โมเด็มไร้สาย
(Wireless Modem)

ผู้จัดทำ

นายนำพล วิมลพิทยารัตน์ 33100169
นายประพันธ์ ฉันทวุฒิเศรษฐี 33100203
นายสรพจน์ เมฆสกุลวงศ์ 33100404

อาจารย์ที่ปรึกษา

อ.เกรียงไกร วงศ์โรจนภรณ์

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์บัณฑิต
สาขาวิศวกรรมโทรคมนาคม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ประจำปีการศึกษา 2536

ปริญญานิพนธ์
ประจำปีการศึกษา ๒๕๓๖
เรื่อง โมเด็มไร้สาย
(Wireless Modem)

ภาควิชา วิศวกรรมโทรคมนาคม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ผู้จัดทำ
นายนำพล วิมลพิทยารัตน์ 33100169
นายประพันธ์ ฉันทวุฒิเศรษฐี 33100203
นายสรพจน์ เมฆสกุลวงศ์ 33100404

_____อาจารย์ที่ปรึกษา
(อ.เกรียงไกร วงศ์โรจนภรณ์)

โมเด็มไร้สาย

(Wireless Modem)

โดย

นายนำพล วิมลพิทยารัตน์

นายประพันธ์ ฉันทวุฒิเศรษฐี

นายสรพจน์ เมฆสกุลวงศ์

อ.เกรียงไกร วงศ์โรจนภรณ์ อาจารย์ที่ปรึกษา

ปีการศึกษา 2536

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ได้นำเสนอการออกแบบ และการทำงานของโมเด็มไร้สายซึ่งประกอบด้วยส่วนสำคัญสองส่วนได้แก่ ส่วนฮาร์ดแวร์ และส่วนซอฟท์แวร์ โดยส่วนฮาร์ดแวร์นั้นประกอบด้วยส่วนอินเตอร์เฟสกับเครื่องคอมพิวเตอร์ ส่วนแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกแบบ FSK ซึ่งใช้ไอซีเบอร์ XR-2206 กับส่วนแปลงสัญญาณอนาล็อกเป็นดิจิตอลโดยใช้ไอซีเบอร์ XR-2211 และส่วนที่ทำการส่ง-รับสัญญาณความถี่วิทยุย่าน 144 MHz สำหรับส่วนซอฟท์แวร์นั้นสามารถทำการรับส่งข้อมูลได้ทั้งแบบ Chat Mode และแบบรับส่งไฟล์ โดยในการรับส่งไฟล์นั้นสามารถทำการบีบอัดข้อมูลก่อนทำให้การส่งไฟล์นั้นเป็นไปได้รวดเร็วยิ่งขึ้น

ABSTRACT

This project presents about designing and principle of wireless modem composed of two important parts ; hardware and software. Hardware compose of an interface which is used to connect between computer and wireless modem , digital to FSK analog signal converter used IC XR-2206 , FSK analog signal to digital converter used IC XR-2211 and 144 MHz radio frequency transmitter and receiver. For software part, it can transmit data in both Chat mode and File mode. Data compression function is an advantage which make the communication more efficience.

สารบัญ

หน้า

บทที่ 1 บทนำ

ปัจจุบันความก้าวหน้าทางเทคโนโลยีการสื่อสารและการประมวลผลคอมพิวเตอร์ได้ก้าวหน้าไปอย่างมาก และก้าวหน้าไปพร้อม ๆ กันกับแนวความคิดในการเชื่อมโยงคอมพิวเตอร์แต่ละหน่วยให้เป็นระบบโครงข่ายขนาดใหญ่ที่ให้ประสิทธิภาพสูงในการทำงานเป็นแนวความคิดใหม่ที่ท้าทาย ผลประโยชน์ที่ได้รับจากการสร้างระบบเครือข่ายคือข้อมูลที่กว้างขวางและทันสมัย โดยระบบฐานข้อมูลที่ช่วยให้ค่าใช้จ่ายโดยรวมในการเก็บข้อมูลถูกลง รวมทั้งก่อให้เกิดแอพพลิเคชั่นมากมาย

ส่วนสำคัญของระบบเครือข่ายคือการสื่อสารในระบบที่จะต้องก้าวไปให้ทันกับความสามารถในการรับส่ง และประมวลผลข้อมูลของคอมพิวเตอร์ การสื่อสารข้อมูลในปัจจุบันมีทั้งแบบที่ใช้ความเร็วสูงมาก ๆ ที่ใช้สำหรับโครงข่ายขนาดเล็ก ระยะทางใกล้ ๆ ภายในองค์การ เช่น ระบบแลน (Local Area Network) หรือระบบที่ใช้ความเร็วในการส่งไม่มากนัก โดยการใช้โครงข่ายสาธารณะที่มีอยู่แล้วเป็นตัวกลาง เช่น โครงข่ายโทรศัพท์ซึ่งก่อให้เกิดความประหยัดในการสร้างโครงข่าย แต่เนื่องจากจุดอ่อนของโทรศัพท์ที่ตอบสนองความถี่ของสัญญาณได้ดีในช่วงความถี่ของเสียง ทำให้โครงข่ายโทรศัพท์ไม่สามารถตอบสนองความสามารถด้านความเร็วของการส่งข้อมูลของคอมพิวเตอร์ได้อย่างเพียงพอ ดังนั้นการหันมาใช้ตัวกลางชนิดอื่น ๆ ที่ใช้ในการสื่อสารจึงได้ถูกนำมาใช้ในการพิจารณา

เนื่องจากข้อจำกัดดังกล่าว จึงได้ขอเสนอคลื่นวิทยุซึ่งเป็นตัวกลางอีกชนิดหนึ่งที่สามารถนำมาชดเชยขีดจำกัดของโทรศัพท์ได้แก่ แบนด์วิดธ์ของอากาศที่สามารถตอบสนองความถี่ของสัญญาณได้อย่างไม่จำกัด ทำให้สามารถเพิ่มความเร็วในการส่งได้ รวมทั้งความสะดวกในการเคลื่อนย้ายคอมพิวเตอร์ แต่เนื่องจากการแพร่กระจายคลื่นวิทยุในอากาศยังเป็นสิ่งที่มีการควบคุมอย่างเข้มงวด ดังนั้นการใช้งานจึงถูกจำกัด และทำให้ขนาดของโครงข่ายไม่สามารถทำให้กว้างขวางได้เท่ากับโครงข่ายโทรศัพท์

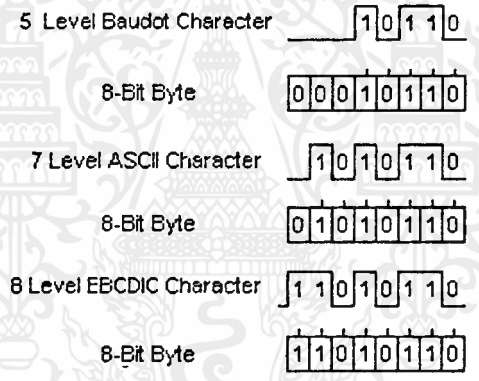
บทที่ 2 ทฤษฎีและหลักการเบื้องต้น

2.1 ประเภทของการสื่อสารข้อมูล

ข้อมูลในระบบการสื่อสารเขียนแสดงได้ด้วยค่าในระบบเลขฐาน 2 โดยใช้ค่าตัวเลข 0 หรือ 1 มาประกอบกันเป็นรหัส แต่ในการส่งเราอาศัยการส่งทางไฟฟ้า ข้อมูลจะถูกเปลี่ยนให้อยู่ในรูปแบบทางไฟฟ้าโดยใช้ค่าสัญญาณไฟฟ้า 2 ระดับคือ สูง และต่ำ

การวัดอัตราเร็วในการส่งได้จากจำนวนบิทที่ส่งไปในหน่วยเวลา โดยทั่วไปใช้หน่วย bit per second (bps) หากบิทหลาย ๆ บิทถูกนำมาใช้เป็นกลุ่ม เช่นในคอมพิวเตอร์ กลุ่มของข้อมูลนี้เรียกว่าไบท์(byte) หนึ่งไบท์ประกอบด้วยหลาย ๆ บิทที่จำเป็นสำหรับแทนอักขระ 1 ตัวปัจจุบันขนาดของไบท์ที่นิยมคือ ไบท์ขนาด 8 บิท

แม้ว่าขนาดของไบท์จะมีขนาดคงที่ แต่ไบท์ก็สามารถใช้แทนอักขระที่มีจำนวนบิทต่ออักขระน้อยกว่า 8 บิทได้ดังรูปที่ 2.1 แสดงตัวอย่างการใช้ไบท์ขนาด 8 บิทแทนรหัส BAUDOT ขนาด 5 บิทต่ออักขระ รหัสแอสกีย์ ขนาด 7 บิทต่ออักขระ และรหัส EBCDIC ขนาด 8 บิทต่ออักขระ



รูปที่ 2.1 แสดงตัวอย่างการใช้ข้อมูล 8 บิทแทนรหัส BAUDOT

ข่าวสารที่เราส่งในระบบสื่อสารข้อมูลนั้นอาจมีขนาดใดก็ได้แต่ในการส่งนั้นจะต้องมีการกำหนดจุดเริ่มต้น และจุดสิ้นสุดของการส่งแต่ละชุด ตามปกติจะจัดแบ่งข่าวสารที่จะส่งเป็นบล็อค ๆ หนึ่งบล็อคคือกลุ่มของบิทจำนวนหนึ่งที่เราส่งออกไปเป็นหน่วยเดียวกัน โดยมีการนำกลุ่มของบิทนั้นไปผ่านกระบวนการบางอย่าง เพื่อใช้ในการควบคุมข้อผิดพลาดที่อาจจะเกิดขึ้น

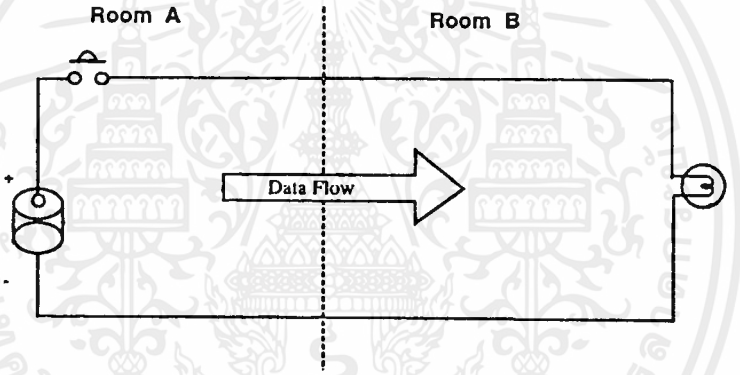
เราสามารถจำแนกวิธีการส่งข้อมูลได้หลายแบบตามคุณสมบัติต่าง ๆ ดังนี้

1.การจำแนกตามทิศทางในการส่งข้อมูล สามารถแบ่งการส่งข้อมูลออกได้เป็น 3 ชนิด ดังนี้

- การรับหรือส่งทางเดียว (Simplex)
- การรับส่งแบบผลัดกันส่ง (Half Duplex)
- การรับส่งสวนทางได้พร้อมกัน (Full Duplex)

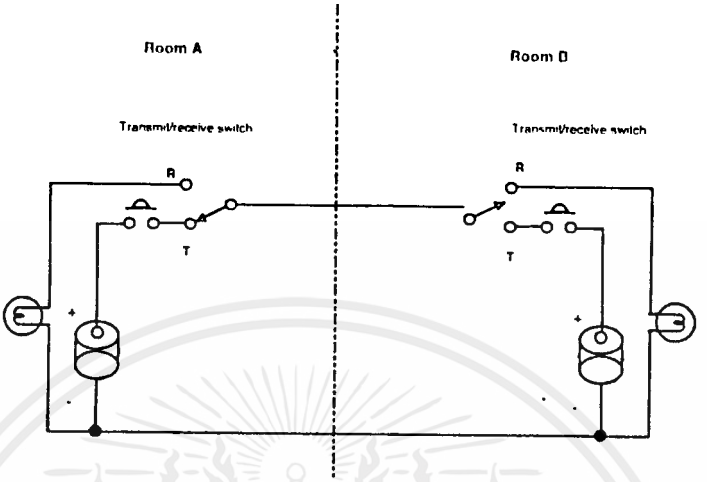
ทั้งสามวิธีมีข้อดีและข้อเสียในตัวของมันเอง ความจริงแล้วเราได้ใช้การรับส่งทั้งสามวิธีนี้ในชีวิตประจำวันอยู่ตลอดเวลาไม่ว่าจะเป็นการชมโทรทัศน์ การฟังเพลง การสนทนา หรือในการทำงานต่าง ๆ

การติดต่อสื่อสารแบบที่รับหรือส่งทางเดียวนั้นเราเรียกมันว่าเป็นการสื่อสารแบบSimplex ตัวอย่างง่าย ๆ ที่เห็นได้ชัดก็คือ การรับส่งโทรทัศน์ และวิทยุกระจายเสียงนั่นเอง สถานีโทรทัศน์จะเป็นตัวส่งและเครื่องรับทำหน้าที่รับแต่เพียงอย่างเดียวจะส่งข่าวหรือภาพกลับมายังสถานีส่งไม่ได้ การสื่อสารแบบ Simplex นี้เรามักจะไม่ค่อยนำมันมาใช้ในการสื่อสารข้อมูลเนื่องจากว่าเราจำเป็นต้องโต้ตอบกันระหว่างการรับส่งข้อมูล หรือบางทีก็เปลี่ยนจากผู้รับเป็นผู้ส่งซึ่งทำไม่ได้สำหรับการติดต่อกันในแบบ Simplex นี้ การสื่อสารแบบ Simplex นอกจากจะใช้สำหรับส่งโทรทัศน์ และวิทยุกระจายเสียงแล้ว เครื่องโทรพิมพ์ตามสำนักงานบางแห่งอาจใช้การติดต่อแบบนี้เช่นกันในการรับข่าวสารจากที่อื่น ๆ เพียงอย่างเดียว



รูปที่ 2.2 แสดงการรับส่งข้อมูลแบบทางเดียว (Simplex)

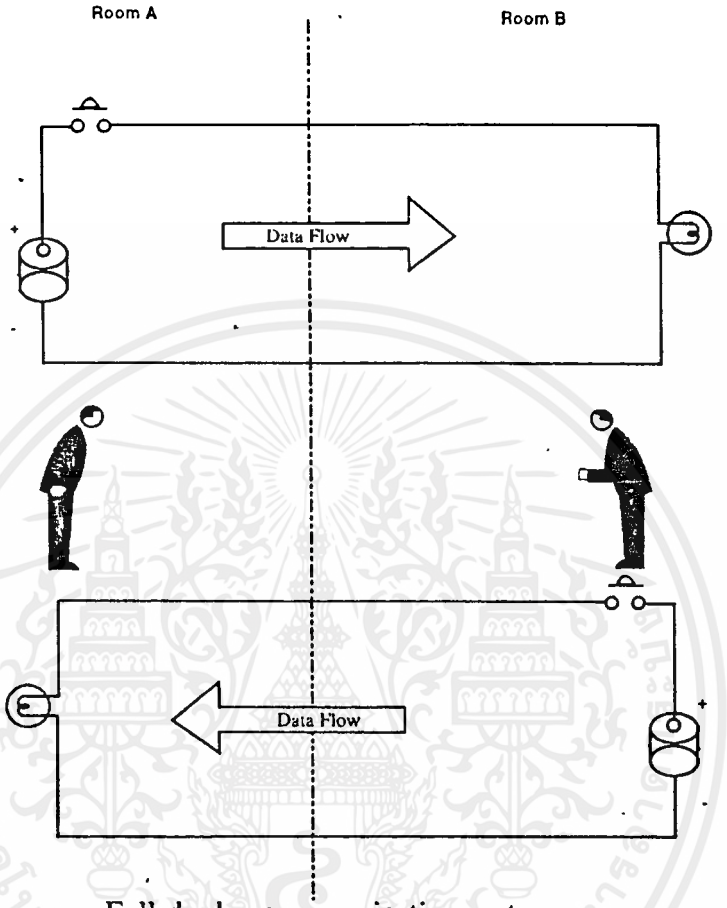
ส่วนการรับส่งแบบที่สองนี้ เราเรียกว่า การรับส่งแบบ Half Duplex มีคุณสมบัติสามารถรับและส่งข้อมูลได้ แต่ว่าต้องสลับกันส่ง จะส่งพร้อมกันทั้งสองด้านไม่ได้ อุปกรณ์ที่ใช้การติดต่อในแบบ Half Duplex ได้แก่ วิทยุมือถือ (Walkie - talkie) และ INTERCOM เป็นต้น เมื่อฝ่ายใดฝ่ายหนึ่งส่งอีกฝ่ายก็จะทำหน้าที่รับ จนกระทั่งฝ่ายแรกส่งจบฝ่ายหลังจึงจะกลับเป็นผู้ส่งได้ และฝ่ายส่งในตอนแรกก็จะเป็นผู้รับ สลับกันเช่นนี้เรื่อยไป ทั้งสองฝ่ายจะเป็นผู้ส่งพร้อมกันไม่ได้ เพราะสัญญาณจะชนกันทำให้ฟังไม่รู้เรื่อง การรับส่งในแบบ Half Duplex นับว่าซับซ้อนกว่าในแบบ Simplex ขึ้นมาเล็กน้อยเพราะทั้งสองด้านสามารถทำหน้าที่รับและส่งได้ตามลำดับแต่ห้ามส่งพร้อมกันเป็นอันขาด



Half-duplex communication system

รูปที่ 2.3 แสดงการรับส่งข้อมูลสวนทางกันได้แบบผลัดกันส่ง (Half Duplex)

แบบที่ซับซ้อนที่สุด ก็คือการรับส่งในแบบสวนทางได้พร้อมกัน ซึ่งเรียกว่า Full Duplex การรับส่งแบบนี้ผู้รับและส่งสามารถรับและส่งพร้อม ๆ กันในเวลาเดียวกันได้ไม่จำเป็นต้องรอให้อีกฝ่ายหนึ่งส่งจบเสียก่อนอย่างใน Half Duplex ตัวอย่างเช่น การพูดโทรศัพท์ ถึงแม้ปกติเมื่อผู้หนึ่งพูดอีกฝ่ายจะคอยฟัง แล้วตอบกลับมาเมื่อฝ่ายแรกพูดจบซึ่งเป็นลักษณะของการติดต่อแบบ Half Duplex ก็ตามแต่ เราอาจจะพูดพร้อม ๆ กัน หรือพูดสวนกลับไปได้ทันทีโดยยังคงฟังอยู่เหมือนเดิม ลักษณะเช่นนี้เราเรียกว่าติดต่อกันในแบบ Full Duplex การสื่อสารข้อมูลระหว่างคอมพิวเตอร์สองเครื่องมีใช้ทั้งแบบ Half Duplex และ Full Duplex ขึ้นอยู่กับลักษณะของการเชื่อมต่อและงานของมัน

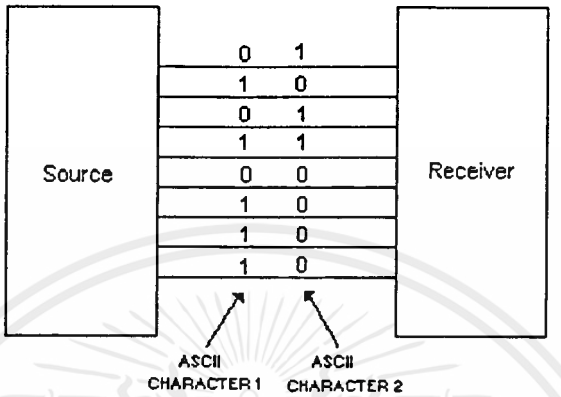


รูปที่ 2.4 แสดงการรับส่งข้อมูลแบบสวนทางกันได้พร้อมกัน (Full Duplex)

2.การจำแนกตามลักษณะการจัดข้อมูลสามารถแบ่งการส่งข้อมูลออกได้เป็น 2 ชนิดดังนี้

- การส่งข้อมูลแบบขนาน (Paralel Transmission)เป็นการส่งข้อมูลทีละไบท์ คือส่งทุกบิทของรหัสที่ประกอบขึ้นเป็นอักขระไปพร้อม ๆ กันในเวลาเดียวกัน การสื่อสารแบบนี้มีข้อดี คือความสามารถในการส่งข้อมูลที่สูงขึ้น แต่มีจุดอ่อนที่สำคัญ คือจำนวนช่องทางการสื่อสารที่จะต้องมีจำนวนเท่ากับจำนวนบิทที่ประกอบขึ้นเป็นอักขระ ซึ่งต้องใช้การมัลติเพล็กซ์ข้อมูลต่าง ๆ เช่น space division multiplex ซึ่งเป็นการแบ่งแชนแนล โดยใช้สายส่งหลาย ๆ เส้น หรือแบบ frequency division multiplex ที่ใช้การแบ่งย่านความถี่ในการส่งแต่ละแชนแนล

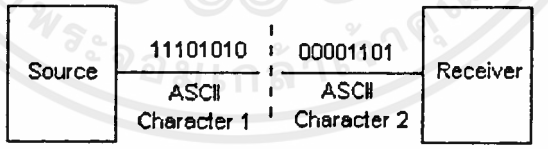
การสื่อสารแบบนี้มักใช้กับการสื่อสารที่มีระยะในการติดต่อไม่ไกลนัก โดยเฉพาะการส่งข้อมูลระหว่างหน่วยประมูลผลของคอมพิวเตอร์กับอุปกรณ์ประกอบซึ่งต้องการความเร็วอย่างมากในการติดต่อ



รูปที่ 2.5 แสดงการส่งข้อมูลแบบขนาน

- การส่งข้อมูลแบบอนุกรม (Series Transmission) การส่งข้อมูลแบบนี้จะกระทำทีละบิทด้วยแชนแนลเพียงแชนแนลเดียว ทางด้านรับจะต้องมีอุปกรณ์สำหรับจัดข้อมูลดังกล่าวเป็นชุดของอักขระ ตามข้อตกลงระหว่างอุปกรณ์ปลายทั้งสองที่สื่อสารกัน

การสื่อสารข้อมูลในปัจจุบัน โดยทั่วไปจะใช้การสื่อสารแบบอนุกรม เนื่องจากงานหลักของการสื่อสารก็คือการสื่อสารระยะไกล ซึ่งต้องการความประหยัดในการวางสาย หรือใช้ช่องความถี่อย่างมีประสิทธิภาพมากกว่าความเร็วในการสื่อสารข้อมูล



รูปที่ 2.6 แสดงการส่งข้อมูลแบบอนุกรม

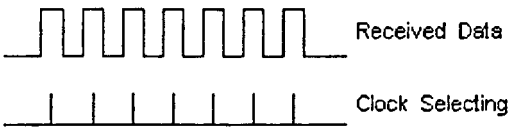
3.การจำแนกตามความสัมพันธ์ของข้อมูล ในการสื่อสารข้อมูลนั้นจะต้องพิจารณาถึงความสัมพันธ์ระหว่างข้อมูล 2 ชนิด คือ

-ความสัมพันธ์ระหว่างบิท (Bit Synchronization)

-ความสัมพันธ์ของอักขระ (Character Synchronization)

ความสัมพันธ์ของบิท หมายถึงว่า ทางด้านรับจะต้องได้รับบิทต่าง ๆ ที่ทางด้านส่งส่งมาได้อย่างถูกต้อง การรับข้อมูลในทางการสื่อสารข้อมูล จะใช้การเลือกตรวจสอบสัญญาณทางไฟฟ้าในสายส่งในช่วงเวลาสั้น ๆ ซึ่งอัตราเร็วในการเลือกแซมปลิ้ง (SAMPLING) ข้อมูลจะต้องเท่ากับอัตราเร็ว

ในการส่งข้อมูลด้วยการสื่อสารนั้น ๆ จึงจะประสบความสำเร็จ และการเลือกก็ต้องเลือกในช่วงกลางของสัญญาณ เพื่อให้เกิดความผิดพลาดเนื่องจากความผิดพลาดของสัญญาณน้อยที่สุดดังรูปที่ 2.7



รูปที่ 2.7 แสดงความสัมพันธ์ระหว่างบิท

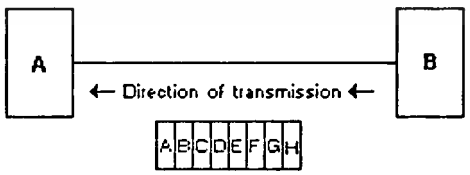
สำหรับสัญญาณที่ใช้ในการกำหนดจังหวะในการเลือกตรวจสอบสถานะของสายส่ง เรียกว่า สัญญาณนาฬิกา (CLOCK SIGNAL)

ความสัมพันธ์ของอักขระเป็นการกำหนดถึงรูปแบบของการรับส่งข้อมูลในระดับอักขระ นั่นคือวิธีในการกำหนดขอบเขตของอักขระและขอบเขตของข่าวสารว่าข้อมูลบิทสตรีมที่รับมาได้ส่วนไหนเป็นข่าวสารจริงส่วนไหนเป็นสัญญาณรบกวนที่เข้ามาในระบบ และในข้อมูลข่าวสารนั้น แต่ละอักขระที่เกิดจากบิทหลาย ๆ บิทรวมกัน เริ่มต้นที่ไหนและสิ้นสุดที่จุดไหนซึ่งถึงแม้เราสามารถรับสัญญาณทางไฟฟ้าจากสายส่งมาได้อย่างถูกต้องด้วยความสัมพันธ์ของบิท แต่ถ้าไม่ได้ใช้ความสัมพันธ์ของอักขระที่ถูกต้อง เราก็จะไม่สามารถได้ข้อมูลข่าวสารที่ด้านส่งต้องการส่งซึ่งถือว่าเป็นการสื่อสารที่ไม่ประสบความสำเร็จ

ในการสื่อสารข้อมูล เราจะจำแนกวิธีการส่งข้อมูลตามการกำหนดความสัมพันธ์ของข้อมูลได้ 2 แบบ คือ

1.การส่งแบบสัมพันธ์(Synchronous Transmission)

การส่งแบบนี้ใช้สำหรับการส่งข้อมูลทั้งชุดไปในครั้งเดียวดังรูป 2.8 เทคนิคการส่งแบบสัมพันธ์นี้ เวลาระหว่างบิทแต่ละบิทจะมีค่าเท่ากัน การส่งมีลักษณะคล้ายกับการส่งข่าวสารในรูปของเลขฐานสองที่มีจำนวนติดต่อกันไป โดยไม่ได้แยกว่าความยาวใดเป็นของช่วงอักษรใดในระบบเช่นนี้ บิทแต่ละบิทจะมีความยาวเท่ากัน ตัวอักษรแต่ละตัวมีช่วงเวลาห่างกันเท่ากับศูนย์ ทางด้านรับนั้นเพียงหาว่าบิทแรกของตัวอักษรตัวแรกคือบิทใด และทราบขนาดหรือจำนวนบิทในหนึ่งตัวอักษรพร้อมทั้งความเร็วในการส่งก็จะสามารถแยกข่าวสารของแต่ละอักษรออกมาได้



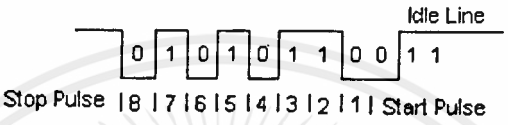
รูปที่ 2.8 แสดงการส่งข้อมูลแบบสัมพันธ์

2.การส่งแบบไม่สัมพันธ์(Asynchronous Transmission)

การส่งแบบนี้ตัวอักขระจะถูกส่งออกไปที่เวลาใดก็ได้โดยไม่จำเป็นต้องมีความสัมพันธ์ระหว่างตัวอักษรว่าต้องมีเวลาที่แน่นอนอย่างไร

การส่งแบบนี้ตัวอักขระจะถูกส่งออกไปที่เวลาใดก็ได้โดยไม่จำเป็นต้องมีความสัมพันธ์ระหว่างตัวอักษรว่าต้องมีเวลาที่แน่นอนอย่างไร

ทางด้านรับจะต้องทราบถึงบิทเริ่มต้นของรหัสแต่ละตัวว่าเริ่มต้นเมื่อใด ซึ่งสามารถกระทำได้โดยการเพิ่มบิทที่เรียกว่า พัลซ์เริ่มต้น (Start Pulse) โดยเติมเข้าไปที่ข้างหน้าชุดของบิทของทุกอักขระ และเมื่อการส่งส่งครบทุกบิทของตัวอักษรแล้วจะต้องมีบิทสำหรับบอกถึงการสิ้นสุดที่เรียกว่าพัลซ์การสิ้นสุด(Stop Pulse)ส่งมาเพื่อให้ทางด้านรับมีเวลาสำหรับการเตรียมข้อมูลของตัวอักษรตัวต่อไปดังรูป 2.9 บางครั้งจึงเรียกระบบการส่งแบบนี้ว่า ระบบการส่งแบบเริ่ม-หยุด(Start-Stop Transmision)



รูปที่ 2.9 แสดงการส่งข้อมูลแบบไม่สัมพันธ์

สำหรับการส่งแบบไม่สัมพันธ์นี้ เนื่องจากทุกอักขระที่ส่งออกมาต่างเป็นอิสระต่อกัน และทุกๆอักขระจะมีบิทเริ่มต้นและบิทสิ้นสุดอย่างน้อยอักขระละ 1 บิท ฉะนั้นหากเกิดสัญญาณรบกวนทำให้เกิดความผิดพลาดในข้อมูล ความผิดพลาดที่เกิดขึ้นนี้จะเกิดกับข้อมูลของอักขระเพียงตัวเดียวเพราะอักขระแต่ละตัวมีความสัมพันธ์กับบิทเริ่มต้นและบิทสิ้นสุดของตัวมันเองเท่านั้น ความผิดพลาดเช่นนี้ถ้าเกิดกับข่าวสารในการส่งแบบสัมพันธ์อาจทำลายข่าวสารทั้งบล็อกได้เพราะความสัมพันธ์จะเสียไปทั้งบล็อก

2.2 โมเด็มและมาตรฐานของโมเด็ม

การรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ในระยะทางไม่ไกลมากนัก เราอาจใช้การรับส่งแบบอนุกรม (RS-232C) ซึ่งส่งข้อมูลดิจิตอลของคอมพิวเตอร์ไปตามสายจนถึงผู้รับได้กรณีนี้เราสามารถรับส่งข้อมูลได้ไกลถึง 35 เมตรตามคุณสมบัติของ RS-232C หรือถ้าสายเคเบิลที่ใช้มีคุณภาพดีอาจส่งได้ไกลถึง 150 เมตรที่ความเร็ว 9,600 บิตต่อวินาที แต่สำหรับระยะทางที่ไกลมาก ๆ เช่น หลายสิบกิโลเมตร หลายร้อยกิโลเมตรจนถึงหลายพันกิโลเมตร การส่งข้อมูลแบบดิจิตอลออกไปโดยตรงจะไม่เหมาะสมหลายอย่าง ปัญหาที่สำคัญก็คือ คลื่นรูปสี่เหลี่ยมของสัญญาณดิจิตอล เมื่อส่งไปไกล ๆ จะเพี้ยนหรือมีรูปร่างผิดไปจากเดิมได้ง่ายทำให้สายส่ง และวงจรรับส่งสัญญาณดิจิตอลต้องถูกออกแบบมาเป็นอย่างดี ราคาของสายส่งสัญญาณแบบดิจิตอลจึงมีราคาแพงกว่าสายส่งสัญญาณแบบอนาล็อกมาก ในทางปฏิบัติเราอาจรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์สองเครื่องโดยใช้สัญญาณดิจิตอลผ่านสายได้ ซึ่งทั้งสายส่งและวงจรเชื่อมต่อทั้งหมดเป็นแบบดิจิตอล แต่ว่าค่าใช้จ่ายจะมีราคาแพงมากจนกระทั่งไม่ค่อยคุ้มที่จะทำเช่นนี้วิธีหลีกเลี่ยงก็คือหาทางส่งข้อมูลไปตามสายส่งในแบบอนาล็อกแทน การทำเช่นนี้เราจำเป็นต้องใช้อุปกรณ์เข้าช่วยแปลงสัญญาณในการรับส่งข้อมูลทั้งสองด้านซึ่งเป็นที่มาของโมเด็มนั่นเอง

สัญญาณเมื่อถึงปลายทาง โมเด็มก็จะแปลงสัญญาณอนาล็อกที่ได้รับกลับมาเป็นสัญญาณดิจิตอลแล้วส่งให้คอมพิวเตอร์ต่อไปในรูปของสัญญาณดิจิตอลผ่านทาง RS-232C เช่นกัน กระบวนการแปลงสัญญาณกลับนี้เรียกว่า Demodulation

สัญญาณอนาล็อกมีคุณสมบัติเหมาะที่จะส่งไปไกล ๆ มากกว่าสัญญาณแบบดิจิตอลเพราะว่าสัญญาณอนาล็อกจะเพี้ยนหรือมีรูปร่างผิดจากเดิมยากกว่า และสูญเสียกำลังในการส่งน้อยกว่าทำให้ส่งได้ระยะทางไกลมากขึ้น นอกจากนี้เรายังสามารถกรองสัญญาณรบกวนบางส่วนที่ไม่ต้องการ (Filter) ออกได้อีกด้วย ราคาของสายส่งและอุปกรณ์เชื่อมต่อก็มีราคาถูก เราจึงจำเป็นต้องใช้โมเด็มในการรับส่งข้อมูลคอมพิวเตอร์ระยะทางไกลผ่านสายส่งอนาล็อก

จากการที่โมเด็มแปลงสัญญาณจากคอมพิวเตอร์ให้กลายเป็นสัญญาณอนาล็อกในการรับส่งข้อมูลนี้เอง ถ้าโมเด็มแปลงสัญญาณออกมาอยู่ในรูปของเสียงซึ่งเป็นสัญญาณอนาล็อกแบบหนึ่งเราก็สามารถรับส่งข้อมูลข้อมูลผ่านทางสายโทรศัพท์ได้ โมเด็มทั่ว ๆ ไปที่เราใช้งานจะเป็นโมเด็มที่แปลงสัญญาณจากคอมพิวเตอร์ให้อยู่ในรูปคลื่นเสียงทั้งหมด มีโมเด็มบางชนิดที่แปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกความถี่สูง แต่โมเด็มแบบนี้มีใช้งานน้อยและส่งข้อมูลโดยใช้สายส่งพิเศษจะส่งผ่านสายโทรศัพท์ธรรมดาไม่ได้ เราจึงเน้นเฉพาะโมเด็มที่ทำงานในย่านความถี่เสียงเท่านั้น ไม่ว่าโมเด็มจะเป็นแบบไหนก็ตาม เมื่อได้รับข้อมูลดิจิตอลจากคอมพิวเตอร์มันจะเปลี่ยนให้กลายเป็นสัญญาณอนาล็อก จากนั้นก็นำสัญญาณอนาล็อกที่ได้นี้มารวมเข้ากับสัญญาณพาหะ (Carrier Wave)แล้วส่งออกไปทางสายส่งข้อมูล สัญญาณพาหะหรือคลื่นพาหะนี้จะทำหน้าที่พาข้อมูลที่อยู่ในรูปสัญญาณอนาล็อกไปจนถึงปลายทาง

รูปที่ 2.10 โมเด็มช่วยให้การรับส่งข้อมูลผ่านสายโทรศัพท์ได้โดยเปลี่ยนสัญญาณให้เป็นเสียง

องค์กรต่างๆได้สร้างมาตรฐานการอินเตอร์เฟสของตนเองขึ้นมาใช้ซึ่งมีอยู่มากมายได้แก่

1. EIA : THE ELECTRONIC INDUSTRIES ASSOCIATION เป็นมาตรฐานกำหนดโดยสมาคมโรงงานอุตสาหกรรมผู้ผลิตอิเลคทรอนิคส์แห่งสหรัฐอเมริกา มาตรฐานนี้ตั้งขึ้นมาใช้กำหนดมาตรฐานของเครื่องมือ อุปกรณ์อิเลคทรอนิคส์ต่าง ๆ การกำหนดมาตรฐานใช้รหัส RS เป็นหลัก เช่น มาตรฐาน RS-232C ใช้กันแพร่หลายในระบบสื่อสารข้อมูลคอมพิวเตอร์โดยจะกล่าวถึงมาตรฐานของสัญญาณไฟฟ้าในการอินเตอร์เฟสเทอร์มินอลเข้ากับโมเด็ม หรืออินเตอร์เฟสเทอร์มินอลเข้ากับไมโครคอมพิวเตอร์ หรือ อินเตอร์เฟสเครื่องพิมพ์เข้ากับคอมพิวเตอร์ นอกจากนี้ยังมีมาตรฐานอื่น ๆ ที่มีการตั้งขึ้นมาใช้ประกอบด้วย EIA RS449 RS422A RS423A

2. CCITT : THE CONSULTATIVE COMMITTEE IN INTERNATIONAL TELEGRAPH AND TELEPHONE เป็นองค์กรสากลกำหนดมาตรฐานของระบบสื่อสารระหว่างประเทศทั้งโทรเลขและโทรศัพท์ สำหรับมาตรฐานของ CCITT ที่ใช้แพร่หลายในการสื่อสารข้อมูล เช่น V.28 (ใช้แทน RS232C ได้) V.10 (ใช้แทน EIA RS423A ได้) V.11 (ใช้แทน EIA RS422A ได้) X.10 (ใช้แทน EIA RS449 ได้) เป็นต้น

3. ISO : THE INTERNATIONAL STANDARD ORGANIZATION เป็นองค์กรกำหนดมาตรฐานทางกายภาพของอุปกรณ์ต่าง ๆ ที่ใช้ในการสื่อสารโทรคมนาคมโดยเฉพาะที่เกี่ยวกับคอมพิวเตอร์และอินฟอร์เมชั่นโปรเซสซิ่ง องค์กรนี้จะประสานงานกับ CCITT อย่างใกล้ชิดดังนั้นมาตรฐานของ ISO จะสามารถใช้แทนมาตรฐานประเภทเดียวกันของ CCITT และ EIA ได้ เช่น ISO/2110 สามารถใช้แทน RS232C และ RS-366A ได้ นอกจากนี้ ISO4902 ยังใช้แทน EIA RS-499 ได้ เป็นต้น

4. BELL SYSTEM เป็นมาตรฐานที่กำหนดโดยองค์กรทางโทรศัพท์ของบริษัท Bell Laboratory มาตรฐานของ Bell ถูกกำหนดขึ้นเพื่อใช้ควบคุมโรงงานผู้ผลิตสินค้าที่ต้องการใช้งานร่วมกับระบบของ Bell แต่ระยะหลัง Bell ก็เริ่มมีการแก้ข้อกำหนดของตนบางส่วนให้เข้ากับ CCITT ได้

มาตรฐานของโมเด็มแบบต่าง ๆ ที่ใช้กันมากนั้นมักจะใช้ตามมาตรฐาน CCITT V-Series ตั้งแต่ความเร็วต่ำไปจนถึงความเร็วสูงได้แก่

- V.21 เป็นมาตรฐานของโมเด็มความเร็ว 300 บิทต่อวินาที ใช้เทคนิคการผสมสัญญาณแบบ FSK (Frequency Shift Keying) รับส่งข้อมูลได้ในแบบ Full Duplex เป็นโมเด็มที่ใช้กับสายโทรศัพท์ ปัจจุบันนี้มีใช้กันน้อยเนื่องจากความเร็วในการรับส่งข้อมูลต่ำ

- V.22 รับส่งข้อมูลด้วยความเร็ว 1,200 บิทต่อวินาที หรือ ลดความเร็วลงมาที่ 600 บิทต่อวินาทีได้ การผสมสัญญาณใช้เทคนิคของ PSK (Phase Shift Keying) รับส่งข้อมูลในแบบ Full Duplex ใช้กับสายโทรศัพท์หรือสายตรงก็ได้ขึ้นอยู่กับโมเด็มว่าถูกออกแบบมาให้ต่อใช้กับสายตรงหรือไม่ ซึ่งจัดเป็นโมเด็มความเร็วปานกลางที่ได้รับความนิยมอยู่ในปัจจุบัน

- V.22 bis รับส่งข้อมูลด้วยความเร็ว 2,400 บิทต่อวินาที หรือลดความเร็วลงมาที่ 1,200 บิทต่อวินาทีได้ การผสมสัญญาณใช้เทคนิคของโมเด็มความเร็วสูง คือ QAM รับส่งข้อมูลแบบ Full Duplex ใช้กับสายโทรศัพท์หรือสายตรงได้ V.22 bis เป็นมาตรฐานของโมเด็มความเร็วปานกลางที่จะเข้ามาแทนที่ V.22

- V.23 เป็นมาตรฐานที่คล้ายกับมาตรฐาน V.22 แต่รับส่งข้อมูลในแบบ Half Duplex คือมีความเร็ว 1,200 บิทต่อวินาที หรือลดความเร็วลงมาที่ 600 บิทต่อวินาที ใช้เทคนิคการผสมสัญญาณแบบ FSK สามารถต่อใช้กับสายโทรศัพท์ได้

- V.26 เป็นมาตรฐานของโมเด็มสายตรงแบบใช้สาย 4 เส้น (4 Wires) รับส่งข้อมูลในแบบ Full Duplex ใช้เทคนิคการผสมสัญญาณชนิด PSK มีความเร็วในการรับส่งข้อมูล 2,400 บิทต่อวินาทีจะนำมาต่อใช้กับสายโทรศัพท์ไม่ได้

- V.26 bis เป็นมาตรฐานเหมือนกับ V.26 แต่สำหรับใช้กับสายโทรศัพท์แทน มีความเร็วในการรับส่งข้อมูลที่ 2,400 บิทต่อวินาที หรือลดความเร็วลงมาที่ 1,200 บิทต่อวินาทีได้ การรับส่งข้อมูลเป็นแบบ Half Duplex ใช้เทคนิคการผสมสัญญาณแบบ PSK

- V.27 เป็นมาตรฐานสำหรับโมเด็มความเร็ว 4,800 บิทต่อวินาทีที่ใช้กับสายตรงเท่านั้น เทคนิคของการผสมสัญญาณเป็นแบบ PSK รับส่งข้อมูลในแบบ Full Duplex ได้ มีความเร็ว 4,800 บิทต่อวินาทีซึ่งถือว่าเป็นความเร็วการรับส่งข้อมูลสูงที่สุดของเทคนิคการผสมสัญญาณแบบ PSK

- V.27 bis คล้ายกับมาตรฐานแบบ V.27 แต่ว่ารับส่งข้อมูลที่ 4,800 บิทต่อวินาที หรือลดความเร็วลงมาที่ 2,400 บิทต่อวินาทีได้ ใช้สำหรับสายตรงแบบ 4 Wires เท่านั้น การผสมสัญญาณก็เป็นแบบ PSK สามารถรับส่งข้อมูลได้ทั้งในแบบ Full Duplex และ Half Duplex

- V.27 ter เป็นมาตรฐานโมเด็มความเร็ว 4,800 บิทต่อวินาที หรือลดความเร็วลงมาที่ 2,400 บิทต่อวินาทีได้สำหรับใช้กับสายโทรศัพท์ การรับส่งข้อมูลเป็นแบบ Half Duplx เท่านั้น ใช้เทคนิคการผสมสัญญาณชนิด PSK

- V.29 จัดเป็นมาตรฐานของโมเด็มความเร็วสูงใช้กับสายตรงแบบ 4 Wires เท่านั้นการรับส่งข้อมูลใช้ได้ทั้ง Full Duplex และ Half Duplex สามารถรับส่งข้อมูลได้ตั้งแต่ 9,600 บิทต่อวินาที หรือลดความเร็วลงมาที่ 7,200 บิทต่อวินาที และ 4,800 บิทต่อวินาที ที่ความเร็ว 9,600 บิทต่อวินาที จะใช้เทคนิคการผสมสัญญาณแบบ QAM

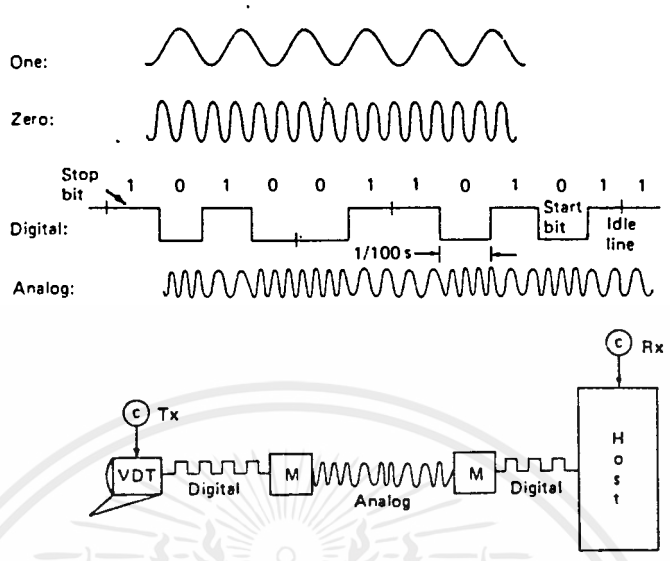
- V.32 เป็นมาตรฐานโมเด็มความเร็วสูงสำหรับใช้กับสายโทรศัพท์ สามารถรับส่งข้อมูลได้ที่ความเร็ว 9,600 บิทต่อวินาที ในแบบ Full Duplex หรือ ลดความเร็วลงมาที่ 4,800 บิทต่อวินาทีได้ มาตรฐาน V.32 นี้ยังใช้งานกับสายตรงแบบ 2 Wires ได้อีกด้วย ใช้เทคนิคการผสมสัญญาณเป็นแบบ QAM ทั้งที่ความเร็ว 9,600 และ 4,800 บิทต่อวินาที

2.3 เทคนิคการมอดูเลทสัญญาณดิจิตอล

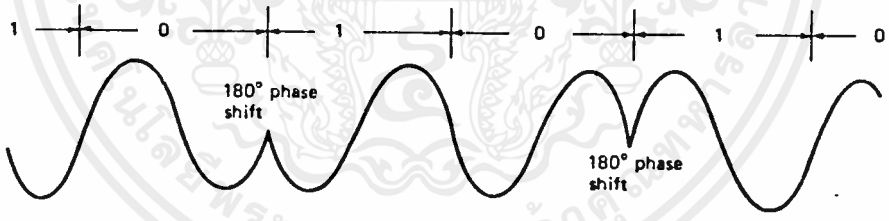
เทคนิคการมอดูเลทสัญญาณดิจิตอลที่ใช้กันมากในปัจจุบันคือ Frequency Shift Keying (FSK) ,Phase Shift Keying (PSK) และ Quadrature Amplitude Modulation (QAM) การมอดูเลทแบบ Frequency Shift Keying ใช้ในโมเด็มความเร็วต่ำ โดยแทน "0" และ "1" ด้วยความถี่ต่างกัน การผสมสัญญาณแบบ FSK มักใช้กับโมเด็มความเร็วประมาณ 300 ถึง 1200 บิทต่อวินาที การผสมสัญญาณแบบ FSK นี้ อัตราการส่งข้อมูลบิทเรทจะเท่ากับบอดเรทเสมอดังรูปที่ 2.11

สำหรับการผสมสัญญาณแบบ Phase Shift Keying (PSK)นั้น ใช้หลักการแทน "0"และ "1" ด้วยการเปลี่ยนแปลงมุมของช่วงสัญญาณในสายส่ง(Phase) เช่นเราอาจกำหนดว่า"0" แทนด้วยมุมต่อเนื่องกันไป และ 1"แทนด้วยมุมเปลี่ยนไปจากเดิม 180 องศา ถ้าเราส่งข้อมูล 0101 รูปร่างของคลื่นในสายส่งจะเป็นดังรูปที่ 2.12

การผสมสัญญาณแบบ PSK นี้อัตราการส่งข้อมูล(Bit Rate) จะมีค่าสูงกว่าอัตราการเปลี่ยนแปลงของสัญญาณ (Baud Rate)ได้ ความเร็วสูงสุดที่ใช้ PSK คือ 9600 บิทต่อวินาที ซึ่งมีการเปลี่ยนแปลงค่ามุมครั้งละ 22.5 องศา การเปลี่ยนแปลงค่ามุม 22.5 องศานี้ นับว่าน้อยมากทำให้ข้อมูลมักจะผิดพลาดอยู่เสมอ ส่วนมากจึงใช้งาน PSK ที่ความเร็ว 4800 บิทต่อวินาที



รูปที่ 2.11 แสดงการมอดูเลทสัญญาณแบบ Frequency Shift Keying



รูปที่ 2.12 แสดงการมอดูเลทสัญญาณแบบ Phase Shift Keying

เทคนิคการมอดูเลทแบบ Quadrature Amplitude Modulation (QAM) เป็นการผสมสัญญาณที่ใช้ทั้งการเปลี่ยนเฟสและขนาดของสัญญาณควบคู่กันไป สำหรับใช้กับโมเด็มความเร็วสูง ซึ่งถ้าใช้การเปลี่ยนเฟสเพียงอย่างเดียวมุมที่เปลี่ยนจะมีค่าน้อยเกินไป ทำให้เกิดข้อผิดพลาดได้ง่าย ถ้าใช้การเปลี่ยนเฟสและขนาดของสัญญาณค่าของมุมก็จะอยู่ห่างกันมากขึ้น ปกติที่มีใช้กันอยู่จะมีเฟสต่างกัน 8 เฟสและขนาดของสัญญาณต่างกัน 4 ระดับ ใช้แทนข้อมูล 16 สถานะ ซึ่งในหนึ่งลูกคลื่นจะสามารถส่งข้อมูลได้คราวละ 4 บิท ความเร็วในการรับส่งข้อมูลของ QAM อยู่ที่ 9600 บิทต่อวินาที โดยใช้ความถี่พาหะ 2400 Hz

2.4 การสื่อสารข้อมูลตามมาตรฐาน RS-232C

ตัวอักษร RS แทน "Recommended Standard", 232 แทนหมายเลขของมาตรฐานส่วนอักษร C แสดงให้รู้ว่า มาตรฐานได้รับการแก้ไขกี่ครั้ง

ตามมาตรฐาน RS-232-C ที่ถูกตีพิมพ์โดย EIA ได้กล่าวถึงการสื่อสารข้อมูลระหว่าง Data Terminal Equipment (DTE) and Data Communication Equipment (DCE) แต่ปัจจุบันตัวย่อ DCE จะแทน Data Circuit Terminating Equipment

อุปกรณ์ DTE : มีลักษณะดังต่อไปนี้

1. เป็นอุปกรณ์ประกอบด้วยตัวส่งข้อมูล (Data Source) หรือตัวรับข้อมูล (Data Link) หรือเป็นทั้งตัวส่งและรับข้อมูลก็ได้

2. เป็นอุปกรณ์ที่ประกอบด้วยฟังก์ชั่นยูนิทต่อไปที่ Control Logic, Buffer store และอุปกรณ์ Input และ Output จำนวนหนึ่งตัวหรือมากกว่าก็ได้ หรือรวมเครื่องคอมพิวเตอร์เข้าไปด้วยก็ได้

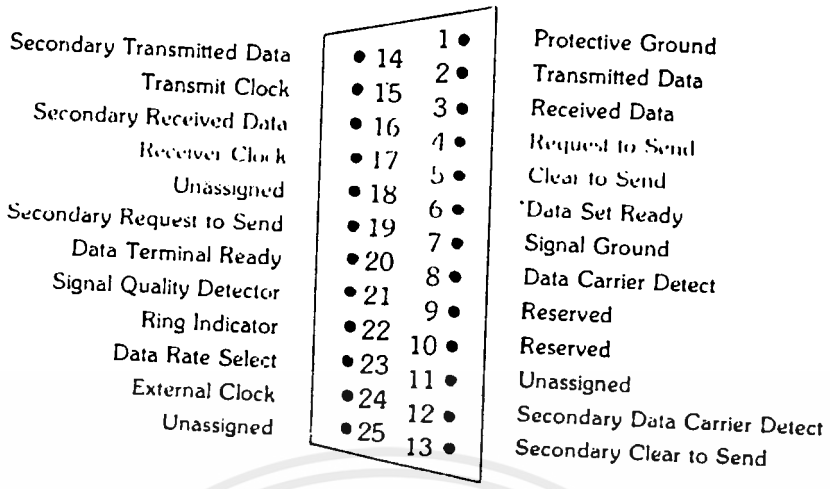
อุปกรณ์ DCE : อุปกรณ์ที่มีฟังก์ชั่นการทำงานต่าง ๆ ที่ทำให้เกิดการเชื่อมต่อทำให้การเชื่อมต่อดำรงต่อไปและยุติการเชื่อมต่อ นอกจากนี้ใช้เปลี่ยนลักษณะของสัญญาณ และสร้างรหัสสัญญาณที่จำเป็นต้องใช้ในการสื่อสารข้อมูลระหว่าง DTE และ DCE โดย DCE อาจเป็นส่วนใดส่วนหนึ่งของคอมพิวเตอร์หรือไม่ก็ได้

เราใช้มาตรฐาน RS-232C ในการสื่อสารข้อมูลแบบอนุกรมระหว่าง DCE กับ DTE โดยอัตราการส่งข้อมูลกำหนดให้อยู่ระห่าง 0-20,000 bit/sec ในการประยุกต์ใช้งาน RS-232C อัตราเร็วสูงสุดที่ใช้ควรจะมีค่าไม่เกิน 19.2 kbit/sec

มาตรฐานนี้กำหนดความยาวของสายเคเบิลที่ใช้ในการสื่อสารข้อมูลไว้ไม่เกิน 50 ฟุต แต่เคเบิลอาจยาวกว่า 50 ฟุตก็ได้ถ้าเรารู้สภาพแวดล้อมของสายเคเบิล และอยู่ในเงื่อนไขที่ถูกกำหนดไว้ในมาตรฐาน

ลักษณะของตัวข้อต่อ (Connector) ของ RS-232C ระหว่างเคเบิ้ลทั้งสองปลายจะใช้ข้อต่อแบบ 25 ขารูปร่างหน้าตัดคล้ายตัว "D" มีชื่อเรียกว่า DB-25 ดังรูปที่ 2.13

การใช้งานจะใช้เพียง 22 ขาไม่ได้ใช้ 3 ขา สัญญาณแต่ละขาทำทำหน้าที่แตกต่างกันออกไป แต่ปกติแล้วการรับส่งข้อมูลทั่วไปเราใช้สัญญาณเพียง 8 ถึง 9 เส้นเท่านั้นก็พอสัญญาณที่เหลือเราไม่นำมาใช้ เนื่องจากว่าบางเส้นเป็นสัญญาณรับส่งข้อมูล และสัญญาณควบคุมของช่องสัญญาณสำรอง (Secondary Channel) บางเส้นปล่อยว่างไว้ และบางเส้นใช้สำหรับงานพิเศษบางอย่างเท่านั้น จึงขอไม่กล่าวถึงในที่นี้



รูปที่ 2.13 แสดงลักษณะของข้อต่อแบบ DB-25

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

-ขาที่ 1 (Protective Ground) เป็นสายดินของอุปกรณ์
-ขาที่ 2 (Transmitted Data) ใช้สำหรับส่งข้อมูล
-ขาที่ 3 (Received Data) ใช้สำหรับรับข้อมูล
-ขาที่ 4 (Request to Send) เป็นสัญญาณขอทำการส่งข้อมูล
-ขาที่ 5 (Clear to Send) เป็นสัญญาณตอบรับว่าเริ่มส่งข้อมูลได้
-ขาที่ 6 (Data Set Ready) เป็นสัญญาณแสดงว่าตัวรับพร้อมที่จะรับข้อมูล
-ขาที่ 7 (Signal Ground) เป็นกราวน์(Ground)ของสัญญาณรับส่ง
-ขาที่ 8 (Data Carrier Detect) เป็นตัวบอกว่าตัวรับและตัวส่งต่อถึงกันแล้ว และพร้อมที่จะทำการรับส่งข้อมูล ในกรณีใช้ต่อกับโมเด็ม ขานี้จะเป็นตัวบอกว่า โมเด็มทั้งสองด้านต่อถึงกันได้แล้ว โดยมีสัญญาณพาหะ (Carrier)ส่งถึงกัน
-ขาที่ 20 (Data Terminal Ready) เป็นสัญญาณแสดงว่าตัวส่งพร้อมที่จะส่งข้อมูล
-ขาที่ 22 (Ring Indicator) เป็นขาแสดงแทนกริ่งโทรศัพท์ที่เรียกเข้ามาการเชื่อมต่อบางอย่างก็จะไม่ใช้ขาที่ 22 นี้ในการทำงาน

ส่วนขาอื่นๆ ที่เหลือนั้น ส่วนมากมีหน้าที่คล้ายกับ 8 ขาแรกที่กล่าวมา และบางเส้นใช้กับงานพิเศษเท่านั้น ขาที่เราใช้สำหรับการรับส่งข้อมูลของข้อต่อแบบ DB-25 จึงเหลือเพียงขา 2,3,4, 5,6,7,8,20 และ 22 ยกเว้นการต่อใช้งานบางอย่างถึงจะต่อครบทุกเส้น

บทที่ 3 การทำงานของโมเด็มไร้สาย

3.1 โครงสร้างการทำงานของโมเด็มไร้สาย

โมเด็มไร้สาย (WIRELESS MODEM) นี้เป็นการส่งข่าวสารข้อมูลระหว่างคอมพิวเตอร์ส่วนบุคคล โดยใช้ตัวกลาง (MEDIA) ที่เป็นอากาศ โดยการใช้คลื่นวิทยุ

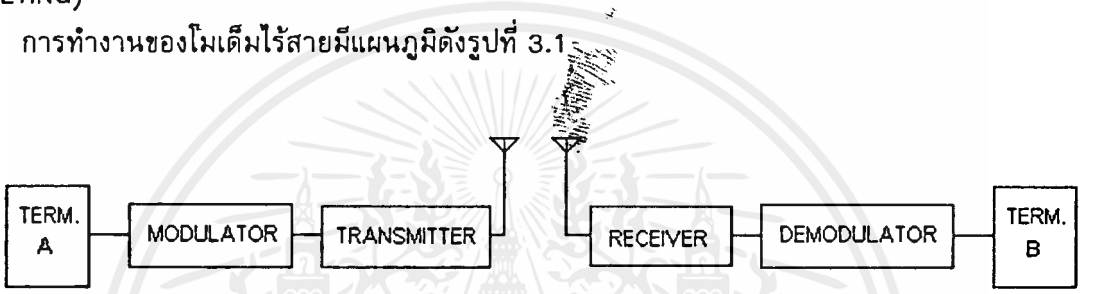
สำหรับลักษณะ(SPECIFICATION)ของโมเด็มไร้สายถูกกำหนดไว้ดังนี้

1.รับส่งข้อมูลจากพอร์ทอนุกรม(SERIAL PORT) ของคอมพิวเตอร์

2.การติดต่อระหว่างโมเด็มใช้ตัวกลางเป็นอากาศ โดยใช้คลื่นวิทยุย่านความถี่ย่าน 144 MHz

3.การมอดูเลทสัญญาณดิจิตอลจากคอมพิวเตอร์ใช้การมอดูเลทแบบ FSK (FREQUENCY SHIFT KEYING)

การทำงานของโมเด็มไร้สายมีแผนภูมิดังรูปที่ 3.1



รูปที่ 3.1 แสดงโครงสร้างการทำงานของโมเด็มไร้สาย

จากรูปเมื่อคอมพิวเตอร์ส่งข้อมูลดิจิตอลมาทางพอร์ทอนุกรม ข้อมูลดังกล่าวผ่านอินเตอร์เฟส RS-232 เนื่องจากข้อมูลของคอมพิวเตอร์มีลักษณะไบโพลาร์ ขนาด ±12 โวลท์ซึ่งแตกต่างจากการทำงานของไอซีของตัวมอดูเลเตอร์ที่ใช้ซึ่งจะรับส่งสัญญาณในช่วง 0 ถึง 5 โวลท์ดังนั้นจึงต้องเปลี่ยนขนาดของไฟฟ้าโดยใช้ไอซี MC-1489 จากนั้นสัญญาณดิจิตอลขนาด 0-5 โวลท์ ดังกล่าวจะผ่านไปยังอุปกรณ์เปลี่ยนสัญญาณดิจิตอลเป็นอนาลอก ซึ่งใช้เทคนิค FSK โดยใช้ไอซีเบอร์ XR-2206 สัญญาณที่ออกมาจะเป็นสัญญาณอนาลอก จากนั้นสัญญาณ FSK ย่านความถี่เสียงที่ได้นี้จะถูกนำไปผ่านวงจรมอดูเลททางความถี่แล้วส่งออกอากาศ

สำหรับทางด้านรับ เมื่อรับสัญญาณความถี่วิทยุดังกล่าวมาก็จะผ่านวงจรดีมอดูเลททางความถี่ ก็จะได้สัญญาณ FSK ย่านความถี่เสียงออกมา หลังจากนั้นก็นำมาผ่านวงจรดีมอดูเลเตอร์เพื่อเปลี่ยนสัญญาณอนาลอก FSK ให้เป็นสัญญาณดิจิตอลขนาด 0-5 โวลท์ โดยใช้ไอซีเบอร์ XR-2211 สัญญาณดิจิตอลดังกล่าวจะส่งผ่านไอซีเบอร์ MC1488 เพื่อเปลี่ยนสัญญาณยูนิโพลาร์ขนาด 0-5 โวลท์ ให้เป็นสัญญาณไบโพลาร์ขนาด ±12 โวลท์ เพื่อส่งผ่านอินเตอร์เฟส RS232 ทางพอร์ทอนุกรมของคอมพิวเตอร์ทางด้านรับต่อไป

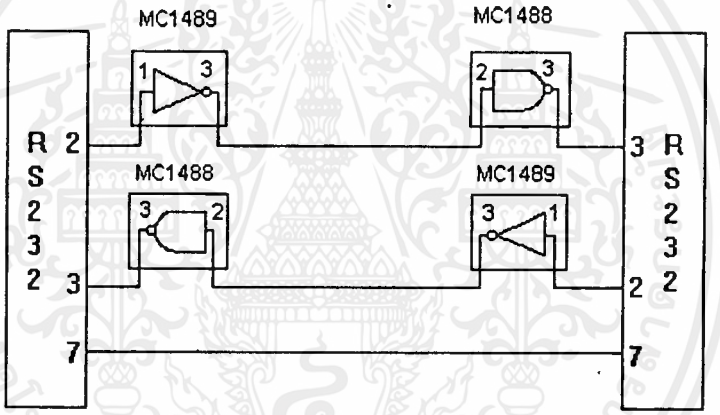
3.2 การทำงานของวงจรอินเตอร์เฟส

การส่งข้อมูลจากคอมพิวเตอร์มายังโมเด็มที่นิยมใช้กันนั้นเป็นการส่งข้อมูลออกมาที่พอร์ท RS-232C ซึ่งเป็นการส่งข้อมูลแบบอนุกรม โดยระดับแรงดันที่ส่งผ่านพอร์ท RS-232 นี้จะอยู่ในระดับ ±12 โวลท์ ดังนั้นตัวโมเด็มจึงจำเป็นต้องมีตัวแปลงระดับแรงดัน ±12 โวลท์ ให้อยู่ในระดับ TTL (0 - 5 โวลท์) ในที่นี้เราใช้ ไอซี MC1489 โดยระดับแรงดันที่เข้าตัวไอซีนี้

- หากเป็น 12 โวลท์ เมื่อผ่าน MC 1489 จะได้ระดับแรงดัน 0 โวลท์
- หากเป็น -12 โวลท์ เมื่อผ่าน MC 1489 จะได้ระดับแรงดัน 5 โวลท์

ในการส่งข้อมูลเข้าพอร์ท RS-232 ก็เช่นกันจำเป็นที่จะต้องเปลี่ยนแรงดัน ในระดับ TTL (0 - 5 โวลท์) ให้อยู่ในระดับแรงดัน ± 12 โวลท์ ในที่นี้เราใช้ ไอซี MC1488 โดยระดับแรงดันที่เข้าตัวไอซีนี้

- หากเป็น 0 โวลท์ เมื่อผ่าน MC 1489 จะได้ระดับแรงดัน 12 โวลท์
- หากเป็น 5 โวลท์ เมื่อผ่าน MC 1489 จะได้ระดับแรงดัน -12 โวลท์



รูปที่ 3.2 แสดงการอินเตอร์เฟสกับเครื่องคอมพิวเตอร์

3.3 การเปลี่ยนสัญญาณดิจิตอลเป็นสัญญาณอนาลอก FSK

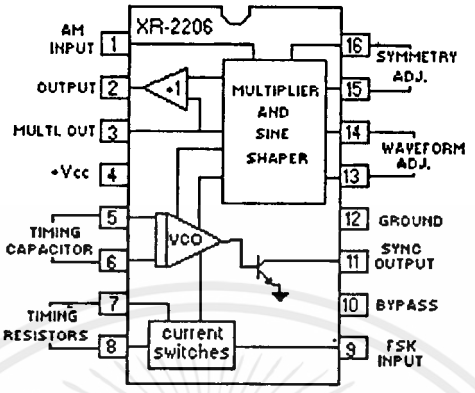
โมเด็มไร้สายที่ได้ทำการออกแบบนั้นใช้ไอซีเบอร์ XR-2206 มาทำหน้าที่เปลี่ยนสัญญาณดิจิตอลเป็นสัญญาณอนาลอก FSK

<u>โครงสร้างภายในของ XR-2206</u>

ไอซีเบอร์ XR-2206 นี้เป็นโมโนลิทิคฟังก์ชันเจนเนอเรเตอร์ มีความสามารถในการผลิตคลื่นรูปไซน์(sine) รูปคลื่นสามเหลี่ยม(triangle) สี่เหลี่ยม(square) แรมพ์(ramp) ได้โดยที่มีความถี่ตั้งแต่ไม่กี่เฮิร์ทจนถึงหลายร้อยกิโลเฮิร์ทโดยต่อกับวงจรภายนอกอีกเล็กน้อย นอกจากนี้ยังสามารถนำไอซี XR-2206 มาควบคุมขนาดและความถี่ (AM หรือ FM) และ Phase Shift or Frequency Shift Keying ได้ด้วย

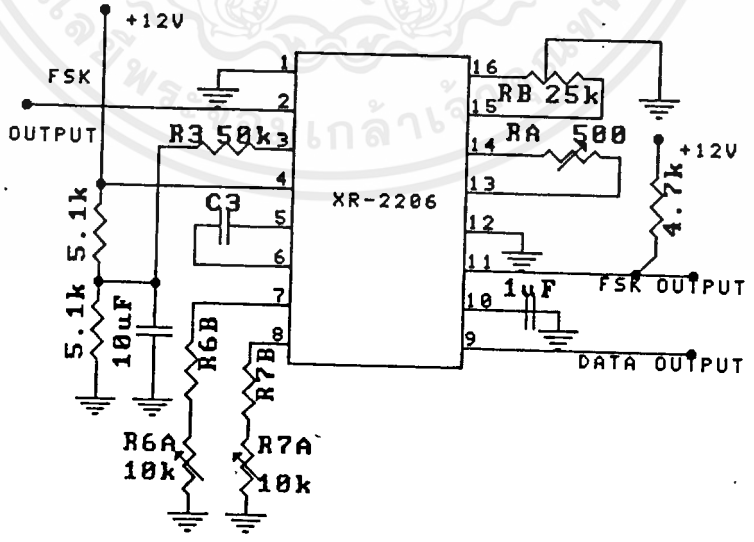
สำหรับ XR-2206 นี้อยู่ในแพคเกจไอซี 16 ขา สามารถที่จะใช้กับไฟเลี้ยง(power supply) ตัวเดียว คือในช่วง 10 ถึง 26 โวลท์ได้ หรืออาจจะใช้ไฟเลี้ยงคู่ในช่วง 5 ถึง 13 โวลท์ ขณะที่

ต้องผลิตสัญญาณคลื่นซายน์ (sine) นั้นค่า t.h.d.ของสัญญาณนั้นจะมีค่า 2.5% เมื่อยังไม่มีการปรับแต่งแต่ก็สามารถปรับให้เหลือเพียง 0.5% ได้โดยการควบคุมของวงจรที่นำมาต่อร่วมโดยที่สัญญาณเอาท์ พุทรูปไซน์นี้จะมีขนาดสูงสุด 2 โวลท์(r.m.s.) และมีเอาท์พุทอิมพิแดนซ์เท่ากับ 600 โอห์ม



รูปที่ 3.3 แสดงบล็อกโครงสร้างแต่ละส่วนของ XR-2206

รูปที่ 3.3 แสดงบล็อกไดอะแกรมแต่ละส่วนของ XR-2206 หัวใจสำคัญของส่วนนี้คือ V.C.O. (Voltage Control Oscillator) ซึ่งเมื่อนำมาใช้เป็นฟรีเควนซี่ ชิฟท์คีย์อิงโมดูเลเตอร์ที่ทำหน้าที่ในการเปลี่ยนสัญญาณดิจิตอลขนาด 0-5 โวลท์เป็นสัญญาณอนาลอกไซน์เวฟ 2 ความถี่ โดยแต่ละความถี่แทนข้อมูล 1 และ 0 จะได้วงจรดังรูปที่ 3.4



รูปที่ 3.4 แสดงการนำไอซี XR-2206 ไปใช้เป็นวงจรฟรีเควนซี่คีย์ชิฟท์อิงโมดูเลเตอร์

การออกแบบดังกล่าวสามารถอธิบายได้โดยดูรูป 3.3 และ 3.4 ทั้งสองประกอบดังนี้

คาปาซิเตอร์จัดเวลา (Timing Capacitor) ต่อระหว่างขา 5 และ 6 จะเป็นอินพุทของ VCO. มีค่าที่เหมาะสมในการใช้งานอยู่ระหว่าง 1,000pF - 100μF จะทำงานร่วมกับตัวต้านทานจัดเวลา(Timing Resister) เพื่อใช้กำหนดความถี่ของสัญญาณไซน์เวฟทั้งสอง โดยความถี่ที่ต้องการสามารถปรับแต่งได้ โดยใช้ความสัมพันธ์ตามสมการ

$$\text{ความถี่ (Hz)} = 1 / RC \quad \text{.....(1)}$$

เมื่อ C คือคาปาซิเตอร์จัดเวลา (Timing Capacitor) และ R คือตัวต้านทานจัดเวลา(Timing Resister)

เนื่องจากเราต้องการความถี่สองความถี่ สำหรับแทนสัญญาณดิจิตอลสูงและต่ำ และกำหนด ให้ใช้ค่าคาปาซิเตอร์เพียงค่าเดียว คือคาปาซิเตอร์จัดเวลา(Timing Capacitor) ที่ต่อระหว่างขา 5 และ 6 ดังนั้นค่าตัวต้านทานจัดเวลา(Timing Resister)จะต้องมี 2 ค่า ซึ่งใช้ตัวต้านทานจัดเวลา(Timing Resister) 2 กลุ่มต่อเข้าที่ขา 7 และ8 โดยกลุ่มที่มีค่าตัวต้านทานมากกว่าจะเป็นตัวสร้างความถี่ต่ำ และกลุ่มที่มีค่าตัวต้านทานน้อยกว่าจะเป็นตัวสร้างความถี่สูง แต่ละกลุ่มจะประกอบด้วยตัวต้านทานที่มีค่าคงที่ และตัวต้านทานที่ปรับค่าได้ที่ต่ออนุกรมกันเพื่อให้ปรับความถี่ได้ถูกต้อง และละเอียดยิ่งขึ้น

สำหรับช่วงของตัวต้านทานจัดเวลา (Timing Resister) ที่เหมาะสมในการใช้งานจะอยู่ในช่วงระหว่าง 4-200 กิโลโอห์ม เพื่อรักษาความคงค่าของอุณหภูมิ (temperature stability) และความเพี้ยนของสัญญาณซายน์ให้อยู่ในช่วงที่ใช้งานได้อย่างมีประสิทธิภาพ

เมื่อให้สัญญาณดิจิตอลอินพุทขนาด 0-5 โวลท์เข้ามาที่ขาที่ 9 ของไอซี

- ไอซีจะใช้ตัวต้านทานจัดเวลา(Timing Resister)ที่ขา 7 ในการให้กำเนิดสัญญาณรูปซายน์ ถ้าหากระดับสัญญาณที่เข้ามามีขนาดมากกว่า 2 โวลท์ หรือขา 9 มีการเปิดวงจร(open circuit)

- ไอซีจะใช้ตัวต้านทานจัดเวลา(Timing Resister)ที่ขา 8 ในการให้กำเนิดสัญญาณรูปซายน์ ถ้าหากระดับสัญญาณที่เข้ามามีขนาดน้อยกว่า 2 โวลท์

การเกิดคลื่นรูปซายน์นี้เริ่มมาจากส่วน VCO ของไอซีซึ่งส่วนนี้จะผลิตรูปคลื่นได้ 2 ชนิดคือรูปคลื่นแรมพ์ซึ่งจะป้อนไปยังส่วนของ Multiplier and Sine Shaper อีกทีหนึ่ง และคลื่นรูปสี่เหลี่ยม (rectangular)ซึ่งจะป้อนออกที่ขาเอาท์พุทที่ขา 11 โดยผ่านทรานซิสเตอร์ ซึ่งการผลิตคลื่นนี้ก็ขึ้นอยู่กับคาปาซิเตอร์จัดเวลา(Timing Capacitor) โดยคาปาซิเตอร์จัดเวลานี้จะเริ่มต้นเก็บประจุ เป็นผลทำให้เกิดช่วงพุ่งขึ้นของคลื่นรูปแรมพ์ และที่อีกเอาท์พุทจะได้สัญญาณ High ที่คลื่นรูปสี่เหลี่ยม (rectangular) จนกระทั่งแรงดันไฟฟ้านั้นจะถึงจุด ๆ หนึ่งเรียกว่า "firing voltage" ที่จุดนี้จะทำให้ได้สัญญาณคลื่นรูปสี่เหลี่ยมกลับกลายเป็นสัญญาณ"Low" และคาปาซิเตอร์จัดเวลา(Timing Capacitor) จะเก็บประจุในทิศทางตรงกันข้ามกับตอนต้นเป็นผลทำให้เกิดช่วงตกลงของคลื่นรูปแรมพ์ซึ่งก็จะตกลงจนถึงจุด"firing voltage"เช่นกันที่จุดนี้ จะทำให้คลื่นรูปสี่เหลี่ยม(rectangular)กลับกลายเป็นระดับสัญญาณ "High" และกระบวนการต่าง ๆก็จะกลับไปกลับมาเช่นนี้เหมือนเดิม

รูปคลื่นแรมพ์ที่ได้จากส่วนของ VCO ของไอซี XR-2206 นี้ จะถูกนำไปเข้ายังส่วนของ multiplier and sine shaper block อีกทีซึ่งส่วนนี้นั้นทำหน้าที่คล้ายกับวงจรขยายความต่างซึ่งจะทำให้เอาท์พุทอิมพิแดนซ์ ที่ขา 3 มีค่าสูง และที่ขา 2 จะเป็นเอาท์พุทบัฟเฟอร์ ที่มีค่าอิมพิแดนซ์ เท่ากับ

600 โอห์ม ในกรณีที่เปิดวงจรที่ขา 13 และ 14 นั้นจะมีผลทำให้เอาท์พุท ที่ขา 2 และ 3 นั้นจะให้คลื่นรูปแรมพ์ออกมา แต่ถ้าต่อตัวต้านทานที่มีค่าประมาณ 200 - 300 โอห์มที่ขา13 และ 14 จะทำให้ยอดของคลื่นรูปแรมพ์ถูกตัดออก ทำให้เอาท์พุทที่ขา 2 และ 3 นั้น ผลิตคลื่นซายน์ออกมา ถ้ามีการปรับแต่งที่ถูกต้องเหมาะสม จะทำให้คลื่นรูปซายน์ที่ได้มีความเพี้ยนเพียง 0.5% เท่านั้น การควบคุมอัตราการขยายสัญญาณเอาท์พุท FSK ทำได้โดยการควบคุมที่ขา 1 หรือขา 3 โดยในโมเด็มไร้สายนี้ใช้การควบคุมอัตราการขยายที่ขา 3 โดยจากรูปที่ 3.2 จะเห็นว่าที่ขา 3 จะต่อกับอินพุทของบัฟเฟอร์แอมปลิไฟเออร์ที่มีอัตราการขยายเท่ากับ 1 ซึ่งทำให้ขาที่ 2 มีเอาท์พุทอิมพิแดนซ์เท่ากับ 600 โอห์ม ดังนั้นเราสามารถใช้ขา 3 ทำหน้าที่เป็นตัวควบคุมแรงดัน สำหรับกำหนดอัตราการขยาย โดยใช้วงจรแบ่งแรงดัน (potential divider) ต่อเข้าที่ขา 3 ทำให้สัญญาณอินพุทที่เข้าไปยังบัฟเฟอร์แอมปลิไฟเออร์ ถูกควบคุมศักดาไฟฟ้าตามวงจรแบ่งแรงดันนั้น โดยปกติแรงดันดังกล่าวจะเป็นครึ่งหนึ่งของไฟเลี้ยงวงจร

<u>ขั้นตอนการคำนวณค่าของตัวต้านทานและค่าของคาปาซิเตอร์</u>

สำหรับโมเด็มไร้สายนี้ได้ใช้เทคนิคการมอดูเลทสัญญาณแบบ FSK ซึ่งได้กำหนดความถี่ต่าง ๆ ตัวอย่างเช่น ถ้าใช้ความถี่ตามมาตรฐานของ CCITT V.23 คือที่บอดเรท 1200 บิทต่อวินาที ความถี่ f_1 และ f_2 ค่า 1300 และ 2100 Hz ตามลำดับดังนั้นสามารถคำนวณค่าของความต้านทานและค่าคาปาซิเตอร์ได้ดังนี้

1.ทำการกำหนดค่าของคาปาซิเตอร์จัดเวลา (Timing Capacitor) ซึ่งควรจะอยู่ในช่วง 1 nF ถึง 100 µF ดังนั้นในโมเด็มไร้สายนี้ได้กำหนดค่าของคาปาซิเตอร์จัดเวลาไว้เท่ากับ 22 nF

2.ทำการคำนวณค่าของตัวต้านทานจัดเวลา (Timing Resistor)จากสมการที่ 1 และความถี่ที่กำหนด จะได้ดังนี้

- ค่าของตัวต้านทานจัดเวลาที่ขา 7 ซึ่งขานี้ใช้กับสัญญาณดิจิตอลที่มีลอจิกเป็น "1" และเนื่องจากความถี่ที่กำหนดลอจิก "1" นั้นมีค่าเท่ากับ 1300 Hz ดังนั้นจะได้

ค่าความต้านทานที่ขา 7 = $1 / (1300 \times 22 \times 10^{-9})$

= 34965.034965 โอห์ม

- ค่าของตัวต้านทานจัดเวลาที่ขา 8 ซึ่งขานี้ใช้กับสัญญาณดิจิตอลที่มีลอจิกเป็น "0" และเนื่องจากความถี่ที่กำหนดลอจิก "0" นั้นมีค่าเท่ากับ 2100 Hz ดังนั้นจะได้

ค่าความต้านทานที่ขา 8 = $1 / (2100 \times 22 \times 10^{-9})$

= 21645.021645 โอห์ม

3.4 การเปลี่ยนสัญญาณอนาลอก FSK ให้เป็นสัญญาณดิจิตอล

ภาครับสัญญาณ FSK นี้จะรับสัญญาณจากภาครับสัญญาณ RF โดยที่ภาคนี้จะทำการแปลงสัญญาณ FSK ที่มี 2 ความถี่ให้กลายเป็นสัญญาณดิจิตอล 0 (แทนด้วยแรงดันประมาณ 0 โวลท์) และ 1 (แทนด้วยแรงดันประมาณ 5 โวลท์)

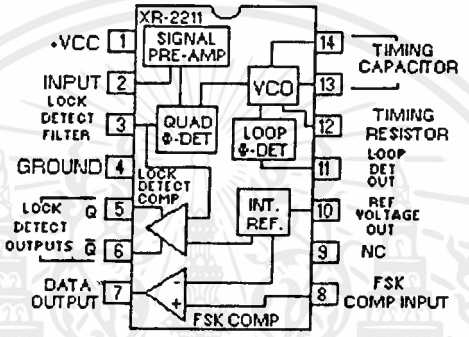
ภาครับนี้ใช้ไอซีเบอร์ XR-2211 ซึ่งได้ถูกออกแบบมาโดยเฉพาะเพื่อทำการแปลงสัญญาณ FSK ให้เป็นสัญญาณดิจิตอล (FSK Demodulation) ,การชิงโครไนเซชั่นข้อมูล (Data

Synchronization) ,การถอดรหัสสัญญาณเสียง (Tone Decoding) การดีเทคสัญญาณ FM (FM Detection) และการดีเทคสัญญาณคลื่นพาห์ (Carrier Detection)

ลักษณะโดยทั่วไปของไอซี XR-2211

ไอซีเบอร์ XR-2211 นี้เป็นไอซีที่ทำงานแบบเฟสล็อกลูป (Phase Locked Loop : PLL) ไฟเลี้ยงที่ใช้กับไอซีนี้อยู่ในช่วงตั้งแต่ 4.5 ถึง 20 โวลท์ และสามารถทำงานในย่านความถี่ตั้งแต่ 0.01 Hz จนถึง 300 kHz นอกจากนั้นยังสามารถรับสัญญาณอินพุทในช่วงที่กว้างได้ตั้งแต่ 2 มิลลิโวลท์จนถาณที่เข้ามามีขนาดน้อยกว่า 2 โวลท์ กอย่างหนึ่งคือ สามารถใช้งานร่วมกับอุปกรณ์ทางลอจิกที่เป็นมาตรฐานได้แก่ ตระกูล DTL ,TTL และ ECL ได้อีกด้วย

โครงสร้างภายในของไอซี XR-2211



รูปที่ 3.5 แสดงโครงสร้างภายในของไอซี XR -2211

โครงสร้างภายในแสดงดังรูปที่ 3.5 โดยมีโครงสร้างหลักเป็นวงจรเฟสล็อกลูปซึ่งประกอบด้วยวงจรปรีแอมปลิฟายเออร์ (Preamplifier) ,วงจรคูณสัญญาณอนาลอก (Analog Multiplier)ซึ่งใช้เป็นวงจรเฟสดีเทกเตอร์ และวงจร VCO (Voltage Control Oscillator) โดยวงจรปรีแอมปลิฟายเออร์นี้ใช้สำหรับขยายสัญญาณอินพุทที่มีขนาดต่ำ ๆ (สูงกว่า 2 มิลลิโวลท์) ให้มีขนาดสูงขึ้น ส่วนวงจรเฟสดีเทกเตอร์แบบคูณนี้ทำงานคล้ายกับเอกคลูซีฟออร์เกต ส่วนวงจร VCO นั้นจะถูกควบคุมความถี่โดยตัวต้านทาน R_0 และกระแสจากวงจรเฟสดีเทกเตอร์

แรงดันอ้างอิง (Reference Voltage : V_R ที่ขา 10) แรงดันที่ขา 10 นี้ใช้เป็นแรงดันอ้างอิงสำหรับแรงดันที่ขา 5 ,8 ,10 และ11โดยที่ขา 10 นี้จะต้องต่อคาปาซิเตอร์ขนาด 0.1μF กับกราวนด์เพื่อบายพาสสัญญาณความถี่สูงลงกราวนด์ และให้วงจรทำงานสม่ำเสมอ

สัญญาณเอาท์พุทจากวงจรลูปเฟสดีเทกเตอร์(Loop Phase Detector Output) ที่ขา 11 นี้เป็นเอาท์พุทที่มีความต้านทานสูงใช้สำหรับลูปเฟสดีเทกเตอร์โดยมีตัวต้านทาน R_1 และคาปาซิเตอร์ C_1 ทำหน้าที่เป็นวงจรลูปฟิลเตอร์ของเฟสล็อกลูป ในกรณีที่ยังไม่มีสัญญาณอินพุทหรือไม่มีความแตกต่างทางเฟสของวงจรเฟสล็อกลูป ระดับแรงดันที่ขา 11 นี้จะมีค่าใกล้เคียงกับแรงดันอ้างอิง VR

การควบคุมความถี่ของวงจร VCO ความถี่ของวงจร VCO ถูกควบคุมจากตัวต้านทาน R_0 ซึ่งสามารถหาได้จากสมการที่ 2

$$f_0 = 1/(R_0C_0) \quad \text{Hz} \quad .....(2)$$

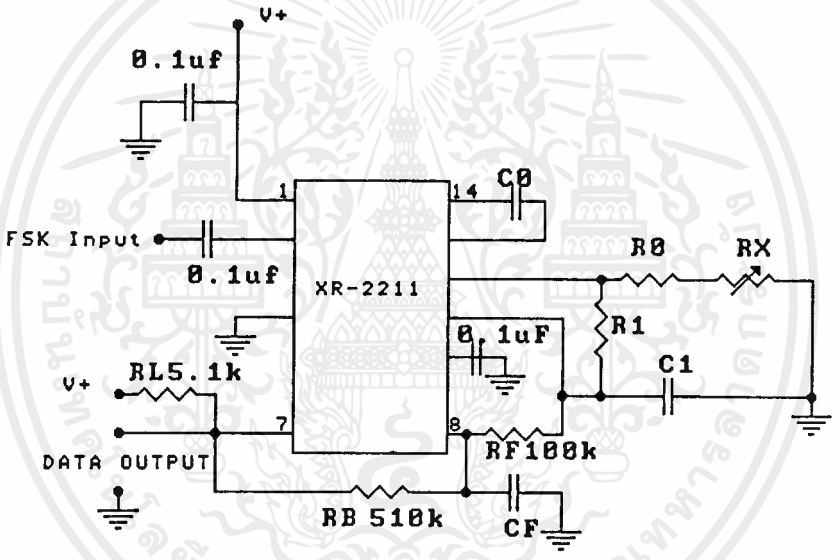
โดยที่ C_0 คือคาปาซิเตอร์ที่ต่อระหว่างขา 13 กับขา 14 และเพื่อความเสถียรภาพของวงจรควรใช้ตัวต้านทาน R_0 มีค่าอยู่ในช่วงระหว่าง 10 กิโลโอห์มและ 100 กิโลโอห์ม

<u>คาปาซิเตอร์ควบคุมวงจร VCO</u> (ที่ขา 13 และ 14) ความถี่ที่ได้จากวงจร VCO แปรผกผันกับค่าของคาปาซิเตอร์ C_0 ที่ต่อระหว่างขา 13 และ 14 คาปาซิเตอร์ C_0 ต้องใช้แบบไม่มีขั้วอยู่ในย่าน 200 พิโคฟารัดจนถึง 10 ไมโครฟารัด

<u>การปรับความถี่ของวงจร VCO</u> ควรใช้ตัวต้านทานที่สามารถปรับค่าได้ต่ออนุกรมกับตัวต้านทานค่าหนึ่ง โดยผลรวมของตัวต้านทานทั้งสองใช้แทนตัวต้านทาน R_0

<u>การนำไอซี XR-2211 ไปใช้ในการดีเทกสัญญาณ FSK</u>

การนำไอซี XR-2211 ไปใช้เพื่อทำการดีเทกรหัสสัญญาณ FSK แสดงดังรูปที่ 3.6



รูปที่ 3.6 แสดงถึงการนำไอซี XR-2211 ไปใช้เพื่อทำการดีเทกรหัสสัญญาณ FSK

จากรูปที่ 3.6 ตัวต้านทาน. R_0 และคาปาซิเตอร์ C_0 ใช้เพื่อกำหนดความถี่กลางของเฟสล็อกลูป (f_0), ตัวต้านทาน R_1 นั้นใช้เพื่อกำหนดแบนด์วิดธ์ ,คาปาซิเตอร์ C_1 ใช้กำหนดค่าคงที่ทางเวลาของลูปฟิลเตอร์และค่าลูปแดมปิง ,คาปาซิเตอร์ C_F และตัวต้านทาน R_F ทำหน้าที่เป็น One Pole Post-detection สำหรับสัญญาณเอาท์พุท , ตัวต้านทาน R_8 (มีค่าประมาณ 510 กิโลโอห์ม)ซึ่งต่อระหว่างขา 7 และ 8 ทำหน้าที่เป็นตัวป้อนกลับทางบวก (Positive Feedback)

<u>ขั้นตอนการกำหนดค่าตัวต้านทานและคาปาซิเตอร์</u>

1.คำนวณความถี่กลางของเฟสล็อกลูป f_0 ดังสมการที่ 3 โดยค่า f_1 และ f_2 คือความถี่ทั้งสองของสัญญาณอินพุท FSK

$$f_0 = (f_1 + f_2) / 2 \quad .....(3)$$

ถ้าทางด้านส่งนั้นได้กำหนดความถี่ไว้ตามมาตรฐานของ CCITT V.23 คือที่บอดเรท 1200บิท ต่อวินาที ซึ่งมีความถี่ f_1 และ f_2มีค่า 1300 และ 2100 Hz ตามลำดับดังนั้นจะได้

$$f_0 = (1300 + 2100) / 2$$
$$= 1700 \text{ Hz}$$

2.เลือกค่าความต้านทาน R_0 ซึ่งค่า R_0 นี้ควรจะอยู่ในช่วง 10 กิโลโอห์มจนถึง 100 กิโลโอห์ม สำหรับค่าความต้านทานของ R_0 ที่ใช้ในโมเด็มไร้สายนั้นมีค่า 10 กิโลโอห์ม

3.คำนวณค่าคาปาซิเตอร์ C_0 ดังสมการที่ 4

$$C_0 = 1 / R_0 f_0 \quad \text{.....(4)}$$

จากค่าความต้านทานของ R_0 จากขั้นตอนที่ 2 นั้นจะได้ค่าของ C_0 ดังนี้

$$C_0 = 1 / (10 \times 10^3 \times 1700)$$
$$= 58.8 \text{ nF}$$

4.คำนวณค่าความต้านทาน R_1 จากสมการที่ 5

$$R_1 = R_0 \times [f_0 / (f_1 - f_2)] \quad \text{.....(5)}$$

จากความถี่จากขั้นตอนที่ 1 และค่าความต้านทานในขั้นตอนที่ 2 จะได้ค่าของ R_1 ดังนี้

$$R_1 = 10 \times 10^3 \times [1700/(2100 - 1300)]$$
$$= 21250 \text{ โอห์ม}$$

5.คำนวณค่าคาปาซิเตอร์ C_1 เพื่อกำหนดลูปแดมปิง (Loop Damping) ซึ่งควรจะมีค่าประมาณ 0.5 ดังนั้นจะได้ค่าของคาปาซิเตอร์ C_1ดังสมการที่ 6

$$C_1 = C_0 / 4 \quad \text{.....(6)}$$

จากค่าของ C_0 ตามขั้นตอนที่ 3 จะได้ค่าของ C_1 ดังนี้

$$C_1 = 58.8 \times 10^{-9} \times 0.25$$
$$= 1.47 \times 10^{-8}$$
$$= 14.7 \text{ nF}$$

6.คำนวณค่าของคาปาซิเตอร์ C_F ถ้ากำหนดตัวต้านทาน R_F มีขนาด 100 กิโลโอห์มและตัวต้านทาน R_B มีขนาด 510 กิโลโอห์ม ดังนั้นจะหาค่าของ C_F ได้จากสมการที่ 7

$$C_F = 3 / \text{Baud Rate } \mu F \quad \text{.....(7)}$$

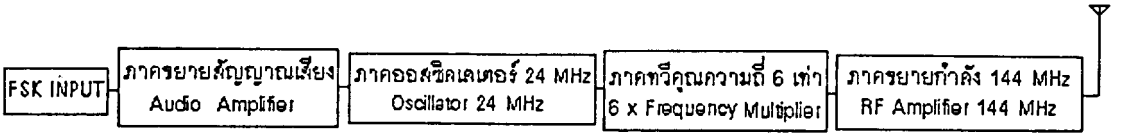
เนื่องจากบอดเรทนั้นใช้เท่ากับ 1200 บิทต่อวินาทีดังนั้นจะได้ค่าของ C_F ดังนี้

$$C_F = 3 / 1200$$
$$= 2.5 \text{ nF}$$

<u>หมายเหตุ</u>

สำหรับค่าของคาปาซิเตอร์ที่ใช้จริงในวงจรนั้นให้ใช้ค่าที่ใกล้เคียงกับค่าที่ได้จากการคำนวณมากที่สุด ส่วนค่าความต้านทานที่ใช้จริงในวงจรนั้นให้ใช้ค่าที่ใกล้เคียงที่น้อยกว่าต่ออนุกรมกับความต้านทานที่ปรับค่าได้เพื่อให้สามารถปรับแต่งให้ได้คุณสมบัติของวงจรที่ดีที่สุด

3.5 หลักการทำงานของวงจรส่งสัญญาณวิทยุ



รูปที่ 3.7 แสดงโครงสร้างการทำงานของวงจรเครื่องส่งสัญญาณวิทยุ

จากรูปที่ 3.7 แสดงโครงสร้างการทำงานของเครื่องส่งสัญญาณวิทยุ สัญญาณ FSK ซึ่งเป็นสัญญาณเสียงจะถูกนำไปเข้าภาคขยายสัญญาณเสียง (AUDIO AMPLIFIER) สัญญาณเสียงที่ออกจากภาคขยายจะถูกส่งต่อเข้าไปมอดูเลทกับสัญญาณความถี่วิทยุ 24 เมกกะเฮิร์ท ซึ่งผลิตขึ้นจากภาคคริสตอลออสซิลเลเตอร์ที่ควบคุมความถี่ด้วยแร่คริสตอล 24 เมกกะเฮิร์ท หลังจากมอดูเลทกับความถี่เสียงแล้ว ความถี่ 24 เมกกะเฮิร์ทจะเปลี่ยนแปลงไปเล็กน้อยตามการเปลี่ยนแปลงของสัญญาณเสียงที่เข้ามามอดูเลทด้วยซึ่งเป็นไปตามหลักการมอดูเลทสัญญาณในระบบFM (FREQUENCY MODULATION) จากนั้นสัญญาณความถี่ 24 เมกกะเฮิร์ทที่ผ่านการมอดูเลทกับสัญญาณเสียงแล้วจะถูกส่งไปยังภาคทวีคูณความถี่ (FREQUENCY MULTIPLIER) เพื่อทวีคูณความถี่ขึ้นไปอีก 6 เท่า จากความถี่ 24เมกกะเฮิร์ทเป็นความถี่144เมกกะเฮิร์ท สัญญาณความถี่144เมกกะเฮิร์ทที่ได้จะถูกส่งไปขยายสัญญาณให้มีกำลังแรงขึ้นที่ภาคขยายกำลัง 144 MHz ก่อนส่งสัญญาณไปออกอากาศที่สายอากาศ

<u>การทำงานของวงจรเครื่องส่งสัญญาณวิทยุ 144 MHz</u>

ในรูปที่ 3.8 เป็นวงจรขยายสัญญาณเสียง (AUDIO AMPLIFIER) และวงจรออสซิลเลเตอร์ซึ่งควบคุมความถี่ด้วยแร่คริสตอล วงจรออสซิลเลเตอร์ดังกล่าวนี้ถึงแม้จะมีการควบคุมความถี่ด้วยแร่คริสตอล แต่ก็เป็นวงจรที่สามารถเปลี่ยนแปลงความถี่ได้บ้างเล็กน้อยเมื่อมีสัญญาณเสียงเข้ามามอดูเลทด้วยเพื่อต้องการให้มีผลในการมอดูเลทสัญญาณแบบ FM วงจรออสซิลเลเตอร์ในลักษณะนี้มีชื่อย่อว่า VXO. (Variable Frequency Crystal Oscillator)

สัญญาณเสียงที่ได้จากภาคแปลงสัญญาณ FSK จะถูกขยายด้วย IC1 และ IC2 เพื่อต้องการให้ได้ระดับสัญญาณสูงมากพอที่จะเป็นไบอัสให้กับวาริแคปไดโอด (Varicab Diode) D1 เกนการขยายสัญญาณของวงจรขยายเสียงนี้หาได้จากอัตราส่วนของ R2 กับ R3 และ R5 กับ R6 จากค่าที่ใช้ในวงจร เกนที่ได้มีค่าประมาณ 1,660

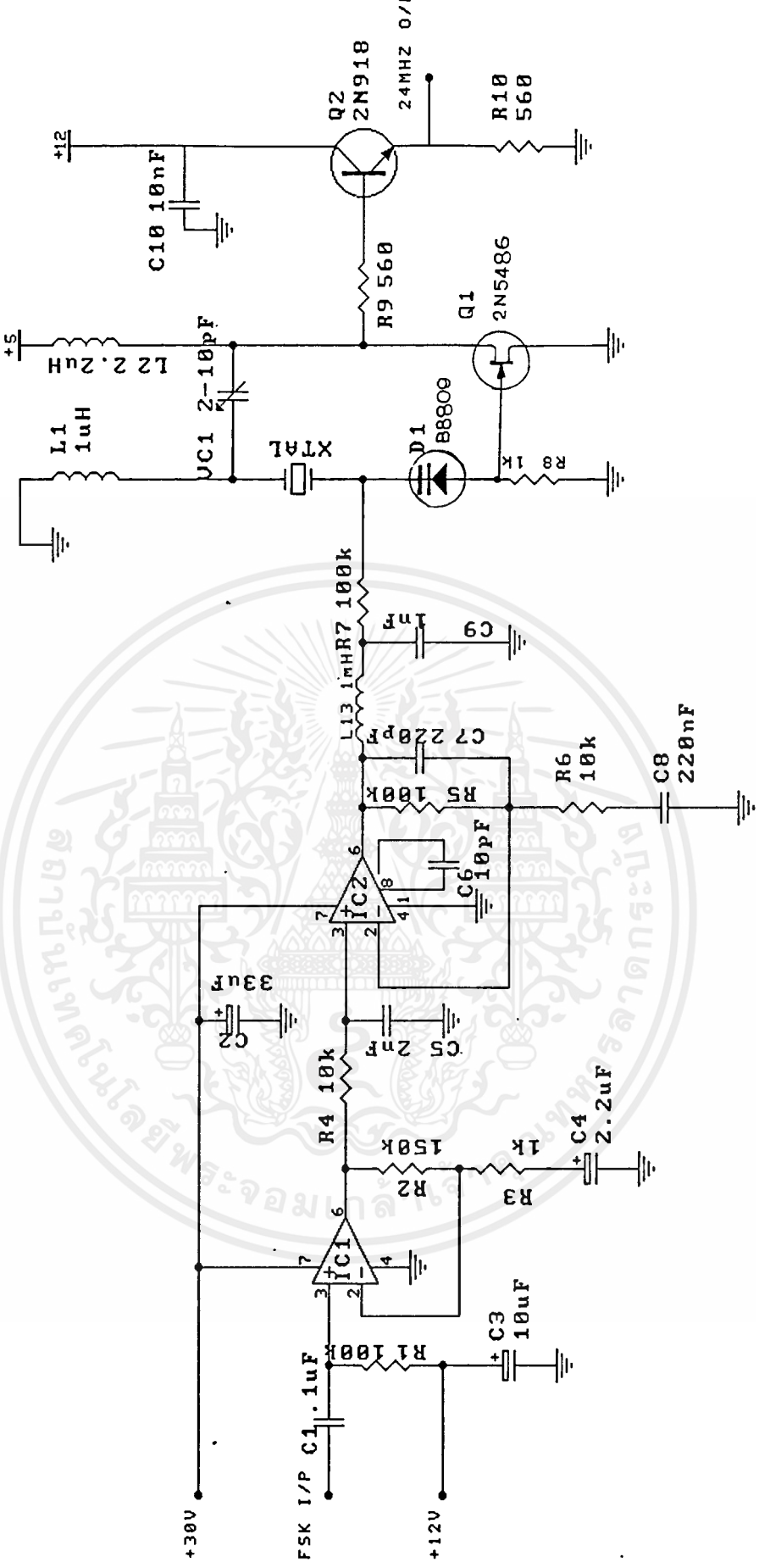
ไฟจ่าย +12โวลท์ที่ผ่านตัวต้านทาน R1เข้าไปยังขา 3 ซึ่งเป็นขาอินพุทสัญญาณบวกของ IC1 เพื่อเป็นการจัดค่าแรงดันไฟเฉลี่ยที่ตกคร่อมวาริแคปไดโอด D1 โดยไอซีออปแอมป์(IC1,IC2)ทั้งสองตัวในวงจรจะทำหน้าที่ในการรักษาระดับแรงไฟนี้

การตอบสนองความถี่ของวงจรขยายสัญญาณเสียงจะถูกจำกัดด้วยวงจรกรองผ่านความถี่ (Bandpass Filter) มี R3 กับ C4 และ R6 กับ C8 ทำหน้าที่ผ่านความถี่ต่ำ(Low pass) ลงกราวน์โดยมีจุดตัด (Cut-Off) ที่ความถี่ 110 Hz ในขณะที่ R4 กับ C5 และ R5 กับ C7 จะเป็นตัวผ่าน

ความถี่สูงลงกราวน์โดยเริ่มตัดที่ความถี่ 4.6 kHz ซึ่งการจำกัดการตอบสนองความถี่ของภาคขยายสัญญาณเสียงนี้จะช่วยลดสัญญาณรบกวนที่นอกเหนือไปจากสัญญาณความถี่เสียงไปด้วยในตัว

วงจรออสซิลเลเตอร์ควบคุมความถี่ด้วยแร่คริสตอลทำงานด้วย FET. Q1 ซึ่งต่อวงจรแบบฮาร์ทเล่ย์ออสซิลเลเตอร์ (Hartley Oscillator) โดยมี L1,VC1 และ L2 ต่อร่วมกันเป็นวงจรจูนที่ความถี่ 24 MHz วาริแคปไดโอด D1 ทำหน้าที่เป็นตัวเปลี่ยนค่าความจุ.(Capacitance) ซึ่งต่ออนุกรมกับแร่คริสตอลเพื่อต้องการให้เกิดผลการมอดูเลททางด้านความถี่ตามวัตถุประสงค์

ทรานซิสเตอร์ Q2 ถูกต่อวงจรแบบ คอมมอนคอลเลคเตอร์(Common Collector)เพื่อต้องการให้เป็นโหลดอิมพีแดนส์สูง (High Impedance Load) แก่วงจรออสซิลเลเตอร์สำหรับทำหน้าที่เป็นบัฟเฟอร์หรือเป็นตัวกันชนในการดึงสัญญาณซึ่งจะมีผลทำให้ภาคออสซิลเลเตอร์มีเสถียรภาพมากขึ้น



รูปที่ 3.8 แสดงวงจรขยายสัญญาณเสียง (AUDIO AMPLIFIER) และวงจรออสซิลเลเตอร์

จากรูปที่ 3.9 สัญญาณจากภาคออสซิลเลเตอร์ที่มีค่าความถี่ประมาณ 24 MHz จะถูกส่งต่อไปยังภาคทวีคูณความถี่ซึ่งมีทรานซิสเตอร์ Q3 ทำหน้าที่เป็นตัวทวีคูณความถี่ขึ้นไป 3 เท่าจากความถี่ 24 MHz เป็นความถี่ 72 MHz และทรานซิสเตอร์ Q4 ทำหน้าที่เป็นตัวทวีคูณความถี่ขึ้นไป 2 เท่าจากความถี่ 72 MHzเป็น ความถี่ 144 MHz

การทวีคูณความถี่ของวงจรนี้เกิดขึ้นได้โดยใช้หลักการส่งพัลส์สั้น ๆ (Short Pulse)เข้าไปในวงจรจูน ในขณะที่เป็นการเว้นสัญญาณของขบวนพัลส์ช่วงสั้น วงจรจูนก็จะออสซิลเลทในตัวเองต่อเนื่องไปจนครบรอบจนถึงพัลส์ลูกใหม่เข้ามา ซึ่งจะมีผลทำให้เกิดการออสซิลเลทอย่างต่อเนื่องขึ้นในวงจรจูนด้วยความถี่เรโซแนนซ์ของตัวมันเอง จากวงจรทวีคูณความถี่ในที่นี้มี Q3 เป็นตัวสร้างขบวนพัลส์สั้น ๆ จากความถี่ที่ส่งเข้ามาทางอินพุททำให้เกิดการออสซิลเลทขึ้นในวงจรจูน L3, C16 และ VC2 ทำให้เกิดสัญญาณออกต่อเนื่องสม่ำเสมอที่ความถี่ 72 MHz ซึ่งเป็นความถี่เรโซแนนซ์ของวงจรจูนนี้

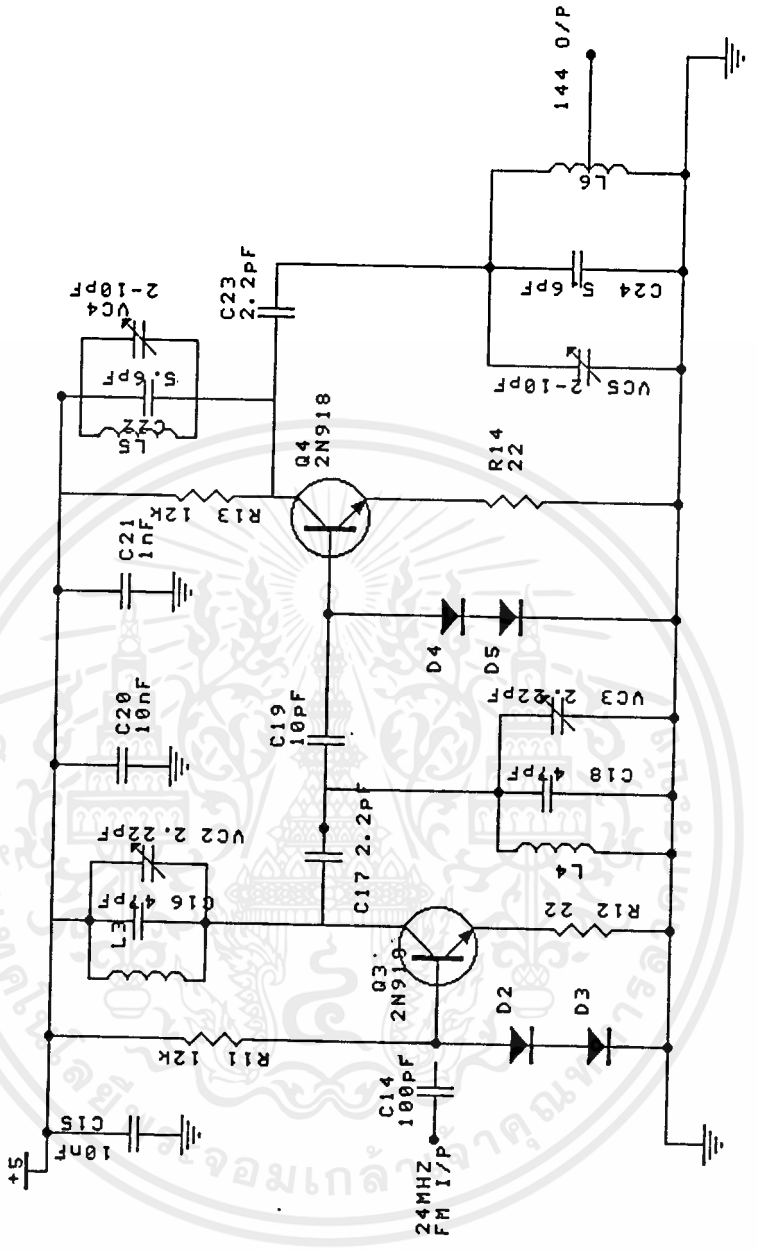
ไดโอด D2 และ D3 เป็นตัวควบคุมแรงดันสัญญาณอินพุทให้มีค่าพีค 1.2 โวลท์โดยสม่ำเสมอ ในขณะเดียวกันที่ตามปรกติจะมีแรงดันตกคร่อมตรงขาเบส-อิมิเตอร์เท่ากับ 0.6 โวลท์ ดังนั้นจึงมีแรงดันพีค (Peak Voltage) 0.6 โวลท์ ตกคร่อมที่ R12 และจะทำให้กระแสคอลเลคเตอร์ที่ไหลผ่านทรานซิสเตอร์ Q3 ค่อนข้างคงที่ จึงเป็นผลทำให้ขนาด (Amplitude) ของสัญญาณออกมีความคงที่มากยิ่งขึ้นแม้ระดับสัญญาณที่เข้ามาทางอินพุทจะมีการเปลี่ยนแปลงไปบ้างก็ตาม

วงจรจูนซึ่งประกอบด้วย L4, C18 และ VC3 ทำหน้าที่เป็นวงจรจูนที่สองเพื่อต้องการจูน ให้ได้ความถี่ฮาร์โมนิคที่บริสุทธิ์ยิ่งขึ้น ทรานซิสเตอร์ Q4 ทำหน้าที่ในการทวีคูณความถี่ 72 MHz เป็นความถี่ 144 MHz ซึ่งใช้หลักการเดียวกันกับที่ได้อธิบายมาแล้วโดยมีการจัดค่าของวงจรจูนซึ่งประกอบด้วย L5, C22 และ VC4 ให้มีความถี่เรโซแนนซ์ที่ 144 MHz และใช้วงจรจูนซึ่งประกอบด้วย L6, C24 และ VC5 ทำหน้าที่เป็นวงจรจูนที่สองเพื่อจูนซ้ำอีกครั้งให้มีความถี่ฮาร์โมนิคบริสุทธิ์ที่ 144 MHz

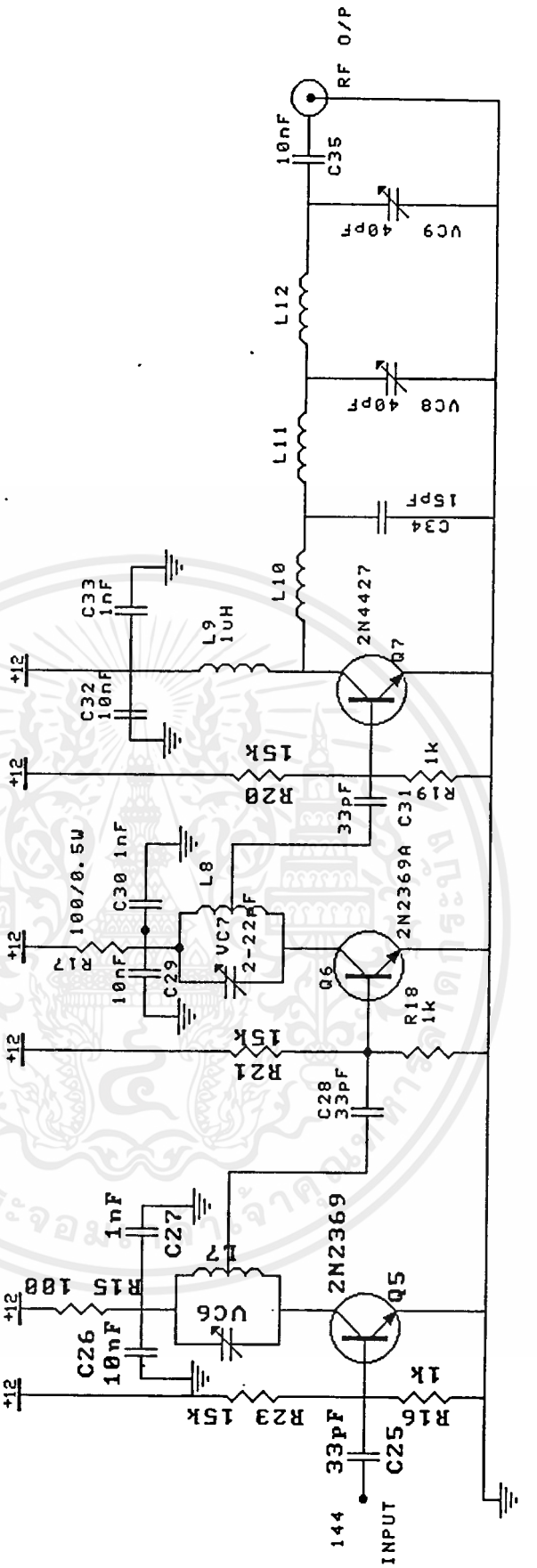
จากรูปที่ 3.10 สัญญาณ RF ความถี่ 144 MHz ที่ได้จากรูปที่ 3.9 จะถูกขยายกำลังขึ้นด้วย Q5,Q6 และ Q7 ที่ถูกจัดเป็นวงจรขยายแบบคลาสเอ (Class A) และสัญญาณที่ได้จากทรานซิสเตอร์แต่ละตัวจะถูกนำไปต่อเข้ากับวงจรจูน (VC6/L7 , VC7/L8 และ L9) ที่ขาคอลเลคเตอร์ของทรานซิสเตอร์เพื่อให้ได้สัญญาณออกเป็นคลื่นรูปซายน์อย่างแท้จริง

ตัวต้านทาน R15 และ R17 ที่ต่อจากแหล่งจ่ายไฟมาเข้า Q5 และ Q6 ทำหน้าที่เป็นตัวป้องกันไม่ให้ทรานซิสเตอร์ทั้งสองตัวนี้ดึงกระแสมากไป ในขณะที่มีการปรับจูนความถี่

วงจรกรองความถี่ต่ำผ่าน (Lowpass Filter) ที่ประกอบด้วย L10, C34, L11, VC8, L12, VC9 และ C35 จะทำหน้าที่เป็นตัวกรองความถี่ 288 MHz ซึ่งเป็นความถี่ฮาร์โมนิคที่ 2 ของความถี่ 144 MHzออก และในขณะเดียวกันก็ทำหน้าที่เป็นตัวแมทชิ่งอิมพิแดนซ์ของวงจร ให้เข้ากับอิมพิแดนซ์ของโหลด 50 โอห์มด้วย



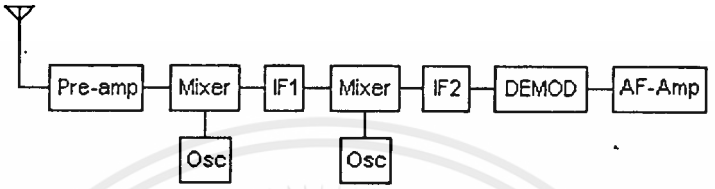
รูปที่ 3.9 แสดงวงจรภาคทวีคูณความถี่ 6 เท่า



รูปที่ 3.10 แสดงวงจรภาคขยายกำลัง 144 MHz

3.6 หลักการทำงานของเครื่องรับสัญญาณวิทยุ 144 MHz

เครื่องรับสัญญาณวิทยุความถี่ 144 MHzนี้ใช้หลักการที่เรียกว่า ดูอัลคอนเวอร์ชั่น (DUAL CONVERSION)ดังรูปที่ 3.11 คือมีการแปลงลงมาเป็นความถี่กลาง 2 ครั้งเนื่องจากเราไม่สามารถแปลงความถี่ในภาคมิกเซอร์ครั้งเดียวจาก 144 MHz เป็น 455 kHz ได้ในครั้งเดียวและนอกจากนั้นก็ยังทำให้ได้ผลไม่ดีเท่าที่ควร ดังนั้นจึงจำเป็นต้องทำการมิกซ์ 2 ครั้ง จาก 144 MHz เป็น 10.7 MHz แล้วจึงมิกซ์อีกครั้งเป็น 455 kHz



รูปที่ 3.11 แสดงโครงสร้างการทำงานของวงจรเครื่องรับสัญญาณวิทยุ

การจัดวงจรให้กับ MC3362 ให้เป็นภาครับทำได้ดังรูป 3.12 เนื่องจากความไวของ MC3362 ไม่พอจึงต้องเพิ่มภาคขยายความถี่วิทยุส่วนหน้า (RF front end) ขึ้นมาโดยใช้ทรานซิสเตอร์ชนิดสัญญาณรบกวนต่ำเบอร์ 2SC3358 จัดวงจรให้ทำงานในคลาสซี ขดลวด L1 เป็นตัวแมตชิ่งอิมพีแดนซ์กับสายอากาศมีตัวเก็บประจุปรับค่า VC1 เป็นตัวปรับให้ได้ความไวสูงสุดสามารถปรับให้ได้ความไวพีคในช่วงความถี่ตั้งแต่ 130 MHz ถึง 170 MHz ส่วนขดลวด L2 เป็นตัวคัปปลิ้งสัญญาณโดยจะรีโซแนนซ์กับค่าความจุของ C3

ในส่วนวงจรภาครับทั้งหมดทำหน้าที่โดย IC1 มี VR1 เป็นตัวปรับความถี่ของโลคัลออสซิเลเตอร์ชุดที่ 1 โดยการปรับแรงดันที่จ่ายให้ภาคกำเนิดความถี่ควบคุมด้วยแรงดันภายในตัวไอซีมีขดลวด L2 และตัวเก็บประจุเป็นเรโซแนนซ์แทงค์ซึ่งจะให้ความถี่สูงสุดเป็น 159 MHz เมื่อปรับ VR1 สูงสุด

ความถี่วิทยุที่จะรับจะถูกป้อนเข้าที่ขา 1 และขา 24 ของ IC1เพื่อทำการมิกซ์ครั้งที่ 1 ได้เป็นความถี่ 10.7 MHz ออกมาที่ขา 20 แล้วผ่านตัวกรองความถี่โดยใช้เซรามิกฟิลเตอร์ CFU1 ขนาด 10.7 MHz ป้อนกลับเข้าไปในไอซีทางขา 18 และขา 19 เพื่อมิกซ์ครั้งที่ 2

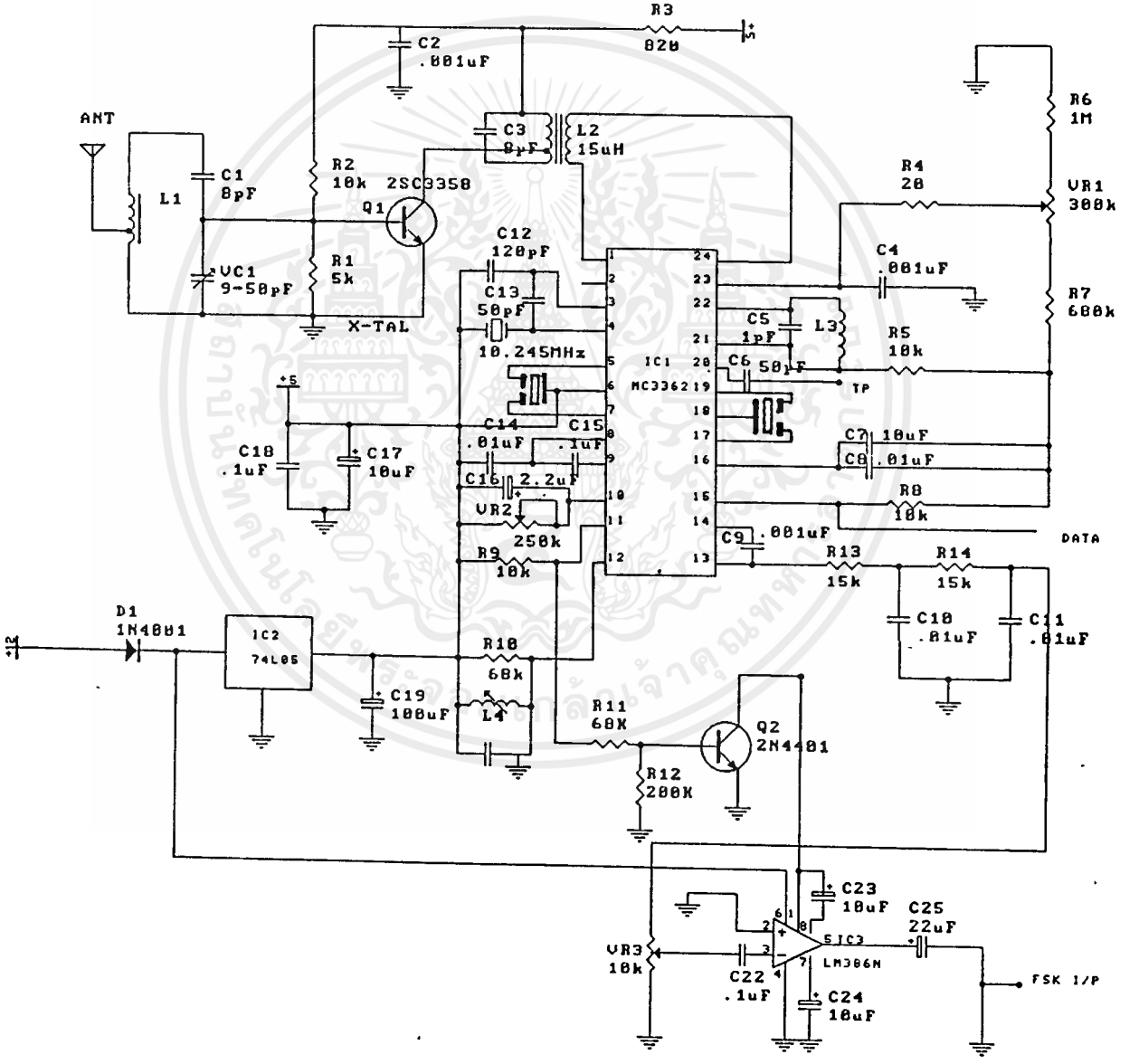
วงจรโลคอลออสซิลเลเตอร์ชุดที่ 2 ใช้คริสตอลความถี่ 10.245 MHz เป็นตัวกำเนิดความถี่ป้อนเข้าที่ขา 3 และขา 4 เพื่อมิกซ์กับความถี่ 10.7 MHz เหลือเป็นความถี่ 455 kHz ออกมาที่ขา 5 ผ่านเซรามิกฟิลเตอร์ความถี่ 455 kHz กลับมาที่ขา 7 เพื่อป้อนให้ภาคลิมิเตอร์ต่อไป

ตัวต้านทานปรับค่า VR2 ต่ออยู่ที่ขา 10 อันเป็นจุดตรวจสอบระดับสัญญาณ ทำหน้าที่เป็นตัวปรับสเควลช์คือให้ตัดเอาท์พุทหากตรวจไม่พบคลื่นพาห์โดยจะมีภาคตรวจจับคลื่นพาห์จากขา 10 ไปขา 11 ภายในไอซี ซึ่งจะให้เอาท์พุทไปควบคุมภาคขยายสัญญาณ FSK ต่อไป

การดีเทคระบบเอฟเอ็มของ MC3362 เป็นแบบควอดราเจอร์ดีเทคเตอร์มีขดลวด L4 เป็นดิสคริมมิเนเตอร์ที่ความถี่ 455 kHz เอาท์พุตที่เป็นสัญญาณ FSK ที่ออกมาที่ขา 13 ผ่านวงจรกรองความถี่ต่ำผ่าน อันประกอบด้วย R13,R14,C10 และ C11

สัญญาณ FSK จากวงจรกรองความถี่ถูกส่งต่อไปให้ IC3 เบอร์ LM386N ทำการขยายกำลังตัวต้านทานปรับค่า VR3 เป็นตัวปรับระดับความแรงของสัญญาณ FSK ส่วนที่ขา 1 ของ IC 3 จะต่ออยู่กับทรานซิสเตอร์ Q2 ซึ่งจะถูกควบคุมจากวงจรตรวจจับคลื่นพาห์อีกทีหนึ่ง ทำหน้าที่เป็นวงจรสเควลช์ (squelch) โดยจะควบคุมให้ IC3 ทำงานเฉพาะเวลาที่ตรวจจับคลื่นพาห์ได้เท่านั้น (ขา 1 ของ LM386N ต่อลงกราวด์ภาคขยายจะไม่ทำงาน) เพื่อเป็นการตัดเสียงช่าเวลาที่มิได้รับสัญญาณใด

ส่วนไอซีเบอร์ 78L05 เป็นไอซีเรกูเลเตอร์รับแรงดัน 9-12 โวลต์ มาทำการลดและควบคุมไว้ที่ 5 โวลต์คงที่เพื่อป้อนให้กับไอซี 1 และวงจรอื่น ๆ

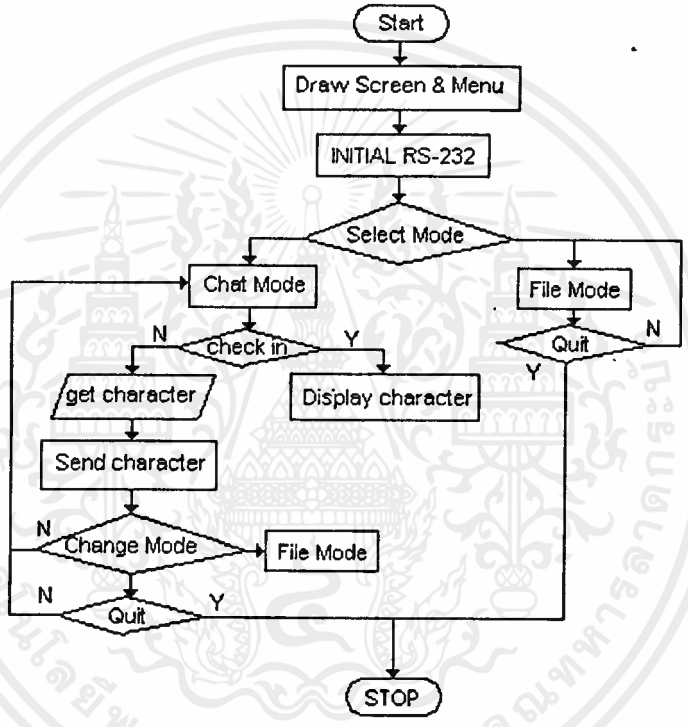


รูปที่ 3.12 แสดงวงจรเครื่องรับสัญญาณวิทยุ 144 MHz

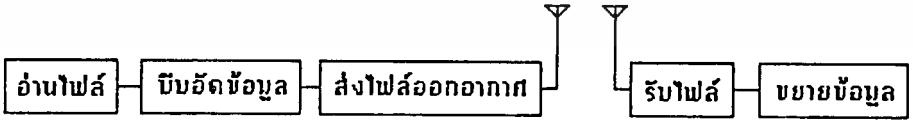
บทที่ 4 โปรแกรมรับและส่งข้อมูล

4.1โครงสร้างการทำงานทั่วไป

สำหรับโปรแกรมที่ได้เขียนขึ้นมานี้สามารถทำการส่งได้ทั้งแบบส่งไฟล์และแบบ Chat mode ซึ่งมีโครงสร้างการทำงานของโปรแกรมดังรูปที่ 4.1 สำหรับการส่งข้อมูลแบบ Chat Mode นั้นก็จะทำการรับส่งข้อมูลตามปกติคือจะทำการส่งข้อมูลออกไปตามที่ได้รับเข้ามา แต่การรับส่งไฟล์นั้นได้นำหลักการของการบีบอัดข้อมูลเข้ามาร่วมด้วยทำให้สามารถส่งข้อมูลจากไฟล์ได้เร็วยิ่งขึ้นซึ่งมีโครงสร้างการทำงานของโปรแกรมรับ-ส่งไฟล์ดังรูปที่ 4.2



รูปที่ 4.1 แสดงการทำงานทั้งหมดของโปรแกรม



รูปที่ 4.2 แสดงการทำงานของฟังก์ชั่นสำหรับการรับ-ส่งไฟล์

จากรูปที่ 4.2 ในตอนแรกนั้นก็จะทำการบีบอัดข้อมูลของไฟล์ที่เลือกขึ้นมาจากเมนู หลังจากนั้นจึงจะทำการส่งข้อมูลที่ได้ทำการบีบอัดแล้วออกไป ส่วนทางด้านรับนั้นเมื่อรับข้อมูลครบหมดแล้วก็จะทำการขยายข้อมูลนั้น

4.2 หลักการบีบอัดข้อมูล

หลักการเข้ารหัสที่ใช้ในโปรแกรมรับส่งนี้ใช้หลักการของ Huffman ซึ่งจะทำการสร้างรหัสที่มีการเปลี่ยนแปลงความยาวของบิทและสัญลักษณ์ที่มีโอกาสพบมากที่สุดโดยจะมีทำให้มีรหัสสั้นที่สุด และก็จะทำการเพิ่มคุณสมบัติเฉพาะเข้าไปทำให้สามารถถอดรหัสได้อย่างถูกต้อง แม้ว่าความยาวของรหัสจะเปลี่ยนแปลงไป ซึ่งในการแปลงรหัสแบบ Huffman นี้จะทำโดยใช้ต้นไม้แปลรหัสเลขฐานสอง (Binary Decoder Tree)

การสร้างต้นไม้แปลรหัสแบบ Huffman นั้นจะทำการสร้างจากล่างขึ้นไปบนโดยเริ่มต้นด้วยกิ่งของต้นไม้และทำงานไปหารากโดยวิธีการสร้างต้นไม้ทำได้โดยนำอักขระแต่ละตัวมาเรียงกันเป็นแถวของ leaf node (เปรียบเสมือนใบ) ซึ่งจะถูกเชื่อมต่อโดย Binary Tree และแต่ละจุดนั้นจะมีน้ำหนักซึ่งเป็นความถี่หรือความน่าจะเป็นของการปรากฏของอักขระแต่ละตัว โดยในการสร้างต้นไม้มีขั้นตอนดังนี้

1.จุดอิสระ 2 จุดซึ่งมีน้ำหนักน้อยที่สุดจะถูกกำหนดขึ้น

2.จุดอีกจุดหนึ่งจะถูกสร้างขึ้นโดยมีน้ำหนักเท่ากับผลรวมของจุด 2 จุดข้างต้นโดยที่จุดใหญ่นี้จะถูกเพิ่มเข้าเป็นจุดอิสระ และจุดเล็ก 2 จุดข้างต้นถูกย้ายออกไป

3.หนึ่งในสองจุดเล็กจะถูกเจาะจงให้เป็นเส้นทางหนึ่งจากจุดใหญ่ เมื่อการแปลรหัส ' 0 ' ส่วนอีกจุดกำหนดเป็น ' 1 '

4.ขั้นตอนที่ทำมาแล้วจะถูกทำซ้ำจนกระทั่งมีเพียงจุดอิสระเพียงจุดเดียวอยู่ทางซ้าย จุดอิสระ นี้จะถูกกำหนดเป็นรากของต้นไม้

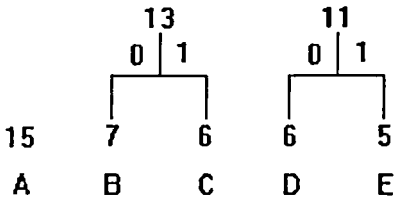
วิธีนี้สามารถประยุกต์ใช้กับสัญลักษณ์ในตัวอย่างดังต่อไปนี้ได้โดยสัญลักษณ์ทั้ง 5 ตัวในข้อความจะถูกแผ่ออกพร้อมกับความถี่ของสัญลักษณ์แต่ละตัวดังต่อไปนี้

15	7	6	6	5
A	B	C	D	E

จุดทั้ง 5 จุดนี้ได้กำหนดให้เป็นกิ่งของต้นไม้แปลรหัส เมื่อขั้นตอนแรกเริ่มต้น มันจะถูกสร้างเป็นจุดอิสระทั้งหมด เราจะมองที่กิ่งของต้นไม้ทั้งหมดเพื่อจะหาจุดอิสระที่มีน้ำหนักน้อยที่สุด 2 จุด ได้แก่ D และ E ซึ่งมีน้ำหนัก 6 และ 5 ตามลำดับ จุดทั้ง 2 จุดนี้จะถูกเชื่อมเข้าด้วยกันเป็นจุดใหญ่ และจะกำหนดน้ำหนักใหม่เป็น 11 แล้วจุด D และ E จะถูกยกเลิกจากการเป็นจุดอิสระ

เมื่อขั้นตอนนี้สำเร็จ เราจะรู้ว่าจุด D และ E จะเป็นจุดที่มีความสำคัญน้อยที่สุดของรหัส D จะถูก ระบุด้วยกิ่ง ' 0 ' ของจุดใหญ่ และ E จะถูกระบุด้วยกิ่ง ' 1 ' ทั้งสองบิตนี้จะเป็น LSB (Least Significant Bit) ของรหัสที่ได้

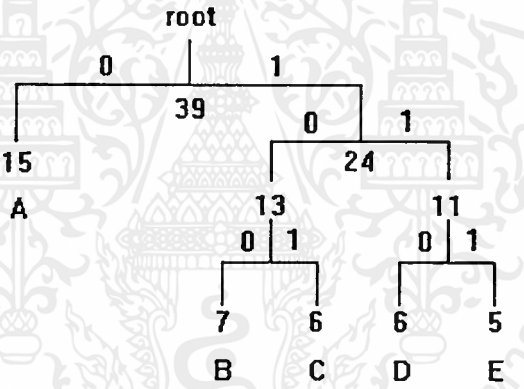
ขั้นตอนต่อไปจุด B และ C จะถูกทำเป็น 2 จุด ซึ่งมีน้ำหนักน้อยที่สุด แล้ว B และ C จะถูกนำมารวมกันเป็นจุดใหญ่จุดใหม่ซึ่งถูกระบุน้ำหนักเป็น 13 แล้ว C และ D จะถูกยกเลิกจากการเป็นจุดอิสระ ซึ่งก็จะได้ต้นไม้แปลรหัสเลขฐานสองดังรูปที่ 4.3



รูปที่ 4.3 แสดงตัวอย่างต้นไม้แปลรหัสเลขฐานสอง

ขั้นตอนต่อไปจุด 2 จุดที่มีน้ำหนักน้อยที่สุด คือจุดใหญ่ของ B/C และ D/E จะถูกเชื่อมต่อเข้าด้วยกันเป็นจุดใหญ่จุดใหม่ซึ่งมีน้ำหนักเป็น 24 และที่จุดใหญ่ของ B/C และ D/E จะไม่นำมาพิจารณาอีก

ในขั้นตอนสุดท้ายจะมีเพียง 2 จุดอิสระอยู่ทางซ้าย จุดใหญ่ที่มีน้ำหนักเป็น 24 จะถูกต่อเข้ากับจุด A เพื่อสร้างจุดใหญ่จุดใหม่ที่มีน้ำหนักเป็น 39 ซึ่งต้นไม้ที่เสร็จสมบูรณ์แล้วแสดงดังรูปที่ 4.4



รูปที่ 4.4 แสดงแสดงตัวอย่างต้นไม้แปลรหัสเลขฐานสองที่เสร็จสมบูรณ์

ในการกำหนดรหัสสำหรับสัญลักษณ์ที่ได้รับ เราจะต้องเดินทางจากกิ่งไปยังรากของ Huffman Tree และรวบรวมบิตใหม่ที่เดินผ่านแต่ละจุดใหญ่ แต่บิตจะถูกส่งกลับมาในลำดับตรงกันข้ามกับที่เราต้องการซึ่งหมายถึงเราต้องเก็บบิตไปไว้ในแสตกแล้วดึงกลับมาเพื่อทำให้เกิดรหัส ด้วยวิธีนี้จะให้โครงสร้างของรหัสของข้อความ ดังตารางต่อไปนี้

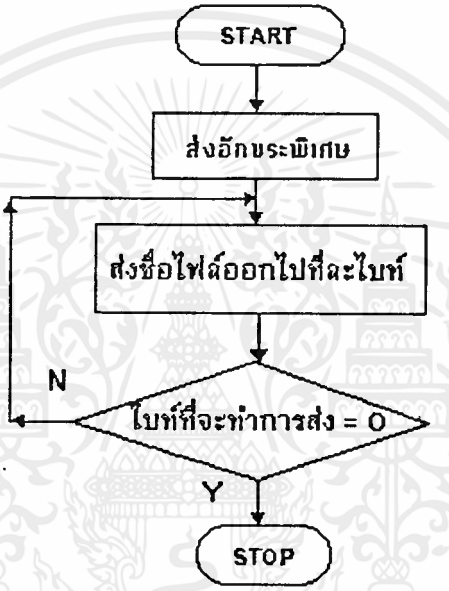
A	0
B	100
C	101
D	110
E	111

4.3 หลักการทำงานของฟังก์ชั่นที่ใช้ในการส่งไฟล์

หลักการในการส่งไฟล์ที่ใช้ในโปรแกรมนี้ก็คือทำการส่งขนาดของไฟล์ไปก่อน จากนั้นจึงส่งข้อมูล ที่อยู่ในไฟล์ออกไป โดยขนาดของไฟล์ที่จะทำการส่งออกไปนั้นใช้เพื่อให้ทางด้านรับทำการรับข้อมูลตามจำนวนไบท์ที่รับเข้ามา เมื่อทำการรับมาครบแล้วจึงค่อยทำการปิดไฟล์

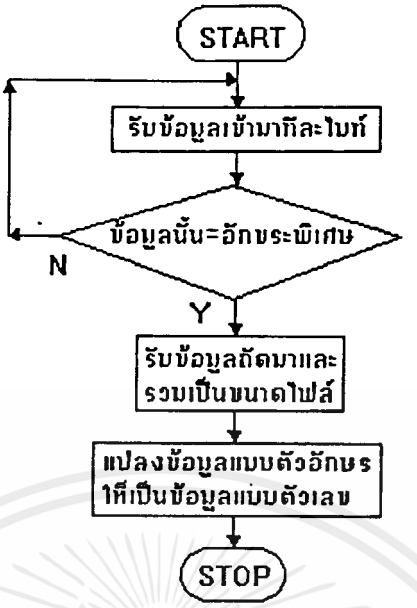
สำหรับฟังก์ชั่นที่ใช้ในการส่งนั้นสามารถแบ่งออกได้เป็น 3 ฟังก์ชั่นหลักได้แก่

1.ฟังก์ชั่น Send_file_name ฟังก์ชั่นนี้เป็นฟังก์ชั่นที่ทำหน้าที่ส่งชื่อไฟล์ไปยังปลายทางโดยในช่วงแรกจะส่งตัวอักษรพิเศษเพื่อบอกว่าข้อมูลที่จะส่งถัดไปเป็นชื่อไฟล์ ซึ่งจะทำการส่งไปทีละตัวอักษรจน กระทั่งตัวสุดท้ายที่มีรหัส ASCII เท่ากับ 0 การทำงานของฟังก์ชั่นนี้แสดงดังรูปที่ 4.5



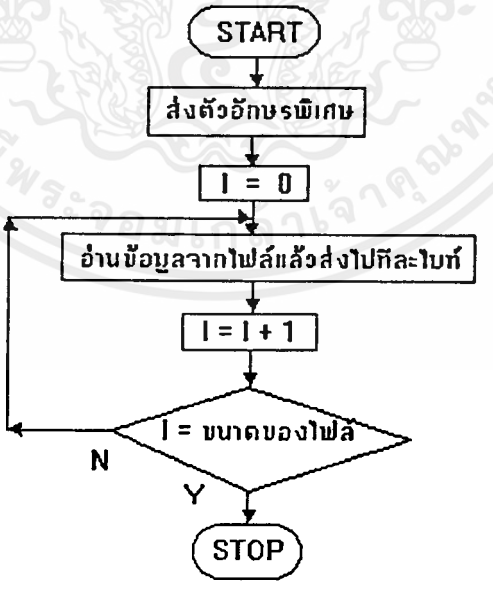
รูปที่ 4.5 แสดงการทำงานของฟังก์ชั่น Send_file_name

2.ฟังก์ชั่น Send_file_size ฟังก์ชั่นนี้เป็นฟังก์ชั่นที่ใช้ในการส่งขนาดของไฟล์ ซึ่งฟังก์ชั่นนี้มีความสำคัญมากถ้าทำการส่งขนาดของไฟล์ผิดพลาดไปก็จะทำให้ข้อมูลที่อยู่ในไฟล์นั้นรับมาได้ไม่ครบหรือรับมาครบแล้วแต่ไม่ทำการปิดไฟล์ ในช่วงแรกของฟังก์ชั่นจะทำการส่งตัวอักษรพิเศษออกไปเพื่อบอกว่าข้อมูลที่จะส่งถัดไปเป็นขนาดของไฟล์ จากนั้นก็จะทำการวัดขนาดของไฟล์ที่จะทำการส่งจากนั้นก็จะทำการแปลงขนาดของไฟล์ที่เป็นข้อมูลแบบตัวเลขให้กลายเป็นข้อมูลแบบตัวอักษร หลังจากนั้นก็จะทำการส่งข้อมูลนั้นออกไปทีละไบท์ การทำงานของฟังก์ชั่นนี้แสดงดังรูปที่ 4.6



รูปที่ 4.6 แสดงการทำงานของฟังก์ชั่น Send_file_size

3.ฟังก์ชั่น Send_file_data ฟังก์ชั่นนี้มีหน้าที่ส่งข้อมูลที่อยู่ในไฟล์ และเช่นเดียวกับฟังก์ชั่นอื่น ๆ คือในช่วงแรกจะทำการส่งตัวอักษรพิเศษเพื่อบอกด้านรับว่าข้อมูลที่จะตามมาเป็นข้อมูลที่อยู่ในไฟล์ จากนั้นก็ทำการส่งข้อมูลที่อยู่ในไฟล์นั้นออกไปทีละไบท์จนครบหมดทั้งไฟล์ การทำงานของฟังก์ชั่นนี้แสดงดังรูปที่ 4.7



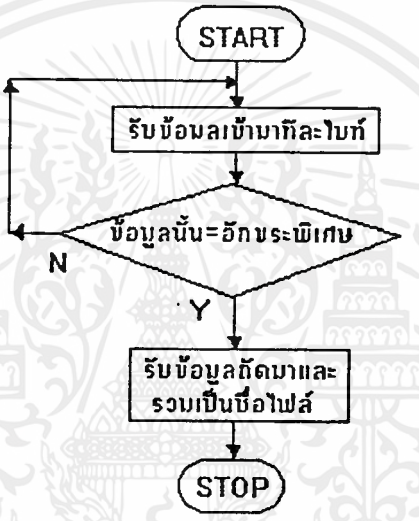
รูปที่ 4.7 แสดงการทำงานของฟังก์ชั่น Send_file_data

4.3 หลักการทำงานของฟังก์ชั่นที่ใช้ในการรับไฟล์

หลักการทำงานของโปรแกรมนี้ก็คือในช่วงแรกจะทำการรับขนาดของไฟล์ หลังจากนั้นจึงทำการรับข้อมูลที่อยู่ในไฟล์ทีละไบท์จนครบตามจำนวนหรือขนาดของไฟล์ที่รับเข้ามาในตอนแรกแล้วจึงปิดไฟล์

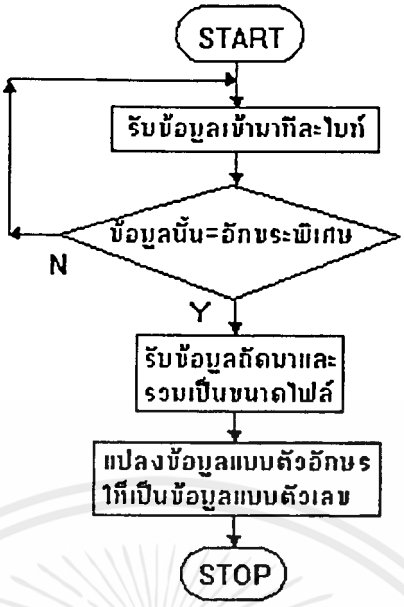
สำหรับฟังก์ชั่นที่ใช้ในการรับไฟล์นั้นสามารถแบ่งออกได้เป็น 3 ฟังก์ชั่นหลักคือ

1.ฟังก์ชั่น Rec_file_name ฟังก์ชั่นนี้เป็นฟังก์ชั่นที่ทำหน้าที่รับชื่อไฟล์เข้ามาโดยในช่วงแรกจะตรวจสอบว่าข้อมูลที่รับเข้ามาเป็นอักขระพิเศษที่ใช้บอกว่าข้อมูลถัดไปเป็นชื่อไฟล์หรือไม่ ซึ่งตัวอักษรพิเศษที่ใช้ในโปรแกรมนี้คือตัว '?' ถ้าไม่ใช่ก็ทำตรวจสอบข้อมูลไบท์ถัดไป ถ้าใช่ก็จะทำการรับข้อมูลถัดไปแล้วรวมเป็นชื่อไฟล์ การทำงานของฟังก์ชั่นนี้สามารถดูได้จากรูปที่ 4.8



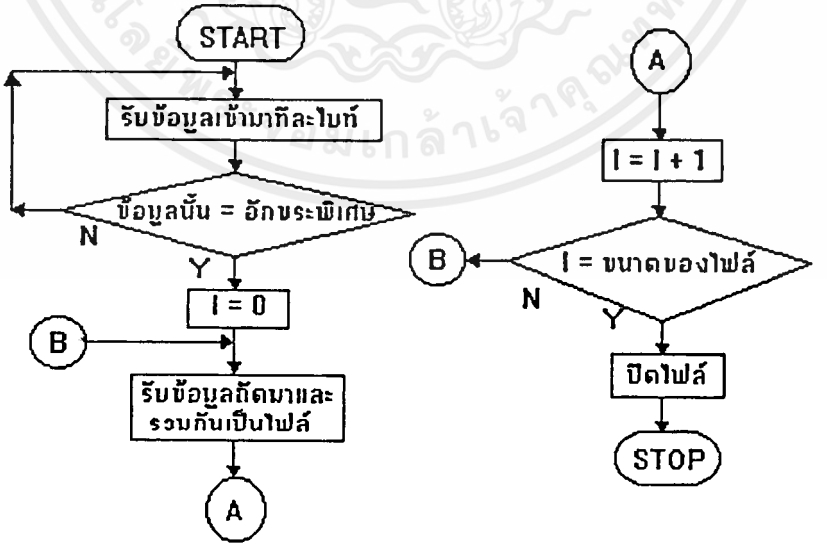
รูปที่ 4.8 แสดงการทำงานของฟังก์ชั่น Rec_file_name

2.ฟังก์ชั่น Rec_file_size ฟังก์ชั่นนี้ทำหน้าที่รับขนาดของไฟล์เข้ามาโดยในช่วงแรกจะทำการตรวจสอบว่าข้อมูลที่รับเข้ามาเป็นอักขระพิเศษที่ใช้บอกว่าข้อมูลถัดไปเป็นขนาดของไฟล์หรือไม่ ซึ่งตัวอักขระพิเศษดังกล่าวที่ใช้ในโปรแกรมตามภาคผนวกนั้นคือตัว '*' เมื่อทำการตรวจสอบแล้วว่าไม่ใช่อักขระพิเศษก็จะทำการตรวจข้อมูลไบท์ถัดไป แต่ถ้าใช่ก็จะทำการรับข้อมูลถัดไปแล้วรวมเป็นขนาดของไฟล์ หลังจากนั้นก็แปลงขนาดของไฟล์ที่เป็นข้อมูลแบบตัวอักษรให้เป็นข้อมูลแบบตัวเลข การทำงานของฟังก์ชั่นนี้แสดงดังรูปที่ 4.9



รูปที่ 4.9 แสดงการทำงานของฟังก์ชั่น Rec_file_size

3.ฟังก์ชั่น Rec_file_data ฟังก์ชั่นนี้ทำหน้าที่รับข้อมูลที่อยู่ในไฟล์ โดยในช่วงแรกจะทำการตรวจสอบว่าข้อมูลที่รับเข้ามาเป็นอักขระพิเศษที่ใช้บอกว่าข้อมูลถัดไปเป็นข้อมูลที่อยู่ในไฟล์หรือไม่ ซึ่งตัวอักขระพิเศษดังกล่าวที่ใช้ในโปรแกรมตามภาคผนวกนั้นคือตัว '$' เมื่อทำการตรวจสอบแล้วว่าไม่ใช่อักขระพิเศษก็จะทำการตรวจข้อมูลไบท์ถัดไป แต่ถ้าใช่ก็จะทำการรับข้อมูลถัดไป แล้วรวมเป็นข้อมูลในไฟล์ พร้อมกันนั้นก็ทำการนับจำนวนไบท์ของข้อมูลที่รับเข้ามาว่าเท่ากับขนาดของไฟล์ที่รับได้จากฟังก์ชั่น Rec_file_size หรือยัง ถ้าไม่เท่าก็จะทำการรับข้อมูลถัดไป แต่ถ้าเท่าก็จะปิดไฟล์



รูปที่ 4.10 แสดงการทำงานของฟังก์ชั่น Rec_file_data

บทที่ 5 ผลการทดลอง

จากการทดลองวัดค่าความผิดพลาดของการส่งข้อมูลออกอากาศโดยใช้โมเด็มไร้สายที่ระยะห่างประมาณ 7 เมตร และทำการเปลี่ยนความถี่ FSK ค่าต่าง ๆ โดยใช้ค่าความห่างของช่วงความถี่ FSK ที่ทำให้ได้อัตราการผิดพลาดของข้อมูลน้อยที่สุดได้ค่าดังตารางที่ 5.1

ความถี่กลาง (เฮิร์ทซ์)	บอดเรท (Baud Rate) (บิทต่อวินาที)	จำนวนตัวอักษรที่ผิดพลาด (ตัวอักษร/2000ตัวอักษร)
2400	600	0
	1200	2
	1800	15
4000	1200	0
	1800	4
	2000	7
	2400	19
5400	1800	0
	2000	4
	2400	7

ตารางที่ 5.1 แสดงอัตราผิดพลาดของข้อมูลต่อความถี่กลาง

จากผลการทดลองสรุปได้ว่า

1. ถ้าความเร็วที่ใช้ในการส่งข้อมูลมีค่าสูงขึ้น อัตราการผิดพลาดของข้อมูลก็จะสูงขึ้นด้วย

2. ที่อัตราเร็วในการส่งเท่ากัน เมื่อความถี่ FSK 2 ความถี่ที่ใช้มีค่าสูงขึ้น ถ้ารับส่งโดยตรงโดยไม่ออกอากาศ (ไม่เปลี่ยนเป็นความถี่วิทยุ) สามารถดีเทคสัญญาณได้ โดยอัตราการผิดพลาดของข้อมูลไม่เพิ่มขึ้น แต่ถ้าส่งออกอากาศด้วยคลื่นวิทยุ อัตราการผิดพลาดของข้อมูลก็จะสูงขึ้นด้วย

จากการทดลองวัดค่าความผิดพลาดของข้อมูลที่ระยะทางต่าง ๆ กันโดยใช้ความเร็วในการส่งข้อมูล 1200 บิทต่อวินาที และใช้ความถี่กลาง 5400 เฮิร์ทซ์ได้ผลการทดลองดังตารางที่ 5.2

ระยะทาง (เมตร)	จำนวนตัวอักษรที่ผิดพลาด (ตัวอักษร/2000ตัวอักษร)
3	0
6	0
9	5
12	12
15	50

ตารางที่ 5.2 แสดงอัตราความผิดพลาดของข้อมูลต่อระยะทาง

นอกจากนี้ที่อัตราเร็วในการส่งเท่ากันเมื่อความถี่ FSK ทั้งสองความถี่ที่ใช้มีค่าต่างกันมาก ถ้ารับส่งโดยตรง โดยไม่ออกอากาศ(ไม่เปลี่ยนเป็นความถี่วิทยุ) สามารถดีเทคสัญญาณได้โดยอัตราการผิดพลาดของข้อมูลไม่เพิ่มขึ้น แต่ถ้าส่งออกอากาศด้วยคลื่นวิทยุอัตราการผิดพลาดของข้อมูลก็จะสูงขึ้น

บทที่ 6 สรุปผลที่ได้จากการทดสอบโมเด็มไร้สาย

จากการทดลองทำการส่งไฟล์ด้วยโมเด็มไร้สายโดยใช้โปรแกรมตามภาคผนวกนั้นพบว่า

1.ขณะที่ยังไม่ทำการส่งข้อมูลไปนั้นในบางครั้งทางด้านรับก็ข้อมูลเข้ามาด้วย ซึ่งข้อมูลที่รับเข้ามานั้นเกิดจากสัญญาณรบกวนภายนอก (Noise)เช่น สัญญาณรบกวนจากเครื่องคอมพิวเตอร์

2.ความถี่ของสัญญาณวิทยุที่ใช้อยู่ในย่านของวิทยุสมัครเล่นทำให้มีโอกาสในการรับข้อมูลได้ผิดพลาดได้ง่าย

3.ถ้าทำการส่งไฟล์ขนาดใหญ่ ๆ ต้องใช้เวลาในการส่งมากหรือกล่าวอีกอย่างหนึ่งว่าบอดเรท(Baud Rate) ที่ใช้ส่งนั้นมีขนาดต่ำเกินไป

4.ทางด้านรับไม่ทราบว่าเมื่อใดจะมีข้อมูลเข้ามา ดังนั้นจะต้องรันโปรแกรม ตลอดเวลาเพื่อเตรียมตัวรับข้อมูลตลอดเวลาทำให้เครื่องคอมพิวเตอร์ด้านรับไม่สามารถใช้งานอื่นได้

5.ในบางครั้งทางด้านรับรับขนาดของไฟล์เข้ามาผิด ถ้าขนาดของไฟล์ที่รับเข้ามามีค่าน้อยกว่าค่าจริงทำให้ข้อมูลที่อยู่ในไฟล์นั้นรับได้ไม่ครบ แต่ถ้าขนาดของไฟล์ที่รับเข้ามามีค่ามากกว่าค่าจริงทำให้โปรแกรมรับไม่ทำการปิดไฟล์ถึงแม้ว่าจะรับข้อมูลมาครบแล้วก็ตาม

6.หลักการบีบอัดข้อมูลที่ใช้นั้นสามารถทำการบีบอัดข้อมูลได้เพียงไม่กี่เปอร์เซ็นต์ซึ่งทำให้ทำการรับส่งได้เร็วขึ้นเพียงเล็กน้อยเท่านั้น

สำหรับแนวทางการพัฒนาทางด้านตัวโมเด็มไร้สายนั้นจะต้องออกแบบให้เครื่องรับส่งสัญญาณวิทยุมีแบนด์วิดธ์แคบ และแบนด์พาสฟิวเตอร์ต้องมีความคมมาก ๆ คือเลือกเอาเฉพาะย่านความถี่ FSK ที่ใช้งานเท่านั้น และต้องออกแบบเครื่องรับให้มี Selectivity มาก ๆ โดยการใช้เฟสล็อกลูปมาควบคุมในส่วนที่ผลิตโลคัลออสซิลเลเตอร์ อีกทั้งต้องออกแบบสายอากาศส่ง-รับให้มีค่าเกนมาก ๆ เพื่อป้องกันสัญญาณรบกวนที่มาจากที่ต่าง ๆ

สำหรับแนวทางการพัฒนาซอฟท์แวร์นั้นควรจะสร้างโปรแกรมที่สามารถตรวจจับข้อผิดพลาดต่าง ๆ และควรจะเขียนโปรแกรมในลักษณะของอินเทอร์รัพท์ในรูปของโปรแกรมฝังตัวคือไม่ต้องรันโปรแกรมตลอดเวลาและจะรันโปรแกรมเมื่อมีข้อมูลเข้ามาเท่านั้น อีกทั้งควรจะใช้เทคนิคการเข้ารหัสที่สามารถทำการบีบอัดข้อมูลได้มากกว่านี้

ภาคผนวก

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <alloc.h>
#include <bios.h>
#include <time.h>
#include <dir.h>

#define On  1
#define Off 0
#define Yes 1
#define No  0
#define F1        0x3B00
#define F2        0x3C00
#define F3        0x3D00
#define F4        0x3E00
#define F5        0x3F00
#define F6        0x4000
#define F7        0x4100
#define F8        0x4200
#define F9        0x4300
#define F10       0x4400
#define ALT_A     0x1E00
#define ALT_B     0x3000
#define ALT_C     0x2E00
#define ALT_D     0x2000
#define ALT_E     0x1200
#define ALT_F     0x2100
#define ALT_G     0x2200
#define ALT_H     0x2300
#define ALT_I     0x1700
#define ALT_J     0x2400
#define ALT_K     0x2500
#define ALT_L     0x2600
#define ALT_M     0x3200
#define ALT_N     0x3100
#define ALT_O     0x1800
#define ALT_P     0x1900
#define ALT_Q     0x1000
#define ALT_R     0x1300
#define ALT_S     0x1F00
#define ALT_T     0x1400
#define ALT_U     0x1600
#define ALT_V     0x2F00
#define ALT_W     0x1100
#define ALT_X     0x2D00
#define ALT_Y     0x1500
#define ALT_Z     0x2C00
#define LEFT      0x4B00
#define RIGHT     0x4D00
#define UP        0x4800
#define DOWN      0x5000
#define ENTER     0x1C0D
#define ESC       0x011B
#define END_OF_STREAM 256
#define max_menu 5

typedef struct {
        int  max;
        int  pos;
        int  x_pos;
        int  y_pos;
        int  width;
        char command[9][20];
} pull;
typedef struct bit_file {
        FILE *file;
        unsigned char mask;
        int rack;
        int pacifier_counter;
} BIT_FILE;
typedef struct tree_node {
        unsigned int count;
        unsigned int saved_count;
        int child_0;
        int child_1;
} NODE;
```

```
typedef struct code     {
        unsigned int code;
        int code_bits;
} CODE;
typedef union char_int  {
        int i;
        char c[2];
} char_int;
typedef union char_long {
        long l;
        char ch[4];
} char_long;

void send();
void drive();
void f_open();
void receive();
void rec_chat();
void dos_shell();
void chat_mode();
void programmer();
void save_config();
void this_program();
void expansion_util();
void compress_util();
void set_user_port();
void main_menu(void);
void set_delay_time();
void initial_chat_mode();
void initial_file_mode();
void send_chat(char *str);
void file_transfer_mode();
void send_out(unsigned char ch);
void prepare_send_chat(char ch);
void print_char(int c,FILE *fp );
void init_menu_command(void);
void CloseInputBitFile(BIT_FILE *bit_file );
void border(int x1,int y1,int x2,int y2);
void clrscr_attr(int x1,int y1,int x2,int y2);
void input_counts(BIT_FILE *input,NODE *nodes );
void FilePrintBinary(FILE *file,unsigned int code,int bits);
void count_bytes(FILE *input,unsigned long *counts );
void scale_counts(unsigned long *counts,NODE *nodes );
void convert_tree_to_code(NODE *nodes,CODE *codes,unsigned int code_so_far,int  bits,int node );
int expand_file();
int  send_file();
int  rec_file();
int check_out();
int compress_file();
int CompressFile(FILE *input,BIT_FILE *output);
int CloseOutputBitFile(BIT_FILE *bit_file );
int OutputBit(BIT_FILE *bit_file,int bit );
int  OutputBits(BIT_FILE *bit_file,unsigned long code,int count );
int  output_counts(BIT_FILE *output,NODE *nodes );
int build_tree(NODE *nodes );
int ExpandFile(BIT_FILE *input,FILE *output);
int  expand_data(BIT_FILE *input,FILE *output,NODE *nodes,int root_node );
int  compress_data(FILE *input,BIT_FILE *output,CODE *codes );
int InputBit(BIT_FILE *bit_file );
long file_size(FILE *fptr);
char *current_directory(char *path);
unsigned long InputBits(BIT_FILE *bit_file,int bit_count );
BIT_FILE *OpenOutputBitFile(char *name );
BIT_FILE *OpenInputBitFile(char *name );

char menu_text[max_menu][6]={"File","Setup","Mode","Util","About");
char menu_xpos[max_menu]={4,11,19,26,33};
char *data_temp_node,*data_temp;
char far *vga;
char comp_menu[13];
char far *fname;
char far *ftemp;
char far *Program;
char *str_send;   // For Chat Mode
char mode;
int  funckey,cancle;
int  delay_time = 0;
int menu_pos=0;
int pull_act=0;
unsigned int data_size;
```

```
long f_size;
int pos_str,pos_x_send;
int pos_y_send,pos_x_rec,pos_y_rec;
int menu_on_off[3];
int Port_number;
int kbpress;
int compression_switch;
int no_user_port;
int no_user_addr[2];
pull p_menu[5];
pull set_menu[7];

void main(int argc,char *argv[])
{
        Program = (char far *) farmalloc(100);
        strcpy(Program,argv[0]);
        vga=(char far *)MK_FP(0xB800,0);
        data_temp_node=(char far *)farmalloc(65535u);
        data_temp=data_temp_node;
        data_size=0;
        clrscr();
        textattr(0x1F);
        init_menu_command();
        main_menu();
        window(1,1,80,25);
        textattr(0x07);
        clrscr();
        farfree(data_temp_node);
}

int readfunc(void)
{
char_int ch;
 if(kbhit())
        {
                ch.i = bioskey(0);
                if(ch.c[0])  funckey = 0;
                else funckey = 1;
                kbpress = Yes;
                return ch.i;
        }
 else return 0;
}

void rs232_initial(void)
{
int i;
char initial_data;
char_int set_baud_rate_data;
        initial_data = 0;
        switch(set_menu[3].pos)
         {
                case 0 : initial_data = initial_data | 0x2 ; break;
                case 1 : initial_data = initial_data | 0x3 ; break;
               }
        if(set_menu[2].pos)  initial_data = initial_data | 4;
        if(set_menu[1].pos!=0)
        switch(set_menu[1].pos)
         {
                case 1 : initial_data = initial_data | 0x8 | 0x10; break;
                case 2 : initial_data = initial_data | 0x8; break;
         }
       switch(set_menu[0].pos)
         {
                case 0 : set_baud_rate_data.i = 0x18;break;
                case 1 : set_baud_rate_data.i = 0x20;break;
                case 2 : set_baud_rate_data.i = 0x30;break;
                case 3 : set_baud_rate_data.i = 0x3A;break;
                case 4 : set_baud_rate_data.i = 0x40;break;
                case 5 : set_baud_rate_data.i = 0x60;break;
                case 6 : set_baud_rate_data.i = 0xC0;break;
                case 7 : set_baud_rate_data.i = 0x180;break;
         }
       if(set_menu[4].pos==0)  Port_number = 0x3F8;
       if(set_menu[4].pos==1)  Port_number = 0x2F8;
       if(set_menu[4].pos==2)  Port_number = no_user_addr[0];
       if(set_menu[4].pos==3)  Port_number = no_user_addr[1];
        outp(Port_number+3,0x80);
        outp(Port_number,set_baud_rate_data.c[0]);
```

```
        outp(Port_number+1,set_baud_rate_data.c[1]);
        outp(Port_number+3,initial_data);
        for(i=1;i<79;i++)               *(vga+i*2+3680)   = 0;
        gotoxy(2,24);
        cprintf("  %s , %s , %s , %s , %s",set_menu[0].command[set_menu[0].pos]
                ,set_menu[1].command[set_menu[1].pos]
                ,set_menu[2].command[set_menu[2].pos]
                ,set_menu[3].command[set_menu[3].pos]
                ,set_menu[4].command[set_menu[4].pos]);
        for(i=1;i<79;i++)
                *(vga+i*2+3680+1) = 0x1F;
}

void init_menu_command(void)
{
FILE *fptr;
char *file_name,ch;
char *baud,*data,*stop,*parity,*no_user_port_str;
char *cp_switch,*port,*user_port_addr_str[2],*delay_time_str;
int size_str,i,flag;
        mode = 'F';
        ftemp = (char far *)farmalloc(13);
        *fname = *ftemp = 0;
        menu_on_off[0] = On;
        menu_on_off[1] = 0;
        menu_on_off[2] = Off;
        p_menu[0].max=6;
        p_menu[0].pos=0;
        p_menu[0].x_pos=2;
        p_menu[0].y_pos=1;
        p_menu[0].width=15;
        strcpy(p_menu[0].command[0],"Open...    F3");
        strcpy(p_menu[0].command[1],"Drive       ");
        strcpy(p_menu[0].command[2],"Receive    F6");
        strcpy(p_menu[0].command[3],"Send       F5");
        strcpy(p_menu[0].command[4],"DOS shell    ");
        strcpy(p_menu[0].command[5],"Quit    Alt+X");
        p_menu[1].max=9;
        p_menu[1].pos=0;
        p_menu[1].x_pos=9;
        p_menu[1].y_pos=1;
        p_menu[1].width=14;
        strcpy(p_menu[1].command[0],"Baud rate   ");
        strcpy(p_menu[1].command[1],"Parity      ");
        strcpy(p_menu[1].command[2],"Data bit    ");
        strcpy(p_menu[1].command[3],"Stop bit    ");
        strcpy(p_menu[1].command[4],"Port        ");
        strcpy(p_menu[1].command[5],"Delay time  ");
        strcpy(comp_menu,"Compress ");
        strcat(comp_menu,"Off");
        strcpy(p_menu[1].command[6],comp_menu);
        strcpy(p_menu[1].command[7],"User Port   ");
        strcpy(p_menu[1].command[8],"Save        ");
        p_menu[2].max=2;
        p_menu[2].pos=0;
        p_menu[2].x_pos=17;
        p_menu[2].y_pos=1;
        p_menu[2].width=20;
        strcpy(p_menu[2].command[0],"Chat Mode         ");
        strcpy(p_menu[2].command[1],"File Transfer Mode");
        p_menu[3].max=2;
        p_menu[3].pos=0;
        p_menu[3].x_pos=24;
        p_menu[3].y_pos=1;
        p_menu[3].width=10;
        strcpy(p_menu[3].command[0],"Compress");
        strcpy(p_menu[3].command[1],"Expand  ");
        p_menu[4].max=2;
        p_menu[4].pos=0;
        p_menu[4].x_pos=31;
        p_menu[4].y_pos=1;
        p_menu[4].width=14;
        strcpy(p_menu[4].command[0],"This Program");
        strcpy(p_menu[4].command[1],"Programmer");
        set_menu[0].max=8;
        set_menu[0].pos=3;
        set_menu[0].x_pos=23;
        set_menu[0].y_pos=2;
        set_menu[0].width=14;
        strcpy(set_menu[0].command[0],"4800 Bit/Sec");
```

```
    strcpy(set_menu[0].command[1],"3600 Bit/Sec");
    strcpy(set_menu[0].command[2],"2400 Bit/Sec");
    strcpy(set_menu[0].command[3],"2000 Bit/Sec");
    strcpy(set_menu[0].command[4],"1800 Bit/Sec");
    strcpy(set_menu[0].command[5],"1200 Bit/Sec");
    strcpy(set_menu[0].command[6]," 600 Bit/Sec");
    strcpy(set_menu[0].command[7]," 300 Bit/Sec");
    set_menu[1].max=3;
    set_menu[1].pos=0;
    set_menu[1].x_pos=23;
    set_menu[1].y_pos=3;
    set_menu[1].width=13;
    strcpy(set_menu[1].command[0],"None Parity");
    strcpy(set_menu[1].command[1],"Even Parity");
    strcpy(set_menu[1].command[2],"Odd  Parity");
    set_menu[2].max=2;
    set_menu[2].pos=1;
    set_menu[2].x_pos=23;
    set_menu[2].y_pos=4;
    set_menu[2].width=17;
    strcpy(set_menu[2].command[0],"7 Bit/Charactor");
    strcpy(set_menu[2].command[1],"8 Bit/Charactor");
    set_menu[3].max=2;
    set_menu[3].pos=0;
    set_menu[3].x_pos=23;
    set_menu[3].y_pos=5;
    set_menu[3].width=12;
    strcpy(set_menu[3].command[0],"1 Stop Bit");
    strcpy(set_menu[3].command[1],"2 Stop Bit");
    set_menu[4].max=2+no_user_port;
    set_menu[4].pos=0;
    set_menu[4].x_pos=23;
    set_menu[4].y_pos=6;
    set_menu[4].width=14;
    strcpy(set_menu[4].command[0],"Port COM 1");
    strcpy(set_menu[4].command[1],"Port COM 2");
    strcpy(set_menu[4].command[2],"User Port 1");
    strcpy(set_menu[4].command[3],"User Port 2");
    set_menu[6].max=2;
    set_menu[6].pos=0;
    set_menu[6].x_pos=23;
    set_menu[6].y_pos=9;
    set_menu[6].width=14;
    strcpy(set_menu[6].command[0],"Compress Off");
    strcpy(set_menu[6].command[1],"Compress On");
    file_name = (char *) malloc(20);
    strcpy(file_name,Program);
    size_str = strlen(file_name);
    *(file_name+size_str-3) = 'C';
    *(file_name+size_str-2) = 'F';
    *(file_name+size_str-1) = 0;
    fptr = fopen(file_name,"r");
    if(fptr!=NULL)
    {
        baud = (char *) malloc(20);      no_user_port_str = (char *) malloc(20);
        data      = (char *) malloc(20); stop = (char *) malloc(20);
        parity    = (char *) malloc(20); port = (char  *) malloc(20);
        cp_switch = (char *) malloc(20); delay_time_str = (char *) malloc(20);
        user_port_addr_str[0] =  (char *) malloc(20);
        user_port_addr_str[1] =  (char *) malloc(20);
        i = flag = 0;
        while(1)
            {
            ch = getc(fptr);
            if(ch==EOF) break;
            if(ch=='\n')
             {
                    switch(flag)
                    {
                        case 0 :  *(baud+i)     = 0;    break;
                        case 1 :  *(parity+i)   = 0;    break;
                        case 2 :  *(data+i)             = 0;    break;
                        case 3 :  *(stop+i)    = 0; break;
                        case 4 :  *(port+i)     = 0; break;
                        case 5 :  *(delay_time_str+i) = 0; break;
                        case 6 :  *(cp_switch+i) = 0; break;
                        case 7 :  *(no_user_port_str+i) = 0; break;
                        case 8 :  *(user_port_addr_str[0]+i) = 0; break;
                        case 9 :  *(user_port_addr_str[1]+i) = 0; break;
                    }
```

```
                                flag = flag + 1; i = 0;
                                continue;
                        }
                        switch(flag)
                        {
                                case 0 :  *(baud+i)     = ch;   break;
                                case 1 :  *(parity+i)   = ch;   break;
                                case 2 :  *(data+i)     = ch;     break;
                                case 3 :  *(stop+i)     = ch;    break;
                                case 4 :  *(port+i)     = ch;   break;
                                case 5 :  *(delay_time_str+i) = ch;break;
                                case 6 :  *(cp_switch+i) = ch;     break;
                                case 7 :  *(no_user_port_str+i) = ch; break;
                                case 8 :  *(user_port_addr_str[0]+i) = ch; break;
                                case 9 :  *(user_port_addr_str[1]+i) = ch; break;
                        }
                        i = i + 1;
                        }
                set_menu[0].pos = *(baud+5) - 48;
                set_menu[1].pos = *(parity+7) - 48;
                set_menu[2].pos = *(data+5) - 48;
                set_menu[3].pos = *(stop+5) - 48;
                set_menu[4].pos = *(port+5) - 48;
                delay_time =  atoi(delay_time_str+6);
                compression_switch = set_menu[6].pos = *(cp_switch+9)-48;
                if(compression_switch)
                 {
                        *(p_menu[1].command[6]+10) = 'n';
                        *(p_menu[1].command[6]+11) = 0;
                 }
                no_user_port = atoi(no_user_port_str);
                for(i=0;i<no_user_port;i++)
                 {
                        no_user_addr[i] = atoi(user_port_addr_str[i]);
                        free(user_port_addr_str[i]);
                        }
                free(baud);   free(data);   free(stop);       free(stop);
                free(port);   free(parity); free(cp_switch);  free(delay_time_str);
         }
        free(file_name);
}

int pull_down(pull *menu)
{
int ch;
void *ptr;
int x1,x2,y1,y2;
int i,j;
        x1=menu->x_pos;
        x2=menu->x_pos+menu->width+1;
        y1=menu->y_pos;
        y2=y1+menu->max+1;
        ptr=(void *)farmalloc(1024);
        window(1,1,80,25);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        for(i=0;i<menu->max;i++) {
                gotoxy(x1+3,y1+2+i);
                printf("%s",menu->command[i]);
        }
        clrscr_attr(x1,y1,x2,y2);
        do
                {
                if(mode=='C')   rec_chat();
                for(i=x1+1;i<x2;i++)
                        *(vga+(y1+menu->pos+1)*160+i*2+1)=0x07;
                ch=readfunc();
                 if(!funckey)
                        if((ch==ENTER)&&(menu==&p_menu[1]))
                        {
                        if(p_menu[1].pos == 5)  set_delay_time();
                        else if(p_menu[1].pos == 7)     set_user_port();
                        else if(p_menu[1].pos == 8) save_config();
                        else
                        {
                                ch=pull_down(&set_menu[p_menu[1].pos]);
                                if(set_menu[6].pos==0)
                                 {
                                        strcpy(comp_menu,"Compress ");
                                        strcat(comp_menu,"Off");
```

```
                                strcpy(p_menu[1].command[6],comp_menu);
                                compression_switch = Off;
                                }
                        else
                                {
                                strcpy(comp_menu,"Compress ");
                                strcat(comp_menu," On");
                                strcpy(p_menu[1].command[6],comp_menu);
                                compression_switch = On;
                                }
                        if(ch==ESC)     ch=0;
                        }       // else of if (p_menu[1].pos == 5)      set_delay_time();
                ) //  if((ch==ENTER)&&(menu==&p_menu[1]))
        if(funckey)
                {
                switch(ch)
                        {
                        case UP:
                                        for(i=x1+1;i<x2;i++)
                                         *(vga+(y1+menu->pos+1)*160+i*2+1)=0x70;
                                        if(menu->pos>0)menu->pos--;
                                         else menu->pos=menu->max-1;
                                        for(i=x1+1;i<x2;i++)
                                         *(vga+(y1+menu->pos+1)*160+i*2+1)=0xF0;
                                        break;
                                case DOWN: {
                                        for(i=x1+1;i<x2;i++)
                                         *(vga+(y1+menu->pos+1)*160+i*2+1)=0x70;
                                        if(menu->pos<menu->max-1)menu->pos++;
                                        else menu->pos=0;
                                        for(i=x1+1;i<x2;i++)
                                         *(vga+(y1+menu->pos+1)*160+i*2+1)=0x0F;
                                        break;
                                         }
                                } // switch
                        }                       //  if(kbhit())
        } while( ( !(funckey &&((ch==LEFT)||(ch==RIGHT)) ) ) && ( !( (!funckey)&&( (ch==ESC)||(ch==ENTER) ) ) ) );
        puttext(x1+1,y1+1,x2+1,y2+1,ptr);                       //|
        free(ptr);
        return(ch);
}

void main_menu(void)
{
int i,j;
int end=0;
char_int ch;
        for(j=1;j<25;j++)
        for(i=0;i<80;i++) {
                *(vga+j*160+i*2+1)=0x1F;
                *(vga+j*160+i*2)=0;
        }
        for(i=2;i<24;i++) {
                *(vga+i*160)='|';
                *(vga+i*160+158)='|';
        }
        for(i=1;i<79;i++) {
                *(vga+i*2+160)='-';
                *(vga+i*2+3520)='-';
                *(vga+i*2+3840)='-';
        }
        for(i=0;i<80;i++)
                *(vga+i*2)=0;
        for(i=0;i<max_menu;i++) {
                gotoxy(menu_xpos[i],1);
                printf("%s",menu_text[i]);
        }
        for(i=0;i<80;i++)
                *(vga+i*2+1)=0x70;
        *(vga+160)='┌';
        *(vga+318)='┐';
        *(vga+3840)='└';
        *(vga+3998)='┘';
        *(vga+3520)='├';
        *(vga+3678)='┤';
        gotoxy(30,2);    cprintf(" Data Link by Radio Wave ");
        gotoxy(54,1);    cprintf("Delay : %d",delay_time);
        gotoxy(68,1);  cprintf("%s",fname);
        for(i=53;i<80;i++)
                *(vga+i*2+1) = 0x70;
```

```
rs232_initial();
for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
        *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x0F;
do
 {
        if(mode=='C')   rec_chat();
        if((!pull_act)||(p_menu[menu_pos].max==0)&&kbhit())
                ch.i=readfunc();
        else
                ch.i=pull_down(&p_menu[menu_pos]);
        if( funckey && kbpress )
         {
                 kbpress = No;
                 switch(ch.i) {
                case LEFT  :
                                        if(menu_on_off[0]==On)
                                         {
                                         for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x70;
                                         if(menu_pos>0)menu_pos--;
                                                else menu_pos=max_menu-1;
                                         for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x0F;
                                         break;
                                         }
                case RIGHT :
                                        if(menu_on_off[0]==On)
                                        {
                                        for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x70;
                                        if(menu_pos<max_menu-1)menu_pos++;
                                                else menu_pos=0;
                                        for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x0F;
                                        break;
                                        }
                        // Hot key checking
                case F3    : break;
                case F5    : send();break;
                case F6    : receive();break;
                case F10   : if(menu_on_off[0]==On)
                                        {
                                        menu_on_off[0] = Off;
                                        for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x70;
                                        }
                                else
                                        {
                                        menu_on_off[0] = On;
                                        for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x0F;
                                        }
                                        break;
                case ALT_X : end=1;
                        }
        } //   if( funckey && kbhit() )
        else
         if(!funckey&&kbpress)
         {
                 kbpress = No;
                 switch(ch.i)
                {
                        case ENTER :if(menu_on_off[0]==On)
                                        if(p_menu[menu_pos].max==0)
                                                pull_act=0;
                                        else
                                              {
                                               if(!pull_act)  pull_act=1;
                                                      else
                                                      {  //Execute Menu Command
                                                       switch(menu_pos)
                                                              {
                                                              case 0:
                                                               switch(p_menu[menu_pos].pos)
                                                                      {
                                                              case 0:f_open();      break;
                                                              case 1:drive();       break;
                                                              case 2:receive();     break;
                                                              case 3:send();        break;
                                                              case 4:dos_shell(); break;
```

```
                                                case 5:end = 1;
                                                ) break;
                                                case 1:rs232_initial();  break;
                                                case 2:
                                                 switch(p_menu[menu_pos].pos)
                                                        {
                                                                case 0:
                                                                                menu_on_off[0] =
                                                                                for(i=0;i<strlen
                                                                                        *(vga+(m
                                                                                chat_mode();
                                                                                break;
                                                                case 1:
                                                                                menu_on_off[0] =
                                                                                for(i=0;i<strlen
                                                                                        *(vga+(m
                                                                                file_transfer_mo
                                                                                break;
                                                        ) break;
                                                case 3:
                                                        switch(p_menu[menu_pos].pos)
                                                {
                                                 case 0:compress_util();  break;
                                                 case 1:expansion_util();    break;
                                                ) break;
                                                case 4:
                                                        switch(p_menu[menu_pos].pos)
                                                {
                                                 case 0:this_program();  break;
                                                 case 1:programmer();    break;
                                                 ) break;
                                                                ) //switch(menu_pos)
                                                        pull_act=0;
                                                                ) //  else if(pull_act)
                                                ) //  else if(p_menu[menu_pos].max==0)
                                        else
                                        {
                                         prepare_send_chat('\r');
                                         pos_x_send = 1;
                                         prepare_send_chat('\n');
                                         pos_x_send = 1;        pos_y_send++;
                                         if(pos_y_send==10)
                                           pos_y_send = 9;
                                         send_chat(str_send);
                                        }
                                         break;
                                case ESC : if(pull_act)pull_act=0;
                                           else
                                            {
                                                menu_on_off[0] = Off;
                                                for(i=0;i<strlen(menu_text[menu_pos])+2;i++)
                                                  *(vga+(menu_xpos[menu_pos]-2)*2+i*2+1)=0x70;
                                            } //end=1;
                                                break;
                                default : if((mode=='C')&&(menu_on_off[0]==Off))
                                                prepare_send_chat(ch.c[0]);  //send_chat(str);
                                          else printf("%c",7);
                                          break;
                        } //   switch(ch.i)
                } //   else
        } while(!end);
}

char *current_directory(char *path)
{
        strcpy(path, "X:\\");      /* fill string with form of response: X:\ */
        path[0] = 'A' + getdisk();    /* replace X with current drive letter */
        getcurdir(0, path+3);  /* fill rest of string with current directory */
        return(path);
}

void f_open()
{
char ch;
char curdir[MAXPATH];
void *ptr;
int x1,x2,y1,y2;
int i,j,out=1;
char *f;
char *temp_fname;
```

```
            x1=8;   x2=60;  y1=5;   y2=8;
            ptr=(void *)farmalloc(1024);
            gettext(x1+1,y1+1,x2+1,y2+1,ptr);
            border(x1,y1,x2,y2);
            current_directory(curdir);
            gotoxy(10,7); printf(" Current directory ==> %s", curdir);
            f = (char *) farmalloc(15);     f[0]=13;
            do {
                    gotoxy(10,8); printf(" Open file name : ");
                    for(i=x1;i<=x2;i++)
                    for(j=y1;j<=y2;j++)
                            *(vga+i*2+j*160+1)=0x70;
                    gotoxy(28,8);
                    window(28,8,59,8);
                    textbackground(LIGHTGRAY);
                    strncpy(ftemp,fname,12);
                    fname = cgets(f);
                    temp_fname = strrchr(fname,'.');
                    *(temp_fname+4) = 0;
                    if (f[1]==0)  {  strcpy(fname,ftemp);   out = 0;  }
                    else     out = 0;
            } while(out);
            window(68,1,80,1);
            clrscr();
            printf("%s",fname);
            for(i=66;i<80;i++)
                    *(vga+i*2+1)=0x70;
            window(1,1,80,25);
            puttext(x1+1,y1+1,x2+1,y2+1,ptr);
            free(ptr);
    }

    void drive()
    {
    int x1,x2,y1,y2;
    char p[31];
    char *path="                                ";
    int i,j,err,out=1;
    void *ptr;
            x1=8;   x2=60;  y1=6;   y2=8;   p[0]=30;
            ptr=(void*)farmalloc(1024);
            gettext(x1+1,y1+1,x2+1,y2+1,ptr);
            border(x1,y1,x2,y2);
            do {
                    gotoxy(10,8); printf(" Change directory to : ");
                    gotoxy(33,8);
                    window(33,8,59,8);
                    textbackground(LIGHTGRAY);
                    path = cgets(p);
                    if (p[1]==0)
                            out = 0;
                    else {
                            err = chdir(path);
                            if (err!=NULL) {
                                    window(x1+2,y1+2,x2,y2);
                                    clrscr();
                                    printf(" Path or file name not found ");
                                    delay(500);
                                    clrscr();
                                    window(1,1,80,25);
                            }
                            else out = 0;
                    }
            } while(out);
            window(1,1,80,25);
            puttext(x1+1,y1+1,x2+1,y2+1,ptr);
            free(ptr);
    }

    void send()
    {
    int x1,y1,x2,y2;
    int flag;
    void *ptr;
            if(mode=='F')
             {
                    ptr = (void *) farmalloc(1024);
                    x1 = 20;                x2 = 60;                y1 = 6;                 y2 = 8;
                    ptr = (void *) farmalloc(1024);
                    gettext(x1-1,y1-1,x2+1,y2+1,ptr);
```

```
                border(x1,y1,x2,y2);
                gotoxy(22,8);
                printf("       Please Wait while Sending    ");
                clrscr_attr(x1,y1,x2,y2);
                flag = send_file();
                border(x1,y1,x2,y2);
                gotoxy(22,8);
                if(flag==-1)
                        printf("      Can't open file to sending      ");
                else if(flag==-2)
                        printf("         Cancel to send file          ");
                else
                        printf("       File Sending already        ");
                clrscr_attr(x1,y1,x2,y2);
                delay(1000);
                puttext(x1-1,y1-1,x2+1,y2+1,ptr);
         }
        else printf("%c",7);
}

int send_file()
{
char_long f_size;   //      stand for file_size
int i,temp;
int x1,x2,y1,y2,center,size_x;
char ch;
char *out_name;
FILE *fptr;
        if(compression_switch==Off)
         {
                fptr = fopen(fname,"rb");
         }
        else
         {
                gotoxy(22,8);
                printf("    Please Wait while Compression   ");
                clrscr_attr(20,6,60,8);
                temp = compress_file();
                if(temp>=0)
                        {
                        delay(350);
                        gotoxy(22,8);
                        printf("       Compression   Already        ");
                        clrscr_attr(20,6,60,8);
                        delay(350);
                        gotoxy(22,8);
                        printf("       Please Wait while Sending    ");
                        clrscr_attr(20,6,60,8);
                        out_name = (char far *) farmalloc(20);
                        strcpy(out_name,fname);
                        for(temp=0;temp<strlen(fname);temp++)
                                if(*(fname+temp)==!.')   break;
                        *(out_name+temp+1)='$';
                        temp = strlen(fname);
                        *(out_name+temp)=0;
                        strcpy(fname,out_name);
                        window(68,1,80,1);
                        clrscr();
                        printf("%s",fname);
                        for(i=66;i<80;i++)
                                *(vga+i*2+1)=0x70;
                        window(1,1,80,25);
                        }
                else
                        {
                         delay(350);
                         gotoxy(22,8);
                         printf("         Can't  Compression         ");
                         clrscr_attr(20,6,60,8);delay(350);
                         gotoxy(22,8);
                         printf("     Sending without compression    ");
                         clrscr_attr(20,6,60,8);delay(350);
                         gotoxy(22,8);
                         printf("       Please Wait while Sending    ");
                         clrscr_attr(20,6,60,8);delay(350);
                        }
                fptr = fopen(fname,"rb");
         }
        if(fptr==NULL)  {  printf("%c",7); return -1;  }
        send_out('!');
```

```
        for(i=0;i<strlen(fname);i++)
                send_out(*(fname+i));
        send_out(0);
        f_size.l = file_size(fptr);
        send_out('@');
        delay(800);
        for(i=0;i<sizeof(long);i++)
         {
                send_out(f_size.ch[i]);
                delay(500);
         }
        send_out('#');          i = 0;
        while( (!kbhit()) && (i<f_size.l) )
                {
                if(i<f_size.l)
                 {
                        ch = (unsigned char) fgetc(fptr);
                        send_out(ch);
                        i = i + 1;
                 }
                }
        fclose(fptr);
        if(i==f_size.l)         return 0;
        else return -2;
}

void dos_shell()
{
char *drv;
void *ptr;
        ptr=(void *)farmalloc(4000);
        gettext(1,1,80,25,ptr);
        drv = (char *) farmalloc(MAXDIR+11);
        drv = (char *) getenv("COMSPEC");
        textbackground(0);
        textattr(15);
        clrscr();
        printf("Type 'Exit' to return\n");
        printf("Data Link by Radio Wave program");
        system((char far *)drv);
        puttext(1,1,80,25,ptr);
        free(ptr);
}

long file_size(FILE *fptr)
{
long i;
        fseek(fptr,0,SEEK_END);
        i = ftell(fptr);
        rewind(fptr);
        return i;
}

void expansion_util()
{
int x1,y1,x2,y2;
int flag;
void *ptr;
        x1 = 25;        x2 = 55;        y1 = 9; y2 = 14;
        menu_on_off[0]=Off;
        ptr=(void*)farmalloc(1024);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(x1+2,y1+2);      printf("        EXPANSION FILE       ");
        gotoxy(x1+2,y1+3);  printf("  Please wait for expansion  ");
        clrscr_attr(x1,y1,x2,y2);
        flag = expand_file();
        if(flag==-1)
        {
                gotoxy(x1+2,y2);
                printf("    Can't open input file    ");
                clrscr_attr(x1,y1,x2,y2);
        }
        else if(flag==-2)
        {
                gotoxy(x1+2,y2);
                printf("    Can't open output file   ");
                clrscr_attr(x1,y1,x2,y2);
        }
        else if(flag==-3)
```

```
        {
                gotoxy(x1+2,y2);
                printf("      Expansion Error       ");
                clrscr_attr(x1,y1,x2,y2);
        }
        else if(flag==0)
        {
                gotoxy(x1+2,y2);
                printf("      Expansion already     ");
                clrscr_attr(x1,y1,x2,y2);
        }
        getch();
        menu_on_off[0]=On;
        puttext(x1+1,y1+1,x2+1,y2+1,ptr);
        free(ptr);
}

int expand_file()
{
FILE *output;
BIT_FILE *input;
char far *out_name;
char ch;
int i,j;
        setbuf(stdout,NULL);
        input = OpenInputBitFile(fname);
        if(input==NULL)
                return -1;
        out_name = (char far *) farmalloc(13);
        for(i=0;i<13;i++)
                *(out_name+i) = NULL;
        for(i=0,j=0;i<12;i++)
         {
                ch = getc(input->file);
                if(ch!='*') { *(out_name+j) = ch;   j = j + 1;  }
         }
        output = fopen(out_name,"wb");
        if(output==NULL)  {     CloseInputBitFile(input);  farfree(out_name);  return -2; }
        i = ExpandFile(input,output);
        if(i<0)   {  farfree(out_name);    return -3;  }
        CloseInputBitFile(input);
        fclose(output);
        farfree(out_name);
        return( 0 );
}

void compress_util()
{
int x1,y1,x2,y2;
int flag;
void *ptr;
        x1=25;  x2=55;  y1=9;   y2=14;
        menu_on_off[0]=Off;
        ptr=(void*)farmalloc(1024);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(x1+2,y1+2);
        printf("     COMPRESSION FILE       ");
        clrscr_attr(x1,y1,x2,y2);
        gotoxy(x1+2,y1+3);
        printf(" Please wait for compression ");
        clrscr_attr(x1,y1,x2,y2);
        flag = compress_file();
        if(flag==-1) {
                gotoxy(x1+2,y2);
                printf("    Can't open input file    ");
                clrscr_attr(x1,y1,x2,y2);
        }
        else
                if(flag==-2) {
                        gotoxy(x1+2,y2);
                        printf("    Can't open output file   ");
                        clrscr_attr(x1,y1,x2,y2);
                }
                else
                        if(flag==-3) {
                                gotoxy(x1+2,y2);
                                printf("      Compression Error      ");
                                clrscr_attr(x1,y1,x2,y2);
                        }
```

```
                        else
                        if(flag==0) {
                                gotoxy(x1+2,y2);
                                printf("     Compression already     ");
                                clrscr_attr(x1,y1,x2,y2);
                        }
                getch();
                menu_on_off[0]=On;
                puttext(x1+1,y1+1,x2+1,y2+1,ptr);
                free(ptr);
        }

        int compress_file()
        {
        BIT_FILE far *output;
        FILE far *input;
        char far *out_name;
        int temp;
        int i;
        char *char_temp;
                setbuf( stdout, NULL );
                input = fopen((char *)fname,"rb");
                if(input==NULL)  return -1;
                out_name = (char *) farmalloc(20);
                _fstrcpy((char *) out_name,(char *)fname);
                for(temp=0;temp<_fstrlen((char *)fname);temp++)
                        if(*(fname+temp)=='.')   break;
                *(out_name+temp+1)='$';
                temp = _fstrlen((char *)fname);
                *(out_name+temp)=0;
                output = OpenOutputBitFile((char *)out_name);
                if(output==NULL) {   farfree(out_name);  fclose(input);   return -2;   }
                char_temp = (char *) farmalloc(12);
                for(i=0;i<12;i++)   *(char_temp+i) = 0;
                if(temp<=12)
                 {
                        for(i=0;i<12-temp;i++)   *(char_temp+i) = '*';
                        strcat(char_temp,(char *)fname);
                 }
                fprintf(output->file,"%s",char_temp);
                temp = CompressFile(input,output);
                if(temp<0)  temp = -3;
                temp = CloseOutputBitFile( output );
                if(temp<0)  temp = -3;
                fclose(input);
                farfree(out_name);
                farfree(char_temp);
                return temp;
        }

        int CompressFile(FILE *input,BIT_FILE *output)
        {
        unsigned long *counts;
        NODE *nodes;
        CODE *codes;
        int root_node;
        int flag;
                counts = ( unsigned long *)     calloc( 256, sizeof( unsigned long ) );
                if ( counts == NULL )           return -1;
                if ( (nodes = (NODE *)  calloc( 514, sizeof( NODE ) ) ) == NULL )
                return -1;
                if ( ( codes = (CODE *)         calloc( 257, sizeof( CODE ) ) ) == NULL )
                {  free(nodes);   free(counts);  return -1;  }
                f_size = file_size(input);
                count_bytes( input, counts );
                scale_counts( counts, nodes );
                flag =  output_counts( output, nodes );
                if(flag<0)
                 {  free( (char *) counts );            free( (char *) nodes );
                        free( (char *) codes );           return -1;   }
                root_node = build_tree( nodes );
                convert_tree_to_code( nodes, codes, 0, 0, root_node );
                flag = compress_data( input, output, codes );
                if(flag<0){  free( (char *) counts );
                free( (char *) nodes ); free( (char *) codes );   return -1;  }
                free( (char *) counts );
                free( (char *) nodes );
                free( (char *) codes );
                return 0;
        }
```

```
BIT_FILE *OpenOutputBitFile(char *name )
{
BIT_FILE *bit_file;
        bit_file = (BIT_FILE *) calloc( 1, sizeof( BIT_FILE ) );
        if ( bit_file == NULL )
        return( bit_file );
        bit_file->file = fopen( name, "wb" );
        bit_file->rack = 0;
        bit_file->mask = 0x80;
        bit_file->pacifier_counter = 0;
        return( bit_file );
}

BIT_FILE *OpenInputBitFile(char *name )
{
BIT_FILE far *bit_file;
        bit_file = (BIT_FILE far *) farmalloc( sizeof( BIT_FILE ) );
        if ( bit_file == NULL )     return( bit_file );
        bit_file->file = (FILE far *) farmalloc(sizeof(FILE));
        bit_file->file = fopen(name,"rb");
        bit_file->rack = 0;
        bit_file->mask = 0x80;
        bit_file->pacifier_counter = 0;
        return( bit_file );
}

int CloseOutputBitFile(BIT_FILE *bit_file )
{
        if( bit_file->mask !=0x80 )
                if( putc( bit_file->rack, bit_file->file ) != bit_file->rack )
                 return -1;
        fclose( bit_file->file );
        free( ( char * ) bit_file);
        return 0;
}

int OutputBit(BIT_FILE *bit_file,int bit )
{
        if (bit )       bit_file->rack  |= bit_file->mask;
        bit_file->mask >>= 1;
        if (bit_file->mask == 0 )
                {
                 if ( putc( bit_file->rack, bit_file->file ) != bit_file->rack)
                  return -1;
                bit_file->rack = 0;
                bit_file->mask = 0x80;
        }
        return 0;
}

int  OutputBits(BIT_FILE *bit_file,unsigned long code,int count )
{
unsigned long mask;
        mask = 1L << (count - 1 );
        while ( mask != 0 )
                {
                if ( mask & code )              bit_file->rack |= bit_file->mask;
                bit_file->mask >>= 1;
                if ( bit_file->mask == 0)
                 {
                        if (putc (bit_file->rack, bit_file->file ) != bit_file->rack)
                                return -1;
                        bit_file->rack = 0;
                        bit_file->mask = 0x80;
                 }
                mask >>=  1;
                }
        return 0;
}

void FilePrintBinary(FILE *file,unsigned int code,int bits)
{
unsigned int mask;
        mask = 1 << (bits - 1 );
        while ( mask !=0 )
                {
                if ( code & mask )
                        fputc( '1', file );
                else
```

```
                        fputc( '0', file );
                mask >>= 1;
                }
}

int  output_counts(BIT_FILE *output,NODE *nodes )
{
int first;
int last;
int next=0;
int i;
        first = 0;
        while ( first < 255 && nodes[ first ].count == 0 )
                first++;
        for ( ; first < 256 ; first = next )
                {
                last = first + 1;
                for ( ; ; )
                        {
                        for ( ; last < 256 ; last++ )
                                if ( nodes[ last ].count == 0 )  break;
                        last--;
                        for ( next = last + 1; next < 256 ; next++ )
                                if ( nodes[ next ].count != 0 )    break;
                        if ( next > 255 )                          break;
                        if ( ( next - last ) > 3 )          break;
                        last = next;
                }
                if (putc( first, output->file ) != first )  return -1;
                if (putc( last, output->file ) != last )          return -1;
                for ( i = first ; i <= last ; i++ )
                {
                        if (putc( nodes[ i ].count, output->file ) != (int) nodes[ i ].count )
                                return -1;
                }
        }
        if ( putc( 0, output->file ) != 0 )     return -1;
        return 0;
}

void count_bytes(FILE *input,unsigned long *counts )
{
long input_marker;
int c;
long i;
        input_marker = ftell( input );
        i = 0;
        while(i<f_size)
        {
                c = getc(input);
                counts[ c ]++;
                i = i + 1;
        }
        fseek( input, input_marker, SEEK_SET);
}

void scale_counts(unsigned long *counts,NODE *nodes )
{
unsigned long max_count;
int i;
        max_count = 0;
        for ( i = 0 ; i < 256 ; i++ )
                if ( counts[ i ] > max_count )
                        max_count = counts[ i ];
        if ( max_count == 0 )
                {
                counts[ 0 ] = 1;
                max_count = 1;
                }
        max_count = max_count / 255;
        max_count = max_count + 1;
        for ( i = 0 ; i < 256 ; i++ )
                {
                nodes[ i ].count = (unsigned int )  ( counts[ i ] / max_count );
                if ( nodes[ i ].count == 0 && counts[ i ] != 0 )
                        nodes[ i ].count = 1;
                }
        nodes[ END_OF_STREAM ].count = 1;
}
```

```
int build_tree(NODE *nodes )
{
int next_free;
int i;
int min_1;
int min_2;
        nodes[ 513 ].count = 0xffff;
        for ( next_free = END_OF_STREAM +1 ; ; next_free++ )
                {
                min_1 = 513;
                min_2 = 513;
                for ( i = 0 ; i < next_free ; i++ )
                        if ( nodes[ i ].count != 0)
                        {
                         if ( nodes[ i ].count < nodes[ min_1 ].count )
                                {
                                 min_2 = min_1;
                                 min_1 = i;
                                }
                                else if ( nodes[ i ].count< nodes[ min_2 ].count )
                                                        min_2 = i;
                        }
                if ( min_2 == 513 )                             break;

                nodes[ next_free ].count = nodes[ min_1 ].count+nodes[ min_2 ].count;
                nodes[ min_1 ].saved_count = nodes[ min_1 ].count;
                nodes[ min_1 ].count = 0;
                nodes[ min_2 ].saved_count = nodes[ min_2 ].count;
                nodes[ min_2 ].count = 0;
                nodes[ next_free ].child_0 = min_1;
                nodes[ next_free ].child_1 = min_2;
        }
        next_free--;
        nodes[ next_free ].saved_count = nodes[ next_free ].count;
        return( next_free );
}

void convert_tree_to_code(NODE *nodes,CODE *codes,unsigned int code_so_far,int  bits,int node )
{
        if ( node <= END_OF_STREAM )
                {
                codes[ node ].code = code_so_far;
                codes[ node ].code_bits = bits;
                return;
                }
        code_so_far <<= 1;
        bits++;
        convert_tree_to_code( nodes, codes, code_so_far, bits,nodes[ node ].child_0 );
        convert_tree_to_code( nodes,codes,code_so_far | 1,bits,nodes[ node ].child_1);
}

void print_char(int c,FILE *fp )
{
        if ( c >= 0x20 && c < 127 )             fprintf( fp,"'%c'",c );
        else            fprintf( fp,"%3d",c );
}

int  compress_data(FILE *input,BIT_FILE *output,CODE *codes )
{
int c,flag;
long i;
        i = 0;
        while(i<f_size)
         {
         c = getc(input);
                i = i + 1;
                flag = OutputBits( output, (unsigned long) codes[ c ].code,codes[ c ].code_bits );
                if(flag<0)  return -1;
         }
        flag = OutputBits( output, (unsigned long) codes[ END_OF_STREAM ].code,codes[ END_OF_STREAM ].code_bits );
        if(flag<0) return -1;
        return 0;
}

void CloseInputBitFile(BIT_FILE *bit_file )
{
    fclose( bit_file->file );
    free( (char*) bit_file );
}
```

```
int InputBit(BIT_FILE *bit_file )
{
int value;
        if ( bit_file->mask == 0x80 )
                bit_file->rack = getc( bit_file->file );
        value = bit_file->rack & bit_file->mask;
        bit_file->mask >>= 1;
        if ( bit_file->mask == 0 )              bit_file->mask = 0x80;
        return( value ? 1:0 );
}

unsigned long InputBits(BIT_FILE *bit_file,int bit_count )
{
unsigned long mask;
unsigned long return_value;
        mask = 1L << ( bit_count - 1 );
        return_value = 0;
        while ( mask != 0 )
                {
                if ( bit_file->mask == 0x80 )
                    bit_file->rack = getc( bit_file->file );
                if ( bit_file->rack & bit_file->mask )                  return_value = mask;
                mask >>= 1;
                if ( bit_file->mask == 0 )              bit_file->mask = 0x80;
        }
        return(return_value);
}

int ExpandFile(BIT_FILE *input,FILE *output)
{
NODE *nodes;
int root_node;
int flag;
        flag = 0;
        if ( ( nodes = (NODE *)calloc( 514, sizeof( NODE ) ) ) == NULL )
                return -1;
        input_counts( input, nodes );
        root_node = build_tree( nodes );
        flag = expand_data( input, output,nodes, root_node );
        free( (char * ) nodes );
        return flag;
}

void input_counts(BIT_FILE *input,NODE *nodes )
{
int first;
int last;
int i;
int c;
        for ( i = 0 ; i < 256 ; i++ )
                nodes[ i ].count = 0;
        first = getc(input->file);
        last  = getc(input->file);
        for ( ; ; )
                {
                for ( i = first ; i <= last ; i++ )
                        {
                                c = getc(input->file);
                                nodes[ i ].count = ( unsigned int ) c;
                        }
                        first = getc(input->file);
                        if ( first == 0 )                                       break;
                        last = getc(input->file);
                }
        nodes[ END_OF_STREAM ].count = 1;
}

int  expand_data(BIT_FILE *input,FILE *output,NODE *nodes,int root_node )
{
int node;
        for ( ; ; )
         {
                node = root_node;
                do
                 {
                        if (InputBit( input ) )         node = nodes[ node ].child_1;
                                else                    node = nodes[ node ].child_0;
                 } while ( node > END_OF_STREAM );
                if ( node == END_OF_STREAM )    break;
                if ( ( putc( node, output ) ) != node )
```

```
                        return -1;
                }
        return 0;
}

void set_delay_time()
{
int x1,x2,y1,y2;
int dt;
char p[6];
int i,j;
void *ptr;
        x1=23;  x2=47;  y1=6;   y2=8;   p[0]=5;
        ptr=(void*)farmalloc(1024);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(x1+2,y2); printf(" Set delay time : ");
        textbackground(LIGHTGRAY);
        window(x2-4,y1+2,x2,y2);
        dt = atoi(cgets(p));
        if (p[1]==0)
                delay_time = 0;
        else
                delay_time = dt;
        window(1,1,80,25);
        window(62,1,65,1);
        clrscr();
        window(1,1,80,25);
        gotoxy(54,1);   printf("Delay : %d",delay_time);
        for(i=53;i<66;i++)
                *(vga+i*2+1)=0x70;
        puttext(x1+1,y1+1,x2+1,y2+1,ptr);
        free(ptr);
}

void this_program()
{
void *ptr;
int x1,x2,y1,y2;
        x1=27;  x2=53;  y1=10;  y2=15;
        ptr=(void*)farmalloc(1024);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(29,12); printf("       PROGRAM          ");
        gotoxy(29,14); printf(" DATA LINK BY RADIO WAVE ");
        gotoxy(29,15); printf("      VERSION 1.00      ");
        clrscr_attr(x1,y1,x2,y2);
        getch();
        puttext(x1+1,y1+1,x2+1,y2+1,ptr);
        free(ptr);
}

void programmer()
{
void *ptr;
int x1,x2,y1,y2;
        x1=21;  x2=56;  y1=10;  y2=14;
        ptr=(void*)farmalloc(1024);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(23,12); printf("Advisor  Mr.Kriangkrai Vongrojporn");
        gotoxy(23,14); printf("Programmer  Prapan Chantavutsettee");
        clrscr_attr(x1,y1,x2,y2);
        getch();
        puttext(x1+1,y1+1,x2+1,y2+1,ptr);
        free(ptr);
}

void clrscr_attr(int x1,int y1,int x2,int y2)
{
int i,j;
        for(i=x1;i<=x2;i++)
        for(j=y1;j<=y2;j++)
                *(vga+i*2+j*160+1)=0x70;
}

void border(int x1,int y1,int x2,int y2)
{
int i,j;
        for(i=x1;i<=x2;i++)
```

```
            for(j=y1;j<=y2;j++)
             {
                    *(vga+i*2+j*160+1)=0x70;
                    *(vga+i*2+j*160)=0;
             }

            for(i=x1;i<=x2;i++) {
                    *(vga+i*2+y1*160+1)=0x70;
                    *(vga+i*2+y2*160+1)=0x70;
                    *(vga+i*2+y1*160)='-';
                    *(vga+i*2+y2*160)='-';
            }
            for(i=y1;i<=y2;i++) {
                    *(vga+x1*2+i*160+1)=0x70;
                    *(vga+x2*2+i*160+1)=0x70;
                    *(vga+x1*2+i*160)='|';
                    *(vga+x2*2+i*160)='|';
            }
            *(vga+y1*160+x1*2)='┌';
            *(vga+y1*160+x2*2)='┐';
            *(vga+y2*160+x1*2)='└';
            *(vga+y2*160+x2*2)='┘';
    }

    void chat_mode()
    {
    void *ptr;
    int x1,y1,x2,y2;
    char ch;
            x1 = 20;        x2 = 60;        y1 = 6; y2 = 8;
            ptr = (void *) farmalloc(1024);
            gettext(x1-1,y1-1,x2+1,y2+1,ptr);
            border(x1,y1,x2,y2);
            gotoxy(22,8);
            if(mode=='F') {  printf("Are you sure to change to Chat Mode : ");  clrscr_attr(x1,y1,x2,y2);  }
            else {    printf("       Now you are in Chat Mode      ");  clrscr_attr(x1,y1,x2,y2);    getch(); }
            if(mode=='F')
             {
                   while(1)
                    {
                            ch = getch();
                            if(ch=='n'||ch=='y'||ch=='N'||ch=='Y')
                            break;
                            else printf("%c",7);
                    }
                   printf("%c",ch);
                   if(ch=='Y'||ch=='y')  {  mode = 'C';  initial_chat_mode();  }
                   }
           puttext(x1-1,y1-1,x2+1,y2+1,ptr);
           free(ptr);
}

 void initial_chat_mode()
 {
 int i,j;
           str_send = (char *) farmalloc(80);
           pos_x_send = pos_y_send = 1;
           pos_x_rec  = pos_y_rec  = 1;
           textcolor(WHITE);
           for(i=1;i<79;i++)
                   for(j=2;j<22;j++)
                           *(vga+2*i+j*160) = 0;
           for(i=2;i<80;i++)
            {
                   gotoxy(i,13);
                   cprintf("-");
            }
           gotoxy(1,13);   cprintf("|");
           gotoxy(80,13);  cprintf("|");
           gotoxy(38,12);  cprintf("SEND");
           gotoxy(37,14);  cprintf("RECEIVE");
           for(j=0;j<3;j++)
            for(i=0;i<80;i++)
                   *(vga+2*i+(11+j)*80*2+1) = 0x1F;
}

void prepare_send_chat(char ch)
{
       window(1,1,80,25);
       window(2,3,79,11);
```

```
            gotoxy(pos_x_send,pos_y_send);
            if( (ch!='\n') && (ch!='\r') )
             {
                    *(str_send+pos_x_send-1) = ch;
                    *(str_send+pos_x_send)   = 0;
             }
            if(ch=='\b')
             {
                     if(pos_x_send>1)
                        {
                     *(str_send+pos_x_send-2) = 0;
                     ch = 32;
                     pos_x_send--;
                     gotoxy(pos_x_send,pos_y_send);
                     cprintf("%c",ch);
                     gotoxy(pos_x_send,pos_y_send);
                           }
                     else
                           printf("%c",7);
             }
            else {  pos_x_send++;  cprintf("%c",ch);  }
            if(pos_x_send==79) { pos_x_send = 1; pos_y_send++; }
            *(vga+2*(pos_x_send-1+2)+160*(pos_y_send-1+3)+1) = 0x1F;
    }

    void send_chat(char *str)
    {
    int i;
            for(i=0;i<strlen(str);i++)
                    send_out(*(str+i));
            send_out('\n');
    }

    void send_out(unsigned char ch)
    {
            while(1)
                   if(check_out())
                    {
                    outp(Port_number,ch);
                    delay(delay_time);
                    break;
                    }
    }

    int check_out()
    {
           return inportb(Port_number+5) & 0x20 ;
    }

    int check_in()
    {
           return 1 & inportb(Port_number+5);
    }

    unsigned char rec_in()
    {
    unsigned char ch;
            ch = inportb(Port_number);
            return ch;
    }

    void rec_chat()
    {
    char ch;
            if(check_in())
                  {
                   ch = rec_in();
                   window(1,1,80,25);
                   window(2,15,79,22);
                   if( (ch=='\n')||(ch=='\r') )
                         {
                                 gotoxy(pos_x_rec,pos_y_rec);
                                 cprintf("\n\r");
                                 pos_x_rec = 1;
                                 if(pos_y_rec == 8)  pos_y_rec = 7;
                                 else pos_y_rec++;
                         }
                   else  {
                   gotoxy(pos_x_rec,pos_y_rec);
```

```
                cprintf("%c",ch);
                pos_x_rec++;
                if(pos_x_rec==78)
                        {   pos_y_rec++;   pos_x_rec = 1;   }
                if(pos_y_rec==9)
                        {   pos_x_rec = 1; pos_y_rec = 7;   }
                      }
                window(1,1,80,25);
               }
}

void file_transfer_mode()
{
void *ptr;
int x1,y1,x2,y2;
char ch;
        x1 = 14;        x2 = 66;        y1 = 6; y2 = 8;
        ptr = (void *) farmalloc(1024);
        window(1,1,80,25);
        gettext(x1-1,y1-1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(17,8);
        if(mode=='C') { printf("Are you sure to change to File Transfer  Mode : ");  clrscr_attr(x1,y1,x2,y2);  }
        else {              printf("       Now you are in File Transfer Mode        ");  clrscr_attr(x1,y1,x2,y2);
        if(mode=='C')
         {
                while(1)
                 {
                        ch = getch();
                        if(ch=='n'||ch=='y'||ch=='N'||ch=='Y')                  break;
                        else printf("%c",7);
                 }
                printf("%c",ch);
                if(ch=='Y'||ch=='y')  {  mode = 'F';  initial_file_mode();  }
                }
       puttext(x1-1,y1-1,x2+1,y2+1,ptr);
       free(ptr);
}

void initial_file_mode()
{
int i,j;
       for(i=1;i<79;i++)
              for(j=2;j<22;j++)
                     *(vga+2*i+j*160) = 0;
       free(str_send);
}

void save_config()
{
int x1,x2,y1,y2;
char ch;
void *ptr;
int  i,flag,size_str;
char *file_name;
FILE *fptr;
       x1 = 20;        x2 = 60;        y1 = 6; y2 = 8;
       ptr = (void *) farmalloc(1024);
       gettext(x1-1,y1-1,x2+1,y2+1,ptr);
       border(x1,y1,x2,y2);
       gotoxy(22,8);
       printf("    Are you sure to save config :  ");
       clrscr_attr(x1,y1,x2,y2);
       while(1)
              {
                     ch = getch();
                     ch = toupper(ch);
                     if(ch=='Y') { flag = 1;  break; }
                     else if(ch=='N') { flag = 0;break; }
                     else printf("%c",7);
              }
       if(flag==1)
              {
              file_name = (char *) farmalloc(100);
              strcpy((char *)file_name,(char *)Program);
              size_str = strlen((char *)file_name);
              *(file_name+size_str-3) = 'C';
              *(file_name+size_str-2) = 'F';
              *(file_name+size_str-1) = 0;
              flag = 0;
```

```
                fptr = fopen((char *)file_name,"w");
                if(fptr!=NULL)
                        {
                        flag = 1;
                        fprintf(fptr,"Baud=%d\n",set_menu[0].pos);
                        fprintf(fptr,"Parity=%d\n",set_menu[1].pos);
                        fprintf(fptr,"Data=%d\n",set_menu[2].pos);
                        fprintf(fptr,"Stop=%d\n",set_menu[3].pos);
                        fprintf(fptr,"Port=%d\n",set_menu[4].pos);
                        fprintf(fptr,"Delay=%d\n",delay_time);
                        fprintf(fptr,"Compress=%d\n",set_menu[6].pos);
                        fprintf(fptr,"Number User Port=%d\n",no_user_port);
                        for(i=0;i<no_user_port;i++)
                                fprintf(fptr,"Address of User Port [%d]=%d\n",i,no_user_addr[i]);
                        fclose(fptr);
                 }
                else
                        {
                                border(x1,y1,x2,y2);
                                gotoxy(22,8);
                                printf("     Can't save configuration      ");
                        }
                }
         puttext(x1-1,y1-1,x2+1,y2+1,ptr);
         free(ptr);
}

void receive()
{
int x1,y1,x2,y2;
int flag;
void *ptr;
        if(mode=='F')
         {
                ptr = (void *) farmalloc(1024);
                x1 = 20;                x2 = 60;                y1 = 6;                 y2 = 8;
                ptr = (void *) farmalloc(1024);
                gettext(x1-1,y1-1,x2+1,y2+1,ptr);
                border(x1,y1,x2,y2);
                gotoxy(22,8);
                printf("     Please Wait while Receiving   ");
                clrscr_attr(x1,y1,x2,y2);
                delay(800);
                gotoxy(22,8);
                printf("       Press any key to cancel     ");
                clrscr_attr(x1,y1,x2,y2);
                flag = rec_file();
                border(x1,y1,x2,y2);
                gotoxy(22,8);
                if(flag==-1)
                        printf("    Can't open file to receiving   ");
                if(flag==-2)
                        printf("       Cancel to receive file      ");
                else
                        printf("        File Receiving already     ");
                clrscr_attr(x1,y1,x2,y2);
                delay(1000);
                puttext(x1-1,y1-1,x2+1,y2+1,ptr);
                free(ptr);
         }
        else printf("%c",7);
}

int rec_file()
{
char_long size;
char  *name;
int special_flag;
int count,limit,i;
unsigned char ch;
FILE *fptr;
        count = 0;  special_flag = 0;
        name = (char *) farmalloc(15);
        while(!kbhit())
         {
                if(check_in())
                {
                        ch = rec_in();
                        if(special_flag == 1)
                                {
```

```
                                *(name+count) = ch;
                                if(*(name+count) == 0)  break;
                                count = count + 1;
                                }
                        if(ch == '!')  special_flag = 1;
                }
        }
        if(kbhit()) { free(name);   return -2;  }
        strcpy(fname,name);
        window(1,1,80,25);
        window(68,1,80,1);
        clrscr();
        printf("%s",fname);
        for(i=66;i<80;i++)
                *(vga+i*2+1)=0x70;
        window(1,1,80,25);
        fptr = fopen(fname,"wb");
        if(fptr==NULL)  { free(name);  return -1;  }
        limit = sizeof(long);
        count = 0;    special_flag = 0;
        while(!kbhit())
                if(check_in())
                {
                        ch = rec_in();
                        if(special_flag==1)
                        {
                                size.ch[count] = ch;
                                count = count + 1;
                                if(count==limit)  break;
                        }
                        if(ch=='@')   special_flag = 1;
                }
        if(kbhit())  { free(name);   return -2;  }
        count = 0;    special_flag = 0;
        while(!kbhit())
                if(check_in())
                {
                        ch = rec_in();
                        if(special_flag==1)
                        {
                                fprintf(fptr,"%c",ch);
                                count = count + 1;
                                if(count==size.l)       break;
                        }
                        if(ch=='#')   special_flag = 1;
                }
        free(name);
        fclose(fptr);
        if(count!=size.l)   return -2;
        return 0;
}

void set_user_port()
{
int x1,x2,y1,y2;
int i;
char ch,*str;
void *ptr;
        x1=23;  x2=60;  y1=6;   y2=8;
        ptr=(void*)farmalloc(1024);
        gettext(x1+1,y1+1,x2+1,y2+1,ptr);
        border(x1,y1,x2,y2);
        gotoxy(x1+2,y2); printf(" Enter number of User Port : ");
        clrscr_attr(x1,y1,x2,y2);
        str = (char *) malloc(5);
        while(1)
        {
                ch = getch();
                if((ch>47) && (ch<51)) { cprintf("%c",ch);       break;  }
                else printf("%c",7);
        }
        no_user_port = ch-48;
        set_menu[4].max=2+no_user_port;
        if(no_user_port>0) set_menu[4].pos=0;
        for(i=0;i<no_user_port;i++)
                {
                        border(x1,y1,x2,y2);
                        gotoxy(x1+2,y2);
                        printf(" Enter Address of User Port %d : ",i);
                        clrscr_attr(x1,y1,x2,y2);
```

```
                    gets(str);
                    no_user_addr[i] = atoi(str);
                }
        puttext(x1+1,y1+1,x2+1,y2+1,ptr);
        free(ptr);
        free(str);
}
```

EXAR

XR-2206

Monolithic Function Generator

GENERAL DESCRIPTION

The XR-2206 is a monolithic function generator integrated circuit capable of producing high quality sine, square, triangle, ramp, and pulse waveforms of high-stability and accuracy. The output waveforms can be both amplitude and frequency modulated by an external voltage. Frequency of operation can be selected externally over a range of 0.01 Hz to more than 1 MHz.

The circuit is ideally suited for communications, instrumentation, and function generator applications requiring sinusoidal tone, AM, FM, or FSK generation. It has a typical drift specification of 20 ppm/°C. The oscillator frequency can be linearly swept over a 2000:1 frequency range, with an external control voltage, having a very small affect on distortion.

FEATURES

Low-Sine Wave Distortion	.5%, Typical
Excellent Temperature Stability	20 ppm/°C, Typical
Wide Sweep Range	2000:1, Typical
Low-Supply Sensitivity	0.01%/V, Typical
Linear Amplitude Modulation	
TTL Compatible FSK Controls	
Wide Supply Range	10V to 26V
Adjustable Duty Cycle	1% to 99%

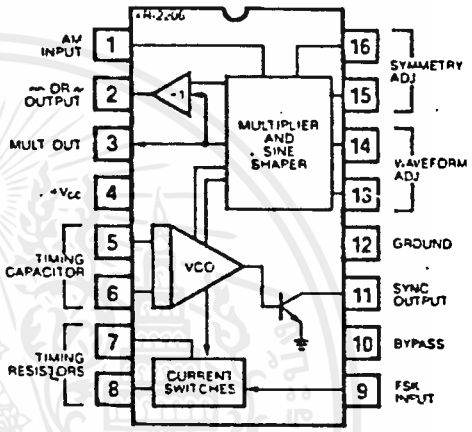
APPLICATIONS

Waveform Generation
Sweep Generation
AM/FM Generation
V/F Conversion
FSK Generation
Phase-Locked Loops (VCO)

ABSOLUTE MAXIMUM RATINGS

Power Supply	26V
Power Dissipation	750 mW
Derate Above 25°C	5 mW/°C
Total Timing Current	6 mA
Storage Temperature	-65°C to +150°C

FUNCTIONAL BLOCK DIAGRAM



ORDERING INFORMATION

Part Number	Package	Operating Temperature
XR-2206M	Ceramic	-55°C to +125°C
XR-2206N	Ceramic	0°C to +70°C
XR-2206P	Plastic	0°C to +70°C
XR-2206CN	Ceramic	0°C to +70°C
XR-2206CP	Plastic	0°C to +70°C

SYSTEM DESCRIPTION

The XR-2206 is comprised of four functional blocks, a voltage-controlled oscillator (VCO), an analog multiplier and sine-shaper; a unity gain buffer amplifier; and a set of current switches.

The VCO actually produces an output frequency porportional to an input current, which is produced by a resistor from the timing terminals to ground. The current switches route one of the timing pins current to the VCO controlled by an FSK input pin, to produce an output frequency. With two timing pins, two discrete output frequencies can be independently produced for FSK Generation Applications.

XR-2206

ELECTRICAL CHARACTERISTICS

Test Conditions: Test Circuit of Figure 1, $V^+ = 12V$, $T_A = 25°$, $C = 0.01\ \mu F$, $R_1 = 100\ k\Omega$, $R_2 = 10\ k\Omega$, $R_3 = 25\ k\Omega$ unless otherwise specified. S_1 open for triangle, closed for sine wave.

PARAMETER	XR-2206M			XR-2206C			UNIT	CONDITIONS
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.		
GENERAL CHARACTERISTCS								
Single Supply Voltage	10		26	10		26	V	
Split-Supply Voltage	±5		±13	±5		±13	V	
Supply Current		12	17		14	20	mA	$R_1 \geq 10\ k\Omega$
OSCILLATOR SECTION								
Max. Operating Frequency	0.5	1		0.5	1		MHz	$C = 1000\ pF$, $R_1 = 1\ k\Omega$
Lowest Practical Frequency		0.01			0.01		Hz	$C = 50\ \mu F$, $R_1 = 2\ M\Omega$
Frequency Accuracy		±1	±4		±2		% of f_o	$f_o = 1/R_1C$
Temperature Stability		±10	±50		±20		ppm/°C	$0°C \leq T_A \leq 75°C$,
Supply Sensitivity		0.01	0.1		0.01		%/V	$R_1 = R_2 = 20\ k\Omega$
Sweep Range	1000:1	2000:1			2000:1		$f_H = f_L$	$V_{LOW} = 10V$, $V_{HIGH} = 20V$, $R_1 = R_2 = 20\ k\Omega$
Sweep Linearity								f_H @ $R_1 = 1\ k\Omega$
10:1 Sweep		2			2		%	f_L @ $R_1 = 2\ M\Omega$
1000:1 Sweep		8			8		%	$f_L = 1\ kHz$, $f_H = 10\ kHz$
FM Distortion		0.1			0.1		%	$f_L = 100\ Hz$, $f_H = 100\ kHz$
Recommended Timing Components								±10% Deviation
Timing Capacitor: C	0.001		100	0.001		100	μF	See Figure 4.
Timing Resistors: R_1 & R_2	1		2000	1		2000	kΩ	
Triangle Sine Wave Output								
Triangle Amplitude		160			160		mV/kΩ	See Note 1, Figure 2.
Sine Wave Amplitude	40	60	80		60		mV/kΩ	Figure 1, S_1 Open
Max. Output Swing		6			6		Vp-p	Figure 1, S_1 Closed
Output Impedance		600			600		Ω	
Triangle Linearity		1			1		%	
Amplitude Stability		0.5			0.5		dB	
Sine Wave Amplitude Stability		4800			4800		ppm/°C	For 1000:1 Sweep
Sine Wave Distortion								See Note 2.
Without Adjustment		2.5			2.5		%	
With Adjustment		0.4	1.0		0.5	1.5	%	$R_1 = 30\ k\Omega$
Amplitude Modulation								See Figures 6 and 7.
Input Impedance	50	100		50	100		kΩ	
Modulation Range		100			100		%	
Carrier Suppression		55			55		dB	
Linearity		2			2		%	For 95% modulation
Square-Wave Output								
Amplitude		12			12		Vp-p	Measured at Pin 11.
Rise Time		250			250		nsec	$C_L = 10\ pF$
Fall Time		50			50		nsec	$C_L = 10\ pF$
Saturation Voltage		0.2	0.4		0.2	0.6	V	$I_L = 2\ mA$
Leakage Current		0.1	20		0.1	100	μA	$V_{11} = 26V$
FSK Keying Level (Pin 9)	0.8	1.4	2.4	0.8	1.4	2.4	V	See section on circuit controls
Reference Bypass Voltage	2.9	3.1	3.3	2.5	3	3.5	V	Measured at Pin 10.

Note 1: Output amplitude is directly proportional to the resistance, R_3, on Pin 3. See Figure 2

Note 2: For maximum amplitude stability, R_3 should be a positive temperature coefficient resistor.

XR-2206

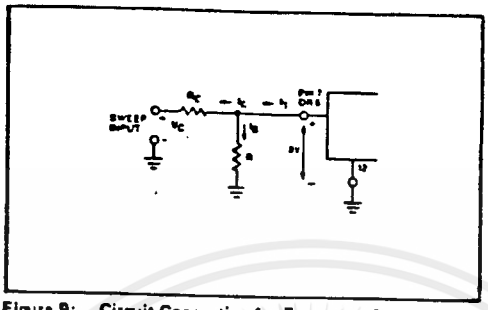


Figure 9: Circuit Connection for Frequency Sweep.

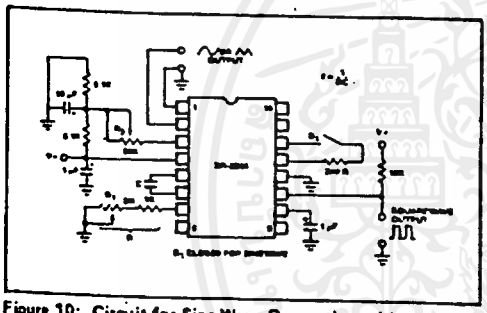


Figure 10: Circuit for Sine Wave Generation without External Adjustment. (See Figure 2 for Choice of R_3.)

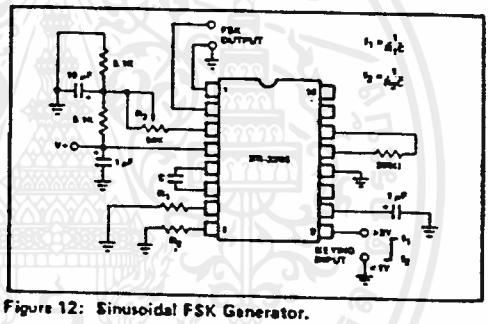


Figure 12: Sinusoidal FSK Generator.

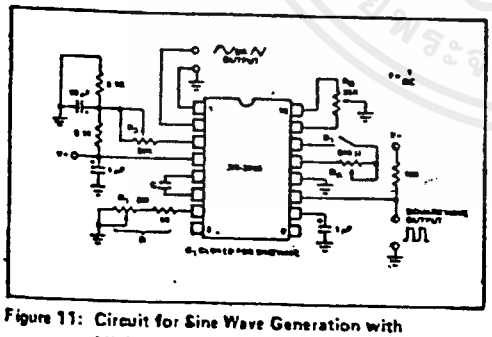


Figure 11: Circuit for Sine Wave Generation with Minimum Harmonic Distortion. (R_3 Determines Output Swing – See Figure 2.)

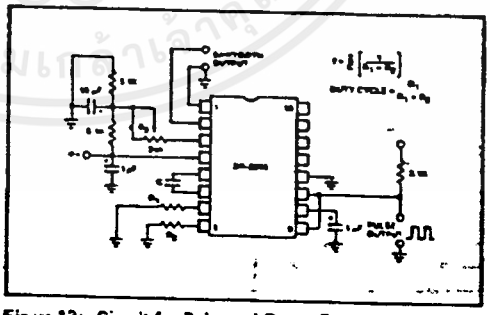


Figure 13: Circuit for Pulse and Ramp Generation.

Frequency-Shift Keying:

The XR-2206 can be operated with two separate timing resistors, R_1 and R_2, connected to the timing Pin 7 and 8, respectively, as shown in Figure 12. Depending on the polarity of the logic signal at Pin 9, either one or the other of these timing resistors is activated. If Pin 9 is open-circuited or connected to a bias voltage $\geq$2V, only R_1 is activated. Similarly, if the voltage level at Pin 9 is $\leq$1V, only R_2 is activated. Thus, the output frequency can be keyed between two levels, f_1 and f_2, as:

$$f_1 = 1/R_1C \text{ and } f_2 = 1/R_2C$$

For split-supply operation, the keying voltage at Pin 9 is referenced to V^-.

Output DC Level Control:

The dc level at the output (Pin 2) is approximately the same as the dc bias at Pin 3. In Figures 10, 11 and 12, Pin 3 is biased midway between V^+ and ground, to give an output dc level of $\approx V^+/2$.

APPLICATIONS INFORMATION

Sine Wave Generation

Without External Adjustment:

Figure 10 shows the circuit connection for generating a sinusoidal output from the XR-2206. The potentiometer, R_1 at Pin 7, provides the desired frequency tuning. The maximum output swing is greater than $V^+/2$, and the typical distortion (THD) is <2.5%. If lower sine wave distortion is desired, additional adjustments can be provided as described in the following section.

The circuit of Figure 10 can be converted to split-supply operation, simply by replacing all ground connections with V^-. For split-supply operation, R_3 can be directly connected to ground.

With External Adjustment:

The harmonic content of sinusoidal output can be reduced to $\approx$0.5% by additional adjustments as shown in Figure 11. The potentiometer, R_A, adjusts the sine-shaping resistor, and R_B provides the fine adjustment for the waveform symmetry. The adjustment procedure is as follows.

1. Set R_B at midpoint, and adjust R_A for minimum distortion.
2. With R_A set as above, adjust R_B to further reduce distortion.

Triangle Wave Generation

The circuits of Figures 10 and 11 can be converted to triangle wave generation, by simply open-circuiting Pin 13 and 14 (i.e., S_1 open). Amplitude of the triangle is approximately twice the sine wave output.

FSK Generation

Figure 12 shows the circuit connection for sinusoidal FSK signal operation. Mark and space frequencies can be independently adjusted, by the choice of timing resistors, R_1 and R_2; the output is phase-continuous during transitions. The keying signal is applied to Pin 9. The circuit can be converted to split-supply operation by simply replacing ground with V^-.

Pulse and Ramp Generation

Figure 13 shows the circuit for pulse and ramp waveform generation. In this mode of operation, the FSK keying terminal (Pin 9) is shorted to the square-wave output (Pin 11), and the circuit automatically frequency-shift keys itself between two separate frequencies during the positive-going and negative-going output waveforms. The pulse width and duty cycle can be adjusted from 1% to 99%, by the choice of R_1 and R_2. The values of R_1 and R_2 should be in the range of 1 kΩ to 2 MΩ.

XR-2206

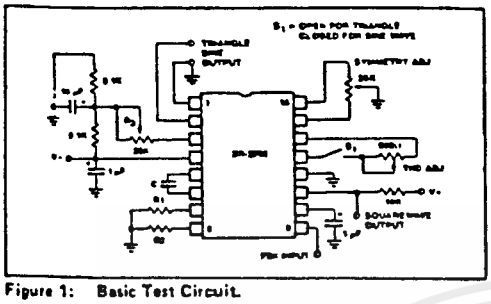


Figure 1: Basic Test Circuit.

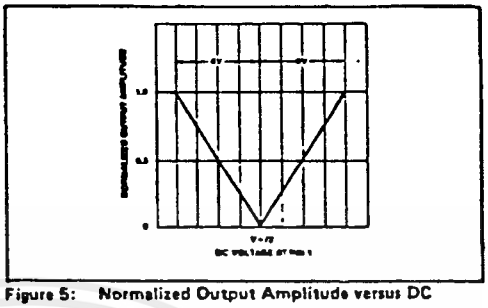


Figure 5: Normalized Output Amplitude versus DC Bias at AM Input (Pin 1).

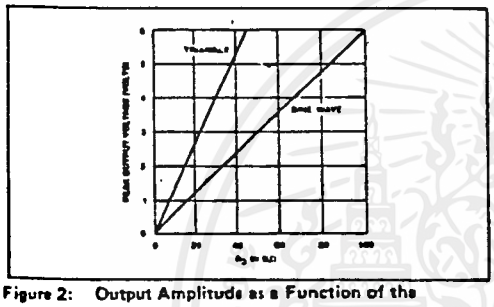


Figure 2: Output Amplitude as a Function of the Resistor, R_3, at Pin 3.

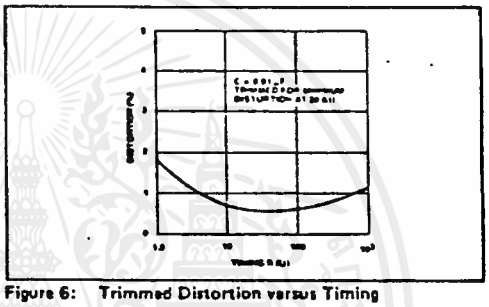


Figure 6: Trimmed Distortion versus Timing Resistor.

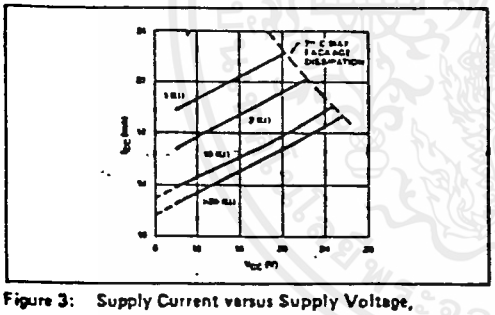


Figure 3: Supply Current versus Supply Voltage, Timing, R_1

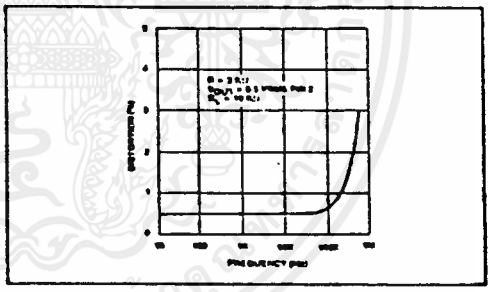


Figure 7: Sine Wave Distortion versus Operating Frequency with Timing Capacitors Varied.

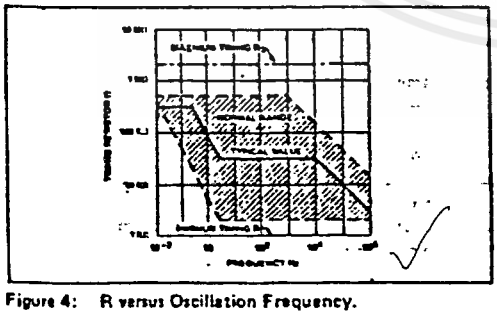


Figure 4: R versus Oscillation Frequency.

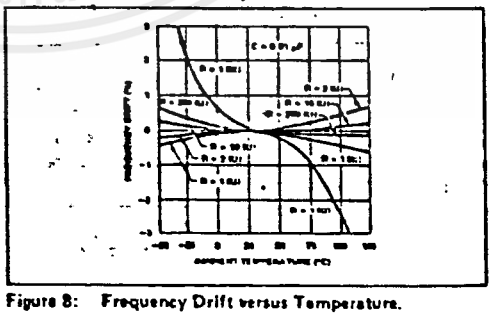


Figure 8: Frequency Drift versus Temperature.

PRINCIPLES OF OPERATION

Description of Controls

Frequency of Operation:
The frequency of oscillation, f_o, is determined by the external timing capacitor, C, across Pin 5 and 6, and by the timing resistor, R, connected to either Pin 7 or 8. The frequency is given as.

$$f_o = \frac{1}{RC} \text{ Hz}$$

and can be adjusted by varying either R or C. The recommended values of R, for a given frequency range, are shown in Figure 4. Temperature stability is optimum for $4\text{ k}\Omega < R < 200\text{ k}\Omega$. Recommended values of C are from 1000 pF to 100 μF.

Frequency Sweep and Modulation:
Frequency of oscillation is proportional to the total timing current, I_T, drawn from Pin 7 or 8:

$$f = \frac{320 I_T \text{ (mA)}}{C\ (\mu F)} \text{ Hz}$$

Timing terminals (Pin 7 or 8) are low-impedance points, and are internally biased at +3V, with respect to Pin 12. Frequency varies linearly with I_T, over a wide range of current values, from 1 μA to 3 mA. The frequency can be controlled by applying a control voltage, V_C, to the activated timing pin as shown in Figure 9. The frequency of oscillation is related to V_C as:

$$f = \frac{1}{RC}\left[1 + \frac{R}{R_C}\left(1 - \frac{V_C}{3}\right)\right] \text{ Hz}$$

where V_C is in volts. The voltage-to-frequency conversion gain, K, is given as:

$$K = \partial f / \partial V_C = -\frac{0.32}{R_C C} \text{ Hz/V}$$

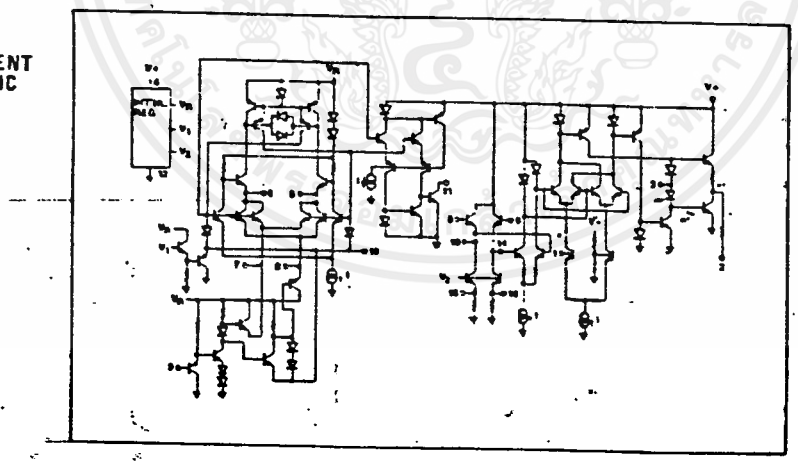
CAUTION: For safe operation of the circuit, I_T should be limited to ≤ 3 mA.

Output Amplitude:
Maximum output amplitude is inversely proportional to the external resistor, R_3, connected to Pin 3 (see Figure 2). For sine wave output, amplitude is approximately 60 mV peak per kΩ of R_3, for triangle, the peak amplitude is approximately 160 mV peak per kΩ of R_3. Thus, for example, $R_3 = 50\text{ k}\Omega$ would produce approximately ± 3V sinusoidal output amplitude.

Amplitude Modulation:
Output amplitude can be modulated by applying a dc bias and a modulating signal to Pin 1. The internal impedance at Pin 1 is approximately 100 kΩ. Output amplitude varies linearly with the applied voltage at Pin 1, for values of dc bias at this pin, within ± 4 volts of $V^+/2$ as shown in Figure 5. As this bias level approaches $V^+/2$, the phase of the output signal is reversed, and the amplitude goes through zero. This property is suitable for phase-shift keying and suppressed-carrier AM generation. Total dynamic range of amplitude modulation is approximately 55 dB.

CAUTION: AM control must be used in conjunction with a well-regulated supply, since the output amplitude now becomes a function of V^+.

EQUIVALENT SCHEMATIC DIAGRAM



FSK Demodulator / Tone Decoder

GENERAL DESCRIPTION

The XR-2211 is a monolithic phase-locked loop (PLL) system especially designed for data communications. It is particularly well suited for FSK modem applications. It operates over a wide supply voltage range of 4.5 to 20 V and a wide frequency range of 0.01 Hz to 300 kHz. It can accommodate analog signals between 2 mV and 3 V, and can interface with conventional DTL, TTL, and ECL logic families. The circuit consists of a basic PLL for tracking an input signal within the pass band, a quadrature phase detector which provides carrier detection, and an FSK voltage comparator which provides FSK demodulation. External components are used to independently set center frequency, bandwidth, and output delay. An internal voltage reference proportional to the power supply provides ratio metric operation for low system performance variations with power supply changes.

The XR-2211 is available in 14 pin DTL ceramic or plastic packages specified for commercial or military temperature ranges.

FEATURES

Wide Frequency Range	0.01 Hz to 300 kHz
Wide Supply Voltage Range	4.5 V to 20 V
DTL/TTL/ECL Logic Compatibility	
FSK Demodulation, with Carrier Detection	
Wide Dynamic Range	2 mV to 3 V rms
Adjustable Tracking Range (±1% to ±80%)	
Excellent Temp. Stability	20 ppm/°C, typ.

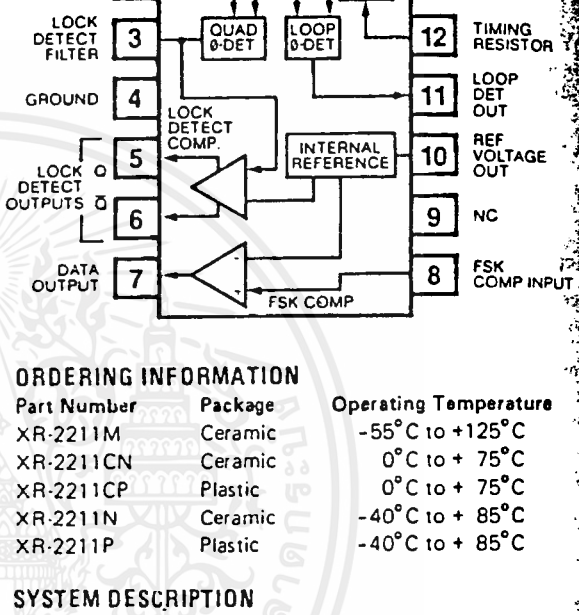
APPLICATIONS

FSK Demodulation
Data Synchronization
Tone Decoding
FM Detection
Carrier Detection

ABSOLUTE MAXIMUM RATINGS

Power Supply	20 V
Input Signal Level	3 V rms
Power Dissipation	
Ceramic Package	750 mW
Derate above T_A = +25°C	6 mW/°C
Plastic Package	625 mW
Derate above T_A = +25°C	5.0 mW/°C

FUNCTIONAL BLOCK DIAGRAM



ORDERING INFORMATION

Part Number	Package	Operating Temperature
XR-2211M	Ceramic	-55°C to +125°C
XR-2211CN	Ceramic	0°C to + 75°C
XR-2211CP	Plastic	0°C to + 75°C
XR-2211N	Ceramic	-40°C to + 85°C
XR-2211P	Plastic	-40°C to + 85°C

SYSTEM DESCRIPTION

The main PLL within the XR-2211 is constructed from an input preamplifier, analog multiplier used as a phase detector, and a precision voltage controlled oscillator (VCO). The preamplifier is used as a limiter such that input signals above typically 2MV RMS are amplified to a constant high level signal. The multipling-type phase detector acts as a digital exclusive or gate. Its output (unfiltered) produces sum and difference frequencies of the input and the VCO output, f input + f input (2 f input) and f input - f input (0 Hz) when the phase detector output to remove the "sum" frequency component while passing the difference (DC) component to drive the VCO. The VCO is actually a current controlled oscillator with its nominal input current (I_0) set by a resistor (R_0) to ground and its driving current with a resistor (R_1) from the phase detector.

The other sections of the XR-2211 act to determine if the VCO is driven above or below the center frequency (FSK comparator), produced both active high and active low outputs to indicate when the main PLL is in lock (quadrature phase detector and lock detector comparator).

ELECTRICAL CHARACTERISTICS

Test Conditions: Test Circuit of Figure 1, $V^+ = V^- = 6V$, $T_A = +25°C$, $C = 5000$ pF, $R_1 = R_2 = R_3 = R_4 = 20$ KΩ, $R_L = 4.7$ KΩ, Binary Inputs grounded, S_1 and S_2 closed unless otherwise specified.

PARAMETERS	XR-2211/2211M			XR-2211C			UNITS	CONDITIONS
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.		
GENERAL								
Supply Voltage	4.5		20	4.5		20	V	
Supply Current		4	7		5	9	mA	$R_0 \geq 10$ KΩ See Fig. 4
OSCILLATOR SECTION								
Frequency Accuracy		±1	±3		±1		%	Deviation from $f_0 = 1/R_0C_0$
Frequency Stability								R_1 = ½
Temperature		±20	±50		±20		ppm/°C	See Fig. 8.
Power Supply		0.05	0.5		0.05		%/V	$V^+ = 12 \pm 1$ V. See Fig. 7.
		0.2			0.2		%/V	$V^+ = 5 \pm 0.5$ V. See Fig. 7.
Upper Frequency Limit	100	300			300		kHz	$R_0 = 8.2$ KΩ, $C_0 = 400$ pF
Lowest Practical								
Operating Frequency			0.01		0.01		Hz	$R_0 = 2$ MΩ, $C_0 = 50$ µF
Timing Resistor, R_0								See Fig. 5.
Operating Range	5		2000	5		2000	KΩ	
Recommended Range	15		100	15		100	KΩ	See Fig. 7 and 8.
LOOP PHASE DETECTOR SECTION								
Peak Output Current	±150	±200	±300	±100	±200	±300	µA	Measured at Pin 11.
Output Offset Current		±1			±2		µA	
Output Impedance		1			1		MΩ	
Maximum Swing	±4	±5		±4	±5		V	Referenced to Pin 10.
QUADRATURE PHASE DETECTOR								Measured at Pin 3.
Peak Output Current	100	150			150		µA	
Output Impedance		1			1		MΩ	
Maximum Swing		11			11		V pp	
INPUT PREAMP SECTION								Measured at Pin 2.
Input Impedance		20			20		KΩ	
Input Signal								
Voltage Required to Cause Limiting		2	10		2		mV rms	
VOLTAGE COMPARATOR SECTIONS								
Input Impedance		2			2		MΩ	Measured at Pins 3 and 8.
Input Bias Current		100			100		nA	
Voltage Gain	55	70		55	70		dB	$R_L = 5.1$ KΩ
Output Voltage Low		300			300		mV	$I_C = 3$ mA
Output Leakage Current		0.01			0.01		µA	$V_O = 12$ V
INTERNAL REFERENCE								
Voltage Level	4.9	5.3	5.7	4.75	5.3	5.85	V	Measured at Pin 10.
Output Impedance		100			100		Ω	

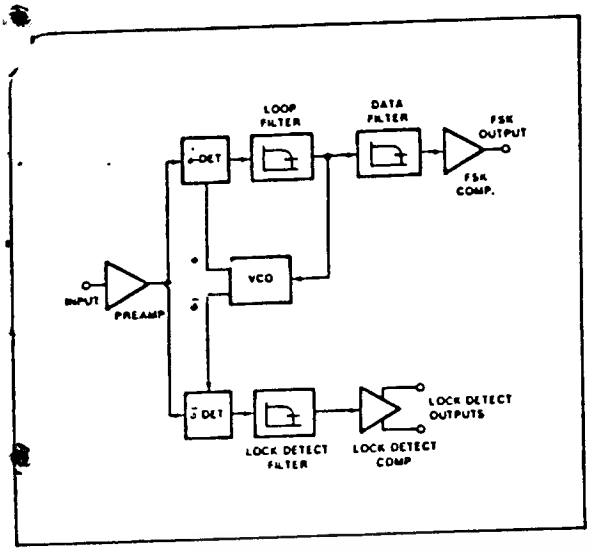


Figure 1: Functional Block Diagram of a Tone and FSK Decoding System Using XR-2211

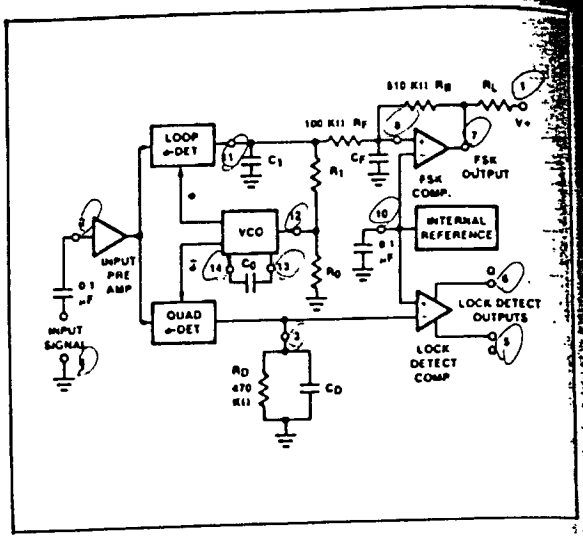


Figure 2: Generalized Circuit Connection for FSK and Tone Detection

Reference Voltage, V_R (Pin 10) This pin is internally biased at the reference voltage level, V_R. V_R = V+/2 - 650 mV. The dc voltage level at this pin forms an internal reference for the voltage levels at Pins 5, 8, 11 and 12. Pin 10 *must* be bypassed to ground with a 0.1 μF capacitor for proper operation of the circuit.

Loop Phase Detector Output (Pin 11) This terminal provides a high impedance output for the loop phase detector. The PLL loop filter is formed by R_1 and C_1 connected to Pin 11 (see Figure 2). With no input signal, or with no phase error within the PLL, the dc level at Pin 11 is very nearly equal to V_R. The peak voltage swing available at the phase detector output is equal to $\pm V_R$.

VCO Control Input (Pin 12) VCO free-running frequency is determined by external timing resistor, R_0, connected from this terminal to ground. The VCO free-running frequency, f_0, is

$$f_0 = \frac{1}{R_0 C_0} \text{ Hz}$$

where C_0 is the timing capacitor across Pins 13 and 14. For optimum temperature stability, R_0 must be in the range of 10 KΩ to 100 KΩ see Figure 8).

This terminal is a low impedance point, and is internally biased at a dc level equal to V_R. The maximum timing current drawn from Pin 12 must be limited to ≤3 mA for proper operation of the circuit.

VCO Timing Capacitor (Pins 13 and 14). VCO frequency is inversely proportional to the external timing capacitor, C_0, connected across these terminals (see Figure 5). C_0 must be nonpolar, and in the range of 200 pF to 10 μF.

VCO Frequency Adjustment: VCO can be fine-tuned by connecting a potentiometer, R_X, in series with R_0 at Pin 12 (see Figure 9).

VCO Free-Running Frequency, f_0: XR-2211 does not have a separate VCO output terminal. Instead, the VCO outputs are internally connected to the phase detector sections of the circuit. However, for set-up or adjustment purposes, VCO free-running frequency can be measured at Pin 3 (with C_D disconnected), with no input and with Pin 2 shorted to Pin 10.

DESIGN EQUATIONS

(See Figure 2 for definition of components.)

1. VCO Center Frequency, f_0.

   $f_0 = 1/R_0C_0$ Hz

2. Internal Reference Voltage, V_R (measured at Pin 10).

   $V_R = V+/2 - 650$ mV

3. Loop Low-Pass Filter Time Constant, τ

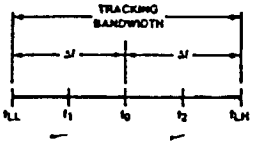
   $\tau = R_1C_1$

Loop Damping, ζ:

$$\zeta = 1/4\sqrt{\frac{C_0}{C_1}}$$

Loop Tracking Bandwidth, $\pm\Delta f/f_0$:

$\Delta f/f_0 = R_0/R_1$



FSK Data Filter Time Constant, τF:

$\tau F = R_F C_F$

Loop Phase Detector Conversion Gain, $K\phi$: ($K\phi$ is the differential dc voltage across Pins 10 and 11, per unit of phase error at phase detector input):

$K\phi = -2V_R/\pi$ volts/radian

VCO Conversion Gain, K_0: (K_0 is the amount of change in VCO frequency, per unit of dc voltage change at Pin 11):

$K_0 = -1/V_R C_0 R_1$ Hz/volt

Total Loop Gain, K_T:

$K_T = 2\pi K\phi K_0 = 4/C_0 R_1$ rad/sec/volt

Peak Phase Detector Current I_A:

$I_A = V_R$ (volts)/25 mA

APPLICATIONS INFORMATION

FSK DECODING:

Figure 9 shows the basic circuit connection for FSK decoding. With reference to Figures 2 and 9, the functions of external components are defined as follows: R_0 and C_0 set the PLL center frequency, R_1 sets the system bandwidth, and C_1 sets the loop filter time constant and the loop damping factor. C_F and R_F form a one-pole post-detection filter for the FSK data output. The resistor R_B (= 510 KΩ) from Pin 7 to Pin 8 introduces positive feedback across the FSK comparator to facilitate rapid transition between output logic states.

Recommended component values for some of the most commonly used FSK bands are given in Table 1.

Design Instructions:

The circuit of Figure 9 can be tailored for any FSK decoding application by the choice of five key circuit components: R_0, R_1, C_0, C_1 and C_F. For a given set of FSK mark and space frequencies, f_1 and f_2, these parameters can be calculated as follows:

a) Calculate PLL center frequency, f_0:

$$f_0 = \frac{f_1 + f_2}{2}$$

b) Choose value of timing resistor R_0, to be in the range of 10 KΩ to 100 KΩ. This choice is arbitrary. The recommended value is $R_0 \equiv 20$ KΩ. The final value of R_0 is normally fine-tuned with the series potentiometer, R_X.

c) Calculate value of C_0 from design equation (1) or from Figure 6:

$C_0 = 1/R_0 f_0$

d) Calculate R_1 to give a Δf equal to the mark space deviation:

$R_1 = R_0[f_0/(f_1-f_2)]$

e) Calculate C_1 to set loop damping. (See design equation no. 4.):

Normally, $\zeta \approx 1/2$ is recommended.

Then: $C_1 = C_0/4$ for $\zeta = 1/2$

f) Calculate Data Filter Capacitance, C_F:

For R_F = 100 KΩ, R_B = 510 KΩ, the recommended value of C_F is:

$C_F \approx 3/(\text{Baud Rate})\ \mu F$

Note: All calculated component values except R_0 can be rounded to the nearest standard value, and R_0 can be varied to fine-tune center frequency, through a series potentiometer, R_X. (See Figure 9.)

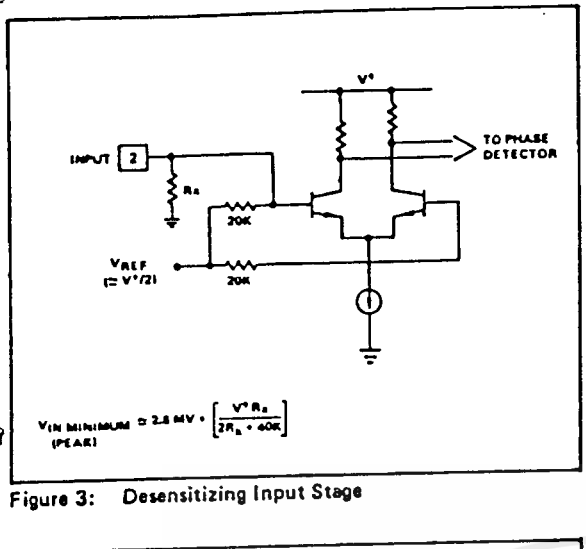


Figure 3: Desensitizing Input Stage

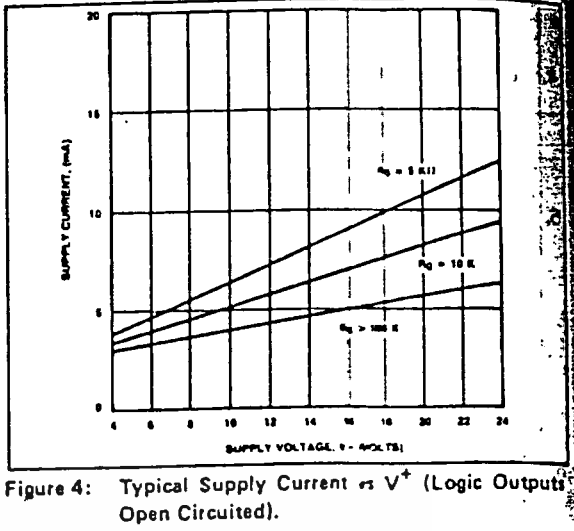


Figure 4: Typical Supply Current vs V^+ (Logic Outputs Open Circuited).

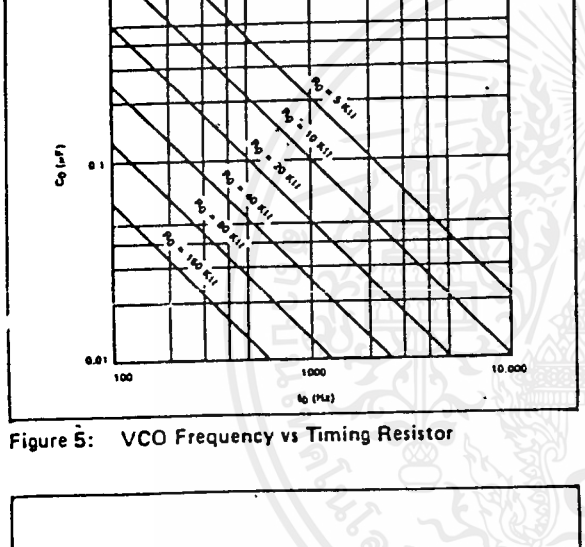


Figure 5: VCO Frequency vs Timing Resistor

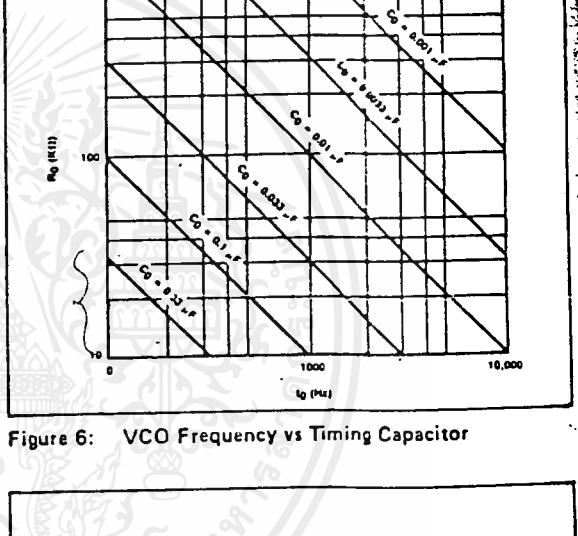


Figure 6: VCO Frequency vs Timing Capacitor

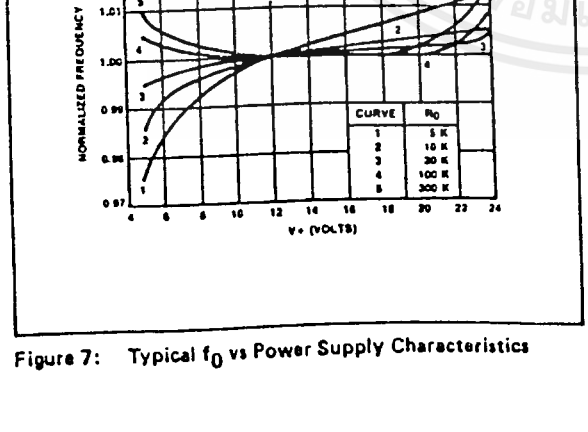


Figure 7: Typical f_0 vs Power Supply Characteristics

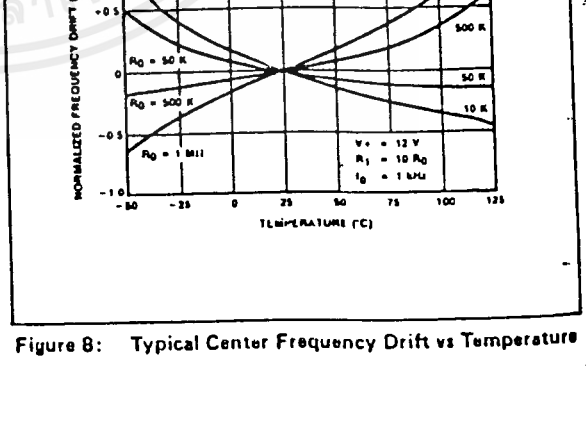


Figure 8: Typical Center Frequency Drift vs Temperature

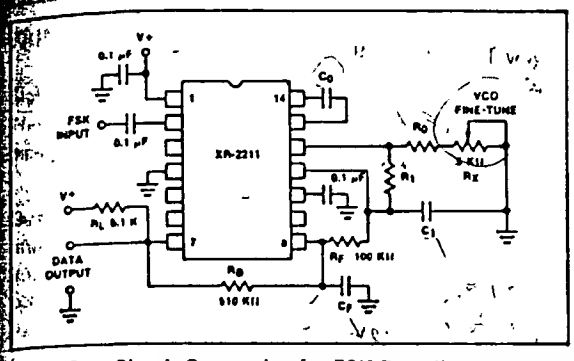


Figure 9: Circuit Connection for FSK Decoding

Design Example:

75 Baud FSK demodulator with mark space frequencies of 1110/1170 Hz:

Step 1: Calculate f_0: f_0 = (1110 + 1170) (1/2) = 1140 Hz
Step 2: Choose R_0 = 20 KΩ (18 KΩ fixed resistor in series with 5 KΩ potentiometer)
Step 3: Calculate C_0 from Figure 6: C_0 = 0.044 μF
Step 4: Calculate $R_1 \cdot R_1$ = R_0 (2240/60) = 380 KΩ
Step 5: Calculate $C_1 \cdot C_1 = C_0/4$ = 0.011 μF

Note: All values except R_0 can be rounded to *nearest* standard value.

Table 1. Recommended Component Values for Commonly Used FSK Bands. (See Circuit of Figure 9.)

FSK BAND	COMPONENT VALUES	
300 Baud f_1 = 1070 Hz f_2 = 1270 Hz	C_0 = 0.039 μF C_1 = 0.01 μF R_1 = 100 KΩ	C_F = 0.005 μF R_0 = 18 KΩ
300 Baud f_1 = 2025 Hz f_2 = 2225 Hz	C_0 = 0.022 μF C_1 = 0.0047 μF R_1 = 200 KΩ	C_F = 0.005 μF R_0 = 18 KΩ
1200 Baud f_1 = 1200 Hz f_2 = 2200 Hz	C_0 = 0.027 μF C_1 = 0.01 μF R_1 = 30 KΩ	C_F = 0.0022 μF R_0 = 18 KΩ

FSK DECODING WITH CARRIER DETECT:

The lock detect section of XR-2211 can be used as a carrier detect option, for FSK decoding. The recommended circuit connection for this application is shown in Figure 10. The open collector lock detect output, Pin 6, is shorted to data output (Pin 7). Thus, data output will be disabled at "low" state, until there is a carrier within the detection band of the PPL, and the Pin 6 output goes "high," to enable the data output.

The minimum value of the lock detect filter capacitance C_D is inversely proportional to the capture range, $\pm\Delta f_c$. This is the range of incoming frequencies over which the loop can acquire lock and is always less than the tracking range. It is further limited by C_1. For most applications, $\Delta f_c > \Delta f/2$. For R_D = 470 KΩ, the approximate minimum value of C_D can be determined by:

C_D (μF) ≥ 16/capture range in Hz.

With values of C_D that are too small, chatter can be observed on the lock detect output as an incoming signal frequency approaches the capture bandwidth. Excessively large values of C_D will slow the response time of the lock detect output.

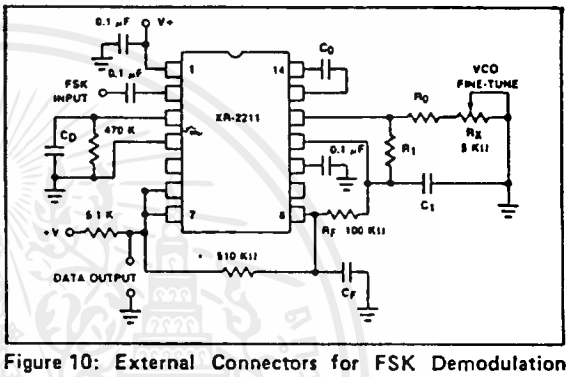


Figure 10: External Connectors for FSK Demodulation with Carrier Detect Capability

Note: Data Output is "Low" When No Carrier is Present.

TONE DETECTION:

Figure 11 shows the generalized circuit connection for tone detection. The logic outputs, Q and $\overline{Q}$ at Pins 5 and 6 are normally at "high" and "low" logic states, respectively. When a tone is present within the detection band of the PLL, the logic state at these outputs become reversed for the duration of the input tone. Each logic output can sink 5 mA of load current.

Both logic outputs at Pins 5 and 6 are open collector type stages, and require external pull-up resistors R_{L1} and R_{L2}, as shown in Figure 11.

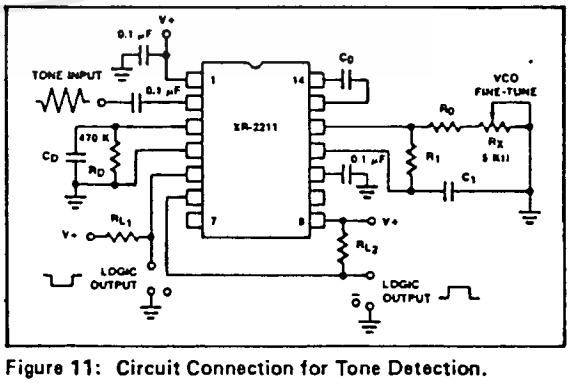


Figure 11: Circuit Connection for Tone Detection.

With reference to Figures 2 and 11, the functions of the external circuit components can be explained as follows: R_0 and C_0 set VCO center frequency; R_1 sets the detection bandwidth; C_1 sets the low pass-loop filter time constant and the loop damping factor. R_{L1} and R_{L2} are the respective pull-up resistors for the Q and $\bar{Q}$ logic outputs.

Design Instructions:

The circuit of Figure 11 can be optimized for any tone detection application by the choice of the 5 key circuit components: R_0, R_1, C_0, C_1 and C_D. For a given input, the tone frequency, f_S, these parameters are calculated as follows:

a) Choose R_0 to be in the range of 15 KΩ to 100 KΩ. This choice is arbitrary.

b) Calculate C_0 to set center frequency, f_0 equal to f_s (see Figure 6): $C_0 = 1/R_0 f_S$

c) Calculate R_1 to set bandwidth $\pm\Delta f$ (see design equation no. 5):

$$R_1 = R_0(f_0/\Delta f)$$

Note: The total detection bandwidth covers the frequency range of $f_0 \pm \Delta f$.

d) Calculate value of C_1 for a given loop damping factor.

$$C_1 = C_0/16\zeta 2$$

Normally $\zeta \approx 1/2$ is optimum for most tone detector applications, giving $C_1 = 0.25\ C_0$.

Increasing C_1 improves the out-of-band signal rejection, but increases the PLL capture time.

e) Calculate value of filter capacitor C_D. To avoid chatter at the logic output, with R_D = 470 KΩ, C_D must be:

$$C_D(\mu F) \geq (16/\text{capture range in Hz})$$

Increasing C_D slows down the logic output response time.

Design Examples:

Tone detector with a detection band of 1 kHz ± 20 Hz:

a) Choose R_0 = 20 KΩ (18 KΩ in series with 5 KΩ potentiometer).
b) Choose C_0 for f_0 = 1 kHz (from Figure 6) · C_0 = 0.05 μF.

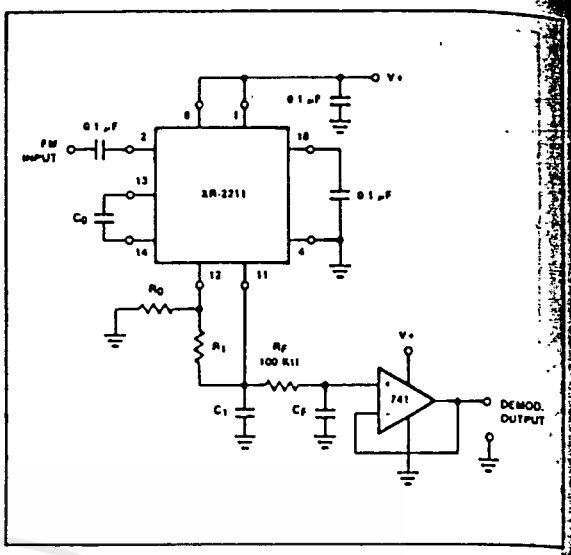


Figure 12: Linear FM Detector Using XR-2211 and an External Op Amp. (See section on Design Equation for Component Values.)

c) Calculate R_1. $R_1 = (R_0)\ (1000/20) = 1$ MΩ.
d) Calculate C_1 for $\zeta = 1/2$. $C_1 = 0.25$. $C_0 = 0.013\ \mu F$.
e) Calculate C_D $C_D = 16/38 = 0.42\ \mu F$.
f) Fine-tune center frequency with 5 KΩ potentiometer, R_X.

LINEAR FM DETECTION:

XR-2211 can be used as a linear FM detector for a wide range of analog communications and telemetry applications. The recommended circuit connection for this application is shown in Figure 12. The demodulated output is taken from the loop phase detector output (Pin 11), through a post-detection filter made up of R_F and C_F, and an external buffer amplifier. This buffer amplifier is necessary because of the high impedance output at Pin 11. Normally, a non-inverting unity gain op amp can be used as a buffer amplifier, as shown in Figure 12.

The FM detector gain, i.e., the output voltage change per unit of FM deviation can be given as:

$$V_{out} = R_1\ V_R/100\ R_0 \text{ Volts/\%deviation}$$

where V_R is the internal reference voltage (V_R = V+/2—650 mV). For the choice of external components R_1, R_0, C_D, C_1 and C_F, see section on design equations.

PRINCIPLES OF OPERATION

Signal Input (Pin 2): Signal is ac coupled to this terminal. The internal impedance at Pin 2 is 20 KΩ. Recommended input signal level is in the range of 10 mV rms to 3 V rms.

Quadrature Phase Detector Output (Pin 3): This is the high impedance output of quadrature phase detector and is internally connected to the input of lock detect voltage comparator. In tone detection applications, Pin 3 is connected to ground through a parallel combination of R_D and C_D (see Figure 2) to eliminate the chatter at lock detect outputs. If the tone detect section is not used, Pin 3 can be left open circuited.

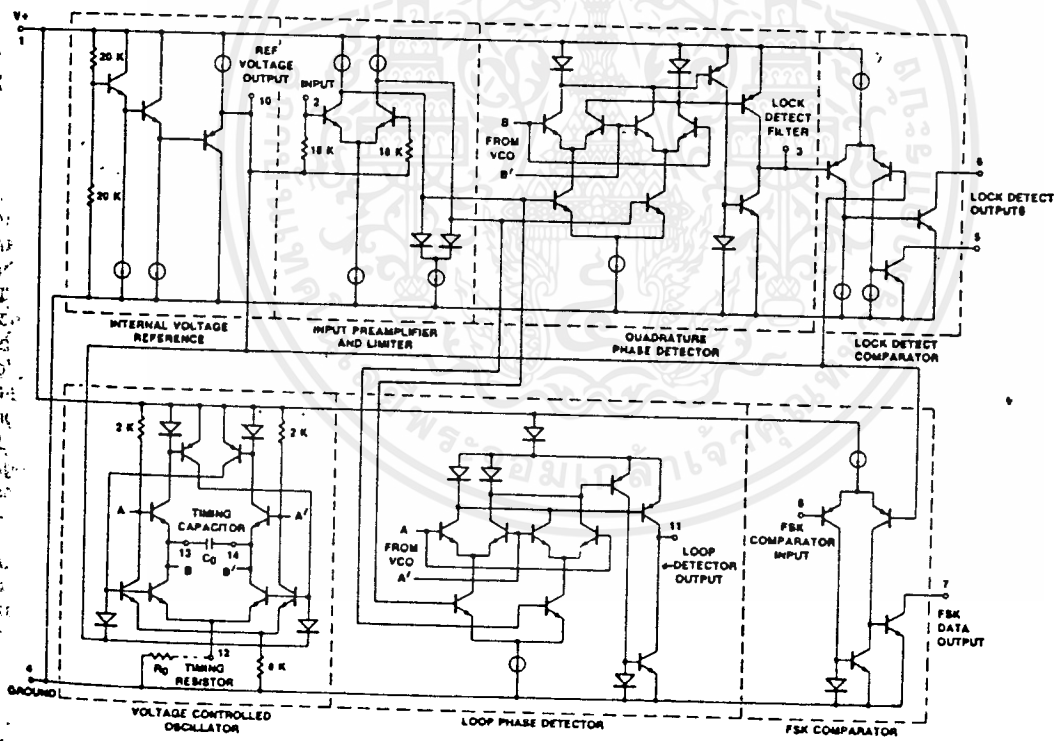
Lock Detect Output, Q (Pin 5). The output at Pin 5 is at "high" state when the PLL is out of lock and goes to "low" or conducting state when the PLL is locked. It is an open collector type output and requires a pull-up resistor, R_L, to V+ for proper operation. At "low" state, it can sink up to 5 mA of load current.

Lock Detect Complement, $\overline{Q}$ (Pin 6): The output at Pin 6 is the logic complement of the lock detect output at Pin 5. This output is also an open collector type stage which can sink 5 mA of load current at low or "on" state.

FSK Data Output (Pin 7): This output is an open collector logic stage which requires a pull-up resistor, R_L, to V+ for proper operation. It can sink 5 mA of load current. When decoding FSK signals, FSK data output is at "high" or "off" state for low input frequency, and at "low" or "on" state for high input frequency. If no input signal is present, the logic state at Pin 7 is indeterminate.

FSK Comparitor Input (Pin 8): This is the high impedance input to the FSK voltage comparator. Normally, an FSK post-detection or data filter is connected between this terminal and the PLL phase detector output (Pin 11). This data filter is formed by R_F and C_F of Figure 2. The threshold voltage of the comparator is set by the internal reference voltage, V_R, available at Pin 10.

EQUIVALENT SCHEMATIC DIAGRAM



MOTOROLA

MC3362

Advance Information

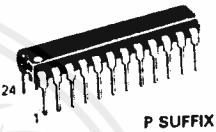
LOW POWER NARROWBAND FM RECEIVER

. . . includes dual FM conversion with oscillators, mixers, quadrature detector, and meter drive carrier detect circuitry. The MC3362 also has buffered first and second local oscillator outputs and a comparator circuit for FSK detection.

- Wide Input Bandwidth:
  - — 200 MHz using Internal Local Oscillator
  - — 450 MHz using External Local Oscillator
- Complete Dual Conversion Circuitry
- Low Voltage: V_{CC} = 2.0 to 7.0 Vdc
- Low Drain Current (3.6 mA (Typ) @ V_{CC} = 3.0 Vdc)
- Excellent Sensitivity: Input 0.7 μV (Typ) for 12 dB SINAD
- Data Shaping Comparator
- Received Signal Strength Indicator (RSSI) with 60 dB Dynamic Range
- Low Number of External Parts Required
- Manufactured in Motorola's MOSAIC Process Technology

LOW POWER DUAL CONVERSION FM RECEIVER

SILICON MONOLITHIC INTEGRATED CIRCUIT



P SUFFIX
PLASTIC PACKAGE
CASE 724-02

DW SUFFIX
PLASTIC PACKAGE
CASE 751E-02
SO-24

FIGURE 1 — TYPICAL APPLICATION IN A PLL FREQUENCY SYNTHESIZED RECEIVER

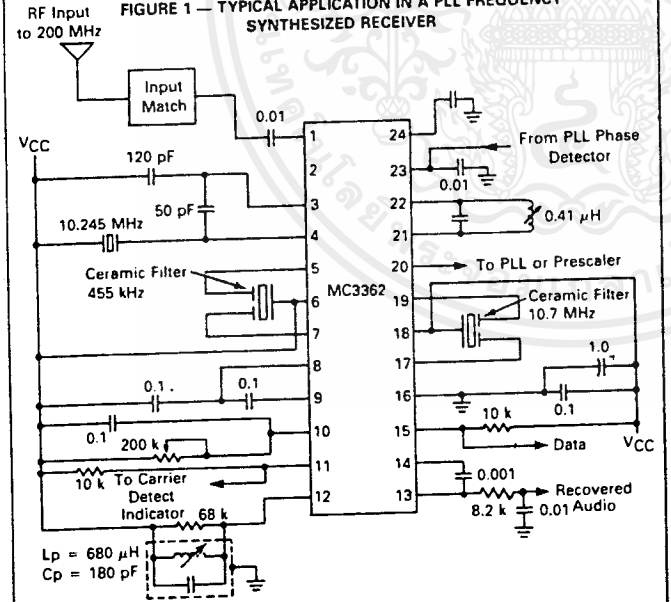
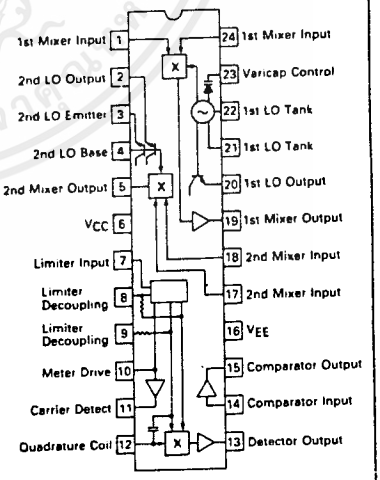


FIGURE 2 — PIN CONNECTIONS AND FUNCTIONAL BLOCK DIAGRAM



8

This document contains information on a new product. Specifications and information herein are subject to change without notice.

MAXIMUM RATINGS (T_A = 25°C, unless otherwise noted)

Rating	Pin	Symbol	Value	Unit
Power Supply Voltage	6	$V_{CC(max)}$	8.0	Vdc
Operating Supply Voltage Range (Recommended)	6	V_{CC}	2.0 to 7.0	Vdc
Input Voltage (V_{CC} ≥ 5.0 Vdc)	1, 24	V_{1-24}	1.0	Vrms
Junction Temperature	—	T_J	150	°C
Operating Ambient Temperature Range	—	T_A	−40 to +85	°C
Storage Temperature Range	—	T_{stg}	−65 to +150	°C

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 Vdc, f_o = 49.7 MHz, Deviation = 3.0 kHz, T_A = 25°C, Test Circuit of Figure 3 unless otherwise noted)

Characteristic	Pin	Min	Typ	Max	Units
Drain Current (Carrier Detect Low — See Figure 5)	6	—	4.5	7.0	mA
Input for −3.0 dB Limiting	—	—	0.7	2.0	μVrms
Recovered Audio (RF signal level = 10 mV)	13	—	350	—	mVrms
Noise Output (RF signal level = 0 mV)	13	—	250	—	mVrms
Carrier Detect Threshold (below V_{CC})	10	—	0.64	—	Vdc
Meter Drive Slope	10	—	100	—	nA/dB
Input for 20 dB (S + N)/N (See Figure 7)	—	—	0.7	—	μVrms
First Mixer 3rd Order Intercept (Input)	—	—	−22	—	dBm
First Mixer Input Resistance (Rp)	—	—	690	—	Ω
First Mixer Input Capacitance (Cp)	—	—	7.2	—	pF
First Mixer Conversion Voltage Gain	—	—	18	—	dB
Second Mixer Conversion Voltage Gain	—	—	21	—	dB
Detector Output Resistance	13	—	1.4	—	kΩ

FIGURE 3 — TEST CIRCUIT

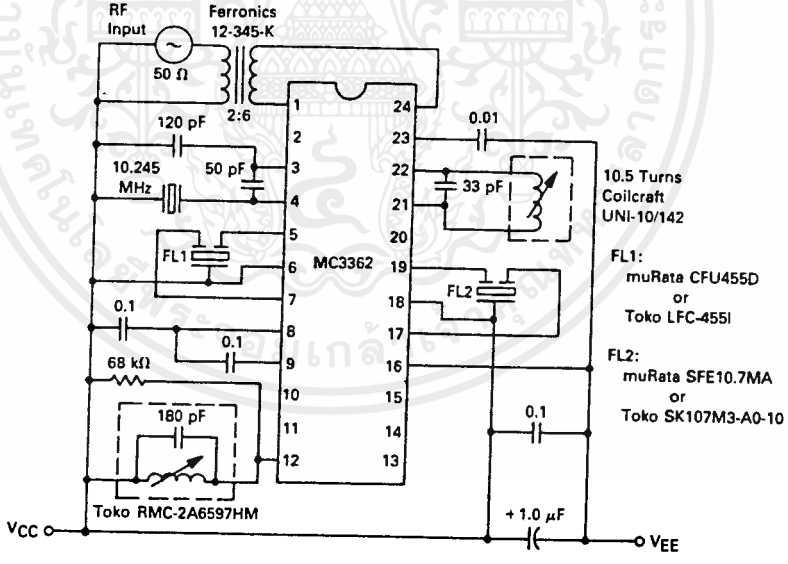


FIGURE 4 — I_{METER} versus INPUT

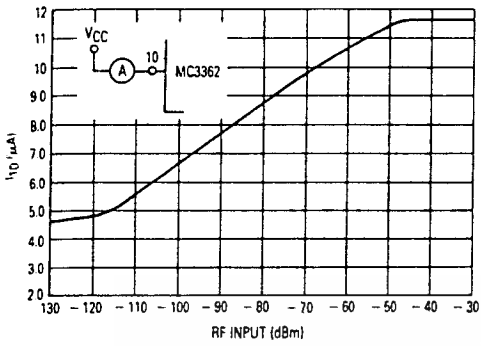


FIGURE 5 — DRAIN CURRENT, RECOVERED AUDIO versus SUPPLY

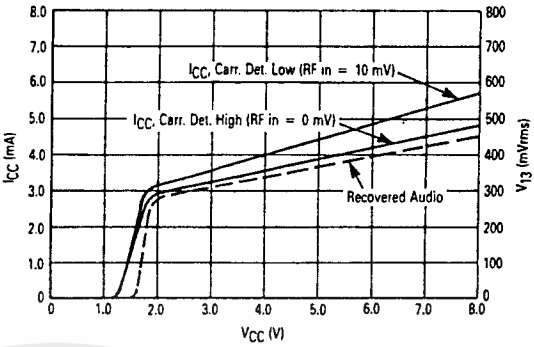


FIGURE 6 — SIGNAL LEVELS

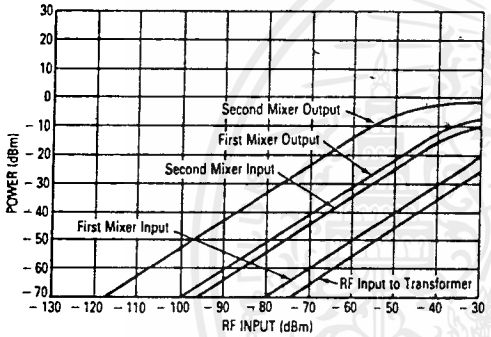


FIGURE 7 — S + N, N, AMR versus INPUT

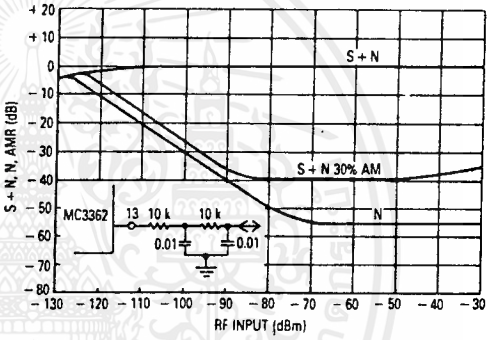


FIGURE 8 — 1ST MIXER 3RD ORDER INTERMODULATION

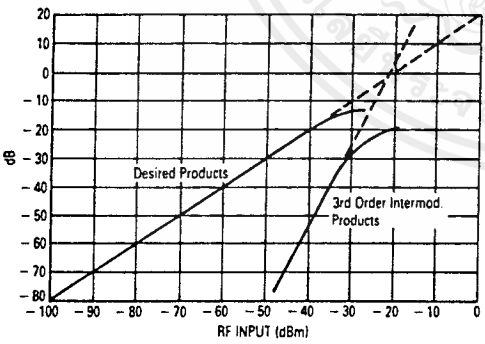
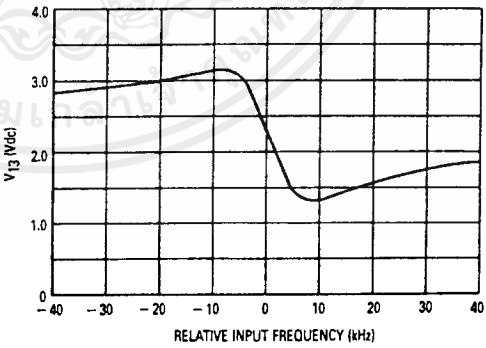


FIGURE 9 — DETECTOR OUTPUT versus FREQUENCY



8

CIRCUIT DESCRIPTION

The MC3362 is a complete FM narrowband receiver from antenna input to audio preamp output. The low voltage dual conversion design yields low power drain, excellent sensitivity and good image rejection in narrowband voice and data link applications.

In the typical application (Figure 1), the first mixer amplifies the signal and converts the RF input to 10.7 MHz. This IF signal is filtered externally and fed into the second mixer, which further amplifies the signal and converts it to a 455 kHz IF signal. After external bandpass filtering, the low IF is fed into the limiting amplifier and detection circuitry. The audio is recovered using a conventional quadrature detector. Twice-IF filtering is provided internally.

The input signal level is monitored by meter drive circuitry which detects the amount of limiting in the limiting amplifier. The voltage at the meter drive pin determines the state of the carrier detect output, which is active low.

APPLICATION

The first local oscillator can be run using a free-running LC tank, as a VCO using PLL synthesis, or driven from an external crystal oscillator. It has been run to 190 MHz.* A buffered output is available at Pin 20. The second local oscillator is a common base Colpitts type which is typically run at 10.245 MHz under crystal control. A buffered output is available at Pin 2. Pins 2 and 3 are interchangeable.

The mixers are doubly balanced to reduce spurious responses. The first and second mixers have conversion gains of 18 dB and 22 dB (typical), respectively, as seen in Figure 6. Mixer gain is stable with respect to supply voltage. For both conversions, the mixer impedances and pin layout are designed to allow the user to employ low cost, readily available ceramic filters. Overall sensitivity and AM rejection are shown in Figure 7. The input level for 20 dB (S + N)/N is 0.7 μV using the two-pole post-detection filter pictured.

*If the first local oscillator (Pins 21 and/or 22) is driven from a strong external source (100 mVrms), the mixer can be used to over 450 MHz

Following the first mixer, a 10.7 MHz ceramic bandpass filter is recommended. The 10.7 MHz filtered signal is then fed into one second mixer input pin, the other input pin being connected to V_{CC}.

The 455 kHz IF is typically filtered using a ceramic bandpass filter then fed into the limiter input pin. The limiter has 10 μV sensitivity for −3.0 dB limiting. Set to 1.0 MHz.

The output of the limiter is internally connected to the quadrature detector, including a quadrature capacitor. A parallel LC tank is needed externally from Pin 12 to V_{CC}. A 68 kΩ shunt resistance is included which determines the peak separation of the quadrature detector; a smaller value will increase the spacing and linearity but decrease recovered audio and sensitivity.

A data shaping circuit is available and can be coupled to the recovered audio output of Pin 13. The circuit is a comparator which is designed to detect zero crossings of FSK modulation. Data rates of 2000 to 35000 baud are detectable using the circuit of Figure 1. Hysteresis is available by connecting a high-valued resistor from Pin 15 to Pin 14. Values below 120 kΩ are not recommended as the input signal cannot overcome the hysteresis.

The meter drive circuitry detects input signal level by monitoring the limiting of the limiting amplifier stages. Figure 4 shows the unloaded current at Pin 10 versus input power. The meter drive current can be used directly (RSSI) or can be used to trip the carrier detect circuit at a specified input power. To do this, pick an RF trip level in dBm. Read the corresponding current from Figure 4 and pick a resistor such that:

$$R_{10} \approx 0.64\ Vdc / I_{10}$$

Hysteresis is available by connecting a high-valued resistor R_H between Pins 10 and 11. The formula is:

$$Hyst. = V_{CC}/(R_H \times 10^{-7})\ dB$$

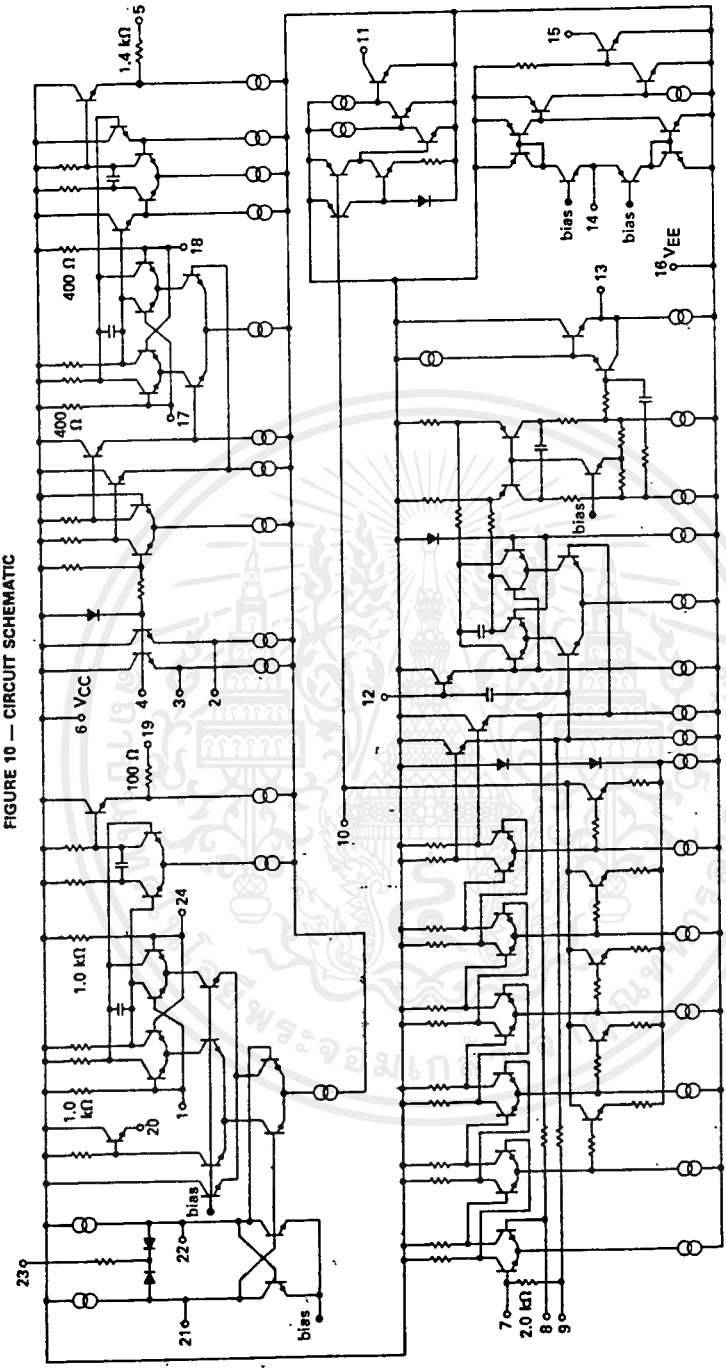


FIGURE 10 — CIRCUIT SCHEMATIC

บรรณานุกรม

1. สุพจน์ ปุณณชัยยะ ,MODEM ,อินฟอร์เมติก บิซิเนส พับลิเคชั่น ,กรุงเทพฯ : 1991
2. รศ.ประทีป บัญญัตินพรัตน์ ,การสื่อสารข้อมูล ,กรุงเทพฯ : 1989
3. เสน่ห์ สายวงศ์ และอวกาศ แก้วเสน่ห์ , ปริญญานิพนธ์เรื่องโมเด็มที่ใช้คลื่นวิทยุ , กรุงเทพฯ : 1991
4. กฤษณ์ คงพัฒนะโยธิน และ กัลยา ปุรสาชิต ,การลดขนาดข้อมูล , กรุงเทพฯ : 1993
5. ไพศาล สงวนหมู่ และยืน ภู่วรวรรณ ,การสื่อสารข้อมูลและไมโครคอมพิวเตอร์เนทเวอร์ค กรุงเทพฯ : 1989
6. Herbert Schildt , The Art of C Elegant Programming Solutions , McGraw-Hill , New York 1991
7. Kent Porter , Stretching Turbo C , McGraw-Hill , New York 1990

กิตติกรรมประกาศ

ในการทำงานใด ๆ ก็ตามจะทำงานเพียงคนใดคนหนึ่งหรือกลุ่มใดกลุ่มหนึ่งย่อมจะไม่เกิดผลสำเร็จแน่นอน สำหรับวิทยานิพนธ์ฉบับที่ได้จัดทำขึ้นนี้ก็เช่นเดียวกันสามารถสำเร็จลุล่วงไปได้ด้วยดีจากความร่วมมือของหลายฝ่าย ท้ายที่สุดนี้ทางคณะผู้จัดทำขอขอบคุณอาจารย์เกรียงไกร วงศ์โรจนภรณ์ซึ่งเป็นอาจารย์ที่ปรึกษาที่คอยให้คำแนะนำ ข้อมูล ตลอดจนอุปกรณ์ต่าง ๆ ขอขอบคุณอาจารย์สุวิพล สิทธิชีวภาคที่ได้ให้คำแนะนำ และอำนวยความสะดวกต่าง ๆ ด้วยดีตลอดมา ขอขอบคุณอาจารย์สมยศ จุณณะปิยะที่ได้ให้คำแนะนำต่าง ๆ รวมทั้งอุปกรณ์ต่าง ๆ ขอขอบคุณอาจารย์นิภา ลีลารุจิ ที่ได้ให้คำปรึกษาแก้ไขข้อบกพร่องต่าง ๆ ขอขอบคุณอาจารย์โมไนย ไกรฤกษ์ที่ได้ให้ยืมเครื่องมือเครื่องใช้ต่าง ๆ ขอขอบคุณอาจารย์ประภากร สุวรรณะที่ให้คำแนะนำปรึกษาต่าง ๆ รวมทั้งเครื่องมือเครื่องใช้ต่าง ๆ ตลอดจนสถานที่ที่ทำให้โครงงานนี้สำเร็จไปได้ด้วยดี และท้ายที่สุดที่จะขาดเสียมิได้ต้องขอขอบคุณคุณธวัชชัย วิมลพิทยารัตน์ที่ได้ช่วยแก้ไขข้อบกพร่องรวมทั้งชิ้นส่วนอุปกรณ์ต่าง ๆ ที่ทำให้วิทยานิพนธ์นี้สำเร็จไปได้ด้วยดี