, a transmit filter, and a transmit amplifier. The modulator is a programmable frequency synthesizer that drives the output frequencies by variable division of the oscillator frequency (4.4336 MHz). The division ratio is set by the states of the Transmit/Receive Standard Input (TRS), the Bit Rate Select inputs (TXR1 and TXR2), and the Digital Data input (TXD).

A switched-capacitor low-pass filter limits the harmonics and noise outside the transmit band and the characteristics of this filter are set by the frequency select inputs as previously described. The harmonics introduced by the transmit filter clock are removed by a continuous low-pass filter.

The transmitter output level varies with power supply voltage and so must be compensated in the 2-wire to 4-wire converter to give a constant output level to the line.

receiver

A continuous low-pass anti-aliasing filter is followed by the receive amplifier, which automatically controls the gain to give a constant output level from the receive filter. The receive filter limits the bandwidth of the signal presented to the demodulator, reducing out-of-band interference, and has very high rejection of the transmit channel frequencies. These are typically present at much higher levels than the received signal.

The group delay equalizer is a switched-capacitor network that compensates the delay introduced by the receive filter and the network. The output from the equalizer is then limited to give an FSK modulated squarewave that is presented to the demodulator.

The demodulator is an edge-triggered multivibrator that triggers off positive and negative going edges. The output of the demodulator is, therefore, a stream of constant-length pulses at a frequency that is double the frequency of the limited input signal. The dc component of this signal is proportional to the received frequency and is extracted by a switched-capacitor, low-pass, post-demodulator filter.

The variation of dc level with received frequency is presented to a comparator that slices at a level externally fixed by the RXB bias adjustment pin. This voltage depends on received bit rate and internal offsets. The comparator output is then the received data at the RXD output.

arrier detect

The carrier detect circuits comprise an energy detector and digital delay. The energy detector compares the total signal level at the output of the receive filter to an externally set threshold level on the CDL input. The comparator has a 2.5-dB hysteresis and a delay to allow for momentary signal loss and to prevent oscillation. The output of the detector is available on the CDT pin where a high level indicates that a carrier is present. The data output is clamped to a MARK condition when the carrier detect output switches off at the end of transmission.

ontrol and timing

An on-chip oscillator runs from an external 4.4336-MHz crystal connected between the OSC1 and OSC2 pins or an external signal driving OSC1. A clock signal equal to 16 times the highest selected bit rate (transmit or receive) is available on the CLK output.

The single-supply rail means that all analog functions are referenced to an internally generated reference. All analog inputs and output must be ac coupled.

ansmit and receive modes

The various modes of operation of the TCM3105 are given in Table 1. The data convention is that a logic high is a mark and a logic low is a space.

TABLE 1. MODES OF OPERATION

STANDARD	TRS	TXR1	TXR2	TRANSMITTED BAUD RATE	RECEIVED BAUD RATE	TRANSMIT FREQUENCY ASSIGNMENTS (Hz)	RECEIVE FREQUENCY ASSIGNMENTS (Hz)	CLK FREQUENCY (kHz)
CCITT V.23	L	L	L	1200	1200	M 1300 S 2100	M 1300 S 2100	19.11
	H	L	L	1200	75	M 1300 S 2100	M 390 S 450	19.11
	L	L	H	600	75	M 1300 S 1700	M 390 S 450	9.56
	H	L	H	600	600	M 1300 S 1700	M 1300 S 1700	9.56
	L	H	L	75	1200	M 390 S 450	M 1300 S 2100	19.11
	H	H	L	75	600	M 390 S 450	M 1300 S 1700	9.56
	L	H	H	75	75	M 390 S 450	M 390 S 450	1.19
BELL 202	$\overline{CLK}$	L	L	1200	1200	M 1200 S 2200	M 1200 S 2200	19.11
	$\overline{CLK}/8$	L	H	1200	150	M 1200 S 2200	M 387 S 487	19.11
	$\overline{CLK}/8$	L	H	1200	5	M 1200 S 2200	M 387 S 0	19.11
	CLK	H	L	150	1200	M 387 S 487	M 1200 S 2200	19.11
	CLK	H	H	150	150	M 387 S 487	M 387 S 487	2.39
	CLK†	H†	L†	5	1200	M 387	M 1200	19.11
	H†	H†	H†			S 0	S 2200	
	H	H	H	Transmit Disabled	1200	Transmit Disabled	M 1200 S 2200	19.11

H = high level, L = low level
†In these modes, the modulation is controlled by the TRS and TXR2 pins. TXD is tied high.

APPLICATION INFORMATION

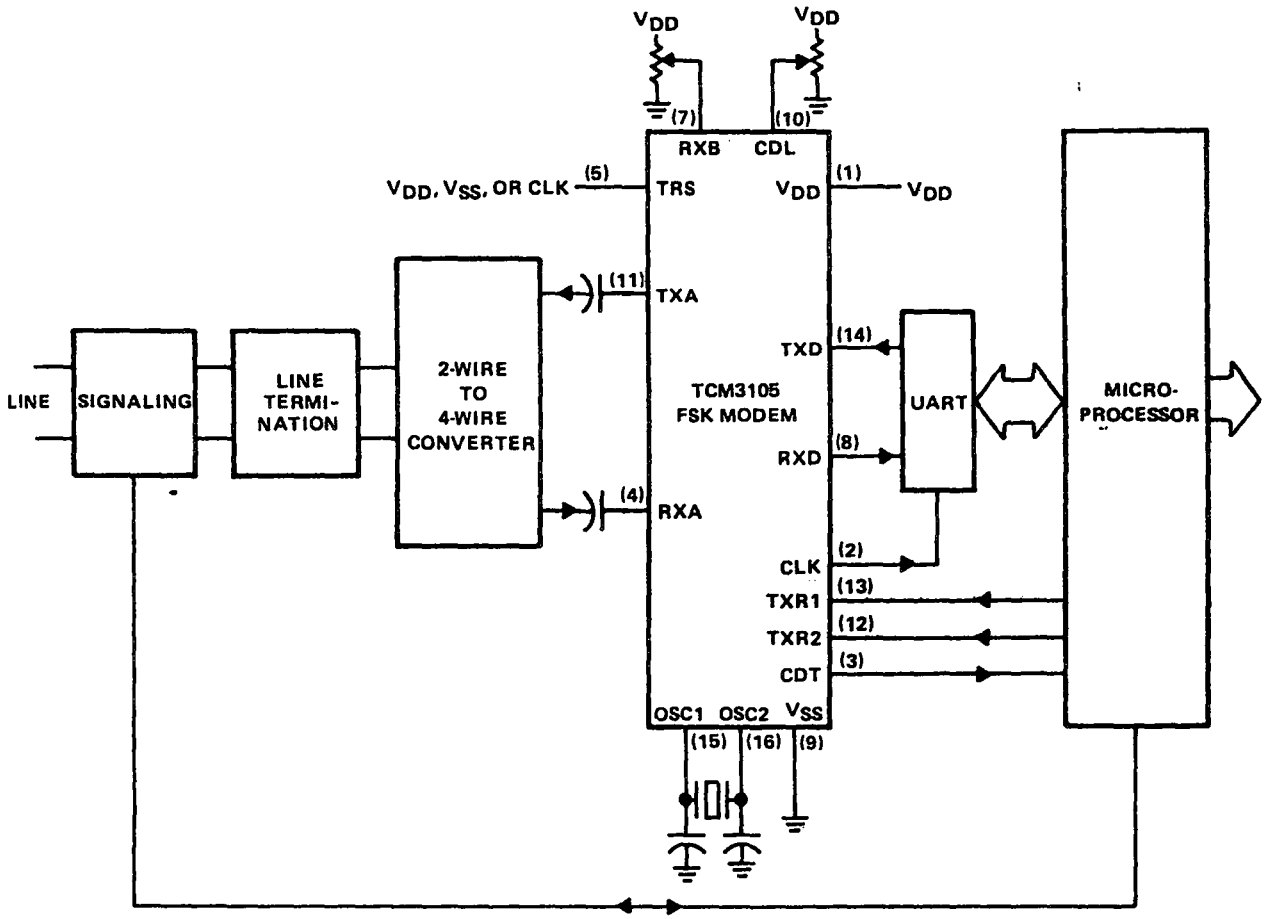


FIGURE 2. TYPICAL SYSTEM CONFIGURATION

APPLICATION INFORMATION

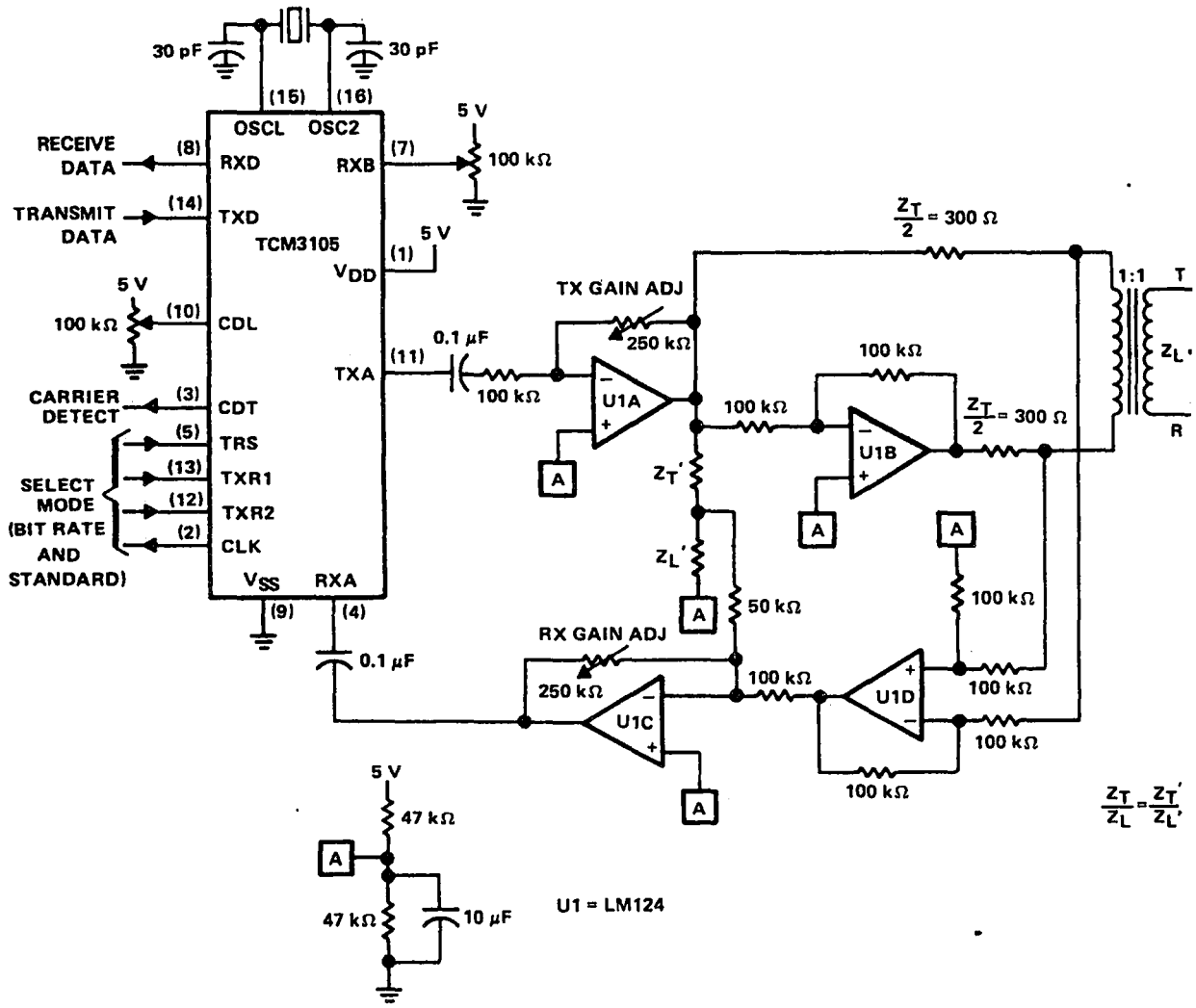


FIGURE 3. TELEPHONE LINE INTERFACE CIRCUIT

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

APPLICATION INFORMATION

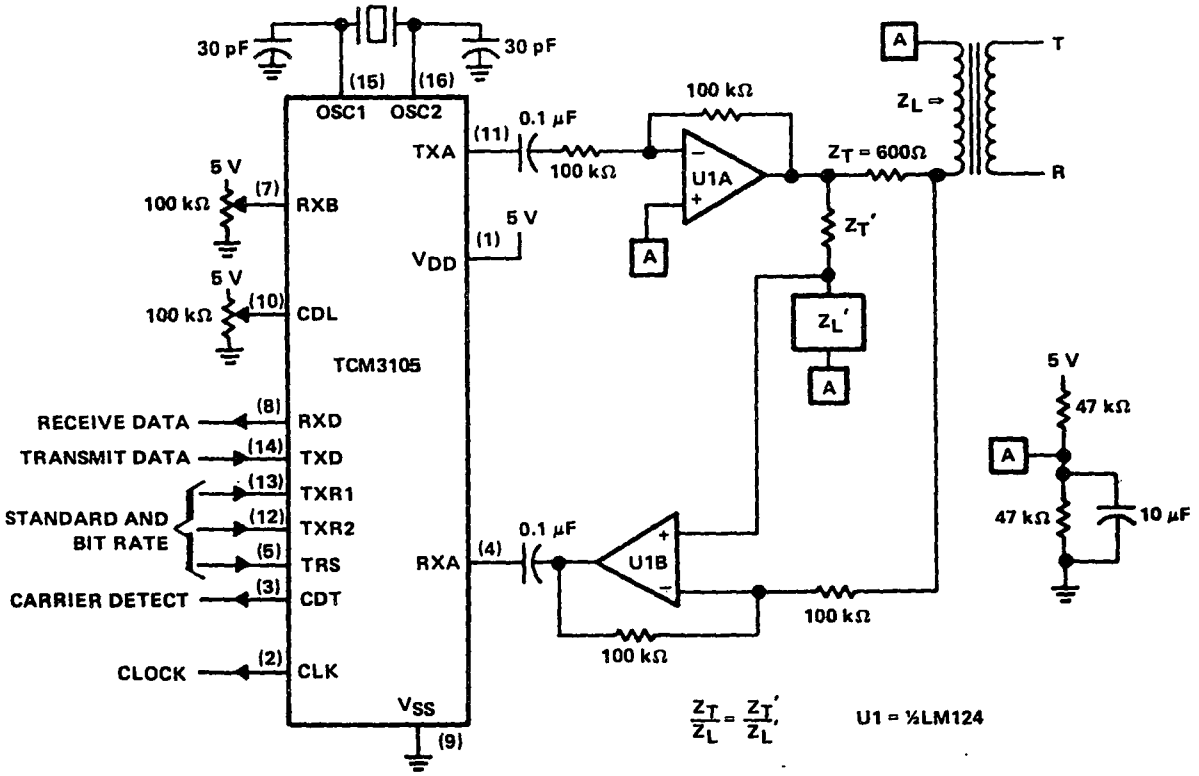


FIGURE 4. SIMPLIFIED TELEPHONE LINE INTERFACE CIRCUIT