ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

ชื่อหัวข้อ ชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X ควบคุมด้วยภาษาซี

The PIC16F87X Microcontroller Demonstator Controled by C Language

ชื่อนักศึกษา

1. นางสาวจิรายุ	ธิป้อ	รหัสประจำตัว	44035362
2. นายทศพล	วงษ์วิบูลย์สิน	รหัสประจำตัว	44035365
3. นายธีระภัทร์	สายไตร	รหัสประจำตัว	44035371

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ที่ปรึกษา อาจารย์สุรพงษ์ สิริพงศ์ดี

อาจารย์ที่ปรึกษาร่วม ผศ.กิติพงค์ มะโน

คณะกรรมการสอบปริญญานิพนธ์		ลายมือชื่อ
1. อาจารย์อมรชัย	ชัยชนะ	
2. อาจารย์สุรพงษ์	สิริพงศ์ดี	
3. อาจารย์สุชิน	อาจหาญ	
4. อาจารย์โกศล	ตราชู	
5. อาจารย์ปิยะ	ศุภวราสุวัฒน์	

วัน/เดือน/ปีที่สอบ วันศุกร์ที่ 4 เมษายน พ.ศ. 2546 เวลา 9:00 น.

สถานที่สอบ ห้อง ค.311 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.วิสุทธิ์ อธิพรธรรม)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 6 เดือน พ.ค. พ.ศ. 2546

ปริญญานิพนธ์

ชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X ควบคุมด้วยภาษาซี

THE PIC16F87X MICROCONTROLLER DEMONSTRATOR CONTROLED BY C LANGUAGE

นางสาวจิรายุ ธิป้อ
นายทศพล วงษ์วิบูลย์สิน
นายธีระภัทร์ สายไตร

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

ปริญญานิพนธ์

เรื่อง ชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X ควบคุมด้วยภาษาซี

The PIC16F87X Microcontroller Demonstrator Controled By C Language

วัตถุประสงค์

1. เพื่อศึกษาโครงสร้างและสถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูล PIC16F87X
2. เพื่อออกแบบวงจรและใบงานการทดลองที่ใช้ในการทดลองการทำงานของโมดูลต่างๆ
3. เพื่อสร้างชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X โดยแยกออกเป็นโมดูลและใบงานการทดลอง
4. เพื่อใช้ทดลองการทำงานของโมดูลแต่ละโมดูลและใบงานในการทดลอง
5. เพื่อสามารถนำชุดทดลองไมโครคอนโทรลเลอร์ตระกูล PIC16F87X และใบงานการทดลองไปใช้ประกอบในการเรียนการสอน

ประโยชน์ที่คาดว่าจะได้รับ

1. มีความรู้ความเข้าใจโครงสร้างและสถาปัตยกรรมของ ตระกูล PIC16F87X
2. ได้วงจรต้นแบบชุดทดลองไมโครคอนโทรลเลอร์ตระกูล PIC16F87X และใบงานการทดลองได้
3. ได้เครื่องต้นแบบชุดทดลองไมโครคอนโทรลเลอร์ตระกูล PIC16F87X และใบงานได้
4. ได้ผลการทดลองของโมดูลแต่ละโมดูลและใบงานการทดลอง
5. สามารถนำชุดทดลองไมโครคอนโทรลเลอร์ตระกูล PIC16F87X และใบงานไปใช้ประกอบในการเรียนการสอนได้

ชื่อหัวข้อ	ชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X ควบคุมด้วยภาษาซี	
นักศึกษา	นางสาวจิรายุ	ธิป้อ
	นายทศพล	วงษ์วิบูลย์สิน
	นายธีระภัทร์	สายไตร
อาจารย์ที่ปรึกษา	อาจารย์สุรพงษ์	สิริพงศ์ดี
อาจารย์ที่ปรึกษาร่วม	ผศ.กิติพงค์	มะโน
หลักสูตร	ครุศาสตร์อุตสาหกรรมบัณฑิต	
สาขาวิชา	อิเล็กทรอนิกส์และคอมพิวเตอร์	
ปีการศึกษา	2545	

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ได้นำเสนอการออกแบบและการสร้างชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X ที่สามารถโปรแกรมไฟล์ได้จากพอร์ตขนานของคอมพิวเตอร์ 25 ขา และยังสามารถต่อใช้งานชุดอุปกรณ์ทดลองต่างๆได้โดยแยกออกเป็นโมดูล ส่วนโมดูลนั้นแยกออกเป็น 6 โมดูล คือ โมดูลหลัก PIC-ICSP การแสดงผลด้วยจอแบบผลึกเหลว การเชื่อมต่อแบบพอร์ตอนุกรม การเชื่อมต่อแบบไอสแควร์ซี การรับอินพุตแบบเครื่องมือวัด การแสดงผลด้วยดอทเมตริกต์ 24x8 นอกจากนี้ได้ออกแบบโปรแกรมการทำงานของชุดทดลองจึงช่วยให้สามารถเข้าใจหลักการทำงาน และการใช้โปรแกรมไมโครคอนโทรลเลอร์ตระกูล PIC16F87X ได้ง่ายขึ้นและยังสามารถนำไปใช้เป็นสื่อการเรียนการสอนในสถานศึกษาต่างๆ ได้

Thesis Title	The PIC16F87X Microcontroller Demonstrator Controled by C Language
Students	Miss Jirayu Tipor
	Mr.Thossapol Wondwiboonsin
	Mr.Teerapat Saitai
Advisor	Mr.Surapong Siripongdee
Co-Advisor	Assist.Prof.Kitipong Mano
Education Level	Bachelor of Science in Industrial Education
Program in	Electronics and Computer
Academic	2002

ABSTRACT

This thesis presents a designing and implementation of PIC16F87X Microcontroller Training Board. The project can be In-Circuit Serial Programming 25 Pins devised. Into Seven module; PIC-ICSP, LCD Display, Board Serial Port Interface, I^2C Interface, Input/Instrument, Output Dotmatrix 24x8 The project helps the user to easier understand the principle of PIC16F87X Microcontroller Training Board and can be applied to use at institute of education.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ลุล่วงไปได้ด้วยดีอันเนื่องมาจากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน ขอขอบคุณอาจารย์สุรพงษ์ สิริพงศ์ดี และคณาจารย์ภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่ให้ความอนุเคราะห์เครื่องมือ และอุปกรณ์ รวมทั้งให้คำแนะนำ แนวความคิด ความรู้ต่างๆ แนวทางแก้ไขปัญหาในการจัดทำปริญญานิพนธ์ ขอขอบคุณห้องสมุดคณะครุศาสตร์อุตสาหกรรม ห้องสมุดคณะวิศวกรรมศาสตร์ ที่ช่วยอำนวยความสะดวกและเอื้อเฟื้อสถานที่ในการค้นคว้าข้อมูล สุดท้ายที่ควรระลึกถึงอย่างยิ่ง บิดา และมารดาที่เป็นผู้ให้ความสนับสนุนด้านการศึกษา และเป็นผู้ให้กำลังใจด้วยดีตลอดมา ตั้งแต่อดีตจนถึงปัจจุบัน

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ไมโครคอนโทรลเลอร์เป็นส่วนหนึ่งของระบบอิเล็กทรอนิกส์ ที่มีในการใช้ควบคุมทางอุตสาหกรรมเป็นอย่างมากซึ่งในปัจจุบันได้มีการพัฒนาไมโครคอนโทรลเลอร์ (Microcontroller) ให้ทำงานในลักษณะ RISC (Reducde Instruction Set Computer) มีคำสั่งการใช้งาน 33-35 คำสั่ง หนึ่งคำสั่งใช้สัญญาณนาฬิกาเพียงลูกเดียวยกเว้นคำสั่งที่ใช้ในการกระโดดคือไมโครคอนโทรลเลอร์ตระกูล PIC ได้รับความนิยมให้นำไปควบคุมสินค้าและผลิตภัณฑ์ต่างๆ ส่งผลให้เกิดความต้องการที่จะเรียนรู้ และใช้งานไมโครคอนโทรลเลอร์ตระกูลดังกล่าว

จึงได้สร้างชุดปฏิบัติการทดลองไมโครคอนโทรลเลอร์ PIC16F87X ขึ้น เพื่อเรียนรู้การทำงานของไมโครคอนโทรลเลอร์ตระกูลนี้ นอกจากนี้ยังช่วยเพิ่มทักษะให้กับผู้เรียนซึ่งการทดลองจะทำให้ผู้เรียนเข้าใจง่ายยิ่งขึ้น

1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังนี้

1. สามารถทำให้ใช้ภาษาซีในการเขียนคำสั่งไมโครคอนโทรลเลอร์
2. สามารถใช้โปรแกรม PCW Compliler IC PIC ได้
3. มีโมดูลการทดลอง 6 โมดูล
   - 3.1 โมดูลหลัก PIC -ICSP
   - 3.2 การแสดงผลด้วยจอแบบผลึกเหลว
   - 3.3 การเชื่อมต่อแบบพอร์ตอนุกรม
   - 3.4 การเชื่อมต่อแบบไอสแควร์ซี
   - 3.5 การรับอินพุตแบบเครื่องมือวัด
   - 3.6 การแสดงผลด้วยดอทเมตริกต์ 24x8

1.3 เนื้อหาโดยสังเขป

เนื้อหาของปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่างๆในแต่ละบทประกอบด้วยเนื้อหาดังต่อไปนี้

บทที่ 1 กล่าวถึงความเป็นมาและความสำคัญของปริญญานิพนธ์ ขีดความสามรถของโครงงาน และเนื้อหาในบทต่างๆ โดยสังเขป

บทที่ 2 ทฤษฎีและหลักการประกอบด้วยเนื้อหาดังต่อไปนี้ คือ ความรู้เบื้องต้นของไมโครคอนโทรลเลอร์ PIC16F87X การจัดการหน่วยความจำของ PIC16F87X รีจิสเตอร์ควบคุมของ PIC16F87X ชุดคำสั่งของ PIC16F87X และการเข้าถึงรีจิสเตอร์และข้อมูลของ PIC 16F87X การใช้งานของอินพุตเอาต์พุตพอร์ต ไทมเมอร์เคาน์เตอร์ภายใน PIC16F87X การใช้งานอีอีพรอม (EEPROM) ภายในคุณสมบัติพิเศษของซีพียู การอินเตอร์รัพต์ของ PIC16F87X การแสดงผลด้วยจอแบบผลึกเหลว การเชื่อมต่อแบบพอร์ตอนุกรม การเชื่อมต่อแบบไอสแควร์ซี การรับอินพุตแบบเครื่องมือวัด การแสดงผลด้วยดอทเมตริกต์ 24x8

บทที่ 3 การออกแบบ การสร้าง และการทำงาน กล่าวถึงเนื้อหาเกี่ยวกับ การออกแบบทางด้านซอฟต์แวร์ ซึ่งแบ่งออกเป็น การออกแบบโปรแกรม PIC-ICSP การส่งข้อมูลโปรแกรมลงหน่วยความจำโปรแกรมของ PIC16F87X การสร้างโปรแกรม PIC-ICSP และ รายละเอียดการสร้าง การใช้ฟังก์ชันเรียกใช้งานโปรแกรม IC-Prog.exe การควบคุมโมดูล PIC-ICSP การใช้งานโปรแกรม IC Prog.exe และการออกแบบทางด้านฮาร์ดแวร์ ซึ่งแบ่งออกเป็นโมดูลหลัก PIC-ICSP การแสดงผลด้วยจอแบบผลึกเหลว การเชื่อมต่อแบบพอร์ตอนุกรม การเชื่อมต่อแบบไอสแควร์ซี การรับอินพุตแบบเครื่องมือวัด การแสดงผลด้วยดอทเมตริกต์ 24x8

บทที่ 4 การทดลองและผลการทดลองประกอบด้วยการทดลองและผลการทดลองโมดูลหลัก PIC-ICSP การแสดงผลด้วยจอแบบผลึกเหลว การเชื่อมต่อแบบพอร์ตอนุกรม การเชื่อมต่อแบบไอสแควร์ซี การรับอินพุตแบบเครื่องมือวัด การแสดงผลด้วยดอทเมตริกต์ 24x8

บทที่ 5 เป็นการสรุปผลการจัดทำโครงงาน ปัญหาที่เกิดขึ้นและแนวทางในการแก้ไขรวมทั้งแนวทางการพัฒนา

ภาคผนวก ก เครื่องต้นแบบชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X

ภาคผนวก ข แสดงรายละเอียดวงจรและแผ่นวงจรพิมพ์ที่ใช้ในโคงงาน

ภาคผนวก ค รายการอุปกรณ์ที่ใช้ในแต่ละวงจร

ภาคผนวก ง แสดงแผนผังการทำงานและรหัสต้นฉบับของโปรแกรมทั้งหมดที่สร้างขึ้นเพื่อประกอบการทำงานของโครงงาน

ภาคผนวก จ ประกอบด้วยใบงานการทดลอง 6 ใบงาน

ภาคผนวก ฉ คู่มือการใช้งานชุดทดลองไมโครคอนโทลเลอร์ PIC16F87X

ภาคผนวก ช รายละเอียดและคุณสมบัติของอุปกรณ์ที่สำคัญในโครงงานนี้

บทที่ 2
ทฤษฎีและหลักการ

2.1 ความรู้เบื้องต้นของไมโครคอนโทรลเลอร์ PIC 16F87X

PIC16F87X เป็นไมโครคอนโทรลเลอร์ในตระกูล PIC (Peripheral Interface Controller) ของบริษัทไมโครชิปเทคโนโลยี (Microchip Technology) ไมโครคอนโทรลเลอร์ตระกูล PIC มีด้วยกันหลายเบอร์แต่ละเบอร์ก็มีความสามารถแตกต่างกันออกไป ภายใน PIC16F87X หน่วยความจำโปรแกรม (Program Memory) แบบแฟลช (Flash) ซึ่งเป็นหน่วยความจำที่สามารถเขียนและลบได้ด้วยสัญญาณไฟฟ้า

ไมโครคอนโทรลเลอร์ PIC16F87X เป็นไมโครคอนโทรลเลอร์สมัยใหม่จัดอยู่ในกลุ่มของไมโครโปรเซสเซอร์แบบ RISC (Reduced Instruction Set computer) คือไมโครคอนโทรลเลอร์ตระกูลนี้จะมีชุดคำสั่งน้อยเพียง 33-35 คำสั่งพื้นฐานเท่านั้นและทุกคำสั่งสามารถทำงานให้เสร็จได้ด้วยการใช้สัญญาณนาฬิกาเพียงลูกเดียว ยกเว้นคำสั่งที่ใช้ในการกระโดดทั้งยังทำงานในลักษณะไปป์ไลน์ (Pipe Line) เหมือนกับไมโครโปรเซสเซอร์สมัยใหม่ ความเร็วในการทำงานจึงสูงมากเมื่อเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นๆ ที่ความถี่ของสัญญาณนาฬิกาเท่ากัน

2.1.1 สถาปัตยกรรมของ PIC16F87X

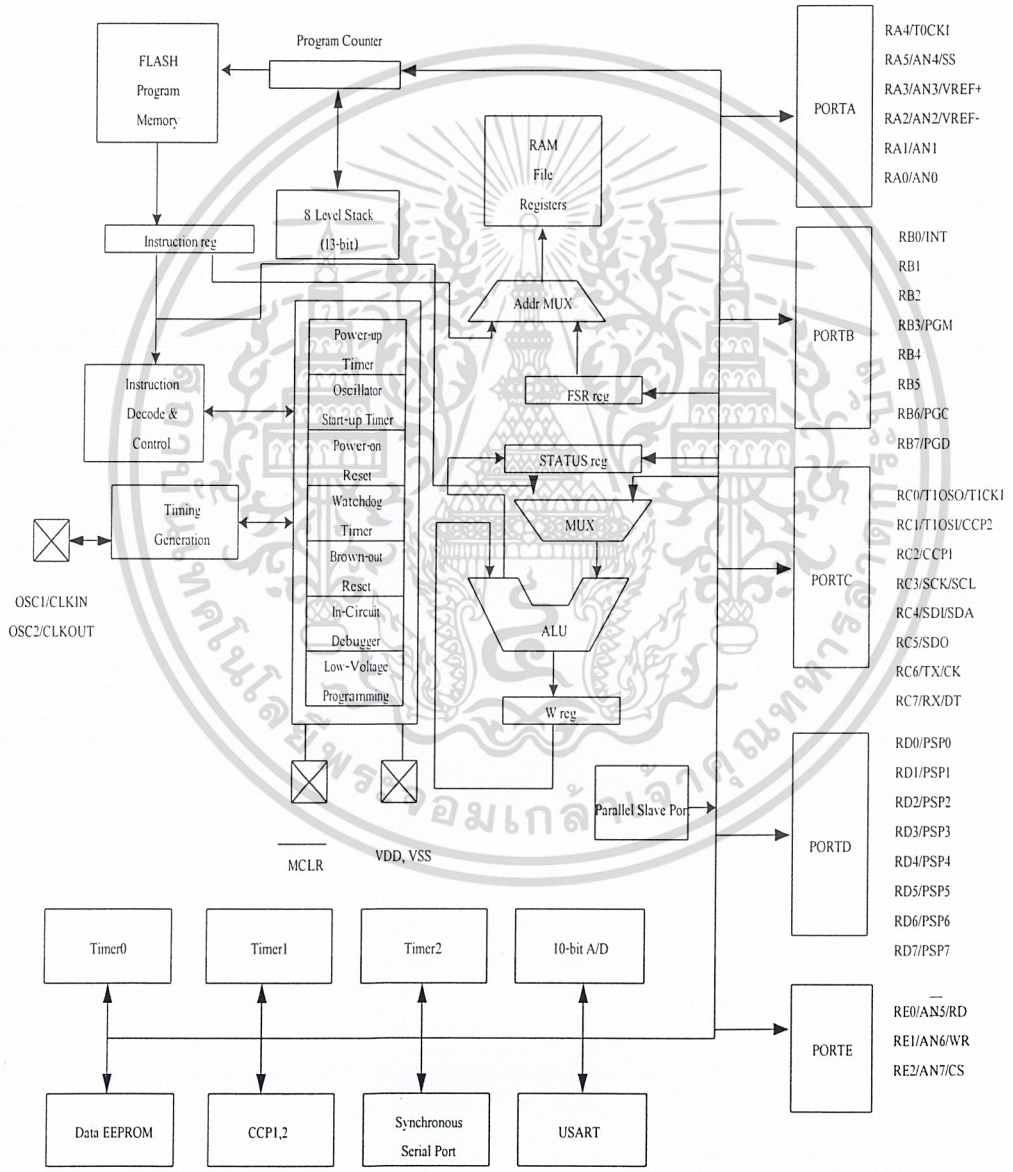
ไมโครคอนโทรลเลอร์ PIC16F87X ได้รับการบรรจุหน่วยประมวลผล หน่วยความจำ หน่วยคำนวณทางคณิตศาสตร์และหน่วยอินพุตเอาต์พุตไว้พร้อมทั้งยังมีไทม์เมอร์และวอตซ์ด็อกครบถ้วนสมบูรณ์

ตารางที่ 2.1 การจัดหน่วยความจำของ PIC 16F87X

การจัดหน่วยความจำของ PIC 16F87X	
ความถี่	DC 4 หรือ 20 MHz
รับค่า	POR, BOR, PWRT, OST
โปรแกรมแฟลช	4 หรือ 8 K
หน่วยความจำข้อมูล	192 หรือ 368 ไบต์
ขัดจังหวะ	13 หรือ 14 แหล่ง

ตารางที่ 2.1 (ต่อ) การจัดหน่วยความจำของ PIC16F87X

การจัดหน่วยความจำของ PIC16F87X	
อินพุตเอาต์พุตพอร์ต	พอร์ต A, B, C, D, E
อินพุตเอาต์พุตพอร์ต	พอร์ต A, B, C, D, E
หน่วยความจำ EEROM	128/256 ไบต์



รูปที่ 2.1 สถาปัตยกรรมของไมโครคอนโทรเลอร์ PIC16F877

2.1.2 การจัดการขาของ PIC16F877

ไมโครคอนโทรลเลอร์ PIC 16F877 สามารถจัดขาต่อใช้งานได้เป็น 4 กลุ่ม คือ

1. กลุ่มขาสัญญาณนาฬิกา

มี 2 ขา คือ OSC1/CLKIN (ขา13) และ OSC2 /CLKOUT (ขา14)

2. กลุ่มขาควบคุม

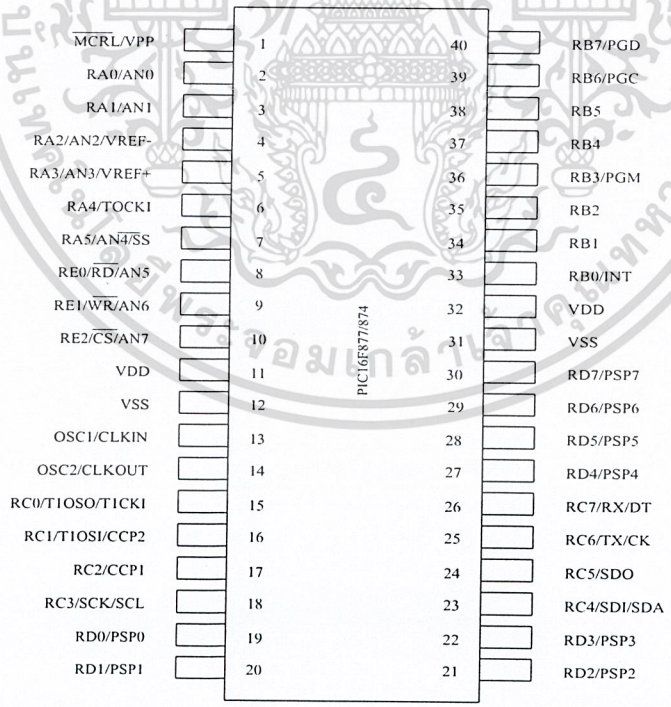
มี 1 ขา คือ MCLR (ขา1)

3. กลุ่มขาพอร์ตอินพุตเอาต์พุต

มี 33 ขา แบ่งเป็นขาพอร์ต A 6 ขา ได้แก่ RA0–RA5 (ขา 2 ถึงขา 7) ขาพอร์ต B ได้แก่ขา RB0–RB7 (ขา 33 ถึงขา 40) ขาพอร์ต C ได้แก่ขา RC0–RC7 (ขา 15 ถึงขา 26) ขาพอร์ต D ได้แก่ RD0–RD3 (ขา 19 ถึงขา 22) RD4–RD7 (ขา 27 ถึงขา 30) และขาพอร์ต E ได้แก่ E0–RE2 (ขา 8 ถึงขา 10)

4. กลุ่มขาไฟเลี้ยง

มี 4 ขา คือ ขา V_{SS} (ขา 12 และขา 3) หรือขาต่อกราวด์ และขา V_{DD} (ขา 11 และขา 32) หรือขาไฟเลี้ยง ปกติใช้ + 5 โวลต์



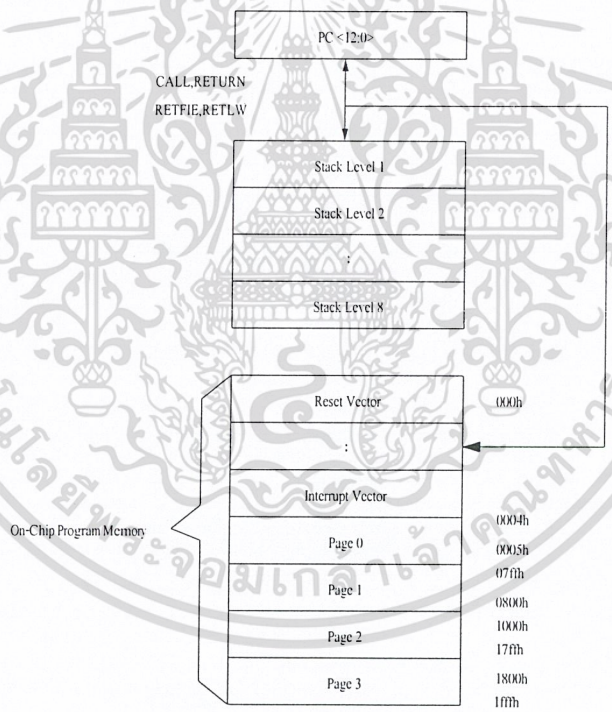
รูปที่ 2.2 การจัดขาใช้งานของ PIC16F877

2.1.3 การจัดหน่วยความจำของ PIC16F87X

การจัดหน่วยความจำของ PIC16F87X แบ่งออกเป็น 3 ส่วน คือ หน่วยความจำโปรแกรม หน่วยความจำข้อมูลและหน่วยความจำส่วนของ EEPROM

2.1.4 หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมของ PIC 16F87X เป็นหน่วยความจำแบบเฟลช ซึ่งสามารถที่จะโปรแกรมลงบนหน่วยความจำนี้ได้ทั้งโหมดโปรแกรม และในขณะทำงานตามปกติ PIC16F87X มีโปรแกรมเคาน์เตอร์ (Program Coumter : PC) ขนาด 13 บิต ที่สามารถอ้างตำแหน่งหน่วยความจำได้ 8K หรือ 14 ตำแหน่ง โดยมีตำแหน่งรีเซตเวคเตอร์ (Reset Vector) อยู่ที่ตำแหน่ง 0000h และที่ตำแหน่งอินเตอร์รัพต์เวคเตอร์ (Interrupt Vector) อยู่ที่ตำแหน่ง 0004h และมีความลึกของสเต็ก (Stack) 8 ระดับ



รูปที่ 2.3 การจัดสรรหน่วยความจำโปรกรมของ PIC 16F87X

1. รีจิสเตอร์ควบคุมของ PIC 16F87X

ใน PIC16F87X มีรีจิสเตอร์ควบคุมที่มีบทบาทสำคัญอยู่ 9 ตัว คือ STATUS, OPTION, INTCON, PCL, PCLATH, PIE1, PIR1, PIE2, PIR2, PCON

2. รีจิสเตอร์ STATUS

เป็นรีจิสเตอร์ที่ใช้แสดงสถานะทางคณิตศาสตร์ของหน่วยประมวลผลทางคณิตศาสตร์ สถานนะการทำงานของ PIC16F87X และใช้เป็นตัวกำหนดการเลือกแบงค์ของหน่วยความจำข้อมูล การเข้าถึงรีจิสเตอร์ STATUS เพื่ออ่านและเขียนข้อมูลสามารถกระทำได้ด้วยวิธีการเดียวกับการอ่านและเขียนรีจิสเตอร์ตัวอื่นๆ

ตารางที่ 2.2 รายละเอียดของบิตต่างๆในรีจิสเตอร์ STATUS

บิต 3	บิต 2	บิต 1	บิต 7	บิต 6	บิต 5	บิต 4	บิต 0
R-1	R/W-X	R/W-X	R/W-0	R/W-0	R/W-0	R-1	R/W-X
PD	Z	DC	IRP	RP1	RP0	TO	C

หมายเหตุ R : อ่านค่าได้ W : เขียนค่าได้ U : ไม่ใช้งาน "0" -n ค่าที่เกิดหลังเพาเวอร์ออนรีเซต

3. รีจิสเตอร์ OPTION

เป็นรีจิสเตอร์ที่ใช้ในการอินเตอร์รัพต์จากสัญญาณจากภายนอก

ตารางที่ 2.3 รายละเอียดของบิตต่างๆ ในรีจิสเตอร์ OPTION

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEG	TOCS	TOSE	PSA	PS2	PS1	PS0

หมายเหตุ R :อ่านค่าได้ W : เขียนค่าได้ U : ไม่ใช้งาน "0"-n ค่าที่เกิดหลังเพาเวอร์ออนรีเซต

ตารางที่ 2.4 การกำหนดอัตราส่วนของปรีสเกลเลอร์

PS2	PS1	PS0	อัตราส่วนเมื่อทำงานกับ WDT	อัตราส่วนเมื่อทำงานกับ TMRO
0	0	0	1:1	1:2
0	0	1	1:2	1:4
0	1	0	1:4	1:8
0	1	1	1:8	1:16
1	0	0	1:16	1:32
1	0	1	1:32	1:64
1	1	0	1:64	1:128
1	1	1	1:128	1:256

4. รีจิสเตอร์ INTCON

เป็นรีจิสเตอร์ที่เก็บค่าบิตของการอีนาเบิลสัญญาณอินเตอร์รัพต์ มีแอดเดรสอยู่ที่ 0x0B มีบทบาทสำคัญมากในเรื่องของการอินเตอร์รัพต์

ตารางที่ 2.5 รายละเอียดของบิตต่างๆในรีจิสเตอร์ INTCON

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

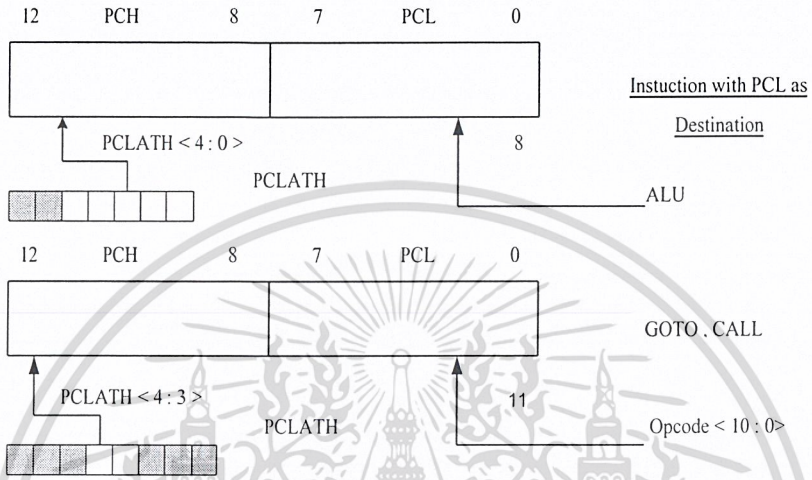
หมายเหตุ R : อ่านค่าได้ W : เขียนค่าได้U : ไม่ใช้งาน "0"-n ค่าที่เกิดหลังเพาเวอร์ออนรีเซต

2.1.5 รีจิสเตอร์โปรแกรมเคาน์เตอร์ PCL และ PCLATH

โปรแกรมเคาน์เตอร์ (Program Counter : PC) เป็นรีจิสเตอร์ที่มีหน้าที่ชี้ตำแหน่งแอดเดรสต่อไปของหน่วยความจำโปรแกรมที่ซีพียูจะต้องไปทำงาน

โปรแกรมเคาน์เตอร์หรือ PC ใน PIC16F87X มีขนาด 13 บิต แบ่งเป็น 2 ส่วนคือ รีจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์ต่ำ หรือ PCL (Program Counter Low Byte) ซึ่งในส่วนนี้มีขนาด 8 บิต สามารถอ่านและเขียนค่าได้โดยตรง ในขณะที่อีกส่วนหนึ่งมีขนาด 5 บิต ไม่สามารถอ่านหรือเขียน

ข้อมูลได้โดยตรงต้องอาศัยการเขียนและอ่านค่าผ่านรีจิสเตอร์ PCLATH โดยรีจิสเตอร์ PCLATH จะทำการเก็บค่าของ 5 บิตบนโปรแกรมเคาน์เตอร์ไว้และถ่ายทอดลงสู่ 5 บิตบนของโปรแกรมเคาน์เตอร์ไบต์สูง (PCH) ก็ต่อเมื่อโปรแกรมเคาน์เตอร์มีการโหลดค่าใหม่เข้ามา ซึ่งจะเกิดขึ้นเมื่อกระทำคำสั่ง CALL หรือ GOTO



รูปที่ 2.4 การถ่ายทอดข้อมูลภายในโปรแกรมเคาน์เตอร์

1. สแต็ก (Stack)

ในไมโครคอนโทรลเลอร์ PIC16F87X ได้จัดสรรสแต็กหรือพื้นที่ในหน่วยความจำ เพื่อใช้ในการเก็บค่าของโปรแกรมเคาน์เตอร์ชั่วคราวไว้ 8 ระดับ หากมีการพุชเป็นครั้งที่ 9 ซีพียูจะนำค่าของ PC ในครั้งที่ 9 นี้เก็บลงในสแต็กที่เก็บค่า PC ในครั้งที่ 1

2. รีจีสเตอร์ W

ในตัวไมโครคอนโทรลเลอร์ PIC16F87X มีรีจิสเตอร์ที่ใช้ในการทำงานหลักคือ รีจิสเตอร์ W หากเปรียบเทียบกับ ไมโครคอนโทรลเลอร์เบอร์อื่นๆ รีจีสเตอร์ W เทียบได้กับแอกคิวมูเลเตอร์ (Accumulator) เมื่อ PIC16F87X กระทำคำสั่งทางคณิตศาสตร์ รีจิสเตอร์ W จะเป็นรีจิสเตอร์ที่ซีพียูติดต่อด้วยการโอนข้อมูลหรือการตรวจสอบข้อมูลจะกระทำที่รีจิสเตอร์ W

3. รีจีสเตอร์ไฟล์ (File Register)

เนื่องจาก PIC16F87X มีรีจิสเตอร์ที่เกี่ยวข้องอยู่หลายตัว จึงมีการจัดรวบรวมรีจีสเตอร์ทั้งหมดที่ต้องใช้การเข้าถึงแบบโดยอ้อมไว้ในลักษณะแฟ้มข้อมูล

รีจิสเตอร์ได้รับการนำมารวมไว้มีชื่อว่า รีจิสเตอร์ไฟล์ (Register File) มีขนาด 8 บิต และมีรีจิสเตอร์ฟังก์ชันพิเศษ ที่ถูกกำหนดหน้าที่และตำแหน่งไว้แล้ว และอีก 68 ตัวจะได้รับการกำหนดให้ใช้งานอย่างอิสระ

2.1.6 การใช้งานตัวรับตัวส่งสัญญาณ

ไมโครคอนโทรลเลอร์ PIC16F87X มีพอร์ตสำหรับติดต่อกับอุปกรณ์ภายนอก 3 พอร์ต คือ พอร์ต A มีขนาด 6 บิต และพอร์ต B กับ พอร์ต C มีขนาด 8 บิต ขาแต่ละของ PIC16F87X สามารถจ่ายกระแสออกได้สูงสุด 25 มิลลิแอมป์ และสามารถรับกระแสสูงสุดต่อขาได้ 20 มิลลิแอมป์ เมื่อใช้ไฟเลี้ยง +5โวลต์ แต่ถ้าจะนำไปขับ LED จะต้องจำกัดกระแสโดยการต่อค่าความต้านทานเข้าไป แต่ถ้าใช้ไฟเลี้ยง +3โวลต์ ก็สามารถที่จะขับ LED ได้โดยตรง

1. พอร์ต A และรีจีสเตอร์ TRISA

พอร์ต A เป็นพอร์ตแบบสองทิศทาง มีขนาด 6 บิต มีรีจิสเตอร์กำหนดทิศทางการถ่ายทอดข้อมูลของพอร์ต A คือ TRISA เมื่อต้องการใช้งานเป็นอินพุตต้องเขียนข้อมูล “1” ไปยังบิตที่ต้องการกำหนดให้เป็นอินพุต และถ้าหากต้องการใช้งานเป็นเอาต์พุตก็จะต้องเขียนข้อมูล “0” ไปยังบิตที่ต้องการให้เป็นเอาต์พุต

สำหรับขาของพอร์ต A บิตที่ 4 หรือ RA/TOCKI มีความแตกต่างจากขาอื่นๆ ตรงที่ว่าขานี้จะมีวงจรชมิตทริกเกอร์ (Schmitt Trigger) ต่อเข้าที่ขาอินพุต เพราะขานี้ใช้เป็นขาอินพุตรับสัญญาณนาฬิกาภายนอกสำหรับขาไทม์เมอร์เคาน์เตอร์ TMRO ภายใน

ขาของพอร์ต A ยังมัลติเพล็กซ์กับแอนะลอกอินพุต (Analog Input) และแอนะลอก V_{REF} อินพุต เมื่อต้องการใช้งานก็จะทำการกำหนดโดยเลือกเคลียร์ หรือเซต (Clearing/Setting) ที่บิตคอนโทรล (Control Bit) ในรีจีสเตอร์ ADCON1 (A/D Control Register1)

ขณะเพาเวอร์ออนรีเซตขาของพอร์ต A จะถูกกำหนดให้เป็นแอนะลอกอินพุต และกำหนดลอจิกเป็น “0”

2. พอร์ต B และรีจิสเตอร์ TRISB

พอร์ต B เป็นพอร์ตแบบสองทิศทาง มีขนาด 8 บิต มีรีจิสเตอร์กำหนดทิศทางการถ่ายทอดข้อมูลของพอร์ต B คือ TRISB การกำหนดทิศทางจะกำหนดเช่นเดียวกับ TRISA

แต่ละขาของพอร์ต B สามารถเลือกให้พูลอัปโดยทำการเคลียร์บิต RBPU ในรีจิสเตอร์ OPTION และการพูลอัปจะถูกยกเลิกโดยอัตโนมัติเมื่อกำหนดให้พอร์ต B เป็นเอาต์พุต และนอกจากนี้การพูลอัปจะถูกยกเลิกเมื่อเกิดเพาเวอร์ออนรีเซต

ในส่วนของขาอีก 4 ขา คือ (RB4-RB7) ยังใช้เป็นขาเพื่อทำการตรวจสอบการเปลี่ยนแปลงข้อมูลหรือระดับสัญญาณของพอร์ต B เพื่อกระตุ้นให้เกิดการอินเตอร์รัพต์ กล่าวคือ จะสามารถ

รับสัญญาณอินเตอร์รัพต์ได้เฉพาะขาที่เป็นอินพุต (ถ้าบิต RB4-RB7 บิตใดบิตหนึ่งเป็นเอาต์พุตจะไม่สามารถรับสัญญาณอินเตอร์รัพต์ได้)

ขณะที่อยู่ในโหมดสลีปสามารถทำให้อุปกรณ์เกิดการอินตอร์รัพต์ได้โดยผู้ใช้ สู่การบริการอินเตอร์รัพต์ย่อยสามารถทำการเคลียร์อินเตอร์รัพต์ได้ดังนี้

2.1 อ่านหรือเขียนจากพอร์ต B อย่างใดอย่างหนึ่งจะทำให้หยุดการทำงาน

2.2 เคลียร์บิต RBIF

3. พอร์ต C และรีจิสเตอร์ TRISC

พอร์ต C เป็นพอร์ตแบบสองทิศทาง มีขนาด 8 บิต มีรีจีสเตอร์กำหนดทิศทางการถ่ายทอดข้อมูลของพอร์ต C คือ TRISC

พอร์ต C จะมัลติเพล็กซ์กับฟังก์ชันอุปกรณ์เสริมมากมาย และพอร์ต C นี้ยังมีบัฟเฟอร์ (Buffer) ทางอินพุตแบบชมิตทริกเกอร์ เมื่อทำการอีนาเบิล (Enable) ฟังก์ชันอุปกรณ์เสริม ควรระวังในการกำหนดค่าของบิต TRISC และในบางอุปกรณ์เสริมจะทำการโอเวอร์ไรด์ (Override) บิต TRIS เพื่อกำหนดให้ขานี้เป็นอินพุตหรือเอาต์พุต และควรจะหลีกเลี่ยงตำแหน่งปลายทาง ผู้ใช้ควรจะอ้างอิงถึงส่วนอุปกรณ์เสริมที่คล้ายคลึงสำหรับการตั้งค่าบิต TRIS ให้ถูกต้อง

4. PORTD และ PORTE

สำหรับ PORTD และ PORTE จะไม่มีอยู่ใน PIC ในตระกูลที่มีขนาดขา 28 ขา ก่อนอื่นมาพูดกันถึง PORTD ก่อน PORTD จะเป็นพอร์ตขนาด 8 บิตซึ่งจะมี Schmitt Trigger input buffer อยู่ในตัว โดยที่เราสามารถกำหนดแต่ละบิตของพอร์ตให้เป็น Input หรือ output ได้โดยอิสระจากกัน PORTD สามารถที่จะทำตัวเป็น parallel slave port ได้อีกด้วย โดยทำได้โดยการ set psp mode bit ซึ่งใน Mode นี้ Buffer ภายในจะกลายเป็นแบบ TTL

สำหรับ PORTE จะมีทั้งหมด 3 ขา คือ RE0/(RD\)/AN5, RE1/(WR\)/AN6 และRE5/(CS\)/AN7 ซึ่งจะมี Schmitt Trigger input buffer อยู่ในตัวโดยที่เราสามารถกำหนดแต่ละบิตของพอร์ตให้เป็น Input หรือ Output พอร์ตสามารถกลายเป็น Control Input สำหรับ Microprocessor port เมื่อทำการ SET PSP MODE BIT ข้อควรระวัง เมื่ออยู่ในโหมดนี้ก็คือ ต้องตรวจดูให้ดี Trise ตั้งแต่บิต 0-2 ถูก set และต้องแน่ใจว่า ADCON1 ถูก Set ให้อยู่ใน mode diggital I/O ซึ่งใน mode นี้ input buffer จะเป็น TTL

2.1.7 การใช้งาน EEPROM ภายใน

PIC16F87X มีหน่วยความจำแบบ EEPROM โดยสามารถอ่านและเขียนในขณะทำงานปกติได้ แต่ต้องโดยการเข้าถึงนั้นจะต้องทำผ่านรีจิสเตอร์ฟังก์ชั่นพิเศษซึ่งต้องใช้ถึง 4 ตัวดังนี้

1. EECON1 และ EECON2

EECON1 เป็นรีจิสเตอร์ที่ใช้ควบคุมการเข้าถึงหน่วยความจำ และ EECON2 ใช้เป็นลำดับ (Sequence) ของการเขียนเท่านั้นไม่สามารถอ่านและเขียนในขณะทำงานปกติ

บิต EEPGO เป็นบิตที่ใช้กำหนดว่าจะเข้าถึง EEPROM หรือหน่วยความจำแบบแฟลช (Flash Memory) คือ ถ้าเป็น "0" จะเป็นการเข้าถึงอีอีพรอม (EEPROM) ถ้าเป็น "1" เป็นการเข้าถึงหน่วยความจำแบบแฟลช

บิต RD และ WR เป็นบิตที่ใช้กำหนดว่าจะเป็นการอ่านหรือเขียน โดยทั้งสองบิตนี้สามารถเซตได้โดยซอฟต์แวร์อย่างเดียวแล้วจะถูกเคลียร์โดยฮาร์ดแวร์ (Hardware) เมื่ออ่านหรือเขียนเสร็จเรียบร้อย

บิต WREN เป็นบิตที่ใช้อีนาเบิลการเขียนและปกติจะถูกดีสเอเบิล (Disable) หรือเป็น "0"

บิต WRERRใช้บอกว่าการเขียนเกิดขัดจังหวะจากการรีเซต (Reset) หรือวอตช์ด็อกทำงาน ซึ่งบิตนี้จะเซต "1" ถ้าเกิดกรณีทั้งสองขณะเขียนข้อมูลลงอีอีพรอมและถ้าไม่เกิดกรณีทั้งสองขณะเขียนข้อมูลจะเป็น "1"

ตารางที่ 2.6 รายละเอียดของรีจิสเตอร์ EECON1

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
U	U	U	R/W-0	R/W-X	R/W-0	R/S-0	R/S-X
-	-	-	EEIF	WRERR	WREN	WR	RD

หมายเหตุ R :อ่านค่าได้ W : เขียนค่าได้ U : ไม่ใช้งาน "0" -n ค่าที่เกิดหลังเพาเวอร์ออนรีเซต

2. EEDATA

EEDATA ใช้เป็นที่พักข้อมูลขนาด 8 บิตที่ต้องการอ่านหรือเขียนมีจำนวน 64 ไบต์ มี แอดเดรสอยู่ที่ 0x00–0x3F การเขียนและอ่านข้อมูลจะกระทำในระดับไบต์หรือครั้งละ 8 บิตเท่านั้น การเขียนข้อมูลลงในหน่วยความจำอีอีพรอมทุกครั้งต้องทำการลบข้อมูลออกก่อนเสมอ ซึ่งอัตราเร็วในการลบข้อมูลจะสูงในขณะที่อัตราเร็วในการเขียนข้อมูลจะขึ้นอยู่กับไทม์เมอร์ในตัว PIC16F87X ซึ่งจะเปลี่ยนแปลงตามแรงดันและอุณหภูมิในขณะที่ทำการเขียนข้อมูลนั้น

เมื่อทำการป้องกันข้อมูลในหน่วยความจำอีอีพรอมแล้วซีพียูสามารถอ่านและเขียนข้อมูลในหน่วยความจำได้เป็นปกติ แต่เครื่องโปรแกรมภายนอกจะไม่สามารถเข้าถึงหน่วยความจำส่วนนี้

3. EEADR

เป็นรีจิสเตอร์ที่ใช้เก็บค่าแอดเดรสของหน่วยความจำข้อมูลอีอีพรอมที่ต้องการเขียนโดยมีค่าตั้งแต่ 00H ถึง FFH (256 ไบต์)

2.2 การเชื่อมต่อแบบไอสแควร์ซี

I^2C (Interface-IC Communication) หมายถึง การติดต่อสื่อสารระหว่างไอซีโดยบัส I^2C การติดต่อสื่อสารแบบ นี้ ได้รับการพัฒนาขึ้นโดยฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการติดต่อให้ไอซี หรือโมดูลสามารถติดต่อสั่งงานและควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นหนึ่งคือสายข้อมูลอีกเส้นคือสายสัญญาณนาฬิกาที่ใช้กำหนดจังหวะการทำงานการต่อร่วมกันของอุปกรณ์บนบัส I^2C ทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัวจะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data Line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock Line) อุปกรณ์ที่ทำการเชื่อมต่อบนบัส I^2C มีหลากหลายไม่ว่าจะเป็นไอซีขยายพอร์ตอินพุตเอาต์พุต (I/O Expander) ไอซีแปลงสัญญาณแอนะลอกเป็นสัญญาณดิจิตอล (ADC) และแปลงสัญญาณดิจิตอลเป็นสัญญาณแอนะลอก (DAC) ไอซีเรียลไทม์คล็อก (RTC) ไอซีขับจอแสดงผลแบบผลึกเหลวหน่วยความจำอีพรอม (EPROM) และไมโครคอนโทรลเลอร์

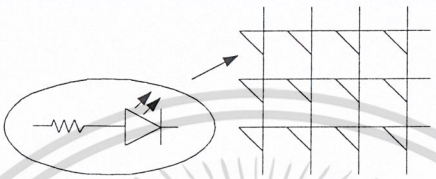
คุณสมบัติโดยทั่วไปของบัส I^2C สาย SDA และ SCL เป็นสายสัญญาณแบบ 2 ทิศทาง (Bi-Directional Line) ต้องมีการต่อตัวต้านทานพูลอัพกับแรงดัน +5 โวลต์เพื่อให้สถานะลอจิกสูงขณะที่ไม่มีการติดต่อใช้งานและมีลักษณะเป็นวงจรเดรนเปิด (Open-Drain) หรือคอลเล็กเตอร์เปิด (Open-Collector)

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลไบต์ ต่อ วินาทีในโหมดปกติ (Standard Mode) และสูงถึง 400 กิโลบิต ต่อ วินาทีในโหมดความเร็วสูง (Fast Mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัสแบบ I^2C นี้จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 พิโคฟารัด การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่า คือ 7 บิต (7-Bit Addressing) หรือ 10 บิต (10-Bit Addressing)

ข้อเด่นอีกประการหนึ่งของบัส I^2C คือ สามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้ใช้ไฟเลี้ยง +12 โวลต์การต่อร่วมกันบนบัส I^2C สามารถกระทำได้ในลักษณะเดียวกับกรณีที่อุปกรณ์ทั้งสองต้องใช้ไฟเลี้ยงเท่ากัน

2.3 ดอทเมตริกซ์

การแสดงผลอย่างง่ายๆ คือ การแสดงผลแบบแถวเดียวจะควบคุมสัญญาณเพียงด้านเดียวคือส่งแต่ข้อมูลแล้วหน่วงเวลาไว้ระยะหนึ่งแล้วจึงส่งสัญญาณตัวต่อไปก็จะทำให้เราเห็นเป็นไฟวิ่งได้ จากนั้นเราสามารถแสดงให้เห็นเป็นตัวอักษรหรือรูปภาพก็ได้ถ้าเราเพิ่มลอจิกการควบคุมซึ่งแทนที่จะควบคุมการติดดับเพียงด้านเดียวก็เป็น 2 ด้าน



รูปที่ 2.5 การต่อไดโอดเปล่งแสงแบบเมตริกซ์

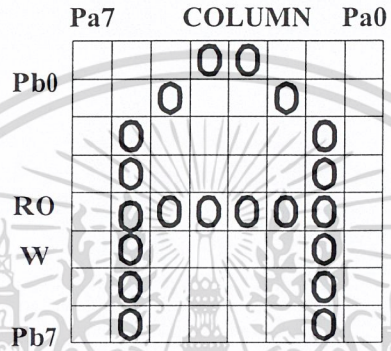
จะเห็นว่าถ้าเราส่งข้อมูลออกไปทางแถวและให้คอลัมน์ใดเป็นศูนย์ (0) จะทำให้ไดโอดเปล่งแสงถูกไบอัสตรง และติดสว่างตามบิตข้อมูลที่ส่งออกไปทางแถว จากหลักการนี้จึงสามารถควบคุมไดโอดเปล่งแสงทุกดวง ในแผงเมตริกซ์ (Matrix) ให้ติดหรือดับได้ตามต้องการ

หลักการเกิดภาพบนไดโอดเปล่งแสงแบบเมตริกซ์เนื่องจากการมองเห็นของตามนุษย์นั้นจะมีการคงภาพ คือ เห็นภาพอยู่นาน 1/6 วินาที ถึงแม้ว่าภาพนั้นจะไม่ปรากฏแล้วก็ตาม จากหลักการของการมองเห็นนี้ ถ้าเราทำให้ไดโอดเปล่งแสงติดและดับตามจุดต่างๆ ได้เร็วกว่า 1/6 เราก็จะเห็นว่าไดโอดเปล่งแสงนั้นติดค้างอยู่

แต่ซีพียูทำงานด้วยความเร็วสูงการที่จะติดและเปลี่ยนตำแหน่งการติดไปเรื่อยๆตามความเร็วของซีพียูจะเห็นว่าไดโอดเปล่งแสงนั้นติดสว่างเรื่องๆ ทุกดวง ก็เพราะว่าความสัมพันธ์ของกระแสความอิ่มตัวของทรานซิสเตอร์รวมทั้งกระแสที่ทำให้ไดโอดเปล่งแสงติดเป็นปัจจัยทำให้เห็นไดโอดเปล่งแสงติดทุกดวงเมื่อไดโอดเปล่งแสงดวงใดติดแล้วควรทำการหน่วงเวลาเพื่อให้มีกระแสเพียงพอที่จะทำให้ไดโอดเปล่งแสงติดสว่างยิ่งหน่วงเวลานานเท่าใดยิ่งทำให้กระแสเข้าใกล้จุดสูงสุดแต่เนื่องจากเราต้องทำการกวาด (Scan) หลายๆ คอลัมน์ถ้าทำการหน่วงเวลามากเกินไปจะทำให้ไดโอดเปล่งแสงในลำดับถัดไปมีวงรอบในการนำกระแสเป็นช่วงๆ ซึ่งระยะเวลาในการนำกระแสของไดโอดเปล่งแสงแต่ละดวงห่างไกลจากความไวแสงของตามนุษย์ ซึ่งมีผลทำให้เห็นเป็นภาพกระพริบหรือพริ้วในการกวาดแบบคอลัมน์นี้จะเห็นว่าเวลาที่จะทำให้ไดโอดเปล่งแสงแต่ละดวงติดนานเท่าไรนั้นขึ้นอยู่กับจำนวนคอลัมน์ของไดโอดเปล่งแสงด้วย

สามารถแก้ไขได้โดยเมื่อจำนวนคอลัมน์มากเกินไปเราควรเปลี่ยนมาเป็นการกวาดทางแถวแทนซึ่งจะทำให้เวลาของไดโอดเปล่งแสงแต่ละดวงที่จะติดมีเวลาคงที่มากยิ่งขึ้นและความสว่างคงที่มากขึ้น

การคิดข้อมูลที่จะนำมาแสดงเป็นตัวอักษรมิได้ถูกกำหนดเป็นมาตรฐานหรือเป็นทฤษฎีแต่ขึ้นอยู่กับฮาร์ดแวร์นั้นๆ ฉะนั้นข้อมูลที่ทำให้ไดโอดเปล่งแสงติดได้ คือ ข้อมูลจะต้องเป็นลอจิกบวกและถ้าจะทำให้ไดโอดเปล่งแสงดับข้อมูลจะต้องเป็นลอจิกลบหรือลอจิกศูนย์



รูปที่ 2.6 การสร้างตัวอักษร

2.4 การสื่อสารข้อมูลแบบอนุกรม

ลักษณะของการสื่อสารข้อมูลแบบอนุกรมจะเป็นการสื่อสารที่ทำการรับ-ส่งข้อมูลที่ใช้สายจำนวนน้อย ซึ่งปกติจะใช้เพียง 1 คู่เท่านั้น คือ สายสัญญาณที่จะใช้เป็นสายข้อมูลและสายกราวด์ ลักษณะของการรับ-ส่งข้อมูล ข้อมูลจะถูกส่งออกไปหรือรับเข้ามาในลักษณะที่เป็นบิตต่อบิต ซึ่งถ้าเปรียบเทียบกันกับการสื่อสารข้อมูลแบบขนานที่จำนวนข้อมูลและอัตราเร็วในการสื่อสารข้อมูลเท่ากันแล้วจะพบว่าการสื่อสารข้อมูลแบบอนุกรมจะต้องใช้เวลาในการรับ-ส่งข้อมูลมากกว่าการรับ-ส่งข้อมูลแบบขนาน แต่เมื่อพิจารณาของการสื่อสารข้อมูลแบบอนุกรมแล้ว จะพบข้อดีของการสื่อสารแบบอนุกรมคือ การใช้สายสัญญาณน้อยกว่ากัน และสามารถส่งสัญญาณในระยะทางที่ไกลกว่า แม้ว่าอัตราการลดทอนหรือผิดเพี้ยนของสัญญาณที่มีผลจากความยาวของสายนำสัญญาณจะมีค่าเท่ากับการสื่อสารข้อมูลแบบขนนาน การสื่อสารข้อมูลแบบอนุกรมจะมีวิธีในการลดผลจากการลดทอนของสัญญาณนี้โดยอาศัยหลักการรับ–ส่งสัญญาณแบบดิฟเฟอเรนเชียลดังนั้นการสื่อสารข้อมูลแบบอนุกรมจึงเหมาะใช้กับการสื่อสารข้อมูลในระยะทางไกลหรือการสื่อสารที่ต้องใช้สายหรือช่องสัญญาณในการรับ–ส่งข้อมูลจำนวนน้อย เช่นการสื่อสารข้อมูลโครงข่ายแบบท้องถิ่น

2.4.1 การสื่อสารข้อมูลแบบอนุกรมที่แบ่งตามทิศทางของข้อมูล

นอกจากที่ได้กล่าวมาแล้วนั้นจะพบว่าการสื่อสารข้อมูลแบบอนุกรมยังสามารถที่จะแบ่งตามลักษณะของทิศทางตามโครงสร้างและความต้องการของระบบดังต่อไปนี้

1. การสื่อสารข้อมูลในทิศทางเดียวตลอดเวลา หรือแบบซิมเพล็กซ์ (Simplex)

เป็นการสื่อสารข้อมูลที่สามารถส่งข้อมูลได้ทางเดียวเท่านั้น เมื่อทำการสื่อสารในทิศทางใดก็จะใช้ทิศทางนั้นตลอดเวลา ไม่มีการเปลี่ยนทิศทาง เช่น การส่งสัญญาณภาพจากโทรทัศน์ไปยังเครื่องรับโทรทัศน์หรือการส่งข้อมูลจากศูนย์บริการไปยังวิทยุติดตามตัว

2. การสื่อสารข้อมูลแบบ 2 ทิศทางตลอดเวลา หรือแบบฮาล์ฟดูเพล็กซ์ (Half Duplex)

เป็นการสื่อสารข้อมูลที่ส่งได้ 2 ทิศทาง โดยจะทำการส่งข้อมูลในลักษณะของการผลัดกันรับและส่ง โดยในขณะเวลาหนึ่งนั้นสัญญาณจะไปได้ในทิศทางเดียวเท่านั้น ดังนั้นอุปกรณ์แต่ละตัวที่จะเชื่อมต่อหรือสื่อสารข้อมูลในลักษณะนี้จะต้องเป็นได้ทั้งตัวรับและตัวส่ง ซึ่งมีชื่อเรียกว่า ทรานซีฟเวอร์ (Tranceiver) และจะต้องมีวงจรที่จะเลือกว่า ณ เวลานั้นจะทำงานเป็นตัวรับหรือตัวส่ง

3. การสื่อสารข้อมูลแบบ 2 ทิศทางตลอดเวลา หรือแบบฟูลดูเพล็กซ์ (Full Duplex)

เป็นการสื่อสารข้อมูลที่คล้ายกับแบบฮาล์ฟดูเพล็กซ์ แต่เป็นการสื่อสารข้อมูลใน 2 ทิศทางแบบตลอดเวลา

2.4.2 การสื่อสารข้อมูลแบบอนุกรมมาตรฐาน RS-232

มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง มาตรฐานนี้ในช่วงแรกจะใช้คอนเนคเตอร์ (Connector) เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่–3 ถึง–12โวลต์ แสดงว่ามีข้อมูล (Mark) และ +3 โวลต์ ถึง +12 โวลต์ แสดงว่าเป็นช่องว่าง (Space)

มาตรฐาน RS-232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminationg : DCE) ไว้ว่าอุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรลเลอร์ หรือ ไมโครคอมพิวเตอร์ (Micro Computer) ซึ่งมีความสามารถในการสร้างบิตข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232

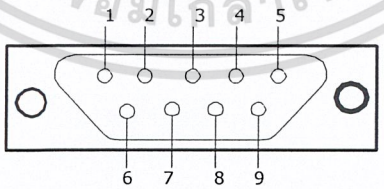
ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัดคือ คอนเนคเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเนคเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเนคเตอร์ที่อยู่ที่โมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานบนคอมพิวเตอร์ พอร์ตอนุกรม RS-232 มักถูกใช้เชื่อมต่อกับโมเด็มหรือเมาส์ (Mouse) โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุดถึง 20 เมตร

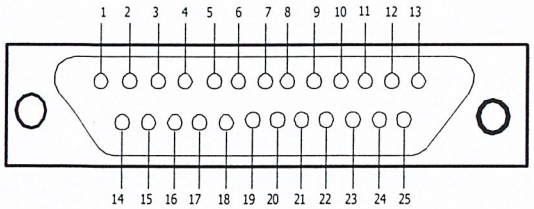
พอร์ตติดต่อสื่อสารกับอุปกรณ์ภายนอกของคอมพิวเตอร์แบบอนุกรมมาตรฐาน RS-232 นี้เป็นระบบการส่งข้อมูลในรูปแบบอนุกรม คือ ข้อมูลจะส่งไปได้ทีละ 1 บิต โดยจะมีอัตราการส่งเป็นบิตต่อวินาที หรืออัตราการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่า Baud Rate โดยจะมีการส่งบิตเริ่มต้น (Start Bit) มีระดับสัญญาณเป็น “0” และบิตข้อมูล (Data Bit) ซึ่งอาจจะมีข้อมูล 7 หรือ 8 บิต และอาจตามด้วยบิตพาริตี้ (Parity Bit) ซึ่งอาจจะเป็นแบบคู่ (Even) หรือแบบคี่ (Odd) เพื่อตรวจความถูกต้องของข้อมูล บิตพาริตี้นี้อาจจะมีหรือไม่มีก็ได้ และสุดท้ายจะตามด้วยบิตจบการทำงาน (Stop Bit) ซึ่งอาจจะมีความกว้างของสัญญาณเป็น 11.5 หรือ 2 บิตก็ได้ ซึ่งการส่งข้อมูลแบบนี้ จำเป็นต้องมีข้อตกลงกันระหว่างการรับกับการส่งคือ

1. ความเร็วในการส่ง Baud Rate
2. จำนวนข้อมูล
3. มีบิตพาริตี้หรือไม่ถ้ามีจะเป็นแบบคู่หรือคี่
4. จำนวนบิตเริ่มต้นเป็น 11.5 หรือ 2 บิต

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเนคเตอร์แบบ DB-25 ตัวผู้หรือ DB-9 ตัวผู้ ซึ่งคอนเนคเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเนคเตอร์แบบ DB-9 เนื่องจากขาอื่นๆ ที่เคยใช้งานในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป



รูปที่ 2.7 คอนเนคเตอร์อนุกรม 9 ขา หรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)



รูปที่ 2.8 คอนเนคเตอร์อนุกรม 25 ขา หรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

สำหรับรายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS-232 มีดังนี้

ขา 1 DCD:Carrier Detect หรืออาจเรียกว่า Carrier Detect :CD ขานี้จะแอกตีฟ (Active) เมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูล เช่น โมเด็ม สำหรับการใช้งานปกติขานี้จะไม่ได้ถูกใช้งานมากนัก

ขา 2 RXD:Receive Data ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามาคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์บัฟเฟอร์

ขา 3 TXD:Transmited Data ขานี้ใช้เพื่อส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

ขา 4 DTR:Data Terminal Ready เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่า ต้องการติดต่อด้วย โดยขา DTR นี้จะต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทาง และขา DTR ของอุปกรณ์ปลายทางจะต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ ถ้าใช้การเชื่อมต่อเป็นแบบ Null Modem ซึ่งใช้สายในการเชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DSR ของตัวมันเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วย

ขา 5 GND:Signal Ground ขากราวด์ของระบบ

ขา 6 DSR:Data Set Ready ใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR

ขา 7 RTS:Request To Send เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null Modem 3 สาย จะต้องเชื่อมต่อขา RTS และ CTS ของตัวมันเองเข้าด้วยกัน เพื่อจะให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

ขา 8 CTS:Clear To Send ขารับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ ข้อมูลที่ขา TXD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

ขา 9 RI:Ring Indicator ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งานจะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มและโปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

ตารางที่ 2.7 การจัดขาของคอนเนคเตอร์พอร์ตอนุกรมตามมาตรฐาน RS-232 แบบ DB-9 และ DB-25

DB-9	DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	DCD : Carrier Detect	อินพุต
2	3	RXD : Receive Data	อินพุต
3	2	TXDTransmited Data	เอาต์พุต
4	20	DTR :Data Terminal Ready	เอาต์พุต
5	7	GND : Signal Ground	-
6	6	DSR : Data Set Ready	อินพุต
7	4	RTS : Request To Send	เอาต์พุต
8	5	CTS : Clear To Send	อินพุต
9	22	RI : Ring Indicator	อินพุต

2.5 การอินเตอร์เฟสกับคีย์บอร์ด

โครงงานที่ถูกสร้างขึ้นส่วนมากแล้วจำเป็นต้องติดต่อกับอุปกรณ์อินพุต เช่น ตัวเซ็นเซอร์ต่างๆ สัญญาณไฟฟ้า, คีย์บอร์ด เป็นต้น โดยเฉพาะคีย์บอร์ดที่ใช้ติดต่อกับผู้ใช้งานในการควบคุมเครื่อง การอินเตอร์เฟสกับคีย์บอร์ดมี 6 รูปแบบที่ใช้งานกันอยู่ทั่วไป สำหรับตรวจจับสถานะการปิด/เปิดของสวิตช์ ในที่นี้ขอยกตัวอย่างเฉพาะการเชื่อมต่อสวิตช์ในแต่ละรูปแบบเป็นหลัก เพื่อให้สามารถมองเห็นแนวทางการเขียนโปรแกรมได้ ซึ่งแต่ละรูปแบบของการเขื่อมต่อสวิตช์ก็จะมีทั้งข้อดี และข้อเสียแตกต่างกันออกไป

1. แบบเชื่อมต่อสวิตช์โดยตรงกับพอร์ต

การเชื่อมต่อสวิตช์โดยตรงกับพอร์ตซึ่งเป็นแบบที่ง่ายที่สุด การกดสวิตช์แต่ละตัวจะทำให้ขาพอร์ตต่อลงกราวด์โดยตรง ซึ่งไม่จำเป็นต้องมีวงจรจำกัดกระแสก็ได้ เพราะว่าภายในตัวไมโครคอนโทรลเลอร์มีตัวต้านทานพูลอัพต่ออยู่แล้ว การเขียนโปรแกรมทำได้ง่ายมาก โดยการใช้คำสั่ง

ตรวจสอบสถานะของแต่ละบิตในพอร์ต เช่นคำสั่ง JB หรือ JNB ข้อเสียของวงจรแบบนี้คือสิ้นเปลืองจำนวนขาพอร์ตจำนวนมาก ถ้าต้องการใช้สวิตซ์มากเท่าใดก็ต้องขยายพอร์ตให้ได้มากตามไปด้วย ซึ่งเป็นเรื่องยุ่งยาก เช่น ถ้าต้องการออกแบบเป็นคีย์บอร์ดสำหรับป้อนตัวอักษรขนาด 60 คีย์ ผู้ออกแบบต้องจัดสร้างขาพอร์ตให้ได้ถึง 60 ขาพอร์ต ซึ่งแทบจะเป็นไปได้ยากและสิ้นเปลืองอุปกรณ์มาก

2. การเชื่อมต่อสวิตซ์แบบเมตริกซ์

การเชื่อมต่อสวิตซ์แบบเมตริกซ์ แต่ละตัวจะถูกต่อกันแบบแถว และคอลัมน์ในรูปแบบของเมตริกซ์ การตรวจสอบการกดคีย์ใดบนคีย์บอร์ดทำได้โยการป้อนค่าตรวจสอบค่าหนึ่งไปยังด้านคอลัมน์ และตรวจสอบการเปลี่ยนแปลงที่เกิดขึ้นทางด้านแถว หรือกล่าวได้ว่าตำแหน่งของการกดคีย์ที่ได้จากการเขียนโปรแกรมทางด้านสแกนคีย์ทางด้านแถว ข้อดีของการเชื่อมต่อสวิตซ์แบบเมตริกซ์ คือ สามารถใช้สวิตซ์ได้มากขึ้นในขณะที่สิ้นเปลืองขาพอร์ตจำนวนน้อย เช่น ใช้ขาพอร์ตเพียง 16 ขาพอร์ต สามารถต่อคีย์บอร์ดได้ถึง 64 คีย์ ส่วนข้อเสียของวงจรนี้ คือ ไม่สามารถรับการกดคีย์บอร์ดพร้อมกันได้ และต้องเขียนโปรแกรมในการตรวจสอบคีย์ที่ยุ่งยากซับซ้อน

3. การเชื่อมต่อสวิตซ์ผ่านชิฟต์รีจิสเตอร์

การเชื่อมต่อสวิตซ์ผ่านชิฟต์รีจิสเตอร์จะเห็นว่าวงจรนี้ต้องการขาพอร์ตน้อยมากเพียง 3 ขาพอร์ต โดยใช้ขาพอร์ตหนึ่งกำหนดโหมดที่ขา Shift/Load ให้อยู่ในสภาวะโหลดเพื่ออ่านสถานะสวิตซ์ทุกตัวเข้าสู่ชิฟต์รีจิสเตอร์ขั้นต่อไป คือ เปลี่ยนโหมดการทำงานให้อยู่ในสภาวะชิฟต์ข้อมูล และให้กำเนิดพัลส์จำนวน 8 พัลส์ เพื่อทำการชิฟต์ข้อมูลสถานะของสวิตซ์ผ่านขาพอร์ต P12 ไปใช้งาน นั่นคือใช้ขาพอร์ตเพียง 3 ขา พอร์ตเท่านั้นก็สามารถทำงานได้แล้ว รวมทั้งการเขียนโปรแกรมก็สามารถทำได้ง่าย ถ้าต้องการเพิ่มจำนวนคีย์มากขึ้น ทำได้โดยเพิ่มชิฟต์รีจิสเตอร์มากขึ้นให้เท่าเทียมกัน อย่างไรก็ตามข้อเสียที่เกิดขึ้น คือ ใช้เวลาในการอ่านสถานะของคีย์ทั้งหมดเป็นเวลานานจนกระทั่งชิฟต์ข้อมูลได้ครบตามจำนวนคีย์

4. แบบเชื่อมต่อสวิตซ์มัลติเพล็กซ์

วงจรการเชื่อมต่อแบบนี้ต้องการขาพอร์ตจำนวน 4 ขาพอร์ต พอร์ต P1.0 P1.1 และ P1.2 ทำหน้าที่ควบคุมไอซีมัลติเพล็กซ์เซอร์ เพื่อเลือกสวิตซ์ S1-S6 ที่ต้องการจะติดต่อด้วยสถานะของคีย์ที่ถูกเลือกจะถูกส่งกลับไปยังไมโครคอนโทรลเลอร์ผ่านขาพอร์ต P1.3 วงจรแบบนี้สามารถเพิ่มวงจรคีย์ได้ง่าย และได้เป็นจำนวนมาก อีกทั้งการเขียนโปรแกรมก็ทำได้ง่ายเช่นเดียวกัน

5. แบบเชื่อมต่อสวิตซ์เข้ากับระบบบัสข้อมูล

วงจรการเชื่อมต่อแบบนี้ทำการต่อสวิตซ์เข้ากับระบบบัสข้อมูล เพื่อทำการอ่านสถานะของสวิตซ์เข้าสู่ระบบบัสข้อมูลของไมโครคอนโทรลเลอร์ การเชื่อมต่อแบบนี้มักถูกนำมาใช้เมื่อไม่มีขา

พอร์ตเหลือไว้ใช้งานเลย การติดต่อกับคีย์บอร์ดทำได้โดยการอ้างตำแหน่งของแอดเดรสไปยังวงจรดีโค้ดเดอร์ และทำการแอกทีฟขาสัญญาณ RD เพื่อส่งสัญญาณอีนาเบิลไปยังไอซีบัฟเฟอร์ส่งผ่านสถานะของไอซีต่างๆ เข้าสู่ระบบบัสข้อมูล ถึงแม้ว่าถ้าพิจารณาดูที่การเขียนโปรแกรมจะเห็นว่าไม่ยากเลย แต่ในทางปฏิบัติการต่อวงจรแบบนี้จะทำให้เกิดความยุ่งยากในทางฮาร์ดแวร์มาก เพราะต้องการเชื่อมต่อวงจรเข้ากับทั้งระบบบัสแอดเดรสและบัสข้อมูล

6. การเชื่อมต่อต่ออุปกรณ์แสดงผลแบบ LCD

ปัจจุบันระบบไมโครคอนโทรลเลอร์ได้หันมาใช้อุปกรณ์แสดงผลแบบแอลซีดีแทนอุปกรณ์แสดงผลแบบเดิมที่ใช้แอลอีดีด้วยเหตุผลต่างๆดังนี้

อุปกรณ์แสดงผลแบบแอลซีดีมีราคาถูกลง สามารถแสดงผลเป็นตัวเลข ตัวอักษร และกราฟิกได้ ในขณะการแสดงผลแบบแอลอีดีไม่สามารถทำได้ทั้งหมด มีอุปกรณ์ควบคุม (Controller) การแสดงผลอยู่ภายในโดยไมโครโปรเซสเซอร์ไม่ต้องเสียเวลาในการสแกนการแสดงผลแต่ละหลัก สามารถสร้างตัวอักขระและกราฟิกต่างๆได้

ในส่วนของความสว่างของการแสดงผลนั้นตัวแอลซีดีจะสู้หลอดแอลซีดีไม่ได้ เพราะว่าการแสดงผลของมันจะต้องใช้แสงจากภายนอก และจะต้องมองในมุมมองที่ตรง แต่จะกินพลังงานน้อยกว่าหลอดแอลซีดีมาก ในการให้แอลซีดีแสดงผลทำได้โดยการเขียนรหัสควบคุมการแสดงผลให้กับมัน และส่งข้อความที่จะแสดงให้กับมันเท่านั้น จากนั้นมันจะแสดงผลของมันโดยอิสระโดยที่ไมโครคอนโทรลเลอร์ไม่ต้องยุ่งกับมันอีกอุปกรณ์แสดงผลแบบแอลซีดีนี้บางครั้งจะเรียกว่าแอลซีดีโมดูลเนื่องจากภายในประกอบด้วยอุปกรณ์ต่างๆหลายส่วน เช่น รีจิสเตอร์คำสั่ง (Instruction Register : IR) ทำหน้าที่รับคำสั่งควบคุมการแสดงผลหน่วยความจำการแสดงผล (Display Data RAM : DDRAM) เป็นหน่วยความจำที่ใช้เก็บข้อมูลที่จะแสดงผล หน่วยความจำรวมตัวอักษร (Character Generator ROM : GROM) เป็นหน่วยความจำที่เก็บสัญลักษณ์และอักขระต่างๆที่จะแสดงผล ถ้าหากแอลซีดีโมดูลต้องการแสดงผลอะไรบ้างมันจะนำเอาข้อมูลแต่ละตัวออกไปจากหน่วยความจำส่วนนี้ หน่วยความจำแรมเก็บตัวอักษร (Character Generator RAM : GRAM) เป็นหน่วยความจำที่ใช้สร้างตัวอักษรเพิ่มเติมขึ้นมาใหม่ได้

ปัจจุบันตัวแอลซีดีโมดูลแบบแสดงผลตัวอักขระมีอยู่หลายรูปแบบ เช่น แบบ 16x2 จะแสดงผลได้ 2 บรรทัด แต่ละบรรทัดแสดงผล 16 ตัวอักษร แบบ 20x1 จะแสดงผล 20 ตัวอักษรบรรทัดเดียว แบบ 20x4 จะแสดงผลได้ 4 บรรทัด แต่ละบรรทัดแสดงผลได้ 20 ตัวอักษร เป็นต้น ขาต่างๆของแอลซีดีมอดูลจะมี 14 ขา

ขา V_{CC} และ V_{SS}จะใช้ต่อกับไฟเลี้ยง +5 โวลต์และกราวด์ ส่วน V_{EE} จะเป็นขาปรับแรงดันไฟเพื่อควบคุมความสว่างของการแสดงผล

ขา RS, Register Select เนื่องจากภายในแอลซีดีมอดูลจะมีรีจิสเตอร์ที่สำคัญภายใน 2 ตัว คือ รีจิสเตอร์เก็บคำสั่ง และเก็บข้อมูล ขา RS นี้จะเป็นตัวเลือกว่าข้อมูลที่ส่งเข้าไปจะเป็นคำสั่งหรือข้อมูล โดยถ้า RS = 0 หมายความว่าข้อมูลที่เข้ามาเป็นคำสั่ง และถ้า RS = 1 หมายความว่าข้อมูลที่เข้ามา เป็นค่าข้อมูลที่จะแสดงผลบนจอแอลซีดี

ขา R/W, Read/Write เป็นขาอินพุต ถ้าขานี้เป็นลอจิก “1” จะเป็นการอ่านข้อมูลจากแอลซีดี ถ้าขานี้เป็น “0” จะเป็นการเขียนข้อมูลลงในแอลซีดี

ขา E, Enable เป็นขาอีนาเบิลให้แอลซีดีทำงานโดยป้อนสัญญาณพัลส์เข้าไปหนึ่งลูกโดยพัลส์นี้จะต้องมีความกว้าง 450 นาโนวินาที เป็นอย่างน้อย

ขา D0-D7 เป็นขาที่รับส่งข้อมูลระหว่างแอลซีดีโมดูลกับอุปกรณ์ภายนอก ในการทำให้แอลซีดีโมดูลทำการแสดงผลเริ่มต้นจะต้องเขียนคำสั่งควบคุมแอลซีดีลงไปก่อนซึ่งจะเป็นการกำหนดให้แสดงผลในลักษณะต่างๆ จากนั้นก็ส่งรหัส ASCII ของตัวอักขระต่างๆที่จะแสดงผลออกไปให้กับแอลซีดี

บทที่ 3
การออกแบบ การสร้าง และการทำงาน

3.1 กล่าวนำ

ชุดปฏิบัติการทดลองไมโครคอนโทรลเลอร์ PIC16F87X ประกอบด้วย 2 ส่วน คือ ส่วนของฮาร์ดแวร์ (Hardware) และส่วนของซอฟต์แวร์ (Software) ซึ่งส่วนของซอฟต์แวร์ คือโปรแกรม PCW สำหรับควบคุมการทำงานของโมดูลหลัก ที่ใช้โหลดข้อมูลโปรแกรมจากคอมพิวเตอร์ ลงในหน่วยความจำโปรแกรมของ PIC16F87X และส่วนที่เป็นฮาร์ดแวร์นั้น ประกอบไปด้วยวงจรต่างๆ ที่ใช้ในการทดลองเชื่อมต่อกับไมโครคอนโทรลเลอร์ PIC16F87X โดยได้ออกแบบและสร้างอยู่ในรูปของ "โมดูล" (Module) แต่ละโมดูลจะเชื่อมต่อกับโมดูลหลัก

3.2 การออกแบบทางด้านฮาร์ดแวร์

3.2.1 โมดูลหลัก PIC-ICSP (PIC- In-Circuit Serial Programming Module)

PIC-ICSP Module เป็นโมดูลหลักสำหรับการทดลองใช้งานไมโครคอลโทรลเลอร์ ตระกูล PIC16F87X ซึ่งสามารถใช้งานได้หลายเบอร์ ทั้งแบบที่มีขาต่อใช้งานแบบ 40 ขา โดยเบอร์ไอซีที่ใช้งานกับ PIC-ICSP Module มีดังนี้

แบบ 40 ขา เช่น เบอร์ PIC16F874 และPIC16F877

PIC-ICSP Module สามารถดาวน์โหลด (Down Load) ไฟล์เฮก (Hex File) จากคอมพิวเตอร์ลงบนตัวไมโครคอนโทรลเลอร์ได้ ในรูปแบบ In-Circuit Serial Programming จากพอร์ตขนานของคอมพิวเตอร์ และสั่งรัน (Run)โปรแกรมได้ทันทีโดยสั่งงานจากเครื่องคอมพิวเตอร์

1. คุณสมบัติของ PIC-ICSP Module

1.1 สามารถใช้ได้กับไมโครคอนโทรลเลอร์ตระกูล PIC16F87X หลายเบอร์ เช่น เบอร์ PIC16F874,PIC16F876 และเบอร์ PIC16F877

1.2 สามารถเลือกโหมดการใช้สัญญาณนาฬิกาได้ทั้ง 2 แบบ คือ แบบคริสตอล (Crystal) 4,20 เมกะเฮิรตซ์ และแบบ RC โปรแกรมในรูปแบบดาวน์โหลดด้วยวิธี In-Circuit Serial Programming

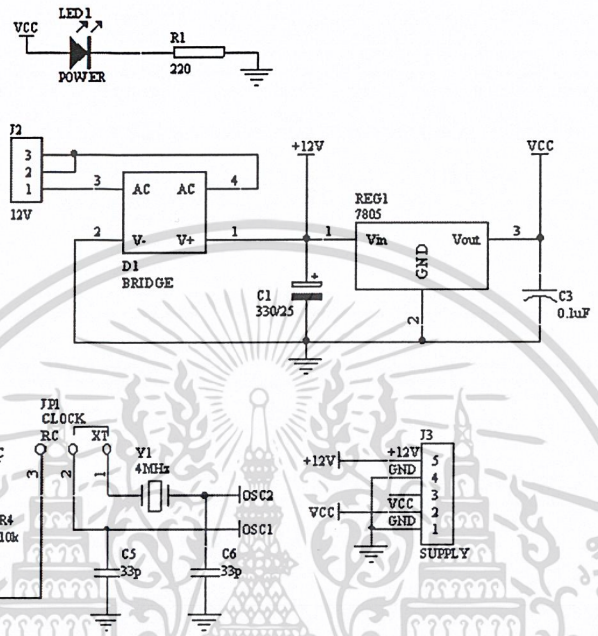
1.3 สั่งทำงาน (Run) หยุดทำงาน (Stop) ได้จากคอมพิวเตอร์

1.4 เชื่อมต่อกับคอมพิวเตอร์ทางพอร์ตขนาน (Parallel Port)

1.5 ใช้แหล่งจ่ายแรงดัน 5โวลต์ และ 12 โวลต์

1.6 โปรแกรมในรูปแบบดาวน์โหลดด้วยวิธี In-Circuit Serial Programming

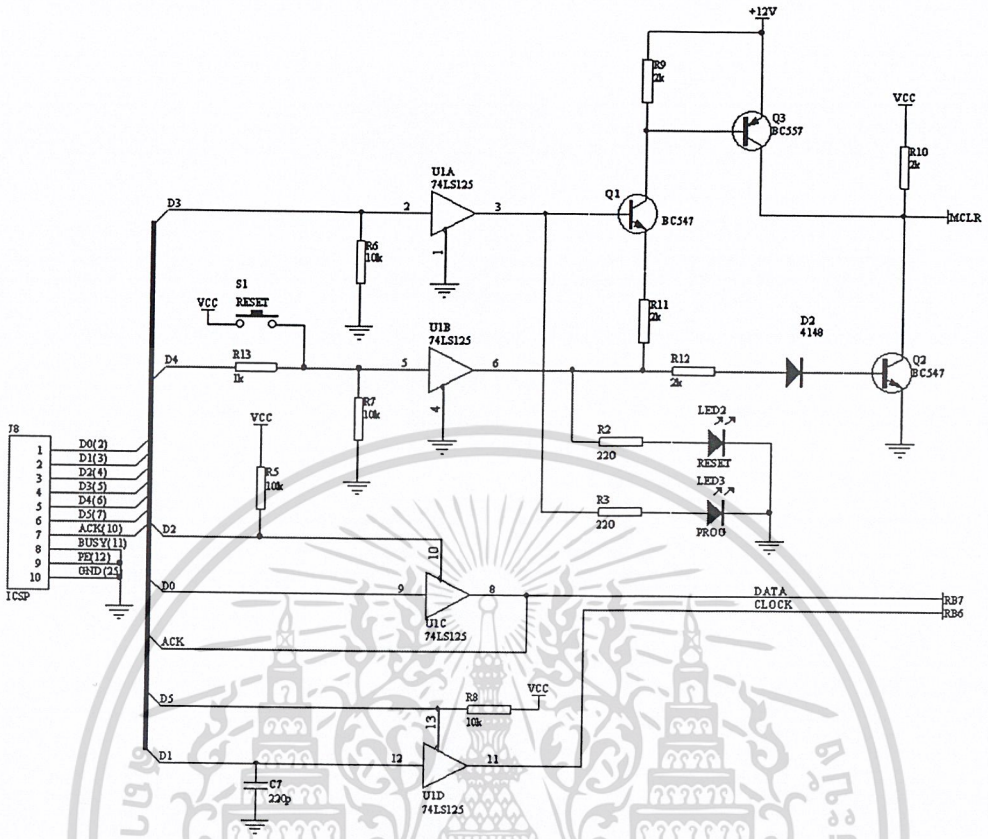
2. วงจรของโมดูล PIC-ICSP Module



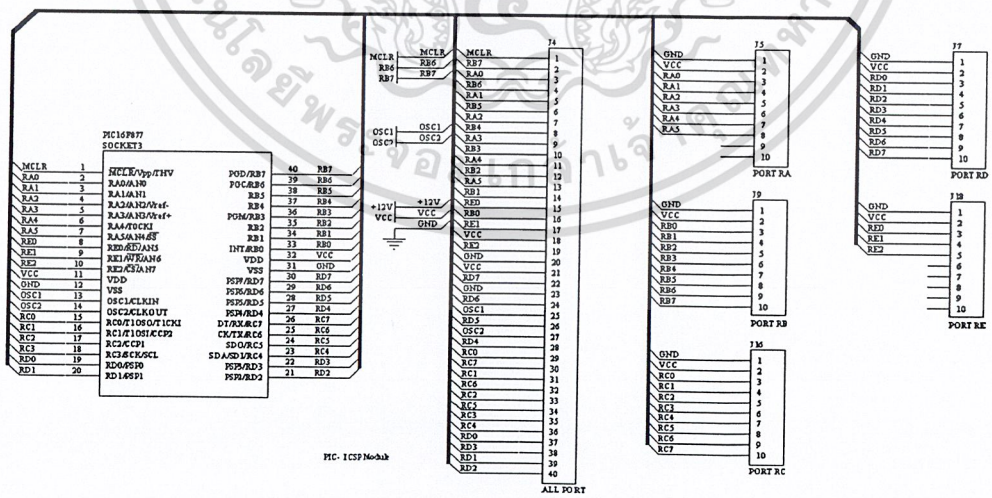
รูปที่ 3.1 ภายจ่ายไฟ และกำเนิดความถี่

คุณสมบัติการโปรแกรมของไมโครคอนโทรลเลอร์ตระกูล PIC16F87X จะใช้ขาสัญญาณจำนวน 3 เส้น คือ Data (RB7) Clock (RB6) และ MCLR ในการโปรแกรม ส่งข้อมูลในแบบอนุกรม ขา MCLR จะรับแรงดันไฟ 12 โวลต์ จะเป็นวงจรสำหรับดาวน์โหลดข้อมูลผ่านทางพอร์ตขนานของคอมพิวเตอร์ โดยจะผ่านไอซีเบอร์ 74LS125 ทำหน้าที่เป็นบัฟเฟอร์ (Buffer) ซึ่งเป็นแบบเอาต์พุต 3 สถานะ (State) ใช้แยกสัญญาณที่พอร์ต RB6 และ RB7 ในขณะรันและโปรแกรม

ที่ขา MCLR จะมีทรานซิสเตอร์สำหรับป้อนแรงดันให้ CPU โดยจะมีแรงดันอยู่ 3 ระดับ คือ แรงดัน 12 โวลต์ สำหรับการโปรแกรม แรงดัน 5 โวลต์ สำหรับขณะรัน และแรงดัน 0 โวลต์ เมื่อรีเซต (Reset) วงจรมี LED ใช้แสดงการโปรแกรมและการรีเซตของซีพียู สำหรับการรีเซตนั้นสามารถสั่งได้ทั้งจากคอมพิวเตอร์หรือโดยการกดสวิตช์บนตัวโมดูล



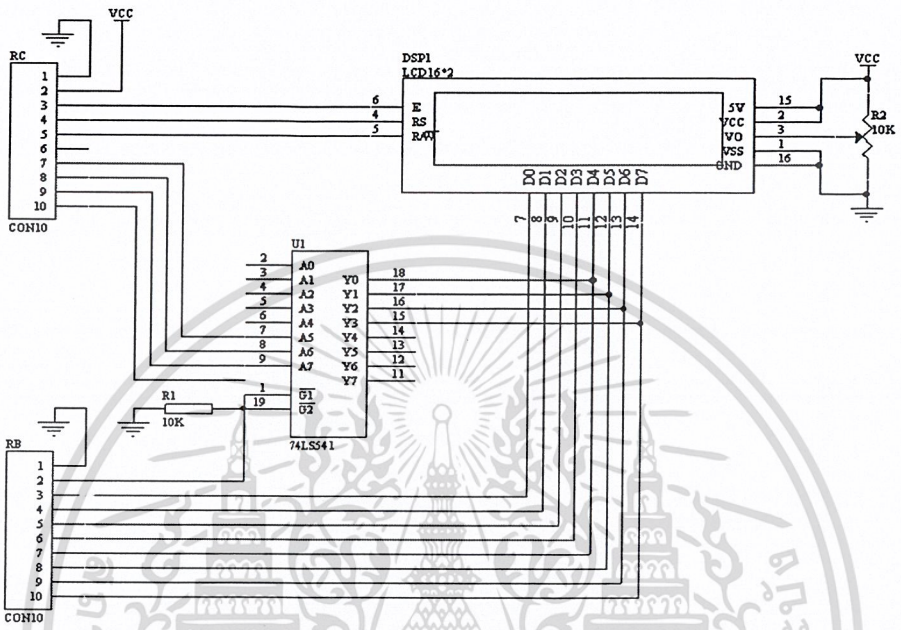
รูปที่ 3.2 ภาคตัดต่อการดาวน์โหลดข้อมูล



รูปที่ 3.3 การเชื่อมต่อ PIC 16F87X กับพอร์ตต่างๆ

3.2.2 การแสดงผลจอแบบผลึกเหลว (LCD Display)

การแสดงผลด้วยจอแบบผลึกเหลวตัวอักษรขนาด 16 x 2 (16 ตัวอักษร 2 บรรทัด) สามารถต่อใช้งานได้ทั้งแบบ 4 บิต และแบบ 8 บิต วงจรมีลักษณะดังรูปที่ 3.4



รูปที่ 3.4 วงจรของการแสดงผลด้วยจอแบบผลึกเหลว

การใช้งานจอแสดงผลแอลซีดี มีมาตรฐานการต่อใช้งาน คือ มีขาสำหรับต่อใช้งานทั้งหมดจำนวน 14 เส้น แบ่งเป็นขาสัญญาณข้อมูล 8 เส้น ขาสัญญาณควบคุม 4 เส้น และเป็นขาสำหรับป้อนแรงดันอีก 2 เส้น มาตรฐานการใช้งานของแอลซีดีสามารถใช้งานได้ทั้งแบบ 4 บิต และแบบ 8 บิต ซึ่งจำนวนบิตนี้คือบิตของสัญญาณข้อมูล เมื่อต้องการใช้งาน 8 บิตก็จะใช้สัญญาณข้อมูลทั้ง 8 เส้น แต่เมื่อต้องการต่อใช้งานแบบ 4 บิตก็ใช้สัญญาณข้อมูล 4 เส้น คือ D4-D7 ส่วนขาที่ไม่ใช้งานก็ต่อลงกราวน์

จากวงจรรูปที่ 3.4 เมื่อต้องการใช้งานจอแสดงผลแอลซีดีแบบ 8 บิต สามารถต่อได้โดยใช้คอนเนคเตอร์จำนวน 2 พอร์ต ใช้เป็นพอร์ตควบคุม 1 พอร์ต ขนาด 3 บิต คือ บิตควบคุมขาอีนาเบิล (E) ขาอินพุตที่ใช้ในการแยกชนิดของข้อมูลว่าเป็นคำสั่ง หรือเป็นข้อมูล ถ้าเป็น “0” ข้อมูลที่เข้ามาจะเป็นคำสั่ง ถ้าเป็น “1” ข้อมูลที่ส่งมาเป็นข้อมูลสำหรับการแสดงผล (RS) และขาที่ใช้เลือกในการอ่านหรือเขียนข้อมูล คือถ้าเป็น “0” เป็นการกำหนดให้เขียน แต่ถ้าเป็น “1” จะเป็นการอ่านข้อมูล (R/W) ของแอลซีดี และใช้เป็นพอร์ตสัญญาณข้อมูลอีก 1 พอร์ต ขนาด 8 บิต และเมื่อต้องการต่อใช้

งานแบบ 4 บิต ทำโดยการถอดคอนเนคเตอร์ที่ใช้ส่งสัญญาณข้อมูล 8 บิตนั้นออก แต่จะใช้สัญญาณข้อมูลผ่านพอร์ตเดียวกับพอร์ตควบคุมรวมทั้งหมด 7 บิต

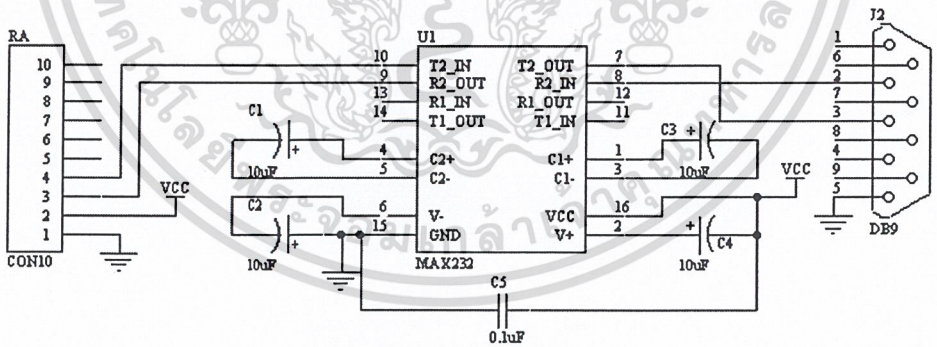
การทำงานของวงจรตัวเลือกว่าจะติดต่อแบบ 4 บิต หรือ 8 บิต จะใช้ไอซีเบอร์ 74LS541 ทำหน้าที่อีนาเบิลหรือดิสเอเบิลสัญญาณข้อมูลขนาด 4 บิต ที่อยู่ในพอร์ตควบคุม

เมื่อต่อสัญญาณข้อมูล 8 บิต เข้าไป จะมีแรงดัน VCC จากคอนเนคเตอร์เข้าไปดิสเอเบิลตัวไอซี 74LS541 ที่ขา G1 และ G2 ทำให้ไอซีตัดสัญญาณจากพอร์ต 4 บิตออก แล้วนำสัญญาณข้อมูลจากพอร์ต 8 บิตไปใช้แทน จากนั้นก็สั่งงานการเชื่อมต่อจากไมโครคอนโทรลเลอร์

ตัวต้านทานปรับค่า R2 สำหรับปรับความสว่าง (Brightness) ให้กับแอลซีดี โดยปรับเมื่อปรับค่า R2 มาทางซ้ายสุดจะทำให้จอแอลซีดีสว่างที่สุด และเมื่อปรับค่า R2 มาทางขวามือสุดก็จะทำให้จอแอลซีดีสว่างน้อยที่สุด

3.2.3 การเชื่อมต่อแบบพอร์ตอนุกรม (Board Serial Port Interface)

จากคุณสมบัติพิเศษของไมโครคอนโทรลเลอร์ตระกูล PIC16F87X สามารถเชื่อมต่ออุปกรณ์สื่อสารในรูปแบบสัญญาณอนุกรมได้โดยผ่าน USART (Universal Synchronous Asynchronous Receiver Transmitter) ในตัวไมโครคอนโทรลเลอร์ โมดูลนี้จะมีชุดอุปกรณ์สื่อสารแบบอนุกรมให้ใช้ในมาตรฐาน RS-232



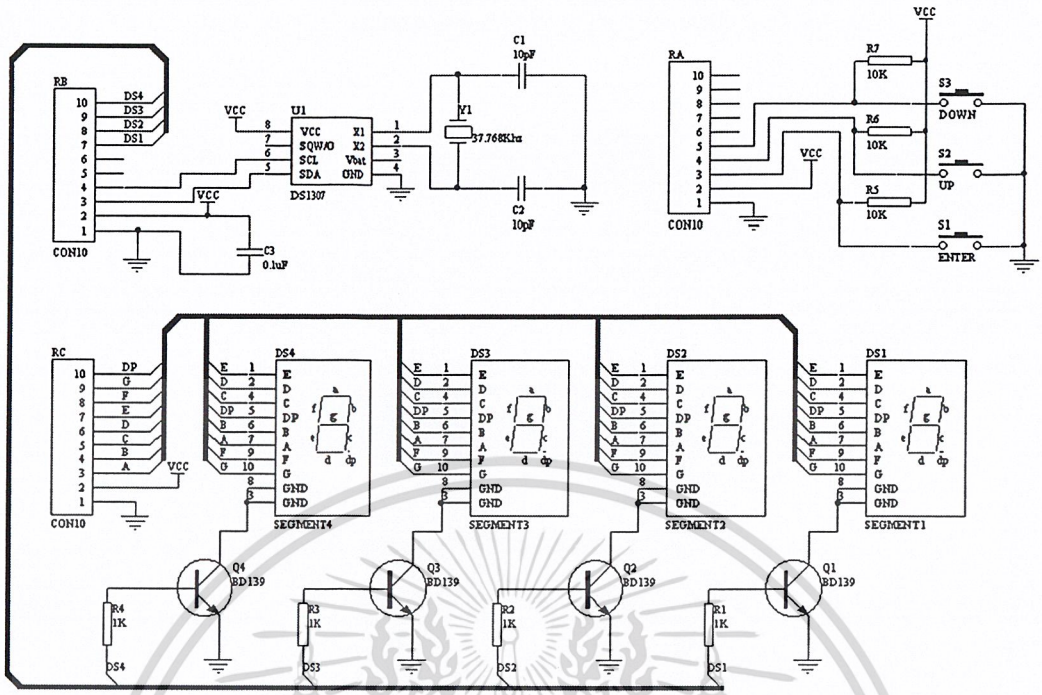
รูปที่ 3.5 วงจรของการเชื่อมต่อแบบพอร์ตอนุกรม

RS-232 Interface ใช้ไอซี MAX232 ในการแปลงสัญญาณจากไมโครคอนโทรลเลอร์ไปสู่มาตรฐาน RS-232 จะติดต่อกับพอร์ต RC6 (TX) และ RC7 (RX) ของไมโครคอนโทรลเลอร์ ด้านจุดเชื่อมต่อจะผ่านคอนเนคเตอร์ DB-9 ชนิด Male มีลักษณะเหมือนกับจุดเชื่อมต่อกับคอมพิวเตอร์ จึงสามารถใช้สายสัญญาณจากคอมพิวเตอร์ในการเชื่อมต่อได้ การใช้งาน RS-232 จะต้องเลือกจั๊มเปอร์ชื่อ RS-232 ไปที่ ON ก่อน โดยจั๊มเปอร์ตัวนี้จะใช้ตัดต่อสัญญาณระหว่างตัวไอซี MAX232 กับพอร์ต RC7 ของไมโครคอนโทรลเลอร์ ดังรูปที่ 3.5

3.3.4 การเชื่อมต่อแบบไอสแควร์ซี (I^2C Interface)

บัส I^2C ประกอบด้วยสาย 2 เส้น คือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัส การทำงานของบัส I^2C อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง อุปกรณ์ที่เป็นผู้รับข้อมูล เรียกว่า ตัวรับ อุปกรณ์บนบัสสามารถเป็นได้ทั้งตัวรับและตัวส่งหรือเป็นตัวรับอย่างเดียว แต่ไม่สามารถเป็นตัวส่งเพียงอย่างเดียวได้ อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I^2C

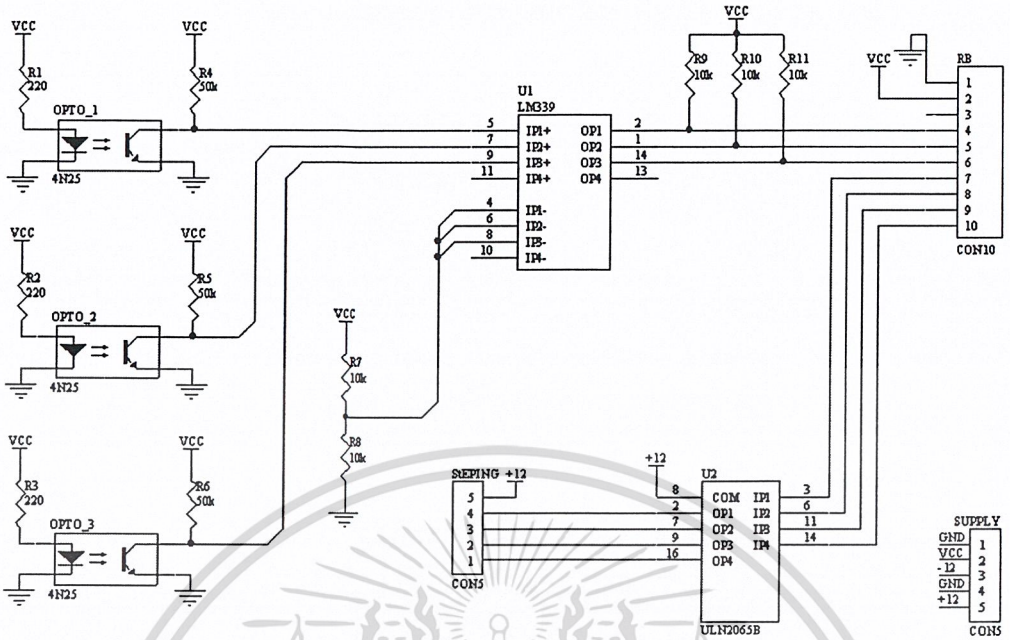
ไอซี DS1307 เชื่อมต่อแบบบัส I^2C โดยจะทำงานเป็นสลาฟเสมอการใช้งานจะติดต่อแบบ I^2C DS1307 ทำงานที่ขา SQW/OUT จะมีสัญญาณพัลส์สี่เหลี่ยมออกมา ในกรณีที่มีการอีนาเบิลวงจรกำเนิดสัญญาณพัลส์ที่รีจิสเตอร์ควบคุม ความถี่ที่ใช้สามารถเลือกได้ 4 ค่าคือ 1Hz, 4.096 kHz, 8.192 kHz และ 32 kHzซึ่งจะทำการเก็บค่าของเวลาไว้ในหน่วยความจำ ซึ่งมีขนาดรวม 64 ไบต์ แต่จะใช้เก็บข้อมูลเวลาอีก 8 ไบต์และสำหรับผู้ใช้อีก 56 ไบต์ การใช้งาน DS1307 จะใช้งานเฉพาะโหมดอ่านข้อมูลเท่านั้นดังแสดงวงจรในรูปที่ 3.6 การใช้งาน DS1307 เป็นไอซีระบบฐานเวลาโดยใช้วงจรออสซิสเลเตอร์ที่ใช้คริสตัล 32.768 kHz ร่วมกันกับคาปาซิสเตอร์ 10 PF 2 ตัว ต่อเข้ากับขา X1 และ X2



รูปที่ 3.6 วงจรของการเชื่อมต่อแบบไอสแควร์ซี

3.2.5 การรับอินพุตแบบเครื่องมือวัด (Input/Instrument)

ประกอบไปด้วยเครื่องมือวัด สำหรับใช้ประกอบการทดลองใบงาน โดยโมดูลนี้จะใช้ไมโครคอนโทรลเลอร์ PIC16F877 เป็นตัวประมวลผลและควบคุมการทำงานประกอบไปด้วยวงจรภาคคีย์สวิตช์ (Switch Key) ที่ใช้เลือกชนิดของเครื่องมือวัด ตัวประมวลผล และภาคแสดงผลสำหรับแสดงสถานะของเครื่องมือวัดว่าเป็นเครื่องมือวัดชนิดใด ซึ่งจะบอกเป็นหน่วยของการวัดของเครื่องมือวัดชนิดนั้น



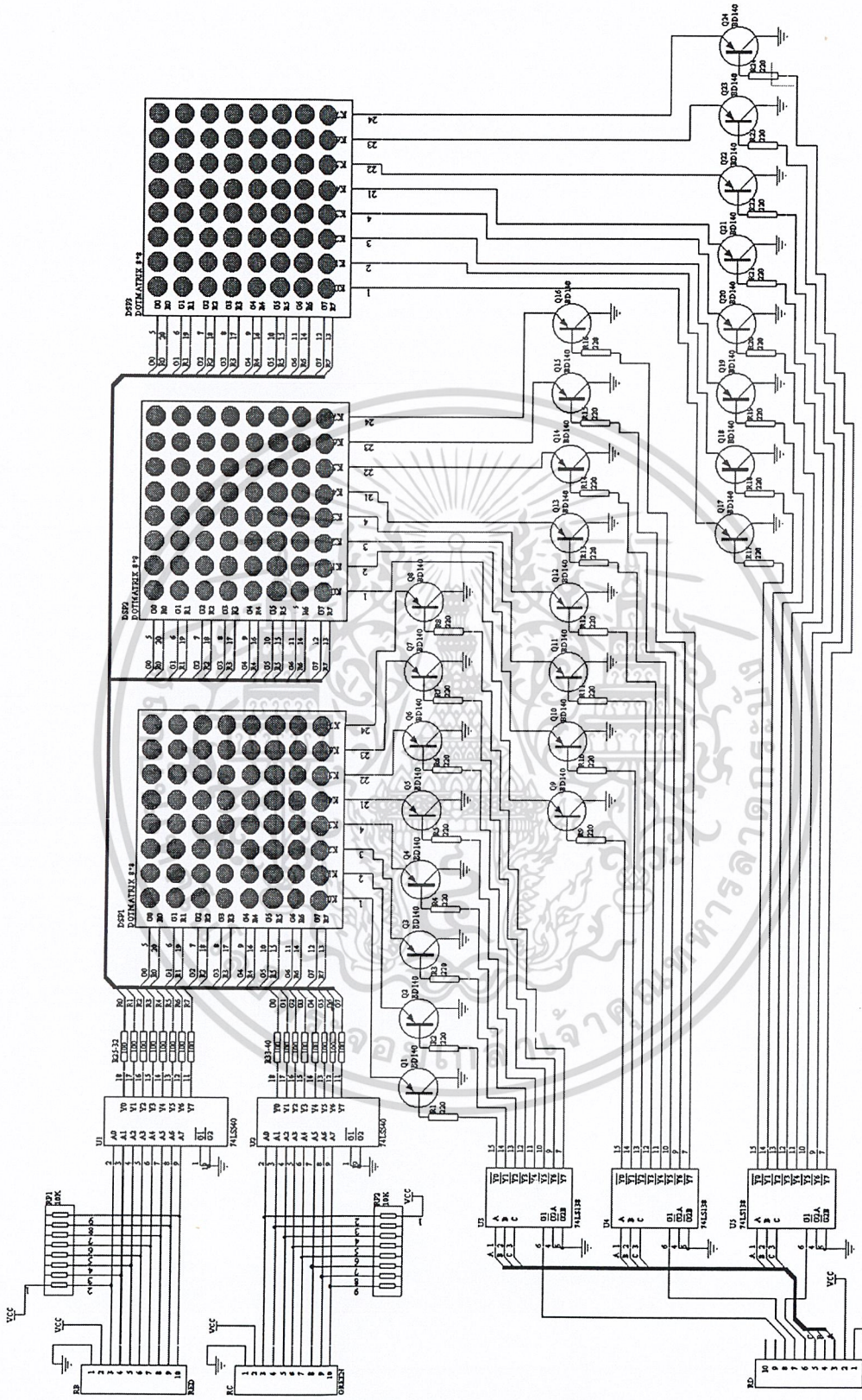
รูปที่ 3.7 วงจรของการรับอินพุตแบบเครื่องมือวัด

การใช้งานที่ขาอินพุตเป็นโฟโต้และส่งสถานะที่วัดได้มายัง LM339 ซึ่งเป็นออปแอมป์ทำหน้าที่เปรียบเทียบสัญญาณที่ได้จากโฟโต้ว่าเป็น “1” หรือ “0” และส่งสถานะไปยัง PIC16F877 และ PIC16F877 จะส่งข้อมูลมายัง ULN2065B เป็นตัวไดร์สแต็ปปิ้งมอเตอร์ว่าจะให้มอเตอร์หมุนหรือหยุด

3.2.6 การแสดงผลด้วยดอทเมตริกซ์ 24x8 (Output Dotmatrix 24x8)

การแสดงผลแอลอีดีดอทเมตริกซ์ขนาด 24x8 จุด (24 แถว 8 คอลัมน์) ทางด้านการเลือกหลักจะส่งข้อมูลผ่านไอซีดีโค้ดเดอร์เบอร์ 74LS138 ขนาดเข้า 3 ออก 8 จำนวน 3 ตัวจะใช้เลือกหลักของแอลอีดีที่ต้องการให้ติด ทรานซิสเตอร์ชนิด เอ็นพีเอ็น (PNP) ทำหน้าที่ขยายกระแสให้แอลอีดี 1 ตัวต่อ 1 คอลัมน์และยังทำหน้าที่กลับสภาวะของสัญญาณด้วยเพราะแอลอีดีเป็นชนิดคอมมอนแคโถด แต่เอาต์พุตของไอซีดีโค้ดเดอร์แอคตีฟที่ลอจิก “0” ดังนั้นกระแสที่ไหลผ่านทรานซิสเตอร์สูงสุดนั้นจะเท่ากับ กระแสที่ให้แอลอีดีจำนวน 16 ดวงติดพร้อมกัน เนื่องจากแอลอีดีในหนึ่งคอลัมน์ประกอบไปด้วยแอลอีดีสีแดงจำนวน 8 ดวง และ สีเขียวอีก 8 ดวง ที่สามารถสั่งให้ติดได้พร้อมกัน จึงใช้ทรานซิสเตอร์เบอร์ BD140 ให้สามารถจ่ายกระแสได้สูงอย่างปลอดภัย

การใช้งานการแสดงผลแอลอีดีดอทเมตริกซ์จะต้องใช้สัญญาณข้อมูลอย่างน้อย 12 บิต เพื่อให้แอลอีดีติด 1 สีและ 20 บิต สำหรับให้แอลอีดีติด 2 สี หรือ 3 สี (สองสีติดพร้อมกัน) ตัวการแสดงผลมีลักษณะดังรูปที่ 3.8

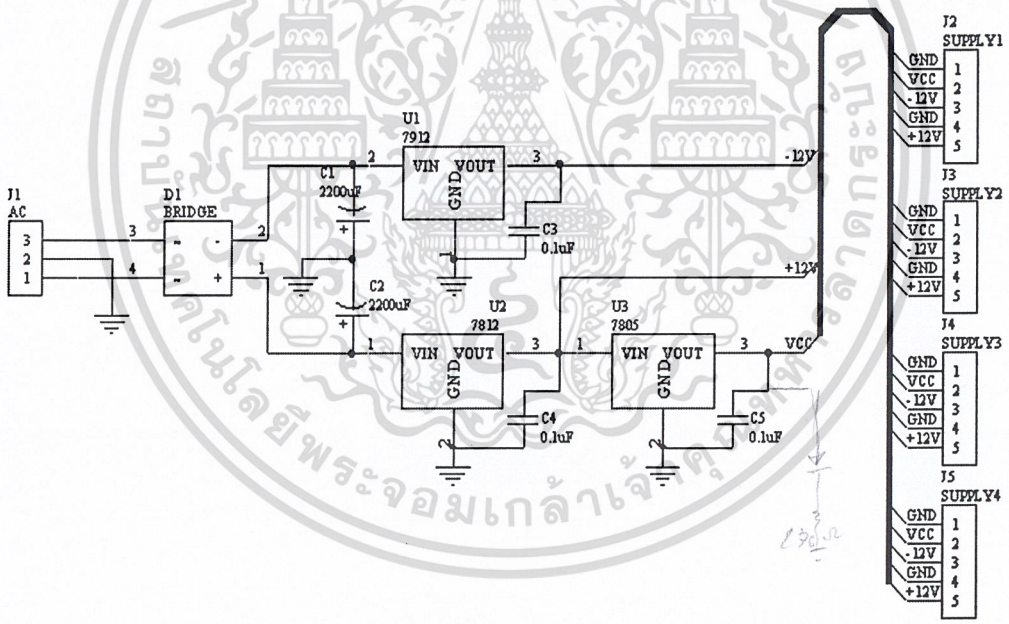


รูปที่ 3.8 วงจรของการแสดงผลด้วยดอทเมตริกซ์ 24x8

3.2.8 โมดูลภาคจ่ายไฟ (Power Supply Module)

โมดูลภาคจ่ายไฟเป็นตัวจ่ายแรงดันในกับโมดูลทุกๆ โมดูล มีแรงดันให้ใช้งานคือ 5 โวลต์ สำหรับจ่ายให้อุปกรณ์ดิจิตอลที่เป็น TTL แรงดัน ±12 โวลต์ สำหรับอุปกรณ์ที่ต้องใช้แรงดันสูง เช่น ออปแอมป์ และมอเตอร์ ภาคจ่ายไฟ

วงจรจะใช้หม้อแปลงชนิด 3 เฟสจากภายนอก ในการจ่ายไฟให้โมดูล ซึ่งแรงดันประมาณ 15-0-15 VAC จากนั้นก็จะนำมาผ่านการเรียงกระแสเป็นกระแสตรง และใช้เร็กกูเรเตอร์ (Regulator) เป็นตัวควบคุมระดับแรงดันคงที่ จากนั้นจะถูกต่อไปใช้งานด้วยคอนเนคเตอร์ขนาด 5 ขา มีแรงดันแต่ละขาคือ 0, 5, -12, 0, +12 โวลต์ และในวงจรภาคจ่ายไฟก็ยังสามารถใช้อะแดปเตอร์ (Adapter) ขนาด 12 โวลต์ ขึ้นไป เป็นตัวจ่ายแทนหม้อแปลงก็ได้ แต่แรงดันที่ได้จะไม่มีแรงดัน –12 โวลต์ ซึ่งจะทำให้โมดูลแปลงระหว่างสัญญาณแอนะลอกและสัญญาณดิจิตอลทำงานไม่ได้ ส่วนโมดูลอื่นๆ ก็ยังทำงานได้ตามปกติ



รูปที่ 3.9 วงจรของโมดูลภาคจ่ายไฟ

3.3 การออกแบบทางด้านซอฟต์แวร์

3.3.1 การออกแบบโปรแกรม PCW

โปรแกรม PCW จะเรียกใช้งานโปรแกรมชื่อ IC-Prog.Exe ในลักษณะ Command Line เพื่อโหลดข้อมูลไปยัง PIC16F87X โดยส่ง Part ของไฟล์นามสกุล . Hex ที่ต้องการ และเบอร์ของไมโครคอนโทรลเลอร์ PIC16F87X ที่ใช้ทดลองให้โปรแกรม IC-Prog.Exe เป็นตัวจัดการโหลดข้อมูลลงบนไมโครคอนโทรลเลอร์ PIC16F87X ดังแสดงขั้นตอนตามแผนผังการทำงานในรูปที่ 3.15

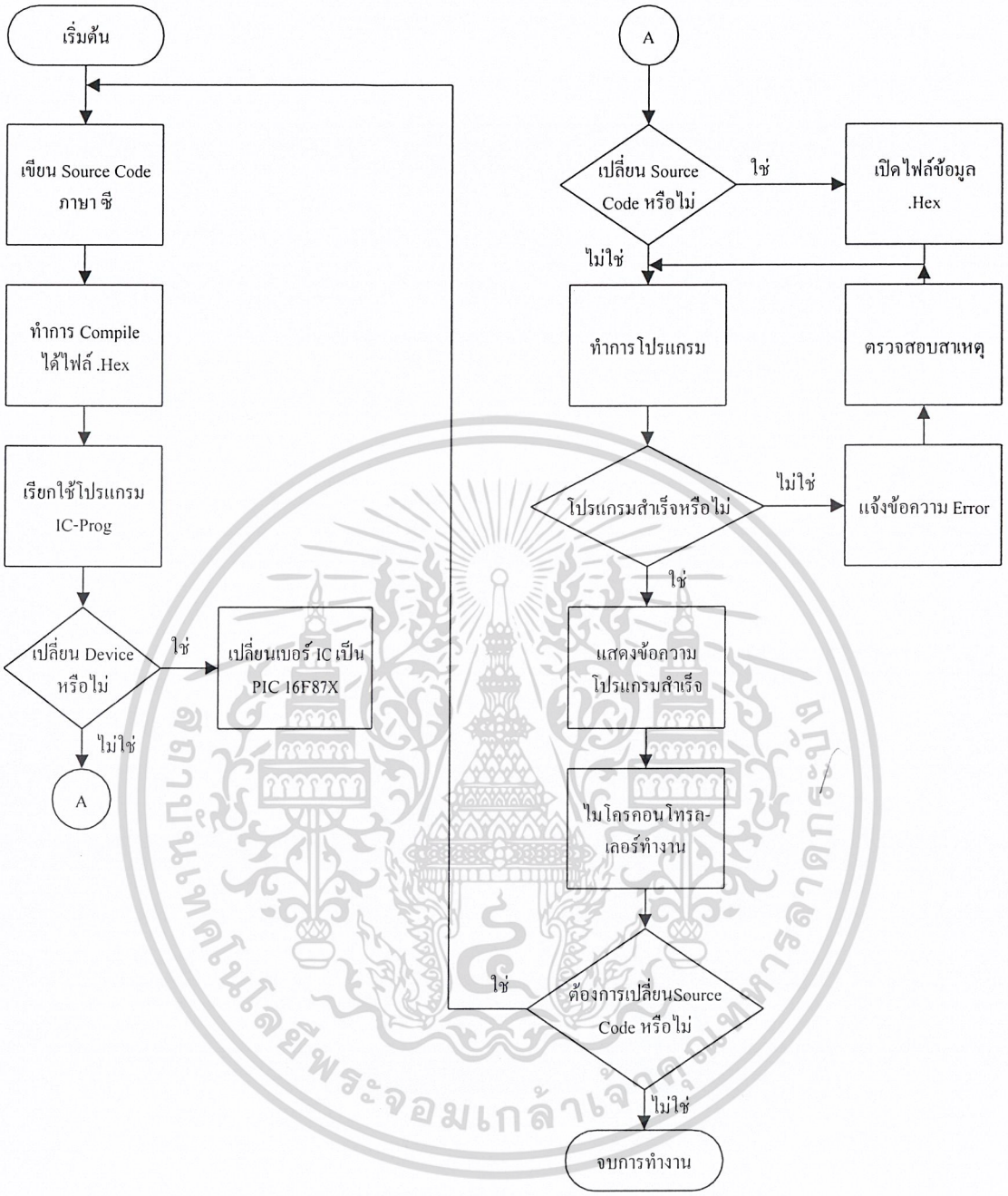
จากรูปที่ 3.9 แสดงถึงขั้นตอนในการปฏิบัติการทดลองไมโครคอนโทรลเลอร์ PIC16F87X โดยทำการเชื่อมต่อโมดูลต่างๆ ตามที่ต้องการทดลองแล้ว ก็ทำการเขียนซอสโค้ด (Source Code) โปรแกรมการทำงานให้แก่ PIC16F87X โดยจะใช้โปรแกรม PCW ในการเขียนจะใช้ภาษาซี (C) จากนั้นก็ทำการคอมไพล์ (Compliler) เป็นไฟล์ .Hex แล้วก็ทำการโหลดข้อมูลจาก .Hex ไฟล์ลงสู่ PIC16F87X ด้วยการป้อนชื่อของไฟล์ .Hex พร้อมเส้นทางของชื่อ ลงในโปรแกรม PCW ซึ่งโปรแกรมจะนำข้อมูลในไฟล์นี้โหลดลงสู่ PIC16F87X โดยผ่านโปรแกรม IC-Prog อีกทีก็เสร็จกระบวนการโปรแกรมต่อไปเมื่อต้องการแก้ไขคำสั่งในไฟล์สามารถแก้ไขแล้วทำการคอมไพล์เป็นไฟล์ .Hex ชื่อเดิม จากนั้นก็สั่งโหลดข้อมูลลงสู่ PIC16F87X ได้ทันที

3.3.2 การใช้ฟังก์ชั่นเรียกใช้งานโปรแกรม IC-Prog.exe

อาร์กิวเมนต์ที่จะส่งไปพร้อมกับคำสั่งเรียกใช้โปรแกรม IC-Prog.Exe นั้นจะนำมาจากค่าคุณสมบัติ (Properties) ในคอมโพแนนต์ (Components) ต่างๆ ที่อยู่บนฟอร์ม เช่น Combo Box ชื่อ Select สำหรับเลือกเบอร์ของ PIC ส่วนปุ่มชื่อ Browse Hex File... สำหรับใส่ชื่อไฟล์ .Hex พร้อม Part ซึ่งค่าต่างๆ เหล่านี้จะถูกนำมาใช้เป็นอาร์กิวเมนต์สำหรับเรียกใช้โปรแกรม IC-Prog.Exe

ตัวอย่างการเรียกใช้ IC-Prog.exe

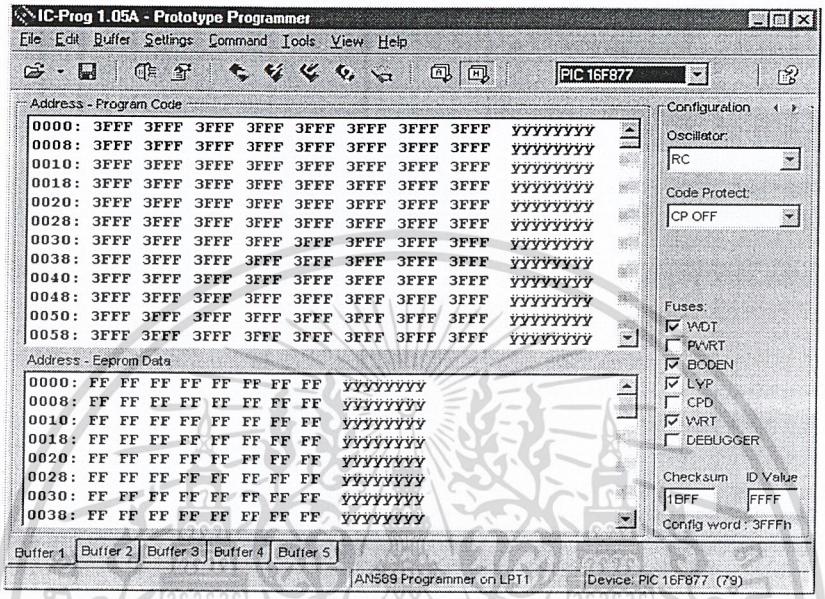
หมายถึง การเรียกโปรแกรมโค้ดพร้อมเปรียบเทียบโค้ดลงบน PIC16F87X ผ่าน Parallel Port1 จาก C : \ File.Hex



รูปที่ 3.10 ขั้นตอนการโหลดข้อมูลโปรแกรมลงบนหน่วยความจำโปรแกรม

3.3.3 การเรียกใช้โปรแกรม IC Prog.exe

โปรแกรมสำหรับดาวน์โหลดข้อมูลและโปรแกรมลงบนไมโครคอนโทรลเลอร์จากคอมพิวเตอร์ผ่านพอร์ตขนาน ทำงานบนระบบปฏิบัติการวินโดวส์



รูปที่ 3.11 โปรแกรม IC Prog.exe

3.3.4 การใช้งานโปรแกรม IC Prog.exe

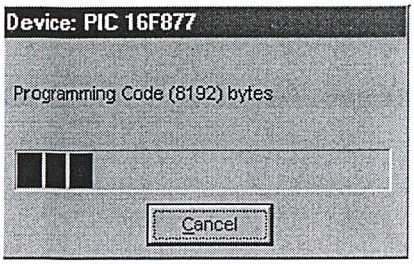
1. เลือกเบอร์ของไมโครคอนโทรลเลอร์ที่ใช้งานจาก 

2. เปิดไฟล์ .Hex จะปรากฏหน้าต่างเปิดไฟล์ ให้เลือกไฟล์ต้องการโปรแกรม

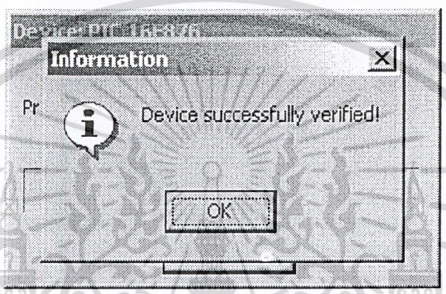
3. แสดงโปรแกรมได้ทันทีโดยคลิกปุ่ม 

จะเริ่มทำการโปรแกรมโดยจะปรากฏหน้าต่างแสดงการโปรแกรมดังรูปที่ 3.11

4.ขณะโปรแกรมข้อมูลลงบนไมโครคอนโทรลเลอร์นี้ จะมีแอลซีดีแสดงสถานะการโปรแกรม บนตัวโมดูลซึ่งในขณะนี้ห้ามกดปุ่ม RESET เป็นอันขาด เพราะจะทำให้เกิดความผิดพลาดของข้อมูลขึ้น



รูปที่ 3.12 หน้าต่างแสดงขณะทำการโปรแกรม

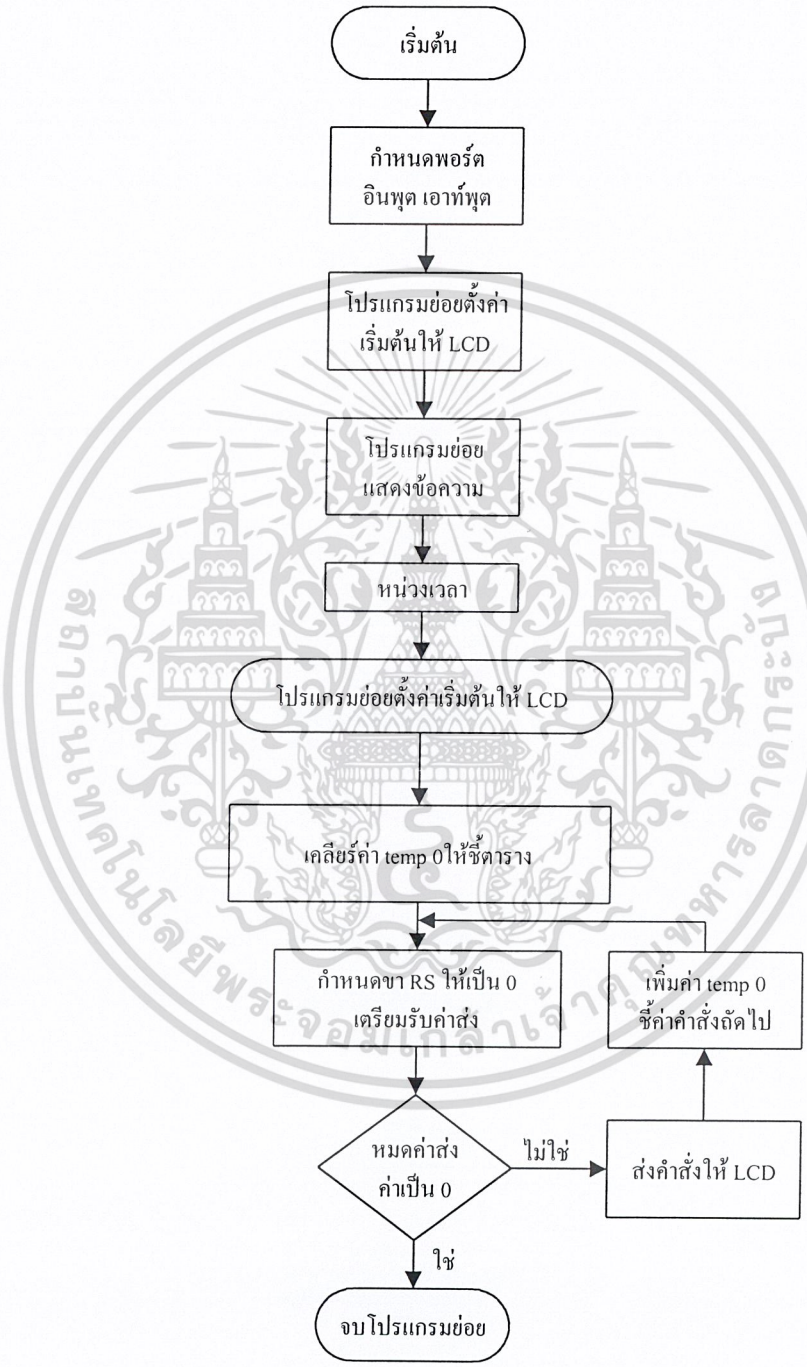


รูปที่ 3.13 เมื่อโปรแกรมเสร็จสมบูรณ์

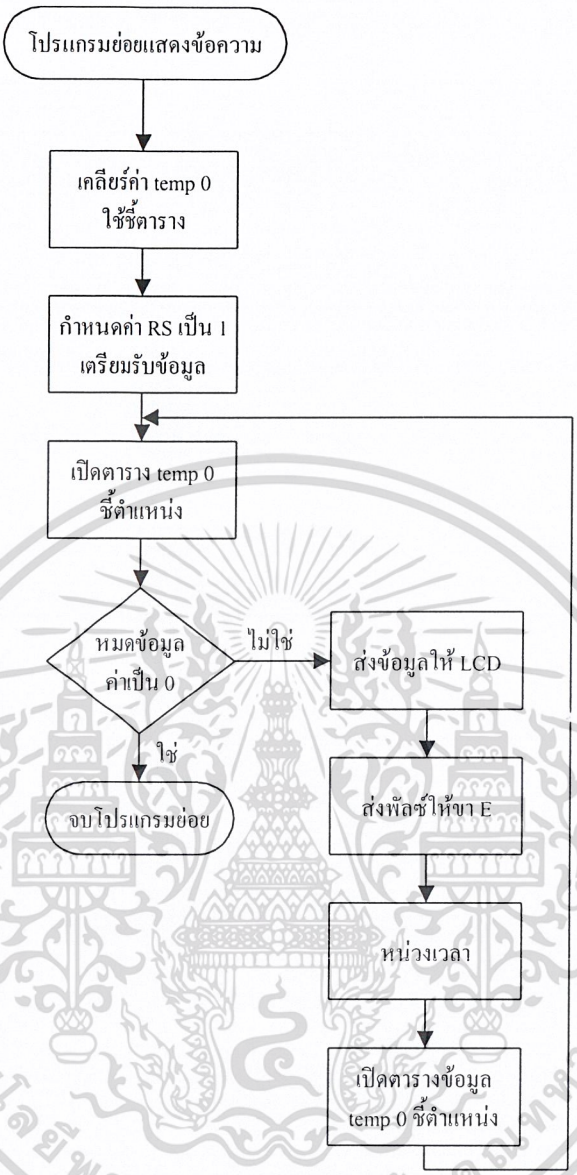
เมื่อโปรแกรมเสร็จ จะแสดงหน้าต่างดังรูปที่ให้คลิกปุ่ม OK จากนั้นทำการรัน CPU โดยการคลิกที่ปุ่ม RUN ไมโครคอนโทรลเลอร์ก็จะเริ่มทำงาน

3.4 การออกแบบโปรแกรมทดลองการทำงานของโมดูล

3.4.1 ผังการทำงานของโปรแกรมทดลองการแสดงผลด้วยจอแบบผลึกเหลว

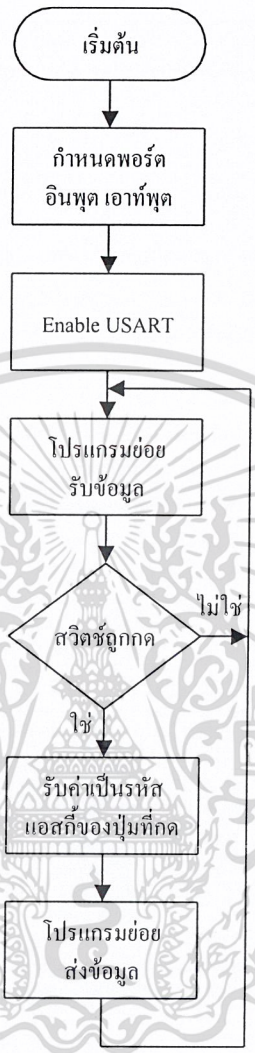


รูปที่ 3.14 ผังการทำงานของโปรแกรมทดลองการแสดงผลด้วยจอแบบผลึกเหลว

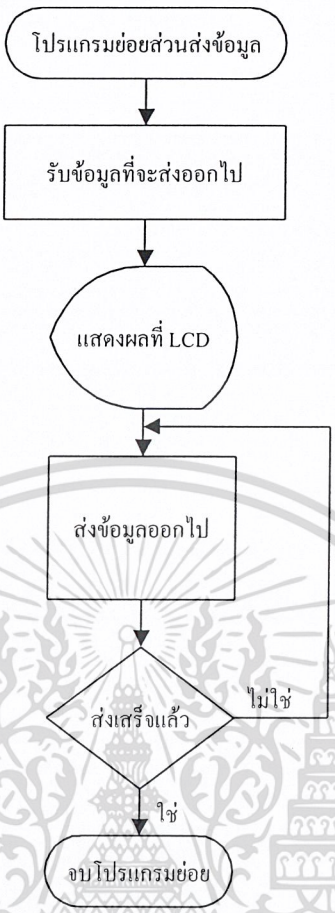


รูปที่ 3.14 (ต่อ) ผังการทำงานของโปรแกรมทดลองการแสดงผลด้วยจอแบบผลึกเหลว

3.4.2 ผังการทำงานของโปรแกรมทดลองการเชื่อมต่อแบบพอร์ตอนุกรม

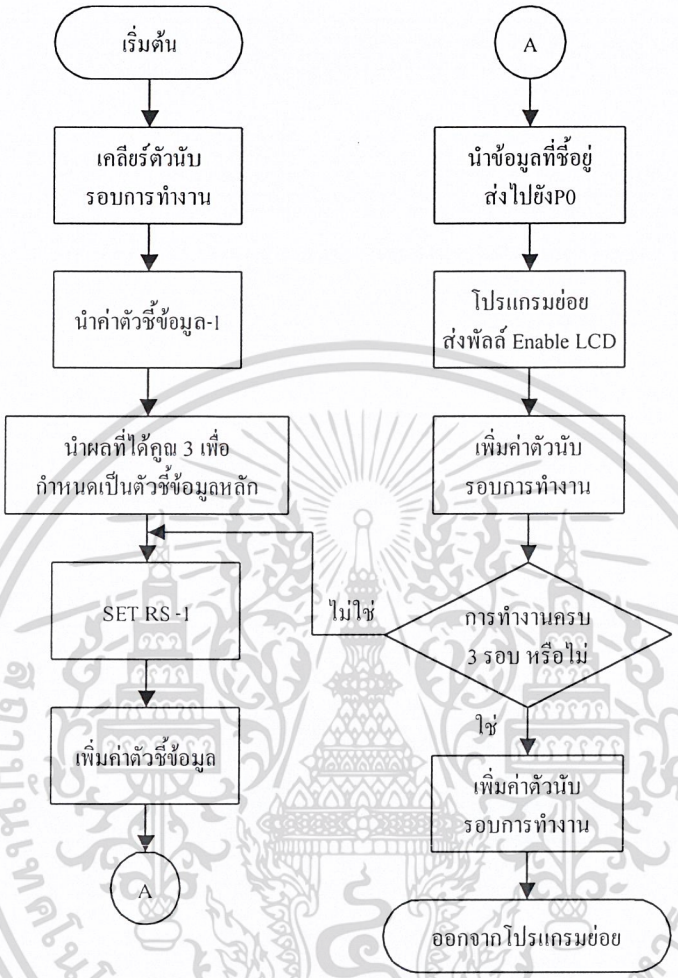


รูปที่ 3.15 ผังการทำงานของโปรแกรมทดลองการเชื่อมต่อแบบพอร์ตอนุกรม RS-232



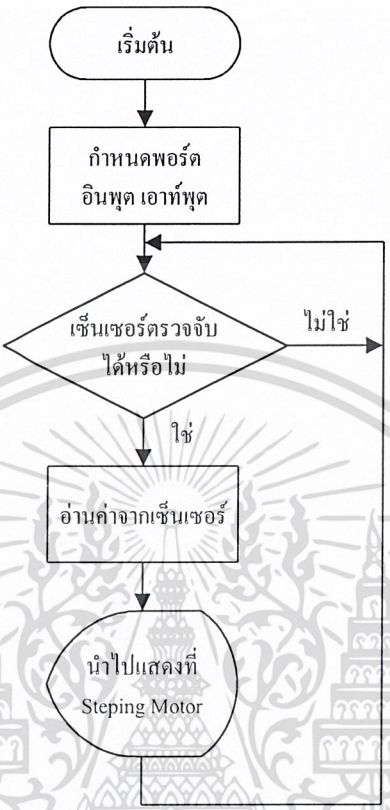
รูปที่ 3.15 (ต่อ) ผังการทำงานของโปรแกรมทดลองการเชื่อมต่อแบบพอร์ตอนุกรม RS-232

3.4.3 ผังการทำงานของโปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี



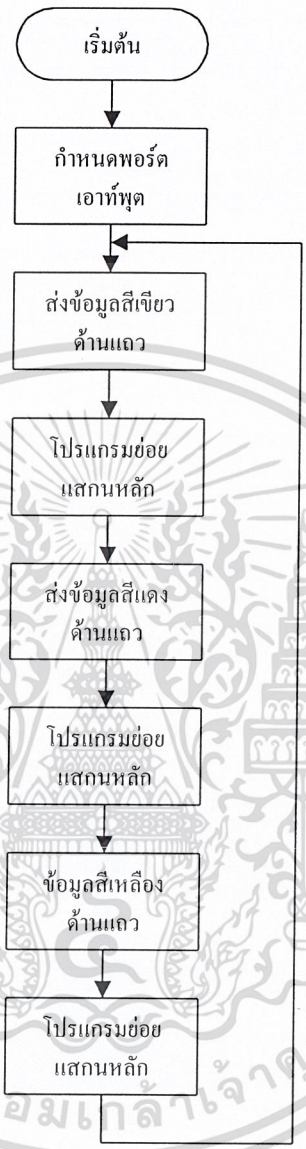
รูปที่ 3.16 ผังการทำงานของโปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี

3.4.4 ผังการทำงานของโปรแกรมทดลองการรับอินพุตแบบเครื่องมือวัด

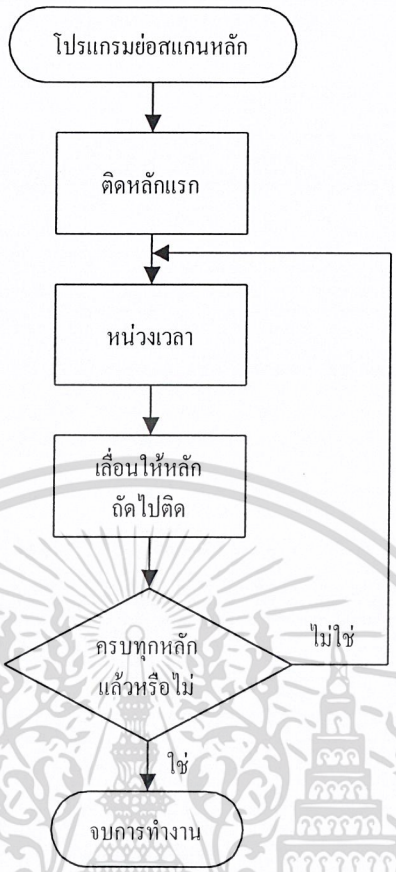


รูปที่ 3.17 ผังการทำงานของโปรแกรมทดลองการรับอินพุตแบบเครื่องมือวัด

3.4.5 ผังการทำงานของโปรแกรมทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8



รูปที่ 3.18 ผังการทำงานของโปรแกรมทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8



รูปที่ 3.18 (ต่อ) ผังการทำงานของโปรแกรมทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลองใช้งานโมดูล PIC-ICSP

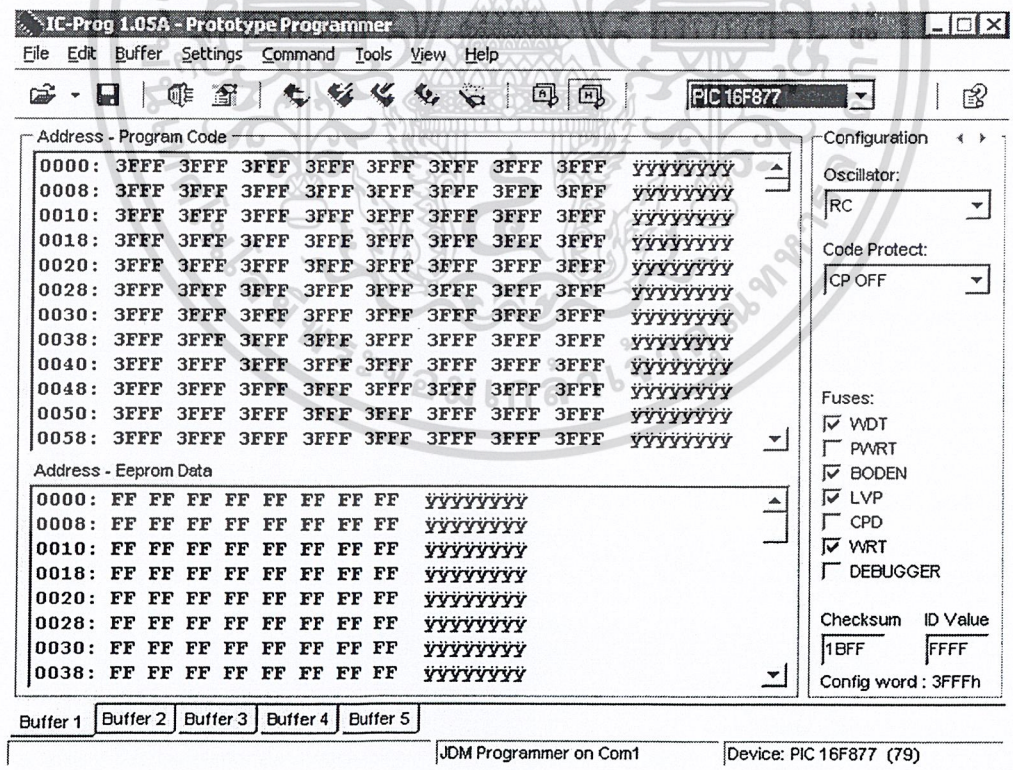
โมดูล PIC-ICSP เป็นตัวควบคุมการทำงานของไมโครคอนโทรลเลอร์ PIC16F87X รวมทั้งการดาวน์โหลดโปรแกรมจากคอมพิวเตอร์ลงสู่ไมโครคอนโทรลเลอร์ การใช้งานโมดูลจะใช้ร่วมกับโปรแกรม PCW ที่ใช้ในการเขียนไฟล์ขึ้นมา ซึ่งได้ทำการทดลองการทำงานดังนี้

4.1.1 ขั้นตอนการทดลอง

1. ทำการเชื่อมต่อตัวโมดูลเข้ากับคอมพิวเตอร์ โดยต่อเข้ากับพอร์ตขนาน

2. จ่ายแรงดันให้แก่โมดูล PIC-ICSP จากโมดูลภาคจ่ายไฟ

3. ทดลองการดาวน์โหลดข้อมูลโปรแกรมลงบนไมโครคอนโทรลเลอร์ โปรแกรมที่ใช้ในการทดลองเป็นโปรแกรม ตั้งค่าเริ่มต้นให้แก่พอร์ต เขียนขึ้นจาก PCW ดังนี้ แสดงดังรูปที่ 4.1



รูปที่ 4.1 โปรแกรมที่ใช้ทดลองดาวน์โหลดข้อมูลของโมดูล PIC-ICSP

ถ้าการดาวน์โหลดไม่มีข้อผิดพลาด โปรแกรมก็จะไม่แสดงข้อความผิดพลาด และจะแสดงข้อความการดาวน์โหลดสมบูรณ์ ระหว่างดาวน์โหลด LED สีเขียวที่ชื่อ PROG ที่อยู่บนโมดูลก็จะสว่างขึ้น

4. อ่านข้อมูลกลับจากไมโครคอนโทรลเลอร์ เพื่อตรวจสอบโค้ดโปรแกรมที่ได้โปรแกรมลงไป ว่าถูกต้องหรือไม่

5. ทดลองควบคุมการทำงานของไมโครคอนโทรลเลอร์ จากโปรแกรมโดยการกดปุ่ม RUN RESET เพื่อดูการติดดับของ LED สีเหลืองและวัดแรงดันตามจุดต่างๆ โดยเฉพาะที่ขา MCLR ของไมโครคอนโทรลเลอร์

6. ในขณะโปรแกรมนำมิเตอร์วัดที่ขา MCLR ของตัวไมโครคอนโทรลเลอร์ จะต้องมีไฟ 12 V

4.1.2 ผลการทดลอง

การทดลองดาวน์โหลดข้อมูลลงบนไมโครคอนโทรลเลอร์ ให้ผลถูกต้อง LED แสดงการโปรแกรมติด เมื่อโปรแกรมเสร็จ LED ก็ติดแทนเพื่อรอรับคำสั่งการทำงาน เมื่อการอ่านข้อมูลเปรียบเทียบกับข้อมูลเดิมถูกต้องตรงกัน ซึ่งแสดงว่า เครื่องโปรแกรมไมโครคอนโทรลเลอร์ PIC16F87X ทำงานแล้ว จากนั้นลองสั่งให้ไมโครคอนโทรลเลอร์ทำงานโดยการกดสวิตช์ RESET เพื่อทำการ RUN เราสามารถรู้ว่าการควบคุมการทำงานถูกต้องหรือไม่ดูได้จากแรงดันที่ขา MCLR เมื่อสั่ง RUN วัดแรงดันได้ 5V เมื่อสั่ง RESET วัดแรงดันได้ 0V แสดงว่า การควบคุมการทำงานถูกต้อง

จากผลการทดลองสามารถบอกได้ว่าโมดูล PIC-ICSP ทำงานได้แล้วประมาณ 80 เปอร์เซ็นต์ เนื่องจากยังไม่ได้ต่อโมดูลอื่นๆ ให้แสดงผลการทำงานจริง ดังนั้นจึงทำการทดลองขั้นต่อไป คือ การเชื่อมต่อโมดูล PIC-ICSP นี้เข้ากับโมดูลแสดงผล แล้วทดลองใช้งาน

4.2 การทดลองใช้งานการแสดงผลด้วยจอแบบผลึกเหลว

การทดลองการแสดงผลด้วยจอแบบผลึกเหลวขนาด 16 อักษร คูณ 2 บรรทัด ทำโดยการส่งข้อมูลที่เป็นข้อความให้ตัวการแสดงผลมีขั้นตอนดังนี้

4.2.1 ขั้นตอนการทดลอง

1. เชื่อมต่อโมดูลแอลซีดีดิสเพลย์เข้ากับโมดูลหลัก โดยใช้การเชื่อมต่อแบบ 8 บิต
2. เชื่อมต่อโมดูลหลักเข้ากับเพาเวอร์ซัพพลาย
3. จากผังการทำงานโปรแกรมที่ได้ออกแบบไว้ ทำการเขียนโปรแกรมทดลองการใช้งานโมดูลแบบ 8 บิต

4. โปรแกรมลงไมโครคอนโทรลเลอร์แล้วสั่งทำงาน การแสดงผลจะต้องแสดงข้อความ เช่น " PIC16F87X" โปรแกรมแสดงดังรูปที่ 4.2

```
#include "LCD.h"
#include "amPicLib.h"

  // lcd
#bit lcd_e  = port_B.0
#bit lcd_rs = port_B.1
#bit lcd_rw = port_B.2
#define lcd_data    port_C
#define lcd_direct(x) set_tris_c(x)
enum {rs_cmd, rs_data};
#define ON_CURSOR      0x0e
#define OFF_CURSOR     0x0c
#define CLEAR          0x01
int lcd_read_byte() {
int val;
  lcd_direct(0xff);
  lcd_rw = 1;
  delay_cycles(1);
  lcd_e = 1;
  delay_cycles(1);
  val = lcd_data;
  lcd_e = 0;
  lcd_direct(0x00);
  return val;
}

void lcd_send_byte(int rs, val) {
  lcd_rs = 0;
  while ( bit_test(lcd_read_byte(), 7) ) ;
  lcd_rs = rs;
  delay_cycles(1);
  lcd_rw = 0;
  delay_cycles(1);
  lcd_e = 0;
  lcd_data = val;
  delay_cycles(1);
  lcd_e = 1;
  delay_cycles(1);
  lcd_e = 0;
}
void lcd_init() {
  lcd_rs = 0;
  delay_ms(15);
  lcd_send_byte(rs_cmd, 0x38);
  delay_ms(1);
  lcd_send_byte(rs_cmd, 0x0c);
```

รูปที่ 4.2 โปรแกรมทดลองการแสดงผลด้วยจอแบบผลึกเหลว

```
lcd_send_byte(rs_cmd, 0x01);
  lcd_send_byte(rs_cmd, 0x06);
}

void lcd_gotoxy(int x, y) {
int addr;
  if (y!=1) addr = 0x40;
  else addr = 0;
  addr += x-1;
  lcd_send_byte(rs_cmd, 0x80|addr);
}

void lcd_putc(unsigned char ch) {
  lcd_send_byte(rs_data, ch);
}

void main() {
  set_tris_b(0x00);
  set_tris_c(0x00);
  port_b = 0;
  port_C = 0;
  lcd_init();
  lcd_putc("PIC16F877");
  lcd_gotoxy(1, 2);
lcd_putc("Microcontroller");
while (true);
}
```

รูปที่ 4.2 (ต่อ) โปรแกรมทดลองการแสดงผลด้วยจอแบบผลึกเหลว

4.2.2 ผลการทดลอง

เมื่อสั่งทำงาน การแสดงผลแสดงข้อความตามโปรแกรมได้ถูกต้อง จะปรากฏตัวอักษรคำว่า PIC16F877บนหน้าจอแอลซีดี แล้วทดลองปรับความสว่างของส่วนแสดงผลที่ VR โดยเมื่อปรับ VR ไปทางด้านซ้ายจะทำให้ ความคมชัดหน้าจอแอลซีดีน้อยลง และเมื่อปรับไปทางขวาจะทำให้แอลซีดีคมชัดขึ้น แสดงว่าการแสดงผลแอลซีดีดิสเพลย์สามารถใช้งานได้

4.3 การทดลองใช้งานการเชื่อมต่อแบบพอร์ตอนุกรม

ในโมดูลนี้เป็นวงจรสื่อสารข้อมูลอนุกรมแบบ RS-232 การทดลองสื่อสารข้อมูลอนุกรมแบบ RS-232 ทดลองโดยการเชื่อมต่อรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ โดยแสดงผลข้อมูลเป็นตัวอักษรบนโปรแกรม Hyper Terminal

4.3.1 ขั้นตอนการทดลอง

1. เชื่อมต่อแบบพอร์ตอนุกรมแบบ RS-232 เข้ากับโมดูลหลัก และพอร์ตอนุกรมของคอมพิวเตอร์

2. จากผังการทำงานที่ได้ออกแบบไว้ เขียนโปรแกรมทดลองการทำงาน ส่งข้อมูลระหว่างคอมพิวเตอร์กับไมโครคอนโทรลเลอร์

3. โปรแกรมลงบนไมโครคอนโทรลเลอร์ แล้วสั่งทำงาน

4. เปิดโปรแกรม Hyper Terminal จากคอมพิวเตอร์ เพื่อใช้ในการสื่อสารข้อมูลอนุกรมแบบ RS-232 กับอุปกรณ์ต่อพ่วง

5. ตั้งค่า Bits per second ไว้ที่ 9600 Parity None และ Stop bits 1

6. สั่งไมโครคอนโทรลเลอร์ทำงาน โดยทดลองกดคีย์บอร์ดคอมพิวเตอร์ เพื่อส่งข้อมูลไปแสดงผลยังไมโครคอนโทรลเลอร์ และสังเกตุการเปลี่ยนแปลงของโปรแกรม Hyper Terminal

```
include "Serial.h"
#int_RDA
RDA_isr() {
char c;
  c = getc();
  printf("You press key : %c", c);
  putc(13);
}

void main() {
  set_tris_a(0xfe);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  enable_interrupts(INT_RDA);
  enable_interrupts(global);


  delay_ms(500);
  printf("Welcom to PIC16F877");
  putc(13);
  printf("Serial Module");
  while (1);
}
```

รูปที่ 4.3 โปรแกรมทดลองการเชื่อมต่อแบบพอร์ตอนุกรม

4.3.2 ผลการทดลอง

เมื่อกดสวิตช์ RESET เพื่อทำการ RUN จะปรากฏข้อความ PIC16F877 บนหน้าจอของโปรแกรม Hyper Terminal แล้วเมื่อทำการกดคีย์บอร์ดของคอมพิวเตอร์ตัวอักษรที่กดก็จะปรากฏบนหน้าจอของโปรแกรม Hyper Terminal

4.4 การเชื่อมต่อแบบไอสแควร์ซี

การทดลองเชื่อมต่ออุปกรณ์ระบบบัส I^2C อุปกรณ์ RTC และ EEPROM บนโมดูล จะใช้การเชื่อมต่อแบบมาตราฐาน I^2C รับส่งข้อมูล ซึ่งการทำงานจะมีลักษณะคล้ายกัน ดังนั้นจะทดลองเชื่อมต่อกับอุปกรณ์ RTC เท่านั้น เพื่อทดลองการรับส่งข้อมูลแบบ I^2C นี้

ไอซีเบอร์ DS1307 เป็น RTC ที่มีการเชื่อมต่อใช้งานแบบ I^2C ส่งข้อมูลฐานเวลาไปให้แก่ไมโครคอนโทรลเลอร์ การทดลองใช้งานจะทำโดยอ่านค่าเวลาออกมาเป็นวินาที แล้วแสดงผลที่ 7 SEGMENT

4.4.1 ขั้นตอนการทดลอง

1. เชื่อมต่อ RTC เข้ากับโมดูลหลัก
2. จากผังการทำงานโปรแกรมที่ได้ออกแบบไว้ ทำการเขียนโปรแกรมทดลองการใช้งานอ่านข้อมูลวินาทีจาก RTC โปรแกรม
3. สั่งไมโครคอนโทรลเลอร์ทำงาน ถ้าใช้งานได้ SEGMENTในโมดูลแสดงผลเป็นค่าของชั่วโมงและนาที

```
#include "I2C.h"
#include "amPicLib.h"

//------------------------------------------------------- DS1307 RTC
#define i2c_SDA  PIN_B0
#define i2c_SCL  PIN_B1
#use i2c(master, sda=i2c_SDA, scl=i2c_SCL)

void init_i2c() {
  output_float(i2c_SCL);
  output_float(i2c_SDA);
}
void write_DS1307 (byte addr, data) {
  i2c_start();
  i2c_write(0xd0);
```

รูปที่ 4.4 โปรแกรมทดลองการติดต่อของการเชื่อมต่อแบบไอสแควร์ซี

```
 i2c_write(addr);
  i2c_write(data);
  i2c_stop();
  delay_ms(8);
}

int read_DS1307 (byte addr) {
int data;
  i2c_start();
  i2c_write(0xd0);
  i2c_write(addr);
  i2c_start();
  i2c_write(0xd1);
  data = i2c_read(0);
  i2c_stop();
  return(data);
}

  // Key ---------------------------------
enum {swEnter=1, swUp=2, swDown=4};
unsigned char keyNow=0, wKey=0;

  // Segment-----------------------------
int segData[5];
int segDigit=0;
int  const number[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f};  // ตัวเลข 0-9

//========================================== declaration
// system
int sysclk=0, sysclk2=0;

  // parameter
enum {modeTime, modeSet};
int NowMode;
int seg_rate=0;
short seg_blink;
int const maxTime[2] = {0x23, 0x59};
int const minTime[2] = {0x00, 0x00};
enum { adrSecond,  adrMinute,  adrHour,  adrDay,  adrDate,  adrMonth,
adrYear};
int buffTime[2], adrTime;

//========================================== Task

void SetMode(int mode) {
int i;
  NowMode = mode;
  switch (NowMode) {
    case modeTime :
      break;
    case modeSet :
```


รูปที่ 4.4 (ต่อ) โปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
      adrTime = 0;
      buffTime[0] = read_DS1307(adrHour);
      buffTime[1] = read_DS1307(adrMinute);
      break;
  }
}

void Seg_Paint() {
int i;
  seg_blink = ~seg_blink;
  switch (NowMode) {
    case modeTime :
      i = read_DS1307(adrHour);
      segData[0] = number[i>>4];
      segData[1] = number[i&15];
      i = read_DS1307(adrMinute);
      segData[2] = number[i>>4];
      segData[3] = number[i&15];
      if (seg_blink) segData[1]|=0x80;
      break;
    case modeSet :
      i = buffTime[0];
      segData[0] = number[i>>4];
      segData[1] = number[i&15] |= 0x80;
      i = buffTime[1];
      segData[2] = number[i>>4];
      segData[3] = number[i&15];
      if (!seg_blink) {
        segData[2*adrTime] = 0;
        segData[(2*adrTime)+1] = 0;
      }
      segData[1] |= 0x80;
      break;
  }
}

//=================================== Events Operating

void Key_OnDown() {
int i;
  switch (NowMode) {
      case modeSet :
            switch (keyNow) {
             case swEnter : adrTime = (adrTime+1) % 2; goto MST1;
                  case swUp :
          i = buffTime[adrTime];
          if (i<maxTime[adrTime]) {
            i++;
            if ((i&0x0f)>9) i+=6;
          }
          else i = minTime[adrTime];
          buffTime[adrTime] = i;
          goto MST1;
```


รูปที่ 4.4 (ต่อ) โปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
      case swDown :
          i = buffTime[adrTime];
          if (i>minTime[adrTime]) {
            i--;
            if ((i&0x0f)>9) i-=6;
          }
          else i = maxTime[adrTime];
          buffTime[adrTime] = i;

          MST1 :
                                seg_blink = 0;
                                seg_rate = 1;
                                break;
                    }
                    break;
      }
}

void Key_OnClick() {

}
void Key_OnFPress() {
int i;
      switch (NowMode) {
      case modeTime :
      if(keyNow==swEnter) SetMode(modeSet);// เข้าสู่โหมดตั้งเวลา
                    break;
            case  modeSet :
                        // บันทึกวันเวลาที่ตั้งใหม่
                if (keyNow==swEnter) {
        write_DS1307(adrHour, buffTime[0]);
        write_DS1307(adrMinute, buffTime[1]);
                    SetMode(modeTime);
                }
      if ((keyNow==swUp)||(keyNow==swDown)) {
        Key_OnDown();
        wKey=5;
      }
                    break;
      }
}

void Key_OnUp() {

}

//=========================== Create Events

void Scan_Key() {                  // สร้างเหตุการณ์ กดปุ่ม
unsigned char k;
  k = (~Port_A) & 7;
```


รูปที่ 4.4 (ต่อ) โปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
  if (k != keyNow) {
    if (keyNow) {
      if (wKey<50) {
        if (wKey) Key_OnClick();
        Key_OnUp();
        keyNow = 0;
      }
      wKey = 0;
    }
    if (k) {
      keyNow = k;
      wKey = 52;                  // 1 Sec
    }
  }
}

void Scan_KeyFPress() {         // สร้างเหตุการณ์กดปุ่มแช่เพื่อตั้งเวลา
  if (wKey) {
    wKey--;
    if (wKey==50) Key_OnDown();
    if (!wKey) Key_OnFPress();
  }
}
void Scan_Seg_Paint() {         // refresh seg ทุกๆ 500 ms ให้วินาทีกระพริบ
  if (seg_rate) seg_rate--;
  if (!seg_rate) {
      seg_rate = 25;                // 500ms
      Seg_Paint();
      }
}

void Scan_Segment() {
  bit_clear(port_B, 4+segDigit);
  segDigit = (segDigit+1) % 4;
  port_C = segData[3-segDigit];
  bit_set(port_B, 4+segDigit);
}

//============================== Multitasking

Task_successive() {
  Scan_Key();
}

Task_interval20ms() {
  Scan_KeyFPress();
  Scan_Seg_Paint();
}

Task_interval500ms() {

}
```


รูปที่ 4.4 (ต่อ) โปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
//============================== isr

#int_RTCC
RTCC_isr() {
  sysclk++;
  Scan_Segment();
}

//==============================Main
void main() {
int i;
  set_tris_b(0x0f);
  set_tris_c(0x00);
  port_b_pullups(TRUE);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_4);
  enable_interrupts(INT_RTCC);
  enable_interrupts(global);
  // Initial
  init_i2c();
  i = read_DS1307(adrSecond);
  if (bit_test(i, 7)) write_DS1307(adrSecond, 0);
  SetMode(modeTime);
  while (true) {
    Task_successive();
    if (sysclk>=20) {
      sysclk %= 20;
      Task_interval20ms();
      if ((++sysclk2)>=25) {
        sysclk2 = 0;
        Task_interval500ms();
      }
    }
  }
}
```

รูปที่ 4.4 (ต่อ) โปรแกรมทดลองการเชื่อมต่อแบบไอสแควร์ซี

4.4.2 ผลการทดลอง

7 SEGMENT จะแสดงค่าเป็นชั่วโมงและนาที โดยที่ DOT ของ 7 SEGMENT จะแสดงการกระพริบแทน ตัวเลขวินาที โดยการตั้งค่าจะสามารถทำได้โดยการกดที่ปุ่ม Enter ค้างเป็นเวลา 1 วินาที ตัวเลขที่แสดงอยู่จะกระพริบและเมื่อกดปุ่ม up หรือปุ่ม down ก็จะเป็นการตั้งเวลาว่าจะให้เริ่มที่เวลาเท่าใด

4.5 การรับอินพุตแบบเครื่องมือวัด

ขาอินพุตของโฟโต้จะส่งสถานะที่ได้มายัง LM339 เป็นออปแอมป์ทำหน้าที่เปรียบเทียบสัญญาณที่ได้จากโฟโต้ว่าเป็น "1" หรือเป็น "0" แล้วส่งข้อมูลไปยัง PIC16F87X ซึ่ง PIC16F87X ก็จะส่งข้อมูลมาให้ ULN2065B เป็นไดร์ฟในการขับสแต็ปปิ้งมอเตอร์ว่าจะทำงานได้หรือไม่

4.5.1 ขั้นตอนการทดลอง

1. เชื่อมต่อโมดูลอินพุตอินสตรูเมนต์เข้ากับโมดูลหลัก

2.จากนั้นก็ทำการโปรแกรมที่ได้เขียนไว้ เพื่อทำการทดลองว่ารับเอาต์พุตจากโฟโต้ทรานซิสเตอร์ออกทางสแต็ปปิ้งมอเตอร์ได้หรือไม่

3. สั่งไมโครคอนโทรลเลอร์ทำงาน โดยการรันโปรแกรมผ่านทาง IC Prog.exe

```
#include "Instrument.h"
#include "amPicLib.h"
void main() {
int Direct=0;
int i=0;
  set_tris_c(0x0f);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_2);
  while (1) {
    if (~bit_test(port_C, 1)) Direct = 2;
    if (~bit_test(port_C, 3)) Direct = 1;
    if (~bit_test(port_C, 2)) Direct = 0;
    switch (Direct) {
      case 0 :
        port_C &= 0x0f;
        i = 0;
        break;
      case 1 :
        i <<= 1;
        if (!i) i=0x10;
        port_C = (port_C & 0x0f)+(i&0xf0);
        delay_ms(20);
        break;
      case 2 :
        i >>= 1;
        if (!(i&0xf0)) i=0x80;
        port_C = (port_C & 0x0f)+(i&0xf0);
        delay_ms(20);
        break;
   }
}
}
```

รูปที่ 4.5 โปรแกรมทดลองการรับอินพุตแบบเครื่องมือวัด

4.5.2 ผลการทดลอง

เมื่อนำวัตถุสีขาวไปปิดที่ OPTO_3 จะทำให้แสตปปิ้งมอเตอร์ทำการหมุนไปทางด้านซ้าย และเมื่อนำวัตถุสีขาวไปปิดที่ OPTO_2 แสตปปิ้งมอเตอร์จะทำการหยุดหมุนทันที และเมื่อนำวัตถุสีขาวไปปิดที่ OPTO_1 จะทำให้แสตปปิ้งมอเตอร์หมุนไปทางด้านขวา

4.6 การแสดงผลด้วยดอทเมตริกซ์ 24x8

การแสดงผลแอลอีดีดอตเมตริกซ์ แสดงผลได้ทั้งสีแดง สีเขียว แต่เมื่อให้ทั้งสองสีติดพร้อมกันก็จะแสดงเป็นสีส้ม การทดลองจะทำโดยให้แอลอีดีติดเป็นไฟวิ่งทีละคอลัมน์ โดยติดรอบละ 1 สีสลับกันไปทั้งสามสี เพื่อดูว่าแอลอีดีทุกดวงทำงานได้หรือไม่

4.6.1 ขั้นตอนการทดลอง

1. ต่อโมดูลแสดงผลแอลอีดีดอตเมตริกซ์ขนาด 24x8 เข้ากับโมดูลหลัก
2. จากผังการทำงานของโปรแกรมที่ออกแบบไว้ ทำการเขียนโปรแกรมที่ใช้ในการทดลองการทำงานของโมดูลแอลอีดีดอตเมตริกซ์ขนาด 24x8 โปรแกรมแสดงดังรูปที่ 4.6
3. สั่งให้ไมโครคอนโทรลเลอร์ทำงาน สังเกตการติดของ LED ว่าติดครบทุกดวงหรือไม่

```
//Matrix
// Port B   Column
// Port C   Red
// Port D   Green
#include "matrix.h"
#include "amPicLib.h"
int mxRed[24];
int mxGrn[24];
int mxD;
#define Col   port_B
#define Red   port_C
#define Grn   port_D
// Font
byte const Font[43][5] = {
  {0x7C, 0x8A, 0x92, 0xA2, 0x7C}, // 0
  {0x00, 0x42, 0xFE, 0x02, 0x00}, // 1
  {0x42, 0x86, 0x8A, 0x92, 0x62}, // 2
  {0x84, 0x82, 0xA2, 0xD2, 0x8C}, // 3
  {0x18, 0x28, 0x48, 0xFE, 0x08}, // 4
  {0xE4, 0xA2, 0xA2, 0xA2, 0x9C}, // 5
  {0x3C, 0x52, 0x92, 0x92, 0x0C}, // 6
  {0x80, 0x80, 0x9E, 0xA0, 0xC0}, // 7
```

รูปที่ 4.6 โปรแกรมทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

```
   {0x00, 0x6C, 0x6C, 0x00, 0x00}, // 8
   {0x00, 0x6A, 0x6C, 0x00, 0x00}, // 9
   {0x10, 0x28, 0x44, 0x82, 0x00}, // <
   {0x28, 0x28, 0x28, 0x28, 0x28}, // =
   {0x00, 0x82, 0x44, 0x28, 0x10}, // >
   {0x40, 0x80, 0x8A, 0x90, 0x60}, // ?
   {0x7C, 0x82, 0xBA, 0xAA, 0x7A}, // @
   {0x7E, 0x90, 0x90, 0x90, 0x7E}, // A
   {0xFE, 0x92, 0x92, 0x92, 0x6C}, // B
   {0x7C, 0x82, 0x82, 0x82, 0x44}, // C
   {0xFE, 0x82, 0x82, 0x44, 0x38}, // D
   {0xFE, 0x92, 0x92, 0x92, 0x82}, // E
   {0xFE, 0x90, 0x90, 0x90, 0x80}, // F
   {0x7C, 0x82, 0x92, 0x92, 0x5E}, // G
   {0xFE, 0x10, 0x10, 0x10, 0xFE}, // H
   {0x00, 0x82, 0xFE, 0x82, 0x00}, // I
   {0x04, 0x02, 0x82, 0xFC, 0x80}, // J
   {0xFE, 0x10, 0x28, 0x44, 0x82}, // K
   {0xFE, 0x02, 0x02, 0x02, 0x02}, // L
   {0xFE, 0x40, 0x30, 0x40, 0xFE}, // M
   {0xFE, 0x20, 0x10, 0x08, 0xFE}, // N
   {0x7C, 0x82, 0x82, 0x82, 0x7C}, // O
   {0xFE, 0x90, 0x90, 0x90, 0x60}, // P
   {0x7C, 0x82, 0x8A, 0x84, 0x7A}, // Q
   {0xFE, 0x90, 0x98, 0x94, 0x62}, // R
   {0x64, 0x92, 0x92, 0x92, 0x4C}, // S
   {0x80, 0x80, 0xFE, 0x80, 0x80}, // T
   {0xFC, 0x02, 0x02, 0x02, 0xFC}, // U
   {0xF8, 0x04, 0x02, 0x04, 0xF8}, // V
   {0xFC, 0x02, 0x1C, 0x02, 0xFC}, // W
   {0xC6, 0x28, 0x10, 0x28, 0xC6}, // X
   {0xE0, 0x10, 0x0E, 0x10, 0xE0}, // Y
   {0x86, 0x8A, 0x92, 0xA2, 0xC2}  // Z
};

void ScanMatrix() {
int i;
  mxD = (mxD+1) % 24;
  i = 23-mxD;
  Red = 0x00;
  Grn = 0x00;
  Col = (0x08<<(i/8)) + (i%8);
  Red = mxRed[mxD];
  Grn = mxGrn[mxD];
}

#int_RTCC
RTCC_isr() {
  ScanMatrix();
}
```

รูปที่ 4.6 (ต่อ) โปรแกรมทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

```
long Speed;
int Color, cor;
enum {cRed=1, cGreen=2, cOrange=3, cAlter=4};

TextShiftLeft(char ch) {
int i, j;
  switch (Color) {
    case cRed : cor = 1; break;
    case cGreen : cor = 2; break;
    case cOrange : cor = 3; break;
    case cAlter : cor = (cor%3)+1; break;
  }

  for (j=0; j<6; j++) {
    for (i=0; i<23; i++) {
      mxRed[i] = mxRed[i+1];
      mxGrn[i] = mxGrn[i+1];
    }
    if ( bit_test(cor, 0) && (j<5) && (ch!=' ') ) mxRed[23]=Font[ch-
'0'][j]; else mxRed[23]=0;
    if ( bit_test(cor, 1) && (j<5) && (ch!=' ') ) mxGrn[23]=Font[ch-
'0'][j]; else mxGrn[23]=0;
    delay_ms(Speed);
  }
}

void main() {
int i, j;
  set_tris_b(0x00);
  set_tris_c(0x00);
  set_tris_d(0x00);

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  enable_interrupts(INT_RTCC);
  enable_interrupts(global);

  for (i=0; i<24; i++) {
    mxRed[i] = 0;
    mxGrn[i] = 0;
  }

  Speed = 50;
  Color = cAlter;
  while (true) {
    TextShiftLeft("THE PIC16F87X MICROCONTROLLER ");
    TextShiftLeft("DEMONSTRATOR CONTROLED BY C LANGUAGE    ");
  }
}
```

รูปที่ 4.6 (ต่อ) โปรแกรมทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

4.6.2 ผลการทดลอง

แอลอีดีติดเป็นคำว่า THE PIC16F87X MICROCONTROLLER DEMONSTRATOR CONTROLED BY C LANGUAGE วิ่งจากทางด้านขวาค่อยๆขยับไปทางด้านซ้าย โดยสลับสีกันทั้ง 3 สี และสามารถสามารถปรับความเร็วในการเลื่อนของตัวอักษรได้

บทที่ 5
บทสรุป

5.1 สรุป

ชุดปฏิบัติการทดลองไมโครคอนโทรลเลอร์ PIC16F87X สร้างขึ้นเพื่อใช้ศึกษาทฤษฎีและหลักการ ตลอดจนการนำไปใช้งานของไมโครคอนโทรลเลอร์ PIC16F87X ซึ่งได้กำหนดขอบเขตของโครงงานไว้ดังนี้

1. โมดูลหลัก PIC-ICSP
2. การแสดงผลด้วยจอแบบผลึกเหลว
3. การเชื่อมต่อแบบพอร์ตอนุกรม
4. การเชื่อมต่อแบบไอสแควร์ซี
5. การรับอินพุตแบบเครื่องมือวัด
6. การแสดงผลด้วยดอทเมตริกซ์ 24x8

5.2 ปัญหาและแนวทางแก้ไข

5.2.1 ปัญหาทางด้านฮาร์ดแวร์

1. ปัญหา การออกแบบวงจรของแต่ละโมดูลในแผงทดลองวงจร อุปกรณ์บางตัวมีขาที่ไม่ตรงกับช่องเสียบอุปกรณ์ทำให้ต้องใช้อุปกรณ์ตัวอื่นแทนในการทดลองจึงต้องเปลี่ยนโครงสร้างของวงจรใหม่ ซึ่งเป็นการเสียเวลาเป็นอย่างมากในการออกแบบแผ่นวงจรพิมพ์ใหม่

แนวทางแก้ไข การทดลองวงจรควรทดลองด้วยอุปกรณ์ที่จะใช้จริงๆทั้งหมด เพื่อให้ผลการทดลองมีความสมบูรณ์มากที่สุด ก่อนจะนำไปออกแบบแผ่นวงจรพิมพ์ เพื่อไม่ให้เกิดความผิดพลาด

2. ปัญหา การออกแบบลายทองแดงของแผ่นวงจรพิมพ์บริเวณจุดบัดกรีเล็กเกินไป ซึ่งทำให้แผ่นวงจรพิมพ์สึกหรอง่าย ลายทองแดงขาดเร็ว แผ่นวงจรพิมพ์ไม่ทนทาน

แนวทางแก้ไข การออกแบบควรใช้ขนาดของลายทองแดงให้ใหญ่ขึ้นกว่าเดิม โดยเฉพาะจุดบัดกรี แต่ก็ไม่ควรเดินให้ติดกันมากจนเกินไปเพราะอาจทำให้เกิดการลัดวงจรได้

5.2.2 ปัญหาทางด้านซอฟต์แวร์

ปัญหา คำสั่งของไมโครคอนโทรลเลอร์ PIC นั้นมีอยู่น้อยจำเป็นจะต้องใช้คำสั่งหลายๆ คำสั่งมาประกอบกันขึ้นเป็นกลุ่มคำสั่งใหม่ที่ประมวลผลบางอย่าง

แนวทางแก้ไข การเขียนโปรแกรมควรจะเขียนให้อยู่ในรูปแบบเป็นโครงสร้างให้มากที่สุด เน้นให้หาจุดผิดพลาดเพื่อแก้ไขโปรแกรมได้ง่ายและเพิ่มการเขียนคำอธิบายในแต่ละส่วนให้มากขึ้นเพื่อการทำความเข้าใจโปรแกรมได้ง่ายในภายหลัง

5.3 แนวทางการพัฒนา

1. การเพิ่มโมดูลสำหรับการทดลอง ให้มากขึ้นเพื่อสามารถทดลองใช้เชื่อมต่อกับอุปกรณ์ได้อย่างหลากหลาย และจะได้เข้าใจถึงวิธีการใช้อุปกรณ์เหล่านั้นด้วย

2. สร้างโมดูลสำหรับต่อขยายพอร์ตเพิ่ม เพื่อให้สามารถต่อใช้งานโมดูลอื่นๆ ได้พร้อมกันหลายๆ ตัว สามารถประกอบกันเป็นงานที่ใหญ่ขึ้นได้

3. ศึกษาภาษาที่ใช้เขียนโปรแกรมสั่งไมโครคอนโทรลเลอร์ เช่น ภาษาซี หรือ ภาษาเบสิก ซึ่งจะทำให้สามารถเขียนสั่งงานไมโครคอนโทรลเลอร์ได้ง่ายขึ้น

4. ใช้โมดูลในแต่ละโมดูลในการทดลองปฏิบัติเป็นใบงานในการเรียนการสอนจริงได้

บรรณานุกรม

กฤษดา ใจเย็น และคณะ. **เรียนรู้ไมโครคอนโทรลเลอร์อย่างง่ายกับเบสิกแสตมป์ 2**. กรุงเทพฯ : อินโนเวตีฟ เอ็กเพอร์ริเมนต์. ม.ป.ป.

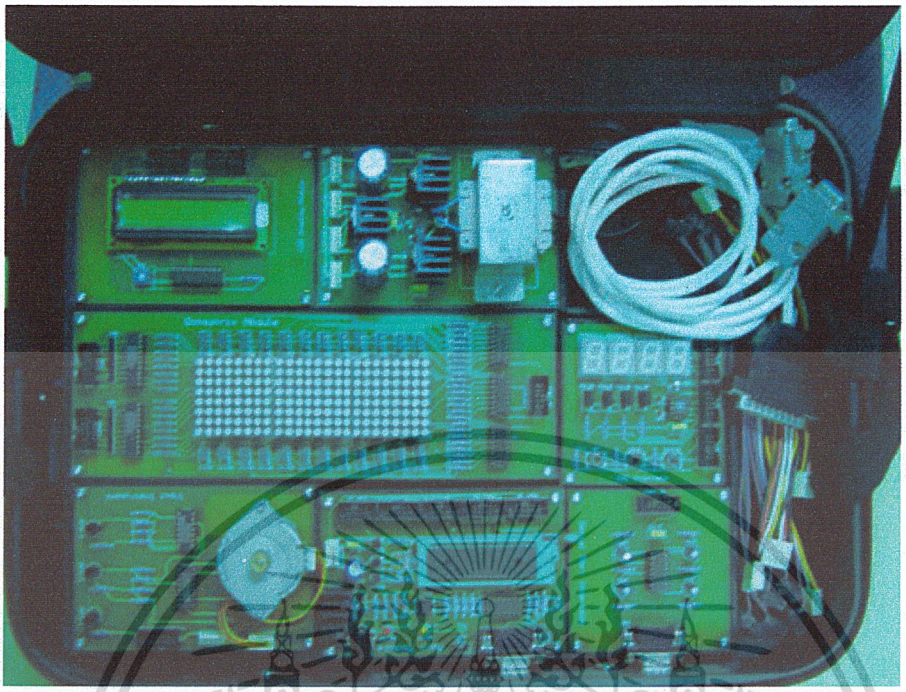
กฤษดา ใจเย็น และชัยวัฒน์ ลิ้มพรจิตวิไล. **เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ PIC16F84**. กรุงเทพฯ : อินโนเวตีฟ เอ็กเพอร์ริเมนต์. ม.ป.ป.

กนก กุสุมาลย์นุกูล และไกลวุฒิ มั่นเสถียรสิน. **คู่มือการเขียนโปรแกรม Delphi4**. พิมพ์ครั้งที่ 2. กรุงเทพฯ : ซัคเซส มีเดีย. ม.ป.ป.

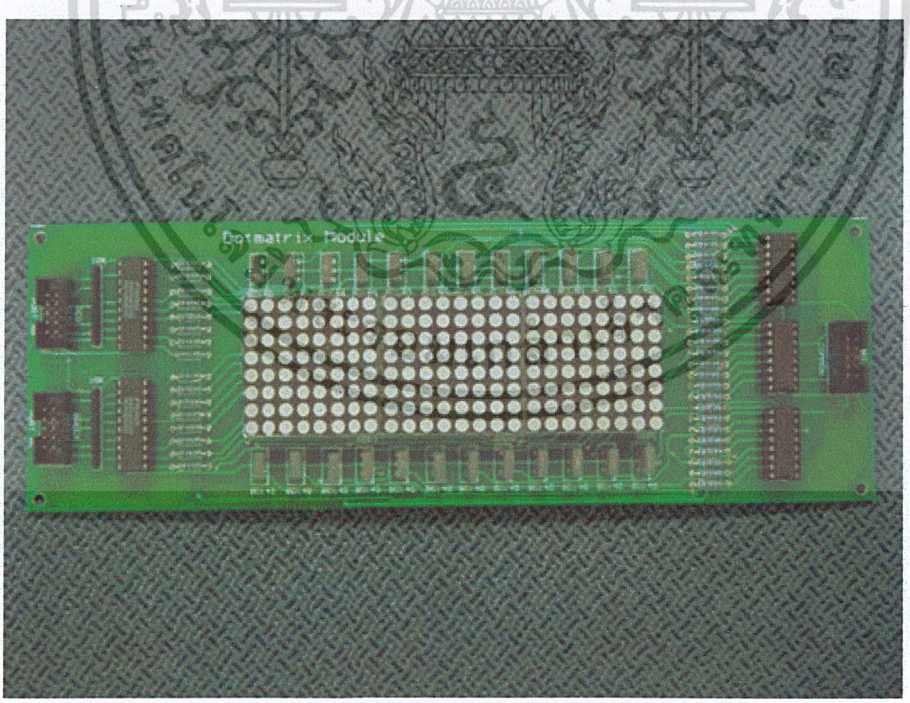
นพพร วัฒนสิทธิ์. "**ชุดฝึกไมโครคอนโทรลเลอร์ 8051**". ปริญญานิพนธ์ครุศาสตร์อุตสาหกรรมบัณฑิต สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.2541

Microchip. **Data sheet**. [Online]. Available : http://www.microchip.com/10/lit/pline/picmicro/index.htm. 2544

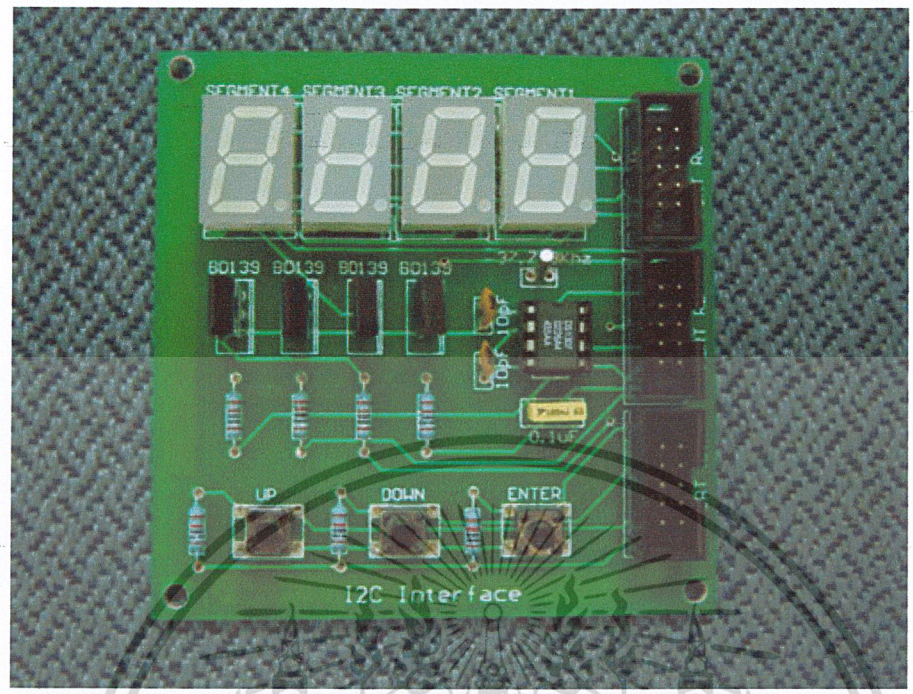
ภาคผนวก ก
เครื่องต้นแบบ



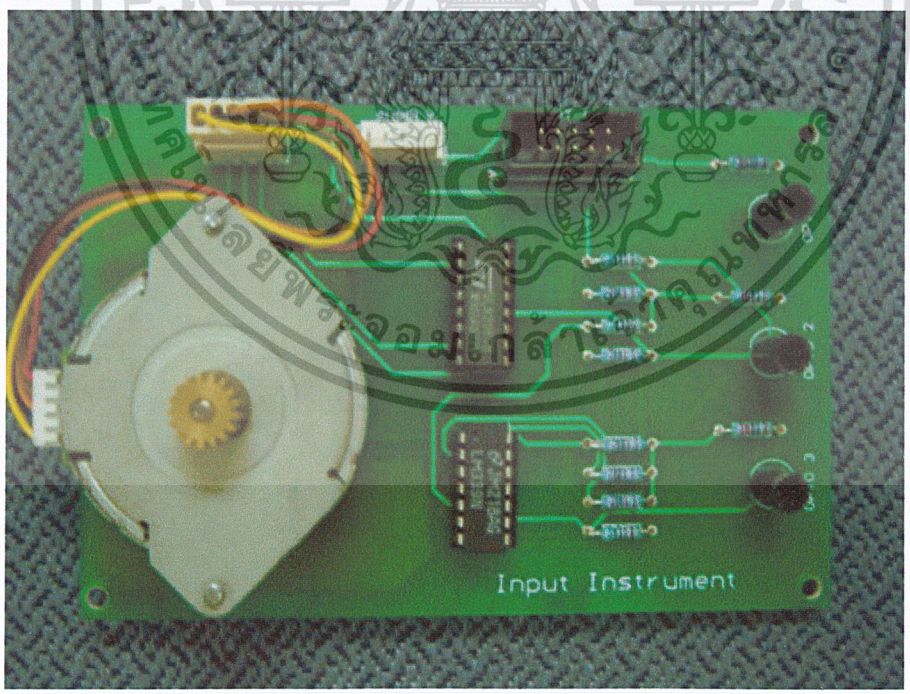
รูปที่ ก.1 เครื่องต้นแบบโมดูลทั้งหมด



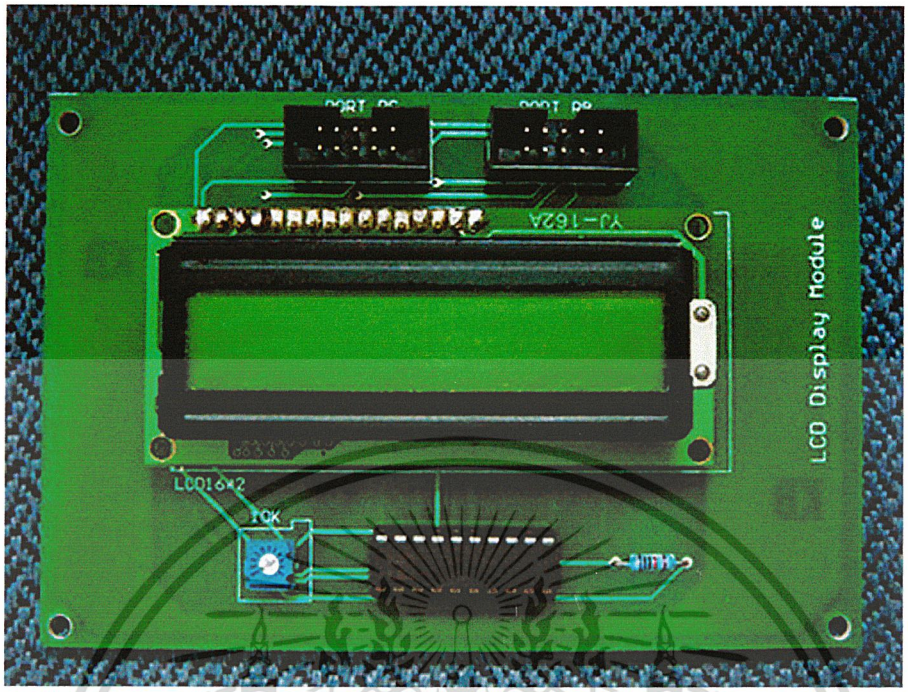
รูปที่ ก.2 การแสดงผลด้วยดอทเมตริกซ์ 24x8



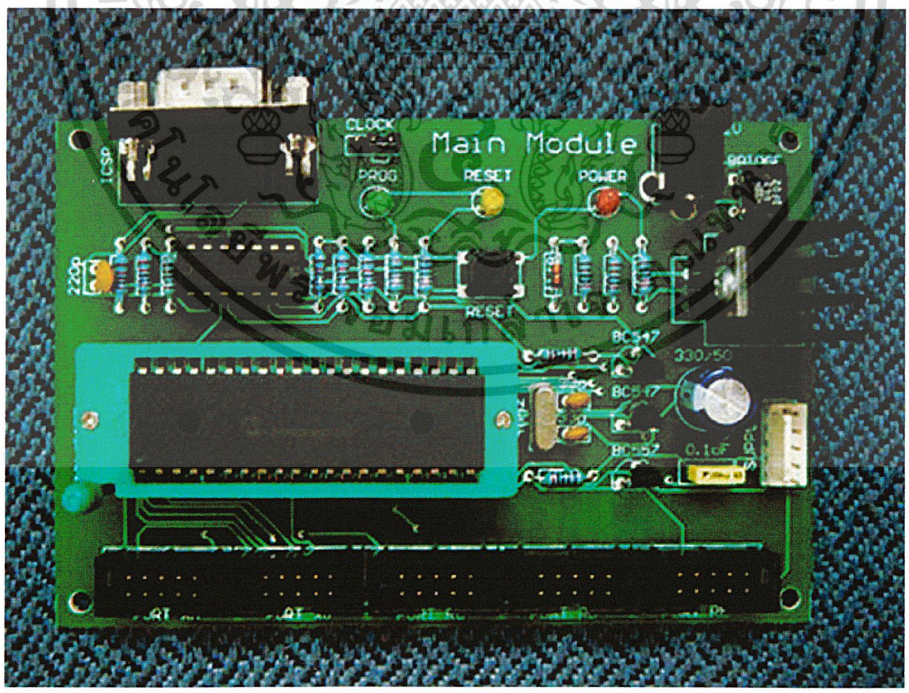
รูปที่ ก.3 การเชื่อมต่อแบบไอสแควร์ซี



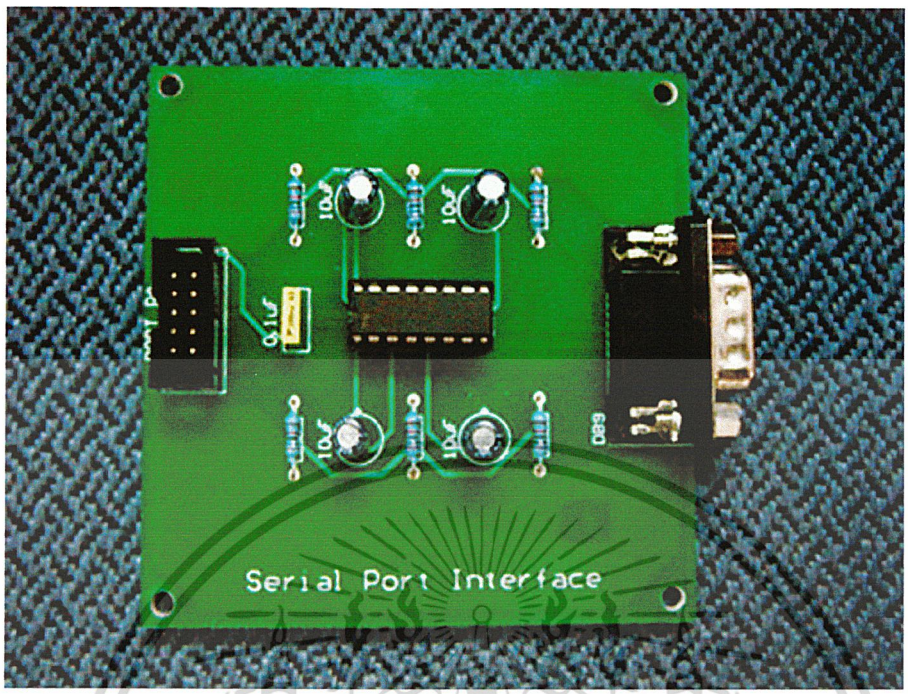
รูปที่ ก.4 การรับอินพุตแบบเครื่องมือวัด



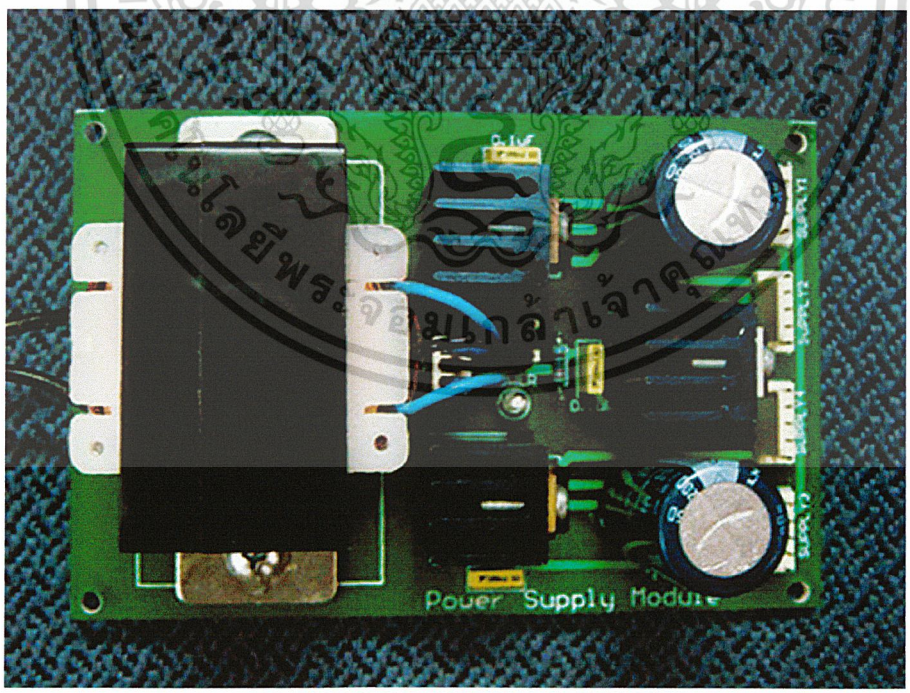
รูปที่ ก.5 การแสดงผลด้วยจอแบบผลึกเหลว



รูปที่ ก.6 โมดูล PIC-ICSP



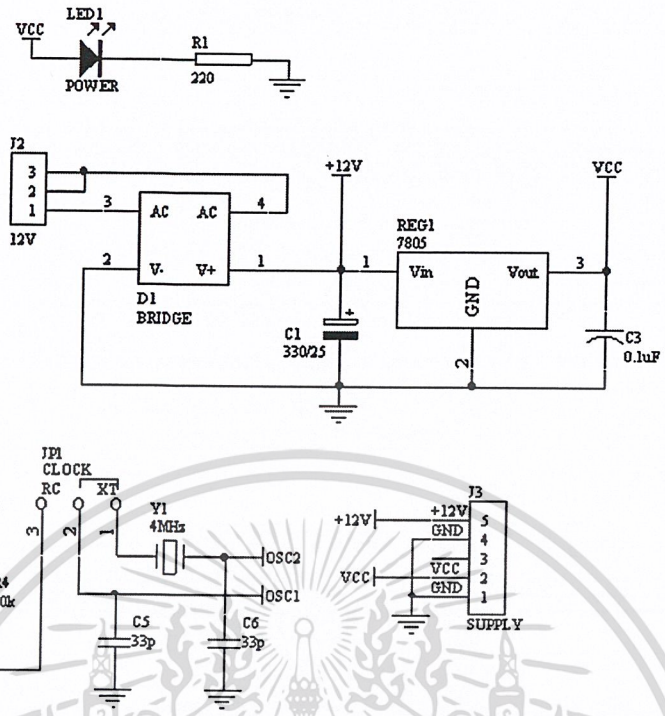
รูปที่ ก.7 การเชื่อมต่อแบบพอร์ตอนุกรม



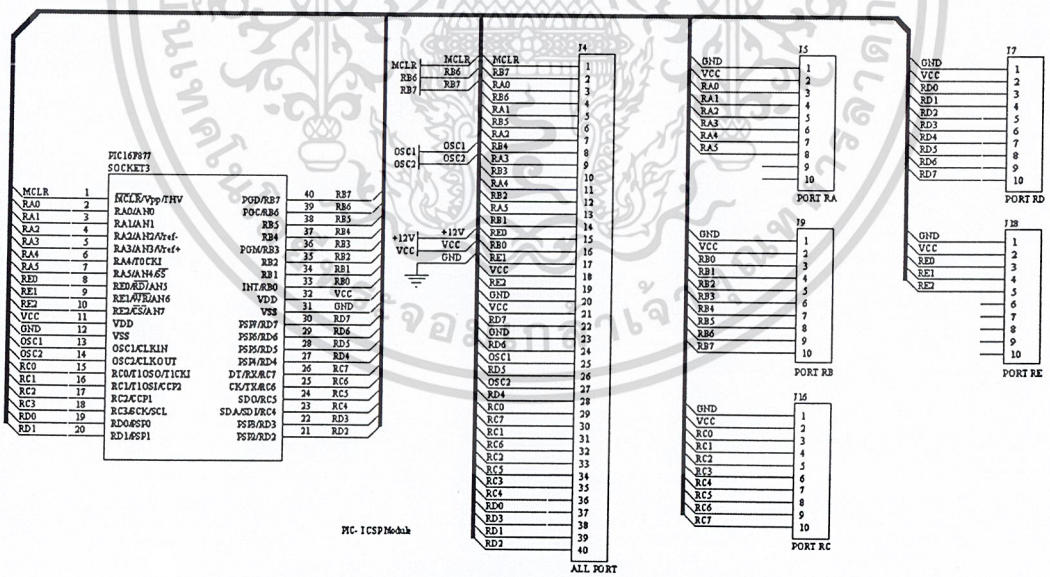
รูปที่ ก.8 โมดูลภาคจ่ายไฟ

ภาคผนวก ข

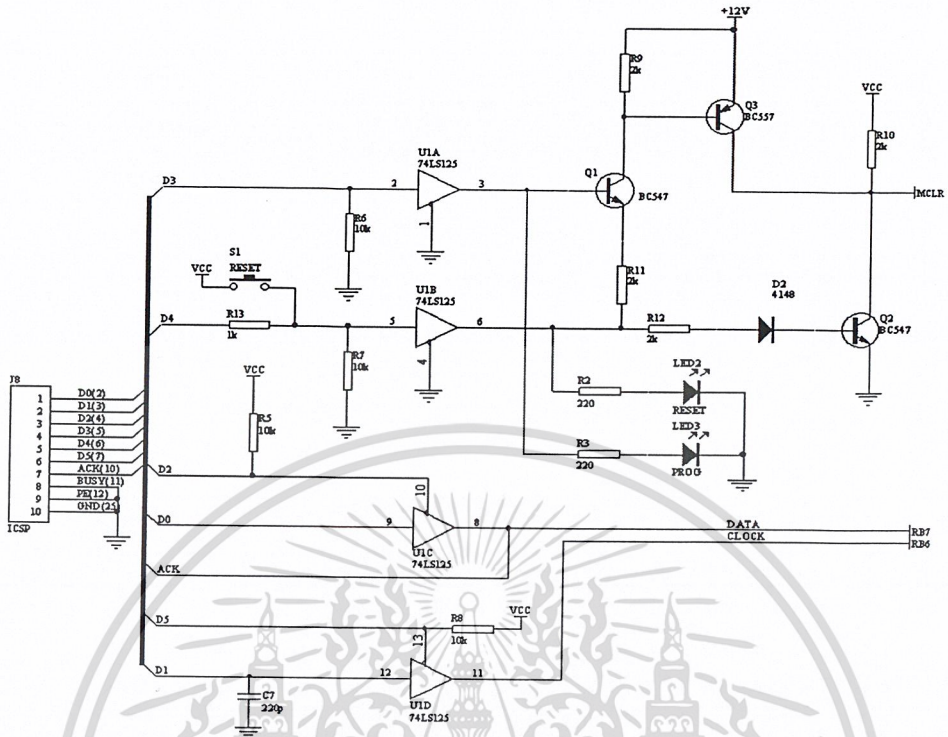
วงจรและแผ่นวงจรพิมพ์



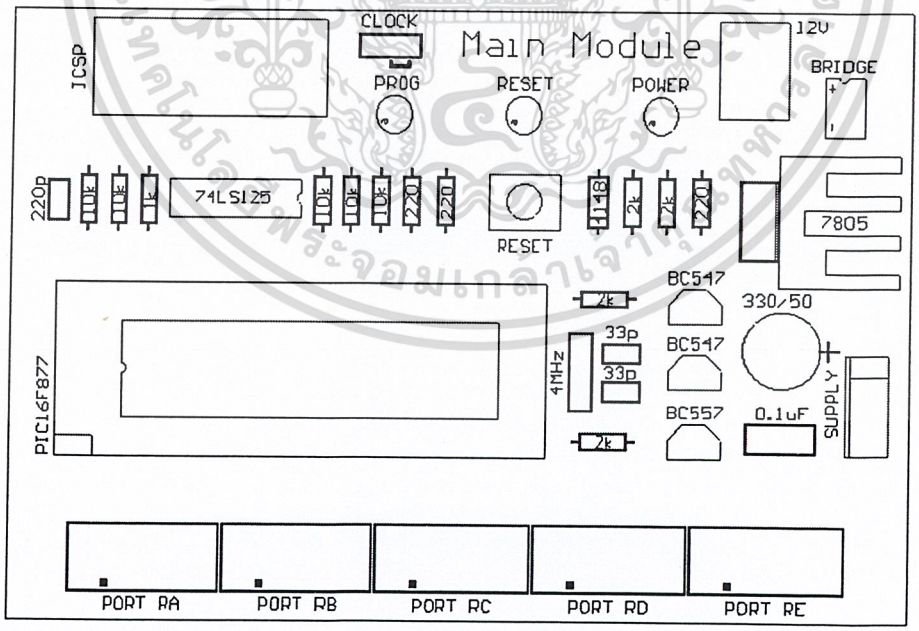
รูปที่ ข.1 วงจรของโมดูลหลัก PIC-ICSP



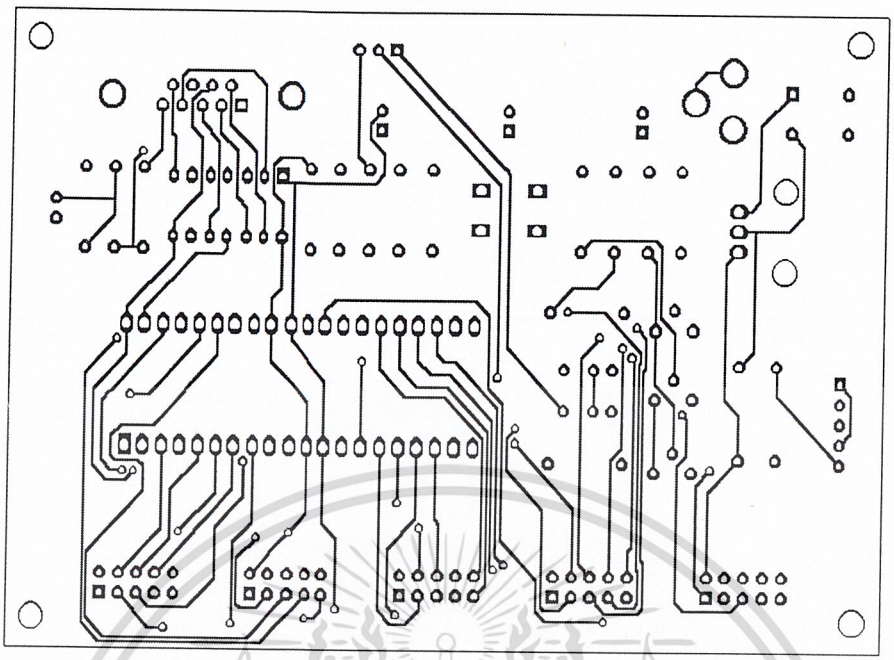
รูปที่ ข.1 (ต่อ) วงจรของโมดูลหลัก PIC-ICSP



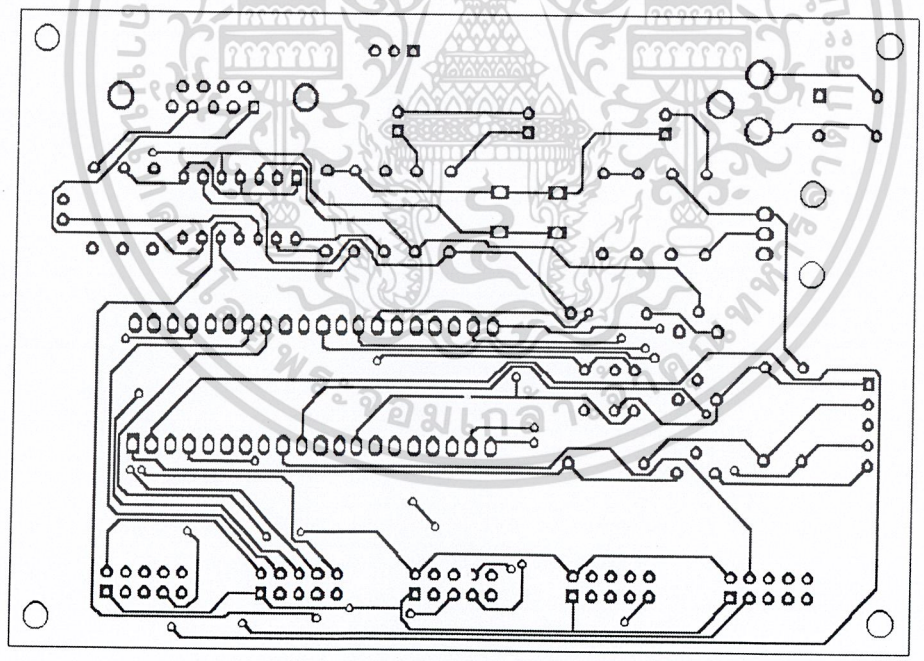
รูปที่ ข.1 (ต่อ) วงจรของโมดูลหลัก PIC-ICSP



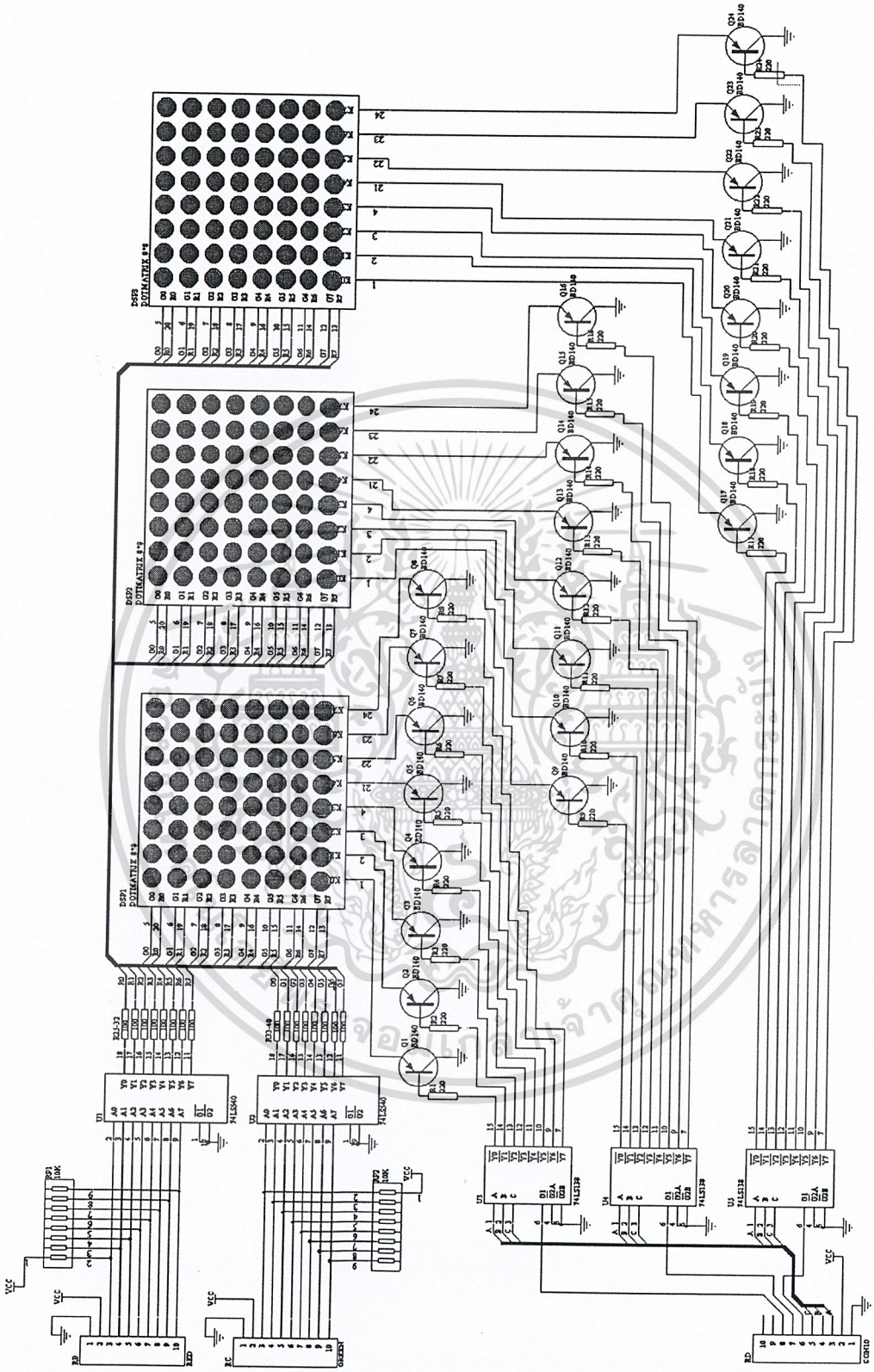
รูปที่ ข.2 การวางอุปกรณ์ของโมดูลหลัก PIC-ICSP



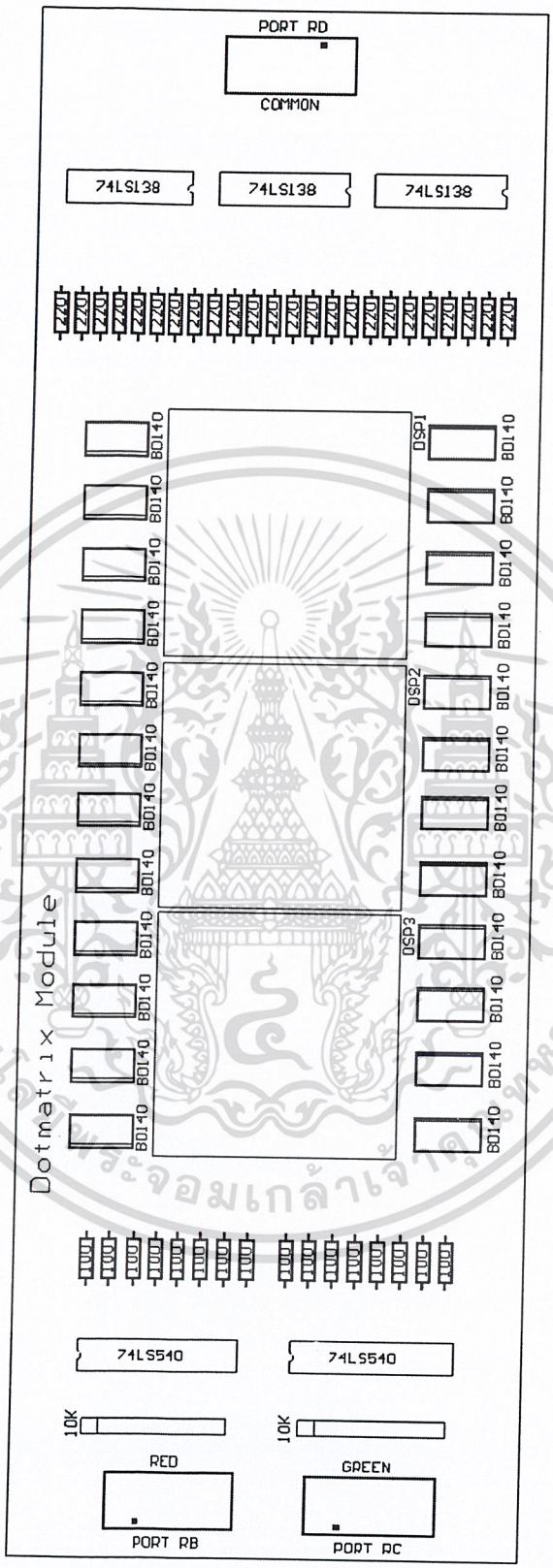
รูปที่ ข.3 วงจรพิมพ์ด้านล่างของโมดูลหลัก PIC-ICSP



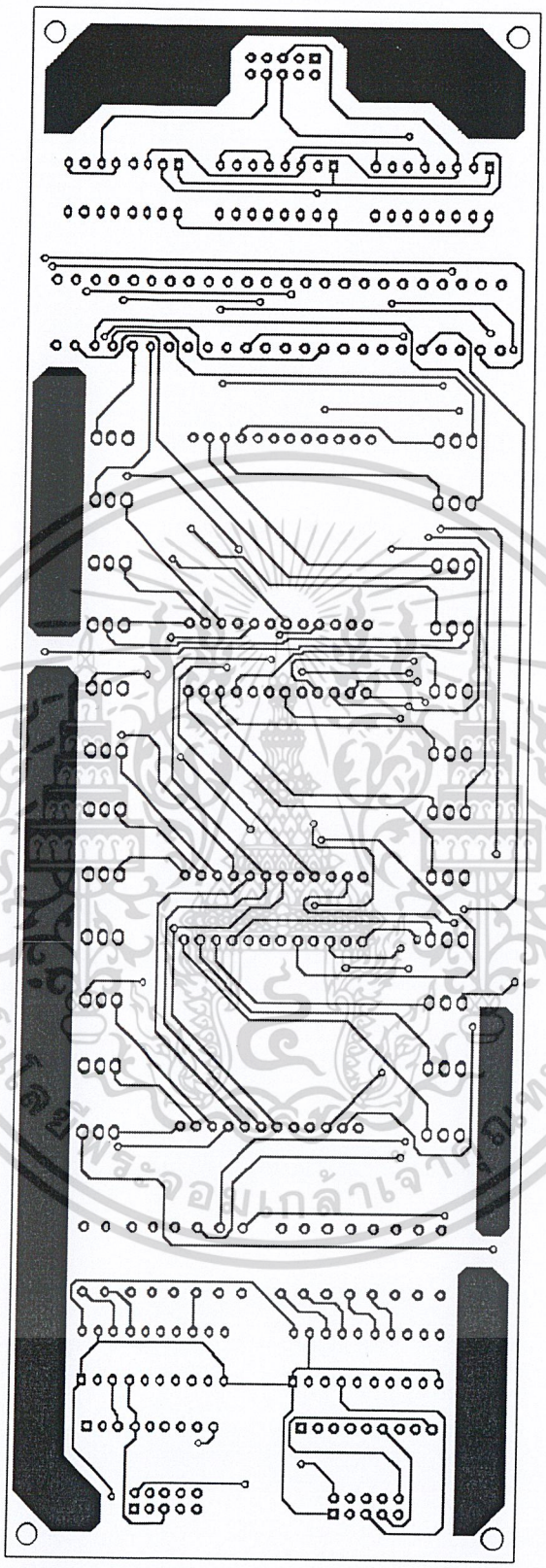
รูปที่ ข.4 วงจรพิมพ์ด้านบนของโมดูลหลัก PIC-ICSP



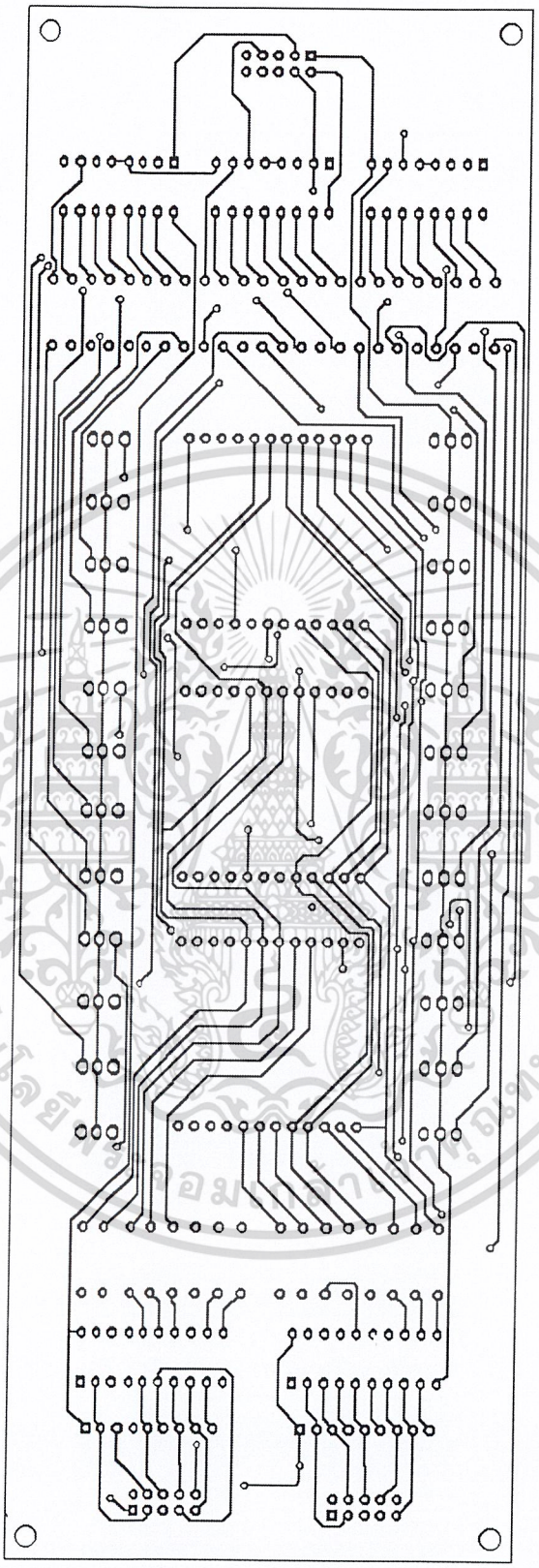
รูปที่ ข.5 วงจรของการแสดงผลด้วยดอทเมตริกซ์ขนาด 24x8



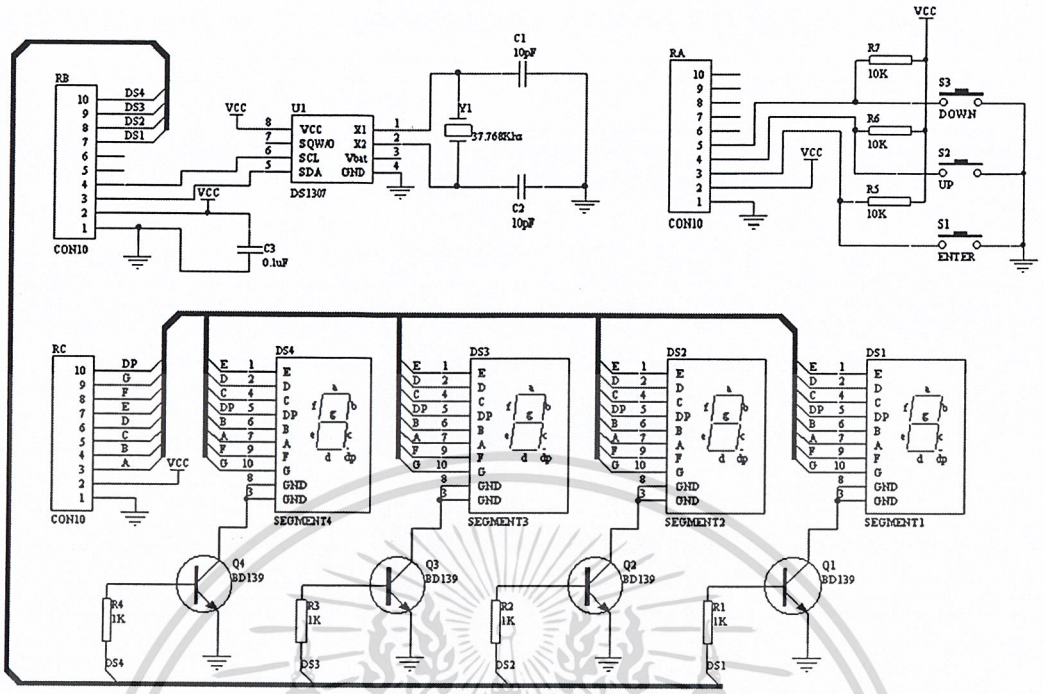
รูปที่ ข.6 การวางอุปกรณ์ของการแสดงผลด้วยดอทเมตริกซ์ขนาด 24x8



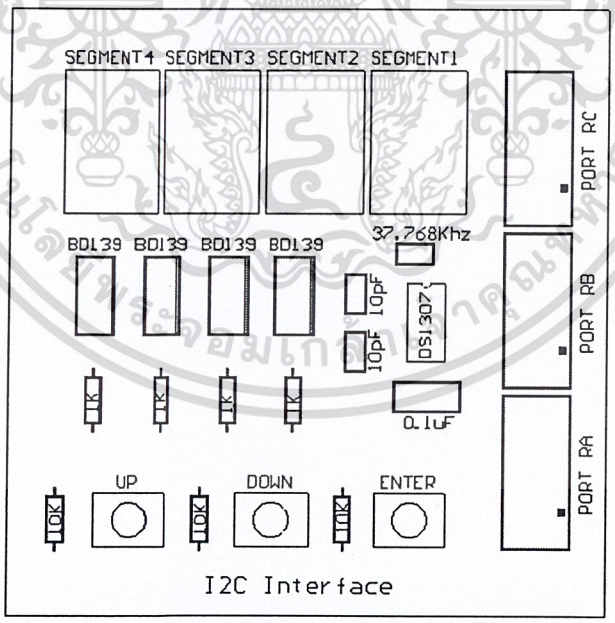
รูปที่ ข.7 วงจรพิมพ์ด้านล่างของการแสดงผลด้วยดอทเมตริกซ์ขนาด 24x8



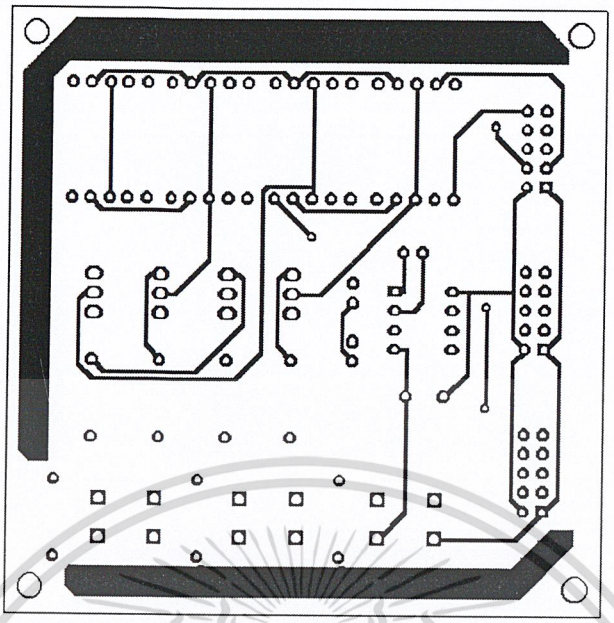
รูปที่ ข.8 วงจรพิมพ์ด้านบนของการแสดงผลด้วยดอทเมตริกซ์ขนาด 24*8



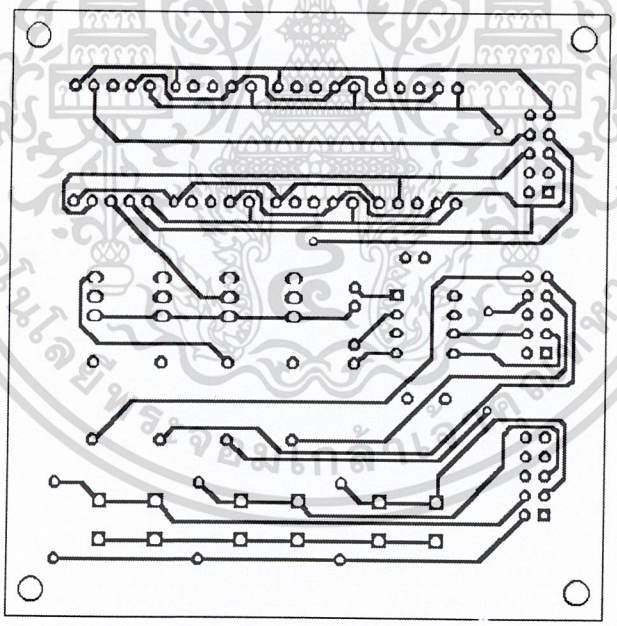
รูปที่ ข.9 วงจรของการเชื่อมต่อแบบไอสแควร์ซี



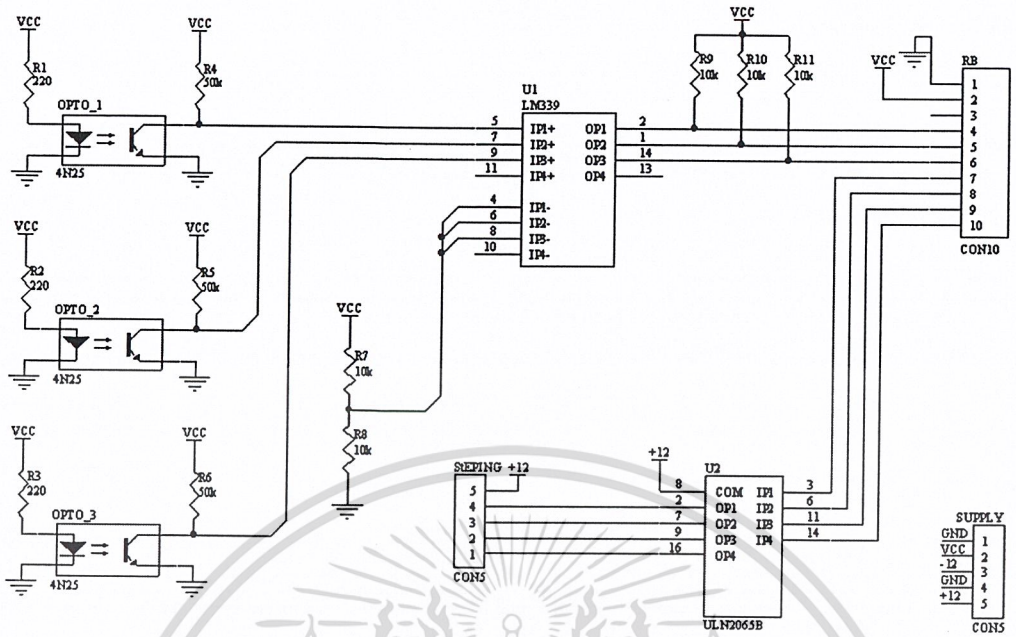
รูปที่ ข.10 การวางอุปกรณ์ของการเชื่อมต่อแบบไอสแควร์ซี



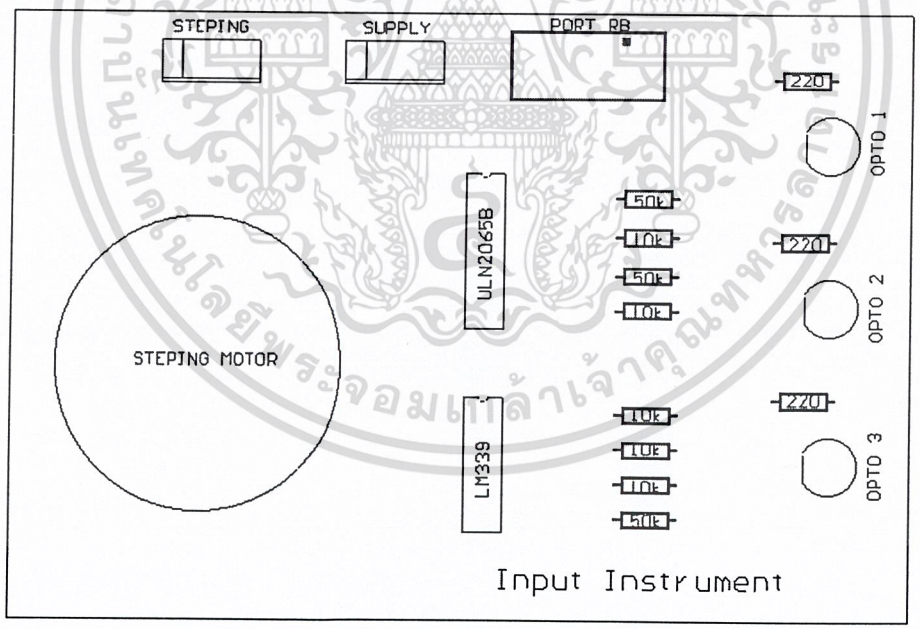
รูปที่ ข.11 วงจรพิมพ์ด้านล่างของการเชื่อมต่อแบบไอสแควร์ซี



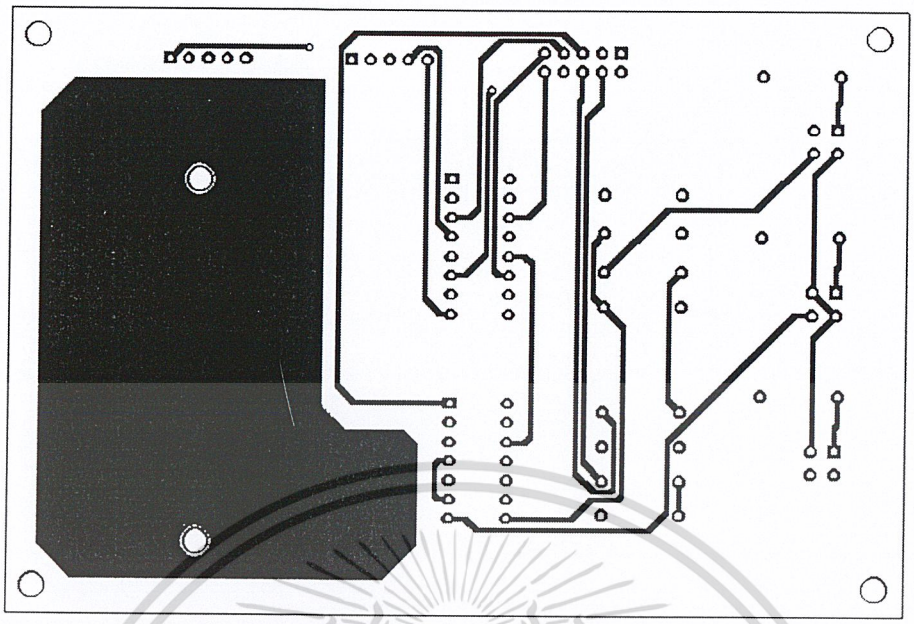
รูปที่ ข.12 วงจรพิมพ์ด้านบนของการเชื่อมต่อแบบไอสแควร์ซี



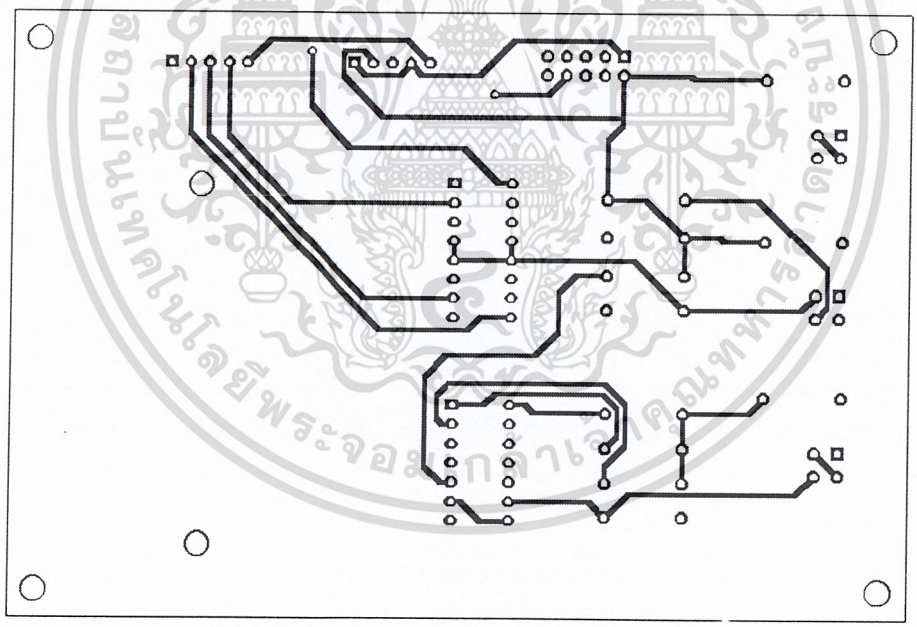
รูปที่ ข.13 วงจรของการรับอินพุตแบบเครื่องมือวัด



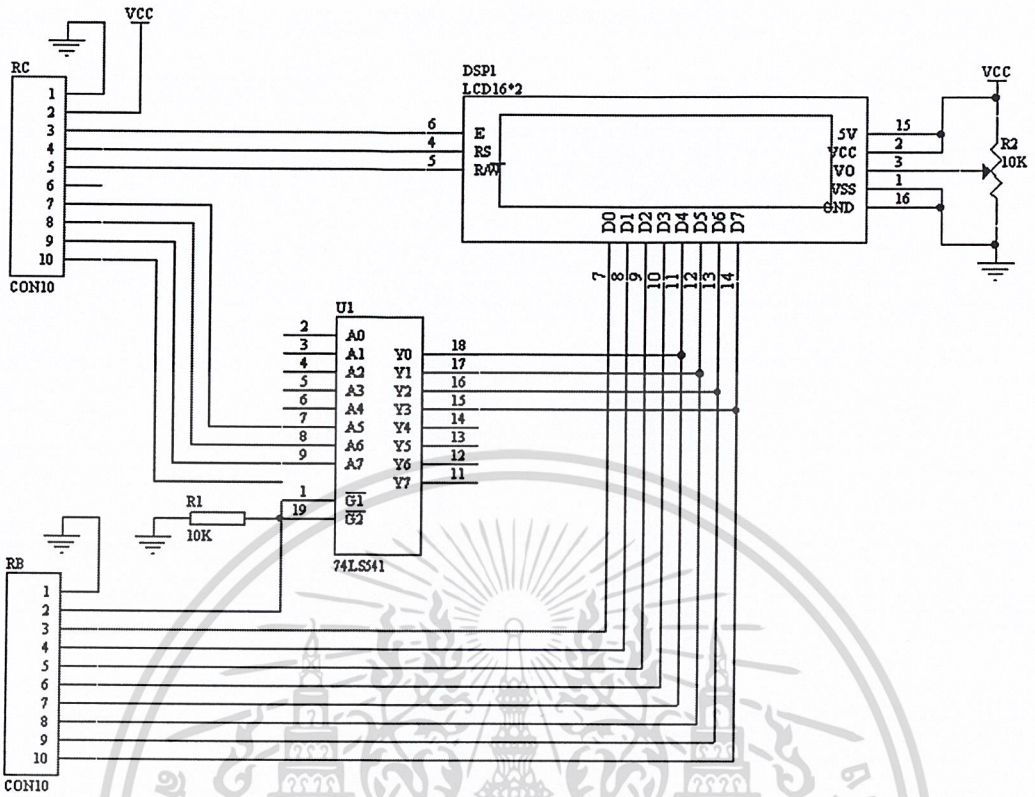
รูปที่ ข.14 การวางอุปกรณ์ของของการรับอินพุตแบบเครื่องมือวัด



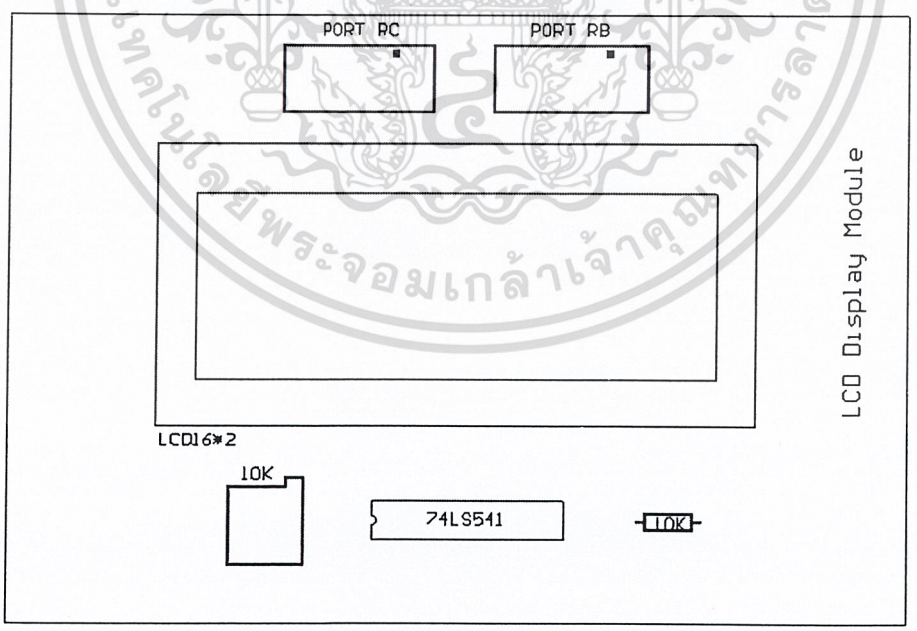
รูปที่ ข.15 วงจรพิมพ์ด้านล่างของการรับอินพุตแบบเครื่องมือวัด



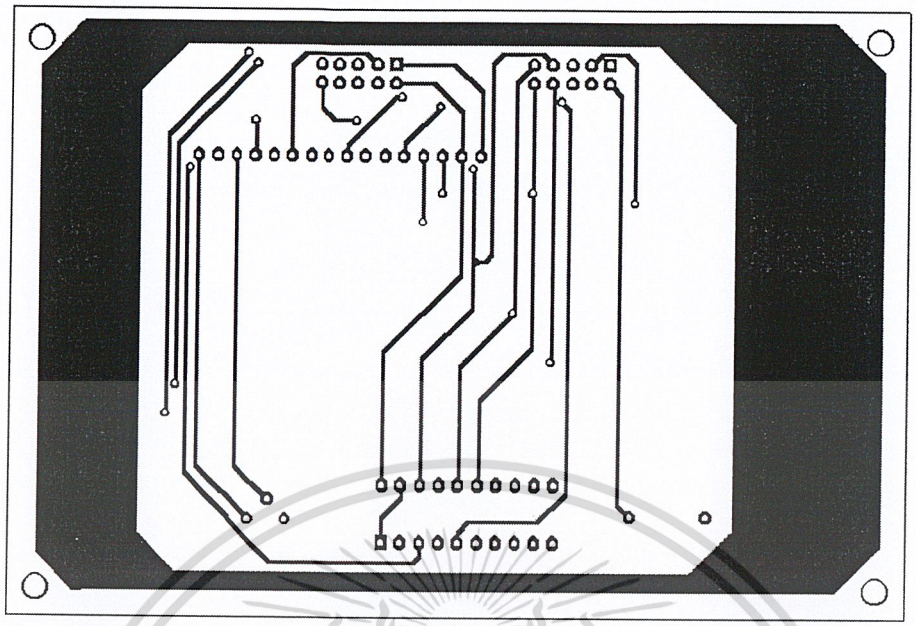
รูปที่ ข.16 วงจรพิมพ์ด้านบนของการรับอินพุตแบบเครื่องมือวัด



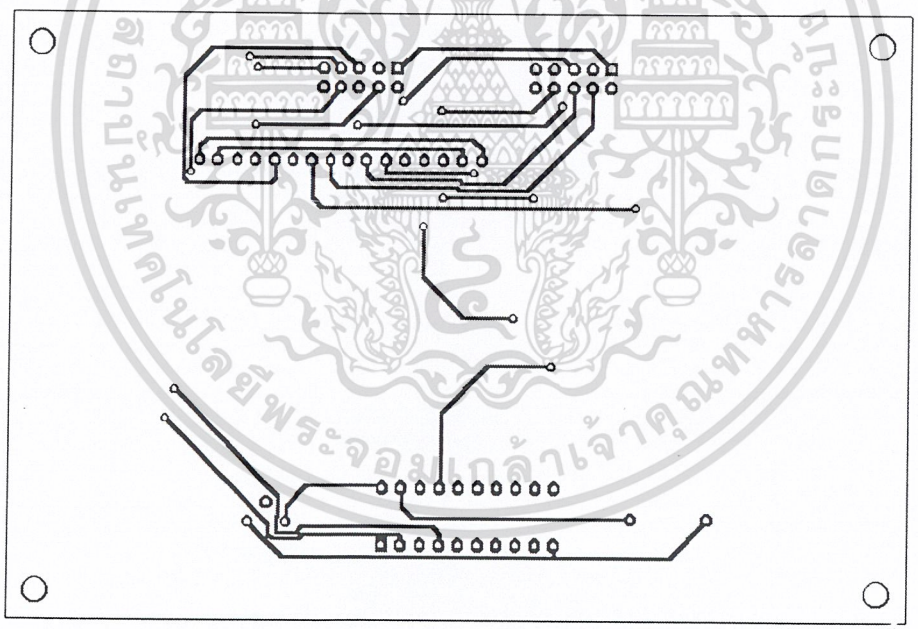
รูปที่ ข.17 วงจรของการแสดงผลด้วยจอแบบผลึกเหลว



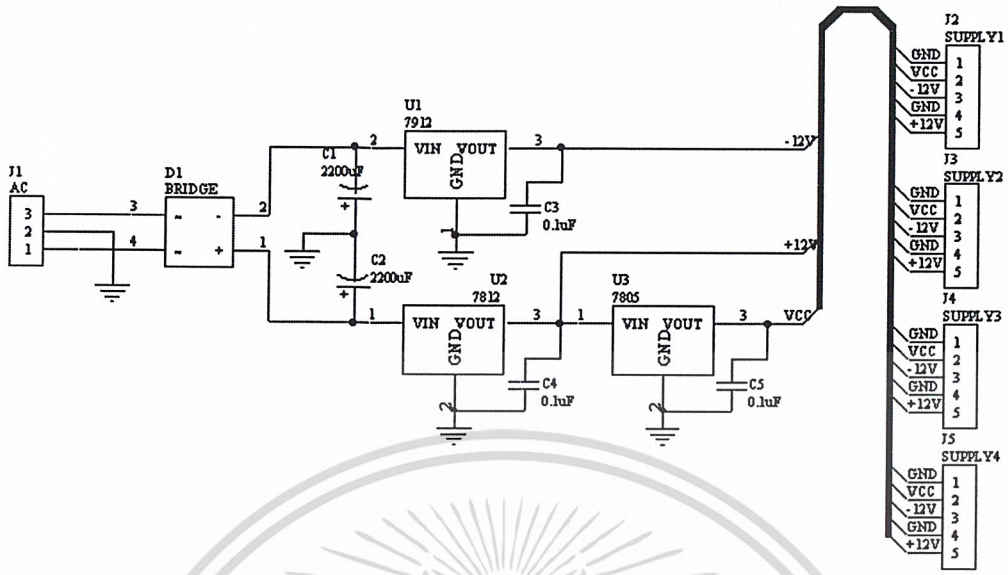
รูปที่ ข.18 การวางอุปกรณ์ของการแสดงผลด้วยจอแบบผลึกเหลว



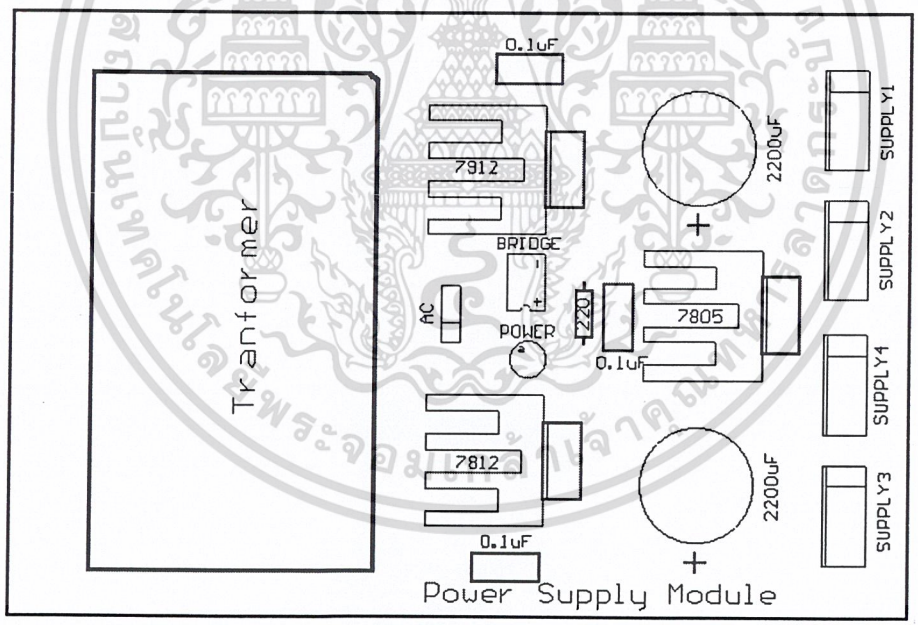
รูปที่ ข.19 วงจรพิมพ์ด้านล่างของการแสดงผลด้วยจอแบบผลึกเหลว



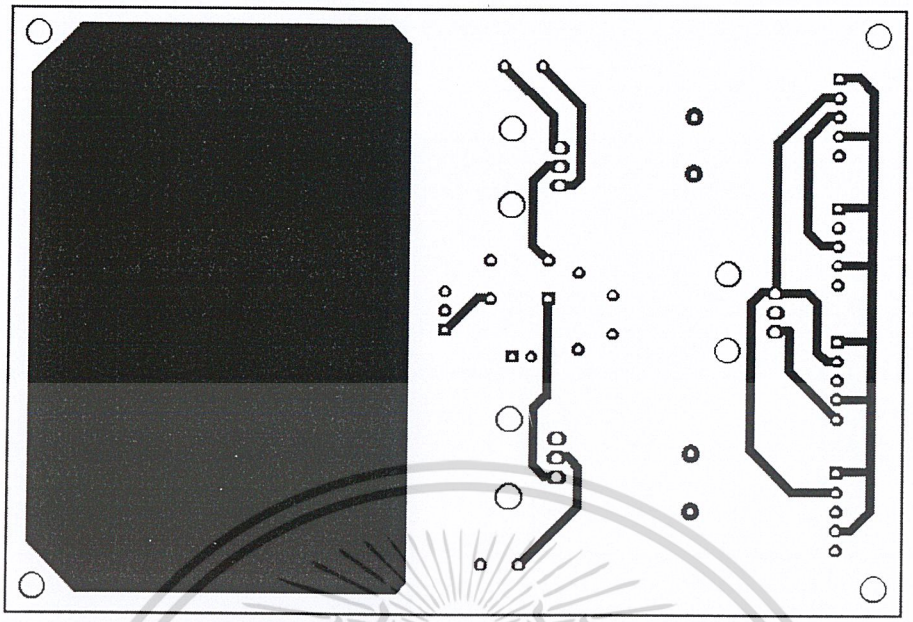
รูปที่ ข.20 วงจรพิมพ์ด้านบนของการแสดงผลด้วยจอแบบผลึกเหลว



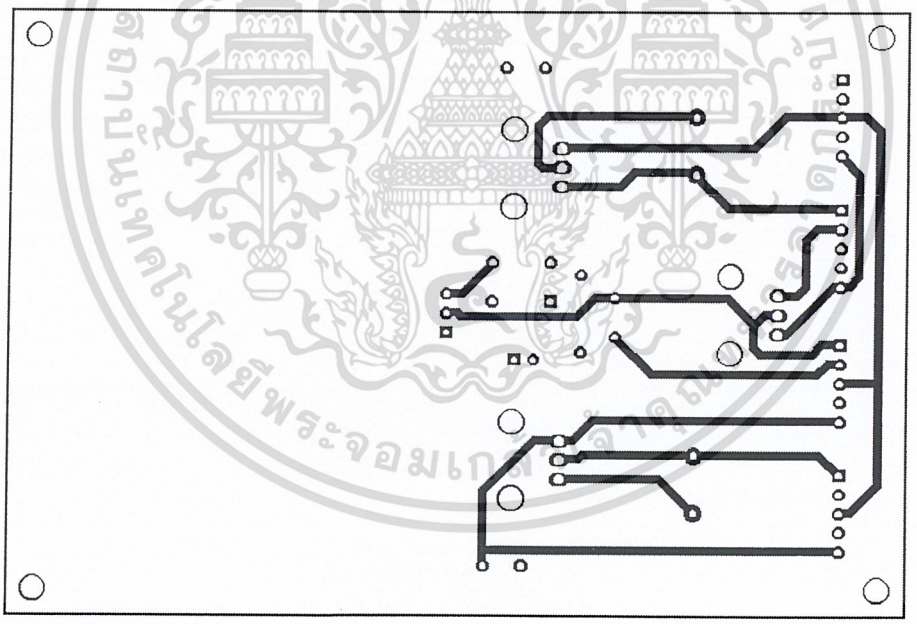
รูปที่ ข.21 วงจรภาคจ่ายไฟ



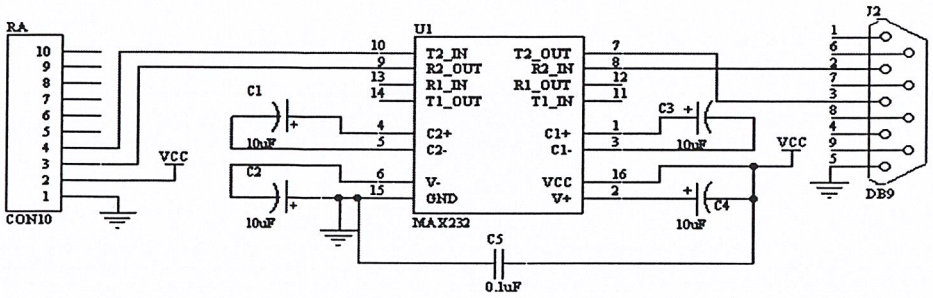
รูปที่ ข.22 การวางอุปกรณ์ภาคจ่ายไฟ



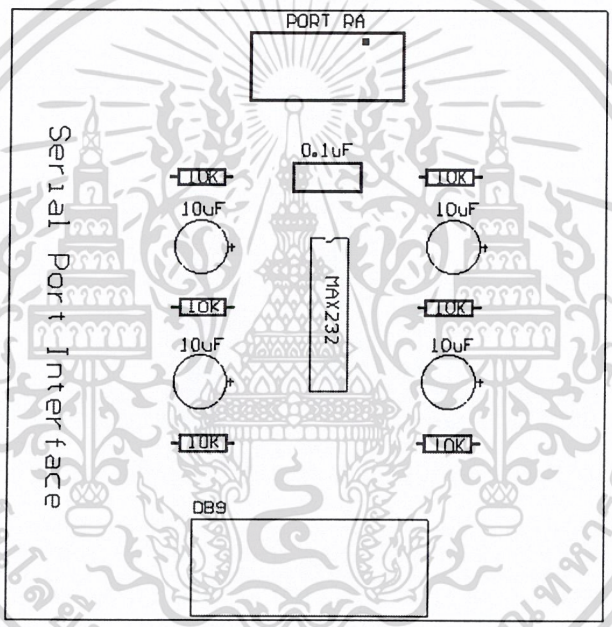
รูปที่ ข.23 วงจรพิมพ์ด้านล่างภาคจ่ายไฟ



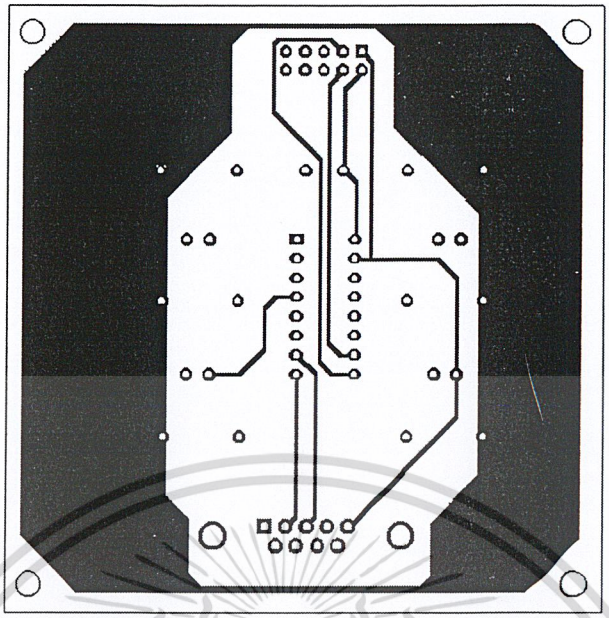
รูปที่ ข.24 วงจรพิมพ์ด้านบนภาคจ่ายไฟ



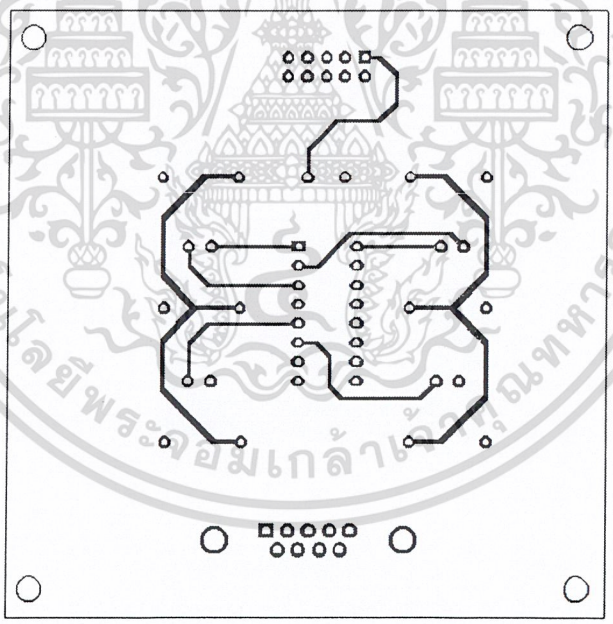
รูปที่ ข.25 วงจรของการเชื่อมต่อแบบพอร์ตอนุกรม



รูปที่ ข.26 การวางอุปกรณ์ของการเชื่อมต่อแบบพอร์ตอนุกรม



รูปที่ ข.27 วงจรพิมพ์ด้านล่างของการเชื่อมต่อแบบพอร์ตอนุกรม



รูปที่ ข.28 วงจรพิมพ์ด้านบนของการเชื่อมต่อแบบพอร์ตอนุกรม

ภาคผนวก ค
รายการอุปกรณ์

ตารางที่ ค.1 รายการอุปกรณ์ของวงจรหลัก PIC-ICSP

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC		
	PIC16F87x	1 ตัว
อุปกรณ์สารกึ่งตัวนำ		
Q1	BC547	2 ตัว
Q2	BC557	1 ตัว
ตัวเก็บประจุ		
C1	33pF	2 ตัว
C2	220pF	1 ตัว
C3	EA4000	1 ตัว
ตัวต้านทาน		
R1	10k	12 ตัว
อุปกรณ์อื่นๆ		
J1-J5	สล็อตเสียบสายแพรแบบ 10 ขา	5 ตัว
DB9	คอนเน็ตเตอร์ตัวผู้แบบลงวงจรพิมพ์	1 ตัว

ตารางที่ ค.2 รายการอุปกรณ์บองวงจรการแสดงผลด้วยจอแบบผลึกเหลว

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC	74LS541	1 ตัว
ตัวความต้านทาน		
R1	10K แบบปรับค่าได้	1 ตัว
R2	10K	1 ตัว
อุปกรณ์อื่นๆ		
J1-J2	สล็อตเสียบสายแพรแบบ 10 ขา	2 ตัว

ตารางที่ ค.3 รายการอุปกรณ์ของวงจรการเชื่อมต่อแบบพอร์ตอนุกรม

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC	MAX 232	1 ตัว
ตัวเก็บประจุ		
C1-C4	10μF	4 ตัว
ตัวความต้านทาน		
C1-C4	10K	4 ตัว
อุปกรณ์อื่นๆ		
DB9	คอนเน็ตเตอร์ ตัวผู้แบบลงวงจรพิมพ์	1 ตัว
J1-J2	สล็อตเสียบสายแพรแบบ 10 ขา	2 ตัว

ตารางที่ ค.4 รายการอุปกรณ์ของวงจรการเชื่อมต่อแบบไอสแควร์ซี

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC	DS1307	1 ตัว
ตัวเก็บประจุ		
C1-C2	10μF	2 ตัว
อุปกรณ์อื่นๆ		
J1-J3	สล็อตเสียบสายแพรแบบ 10 ขา	3 ตัว

ตารางที่ ค.5 รายการอุปกรณ์ของวงจรการรับอินพุตแบบเครื่องมือวัด

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC		
IC1	LM339	1 ตัว
IC2	ULN2065	1 ตัว
ตัวความต้านทาน		
R1	220Ω	3 ตัว
R2	50K	3 ตัว
R3	10K	5 ตัว
อุปกรณ์อื่นๆ		
J1-J2	สล็อตเสียบสายแพร	1 ตัว
OPTO1-OPTO3	ทรานซิสเตอร์ทำงานด้วยแสง	1 ตัว

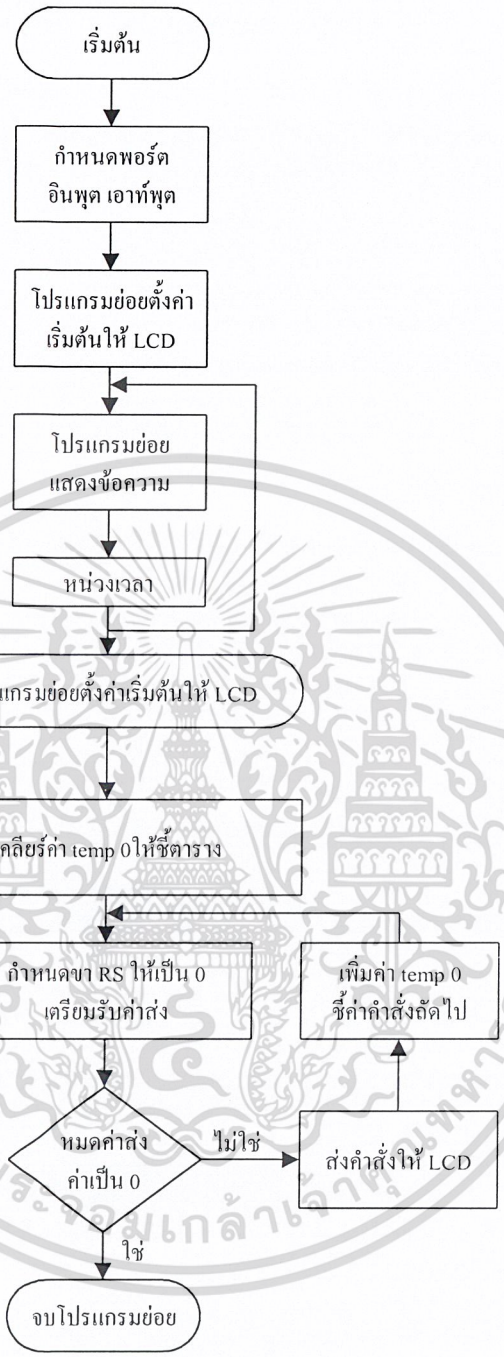
ตารางที่ ค.6 รายการอุปกรณ์ของการแสดงผลด้วยดอตเมตริกซ์ขนาด 24x8

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC		
IC1-IC3	74LS138	3 ตัว
IC4-IC5	74LS541	2 ตัว
สารกึ่งตัวนำ		
C1	BD140	24 ตัว
ตัวต้านทาน		
R1-R16	100	16 ตัว
R17-R24	220	24 ตัว
อุปกรณ์อื่นๆ		
J1-J3	สล็อตเสียบสายแพรขนาด 10 ขา	3 ตัว
DSP1-3	Dotmatrix 8*8	3 ตัว

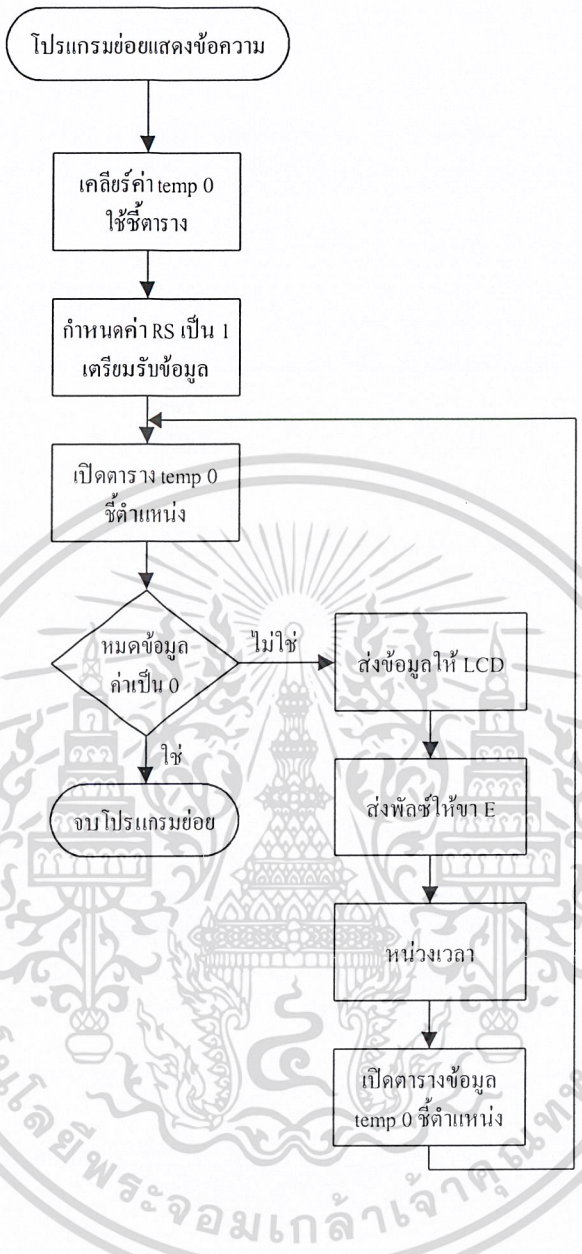
ตารางที่ ค.7 รายการอุปกรณ์ของวงจรภาคจ่ายไฟ

ชื่ออุปกรณ์	รายละเอียด	จำนวน
IC		
IC1	7805	1 ตัว
IC2	7812	1 ตัว
IC3	7912	1 ตัว
อุปกรณ์สารกึ่งตัวนำ		
D	บริดจ์ขนาด 1 แอมป์	1 ตัว
ตัวเก็บประจุ		
C1	0.1μF	3 ตัว
C2	2200μF/50V	2 ตัว
ตัวความต้านทาน		
R	220	1 ตัว
อุปกรณ์อื่นๆ		
J1-J4	สล็อตเสียบสายไฟขนาด 5 เส้น	4 ตัว

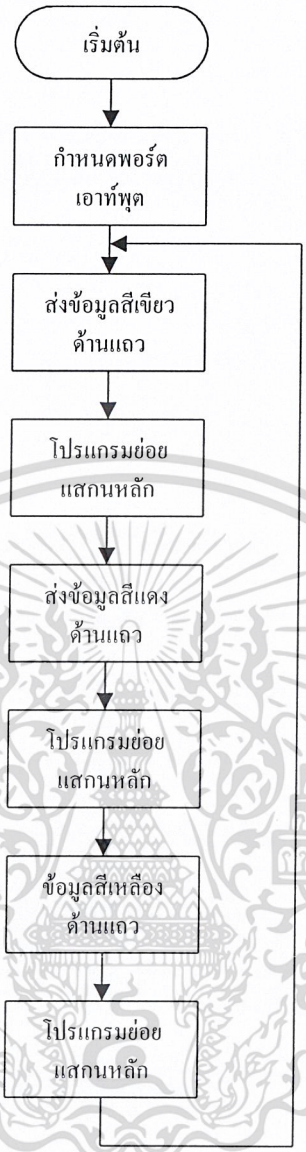
ภาคผนวก ง
แผนผังการทำงานและรหัสต้นฉบับของโปรแกรม



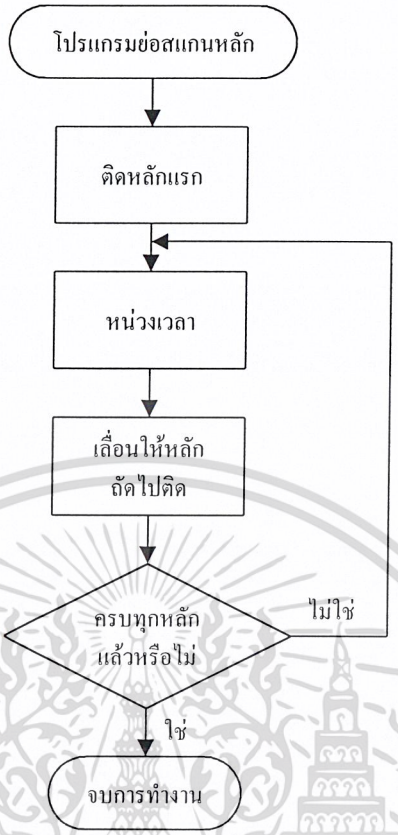
รูปที่ ง.1 ผังงานโปรแกรมของการแสดงผลด้วยจอแบบผลึกเหลว



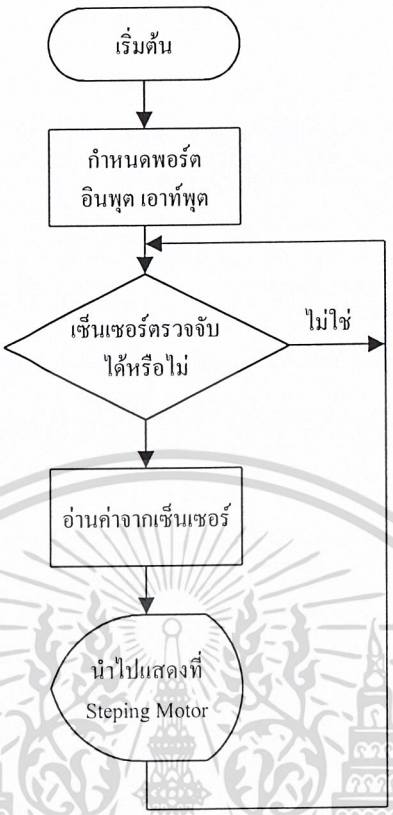
รูปที่ ง.1 (ต่อ) ผังงาน โปรแกรมของการแสดงผลด้วยจอแบบผลึกเหลว



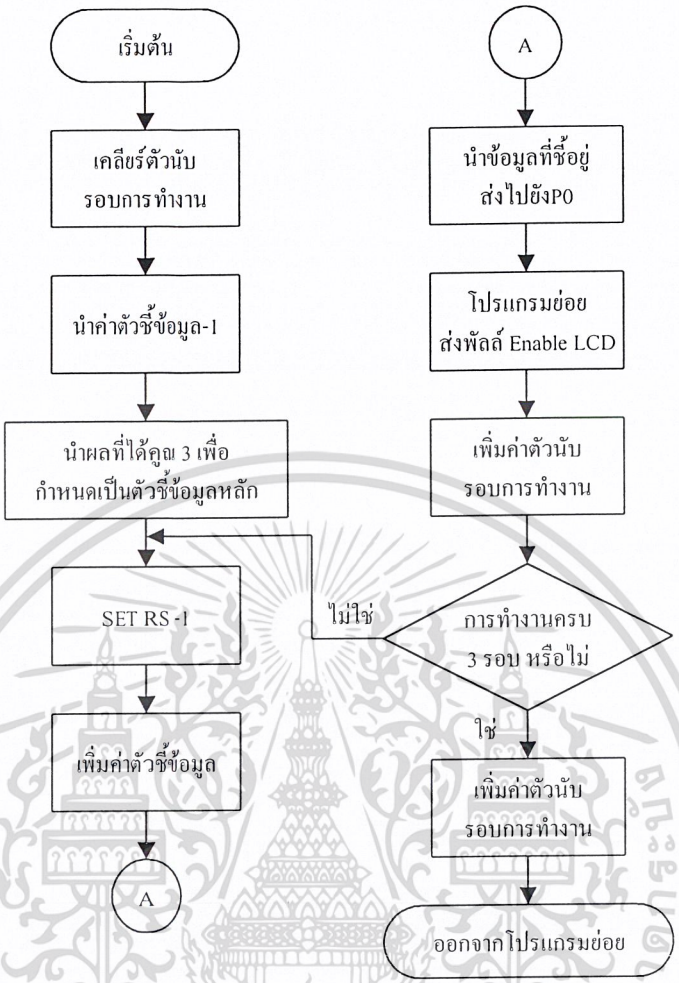
รูปที่ ง.2 แผนผังโปรแกรมของการแสดงผลด้วยดอทเมตริกซ์ขนาด 24x8



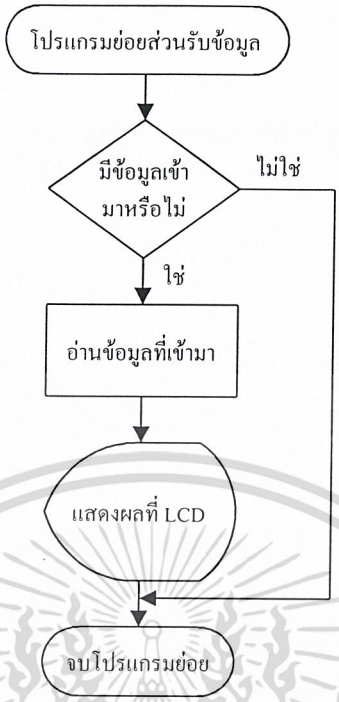
รูปที่ ง.2 (ต่อ) แผนผังโปรแกรมของการแสดงผลด้วยดอทเมตริกซ์ขนาด 24x8



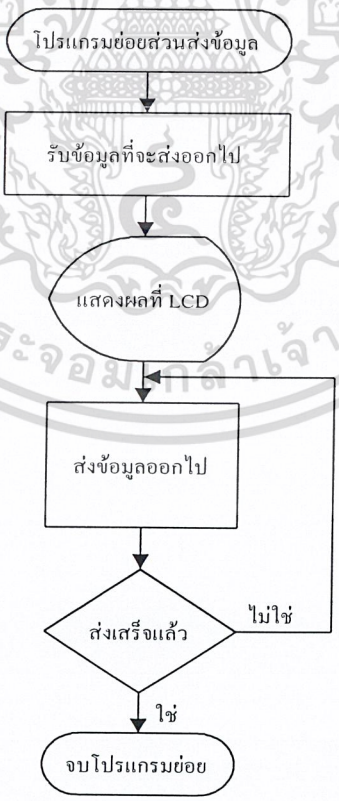
รูปที่ ง.3 แผนผังโปรแกรมของโมดูลอินพุตแบบเครื่องมือวัด



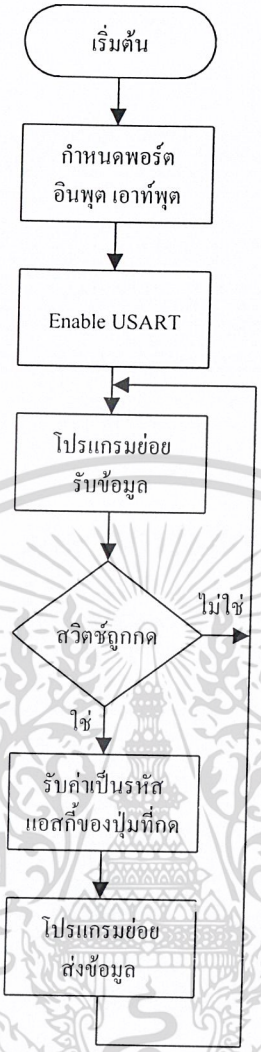
รูปที่ ง.4 แผนผังโปรแกรมของการเชื่อมต่อแบบไอสแควร์ซี



รูปที่ ง.5 แผนผังโปรแกรมของการเชื่อมต่อแบบพอร์ตอนุกรมในการรับข้อมูล



รูปที่ ง.6 แผนผังโปรแกรมของการเชื่อมต่อแบบพอร์ตอนุกรมในการส่งข้อมูล



รูปที่ ง.7 แผนผังโปรแกรมของการเชื่อมต่อแบบพอร์ตอนุกรม

โปรแกรมของการแสดงผลด้วยจอแบบผลึกเหลว

```
// LCD
// Port_B   lcd_control
// Port_C   lcd_data


#include "LCD.h"
#include "amPicLib.h"

  // lcd
#bit lcd_e  = port_B.0
#bit lcd_rs = port_B.1
#bit lcd_rw = port_B.2
#define lcd_data    port_C
#define lcd_direct(x) set_tris_c(x)
enum {rs_cmd, rs_data};
#define ON_CURSOR     0x0e
#define OFF_CURSOR    0x0c
#define CLEAR         0x01

int lcd_read_byte() {
int val;
  lcd_direct(0xff);
  lcd_rw = 1;
  delay_cycles(1);
  lcd_e = 1;
  delay_cycles(1);
  val = lcd_data;
  lcd_e = 0;
  lcd_direct(0x00);
  return val;
}

void lcd_send_byte(int rs, val) {
  lcd_rs = 0;
  while ( bit_test(lcd_read_byte(), 7) ) ;
  lcd_rs = rs;
  delay_cycles(1);
  lcd_rw = 0;
  delay_cycles(1);
  lcd_e = 0;
  lcd_data = val;
  delay_cycles(1);
  lcd_e = 1;
  delay_cycles(1);
  lcd_e = 0;
}

void lcd_init() {
  delay_ms(15);
  lcd_send_byte(rs_cmd, 0x38);
  delay_ms(1);
  lcd_send_byte(rs_cmd, 0x0c);
  lcd_send_byte(rs_cmd, 0x01);
```

```
  lcd_send_byte(rs_cmd, 0x06);
}

void lcd_gotoxy(int x, y) {
int addr;
  if (y!=1) addr = 0x40;
  else addr = 0;
  addr += x-1;
  lcd_send_byte(rs_cmd, 0x80|addr);
}

void lcd_putc(unsigned char ch) {
  lcd_send_byte(rs_data, ch);
}

void main() {
  set_tris_b(0x00);
  set_tris_c(0x00);
  port_b = 0;
  port_C = 0;

  lcd_init();

  lcd_putc("PIC16F877");
  lcd_gotoxy(1, 2);
  lcd_putc("Microcontroller");
  while (true);
}
```

โปรแกรมของการแสดงผลด้วยดอตเมตริกต์ขนาด 24x8

```
//Matrix
// Port B   Column
// Port C   Red
// Port D   Green
#include "matrix.h"
#include "amPicLib.h"
int mxRed[24];
int mxGrn[24];
#define Col   port_B
#define Red   port_C
#define Grn  port_D
// Font
byte const Font[43][5] = {
  {0x7C, 0x8A, 0x92, 0xA2, 0x7C}, // 0
  {0x00, 0x42, 0xFE, 0x02, 0x00}, // 1
  {0x42, 0x86, 0x8A, 0x92, 0x62}, // 2
  {0x84, 0x82, 0xA2, 0xD2, 0x8C}, // 3
  {0x18, 0x28, 0x48, 0xFE, 0x08}, // 4
  {0xE4, 0xA2, 0xA2, 0xA2, 0x9C}, // 5
  {0x3C, 0x52, 0x92, 0x92, 0x0C}, // 6
  {0x80, 0x80, 0x9E, 0xA0, 0xC0}, // 7
  {0x6C, 0x92, 0x92, 0x92, 0x6C}, // 8
```

```
  {0x60, 0x92, 0x92, 0x94, 0x78}, // 9
  {0x00, 0x6C, 0x6C, 0x00, 0x00}, // :
  {0x00, 0x6A, 0x6C, 0x00, 0x00}, // ;
  {0x10, 0x28, 0x44, 0x82, 0x00}, // <
  {0x28, 0x28, 0x28, 0x28, 0x28}, // =
  {0x00, 0x82, 0x44, 0x28, 0x10}, // >
  {0x40, 0x80, 0x8A, 0x90, 0x60}, // ?
  {0x7C, 0x82, 0xBA, 0xAA, 0x7A}, // @
  {0x7E, 0x90, 0x90, 0x90, 0x7E}, // A
  {0xFE, 0x92, 0x92, 0x92, 0x6C}, // B
  {0x7C, 0x82, 0x82, 0x82, 0x44}, // C
  {0xFE, 0x82, 0x82, 0x44, 0x38}, // D
  {0xFE, 0x92, 0x92, 0x92, 0x82}, // E
  {0xFE, 0x90, 0x90, 0x90, 0x80}, // F
  {0x7C, 0x82, 0x92, 0x92, 0x5E}, // G
  {0xFE, 0x10, 0x10, 0x10, 0xFE}, // H
  {0x00, 0x82, 0xFE, 0x82, 0x00}, // I
  {0x04, 0x02, 0x82, 0xFC, 0x80}, // J
  {0xFE, 0x10, 0x28, 0x44, 0x82}, // K
  {0xFE, 0x02, 0x02, 0x02, 0x02}, // L
  {0xFE, 0x40, 0x30, 0x40, 0xFE}, // M
  {0xFE, 0x20, 0x10, 0x08, 0xFE}, // N
  {0x7C, 0x82, 0x82, 0x82, 0x7C}, // O
  {0xFE, 0x90, 0x90, 0x90, 0x60}, // P
  {0x7C, 0x82, 0x8A, 0x84, 0x7A}, // Q
  {0xFE, 0x90, 0x98, 0x94, 0x62}, // R
  {0x64, 0x92, 0x92, 0x92, 0x4C}, // S
  {0x80, 0x80, 0xFE, 0x80, 0x80}, // T
  {0xFC, 0x02, 0x02, 0x02, 0xFC}, // U
  {0xF8, 0x04, 0x02, 0x04, 0xF8}, // V
  {0xFC, 0x02, 0x1C, 0x02, 0xFC}, // W
  {0xC6, 0x28, 0x10, 0x28, 0xC6}, // X
  {0xE0, 0x10, 0x0E, 0x10, 0xE0}, // Y
  {0x86, 0x8A, 0x92, 0xA2, 0xC2}  // Z
};

void ScanMatrix() {
int i;
 mxD = (mxD+1) % 24;
i = 23-mxD;
  Red = 0x00;
  Grn = 0x00;
  Col = (0x08<<(i/8)) + (i%8);
  Red = mxRed[mxD];
  Grn = mxGrn[mxD];
}

#int_RTCC
RTCC_isr() {
  ScanMatrix();
}

long Speed;
int Color, cor;
enum {cRed=1, cGreen=2, cOrange=3, cAlter=4};

TextShiftLeft(char ch) {
int i, j;
```

```
  switch (Color) {
    case cRed : cor = 1; break;
    case cGreen : cor = 2; break;
    case cOrange : cor = 3; break;
    case cAlter : cor = (cor%3)+1; break;
  }

  for (j=0; j<6; j++) {
    for (i=0; i<23; i++) {
      mxRed[i] = mxRed[i+1];
      mxGrn[i] = mxGrn[i+1];
    }
    if ( bit_test(cor, 0) && (j<5) && (ch!=' ') ) mxRed[23]=Font[ch-
'0'][j]; else mxRed[23]=0;
    if ( bit_test(cor, 1) && (j<5) && (ch!=' ') ) mxGrn[23]=Font[ch-
'0'][j]; else mxGrn[23]=0;
    delay_ms(Speed);
  }
}

void main() {
int i, j;
  set_tris_b(0x00);
  set_tris_c(0x00);
  set_tris_d(0x00);

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  enable_interrupts(INT_RTCC);
  enable_interrupts(global)
    mxRed[i] = 0;
    mxGrn[i] = 0;
  }

  Speed = 75;
  Color = cAlter;
  while (true) {
    TextShiftLeft("THE PIC16F87X MICROCONTROLLER ");
    TextShiftLeft("DEMONSTRATOR CONTROLED BY C LANGUAGE    ");
  }
}
```

โปรแกรมของการรับอินพุตแบบเครื่องมือวัด

```
#include "Instrument.h"
#include "amPicLib.h"

void main() {
int Direct=0;
int i=0;

  set_tris_c(0x0f);
  setup_adc_ports(NO_ANALOGS);
```

```
  setup_adc(ADC_CLOCK_DIV_2);

  while (1) {
    if (~bit_test(port_C, 1)) Direct = 2;
    if (~bit_test(port_C, 3)) Direct = 1;
    if (~bit_test(port_C, 2)) Direct = 0;

    switch (Direct) {
      case 0 :
        port_C &= 0x0f;
        i = 0;
        break;
      case 1 :
        i <<= 1;
        if (!i) i=0x10;
        port_C = (port_C & 0x0f)+(i&0xf0);
        delay_ms(20);
        break;
      case 2 :
        i >>= 1;
        if (!(i&0xf0)) i=0x80;
        port_C = (port_C & 0x0f)+(i&0xf0);
delay_ms(20);
        break;
    }
  }
  }
```

โปรแกรมของการเชื่อมต่อแบบไอสแควร์ซี

```
// I2C

#include "I2C.h"
#include "amPicLib.h"

//------------------------------ DS1307 RTC
#define i2c_SDA  PIN_B0
#define i2c_SCL  PIN_B1
#use i2c(master, sda=i2c_SDA, scl=i2c_SCL)

void init_i2c() {
  output_float(i2c_SCL);
  output_float(i2c_SDA);
}

void write_DS1307 (byte addr, data) {
  i2c_start();
  i2c_write(0xd0);
  i2c_write(addr);
  i2c_write(data);
  i2c_stop();
  delay_ms(8);
}
```

```
int read_DS1307 (byte addr) {
int data;
  i2c_start();
  i2c_write(0xd0);
  i2c_write(addr);
  i2c_start();
  i2c_write(0xd1);
  data = i2c_read(0);
  i2c_stop();
  return(data);}
// Key ------------------
enum {swEnter=1, swUp=2, swDown=4};
unsigned char keyNow=0, wKey=0;
  // Segme----------------
int segData[5];
int segDigit=0;
int const number[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f};  // ตัวเลข 0-9

//========================= declaration
  // system
int sysclk=0, sysclk2=0;


// parameter
enum {modeTime, modeSet};
int NowMode;
int seg_rate=0;
short seg_blink;
int const maxTime[2] = {0x23, 0x59};
int const minTime[2] = {0x00, 0x00};
enum {adrSecond, adrMinute, adrHour, adrDay, adrDate, adrMonth,
adrYear};
int buffTime[2], adrTime;
//========================= Task

void SetMode(int mode) {
int i;
  NowMode = mode;
  switch (NowMode) {
    case modeTime :
      break;
    case modeSet :
      adrTime = 0;
      buffTime[0] = read_DS1307(adrHour);
      buffTime[1] = read_DS1307(adrMinute);
      break;
  }
}

void Seg_Paint() {
int i;
  seg_blink = ~seg_blink;
  switch (NowMode) {
    case modeTime :
      i = read_DS1307(adrHour);
      segData[0] = number[i>>4];
```

```
      segData[1] = number[i&15];
      i = read_DS1307(adrMinute);
      segData[2] = number[i>>4];

      segData[3] = number[i&15];
      if (seg_blink) segData[1]|=0x80;
      break;
  case modeSet:

      i = buffTime[0];
      segData[0] = number[i>>4];
      segData[1] = number[i&15] |= 0x80;
      i = buffTime[1];
      segData[2] = number[i>>4];
      segData[3] = number[i&15];
      if (!seg_blink) {
        segData[2*adrTime] = 0;
        segData[(2*adrTime)+1] = 0;
      }
      segData[1] |= 0x80;
      break;
  }
}
//========================== Events Operating


void Key_OnDown() {
int i;
  switch (NowMode) {
      case modeSet :
            switch (keyNow) {
                  case swEnter : adrTime = (adrTime+1) % 2;
goto MST1;
                        case swUp :
          i = buffTime[adrTime];
          if (i<maxTime[adrTime]) {
            i++;
            if ((i&0x0f)>9) i+=6;
          }
          else i = minTime[adrTime];
          buffTime[adrTime] = i;
          goto MST1;
        case swDown :
          i = buffTime[adrTime];
          if (i>minTime[adrTime]) {
            i--;
            if ((i&0x0f)>9) i-=6;
          }
          else i = maxTime[adrTime];
          buffTime[adrTime] = i;
MST1 :
                        seg_blink = 0;
                        seg_rate = 1;
                        break;
                  }
                  break;
      }
```

```
}
void Key_OnClick() {

}
void Key_OnFPress() {
int i;
switch (NowMode) {
      case modeTime :
      if (keyNow==swEnter) SetMode(modeSet);   // เข้าสู่โหมดตั้งเวลา
                  break;
            case modeSet :
                        // บันทึกวันเวลาที่ตั้งใหม่
              if (keyNow==swEnter) {
        write_DS1307(adrHour, buffTime[0]);
        write_DS1307(adrMinute, buffTime[1]);
                  SetMode(modeTime);
            }
      if ((keyNow==swUp)||(keyNow==swDown)) {
        Key_OnDown();
        wKey=5;
      }
                  break;
      }
}
void Key_OnUp() {}//========================== Create Events
void Scan_Key() {              // สร้างเหตุการณ์ กดปุ่ม
unsigned char k;
  k = (~Port_A) & 7;
  if (k != keyNow) {
    if (keyNow) {
      if (wKey<50) {
        if (wKey) Key_OnClick();
        Key_OnUp();
        keyNow = 0;
      }
      wKey = 0;
    }
    if (k) {
      keyNow = k;
      wKey = 52;                  // 1 Sec
    }
  }
}
void Scan_KeyFPress() {        // สร้างเหตุการณ์กดปุ่มแช่
  if (wKey) {
    wKey--;
    if (wKey==50) Key_OnDown();
    if (!wKey) Key_OnFPress();
  }
void Scan_Seg_Paint() {        // refresh seg ทุกๆ 500 ms
  if (seg_rate) seg_rate--;
  if (!seg_rate) {
      seg_rate = 25;                  // 500ms
      Seg_Paint();
    }
}
```

```
void Scan_Segment() {
  bit_clear(port_B, 4+segDigit);
  segDigit = (segDigit+1) % 4;
  port_C = segData[3-segDigit];
  bit_set(port_B, 4+segDigit);
}
//=========================== Multitasking
Task_successive() {
  Scan_Key();
}
Task_interval20ms() {
  Scan_KeyFPress();
  Scan_Seg_Paint();
}

Task_interval500ms() {

}

//============================ isr

#int_RTCC
RTCC_isr() {
  sysclk++;
  Scan_Segment();
}

//============================ Main

void main() {
int i;
  set_tris_b(0x0f);
  set_tris_c(0x00);
  port_b_pullups(TRUE);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_4);
  enable_interrupts(INT_RTCC);
  enable_interrupts(global);
  while (true) {
    Task_successive();
    if (sysclk>=20) {
      sysclk %= 20;
      Task_interval20ms();
      if ((++sysclk2)>=25) {
        sysclk2 = 0;
        Task_interval500ms();
      }
    }
  }
}
```

โปรแกรมของการเชื่อมต่อแบบพอร์ตอนุกรม

```
//PORT RC
#include "Serial.h"
#int_RDA
RDA_isr() {
char c;
  c = getc();
  printf("You press key : %c", c);
  putc(13);
  putc(10);
}
void main() {
  set_tris_a(0xfe);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  enable_interrupts(INT_RDA);
  enable_interrupts(global);

  delay_ms(500);
  printf("Serial UART Communication <baudrate 9600 bps ");
  putc(13);
  putc(10);
  printf("For PIC 16F87X Microcontroller Trianning System");
  putc(13);
  putc(10);
  putc(10);
  printf("Press The Character On your Keyboard ");
  putc(13);
  putc(10);
  putc(10);
  while (1);
}
```

ภาคผนวก จ

ใบงาน

ใบงานที่ 1

การเชื่อมต่อ PIC 16F87X กับการแสดงผลด้วยจอแบบผลึกเหลว

วัตถุประสงค์เชิงพฤติกรรม

เพื่อให้นักศึกษาสามารถ

1. ใช้งาน LCD แบบตัวอักษรหรือแบบอักขระได้
2. เขียนโปรแกรมเพื่อควบคุมให้ PIC16F877 แสดงผลออกทาง LCD แบบตัวอักษรหรือแบบอักขระได้
3. ใช้งาน LCD ในโหมด 4 บิตและ 8 บิตได้

อุปกรณ์ในการทดลอง

1. โมดูลหลัก PIC-ICSP
2. โมดูลแสดงผลแบบผลึกเหลว (LCD)
3. คอมพิวเตอร์พีซีที่ติดตั้งโปรแกรม PCW Compiler
4. สายเชื่อมต่อระหว่างโมดูลหลัก PIC-ICSP กับคอมพิวเตอร์พีซี
5. สายเชื่อมต่อพอร์ตระหว่างโมดูล

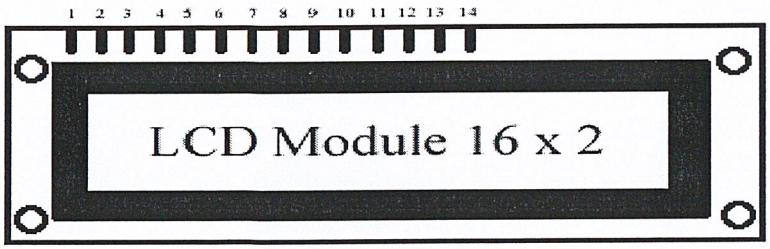
ทฤษฎีเบื้องต้น

จอแสดงผลแบบผลึกเหลวหรือ LCD (Liquid Crystal Display) แบ่งออกเป็น 3 แบบตามลักษณะการแสดงผลคือ LCD แบบอักขระ (Character LCD Module), LCD แบบกราฟฟิก (Graphic LCD Module) และ LCD แบบเซกเมนต์ (Segment LCD Module)

LCD แบบอักขระเป็น LCD ที่สามารถแสดงตัวอักษร ตัวเลขและเครื่องหมายต่างๆได้โดยสร้างจากจุดเล็กๆ ทางแนวตั้งและแนวนอนหรือ dot matrix โดยทั่วไปจะมี 2 ขนาดคือ 5x7 จุดและ 5x10 จุด

LCD แบบกราฟฟิกมีลักษณะโครงสร้างคล้ายกับ LCD แบบอักขระ สามารถแสดงข้อมูลเป็นทั้งแบบตัวอักขระ ตัวเลข เครื่องหมายและรูปภาพได้ ความละเอียดของภาพก็จะขึ้นอยู่กับความละเอียดของ dot matrix ของ LCD ตัวนั้นๆของขนาด LCD แบบนี้มีอยู่หลายขนาดให้เลือกใช้

LCD แบบเซกเมนต์ (Segment) เป็นแบบเล็กสุดมีลักษณะการแสดงผลเป็นเซกเมนต์คล้ายกับ LED ตัวเลข 7 ส่วนโดยปกติมักจะมีมากกว่า 1 หลัก



รูปที่ จ.1 รูปร่างและการจัดขาของจอแสดงผลแบบผลึกเหลวแบบอักขระ

LCD ขนาด 16 ตัวอักษรจำนวน 2 บรรทัด (LCD 16x2)

LCD ที่เราใช้ในการทดลองนี้เป็น LCD อีกตัวที่มีโครงสร้างเป็นมาตรฐานมีขาที่ต่อใช้งาน 16 ขา มีการจัดขาดังรูปที่ 1.1 รายละเอียดของแต่ละขามีดังนี้

Vss (ขา 1) = ต่อกราวด์

Vcc (ขา 2) = ต่อไฟเลี้ยง 5 โวลต์

Vo (ขา 3) = เป็นขาอินพุต รับแรงดันเพื่อรับความเข้มของการแสดงผล

RS (ขา 4) = เป็นขาอินพุต ใช้ในการรับชนิดของข้อมูล ถ้าขานี้เป็น “0” ข้อมูลที่ส่งมาจะเป็นคำสั่งแต่ถ้าเป็น “1” ข้อมูลนี้จะเป็นข้อมูลสำหรับแสดงผล

R/W (ขา 5) = เป็นขาที่ใช้เลือกการอ่านหรือเขียนข้อมูลกับ LCD ถ้าเป็น “0” เป็นการกำหนดให้เขียนข้อมูลแต่ถ้าเป็น “1” จะเป็นการอ่านข้อมูล

E (ขา 6) = เป็นอินาเบิล LCD ให้ทำงาน

D0-D7 (ขา 7-14) = เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่าง LCD กับอุปกรณ์

ไฟ Back Light ขา 15-16 เป็นขาที่ใช้ในการเพิ่มความสว่างของ LCD

ขา R/W, RS และขา E จะใช้งานร่วมกันดังตารางการทำงานที่ 10.1

ตารางที่ จ.1 ความสัมพันธ์ของการทำงานร่วมกันของขา RS, R/W และ E

RS	R/W	E	การทำงาน
0	0		เขียนคำสั่ง
0	1		อ่านสถานะของ LCD
1	0		เขียนข้อมูล
1	1		อ่านข้อมูล

คำสั่งควบคุม LCD

การเขียนคำสั่งลงในตัวควบคุม จะต้องกำหนดให้ RS, R/W และขา E จะต้องเป็น “0” แล้วเขียนคำสั่งตามไป คำสั่งในการควบคุมมี 10 คำสั่งดังนี้

1. คำสั่งเคลียร์ตัวแสดงผล (Clear display) มีข้อมูลเป็นคำสั่ง 0x01 เป็นคำสั่งที่ใช้ในการเขียนข้อมูลช่องว่างหรือ space เข้าไปใน DDRAM ทั้งหมด เมื่อตัวควบคุมเอ็กซิคิ้วต์คำสั่งนี้จะทำการกำหนดแอดเดรสของ DDRAM เป็น 0 เคอร์เซอร์จะกลับไปอยู่ที่ ตำแหน่งซ้ายมือสุดของจอแสดงผล แล้วเซตบิต I/D ให้เป็น “1”

2. คำสั่ง Return Home ต้องกำหนดให้บิต “1” ของข้อมูลเป็น “1” เป็นคำสั่งให้เคอร์เซอร์เคลื่อนที่กลับไปยังตำแหน่งซ้ายสุดของจอแสดงผล แต่ข้อมูลบนตัวแสดงผลไม่เปลี่ยนแปลงนั่นคือข้อมูลของคำสั่งนี้จะเป็น 0x02 หรือ 0x03 ก็ได้

3. คำสั่งเลือกโหมดการป้อนกลับ (Entry mode set) มีรายละเอียดดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	0	0	1	I/D	S

รูปที่ จ.2 คำสั่งเลือกโหมดการป้อนกลับ

บิต S เป็นบิตที่ใช้ในการกำหนดลักษณะของการแสดงผลเมื่อมีการป้อนข้อมูล ถ้าหากบิต S เป็น “1” เมื่อเกิดข้อมูลใหม่บนจอแสดงผล ตัวเคอร์เซอร์จะอยู่กับที่แต่ตัวอักษรข้อมูลเดิมจะถูกดันไปทางซ้าย แต่ถ้าหากบิตนี้เป็น “0” เมื่อเกิดข้อมูลใหม่ตัวเคอร์เซอร์จะเลื่อนไปทางขวามือ

บิต I/D เป็นบิตที่ใช้ในการกำหนดว่าเมื่อเขียนหรืออ่านข้อมูลแล้ว ทำให้แอดเดรสของ DDRAM เพิ่มขึ้นหรือลดลงหนึ่งแอดเดรสโดยถ้าบิตนี้เป็น “1” แอดเดรสของ DDRAM จะเพิ่มขึ้น แต่ถ้าเป็น “0” แอเดรสจะลดลง

4. คำสั่งควบคุมการแสดงผล มีรายละเอียดดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	0	1	D	C	B

รูปที่ จ.3 คำสั่งควบคุมการแสดงผล

บิต D ใช้ควบคุมการเปิดปิดจอแสดงผล ถ้าเป็น “0” จะเป็นการปิดจอแสดงผล

บิต C ใช้ควบคุมการแสดงตัวเคอร์เซอร์บนจอแสดงผล ถ้าต้องการให้มีเคอร์เซอร์แสดงผล ต้องกำหนดให้บิตนี้เป็น “1” ถ้ากำหนดให้เป็น “0” จะเป็นการเปิดเคอร์เซอร์หรือไม่แสดงเคอร์เซอร์

บิต B ใช้ควบคุมการกระพริบของเคอร์เซอร์ถ้าบิตเป็น “1” เคอร์เซอร์จะกระพริบ

5. คำสั่งควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร มีรายละเอียดดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	**0**	**0**	**1**	**S/C**	**R/L**	*	*

รูปที่ จ.4 การควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร

การควบคุมการเลื่อนเคอร์เซอร์และตัวอักษรบนจอแสดงผล ขึ้นอยู่กับการกำหนดบิต S/C และ R/L สรุปได้ดังนี้

ตารางที่ จ.2 การควบคุมการเลื่อนเคอร์เซอร์บนจอแสดงผล

S/C	R/L	ลักษณะการเลื่อนเคอร์เซอร์	ข้อมูลคำสั่ง
0	0	ไปทางซ้าย	0x10-0x13
0	1	ไปทางขวา	0x14-0x17
1	0	ตัวอักษรใหม่ไปทางซ้าย	0x18-0x1B
1	1	ตัวอักษรใหม่ไปทางขวา	0x1C-0xF

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	**0**	**1**	**DL**	**N**	**F**	*	*

รูปที่ จ.5 การกำหนดฟังก์ชั่นการทำงาน

คำสั่งกำหนดฟังก์ชั่นการทำงาน ดังรูปที่ 1.5 มีรายละเอียดดังนี้

- บิต DL ใช้กำหนดจำนวนบิตที่ใช้ติดต่อส่งผ่านข้อมูลถ้าบิตนี้เป็น “0” จะเป็นการติดต่อแบบ 4 บิต แต่ถ้าเป็น “1” จะเป็นการติดต่อแบบ 8 บิต

- บิต N ใช้กำหนดจำนวนบรรทัดของการแสดงผล ถ้าเป็น “0” จะแสดงผล 1 บรรทัด ถ้าเป็น “1” จะแสดง 2 บรรทัด หรือในกรณีที่จอแสดงผลสามารถแสดงได้มากกว่า 2 บรรทัด

- บิต F ใช้เลือกความละเอียดของตัวอักษรในการแสดงผล ถ้าบิตนี้เป็น “0” จะเป็นการแสดงผลแบบ 5x7 จุด และถ้าเป็น “1” จะแสดงผลเป็นแบบ 5x10 จุด

6. คำสั่งเลือกแอดเดรสของ CGRAM เมื่อต้องการกำหนดแอดเดรสของ CGRAM ต้องกำหนดให้บิต 7 เป็น “0” บิต 6 เป็น “1” ส่วนอีก 6 บิตที่เหลือจะแทนด้วยค่าแอดเดรสของ CGRAM จะต้องทำการกำหนดแอดเดรสด้วยคำสั่งนี้ก่อนที่จะเขียนหรืออ่านข้อมูลให้ CGRAM โดยแอดเดรสจะอยู่ระหว่าง 0x00-0x3F

7. คำสั่งเลือกแอดเดรสของ DDRAM ก่อนที่จะทำการอ่านหรือเขียนข้อมูลโดยบิต 7 ต้องเป็น “1” และข้อมูลบิต 7 ที่เหลือ เป็นค่าแอดเดรสของ DDRAM ซึ่งแอดเดรสจะอยู่ระหว่าง 0x8C-0xFF นอกจากนี้จำนวนแอดเดรสยังขึ้นอยู่กับการกำหนดสถานะที่บิต N หากบิต N เป็น “0” แอดเดรสจะอยู่ระหว่าง 0x80-0xCF และถ้าบิต N เป็น “1” แอดเดรสจะมี 2 ช่วงคือ 0x8C-0x87 และ 0xC0-0xC7

8. คำสั่งอ่าน Busy และแอดเดรส มีรายละเอียดดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
BF	**A**	**A**	**A**	**A**	**A**	**A**	**A**

รูปที่ จ.6 การอ่าน Busy และแอดเดรส

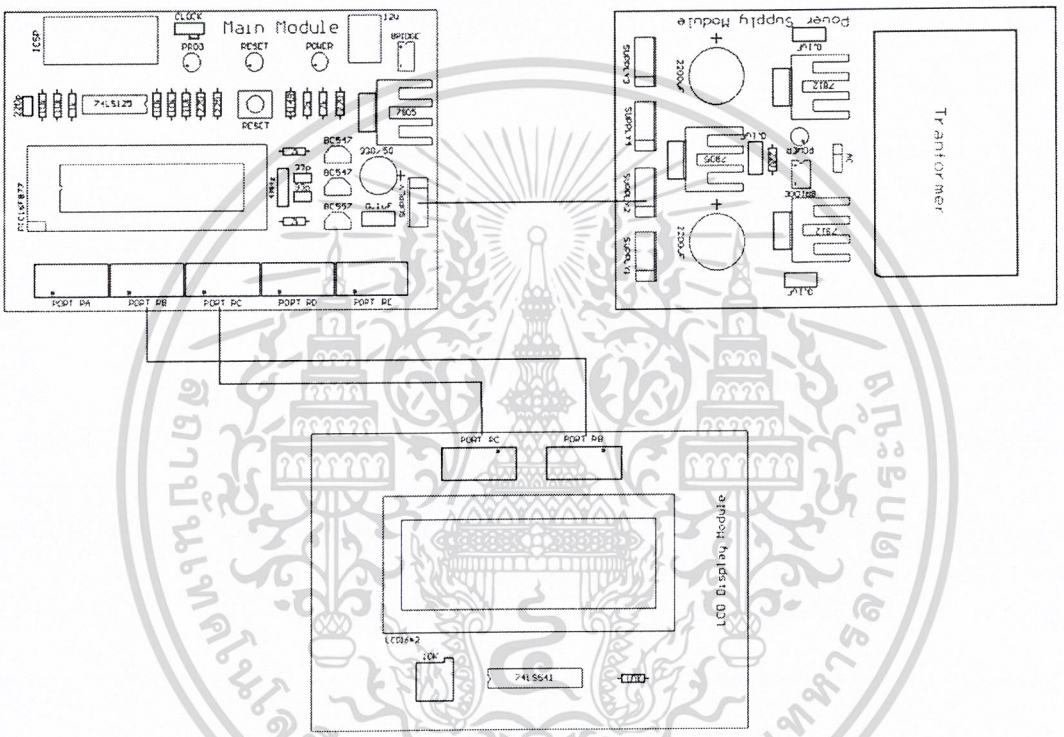
จะเป็นแฟลกบอกสถานะของตัวควบคุม LCD ว่าพร้อมจะรับข้อมูลอยู่หรือไม่ ถ้าหากบิต BF เป็น “0” แสดงว่าตัวควบคุม LCD พร้อมจะรับข้อมูลหรือคำสั่ง แต่ถ้าเป็น “1” แสดงว่าตัวควบคุม LCD ยังอยู่ในกระบวนการทำงานภายในหรือกำลังประมวลผลข้อมูลอยู่ไม่พร้อมที่จะรับข้อมูลหรือคำสั่ง

เมื่อต้องการอ่านแฟลกต้องกำหนดให้ขา R/W เป็น “1” ด้วย แต่สัญญาณที่ RS ยังต้องเป็น “0” อยู่เพราะข้อมูลนี้ยังเป็นข้อมูลคำสั่ง

จังหวะการทำงานของ LCD

การใช้งาน LCD ต้องเขียนโปรแกรมเพื่อหน่วงเวลาให้ LCD พร้อมที่จะทำงาน โดยเมื่อเริ่มจ่ายแรงดันให้กับ LCD ต้องรอ 5 มิลลิวินาที เพื่อให้ LCD เตรียมความพร้อมหรือ Initial หลังจาก

นั้นจะต้องกำหนดลอจิกขา RS ของ LCD แล้วต้องหน่วงเป็นเวลาประมาณ 125 ไมโครวินาที เพื่อให้ไมโครคอนโทรลเลอร์ใน LCD แปลความหมายของลอจิกที่ขา RS ว่าข้อมูลต่อไปที่ได้รับเป็นรหัสคำสั่งหรือข้อมูลที่ต้องการแสดงผล จากนั้นจะเป็นการส่งข้อมูลมารอที่บัสข้อมูล D0-D7 (กรณีทำงานในโหมด 8 บิต) ต่อไปจะเป็นการส่งสัญญาณพัลส์ไปที่ขา E เพื่ออินาเบิล LCD ให้รับข้อมูลจากบัส (Bus) ข้อมูลเข้าไปโดยพัลส์ที่ป้อนเข้าที่ขา E ของ LCD ต้องเป็นพัลส์ขอบขาขึ้นจากนั้นทำการหน่วงเวลา 125 ไมโครวินาที



รูปที่ จ.7 การเชื่อมต่อ PIC16F877 กับโมดูล LCD

การต่อ PIC16F876 กับ LCD

การต่อ PIC16F876 กับ LCD แบบอักขระ 8 บิต เราจะกำหนดให้พอร์ต A เป็นขาของการควบคุม โดย RA1 ต่ออยู่กับขา E, RA2 ต่ออยู่กับขา RS และ RA3 ต่ออยู่ กับ R/W ส่วนพอร์ต B จะต่ออยู่กับขาข้อมูล D0-D7 ทั้ง 8 เส้น

การใช้งาน LCD แบบ 4 บิต

การต่อขาใช้งาน LCD แบบ 4 บิตนั้นที่ขาข้อมูลเราจะใช้เพียง 4 บิตบนเท่านั้น ส่วนการโปรแกรม ถ้าทำงานในแบบ 8 บิตเมื่อป้อนข้อมูลให้ LCD แล้วต้องป้อนสัญญาณพัลส์ให้แก่ขา E แล้วตามด้วยการหน่วงเวลา 125 ไมโครวินาที แต่ถ้าทำงานแบบ 4 บิตต้องแยกส่งข้อมูลทีละ 4 บิต

โดยเริ่มจากส่งข้อมูล 4 บิตบนก่อนและส่งสัญญาณพัลส์ที่ขา E จากนั้นจึงส่งข้อมูล 4 บิตล่างและส่งสัญญาณพัลส์ที่ขา E ตามด้วยการหน่วงเวลา 125 ไมโครวินาที

```
// LCD
// Port_B   lcd_control
// Port_C   lcd_data
#include "LCD.h"
#include "amPicLib.h"
 // lcd
#bit lcd_e  = port_B.0
#bit lcd_rs = port_B.1
#bit lcd_rw = port_B.2
#define lcd_data    port_C
#define lcd_direct(x) set_tris_c(x)
enum {rs_cmd, rs_data};
#define ON_CURSOR      0x0e
#define OFF_CURSOR     0x0c
#define CLEAR          0x01
int lcd_read_byte() {
int val;
  lcd_direct(0xff);
  lcd_rw = 1;
  delay_cycles(1);
  lcd_e = 1;
  delay_cycles(1);
  val = lcd_data;
  lcd_e = 0;
  lcd_direct(0x00);
  return val;
}

void lcd_send_byte(int rs, val) {
  lcd_rs = 0;
  while ( bit_test(lcd_read_byte(), 7) ) ;
  lcd_rs = rs;
  delay_cycles(1);
  lcd_rw = 0;
  delay_cycles(1);
  lcd_e = 0;
  lcd_data = val;

  delay_cycles(1);
  lcd_e = 1;
  delay_cycles(1);
  lcd_e = 0;
}
```

รูปที่ จ.8 โปรแกรมการแสดงผลของLCD 8 บิต

```
void lcd_init() {
  lcd_rs = 0;
  delay_ms(15);
  lcd_send_byte(rs_cmd, 0x38);
  delay_ms(1);
  lcd_send_byte(rs_cmd, 0x0c);
  lcd_send_byte(rs_cmd, 0x01);
  lcd_send_byte(rs_cmd, 0x06);
}

void lcd_gotoxy(int x, y) {
int addr;
  if (y!=1) addr = 0x40;
  else addr = 0;
  addr += x-1;
  lcd_send_byte(rs_cmd, 0x80|addr);
}

void lcd_putc(unsigned char ch) {
  lcd_send_byte(rs_data, ch);
}

void main() {
  set_tris_b(0x00);
  set_tris_c(0x00);
  port_b = 0;
  port_C = 0;

  lcd_init();

  lcd_putc("PIC16F877");
  lcd_gotoxy(1, 2);
  lcd_putc("Microcontroller");
  while (true);
}
```

รูปที่ จ.8 (ต่อ) โปรแกรมการแสดงผลของLCD 8 บิต

ลำดับขั้นตอนการทดลอง

1. ทำการเปิดโปรแกรม PCW สร้างไฟล์ใหม่ขึ้นมาเป็นนามสกุล .C
2. ทำการเขียนโปรแกรมตามรูปที่ จ.8
3. ทำการคอมไพล์แบบ 14 บิท แล้วจะได้ไฟล์ที่เป็น .HEX มาใช้งาน
4. เปิดโปรแกรม IC-Prog เปิดไฟล์ที่สร้างไว้ แล้วทำการเบิร์นข้อมูลลงPIC 16F877 กดปุ่มRESET เพื่อทำการ RUN โปรแกรมที่เขียนไว้
5. สังเกตุผลที่ได้บนหน้าจอ แอลซีดี

.....

.....

.....

7.จากข้อ 6 ถ้าต้องการเปลี่ยนข้อความที่แสดงบนหน้าจอ แอลซีดี จะต้องเปลี่ยนโปรแกรมในส่วนใด

.....

.....

.....

.....

.....

8. ถ้าต้องการให้ข้อความที่แสดงบนหน้าจอ แอลซีดี กระพริบได้ จะต้องเขียนโปรแกรมเพิ่มอย่างไร

.....

.....

.....

.....

.....

สรุปผลการทดลอง

.....

.....

.....

.....

.....

คำถามท้ายการทดลอง

1. จงอธิบายหลักการทำงานของโปรแกรมดังรูปที่ จ.8

2. เขียนโปรแกรมแสดงชื่อของท่านเอง ในบรรทัดที่1 และนามสกุลในบรรทัดที่ 2 และให้ตัวอักษรนั้นกระพริบด้วย บนหน้าจอ แอลซีดี

ใบงานที่ 2
การสื่อสารข้อมูลกับคอมพิวเตอร์ผ่านพอร์ตอนุกรม RS-232

วัตถุประสงค์เชิงพฤติกรรม

เพื่อให้นักศึกษาสามารถ

1. อธิบายหลักการสื่อสารข้อมูลกับคอมพิวเตอร์แบบอนุกรม RS-232ได้
2. ใช้งานพอร์ตการสื่อสารแบบอนุกรมของไมโครคอนโทรลเลอร์ PIC16F877 ได้
3. เชื่อมต่อไมโครคอนโทรลเลอร์ PIC16F877 เข้ากับกับคอมพิวเตอร์ผ่านพอร์ตอนุกรมมาตรฐาน RS-232ได้

เครื่องมือและอุปกรณ์

1. โมดูลหลัก PIC-ICSP
2. โมดูลสื่อสารข้อมูลอนุกรม
3. คอมพิวเตอร์พีซีที่ติดตั้งโปรแกรม PCW Compiler และโปรแกรมสื่อสารข้อมูลอนุกรม HYper Terminal
4. สายเชื่อมต่อระหว่างคอมพิวเตอร์พีซีและโมดูลสื่อสารข้อมูลอนุกรม
5. สายเชื่อมต่อพอร์ตระหว่างโมดูล

ทฤษฎีเบื้องต้น

RS-232 นั้นเป็นระบบการส่งข้อมูลในรูปแบบอนุกรมคือ ข้อมูลจะส่งไปได้ทีละบิตโดยจะมีอัตราการส่งข้อมูลเป็นบิตต่อวินาทีหรืออัตราการเปลี่ยนแปลงของสัญญาณใน 1 วินาทีเรียกว่า Baud rate โดยจะมีการส่งบิตเริ่มต้น (Start Bit) มีระดับสัญญาณเป็น 0 และบิตข้อมูล (Data bit) ซึ่งจะมีข้อมูล 7 บิตหรือ 8 บิต และอาจตามด้วยบิตพาริตี้ (Parity bit) อาจจะเป็นแบบคู่ (Even) หรือแบบคี่ (Odd) เพื่อตรวจสอบความถูกต้องของข้อมูล บิตพาริตี้นั้นจะมีหรือไม่มีก็ได้และสุดท้ายจะต้องตามด้วยบิตหยุด (Stop bit) ซึ่งจะมีความกว้างของสัญญาณเป็น 1, 1.5, 2 บิตก็ได้ ซึ่งจะเห็นว่าในหารส่งข้อมูลแบบนี้จึงจำเป็นต้องมีข้อตกลงกันระหว่างการรับและการส่งคือ

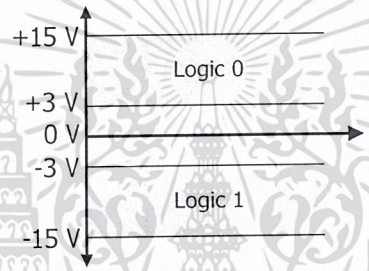
1. ความเร็วในการส่ง Boud rate
2. จำนวนข้อมูล
3. มีบิตพาริตี้หรือ ถ้ามีจะเป็นแบบคู่หรือแบบคี่

4. จำนวนบิตหยุด 1, 1.5 หรือ 2 บิต

ในการส่งแบบ RS-232C นี้เราสามารถเลือก Baud rate ได้หลาย Boud rate เช่น 110, 200, 300, 1200, 2400, 4800, 9600 เช่นถ้าเราส่งข้อมูล 8 บิตโดย Boud rate 2400 แล้ว 1 บิตต้องใช้เวลา 1/2400 = 416 ไมโครวินาที

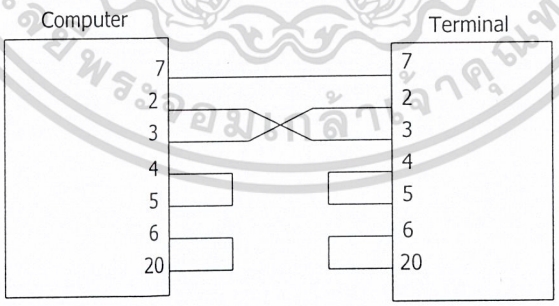
ลักษณะของสัญญาณ RS-232C

เพื่อให้การส่งข้อมูลสามารถไปได้ไกลขึ้นจึงกำหนดให้การส่งสัญญาณมีระดับความแตกต่างเป็นบวก 15 โวลต์ หรืออาจจะเป็นบวก 12 โวลต์ และลบ 12 โวลต์ก็ได้มีระดับสัญญาณลอจิกดังรูปที่ 14.1



รูปที่ จ.9 ระดับสัญญาณลอจิก RS-232C

การกำหนดจุดข้อต่อของ RS-232C



รูปที่ จ.10 การกำหนดจุดต่อของ RS-232C

สัญญาณของขาต่างๆ ที่ใช้งาน

ขาที่ 2 Transmit data

ขาที่ 3 Receive data เป็นขาที่ได้รับสัญญาณข้อมูลจากเครื่องอื่น

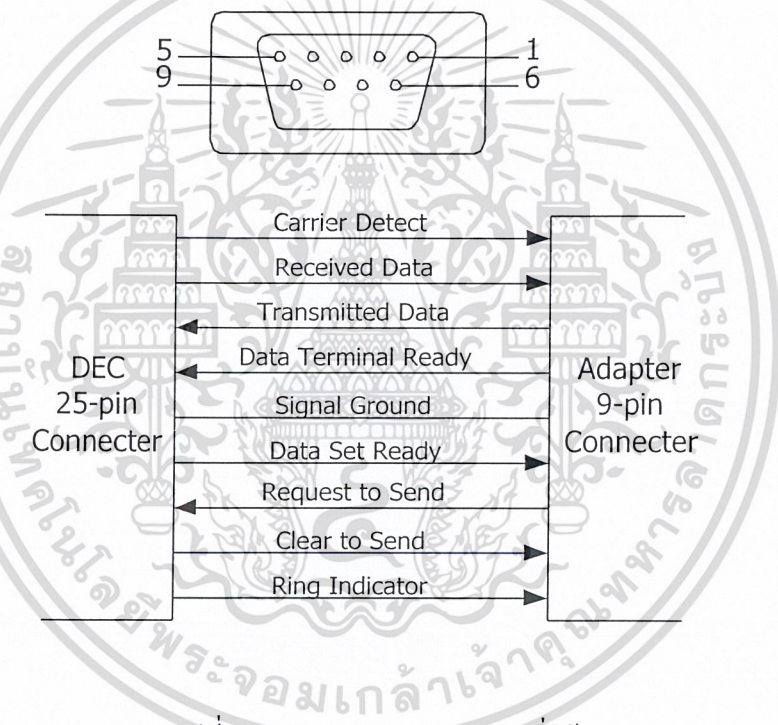
ขาที่ 4 Request to sent เป็นขาที่บอกเครื่องอื่นๆ ว่าตัวเองพร้อมที่จะส่งข้อมูล

ขาที่ 5 Clear to send เป็นขาที่บอกต่อเครื่องอื่นๆ ว่าตัวเองพร้อมที่จะรับข้อมูล

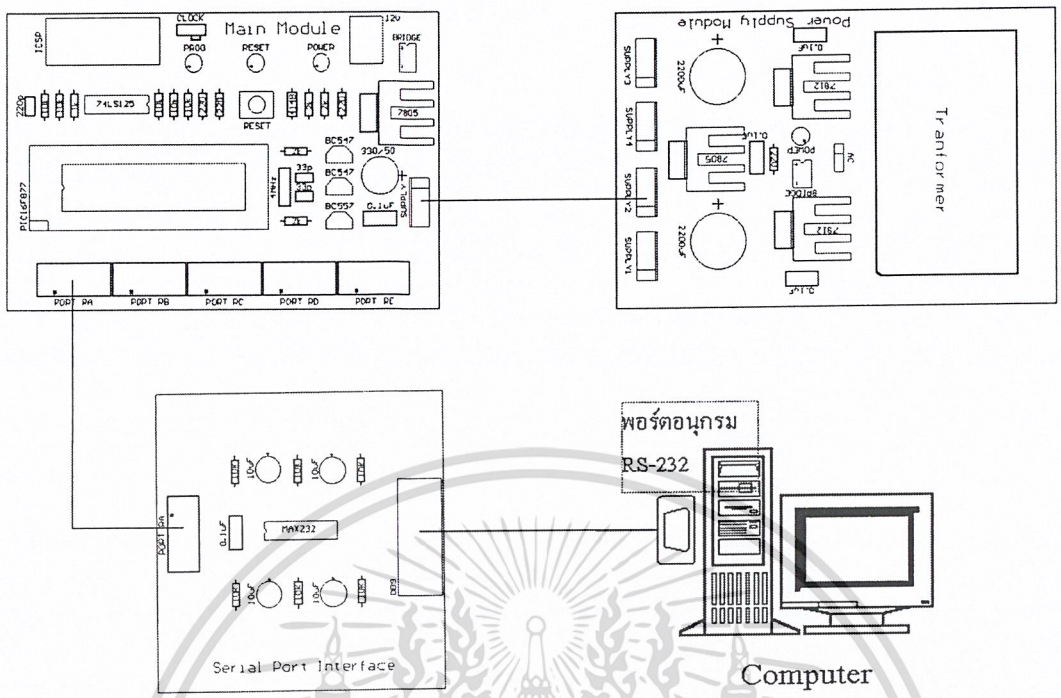
ขาที่ 6 Data set ready เป็นขาที่บอกไมโครคอมพิวเตอร์ว่าโมเด็มต่อเข้ากับสายโทรศัพท์เรียบร้อยแล้วและพร้อมที่จะส่งได้แล้ว

ขาที่ 7 Signal ground เป็นขา GND ของสัญญาณ

ขาที่ 20 Data terminal ready เป็นขาบอกว่าโมเด็มไมโครคอมพิวเตอร์พร้อมที่จะติดต่อด้วย โดยทั่วไปเราสามารถต่อใช้เครื่องคอมพิวเตอร์ติดต่อกันทาง RS-232C ได้ง่ายๆ โดยใช้สัญญาณ Transmit data เพียง 2 เส้นโดยต่อได้ดังรูป จ.11



รูปที่ จ.11 สัญญาณของขาต่างๆ ที่ใช้งาน



รูปที่ จ.12 การเชื่อมต่อโมดูลสื่อสารข้อมูลกับคอมพิวเตอร์

```
#include "Serial.h"

#int_RDA
RDA_isr() {
char c;
  c = getc();
  printf("You press key : %c", c);
  putc(13);
}

void main() {
  set_tris_a(0xfe);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  enable_interrupts(INT_RDA);
  enable_interrupts(global);

  delay_ms(500);
  printf("Welcom to PIC16F877");
  putc(13);
  printf("Serial Module");
  while (1);
}
```

รูปที่ จ.13 โปรแกรมที่ใช้เชื่อมต่อกับ RS 232

ลำดับขั้นตอนการทดลอง

1. ทำการเปิดโปรแกรม PCW สร้างไฟล์ใหม่ขึ้นมาเป็นนามสกุล .C

2. ทำการเขียนโปรแกรมตามรูปที่ จ.13

3. ทำการคอมไพล์แบบ 14 บิท แล้วจะได้ไฟล์ที่เป็น .HEX มาใช้งาน

4. ต่อสายเชื่อมต่อตามรูปที่ จ.12

5. เปิดโปรแกรม IC-Prog เปิดไฟล์ที่สร้างไว้แล้ว ทำการเบิร์นข้อมูลลง PIC16F877 กดปุ่ม RESET เพื่อทำการ RUN โปรแกรมที่เขียนไว้

6. ปิดโปรแกรม Hyper Terminal ตั้งค่า Bits per second ไว้ที่ 9600 Data bits 8 Stop bits 1 แล้วทำการกดปุ่ม RESET บนโมดูลหลักอีกครั้งหนึ่ง

7. สังเกตผลที่ได้บนหน้าจอ Hyper Terminal

.....

.....

.....

.....

.....

8. การตั้งค่าบอร์ดเรท สามารถทำได้โดยส่วนใดของโปรแกรม

.....

.....

.....

.....

9. ลองพิมพ์ตัวอักษรใดๆก็ได้ลงบนคีย์บอร์ด แล้วสังเกตผลที่ได้

.....

.....

.....

.....

สรุปผลการทดลอง

.....

.....

.....

.....

.....

.....

.....

.....

คำถามท้ายการทดลอง

1. จงอธิบายอธิบายหลักการสื่อสารข้อมูลกับคอมพิวเตอร์ผ่านพอร์ตอนุกรม RS-232 ได้
2. ถ้าต้องการเปลี่ยน Baud rate ในการสื่อสารข้อมูลจะต้องเปลี่ยนค่าในรีจิสเตอร์ใด

ใบงานที่ 3

การเชื่อมต่อ PIC16F87X กับการแสดงผลด้วยดอทเมตริกซ์ 24x8

วัตถุประสงค์เชิงพฤติกรรม

เพื่อให้นักศึกษาสามารถ

1. อธิบายหลักการของการใช้งาน Dotmatrix ได้
2. เข้าใจหลักการของการใช้ PIC16F877 เชื่อมต่อกับ Dotmatrix ได้
3. ใช้งาน Dotmatrix ในรูปแบบต่างๆได้

เครื่องมือและอุปกรณ์

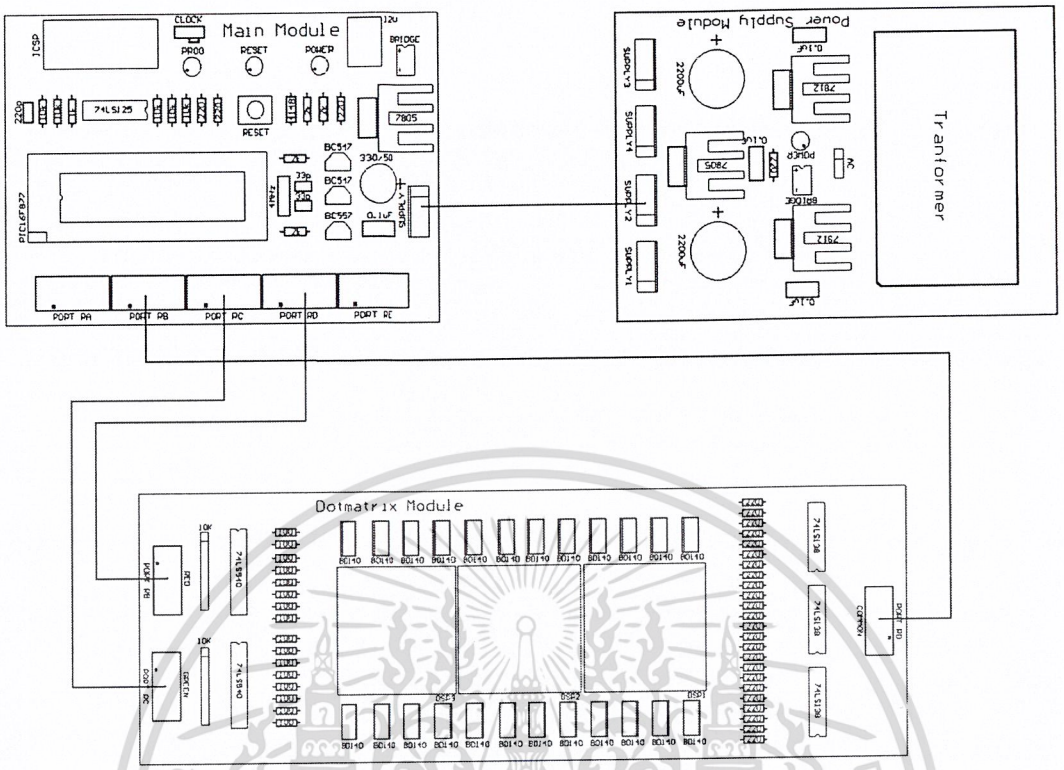
1. โมดูลหลัก PIC-ICSP
2. การแสดงผลด้วยดอทเมตริกซ์ 24x8
3. คอมพิวเตอร์พีซีที่ติดตั้งโปรแกรม PCW Compiler
4. สายเชื่อมต่อระหว่างโมดูลหลัก PIC-ICSP กับคอมพิวเตอร์พีซี
5. สายเชื่อมต่อพอร์ตระหว่างโมดูล

ทฤษฎีเบื้องต้น

LED แบบดอทเมทริกเป็นลักษณะของการนำ LED มาเรียงต่อกันเป็นแถว และใช้คอมมอนร่วมกัน ลดความยุ่งยากในการต่อ LED ให้ใช้งานได้ง่ายขึ้น Dotmatrix ตามมาตราฐานแล้วจะแบ่งเป็นสองชนิดใหญ่ๆคือ คอมมอนแคโถด และคอมมอนแอโนด

ขนาดของ Dotmatrix มีหลายขนาดให้เลือกใช้งานตามความต้องการ ตัวอย่างเช่นขนาด 5x7 และ8x8 เป็นต้น การใช้งาน Dotmatrix นั้น จะสามารถทำได้โดย

1. คอมมอนแคโถด จะสามารถใช้งานได้โดยการต่อขาคอมมอนเข้ากับ GND และต่อVCC เข้ากับขาที่เหลือทั้งหมด เพื่อเป็นการทำให้ไฟติดสว่างทุกดวงถ้าทำการเชื่อมต่อกับ ไมโครคอนโทรลเลอร์ ก็ให้นำขาที่เหลือนี้ต่อกับเอาต์พุตของไมโครคอนโทรลเลอร์แทน
2. คอมมอนแอโนดให้ทำการต่อเข้ากับ VCC เข้ากับขาคอมมอนและให้นำขาที่เหลือต่อกับเอาต์พุตของไมโครคอนโทรลเลอร์ หรือ GND



รูปที่ จ.14 การเชื่อมต่อโมดูลหลัก PIC-ICSP และการแสดงผลด้วยดอทเมตริกซ์ 24x8

```
//Matrix
// Port B   Column
// Port C   Red
// Port D   Green
#include "matrix.h"
#include "amPicLib.h"
int mxRed[24];
int mxGrn[24];
int mxD;
#define Col   port_B
#define Red   port_C
#define Grn   port_D
// Font
byte const Font[43][5] = {
  {0x7C, 0x8A, 0x92, 0xA2, 0x7C}, // 0
  {0x00, 0x42, 0xFE, 0x02, 0x00}, // 1
  {0x42, 0x86, 0x8A, 0x92, 0x62}, // 2
  {0x84, 0x82, 0xA2, 0xD2, 0x8C}, // 3
     {0x18, 0x28, 0x48, 0xFE, 0x08}, // 4
  {0xE4, 0xA2, 0xA2, 0xA2, 0x9C}, // 5
  {0x3C, 0x52, 0x92, 0x92, 0x0C}, // 6
  {0x80, 0x80, 0x9E, 0xA0, 0xC0}, // 7
 {0x6C, 0x92, 0x92, 0x92, 0x6C}, // 8
```


รูปที่ จ.15 โปรแกรมที่ใช้ในการทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

```
 {0x60, 0x92, 0x92, 0x94, 0x78}, // 9
{0x00, 0x6C, 0x6C, 0x00, 0x00}, // :
{0x00, 0x6A, 0x6C, 0x00, 0x00}, // ;
  {0x10, 0x28, 0x44, 0x82, 0x00}, // <
  {0x28, 0x28, 0x28, 0x28, 0x28}, // =
  {0x00, 0x82, 0x44, 0x28, 0x10}, // >
  {0x40, 0x80, 0x8A, 0x90, 0x60}, // ?
  {0x7C, 0x82, 0xBA, 0xAA, 0x7A}, // @
  {0x7E, 0x90, 0x90, 0x90, 0x7E}, // A
  {0xFE, 0x92, 0x92, 0x92, 0x6C}, // B
  {0x7C, 0x82, 0x82, 0x82, 0x44}, // C
  {0xFE, 0x82, 0x82, 0x44, 0x38}, // D
  {0xFE, 0x92, 0x92, 0x92, 0x82}, // E
  {0xFE, 0x90, 0x90, 0x90, 0x80}, // F
  {0x7C, 0x82, 0x92, 0x92, 0x5E}, // G
  {0xFE, 0x10, 0x10, 0x10, 0xFE}, // H
  {0x00, 0x82, 0xFE, 0x82, 0x00}, // I
  {0x04, 0x02, 0x82, 0xFC, 0x80}, // J
  {0xFE, 0x10, 0x28, 0x44, 0x82}, // K
  {0xFE, 0x02, 0x02, 0x02, 0x02}, // L
  {0xFE, 0x40, 0x30, 0x40, 0xFE}, // M
  {0xFE, 0x20, 0x10, 0x08, 0xFE}, // N
  {0x7C, 0x82, 0x82, 0x82, 0x7C}, // O
  {0xFE, 0x90, 0x90, 0x90, 0x60}, // P
  {0x7C, 0x82, 0x8A, 0x84, 0x7A}, // Q
  {0xFE, 0x90, 0x98, 0x94, 0x62}, // R
  {0x64, 0x92, 0x92, 0x92, 0x4C}, // S
  {0x80, 0x80, 0xFE, 0x80, 0x80}, // T
  {0xFC, 0x02, 0x02, 0x02, 0xFC}, // U
  {0xF8, 0x04, 0x02, 0x04, 0xF8}, // V
  {0xFC, 0x02, 0x1C, 0x02, 0xFC}, // W
  {0xC6, 0x28, 0x10, 0x28, 0xC6}, // X
  {0xE0, 0x10, 0x0E, 0x10, 0xE0}, // Y
  {0x86, 0x8A, 0x92, 0xA2, 0xC2}  // Z
};
void ScanMatrix() {
int i;
  mxD = (mxD+1) % 24;
  i = 23-mxD;
  Red = 0x00;
  Grn = 0x00;
  Col = (0x08<<(i/8)) + (i%8);
  Red = mxRed[mxD];
  Grn = mxGrn[mxD];}
#int_RTCC
RTCC_isr() {
  ScanMatrix();}
long Speed;
int Color, cor;
enum {cRed=1, cGreen=2, cOrange=3, cAlter=4};
TextShiftLeft(char ch) {
int i, j;
  switch (Color) {
```

รูปที่ จ.15 (ต่อ) โปรแกรมที่ใช้ในการทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

```
case cRed : cor = 1; break;
    case cGreen : cor = 2; break;
case cOrange : cor = 3; break;
    case cAlter : cor = (cor%3)+1; break;}

  for (j=0; j<6; j++) {
    for (i=0; i<23; i++) {
      mxRed[i] = mxRed[i+1];
      mxGrn[i] = mxGrn[i+1];
    }

    if ( bit_test(cor, 0) && (j<5) && (ch!=' ') ) mxRed[23]=Font[ch-
'0'][j]; else mxRed[23]=0;
    if ( bit_test(cor, 1) && (j<5) && (ch!=' ') ) mxGrn[23]=Font[ch-
'0'][j]; else mxGrn[23]=0;
    delay_ms(Speed);
  }
}

void main() {
int i, j;
  set_tris_b(0x00);
  set_tris_c(0x00);
  set_tris_d(0x00);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
  enable_interrupts(INT_RTCC);
  enable_interrupts(global);
  for (i=0; i<24; i++) {
    mxRed[i] = 0;
    mxGrn[i] = 0;
  }  Speed = 50;
  Color = cAlter;
  while (true) {
    TextShiftLeft("THE PIC16F87X MICROCONTROLLER ");
    TextShiftLeft("DEMONSTRATOR CONTROLED BY C LANGUAGE    ");
  }
}
```

รูปที่ จ.15 (ต่อ) โปรแกรมที่ใช้ในการทดลองการแสดงผลด้วยดอทเมตริกซ์ 24x8

ลำดับขั้นตอนการทดลอง

1. ต่อเชื่อมสายตามรูปที่ จ.14
2. ทำการเปิดโปรแกรม PCW สร้างไฟล์ใหม่ขึ้นมาเป็นนามสกุล .C
3. ทำการเขียนโปรแกรมตามรูปที่ จ.15
4. ทำการคอมไพล์แบบ 14 บิต แล้วจะได้ไฟล์ที่เป็น .HEX มาใช้งาน
5. เปิดโปรแกรม IC-Prog เปิดไฟล์ที่สร้างไว้ แล้วทำการเบิร์นข้อมูลลง PIC16F877
6. กดปุ่มRESET เพื่อทำการ RUN โปรแกรมที่เขียนไว้

7. สังเกตุผลที่ได้บนแอลอีดีดอทเมทริกต์

.....

.....

.....

.....

.....

8.การเปลี่ยนความเร็วในการเลื่อนของตัวอักษรสามารถทำได้โดยการเปลี่ยนโปรแกรมอย่างไร

.....

.....

.....

.....

.....

9. การเปลี่ยนสีของตัวอักษรจะต้องทำการเปลี่ยนโปรแกรมอย่างไร

.....

.....

.....

.....

.....

.....

.....

.....

.....

สรุปผลการทดลอง

.....

.....

.....

.....

.....

คำถามท้ายการทดลอง

ให้ผู้ทดลองทำการเปลี่ยนสีของตัวอักษรให้ติดสลับสีกันให้ครบทั้ง 3 สี

ใบงานที่ 4
การเชื่อมต่อ PIC16F87X กับการเชื่อมต่อแบบไอสแควร์ซี

วัตถุประสงค์เชิงพฤติกรรม

เพื่อให้นักศึกษาสามารถ

1. ใช้งานการติดต่อข้อมูลแบบ I2C ได้
2. เขียนโปรแกรมติดต่อข้อมูลแบบ I2C ได้
3. ใช้งาน PIC16F877 ติดต่อกับโมดูล I2C Interface ได้

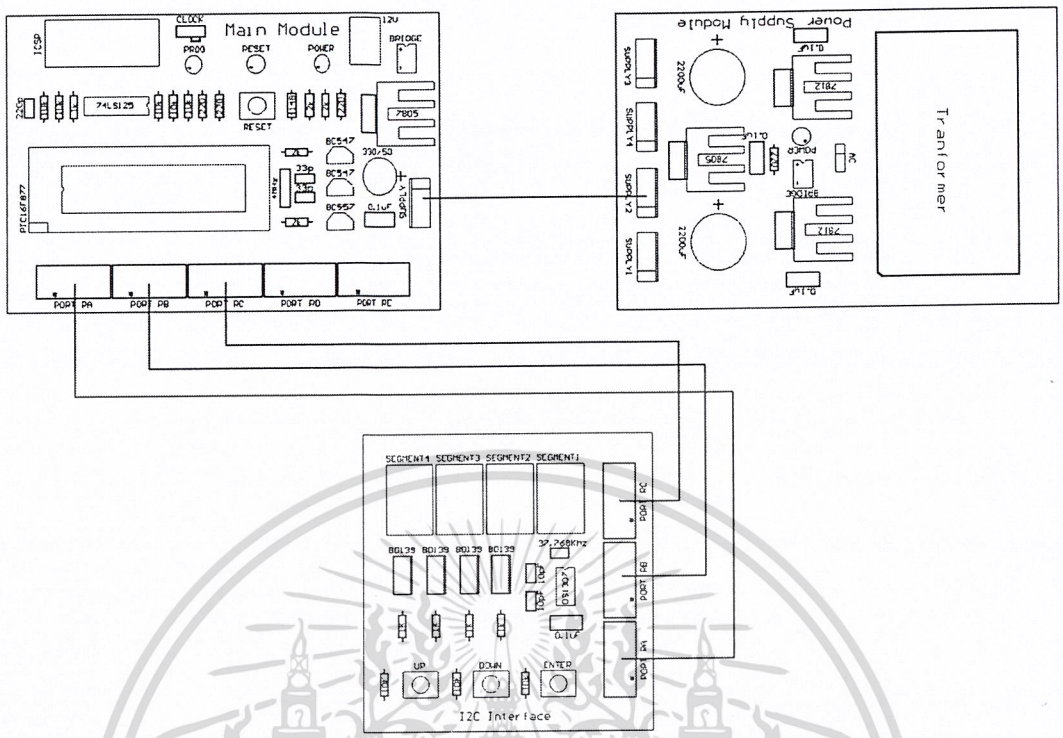
เครื่องมือและอุปกรณ์

1. โมดูลหลัก PIC ICSP
2. การเชื่อมต่อแบบไอสแควร์ซี
3. คอมพิวเตอร์พีซีที่ติดตั้งโปรแกรม PCW Compiler
4. สายเชื่อมต่อระหว่างโมดูลหลัก PIC ICSPกับคอมพิวเตอร์พีซี
5. สายเชื่อมต่อระหว่างโมดูล

ทฤษฎีเบื้องต้น

I2C ย่อมาจาก Inter-IC Communication หมายถึงการติดต่อสื่อสารระหว่างไอซีโดยบัส I2C ได้รับการพัฒนาขึ้นโดยฟิลิปส์ ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อสั่งงานและควบคุมภายใต้สัญญาณเพียง 2เส้น เส้นหนึ่งคือสายข้อมูลอีกสายหนึ่งคือสายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I2C ทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการทำการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I2C มีชื่อเรียกอย่างเป็นทางการว่าสายข้อมูลอนุกรมหรือ SDA ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรมหรือ SCL



รูปที่ จ.16 การเชื่อมต่อโมดูลหลักกับการเชื่อมต่อแบบไอสแควร์ซี

```
#include "I2C.h"
#include "amPicLib.h"

//------------------------------ DS1307 RTC
#define i2c_SDA  PIN_B0
#define i2c_SCL  PIN_B1
#use i2c(master, sda=i2c_SDA, scl=i2c_SCL)

void init_i2c() {
  output_float(i2c_SCL);
  output_float(i2c_SDA);
}

void write_DS1307 (byte addr, data) {
  i2c_start();
  i2c_write(0xd0);
  i2c_write(addr);
  i2c_write(data);
  i2c_stop();
  delay_ms(8);}
int read_DS1307 (byte addr) {
int data;
```

รูปที่ จ.17 โปรแกรมที่ใช้ในการทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
  i2c_write(0xd0);
  i2c_write(addr);
  i2c_start();
  i2c_write(0xd1);
  data = i2c_read(0);
  i2c_stop();
  return(data);
}

  // Key ------------------------------------------------
enum {swEnter=1, swUp=2, swDown=4};
unsigned char keyNow=0, wKey=0;

  // Segment-------------------------------------------------
int segData[5];
int segDigit=0;
int const number[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f};  //        0-9

//==================== declaration
  // system
int sysclk=0, sysclk2=0;

  // parameter
enum {modeTime, modeSet};
int NowMode;
int seg_rate=0;
short seg_blink;

int const maxTime[2] = {0x23, 0x59};
int const minTime[2] = {0x00, 0x00};
enum {adrSecond, adrMinute, adrHour, adrDay, adrDate, adrMonth,
adrYear};
int buffTime[2], adrTime;

//====================== Task

void SetMode(int mode) {
int i;
  NowMode = mode;
  switch (NowMode) {
    case modeTime :
          case modeSet :

      adrTime = 0;
      buffTime[0] = read_DS1307(adrHour);
      buffTime[1] = read_DS1307(adrMinute);
      break;
  }
}
```

รูปที่ จ.17 (ต่อ) โปรแกรมที่ใช้ในการทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
 switch (NowMode) {
    case modeTime :
      i = read_DS1307(adrHour);
      segData[0] = number[i>>4];
      segData[1] = number[i&15];
      i = read_DS1307(adrMinute);
      segData[2] = number[i>>4];
      segData[3] = number[i&15];
      if (seg_blink) segData[1]|=0x80;
      break;
    case modeSet :
      i = buffTime[0];
      segData[0] = number[i>>4];
      segData[1] = number[i&15] |= 0x80;
      i = buffTime[1];
      segData[2] = number[i>>4];
      segData[3] = number[i&15];
      if (!seg_blink) {
        segData[2*adrTime] = 0;
        segData[(2*adrTime)+1] = 0;
      }
      segData[1] |= 0x80;
      break;
  }}


//================== Events Operating

void Key_OnDown() {
int i;
  switch (NowMode) {
                switch (keyNow) {
             case swEnter : adrTime = (adrTime+1) % 2; goto MST1;
                          case swUp :
          i = buffTime[adrTime];
          if (i<maxTime[adrTime]) {
            i++;
          }

          else i = minTime[adrTime];
          buffTime[adrTime] = i;
          goto MST1;
        case swDown :
          i = buffTime[adrTime];
          if (i>minTime[adrTime]) {
            i--;
            if ((i&0x0f)>9) i-=6;}
          else i = maxTime[adrTime];
          buffTime[adrTime] = i;
```

รูปที่ จ.17 (ต่อ) โปรแกรมที่ใช้ในการทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
MST1 :
                    seg_blink = 0;
                    seg_rate = 1;
                    break;
              }
              break;
       }
}

void Key_OnClick() {

}

void Key_OnFPress() {
int i;
       switch (NowMode) {
              case modeTime :
                    if (keyNow==swEnter) SetMode(modeSet);
                    break;
              case modeSet :

                 if (keyNow==swEnter) {
          write_DS1307(adrHour, buffTime[0]);
          write_DS1307(adrMinute, buffTime[1]);
                    SetMode(modeTime);
                 }
       if ((keyNow==swUp)||(keyNow==swDown)) {
          Key_OnDown();
          wKey=5;
       }
              break;
}
}

void Key_OnUp() {
}

//====================== Create Events

void Scan_Key() {
unsigned char k;
  k = (~Port_A) & 7;
  if (k != keyNow) {
    if (keyNow) {
      if (wKey<50) {
        if (wKey) Key_OnClick();
        Key_OnUp();
    keyNow = 0;
  }
  wKey = 0;
 }
```

รูปที่ จ.17 (ต่อ) โปรแกรมที่ใช้ในการทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
f (k) {
keyNow = k;
      wKey = 52;                // 1 Sec
    }}
void Scan_KeyFPress() {
  if (wKey) {
    wKey--;
    if (wKey==50) Key_OnDown();
    if (!wKey) Key_OnFPress();
  }
void Scan_Seg_Paint() {
  if (seg_rate) seg_rate--;
  if (!seg_rate) {
      seg_rate = 25;          // 500ms
      Seg_Paint();
      }}
void Scan_Segment() {
  bit_clear(port_B, 4+segDigit);
  segDigit = (segDigit+1) % 4;
  port_C = segData[3-segDigit];
  bit_set(port_B, 4+segDigit);}
//====================Multitasking
Task_successive() {
  Scan_Key();}
Task_interval20ms() {
  Scan_KeyFPress();
  Scan_Seg_Paint();}
Task_interval500ms() {
}
//======================= isr
#int_RTCC
RTCC_isr() {
  sysclk++;
  Scan_Segment();}
//====================== Main
void main() {

int i;
  set_tris_b(0x0f);
  set_tris_c(0x00);
  port_b_pullups(TRUE);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_8);
  setup_counters(RTCC_INTERNAL,RTCC_DIV_4);
  enable_interrupts(INT_RTCC);
  enable_interrupts(global);
// Initial
  init_i2c();
  i = read_DS1307(adrSecond);
  if (bit_test(i, 7)) write_DS1307(adrSecond, 0);
  SetMode(modeTime);
  while (true) {
 Task_successive();
if (sysclk>=20) {
```

รูปที่ จ.17 (ต่อ) โปรแกรมที่ใช้ในการทดลองการเชื่อมต่อแบบไอสแควร์ซี

```
sysclk %= 20
Task_interval20ms();
            if ((++sysclk2)>=25) {
        sysclk2 = 0;
        Task_interval500ms();
      }
    }
  }
}
```

รูปที่ จ.17 (ต่อ) โปรแกรมที่ใช้ในการทดลองการเชื่อมต่อแบบไอสแควร์ซี

ลำดับขั้นการทดลอง

1. ต่อวงจรตามรูปที่ จ.16
2. ทำการเปิดโปรแกรม PCW สร้างไฟล์ใหม่ขึ้นมาเป็นนามสกุล .C
3. ทำการเขียนโปรแกรมตามรูปที่ จ.17
4. ทำการคอมไพล์แบบ 14 บิท แล้วจะได้ไฟล์ที่เป็น .HEX มาใช้งาน
5. เปิดโปรแกรม IC-Prog เปิดไฟล์ที่สร้างไว้ แล้วทำการเบิร์นข้อมูลลง PIC 16F877
6. กดปุ่มRESET เพื่อทำการ RUN โปรแกรมที่เขียนไว้
7. สังเกตุผลที่ได้บน 7 SEGMENT ทั้ง4 ตัว

.....

.....

.....

.....

8.จากโปรแกรมเราสามารถที่จะทำการตั้งเวลาได้อย่างไรอธิบายโดยดูจากโปรแกรมที่ได้เขียนไว้

.....

.....

.....

.....

.....

.....

.....

สรุปผลการทดลอง

.....

.....

.....

.....

.....

คำถามท้ายการทดลอง

เราสามารถเปลี่ยนโหมดการนับของเวลาให้เป็นการนับที่แสดงแต่วินาทีได้อย่างไรจงเขียนโปรแกรม

ใบงานที่ 5
การเชื่อมต่อ PIC16F87X กับการรับอินพุตแบบเครื่องมือวัด

วัตถุประสงค์เชิงพฤติกรรม

เพื่อให้นักศึกษาสามารถ

1. ใช้งาน PIC16F877 ร่วมกับ Instrument ได้
2. เขียนโปรแกรมที่ใช้ PIC16F877 ติดต่อกับ Input Instruent ได้
3. ประยุกต์ใช้งานการใช้งานกับ Instrument แบบอื่นๆ ได้

เครื่องมือและอุปกรณ์

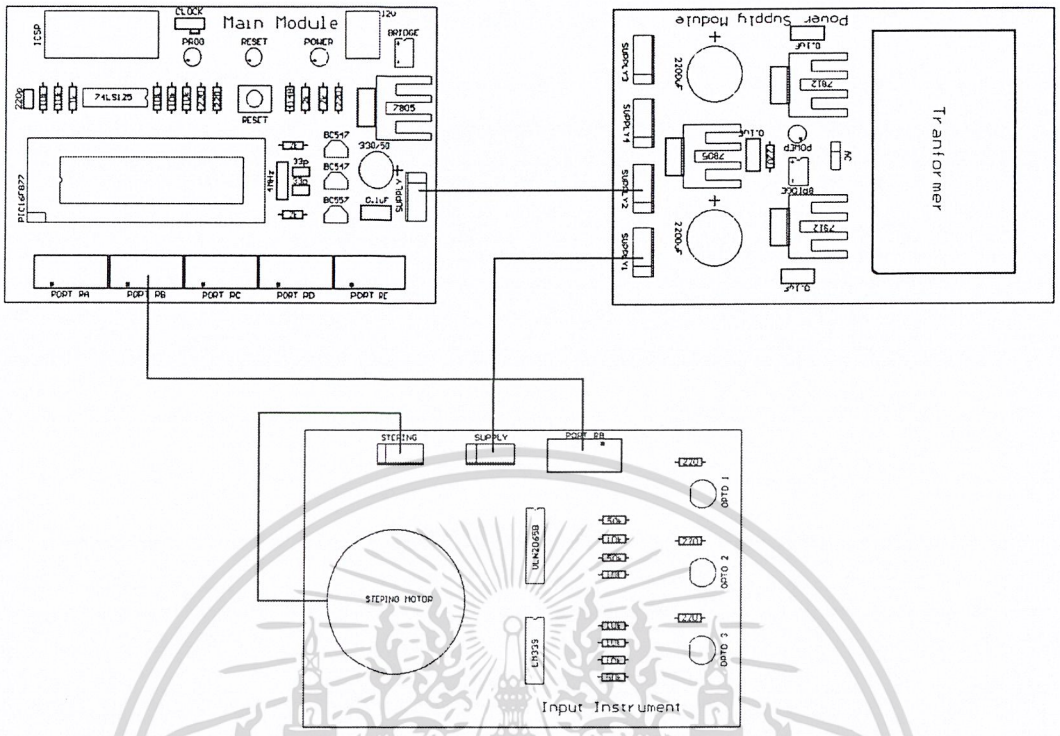
1. โมดูลหลัก PIC-ICSP
2. โมดูล Input Instrument
3. คอมพิวเตอร์พีซีที่ติดตั้งโปรแกรม PCW Compiler
4. สายเชื่อมต่อระหว่างโมดูลหลัก PIC-ICSP กับคอมพิวเตอร์พีซี
5. สายเชื่อมต่อพอร์ตระหว่างโมดูล

ทฤษฎีเบื้องต้น

โฟโต้ทรานซิสเตอร์ (Photo transistor)

เป็นตัวตรวจจับทางแสงที่ทำงานได้ดีกว่าโฟ-โต้เซลล์มาก ซึ่งสามารถที่จะตอบสนองได้รวดเร็วถึง 1 ไมโครวินาทีที่เดียวการทำงานมีหลักการทำงานคล้ายกับโฟโต้ไดโอด แต่โฟโต้ทรานซิสเตอร์จะมีการขยายกระแสผ่านออกมาที่เอาท์พุตด้วย ทำให้โฟโต้ทรานซิสเตอร์ถูกนำมาประยุกต์ใช้งานได้อย่างกว้างขวาง โฟโต้ทรานซิสเตอร์ทุกแบบ จะมีโครงสร้างเป็นชนิด NPN สารที่ถูกนำมาใช้ผลิตได้แก่ ซีลีเนียม, ซิลิกอน หรือ เยอรมันเนียม สารแต่ละชนิดจะมีการตอบสนองต่อสเปกตรัมของคลื่นแสงในย่านที่แตกต่างกันออกไป

โฟโต้ทรานซิสเตอร์ที่สร้างมาจากสารซิลีเนียม จะสามารถตอบสนองต่อสเปกตรัมของคลื่นแสงที่คนเราสามารถมองเห็นได้ ซึ่งมีลักษณะการตอบสนองได้ใกล้เคียงกับสายตาของคนเรา โฟโต้ทรานซิสเตอร์ที่สร้างมาจากสารซิลิกอน จะมีการตอบสนองได้ดีต่อสเปกตรัมของแสงในย่านของแสงอินฟราเรดหรือใกล้เคียง



รูปที่ จ.18 การเชื่อมต่อโมดูลหลักกับการรับอินพุตเครื่องมือวัด

```
#include "Instrument.h"
#include "amPicLib.h"

void main() {
int Direct=0;
int i=0;
  set_tris_c(0x0f);
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_CLOCK_DIV_2);
  while (1) {
    if (~bit_test(port_C, 1)) Direct = 2;
    if (~bit_test(port_C, 3)) Direct = 1;
    if (~bit_test(port_C, 2)) Direct = 0;
      switch (Direct) {
      case 0 :
        port_C &= 0x0f;
        i = 0;
   break;
```

รูปที่ จ.19 โปรแกรมที่ใช้ในการทดลองการรับอินพุตเครื่องมือวัด

```
      case 1 :
        i <<= 1;
        if (!i) i=0x10;
        port_C = (port_C & 0x0f)+(i&0xf0);
        delay_ms(20);
        break;
      case 2 :
        i >>= 1;
        if (!(i&0xf0)) i=0x80;
        port_C = (port_C & 0x0f)+(i&0xf0);
        delay_ms(20);
        break;
    }
  }
  }
```

รูปที่ จ.19 (ต่อ) โปรแกรมที่ใช้ในการรับอินพุตเครื่องมือวัด

ลำดับขั้นการทดลอง

1. เชื่อมต่ออุปกรณ์ตามรูปที่ จ.18
2. ทำการเปิดโปรแกรม PCW สร้างไฟล์ใหม่ขึ้นมาเป็นนามสกุล .C
3. ทำการเขียนโปรแกรมตามรูปที่ จ.19
4. ทำการคอมไพล์แบบ 14 บิต แล้วจะได้ไฟล์ที่เป็น .HEX มาใช้งาน
5. เปิดโปรแกรม IC-Prog เปิดไฟล์ที่สร้างไว้ แล้วทำการเบิร์นข้อมูลลง PIC16F877
6. กดปุ่ม RESET เพื่อทำการ RUN โปรแกรมที่เขียนไว้
7. สังเกตุผลที่ได้จากการหมุนของแสตปปิ้งมอเตอร์ เมื่อนำวัตถุสีขาวมาปิดที่ OPTO_1-OPTO_3

.....

.....

.....

.....

.....

.....

.....

.....

.....

8. จากโปรแกรมเราสามารถเปลี่ยนทิศทางการหมุนของแสตปปิ้งมอเตอร์ได้อย่างไร

.....

.....

.....

.....

.....

9.ถ้าต้องการเปลี่ยนความเร็วในการหมุนของแสตปปิ้งมอเตอร์จะต้องทำการเปลี่ยนโปรแกรมอย่างไร

.....

.....

.....

.....

.....

สรุปผลการทดลอง

.....

.....

.....

.....

.....

.....

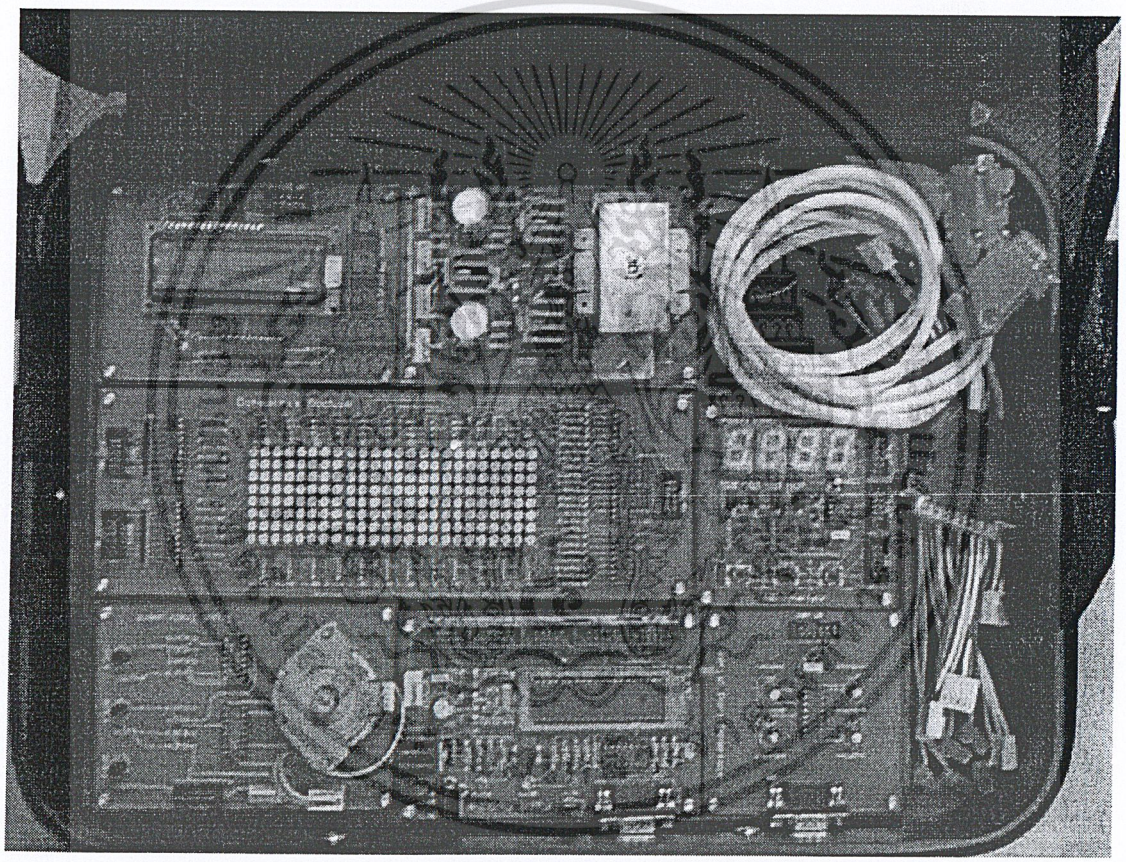
คำถามท้ายการทดลอง

จงเขียนโปรแกรมที่ทำให้แสตปปิ้งมอเตอร์ สามารถหมุนและหยุดได้โดยรับเงื่อนไขจาก OPTO ทั้ง 3 ตัว

ภาคผนวก ฉ
คู่มือการใช้งาน

คู่มือการใช้งาน

ชุดทดลองไมโครคอนโทรลเลอร์ PIC16F87X ควบคุมด้วยภาษาซี



สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์อุตสาหกรรม คณะครุศาสตร์อุตสาหกรรม

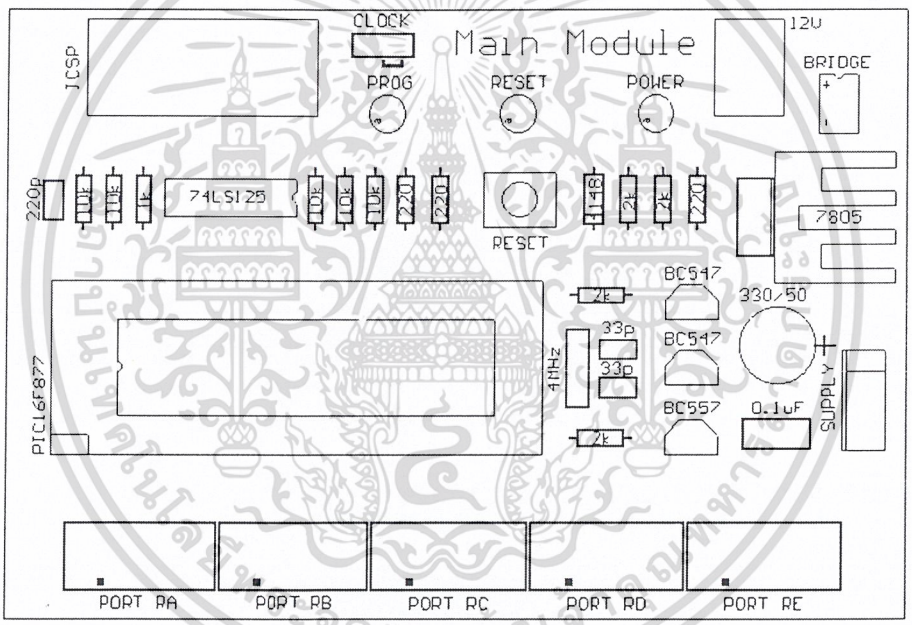
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

1. คู่มือการใช้งานชุดทดลองไมโครคอนโทรลเลอร์ PIC16F877

ชุดทดลองไมโครคอนโทรลเลอร์ PIC16F877 ได้ถูกออกแบบโดยเน้นให้เกิดความสะดวกในการทดลองใช้ไมโครคอนโทรลเลอร์ PIC16F877 เชื่อมต่อกับอุปกรณ์เพื่อควบคุมการทำงาน โดยได้ออกแบบเป็นโมดูลสำหรับทดลองเรื่องต่างๆ เมื่อต้องการทำการทดลองเรื่องใดก็นำโมดูลนั้นมาเสียบสายเชื่อมกับพอร์ตในโมดูลหลัก ซึ่งเป็นโมดูลของไมโครคอนโทรลเลอร์แล้วทำการโปรแกรมการทำงานลงบนไมโครคอนโทรลเลอร์ เพื่อการทดลองหรือพัฒนาโปรแกรม

2. ส่วนประกอบและปุ่มควบคุม

2.1 โมดูลหลัก PIC-ICSP (PIC-In-Cirrcuit Serial Programming Module)

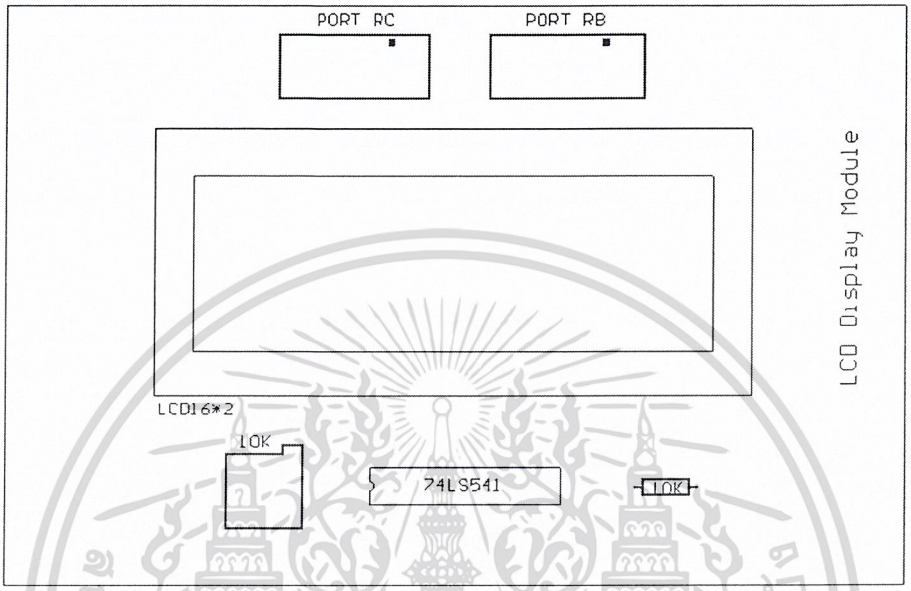


รูปที่ ฉ.1 จุดเชื่อมต่อใช้งานของโมดูลหลัก PIC-ICSP

ALL PORT จากพอร์ตนี้	คือใช้ต่อกับโมดูลทดลองวงจร ขาใช้งานของ PIC ทั้งหมดจะถูกเชื่อมต่อ
SOCKET-BOARD	คือเป็นซ็อกเก็ตแบบ ZIP สำหรับเสียบตัวไมโครคอนโทรล แบบ 40 ขา โดยมีซ็อกเก็ตสำหรับ เสียบ ZIP อีกทีให้ตรงกับจำนวนขาที่ต้องการใช้
ICSP	คือสำหรับเสียบต่อสายดาวน์โหลดข้อมูลไปยังคอมพิวเตอร์
SUPPLY	คือสำหรับต่อเข้ากับภาคจ่ายไฟ
PORTR A-PORTR E8	คือเป็นพอร์ตสำหรับต่อใช้งาน เชื่อมต่อกับโมดูลอื่นๆ

CLOCK คือจั๊มเปอร์เลือกสัญญาณนาฬิกาของไมโครคอนโทรลเลอร์

2.2 การแสดงผลด้วยจอแบบผลึกเหลว (LCD Display)

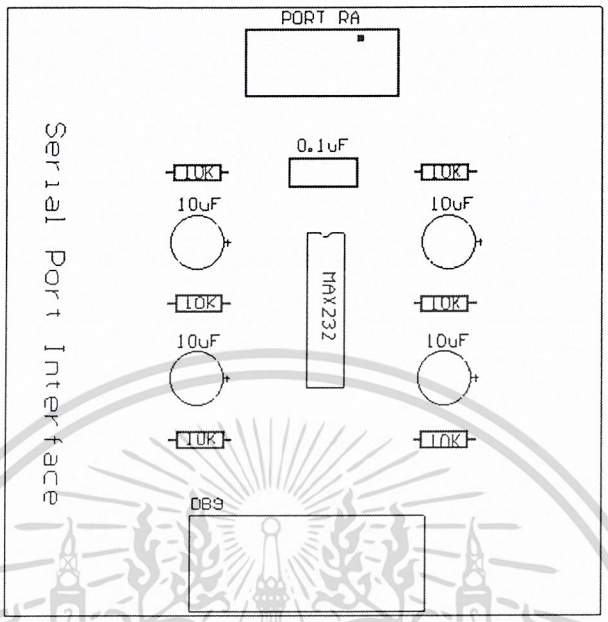


รูปที่ ฉ.2 จุดเชื่อมต่อการใช้งานของการแสดงผลด้วยจอแบบผลึกเหลว

CONTROL คือพอร์ตสัญญาณควบคุม LCD ใช้สัญญาณ 3 จากบิตล่าง

DATA คือพอร์ตรับส่งสัญญาณข้อมูลขนาด 8 บิต

2.3 การเชื่อมต่อแบบพอร์ตอนุกรม (Board Serial Port Interface)



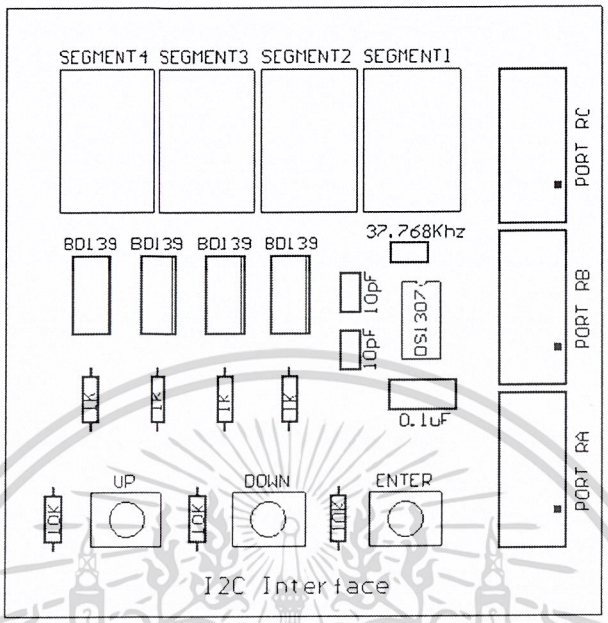
รูปที่ ฉ.3 จุดเชื่อมต่อใช้งานของการเชื่อมต่อแบบพอร์ตอนุกรม

PORT RC คือสำหรับต่อเข้ากับพอร์ต C บิต 7 (RX), บิต 6(TX) และบิต 5 (DIR) ซึ่งเป็นจุดต่อสำหรับ USART ของไมโครคอนโทรลเลอร์

RS-232C คือคอนเน็ตเตอร์ DB-9 ตัวเมีย สำหรับเชื่อมต่อกับพอร์ตอนุกรมคอมพิวเตอร์

SELECT คือจั๊มเปอร์เพื่อเลือกการใช้มาตรฐาน RS-485 หรือ RS-422

2.4 การเชื่อมต่อแบบไอสแควร์ซี (I^2C Interface)

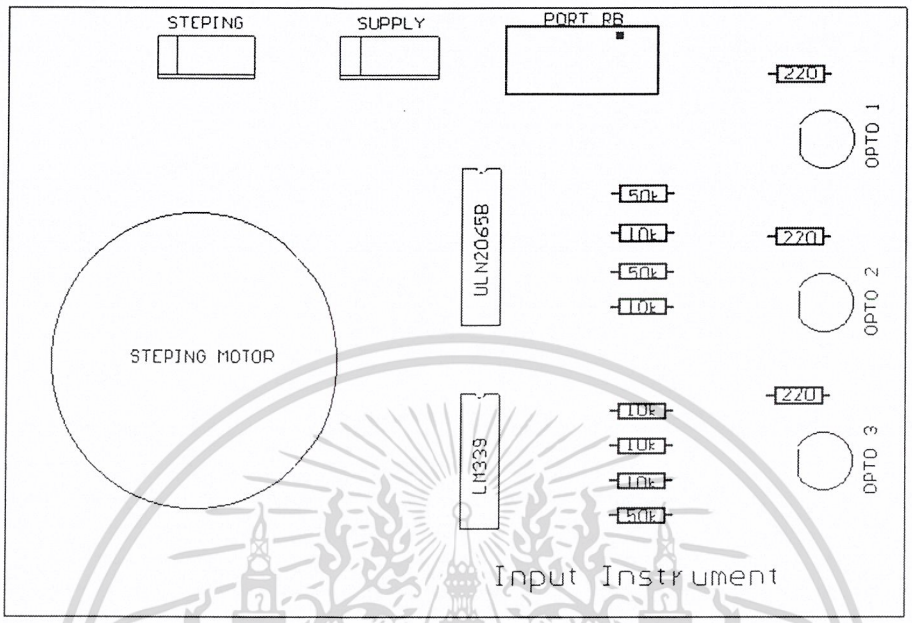


รูปที่ ฉ.4 จุดเชื่อมต่อใช้งานของการเชื่อมต่อแบบไอสแควร์ซี

RTC คือจะใช้สัญญาณจะใช้สัญญาณ 2 บิตจากพอร์ต 1 พอร์ต รับส่งข้อมูลแบบ I^2C

CONTROL คือพอร์ตเชื่อมต่อใช้สัญญาณบิต 2 และบิต 3 เป็นบัสสำหรับ I^2C

2.5 การรับอินพุตแบบเครื่องมือวัด (Input/Instrument)



รูปที่ ฉ.5 จุดเชื่อมต่อใช้งานของการรับอินพุตแบบเครื่องมือวัด

SUPPLY คือสำหรับป้อนแรงดันให้แก่โมดูลเมื่อต้องการให้แรงดันสูงในการขับมอเตอร์

PORT RB คือเป็นคอนเน็ตเตอร์สำหรับเชื่อมต่อเพื่อทดลองกับโมดูลอื่นๆ

ภาคผนวก ช

รายละเอียดและคุณสมบัติของอุปกรณ์

MICROCHIP

PIC16F87X

28/40-pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

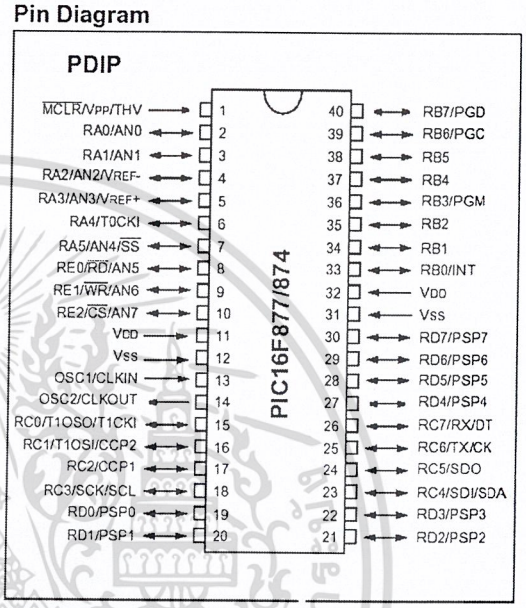
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High-performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM data memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code-protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low-power, high-speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial and Industrial temperature ranges
- Low-power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

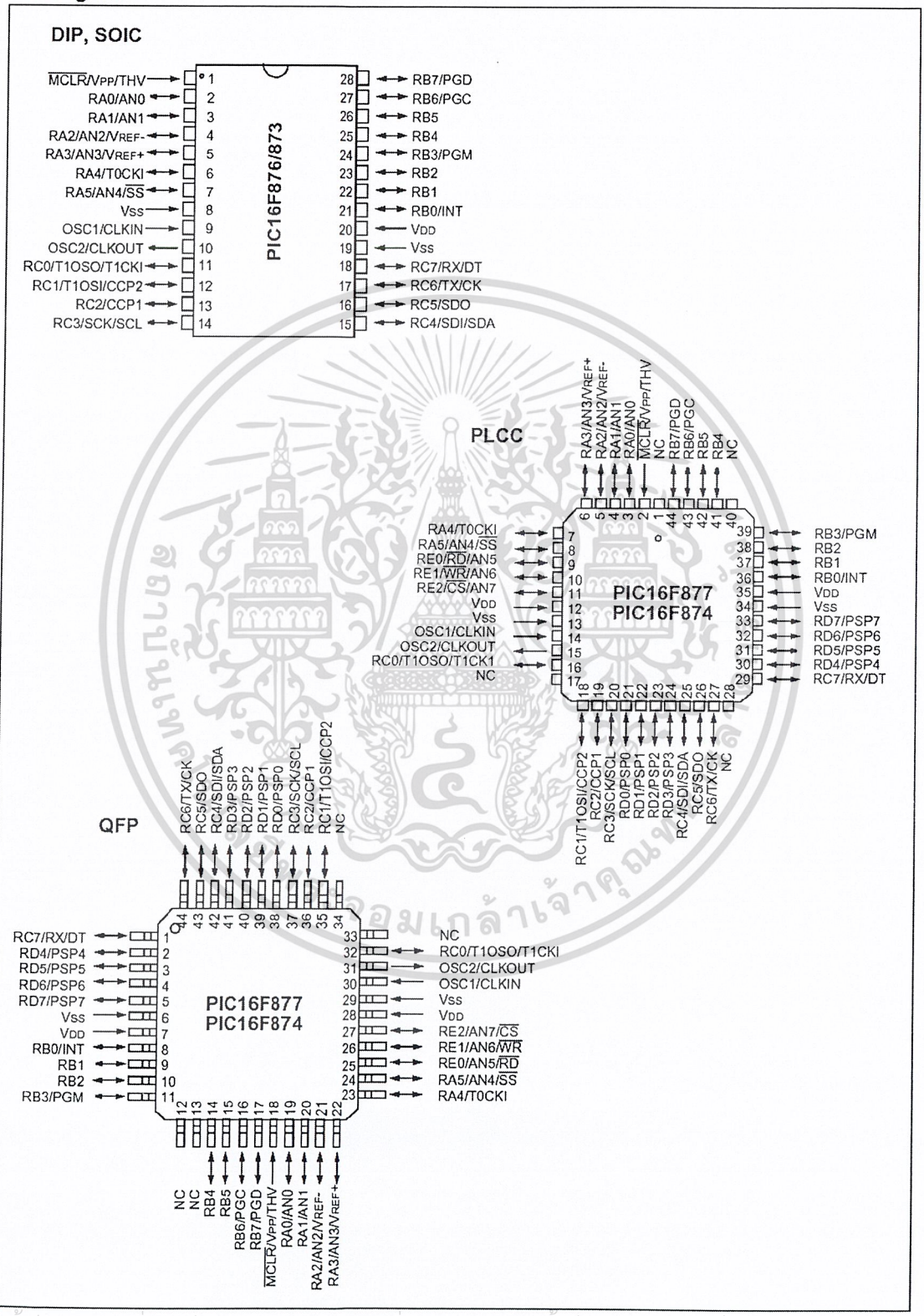
PDIP



Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master Mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Pin Diagrams



Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions

PIC16F87X

Table of Contents

To Our Valued Customers

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number. e.g., DS30000A is version A of document DS30000.

New Customer Notification System

Register on our web site (www.microchip.com/cn) to receive the most current information on our products.

Errata

An errata sheet may exist for current devices, describing minor operational differences (from the data sheet) and recommended workarounds. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 786-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Corrections to this Data Sheet

We constantly strive to improve the quality of all our products and documentation. We have spent a great deal of time to ensure that this document is correct. However, we realize that we may have missed a few things. If you find any information that is missing or appears in error, please:

- Fill out and mail in the reader response form in the back of this data sheet.
- E-mail us at webmaster@microchip.com.

We appreciate your assistance in making this a better document.

1.0 DEVICE OVERVIEW

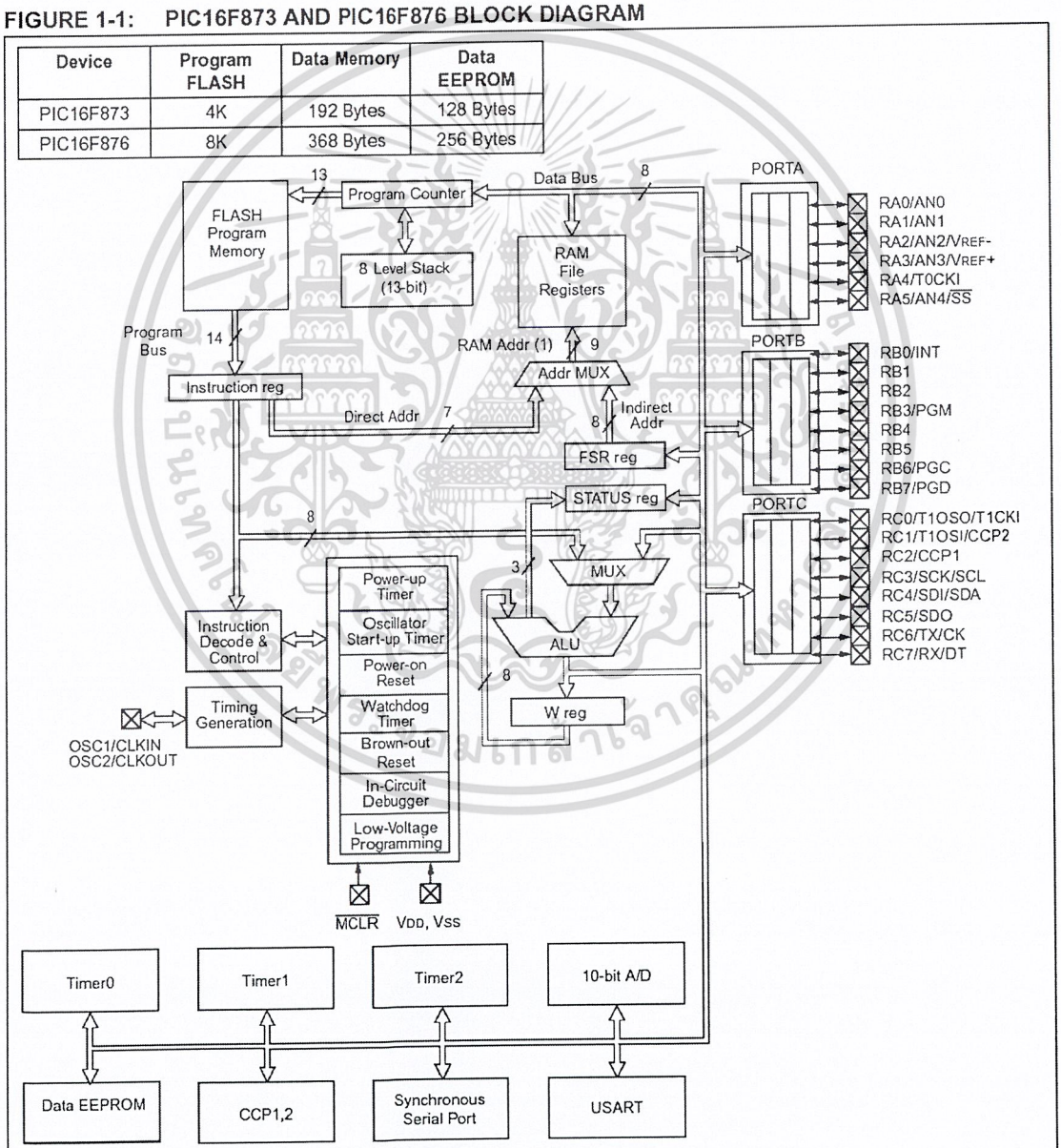
This document contains device-specific information. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be obtained from your local Microchip Sales Representative or downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

There are four devices (PIC16F873, PIC16F874, PIC16F876 and PIC16F877) covered by this data sheet. The PIC16F876/873 devices come in 28-pin packages and the PIC16F877/874 devices come in 40-pin packages. The 28-pin devices do not have a Parallel Slave Port implemented.

The following two figures are device block diagrams sorted by pin number; 28-pin for Figure 1-1 and 40-pin for Figure 1-2. The 28-pin and 40-pin pinouts are listed in Table 1-1 and Table 1-2, respectively.

FIGURE 1-1: PIC16F873 AND PIC16F876 BLOCK DIAGRAM

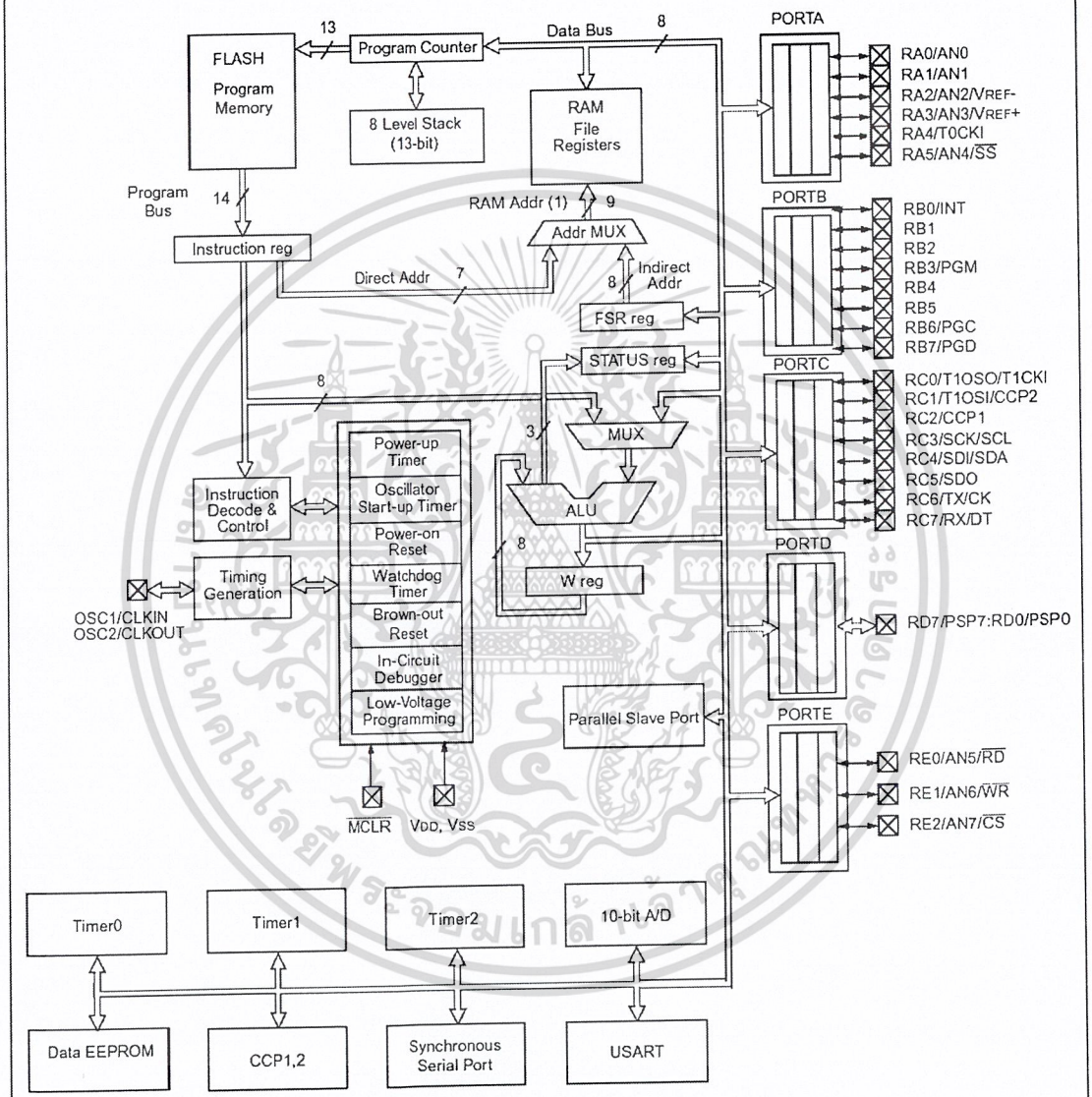
Device	Program FLASH	Data Memory	Data EEPROM
PIC16F873	4K	192 Bytes	128 Bytes
PIC16F876	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

TABLE 1-1: PIC16F873 AND PIC16F876 PINOUT DESCRIPTION

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	9	9	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP/THV	1	1	I/P	ST	Master clear (reset) input or programming voltage input or high voltage test mode control. This pin is an active low reset to the device.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2	I/O	TTL	RA0 can also be analog input0
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input1
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage
RA4/T0CKI	6	6	I/O	ST	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	21	21	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL	RB3 can also be the low voltage programming input
RB4	25	25	I/O	TTL	Interrupt on change pin.
RB5	26	26	I/O	TTL	Interrupt on change pin.
RB6/PGC	27	27	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming data.
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11	I/O	ST	RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI/CCP2	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	14	14	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I^2C modes.
RC4/SDI/SDA	15	15	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	16	16	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS(4)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP/THV	1	2	18	I/P	ST	Master clear (reset) input or programming voltage input or high voltage test mode control. This pin is an active low reset to the device.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19	I/O	TTL	RA0 can also be analog input0
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/counter. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input
RB4	37	41	14	I/O	TTL	Interrupt on change pin.
RB5	38	42	15	I/O	TTL	Interrupt on change pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming data.
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I^2C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUED)

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	
						PORTE is a bi-directional I/O port.
RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
VSS	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
VDD	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

2.0 MEMORY ORGANIZATION

There are three memory blocks in each of these PICmicro MCUs. The Program Memory and Data Memory have separate buses so that concurrent access can occur and is detailed in this section. The EEPROM data memory block is detailed in Section 4.0.

Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

2.1 Program Memory Organization

The PIC16F87X devices have a 13-bit program counter capable of addressing an 8K x 14 program memory space. The PIC16F877/876 devices have 8K x 14 words of FLASH program memory and the PIC16F873/874 devices have 4K x 14. Accessing a location above the physically implemented address will cause a wrap-around.

The reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PIC16F877/876 PROGRAM MEMORY MAP AND STACK

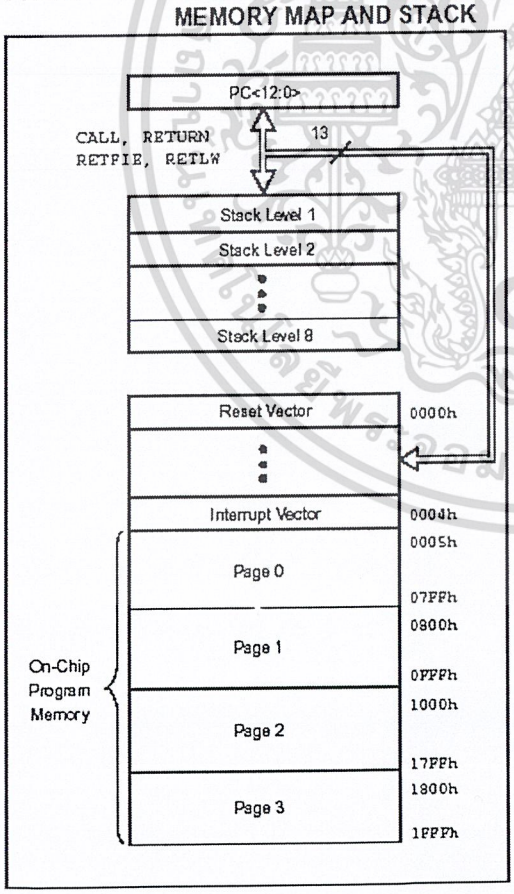
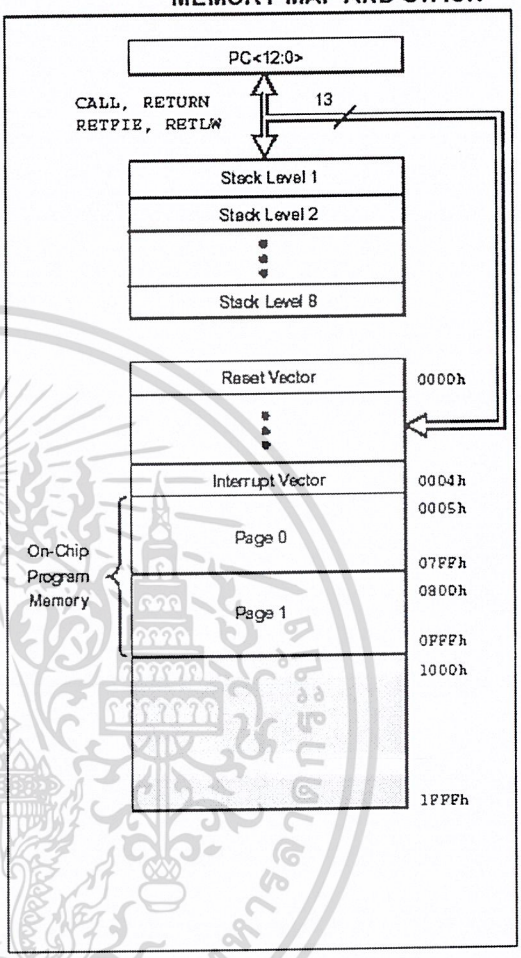


FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK



2.2 Data Memory Organization

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1(STATUS<6>) and RP0 (STATUS<5>) are the bank select bits.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

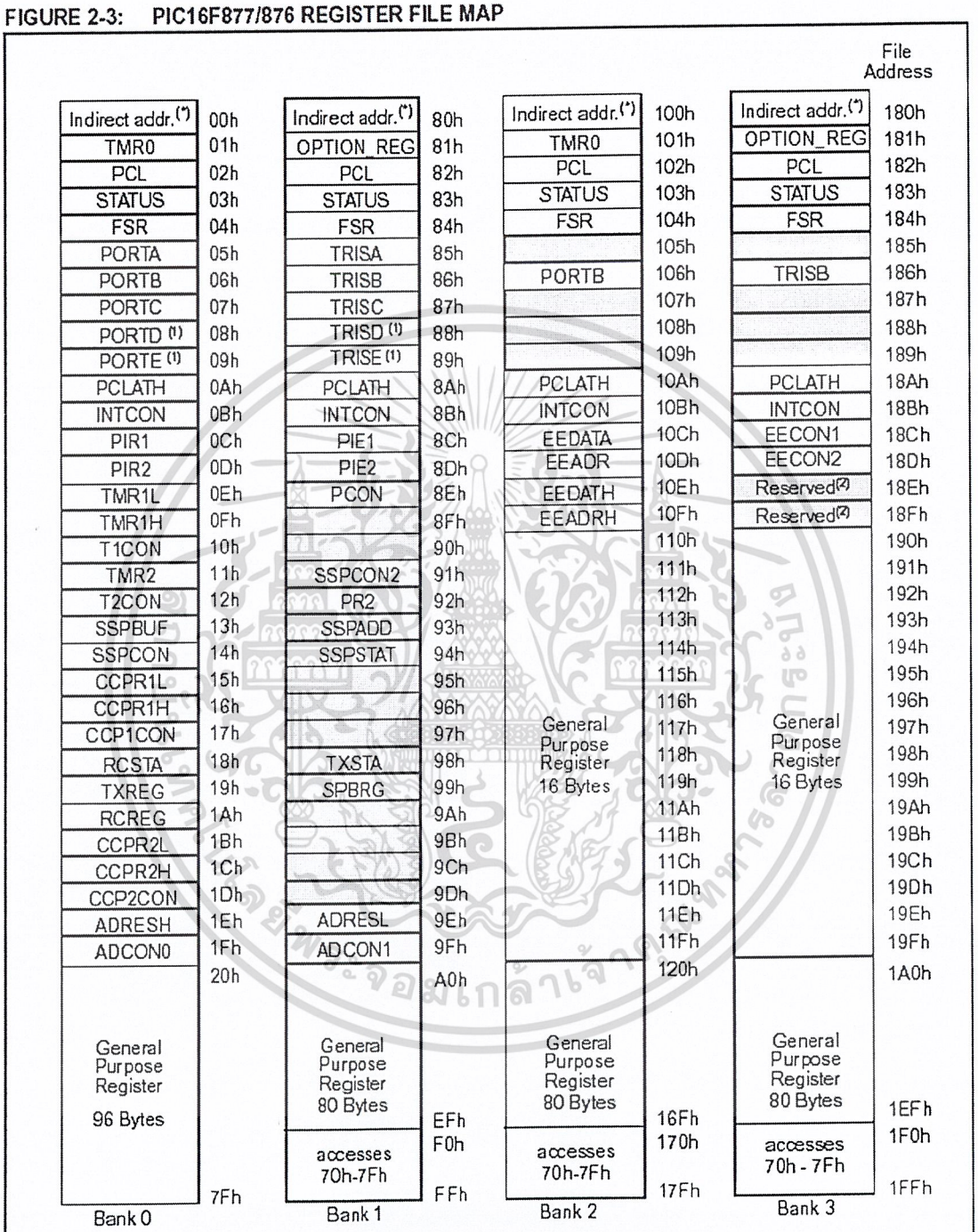
Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some "high use" Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

Note: EEPROM Data Memory description can be found in Section 4.0 of this Data Sheet

2.2.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly, or indirectly through the File Select Register FSR.

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP



Unimplemented data memory locations, read as '0'.

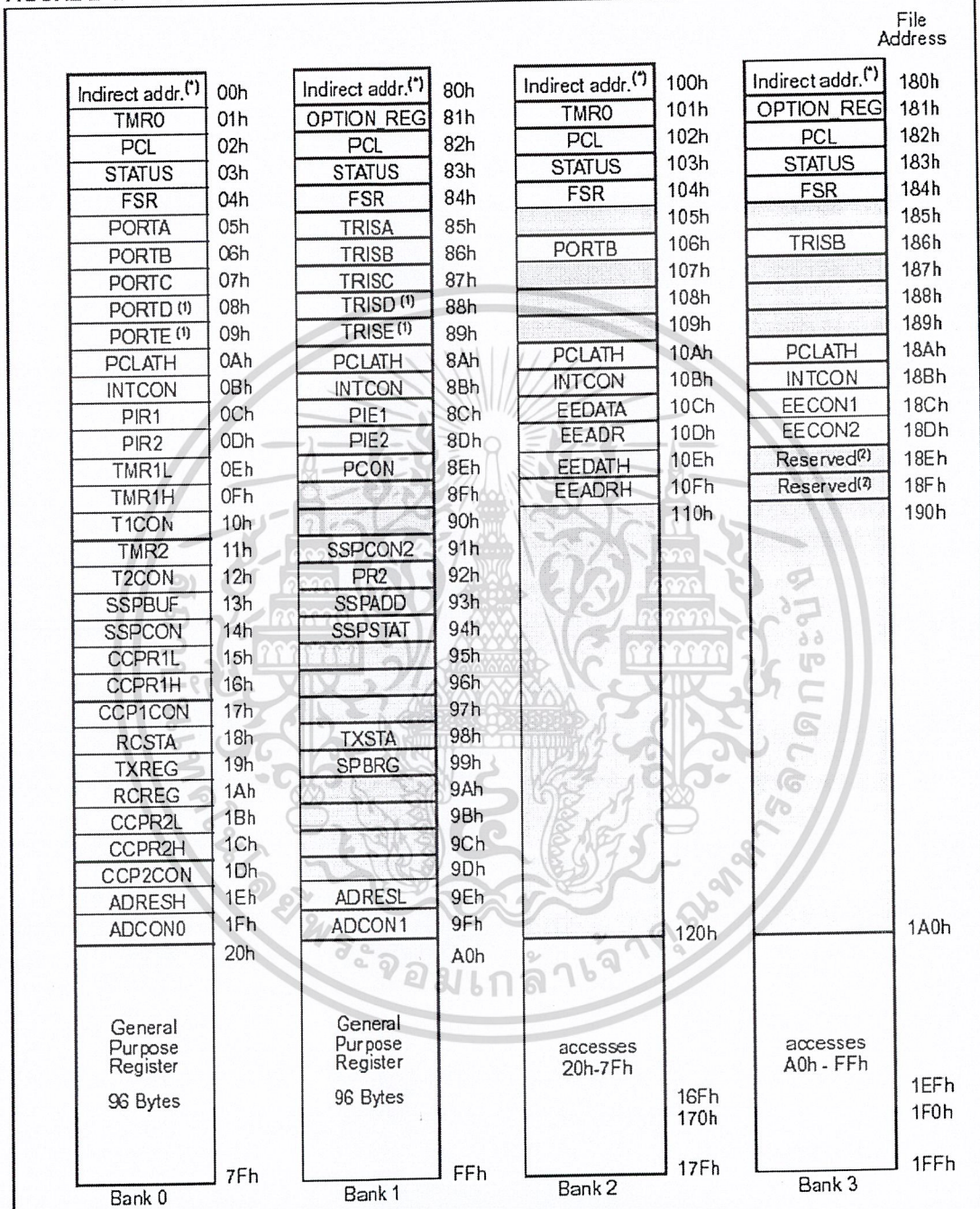
* Not a physical register.

Note 1: These registers are not implemented on 28-pin devices.

2: These registers are reserved, maintain these registers clear.

PIC16F87X

FIGURE 2-4: PIC16F874/873 REGISTER FILE MAP



Unimplemented data memory locations, read as '0'.

* Not a physical register.

Note 1: These registers are not implemented on 28-pin devices.

2: These registers are reserved, maintain these registers clear.

2.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 2-1.

The Special Function Registers can be classified into two sets; core (CPU) and peripheral. Those registers associated with the core functions are described in detail in this section. Those related to the operation of the peripheral features are described in detail in the peripheral feature section.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets (2)
Bank 0											
00h(4)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	0000 0000
01h	TMR0	Timer0 module's register								xxxx xxxx	uuuu uuuu
02h(4)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	0000 0000
03h(4)	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
04h(4)	FSR	Indirect data memory address pointer								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	uuuu uuuu
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	uuuu uuuu
08h(5)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	uuuu uuuu
09h(5)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu
0Ah(1,4)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	---0 0000
0Bh(4)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	(6)	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	-r-0 0--0
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
11h	TMR2	Timer2 module's register								0000 0000	0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
19h	TXREG	USART Transmit Data Register								0000 0000	0000 0000
1Ah	RCREG	USART Receive Data Register								0000 0000	0000 0000
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0	0000 00-0

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Other (non power-up) resets include external reset through MCLR and Watchdog Timer Reset.

3: Bits PSPIE and PSPIF are reserved on the 28-pin devices; always maintain these bits clear.

4: These registers can be addressed from any bank.

5: PORTD, PORTE, TRISD, and TRISE are not physically implemented on the 28-pin devices, read as '0'.

6: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets (2)
Bank 1											
80h(4)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	0000 0000
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h(4)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	0000 0000
83h(4)	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
84h(4)	FSR	Indirect data memory address pointer								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
86h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
88h(5)	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h(5)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111
8Ah(1,4)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	---0 0000
8Bh(4)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
8Ch	PIE1	PSPIE(3)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	(6)	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0	-r-0 0--0
8Eh	PCON	—	—	—	—	—	—	POR	BOR	---- --qq	---- --uu
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	0000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	0000 0000
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF	0000 0000	0000 0000
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	—	Unimplemented								—	—
9Dh	—	Unimplemented								—	—
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	0--- 0000

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Other (non power-up) resets include external reset through MCLR and Watchdog Timer Reset.

3: Bits PSPIE and PSPIF are reserved on the 28-pin devices; always maintain these bits clear.

4: These registers can be addressed from any bank.

5: PORTD, PORTE, TRISD, and TRISE are not physically implemented on the 28-pin devices, read as '0'.

6: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets (2)
Bank 2											
100h(4)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	0000 0000
101h	TMR0	Timer0 module's register								xxxx xxxx	uuuu uuuu
102h(4)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	0000 0000
103h(4)	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
104h(4)	FSR	Indirect data memory address pointer								xxxx xxxx	uuuu uuuu
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	uuuu uuuu
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,4)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	---0 0000
10Bh(4)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
10Ch	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
10Dh	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
10Eh	EEDATH	—	—	EEPROM data register high byte						xxxx xxxx	uuuu uuuu
10Fh	EEADRH	—	—	—	EEPROM address register high byte					xxxx xxxx	uuuu uuuu
Bank 3											
180h(4)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	0000 0000
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
182h(4)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	0000 0000
183h(4)	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
184h(4)	FSR	Indirect data memory address pointer								xxxx xxxx	uuuu uuuu
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,4)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	---0 0000
18Bh(4)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	x--- u000
18Dh	EECON2	EEPROM control register2 (not a physical register)								---- ----	---- ----
18Eh	—	Reserved maintain clear								0000 0000	0000 0000
18Fh	—	Reserved maintain clear								0000 0000	0000 0000

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Other (non power-up) resets include external reset through MCLR and Watchdog Timer Reset.

3: Bits PSPIE and PSPIF are reserved on the 28-pin devices; always maintain these bits clear.

4: These registers can be addressed from any bank.

5: PORTD, PORTE, TRISD, and TRISE are not physically implemented on the 28-pin devices, read as '0'.

6: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

DALLAS SEMICONDUCTOR

DS1307
64 X 8 Serial Real Time Clock

www.dalsemi.com

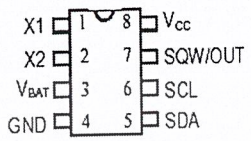
FEATURES

- Real time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap year compensation valid up to 2100
- 56 byte nonvolatile RAM for data storage
- 2-wire serial interface
- Programmable squarewave output signal
- Automatic power-fail detect and switch circuitry
- Consumes less than 500 nA in battery backup mode with oscillator running
- Optional industrial temperature range -40°C to +85°C
- Available in 8-pin DIP or SOIC
- Recognized by Underwriters Laboratory

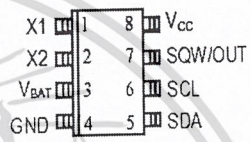
ORDERING INFORMATION

DS1307	8-Pin DIP
DS1307Z	8-Pin SOIC (150 mil)
DS1307N	8-Pin DIP (Industrial)
DS1307ZN	8-Pin SOIC (Industrial)

PIN ASSIGNMENT



DS1307 8-Pin DIP (300 mil)



DS1307Z 8-Pin SOIC (150 mil)

PIN DESCRIPTION

V_{CC}	- Primary Power Supply
X1, X2	- 32.768 kHz Crystal Connection
V_{BAT}	- +3V Battery Input
GND	- Ground
SDA	- Serial Data
SCL	- Serial Clock
SQW/OUT	- Square wave/Output Driver

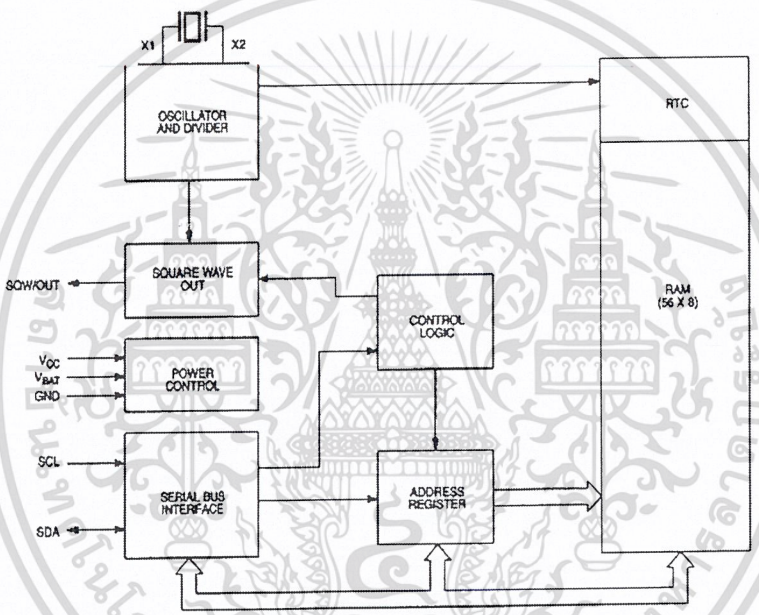
DESCRIPTION

The DS1307 Serial Real Time Clock is a low power, full BCD clock/calendar plus 56 bytes of nonvolatile SRAM. Address and data are transferred serially via a 2-wire bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit which detects power failures and automatically switches to the battery supply.

OPERATION

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT} the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When V_{CC} falls below V_{BAT} the device switches into a low current battery backup mode. Upon power up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT}+0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the Serial Real Time Clock.

DS1307 BLOCK DIAGRAM Figure 1



SIGNAL DESCRIPTIONS

V_{CC}, GND - DC power is provided to the device on these pins. V_{CC} is the +5 volt input. When 5 volts is applied within normal limits, the device is fully accessible and data can be written and read. When a 3-volt battery is connected to the device and V_{CC} is below 1.25 x V_{BAT}, reads and writes are inhibited. However, the Timekeeping function continues unaffected by the lower input voltage. As V_{CC} falls below V_{BAT} the RAM and timekeeper are switched over to the external power supply (nominal 3.0V DC) at V_{BAT}.

V_{BAT} - Battery input for any standard 3-volt lithium cell or other energy source. Battery voltage must be held between 2.0 and 3.5 volts for proper operation. The nominal write protect trip point voltage at which access to the real time clock and user RAM is denied is set by the internal circuitry as 1.25 x V_{BAT} nominal. A lithium battery with 48 mAhr or greater will back up the DS1307 for more than 10 years in the absence of power at 25 degrees C.

SCL (Serial Clock Input) - SCL is used to synchronize data movement on the serial interface.

SDA (Serial Data Input/Output) - SDA is the input/output pin for the 2-wire serial interface. The SDA pin is open drain which requires an external pullup resistor.

SQW/OUT (Square Wave/ Output Driver) - When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square wave frequencies (1 Hz, 4 kHz, 8 kHz, 32 kHz). The SQW/OUT pin is open drain which requires an external pullup resistor. SQW/OUT will operate with either Vcc or Vbat applied.

X1, X2 - Connections for a standard 32.768 kHz quartz crystal. The internal oscillator circuitry is designed for operation with a crystal having a specified load capacitance (CL) of 12.5 pF.

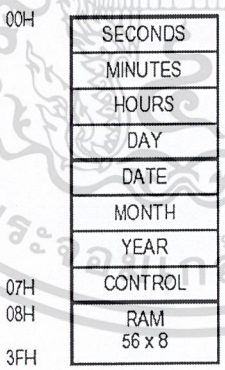
For more information on crystal selection and crystal layout considerations, please consult Application Note 58, "Crystal Considerations with Dallas Real Time Clocks." The DS1307 can also be driven by an external 32.768 kHz oscillator. In this configuration, the X1 pin is connected to the external oscillator signal and the X2 pin is floated.

Please review Application Note 95, "Interfacing the DS1307 with a 8051-Compatible Microcontroller" for additional information.

RTC AND RAM ADDRESS MAP

The address map for the RTC and RAM registers of the DS1307 is shown in Figure 2. The real time clock registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multi-byte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

DS1307 ADDRESS MAP Figure 2



CLOCK AND CALENDAR

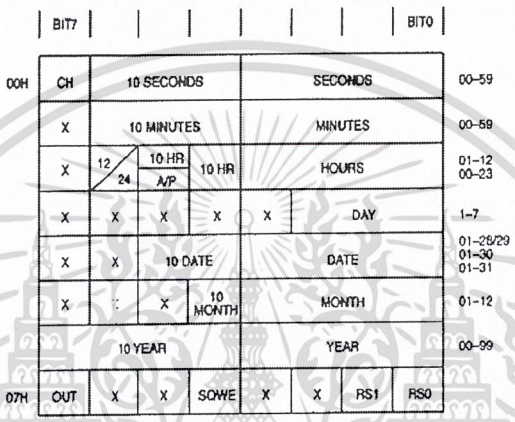
The time and calendar information is obtained by reading the appropriate register bytes. The real time clock registers are illustrated in Figure 3. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the Binary-Coded Decimal (BCD) format. Bit 7 of Register 0 is the Clock Halt (CH) bit. When this bit is set to a 1, the oscillator is disabled. When cleared to a 0, the oscillator is enabled.

Please note that the initial power on state of all registers is not defined. Therefore it is important to enable the oscillator (CH bit=0) during initial configuration.

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10 hour bit (20-23 hours).

On a 2-wire START, the current time is transferred to a second set of registers. The time information is read from these secondary registers, while the clock may continue to run. This eliminates the need to re-read the registers in case of an update of the main registers during a read.

DS1307 TIMEKEEPER REGISTERS Figure 3



CONTROL REGISTER

The DS1307 Control Register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	X	X	SQWE	X	X	RS1	RS0

OUT (Output control): This bit controls the output level of the SQW/OUT pin when the square wave output is disabled. If SQWE=0, the logic level on the SQW/OUT pin is 1 if OUT=1 and is 0 if OUT=0.

SQWE (Square Wave Enable): This bit, when set to a logic 1, will enable the oscillator output. The frequency of the square wave output depends upon the value of the RS0 and RS1 bits.

RS (Rate Select): These bits control the frequency of the square wave output when the square wave output has been enabled. Table 1 lists the square wave frequencies that can be selected with the RS bits.

SQUAREWAVE OUTPUT FREQUENCY Table 1

RS1	RS0	SQW OUTPUT FREQUENCY
0	0	1 Hz
0	1	4.096 kHz
1	0	8.192 kHz
1	1	32.768 kHz

ประวัติผู้แต่ง

ชื่อ-สกุล นางสาวจิรายุ ธิป้อ

วัน เดือน ปีเกิด 18 มีนาคม 2523

ภูมิลำเนา 16 หมู่ 3 ตำบลหัวเสือ อำเภอแม่ทะ
จังหวัด ลำปาง 52150 โทรศัพท์ 01-6586115

ประวัติการศึกษา

ประถมศึกษา โรงเรียนบ้านดอนมูล จังหวัดลำปาง

มัธยมศึกษาตอนต้น โรงเรียนห้วยมะเกลือวิทยา จังหวัดลำปาง

ประกาศนียบัตรวิชาชีพ วิทยาลัยเทคนิคลำปาง

ประกาศนียบัตรวิชาชีพชั้นสูง สถาบันเทคโนโลยีราชมงคลวิทยาเขตตาก

ปริญญาตรี สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสาตร์วิศวกรรม
คณะครุศาสตร์อุตสาหกรรม สจล.

คติพจน์ ลำบากวันนี้ดีกว่าลำบากวันหน้า

ประวัติผู้แต่ง

ชื่อ-สกุล นายทศพล วงษ์วิบูลย์สิน

วัน เดือน ปีเกิด 5 มกราคม 2524

ภูมิลำเนา 688/72 ซอย จรัญสนิทวงศ์ 68 เขต บางพลัด
แขวง บางพลัด กรุงเทพฯ 10700 โทรศัพท์ 06-8107374

ประวัติการศึกษา

ประถมศึกษา โรงเรียนบูรณวิทย์ จังหวัดกรุงเทพฯ

มัธยมศึกษาตอนต้น โรงเรียนนวมินทราชูทิศ สตรีวิทยาพุทธมลฑล
จังหวัดกรุงเทพฯ

ประกาศนียบัตรวิชาชีพ โรงเรียนเทคโนโลยีสยาม

ประกาศนียบัตรวิชาชีพชั้นสูง สถาบันเทคโนโลยีราชมงคลวิทยาเขตนนทบุรี

ปริญญาตรี สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสาตร์วิศวกรรม
คณะครุศาสตร์อุตสาหกรรม สจล.

คติพจน์ พรุ่งนี้ต้องดีกว่าวันนี้

ประวัติผู้แต่ง

ชื่อ-สกุล	นายธีระภัทร์ สายไตร
วัน เดือน ปีเกิด	1 มีนาคม 2523
ภูมิลำเนา	10/17 หมู่ 11 แขวง หนองค้างพลู เขต หนองแขม กรุงเทพฯ 10160 โทรศัพท์ 06-6274509

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนวีรสุนทร จังหวัดกรุงเทพฯ
มัธยมศึกษาตอนต้น	โรงเรียนปัญญาวรคุณ จังหวัดกรุงเทพฯ
ประกาศนียบัตรวิชาชีพ	-
ประกาศนียบัตรวิชาชีพชั้นสูง	โรงเรียนอินเตอร์เทคโนโลยี
ปริญญาตรี	สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม สจล.

คติพจน์ ทำวันนี้ให้ดีที่สุด