รายการอ้างอิง

1. J.Michael Jacob. Industrial Control Electronics . Prentice-Hall International Editions,1989.
2. International Business Machine Corporation . Synchronous Data Link Control . 1986.
3. William Stallings,Ph.D. , Handbook of Computer-Communications Standard . Volume 1 & Volume 2 , Macmillan Publishing Company , 1987.
4. Richard W. Markley , Data Communications and Interoperability . Prentice -Hall, Inc,1990.
5. Philips Data Systems , High level Data Link Control . 1983.
6. Tomasi , Vincent F. Alisouskas. , Telecommunications Prentice Hall , 1988.
7. คู่มือไอซีซิพซัพพอร์ทและหน่วยความจำ. บริษัท ซีเอ็ดยูเคชั่น จำกัด , 2529.
8. 8-Bit Embedded Controller Handbook.Santa Clara,USA:Intel Corporation, 1990.
9. PAL Device Data Book. Santa Clara,USA : Advanced Micro Devices,Inc.,1982.
10. REFERANCE MANUAL. California USA :FRANKLIN SOFTWARE,INC.,1987-1990.
11. ICE-51FX/PC In-Circuit Emulator. USA : INTEL CORPORATION, 1990.
12. Embedded Control Application Handbook.Santa Clara,USA:Intel Corporation, 1990.
13. Linear Data Book.Santa Clara,California,USA:Nation Semiconduction Corporation, 1990.
14. Linear Application Handbook. Santa Clara,California,USA:Nation Semiconduction Corporation, 1990.
15. Weber,Samuel.Circuit for Electronics Engineering. NEW YORK : McGraww Hill, Inc.,1977.
16. Kenneth L.Short.Microprocessors and Programmed Logic. India:Prentice_Hall of India Private Limited,1988.
17. PC/AT Technical Referance.IBM Corp.Personal Computer,1984.
18. Douglas V.Hall.Microprocessor and Interface Programming and Hardware. NEW YORK : McGraww Hill, Inc.,1992

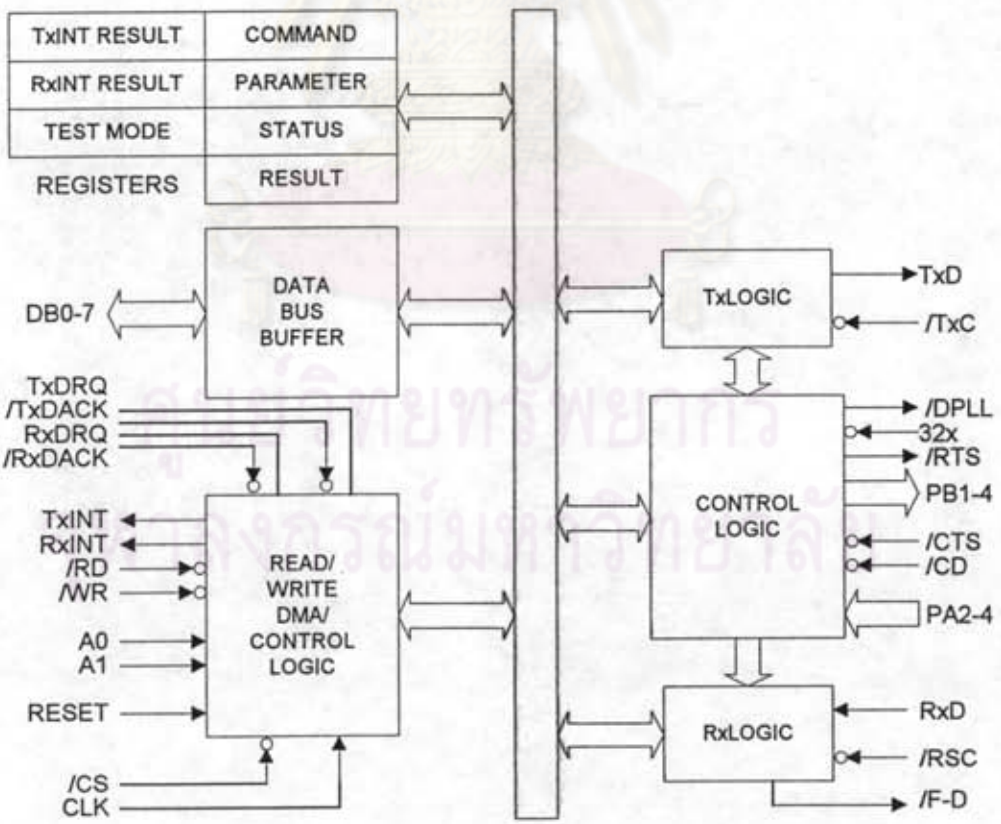
ภาคผนวก

ภาคผนวก ก.

การใช้งานชิพควบคุมโปรโตคอลเอชดีแอลซี

ชิพควบคุมโปรโตคอลเอชดีแอลซีเบอร์ Intel 8273 เป็นชิพรองรับมาตรฐานโปรโตคอลแบบเอชดีแอลซีหรือเอสดีแอลซี (HDLC/SDLC Compatible) มีข้อดีในการใช้งานมาก เช่น สามารถให้การคำนวนและตรวจสอบฟิลด์ตรวจสอบลำดับเฟรม (FCS) อย่างอัตโนมัติ, สามารถโปรแกรมให้มีการส่งเข้ารหัสข้อมูล NRZI และมีการรับถอดรหัสข้อมูล NRZI , มีส่วนดิจิตอลเฟสล็อคลูป (DPLL) ช่วยในการรับข้อมูล เป็นต้น

ส่วนประกอบภายใน 8273



รูปที่ ก.1 แสดงบล็อกไดอะแกรมส่วนต่าง ๆ ภายใน 8273

รูปที่ ก.1 แสดงบล็อกไดอะแกรมของส่วนต่าง ๆ ภายใน 8273 แบ่งเป็น 2 ส่วนหลักคือ ส่วนเชื่อมโยงกับซีพียู และส่วนเชื่อมโยงกับโมเดม

ส่วนเชื่อมโยงกับซีพียู ในส่วนนี้ประกอบด้วยส่วนสำคัญ 3 ส่วนคือ

1. รีจิสเตอร์ภายใน 7 ชุด
2. บัฟเฟอร์บัสข้อมูล
3. โลจิกควบคุมการอ่าน,การเขียนและการถ่ายเทข้อมูล

1. รีจีสเตอร์ภายใน มีด้วยกัน 7 ชุด ซึ่งมีหน้าที่สำคัญในการทำงานของตัวควบคุมโปรโตคอล โดยรีจีสเตอร์แต่ละชุดมีหน้าที่ต่างกันดังแสดงในตารางที่ ก.1 มีรายละเอียดดังนี้

Command : เป็นรีจีสเตอร์รับคำสั่งจากซีพียู

Parameter : เป็นพารามิเตอร์ของ Command ซึ่งบางคำสั่งต้องการรีจิสเตอร์ส่วนนี้

Result : เป็นรีจีสเตอร์ผลลัพธ์ ใช้สำหรับเก็บผลลัพธ์จากการทำงานคำสั่งหนึ่ง ๆ

TxINT Result : เป็นรีจีสเตอร์แสดงผลลัพธ์การส่ง

RxINT Result : เป็นรีจิสเตอร์ แสดงผลลัพธ์การรับ

Status : เป็นรีจิสเตอร์เก็บสถานะการทำงานของ 8273

A1	A0	/TxDACK	/RxDACK	/CS	/RD	/WR	Register
0	0	1	1	0	1	0	Command
0	0	1	1	0	0	1	Status
0	1	1	1	0	1	0	Parameter
0	1	1	1	0	0	1	Result
1	0	1	1	0	1	0	Reset
1	0	1	1	0	0	1	TxINT
1	1	1	1	0	1	0	-
1	1	1	1	0	0	1	RxINT
*	*	0	1	1	1	0	Transmit Data
*	*	1	0	1	0	1	Receive Data

ตารางที่ ก.1 แสดงรีจิสเตอร์ทั้งหมดใน 8273

2.บัฟเฟอร์บัสข้อมูล เป็นบัฟเฟอร์ข้อมูลระหว่างบัสข้อมูลของระบบส่วนซีพียูกับบัสข้อมูลใน 8273

3. โลจิกควบคุมการอ่าน,การเขียนและการถ่ายเทข้อมูล

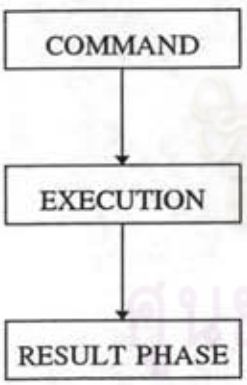
ขาสัญญาณ /RD และ /WR ใช้กำหนดทิศทางในการถ่ายเทข้อมูล ซึ่งการถ่ายเทข้อมูลสามารถกระทำได้ 2 วิธีคือ วิธีการถ่ายเทข้อมูลโดย DMA (Direct Memory Access) และวิธีการถ่ายเทข้อมูลโดยอินเทอร์รัพต์ ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อต่อ ๆ ไป

ส่วนเชื่อมโยงโมเดม ผู้ใช้สามารถเลือกได้ว่าจะใช้ฟังก์ชันการควบคุมโมเดมหรือไม่ ซึ่งสามารถใช้ได้กับมาตรฐานของโมเดมทั่วไป ผู้วิจัยไม่ขอกล่าวถึงรายละเอียดในส่วนนี้ เนื่องจากงานวิจัยนี้ไม่ได้ใช้ฟังก์ชันการควบคุมโมเดม

การทำงานในแต่ละคำสั่งของ 8273 จะต้องประกอบด้วยลำดับเฟส 3 ลำดับ คือ เฟสคำสั่ง, เฟสปฏิบัติการ และเฟสผลลัพธ์ แสดงดังรูปที่ ก.2

<u>เฟสคำสั่ง</u>

ขณะอยู่ในเฟสคำสั่ง เมื่อมีการเขียนคำสั่งไปยังรีจิสเตอร์คำสั่ง จะต้องรู้ชนิดของการทำงานของคำสั่งนั้น ว่าจะต้องเขียนพารามิเตอร์เพิ่มเติมให้หรือไม่ ซึ่งบางคำสั่งต้องการพารามิเตอร์ที่ตามมาถึง 4 พารามิเตอร์



ซีพียูเขียนคำสั่งและพารามิเตอร์ไปยังรีจิสเตอร์คำสั่งและพารามิเตอร์ของ 8273

เมื่อเฟสคำสั่งเสร็จสมบูรณ์ 8273 จะทำงานในเฟส

เมื่อเฟสปฏิบัติการเสร็จสมบูรณ์ ซีพียูจะต้องอ่านรีจิสเตอร์ส่วนผลลัพธ์

รูปที่ ก.2 แสดงลำดับเฟสปฏิบัติการของ 8273

รายละเอียดของรีจิสเตอร์สถานะ มีรูปแบบและความหมายดังนี้

CBSY	CBF	CPBF	CRBF	RxINT	TxINT	RxIRA	TxIRA

บิต 7 CBSY (Command Busy) หากมีการเขียนคำสั่งในรีจิสเตอร์คำสั่ง บิตนี้จะถูกเซต และเมื่อเฟสคำสั่งเสร็จสมบูรณ์ บิตนี้จะถูกรีเซต

บิต 6 CBF (Command Buffer Full) หาก 8273 ยังไม่สามารถรับคำสั่ง บิตนี้จะถูกเซตเมื่อ 8273 สามารถรับคำสั่งได้ บิตนี้จะถูกรีเซต แต่ไม่ได้เป็นการระบุว่าการทำงานของคำสั่งนั้นได้เริ่มต้นแล้ว

บิต 5 CPBF (Command Parameter Buffer Full) หากมีการเขียนพารามิเตอร์ในรีจิสเตอร์ พารามิเตอร์แล้ว บิตนี้จะถูกเซต เมื่อ 8273รับค่าพารามเตอร์ครบทุกตัวแล้ว บิตนี้จะถูกรีเซต

บิต 4 CRBF (Command Result Buffer Full) เมื่อเกิดผลลัพธ์ทันทีทันใดในรีจิสเตอร์ผลลัพธ์ของการทำงานบางคำสั่ง บิตนี้จะถูกเซต เมื่อซีพียูมาอ่านค่าผลลัพธ์แล้ว บิตนี้จะถูกรีเซต

บิต 3 RxINT (Receive Interrrupt) บิตนี้จะถูกเซตเมื่อชุดรับของ 8273 เกิดสัญญาณอินเทอร์รัพต์ที่ขา 11 เพื่อให้ซีพียูมาอ่านผลลัพธ์ของการรับข้อมูลหรืออ่านข้อมูลที่ถูกส่งมายังส่วนรับของ 8273 เมื่อซีพียูให้บริการดังกล่าวแล้ว บิตนี้จะถูกรีเซต

บิต 2 TxINT (Transmitter Interrupt) บิตนี้จะถูกเซตเมื่อชุดส่งของ 8273 เกิดสัญญาณอินเทอร์รัพต์ที่ขา 2 เพื่อให้ซีพียูมาอ่านผลลัพธ์การส่งข้อมูลหรือเขียนข้อมูลที่ต้องการส่งให้ 8273 เมื่อซีพียูให้บริการดังกล่าวแล้ว บิตนี้จะถูกรีเซต

บิต 1 RxIRA (Receiver Interrupt Result Available) หากมีผลลัพธ์ในรีจิสเตอร์แสดงผลลัพธ์การรับข้อมูล บิตนี้จะถูกเซตโดย 8273 เมื่อซีพียูอ่านผลลัพธ์นี้แล้ว บิตนี้จะถูกรีเซต

บิต 0 TxIRA (Transmitter Interrupt Result Available) หากมีผลลัพธ์ในรีจิสเตอร์แสดงผลลัพธ์การส่งข้อมูล บิตนี้จะถูกเซตโดย 8273 เมื่อซีพียูอ่านผลลัพธ์นี้แล้ว บิตนี้จะถูกรีเซต

ผังงานของเฟสคำสั่งแสดงดังรูปที่ ก.3 ระบุไว้ว่า คำสั่งจะไม่สามารถเกิดขึ้นได้ถ้ารีจิสเตอร์สถานะบอกว่า 8273 'busy' ขณะเดียวกันในการเขียนพารามิเตอร์หนึ่ง หากมีการระบุว่าบัฟเฟอร์พารามิเตอร์ 'full' การทำงานของ 8273 จะเกิดข้อผิดพลาด

<u>เฟสปฏิบัติการ</u>

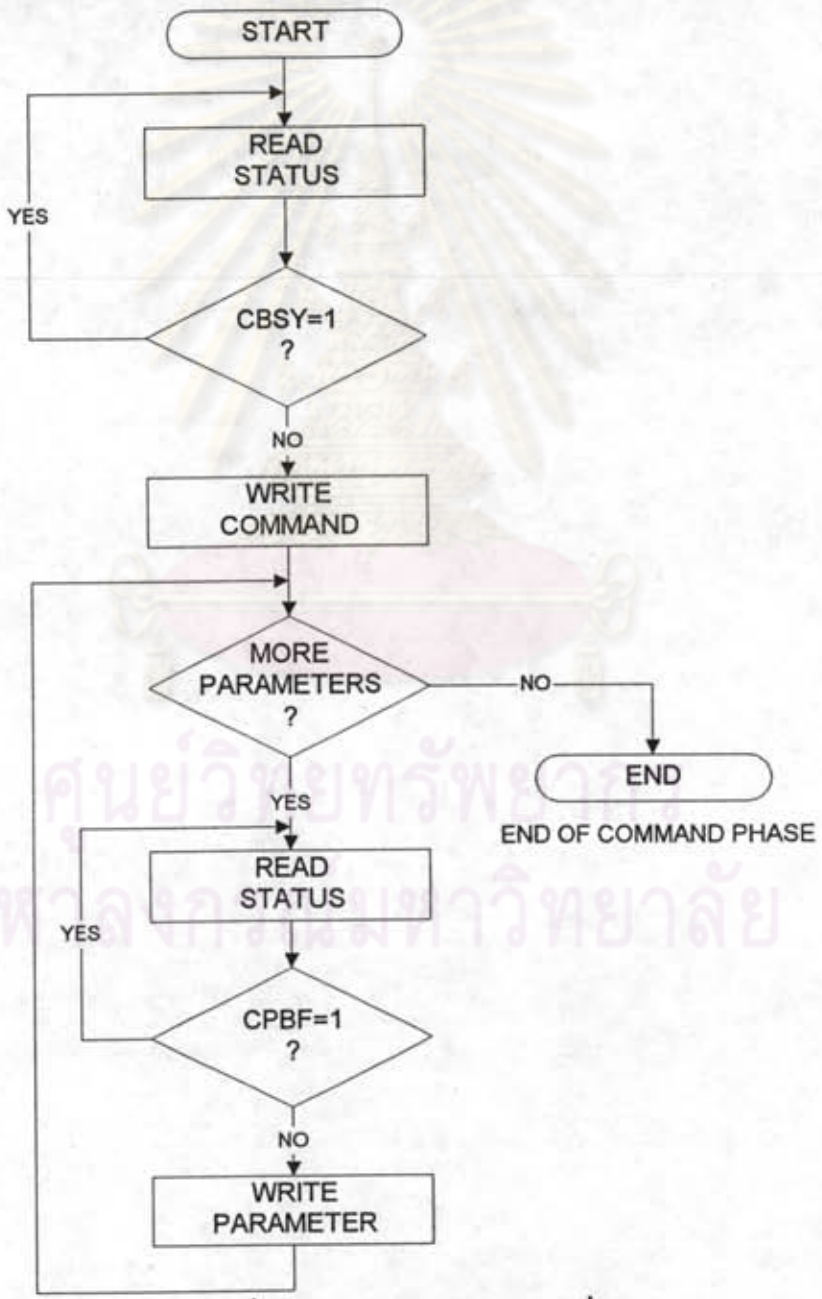
หลังจาก 8273 รับพารามิเตอร์ครบทุกตัวแล้ว จะเข้าสู่เฟสปฏิบัติการ ซีพียูไม่ต้องปฏิบัติการในส่วนนี้หากการถ่ายเทข้อมูลเป็นแบบ DMA แต่หากการถ่ายเทข้อมูลเป็นแบบอินเทอร์รัพต์แล้วซีพียูจะต้องให้บริการการถ่ายเทข้อมูลทุก ๆ ไบต์

เฟสผลลัพธ์

สาเหตุการเกิดเฟสผลลัพธ์มี 2 สาเหตุคือ

1. การปฏิบัติการเสร็จสมบูรณ์
2. ตรวจพบมีข้อผิดพลาดขณะปฏิบัติการ

ผลลัพธ์ในเฟสนี้ แบ่งได้เป็น 2 ประเภทคือ 1. ผลลัพธ์ทันทีทันใด (Immediate Result) และ 2 ผลลัพธ์ไม่เกิดขึ้นทันทีทันใด (Non-immediate Result)



รูปที่ ก.3 แสดงผังงานส่วนเฟสคำสั่ง

ผลลัพธ์ทันทีทันใด (Immediate Result) ผลลัพธ์ส่วนนี้เกิดขึ้นจากการอ่านพอร์ตA หรือการอ่านพอร์ตB ซึ่งไม่เกี่ยวข้องกับการถ่ายเทข้อมูล

ผลลัพธ์ไม่เกิดขึ้นทันทีทันใด (Non-immediate Result) เมื่อเกิดผลลัพธ์จากคำสั่งในการถ่ายเทข้อมูล 8273 จะเกิดสัญญาณอินเทอร์รัพต์ เพื่อให้ซีพียูอ่านผลลัพธ์การถ่ายเทข้อมูลนั้น ๆ สำหรับรหัสผลลัพธ์การรับข้อมูลแสดงดังตารางที่ ก.2 และรหัสผลลัพธ์การส่งข้อมูลแสดงดังตารางที่ ก.3

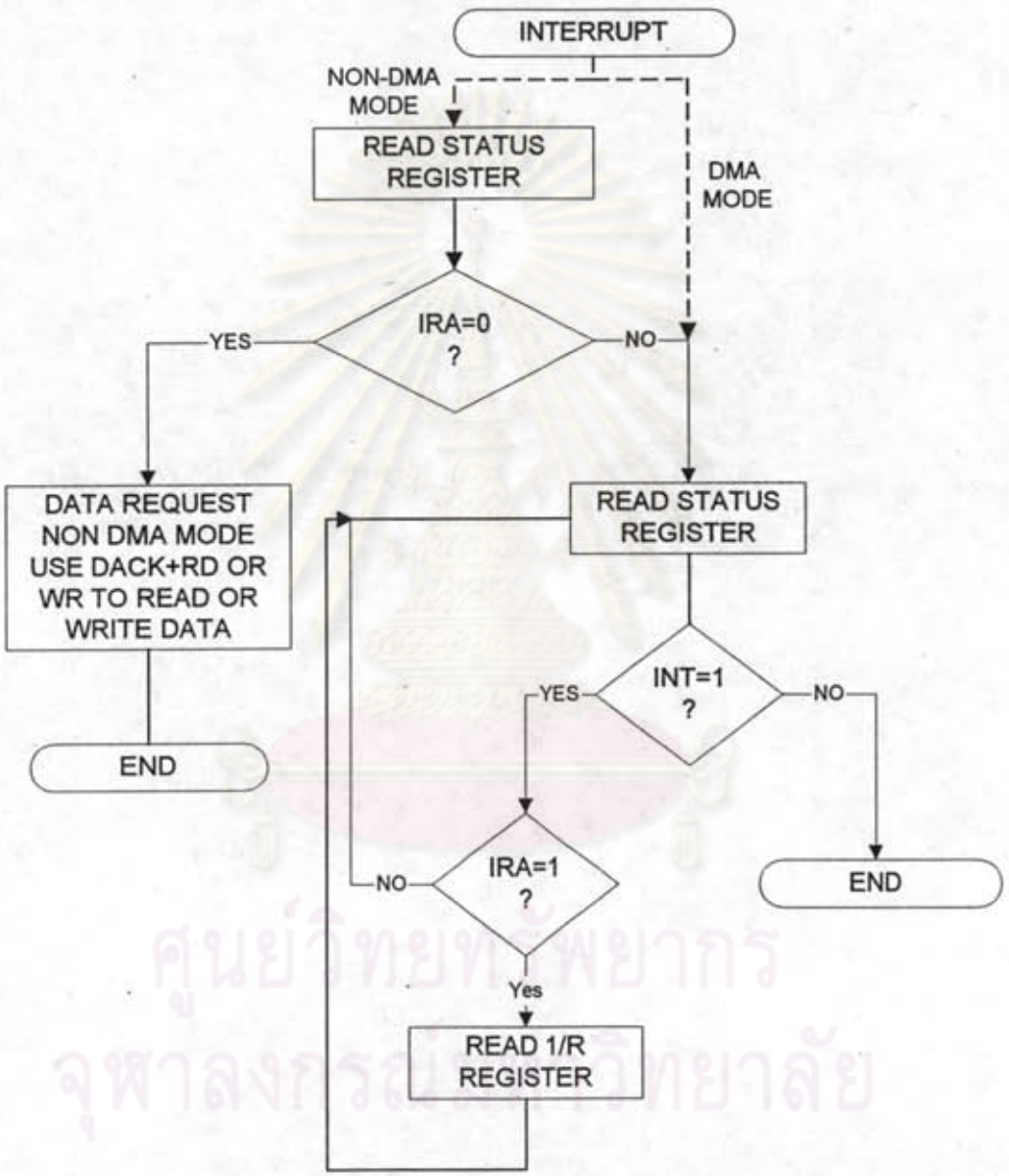
D7	D6	D5	D4	D3	D2	D1	D0	Rx Interrupt Result Code	Rx Status After INT
*	*	*	0	0	0	0	0	A1 match or general Rx	Active
*	*	*	0	0	0	0	1	A2 match	Active
0	0	0	0	0	0	1	1	CRC error	Active
0	0	0	0	0	1	0	0	Abort detected	Active
0	0	0	0	0	1	0	1	Idle detected	Disable
0	0	0	0	0	1	1	0	EOP detected	Disable
0	0	0	0	0	1	1	1	Frame less than 32 bits	Active
0	0	0	0	1	0	0	0	DMA overrun detected	Disable
0	0	0	0	1	0	0	1	Memory buffer overflow	Disable
0	0	0	0	1	0	1	0	Carrier detected failure	Disable
0	0	0	0	1	0	1	1	Receive interrupt overrun	Disable

ตารางที่ ก.2 รหัสผลลัพธ์การรับข้อมูล

D7	D6	D5	D4	D3	D2	D1	D0	Tx INT Result Code
0	0	0	0	1	1	0	1	Early transmit interrupt
0	0	0	0	1	1	0	1	Frame Tx complete
0	0	0	0	1	1	1	0	DMA underrun
0	0	0	0	1	1	1	1	Clear to Send error
0	0	0	1	0	0	0	0	Abort complete

ตารางที่ ก.3 รหัสผลลัพธ์การส่งข้อมูล

ผังงานของโปรแกรมการอ่านผลลัพธ์การส่งข้อมูลหรือการรับข้อมูล และการเขียนหรือการอ่านข้อมูลในขณะที่เกิดอินเทอร์รัพต์แต่ละครั้ง แสดงดังรูปที่ ก.4



รูปที่ ก.4 แสดงผังงานของเฟสผลลัพธ์ และการรับส่งข้อมูลที่เกิดจากสัญญาณอินเทอร์รัพต์

<u>รายละเอียดของคำสั่งใน 8273</u>

สำหรับคำสั่งที่ใช้ในการควบคุมการทำงานของ 8273 สามารถแบ่งได้เป็น 6 คำสั่งคือ

1.คำสั่งเซตหรือรีเซตการเริ่มต้นการทำงาน (Initialize set/reset Commands)

2.คำสั่งอุปกรณ์ถูกรีเซต (Reset Device Command)

3.คำสั่งรับข้อมูล (Receive Commands)

4.คำสั่งส่งข้อมูล (Transmit Commands)

5.คำสั่งยกเลิกการส่งข้อมูล (Abort Transmit Commands)

6.คำสั่งควบคุมโด็ม (Modem Control Commands)

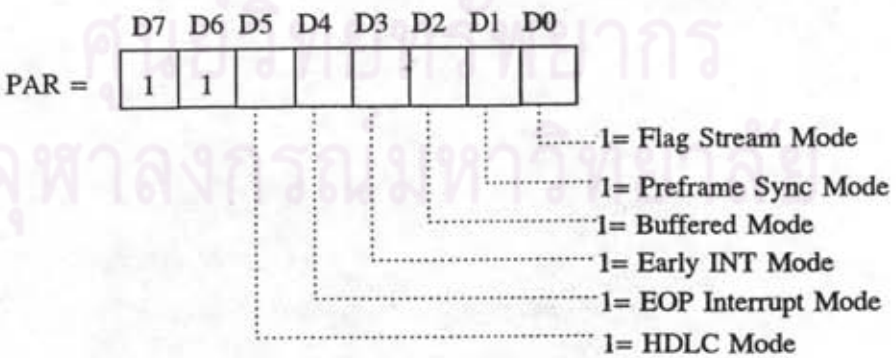
1.1 คำสั่ง 'Set One-bit Delay' CMD = A4 H, PAR = 80H เมื่อป้อนคำสั่งนี้แก่ 8273 ข้อมูลที่ส่งออกไปจะถูกส่งช้าลงเป็นเวลา 1 บิตข้อมูล ปกติจะใช้กับระบบที่ต่อกันแบบวนรอบ (Loop configurations)

1.2 คำสั่ง 'Reset One-bit Delay' CMD = 64 H, PAR = 7F H เป็นการยกเลิกการส่งแบบช้าลงเป็นเวลา 1 บิตข้อมูล

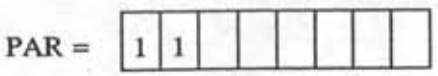
1.3 คำสั่ง 'Set Data Transfer Mode' CMD = 97 H , PAR = 01 H เป็นการกำหนดให้ 8273 มีการถ่ายเทข้อมูลโดยอาศัยการอินเทอร์รัพต์

1.4 คำสั่ง 'Reset Data Transfer Mode' CMD = 57 H, PAR = F E H เป็นการกำหนดให้ 8273 , มีการถ่ายเทข้อมูลแบบ DMA

1.5 คำสั่ง 'Set Operating Mode' CMD = 91 H ส่วนพารามิเตอร์มีรายละเอียดดังนี้

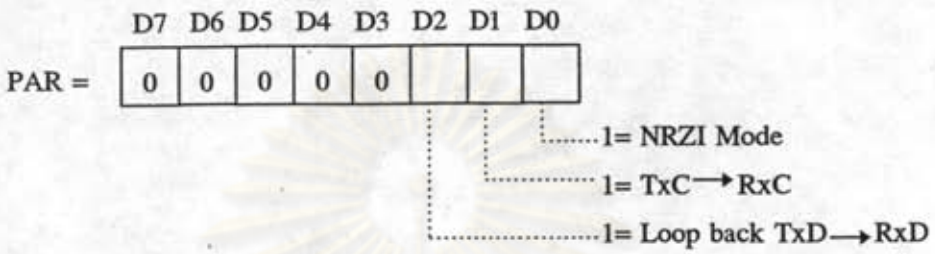


1.6 คำสั่ง 'Reset Operating Mode' CMD = 51 H ส่วนพารามิเตอร์มีรายละเอียดดังนี้



คำสั่งในโมดการปฏิบัติการนี้ จะทำงานตามการเซตหรือรีเซตของแต่ละบิต

1.7 คำสั่ง 'Set Serial I/O Mode' CMD = A0 H ส่วนพารามิเตอร์เป็นดังนี้



1.8 คำสั่ง 'Reset Serial I/O Mode' CMD = 60 H ส่วนพารามิเตอร์เป็นดังนี้

2.คำสั่งอุปกรณ์ถูกรีเซต (Reset Device Command) TMR = 01 H และ TMR = 00H

การรีเซตในส่วนนี้ทำได้โดยส่งค่า 01 H และ 00 H ให้กับรีจิสเตอร์ทดสอบ (TEST Register) คำสั่งนี้จะมีผลเหมือนกับการรีเซตที่ขา 4 ของ 8273 ซึ่งมีผลดังนี้

1. สัญญาณควบคุมโมเดม ถูกทำให้เป็น 'high' (Inactive)
2. ข้อมูลในรีจิสเตอร์สถานะถูกล้าง
3. คำสั่งต่าง ๆ จะถูกยกเลิก
4. 8273 เข้าสู่สถานะว่าง จนกว่าจะมีคำสั่งมาควบคุม
5. คำสั่งการเริ่มต้นการทำงานถูกรีเซตทั้งหมด และการรับส่งข้อมูลจะถือเป็นการรับส่งแบบ DMA
6. อุปกรณ์จะถือว่าไม่ได้เป็นการต่อแบบลูป

3.คำสั่งรับข้อมูล (Receive Commands) คำสั่งในส่วนนี้ประกอบด้วยคำสั่งรับข้อมูล 4 คำสั่งคือ

3.1 คำสั่งข้อมูลทั่วไป (General Receive) CMP = CO H, PAR1 = B0, PAR2 = B1 ข้อสังเกตุของคำสั่งในกลุ่มนี้มีดังนี้

1. ถ้ากำหนดเป็นโมดบัฟเฟอร์ การกำหนดความยาวของเฟรมข้อมูลที่รับได้ RO : R1 จะเป็นจำนวนของไบต์ข้อมูลทั้งหมด
2. ถ้ากำหนดเป็นไม่ใช่โมดบัฟเฟอร์ ความยาวของเฟรมข้อมูลที่รับได้ จะเป็นความยาวของไบต์ข้อมูลในฟิลด์ข้อสนเทศบวก 2

3. ส่วน FCS ไม่ถูกถ่ายเทเข้าสู่หน่วยความจำ

4. ถ้าในหนึ่งเฟรมข้อมูล มีข้อมูลน้อยกว่า 32 บิต จะยกเลิกเฟรมข้อมูลนั้น

5. ถ้ากำหนดเป็นไม่ใช่โมดบัฟเฟอร์ จะเกิดสัญญาณอินเทอร์รัพต์เมื่อรับข้อมูลน้อยกว่า 32 บิต ในเฟรมหนึ่ง ๆ ดังนั้นการร้องขอถ่ายเทข้อมูลจะเกิดขึ้น

6. ส่วนรับของ 8273 จะยกเลิการรับข้อมูล เมื่อได้รับอักขระว่างขณะที่รับเฟรมข้อมูล ซีพียูจะต้องมีคำสั่งรับข้อมูล จึงจะทำให้ส่วนรับทำงานได้

7. ระหว่างแฟลกสุดท้าย หากมีการรับอักขระยกเลิก (Abort) จะไม่เกิดสัญญาณอินเทอร์รัพต์

3.2 คำสั่งรับข้อมูลเฉพาะที่เลือก (Selective Receive) CMD = C1 H, PAR1 = B0, PAR2 = B1, PAR3 = A1, PAR4 = A2 เป็นโมดการรับข้อมูลเฉพาะเฟรมที่มีฟิลด์เขตที่อยู่ตรงกับฟิลด์เขตที่อยู่ที่เขียนให้พารามิเตอร์ของ 8273 (A1 หรือ A2)

3.3 คำสั่งรับข้อมูลแบบลูป (Selective Loop Receive) CMD = C2H, PAR1 = B0, PAR2 = B1, PAR3 = A1, PAR4 = A2

3.4 คำสั่งยกเลิกการรับ (Receive Disable) CMD = C5 H เป็นคำสั่งที่ยกเลิกการรับข้อมูลอย่างทันที

4.คำสั่งส่งข้อมูล (Transmit Commands) คำสั่งส่วนนี้ประกอบด้วยคำสั่ง 3คำสั่ง คือ

4.1 คำสั่งส่งเฟรมข้อมูล (Transmit Frame) CMD = C8 H, , PAR1 = L0, PAR2 = L1, PAR3 = A, PAR4 = C

ถ้าการส่งข้อมูลเป็นแบบโมดบัฟเฟอร์ การกำหนดค่า L0 , L1 ซึ่งเป็นค่าความยาวของฟิลด์ข้อสนเทศและฟิลด์เขตที่อยู่จะต้องถูกเขียนไปยังรีจิสเตอร์พารามิเตอร์

ถ้าการส่งข้อมูลไม่เป็นแบบโมดบัฟเฟอร์ ค่าความยาวของเฟรมที่ส่งจะต้องเป็นค่าความยาวของฟิลด์ข้อสนเทศบวกกับฟิลด์เขตที่อยู่และฟิลด์ควบคุม ซึ่งจะต้องถูกเขียนไปยังรีจิสเตอร์ พารามิเตอร์ ส่วน L0: L1

4.2 คำสั่งเฟรมข้อมูลแบบลูป (Loop Transmit) CMD = C8 H, , PAR1 = L0, PAR2 = L1, PAR3 = A, PAR4 = C

4.3 คำสั่งส่งผ่านข้อมูลโดยตรง (Transmit transparent) CMD = C9 H, , PAR1 = L0, PAR2 = L1 8273 จะส่งข้อมูลโดยไม่คำนึงถึงโปรโตคอล, ไม่มีการแทรกบิต 0, ไม่มีแฟลก, ไม่มีส่วนตรวจสอบลำดับเฟรม

5.คำสั่งยกเลิกการส่งข้อมูล (Abort Transmit Commands) คำสั่งในส่วนนี้ประกอบด้วยคำสั่ง 3 คำสั่งดังนี้

5.1 คำสั่งยกเลิกการส่งเฟรมข้อมูล (Abort Transmit Frame) CMD = CCH หลังจากมีการส่งอักขระยกเลิก (บิตที่เป็น '1' ติดกันจำนวน 8 บิต) ชุดส่งจะเซตเป็นการส่งแฟลกหรืออักขระว่างเมื่อระบุเป็นโมดกระแสแฟลก

5.2 คำสั่งยกเลิกการส่งเฟรมข้อมูลแบบลูป (Abort Loop Transmit) CMD = CEH หลังจากแฟลกถูกส่งชุดส่งจะเซตเป็นโมด 'One Bit Delay'

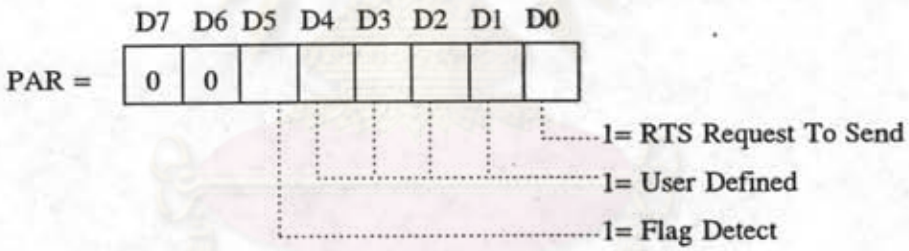
5.3 คำสั่งยกเลิกการส่งเฟรมข้อมูลโดยตรง (Abort Transmit Transparent) CMD = CDH ชุดส่งข้อมูลจะเซตเป็นการส่งแฟลกหรืออักขระว่างเมื่อระบุเป็นโมดกระแสแฟลก

6.คำสั่งควบคุมโมเดม (Modem Control Commands) คำสั่งส่วนนี้ใช้เป็นชุดควบคุมพอร์ต ควบคุมโมเดม เมื่อมีคำสั่งอ่านพอร์ต A หรือ พอร์ต B จะปฏิบัติการ แล้วให้ผลลัพธ์ในรีจิสเตอร์ผลลัพธ์ คำสั่งในส่วนนี้ประกอบด้วย 4 คำสั่งดังนี้

6.1 คำสั่งอ่านพอร์ต A (Read Port A) CMD = 22H

6.2 คำสั่งอ่านพอร์ต B (Read Port B) CMD = 23H

6.3 คำสั่งเซตพอร์ต B (Set Port B Bits) CMD = A3H รูปแบบของพารามิเตอร์เป็นดังนี้



6.4 คำสั่งรีเซตพอร์ต B (Reset Port B Bits) CMD = 63 H รูปแบบของพารามิเตอร์เป็นดังนี้รูปแบบเหมือนกับคำสั่งเซตบิตในพอร์ต B

ตารางสรุปคำสั่งของ 8273 แสดงดังตารางที่ ก.4

Command Description	Command (HEX)	Parameter	Results	Result Port	Completion Interrupt
Set One Bit Delay	A4	Set Mask	None	-	No
Reset One Bit Delay	64	Reset Mask	None	-	No
Set Data Transfer	97	Set Mask	None	-	No
Reset Data Transfer	57	Reset Mask	None	-	No

ตารางที่ ก.4 สรุปคำสั่งใน 8273

Reset Data Transfer	57	Reset Mask	None	-	No
Set Operating Mode	91	Set Mask	None	-	No
Reset Operating Mode	51	Reset Mask	None	-	No
Set Serial I/O Mode	A0	Set Mask	None	-	No
Reset Serial I/O Mode	60	Reset Mask	None	-	No
General Receive	C0	B0,B1	RIC,R0,R1, (A,C)(2)	RXI/R	Yes
Selective Receive	C1	B0,B1,A1,A2	RIC,R0,R1, (A,C)(2)	RXI/R	Yes
Selective Loop Receive	C2	B0,B1,A1,A2	RIC,R0,R1, (A,C)(2)	-	Yes
Receive Disable	C5	None	None	TXI/R	No
Transmit Frame	C8	L0,L1,(A,C)(1)	TIC	TXI/R	Yes
Loop Transmit	CA	L0,L1,(A,C)(1)	TIC	TXI/R	Yes
Transmit Transparent	C9	L0,L1	TIC	TXI/R	Yes
Abort Transmit Frame	CC	None	TIC	TXI/R	Yes
Abort Loop Transmit	CE	None	TIC	TXI/R	Yes
Abort Transmit Trans.	CD	None	TIC	TXI/R	Yes
Read Port A	22	None	Port Value	Result	No
Read Port B	23	None	Port Value	Result	No
Set Port B bit	A3	Set Mask	None	-	No
Reset Port B Bit	63	Reset mask	None	-	No

ตารางที่ ก.4 สรุปคำสั่งใน 8273(ต่อ)

8273 Command Summary Key

B0 : Least significant byte of the receive buffer length

B1 : Most significant byte of the receive buffer length

L0 : Least significant byte of the Tx frame length

L1 : Most significant byte of the Tx frame length

A1 : Receive frame address match field one.

A2 : Receive frame address match field two

A : Address field of receive frame.If non-buffered mode is specified, this result is not provided

C : Control field of receive frame.If non-buffered mode is specified, this result is not provided

RXI/R : Receive interrupt result register

TXI/R : Transmit interrupt result register

R0 : Least significant byte of the length of the frame received

R1 : Most significant byte of the length of the frame received

RIC : Receive interrupt result code

TIC : Transmit interrupt result code

ภาคผนวก ข

คู่มือการใช้งานของโมดูลระยะไกล

โมดูลระยะไกลในงานวิจัยนี้มี 5 ชนิด ดังได้กล่าวไว้ในหัวข้อย่อยบทที่ 6. ซึ่งโมดูลแต่ละชนิดเหมาะสมกับสัญญาณอินพุตหรือเอาต์พุตที่แตกต่างกัน ในการใช้งานให้เหมาะสมกับโมดูลแต่ละชนิดได้โดยพิจารณาตามหัวข้อต่อไปนี้

1.คุณลักษณะของโมดูลระยะไกล

1.1 ลักษณะของแผงด้านหน้าของโมดูล

*ภาคแสดงความพร้อมของโมดูล

การแสดงผล : LED สีเขียว บอกสถานะว่าโมดูลพร้อมที่จะทำงานหรือไม่

*ภาคแสดงสถานะของอินพุตหรือเอาต์พุต

การแสดงผล : LED สีแดง 16 ชุด บอกสถานะ 'ON' หรือ ' OFF' ของแต่ละช่องสัญญาณของโมดูลดิจิตอลอินพุต หรือโมดูลดิจิตอลเอาต์พุต

*ภาคแสดงสถานะของวงจรเตือน

การแสดงผล : LED สีแดง 2 ชุด ชุดหนึ่งใช้สำหรับเตือนเมื่อขนาดของสัญญาณอินพุตมีค่ามากกว่าขนาดระดับสูงที่กำหนด ส่วนอีกชุดหนึ่ง ใช้สำหรับเตือนเมื่อขนาดของสัญญาณอินพุตมีค่าต่ำกว่าขนาดระดับต่ำกว่าที่กำหนด

1.2 การกำหนดฟิลด์เขตที่อยู่

สวิตช์ที่ 1 ถึง 5 ของดิพสวิตช์ใช้กำหนดหมายเลขฟิลด์เขตที่อยู่ของโมดูลระยะไกล

สวิตช์ที่ 6,7 และ 8 ของดิพสวิตช์ใช้กำหนดชนิดของโมดูลระยะไกล

1.3 โมดูลแบบอินพุต

ชนิดของอินพุต : 1.ดิจิตอล จำนวน 16 ช่องสัญญาณ

2.แอนะลอก 4-20 มิลลิแอมแปร์ จำนวน 3 ช่องสัญญาณ, และ 1-5 โวลต์ จำนวน 3 ช่องสัญญาณ

ชนิดของเอาต์พุต : 1.ดิจิตอล จำนวน 16 ช่องสัญญาณ

2.แอนะลอก 4-20 มิลลิแอมแปร์ จำนวน 3 ช่องสัญญาณ และ 1-5

โวลต์ จำนวน 3 ช่องสัญญาณ

3.แอนะลอก 0-40 มิลลิแอมแปร์ จำนวน 3 ช่องสัญญาณ และ 0-10 โวลต์ จำนวน 3 ช่องสัญญาณ

1.4 แหล่งจ่ายไฟขาเข้า : 5 Vdc และ +/-12 Vdc

2. ลักษณะด้านหลังของโมดูลระยะไกล (Rear Panel)

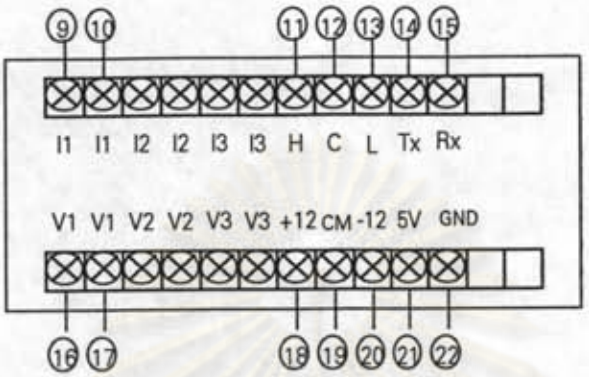
ลักษณะด้านหลังของโมดูลระยะไกล สามารถแบ่งเป็น 2 กลุ่มใหญ่ ๆ คือ กลุ่มดิจิตอลและกลุ่มแอนะลอก

2.1 กลุ่มดิจิตอล

รูปที่ ข.1 แสดงลักษณะด้านหลัง

① ช่องสัญญาณ 1 ขั้ว +
② ช่องสัญญาณ 1 ขั้ว -
③ สัญญาณ Data +
④ สัญญาณ Data -
⑤ แหล่งจ่ายไฟ ขั้ว + 5 Volts
⑥ แหล่งจ่ายไฟ ขั้ว Ground
⑦ แหล่งจ่ายไฟ ขั้ว +24 Volts
⑧ แหล่งจ่ายไฟ ขั้ว Common

2.1 กลุ่มแอนะลอก

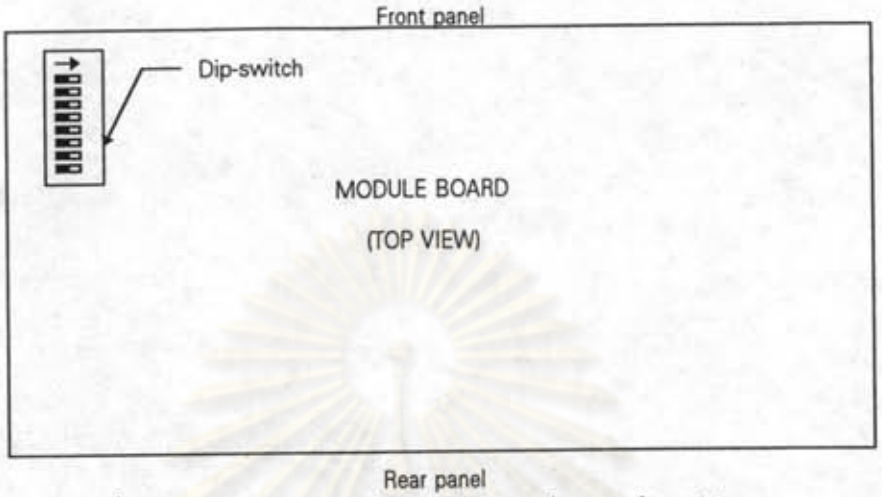


รูปที่ ข.2 แสดงลักษณะด้านหลักของโมดูลกลุ่มแอนะลอก

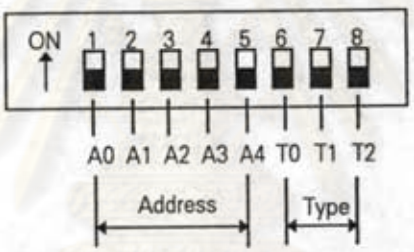
⑨ สัญญาณกระแส ช่อง 1 ขั้ว +
⑩ สัญญาณกระแส ช่อง 1 ขั้ว -
⑪ หน้าสัมผัสการเตือน High Alarm
⑫ หน้าสัมผัสการเตือนร่วม
⑬ หน้าสัมผัสการเตือน Low Alarm
⑭ สัญญาณ Data +
⑮ สัญญาณ Data -
⑯ สัญญาณแรงดัน ช่อง 1 ขั้ว +
⑰ สัญญาณแรงดัน ช่อง 1 ขั้ว -
⑱ แหล่งจ่ายไฟ ขั้ว +12 Volts
⑲ แหล่งจ่ายไฟ ขั้ว Common
⑳ แหล่งจ่ายไฟ ขั้ว -12 Volts
㉑ แหล่งจ่ายไฟ ขั้ว +5 Volts
㉒ แหล่งจ่ายไฟ ขั้ว Ground

3. ดิพสวิตช์เพื่อกำหนดชนิดของโมดูลระยะไกลและฟิลด์เบตที่อยู่

ดิพสวิตช์ใช้ในการกำหนดชนิดของโมดูลระยะไกลและฟิลด์เบตที่อยู่ ตำแหน่งของดิพสวิตช์จะอยู่บนบอร์ด ดังแสดงในรูปที่ ข.3 และรูปที่ ข.4 ชนิดของโมดูลและฟิลด์เบตที่อยู่ของดิพสวิตช์ดังกล่าว



รูปที่ 10.3 แสดงลักษณะตำแหน่งดิพสวิตซ์บนบอร์ดของโมดูล



รูปที่ ข.4 แสดงตำแหน่งชุดสวิตซ์ที่เลือกฟีลด์เขตที่อยู่และชนิดของโมดูล

A0-A5 ชุดสวิตซ์เลือกฟีลด์เขตที่อยู่

T0-T2 ชุดสวิตซ์เลือกชนิดของโมดูลระยะไกล

เพื่อกำหนดชนิดของโมดูลให้สัมพันธ์กันระหว่างฮาร์ดแวร์กับซอฟต์แวร์ แสดงได้ดังตารางที่ ข.1

ชุดสวิตซ์เลือกชนิดโมดูลระยะไกล			ชนิดของโมดูล
T0	T1	T2	
OFF	ON	OFF	Digital input
ON	ON	OFF	Digital output
ON	OFF	OFF	Analog input (4-20 mA,1-5 V)
OFF	OFF	ON	Analog output (4-20 mA,1-5 V)
ON	OFF	ON	Analog output (0-40 mA,0-10 V)

ตารางที่ ข.1การกำหนดชนิดของโมดูลให้สัมพันธ์กับโมดูลระยะไกล

ภาคผนวก ค

โปรแกรมของโมดูลระยะไกล

```
#pragma CODE SYMBOLS DEBUG  /* Franklin c51 compiler option */

#include <STDIO.H>
#include <ABSACC.H>
#include <MATH.H>
#include "83C51FA.H"
#include "GLOBAL_R.H"
#include "MODE_RT.H"
#include "PORT_RT.H"
#include "SYSTEM.H"

bit                 test_ad;
unsigned char       *buffer_in,*buffer_out;
unsigned char       *buffer_start,*buffer_end;
unsigned char       flag_tx,flag_rx,check_error_flag,receive_flag;
unsigned char       transmit_flag,long_tx,long_rx,mode;
unsigned char       result_rx[5],result_tx,par_num,wd_flag,wd_count;
unsigned char       cmd,longt,control,tst,err,bumper,tmp,shift;
idata unsigned char par[4],long_result,sampl,set_sampl,input_flag[7];
unsigned char       i,j,x,z;
idata unsigned char data_tx[6],data_rx[10],data_rx_test[10];
idata unsigned char data_tx_test[10] = {'\x1f','\x2f','\x55','\x00','\x5f',
                    '\x0f','\x3f','\xf7','\x79','\x11'};
```

```
transfer(void) interrupt 0 using 0{
data unsigned char j;
 if (IE0){
  if (RxIBIT BUILD){
   if (RxRBIT == NOK){                    /* receive data */
    if (tst ON)  data_rx_test[long_rx] = RXDACK;
    else data_rx[long_rx] = RXDACK;
    long_rx++;
   }
   else{
    while (RxIBIT BUILD){                 /* result rx data */
     if (RxRBIT BUILD){
    result_rx[long_result] = RxINT;
    long_result++;
      }
    }
    long_result = 0;long_rx = 0;
    flag_rx SET;
   }
  }
  else{
   if (TxRBIT BUILD){                     /* result tx data */
    result_tx = TxINT;
    long_tx = 0;
    flag_tx SET;
   }
   else{                                  /* transmit data */
    if (tst ON)  TXDACK = data_tx_test[long_tx];
    else TXDACK = data_tx[long_tx];
    long_tx++;
   }
  }
 }
}
```

```
sampling(void) interrupt 6 using 1{
idata unsigned int  val_new[7],val_old[7],valh;
data unsigned char chnl,chnh,chnl_old,chnh_old,chan;
data unsigned      ad_ref_min,ad_ref_max;

 if(CCF0) {
   EC RESET;
   CCF0 RESET;
   if ((CCAPM4&0x7f)!=0x48)
     CCAPM4 = 0x48;
   wd_flag = WD_RST;
   CCAP4L = CL+LOW_WMATCH;
   CCAP4H = CH+(unsigned char)CY + HIG_WMATCH;
   wd_flag = WD_SET;
   CCAP0L += LOW_MATCH;
   CCAP0H += (unsigned char)CY + HIG_MATCH;
   sampl RESET ;
   switch(IO_TYPE){
      case AOUT1 :
      case AOUT2 : z++;
                   if(z<7){
                    if(input_flag[z] ON){
                      chan = (0x80|(z<<4));
                      val_old[z] |= chan;
                      P2 = (val_old[z]&0xff);
                      P0 = (val_new[z]&0xff);
                      for(i=0;i<2;i++);
                      OUTEN = 0;
                      for(i=0;i<15;i++);
                      OUTEN = 1;
                    }
                   }
                   else z = 0;break;
      case AIN :z++;
                if(z<7){
                 if(input_flag[z] ON){
                   chan = (0x70&(z<<4));
                   val_new[z] = (int)adc12(chan);
                   if((0<=val_new[z])&&(val_new[z]<=0xfff))
                     test_ad = OK;
                   else
                     test_ad = NOK;
                   if((abs)(val_new[z]-val_old[z]) > val_x[z]){
```

```
                    val_old[z] = val_new[z];
                    valh = val_new[z];
                    valh >>= 8;
                  }
                }
              }
              else z = 0; break;
        case DIN :chnl = CNTL;chnh = CNTL;  /* test for work */
              if ((chnl != chnl_old)||(chnh != chnh_old)){
                chnl_old = chnl;
                chnh_old = chnh;
                alive RESET;
              }
              break;
    }
  }
 EC SET;
 }
}
```

```
main () {
data unsigned char  i,j,mode; /* general counter */

 for(i=0; i<255; i++)          /* power on delay */
  for(j=0; j<255; j++);
 if(wd_flag==WD_SET) {         /* watchdog checking */
  if(++wd_count==CNT_OUT) {
    HWRDY = LED_OFF;
    while(1);
  }
 }
 else{
  wd_flag  = wd_count = NOK;
  HWRDY = LED_ON;
 }
 hdlc_check();              /* 8273 loopback check */
 variable_default();
 i = ADDRESS;
 if((0x0<ADDRESS)&(ADDRESS<= 0x1e))       /* Address checking */
  HWRDY = LED_ON;
 else{
  HWRDY = LED_OFF;
  while(1);
 }
 init_timebase();
 init_system_timer();
 transmit_flag RESET;
 long_tx = long_rx = 0;
 command_receive();
 while(1){
 if(flag_rx ON){
  check_receive();
  if (transmit_flag ON){
  if (data_rx[1] == 0xff){       /* test mode checking */
    longt = 2;
    control = 0xff;
    command_transmit(longt);
    while(flag_tx != '1');
    transmit_flag RESET;
    command_receive();
   }
   else{
    mode = data_rx[0];
```

```
    switch(mode){
      case REQUEST : request();break;
      case CONTROL : contrl();break;
      default      : control=ONE_F,mode=MOD_ERR;
    }
  }
  if (transmit_flag ON){
  data_tx[0] = ADDRESS;
  data_tx[1] = control;
  data_tx[2] = mode;
  command_transmit(longt);
  while(flag_tx != '1');
  transmit_flag RESET;
  if(test_ad){
    test_ad = NOK;
    ad_ref_min = adc12(AD1);
    ad_ref_max = adc12(AD5);
    if(!test_ad){
      HWRDY = LED_ON;
      if((ad_ref_min >= 0x31b)&&(ad_ref_min <= 0x33b)){
        if(ad_ref_max < 0xff0){
          ad_ref_max = 0;
          HWRDY = LED_OFF;
        }
      }
      else{
        ad_ref_min = 0;
        HWRDY = LED_OFF;
      }
    }
  }
  command_receive();
  }
  }
 }
 }
}
```

```
void variable_default(){
data unsigned char i;
  set_sampl = DEF_SMP;
  set_alive = DEF_ALV;
  for(i=1;i<7;i++){
    input_flag[i] SET;
    val_old[i] = 0x0;
    val_new[i] = 0x0;
    val_x[i] = DEF_RT;
  }
}
void init_timebase(){
  TMOD = 2;
  TH0  = 6;
  ET0  = 0;
  TR0  = OK;
}
void init_system_timer(){
  CH = 0;
  CL = 1;
  CCAPM0 = 0x49;
  CCAP0L = 0x90;
  CCAP0H = 0x01;
  CCAPM1 = 0x49;
  CCAP1L = 0xe8;
  CCAP1H = 0x03;
  CCAPM4 = 0x08;
  CCAP4L = 0xff;
  CCAP4H = 0xff;
  CMOD   = 0x44;
  EX0    = OK;
  EC     = OK;  /* Enable PCA interrupt */
  EA     = OK;  /* Enable All control bit */
  CR     = OK;  /* Count Run bit turn on timer */
}

/* -------- 12 bits a/d successive approximation function -------- */
unsigned adc12(unsigned char channel) {
data unsigned        result;
data unsigned char   i;
                  /* mask bits, enable mux and clear MSB nibble */
  result = 0;
  P2  |= 0xf0;       /* clear mux to no value first */
```

```
for(i=0; i<2; i++); /* delay */
P2  = 0;
P2 |= channel;      /* set channel */
P0  = 0x0;          /* reset P0 and delay mux */
P2B7 = OK;           /* turn on mux */
                  /* successive approximation begin here */
for(i=0; i<2; i++);
P2B3 = 1;
for(i=0; i<2; i++);
if(COMP)
  P2B3 = 0;
P2B2 = 1;
for(i=0; i<2; i++);
if(COMP)
  P2B2 = 0;
P2B1 = 1;
for(i=0; i<2; i++);
if(COMP)
  P2B1 = 0;
P2B0 = 1;
for(i=0; i<2; i++);
if(COMP)
  P2B0 = 0;
P0B7 = 1;
for(i=0; i<2; i++);
if(COMP)
  P0B7 = 0;
P0B6 = 1;
for(i=0; i<2; i++);
if(COMP)
  P0B6 = 0;
P0B5 = 1;
for(i=0; i<2; i++);
if(COMP)
  P0B5 = 0;
P0B4 = 1;
for(i=0; i<2; i++);
if(COMP)
  P0B4 = 0;
P0B3 = 1;
for(i=0; i<2; i++);
if(COMP)
  P0B3 = 0;
```

```
  P0B2 = 1;
  for(i=0; i<2; i++);
  if(COMP)
    P0B2 = 0;
  P0B1 = 1;
  for(i=0; i<2; i++);
  if(COMP)
    P0B1 = 0;
  P0B0 = 1;
  for(i=0; i<2; i++);
  if(COMP)
    P0B0 = 0;
  result |= (P2 & 0xf);
  result <<= 8;
  result |= P0;
  P2 |= P0 |= 255;
  return result;
}

void command_receive(){
  cmd = 0xc1;              /* selective receive */
  par[1] = 0x0a;
  par[2] = 0x00;
  par[3] = ADDRESS;
  par[4] = GLOBAL;
  par_num = 4;
  ENA_RX = 0x01;
  command(par_num);
}
void check_receive(){
  flag_rx RESET;
  if ((0x0f&result_rx[0]) > 0x01)
    error_receive();
  else{
    if ((0xf0&result_rx[0]) < 0xe0)
    error_receive();
    else transmit_flag SET;       /* receive complete */
  }
}
void command_transmit(unsigned char long_frm){
  par_num = 4;
  cmd = 0xc8;
  par[1] = long_frm;
```

```
  par[2] = 0x0;
  par[3] = ADDRESS;
  par[4] = control;
  ENA_TX = 0x03;
  command(par_num);
}
void error_receive(){
  longt = 2;
  control = 0xa9;          /* RNR control code */
  command_transmit(longt);
  while(flag_tx != '1');
  transmit_flag RESET;
  command_receive();
  while(flag_rx != '1');
}

void request(){
   switch(IO_TYPE){
     case AIN :
                data_tx[3] = channel;
                data_tx[4] = valh;
                data_tx[5] = val_new[z]&0xff;
                longt = 6;
                break;
     case DIN : n = data_rx[3];
                bt = 1;
                bt <<= n;
                if ((old_cmd &bt) == 0)
                    data_tx[4] = '0';
                else
                    data_tx[4] = '1';
                longt = 5;
                break;
   }
  }
  control = ONE_F;       /* control field = 00001001 */
}
```

```
void contrl(){
data unsigned char stt,datah,channel,n;
data unsigned char old_cmd,bt;
   channel = data_rx[3];
   datah = data_rx[4];
   datal = data_rx[5];
   if(IO_TYPE >= 0x60){
    switch(IO_TYPE){
      case AOUT1 :
      case AOUT2 : if ((0x00 < channel)&&(channel < 0x70)){
                     P2 = (data_rx[3]|data_rx[4]);
                     P0 = data_rx[5];
                     for(i=0;i<2;i++);
                     OUTEN = 0;
                     for(i=0;i<15;i++);
                     OUTEN = 1;
                     n = channel; n >>= 4;
                     val_old[n] = data_rx[4];
                     val_new[n] = data_rx[5];
                     longt = 6;
                     data_tx[3] = channel;
                     data_tx[4] = data_rx[4];
                     data_tx[5] = data_rx[5];
                   }break;
    case DOUT  : n = channel;
                 bt = 1;
                 bt <<= n;
                 if(datah == '1')
                   old_cmd != bt;
                 if(datah == '0'){
                   bt = ~bt;
                   old_cmd &= bt;
                 }
                 if(channel<0x08) CNTL = (old_cmd&0xff);
                 else  CNTH = (old_cmd >> 8);
                 longt = 5;
                 data_tx[3] = channel;
                 data_tx[4] = datah;
                 break;
   }
  }
  control = ONE_F;       /* control field = 00001001 */
}
```

```
void hdlc_check(){
  data unsigned char i;
  tst SET;
  init_timer_mode3();
  IT0 = NOK;              /* Interrupt 0 Type low level */
  EX0 = OK;               /* Enable External interrupt0 */
  EA  = OK;               /* Enable All control bit */
  ENA_INT = 0x01;
  init_8273_test();
  long_tx = long_rx = 0;
  cmd = 0xc1; par[1] = 0x0a; par[2] = 0x00; /* selective receive */
  par[3] = ADDRESS;par[4] = GLOBAL;
  command(4);
  cmd = 0xc8; par[1] = 0x0a; par[2] = 0x00; /* transmit command */
  command(2);
  while(flag_tx != '1');
  while(flag_rx != '1');
  data_checking(10,&data_rx_test[0],&data_tx_test[0]);
  if (check_error_flag ON){
    HWRDY = LED_OFF;
    while(1);
  }
  else HWRDY = LED_ON;
  result_tx_checking();
  result_rx_checking(&result_rx[0]);
  tst RESET;
  flag_rx = flag_tx RESET;
  operate_8273();
}

void command(unsigned int p_num){
  data unsigned char i,j,status;
  status = CMD_STT;       /*  read status 8273 */
  while ((status &= 0x80) != 0)     /* check CBSY bit */
    status = CMD_STT;
  CMD_STT = cmd;              /* write command to 8273 */
  for (i=1; i<=p_num; i++){
    status = CMD_STT;       /* read status 8273 */
    while ((status &= 0x20) != 0)   /* check CPBF bit */
      status = CMD_STT;
    PAR_RST = par[i];       /* write parameter to 8273 */
  }
```

```
}

void init_8273_test(){
  data unsigned char i;
  par_num = 1;
  TxINT = 0x01;              /* reset 8273 status reg. */
  for (i=0;i<15;i++);
  TxINT = 0x00;
  for (i=0;i<15;i++);
  cmd = 0x64;
  par[1] = 0x7f;
  command(par_num);
  cmd = 0x97;                /* interrupt transfer */
  par[1] = 0x01;
  command(par_num);
  cmd = 0x91;                /* hdlc */
  par[1] = 0x20;
  command(par_num);
  cmd = 0xa0;                /* loopback tx->rx & NRZI*/
  par[1] = 0x07;
  command(par_num);
}

void operate_8273(){
  TxINT = 0x01;              /* reset 8273 status reg. */
  for (i=0;i<15;i++);
  TxINT = 0x00;
  for (i=0;i<15;i++);
  cmd = 0x64;              /* reset one bit delay 64 7f*/
  par[1] = 0x7f;
  command(par_num);
  cmd = 0x97;                /* interrupt transfer */
  par[1] = 0x01;
  command(par_num);
  cmd = 0x91;                /* hdlc & flag mode */
  par[1] = 0x25;
  command(par_num);
  cmd = 0xa0;                /* NRZI code */
  par[1] = 0x01;
  command(par_num);
}

void init_timer_mode3(){          /* baud rate setting */
```

```
 TIM_MODE = 0xb6;          /* mode3 setting */
 TIMER2 = 0x60;           /* baud = 1 kbps 60  73 */
 TIMER2 = 0x00;
 TIM_MODE = 0x76;          /* mode3 setting */
 TIMER1 = 0x21;
 TIMER1 = 0x00;
}
void data_checking(int byte_num,char *rx_data,char *tx_data){
 unsigned int i;
 check_error_flag RESET;
 for (i=1;i<=byte_num;i++){
  if (*rx_data != *tx_data){
   check_error_flag SET;
   break;
  }
  else {
   rx_data++;
   tx_data++;
  }
 }
}
void result_tx_checking(){
 if (flag_tx ON){
  if (result_tx == 0x0d)          /* code = 0x0d complete */
    HWRDY = LED_ON;
  else {
   HWRDY = LED_OFF;
   while(1);
  }
 }
 else{
  HWRDY = LED_OFF;
 }
}

void result_rx_checking(unsigned char *result){
 if (flag_rx ON){
  if ((*result&0x0f) > 1){          /* check for A0 or A1 */
   HWRDY = LED_OFF;
   while(1);
  }
  else {
   if ((*result&0xf0) == 0xe0)    /* check data last byte */
```

```
            HWRDY = LED_ON;
          else HWRDY = LED_OFF;
        }
      }
      else{
       HWRDY = LED_OFF;
      }
    }
```

83C51FA.H

```
    /*  Device list
        ROMless
          1. 80C51FA
        EPROM version
          1. 87C51FA -- 8k
          2. 87C51FB -- 16k
          3. 87C51FC -- 32k
    */
    /*  BYTE Registers  */
    sfr P0    = 0x80; /* port 0 */
    sfr SP    = 0x81; /* internal Stack Pointer */
    sfr DPL   = 0x82; /* Data Pointer low Byte  */
    sfr DPH   = 0x83; /* Data Pointer high Byte */
    sfr PCON  = 0x87; /* Power CONtrol register */
    sfr TCON  = 0x88; /* Timer/counter CONtrol register */
    sfr TMOD  = 0x89; /* Timer/counter MODe control register */
    sfr TL0   = 0x8A; /* Timer 0 Low byte */
    sfr TL1   = 0x8B; /* Timer 1 Low byte */
    sfr TH0   = 0x8C; /* Timer 0 High byte */
    sfr TH1   = 0x8D; /* Timer 1 High byte */
    sfr P1    = 0x90; /* port 1 */
    sfr SCON  = 0x98; /* Serial port CONtrol register */
    sfr SBUF  = 0x99; /* Serial BUFfer */
    sfr P2    = 0xA0; /* port 2 */
    sfr IE    = 0xA8; /* Interrupt Enable register */
    sfr SADDR = 0xA9; /* for multiprocessor communication */
    sfr P3    = 0xB0; /* port 3 */
    sfr IP    = 0xB8; /* Interrupt Priority register */
    sfr SADEN = 0xB9; /* for multiprocessor communication */
    sfr T2CON = 0xC8; /* Timer/counter 2 CONtrol register */
    sfr T2MOD = 0xC9; /* Timer 2 Mode Control Register */
    sfr RCAP2L= 0xCA; /* Timer/counter 2 CAPture Register low byte */
```

```
sfr RCAP2H= 0xCB; /* Timer/counter 2 CAPture Register high byte */
sfr TL2   = 0xCC; /* Timer/counter 2 Low byte */
sfr TH2   = 0xCD; /* Timer/counter 2 High byte */
sfr PSW   = 0xD0; /* Program Status Word */
sfr CCON  = 0xD8; /* PCA Counter Control Register */
sfr CMOD  = 0xD9; /* PCA Counter Mode Register */
sfr CCAPM0= 0xDA; /* PCA Module 0 Compare/CAPture Register */
sfr CCAPM1= 0xDB; /* PCA Module 1 Compare/CAPture Register */
sfr CCAPM2= 0xDC; /* PCA Module 2 Compare/CAPture Register */
sfr CCAPM3= 0xDD; /* PCA Module 3 Compare/CAPture Register */
sfr CCAPM4= 0xDE; /* PCA Module 4 Compare/CAPture Register */
sfr ACC   = 0xE0; /* ACCumulator */
sfr CL    = 0xE9; /* PCA Count value Low byte */
sfr CCAP0L= 0xEA; /* PCA Compare/CAPture value module 0 Low byte */
sfr CCAP1L= 0xEB; /* PCA Compare/CAPture value module 1 Low byte */
sfr CCAP2L= 0xEC; /* PCA Compare/CAPture value module 2 Low byte */
sfr CCAP3L= 0xED; /* PCA Compare/CAPture value module 3 Low byte */
sfr CCAP4L= 0xEE; /* PCA Compare/CAPture value module 4 Low byte */
sfr B     = 0xF0; /* register B */
sfr CH    = 0xF9; /* PCA Count value High byte */
sfr CCAP0H= 0xFA; /* PCA Compare/CAPture value module 0 Low byte */
sfr CCAP1H= 0xFB; /* PCA Compare/CAPture value module 1 Low byte */
sfr CCAP2H= 0xFC; /* PCA Compare/CAPture value module 2 Low byte */
sfr CCAP3H= 0xFD; /* PCA Compare/CAPture value module 3 Low byte */
sfr CCAP4H= 0xFE; /* PCA Compare/CAPture value module 4 Low byte */

/* BIT Registers */
/* TCON */
sbit TF1  = 0x8F; /* Timer 1 overFlow flag */
sbit TR1  = 0x8E; /* Timer 1 Run control   */
sbit TF0  = 0x8D; /* Timer 0 overFlow flag */
sbit TR0  = 0x8C; /* Timer 0 Run control   */
sbit IE1  = 0x8B; /* External interrupt 1  */
sbit IT1  = 0x8A; /* Interrupt 1 type control bit */
sbit IE0  = 0x89; /* External interrupt 0  */
sbit IT0  = 0x88; /* Interrupt 0 type control bit */
/* P0 */
sbit P0B7 = 0x87;
sbit P0B6 = 0x86;
sbit P0B5 = 0x85;
sbit P0B4 = 0x84;
sbit P0B3 = 0x83;
sbit P0B2 = 0x82;
```

```
sbit P0B1   = 0x81;
sbit P0B0   = 0x80;

/*  SCON  */
sbit SM0   = 0x9F;  /* Serial port Mode bit specifier 0 when PCON.6 is reset */
sbit FE    = 0x9F;  /* Framming Error when PCON.6 is set */
sbit SM1   = 0x9E;  /* Serial port Mode bit specifier 1 */
sbit SM2   = 0x9D;  /* multiprocessor communication enable */
sbit REN   = 0x9C;  /* Received ENable */
sbit TB8   = 0x9B;  /* 9th bit Transmitted for mode 2 & 3 */
sbit RB8   = 0x9A;  /* SCON.2 */
sbit TI    = 0x99;  /* Transmit interrupt flag */
sbit RI    = 0x98;  /* Receive interrupt flag */
/* P1 */
sbit P1B7 = 0x97;
sbit CEX4 = 0x97; /* External I/O for Compare/Capture Module 4 */
sbit P1B6 = 0x96;
sbit CEX3 = 0x96; /* External I/O for Compare/Capture Module 3 */
sbit P1B5 = 0x95;
sbit CEX2 = 0x95; /* External I/O for Compare/Capture Module 2 */
sbit P1B4 = 0x94;
sbit CEX1 = 0x94; /* External I/O for Compare/Capture Module 1 */
sbit P1B3 = 0x93;
sbit CEX0 = 0x93; /* External I/O for Compare/Capture Module 0 */
sbit P1B2 = 0x92;
sbit ECI  = 0x92; /* External Count Input to the PCA */
sbit P1B1 = 0x91;
sbit T2EX = 0x91; /* Timer/Counter 2 Capture/Reload Trigger and Direction Control
*/
sbit P1B0 = 0x90;
sbit T2   = 0x90; /* External Count Input to Timer/Counter 2 */

/* P2 */
sbit P2B7   = 0xA7;
sbit P2B6   = 0xA6;
sbit P2B5   = 0xA5;
sbit P2B4   = 0xA4;
sbit P2B3   = 0xA3;
sbit P2B2   = 0xA2;
sbit P2B1   = 0xA1;
sbit P2B0   = 0xA0;
```

```
/* IE */
sbit EA   = 0xAF; /* Enable All (global enable) */
sbit EC   = 0xAE; /* PCA Enable */
sbit ET2  = 0xAD; /* Timer 2 Enable */
sbit ES   = 0xAC; /* Serial port Enable */
sbit ET1  = 0xAB; /* Timer 1 Enable */
sbit EX1  = 0xAA; /* eXternal 1 Enable */
sbit ET0  = 0xA9; /* Timer 0 Enable */
sbit EX0  = 0xA8; /* eXternal 0 Enable */

/* IP */
                  /* bit 7 for future use */
sbit PPC  = 0xBE; /* PCA Priority */
sbit PT2  = 0xBD; /* Timer 2 Priority */
sbit PS   = 0xBC; /* Serial port Priority */
sbit PT1  = 0xBB; /* Timer 1 Priority */
sbit PX1  = 0xBA; /* eXternal 1 Priority */
sbit PT0  = 0xB9; /* Timer 0 Priority */
sbit PX0  = 0xB8; /* eXternal 0 Priority */
/* P3 */
sbit P3B7 = 0xB7;
sbit RD   = 0xB7; /* ReaD pin */
sbit P3B6 = 0xB6;
sbit WR   = 0xB6; /* WRite pin */
sbit P3B5 = 0xB5;
sbit T1   = 0xB5; /* Timer 1 pin */
sbit P3B4 = 0xB4;
sbit T0   = 0xB4; /* Timer 0 pin */
sbit P3B3 = 0xB3;
sbit INT1 = 0xB3; /* external INTerrupt 1 pin */
sbit P3B2 = 0xB2;
sbit INT0 = 0xB2; /* external INTerrupt 0 pin */
sbit P3B0 = 0xB0;
sbit TXD  = 0xB1; /* serial Transmit Data pin */
sbit P3B1 = 0xB1;
sbit RXD  = 0xB0; /* serial Receive Data pin */


/* T2CON */
sbit TF2  = 0xCF; /* Timer 2 overflow Flag */
sbit EXF2 = 0xCE; /* Timer 2 EXternal Flag */
sbit RCLK = 0xCD; /* Receive CLocK flag */
sbit TCLK = 0xCC; /* Transmit CLocK flag */
```

```
        sbit EXEN2 = 0xCB;  /* Timer 2 external ENable flag */
        sbit TR2   = 0xCA;  /* Timer 2 Run control */
        sbit C_T2  = 0xC9;  /* Counter/Timer 2 select */
        sbit CP_RL2= 0xC8;  /* Capture/Reload flag */

        /* CCON : PCA Counter Control Register */
        sbit CF    = 0xDF;  /* Counter overflow Flag */
        sbit CR    = 0xDE;  /* Run Control bit */
                            /* bit 5 for future use */
        sbit CCF4  = 0xDC;  /* Module 4 interrupt Flag */
        sbit CCF3  = 0xDB;  /* Module 3 interrupt Flag */
        sbit CCF2  = 0xDA;  /* Module 2 interrupt Flag */
        sbit CCF1  = 0xD9;  /* Module 1 interrupt Flag */
        sbit CCF0  = 0xD8;  /* Module 0 interrupt Flag */
        /*  PSW  */
        sbit CY    = 0xD7;  /* CarrY flag */
        sbit AC    = 0xD6;  /* Auxiliary Carry flag */
        sbit F0    = 0xD5;  /* Flag 0 avialable to the user for general purpose */
        sbit RS1   = 0xD4;  /* Register bank Selector bit 1 */
        sbit RS0   = 0xD3;  /* Register bank Selector bit 0 */
                            /* bit 2 for future use */
        sbit OV    = 0xD2;  /* OVer flow flag */
sbit P     = 0xD0;  /* Parity flag */

GLOBAL_R.H
        #define OK       1
        #define NOK      0
        #define LED_ON   1
        #define LED_OFF  0
        #define DIG_ON   0
        #define DIG_OFF  1
        #define REQUEST  1
        #define WD_SET   0xa5
        #define WD_RST   0
        #define DEF_SMP  0x01   /* true DEF_SMP 0x0a, DEF_ALV 0x1e */
        #define DEF_ALV  0x01
        #define DEF_RT   0x004
        #define TEST     0xff
        #define START    0x0020
        #define BUF_SIZ  0x7fc  /* 2k RAM size */
        #define H_ALRM   P3B0
        #define L_ALRM   P3B1
        #define COMP     P3B3
```

```
#define HWRDY    P3B4
#define OUTEN    P3B5
#define GLOBAL   0x1f
#define ADDRESS  (P1&0x1f)
#define IO_TYPE  (P1&0xe0)
#define CNT_OUT  150
#define AD1      0x00
#define AD5      0x70
/* functions */
void hdlc_check();
void init_8273_test();
void operate_8273();
void init_timer_mode3();
void set_timer0(int live);
void data_checking(int byte_num,char *rx_data,char *tx_data);
void result_tx_checking();
void result_rx_checking(char *result);
void command_receive();
void check_receive();
void command_transmit(unsigned char long_frm);
void check_transmit();
void error_receive();
void request();
void contrl();
void variable_default();
void init_timebase();
void init_system_timer();
unsigned adc12(unsigned char channel);
void command(unsigned int p_num);
```

MODE_RT.H

```
/* control mode */
#define REQSTD  0x01
#define CONTROL 0x02
#define ADD_TYP 0x03
#define RESET_T 0x04
#define SET_VAR 0x05
#define MOD_ERR 0x07
#define ONE_F   0x09
/*I/O type of module*/
#define AIN     0x20
#define DIN     0x40
#define DOUT    0x60
```

```
#define AOUT1   0x80
#define AOUT2   0xa0
```

PORT_RT.H

```
/* timer port */
#define TIMER0   XBYTE[0x8000]
#define TIMER1   XBYTE[0x8001]
#define TIMER2   XBYTE[0x8002]
#define TIM_MODE XBYTE[0x8003]
/* hdlc of 8273 port */
#define CMD_STT  XBYTE[0x8004]
#define PAR_RST  XBYTE[0x8005]
#define TxINT    XBYTE[0x8006]
#define RxINT    XBYTE[0x8007]
#define RXDACK   XBYTE[0x8008]
#define TXDACK   XBYTE[0x800c]
/* switch enable port */
#define ENA_INT  XBYTE[0x8010]
#define DIS_ALL  XBYTE[0x8011]
#define ENA_TX   XBYTE[0x8012]
#define ENA_RX   XBYTE[0x8013]
/* status of P0 8751*/
#define CNTL     XBYTE[0x8014]
#define CNTH     XBYTE[0x8018]
/* status bit of 8273 */
#define RxIBIT   ((CMD_STT) & 0x08)
#define TxIBIT   ((CMD_STT) & 0x04)
#define RxRBIT   ((CMD_STT) & 0x02)
#define TxRBIT   ((CMD_STT) & 0x01)

#define SET   = '1'
#define RESET = '0'
#define BUILD != 0
#define ON    == '1'
#define OFF   == '0'
```

SYSTEM.H

```
#define LOW_MATCH      0x90
#define HIG_MATCH      0x01
#define LW_MATCH       0xe8
#define HG_MATCH       0x03
#define LOW_WMATCH     0x91
#define HIG_WMATCH     0x01
```

ประวัติผู้เขียน

นาย ทวีศักดิ์ เรืองพีระกุล เกิดวันที่ 25 พฤศจิกายน พ.ศ. 2511 ที่จังหวัดพิจิตร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า จากคณะวิศวกรรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์ ปีการศึกษา 2533 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ณ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย