เครื่องตรวจสอบระบบสัญญาณแฟ็กซ์กรุ๊ปสาม

G.3 Facsimile Protocol Monitor

โดย

นายไพโรจน์	แซ่ลิ่ม	รหัส 36013025
นายวรพจน์	จันทร์ผ่องแสง	รหัส 36013028
นายสมรักษ์	เพชรชาตรี	รหัส 36013037

อาจารย์ที่ปรึกษา

ผศ. สมยศ จุณณะปิยะ

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

ปริญญานิพนธ์ปีการศึกษา 2538

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องตรวจสอบระบบสัญญาณแฟ็กซ์กรุ๊ปสาม

Facsimile Protocol Monitor (G.3)

ผู้จัดทำ

1. นาย ไพโรจน์ แซ่ลิ่ม รหัส 36013025
2. นาย วรพจน์ จันทร์ผ่องแสง รหัส 36013028
3. นาย สมรักษ์ เพชรชาตรี รหัส 36013037

..... อาจารย์ที่ปรึกษา

(ผศ. สมยศ จุณณะปิยะ)

เครื่องตรวจสอบระบบสัญญาณแฟ็กซ์กรุ๊ปสาม

G.3 Facsimile Protocol Monitor

โดย นาย ไพโรจน์ แซ่ลิ่ม รหัส 36013025

นาย วรพจน์ จันทร์ผ่องแสง รหัส 36013028

นาย สมรักษ์ เพชรชาตรี รหัส 36013037

อาจารย์ที่ปรึกษา ผศ. สมยศ จุณณะปิยะ

บทคัดย่อ

ระบบสัญญาณของแฟ็กซ์(Facsimile Protocol) ยังเป็นที่รู้จักกันเฉพาะในกลุ่มผู้ผลิตเครื่องแฟ็กซ์และเครื่องแฟ็กซ์ในยุคแรกมีระบบการจับสัญญาณที่ยังไม่เป็นมาตราฐานเดียวกันทั่วโลก เช่น G.1 ,G.2 และยังเป็น Analog Facsimile ซึ่งมีข้อเสียในการรับส่งที่จะต้องได้รับการปรับปรุงอยู่มาก ต่อมาแฟ็กซ์ G. 3 ซึ่งเป็น Digital Facsimile เครื่องแฟ็กซ์ต่างบริษัทกันสามารถรับส่งกันได้ โดยใช้ protocol ที่เป็น HDLC เดียวกันเครื่องตรวจสอบระบบสัญญาณนี้จะแสดงขั้นตอนการรับส่งแฟ็กซ์ว่ามี protocol การรับส่งอย่างไรบ้างในระหว่างการ Handshake และแสดงให้เห็นข้อมูลต่างๆในแต่ละฟิลด์ของแฟรม HDLC

ABSTRACT

The signal system of facsimile protocol is well-known in the group of producers. The first time ,the signal system of facsimile protocol is not the same standard in the world ,for example, G.1,G.2 , and they are still the analog facsimile. They should be improved some worst transfer point. The fax G. 3 is the digital facsimile and can send the message to other companies. This way can be done by protocol that is the same as HDLC. The fascimile protocol monitor shows the steps of protocol signal system for transferring between the fascimile machines (Handshake),and data field of HDLC frame.

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

ถึงแม้แฟ็กซ์ จะมีใช้งานมานานตั้งแต่ปี 1843 แต่เครื่องแฟ็กซ์ ก็ไม่เป็นที่แพร่หลายจนกระทั่งปี 1970 และจนเมื่อในปี 1980 (CCITT) The International Telegraph and Telephone Consultative Committee ได้กำหนดมาตรฐาน fax group 3 แบบ digital ขึ้นมา ต่อมาทำให้มีหลายบริษัทได้ผลิตเครื่องแฟ็กซ์ ออกจำหน่าย และใช้ในกิจการธุรกิจต่าง ๆ จนกระทั่งกลางปี 1980 เป็นจุดที่ เครื่องแฟ็กซ์ เป็นที่แพร่หลายอย่างเต็มที่ การทำงาน ของเครื่องแฟ็กซ์ โดยคราว ๆ แล้ว เครื่องแฟ็กซ์ จะทำการอ่านหรือ "scan" หน้ากระดาษที่ต้องการ จะส่งเพื่อแปรสัญญาณทางไฟฟ้า ซึ่งแทนตัวข้อมูลบนกระดาษ ตัวอ่านจะทำการอ่านที่ตรงมุมบนด้านซ้ายเป็นอันดับแรก เครื่องจะอ่านจุดดำบนตัวอักษรหรือข้อความในกระดาษโดยจุดดำและขาวบนกระดาษจะถูก scan อ่านเป็นจุดประมาณ 1,728 จุดต่อ 1 เส้น scan และจะ scan ในบรรทัดต่อ ๆ ไปโดยจะใช้จำนวนเส้น scan ประมาณ 10-20 เส้น scan ต่อ 1 เส้นบรรทัดตัวอักษร ในหนึ่งหน้ากระดาษจะมีจำนวนเส้น scan ทั้งหมด ประมาณ 2,200 เส้นscan เมื่อเป็น mode การ scan อย่างละเอียด การแปลข้อมูลเส้น scan เป็นสัญญาณทางไฟฟ้าจะใช้อุปกรณ์อีเล็กทรอนิกส์ที่มีความไวต่อแสง ("CCD" Charge Coupled Device Silicon Ship) โดยให้ขนาด pulse สูงเมื่อเป็น white dot และ เป็น pulse ต่ำเมื่อเป็น black dot สัญญาณที่ image มานี้จะผ่านขบวนการ compress เพื่อทำให้การส่งผ่านไปปลายทางเร็วขึ้นเมื่อรวมกับการเข้า code ที่กระทำโดย modem สัญญาณที่เป็นลักษณะ tone แบบ analog ของแฟ็กซ์ G3 จะส่งผ่านไปในคู่สายโทรศัพท์ด้วย mode digital ที่ความเร็วสูง และทำงานแบบ synchronous ทางด้านรับก็จะทำการ synchronize กับสัญลักษณ์ของสัญญาณที่รับมา ผลของการถอดรหัสจะถูกนำเก็บไว้ใน line memory ในเครื่อง fax ด้านรับตำแหน่งของจุดดำ ของเส้น scan ที่ด้านส่งส่งมาจะถูกนำเก็บไว้ในตำแหน่ง memory ที่สอดคล้องกัน ดังนั้นจุดดำแต่ละจุด จะถูกตำแหน่งเหมือนเดิม เวลาที่แต่ละ "dot" จะถูกพิมพ์จะทำภายหลังจากช่วงเวลารับเสร็จสิ้นแล้ว การวางจุดดำลงบนกระดาษจะเป็นสิ่งที่ยากมากก่อนหน้าที่ระบบ digital แฟ็กซ์ จะได้รับการพัฒนามาดังตัวอย่าง anlog fax (G.2 หรือก่อนนั้น) จะมี "spot" การ scan เพียงหนึ่งที่ที่แฟ็กซ์ด้านส่งและมี "spot" การ record 1 ที่ที่ด้านรับ เมื่อด้านส่งเริ่มส่งหน้ากระดาษ "spot" ตัวรับจะต้องหาตำแหน่งการ scan ของ spot โดยจะเริ่มที่ตรงจุดที่แน่นอน และในเวลาเดียวกันจะต้องวิ่งไปด้วยความเร็วที่แน่นอนเช่นกัน เพราะว่าแฟ็กซ์ ด้านรับไม่มีทางที่จะ synchronize กับสัญญาณแฟ็กซ์ ด้านส่งแฟ็กซ์ ในรุ่นดังกล่าวจึงต้องการ clock ที่มีความแน่นอนสูงมาก ๆ (frequency standard) ซึ่งก็เป็นเรื่องที่ยุ่งยากลำบากมาก

บทที่ 2

ทฤษฎีและหลักการ

มาตราฐานของแฟ็กซ์

CCITT (International Telegraph and Telephone Consultative Committee) ซึ่งเป็นองค์การที่สนับสนุนของผู้เชี่ยวชาญทางด้านเทคนิค เป็นหน่วยงานหนึ่งของ ("ITU" ; The International Telecommunication Union) และอยู่ภายใต้การคุ้มครอง หรือเป็นตัวแทนของ องค์การสหประชาชาติ "The United Nation"

CCITT จะศึกษาความต้องการทางด้านอุปกรณ์ทางการสื่อสารและบริการแบบใหม่ ๆ และปรับปรุง เปลี่ยนแปลงคำแนะนำซึ่งเสมือนเป็น standard สำหรับการติดต่อระหว่างชาติ ลักษณะการเป็นสมาชิก ทั้งที่เปิดให้ใช้ การให้บริการ บริษัทที่ทำการผลิต องค์การมาตรฐานระหว่างชาติและตัวแทนของรัฐบาล

ข้อมูลทางด้าน facsimile ถูกจัดหาจาก EIA/TIA TR-29 ซึ่งเป็นคณะกรรมการทางด้าน facsimile จากอเมริกา CCITT จะประกาศหรือตีพิมพ์เกี่ยวกับ ผลการพัฒนาทางด้านเทคนิคและวิธีการปฏิบัติ ที่มีประสิทธิภาพ โดยดูจากประสิทธิภาพที่มองเห็นจากประสิทธิภาพในด้านระบบการสื่อสารต่างๆ เพิ่มคุณประโยชน์และทำให้มันมีประโยชน์และใช้งานได้เป็นสาธารณะ เป็นความรับผิดชอบของ CCITT ที่จะพิจารณารายละเอียดของการทำงานทางด้านเทคนิครวมถึง "factor" ที่ต้องการต่างๆ เพื่อให้แน่ใจว่า สามารถทำงานรวมกันในระดับนานาชาติได้ ส่วนในเรื่องระบบทาง hardware , software เช่น ระบบการพิมพ์ ระบบการ feed กระดาษ, front ของตัวหนังสือ, ประเภทของกระดาษที่ใช้ เป็นส่วนที่อยู่ในข่ายการพิจารณาของ CCITT

CCITT จะมีการประชุมปรึกษากันทุก 4 ปี เพื่อปรับปรุงข้อแนะนำใหม่ ๆ "new recommudations" และจะจัดพิมพ์ในปีถัดไปเรียกว่า "Blue Bode" ส่วนมากตามมาตรฐานของ facsimile ที่ recommendation โดย CCITT จะอยู่ในส่วน Study Group (SG)

SGI : จะกล่าวถึงการ และคุณภาพของการบริการ
group นี้จะอธิบายถึงคำจำกัดความและการทำงานที่สังเกตุเห็นได้เวลาบริการทางด้าน "Telegraph และ telematic" เช่น facsimile, teletex และ Videotex โดยใช้ Series " F recommendation " ยกตัวอย่าง recommendation ที่เกี่ยวกับ fax เช่น F.160-F.353, F.600, F.601, F.710-F.730

SG VII : จะกล่าวถึง network ในการสื่อสารข้อมูล
group นี้จะใช้ Series "X recommendation" ตัวอย่าง recommendation ที่เกี่ยวกับ fax เช่น

x.1- x.32	เป็นส่วนกล่าวถึงการเชื่อมต่อ
x.40 - x.181	เป็นส่วนกล่าวถึงระบบการส่ง ลักษณะสัญญาณและการ switching
x.200 - x.219	จะกล่าวถึง mode และชี้ส่วนที่ใช้ใน OSI (Open System Interconnection)
x.220 - x.290	จะกล่าวถึง protocal ที่ใช้ใน OSI เกี่ยวกับทางด้าน facsimile
x.400 - x.420	เป็นส่วนที่ว่าถึงระบบการ "Handling" ข้อมูลข่าวสาร

SG VIII : จะกล่าวถึงส่วนของเครื่องมือและอุปกรณ์ตลอดจน protocal ที่ใช้ในบริการ facsimile recommendation มาตรฐานของ fax group 1, 2, 3 และ 4 สมบูรณ์เมื่อในปี 1984 ดังตารางที่ 1นี้

ตารางที่ 1

CCITT Facsimile Recommendation

Grupe	Recommendation	
1	T.2	Standrdization of Group 1 Facsimile Apparatus for Document Transmission
2	T.3	Standrdization of Group 2 Facsimile Apparatus for Document Transmission
3	T.4	Standrdization of Group 3 Facsimile Apparatus for Document Transmission
2,3	T.30	Proceduces for Document Facsimile Transmission in the General Switched Telephone Network
4	T.6	Facsimile Coding Schemes and Coding Control Function for Group 4 Facsimile Apparatus
4	T.60	Terminal Equipment for Use in the Teletex Service
4	T.61	Character Repertoire and Coded Character Sets for the International Teletex Service
4	T.62	Control Procedures for the Teletex and Group 4 Facsimile Service
4	T.62bis	Control Procedures for the Teletex and Group 4 Facsimile Service Based on Recommendations X.215/ X.225
4	T.70	Network-Independent Basic Transport Service for the Telematic Services
4	T.501	A Document Application Profile MM for Interchange of Formatted Mixed Mode Document
4	T.502	A Document Application Profile PM1 for Interchange of Processable Form Document
4	T.503	A Document Application Profile for Interchange of Group 4 Facsimile Document
4	T.521	Communication Application Profile BT0 for Document Bulk Transfer Based on Session Service(according to rules defined in Recommendation T.62bis)
4	T.522	Communication Application Profile BT1 for Document Bulk Transfer
4	T.561	Terminal Characteristics for Mixed Mode of Operation MM
4	T.562	Terminal Characteristics for Teletex Processable Mode of Operation PM.1
4	T.563	Terminal Characteristics for Group 4 Facsimile Apparatus
4	T.400	Introduction to Document Architecture ,Transfer and Manipulation
	Open Document	Architectures (ODA) and Interchange Format:
4	T.411	Introduction and General Principles
4	T.412	Document Structures
4	T.414	Document Profile
4	T.415	Open Document Interchange Format (ODIF)
4	T.416	Character Content Architectures
4	T.417	Raster Graphics Content Architectures
	Document	Transfer and Manipulation (DTAM) Service and Protocal:
4	T.431	Introduction and General Principle
4	T.432	Service Definition
4	T.433	Protocal Specification
4	T.441	Document Transfer and Manipulation (DTAM) —Operational Structure

group 1 และ 2 ได้ถูกแยกออกไปเป็นเวลามากกว่า 5 ปีแล้ว บางส่วนของ fax Group 2 ยังคงมีให้บริการอยู่ แต่คุณภาพการ "copy' ไม่ค่อยดี และใช้ระยะเวลานานในการส่ง fax และมีราคาแพงเมื่อเทียบกับ fax Group 3 การ

ทำงานของ Group 3 ถูกพัฒนาอย่างเต็มที่ ตั้งแต่ปี 1980 ตั้งแต่เริ่มประกาศ standard ส่วนที่มีการปรับปรุงต่อๆ ไป จะเป็น standard ของส่วนย่อย ๆ เช่น การปรับปรุง ความละเอียด, ความเร็ว และการเชื่อมต่อทาง digital ในตอนนี้ 99.7% จะเป็นการใช้งาน fax Group 3 ส่วน fax Group 4 มาตรฐาน ยังอยู่ภายใต้การเปลี่ยนแปลงดังนั้น fax Group 4 จะมีการใช้งานที่จำกัด เพราะมันถูกออกแบบให้ทำงานบน network digital ซึ่งใช้เวลาหลายปีที่จะทำให้ ISDN เป็น network แบบ worldwide ได้แบบ (network โทรศัพท์) PSTN เช่น ปัจจุบันนี้

SG XVIII : จะกล่าวถึงส่วนการสื่อสารข้อมูลบน network โทรศัพท์ของ facsimile
ส่วนที่น่าสนใจเป็นพิเศษ คือ modem series V ที่มีความเร็วสูง modem แบบ Single Carrier ในปัจจุบันมี คุณประโยชน์สูงกว่าแบบ multicarrier และ modem บางตัวยังทำหน้าที่เป็น fax ได้อีกด้วย หรืออีกนัยหนึ่ง ภายในตัวเครื่อง facsimile จะมี modem อยู่ภายในนั้นเอง ซึ่ง CCITT ได้รวมเอากลุ่มพิเศษเทคนิคทาง modem เข้ารวมกลุ่ม SG VIII ซึ่งเป็น recommendation ทางด้าน fax นอกจากนี้ในกลุ่ม "Telematic Services" เช่น บริการ Teleten, Telefax, Datafax, Videoten, Bureaufax มีความเป็นไปได้ที่จะใช้ modem เป็นส่วนประกอบในด้านการจัดส่งสัญญาณได้ทั้งหมด

BFICC : British Facimile Cansultative Committee

BFICC มิได้ประกาศออกมาเป็นมาตรฐานระหว่างชาติทางด้าน facsimile แต่อย่างไรก็ตาม BFICC ก็เป็นองค์กรที่ช่วยเป็นผู้นำ CCITT อย่างมากทางด้าน facsimile แนวความคิดของ BFICC, CCITT ได้นำมาเป็นมาตรฐาน ตัวอย่างเช่น ระบบ (ECM ; error-correcting mode) ซึ่งถูกพัฒนาและทดสอบโดย BFICC ซึ่ง CCITT ก็นำมาเป็น facility ส่วนหนึ่งในการรับส่ง fax

CIAJ : Communications Industry Association of Japan

บริษัทที่ผลิตเครื่อง fax ในญี่ปุ่นได้รวมตัวกันจัดองค์กรนี้ขึ้นมา CIAJ ก็มีบทบาทต่อ CCITT มาก ในด้านเทคนิคใหม่ ๆ และมาตรฐานของ CCITT บางส่วนก็นำมาจาก CIAJ วิธีการ test ความเข้ากันได้ของ fax Group 3 ในช่วงต้น ๆ ซึ่งมาจากโรงงานผลิตที่ต่างกัน ช่วยแก้ปัญหาอย่างมากในช่วง fax Group 3 เริ่มใช้งาน

EIA/TIA TR-29 Facimile Systems and Equipment

EIA ย่อมาจาก The Electronic Industries Association ซึ่งเป็นองค์กรทางด้านการค้าทางด้านอุตสาหกรรมทางด้านอีเล็กทรอนิกส์ (electronics)

TIA ย่อมาจาก The Telecommunication Industrial ซึ่งเป็นส่วนหนึ่งของ EIA ที่จัดการทางด้าน telecommunication

TR-29, Facimile Equipment and Systems ถูกจัดตั้งขึ้นมาในปี 1960 และได้จัดทำ U.S. fax Standard ในปี 1966 ซึ่งในเวลานั้น โรงงานอุตสาหกรรมของเครื่อง fax ส่วนมากอยู่ในอเมริกา แต่ไม่มีการจัดตั้งเป็นมาตรฐานขึ้น และในปี 1976, TR-29 กลายมาเป็น กลุ่ม fax เทคนิคของอเมริกา ซึ่งเป็นส่วนหนึ่งใน CCITT TR-29 เป็นแนวหน้าในการกำหนดมาตรฐาน fax Group 3 และ Group 4 มันถูกใช้เป็นมาตรฐานในอเมริกา (U.S.national Standards) และดี

พิมพ์มาเป็นมาตรฐานก่อนหน้าที่ CCITT จะจัดทำมาตรฐานนี้ออกมา ดังนั้นเพื่อหลีกเลี่ยงความสับสนในมาตรฐานระหว่างชาติ (ISO ; The International Standard Organization) ได้พยายามที่จะทำมาตรฐานเบื้องต้น แต่นี้ก็ไม่ใช่ปัญหาอีกต่อไป หากระบุมาตรฐาน EIA ก่อนแล้วกำกับด้วยมาตรฐาน CCITT เช่น

EIA-465 ซึ่งอธิบายถึงกลวิธีในการส่งสัญญาณของเครื่อง fax ซึ่งก็คือ (CCITT T.4)

EIA-466 จะอธิบายถึงลำดับขั้นตอนในการส่งเอกสารผ่าน facsimile ซึ่งก็คือ (CCITT T.30)

EIA-578 และ EIA-592 จะอธิบายถึงการ Control facsimile (มองเป็น DCE) แบบ asynchronous ซึ่งยังไม่มีมาตราฐานในส่วนนี้ของ CCITT

U.S. Department of Defense (DOD)

เป็นมาตรฐานทางทหาร MIL-STD-188-161D ซึ่งจะอธิบายถึงมาตรฐานการทำงานภายใน และการกระทำการ ของ Digital facsimile รวมถึงเครื่องมืออุปกรณ์ของ Digital facsimile เพื่อการติดต่อกันในระยะใกล้และไกล แบบมีระบบการโจรกรรม ระบบนี้จะส่งเอกสารด้วยสัญญาณทาง digital แบบ bit ต่อ bit ในระบบ asychronous โดยใช้ clock แบบ external (คือสร้าง clock 100 "clock recovery") โดยความเร็ว 2,400, 4,800 และ 9,600 Bit/Sec การจับโจรกรรมจะใช้วิธี เข้ารหัสข้อมูล (encryption) โดยแบ่งเป็นกลุ่มการจัดส่งดังนี้

- Type I Mode ใช้ส่งข้อมูลแบบขาวดำ รวมทั้งรูปภาพ
- Type II Mode เป็นแบบที่ใช้กับเอกสารที่มี tone หรือระดับขาวดำที่แตกต่างกันอย่างต่อเนื่อง เช่น รูปถ่าย

ISO : International Standard Organization

ISO มิใช่หน่วยงานทางราชการ แต่เป็นองค์กรซึ่งจัดเตรียมมาตรฐาน สำหรับในงานอุตสาหกรรม และการค้า ISO เป็นส่วนหนึ่งของ CCITT ประมาณ 400 หน่วยงานที่เกี่ยวข้องกับ ISO, มาตรฐาน ISO จะครอบคลุมในหลาย ๆ ด้านรวมทั้ง fax ส่วนที่เกี่ยวข้องได้แก่ มาตรฐานขนาดของกระดาษ และระบบการประมวลผลข้อมูล (รวมถึงการสื่อสัญญาณข้อมูล) แม้ทางด้านการโทรคมนาคม ส่วนใหญ่ถูกออกแบบตามแผนซึ่งกำหนดโดย ITU/CCITT/CCIR แต่เทคโนโลยีโดยรวมทำให้มาตรฐานเทคโนโลยี มาใช้ร่วมกันได้

ตัวอย่างเช่น ISO ได้พัฒนาระบบ OSI : Open System Interconnection Model สำหรับการติดต่อสื่อสารระหว่าง computer แต่ CCITT ก็มีระบบของตัวเอง และแยกเป็นมาตรฐานออกมา ในตอนนี้ภาระหน้าที่ fax จะถูกรวมเข้าเป็นหน้าที่โดยทั่วไปของ PC, OSI model มีความสำคัญต่อ fax ในปัจจุบัน และจะเพิ่มความสำคัญขึ้นอีกในอนาคต อีกตัวอย่างคือ ISO/IEC/CCITT WG8 จะว่าด้วยการประมวลผลภาพ และบีบอัดข้อมูล และเข้ารหัสสี ซึ่งคาดหวังว่าจะนำไปใช้ในระบบ fax สี ปัจจุบัน fax สีได้มีปรากฏใช้งานในญี่ปุ่น

Group 3 Facsimile

มาตรฐาน fax group 3 อยู่ภายใต้คำแนะนำ (Recommendation) T.4 และ T.30 ใน CCITT Blue Bode ฉบับสมบูรณ์ในปี 1988 แต่ recommendations ที่ใช้อยู่ในขณะนี้ได้เพิ่มมาตรฐาน V.17 14.4 kb/s และการบีบอัดข้อมูลแบบ MMR (Modefied Modified Read) และมีโหมดแก้ไขความผิดพลาด ECM (Error-Correction mode) เพราะมีการ

ประชุม CCITTทุกๆ 4 ปี ส่วนที่เพิ่มเข้ามาเพื่อใช้ประโยชน์แฟกซ์ G.3 มาตรฐานได้มากขึ้น ส่วนที่เพิ่มขึ้นมานี้ไม่เหมือนกับมาตรฐานแฟกซ์ G.3 เริ่มต้นเลย ดังตารางที่ 2

ตารางที่ 2

CCITT Group 3 Facsimile Specifications

Item	Standard	Option		Small Copy (A5&A6)			
Scan width							
in	8.46	10	11.9	4.2	5.9	5.9	4.2
mm	215	255	303	107	151	151	107
Pels/line	1728	2048	2432	864	1216	1728	1728
H/in	203			203	203	290	406
/mm	8			8	8	11.4	16
V/in	97.8	1.96		196/392	138/176	138/176	196/392
/mm	3.85	7.7		7.7/154	5.44/10.9	5.44/10.9	7.7/15.4
Ms/line	20	0,5,10,40					
Coding	Modified Huffman	Modified Read ,Modified -Modified Read					
Modem							
Fax signal	V.27ter	V.29		V.17			
Bits/s	2400/4800	9600/7200		14400,1200,9600,7200			
Handshake	V.21(Ch 2)	V.27ter					
Bits/s	300	2400					
Error correction	None	Error Correction Mode					

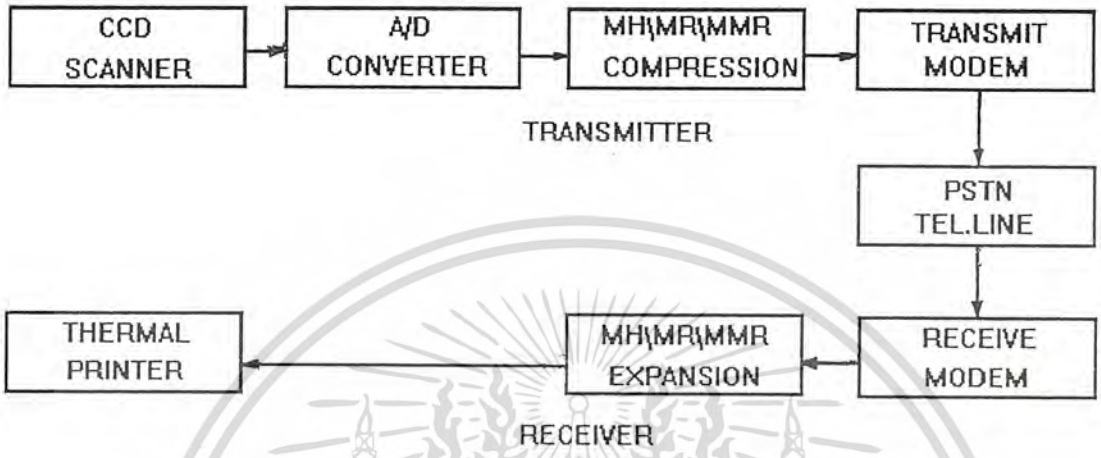
ประวัติความสำเร็จของ fax G.3

ความสำเร็จของ facsimile แบบ digital group 3 นี้ เป็นผลมาจากความร่วมมือกันของระหว่างชาติในการกำหนดมาตรฐานอันที่ชาญฉลาด เพื่อที่จะเพิ่มสิ่งที่ต้อการที่ดีๆและเป็นลักษณะเฉพาะ โลกของ fax ได้ถูกพัฒนาจาก fax Group 2 (Analog facsimile) ไปอย่างสิ้นเชิง เป็น fax Group 3 (Digital facsimile) ในปี 1980 เดิมที่ modem V.29 ถูกออกแบบมาใช้งานที่ความเร็วสูงบนสายตรง (Leased line) แต่บริษัท Bell System ได้ให้คำแนะนำว่า modem V.29 นี้ไม่สามารถใช้งานในคู่สายโทรศัพท์ (PSTN : Public Switching Telephone Network) ที่ความเร็วสูงกว่า 4.8 kb/s อย่างไรก็ตาม fax แบบ digital ประสบความสำเร็จในการใช้ modem V.29 ในโหมด half duplex บนชุมสายโทรศัพท์ (PSTN) ที่ความเร็ว 9.6 kb/s และมาตรฐาน V.29 ได้เพิ่มเอา option ของ fax G.3 Standard ใส่ไว้ด้วย สิ่งหนึ่งซึ่งสำคัญมากในมาตรฐาน fax G.3 คือ ความยืดหยุ่นในการใช้งานของมันซึ่งยอมให้บริษัทผู้ผลิตต่างๆ ได้เพิ่มรูปแบบ (feature) ในการรับส่งพิเศษเฉพาะตัวเข้าไปโดยยังเข้ากันได้กับมาตรฐาน fax G.3 รูปแบบที่เพิ่มให้เลือกมีดังนี้

- Acceptably High Resolution มาตรฐานความละเอียดของภาพเป็น 203เส้นต่อนิ้ว ทางแนวนอนและ 98 เส้นต่อนิ้ว ทางแนวตั้งให้คุณภาพการสำเนาสามารถอ่านได้อย่างดีทีเดียว สำหรับเอกสารที่เป็นข้อความโดยส่วนมาก และใช้ 196 เส้นต่อนิ้วเพื่อเพิ่มความชัดเจนในการอ่านสำหรับข้อความที่ตัวหนังสือมีขนาดเล็ก ความละเอียดในทางเลือกสองอย่างนี้ไม่มีใน fax G.1, G.2 ซึ่งเอกสารที่ได้สามารถอ่านได้ แต่เป็นสำเนาที่คลุมเครือ ความละเอียดมาตรฐานเพียงพอ สำหรับตัวหนังสือ, หัวข้อเรื่อง, จดหมายธุรกิจ ส่วนสิ่งตีพิมพ์ต่าง ๆ เช่น หนังสือ, นิตยสาร และหนังสือพิมพ์ จะต้องใช้ความละเอียดที่มากกว่านี้ โดยการบีบอัดข้อมูล แบบ MR และ MMR ทำให้มีประสิทธิภาพดีกว่าที่จะส่งโดยตรงๆ fax G.3 โดยมากจะมีความสามารถที่จะเลือก โหมดความละเอียดสูง
- Page Per Minute Speed เครื่อง fax G.3 จะใช้เวลาในการส่งเอกสาร 2-7 แผ่นในเวลา 1 นาที ภายหลัง 15 วินาทีของการส่งแผ่นแรกและ hadshake สัญญาณกับแผ่นที่ส่งและแผ่นอื่น ๆ ที่ตามมาไม่ต้องการการ handshake อีกต่อไป เวลาใช้ในการส่งขึ้นอยู่กับจำนวนของจุดดำ และความละเอียดของเอกสาร fax G.1 ใช้เวลาประมาณ 6.5 นาทีต่อการส่งเอกสาร 1 ทุก ๆ แผ่น แต่ทุก ๆ แผ่นที่ส่งใน G.1 ทำการส่งด้วยมือต้องบวกเวลาเพิ่มเข้าไปประมาณ 1-7 นาทีต่อแผ่น ส่วน fax G.2 เป็นแบบส่งอัตโนมัติ และใช้เวลาประมาณ 3.5 นาที ต่อการส่ง 1 หน้า คือเร็วเป็น 2 เท่าของ G.1
- User - Friendly Operation เครื่อง fax G.3 มีการใช้งานที่ง่าย อาจกล่าวได้ว่าง่ายกว่าการใช้ เครื่องถ่ายเอกสารเสียอีก การรับ fax ก็ไม่จำเป็นต้องมีคนไปจัดการ ช่วง standby จะกิน power ต่ำ และจะตอบรับอัตโนมัติเมื่อมีการเรียกเข้ามา เมื่อมันรับ fax เสร็จจะ hang up line และปล่อยให้คู่สายพร้อมที่จะได้รับการเรียกครั้งต่อไป fax G.3 ส่วนมากจะมีการป้อนกระดาษแบบอัตโนมัติ และสามารถป้อนกระดาษได้หลายๆ หน้าทีเดียว fax G.3 เมื่อเริ่มการรับส่งจะพยายาม handshake กันที่ความเร็วสูงที่สุดเท่าที่ ทำได้ก่อน หากใช้งานที่ความเร็วสูงไม่ได้ก็จะลดความเร็วลงมาจนกว่าเป็นที่พอใจทั้งสองฝ่าย
- Universal Compatibility เครื่อง fax G.3 สามารถใช้งานเข้ากันได้ คือ แม้จะต่างบริษัทผลิตกัน คล้าย ๆ กับการใช้งานรวมกันได้กับเครื่องโทรศัพท์ที่ต่างยี่ห้อกัน และไม่จำเป็นว่า fax ปลายทางทำสำเนาโดยใช้ laser copy, thermal recording หรือวิธีอื่น ๆ
- Use of Regular Telephone Lines (PSTN) ระบบชุมสายโทรศัพท์สามารถที่จะใช้งานเข้ากันได้ทั่วโลก ดังนั้นเครื่อง fax ทุกยี่ห้อก็สามารถเรียกถึงกันได้ หากไม่ได้ผ่านชุมสายโทรศัพท์ (PSTN) ก็สามารถใช้ผ่าน VOICE channel หรือ digital channel ก็ได้ แต่ทางเลือกเหล่านี้ ค่อนข้างมีขีดจำกัด และการต่อคู่สายเข้ากับ fax ก็เป็นเรื่องยากและมีราคาค่าใช้จ่ายแพง
- Step-Down Modem เนื่องจากในเครื่อง fax ทุกตัวจะใช้การรับส่งมาตรฐานเดียวกัน modem หรืออาจกล่าวได้ว่าในเครื่อง fax ก็ประกอบไปด้วยตัว modem ซึ่งเวลารับส่ง modem จะทำการตรวจสอบ คุณสมบัติของคู่สายโทรศัพท์อย่างอัตโนมัติ และเลือกความเร็วที่สูงที่สุดที่สามารถทำได้ และยังมีส่วนของวงจร equalization เพื่อลด distortion บนคู่สายโทรศัพท์ เพื่อทำให้สามารถใช้งานที่ความเร็วสูงสุดได้

สถาปัตยกรรมของ G.3

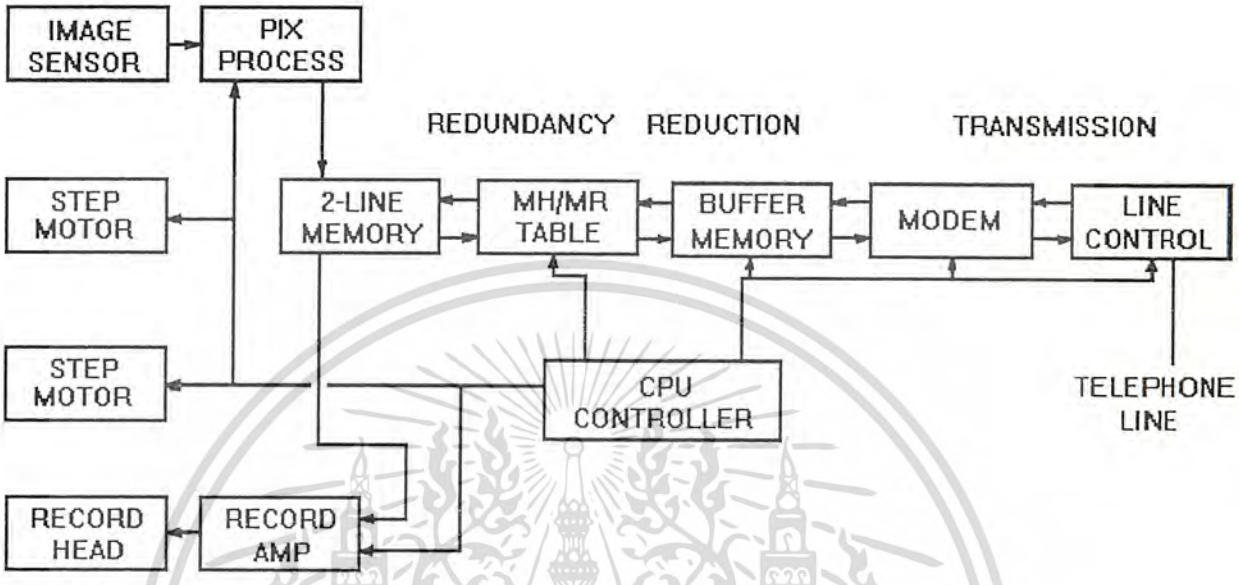
เพื่อให้เข้าใจสถาปัตยกรรมของ fax G.3 จะอธิบายด้วย Block diagram สำหรับหน้าที่ของ ระบบส่งและระบบรับ



รูปที่ 1 Group 3 block diagram

ทางด้านส่งของเครื่อง fax จะมีตัว scanner ซึ่งเป็นชนิด "CCD" ซึ่งมันจะจำลองหน้ากระดาษที่จะทำการส่งโดย Focus จุดเหล่านั้นลงบน CCD ต่อการ scan 1 เส้นหน้ากระดาษใช้ 1,728 จุด CCD จะทำการอ่านความสว่าง ของจุดโดยทุกระยะความสูง 0.01 นิ้ว หรือ (0.254 mm) และ จะกำเนิด pulse ให้แทนจุดแต่ละจุด 1728 pulse ต่อเส้น scan ขนาด amplitude ของ pulse จะแทนความสว่างของจุด ๆ นั้น เมื่อเสร็จไป 1 เส้น ก็จะทำในเส้นถัดไป วงจร A/D Converter จะเปลี่ยนจากสัญญาณ Analog เป็น digital ซึ่งหนึ่งจุดจะแทน 1 bit หน่วยความจำ two-line จะเก็บรายละเอียดของจุดในเส้น scan ของเส้น scan ที่ถัดกับ ขบวนการบีบอัดข้อมูลและเข้ารหัสจะใช้เทคนิค Modified Hrffman (MH), Modified Read (MR) และ Modified Modified Read (MMR) เรียกขบวนการนี้ว่า Source acding หรือ redundancy reduction หน่วยความจำ buffer จะเก็บผลของ code MH/MR/MMR ส่งต่อให้ modern การเข้ารหัสจะใช้วิธีการ เปลี่ยนเป็น codeword ซึ่งจะบรรจุไปด้วยข้อมูลของภาพ ซึ่งจะย่อข้อมูลได้ 1/5 ถึง 1/20 เท่าของจำนวน bit ที่เข้าแต่ละชุด ในตัวของ modem จะมีขบวนการบีบอัดข้อมูลทาง digital อีกที แล้วเปลี่ยนเป็นสัญญาณ analog แล้วส่งผ่าน line โทรศัพท์ หาก modem ติดต่อที่ 14.4 kb/s modem จะใช้ 6 bit ของข้อมูลเพื่อแทนความแตกต่าง 1 สถานะใน 128 สถานะ ความแตกต่างของการ modulate แบบ PAM (Pulse Amplitude modulation) ของสัญญาณ analog เนื่องจากคู่สายโทรศัพท์จะรับอัตราในการ modulate ได้เพียง 2400 baud เพื่อจะให้บรรจุจำนวน 14.4 kbit ในเวลา 1 วินาที modem จะต้อง

มีการเข้ารหัส 6:1 ตามมาตรฐาน FCC, CSA และ PTT จะสร้างหน่วย LCU (Built-in line Connection Unit) เพื่อให้ connect RJ-II หรือมาตรฐานอื่นต่อเข้า PSTN



รูปที่ 2 Group 3 apparatus block diagram

modem ทางด้านรับจะทำการถอดรหัสสัญญาณ analog ที่ได้รับมาสร้างเป็นสัญญาณทาง digital และทำการขยาย ส่วนที่มีการเข้ารหัส MH/MR/MMR เพื่อแทนข้อมูลของจุดขาวดำ เครื่องพิมพ์ที่ใช้หัวพิมพ์ความร้อนจะมีความละเอียด 203 จุด/นิ้ว พิมพ์ลงบนกระดาษความร้อนจะทำให้เกิดจุด ส่วนที่ไม่ mark จะเป็นสีขาว fax G.3 เกือบทั้งหมดจะส่งหรือรับคนละเวลากัน (ระบบ half duplex) ดังนั้นอุปกรณ์ทาง mechanic และวงจรจะถูกใช้งานตอนรับหรือตอนส่ง ตอนใดตอนหนึ่งเท่านั้น เช่นกัน LCU (Line Control Unit), modem, หน่วยความจำ buffer, ต่อเข้ารหัส และ line-memory จะทำงานในโหมด half-duplex

ขนาดหน้ากระดาษที่แตกต่างกันเล็กน้อย ในตัวหนังสือในอเมริกาใช้ 8.5x11นิ้ว (216x279 mm) ส่วนในขนาด ISO ใช้ A4, 210x297 mm (8.27x11.69 นิ้ว) และเครื่อง fax ส่วนมากจะถูกออกแบบมา ใช้กับกระดาษ 2 ขนาดนี้

<u>การเข้ารหัสของ fax Group 3</u>

เมื่อกำเนิด fax group 3 ขึ้นได้มีการพัฒนาการเข้ารหัสเพื่อจะแยกและจัดประเภทของข้อมูลที่มากมาย ตัวอย่างเช่น บริเวณส่วนที่เป็นกระดาษขาวใน เอกสารที่จะส่งซึ่งเป็นข้อมูลลักษณะซ้ำ ๆ กันและมีจำนวนมาก การเข้ารหัสจึงมีความจำเป็นในการลดเวลาการส่ง ในครั้งแรกการเข้ารหัสแบบ Huffman ถูกนำมาใช้ เป็นลักษณะการเข้า code ทีละเส้น scan แต่ละเส้น scan จะประกอบด้วย code word จำนวนหนึ่ง แต่ก็มีข้อบกพร่องตรงที่ ใน 1 เส้น scan จะมี 1728 จุดจะต้องมี code word สำหรับ white run 1728 แบบ และ code word สำหรับ block run

1728 แบบ ซึ่งมากมายมหาศาล ในการที่จะรองรับความเป็นไปได้ ของจุดบนเส้น scan 1 เส้น จึงเกิดการพัฒนาระบบการเข้ารหัสแบบใหม่ ๆ

- Modified Huffman Code (MH)

ภายใต้ recommendation ของ CCITT T.4 ได้ระบุว่าเครื่อง fax จะต้องมีการเข้ารหัสแบบ Modified Huffman run-length coding โดยในเส้น scan จะเริ่มจากจุดขาว ซึ่งเป็นที่ว่างบนกระดาษ ก่อนที่จะเจอจุดดำ ซึ่งเป็นส่วนของข้อมูล จุดสีขาวนี้ใน file ชนิด text file จะมองเป็น string ตัวหนึ่งและมีศัพท์ในเรื่อง fax ว่า "RUN" ซึ่งอาจจะเป็น run ตลอดเส้น scan ก็ได้กรณีเป็นบรรทัดที่ไม่มีตัวหนังสือ

ใน 1 เส้น scan จะประกอบไปด้วย code word ที่เปลี่ยนแปลงไปตามความยาวของเส้น scan เพื่อแทนข้อมูลที่เป็นจุดขาวและดำ ในความยาว 1 เส้น scan 215 m.m. จะใช้จำนวนจุดทั้งหมด 1728 จุด ส่วนของ run length (ขาวหรือดำ) ที่เกิน 64 จุด เป็นหรือมากกว่าเป็นจำนวนเท่าของ 64 จะมีรหัสอีกส่วนหนึ่ง เรียกว่า "make-up code" ส่วน run length (ขาวหรือดำ) ที่อยู่ในช่วง (0-63) จุด รหัสจะเป็นส่วนของ "terminate code" ด้วยวิธีการนี้จะมีเพียง 92 binary code เท่านั้น ก็สามารถครอบคลุม ความน่าจะเป็นของรูปแบบจุดในเส้น scan 1728 จุด ด้วยวิธีการนี้แทนที่จะส่งเส้น scan ขาวในหน้ากระดาษด้วย 1728 จุด MH code จะส่งเพียง 9 bit code word เท่านั้น

ตารางที่3

แสดงการเข้ารหัสของ Modified Huffman Terminate

White run length	Code word	Back run length	Code word
0	00110101	0	0000110111
1	000111	1	01
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111

ตารางที่3 (ต่อ)

White run length	Code word	Back run length	Code word
15	110101	15	000011000
16	101010	16	0000010111
17	101011	17	0000011000
18	0100111	18	0000001000
19	0001100	19	00001100111
20	0001000	20	00001101000
21	0010111	21	00001101100
22	0000011	22	00000110111
23	0000100	23	00000101000
24	0101000	24	00000010111
25	0101011	25	00000011000
26	0010011	26	000011001010
27	0100100	27	000011001011
28	0011000	28	000011001100
29	00000010	29	000011001101
30	00000011	30	000001101000
31	00011010	31	000001101001
32	00011011	32	000001101010
33	00010010	33	000001101011
34	00010011	34	000011010010
35	00010100	35	000011010011
36	00010101	36	000011010100
37	00010110	37	000011010101
38	00010111	38	000011010110
39	00101000	39	000011010111
40	00101001	40	000001101100
41	00101010	41	000001101101
42	000101011	42	000011011010
43	00101100	43	000011011011
44	00101101	44	000001010100

ตารางที่3 (ต่อ)

White run length	Code word	Back run length	Code word
45	00000100	45	000001010101
46	00000101	46	000001010110
47	00001010	47	000001010111
48	00001011	48	000001100100
49	01010010	49	000001100101
50	01010011	50	000001010010
51	01010100	51	000001010011
52	01010101	52	000000100100
53	00100100	53	000000110111
54	00100101	54	000000111000
55	01011000	55	000000100111
56	010111001	56	000000101000
57	01011010	57	000001011000
58	01011011	58	000001011001
59	01001010	59	000000101011
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111

โดยดูจากตาราง Binary 9 bit คือ "010011011" ที่เป็น code word ของ white run lengths จะแทนจุดที่ต่อเรียงกัน 1728 จุด จากตัวอย่างนี้จะมีอัตราส่วนการอัดข้อมูลเท่ากับ 1728/9 =192 เท่า แต่หากดูที่ white run length ที่เป็น 2 (จุดขาว 2 จุดต่อกัน) จะใช้ code word ถึง 4 bit คงมิใช่เป็นการอัดข้อมูลแน่ ทั้งนี้เนื่องจากว่าตารางนี้ได้ถูกออกแบบ มาโดยพิจารณาจากความเป็นจริงของข้อมูลที่ส่ง เช่น white run length ที่เป็น 2 มีโอกาสเกิดน้อยมาก เพราะมักจะเกิดเป็น Block run length มากกว่า

จุดสิ้นสุดของ scan code แต่ละเส้นจะมี "EOL cde" มีรหัสเป็น (000000000001) EOL code จะทำหน้าที่เป็น line synchronization code เมื่อด้านรับได้รับ 1 เส้น scan จะตรวจสอบว่ามี error ใน line scan ที่ได้รับมาหรือเปล่า และจะ decode 1 scan line ออกมาว่ามีจำนวนจุดเป็น 1728 จุด หรือไม่ สำหรับเครื่อง fax บางแบบจะใช้จำนวนจุดในหนึ่งเส้น scan เป็น 2048 หรือ 2432 จุด และบิทที่ตามหลัง EOL มาหากเป็น "1" เป็นการบอกว่า

ตัวอย่างการเข้ารหัส MH code

Run Length	คือ	2w	5B	5W	2B	585W	รวมทั้งหมด 599 จุด
MH Code		0111	0011	1100	11	01101000+10100	
จำนวน Bit		4	4	4	2	13	รวม 27 Bit

Compression ratio 599/27 = 22.2

รูปที่ 3 ตัวอย่างการเข้ารหัสแบบ Modified Huffman

สแกนไลน์ ที่จะรับต่อเป็นการเข้ารหัสแบบ one-dimension (MH) coding หากเป็น "0" จะเป็น two-dimension (MR) coding

- Modified Read (MR) และ Modified Modified Read Code (MMR)

ข้อความส่วนมากที่อยู่ในเอกสารที่ต้องการส่งจะมีความเกี่ยวพัน หรือเรียงกันซึ่งเป็น ระบบความสัมพันธ์แบบหนึ่ง MR code เป็นการเข้า code โดยใช้ตัว address แบบสัมพันธ์กัน เพื่อประโยชน์ในการบีบอัดข้อมูลสูง ๆ มากกว่าแบบ MH code ถึงแม้ code ในลักษณะนี้ไม่ถูก recommendation โดย CCITT ก็ตาม แต่ชื่อนี้ก็เป็นที่รู้จักกันโดยทั่วไป

เวลาที่ใช้ในการส่งจริง ๆ จะขึ้นอยู่กับความเร็วของ modem แต่หากใช้ MR code จำนวน bit จะใช้น้อยกว่า MH code และยิ่งเป็นแบบ Modified Modified Read (MMR) ก็ยิ่งน้อยไปใหญ่

หลักการคือ โดยปกติแล้วเส้นสแกน 2 เส้นที่ติดกันจุดดำที่เกิดขึ้นในเส้นสแกนอันดับ 2 จะเริ่มตรงแนวเดียวกับ จุดดำที่เกิดในเส้นสแกนอันดับ 1 ในกรณีนี้ในการเข้ารหัสแบบ MR/MMR vertical mode จะใช้เพียง 1 บิท ในการแสดงเงื่อนไขอย่างนี้ แต่ในกรณีที่ในหน้ากระดาษ มีจุดที่มีความสัมพันธ์กับแนวตั้งน้อยมากจะไปใช้ pass mode หรือไม่ก็ horizantal mode แทน ในการเข้ารหัสความแตกต่างระหว่างเส้นสแกน 2 เส้น line memory ในเครื่องแฟ็กซ์ จะเก็บทุก ๆ จุดของเส้นเส้นสแกนทั้งสอง

เส้นเส้นสแกนแรกที่อยู่ในหน่วยความจำถูกเรียกว่า reference line มันได้ถูกเข้ารหัสไปก่อนแล้ว แต่ตำแหน่งของจุดดำในเส้นนี้ ยังอยู่ในหน่วยความจำใช้อ้างอิงเส้นเส้นสแกนถัดไป (เส้นถัดไปเรียกว่า coding line) หากดูที่ reference line และ coding line จากซ้ายไปขวาจนเจอจุดที่มีการเปลี่ยนสี จากขาวไปดำหรือดำไปขาว จุดๆ นี้เรียกว่า changing pel (CP) หากคู่ของจุดนี้เกิดติดกัน บน line คู่หนึ่งมากกว่า 3 จุด จะใช้ โหมด รหัสเป็น Vertical mode หากคู่ของ CP ไม่อยู่ในช่วง ± 3 จุด จะเลือกใช้ 2 โหมด ที่เหลือ ตัวอย่างกรณี pass mode ใช้เมื่อจุด CP เป็น Reference line ไม่ถูกจับคู่หรือผ่านไป 2 จุด mode vertical สามารถกลับนำมาใช้ อีกได้ หากเกิดคู่ของ CP อยู่ในช่วย ± 3 จุด อีกส่วน horizontal mode จะใช้เมื่อเกิดจุด CP ติดต่อกัน 2 จุดบนเส้น coding line จากตัวอย่างนี้เป็นแบบ resolution (K=2) กล่าวคือ เส้นแรกเข้า code แบบ MH code เส้นที่ 2 ใช้ 2-D coding สลับกันอย่างนี้ (Hor, Ver, Pass) แต่ยังมีแบบ Fine Resolution (K=4) คือเส้นแรกเข้ารหัสแบบ MH อีก 3 เส้นถัดมาจะเข้ารหัสแบบ 2-D code และกลับมาใช้ MH code ในเส้นถัดไป ทั้งสองแบบนี้เรียกว่า MR coding ส่วนอีกแบบเป็น MMR coding (K=∞) เส้นแรกจะใช้ code

word แทนจุดขาวตลอดเส้น (ปกติเอกสารเป็นเช่นนั้น) เส้นถัดจากนั้นไปจนจบแผ่นจะใช้ 2-D code ตลอด ตัว K เป็นตัว factor บอกว่าจะกลับมาใช้รูปแบบเก่าทุก ๆ กี่ครั้ง

Scan line—

Reference WWWWWBBBBWWWWWWWBBBBWWWWWWWBBWWWWWWWWWWW

Coding WWWWBBBBBWWWWWWWWWWWWWWWWWWWBBWWWWBBBWWWW

Changing Pels—

Reference line * * * * * *

Coding line * * * * * *

Coding Mode V V P V V H

V; Vertical, H; Horizontal, P; Pass

W; White Pel, B; Black Pel

รูปที่ 4 โหมดการเข้ารหัสแบบ Modified read

Digital Synchronization

ก่อนหน้าที่เทคนิคทางดิจิตอลของแฟ็กซ์ จะได้รับการพัฒนาข้อมูลของแฟ็กซ์ จะถูกส่งในลักษณะเดียวกับ analog แนวความคิดในการ process สัญญาณ โดยใช้ clock การ quantizing และการเก็บข้อมูลทางดิจิตอล ซึ่งค่อนข้างจะเป็นขบวนการทางวิทยาศาสตร์ ระบบการสแกน และการจัดเก็บเป็นไปในลักษณะกลไกซึ่งเป็น การยากที่จะลากเส้นตรงสักเส้นในทางด้านรับ และปกติแล้วก็ใช้ photosensor เพียงตัวเดียวในการสแกน และการเขียนจุดบน record ในทางกลไกแล้วก็ปรับแต่งได้ลำบาก และมักจะเกิดการ drift ระหว่าง clock ทางด้านส่งและ clock ทางด้านรับ และ modem ทาง digital ก็ไม่สามารถส่งสัญญาณซึ่งให้ทางด้านรับจะ lock clock ได้ง่าย ๆ เมื่อเทคนิคทางดิจิตอลถูกนำมาพัฒนากับเครื่องแฟ็กซ์ ระบบการ scan และการสร้างจุดทางกลไกก็ถูกยกเลิกไป และ โมเด็ม ทางด้านรับสามารถ synchronize กับทางด้านส่งได้ การใช้ระบบความถี่มาตรฐานก็เลิกใช้ไป กลไกการ scan จุดบนเอกสารก็ถูกแทนด้วย CCD chip โดยอ่านตามเส้นสแกน ใน step ที่แน่นอนซึ่งอาศัย clock ทางด้านส่ง

ระบบการทำงานแบบ Synchronous ของ โมเด็ม ถูกนำมาใช้กับระบบสัญญาณของ แฟ็กซ์ Group 3 โดยอาศัยการส่งสัญญาณแบบต่อเนื่อง ถึงแม้ขณะไม่มี data จะส่งและเนื่องจากโมเด็ม ทางด้านรับ synchronize กับสัญญาณที่รับเข้ามาทำให้มันสามารถ lock ตัวเองเข้ากับความเร็วของ โมเด็ม ทางด้านส่ง ในระบบแฟ็กซ์ G.3 นี้ตัว mark จุดไม่จำเป็นต้องเคลื่อนที่ไปตามระยะเวลาที่แน่นอนทุกๆตำแหน่งในหน้ากระดาษจะมีพิกัดบอก ข้อมูลสัญญาณของแฟ็กซ์ ทางด้านรับจะถูก print เมื่อบรรทัดถัดไปที่รับเข้ามาได้รับ การถอดรหัสก่อนและจัดเก็บก่อนเวลาในการ print ค่อนข้างช้ากว่าเวลาในการสแกนเส้นเล็กน้อย clock ของ โมเด็มทางด้านรับจะเป็นตัว lock กับขบวนข้อมูลที่รับเข้ามา แม้จะมีการ shift ของความถี่ carrier ทางด้านรับไป 2-3 hertz ก็ตาม

Protocal

Protocal เป็นชุดของข้อตกลงที่ผู้กระทำการสื่อสารทั้งสองฝ่ายยอมรับกัน และยอมรับกันในระดับระหว่างชาติ ทั้งในทาง physical communication เช่น mail และ electronic communication เช่น โทรศัพท์ facsimile ใน แฟ็กซ์ Group 3 นั้นเป็นข้อตกลงกันในกฎเกณฑ์การส่งระหว่าง fax ตัวส่งและ fax ตัวรับเอกสารที่จะจัดส่งต้องอยู่ในขนาดที่แน่นอนและจำกัดเมื่อเริ่มทำการส่ง electronic protocal หรืออาจเรียกว่า"handshaking" หากเกิดไม่สามารถเข้ากันได้ ในขั้นตอนนี้จะไม่เกิดการส่งขึ้น ผู้ใช้เครื่องแฟ็กซ์ ไม่จำเป็นต้องรู้ถึงรายละเอียดของส่วนทาง electronic ในส่งของแฟ็กซ์ ตาม Recommendation ของ CCITT T.30 ได้ระบุขั้นตอนการส่งเอกสารโดยใช้ผ่านชุมสายโทรศัพท์ (PSTN) การทำการเรียกจาก fax ต้นทางเพื่อทำการส่งเอกสารถูกจัดแบ่งออกเป็น 5 ขั้นตอนดังต่อไปนี้

1) Phase A - Call Setup

เป็นการเรียกจาก fax หมุนหมายเลขโทรศัพท์ที่ต่อกับ fax อีกด้านหนึ่ง สัญญาณ Ring Signal และ CNG จะได้รับทางแฟ็กซ์ ตัวที่ถูกเรียก สัญญาณ CNG เป็น tone จะเป็นเสียง beep ว่าเป็นการเรียกจากเครื่องแฟ็กซ์ มิใช่โทรศัพท์เรียกมา เครื่องแฟ็กซ์ ทางด้านรับจะตอบ Ring Signal โดยยกหู " off-hook" และกระตุ้นให้ เครื่อง fax ทำงานหลังจากนั้น 1 วินาที แฟ็กซ์ที่ถูกเรียกจะส่ง "station Identification" (CSI) ของตัวเอง โดยส่งนำไปก่อนด้วยความถี่ 2100 Hz ยาวนาน 3 วินาที

2) Phase B - Premessage Procedure

เครื่องแฟ็กซ์ ทางด้านส่งจะจัดส่ง "Digital Identification" (DIS) ที่ความเร็ว 300 Bit/Sec เป็นการบอกคุณสมบัติ และ option อื่น ๆ ของ fax ทางด้านรับไปให้ตัวส่ง fax ทางด้านส่งก็จะตรวจดู คุณสมบัติเครื่อง fax ทางด้านรับแล้วจัดส่ง "Digital Command Signal" (DCS) กลับไปให้ทางตัวรับ ว่าตกลงใช้คุณสมบัติข้อใดในการรับส่ง ต่อจากนั้นแฟ็กซ์ตัวส่งจะตรวจสอบคุณภาพของสายสัญญาณ โดยทดสอบที่ความเร็วสูงที่สุด เรียกว่า "Training Signal " (TCF) หากทางด้านรับรับผลสัญญาณดีหรือมี error น้อย ก็จะส่ง (CFR) กลับไป และครื่องแฟ็กซ์ ทั้งสองก็เริ่มเข้าขั้นตอนการรับและส่ง

3) Phase C-Message Transmission

เครื่องแฟ็กซ์ ทางด้านส่งจะเริ่มส่งข้อมูลของเอกสารหรือรูปภาพของ 1 หน้ากระดาษทั้งหมดออกไปหาแฟ็กซ์ตัวรับ

4) Phase D- Postmessage Procedure

เครื่อง แฟ็กซ์ ด้านส่งจะจัดส่งสัญญาณ "Return to Cantrol" (RTC) เป็นการให้เครื่องแฟ็กซ์ ทั้งสองด้านมาติดต่อกันที่ speed 300 Bit/Sec อีกครั้งแล้วส่งสัญญาณ "End of Procedure" (EOP) เมื่อทางด้านรับได้รับ ก็จะตอบกลับด้วยสัญญาณ "Message Confirmation" (MCF) เป็นการแสดงถึงว่าเครื่องแฟ็กซ์ ทางด้านรับได้รับเอกสารสมบูรณ์แล้ว

5) Phase E-Call Release

เครื่อง fax ทางด้านเรียกจะส่ง "Disconnect Signal" (DCN) ไปพร้อมกับตัวเอง หลุดออกจาก การครองคู่สายโทรศัพท์ ภายหลังจากการตอบ Ring Signal เครื่อง fax ผู้ถูกเรียกจะค่อยไป 1 วินาที และจะส่งความถี่ 2,100 Hz 3 วินาที ตามด้วยหยุด 75 มิลิวินาที เรียกว่าเป็น CED tone หลังจากนั้นจะตามด้วย DIS ด้วย speed 300 Bit/Sec เพื่อบอก fax ตัวเรียกถึง มาตรฐานและส่วนที่มีให้เลือกที่ตัวรับเองสามารถทำได้

หากเครื่อง fax ทำงานแบบ manual ผู้ทำการเรียกจะต้องกดปุ่ม SEND ภายหลังจากได้รับสัญญาณ CED เพื่อจะเริ่มต่อเครื่อง fax ตัวเรียกเข้ากับคู่สายโทรศัพท์แล้ว fax ตัวเรียกก็จะเตรียมพร้อมจะรับสัญญาณ DIS ต่อไป สัญญาณ DIS หรือสัญญาณ Handshake อื่น ๆ จะส่งซ้ำในช่วงเวลา 3 วินาที ถ้าไม่ได้รับ การตอบสนอง จาก fax ฝั่งตรงข้าม

หากเครื่อง fax ตัวเรียกเป็นแบบอัตโนมัติ ตัวส่งจะเริ่มส่ง CNG tone ภายหลังจากการหมุนโทรศัพท์ CNG มีความถี่ 1,100 Hz ในช่วงเวลา 0.5 วินาที และจะส่งทุก ๆ 3 วินาที

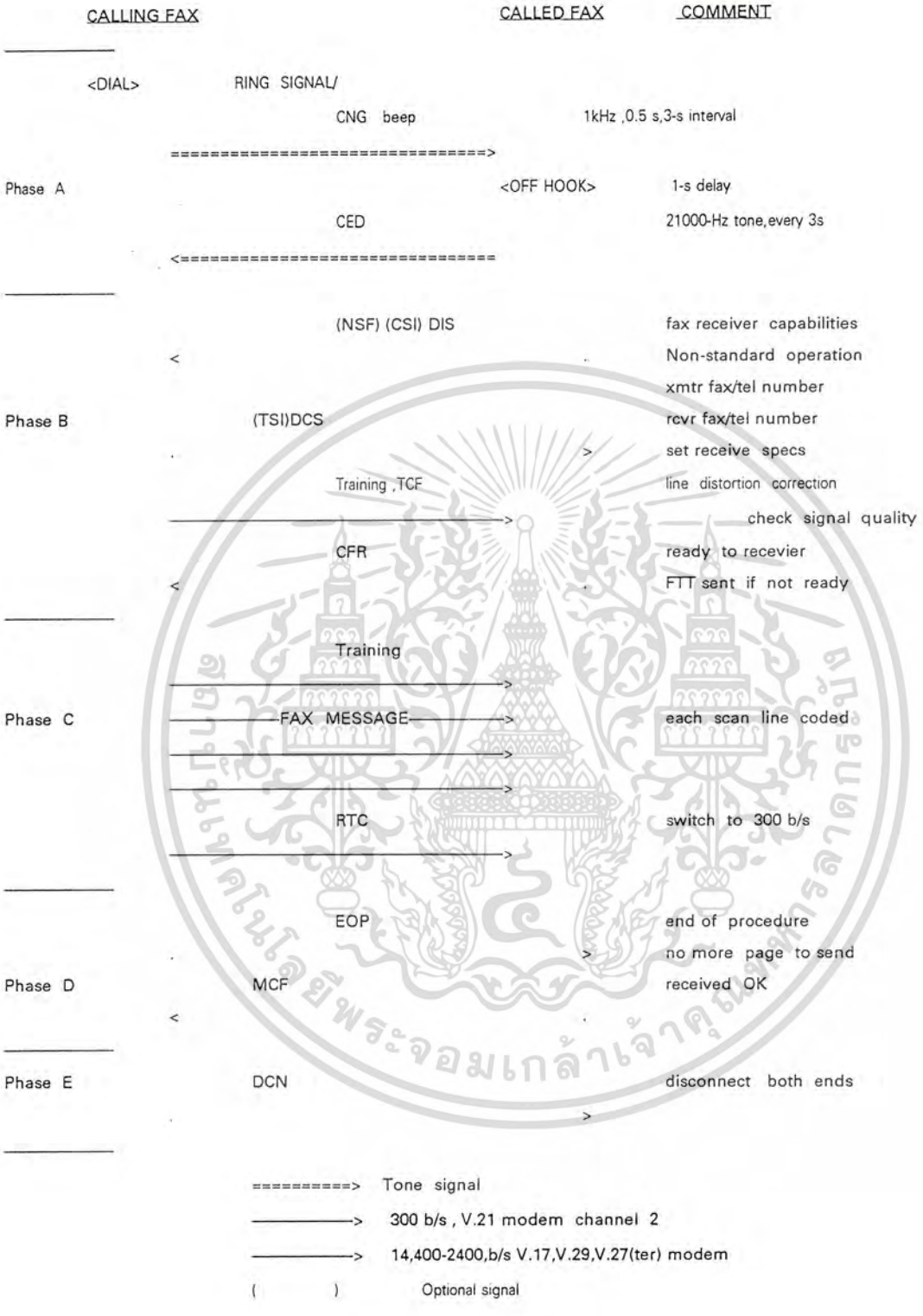
Nonstandard Facility Call

คุณสมบัติข้อนี้มีอยู่เฉพาะใน แฟ็กซ์ Group 3 facility ตัวนี้จะร้องขอในช่วงที่มีการ handshake ซึ่งเป็นการบอกลักษณะการติดต่อว่าไม่ติดต่อกันในระบบแฟ็กซ์ Group 3 แบบปกติ "Nonstandard Facility" (NSF) เป็นการร้องขอจากฝ่ายรับว่าตัวเองมีความสามารถที่จะไม่ใช้ระบบตาม T Series ตามที่ CCITT แนะนำ เมื่อทาง fax ตัวส่งได้รับ NSF หากตอบด้วย "Nonstandard setup" (NSS) เพื่อ lock fax ทางด้านรับ ให้ปฏิบัติการตาม nonstandard ตามที่ระบุไว้ใน NSF บริษัทผู้ผลิตบางรายจะมีระบบ error-correction เฉพาะแบบเองแฟ็กซ์ Group 3 ที่ตอบ NSS กลับไปจะต้องมีความสามารถที่จะปฏิบัติตาม nonstandard ตามที่ฝ่ายตรงข้ามต้องการ

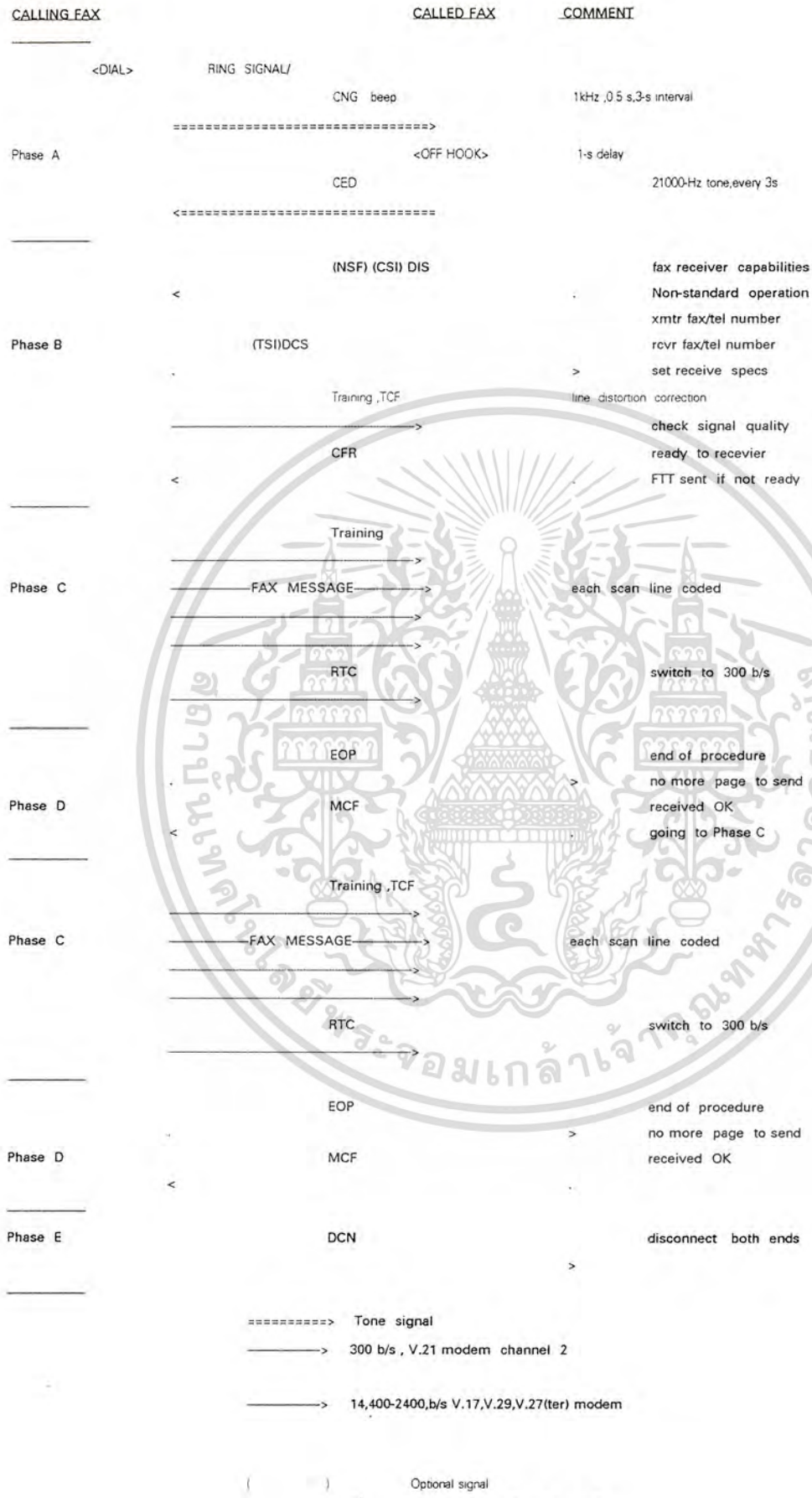
ส่วนสัญญาณ CSI จะเป็น code ที่บอกหมายเลขโทรศัพท์เครื่อง fax ตัวที่ถูกเรียก อาจเป็นชื่อ หรือข้อความ เพื่อให้ทางด้านส่งได้ตรวจสอบว่าส่งมาถูกที่ ทางด้านส่งก็จะส่งสัญญาณ TSI ซึ่งบรรจุไปด้วย หมายเลขโทรศัพท์ของตัวเองส่งให้กับทางด้านรับ

Polling Called Party

Polling เป็นระบบที่จะมีเครื่อง fax ที่เรียกว่า central fax ซึ่งจะ program ให้เรียกไปยังเครื่อง fax เครื่องอื่นๆ และให้เครื่อง fax ที่ถูกเรียกส่งเอกสารกลับมา จะใช้ในกรณีที่ต้องการให้ปลายทางส่งเอกสาร มาวันละครั้งหรือมากกว่า เข้ามายัง Central fax ระบบนี้จะช่วยแก้ปัญหา line-busy ได้มาก เครื่อง fax ใน Group 3 สามารถ program หน่วยเวลาให้เรียกแบบอัตโนมัติในเวลาใด ๆ รวมถึงตอนกลางคืน ภายหลังจากการเรียก จะได้รับ NSF กลับมา และตามด้วย DIS ตัวเรียกจะส่ง NSC, CIG และ DTC กลับไปยังตัวถูกเรียก หากผู้ถูกเรียกจะต้องการใช้ nonstandard facility จะตอบ NSS กลับมา และตามด้วย TSI ซึ่งเป็นหมายเลข เบอร์โทรศัพท์ผู้ถูกเรียก การเช็คคุณภาพของคู่สายสลังเกตว่าจะเป็นหน้าที่ของฝ่ายส่งเสมอ ดังนั้นตัวถูกเรียก ซึ่งทำหน้าที่ส่ง จะส่ง TCF เช็คคุณภาพ line ส่วนขั้นตอนที่เหลือเหมือนแบบเดิม ๆ ที่ผ่านมา ต่างกันตรงที่ ผู้ส่งจะเป็นผู้ถูกเรียกเท่านั้น



รูปที่ 5 แสดงการส่งแฟ็กซ์ 1 แผ่น



รูปที่ 6 แสดงการส่งแฟกซ์ 2 หน้า

ตารางที่ 5

Group 3 Handshake abbreviation

Abbreviation	Function	Signal Format
CED	Called station identification	2100 Hz
CFR	Confirmation to receive	X010 0001 [1850 or 1650 Hz for 3s]
CRP	Command repeat	X101 1000
CIG	Calling subscriber identification	1000 0010
CNG	Calling tone	1100 Hz on 0.5 .off 3 s
CSI	Called subscriber identification	0000 0010
CTC	Continue to correct	X100 1000
CTR	Response to continue to correct	X010 0011
DCN	Disconnect	X101 1111
DCS	Digital command signal	X100 0001
DIS	Digital identification signal	0000 0001
DTC	Digital transmit signal	1000 0001
EOL	End-of-line	00000000000 (11 zeros)
EOM	End of message	X111 0001 [1100Hz]
EOP	End of procedure	X111 0100
EOR	End of retransmission	X111 0011
ERR	Response to end of retransmission	X011 1000
FCD	Facsimile codes data	0110 0000
FCF	Facsimile control field	-
FIF	Facsimile information field	-
FTT	Failure to train	X010 0010
GC	Group command	G1 1300Hz for 1.5-10.0s
		G2 2100Hz for 1.5-10.0s
GI	Group identification	G1/G2 1650/ 1850 Hz on 0.5,off 3s
HDLC	High level data link control	-
LCS	Line conditioning signal	1100Hz
MCF	Message confirmation	X011 0001 [1650 or 1850 Hz]
MPS	Multipage signal	X111 0010
NSC	Nonstandard facilities command	1000 0100
NSF	Nonstandard facilities	0000 0100
NSS	Nonstandard setup	X100 0100
PIN	Procedure interrupt negative	X011 0100
PIP	Procedure interrupt positive	X011 0101
PIS	Procedure interrupt signal	432 Hz for 3 s
PPS	Partial page signal	X111 1101
PPR	Partial page request	X011 1101
PRI-EOM	Procedure interrupt - EOM	X111 1001
PRI-EOP	Procedure interrupt - EOP	X111 1100
PRI-MPS	Procedure interrupt - MPS	X111 1010
RCP	Return to control for partial page	0110 0001
RNR	Receive not ready	X011 0111

ตารางที่ 5 (ต่อ)

Group 3 Handshake abbreviation

Abbreviation	Function	Signal Format
RR	Receive ready	X111 0110
RTN	Retrain negative	X011 0010
RTP	Retrain positive	X011 0011
RTC	Return to control	six EOLs
TCF	Training check	Zeros for 1.5 s
TSI	Transmitting subscriber identification	X100 0010
[. . .]	Alternative tonal signalling for Group 1 or 2	

Handshake Signal Formats

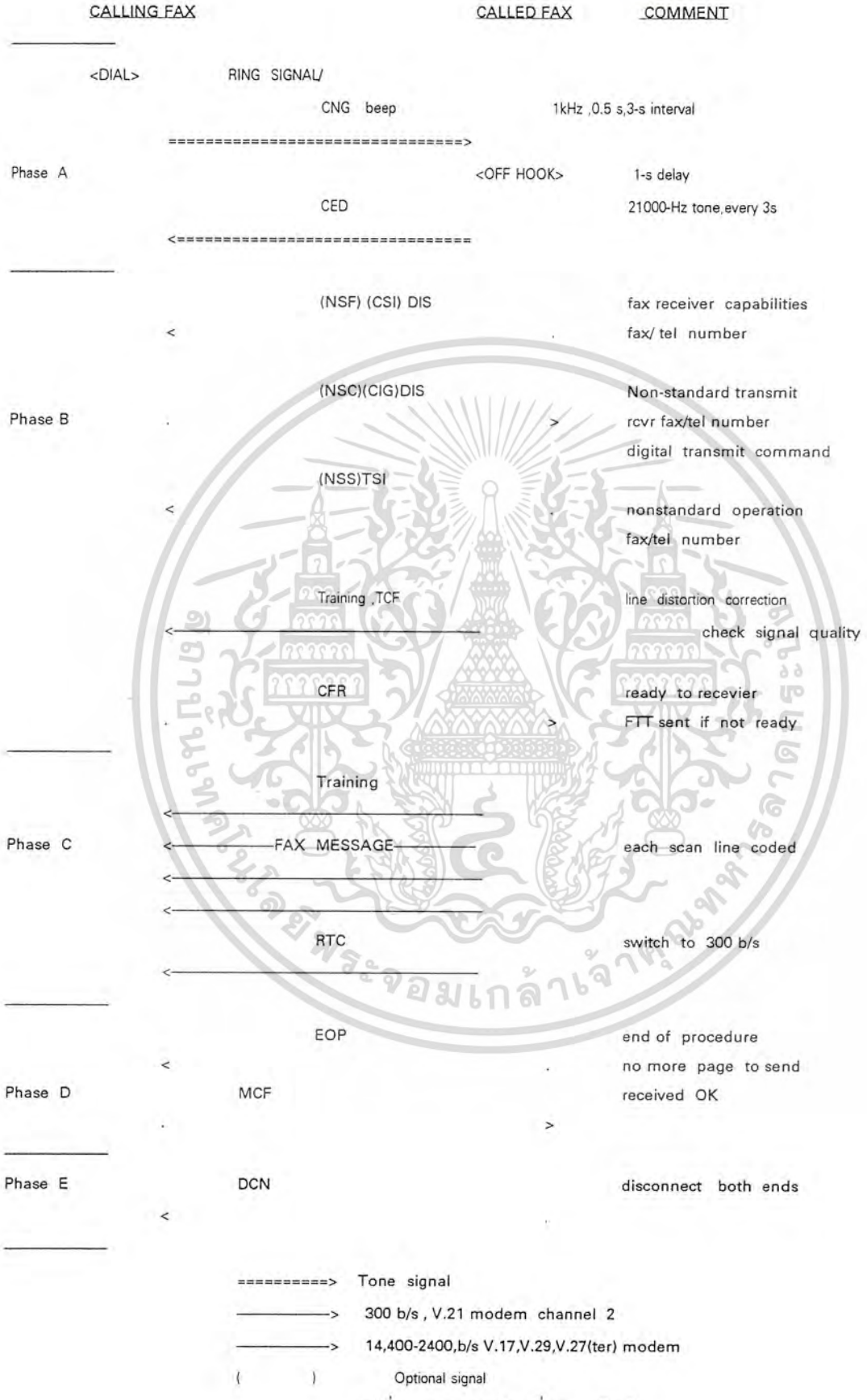
เมื่อเริ่มขบวนการ handshake ครั้งแรกทางด้านรับจะ set ตัวเองเป็น V.21 ใน speed 300 Bit/Sec เพื่อส่งข้อมูลกลับไปยังผู้เรียก ระบบ Frame HDLC จะนำมาใช้ในการ handshake ขบวนการ handshake สามารถกระทำได้ที่ความเร็ว 2.4 Kbit/Sec แต่ก็แทบจะไม่ใช้เลย ลำดับขั้นตอนการส่งมีดังนี้

1. สัญญาณ flag จะถูกส่งล่วงหน้าก่อนส่งสัญญาณ DIS ยาวนานประมาณ 1 วินาที เพื่อเช็คระบบ echo suppressor ของคู่สายโทรศัพท์ทำงานในทิศทางที่ถูกต้อง นอกจากนี้การ synchronize ของ frame จะอาศัยสัญญาณตัวนี้
2. byte แรกของ HDLC frame คือ address field เพื่อจะบอกว่าเป็น station ใด ๆ กรณีใช้งาน ในระบบ multipoint network แต่ในกรณีใช้คู่สายโทรศัพท์ PSTN จะถูกกำหนดเป็น 1111 1111
3. ส่วนของ control field ของ HDLC จะถูกกำหนดเป็น 1100 1000 เพื่อระบบว่าเป็น frame สุดท้ายหรือไม่มี frame อื่น ๆ ตามมาจากด้านที่ส่ง frame นี้ออกมา แต่หากเป็น 1100 0000 จะหมายถึงจะมี frame อื่น ๆ ตามมา ส่วน " frame error-checking sequence" จะมีขนาด 2 Byte ซึ่งคำนวณมาจากข้อมูลใน frame ที่จัดส่ง

DIS/DTC Signal

สัญญาณ DIS จะถูกจัดส่งด้วย speed 300 Bit/Sec และส่งจากแฟกซ์เครื่องรับเพื่อตอบการเรียกข้อมูล ภายในจะบอกถึง ความสามารถทั้งที่เป็น standard และ nonstandard บอกให้ตัวเรียกทราบ ส่วน DTC จะมีลักษณะข้อมูลเหมือนกับ DIS แต่จะส่งจากตัวส่งไปยังตัวรับ เพื่อยืนยันว่าจะใช้ความสามารถตัวใด ที่มีเหมือนกันทั้งสองข้อมูลต่าง ๆ ดังต่อไปนี้จะอยู่ในส่วนของ frame ดังกล่าว

1. ชนิดของโมเด็ม ที่จะใช้ในการติดต่อ
2. จำนวนของจุดต่อการสแกน 1 เส้น
3. จำนวนของจุดต่อ recording line
4. จำนวนของเส้นสแกน ต่อนิ้ว (หรือมิลิเมตร) ในแนวตั้ง



รูปที่ 7 แสดงการโพลลิ่งของแฟ็กซ์

5. เทคนิคในการบีบอัดข้อมูลที่ใช้ในการส่ง
6. ความยาวของหน้ากระดาษสูงสุด
7. เวลาสูงสุดที่ต้องการทางแฟ็กซ์ ด้านรับในการบันทึกเส้นสแกน
8. จะใช้ "Error-correction Mode" (ECM) หรือไม่
9. จะใช้ nonstandard mode หรือไม่
10. จะใช้หน้ากระดาษขนาดเล็ก (A5 หรือ A6) ในเครื่องแฟ็กซ์ หรือไม่
11. จะใช้ V.17 14.4 Kbit/S หรือไม่

สัญญาณ DCS เครื่องแฟ็กซ์ ผู้เรียกจะส่งออกไปอย่างอัตโนมัติ โดยจะถอดรหัสจากสัญญาณ DIS แล้วเลือก "feature" รูปแบบที่สามารถเข้ากันได้ทั้ง 2 ฝ่ายในการรับส่ง หลังจากส่ง DCS จะทำการเช็คคุณภาพคู่สายสัญญาณโดยการส่ง "Training Signal" (TCF) คือส่งเลข "0" ไปอย่างต่อเนื่องยาวนาน 1.5 วินาที หากทางด้านรับได้รับถูกต้องตลอดก็จะตอบกลับด้วยสัญญาณ CFR เมื่อทางด้านรับได้รับก็จะทำการส่งข้อความ(fax data) เมื่อสิ้นสุดการส่งแฟ็กซ์ 1 หน้า ทางด้านส่งจะส่ง "Return to Command" (RTC) ไปทางด้านรับด้วย ความเร็ว 300 Bit/Sec ตามด้วย "End Of Procedure" (EOP) ทางด้านรับจะส่ง "Message Confirmation" (MCF) เพื่อแสดงว่าได้รับแฟ็กซ์อย่างถูกต้องแล้ว เครื่องแฟ็กซ์ ทั้งสองก็หลุดออกจากคู่สายโทรศัพท์

ต่อไปนี้เป็นรายละเอียดการจัดเรียงบิท รวมถึงบิทที่เพิ่มจาก 40 บิทเข้าไปใน T.30 protocal แฟ็กซ์ Group 3 รุ่นเก่าจะไม่มีส่วนเพิ่มเติมนี้แต่อย่างไรก็ตาม fax รุ่นใหม่ก็เข้ากันได้กับเครื่องแฟ็กซ์ รุ่นเก่า และส่วนที่เพิ่มพิเศษบิทที่ 41-49 มีการกำหนดใช้เมื่อเดือน ตุลาคม 1991 สำหรับใน mode ความละเอียดสูง และจะสังเกตุบิท 24, 32 และ 40 จะเป็น "entenal field" ใช้ในการขยาย facility ใหม่ ๆ นำมาใช้

Error-Concealment Techniques

เนื่องจากในระบบคู่สายโทรศัพท์มักจะเกิด error เข้ามาเสมอ ๆ จะมีผลต่อเอกสารทางด้านรับ แต่ก็สามารถหยุดขบวนการได้เมื่อตรวจพบความผิดพลาด จุดประสงค์ของ "error- concealment" เพื่อทำให้ผล error ที่เกิดในคู่สายโทรศัพท์ มีผลน้อยที่สุดต่อเอกสารที่พิมพ์ออกมาทางด้านรับ โดยอาศัยหลักการที่ว่า "end of line" (EOL) จะถูกส่งระหว่างทุก ๆ scan line ถ้าทางด้านรับไม่สามารถ detect EOL ในตำแหน่งที่เหมาะสมก็แสดงว่าได้เกิด error ขึ้นใน line โทรศัพท์ เหตุการณ์ต่อไปนี้เป็นไปได้ว่าเกิด error ขึ้นในสาย

- ได้รับสัญญาณ EOL ก่อนหน้าที่จะได้รับจุดจำนวน 1,728 จุดจนครบ
- มีจำนวนจุดมากกว่า 1,728 จุด ถึงจะตามด้วย EOL
- ไม่มีรูปแบบ Code word ที่ได้รับเข้ามาตรงตามตาราง Code word ที่มีอยู่
- เมื่อถอดรหัส scan line เส้นปัจจุบันแล้วอ้างอิงกับเส้น scan line ก่อนหน้านี้ไม่ได้เลย

ผลของ error ที่มีผลต่อเอกสารทางด้านรับจะอยู่ในเทอม "error sensitivity factor" (ESF) ESF ถูกนิยามว่าเป็นจำนวนของจุดที่ผิดพลาดทางด้านรับที่มีผลจากระบบการส่งหารด้วยจำนวนบิทที่ทำการส่งในเที่ยวที่เกิด error ESF จะแสดงถึง บิทที่ error ที่เกิดจากการส่งที่มีผลต่อเอกสารทางด้านรับ ในจำนวนการเข้ารหัส การเข้ารหัสแบบ

ตารางที่ 6

การจัดบิท DIS และ DTC

Bit Number	DIS and DTC	DCS
1	Transmitter -T.2 opration	
2	Receiver -T.2 operation	Receiver -T.2 operation
3	T.2 IOC = 176	T.2 IOC = 176
4	Transmitter -T.3 opration	
5	Receiver -T.3 operation	Receiver -T.3 operation
6	Reserved for future T.3 opration features	
7	Reserved for future T.3 opration features	
8	Reserved for future T.3 opration features	
9	Transmitter -T.4 opration	
10	Receiver -T.4 operation	Receiver -T.4 operation
11,12,13,14	Data signalling rate	Data signalling rate
0,0,0,0	V.27ter fall back mode	2400 bit / s/ V.27 ter
0,1,0,0	V.27ter	4800 bit / s/ V.27ter
1,0,0,0	V.29	9600 bit / s/ V.29
1,1,0,0	V.27ter and V.29	7200 bit / s/ V.29
0,0,1,0	not used	14400 bit / s/ V.33
0,1,1,0	reserved	12000 bit / s/ V.33
1,0,1,0	not used	reserved
1,1,1,0	V.27ter , V.29 and V.33	reserved
0,0,0,1	not used	14400 bit / s/ V.17
0,1,0,1	reserved	12000 bit / s/ V.17
1,0,0,1	not used	9600 bit / s/ V.17
1,1,0,1	V.27ter , V.29 , V.33 and V17	7200 bit / s/ V.17

ตารางที่ 6 (ต่อ)

การจัดบิท DIS และ DTC

Bit Number	DIS and DTC	DCS
0,0,1,1	not used	reserved
0,1,1,1	reserved	reserved
1,0,1,1	not used	reserved
1,1,1,1	reserved	reserved
15	Vertical resolution = 7.7 line/ mm	Vertical resolution = 7.7 line/ mm
16	Two dimensional coding capability	Two dimensional coding
17,18	Recording width capabilities	Recording width
(0,0)	1728 pels for 215 mm	1728 pels for 215 mm
(0,1)	1728 pels for 215 mm and 2048 pels for 255 mm and 2432 pels for 303 mm	2432 pels for 303 mm
(1,0)	1728 pels for 215 mm and 2048 pels for 255 mm	2048 pels for 255 mm
(1,1)	Invalid (see Note 7.)	Invalid
19,20	Maximum recording length capability	Maximum recording length
(0,0)	A4 (297 mm)	A4 (297 mm)
(0,1)	Unlimited	Unlimited
(1,0)	A4 (297 mm) and B4 (364 mm)	B4 (364 mm)
(1,1)	Invalid	Invalid
21,22,23	Receiver time for printing a line	Minimum scan line time

ตารางที่ 6 (ต่อ)

การจัดบิท DIS และ DTC

Bit Number	DIS and DTC	DCS
(0,0,0)	20 ms at 3.85 or 7.7 1/ mm	20 ms
(0,0,1)	40 ms at 3.85 or 7.7 1/ mm	40 ms
(0,1,0)	10 ms at 3.85 or 7.7 1/ mm	10 ms
(1,0,0)	5 ms at 3.85 or 7.7 1/ mm	5 ms
(0,1,1)	10 ms at 3.85 1/ mm : 5 ms at 7.7 1/ mm	
(1,1,0)	20 ms at 3.85 1/ mm :10 ms at 3.85 1/ mm	
(1,0,1)	40 ms at 3.85 1/ mm : 20 ms at 7.7 1/ mm	
(1,1,1)	0 ms	0 ms
24	Extend Field	Extend Field
25	2400 bit / s handshaking	2400 bit / s handshaking
26	Uncompressed mode	Uncompressed mode
27	Error correction mode	Error correction mode
28	Set to " 0 "	Frame size 0 = 256 octets
		1 = 64 octets
29	Error limiting mode	Error limiting mode
30	Reserved for G4 capability on PSTN	Reserved for G4 capability on PSTN
31	T.6 coding capability	T.6 coding capability
32	Extend Field	Extend Field
33	Validity of bits 17,18	Recording width
(0)	Bit 17, 18 are valid	Recording width indicated by bits 17,18

ตารางที่ 6 (ต่อ)

การจัดบิท DIS และ DTC

Bit Number	DIS and DTC	DCS
(1)	Bit 17, 18 are invalid	Recording width indicated by this field bit
	Recording width capabilities	Middle 1216 elements of 1728 pels
34	1216 pels for 151 mm	
35	864 pels for 107 mm	Middle 864 elements of 1728 pels
36	1728 pels for 151 mm	Invalid
37	1728 pels for 107 mm	Invalid
38	Reserved for future	
39	Reserved for future	
40	Extend Field	Extend Field
41	Capability to emit nonfacsimile file	Invalid
42	Facsimile service info file (FSI)	Facsimile service info file (FSI)
43	Binary file transfer (BFT)	Binary file transfer (BFT)
44	Document transfer mode (DTM)	Document transfer mode (DTM)
45	Electronic data transfer (EDI)	Electronic data transfer (EDI)
46	Basic transfer mode (BTF)	Basic transfer mode (BTF)
47	Reserved for future file transfer mode	Reserved for future file transfer mode
48	Extend field	Extend field
49	Reserved for future file transfer mode	Reserved for future file transfer mode

NOTE on bit 28 :

For ECM ,set to "0"

for BFT. either "0 " or "1" can be used to specify maximium frame size.

NOTES to Table

1. T.2 fax units must have an index of cooperation (ICO) of 264

2. T.3 fax units must have an index of cooperation (ICO) of 264

3. T.4 fax units must have 297 mm recording paper length capabilities

4.

DIS or DTC frame copabilities	B/s ,equipment operable at
V.27ter	4800 or 2400
V.29	9600 or 7200
V.33	14400 or 12000
V.17	14400,12000,9600 or 7200

5.Recoring width tolerances are + or - 1%

6. The standard DIS DTC,and Dcs field is 24 bits long . " Extend field "bit set to 1 ,adds 8 bits (32,40,48).

7. If DIS bits 17,18 are received as (1,1),interpret as (0,1)

8. Bit 28 of DCS is valid only when bit27 is set to 1 for error correction mode

9. When bit 33 is set to 1 in DCS , meaning of bit 15 is modified to mean vertical resolution is higher than 7.71 /mm

10.When the recording width is A4 only ,the field consisting of bits 33-40 need not be present

11. The optional T.4 error correction mode of operation requires 0 ms of the minimum scan line time capability. Bit 21 -23 indicate the minimum scan line time , regardless of availability of the error corection mode . In the case of error correction mode , the sender sends DCS signal with bits 21-23 set to 1.1.1 indicating 0 ms capability . In the case of normal G3 transmission , the sender sends DCS signal with bits 21-23 set to the appropratenes according to the capabilities of the two machinea

12. T.6 coding capability specified by bit 31 is valid only when bit 27 (error correction mode) is set as a "1"

"Modified Read" (MR) จะมีความไวต่อ error ได้มากกว่าการเข้ารหัสแบบ "Modified Huffman" (MH) เนื่องจากแบบ MR จะบีบข้อมูลถึง 2 มิติ ในขณะที่แบบ MH เป็นแบบใช้ code word

Group 3 Error-Control Options

แฟกซ์ ใน Group 3 มาตรฐานจะไม่ใช้เทคนิค error-control หากเลือกใช้ option MR Mode จะมีการทำ forward error correction เพื่อแก้ไขความผิดพลาดในช่วงทำการส่งเอกสารระบบ error-control อีกแบบที่มีในแฟกซ์บางตัว จะใช้วิธีคำนวณจำนวนของ line ที่ error โดยทางด้านรับ detect จาก EOL หากเกินค่า limit ที่ต้องขอรับส่งใหม่ ส่วนที่ CCITT ได้ให้ option ไว้คือใน MR mode และได้ระบุว่า error-control ที่มีใช้ได้ 2 แบบ คือ แบบแรกจะจำกัดจำนวนของ error แต่ไม่ทำการแก้ไข แต่แบบที่สองจะใช้วิธี error-correction ในแบบแรกจะใช่ในแบบ "One-dimensional MH coding" ลำดับขั้นโดยจะแบ่งจำนวนจุด 1,728 จุดออกเป็น 12 กลุ่มแต่ละกลุ่มมี 144 จุด ในแต่ละกลุ่มจะถูกแยกว่าเป็นจุดขาวทั้งหมดหรือไม่ขาวทั้งหมด โดยใช้ 12 Bit นำหน้าแต่ละเส้นสแกน เพื่อระบุว่าเป็นขาวทั้งหมดหรือไม่ กลุ่มที่เป็นจุดขาวทั้งหมด จะไม่ถูกการเข้ารหัส กลุ่มที่ไม่ขาวทั้งหมดจะถูกเข้ารหัสแยกออกไปแทน MH วิธีการนี้เพื่อจำกัด error ให้เกิดแค่ครึ่งเส้น สแกน แทนที่จะเกิดทั้งเส้น scan แต่อย่างไรก็ตาม option ตัวนี้ยังไม่ได้รับการพิสูจน์ในทางปฏิบัติ

ในแบบที่สองถูกออกแบบมาให้เป็นสากลมากกว่า ดังนั้นจึงมีการนำมาใช้งานจริง จุดสำคัญในการออกแบบ ระบบerror-control คือจะต้องการให้เกิดการเปลี่ยนแปลงจากการ Recommendation ของ CCITT T.4 และ T.30 แบบนี้จึงได้มีใช้ทั้งในแบบ MH และ MR mode

ระบบ error-control ที่ถูกเลือกโดย CCITT คือใช้วิธี HDLC frame และทำการส่งแบบ half duplex มีขั้นตอนพอสรุปได้ดังนี้

1) ขบวนของข้อมูลจะถูกแบ่งใส่ลงใน "HDLC frame" frame ละ 256 Bytes

2) รหัสเช็ค error ที่ถูกสร้างขึ้นและจัดส่งไปกับ frame นั้น ซึ่งให้ทางด้านรับตรวจสอบ error ใน frame

3) หลังจากส่งหน้ากระดาษไปอย่างสมบูรณ์ ทางด้านรับจะขอให้ส่งมาใหม่ใน frame ที่มี error

4) ตัวส่งจะส่ง frame ที่เดิมมี error

5) ภายหลังจากการส่งซ้ำ ๆ ที่ frame เดิมถึง 4 ครั้ง ทางด้านส่งอาจจะหยุดหรือส่งต่อด้วยการสั่งให้ modem ลด speed ตัวเองลงมา

Nonstandard Operation

แฟ็กซ์ใน Group 3 จะยอมให้ทางด้านส่งและทางด้านรับ ไปใช้ระบบการส่งที่ไม่เป็นมาตรฐาน CCITT แต่จะใช้ตามมาตรฐานบริษัทที่ตนเองผลิตกรณีเป็นเครื่อง fax บริษัทเดียวกัน โดยบอกผ่านทางสัญญาณพื้นฐาน คือ NSF "Nonstandard facility" มีรหัส Binary เป็น (0000 0100) แทนว่าขอรับส่งเป็นแบบ nonstandard และ CCITT ได้มี Recommendation T.35 อธิบายส่วนนี้ว่า สัญญาณ NSF มักจะต้องตามด้วย 1 Byte ซึ่งแทน "Country code", "Provider code" และอย่างน้อยอีก 1 Byte หรือมากกว่าที่ใช้ อธิบายรายละเอียด รูปแบบที่จะใช้ สามารถดูจากตารางที่ 7ตัวอย่างด้านล่าง

รูปแบบที่ระบุมาใน NSF เครื่อง fax ต่างบริษัทที่ผลิตกันจะไม่รู้จัก หากทั้งสองฝ่าย handshake ใน NSF mode ได้สำเร็จ รูปแบบการรับส่งของ fax ก็จะเปลี่ยนไปใน mode การทำงานที่เหมาะสม ตัวอย่างเช่น อาจมี algorithms ในการ compress ข้อมูลที่ไม่เป็น มาตรฐาน, การเข้ารหัสลับข้อมูล "emeryption" หรือ set modem เป็น high speed หรืออื่น ๆ

ตารางที่ 7 U.S. provider Code Assignments

Company	City	Provider Code
Adobe	Mountain View CA	004400
AT&T Bell LAbs	Middletown NJ	004A00
Bogosian Engineering	Fremont CA	004200
Brooktrout Technology	Wellesley Hills MA	006200
Castelle	Santa Clara CA	006600
Data Race	San Antonio TX	004C00
Fremont Communication	Fremont CA	004600
Gammalink Graphics	Palo Alto CA	006400
Hayes Microcomputer	Atlanta GA	004800
Hybrid Fax	Redwood City CA	006800
Murata Business System	Dallas TX	00A200
NetExpress	Vienna VA	008400
TRW Electronic Proaucts	San Luis Obispo CA	004E00
Wang Laboratories	Lowell MA	008200
Xerox	Lewisville TX	009800

แฟ็กซ์โมเด็ม

ที่ตัวส่ง fax ตัว fax modem จะรับข้อมูลของภาพทาง digital ซึ่งถูกเข้ารหัสข้อมูลมาแล้ว จะนำมา modulate มาเป็นสัญญาณทาง analog และส่งเข้าสู่คู่สายโทรศัพท์ ที่ fax ทางด้านรับ modem ทางด้านรับจะแปลงจากสัญญาณทาง analog มาเป็นสัญญาณทาง digital ซึ่งแทนข้อมูลทางเอกสาร ที่ส่งมาจากทางด้านส่ง ตาม Recommendation T.4 จะระบุถึงกรรมวิธีในการ modulate และ demodulate ของ modem ซึ่งสร้างไว้ในเครื่อง fax ใน Group 3. โดยปกติตาม recommendation modem ใน V.27 ter จะใช้ส่งข้อมูล ข่าวสาร ของ fax โดยใช้ speed 4.8 และ 2.4 Kbit/sec ใน V.21 ใช้ในการส่งสัญญาณในการ handshake ด้วย 300 Bit/Sec ซึ่งจะทำการส่งก่อนและหลังการส่งข้อความ และเนื่องจาก speed ต่ำจึงไม่จำเป็นต้องใช้ line equalizer โดย mdem ทางด้านรับจะเป็น automatic equalizer อยู่แล้วและยังชดเชย amplitude distortion และ envelop-delay distortion จึงทำให้ลดอัตรา error ได้มาก ส่วน option modem V.29 จะใช้สำหรับส่ง fax ด้วย speed 9.6 หรือ 7.2 Kbit/Sec และ V.17 ใช้สำหรับส่ง fax ด้วย speed 14.4 หรือ 12.0 Kbit/Sec

fax Group 3 จะทำการตรวจสอบคุณภาพของคู่สายโทรศัพท์อย่างอัตโนมัติ ก่อนหน้าที่จะทำการส่งข้อมูล modem จะทำการส่งสัญญาณ TCF (Training Signal) ซึ่งทางด้านรับรู้ format นี้เช่นกัน โดยจะตรวจสอบที่ speed สูงที่สุด หากมี error มากจะลด speed ลงมา แล้วทดสอบคุณภาพสายอีกครั้งหนึ่ง โดยมีลำดับดังนี้เริ่มที่ V.29 speed 9,600 BPS แล้วลดมา 7,200 BPS ต่อมาเป็น V.27 ter ที่ speed 4,800 BPS แล้วลดมาเป็น 2,400 BPS แต่หากเครื่อง fax ตัวใดมี top rate ที่ 4,800 BPS ก็จะเริ่มที่นั้น modem ความเร็วสูงที่ได้รับการยอมรับจาก CCITT ล่าสุดเป็น V.17 ผ่านชุมสายโทรศัพท์ เช่นกันวิ่งด้วย speed 14.4 Kbit/Sec แบบ half-duplex โดยใช้ trellis code modem (TCM) และมีความสามารถลด speed ตัวเองได้ เป็น 12, 9.6 และ 7.2 Kbit/Sec ซึ่งทั้งหมดนี้ก็ใช้ TCM เช่นกันระบบ TCM จะเป็นเทคนิค forward error-correcting ซึ่งสามารถช่วยเพิ่ม Signal to noise ได้อีก 3-4 dB fax modem ที่ใช้ V.17 ก็ยังคงมี V.29, V.27ter และ V.21 เพื่อความ compatible กับ fax Group 3 ที่มีอยู่เดิม และล่าสุดนี้บริษัทผู้ผลิต modem ผลิต modem V.33 ซึ่งรวม V.17, V.29, V.27ter และ V.21 ไว้ในตัวเดียวกัน

ในการพัฒนาแฟ็กซ์จะต้องเกี่ยวข้องกับเทคโนโลยีการส่งข้อมูลด้วยโมเด็ม ในชั้นกายภาพ (Physical Layer) แฟ็กซ์โมเด็ม และดาต้าโมเด็มจะเหมือนกัน การไหลของข้อมูล การมอดูเลชั่น และการตรวจสอบการผิดพลาด (Errer Detective) วิธีที่ใช้ในดาต้าโมเด็มจะใช้กับแฟ็กซ์โมเด็ม

โดยทั่วไปแล้วระหว่างดาต้าและแฟ็กซ์โมเด็มอาจมีการเปลี่ยนแปลงข้อมูล การส่งภาพของแฟ็กซ์ หลังจากถูกเปลี่ยนเป็นเลขไบนารี การสื่อสารแฟ็กซ์ จะมีการแตกต่างเล็กน้อยสำหรับเปลี่ยนภาพบิทแม็ท (Bitmapped Graphice) ระหว่างซอฟแวร์ที่ทำงานร่วมกัน หลักการที่แตกต่างกันระหว่างโมเด็มที่ออกแบบเพื่อส่งดาต้า และแฟ็กซ์จะอยู่ในโพรโทคอลในส่วนการสื่อสาร แฟ็กซ์โมเด็มต้องรองรับโพรโทคอลที่บรรยายใน CCITT T.4, T.6 และ T.30 นอกจากนั้นแฟ็กซ์โมเด็มต้องเตรียมไว้กับคอมมานด์ใหม่ ๆ และตอบสนองซอฟท์แวร์รุ่นแรก ๆ และปฏิบัติในส่วนของแฟ็กซ์

โครงสร้างคลาส ETA

แฟ็กซ์โมเด็มจะพัฒนาได้จะต้องมีองค์การหลาย ๆ องค์การจึงทำให้แฟ็กซ์โมเด็มที่ผลิตไม่สามารถใช้กันได้ ดังนั้นแม้ว่าจะมีหลายบริษัทที่ผลิต ทำให้ผู้ใช้ต้องฝึกให้ชำนาญกับซอฟแวร์ที่มีมากมาย การใช้แฟกซ์

โมเด็มต้องการซอฟแวร์ที่สามารถใช้ได้กับโมเด็มที่ผลิตขึ้น ในการพัฒนาซอฟแวร์ต้องพยายามพัฒนาให้เป็นโปรแกรมควบคุมแฟ็กซ์ทั่ว ๆ ไป อาจเพิ่มเติมการรองรับโมเด็มได้กว้างขึ้นในภายหลัง ปัญหานี้ ETA ได้ถูกพัฒนาขึ้น ลำดับมาตราฐานที่กำหนดโพรโทคอลและคำสั่งสำหรับได้ใช้ระหว่าง DTE และแฟ็กซ์ DCE โดยจะแยกโมเด็มเป็น 3 คลาส กำหนดได้โดยใช้ความสามารถของโมเด็มในส่วนของแฟ็กซ์ที่เป็นอิสระกับ DTE ในส่วนนี้จะอธิบายถึง 3 คลาสของแฟ็กซ์โมเด็ม

การกำหนดคลาส

เครื่องแฟ็กซ์เป็นเครื่องมือที่มีเฟสต่าง ๆ จากรูปที่ เตรียมจะถูกเปลี่ยนเป็นข้อมูลผ่านช่องการสื่อสารและถูกเปลี่ยนเป็นภาพอีกครั้ง การทำงานส่วนนี้จะใช้ PC และแฟ็กซ์โมเด็ม ทั้งสองจะตอบสนองในการจัดการรายละเอียดระหว่างอุปกรณ์ 2 ตัว EIA สำหรับแฟ็กซ์โมเด็มทั้งคู่จะกำหนดการแยกและตั้งรายละเอียดของการอินเตอร์เฟส รูปที่แสดงการอินเตอร์เฟสในส่วนของแฟ็กซ์ได้ดีโดยการแบ่งคลาส

รูปที่ 8 EIA fax modem class interface partitioning

User Level	T.4/T.6	T.30	Physical Level
Image reduction and reproduction	image encoding and decoding	Fax session protocal	Modem V-protocals, data pump HDLC

EIA กำหนดคลาส 1 แฟ็กซ์โมเด็มใน EIA/TIA - 578 คลาส 1 จะหมายถึงความง่ายของแฟ็กซ์โมเด็มที่เตรียมทำงานเล็กน้อยที่จำเป็นสำหรับสนับสนุนกรุ๊ป 3 ของแฟกซ์ รูปที่ 8 แสดง PC จะตอบสนองการเข้ารหัสของรูป (ในT4) และจัดการสนับสนุนการสื่อสาร คลาส 1 แฟ็กซ์โมเด็มจะเตรียมสำหรับการทำงานที่ต้องการคือ

GSTN Interface

Autodialing

V serier Signal Conversion (Modulation)

Data Transmission and Reception

HDLC data framing , transparency and error detection

Control command และresponses

ส่วนของแฟ็กซ์ที่ใช้ในคลาส 1 โมเด็มต้องจัดการโดยใช้ซอฟท์แวร์ควบคุม การใช้ส่วนรับส่งข้อมูล การถอดรหัสและลำดับขั้นตอนที่ต้องการของ T.30 จะไม่ได้ผลในการควบคุมส่วนของแฟ็กซ์จะมีคำสั่งของมันเอง

คลาส 2 แฟ็กซ์โมเด็มจะรับผิดชอบหน้าที่มากกว่าคลาส 1 ดังในรูปที่ 8 โดยจะรับผิดชอบส่วนของแฟ็กซ์โดยคลาส 2 โมเด็มจะรับผิดชอบส่วนดังต่อไปนี้

GSTN Interface

Autodialing

V server signal conversion

T.30 protocol implementation

Session status reporting

Phase C data transfer

Padding for minimum scan line time

Quality check on receive data

Packet protocal for DTE/DCE interface

การออบแบบคลาส 2.0 (.0 จะมีการปรับปรุงระดับ) จะสามารถใช้เฉพาะแฟ็กซ์โมเด็มที่ยอมรับ TIA/EIA-592 ระหว่างการพัฒนาเวอร์ชั่นที่เป็นมาตราฐานจะถูกออกแบบโดย SP-2388 คลาส 2 โมเด็มจะทำหน้าที่ขณะเรียกเทอร์มินอล, จัดการส่วนของการสื่อสาร, การส่งข้อมูลแฟ็กซ์ และอาจจะทำการเลือกโดยเปลี่ยนระหว่าง T.4 (กรุ๊ป 3) และ T.6 (กรุ๊ป 4) รูปแบบของข้อมูล เครื่อง PC จะตอบสนองเตรียมการและบีบอัดข้อมูลสำหรับการสื่อสาร และแปลงกลับที่เครื่องรับ เครื่อง PC จะเปลี่ยนข้อมูลที่เป็นรูปภาพทันทีโดยโมเด็ม คำสั่งอินเตอร์เฟสของคลาส 2 โมเด็มไม่จำเป็นต้องคอมเพ็กทีเบิลกับคลาส 1 โมเด็ม

คลาส 3 แฟ็กซ์โมเด็มถูกใช้ในการเรียน มันถูกวางแผนให้ทำงานจากเครื่อง PC ไปโมเด็มแสดงในรูปที่ 8 หน้าที่ที่เพิ่มขึ้นจาก T.30 และหน้าที่ทางกายภาพของโมเด็มคลาส 3 จะทำการเปลี่ยนไฟล์ของข้อมูลภาพใน T.4 หรือ T.6 โดยบีบอัดข้อมูลสำหรับส่ง โมเด็มอาจจะทำการขยายข้อมูลที่ถูกบีบอัด โดยคลาส 3 นี้จะถูกพัฒนาโดย EIA

EIA FAX Command

คำสั่ง ETA สำหรับควบคุม คลาส 1 และ คลาส 2 จะถูกออกแบบให้ขยายจากชุดคำสั่ง AT (AT Command Set) คำสั่งในดาต้าโมเด็ม ตัวอักษรที่ส่งโมเด็มถูกเรียกว่า คอมมานด์ไลน์ (Command Live) และต้องเริ่มต้นโดยตัวอักษร AT หรอ at คอมมานด์ไลน์จะเป็นตัวอักษร ASCII และจะลบคำสั่งเพื่อเป็นตัวอักษณ CR (Carriage Return)

ตัวอักษณเหล่านี้จะถูกตรวจสอบและแปลงเป็นคำสั่งช่องว่าง และตัวอักษรควบคุมอื่น ๆ เช่น ASCII 13 (Carriage Return) และ ASCII 8 (Backspace) ที่ปรากฏในคอมมานด์ไลน์จะไม่ผิด

Class 1 Synax

คำสั่ง EIA เพื่อควบคุมแฟ็กซ์ จะเริ่มต้นด้วยตัว +F สามรูปแบบทั่ว ๆ ไปของคำสั่ง ขึ้นอยู่กับรายละเอียดของคำสั่ง ทั้งสามคำสั่ง คือ ความสามารถ (Capabilities), สถานะ (Status) และจัดระบบ (Set)

การกำหนดความสามารถของโมเด็ม ประโยคคำสั่งความสามารถที่ที่ใช้ จะมีรูปแบบ

+F command =?

โมเด็มจะตอบสนองโดยแสดงขอบเขตที่ตัวเองตอบสนองคลาส 1 โมเด็ม ตอบสนองคอมมานด์ไลน์ AT+F CLASS ? จะบอกเราให้ทราบคลาสที่ตอบสนอง จากคำสั่งจะตอบสนอง

0,1

การตอบสนองจะแสดงเป็น 1 เมื่อสามารถตอบสนองคลาส 1 และ คลาส 0 จะเป็นเฉพาะดาต้าโมเด็ม

คำสั่งที่สองที่จะบอกค่าของพารามิเตอร์ที่ทำงานโดยคำสั่ AT+F class ? โมเด็มที่ทำงานในคลาส 1 ก็จะตอบสนองแสดงข้อความ

1

คำสั่งที่ถูกใช้กำหนดคลาสเพื่อควบคุมการทำงานของโมเด็มการให้ทำงานในคลาสที่กำหนดคือ

+F Commend Val

โดยจะแทนค่า Val ด้วยค่าที่เป็นตัวเลขหรืออักขระ ถ้าเป็นตัวเลขก็จะแทนดาต้าโมเด็ม AT Command และคลาส1 เมื่อมีค่าเป็น 1 คอมมานด์ไลน์จะสิ้นสุดเมื่อมี CR หรือไม่ก็ลงท้ายด้วยเครื่องหมาย ; ยกเว้นคำสั่ง +FTS และ +FRS

แฟ็กซ์โมเด็มอาจจะไปรบกวนตอบสนองไม่เป็นอักขระ หรือตัวเลย ตัวอักขระจะตอบสนองเมื่อตามหลังด้วย CR และไลน์ฟีด (Line Feed) ตัวเลขจะตอบสนองเมื่อตามหลังด้วย CR ผลที่ได้ของรหัสคือ OK (0) , CONNECT (1) , No CARRIER (3) และ ERROR (4)

Class 2 Syntax

EIA คลาส 2 จะมีจุดที่สำคัญเหมือนกันที่กำหนดในคลาส 1 โดยทั้งหมดขอการเริ่มต้นด้วย +F โดยจะมีคำสั่งทั้ง ๆ ไป 3 คำสั่ง (เหมือนกับคลาส 1) สำหรับคลาส 2 การใช้คำสั่งจัดค่าพารามิเตอร์ โดยจะรองรับตัวเลขหรือสตริงไม่เหมือนกับคลาส 1 อย่างไรก็ตามคลาส 2 ตัวเลขต้องเป็นเลขฐานสิบหกตัวเลขที่ทำจะเป็นตัวอักษร "0" ถึง "9" (ASCII 30H ถึง 39H) และ "A" ถึง "F" (ASCII 41 h ถึง 46 h) สตริงจะประกอบด้วยตัวอักษร ASCII และมีตัวอักษร (") ขั้นท้ายและหน้าของสตริง

การตอบสนองอาจจะมีค่าหนึ่งค่า หรือหลายค่า หรืออาจจะขอบเขตของค่า เพื่อให้เข้ากันได้กับคลาส 1 โมเด็ม คลาส 2 จะตอบสนองคำสั่ง AT+ F Class =? โดยไม่เปลี่ยนแปลงในการแสดงค่าหลาย ๆ ค่า โดยจะแยกค่าด้วยอักษรคอมม่า (Comma) (,) ตัวอย่างการตอบสนอง เช่น (0, 2, 4, 8) ค่าที่เป็นขอบเขตทั้ง 2 ค่าจะขั้นด้วยเครื่องหมาย - (Hyphen) ค่าขอบเขตนี้จะมีค่า 0 ถึง FF คำสั่งคลาส 2 อาจประกอบด้วยค่าต่าง ๆ ตามลำดับอยู่ภายในวงเล็บขั้นด้วยคอมม่า (,) โดยช่องว่างจะถูกละทิ้ง

(0 , 1, 2) , (0), (0 - 3)

คำสั่งคลาส 2 จะกระทำตามคอมมานด์ไลน์ จากซ้ายไปขวา แต่ละคำสั่งจะมีการกระทำเฉพาะโดยจะไม่พิจารณาอะไรก็ตามที่เกิดขึ้นภายหลัง ถ้าคำสั่งทั้งหมดถูกต้องก็จะมีค่าแสดงผลสถานะสุดท้าย คำสั่งแสดงผลผิดพลาด (Error) หรือถ้าคำสั่งโมฆะถูกพบการกระทำของคอมมานด์ไลน์หยุด และไม่ทำคำสั่งใด ๆ คลาส 2 จะตอบสนองผล

0 OK

1 CONNECT

3 RING

4 ERROR

6 NO DIALTONE

7 BUSY

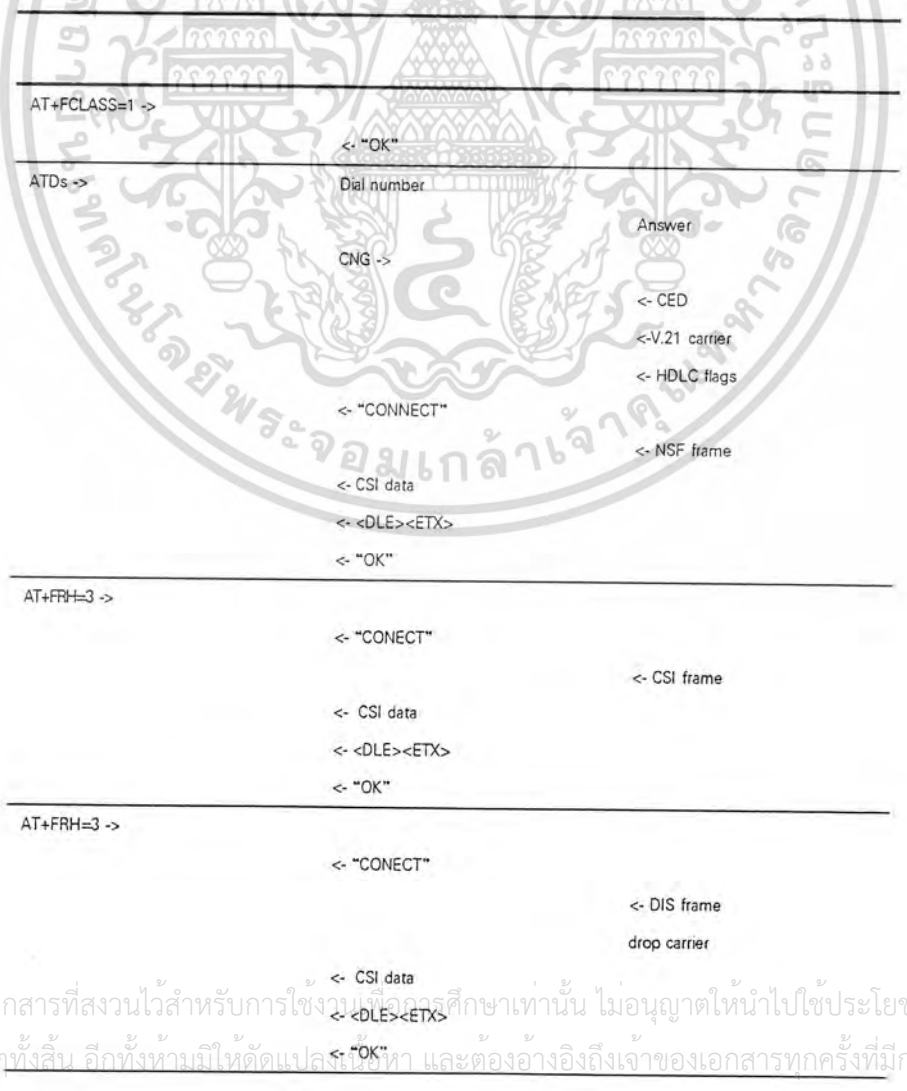
8 NO ANSWER

EIA/TIA -578 (SP-2388B) จะเป็นชุดคำสั่งที่จำเป็นในการปฏิบัติในส่วนของแฟกซ์ที่ใช้โดยแฟกซ์โมเด็ม

การทำงานคลาส 1

คลาส 1 ต้องการการทำงานที่ดีจากเครื่อง PC (DTE) การควบคุมและดาต้าต้องถูกจัดรูปแบบโดยเครื่อง PC และเมื่อถูกส่งไปที่โมเด็มอย่างเดียวกัน ข้อมูลที่รับมาโดยโมเด็มจะถูกผ่านไปที่ เครื่อง PC และผ่านการถอดรหัสและแปล รูปที่ 9 แสดงลำดับของส่วนแฟกซ์ที่คลาส 1 โมเด็มเป็นผู้เริ่มเรียก และส่งข้อมูล 1 หน้ากระดาษโดยปราศจาก error

รูปที่ 9 A simple Class 1 fax session sending a signal page to a remote fax terminal



รูปที่ 9 (ต่อ) A simple Class 1 fax session sending a signal page to a remote fax terminal

AT+FRH=3 ->		
	<- "NO CARRIER"	
AT+FTS=20 ->	(200 ms of silence)	
AT+FTH=3 ->	V.21 carrier ->	
	<- "CONNECT"	
TSI data ->		
<DLE><ETX> ->	TSI fyrame ->	
	<- "CONNECT"	
DCS data ->		
<DLE><ETX> ->	DCS frame ->	
	drop carrier	
	<- "OK"	
AT+FTS=8	wait 80 ms	
AT+FTM=96 ->	V.229 carrier ->	
	<- "CONNECT"	
TCF data ->		
<DLE><ETX>	TCF frame ->	
	drop carrier	
	<- "OK"	
AT+FRH=3 ->		<- V.21 carrier
	<- "CONNECT"	
		<- CFR frame
		drop carrier
	<- CFR data	
	<- <DLE><ETX>	
	<- "OK"	
AT+FRH=3 ->		
	<- "NO CARRIR"	
AT+FTS=20 ->	(200 ms of silence)	
AT+FTM=96 ->	V.29 carrier ->	
	<- "CONNECT"	
<image data> ->		
<DLE><ETX> ->	<image data> ->	
	drop carrier	
	<- "OK"	
AT+FTS=8 ->	wait 80 ms	
AT+FTH=3 ->	V.21 carrier ->	
	<- "CONNECT"	
EOP data ->		
<DLE><ETX> ->	EOP frame ->	
	drop carrier	
	<- "OK"	

รูปที่ 9 (ต่อ)

AT+FRH=3 ->		<- V.21 carrier
	<- "C2ONNECT"	
		<- MCF frame
	<- MCF data	drop carrier
	<- <DLE><ETX>	
	<- "OK"	
AT+FRH=3 ->		
	<- "NO CARRIER"	
AT+FTS=20 ->	(200 ms of silence)	
AT+FTH=3 ->	V.221 carrier ->	
	<- "CONNECT"	
DCN data ->	DCN frame ->	
	drop carrier	
	<- "OK"	
ATHO ->	hang up	

ส่วนเริ่มเมื่อเครื่อง PC ตั้งค่าโมเด็มเป็นโหมดแฟ็กซ์ และไดแอ็ล ไปยังเลขหมายของแฟ็กซ์ปลายทาง โมเด็มจะส่งสัญญาณ CNG (Calling) ความถี่ 1,100 Hz 0.5 วินาที และหยุด 3 วินาที แฟ็กซ์ปลายทางจะตอบสัญญาณ CEO (Called Station Identification) สัญญาณต่อเนื่องความถี่ 2,100 Hz ประมาณ 2.6 ถึง 4 วินาที

แฟ็กซ์ปลายทางจะสร้างความถี่สำหรับ 3000 bps V.21 และส่ง HDLC มาเมื่อ HDLC กับมาได้ก็จะทำให้โมเด็มตอบข้อความ Connect กับเครื่อง PC

โมเด็มจะรับเฟรม HDLC จากแฟ็กซ์ปลายทางโดยจะส่งข้อมูลดังแสดงในรูปที่ 10 ไปเครื่อง PC โดยไบท์แรกจะเป็นที่อยู่ที่ซึ่งจะเป็น FFh สำหรับการสื่อสารบน GSTN

รูปที่ 10 ตัวอย่างแฟรม NSF

FF	03	20	...data...		3F6 C3	10 03
HDLC address field	HDLC control field - indicates not the last frame	Facsimile control field -identtifies NSF	NSF	FCS	DLE/ETX	

เป็นส่วนของการควบคุม (Control Filed) โดยจะสมมุติให้มีค่าใดค่าหนึ่งใน 2 ค่า คือ C0h จะบอกว่าแฟรมนี้ไม่ใช่แฟรมสุดท้าย ยังมีแฟรมถัดมาอีก ถ้าเป็น C8h จะบอกว่าแฟรมนี้เป็นสุดท้ายอย่างไรก็ตามไบท์ที่ถูกปล่อยจากเครื่อง PC จะเป็นบิทที่ถูกกลับลำดับ (Bit Order Reversed) กับที่แสดงใน T.30 เช่น O3h (00000011b) จะกลายเป็น C0h (11000000b)

ไบท์ถัดไป คือ FCF (Facsimile Control Field) และพิสูจน์ชนิดของข้อมูลที่ถูกเปลี่ยนไปก่อนที่จะรับ ก็ต้องมีการกลับลำดับใน T3D เช่น O4h โดยจะบอกการทำงาน NSF จะเป็นรูปแบบเฉพาะของแต่ละผลิตภัณฑ์ ที่ไม่มีใน T.30

ที่ตามมาก็คือแฟรม 16 บิท CRC (Check FCS) ที่ส่งไปยังเครื่อง PC โดย FCS ที่ถูกส่งไปจะมี 2 ไบท์ ไบท์สูงจะถูกส่งก่อนไบท์ต่ำ แม้ว่าบิทแต่ละไบท์จะถูกกลับลำดับ แฟรม FCS จะเป็นข้อมูลอย่างเดียว เครื่อง PCไม่จำเป็นต้องตรวจสอบ FCS โมเด็มจะคำนวณแฟรม FCS และเปรียบเทียบกับ FCS ที่รับเข้ามากับแฟรมสุดท้ายจะถูกปิดด้วย 2 ไบท์ของลำดับ คือ 10 h (DLE) 03h (ETX)

ในตัวอย่างถัดไปในรูปที่ 9 การทำงาน CSI เครื่อง PC จะอ่านโดยใช้คำสั่ง AT+FRH=3 (อ่าน HDLC frequency ใช้ 300 bps V.21 class 2 module) แฟรมนี้จะตามด้วยเฟรมทั่ว ๆ ไป คือ NSI แฟรม CSI แฟรมจะตรวจสอบโดยไฟล์ควบคุมจะเป็น 02h (จะเป็น 40h ที่เครื่อง PC)

ในหลายกรณีตัวอักษร ASCII ที่แทนเบอร์โทรศัพท์ที่แฟ็กซ์ด้านรับรหัส ASCII ที่ถูกส่งตัวสุดท้ายก่อน โดยบิทจะไม่ถูกกลับ เช่น เบอร์เป็น “800-555-1212” ของเครื่อง PC ที่รับจะเป็น 2121-535-008” และตามด้วย FCS และ DCE/EIX รูปแฟรมที่อ่านได้ถัดไปแสดงในรูปที่ 11ในกรณี FFh เป็นที่อยู่และตามด้วยไฟล์ควบคุม C8h (13h ถูกกลับ bit โดย PC) โดย PC จะเป็นแฟรมที่บอกที่อยู่ของแฟ็กซ์ปลายทางด้วย ไบท์ทั่วไป FCF คือ 01 (ถูกกลับเป็น 80 h) DIS (Digital Identification Signal) คือแฟรมที่ตามมา DIS จะมีบิทต่ำสุด คือ 24 บิท (3 ไบท์) จะบอกถึงความสามารถของแฟ็กซ์ด้านรับข้อมูลที่ถูกใช้ โดยการส่งแฟ็กซ์ที่กำหนดค่าที่เหมาะสม

รูปที่ 11 ตัวอย่างแฟรม DIS

FF	13	80	00 CE B8 00	62 28	10 03
HDLC address field	HDLC control field - indicates not the last frame	Facsimile control field -identtifies NSF	DIS data	FCS	DLE/ETX

ไบท์แรกของ DSI เป็นข้อมูลของกรุ๊ปของแฟ็กซ์ ว่าเป็นกรุ๊ป1หรือ2 สามารถละทิ้งได้ ก็จะเป็นกรุ๊ป 3 ต่อจากนั้น3 ไบท์ถัดมาเป็นของ DCS (CEh B8h 00h) ต้องถูกขยายและกลับบิท

เมื่อเครื่อง PC รับ NO CARRIER จากคำสั่ง AT + FRH = 3 โมเด็มจะเตรียมพร้อมจะหมุนเบอร์อีกครั้ง ถ้ารับข้อความ Connect เครื่อง PCจะส่ง TSI (Transmitting Subscriber Identification) ข้อมูลไปโมเด็ม เครื่อง PC ไม่สามารถคำนวณและส่ง FCS จากแฟรม จะเป็นส่วนรับผิดชอบของโมเด็ม แฟรม TSI โดยปกติจะเป็นเลขของผู้เรียก (Calling Station) และอาจจะถูกใช้โดยโมเด็มตัวรับเพื่อจะเพิ่มความปลอดภัยของข้อมูล ถัดจากนั้นเครื่อง PC จะส่ง DCS (Digital Command Signal) ที่เลือกความสามารถจาก
แฟรม DIS ที่จะติดต่อกัน จากรูปจะเห็นว่าเครื่อง PC จะทำงานใน V.29 คือ 9600bps หลังจากนั้นเครื่อง PC จะให้โมเด็มหยุด 80 ms เพื่อให้โมเด็มปรับตัวมันเอง

หลังจากโมเด็มปรับเป็น V.29 แล้วก็จะส่งข้อความ connect เครื่อง PC ก็จะส่ง TCF (Traning Check) เพื่อให้สัญญาณดิจิตอลเกิดการซิงค์โคนไนซ์ และการกำหนดความเร็วการส่งข้อมูลในสาย โดยเครื่อง PC จะรอ CFR (Confirmation to Receive) จากโมเด็มที่ 300 bps การรับแฟรมนี้จะต้องสมบูรณ์ การสื่อสารข้อมูลจริง ๆ จึงจะเริ่ม

ขึ้น หากเกิดปัญหาขึ้นก็จะทำการลดค่าอัตราการสื่อสารข้อมูลลง ความเร็วที่สามารถติดต่อกันได้ถูกต้องแล้ว เครื่อง PC ก็จะส่งข้อมูลไปให้โมเด็ม โดยควบคุมการไหลของข้อมูลระหว่าง เครื่อง PC กับโมเด็ม วิธีการเข้ารหัสจะเป็นตามใน T.4 ด้านรับจะสามารถทำการซิงค์โคไนซ์ใหม่ เมื่อเกิดการผิดพลาด แต่ก็จะมีตัวตรวจสอบความผิดพลาดข้อมูล (Data Error Detection) และควบคุมการทำงาน หลังจากส่งข้อมูลเสร็จ 1 หน้าไม่มีหน้าต่อไป เครื่อง PC จะส่ง EOP (End of Procedures)

เครื่อง PC จะรู้คำตอบของแฟ็กซ์ปลายทาง โดยถ้าการส่งถูกต้องก็จะส่ง MCF (Message Confirmation) ให้กับโมเด็ม เครื่อง PC ก็จะส่งคำสั่ง AT+FTH = 3 ให้โมเด็ม และส่ง DCN (disconnect) เพื่อออกจาก

Class 2 Operation

แสดงในรูปเริ่มต้นจัดโมเด็มอยู่ในคลาส 2 PC ส่งคำสั่ง ATD ไดแอ็ดไปยังแฟ็กซ์ปลายทาง

รูปที่ 12 ตัวอย่าง คลาส 2

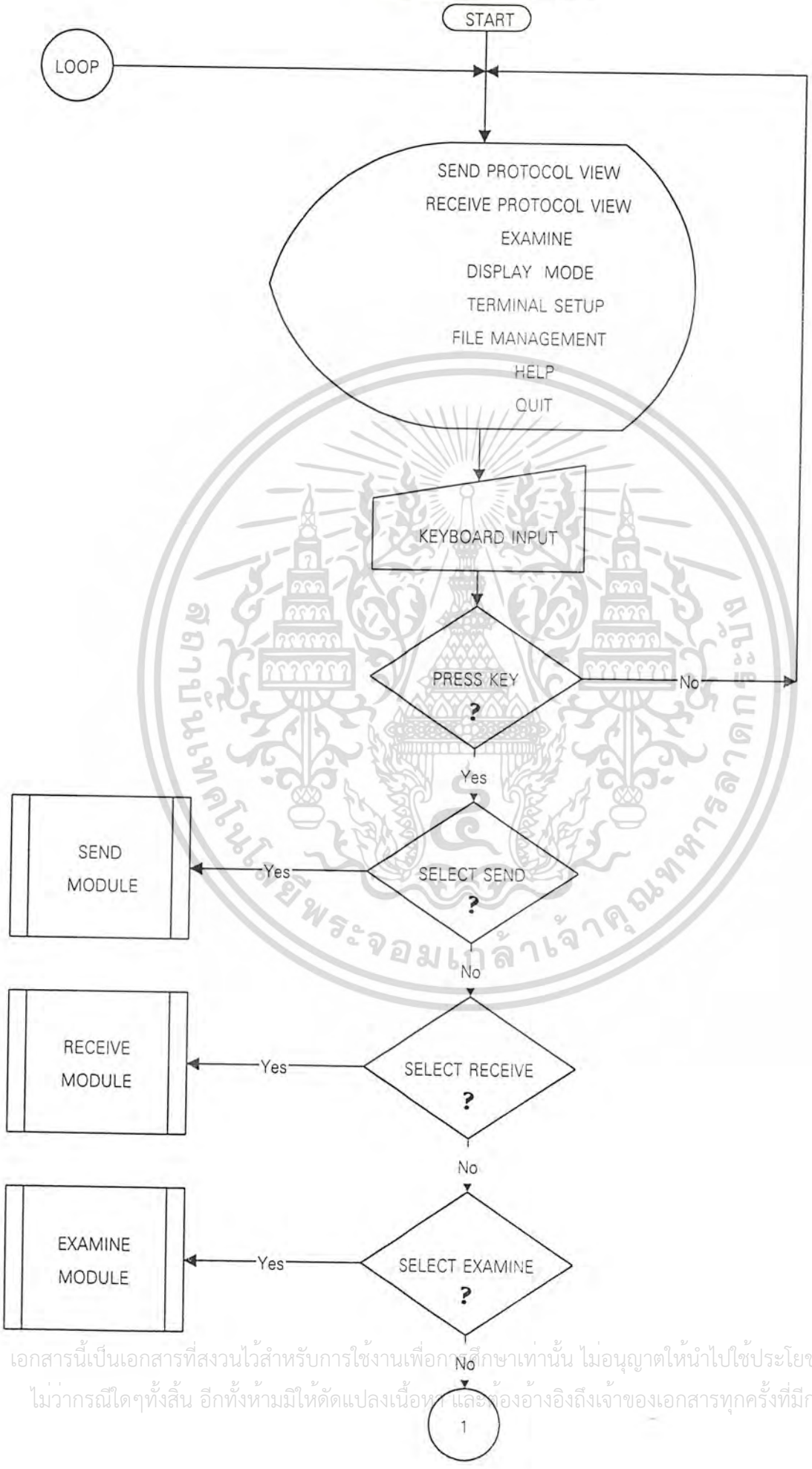
AT+FCLASS=22.0 ->		
	<- "OK"	
ATDs ->	dial number	
		answer
	CNG ->	
		<- CED
	<- "+FCO"	<- HDLC flags
	<- "+FNF:*nsf*"	<- NSF frame
	<- "+FCI:*csi*"	<- CSI frame
	<- "+FIS:*dis*"	<- DIS frame
	<- "OK"	
AT+FDT ->	TSI frame ->	
	DCS frame ->	
	TCF frame ->	
	<- "+FCS:*cfr*"	<- CFR frame
	<- "CONNECT"	
<image data> ->	<image data> ->	
<DLE><2Eh> ->	EOP frame ->	
	DCN frame ->	<- MCF frame
	<- "OK"	
ATHO ->	hang up	hang up

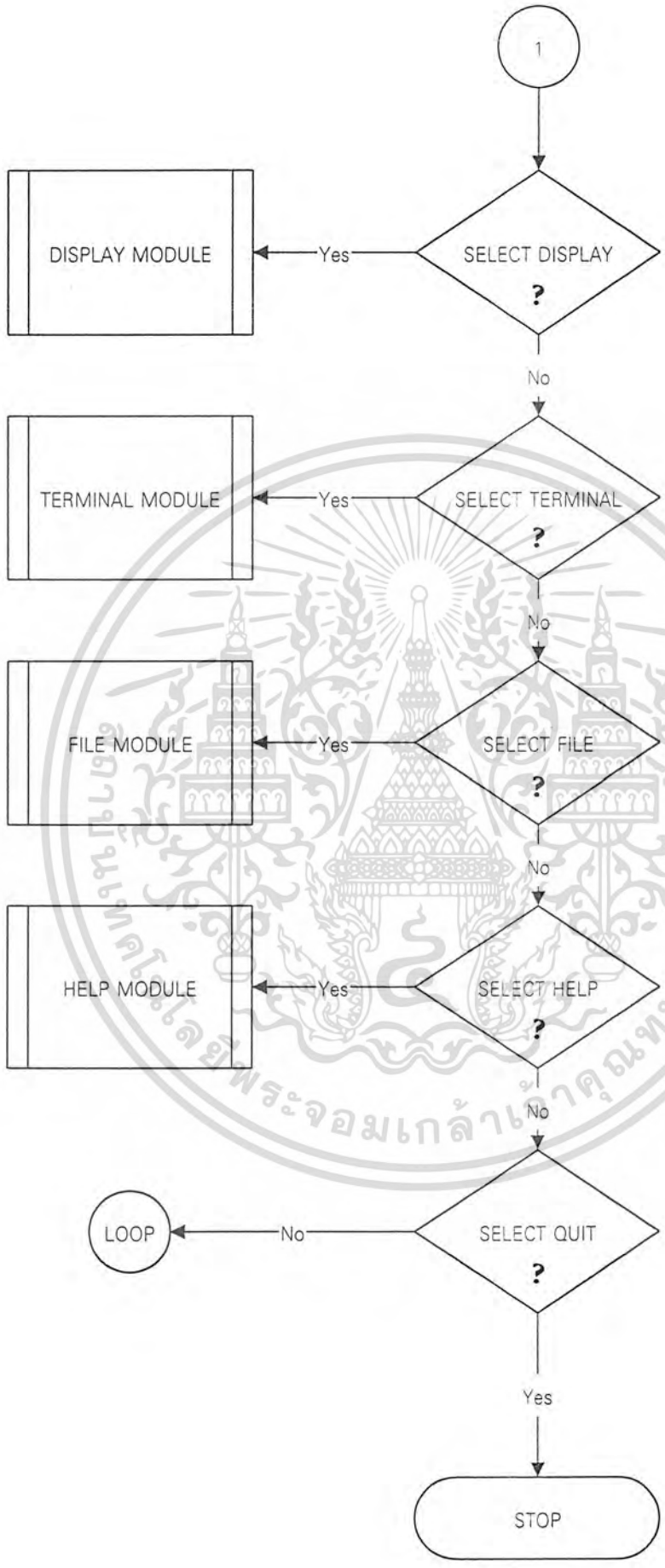
เมื่อต่อกันได้ก็จะส่ง + FCO ตามไปให้ PC จากนั้นโมเด็มจะแลกเปลี่ยน NSF, CSI และ DSI frame ในแต่ละขั้นตอนแฟรมจะถูกส่งโดยตัวโมเด็ม และสตริงแบบ ASCII จะถูกส่งไปให้ PC เป็นผลลัพท์เฉพาะของ DIS แฟรมจะถูกกระจายและส่งในเครื่อง PC เป็นลำดับของตัวอักษรซึ่งแยกกันด้วยเครื่องหมายคอมม่า

เครื่อง PC จะสั่ง AT+FDT โมเด็มจะทำให้พารามิเตอร์ต่าง ๆ เข้ากันได้ รวมทั้งค่าความเร็วในการส่งข้อมูล หลังจากเชื่อมโยงเสร็จก็จะส่ง Connect ให้กับเครื่อง PC เครื่อง PC ก็จะส่งข้อมูลไปที่โมเด็ม

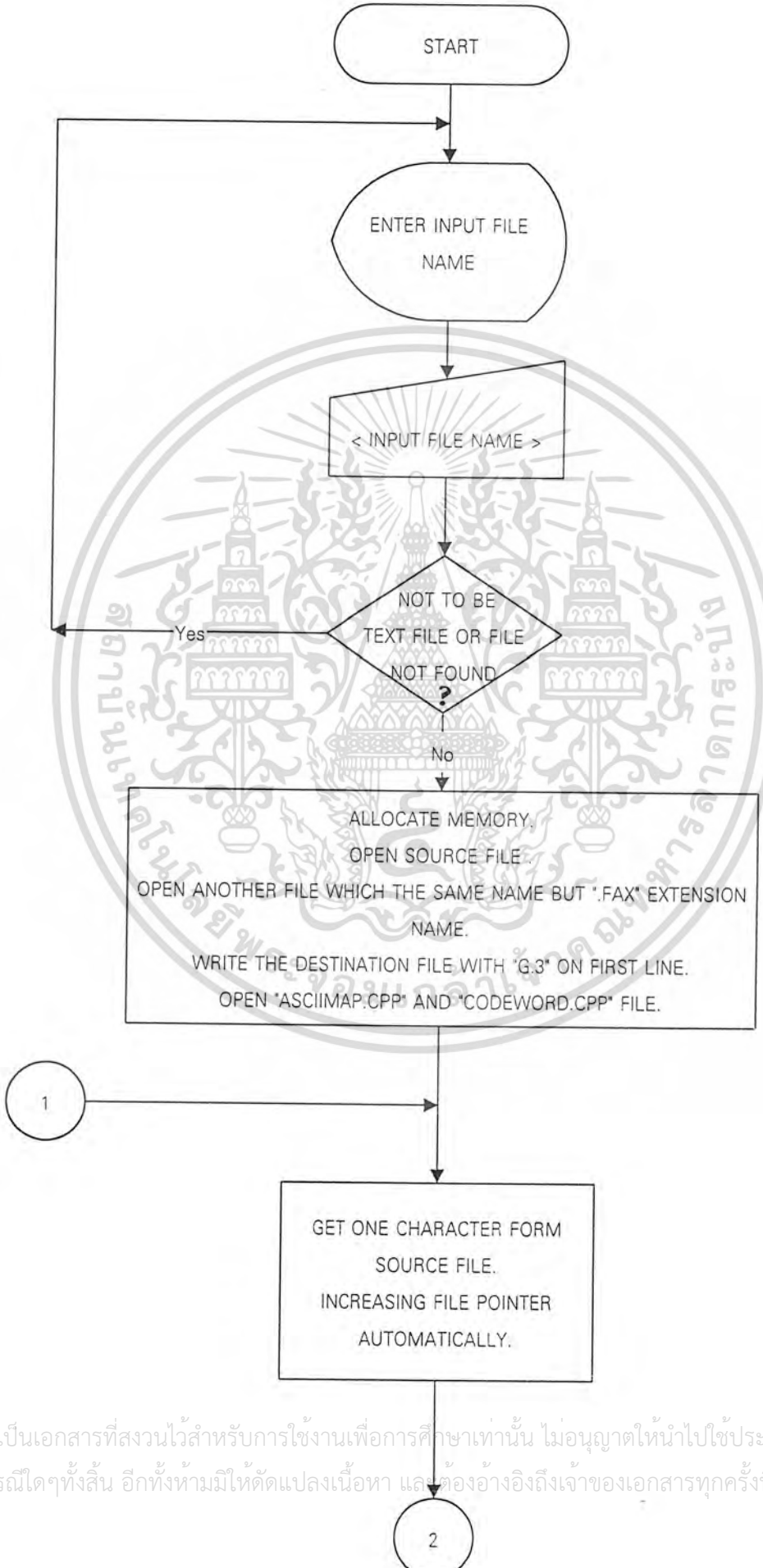
เครื่อง PC จะส่ง <DLE><2EH> ต่อท้ายข้อมูลเพื่อบอกให้โมเด็มว่าเป็นหน้าสุดท้ายของการส่ง โมเด็มก็จะส่ง EOP ไปให้แฟ็กซ์ตัวที่ติดต่อ หลังจากนั้นก็ส่ง DCN ไปและส่งข้อวาม OK ให้เครื่อง PC เครื่อง PC ก็จะสั่งวางหูโดยใช้คำสั่ง ATH0

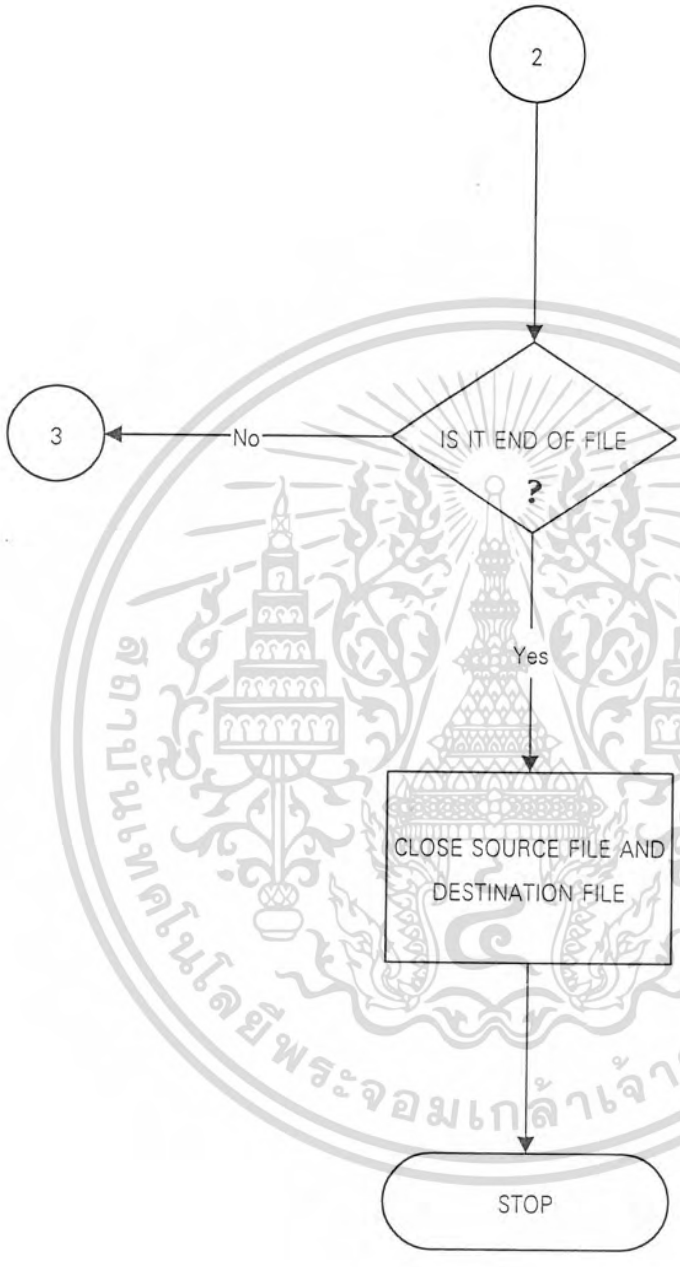
Flow Chart Of Main Program

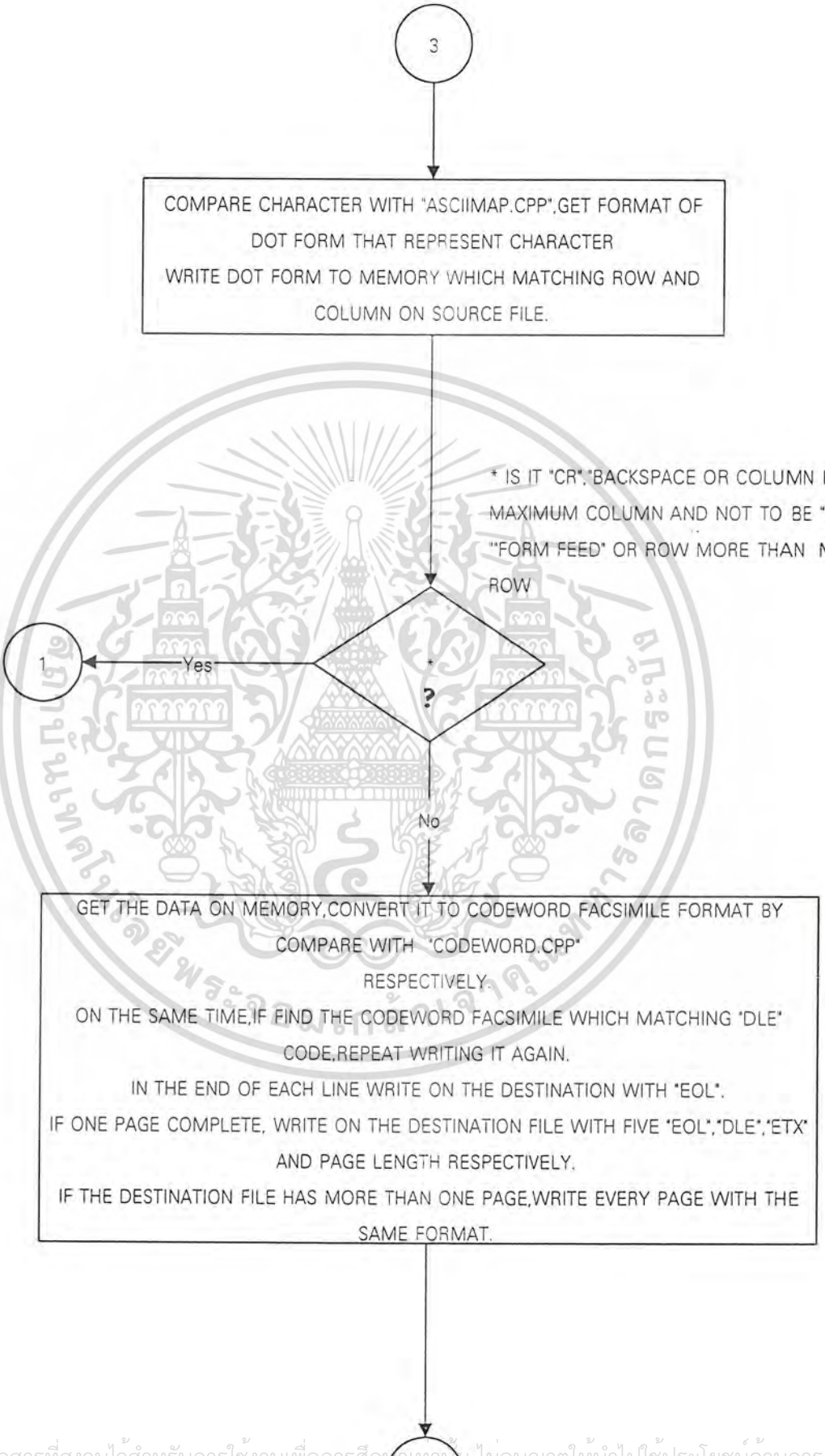


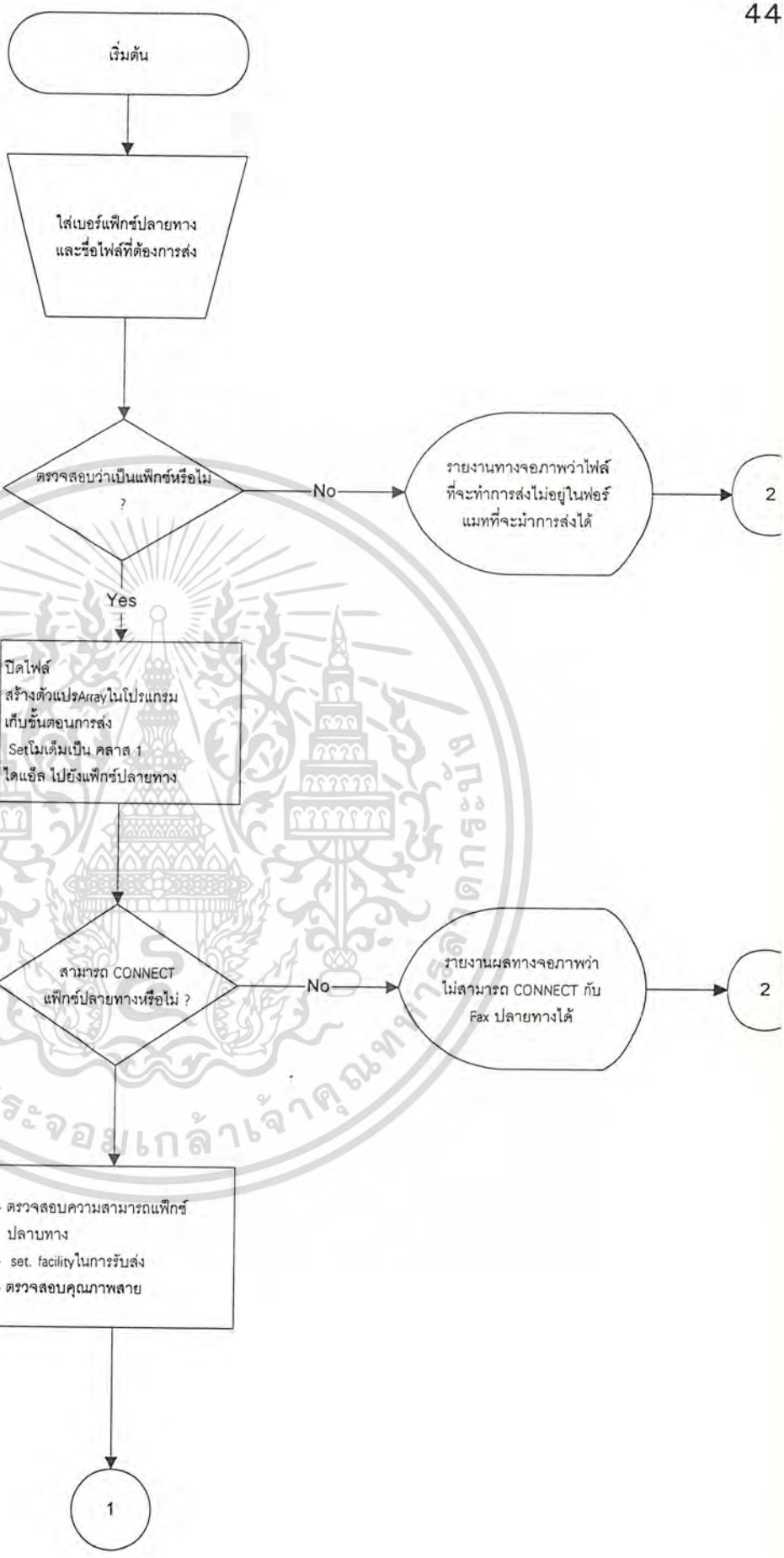


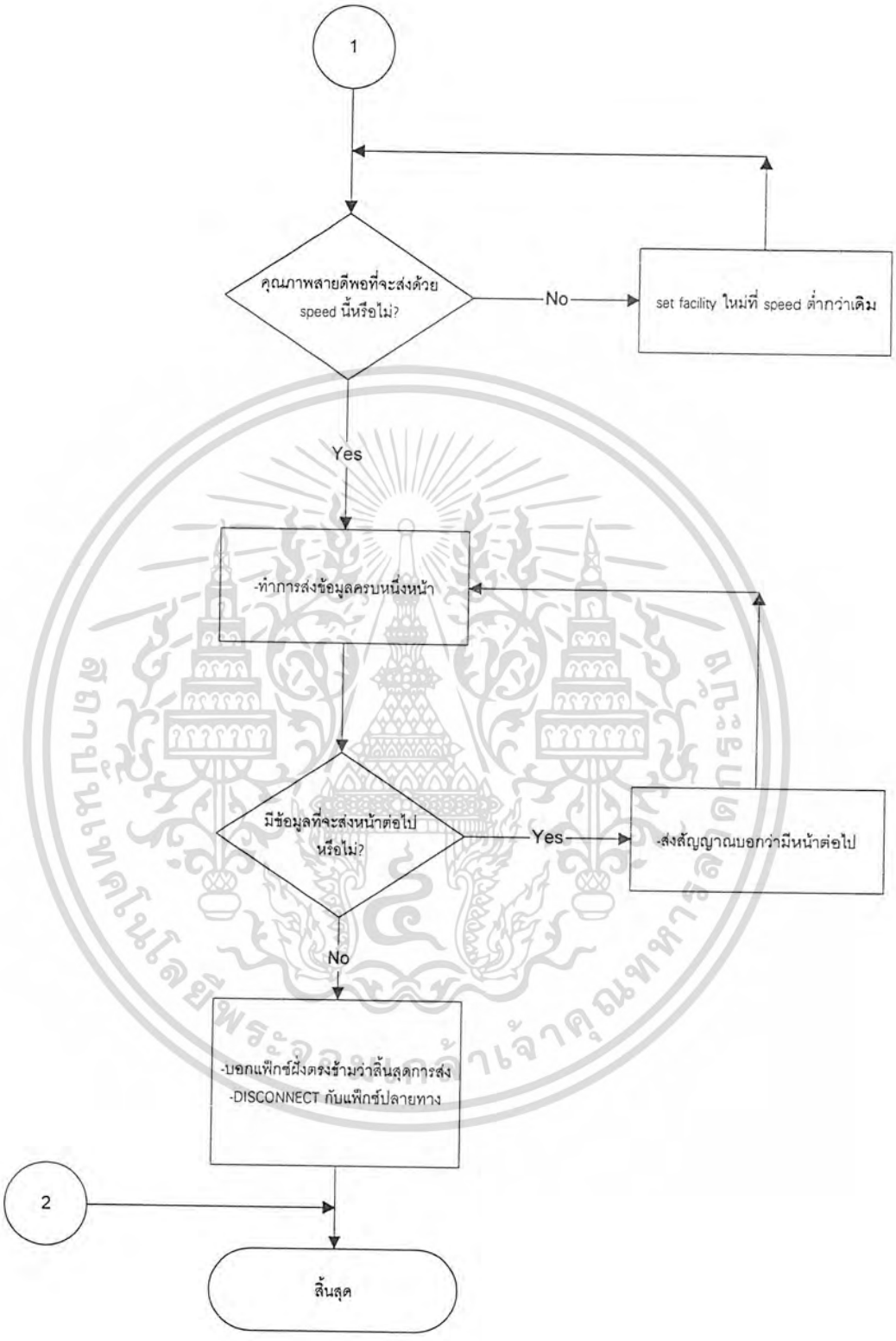
FILE CONVERTED MODULE FLOW CHART

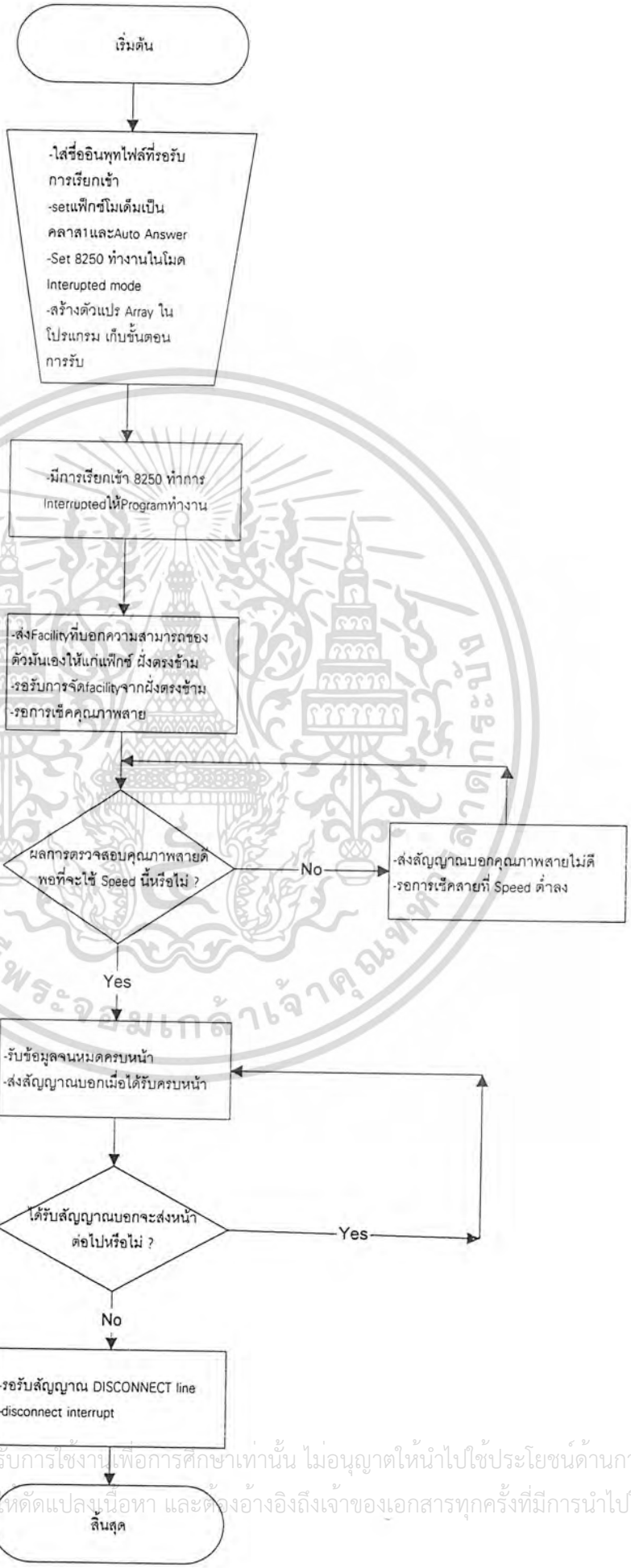




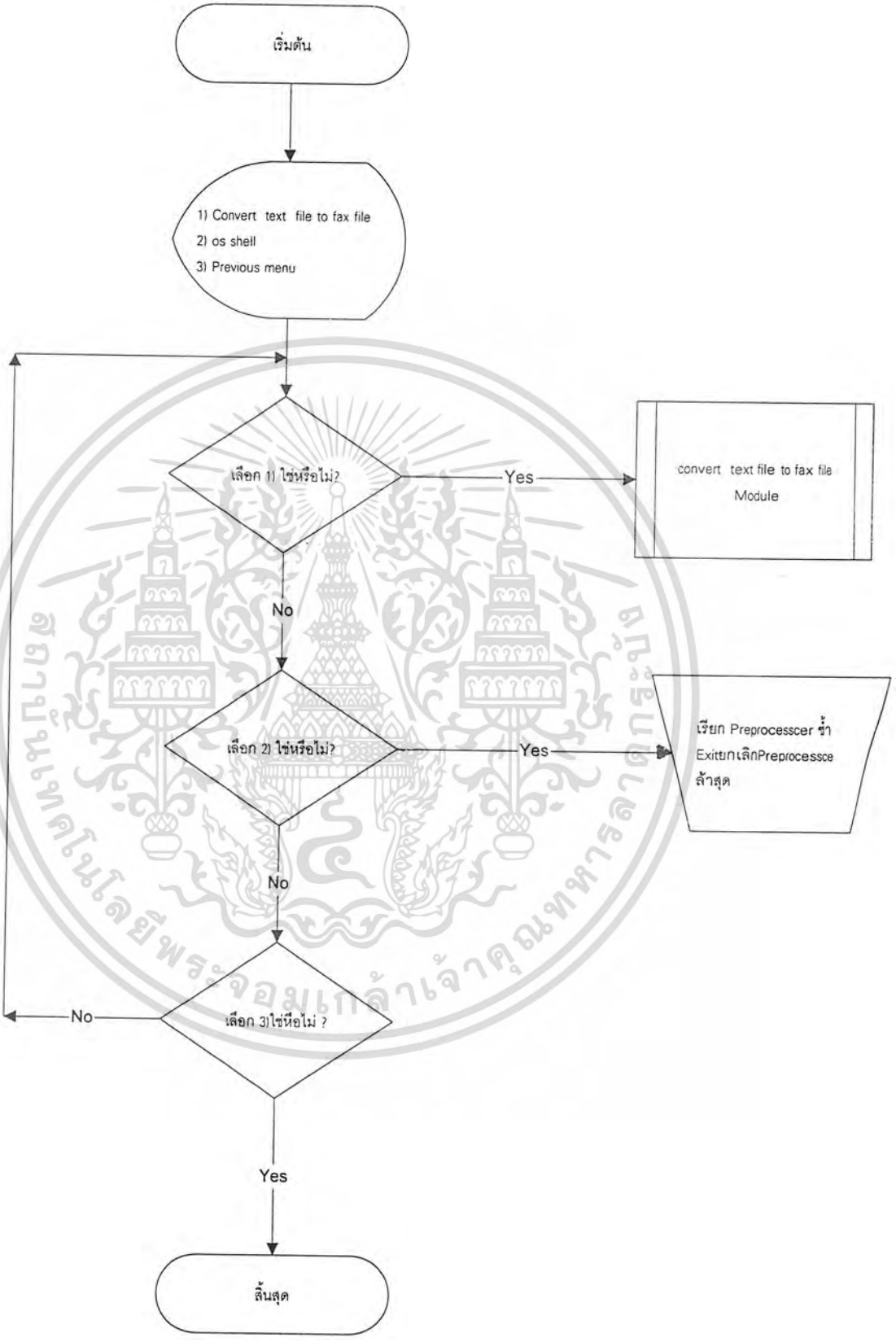




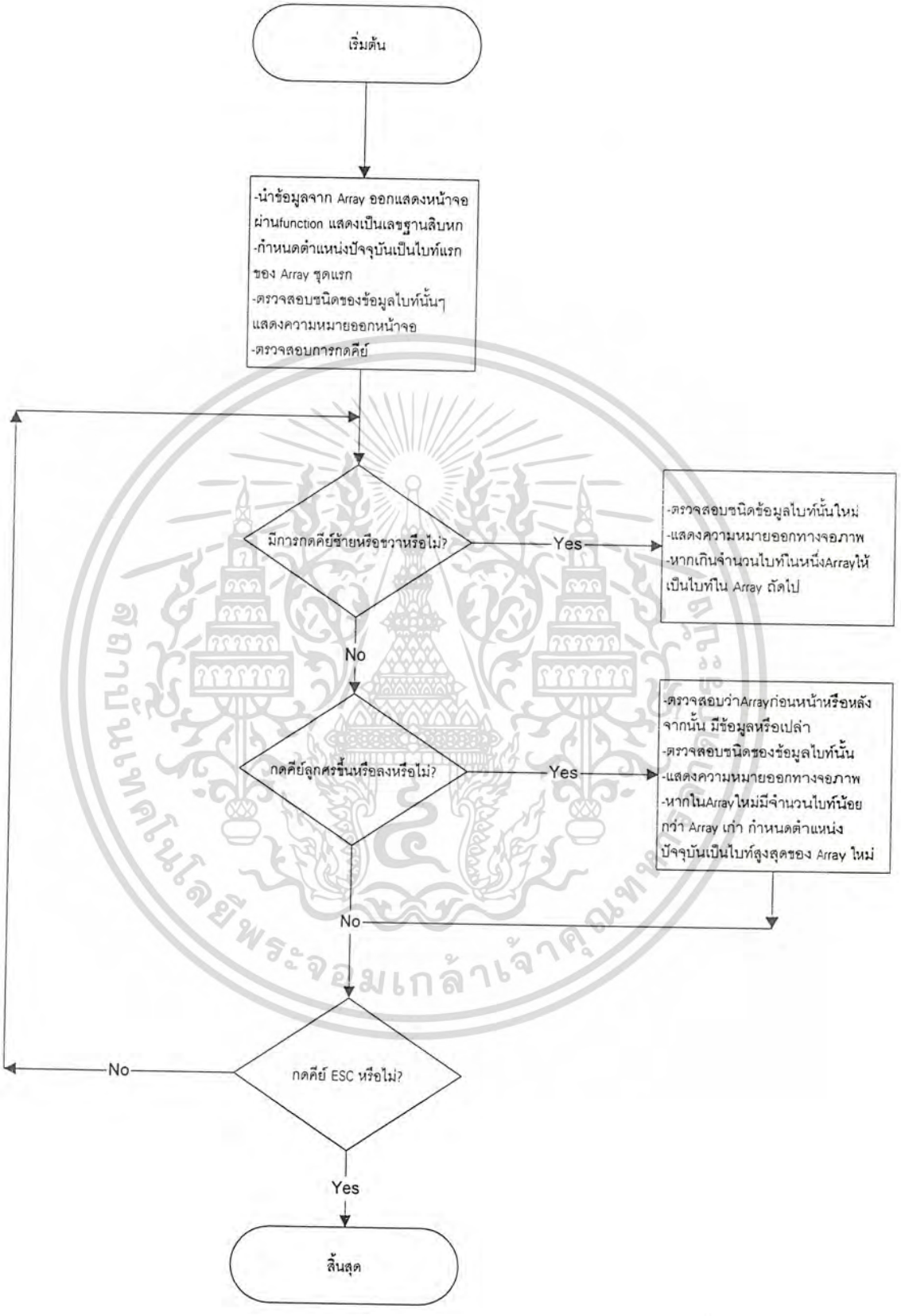




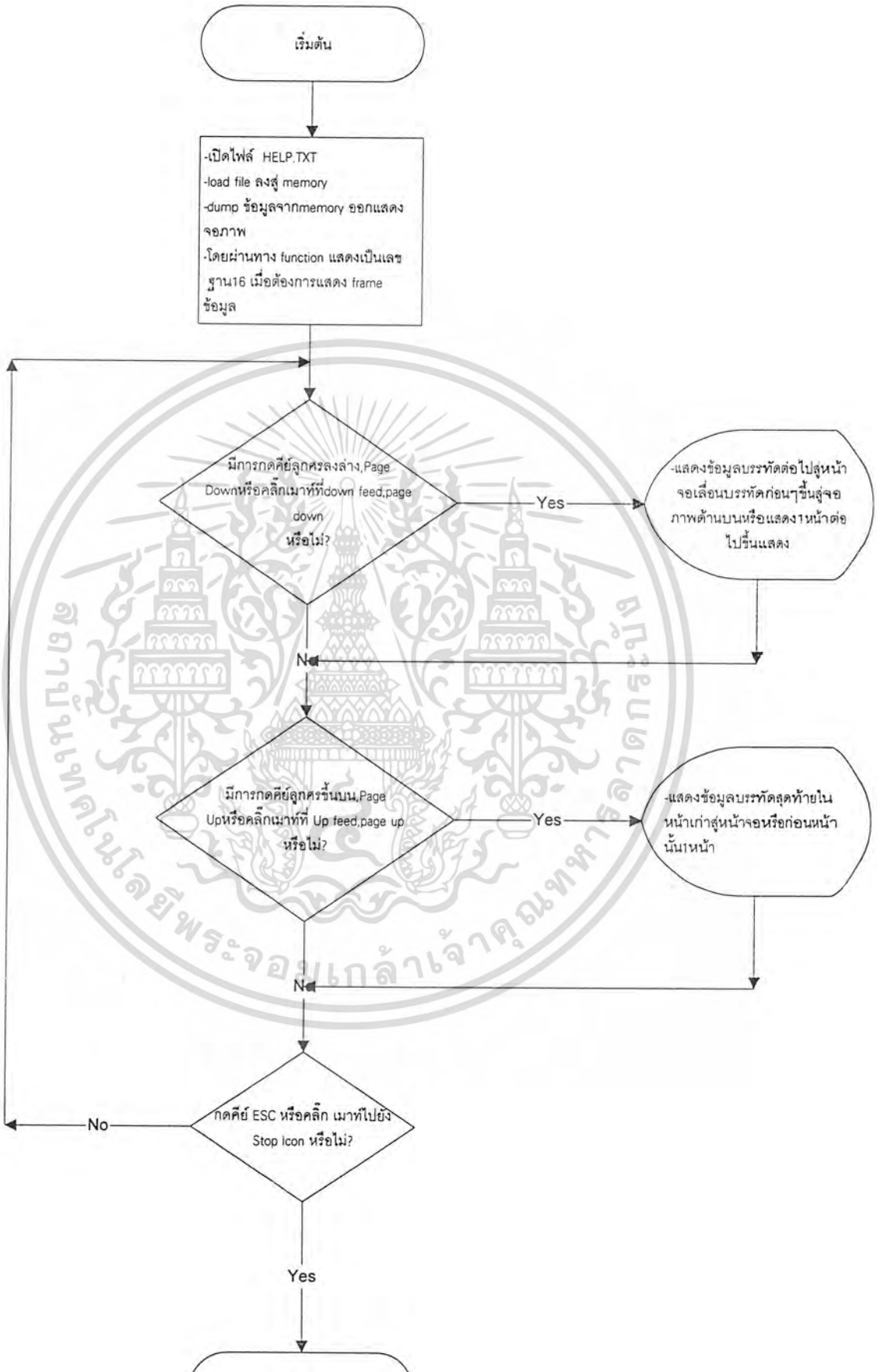
FILE MANAGEMANT



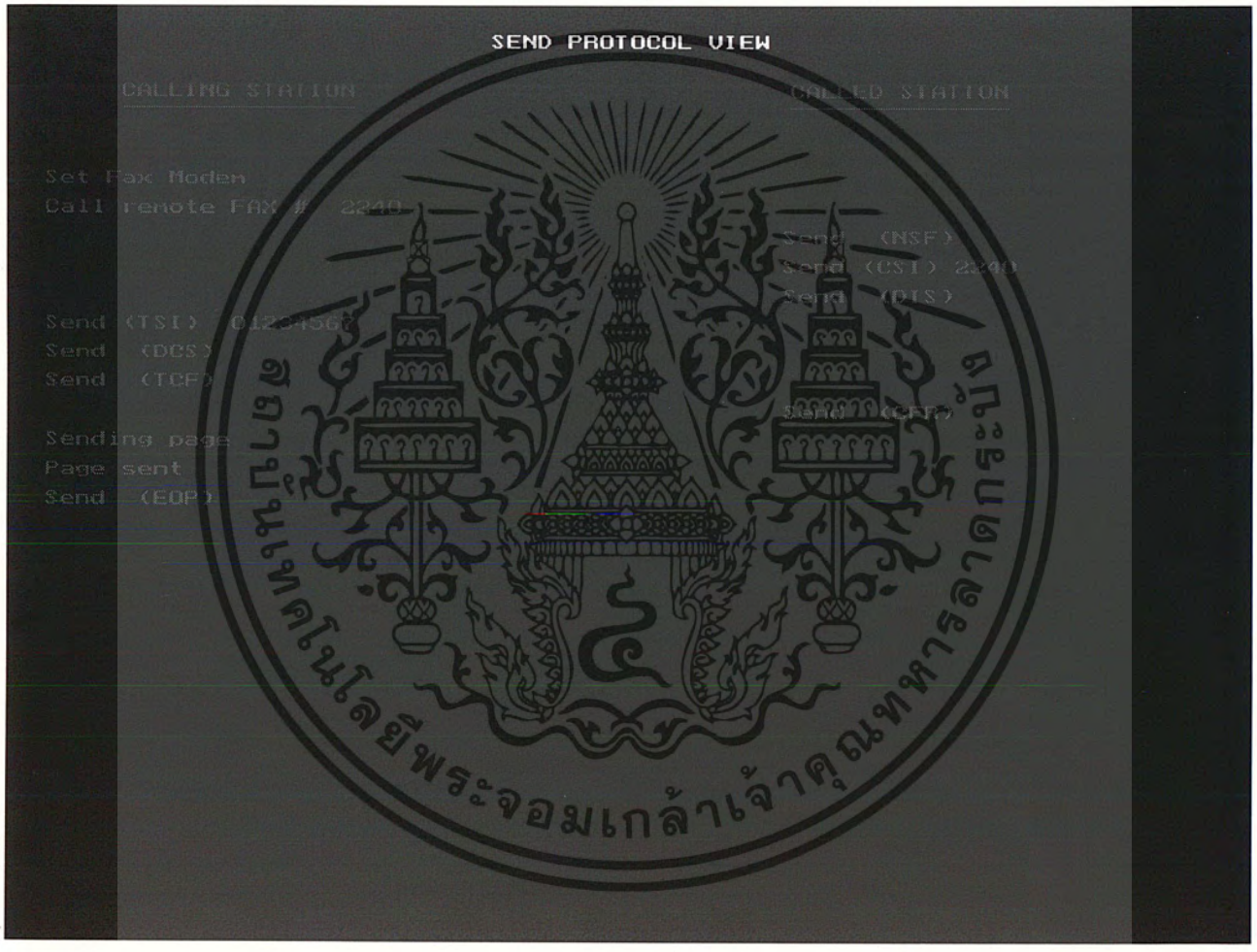
EXAMINE



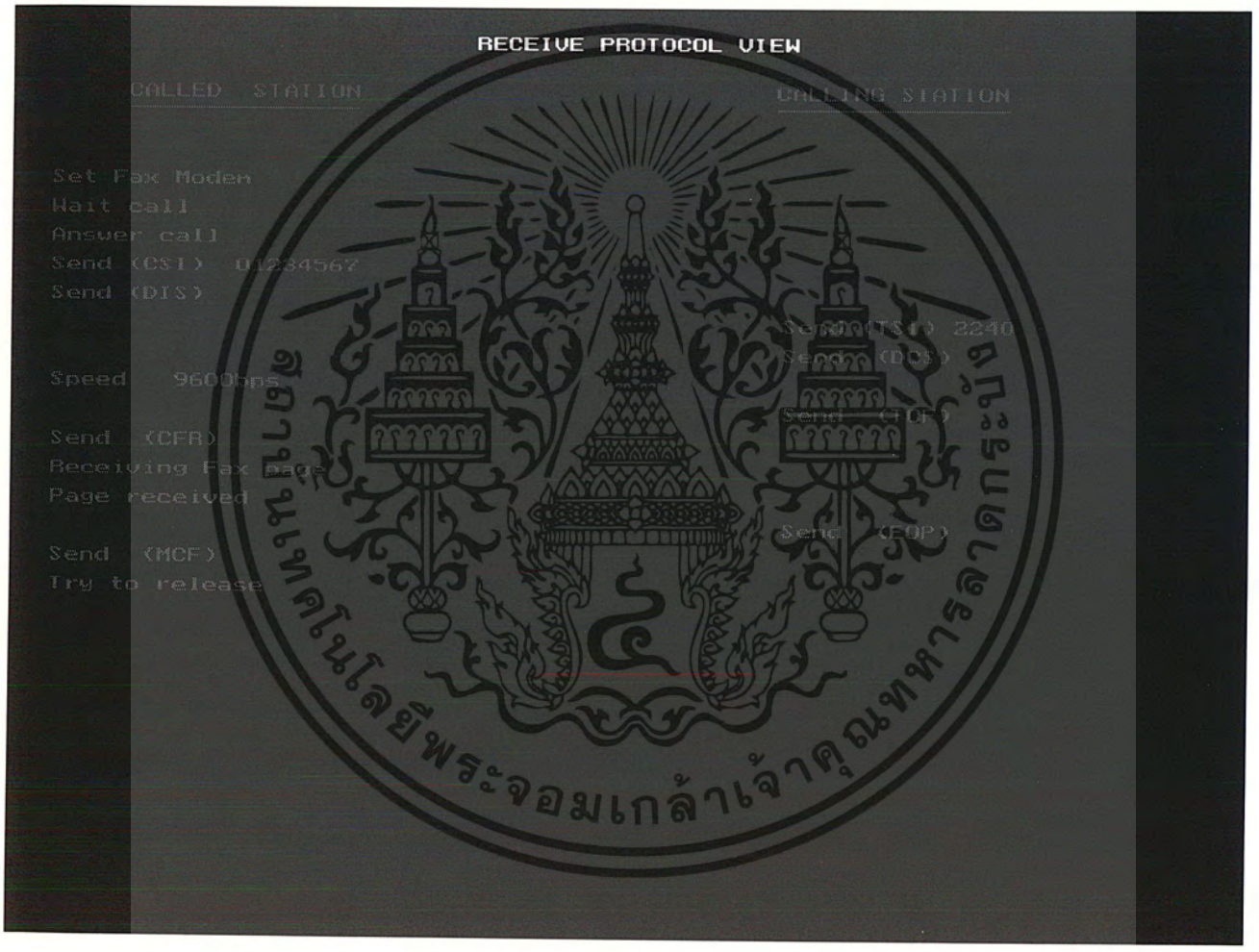
HELP



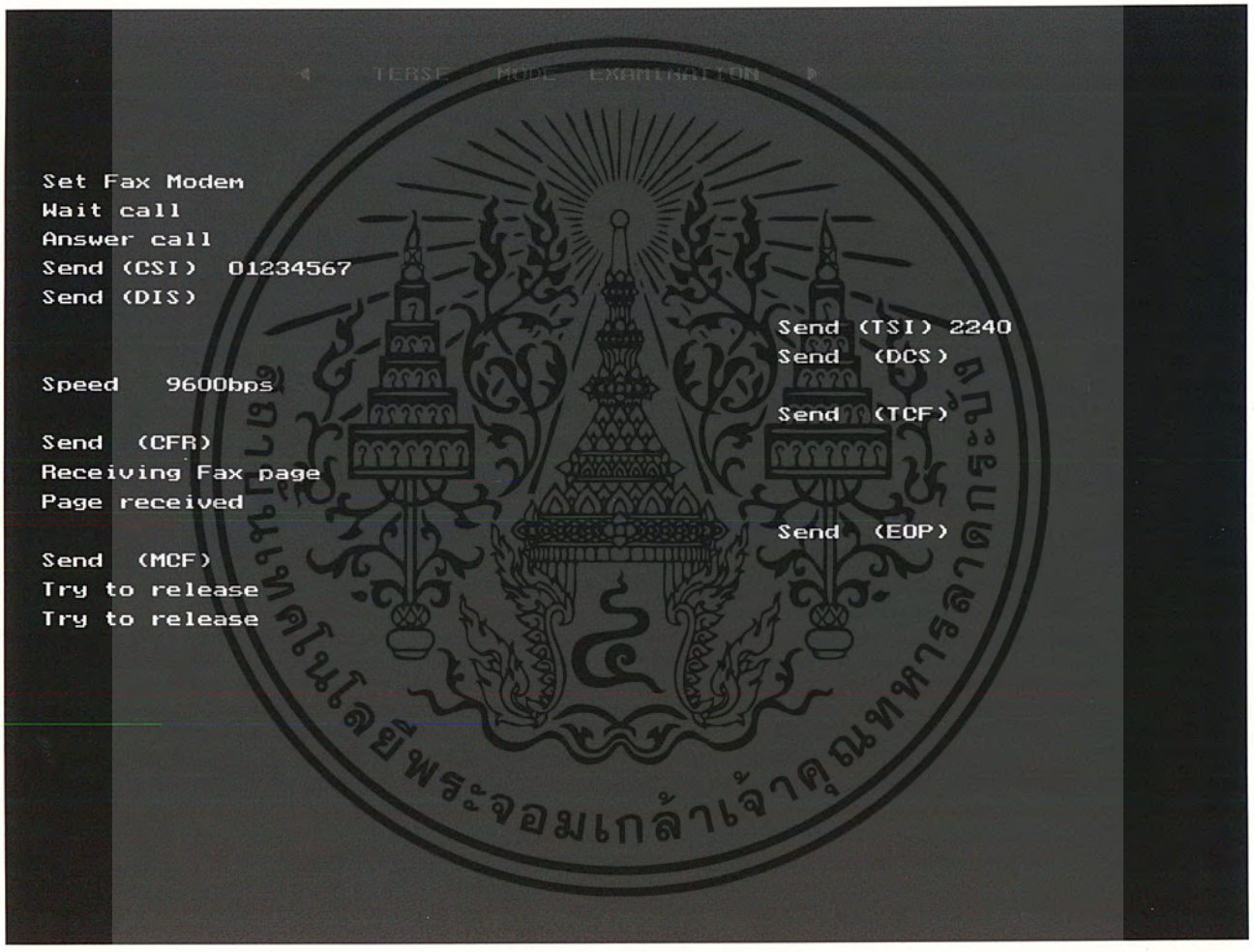
บทที่ 4
การทดลองและผลการทดลอง



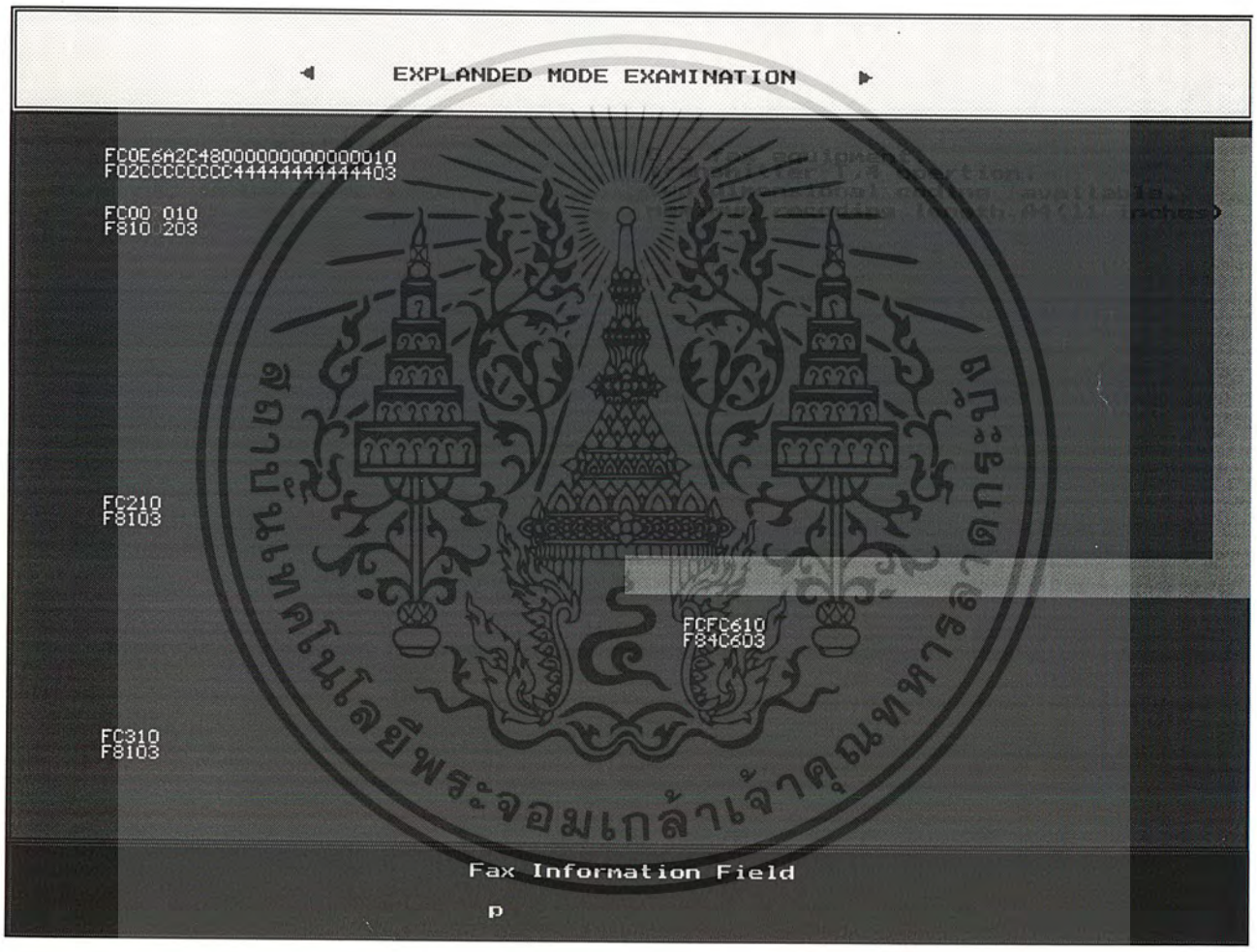
แสดง protocol การส่งแฟ็กซ์



แสดง protocol การรับแฟ็กซ์



แสดงการตรวจสอบในTERSE MODE



แสดงการตรวจสอบในEXPLANDED MODE

แสดงข้อมูลใน text file

```
                KMIT'L

   ...Mr Pairode Cheevaphruthinun...

   ...Mr Worapote   Chanpongsaeng...

   ...Mr Somrak       Petchartee ...

        Test Data Fax to View

 0  ABCDEFGHIJKLMNOPQRSTUVWXYZ

 1  abcdefghijklmnopqrstuvwxyz

 2  123456789 123456789 123456789

 3  123456789 123456789 123456789

 4  123456789 123456789 123456789

 5  ~`!@#$%^&*()_-+=|\{}[]"':;

 6  ~`!@#$%^&*()_-+=|\{}[]"':;

 7  <>,.?!@#$%^&*()_=<>{}[]/@&

 8  <>,.?!@#$%^&*()_=<>{}[]/@&

 9  ABCDEFGHIJKLMNOPQRSTUVWXYZ

10  ABCDEFGHIJKLMNOPQRSTUVWXYZ

11  abcdefghijklmnopqrstuvwxyz

12  123456789 123456789 123456789

13  123456789 123456789 123456789


C:\WINDOWS>
```

แสดงข้อมูลที่สามารถรับได้

```
                KMIT'L
   ...Mr Pairode Cheevaphruthinun...
   ...Mr Worapote   Chanpongsaeng...
   ...Mr Somrak       Petchartee ...
        Test Data Fax to View
 0  ABCDEFGHIJKLMNOPQRSTUVWXYZ
 1  abcdefghijklmnopqrstuvwxyz
 2  123456789 123456789 123456789
 3  123456789 123456789 123456789
 4  123456789 123456789 123456789
 5  ~`!@#$%^&*()_-+=|\{}[]"':;
 6  ~`!@#$%^&*()_-+=|\{}[]"':;
 7  <>,.?!@#$%^&*()_=<>{}[]/@&
 8  <>,.?!@#$%^&*()_=<>{}[]/@&
 9  ABCDEFGHIJKLMNOPQRSTUVWXYZ
10  ABCDEFGHIJKLMNOPQRSTUVWXYZ
11  abcdefghijklmnopqrstuvwxyz
12  123456789 123456789 123456789
13  123456789 123456789 123456789
14  123456789 123456789 123456789
15  ~`!@#$%^&*()_-+=|\{}[]"':;
16  ~`!@#$%^&*()_-+=|\{}[]"':;
17  <>,.?!@#$%^&*()_=<>{}[]/@&
18  <>,.?!@#$%^&*()_=<>{}[]/@&
19  ABCDEFGHIJKLMNOPQRSTUVWXYZ
20  ABCDEFGHIJKLMNOPQRSTUVWXYZ
21  abcdefghijklmnopqrstuvwxyz
22  123456789 123456789 123456789
23  123456789 123456789 123456789
24  123456789 123456789 123456789
25  ~`!@#$%^&*()_-+=|\{}[]"':;
26  ~`!@#$%^&*()_-+=|\{}[]"':;
27  <>,.?!@#$%^&*()_=<>{}[]/@&
28  <>,.?!@#$%^&*()_=<>{}[]/@&
29  ABCDEFGHIJKLMNOPQRSTUVWXYZ
        Test Data Fax to View
30  ABCDEFGHIJKLMNOPQRSTUVWXYZ
31  abcdefghijklmnopqrstuvwxyz
32  123456789 123456789 123456789
33  123456789 123456789 123456789
34  123456789 123456789 123456789
35  ~`!@#$%^&*()_-+=|\{}[]"':;
 Page 1  Zoom 2
```

บทที่ 5
บทวิจารณ์และบทสรุป

เครื่องแฟกซ์ที่มีใช้อยู่ในปัจจุบันจะเป็นกรุ๊ปสามเกือบทั้งหมดเนื่องจากเหตุผลการเข้ากันได้จากเครื่องแฟกซ์ต่างยี่ห้อกันเพราะเป็นระบบการรับส่งแบบดิจิตอลความสามารถของเครื่องแฟกซ์ในกรุ๊ปนี้อีกสิ่งหนึ่งคือสามารถที่จะทำการเพิ่มพารามิเตอร์เพื่อของใช้ระบบการรับส่งแบบพิเศษอื่น ๆ เช่น การบีบอัดข้อมูล,การแก้ไขข้อมูลเมื่อเกิดการผิดพลาดขณะทำการส่ง,การเข้ารหัสข้อมูลเพื่อความลับ และดูเหมือนจะมีการเพิ่มส่วนนี้ออกไปอีกเรื่อย ๆ นอกจากนี้เครื่องแฟกซ์บางยี่ห้อจะมี โปรโตรคอลเป็นของตัวเองและใช้เมื่อทำการรับส่งกับเครื่องแฟกซ์ที่เป็นยี่ห้อเดียวกัน ส่วนแฟกซ์โมเด็มเป็นประดิษฐกรรมที่รวมเอาเครื่องแฟกซ์และโมเด็มเข้าด้วยกันการควบคุมการทำงานจะอาศัยการควบคุมทางซอฟ์ทแวร์ซึ่งประเภทของซอฟ์ทแวร์ที่จะมาทำการควบคุมขึ้นอยู่กับการเลือกใช้ class ของตัวแฟกซ์โมเด็ม คลาสตามความหมายของแฟกซ์โมเด็มคือคลาสที่ต่างกันคือระดับในการที่จะไปควบคุมการทำงานของตัวแฟกซ์โมเด็มได้มากน้อยแตกต่างกันดังนี้

class 1 :แฟกซ์โมเด็มจะทำงานเฉพาะส่วน physical and data link layer(ขบวนการในการติดต่อถึงกันและทำการถอดแฟรม HDLC ออกมา) ส่วนขั้นตอนหรือลำดับในการรับส่งข้อมูลตลอดจนการเข้ารหัสและการประมวลผลข้อมูล จะเป็นหน้าที่ของซอฟท์แวร์ที่จะทำการควบคุมตัวแฟกซ์โมเดมเอาเอง

class 2:แฟกซ์โมเดมจะทำงานในส่วนทำการติดต่อถึงกัน,ถอดแฟรม HDLC และขั้นตอนทั้งหมดในการรับส่งข้อมูลส่วนของการเข้ารหัสและการประมวลผลข้อมูลเป็นหน้าที่ของซอฟท์แวร์

class 3:แฟกซ์โมเดมจะทำงานในส่วนทำการติดต่อถึงกัน,การถอดแฟรม HDLC,ขั้นตอนในการรับส่งข้อมูลและการเข้ารหัสและถอดรหัสข้อมูลส่วนการประมวลผลข้อมูลจะเป็นหน้าที่ของซอฟท์แวร์

โปรเจ็คชิ้นนี้จะในงานใน class 1 อย่างเดียว ทั้งนี้เพื่อวัตถุประสงค์ในการตรวจสอบข้อมูลในแฟรม HDLC และแสดงออกทางจอ คอมพิวเตอร์ว่าทั้งสองด้านที่ทำการรับส่งข้อมูลกันมีการส่งแฟรมอะไรก่อนหลัง

ภาคผนวก

โปรแกรม SOURCE CODE

```
//********************** ASCIIMAP.CPP ****************************//

UINT    ascii_map[256][11] =
  {
  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 0 -
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x3FFC, 0xC003, 0x0033, 0xC003,      // 1 -
    0xC003, 0xCFF3, 0xC3C3, 0xC003,
    0x3FFC, 0x0000 },

  { 0x0000, 0x3FFC, 0xFFFF, 0xF3CF, 0xFFFF,      // 2 -
    0xFFFF, 0xF00F, 0xFC3F, 0xFFFF,
    0x3FFC, 0x0000 },

  { 0x0000, 0x0000, 0x3CF0, 0xFFFC, 0xFFFC,      // 3 -
    0xFFFC, 0x3FF0, 0x0FC0, 0x0300,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0300, 0x0FC0, 0x3FF0,      // 4 -
    0xFFFC, 0x3FF0, 0x0FC0, 0x0300,
    0x0000, 0x0000 },

  { 0x0000, 0x0FC0, 0x3FF0, 0x0FC0, 0xFFFC,      // 5 -
    0xFFFC, 0x3FF0, 0x0300, 0x3FF0,
    0x0000, 0x0000 },

  { 0x0000, 0x0300, 0x0300, 0x0FC0, 0x3FF0,      // 6 -
    0xFFFC, 0x3FF0, 0x0FC0, 0x3FF0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x03C0, 0x0FF0,      // 7 -
    0x0FF0, 0x03C0, 0x03C0, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 8 -
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 9 -
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 10 -
    0x0000, 0x0000, 0x0000, 0x0000,
```

```
    0x0000, 0x0000 },

  { 0x0000, 0x00FF, 0x003F, 0x00FF, 0x3FF3,      // 11 -
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 12 -
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 13 -
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x3FFF, 0x3C0F, 0x3FFF, 0x3C0F,      // 14 -
    0x3C0F, 0x3C3F, 0xFC3C, 0xF000,
    0x0000, 0x0000 },

  { 0x0000, 0xC3C3, 0x3300, 0x0FF0, 0xFC3F,      // 15 -
    0xFC3F, 0x0FF0, 0x3300, 0xC3C3,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xC000, 0xFC00, 0xFFC0,      // 16 - □
    0xFFFC, 0xFFC0, 0xFC00, 0xC000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x000C, 0x00FC, 0x0FFC,      // 17 - □
    0xFFFC, 0x0FFC, 0x00FC, 0x000C,
    0x0000, 0x0000 },

  { 0x0000, 0x03C0, 0x0FF0, 0x3FFC, 0x03C0,      // 18 - □
    0x03C0, 0x3FFC, 0x0FF0, 0x03C0,
    0x0000, 0x0000 },

  { 0x0000, 0x3C3C, 0x3C3C, 0x3C3C, 0x3C3C,      // 19 - □
    0x3C3C, 0x0000, 0x0000, 0x3C3C,
    0x0000, 0x0000 },

  { 0x0000, 0x3FFF, 0xF3CF, 0xF3CF, 0x3FCF,      // 20 - □
    0x03CF, 0x03CF, 0x03CF, 0x03CF,
    0x0000, 0x0000 },

  { 0x0000, 0x0FFC, 0x3C0F, 0x0FC0, 0x3CF0,      // 21 - □
    0x3CF0, 0x0FC0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 22 - □
    0x3FFC, 0x3FFC, 0x3FFC, 0x3FFC,
    0x0000, 0x0000 },

  { 0x0000, 0x03C0, 0x0FF0, 0x3FFC, 0x03C0,      // 23 - □
```

```
  0x3FFC, 0x0FF0, 0x03C0, 0xFFFF,
  0x0000, 0x0000 },

{ 0x0000, 0x03C0, 0x0FF0, 0x3FFC, 0x03C0,     // 24 - □
  0x03C0, 0x03C0, 0x03C0, 0x03C0,
  0x0000, 0x0000 },

{ 0x0000, 0x03C0, 0x03C0, 0x03C0, 0x03C0,     // 25 - □
  0x03C0, 0x3FFC, 0x0FF0, 0x03C0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,     // 26
  0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0F00, 0x3C00,     // 27 -
  0xFFFC, 0x3C00, 0x0F00, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0xF000, 0xF000,     // 28 - □
  0xF000, 0xFFFC, 0xFFFC, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0C30, 0x3C3C, 0xFFFF,     // 29 - □
  0x3C3C, 0x0C30, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x03C0, 0x0FF0, 0x3FFC,     // 30 - ↑
  0xFFFF, 0xFFFF, 0xFFFF, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF,     // 31 - ↓
  0x3FFC, 0x0FF0, 0x03C0, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,     // 32 -
  0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0F00, 0x3FC0, 0x3FC0, 0x0F00,     // 33 - !
  0x0F00, 0x0F00, 0x0000, 0x0F00,
  0x0000, 0x0000 },

{ 0x0000, 0x3CF0, 0x3CF0, 0x3CF0, 0x0000,     // 34 - "
  0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x3CF0, 0x3CF0, 0xFFFC, 0x3CF0,     // 35 - #
  0xFFFC, 0x3CF0, 0x3CF0, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0F00, 0x3FF0, 0xF000, 0x3FC0,     // 36 - $
```

```
    0x00F0, 0x00F0, 0xFFC0, 0x0F00,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xF03C, 0xF0F0, 0x03C0,      // 37 - %
    0x0F00, 0x3C3C, 0xF03C, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0FC0, 0x3CF0, 0x0FC0, 0x3F3C,      // 38 - &
    0xF3F0, 0xF0F0, 0xF0F0, 0x3F3C,
    0x0000, 0x0000 },

  { 0x0000, 0x3C00, 0x3C00, 0xF000, 0x0000,      // 39 - '
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x03C0, 0x0F00, 0x3C00, 0x3C00,      // 40 - (
    0x3C00, 0x3C00, 0x0F00, 0x03C0,
    0x0000, 0x0000 },

  { 0x0000, 0x3C00, 0x0F00, 0x03C0, 0x03C0,      // 41 - )
    0x03C0, 0x03C0, 0x0F00, 0x3C00,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x3C3C, 0x0FF0, 0xFFFF,      // 42 - *
    0x0FF0, 0x3C3C, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0F00, 0x0F00,      // 43 - +
    0xFFF0, 0x0F00, 0x0F00, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 44 - ,
    0x0000, 0x0F00, 0x0F00, 0x0F00,
    0x3C00, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0xFFF0,      // 45 - -
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 46 - .
    0x0000, 0x0000, 0x0F00, 0x0F00,
    0x0000, 0x0000 },

  { 0x0000, 0x000F, 0x003C, 0x00F0, 0x03C0,      // 47 - /
    0x0F00, 0x3C00, 0xF000, 0xC000,
    0x0000, 0x0000 },

  { 0x0000, 0x3FF0, 0xF03C, 0xF0FC, 0xF3FC,      // 48 - 0
    0xFF3C, 0xFC3C, 0xF03C, 0x3FF0,
    0x0000, 0x0000 },

  { 0x0000, 0x0F00, 0x3F00, 0x0F00, 0x0F00,      // 49 - 1
```

```
  0x0F00, 0x0F00, 0x0F00, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0x00F0, 0x0FC0,     // 50 - 2
  0x3C00, 0xF000, 0xF030, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0x00F0, 0x0FC0,     // 51 - 3
  0x00F0, 0x00F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x03F0, 0x0FF0, 0x3CF0, 0xF0F0,     // 52 - 4
  0xFFFC, 0x00F0, 0x00F0, 0x03FC,
  0x0000, 0x0000 },

{ 0x0000, 0xFFF0, 0xF000, 0xF000, 0xFFC0,     // 53 - 5
  0x00F0, 0x00F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x0FC0, 0x3C00, 0xF000, 0xFFC0,     // 54 - 6
  0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0xFFF0, 0xF0F0, 0x00F0, 0x03C0,     // 55 - 7
  0x0F00, 0x0F00, 0x0F00, 0x0F00,
  0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0xF0F0, 0x3FC0,     // 56 - 8
  0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0xF0F0, 0xF0F0,     // 57 - 9
  0x3FF0, 0x00F0, 0x03C0, 0x3F00,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0F00, 0x0F00, 0x0000,     // 58 - :
  0x0000, 0x0F00, 0x0F00, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0F00, 0x0F00, 0x0000,     // 59 - ;
  0x0000, 0x0F00, 0x0F00, 0x0F00,
  0x3C00, 0x0000 },

{ 0x0000, 0x0000, 0x03C0, 0x0F00, 0x3C00,     // 60 - <
  0xF000, 0x3C00, 0x0F00, 0x03C0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0xFFF0, 0x0000,     // 61 - =
  0x0000, 0xFFF0, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3C00, 0x0F00, 0x03C0,     // 62 - >
```

```
0x00F0, 0x03C0, 0x0F00, 0x3C00,
0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0x00F0, 0x03C0,      // 63 - ?
0x0F00, 0x0F00, 0x0000, 0x0F00,
0x0000, 0x0000 },

{ 0x0000, 0x3FF0, 0xF03C, 0xF3FC, 0xF3FC,      // 64 - @
0xF3FC, 0xF3F0, 0xF000, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0F00, 0x3FC0, 0xF0F0, 0xF0F0,      // 65 - A
0xFFF0, 0xF0F0, 0xF0F0, 0xF0F0,
0x0000, 0x0000 },

{ 0x0000, 0xFFF0, 0x3C3C, 0x3C3C, 0x3FF0,      // 66 - B
0x3C3C, 0x3C3C, 0x3C3C, 0xFFF0,
0x0000, 0x0000 },

{ 0x0000, 0x0FF0, 0x3C3C, 0xF000, 0xF000,      // 67 - C
0xF000, 0xF000, 0x3C3C, 0x0FF0,
0x0000, 0x0000 },

{ 0x0000, 0xFFC0, 0x3CF0, 0x3C3C, 0x3C3C,      // 68 - D
0x3C3C, 0x3C3C, 0x3CF0, 0xFFC0,
0x0000, 0x0000 },

{ 0x0000, 0xFFFC, 0x3C0C, 0x3CC0, 0x3FC0,      // 69 - E
0x3CC0, 0x3C00, 0x3C0C, 0xFFFC,
0x0000, 0x0000 },

{ 0x0000, 0xFFFC, 0x3C0C, 0x3CC0, 0x3FC0,      // 70 - F
0x3CC0, 0x3C00, 0x3C00, 0xFF00,
0x0000, 0x0000 },

{ 0x0000, 0x0FF0, 0x3C3C, 0xF000, 0xF000,      // 71 - G
0xF0FC, 0xF03C, 0x3C3C, 0x0FFC,
0x0000, 0x0000 },

{ 0x0000, 0xF0F0, 0xF0F0, 0xF0F0, 0xFFF0,      // 72 - H
0xF0F0, 0xF0F0, 0xF0F0, 0xF0F0,
0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0x0F00, 0x0F00, 0x0F00,      // 73 - I
0x0F00, 0x0F00, 0x0F00, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x03FC, 0x00F0, 0x00F0, 0x00F0,      // 74 - J
0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0xFC3C, 0x3C3C, 0x3CF0, 0x3FC0,      // 75 - K
```

```
    0x3CF0, 0x3C3C, 0x3C3C, 0xFC3C,
    0x0000, 0x0000 },

  { 0x0000, 0xFF00, 0x3C00, 0x3C00, 0x3C00,      // 76 - L
    0x3C00, 0x3C0C, 0x3C3C, 0xFFFC,
    0x0000, 0x0000 },

  { 0x0000, 0xF03C, 0xFCFC, 0xFFFC, 0xFFFC,      // 77 - M
    0xF33C, 0xF03C, 0xF03C, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0xF03C, 0xFC3C, 0xFF3C, 0xF3FC,      // 78 - N
    0xF0FC, 0xF03C, 0xF03C, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0x0FC0, 0x3CF0, 0xF03C, 0xF03C,      // 79 - O
    0xF03C, 0xF03C, 0x3CF0, 0x0FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xFFF0, 0x3C3C, 0x3C3C, 0x3FF0,      // 80 - P
    0x3C00, 0x3C00, 0x3C00, 0xFF00,
    0x0000, 0x0000 },

  { 0x0000, 0x3FC0, 0xF0F0, 0xF0F0, 0xF0F0,      // 81 - Q
    0xF0F0, 0xFCF0, 0xF3F0, 0x3FC0,
    0x03F0, 0x0000 },

  { 0x0000, 0xFFF0, 0x3C3C, 0x3C3C, 0x3FF0,      // 82 - R
    0x3CF0, 0x3C3C, 0x3C3C, 0xFC3C,
    0x0000, 0x0000 },

  { 0x0000, 0x3FC0, 0xF0F0, 0xF000, 0x3FC0,      // 83 - S
    0x00F0, 0x00F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xFFF0, 0xCF30, 0x0F00, 0x0F00,      // 84 - T
    0x0F00, 0x0F00, 0x0F00, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xF0F0, 0xF0F0, 0xF0F0, 0xF0F0,      // 85 - U
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xF0F0, 0xF0F0, 0xF0F0, 0xF0F0,      // 86 - V
    0xF0F0, 0xF0F0, 0x3FC0, 0x0F00,
    0x0000, 0x0000 },

  { 0x0000, 0xF03C, 0xF03C, 0xF03C, 0xF33C,      // 87 - W
    0xFFFC, 0xFCFC, 0xF03C, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0xF03C, 0xF03C, 0x3CF0, 0x0FC0,      // 88 - X
```

```
    0x0FC0, 0x3CF0, 0xF03C, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,     // 89 - Y
    0x0F00, 0x0F00, 0x0F00, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xFFFC, 0xF03C, 0xC0F0, 0x03C0,     // 90 - Z
    0x0F00, 0x3C0C, 0xF03C, 0xFFFC,
    0x0000, 0x0000 },

  { 0x0000, 0x3FC0, 0x3C00, 0x3C00, 0x3C00,     // 91 - [
    0x3C00, 0x3C00, 0x3C00, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xF000, 0x3C00, 0x0F00, 0x03C0,     // 92 - \
    0x00F0, 0x003C, 0x000F, 0x0003,
    0x0000, 0x0000 },

  { 0x0000, 0x3FC0, 0x03C0, 0x03C0, 0x03C0,     // 93 - ]
    0x03C0, 0x03C0, 0x03C0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0300, 0x0FC0, 0x3CF0, 0xF03C,     // 94 - ^
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,     // 95 - _
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFFFF, 0x0000 },

  { 0x0000, 0x0F00, 0x0F00, 0x03C0, 0x0000,     // 96 - `
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3FC0, 0x00F0,     // 97 - a
    0x3FF0, 0xF0F0, 0xF0F0, 0x3F3C,
    0x0000, 0x0000 },

  { 0x0000, 0xFC00, 0x3C00, 0x3FF0, 0x3C3C,     // 98 - b
    0x3C3C, 0x3C3C, 0x3C3C, 0xF3F0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3FC0, 0xF0F0,     // 99 - c
    0xF000, 0xF000, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x03F0, 0x00F0, 0x3FF0, 0xF0F0,     // 100 - d
    0xF0F0, 0xF0F0, 0xF0F0, 0x3F3C,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3FC0, 0xF0F0,     // 101 - e
```

```
    0xFFF0, 0xF000, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0FC0, 0x3CF0, 0x3C00, 0xFF00,      // 102 - f
    0x3C00, 0x3C00, 0x3C00, 0xFF00,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3F3C, 0xF0F0,      // 103 - g
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FF0,
    0x00F0, 0xFFC0 },

  { 0x0000, 0xFC00, 0x3C00, 0x3CF0, 0x3F3C,      // 104 - h
    0x3C3C, 0x3C3C, 0x3C3C, 0xFC3C,
    0x0000, 0x0000 },

  { 0x0000, 0x0F00, 0x0000, 0x3F00, 0x0F00,      // 105 - i
    0x0F00, 0x0F00, 0x0F00, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x00F0, 0x0000, 0x0000, 0x00F0,      // 106 - j
    0x00F0, 0x00F0, 0x00F0, 0xF0F0,
    0xF0F0, 0x3FC0 },

  { 0x0000, 0xFC00, 0x3C00, 0x3C3C, 0x3CF0,      // 107 - k
    0x3FC0, 0x3CF0, 0x3C3C, 0xFC3C,
    0x0000, 0x0000 },

  { 0x0000, 0x3F00, 0x0F00, 0x0F00, 0x0F00,      // 108 - l
    0x0F00, 0x0F00, 0x0F00, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF0F0, 0xFFFC,      // 109 - m
    0xFFFC, 0xF33C, 0xF33C, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xFFC0, 0xF0F0,      // 110 - n
    0xF0F0, 0xF0F0, 0xF0F0, 0xF0F0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3FC0, 0xF0F0,      // 111 - o
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF3F0, 0x3C3C,      // 112 - p
    0x3C3C, 0x3C3C, 0x3C3C, 0x3FF0,
    0x3C00, 0xFF00 },

  { 0x0000, 0x0000, 0x0000, 0x3F3C, 0xF0F0,      // 113 - q
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FF0,
    0x00F0, 0x03FC },

  { 0x0000, 0x0000, 0x0000, 0xF3F0, 0x3F3C,      // 114 - r
```

```
    0x3C3C, 0x3C00, 0x3C00, 0xFF00,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3FC0, 0xF030,    // 115 - s
    0x3F00, 0x03F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0300, 0x0F00, 0x3FF0, 0x0F00,    // 116 - t
    0x0F00, 0x0F00, 0x0F30, 0x03C0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF0F0, 0xF0F0,    // 117 - u
    0xF0F0, 0xF0F0, 0xF0F0, 0x3F3C,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF0F0, 0xF0F0,    // 118 - v
    0xF0F0, 0xF0F0, 0x3FC0, 0x0F00,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF03C, 0xF03C,    // 119 - w
    0xF33C, 0xFFFC, 0xFFFC, 0x3CF0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF03C, 0x3CF0,    // 120 - x
    0x0FC0, 0x0FC0, 0x3CF0, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0xF0F0, 0xF0F0,    // 121 - y
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FF0,
    0x00F0, 0xFFC0 },

  { 0x0000, 0x0000, 0x0000, 0xFFF0, 0xC3C0,    // 122 - z
    0x0F00, 0x3C00, 0xF030, 0xFFF0,
    0x0000, 0x0000 },

  { 0x0000, 0x03F0, 0x0F00, 0x0F00, 0xFC00,    // 123 - {
    0x0F00, 0x0F00, 0x0F00, 0x03F0,
    0x0000, 0x0000 },

  { 0x0000, 0x03C0, 0x03C0, 0x03C0, 0x0000,    // 124 - |
    0x03C0, 0x03C0, 0x03C0, 0x03C0,
    0x0000, 0x0000 },

  { 0x0000, 0xFC00, 0x0F00, 0x0F00, 0x03F0,    // 125 - }
    0x0F00, 0x0F00, 0x0F00, 0xFC00,
    0x0000, 0x0000 },

  { 0x0000, 0x3F3C, 0xF3F0, 0x0000, 0x0000,    // 126 - ~
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0300, 0x0FC0, 0x3CF0,    // 127 ■
```

```
0xF03C, 0xF03C, 0xF03C, 0xFFFC,
0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0xF000, 0xF0F0,       // 128 - «
0x3FC0, 0x03C0, 0x00F0, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xF0F0, 0x0000, 0xF0F0,       // 129 - »
0xF0F0, 0xF0F0, 0xF0F0, 0x3FFC,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x03F0, 0x0000, 0x3FC0,       // 130 -
0xF0F0, 0xFFF0, 0xF000, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3FFC, 0xF00F, 0x0FF0,       // 131 -
0x003C, 0x0FFC, 0x3C3C, 0x0FFF,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xF0F0, 0x0000, 0x3FC0,       // 132 -
0x00F0, 0x3FF0, 0xF0F0, 0x3FFC,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFC00, 0x0000, 0x3FC0,       // 133 - ...
0x00F0, 0x3FF0, 0xF0F0, 0x3FFC,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0F00, 0x0F00, 0x3FC0,       // 134 -
0x00F0, 0x3FF0, 0xF0F0, 0x3FFC,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x3FC0, 0xF000,       // 135 -
0xF000, 0x3FC0, 0x00F0, 0x0FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3FFC, 0xF00F, 0x0FF0,       // 136 -
0x3C3C, 0x3FFC, 0x3C00, 0x0FF0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xF0F0, 0x0000, 0x3FC0,       // 137 -
0xF0F0, 0xFFF0, 0xF000, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFC00, 0x0000, 0x3FC0,       // 138 -
0xF0F0, 0xFFF0, 0xF000, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xF0F0, 0x0000, 0x3F00,       // 139 -
0x0F00, 0x0F00, 0x0F00, 0x3FC0,
0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3FF0, 0xF03C, 0x0FC0,       // 140 -
```

```
    0x03C0, 0x03C0, 0x03C0, 0x0FF0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xFC00, 0x0000, 0x3F00,      // 141 -
    0x0F00, 0x0F00, 0x0F00, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0xF03C, 0x0FC0, 0x3CF0, 0xF03C,      // 142 -
    0xFFFC, 0xF03C, 0xF03C, 0xF03C,
    0x0000, 0x0000 },

  { 0x0000, 0x0F00, 0x0F00, 0x0000, 0x3FC0,      // 143 -
    0xF0F0, 0xFFF0, 0xF0F0, 0xF0F0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x03F0, 0x0000, 0xFFF0,      // 144 - ณ
    0x3C00, 0x3FC0, 0x3C00, 0xFFF0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0x0000, 0x3FFF, 0x00F0,      // 145 -
    0x3FFF, 0xF0F0, 0xF0F0, 0x3FFF,
    0x0000, 0x0000 },

  { 0x0000, 0x0FFC, 0x3CF0, 0xF0F0, 0xFFFC,      // 146 -
    0xF0F0, 0xF0F0, 0xF0F0, 0xF0FC,
    0x0000, 0x0000 },

  { 0x0000, 0x3FC0, 0xF0F0, 0x0000, 0x3FC0,      // 147 -
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xF0F0, 0x0000, 0x3FC0,      // 148 -
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xFC00, 0x0000, 0x3FC0,      // 149 -
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
    0x0000, 0x0000 },

  { 0x0000, 0x3FC0, 0xF0F0, 0x0000, 0xF0F0,      // 150 -
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FFC,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xFC00, 0x0000, 0xF0F0,      // 151 -
    0xF0F0, 0xF0F0, 0xF0F0, 0x3FFC,
    0x0000, 0x0000 },

  { 0x0000, 0x0000, 0xF0F0, 0x0000, 0xF0F0,      // 152 -
    0xF0F0, 0xF0F0, 0x3FF0, 0x00F0,
    0xFFC0, 0x0000 },
```

```
{ 0x0000, 0xF00F, 0x03C0, 0x0FF0, 0x3C3C,      // 153 -
  0x3C3C, 0x0FF0, 0x03C0, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0xF0F0, 0x0000, 0xF0F0, 0xF0F0,      // 154 -
  0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x03C0, 0x03C0, 0x3FFC, 0xF000,      // 155 -
  0xF000, 0x3FFC, 0x03C0, 0x03C0,
  0x0000, 0x0000 },

{ 0x0000, 0x0FC0, 0x3CF0, 0x3C30, 0xFF00,      // 156 -
  0x3C00, 0xFC3C, 0xFC3C, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0xF0F0, 0xF0F0, 0x3FC0, 0xFFF0,      // 157 -
  0x0F00, 0xFFF0, 0x0F00, 0x0F00,
  0x0000, 0x0000 },

{ 0x0000, 0xFFC0, 0xF0F0, 0xF0F0, 0xF0F0,      // 158 -
  0xFF00, 0xF03C, 0xF0FF, 0xF03C,
  0xF03F, 0x0000 },

{ 0x0000, 0x00FC, 0x03CF, 0x03C0, 0x0FF0,      // 159 -
  0x03C0, 0x03C0, 0x03C0, 0xF3C0,
  0x3F00, 0x0000 },

{ 0x0000, 0x03F0, 0x0000, 0x3FC0, 0x00F0,      // 160 -
  0x3FF0, 0xF0F0, 0xF0F0, 0x3FFC,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0FC0, 0x0000, 0x3F00,      // 161 - ก
  0x0F00, 0x0F00, 0x0F00, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x03F0, 0x0000, 0x3FC0,      // 162 - ข
  0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x03F0, 0x0000, 0xF0F0,      // 163 - ฃ
  0xF0F0, 0xF0F0, 0xF0F0, 0x3FFC,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFFC0, 0x0000, 0xFFC0,      // 164 - ค
  0xF0F0, 0xF0F0, 0xF0F0, 0xF0F0,
  0x0000, 0x0000 },

{ 0x0000, 0xFFF0, 0x0000, 0xF0F0, 0xFCF0,      // 165 - ฅ
  0xFFF0, 0xF3F0, 0xF3F0, 0xF0F0,
  0x0000, 0x0000 },
```

```
{ 0x0000, 0x0000, 0x0FF0, 0x3CF0, 0x3CF0,      // 166 - ฆ
  0x0FFC, 0x0000, 0x3FFC, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0FC0, 0x3CF0, 0x3CF0,      // 167 - ง
  0x0FC0, 0x0000, 0x3FF0, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0F00, 0x0000, 0x0F00, 0x0F00,      // 168 - จ
  0x3C00, 0xF000, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFF0,      // 169 - ฉ
  0xF000, 0xF000, 0xF000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFF0,      // 170 - ช
  0x00F0, 0x00F0, 0x00F0, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0xF00F, 0xF03C, 0xF0F0, 0xF3FC,      // 171 - ซ
  0x0F0F, 0x3C3C, 0xF0F0, 0x00FF,
  0x0000, 0x0000 },

{ 0x0000, 0xF00F, 0xF03C, 0xF0F0, 0xF3CF,      // 172 - ฌ
  0x0F3F, 0x3CFF, 0xF0FF, 0x000F,
  0x0000, 0x0000 },

{ 0x0000, 0x03C0, 0x03C0, 0x0000, 0x03C0,      // 173 - ญ
  0x03C0, 0x03C0, 0x03C0, 0x03C0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0F0F, 0x3C3C, 0xF0F0,      // 174 - ฎ
  0x3C3C, 0x0F0F, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xF0F0, 0x3C3C, 0x0F0F,      // 175 - ฏ
  0x3C3C, 0xF0F0, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0xC0C0, 0x0C0C, 0xC0C0, 0x0C0C, 0xC0C0,      // 176 - ฐ
  0x0C0C, 0xC0C0, 0xC0C0, 0x0C0C,
  0xC0C0, 0x0C0C },

{ 0x0000, 0x3333, 0x0000, 0x3333, 0x0000,      // 177 - ฑ
  0x3333, 0x0000, 0x0000, 0x3333,
  0x0000, 0x3333 },

{ 0xFCFC, 0xF3CF, 0x3F3F, 0xF3CF, 0xFCFC,      // 178 - ฒ
  0xF3CF, 0x3F3F, 0x3F3F, 0xF3CF,
  0xFCFC, 0xF3CF },
```

```
{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,      // 179 - ณ
  0x03C0, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,      // 180 - ด
  0xFFC0, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0xFFC0,      // 181 - ต
  0x03C0, 0xFFC0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,      // 182 - ถ
  0xFF3C, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 183 - ท
  0xFFFC, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFC0,      // 184 - ธ
  0x03C0, 0xFFC0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0xFF3C,      // 185 - น
  0x003C, 0xFF3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,      // 186 - บ
  0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFC,      // 187 - ป
  0x003C, 0xFF3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0xFF3C,      // 188 - ผ
  0x003C, 0xFFFC, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,      // 189 - ฝ
  0xFFFC, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0xFFC0,      // 190 - พ
  0x03C0, 0xFFC0, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 191 - ฟ
  0xFFC0, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },
```

```
{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,      // 192 - ภ
  0x03FF, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,      // 193 - ม
  0xFFFF, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 194 - ย
  0xFFFF, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,      // 195 - ร
  0x03FF, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 196 - ฤ
  0xFFFF, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,      // 197 - ล
  0xFFFF, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03FF,      // 198 - ฦ
  0x03C0, 0x03FF, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,      // 199 - ว
  0x0F3F, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3F,      // 200 - ศ
  0x0F00, 0x0FFF, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0FFF,      // 201 - ษ
  0x0F00, 0x0F3F, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0xFF3F,      // 202 - ส
  0x0000, 0xFFFF, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF,      // 203 - ห
  0x0000, 0xFF3F, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3F,      // 204 - ฬ
  0x0F00, 0x0F3F, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },
```

```
{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF,     // 205 - อ
  0x0000, 0xFFFF, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0xFF3F,     // 206 - ฮ
  0x0000, 0xFF3F, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0xFFFF,     // 207 - ฯ
  0x0000, 0xFFFF, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,     // 208 - ะ
  0xFFFF, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF,     // 209 - ั
  0x0000, 0xFFFF, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,     // 210 - า
  0xFFFF, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,     // 211 - ำ
  0x0FFF, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03FF,     // 212 - ิ
  0x03C0, 0x03FF, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x03FF,     // 213 - ี
  0x03C0, 0x03FF, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,     // 214 - ึ
  0x0FFF, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C, 0x0F3C,     // 215 - ื
  0xFFFF, 0x0F3C, 0x0F3C, 0x0F3C,
  0x0F3C, 0x0F3C },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0xFFFF,     // 216 - ุ
  0x03C0, 0xFFFF, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,     // 217 - ู
  0xFFC0, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },
```

```
{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 218 -
  0x03FF, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,      // 219 -
  0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
  0xFFFF, 0xFFFF },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // 220 - _
  0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
  0xFFFF, 0xFFFF },

{ 0xFF00, 0xFF00, 0xFF00, 0xFF00, 0xFF00,      // 221 - □
  0xFF00, 0xFF00, 0xFF00, 0xFF00,
  0xFF00, 0xFF00 },

{ 0x00FF, 0x00FF, 0x00FF, 0x00FF, 0x00FF,      // 222 - □
  0x00FF, 0x00FF, 0x00FF, 0x00FF,
  0x00FF, 0x00FF },

{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,      // 223 - ■
  0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x3F3C, 0xF3F0,      // 224 - เ
  0xF0C0, 0xF3F0, 0x3F3C, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3FC0, 0xF0F0, 0xFFC0,      // 225 - แ
  0xF0F0, 0xFFC0, 0xF000, 0xF000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFFF0, 0xF0F0, 0xF000,      // 226 - โ
  0xF000, 0xF000, 0xF000, 0xF000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFFFC, 0x3CF0, 0x3CF0,      // 227 - ใ
  0x3CF0, 0x3CF0, 0x3CF0, 0x3CF0,
  0x0000, 0x0000 },

{ 0x0000, 0xFFF0, 0xF0F0, 0x3C00, 0x0F00,      // 228 - ไ
  0x3C00, 0xF0F0, 0xF0F0, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x3FFC, 0xF3C0,      // 229 - ๅ
  0xF3C0, 0xF3C0, 0xF3C0, 0x3F00,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3C3C, 0x3C3C, 0x3C3C,      // 230 - ๆ
  0x3C3C, 0x3C3C, 0x3FF0, 0x3C00,
  0xF000, 0x0000 },
```

```
{ 0x0000, 0x0000, 0x3F3C, 0xF3F0, 0x03C0,      // 231 - ฯ
  0x03C0, 0x03C0, 0x03C0, 0x03C0,
  0x0000, 0x0000 },

{ 0x0000, 0xFFF0, 0x0F00, 0x3FC0, 0xF0F0,      // 232 - ะ
  0xF0F0, 0x3FC0, 0x0F00, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0x0FC0, 0x3CF0, 0xF03C, 0xFFFC,      // 233 - ั
  0xF03C, 0xF03C, 0x3CF0, 0x0FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x0FC0, 0x3CF0, 0xF03C, 0xF03C,      // 234 - ี
  0x3CF0, 0x3CF0, 0x3CF0, 0xFCFC,
  0x0000, 0x0000 },

{ 0x0000, 0x03F0, 0x0F00, 0x03C0, 0x3FF0,      // 235 - ึ
  0xF0F0, 0xF0F0, 0xF0F0, 0x3FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x3FFC, 0xF3CF,      // 236 - ื
  0xF3CF, 0x3FFC, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x003C, 0x00F0, 0x3FFC, 0xF3CF,      // 237 - ุ
  0xF3CF, 0x3FFC, 0x3C00, 0xF000,
  0x0000, 0x0000 },

{ 0x0000, 0x0FC0, 0x3C00, 0xF000, 0xFFC0,      // 238 - ู
  0xF000, 0xF000, 0x3C00, 0x0FC0,
  0x0000, 0x0000 },

{ 0x0000, 0x3FC0, 0xF0F0, 0xF0F0, 0xF0F0,      // 239 - ฺ
  0xF0F0, 0xF0F0, 0xF0F0, 0xF0F0,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0xFFF0, 0x0000, 0xFFF0,      // 240 - ๐
  0x0000, 0xFFF0, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0F00, 0x0F00, 0xFFF0, 0x0F00,      // 241 - ๑
  0x0F00, 0x0000, 0x0000, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0x3C00, 0x0F00, 0x03C0, 0x0F00,      // 242 - ๒
  0x3C00, 0x0000, 0x0000, 0xFFF0,
  0x0000, 0x0000 },

{ 0x0000, 0x03C0, 0x0F00, 0x3C00, 0x0F00,      // 243 - ๓
  0x03C0, 0x0000, 0x0000, 0xFFF0,
  0x0000, 0x0000 },
```

```
{ 0x0000, 0x00FC, 0x03CF, 0x03CF, 0x03C0,       // 244 - ๔
  0x03C0, 0x03C0, 0x03C0, 0x03C0,
  0x03C0, 0x03C0 },

{ 0x03C0, 0x03C0, 0x03C0, 0x03C0, 0x03C0,       // 245 - ๕
  0x03C0, 0xF3C0, 0xF3C0, 0x3F00,
  0x0000, 0x0000 },

{ 0x0000, 0x0F00, 0x0F00, 0x0000, 0xFFF0,       // 246 - ๖
  0x0000, 0x0F00, 0x0F00, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x3F3C, 0xF3F0, 0x0000,       // 247 - ๗
  0x3F3C, 0xF3F0, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0FC0, 0x3CF0, 0x3CF0, 0x0FC0,       // 248 - ๘
  0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x03C0,       // 249 - ๙
  0x03C0, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,       // 250 - ๚
  0x03C0, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x00FF, 0x00F0, 0x00F0, 0x00F0,       // 251 - ๛
  0xFCF0, 0x3CF0, 0x3CF0, 0x0FF0,
  0x03F0, 0x0000 },

{ 0x0000, 0x3FC0, 0x3CF0, 0x3CF0, 0x3CF0,       // 252 - ๜
  0x3CF0, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x3F00, 0x03C0, 0x0F00, 0x3C00,       // 253 - ๝
  0x3FC0, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0FF0, 0x0FF0,       // 254 - ๞
  0x0FF0, 0x0FF0, 0x0FF0, 0x0000,
  0x0000, 0x0000 },

{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,       // 255 - □
  0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000 }
};
```

```
//********************** CODEWORD.CPP ***************************//

struct  code_entry                  // codeword translation entries
  {
  unsigned
  char code;                        // codeword (right justified)
  int  value;                       // value of codeword
  } code_bl2[] =                    // 2 bit black codewords
    {
    {  2,   3 },                    // 10
    {  3,   2 },                    // 11
    {  0,   0 }
    },
  code_bl3[] =                      // 3 bit black codewords
    {
    {  2,   1 },                    // 010
    {  3,   4 },                    // 011
    {  0,   0 }
    },
  code_wh4[] =                      // 4 bit white codewords
    {
    {  7,   2 },                    // 0111
    {  8,   3 },                    // 1000
    { 11,   4 },                    // 1011
    { 12,   5 },                    // 1100
    { 14,   6 },                    // 1110
    { 15,   7 },                    // 1111
    {  0,   0 }
    },
  code_bl4[] =                      // 4 bit black codewords
    {
    {  2,   6 },                    // 0010
    {  3,   5 },                    // 0011
    {  0,   0 }
    },
  code_wh5[] =                      // 5 bit white codewords
    {
    {  7,  10 },                    // 00111
    {  8,  11 },                    // 01000
    { 18, 128 },                    // 10010
    { 19,   8 },                    // 10011
    { 20,   9 },                    // 10100
    { 27,  64 },                    // 11011
    {  0,   0 }
    },
  code_bl5[] =                      // 5 bit black codewords
    {
    {  3,   7 },                    // 00011
    {  0,   0 }
    },
```

```
code_wh6[] =                    // 6 bit white codewords
  {
  {  3,  13 },                  // 000011
  {  7,   1 },                  // 000111
  {  8,  12 },                  // 001000
  { 23, 192 },                  // 010111
  { 24, 1664 },                 // 011000
  { 42,  16 },                  // 101010
  { 43,  17 },                  // 101011
  { 52,  14 },                  // 110100
  { 53,  15 },                  // 110101
  {  0,   0 }
  },
code_bl6[] =                    // 6 bit black codewords
  {
  {  4,   9 },                  // 000100
  {  5,   8 },                  // 000101
  {  0,   0 }
  },
code_wh7[] =                    // 7 bit white codewords
  {
  {  3,  22 },                  // 0000011
  {  4,  23 },                  // 0000100
  {  8,  20 },                  // 0001000
  { 12,  19 },                  // 0001100
  { 19,  26 },                  // 0010011
  { 23,  21 },                  // 0010111
  { 24,  28 },                  // 0011000
  { 36,  27 },                  // 0100100
  { 39,  18 },                  // 0100111
  { 40,  24 },                  // 0101000
  { 43,  25 },                  // 0101011
  { 55, 256 },                  // 0110111
  {  0,   0 }
  },
code_bl7[] =                    // 7 bit black codewords
  {
  {  4,  10 },                  // 0000100
  {  5,  11 },                  // 0000101
  {  7,  12 },                  // 0000111
  {  0,   0 }
  },
code_wh8[] =                    // 8 bit white codewords
  {
  {  2,  29 },                  // 00000010
  {  3,  30 },                  // 00000011
  {  4,  45 },                  // 00000100
  {  5,  46 },                  // 00000101
  { 10,  47 },                  // 00001010
  { 11,  48 },                  // 00001011
  { 18,  33 },                  // 00010010
  { 19,  34 },                  // 00010011
```

```
    {  20,   35 },          // 00010100
    {  21,   36 },          // 00010101
    {  22,   37 },          // 00010110
    {  23,   38 },          // 00010111
    {  26,   31 },          // 00011010
    {  27,   32 },          // 00011011
    {  36,   53 },          // 00100100
    {  37,   54 },          // 00100101
    {  40,   39 },          // 00101000
    {  41,   40 },          // 00101001
    {  42,   41 },          // 00101010
    {  43,   42 },          // 00101011
    {  44,   43 },          // 00101100
    {  45,   44 },          // 00101101
    {  50,   61 },          // 00110010
    {  51,   62 },          // 00110011
    {  52,   63 },          // 00110100
    {  53,    0 },          // 00110101
    {  54,  320 },          // 00110110
    {  55,  384 },          // 00110111
    {  74,   59 },          // 01001010
    {  75,   60 },          // 01001011
    {  82,   49 },          // 01010010
    {  83,   50 },          // 01010011
    {  84,   51 },          // 01010100
    {  85,   52 },          // 01010101
    {  88,   55 },          // 01011000
    {  89,   56 },          // 01011001
    {  90,   57 },          // 01011010
    {  91,   58 },          // 01011011
    { 100,  448 },          // 01100100
    { 101,  512 },          // 01100101
    { 103,  640 },          // 01100111
    { 104,  576 },          // 01101000
    {   0,    0 }
  },
code_bl8[] =                // 8 bit black codewords
  {
  {   4,   13 },            // 00000100
  {   7,   14 },            // 00000111
  {   0,    0 }
  },
code_wh9[] =                // 9 bit white codewords
  {
  { 152, 1472 },            // 010011000
  { 153, 1536 },            // 010011001
  { 154, 1600 },            // 010011010
  { 155, 1728 },            // 010011011
  { 204,  704 },            // 011001100
  { 205,  768 },            // 011001101
  { 210,  832 },            // 011010010
  { 211,  896 },            // 011010011
```

```
    { 212,  960 },                  // 011010100
    { 213, 1024 },                  // 011010101
    { 214, 1088 },                  // 011010110
    { 215, 1152 },                  // 011010111
    { 216, 1216 },                  // 011011000
    { 217, 1280 },                  // 011011001
    { 218, 1344 },                  // 011011010
    { 219, 1408 },                  // 011011011
    {   0,    0 }
    },
code_bl9[] =                        // 9 bit black codewords
    {
    {  24,   15 },                  // 000011000
    {   0,    0 }
    },
code_bl10[] =                       // 10 bit black codewords
    {
    {   8,   18 },                  // 0000001000
    {  15,   64 },                  // 0000001111
    {  23,   16 },                  // 0000010111
    {  24,   17 },                  // 0000011000
    {  55,    0 },                  // 0000110111
    {   0,    0 }
    },
code_bl11[] =                       // 11 bit black codewords
    {
    {  23,   24 },                  // 00000010111
    {  24,   25 },                  // 00000011000
    {  40,   23 },                  // 00000101000
    {  55,   22 },                  // 00000110111
    { 103,   19 },                  // 00001100111
    { 104,   20 },                  // 00001101000
    { 108,   21 },                  // 00001101100
    {   0,    0 }
    },
code_bl12[] =                       // 12 bit black codewords
    {
    {  36,   52 },                  // 000000100100
    {  39,   55 },                  // 000000100111
    {  40,   56 },                  // 000000101000
    {  43,   59 },                  // 000000101011
    {  44,   60 },                  // 000000101100
    {  51,  320 },                  // 000000110011
    {  52,  384 },                  // 000000110100
    {  53,  448 },                  // 000000110101
    {  55,   53 },                  // 000000110111
    {  56,   54 },                  // 000000111000
    {  82,   50 },                  // 000001010010
    {  83,   51 },                  // 000001010011
    {  84,   44 },                  // 000001010100
    {  85,   45 },                  // 000001010101
    {  86,   46 },                  // 000001010110
```

```
    {  87,   47 },          // 0000001010111
    {  88,   57 },          // 0000001011000
    {  89,   58 },          // 0000001011001
    {  90,   61 },          // 0000001011010
    {  91,  256 },          // 0000001011011
    { 100,   48 },          // 0000001100100
    { 101,   49 },          // 0000001100101
    { 102,   62 },          // 0000001100110
    { 103,   63 },          // 0000001100111
    { 104,   30 },          // 0000001101000
    { 105,   31 },          // 0000001101001
    { 106,   32 },          // 0000001101010
    { 107,   33 },          // 0000001101011
    { 108,   40 },          // 0000001101100
    { 109,   41 },          // 0000001101101
    { 200,  128 },          // 0000011001000
    { 201,  192 },          // 0000011001001
    { 202,   26 },          // 0000011001010
    { 203,   27 },          // 0000011001011
    { 204,   28 },          // 0000011001100
    { 205,   29 },          // 0000011001101
    { 210,   34 },          // 0000011010010
    { 211,   35 },          // 0000011010011
    { 212,   36 },          // 0000011010100
    { 213,   37 },          // 0000011010101
    { 214,   38 },          // 0000011010110
    { 215,   39 },          // 0000011010111
    { 218,   42 },          // 0000011011010
    { 219,   43 },          // 0000011011011
    {   0,    0 }
  },
code_bl13[] =               // 13 bit black codewords
  {
    {  74,  640 },          // 0000001001010
    {  75,  704 },          // 0000001001011
    {  76,  768 },          // 0000001001100
    {  77,  832 },          // 0000001001101
    {  82, 1280 },          // 0000001010010
    {  83, 1344 },          // 0000001010011
    {  84, 1408 },          // 0000001010100
    {  85, 1472 },          // 0000001010101
    {  90, 1536 },          // 0000001011010
    {  91, 1600 },          // 0000001011011
    { 100, 1664 },          // 0000001100100
    { 101, 1728 },          // 0000001100101
    { 108,  512 },          // 0000001101100
    { 109,  576 },          // 0000001101101
    { 114,  896 },          // 0000001110010
    { 115,  960 },          // 0000001110011
    { 116, 1024 },          // 0000001110100
    { 117, 1088 },          // 0000001110101
    { 118, 1152 },          // 0000001110110
```

```
    { 119, 1216 },                  // 0000001110111
    {   0,    0 }
    },
  *code_table[13][2] =              // codeword table pointer
    {
    {       0,       0 },          // 1 bit entry
    {       0, code_bl2  },        // 2 bit entries
    {       0, code_bl3  },        // 3 bits entries
    { code_wh4, code_bl4  },       // 4 bits entries
    { code_wh5, code_bl5  },       // 5 bits entries
    { code_wh6, code_bl6  },       // 6 bits entries
    { code_wh7, code_bl7  },       // 7 bits entries
    { code_wh8, code_bl8  },       // 8 bits entries
    { code_wh9, code_bl9  },       // 9 bits entries
    {       0, code_bl10 },        // 10 bits entries
    {       0, code_bl11 },        // 11 bits entries
    {       0, code_bl12 },        // 12 bits entries
    {       0, code_bl13 },        // 13 bits entries
    };

struct  encode_entry                // encode entry
  {
  unsigned
  char code,                        // codeword (right justified)
    bits;                           // length of codeword in bits
  } encode_table[91][2] =           // encode codeword table
    {
    { {  53,  8 }, {  55, 10 } },   //   0
    { {   7,  6 }, {   2,  3 } },   //   1
    { {   7,  4 }, {   3,  2 } },   //   2
    { {   8,  4 }, {   2,  2 } },   //   3
    { {  11,  4 }, {   3,  3 } },   //   4
    { {  12,  4 }, {   3,  4 } },   //   5
    { {  14,  4 }, {   2,  4 } },   //   6
    { {  15,  4 }, {   3,  5 } },   //   7
    { {  19,  5 }, {   5,  6 } },   //   8
    { {  20,  5 }, {   4,  6 } },   //   9
    { {   7,  5 }, {   4,  7 } },   //  10
    { {   8,  5 }, {   5,  7 } },   //  11
    { {   8,  6 }, {   7,  7 } },   //  12
    { {   3,  6 }, {   4,  8 } },   //  13
    { {  52,  6 }, {   7,  8 } },   //  14
    { {  53,  6 }, {  24,  9 } },   //  15
    { {  42,  6 }, {  23, 10 } },   //  16
    { {  43,  6 }, {  24, 10 } },   //  17
    { {  39,  7 }, {   8, 10 } },   //  18
    { {  12,  7 }, { 103, 11 } },   //  19
    { {   8,  7 }, { 104, 11 } },   //  20
    { {  23,  7 }, { 108, 11 } },   //  21
    { {   3,  7 }, {  55, 11 } },   //  22
    { {   4,  7 }, {  40, 11 } },   //  23
    { {  40,  7 }, {  23, 11 } },   //  24
```

```
{{  43,  7 }, {  24, 11 }},     // 25
{{  19,  7 }, { 202, 12 }},     // 26
{{  36,  7 }, { 203, 12 }},     // 27
{{  24,  7 }, { 204, 12 }},     // 28
{{   2,  8 }, { 205, 12 }},     // 29
{{   3,  8 }, { 104, 12 }},     // 30
{{  26,  8 }, { 105, 12 }},     // 31
{{  27,  8 }, { 106, 12 }},     // 32
{{  18,  8 }, { 107, 12 }},     // 33
{{  19,  8 }, { 210, 12 }},     // 34
{{  20,  8 }, { 211, 12 }},     // 35
{{  21,  8 }, { 212, 12 }},     // 36
{{  22,  8 }, { 213, 12 }},     // 37
{{  23,  8 }, { 214, 12 }},     // 38
{{  40,  8 }, { 215, 12 }},     // 39
{{  41,  8 }, { 108, 12 }},     // 40
{{  42,  8 }, { 109, 12 }},     // 41
{{  43,  8 }, { 218, 12 }},     // 42
{{  44,  8 }, { 219, 12 }},     // 43
{{  45,  8 }, {  84, 12 }},     // 44
{{   4,  8 }, {  85, 12 }},     // 45
{{   5,  8 }, {  86, 12 }},     // 46
{{  10,  8 }, {  87, 12 }},     // 47
{{  11,  8 }, { 100, 12 }},     // 48
{{  82,  8 }, { 101, 12 }},     // 49
{{  83,  8 }, {  82, 12 }},     // 50
{{  84,  8 }, {  83, 12 }},     // 51
{{  85,  8 }, {  36, 12 }},     // 52
{{  36,  8 }, {  55, 12 }},     // 53
{{  37,  8 }, {  56, 12 }},     // 54
{{  88,  8 }, {  39, 12 }},     // 55
{{  89,  8 }, {  40, 12 }},     // 56
{{  90,  8 }, {  88, 12 }},     // 57
{{  91,  8 }, {  89, 12 }},     // 58
{{  74,  8 }, {  43, 12 }},     // 59
{{  75,  8 }, {  44, 12 }},     // 60
{{  50,  8 }, {  90, 12 }},     // 61
{{  51,  8 }, { 102, 12 }},     // 62
{{  52,  8 }, { 103, 12 }},     // 63
{{  27,  5 }, {  15, 10 }},     // 64
{{  18,  5 }, { 200, 12 }},     // 128
{{  23,  6 }, { 201, 12 }},     // 192
{{  55,  7 }, {  91, 12 }},     // 256
{{  54,  8 }, {  51, 12 }},     // 320
{{  55,  8 }, {  52, 12 }},     // 384
{{ 100,  8 }, {  53, 12 }},     // 448
{{ 101,  8 }, { 108, 13 }},     // 512
{{ 104,  8 }, { 109, 13 }},     // 576
{{ 103,  8 }, {  74, 13 }},     // 640
{{ 204,  9 }, {  75, 13 }},     // 704
{{ 205,  9 }, {  76, 13 }},     // 768
{{ 210,  9 }, {  77, 13 }},     // 832
```

```
{{ 211,  9 }, {  114, 13 }},     // 896
{{ 212,  9 }, {  115, 13 }},     // 960
{{ 213,  9 }, {  116, 13 }},     // 1024
{{ 214,  9 }, {  117, 13 }},     // 1088
{{ 215,  9 }, {  118, 13 }},     // 1152
{{ 216,  9 }, {  119, 13 }},     // 1216
{{ 217,  9 }, {   82, 13 }},     // 1280
{{ 218,  9 }, {   83, 13 }},     // 1344
{{ 219,  9 }, {   84, 13 }},     // 1408
{{ 152,  9 }, {   85, 13 }},     // 1472
{{ 153,  9 }, {   90, 13 }},     // 1536
{{ 154,  9 }, {   91, 13 }},     // 1600
{{  24,  6 }, {  100, 13 }},     // 1664
{{ 155,  9 }, {  101, 13 }}      // 1728
};
```

```
//**********File Converted Module **********//

#include <stdio.h>                    // standard i/o library
#include <process.h>                  // exit func_
#include <stdarg.h>                   // variable argument list
#include <string.h>                   // string handling routines
#include <stdlib.h>                   // std conversion routines
#include <assert.h>                   // assertion routines
#include <dos.h>                      // dos functions
#include <ctype.h>                    // character routines
#include <conio.h>                    // console functions
#include <bios.h>                     // bios functions
#include <dir.h>                      // directory routines
#include <malloc.h>                   // memory routines
#include <io.h>                       // file i/o functions
#include <fcntl.h>                    // access symbolics
#include <graphics.h>                 // graphics routines
#include <sys\stat.h>                 // dos create fnc flags
#include "keys.h"                     // keyboard definitions
#define COUNT(x)    (sizeof(x) / sizeof(x[0]))    // item count
#define NOT         !                 // shorthand logical
#define BYTE        char              // single byte
#define UINT        unsigned int      // unsigned integer
#define UCHAR       unsigned char     // unsigned char
#define MAX_PATH    79                // maximum path length
#define MIX(x,y)    ((x << 4) + (y))  // mix colors for fg and bg
#define FG(x)       (unsigned char) x >> 4  // extract foreground color
#define BG(x)       x & 0x07          // ..and background color
#define IN(x)       inportb(base + x)       // read a UART register
#define OUT(x,y)    outportb(base + x, y)   // ..and write a register
#define PELS        1728              // pixels per line
#define LINE        PELS / 8          // bytes per line
#define LINES       1143              // lines per page
#define PAGE        ((long)LINE * LINES)    // bitmap size
#define COLUMNS     PELS / 16         // max chars per line
#define ROWS        LINES / 22        // ..and max lines per page
#define DLE         0x10              // DLE character
#define ETX         0x3               // ..and ETX character
#define ESC_CHAR    "\x1b"            // escape char for printer
#define HDR_LEN     128               // header length
#define BUF_SIZE    256               // buffer size
#include "codeword.cpp"               // codewords for G3 encoding
#include "asciimap.cpp"               // bitmap of ASCII chars


void    con(char *),
            f_write_hdr(void),
            f_write_page(),
        *malloc_chk(long n),
            f_build_char(char cc,int  c, int r),
            f_write_out(int bits,int len),
```

```
        f_encode(int cnt,int type),
    quit_with(char *msg, ...),
    snd();

int   f_build_page(FILE *f),
        f_locate(char *s),
        f_scan_out(char huge *p,int  i,int  type),
        f_get_bit(void),
        f_get_bit(void),
        f_get_byte(void),
        f_get_next_byte(void),
        f_handle =0,                    // G3 file handle
        f_pgcnt,                        // maximum page count
        f_write_flag,                   // write_out bit buffer flag
        f_write_cnt;                    // write_out global counter


char  *reverse_scan(char *p,char c,int  len),
        *first_nonblank(char *s),
    *f_buffer,                          // work buffer
    *f_ptr,                             // ..and current pointer
        f_filename[MAX_PATH],           // fax filename
        fileconvert[]=" File converted module active !",
        create_msg[]   = "\n\r Creating: %s\n\r",
        status_conv[]  = "Converting: %s",
        status_file[]  = "%-25.25s %d Page",
//      huge *page,                     // page bitmap (1728 x 1143)
        write_msg[]    = "   Page %d, %ld%% complete   \r",
        no_memory[]    = "Insufficient memory to continue processing\n",
        read_error[]   = "Error reading G3 file\n";


UINT  get_bit(unsigned char *s,int  n);
long  f_page;                           // current G3 page size


char huge *page = (char *) malloc_chk(PAGE);      // get memory for bitmap
/* ************************************************************** *
 *  con() – build a fax file from an ASCII text file
 * ************************************************************** */

void  con(char *av)

{
FILE  *fi;                              // input file
int   i;                                // string index
```

```
char    buf[60], buf2[60],              // format buffers
        *p;                             // ..and pointer

//if((l=strlen(av) == 0)
//return;

snd();
snd();

if ((fi = fopen(av, "rb")) == 0){       // q. open ok?
   snd();
   snd();
   return;
   }
else

   f_handle=-1;                         // a. no .. just return


strcpy(buf, av);                        // copy base filename

if ((p = strrchr(buf, '.')) == NULL)    // q. file extension found?
   p = &buf[strlen(buf)];               // a. no .. point to end

strcpy(p, ".fax");                      // copy in extension

if ((f_handle = open(buf,               // q. open new file ok?
            O_BINARY | O_TRUNC | O_RDWR | O_CREAT)) == -1)
   return;                              // a. no .. just return


if ((i = 34 - (strlen(av) +             // q. able to center the
         strlen(status_conv) - 2)) < 0)     // .. 'Converting: XXXX" msg?
   i = 0;                               // a. no .. flush left

memset(buf2, ' ', sizeof(buf2));        // clear area to blanks
sprintf(&buf2[i / 2], status_conv, av);     // format build information

f_write_hdr();                          // write header information

for (f_pgcnt = 1; ; f_pgcnt++)          // loop building and writing
  {
  if (f_build_page(fi) == 0)            // q. read ASCII ok?
         break;                         // a. no .. end of file

  f_write_page();                       // encode bitmap & write file
  }

fclose(fi);                             // close input file
close(f_handle);                        // ..and output file
f_handle = -1;                          // ..and reset global
```

```
f_locate(buf);                  // set up to view/print file
sprintf(buf2, status_file,      // format file information
        buf, f_pgcnt);          // ..for status line

if (f_pgcnt > 1)                // q. more than 1 page fax?
   strcat(buf2, "s");           // a. yes .. pluralize page

//return(0);
}



/* ************************************************************** *
 * f_build_page() – read an ASCII text file and build a bitmap
 *
 * returns: 0 = end of file reached
 *          n = characters encoded
 * ************************************************************** */

int     f_build_page(FILE *f)           // input file pointer
{
int     i,                              // chars processed
        lines,                          // lines counter
        c, r,                           // column and row (0 based)
        cc;                             // character buffer
char    huge *p;                        // bitmap line pointer


for (lines = 0, p = page;               // for the whole bitmap
        lines < LINES;                  // ..run through line
        lines++, p += LINE)             // ..by line
   memset(p, 0, LINE);                  // ..and clear white space

c = r = 0;                              // start column & row at top

for (i = 0; ; i++)                      // loop building bitmap
   {
   if ((cc = fgetc(f)) == EOF)          // q. get a good character?
        return(i);                      // a. no .. return w/count


   f_build_char(cc, c, r);              // bld bitmap at column & row

   switch (cc)                          // update column & row
        {
        case LF:                        // line feed
           if (++r >= ROWS)             // q. reach bottom limit?
                return(i);              // a. yes .. then page is done

        break;                          // else .. get next character
```

```
        case CR:                    // carriage return
            c = 0;                  // start at next beginning
            break;                  // ..and get next character

        case 12:                    // form feed
            return(++i);            // return w/nbr chars processed

        case BACKSPACE:             // backspace
            if (--c < 0)            // q. backup too far?
                c = 0;              // a. yes .. goto column 0

            break;                  // else .. get next character

        case TAB:                   // tab
            c = (c & ~0x7) + 8;     // move to next tab stop

                if (c >= COLUMNS)           // q. too far out?
                {
                c = 0;                  // a. yes .. just wrap around

                if (++r >= ROWS)        // q. reach bottom of page?
                    return(i);          // a. yes .. stop here
                }

            break;                  // else .. get next character

        default:                    // everything else
            if (++c >= COLUMNS)         // q. reach right margin?
                {
                c = 0;                  // a. yes .. just wrap around

                if (++r >= ROWS)        // q. reach bottom of page?
                    return(i);          // a. yes .. stop here
                }
        }
    }
}


/* ************************************************************** *
 * f_write_page() -- write a fax file page
 * ************************************************************** */

void    f_write_page(void)          // window to display msgs in
{
int     i, j = 0,                   // loop counter
        line,                       // line counter
        type;                       // pixel type
char    buf[60],                    // work buffer
        huge *p;                    // ..and pointer
long    start_pos,                  // page starting position
        page_len;                   // ..and page length
```

```
static
long    percent,                        // percent complete
        last,                           // last display time
        far *timer = (long far *)       // BIOS timer tick counter
                                        MK_FP(0x40, 0x6c);  // ..down in low memory


lseek(f_handle, 0L, SEEK_END);          // goto end of file
start_pos = tell(f_handle);             // ..and get position info

memset(buf, 0, sizeof(buf));            // clear buffer to nulls
write(f_handle, buf, 4);                // write page length
write(f_handle, buf, 10);               // ..and some nulls

f_write_flag = 0;                       // clear output flag
f_write_out(1, 12);                     // ..and write an EOL

for (p = page, line = 0; line < LINES;      // for each line in the bitmap
        line++, p += LINE)                  // ..encode into G3 format
    {
    f_write_cnt = 0;                        // clear output byte count

    if ((*timer - last) > 9)                // q. has half a second past?
        {
        percent = (line * 100L) / LINES;    // a. yes .. get percentage
        sprintf(buf, write_msg,             // build status message
            f_pgcnt, percent);              // ..into a work buffer
        last = *timer;                      // ..and pickup current time
        }

    if (reverse_scan((char *) p, 0, LINE))  // q. anything on line?
        {                                   // a. yes .. process
        for (i = type = 0; i < PELS; i += j)// scan across the bitmap line
            {
            j = f_scan_out(p, i, type);     // scan bitmap for pixels

            type = (type + 1) & 1;          // flip-flop the type code
            }
        }
    else
        f_encode(1728, 0);                  // else .. put out blank line

    if (f_write_flag)                       // q. any residual bits?
        f_write_out(0, 8 - f_write_flag);   // a. yes .. put out the rest

    if (i < 40)                             // q. need to pad out line?
        write(f_handle, (void *) &buf,      // a. yes .. write a string
            40 - i);                        // ..of up to 40 null bytes

    f_write_out(1, 12);                     // then put out an EOL
    }
```

```
for (i = 0; i < 5; i++)                    // at end of page
   f_write_out(1, 12);                     // ..put out 5 more EOLs

buf[0] = DLE;                              // set up termination
buf[1] = ETX;                              // ..with DLE ETX
write(f_handle, buf, 2);                   // ..then put them out

page_len = tell(f_handle) - start_pos - 4;  // compute page data length
lseek(f_handle, start_pos, SEEK_SET);       // position to length field
write(f_handle, (void *) &page_len, 4);     //  and update field

}


/* ************************************************************** *
 *  f_write_hdr() – write a fax file header
 * ************************************************************** */

void    f_write_hdr(void)
{
char    buf[HDR_LEN];                      // header buffer


memset(buf, 0, HDR_LEN);                   // clear buffer to nulls
strcpy(buf, "G3");                         // ..and put in our marker
write(f_handle, buf, HDR_LEN);             // write header
}

/* ************************************************************** *
 *  f_locate() – find a G3 formatted file and check its integrity
 *  returns: 0 = file opened
 *           1 = file not found
 * ************************************************************** */

int     f_locate(char *s)                  // source filename
{
long    file_size,                         // file size
        pos;                               // current file position


if (f_handle != -1)                        // q. already open?
   {
   close(f_handle);                        // a. yes .. close file
   f_handle = -1;                          // ..and clear flag
   }

if (NOT first_nonblank(s))                 // q. empty string?
   return(1);                              // a. yes .. just return

if ((f_handle = open(s,                    // q. find and open file ok?
            O_RDONLY | O_BINARY)) == -1)
```

```
    return(1);                          // a. no .. return w/error

file_size = lseek(f_handle, 0L, SEEK_END);  // get file size
lseek(f_handle, 0L, SEEK_SET);              // ..and back to file start

if ((read(f_handle, page, 128) != 128) ||   // q. file header available
        strncmp((char *) page, "G3", 2))    // ..and our format?
    {
    close(f_handle);                        // a. no .. just close file
    f_handle = -1;                          // ..clear flag
    return(1);                              // ..and return to caller
    }

pos = lseek(f_handle, 0L, SEEK_CUR);        // get starting position

for (f_pgcnt = 0;;)                         // count available fax pages
    {
    if (read(f_handle, (char *) &f_page,    // q. read in enough to cover
          sizeof(long)) != sizeof(long))    // ..the page length field?
        break;                              // a. no .. exit loop

    if ((f_page < 1) ||                     // q. page size non-positive?
        ((f_page + pos + 4) > file_size))   // ..or bigger than the file?
        break;                              // a. yes .. exit loop

    if ((pos = lseek(f_handle, f_page,      // q. properly position to the
          SEEK_CUR)) == -1L)                // ..start of the next page?
        break;                              // a. no .. exit loop

    f_pgcnt++;                              // count this page
    }

strcpy(f_filename, s);                      // save filename ..
return(0);                                  // ..then return all ok

}



/* ************************************************************** *
 *  malloc_chk() - allocate memory with error processing
 * ************************************************************** */

void   *malloc_chk(long n)                  // size of block
{
void   *p;                                  // temporary pointer


if (NOT (p = (void *) _fmalloc(n)))         // q. enough memory?
    quit_with(no_memory);                   // a. no .. give error msg
```

```
return(p);                          // else .. return w/address

}



/* ************************************************************** *
 * f_build_char() – fill in bitmap position with a character
 * ************************************************************** */

void    f_build_char(char cc,int  c, int r)
{
UINT    *ce,                        // table entry pointer
        i;                          // loop counter
char    huge *p,                    // bitmap pointer
        ch1, ch2;                   // high and low bytes


ce = &ascii_map[cc][0];             // get entry in table

p = &page[((r * 22L) * LINE) + (c * 2)];    // get starting point

for (i = 0; i < 11; i++, ce++)      // for each of the chars rows
   {
   if (*ce)                         // q. anything to put in bitmap?
      {                             // a. yes .. process bits
      ch1 = *ce >> 8;               // get high order byte
      ch2 = *ce & 0xff;             // ..and low order byte

      *p |= ch1;                    // 'or' in each cols bits
      *(p + 1) |= ch2;              // ..for both bytes
      p += LINE;                    // move to next line

      *p |= ch1;                    // 'or' in each cols bits
      *(p + 1) |= ch2;              // ..for both bytes
      p += LINE;                    // move to next line
      }
   else
      p += LINE * 2;                // else .. skip two bitmap lines
   }

}



/* ************************************************************** *
 * f_write_out() – write bits to output fax file
 * ************************************************************** */

void    f_write_out(int bits,int len)
{
static  unsigned
```

```
    char    w;                      // byte wide queue


    if (len > 16)                   // q. too many bits?
        return;                     // a. yes .. just return

    bits <<= 16 - len;              // shift to MSB end of word

    while (len--)                   // while there is input
        {
        w >>= 1;                    // shift queue over by one

        if (bits & 0x8000)          // q. source bit on?
            w |= 0x80;              // a. yes .. turn on dest

        if (++f_write_flag == 8)    // q. reach limit?
            {
            write(f_handle, (void *) &w, 1);    // a. yes .. write a byte

            if (w == DLE)           // q. special character?
                write(f_handle, (void *) &w, 1);// a. yes .. write it again

            f_write_flag = 0;       // clear the flag
            f_write_cnt++;          // ..and tally the byte
            }

        bits <<= 1;                 // shift source by one bit
        }
    }


/* ************************************************************** *
 *  reverse_scan() -- backscan for dissimilar character
 * ************************************************************** */


char    *reverse_scan(char *p,char c,int  len)


{


for (p += len - 1; len--; p--)      // loop thru memory
    if (*p != c)                    // q. find last one?
        return(p);                  // a. yes .. return w/address

return(0);                          // else .. return empty handed

}
```



```
/* ************************************************************** *
```

```
 * f_scan_out() – scan the bitmap and output a codeword
 * ------------------------------------------------------------------ */

int     f_scan_out(char huge *p,int  i,int  type)


{
int     cnt;                            // work counter


for (cnt = 0; i < PELS; i++, cnt++)         // scan bitmap line
   if (get_bit(                         // q. find a bit which is not
         (unsigned char *) p, i) != type)// ..the same as type?
      break;                            // a. yes .. exit loop

f_encode(cnt, type);                    // build/output codeword
return(cnt);                            // ..then return with count

}


/* ------------------------------------------------------------------ *
 * f_encode() – encode a string of bits into G3 format
 * ------------------------------------------------------------------ */

void    f_encode(int cnt,int type)

{
struct  encode_entry *ee;               // encode entry


if (cnt > 63)                           // q. big run of bits?
   {
   ee = &encode_table[(cnt / 64) + 63][type];  // a. yes .. get pointer
   f_write_out(ee->code, ee->bits);         // ..write make-up code
   cnt %= 64;                           // ..and update bit count
   }


ee = &encode_table[cnt][type];          // get element pointer
f_write_out(ee->code, ee->bits);        // write terminating code


}


/* ------------------------------------------------------------------ *
 * first_nonblank() – find first non-blank character
 * ------------------------------------------------------------------ */

char    *first_nonblank(char *s)        // string to look through
{
for (; *s; s++)                         // loop thru string
```

```
    if (NOT isspace(*s))            // q. find a non-blank char?
        return(s);                  // a. yes .. return w/address

return(0);                          // else .. string is blank

}



/* ********************************************************************* *
 * quit_with() – give an error message, then return to DOS
 * ********************************************************************* */

void    quit_with(char *msg, ...)       // quit with an error message
{
printf("msg");
// va_list list;                        // variable list


//if (full_screen)                      // q. in full screen mode?
//  {
//    term->Close();                    // a. yes .. close term window
//    window(1, 1, 80, max_lines);      // set up termination screen
//    textcolor(FG(mono_1));            // ..with foreground
//    textbackground(BG(mono_1));       // ..and background colors
//    clrscr();                         // ..and clear screen
//    CURSOR();                         // ..and set cursor to normal
//    printf(copyright);                // display program banner
//  }

//_dos_setvect(0x1b, old_break);        // restore old ^break handler

//va_start(list, msg);                  // set up variable list
//vprintf(msg, list);                   // give error message
//
//exit(rc);                             // ..and then quit

}

/* ********************************************************************* *
 * f_get_bit() – get the next bit from the G3 file
 * returns: -1 = end of data or error
 *           0 = zero bit found
 *           1 = one bit found
 * ********************************************************************* */

int     f_get_bit(void)
{
static
int     w;                          // work byte
        c = 0;                      // bit count
```

```
w >>= 1;                        // move next bit into place

if (f_ptr == 0 || c == 0)       // q. 1st call or out of data?
   {                            // a. yes .. get a byte
   if ((w = f_get_byte()) == -1)    // q. out of data?
      return(-1);               // a. yes .. return EOF

   c = 8;                       // number of available bits
   }

c--;                            // show another one used
return(w & 1);                  //  .then rtn a bit to caller

}

/* ********************************************************************
 * get_bit() - get a bit from a string of bits
 * ******************************************************************** */

UINT    get_bit(unsigned char *s,int   n)

{

return((s[n / 8] >> (7 - (n % 8))) & 1);    // return with requested bit

}



/* ********************************************************************
 * f_get_byte() - get the next value from the G3 file handling DLEs
 *
 * returns: -1 = end of data or error
 *          n = next byte
 * ******************************************************************** */

int     f_get_byte(void)
{
int     v;                      // work value


while (1)
   {
   if ((v = f_get_next_byte()) == -1)    // q. out of data?
      return(-1);               // a. yes .. return EOF

   if (v == DLE)                // q. DLE character?
      {                         // a. yes .. get another
      if ((v = f_get_next_byte()) == -1) // q. get another char ok?
         return(-1);            // a. no .. return EOF
```

```
        if (v == DLE)                   // q. another DLE?
            return(v);                  // a. yes .. return the goods
        }
    else
        return(v);                      // else  return w/character
    }
}



/* ********************************************************************
 * f_get_next_byte() – get the next byte from the G3 file
 *
 * returns: -1 = end of data or error
 *          n = next byte
 * ******************************************************************** */

int     f_get_next_byte(void)
{
static
int     rb;                             // remaining buffer count


if (f_ptr == 0 || rb == 0)              // q. 1st call or out of data?
    {                                   // a. yes .. get the next blk
    if (f_page == 0)                    // q. out of data to read?
        return(-1);                     // a. yes .. return all done

    if (f_buffer == 0)                  // q. buffer allocated?
        f_buffer = (char *)             // a. no .. get a buffer
            malloc_chk(BUF_SIZE);


    rb = (int)((f_page > BUF_SIZE) ?    // set count to smaller of
            BUF_SIZE : f_page);         // ..what's left or buffer size

    if (read(f_handle, f_buffer, rb) != rb) // q. file read ok?
        quit_with(read_error);          // a. no .. give error message

    f_page -= rb;                       // deduct what was read
    f_ptr = f_buffer;                   // set up character pointer
    }

rb--;                                   // decriment remaining count
return(*(unsigned char *) f_ptr++);     // ..and return character

}

};
```

```
//****************************** Examine module ************************//
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <graphics.h>
#include <stdlib.h>
#include <bios.h>
#include <alloc.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <process.h>
#include "snd.cpp"
#define INTR 0x1C    /* The clock tick interrupt */


#ifdef __cplusplus
   #define __CPPARGS ...
#else
   #define __CPPARGS
#endif

void interrupt ( *oldhandler)(__CPPARGS);
void puthex(int ,int ,char * );
void search(int ,int *,int *,int *);    //0: up,1:dn,2:lf,3:rh/i1/i2/i3
void exam(int i1,int i2,int i3);


struct copies
{
unsigned char msg[256];
int len;
} sndrcve[20][4];

char *frame[4]={"Address Field",                    //0
                "Control Field",                    //1
                "Fax Control Field",                //2
                "Fax Information Field");           //3

char *detail[80]={"half duplex address",            //0
                  "final frame of transmission",    //1
                  "another frame follow",           //2
                  "digital identification signal(DIS)",  //3
                  "called subscriber identification(CSI)",  //4
                  "nonstandard facilities(NSF)",    //5
                  "digital transmit command(DTC)",  //6
                  "calling subscriber identification(CIG)",  //7
                  "nonstandard facilities command(NSC)",  //8
                  "digital command signal(DCS)",    //9
                  "transmitting subscriber ID (TSI)",  //10
                  "nonstandard facilities setup (NSS)",  //11
```

```
//          "training check frame (TCF)",          //
            "confirmation to receive (CFR)",        //12
            "failure to train (FTT)",              //13
            "end of message (EOM)",                 //14
            "multipage signal (MPS)",               //15
            "end of procedure (EOP)",               //16
            "procedure interrupts-end of message (PRI-EOM)", //17
            "procedure interrupts-multipage signal(PRI-MPS)",//18
            "procedure interrupts-end of procedure(PRI-EOP)",//19
            "message confirmation (MCF)",           //20
            "retrain positive (RTP)",              //21
            "retrain negative (RTN)",              //22
            "procedure interrupt positive (PIP)",   //23
            "procedure interrupt negative (PIN)",   //24
            "disconnect (DCN)",                    //25
            "command repeat (CRP)",                 //26
            "null character",                      //27
            "start of heading",                     //28
            "start of text",                       //29
            "end of text",                         //30
            "end of transmission",                  //31
            "enquiry",                             //32
            "acknowledge affirmative",              //33
            "audible alarm",                       //34
            "backspace one position",               //35
            "physical horizontal tab",              //36
            "line feed",                           //37
            "physical vertical tab",                //38
            "form feed",                           //39
            "carriage return",                      //40
            "shift out",                           //41
            "shift in",                            //42
            "data link escape",                     //43
            "xon or resume ",                       //44
            "device control",                       //45
            "xoff or pause",                        //46
            "device control",                       //47
            "negative acknowlage",                  //48
            "synchronous idle",                     //49
            "end of transmission",                  //50
            "cancel",                              //51
            "end of medium",                        //52
            "substitute",                          //53
            "escape",                              //54
            "file separator",                      //55
            "group separator",                      //56
            "record reparator",                     //57
            "unit separator",                      //58
            "delete",                              //59
            "..?..");                              //60
```

```
int userfont,gflag=1,i1=0,i2=0,i3;

void interrupt handler(__CPPARGS)
{
static int count=0,flag=0;

if((count>10)&&(gflag==1)){
if(flag==0){
setcolor(WHITE);
puthex((50+300*(i3-2)+6*i1),50+30*i2,&sndrcve[i2][i3].msg[i1]);
flag=1;
delay(50);
}
if(flag==1){
setcolor(LIGHTRED);
puthex((50+300*(i3-2)+6*i1),50+30*i2,&sndrcve[i2][i3].msg[i1]);
flag=0;
delay(70);
}
count=0;
}

else count+=1;

/* call the old routine */
  oldhandler();
}


void main()
{
int  gd=9,gm=2,c;

/* save the old interrupt vector */
  oldhandler = getvect(INTR);

/* install the new interrupt handler */
  setvect(INTR, handler);


   initgraph(&gd,&gm,"");


while(i2<4){
    while(i1<20){
          sndrcve[i1][i2].msg[0]='\0';
          sndrcve[i1][i2].len=0;
          i1+=1;
            }
          i1=0;
          i2+=1;
    }
```

```
memcpy(sndrcve[1][2].msg,"□ภ□้ฬฬL□□□□□□□□□□□□□□",25);
memcpy(sndrcve[2][2].msg,"□ฅ□",3);
memcpy(sndrcve[5][2].msg,"□ฅ!□□□",6);
memcpy(sndrcve[7][2].msg,"□ฅ1□□□□oใ□□            ",23);
memcpy(sndrcve[3][3].msg,"□ภย□□□□□□□□□□□□□□□□□□□□□i□□",27);
memcpy(sndrcve[4][3].msg,"□ฅม",3);
memcpy(sndrcve[6][3].msg,"□ฅαฬf□□oใ□□            ",23);

sndrcve[1][2].len=25;
sndrcve[2][2].len=3;
sndrcve[5][2].len=6;
sndrcve[7][2].len=23;
sndrcve[3][3].len=27;
sndrcve[4][3].len=3;
sndrcve[6][3].len=23;

setfillstyle(SOLID_FILL,WHITE);
bar(1,1,639,53);
setcolor(BLACK);
moveto(2,2);
lineto(638,2);
lineto(638,52);
lineto(2,52);
lineto(2,2);

setfillstyle(SOLID_FILL,LIGHTBLUE);
bar(1,54,639,430);
setcolor(BLACK);
moveto(2,55);
lineto(638,55);
lineto(638,429);
lineto(2,429);
lineto(2,55);

setfillstyle(SOLID_FILL,BLACK);
bar(1,431,639,479);
setcolor(LIGHTBLUE);
moveto(2,432);
lineto(638,432);
lineto(638,478);
lineto(2,478);
lineto(2,432);


setcolor(DARKGRAY);
outtextxy(150,30,"     EXPLANDED MODE EXAMINATION    ");
//outtextxy(150,38,"     ——————————     ");
setcolor(WHITE);
for(i3=2;i3<4;i3++)
for(i2=1;i2<8;i2++){
for(i1=0;i1<strlen(sndrcve[i2][i3].msg);i1++)
puthex((50+300*(i3-2)+6*i1),50+30*i2,&sndrcve[i2][i3].msg[i1]);
```

```
}

i1=i2=0;i3=2;
search(1,&i1,&i2,&i3);
setcolor(LIGHTRED);
puthex((50+300*(i3-2)+6*i1),50+30*i2,&sndrcve[i2][i3].msg[i1]);

while(c != 1)
{
c=0;
if(bioskey(1)!=0) c=(bioskey(0)&0xFF00)>>8;
if((c==72)||(c==80)||(c==75)||(c==77))
{
setcolor(WHITE);
gflag=0;
puthex((50+300*(i3-2)+6*i1),50+30*i2,&sndrcve[i2][i3].msg[i1]);
gflag=1;
switch(c)
{
case 27:
exit;
break;

case 72:
if(i2>1)
{
i2-=1;
search(0,&i1,&i2,&i3);
}
break;

case 80:
if(i2<10)
{
i2+=1;
search(1,&i1,&i2,&i3);
}
break;

case 75:
if((i1==0)&&(i3==3)) {
i3=2;i1=sndrcve[i2][i3].len-1;
search(2,&i1,&i2,&i3);
}
else {if(i1>0)i1-=1;}
break;

case 77:
if((i1==sndrcve[i2][i3].len-1)&&(i3==2)){
i3=3;i1=0;
search(3,&i1,&i2,&i3);
}
```

```
else (if(i1<sndrcve[i2][i3].len-1) i1+=1;}

break;

default:
c=0;
break;
}
setcolor(LIGHTRED);
gflag=0;
puthex((50+300*(i3-2)+6*i1),50+30*i2,&sndrcve[i2][i3].msg[i1]);
exam(i1,i2,i3);
gflag=1;
}
}


closegraph();
setvect(INTR, oldhandler);
}

void puthex(int x,int y,char ch[1])
{
userfont = installuserfont("litt.chr");
settextstyle(userfont, HORIZ_DIR, 4);
int c=0;
unsigned int iu_dl,iu_dh;
char ud[3],bck[1];
bck[0]=ch[0];
iu_dh=(unsigned int)((ch[0]>>4)&0x0F);
iu_dl=(unsigned int)(bck[0]&0x0F);

itoa(iu_dh,ud,16);
ud[2]='\0';
*(ud)=toupper(*(ud));

for(c=0;c<2;c++){
outtextxy(x,y,ud);
itoa(iu_dl,ud,16);
ud[2]='\0';
*(ud)=toupper(*(ud));
y+=8;
}
settextstyle(0,0,2);
}

void search(int i ,int *ii1,int *ii2,int *ii3)
{
int loop=1;
while(loop){
again:
switch(i)
```

```
{
case 0:
while(sndrcve[*ii2][*ii3].len==0)
{*ii2-=1;if(*ii2==1){i=1;goto again;}}
if(*ii1>sndrcve[*ii2][*ii3].len-1)
{*ii1=sndrcve[*ii2][*ii3].len-1;}
loop=0;
break;

case 1:
while(sndrcve[*ii2][*ii3].len==0)
{*ii2+=1;if(*ii2>10){i=0;goto again;}}
if(*ii1>sndrcve[*ii2][*ii3].len-1)
{ *ii1=sndrcve[*ii2][*ii3].len-1;}
loop=0;
break;

case 2:
while(sndrcve[*ii2][*ii3].len==0)
{*ii2+=1;if(*ii2>10){i=0;goto again;}}
*ii1=sndrcve[*ii2][*ii3].len-1;
loop=0;
break;

case 3:
while(sndrcve[*ii2][*ii3].len==0)
{*ii2+=1;if(*ii2>10){i=0;goto again;}}
*ii1=0;
loop=0;
break;

default:
break;
}
}
}


void exam(int i1,int i2,int i3)
{
char out[2];
settextstyle(0,0,1);
setfillstyle(0,0);
bar(4,440,635,475);
setcolor(YELLOW);
if(i1<3) outtextxy(250-(strlen(frame[i1])/2),440,frame[i1]);
if(i1>2) outtextxy(250-(strlen(frame[3])/2),440,frame[3]);

switch(sndrcve[i2][i3].msg[i1])
{
case 0xff:
outtextxy(250-(strlen(detail[0])/2),460,detail[0]);
```

```
break;

case 0xc8:
outtextxy(250-(strlen(detail[1])/2),460,detail[1]);
break;

case 0xc0:
outtextxy(250-(strlen(detail[2])/2),460,detail[2]);
break;

case 0x01:
if(i1==2)outtextxy(250-(strlen(detail[3])/2),460,detail[3]);

else outtextxy(250-(strlen(detail[28])/2),460,detail[28]);

break;

case 0x02:
if(i1==2)outtextxy(250-(strlen(detail[4])/2),460,detail[4]);

else outtextxy(250-(strlen(detail[29])/2),460,detail[29]);

break;

case 0x04:
if(i1==2)outtextxy(250-(strlen(detail[5])/2),460,detail[5]);

else outtextxy(250-(strlen(detail[31])/2),460,detail[31]);

break;

case 0x81:
outtextxy(250-(strlen(detail[6])/2),460,detail[6]);
break;

case 0x82:
outtextxy(250-(strlen(detail[7])/2),460,detail[7]);
break;

case 0x84:
outtextxy(250-(strlen(detail[8])/2),460,detail[8]);
break;

case 0xc1:
outtextxy(250-(strlen(detail[9])/2),460,detail[9]);
break;

case 0x41:
outtextxy(250-(strlen(detail[9])/2),460,detail[9]);
break;
```

```
case 0xc2:
outtextxy(250-(strlen(detail[10])/2),460,detail[10]);
break;

case 0x42:
outtextxy(250-(strlen(detail[10])/2),460,detail[10]);
break;

case 0xc3:
outtextxy(250-(strlen(detail[11])/2),460,detail[11]);
break;

case 0x43:
outtextxy(250-(strlen(detail[11])/2),460,detail[11]);
break;

case 0xa1:
outtextxy(250-(strlen(detail[12])/2),460,detail[12]);
break;

case 0x21:
outtextxy(250-(strlen(detail[12])/2),460,detail[12]);
break;

case 0xa2:
outtextxy(250-(strlen(detail[13])/2),460,detail[13]);
break;

case 0x22:
outtextxy(250-(strlen(detail[13])/2),460,detail[13]);
break;

case 0xf1:
outtextxy(250-(strlen(detail[14])/2),460,detail[14]);
break;

case 0x71:
outtextxy(250-(strlen(detail[14])/2),460,detail[14]);
break;

case 0xf2:
outtextxy(250-(strlen(detail[15])/2),460,detail[15]);
break;

case 0x72:
outtextxy(250-(strlen(detail[15])/2),460,detail[15]);
break;

case 0xf4:
outtextxy(250-(strlen(detail[16])/2),460,detail[16]);
break;
```

```
case 0x74:
outtextxy(250-(strlen(detail[16])/2),460,detail[16]);
break;

case 0xf9:
outtextxy(250-(strlen(detail[17])/2),460,detail[17]);
break;

case 0x79:
outtextxy(250-(strlen(detail[17])/2),460,detail[17]);
break;

case 0xfa:
outtextxy(250-(strlen(detail[18])/2),460,detail[18]);
break;

case 0x7a:
outtextxy(250-(strlen(detail[18])/2),460,detail[18]);
break;


case 0xfc:
outtextxy(250-(strlen(detail[19])/2),460,detail[19]);
break;

case 0x7c:
outtextxy(250-(strlen(detail[19])/2),460,detail[19]);
break;

case 0xb1:
outtextxy(250-(strlen(detail[20])/2),460,detail[20]);
break;

case 0x31:
outtextxy(250-(strlen(detail[20])/2),460,detail[20]);
break;

case 0xb3:
outtextxy(250-(strlen(detail[21])/2),460,detail[21]);
break;

case 0x33:
outtextxy(250-(strlen(detail[21])/2),460,detail[21]);
break;

case 0xb2:
outtextxy(250-(strlen(detail[22])/2),460,detail[22]);
break;

case 0x32:
outtextxy(250-(strlen(detail[22])/2),460,detail[22]);
break;
```

```
case 0xb5:
outtextxy(250-(strlen(detail[23])/2),460,detail[23]);
break;

case 0x35:
outtextxy(250-(strlen(detail[23])/2),460,detail[23]);
break;

case 0xb4:
outtextxy(250-(strlen(detail[24])/2),460,detail[24]);
break;

case 0x34:
outtextxy(250-(strlen(detail[24])/2),460,detail[24]);
break;

case 0xdf:
outtextxy(250-(strlen(detail[25])/2),460,detail[25]);
break;

case 0x5f:
outtextxy(250-(strlen(detail[25])/2),460,detail[25]);
break;

case 0xd8:
outtextxy(250-(strlen(detail[26])/2),460,detail[26]);
break;

case 0x58:
outtextxy(250-(strlen(detail[26])/2),460,detail[26]);
break;

case 0:
outtextxy(250-(strlen(detail[27])/2),460,detail[27]);
break;

case 3:
outtextxy(250-(strlen(detail[30])/2),460,detail[30]);
break;

case 5:
outtextxy(250-(strlen(detail[32])/2),460,detail[32]);
break;

case 6:
outtextxy(250-(strlen(detail[33])/2),460,detail[33]);
break;

case 7:
outtextxy(250-(strlen(detail[34])/2),460,detail[34]);
break;
```

```
case 8:
outtextxy(250-(strlen(detail[35])/2),460,detail[35]);
break;

case 9:
outtextxy(250-(strlen(detail[36])/2),460,detail[36]);
break;


case 10:
outtextxy(250-(strlen(detail[37])/2),460,detail[37]);
break;

case 11:
outtextxy(250-(strlen(detail[38])/2),460,detail[38]);
break;

case 12:
outtextxy(250-(strlen(detail[39])/2),460,detail[39]);
break;

case 13:
outtextxy(250-(strlen(detail[40])/2),460,detail[40]);
break;

case 14:
outtextxy(250-(strlen(detail[41])/2),460,detail[41]);
break;

case 15:
outtextxy(250-(strlen(detail[42])/2),460,detail[42]);
break;

case 16:
outtextxy(250-(strlen(detail[43])/2),460,detail[43]);
break;

case 17:
outtextxy(250-(strlen(detail[44])/2),460,detail[44]);
break;

case 18:
outtextxy(250-(strlen(detail[45])/2),460,detail[45]);
break;


case 19:
outtextxy(250-(strlen(detail[46])/2),460,detail[46]);
break;
case 20:
```

```
outtextxy(250-(strlen(detail[47])/2),460,detail[47]);
break;

case 21:
outtextxy(250-(strlen(detail[48])/2),460,detail[48]);
break;

case 22:
outtextxy(250-(strlen(detail[49])/2),460,detail[49]);
break;

case 23:
outtextxy(250-(strlen(detail[50])/2),460,detail[50]);
break;

case 24:
outtextxy(250-(strlen(detail[51])/2),460,detail[51]);
break;

case 25:
outtextxy(250-(strlen(detail[52])/2),460,detail[52]);
break;

case 26:
outtextxy(250-(strlen(detail[53])/2),460,detail[53]);
break;

case 27:
outtextxy(250-(strlen(detail[54])/2),460,detail[54]);
break;

case 28:
outtextxy(250-(strlen(detail[55])/2),460,detail[55]);
break;

case 29:
outtextxy(250-(strlen(detail[56])/2),460,detail[56]);
break;

case 30:
outtextxy(250-(strlen(detail[57])/2),460,detail[57]);
break;

case 31:
outtextxy(250-(strlen(detail[58])/2),460,detail[58]);
break;

case 127:
outtextxy(250-(strlen(detail[59])/2),460,detail[59]);
break;
```

```
default:
out[0]=sndrcve[i2][i3].msg[i1];
out[1]='\0';
outtextxy(250,460,out);
break;

}
}
```

```
// ********** Keys include **********//

#define F1          0x100 + '\x3b'      // F1 function key
#define F2          0x100 + '\x3c'      // F2
#define F3          0x100 + '\x3d'      // F3
#define F4          0x100 + '\x3e'      // F4
#define F5          0x100 + '\x3f'      // F5
#define F6          0x100 + '\x40'      // F6
#define F7          0x100 + '\x41'      // F7
#define F8          0x100 + '\x42'      // F8
#define F9          0x100 + '\x43'      // F9
#define F10         0x100 + '\x44'      // F10
#define F11         0x100 + '\x85'      // F11
#define F12         0x100 + '\x86'      // F12

#define UP          0x100 + '\x48'      // up
#define DOWN        0x100 + '\x50'      // down
#define LEFT        0x100 + '\x4b'      // left arrow
#define RIGHT       0x100 + '\x4d'      // right arrow
#define HOME        0x100 + '\x47'      // home
#define END         0x100 + '\x4f'      // end
#define PAGE_UP     0x100 + '\x49'      // page up
#define PAGE_DOWN   0x100 + '\x51'      // page down

#define C_UP        0x100 + '\x8d'      // ctrl up
#define C_DOWN      0x100 + '\x91'      // ctrl down
#define C_LEFT      0x100 + '\x73'      // ctrl left arrow
#define C_RIGHT     0x100 + '\x74'      // ctrl right arrow
#define C_HOME      0x100 + '\x77'      // ctrl home
#define C_END       0x100 + '\x75'      // ctrl end
#define C_PAGE_UP   0x100 + '\x84'      // ctrl page up
#define C_PAGE_DOWN 0x100 + '\x76'      // ctrl page down

#define SPACE       ' '                 // spacebar
#define CR          '\r'                // carriage return
#define LF          '\n'                // linefeed
#define ESC         '\x1b'              // escape
#define BACKSPACE   '\b'                // backspace
#define DELETE      0x100 + '\x53'      // delete key
#define INSERT      0x100 + '\x52'      // insert key
#define TAB         '\t'                // tab
#define BELL        '\a'                // bell string
```

```
//****************************** Main Menu ******************************//

#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <graphics.h>
#include <stdlib.h>
#include <bios.h>
#include <alloc.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <process.h>
#include <mem.h>
#include "convert.cpp"
#include "terminal.cpp"

#define INTR 0x1C    /* The clock tick interrupt */
#ifdef __cplusplus
   #define __CPPARGS ...
#else
   #define __CPPARGS
#endif


void  more(),kdb(),load_f_mem(),setsystemcolor(void),
      view1(), view2(),
      save_screen( void far  *bf[4]),
      restore_screen( void far  *bf[4]),
      init_mouse(),
      iconu(int,int,int,int,char * ),
      icond(int,int,int,int,char * ),
      iconn(int,int,int,int,char * ),
      show_mouse(),
      click_mouse(),
      bckspc(int,int,int,int),
      interrupt ( *oldhandler)(__CPPARGS),
      hp();



int  ch1,dispflag=1,key=0,      //default terse mode
     gd=9,gm=2,rtrn,already=0,  //default no data as  not yet sendIreceive
     x,y,button=0,x2,y2,x3,y3,
     f=0,f1=0,userfont,
     pulldn(int x,int y,int cold,int cnew,int num,char *[]),
     key_function(),
     loadimage_to_screen(int x,int y,char filename[]);
char * getat(int x,int y,int max);
```

```
long int far maxx,maxy;

char *in,*c_om='2',*i_rq='0x3f8',*s_peed='9600';


char *sub4[4] = {"CONVERT TEXT FILE TO FAX FILE",
                 "VIEW FAX FILE                ",//"VIEW FAX FILE",
                 "OS SHELL                     ",//"PRINT FAX FILE",
                                "PREVIOUS MENU"},
     *sub5[3] = {"TERSE  MODE",
                 "EXPAND MODE",
                 "TERMINAL MODE"};


void  far *ptrb[4];

unsigned far size;

union REGS regs;


void interrupt handler1(__CPPARGS)
{
static int count=0,flag=0;
//setfillstyle(0,0);
//bar(490,380,510,400);
if(count>10){                //10:30
setcolor(BLACK);
switch(flag)
{
case 0:
outtextxy(500,390,"\\");
flag=1;
break;

case 1:
outtextxy(500,390,"|");
flag=2;
break;

case 2:
outtextxy(500,390,"-");
flag=3;
break;

case 3:
outtextxy(500,390,"/");
flag=4;
break;

case 4:
```

```
outtextxy(500,390,"-");
flag=0;
break;

default:
break;
}
delay(30);
count=0;
}

else count+=1;

/* call the old routine */
  oldhandler();
}



main()
  {
    regs.x.ax=0;
    int86(0x33,&regs,&regs);        /* reset mouse */
    if(regs.x.ax==0) {              /* check mouse install */
          f1=0;
          snd();clrscr();
          printf("mouse not found");
          delay(900);
          clrscr();
                    }
          else
           {
          f1=1;
         init_mouse();
           }


    initgraph(&gd,&gm,"");
//   setsystemcolor();
    loadimage_to_screen(50,0,"lo1.pcx");
    delay(1000);
    cleardevice();
    maxx = getmaxx();
    maxy = getmaxy();
//  userfont = installuserfont("litt.chr");
//  settextstyle(userfont,HORIZ_DIR,5);

    view1();
    view2();
    save_screen(ptrb);
    do {
```

```
                if(f1){
                show_mouse();
                click_mouse();
                }
            if((key=key_function())!=0) kdb();
            key=0;

    }while(button!=3);

    closegraph();
    system("mode co80");
    return 0;
    }
void kdb(void)
{
switch(key)
{
    case 0x3b:
                icond(5,455,80,479,"F1:SEND");
                iconn(350,150,600,200,"NOT COMPLETITION");
                delay(1600);
                restore_screen(ptrb);
                break;

    case 0x3c:
                icond(85,455,160,479,"F2:RECE");
                iconn(350,185,600,235,"NOT COMPLETITION");
                delay(1600);
                restore_screen(ptrb);
                break;

    case 0x3d:
                icond(165,455,240,479,"F3:EXAM");
    if(! already){ iconn(350,220,600,270,"    NO DATA ON MEMMORY");
                snd();}

      else; // srxy(,,,,);          //@@

                delay(1600);
                restore_screen(ptrb);
                break;

    case 0x3e:
                icond(245,455,320,479,"F4:DISP");
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                bar(385,270,500,320);
                setcolor(BLACK);
                moveto(385,270);
                lineto(385,320);
                lineto(500,320);
                lineto(500,270);
```

```
            lineto(385,270);
    rtrn=pulldn(390,280,BLUE,WHITE,3,sub5);
            switch(rtrn)
            {
            case 1:
            dispflag=1;
            break;

  case 2:
    dispflag=2;
    break;

  case 3:
    cleardevice();
    terminal();
    break;

            default:
            break;
            }

            restore_screen(ptrb);
            break;
  case 0x3f:
            icond(325,455,400,479,"F5:SETU");
    setfillstyle(SOLID_FILL,LIGHTGRAY);
    bar(400,310,635,380);
    setcolor(BLACK);
    moveto(400,310);
    lineto(400,380);
    lineto(635,380);
    lineto(635,310);
    lineto(400,310);
    setcolor(BLUE);
    outtextxy(405,315,"    Serial Port Setup");
    setcolor(BLACK);
    outtextxy(405,330,"COM#");                  //comport
          setfillstyle(SOLID_FILL,WHITE);
   bar(450,325,466,340);
   outtextxy(500,330,"IRQ#");                   //irq
   setfillstyle(SOLID_FILL,WHITE);
   bar(550,325,590,340);
   outtextxy(405,360,"SPEED");                  //speed
          setfillstyle(SOLID_FILL,WHITE);
   bar(450,355,490,370);
   c_om=getat(450,330,2);
   i_rq=getat(550,330,5);
   s_peed=getat(450,360,5);
          restore_screen(ptrb);
          break;
```

```
case 0x40:
        icond(405,455,480,479,"F6:FILE");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        bar(350,325,630,400);
        setcolor(BLACK);
        moveto(350,325);
        lineto(350,400);
        lineto(630,400);
        lineto(630,325);
        lineto(350,325);
        rtrn=pulldn(360,330,BLUE,WHITE,4,sub4);
        switch(rtrn)
        {
        case 1:
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        bar(400,350,635,420);
        setcolor(BLACK);
        moveto(400,350);
        lineto(400,420);
        lineto(635,420);
        lineto(635,350);
        lineto(400,350);
        outtextxy(405,370,"INPUT FILE NAME");
        setfillstyle(SOLID_FILL,WHITE);
        bar(535,365,633,380);
        in=getat(540,370,12);
        con(in);
        break;

        default:
        break;
        }
        restore_screen(ptrb);
        break;

case 0x41:
        icond(485,455,560,479,"F7:HELP");
        iconn(350,360,600,410,"NOT COMPLETITION");
    hp();
        restore_screen(ptrb);
        break;

case 0x42:
        icond(565,455,635,479,"F8:QUIT");
        delay(500);
        button=3;
        break;

default:
        break;
}
```

```
}

char * getat(int x,int y,int max)
{
        char c,c1[2],*cc="";
    int col=0,lm=x;
        do{
        c=getch();
        cc[col]=c;c1[0]=c;c1[1]='\0';
        cc[col+1]='\0';
        if(c=='\r'){cc[col]='\0';break;}

    if(((int)(c&0xFF)==0x8)&&(col != 0)){
    bckspc(x+8*(col-1),y,x+8*(col-1)+7,y+8);
    col-=1;
    lm-=8;

        }
        else

        {
    outtextxy(lm,y,c1);
        col+=1;
    lm+=8;
        }
    }while(col<max);

        return(cc);
}

void    bckspc(int left,int top,int right,int bottom)
{
int i,j,k=15;
for(i=left;i<=right;i++)
  for(j=top;j<=bottom;j++)
    putpixel(i,j,k);
}



void view1()
  {
    setbkcolor(BLACK);
    setcolor(WHITE);
    rectangle(1,1,638,478);
    rectangle(2,2,637,477);
    rectangle(7,7,632,92);
    rectangle(8,8,631,91);
    rectangle(9,9,630,90);
    setfillstyle(SOLID_FILL,WHITE);
```

```
    floodfill(3,3,WHITE);
    setfillstyle(SOLID_FILL,CYAN);
    floodfill(10,10,WHITE);
    setcolor(DARKGRAY);
    moveto(7,7);
    lineto(7,92);
    lineto(632,92);
    lineto(632,7);
    lineto(7,7);
    moveto(8,8);
    lineto(8,91);
    lineto(631,91);
    lineto(631,8);
    lineto(8,8);
    settextstyle(0,0,1);
    setcolor(BLACK);
    outtextxy(170,37,"FACSIMILE  PROTOCOL ANALYZER (G.3) ");
    outtextxy(170,59,"TELECOMMUNICATION ENGINEERING KMITL.");

  }

void view2()
  {
    setviewport(0,0,639,479,0);
    setcolor(BROWN);
    iconu(200,150,400,180,"SEND PROTOCOL VIEW    ");
    iconu(200,185,400,215,"RECEIVE PROTOCOL VIEW ");
    iconu(200,220,400,250,"EXAMINE               ");
    iconu(200,255,400,285,"DISPLAY  MODE         ");
    iconu(200,290,400,320,"TERMINAL SETUP        ");
    iconu(200,325,400,355,"FILE MANAGEMENT       ");
    iconu(200,360,400,390,"HELP                  ");
    iconu(200,395,400,425,"QUIT                  ");
    iconu(5,455,80,479,"F1:SEND");
    iconu(85,455,160,479,"F2:RECE");
    iconu(165,455,240,479,"F3:EXAM");
    iconu(245,455,320,479,"F4:DISP");
    iconu(325,455,400,479,"F5:SETU");
    iconu(405,455,480,479,"F6:FILE");
    iconu(485,455,560,479,"F7:HELP");
    iconu(565,455,635,479,"F8:QUIT");
  }
int pulldn(int x,int y,int co,int cn,int num,char *lst[])
  {
   int cnt=0,ln=y;
   setcolor(cn);
   do{
    outtextxy(x,ln,lst[cnt]);
    setcolor(co);
    cnt+=1;ln+=20;
   }while(cnt<num);
   cnt=0,ln=y;
```

```
do{
 key=bioskey(0);
 if(key==0x4800)
 {
 if(ln==y){
 setcolor(co);
 outtextxy(x,ln,lst[0]);
 ln+=20*num-20;
 setcolor(cn);
 outtextxy(x,ln,lst[num-1]);
                }
 else {
 setcolor(co);
 outtextxy(x,ln,lst[(ln-y)/20]);
 setcolor(cn);
 ln-=20;
 outtextxy(x,ln,lst[(ln-y)/20]);
                }
 }
 if(key==0x5000)
 {
 if(ln==y+20*num-20){
 setcolor(co);
 outtextxy(x,ln,lst[num-1]);
 ln=y;
 setcolor(cn);
 outtextxy(x,ln,lst[0]);
                }
 else {
 setcolor(co);
 outtextxy(x,ln,lst[(ln-y)/20]);
 setcolor(cn);
 ln+=20;
 outtextxy(x,ln,lst[(ln-y)/20]);
                }
 }
 if(key==0x1C0D)return((ln-y+20)/20);

 } while(1);
 }


void iconu(int x1,int y1,int x2,int y2,char ch[20])
  {
   setfillstyle(SOLID_FILL,LIGHTGRAY);
   setlinestyle(0,0,1);
   bar(x1,y1,x2,y2);
   setcolor(DARKGRAY);
   moveto(x1,y1);
   lineto(x1,y2);
   lineto(x2,y2);
   lineto(x2,y1);
```

```
    lineto(x1,y1);
    setcolor(8);
    setlinestyle(0,0,3);
    line(x2-4,y1+4,x2-4,y2-4);
    line(x1+4,y2-4,x2-4,y2-4);
    setcolor(WHITE);
    setlinestyle(0,0,1);
    line(x1+4,y1+4,x2-5,y1+4);
    line(x1+4,y1+4,x1+4,y2-5);
    setcolor(BLUE);
    settextstyle(0,0,1);
    outtextxy(x1+12,y1+10,ch);
   }

void iconn(int x1,int y1,int x2,int y2,char ch[20])/* Function icon select */
   {
    setfillstyle(SOLID_FILL,DARKGRAY);
    bar(x1+4,y1+4,x2+4,y2+4);
    setfillstyle(SOLID_FILL,LIGHTGRAY);
    bar(x1,y1,x2,y2);
    setcolor(YELLOW);
    settextstyle(0,0,1);
    outtextxy(x1+15,y1+11,ch);
   }

void icond(int x1,int y1,int x2,int y2,char ch[20])/* Function icon select */
   {
    setfillstyle(SOLID_FILL,LIGHTGRAY);
    bar(x1,y1,x2,y2);
    setcolor(DARKGRAY);
    moveto(x1,y1);
    lineto(x1,y2);
    lineto(x2,y2);
    lineto(x2,y1);
    lineto(x1,y1);
    setcolor(DARKGRAY);
    setlinestyle(0,0,1);
    line(x1+4,y1+4,x2-4,y1+4);
    line(x1+4,y1+4,x1+4,y2-4);
    line(x1+4,y2-4,x2-4,y2-4);
    line(x2-4,y1+4,x2-4,y2-4);
    setcolor(DARKGRAY);
    setlinestyle(0,0,2);
    line(x1+5,y1+5,x2-5,y1+5);
    line(x1+5,y1+5,x1+5,y2-5);
    setcolor(WHITE);
    setlinestyle(0,0,1);
    line(x1+5,y2-5,x2-5,y2-5);
    line(x2-5,y2-5,x2-5,y1+5);
    setcolor(BLUE);
    settextstyle(0,0,1);
    outtextxy(x1+13,y1+10,ch);
```

```
}

void init_mouse()                    /* Function initial mouse */
  {
    regs.x.ax=7;
    regs.x.cx=3;
    regs.x.dx=635;
    int86(0x33,&regs,&regs);        /* set min/max Hor. mouse position */
    regs.x.ax=8;
    regs.x.cx=3;
    regs.x.dx=475;
    int86(0x33,&regs,&regs);        /* set min/max Ver. mouse position */
    show_mouse();                   /* mouse show */
  }

void show_mouse()                    /* Function show mouse */
  {
    regs.x.ax=1;
    int86(0x33,&regs,&regs);
  }

hide_mouse()                         /* Funtion hide mouse */
  {
    regs.x.ax=2;
    int86(0x33,&regs,&regs);
    return(0);
  }

void click_mouse()      /* Function recieve data when mouse to click */
  {
    regs.x.ax=3;
    int86(0x33,&regs,&regs);
    button=regs.x.bx&3;
    x=regs.x.cx;
    y=regs.x.dx;
    regs.h.ah=0x0c;
    int86(0x21,&regs,&regs);


if(x<=400 && x>=200 && y<=180 && y>=150){
// ||(x>=5 && y>=455 && x<=80 && y<=479)){
                if(button&1) {
                f=1;
        already=1;
                        hide_mouse();
                        icond(200,150,400,180,"SEND PROTOCOL VIEW    ");
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                bar(385,270,500,320);
                setcolor(BLACK);
                moveto(385,270);
                lineto(385,320);
                lineto(500,320);
```

```
                lineto(500,270);
                lineto(385,270);
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                bar(400,350,635,420);
                setcolor(BLACK);
                moveto(400,350);
                lineto(400,420);
                lineto(635,420);
                lineto(635,350);
                lineto(400,350);
                outtextxy(405,370,"INPUT FILE NAME");
                setfillstyle(SOLID_FILL,WHITE);
                bar(535,365,610,380);
//          in=getat(540,370,12);

                }
                if(button&2) {
                        hide_mouse();
                        view2();
                }
        }
    if(x<=400 && x>=200 && y<=215 && y>=185) {
                if(button&1) {
                f=2;
        already=1;
                        hide_mouse();
                        icond(200,185,400,215,"RECIEVE PROTOCOL VIEW ");
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                bar(385,270,500,320);
                setcolor(BLACK);
                moveto(385,270);
                lineto(385,320);
                lineto(500,320);
                lineto(500,270);
                lineto(385,270);
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                bar(400,350,635,420);
                setcolor(BLACK);
                moveto(400,350);
                lineto(400,420);
                lineto(635,420);
                lineto(635,350);
                lineto(400,350);
                outtextxy(405,370,"INPUT FILE NAME");
                setfillstyle(SOLID_FILL,WHITE);
                bar(535,365,610,380);
//          in=getat(540,370,12);

                }
                if(button&2) {
                        hide_mouse();
                        view2();
```

```
            }
        }
if(x<=400 && x>=200 && y<=250 && y>=220) {
            if(button&1) {
            f=3;
                    hide_mouse();
                    icond(200,220,400,250,"EXAMINE              ");
if(! already){ iconn(350,220,600,270,"    NO DATA ON MEMMORY");
             snd();}

  else; // srxy(....);


            }
            if(button&2) {
                    hide_mouse();
                    view2();
            }
        }
if(x<=400 && x>=200 && y<=285 && y>=255) {
            if(button&1) {
            f=4;
                    hide_mouse();
                    icond(200,255,400,285,"DISPLAY  MODE        ");
            setfillstyle(SOLID_FILL,LIGHTGRAY);
            bar(385,270,500,320);
            setcolor(BLACK);
            moveto(385,270);
            lineto(385,320);
            lineto(500,320);
            lineto(500,270);
            lineto(385,270);
  rtrn=puildn(390,280,BLUE,WHITE,3,sub5);
            switch(rtrn)
            {
            case 1:
            dispflag=1;
            break;

            case 2:
            dispflag=2;
            break;

  case 3:
  hide_mouse();
  cleardevice();
  terminal();

  break;
            default:
```

```
                break;
                }
                }

                if(button&2) {
                        hide_mouse();
                        view2();
                  }
               }
if(x<=400 && x>=200 && y<=320 && y>=290) {
                if(button&1) {
                f=5;
              hide_mouse();
                        icond(200,290,400,320,"TERMINAL SETUP       ");
        setfillstyle(SOLID_FILL,LIGHTGRAY);
        bar(400,310,635,380);
        setcolor(BLACK);
        moveto(400,310);
        lineto(400,380);
        lineto(635,380);
        lineto(635,310);
        lineto(400,310);
        setcolor(BLUE);
        outtextxy(405,315,"     Serial Port Setup");
        setcolor(BLACK);
        outtextxy(405,330,"COM#");                  //comport
             setfillstyle(SOLID_FILL,WHITE);
        bar(450,325,466,340);
        outtextxy(500,330,"IRQ#");                  //irq
        setfillstyle(SOLID_FILL,WHITE);
        bar(550,325,590,340);
        outtextxy(405,360,"SPEED");                 //speed
             setfillstyle(SOLID_FILL,WHITE);
        bar(450,355,490,370);
        c_om=getat(450,330,2);
        i_rq=getat(550,330,5);
        s_peed=getat(450,360,5);
                }
                if(button&2) {
                        hide_mouse();
                        view2();
                  }
               }
if(x<=400 && x>=200 && y<=355 && y>=325) {
                if(button&1)
                {
                f=6;
                hide_mouse();
                icond(200,325,400,355,"FILE MANAGEMENT      ");
                setfillstyle(SOLID_FILL,LIGHTGRAY);
        bar(350,325,630,400);
                setcolor(BLACK);
```

```
                moveto(350,325);
                lineto(350,400);
                lineto(630,400);
                lineto(630,325);
                lineto(350,325);
        rtrn=pulldn(360,330,BLUE,WHITE,4,sub4);
                switch(rtrn)
                {
                case 1:
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                bar(400,350,635,420);
                setcolor(BLACK);
                moveto(400,350);
                lineto(400,420);
                lineto(635,420);
                lineto(635,350);
                lineto(400,350);
                outtextxy(405,370,"INPUT FILE NAME");
                setfillstyle(SOLID_FILL,WHITE);
                bar(535,365,610,380);
                in=getat(540,370,12);

/* save the old interrupt vector */
   oldhandler = getvect(INTR);
/* install the new interrupt handler1 */
   setvect(INTR, handler1);

        con(in);                //...call converted module

/* reset the old interrupt handler */
   setvect(INTR, oldhandler);

        break;

                case 2:
                snd();
                break;

                case 3:
        cleardevice();
        closegraph();
        system("mode co80");
        printf("\n\n G.3 Fax Protocol\n\n'EXIT'  to return program\n");
        system("command");
        initgraph(&gd,&gm,"");
                break;

                default:
                break;
                }
                }
                if(button&2) {
```

```
                hide_mouse();
                        view2();
                    }
                }
    if(x<=400 && x>=200 && y<=390 && y>=360) {
                    if(button&1) {
                    f=7;
                hide_mouse();
                            icond(200,360,400,390,"HELP                ");
                hp();
    //          iconn(350,360,600,410,"NOT COMPLETITION");
                    }
                    if(button&2) {
                            hide_mouse();
                            view2();
                    }
                }
    if(x<=400 && x>=200 && y<=425 && y>=395) {
                    if(button&1) {
                    f=8;
                            hide_mouse();
                            icond(200,395,400,425,"QUIT                ");
                            button=3;
                    }
                    if(button&2) {
                            hide_mouse();
                            view2();
                    }
                }
    switch(f)
    {
    case 1:
    if(x>=400 || x<=200 || y>=180 || y<=150){
     restore_screen(ptrb);          /* restore the screen */
     f=0;
     }
     break;


     case 2:
     if(x>=400 || x<=200 || y>=215 || y<=185){
     restore_screen(ptrb);
     f=0;
     }
     break;

     case 3:
     if(x>=400 || x<=200 || y>=250 || y<=220){
     restore_screen(ptrb);
     f=0;
     }
     break;
```

```
        case 4:
        if(x>=400 || x<=200 || y>=285 || y<=255){
        restore_screen(ptrb);
        f=0;
        }
        break;

        case 5:
        if(x>=400 || x<=200 || y>=320 || y<=290){
        restore_screen(ptrb);
        f=0;
        }
        break;

        case 6:
//      if(x>=400 || x<=200 || y>=355 || y<=325){
        restore_screen(ptrb);
        f=0;
//      }
        break;

        case 7:
//      if(x>=400 || x<=200 || y>=390 || y<=360){
        restore_screen(ptrb);
        f=0;
//      }
        break;

        case 8:
//      if(x>=400 || x<=200 || y>=425 || y<=395){
        restore_screen(ptrb);
        f=0;
        break;

        default:
        break;

        }

    }


void restore_screen(void far *bf[4])
{
  int ystart=0, yend, yincr, block;

  yincr =(int)(maxy+1) / 4;
  yend = yincr;

  for (block=0; block<=3; block++)
  {
```

```
        putimage(0, ystart, bf[block], COPY_PUT);
//      farfree(bf[block]);
        ystart = yend + 1;
        yend += yincr + 1;
    }
}
void save_screen(void far  *bf[4])
{
    unsigned size;
    int ystart=0, yend, yincr, block;

    yincr =(int)(maxy+1) / 4;
    yend = yincr;
    size = imagesize(0, ystart,(int)maxx, yend);
/* get byte size of image */

    for (block=0; block<=3; block++)
    {
      if ((bf[block] = farmalloc(size)) == NULL)
      {
              closegraph();
              printf("Error: not enough heap space in save_screen().\n");
              exit(1);
      }

      getimage(0, ystart,(int)maxx, yend, bf[block]);
      ystart = yend + 1;
      yend += yincr + 1;
    }
}

void load_f_mem()
{
char mem[1024];
char txt[20];
int r=0,i1=0,i2=0,row=50,col=10;
FILE *f_m;
if((f_m=fopen("help.txt","rt")) == NULL){exit(0);}
while(! feof(f_m)){
*(mem+r)=getc(f_m);
fseek(f_m,(long int)r,0);
r+=1;
}
cleardevice();
setcolor(WHITE);

while(i1<r){
if(*(mem+i1)==0x0A){
txt[i2]='\0';i1+=2;row+=10;col=10;

if(i2>1)outtextxy(col,row,txt);
```

```
i2=0;}

txt[i2]=*(mem+i1);
i2+=1;
i1+=1;
}
delay(1600);

}

int  key_function()
   {
     regs.h.ah=0x0c;
     int86(0x21,&regs,&regs);
     return(inport(0x60)&0xFF);
   }



void hp()
{
int   userfont,ki=0;
char *head="Protocol Detail",
     *arup="-",
     *ardn="",
     *stop="",
     *pgup="U",
     *pgdn="D";

char mem[17000];  //*mem;
char txt[80];
int ha=0,r=0,s1=0,s2=0,s3=0,i1=0,i2=0,row=30,col=10,cr_no=0,f_p[500],
   curr=0,dummy=0,curro=0,dumro=0;
FILE *f_m;



if((f_m=fopen("help.txt","rb")) == NULL){exit(0);}
while(! feof(f_m)){
*(mem+r)=getc(f_m);       //r:number of bytes
r+=1;
}
fclose(f_m);
cleardevice();
//iconn(250,5,400,30,head);
setfillstyle(SOLID_FILL,DARKGRAY);
bar(250+4,5+4,400+4,30+4);
setfillstyle(SOLID_FILL,YELLOW);
bar(250,5,400,30);
setcolor(BLUE);
```

```
settextstyle(0,0,1);
outtextxy(250+15,5+11,head);

setcolor(LIGHTGREEN);

txt[0]=0x20;
i2=1;
while(cr_no<41)              //i1:count char
{
if(*(mem+i1)=='\r'){
txt[i2]='\0';i1+=1;f_p[cr_no]=i1;row+=10;col=10;cr_no+=1;

for(s3=0;s3<2;s3++)
for(s1=0;s1 < 80;s1++){txt[s1]=txt[s1+1];}

if(i2>0){
outtextxy(col,row,txt);
 }

i2=0;
txt[0]=0x20;
i2=1;
}

txt[i2]=*(mem+i1);
i2+=1;
i1+=1;

}
curr=cr_no-1;curro=row;

while(i1<r)          //     ;check total cr
{
if(*(mem+i1)=='\r'){
i1+=1;f_p[cr_no]=i1;cr_no+=1;
}
i1+=1;
}
//
//iconu(610,425,639,449,arup);
iconu(5,451,34,475,pgup);
iconu(39,451,68,475,pgdn);
iconu(576,451,605,475,arup);
iconu(610,451,639,475,ardn);
iconu(610,1,639,25,stop);
show_mouse();
while(ki!=1)
{
    regs.x.ax=3;
    int86(0x33,&regs,&regs);
    button=regs.x.bx&3;
    x=regs.x.cx;
```

```
        y=regs.x.dx;
        regs.h.ah=0x0c;
        int86(0x21,&regs,&regs);

        ki=key_function();

if((x<=605 && x>=576 && y<=475 && y>=451) || (ki==0x48)){
            if((button&1)||(ki==0x48)) {            //arrow up
hide_mouse();
icond(576,451,605,475,arup);
if((curr-41)>=0)
{
dummy=curr-1,dumro=curro;
curr-=1;}

else {
delay(250);
iconu(576,451,605,475,arup);
show_mouse();snd();goto jump;}

while((ha<41)&&(i1<r+1)&&((curr-40)>=0))
{
i1=f_p(dummy);
i2=0;
setcolor(LIGHTGREEN);
while(*(mem+i1)!='\r')
{
txt[i2]=*(mem+i1);
i2+=1;i1+=1;
}
i1+=1;
txt[i2]='\0';
//for(s3=0;s3<2;s3++)
for(s1=0;s1 < 80;s1++){txt[s1]=txt[s1+1];}
setfillstyle(0,0);
bar(10,dumro-5,639,dumro+8);
if(i2>0){
outtextxy(col,dumro,txt);
 }
dumro-=10;col=10;dummy-=1;
ha+=1;
}
ha=0;
//iconu(610,451,639,475,ardn);
//iconu(610,425,639,449,arup);
iconu(576,451,605,475,arup);
iconu(610,451,639,475,ardn);
iconu(610,1,639,25,stop);
show_mouse();
ki=0;
                    }
```

```
            if(button&2) {
                snd();
                    }
                }
if((x<=639 && x>=610 && y<=475 && y>=451)||(ki==0x50)) {
            if((button&1)||(ki==0x50)) {            //arrow down
hide_mouse();
icond(610,451,639,475,ardn);
if( f_p[curr+1] < r-80)
{
dummy=curr+1;dumro=curro;
curr+=1;
}
else {
delay(250);
iconu(610,451,639,475,ardn);
show_mouse();snd();goto jump;}

while(ha<41)
{
i1=f_p[dummy]+1;
i2=0;
setcolor(LIGHTGREEN);
while(*(mem+i1)!='\r')
{
txt[i2]=*(mem+i1);
i2+=1;i1+=1;
}
i1+=1;
txt[i2]='\0';
//for(s3=0;s3<2;s3++)
//for(s1=0;s1 < 80;s1++){txt[s1]=txt[s1+1];}

setfillstyle(0,0);
bar(10,dumro-5,639,dumro+8);
if(i2>0){
outtextxy(col,dumro,txt);
 }
dumro-=10;col=10;dummy-=1;
ha+=1;
}
ha=0;
//iconu(610,451,639,475,ardn);
//iconu(610,425,639,449,arup);
iconu(576,451,605,475,arup);
iconu(610,451,639,475,ardn);
iconu(610,1,639,25,stop);
show_mouse();
ki=0;
                    }
                if(button&2) {
                snd();
```

```
                }
            }

if((x<=34 && x>=5 && y<=475 && y>=451) || (ki==0x49)){
        if((button&1)||(ki==0x49)) {        //page  up
hide_mouse();
icond(5,451,34,475,pgup);
if((curr-41)>=0)
{
dummy=curr-40,dumro=curro;
curr-=40;}

else {
delay(250);
iconu(5,451,34,475,pgup);
show_mouse();snd();goto jump;}

while((ha<41)&&(i1<r+1)&&((curr-40)>=0))
{
i1=f_p[dummy];
i2=0;
setcolor(LIGHTGREEN);
while(*(mem+i1)!='\r')
{
txt[i2]=*(mem+i1);
i2+=1;i1+=1;
}
i1+=1;
txt[i2]='\0';
//for(s3=0;s3<2;s3++)
for(s1=0;s1 < 80;s1++)(txt[s1]=txt[s1+1];}
setfillstyle(0,0);
bar(10,dumro-5,639,dumro+8);
if(i2>0){
outtextxy(col,dumro,txt);
}
dumro-=10;col=10;dummy-=1;
ha+=1;
}
ha=0;
//iconu(610,451,639,475,ardn);
//iconu(610,425,639,449,arup);
iconu(5,451,34,475,pgup);
iconu(39,451,68,475,pgdn);
iconu(576,451,605,475,arup);
iconu(610,451,639,475,ardn);
iconu(610,1,639,25,stop);
show_mouse();
ki=0;
                }
```

```
            if(button&2) {
                snd();
                    }
                }
 if((x<=68 && x>=39 && y<=475 && y>=451)||(ki==0x51)) {
            if((button&1)||(ki==0x51)) {          //page down
 hide_mouse();
 icond(39,451,68,475,pgdn);
 if( f_p[curr+40] < r-80)
 {
 dummy=curr+40;dumro=curro;
 curr+=40;
 }
 else {
 delay(250);
 iconu(39,451,68,475,pgdn);
 show_mouse();snd();goto jump;}

 while(ha<41)
 {
 i1=f_p[dummy]+1;
 i2=0;
 setcolor(LIGHTGREEN);
 while(*(mem+i1)!='\r')
 {
 txt[i2]=*(mem+i1);
 i2+=1;i1+=1;
 }
 i1+=1;
 txt[i2]='\0';
 //for(s3=0;s3<2;s3++)
 //for(s1=0;s1 < 80;s1++)(txt[s1]=txt[s1+1];}

 setfillstyle(0,0);
 bar(10,dumro-5,639,dumro+8);
 if(i2>0){
 outtextxy(col,dumro,txt);
 }
 dumro-=10;col=10;dummy-=1;
 ha+=1;
 }
 ha=0;
 //iconu(610,451,639,475,ardn);
 //iconu(610,425,639,449,arup);
 iconu(5,451,34,475,pgup);
 iconu(39,451,68,475,pgdn);
 iconu(576,451,605,475,arup);
 iconu(610,451,639,475,ardn);
 iconu(610,1,639,25,stop);
 show_mouse();
 ki=0;
            }
```

```
                if(button&2) {
                snd();
                }
        }

if((x<=639 && x>=610 && y<=25 && y>=1)||(ki==1)) {
                if(button&1) {
                hide_mouse();
                icond(610,1,639,25,stop);
                show_mouse();
                delay(300);
                ki=1;
                }
                if(button&2) {
                snd();
                }
                }

jump:
}

  /* free memory */
//   free(mem);

hide_mouse();
}
/*********************************************************************/
void setsystemcolor(void)
 {
 char rgb[48] = {
0, 0, 0, 42,0,0,0,42,0,42,42,0,0,0,42,42,0,42,0,42,42,21,21,21,
42,42,42,63,0,0,0,63,0,63,63,0,0,0,63,63,0,63,0,63,63,63,63,63
  };
 char color_id[16]={0,1,2,3,4,5,20,7,56,57,58,59,60,61,62,63};
 int i;
 for(i=0;i<16;i++) setrgbpalette(color_id[i],rgb[i*3],rgb[i*3+1],rgb[i*3+2]);
 }
/*********************************************************************/
/*********************************************************************/
int loadimage_to_screen(int x,int y,char filename[])
 {
 FILE *fp=NULL;
 char *scanline=NULL,*temp,*ptr;
 int c,i,j,sx,bpl,bpl4;
 struct {
                char manufacturer,version,encoding,bits_per_pixel;
                int xmin,ymin,xmax,ymax,hres,vres;
                char palette[48],reserved,color_planes;
                int bytes_per_line,palette_type;
                char filler[58];
                } pcxhead;
```

```
if((fp = fopen(filename,"rb")) == NULL) {
  printf("Can not open file %s%c",filename,7);  getch();  return(0);
  }
fread(&pcxhead,sizeof(pcxhead),1,fp);
if(pcxhead.manufacturer != 10 || pcxhead.bits_per_pixel > 4) {
  fclose(fp);  printf("Need PCX 16 color only!%c",7);  getch();  return(0);
  }

if((scanline = (char *)malloc(4+400+400)) == NULL) {
  fclose(fp);printf("Can not allocate memory!%c",7); getch(); return(NULL);
  }

temp = scanline+4+400;
i = pcxhead.xmax;
*(scanline+0) = i%256;
*(scanline+1) = i/256;
*(scanline+2) = 0;
*(scanline+3) = 0;

j = 0;
ptr = temp;
bpl = pcxhead.bytes_per_line;
bpl4 = bpl*4;
if((pcxhead.xmax+1)%8) sx = (pcxhead.xmax+1)/8+1;
else sx = (pcxhead.xmax+1)/8;
while(1) {
  if((c = getc(fp)) == EOF) break;
  if(c > 0xc0) { i = c&0x3f;  c = getc(fp);}
  else i = 1;
  while(i-- > 0) {
    if(j++ == bpl4) {
            movmem(temp,scanline+sx*3+4,sx);
            movmem(temp+bpl,scanline+sx*2+4,sx);
            movmem(temp+bpl*2,scanline+sx+4,sx);
            movmem(temp+bpl*3,scanline+4,sx);
            putimage(x,y++,scanline,0);
            j = 1;
            ptr = temp;
            }
    *ptr++ = c;
    }
  }
if(j == bpl4) {
  movmem(temp,scanline+sx*3+4,sx);
  movmem(temp+bpl,scanline+sx*2+4,sx);
  movmem(temp+bpl*2,scanline+sx+4,sx);
  movmem(temp+bpl*3,scanline+4,sx);
  putimage(x,y++,scanline,0);
  }

fclose(fp);
free(scanline);
```

```
return(1);
}
```

```
/************************** snd(viod) ************************************//

 #include <stdlib.h>
void  snd(void)
{
   long  i,freq;
   int ii;
   for(ii=600;ii<20000;ii++) sound(ii);

   for(i=2000; i> 10; i-=8){
      freq=random(i);
      sound(freq);
   }
   nosound();
}
```

```
//****************** send & receive module ***************/

#include <stdio.h>                    // standard i/o library
#include <stdarg.h>                   // variable argument list
#include <string.h>                   // string handling routines
#include <stdlib.h>                   // std conversion routines
#include <dos.h>                      // dos functions
#include <ctype.h>                    // character routines
#include <conio.h>                    // console functions
#include <bios.h>                     // bios functions
#include <io.h>                       // i/o functions
#include <dir.h>                      // directory routines
#include <sys\stat.h>                 // ..attribute bits
#include <fcntl.h>                    // ..and file control
#include <graphics.h>
#include "keys.h"                     // keyboard definitions
#include "snd.cpp"


#define TICKS_DAY 1573040L            // number of ticks per day
#define ALLOW_ALT   1                 // allow alt key in get_key()
#define NO_ALT      0                 // ..and supress alt key
#define FAX_CLASS1  "AT +FCLASS=1\r"      // enter FAX class 1 mode
#define FAX_TX_HDLC "AT +FTH=3\r"         // enter HDLC transmit mode
#define FAX_RX_HDLC "AT +FRH=3\r"         // enter HDLC recieve mode
#define FAX_TX_DATA "AT +FTM=%d\r"        // enter FAX transmit mode
#define FAX_RX_DATA "AT +FRM=%d\r"        // enter FAX receive mode
#define FAX_TX_SPD  "AT +FTM=?\r"         // get FAX transmit speeds
#define FAX_RX_SPD  "AT +FRM=?\r"         // get FAX receive speeds
#define FAX_SILENT  "AT +FRS=8\r"         // 80 ms of silence
#define FAX_SILENT1 "AT +FRS=20\r"        // 200 ms of silence
#define FAX_CLASS1  "AT +FCLASS=1\r"      // enter FAX class 1 mode
#define FAX_MODEM   "AT +FCLASS=0\r"      // return to non-FAX mode
#define FAX_ANSWER  "AT A\r"              // answer an incoming call
#define FAX_HANGUP  "AT H\r"              // disconnect from line
#define FAX_DIAL    "ATDT"                // dial command
#define FAX_OK      "\r\nOK\r\n"          // OK response
#define FAX_ERR     "\r\nERROR\r\n"       // ERROR response
#define FAX_NO_CARR "\r\nNO CARRIER\r\n"  // NO CARRIER response
#define FAX_CONN    "\r\nCONNECT\r\n"     // CONNECT response
#define FAX_RING    "\r\nRING\r\n"        // RING message
#define FAX_SETMDM  "AT Q0 V1 E0\r"       // set modem parameters
#define FAX_RSTMDM  "AT Z\r"              // reset modem
#define FAX_ADDR    0xff                  // value for address byte
#define FAX_CTL     0xC0                  // value for control field
#define FAX_CTL_FF  0xC8                  // control field + final frame
#define FAX_FCF_DIS 0x01                  // digital ID signal
#define FAX_FCF_CSI 0x02                  // called subscriber ID
#define FAX_FCF_NSF 0x04                  // non-standard facilities
#define FAX_FCF_DCS 0xC1                  // digital command signal
#define FAX_FCF_TSI 0xC2                  // transmitting subscriber ID
#define FAX_FCF_CFR 0x21                  // confirmation to receive
#define FAX_FCF_FTT 0x22                  // failure to train
```

```
#define FAX_FCF_EOM 0xF1            // end of message
#define FAX_FCF_MPS 0xF2            // multipage signal
#define FAX_FCF_EOP 0xF4            // end of procedure
#define FAX_FCF_MCF 0x31            // message confirmation
#define FAX_FCF_DCN 0xDF            // disconnect
#define FINAL       8               // final frame
#define NON_FINAL   0               // non-final frame
#define TRANSMIT    0               // connect in send mode
#define RECEIVE     1               // connect in receive mode
#define FAX_T9600   0x80            // 9600 transmit
#define FAX_T7200   0x40            // 7200 transmit
#define FAX_T4800   0x20            // 4800 transmit
#define FAX_T2400   0x10            // 2400 transmit
#define FAX_R9600   0x08            // 9600 receive
#define FAX_R7200   0x04            // 7200 receive
#define FAX_R4800   0x02            // 4800 receive
#define FAX_R2400   0x01            // 2400 receive
#define CURSOR()    _setcursortype(_NORMALCURSOR)   // normal text cursor
#define BIGCURSOR() _setcursortype(_SOLIDCURSOR)    // insert mode cursor
#define NOCURSOR()  _setcursortype(_NOCURSOR)       // turn off cursor
#define COUNT(x)    (sizeof(x) / sizeof(x[0]))      // item count
#define NOT         !               // shorthand logical
#define BYTE        char            // single byte
#define UINT        unsigned int    // unsigned integer
#define UCHAR       unsigned char   // ..and unsigned character
#define ULONG       unsigned long   // ..and unsigned long
#define MAX_PATH    79              // maximum path length
#define MIX(x,y)    ((x << 4) + (y))    // mix colors for fg and bg
#define FG(x)       (unsigned char) x >> 4  // extract foreground color
#define BG(x)       x & 0x07        // ..and background color
#define IN(x)       inportb(base + x)   // read a UART register
#define OUT(x,y)    outportb(base + x, y)   // ..and write a register
#define NULLPTR(x)  &x ? x : ''     // make null ptr point to null
#define LAST(s)     s[strlen(s) - 1]    // last character in string
#define SECS(x)     (long) (x * 182L) / 10L // seconds to ticks conversion
#define TRUE        1               // true value
#define FALSE       0               // false value
#define SOH         1               // start of header
#define STX         2               // start of text
#define ETX         3               // end of text
#define EOT         4               // end of transmission
#define ACK         6               // positive acknowledgment
#define XOFF        19              // flow control X-OFF
#define DLE         16              // data link escape
#define NAK         21              // negative acknowledgment
#define CAN         24              // cancel process
/* ************************************************************** *
 * UART Register Definitions
 * ************************************************************** */
                                    // UART regs (base address +)
#define RBR         0               // receive buffer register
#define THR         0               // transmit holding register
```

```
#define DLL     0               // divisor latch LSB
#define DLM     1               // divisor latch MSB
#define IER     1               // interrupt enable register
#define IIR     2               // interrupt id register
#define FCR     2               // FIFO control register
#define AFR     2               // alternate function register
#define LCR     3               // line control register
#define MCR     4               // modem control register
#define LSR     5               // line status register
#define MSR     6               // modem status register
#define SCR     7               // scratch register

                                // interrupt enable register
#define IER_RBF   0x01          // receive buffer full
#define IER_TBE   0x02          // transmit buffer empty
#define IER_LSI   0x04          // line status interrupt
#define IER_MSI   0x08          // modem status interrupt
#define IER_ALL   0x0f          // enable all interrupts

                                // interrupt id register
#define IIR_PEND  0x01          // interrupt pending = 0
#define IIR_II    0x06          // interrupt id bits
                                // 000 = modem status change
                                // 001 = trans holding empty
                                // 010 = receive buffer full
                                // 110 = receive fifo full
                                // 011 = line status change
#define IIR_MSI   0x00          // modem status interrupt
#define IIR_TBE   0x02          // transmit buffer empty
#define IIR_RBF   0x04          // receive buffer full
#define IIR_LSI   0x06          // line status interrupt
#define IIR_RFF   0x0c          // receive fifo threshold
                                // fifo control register
#define FCR_FIFO  0x01          // fifo enable
#define FCR_RCVR  0x02          // receiver fifo reset
#define FCR_XMIT  0x04          // transmit fifo reset
#define FCR_DMA   0x08          // DMA mode select
#define FCR_TRIGGER 0xc0        // receiver trigger select
                                // 00 = 1 byte
                                // 01 = 4 bytes
                                // 10 = 8 bytes
                                // 11 = 14 bytes
#define FCR_16550 0xc7          // 16550 fifo enable/reset
                                // line control register
#define LCR_WLEN  0x03          // word length
                                // 10 = 7 bits
                                // 11 = 8 bits
#define LCR_STOP  0x04          // stop bits
                                // 0 = 1 stop bit
                                // 1 = 2 stop bits
#define LCR_PARITY 0x08         // parity enable
                                // 0 = no parity
                                // 1 = send/check parity
#define LCR_EVEN  0x10          // even/odd parity
```

```
                                    //  0 = odd parity
                                    //  1 = even parity
#define LCR_BREAK   0x40            // break, set to xmit break
#define LCR_DLAB    0x80            // divisor latch access bit
                                                        // modem control register
#define MCR_DTR     0x01            // DTR control
#define MCR_RTS     0x02            // RTS control
#define MCR_OUT2    0x08            // OUT2 control
#define MCR_DO      0x0b            // dtr, rts & out2 enabled
                                                        // line status register
#define LSR_DR      0x01            // data ready
#define LSR_ORUN    0x02            // overrun error
#define LSR_PRTY    0x04            // parity error
#define LSR_FRM     0x08            // framing error
#define LSR_BRK     0x10            // break interrupt
#define LSR_THRE    0x20            // transmit holding reg empty
#define LSR_TSRE    0x40            // transmit shift reg empty
#define LSR_ERROR   0x1e            // error conditions
                                                        // modem status register
#define MSR_DCTS    0x01            // delta clear to send
#define MSR_DDSR    0x02            // delta data set ready
#define MSR_TERI    0x04            // trailing edge ring indicator
#define MSR_DCD     0x08            // delta carrier detect
#define MSR_CTS     0x10            // clear to send
#define MSR_DSR     0x20            // data set ready (modem ready)
#define MSR_RI      0x40            // ring indicated
#define MSR_CD      0x80            // carrier detected
#define CMD_INIT    commands[0][1]      // commands[] index
#define CMD_DIAL    commands[1][1]
#define CMD_EXECUTE commands[2][1]
#define CMD_RESET   commands[3][1]
#define CMD_CONNECT commands[4][1]
#define CMD_NO_CONN commands[5][1]
#define CMD_OK      commands[6][1]
#define CMD_ERROR   commands[7][1]

/* ********************************************************************** *
 *  Messages and strings
 * ********************************************************************** */

char    about_msg[]     = " Press any key to continue .. ",
            unknown_cmd[]   = "Unknown command: %s\n",
            bad_setting[]   = "Bad %s= setting: %s\n",
            bad_port[]      = "Bad base port setting in COM%d: %s\n",
            bad_irq[]       = "Bad interrupt setting in COM%d: %s\n",
            bad_pc_cmd[]    = "Bad PC: command\n",
            bad_parm[]      = "Bad value in COMSETTING statement: %s\n",
            pb_overflow[]   = "Too many phonebook entries\n",
            pb_setting[]    = "Phonebook entry: %s \n  has a bad %s setting: %s\n",
            pb_format[]     = "%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",
        *fld_name[]  =  { "Name", "Telephone", "Fax Number",
                                    "Modem Setup", "Speed", "Parity",
                                    "Data Bits", "Stop Bits" },
```

```
        no_ports[]      = "No communication ports were found\n",
    *line_error[] = { "Overrun", "Parity", "Framing", "Break" },
        overflow_msg[] = "\r\nBuffer Overflow: %d chars lost\r\n",
        hangup_msg[]   = " Issuing hangup to modem\r\n",
        hangup_done[]  = " ..hangup completed",
    *dial_msg[]    = { "Setting communications port parameters",// 0
                                    "Sending reset command to modem",       // 1
                "..user terminated dialing sequence",   // 2
                                    "...timeout waiting for modem response", // 3
                                    "...reset completed properly",          // 4
                                    "...reset command rejected by modem",   // 5
                "Sending modem setup commands",         // 6
                                    "...modem setup completed successfully", // 7
                                    "...modem rejected setup string",       // 8
                                    "Sending phone number to modem",        // 9
                "...connection established",            // 10
                "..connection failed",                  // 11
                "Phone number entry blank"              // 12
                                    "...dial sequence terminated",
                "...",                                  // 13
                " command not found in .CFG file" },    // 14
    *endings[]     = { "\r", "\n" },
    *dl_msg[]      = { " Filename: ",                   // 0
                " Overwrite file? ",                    // 1
                                    "\r\n..user terminated download",       // 2
                                    "\r\n..too many errors",                // 3
                                    "\r\n..sender terminated download",     // 4
                                    "\r\n..unable to create output file",   // 5
                                    "\r %lu packets  %u errors",            // 6
                                    "\r\n %lu data bytes using XMODEM %s",  // 7
                                    "\r\n%d files transferred, %lu bytes.", // 8
                                    "\r\n%s using %s\r\n",                  // 9
                                    "\r\n..timeout error",                  // 10
                                    "\r\nTransferring %s: Len=%lu\r\n",     // 11
                                    "\r\n..unrecoverable sequence error",   // 12
                                    "\r %lu bytes left (%u errors)      ",  // 13
                                    "\r\n..user terminated upload",         // 14
                                    "\r\n..receiver terminated send",       // 15
                                    "Enter the names of files to send:\r\n" // 16
                                      "1:\r\n2:\r\n3:\r\n4:\r\n5:"
                                    },
    *fax_msgs[]    = { "%s Fax file %s",                // 0
                                    "Initializing modem",           // 1
                                    "Awaiting call",                // 2
                                    "Answering call",               // 3
                "Sending our ID",                       // 4
                "Sending (DIS)",                        // 5
                "(TSI) received",                       // 6
                "(DCS) received",                       // 7
                "(NSF) received",                       // 8
                "Receiving (TCF)",                      // 9
                "Confirming (TCF)",                     // 10
```

```
                "Receiving Fax page",           // 11
                "Page received",                // 12
                "(MPS) received",               // 13
                "(EOM) received",               // 14
                "(EOP) received",               // 15
                "Disconnect received",          // 16
                "Error!!",                      // 17
                "User cancelled FAX procedure", // 18
                "Retrying TCF receive",         // 19
                "Timeout awaiting response",    // 20
                "Unexpected response",          // 21
        "Calling remote FAX # ",                // 22
        "(CSI) received",               // 23
        "(DIS) received",               // 24
        "Sending (TSI)",                // 25
        "Sending (DCS)",                // 26
        "Sending (TCF)",                // 27
        "(TCF) confirmed",              // 28
                "Sending page",                 // 29
                "(TCF) failed ",                // 30
                "Page sent",                    // 31
                "Sending  (MPS)",               // 32
                "Sending  (EOP)",               // 33
                "Sending  (DCN)",               // 34
                "*** XOFF received ***",        // 35
                "                  ",           // 36
                "Modem did not respond to reset", // 37
                "Modem reset",                  // 38
                "Modem reset error",            // 39
        " Filename: ",                  // 40
                "Attempting 9600bps",           // 41
                "Attempting 7200bps",           // 42
                "Attempting 4800bps",           // 43
                "Attempting 2400bps"            // 44
                };
char *fax_errors[] = { "No error\r\n",          // 0
                "Modem initialization error",   // 1
                "Modem did not connect",        // 2
                "Error receiving HDLC frame",   // 3
        "No (DCS) received",            // 4
                "Unsupported speed requested",  // 5
        "Invalid (TCF) format",         // 6
        "(TCF) improperly terminated",  // 7
        "Improper (TCF) length",        // 8
                "Unable to open fax file",      // 9
                "Connection failed... try again later.", // 10
        "No (DIS) received",            // 11
                "Unable to establish session",  // 12
                "Invalid file format",          // 13
                "Expected confirmation not received",   // 14
        "Timeout sending FAX data"      // 15
                };
```

```
        no_memory[]    = "\nUnable to allocate %d bytes of memory\n",
        open_error[]   = "\nError opening file (%s)\n",
        read_error[]   = "\nError reading file (%s)\n",
        write_error[]  = "\nError writing file (%s)\n",
        stop_here[]    = "\nStopping at user's request\n",
        answer_yes[]   = "Yes\r",
        answer_no[]    = "No\r",
        cfg_extension[]= "*.CFG",
        pb_file[]      = "POLYCOMM.PB",
        delimit_1[]    = "= \t;\n",
        delimit_2[]    = ", \t;\n",
        delimit_3[]    = ";\n",
        delimit_4[]    = " ",
        delimit_eol[]  = "\n",
        beep[]         = "\a",
        help[]         =
            " Usage:  PolyComm  config\n\n"
        " Where:  config  is the name of the configuration file\n";


struct copies
{
char msg[256];
int len;
} sndrcve[20][4];


static int      terse_0=0,terse_1=0,exp_0=0,exp_1=0;

/*****************************************************************
 *    function define
 *****************************************************************/
int    wait_for(char *,char *,long ),
       time_out(long *),              // time to wait in ticks
       get_key(int );                 // nonzero = allow alt_key

ULONG  get_time(void),                // retrieve time in ticks
           elapsed_time(ULONG );      // start time in ticks


void   wait_ms(long ),                // milliseconds to wait
           *malloc_chk(int ),         // size to allocate
       checkerrors(void),
       wait(long ),                   // time to wait in ticks
       puthex(int ,int ,char ch[1]);
/* ***************************************************************** *
 *    constant
 * ***************************************************************** */
int base=0x2f8;


char  *commands[8][2] =               // command strings
                {
                { "INIT",    "ATZ"    }, // initialization command
```

```
    { "DIAL",       "ATD"       }, // cmd to start dialing
    { "EXECUTE",    "\r"        }, // cmd to cause execution
    { "RESETCMD",   "ATV1Q0"    }, // command to reset modem
    { "CONNECT",    "CONNECT"   }, // connection established
    { "NO-CONNECT", "NO CARRIER" }, // no connection message
    { "OK",         "OK"        }, // OK response
    { "ERROR",      "ERROR"     } // error response
    };

/* ************************************************************ *
 *   8259 Programmable Interrupt Controller Definitions
 * ************************************************************ */
#define I8259       0x20            // control register address
#define EOI         0x20            // end of interrupt command
#define I8259M      0x21            // mask register

/* ************************************************************ *
 *  Fax specific variables, structs, etc.                       *
 * ************************************************************ */

char    dis_msg[] =             // DIS frame data
        { 0,                    // first byte:
                                //   not G1, G2
          0x70,                 // second byte:
                                //   T.4 operation
                                //   9600/7200/4800/2400 bps
                                //   3.85 lines/mm
                                //   one-dimensional coding
          0x02                  // third byte:
          };                    //   1728 pels/215mm
                                //   A4 paper only
                                //   40ms receive time

char    dcs_msg[] =             // DCS frame data
        { 0,                    // first byte:
                                //   not G1, G2
          0x40,                 // second byte:
                                //   T.4 operation
                                //   speed to be set
                                //   3.85 lines/mm
                                //   one-dimensional coding
          0x02                  // third byte:
          };                    //   1728 pels/215mm
                                //   A4 paper only
                                //   40ms per line

struct HDLC_msg                 // format of HDLC message
    {
    UCHAR   addr;               // address byte (always 0xff)
    UCHAR   ctl_fld;            // control field
```

```
    UCHAR   fax_ctl_fld;            // FAX control field (FCF)
    UCHAR   data[253];              // optional data
    int     len;                    // received frame length
    };

struct FxStat                       // Fax status structure
    {
    void    *f_parm;                // Fax init parameter
    char    f_msg[50];              // message string
    char    *f_ptr;                 // pointer to message
    };

struct FxHdr                        // Fax file header structure
    {
    char    ff_type[3];             // file type (G3)
    char    ff_dcs[16];             // DCS for transfer
    char    ff_id[21];              // original station ID
    char    ff_reserved[88];        // reserved space
    };


/*****************************************/
/*  Comm class                           */
/*****************************************/
class Comm
    {
    public:
            Comm(int b,             // define a comm instance
                 int i,             //   base addr, interrupt
                 int d,             //   baud rate divisor
                 int l,             //   line control setting
                 int fc = 0,        //   flow control flag
                 UINT si = 3200,    //   input queue size
                 UINT so = 1500);   //   and output queue size
          UINT Read(char *c,        // read a character from queue
                    char *m,        // ..and get modem status reg
                    char *l),       //   and line status register
            Set8n(void),            // set 8 data bits, no parity
            ICount(void),           // get depth of input queue
            OCount(void),           // ..or output queue
            IFree(void),            // free space in input queue
            OFree(void);            // ..or output queue
        int Modem(void),            // rtn modem status register
            ModemChanged(void),     // rtn TRUE if msr changed
            IFlow(void),            // rtn input flow ctrl status
            IEmpty(void),           // rtn TRUE if input queue
            OEmpty(void);           // ..or output queue empty
       long GetSpeed(void);         // rtn current speed in bps
       void SetSpeed(int d),        // set up port's divisor
            SetBPS(long s),         // set up port's speed
            SetLine(int l),         // ..and line control register
```

```
        Write(int c),              // write a character
        Write(char *s),            // ..or a string of characters
        Write(char *s, int l),     // ..or a block of characters
        IClear(void),              // clear input queue
        OClear(void),              // ..and output queue
        DTR(long t = 1500L),       // lower DTR temporarily
        RTS(int),                  // RTS signal control
        IntRoutine(void);          // interrupt service routine
    ~Comm();                       // destructor

private:
    UINT base,                     // base port address
        irq,                       // interrupt number
        divisor,                   // baud rate divisor
        line,                      // initial line control
        i_size,                    // input buffer size
        o_size,                    // output buffer size
        i_start,                   // flow ctl restart limit
        i_stop,                    //   and upper stop limit
        i_count,                   // characters in input queue
        o_count,                   // ..and output queue
        Deque(void);               // deque an output queue char
    char *i_buf,                   // input buffer
        *i_get,                    // ..and nxt user get location
        *i_put,                    // ..and nxt comm put location
        *i_limit,                  // ..and last location
        *i_last,                   // ..and last i_put location
        i_of,                      // ..and input overflow flag
        *o_buf,                    // output buffer
        *o_get,                    // ..and nxt comm get location
        *o_put,                    // ..and nxt user put location
        *o_limit,                  // ..and last location
        msr_changed,               // msr changed flag
        int_msr,                   // last interrupt msr
        int_lsr,                   // ..and last interrupt lsr
        fifo,                      // 16550 flag
        flow,                      // flow control enable flag
        o_flow,                    // output flow controlled flag
        i_flow,                    // input flow controlled flag
        empty_trans;               // empty transmitter flag
    void InstallInt(void),         // install interrupt svc rtn
        DeInstallInt(void),        // de-install interrupt rtn
        SetLimits(void),           // set up flow ctl limits
        CheckFifo(void),           // set up fifo flags
        Queue(int c,               // queue char to input queue
              int m,               //   saving modem status reg
              int l),              //   and line status register
    interrupt (*old_comm)(...);    // old comm interrupt pointer
};
Comm    *comm;                     // current Comm instance
void    interrupt far comm_int(...);  // comm interrupt routine
```

```
/*****************************************
 * Protocol class                        *
 *****************************************/
class Protocol
  {
  public:
        int  TimeOut(long *n);          // check for timeout
        UINT CRC(char *s, int l),       // CRC calculation
             CheckSum(char *s, int l);  // checksum calculation
        void Purge(Comm *c, long to);   // purge incoming data
  };


/* ************************************************************** *
 *  Fax class definition                                          *
 * ************************************************************** */

class Fax  Protocol
  {
  public:
        Fax    (Comm *,              // Fax instance contstructor
        char *);              //  Comm instance, station ID

               ~Fax(void);           // destructor

        void   Send(char *, char *),  // send a fax
               Receive(char *),       // receive a fax
               fax_stat(int,int,struct FxStat *);
        Comm   *cp;                   // Comm instance

  private:
        void   Display(char c),
               Display(char *s,int);

        int    Send_CSI(void),        // send optional CSI
               HDLC_Mode(int),        // set HDLC tx/rx mode
               Data_Mode(int, int),   // set data tx/rx mode
               Get_Speeds(void),      // get tx and rx speeds
               Get_Line(char *, int), // get a CR-terminate line
               Get_Char(char *, int), // get a character
               Rcv_Hmsg(void),        // receive an HDLC frame
               Send_Our_ID(UCHAR),    // send our ID to remote
               Send_Hmsg(UCHAR,UCHAR,int), // send an HDLC frame
               Send_TSI(void),        // send TSI frame to remote
               Send_DIS(void),        // send DIS frame to remote
               Send_DCS(void),        // send DCS frame to remote
               Init_Modem(void),      // initialize modem
               connected;             // HDLC link active
        UINT   oldparms,              // old communications parameters
               pbi;                   // page buffer index
        long   oldspeed;              // old commuincations speed
```

```
    void    Reverse_Bytes(UCHAR *, int),// reverse bits in char array
            Display_ID(char *, UCHAR *),// display remote station ID
            Display_Msg(int,int),       // display a fax_msgs message
            Reset_Modem(void);          // reset modem and comm parms
    UCHAR   speeds,                     // speeds supported
                                                    // 1... .... 9600 transmit
                                                    // .1.. .... 7200
                                                    // ..1. .... 4800
                                                    // ...1 .... 2400
                                                    // .... 1... 9600 receive
                                                    // .... .1.. 7200
                                                    // .... ..1. 4800
                                                    // .... ...1 2400
        *pagebuf,                   // page buffer area
            Reverse_Byte(UCHAR value);  // reverse bits in a byte
    char    *station;               // our station id
    struct  FxStat fs;              // fax status structure
    struct  FxHdr  fh;              // fax file header structure
    union   {                       // accesss HDLC msg as chars
            struct  HDLC_msg hmsg;  // HDLC message area
            UCHAR   cmsg[256];      // .. same as char array
            };
};

/* ************************************************************
 * Fax -- Fax instance constuctor
 * ************************************************************/

Fax::Fax(Comm *ci,                  // comm instance
        char *sid)                  // .. our station ID (telno)
//      int (*sr)(0,fs)|            // .. status routine
//          void *w)                // .. fax status parameter

{

cp = ci;                            // save comm instance pointer
//stat = (*)(0,fs); //sr;           // .. and status routine
station = sid;                      // .. and station ID

connected = 0;                      // no HDLC connection active

//fs.f_parm = w;                    // save fax status parameter

strcpy(fh.ff_type, "G3");           // preset the file type
memset(fh.ff_reserved, ' ',         // .. and clear reserved area
        sizeof(fh.ff_reserved));

pagebuf = new UCHAR[1024];          // allocate page buffer

}
```

```
/* ***************************************************************
 *  ~Fax -- destructor
 * ***************************************************************/

Fax::~Fax(void)                    // Fax destructor

{
delete pagebuf;                    // free pagebuf memory
}



/* ***************************************************************
 *  Init_Modem -- initialize the fax modem
 *
 *  returns: -2 = User pressed ESC
 *           -1 - Timeout
 *            0 = successful: OK response from modem
 *            1 = ERROR response from modem
 * ***************************************************************/

int Fax::Init_Modem(void)

{
int     rc,                        // return code
        i;                         // work counter for loops
char    buff[80],                  // work buffer for speeds
       *c,                         // work pointer
       *t;                         // token work pointer


cp->Write("\r");                   // send a <CR> to modem
Purge(cp, 2);                      //  kill any receive messages

oldspeed = cp->GetSpeed();         // get the old link speed
oldparms = cp->Set8n();            // .. and old comm parameters
                                   //  .. while setting 8,n,1

cp->SetBPS(19200L);                // set new comm speed

for (rc = i = 0; (rc == 0) && i++ < 3;)    // try three times
  {
  cp->Write(FAX_SETMDM);           // set the modem parameters

  rc = wait_for(FAX_OK, FAX_ERR, 2);       // wait for OK response
  }

if (rc < 0)                        // q. user cancellation?
  return(rc - 1);                  // a. yes .. return error

speeds = 0;                        // reset speeds supported
```

```
cp->Write(FAX_CLASS1);                  // place modem in CLASS 1 mode

rc = wait_for(FAX_OK, FAX_ERR, 5) - 1;      // wait for modem response

if (rc)                                 // q. modem respond ok?
  return(rc);                           // a. no .. tell the caller

cp->Write(FAX_TX_SPD);                  // retrieve transmit speeds

rc = Get_Line(buf, 5);                  // kill the first CR
rc = Get_Line(buf, 5);                  // get the line from the mode

if (rc)                                 // q. any error?
  return(rc);                           // a. yes.. return w/error

Purge(cp, 1);                           // kill additional characters

c = (*buf == '\n') ? buf+1 : buf;       // select start point

while ((t = strtok(c, ",")) != 0)       // while there are tokens
  {
  c = NULL;                             // continue searching buf

  i = atoi(t);                          // get the token's value

  switch(i)                             // for various values..
        {
        case 24:                        // 2400 found
          speeds |= FAX_T2400;          // .. set set the speed flag
          break;

        case 48:                        // 4800 found
          speeds |= FAX_T4800;          // .. set set the speed flag
          break;

        case 72:                        // 7200 found
          speeds |= FAX_T7200;          // .. set set the speed flag
          break;

        case 96:                        // 9600 found
          speeds |= FAX_T9600;          // .. set set the speed flag
          break;
        }
  }

cp->Write(FAX_RX_SPD);                  // retrieve receive speeds

rc = Get_Line(buf, 5);                  // kill the first CR
rc = Get_Line(buf, 5);                  // get the line from the modem
if (rc)                                 // q. any error?
```

```
    return(rc);                     // a. yes.. return w/error

Purge(cp, 1);                       // kill additional characters

c = (*buf == '\n') ? buf+1 : buf;       // select start point

while ((t = strtok(c, ",")) != 0)       // while there are tokens
    {
    c = NULL;                       // continue searching buf

    i = atoi(t);                    // get the token's value

    switch(i)                       // for various values..
            {
            case 24:                        // 2400 found
                speeds |= FAX_R2400;            // .. set set the speed flag
                break;

            case 48:                        // 4800 found
                speeds |= FAX_R4800;            // .. set set the speed flag
                break;

            case 72:                        // 7200 found
                speeds |= FAX_R7200;            // .. set set the speed flag
                break;

            case 96:                        // 9600 found
                speeds |= FAX_R9600;            // .. set set the speed flag
                break;
            }
    }

return(0);                          // return ok
}


/* ********************************************************************** *
 *  Reset_Modem -- reset modem and communications parameters
 * ********************************************************************** */

void Fax::Reset_Modem(void)

{
int     rc;                         // return code

cp->Write("\r");                    // send a <CR> to modem
cp->DTR(1);                         // lower DTR
Purge(cp, 1);                       // .. kill any receive messages

cp->SetBPS(oldspeed);               // reset the comm speed
cp->SetLine(oldparms);              // .. and the comm parms
```

```
cp->Write(FAX_RSTMDM);                  // try to reset the modem

switch (rc = wait_for(FAX_OK, FAX_ERR, 2))  // based on response
   {
   case -1:                             // user pressed escape
            return;                     // .. return without message

   case 0:                              // time out
   case 1:                              // received OK
   case 2:                              // received ERROR
     Display_Msg(37+rc,0);              // display appropriate message
   break;
   }
}


/* ********************************************************************** *
 *  Send_Our_ID - send our ID to the other station
 *
 *  returns: -2 = User pressed ESC
 *           -1 - Timeout
 *            0 = successful; CONNECT response from modem
 *            1 = NO CARRIER response from modem
 * ********************************************************************** */

int Fax::Send_Our_ID(UCHAR ctl_byte)        // FAX control field

{
int     i, j;                           // work variables
UCHAR   *ch;                            // work pointer

ch = hmsg.data;                         // get the data address
                                        // prepare "EOT" to .data
for (i = 0; i < 20; ch[i++] = 0x04);        // set to 'reversed' blanks

if (station != NULL)                    // q. station ID set?
   {                                    // a. yes ..
   i = strlen(station);                 // get station ID length

   i = i > 20 ? 20 : i;                 // max ID length = 20

   for (j = 0; i--;)                    // for each char in station ID
            {
if(strchr("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ",station[i]) != 0)
 ch[j++] =Reverse_Byte(station[i]);      // a. yes.. reverse & copy it

            }
   }
 sprintf(fs.f_msg, "%s %s","Sending our ID#",station);
 fax_stat(0,1,&fs);                     // @@ modi show our id
```

```
return(Send_Hmsg(NON_FINAL, ctl_byte, 20)); // send the HDLC message
}



/* ********************************************************************** *
 * Display_ID – display remote station ID
 * ********************************************************************** */

void Fax::Display_ID(char *pf,              // prefix
                     UCHAR *id)             // id string

{
char    cw[21];                     // work area

int     i, j;                       // work variables

for (i = 20; i–;)                   // backscan the ID
  if (id[i] != ' ')                 // q. blank?
          break;                    // a. no .. get out now

if (i++ == -1)                      // q. ID found?
  return;                           // a. no .. leave now


for (j = 0; i–; cw[j++] = id[i]);   // copy the ID in reverse
cw[j] = 0;                          // .. and end the string

sprintf(fs.f_msg, "%s %s", pf, cw);     // put message in buffer
fax_stat(1,1,&fs);
}



/* ********************************************************************** *
 * Rcv_Hmsg – receive an HDLC frame
 *
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successfully received
 *           1 = NO CARRIER or error
 * ********************************************************************** */

int Fax::Rcv_Hmsg(void)

{
int     rc,                         // return code
        loop = 1,                   // loop until finished
        dleflag = FALSE;            // dle not seen yet
```

```
char    *nxtchar,                       // next receive address
        wc;                             // work for char reads
long    timer;                          // workspace for timer

hmsg.len = 0;                           // initialize length
nxtchar = (char *) cmsg;                // .. and next receive pointer
if ((rc = HDLC_Mode(RECEIVE)) != 0)     // q. connect ok?
  return(rc);                           // a. no .. leave w/error
timer = SECS(5);                        // start 5 second timer

while(loop)                             // enter a loop ..
  switch(cp->Read(nxtchar, &wc, &wc))   // attempting to read bytes
    {
    case -1                             // no character available
        if (TimeOut(&timer))            // q. timeout?
          return(-1);                   // a. yes.. tell the caller

//      if ((stat)(0, &fs))             // q. user press escape?
//        return(-2);                   // a. yes.. tell the caller

        continue;                       // else .. continue loop

      case 0:                           // character received
        hmsg.len++;                     // increment received count

        if (dleflag)                    // q. previous char DLE?
          {                             // a. yes ..
          dleflag = FALSE;              // .. reset the flag

          if (*nxtchar == ETX)          // q. DLE ETX sequence?
            loop = 0;                   // a. yes .. exit the loop
          }

        else if (*nxtchar == DLE)       // q. char a DLE?
          dleflag = TRUE;               // a. yes .. show true

        nxtchar++;                      // point at next character
        break;                          // end this case

      default:                          // lost characters
          return(1);                    // .. show receive unsuccessful
      }

rc = wait_for(FAX_OK, FAX_ERR, 5) - 1;  // OK should follow message
Reverse_Bytes(cmsg, hmsg.len - 2);      // reverse the bits

//@@ data  live in  cmsg var in COMM  class as complete massage
if(exp_0>exp_1){
memcpy(sndrcve[exp_0+1][3].msg,cmsg,256);//strlen(cmsg));
sndrcve[exp_0+1][3].len=hmsg.len;
exp_1=exp_0+1;
}
```

```
if(exp_0<=exp_1){
memcpy(sndrcve[exp_1][3].msg,cmsg,256);//strlen(cmsg));
sndrcve[exp_1][3].len=hmsg.len;
exp_1+=1;
}
//@@
return(rc);                          // return to caller

}



/* ************************************************************** *
 * Send_Hmsg – send an HDLC frame
 *
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful; CONNECT response from modem
 *           1 = NO CARRIER response from modem
 * ************************************************************** */

int Fax::Send_Hmsg(UCHAR finalflg,          // final flag
                   UCHAR ctl_byte,          // fax control field
                   int   len)               // length of data bytes

{
int    rc;                           // return code

hmsg.addr        = FAX_ADDR;               // set the address field
hmsg.ctl_fld     = FAX_CTL | finalflg;     // .. and the control field
hmsg.fax_ctl_fld = ctl_byte;               // .. and fax control field
hmsg.data[len]   = DLE;                    // .. add DLE
hmsg.data[len+1] = ETX;                    // .. and ETX
//@@ hmsg is the same location with cmsg because it is union var.
if(exp_1>exp_0){
memcpy(sndrcve[exp_1+1][2].msg,cmsg,256);//len+1);//strlen(cmsg));
sndrcve[exp_1+1][2].len=hmsg.len;
exp_0=exp_1+1;
}
if(exp_1<=exp_0){
memcpy(sndrcve[exp_0][2].msg,cmsg,256);//len+1);//strlen(cmsg));
sndrcve[exp_0][2].len=hmsg.len;
exp_0+=1;
}
//@@
Reverse_Bytes(cmsg, len+3);                // reverse the bits

if (NOT connected)                         // q. connected already?
  {                                        // a. no .. connect now
  rc = HDLC_Mode(TRANSMIT);                // attempt the connection
```

```
  if (rc)                          // q. connect ok?
    return(rc);                    // a. no .. leave w/error

  connected = TRUE;                // else .. show connected.
  }

cp->Write((char *) cmsg, len+5);   // send to remote

if (finalflg)                      // q. final flag on?
  {                                // a. yes ..
  rc = wait_for(FAX_OK, FAX_ERR, 5) - 1;  // .. wait for OK
  connected = FALSE;               // .. and we're not connected
  }

else
  rc = wait_for(FAX_CONN,          // else .. wait for CONNECT
          FAX_NO_CARR, 60) - 1;    // .. or NO CARRIER

return(rc);                        // return to caller

}

/* ************************************************************** *
 * Send_CSI – Send the called subscriber ID signal to caller
 *
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful; CONNECT response from modem
 *           1 = NO CARRIER response from modem
 * ************************************************************** */

int Fax::Send_CSI(void)

{

return(Send_Our_ID(FAX_FCF_CSI));  // send CSI frame to caller

}

/* ************************************************************** *
 * Send_TSI – Send the transmitting subscriber ID signal
 *
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful
 *           1 = unsuccessful
```

```
 * ********************************************************************** */

int Fax::Send_TSI(void)

{

return(Send_Our_ID(FAX_FCF_TSI));          // send TSI frame

}



/* ********************************************************************** *
 *  Send_DIS – Send the digital ID signal to caller
 *
 *  returns: -2 = User pressed ESC
 *           -1 - Timeout
 *            0 = successful
 *            1 = unsuccesful
 * ********************************************************************** */

int Fax::Send_DIS(void)

{
int    i;                          // work variable

for (i = 3; i--; hmsg.data[i] = dis_msg[i]);// copy the DIS to hmsg.data

return(Send_Hmsg(FINAL, FAX_FCF_DIS, 3));   // send the DIS

}



/* ********************************************************************** *
 *  Send_DCS – Send the digital control signal to called station
 *
 *  returns: -2 = User pressed ESC
 *           -1 - Timeout
 *            0 = successful
 *            1 = unsuccesful
 * ********************************************************************** */

int Fax::Send_DCS(void)

{
int    i;                          // work variable

for (i = 3; i--; hmsg.data[i] = dcs_msg[i]);// copy the DCS to hmsg.data

return(Send_Hmsg(FINAL, FAX_FCF_DCS, 3));   // send the DCS
```

```
}

/* ************************************************************** *
 * HDLC_Mode – Enter HDLC mode (transmit or receive)
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful; CONNECT response from modem
 *           1 = ERROR response from modem
 * ************************************************************** */

int Fax::HDLC_Mode(int dir)              // setup for HDLC tx or rx
                                         // 0 = transmit, 1 = receive
{
int rc;                                  // return code

if ((dir == TRANSMIT) && NOT connected)  // q. we transmitting?
  {                                      // a. yes ..
  cp->Write(FAX_SILENT);                 // .. request 80ms silence

  rc = wait_for(FAX_OK,                  // wait for response
        FAX_ERR, 5) - 1;

  if (rc)                                // q. any problems?
        return(rc);                      // a. yes .. return the error
  }

else if (connected)                      // q. connected already?
  {                                      // a. yes ..
  connected = FALSE;                     // .. reset connect status
  return(0);                             // .. and return ok
  }

cp->Write(dir ? FAX_RX_HDLC : FAX_TX_HDLC); // start requested HDLC mode

rc = wait_for(FAX_CONN, FAX_NO_CARR, 60)-1; // see if we connect

return(rc);                              // return to caller

}

/* ************************************************************** *
 * Data_Mode() – Enter Fax Data mode (transmit or receive)
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful; CONNECT response from modem
 *           1 = NO CARRIER response from modem
 * ************************************************************** */
```

```
int Fax::Data_Mode(int dir,                    // 0 = transmit, 1 = receive
                   int speed)                  // speed for transfer

{
int rc;                                        // return code
char   msg[20];                                // work area for message

if ((dir == TRANSMIT) && NOT connected)        // q. we transmitting?
  {                                            // a. yes ..
  cp->Write(FAX_SILENT);                       //  . request 80ms silence

  rc = wait_for(FAX_OK,                        // wait for response
                FAX_ERR, 5) - 1;

  if (rc)                                      // q. any problems?
        return(rc);                            // a. yes .. return the error
  }

sprintf(msg,                                   // in message area
    dir ? FAX_RX_DATA : FAX_TX_DATA,           // .. select direction of xfer
    speed);                                    // .. and build message


cp->Write(msg);                                // write to modem

rc = wait_for(FAX_CONN, FAX_NO_CARR, 60)-1; // see if we connect

return(rc);                                    // return to caller

}


/* ********************************************************************** *
 * Get_Line - retrieve a CR-terminated line of information
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful
 *           1 = data overrun
 * ********************************************************************** */

int Fax::Get_Line(char *buf,                   // buffer to contain info
                  int  secs)                   // length of timeout

{
int    loop = TRUE;                            // loop condition
char   wc;                                     // work for msr, lsr
long   timer;                                  // timer value

timer = SECS(secs);                            // initialize timer
```

```
while(loop)                          // for as long as necessary
  {
  switch(cp->Read(buf, &wc, &wcl)    // attempting to read a byte
    {
        case -1:                     // no character available
      if (TimeOut(&timer))           // q. timeout?
        return(-1);                  // a. yes.. tell the caller

//          if ((stat)(0, &fs))      // q. user press escape?
//        return(-2);                // a. yes.. tell the caller

          continue;                  // else .. continue loop

        case 0:                      // character received
      if (*buf == '\r')              // q. character a CR?
          {                          // a. yes ..
        *buf = 0;                    // .. null it out
        loop = FALSE;                // .. and end the loop
        }

      else                           // else ..
          buf++;                     // .. point to next char position

      break;                         // end this case

    default:                         // lost characters
      return(1);                     // .. show receive unsuccessful
    }
  }

return(0);                           // show we finished ok
}

/* ************************************************************** *
 * Get_Char - retrieve a single character
 * returns: -2 = User pressed ESC
 *          -1 - Timeout
 *           0 = successful
 *           1 = data overrun
 * ************************************************************** */

int Fax::Get_Char(char *c,           // character to retrieve
                  int  secs)         // length of timeout

{
int   loop = TRUE;                   // loop condition
char  wc;                            // work for msr, lsr
long  timer;                         // timer value

timer = SECS(secs);                  // initialize timer
```

```
while(loop)                          // for as long as necessary ..
  {
  switch(cp->Read(c, &wc, &wc))      // attempting to read a byte
    {
    case -1:                         // no character available
      if (TimeOut(&timer))           // q. timeout?
        return(-1);                  // a. yes.. tell the caller

//        if ((stat)(0, &fs))        // q. user press escape?
//          return(-2);              // a. yes.. tell the caller

          continue;                  // else .. continue loop

    case 0:                          // character received
      loop = FALSE;                  // .. end the loop
      break;

        default:                     // lost characters
          return(1);                 // .. show receive unsuccessful
        }
  }

return(0);                           // show we finished ok
}

/* ------------------------------------------------------------------ *
 * Display_Msg -- display a message without parameters
 * ------------------------------------------------------------------ */

void Fax::Display_Msg(int msgno,int sta)

{

fs.f_ptr = fax_msgs[msgno];          // set the address
fax_stat(sta,2,&fs);

}



/* ------------------------------------------------------------------ *
 * Reverse_Byte() -- Reverse the bits in a byte
 * ------------------------------------------------------------------ */

UCHAR Fax::Reverse_Byte(UCHAR value)     // byte to reverse

{

  _asm  mov cx, 8                    // cx = bits to shift
```

```
_asm  mov al, value                    // al = starting value

top_loop:                              // top of reverse loop
_asm  shl  ah, 1                       // shift ah up by one

_asm  shr  al, 1                       // shift out a bit
_asm  adc  ah, 0                       // .. add carry into ah

_asm  loop top_loop                    // .. until all bits moved

_asm  mov  value, ah                   // save reversed value

return(value);                         // .. and return it

}


/* ************************************************************** *
 *  Reverse_Bytes -- reverse the bits in the bytes of a string
 * ************************************************************** */

void Fax::Reverse_Bytes(UCHAR *str,        // string to reverse
                        int   len)         // length of string

{

while(len--)                           // while there are bytes
   {
   *str = Reverse_Byte(*str);          // .. reverse the bits
   str++;                              // .. next byte
   }

}


/* ************************************************************** *
 *  Receive -- Receive a Facsimile transmission
 * ************************************************************** */

void Fax::Receive(char *faxname)

{
int    rc = 0,                         // return code
       speed,                          // link speed
       fileopen = FALSE,               // receive file not open
       dcs_received = FALSE,           // DCS not yet received
       i,                              // work variable
       error = 0,                      // error number
       dleflag = FALSE,                // DLE sequence found flag
       loop = TRUE;                    // loop until complete
```

```
FILE   *faxfile;                    // fax file

char    c;                          // work character

ULONG   pagelen,                    // length of received page
        reclen,                     // offset of page record length
        et,                         // elapsed time
        st;                         // work for timer

enum    RcvStates                   // FAX receive states
        {
        PhaseA,                     // phase A - make connection
        AwaitCall,                  // wait for a call
        AnswerCall,                 // answer incoming call
        SendCSI,                    // send our CSI frame, if needed
        PhaseB,                     // negotiate session parameters
        GetFrames,                  // get HDLC frames
        SetSpeed,                   // set the link speed
        PrepTCF,                    // prepare to receive TCF
        FailConnTCF,                // fail the TCF connect
        RcvTCF,                     // receive/test TCF
        RcvTCF1,                    // get rest of TCF
        FailTCF,                    // fail the TCF
        Confirm,                    // confirm training check frame
        PhaseC,                     // receive the FAX data
        RcvPage,                    // receive 1 page of data
        EndPage,                    // page complete
        PhaseD,                     // post message procedure
        PhaseE,                     // call is done .. disconnect
        RcvError,                   // error during receive
        UserCan,                    // user cancelled transmission
        ExitFax                     // exit FAX receive
        } state;

strcpy(fh.ff_id, "          ");     // blank out the ID string
state = PhaseA;                     // start in Phase A

if ((faxfile = fopen(faxname, "wb+"))     // q. receive file open ok?
                        == NULL)
    {                               // a. no ..
    error = 9;                      // .. set the error code
    state = RcvError;               // .. and declare an error
    }

fileopen = TRUE;                    // show the file is open

while(loop)                         // top of state machine
  switch(state)                     // perform next state
    {
    case PhaseA:                    // phase A - make connection
                Display_Msg(1,0);   // update the status
          if ((rc = Init_Modem()) != 0)     // q. modem init ok?
```

```
        {
        error = 1;                  // a. no .. exit now
              state = RcvError;             // .. declare the error
              continue;
              }

      state = AwaitCall;              // else . wait for a ring

            Display_Msg(2,0);               // update the status
            break;                          // end this state

    case AwaitCall:                 // wait for a call
      rc = wait_for(FAX_RING,       // wait for a RING from modem
                              FAX_RING, 10);

            if (rc > 0)                     // q. RING arrive?
            {                               // a. yes
            state = AnswerCall;             // .. go answer the call
            continue;
            }

      if (rc == -1)                 // q. user press ESC?
            state = UserCan;                // a. yes .. exit now

            break;                          // else .. continue waiting

    case AnswerCall:                // answer incoming call
            Display_Msg(3,0);               // update the status
      cp->Write(FAX_ANSWER);        // send the answer command

            rc = wait_for(FAX_CONN,         // wait for a carrier
                  FAX_NO_CARR, 60);

            if (-rc == 0)                   // q. connect?
                  {
            state = SendCSI;                // a. yes .. send our CSI
      connected = TRUE;             // .. show we're connected
      }

      else                          // else ..
        {
      state = RcvError;             // .. process general error
            error = 2;                      // .. show the error type
            }

            break;                          // .. next state

    case SendCSI:                   // send our CSI frame
//        Display_Msg(4,0);               // update the status
      rc = Send_CSI();              // .. send our ID

      if (rc)                       // q. error?
```

```
        state = RcvError;               // a. yes .. declare the error

            else
                state = PhaseB;             // select next state

        break;

    case PhaseB:                        // negotiate session parameters
            Display_Msg(5,0);               //
        state = ((rc = Send_DIS()) != 0) ? // q. DIS send go ok?
                                                RcvError : // a. no .. declare error
                                                GetFrames; // else .. get frames
            break;

    case GetFrames:                     // get HDLC frames
            if ((rc = Rcv_Hmsg()) != 0)     // q. any error getting a frame?
        if (rc < 0)                 // a. yes .. timeout or ESC?
            {                       // a. yes
                        error = 3;              // .. error receiving HDLC frame
            state = RcvError;       // .. leave with error
            continue;               // .. continue process
                    }
            else                    // else .. NO CARRIER
                {
            cp->Write(FAX_SILENT1); // wait for 200ms silence

            rc = wait_for(FAX_OK,   // wait for the OK response
                    FAX_ERR,
                    10) - 1;

                state = SetSpeed;       // .. and set the speed
                }

            if (rc)                 // q. any error yet?
            {                   // a. yes ..
                state = RcvError;           // .. declare an error
            continue;
            }

        switch(hmsg.fax_ctl_fld)        // process based on message type
            {
                case FAX_FCF_TSI:           // transmitting subscriber ID
                        Display_Msg(6,1);           // update the status
                        Reverse_Bytes(hmsg.data,    // .. reset the bytes to normal
                            20);
                hmsg.data[20] = 0;          // .. end string in zero

                strcpy((char *) fh.ff_id,   // .. copy it to file header
                    (char *) hmsg.data);

                        Display_ID("Called by:",    // .. and display the remote ID
                                                    hmsg.data);
```

```
                break;

case FAX_FCF_DCN:           // disconnect
                Display_Msg(16,1);          // update the status
                state = PhaseE;             // .. and hang up
                continue;

        case FAX_FCF_DCS:           // digital command signal
                Display_Msg(7,1);           // update the status
                memcpy(fh.ff_dcs, hmsg.data,// .. move the DCS to the header
                        hmsg.len - 7);

                dcs_received = 1;           // show we got DCS
                break;
        }

    if (hmsg.ctl_fld == FAX_CTL_FF)   // q. final frame?
        {                             // a. yes ..
        state = SetSpeed;             // .. set link speed

        fseek(faxfile, 0L, SEEK_SET); // go to start of file
        fwrite(&fh, 128, 1, faxfile); // .. and write the header
    }

break;

case SetSpeed:                  // set the link speed
  if (dcs_received == 0)        // q. dcs received?
    {                           // a. no ..
        error = 4;              // .. show the error
        state = RcvError;       // .. and go to error state
        continue;               // .. continue with next state
    }

    switch (fh.ff_dcs[1] & 0x30)    // get modulation value
        {
        case 0x00:              // q. speed 2400?
                speed = 24;             // a. yes .. set speed
                i = 44;                 // .. and message
                break;

        case 0x10:              // q. speed 4800?
                speed = 48;             // a. yes .. set speed
                i = 43;                 // .. and message
                break;

        case 0x30:              // q. speed 7200?
                speed = 72;             // a. yes .. set speed
                i = 42;                 // .. and message
                break;

        case 0x20:              // q. speed 9600?
```

```
                    speed = 96;              // a. yes .. set speed
                    i = 41;                  // .. and message
                    break;
                }

                Display_Msg(i,0);            // display speed message
                state = PrepTCF;             // next, prepare to test link
                break;

case PrepTCF:                                // prepare to receive TCF
    rc = Data_Mode(RECEIVE, speed);          // start the receive

    if (rc == 1)                             // q. did connection fail?
        state = FailConnTCF;                 // a. yes.. fail TCF connect

    else if (rc != 0)                        // q. other failure?
        state = RcvError;                    // a. yes .. declare an error

        else
        {
        state = RcvTCF;                      // receive training check frame
            Display_Msg(9,1);                // update the status
        }

    break;

case FailConnTCF:                            // fail the TCF connect
        rc = Send_Hmsg(FINAL,                // send failure to train
                       FAX_FCF_FTT, 0);

    if (rc)                                  // q. retrain send fail?
            state = RcvError;                // a. yes .. declare an error

    else
            state = GetFrames;               // get a new set of frames

    break;

case RcvTCF:                                 // receive/test TCF
        rc = Get_Char(&c, 5);                // wait a max of 5 secs for data

    if (rc != 0)                             // q. any error
      {                                      // a. yes ..
      state = RcvError;                      // .. declare an error
      continue;                              // .. and continue processing
      }

        if ((UCHAR) c == 0)                  // q. zero byte?
          state = RcvTCF1;                   // a. yes .. receive the rest

    break;
```

```
case RcvTCF1:                   // get rest of TCF
        st = get_time();        // get time of day in ticks

    for(;;)                     // get characters
        {
        rc = Get_Char(&c, 1);   // get a character

    if (rc != 0)                // q. any error?
        {                       // a. yes
        Purge(cp, 1);           // .. purge the comm line
        error = 7;              // .. TCF too short
        state = RcvError;       // .. declare an error
        break;                  // .. and exit loop
                }

        if (c != 0)             // q. end of sequence?
                break;          // a. yes .. exit loop
    }

if (rc != 0)                    // q. was there an error?
    continue;                   // a. yes .. continue process

et = elapsed_time(st);          // calculate elapsed time

        if ((et < 24) || (et > 31))     // q. 1.5 seconds +/- 10%?
        {                       // a. yes ..
        Purge(cp, 1);           // .. purge the comm line
    state = FailTCF;            // .. declare train failure
    continue;                   // .. and continue loop
    }

rc = wait_for(FAX_NO_CARR,      // wait for no carrier
        FAX_ERR, 5) - 1;        // .. or other information

        if (rc != 0)            // q. NO CARRIER received?
        {                       // a. no ..
        state = RcvError;       // .. declare receieve error
    continue;                   // .. and continue loop
        }

state = Confirm;                // else .. confirm TCF
        break;                  // .. and continue processing

case FailTCF:                   // fail the TCF
        Display_Msg(19,0);      // update the status

        rc = Send_Hmsg(FINAL,   // send failure to train
            FAX_FCF_FTT, 0);

    if (rc)                     // q. retrain send fail?
        state = RcvError;       // .. declare an error
```

```
else
        state = GetFrames;          // get a new set of frames

    break;

case Confirm:                       // confirm training check frame
        Display_Msg(10,0);          // update the status
    rc = Send_Hmsg(FINAL,           // send the confirmation
            FAX_FCF_CFR, 0);

    if (rc != 0)                    // q. confirmation go ok?
      {
        state = RcvError;           // a. no .. show the error
        continue;                   // .. continue processing
        }

    state = PhaseC;                 // start receiving FAX
    break;

case PhaseC:                        // receive the FAX data
    rc = Data_Mode(RECEIVE, speed); // start receiving the FAX

      if (rc != 0)                  // q. receive start ok?
        {
        state = RcvError;           // a. no .. show the error
      continue;                     // .. continue processing
        }

    Display_Msg(11,0);              // update the status

    state = RcvPage;                // receive a page of data

        dleflag = FALSE;            // set no DLE seen yet

        pbi = 0;                    // reset  the page buffer index
    pagelen = 0;                    // .. and no page data yet

    fseek(faxfile, 0L, SEEK_END);   // go to end of file
    reclen = ftell(faxfile);        // .. save the position
    fwrite(&pagelen, 4, 1, faxfile); // .. write the page's length
    break;

case RcvPage:                       // receive 1 page of data
        rc = Get_Char(&c, 1);       // get a character

    if (rc != 0)                    // q. receive ok?
            {                       // a. no ..
      state = RcvError;             // .. declare an error
      continue;                     // .. and continue processing
            }
```

```
pagebuf[pbi++] = c;              // save the character
        pagelen++;                       // increment bytes in page


if (pbi == 1024)                 // q. buffer full?
   {                             // a. yes ..
   fwrite(pagebuf, 1024, 1,      //    write out a page
          faxfile);

   pbi = 0;                      // reset page buffer index
         }

      if (dleflag)               // q. previous char DLE?
         {                       // a. yes ..
   dleflag = FALSE;              // .. reset the flag

   if (c == ETX)                 // q. DLE ETX sequence?
    {                            // a. yes
                state = EndPage;         //    process end of page
                Display_Msg(12,0);       // update the status
    }
      }

      else if (c == DLE)         // q. char a DLE?
   dleflag = TRUE;               // a. yes .. show true

   break;                        // continue processing

case EndPage:                    // Page complete
   rc = wait_for(FAX_NO_CARR,    // we should now have no carrier
                 FAX_ERR, 5) - 1;

      if (pbi)                   // buffer contain data?
         fwrite(pagebuf, pbi, 1,       // a. yes.. write out a page
          faxfile);

   fseek(faxfile, reclen, SEEK_SET);  // set postion to length record
   fwrite(&pagelen, 4, 1, faxfile);   // update the page's length
         fseek(faxfile, 0L, SEEK_END);      // .. return to end of file

   if (rc != 0)                  // q. correct response?
         {                       // a. no ..
         state = RcvError;             // .. declare an error
         continue;
    }

   state = PhaseD;               // start post message procedure
   break;

case PhaseD:                     // post message procedure
   rc = Rcv_Hmsg();              // get the next message
```

```
        if (rc != 0)                    // q. correct response?
            {                           // a. no
        state = RcvError;               // .. declare an error
            continue;
        }

            i = hmsg.fax_ctl_fld;       // save the response

    rc = Send_Hmsg(FINAL,               // send the final frame
                                        FAX_FCF_MCF, 0);    // .. message confirmation

            if (rc != 0)                // q. correct response?
        {                               // a. no
        state = RcvError;               // .. declare an error
        continue;
        }

    switch(i)                           // based on the response..
            {
            case FAX_FCF_MPS:           // multi-page signal
                    Display_Msg(13,1);          // update the status
                    state = PhaseC;             // .. receive next page
            break;

        case FAX_FCF_EOM:               // end of message
                    Display_Msg(14,1);          // update the status
                    state = PhaseB;             // .. renegotiate & continue
            break;

            case FAX_FCF_EOP:           // end of procedure
                    Display_Msg(15,1);          // update the status
                    state = PhaseE;             // .. disconnect now
            break;

        case FAX_FCF_DCN:               // disconnect
                    Display_Msg(16,0);          // update the status
            state = PhaseE;             // .. and hang up
            continue;

              default:
                    state = RcvError;           // unknown response
                    break;
        }

        break;                          // continue processing

    case PhaseE:                        // call is done .. disconnect
        cp->Write(FAX_HANGUP);          // hangup the modem
        state = ExitFax;                // .. and exit the fax receive
            break;

    case RcvError:                      // error during receive
```

```
Display_Msg(17,0);              // update the status

      if (rc == -2)             // q. user cancellation?
        {                       // a. yes ..
        state = UserCan;        //  .. show the reason
        continue;               //  .. continue processing
            }

          else if (error)       // q. error set?
            {                   // a. yes ..
        fs.f_ptr = fax_errors[error]; //  .. set the message pointer
fax_stat(0,2,&fs);
          }

            else if (rc == -1)          // q. timeout?
              Display_Msg(20,0);        // a. yes .. show the message

            else if (rc == 1)           // q. unexpected response?
              Display_Msg(21,0);        // a. yes .. tell the user

      state = ExitFax;          //  .. set new state
      break;

    case UserCan:               // user cancelled transmission
      Display_Msg(18,2);        // update the status
      state = ExitFax;          //  .. set new state
      break;

    case ExitFax:               // exit FAX receive
      if (fileopen)             // q. fax file open?
        fclose(faxfile);        // a. yes .. close the file

      Reset_Modem();            // reset the modem

      loop = FALSE;             //  .. end the loop
      break;                    //  .. and return to caller
  }
}


/* ************************************************************************
 * Send – Send a Facsimile
 * ************************************************************************ */

void Fax::Send(char *faxname,   // fax file name
               char *telno)     // telephone number

{
int   rc = 0,                   // return code
      speed,                    // link speed
      nextspeed,                // next link speed
```

```
        fileopen = FALSE,           // receive file not open
        dis_received = FALSE,       // DIS not yet received
        i,                          // work variable
        error = 0,                  // error number
        loop = TRUE;                // loop until complete


FILE    *faxfile;                   // fax file

char    dis[16],                    // DIS received
        csi[21],*nw,                // CSI received
            c,                      // work character
            w;                      // work register


ULONG   pagelen,                            // length of received page
            st,                             // work for elapsed time
            t1,
            t2;


unsigned long far   *tim1,
                    *tim2;

enum    SndStates                   // FAX send states
            {
            PhaseA,                 // phase A - make connection
            PhaseB,                 // get HDLC frames
            SetSpeed,               // set the link speed
            SendTSI,                // select speed, send ID & DCS
            SendTCF,                // send the TCF
            GetConfTCF,             // get TCF confirmation
            PhaseC,                 // send the document
            SendPage,               // send a page of data
            PhaseD,                 // post message process
            AnotherPage,            // another page to send
            AllSent,                // send complete
            PhaseE,                 // disconnect
            SndError,               // error during send
            UserCan,                // user cancelled transmission
            ExitFax                 // exit FAX receive
            } state;


strcpy(fh.ff_id,station);   // blank out the ID string
state = PhaseA;                     // start in Phase A


if ((faxfile = fopen(faxname, "rb"))    // q. send file open ok?
                        == NULL)
  {                             // a. no ..
  error = 9;                    // .. set the error code
  state = SndError;             // .. and declare an error
```

```
    }

fread(&fh, 128, 1, faxfile);            // read the header

if (strcmp(fh.ff_type, "G3"))           // q. G3 file?
   {
   snd();                               // a. no ..
   error = 13;                          //  .. show invalid format
   state = SndError;                    //  .. and declare an error
   }

fileopen = TRUE;                        // show the file is open

fread(&pagelen, 4, 1, faxfile);         // read in the first page length

while(loop)                             // top of state machine
 switch(state)                          // perform next state
  {
  case PhaseA:                          // phase A - make connection
          Display_Msg(1,0);             // update the status
          if ((rc = Init_Modem()) != 0) // q. modem init ok?
          {
          error = 1;                    // a. no .. exit now
          state = SndError;             //  .. declare the error
          continue;
          }

sprintf(fs.f_msg, "%s %s",fax_msgs[22],telno);    // put message in buffer
fax_stat(0,1,&fs);
//        Display_Msg(22,0);            // update the status

          cp->Write(FAX_DIAL);          // send the dial command
          cp->Write(telno);             //  .. and the phone number
          cp->Write('\r');              //  .. and finish the command

          rc = wait_for(FAX_CONN,       // wait for a connection
                        FAX_NO_CARR, 120);

        if (!rc)                        // q. any error
          {                             // a. yes ..
          if (rc == 1)                  // q. connection fail?
                  error = 10;           // a. yes .. show connect failed

          state = SndError;             // declare an error
          continue;                     //  .. go to next state
          }

        state = PhaseB;                 // get frames
        connected = TRUE;               //  ... show we are connected
        break;

  case PhaseB:                          // get HDLC frames
```

```
if ((rc = Rcv_Hmsg()) != 0)        // q. any error getting a frame?
   if (rc < 0)                     // a. yes .. timeout or ESC?
           (                       // a. yes ..
           error = 3;              // .. error receiving HDLC frame
           state = SndError;       // .. leave with error
           continue;               // .. continue process
           )
   else                            // else .. NO CARRIER
           (
           cp->Write(FAX_SILENT1);  // wait for 200ms silence

           rc = wait_for(FAX_OK,    // wait for the OK response
                         FAX_ERR,
                         10) - 1;

//         state = SetSpeed;        // .. and set the speed
           }

   if (rc)                         // q. any error yet?
           (                       // a. yes ..
           state = SndError;       // .. declare an error
           continue;
           )

   switch(hmsg.fax_ctl_fld)        // process based on message type
           {
           case FAX_FCF_CSI:       // called subscriber ID
      Reverse_Bytes(hmsg.data,20); // modi.reset the bytes to normal
      hmsg.data[20] = 0;           // .. end string in zero
     strcpy(csi,(char *) hmsg.data); // .. copy it to work area
        Display_ID("Send (CSI)",   // .. and display the remote ID
              (UCHAR *) csi);
      break;

           case FAX_FCF_DIS:       // digital information signal
                   Display_Msg(24,0);      // update the status
                   memcpy(dis, hmsg.data,  // .. move the DIS to work area
                          hmsg.len - 7);
   state = SetSpeed;               // modi @@
                   dis_received = 1;       // show we got DIS
                   break;

           case FAX_FCF_NSF:       // Non-standard facilities
                   Display_Msg(8,0);       // update the status
                   break;          // .. we don't process these
           }

   break;

case SetSpeed:                     // set the link speed
   if (dis_received == 0)          // q. dis received?
```

```
        {
                                                    // a. no ..
        error = 11;             // .. show the error
        state = SndError;       // .. and go to error state
        continue;               // .. continue with next state
        }

      switch (dis[1] & 0x30)        // select initial speed
        {
        case 0x30:              // q. speed 7200?
//      nextspeed = 72;         // a. yes .. set speed
//      break;
        case 0x20:              // q. speed 9600?
                nextspeed = 96;         // a. yes .. set speed
                break;

        case 0x10:              // q. speed 4800?
                nextspeed = 48;         // a. yes .. set speed
                break;

        case 0x00:              // q. speed 2400?
                nextspeed = 24;         // a. yes .. set speed
                break;
        }

        state = SendTSI;        // next, tell 'em about us
        break;

    case SendTSI:               // select speed, send ID & DCS
        dcs_msg[1] &= 0xcf;     // set off speed bits

        switch (speed = nextspeed)      // based on selected speed
        {
        case 96:                // q. speed = 9600?
          dcs_msg[1] |= 0x20;   // a. yes .. set speed

          if ((dis_msg[1] | 0x30)       // q. 7200 supported?
                                        == 0x30)
                nextspeed = 72;         // a. yes .. next speed = 7200

          else
                nextspeed = 48;         // else .. next speed = 4800

          i = 41;               // select speed message
          break;

        case 72:                // q. speed = 7200?
          dcs_msg[1] |= 0x30;   // a. yes .. set speed
          nextspeed = 48;       // .. next speed = 4800
          i = 42;               // select speed message
          break;
```

```
                case 48:                    // q. speed = 4800?
                    dcs_msg[1] |= 0x10;      // a. yes .. set speed
                    nextspeed = 24;          // .. next speed = 2400
                    i = 43;                  // select speed message
                    break;

                case 24:                    // q. speed = 2400?
                    error = 12;              // a. yes.. can't connect
                    state = SndError;        // .. declare the error
                    i = 44;                  // select speed message
                    continue;                // .. continue state machine
                }

                Display_Msg(25,0);           // update the status
                rc = Send_TSI();             // .. send our ID

                if (rc)                      // q. error?
                    {
                    state = SndError;        // a. yes ..
                    continue;                // .. declare the error
                }

                Display_Msg(26,0);           // update status again

            state = ((rc = Send_DCS()) != 0) ? // q. DCS send go ok?
                        SndError : // a. no .. declare error
                                   SendTCF ; // else .. send the check frame
                break;

        case SendTCF:                        // send the TCF
                Display_Msg(i,0);            // show speed
                Display_Msg(27,0);           // update the status

            rc = Data_Mode(TRANSMIT, speed); // go to data mode

                if (rc)                      // q. any error?
                    {                        // a. yes ..
                    state = (rc != 1) ? SndError : // .. select general error
                                        GetConfTCF; // .. or just connection failure

                    continue;
                    }


                tim1 = ( unsigned long far *)MK_FP(0x40,0x6C);
                t1=*tim1;
lpp:
                cp->Write(0);                // send the TCF
                tim2 = ( unsigned long far *)MK_FP(0x40,0x6C);
                t2=*tim2;

                if(t2 < (t1+5))
                goto lpp;
```

```
//for (st = get_time();elapsed_time(st) < 28;)//..let the modem send zeroes
                                        // .. for 1.5 seconds


        cp->Write(DLE);                 // .. ended with DLE
        cp->Write(ETX);                 // .. ETX

            rc = wait_for(FAX_NO_CARR,      // wait for no carrier or OK
                FAX_OK, 5);             // .. for 5 seconds

            if (rc- < 1)                    // q. error?
          state = SndError;             // a. yes .. declare the error

        else
                state = GetConfTCF;         // else .. try for confirmation

        break;

    case GetConfTCF:                    // get TCF confirmation

//      Display_Msg(28,1);              // update the status
        if ((rc = Rcv_Hmsg()) != 0)     // q. any error getting a frame?
            {                           // a. yes ..
          error = 3;                    // .. error receiving HDLC frame
          state = SndError;             // .. leave with error.
          continue;                     // .. continue process
                }

        switch(hmsg.fax_ctl_fld)        // process based on message type
          {
                case FAX_FCF_CFR:           // confirmation
      Display_Msg(28,1);                // update the status
            state = PhaseC;             // send the document
            break;

                case FAX_FCF_FTT:           // failure to train
            Display_Msg(30,1);          // update the status
            state = SendTSI;            // next speed and retry session
            break;

          default:                      // unknown message
                        error = 21;         // unexpected response
            state = SndError;           // .. declare an error
            break;
          }
        break;

    case PhaseC:                        // send the document
            Display_Msg(29,0);              // update the status
        rc = Data_Mode(TRANSMIT, speed);  // .. set data mode
```

```
            if (rc)                     // q. any error?
            {                           // a. yes ..
            state = SndError;           // .. declare an error
            continue;
                }

        state = SendPage;               // else .. send the page
        break;

    case SendPage:                      // send a page of data
            i = pagelen > 128 ?         // determine number of bytes
            128 : (int) pagelen;        //   to read and send

        pagelen -= i;                   // decrement page length

        fread(pagebuf, i, 1, faxfile);  // read data from the file
            cp->VWrite((char *) pagebuf, i);  // .. and send the data

        st = SECS(10);                  // for max of 10 seconds

            while ((cp->QCount() > 256) &&    // let the buffer empty
                    error == 0) //  .. while there's no error
                if (time_out(&(long) st))     // q. timeout?
                error = 15;             // a. yes .. set the error code

            if (error)                  // q. error occur?
            {                           // a. yes ..
            state = SndError;           // .. declare an error
            continue;                   // .. and continue processing
            }

        if (cp->Read(&c, &w, &w) == 0)  // q. char available?
            if (c == XOFF)              // a. yes .. is it XOFF?
            {                           // a. yes ..
//                  Display_Msg(35,0);          //  .. show XOFF received
            sound(4500);
            delay(10);
            nosound();
                    while(cp->Read(&c, &w, &w)  // .. wait for another char
                                == 0xffff);
            sound(3000);
            delay(10);
            nosound();
//                  Display_Msg(36,0);          // .. erase XOFF message
                    }

        if (pagelen == 0)               // q. page complete?
                state = PhaseD;             // a. yes .. continue

        break;
```

```
case PhaseD:                    // post message process
    fread(&pagelen, 4, 1, faxfile);  // read next page length

    while (cp->OEmpty() != 0);      // wait for transmit complete

    rc = wait_for(FAX_OK,           // should get OK
            FAX_ERR, 5) - 1;

        if (rc)                     // q. did send complete?
            {                       // a. no ..
        state = SndError;           // .. declare an error
        continue;
            }

        Display_Msg(31,0);              // update the status

        if (pagelen)                    // q. more pages?
            state = AnotherPage;        // a. yes ... send another

    else
        state = AllSent;            // else .. all sent

        break;

case AnotherPage:                   // another page to send
        Display_Msg(32,0);              // update the status

    rc = Send_Hmsg(FINAL,           // send the multipage signal
                   FAX_FCF_MPS, 0);

    if (rc != 0)                    // q. send ok?
    {                               // a. no ..
            state = SndError;       // .. declare an error
        continue;                   // .. continue processing
    }

        if ((rc = Rcv_Hmsg()) != 0)     // q. any error getting a frame?
            {                           // a. yes ..
        error = 3;                  // .. error receiving HDLC frame
        state = SndError;           // .. leave with error.
        continue;                   // .. continue process
        }

    switch(hmsg.fax_ctl_fld)        // process based on message type
        {
        case FAX_FCF_MCF:           // message confirmation
            state = PhaseC;         // send next page
                    break;

            default:
                    error = 14;             // unexpected response
                    state = SndError;       // .. declare an error
```

```
                continue;
            }
        break;

case AllSent:                   // send complete
        Display_Msg(33,0);              // update the status

        rc = Send_Hmsg(FINAL,           // send end of procedure
                        FAX_FCF_EOP, 0);

        if (rc != 0)                    // q. send ok?
          {                             // a. no ..
          state = SndError;             // .. declare an error
          continue;                     // .. continue processing
          }

        if ((rc = Rcv_Hmsg()) != 0)     // q. any error getting a frame?
          {                             // a. yes ..
          error = 3;                    // .. error receiving HDLC frame
          state = SndError;             // .. leave with error
          continue;                     // .. continue process
          }

        switch(hmsg.fax_ctl_fld)        // process based on message type
          {
          case FAX_FCF_MCF & 0x7f:      // message confirmation
                state = PhaseE;         // send next page
                break;

          default:
                error = 14;             // unexpected response
                state = SndError;       // .. declare an error
/*              cp->Write("+++\r");     //@@modi
                rc = wait_for("OK",",5);
                if(rc == 1)
                {
                cp->Write("ath\r");     //@@ modi
                Reset_Modem();
                }       */
                continue;
          }
        break;

case PhaseE:                    // disconnect
        Display_Msg(34,0);              // update the status

        rc = Send_Hmsg(FINAL,           // send end of procedure
                        FAX_FCF_DCN, 0);

        if (rc != 0)                    // q. send ok?
          {                             // a. no ..
          state = SndError;             // .. declare an error
```

```
                continue;                   // .. continue processing
                }

                state = ExitFax;            // exit fax send procedure
                break;


        case SndError:                      // error during send
                Display_Msg(17,2);          // update the status

                if (rc == -2)               // q. user cancellation?
                  {                         // a. yes ..
                  state = UserCan;          // .. show the reason
                  continue;                 // .. continue processing
                  }

                else if (error)             // q. error set?
                  {                         // a. yes ..
                  fs.f_ptr = fax_errors[error]; // .. set the message pointer
fax_stat(0,2,&fs);
                  }

                else if (rc == -1)          // q. timeout?
                  Display_Msg(20,2);        // a. yes .. show the message

                else if (rc == 1)           // q. unexpected response?
                  Display_Msg(21,0);        // a. yes .. tell the user

                state = ExitFax;            // .. set new state
                break;

        case UserCan:                       // user cancelled transmission
                Display_Msg(18,2);          // update the status
                state = ExitFax;            // .. set new state
                break;

        case ExitFax:                       // exit FAX receive
                if (fileopen)               // q. fax file open?
                  fclose(faxfile);          // a. yes .. close the file

                Reset_Modem();              // .. reset the modem

                loop = FALSE;               // .. end the loop
                break;                      // .. and return to caller
    }
}

/* ************************************************************** *
 * Comm - define communications port instance
 * ************************************************************** */

Comm::Comm(int b,                           // comm port base address
```

```
                int i,                  // interrupt number
        int d,                  // baud rate divisor
        int l,                  // line control setting
                int fc,                 // flow control flag
                UINT si,                // input queue size
                UINT so)                // output queue size
{
UINT    max_size = 64535U / 3;          // max number of input chars

base = b;                       // set up instance base addr
irq = i;                        // ..interrupt request
o_size = so;                    // ..output buffer size
flow = fc;                      // ..flow control flag
CheckFifo();                    // ..fifo flag if available
SetSpeed(d);                    // ..line speed
SetLine(l);                     // ..line format

if (si < 512)                   // q. less than .5k of buffer?
   si = 512;                    // a. yes ... set to minimum

if (si > max_size)              // q. greater than maximum?
   si = max_size;               // a. yes .. set to max size

if (so < 256)                   // q. really small output buf?
   so = 256;                    // a. yes .. set to minimum

i_size = si;                    // save input buffer size
empty_trans = 1;                // ..set empty transmitter flag
msr_changed = 1;                // ..set msr changed flag
o_flow = i_flow = 0;            // ..clear active flags
o_count = i_count = 0;          // ..and current queue counts

i_buf = i_get = i_put =         // allocate input buffer
        new char[i_size * 3];   // ..for data, lsr and msr
i_limit = i_buf + (i_size - 1) * 3;     // ..set limit address
memset(i_buf, 0, i_size * 3);   // ..and clear area to nulls

o_buf = o_get = o_put = new char[o_size];   // allocate output buffer
o_limit = o_buf + o_size - 1;   // ..set limit address
memset(o_buf, 0, o_size);       // ..and clear area to nulls
SetLimits();                    // set up flow ctl limits
InstallInt();                   // ..and interrupt routine
int_msr = IN(MSR);              // set up initial MSR

}

/* ********************************************************************
 * SetLimits - set up receive buffer flow control limits
 ******************************************************************** */
```

```
void    Comm::SetLimits(void)
{

i_stop = (UINT) (115200L / 10) / divisor;   // get 1 second in characters

if (i_stop > i_size)                    // q. limit too high?
   i_stop = i_size / 2;                 // a. yes .. set at half
else
   i_stop = i_size - i_stop;            // else .. set limit point

i_start = i_stop / 2;                   // set restart point at 50%

}


/* ********************************************************************** *
 * CheckFifo -- if UART is a 16550, set FIFO variable to true
 * ********************************************************************** */

void    Comm::CheckFifo(void)
{

fifo = 0;                               // assume standard uart
OUT(FCR, 0xcf);                         // try to enable FIFOs

if ((IN(IIR) & 0xc0) != 0xc0)           // q. FIFO bits found?
   return;                              // a. no .. just return

OUT(FCR, 0);                            // turn off FIFOs
fifo = 1;                               // set fifo available flag

}


/* ********************************************************************** *
 * SetSpeed -- set up port baud rate divisor
 * ********************************************************************** */

void    Comm::SetSpeed(int d)           // baud rate divisor
{

divisor = d;                            // save divisor for later
OUT(IER, 0);                            // clear interrupts enable

if (fifo)                               // q. is UART a 16550 chip?
   OUT(FCR, FCR_16550);                 // a. yes .. enable fifo

IN(LSR);                                // read/reset line status reg
IN(MSR);                                // ..and modem status register
IN(IIR);                                // ..and interrupt id register
```

```
IN(RBR);                        // ..and receive buffer reg

_asm   cli                      // stop interrupts
OUT(LCR, IN(LCR) | LCR_DLAB);           // set divisor latch bit
OUT(DLM, d >> 8);                       // out msb portion of divisor
OUT(DLL, d & 0xff);                     // ..then the lsb portion
OUT(LCR, IN(LCR) & ~LCR_DLAB);          // clear divisor latch bit
OUT(MCR, MCR_DO);                       // enable DTR, OUT2 and RTS
_asm   sti                      // ..and interrupts

OUT(IER, IER_RBF | IER_TBE | IER_MSI);    // then enable UART interrupts

}



/* ********************************************************************* *
 * SetBPS - set port's speed
 * ********************************************************************* */

void   Comm::SetBPS(long s)             // speed in BPS
{

SetSpeed((int)(115200L / s));           // calc and set divisor

}


/* ********************************************************************* *
 * GetSpeed - retrieve current port speed
 * ********************************************************************* */

long   Comm::GetSpeed(void)
{

return((long) (115200L / (long) divisor));  // return current speed

}



/* ********************************************************************* *
 * SetLine - set up port's line control register
 * ********************************************************************* */

void   Comm::SetLine(int l)             // new line control setting
{

line = l & ~LCR_DLAB;                   // save new value in instance
OUT(LCR, line);                 // ..and write new LCR
```

```
}


/* ************************************************************************ *
 * Set8n – set 8 data bits, no parity
 * ************************************************************************ */

UINT    Comm::Set8n(void)
{
UINT    oldline;                        // old LCR value


oldline = IN(LCR);                      // get the current LCR value

line = ((oldline & LCR_STOP) | LCR_WLEN);   // setup 8 databits, no parity
OUT(LCR, line);                         // ..and write new LCR

return(oldline);                        // return old LCR value

}


/* ************************************************************************ *
 * InstallInt – install interrupt service routine
 * ************************************************************************ */

void    Comm::InstallInt(void)
{
int     mask;                           // interrupt mask


old_comm = getvect(irq + 8);            // save old comm interrupt rtn
comm = this;                            // ..save instance address
setvect(irq + 8, comm_int);             // ..establish new routine

mask = inportb(I8259M);                 // get current interrupt mask
mask &= ~(1 << irq);                    // determine new mask
outportb(I8259M, mask);                 // ..and put in place

}


/* ************************************************************************ *
 * DeInstallInt – de-install interrupt service routine
 * ************************************************************************ */

void    Comm::DeInstallInt(void)
{
int     mask;                           // interrupt mask
```

```
OUT(IER, 0);                    // disable UART interrupt
OUT(FCR, 0);                    // ..and fifo, if available
setvect(irq + 8, old_comm);     // re-establish old comm rtn

mask = inportb(I8259M);         // get current interrupt mask
mask |= (1 << irq);             // determine new mask
outportb(I8259M, mask);         // ..and turn off comm int

}


/* ********************************************************************* *
 * ModemChanged -- returns TRUE if the modem status register changed
 * ********************************************************************* */

int     Comm::ModemChanged(void)
{
int     rtn;                    // return value


rtn = msr_changed;              // get current status
msr_changed = 0;                // ..and clear flag
return(rtn);                    // ..and return current status

}


/* ********************************************************************* *
 * Modem -- returns the modem status register
 * ********************************************************************* */

int     Comm::Modem(void)
{

return(int_msr);                // return last modem status

}


/* ********************************************************************* *
 * IFlow -- return status of input flow control
 * ********************************************************************* */

int     Comm::IFlow(void)
{

return(i_flow);                 // rtn TRUE if input flow
                                // ..has been restricted
}
```

```
/* ********************************************************************** **
 * Read – read a character, the msr and lsr from the port's input queue
 *
 * Returns: -1 = input queue was empty
 *           0 = character returned
 *           n = number of characters overflowed buffer
 * ********************************************************************** */

UINT    Comm::Read(char *c,                 // queued character
                   char *m,                 // modem status register
                   char *l)                 // line status register
{


if (i_count == 0)                           // q. input queue empty?
   return(-1);                              // a. yes .. return w/err code

*c = *i_get++;                              // get char from input queue
*m = *i_get++;                              // ..then get modem status
*l = *i_get++;                              // ..then get line status
i_count--;                                  // ..finally decrement count

if (i_get > i_limit)                        // q. reached end of buffer?
   i_get = i_buf;                           // a. yes .. set to beginning

if (i_flow && (ICount() <= i_start))        // q. port need restarting?
   RTS(1);                                  // a. yes .. raise RTS line

return(((UCHAR) *l == 0xff) ?               // return with character or
          ((*m << 8) + *c) : 0);  // ..with overflow count
                                            // ..based on lsr flag
}


/* ********************************************************************** **
 * IEmpty – return TRUE if input queue is empty
 * ********************************************************************** */

int     Comm::IEmpty(void)
{

return(i_count == 0) &&                     // TRUE if queue empty
          ((UCHAR) i_put[2] != 0xff));    // ..and not in overflow

}
```

```
/* ********************************************************************** **
 * OEmpty – return TRUE if output queue empty
 * ********************************************************************** */

int     Comm::OEmpty(void)
{

return(o_count == 0);                   // TRUE if queue empty

}



/* ********************************************************************** **
 * ICount – get depth of the input queue
 * ********************************************************************** */

UINT    Comm::ICount(void)
{

return(i_count);                        // return nbr in queue

}

/* ********************************************************************** **
 * OCount – get depth of the output queue
 * ********************************************************************** */

UINT    Comm::OCount(void)
{

return(o_count);                        // return nbr in queue

}



/* ********************************************************************** **
 * IFree – get free space in input queue
 * ********************************************************************** */

UINT    Comm::IFree(void)
{

return(i_size - i_count);               // buffer size less in-use

}
```

```
/* ********************************************************************** **
 * OFree – get free space in output queue
 * ********************************************************************** */

UINT    Comm::OFree(void)
{

return(o_size - o_count);               // buffer size less in-use

}


/* ********************************************************************** **
 * Write – write a character to the output queue
 * ********************************************************************** */

void    Comm::Write(int c)              // character to queue up
{

while (NOT OFree())                     // q. room in output queue?
    ;                                   // a. no .. wait a bit

*o_put++ = c;                           // put char into output queue

if (empty_trans)                        // q. pump need priming?
    {                                   // a. yes .. try to do output
    empty_trans = 0;                    // clear empty tranmitter flag
    o_count++;                          // ..and count character

    if ((NOT flow || IN(MSR) & MSR_CTS)     // q. output flow satisfied?
            && (c = Deque()) != -1)     // ..and queue not empty?
        OUT(THR, c);                    // a. yes .. send char to port
    }

else
    o_count++;                          // else .. count characters

if (o_put > o_limit)                    // q. reach end of buffer?
    o_put = o_buf;                      // a. yes .. set to beginning

}


/* ********************************************************************** *
 * Write – write a string of characters to output queue
 * ********************************************************************** */

void    Comm::Write(char *s)            // string of chars to output
{
```

```
for (; *s;)                    // for each char in the string
    Write(*s++);               // write to the output queue

}



/* ------------------------------------------------------------------ *
 * Write -- write a block of characters to output queue
 * ------------------------------------------------------------------ */

void    Comm::Write(char *s,           // block of chars to output
                    int  l)            // length of block
{

for (; l--;)                   // for each char in the block
    Write(*s++);               // write to the output queue

}



/* ------------------------------------------------------------------ **
 *    IClear -- clear input queue of unread characters
 * ------------------------------------------------------------------ */

void    Comm::IClear(void)
{

_asm    cli                    // stop interrupts
i_get = i_put = i_buf;         // reset buffer pointers
i_count = 0;                   // ..and queue count
_asm    sti                    // ..and re-enable interrupts

}



/* ------------------------------------------------------------------ **
 *    OClear -- clear output queue of unsent characters
 * ------------------------------------------------------------------ */

void    Comm::OClear(void)
{

_asm    cli                    // stop interrupts
o_get = o_put = o_buf;         // reset buffer pointers
o_count = 0;                   // ..and queue count
_asm    sti                    // ..and re-enable interrupts

}
```

```
/* ********************************************************************** *
 * Queue – put a character into the input queue
 * ********************************************************************** */

void    Comm::Queue(int c,              // character to store
                    int m,              // modem status register
                    int l)              // line status register
{

if (flow && (ICount() >= i_stop) &&     // q. flow control needed?
        NOT i_flow)                     // ..and not already on
   RTS(0);                              // a. yes .. clear RTS line

switch (IFree())                        // based on avail queue space
   {
   case 0:                              // full input queue
      if (i_of)                         // q. in overflow?
         {                              // a. yes .. count lost chars
            if (*(UINT *) i_last < 65534U)  // q. within lost count range?
         (*(UINT *) i_last)++;          // a. yes .. tally another one
            }
      else
            {
            i_of = 1;                   // set overflow flag
         *(UINT *) i_last = 2;          // init counter to 2
         i_last[2] = 0xff;              // ..and set error flag in lsr
            }
      break;                            // ..and return to caller

   case 1:                              // almost full
      i_last = i_put;                   // save last saved addr
      i_of = 0;                         // clear overflow flag

   default:                             // from empty to almost full
      i_count++;                        // count characters in queue
      *i_put++ = c;                     // save character in queue
      *i_put++ = m;                     // ..and modem status register
      *i_put++ = l;                     // ..and line status register

      if (i_put > i_limit)              // q. reach end of buffer?
            i_put = i_buf;              // a. yes .. set to beginning
   }
}

/* ********************************************************************** *
 * Deque – get a character from the output queue
 * ********************************************************************** */
```

```
UINT    Comm::Deque(void)
{
char    c;                          // work character


if (o_count == 0)                    // q. output queue empty?
   return(-1);                      // a. yes .. rtn empty handed

c = *o_get++;                        // get char from output queue
o_count--;                          // show character being removed

if (o_get > o_limit)                 // q. reached end of buffer?
   o_get = o_buf;                    // a. yes .. set to beginning

return(c & 0xff);                   // ..and rtn with char to send

}


/* ********************************************************************** **
 *    ~Comm - destructor
 * ********************************************************************** */

Comm::~Comm(void)
{

OUT(MCR, 0);                         // take down DTR and RTS
DeInstallInt();                     // remove interrupt service
delete i_buf;                       // free input
delete o_buf;                       // ..and output queues

}


/* ********************************************************************** *
 * DTR - cycle DTR modem signal
 * ********************************************************************** */

void    Comm::DTR(long t)               // time to hold DTR low
{

OUT(MCR, MCR_DO & ~MCR_DTR);            // set off DTR control line
wait_ms(t);                         // wait a little bit
OUT(MCR, MCR_DO);                    // ..and restore DTR

}
```

```
/* ********************************************************************* *
 * RTS – control RTS modem signal
 * ********************************************************************* */

void    Comm::RTS(int f)                // RTS enable/disable flag
{

OUT(MCR, MCR_DO - (f ? 0 : MCR_RTS));       // set mcr register
i_flow = NOT f;                         // ..and set flow ctl'd flag

}



/* ********************************************************************* **
 *    IntRoutine – communications port interrupt service routine
 * ********************************************************************* */

void    Comm::IntRoutine(void)
{
int     c,                              // output character
            cnt,                        // loop counter
            first_cycle;                // first cycle flag
char    iir,                            // interrupt id register
        lsr;                            // working line status reg


while (((iir = IN(IIR)) & IIR_PEND) == 0)   // while there is work to do
    {
    switch (iir & IIR_II)               // handle each interrupt
        {
            case IIR_MSI:                       // modem status interrupt
        msr_changed = 1;                // set msr changed flag

                if ((int_msr = IN(MSR)) &       // q. modem status register
                    MSR_CTS)            // ..ready for transmits?
                        {
                    if (o_flow ||               // q. flow controlled?
                                        IN(LSR) & LSR_THRE) // ..or transmitter empty?
                            {
                o_flow = 0;             // a. yes .. clear flag

                            if ((c = Deque()) == -1)// q. output queue empty?
                    empty_trans = 1;    // a. yes .. set flag
                else
                                    OUT(THR, c);        // else .. put out a character
                }
                        }
            continue;                   // ..and check next interrupt

            case IIR_LSI:               // line status interrupt
                int_lsr = IN(LSR);      // read line status register
```

```
            continue;                   // ..and check again

        case IIR_TBE:                   // transmitter buffer empty
            cnt = fifo ? 15 : 1;        // set up output fifo size
                first_cycle = 1;            // ..and first cycle flag

            for (; cnt--;)              // loop outputing characters
                                {
                if (flow &&             // q. flow control enabled
                                NOT (IN(MSR) & MSR_CTS))// ..and receiver not ready?
                                {
                    o_flow = first_cycle;   // a. yes .. show flow ctl'd
                                // ..if nothing went out
                                break;              // ..and check next interrupt
                }

                if ((c = Deque()) == -1)    // q. output queue empty?
                    {
                    empty_trans = 1;        // a. yes .. set flag
                                break;              // ..and exit this loop
                    }

                                OUT(THR, c);            // put out another character
                first_cycle = 0;            // ..and clear flag
                                }

            continue;                   // ..and check again

        case IIR_RBF:                   // receiver buffer full
                while ((lsr = IN(LSR)) & LSR_DR)// while data is available
                                Queue(IN(RBR), int_msr,     // ..get and store
                    lsr);               // ..in the input queue

                continue;                   // check for next interrupt
        }
    }

_asm    mov     al, EOI                 // al = end of interrupt cmd
_asm    out     I8259, al               // send EOI to int controller

}



/* ************************************************************** *
 * comm_int() – communications port interrupt service routine header
 * ************************************************************** */

#pragma option -O2-b-e                  // no global reg allocation
                                        // ..or dead code elimination

void    interrupt far comm_int(...)
```

```
{

_asm   sti                          // re-enable interrupts
comm->IntRoutine();                 // use object's interrupt rtn

}

/* ********************************************************************
 *  CheckSum - calculate an 8-bit checksum for message
 * ******************************************************************** */

UINT    Protocol::CheckSum(char *s,         // message data
                           int  l)          // ..and length
{
unsigned
char    acc = 0;                    // accumulator


while (l--)                         // until out of length
  acc += *(unsigned char *) s++;    // ..accumulate checksum

return(acc);                        // return the byte checksum

}


/* ********************************************************************
 *  CRC() - calculate CRC for message
 * ******************************************************************** */

UINT    Protocol::CRC(char *s,          // message data
              int  len)                 // ..and length
{
UINT    acc = 0,                        // crc accumulator
        i;                              // loop control


while (len--)                           // loop thru entire message
  {
  acc = acc ^ (*s++ << 8);              // xor in new byte from msg


  for (i = 0; i++ < 8;)                 // loop thru each of 8 bits
    if (acc & 0x8000)                   // q. this bit on?
          acc = (acc << 1) ^ 0x1021;    // a. yes .. shift and xor
    else
      acc <<= 1;                        // else .. just shift
  }


_asm    mov     ax, acc                 // ax = crc value
_asm    xchg    ah, al                  // swap high and low bytes
return (_AX);                           // ..and return the crc
```

```
}


/* ************************************************************************ *
 * TimeOut - check for an event timeout
 * ************************************************************************ */

int     Protocol::TimeOut(long *n)          // time to wait in ticks
{
static  unsigned
long    far *timer = (unsigned long far *)  // BIOS timer tick counter
                     MK_FP(0x40, 0x6c),     // ..down in low memory
        last,                               // last accessed time
        work;                               // work variable


work = *timer;                              // get current time

if (*n > 0)                                 // q. first time call?
   {
   *n = -*n;                                // a. yes .. change sign
   last = work;                             // ..and initialize counters
   }

if (work != last)                           // q. time pass?
   {                                        // a. yes .. see how much
   if (work <= last)                        // q. clock go past midnite?
      (*n)++;                               // a. yes .. count as 1 tick
   else
      *n += (UINT)(work - last);            // else .. count everything

   last = work;                             // start again w/curr time
   }

return(*n >= 0L);                           // return TRUE at timeout time

}


/* ************************************************************************ *
 * Purge - Purge the line until a timeout occurs
 * ************************************************************************ */

void    Protocol::Purge(Comm *c,            // comm instance to purge
                        long to)            // timeout value in seconds
{
long    timeout;                            // work timeout value
char    ch,                                 // character buffer
        lsr, msr;                           // lsr & msr for read request
```

```
timeout = SECS(to);                    // set the timeout value

for(;;)                                // forever
   {
   if (TimeOut(&timeout))              // q. timeout occur?
      return;                          // a. yes .. exit now

   if (c->Read(&ch, &lsr, &msr) == 0)  // q. any characters?
            timeout = SECS(to);        // a. yes.. restart timeout
   }
}
/* ************************************************************** *
 * wait_for() -- wait for response strings
 * returns: -1 = abort by user
 *           0 = timeout
 *           1 = response 1 found
 *           2 = response 2 found
 * ************************************************************** */

int    wait_for(char *s1,              // response string #1
            char *s2,                  // ..and response string #2
            long t)                    // seconds to wait
{
char   c,                              // work character
       m, l;                           // modem and line status
int    len,                            // string length
       rc = 0;                         // return code
char   *b = 0;                         // work buffer pointer


t = SECS(t);                           // convert seconds to ticks
len = strlen(s1);                      // get first string's length

if (len < strlen(s2))                  // q. second string longer?
   len = strlen(s2);                   // a. yes .. save other's len


b = (char *) malloc_chk(len + 1);      // get a work buffer
memset(b, ' ', len);                   // ..clear to blanks
b[len] = 0;                            // ..and make into a string

while (1)
   {
   if (time_out(&t) || (rc != 0))      // q. timed out waiting?
      {
      free(b);                         // a. yes .. free buffer
      return(rc);                      // ..and return to caller
      }

   if (get_key(NO_ALT) == ESC)         // q. get an ESC key?
      rc = -1;                         // a. yes .. quit waiting
```

```
    if (comm->Read(&c, &m, &l) != 0)        // q. character available?
        continue;                           // a. no .. just loop around

    memmove(b, &b[1], len - 1);             // shift buffer up by one
    b[len - 1] = c;                         // put in new character

    if (strstr(b, s1))                      // q. find response string #1?
            rc = 1;                         // a. yes .. quit and return

    if (strstr(b, s2))                      // q. find response string #2?
        rc = 2;                             // a. yes .. quit and return
    }
}

/* ************************************************************** *
 *  get_time() - retrieve the time in ticks                       *
 * ************************************************************** */

ULONG   get_time(void)                      // retrieve time in ticks
{
static
ULONG   far *timer = (unsigned long far *)  // BIOS timer tick counter
                                MK_FP(0x40, 0x6c);  //  ..down in low memory

return(*timer);                             // return current value
}
/* ************************************************************** *
 *  elapsed_time() - return elapsed time in ticks                 *
 * ************************************************************** */

ULONG   elapsed_time(ULONG st)              // start time in ticks
{
static
ULONG   far *timer = (unsigned long far *)  // BIOS timer tick counter
                    MK_FP(0x40, 0x6c);  //  ..down in low memory

if (st > *timer)                            // q. start time greater?
    return((TICKS_DAY - st) + *timer);      // a. yes .. add in rollover;

else
    return(*timer - st);                    // return elapsed ticks
}
/* ************************************************************** *
 *  time_out() - check for a timeout event                        *
 * ************************************************************** */

int     time_out(long *n)                   // time to wait in ticks
{
static  unsigned
long    far *timer = (unsigned long far *)  // BIOS timer tick counter
                    MK_FP(0x40, 0x6c),  //  ..down in low memory
        last,                               // last accessed time
```

```
        work;                   // work variable


work = *timer;                  // get current time

if (*n > 0)                     // q. first time call?
   {
   *n = -*n;                    // a. yes .. change sign
   last = work;                 // ..and initialize counters
   }

if (work != last)               // q. time pass?
   {                            // a. yes .. see how much
   if (work <= last)            // q. clock go past midnite?
           (*n)++;              // a. yes .. count as 1 tick
   else
           *n += (UINT)(work - last);   // else ..count everything

   last = work;                 // start again w/curr time
   }

return(*n >= 0L);               // return TRUE at timeout time

}
/* ************************************************************** *
 *  wait_ms() - wait in milliseconds
 * ************************************************************** */

void    wait_ms(long ms)        // milliseconds to wait
{

wait((ms + 54) / 55);           // convert then wait in ticks

}
/* ************************************************************** *
 *  malloc_chk() - allocate memory with error processing
 * ************************************************************** */

void    *malloc_chk(int s)      // size to allocate
{
void    *p;                     // work pointer

if ((p = malloc(s)) == 0)       // q. out of memory?
    return(NULL);
//  quit_with(no_memory, s);    // a. yes .. give error msg

return(p);                      // finally rtn with pointer

}
/* ************************************************************** *
 *  wait() - wait for a given number of timer ticks
 * ************************************************************** */
```

```
void    wait(long n)                    // time to wait in ticks
{
long    far *timer = (long far *)       // BIOS timer tick counter
                                        MK_FP(0x40, 0x6c);  // ..down in low memory
        start, work;                    // start tick count

start = *timer;                         // get current time

while (n > 0)                           // loop 'til n ticks past
   {
   if ((work = *timer) != start)        // q. time pass?
      {                                 // a. yes .. see how much
      if (work < start)                 // q. clock go past midnite?
                n--;                    // a. yes .. count as 1 tick
       else
                n -= (UINT)(work - start);      // else .. count everything

      start = work;                     // start again w/curr time
            }

   else
            kbhit();                    // else .. check keyboard
   }
}

/* ************************************************************** *
 * get_key() - get a key (including function keys)
 * ************************************************************** */

int     get_key(int alt_key)            // nonzero = allow alt_key
{
static
int     k;                              // local key variable


if ((k = bioskey(1)) != 0)              // q. key available?
   {                                    // a. yes .. process it
   if (k == -1)                         // q. control break?
            {
      k = 0;                            // a. yes .. clear key,
      wait(1);                          // ..wait a tick, then return
            }
    else
            {
      k = bioskey(0);                   // else .. get waiting key

      if (NOT (k & 0xff))               // q. fnc or extended key?
               k = 0x100 + (k >> 8);            // a. yes .. show special key
       else
          k &= 0xff;                    // else .. force regular key
      }
```

```
  }
else if (alt_key &&                    // q. allowing alt key?
          (_bios_keybrd(_KEYBRD_SHIFTSTATUS) // ..and one pressed?
       & 0x08))
  k = 0x100;                           // a. yes .. special key
else
  k = 0;                               // else .. nothing available

return(k);                             // return w/key if available

}



/* ------------------------------------------------------------------ *
 * List class definition
 * ------------------------------------------------------------------ */

class List
  {
  public:
        List(void);                    // build a list header
        List(char *s,                  // create a single entry
             char *d);                 // ..with string and data
        List(List *l,                  // add to end of list
             char *s,                  // ..string name
             char *d);                 // ..and data
        List(char *s,                  // put at the head of list
             char *d,                  // ..string and data
             List *l);                 // ..list to chain to
        char *Find(char *s);           // find string name
       ~List();                        // destructor

  private:
        void EntryInit(char *s,        // initialize a entry
                       char *d);       // ..with string and data
        char *string_name,             // string name
             *data;                    // ..and data
        List *prev,                    // pervious item pointer
             *next;                    // next item pointer
  };

List   user_commands;                  // define list header for
                                       // ..user's modem commands

/* ------------------------------------------------------------------ *
 * List -- build list header
 * ------------------------------------------------------------------ */

List::List(void)
{
```

```
EntryInit("", "");                    // initialize new instance

}



/* ------------------------------------------------------------ *
 * List -- build a single list entry
 * ------------------------------------------------------------ */

List::List(char *s,                   // string name
           char *d)                   // ..and data
{

EntryInit(s, d);                      // initialize new instance

}



/* ------------------------------------------------------------ *
 * List -- add an entry to the end of the list
 * ------------------------------------------------------------ */

List::List(List *l,                   // list to chain to
           char *s,                   // string name
           char *d)                   // ..and data
{

EntryInit(s, d);                      // build base instance

while (l->next)                       // loop thru..
   l = l->next;                       // ..to the end of the list

l->next = this;                       // put this at the end
this->prev = l;                       // ..and backward chain

}



/* ------------------------------------------------------------ *
 * List -- put an entry at the head of the list
 * ------------------------------------------------------------ */

List::List(char *s,                   // string name
           char *d,                   // ..and data
           List *l)                   // list to chain into
{

EntryInit(s, d);                      // build base instance
```

```
this->prev = l;                    // set up backward link
this->next = l->next;              // ..and forward link
l->next = this;                    // update list anchor

if (this->next)                    // q. any more after us?
   (this->next)->prev = this;      // a. yes .. set up link

}



/* ********************************************************************* *
 *  EntryInit – initialize list entry instance
 * ********************************************************************* */

void List::EntryInit(char *s,              // string name
                     char *d)              // ..and data
{

string_name = data = (char *) 0;           // init string pointers
prev = next = 0;                           // ..and list pointers

if (*s)                                    // q. string name given?
   {
   string_name = new char[strlen(s) + 1];  // a. yes .. get memory and
   strcpy(string_name, s);                 // ..copy for this instance
   }

if (*d)                                    // q. data given
   {
   data = new char[strlen(d) + 1];         // a. yes .. get memory and
   strcpy(data, d);                        // ..copy for this instance
   }
}



/* ********************************************************************* *
 *  Find – find a list entry by the string name
 * ********************************************************************* */

char    *List::Find(char *s)               // string name to search on
{
List *l = this;                            // work pointer

for (;;)                                   // loop thru the list
   {
   if (l->string_name &&                   // q. string available?
         NOT stricmp(s, l->string_name)) // ..and find the entry?
      return(l->data);                     // a. yes .. quit here
```

```
    if ((l = l->next) == 0)          // q. end of list?
        break;                       // a. yes .. exit loop
    }

return(0);                           // else return empty-handed

}



/* ********************************************************************* *
 *  ~List -- object destructor
 * ********************************************************************* */

List::~List()
{

if (string_name)                     // q. string name given?
    delete string_name;              // a. yes .. de-alloc space

if (data)                            // q. data string available?
    delete data;                     // a. yes .. de-alloc space

if (this->next)                      // q. anything after this?
    (this->next)->prev = this->prev;     // a. yes .. de-chain prev

if (this->prev)                      // q. anything before this?
    (this->prev)->next = this->next;     // a. yes .. de-chain next

}

/* ********************************************************************* *
 *
 *  fax_stat() -- FAX status routine
 *
 * ********************************************************************* */
void Fax::fax_stat(int sta,int fn,           // function code
                   struct FxStat *fs)        // pointer to structure

{

switch(fn)                           // based on the function
  {
  case 0:                            // ESC key pressed = TRUE
    break;

  case 1:                            // display a message
    Display(fs->f_msg,sta);              // .. display it
    break;

  case 2:                            // display a message by pointer
    Display(fs->f_ptr,sta);              // .. display it
```

```
        break;

    }

}


/* ------------------------------------------------------------------ *
 *
 *  Display - display a character in a window
 *
 * ------------------------------------------------------------------ */

void    Fax::Display(char c)            // character to display
{

cprintf("%c", c);                       // display string in window

}


/* ------------------------------------------------------------------ *
 *
 *  Display - display string in window
 *
 * ------------------------------------------------------------------ */

void    Fax::Display(char *s,int sta)           // string to display
{
static  int row=70;
//@@@
if(sta == 0){
if(terse_1>terse_0){
memcpy(sndrcve[terse_1+1][0].msg,s,strlen(s));
terse_0=terse_1+1;
}
else {memcpy(sndrcve[terse_0][0].msg,s,strlen(s));
terse_0+=1;
}
}
if(sta == 1){
if(terse_0>terse_1){
memcpy(sndrcve[terse_0+1][1].msg,s,strlen(s));
terse_1=terse_0+1;
}

else { memcpy(sndrcve[terse_1][0].msg,s,strlen(s));
terse_1+=1;
}
}
//@@@
```

```
        switch(sta)
                {
                case 0:
                row+=15;

outtextxy(20,row,s);                    // display string in window
                break;

                case 1:
                row+=15;
outtextxy(400,row,s);                   // display string in window
                break;

                case 2:
                row+=15;
outtextxy(200,row,s);                   // display string in window
                break;

                default:
                break;
                }

}

/* check for and report any graphics errors */
void checkerrors(void)
{
  int errorcode;

  /* read result of last graphics operation */
  errorcode = graphresult();
  if (errorcode != grOk)
  {
    sndl();
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
}


void puthex(int x,int y,char ch[1])
{
int c=0;
unsigned int iu_dl,iu_dh;
char ud[3],bck[1];
bck[0]=ch[0];
iu_dh=(unsigned int)((ch[0]>>4)&0x0F);
iu_dl=(unsigned int)(bck[0]&0x0F);
itoa(iu_dh,ud,16);
```

```
ud[2]='\0';
*(ud)=toupper(*(ud));

for(c=0;c<2;c++){
outtextxy(x,y,ud);
itoa(iu_dl,ud,16);
ud[2]='\0';
*(ud)=toupper(*(ud));
y+=8;
}
}


/*****************************************************************
 *      _main()     Module                                       *
 *****************************************************************/
Fax fax(comm,"2331050");

void   main(int argc, char *argv[])         // command line token count
{
if (argc ==1 || ! strcmp(argv[1], "/?"))
{
clrscr();
printf("      SRY    :0,1,2,3   Fax filename   :Telephone\n");
snd();
exit(0);
}
int gd = DETECT, gm,c,i1,i2,i3,userfont=0,dispflag=1;
char *buff;
//int    gd=9,gm=2,c,userfont=0;
//registerfarbgidriver(<_EGAVGA_fdriver>);
//userfont=installuserfont("litt.chr");


     initgraph(&gd,&gm,"");
     userfont = installuserfont("litt.chr");
     checkerrors();
//     settextstyle(userfont, HORIZ_DIR, 4);

switch(atoi(argv[1]))
{
case 0:
dispflag=1;
     setbkcolor(BLACK);
     setcolor(WHITE);
     rectangle(7,7,632,30);
     rectangle(8,8,631,31);
     setfillstyle(SOLID_FILL,CYAN);
     floodfill(10,10,WHITE);
     rectangle(7,34,318,450);          // left windows display
     rectangle(320,34,633,450);        // right windows display
     setfillstyle(SOLID_FILL,WHITE);
```

```
        floodfill(10,35,WHITE);
        floodfill(321,35,WHITE);
        setcolor(LIGHTBLUE);
    for(c=0 ; c<3; c++)
        {
        line(7,7+c,633,7+c);
        line(7,30+c,633,30+c);
        line(7,450+c,633,450+c);
        line(7+c,7,7+c,450);
        line(633-c,7,633-c,450);
        line(318+c,30,318+c,450);
        }
        setcolor(YELLOW);

    outtextxy(250,15,"SEND PROTOCOL VIEW");
        setcolor(BROWN);
    outtextxy(20,40,"    CALLING STATION                    CALLED STATION");
    outtextxy(20,45,"    ---------------                    ---------------");
        setcolor(GREEN);
    fax.cp = new Comm(0x2f8,3,9600,3,0,4096,512);
    fax.cp->Write(CMD_EXECUTE);          // execute previous command
    wait_ms(250);                        // .. let it execute
    fax.cp->Write(CMD_EXECUTE);          // .. and.. kill it
    fax.cp ->Write(CMD_INIT);            // send modem init string
    fax.cp ->Write(CMD_EXECUTE);         // .. execute init command
    fax.Send(argv[2],argv[3]);           // send a fax
    break;

    case 1:
    dispflag=1;
        setbkcolor(BLACK);
        setcolor(WHITE);
        rectangle(7,7,632,30);
        rectangle(8,8,631,31);
        setfillstyle(SOLID_FILL,CYAN);
        floodfill(10,10,WHITE);
        rectangle(7,34,318,450);         // left windows display
        rectangle(320,34,633,450);       // right windows display
        setfillstyle(SOLID_FILL,WHITE);
        floodfill(10,35,WHITE);
        floodfill(321,35,WHITE);
        setcolor(LIGHTBLUE);
    for(c=0 ; c<3; c++)
        {
        line(7,7+c,633,7+c);
        line(7,30+c,633,30+c);
        line(7,450+c,633,450+c);
        line(7+c,7,7+c,450);
        line(633-c,7,633-c,450);
        line(318+c,30,318+c,450);
        }
        setcolor(YELLOW);
```

```
outtextxy(240,15,"RECEIVE PROTOCOL VIEW");
    setcolor(BROWN);
outtextxy(20,40,"    CALLED  STATION                    CALLING STATION");
outtextxy(20,45,"    ____________                       _____________");
    setcolor(GREEN);
fax.cp = new Comm(0x2f8,3,9600,3,0,4096,512);
fax.cp->Write(CMD_EXECUTE);              // execute previous command
wait_ms(250);                          // .. let it execute
fax.cp->Write(CMD_EXECUTE);              // .. and.. kill it.
fax.cp ->Write(CMD_INIT);               // send modem init string
fax.cp ->Write(CMD_EXECUTE);             // .. execute init command
fax.Receive(argv[2]);
break;

case 2:
dispflag=2;
//
/*    setbkcolor(BLACK);
    setcolor(WHITE);
    rectangle(7,7,632,30);
    rectangle(8,8,631,31);
    setfillstyle(SOLID_FILL,CYAN);
    floodfill(10,10,WHITE);
    rectangle(7,34,318,450);              // left windows display
    rectangle(320,34,633,450);            // right windows display
    setfillstyle(SOLID_FILL,WHITE);
    floodfill(10,35,WHITE);
    floodfill(321,35,WHITE);
    setcolor(LIGHTBLUE);
for(c=0 ; c<3; c++)
    {
    line(7,7+c,633,7+c);
    line(7,30+c,633,30+c);
    line(7,450+c,633,450+c);
    line(7+c,7,7+c,450);
    line(633-c,7,633-c,450);
    line(318+c,30,318+c,450);
    }
    setcolor(YELLOW);
//
    setfillstyle(SOLID_FILL,WHITE);  //@@
    floodfill(10,35,BLACK);          //@@
    floodfill(321,35,BLACK);         //@@
*/
setcolor(WHITE);
outtextxy(250,15,"SEND PROTOCOL VIEW");
    setcolor(BROWN);
outtextxy(20,40,"    CALLING STATION                    CALLED STATION");
outtextxy(20,45,"    _____________                      ____________");
    setcolor(GREEN);
fax.cp = new Comm(0x2f8,3,9600,3,0,4096,512);
```

```
fax.cp->Write(CMD_EXECUTE);              // execute previous command
wait_ms(250);                          // .. let it execute
fax.cp->Write(CMD_EXECUTE);              // .. and.. kill it.
fax.cp ->Write(CMD_INIT);               // send modem init string
fax.cp ->Write(CMD_EXECUTE);             // .. execute init command
fax.Send(argv[2],argv[3]);             // send a fax
break;

case 3:
dispflag=2;
//
/*    setbkcolor(BLACK);
      setcolor(WHITE);
      rectangle(7,7,632,30);
      rectangle(8,8,631,31);
      setfillstyle(SOLID_FILL,CYAN);
      floodfill(10,10,WHITE);
      rectangle(7,34,318,450);          // left windows display
      rectangle(320,34,633,450);        // right windows display
      setfillstyle(SOLID_FILL,WHITE);
      floodfill(10,35,WHITE);
      floodfill(321,35,WHITE);
      setcolor(LIGHTBLUE);
  for(c=0 ; c<3; c++)
      {
      line(7,7+c,633,7+c);
      line(7,30+c,633,30+c);
      line(7,450+c,633,450+c);
      line(7+c,7,7+c,450);
      line(633-c,7,633-c,450);
      line(318+c,30,318+c,450);
      }
      setcolor(YELLOW);
//
      setfillstyle(SOLID_FILL,WHITE);
      floodfill(10,35,BLACK);
      floodfill(321,35,BLACK);
*/
setcolor(WHITE);
outtextxy(240,15,"RECEIVE PROTOCOL VIEW");
      setcolor(BROWN);
outtextxy(20,40,"    CALLED  STATION                    CALLING STATION");
outtextxy(20,45,"    ____________                       ____________");
      setcolor(GREEN);
fax.cp = new Comm(0x2f8,3,9600,3,0,4096,512);
fax.cp->Write(CMD_EXECUTE);              // execute previous command
wait_ms(250);                          // .. let it execute
fax.cp->Write(CMD_EXECUTE);              // .. and.. kill it.
fax.cp ->Write(CMD_INIT);               // send modem init string
fax.cp ->Write(CMD_EXECUTE);             // .. execute init command
fax.Receive(argv[2]);
break;
```

```
default:
break;
}
fax.cp ->~Comm();
OUT(THR,'a');                    // send atz to port
OUT(THR,'t');
OUT(THR,'h');
OUT(THR,'\r');

//********************** HDLC ****************************
cleardevice();
setcolor(WHITE);
switch(dispflag)
{
case 1:
for(c=0;c<20;c++) outtextxy(20,70+20*c,sndrcve[c][0].msg);
for(c=0;c<20;c++) outtextxy(400,70+20*c,sndrcve[c][1].msg);
break;

case 2:
settextstyle(userfont, HORIZ_DIR, 4);
for(i3=2;i3<4;i3++)
for(i2=0;i2<20;i2++){
for(i1=0;i1<strlen(sndrcve[i2][i3].msg);i1++)
puthex((20+300*(i3-2)+6*i1),70+25*i2,&sndrcve[i2][i3].msg[i1]);
}
settextstyle(0,0,2);
break;
default:
break;
}
closegraph();
}
```

กิตติกรรมประกาศ

ขอพระขอบคุณอาจารย์ทุกๆท่านที่ได้ให้ความรู้ และคำแนะนำที่ดี โดยเฉพาะขอขอบพระคุณ อาจารย์ สมยศ จุณณะปิยะ ที่ค่อยให้คำแนะนำและบอกถึงแนวทางแก้ไขปัญหาต่างๆ นอกจากนั้นขอขอบพระคุณ คุณ กิจวัตร ปานดำรง วิศวกรประจำชุมสาย Thaipak การสื่อสารแห่งประเทศไทย ที่คอยให้คำแนะนำ และแก้ปัญหาในการเขียนโปรแกรม ตลอดจนข้อมูลในการค้นคว้า

เอกสารอ้างอิง

FAX:DIGITAL FACSIMILE TECHNOLOGY AND APPLICATION (Second edition)

-Dennis Bodson , Kenneth R. McConnell , Richard Schaphorst

-Artech House Boston London.

ACTIVE HIGH SPEED DATA / FACSIMILE MODEM User's MANUAL

-Quantum Data Systems LTD.

PROGRAMMER 'S TECHNICAL REFERENCE : DATA AND FAX COMMUNICATIONS

-Robert L. Hummel

-Ziff-Davis Press Emeryville , California

PC MAGAZINE C++ COMMUNICATIONS UTILITIES

-Michael Holmes and Bob Flanders

-Ziff-Davis Press Emeryville , California

BORLAND C++ HANDBOOK (Fourth edition)

-Chris H. Pappas & William H. Murray ,III

-Osborne McGraw-Hill