ปีการศึกษา 2531

ข่ายข้อมูลผ่านระบบวิทยุ

(DATA LINK BY RADIO)

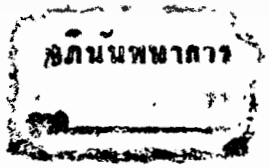
โดย

นาย วิชัย โกเมนตระการ

นาย วิทวัส เคารพธรรม

อาจารย์ที่ปรึกษา

ผศ. ณรงค์ เหมกรณ์



ปริญญานิพนธ์ปีการศึกษา 2531

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ข่ายข้อมูลผ่านระบบวิทยุ

ผู้จัดทำ

1.นาย วิชัย โกเมนตระการ 28.1213

2.นาย วิทวัส เดารพธรรม 28.1219

.....อาจารย์ที่ปรึกษา

(ผศ. ณรงค์ เหมกรณ์)

ข่ายข้อมูลผ่านระบบวิทยุ

วิชัย โกเมนตระการ
วิทวัส เคารพธรรม
ผศ.ณรงค์ เหมกรณ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2531

บทคัดย่อ

ปริญญานิพนธ์เรื่องข่ายข้อมูลผ่านระบบวิทยุนี้ เป็นการนำเอาระบบวิทยุมาประยุกต์ใช้ เพื่อให้อุปกรณ์ประเภทอุปกรณ์ปลายทางข้อมูล (DATA TERMINAL EQUIPMENT (DTE)) สามารถสื่อสารข้อมูลถึงกันได้ ซึ่งมีประโยชน์สำหรับการสื่อสารข้อมูลเป็นระยะทางไกลๆ โดยไม่เกิดปัญหาทางด้านการวางสาย ลักษณะการส่งข้อมูลเป็นการส่งแบบฮาล์ฟดูเพล็กซ์ (HALF DUPLEX) คือทิศทางการส่งข้อมูลเป็นแบบ 2 ทิศทางทั้งไปและกลับในช่วงเวลาที่ต่างกัน ส่วนที่ได้จัดทำขึ้นคือ

1. โมเด็ม (MODEM) ทำหน้าที่แปลงสัญญาณข้อมูล (DATA SIGNAL) ซึ่งมีลักษณะเป็นสัญญาณดิจิตอล (DIGITAL SIGNAL) ให้เป็นสัญญาณเสียงพูด (VOICE SIGNAL) ที่มีลักษณะเป็นสัญญาณอนาล็อก (ANALOG SIGNAL) ทางด้านส่ง และกลับกันทางด้านรับ เพื่อเป็นตัวเชื่อมโยงระหว่างอุปกรณ์ปลายทางข้อมูลกับเครื่องรับส่งวิทยุ

2. โปรแกรมควบคุมการสื่อสารข้อมูล อุปกรณ์ปลายทางข้อมูลที่ใช้คือเครื่องคอมพิวเตอร์ (COMPUTER) จำเป็นต้องมีโปรแกรมเพื่อให้เครื่องคอมพิวเตอร์สามารถทำการสื่อสารข้อมูลระหว่างกันได้

หวังว่าปริญญานิพนธ์นี้ จะสามารถนำไปใช้งานในการสื่อสารข้อมูลได้ และเป็นแนวทางในการศึกษาทางด้านนี้ต่อไป

DATA LINK BY RADIO

Wichai Komentrakarn

Wittawat Khaoroptham

Assistant Professor Narong Haemakorn Advisor

1988

Abstract

"DATA LINK BY RADIO" applies the useful of radio system to transfer data between Data Terminal Equipment (DTE). DTE will be able to communicate with each other in long distance with no problem in cable lying. The typical of signal transmission is Half Duplex (HDX) which can transfer data in bidirectional but not at the same time. The thesis consist of 2 parts.

1. MODEM (MOdulator and DEModulator)

MODEM interfaces between DTE and Radio Transceiver. It modulates data signal to voice signal when tranmits data, and demodulates voice signal to data signal when receives data.

2. Communication Program

DTE used in this thesis is Computer. It must have program to run the Computer for transmitting and receiving data.

Expected that,this thesis will be the useful for data communication and the guideline to study in the future.

สารบัญ

บทที่ 1
บทนำ

การติดต่อสื่อสารข้อมูลระหว่างคอมพิวเตอร์นั้น ปกติสามารถกระทำได้โดยวิธีการเชื่อมโยงสายเข้าด้วยกันโดยตรง ดังจะแบ่งวิธีการส่งตามลักษณะการเดินทางของข้อมูลได้เป็น 2 วิธี คือ

1. การส่งข้อมูลแบบขนาน ข้อมูลจะถูกส่งออกไปทีเดียวทั้งไบท์ (BYTE) โดยบิท (BIT) ทุกบิทที่อยู่ใน 1 ไบท์จะถูกส่งออกไปพร้อมๆกัน ซึ่งจำเป็นที่จะต้องใช้ช่องสัญญาณ (CHANNEL) ในการส่งอย่างน้อยเท่ากับจำนวนบิทใน 1 ไบท์ เช่น อินเตอร์เฟส IEEE-488

2. การส่งข้อมูลแบบอนุกรม ข้อมูลจะถูกส่งออกไปทีละ 1 บิท ช่องสัญญาณในการส่งอาจจะใช้เพียง 1 ช่องสัญญาณเท่านั้น เช่น อินเตอร์เฟส EIA RS-232C อินเตอร์-เฟส EIA RS-422A

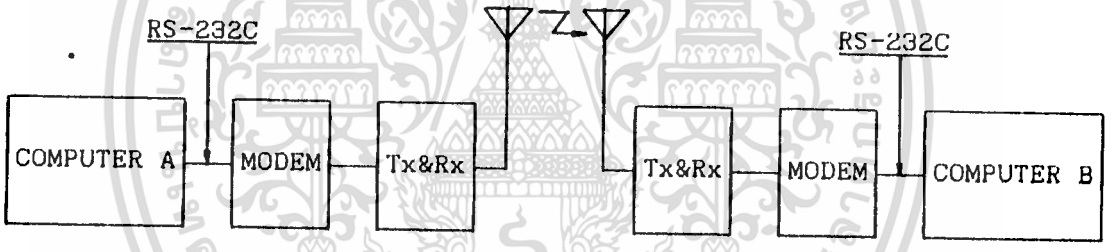
ด้วยเหตุที่ว่า การสื่อสารข้อมูลระหว่างคอมพิวเตอร์โดยใช้สายเชื่อมต่อกันโดยตรงนั้น ไม่สามารถกระทำได้ในระยะทางไกลๆ เนื่องจากสภาพความเป็นตัวเก็บประจุที่เกิดขึ้นในสาย (STRAY CAPACITANCE) เมื่อนำมาใช้ในการส่งข้อมูลที่มีความเร็วสูง อาจจะเกิดการผิดพลาดของข้อมูลขึ้นได้เมื่อถึงปลายทาง ประกอบกับความเจริญก้าวหน้าทางด้านคอมพิวเตอร์ พัฒนามาเป็นโครงข่ายคอมพิวเตอร์ (COMPUTER NETWORK) โดยที่คอมพิวเตอร์อาจจะต้องอาศัยข้อมูลจากคอมพิวเตอร์ตัวอื่นที่อยู่ห่างไกลออกไป จึงจำเป็นต้องหาวิธีอื่น เพื่อให้การติดต่อสื่อสารข้อมูลในระยะทางไกลบรรลุผลได้ วิธีหนึ่งที่เหมาะสมสำหรับโครงข่ายคอมพิว-เตอร์ก็คือ ทำการสื่อสารข้อมูลผ่านสายโทรศัพท์ ทั้งนี้เป็นเพราะว่า สายโทรศัพท์วางเป็นโครงข่ายครอบคลุมพื้นที่ทั่วไปอยู่แล้ว แต่ปัญหาจากการใช้ข่ายสายโทรศัพท์นี้ยังมีอยู่ เนื่องจากช่องสัญญาณโทรศัพท์สามารถส่งผ่านสัญญาณที่มีความถี่สูงสุดได้ประมาณไม่เกิน 4 kHz จึงต้องพยายามทำให้สัญญาณข้อมูลมีแถบกว้างความถี่ (BANDWIDTH) เหมาะสมกับสายโทรศัพท์ โดยใช้วิธีแปลงสัญญาณข้อมูลซึ่งมีลักษณะเป็นสัญญาณดิจิตอลให้เป็นสัญญาณเสียงพูด เรียกวิธีการนี้ว่า โมดูเลชั่น (MODULATION) อุปกรณ์ที่ทำหน้าที่นี้คือ โมเด็ม (MODEM)

การวางโครงข่ายสายโทรศัพท์นั้นยังไม่สามารถวางได้ทั่วถึงทุกจุด ปัญหานี้สามารถแก้ไขได้โดยการนำ"ระบบวิทยุ"เข้ามาช่วย เพราะระบบวิทยุสามารถทำการติดต่อได้เป็นระยะทางไกลๆ โดยไม่มีปัญหาในเรื่องของการวางสาย ระบบวิทยุที่ใช้ในโครงงานนี้เดิมเป็นระบบ

ที่ใช้ในการส่งเสียงพูดอยู่แล้ว จึงต้องทำการแปลงสัญญาณข้อมูลให้เป็นสัญญาณเสียงพูดเช่นเดียวกับระบบโทรศัพท์ โดยอาศัยหลักการเดียวกัน

การส่งข้อมูลผ่านสายโทรศัพท์และระบบวิทยุนั้น จะทำการส่งข้อมูลแบบอนุกรม เพื่อเป็นการประหยัดช่องสัญญาณ เพราะใช้เพียงแค่ 1 หรือ 2 ช่องสัญญาณเท่านั้น

โครงงานนี้เป็นโครงงานการสื่อสารข้อมูลผ่านระบบวิทยุ และทำการแปลงสัญญาณข้อมูลเป็นสัญญาณเสียงพูด โดยการโมดูเลชั่นแบบความถี่ (FREQUENCY MODULATION) ซึ่งในระบบการส่งสัญญาณดิจิตอลเรียกว่า การเข้ารหัสแบบความถี่ (FREQUENCY SHIFT KEYING (FSK) โดยมีส่วนประกอบของโครงงาน ดังแสดงด้วยบล็อกไดอะแกรม (BLOCK DIAGRAM) ในรูปที่ 1.1



รูปที่ 1.1 บล็อกไดอะแกรมของข่ายข้อมูลผ่านระบบวิทยุ

จากรูปที่ 1.1 คอมพิวเตอร์ A ทำการส่งและรับข้อมูลกับคอมพิวเตอร์ B ไปและกลับทั้ง 2 ทิศทางในช่วงเวลาที่ต่างกัน โดยในขณะที่คอมพิวเตอร์ A เป็นฝ่ายส่งข้อมูลนั้น จะส่งข้อมูลออกไปแบบอนุกรมทางอินเตอร์เฟส RS-232C ไปให้กับโมเด็ม จากนั้นโมเด็มจะทำการแปลงสัญญาณข้อมูลให้เป็นสัญญาณเสียงพูด ด้วยวิธีการเข้ารหัสแบบความถี่ คือใช้ความถี่ค่าหนึ่งแทนบิท ø และใช้ความถี่อีกค่าหนึ่งแทนบิท 1 จากนั้นสัญญาณเสียงพูดจะถูกส่งไปให้เครื่องรับส่งวิทยุ ชนิดโมดูเลชั่นแบบความถี่ เพื่อไปโมดูเลทกับคลื่นพาห์ (CARRIER WAVE) ย่านความถี่ 144-146 MHz จากนั้นคลื่นวิทยุที่ถูกโมดูเลทแล้วจะถูกส่งออกผ่านสายอากาศไป เมื่อคลื่นวิทยุเดินทางไปถึงสายอากาศของเครื่องรับส่งวิทยุ ทางด้านคอมพิวเตอร์ B ที่กำลังทำหน้าที่รับข้อมูลอยู่นั้น คลื่นวิทยุจะถูกดีโมดูเลทโดยเครื่องรับส่งวิทยุให้กลายเป็นสัญญาณเสียงพูดเหมือนเดิม หลังจากนั้นสัญญาณเสียงพูดจะส่งไปให้กับโมเด็ม เพื่อเปลี่ยนเป็นสัญญาณข้อมูลอีกทีหนึ่ง แล้วส่งผ่าน

ทางอินเตอร์เฟส RS-232C ไปให้คอมพิวเตอร์ B ทำการประมวลผลต่อไป ในกรณีที่คอมพิว-เตอร์ B ต้องการจะส่งข้อมูลบ้าง จะสามารถทำการส่งได้ก็ต่อเมื่อคอมพิวเตอร์ A หยุดส่งข้อมูลแล้ว ลักษณะการส่งข้อมูลจะคล้ายกันกับที่ได้อธิบายมาแล้วข้างต้นแต่ทิศทางจะกลับกัน โดยคอมพิวเตอร์ B เป็นฝ่ายส่งข้อมูล และคอมพิวเตอร์ A เป็นฝ่ายรับข้อมูลแทน

จากการทำงานข้างบนจะเห็นว่า คอมพิวเตอร์ A และคอมพิวเตอร์ B ทำหน้าที่เป็นอุปกรณ์ปลายทางข้อมูล ส่วนโมเด็มและเครื่องรับส่งวิทยุ จะทำหน้าที่เป็นอุปกรณ์สื่อสารข้อมูล (Data Communication Equipment (DCE)) โดยที่อุปกรณ์อินเตอร์เฟสระหว่างอุปกรณ์ปลายทางข้อมูลกับอุปกรณ์สื่อสารข้อมูลนั้นคือ อินเตอร์เฟส EIA RS-232C

บทที่ 2

ทฤษฎีและหลักการ

2.1 การสื่อสารข้อมูล (DATA COMMUNICATION)

การสื่อสารข้อมูล เป็นการส่งข่าวสารดิจิตอล (DIGITAL INFORMATION) ซึ่งโดยมากอยู่ในรูปของเลขฐานสอง จากแหล่งกำเนิดไปยังจุดหมายปลายทาง ข้อมูลจากแหล่งกำเนิดจะอยู่ในลักษณะสัญญาณดิจิตอล และข้อมูลที่รับได้ก็จะอยู่ในลักษณะสัญญาแดิจิตอลเช่นเดียวกัน ถึงแม้ว่าข้อมูลจะสามารถส่งได้ในลักษณะสัญญาณอนาล็อกหรือสัญญาณดิจิตอลก็ตาม ข่าวสารจากแหล่งกำเนิด อาจจะเป็นรหัสของตัวอักษร ตัวเลขหรือเครื่องหมายที่อยู่ในรูปของเลขฐานสอง เช่น รหัส ASCII หรือรหัส EBCDIC, รหัสของไมโครโปรเซสเซอร์ (MICROPROCESSOR OP-CODES), รหัสควบคุม, รหัสที่อยู่ของผู้ใช้ (USER ADDRESS), โปรแกรมคอมพิวเตอร์, ข่าวสารฐานข้อมูล (DATA BASE INFORMATION)

โครงข่ายการสื่อสารข้อมูล (DATA COMMUNICATION NETWORK) แบบง่ายๆก็เช่น การเชื่อมโยงคอมพิวเตอร์ส่วนบุคคล (PERSONAL COMPUTER) หรือไมโครคอมพิวเตอร์ (MICRO COMPUTER) สองเครื่องในระยะทางใกล้ด้วยสายธรรมดาเข้าด้วยกันโดยตรง แต่ถ้าเป็นระยะทางไกลอาจจะต้องใช้สายโทรศัพท์แทน หรือถ้าจะให้เป็นโครงข่ายการสื่อสารข้อมูลที่ยุ่งยาก อาจจะประกอบด้วยคอมพิวเตอร์เมนเฟรม (MAINFRAME COMPUTER) หนึ่งเครื่องหรือมากกว่า กับอุปกรณ์ปลายทางข้อมูลเป็นจำนวนร้อยๆเครื่อง

2.1.1 รหัสการสื่อสารข้อมูล

ในการที่จะแลกเปลี่ยนข่าวสารระหว่างอุปกรณ์ปลายทางข้อมูลได้นั้น จะต้องมีรหัสที่เป็นมาตรฐานในการเข้ารหัสตัวอักษร ตัวเลข เครื่องหมาย อักขระควบคุม อักขระสื่อสาร ซึ่งรหัสมาตรฐานที่ใช้กันอยู่โดยทั่วๆไป คือ รหัส EBCDIC (Extended Binary Coded Decimal Interchage Code), รหัส ASCII (American Standard Code for Information Interchage) แต่รหัส EBCDIC นั้นใช้ในการสื่อสารข้อมูลเพียงบางแห่งเท่านั้น เนื่องจากรหัส EBCDIC นั้นใน 1 ไบท์จะประกอบด้วย 8 บิท ซึ่งไม่มีที่ว่างเหลือสำหรับบิทพาริตี้ (PARITY BIT) ที่ใช้สำหรับตรวจสอบการผิดพลาดของข้อมูลที่เกิดขึ้นในการสื่อสาร รหัสที่ใช้มากในการสื่อสารข้อมูลก็คือ รหัส ASCII ซึ่งใน 1 ไบท์จะประกอบไปด้วย 7 บิท โดยบิทที่ 8 ใช้เป็นบิทพาริตี้ รหัส ASCII สามารถดูได้จากตารางที่ 2.1

ตารางที่ 2.1 แสดงรหัส ASCII

Bit Number															
							0	0	0	0	1	1	1	1
							0	0	1	1	0	0	1	1
							0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	Row \ Column	0	1	2	3	4	5	6	7
0	0	0	0				0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1				1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0				2	STX	DC2	"	2	B	R	b	r
0	0	1	1				3	ETX	DC3	#	3	C	S	c	s
0	1	0	0				4	EOT	DC4	$	4	D	T	d	t
0	1	0	1				5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0				6	ACK	SYN	&	6	F	V	f	v
0	1	1	1				7	BEL	ETB	'	7	G	W	g	w
1	0	0	0				8	BS	CAN	(	8	H	X	h	x
1	0	0	1				9	HT	EM	)	9	I	Y	i	y
1	0	1	0				10	LF	SS	*	:	J	Z	j	z
1	0	1	1				11	VT	ESC	+	;	K	[	k	{
1	1	0	0				12	FF	FS	,	<	L	\	l	|
1	1	0	1				13	CR	GS	-	=	M	]	m	}
1	1	1	0				14	SO	RS	.	>	N	^	n	~
1	1	1	1				15	SI	US	/	?	O	_	o	DEL

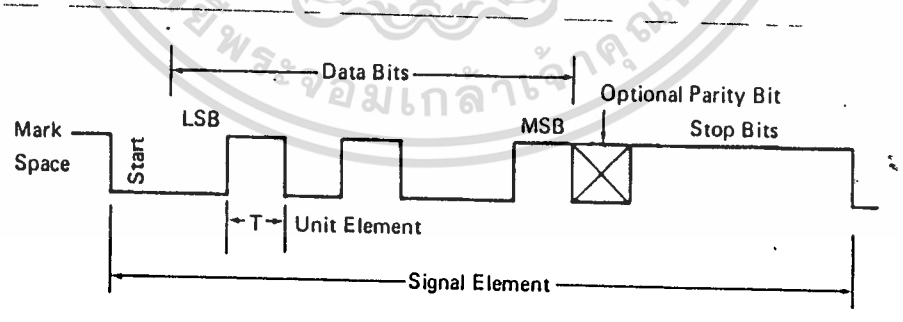
2.1.2 การส่งข้อมูลแบบขนาน และแบบอนุกรม

ในการสื่อสารข้อมูลนั้น ลักษณะการส่งสัญญาณข้อมูล มีได้ 2 ลักษณะคือ

2.1.2.1 การส่งข้อมูลแบบขนาน ลักษณะของการส่งข้อมูลแบบขนาน ทำได้โดยการส่งข้อมูลออกมาทีละ 1 ไบท์ ซึ่งถ้าใน 1 ไบท์มี 8 บิท ทั้ง 8 บิทจะถูกส่งจากอุปกรณ์ส่งไปยังอุปกรณ์รับพร้อมกัน และช่องสัญญาณในการส่งข้อมูลจะต้องมีอย่างน้อย 8 ช่องสัญญาณ โดยมากช่องสัญญาณในการส่งจะใช้สายเคเบิลแบบที่มีตัวนำหลายเส้น อาจจะเป็นสายเคเบิลชนิดแบนหรือกลมก็ได้ โดยที่ระยะทางระหว่าง 2 เครื่องไม่ควรไกลเกินไป เนื่องจากสาเหตุต่างๆหลายสาเหตุ เช่น การที่สัญญาณถูกลดทอนไปกับความต้านทานของสาย การผิดเพี้ยนของสัญญาณเพราะสภาพความเป็นตัวเก็บประจุในสาย การเดินทางมาถึงอุปกรณ์รับไม่พร้อมกันของแต่ละบิทเพราะสภาพความไม่สมดุลย์ของตัวนำแต่ละเส้นในสายเคเบิล และการที่ระดับของกราวนด์ทางไฟฟ้าที่อุปกรณ์รับผิดไปจากอุปกรณ์ส่ง สาเหตุเหล่านี้อาจจะทำให้เกิดการผิดพลาดของข้อมูลขึ้นได้ นอกจากสายที่เป็นทางเดินของข้อมูลแล้ว อาจจะมีทางเดินของสัญญาณควบคุมต่างๆอีก เช่น สายที่ใช้ควบคุมการโต้ตอบ (HAND SHAKE) ซึ่งการส่งข้อมูลแบบขนานจะเห็นได้จากการส่งข้อมูลจากคอมพิวเตอร์ไปยังเครื่องพิมพ์เป็นส่วนใหญ่

2.1.2.2 การส่งข้อมูลแบบอนุกรม ลักษณะของการส่งข้อมูลแบบอนุกรมนั้น ข้อมูลจะถูกส่งออกมาทีละบิทจากอุปกรณ์ส่งไปยังอุปกรณ์รับ ช่องสัญญาณในการส่งข้อมูลอาจจะใช้เพียง 1 หรือ 2 ช่องสัญญาณเท่านั้น ทำให้ค่าใช้จ่ายในการสื่อสารถูกกว่าแบบขนาน แต่อัตราการส่งข้อมูลจะล่าช้ากว่าการส่งแบบขนาน การส่งข้อมูลแบบอนุกรม ข้อมูลที่ต้องการส่งซึ่งจะอยู่ในลักษณะเป็นไบท์จะถูกทยอยส่งทีละบิท และทางอุปกรณ์รับจะต้องรับข้อมูลเข้ามาทีละบิทแล้วมารวมกันเป็นไบท์ ซึ่งทางด้านอุปกรณ์รับจะต้องคอยตรวจสอบว่า บิทใดเป็นบิทเริ่มแรกของไบท์นั้น การตรวจสอบขึ้นอยู่กับรูปแบบของบิทที่ทยอยส่งจากอุปกรณ์ส่ง ซึ่งมีอยู่ 2 รูปแบบคือ

ก. การส่งแบบอะซิงโครนัส (ASYNCHRONOUS TRANSMISSION) ในการส่งแบบอะซิงโครนัส ข้อมูลที่ต้องการส่งแต่ละตัวหรือแต่ละไบท์นั้นจะมีสัญญาณสำหรับตรวจสอบบิทแรกภายในตัวของมันเอง โดยแต่ละไบท์จะถูกเพิ่มด้วยบิทเริ่มต้น (START BIT) นำหน้าไบท์นั้น และบิทสิ้นสุด (STOP BIT) ตามหลังไบท์นั้น ซึ่งอาจจะมีการเพิ่มบิทพาริตี้ก่อนบิทสิ้นสุดก็ได้ ระยะเวลาระหว่างข้อมูลแต่ละไบท์ไม่จำเป็นต้องแน่นอน เพราะที่อุปกรณ์รับจะตรวจสอบบิทแรกทีละไบท์เท่านั้น โดยขณะไม่มีการส่งข้อมูลสภาพลอจิกจะเป็น "1" อุปกรณ์รับจะคอยตรวจสอบจากการเปลี่ยนลอจิก "1" เป็นลอจิก "Ø" (บิทเริ่มต้นมีลอจิกเป็น "Ø") ซึ่งหมายถึงบิทที่ตามมาหลังจากบิทเริ่มต้นก็คือบิทแรกของไบท์นั้น รูปแบบการจัดเรียงบิทในการส่งแบบอะซิงโครนัสแสดงดังรูปที่ 2.1



รูปที่ 2.1 แสดงการจัดเรียงบิทในแต่ละไบท์ของการส่งแบบอะซิงโครนัส

ข. การส่งแบบซิงโครนัส (SYNCHRONOUS TRANSMISSION) ในการส่งแบบซิงโครนัส ข้อมูลจะถูกจัดเป็นขบวนที่มีจำนวนไบท์ที่แน่นอน โดยแต่ละไบท์จะอยู่ติดกัน ขบวนของข้อมูลนี้จะถูกนำด้วยอักขระเริ่มต้น ซึ่งอักขระเริ่มต้นนี้ อุปกรณ์รับจะใช้สำหรับตรวจสอบว่าบิทแรกของไบท์แรกอยู่ที่ใด การส่งแบบซิงโครนัสประสิทธิภาพจะสูงกว่าแบบอะซิงโครนัสมาก

เพราะไม่มีบิทเริ่มต้นและบิทสิ้นสุดในแต่ละไบท์ มีแต่อักขระเริ่มต้นเพียงไม่กี่ไบท์เท่านั้น

2.1.3 ทิศทางการสื่อสารข้อมูล

ช่องสัญญาณ อาจจะตีความหมายได้ว่า ช่องทางหนึ่งช่องทางบนสายที่สัญญาณไฟฟ้าสามารถไหลผ่านได้ ส่วนสายนั้นหมายถึง ส่วนประกอบของระบบที่เชื่อมโยงระหว่างสถานีสื่อสารสองสถานี ทิศทางการสื่อสารข้อมูลในช่องสัญญาณสามารถแบ่งได้เป็น 3 ลักษณะคือ

2.1.3.1 ซิมเพล็กซ์ (SIMPLEX) หมายถึง การส่งข้อมูลในทิศทางเดียวในการสื่อสารแบบนี้อุปกรณ์สื่อสารด้านหนึ่ง จะส่งข้อมูลไปในช่องสัญญาณเท่านั้น และจะไม่รับข้อมูลจากช่องสัญญาณ แต่อุปกรณ์สื่อสารอีกด้านหนึ่งจะรับข้อมูลจากช่องสัญญาณเท่านั้น และจะไม่ส่งข้อมูลไปในช่องสัญญาณเลย

2.1.3.2 ฮาล์ฟดูเพล็กซ์ (HALF DUPLEX) หมายถึงการสื่อสารข้อมูลใน 2 ทิศทาง แต่ในช่วงเวลาหนึ่งได้เพียงทิศทางเดียวเท่านั้น อุปกรณ์สื่อสารทั้ง 2 ด้านจะผลัดกันรับผลัดกันส่ง การสื่อสารแบบนี้ส่วนใหญ่แล้วจะใช้ระบบสาย 2 เส้น (2W)

2.1.3.3 ฟูลดูเพล็กซ์ (FULL DUPLEX) หมายถึงการสื่อสารข้อมูลใน 2 ทิศทางพร้อมกัน การสื่อสารแบบนี้ใช้ได้ทั้งระบบสาย 2 เส้นและสาย 4 เส้น (4W) แต่ในระบบสาย 2 เส้น จะต้องอาศัยเทคนิคการแบ่งความถี่เข้าช่วย คือจะส่งในความถี่ช่วงหนึ่ง และจะรับในความถี่อีกช่วงหนึ่ง

2.2 โมเด็ม

การที่จะทำให้อุปกรณ์ปลายทางข้อมูล เช่น คอมพิวเตอร์ สามารถทำการสื่อสารข้อมูลเป็นระยะทางไกลๆได้ เป็นการสะดวกที่จะอาศัยโครงข่ายของช่องสัญญาณเสียง (VOICE-BANDWIDTH CHANNEL) ที่กระจายครอบคลุมโดยทั่วไปอยู่แล้ว เช่น ระบบโทรศัพท์ ระบบวิทยุ ในการที่จะส่งสัญญาณข้อมูลที่มีลักษณะเป็นสัญญาณดิจิตอลไปในช่องสัญญาณอนาล็อก ที่ส่วนใหญ่จะมีแถบกว้างความถี่ (BANDWIDTH) 300 ถึง 3400 เฮิรท์ซ (HERTZ) จำเป็นที่จะต้องนำเอาสัญญาแข้อมูลไปโมดูเลทกับคลื่นพาห์ความถี่เสียง (VOICE FREQUENCY CARRIER SIGNAL) ให้กลายเป็นสัญญาณเสียงพูดที่เครื่องส่ง และทำการดีโมดูเลทสัญญาณเสียงพูดให้กลับเป็นสัญญาณข้อมูลอีกครั้งที่เครื่องรับ อุปกรณ์ที่ทำการโมดูเลทและดีโมดูเลทนี้เรียกว่า โมเด็ม

คลื่นพาห์ความถี่เสียงจะถูกโมดูเลทโดยการเปลี่ยนแปลงขนาด (AMPLITUDE) หรือความถี่ (FREQUENCY) หรือเฟส (PHASE) เป็นสัญญาณเสียงพูดเพื่อส่งเข้าไปในช่องสัญญาณ

ซึ่งการเปลี่ยนแปลงนี้ขึ้นอยู่กับลอจิกของแต่ละบิทของข้อมูล โดยที่การเลือกชนิดของการโมดูเลชั่นนั้นส่วนใหญ่จะขึ้นอยู่กับอัตราการส่งข้อมูล ดังจะกล่าวถึงต่อไปนี้

2.2.1 การโมดูเลชั่นแบบขนาด (AMPLITUDE MODULATION) หรือในการส่งสัญญาณข้อมูลจะเรียกว่า การเข้ารหัสแบบขนาด (AMPLITUDE SHIFT KEYING (ASK)) คลื่นพาห์ความถี่เสียงจะมีขนาดค่าหนึ่งสำหรับลอจิก "1" และจะมีขนาดอีกค่าหนึ่งหรืออาจจะเป็นศูนย์สำหรับลอจิก "0" การเข้ารหัสแบบขนาดนี้ไม่ค่อยนิยมใช้เพราะจะถูกรบกวนจากสัญญาณรบกวนได้ง่าย ใช้สำหรับส่งข้อมูลที่มีความเร็วต่ำเท่านั้น

2.2.2 การโมดูเลชั่นแบบความถี่ (FREQUENCY MODULATION) หรือในการส่งสัญญาณข้อมูลเรียกว่า การเข้ารหัสแบบความถี่ (FREQUENCY SHIFT KEYING (FSK)) ความถี่ของคลื่นพาห์จะถูกเปลี่ยนไปตามลอจิก คือมีความถี่ค่าหนึ่งสำหรับลอจิก "1" และจะมีความถี่อีกค่าหนึ่งสำหรับลอจิก "0"

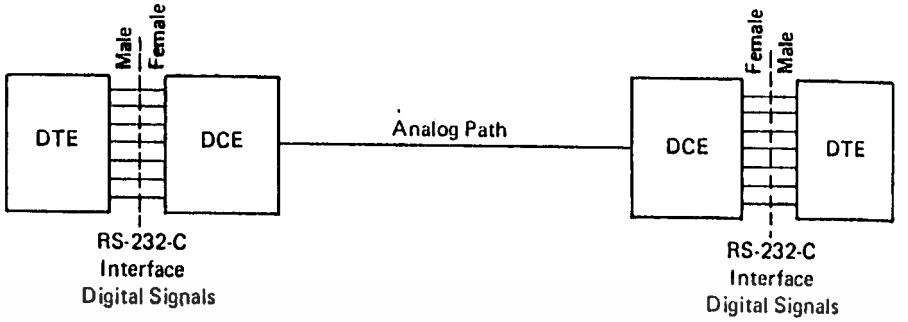
2.2.3 การโมดูเลชั่นแบบเฟส (PHASE MODULATION) หรือในการส่งสัญญาณข้อมูลเรียกว่า การเข้ารหัสแบบเฟส (PHASE SHIFT KEYING (PSK)) เฟสของคลื่นพาห์จะมีค่าหนึ่งสำหรับลอจิก "1" และจะมีอีกค่าหนึ่งสำหรับลอจิก "0" การเข้ารหัสแบบเฟสนี้เหมาะสำหรับการส่งข้อมูลที่มีความเร็วสูง

นอกจากนี้แล้วยังสามารถอาศัยทั้งการเข้ารหัสแบบขนาด และการเข้ารหัสแบบเฟสสำหรับการส่งข้อมูลที่มีความเร็วสูงมากได้อีก

โมเด็มจัดเป็นอุปกรณ์สื่อสารข้อมูล การอินเตอร์เฟสเข้ากับอุปกรณ์ปลายทางข้อมูลส่วนใหญ่จะอาศัยอินเตอร์เฟสที่เป็นมาตรฐานเช่น อินเตอร์เฟส RS-232C

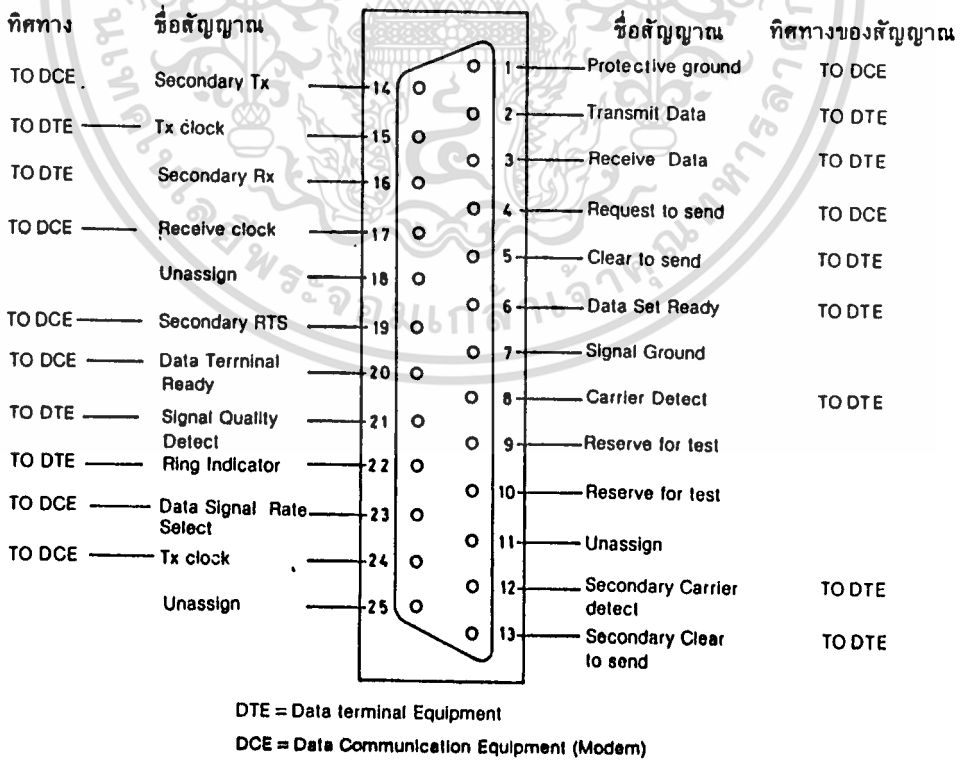
2.3 อินเตอร์เฟส EIA RS-232C

อินเตอร์เฟส RS-232C เป็นอินเตอร์เฟสมาตรฐานที่ใช้ในการเชื่อมต่อระหว่างอุปกรณ์ปลายทางข้อมูล กับอุปกรณ์สื่อสารข้อมูล หรืออาจจะเชื่อมต่อระหว่างอุปกรณ์ปลายทางข้อมูลเองก็ได้ ซึ่งพัฒนาขึ้นมาโดยสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิคส์ของอเมริกา (Electronic Industries Association (EIA)) เพื่อใช้ในการติดต่อสื่อสารข้อมูลแบบไบนารี่ (BINARY DATA) แบบอนุกรม ดังแสดงในรูปที่ 2.2



รูปที่ 2.2 การใช้ RS-232C เชื่อมระหว่างอุปกรณ์ปลายทางข้อมูลกับอุปกรณ์สื่อสารข้อมูล

อินเตอร์เฟส RS-232C ส่วนใหญ่มีลักษณะของข้อต่อและขาเป็นแบบ DB-25 (D-TYPE 25 PIN CONNECTOR) ดังแสดงในรูปที่ 2.3 ถึงแม้ว่าตามมาตรฐานจะไม่ได้กำหนดกฎเกณฑ์ที่แน่นอนถึงลักษณะของข้อต่อและขาก็ตาม



รูปที่ 2.3 การกำหนดข้อต่อแบบ DB-25 ที่ใช้สำหรับ RS-232C

หน้าที่และระดับสัญญาณของขาสัญญาณต่างๆของอินเตอร์เฟส RS-232C ที่อุปกรณ์ปลายทางข้อมูล (DTE) กับอุปกรณ์สื่อสารข้อมูล (DCE) ใช้ติดต่อระหว่างกันมีดังนี้

PROTECTIVE GROUND (PG ขาที่ 1) เป็นขาที่ต่อกับตัวถัง (FRAME) ของอุปกรณ์ ซึ่งโดยปกติแล้วจะต่อกับกราวด์ภายนอก (EXTERNAL GROUND) และต่ออยู่กับชีลด์ของสายเคเบิ้ลที่เชื่อมต่อระหว่าง DTE กับ DCE เพื่อป้องกันไม่ให้ไฟฟ้าสถิตย์และสนามแม่เหล็กใดๆ ที่เกิดขึ้น มีผลกระทบต่อสายสัญญาณต่างๆ แต่การต่อกับตัวถังนั้นควรจะต่อกับตัวถังเพียงด้านใดด้านหนึ่งเท่านั้น

TRANSMITTED DATA (TD ขาที่ 2) เป็นขาที่ใช้ส่งสัญญาณข้อมูลแบบอนุกรมจาก DTE ไปให้ DCE หรือต่อไปให้ DTE ตัวอื่น โดยในขณะที่ไม่มีสัญญาแส่งออกจะมีสถานะเป็น "1"

RECEIVED DATA (RD ขาที่ 3) เป็นขาที่ใช้รับสัญญาณข้อมูลจาก DCE หรือจาก DTE ตัวอื่น เมื่อไม่มีสัญญาณเข้ามาจะมีสถานะเป็น "1" และในกรณีที่ส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ ขาสัญญาณนี้จะต้องเป็น "1" เสมอเมื่อ RTS อยู่ในสถานะ "ON"

REQUEST TO SEND (RTS ขาที่ 4) เป็นขาที่ DTE ใช้ส่งสัญญาณควบคุมไปให้ DCE เพื่อเป็นการบอก DCE ว่า DTE ต้องการที่จะส่งข้อมูลออกไปแล้ว ให้ DCE เตรียมตัวรับข้อมูลจาก DTE ด้วย

CLEAR TO SEND (CTS ขาที่ 5) เป็นขาที่ DCE ใช้ส่งสัญญาณควบคุมไปยัง DTE เพื่อแจ้งให้ DTE รู้ว่า DCE พร้อมที่จะรับข้อมูลจาก DTE เพื่อทำการส่งออกแล้ว

DATA SET READY (DSR ขาที่ 6) เป็นขาที่ DCE ใช้ส่งสัญญาแควบคุมไปแจ้งให้ DTE ทราบว่า ขณะนี้โมเด็มต่อเข้ากับช่องสัญญาณของโทรศัพท์เรียบร้อยแล้ว

SIGNAL GROUND (SG ขาที่ 7) ทำหน้าที่เป็นระดับแรงดันอ้างอิงสำหรับทุกๆวงจรของสายสัญญาณทุกสาย ยกเว้น PROTECTIVE GROUND

DATA TERMINAL READY (DTR ขาที่ 20) เป็นขาที่ DTE ใช้แสดงให้รู้ว่าพร้อมที่จะทำการส่งหรือรับข้อมูลแล้ว ซึ่ง DTR จะต้องอยู่ในสถานะ "ON" ก่อนที่ DCE จะมีสถานะของ DSR เป็น "ON"

RING INDICATOR (RI ขาที่ 22) เป็นขาที่ DCE จะให้สถานะเป็น "ON" ออกมา เมื่อ DCE ได้รับสัญญาณ RINGING แต่ขานี้จะไม่ทำงานถ้า DTR มีสถานะภาพเป็น "OFF"

RECEIVED LINE SIGNAL DETECTOR (RLSD หรือ CD ขาที่ 8) เป็นขาสัญญาณควบคุมซึ่ง DCE แจ้งให้ DTE ทราบว่าตัวมันได้รับสัญญาณคลื่นพาห์ (CARRIER SIGNAL) จาก DCE ฝ่ายส่งแล้ว ขา RLSD จะมีสถานะ "OFF" เมื่อ RTS อยู่ในสถานะ "ON"

SIGNAL QUALITY DETECTOR (SQ ขาที่ 21) สัญญาณที่ขานี้เป็นขาที่ DCE ใช้แสดงว่า สัญญาณข้อมูลที่ได้รับมามีโอกาสเกิดการผิดพลาดสูงหรือไม่ ถ้าเป็นสถานะ "ON" แสดงว่าโอกาสเกิดการผิดพลาดจะต่ำ แต่ถ้ามีสถานะ "OFF" แสดงว่ามีโอกาสสูงที่จะรับข้อมูลผิดพลาด เพราะ DCE จะตรวจสอบกับสัญญาณคลื่นพาห์ว่ามีคุณภาพดีหรือไม่ มีสัญญาณรบกวนมากหรือเปล่า

DATA SIGNAL RATE SETECTOR (ขาที่ 23) เป็นขาสัญญาณที่ใช้ควบคุมอัตราเร็วในการส่งข้อมูล ทำให้สามารถส่งข้อมูลด้วยอัตราเร็ว 2 อัตราได้ เมื่ออยู่ในสถานะ "ON" อัตราเร็วในการส่งข้อมูลที่สูงกว่าจะถูกเลือกใช้ ขาสัญญาณนี้สามารถควบคุมได้จาก DTE หรือ DCE ขึ้นอยู่กับการนำไปใช้งาน

TRANSMITTER SIGNAL ELEMENT TIMING (TSET ขาที่ 24) เป็นขาที่ DTE ใช้ในการส่งสัญญาณ เพื่อไปควบคุมจังหวะการทำงาน (CLOCKING SIGNAL) ของ DCE ในการส่งข้อมูล

TRANSMITTER SIGNAL ELEMENT TIMING (TSET ขาที่ 15) เป็นขาที่ DCE ใช้ส่งสัญญาณ เพื่อไปควบคุมจังหวะการทำงานของ DTE ในการส่งข้อมูล สัญญาณควบคุมจังหวะการทำงานนี้จะมีใช้เฉพาะใน ซิงโครนัสโมเด็ม (SYNCHRONUS MODEM) เท่านั้น

RECEIVER SIGNAL ELEMENT TIMING (RSET ขาที่ 17) เป็นขาที่ DCE ใช้ส่งสัญญาณ เพื่อไปควบคุมจังหวะการทำงานของ DTE ในการรับข้อมูล

SECONDARY TRANSMITTED DATA (STD ขาที่ 14)

SECONDARY RECEIVED DATA (SRD ขาที่ 16)

SECONDARY REQUEST TO SEND (SRTS ขาที่ 19)

SECONDARY CLEAR TO SEND (SCTS ขาที่ 13)

SECONDARY RECEIVED LINE SIGNAL DETECTOR (SRLSD ขาที่ 12)

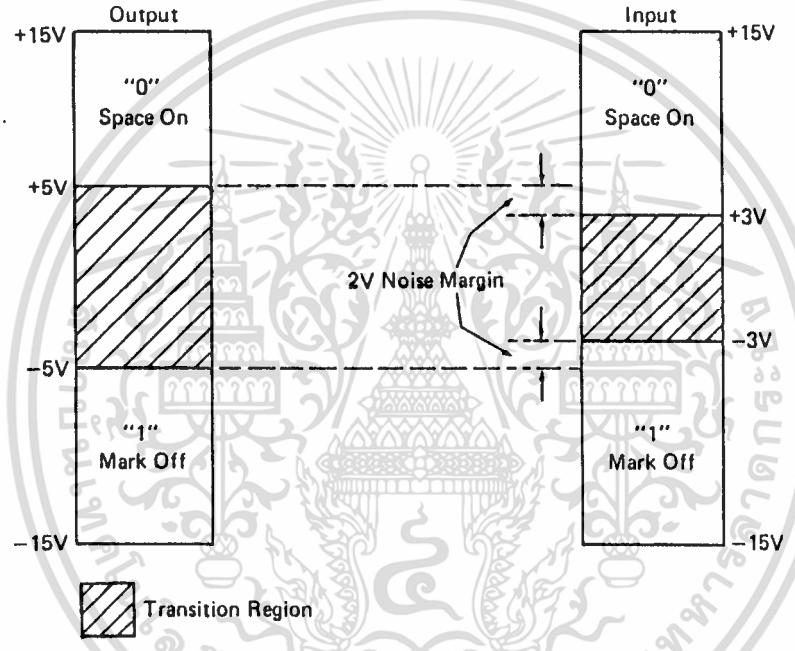
ขาสัญญาณ SECONDARY ทั้ง 5 ขานี้ จะใช้สำหรับช่องสัญญาณช่องที่ 2 (SECONDARY CHANNEL) ในลักษณะเช่นเดียวกับช่องสัญญาณที่ 1 แต่ช่องสัญญาณที่ 2 นี้จะมีทิศทางตรงกันข้ามกับช่องสัญญาณที่ 1 และช่องสัญญาณที่ 2 นี้จะอยู่ภายใต้การควบคุมของขา

สัญญาณ DTR และ DSR ด้วย

ตารางที่ 2.2 แสดงคีย์พารามิเตอร์ (KEY PARAMETERS) ตามข้อกำหนดของ EIA RS-232C

CHARACTERISTICS	EIA RS-232C
FORM OF OPERATION	SINGLE-ENDED
MAX. CABLE LENGTH	15 m
MAX. DATA RATE	20 kbps
DRIVER OUTPUT VOLTAGE,OPEN CIRCUIT	±25 V (max.)
DRIVER OUTPUT VOLTAGE,LOADED CIRCUIT	±5 TO ±15 V (min.)
DRIVER OUTPUT RESISTANCE,POWER OFF	R_o = 300 ohm (min.)
DRIVER OUTPUT SHORT CIRCUIT CURRENT I_{sc}	±500 mA (max.)
DRIVER OUTPUT SLEW RATE	30 V/microsec (max.)
RECEIVER INPUT RESISTANCE R_{in}	3 TO 7 kohm
RECEIVER INPUT THRESHOLDS	-3 TO +3 V (max.)
RECEIVER INPUT VOLTAGE	-25 TO +25 V (max.)
INTERCONNECTING CABLE TYPE	TWISTED-PAIR WIRE OR FLAT CABLE CONDUCTOR PAIR
CONDUCTOR SIZE:	
COPPER WIRE (SOLID OR STRANDED)	24 AWG OR LARGER
OTHER (RESISTANCE PER CONDUCTOR)	R ⩽ 10 ohm/100 m
CAPACITANCE:	
MUTUAL PAIR	C ⩽ 66 pF/m
STRAY	C ⩽ 130 pF/m
PAIR-TO-PAIR CROSSTALK (BALANCED)	A ⩾ 40 dB
ATTENUATION AT 150 kHz	

ระดับสัญญาณทางไฟฟ้าที่ขาต่างๆของอินเตอร์เฟส RS-232C เมื่อเทียบกับระดับกราวนด์อ้างอิง (SIGNAL GROUND) แล้ว จะต้องอยู่ในระดับไม่เกิน ±25 โวลท์ และเมื่อเกิดการช็อตกันระหว่างขาต่างๆ จะต้องไม่เกิดความเสียหายขึ้นกับอุปกรณ์ใดๆที่ต่ออยู่กับขาของอินเตอร์เฟส RS-232C รวมทั้งตัวมันเองด้วย ลักษณะระดับสัญญาณทางไฟฟ้าที่เป็นสัญญาณเอาท์-พุท (OUTPUT) และสัญญาณอินพุท (INPUT) จะมีลักษณะดังแสดงในรูปที่ 2.4 โดยมีความหมายของแต่ละระดับสัญญาณดังแสดงในตารางที่ 2.3



รูปที่ 2.4 ลักษณะระดับสัญญาณทางไฟฟ้าของ อินเตอร์เฟส RS-232C

ตารางที่ 2.3 ความหมายของแต่ละระดับสัญญาณทางไฟฟ้าของ อินเตอร์เฟส RS-232C

ขาสัญญาณข้อมูลส่ง (TD) และขาสัญญาณข้อมูลรับ (RD)		
BINARY STATE	"1"	"0"
SIGNAL CONDITION	MARK	SPACE
VOLTAGE	NEGATIVE	POSITIVE
ขาสัญญาณควบคุมต่างๆ		
CONTROL FUNCTION	OFF	ON
VOLTAGE	NEGATIVE	POSITIVE

ข้อจำกัดของสลูว์เรท เกี่ยวเนื่องมาจากปัญหาของครอสทอล์ค (CROSSTALK) ระหว่างตัวนำแต่ละเส้นในสายเคเบิ้ล ถ้าสัญญาณมีการเปลี่ยนแปลงเร็วมากเท่าไหร่ ก็ยิ่งจะมีครอสทอล์คมากขึ้นเท่านั้น อุปกรณ์ต่างๆที่นำมาใช้งานร่วมกับอินเตอร์เฟส RS-232C ไม่จำเป็นต้องมีมาตรฐานตามข้อบังคับของ EIA ก็ได้ขึ้นอยู่กับความต้องการของผู้ใช้และผู้ผลิต อินเตอร์เฟส RS-232C สามารถใช้งานได้กับการส่งสัญญาณที่ความเร็วสูงถึง 20 kbps โดยที่ระยะห่างระหว่างอุปกรณ์ปลายทางข้อมูลกับอุปกรณ์สื่อสารข้อมูล อาจจะยาวเกินกว่า 50 ฟุต หรือ 15 เมตรก็ได้ ถ้าสภาพความเป็นตัวเก็บประจุที่จุดอินเตอร์เฟสมีค่าไม่เกิน 2500 pF

2.4 อะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัส (Asynchronous Communications Adapter)

อะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัสที่ใช้ในคอมพิวเตอร์ส่วนบุคคล IBM เป็นอะแดปเตอร์ที่สร้างขึ้นมาเพื่อใช้สำหรับการสื่อสารข้อมูลแบบอะซิงโครนัสเท่านั้น ตามมาตรฐานของอินเตอร์เฟส EIA RS-232C อะแดปเตอร์นี้สามารถที่จะเพิ่มบิทเริ่มต้น บิทสิ้นสุด บิทพาริตี้ได้ โดยที่สามารถจะควบคุมการทำงานของอะแดปเตอร์ จากการโปรแกรมผ่านคอมพิวเตอร์ ซึ่งสามารถที่จะควบคุมอัตราเร็วในการส่งข้อมูลจาก 50 bps ถึง 9600 bps จำนวนบิทใน 1 ไบท์ตั้งแต่ 5 ถึง 8 บิท จำนวนบิทสิ้นสุด 1, 1.5 หรือ 2 บิท

หัวใจในการทำงานของอะแดปเตอร์นี้คือ IC INS8250 LSI การทำงานของอะแดปเตอร์โดยการโปรแกรมข้อมูลผ่านคอมพิวเตอร์ทาง I/O ตำแหน่งที่ H&3F8 ถึง H&3FF สำหรับอะแดปเตอร์ตัวที่ 1 และทาง I/O ตำแหน่งที่ H&2F8 ถึง H&2FF สำหรับอะแดปเตอร์ตัวที่ 2 เพื่อไปควบคุมการทำงานของ 8250 บิท A0, A1 และ A2 บนตำแหน่ง I/O ใช้สำหรับเลือกรีจิสเตอร์ (REGISTER) ต่างๆบน 8250 เพื่อเลือกรูปแบบการทำงานของ 8250 และบิทสำหรับเลือกการหาร (DIVISOR LATCH ACCESS BIT (DLAB)) ซึ่งเป็นบิท 7 บนรีจิสเตอร์ควบคุมสาย (LINE CONTROL REGISTER) ก็ใช้สำหรับเลือกรีจิสเตอร์ด้วย ลักษณะการเลือกรีจิสเตอร์บน 8250 ทาง I/O ตำแหน่งต่างๆ สามารถดูได้จากตารางที่ 2.4 และสถานะของบิทต่างๆในรีจิสเตอร์เมื่อซามาสเตอร์รีเซท (MASTER RESET) ของ 8250 ถูกเปลี่ยนเป็นลอจิก "1" สามารถดูได้จากตารางที่ 2.5

ตารางที่ 2.4 แสดงการเลือกรีจิสเตอร์บน 8250 ทาง I/O ตำแหน่งต่างๆ

I/O ตำแหน่งที่ H&3F8 ถึง H&3FF และตำแหน่งที่ H&2F8 ถึง H&2FF											
A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	DLAB	REGISTER
1	1/0	1	1	1	1	1	X	X	X		
							0	0	0	0	RECEIVE BUFFER (READ), TRANSMIT HOLDING REG. (WRITE)
							0	0	1	0	INTERRUPT ENABLE
							0	1	0	X	INTERRUPT IDENTIFICATION
							0	1	1	X	LINE CONTROL
							1	0	0	X	MODEM CONTROL
							1	0	1	X	LINE STATUS
							1	1	0	X	MODEM STATUS
							1	1	1	X	NONE
							0	0	0	1	DIVISOR LATCH (LSB)
							0	0	1	1	DIVISOR LATCH (MSB)

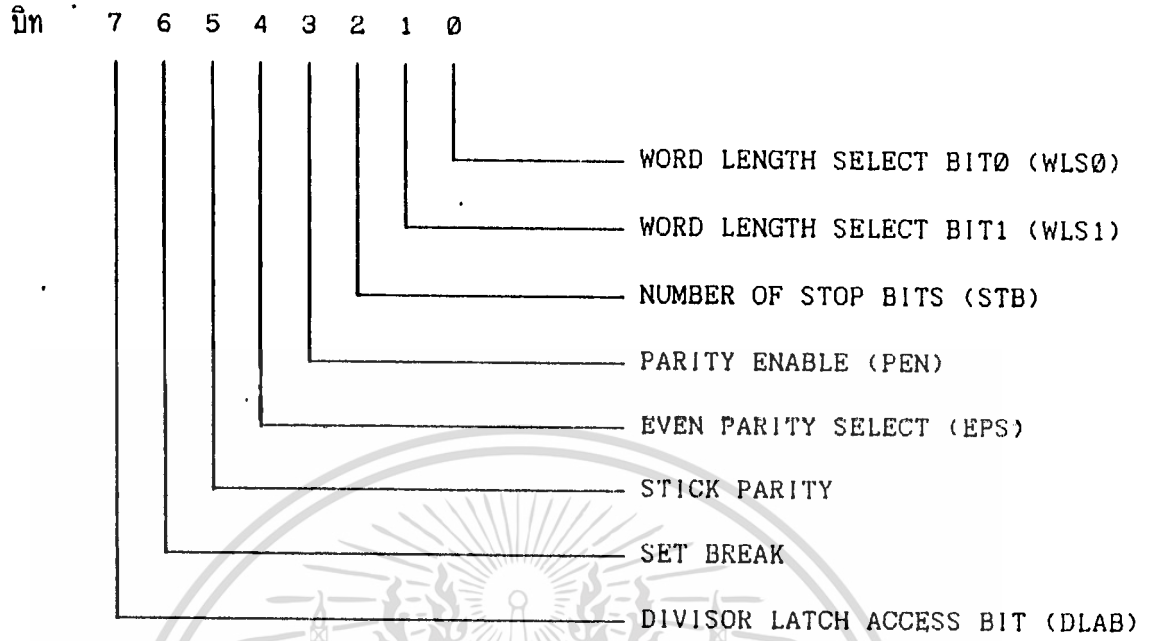
หมายเหตุ บิท 8 ถ้าเป็นลอจิก "1" หมายถึงการเลือกอะแดปเตอร์ตัวที่ 1 แต่ถ้าเป็นลอจิก "0" หมายถึงการเลือกอะแดปเตอร์ตัวที่ 2

ตารางที่ 2.5 สถานะของบิทต่างๆในรีจิสเตอร์บน 8250 เมื่อขามาสเตอร์รีเซทถูกรีเซท

REGISTER	สถานะ
RECEIVE BUFFER (READ)	คงเดิม
TRANSMIT HOLDING REG.(WRITE)	คงเดิม
INTERRUPT ENABLE	ทุกบิทเป็น "0" (บิท 0-3 ถูกรีเซท, บิท 4-7 เป็น "0" ตลอด)
INTERRUPT IDENTIFICATION	บิท 0 เป็น "1",บิท 1-2 เป็น "0", บิท 3-7 เป็น "0" ตลอด
LINE CONTROL	ทุกบิทเป็น "0"
MODEM CONTROL	ทุกบิทเป็น "0"
LINE STATUS	บิท 0-4,7 เป็น "0",บิท 5-6 เป็น "1"
MODEM STATUS	บิท 0-3 เป็น "0", บิท 4-7 ขึ้นอยู่กับสัญญาณอินพุท
DIVISOR LATCH	คงเดิม

2.4.1 รีจิสเตอร์ควบคุมสาย (LINE CONTROL REGISTER)

การจัดรูปแบบบิทของข้อมูลในการรับส่งแบบอะซิงโครนัส กระทำได้โดยการโปรแกรมไปบนรีจิสเตอร์ควบคุมสาย ซึ่งอาจจะต้องเรียกข้อมูลบนรีจิสเตอร์ควบคุมสายมาตรวจสอบดูก่อน เพื่อจะได้จัดรูปแบบบิทของข้อมูลในการรับส่งได้ถูกต้อง รีจิสเตอร์ควบคุมสายประกอบไปด้วยบิทต่างๆดังแสดงในรูปที่ 2.5



รูปที่ 2.5 แสดงบิทต่างๆบนรีจิสเตอร์ควบคุมสาย

ซึ่งบิทต่างๆมีหน้าที่ในการจัดรูปแบบบิทของข้อมูลในการรับส่งดังต่อไปนี้

บิท 0 และบิท 1 สองบิทนี้ใช้กำหนดจำนวนบิทในแต่ละไบท์ของข้อมูลในการรับส่ง ดังตารางที่ 2.6

ตารางที่ 2.6 การกำหนดจำนวนบิทในแต่ละไบท์ของบิท 0 และบิท 1

บิท 1	บิท 0	จำนวนบิทในแต่ละไบท์
0	0	5 บิท
0	1	6 บิท
1	0	7 บิท
1	1	8 บิท

บิท 2 บิทนี้ใช้กำหนดจำนวนบิทสิ้นสุดของข้อมูลในการรับส่ง ถ้าบิท 2 นี้เป็น "Ø" บิทสิ้นสุดจะเป็น 1 บิททั้งในการส่งและตรวจสอบในการรับ และถ้าบิท 2 เป็น "1" บิทสิ้นสุดจะเป็น 1.5 บิท สำหรับข้อมูลความยาว 5 บิท และจะเป็น 2 บิท สำหรับข้อมูลความยาว 6,7 และ 8 บิท

บิท 3 บิทนี้ใช้กำหนดการมีบิทพาริตี้หรือไม่ ถ้าบิทนี้เป็น "1" จะมีการสร้างบิทพาริตี้ที่ฝ่ายส่งและจะมีการตรวจสอบบิทพาริตี้ที่ฝ่ายรับ โดยบิทพาริตี้นี้จะอยู่ระหว่างบิทสุดท้ายของข้อมูลกับบิทสิ้นสุด

บิท 4 เป็นบิทที่ใช้กำหนดชนิดของบิทพาริตี้ ถ้าบิทนี้เป็น "Ø" บิทพาริตี้จะเป็นพาริตี้คี่ (ODD PARITY) และถ้าเป็น "1" จะเป็นพาริตี้คู่ (EVEN PARITY)

บิท 5 ถ้าบิท 3 เป็น "1" และบิท 5 เป็น "1" ด้วย บิทพาริตี้จะเป็น "Ø" กรณีบิท 4 เป็น "1" และบิทพาริตี้จะเป็น "1" กรณีบิท 4 เป็น "Ø"

บิท 6 ถ้าบิท 6 เป็น "1" สัญญาณข้อมูลอนุกรมที่ส่งออกจะเป็น "Ø" ตลอดเวลา โดยไม่สนใจการกระทำของฝ่ายตรงข้ามว่ากำลังทำการส่งข้อมูลหรือไม่ และสามารถล้มเลิกสภาวะนี้ได้เมื่อให้บิท 6 กลับเป็น "Ø"

บิท 7 เป็นบิทสำหรับเลือกการหาร ถ้าบิท 7 เป็น "1" จะสามารถโปรแกรมให้เขียนและอ่านจากรีจิสเตอร์ตัวหาร (DIVISOR LATCHES REGISTER) ได้ แต่ถ้าบิท 7 เป็น "Ø" จึงจะสามารถโปรแกรมให้ติดต่อกับรีจิสเตอร์บัฟเฟอร์รับ (RECEIVER BUFFER REGISTER) รีจิสเตอร์รักษาข้อมูลส่ง (TRANSMITTER HOLDING REGISTER) รีจิสเตอร์อินเตอร์รัปเอ็นเอเบิ้ล (INTERRUPT ENABLE REGISTER)

2.4.2 ตัวกำหนดอัตราเร็วในการรับส่ง
(PROGRAMMABLE BAUD RATE GENERATOR)

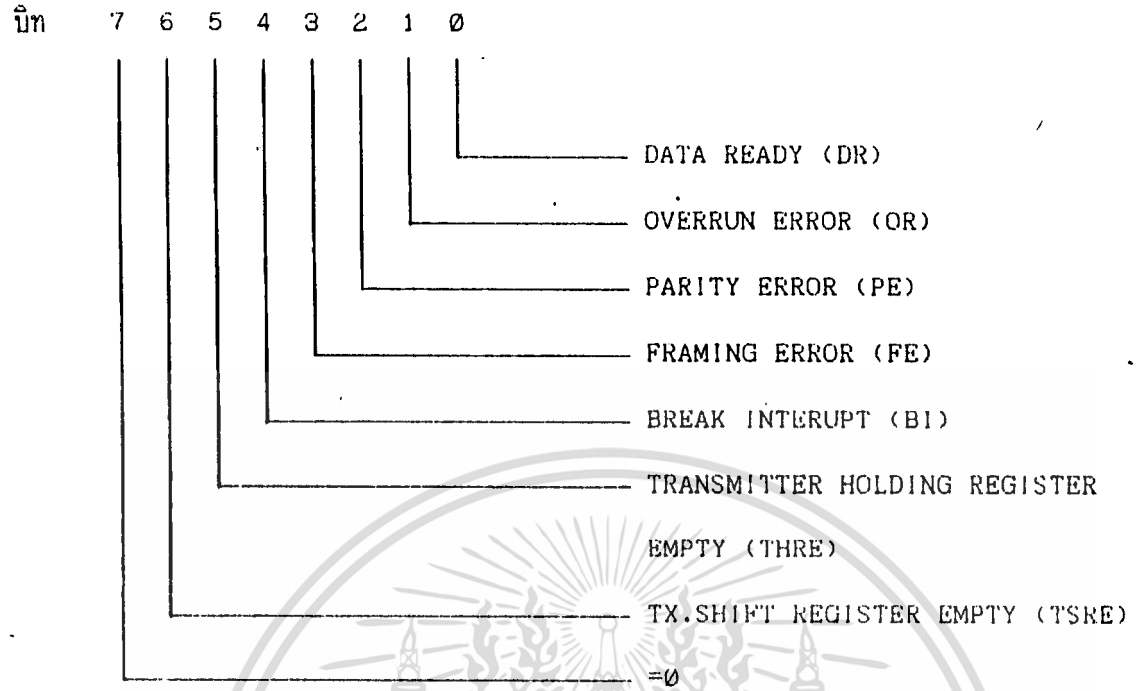
ใน 8250 จะมีตัวกำหนดอัตราเร็วในการรับส่งอยู่ซึ่งสามารถโปรแกรมได้ โดยตัวกำหนดอัตราเร็วจะนำเอาสัญญาณนาฬิกา (1.8432 MHz) มาหารด้วยตัวหารที่มีค่าจาก 1 ถึง $2^{16}-1$ ความถี่ที่ออกจากตัวกำหนดอัตราเร็วจะมีค่าเป็น 16 เท่าของอัตราเร็วในการรับส่งข้อมูล ซึ่งจะมีรีจิสเตอร์ตัวหารขนาด 8 บิท 2 ตัวต่อกันเป็นเลขฐานสอง 16 บิท ค่าของตัวหารนี้ควรจะถูกป้อนเข้าไปในรีจิสเตอร์ตัวหารตอนเริ่มแรกของการทำงาน เพื่อให้แน่ใจว่าอัตราเร็วในการรับส่งข้อมูลถูกต้องเป็นไปตามที่ต้องการ ค่าตัวหารสามารถดูได้จากตารางที่ 2.7

ตารางที่ 2.7 ค่าตัวหารในการกำหนดอัตราเร็วในการรับส่งข้อมูล

อัตราเร็วที่ต้องการ	ค่าตัวหาร (เลขฐานสิบ)	ค่าตัวหาร (เลขฐานสิบหก)	เปอร์เซ็นต์การผิดพลาด
50	2304	900	-
75	1536	600	-
110	1047	417	0.026
134.5	857	359	0.058
150	768	300	-
300	384	180	-
600	192	0C0	-
1200	96	060	-
1800	64	040	-
2000	58	03A	0.69
2400	48	030	-
3600	32	020	-
4800	24	018	-
7200	16	010	-
9600	12	00C	-

2.4.3 รีจิสเตอร์บอกสถานะสาย (LINE STATUS REGISTER)

รีจิสเตอร์บอกสถานะสาย เป็นรีจิสเตอร์ที่บอกสถานะของการรับส่งข้อมูล ประกอบไปด้วยบิทต่างๆดังแสดงในรูปที่ 2.6



รูปที่ 2.6 แสดงบิทต่างๆบนรีจิสเตอร์บอกสถานะสาย

บิท Ø เป็นบิทที่บอกถึงความพร้อมของข้อมูลที่รับเข้ามา บิท Ø จะเป็น "1" เมื่อข้อมูลที่รับเข้ามาถูกเก็บเข้าไปในรีจิสเตอร์บัฟเฟอร์รับเรียบร้อยแล้ว และ บิท Ø อาจจะถูกรีเซทเป็น "Ø" เมื่อคอมพิวเตอร์ได้อ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์รับแล้ว หรืออาจจะถูกรีเซทโดยคอมพิวเตอร์เองก็ได้

บิท 1 เป็นบิทที่บอกว่า คอมพิวเตอร์ไม่ได้อ่านข้อมูลตัวที่แล้วไปจากรีจิสเตอร์บัฟเฟอร์รับ ก่อนที่ข้อมูลตัวถัดมาถูกเก็บเข้าไปในรีจิสเตอร์บัฟเฟอร์รับ นั่นหมายถึงข้อมูลตัวที่แล้วได้หายไป ซึ่งบิท 1 จะถูกรีเซทเมื่อคอมพิวเตอร์ได้อ่านข้อมูลจากรีจิสเตอร์บอกสถานะสายแล้ว

บิท 2 เป็นบิทที่บอก การผิดพลาดของบิทพาริตี้ของข้อมูลที่รับเข้ามา บิท 2 จะเป็น "1" เมื่อตรวจพบการผิดพลาดของบิทพาริตี้ และจะเป็น "Ø" เมื่อคอมพิวเตอร์ได้อ่านข้อมูลจากรีจิสเตอร์บอกสถานะสายแล้ว

บิท 3 เป็นบิทที่บอกว่า บิทสิ้นสุดของข้อมูลที่รับเข้ามาไม่ถูกต้องจากการตรวจพบว่าบิทสิ้นสุดเป็น "Ø"

บิท 4 เป็นบิทที่บอกว่าสายข้อมูลรับมีสถานะเป็น "Ø" ยาวนานเกิน

กว่าเวลาของข้อมูลหนึ่งตัว (บิทเริ่มต้น+บิทข้อมูล+บิทพาริตี้+บิทสิ้นสุด)

บิท 5 เมื่อรีจิสเตอร์รักษาข้อมูลส่งว่างลง แสดงว่า 8250 พร้อมที่จะรับข้อมูลตัวใหม่สำหรับการส่งแล้ว ทำให้บิท 5 มีสถานะเป็น "1" แต่จะถูกรีเซทเป็น "0" เมื่อคอมพิวเตอร์ได้ส่งข้อมูลไปบนรีจิสเตอร์รักษาข้อมูลส่งแล้ว

บิท 6 เมื่อรีจิสเตอร์ข้อมูลส่ง (TRANSMITTER SHIFT REGISTER) ว่างลง บิท 6 จะกลายเป็น "1" ซึ่งจะถูกรีเซทเป็น "0" เมื่อมีการถ่ายข้อมูลจากรีจิสเตอร์รักษาข้อมูลส่งไปยังรีจิสเตอร์ข้อมูลส่ง บิท 6 นี้จะเป็นบิทสำหรับอ่านเท่านั้น

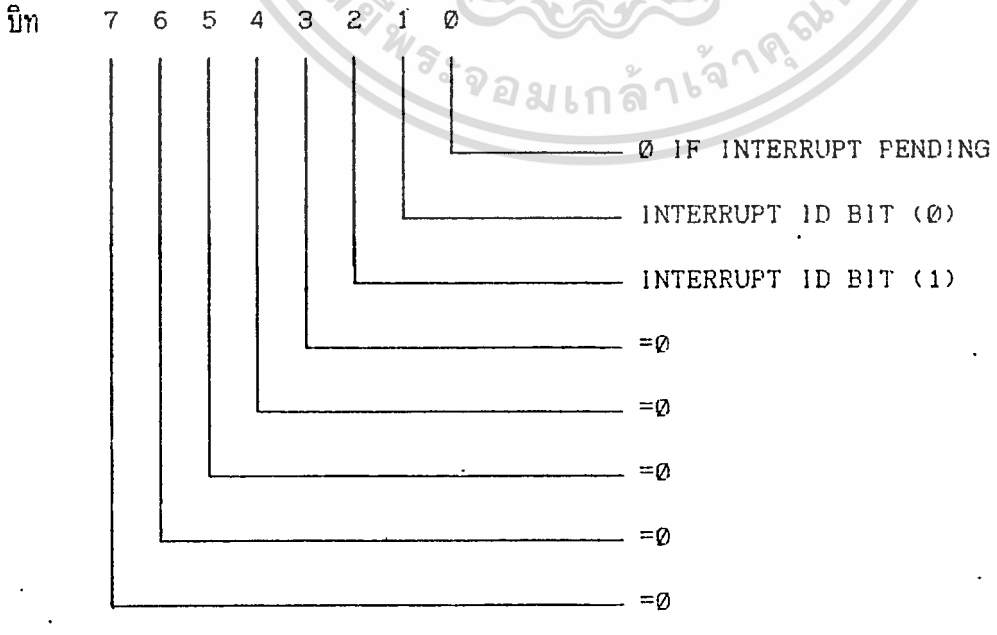
บิท 7 บิทนี้จะเป็น "0" ตลอด

2.4.4 รีจิสเตอร์แสดงการอินเตอร์รัป

(INTERRUPT IDENTIFICATION REGISTER)

การอินเตอร์รัปของ 8250 ทำให้สามารถเชื่อมโยง 8250 เข้ากับไมโครโปรเซสเซอร์ที่ใช้อยู่โดยทั่วไปได้ ซึ่งระดับการอินเตอร์รัปของ 8250 ถูกจัดเป็น 4 ระดับตามความสำคัญจากสาเหตุต่างๆคือ ระดับที่ 1 จากสถานะสายตัวรับ ระดับที่ 2 จากความพร้อมของข้อมูลรับ ระดับที่ 3 จากรีจิสเตอร์รักษาข้อมูลส่งว่างลง ระดับที่ 4 จากสถานะโมเด็ม

รีจิสเตอร์แสดงการอินเตอร์รัปบอกให้ทราบว่ามีการอินเตอร์รัปรออยู่ และจะบอกชนิดของการอินเตอร์รัประดับสูงสุดด้วย ซึ่งประกอบด้วยบิทต่างๆดังรูปที่ 2.7



รูปที่ 2.7 แสดงบิทต่างๆบนรีจิสเตอร์แสดงการอินเตอร์รัป

บิท 0 ถ้ามีการอินเตอร์รัปรออยู่บิท 0 นี้จะเป็น "0"

บิท 1 และบิท 2 จะบอกถึงชนิดของการอินเตอร์รัประดับสูงสุดซึ่งสามารถดูชนิดของการอินเตอร์รัปได้จากตารางที่ 2.8

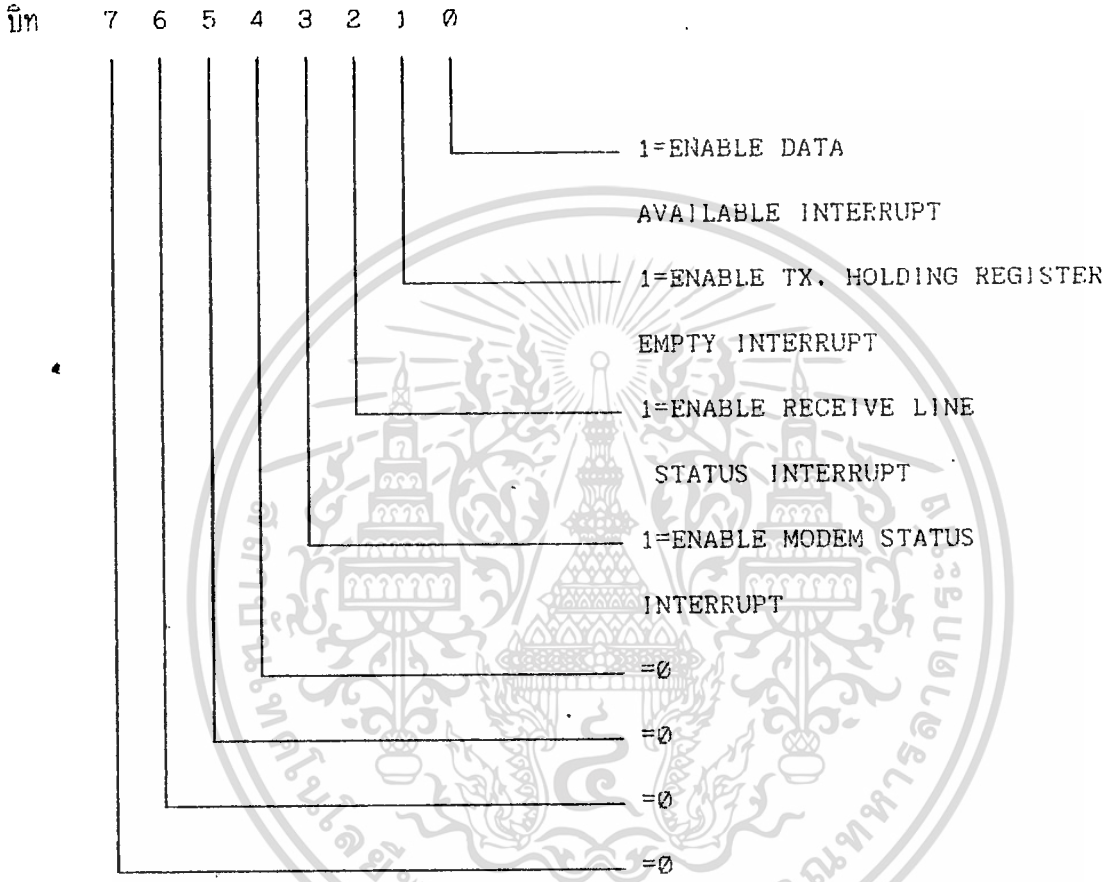
บิท 3 ถึงบิท 7 จะเป็น "0" ตลอด

ตารางที่ 2.8 แสดงชนิดของการอินเตอร์รัป

บิท2	บิท1	บิท0	ระดับ	ชนิด	สาเหตุ	การรีเซท
0	0	1		-	-	
1	1	0	สูงสุด	สถานะสายตัวรับ	ข้อมูลรับหายไป บิทพาริตี้ผิดพลาด บิทสิ้นสุดผิดพลาด สายข้อมูลรับเป็น"0" ตลอด	อ่านข้อมูลจาก รีจิสเตอร์บอกสถานะสาย
1	0	0	2	ข้อมูลรับพร้อม	ข้อมูลรับพร้อม	อ่านข้อมูลจาก รีจิสเตอร์บัฟเฟอร์รับ
0	1	0	3	รีจิสเตอร์รักษา ข้อมูลส่งว่าง	รีจิสเตอร์รักษา ข้อมูลส่งว่าง	อ่านข้อมูลจากรีจิสเตอร์ แสดงการอินเตอร์รัป เขียนข้อมูลไปบนรีจิสเตอร์ รักษาข้อมูลส่ง
0	0	0	4	สถานะโมเด็ม	สัญญาณ CTS สัญญาณ DSR สัญญาณ RI สัญญาณ RLSD	อ่านข้อมูลจาก รีจิสเตอร์บอกสถานะโมเด็ม

2.4.5 รีจิสเตอร์อินเตอร์รัปเอ็นเอเบิ้ล (INTERRUPT ENABLE REGISTER)

รีจิสเตอร์อินเตอร์รัปเอ็นเอเบิ้ล เป็นรีจิสเตอร์ที่ใช้สำหรับควบคุมการอินเตอร์รัปชนิดต่างๆว่าจะให้มีการทำงานหรือไม่ ประกอบไปด้วยบิทต่างๆดังรูปที่ 2.8



รูปที่ 2.8 แสดงบิทต่างๆบนรีจิสเตอร์อินเตอร์รัปเอ็นเอเบิ้ล

บิท 0 บิทนี้ควบคุมให้มีการอินเตอร์รัป จากความพร้อมของข้อมูลรับ เมื่อบิท 0 เป็น "1"

บิท 1 บิทนี้ควบคุมให้มีการอินเตอร์รัป จากการว่างของรีจิสเตอร์รักษาข้อมูลส่ง เมื่อบิท 1 เป็น "1"

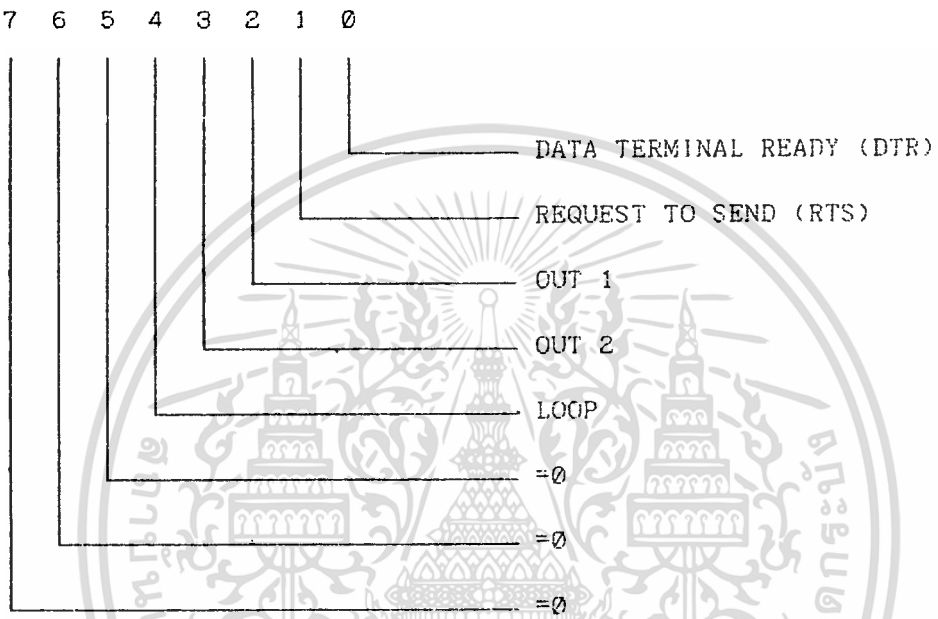
บิท 2 บิทนี้ควบคุมให้มีการอินเตอร์รัปจากสถานะสายรับ เมื่อบิท 2 เป็น "1"

บิท 3 บิทนี้ควบคุมให้มีการอินเตอร์รัปจากสถานะโมเด็ม เมื่อบิท 3

เป็น "1"

2.4.6 รีจิสเตอร์ควบคุมโมเด็ม (MODEM CONTROL REGISTER).

รีจิสเตอร์ควบคุมโมเด็ม เป็นรีจิสเตอร์ที่ใช้ควบคุมการอินเตอร์เฟสกับโมเด็ม หรืออุปกรณ์ข้อมูลอื่นๆ ประกอบไปด้วยบิทต่างๆดังรูปที่ 2.9



รูปที่ 2.9 แสดงบิทต่างๆบนรีจิสเตอร์ควบคุมโมเด็ม

บิท 0 ถ้าบิทนี้เป็น "1" ขาสัญญาณ DTR จะเป็น "0" และถ้าเป็น "0" ขาสัญญาณ DTR จะเป็น "1"

บิท 1 บิทนี้ใช้ควบคุมขาสัญญาณ RTS ในลักษณะเช่นเดียวกับที่ได้อธิบายในกรณีบิท 0

บิท 2 บิทนี้ใช้ควบคุมขาสัญญาณ OUT1 ในลักษณะเช่นเดียวกับที่ได้อธิบายในกรณีบิท 0 ซึ่งเป็นขาสัญญาณที่มีเผื่อให้ผู้ใช้ใช้งานได้ตามต้องการ

บิท 3 บิทนี้ใช้ควบคุมขาสัญญาณ OUT2 ในลักษณะเช่นเดียวกับที่ได้อธิบายในกรณีบิท 0 ซึ่งเป็นขาสัญญาณที่มีเผื่อให้ผู้ใช้ใช้งานได้ตามต้องการ

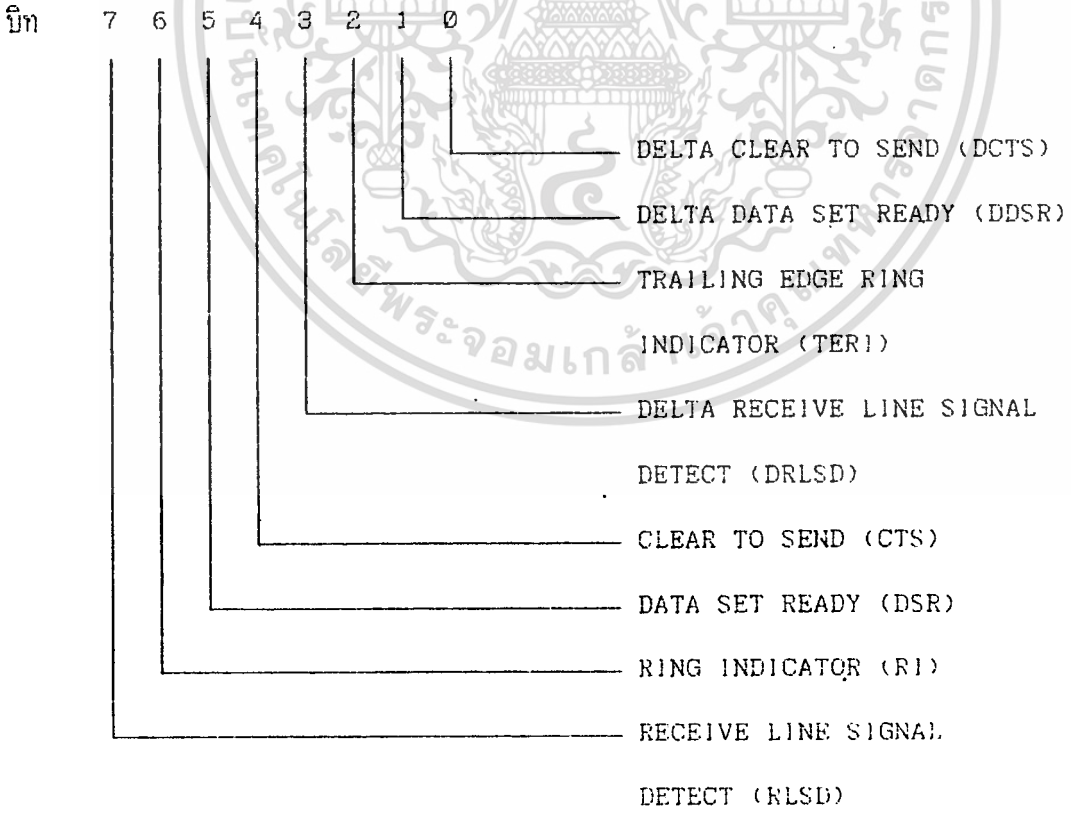
บิท 4 บิทนี้ใช้ควบคุมการป้อนข้อมูลกลับเพื่อใช้ทดสอบ 8250 เมื่อบิท 4 นี้เป็น "1" ขาสัญญาณส่งข้อมูลอนุกรมจะเป็น "1" ขาสัญญาณรับข้อมูลอนุกรมจะถูกเปิด

วงจร ข้อมูลจากรีจิสเตอร์ข้อมูลส่งจะถูกป้อนกลับให้รีจิสเตอร์ข้อมูลรับ (RECEIVER SHIFT REGISTER) ขาอินพุทสัญญาณควบคุมโมเด็มภายนอก (CTS,DSR,RLSD และ RI) จะถูกเปิดวงจร และเอาท์พุทสัญญาณควบคุมโมเด็ม (DTR,RTS,OUT1 และ OUT2) จะต่อเข้ากับอินพุทสัญญาณควบคุมโมเด็มเป็นการภายใน การกระทำแบบนี้ ข้อมูลที่ถูกส่งจะถูกรับทันที ทำให้สามารถตรวจสอบการส่งและการรับข้อมูลของ 8250 ได้

บิท 5 ถึงบิท 7 จะเป็น "0" ตลอด

2.4.7 รีจิสเตอร์บอกสถานะโมเด็ม (MODEM STATUS REGISTER)

รีจิสเตอร์บอกสถานะโมเด็ม จะบอกถึงสถานะปัจจุบันของสัญญาณควบคุมจากโมเด็ม โดยที่ 4 บิทบนรีจิสเตอร์บอกสถานะโมเด็ม จะบอกถึงการเปลี่ยนแปลงของสัญญาณควบคุมจากโมเด็ม ซึ่งจะทำให้ 4 บิทนี้เป็น "1" และจะถูกรีเซทเป็น "0" เมื่อคอมพิวเตอร์อ่านข้อมูลจากรีจิสเตอร์บอกสถานะโมเด็ม รีจิสเตอร์บอกสถานะโมเด็มประกอบไปด้วยบิทต่างๆ ดังรูปที่ 2.10



รูปที่ 2.10 แสดงบิทต่างๆบนรีจิสเตอร์บอกสถานะโมเด็ม

บิท 0 เป็นบิทที่บอกว่าขาสัญญาณ CTS ได้เปลี่ยนสถานะแล้วหลังจากครั้งสุดท้ายที่คอมพิวเตอร์ได้อ่านข้อมูลจากรีจิสเตอร์บอกสถานะโมเด็มไป

บิท 1 เป็นบิทที่บอกว่าขาสัญญาณ DSR ได้เปลี่ยนสถานะแล้วหลังจากครั้งสุดท้ายที่คอมพิวเตอร์ได้อ่านข้อมูลจากรีจิสเตอร์บอกสถานะโมเด็มไป

บิท 2 เป็นบิทที่บอกว่าขาสัญญาณ RI ได้เปลี่ยนจาก "1" ไปเป็น "0"

บิท 3 เป็นบิทที่บอกว่าขาสัญญาณ RLSD ได้เปลี่ยนสถานะแล้ว

บิท 4 บิทนี้จะมีสถานะตรงข้ามกับขาสัญญาณ CTS

บิท 5 บิทนี้จะมีสถานะตรงข้ามกับขาสัญญาณ DSR

บิท 6 บิทนี้จะมีสถานะตรงข้ามกับขาสัญญาณ RI

บิท 7 บิทนี้จะมีสถานะตรงข้ามกับขาสัญญาณ RLSD

2.5 ภาษา BASICA กับการสื่อสารข้อมูล

ภาษา BASICA ของ IBM ที่เขียนขึ้นบนคอมพิวเตอร์ส่วนบุคคล IBM สามารถควบคุมให้คอมพิวเตอร์ส่วนบุคคล IBM ทำการสื่อสารข้อมูลผ่านทางอินเตอร์เฟส RS-232C ได้ โดยภาษา BASICA จะมีคำสั่งที่ใช้ควบคุมอะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัสอยู่ โดยที่สามารถควบคุมอัตราเร็วในการส่งข้อมูล (bps) การสร้างและตรวจสอบบิทพาริตี้ จำนวนบิทของข้อมูลแต่ละไบท์ ความยาวของบิทสิ้นสุด การส่งสัญญาณ RTS การตรวจสอบสัญญาณ CTS,DSR และ RLSD

ลักษณะโปรแกรมการสื่อสารข้อมูลของภาษา BASICA เป็นการเขียนและอ่านข้อมูลจากบัฟเฟอร์การสื่อสาร โดยที่ภาษา BASICA จะมองบัฟเฟอร์นี้เสมือนเป็นแฟ้มข้อมูลแฟ้มหนึ่ง ดังนั้นคำสั่งต่างๆที่เกี่ยวกับแฟ้มข้อมูล จะนำมาใช้กับแฟ้มสื่อสารได้ คำสั่งที่ BASICA เข้าไปควบคุมอะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัสโดยตรงคือ

```
OPEN "COMn: [baud][,[parity][,[data length][,[stop]
[,RS][,CS[n]][,DS[n]][,CD[n]][,BIN][,ASC][,PE][,LF]
]]]"[FOR mode] AS[#]filenumber [LEN=number]
```

COMn n จะมีค่าเป็น 1 หรือ 2 เพื่อบอกว่าจะใช้อะแดปเตอร์สื่อสาร

ข้อมูลแบบอะซิงโครนัสตัวที่ 1 หรือตัวที่ 2

baud เป็นค่าที่ใช้บอกถึงอัตราเร็วที่ต้องการในการรับส่งข้อมูล มีหน่วยเป็นบิทต่อวินาที ค่าที่สามารถเลือกใช้ได้คือค่า 75,110,150, 300,600,1200,2400,4800 และ 9600 บิทต่อวินาที ถ้าไม่มีการใส่ในส่วนนี้จะถือว่ามีความเร็วเท่ากับ 300 บิทต่อวินาที

parity เป็นอักษรตัวเดียว สำหรับบอกชนิดของการสร้างและตรวจสอบบิทพาริตี้ของข้อมูล มีรายละเอียดดังต่อไปนี้

S:SPACE ในช่วงที่เป็นบิทพาริตี้จะส่งสเปซ หรือลอจิก "0"

O:ODD การสร้างและตรวจสอบบิทพาริตี้จะเป็นแบบคี่

M:MARK ในช่วงที่เป็นบิทพาริตี้จะส่งมาร์ค หรือลอจิก "1"

E:EVEN การสร้างและตรวจสอบบิทพาริตี้จะเป็นแบบคู่

N:NONE ไม่มีการส่งบิทพาริตี้

ถ้าไม่มีการใส่ในส่วนนี้ ถือว่าเป็นการสร้างและตรวจสอบบิทพาริตี้แบบคู่

data length คือตัวเลขที่ใช้บอกว่าข้อมูลมีกี่บิท ค่าที่ให้เลือกมี 5,6,7 และ 8 ถ้าไม่มีการใส่ในส่วนนี้ถือว่าข้อมูลมี 7 บิท

stop เป็นตัวเลขที่ใช้บอกความยาวของบิทสิ้นสุด มีค่าที่ให้เลือกใช้คือ 1 และ 2 ถ้าไม่มีการใส่ในส่วนนี้ถือว่ามีความยาวเท่ากับ 1 บิท

filenumber เป็นตัวเลขบอกชื่อแฟ้ม

number เป็นตัวเลขบอกจำนวนไบท์สูงสุดที่สามารถอ่านจากบัฟเฟอร์สื่อสาร ถ้าไม่มีการใส่ถือว่าเท่ากับ 128 ไบท์

RS ถ้ามีการใส่ในส่วนนี้ จะไม่มีสัญญาณ RTS ส่งออกทางอะแดปเตอร์สื่อสารข้อมูล

CS[n] n มีค่าตั้งแต่ 0 ถึง 65535 ใช้กำหนดให้คอมพิวเตอร์รอสัญญาณ CTS เป็นเวลา n มิลลิวินาที ถ้า n มีค่าเท่ากับ 0 จะไม่มีการตรวจเช็คสัญญาณ CTS ถ้าไม่มีการใส่ในส่วนนี้ถือว่า n มีค่าเท่ากับ 1000 แต่ถ้ามีการใส่ RS n จะมีค่าเท่ากับ 0

DS[n] เช่นเดียวกับ CS[n] แต่สัญญาณที่ตรวจเช็คเป็นสัญญาณ DSR ถ้า

	ไม่มีการใส่ในส่วนนี้ถือว่า n มีค่าเท่ากับ 1000
CD[n]	เช่นเดียวกับ CS[n] แต่สัญญาณที่ตรวจเช็คเป็นสัญญาณ RLSD ถ้าไม่มีการใส่ในส่วนนี้ถือว่า n มีค่าเท่ากับ 0
LF	เป็นสัญญาแทีบอกให้มีการขึ้นบรรทัดใหม่ (LINE FEED) ทุกครั้งที่มีการส่งสัญญาณการเลื่อนกลับของแคร่ (CARRIAGE RETURN)
BIN	เป็นการบอกว่าข้อมูลที่ส่งไม่มีความหมายในตัวเอง มีแต่ลักษณะการส่งบิท "1" และบิท "0" เท่านั้น
ASC	เป็นการบอกว่าข้อมูลที่ส่งมีความหมายในตัวเองด้วย เช่นการเลื่อนกลับของแคร่
PE	ถ้าไม่มีการใส่ในส่วนนี้ แสดงว่าไม่มีการตรวจสอบบิทพาริตี้ทางด้านรับ
FOR mode	เป็นการกำหนดว่า แฟ้มสื่อสารนี้เป็นแฟ้มแบบส่งหรือแบบรับ หรือเป็นทั้งแบบส่งและแบบรับ

บทที่ 3

การออกแบบและการสร้าง

3.1 การออกแบบโมเด็ม

โมเด็มสำหรับการสื่อสารข้อมูลผ่านระบบวิทยุ จะต้องสามารถทำหน้าที่ต่างๆดังต่อไปนี้ได้ คือ

3.1.1 การอินเตอร์เฟสเข้ากับอุปกรณ์ปลายทางข้อมูลหรือคอมพิวเตอร์

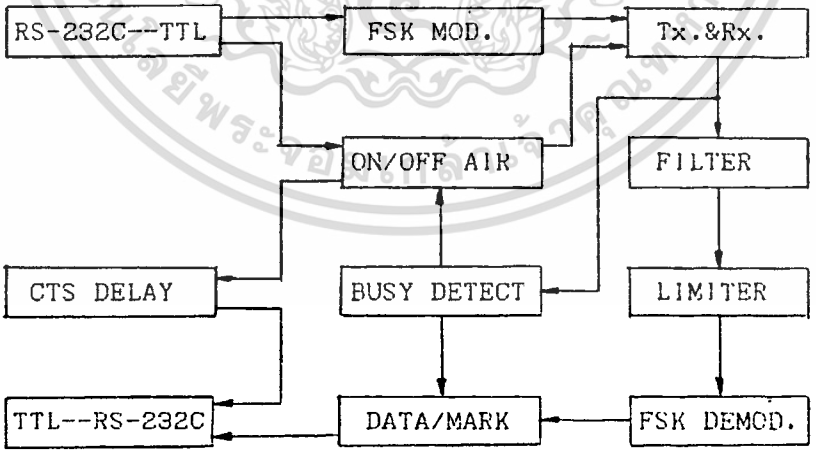
3.1.2 การโมดูเลชั่น และการดีโมดูเลชั่น

3.1.3 การตรวจสอบการออกอากาศจากเครื่องส่งวิทยุอื่น

3.1.4 การควบคุมการออกอากาศของเครื่องส่งวิทยุ

3.1.5 การส่งสัญญาณเสียงพูดไปให้กับเครื่องส่งวิทยุ และการรับสัญญาณเสียงพูดจากเครื่องรับวิทยุ

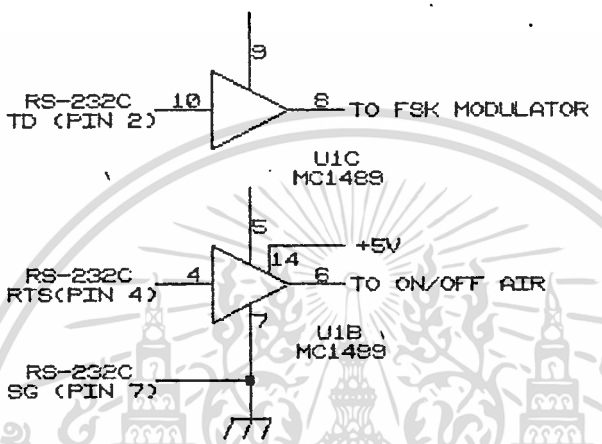
จากหน้าที่ต่างๆของโมเด็มสำหรับการสื่อสารข้อมูลผ่านระบบวิทยุ นำมาเขียนเป็นบล็อคไดอะแกรมได้ดังรูปที่ 3.1



รูปที่ 3.1 บล็อคไดอะแกรมของโมเด็มสำหรับการสื่อสารข้อมูลผ่านระบบวิทยุ

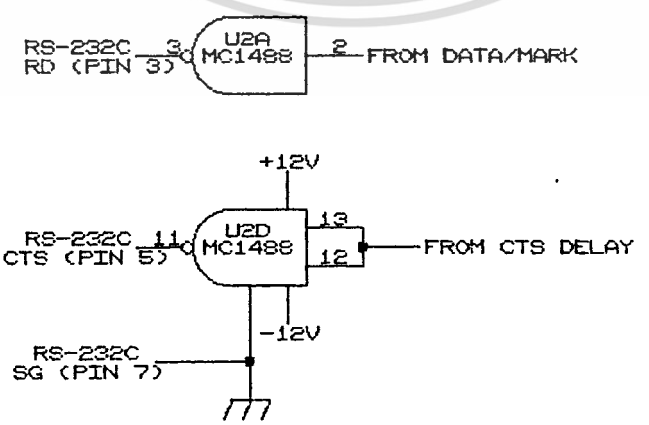
3.2 การสร้างโมเด็ม 300 บิทต่อวินาที

3.2.1 การเปลี่ยนระดับสัญญาณ RS-232C เป็นระดับสัญญาณ TTL นำเอาไอซี MC 1489 มาใช้งาน ซึ่ง MC1489 เป็นไอซีที่ผลิตขึ้นมาเพื่อทำหน้าที่เป็น RS-232C LINE RECEIVER การนำไปใช้งาน MC1489 ต้องการแหล่งจ่ายไฟ +5 โวลท์ ดังแสดงในรูปที่ 3.2



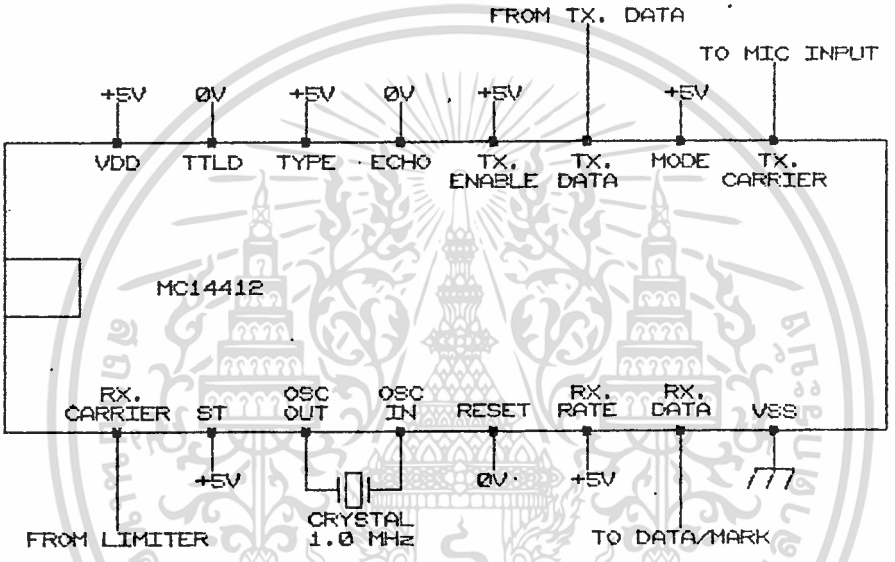
รูปที่ 3.2 การนำ MC1489 ไปใช้งาน

3.2.2 การเปลี่ยนระดับสัญญาณ TTL เป็นระดับสัญญาณ RS-232C นำเอาไอซี MC1488 มาใช้งาน ซึ่งเป็นไอซีที่ผลิตขึ้นมาเพื่อทำหน้าที่เป็น RS-232C LINE DRIVER MC1488 ต้องการแหล่งจ่ายไฟ +12 โวลท์ และ -12 โวลท์ การนำไปใช้งานดังแสดงในรูปที่ 3.3



รูปที่ 3.3 การนำ MC1488 ไปใช้งาน

3.2.3 วงจรโมดูเลเตอร์ของการเข้ารหัสแบบความถี่ นำเอาไอซี MC14412 มาใช้ในโมเด็มนี้ ซึ่งเป็นไอซีที่ทำหน้าที่แปลงสัญญาณข้อมูลให้เป็นสัญญาณเสียงพูดโดยการเข้ารหัสแบบความถี่ อัตราเร็วในการส่งข้อมูลเท่ากับ 300 บิทต่อวินาที การทำงานของ MC14412 เมื่อต่อขาต่างๆตามรูปที่ 3.4 ถ้าสัญญาณข้อมูลส่งเป็น "1" สัญญาณเสียงพูดที่ได้จะมีความถี่ 1270 เฮิทซ์ แต่ถ้าสัญญาณข้อมูลส่งเป็น "0" สัญญาณเสียงพูดที่ได้จะมีความถี่ 1070 เฮิทซ์

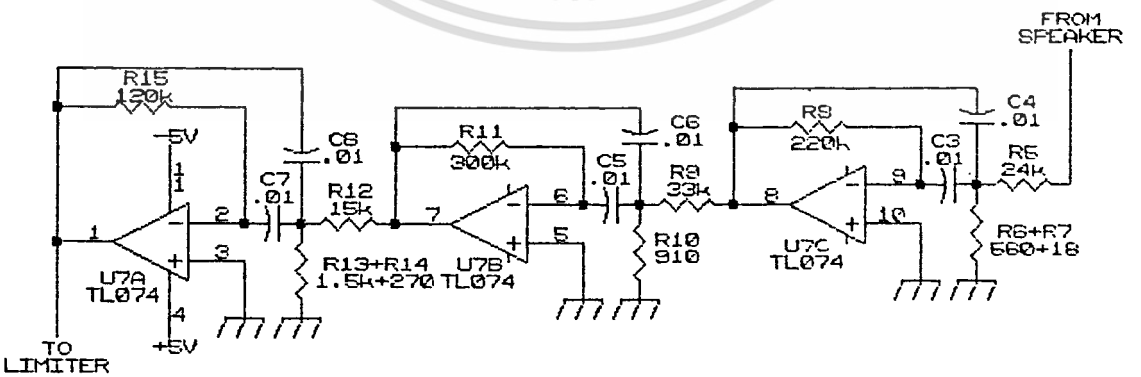


รูปที่ 3.4 การนำ MC14412 ไปใช้งาน

3.2.4 วงจรดีโมดูเลเตอร์ของการเข้ารหัสแบบความถี่ วงจรส่วนนี้นำเอาไอซี MC14412 มาใช้เช่นเดียวกับวงจรโมดูเลเตอร์ โดยจะทำหน้าที่แปลงสัญญาณสแควร์ (SQUARE WAVE) ที่มีความถี่เท่ากับสัญญาณเสียงพูดที่รับได้ให้เป็นสัญญาณข้อมูลอีกครั้ง เหตุที่ต้องใช้สัญญาณสแควร์ เพราะภาคดีโมดูเลเตอร์ของ MC14412 จะทำงานได้ก็ต่อเมื่ออินพุทเป็นสัญญาณสแควร์เท่านั้น จากรูปที่ 3.4 MC14412 ยังสามารถนำไปใช้ในวงจรดีโมดูเลเตอร์ได้ด้วย เนื่องจาก MC14412 มีทั้งภาคโมดูเลเตอร์และภาคดีโมดูเลเตอร์อยู่ภายในตัวเดียวกัน ดังนั้นในโมเด็มจึงใช้ MC14412 เพียงตัวเดียวเท่านั้น โดยถ้าสัญญาณสแควร์มีความถี่เท่ากับ 1270 เฮิทซ์ สัญญาณข้อมูลรับจะเป็น "1" แต่ถ้าสัญญาณสแควร์มีความถี่เท่ากับ 1070 เฮิรซ์ สัญญาณข้อมูลรับจะเป็น "0"

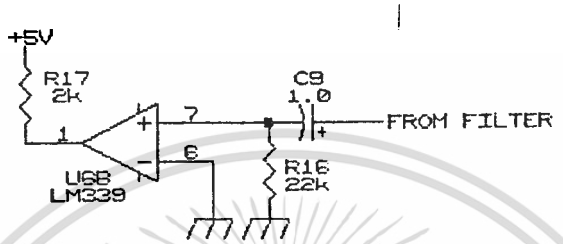
3.2.5 ฟิลเตอร์ นำเอาฟิลเตอร์ชนิดแอคทีฟแบนด์พาสฟิลเตอร์แบบมัลติเพิลฟีดแบค (MULTIPLE FEEDBACK ACTIVE BANDPASS FILTER) มาใช้งาน โดยต้องการให้สัญญาณเสียงพูดที่รับได้ผ่านเฉพาะย่านความถี่ตั้งแต่ 1020 เฮิทซ์ ถึง 1320 เฮิทซ์ ซึ่งจะทำการขยายสัญญาณย่านความถี่นี้ และจะลดทอนสัญญาณย่านความถี่ที่ต่ำกว่าและสูงกว่านี้ เพื่อป้องกันสัญญาณรบกวน (NOISE) ที่ไม่ต้องการผ่านไปสู่วงจรดีโมดูเลเตอร์ ซึ่งอาจจะทำให้ข้อมูลที่ได้จากการดีโมดูเลทผิดพลาดได้ เหตุที่เลือกใช้ฟิลเตอร์ชนิดแอคทีฟเพราะ ไม่มีตัวเหนี่ยวนำ (INDUCTOR) ซึ่งยุ่งยากในการพันให้ได้ค่าตามต้องการ มีความสามารถในการขยายสัญญาณ อินพุทอิมพีแดนซ์ (INPUT IMPEDANCE) สูง เอาท์พุทอิมพีแดนซ์ (OUTPUT IMPEDANCE) ต่ำ

เส้นแสดงผลตอบสนองต่อความถี่ของฟิลเตอร์ที่ใช้ในวงจรโมเด็มนี้ จะต้องมีความลาดชันที่สูง มีย่านความถี่ผ่านกว้างและราบเรียบตลอดย่านความถี่ ซึ่งฟิลเตอร์ชนิดแอคทีฟแบนด์พาสฟิลเตอร์แบบมัลติเพิลฟีดแบคที่มีออร์เดอร์เท่ากับ 2 จะมีแต่เพียงความลาดชันสูงเท่านั้น ดังนั้นถ้าต้องการให้ได้ผลตอบสนองต่อความถี่ตามต้องการ จะต้องนำเอาแอคทีฟแบนด์พาสฟิลเตอร์แบบมัลติเพิลฟีดแบคที่มีออร์เดอร์เท่ากับ 2 มาต่อแคสเคด (CASCADE) กัน 3 ตัว โดยแต่ละตัวมีค่าความถี่กลางที่ต่างกัน แต่เรียงกันอยู่ในย่านความถี่ที่ต้องการ ฟิลเตอร์ที่ได้จะมีออร์เดอร์เท่ากับ 6 เนื่องจากการออกแบบฟิลเตอร์ที่มีออร์เดอร์เท่ากับ 6 มีความยุ่งยากและสลับซับซ้อน จึงได้นำเอาฟิลเตอร์ที่ได้มีการออกแบบไว้แล้วสำหรับโมเด็มที่มีความถี่ใช้งานเช่นเดียวกับโมเด็มนี้มาใช้งานดังแสดงในรูปที่ 3.5



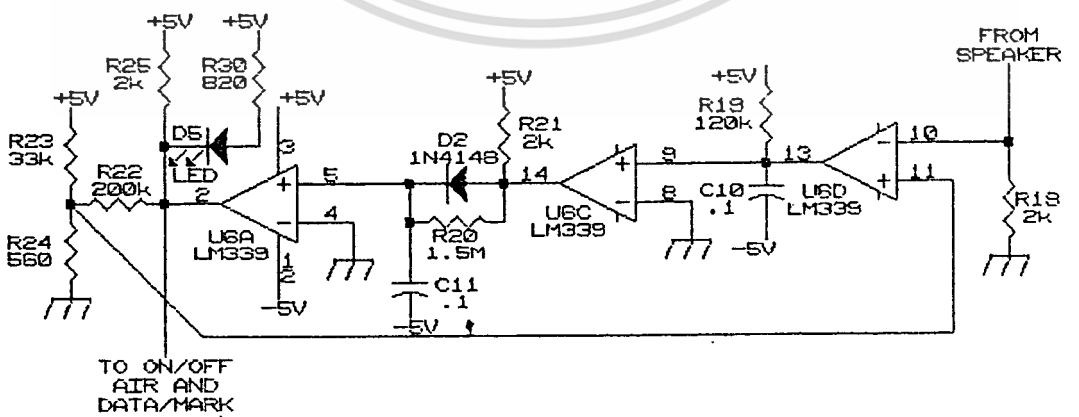
รูปที่ 3.5 ฟิลเตอร์ชนิดแอคทีฟแบนด์พาสฟิลเตอร์แบบมัลติเพิลฟีดแบคสำหรับโมเด็ม

3.2.6 ลิมิตเตอร์ (LIMITER) จะแปลงสัญญาณเสียงพูดที่มาจากวงจรฟิลเตอร์ ให้กลายเป็นสัญญาณสแควร์ที่มีความถี่เดียวกัน โดยนำเอาไอซี LM339 มาใช้งาน ซึ่งเป็นไอซีคอมพาราเตอร์ ทำการเปรียบเทียบระดับของสัญญาณจากวงจรฟิลเตอร์กับกราวนด์ ดังแสดงในรูปที่ 3.6



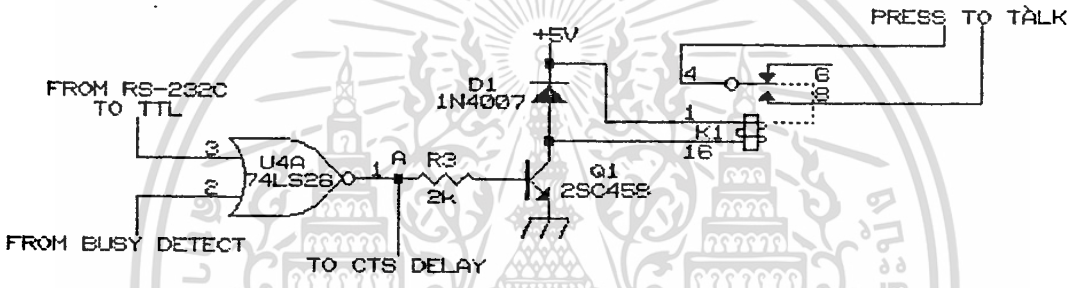
รูปที่ 3.6 วงจรลิมิตเตอร์

3.2.7 วงจรตรวจสอบการออกอากาศจากเครื่องรับส่งวิทยุอื่น ในขณะที่เครื่องรับส่งวิทยุเครื่องอื่นทำการออกอากาศอยู่นั้น เครื่องรับส่งวิทยุที่ต่ออยู่กับโมเด็มจะสามารถรับสัญญาณวิทยุได้ และจะให้เสียงออกมาที่ช่องลำโพง วงจรส่วนนี้ทำหน้าที่ตรวจสอบดูว่ามีสัญญาณออกมาที่ช่องลำโพงหรือไม่ โดยจะเปรียบเทียบเฉพาะพีคของสัญญาณที่มาจากช่องลำโพง ซึ่งอาศัยไอซี LM339 ในการเปรียบเทียบนี้ ดังแสดงในรูปที่ 3.7



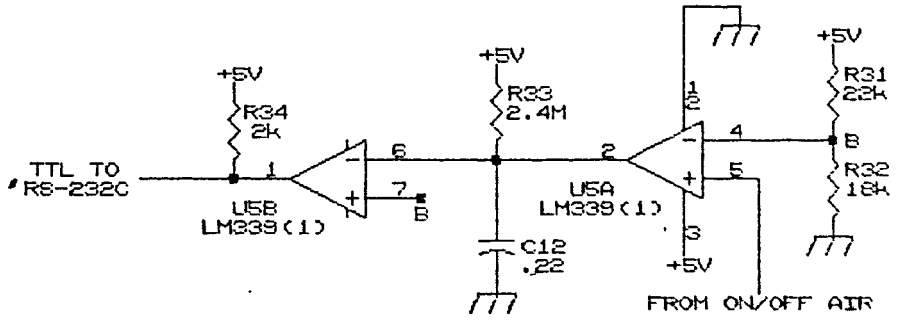
รูปที่ 3.7 วงจรตรวจสอบการออกอากาศจากเครื่องรับส่งวิทยุอื่น

3.2.8 วงจรควบคุมการออกอากาศของเครื่องรับส่งวิทยุ เมื่อคอมพิวเตอร์ต้องการที่จะส่งข้อมูล จะส่งสัญญาณ REQUEST TO SEND มาทางอินเตอร์เฟส RS-232C ให้กับโมเด็ม ซึ่งสัญญาณ RTS นี้จะผ่านมายังวงจรส่วนนี้ เพื่อควบคุมการออกอากาศของเครื่องรับส่งวิทยุ ซึ่งเครื่องรับส่งวิทยุจะสามารถออกอากาศได้นั้นขึ้นอยู่กับในขณะนั้นมีการออกอากาศช่องเครื่องรับส่งวิทยุเครื่องอื่นหรือไม่ โดยขึ้นอยู่กับวงจรตรวจสอบการออกอากาศจากเครื่องรับส่งวิทยุเครื่องอื่น วงจรส่วนนี้จะไปควบคุมให้คอนแทค (CONTACT) ของรีเลย์ (RELAY) ต่อกัน ซึ่งคอนแทคของรีเลย์จะต่ออยู่กับช่อง PRESS TO TALK ของเครื่องรับส่งวิทยุ ดังแสดงในรูปที่ 3.8



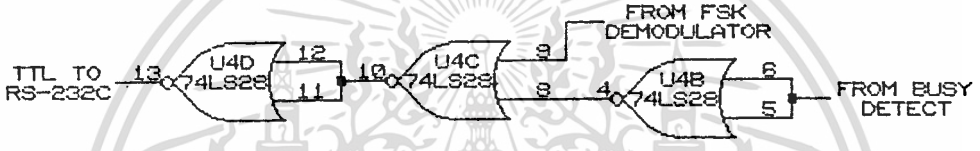
รูปที่ 3.8 วงจรควบคุมการออกอากาศของเครื่องรับส่งวิทยุ

3.2.9 วงจรหน่วงเวลาการตอบสัญญาณ CLEAR TO SEND โมเด็มจะตอบสัญญาณ CLEAR TO SEND ไปให้กับคอมพิวเตอร์ หลังจากที่โมเด็มสามารถควบคุมให้มีการออกอากาศได้ สัญญาณ CTS จะไม่ถูกส่งไปให้กับคอมพิวเตอร์ทันทีที่มีการออกอากาศ แต่จะหน่วงเวลาไว้ชั่วครู่หนึ่ง เพื่อให้แน่ใจว่าสัญญาณที่เอาท์พุทของวงจรดีโมดูเลเตอร์ทางด้านตรงข้ามเป็น "1" ก่อนที่จะทำการส่งข้อมูลออกไป จึงจะทำให้ข้อมูลที่ส่งไปไม่เกิดการสูญหาย ดังแสดงวงจรในรูปที่ 3.9



รูปที่ 3.9 วงจรหน่วงเวลาการตอบสัญญาณ CTS

3.2.10 วงจรให้ข้อมูลรับผ่าน (DATA/MARK) ในขณะที่ไม่มีการออกอากาศจากเครื่องรับส่งวิทยุเครื่องอื่น แต่ที่เอาท์พุทของฟิลเตอร์จะมีสัญญาณรบกวนออกมา ซึ่งสัญญาณรบกวนนี้หลังจากไปผ่านวงจรลิมิตเตอร์ แล้วไปเข้าวงจรดีโมดูเลเตอร์ จะมีผลทำให้สัญญาณที่ได้เป็น "0" หรือเป็น "1" ไม่แน่นอน ซึ่งในการส่งข้อมูลแบบอะซิงโครนัส ถ้าในระหว่างไม่มีการส่งข้อมูล จะต้องเป็น "1" ตลอด ซึ่งถ้าเกิดเป็น "0" จะทำให้คอมพิวเตอร์ฝ่ายรับคิดว่าเป็นบิทเริ่มต้น และจะต้องมีบิทของข้อมูลตามหลังบิทเริ่มต้นมา ทั้งๆที่ยังไม่มีการส่งข้อมูล วงจรส่วนนี้จะส่ง "1" ไปให้กับคอมพิวเตอร์ ในขณะไม่มีการออกอากาศจากเครื่องรับส่งวิทยุเครื่องอื่น ดังแสดงในรูปที่ 3.10

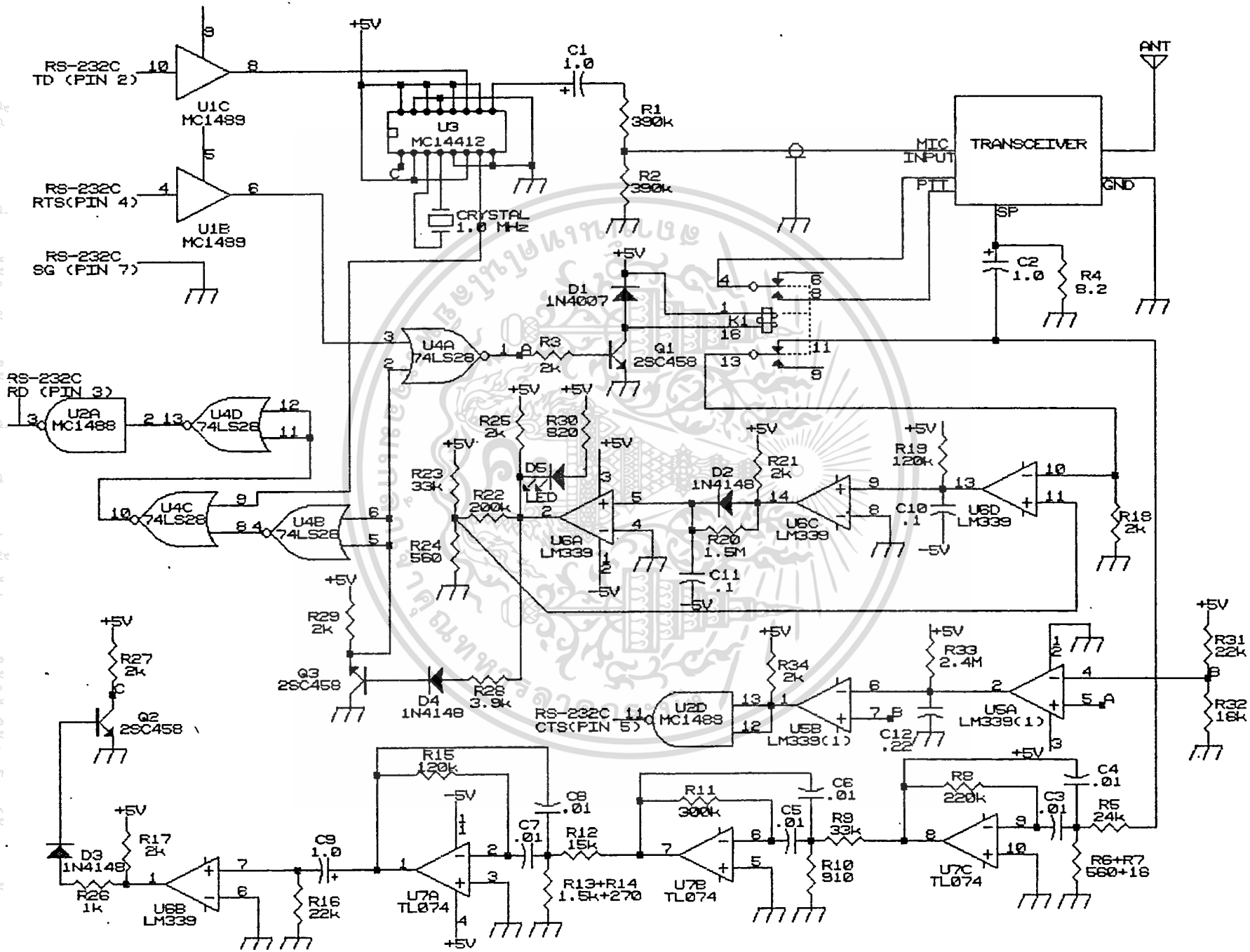


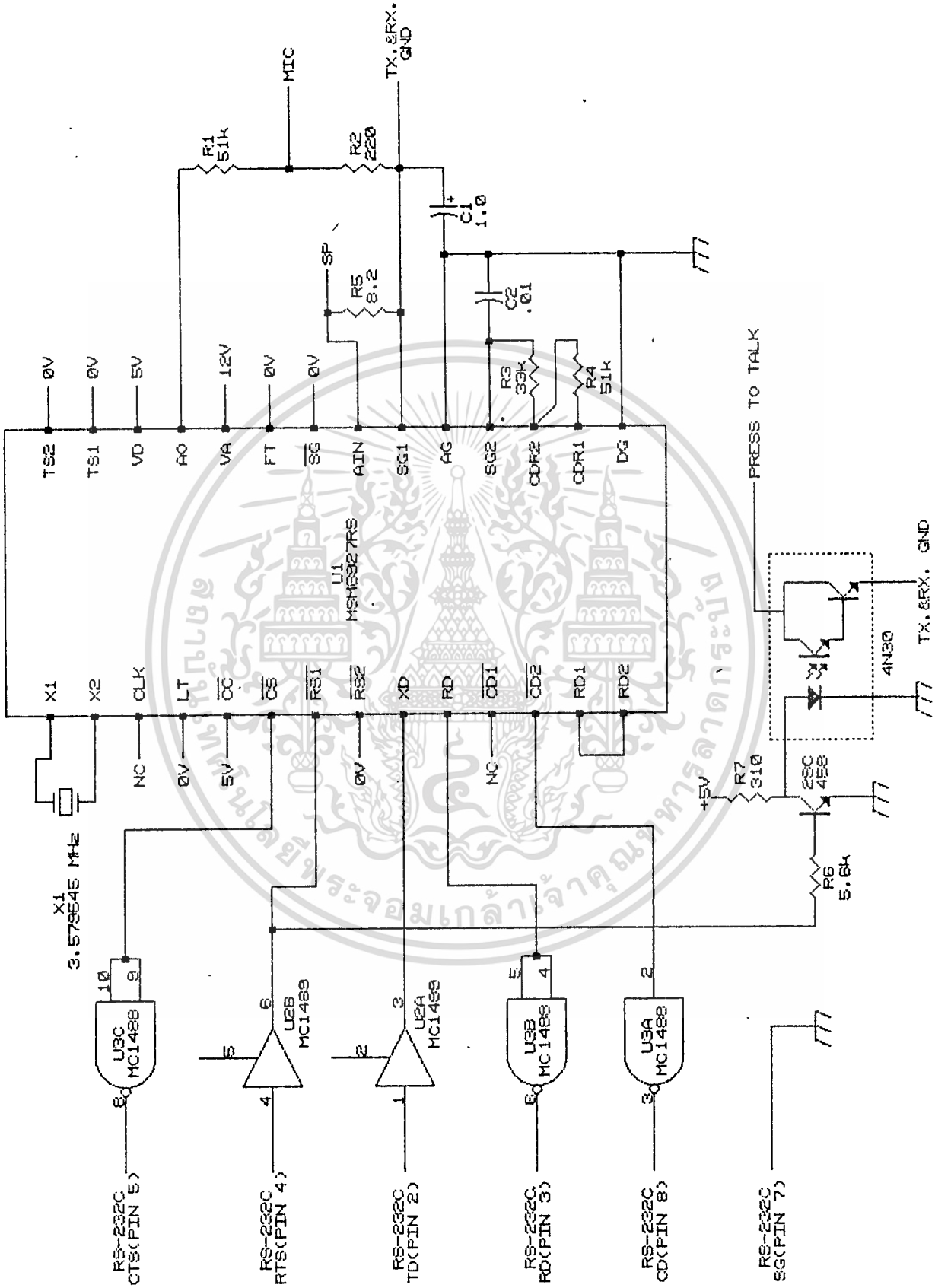
รูปที่ 3.10 วงจรให้ข้อมูลรับผ่าน

จากการนำวงจรในส่วนต่างๆของโมเด็มมาต่อเข้าด้วยกัน จะได้วงจรของโมเด็ม 300 บิทต่อวินาทีสำหรับการสื่อสารข้อมูลผ่านระบบวิทยุ ดังรูปที่ 3.11

3.3 การสร้างโมเด็ม 1200 บิทต่อวินาที

โมเด็ม 1200 บิทต่อวินาที นำเอาไอซี MSM6927RS มาใช้งาน ซึ่ง MSM6927RS สามารถทำงานตามบล็อคไดอะแกรมจากรูปที่ 3.1 ได้เกือบหมด ยกเว้นการเปลี่ยนระดับสัญญาณไปมาระหว่างระดับสัญญาณ TTL กับระดับสัญญาณ RS-232C และการควบคุมการออกอากาศของเครื่องรับส่งวิทยุ การนำ MSM6927RS ไปใช้งานจึงต้องอาศัย MC1488, MC1489 และ 4N30 (OPTO ISOLATOR) ซึ่งจะได้วงจรของโมเด็ม 1200 บิทต่อวินาทีสำหรับการสื่อสารข้อมูลผ่านระบบวิทยุ ดังรูปที่ 3.12


RS-232C
TD (PIN 2)
U1C
MC1489
RS-232C
RTS(PIN 4)
U1B
MC1489
RS-232C
SG (PIN 7)
+5V
C1
1.0
U3
MC14412
CRYSTAL
1.0 MHz
R1
390k
R2
390k
ANT
TRANSCEIVER
MIC
INPUT
PTT
GND
SP
C2
1.0
R4
8.2
D1
1N4007
K1
Q1
2SC458
U4A
74LS28
R3
2k
RS-232C
RD (PIN 3)
U2A
MC1488
U4D
74LS28
U4C
74LS28
U4B
74LS28
R25
2k
R30
820
R23
33k
R22
200k
D5
LED
R24
560
U6A
LM339
D2
1N4148
R21
2k
R20
1.5M
C11
.1
U6C
LM339
R19
120k
C10
.1
U6D
LM339
R18
2k
R29
2k
Q3
2SC458
D4
1N4148
R28
3.9k
R27
2k
Q2
2SC458
RS-232C
CTS(PIN 5)
U2D
MC1488
R34
2k
U5B
LM339(1)
R33
2.4M
C12
.22
U5A
LM339(1)
R31
22k
R32
18k
R15
120k
C8
.01
C7
.01
R12
15k
R13+R14
1.5k+270
U7A
TL074
R11
300k
C6
.01
C5
.01
R9
33k
R10
910
U7B
TL074
R8
220k
C3
.01
C4
.01
R5
24k
R6+R7
560+16
U7C
TL074
D3
1N4148
R17
2k
R26
1k
U6B
LM339
C9
1.0
R16
22k



รูปที่ 3.12 วงจรของโมเด็ม 1200 บิทต่อวินาทีสำหรับการสื่อสารข้อมูลผ่านระบบวิทยุ

3.4 การเขียนโปรแกรมภาษา BASICA

ในการเขียนโปรแกรมภาษา BASICA เพื่อควบคุมคอมพิวเตอร์ส่วนบุคคล IBM ในการสื่อสารข้อมูล จะต้องนำคำสั่งสำหรับเปิดแฟ้มข้อมูลสื่อสาร คำสั่งเขียนข้อมูลลงในแฟ้มข้อมูล คำสั่งอ่านข้อมูลจากแฟ้มข้อมูล และคำสั่งปิดแฟ้มข้อมูล มาใช้งาน เช่นคำสั่ง

```
OPEN "COMn: [baud][,[parity][,[data length][,[stop]
[,RS][,CS[n]][,DS[n]][,CD[n]][,BIN][,ASC][,PE][,LF]
]]]"[FOR mode] AS[#]filenumber[LEN=number]
WRITE#filenumber,string,string,...
INPUT#filenumber,string,string,...
CLOSE#filenumber
```

โปรแกรมภาษา BASICA ที่เขียนขึ้นสำหรับการสื่อสารข้อมูลมีดังต่อไปนี้

BP1 เป็นโปรแกรมสำหรับทดลองการส่งข้อมูล โฟลว์ชาร์ทของโปรแกรมดูได้จากรูปที่ 3.13ก. ส่วนโปรแกรมสามารถดูได้จากภาคผนวก ก.1

BP2 เป็นโปรแกรมสำหรับทดลองการรับข้อมูล โฟลว์ชาร์ทของโปรแกรมดูได้จากรูปที่ 3.13ข. ส่วนโปรแกรมสามารถดูได้จากภาคผนวก ก.2

BP3 เป็นโปรแกรมสำหรับทดลองการส่งและรับข้อมูล สำหรับคอมพิวเตอร์ที่เริ่มรับข้อมูลก่อน โฟลว์ชาร์ทของโปรแกรมดูได้จากรูปที่ 3.15ก. ส่วนโปรแกรมสามารถดูได้จากภาคผนวก ก.3

BP4 เป็นโปรแกรมสำหรับทดลองการส่งและรับข้อมูล สำหรับคอมพิวเตอร์ที่เริ่มรับข้อมูลก่อน โฟลว์ชาร์ทของโปรแกรมดูได้จากรูปที่ 3.15ข. ส่วนโปรแกรมสามารถดูได้จากภาคผนวก ก.4

3.5 การเขียนโปรแกรมภาษา TURBO Pascal

โปรแกรมภาษา TURBO Pascal นั้น ไม่มีคำสั่งสำหรับการสื่อสารข้อมูลโดยเฉพาะเหมือนกับภาษา BASICA ดังนั้นการเขียนโปรแกรมภาษา TURBO Pascal เพื่อควบคุมคอมพิวเตอร์ส่วนบุคคล IBM ในการสื่อสารข้อมูล จะต้องอาศัยคำสั่งสำหรับการเขียนข้อมูลไปที่ I/O ตำแหน่งที่ H&3F8 ถึงตำแหน่งที่ H&3FF และคำสั่งการอ่านข้อมูลจาก I/O ตำแหน่งดังกล่าว เพื่อเขียนและอ่านข้อมูลจากรีจิสเตอร์ของ 8250 ในอะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัส คำสั่งสำหรับการเขียนและอ่านข้อมูลไปที่ I/O ตำแหน่งต่างๆคือ

port[I/O number] := รหัสของข้อมูล

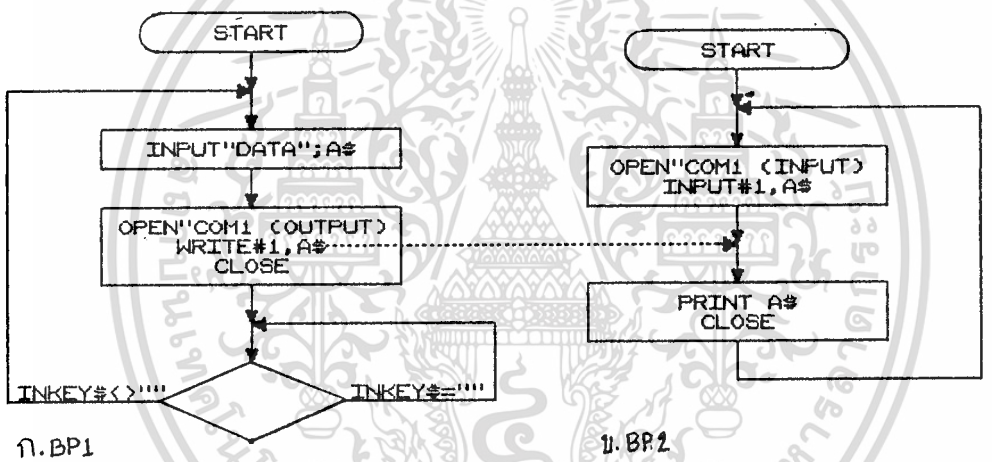
variable := port[I/O number]

โปรแกรมภาษา TURBO Pascal ที่เขียนขึ้นสำหรับการสื่อสารข้อมูลมีดังต่อไปนี้

PP1 เป็นโปรแกรมสำหรับทดลองการส่งข้อมูล โฟลว์ชาร์ทของโปรแกรมดูได้จากรูปที่ 3.14ก. ส่วนโปรแกรมสามารถดูได้จากภาคผนวก ก.5

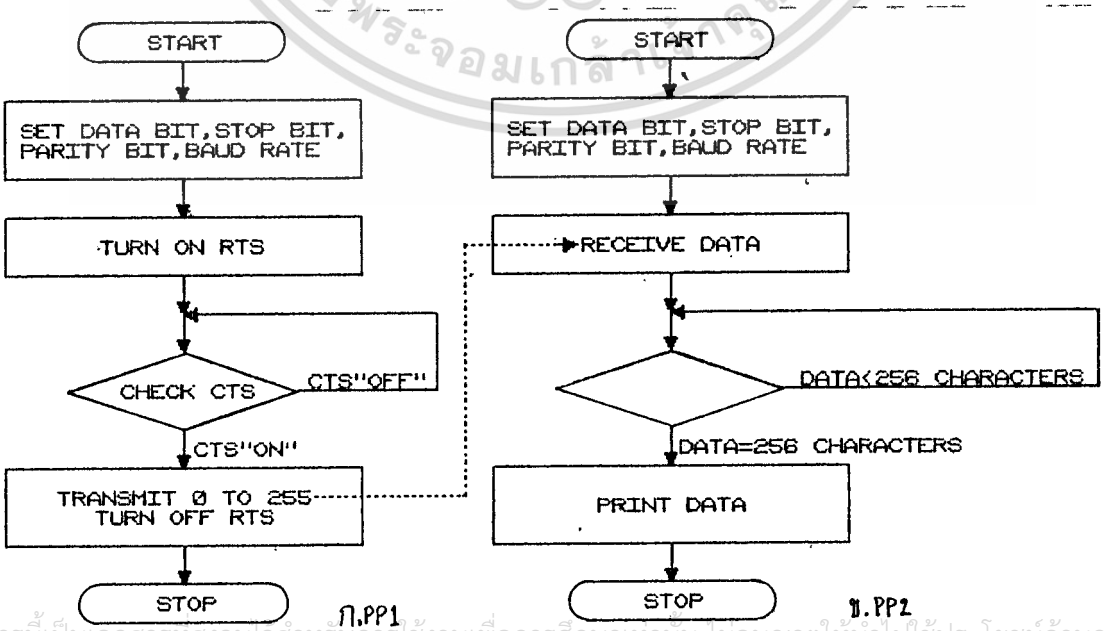
PP2 เป็นโปรแกรมสำหรับทดลองการรับข้อมูล โฟลว์ชาร์ทของโปรแกรมดูได้จากรูปที่ 3.14ข. ส่วนโปรแกรมสามารถดูได้จากภาคผนวก ก.6

COMM เป็นโปรแกรมสำหรับการสื่อสารข้อมูลสำหรับใช้งานจริง ซึ่งโปรแกรมสามารถดูได้จากภาคผนวก ก.7



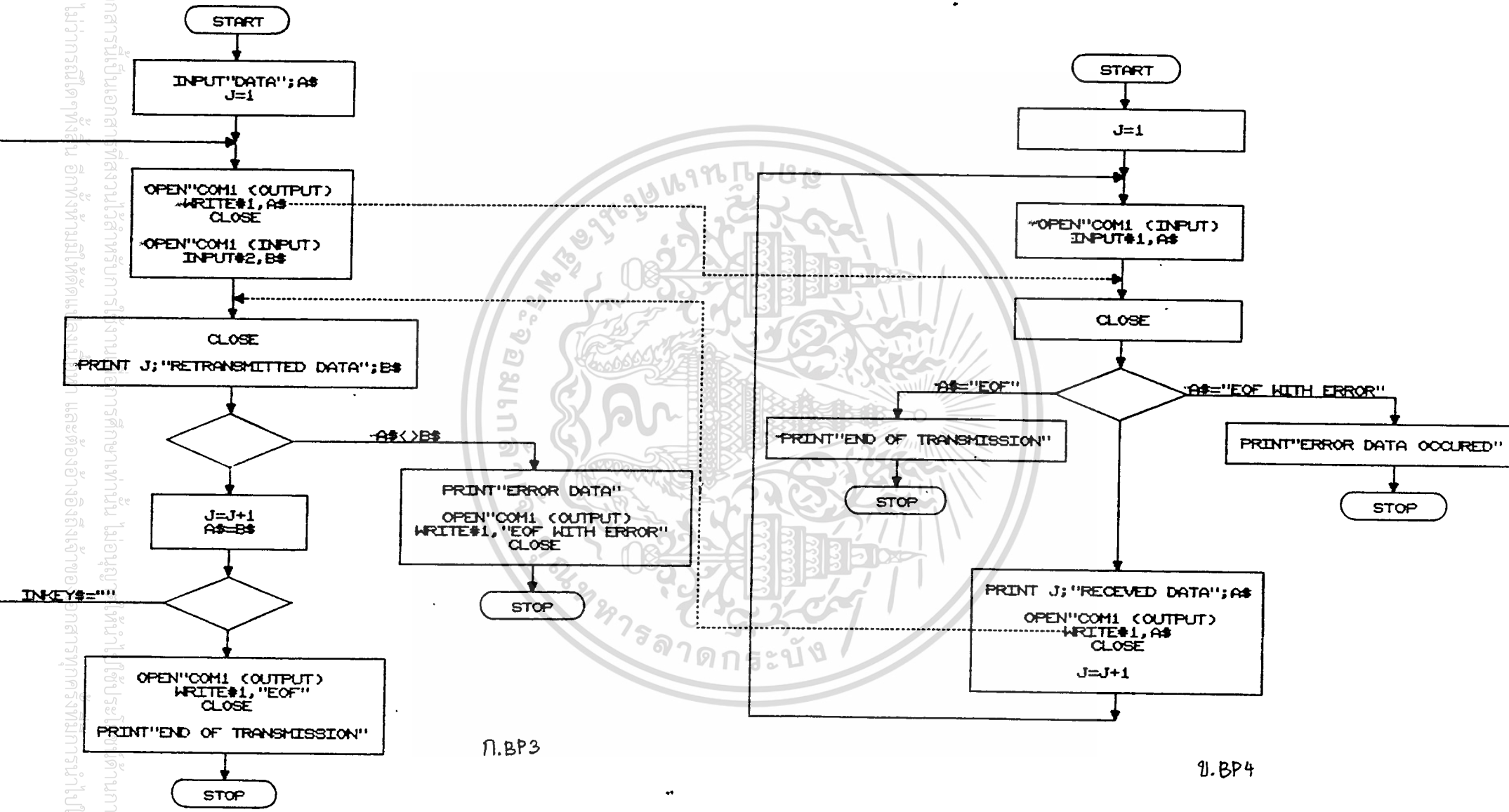
ก.BP1 ข.BP2

รูปที่ 3.13 โฟลว์ชาร์ทของโปรแกรม ก.BP1 ข.BP2



ก.PP1 ข.PP2

รูปที่ 3.14 โฟลว์ชาร์ทของโปรแกรม ก.PP1 ข.PP2



รูปที่ 3.15 โฟลว์ชาร์ทของโปรแกรม ก.BP3 ข.BP4

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลองวัดค่าต่างๆของวงจรโมเด็ม 300 บิทต่อวินาที

4.1.1 การวัดความถี่ของสัญญาณเสียงพูดที่ได้จากวงจรโมดูเลเตอร์

สัญญาณข้อมูลเป็น "1" สัญญาณเสียงพูดที่ได้จะมีความถี่ = 1270 เฮิทซ์

สัญญาณข้อมูลเป็น "0" สัญญาณเสียงพูดที่ได้จะมีความถี่ = 1073 เฮิทซ์

4.1.2 การวัดระดับสัญญาณในการทำงานของวงจรตรวจสอบการออกอากาศ

ก. วงจรตรวจสอบการออกอากาศไม่ได้ต่ออยู่ในวงจรโมเด็ม

ขณะวงจรตรวจสอบการออกอากาศ "OFF" ระดับสัญญาณที่ทำให้วงจร "ON" = 190 $mV._{p-p}$

ขณะวงจรตรวจสอบการออกอากาศ "ON" ระดับสัญญาณที่ทำให้วงจร "OFF" = 150 $mV._{p-p}$

ข. วงจรตรวจสอบการออกอากาศต่ออยู่ในวงจรโมเด็ม

ขณะวงจรตรวจสอบการออกอากาศ "OFF" ระดับสัญญาณที่ทำให้วงจร "ON" = 180 $mV._{p-p}$

ขณะวงจรตรวจสอบการออกอากาศ "ON" ระดับสัญญาณที่ทำให้วงจร "OFF" = 140 $mV._{p-p}$

4.1.3 การวัดเวลาในการตอบสัญญาณ CTS หลังจากที่คอมพิวเตอร์ส่งสัญญาณ RTS

ทำการวัดได้โดยการเขียนโปรแกรมภาษา BASICA โดยการเปิดแฟ้มข้อมูลสื่อสาร และตั้งค่าเวลาการรอรับสัญญาณ CTS จนเกิด DEVICE TIME OUT

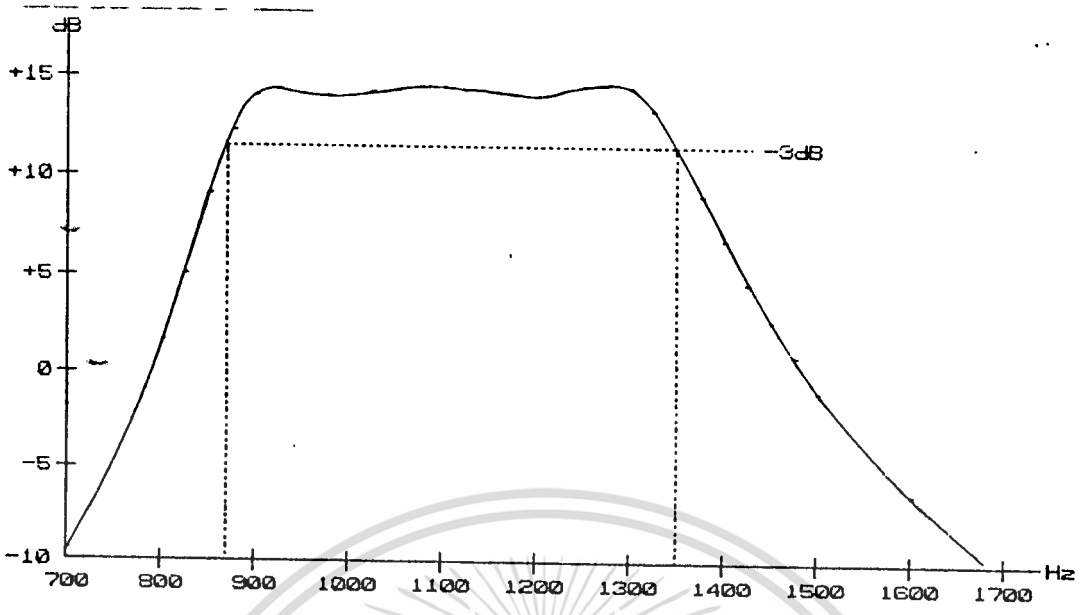
จะได้เวลาในการตอบสัญญาณ CTS = 425 มิลลิวินาที

4.1.4 การวัดผลตอบสนองต่อความถี่ของวงจรฟิลเตอร์

เมื่อป้อนสัญญาณซายน์ที่มีระดับสัญญาณ 400 $mV._{p-p}$ ให้กับอินพุทของวงจรฟิลเตอร์ และทำการวัดระดับสัญญาณที่เอาท์พุทของวงจรฟิลเตอร์ จะได้ผลตอบสนองต่อความถี่ดังตารางที่ 4.1 ซึ่งจะได้เส้นแสดงผลตอบสนองต่อความถี่ดังรูปที่ 4.1

ตารางที่ 4.1 แสดงผลตอบสนองต่อความถี่ของวงจรฟิลเตอร์

ความถี่	แกน (dB)	ความถี่	แกน (dB)
300	-36.48	1250	14.61
400	-32.04	1275	14.71
500	-26.47	1300	14.51
600	-18.60	1325	13.42
700	-9.43	1350	11.48
800	1.76	1375	9.10
825	5.11	1400	6.85
850	9.10	1425	4.74
875	12.31	1450	2.77
900	14.09	1475	1.02
925	14.40	1500	-0.80
950	14.19	1600	-6.24
975	14.09	1700	-10.83
1000	14.09	1800	-14.20
1025	14.30	1900	-17.30
1050	14.40	2000	-20.00
1075	14.61	2100	-22.50
1100	14.61	2200	-24.26
1125	14.40	2300	-26.02
1150	14.30	2400	-27.69
1175	14.19	2500	-29.43
1200	14.09	2600	-30.46
1225	14.30	2700	-32.04



รูปที่ 4.1 เส้นแสดงผลตอบสนองต่อความถี่ของวงจรฟิลเตอร์

จากผลตอบสนองต่อความถี่ของฟิลเตอร์ จะได้จุดตัดความถี่ต่ำ (LOW FREQUENCY CUT OFF) ที่ความถี่ประมาณ 870 เฮิทซ์ และจุดตัดความถี่สูง (HIGH FREQUENCY CUT OFF) ที่ความถี่ประมาณ 1350 เฮิทซ์ ดังนั้นแถบกว้างความถี่ผ่านของฟิลเตอร์จะมีค่าประมาณ 480 เฮิทซ์

4.1.5 การวัดระดับสัญญาณที่จุดต่างๆของวงจรโมเด็มขณะใช้งานจริง จะได้ผลการวัดระดับสัญญาณดังตารางที่ 4.2

ตารางที่ 4.2 ระดับสัญญาณที่จุดต่างๆของวงจรโมเด็มขณะใช้งานจริง

จุดที่วัด	ขณะส่งข้อมูล	ขณะรอการรับส่ง	ขณะรับข้อมูล
MC1489 (10)	+10V และ -10V	-10V	-10V
MC1489 (8)	+0.2V และ +5V	+5V	+5V
MC1489 (4)	+10V	-10V	-10V
MC1489 (6)	+0.2V	+5V	+5V
74LS28 (3)	+0.2V	+5V	+5V
MC14412 (11)	+0.2V และ +5V	+5V	+5V
MC14412 (9)	← AC $1V_{P-P}$ + DC 1.7V →		

จุดที่วัด	ขณะส่งข้อมูล	ขณะรอการรับส่ง	ขณะรับข้อมูล
MC14412 (9)ผ่านC	← AC $1V_{P-P}$ →		
MIC INPUT	AC$9mV_{P-P}$+DC4.5mV	← AC $9mV_{P-P}$ →	
74LS28 (1)	+3.4V	+Ø.1V	+Ø.1V
74LS28 (2,5,6)	+Ø.Ø5V	+Ø.Ø5V	+5V
ขา RELAY ที่ต่อกับ-	+Ø.15V	+5V	+5V
ขา C ของ Tr.			
OUTPUT ของ BUSY	+2.5V	+2.5V	-2.6V
DETECT			
SPEAKER OUTPUT	DC 2Ømv	ØV	AC $46Ømv_{P-P}$
SP.OUTPUT ผ่าน C	DC 1Ømv	ØV	AC $46Ømv_{P-P}$
FILTER OUTPUT	NOISE $6mV_{P-P}$ + DC $-3mV_{P-P}$	NOISE $6mV_{P-P}$ + DC $1mV_{P-P}$	AC $2.5V_{P-P}$
LIMITER OUTPUT	← +2.6V และ -4.9V →		
MC14412 (1)	← +5V และ +Ø.Ø2V →		
MC14412 (7)	← +5V และ +Ø.Ø2V →		
MC1488 (12,13)	+Ø.2V	+5V	+5V
MC1488 (11)	+1ØV	-1Ø.5V	-1Ø.5V
MC1488 (2)	+4.35V	+4.35V	+4.35 และ +Ø.15V
MC1488 (3)	-1Ø.5V	-1Ø.5V	-1Ø.5 และ +1ØV
74LS28 (13)	+4.35V	+4.35V	+4.35 และ +Ø.15V
74LS28(1Ø,11,12)	+Ø.15V	+Ø.15V	+Ø.15 และ +4.35V
74LS28 (4,8)	+4.35V	+4.35V	+Ø.15V

4.2 การทดลองวัดค่าต่างๆของวงจรโมเด็ม 1200 บิทต่อวินาที

4.2.1 การวัดความถี่ของสัญญาณเสียงพูดที่ได้จากวงจรโมดูเลเตอร์

สัญญาณข้อมูลเป็น "1" สัญญาณเสียงพูดที่ได้จะมีความถี่ = 1298.5 เฮิทซ์

สัญญาณข้อมูลเป็น "0" สัญญาณเสียงพูดที่ได้จะมีความถี่ = 2097.6 เฮิทซ์

4.2.2 การวัดระดับสัญญาณในการทำงานของวงจรตรวจสอบการออกอากาศ

ขณะวงจรตรวจสอบการออกอากาศ "OFF" ระดับสัญญาณที่ทำให้วงจร "ON" = 8 mV.$_{p-p}$

ขณะวงจรตรวจสอบการออกอากาศ "ON" ระดับสัญญาณที่ทำให้วงจร "OFF" = 6 mV.$_{p-p}$

4.2.3 การวัดเวลาในการตอบสัญญาณ CTS หลังจากที่คอมพิวเตอร์ส่งสัญญาณ RTS

ทำการวัดได้เช่นเดียวกับการวัดของโมเด็ม 300 บิทต่อวินาที

จะได้เวลาในการตอบสัญญาณ CTS = 270 มิลลิวินาที

4.2.4 การวัดระดับสัญญาณเสียงพูดของข้อมูลส่ง

สัญญาณเสียงพูดที่ได้จากวงจรโมดูเลเตอร์ = 4.8 V.$_{p-p}$

สัญญาณเสียงพูดที่ป้อนให้กับเครื่องส่งวิทยุ = 20 mV.$_{p-p}$

4.3 การทดลองรับส่งข้อมูลโดยใช้โมเด็ม 300 บิทต่อวินาที

4.3.1 การทดลองรับส่งข้อมูลแบบซิมเพล็กซ์ ก่อนปรับปรุงวงจรโมเด็ม

คอมพิวเตอร์ A เป็นฝ่ายส่งข้อมูล โดยใช้โปรแกรม BP1 และคอมพิวเตอร์ B เป็นฝ่ายรับข้อมูล โดยใช้โปรแกรม BP2

ก.ทดลองส่งตัวอักษร "1" เป็นจำนวน 25 ครั้ง ทั้งหมด 10 การทดลอง

รับตัวอักษร "1" ครบจำนวน 25 ครั้ง ได้ 8 การทดลอง

รับตัวอักษร "1" ได้ 18 ครั้ง แล้วเกิด "DEVICE I/O ERROR" 1 การทดลอง

รับตัวอักษร "1" ได้ 7 ครั้ง แล้วเกิด "DEVICE I/O ERROR" 1 การทดลอง

ข.ทดลองส่งตัวอักษร "ABCDEFGHIJKLMNOPQRSTUVWXYZ" เป็นจำนวน 25 ครั้ง ทั้งหมด 10 การทดลอง

รับตัวอักษร "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ครบจำนวน 25 ครั้ง ได้ 9 การทดลอง

รับตัวอักษร "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ได้ 14 ครั้ง แล้วเกิด "DEVICE I/O ERROR" 1 การทดลอง

4.3.2 การทดลองรับส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ หลังปรับปรุงวงจรโมเด็ม

คอมพิวเตอร์ A เป็นฝ่ายเริ่มส่งข้อมูล โดยใช้โปรแกรม BP3 และคอมพิวเตอร์ B เป็นฝ่ายเริ่มรับข้อมูล โดยใช้โปรแกรม BP4

ก.ทดลองส่งรับตัวอักษร "a" ไปกลับทั้งหมด 600 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดใดๆ

ข.ทดลองส่งรับตัวอักษร "abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ" ไปกลับทั้งหมด 503 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดใดๆ

ค.ทดลองส่งรับตัวอักษร "1234567890" ไปกลับทั้งหมด 551 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดใดๆ

คอมพิวเตอร์ B เป็นฝ่ายเริ่มส่งข้อมูล โดยใช้โปรแกรม BP3 และคอมพิวเตอร์ A เป็นฝ่ายเริ่มรับข้อมูล โดยใช้โปรแกรม BP4

ก.ทดลองส่งรับตัวอักษร "a" ไปกลับทั้งหมด 551 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดใดๆ

ข.ทดลองส่งรับตัวอักษร "1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" ไปกลับทั้งหมด 558 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดใดๆ

4.4 การทดลองรับส่งข้อมูลโดยใช้โมเด็ม 1200 บิทต่อวินาที

4.4.1 การทดลองรับส่งข้อมูลแบบซิมเพล็กซ์ โดยใช้ภาษา BASICA

คอมพิวเตอร์ A เป็นฝ่ายส่งข้อมูล โดยใช้โปรแกรม BP1 และคอมพิวเตอร์ B เป็นฝ่ายรับข้อมูล โดยใช้โปรแกรม BP2

ก.ทดลองส่งตัวอักษร "A" เป็นจำนวน 25 ครั้ง ทั้งหมด 10 การทดลอง

รับอักษร "A" ครบจำนวน 25 ครั้ง ได้ 9 การทดลอง

รับอักษร "A" ได้ 17 ครั้ง แล้วเกิด "DEVICE I/O ERROR" 1 การทดลอง

ข.ทดลองส่งตัวอักษร"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" เป็นจำนวน 25 ครั้ง ทั้งหมด 10 การทดลอง

รับอักษร"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" ครบจำนวน 25 ครั้ง ได้ทั้งหมด 10 การทดลอง

4.4.2 การทดลองรับส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ โดยใช้ภาษา BASICA

คอมพิวเตอร์ A เป็นฝ่ายเริ่มส่งข้อมูล โดยใช้โปรแกรม BP3 และคอมพิวเตอร์ B เป็นฝ่ายเริ่มรับข้อมูล โดยใช้โปรแกรม BP4

ก.ทดลองส่งรับตัวอักษร "A" ไปกลับทั้งหมด 25 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดของข้อมูล

ข.ทดลองส่งรับตัวอักษร "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" ไปกลับทั้งหมด 25 ครั้ง แล้วสั่งให้หยุด ไม่เกิดการผิดพลาดของข้อมูล

4.3.3 การทดลองรับส่งข้อมูลแบบซิมเพล็กซ์ โดยใช้ภาษา TURBO Pascal

คอมพิวเตอร์ A เป็นฝ่ายส่งข้อมูล โดยใช้โปรแกรม PP1 และคอมพิวเตอร์ B เป็นฝ่ายรับข้อมูล โดยใช้โปรแกรม PP2

ก.ทดลองส่งข้อมูลรหัส 0 ถึง 255 ทั้งหมด 10 การทดลอง

รับข้อมูลรหัส 0 ถึง 255 ได้ทั้งหมด 10 การทดลอง ไม่เกิดการผิดพลาดของข้อมูล

ข.ทดลองส่งข้อมูลรหัส 0 ถึง 252 ทั้งหมด 10 การทดลอง

ครั้งที่ 1 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 255,255

ครั้งที่ 2 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 253,255

ครั้งที่ 3 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 119,255

ครั้งที่ 4 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 191

ครั้งที่ 5 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 255

ครั้งที่ 6 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 245

ครั้งที่ 7 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 223,255

ครั้งที่ 8 รับได้รหัสข้อมูล 0 ถึง 252

ครั้งที่ 9 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 255,255

ครั้งที่ 10 รับได้รหัสข้อมูล 0 ถึง 252 และตามมาด้วยรหัส 255

บทที่ 5 บทวิจารณ์และสรุป

ข้อเสียอย่างหนึ่งของการสื่อสารข้อมูลผ่านระบบวิทยุแบบฮาล์ฟดูเพล็กซ์ คือการที่คอมพิวเตอร์ทั้งสองฝ่ายไม่ได้ทำการรับหรือทำการส่งตลอดเวลา โดยอาจจะมีสัญญาณรบกวนเข้าไปในวงจรโมเด็มของฝ่ายรับ ในขณะที่ฝ่ายหนึ่งกำลังจะเริ่มส่งข้อมูล หรือขณะที่อีกฝ่ายหนึ่งเพิ่งจะหยุดส่งข้อมูล ซึ่งสามารถเห็นได้จากการที่ภาษา BASICA ฟ้องการรบกวนออกมาว่าเกิด "DEVICE I/O ERROR" ทั้งๆที่ไม่ได้เกิดการผิดพลาดของข้อมูล แต่ข้อดีอย่างหนึ่งของการสื่อสารข้อมูลแบบฮาล์ฟดูเพล็กซ์ คือไม่เป็นการสิ้นเปลืองช่องสัญญาณ โดยในที่นี้หมายถึงคลื่นความถี่วิทยุ (RADIO WAVE) ถูกใช้เพียงความถี่เดียวเท่านั้น แต่การสื่อสารข้อมูลแบบฟูลดูเพล็กซ์จะต้องอาศัยคลื่นความถี่วิทยุถึง 2 ความถี่ และเป็นการยากในการเขียนโปรแกรมควบคุมให้คอมพิวเตอร์ที่มีไมโครโปรเซสเซอร์เพียงตัวเดียว ทำการสื่อสารข้อมูลแบบฟูลดูเพล็กซ์จริงๆ ถึงแม้จะทำได้ เพียงแค่การรับข้อมูลที่มีความเร็วสูงอย่างเดียว ไมโครโปรเซสเซอร์ก็อาจจะทำงานไม่ทัน ทั้งๆที่ไม่ได้ทำการส่งข้อมูลเลย ดังนั้นสำหรับคอมพิวเตอร์ที่มีไมโครโปรเซสเซอร์เพียงตัวเดียว การสื่อสารข้อมูลแบบฮาล์ฟดูเพล็กซ์จะเป็นการเหมาะสมมากกว่า

การอาศัยสัญญาณอนาล็อกเพื่อให้การส่งสัญญาณดิจิตอลสามารถจะกระทำได้ สัญญาณอนาล็อกไม่จำเป็นต้องมีความถี่อยู่ในย่านสัญญาณเสียงพูด ซึ่งจะเป็นตัวจำกัดความเร็วสูงสุดในการรับส่งสัญญาณดิจิตอล แต่ที่ใช้การแปลงสัญญาณข้อมูลให้เป็นสัญญาณเสียงพูดก็เพราะเครื่องรับส่งวิทยุที่ใช้ส่งสัญญาณเสียงพูดมีการใช้งานกันกว้างขวางอยู่แล้ว

การรับส่งข้อมูลแบบอะซิงโครนัส มีประสิทธิภาพต่ำกว่าการรับส่งแบบซิงโครนัสมาก เมื่อทำการรับส่งข้อมูลที่มีจำนวนมากๆ จะทำให้เสียเวลาในการรับส่งข้อมูลมากโดยไม่จำเป็น แต่ที่ต้องทำการส่งแบบอะซิงโครนัสเพราะ เครื่องคอมพิวเตอร์ส่วนบุคคล IBM ที่สามารถหามาใช้ทำงาน มีแต่อะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัสเท่านั้น

จากการทำงานในช่วงแรกที่อาศัยภาษา BASICA ในการเขียนโปรแกรมควบคุมการสื่อสารข้อมูล ภาษา BASICA จะฟ้องการผิดพลาดที่เกิดจากการบกวนออกมาเป็น "DEVICE I/O ERROR" ซึ่งไม่สามารถหาสาเหตุการผิดพลาดดังกล่าวได้ แต่ในการเขียนโปรแกรมภาษา TURBO Pascal นั้นสามารถที่จะตรวจหาสาเหตุได้ จะเห็นได้จากสัญญาณรบกวนในขณะที่เพิ่งจะหยุดส่งข้อมูล จะทำให้มีข้อมูลเกิดขึ้นที่ภาคดิโมดูเลเตอร์ ตามหลังข้อมูลที่แท้จริงออกมา ที่กล่าวมาแล้วนี้ คงจะเป็นประโยชน์ในการพัฒนาการสื่อสารข้อมูลต่อไป

ภาคผนวก

ก.1 โปรแกรม BP1

```
10 INPUT"DATA";A$
20 OPEN"COM1:1200,,,,DS"FOR OUTPUT AS#1
30 WRITE#1,A$
40 CLOSE
50 IF INKEY$="" THEN 50
60 GOTO 20
```

ก.2 โปรแกรม BP2

```
10 OPEN"COM1:1200,,,,RS,DS"FOR INPUT AS#1
20 INPUT#1,A$
30 PRINT A$
40 CLOSE
50 GOTO 10
```

ก.3 โปรแกรม BP3

```
10 INPUT"TRANSMITTED DATA TO RECEIVER        ";A$
20 J=1
30 OPEN"COM1:1200,,,,DS"FOR OUTPUT AS#1
40 WRITE#1,A$
50 CLOSE
60 OPEN"COM1:1200,,,,RS,DS"FOR INPUT AS#2
70 INPUT#2,B$
80 CLOSE
90 PRINT"       ";J;". RETRANSMITTED DATA FROM RECEIVER       ";B$
100 IF A$<>B$ THEN PRINT"            ERROR DATA":GOTO 150
```

```
110 J=J+1
120 A$=B$
130 IF INKEY$<>"" THEN 190
140 GOTO 30
150 OPEN"COM1:1200,,,,DS"FOR OUTPUT AS#1
160 WRITE#1,"EOF WITH ERROR"
170 CLOSE
180 END
190 OPEN"COM1:1200,,,,DS"FOR OUTPUT AS#1
200 WRITE#1,"EOF"
210 CLOSE
220 PRINT"                END OF TRANSMISSION"
230 END
```

ก.4 โปรแกรม BP4

```
10 J=1
20 OPEN"COM1:1200,,,,RS,DS"FOR INPUT AS#1
30 INPUT#1,A$
40 CLOSE
50 IF A$="EOF WITH ERROR" THEN PRINT"          ERROR DATA OCCURED":END
60 IF A$="EOF" THEN PRINT"              END OF TRANSMISSION":END
70 PRINT "        ";J;". RECEIVED DATA FROM TRANSMITTER    ";A$
80 OPEN"COM1:1200,,,,DS"FOR OUTPUT AS#2
90 WRITE#2,A$
100 CLOSE
110 J=J+1
120 GOTO 20
```

ก.5 โปรแกรม PP1

```
program   comm1;  {program to transmit data by radio}
var
   status  :byte;
   i       :integer;
procedure set commparameter;

    begin
         port[$3fb]  :=$83;          {set 8 bit word,1 stop bit,
                                      no parity bit,DLAB=1}
         port[$3f9]  :=$00;          {set Baud Rate}
         port[$3f8]  :=$60;          {=1200 bps}
         port[$3fb]  :=$03;          {set DLAB=0}
    end;

procedure RTS_CTS;

    begin
         port[$3fc]  :=$02;                {RTS}
         status      :=$00;
            while (status =$00) do
               begin
                 status   :=port[$3fe];  {read Modem Status Register}
                 status   :=status and $10; {CTS,status=$10}
               end;
    end;
```

```
procedure transmit;
  begin
    for i :=0 to 255 do
     begin
         status      :=$00;
           while (status =$00) do
             begin
               status   :=port[$3fd];     {read Line Status Register}
               status   :=status and $20; {Transmit Holding Register
                                           empty,status=$20}
             end;
         port[$3f8]  :=i;
     end;
  end;
begin
     set_commparameter;
     RTS_CTS;
     writeln('BEGIN OF TRANSMISSION');
     transmit;
     writeln('END OF TRANSMISSION');
     port[$3fc]  :=$00;                    {turn off RTS}
end.
```

ก.6 โปรแกรม PP2

```
program    comm2;   {program to receive data by radio}
var
  status   :byte;
  i        :integer;
  data     :array[0..255] of integer;
procedure set_comparameter;

    begin
        port[$3fb]  :=$83;          {set 8 bit word,1 stop bit,
                                     no parity bit,DLAB=1}
        port[$3f9]  :=$00;          {set Baud Rate}
        port[$3f8]  :=$60;          {=1200 bps}
        port[$3fb]  :=$03;          {set DLAB=0}
    end;

procedure receive;
    begin
      status  :=port[$3f8];         {clear any data receive before}
      for i := 0 to 255 do
        begin
         status  :=$00;
         while (status=$00) do
             begin
              status :=port[$3fd];       {read Line Status Register}
              status :=status and $01; {any data receive,status=$01;}
             end;
        data[i]  :=port[$3f8];
```

```
        end;
    end;

begin
     set_comparameter;
    receive;
    for i := 0 to 255 do
       begin
         write(data[i],'   ');
       end;
end.
```

ก.7 โปรแกรม COMM

```
program  comm;  {program for project DATA LINK BY RADIO}
var
  a,b          :char;
  i,j,k,l,m,n  :integer;
  m1,m2,m3,m4  :integer;
  x            :real;
  status       :byte;
  d            :byte;
  data         :string[255];
  code         :array[1..1024] of byte;
  source,dest  :file;
  sourcename   :string[14];
  destname     :string[14];
  numread      :integer;
  numwrite     :integer;

procedure  title;
  begin
    clrscr;
    gotoxy(26,3);
    write('WELCOME TO DATA LINK BY RADIO');
    gotoxy(40,6);
    write('BY');
    gotoxy(22,8);
    write('1. WICHAI      KOMENTRAKARN     28.1213');
    gotoxy(22,9);
    write('2. WITTAWAT    KHAOROPTHAM      28.1219');
    gotoxy(37,12);
    write('ADVISOR');
    gotoxy(22,14);
    write('ASSISTANT PROFESSOR  NARONG HEAMAKORN');
    gotoxy(19,17);
    write('DEPARTMENT OF TELECOMMUNICATION ENGINEERING');
    gotoxy(30,18);
    write('FACULTY OF ENGINEERING');
    gotoxy(22,19);
    write('KING MONGKUT S INSTITUTE OF TECHNOLOGY');
    gotoxy(34,19);
    write(chr(#27));
```

```
    gotoxy(32,20);
    write('LADKRABANG CAMPUS');
    gotoxy(28,23);
    write('PRESS ANY KEY TO CONTINUE');
    gotoxy(40,24);
    read(kbd,a);
  end;

procedure set_commparameter;
  begin
    port[$3fb]  :=$9f;  {set 8 bits word,2 stop bits,even parity,DLAB=1}
    port[$3f9]  :=$00;  {set baud rate =}
    port[$3f8]  :=$60;  {1200 baud}
    port[$3fb]  :=$1f;  {set DLAB=0}
  end;

procedure contact;
label    again;
  begin
    clrscr;
    port[$3fc]  :=$02;                {turn on RTS}
    gotoxy(11,3);
    write('WAITING FOR CTS ');
    status      :=$00;
    j           :=1;
    k           :=1;
    while (status<>$10) do
      begin
        status  :=port[$3fe];         {read Modem Status Register}
        status  :=status and $10;     {CTS on,status=$10}
        j :=j+1;
        if j =15000 then
          begin
            write('.');
            j :=1;
            k :=k+1;
          end;
        if k =4 then
          begin
            port[$3fc]  :=$00;                {turn off RTS}
```

```
          clrscr;
          gotoxy(11,5);
          write('CTS  DO NOT TURN ON');
          gotoxy(11,6);
          write('TRY TO COMMUNICATE AGAIN');
          gotoxy(31,17);
          write (' ');
          exit;
        end;
    end;
  gotoxy(11,4);
  write('O.K.');
  for i :=1 to 64 do
    begin
      status  :=$00;
      while (status<>$20) do
        begin
          status  :=port[$3fd];     {read Line Status Register}
          status  :=status and $20; {Transmit Holding Register
                                     empty,status=$20}
        end;
      port[$3f8]  :=$31;
    end;
  status       :=$00;
  while (status<>$60) do
    begin
      status  :=port[$3fd];     {read Line Status Register}
      status  :=status and $60; {Transmit Holding Register empty,
                                 Transmit Shift Register empty,
                                 status=$60}
    end;
  port[$3fc]  :=$00;                {turn off RTS}

  gotoxy(11,5);
  write('WAITING FOR RECEIVED CODE ');
  status       :=port[$3f8];    {clear any data receive before}
  for i := 1 to 64 do
    begin
      status  :=$00;
      j :=1;
```

```
k :=1;
while (status<>$01) do
  begin
    status  :=port[$3fd];     {read Line Status Register}
    status  :=status and $01; {any data receive,status=$01}
    j       :=j+1;
    if j =15000 then
      begin
        write('.');
        j :=1;
        k :=k+1;
      end;
   if k =4 then
     begin
       port[$3fc]  :=$00;                {turn off RTS}
       clrscr;
       gotoxy(11,5);
       write('NO CODE RECEIVED');
       gotoxy(11,6);
       write('TRY TO COMMUNICATE AGAIN');
       gotoxy(31,17);
       write (' ');
       exit;
     end;
  end;
code[i]  :=port[$3f8];
if code[i]<>$32 then
  begin
    port[$3fc]  :=$00;              {turn off RTS}
    clrscr;
    gotoxy(11,5);
    write('RECEIVED CODE ERROR');
    gotoxy(11,6);
    write('TRY TO COMMUNICATE AGAIN');
    gotoxy(31,17);
    write (' ');
    exit;
  end;
end;
gotoxy(11,6);
```

```
write('O.K.');
clrscr;
again:
write('A: ');
readln(data);
m  :=ord(data[0]);
port[$3fc]  :=$02;              {turn on RTS}
status      :=$00;
while (status<>$10) do
  begin
    status  :=port[$3fe];       {read Modem Status Register}
    status  :=status and $10;   {CTS on,status=$10}
  end;
for i :=1 to m+1 do
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];     {read Line Status Register}
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    if i <>m+1
      then
        port[$3f8]  :=ord(data[i])
      else
        port[$3f8]  :=$0d;
  end;
status      :=$00;
while (status<>$60) do
  begin
    status  :=port[$3fd];     {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
                               Transmit Shift Register empty,
                               status=$60}
  end;
port[$3fc]  :=$00;              {turn off RTS}
if data ='EOT' then
  begin
    clrscr;
```

```
        gotoxy(11,6);
        write('END OF TRANSMISSION');
        exit;
      end;
    write('B: ');
    status  :=port[$3f8];           {clear any data receive}
    n  :=1;
    d  :=$00;
    while (d<>$0d) do
      begin
        status  :=$00;
        while (status<>$01) do
          begin
            status  :=port[$3fd];     {read Line Status Register}
            status  :=status and $01; {any data receive,status=$01}
          end;
        d       :=port[$3f8];
        data[n] :=chr(d);
        n       :=n+1;
      end;
    data[0]  :=chr(n-2);
    writeln(data);
    if data ='EOT' then
      begin
        clrscr;
        gotoxy(11,6);
        write('END OF TRANSMISSION');
        exit;
      end;
    goto again;
  end;

procedure send_message;
label    again;
  begin
    clrscr;
    port[$3fc]  :=$02;              {turn on RTS}
    gotoxy(11,3);
    write('WAITING FOR CTS ');
    status      :=$00;
```

```
j           :=1;
k           :=1;
while (status<>$10) do
  begin
    status  :=port[$3fe];          {read Modem Status Register}
    status  :=status and $10;      {CTS on,status=$10}
    j :=j+1;
    if j =15000 then
      begin
        write('.');
        j :=1;
        k :=k+1;
      end;
    if k =4 then
      begin
        port[$3fc]  :=$00;                  {turn off RTS}
        clrscr;
        gotoxy(11,5);
        write('CTS  DO NOT TURN ON');
        gotoxy(11,6);
        write('TRY TO COMMUNICATE AGAIN');
        gotoxy(31,17);
        write (' ');
        exit;
      end;
  end;
gotoxy(11,4);
write('O.K.');
for i :=1 to 64 do
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    port[$3f8]  :=$33;
  end;
status      :=$00;
```

```
while (status<>$60) do
  begin
    status  :=port[$3fd];      {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
                               Transmit Shift Register empty,
                               status=$60}
  end;
port[$3fc]  :=$00;              {turn off RTS}

gotoxy(11,5);
write('WAITING FOR RECEIVED CODE ');
status      :=port[$3f8];    {clear any data receive before}
for i := 1 to 64 do
  begin
    status  :=$00;
    j :=1;
    k :=1;
    while (status<>$01) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $01; {any data receive,status=$01}
        j       :=j+1;
        if j =15000 then
          begin
            write('.');
            j :=1;
            k :=k+1;
          end;
       if k =4 then
         begin
           port[$3fc]  :=$00;                {turn off RTS}
           clrscr;
           gotoxy(11,5);
           write('NO CODE RECEIVED');
           gotoxy(11,6);
           write('TRY TO COMMUNICATE AGAIN');
           gotoxy(31,17);
           write (' ');
           exit;
         end;
```

```
        end;
      code[i]  :=port[$3f8];
      if code[i]<>$34 then
        begin
          port[$3fc]  :=$00;                {turn off RTS}
          clrscr;
          gotoxy(11,5);
          write('RECEIVED CODE ERROR');
          gotoxy(11,6);
          write('TRY TO COMMUNICATE AGAIN');
          gotoxy(31,17);
          write (' ');
          exit;
        end;
    end;
  gotoxy(11,6);
  write('O.K.');
  clrscr;
  writeln('MESSAGE :');
  again:
  readln(data);
  m  :=ord(data[0]);
  port[$3fc]  :=$02;                {turn on RTS}
  status      :=$00;
  while (status<>$10) do
    begin
      status  :=port[$3fe];        {read Modem Status Register}
      status  :=status and $10;    {CTS on,status=$10}
    end;
  for i :=1 to m+1 do
    begin
      status  :=$00;
      while (status<>$20) do
        begin
          status  :=port[$3fd];     {read Line Status Register}
          status  :=status and $20; {Transmit Holding Register
                                     empty,status=$20}
        end;
      if i <>m+1
      then
```

```
            port[$3f8]  :=ord(data[i])
          else
            port[$3f8]  :=$0d;
        end;
      status       :=$00;
      while (status<>$60) do
        begin
          status  :=port[$3fd];      {read Line Status Register}
          status  :=status and $60; {Transmit Holding Register empty,
                                     Transmit Shift Register empty,
                                     status=$60}
        end;
      port[$3fc]  :=$00;                {turn off RTS}
      if data ='EOT' then
       begin
          clrscr;
          gotoxy(11,6);
          write('END OF TRANSMISSION');
          exit;
        end;
      delay (1500);
      goto again;
    end;

  procedure tran_to;
  var
   buffer       :array[1..16384] of byte;
    begin
      clrscr;
      port[$3fc]  :=$02;                {turn on RTS}
      gotoxy(11,3);
      write('WAITING FOR CTS ');
      status       :=$00;
      j            :=1;
      k            :=1;
      while (status<>$10) do
        begin
          status  :=port[$3fe];         {read Modem Status Register}
          status  :=status and $10;     {CTS on,status=$10}
          j :=j+1;
```

```
      if j =15000 then
        begin
          write('.');
          j :=1;
          k :=k+1;
        end;
      if k =4 then
        begin
          port[$3fc]  :=$00;                {turn off RTS}
          clrscr;
          gotoxy(11,5);
          write('CTS  DO NOT TURN ON');
          gotoxy(11,6);
          write('TRY TO COMMUNICATE AGAIN');
          gotoxy(31,17);
          write (' ');
          exit;
        end;
    end;
  gotoxy(11,4);
  write('O.K.');
  for i :=1 to 64 do
    begin
      status  :=$00;
      while (status<>$20) do
        begin
          status  :=port[$3fd];      {read Line Status Register}
          status  :=status and $20; {Transmit Holding Register
                                     empty,status=$20}
        end;
      port[$3f8]  :=$35;
    end;
  status      :=$00;
  while (status<>$60) do
    begin
      status  :=port[$3fd];      {read Line Status Register}
      status  :=status and $60; {Transmit Holding Register empty,
                                 Transmit Shift Register empty,
                                 status=$60}
    end;
```

```
port[$3fc]  :=$00;                {turn off RTS}

gotoxy(11,5);
write('WAITING FOR RECEIVED CODE ');
status      :=port[$3f8];     {clear any data receive before}
for i := 1 to 64 do
  begin
    status  :=$00;
    j :=1;
    k :=1;
    while (status<>$01) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $01; {any data receive,status=$01}
        j       :=j+1;
        if j =15000 then
          begin
            write('.');
            j :=1;
            k :=k+1;
          end;
        if k =4 then
          begin
            port[$3fc]  :=$00;                {turn off RTS}
            clrscr;
            gotoxy(11,5);
            write('NO CODE RECEIVED');
            gotoxy(11,6);
            write('TRY TO COMMUNICATE AGAIN');
            gotoxy(31,17);
            write (' ');
            exit;
          end;
      end;
    code[i]  :=port[$3f8];
    if code[i]<>$36 then
      begin
        port[$3fc]  :=$00;                {turn off RTS}
        clrscr;
        gotoxy(11,5);
```

```
          write('RECEIVED CODE ERROR');
          gotoxy(11,6);
          write('TRY TO COMMUNICATE AGAIN');
          gotoxy(31,17);
          write (' ');
          exit;
        end;
    end;
gotoxy(11,6);
write('O.K.');
clrscr;
write ('TRANSFER FILE                    TO ANOTHER COMPUTR');
gotoxy(15,1);
readln(sourcename);
assign(source,sourcename);
reset(source,1);
m  :=ord(sourcename[0]);
port[$3fc]  :=$02;                    {turn on RTS}
status      :=$00;
while (status<>$10) do
  begin
    status  :=port[$3fe];             {read Modem Status Register}
    status  :=status and $10;         {CTS on,status=$10}
  end;
for i :=1 to m+1 do
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];         {read Line Status Register}
        status  :=status and $20;     {Transmit Holding Register
                                       empty,status=$20}
      end;
    if i <>m+1
      then
        port[$3f8]  :=ord(sourcename[i])
      else
        port[$3f8]  :=$0d;
  end;
delay (300);
```

```
    repeat
      blockread(source,buffer,16384,numread);
      port[$3fc]  :=$02;                {turn on RTS}
      status      :=$00;
      while (status<>$10) do
        begin
          status  :=port[$3fe];         {read Modem Status Register}
          status  :=status and $10;     {CTS on,status=$10}
        end;
     . if numread<>0 then
        begin
          for i :=1 to numread do
            begin
              status  :=$00;
              while (status<>$20) do
                begin
                  status  :=port[$3fd];     {read Line Status Register}
                  status  :=status and $20; {Transmit Holding Register
                                             empty,status=$20}
                end;
              port[$3f8]  :=buffer[i];
            end;
        end;
    until numread =0;
      status      :=$00;
      while (status<>$60) do
        begin
          status  :=port[$3fd];     {read Line Status Register}
          status  :=status and $60; {Transmit Holding Register empty,
                                     Transmit Shift Register empty,
                                     status=$60}
        end;
      delay (5000);
      port[$3fc]  :=$00;                {turn off RTS}
      clrscr;
      gotoxy(11,6);
      write('END OF TRANSFER FILE');
      close(source);
  end;
```

```
procedure tran_from;
var
 buffer1      :array[1..3,1..16384] of byte;
label    wrt;
  begin
    clrscr;
    port[$3fc]  :=$02;                {turn on RTS}
    gotoxy(11,3);
    write('WAITING FOR CTS ');
    status      :=$00;
    j           :=1;
    k           :=1;
    while (status<>$10) do
      begin
        status  :=port[$3fe];         {read Modem Status Register}
        status  :=status and $10;     {CTS on,status=$10}
        j :=j+1;
        if j =15000 then
          begin
            write('.');
            j :=1;
            k :=k+1;
          end;
        if k =4 then
          begin
            port[$3fc]  :=$00;                {turn off RTS}
            clrscr;
            gotoxy(11,5);
            write('CTS  DO NOT TURN ON');
            gotoxy(11,6);
            write('TRY TO COMMUNICATE AGAIN');
            gotoxy(31,17);
            write (' ');
            exit;
          end;
      end;
    gotoxy(11,4);
    write('O.K.');
    for i :=1 to 64 do
      begin
```

```
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    port[$3f8]  :=$37;
  end;
status      :=$00;
while (status<>$60) do
  begin
    status  :=port[$3fd];      {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
                               Transmit Shift Register empty,
                               status=$60}
  end;
port[$3fc]  :=$00;                {turn off RTS}

gotoxy(11,5);
write('WAITING FOR RECEIVED CODE ');
status      :=port[$3f8];    {clear any data receive before}
for i := 1 to 64 do
  begin
    status  :=$00;
    j :=1;
    k :=1;
    while (status<>$01) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $01; {any data receive,status=$01}
        j       :=j+1;
        if j =15000 then
          begin
            write('.');
            j :=1;
            k :=k+1;
          end;
       if k =4 then
        begin
```

```
            port[$3fc]  :=$00;                {turn off RTS}
            clrscr;
            gotoxy(11,5);
            write('NO CODE RECEIVED');
            gotoxy(11,6);
            write('TRY TO COMMUNICATE AGAIN');
            gotoxy(31,17);
            write (' ');
            exit;
          end;
      end;
    code[i]  :=port[$3f8];
    if code[i]<>$38 then
      begin
        port[$3fc]  :=$00;                {turn off RTS}
        clrscr;
        gotoxy(11,5);
        write('RECEIVED CODE ERROR');
        gotoxy(11,6);
        write('TRY TO COMMUNICATE AGAIN');
        gotoxy(31,17);
        write (' ');
        exit;
      end;
  end;
gotoxy(11,6);
write('O.K.');
clrscr;
write ('TRANSFER FILE                FROM ANOTHER COMPUTR');
gotoxy(15,1);
readln(destname);
m  :=ord(destname[0]);
port[$3fc]  :=$02;                {turn on RTS}
status      :=$00;
while (status<>$10) do
  begin
    status  :=port[$3fe];         {read Modem Status Register}
    status  :=status and $10;     {CTS on,status=$10}
  end;
for i :=1 to m-1 do
```

```
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    if i <>m+1
      then
        port[$3f8]  :=ord(destname[i])
      else
        port[$3f8]  :=$0d;
  end;
status       :=$00;
while (status<>$60) do
  begin
    status  :=port[$3fd];      {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
                               Transmit Shift Register empty,
                               status=$60}
  end;
port[$3fc]  :=$00;                {turn off RTS}
status  :=port[$3f8];             {clear any data receive}
for i:=1 to 3 do
  begin
    for j:=1 to 16384 do
      begin
        k  :=1;
        l  :=1;
        status   :=$00;
        while (status<>$01) do
          begin
            status  :=port[$3fd];      {read Line Status Register}
            status  :=status and $01; {any data receive,status=$01}
            k       :=k+1;
            if k =1500 then
              begin
                k   :=1;
                l   :=l+1;
```

```
                end;
              if (l =65) then goto wrt;
            end;
          buffer1[i,j]   :=port[$3f8];
        end;
      end;
    wrt:
    assign(dest,destname);
    rewrite(dest,1);
    n  :=(i-1)*16384+(j-1);
    repeat
      blockwrite(dest,buffer1,n,numwrite);
    until numwrite=n;
    clrscr;
    gotoxy(11,6);
    write('END OF TRANSFER FILE');
    close(dest);
  end;


procedure contact_receive;
label    again;
  begin
    clrscr;
    port[$3fc]  :=$02;                {turn on RTS}
    gotoxy(11,3);
    write('WAITING FOR CTS ');
    status      :=$00;
    j           :=1;
    k           :=1;
    while (status<>$10) do
      begin
        status  :=port[$3fe];         {read Modem Status Register}
        status  :=status and $10;     {CTS on,status=$10}
        j :=j+1;
        if j =15000 then
          begin
            write('.');
            j :=1;
            k :=k+1;
```

```
        end;
      if k =4 then
        begin
          port[$3fc]  :=$00;                {turn off RTS}
          clrscr;
          gotoxy(11,5);
          write('CTS  DO NOT TURN ON');
          gotoxy(11,6);
          write('TRY TO COMMUNICATE AGAIN');
          gotoxy(31,17);
          write (' ');
          exit;
        end;
    end;
  gotoxy(11,4);
  write('O.K.');
  for i :=1 to 64 do
    begin
      status  :=$00;
      while (status<>$20) do
        begin
          status  :=port[$3fd];     {read Line Status Register}
          status  :=status and $20; {Transmit Holding Register
                                     empty,status=$20}
        end;
      port[$3f8]  :=$32;
    end;
  status      :=$00;
  while (status<>$60) do
    begin
      status  :=port[$3fd];     {read Line Status Register}
      status  :=status and $60; {Transmit Holding Register empty,
                                 Tranamit Shift Register empty,
                                 status=$60}
    end;
  port[$3fc]  :=$00;                {turn off RTS}

  clrscr;
  again:
  write('A: ');
```

```
status  :=port[$3f8];           {clear any data receive}
n  :=1;
d  :=$00;
while (d<>$0d) do
  begin
    status  :=$00;
    while (status<>$01) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $01; {any data receive,status=$01}
      end;
    d       :=port[$3f8];
    data[n] :=chr(d);
    n       :=n+1;
  end;
data[0]  :=chr(n-2);
writeln(data);
if data ='EOT' then
  begin
    clrscr;
    gotoxy(11,6);
    write('END OF TRANSMISSION');
    exit;
  end;
write('B: ');
readln(data);
m  :=ord(data[0]);
port[$3fc]  :=$02;                {turn on RTS}
status      :=$00;
while (status<>$10) do
  begin
    status  :=port[$3fe];         {read Modem Status Register}
    status  :=status and $10;     {CTS on,status=$10}
  end;
for i :=1 to m+1 do
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
```

```
            status  :=status and $20; {Transmit Holding Register
                                        empty,status=$20}
          end;
        if i <>m+1
          then
            port[$3f8]  :=ord(data[i])
          else
            port[$3f8]  :=$0d;
      end;
    status      :=$00;
    while (status<>$60) do
      begin
        status  :=port[$3fd];     {read Line Status Register}
        status  :=status and $60; {Transmit Holding Register empty,
                                   Transmit Shift Register empty,
                                   status=$60}
      end;
    port[$3fc]  :=$00;                {turn off RTS}
    if data ='EOT' then
      begin
        clrscr;
        gotoxy(11,6);
        write('END OF TRANSMISSION');
        exit;
      end;
    goto again;
  end;

procedure send_message_receive;
label    again;
  begin
    clrscr;
    port[$3fc]  :=$02;                {turn on RTS}
    gotoxy(11,3);
    write('WAITING FOR CTS ');
    status      :=$00;
    j           :=1;
    k           :=1;
    while (status<>$10) do
      begin
```

```
    status  :=port[$3fe];          {read Modem Status Register}
    status  :=status and $10;      {CTS on,status=$10}
    j :=j+1;
    if j =15000 then
      begin
        write('.');
        j :=1;
        k :=k+1;
      end;
    if k =4 then
      begin
        port[$3fc]  :=$00;                  {turn off RTS}
        clrscr;
        gotoxy(11,5);
        write('CTS  DO NOT TURN ON');
        gotoxy(11,6);
        write('TRY TO COMMUNICATE AGAIN');
        gotoxy(31,17);
        write (' ');
        exit;
      end;
  end;
gotoxy(11,4);
write('O.K.');
for i :=1 to 64 do
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    port[$3f8]  :=$34;
  end;
status      :=$00;
while (status<>$60) do
  begin
    status  :=port[$3fd];      {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
```

```
                                    Transmit Shift Register empty,
                                    status=$60}
      end;
    port[$3fc]  :=$00;                {turn off RTS}

    clrscr;
    writeln('MESSAGE :');
    again:
    status  :=port[$3f8];             {clear any data receive}
    n  :=1;
    d  :=$00;
    while (d<>$0d) do
      begin
        status  :=$00;
        while (status<>$01) do
          begin
            status  :=port[$3fd];     {read Line Status Register}
            status  :=status and $01; {any data receive,status=$01}
          end;
        d       :=port[$3f8];
        data[n] :=chr(d);
        n       :=n+1;
      end;
    data[0]  :=chr(n-2);
    writeln(data);
    if data ='EOT' then
      begin
        clrscr;
        gotoxy(11,6);
        write('END OF TRANSMISSION');
        exit;
      end;
    delay (1500);
    goto again;
  end;

procedure tran_to_receive;
var
  buffer1     :array[1..3,1..16384] of byte;
label     wrt;
```

```
begin
  clrscr;
  port[$3fc]  :=$02;                {turn on RTS}
  gotoxy(11,3);
  write('WAITING FOR CTS ');
  status      :=$00;
  j           :=1;
  k           :=1;
  while (status<>$10) do
    begin
      status  :=port[$3fe];         {read Modem Status Register}
      status  :=status and $10;     {CTS on,status=$10}
      j :=j+1;
      if j =15000 then
        begin
          write('.');
          j :=1;
          k :=k+1;
        end;
      if k =4 then
        begin
          port[$3fc]  :=$00;                {turn off RTS}
          clrscr;
          gotoxy(11,5);
          write('CTS  DO NOT TURN ON');
          gotoxy(11,6);
          write('TRY TO COMMUNICATE AGAIN');
          gotoxy(31,17);
          write (' ');
          exit;
        end;
    end;
  gotoxy(11,4);
  write('O.K.');
  for i :=1 to 64 do
    begin
      status  :=$00;
      while (status<>$20) do
        begin
          status  :=port[$3fd];     {read Line Status Register}
```

```
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    port[$3f8]  :=$36;
  end;
status      :=$00;
while (status<>$60) do
  begin
    status  :=port[$3fd];      {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
                               Transmit Shift Register empty,
                               status=$60}
  end;
port[$3fc]  :=$00;                {turn off RTS}

clrscr;
write('RECEIVE FILE                FROM ANOTHER COMPUTER');
gotoxy(14,1);
status  :=port[$3f8];              {clear any data receive}
n  :=1;
d  :=$00;
while (d<>$0d) do
  begin
    status  :=$00;
    while (status<>$01) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $01; {any data receive,status=$01}
      end;
    d           :=port[$3f8];
    destname[n] :=chr(d);
    n           :=n+1;
  end;
destname[0]     :=chr(n-2);
writeln(destname);
for i:=1 to 3 do
  begin
    for j:=1 to 16384 do
      begin
        k  :=1;
```

```
            l  :=1;
            status   :=$00;
            while (status<>$01) do
            ' begin
                status  :=port[$3fd];      {read Line Status Register}
                status  :=status and $01; {any data receive,status=$01}
                k       :=k+1;
                if k =1500 then
                  begin
                    k   :=1;
                    l   :=l+1;
                  end;
                if l =22 then goto wrt;
              end;
            buffer1[i,j]   :=port[$3f8];
          end;
      end;
    wrt:
    assign(dest,destname);
    rewrite(dest,1);
    n  :=(i-1)*16384+(j-1);
    repeat
      blockwrite(dest,buffer1,n,numwrite);
    until numwrite=n;
    clrscr;
    gotoxy(11,6);
    write('END OF RECEIVE FILE');
    close(dest);
  end;

procedure tran_from_receive;
var
 buffer       :array[1..16383] of byte;
  begin
    clrscr;
    port[$3fc]  :=$02;                  {turn on RTS}
    gotoxy(11,3);
    write('WAITING FOR CTS ');
    status      :=$00;
    j           :=1;
```

```
k           :=1;
while (status<>$10) do
  begin
    status  :=port[$3fe];          {read Modem Status Register}
    status  :=status and $10;      {CTS on,status=$10}
    j :=j+1;
    if j =15000 then
      begin
        write('.');
        j :=1;
        k :=k+1;
      end;
    if k =4 then
      begin
        port[$3fc]  :=$00;                {turn off RTS}
        clrscr;
        gotoxy(11,5);
        write('CTS  DO NOT TURN ON');
        gotoxy(11,6);
        write('TRY TO COMMUNICATE AGAIN');
        gotoxy(31,17);
        write (' ');
        exit;
      end;
  end;
gotoxy(11,4);
write('O.K.');
for i :=1 to 64 do
  begin
    status  :=$00;
    while (status<>$20) do
      begin
        status  :=port[$3fd];      {read Line Status Register}
        status  :=status and $20; {Transmit Holding Register
                                   empty,status=$20}
      end;
    port[$3f8]  :=$38;
  end;
status      :=$00;
while (status<>$60) do
```

```
  begin
    status  :=port[$3fd];      {read Line Status Register}
    status  :=status and $60; {Transmit Holding Register empty,
                               Transmit Shift Register empty,
                               status=$60}
  end;
port[$3fc]  :=$00;               {turn off RTS}

clrscr;
write('TRANSMIT FILE                    TO ANOTHER COMPUTER');
gotoxy(15,1);
status  :=port[$3f8];            {clear any data receive}
n  :=1;
d  :=$00;
while (d<>$0d) do
  begin
    status  :=$00;
    while (status<>$01) do
      begin
        status  :=port[$3fd];     {read Line Status Register}
        status  :=status and $01; {any data receive,status=$01}
      end;
    d              :=port[$3f8];
    sourcename[n] :=chr(d);
    n              :=n+1;
  end;
sourcename[0]     :=chr(n-2);
writeln(sourcename);
assign(source,sourcename);
reset(source,1);
port[$3fc]  :=$02;               {turn on RTS}
status      :=$00;
while (status<>$10) do
  begin
    status  :=port[$3fe];        {read Modem Status Register}
    status  :=status and $10;    {CTS on,status=$10}
  end;
repeat
  blockread(source,buffer,16384,numread);
  port[$3fc]  :=$02;                {turn on RTS}
```

```
      status      :=$00;
      while (status<>$10) do
        begin
          status  :=port[$3fe];          {read Modem Status Register}
          status  :=status and $10;      {CTS on,status=$10}
        end;
      if numread<>0 then
        begin
          for i :=1 to numread do
            begin
              status  :=$00;
              while (status<>$20) do
                begin
                  status  :=port[$3fd];      {read Line Status Register}
                  status  :=status and $20; {Transmit Holding Register
                                             empty,status=$20}
                end;
              port[$3f8]  :=buffer[i];
            end;
        end;
    until numread =0;
      status      :=$00;
      while (status<>$60) do
        begin
          status  :=port[$3fd];      {read Line Status Register}
          status  :=status and $60; {Transmit Holding Register empty,
                                     Transmit Shift Register empty,
                                     status=$60}
        end;
      delay (10000);
      port[$3fc]  :=$00;                 {turn off RTS}
      clrscr;
      gotoxy(11,6);
      write('END OF TRANSMIT FILE');
      close(source);
  end;


procedure stand_by;
  begin
```

```
clrscr;
gotoxy(11,5);
write('WAITING FOR RECEIVED CODE ');
status :=port[$3f8];    {clear any data receive before}
k      :=1;
m      :=1;
m1     :=1;
m2     :=1;
m3     :=1;
m4     :=1;
while (m<>65) do
  begin
    status  :=$00;
    j :=1;
    while (status<>$01) do
     begin
       status  :=port[$3fd];     {read Line Status Register}
       status  :=status and $01; {any data receive,status=$01}
       j       :=j+1;
       if j =15000 then
         begin
           write('.');
           j :=1;
         end;
      end;
    code[k]   :=port[$3f8];
    if code[k] =$31 then m1  :=m1+1;
          if m1 =65 then m   :=m1;
    if code[k] =$33 then m2  :=m2+1;
          if m2 =65 then m   :=m2;
    if code[k] =$35 then m3  :=m3+1;
          if m3 =65 then m   :=m3;
    if code[k] =$37 then m4  :=m4+1;
          if m4 =65 then m   :=m4;
    k  :=k+1;
  end;
if m1 =65 then contact_receive;
if m2 =65 then send_message_receive;
if m3 =65 then tran_to_receive;
if m4 =65 then tran_from_receive;
```

```
  end;

procedure menu;
  begin
    repeat
      gotoxy(11,8);
      write('MAIN MENU OF DATA LINK BY RADIO');
      gotoxy(11,10);
      write('1. CONTACT');
      gotoxy(11,11);
      write('2. SEND MESSAGE');
      gotoxy(11,12);
      write('3. TRANSFER FILE TO ANOTHER COMPUTER');
      gotoxy(11,13);
      write('4. TRANSFER FILE FROM ANOTHER COMPUTER');
      gotoxy(11,14);
      write('5. STAND BY');
      gotoxy(11,15);
      write('6. QUIT');
      gotoxy(11,17);
      write('PLEASE SELECT (1-6)');
      gotoxy(31,17);
      if a<>'5' then read(kbd,a);
      case a of
        '1':contact;
        '2':send_message;
        '3':tran_to;
        '4':tran_from;
        '5':stand_by;
      end;
    until a='6'
  end;


BEGIN
  TITLE;
  SET_COMMPARAMETER;
  clrscr;
  MENU;
END.
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำเร็จขึ้นมาได้ด้วยความอนุเคราะห์ของหลายๆท่าน ที่ได้ให้คำปรึกษาและความช่วยเหลือในด้านต่างๆ จึงขอขอบคุณมา ณ.กิตติกรรมประกาศนี้

ขอขอบคุณ ผศ.ณรงค์ เหมกรณ์ และ อาจารย์ในภาคโทรคมนาคมทุกท่านที่ให้คำปรึกษา , พี่วารี , น้องวิสุทธินันท์ที่ให้ยืมคอมพิวเตอร์ รวมทั้งเพื่อนๆทุกคนในภาคฯ

ผู้จัดทำ

หนังสืออ้างอิง

1. น.ต.ดร.ไพศาล สงวนหมู่ และรศ.ยืน ภู่วรวรรณ, "การสื่อสารข้อมูล และไมโครคอมพิวเตอร์เนตเวิร์ค", 243 หน้า, 2529.

2. WILLIAM SINNEMA & TOM McGOVERN, "DIGITAL, ANALOG,AND DATA COMMUNICATION", PRENTICE-HALL, 524 p., 1986.

3. WAYNE TOMASI, "ADVANCED ELECTRONIC COMMUNICATIONS SYSTEMS", PRENTICE-HALL, 373 p.,1987.

4. PAUL BATES,P.ENG , "PRACTICAL DIGITAL AND DATA COMMUNICATIONS", PRENTICE-HALL, 246 p., 1987.

5. BORLAND INTERNATIONAL,"TURBO PASCAL VERSION 3.Ø REFERENCE MANUAL", BORLAND INTERNATIONAL, 376 p., 1985.

6. MICROSOFT CORP., "BASIC", IBM PERSONAL COMPUTER HARDWARE REFERENCE LIBRARY, 426 p., 1982.

7. IBM CORP., "TECHNICAL REFERENCE", IBM PERSONAL COMPUTER HARDWARE REFERENCE LIBRARY, 6Ø7 p., 1983.