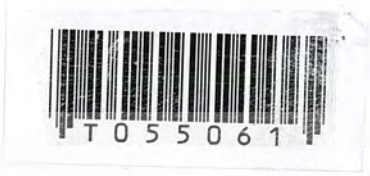
การออกแบบเครื่องบันทึกสัญญาณโดยคอมพิวเตอร์ส่วนบุคคล

DESIGN OF RECORDER USING PERSONAL COMPUTER



นาย ชยวี คุณโลกย์
นาย ธเนศ จงศุภนิมิต
นาย วุฒิชัย แก้วประเสริฐ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมการวัดคุม
ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2546

DESIGN OF RECORDER USING PERSONAL COMPUTER

CHAYAWEE KHUNNALOK

TANATE JONGSUPANIMIT

WUTTICHAI KEAWPRASERT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ การออกแบบเครื่องบันทึกสัญญาณโดยคอมพิวเตอร์ส่วนบุคคล

DESIGN OF RECORDER USING PERSONAL COMPUTER

นักศึกษาผู้จัดทำ

นาย ชยวี	คุณโลกย์	รหัสประจำตัว	44015423
นาย ธเนศ	จงสุกนิมิต	รหัสประจำตัว	44015427
นาย วุฒิชัย	แก้วประเสริฐ	รหัสประจำตัว	44015445

ปริญญา วิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมการวัดคุม

ปีการศึกษา 2546

อาจารย์ผู้ควบคุมปริญญานิพนธ์	**ลายมือชื่อ**
รศ.ประสิทธิ์ จุลเสรีวงศ์	
อ.อัมพวัน ใจกล้า	

วัน/เดือน/ปี ที่สอบ วันพุธที่ 24 มีนาคม พ.ศ. 2547

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(รศ.ประสิทธิ์ จุลเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	การออกแบบเครื่องบันทึกสัญญาณโดยคอมพิวเตอร์ส่วนบุคคล	
	DESIGN OF RECORDER USING PERSONAL COMPUTER	
นักศึกษาผู้จัดทำ	นาย ชยวี	คุณโลกย์
	นาย ธเนศ	จงศุภนิมิต
	นาย วุฒิชัย	แก้วประเสริฐ
อาจารย์ที่ปรึกษา	รศ.ประสิทธิ์	จุลเสรีวงศ์
	อ.อัมพวัน	ใจกล้า
ปีการศึกษา	2546	

บทคัดย่อ

การออกแบบเครื่องบันทึกสัญญาณเพื่อบันทึกค่าสัญญาณอนาลอกที่ได้จากตัวตรวจจับโดยใช้คอมพิวเตอร์ส่วนบุคคลนั้น สามารถลดข้อจำกัดในการใช้เครื่องบันทึกสัญญาณ แบบที่เคยใช้มาในอดีต ซึ่งทำให้การใช้งานไม่สามารถตอบสนองความต้องการของผู้ใช้งานได้ทั้งหมด ดังนั้นการออกแบบเครื่องบันทึกสัญญาณด้วยตัวของผู้ใช้เอง จึงเป็นทางเลือกให้สามารถออกแบบการทำงานของเครื่องนั้นให้เป็นไปตามความต้องการและมีประสิทธิภาพ ทำให้ความสามารถในการเก็บข้อมูลทำได้สะดวกเพราะสามารถออกแบบให้คลอบคลุมการทำงานได้อย่างเหมาะสม โดยสามารถปรับแต่งฐานเวลาในการบันทึกข้อมูล และในขณะที่บันทึกข้อมูลสามารถทำการเรียกค่าข้อมูลดูได้จากหน้าจอของคอมพิวเตอร์ เพื่อประโยชน์ในการหาข้อผิดพลาดของตัวอุปกรณ์ที่เราต้องการตรวจวัด นอกจากนั้นเราสามารถนำมาใช้ในการสอบเทียบอุปกรณ์ได้อีกด้วย โดยสามารถนำข้อมูลที่ได้นั้นให้แสดงผลออกทางเครื่องพิมพ์ได้

Thesis Title	Design Of Recorder Using Personal Computer
Authors	Mr.Chayawee Khunnalok
	Mr.Tanate Jongsupanimit
	Mr.Wuttichai Keawprasert
Thesis Advisor	Assoc.Prof.Prasit Julsereewong
	Miss.Amphawan Chaikla
Year	2003

ABSTRACT

To design the signal recorder for recording analog signal verified by personal computer can reduce the limitation of using former signal recorder, but cannot respond need to all of the users. If the users can perfectly design the signal recorder by themselves, it is very useful, and convenient to record the data. The signal recorder can also adapt timebase in recording. While recording, we can read the data at the computer's screen in the same time. This process is also useful for checking the error in the equipment. Moreover, we can calibrate the equipment, and also display the data by printing.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี เพราะได้รับความเมตตาจาก รองศาสตราจารย์ ประสิทธิ์ จุลเสรีวงศ์ และ อาจารย์อัมพวัน ใจกล้า ที่ได้ให้คำแนะนำแก่คณะผู้วิจัยตลอดมา อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่างๆ ที่จำเป็นในการทำปริญญานิพนธ์นี้ทางคณะผู้วิจัยรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณอาจารย์ภาควิชาวิศวกรรมการวัดคุมทุกท่านที่ให้คำแนะนำอันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

และที่ลืมเสียมิได้คือ ขอกราบขอบพระคุณคุณพ่อและคุณแม่ อันเป็นที่รักยิ่งที่สนับสนุนและเป็นแรงบันดาลใจในการทำปริญญานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ทางคณะผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 กล่าวนำ

ในการบันทึกค่าต่าง ๆ ที่สามารถวัดได้ในกระบวนการผลิตของระบบอุตสาหกรรมนิยมใช้เครื่องบันทึกค่า (Data recorder) ในการแสดงผล และเครื่องเก็บข้อมูล (Data logger) ในการเก็บค่าที่วัดได้ แต่เนื่องจากอุปกรณ์ทั้งสองยังมีข้อจำกัดในเรื่องราคาที่ค่อนข้างสูงและข้อจำกัดในการใช้งาน จึงนำคอมพิวเตอร์ส่วนบุคคล (PC) มาประยุกต์ใช้ทดแทนข้อจำกัดของอุปกรณ์ทั้งสอง

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้จะเป็นการศึกษาและพัฒนาออกแบบ สร้างเครื่องบันทึกสัญญาณโดยคอมพิวเตอร์ส่วนบุคคลให้สามารถแสดงค่าต่าง ๆ ในกระบวนการอุตสาหกรรมได้ นอกจากนั้นยังสามารถที่จะบันทึกข้อมูลที่แสดงผลทางหน้าจอคอมพิวเตอร์ได้อีกด้วย อีกทั้งยังสามารถที่จะเรียกข้อมูลที่ได้บันทึกไว้ในคอมพิวเตอร์กลับมาตรวจสอบได้ เนื่องจากเครื่องบันทึกค่า (Data recorder) และเครื่องเก็บข้อมูล (Data logger) เป็นอุปกรณ์ที่มีราคาแพง รวมถึงการเชื่อมต่อทำได้ยุ่งยาก

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ กล่าวถึงการศึกษาอุปกรณ์ที่ทำหน้าที่ในการแปลงสัญญาณอนาลอกเป็นดิจิตอล (A/D) การเชื่อมต่อกับไมโครคอนโทรลเลอร์ และส่วนเชื่อมต่อกับคอมพิวเตอร์ รวมถึงส่วนประกอบและการทำงานของโปรแกรมที่ใช้ในการแสดงค่าและเก็บบันทึกข้อมูล

1.4 รายละเอียดของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ จะมีขั้นตอนการศึกษาที่เริ่มจากการศึกษาการทำงาน ของตัวแปลงสัญญาณอนาลอกเป็นดิจิตอล (A/D) อีกทั้งยังรวมถึงการเชื่อมต่อไมโครคอนโทรลเลอร์กับตัวแปลงสัญญาณ ตลอดจนในส่วนของโปรแกรมที่ใช้ในการแสดงค่าและเก็บบันทึกข้อมูล

บทที่ 2
ไมโครคอนโทรลเลอร์ PIC16F877

ในปัจจุบันนี้มีไมโครคอนโทรลเลอร์หลายรุ่นหลายตระกูล ถูกผลิตขึ้นเพื่อตอบสนองการใช้งานที่แตกต่างกัน โดยมีความแตกต่างทั้งความเร็วสัญญาณนาฬิกา ความยาวเวิร์ด ข้อมูลขนาดของหน่วยความจำ จำนวนอินพุท เอาท์พุท เอาท์พุทพอร์ต แต่โดยทั่วไปไมโครคอนโทรลเลอร์สามารถจำแนกออกได้เป็น 2 กลุ่ม ตามสถาปัตยกรรมที่ใช้คือ ไมโครคอนโทรเลอร์แบบ CISC (Complex Instruction Set Computer) และ ไมโครคอนโทรลเลอร์แบบ RISC (Reduce Instruction Set Computer) โดย CISC เช่น MCS-51 จะมีจุดเด่นตรงที่ มีจำนวนชุดคำสั่งมาก ซึ่งแต่ละคำสั่งจะใช้จำนวนสัญญาณนาฬิกาต่างกัน ส่วน RISC มีชุดคำสั่งน้อยกว่าโดยชุดคำสั่งถูกจัดให้มีโครงสร้างการทำงานคล้ายกัน ทำให้สามารถทำงานแบบไปป์ไลน์ได้ โดยใช้ สัญญาณนาฬิกาเพียงหนึ่งลูกต่อหนึ่งคำสั่ง ดังนั้นความเร็วของ RISC จะสูงกว่า CISC ประมาณสองถึงสี่เท่าที่ความเร็วสัญญาณนาฬิกาเดียวกัน แต่ก็อาจจะไม่เป็นเช่นนั้นเสมอไป เนื่องจากชุดคำสั่งที่น้อยกว่าทำให้ RISC ต้องมีคำสั่งมากกว่าหนึ่งคำสั่งเพื่อทำงานอย่างเดียวกัน CISC ที่ใช้เพียงคำสั่งเดียวนอกจากนี้ชุดคำสั่งที่ง่ายกว่าของ RISC จึงส่งผลทำให้ขนาดของวงจรในชิปเล็กกว่าการเพิ่มวงจรพิเศษ เช่น วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล

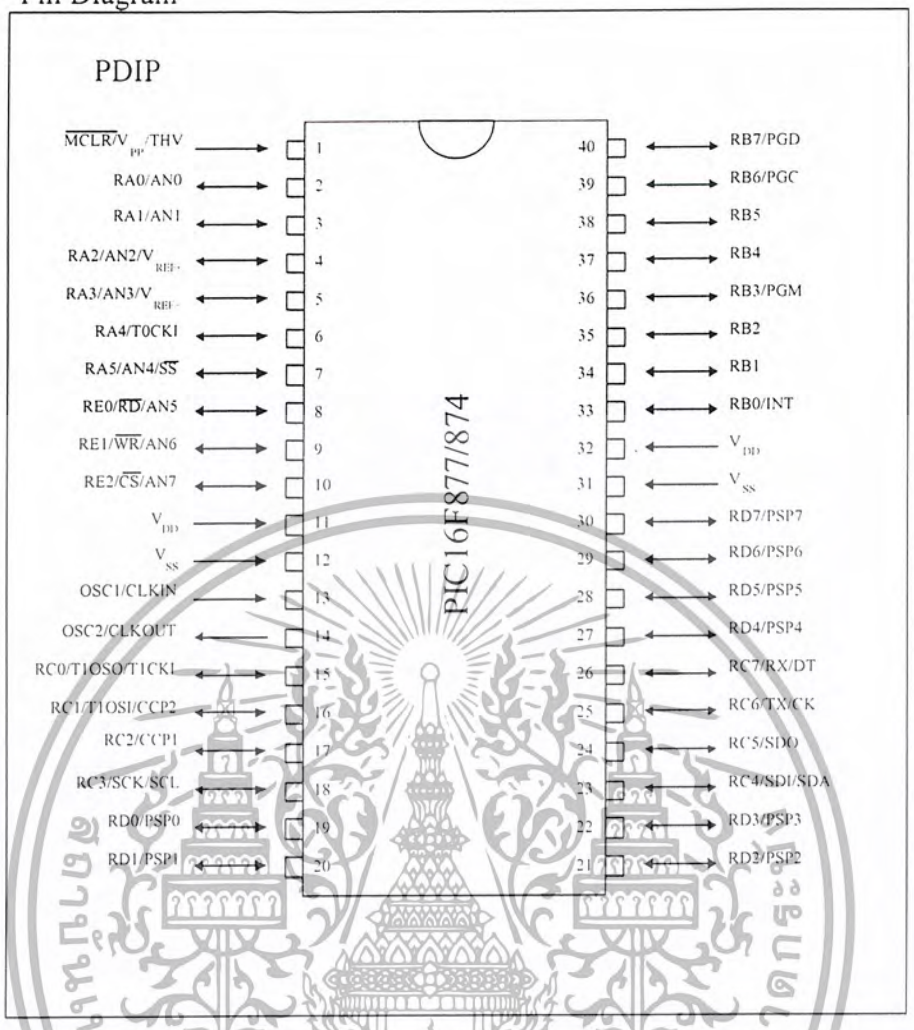
ไมโครคอนโทรลเลอร์ตระกูล PIC ของบริษัทไมโครชิพเทคโนโลยี จะใช้สถาปัตยกรรมแบบ RISC จะแตกต่างจากไมโครคอนโทรลเลอร์ตระกูล 8051 ที่เป็นสถาปัตยกรรมแบบ CISC

สำหรับตัวไมโครคอนโทรลเลอร์เบอร์ PIC16F877 ออกสู่ตลาดครั้งแรกในปี ค.ศ. 1998 มีหน่วยความจำโปรแกรมแบบแฟลช ขนาด 8 กิโลเวิร์ด นอกจากนั้นยังมีไมโครคอนโทรลเลอร์ในรุ่นเดียวกันอีก 3 เบอร์ คือ PIC16F873 , PIC16F874 และ PIC16F876 ซึ่งจะมีขนาดหน่วยความจำโปรแกรม และจำนวนพอร์ทอินพุท / เอาท์พุทแตกต่างกันไป

2.1 คุณสมบัติของไมโครคอนโทรลเลอร์ PIC16F877

ไมโครคอนโทรเลอร์เบอร์ PIC16F877 เป็นผลิตภัณฑ์ ของบริษัทไมโครชิพเทคโนโลยีเป็นไมโครคอนโทรลเลอร์แบบ RISC มีโมดูลสำหรับจัดการสิ่งต่าง ๆ หลายส่วน อาทิเช่น วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล (A/D) ส่วนที่ติดต่อกับบัสแบบ I^2C รวมถึงส่วนที่ติดต่อส่งข้อมูลอนุกรมอะซิงโครนัส UART สามารถทำงานที่สัญญาณนาฬิกาสูงสุดถึง 20 เมกกะเฮิร์ทซ์ ที่กระแส 7 มิลลิแอมป์ ดังภาพที่ 2.1 คือ ลักษณะภายนอกพร้อมคำอธิบายตำแหน่งขาของการต่อใช้งาน

Pin Diagram



ภาพที่ 2.1 ไมโครคอนโทรลเลอร์ PIC16F877

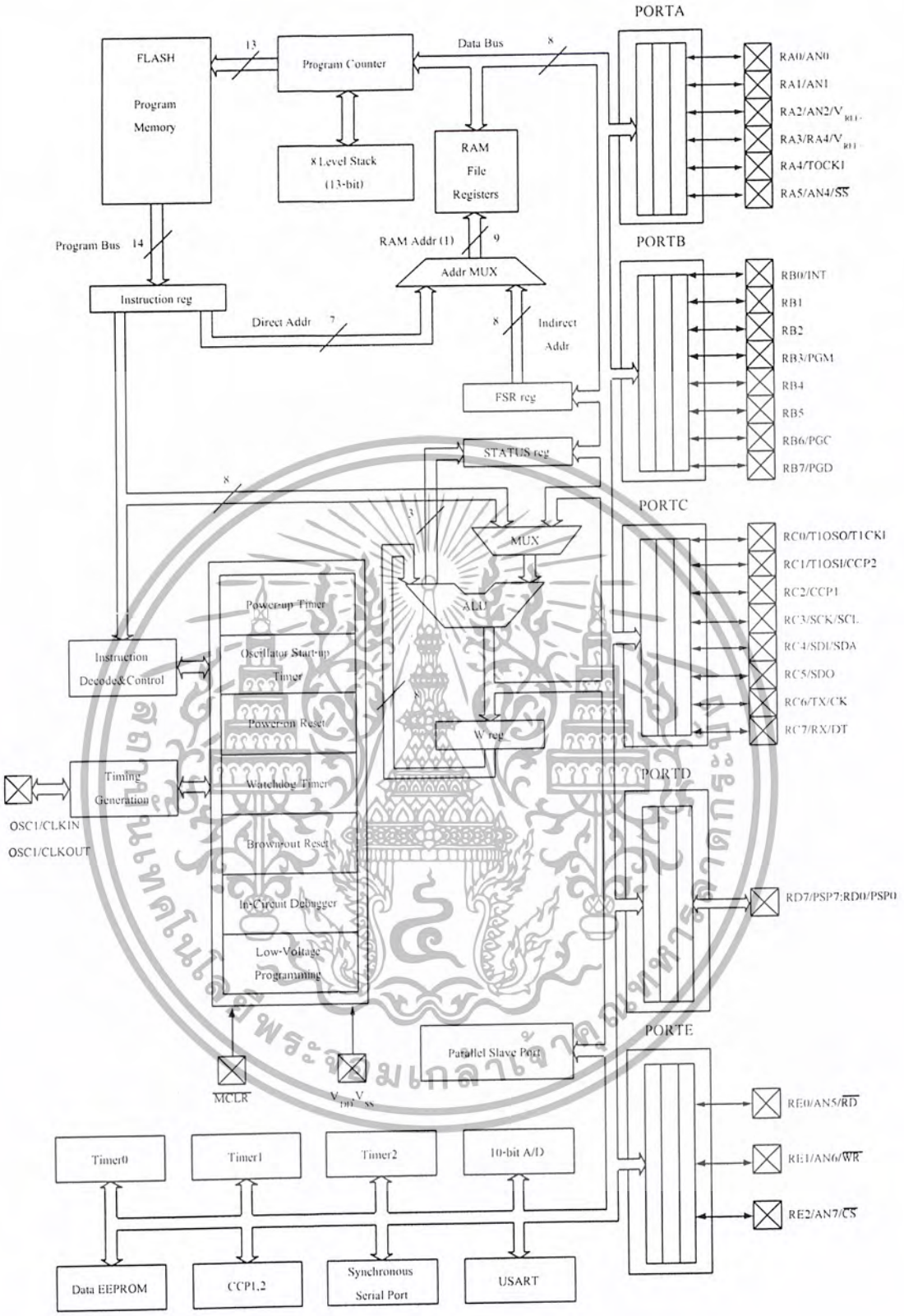
ส่วนสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ PIC16F877 ดูได้จากภาพที่ 2.2 ซึ่งจะประกอบไปด้วย โครงสร้างภายในและแบ่งออกเป็นส่วนการทำงานแต่ละส่วนพร้อมรายละเอียดในหัวข้อต่อไป

2.2 สถาปัตยกรรมแบบ RISC (Reduce Instruction Set computer)

แบ่งออกเป็นส่วนการทำงานแต่ละส่วนพร้อมรายละเอียดดังนี้

1. ใช้การประมวลผลแบบไปป์ไลน์
2. ทุกคำสั่งประมวลผลเสร็จภายใน 1 รอบสัญญาณนาฬิกายกเว้นการเรียกโปรแกรมย่อย ซึ่งใช้ 2 รอบสัญญาณนาฬิกา
3. ความเร็วสัญญาณนาฬิกาสูงสุด 20 เมกะเฮิร์ทซ์

4. บัสคำสั่งขนาด 14 บิทแยกอิสระกับบัสข้อมูลขนาด 8 บิท
5. หน่วยความจำโปรแกรมแบบแฟลชขนาด 8 กิโลเวิร์ด (1 เวิร์ดของคำสั่งมีขนาด 14บิท)
6. หน่วยความจำข้อมูลแบบ SRAM ขนาด 368 ไบท์
7. หน่วยความจำข้อมูลแบบ EEPROM ขนาด 256 ไบท์
8. สามารถอ้างอิงแอดเดรสแบบโดยตรงโดยอ้อมและแบบสัมพัทธ์
9. มีฮาร์ดแวร์แสตก 8 ระดับ
10. อินเตอร์รัพท์สูงสุด 14 แหล่ง
11. อินพุท / เอาท์พุทพอร์ทรวม 32 บิท (พอร์ท A,B,C,D,E)
12. ตัวจับเวลา / ตัวนับขนาด 8 บิท 2 ตัว และ ขนาด 16 บิท 1 ตัววงจรตรวจจับเปรียบเทียบและกำเนิดสัญญาณแบบ PWM 2 ตัว
13. สนับสนุนการสื่อสารอนุกรมแบบ Synchronous Serial Port , I2C (Master/Slave) และ Universal Synchronous Asynchronous Receiver Transmitter (USART)
14. สนับสนุนการสื่อสารแบบขนาน ขนาด 8 บิท
15. วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลความละเอียด 10 บิท 8 ช่องสัญญาณ
16. สามารถจะโปรแกรมไมโครคอนโทรลเลอร์ที่อยู่ภายในระบบได้โดยตรงผ่านพอร์ท 2 พอร์ท (In Circuit Serial Programming : ICSP)
17. ระบบรีเซตอัตโนมัติ เมื่อเริ่มจ่ายกระแสไฟฟ้าเข้าสู่ไมโครคอนโทรลเลอร์ (Power On Reset :POR)
18. มีการหน่วงเวลารอไฟเลี้ยง และ ออสซิลเลเตอร์ ให้อยู่ในสภาวะที่เสถียร (Power Up Timer : PWRT,Oscillator Start Up Timer : OST)
19. ใช้กระแสน้อยกว่า 2 มิลลิแอมป์ที่ 4 เมกะเฮิร์ทซ์ และ น้อยกว่า 1 ไมโครแอมป์เมื่ออยู่ในโหมด Sleep
20. สามารถป้องกันการอ่านโปรแกรมจากไอซีได้
21. จำนวนรอบการเขียนหน่วยความจำโปรแกรมแบบแฟลช 1000 ครั้งและรอบการเขียนหน่วยความจำข้อมูลแบบ EEPROM 10,000,000 ครั้ง



ภาพที่ 2.2 สถาปัตยกรรมภายในของ PIC16F877

ตารางที่ 2.1 แสดงการจัดขาและหน้าที่ของแต่ละขาใน PIC16F877

ชื่อขา	ตำแหน่งขา	รูปแบบ	อธิบายหน้าที่
OSC1/CLKIN	13	I	สัญญาณออสซิลเลเตอร์ขาเข้า
OSC2/CLKOUT	14	O	สัญญาณออสซิลเลเตอร์ขาออก
MCRL/Vpp/THV	1	I/P	รีเซตเมื่อเป็นลอจิกต่ำ/แรงดันสูงเพื่อโปรแกรม IC
			พอร์ท A เป็นพอร์ท 2 ทิศทาง
RA0/AN0	2	I/O	RA0 เป็นอนาลอกอินพุทช่อง 0
RA1/AN1	3	I/O	RA1 เป็นอนาลอกอินพุทช่อง 1
RA2/AN2/Vref-	4	I/O	RA2 เป็นอนาลอกอินพุทช่อง 2 หรือแรงดันอ้างอิง
RA3/AN3/Vref+	5	I/O	ลบ
RA4/T0CKI	6	I/O	RA3 เป็นอนาลอกอินพุทช่อง 3 หรือแรงดันอ้างอิง
RA5/SS/AN4	7	I/O	บวก
			RA5 เป็นอนาลอกอินพุทช่อง 4
			พอร์ท B เป็นพอร์ท 2 ทิศทางโปรแกรมให้มีตัวต้านทานพูลอัพได้
RB0/INT	33	I/O	RB0 เป็นขารับอินเตอร์รัพท์ภายนอก
RB1	34	I/O	
RB2	35	I/O	
RB3/PGM	36	I/O	RB3 เป็นขาโปรแกรมแรงดันต่ำ
RB4	37	I/O	RB4 สร้างอินเตอร์รัพท์เมื่อขาเปลี่ยนสถานะ
RB5	38	I/O	RB5 สร้างอินเตอร์รัพท์เมื่อขาเปลี่ยนสถานะ
RB6/PGC	39	I/O	RB6 สร้างอินเตอร์รัพท์เมื่อขาเปลี่ยนสถานะ หรือขารับสัญญาณนาฬิกาสำหรับการโปรแกรม IC
RB7/PGD	40	I/O	RB7 สร้างอินเตอร์รัพท์เมื่อขาเปลี่ยนสถานะหรือขารับสัญญาณข้อมูลสำหรับการโปรแกรม IC

ตารางที่ 2.1 (ต่อ)

			พอร์ท C เป็นพอร์ท 2 ทิศทาง
RC0/T10SO/T1C	15	I/O	
LK	16	I/O	RC3 เป็นขาสัญญาณนาฬิกาสำหรับการสื่อสารแบบ
RC1/T1OSI/CCP	17	I/O	I^2C
2	18	I/O	RC3 เป็นขาสัญญาณข้อมูลสำหรับการสื่อสารแบบ
RC2/CCP1	23	I/O	I^2C
RC3/SCK/SCL	24	I/O	RC6 เป็นขาส่งข้อมูลสำหรับ USART แบบอะซิง
RC4/SDI/SDA	25	I/O	โครนัส
RC5/SDO	26	I/O	RC7 เป็นขาส่งข้อมูลสำหรับ USART แบบอะซิง
RC6/TX/CK			โครนัส
RC7/RX/DT			
			พอร์ท D เป็นพอร์ท 2 ทิศทางหรือพอร์ทขนาน
RD0/PSP0	19	I/O	
RD0/PSP0	20	I/O	
RD0/PSP0	21	I/O	
RD0/PSP0	22	I/O	
RD0/PSP0	27	I/O	
RD0/PSP0	28	I/O	
RD0/PSP0	29	I/O	
RD0/PSP0	30	I/O	
			พอร์ท E เป็นพอร์ท 2 ทิศทาง
RE0/RD/AN5	8	I/O	RE0 เป็นสัญญาณอ่าน สำหรับพอร์ทขนาน หรือ เป็นอนาลอกอินพุท ช่อง 5
RE1/WR/AN6	9	I/O	RE1 เป็นสัญญาณเขียน สำหรับพอร์ทขนาน หรือ เป็นอนาลอกอินพุทช่อง 6
RE2/CS/AN7	10	I/O	RE2 เป็นสัญญาณเลือก สำหรับพอร์ทขนาน หรือ เป็นอนาลอกอินพุทช่อง 7
Vss	12,31	P	กราวนด์
Vdd	11,32	P	แรงดันบวก

I = Input , O = Output , P = Power

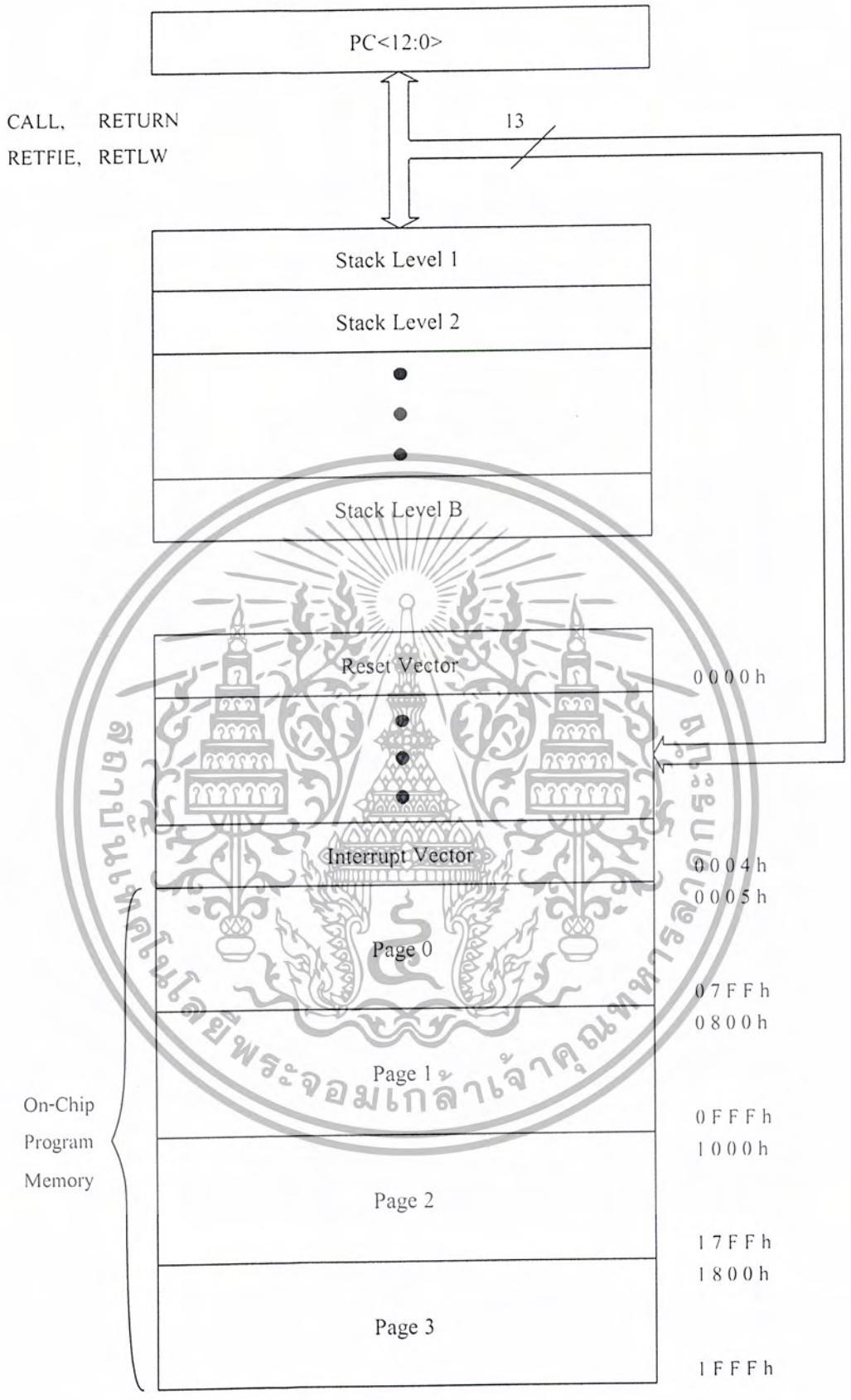
2.3 การจัดการหน่วยความจำ

หน่วยความจำของไมโครคอนโทรเลอร์ PIC16F877 สามารถที่จะแบ่งได้ 3 ส่วนคือ หน่วยความจำโปรแกรม หน่วยความจำข้อมูล หรือ รีจิสเตอร์ และหน่วยความจำข้อมูลแบบ EEPROM ซึ่งข้อมูลจะไม่สูญหายแม้ไม่จ่ายไฟเลี้ยง

บัสของหน่วยความจำโปรแกรม และ หน่วยความจำข้อมูลจะถูกแยกออกจากกัน ดังนั้นการเข้าถึงหน่วยความจำทั้งสองแบบสามารถเกิดขึ้นได้พร้อมกัน

2.3.1 หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมขนาด 8 กิโลเวิร์ดจะถูกแบ่งออกเป็น 4 เพจ เพจละ 2 กิโลเวิร์ดโดยมีตำแหน่งของรีเซตเวคเตอร์อยู่ที่ 0000h และตำแหน่งของอินเตอร์รัพท์เวคเตอร์อยู่ที่ 0004h ดูภาพที่ 2.3 ประกอบ

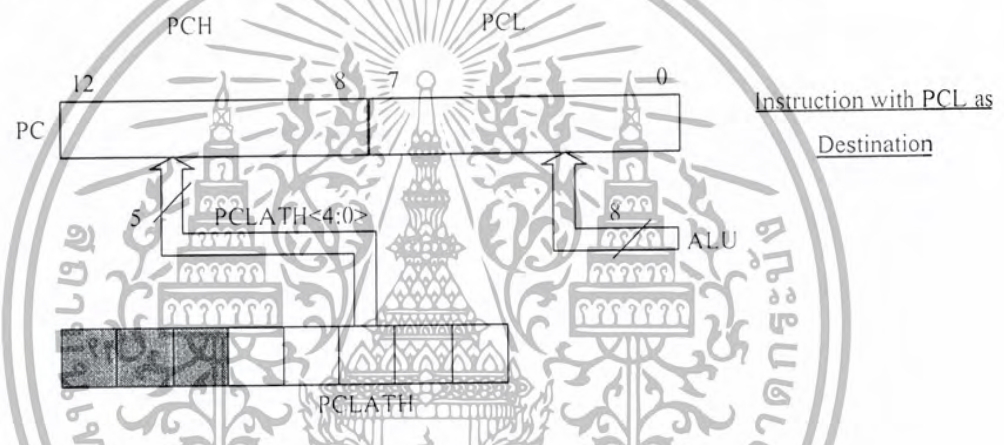


ภาพที่ 2.3 แผนผังหน่วยความจำโปรแกรมและแสตกของ PIC16F877

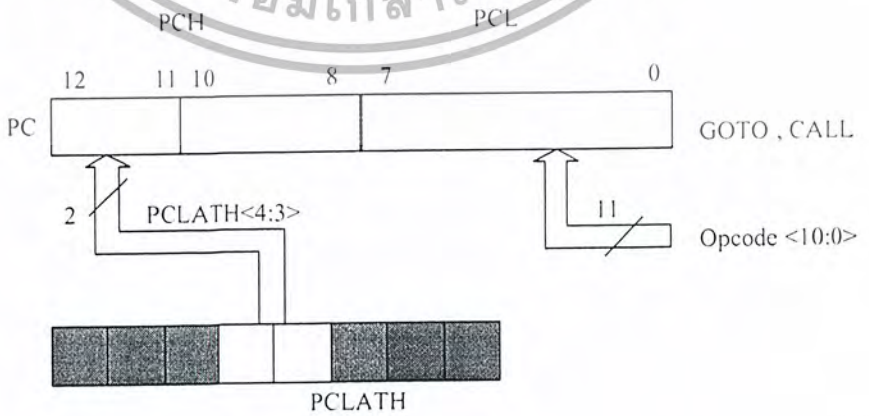
2.3.2 โปรแกรมเคาน์เตอร์ (Program counter : PC) และแสตก (Stack)

โปรแกรมเคาน์เตอร์มีขนาด 13 บิท ได้ไบท์ต่ำได้จากรีจิสเตอร์ PCL ซึ่งสามารถอ่านและเขียนได้ ส่วนไบท์สูง (PC<12:8>) ไม่สามารถอ่านได้แต่สามารถเขียนได้โดยอ้อมผ่านทางรีจิสเตอร์ PCLATH เมื่อเกิดการรีเซตรีจิสเตอร์ทั้งสองจะกลายเป็นศูนย์

จากภาพที่ 2.4 แสดงการใส่ค่าลงในโปรแกรมเคาน์เตอร์ในสองสถานการณ์ รูปแรกค่าไบท์ต่ำได้จาก PCL และ ไบท์สูงได้จาก PCLATH<4:0> ส่วนรูปที่สองนั้นเกิดจากการทำคำสั่ง GOTO หรือ CALL ซึ่ง 11 บิทล่างของโปรแกรมเคาน์เตอร์ได้จาก Opcode <10:0> ส่วน 2 บิทบนนั้นจะได้มาจาก PCLATH<4:3> ทำให้การกระโดดทำได้ภายใน 2048 ตำแหน่งในเพจเดียวกันเท่านั้น การอ้างถึงตัวแอดเดรสที่ไกลกว่านี้ทำได้โดยเปลี่ยนแปลงค่าใน PCLATH<4:3> ซึ่งเป็นค่าที่จะถูกใส่เอาไว้ในตัวโปรแกรมเคาน์เตอร์ 2 บิทบน ทำให้สามารถอ้างแอดเดรสได้ตลอด 8192 ตำแหน่งแต่ถ้าเป็นการอ้างแอดเดรสที่สูงกว่ากำหนดจะทำให้เกิดการวนซ้ำตำแหน่งขึ้น



ภาพที่ 2.4 (ก) การใส่ค่าลงในโปรแกรมเคาน์เตอร์ค่าไบท์ต่ำได้จาก PLC และไบท์สูงได้จาก PCLATH



ภาพที่ 2.4 (ข) การใส่ค่าลงในโปรแกรมเคาน์เตอร์จากการทำคำสั่ง GOTO

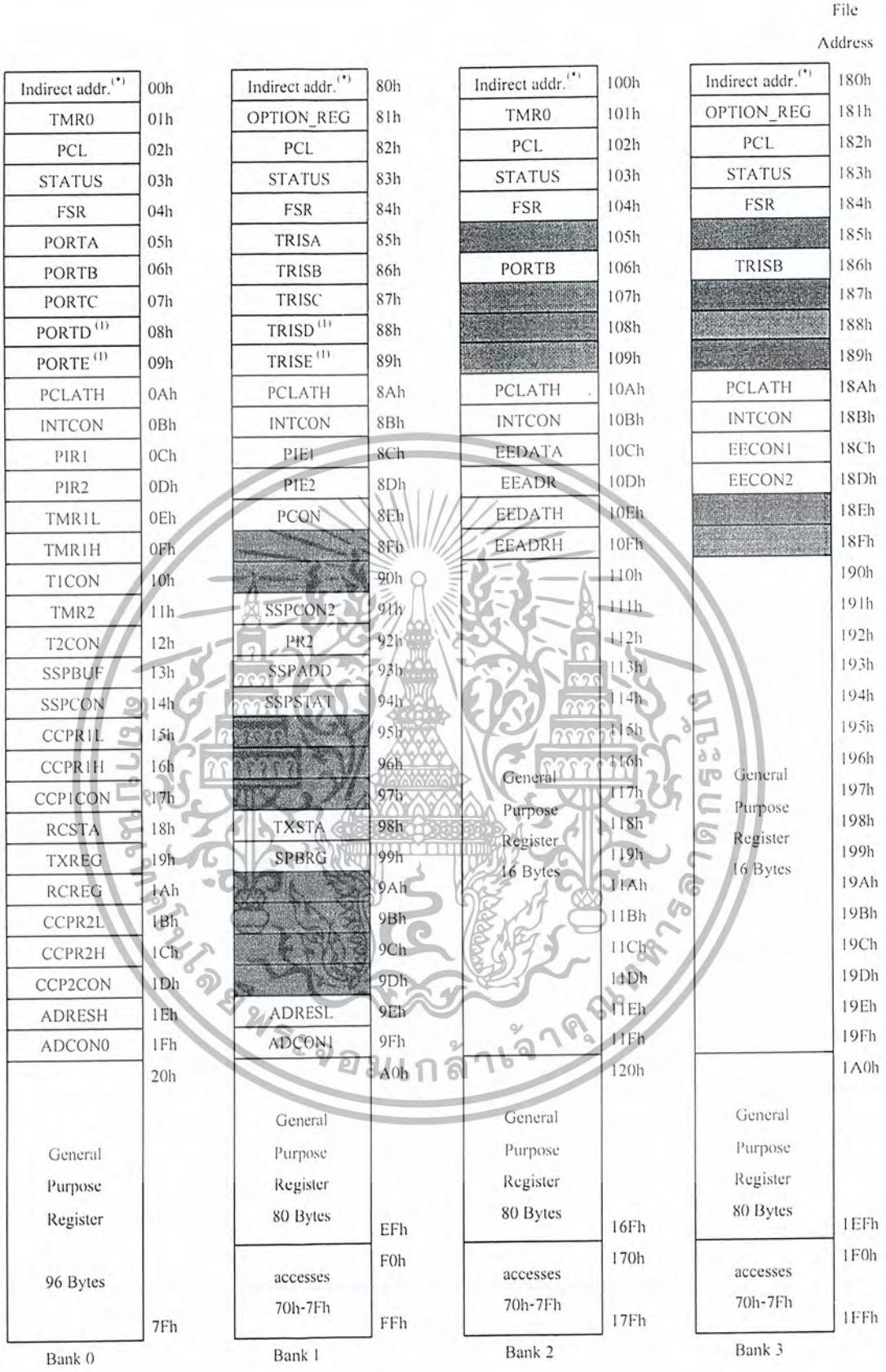
แสตกของไมโครคอนโทรลเลอร์ PIC16F877 จะมี 8 ระดับ แสตกไม่ใช่ส่วนของโปรแกรม หรือข้อมูลและแสตกพอยน์เตอร์ก็ไม่สามารถอ่านหรือเขียนได้ โปรแกรมเคาน์เตอร์จะถูกใส่ (Push) ลงในแสตกเมื่อเกิดการทำคำสั่ง CALL หรือเกิดการเรียกโปรแกรมบริหารอินเตอร์รัพท์และจะคืนสู่โปรแกรมเคาน์เตอร์เมื่อทำคำสั่ง RETURY,RETLW หรือ RETFIE

หากแสตกถูกใส่เกิน 8 ครั้ง ครั้งที่ 9 ค่าในโปรแกรมเคาน์เตอร์จะหันค่าที่ใส่ในแสตกครั้งแรกและวนทับไปเรื่อยๆ ใน PIC16F877 จะไม่มีแฟลกที่บ่งบอกการเกิดแสตกโอเวอร์โฟลว์ และแสตกอันเดอร์โพลว์

2.3.3 หน่วยความจำข้อมูล (รีจิสเตอร์)

หน่วยความจำข้อมูลถูกแบ่งออกเป็น 4 แบงก์ แต่ละแบงก์มี 128 ไบท์ประกอบด้วยรีจิสเตอร์ทั่วไป และรีจิสเตอร์เฉพาะ การเข้าถึงแต่ละแบงก์ทำโดยการเปลี่ยนบิท RP1 และ RP0 ในรีจิสเตอร์สถานะ รีจิสเตอร์เฉพาะบางตัวที่สำคัญจะมีซ้ำอยู่ในหลายแบงก์เพื่อความรวดเร็วในการเข้าถึง

การเข้าถึงรีจิสเตอร์ สามารถอ้างอิงแบบโดยตรง หรือ โดยอ้อมผ่านทางไฟล์ซีเล็กรีจิสเตอร์ รีจิสเตอร์เฉพาะสามารถแบ่งเป็นรีจิสเตอร์ที่ถูกใช้โดยหน่วยประมวลผลกลาง และรีจิสเตอร์สำหรับโมดูลที่ทำหน้าที่พิเศษเช่น ส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอลสำหรับรีจิสเตอร์เฉพาะที่ใช้โดยหน่วยประมวลผลกลางที่สำคัญคือ รีจิสเตอร์สถานะ



ภาพที่ 2.5 แผนผังหน่วยความจำข้อมูล

2.3.4 รีจิสเตอร์สถานะ (Status Register)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	/TO	/PD	Z	DC	C
บิท7							บิท0

IRP บิทเลือกรีจิสเตอร์แบงก์ (ใช้สำหรับการอ้างอิงแอดเดรสโดยอ้อม)

1 = แบงก์ 2,3 (100h-1FFh) 0= แบงก์ 0,1 (00h-FFh)

RP1:RP0 บิทเลือกรีจิสเตอร์แบงก์ (ใช้สำหรับการอ้างอิงแอดเดรสโดยตรง)

11 = แบงก์ 3 (180h-1FFh) 10=แบงก์ 2 (100h – 17Fh)

01 = แบงก์ 1 (80h-1FFh) 00 = แบงก์ 0 (00h – 17Fh)

TO บิทว๊อชด็อกไทม์เอาท์

1 = หลังจากจ่ายกระแสไฟ หรือใช้คำสั่ง CLRWDT หรือ SLEEP

0 = เกิดว๊อชด็อกไทม์เอาท์

PD บิทลดพลังงาน

1 = หลังจากจ่ายกระแสไฟ หรือใช้คำสั่ง CLRWDT

0 = หลังใช้คำสั่ง SLEEP

Z บิทศูนย์

1 = ผลลัพธ์ของการกระทำทางคณิตศาสตร์หรือตรรกะ เป็น 0

0 = ผลลัพธ์ของการกระทำทางคณิตศาสตร์หรือตรรกะ ไม่เป็น 0

DC บิทตัวทด/ยืม * (เป็นผลจากการทำคำสั่ง

ADDWF,ADDLW,SUBLW,SUBWF)

1 = เกิดการทดจาก 4 บิทล่าง 0= ไม่เกิดจากการทด 4 บิทล่าง

C บิทตัวทด/ยืม * (เป็นผลจากการทำคำสั่ง

ADDWF,ADDLW,SUBLW,SUBWF)

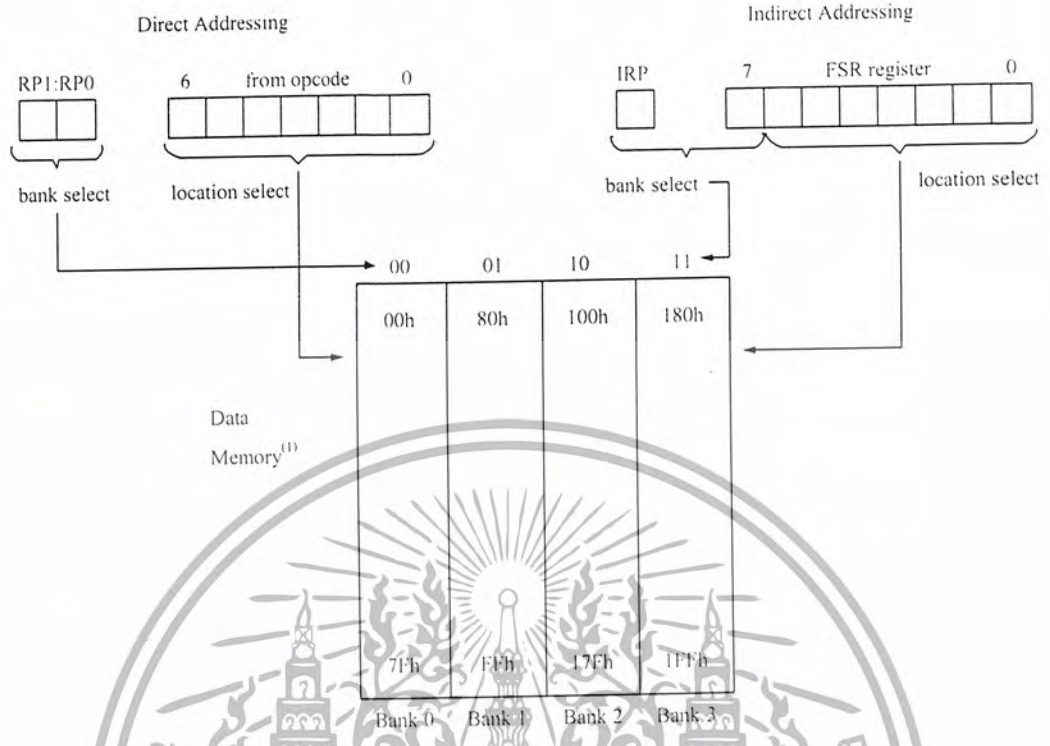
1 = เกิดการทดจาก 4 บิทบน 0= ไม่เกิดจากการทด 4 บิทบน

* สำหรับการยืมผลที่บิทนี้จะกลับกัน

2.3.5 การอ้างอิงแอดเดรสของรีจิสเตอร์โดยอ้อม

การอ้างอิงแอดเดรสของรีจิสเตอร์โดยอ้อมทำได้โดยใช้รีจิสเตอร์ INDF ซึ่งคำสั่งใด ๆ ที่ใช้รีจิสเตอร์ตัวนี้จะเป็นการเข้าถึงรีจิสเตอร์ตัวที่ถูกชี้ โดยใช้ไฟล์ซีเล็กรีจิสเตอร์ (FSR) ซึ่งในการอ่านค่าจากรีจิสเตอร์ INDF เอง (FSR ชี้ไปที่ตำแหน่ง 00) จะได้ค่า 00h และการเขียนค่าไปที่ตัวมันเองจะไม่มีอะไรเกิดขึ้น

เนื่องจากรีจิสเตอร์ FSR มีเพียง 8 บิทการอ้างถึงรีจิสเตอร์ได้ทั้งหมดจะต้องใช้อีกหนึ่งบิท จากบิท IRP (STATUS < 7 >) ดูภาพที่ 2.6 ประกอบ



ภาพที่ 2.6 การอ้างอิงแอดเดรสของรีจิสเตอร์โดยตรงและโดยอ้อม

2.4 ความสามารถทั่วไปของไมโครคอนโทรลเลอร์

2.4.1 การอินเตอร์รัพท์

PIC16F877 มีการอินเตอร์รัพท์รวม 14 แบบ โดยมีบิท GIE (Global Interrupt enable) ในรีจิสเตอร์ INTCON เป็นตัวควบคุมการอินเตอร์รัพท์ทั้งหมด นอกจากนี้ในรีจิสเตอร์ INTCOM ยังมีบิทควบคุมการอินเตอร์รัพท์ของพอร์ท RB0/INT พอร์ท B และ TMR0 อีกด้วยส่วนอินเตอร์รัพท์อื่น ๆ มีรีจิสเตอร์ PIE1 , PIE2 , PIR1 และ PIR2 เป็นตัวควบคุม

เมื่อเกิดการอินเตอร์รัพท์ขึ้น บิท GIE จะถูกเคลียร์ เพื่อป้องกันการเกิดอินเตอร์ซ้อนค่า ในโปรแกรมเคาน์เตอร์ขณะนั้นจะถูกเก็บในแสตก และ โปรแกรมเคาน์เตอร์จะชี้ไปที่ตำแหน่ง 0004h ซึ่งเป็นตำแหน่งของโปรแกรมการการบริหารอินเตอร์รัพท์ (Interrupt Service Rountine) ด้วยการโพลลิ่ง (Polling) อินเตอร์รัพท์แฟลก จะสามารถตรวจสอบได้ว่าเกิดอินเตอร์รัพท์จากที่ใด

คำสั่ง RETFIE ใช้เพื่อออกจากโปรแกรมบริการอินเตอร์รัพท์ โดยบิท GIE จะถูกเซตโดยอัตโนมัติเพื่ออนุญาตการอินเตอร์รัพท์อีกครั้ง

อินเตอร์รัพท์แฟลกจะต้องถูกเคลียร์โดยโปรแกรมก่อนที่จะออกจากโปรแกรมบริหาร หรือ อินเตอร์รัพท์ หากไม่มีการเคลียร์อินเตอร์รัพท์แฟลก เมื่อบิท GIE ถูกเซต จะเกิดการเรียกโปรแกรมบริการอินเตอร์รัพท์ซ้ำ

2.4.2 รีจิสเตอร์ควบคุมการอินเตอร์รัพท์ (Interrupt Control Register :INTCON)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T01E	INTE	RBIE	T0IF	INIF	RBIF
บิท 7							บิท0

GIE บิทควบคุมการอินเตอร์รัพท์ทั้งหมด

1 = อนุญาตการอินเตอร์รัพท์ทั้งหมด

0 = ไม่อนุญาตการอินเตอร์รัพท์ทั้งหมด

PEIE บิทการควบคุมการอินเตอร์รัพท์ย่อย

1 = อนุญาตการอินเตอร์รัพท์

0 = ไม่อนุญาตการอินเตอร์รัพท์

T0IE บิทควบคุมการอินเตอร์รัพท์เมื่อ TMRO เกิดโอเวอร์โฟลว์

1 = อนุญาตการอินเตอร์รัพท์ของ TMRO

0 = ไม่อนุญาตการอินเตอร์รัพท์ของ TMRO

INTE บิทการควบคุมการอินเตอร์รัพท์ของขา RB0/INT

1 = อนุญาตการอินเตอร์รัพท์ของ RB0/INT

0 = ไม่อนุญาตการอินเตอร์รัพท์ของ RB0/INT

RBIE บิทควบคุมการอินเตอร์รัพท์เมื่อพอร์ท B เปลี่ยนสถานะ

1 = อนุญาตการอินเตอร์รัพท์ของพอร์ท B

0 = ไม่อนุญาตการอินเตอร์รัพท์ของพอร์ท B

T0IE บิทแสดง TMRO เกิดโอเวอร์โฟลว์

1 = TMRO เกิดโอเวอร์โฟลว์ (ต้องรีเซตโดยโปรแกรม)

0 = TMRO ไม่เกิดโอเวอร์โฟลว์

INTE บิทแสดงการอินเตอร์รัพท์ของขา RB0/INT

1 = เกิดการอินเตอร์รัพท์ของขา RB0/INT(ต้องรีเซตโดยโปรแกรม)

0 = ไม่เกิดการอินเตอร์รัพท์ของขา RB0/INT

RBIF บิทแสดงการอินเตอร์รัพท์เมื่อพอร์ท B เปลี่ยนสถานะ

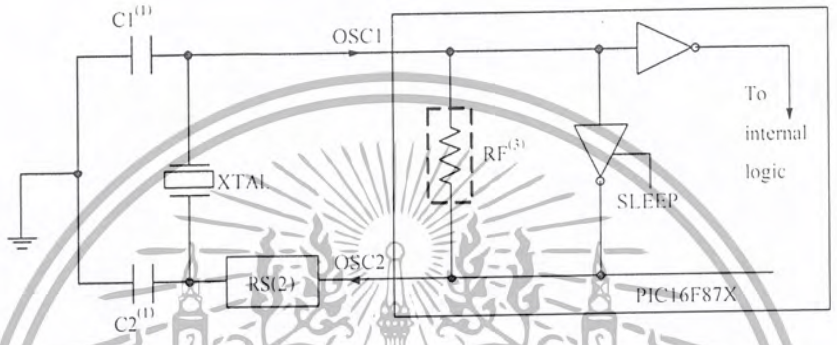
1 = เกิดการอินเตอร์รัพท์ของพอร์ท B (ต้องรีเซตโดยโปรแกรม)

0 = ไม่เกิดการอินเตอร์รัพท์ของพอร์ท B

2.4.3 การใช้งานออสซิลเลเตอร์

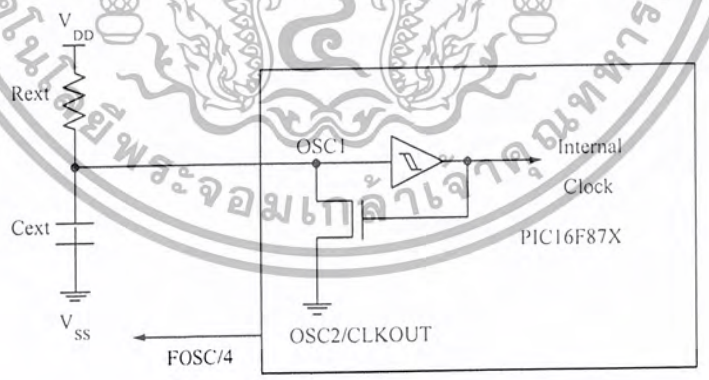
PIC16F877 สามารถเลือกทำงานได้กับออสซิลเลเตอร์ 4 แบบโดยการโปรแกรมบิท FOSCI และ FOSC0 ในรีจิสเตอร์ Configuration ออสซิเลเตอร์มี 4 แบบดังนี้

- LP คริสตอลพลังงานต่ำ
- XT คริสตอล/เรโซเนเตอร์
- HS คริสตอล/เรโซเนเตอร์ ความเร็วสูง
- RC รีจิสเตอร์ – คาปาซิสเตอร์



ภาพที่ 2.7 การต่อคริสตอลออสซิลเลเตอร์

สำหรับคริสตอลออสซิลเลเตอร์ ดังภาพที่ 2.7 ควรเลือกใช้คริสตอลแบบตัดขนาน (parallel cut crystal) เพื่อให้ได้ความถี่ใกล้เคียงกับที่ระบุไว้มากที่สุด



ภาพที่ 2.8 การต่อรีซิสเตอร์ คาปาซิเตอร์ ออสซิลเลเตอร์

การใช้รีซิสเตอร์ คาปาซิเตอร์ ออซิลเลเตอร์ เหมาะสำหรับงานที่ไม่ต้องการความถี่เที่ยงตรงมากนัก โดยค่าความถี่นั้นจะขึ้นอยู่กับค่าแรงดัน V_{DD}+ อุณหภูมิ ค่าความต้านทาน และ ความจุของรีจิสเตอร์และคาปาซิสเตอร์ ภาพที่2.8 จะแสดงการต่อรีซิสเตอร์ คาปาซิเตอร์ ออสซิเลเตอร์

2.4.4 โหมดประหยัดพลังงาน (Sleep Mode)

เมื่อไมโครคอนโทรลเลอร์ทำคำสั่ง SLEEP จะเป็นการเข้าสู่โหมดประหยัดพลังงาน บิทลดพลังงานในรีจิสเตอร์สถานะจะถูกเคลียร์ และออสซิเลเตอร์จะถูกปิดแต่พอร์ทอินพุทเอาท์พุทจะยังคงสถานะเดิมอยู่ (1,0 หรือ Hi – Impedance)

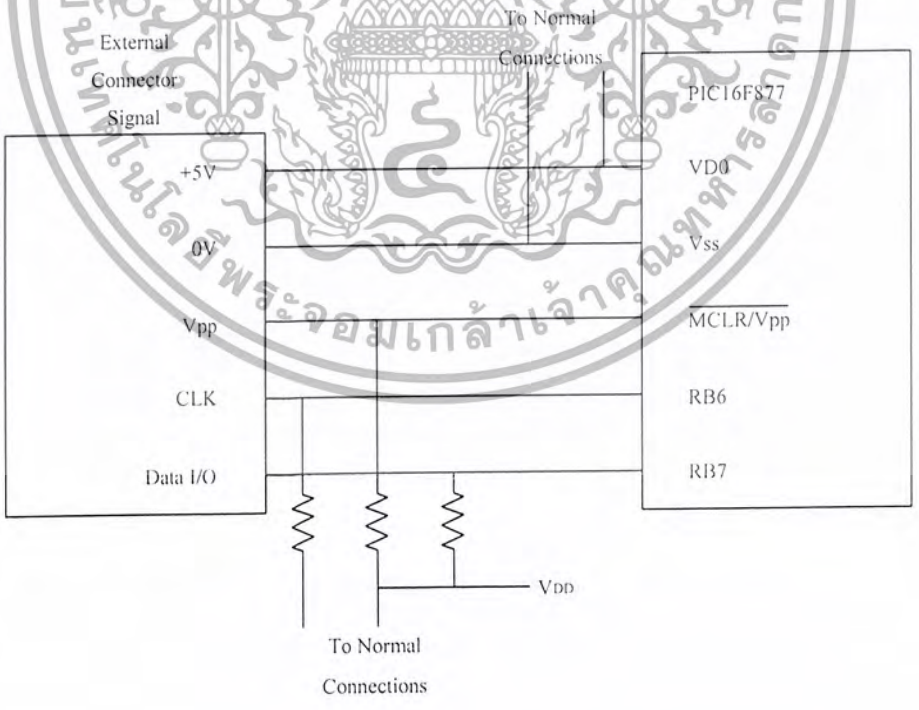
การออกจากโหมดประหยัดพลังงาน จะเกิดขึ้นเมื่อมีเหตุการณ์หนึ่งดังต่อไปนี้

- มีสัญญาณรีเซตที่ขา / MCRL
- การกระตุ้นจากวัอชด็อกไทเมอร์
- เกิดการอินเตอร์รัพท์

การเกิดอินเตอร์รัพท์จะทำให้ออกจากโหมดประหยัดพลังงาน โดยไม่สนว่าได้อนุญาตการอินเตอร์รัพท์หรือไม่ และหากไม่อนุญาตการอินเตอร์รัพท์ คำสั่งต่อจาก SLEEP จะถูกประมวลผลต่อไปตามปกติ หากอนุญาตการอินเตอร์รัพท์ คำสั่งต่อจาก SLEEP จะถูกประมวลผลแล้วกระโดดไปยังโปรแกรมบริหารอินเตอร์รัพท์

2.4.5 การโปรแกรมไมโครคอนโทรลเลอร์ในระบบ (In-Circuit Serial Programming)

PIC16F877 สามารถโปรแกรมได้แม้อยู่ในวงจรโดยมีสัญญาณนาฬิกาและสัญญาณข้อมูลไฟเลี้ยง กราวนด์ และแรงดันไฟสูงสำหรับการโปรแกรม ดังภาพที่ 2.9



ภาพที่ 2.9 การต่อไมโครคอนโทรลเลอร์เพื่อทำการโปรแกรม

2.5 ส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอล

ส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอลในไมโครคอนโทรลเลอร์มีอนาลอกอินพุท 8 ช่อง และ สามารถเปลี่ยนการใช้งานเป็นดิจิตอลพอร์ทได้ด้วยขั้นตอนการทำงานเริ่มที่แรงดันที่อนาลอกอินพุทจะถูกประจุให้ตัวเก็บประจุแซมเปิ้ลแอนด์โฮล (Sample and Hold Capacitor) จากนั้นจะถูกแปลงด้วย วิธี Successive Approximation ที่ความละเอียด 10 บิท แรงดันอ้างอิงทั้งบวกและลบสามารถเลือกใช้จากวงจรภายนอกหรือจาก Vdd และ Vss

การแปลงสัญญาณทำได้แม้อยู่ในสภาวะ Sleep mode โดยต้องเลือกสัญญาณนาฬิกาที่ใช้ในการแปลงจากวงจรรีซิสเตอร์คาปาซิเตอร์ออสซิลเลเตอร์ภายใน

2.5.1 รีจิสเตอร์ที่เกี่ยวข้องกับการแปลงสัญญาณ

มีรีจิสเตอร์ 4 ตัวที่เกี่ยวข้องกับการแปลงสัญญาณ ได้แก่

- ADCON0 ควบคุมการทำงาน
- ADCON1 ควบคุมหน้าที่ของพอร์ท
- ADRESH เก็บผลลัพธ์ไบท์สูง
- ADRESL เก็บผลลัพธ์ไบท์ต่ำ

รีจิสเตอร์ ADCON0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
บิท 7							บิท0

ADCS1 : ADCS0 บิทเลือกความถี่สัญญาณนาฬิกา

00 = Fosc / 2 01 = Fosc/8

10 = Fosc / 32 11 = Frc

CHS2 : CHS0 บิทเลือกช่องสัญญาณอนาลอก

000 = ช่อง 0 (RA0 / AN0) 001 = ช่อง 1 (RA1 / AN1)

010 = ช่อง 2 (RA2 / AN2) 011 = ช่อง 3 (RA3 / AN3)

100 = ช่อง 4 (RA5 / AN4) 101 = ช่อง 5 (RE0 / AN5)

101 = ช่อง 6 (RE1 / AN6) 111 = ช่อง 7 (RA2 / AN7)

GO/DONE บิทสถานะของส่วนแปลงสัญญาณ

1 = ส่วนแปลงสัญญาณกำลังทำงาน (การเซตบิทนี้เป็นคำสั่งเริ่มทำงาน)

0 = ไม่มีการทำงาน (ถูกเคลียร์โดยฮาร์ดแวร์)

ADON เปิด/ปิด ส่วนแปลงสัญญาณ

1 = เปิดส่วนแปลงสัญญาณ

0 = ปิดส่วนแปลงสัญญาณ ทำให้ส่วนนี้ไม่มีการใช้พลังงาน

รีจิสเตอร์ ADCON1

R/W -0	U-0	U-0	U-0	R/W -0	R/W -0	R/W -0	R/W -0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

บิท7 บิท 0

รูปแบบของผลลัพธ์ในรีจิสเตอร์ ADRESH และ ADRESL

1 = จัดชิดขวา โดย 2 บิทบนเก็บใน ADRESH และ 8 บิทล่างเก็บใน ADRESL

0 = จัดชิดขวา โดย 8 บิทบนเก็บใน ADRESH และ 2 บิทล่างเก็บใน ADRESL

ตารางที่2.2 PCFG3 : PCFG0 กำหนดหน้าที่ของพอร์ท

PCFG3 : PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	Vref+	Vref-	Chan /Ref
0000	A	A	A	A	A	A	A	A	Vdd	Vss	8/0
0001	A	A	A	A	Vref+	A	A	A	RA3	Vss	7/1
0010	D	D	D	A	A	A	A	A	Vdd	Vss	5/0
0011	D	D	D	A	Vref+	A	A	A	RA3	Vss	4/1
0100	D	D	D	D	A	D	A	A	Vdd	Vss	3/0
0101	D	D	D	D	Vref+	D	A	A	RA3	Vss	2/1
0111	D	D	D	D	D	D	D	D	Vdd	Vss	0/0
1000	A	A	A	A	Vref+	Vref-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	Vdd	Vss	6/0
1010	D	D	A	A	Vref+	A	A	A	RA3	Vss	5/1
1011	D	D	A	A	Vref+	Vref-	A	A	RA3	RA2	4/2
1100	D	D	D	A	Vref+	Vref-	A	A	RA3	RA2	3/2
1101	D	D	D	D	Vref+	Vref-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	Vdd	Vss	1/0
1111	D	D	D	D	Vref+	Vref-	D	A	RA3	RA2	1/2

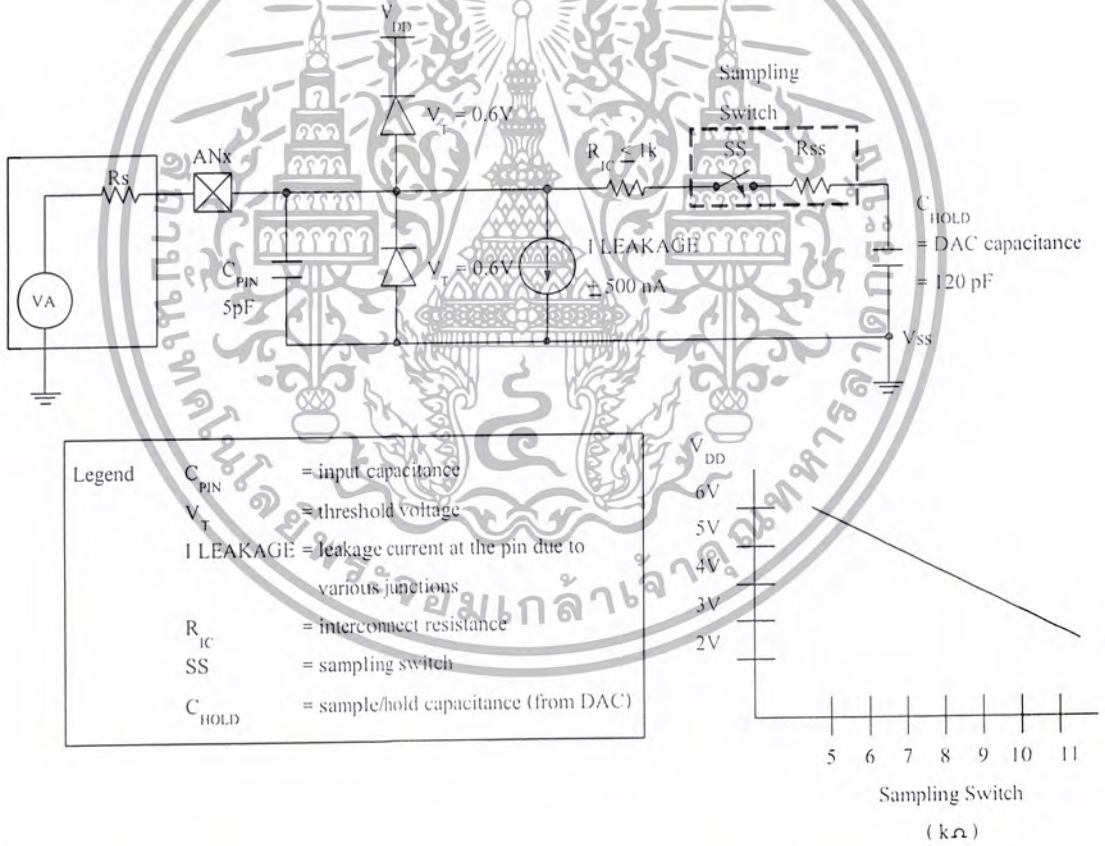
A = ใช้พอร์ทเป็นอนาลอกอินพุท D = ใช้พอร์ทเป็นดิจิตอล

2.5.2 รีจิสเตอร์เก็บผลลัพธ์

รีจิสเตอร์ ADRESH และ ADRESL ใช้เก็บผลลัพธ์ไบท์สูง และ ไบท์ต่ำที่ได้จากการแปลงสัญญาณอนาลอกเป็นดิจิตอลขนาด 10 บิท แต่ขนาดรวมของรีจิสเตอร์สองตัวนี้เป็น 16 บิท ทำให้ผู้ใช้สามารถเลือกรูปแบบการเก็บข้อมูลในรีจิสเตอร์ให้จัดชิดซ้าย หรือ ชิดขวาได้โดยการกำหนดบิท ADFM ในรีจิสเตอร์ ADCON1

2.6 ข้อกำหนดในการใช้งาน

เพื่อให้การแปลงสัญญาณได้ความแม่นยำมากที่สุด จะต้องมีการออกแบบให้มีเวลาในการประจุคาปาซิเตอร์เพียงพอ เวลาที่ใช้ขึ้นอยู่กับค่าของทรานซิสเตอร์ภายในโมดูล และค่าอิมพีแดนซ์ของแหล่งกำเนิดสัญญาณ โดยค่าอิมพีแดนซ์ของแหล่งกำเนิดสัญญาณไม่ควรมากเกิน 10 กิโลโอห์ม ดูได้จากภาพที่ 2.10 ได้



ภาพที่ 2.10 วงจรเสมือนแสดงการประจุคาปาซิเตอร์ C_{HOLD}

บทที่ 3
การสื่อสารแบบอนุกรม

การสื่อสารมีด้วยกัน 2 รูปแบบคือ การสื่อสารแบบขนานและการสื่อสารแบบอนุกรม การสื่อสารแบบขนานเป็นการรับ หรือส่งข้อมูลคราวละมากกว่า 1 บิต ในเวลาเดียวกันทำให้การรับและส่งข้อมูลมีความเร็วสูง แต่จำนวนของสายสัญญาณที่ใช้ในการส่งผ่านข้อมูล ต้องมีมากตามจำนวนบิตของข้อมูลที่ทำการส่งนอกจากนั้นยังมีสายที่ใช้สำหรับควบคุมและตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตข้อมูลก็ได้ ในขณะที่การรับส่งข้อมูลแบบอนุกรมจะเป็นการรับส่งข้อมูลครั้งละ 1 บิต การรับส่งข้อมูลแบบอนุกรมจึงมีข้อดีในเรื่องของ จำนวนสายสัญญาณที่น้อยมาก จะไม่แปรตามจำนวนบิตของข้อมูลแต่ความเร็วในการสื่อสารจะลดลง และโปรแกรมควบคุมจะมีความซับซ้อนมากกว่า

การสื่อสารแบบอนุกรมยังสามารถแบ่งได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัส และ การสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมรับการรับ และส่งสัญญาณด้วยซึ่งตัวอย่างการส่งข้อมูลแบบซิงโครนัส คือ คีย์บอร์ดของคอมพิวเตอร์ หรือ บัสแบบ I^2C ซึ่งสายเส้นหนึ่งเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นเป็นสายของข้อมูล ดังนั้นในการติดต่อกันแบบซิงโครนัสนี้ จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้น คือ สัญญาณนาฬิกา ข้อมูลและกราวนด์

การสื่อสารอนุกรมแบบอะซิงโครนัสรับและส่งข้อมูล โดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วย แต่จะใช้การกำหนดค่าอัตราความเร็วในการรับและส่งข้อมูลให้มีค่าเท่านั้น ซึ่งเรียกอัตราความเร็วนี้ว่า อัตราบอดหรือบอดเรท มีหน่วยเป็นบิตต่อวินาที

3.1 การสื่อสารอนุกรมแบบ RS-232

สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ ได้วางมาตรฐานที่มีชื่อว่า EIA RS-232 เป็นมาตรฐานสื่อสารอนุกรมแบบอะซิงโครนัส 2 ทิศทาง มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต ระดับสัญญาณตั้งแต่ -3 โวลท์ จนถึง -12 โวลท์ แสดงว่ามีข้อมูลหรือเทียบเท่าลอจิกหนึ่งและ +3โวลท์ ถึง +12 โวลท์แสดงว่าเป็นช่องว่างหรือ ลอจิกศูนย์

มาตรฐาน RS-232 กำหนดรูปแบบการสื่อสารข้อมูลกัน ระหว่างอุปกรณ์เชื่อมต่อข้อมูลกับวงจรข้อมูลปลายทางอุปกรณ์ DTE นั้น จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวอย่างเช่น ไมโครคอนโทรลเลอร์ หรือ ถ้ากำหนดค่าเป็นพาริตี้คู่ ค่าของบิทพาริตี้จะต้องมีลอจิกเป็นศูนย์แต่ถ้ากำหนดเป็นพาริตี้ที่ค่าของ บิทพาริตี้จะต้องเป็นหนึ่งเพื่อให้จำนวนบิทที่เป็นหนึ่งรวมบิทพาริตี้มีค่า

เป็นคี่ บิทพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลซึ่งทางภาครับ ต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ที่ตรงกันไว้ด้วยหากเกิดความผิดพลาดในการส่งสัญญาณทางภาครับจะแสดงข้อผิดพลาดให้ผู้ใช้ทราบ กระบวนการดังกล่าวเป็นวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้น ในการรับส่งข้อมูลที่ง่ายที่สุด แต่สามารถตรวจสอบได้ เมื่อบิตของข้อมูลที่ทำการรับส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตที่ผิดพลาดมากกว่า 1 บิตการตรวจสอบด้วยวิธีนี้จะมีไม่ได้ผล สำหรับการตั้งพาริตี้เป็น None นั้นทั้งภาครับและภาคส่งจะไม่มีการตรวจสอบพาริตี้

3.1.1 อัตราความเร็วในการรับส่งข้อมูลแบบอะซิงโครนัส

อัตราความเร็วในการรับส่งข้อมูลของ การรับส่งข้อมูลแบบอะซิงโครนัส หรือ อัตราบอดหรือบอดเรทที่ใช้สำหรับพอร์ทอนุกรม RS-232 มีหลายค่า ได้แก่ 110 , 150 , 300 , 600 , 1200 , 2400 , 4800 , 9600 , และ19200 บิตต่อวินาทีโดยมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ เนื่องจาก บอดเรท คือค่าของจำนวนบิตที่สามารถส่งได้ใน 1 วินาที สมมติ ว่าข้อมูลอนุกรมมีขนาด 8 บิตไม่มีการตรวจสอบพาริตี้มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิตความยาวของข้อมูล 1 ไบท์ จะเท่ากับ 10 บิตถ้าใช้บอดเรทในการส่งข้อมูลเท่ากับ 9600 บิต ต่อวินาทีก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบท์ต่อวินาที

3.1.2 คอนเน็กเตอร์สำหรับพอร์ท RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้ คอนเน็กเตอร์แบบ DB-25 หรือ DB-9 โดยที่คอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับ คอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่นๆ ที่เคยมีการใช้งานในอดีตไม่ค่อยสำคัญมากนักจึงถูกยกเลิกไปโดยประกอบไปด้วยขาต่างๆ ดังนี้

Receive Data : RD หรือ RxD ขานี้ใช้เพื่อรับข้อมูลอนุกรมเข้ามายังคอมพิวเตอร์ โดยจะนำข้อมูลที่อ่านได้ไปเก็บไว้ในรีจิสเตอร์บัฟเฟอร์

Transmitted Data : TD หรือ TxD ขานี้ใช้เพื่อส่งข้อมูลอนุกรมออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับข้อมูลออกไป

Signal Ground : GND เป็นขากราวนด์ของสัญญาณ

Data Terminal Ready : DTR เป็นขาที่ใช้สำหรับส่งสัญญาณออกไปจากคอมพิวเตอร์ เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าต้องการที่จะติดต่อกับอุปกรณ์ปลายทางโดยขา DTR นี้ จะต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทางและขา DTR ของอุปกรณ์ปลายทางจะต้องเชื่อมกับขา DSR ของคอมพิวเตอร์และถ้าใช้การเชื่อมต่อแบบ 3 สายต้องเชื่อมต่อขา DTR และ DSR ของพอร์ทอนุกรมเข้าด้วยกันและต้องเชื่อมเข้ากับขา DCD ให้ถึงกันด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

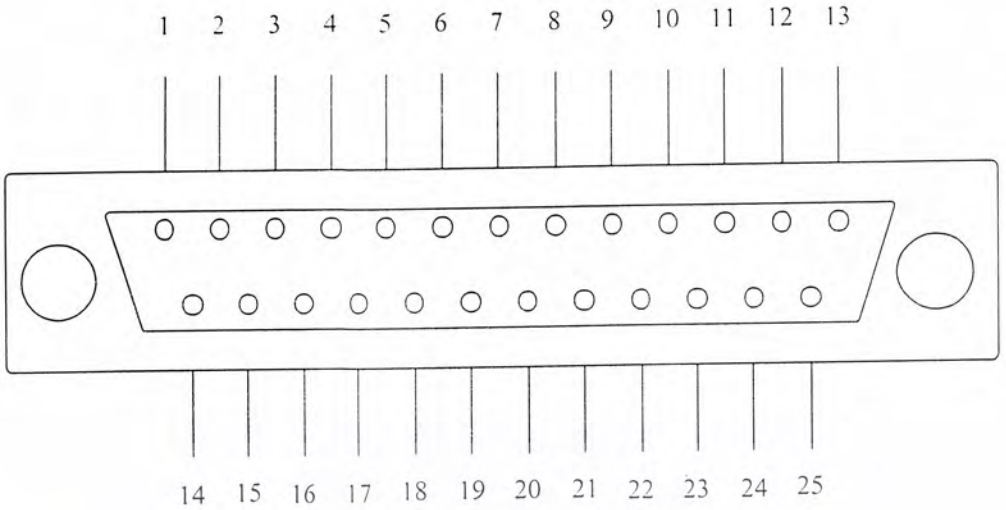
Data Set Ready : DSR ขานี้จะใช้ควบคู่กับขา DTR เพื่อใช้สำหรับตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ซึ่งในกรณีที่มีการเชื่อมต่อแบบ 3 สายจะต้องเชื่อมต่อขาRTS และCTS เข้าด้วยกันเพื่อให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

Clear To Send : CTS จะเป็นขาอินพุทที่ทำหน้าที่รอรับสัญญาณที่ส่งถูกเข้ามา เมื่อมีการส่งสัญญาณเข้ามาที่ขานี้ ข้อมูลจะถูกส่งออกไปที่ขา TxD ขานี้จะใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลแล้วหรือยัง

Data Carrier Detect : DCD ขานี้จะแอกทีฟเมื่อได้รับสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูล เช่น โมเด็มสำหรับการใช้งานปกติ ขานี้ไม่ได้ถูกใช้งานมากนัก

Ring indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ซึ่งปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มแล้วต้องการตรวจสอบสัญญาณเรียกจากสายโทรศัพท์ ดูลักษณะของคอนเน็กเตอร์ได้ในภาพที่ 3.1

ภาพที่ 3.1 คอนเน็กเตอร์อนุกรม 9 ขาหรือแบบ DB9

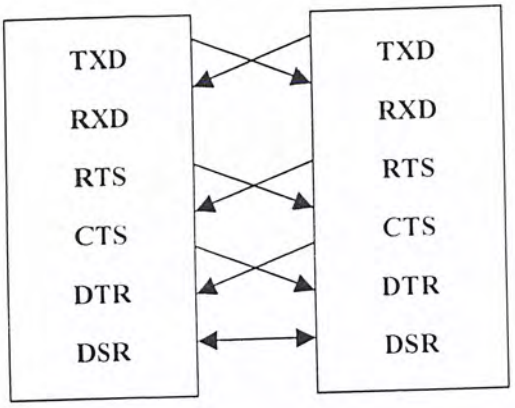


ภาพที่ 3.2 คอนเน็กเตอร์ขนาน 25 ขา หรือแบบ DB 25

ตารางที่ 3.1 รายละเอียดของขาคอนเน็กเตอร์

คอนเน็กเตอร์ DB-9	คอนเน็กเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	Data Carrier Detect : DCD	อินพุท
2	3	Received Data : RxD	อินพุท
3	2	Transmited Data : TxD	เอาท์พุท
4	20	Data Terminal Ready : DTR	เอาท์พุท
5	7	Signal Ground : GND	-
6	6	Data Set Ready : DST	อินพุท
7	4	Request To Send : RTS	เอาท์พุท
8	5	Clear To Send : CTS	อินพุท
9	22	Ring Indicator : RI	อินพุท

สำหรับการเชื่อมต่อสายระหว่างคอมพิวเตอร์กับอุปกรณ์ภายนอกโดยแสดงดังในภาพที่ 3.3 ลูกศรในรูปแสดงถึงทิศทางของข้อมูล และการเชื่อมต่อในภาพที่ 3.3 (ก) 1 จะเป็นการเชื่อมต่อแบบ Null Modem ส่วนการเชื่อมต่อในภาพที่ 3.3 (ข) เป็นการเชื่อมต่อโดยใช้สายสัญญาณน้อยที่สุดเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวนด์



ภาพที่ 3.3 (ก) การเชื่อต่อแบบ Null Modem

ภาพที่ 3.3 (ข) การเชื่อมต่อโดยใช้สายสัญญาณน้อยสุดเพียง 3 เส้น

3.1.3 Universal Asynchronous Receiver Transmitter

Universal Asynchronous Receiver Transmitter หรือ UART หมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสหน้าที่หลักของ UART คือแปลงสัญญาณข้อมูลแบบขนานที่ส่งมาจากหน่วยประมวลผลกลาง ให้เป็นสัญญาณอนุกรมแบบอะซิงโครนัสแล้วทำการส่งออกไปและแปลงสัญญาณอนุกรมแบบอะซิงโครนัส ที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่หน่วยประมวลผลกลาง ซึ่งนอกจาก UART จะทำหน้าที่แปลงรูปของข้อมูลแล้วยังแจ้งรายละเอียดอื่น ๆ ของข้อมูลให้คอมพิวเตอร์รับทราบด้วย อาทิ อัตราเร็วในการรับหรือบอดเรท รูปแบบการส่งข้อมูลความผิดพลาดที่เกิดขึ้นระหว่างส่งข้อมูล

ภายใน UART จะมีวงจรสร้างบอดเรทที่โปรแกรมได้ โดยจะกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้จะมีขนาด 16 บิต ทำให้สามารถกำหนดตัวหารอยู่ในช่วง 1 ถึง 65,535 ในเครื่องคอมพิวเตอร์ทั่วไปจะมี UART ที่ใช้งานกันอยู่ 2 เบอร์คือ เบอร์ 8250 และ 16550 สำหรับ UART เบอร์ 8250 เป็น UART มาตรฐานที่มีใช้ในคอมพิวเตอร์รุ่น XT โดย UART

เบอร์นี้มีบัฟเฟอร์สำหรับรับและส่งข้อมูลเป็นตำแหน่งเดียวกัน ทำให้การรับและส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิตต่อวินาที สำหรับ UART เบอร์ 16550 ถูกใช้ในคอมพิวเตอร์ รุ่น AT จะเพิ่มส่วนของชิฟท์รีจิสเตอร์แบบ FIFO ขนาด 16 ไบท์เข้าไป ทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ระดับ 256 กิโลบิตต่อวินาทีได้

ไอซี UART เหล่านี้จะมีระดับแรงดันของลอจิกเป็นแบบทีทีแอล (TTL) แต่เพื่อให้การรับส่งข้อมูลสามารถทำให้ได้ที่ระยะทางมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิกศูนย์จะมีระดับแรงดัน -3 โวลท์ ถึง -12 โวลท์ และลอกจิกหนึ่งมีระดับแรงดัน +3 โวลท์จนถึง +12 โวลท์

3.1.4 ลักษณะสัญญาณอินพุทและเอาท์พุทของพอร์ท RS-232

สัญญาณเอาท์พุทRTSและDTRรวมทั้งสัญญาณแสดงสถานะอินพุท CTS , DSR และ DCD จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้ง RxD และ TxD จะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาท์พุทออกมาเป็นแบบทีทีแอลดังนั้นสัญญาณที่ส่งออกมาจาก UART จะเข้าสู่วงจรขับ เพื่อแปลงระดับสัญญาณให้เป็นไปตามมาตรฐานRS-232 ก่อนที่จะส่งค่าออกจากคอมพิวเตอร์ สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับในลักษณะนี้เช่นเดียวกัน เพื่อให้ได้สัญญาณในระดับเดียวกันได้แต่สำหรับวงจรขับที่ใช้ทั้งภายในคอมพิวเตอร์ และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะกลับสถานะสัญญาณ

3.1.5 แอดเดรสของพอร์ทอนุกรม

แอดเดรสฐานของพอร์ทอนุกรมในคอมพิวเตอร์มี 4 ตำแหน่ง เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสจะทำการตรวจสอบแอดเดรสของพอร์ทอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ทอนุกรม จากนั้นไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบท์ นอกจากนั้นบิทที่ 3-1 ของหน่วยความจำตำแหน่ง 0000:041H ยังใช้เพื่อแสดงจำนวนของพอร์ทอนุกรมที่มีอยู่ในคอมพิวเตอร์อีกด้วย

ตารางที่ 3.2 แอดเดรสฐานและหน่วยความจำที่เก็บแอดเดรสฐานของพอร์อนุกรม

พอร์ท	แอดเดรสฐาน	หน่วยความจำที่เก็บแอดเดรสฐาน
COM1	3F8H	0000:0400H ถึง 0000:0401H
COM2	2F8H	0000:0402H ถึง 0000:0403H
COM3	3E8H	0000:0404H ถึง 0000:0405H
COM4	2E8H	0000:0406H ถึง 0000:0407H

3.1.6 รีจิสเตอร์ของพอร์ทอนุกรม

พอร์ทอนุกรมมีรีจิสเตอร์ขนาด 8 บิท 8 ตัวที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอร์ภายในพอร์ทอนุกรมสามารถคำนวณได้จากแอดเดรสฐานของพอร์ทอนุกรม ยกตัวอย่างเช่น พอร์ทอนุกรม COM1 มีแอดเดรสอยู่ที่ 3F8H แอดเดรสของรีจิสเตอร์ต่างๆ จะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ทอนุกรมดังนี้

รีจิสเตอร์บัฟเฟอร์ 00H

เป็นรีจิสเตอร์สำหรับเก็บข้อมูลที่รับเข้ามา หรือพักข้อมูลก่อนที่จะส่งออกไปมีแอดเดรสอยู่ที่ Base + 00H ถ้าเป็น COM1 จะมีแอดเดรสอยู่ที่ 3F8H การติดต่อกับรีจิสเตอร์นี้เพื่อส่งข้อมูลจะต้องกำหนดบิท DLAB ในรีจิสเตอร์กำหนดรูปแบบข้อมูล มีสถานะเป็นศูนย์เมื่อเขียนข้อมูลไปยังแอดเดรสนี้ข้อมูลจะถูกส่งออกไปแบบอนุกรมในกรณีรับข้อมูลเมื่อ UARTรับข้อมูลเข้ามาและแปลงเป็นแบบขนานแล้วข้อมูลแบบขนานจะถูกส่งไปยัง รีจิสเตอร์บัฟเฟอร์หลังจากมีการอ่านค่าจากรีจิสเตอร์นี้แล้ว รีจิสเตอร์จะถูกเคลียร์เพื่อให้พร้อมสำหรับการรับข้อมูลในไบท์ต่อๆ ไป

รีจิสเตอร์ 01H

เป็นรีจิสเตอร์ที่อนุญาตการอินเตอร์รัพท์ ซึ่งใช้เพื่อเซตโหมดในอินเตอร์รัพท์ของพอร์ทอนุกรมซึ่งเป็นการกำหนดให้ UART สร้างสัญญาณอินเตอร์รัพท์ขึ้นมา มีแอดเดรสอยู่ที่ Base + 01H ถ้าเป็น COM1 จะมีแอดเดรสอยู่ที่ 3F9H รายละเอียดในแต่ละบิทของรีจิสเตอร์ 01H มีดังนี้

-	-	-	-	SINP	ERBK	TBE	RxRD

บิท7 บิท0

SINP 1 = อนุญาตการอินเตอร์รัพท์เมื่อมีการเปลี่ยนสถานะที่ขาอินพุท CTS , DSR , DCD หรือขา RI

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

ERBK 1 = อนุญาตการอินเตอร์รัพท์ เมื่อเกิดความผิดพลาดขึ้นจากบิทพาริตี้ โอเวอร์รันที่เฟรมข้อมูลหรือการหยุดข้อมูล

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

TNE 1 = อนุญาติการอินเตอร์รัพท์เมื่อรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

RxRD 1 = อนุญาตการอินเตอร์รัพท์เมื่อข้อมูล 1 ไบท์ ถูกเก็บลงในรีจิสเตอร์บัฟเฟอร์แล้ว

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

รีจิสเตอร์ 02H

เป็นรีจิสเตอร์อนุญาตการอินเตอร์รัพท์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัพท์ เมื่อมีการอินเตอร์รัพท์เกิดขึ้น มีแอดเดรสอยู่ที่ Base + 02H โดยมีรายละเอียดของแต่ละบิตดังนี้

-	-	-	-	-	ID1	ID0	PND

บิท7 บิท0

ID1:ID0 00 = เกิดการอินเตอร์รัพท์เนื่องจากการเปลี่ยนแปลงของขาอินพุท (มีนัยสำคัญเป็นอันดับ4หรือนัยสำคัญต่ำสุด)

01 = เกิดการอินเตอร์รัพท์เนื่องจากรีจิสเตอร์บัฟเฟอร์ สำหรับส่งข้อมูลที่ส่งไปนั้น (มีนัยสำคัญอันดับ 3)

10 = เกิดการอินเตอร์รัพท์เนื่องจากข้อมูลจะถูกเก็บลงภายในตัวรีจิสเตอร์บัฟเฟอร์ สำหรับรับข้อมูลแล้ว (มีนัยสำคัญอันดับ2)

11 = เกิดการอินเตอร์รัพท์เนื่องจาก ความผิดพลาดในการที่ส่งข้อมูลออก หรือเกิดจากการหยุดกะทันหัน (มีนัยสำคัญเป็นอันดับ 1 หรือนัยสำคัญสูงสุด)

เมื่อมีการอินเตอร์รัพท์ขึ้นนั้น จะต้องมีการเคลียร์ค่าก่อนที่จะให้เกิดอินเตอร์รัพท์ครั้งต่อไป โดยสามารถทำได้ดังนี้

กรณีที่เกิดอินเตอร์รัพท์เนื่องจากการเปลี่ยนแปลงของขาอินพุท(ID1:ID0 = 0:0)จะต้องอ่านค่าจากรีจิสเตอร์ 06H เพื่อเคลียร์ค่าจากการอินเตอร์รัพท์

กรณีที่เกิดอินเตอร์รัพท์เนื่องจากบัฟเฟอร์สำหรับส่งข้อมูลว่าง(ID1:ID0 = 0:0) จะต้องเขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล (รีจิสเตอร์ 00H) หรือ อ่านค่าที่รีจิสเตอร์แสดงสถานะอินเตอร์รัพท์ (รีจิสเตอร์ 02H) เพื่อเคลียร์ค่าการอินเตอร์รัพท์

กรณีที่เกิดการอินเตอร์รัพท์ เนื่องจากเก็บข้อมูลลงในรีจิสเตอร์บัฟเฟอร์ สำหรับรับข้อมูลเรียบร้อยสามารถเคลียร์ค่าของอินเตอร์รัพท์ หรือโดยการอ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์ สำหรับเก็บข้อมูล (รีจิสเตอร์ 00H)

กรณีที่เกิดการอินเตอร์รัพท์ เนื่องจากความผิดพลาดในการรับส่งข้อมูล หรือเกิดการหยุดกะทันหันจะต้องเคลียร์ค่าของอินเตอร์รัพท์ โดยการที่อ่านค่าจากรีจิสเตอร์แสดงสถานการณ์รับและส่งข้อมูลแบบอนุกรม(รีจิสเตอร์ 05H)

PND 1 = ไม่มีการอินเตอร์รัพท์ 2 = มีการอินเตอร์รัพท์เกิดขึ้น

รีจิสเตอร์ 03H

เป็นรีจิสเตอร์กำหนดรูปแบบของข้อมูล มีรายละเอียดของแต่ละบิทดังนี้

DLAB	BRK	PAR2	PAR1	PAR0	STOP	DAB1	DAB0

บิท7 บิท0

DLAB 1 = เข้าสู่โหมดการหารค่าบอดเรท

0 = เป็นการเข้าถึงรีจิสเตอร์บัฟเฟอร์(รีจิสเตอร์ 00H) และรีจิสเตอร์ตัวที่ใช้สำหรับการอินเตอร์รัพท์ (รีจิสเตอร์ 00H) เมื่อบิท DLAB มีสถานะเป็น 1 รีจิสเตอร์บัฟเฟอร์และรีจิสเตอร์สำหรับการเกิดอินเตอร์รัพท์ จะถูกใช้สำหรับการโหลดค่าการหารความถี่สำหรับการกำหนดค่าบอดเรทรีจิสเตอร์ 00H ใช้ในการกำหนดค่าการหารในไบท์ต่ำส่วนรีจิสเตอร์ 01H นั้น ใช้ในการกำหนดค่าการหารในไบท์สูงโดยค่าบอดเรทเท่ากับ 115200 หารด้วยค่าตัวหาร 16 บิท (ค่า 115200 มาจากการหารคริสตอลความถี่ 1.8432 เมกกะเฮิร์ทซ์ที่อยู่ในวงจร UART)

BRK 1 = กำหนดให้สามารถหยุดการรับส่งข้อมูลกะทันหันได้

0 = ไม่มีการหยุด

PAR2:PAR1:PAR0 ใช้กำหนดบิทพาริตี้

000 = ไม่ใช้บิทพาริตี้ 001 = กำหนดพาริตี้คี่

011 = กำหนดพาริตี้คู่ 101 = มาร์ค (Mark)

111 = ช่องว่าง (Space)

STOP 1 = มีบิทบิดท้าย 2 บิท

DAB1:DAB0 ใช้ในการกำหนดจำนวนบิทของข้อมูล

00 = จำนวนบิทข้อมูลเท่ากับ 5 บิท 01 = จำนวนบิทข้อมูลเท่ากับ 6 บิท

10 = จำนวนบิทข้อมูลเท่ากับ 7 บิท 11 = จำนวนบิทข้อมูลเท่ากับ 8 บิท

รีจิสเตอร์ 04H

เป็นรีจิสเตอร์ควบคุมโมเด็ม ที่ใช้ตรวจสอบบิทสำหรับการติดต่อกับโมเด็มเช่น RTS หรือ DTR มีแอดเดรสอยู่ที่ Base + 04H มีรายละเอียดหน้าที่ของแต่ละบิทดังนี้

-	-	-	LOOP	OUT2	OUT1	RTS	DTR

บิท7 บิท0

LOOP 1 = อนุญาตการส่งค่ากลับ 0 = ไม่อนุญาตการส่งค่ากลับ

RST 1 = อนุญาตการใช้งานขา RTS 0 = ไม่อนุญาตการใช้งาน

DTR 1 = อนุญาตการใช้งานขา 0=ไม่อนุญาตการใช้งาน

รีจิสเตอร์ 05H

เป็นรีจิสเตอร์แสดงสถานะการรับและการส่งข้อมูลแบบอนุกรมของ UART ซึ่งจะใช้งานร่วมกับรีจิสเตอร์สำหรับแสดงโหมด และสถานะของการอินเตอร์รัพท์ (รีจิสเตอร์ 02H) เพื่อแสดงสาเหตุของการเกิดอินเตอร์รัพท์ มีรายละเอียดหน้าที่ของแต่ละบิทดังนี้

-	TXE	TBE	BREK	FRAME	PARE	OVFE	RxRD

บิท7 บิท0

TXE (Transmitter Empty)

1 = รีจิสเตอร์บัฟเฟอร์ส่งข้อมูลและชิฟท์รีจิสเตอร์ว่าง

0 = มีข้อมูลเก็บอยู่ในรีจิสเตอร์บัฟเฟอร์ส่งข้อมูลและชิฟท์รีจิสเตอร์

TBE (Transmitter Buffer Empty)

1 = รีจิสเตอร์บัฟเฟอร์ส่งข้อมูลว่าง

0 = มีข้อมูลเก็บอยู่ในรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล

BREAK (Break)

1 = UART ตรวจพบการหยุดรับส่งข้อมูลกะทันหัน

0 = ไม่มีการหยุดรับส่งข้อมูลกะทันหัน

FRAME (Frame Error)

1 = UART ตรวจพบความผิดพลาดแบบเฟรม

0 = ไม่มีข้อผิดพลาด

PARE (Parity Error)

1 = UART ตรวจพบความผิดพลาดแบบพาริตี้

0 = ไม่มีข้อผิดพลาด

OVRE (Overrun Error)

1 = UART ตรวจพบความผิดพลาดแบบโอเวอร์รัน

0 = ไม่มีข้อผิดพลาด

RxRD (Received Data Ready)

1 = บัฟเฟอร์รับข้อมูลเต็ม

0 = ไม่มีข้อมูลในบัฟเฟอร์รับข้อมูล

รีจิสเตอร์ 06H

เป็นรีจิสเตอร์แสดงสถานะของโมเด็ม ใช้แสดงสถานะของขา DCD , RI , DSR and CTS มีแอดเดรสอยู่ที่ Base + 06H มีรายละเอียดหน้าที่ของแต่ละบิทดังนี้

DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS
บิท7							บิท0

DCD 1 = สัญญาณที่ขา DCD เป็น 1 0 = สัญญาณที่ขา DCD เป็น 0

RI 1 = สัญญาณที่ขา RI เป็น 1 0 = สัญญาณที่ขา RI เป็น 0

DSR 1 = สัญญาณที่ขา DSR เป็น 1 0 = สัญญาณที่ขา DSR เป็น 0

CTS 1 = สัญญาณที่ขา CTS เป็น 1 0 = สัญญาณที่ขา CTS เป็น 0

DDCD (Delta Data Carrier Detect)

1 = บิท DCD มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท DCD ไม่มีการเปลี่ยนแปลง

DRI (Delta Ring Indicator)

1 = บิท RI มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท RI ไม่มีการเปลี่ยนแปลง

DDSR(Delta Data Set Ready)

1 = บิท DSR มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท DSR ไม่มีการเปลี่ยนแปลง

DCTS (Delta Clear To Send)

1 = บิท CTS มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท CTS ไม่มีการเปลี่ยนแปลง

รีจิสเตอร์ 07H

เป็นรีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว หรือใช้ทำหน้าที่เป็นหน่วยความจำแรมขนาด 1 ไบท์ ซึ่งจะไม่มีผลใดๆ กับการใช้งาน UART

3.2 ระบบบัส I^2C

ระบบบัสแบบ I^2C ย่อมาจาก Inter IC ถูกพัฒนาโดยบริษัทฟิลลิปส์ เป็นการสื่อสารอนุกรมแบบซิงโครนัส 2 ทิศทาง โดยมีลักษณะสำคัญดังนี้

1. ใช้สายสัญญาณเพียง 2 สายในการติดต่อสื่อสารระหว่างกัน ไอซีมาสเตอร์กับไอซีสเลฟ สายหนึ่งเป็นสัญญาณข้อมูล หรือ SDA อีกสายเป็นสัญญาณนาฬิกาหรือ SCL

2. ไอซีที่ต่อพ่วงบนบัสแต่ละตัวมีแอดเดรสเฉพาะที่แตกต่างกัน สามารถอ้างถึง ได้โดยใช้ซอฟต์แวร์

3. บนบัสเดียวกันสามารถมีไอซีมาสเตอร์ได้มากกว่าหนึ่งตัว เนื่องจากมีระบบป้องกันการใช้งานบัสพร้อมกัน

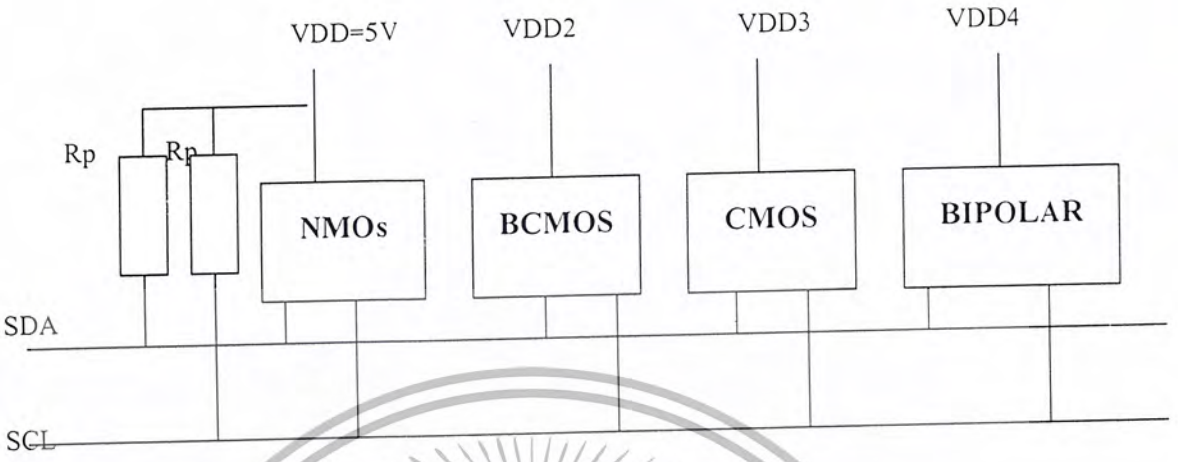
4. การส่งข้อมูลเป็นแบบอนุกรม 8 บิท สองทิศทางที่ความเร็ว 100 กิโลบิทต่อวินาทีใน Standard Mode , 400 กิโลบิตรต่อวินาทีใน Fast Mode และ 3.4 เมกกะบิทต่อวินาทีใน High Speed Mode

5. จำนวนไอซีที่ต่อบนบัส สามารถมีได้มากเท่าที่ค่าความจุรวมของบัสทั้งหมดที่รวมแล้วไม่เกิน 400 พิโคฟารัด

3.2.1 การต่อเชื่อมไอซีบนบัส

ไอซีแต่ละตัวสามารถต่อเชื่อมเข้ากับบัสได้โดยตรงโดยสายสัญญาณ SDAและSCL จำเป็นต้องมีตัวต้านทานพูลอัพอยู่กับแรงดันบวกตลอดเวลา เพื่อทำให้แรงดันในสายเป็นลอจิกหนึ่งเมื่อไม่

มีการส่งข้อมูล และ บัสแบบ I^2C สามารถใช้กับ ไอซีที่มีระดับแรงดันต่างกันได้ด้วย ดูจากภาพที่ 3.2 ประกอบ

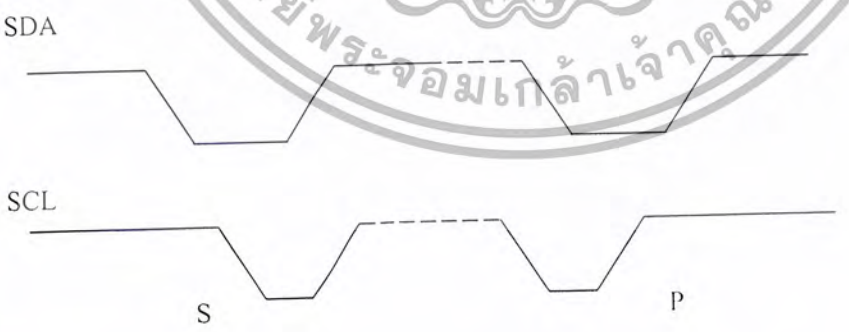


ภาพที่ 3.4 การต่อไอซีบนบัส I^2C

3.2.2 สภาวะต่างๆ ของบัส

สภาวะบัสว่าง สายสัญญาณ SDA และ SCL เป็นลอจิกสูงทั้งคู่คือไม่มีอุปกรณ์ตัวใดใช้บัสอยู่สภาวะเริ่มต้น สายสัญญาณ SDA เปลี่ยนจากลอจิกหนึ่งเป็นศูนย์ในขณะที่สายสัญญาณ SCL เป็นลอจิกหนึ่งเป็นการเริ่มต้นการติดต่อ

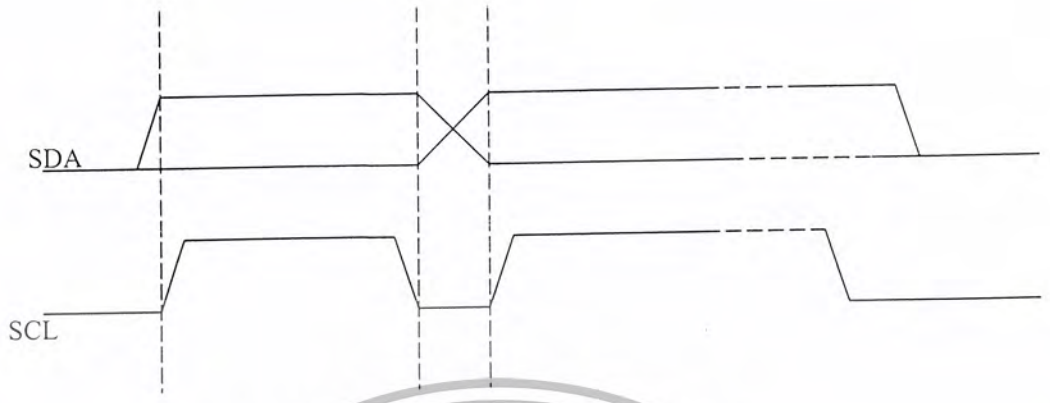
สภาวะหยุดสายสัญญาณ SDA เปลี่ยนจากลอจิกศูนย์เป็นหนึ่ง ในขณะที่สายสัญญาณ SCL เป็นลอจิกหนึ่งเป็นการหยุดการติดต่อดูภาพที่ 3.5 ประกอบ



ภาพที่ 3.5 สภาวะเริ่มต้นและสภาวะหยุดของบัส I^2C

สภาวะส่งข้อมูลนั้นสายสัญญาณ SDAจะต้องไม่มีการเปลี่ยนแปลงสถานะเมื่อสายสัญญาณ SCL มีลอกจิกหนึ่ง ข้อมูลบิทนั้นจะเท่ากันสถานะของสายสัญญาณ SDA และเมื่อสายสัญญาณ

SCL เปลี่ยนกลับมาเป็นลอจิกศูนย์ จึงจะสามารถเปลี่ยนสถานะในสายสัญญาณ SDA ได้ ดูภาพที่ 3.6 ประกอบ

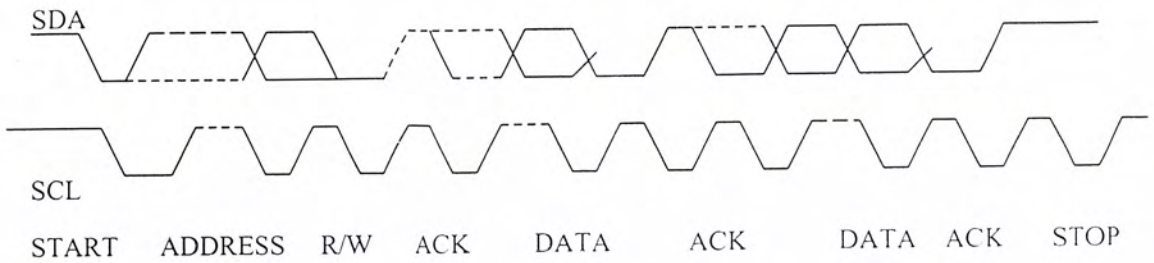


ภาพที่ 3.6 สภาวะส่งข้อมูลของบัส I^2C

สภาวะรับรู้ คือ เมื่อตัวส่งข้อมูลส่งครบ 1 ไบท์ หรือ 8บิทแล้วตัวรับข้อมูลจะต้องตอบสนองการส่งข้อมูลนั้นโดยดึงสายสัญญาณ SDA ลงเป็นลอจิกศูนย์

3.2.3 รูปแบบการติดต่อสื่อสาร

การติดต่อสื่อสารกันระหว่างไอซี ไอซีมาสเตอร์จะเป็นตัวควบคุมสัญญาณ SCL หรือเป็นตัวสร้างสัญญาณนาฬิกาตลอดการติดต่อ การติดต่อเริ่มจากไอซีมาสเตอร์สร้างสภาวะเริ่ม แล้วส่งแอดเดรสและสัญญาณอ่านหรือเขียนข้อมูลลงบนบัส ไอซีสเลฟที่มีแอดเดรสตรงกับแอดเดรสที่ส่งมาจะสร้างสัญญาณรับรู้ตอบกลับไป จากนั้นไอซีมาสเตอร์อาจจะส่งสัญญาณควบคุมหรือข้อมูลให้ไอซีสเลฟนั้น เมื่อครบแต่ละบิทไอซีสเลฟจะต้องสร้างสัญญาณรับรู้กลับไปทุกครั้ง แต่ถ้าเป็นการอ่านข้อมูลจากไอซีสเลฟ ไอซีมาสเตอร์จะต้องสร้างสัญญาณรับรู้ให้ไอซีสเลฟ หลังจากเสร็จสิ้นการติดต่อไอซีมาสเตอร์จะสร้างสภาวะหยุดขึ้น ดังภาพที่ 3.7 เป็นตัวอย่างการติดต่อสื่อสารกันอย่างสมบูรณ์



ภาพที่ 3.7 สัญญาณสื่อสารของบัส I^2C

3.2.4 การอ้างอิงแอดเดรส

การอ้างอิงแอดเดรสแบ่งเป็นแบบ 7 บิท และ 10 บิทโดยการอ้างอิงแอดเดรสแบบ 7 บิท ข้อมูลใน 7 บิทแรกจะเป็นค่าแอดเดรสเริ่มจากบิทที่มีนัยสำคัญสูงสุด และบิทที่เหลือจะเป็นบิทควบคุมว่าจะอ่านหรือเขียนข้อมูล (R/W) ดังภาพที่ 3.8

ภาพที่ 3.8 รูปแบบแอดเดรส 7 บิทของบัส I^2C

สำหรับการอ้างอิงแอดเดรสแบบ 10 บิท จะต้องใช้ 2 ไบท์โดย 5 บิทในไบท์แรกจะต้องเป็น 11110 เท่านั้น ตามด้วยแอดเดรส 2 บิท และบิทควบคุม R/W ส่วนไบท์ที่สองจะเป็นแอดเดรส 8 บิทที่เหลือ

3.3 การขับโมดูลแสดงผลแบบผลึกเหลว (LCD)

อุปกรณ์ในปัจจุบันนี้ในส่วนแสดงผลนั้นจะใช้ LCD เป็นส่วนใหญ่ไม่ว่าจะเป็นเครื่องเล่นวีดีโอ เครื่องวัดคุมต่าง ๆ เราพอจะแบ่งออกเป็นพวกๆ ได้ดังนี้

1. Character LCD Module
2. Graphic LCD Module
3. Segment Display Type LED Module

ในโมดูล LCD จะมีส่วนประกอบหลัก ๆ 3 ส่วนดังนี้

1. ตัวแสดงผล (Display) ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็น โดยอาศัยแสดงจากภายนอก ดังนั้นจึงต้องเป็นมุมในการมองข้อมูลที่แสดงผลบนจอ LCD

2. ตัวควบคุม (Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาควบคุมการทำงานของโมดูล LCD เช่น ลบจอภาพแสดงตัวอักษร หรือ เลื่อนเคอร์เซอร์ เป็นต้น ตัวควบคุมนี้จะใช้ชิปควบคุมโดยเฉพาะชิปที่นิยมใช้คือเบอร์ HD44780 และ HD44780 จะใช้ควบคุม LCD แบบอักขระส่วน HD61830 จะใช้ควบคุม LCD แบบกราฟฟิก

3. ตัวขับ (Driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงผล ตามที่กำหนดชิปที่ใช้ทำหน้าที่เป็นตัวขับได้แก่ เบอร์ HD444100H และ MSM5259 เป็นต้น

3.3.1 โครงสร้างภายในของตัวควบคุมโมดูล LCD

ในการใช้งานโมดูล LCD จำเป็นต้องทำความเข้าใจเกี่ยวกับโครงสร้างและคำสั่งที่ใช้ในการควบคุมให้ดีเสียก่อนโดยโมดูล LCDแบบอักขระสามารถเข้าใจได้ง่ายจากภาพที่ 3.9 เป็นบล็อกไดอะแกรมภายในของชิปควบคุม LCDเบอร์ HD44780 ซึ่งใช้ในโมดูล LCD แบบอักขระประกอบด้วย

บัฟเฟอร์อินพุทเอาท์พุท เป็นส่วนที่ใช้ในการติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอก เพื่อจะถ่ายทอดข้อมูลเข้าออกภายในตัวควบคุม

รีจิสเตอร์คำสั่ง (Instruction Register: IR) เป็นรีจิสเตอร์ที่ใช้รับข้อมูลคำสั่งจาก อุปกรณ์ภายนอกเพื่อนำไปควบคุมการแสดงผล

รีจิสเตอร์ข้อมูล (Data Register: DR) เป็นรีจิสเตอร์ที่ใช้สำหรับรับข้อมูล จากอุปกรณ์ภายนอก เพื่อถ่ายทอดไปยังหน่วยความจำที่ทำหน้าที่เก็บข้อมูลแสดงผล หรือ นำข้อมูลไปสร้างตัวอักษรเพิ่มเติมในแรมเก็บตัวอักษร

แรมเก็บข้อมูลแสดงผล (Display Data Ram : DDRAM) เป็นหน่วยความจำแรม ทำหน้าที่เก็บข้อมูลที่มาจากรีจิสเตอร์ DR ตัวควบคุมจะนำข้อมูลใน DDRAM ไปเปิดตารางของตัวอักษรที่เก็บไว้ในหน่วยความจำรอมและแรมเก็บตัวอักษรเพื่อนำไปแสดงผล

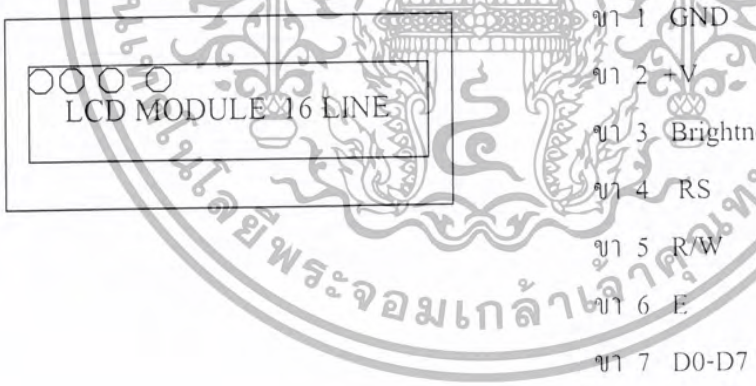
รอมเก็บตัวอักษร (Charater Generator ROM : CGROM) เป็นหน่วยความจำรอมที่ใช้เก็บข้อมูลตัวอักษรหรือสัญลักษณ์ที่สามารถอ่านออกไปแสดงผลได้มีขนาด 7200 บิตโดยจะถูกอ่านด้วยค่าของข้อมูลใน DDRAM

แรมเก็บตัวอักษร (Charater Generator RAM : CGROM) เป็นหน่วยความจำแรมที่ใช้เก็บอักษรที่มีการสร้างเพิ่มเติมขึ้นมาใหม่ ในกรณีที่ตัวอักษรใน CGRAM ไม่เพียงพอ มีขนาด 512 บิท การเขียนและอ่านค่าไปใช้นั้นทำได้เช่นเดียวกับ CGROM คือเขียนข้อมูลลงใน DDRAM แล้วตัวควบคุมจะมาอ่านค่าจาก CGRAM

แฟลก BUSY เป็นส่วนที่ทำหน้าที่แจ้งสถานะการทำงานของตัวควบคุม ให้อุปกรณ์ที่อยู่ภายนอกทราบว่าตัวควบคุมพร้อมที่จะรับข้อมูลหรือคำสั่งหรือไม่ดังนั้นก่อนการส่งข้อมูลหรือคำสั่งมายังตัวควบคุมต้องตรวจสอบสถานะของแฟลก BUSY นี้เสียก่อน

3.3.2โมดูล LCD ขนาด 16 ตัวอักษร 1บรรทัด

สำหรับโมดูล LCD ที่ใช้ในการทำโครงงานเป็นขนาด 16 ตัวอักษร 1บรรทัด เนื่องจากราคาถูก และเป็นโมดูล LCD ที่มีโครงสร้างเป็นมาตรฐาน รวมทั้งมีผู้ผลิตหลายราย และมีการระบุเบอร์ที่แตกต่างกันออกไปตามผู้ผลิต อาทิ เช่น LM020L ของฮิตาชิ , DMC-16117A ของคอปเทร็กซ์ เป็นต้น แต่อย่างไรก็ตามคอนโทรลเลอร์ที่ใช้ ก็อเบอร์เดียวกันคือ HD44780 ของฮิตาชิโมดูลขนาด 16x1 มีขาที่ต่อใช้งานทั้งสิ้น 14 ขามีการจัดขาดังในภาพที่ 3.10



ภาพที่ 3.9 รูปร่างและการจัดขาโมดูล LCD

สำหรับรายละเอียดการทำงานของแต่ละขามีดังนี้

1. Vss (ขา1) ต่อกราวนด์
2. Vdd (ขา2) ต่อไฟเลี้ยง +5 โวลต์
3. Vo (ขา3) เป็นขาอินพุทรับแรงดันเพื่อปรับความเข้มของการแสดงผล

4. RS (ขา4) เป็นขาอินพุทใช้ในการแยกชนิดของข้อมูล ที่ทำการประมวลผลในขณะนั้นว่าเป็นคำสั่งสำหรับรีจิสเตอร์ IR เป็นข้อมูลสำหรับรีจิสเตอร์ DR โดยถ้าขานี้เป็น "0" ข้อมูลที่ส่งมาจะเป็นคำสั่งแต่ถ้าขานี้เป็น "1" ข้อมูลที่ส่งมาจะเป็นข้อมูลสำหรับการแสดงผล

5. R/W (ขา5) เป็นขาที่ใช้เลือกการอ่านหรือเขียนข้อมูลกับLCDถ้าเป็น "0" เป็นการกำหนดให้เขียนข้อมูล แต่ถ้าเป็น "1" จะเป็นการอ่านข้อมูล

6. E (ขา6) เป็นขาอีนาเบิล LCD ให้ทำงาน

7. D0-D7 (ขา7-14) เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่าง LCD กับอุปกรณ์ภายนอกขนาด 8 บิท

บทที่ 4
ทฤษฎีเกี่ยวกับการเขียนโปรแกรม Delphi ขั้นพื้นฐาน

ในอดีตภาษา Pascal เป็นภาษาหนึ่งที่มีผู้นิยมใช้อย่างแพร่หลาย เพราะเป็นภาษาที่ทำความเข้าใจได้ง่าย มีโครงสร้างที่เป็นระบบ และมีประสิทธิภาพสูงทัดเทียมภาษาโปรแกรมภาษาอื่น ซึ่งปัจจุบันโปรมแกรม Borland Delphi 5 ถือเป็นโปรแกรมสร้างแอพพลิเคชั่น ที่มีผู้นิยมใช้มากที่สุด และมีพื้นฐานการทำงานจากโครงสร้างภาษา Pascal ที่เข้าใจง่าย

การพัฒนาแอปพลิเคชั่นขึ้นมาใช้งาน จะต้องมีการเขียนคำสั่งให้ทำงานตามที่ต้องการ โดยใช้ภาษาต่าง ๆ ที่มีอยู่มากมายในปัจจุบันนี้ ภาษาที่กำลังเป็นที่นิยม คือ ภาษาการมองแบบเชิงวัตถุ (Object - Oriented Programming Language) หรือภาษาออบเจ็กต์ และที่นิยมใช้และรู้จักคือภาษา C++ และ javaรวมทั้งภาษา Object Pascalซึ่งจะอยู่ภายใต้โปรแกรมDelphi ใช้ในการเขียนโปรแกรมได้ทั้งแบบทำงานบนระบบ Dos และWindows

4.1 เริ่มต้นทำความรู้จักกับ Object Pascal

เริ่มแรกการเขียนโปรแกรมเพื่อให้คอมพิวเตอร์ทำงาน จะใช้คำสั่งที่เป็นภาษาที่อยู่ในระดับล่างสุด (Lowest Level) ซึ่งมีลักษณะเป็นเลข 0 กับ 1 ที่เรียงต่อกันไปเรื่อย ๆ ซึ่งมีชื่อเรียกว่าภาษาเครื่อง (Machine Language) เช่น 0101010 เป็นต้นแต่เนื่องจากเป็นภาษารูปแบบนี้เข้าใจยากสำหรับผู้เขียนโปรแกรม จึงได้มีการคิดค้นภาษาที่ใช้ชื่อและสัญลักษณ์เป็นคำสั่งแทนภาษาเครื่อง (Lowest Level) โดยชุดคำสั่งนี้มีชื่อเรียกว่า Instruction Set และภาษานี้ถูกเรียกว่า แอสเซมบลี้ (Assembly Language) เช่น mov ax, bx เป็นต้น

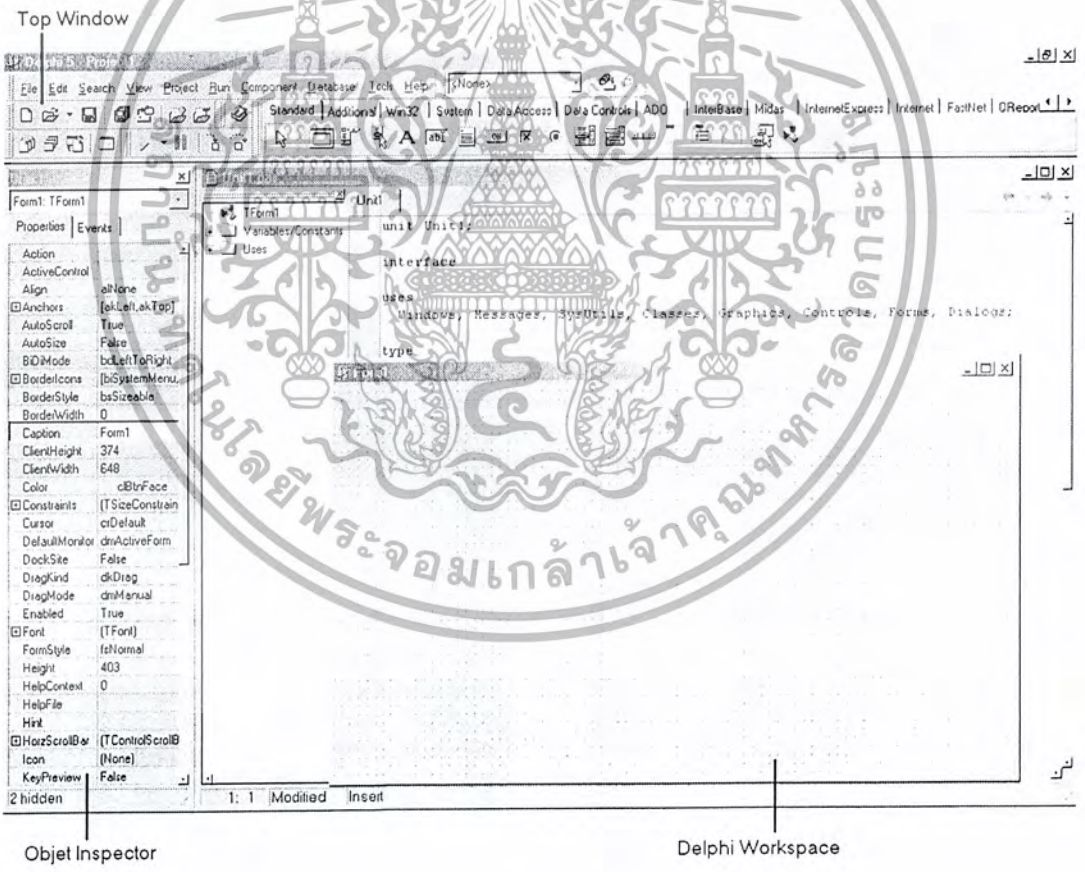
ต่อมาได้มีการพัฒนารูปแบบภาษาให้ผู้เขียนโปรแกรมสามารถทำความเข้าใจได้ง่าย และใช้กับเครื่องคอมพิวเตอร์ได้หลายระบบ (Protability) มากขึ้นซึ่งภาษาเหล่านี้จัดอยู่ในระดับสูง (High Level) โดยที่มีภาษาแรกที่ถูกสร้างขึ้นมาใช้ คือ ภาษา Fortran ในปี ค.ศ. 1954-1957 หลังจากนั้นมีการสร้างภาษา Cobol และ ภาษา Algo60ในปี ค.ศ. 1958-1960 ภาษา BASIC ในปีค.ศ. 1960 ซึ่งภาษา Algo60 (พัฒนาต่อมาเป็น Algo68) ได้กลายเป็นภาษาต้นแบบ ที่ใช้สร้างภาษาอื่น ๆ เช่น Modula-2 , Ada และภาษา Pascal โดย Niklaus Wirth ต่อมาในปี ค.ศ. 1970 Dennis Ritchie ได้สร้างภาษา C ขึ้น ซึ่งภาษา C และ Pascal กลายเป็นภาษาที่นิยมใช้กันมากในช่วงเวลานั้น

ต่อมาได้มีการพัฒนาภาษา C เป็นภาษา C+ + ที่เป็นภาษาเชิงออบเจ็กต์ OOL (Object-Oriented Language) โดย Dr. Bjaine Stroustrup ในปีค.ศ. 1980 ที่มีต้นแบบคือ ภาษา Smalltalk โดยมีแนวคิดแบบคลาส (Class) มาจาก Simula 67

สำหรับภาษา Pascal ที่นิยมใช้จะมีคอมไพเลอร์จากบริษัทบอร์แลนด์ คือ Turbo Pascal สร้างขึ้นในปี ค.ศ. 1985 ซึ่งได้รับความนิยมสูงจากนั้นได้มีการพัฒนาอย่างต่อเนื่อง โดยในที่สุดเปลี่ยนมาเป็น Delphi ที่มีลักษณะแบบวิชวลโปรแกรมมิ่ง (Visual Promming) ในปี ค.ศ. 1995 โดยนำภาษา Pascal มาปรับปรุงใหม่ และเพิ่มเติมในเรื่องของ Object-Oriented Programming พร้อมเปลี่ยนชื่อเรียกเป็นภาษา Object Pascal

4.2 ความสามารถของ Delphi

การเขียนโปรแกรมต่าง ๆ ขึ้นมานั้นจะต้องมีเครื่องมือ (Tools) ที่จะมาช่วยในการเขียนโปรแกรมให้มีความสะดวกและรวดเร็วยิ่งขึ้น ซึ่งในการเขียนโปรแกรมของแต่ละภาษานั้นจะมีเครื่องมือที่เรียกว่า Integrated Development Environment (ide) ใน Delphi ก็เช่นกันจะมี ide ของตนเอง ซึ่ง ide ของ Delphi สามารถพัฒนาโปรแกรมได้ใน 2 ลักษณะคือ Win32 Console Application และ Win32 Graphical User Interface (GUI) Application



ภาพที่ 4.1 หน้าจอของโปรแกรม Delphi

4.2.1 หน้าจอของ Delphi จะแบ่งการทำงานแต่ละกลุ่มออกเป็น 3 ส่วนหลัก

เมื่อเปิดโปรแกรม Delphi ขึ้นมาแล้ว ตัวโปรแกรมนั้นสามารถที่จะแบ่งได้ออกเป็น 3 ส่วนหลัก ๆ ซึ่งแต่ละส่วนจะทำหน้าที่ในการทำงานแตกต่างกันออกไป โดยที่ผู้ใช้งานนั้นสามารถที่จะกำหนดลำดับเหตุการณ์รวมทั้งออกแบบหน้าตาของโปรแกรมและควบคุมการทำงานของโปรแกรมได้จาก 3 ส่วน หลัก ๆ ดังภาพที่ 4.1

ส่วนที่หนึ่ง TOP Window : จะประกอบด้วยส่วนการทำงานที่เป็นMain Menu ,Tools Bar, และ Component Palette โดยที่ Tools Bar จะมีการทำงานเหมือน Main Menu แต่อยู่ในแบบรูปภาพ (Icon) เช่น การเปิดไฟล์ การเซฟไฟล์ และการคอมไพล์โปรแกรม ส่วน Component Palette จะเก็บคอมโพเนนต์ (Component) ต่าง ๆ ที่จะนำไปวางลงในฟอร์มของโปรแกรมที่ต้องการสร้างขึ้นมา เช่น ปุ่มกด ข้อความ ช่องรับข้อความ และอื่น ๆ ในส่วนนี้จะมีการจัดแบ่งคอมโพเนนต์ออกเป็นกลุ่ม ๆ โดยมีแท็บด้านบนจะเป็นตัวบอกชื่อของแต่ละกลุ่ม

ส่วนที่สอง Object Inspector : โดยปกติมักจะอยู่ด้านซ้ายของจอภาพในส่วนนี้มีหน้าที่ในการจัดการกับคอมโพเนนต์ที่นำมาใช้วางลงในฟอร์ม ซึ่งส่วนที่อยู่ข้างบนจะบอกถึงคอมโพเนนต์ที่ถูกเลือกอยู่ว่ามีชื่ออะไร และ เป็นออบเจ็กต์แบบไหน สำหรับช่วงล่างจะมีการแบ่งออกเป็น 2 ส่วน ตามชื่อของแท็บ คือ

- Properties Tab : จะเป็นส่วนที่จะใช้กำหนดคุณสมบัติ (Property) ให้กับคอมโพเนนต์ที่เลือก โดยด้านซ้ายจะเป็นชื่อของแต่ละคุณสมบัติส่วนด้านขวาจะเป็นการกำหนดค่าให้กับคุณสมบัติ ซึ่งค่าต่างๆ ที่กำหนดไว้จะมีผลต่อคอมโพเนนต์ เช่น การกำหนดขนาดของคอมโพเนนต์ ความกว้าง (Width) ความสูง (Height) เป็นต้น

- Events Tab : เป็นส่วนของอีเว้นต์ (Events) ต่างๆ ของคอมโพเนนต์ที่กำลังใช้งานอยู่แต่ละอีเว้นต์เกิดขึ้น และมีการทำงานขึ้นอยู่กับเหตุการณ์ที่ผู้ใช้มีการทำอะไรบางอย่างกับคอมโพเนนต์นั้น ๆ เช่น เมื่อมีการใช้เม้าส์คลิกที่คอมโพเนนต์ก็จะเกิดอีเว้นต์ที่ชื่อ OnMouseDown และเมื่อเกิดอีเว้นต์ก็จะทำให้เมธอดที่เรากำหนดไว้ทำงานขึ้นมาจากรูปเมธอด FormMouse จะถูกเรียกให้ทำงานเมื่อเกิดอีเว้นต์ OnMouseDown ลักษณะการทำงานในส่วนนี้เราเรียกว่า Event Handler ทั้งส่วนที่เป็นคุณสมบัติและอีเว้นต์ของแต่ละคอมโพเนนต์นั้นจะไม่เหมือนกันหมด โดยที่จะขึ้นอยู่กับชนิดของคอมโพเนนต์

ส่วนที่สาม Delphi Workspace : เป็นส่วนสำคัญที่ใช้ในการสร้างโปรแกรมขึ้นมาใช้งาน การสร้างโปรแกรมในแต่ละครั้งจะมีการใช้งานที่แบ่งออกเป็น

- Form Designer : ในส่วนนี้จะมีการทำงานในลักษณะ User Interface เป็นการออกแบบหน้าจอโปรแกรม โดยจะมีฟอร์มในลักษณะเป็นวินโดวส์ และเป็นที่รวบรวม (Container) ของคอมโพเนนต์ต่าง ๆมาวางไว้ เมื่อเริ่มต้นเรียกให้โปรแกรม Delphi ขึ้นมาทำงานจะมีการสร้างฟอร์มที่ชื่อForm 1 โดยอัตโนมัติ

- Code Editor : จะอยู่ด้านหลัง From Designer (ถ้ามองไม่เห็น) เป็นส่วนที่ใช้สำหรับเขียนคำสั่ง Object Pascal ให้โปรแกรมทำงานตามต้องการ เมื่อเริ่มต้นเรียก Delphi ขึ้นมาใช้จะได้ไฟล์ยูนิตชื่อ Unit 1 ส่วนด้านซ้ายจะเป็น Code Explorer ที่จะแสดงให้ทราบว่าโปรแกรมที่สร้างขึ้นมานั้นจะประกอบด้วยส่วนต่าง ๆ อะไรบ้าง และเมื่อมีการเขียนคำสั่งก็จะรู้ว่ากำลังทำงานอยู่ที่ไฟล์ใดหรือส่วนไหนของโปรแกรม

4.3 ระบบการจัดการของโปรแกรม (Program Organization)

การสร้างแอพพลิเคชั่น หรือ โปรแกรมสำเร็จรูปให้ทำงานได้ในแต่ละครั้งนั้น จะต้องมีการคอมไพล์ไฟล์โค้ดคำสั่งมากมายหลายบรรทัด และถ้าหากต้องการเปลี่ยนแปลงแก้ไขแอพพลิเคชั่นเราก็จะต้องนำโค้ดคำสั่งนั้นมาทำการแก้ไขใหม่แล้ว จึงคอมไพล์ใหม่อีกครั้ง ดังนั้นจึงมีไฟล์ต่าง ๆ ที่เกี่ยวข้องในการสร้างแอพพลิเคชั่นมากมาย จะมีไฟล์หลักคือ ไฟล์ที่มีนามสกุล .DPR , .PAS , .DPK ไฟล์ที่ใช้สร้างโปรแกรม คือ ไฟล์ที่มีนามสกุล .DFM , .RES , .DOF , .DSK และ ไฟล์ที่ใช้ในการคอมไพล์ คือ ไฟล์ที่มีนามสกุล .DCU , .Dcp , .BPL (DPL) , .DRC นอกจากนี้เวลาสร้างโปรแกรมขึ้นมาแต่ละครั้ง จึงควรที่จะแยกโฟลเดอร์ในการเก็บไฟล์ต่าง ๆ เหล่านี้ของแต่ละโปรแกรม เพื่อป้องกันความสับสนในการเรียกใช้และการคอมไพล์

4.4 การสร้างโปรแกรมแบบ Windows Application

ในปัจจุบันการทำงานบนวินโดวส์นั้นเป็นที่นิยมมากดังนั้นการสร้างโปรแกรมที่ทำงานบนวินโดวส์ จึงมีความสำคัญ Delphi ก็สนับสนุนการสร้างโปรแกรมบนวินโดวส์ ที่เรียกว่า Win32 Graphical User Interface (GUI) Application จะใช้ Compiler Directive คือ {$APPTYPE GUI} และจะเป็น Delphi Application ที่มีการใช้คอมโพเนนต์ Visual Component Library (VCL) โดยขั้นตอนการสร้างโปรแกรมบนวินโดวส์มีดังนี้

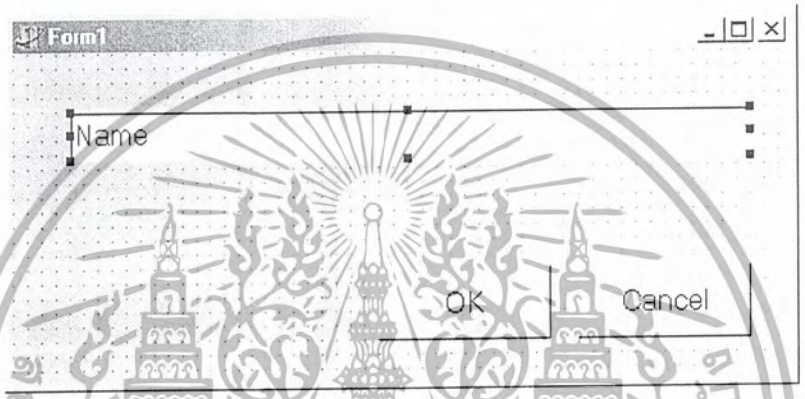
4.4.1 ออกแบบแอพพลิเคชัน

ก่อนอื่นที่เราจะสร้างโปรแกรมขึ้นมาใช้งานตามจุดประสงค์ของเรานั้นสิ่งแรกที่ต้องทำ คือต้องทราบความต้องการในการใช้งานของเราก่อนว่า ต้องการให้โปรแกรมที่เราสร้างขึ้นนั้นมีรูปร่างหน้าตาอย่างไร และ ความสามารถในการทำงานของโปรแกรมที่เราจะสร้างขึ้น มีความสามารถที่

จะทำอะไรได้บ้าง ควรที่จะเขียนลำดับการทำงานที่ชัดเจน หรือ วาดเค้าโครงร่างขั้นตอนในการทำงานให้ชัดเจนบนกระดาษเสียก่อน โดยที่เราอาจจะเขียนลำดับการทำงานของโปรแกรมให้อยู่ในรูปของ Flowchart ก็ได้

4.4.2 ตกแต่งหน้าตาแอพพลิเคชัน

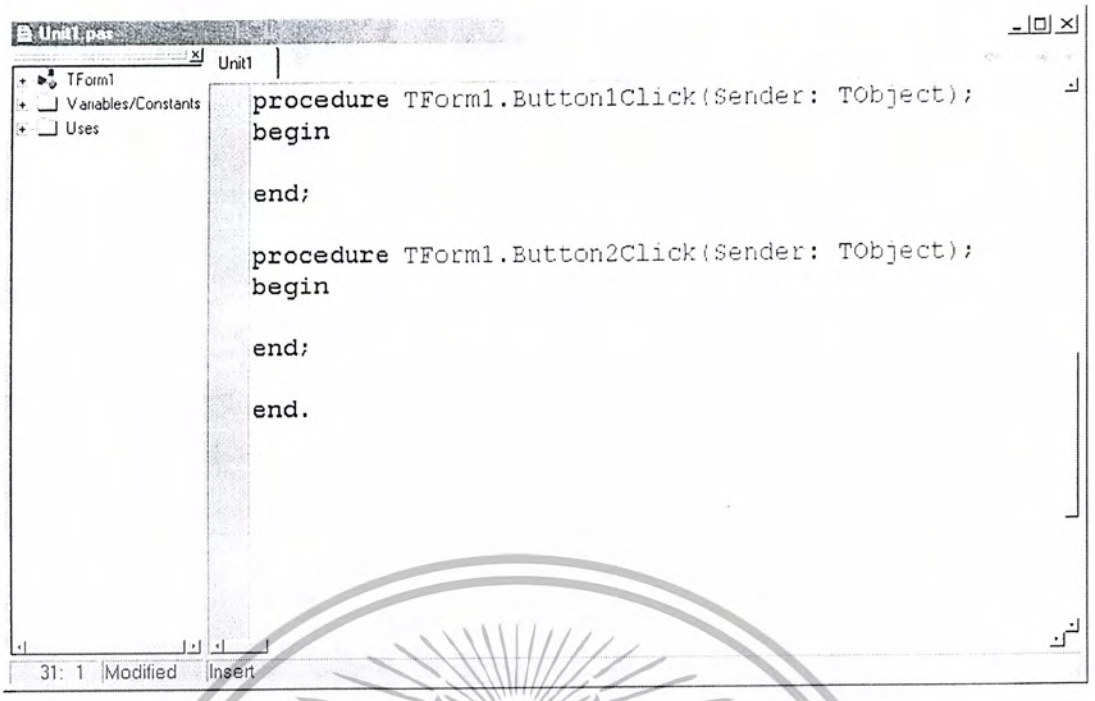
ในขั้นตอนนี้เราจะตกแต่งหน้าตาแอพพลิเคชันที่เราได้ออกแบบไว้ในขั้นตอนแรก โดยจะใส่คอมโพเนนต์ต่าง ๆ ลงใน Form เช่น ใส่ตัว Button , Checkbox , Edit ลงใน From ตามที่เราได้ออกแบบ หรือ ตัวควบคุมอื่นลงใน Form ดังภาพที่ 4.2



ภาพที่ 4.2 Delphi Workspace

4.4.3 เขียนคำสั่งลงใน Code Editor

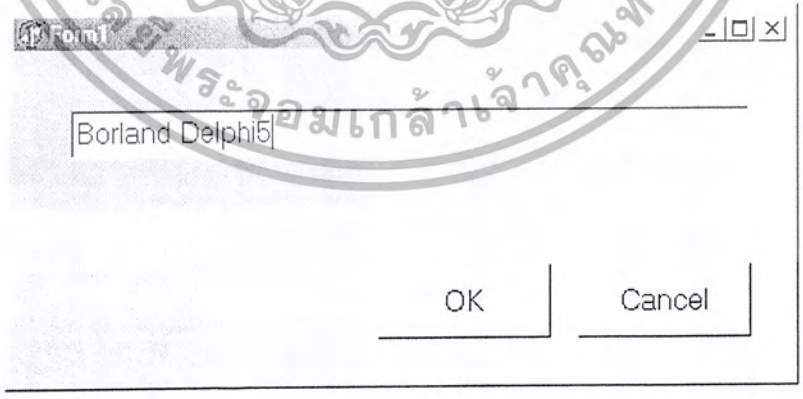
ในส่วนนี้การออกแบบหน้าตาโปรแกรมจะเสร็จสิ้นเรียบร้อยแล้ว แต่จะยังไม่ทำงาน ถ้ายังไม่ได้เขียนคำสั่งให้ตัวโปแกรมทำงานก่อน ซึ่งสามารถที่จะเขียนคำสั่งได้ใน Code Editor โดยให้ดับเบิ้ลคลิกบนคอมโพเนนต์ ที่เราสร้างไว้ โปรแกรม Delphi จะแสดง Code Editor ขึ้นมาข้างบนและสร้างเมธอดให้ ดังภาพที่ 4.3



ภาพที่ 4.3 หน้าต่าง Code Editor

4.4.4 ทดสอบโปรแกรม

หลังจากที่เราเขียน Code Editor เสร็จแล้ว ก็จะมาถึงขั้นตอนที่เราจะทดสอบการทำงานของโปรแกรมที่เราสร้างขึ้นซึ่งประกอบด้วย คอมโพเนนต์ ต่างๆที่ใส่ไว้ในฟอร์ม และคำสั่งที่ควบคุมการทำงานของคอมโพเนนต์ต่างๆ โดยที่สามารถทดสอบได้โดยการกด F9 หลังจากกด F9 แล้วจะได้ ฟอร์มตามภาพที่ 4.4



ภาพที่ 4.4 หน้าตาโปรแกรมที่ทดสอบแล้ว

4.4.5 บันทึกโปรแกรม

เมื่อโปรแกรมเสร็จสมบูรณ์ทั้งส่วนการออกแบบและการเขียนคำสั่ง ก็จะมาถึงขั้นตอนที่เราจะทำการบันทึกไฟล์ เนื่องจากโปรแกรมนี้มีการใช้ไฟล์โปรแกรม และไฟล์ยูนิตโดยคลิ๊กที่ Main menu เลือก File > Save All โปรแกรมจะให้มีการบันทึกไฟล์ยูนิต ก่อนเมื่อ Save ไฟล์ยูนิตแล้วจะต้องทำการบันทึกไฟล์โปรแกรมอีกเหตุที่ต้องบันทึก ไฟล์ยูนิตก่อนเพราะว่าเมื่อมีการเปลี่ยนชื่อของไฟล์ยูนิตแล้ว Delphi จะต้องเปลี่ยนชื่อของไฟล์ยูนิตที่เรียกอยู่ในไฟล์โปรแกรมด้วย และในแต่ละโปรแกรมการตั้งชื่อของไฟล์โปรแกรมและไฟล์ยูนิตจะต้องไม่ซ้ำกันเด็ดขาด

ภาพที่ 4.5 บันทึกโปรแกรม

บทที่ 5
การออกแบบเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคล

5.1 การทำงานของเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคล

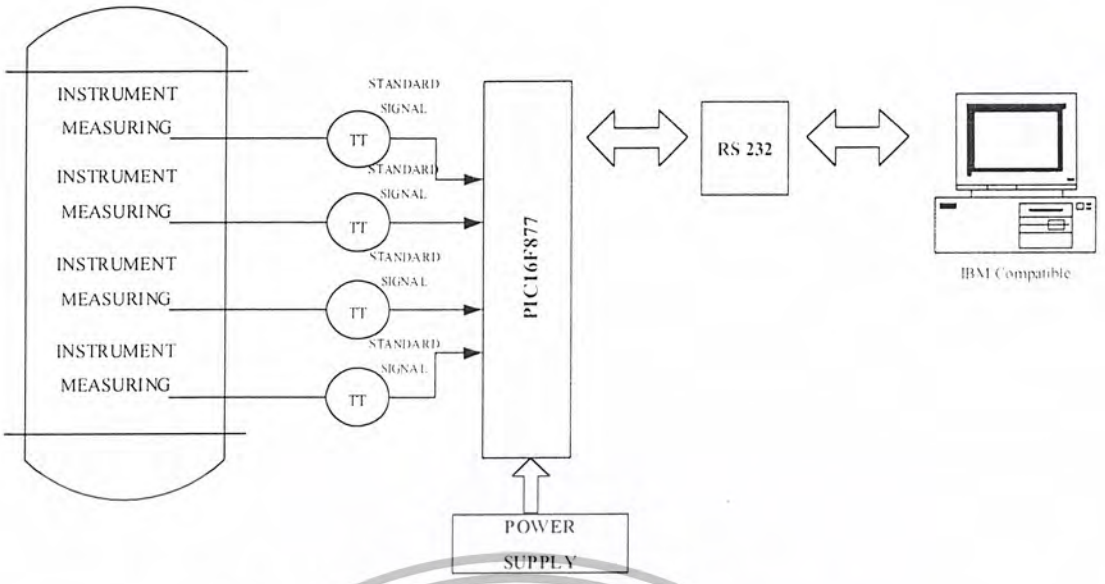
การทำงานของเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคลนั้น จะมีรายละเอียดการทำงานดังนี้

- เก็บข้อมูลที่เป็นสัญญาณอนาลอกในรูปของสัญญาณแรงดัน 1-5 Vdc ได้ 4 ช่องสัญญาณ โดยความสามารถในการเก็บข้อมูลนี้ขึ้นอยู่กับฮาร์ดดิสก์ของเครื่องคอมพิวเตอร์
- ทำการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลขนาด 10 บิท โดยใช้วงจรที่อยู่ภายในตัวไมโครคอนโทรลเลอร์
- สามารถปรับแต่งฐานเวลาในการบันทึกข้อมูลได้ โดยเลือกที่โปรแกรมแอปพลิเคชั่นที่เครื่องคอมพิวเตอร์
- ในขณะที่บันทึกข้อมูล สามารถเรียกดูค่าข้อมูลได้บนหน้าจอคอมพิวเตอร์โดยค่าข้อมูลที่เข้ามาจะถูกบันทึกให้อยู่ในรูปของอุณหภูมิ โดยย่านการวัดสามารถเลือกได้บนเครื่องคอมพิวเตอร์
- การแสดงค่าอุณหภูมิจะแสดงได้ 4 ช่องสัญญาณพร้อมทั้งแสดง วัน เดือน ปี และ ช่วงเวลาที่เก็บ
- สามารถบันทึกข้อมูลอัตโนมัติในแต่ละวันได้ และในกรณีที่ไฟดับหรือสัญญาณขาดหายสามารถบันทึกต่อในวันเดิมเพื่อสะดวกในการเรียกข้อมูลกลับมาดูได้
- สามารถเรียกข้อมูลในแต่ละวันกลับมาดูโดยการเก็บข้อมูลในลักษณะดาต้าล็อกเกอร์
- สามารถพิมพ์กราฟออกทางเครื่องพิมพ์ได้

5.2 โครงสร้างฮาร์ดแวร์ของเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคล

โครงสร้างฮาร์ดแวร์ของเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคลได้ออกแบบแสดงได้ดังภาพที่ 5.1 ซึ่งประกอบไปด้วย

1. ไมโครคอนโทรลเลอร์ PIC16F877 ควบคุมการทำงาน
2. ส่วนแปลงอนาลอกเป็นดิจิตอลอยู่ภายในตัวไมโครคอนโทรลเลอร์ขนาด 10 บิท
3. ส่วนที่ติดต่อกับพอร์ทอนุกรม RS232-c ของเครื่องคอมพิวเตอร์ โดยใช้ IC MAX 232 ระบบแรงดัน
4. เครื่องคอมพิวเตอร์ส่วนบุคคล (PC)
5. แหล่งจ่ายไฟอะแด็ปเตอร์ 12 V/500mA (Supply)
6. อินพุทอนาลอก 4 ช่องสัญญาณ



ภาพที่ 5.1 บล็อกไดอะแกรมของเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคล

5.3 การทำงานเครื่องบันทึกสัญญาณโดยใช้คอมพิวเตอร์ส่วนบุคคล

เครื่องบันทึกสัญญาณโดยใช้ไมโครคอนโทรลเลอร์ PIC16F877 สำหรับควบคุมการทำงาน ซึ่งในไมโครคอนโทรลเลอร์ตัวนี้จะมี ADC ขนาด 10 บิท 8 แชนแนลอยู่ภายใน ซึ่งใช้สำหรับการวัดสัญญาณอนาลอกจากโพรบทั้ง 4 ที่เอาไปติดตั้งเพื่อวัดอุณหภูมิที่ต้องการ จากนั้นหลังจากที่ไมโครคอนโทรลเลอร์ได้รับสัญญาณอนาลอกเข้ามาแล้ว (ระดับแรงดันจะอยู่ในช่วง 0 -5 Vdc) จากนั้น จะแปลงเป็นค่าสัญญาณอนาลอกให้เป็นค่าสัญญาณดิจิตอล แล้วมาเก็บไว้ในหน่วยความจำของไมโครคอนโทรลเลอร์

จากนั้นไมโครคอนโทรลเลอร์จะรอเซ็ตสภาวะที่คอมพิวเตอร์ส่งสัญญาณมาในโปรแกรมนี้ โดยโปรแกรมจะเขียนค่าให้ไมโครคอนโทรลเลอร์ รอให้คอมพิวเตอร์ส่งค่าตัวแปร " X " มาให้ จากนั้นจะทำการส่งค่าสัญญาณดิจิตอลไปให้เครื่องคอมพิวเตอร์แสดงค่าเป็นกราฟต่อไป

ส่วนของการเชื่อมต่อกับเครื่องคอมพิวเตอร์ผ่านพอร์ทอนุกรม RS-232 โดยให้ไอซีเบอร์ MAX 232 เป็นตัวอินเตอร์เฟสระหว่าง PIC16F877 ซึ่งมีโมดูลการสื่อสารข้อมูลอนุกรม (UART) อยู่ภายในเข้ากับเครื่องคอมพิวเตอร์

5.3.1 ไมโครคอนโทรเลอร์

ไมโครคอนโทรเลอร์ PIC16F877 ใช้สัญญาณนาฬิกาจากตัวผลึกคริสตอลที่สร้างความถี่ 4 เมกะเฮิร์ทซ์ เป็นตัวกำหนดจังหวะการทำงาน

5.3.2 วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล

วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล ในตัวไมโครคอนโทรลเลอร์โดยจะใช้วิธีแปลงแบบ Successive Approximation ที่ความละเอียด 10 บิท

5.3.3 การสื่อสารอนุกรมแบบ RS232

ในกรณีนี้จะต้องทำการติดตั้งไอซี Line Driver เพื่อเปลี่ยนระดับสัญญาณทางไฟฟ้าของขาสัญญาณสำหรับรับส่ง ข้อแบบ TTL ของ CPU (RX และ TX) ให้เป็นระดับสัญญาณทางไฟฟ้าแบบ RS232 (± 12V) โดยการติดตั้งไอซีเบอร์ MAX232 เพื่อทำหน้าที่เปลี่ยนระดับสัญญาณ TTL จาขาสัญญาณส่งข้อมูล (TX) ของ CPU ให้เป็นระดับสัญญาณ ± 12 V สำหรับส่งไปยังขารับสัญญาณ (RX) ของอุปกรณ์ภายนอก และในทางกลับกัน ก็จะทำหน้าที่เปลี่ยนระดับสัญญาณส่ง (TX) แบบ RS232 (± 12V) จากอุปกรณ์ภายนอกให้กลับมาเป็นระดับ TTL เพื่อส่งให้กับขารับข้อมูล (RX) ของ CPU ด้วย โดยเมื่อเปลี่ยนระดับสัญญาณในการรับส่งข้อมูลจาก TTL มาเป็นแบบ RS232 นี้แล้วจะทำให้สามารถทำการ รับ- ส่ง ข้อมูลกับอุปกรณ์ภายนอกที่ใช้ระดับสัญญาณทางไฟฟ้าในการ รับ-ส่ง แบบเดียวกัน (RS232) ได้ไกลขึ้น ประมาณ 50 ฟุต หรือ ประมาณ 15 เมตร โดยสามารถทำการ รับ- ส่ง ข้อมูลกับอุปกรณ์ต่างๆ ได้ในลักษณะของตัวต่อตัว (Point - to - Point) เท่านั้น

สำหรับสายสัญญาณที่จะนำมาใช้สำหรับทำการสื่อสารแบบ RS232 นั้นจะใช้สัญญาณเพียง 2-3 เส้นเท่านั้น ทั้งนี้ขึ้นอยู่กับความต้องการในการสื่อสารว่าต้องการสื่อสารแบบทิศทางเดียวหรือสองทิศทาง

- การสื่อสาร RS232 แบบสองทิศทาง ซึ่งจะมีทั้งการรับข้อมูลและส่งข้อมูลไปมา ระหว่างด้านรับ และด้านส่ง โดยในกรณีนี้จะต้องใช้สายสัญญาณจำนวน 3 เส้น สัญญาณรับข้อมูล (RXD) สัญญาณส่งข้อมูล (TXD) และสัญญาณอ้างอิง (GND) โดยในการเชื่อมต่อสายนั้นจะต้องทำการสลับสัญญาณกับอุปกรณ์ปลายทางด้วย คือ สัญญาณส่ง (TXD) จากบอร์ด CP-PIC V3.0&V4.0 จะต้องต่อเข้ากับสัญญาณรับ (RXD) ของอุปกรณ์ และสัญญาณส่ง (TXD) จากอุปกรณ์ก็ต้องต่อกับสัญญาณรับ (RXD) ของบอร์ดส่วนสัญญาณอ้างอิง (GND) จะต้องต่อตรงถึงกัน จึงจะสามารถทำการรับ-ส่ง ข้อมูลถึงกันได้

5.4 โปรแกรมการแสดงผล

โปรแกรมที่ใช้เขียนในการทำปริญญานิพนธ์นี้จะประกอบไปด้วย โปรแกรมภาษาซี ซึ่งจะใช้เขียนโปรแกรมให้กับตัวคอนโทรลเลอร์ และโปรแกรม Delphi ที่ใช้เขียนให้แสดงผลที่หน้าจอคอมพิวเตอร์ มีรายละเอียดดังนี้คือ

5.4.1 แผนผังลำดับการทำงานของโปรแกรมภาษาซี

แผนผังการทำงานจะถูกกำหนดโดยเริ่มแรกตัวไมโครคอนโทรลเลอร์จะต้องกำหนดค่าเริ่มต้น หรือสภาวะเตรียมพร้อมที่จะทำงาน จากนั้นเมื่อตัวตรวจวัด (Prob) ทั้ง 4 ช่องสัญญาณตรวจวัด และผ่านอุปกรณ์เพื่อให้ได้ระดับของสัญญาณมาตรฐาน (1-5 Vdc) โดยผ่านวงจร ADC ภายในตัวไมโครคอนโทรลเลอร์เป็นสัญญาณดิจิตอลและเก็บไว้ในหน่วยความจำภายใน รอกระทั่งมีคำสั่งจากคอมพิวเตอร์ ตัวคอนโทรลเลอร์จะส่งค่าต่อไป

ภาพที่ 5.2 แผนผังลำดับการทำงานของโปรแกรมภาษาซีของไมโครคอนโทรลเลอร์

5.4.2 Source program ภาษา C

```
#include <16F877.h>
#device ADC=10
#include <STDLIB.H>

#USE delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#fuses HS,NOWDT,PUT,NOPROTECT,BROWNOUT,WRT,NOCPD,NOLVP

void atd(void)
{
  unsigned long x,y,z,w;
  set_adc_channel(0);
  delay_us(20);
  x=Read_ADC();
  printf("a%lu",x);
  delay_ms(20);

  set_adc_channel(1);
  delay_us(20);
  y=Read_ADC();
  printf("b%lu",y);

  delay_ms(20);

  set_adc_channel(2);
  delay_us(20);
  z=Read_ADC();
  printf("c%lu",z);
  delay_ms(20);
```

```
  set_adc_channel(3);
  delay_us(20);
  w=Read_ADC();
  printf("d%lu@",w);
  delay_ms(20);
}


void main(void)
{ unsigned char xxx=255;

  setup_port_a(ALL_ANALOG);
  setup_adc(ADC_CLOCK_DIV_2);

  start:  xxx=getch();
     if(xxx=='x')//if rec is 'x' to send value to computer
     {
       atd();

     }

stat:goto start;

 }
```

5.4.3 แผนผังลำดับการทำงานของโปรแกรม Delphi

ในส่วนของโปรแกรมแสดงผลนั้นจะเขียนโดยโปรแกรม Delphi โดยอัตราการเชื่อมต่อชุดไมโครคอนโทรลเลอร์กับคอมพิวเตอร์ มี 3 ค่าคือ 4800 , 9600 , 19200 บิทต่อวินาที เมื่อเปิดโปรแกรมเริ่มทำงานแล้ว หน้าจอหลักจะขึ้นมาเพื่อรับคำสั่งในการโหลดข้อมูลโดยการทำงานของโปรแกรมแบ่งออกเป็น 2 โหมด คือ

1. โหมดแสดงกราฟสัญญาณแบบช่วงเวลาจริง (Real time) มีวิธีการตั้งค่าดังนี้

- เลือกคอมพิวเตอร์ Port (Com1, Com2, Com3, Com4)

- เลือกอัตราบอตเรต (4800,9600,19200)
- กดปุ่ม Start
- กดปุ่ม Real Time

2. โหมดดูข้อมูลย้อนหลัง(Dara logger)

- เลือก Computer Port (Com1, Com2, Com3, Com4)
- เลือกวันที่ต้องการ (ดับเบิ้ลคลิ๊ก)
- กดปุ่ม Show Data To Grid
- เลือกช่องสัญญาณที่ต้องการดู

โดยกราฟนั้นแต่ละค่าจะมีค่าอยู่แล้วโดยแกน Y จะเป็นค่าของอุณหภูมิส่วนในแกน X จะเป็นค่าของ เวลา วัน เดือน และ ปี

5.4.4 Source Code ของโปรแกรม Delphi

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  CommDrv, Grids, StdCtrls, ExtCtrls, FileCtrl, ComCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    ra1: TRadioGroup;
    ra2: TRadioGroup;
    ra3: TRadioGroup;
    st1: TStringGrid;
    com1: TCommPortDriver;
    Edit1: TEdit;
    Button1: TButton;
    Label1: TLabel;
    f1: TFileListBox;
    DirectoryListBox1: TDirectoryListBox;
    DriveComboBox1: TDriveComboBox;
    Timer1: TTimer;
    Button2: TButton;
    Label2: TLabel;
    ri: TRichEdit;
    Button3: TButton;
    ri1: TRichEdit;
    Button5: TButton;
    Button6: TButton;
```

```
    max: TEdit;
    min: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    Button7: TButton;
    Button4: TButton;
    CheckBox1: TCheckBox;
    procedure com1ReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ra3Click(Sender: TObject);
    procedure ra2Click(Sender: TObject);
    procedure f1DblClick(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure ra1Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  status:string;
  statuSREALTIME:integer;
```

```
implementation

{$R *.DFM}


procedure TForm1.com1ReceiveData(Sender: TObject;
 DataPtr: Pointer; DataSize: Integer);
var p: pchar;
   s: string;
begin
{
protocal มีดังนี้ aแล้วตามด้วยข้อมูล
            bแล้วตามด้วยข้อมูล
            cแล้วตามด้วยข้อมูล
            dแล้วตามด้วยข้อมูล
            @

ex
  a123b346c890d894@
  123,346,890,894,
}
 p := DataPtr;
 while DataSize > 0 do
 begin
  if(P^='@')then
  begin
    status:='finish';
   if button2.Caption='start' then
   begin
    timer1.Enabled:=false;
    com1.disconnect;
   end;
```

```
  ri1.Text:=ri1.text+',';
end
else
if(P^='a')then
begin
  ri1.text:=ri1.text+',';
end
else
if(P^='b')then
begin
  ri1.text:=ri1.text+',';
end
else
if(P^='c')then
begin
  ri1.text:=ri1.text+',';
end
else
if(P^='d')then
begin
  ri1.text:=ri1.text+',';
end
else
begin
  ri1.text:=ri1.text+p^;
end;

  dec( DataSize );
  inc( p );
end;
```

```
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
timer1.Interval:=strtoint(edit1.text);
end;

procedure TForm1.Button2Click(Sender: TObject);
var sc:string;
var xx:integer;
begin

if button2.Caption='start' then
begin

    button2.Caption:='stop';
    sc:=datetostr(now);

   for xx:=0 to strlen(pchar(sc)) do
   begin
    if sc[xx] ='/' then
    begin
       sc[xx]:='_';
    end;
   end;


    //ถ้ามีไฟล์นี้อยู่ก็ทำการเปิดไฟล์นี้ขึ้นมาเพื่อต่อข้อมูล แต่จะโหลดเมื่อไม่มีการ load เท่านั้น
    if(FileExists(sc+'.jjj'))then
    begin
       ri1.Lines.LoadFromFile(sc+'.jjj');
       ri1.update;
```

```
    sleep(1000);
  end;


 com1.connect;
 timer1.Enabled:=true;



end
else
begin
  button2.Caption:='start';
//ยังจำเป็นต้องอ่านข้อมูลเข้ามาก่อนเพราะอาจจะมีข้อมูลที่ยังค้างอยู่อีกต้องรับมาให้หมด
com1.Disconnect;
end;

end;

procedure TForm1.Timer1Timer(Sender: TObject);
var sc:string;
var xx:integer;
begin
//ตรวจสอบ วันเวลาและเซฟ
//form1.caption:=datetostr(now);


if button2.Caption='stop' then  //ถ้ากดปุ่ม start อยู่ก็ให้อ่านค่าจาก atd
begin
  if status='finish' then
  begin
```

การออกแบบเครื่องบันทึกสัญญาณโดยคอมพิวเตอร์ส่วนบุคคล

DESIGN OF RECORDER USING PERSONAL COMPUTER

นาย ชยวี คุณโลกย์

นาย ธเนศ จงศุภนิมิต

นาย วุฒิชัย แก้วประเสริฐ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมการวัดคุม

ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

```
    xx:=0;
    sc:=datetostr(now);
   for xx:=0 to strlen(pchar(sc)) do
   begin
    if sc[xx] ='/' then
    begin
       sc[xx]:='_';
    end;
   end;

    ri1.Lines.SaveToFile(sc+'.jjj');
    ri1.text:=ri1.text+timetostr(now);
    com1.SendString('x');//request data from atd
    status:='';



    //com1.SendString('x');// บอก pic ว่าต้องการข้อมูลชุดถัดไป
  end;
end;

f1.Update;

//แสดงการ ploat แบบ realtime เมื่อวันที่และเวลาตรงกันกับเวลาปัจจุบัน
if statuSREALTIME=8then
begin
 ri.text:=ri1.text;
 sleep(10);
 ri.Update;
 Button3Click(Sender);
```

```
 sleep(500);
 ra1Click(sender);
end;



end;

procedure TForm1.FormCreate(Sender: TObject);
begin
st1.Cells[0,0]:='record';
st1.Cells[1,0]:='datetime';
st1.Cells[2,0]:='tmp1';
st1.Cells[3,0]:='tmp2';
st1.Cells[4,0]:='tmp3';
st1.Cells[5,0]:='tmp4';
st1.Cells[6,0]:='diff2';
st1.Cells[7,0]:='diff3';
st1.Cells[8,0]:='diff4';
status:='finish';
if FileExists(datetostr(now)) then
begin
 ri1.Lines.LoadFromFile(datetostr(now));
end;



end;

procedure TForm1.ra3Click(Sender: TObject);
begin
if ra3.ItemIndex=0 then
```

```
begin
com1.ComPortSpeed:=br4800;
end
else
if ra3.ItemIndex=1 then
begin
com1.ComPortSpeed:=br9600;
end
else
if ra3.ItemIndex=2 then
begin
com1.ComPortSpeed:=br19200;
end;



end;

procedure TForm1.ra2Click(Sender: TObject);
begin
if ra2.ItemIndex=0 then
begin
com1.ComPort:=pncom1;

end
else
if ra2.ItemIndex=1 then
begin
com1.ComPort:=pncom2;
end
else
if ra2.ItemIndex=2 then
begin
```

```
com1.ComPort:=pncom3;
end
else
if ra2.ItemIndex=3 then
begin
com1.ComPort:=pncom4;
end;

end;

procedure TForm1.f1DblClick(Sender: TObject);
//ตรวจสอบว่าใช่วันที่ปัจจุบันไหม ถ้าใช่ก็ทำการแสดงสถานะว่า ploat ได้แบบ realtime
var sc:string;
var xx:integer;
begin
 sc:=datetostr(now);
   for xx:=0 to strlen(pchar(sc)) do
   begin
    if sc[xx] ='/' then
    begin
       sc[xx]:='_';
    end;
   end;

 if(f1.filename=f1.Directory+'\'+sc+'.jjj')then
 begin
   statuSREALTIME:=8;
//   form1.caption:=f1.Directory+'\'+sc+'.jjj';

 end
 ELSE
 BEGIN
```

```
  statuSREALTIME:=0;
END;

ri.Lines.LoadFromFile(f1.FileName);
ri.Update;
sleep(1000);

end;

procedure TForm1.Button3Click(Sender: TObject);
var x,coll,roww:integer;
var d:string;
begin

for x:=1 to st1.rowcount do
begin
st1.rows[x].Clear;
end;

//st1.Cols.LoadFromFile

ri.selstart:=0;
ri.SelLength:=1;
x:=0;
d:='';
coll:=1;
roww:=1;
```

```
while ri.seltext<>'' do
begin

   ri.selstart:=x;
   ri.SelLength:=1;

 if(ri.seltext<>',')then
 begin
   d:=ri.SelText;
//   memo1.text:=memo1.text+d;
//   memo1.update;
   st1.Cells[coll,roww]:=st1.Cells[coll,roww]+d;
 end
 else
 begin
   d:='';
   coll:=coll+1;
   if coll>5 then
   begin
    roww:=roww+1;
    if roww>=st1.RowCount then
    begin
      st1.rowcount:=st1.rowcount+1;
    end;
    coll:=1;
   end;
 end;
   x:=x+1;

end;
```

```
//////// differrent of temperature
   roww:=1;
   while((st1.Cells[2,roww]<>'')and(st1.Cells[3,roww]<>''))do
   begin
      st1.Cells[6,roww]:= inttostr(strtoint(st1.Cells[2,roww])-strtoint(st1.Cells[3,roww]));
      inc(roww);
   end;

   roww:=1;
   while((st1.Cells[2,roww]<>'')and(st1.Cells[4,roww]<>''))do
   begin
      st1.Cells[7,roww]:= inttostr(strtoint(st1.Cells[2,roww])-strtoint(st1.Cells[4,roww]));
      inc(roww);
   end;

   roww:=1;
   while((st1.Cells[2,roww]<>'')and(st1.Cells[5,roww]<>''))do
   begin
      st1.Cells[8,roww]:= inttostr(strtoint(st1.Cells[2,roww])-strtoint(st1.Cells[5,roww]));
      inc(roww);
   end;


   roww:=1;
   while((st1.Cells[2,roww]<>'')and(st1.Cells[6,roww]<>''))do
   begin
      st1.Cells[9,roww]:= inttostr(strtoint(st1.Cells[2,roww])-strtoint(st1.Cells[6,roww]));
      inc(roww);
   end;



end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
var sc:string;
begin
end;

procedure TForm1.ra1Click(Sender: TObject);
var x,xx,xv,xb,xn,coll,roww,rowwx:integer;
var d:string;
begin

   coll:=1;
   roww:=1;
   x:=20;
if not(checkbox1.Checked) then
begin
   image1.canvas.Brush.Color:=clwhite;
   image1.canvas.Rectangle(0,0,image1.width,image1.height+1);
end;
   for xv:=0 to image1.Width-1 do
   begin
    xb:=xv*40;
    image1.canvas.MoveTo(xb,image1.height);
    image1.canvas.lineTo(xb,image1.height-5);

   end;
```

```
for xv:=0 to image1.height-1 do
begin
 xb:=xv*4;
 image1.canvas.MoveTo(1,xb);
 image1.canvas.lineTo(10,xb);

end;



image1.update;


if ra1.ItemIndex=0 then
begin

 while((st1.Cells[1,roww]<>'')and(st1.Cells[2,roww]<>''))do
 begin

 //กำหนดช่วง

if(strtoint(st1.Cells[2,roww])>=strtoint(min.text))and(strtoint(st1.Cells[2,roww])<=strtoint(max.te
xt)) then
 begin

 //พล๊อตแบบตามข้อมูล date time
 // image1.Canvas.Rectangle(strtoint(st1.Cells[1,roww])+3,image1.height-
strtoint(st1.Cells[2,roww])+3,strtoint(st1.Cells[1,roww]),image1.height-
strtoint(st1.Cells[2,roww]));
 //ทำการลบหน้าจอก่อนให้เป็นสีขาว
```

```
image1.Canvas.pen.Color:=clblue;

image1.Canvas.Rectangle(x+3,image1.height-strtoint(st1.Cells[2,roww])+3,x.image1.height-
strtoint(st1.Cells[2,roww]));
//      form1.canvas.Font.Size:=3;
image1.canvas.Font.color:=clblue;
image1.Canvas.TextOut(x,3,st1.Cells[1,roww]);
image1.canvas.Font.color:=clgreen;

image1.Canvas.TextOut(3+x,image1.height-strtoint(st1.Cells[2,roww])-
20,st1.Cells[2,roww]);

if strtoint(min.text)=0 then
begin
if roww = 1 then
begin
 image1.canvas.moveto(x,image1.height-strtoint(st1.Cells[2,roww]));

 if  st1.Cells[2,roww+1]<>'' then
 begin
 image1.canvas.lineto(x+50,image1.height-strtoint(st1.Cells[2,roww+1]));
 end;

end
else
begin
 if  st1.Cells[2,roww+1]<>'' then
 begin
   image1.canvas.moveto(x,image1.height-strtoint(st1.Cells[2,roww]));
  //ทำการตรวจสอบช่วง range อีกครั้งนึงเพื่อป้องกันช่วงที่ไม่ต้องการ
   image1.canvas.lineto(x+50,image1.height-strtoint(st1.Cells[2,roww+1]));
```

```
      end;
      end;

    end;

  end;

    roww:=roww+1;
    x:=x+50;

  end;
 end
 else
 if ra1.ItemIndex=1 then
 begin

  while((st1.Cells[1,roww]<>'')and(st1.Cells[3,roww]<>''))do
  begin
//พล๊อตแบบตามข้อมูล date time
  //ทำการลบหน้าจอก่อนให้เป็นสีขาว
  //กำหนดช่วง

if(strtoint(st1.cells[3,roww])>=strtoint(min.text))and(strtoint(st1.cells[3,roww])<=strtoint(max.te
xt)) then
  begin

  //พล๊อตแบบตามข้อมูล date time
  //  image1.Canvas.Rectangle(strtoint(st1.Cells[1,roww])+3,image1.height-
strtoint(st1.cells[3,roww])+3,strtoint(st1.Cells[1,roww]),image1.height-
strtoint(st1.cells[3,roww]));
```

```
//ทำการลบหน้าจอก่อนให้เป็นสีขาว


      image1.Canvas.pen.Color:=clred;
      image1.Canvas.Rectangle(x+3,image1.height-strtoint(st1.cells[3,roww])+3,x,image1.height-
strtoint(st1.cells[3,roww]));
//       form1.canvas.Font.Size:=3;
      image1.canvas.Font.color:=clred;
      image1.Canvas.TextOut(x,3,st1.Cells[1,roww]);
      image1.canvas.Font.color:=clgreen;

      image1.Canvas.TextOut(3+x,image1.height-strtoint(st1.cells[3,roww])-
20,st1.cells[3,roww]);

      if strtoint(min.text)=0 then
      begin
      if roww =1 then
      begin
       image1.canvas.moveto(x,image1.height-strtoint(st1.cells[3,roww]));

       if  st1.cells[3,roww+1]<>'' then
       begin
       image1.canvas.lineto(x+50,image1.height-strtoint(st1.cells[3,roww+1]));
       end;

      end
      else
      begin
       if  st1.cells[3,roww+1]<>'' then
       begin
         image1.canvas.moveto(x,image1.height-strtoint(st1.cells[3,roww]));
         //ทำการตรวจสอบช่วง range อีกครั้งนึงเพื่อป้องกันช่วงที่ไม่ต้องการ
```

```
        image1.canvas.lineto(x+50,image1.height-strtoint(st1.cells[3,roww+1]));
      end;
      end;


     end;


   end;

      roww:=roww+1;
      x:=x+50;

   end;
  end
  else
  if ra1.ItemIndex=2 then
  begin

   while((st1.Cells[1,roww]<>'')and(st1.Cells[4,roww]<>''))do
   begin
//พล๊อตแบบตามข้อมูล date time
//  image1.Canvas.Rectangle(strtoint(st1.Cells[1,roww])+3,image1.height-
strtoint(st1.Cells[2,roww])+3,strtoint(st1.Cells[1,roww]),image1.height-
strtoint(st1.Cells[2,roww]));
   //ทำการลบหน้าจอก่อนให้เป็นสีขาว
   //กำหนดช่วง

if(strtoint(st1.cells[4,roww])>=strtoint(min.text))and(strtoint(st1.cells[4,roww])<=strtoint(max.te
xt)) then
   begin

   //พล๊อตแบบตามข้อมูล date time
```

```
    // image1.Canvas.Rectangle(strtoint(st1.Cells[1,roww])+3,image1.height-
strtoint(st1.cells[4,roww])+3,strtoint(st1.Cells[1,roww]),image1.height-
strtoint(st1.cells[4,roww]));
   //ทำการลบหน้าจอก่อนให้เป็นสีขาว


      image1.Canvas.pen.Color:=cllime;

      image1.Canvas.Rectangle(x+3,image1.height-strtoint(st1.cells[4,roww])+3,x,image1.height-
strtoint(st1.cells[4,roww]));
//       form1.canvas.Font.Size:=3;
      image1.canvas.Font.color:=cllime;
      image1.Canvas.TextOut(x,3,st1.Cells[1,roww]);
      image1.canvas.Font.color:=clgreen;

      image1.Canvas.TextOut(3+x,image1.height-strtoint(st1.cells[4,roww])-
20,st1.cells[4,roww]);

      if strtoint(min.text)=0 then
      begin
      if roww = 1 then
      begin
       image1.canvas.moveto(x,image1.height-strtoint(st1.cells[4,roww]));

       if st1.cells[4,roww+1]<>'' then
       begin
       image1.canvas.lineto(x+50,image1.height-strtoint(st1.cells[4,roww+1]));
       end;

      end
      else
      begin
```

```
if st1.cells[4,roww+1]<>'' then
begin
  image1.canvas.moveto(x,image1.height-strtoint(st1.cells[4,roww]));
 //ทำการตรวจสอบช่วง range อีกครั้งนึงเพื่อป้องกันช่วงที่ไม่ต้องการ
  image1.canvas.lineto(x+50,image1.height-strtoint(st1.cells[4,roww-1]));
end;
end;

end;

end;

  roww:=roww+1;
  x:=x+50;

end;
end
else
if ra1.ItemIndex=3 then
begin

while((st1.Cells[1,roww]<>'')and(st1.Cells[5,roww]<>''))do
begin
//พล๊อตแบบตามข้อมูล date time
//  image1.Canvas.Rectangle(strtoint(st1.Cells[1,roww])+3,image1.height-
strtoint(st1.Cells[2,roww])+3,strtoint(st1.Cells[1,roww]),image1.height-
strtoint(st1.Cells[2,roww]));
  //ทำการลบหน้าจอก่อนให้เป็นสีขาว
  //กำหนดช่วง
```

```
if(strtoint(st1.cells[3,roww])>=strtoint(min.text))and(strtoint(st1.cells[5,roww])<=strtoint(max.te
xt)) then
   begin

   //พล๊อตแบบตามข้อมูล date time
   //  image1.Canvas.Rectangle(strtoint(st1.Cells[1,roww])+3,image1.height-
strtoint(st1.cells[5,roww])+3,strtoint(st1.Cells[1,roww]),image1.height-
strtoint(st1.cells[5,roww]));
   //ทำการลบหน้าจอก่อนให้เป็นสีขาว


      image1.Canvas.pen.Color:=clnavy;
      image1.Canvas.Rectangle(x+3,image1.height-strtoint(st1.cells[5,roww])+3,x,image1.height-
strtoint(st1.cells[5,roww]));
//       form1.canvas.Font.Size:=3;
      image1.canvas.Font.color:=clnavy;
      image1.Canvas.TextOut(x,3,st1.Cells[1,roww]);
      image1.canvas.Font.color:=clgreen;


      image1.Canvas.TextOut(3+x,image1.height-strtoint(st1.cells[5,roww])-
20,st1.cells[5,roww]);


      if strtoint(min.text)=0 then
      begin
      if roww = 1 then
      begin
       image1.canvas.moveto(x,image1.height-strtoint(st1.cells[5,roww]));


       if  st1.cells[5,roww+1]<>'' then
       begin
       image1.canvas.lineto(x+50,image1.height-strtoint(st1.cells[5,roww+1]));
```

```
        end;


       end
       else
       begin
        if  st1.cells[5,roww+1]<>'' then
        begin
          image1.canvas.moveto(x,image1.height-strtoint(st1.cells[5,roww]));
         //ทำการตรวจสอบช่วง range อีกครั้งนึงเพื่อป้องกันช่วงที่ไม่ต้องการ
          image1.canvas.lineto(x+50,image1.height-strtoint(st1.cells[5,roww+1]));
       end;
       end;


       end;


     end;


       roww:=roww+1;
       x:=x+50;


    end;
  end;



end;
```

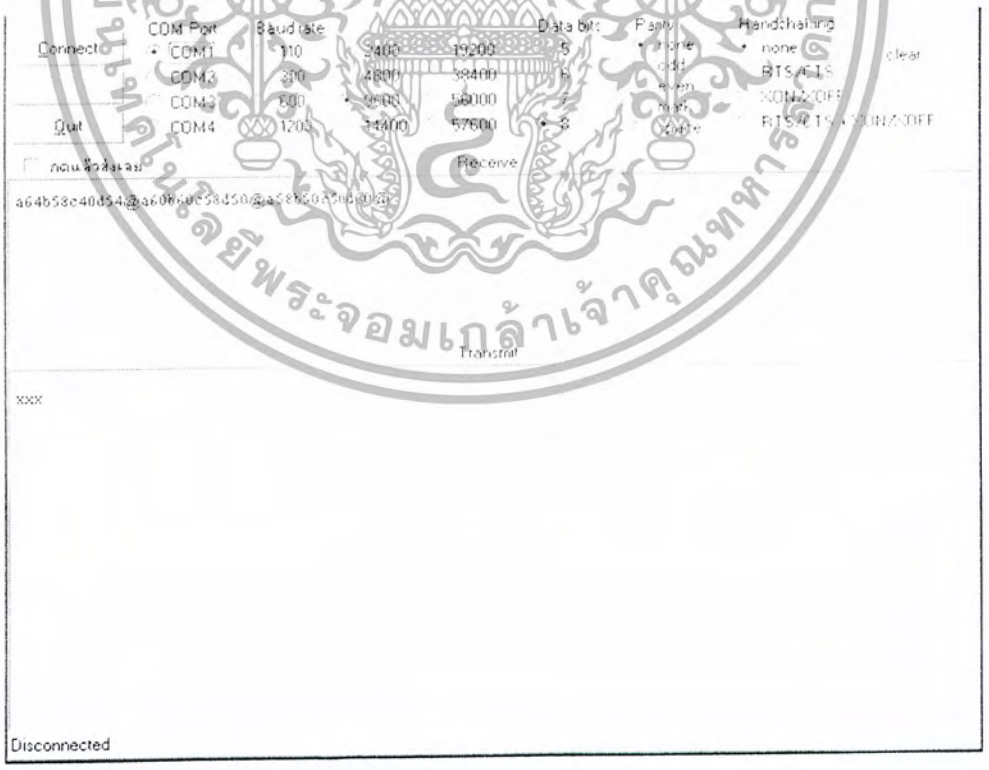
```
procedure TForm1.Button5Click(Sender: TObject);
begin
image1.Picture.SaveToFile('c:\bitmap.bmp');
winexec('C:\Program Files\Accessories\MSPAINT.EXE c:\bitmap.bmp',1);

end;

procedure TForm1.Button6Click(Sender: TObject);
begin
ri.text:='';
end;

procedure TForm1.Button7Click(Sender: TObject);
begin

if 'y'=inputbox('delete file', 'type y no type n','n') then
begin
deletefile(f1.filename);
f1.Update;
end;

end;

end.
```

บทที่ 6

การทดสอบการทำงาน

6.1 ทดสอบการส่งค่าระหว่างฮาร์ดแวร์กับคอมพิวเตอร์

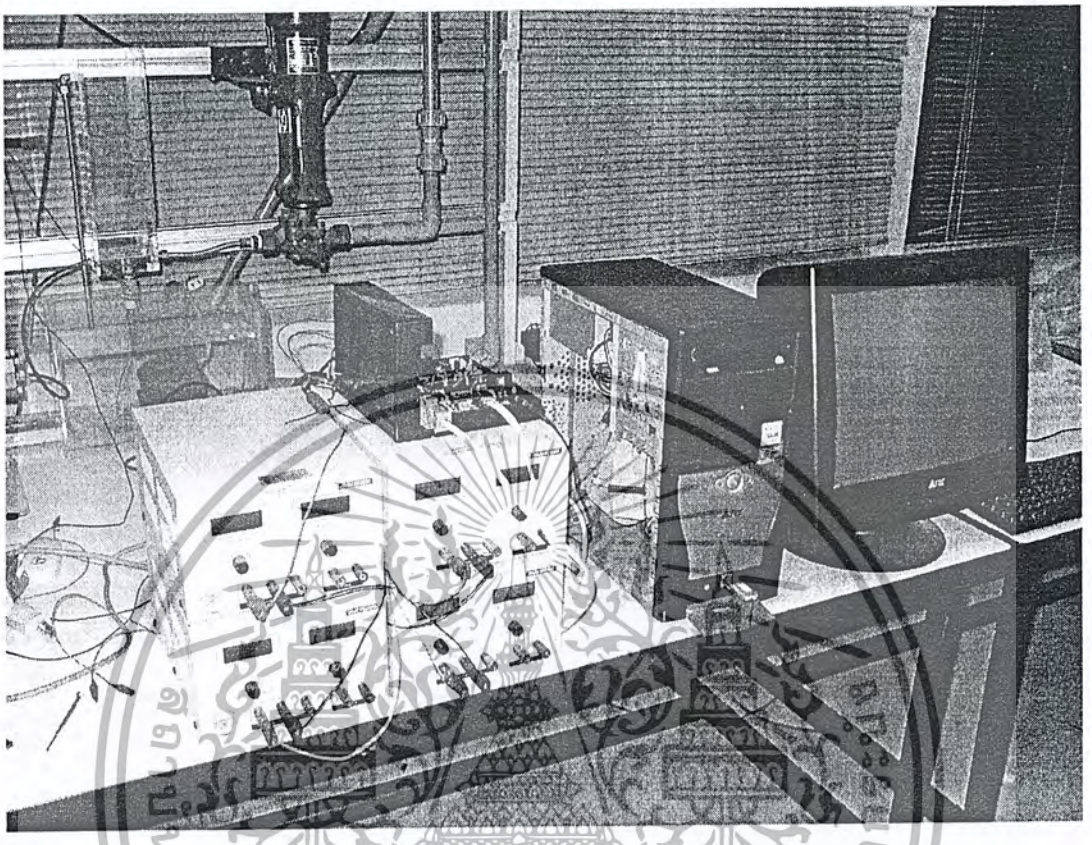
1. เชื่อมต่อชุดไมโครคอนโทรลเลอร์กับคอมพิวเตอร์
2. ทำการป้อนโปรแกรมให้ไมโครคอนโทรลเลอร์
3. ตั้งค่าโปรโตคอลระหว่างฮาร์ดแวร์กับคอมพิวเตอร์ดังนี้
   - a ค่าอุณหภูมิที่วัดได้
   - b ค่าอุณหภูมิที่วัดได้
   - c ค่าอุณหภูมิที่วัดได้
   - d ค่าอุณหภูมิที่วัดได้
   - @
4. ป้อนสัญญาณอินพุท (supply) เป็นแรงดันอนาลอก 1 ถึง 5 Vdc
5. ตรวจสอบการส่งค่าโดยใช้โปรแกรมที่เขียนขึ้น ดังภาพที่ 6.1
6. ปรับค่าสัญญาณอินพุทเพื่อสังเกตการเปลี่ยนแปลง



ภาพที่ 6.1 การทดสอบส่งค่าระหว่างฮาร์ดแวร์กับคอมพิวเตอร์โดยโปรแกรม Comtest

6.2 ทดสอบกับเครื่องจ่ายแรงดันแบบปรับค่าได้ (Variable Power Supply)

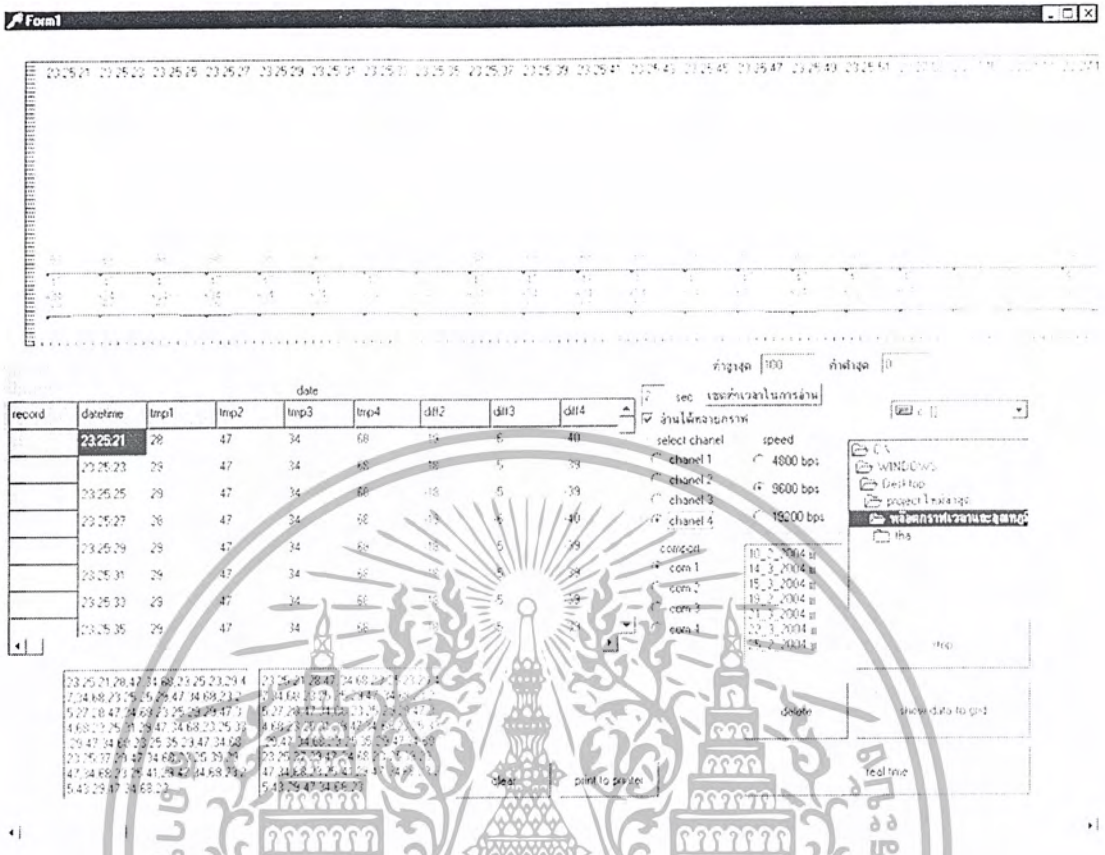
1. เชื่อมต่อชุดไมโครคอนโทรลเลอร์กับคอมพิวเตอร์ ดังภาพที่ 6.2



ภาพที่ 6.2 การเชื่อมต่อเครื่องจ่ายแรงดันแบบปรับค่าได้กับคอมพิวเตอร์

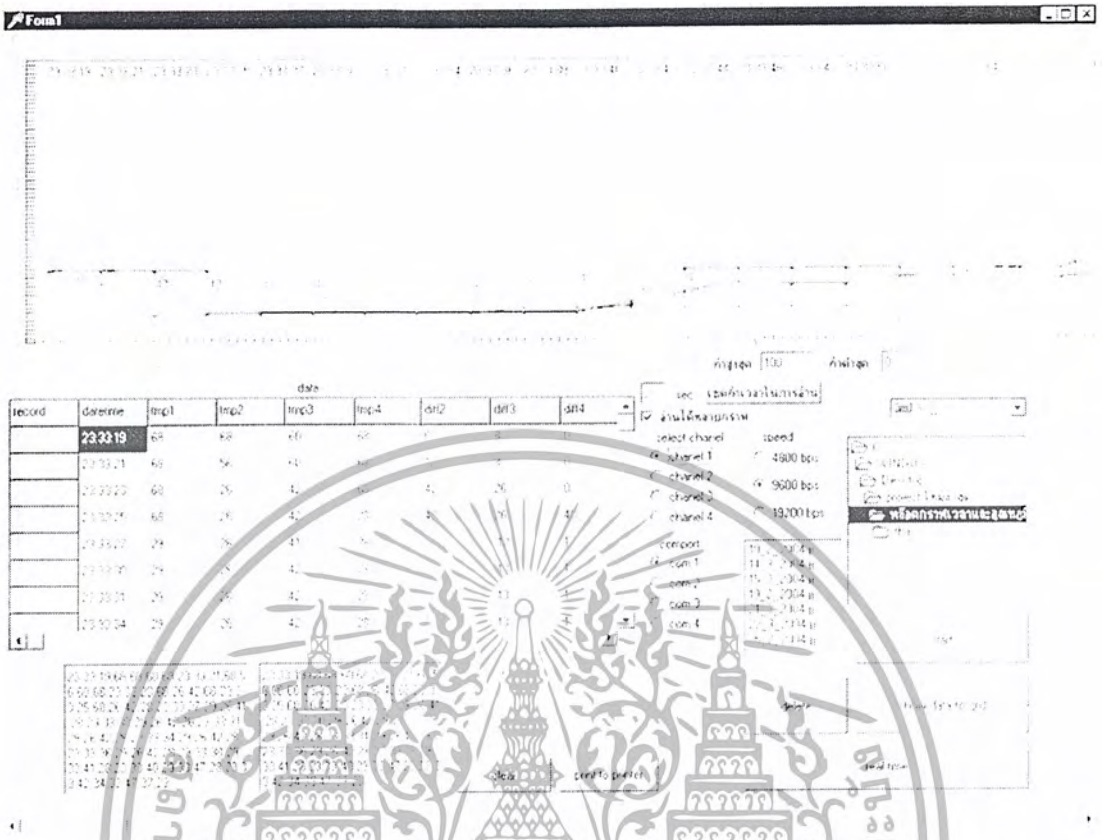
2. ป้อนสัญญาณอินพุท (supply) เป็นแรงดันอนาลอก 1 ถึง 5 Vdc

3. อ่านค่าที่ได้จากกราฟและบันทึกผล ดังภาพที่ 6.3



ภาพที่ 6.3 ค่าที่ได้จากการเชื่อมต่อเครื่องจ่ายแรงดันแบบปรับค่าได้กับคอมพิวเตอร์

4. ปรับระดับแรงดันของเครื่องจ่ายแรงดันแบบปรับค่าได้และบันทึกผล ดังภาพที่ 6.4



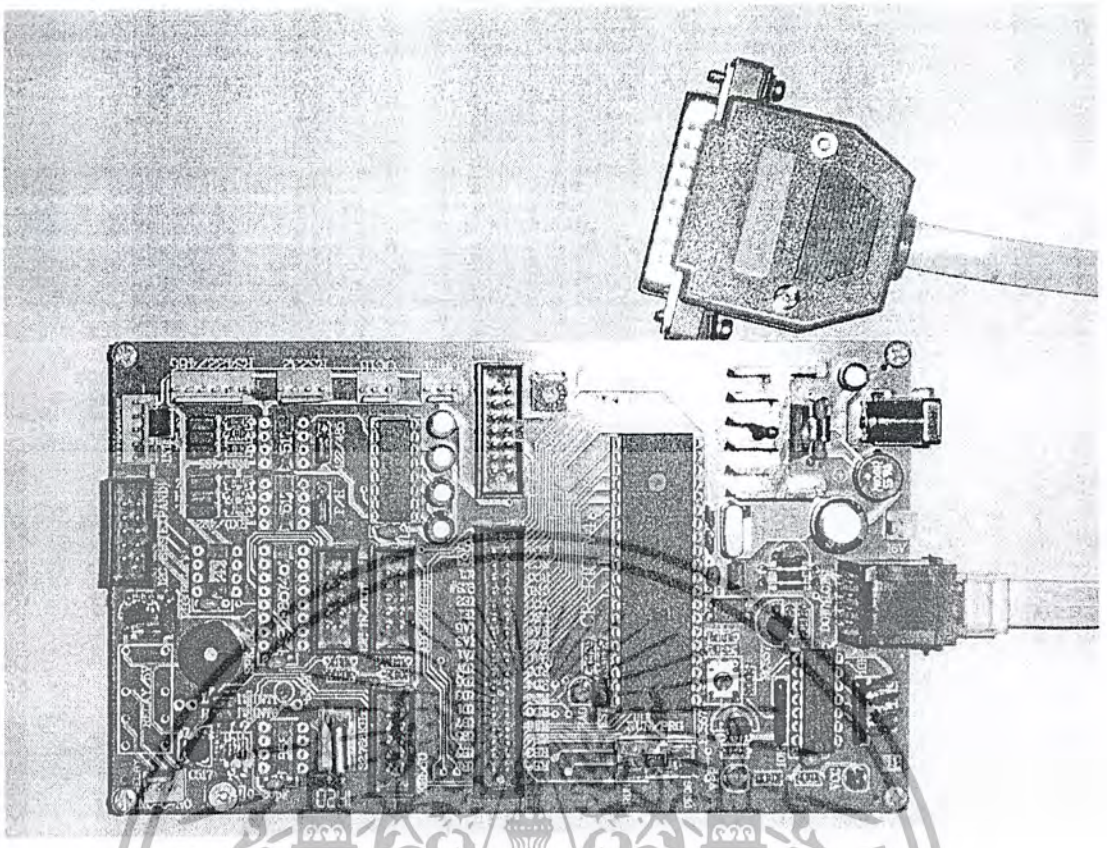
ภาพที่ 6.4 ค่าที่ได้จากปรับระดับแรงดันของเครื่องจ่ายแรงดันแบบปรับค่าได้

6.3 ทดสอบกับอุปกรณ์วัดอุณหภูมิเทอร์มิสเตอร์

1. ต่อวงจรเพื่อวัดค่าสัญญาณเอาท์พุทที่เป็นแรงดันอนาลอก 1 ถึง 5 Vdc
2. เชื่อมต่อชุดไมโครคอนโทรลเลอร์กับคอมพิวเตอร์
3. อุปกรณ์ตรวจวัดอุณหภูมิเทอร์มิสเตอร์

   Range 20-60 องศา

   PV เริ่มจากอุณหภูมิห้อง 22 องศา
4. ทำการ set ค่าต่างๆ ในโปรแกรม
5. บันทึกค่าข้อมูลโดยจับเวลาตั้งให้อ่านค่า ทุกๆ 1 นาทีเป็นเวลา 18 นาที
6. เก็บบันทึกค่า
7. เลือกดูช่องสัญญาณที่ต้องการ
8. ทดลองบันทึกค่าลงตาราง



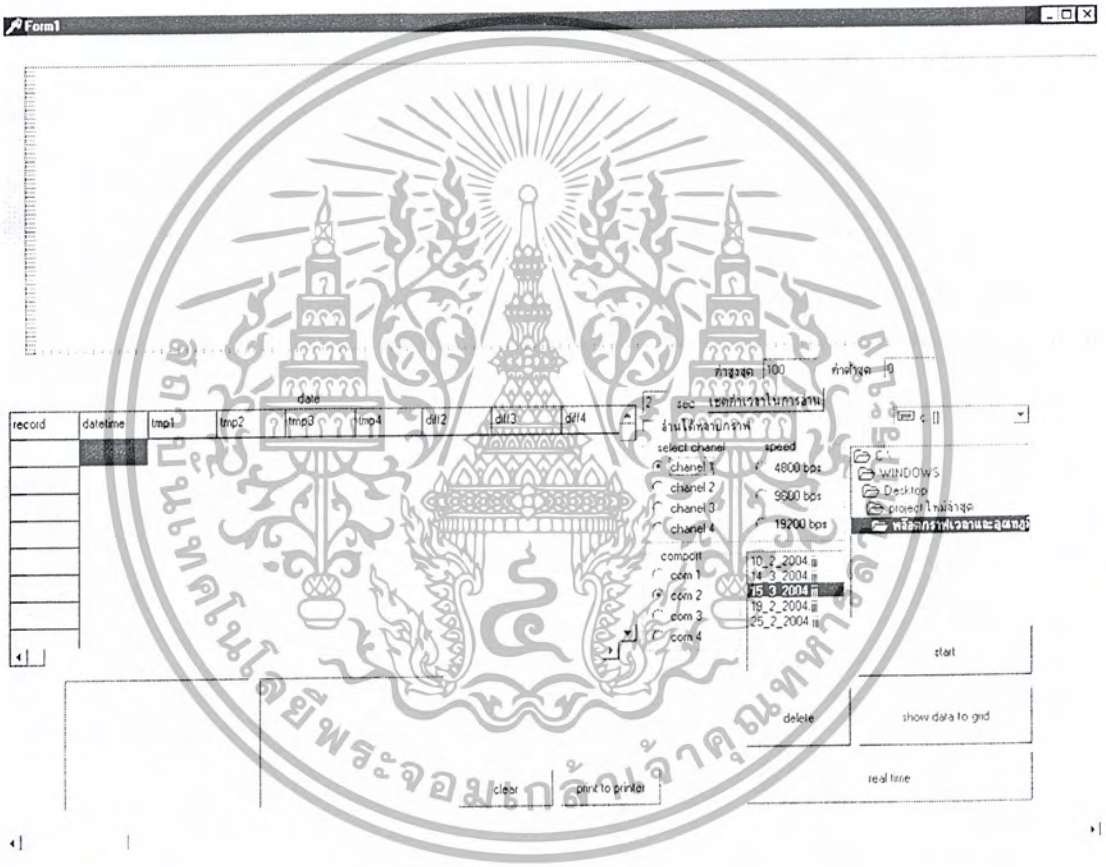
ภาพที่ 6.5 ชุดไมโครคอนโทรลเลอร์

ตารางที่ 6.1 ค่าของอุณหภูมิที่วัดได้ตามเวลาต่าง ๆ

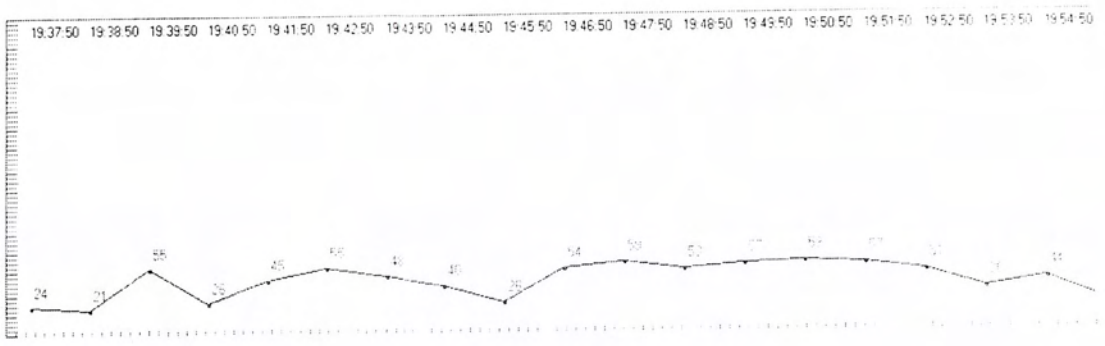
Datetime	tmp1	tmp2	tmp3	Tmp4
19:37:50	24	55	56	57
19:38:50	21	25	29	54
19:39:50	55	54	56	34
19:40:50	26	30	45	45
19:41:50	45	26	39	46
19:42:50	55	26	27	45
19:43:50	48	56	34	39
19:44:50	40	48	42	40
19:45:50	26	25	29	28
19:46:50	54	56	32	41
19:47:50	59	56	24	39
19:48:50	53	32	54	21

ตารางที่ 6.1 (ต่อ)

19:49:50	57	41	40	30
19:50:50	59	40	45	46
19:51:50	57	28	60	35
19:52:50	51	53	59	27
19:53:50	36	50	54	55
19:54:50	44	56	33	21

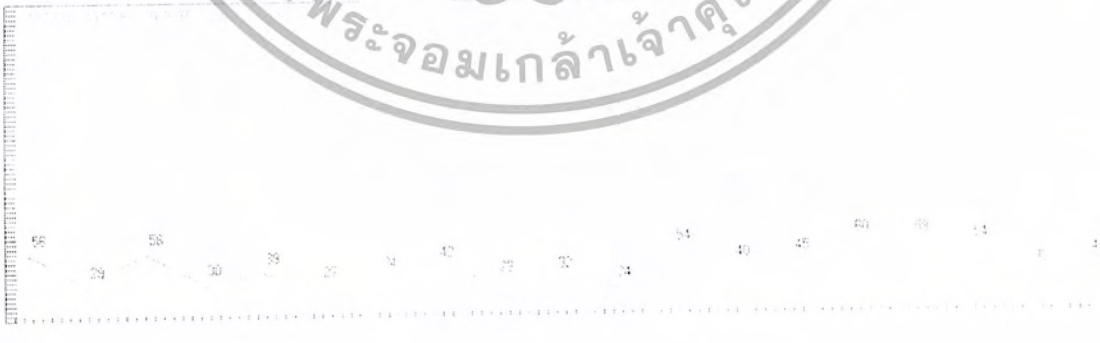


ภาพที่ 6.6 โปรแกรมที่ใช้วัดอุณหภูมิโดยจะพล็อตค่าเป็นกราฟของแต่ละ Channel

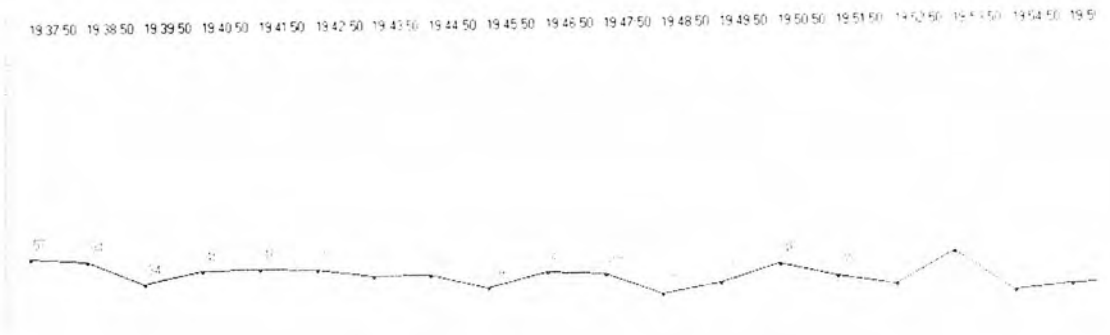


ภาพที่ 6.7 ค่าของอุณหภูมิที่ Channel 1

ภาพที่ 6.8 ค่าของอุณหภูมิที่ Channel 2



ภาพที่ 6.9 ค่าของอุณหภูมิที่ Channel 3



ภาพที่ 6.10 ค่าของอุณหภูมิที่ Channel 4

ภาพที่ 6.11 ค่าของอุณหภูมิทั้ง 4 Channel

6.4 สรุปผลการทดลอง

ในการทดสอบการส่งค่าระหว่างฮาร์ดแวร์กับคอมพิวเตอร์ จะเห็นได้ว่าหลังจากที่เราป้อนโปรแกรมให้ไมโครคอลโทรลเลอร์ และทำการตั้งโปรโตคอลของฮาร์ดแวร์กับเครื่องคอมพิวเตอร์ให้ตรงกันแล้ว เมื่อทำการเปลี่ยนค่าอินพุทที่เข้ามาจะทำให้มีการเปลี่ยนแปลงของข้อมูลในแต่ละชุดที่ป้อนกับคอมพิวเตอร์เพื่อให้คอมพิวเตอร์แสดงค่าออกมาทางจอภาพ

ส่วนการทดลองในส่วนที่สอง เป็นการต่อเครื่องบันทึกสัญญาณเข้ากับเครื่องจ่ายไฟแบบสามารถปรับค่าได้ โดยเริ่มจากต่อเครื่องจ่ายไฟ 4 ตัวกับตัวคอนเวอร์เตอร์โดยตั้งค่าให้ระดับแรงดันแต่ละตัวนั้นต่างกันสังเกตกราฟที่ได้ดังภาพที่ 6.3 จะเป็นลักษณะกราฟเส้นตรง จากนั้นเราทำการปรับค่าแรงดันที่ป้อนให้จะเห็นได้ว่ากราฟที่ได้มีการเปลี่ยนแปลงตามการปรับค่า ดังภาพที่ 6.4

ในการทดลองที่ 6.3 ทดลองทำการวัดค่าในกระบวนการ (PV) โดยใช้เทอร์มิสเตอร์กับเครื่องต้มน้ำซึ่งปรับอุณหภูมิได้ โดยเราปรับให้ค่าอุณหภูมิ ขึ้น หรือ ลง ตามความต้องการที่อุณหภูมิห้องประมาณ 22 องศาเซลเซียส สังเกตค่าทางแกน X จะมีการวิ่งไปทางเวลาโดยจะสัมพันธ์กับแกนของอุณหภูมิ (แกน Y) โดยจะเพิ่มและวิ่งไปเรื่อยๆ หลังจากนั้นก็ปรับอุณหภูมิขึ้น บันทึกค่าที่ได้เป็นไปตามตารางที่ 6.1 ซึ่งกราฟที่อ่านได้จะตรงกับอุณหภูมิตั้งไว้บนเครื่องต้มน้ำ

บทที่ 7
บทสรุปและแนวทางการพัฒนา

7.1 สรุปผลการทำงาน

จากการทดลองสามารถสรุปการทำงานได้ดังนี้

1.วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล (A/D Converter) สามารถแปลงค่าได้ถูกต้องที่ความละเอียด 10 บิท

2. สามารถรับข้อมูลอนาลอกอินพุท แรงดัน 1-5 Vdc พร้อมกันทั้ง 4 ช่องสัญญาณได้

3. สามารถตั้งค่าฐานเวลาที่จะอ่านค่าจากไมโครคอนโทรลเลอร์ได้

4. สามารถแสดงผลในรูปแบบของกราฟและตารางค่าของข้อมูลได้

5. สามารถย้อนข้อมูลกลับมาดูได้และแสดงเป็นกราฟ

6. สามารถนำรูปกราฟที่ได้พิมพ์ออกทางเครื่องพิมพ์

7.2 ปัญหา

1. รูปแบบของการ รับ – ส่ง ข้อมูลของคอมพิวเตอร์ กับคอนโทรลเลอร์จะรับได้เพียง 4 ช่องสัญญาณเท่านั้น

2. รูปกราฟยังไม่สวยงาม

3. สามารถวัดได้เพียงอุณหภูมิ และรับอินพุทในลักษณะเป็นแรงดันเท่านั้น

7.3 แนวทางการพัฒนา

1. สามารถตั้งค่าเวลาให้โปรแกรมเริ่มทำงานและหยุดการทำงานแบบอัตโนมัติได้

2. สามารถใช้งานได้ครอบคลุมทั้งอุณหภูมิในระดับต่างๆ ได้

3. สามารถรับอินพุทที่นอกเหนือจากแรงดันได้

4. นำคอมโพเนนท์ของกราฟที่สวยงามเข้ามาใช้

5. ดัดแปลงบางส่วนของโปรแกรมเพื่อความยืดหยุ่นในการใช้งาน

บรรณานุกรม

1. ชัยวัฒน์ ลิ้มพรจิตวิไล. "เรียนรู้การเชื่อมต่อไอซีกับอุปกรณ์ภายนอกผ่านพอร์ทอนุกรม" .บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด. 2542.

2. ธีรวัฒน์ ประกอบผล. "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์", สมาคมส่งเสริมเทคโนโลยี (ไทย - ญี่ปุ่น). พิมพ์ครั้งที่3. 2542.

3. ประยงค์ อู่ประสิทธิ์วงศ์. "คู่มือการใช้งานโปรแกรม Borland Delphi 5 ฉบับ Object Pascal", บริษัท ซัคเซส มีเดีย จำกัด. 2543.

4. กนก กุสุมาลย์นุกูล. "คู่มือการใช้งานโปรแกรม Borland Delphi 5 ฉบับ การใช้งานจริง", บริษัท ซัคเซส มีเดีย จำกัด. 2543.

5. สัจจะ จรัสรุ่งรวีวร. "เริ่มต้นอย่างมืออาชีพด้วย Delphi 7 ฉบับสมบูรณ์", พิมพ์ครั้งที่1. สำนักพิมพ์อินโฟเพรส. 2546.

7. ธีรบูลย์ หล่อวิเชียรรุ่ง. "เรียนรู้และปฏิบัติการระบบคาต้าแอกควิซิชั่นอย่างง่าย", บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด. 2541.

7. "PIC16F87X Data Sheet", DS30292C. Microchip Technology Incorporated. 2001.

8. "The I^2C – Bus Specification", Phillips Semiconductors. 2000.

ภาคผนวก

ภาคผนวก ก

การแปลงสัญญาณอนาลอกดิจิตอลแบบซักแซสซีฟแอปพร็อกซิเมชั่น

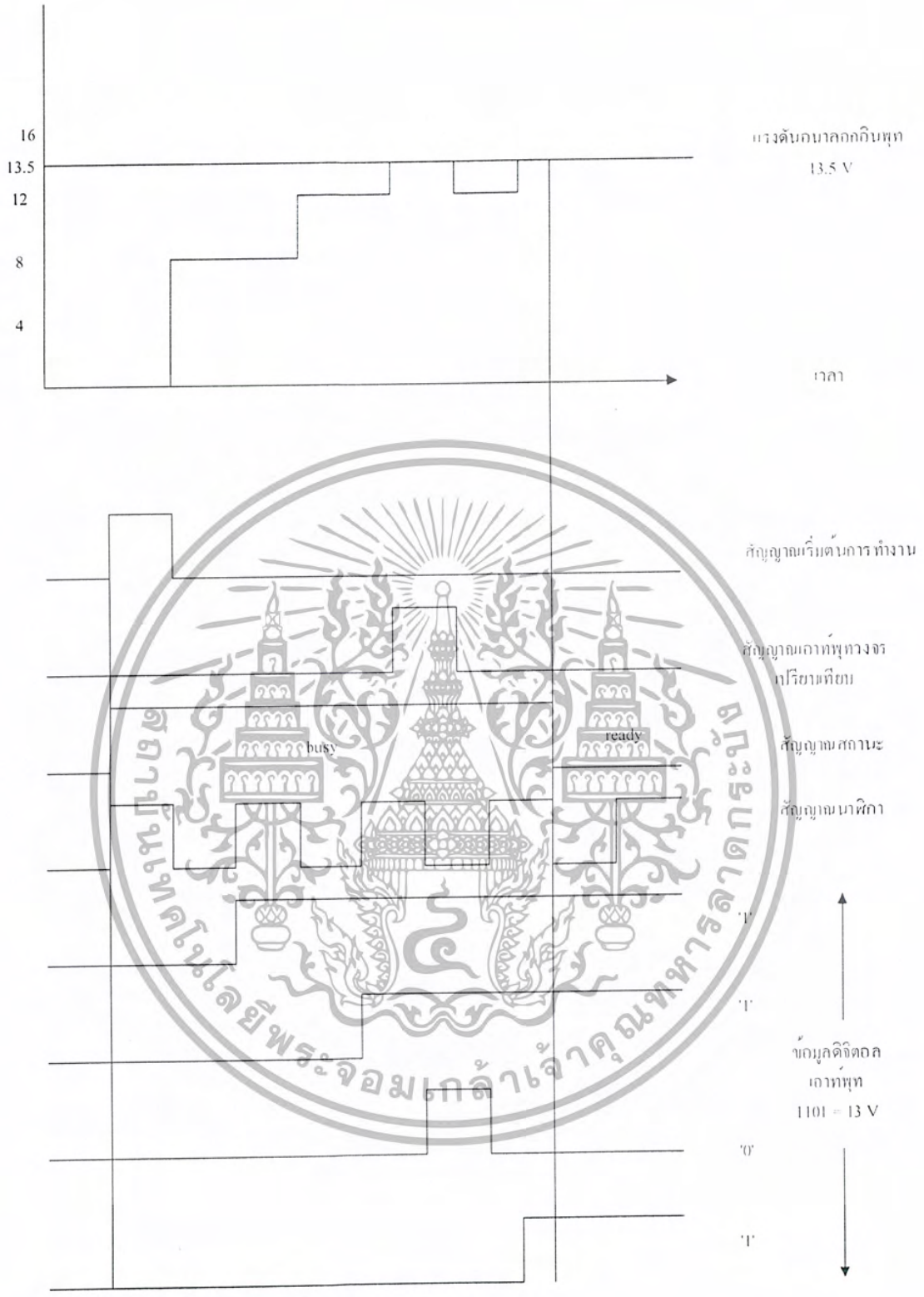
(Successive Approximation ADC)

การแปลงสัญญาณอนาลอกเป็นดิจิตอล (A/D) ที่ได้รับความนิยมสูงและมีประสิทธิภาพดี คือ การแปลงแบบซักแซสซีฟแอปพร็อกซิเมชั่น ไอซี ADC ต่อไปเราจะต้องทำการเข้าใจพื้นฐานการทำงานของวงจร ADC แบบนี้ก่อน

ถ้าจะแปลเป็นภาษาไทยอาจเรียกกระบวนการ ADC แบบซักแซสซีฟแอปพร็อกซิเมชั่นนี้ว่าเป็นการแปลงแบบประมาณค่าใกล้เคียง ไดอะแกรมแสดงการทำงานแสดงในภาพที่ 1 ส่วนสำคัญหลัก คือ วงจรเปรียบเทียบแรงดันวงจรแปลงสัญญาณดิจิตอลเป็นอะนาลอกหรือ DAC สัญญาณนาฬิกา และส่วนควบคุมลอจิก

ภาพที่ 1 ไดอะแกรมแสดงการทำงานของวงจร ADC แบบซักแซสซีฟแอปพร็อกซิเมชั่น

วงจร ADC แบบซักแซสซีฟแอปพร็อกซิเมชั่น นี้จะใช้รีจิสเตอร์เลขฐานสองหรือไบนารีรีจิสเตอร์ในการส่งข้อมูลดิจิตอลของวงจร DAC ภายในแต่ละบิตของรีจิสเตอร์จะเซตและรีเซตโดยการควบคุมจากวงจรควบคุมต่อไปนี้จะอธิบายถึงการทำงานของ ADC แบบนี้ไปทีละขั้นตอนขอให้พิจารณาไดอะแกรมแสดงเวลา ดังภาพที่ 2 ร่วมด้วย



ภาพที่ 2 ไดอะแกรมเวลาแสดงการทำงานของ ADC แบบซักแซสซีฟแอปพร็อกซิเมชั่น

กำหนดให้แรงดันอนาลอกอินพุต มีค่า 13.5 V

1. ส่งสัญญาณเริ่มต้นการทำงานมายังซักแซสซีฟแอปพรอกซิเมชั่น
2. ขณะนี้สถานะของรีจิสเตอร์จะไม่ว่างสัญญาณนาฬิกาลูกแรกถูกส่งเข้ามาเพื่อกำหนดให้ค่าของรีจิสเตอร์เท่ากับ 0000
3. เอาท์พุทของ DAC จะเป็น 0 V ส่งไปในวงจรเปรียบเทียบเพื่อเปรียบเทียบกับ Vin ในขณะนี้จะได้เอาท์พุทเท่ากับ -5V กำหนดเป็น ลอกจิก "0"
4. เมื่อสัญญาณนาฬิกาลูกต่อไปเข้ามาจะทำการเซตบิต MSB จะรีจิสเตอร์ เป็น"1"
5. ในกรณีนี้เป็น ADC ขนาด 4 บิตดังนั้นการที่บิต MSB เซตจะทำให้วงจร DAC แปลงค่าเป็นแรงดัน8V นำไปเปรียบเทียบที่วงจรเปรียบเทียบแรงดันแต่ก็ยังน้อยกว่า Vin ดังนั้นเอาท์พุทของวงจรเปรียบเทียบยังคงเป็น "0" ทำให้รีจิสเตอร์ยังคงค่าบิต MSB เป็น "1" ต่อไป
6. ต่อมาบิต B2 จะเซตซึ่งจะมีค่าเท่ากับ 4V นำ ไปรวมกับค่าของบิต MSB ที่มีอยู่ 8V เช่น 12 V นำไปเปรียบเทียบกับ Vin ก็ยังน้อยกว่ารีจิสเตอร์จึงยังคงค่า B2 ไว้ที่ "1" เช่นกัน
7. ต่อมาบิต B1 เซตทำให้แรงดันเอาท์พุทมา DAC กลายเป็น 8+4+2=14V ซึ่งมากกว่า Vin ทำให้วงจรเปรียบเทียบเกิดการเปลี่ยนสถานะเป็น "1" ซึ่งจะส่งสัญญาณมาควบคุมให้ B1 กลายเป็น "0"
8. เมื่อบิต LSB ถูกเซต จะมีค่าแรงดัน 1 V เข้ามารวมกับค่าของ B3 , B2 และ B1 เป็น 8+4+0+1=13V นำไปเปรียบเทียบกับ Vin ปรากฏว่าน้อยกว่า Vin ทำให้บิต B0 หรือ LSB มีค่าเป็น "1"
9. ขณะนี้ทุกบิตในรีจิสเตอร์ถูกนำมาแปลงค่าเรียบร้อยแล้วทำให้สถานะของรีจิสเตอร์กลับมาเป็นพร้อมทำงาน
10. ข้อมูลดิจิตอลที่ได้จากการ ADC แบบนี้ จะมีค่า 1101 หรือ 13V ซึ่งใกล้เคียงกับVin 13.5 V มากที่สุดถ้าหากรีจิสเตอร์นี้มีจำนวนบิตมากกว่านี้ ความละเอียดของข้อมูลที่แปลงได้จะมีความใกล้เคียงมากขึ้นช่วงเวลาของการแปลงสัญญาณจะเริ่มต้นขึ้น ตั้งแต่สัญญาณนาฬิกาลูกแรกถูกส่งเข้าไปเตรียมระบบไปจนถึงเมื่อสภาวะของรีจิสเตอร์กลับมาเป็น พร้อมทำงานอีกครั้งหนึ่งซึ่งจะต้องใช้จำนวนสัญญาณนาฬิกาเท่ากับ n-1 พลัลซ์ โดย n เท่ากับจำนวนบิตของรีจิสเตอร์

ดังนั้นถ้าหาก ADC แบบซักแซสซีฟแอปพรอกซิเมชั่นขนาด 4 บิต ตามตัวอย่างที่อธิบายมานี้ใช้สัญญาณนาฬิกาความถี่ 50kHz เวลาที่ใช้ทั้งหมดในการแปลงสัญญาณจะคำนวณได้ดังนี้

คำนวณคาบเวลาของสัญญาณนาฬิกา

(1) $Fclk = 50kHz = 50x10^3$

$T = 1/(50x10^3) = 20$ มิลลิวินาที

(2) จำนวนสัญญาณนาฬิกาทั้งหมดที่ใช้ในการแปลงเท่ากับ n+1 . n มีค่าเท่ากับ 4 เนื่องจากมีจำนวน 4 บิตดังนั้นจำนวนสัญญาณ นาฬิกาที่ใช้ทั้งหมดจึงเท่ากับ 4+1 = 5

(3) เวลาทั้งหมดที่ใช้ เท่ากับ 5 X 20 = 100 มิลลิวินาที

จะเห็นว่าวงจร ADC แบบซักแซสซีฟแอปพรอกซิเมชั่นนี้ มีความเร็วในการทำงานที่สูงพอสมควรเหมาะอย่างยิ่งในการนำใช้กับไมโครคอลโทรลเลอร์ขนาดกลางอย่าง MCS-51

ความเที่ยงตรงของวงจร ADC

ความเที่ยงตรงของวงจร ADC เป็นการเปรียบเทียบแรงดันอนาลอกของวงจร ADC กับแรงดันที่ควรจะเกิดขึ้นจริง ยกตัวอย่างที่ข้อมูลดิจิตอลสูงสุดของวงจร ADC ขนาด 8 บิตเมื่อเทียบเป็นแรงดันอนาลอก ควรที่จะมีค่าเท่ากับ 5V แต่จากการคำนวณในตัวอย่างก่อนหน้านี้ได้ค่าแรงดัน 4.9804 V นั้นคือเกิดความผิดพลาดไป 0.0195 V หรือ 19.5 mV แต่การบอกค่าความเที่ยงตรงของวงจร ADC มักระบุเป็นจำนวนที่เทียบกับ VLSB ดังนั้นในวงจร ADC ขนาด 8 บิตที่ยกเป็นตัวอย่างนี้จึงมีค่าความเที่ยงตรงเป็น $\pm$ 1/2LSB

ค่าเวลาในการแปลงสัญญาณ

เป็นค่าของเวลาทั้งหมดที่วงจร ADC แบบวงจรนับแรมป์และแบบซักแซสซีฟแอปพรอกซิเมชั่นใช้ในการแปลงสัญญาณอะนาลอกเป็นดิจิตอลจนเสร็จสิ้น พารามิเตอร์ตัวนี้มักจะปรากฏในคุณสมบัติของไอซีที่ทำงานเป็นวงจร ADC เมื่อไอซีแปลงสัญญาณเสร็จสิ้นลงแล้ว จะส่งสัญญาณที่เรียกว่า EOC ออกมาก

ค่าเวลาในการแปลงสัญญาณของวงจร ADC จะขึ้นอยู่กับจำนวนบิตของวงจร ค่าความถี่ของสัญญาณนาฬิกาที่ใช้ในการแปลงสัญญาณและขนาดของสัญญาณอนาลอกอินพุท

ภาคผนวก ข

PIC16F87X
Data Sheet

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

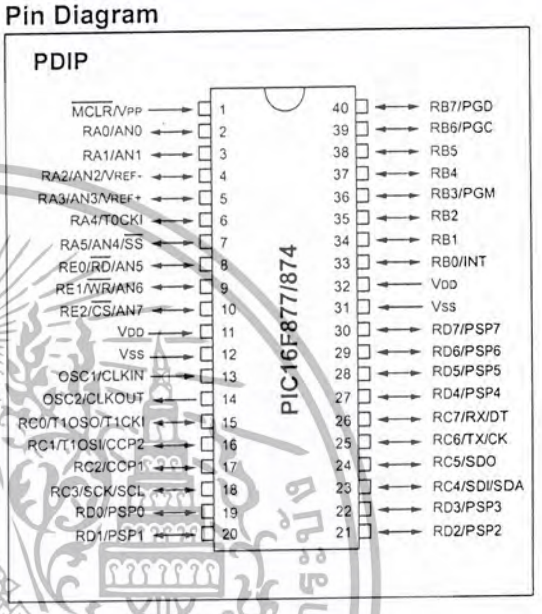
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

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, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

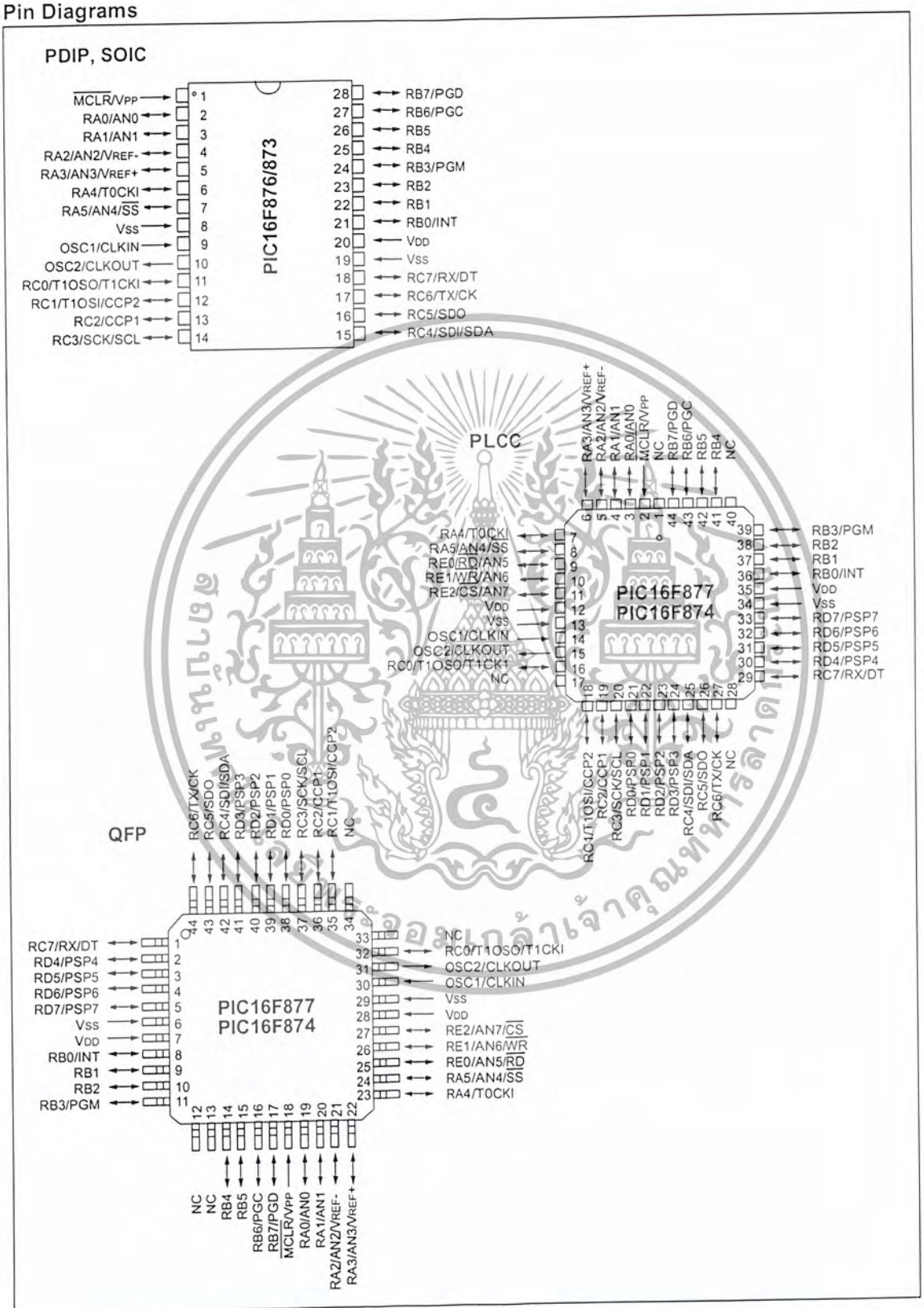
PDIP



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 I²C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

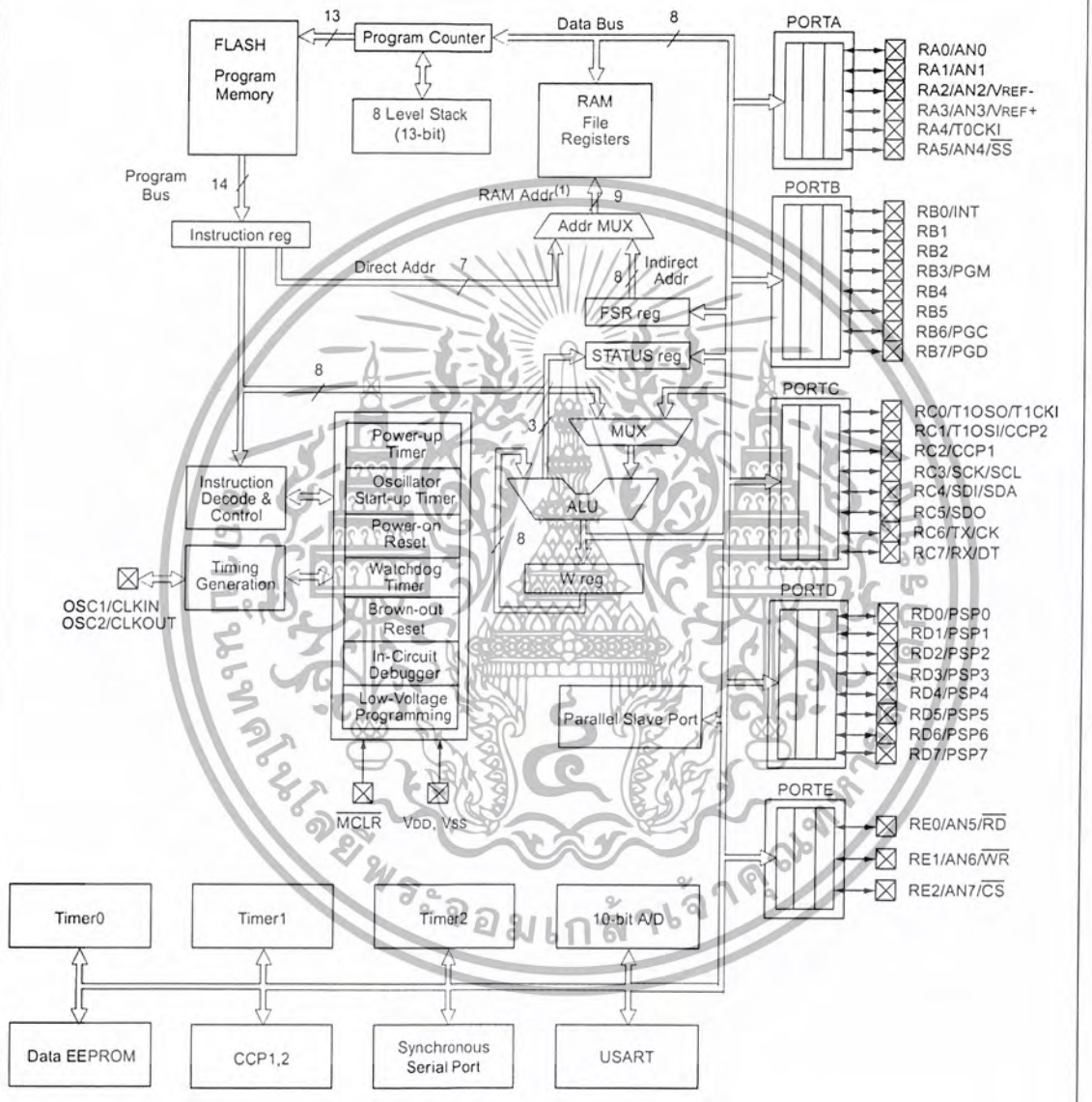
Pin Diagrams



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

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 1-2: 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	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. 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 soft-ware 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.

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
						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.
						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 the PIC16F87X 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 wraparound.

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

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

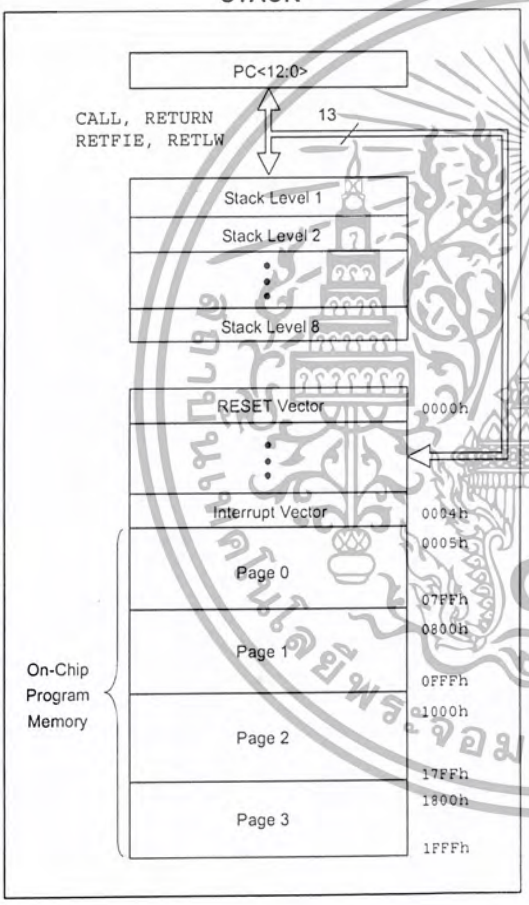
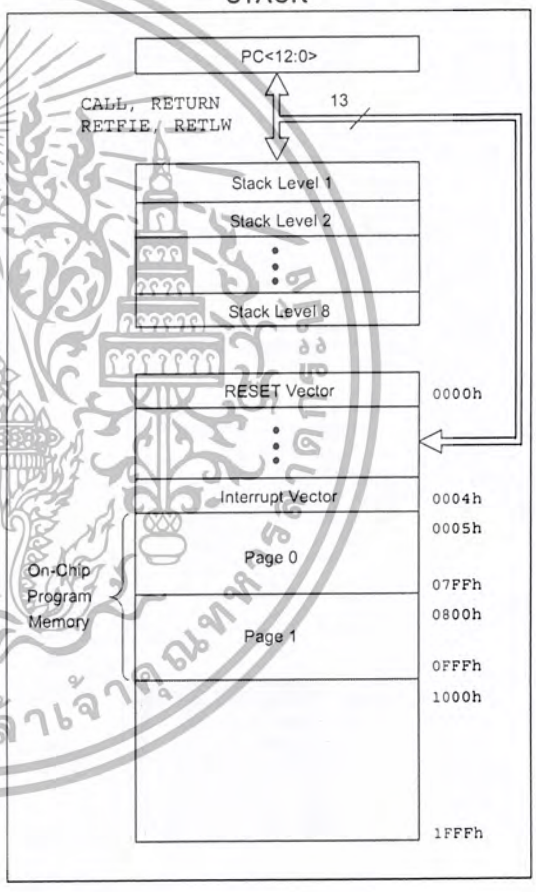


FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK



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

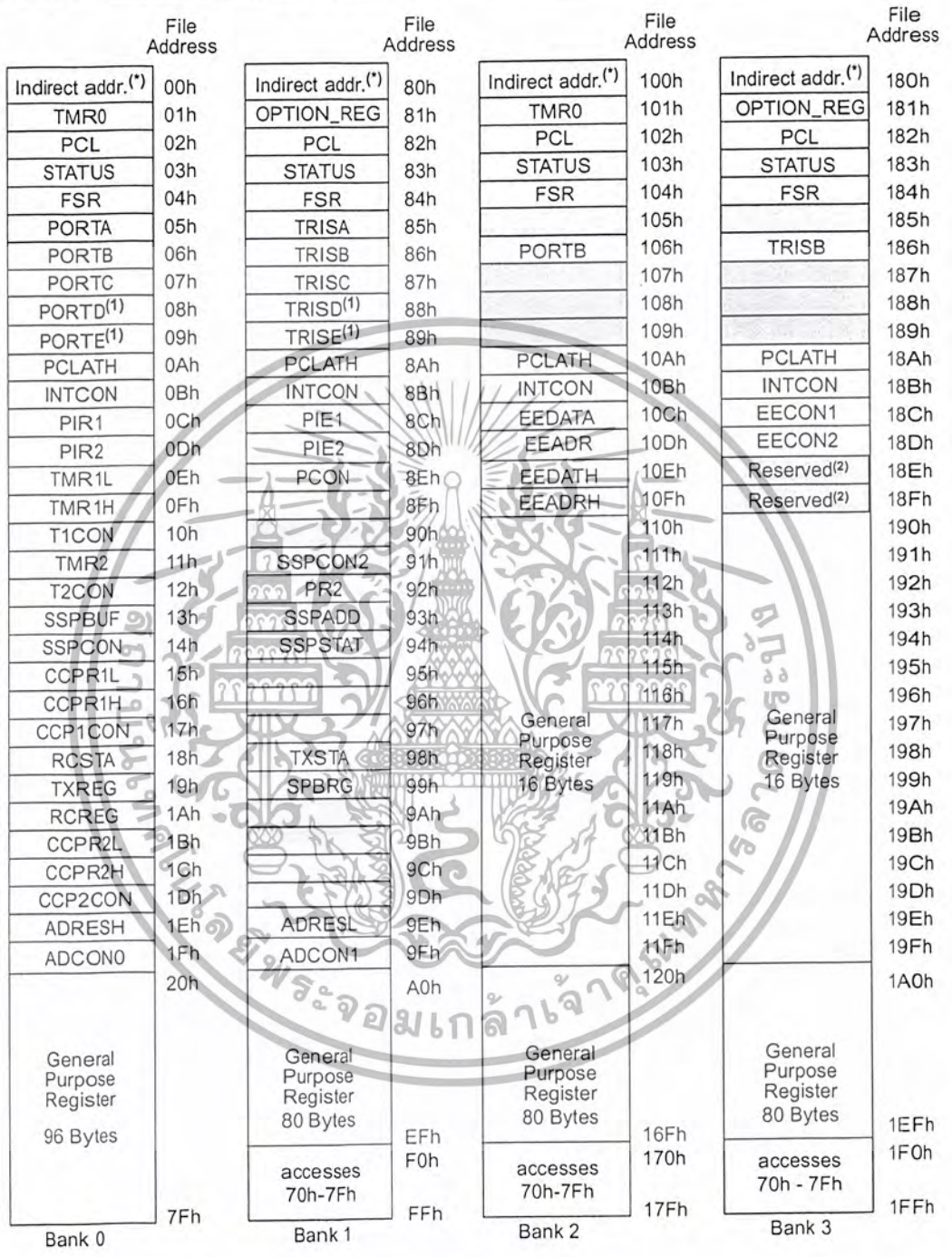
Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some frequently used Special 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



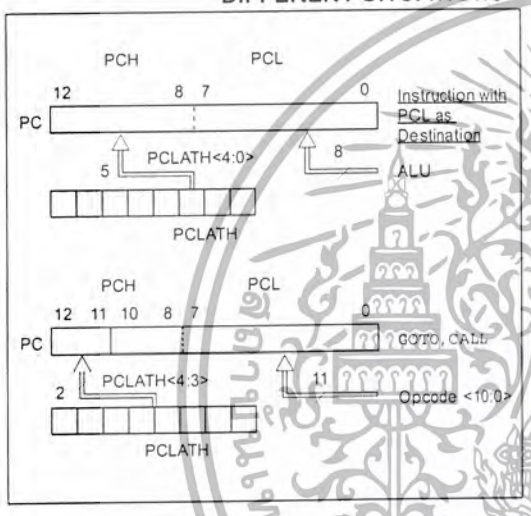
Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

2.3 PCL and PCLATH

The program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any RESET, the upper bits of the PC will be cleared. Figure 2-5 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in the figure shows how the PC is loaded during a CALL or GOTO instruction (PCLATH<4:3> → PCH).

FIGURE 2-5: LOADING OF PC IN DIFFERENT SITUATIONS



2.3.1 COMPUTED GOTO

A computed GOTO is accomplished by adding an offset to the program counter (ADDWF PCL). When doing a table read using a computed GOTO method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block). Refer to the application note, *"Implementing a Table Read"* (AN556).

2.3.2 STACK

The PIC16F87X family has an 8-level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

Note 1: There are no status bits to indicate stack overflow or stack underflow conditions.

2: There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the CALL, RETURN, RETLW and RETFIE instructions, or the vectoring to an interrupt address.

2.4 Program Memory Paging

All PIC16F87X devices are capable of addressing a continuous 8K word block of program memory. The CALL and GOTO instructions provide only 11 bits of address to allow branching within any 2K program memory page. When doing a CALL or GOTO instruction, the upper 2 bits of the address are provided by PCLATH<4:3>. When doing a CALL or GOTO instruction, the user must ensure that the page select bits are programmed so that the desired program memory page is addressed. If a return from a CALL instruction (or interrupt) is executed, the entire 13-bit PC is popped off the stack. Therefore, manipulation of the PCLATH<4:3> bits is not required for the return instructions (which POPs the address from the stack).

Note: The contents of the PCLATH register are unchanged after a RETURN or RETFIE instruction is executed. The user must rewrite the contents of the PCLATH register for any subsequent subroutine calls or GOTO instructions.

Example 2-1 shows the calling of a subroutine in page 1 of the program memory. This example assumes that PCLATH is saved and restored by the Interrupt Service Routine (if interrupts are used).

EXAMPLE 2-1: CALL OF A SUBROUTINE IN PAGE 1 FROM PAGE 0

```
        ORG 0x500
        BCF PCLATH,4
        BSF PCLATH,3    ;Select page 1
                        ;(800h-FFFh)
        CALL SUB1_P1    ;Call subroutine in
        :               ;page 1 (800h-FFFh)
        :
        ORG 0x900       ;page 1 (800h-FFFh)
SUB1_P1
        :               ;called subroutine
                        ;page 1 (800h-FFFh)
        :
        RETURN          ;return to
                        ;Call subroutine
                        ;in page 0
                        ;(000h-7FFh)
```

2.5 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

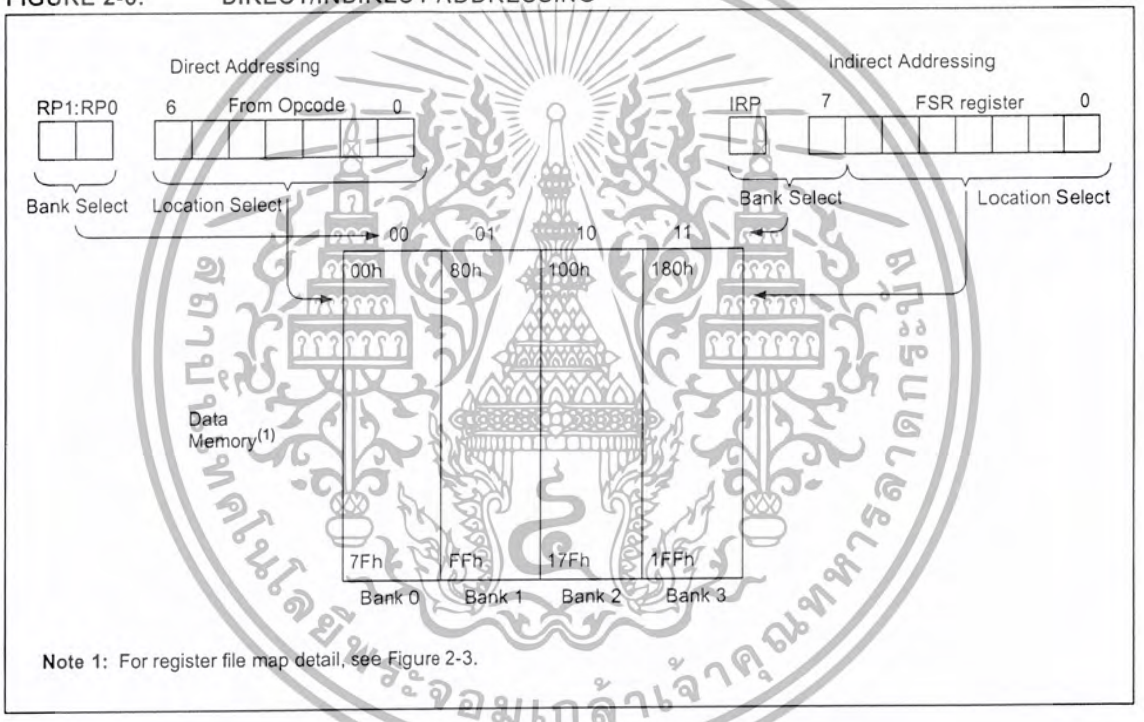
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-6.

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

EXAMPLE 2-2: INDIRECT ADDRESSING

```
        MOVLW  0x20     ;initialize pointer
        MOVWF  FSR      ;to RAM
NEXT    CLRF   INDF     ;clear INDF register
        INCF   FSR,F    ;inc pointer
        BTFSS  FSR,4    ;all done?
        GOTO   NEXT     ;no clear next
CONTINUE
        :               ;yes continue
```

FIGURE 2-6: DIRECT/INDIRECT ADDRESSING



Note 1: For register file map detail, see Figure 2-3.

11.0 ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices.

The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, Vss, RA2, or RA3.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register, shown in Register 11-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 11-2, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O.

Additional information on using the A/D module can be found in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 11-1: ADCON0 REGISTER (ADDRESS: 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/DONE:** A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 11-2: ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

Note 1: These channels are not available on PIC16F873/876 devices.
2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

The ADRESH:ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D result register pair, the GO/$\overline{\text{DONE}}$ bit (ADCON0<2>) is cleared and the A/D interrupt flag bit ADIF is set. The block diagram of the A/D module is shown in Figure 11-1.

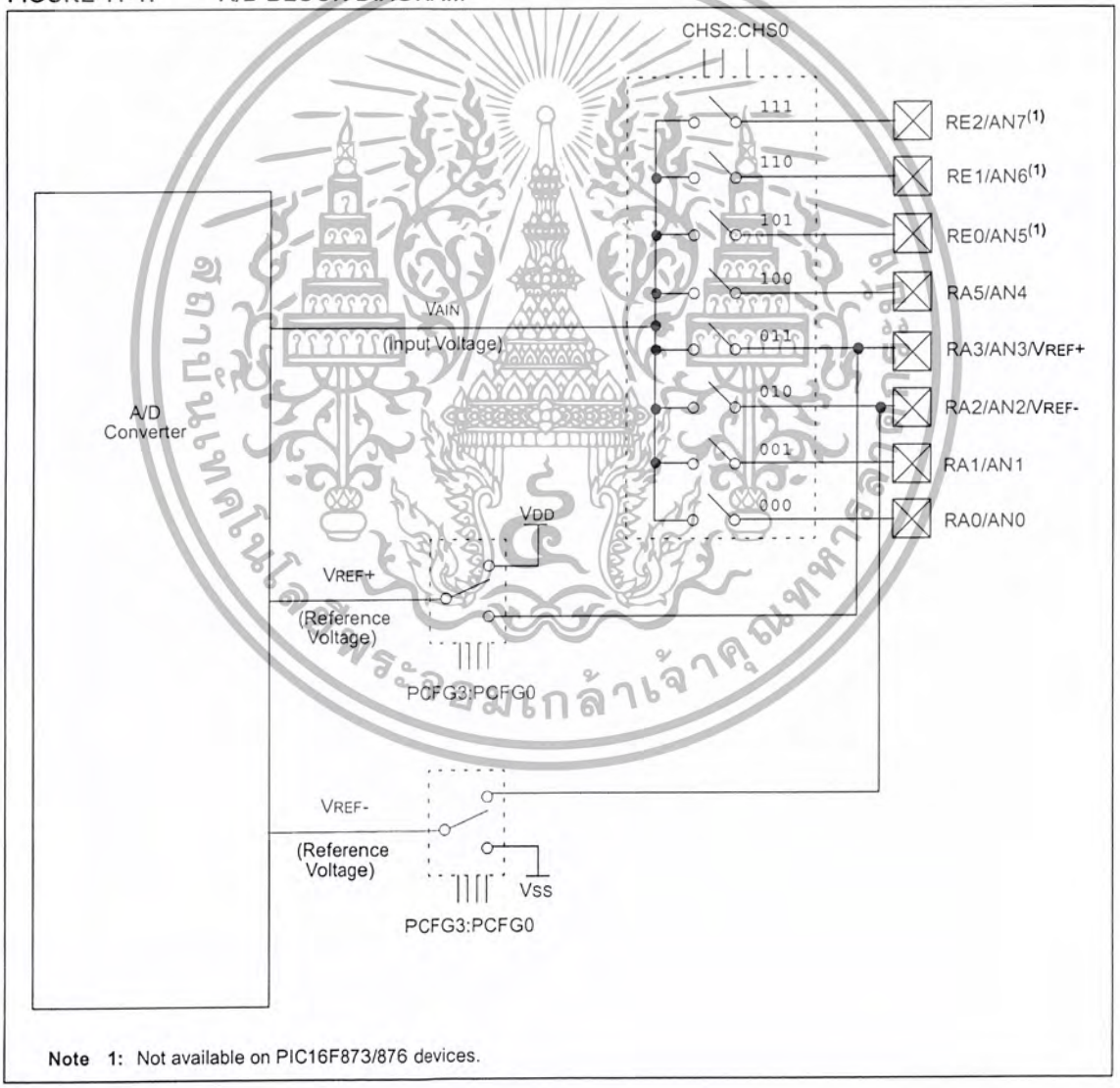
After the A/D module has been configured as desired, the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs.

To determine sample time, see Section 11.1. After this acquisition time has elapsed, the A/D conversion can be started.

These steps should be followed for doing an A/D Conversion:

1. Configure the A/D module:
   - Configure analog pins/voltage reference and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D conversion clock (ADCON0)
   - Turn on A/D module (ADCON0)
2. Configure A/D interrupt (if desired):
   - Clear ADIF bit
   - Set ADIE bit
   - Set PEIE bit
   - Set GIE bit
3. Wait the required acquisition time.
4. Start conversion:
   - Set GO/$\overline{\text{DONE}}$ bit (ADCON0)
5. Wait for A/D conversion to complete, by either:
   - Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared (with interrupts enabled); OR
   - Waiting for the A/D interrupt
6. Read A/D result register pair (ADRESH:ADRESL), clear bit ADIF if required.
7. For the next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as TAD. A minimum wait of 2TAD is required before the next acquisition starts.

FIGURE 11-1: A/D BLOCK DIAGRAM



Note 1: Not available on PIC16F873/876 devices.

11.1 A/D Acquisition Requirements

For the A/D converter to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the input channel voltage level. The analog input model is shown in Figure 11-2. The source impedance (RS) and the internal sampling switch (RSS) impedance directly affect the time required to charge the capacitor CHOLD. The sampling switch (RSS) impedance varies over the device voltage (VDD), see Figure 11-2. **The maximum recommended impedance for analog sources is 10 kΩ.** As the impedance is decreased, the acquisition time may be decreased.

After the analog input channel is selected (changed), this acquisition must be done before the conversion can be started.

To calculate the minimum acquisition time, Equation 11-1 may be used. This equation assumes that 1/2 LSb error is used (1024 steps for the A/D). The 1/2 LSb error is the maximum error allowed for the A/D to meet its specified resolution.

To calculate the minimum acquisition time, TACQ, see the PICmicro™ Mid-Range Reference Manual (DS33023).

EQUATION 11-1: ACQUISITION TIME

TACQ = Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient

= TAMP + TC + TCOFF
= 2µs + TC + [(Temperature -25°C)(0.05µs/°C)]
TC = CHOLD (RIC + RSS + RS) ln(1/2047)
= - 120pF (1kΩ + 7kΩ + 10kΩ) ln(0.0004885)
= 16.47µs
TACQ = 2µs + 16.47µs + [(50°C -25°C)(0.05µs/°C)
= 19.72µs

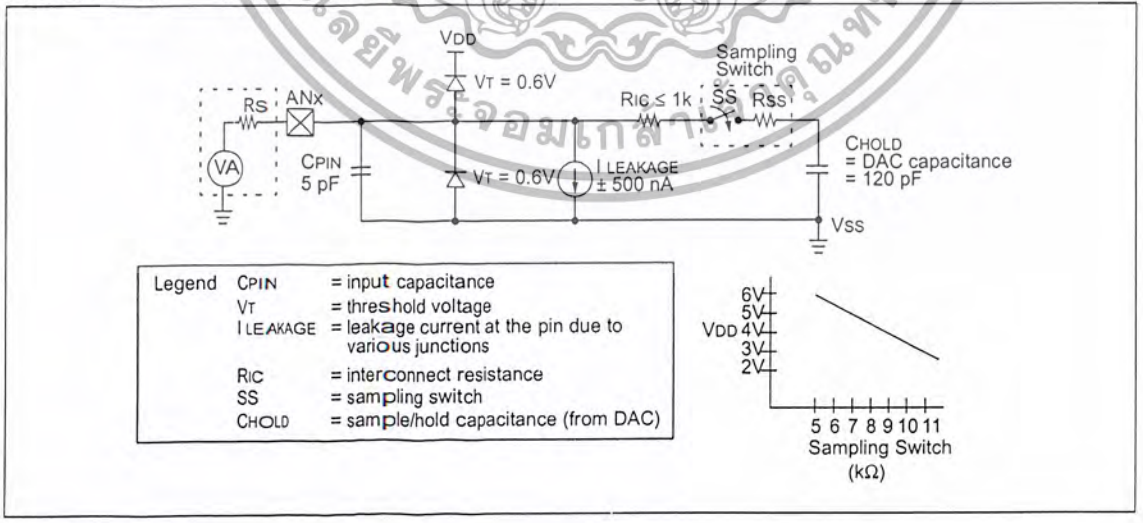
Note 1: The reference voltage (VREF) has no effect on the equation, since it cancels itself out.

2: The charge holding capacitor (CHOLD) is not discharged after each conversion.

3: The maximum recommended impedance for analog sources is 10 kΩ. This is required to meet the pin leakage specification.

4: After a conversion has completed, a 2.0TAD delay must complete before acquisition can begin again. During this time, the holding capacitor is not connected to the selected A/D input channel.

FIGURE 11-2: ANALOG INPUT MODEL



11.2 Selecting the A/D Conversion Clock

The A/D conversion time per bit is defined as TAD. The A/D conversion requires a minimum 12TAD per 10-bit conversion. The source of the A/D conversion clock is software selected. The four possible options for TAD are:

- 2Tosc
- 8Tosc
- 32Tosc
- Internal A/D module RC oscillator (2-6 µs)

For correct A/D conversions, the A/D conversion clock (TAD) must be selected to ensure a minimum TAD time of 1.6 µs.

Table 11-1 shows the resultant TAD times derived from the device operating frequencies and the A/D clock source selected.

TABLE 11-1: TAD vs. MAXIMUM DEVICE OPERATING FREQUENCIES (STANDARD DEVICES (C))

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS1:ADCS0	Max.
2Tosc	00	1.25 MHz
8Tosc	01	5 MHz
32Tosc	10	20 MHz
RC(1, 2, 3)	11	(Note 1)

Note 1: The RC source has a typical TAD time of 4 µs, but can vary between 2-6 µs.
2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for SLEEP operation.
3: For extended voltage devices (LC), please refer to the Electrical Characteristics (Sections 15.1 and 15.2).

11.3 Configuring Analog Port Pins

The ADCON1 and TRIS registers control the operation of the A/D port pins. The port pins that are desired as analog inputs must have their corresponding TRIS bits set (input). If the TRIS bit is cleared (output), the digital output level (VOH or VOL) will be converted.

The A/D operation is independent of the state of the CHS2:CHS0 bits and the TRIS bits.

Note 1: When reading the port register, any pin configured as an analog input channel will read as cleared (a low level). Pins configured as digital inputs will convert an analog input. Analog levels on a digitally configured input will not affect the conversion accuracy.

2: Analog levels on any pin that is defined as a digital input (including the AN7:AN0 pins), may cause the input buffer to consume current that is out of the device specifications.

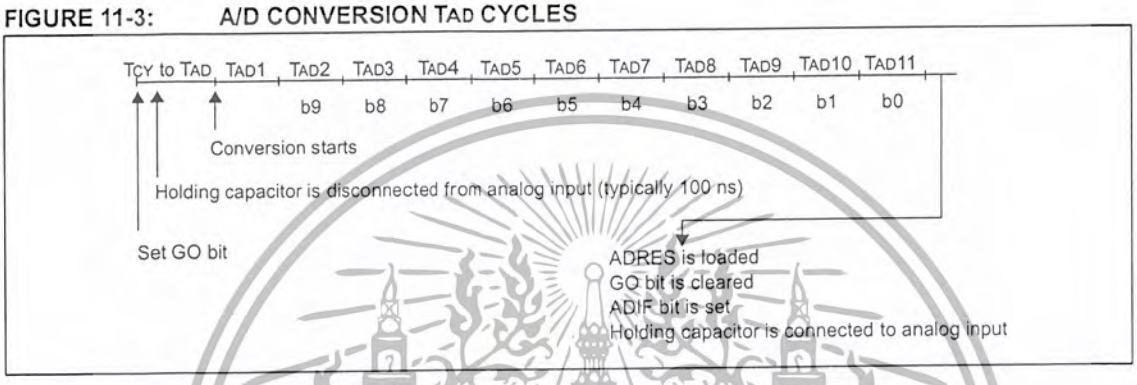
11.4 A/D Conversions

Clearing the GO/$\overline{\text{DONE}}$ bit during a conversion will abort the current conversion. The A/D result register pair will NOT be updated with the partially completed A/D conversion sample. That is, the ADRESH:ADRESL registers will continue to contain the value of the last completed conversion (or the last value written to the ADRESH:ADRESL registers). After the A/D conversion is aborted, a 2TAD wait is required before the next acquisition is started. After this 2TAD wait, acquisition on the selected channel is automatically started. The GO/$\overline{\text{DONE}}$ bit can then be set to start the conversion.

In Figure 11-3, after the GO bit is set, the first time segment has a minimum of TCY and a maximum of TAD.

Note:	The GO/$\overline{\text{DONE}}$ bit should **NOT** be set in the same instruction that turns on the A/D.

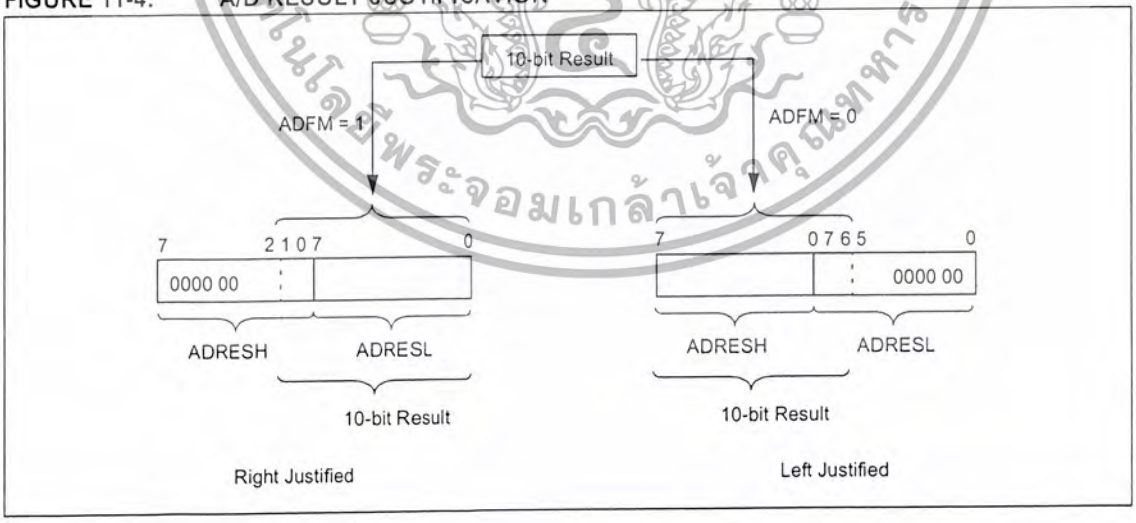
FIGURE 11-3: A/D CONVERSION TAD CYCLES



11.4.1 A/D RESULT REGISTERS

The ADRESH:ADRESL register pair is the location where the 10-bit A/D result is loaded at the completion of the A/D conversion. This register pair is 16-bits wide. The A/D module gives the flexibility to left or right justify the 10-bit result in the 16-bit result register. The A/D Format Select bit (ADFM) controls this justification. Figure 11-4 shows the operation of the A/D result justification. The extra bits are loaded with '0's'. When an A/D result will not overwrite these locations (A/D disable), these registers may be used as two general purpose 8-bit registers.

FIGURE 11-4: A/D RESULT JUSTIFICATION



11.5 A/D Operation During SLEEP

The A/D module can operate during SLEEP mode. This requires that the A/D clock source be set to RC (ADCS1:ADCS0 = 11). When the RC clock source is selected, the A/D module waits one instruction cycle before starting the conversion. This allows the SLEEP instruction to be executed, which eliminates all digital switching noise from the conversion. When the conversion is completed, the GO/DONE bit will be cleared and the result loaded into the ADRES register. If the A/D interrupt is enabled, the device will wake-up from SLEEP. If the A/D interrupt is not enabled, the A/D module will then be turned off, although the ADON bit will remain set.

When the A/D clock source is another clock option (not RC), a SLEEP instruction will cause the present conversion to be aborted and the A/D module to be turned off, though the ADON bit will remain set.

Turning off the A/D places the A/D module in its lowest current consumption state.

Note:	For the A/D module to operate in SLEEP, the A/D clock source must be set to RC (ADCS1:ADCS0 = 11). To allow the conversion to occur during SLEEP, ensure the SLEEP instruction immediately follows the instruction that sets the GO/DONE bit.

11.6 Effects of a RESET

A device RESET forces all registers to their RESET state. This forces the A/D module to be turned off, and any conversion is aborted. All A/D input pins are configured as analog inputs.

The value that is in the ADRESH:ADRESL registers is not modified for a Power-on Reset. The ADRESH:ADRESL registers will contain unknown data after a Power-on Reset.

TABLE 11-2: REGISTERS/BITS ASSOCIATED WITH A/D

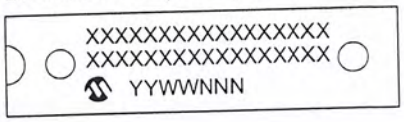
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on MCLR, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h(1)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h(1)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used for A/D conversion.
Note 1: These registers/bits are not available on the 28-pin devices.

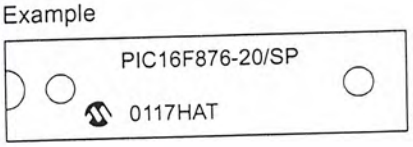
17.0 PACKAGING INFORMATION

17.1 Package Marking Information

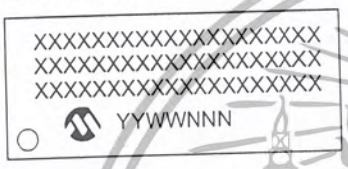
28-Lead PDIP (Skinny DIP)



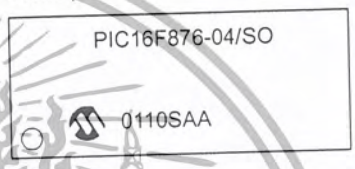
Example



28-Lead SOIC



Example



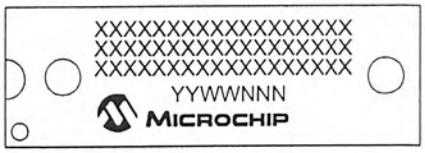
Legend:	XX...X	Customer specific information*
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week '01')
	NNN	Alphanumeric traceability code

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.

* Standard OTP marking consists of Microchip part number, year code, week code, facility code, mask rev#, and assembly code. For OTP marking beyond this, certain price adders apply. Please check with your Microchip Sales Office. For QTP devices, any special marking adders are included in QTP price.

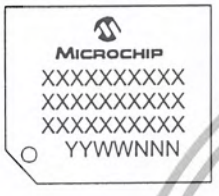
Package Marking Information (Cont'd)

40-Lead PDIP

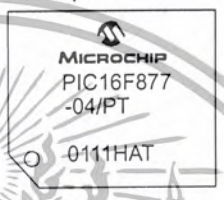


Example

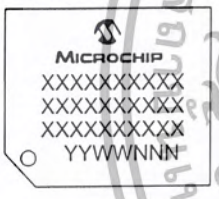
44-Lead TQFP



Example

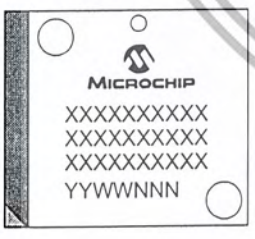


44-Lead MQFP

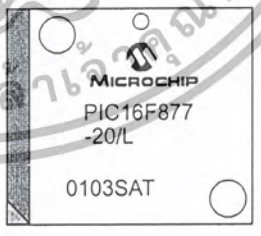


Example

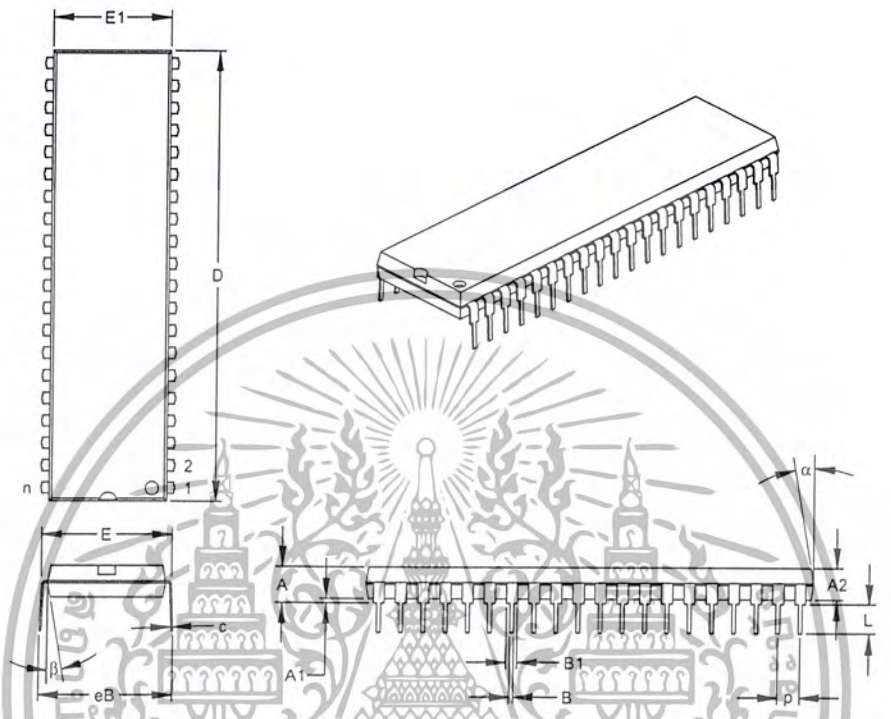
44-Lead PLCC



Example



40-Lead Plastic Dual In-line (P) – 600 mil (PDIP)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		40			40	
Pitch	P		.100			2.54	
Top to Seating Plane	A	.160	.175	.190	4.06	4.45	4.83
Molded Package Thickness	A2	.140	.150	.160	3.56	3.81	4.06
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.595	.600	.625	15.11	15.24	15.88
Molded Package Width	E1	.530	.545	.560	13.46	13.84	14.22
Overall Length	D	2.045	2.058	2.065	51.94	52.26	52.45
Tip to Seating Plane	L	.120	.130	.135	3.05	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.030	.050	.070	0.76	1.27	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.620	.650	.680	15.75	16.51	17.27
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.
JEDEC Equivalent: MO-011
Drawing No. C04-016

APPENDIX A: REVISION HISTORY

Version	Date	Revision Description
A	1998	This is a new data sheet. However, these devices are similar to the PIC16C7X devices found in the PIC16C7X Data Sheet (DS30390). Data Memory Map for PIC16F873/874, moved ADFM bit from ADCON1<5> to ADCON1<7>.
B	1999	FLASH EEPROM access information.
C	2000	DC characteristics updated. DC performance graphs added.

APPENDIX B: DEVICE DIFFERENCES

The differences between the devices in this data sheet are listed in Table B-1.

TABLE B-1: DEVICE DIFFERENCES

Difference	PIC16F876/873	PIC16F877/874
A/D	5 channels, 10-bits	8 channels, 10-bits
Parallel Slave Port	no	yes
Packages	28-pin PDIP, 28-pin windowed CERDIP, 28-pin SOIC	40-pin PDIP, 44-pin TQFP, 44-pin MQFP, 44-pin PLCC

APPENDIX C: CONVERSION CONSIDERATIONS

Considerations for converting from previous versions of devices to the ones listed in this data sheet are listed in Table C-1.

TABLE C-1: CONVERSION CONSIDERATIONS

Characteristic	PIC16C7X	PIC16F87X
Pins	28/40	28/40
Timers	3	3
Interrupts	11 or 12	13 or 14
Communication	PSP, USART, SSP (SPI, I^2C Slave)	PSP, USART, SSP (SPI, I^2C Master/Slave)
Frequency	20 MHz	20 MHz
Voltage	2.5V - 5.5V	2.0V - 5.5V
A/D	8-bit	10-bit
CCP	2	2
Program Memory	4K, 8K EPROM	4K, 8K FLASH
RAM	192, 368 bytes	192, 368 bytes
EEPROM data	None	128, 256 bytes
Other	—	In-Circuit Debugger, Low Voltage Programming

ON-LINE SUPPORT

Microchip provides on-line support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape or Microsoft Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available by using your favorite Internet browser to attach to:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events

Systems Information and Upgrade Hot Line

The Systems Information and Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-480-792-7302 for the rest of the world.

001024

PIC16F87X PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	X	/XX	XXX
Device	Temperature Range	Package	Pattern

Device	PIC16F87X[1], PIC16F87XT[2]; VDD range 4.0V to 5.5V PIC16LF87X[1], PIC16LF87XT[2]; VDD range 2.0V to 5.5V	
Frequency Range	04 = 4 MHz 10 = 10 MHz 20 = 20 MHz	
Temperature Range	blank = 0°C to +70°C (Commercial) I = -40°C to +85°C (Industrial) E = -40°C to +125°C (Extended)	
Package	PQ = MQFP (Metric PQFP) PT = TQFP (Thin Quad Flatpack) SO = SOIC SP = Skinny plastic DIP P = PDIP L = PLCC	

Examples:

a) PIC16F877 - 20/P 301 = Commercial temp., PDIP package, 4 MHz, normal VDD limits, QTP pattern #301.

b) PIC16LF876 - 04I/SO = Industrial temp., SOIC package, 200 kHz, Extended VDD limits.

c) PIC16F877 - 10E/P = Extended temp., PDIP package, 10MHz, normal VDD limits.

Note 1: F = CMOS FLASH
LF = Low Power CMOS FLASH
2: T = in tape and reel - SOIC, PLCC, MQFP, TQFP packages only.

* JW Devices are UV erasable and can be programmed to any device configuration. JW Devices meet the electrical requirement of each oscillator type.

Sales and Support

Data Sheets

Products supported by a preliminary Data Sheet may have an errata sheet describing minor operational differences and recommended workarounds. To determine if an errata sheet exists for a particular device, please contact one of the following:

1. Your local Microchip sales office
2. The Microchip Corporate Literature Center U.S. FAX: (480) 792-7277
3. The Microchip Worldwide Site (www.microchip.com)

Please specify which device, revision of silicon and Data Sheet (include Literature #) you are using.

New Customer Notification System

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