วาล์วควบคุม

CONTROL VALVE

โดย

นายปรัชญา สกุลวิทยสถาวร

นายปริญญา อะทะ

นายพงศธร ไชยรักษา

นายศุภศักดิ์ ธนะกิจรุ่งเรือง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ปีการศึกษา 2550

ภาควิชาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง วาล์วควบคุม
CONTROL VALVE

ผู้จัดทำ นายปรัชญา สกุลวิทยสถาวร รหัส 47010432
นายปริญญา อะทะ รหัส 47010437
นายพงศธร ไชยรักษา รหัส 47010472
นายศุภศักดิ์ ธนะกิจรุ่งเรือง รหัส 47010787

.....อาจารย์ที่ปรึกษา
(อาจารย์วรรณดี เพชรมณีล้ำค่า)

วาล์วควบคุม

โดย

นายปรัชญา สกุลวิทยสถาวร รหัส 47010432

นายปริญญา อะทะ รหัส 47010437

นายพงศธร ไชยรักษา รหัส 47010472

นายศุภศักดิ์ ธนะกิจรุ่งเรือง รหัส 47010787

อาจารย์ที่ปรึกษา

อาจารย์วรรณดี เพชรมณีล้ำค่า

ปีการศึกษา 2550

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอทฤษฎีและการควบคุมการเปิด-ปิด วาล์ว อัตโนมัติ โดยโครงสร้างของระบบประกอบด้วย เกทวาล์วซึ่งติดอยู่กับ เฟืองทด และเซอร์โวมอเตอร์ แป้นพิมพ์ จอแอลซีดีแสดงผล ไมโครคอนโทรลเลอร์ และวงจรอิเล็กทรอนิกส์ ที่เกี่ยวข้อง จุดมุ่งหมายของโครงงานนี้ คือการควบคุมการเปิด-ปิด ของวาล์ว โดยการป้อนค่าเปอร์เซ็นของตำแหน่งวาล์ว ตามที่ต้องการ ตั้งแต่ 0 – 100 % แล้ว หน้าจอแอลซีดีแสดงผลค่าที่ป้อนเข้าไปและ วาล์วปิด-เปิดตามค่าที่ป้อนเข้าไป

ขั้นตอนการดำเนินการ เริ่มจากเลือกหาวาล์วที่จะนำมาควบคุมให้เหมาะสม ออกแบบโครงสร้างของการคัปปิ้ง(Coupling) ระหว่าง เฟืองทด วาล์ว และเซอร์โวมอเตอร์ ศึกษาการทำงานของไมโครคอนโทรลเลอร์แต่ละประเภท และเลือกหาแบบที่เหมาะสม ออกแบบวงจรอิเล็กทรอนิกส์ที่สอดคล้อง การทำงานใช้ ไอซี PIC ไมโครคอนโทรลเลอร์ 2 ตัว ตัวหนึ่งทำการจัดการเกี่ยวกับแป้นพิมพ์และจอแสดงผลและควบคุมโปรแกรมทั้งหมด อีกตัวหนึ่งทำหน้าที่ควบคุมเซอร์โวมอเตอร์ ซึ่งใช้การเขียนโปรแกรมภาษาซี เพื่อที่จะรับค่าที่ป้อนเข้ามา แล้วกำหนด พัลส์ (Pulse) ในการหมุนเซอร์โวมอเตอร์ตามที่ค่าที่ป้อนเข้ามา แล้วนำไปขับเซอร์โวมอเตอร์ให้หมุนตามที่ต้องการ

CONTROL VALVE

By

Mr. Prachya Skulwittayasathaworn

Mr. Parinya Atha

Mr. Pongsathorn Chairaksa

Mr. Supasak Tanakitrungruang

Advisor

Miss Wandee Petchmaneelumka

Academic Year 2007

ABSTRACT

This thesis presents theory and automatic valve control procedures. The system consists of gate valve coupling with gear and servo motor, Key-Pad, LCD Display, Microcontroller and related electronic circuits. The objective of this project is to control the valve by applying the value of percentage in 0-100% of the valve position via microcontroller. The value can be monitored through LCD display.

Procedure of making this project begins to research the valve for suitable control. Then the structure of coupling component and the related electronic circuits are designed. After that 2 PIC microcontroller with C language is used in this project. One is operated for key-pad, LCD display and control program. Another one is performed for control the servo motor.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้ สามารถที่จะสำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับการช่วยเหลือเป็นอย่างดี จาก อาจารย์วรรณดี เพชรมณีล้ำค่า ที่ได้ให้คำปรึกษาแนะนำที่ดีมาโดยตลอดตั้งแต่ต้น รวมทั้งคอยถามถึงความคืบหน้าอยู่ตลอดเวลา และความช่วยเหลืออื่นๆ ที่เป็นประโยชน์ต่อโครงงาน ผู้จัดทำรู้สึกซาบซึ้งและกราบขอบพระคุณอย่างสูง

ขอขอบพระคุณ รศ.ดร.วันชัย ริ้วรุจา ที่แนะนำสถานที่ที่ซื้ออุปกรณ์ และอุปกรณ์ที่เหมาะสม รวมถึงการแนะนำ สั่งสอน ติเตียน ให้ทำงานอย่างมีคุณภาพและมีมาตรฐานที่ดี

ขอบคุณเพื่อนๆ พี่ๆ ที่ให้ความช่วยเหลือในส่วนของการเขียนโปรแกรม ซึ่งเป็นส่วนที่สำคัญของปริญญานิพนธ์นี้

สุดท้ายนี้ผู้จัดทำขอกราบขอบพระคุณบิดา มารดา และครอบครัว ที่คอยเป็นกำลังใจที่ดีตลอดการทำงาน รวมถึงสนับสนุนงบประมาณช่วยเหลือ ตลอดจนเป็นแรงบันดาลใจที่ดีที่สุดที่ทำให้โครงงานนี้สำเร็จสมบูรณ์ลงได้

ผู้จัดทำ
นาย ปรัชญา สกุลวิทยสถาวร
นาย ปริญญา อะทะ
นาย พงศธร ไชยรักษา
นาย ศุภศักดิ์ ธนะกิจรุ่งเรือง

สารบัญ

สารบัญ

สารบัญ (ต่อ)

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 กล่าวนำ

จากการที่ในปัจจุบันตามโรงงานอุตสาหกรรมหลายๆแห่งได้มีการใช้อุปกรณ์ที่มีการควบคุมแบบอัตโนมัติอย่างแพร่หลาย ก่อให้เกิดความต้องการบุคลากรที่มีความรู้ความสามารถ ในการที่จะทำให้อุปกรณ์เหล่านี้สามารถใช้งานได้อย่างมีเสถียรภาพตลอดการดำเนินงาน ด้วยเหตุนี้นักศึกษาสาขาวิศวกรรมระบบควบคุมจึงจำเป็นต้องเรียนรู้และศึกษาทฤษฎีต่างๆตลอดจนปฏิบัติการทดลอง ซึ่งเป็นส่วนสำคัญที่จะช่วยให้เกิดความเข้าใจและสามารถยืนยันทฤษฎีที่ได้ศึกษามา เพื่อที่จะนำความรู้ที่ได้ไปประยุกต์ใช้ในการออกแบบและทำให้สามารถควบคุมอุปกรณ์ให้เป็นไปตามที่ระบบต้องการได้

สำหรับปริญญานิพนธ์นี้ ทางคณะผู้จัดทำได้เลือกทำการศึกษาและสร้างชิ้นงานเกี่ยวกับการควบคุมการเปิด-ปิดวาล์วอัตโนมัติ(Control Valve) โดยได้จำลองกระบวนการควบคุม ซึ่งถือเป็นตัวอย่างของกระบวนการควบคุมการเปิด-ปิดวาล์วอัตโนมัติ ซึ่งถือว่าไม่ซับซ้อนมากนัก

ในการศึกษาการควบคุมอุปกรณ์อัตโนมัตินี้นั้น ได้ทำการใช้ เซอร์โวมอเตอร์ เป็นตัวควบคุมการเปิด-ปิดวาล์วให้ได้ค่าตามต้องการ ซึ่งเป้าหมายของการควบคุมคือ เพื่อให้วาล์วเปิด-ปิดได้ตามค่าที่ป้อนเข้าไปอย่างถูกต้องและผิดพลาดน้อยที่สุด

1.2 วัตถุประสงค์ในการทำปริญญานิพนธ์

1. เพื่อศึกษาทฤษฎีการควบคุมอุปกรณ์แบบอัตโนมัติ
2. เพื่อศึกษาอุปกรณ์ต่างๆที่จะนำมาสร้างให้เป็นตัวควบคุมแบบอัตโนมัติ และทดลองอุปกรณ์ต่างๆที่จะนำมาใช้ งานจริง
3. เพื่อสร้างตัวควบคุมการเปิด-ปิดวาล์วแบบอัตโนมัติ โดยใช้เซอร์โวมอเตอร์เป็นตัวควบคุมการเปิด-ปิดของวาล์วโดยมีการป้อนค่าที่ต้องการผ่านแป้นพิมพ์ (Key-Pad) ให้มีการแสดงผลทางหน้าจอแอลซีดี (LCD) ตามที่ได้ป้อนค่าเข้าไป โดยทั้งหมดนี้มีไมโครคอนโทรลเลอร์เป็นตัวประมวลผลทั้งหมด

1.3 ขั้นตอนการศึกษาและการจัดทำโครงงาน

1. ศึกษาส่วนประกอบทั้งหมดที่เกี่ยวข้องในการทดลองทั้งหมดในการทำตัววาล์วควบคุม ค้นคว้าข้อมูลและศึกษาการทำงานทางกลและทางอิเล็กทรอนิกส์ ควบคู่กับการศึกษาการทำงานของอุปกรณ์ต่างๆที่ควบคุมแบบอัตโนมัติ และศึกษาการเขียนโปรแกรม

2. จัดหาวัสดุและส่วนประกอบต่างๆทั้งทางกลและทางอิเล็กทรอนิกส์ที่จะนำมาใช้ประกอบขึ้นเป็นอุปกรณ์ ได้แก่ แป้นพิมพ์ขนาด 4x4,จอแสดงผลแอลซีดี (LCD) ขนาด 16x1,เซอร์โวมอเตอร์,ตัวต้านทานปรับค่าได้10 รอบ และเกทวาล์ว
3. ทดสอบโปรแกรมควบคุมการทำงานของอุปกรณ์แต่ละชิ้น
4. ประกอบเป็นโครงสร้างชุด ควบคุมการเปิด-ปิดวาล์วอัตโนมัติ และทดสอบการทำงานของอุปกรณ์ทั้งหมด
5. เขียนโปรแกรมเพื่อควบคุมการทำงานของอุปกรณ์ที่ประกอบขึ้นมาให้สามารถควบคุมได้ทั้งหมด พร้อมทดสอบการทำงานของโปรแกรม
6. ทดลองและบันทึกผลการทดลอง รวมถึงสรุปผลการทำงานทั้งหมด

1.4 รายละเอียดของปริญญานิพนธ์

เนื้อหาที่จะกล่าวในปริญญานิพนธ์ฉบับนี้ประกอบด้วย

บทที่ 1 บทนำ กล่าวนำถึงวัตถุประสงค์ ขั้นตอนการศึกษา และการจัดทำโครงงาน พร้อมทั้งรายละเอียดของปริญญานิพนธ์แต่ละบท

บทที่ 2 ทฤษฎีและความรู้ที่เกี่ยวข้อง กล่าวถึง วาล์วแต่ละประเภทเป็นอย่างไร หลักการทำงานของ PIC ไมโครคอนโทรลเลอร์และ เซอร์โวมอเตอร์ เพื่อนำไปประยุกต์ใช้ในการจัดทำโครงงาน

บทที่ 3 การคำนวณและการสร้าง กล่าวถึง การศึกษาค้นคว้า อธิบายส่วนประกอบและหลักการเลือก ขั้นตอนในการสร้าง และหลักการทำงานส่วนต่างๆ

บทที่ 4 การทดลอง กล่าวถึง การทดลองวงจรและภาคขับมอเตอร์ การทดลองการป้อนค่าและแสดงผล และการทดลองสัญญาณป้อนกลับในตัวต้านทานปรับค่าได้

บทที่ 5 บทวิจารณ์และสรุป กล่าวถึง บทสรุปของโครงงาน ปัญหาที่เกิดขึ้น แนวทางในการแก้ไข และแนวทางในการปรับปรุงโครงงานต่อไป

บทที่ 2
ทฤษฎีและความรู้ที่เกี่ยวข้อง

จากที่ได้กล่าวมาในบทที่ 1 แล้วว่า ก่อนที่จะทำการสร้างชิ้นโครงงานตลอดจนรายละเอียดต่างๆ จำเป็นอย่างยิ่งที่จะต้องมีการศึกษาองค์ประกอบต่างๆ ที่จำเป็นของระบบควบคุมให้เข้าใจอย่างถ่องแท้เสียก่อน เริ่มจากตัววาล์วซึ่งในปัจจุบันมีหลากหลายประเภท ซึ่งขึ้นอยู่กับการใช้งาน แล้วเลือกวาล์วที่นำมาควบคุมที่เหมาะสม การควบคุมอัตโนมัตินั้นต้องรู้ทฤษฎีระบบควบคุมแบบต่างๆ รวมถึงการปรับแต่งค่าตัวควบคุม จากนั้นจะเป็นส่วนของส่วนประกอบทางกล ซึ่งก็คือเซอร์โวมอเตอร์ ว่ามันมีหลักการทำงานอย่างไร ในส่วนสุดท้ายจะเป็นการทำงานของไมโครคอนโทรลเลอร์ โดยในที่นี้เลือกใช้ เบอร์
PIC16F877 โดยมีรายละเอียด ดังนี้

2.1) วาล์ว (Valve)

วาล์ว ถ้าแยกเป็นประเภทของตัวถัง ในปัจจุบันมีมากมายหลายประเภท โดยจะอธิบายแบบที่นิยมใช้งานกันในโรงงานอุตสาหกรรม ดังนี้

2.1.1) Globe Valve เป็นแบบที่นิยมใช้กันมากที่สุด ซึ่งมีชนิดและขนาดที่ต่างๆกัน สำหรับทิศทางการไหลนั้นมีทั้งชนิด Flow-to-Close กล่าวคือ ความดันการไหลจะไปดัน Valve Plug เข้าหาบ่าวาล์ว ส่วนแบบ Flow-to-open ซึ่งหมายถึงความดันของการไหลนั้นจะดัน Valve Plug ให้ถอยห่างออกจากบ่าวาล์ว โดยชนิดนี้จะนิยมใช้มากกว่าเพราะเป็นชนิดที่ควบคุมให้มีเสถียรภาพได้ดีส่วน Valve Body ในแบบ Globe Valve นี้ยังแบ่งย่อยตามลักษณะของช่องทางเดินได้อีก 2 แบบ คือ Single-Port Globe Valve และ Double-Port Globe Valve สำหรับลักษณะของ Globe Valve โดยทั่วไปจะเป็นดังรูปที่ 2.1

รูปที่ 2.1 แสดงลักษณะทั่วไปของ Globe Valve

2.1.2) Butterfly Valve เป็น Valve Body แบบหนึ่งของ Rotary shaft Valve มีโครงสร้างและส่วนประกอบง่ายๆมีขนาดตั้งแต่ 2 นิ้ว ขึ้นไปจนถึง 72 นิ้ว เหมาะที่จะนำไปใช้งานในระบบที่มีความดันต่ำ แต่มีปริมาณการไหลมาก คุณลักษณะการไหลเป็นแบบ Equal Percentage

Butterfly valve ที่มีลักษณะของ Disc เป็นแบบ Convetional Disc คือแนวศูนย์กลางของ Shaft อยู่ในแนวเดียวกับ Disc แต่ถ้าเป็นแบบ Eccentric Disc จะออกแบบให้แนวศูนย์กลางของ Disc อยู่เยื้องศูนย์กับแนวของ Shaft ซึ่งมีข้อดีคือ ทำให้ไดแรงในการปิดสนิท (Shut Off) ได้มากกว่าแบบแรก ดังนั้นอัตราการรั่วซึม (Leakage) จะน้อยกว่าแบบแรก ลักษณะทั่วไปของ Butterfly Valve มีลักษณะดังรูปที่ 2.2

รูปที่ 2.2 แสดงลักษณะทั่วไปของ Butterfly Valve

2.1.3) Ball Valve จัดเป็น Valve Body แบบหนึ่งที่มีการเคลื่อนไหวแบบ Rotary ที่ออกแบบให้เกิด Pressure Drop น้อยที่สุด (ในกรณีที่เป็นแบบ Full Ball) เมื่อมีอัตราการไหลมาก คือมี Flow Capacity มากกว่า Globe Valve เมื่อมีขนาดเท่ากัน Ball Valve จัดอยู่ในพวก High Recovery Valve ทำให้เกิดความดันตกคร่อมน้อย แต่โอกาสที่จะเกิด Cavitations นั้นมีมากกว่าแบบอื่นๆแต่โดยทั่วไปเราจะแบ่งประเภทของ Ball Valve ออกเป็น 2 ประเภท ตามลักษณะการออกแบบของผู้ผลิต คือ Full Ball Valve หรือเรียกว่า Full Port และ Segmented Ball Valve โดยที่ลักษณะทั่วไปของBall Valve จะเป็นดังรูปที่ 2.3

รูปที่ 2.3 แสดงลักษณะทั่วไปของ Ball Valve

2.1.4) Gate Valve เกทวาล์วที่ปรากฏดังรูปที่ 2.3 เป็นเกทวาล์วที่ชิ้นส่วนปิดเป็นรูปลิ่ม โดยจุดมุ่งหมายที่ทำให้เป็นรูปลิ่มเพื่อทำให้เกิดแรงปิดวาล์วเพิ่มสูงขึ้น อันสามารถใช้เกทวาล์วรูปลิ่มบ่าโลหะใช้กันรั่วซึ่งไม่เพียงแต่ด้านความดันของไหลที่สูงเท่านั้นยังใช้กับความดันของไหลต่ำได้ด้วย ดังนั้นระดับของการผนึกแน่นบ่าวาล์วที่สามารถทำสำเร็จได้ด้วยเกทวาล์วลิ้นลิ่มบ่าโลหะมีศักยภาพสูงกว่าด้วยเกทวาล์วลิ้นขนานบ่าโลหะ อย่างไรก็ตามแรงการปิดวาล์วต้นกระแสเนื่องจากการอัดลิ่มไม่สูงพอที่จะผนึกกันรั่วบ่าวาล์วด้านต้นกระแสได้ด้วยเกทวาล์วลิ้นลิ่มบ่าโลหะ

สำหรับตัวเรือนของวาล์วชนิดนี้มีโครงนำสำหรับให้ลิ้นลิ่มเคลื่อนที่โดยโครงนำช่วยป้องกันลิ่มจากการหมุนระหว่างการเคลื่อนที่เพื่อให้มั่นใจว่า มีการตรงศูนย์อย่างเหมาะสมของส่วนปิดวาล์วและทำให้ลิ่มแยกออกจากบ่าวาล์วด้านปลายกระแส ยกเว้นสำหรับระยะทางสั้นๆใกล้ๆ ของตำแหน่งวาล์วปิดดังนั้นลดการสึกหรอของส่วนปิดวาล์วให้น้อยลง กรณีที่แตกต่างไปคือวาล์วที่ลิ่มถูกนำเพียงลำพังด้วยแผ่นไดอะแกรมที่ทาบอยู่บนผิวหน้าลิ้นลิ่ม

ทางด้านข้อจำกัด เกทวาล์วลิ้นลิ่มไม่สามารถประกอบกับท่อตามเป็นเกทวาล์วท่อได้สะดวกนัก และการขยายตัวทางความร้อนของก้านวาล์วสามารถทำให้ส่วนปิดวาล์วรับแรงมากเกินไป ยิ่งไปกว่านั้นส่วนปิดวาล์วของเกทวาล์วลิ้นลิ่มมีแนวโน้มที่จะดักเศษของแข็งที่ถูกพามากับของไหลที่ไหลผ่านมากยิ่งกว่าเกทวาล์วลิ้นขนาน และอีกอย่างคือเกทวาล์วลิ้นลิ่มปกติไม่เหมาะสำหรับหน้าที่ในการปรับแต่งการไหล แต่ใช้เป็นหลักสำหรับหน้าที่เปิด-ปิด และสภาพการทำงานไม่บ่อยนัก

รูปที่ 2.4 แสดงลักษณะทั่วไปของ Gate Valve

2.1.5) Check Valve เช็ควาล์วหรือวาล์วกันกลับ (Nonreturn Valve) เป็นวาล์วที่ทำงานโดยอัตโนมัติ ซึ่งจะเปิดเมื่อมีของไหลไหลผ่าน และปิดเมื่อของไหลมีทิศทางย้อนทางเดิม

จากลักษณะการควบคุมด้วนการไหลดังกล่าวเป็นความต้องการในการป้องกันการไหลย้อนกลับเนื่องจากทำให้เกิดผลบางอย่างที่ไม่พึงปรารถนาหากมีการไหลย้อนกลับเกิดขึ้น เช่น ทำให้อุปกรณ์ส่งของไหลจำพวกปั๊มต่างๆหรือเครื่องอัดอากาศหมุนกลับทิศทางอันจะก่อเกิดความเสียหายแก่อุปกรณ์ได้ นอกจากนี้เช็ค

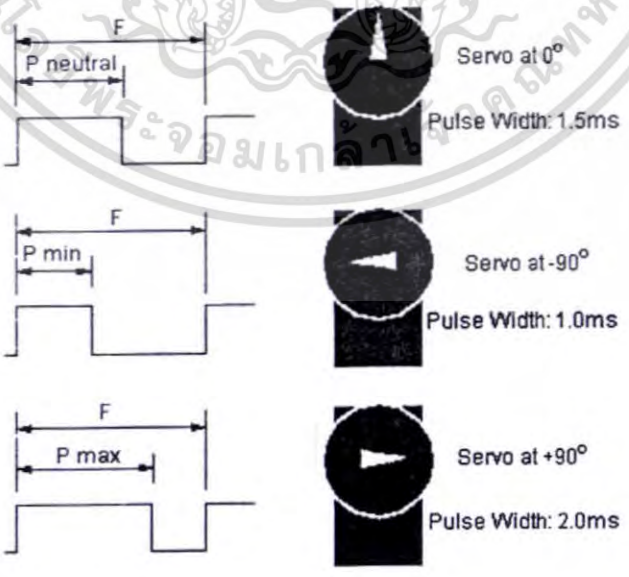
วาล์วอาจจำเป็นในเส้นทางน้ำป้อนที่บางครั้งอาจเกิดความดันเพิ่มขึ้นอย่างกะทันหันทางด้านปลายความดันต่ำจนกระทั่งมีความดันสูงกว่าความดันทางด้านปลายความดันสูง

การทำงานของเช็ควาล์วนั้นจะทำงานในลักษณะที่หลีกเลี่ยงการเกิดความดันที่มีลักษณะขึ้นๆลงๆเป็นระลอกๆสูงเกินไปอันเป็นผลจากการปิดวาล์วและอีกอย่างคือหลีกเลี่ยงการเคลื่อนที่แกว่งไป-มาอย่างรวดเร็วของลิ้นปิดวาล์ว

รูปที่ 2.5 แสดงลักษณะทั่วไปของ Check Valve

2.2) หลักการทำงานของเซอร์โวมอเตอร์

การควบคุมการทำงานของเซอร์โวมอเตอร์ (Servo Motor) ทำได้โดย การป้อนสัญญาณความกว้างพัลส์ให้กับมอเตอร์ ซึ่งตำแหน่งและทิศทางการหมุนของมอเตอร์นี้จะขึ้นอยู่กับขนาดของความกว้างของพัลส์นั้นๆ โดยทั่วไปแล้วความกว้างของสัญญาณพัลส์ จะมีจุดให้อ้างอิง 3 จุด ดังรูปที่2.6คือ



รูปที่2.6 การทำงานของเซอร์โวมอเตอร์

- สัญญาณความกว้างพัลส์ขนาด 1.5 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม 0 องศา หรือจุดกึ่งกลางของมอเตอร์
- สัญญาณความกว้างพัลส์ขนาด 1 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม -90 องศา หรือในทิศทางทวนเข็มนาฬิกา
- สัญญาณความกว้างพัลส์ขนาด 2 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม +90 องศา หรือในทิศทางตามเข็มนาฬิกา

2.3) ไมโครคอนโทรลเลอร์ PIC 16F877

ในปัจจุบันไมโครคอนโทรลเลอร์ตระกูล PIC มีการพัฒนาและผลิตออกมาหลายเบอร์ โดยทั่วไปจะแบ่งออกเป็น 3 แบบ คือ

1. สถาปัตยกรรมแบบ 12-Bit Core (Base-Line) เป็นกลุ่มของไมโครคอนโทรลเลอร์ที่มีขนาดเล็กมีโครงสร้างของคำสั่งเพียง 12 bits และค่อนข้างมีข้อจำกัดในการใช้งาน เนื่องจากมีหน่วยความจำ RAM และ STACK ค่อนข้างจำกัด
2. สถาปัตยกรรมแบบ 14-Bit Core (Mid-Range) เป็นกลุ่มของไมโครคอนโทรลเลอร์ที่มีขนาดกลางมีโครงสร้างคำสั่ง 14 bits มีทั้งแบบที่โปรแกรมครั้งเดียว (OTP : One Time Programmable) และแบบแฟลช (Flash Memory)
3. สถาปัตยกรรมแบบ 16-Bit Core (High-End) เป็นกลุ่มของไมโครคอนโทรลเลอร์ที่อยู่ในกลุ่มระดับสูงซึ่งได้มีการพัฒนาโครงสร้างสถาปัตยกรรม ทั้งในเรื่องของหน่วยความจำ ความเร็ว และ คุณสมบัติอื่น ๆ ที่เหนือกว่าสองกลุ่มที่ผ่านมามาก มีการจัดวางหน่วยความจำโปรแกรมอยู่ในเพจ (Page) เดียวกัน ทำให้ไม่มีปัญหาเรื่องรอยต่อของหน่วยความจำ

นอกจากนี้แล้ว PIC ยังแบ่งออกเป็นประเภทของหน่วยความจำอีกด้วย โดยจะมีการจำแนกเป็น 3 ประเภท คือ

1. C เช่น PIC16CXXX คือ มีโครงสร้างหน่วยความจำเป็น EPROM จัดอยู่ในจำพวกอุปกรณ์ OTP แต่สามารถลบได้ด้วยแสง UV
2. CR เช่น PIC16CRXXX คือ มีโครงสร้างหน่วยความจำเป็น ROM จัดอยู่ในจำพวกอุปกรณ์ OTP ไม่สามารถลบได้
3. F เช่น PIC16FXXX คือ มีโครงสร้างหน่วยความจำเป็น FLASH Memory สามารถทำการโปรแกรม ลบ แล้วโปรแกรมซ้ำได้หลายครั้ง

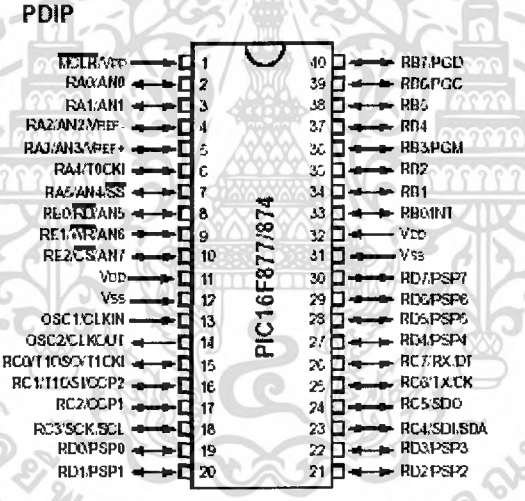
2.3.1 แรงดันในการทำงาน

ช่วงแรงดันการทำงานของ PIC โดยปกติมาตรฐานแล้วจะอยู่ระหว่าง 4.5-6.0 V แต่จะมีบางเบอร์ที่ออกแบบมาให้สามารถทำงานได้ในช่วงแรงดันต่ำ ประมาณ 2.5-6 V ได้ซึ่งจะมีการระบุโค้ดให้ทราบ โดยมีการเพิ่มรหัสตัว Lเข้าไปในเบอร์ของอุปกรณ์ เช่น PIC16LFXXX เป็นต้น

2.3.2 สัญญาณนาฬิกา

PIC จะใช้สัญญาณนาฬิกาโดยมองเป็นลักษณะของ วงรอบ (cycle) ซึ่งระบุเอาไว้ว่า 1 คำสั่งนั้นจะประกอบไปด้วย 1-2 วงรอบ โดยแต่ละวงรอบนั้นจะแบ่งเป็น 4 ส่วน คือ Q1, Q2, Q3 และ Q4 ด้วยเหตุนี้ความเร็วโดยรวมของ PIC จึงเท่ากับ ค่าความถี่ของสัญญาณนาฬิกา หารด้วย 4

$$1\ \text{cycle} = Q_1 + Q_2 + Q_3 + Q_4 = \frac{XTAL}{4}$$



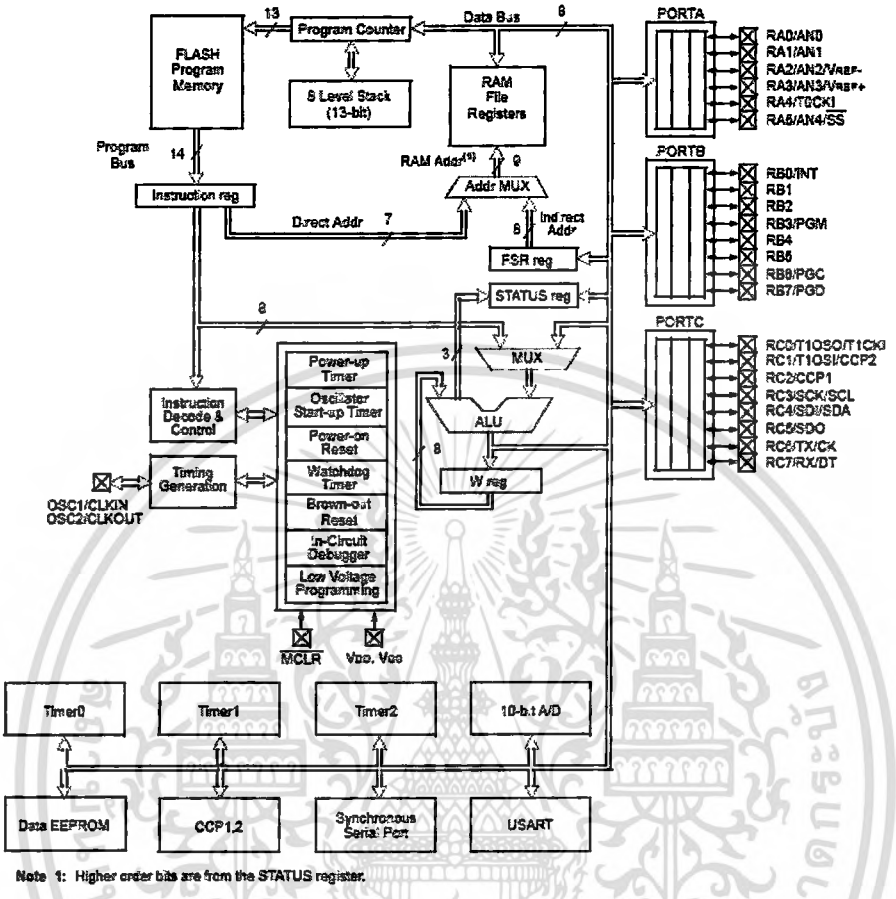
รูปที่ 2.7 การจัดขาของ PIC16F877

2.3.3 คุณสมบัติของ PIC16F877

1. ซีพียูเป็นแบบ RISC (Reduce Instruction-Set) มีคำสั่งให้ใช้งาน 35 คำสั่ง
2. คำสั่งหนึ่งๆใช้เวลาทำงาน 1 ถึง 2 Cycle
3. ทำงานได้สูงสุดที่ 20MHz (PIC16F877-20/P)
4. ทำงานแบบ Pipe-line (มี 2 ท่อ) ทำให้ ณ เวลาหนึ่งทำงาน 2 อย่างพร้อมๆกันได้
5. หน่วยความจำโปรแกรมเป็นแบบ Flash มีขนาด 8KWord (1 word=14 บิต)
6. มี RAM ขนาด 368 ไบต์

7. มี EEPROM ขนาด 256 ไบต์
8. ตอบสนองกับอินเตอร์รัพได้ทั้งหมด 14 แหล่ง
9. มี Stack ให้ใช้ได้สูงสุด 8 ระดับ
10. มีระบบ POR (Power On Reset), PWRT (Power Up Timer), OST (Oscillator Start-up timer)
11. มีระบบ WDT (Watchdog timer)
12. มีระบบCP (Code Protection) และสามารถเลือกระดับการป้องกันได้
13. มีโหมดประหยัดเงิน
14. สัญญาณนาฬิกามีหลายโหมดให้เลือกใช้งาน คือ อาจจะใช้ XTAL หรือ วงจร RC ก็ได้
15. สามารถโปรแกรมด้วยไฟ +5VDC ได้
16. สามารถโปรแกรมแบบ In-Circuit Serial Programming
17. ทำงานที่ไฟเลี้ยง 2VDC ถึง 5.5VDC
18. Current Sink และ Current Source อยู่ที่ 25mA
19. มี Timer/Counter 3 ตัว
20. มีโมดูล Capture/Compare/PWM อีก 2 ชุด
21. มี A-TO-D Converter แบบ 10 บิต จำนวน 8 ช่องนำเข้า ในตัวเอง
22. มีระบบ USART สำหรับการสื่อสารแบบอนุกรม
23. มีระบบตรวจระดับไปเลี้ยง BODEN (Brown-out reset) เพื่อสร้างสัญญาณรีเซตซีพียู BOR (Brown-Out Reset)
24. มี I/O พอร์ตทั้งหมด 5 พอร์ต

2.3.4 โครงสร้างการทำงานภายในของ PIC16F877



รูปที่2.8 โครงสร้างการทำงานภายใน PIC16F877

2.3.5 รายละเอียดของขา PIC16F877

ตารางที่ 2.1 รายละเอียดของขา PIC16F877

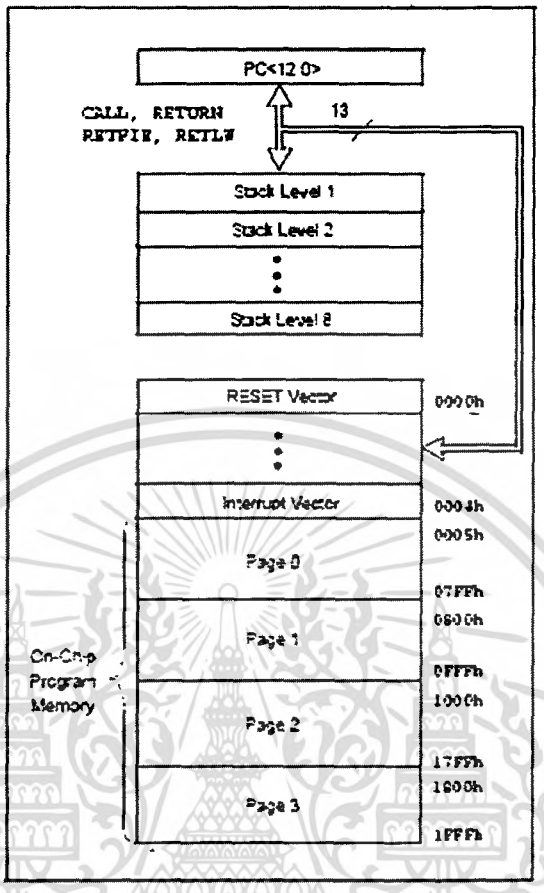
ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียด
OSC1/CLKIN	13	I	ST/CMOS	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	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	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	I/O	ST	RA4 can also be the clock input to the Timer0 timer/ counter. Output is open drain type.
RA5/SS/AN4	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
				PORTB is a bi-directional I/O port. PORTB can be soft-ware programmed for internal weak pull-up on all inputs. RB0 can also be the external interrupt pin.
RB0/INT	33	I/O	TTL/ST	
RB1	34	I/O	TTL	
RB2	35	I/O	TTL	
RB3/PGM	36	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	I/O	TTL	Interrupt-on-change pin.
RB5	38	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	I/O	TTL/ST	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	I/O	TTL/ST	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
				PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	I/O	ST	RC3 can also be the synchronous serial clock input/ output for both SPI and I2C modes.
RC4/SDI/SDA	23	I/O	ST	RC4 can also be the SPI Data-In (SPI mode) or data I/O (I2C mode).
RC5/SDO	24	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.

ตารางที่ 2.1 (ต่อ) รายละเอียดของขา PIC16F877

ชื่อขา	ตำแหน่งขา	ชนิดของขา	ชนิดของวงจรบัฟเฟอร์	รายละเอียด
RD0/PSP0 RD1/PSP1 RD2/PSP2 RD3/PSP3 RD4/PSP4 RD5/PSP5 RD6/PSP6 RD7/PSP7	19 20 21 22 27 28 29 30	I/O I/O I/O I/O I/O I/O I/O I/O	ST/TTL ST/TTL ST/TTL ST/TTL ST/TTL ST/TTL ST/TTL ST/TTL	PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RE0/RD/AN5 RE1/WR/AN6 RE2/CS/AN7	8 9 10	I/O I/O I/O	ST/TTL ST/TTL ST/TTL	PORTE is a bi-directional I/O port. RE0 can also be read control for the parallel slave port, or analog input5. RE1 can also be write control for the parallel slave port, or analog input6. RE2 can also be select control for the parallel slave port, or analog input7.
VSS	12 31	P	—	Ground reference for logic and I/O pins.
VDD	11 32	P	—	Positive supply for logic and I/O pins.
NC	—		—	These pins are not internally connected. These pins should be left unconnected.

2.3.6 การจัดสรรหน่วยความจำโปรแกรมของ PIC16F877

PIC16F877 จะมีขนาดของหน่วยความจำโปรแกรม ซึ่งสามารถอ้างอิงได้ถึง8K byte โดยPIC16F877 จะมีขนาดหน่วยความจำเท่ากับ 8K x 14 bits ซึ่งตำแหน่ง Reset Vector จะอยู่ที่ 0000h และ Interrupt Vector จะอยู่ที่ 004h PIC จะแบ่งหน่วยความจำโปรแกรมออกเป็นหน้า ซึ่งแต่ละหน้า ก็จะมีขนาด 2 Kbytes ซึ่งคำสั่ง CALL และ GOTO สามารถ สั่งให้ Program Counter กระโดดไปมาได้ในช่วง Page เท่านั้น แต่ถ้าเมื่อต้องการกระโดดจาก Page หนึ่ง ไปยังอีก Page หนึ่ง ดังนั้นจึงต้องไปควบคุม PCALTH<4:3> (Bit Address ที่ 12 และ 13 ให้ชี้ไปยังPageที่ต้องการเสียก่อนหลังจากนั้นจึงเรียกคำสั่ง CALL หรือ GOTO ตามอีกที)



รูปที่ 2.9 การจัดสรรหน่วยความจำโปรแกรมของ PIC16F877

การเลือกหน้า ของหน่วยความจำโปรแกรม จะต้องเลือกในรีจิสเตอร์ PCALTH โดยการระบุตำแหน่งที่บิต 3 และบิต 4

ตารางที่ 2.2 ค่า PCLATH<4:3> ในการเลือกหน้า ของ PIC16F877

PCLATH<4:3>	PAGE
00	0
01	1
10	2
11	3

เมื่อใช้คำสั่ง CALL ไปที่ Routine ใด Routine หนึ่งแล้ว แล้วจะใช้คำสั่ง RETURN ในการกลับไปตำแหน่งเดิม การ RETURN กลับนั้นไม่จำเป็นต้องสั่ง PCALTH ให้ชี้ไปยังหน้า ก่อนหน้าที่จะเรียก CALL เพราะค่า Address ดังกล่าวจะถูกเก็บไว้ใน STACK อยู่แล้วแต่สำหรับคำสั่ง GOTO เวลาข้ามหน้า จะต้องสั่งให้ PCALTH ชี้ไปยังหน้าที่ต้องการจะไปทุกครั้ง

2.3.7 การจัดสรรหน่วยความจำข้อมูล RAM ของ PIC16F877

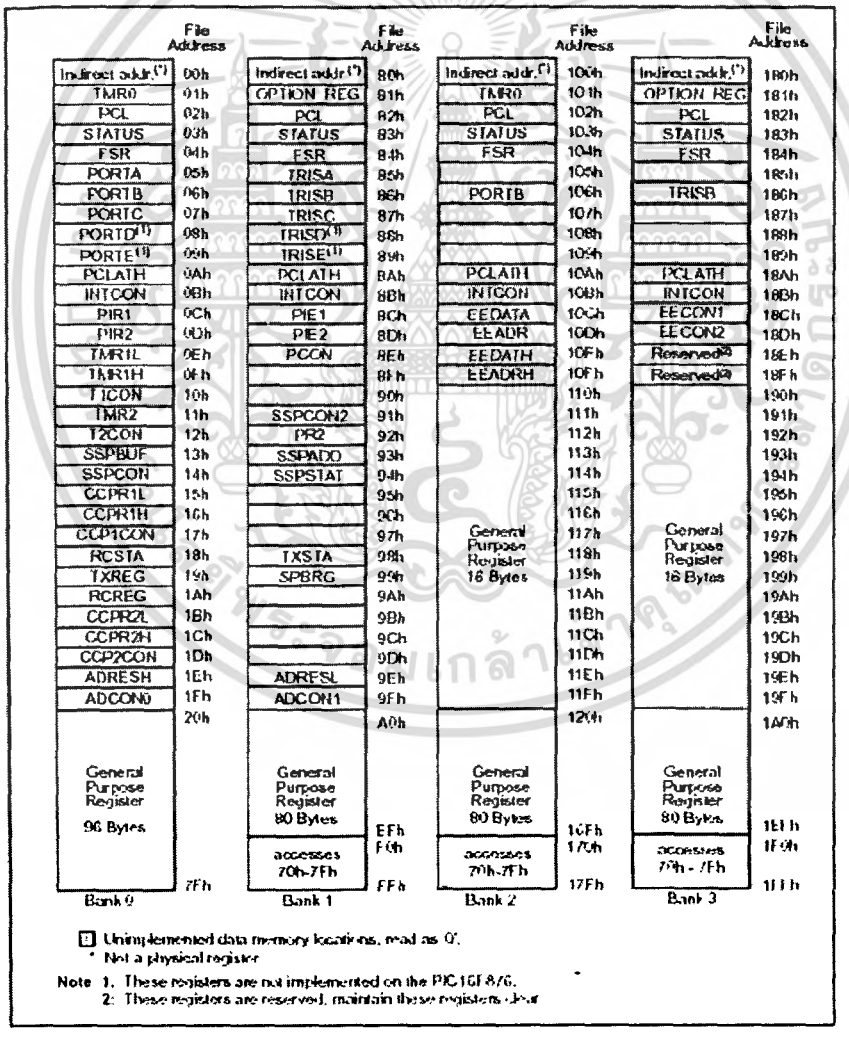
PIC16F877 มีหน่วยความจำข้อมูล RAM สำหรับใช้วานทั่วไป 368 bytes และมีรีจิสเตอร์ไฟล์ 8 bits 57 ตัว ดังรูปที่2.19 แต่ละ Bank มีขนาดสูงสุด 128 bytes แต่มีการใช้งานได้จริงในแต่ละ Bank ต่างกันโดยในแต่ละ Bank มีการจัดสรรพื้นที่ดังนี้

- Bank 0 มีช่วง Address 0x00-0x7F
  -Address 0x00-0x1F เป็นพื้นที่รีจิสเตอร์ไฟล์
  -Address 0x20-0x7F เป็นพื้นที่ของหน่วยความจำข้อมูลสำหรับใช้งานทั่วไป 96 bytes
- Bank 1 ทีช่วง Address 0x80-0xFF
  -Address 0x80-0x9F เป็นพื้นที่ของรีจิสเตอร์ไฟล์ แต่มีบาง Address ไม่ใช้งาน
  - Address 0xA0-0xEF เป็นพื้นที่ของหน่วยความจำข้อมูลสำหรับใช้งานทั่วไป 80 bytes
  - Address 0xF0-0xFF บรรจุข้อมูลเหมือนกับใน Address 0x70-0x7F ใน Bank 0 เพื่อช่วยให้สามารถใช้ข้อมูลจาก Address 0x70-0x7F ได้ง่ายขึ้นโดยไม่ต้องเปลี่ยน Bank
- Bank 2 มีช่วง Address 0x100-0x17F
  - Address 0x100-0x10F เป็นพื้นที่ของรีจิสเตอร์ไฟล์ แต่มีบาง Address ไม่ใช้งาน
  - Address 0x110-0x11F เป็นพื้นที่ของหน่วยความจำข้อมูลสำหรับใช้งานทั่วไป 16 bytes
  - Address 0x120-0x16F เป็นพื้นที่ของหน่วยความจำข้อมูลสำหรับใช้งานทั่วไป 80 bytes
  - Address 0x170-0x17F บรรจุข้อมูลเหมือนกับใน Address 0x70-0x7F ใน Bank 0 เพื่อช่วยให้สามารถใช้ข้อมูลจาก Address 0x70-0x7F ได้ง่ายขึ้น โดยไม่ต้องเปลี่ยน Bank
- Bank 3 มีช่วง Address 0x180-0x1FF
  - Address 0x180-0x18F เป็นพื้นที่ของรีจิสเตอร์ไฟล์ แต่มีบาง Address ไม่ใช้งาน
  - Address 0x190-0x19F เป็นพื้นที่ของหน่วยความจำข้อมูลสำหรับใช้งานทั่วไป 16 bytes
  - Address 0x1A0-0x1EF เป็นพื้นที่ของหน่วยความจำข้อมูลสำหรับใช้งานทั่วไป 80 bytes
  - Address 0x1F0-0x1FF บรรจุข้อมูลเหมือนกับใน Address 0x70-0x7F ใน Bank 0 เพื่อช่วยให้สามารถใช้ข้อมูลจาก Address 0x70-0x7F ได้ง่ายขึ้น โดยไม่ต้องเปลี่ยน Bank

การเลือก Bank ของหน่วยความจำข้อมูล RAM จะต้องเลือกในรีจิสเตอร์ STATUS โดยการระบุตำแหน่งที่บิต 5 และบิต 6 (RP0, RP1)

ตารางที่ 2.3 ค่า RP0 และค่า RP1 ในการเลือก Bank ของ PIC16F877

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



รูปที่2.10 การจัดสรรหน่วยความจำข้อมูล RAM และตำแหน่งรีจิสเตอร์ ของ PIC16F877

2.3.8 รีจิสเตอร์หลักของ PIC16F877

2.3.8.1 รีจิสเตอร์ Configuration word

มีขนาด 14 bits อยู่ที่ Address 2007h ใน Program Memory โดยการทำงานเบื้องต้นของ PIC จะถูกกำหนดที่หน่วยความจำตรงนี้ ไม่ว่าจะเป็น Enable/Disable Power-up timer, Enable/Disable Watchdog, Oscillator bits (กำหนดที่มาของสัญญาณนาฬิกา) หน่วยความจำที่ตำแหน่งนี้ จะต้องกำหนดในขณะทำการเขียนโปรแกรมลงสู่ Flash Memory ของ PIC

2.3.8.2 รีจิสเตอร์ STATUS

เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลแสดงสถานการณ์ทำงานของ PIC16F877

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	$\overline{TO}$	$\overline{PD}$	Z	DC	C
Bit 7							Bit0

รูปที่2.11 รีจิสเตอร์ STAUS

บิต 7 **IRP** (Indirect Register Bank Select bit) ใช้เลือก Bank ของหน่วยความจำข้อมูล RAM เมื่อใช้การอ้างตำแหน่งโดยอ้อม

1 = Bank 2, 3 (100h-1FFh)

0 = Bank 0, 1 (00h-FFh)

บิต 6-5 **RP1:RP0** (Register Bank Select bits) ใช้เลือก Bank ของหน่วยความจำข้อมูล RAM เมื่อใช้การอ้างตำแหน่งโดยตรง

11 = Bank 3 (180h-1FFh)

10 = Bank 2 (100h-17Fh)

11 = Bank 1 (80h-FFh)

11 = Bank 0 (00h-7Fh)

บิต 4 **TO** (Time-out bit) บิตแสดงการทำงานโหมดประหยัดพลังงาน

1 = After power-up or by the CLRWDT instruction

0 = By execution of the SLEEP instruction

บิต 2 **Z** (Zero bit) ใช้แสดงผลการกระทำคำสั่งทางคณิตศาสตร์

1 = The result of an arithmetic or logic operation is zero

0 = The result of an arithmetic or logic operation is not zero

บิต 1 **DC** (Digit carry/borrow bit) ใช้แสดงการทดหรือยืมระหว่างหลักในกรณีคำสั่ง ADDWF หรือ ADDLW

1 = A carry-out from the 4th low order bit of the result occurred

0 = No carry-out from the 4th low order bit of the result

และแสดงค่ากลับกันเมื่อกระทำคำสั่ง SUBWF หรือ SUBLW

บิต 0 **C** (Carry/borrow bit) ใช้แสดงการทดหรือยืมของบิต MSB

1 = A carry-out from the Most Significant bit of the result occurred

0 = No carry-out from the Most Significant bit of the result occurred

2.3.8.3 รีจิสเตอร์ W เป็นรีจิสเตอร์ที่มีบทบาทสำคัญ เพราะในการประมวลผลทางคณิตศาสตร์จะต้องกระทำผ่านรีจิสเตอร์ W และยังทำหน้าที่เป็นตัวกลางในการส่งผ่านสถานะของ Output ไปยัง I/O PORT อีกด้วย

2.3.8.4 Program Counter (PC) เป็นรีจิสเตอร์พิเศษที่ใช้ระบุ Address ของ Program Memory ที่กำลังทำการประมวลผล ซึ่งจะเป็น Counter ขนาด 13 bits โดยทั่วไปแล้ว Counter ตัวนี้จะเพิ่มขึ้น 1 ทุกๆ ครั้งเมื่อมีการประมวลผลคำสั่งเกิดขึ้น 1 ครั้ง ซึ่งค่าที่แสดงก็คือตำแหน่งของคำสั่งต่อไปที่จะทำการประมวลผล แต่เมื่อประมวลคำสั่ง JUMP ตัว counter จะมีค่าเท่ากับตำแหน่งที่คำสั่ง JUMP นั้นอ้างถึง

2.3.8.5 Stack เป็นหน่วยความจำสำรองสำหรับเก็บค่าของ PC ขนาด 13 bits โดยเก็บข้อมูลได้ 8 ระดับ โดยเก็บตำแหน่งของ PC เข้าเมื่อมีคำสั่ง CALL และส่งตำแหน่งที่เก็บไว้ออกไปยัง PC เมื่อมีคำสั่ง RETURN โดยการเก็บจะเป็นแบบ LIFO (Last In First Out)

2.3.8.6 PORTA, PORTB, ... เก็บค่าสถานะของ PORT นั้นๆ

2.3.8.7 TRISA, TRISB, ... ใช้กำหนดทิศทางของขาของ PORT นั้นๆ ว่าขาใดเป็น Input หรือ Output โดยถ้ากำหนดให้เป็น 0 จะเป็น Output ถ้าให้เป็น 1 จะเป็น Input

2.3.8.8 รีจิสเตอร์ CCP1CON และ CCP2CON เป็นรีจิสเตอร์ควบคุมโมดูล CCP1 และ CCP2 (Address 17h, 1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0

รูปที่2.12 รีจิสเตอร์ CCPxCON

บิต 7-6 . ไม่ใช้งานอ่านค่าเป็น "0"

บิต 5-4 CCPxX : CCPxY ใช้เก็บค่าดิวตี้ไซเกิลของสัญญาณ PWM ใน 2 บิตล่าง จะใช้งานเมื่อโมดูล CCPx ได้รับการกำหนดให้ทำงานในโหมด PWM ส่วนข้อมูล 8 บิตบน จะใช้จากรีจิสเตอร์ CCPRxL

บิต 3-0 CCPxM3: CCPxM0 ใช้เลือกการทำงานของโมดูล CCPx

"0000" – หยุดหรือปิดการใช้งานของโมดูล CCPx ทำให้โมดูล CCPx เกิดรีเซต

"0100" – ทำงานในโหมดตรวจจับสัญญาณ โดยตรวจจับที่ทุกๆ ขอบขาลงของสัญญาณ

"0101" – ทำงานในโหมดตรวจจับสัญญาณ โดยตรวจจับที่ทุกๆ ขอบขาขึ้นของสัญญาณ

"0110" – ทำงานในโหมดตรวจจับสัญญาณ โดยตรวจจับที่ทุกๆ ขอบขาขึ้นที่ 4 ของสัญญาณ

"0111" – ทำงานในโหมดตรวจจับสัญญาณ โดยตรวจจับที่ทุกๆ ขอบขาขึ้นที่ 16ของสัญญาณ

"1000" – ทำงานในโหมดตรวจเปรียบเทียบข้อมูล แจ้งผลการเปรียบเทียบด้วยการเซตเอาท์พุต เมื่อเท่ากัน (CCPxIF จะเซต)

"1001" – ทำงานในโหมดตรวจเปรียบเทียบข้อมูล แจ้งผลการเปรียบเทียบด้วยการเคลียร์เอาท์พุตเมื่อเท่ากัน (CCPxIF จะเซต)

"1010" – ทำงานในโหมดตรวจเปรียบเทียบข้อมูล เมื่อค่าเท่ากันจะทำการอินเตอร์รัปต์ (CCPxIF จะเซต แต่ สถานะที่ CCPx จะไม่ได้รับผลกระทบ)

"1011" – ทำงานในโหมดตรวจเปรียบเทียบข้อมูล เมื่อค่าเท่ากันจะทำการกำเนิดสัญญาณกระตุ้นเหตุการณ์พิเศษ Trigger Special Event (CCPxIF จะเซต แต่สถานะที่ CCPx จะไม่ได้รับผลกระทบ และทำการรีเซตค่า TMR1)

"11xx" – ทำงานในโหมดสร้างสัญญาณ PWM

2.3.9 การทำงานในโหมด PWM

โมดูล CCPx จะทำการกำเนิดสัญญาณมอดูเลชั่นทางความกว้างพัลซ์ (Pulse Width Modulation: PWM) ความละเอียด 10 บิต สัญญาณ PWM ที่สร้างขึ้นจะส่งออกทางขา RC2/CCP1 หรือ RC1/T1OSI/CCP2

2.3.9.1 หลักการสร้างสัญญาณ PWM

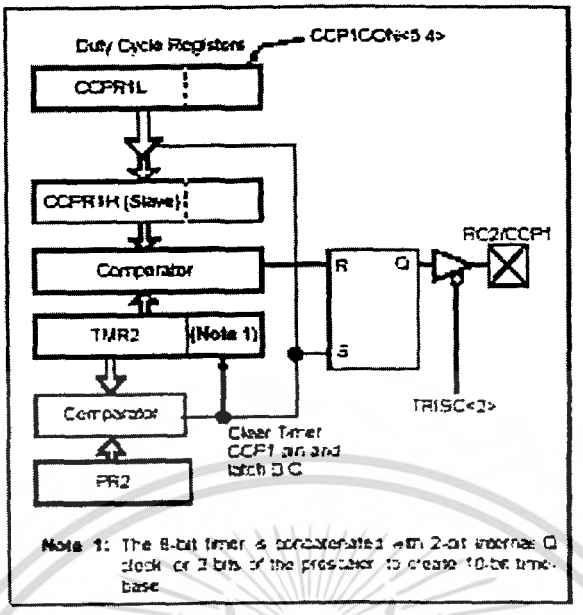
1. กำหนดค่าให้แก่รีจิสเตอร์ PR2 เพื่อกำหนดค่าเวลาของสัญญาณ PWM

2. กำหนดค่าดิวตี้ไซเกิล โดยเขียนข้อมูลลงในรีจิสเตอร์ CCPRxL ร่วมกับบิต 5-4 ของรีจิสเตอร์ CCPxCON

3. กำหนดให้ขาพอร์ท RC2/CCP1 หรือ RC1/T1OSI/CCP2 เป็นเอาท์พุตเพื่อเป็นทางออกของสัญญาณ PWM โดยการเคลียบิตที่ 2 หรือ 1 ของรีจิสเตอร์ TRISC

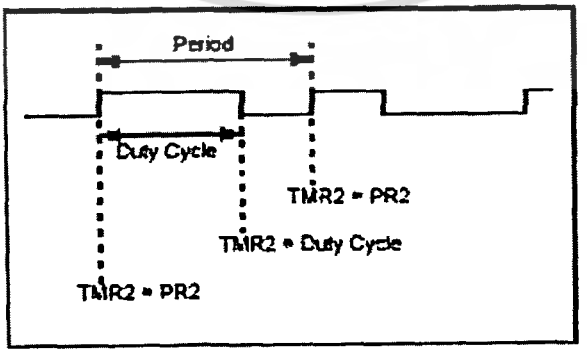
4. กำหนดค่าปรีสเกลเลอร์ของ TMR2 และเอนเนเบิลการทำงานของ ไทเมอร์ 2

5. กำหนดให้โมดูล CCPx ทำงานให้โหมด PWM



รูปที่2.13 การทำงานของโมดูล CCP1 เพื่อสร้างสัญญาณ PWM

จากรูปที่ 2.22 และ รูปที่ 2.23 ทันทีที่โมดูล CCPx เริ่มทำงนค่าของ TMR2 จะเพิ่มขึ้นจนเท่ากับ PR2 ที่ขาพอร์ท RC2/CCP1 หรือ RC1/T1OSI/CCP2 จะเกิดลอจิก "1" และคงสถานะอยู่เช่นนั้นแล้วค่าของ TMR2 จะเคลียร์และเริ่มนับเพิ่มขึ้นใหม่ หลังจากนั้นค่าดิวตี้ไซเกิล ที่กำหนดไว้ในรีจิสเตอร์ CCPRxL และ 2 บิตใน CCPxCON จะถูกถ่ายทอดไปยัง CCPRxH และ 2 บิตในหน่วยความจำพิเศษเพื่อเปรียบเทียบกับค่าใน TMR2 และค่าปรีสเกลเลอร์ 2 บิต ซึ่งเพิ่มค่าขึ้นเรื่อยๆจนกระทั่งเมื่อข้อมูลทั้ง 2 กลุ่มเท่ากัน จะส่งสัญญาณไป ทำให้ขาพอร์ต RC2/CCP1 หรือ RC1/T1OSI/CCP2 กลับมาเป็นลอจิก "0" และคงสถานะอยู่เช่นนั้นจนกระทั่งค่าของ TMR2 เท่ากับ PR2 อีกครั้ง ก็จะเกิดสัญญาณลอจิก "1" เป็นการเริ่มต้นรอบใหม่ของสัญญาณ และจะทำงานวนเช่นนี้จนกระทั่งมีการดิสเอเบิ้ล



รูปที่2.14 เอาต์พุตของสัญญาณ PWM

ค่าดิวตี้ไซเกิลของสัญญาณ PWM ที่สร้างขึ้นนี้สามารถเปลี่ยนแปลงได้ตลอดเวลา จากการเปลี่ยนค่าที่ CCPRxL และ 2 บิตใน CCPxCON แต่ค่าของคาบเวลาหรือความถี่จะเปลี่ยนแปลงไม่ได้จนกว่าจะหยุดการทำงาน แล้วกำหนดค่าของคาบเวลาลงในรีจิสเตอร์ PR2 ใหม่ ดังนั้นสัญญาณ PWM ที่สร้างขึ้นจึงสามารถกำหนดค่าดิวตี้ไซเกิลได้ตามความต้องการ

2.3.9.2 การกำหนดคาบเวลาของสัญญาณ PWM ทำได้โดยการเขียนข้อมูลลงในรีจิสเตอร์ PR2 (รีจิสเตอร์คาบเวลาของไทเมอร์ 2) แล้วนำค่าของ PR2 มาคำนวณหาค่าคาบเวลาของสัญญาณ PWM

คาบเวลาของสัญญาณ PWM = (ค่าในรีจิสเตอร์ PR + 1) x 4 x T_{OSC} x ค่าปรีสเกลเลอร์ของ TMR2

โดยที่ T_{OSC} คือ คาบเวลาของสัญญาณนาฬิกาหลักมีหน่วยเป็นวินาที
ค่าในรีจิสเตอร์ทั้งหมดคำนวณในรูปของเลขฐานสิบ

ความถี่ของสัญญาณ PWM = 1/คาบเวลาของสัญญาณ PWM

2.3.9.3 การกำหนดค่าดิวตี้ไซเกิลของสัญญาณ PWM ทำได้โดยการเขียนข้อมูลไปยังรีจิสเตอร์ CCPRxL ร่วมกับบิต 5-4 ของรีจิสเตอร์ CCPxCON ทำให้สามารถกำหนดความละเอียดของสัญญาณ PWM ได้สูงสุดถึง 10 บิต โดย 8 บิต บนจะใช้ข้อมูลในรีจิสเตอร์ CCPRxL ส่วนใน 2 บิต ล่างใช้ข้อมูลในบิต 5-4 ของรีจิสเตอร์ CCP1CON

ดิวตี้ไซเกิลของสัญญาณ PWM = (CCPR: CCP1CON<5:4>)$_{10bit}$ x T_{OSC}

โดยที่ T_{OSC} คือ คาบเวลาของสัญญาณนาฬิกาหลักมีหน่วยเป็นวินาที
ค่าในรีจิสเตอร์ทั้งหมดคำนวณในรูปของเลขฐานสิบ

2.3.9.4 ความละเอียดของสัญญาณ PWM จะขึ้นอยู่กับบิตปรีสเกลเลอร์ของไทเมอร์ 2 ในขณะที่ความละเอียดสูงสุดของสัญญาณ PWM จะสัมพันธ์กับความถี่ที่กำหนด ความถี่ของสัญญาณนาฬิกา และค่าปรีสเกลเลอร์

$$\text{ความละเอียด (บิต)} = \frac{\log(\frac{f_{OSC}}{f_{PWM}xN})}{\log 2}$$

โดยที่ f_{OSC} คือ ความถี่ของสัญญาณนาฬิกาหลักมีหน่วยเป็น Hz
f_{PWM} คือ ความถี่ของสัญญาณ PWM มีหน่วยเป็น Hz

N คือ ค่าปรีสเกลเลอร์ในไทเมอร์ 2

สัญญาณ PWM จะมีความละเอียดสูงสุดเมื่อค่าของปรีสเกลเลอร์เท่ากับ 1

ตารางที่ 2.4 ตัวอย่างขนาดความถี่และความละเอียดของสัญญาณ PWM ที่ความถี่ 20 MHz

ความถี่ของสัญญาณ PWM (kHz)	**1.22**	**4.88**	**19.53**	**78.12**	**156.3**	**208.3**
ค่าปรีสเกลเลอร์ (1, 4, 16)	16	4	1	1	1	1
ค่ารีจิสเตอร์ PR2	FFh	FFh	FFh	3Fh	1Fh	17h
ความละเอียดสูงสุด (บิต)	10	10	10	8	7	5.5

บทที่ 3
การคำนวณและการสร้าง

3.1 การศึกษาค้นคว้า

สำหรับการศึกษาค้นคว้าเพื่อที่จะทำการทดลองและสร้างตัวควบคุมการเปิด-ปิดวาล์วแบบอัตโนมัตินั้น ทางคณะผู้จัดทำได้ทำการศึกษาทั้งจากการวิจัยหรือศึกษาจากเอกสารต่างๆที่เกี่ยวข้องกับทุกๆส่วนประกอบในตัวระบบ และได้ศึกษาจากการทดลองการทำงานจริงควบคู่ไปเพื่อให้ได้ระบบที่มีเสถียรภาพมากที่สุด โดยการศึกษาค้นคว้ามีรายละเอียดดังนี้

3.1.1 การวิจัยหรือศึกษาจากเอกสารต่างๆที่เกี่ยวข้อง

ในการดำเนินการในส่วนนี้ ได้มีการค้นคว้าศึกษาทั้งในหนังสือ เอกสารต่างๆ และค้นคว้าทางเว็บไซต์ โดยศึกษาในด้านหลักการทำงานของส่วนประกอบทั้งหมดในระบบที่เกี่ยวข้อง คือ เกทวาล์ว จอแสดงผลแอลซีดี ไมโครคอนโทรลเลอร์PIC16F877 ศึกษาเกี่ยวกับความสัมพันธ์ต่างๆและวงจรอิเล็กทรอนิกส์ที่เกี่ยวข้องกับระบบเพื่อที่จะทำการลงมือปฏิบัติจริง รวมทั้งได้ศึกษาในส่วนของโปรแกรมที่จะนำมาใช้ควบคุมส่วนต่างๆ จากหนังสือที่เกี่ยวข้อง

3.1.2 การวิจัยทดลอง

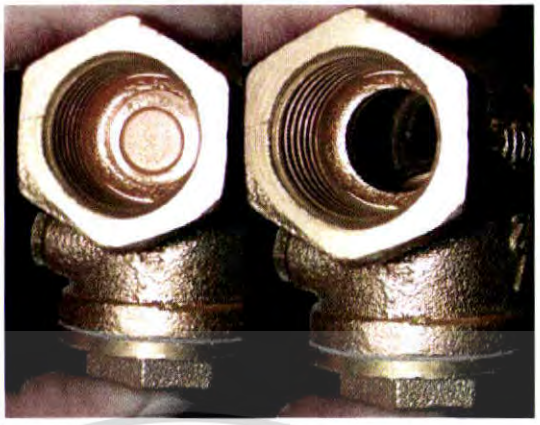
สำหรับการศึกษาดำเนินงานในส่วนนี้นั้น เป็นการศึกษาควบคู่กับการทดลองปฏิบัติจริง ซึ่งการทดลองนั้นอาจจะมีความผิดพลาดหรือไม่ผิดพลาดก็เป็นได้ ดังนั้นจึงได้มีการศึกษาในส่วนนี้ควบคู่กับการทดลองเพื่อจะได้ดำเนินการแก้ไขปรับปรุงในส่วนที่ผิดภาพ เพื่อให้ได้ผลการทดลองที่มีเสถียรภาพมากที่สุด

3.2 ส่วนประกอบและหลักการเลือก

3.2.1 วาล์ว

วาล์วที่เลือกให้เป็นวาล์วชนิด เกทวาล์ว (Gate Valve) ขนาด 6 หุล หรือ ¼ นิ้ว ซึ่งเป็นวาล์วที่มีลักษณะ คือ ตัวถังวาล์วจะเป็นเหล็กทองเหลือง ส่วนหัววาล์วถอดออกได้ และ หัวหมุนได้ 360 องศาตัวลิ่มเปิด-ปิดขึ้นลงโดยที่การเปิด-ปิดแต่ละครั้งนั้น ต้องหมุนวาล์ว 5 รอบ จึงจะเปิด-ปิดได้สนิท โดยลักษณะของวาล์วที่ใช้จะเป็นดังรูปที่3.1 และ รูปที่3.2

รูปที่3.1 ลักษณะของเกทวาล์วที่ใช้



รูปที่3.2 ลักษณะของวาล์วเมื่อ ปิด-เปิด จนสุด

สาเหตุที่เลือกใช้เกทวาล์ว

- เป็นวาล์วที่มีค่าความฝืดเหมาะสมและสอดคล้องกับตัวมอเตอร์ที่ใช้ โดยไม่มากหรือน้อยจนเกินไป
- สามารถควบคุมการเปิด-ปิดคิดเป็นเปอร์เซ็นต์ได้ละเอียดกว่าวาล์วอีกหลายๆชนิด เนื่องจากเป็นวาล์วที่หมุนได้ 360 องศา และต้องหมุนถึง 5 รอบจึงจะเปิด-ปิดจนสุด ซึ่งจะทำให้เกิดค่าผิดพลาดที่น้อยหากเปรียบเทียบกับวาล์วชนิดอื่นๆ
- เป็นอีกหนึ่งทางเลือกใหม่ในการศึกษา เนื่องจากว่าเป็นการได้ทดลองใช้วาล์วชนิดอื่นๆ ซึ่งแตกต่างจากในโรงงานอุตสาหกรรม เพื่อทำให้เป็นอีกหนึ่งทางเลือกต่อไป

3.2.2 เซอร์โวมอเตอร์

รูปที่ 3.3 แสดงลักษณะทั่วไปของเซอร์โวมอเตอร์

ลักษณะทั่วไป DC Servo Motor ดังรูปที่3.3 เป็นมอเตอร์ไฟฟ้ากระแสตรง ที่ถูกประกอบรวมกับ ชุดเกียร์ และ ส่วนควบคุม ต่างๆ ไว้ในโมดูลเดียวกัน หรือภายในกล่องพลาสติกเดียวกัน โดยมอเตอร์ชนิดนี้จะมีสายต่อใช้งาน 3 สายด้วยกัน คือ Vcc, GND และ สายสัญญาณควบคุม (Control Line) ซึ่งสามารถควบคุมให้มอเตอร์หมุนซ้าย หรือ ขวาได้จากสายสัญญาณ

เพียงสายเดียว ปกติไม่สามารถหมุนรอบ 360 องศาได้ ต้องทำการปรับแต่ง (Modify) ดัดแปลงชิ้นส่วนบางอย่างของมอเตอร์ อนึ่ง โครงงานนี้ต้องการให้หมุนรอบ 360 องศา เพราะฉะนั้นจึงต้องมีการปรับแต่งมอเตอร์

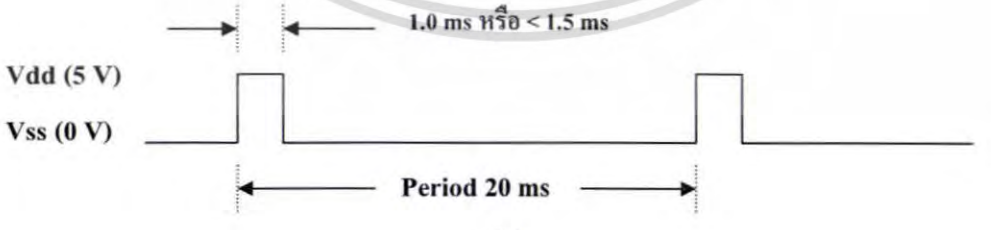
Servo Motor ที่นำมาใช้เป็นรุ่น S03T 2 Ball Bearings

- Speed ที่ 4.8 V เท่ากับ 0.33 sec/60 degrees และ ที่ 6 V เท่ากับ 0.27 sec/60 degrees
- Torque ที่ 4.8 V เท่ากับ 7.20 kg-cm และที่ 6 V เท่ากับ 8.00 kg-cm

รูปที่3.4 แสดง การปรับแต่งเซอร์โวมอเตอร์

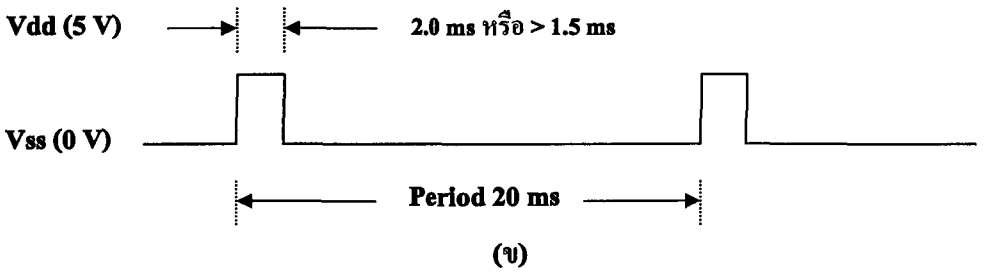
การปรับแต่ง DC Servo Motor ดังรูปที่3.4

1. การต่อตัวต้านทานคงที่ 2 ตัวอนุกรม แทนตัวต้านทานปรับค่าได้
2. ตัดชิ้นส่วนของแกนเฟืองที่ทำหน้าที่หยุดมอเตอร์ (Tab Stop) ออก
3. การดัดแปลงตัวต้านทานปรับค่าได้ (VR) ให้สามารถหมุนได้รอบทิศทาง (360 องศา)



(ก)

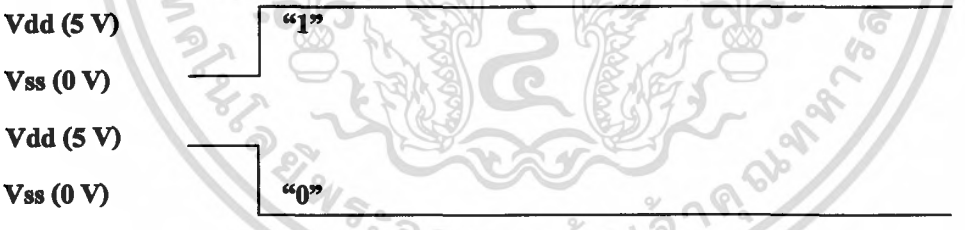
รูปที่3.5 พัลส์การหมุน



รูปที่3.5 พัลส์การหมุน (ต่อ)

หลักการควบคุม DC Servo Motor หลังจากปรับแต่งแล้ว

1. การควบคุมให้มอเตอร์หมุนทางด้านซ้าย จะต้องป้อนสัญญาณพัลส์ที่ มีขนาดความกว้างพัลส์ 1 ms หรือ ให้น้อยกว่า 1.5 ms โดยจะต้องป้อนสัญญาณพัลส์นี้ทุกๆ 20 ms ดังรูปที่ 3.5(ก)(หรือในช่วงประมาณ 20 ms – 30 ms)
2. การควบคุมให้มอเตอร์หมุนทางด้านขวา จะต้องป้อนสัญญาณพัลส์ที่ มีขนาดความกว้างพัลส์ 2 ms หรือไม่ต่ำกว่า 1.5 ms โดยจะต้องป้อนสัญญาณพัลส์นี้ทุกๆ 20 ms ดังรูปที่ 3.5(ข) (หรือในช่วงประมาณ 20 ms – 30 ms)
3. การควบคุมให้มอเตอร์หยุดหมุน ทำได้โดยการส่งลอจิก “0” หรือ “1” ให้กับมอเตอร์ หรือการไม่จ่ายสัญญาณพัลส์ให้กับมอเตอร์

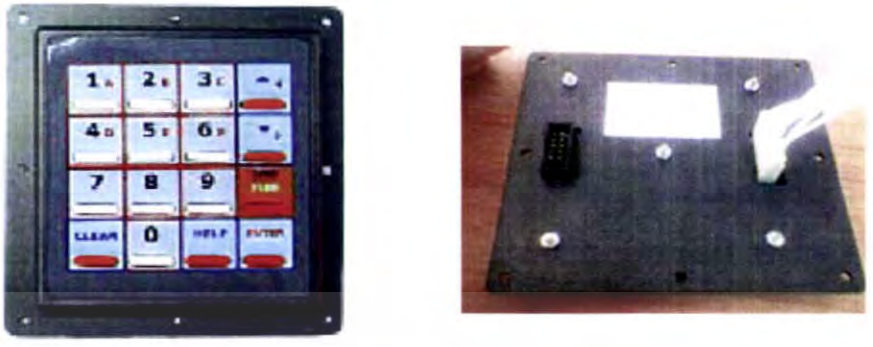


รูปที่3.6 พัลส์การหยุดเซอร์โวมอเตอร์

สาเหตุที่เลือกใช้ Servo Motor

- เนื่องจาก Servo Motor สามารถนำมาปรับแต่งให้เหมาะสมกับชิ้นงานที่เราทำได้ และมีกำลังในการขับวาล์วได้มาก
- มีสาย GND, Vcc และ Signal แยกจากกันชัดเจน ทำให้ง่ายต่อการติดตั้ง
- สามารถหมุนได้ 360 องศา หากมีการปรับแต่ง ซึ่งตรงกับวาล์วที่ใช้ซึ่งหมุนได้ 360 องศา
- มีหลักในการป้อนพัลส์ที่ชัดเจน ทำให้สามารถเขียนโปรแกรมได้ง่ายขึ้น

3.2.3 แป้นพิมพ์ (Key pad)



รูปที่3.7 แสดงภาพ แป้นพิมพ์ ขนาด 4x4

ลักษณะทั่วไป ดังรูปที่3.5 เป็นแป้นพิมพ์ที่มีชื่อรุ่นคือ ET-KEY BOARD 4x4 (P-ET-A-00090) เป็น Keyboard SW ขนาด 4x4 ตัวแป้นพิมพ์ทำด้วยพลาสติกอย่างดีสีดำ คีย์กดเป็นสีขาวและสีส้ม ขนาดคีย์แบบ 16 คีย์ จัดเป็นรูปแบบคีย์ 4x4 Matrix Key มีขั้นต่อทั้งหมด 8 Pin ซึ่งแป้นพิมพ์มีขนาด 10.5 x 11 x 1.1 cm.

สาเหตุที่เลือกใช้แป้นพิมพ์รุ่นนี้

- เนื่องจากเป็นแป้นพิมพ์ขนาด 4X4 ทำให้มีจำนวนคีย์กดที่เพียงพอสำหรับฟังก์ชันที่ใช้
- มีความแข็งแรงทนทาน สามารถใช้งานได้ยาวนาน

3.2.4 จอแสดงผล (LCD)

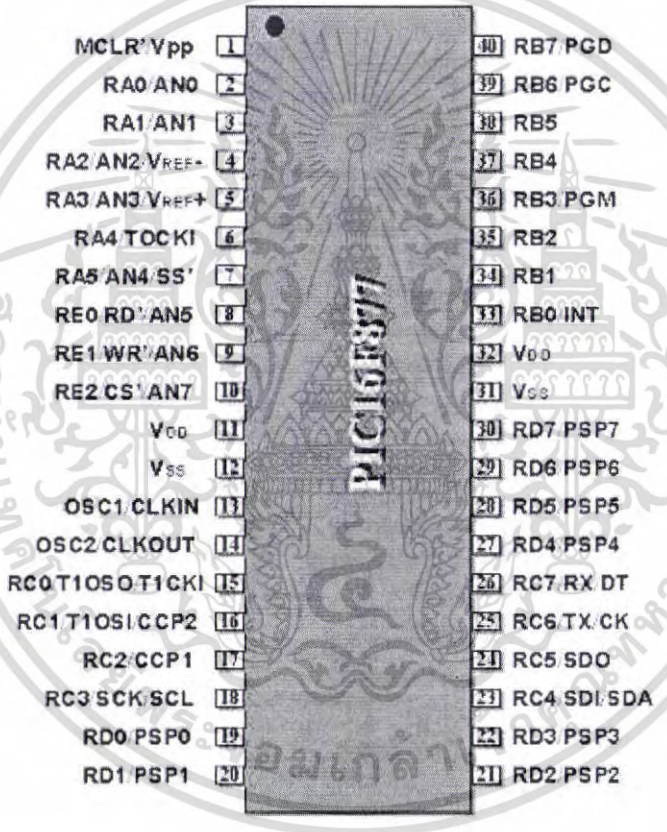
รูปที่3.8 แสดงภาพ จอแสดงผล (LCD) ขนาด 16 ตัวอักษร 1 บรรทัด

ลักษณะทั่วไป ดังรูป3.6 เป็นจอแสดงผล ขนาด 16 ตัวอักษร 1 บรรทัด (16 Characters 1 Line) เป็นแบบ Dot Matrix LCD Module สามารถแสดงตัวเลขหรือตัวอักษรได้ มีชุด CPU Control ในตัวไม่จำเป็นต้องให้ CPU จากบอร์ดมาควบคุมตลอดเวลา มีส่วน Character Generator อยู่ในตัวเป็นภาษาอังกฤษตัวอักษรขนาด 5x7 DOT

สาเหตุที่เลือกใช้จอแสดงผลรุ่นนี้

- เนื่องจากต้องการแสดงผลเพียงตัวอักษรไม่กี่ตัว จึงเหมาะกับการใช้จอแสดงผลที่มีขนาด 1 บรรทัด
- จอแสดงผลนี้มี CPU Control ในตัวไม่จำเป็นต้องให้ CPU จากบอร์ดมาควบคุมตลอดเวลา ทำให้ไม่ต้อง เพิ่มตัวควบคุมจอแสดงผลในวงจรอีก

3.2.5 PIC 16F877

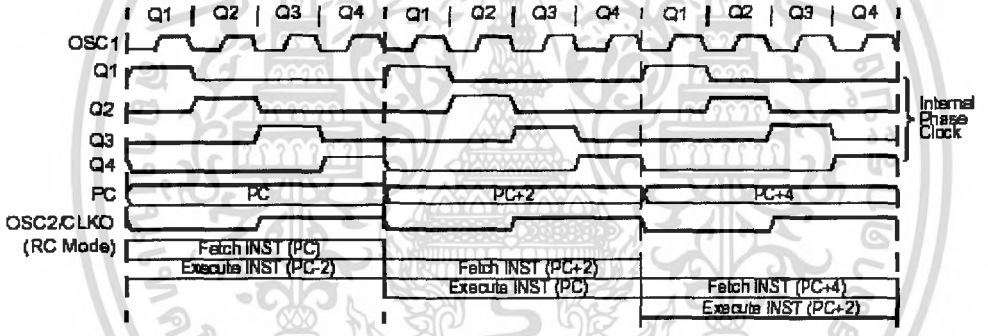


รูปที่3.9 ลักษณะของ PIC16F877

ไมโครคอนโทรลเลอร์ที่ใช้โดยทั่วไป พื้นฐานการทำงานของไมโครคอนโทรลเลอร์ก็ คือ ระบบดิจิตอล โดยค่าเอาท์พุตที่ได้จากไมโครคอนโทรลเลอร์จะเป็น 0 กับ 1 แต่ก็สามารถนำมาประยุกต์เชื่อมต่อกับอุปกรณ์ภายนอกต่างๆมากมาย

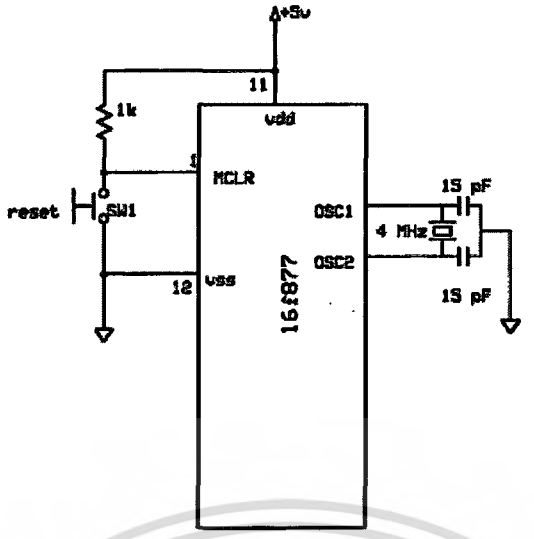
ลักษณะโดยทั่วไปจากรูปที่ 3.7 จากรูปขาของ PIC16F877 แต่ละขาจะมีหน้าที่แตกต่างกันไปซึ่งแยกออกเป็น PORT A , PORT B , PORT C , PORT D, PORT E โดยพื้นฐานแล้วพอร์ตแต่ละพอร์ตสามารถทำงานเป็นอินพุตและเอาท์พุตเป็นดิจิตอล ยกเว้น PORT A และ PORT E ที่สามารถทำงานเป็นตัวรับ สัญญาณอนาลอกแปลงเป็นค่าดิจิตอลเพื่อนำมาวัดปริมาณทางฟิสิกส์ต่างๆ ที่เห็นได้อย่างชัดเจน คือ นำมาวัดความต่างศักย์

สำหรับไมโครคอนโทรลเลอร์ที่ใช้นี้จะต้องมีสัญญาณนาฬิกาภายในตัวเพื่อสร้างพัลซ์มาป้อนให้กับมอเตอร์เพื่อให้มอเตอร์ทำงานตามต้องการซึ่งใน PIC16F877 นี้มีสัญญาณนาฬิกาภายในตัวอยู่แล้วโดยที่หนึ่งไซเคิล (Clock Bus) ของซีพียูจะประกอบไปด้วย สัญญาณนาฬิกาภายนอกจำนวน 4 ไซเคิล คือ Q1 , Q2 , Q3 และ Q4 ตามรูปที่ 3.8 ดังนั้นความความถี่ที่ซีพียูประมวลผลต่อหนึ่งคำสั่งเท่ากับความถี่ของสัญญาณนาฬิกาภายนอกหารด้วย 4 หรือหากจะพิจารณาความเร็วของไมโครคอนโทรลเลอร์ตระกูล PIC สามารถประมวล ผลต่อหนึ่งคำสั่งเท่ากับ 1/4 เท่าของความถี่ออสซิเลเตอร์ภายนอก



รูปที่ 3.10 สัญญาณนาฬิกาที่ออกมาจาก PIC16F877

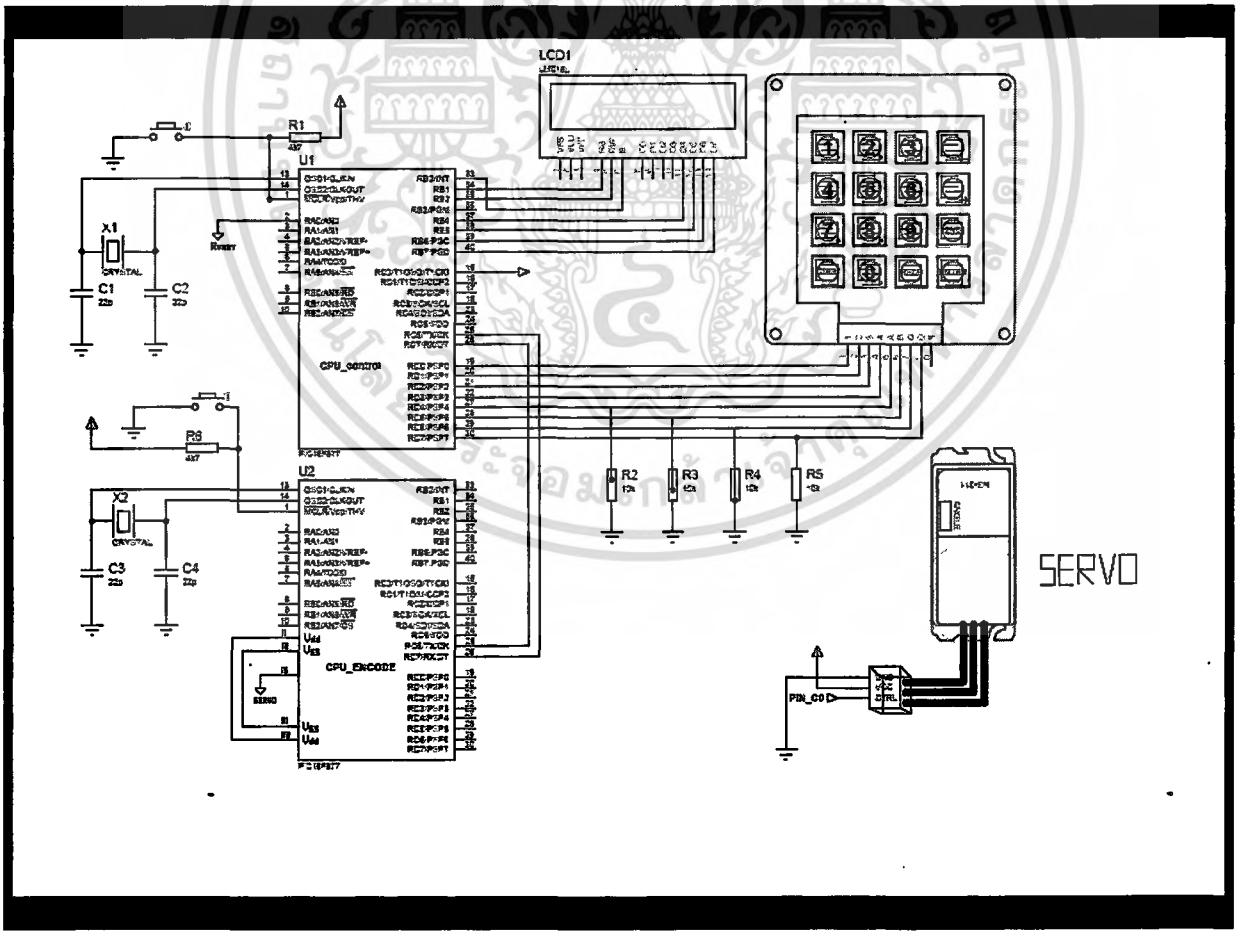
วงจรดังรูปที่ 3.9 เป็นวงจรพื้นฐานที่ต้องต่อทุกครั้งเพื่อให้ไมโครคอนโทรลเลอร์ทำงาน ส่วนขาที่เหลือจะนำมาเชื่อมต่อกับ อุปกรณ์ภายนอกเพื่อควบคุมตามที่ต้องการได้



รูปที่ 3.11 วงจรพื้นฐานที่ต่อเพื่อให้ PIC16F877 ทำงาน

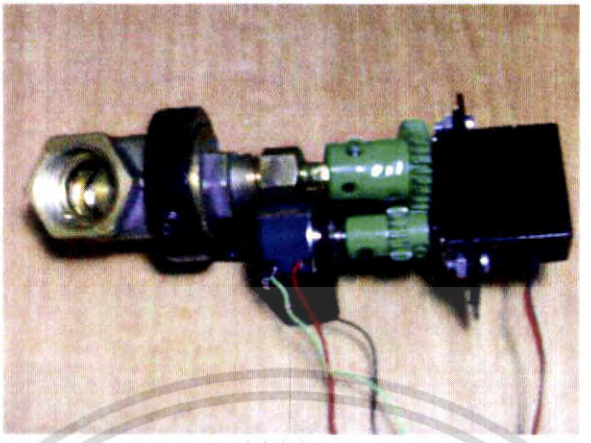
3.3 ขั้นตอนในการสร้าง

3.3.1) สร้างแผงวงจรดังรูปที่ 3.12 เพื่อควบคุมการทำงานทั้งหมดของคอนโทรลวาล์ว



รูปที่3.12 แสดงแผงวงจรควบคุม

3.3.2) ออกแบบและประกอบตัววาล์วเข้ากับเฟือง ดังรูปที่3.13 โดยเฟืองที่ใช้มีอัตราการทดเฟืองที่ 1 ต่อ 2



รูปที่ 3.13 แสดงวาล์วที่คัปปิ้ง(Coupling) แล้ว

3.3.3) นำวาล์วพร้อมตัวเฟืองที่ประกอบแล้ว มาต่อเข้ากับเซอร์โวมอเตอร์และตัวต้านทานปรับค่าได้ โดยต้องนำตัวต้านทานปรับค่าได้มาคัปปิ้งให้ติดกับเฟือง โดยตัวต้านทานที่ปรับค่าได้เป็นตัวจับตำแหน่งของวาล์ว เพื่อส่งข้อมูลไปยังไมโครคอนโทรลเลอร์ว่าวาล์วเปิด-ปิดไปกี่เปอร์เซ็นต์แล้วแสดงค่าที่จอแสดงผลแอลซีดี (LCD) ดังรูปที่3.14

รูปที่ 3.14 แสดงจอแอลซีดีซึ่งป้อนค่าเข้าไปแล้ว

3.4 หลักการทำงาน

3.4.1 หลักการทำงานของไมโครคอนโทรลเลอร์

U1 คือ ตัวควบคุม โปรแกรม ทำหน้าที่ รับค่า ที่แปลงรหัส จาก ตัวต้านทานปรับค่าได้ เพื่อ ควบคุม การเปิด – ปิด ของ วาล์ว โดยจะบอกเป็นช่วงเปอร์เซ็นต์ (%) ตั้งแต่ 0 – 100 %

U2 คือ ตัว ควบคุมเซอร์โวมอเตอร์ ทำหน้าที่ เปิด – ปิด เซอร์โวมอเตอร์

โดยมีขั้นตอนการทำงาน ดังนี้

1. เลือก สถานะ ของ วาล์ว ว่าจะ ให้

   วาล์ว เปิด (Valve = on)

   วาล์ว ปิด (Valve = off)

   จากการกด KEY ขึ้น หรือ ลง จากนั้น ทำการ กด ENTER
2. ตั้งค่า (%) เพื่อ เปิด – ปิด วาล์ว โดย ป้อนค่าได้ 0 – 100 (%) จากนั้น กด ENTER
3. U1 ทำการ ส่ง Protocol ไปให้ U2 เพื่อ ควบคุม ให้ U2 ทำการขับเซอร์โวมอเตอร์
   - Protocol "A" สั่ง ให้วาล์ว on
   - Protocol "B" สั่ง ให้วาล์ว off
4. เมื่อเซอร์โวมอเตอร์ ทำงาน U1 ทำการแปลงค่ารหัส ค่า (%) เปิด – ปิด ของ วาล์ว เมื่อ ค่าที่ทำการแปลงรหัสแล้ว (%) มีค่า มากกว่า ค่าที่ตั้ง (set)ไว้ U1 จะส่ง Protocol "e" ไป U2 เพื่อทำการสั่งให้เซอร์โวมอเตอร์หยุดหมุน

3.4.2 หลักการควบคุมเซอร์โวมอเตอร์

เนื่องจากช่วงแรงดันของเซอร์โวมอเตอร์อยู่ระหว่าง 0-5 โวลต์ และวาล์ว หมุน 100 เปอร์เซ็นต์ เท่ากับ 5 รอบ เฟืองทดมีอัตราทดเท่ากับ 1 ต่อ 2 เพราะฉะนั้นเซอร์โวมอเตอร์จะหมุนได้ 10 รอบ จึงจะเท่ากับ 100 เปอร์เซ็นต์ โดยความสัมพันธ์ของจำนวนซี่ เปอร์เซ็นต์ และแรงดัน แสดงดัง ตารางที่ 3.1

ตารางที่ 3.1 แสดงตารางเปรียบเทียบในการควบคุมเซอร์โวมอเตอร์

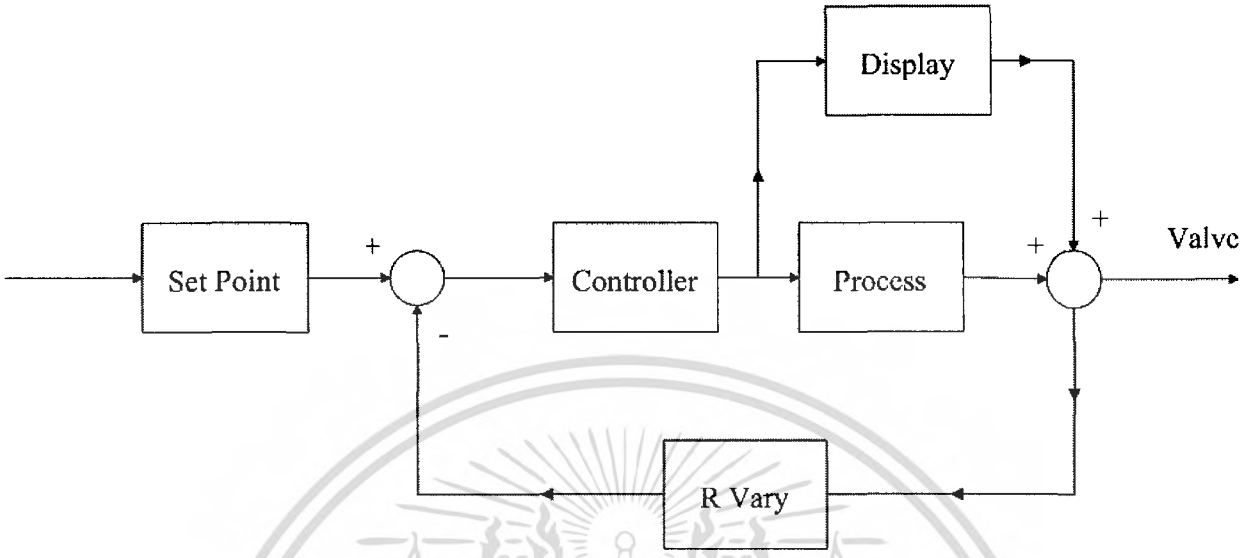
จำนวนซี่	v	volt
0	0	0
1	0.67	0.034
2	1.34	0.068
3	2.01	0.102
4	2.68	0.136
5	3.35	0.17
6	4.02	0.204
7	4.69	0.238
8	5.36	0.272
9	6.03	0.306
10	6.7	0.34
11	7.37	0.374
12	8.04	0.408
13	8.71	0.442
14	9.38	0.476
15	10.05	0.51

เมื่อได้ค่าตารางที่ 3.1 แล้ว ก็สามารถสร้างตารางเปรียบเทียบจาก 0-100 % ได้ ดังตารางที่ 3.2

ตารางที่ 3.2 แสดงตารางเปรียบเทียบในการควบคุมเซอร์โวมอเตอร์จากช่วง 0-100 %

%	volt	จำนวนซี่	จำนวนรอบ
0	0	0	0
10	0.5	15	1
20	1	30	2
30	1.5	45	3
40	2	60	4
50	2.5	75	5
60	3	90	6
70	3.5	105	7
80	4	120	8
90	4.5	135	9
100	5	150	10

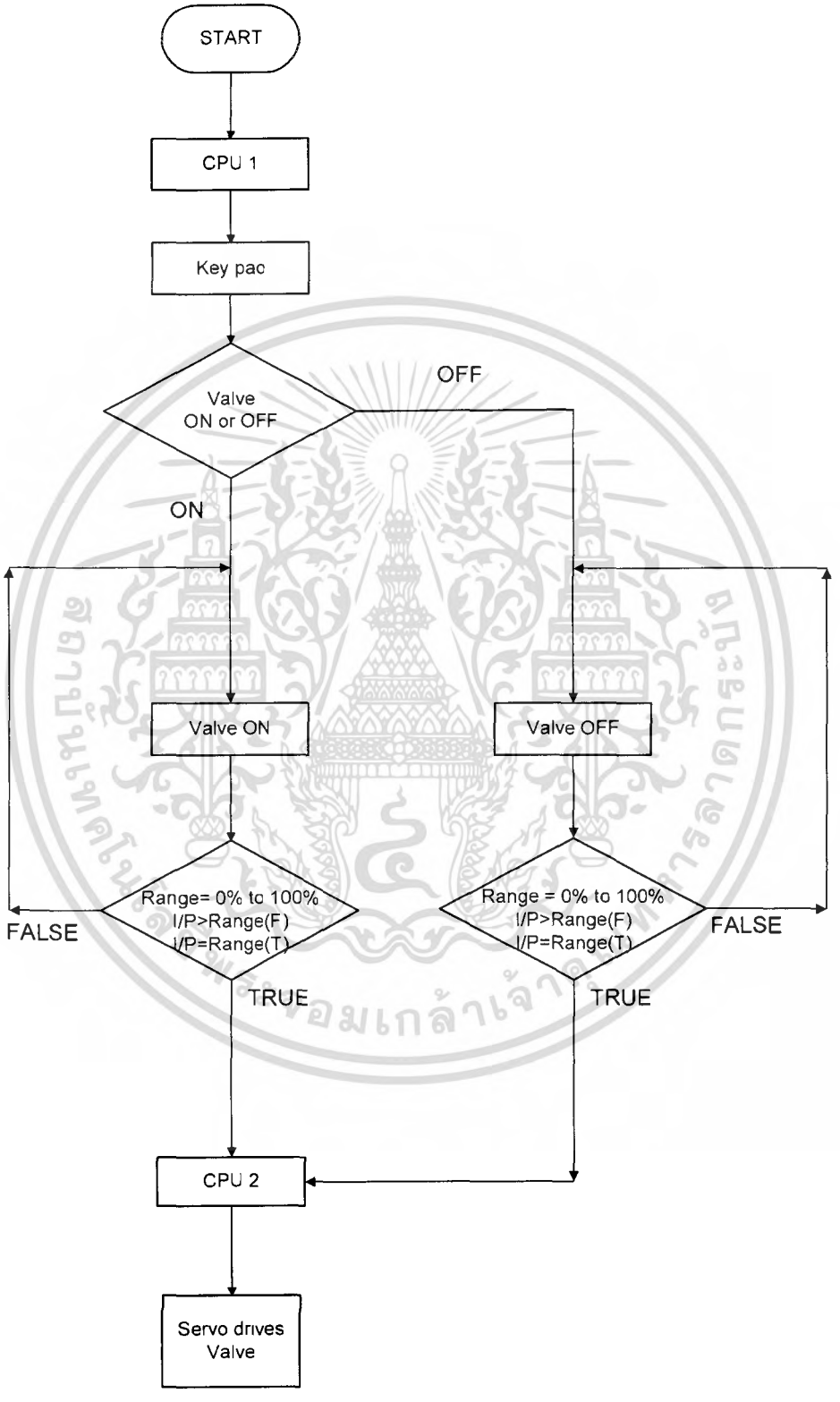
3.5 บล็อกไดอะแกรมของระบบ



รูปที่3.15 แสดงบล็อกไดอะแกรม

รับค่า Set Point ซึ่งก็คือค่าที่เราป้อนเข้าไปในแป้นพิมพ์ แล้วส่งไปยังตัวควบคุม ซึ่งตัวควบคุมจะมี Output 2 ตัว คือ ตัวแสดงหน้าจอแอลซีดี เป็นตัวแสดงค่าเปอร์เซ็นต์ที่เราป้อนเข้าไป ส่วนอีกตัวหนึ่ง เข้าไปในกระบวนการ ซึ่งก็คือเซอร์โวมอเตอร์นั่นเอง และเซอร์โวมอเตอร์ก็ไปขับวาล์วอีกทีหนึ่ง จากนั้นก็มีการป้อนกลับโดยตัวต้านทานปรับค่าได้ ซึ่งคัปปิ้ง (Coupling) ติดกับวาล์ว ว่าขณะนั้นตำแหน่งของวาล์วอยู่ที่กี่เปอร์เซ็นต์แล้ว

3.6 แผนผังการทำงาน (Flow Chart)



รูปที่3.16 แสดงแผนผังการทำงาน

บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดลองและผลการทดลอง วงจรและภาคขับมอเตอร์ และการทำงานของวาล์วควบคุมการเปิด-ปิดแบบอัตโนมัติ โดยมีรายละเอียดของการทดลองดังนี้

4.1 การทดลองวงจรและภาคขับมอเตอร์

ในส่วนนี้เป็นการศึกษาเฉพาะส่วนวงจรและภาคขับมอเตอร์ โดยเริ่มจากการนำวงจรที่สร้างขึ้นมาโปรแกรมเข้าไปในตัวไมโครคอนโทรลเลอร์ เพื่อปล่อยสัญญาณพัลซ์ (Pulse) มาสู่ภาคการขับมอเตอร์ ซึ่งเราจะสามารถกำหนดพัลซ์ ค่าต่างๆ เพื่อที่จะดูว่ามอเตอร์หมุนไปทางไหน เพื่อนำมาปรับปรุงแก้ไขให้ได้การหมุนตามต้องการ จากนั้นนำก็โปรแกรมเข้าวงจรเพื่อทดลองภาคมอเตอร์ที่ประกอบเข้ากับตัววาล์วและเฟืองแล้วมาดูว่ามอเตอร์ทำงานได้อย่างราบรื่นหรือมีข้อผิดพลาดอย่างไรเพื่อจะได้นำมาแก้ไขต่อไป โดยผลการทดลองการโปรแกรมป้อนค่าพัลซ์เข้าภาคมอเตอร์จะได้ผลดังนี้

ตารางที่ 4.1 ตารางแสดงผลการป้อนสัญญาณพัลส์

Pulse Input (ms)	Motor Action	Reaction
1.0	Turn left	Not smooth
1.1	Turn left	Not smooth
1.2	Turn left	Not smooth
1.3	Turn left	smooth
1.4	Turn left	Not smooth
1.5	stop	stop
1.6	Turn right	Not smooth
1.7	Turn right	Not smooth
1.8	Turn right	Not smooth
1.9	Turn right	smooth
2.0	Turn right	Not smooth

4.2 การทดลองการป้อนค่าและแสดงผล

ในการทดลองส่วนนี้ เป็นการทดลองโดยการเขียนโปรแกรมเพื่อสร้างความสัมพันธ์ระหว่างแป้นพิมพ์ (Key-Pad) และจอแสดงผลแอลซีดี (LCD) ผ่านวงจรหลัก ซึ่งในส่วนของแป้นพิมพ์ก็ได้ทำการสแกนคีย์และโปรแกรมให้เข้าตัวไมโครคอนโทรลเลอร์เพื่อให้ใช้งานได้ตามต้องการ ในส่วนของจอแสดงผลแอลซีดี (LCD) ก็มีการทดสอบว่าเมื่อมีการป้อนค่าจากแป้นพิมพ์แล้ว มีการแสดงค่าที่หน้าจอแสดงผลแอลซีดี (LCD) ได้จริง

4.3 การทดลองสัญญาณป้อนกลับในตัวต้านทานปรับค่าได้

ในการทดลองส่วนนี้ เนื่องจากตัวต้านทานปรับค่าได้มีความสำคัญในการป้อนค่ากลับมาที่อินพุตของตัวควบคุมเพื่อนำมาเปรียบเทียบกับค่าเซ็ตพอยต์ (Set point) เพราะฉะนั้นจึงต้องมีการทดลองหาค่าความต้านทานที่เปลี่ยนไปเพื่อเป็นการบอกตำแหน่งของวาล์วว่าเปิด-ปิดไปเท่าไรแล้ว โดยมีตารางการทดลองดังนี้

ตารางที่ 4.2 ความสัมพันธ์ระหว่างตัวต้านทานปรับค่าได้กับการเปิด-ปิดวาล์ว

จำนวนรอบตัวต้านทานปรับค่าได้	ค่าความต้านทาน (kilo Ohm)	% ที่เปิด-ปิดของวาล์ว	ค่าแรงดัน (Volt)
1	1.05	10.04	0.51
2	2.11	20.12	1.02
3	3.08	30.03	1.53
4	4.03	40.08	2.06
5	5.12	50.15	2.56
6	6.01	60.04	3.02
7	7.05	70.13	3.57
8	8.02	80.07	4.04
9	9.03	90.14	4.51
10	10.13	100.00	5.01

4.4 การเปรียบเทียบค่าระหว่างผลการทดลองกับค่าทางทฤษฎีและค่าความผิดพลาดของอุปกรณ์

ในการทดลองส่วนนี้ จะเป็นการทดลองวัดค่าแรงดันของตัวต้านทานปรับค่าได้ 10 รอบ ของแต่ละ % ที่เปลี่ยนไป เทียบกับค่าแรงดันของตัวต้านทานปรับค่าได้ 10 รอบ ในทางทฤษฎี และความผิดพลาดของอุปกรณ์ โดยมีตารางเปรียบเทียบผลการทดลองกับผลทางทฤษฎี ดังนี้

ตารางที่ 4.3 ตารางเปรียบเทียบผลการทดลองกับผลทางทฤษฎี

% การเปิดปิดวาล์ว	ค่าทางทฤษฎี	ค่าจากการวัดจริง	ค่าความผิดพลาดที่เกิดขึ้น	% ความผิดพลาด
	แรงดันจากตัวต้านทานปรับค่าได้ (mV)	แรงดันจากตัวต้านทานปรับค่าได้ (mV)		
0	5.000	4.954	0.046	0.920
1	4.950	4.954	0.004	0.081
2	4.900	4.953	0.053	1.082
3	4.850	4.952	0.102	2.103
4	4.800	4.805	0.005	0.104
5	4.750	4.703	0.047	0.989
6	4.700	4.647	0.053	1.128
7	4.650	4.613	0.037	0.796
8	4.600	4.546	0.054	1.174
9	4.550	4.494	0.056	1.231
10	4.500	4.452	0.048	1.067
11	4.450	4.391	0.059	1.326
12	4.400	4.354	0.046	1.045
13	4.350	4.296	0.054	1.241
14	4.300	4.246	0.054	1.256
15	4.250	4.205	0.045	1.059
16	4.200	4.143	0.057	1.357
17	4.150	4.107	0.043	1.036
18	4.100	4.042	0.058	1.415
19	4.050	3.991	0.059	1.457

ตารางที่ 4.3 ตารางเปรียบเทียบผลการทดลองกับผลทางทฤษฎี (ต่อ)

% การเปิดปิดวาล์ว	ค่าทางทฤษฎี	ค่าจากการวัดจริง	ค่าความผิดพลาดที่เกิดขึ้น	% ความผิดพลาด
	แรงดันจากตัวต้านทานปรับค่าได้ (mV)	แรงดันจากตัวต้านทานปรับค่าได้ (mV)		
20	4.000	3.942	0.058	1.450
21	3.950	3.891	0.059	1.494
22	3.900	3.841	0.059	1.513
23	3.850	3.804	0.046	1.195
24	3.800	3.756	0.044	1.158
25	3.750	3.693	0.057	1.520
26	3.700	3.643	0.057	1.541
27	3.650	3.595	0.055	1.507
28	3.600	3.557	0.043	1.194
29	3.550	3.502	0.048	1.352
30	3.500	3.450	0.050	1.429
31	3.450	3.403	0.047	1.362
32	3.400	3.342	0.058	1.706
33	3.350	3.292	0.058	1.731
34	3.300	3.241	0.059	1.788
35	3.250	3.202	0.048	1.477
36	3.200	3.144	0.056	1.750
37	3.150	3.093	0.057	1.810
38	3.100	3.043	0.057	1.839
39	3.050	2.994	0.056	1.836
40	3.000	2.952	0.048	1.600
41	2.950	2.901	0.049	1.661
42	2.900	2.847	0.053	1.828
43	2.850	2.808	0.042	1.474

ตารางที่ 4.3 ตารางเปรียบเทียบผลการทดลองกับผลทางทฤษฎี (ต่อ)

% การเปิดปิดวาล์ว	ค่าทางทฤษฎี	ค่าจากการวัดจริง	ค่าความผิดพลาดที่เกิดขึ้น	% ความผิดพลาด
	แรงดันจากตัวต้านทานปรับค่าได้ (mV)	แรงดันจากตัวต้านทานปรับค่าได้ (mV)		
44	2.800	2.753	0.047	1.679
45	2.750	2.692	0.058	2.110
46	2.700	2.646	0.054	2.000
47	2.650	2.582	0.068	2.566
48	2.600	2.555	0.045	1.731
49	2.550	2.501	0.049	1.922
50	2.500	2.454	0.046	1.840
51	2.450	2.402	0.048	1.959
52	2.400	2.353	0.047	1.958
53	2.350	2.301	0.049	2.085
54	2.300	2.265	0.035	1.522
55	2.250	2.207	0.043	1.911
56	2.200	2.152	0.048	2.182
57	2.150	2.101	0.049	2.279
58	2.100	2.060	0.040	1.905
59	2.050	2.010	0.040	1.951
60	2.000	1.961	0.039	1.950
61	1.950	1.913	0.037	1.897
62	1.900	1.864	0.036	1.895
63	1.850	1.802	0.048	2.595
64	1.800	1.765	0.035	1.944
65	1.750	1.703	0.047	2.686
66	1.700	1.656	0.044	2.588
67	1.650	1.602	0.048	2.909

ตารางที่ 4.3 ตารางเปรียบเทียบผลการทดลองกับผลทางทฤษฎี (ต่อ)

% การเปิดปิดวาล์ว	ค่าทางทฤษฎี	ค่าจากการวัดจริง	ค่าความผิดพลาดที่เกิดขึ้น	% ความผิดพลาด
	แรงดันจากตัวต้านทานปรับค่าได้ (mV)	แรงดันจากตัวต้านทานปรับค่าได้ (mV)		
68	1.600	1.567	0.033	2.063
69	1.550	1.522	0.028	1.806
70	1.500	1.464	0.036	2.400
71	1.450	1.413	0.037	2.552
72	1.400	1.362	0.038	2.714
73	1.350	1.311	0.039	2.889
74	1.300	1.268	0.032	2.462
75	1.250	1.205	0.045	3.600
76	1.200	1.152	0.048	4.000
77	1.150	1.104	0.046	4.000
78	1.100	1.063	0.037	3.364
79	1.050	1.024	0.026	2.476
80	1.000	0.962	0.038	3.800
81	0.950	0.911	0.039	4.105
82	0.900	0.866	0.034	3.778
83	0.850	0.827	0.023	2.706
84	0.800	0.762	0.038	4.750
85	0.750	0.721	0.029	3.867
86	0.700	0.666	0.034	4.857
87	0.650	0.616	0.034	5.231
88	0.600	0.561	0.039	6.500
89	0.550	0.524	0.026	4.727
90	0.500	0.483	0.017	3.400

ตารางที่ 4.3 ตารางเปรียบเทียบผลการทดลองกับผลทางทฤษฎี (ต่อ)

% การเปิดปิดวาล์ว	ค่าทางทฤษฎี	ค่าจากการวัดจริง	ค่าความผิดพลาดที่เกิดขึ้น	% ความผิดพลาด
	แรงดันจากตัวต้านทานปรับค่าได้ (mV)	แรงดันจากตัวต้านทานปรับค่าได้ (mV)		
91	0.450	0.435	0.015	3.333
92	0.400	0.383	0.017	4.250
93	0.350	0.316	0.034	9.714
94	0.300	0.278	0.028	9.333
95	0.250	0.217	0.033	13.200
96	0.200	0.163	0.037	18.500
97	0.150	0.132	0.018	12.000
98	0.100	0.081	0.019	19.000
99	0.050	0.047	0.003	6.000
100	0.000	0.000	0.000	0.000

บทที่ 5
บทวิจารณ์และสรุป

5.1 สรุป

หลังจากได้ทำชิ้นโครงงานเสร็จเรียบร้อย พบว่าเมื่อป้อนค่าเปิดวาล์วไปจะมีข้อผิดพลาดอยู่เล็กน้อย กล่าวคือ เมื่อสั่งให้วาล์วเปิด เซอร์โวมอเตอร์จะหมุนกลับทางไปขณะหนึ่ง แล้วจึงหมุนไปตามเดิม ซึ่งเป็นข้อผิดพลาดที่ยอมรับได้ ส่วนอื่นๆ นั้นไม่มีข้อผิดพลาด เมื่อเทียบกับตอนแรกซึ่งใช้เซนเซอร์จับเอาที่ตัวเฟือง ซึ่งให้ค่าที่ไม่แน่นอน ไม่เสถียร ไม่มี ความน่าเชื่อถือ (Reliability) จึงเปลี่ยนมาใช้ ตัวต้านทานปรับค่าได้ 10 รอบ มาเป็นตัวป้อนกลับ (Feedback) เพื่อเปรียบเทียบกับค่าที่ตั้งไว้ (Set Point) แล้วส่งไปให้ตัวควบคุม (Controller) ประมวลผลอีกทีหนึ่ง ซึ่งพอเปลี่ยนมาใช้ตัวต้านทานปรับค่าได้แล้วเซอร์โวมอเตอร์หมุนได้อย่างราบเรียบมากขึ้น

5.2 ปัญหาที่พบและแนวทางแก้ไข

จากการศึกษาและทำโครงงานได้เกิดปัญหาคือ ตัวมอเตอร์เซอร์โว ไม่เสถียร กล่าวคือ พัลส์ที่จ่ายให้กับเซอร์โวมอเตอร์ ซึ่งอยู่ในส่วนโปรแกรมภาษาซี ในส่วนของ open servo ไม่ตรงกับทฤษฎีและไม่แน่นอน เมื่อมีการเคลื่อนย้าย หรือ มีการกระทบกระเทือนแม้เพียงเล็กน้อย ก็ทำให้พัลส์ที่จ่ายให้กับมอเตอร์เปลี่ยนไปจากเดิม ได้ทำการแก้ไข คือ เปลี่ยนตัวเซอร์โวมอเตอร์ตัวใหม่มาใช้ แต่ปัญหาลักษณะเดิมก็ยังคงมีอยู่ และไม่ใช่แค่ส่วนของเซอร์โวมอเตอร์ ส่วนของ จอแสดงผลแอลซีดี และแป้นพิมพ์ ก็มีการผิดพลาดเช่นกัน แนวทางแก้ไข คือ นำตัวไอซีPIC ไมโครคอนโทรลเลอร์ มาเขียนโปรแกรมใส่ใหม่ ซึ่งก็สามารถแก้ปัญหาได้ในระดับหนึ่ง ปัญหานี้สันนิษฐานว่า เป็นความคลาดเคลื่อนเนื่องจากได้รับแรงกลหรือรับแรงกระทบกระเทือนจึงทำให้เซอร์โวมอเตอร์ซึ่งมีวงจรอิเล็กทรอนิกส์ควบคุมอยู่ข้างใน เกิดการคลาดเคลื่อน

เนื่องจากผู้จัดทำไม่มีประสบการณ์ในการคัปปิ้ง (Coupling) ตัวเฟืองทด เซอร์โวมอเตอร์ ตัวต้านทาน 10 รอบ และตัววาล์ว ทำให้ในช่วงแรก ได้ทำการเชื่อมติดตัวต้านทาน 10 รอบ ซึ่งทำให้ไม่สามารถถอดมาแก้ไขเปลี่ยนแปลงได้ ซึ่งผิดหลักทางวิศวกรรมที่ว่า อุปกรณ์ทุกอย่างต้องสามารถถอดออกมาได้ทุกชิ้น เพราะว่าสิ่งของทุกอย่างในโลกย่อมมีเสื่อม จึงต้องมีการเปลี่ยนแปลง เมื่อมีการเปลี่ยนแปลง จึงต้องสามารถถอดเปลี่ยนได้ จึงได้ทำการแก้ไข โดยใช้สกรูเกลียวหนอนมายึดแทนซึ่ง ตัวสกรูเกลียวหนอนสามารถนำประแจหกเหลี่ยมมาไข เข้า-ออก เพื่อถอดได้ เมื่อเปลี่ยนเรียบร้อยแล้ว การเปลี่ยนแปลงแก้ไขตัวโครงงานทำได้อย่างสะดวกมากยิ่งขึ้น

5.3 ข้อเสนอแนะและแนวทางในการค้นคว้าพัฒนา

วาล์วควบคุม (Control Valve) นี้สามารถนำไปค้นคว้าพัฒนาต่อได้ ตัวแปรที่สำคัญในระบบควบคุมอีกตัวหนึ่งคือ การไหล เนื่องจากตัวแปรอื่นๆ ที่ต้องการควบคุมจะถูกควบคุมโดยปริมาณของอัตราการไหลเกือบทั้งสิ้น โดยส่งผลไปที่วาล์วควบคุม ดังนั้น วาล์วควบคุมนี้สามารถนำมาค้นคว้า พัฒนาต่อได้โดยการนำมาวัดอัตราการไหลของของไหลซึ่ง ตัวที่นำมาวัดนั้นก็อาจจะเป็นการจำลองแผ่นออริฟิด (Orifice) ซึ่งเป็นการวัดอัตราไหลแบบวัดความดันที่แตกต่าง และยังจำเป็นต้องเขียนโปรแกรมให้เหมาะสมกับเพื่อเป็นตัวป้อนกลับให้ตัวควบคุมทำหน้าที่ได้อย่างมีประสิทธิภาพ เพื่อที่จะสามารถนำไปประยุกต์ใช้กับอุตสาหกรรมที่ต้องใช้การควบคุมของไหล อาทิเช่น อุตสาหกรรมปิโตรเคมี อุตสาหกรรมอาหาร เป็นต้น

เอกสารอ้างอิง

[1] Philip L. Skousen .**Valve Handbook**. USA: Mcgraw Hill.1998.

[2] ตระการ ก้าวกสิกรรม, พิศักดิ์ เจริญภักดี. **คู่มือการเลือกใช้งานวาล์ว**. กรุงเทพมหานคร: เอ็มแอนด์ดี. 2539.

[3] ประสิทธิ์ จุลเสรีวงศ์. **วิศวกรรมการวัดคุม**. พิมพ์ครั้งที่1. กรุงเทพมหานคร: สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2549.

[4] โยธิน เปรมปราณีรัชต์. **ระบบเซอร์โวและอิเล็กทรอนิกส์คอนโทรลมอเตอร์**. กรุงเทพมหานคร: สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.2544.

[5] กฤษดา ใจเย็น. **สนุกกับไมโครคอนโทรลเลอร์ฉบับ PIC**. กรุงเทพมหานคร: อินโนเวตีฟ เอ็กเพอริเมนต์.2547.

[6] วัชรินทร์ เคารพ. **เรียนรู้และเข้าใจสถาปัตยกรรมไมโครคอนโทรลเลอร์ PIC16F877**. พิมพ์ครั้งที่1. กรุงเทพมหานคร: อีทีที.2547.

ภาคผนวก

ภาคผนวก ก
ส่วนโปรแกรมภาษาซีที่ใช้ในโครงงาน

```
/* code main control */
/* A/D */
#include <16F877.h>
#define TxD PIN_C6
#define RxD PIN_C7
#device ADC=10              // set ADC 10 BIT. //
#fuses HS,NOLVP,NOWDT,NOPROTECT
#use delay (clock=20000000)
#use rs232(baud=9600,xmit=TxD,rcv=RxD)

#define use_portb_lcd       // Set port LCD port b. //
#include <LCD.c>            // open file LCD control. //

#include "math.h"
#use fast_io(A)
#define Vbe = 0.00488758553274682306940371456500049

char data_code[11] = { 0x30,0x31,0x32,0x33,0x34,
                0x35,0x36,0x37,0x38,0x39,0xFE}; // -- ascii code num "0-9" --//

// ---------- MAIN PROGRAMS ---------------------- //
void main(void)
{
    unsigned char num[4];          // value user input. //
    unsigned char position;        // position num. //
    unsigned char aa,bb,cc;
    unsigned char start_servo=0;   // value set mode . //
    unsigned char valve=0;         // value set valve is on/off. //
```

```
    int16 value;
    float volt;
    float sum;
    unsigned char control;
    unsigned char select_valve;

    setup_port_a(ALL_ANALOG);
    setup_adc(ADC_CLOCK_INTERNAL);


    set_tris_d(0b11110000);                 // set port d output_port PIN D0 - D3 and input_port PIN D4 -
D7. //
    lcd_init();          delay_ms(100);     // Inition LCD. //
    lcd_command(0x80);                      // set write LCD upper 8 char. //
    lcd_putc("Control");                   // function printf LCD. //
    lcd_command(0xC0);                      // set write LCD lowwer 8 char. //
    lcd_putc("Value."); delay_ms(2000);     // function printf LCD. //
    lcd_command(0x01);                      // clear screan LCD. //

// ------------------------------ loop while TRUE. ------------------------------- //
    while(TRUE)
    {
      // -------- programs set on valve on or valve. ---------- //
      if(start_servo==0)
      {
        lcd_command(0x01);          // clear screan LCD. //
        lcd_command(0x80);          // set write LCD upper 8 char. //
        lcd_putc("valve = ");       // function printf LCD. //
        while(start_servo==0)
        {
            lcd_command(0xC0);      // set write LCD lowwer 8 char. //
            if(valve==0)
```

```
        { lcd_putc("ON "); }        // function printf LCD. //
      else { lcd_putc("OFF");}      // function printf LCD. //
      // ------------------------------ scan COL 3 ------------------------- //
      output_d(0x08);
      if((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7)))
      {
        if(input(pin_d4))   {  valve++; if(valve==2) valve=0;    }  // ROW_0
        if(input(pin_d5))   {  valve--; if(valve==255) valve=1;  }  // ROW_1
        if(input(pin_d6))   {  }  // ROW_2}
        if(input(pin_d7))
        {
          start_servo=1;
          lcd_command(0x01);  delay_ms(100);       // clear screan. //
        }                                                          // ROW_3
        while((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7))) {delay_us(100);}
      }
      // ------------------------- END SCAN COL 3 ---------------------- //
    }
  }
////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ---------------------------------- programs set input. ------------------------------------------------- //
    if(start_servo==1)
    {
      position=0; aa=10;bb=10;cc=10;
      lcd_command(0x80);                  // set write LCD upper 8 char. //
      if(valve==0)
        { lcd_putc("valve on"); }         // function printf LCD. //
      else
      {
        lcd_putc("valve of");             // function printf LCD. //
        lcd_command(0xC0);                // set write LCD lowwer 8 char. //
        Lcd_putc("f");                    // function printf LCD. //
```

```
        }
        while(start_servo==1)
        {
          if(position==0) {aa=10;bb=10;cc=10;}
          if(position==1) {if(num[0]==0) {position=0;} else {aa=10;bb=10;cc=num[0];}}
          if(position==2) {aa=10;bb=num[0];cc=num[1];}
          if(position==3) {if((num[0]==1)&&(num[1]==0)&&(num[2]==0)) {aa=1;bb=0;cc=0;} else
position=0;}
          if(position==4) {aa=10;bb=10;cc=10; position=0;}
          lcd_gotoxy(2,3);
          lcd_putc("= ");
          lcd_putc(data_code[aa]);
          lcd_putc(data_code[bb]);
          lcd_putc(data_code[cc]);
          lcd_putc(" %");
          // ------------------------- scan COL_0. ------------------------- //
          output_d(0x01);
          if((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7)))
          {

            if(input(pin_d4))  { num[position] = 1; position++; }        // ROW_0
            if(input(pin_d5))  { num[position] = 4; position++; }        // ROW_1
            if(input(pin_d6))  { num[position] = 7; position++; }        // ROW_2
            if(input(pin_d7))  { num[0] = 10; num[1] = 10; position=0; } // ROW_3
            while((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7))) {delay_us(100);}
          }
          // ------------------------- END SCAN COL 0 ---------------------- //
          // ---------------------------- scan COL_1 ------------------------- //
          output_d(0x02);
          if((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7)))
          {
            if(input(pin_d4))  { num[position] = 2; position++; }  // ROW_0
```

```
  if(input(pin_d5))   { num[position] = 5; position++;  }   // ROW_1
  if(input(pin_d6))   { num[position] = 8; position++;  }   // ROW_2
  if(input(pin_d7))   { num[position] = 0; position++;  }   // ROW_3
  while((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7))) {delay_us(100);}
}
// ---------------------------- END SCAN COL 1 ------------------------ //
// ------------------------------ scan COL_2 -------------------------- //
output_d(0x04);
if((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7)))
{
  if(input(pin_d4))   { num[position] = 3; position++;  }   // ROW_0
  if(input(pin_d5))   { num[position] = 6; position++;  }   // ROW_1
  if(input(pin_d6))   { num[position] = 9; position++;  }   // ROW_2
  if(input(pin_d7))
  {
    start_servo=0;
    lcd_command(0x01);  delay_ms(100);       // clear screan. //
  }                                                          // ROW_3
  while((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7))) {delay_us(100);}
}
// ---------------------------- END SCAN COL 2 ------------------------ //
// ------------------------------ scan COL 3 -------------------------- //
output_d(0x08);
if((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7)))
{
  if(input(pin_d4))   {  }   // ROW_0
  if(input(pin_d5))   {  }   // ROW_1
  if(input(pin_d6))   {  }   // ROW_2}
  if(input(pin_d7))
  {

    if(position==1) {select_valve = num[0];}
```

```
            if(position==2) {select_valve = ((num[0])*10)+num[1];}
            if(position==3) {select_valve = ((num[0])*100)+((num[1])*10)+num[2];}
            start_servo=2;
          }                                                    // ROW_3
          while((input(pin_d4))||(input(pin_d5))||(input(pin_d6))||(input(pin_d7))) {delay_us(100);}
        }
        // ---------------------------- END SCAN COL 3 ----------------------- //
        ////////////////////////////////////////////////////////////////////////////////////////
      }
    }
////////////////////////////////////////////////////////////////////////////////////////////////////////
// ---------------------------------------- programs start servo. ------------------------------------------------ //
if(start_servo==2)
{
      lcd_command(0x01);      delay_ms(100);      // clear screan. //
      set_adc_channel(0);
      aa = 0; bb =0; cc = 0;
      lcd_command(0x80);                  // set write LCD upper 8 char. //
      if(valve==0)
      {
        lcd_putc("valve on");             // function printf LCD. //
        putc('A'); delay_ms(5);           // open valve. //
      }
      else
      {
        lcd_putc("valve of");             // function printf LCD. //
        lcd_command(0xC0);                // set write LCD lowwer 8 char. //
        Lcd_putc("f");                    // function printf LCD. //
        putc('B'); delay_ms(5);           // close valve. //
      }
      value = Read_ADC();
      volt = 0.004887585532746823069403714565004 9 * (float) value ;
```

```
sum = volt*19.71;
if (valve==0)
{
 control = (unsigned char)sum;
 if(control >= select_valve)        // stop servo motor. //
 {
   putc('e');  start_servo=0;
 }
}
else
{
   control = 102 - (unsigned char) sum;
   if(control >= select_valve)        // stop servo motor. //
   {
     putc('e');  start_servo=0;
   }
}
while(start_servo==2)
{
   value = Read_ADC();
   volt = 0.004887585532746823069403714565004 9 * (float) value ;
   sum = volt*19.71;
   if (valve==0)
   {
    control = (unsigned char)sum+2;
    if(control >= select_valve)        // stop servo motor. //
    {
     putc('e');  start_servo=0;
    }
   }
   else
   {
```

```
            control = 102 - (unsigned char) sum;
            if(control >= select_valve)         // stop servo motor. //
            {
               putc('e');  start_servo=0;
            }
          }
          aa = control/100;
          bb = (control%100)/10;
          cc = (control%100)%10;
          if((aa==0)&&(bb==0)&&(cc==0)) {aa=10;bb=10;cc=10;}
          if((aa==0)&&(bb==0)) {aa=10;bb=10;}
          if (aa==0) {aa=10;}
          lcd_gotoxy(2,3);
          lcd_putc("= ");
          lcd_putc(data_code[aa]);
          lcd_putc(data_code[bb]);
          lcd_putc(data_code[cc]);
          lcd_putc(" %");
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    }
    // ------------------------------ END LOOP TRUE ---------------------------------- //
}
// ------------------------------------ END MAIN PROGRAMS ---------------------------------- //



#include <16F877.h>
#define TxD PIN_C6
#define RxD PIN_C7
#fuses HS,NOLVP,NOWDT,NOPROTECT
#use delay (clock=20000000)
```

```
#use rs232(baud=9600,xmit=TxD,rcv=RxD)
#use fast_io(A)

#define SERVO PIN_C0       // control serv. //

// ------------------ Function Control Servo motor ----------------------------- //
// function pulse close value. //
void Servo_open(void)
{output_high(SERVO); delay_ms(1.7); output_low(SERVO); delay_ms(20);} // ** //
// function pulse open value. //
void Servo_close(void)
{output_high(SERVO); delay_ms(1.2); output_low(SERVO); delay_ms(20);}
// function pulse stop value. //
void Servo_stop(void)
{output_low(SERVO); delay_ms(20);}


// ------------------ Function Control Servo motor ----------------------------- //
void open_valve()

{
  Servo_open();
  Servo_stop();
}

void close_valve()
{
  Servo_close();
  Servo_stop();
}

void stop_valve()
```

```
{
  Servo_stop();
  Servo_stop();
}
////////////////////////////////////////////////////////////////////////////////

// ---------- function serial intrrupt ------------------ //
char valve;       // value control servo motor. //

#int_rda
void rs232_isr()
{
  valve = getc();
}
////////////////////////////////////////////////////////////
// ----------- MAIN PROGRAMS ------------------- //
void main(void)
{
    unsigned char mode = 0;
    // mode 0 = stop servo. //
    // mode 1 = open valve. //
    // mode 2 = close valve. //

    enable_interrupts(GLOBAL);
    enable_interrupts(INT_RDA);
    set_tris_c(0b11111110);                 // set pin C0 output servo. //

                                            -

// ------------------------------ loop while TRUE. ---------------------------------- //
    while(TRUE)
    {
```

```
// ------------------- mode open valve ---------------------------------------- //
    if(mode==1)
    {
      while(mode==1)
      {
        open_valve();
        // ---- if recv from control == e stop servo. ----- //
        if(valve=='e'){mode = 0;}
        // ------------------------------------------------ //
      }
    }
// ------------------------------------------------------------------------- //
// ------------------- mode close valve --------------------------------------- //
    if(mode==2)
    {
      while(mode==2)
      {
        close_valve();
        // ---- if recv from control == e stop servo. ----- //
        if(valve=='e'){mode = 0;}
        // ------------------------------------------------ //
      }
    }
// ------------------------------------------------------------------------- //
  if(mode==0)

  {
    while(mode==0)
    {
      stop_valve();
      // ---- if recv from control == A open valve. ----- //
      if(valve=='A'){mode = 1;}
```

```
            // ------------------------------------------------ //
            // ---- if recv from control == B close valve. ----- //
            if(valve=='B'){mode = 2;}
            // ------------------------------------------------ //
          }
        }
      }
      // ------------------------------ END LOOP TRUE ---------------------------------- //
}
// ------------------------------------ END MAIN PROGRAMS ---------------------------------- //
```

ภาคผนวก ข

เอกสารคู่มืออุปกรณ์อิเล็กทรอนิกส์

ข้อมูลของ PIC ไมโครคอนโทรลเลอร์ เบอร์ 16F877

MICROCHIP

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

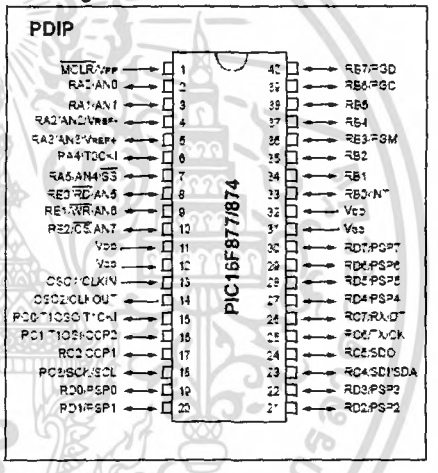
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

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

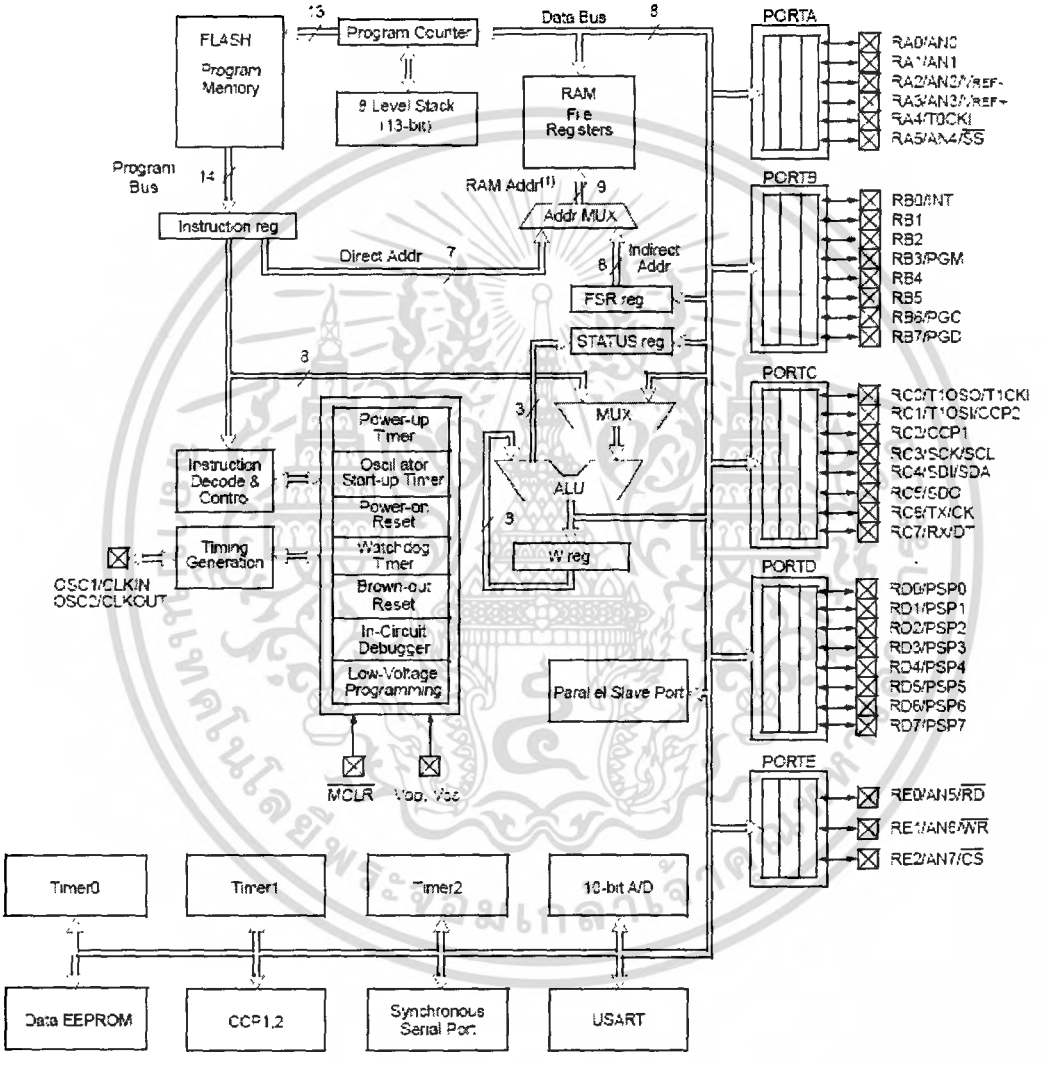
- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. 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)

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

PIC16F87X

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 software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt-on-change pin
RB5	38	42	15	I/O	TTL	Interrupt-on-change pin
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.

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

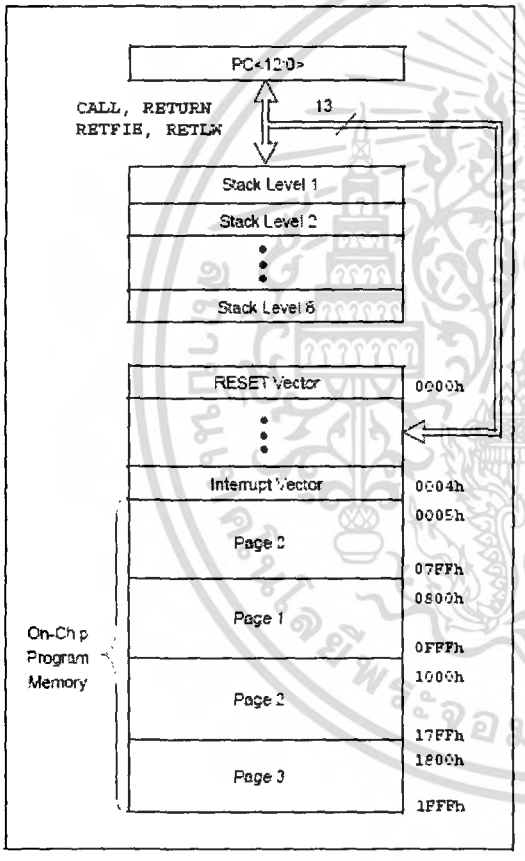


FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK

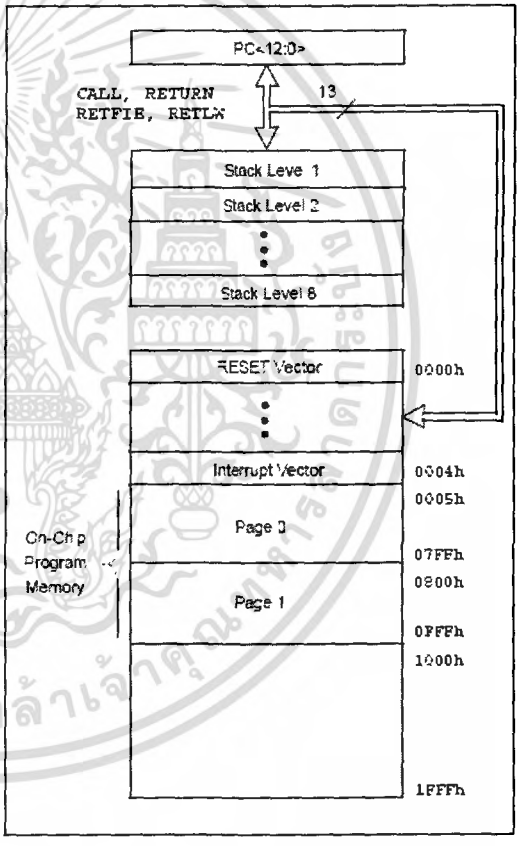


FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP

File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h		190h	
11h	TMR2	91h	SSPCON2	111h		191h	
12h	T2CON	92h	PR2	112h		192h	
13h	SSPBUF	93h	SSPADD	113h		193h	
14h	SSPCON	94h	SSPSTAT	114h		194h	
15h	CCPR1L	95h		115h		195h	
16h	CCPR1H	96h		116h		196h	
17h	CCP1CON	97h		117h	General Purpose Register 16 Bytes	197h	General Purpose Register 16 Bytes
18h	RCSTA	98h	TXSTA	118h		198h	
19h	TXREG	99h	SPBRG	119h		199h	
1Ah	RCREG	9Ah		11Ah		19Ah	
1Bh	CCPR2L	9Bh		11Bh		19Bh	
1Ch	CCPR2H	9Ch		11Ch		19Ch	
1Dh	CCP2CON	9Dh		11Dh		19Dh	
1Eh	ADRESH	9Eh	ADRESL	11Eh		19Eh	
1Fh	ADCON0	9Fh	ADCON1	11Fh		19Fh	
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h	General Purpose Register 80 Bytes
7Fh		EFh		16Fh		1EFh	
		F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h	accesses 70h - 7Fh
		FFh		17Fh		1FFh	

☐ 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.

8.3 PWM Mode (PWM)

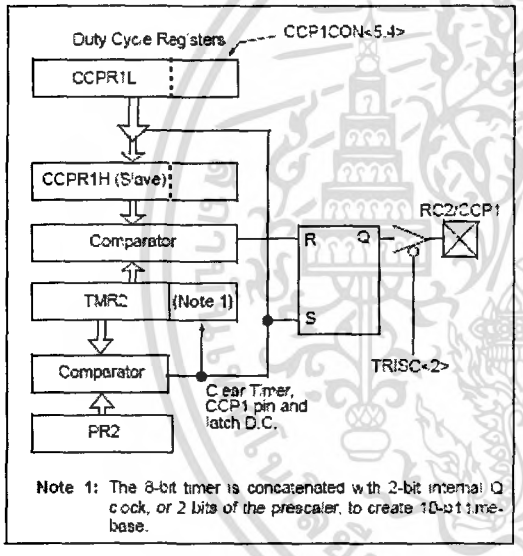
In Pulse Width Modulation mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with the PORTC data latch, the TRISC<2> bit must be cleared to make the CCP1 pin an output.

Note:	Clearing the CCP1CON register will force the CCP1 PWM output latch to the default low level. This is not the PORTC I/O data latch.

Figure 8-3 shows a simplified block diagram of the CCP module in PWM mode.

For a step-by-step procedure on how to set up the CCP module for PWM operation, see Section 8.3.3.

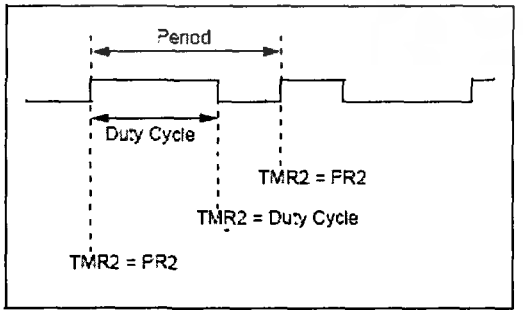
FIGURE 8-3: SIMPLIFIED PWM BLOCK DIAGRAM



Note 1: The 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time-base.

A PWM output (Figure 8-4) has a time-base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

FIGURE 8-4: PWM OUTPUT



8.3.1 PWM PERIOD

The PWM period is specified by writing to the PR2 register. The PWM period can be calculated using the following formula:

$$\text{PWM period} = [(PR2) + 1] \cdot 4 \cdot Tosc \cdot (\text{TMR2 prescale value})$$

PWM frequency is defined as *1 / [PWM period]*.

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCP1 pin is set (exception: if PWM duty cycle = 0%, the CCP1 pin will not be set)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

Note:	The Timer2 postscaler (see Section 7.1) is not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

8.3.2 PWM DUTY CYCLE

The PWM duty cycle is specified by writing to the CCPR1L register and to the CCP1CON<5:4> bits. Up to 10-bit resolution is available. The CCPR1L contains the eight MSbs and the CCP1CON<5:4> contains the two LSbs. This 10-bit value is represented by CCPR1L:CCP1CON<5:4>. The following equation is used to calculate the PWM duty cycle in time:

$$\text{PWM duty cycle} = (CCPR1L{:}CCP1CON{<}5{:}4{>}) \cdot Tosc \cdot (\text{TMR2 prescale value})$$

CCPR1L and CCP1CON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR1H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). In PWM mode, CCPR1H is a read-only register.

The CCPR1H register and a 2-bit internal latch are used to double buffer the PWM duty cycle. This double buffering is essential for glitch-free PWM operation.

When the CCPR1H and 2-bit latch match TMR2, concatenated with an internal 2-bit Q clock, or 2 bits of the TMR2 prescaler, the CCP1 pin is cleared.

The maximum PWM resolution (bits) for a given PWM frequency is given by the formula:

$$\text{Resolution} = \frac{\log\left(\frac{Fosc}{FPWM}\right)}{\log(2)} \text{ bits}$$

Note:	If the PWM duty cycle value is longer than the PWM period, the CCP1 pin will not be cleared.

8.3.3 SETUP FOR PWM OPERATION

The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the TRISC<2> bit.
4. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP1 module for PWM operation.

TABLE 8-3: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFFh	0xFFh	0xFFh	0x3Fh	0x1Fh	0x17h
Maximum Resolution (bits)	10	10	10	8	7	5.5

TABLE 8-4: REGISTERS ASSOCIATED WITH CAPTURE, COMPARE, AND TIMER1

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
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
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by Capture and Timer1

Note 1: The PSP is not implemented on the PIC16F873/876; always maintain these bits clear

TABLE 8-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
92h	PR2	Timer2 Module's Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
18h	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PWM and Timer2

Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.