น้ำพุดนตรี

DANCING FOUNTAIN

โดย

นายพีระพัฒน์ ไวถนอมทรัพย์

นายวรวุฒิ สังข์แก้ว

นายวีระวุฒิ ปุ่นแย้ม

นายสถาพร เครือเหมย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

น้ำพุดนตรี

DANCING FOUNTAIN

โดย

นายพีระพัฒน์ ไวถนอมทรัพย์ เลขที่ประจำตัวนักศึกษา 49015154

นายวรวุฒิ สังข์แก้ว เลขที่ประจำตัวนักศึกษา 49015208

นายวีระวุฒิ ปุ่นแย้ม เลขที่ประจำตัวนักศึกษา 49015210

นายสถาพร เครือเหมย เลขที่ประจำตัวนักศึกษา 49015211

อาจารย์ที่ปรึกษา

รศ.ดร.ชูชาติ ปิณฑวิรุจน์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญญานิพนธ์ปีการศึกษา 2551

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง น้ำพุดนตรี

ผู้จัดทำ

1. นายพีระพัฒน์ ไวถนอมทรัพย์ รหัส 49015154
2. นายวรวุฒิ สังข์แก้ว รหัส 49015208
3. นายวีระวุฒิ ปุ่นแย้ม รหัส 49015210
4. นายสถาพร เครือเหมย รหัส 49015211

.....อาจารย์ที่ปรึกษา

(รศ.ดร. ชูชาติ ปิณฑวิรุจน์)

23/มิ.ย./52

น้ำพุดนตรี

นาย พีระพัฒน์ ไวถนอมทรัพย์ รหัส 49015154
นาย วรวุฒิ สังข์แก้ว รหัส 49015208
นาย วีระวุฒิ ปุ่นแย้ม รหัส 49015210
นาย สถาพร เครือเหมย รหัส 49015211
รศ.ดร.ชูชาติ ปิณฑวิรุจน์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2551

บทคัดย่อ

น้ำพุเป็นอุปกรณ์ที่ใช้ ในงานตกแต่งสวนทั้งกลางแจ้งและภายนอกอาคารเพื่อความสวยงามอีกทั้งยังเป็นจุดสนใจของผู้ที่มาเยี่ยมชม และหากเป็นน้ำพุที่สามารถปรับระดับความสูงของระดับน้ำได้ตามจังหวะเสียงดนตรีได้ด้วยแล้วจะยิ่งกลายเป็นจุดสนใจมากขึ้นด้วยแต่ในการติดตั้งน้ำพุที่มีความสามารถดังกล่าวจำเป็นต้องมีค่าใช้จ่ายในการติดตั้งและการบำรุงรักษาสูงด้วยเช่นกัน

โครงงานนี้นำเสนอการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ในการควบคุมระดับของการเปิดและปิดวาล์วน้ำที่หัวของน้ำพุแต่ละหัวด้วยเซอร์โวมอเตอร์ รูปแบบการทำงานของน้ำพุจะสามารถเลือกรูปแบบได้จากการโปรแกรมลงบนตัวไมโครคอนโทลเลอร์เองหรือทำการกำหนดรูปแบบจากเครื่องคอมพิวเตอร์และในรูปแบบหนึ่งเป็นการรับสัญญาณเสียงจากแหล่งกำเนิดเสียงเข้ามาเพื่อกำหนดระดับของน้ำให้สูงต่ำตามจังหวะเสียงดนตรี

ในโครงงานนี้ได้ใช้ไมโครคอนโทรลเลอร์ PIC16F877 ของบริษัทMicrochip ควบคุมวาล์วด้วยเซอร์โวมอเตอร์Futaba S3003 วาล์วน้ำที่ใช้เป็นบอลวาล์วที่นำมาประกอบกับแท่นเพื่อให้สามารถติดตั้งเซอร์โวมอเตอร์เพื่อควบคุมวาล์วได้

Dancing Fountain

Mr. Peerapat Waitanomsub ID.49015154
Mr.Worawut Sangkaew ID.49015208
Mr.Weerawut Punyam ID.49015210
Mr.Sathaporn Khrurmey ID.49015211
Assoc.Prof.Dr.Chuchart Pindhaviroj Adviser
Education Year 2008

Abstract

Fountain is one of most popular decorations both indoor and outdoor of the buildings. If the height of fountain can be adjusted according to the beat of music it will attract more attentions. On the other hand, the construction and maintenance cost are also high as well.

This project presents the application of Microcontroller in controlling the turning on / off of the fountain nozzles with servo motors. The operation of the fountain can be selected from the programming microcontrollers or by operating from the computer. Besides, it can be controlled by the sound signals to identify the height of the fountains according to the beat of the music.

In this project, the Microchip's PIC 16F877 microcontroller and Futaba S3003 servomotor are used. The ball valves are attached to the base in order to make them be able to install servomotor or control the valves.

กิตติกรรมประกาศ

โครงงานนี้จะไม่สามารถลุล่วงลงไปได้ ถ้าไม่ได้รับความร่วมมือและความช่วยเหลือจากหลายๆ ฝ่าย โดยเฉพาะอาจารย์ รศ.ดร. ชูชาติ ปิณฑวิรุจน์ (อาจารย์ที่ปรึกษา) และอาจารย์ในภาควิชาอิเล็กทรอนิกส์ทุกท่าน ที่ให้การอุปการะในการให้คำปรึกษาและแนะนำเกี่ยวกับโครงงานในครั้งนี้และให้ยืมใช้เครื่องมือและอุปกรณ์อิเล็กทรอนิกส์ในการทดลอง และสั่งสอนให้ความรู้จนสามารถนำมาประยุกต์ใช้งานในการทำโครงงานครั้งนี้ อีกทั้ง พี่ๆ ห้อง CT LAB ทุกท่าน และเพื่อนทุกท่านที่ได้ร่วมทำงานและแลกเปลี่ยนความรู้ ความคิดเห็นซึ่งกันและกัน จนทำให้ผลงานที่ได้มีประสิทธิภาพและเป็นประโยชน์ต่อส่วนรวมอีกด้วย

นายพีระพัฒน์ ไวถนอมทรัพย์
นายวรวุฒิ สังข์แก้ว
นายวีระวุฒิ ปุ่นแย้ม
นายสถาพร เครือเหมย
ผู้จัดทำโครงงาน

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

เรื่อง **หน้า**

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูปภาพ

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญตาราง

หน้า

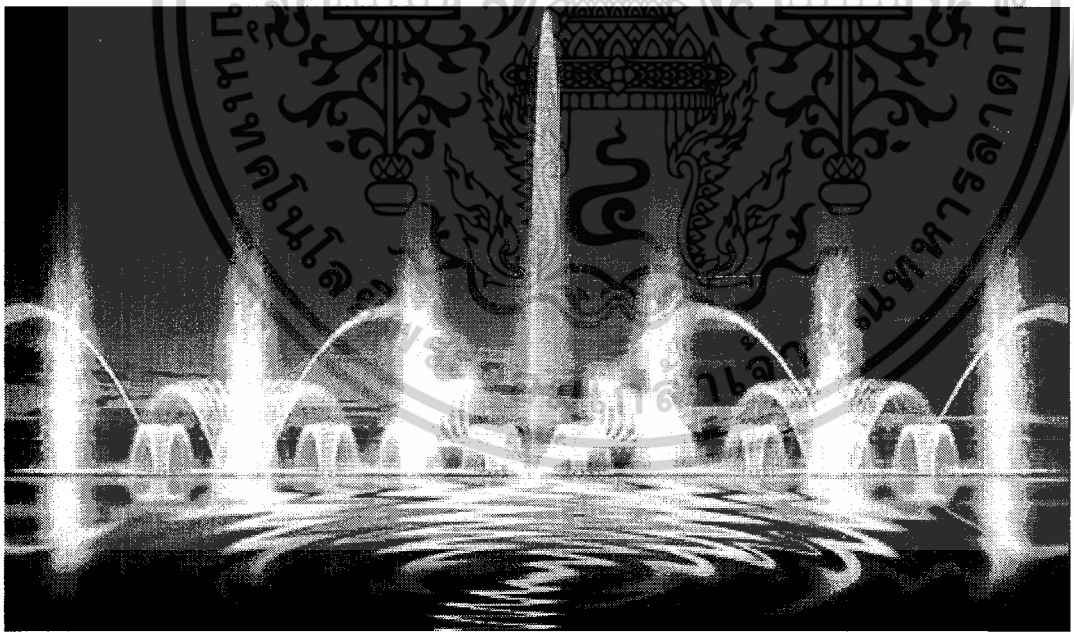
บทที่ 1
บทนำ

น้ำพุเกิดจากการเคลื่อนที่ของน้ำเนื่องจากแรงดันทำให้น้ำเคลื่อนที่ไปสู่ที่ที่มีแรงดันต่ำกว่าดังจะเห็นได้จากการกำเนิดน้ำพุร้อนที่มีการพุ่งขึ้นมาจากภายใต้ผิวโลกสู่บนผิวโลก การที่น้ำเคลื่อนตัวผ่านชั้นหินและดินใต้ผิวโลกได้นั้นเกิดจากการที่ทางน้ำใต้ดินได้รับความร้อนจากชั้นหินเหลวที่มีความร้อนสูงน้ำที่ผ่านชั้นหิวเหลวนี้จะได้รับความร้อนทำให้น้ำเดือดและเกิดการขยายตัวของน้ำเป็นแรงดันอยู่ใต้ชั้นดินและหินเมื่อแรงดันสูงจนถึงจุดหนึ่งที่ชั้นของดินและหินไม่สามารถทนได้น้ำที่เดือดและมีแรงดันมากจะดันจนทะลุผ่านชั้นดินและหินที่อยู่ใต้ผิวขึ้นมาอยู่ที่บนพื้นผิวโลกด้วยแรงดันที่สูงมากพอเมื่อน้ำเดือดนี้พ้นจากผิวโลกแรงเฉื่อยที่มีอยู่จะทำให้น้ำเดือดพุ่งสูงขึ้นไปพ้นจากผิวโลกตามแรงดันของน้ำเดือดที่เหลืออยู่จึงเกิดเป็นน้ำพุร้อน ส่วนน้ำเดือดที่เมื่อผ่านชั้นดินและหินที่อยู่ใต้ผิวโลกขึ้นมาแล้วแรงดันไม่เพียงพอก็จะเกิดเป็นบ่อน้ำร้อน ทั้งสองจึงกลายเป็นแหล่งท่องเที่ยวทางธรรมชาติที่น่าสนใจของนักท่องเที่ยว

รูปที่ 1.1 น้ำพุตามธรรมชาติ

ด้วยหลักการของการเกิดน้ำพุร้อนข้างต้นทำให้มีการพัฒนาสร้างน้ำพุขึ้น โดยทำให้น้ำเกิดแรงดันขึ้นแล้วใช้ปั๊มน้ำสร้างแรงดันสูงให้น้ำไปตามท่อที่มีปลายเปิดสิ่งที่เกิดขึ้นคือเมื่อน้ำไปถึงปลาย

ของท่อน้ำจะมีแรงเฉื่อยที่มากพอที่จะเคลื่อนที่ต่อไปได้เช่นเดียวกันกับการกำเนิดน้ำพุร้อนส่วน ความสูงของน้ำจะถูกกำหนดโดยแรงดันที่ปั๊มทำได้ ขนาดของท่อ และขนาดของปลายท่อ หากที่ค่าที่เหมาะสมจะทำให้ระบบของน้ำพุมีประสิทธิภาพสูงสุด ในปัจจุบันมีการปรับปรุงลักษณะของน้ำพุให้มีความแปลกใหม่และสวยงามมากขึ้นโดยอาจจะมีการทำให้เป็นเส้นของสายน้ำที่มีขนาดเล็กแต่มีจำนวนสายที่มากและมีการปรับระดับของความสูงโดยปรับขนาดของรูที่สายน้ำต้องพุ่งออกไม่เท่ากันทำให้ในหนึ่งหัวของน้ำพุมีสายน้ำหลายสายที่มีความสูงไม่เท่ากันเกิดความสวยงามและแปลกตามากขึ้น แต่การพัฒนาไม่ได้หยุดเพียงแค่นั้น ได้มีการพัฒนารูปแบบของน้ำพุที่แปลกออกไปโดยทำให้น้ำพุมีการพุ่งของสายน้ำที่ไม่พร้อมกันแต่ในการพุ่งจะมีรูปแบบจนสามารถใช้น้ำพุในการสร้างเป็นภาพต่างๆ ได้อีกทั้งยังมีการทำให้น้ำพุสามารถทำงานตามจังหวะของเสียงดนตรีดังเช่นในงาน “พืชสวนโลกเฉลิมพระเกียรติ” ที่ผ่านมาได้มีการใช้น้ำพุในการแสดงน้ำพุดนตรีโดยสายน้ำที่พุ่งออกมาจะเป็นไปตามจังหวะของเสียงดนตรี และยังมีการฉายภาพยนตร์ลงบนม่านน้ำโดยม่านน้ำนี้มีลักษณะการทำงานเหมือนน้ำพุทำให้เป็นที่สนใจของผู้ที่มาเยี่ยมชมอย่างมาก



รูปที่ 1.2 น้ำพุกับงานแสงสี

โครงงานนี้จะเป็นการทำน้ำพุที่มีขนาดเล็กเหมาะสมกับการตกแต่งสวนขนาดเล็กที่มีขนาดพื้นที่ไม่มากนัก ในโครงงานนี้จะเป็นการประยุกต์ใช้งานไมโครคอนโทรเลอร์ในงานควบคุมการเปิดปิดวาล์วน้ำที่หัวของน้ำพุแต่ละหัวโดยใช้เซอร์โวมอเตอร์หนึ่งตัวต่อหนึ่งหัวน้ำพุใช้โซลินอยด์วาล์วที่มีราคาสูงกว่าและยังได้มีการปรับแต่งให้วาล์วน้ำที่หัวของน้ำพุมีการหมุนเปิดปิดที่ตัวเพื่อลดกำลังของเซอร์โวมอเตอร์ที่ต้องขับ

1.1 วัตถุประสงค์

1. เพื่อศึกษาหลักการและประยุกต์ใช้งานเซอร์โวมอเตอร์และไมโครคอนโทรลเลอร์
2. เพื่อศึกษาหลักการและประยุกต์ใช้งานการสื่อสารแบบอนุกรมระหว่างอุปกรณ์อิเล็กทรอนิกส์กับคอมพิวเตอร์
3. เพื่อศึกษาวิธีการใช้งานโปรแกรมซี พลัส พลัส บิลเดอร์โปรแกรมซีคอมไพล์เลอร์ และการรับส่งข้อมูลแบบไร้สายผ่านพอร์ตอนุกรม
4. เพื่อศึกษาและประยุกต์ให้ไมโครคอนโทรเลอร์ PIC

1.2 ขอบเขตของโครงงาน

น้ำพุดนตรีจะใช้เซอร์โวมอเตอร์ในการควบคุมการเปิดและปิด ของหัวน้ำพุจำนวน 6 หัว ซึ่งควบคุมระดับความสูงของน้ำพุแต่ละหัวได้โดยการควบคุมจังหวะการเปิดและปิด โดยไมโครคอนโทรลเลอร์ PIC16F877A รูปแบบการเปิดและปิดสามารถควบคุมผ่านโปรแกรมบนเครื่องคอมพิวเตอร์ได้โดยข้อมูลจะถูกเก็บไว้ใน EEPROM ในตัวไมโครคอนโทรเลอร์ อีกทั้งยังสามารถรับอินพุท (Input) ที่เป็นสัญญาณเสียงจากภายนอกเข้ามาควบคุมระดับของน้ำพุได้ และสามารถทำการรับ-ส่งสัญญาณข้อมูลผ่านทางระบบไร้สาย(Wireless)ระหว่างเครื่องคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ได้

1.3 ประโยชน์ที่จะได้รับจากโครงงาน

ประโยชน์ที่จะได้รับจากโครงงานนี้คือนำความรู้ที่ได้ศึกษามาประยุกต์ใช้งานในการออกแบบวงจรและใช้ทักษะในการเขียนโปรแกรมเพื่อควบคุมการทำงานของไมโครคอนโทรเลอร์อีกทั้งยังสามารถเขียนโปรแกรม ซี พลัส พลัส บิลเดอร์ เพื่อติดต่อกับผู้ใช้งาน นอกจากนี้ยังเสริมทักษะในการแก้ปัญหาซึ่งจะสามารถนำไปพัฒนาในงานต่างๆ ได้ในอนาคต

1.4 รายละเอียดโดยย่อของโครงงาน

น้ำพุดนตรีประกอบไปด้วยส่วนควบคุมหัวน้ำซึ่งจะใช้วาล์วทำหน้าที่เป็นประตูน้ำและใช้เซอร์โวมอเตอร์เป็นตัวส่งกำลังหมุนวาล์วโดยที่ตัวของบอลวาล์วจะถูกปรับแต่งให้มีความคล่องตัวมากกว่าปกติ ส่วนของตัวเก็บโปรแกรมและประมวลผลจะถูกเก็บข้อมูลของน้ำพุไว้ใน PIC16F877A โดยในขั้นต้นนี้น้ำพุจะทำงานตามที่ได้โปรแกรมไว้

1.5 รายละเอียดของปริญญานิพนธ์

โครงงานน้ำพุดนตรีประกอบไปด้วยระบบต่างๆ ที่ต้องประมวลความรู้ที่ได้เรียนมาโดยมีเนื้อหาต่างๆ ดังนี้

บทที่ 1 บทนำ กล่าวถึงขอบเขตและรายละเอียดโดยย่อของโครงงาน

บทที่ 2 ทฤษฎีที่เกี่ยวข้อง กล่าวถึงทฤษฎีที่เกี่ยวข้องหลักการพื้นฐานต่างๆ ที่ใช้ในการออกแบบ

บทที่ 3 การออกแบบ กล่าวถึงอุปกรณ์ที่ใช้ในการสร้างและออกแบบรวมถึงวิธีการในการออกแบบ

บทที่ 4 ผลการทดลอง กล่าวถึงผลที่ได้จากการทดลองใช้งานในรูปแบบต่างๆ

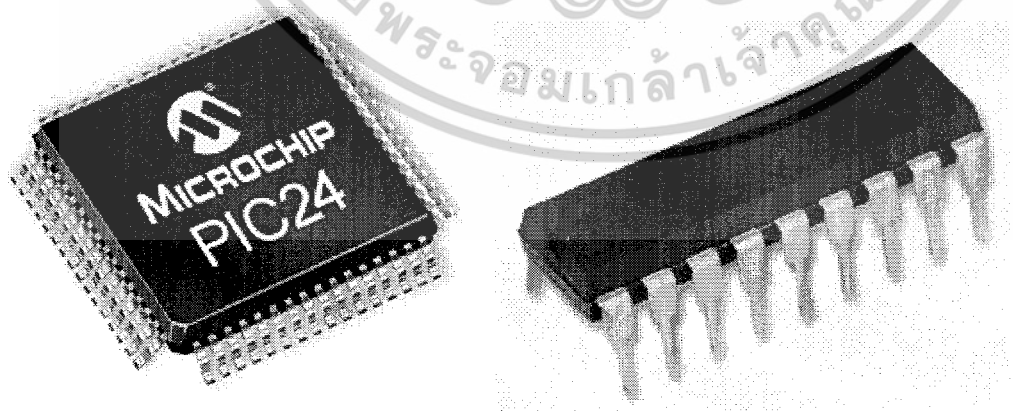
บทที่ 5 สรุปผลการทดลอง กล่าวถึงปัญหาที่เกิดขึ้นจากการทดลองแนวทางและวิธีการแก้ไขในอนาคต

บทที่ 2
ทฤษฎีและหลักการ

2.1 ไมโครคอนโทรลเลอร์ (Microcontroller)

มาจากคำสองคำ คำหนึ่งคือ ไมโคร (Micro) หมายถึงขนาดเล็ก และคำว่า คอนโทรลเลอร์ หมายถึง ตัวควบคุมหรืออุปกรณ์ควบคุม ดังนั้นไมโครคอนโทรลเลอร์ จึงหมายถึง อุปกรณ์ควบคุมขนาดเล็ก แต่ในตัวอุปกรณ์ควบคุมขนาดเล็กนี้ได้บรรจุความสามารถที่คล้ายคลึงกับคอมพิวเตอร์ที่คนส่วนใหญ่คุ้นเคย กล่าวคือ ภายในไมโครคอนโทรลเลอร์ได้รวมเอาซีพียู (CPU), หน่วยความจำและพอร์ต (port) ซึ่งเป็นส่วนประกอบสำคัญของระบบคอมพิวเตอร์ไว้ด้วยกัน โดยรวมกันอยู่ภายในตัวถังเดียวกัน

ซีพียูจะติดต่อกับหน่วยความจำโปรแกรมเพื่ออ่านคำสั่งที่ระบุไว้ โดยต้องทำการอ้างอิงตำแหน่งของหน่วยความจำผ่านสายสัญญาณที่เรียกว่า แอดเดรสบัส(address bus)แล้วทำการอ่านข้อมูลคำสั่งออกมาจากหน่วยความจำโปรแกรมในแอดเดรสนั้นๆจากนั้นทำการประมวลผลโดยมีหน่วยความจำข้อมูลแรมเป็นที่พักข้อมูลที่อยู่ในระหว่างการประมวลผลหรืออาจมองว่าหน่วยความจำข้อมูลแรมเป็นเหมือนกระดาษทดในการคำนวณก็ได้ ข้อมูลในการประมวลผลจะส่งผ่านสายสัญญาณที่เรียกว่า บัสข้อมูล(data bus) แล้วส่งต่อไปยังอุปกรณ์ภายนอกผ่านทางพอร์ตอินพุต(input)เอาท์พุต (output)

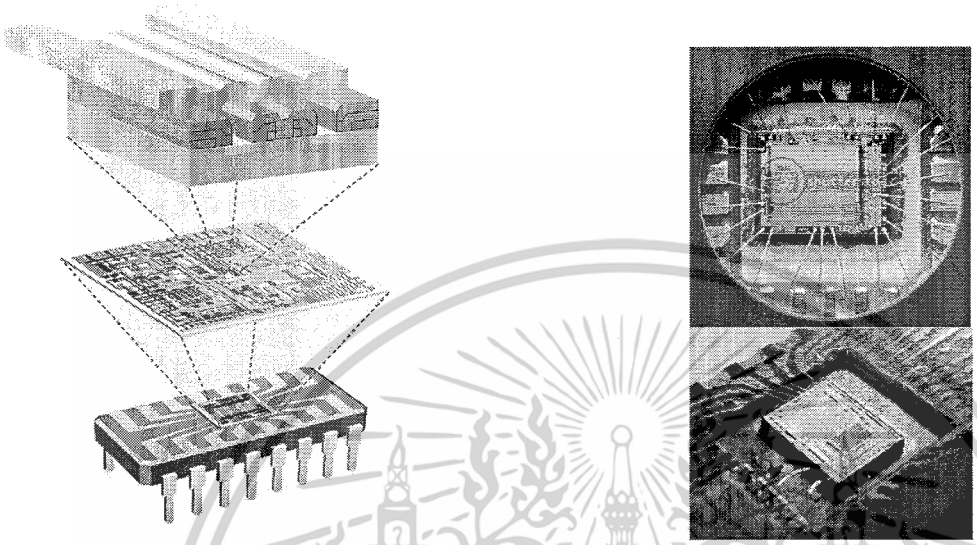


รูปที่ 2.1 ไมโครคอนโทรลเลอร์

2.1.1 หน่วยประมวลผลกลางหรือซีพียู (CPU: Central Processing Unit)

เป็นเหมือนมันสมองของไมโครคอนโทรลเลอร์โดยซีพียูนี้ทำหน้าที่ประมวลผลข้อมูลที่เข้ามาในระบบ แล้วทำการส่งต่อไปยังส่วนต่างๆเพื่อควบคุมการทำงานต่อไป หัวใจหลักของซีพียูคือหน่วยคำนวณทางคณิตศาสตร์และลอจิก(ALU : Arithmetic and logic unit) ซึ่งได้รับการกำหนดจังหวะการทำงานจากส่วนควบคุมลำดับการทำงาน โดยจังหวะการทำงานนั้นจะสัมพันธ์กับสัญญาณนาฬิกาเมื่อซีพียูทำการติดต่อกับหน่วยความจำสิ่งที่ปรากฏขึ้นบนบัสข้อมูลภายในซีพียูคือ รหัสคำสั่ง(instruction code)ต้องผ่านการทำงานของส่วนถอดรหัสคำสั่ง(instruction decoder)เสียก่อนจะได้เป็นข้อมูลคำสั่งที่ซีพียูเข้าใจได้และสามารถดำเนินการต่อได้หลังจากที่หน่วยคำนวณทางคณิตศาสตร์และลอจิกประมวลผลแล้วก็จะส่งข้อมูลมายังส่วนเชื่อมต่อรีจิสเตอร์ภายในซีพียูเพื่อติดต่อกับส่วนอื่นๆต่อไป

การทำงานของซีพียูมีด้วยกันสองจังหวะคือ เฟตช์(fetch) และ เอ็กซิคิวต์(executed)โดยการทำงานจะเริ่มต้นจากการเฟตช์ ซึ่งก็คือการเรียกหรือการเข้าถึงคำสั่ง แล้วทำการถอดรหัสเป็นภาษาเครื่องเพื่อเตรียมประมวลผลจากนั้นจะเป็นจังหวะของการเอ็กซิคิวต์ซึ่งก็คือการกระทำตามคำสั่งที่กำหนดให้เสร็จสิ้น การที่จะระบุว่าไมโครคอนโทรลเลอร์มีขีดความสามารถในการประมวลผลเป็นอย่างไร จะพิจารณาที่ความสามารถในการประมวลผลของซีพียู หากซีพียูสามารถประมวลผลข้อมูลได้สูงสุด 8 บิต นั่นคือไมโครคอนโทรลเลอร์นี้เป็นแบบ 8 บิตแต่ในซีพียูในไมโครคอนโทรลเลอร์สมัยใหม่บางตัวมีขนาด 8 บิตแต่สามารถประมวลผลกับข้อมูล 16 บิตได้ ทำให้บางครั้งผู้ผลิตจึงระบุออกมาว่า ไมโครคอนโทรลเลอร์ตัวนี้ทำงานแบบ 16 บิตอาจกล่าวได้ว่า เป็นไมโครคอนโทรลเลอร์ 16 บิตเทียม เพราะถ้าหากเป็นแบบ 16 บิตแท้ ซีพียูต้องรองรับข้อมูลได้เต็ม 16 บิต หรือถ้าอ่านในดาต้าชีตของไมโครคอนโทรลเลอร์ตระกูลนั้นๆจะต้องระบุว่าเป็น 16–bit core ดังนั้นจึงต้องพิจารณารายละเอียดในส่วนนี้ด้วย



รูปที่ 2.2 ซีพียู

2.1.2 หน่วยความจำ

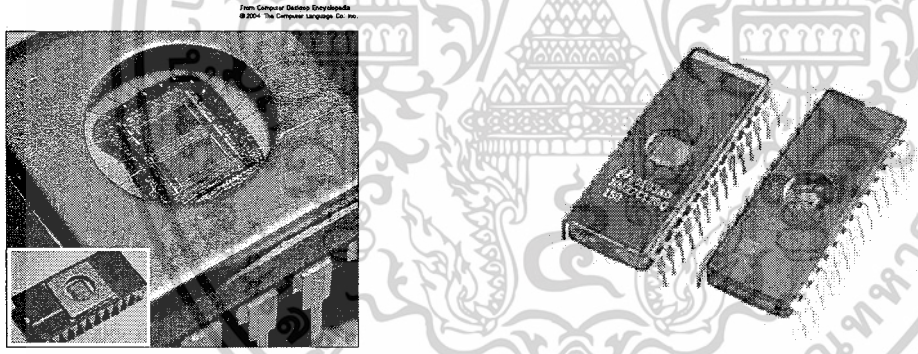
ในไมโครคอนโทรลเลอร์จะประกอบด้วยหน่วยความจำ 3 แบบคือ หน่วยความจำโปรแกรม (program memory), หน่วยความจำข้อมูลแรม (RAM data memory) และหน่วยความจำข้อมูลอีอีพรอม (EEPROM data memory)

2.1.3 หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมเป็นที่สำหรับเก็บข้อมูลคำสั่งของโปรแกรมควบคุมที่ผู้พัฒนาเขียนขึ้น หรือเรียกว่า โปรแกรมมอนิเตอร์(monitor program)ซีพียูจะเข้ามาติดต่อเพื่ออ่านข้อมูลรหัสคำสั่งจากหน่วยความจำในส่วนนี้แล้วไปประมวลผลเพื่อควบคุมการทำงานของระบบทั้งหมดต่อไปเรียกได้ว่า มีความสำคัญเท่าๆกับซีพียูเลยทีเดียว หน่วยความจำโปรแกรมนี้มักมีขนาดใหญ่และถ้ายิ่งมีขนาดใหญ่มากเท่าใด ก็จะสามารถบรรจุโปรแกรมที่มีความซับซ้อนหรือสามารถเก็บตารางข้อมูลที่ใช้ในการประมวลผลได้มากตาม โดยทั่วไปมีความจุไม่น้อยกว่า 512 ไบต์ แต่จะให้ดีควรมีความจุ 1 กิโลไบต์ขึ้นไปจึงจะช่วยให้การเขียนโปรแกรมควบคุมอิสระเพิ่มมากขึ้น ขนาดของหน่วยความจำโปรแกรมจะแปรตามความก้าวหน้าของเทคโนโลยีมีกรพัฒนาให้ไมโครคอนโทรลเลอร์มีความจุของหน่วยความจำโปรแกรมสูงขึ้นเรื่อยๆเป็น 4, 8, 16, 32 และ 64 กิโลไบต์และยังไม่สิ้นสุดเท่านี้เชื่อแน่ว่าต้องมีการ

พัฒนาไมโครคอนโทรลเลอร์ให้มีความจุของหน่วยความจำโปรแกรมสูงเป็นหลักร้อยกิโลไบต์หรือหลักเมกะไบต์ ชนิดของหน่วยความจำโปรแกรมที่ใช้ในไมโครคอนโทรลเลอร์มีอยู่ 3 แบบ ที่นิยมกันคือ แบบอีพรอม (EPROM: Erasable Programmable Read-Only Memory), แบบอีอีพรอม (Electrically Erasable Read-Only Memory) และแบบแฟลช (Flash) ความแตกต่างอยู่ที่จำนวนครั้งในการลบและเขียนข้อมูลทับลงไปใหม่โดยสามารถสรุปได้ดังนี้

2.1.3.1 แบบอีพรอม ยังแบ่งเป็น 2 แบบคือ แบบโปรแกรมได้หลายครั้งและแบบโปรแกรมได้ครั้งเดียวถ้าหากเป็นแบบโปรแกรมได้หลายครั้งบนตัวถังของไมโครคอนโทรลเลอร์จะมีหน้าต่างกระจกติดอยู่สามารถมองเห็นชิปภายในได้เวลาลบต้องลบด้วยแสงอัลตร้าไวโอเลตจำนวนในการโปรแกรมใหม่อยู่ระหว่าง 10-100 ครั้งแต่ถ้าเป็นแบบโปรแกรมได้ครั้งเดียวหรือ OTP (One-time programmable) จะไม่สามารถลบได้ตัวถังจะถูกปิดมิดชิดเหมือไอซีธรรมดา



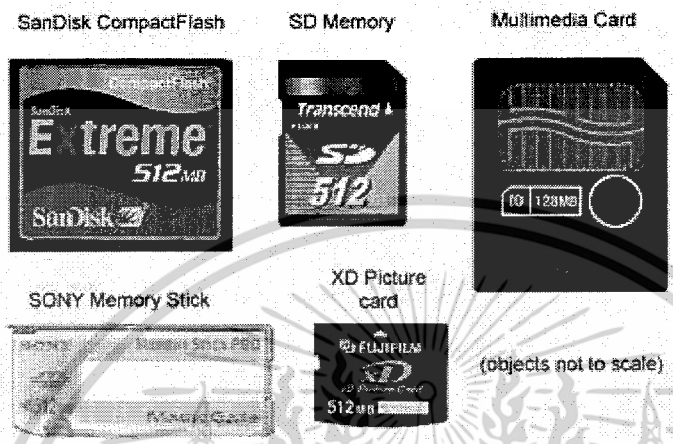
รูปที่ 2.3 หน่วยความจำแบบอีพรอม

2.**1.3.2 แบบอีอีพรอม** หน่วยความจำแบบนี้จะลบและเขียนใหม่ได้ส่วนสัญญาณไฟฟ้า ในอดีตเป็นที่นิยมมากเนื่องจากสามารถลบและเขียนได้เป็นร้อยรอบขึ้นไป ในบางตระกูลถึง 1 ล้านครั้งแต่ในปัจจุบันแบบนี้ไม่เป็นที่นิยมใช้ในไมโครคอนโทรลเลอร์เนื่องจากต้นทุนสูง

รูปที่ 2.4 หน่วยความจำแบบอีอีพรอม

2.1.3.3 แบบแฟลช หน่วยความจำโปรแกรมชนิดนี้สามารถลบและเขียนได้ด้วยสัญญาณไฟฟ้า แตกต่างกับแบบอีอีพรอมในเชิงการใช้งานตรงที่กระบวนการลบข้อมูลหน่วยความจำโปรแกรมแบบแฟลชจะไม่สามารถเลือกลบได้เฉพาะเจาะจงบางแอดเดรสบางตำแหน่งได้ เมื่อทำการลบข้อมูลจะต้องลบทั้งหมดหน่วยความจำโปรแกรมแบบนี้ได้รับความนิยมมากเนื่องจากราคาไม่สูงและสามารถโปรแกรมได้เป็นร้อยครั้งขึ้นไป แต่โดยปกติมักเริ่มที่1,000ครั้งในบางรุ่นสูงเป็นหมื่นครั้งและเป็นแสนครั้งขึ้นอยู่กับแรงดันที่ใช้ในการโปรแกรม

ขนาดข้อมูลของหน่วยความจำโปรแกรมขึ้นอยู่กับผู้ผลิตไมโครคอนโทรลเลอร์ยกตัวอย่างเช่น ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ขนาดของหน่วยความจำโปรแกรมเป็น 8 บิต ถ้าเป็นขนาด 16 บิต แต่ขนาดของหน่วยความจำโปรแกรมไม่ได้เป็นตัวระบุความสามารถในการประมวลผลของไมโครคอนโทรลเลอร์ตัวอย่างเช่น PIC กับ AVR ต่างมีขนาดของหน่วยความจำโปรแกรมสูงกว่า 8 บิต แต่ทั้งคู่ต่างเป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ทั้งนี้เพราะซีพียูเป็นแบบ 8 บิต ขนาดของหน่วยความจำจะส่งผลต่อการระบุความจุของหน่วยความจำโปรแกรมภายในไมโครคอนโทรลเลอร์เบอร์นั้นๆตัวอย่างเช่น PIC16F628 มีหน่วยความจำโปรแกรมความจุ 2K×14 บิต หมายความว่า PIC16F628 มีหน่วยความจำขนาด 14 บิตอยู่ทั้งสิ้น 2,048 ตำแหน่งหรือเรียกว่ามีความจุ 2 กิโลเวิร์ด ทั้งนี้เนื่องจากขนาดหน่วยความจำโปรแกรมสูงกว่า 8 บิต จึงเรียกว่า ไบต์ไม่ได้ขนาดของข้อมูลที่มากกว่า 8 บิต มักถูกเรียกว่าเวิร์ดสำหรับไมโครคอนโทรลเลอร์ PIC16F628 ขนาดของข้อมูล 1 เวิร์ดคือ 14 บิตแต่ถ้าเป็นในตระกูล MCS-51 จะสามารถระบุเป็นหน่วยไบต์ตรงๆ AVR จะเหมือนกับ PIC คือต้องระบุเป็นเวิร์ด



รูปที่ 2.5 หน่วยความจำแบบแฟลช

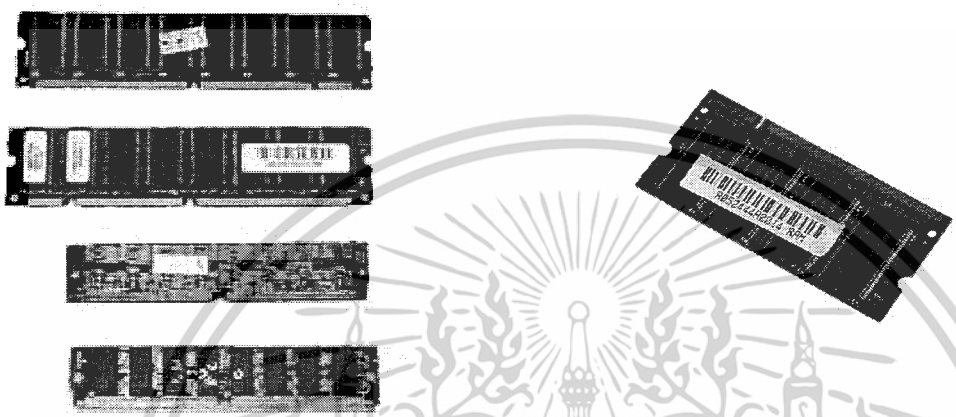
2.1.4 หน่วยความจำข้อมูลแรม

เป็นหน่วยความจำที่ต้องมีในไมโครคอนโทรลเลอร์ทุกตัวเพราะใช้เป็นพื้นที่สำหรับเก็บข้อมูลทั้งในระหว่างและหลังจากการประมวลผล ยิ่งมีมากยิ่งช่วยให้การทำงานสะดวกเพราะหน่วยความจำแรมมีอัตราเร็วในการอ่านเขียนสูงมากและไม่จำกัดจำนวนรอบในการอ่านเขียนในพื้นที่ของหน่วยความจำข้อมูลแรมจะแบ่งออกเป็น 2 ส่วนคือ ส่วนของข้อมูลทั่วไปสำหรับเก็บค่าตัวแปรและส่วนของรีจิสเตอร์

โดยปกติแล้วหน่วยความจำข้อมูลแรมจะมีความจุไม่มากเมื่อเทียบกับหน่วยความจำโปรแกรมในบางตัวอยู่ระดับสิบไบต์ แต่ถ้าไมโครคอนโทรลเลอร์มีความสามารถสูงขึ้นความจุของหน่วยความจำข้อมูลแรมจะเพิ่มมากขึ้นตามไปด้วย ทั้งนี้เพราะต้องเพิ่มในส่วนของรีจิสเตอร์ตามความสามารถที่สูงขึ้นของไมโครคอนโทรลเลอร์

ทางด้านขนาดของหน่วยความจำข้อมูลแรม โดยส่วนใหญ่จะมีขนาด 8 บิต แต่สามารถต่อร่วมกันเป็น 16 บิตได้ ส่วนการจัดสรรตำแหน่งแอดเดรสของหน่วยความจำข้อมูลจะขึ้นอยู่กับสถาปัตยกรรมของไมโครคอนโทรลเลอร์หากเป็นแบบฮาร์วาร์ด (Harvard) จะได้รับการจัดสรรให้อยู่แยกจากหน่วยความจำโปรแกรมจึงทำให้มีแอดเดรสที่เหมือนกันได้นั่นคือผู้ใช้งานจะพบแอดเดรสเดียวกันทั้งหน่วยความจำโปรแกรมและหน่วยความจำข้อมูล แต่จริงๆแล้วอยู่ต่างที่กัน จะพบในไมโครคอนโทรลเลอร์ MCS-51, PIC, AVR เป็นต้น แต่ถ้าแบบพรินซ์ตัน (Princeton) จะจัดสรรให้อยู่

ในบริเวณเดียวกันดังนั้นค่าแอดเดรสจะไม่มีทางตรงกัน จะพบในไมโครคอนโทรลเลอร์ 68HCxxx ของ Motorola



รูปที่ 2.6 หน่วยความจำแบบแรม

2.1.5 หน่วยความจำข้อมูลอีอีพรอม

เป็นหน่วยความจำข้อมูลพิเศษที่ไมโครคอนโทรลเลอร์บางเบอร์ บางรุ่น บางตระกูลไม่มี ใช้สำหรับเก็บข้อมูลที่ต้องการรักษาไว้เมื่อไม่มีการจ่ายไฟเลี้ยงให้แก่ไมโครคอนโทรลเลอร์ การติดต่อเพื่ออ่านเขียนจะมีลักษณะเป็นพิเศษขึ้นอยู่กับไมโครคอนโทรลเลอร์แต่ละเบอร์ ขนาดของหน่วยความจำแบบนี้มักเท่ากับ 8 บิต ส่วนความจุก็แตกต่างกันไปมีตั้งแต่ไม่กี่สิบไบต์จนถึงเป็นกิโลไบต์

การอ่านเขียนหน่วยความจำแบบนี้จะใช้สัญญาณไฟฟ้าทั้งหมดและสามารถรักษาข้อมูลล่าสุดไว้แม้ว่าจะไม่มีการจ่ายไฟเลี้ยงให้แก่ไมโครคอนโทรลเลอร์แล้วก็ตาม สำหรับจำนวนรอบในการเขียนโดยปกติอยู่ในหลักล้านครั้งขึ้นไป

2.1.6 รีจิสเตอร์ (Register)

เป็นหน่วยความจำพิเศษที่มีบทบาทสูงมากในการทำงานของไมโครคอนโทรลเลอร์สามารถที่จะอ่านและเขียนข้อมูลได้ตลอดเวลาจนกว่าจะหยุดจ่ายไฟเลี้ยงให้แก่ไมโครคอนโทรลเลอร์ หน้าที่หลักคือใช้เก็บข้อมูลในการทำงานของไมโครคอนโทรลเลอร์โดยข้อมูลที่เก็บนี้มีทั้งข้อมูลแสดงสถานะการทำงานข้อมูลสำหรับควบคุมการทำงานโมดูลย่อยต่างๆภายในไมโครคอนโทรลเลอร์ ข้อมูลที่รับเข้ามา

จากพอร์ตอินพุต ข้อมูลที่ต้องการส่งออกไปยังอุปกรณ์ภายนอกผ่านพอร์ตเอาต์พุต โดยข้อมูลแต่ละประเภทก็จะถูกจัดเก็บลงในรีจิสเตอร์ที่แตกต่างกันตามหน้าที่การทำงาน

หน่วยความจำที่นำมาใช้เป็นรีจิสเตอร์มีด้วยกัน 2 ลักษณะขึ้นอยู่กับสถาปัตยกรรมของไมโครคอนโทรลเลอร์ หากเป็นแบบพรินซ์ตันรีจิสเตอร์จะมีอยู่ด้วยกัน 2 ส่วน ส่วนแรกจะอยู่ร่วมกับซีพียูหรือเรียกว่า รีจิสเตอร์ซีพียู ส่วนที่สองจะอยู่แยกต่างหากซึ่งมักเป็นรีจิสเตอร์ควบคุมพอร์ตอินพุตเอาท์พุตและรีจิสเตอร์แสดงสถานะ แต่ในสถาปัตยกรรมแบบฮาร์วาร์ดจะใช้บางส่วนในหน่วยความจำข้อมูลแรมภายในไมโครคอนโทรลเลอร์ อย่างไรก็ตามข้อมูลในรีจิสเตอร์จะคงอยู่ตราบเท่าที่ยังจ่ายไฟเลี้ยงให้แก่ไมโครคอนโทรลเลอร์ ซีพียูสามารถอ่านเขียนรีจิสเตอร์ได้ตลอดเวลาเท่ากับอายุการใช้งานของไมโครคอนโทรลเลอร์ อาจกล่าวได้ว่ารีจิสเตอร์คือหน่วยความจำข้อมูลที่มีการระบุชื่อชัดเจนมีแอดเดรสและฟังก์ชั่นการทำงานที่เฉพาะเจาะจงตามที่กำหนดโดยผู้ผลิตไมโครคอนโทรลเลอร์

2.1.6.1 รีจิสเตอร์ตัวนับโปรแกรมหรือโปรแกรมเคาน์เตอร์ (PC)

การที่ซีพียูสามารถติดต่อกับหน่วยความจำโปรแกรมเพื่ออ่านข้อมูลคำสั่งได้อย่างถูกต้องเป็นผลมาจากรีจิสเตอร์หน้าที่พิเศษตัวหนึ่งคือ รีจิสเตอร์ตัวนับโปรแกรม หรือ PC (Program Counter) โดย PC จะเป็นตัวชี้ตำแหน่งแอดเดรสของหน่วยความจำโปรแกรมที่ซีพียูต้องไปกระทำในลำดับถัดไป โดยปกติแล้วค่าของ PC จะเปลี่ยนแปลงโดยอัตโนมัติขึ้นอยู่กับผลการทำงานที่เกิดขึ้นในไมโครคอนโทรลเลอร์บางตระกูลสามารถเข้าถึงรีจิสเตอร์ PC เพื่อทำการอ่านเขียนได้ ในบางตระกูลก็ไม่สามารถทำได้

ขนาดของรีจิสเตอร์ PC ขึ้นอยู่กับความจุของหน่วยความจำโปรแกรมภายในไมโครคอนโทรลเลอร์นั้นเช่น MCS-51 สามารถมีหน่วยความจำโปรแกรมได้สูงสุด 64 กิโลไบต์หรือ 65,536 ตำแหน่ง ขนาดของรีจิสเตอร์ PC จึงมีได้เท่ากับ 16 บิต ส่วนในไมโครคอนโทรลเลอร์ PIC อนุกรม 14 บิต มีหน่วยความจำโปรแกรมได้สูงสุด 8 กิโลเวิร์ด หรือ 8,192 ตำแหน่ง รีจิสเตอร์ PC จึงมีขนาด 13 บิต เป็นต้น

2.1.6.2 สแต็กในไมโครคอนโทรลเลอร์

สแต็ก(stack)เป็นหน่วยความจำส่วนพิเศษที่ไมโครคอนโทรลเลอร์ทุกตัวต้องมีโดยหน้าที่ของมันคือ เก็บข้อมูลที่ยังต้องการอยู่ของรีจิสเตอร์และเมื่อข้อมูลนั้นถูกนำมาเก็บไว้ในสแต็กแล้วก็สามารถที่จะเปลี่ยนข้อมูลในรีจิสเตอร์ตัวนั้นๆได้ทันทีหลังจากที่ทำงานเรียบร้อยจึงกลับมาอ่านข้อมูลเดิมกลับ

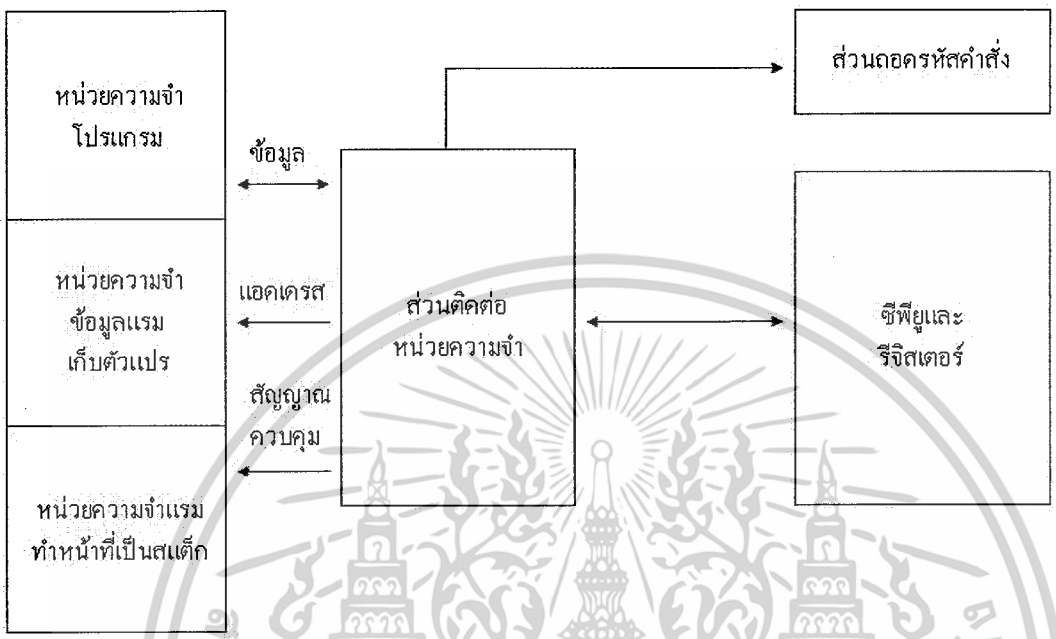
จากสแต็ก การเก็บข้อมูลของสแต็กจะมีลักษณะเป็นระดับหรือเป็นชั้น ข้อมูลที่เก็บเข้ามาก่อนจะต้องอ่านออกทีหลังเป็นแบบ FILO(First In Last Out) และจำนวนระดับหรือจำนวนชั้นของสแต็กก็มีจำกัด

ในไมโครคอนโทรลเลอร์ส่วนใหญ่จะมีความจุของสแต็กไม่น้อยกว่า 8 ระดับ การที่ยิ่งมีขนาดของสแต็กมากหรือมีจำนวนระดับมาก ก็จะยิ่งช่วยให้การทำงานสะดวกขึ้น เพราะในการประมวลผลมีโอกาสที่ต้องพักข้อมูลในรีจิสเตอร์หลักที่สำคัญเพื่อไปทำงานอื่นก่อน หลังจากนั้นจึงจะกลับมาทำงานต่อ โดยเฉพาะอย่างยิ่งกับงานที่มีการอินเตอร์รัปต์หรือขัดจังหวะซีพียูบ่อยๆ รวมถึงงานที่มีการกระโดดไปทำงานที่โปรแกรมย่อยจำนวนมาก เพราะเมื่อต้องกระโดดออกจากโปรแกรมหลักไปทำงานที่โปรแกรมย่อย ก็ต้องเก็บข้อมูลของรีจิสเตอร์หลักที่ทำงานค้างอยู่ลงในสแต็ก หลังจากนั้นกระโดดไปที่โปรแกรมย่อยที่มีความต้องการเขียนข้อมูลในรีจิสเตอร์ตัวเดียวกันนี้ หลังจากทำงานแล้วจึงกลับมาที่โปรแกรมหลัก แล้วอ่านค่าเดิมก่อนหน้านี้กลับมาทำงานต่อ ทว่า ในงานบางลักษณะมีการกระโดดไปทำงานยังโปรแกรมย่อยซ้อนกัน 2-3 ชั้น ทำให้ต้องมีการเก็บข้อมูลไว้ในสแต็กมากขึ้น หากความจุของสแต็กมีน้อยก็จะไม่สามารถรองรับการทำงานในลักษณะนี้ได้

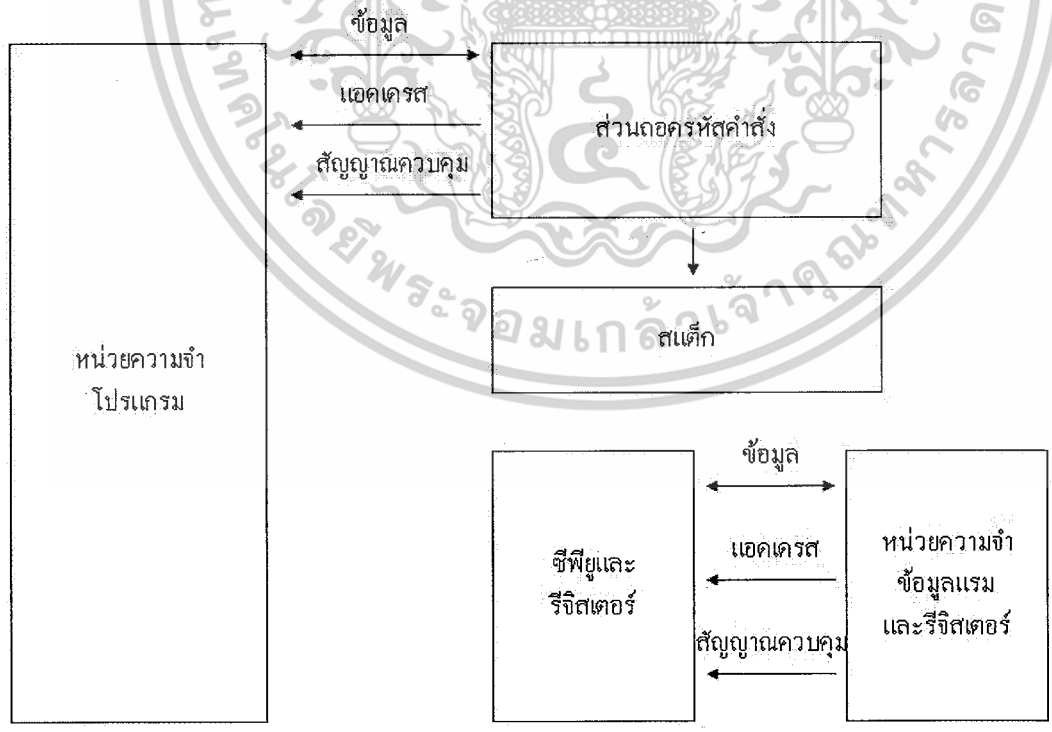
ขนาดของสแต็กโดยปกติจะต้องเท่ากับขนาดของรีจิสเตอร์ตัวนับโปรแกรมหรือ PC เพราะมีโอกาสที่จะต้องเก็บค่าของ PC ไว้ในสแต็กด้วย

2.2 สถาปัตยกรรมของไมโครคอนโทรลเลอร์

เป็นที่ยอมรับกันว่าสถาปัตยกรรมของไมโครคอนโทรลเลอร์มีด้วยกัน 2 แบบคือ **พรินซ์ตัน (Princeton)** หรือ **ฟอนนิวแมน (Von Neumann)** และ **ฮาร์วาร์ด (Harvard)** ในรูปที่ 2.7 และ 2.8 แสดงการจัดสรรหน่วยความจำและรีจิสเตอร์ในสถาปัตยกรรมของไมโครคอนโทรลเลอร์ทั้งสองแบบ พิจารณาในรูปที่ 2.7 ก่อนเป็นการจัดสรรในสถาปัตยกรรมแบบพรินซ์ตัน จะเห็นได้ว่า มีโครงสร้างที่เรียบง่ายไม่ซับซ้อน ส่วนของหน่วยความจำโปรแกรมกับหน่วยความจำข้อมูลจะได้รับการจัดสรรให้อยู่รวมกัน ติดต่อกับซีพียูส่วนจัดการเชื่อมต่อหน่วยความจำ และภายในซีพียูจะมีรีจิสเตอร์บรรจุอยู่ ข้อดีของสถาปัตยกรรมแบบนี้คือ ออกแบบง่าย เพราะหน่วยความจำทั้งหมดอยู่รวมกัน สามารถเข้าถึงได้ง่าย หน่วยความจำแรมหากมีขนาดใหญ่เพียงพอก็จะสามารถเก็บได้ทั้งโปรแกรมควบคุมการทำงานและข้อมูลของตัวแปรในการประมวลผล ข้อด้อยของสถาปัตยกรรมแบบนี้ก็คือ ความเร็วในการประมวลผลเนื่องจากหน่วยความจำอยู่รวมกัน จึงต้องติดต่อหน่วยความจำโปรแกรมสลับกับหน่วยความจำข้อมูล ส่งผลซีพียูต้องใช้จำนวนไซเกิลในการทำงานมาก แต่ข้อด้อยนี้สามารถชดเชยได้หากไมโครคอนโทรลเลอร์สามารถทำงานกับสัญญาณนาฬิกาที่มีความถี่สูงมากได้



รูปที่ 2.7 สถาปัตยกรรมแบบพรินซ์ตันหรือฟอนนิวแมน



รูปที่ 2.8 สถาปัตยกรรมแบบฮาร์วาร์ด

ในขณะที่สถาปัตยกรรมแบบฮาร์วาร์ด ซึ่งแสดงในรูปที่ 2.8 จะแยกส่วนของหน่วยความจำข้อมูลและรีจิสเตอร์ออกจากหน่วยความจำโปรแกรม ทำให้ไซเกิลการทำงานลดลง เนื่องจากสามารถติดต่อหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลได้เร็วกว่า นั่นคือทำงานได้เร็วกว่าแบบพรินซ์ตัน นอกจากนั้นในสถาปัตยกรรมแบบนี้ในขณะที่ซีพียูกำลังเอ็กซิคิวต์คำสั่งในปัจจุบันอยู่สามารถที่จะเฟตช์คำสั่งถัดไปได้ ยิ่งเป็นการเพิ่มความเร็วในการทำงานให้กับไมโครคอนโทรลเลอร์

2.3 การทำงานของไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์จะสามารถทำงานได้เมื่อจ่ายไฟเลี้ยงและต่อวงจรกำเนิดสัญญาณนาฬิกาให้แก่มัน จากนั้นซีพียูภายในไมโครคอนโทรลเลอร์จะติดต่อกับหน่วยความจำโปรแกรมเพื่ออ่านข้อมูลคำสั่งแล้วทำงานตามคำสั่งที่บรรจุอยู่ในหน่วยความจำโปรแกรม

นั่นหมายความว่า ต้องที่การเขียนข้อมูลในหน่วยความจำโปรแกรมก่อน โดยไมโครคอนโทรลเลอร์แต่ละเบอร์จะมีรูปแบบของข้อมูลคำสั่งที่แตกต่างกัน ซึ่งจะต้องอาศัยกระบวนการเขียนโปรแกรม (programming) ภาษาที่ใช้ในการเขียนโปรแกรมสามารถแบ่งได้ 2 ระดับ คือ ภาษาสูง (high language) และภาษาแอสเซมบลี (assembly language) โดยปกติไมโครคอนโทรลเลอร์ต้องการโปรแกรมที่เขียนด้วยภาษาแอสเซมบลี เนื่องจากสามารถทำงานได้รวดเร็วผ่านกระบวนการแปลงข้อมูลคำสั่งเป็นข้อมูลเลขฐานสิบหกเพื่อทำงานตามคำสั่งเพียง 1 ขั้นตอนคือ แปลงจากภาษาแอสเซมบลีเป็นข้อมูลเลขฐานสิบหก หรือที่เรียกว่า ออปโค้ด (Opcode) แต่ข้อเสียของการเขียนโปรแกรมด้วยภาษาแอสเซมบลีก็คือ ผู้เขียนต้องทำความเข้าใจในชุดคำสั่งของไมโครคอนโทรลเลอร์เบอร์นั้นๆ อย่างถ่องแท้ และเมื่อเปลี่ยนเบอร์ไมโครคอนโทรลเลอร์ก็ต้องทำการเรียนรู้และทำความเข้าใจชุดคำสั่งใหม่ ซึ่งอาจทำให้เสียเวลามาก และการเขียนโปรแกรมด้วยภาษาแอสเซมบลี ผู้เขียนต้องมีทักษะในการเขียนโปรแกรมสูงพอสมควร และเข้าใจถึงสถาปัตยกรรมของไมโครคอนโทรลเลอร์เป็นอย่างดี

ในขณะที่การเขียนโปรแกรมด้วยภาษาสูง อาทิ ภาษาซี ภาษาเบสิก ต้องผ่านกระบวนการที่เรียกว่า คอมไพล์ (compile) เพื่อแปลงภาษาระดับสูงเหล่านั้นเป็นภาษาเครื่องหรือออปโค้ดของไมโครคอนโทรลเลอร์เบอร์นั้นๆ เสียก่อน และโปรแกรมที่ใช้ในการคอมไพล์นั้นเรียกว่า คอมไพเลอร์ (compiler) มักจะมีราคาแพง เมื่อใช้เครื่องมือทางซอฟต์แวร์ตัวนี้ ทำให้ผู้เขียนโปรแกรมอาจไม่จำเป็นต้องศึกษาสถาปัตยกรรมชุดคำสั่งของไมโครคอนโทรลเลอร์เบอร์นั้นๆ อย่างลึกซึ้งเท่ากับการเขียนโปรแกรมด้วยภาษาแอสเซมบลี ทั้งนี้เพราะคอมไพลเลอร์จะทำหน้าที่ในส่วนนี้แทน ดังนั้นเมื่อ

ผู้ใช้งานเปลี่ยนเบอร์ไมโครคอนโทรลเลอร์ก็เพียงแต่จัดหาโปรแกรมคอมไพเลอร์ที่เหมาะสมมาใช้งาน และศึกษาสถาปัตยกรรมของไมโครคอนโทรลเลอร์เบอร์ใหม่อีกเพียงเล็กน้อยสามารถใช้งานได้ แต่ข้อเสียของการใช้คอมไพเลอร์คือ ราคาแพงมาก

2.4 ภาษา C กับไมโครคอนโทรลเลอร์ PIC

ทำไมต้องเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ด้วยภาษา C ทั้งๆ ที่ภาษาแอสเซมบลีก็มีความสามารถสูงและใช้งานได้ดีกับไมโครคอนโทรลเลอร์อยู่แล้ว ถ้าเป็นอดีตคำตอบนี้จะเป็นคำตอบที่ถูกต้องที่สุดแต่ปัจจุบันมีทางเลือกที่ดีกว่าในการพัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์ด้วยภาแอสเซมบลีนั่นคือภาษา C ด้วยเหตุผลดังนี้

1. ภาษา C เป็นภาษามาตรฐานไม่ขึ้นกับฮาร์แวร์
2. ภาษา C ใช้หลักการเขียนโปรแกรมแบบสมัยใหม่มีทั้งเป็นโครงสร้างและออบเจ็กต์
3. ภาษา C มีความง่ายในการศึกษาและเรียนรู้และพัฒนาโปรแกรมมากกว่าภาษาแอสเซมบลี
4. ภาษา C มีความสามารถเทียบเท่าหรือใกล้เคียงภาษาแอสเซมบลีและอาจเหนือกว่าในบางด้าน
5. ภาษา C มีความยืดหยุ่นในการโยกย้ายไปใช้งานกับไมโครคอนโทรลเลอร์ตระกูลอื่น
6. ภาษา C มีความเข้ากันได้กับภาษาแอสเซมบลีในระดับซอร์สโค้ด
7. ภาษา C มีเครื่องมือในการพัฒนาโปรแกรมที่ก้าวหน้ามากกว่าด้วย IDE
8. ตัวแปลภาษาหรือคอมไพล์เลอร์และตัวเชื่อมโยงของภาษา C มีการพัฒนาอยู่ตลอดเวลาจนมีความสามารถสูง
9. สามารถพัฒนางานที่มีความซับซ้อนและมีรูปแบบเป็นผังงานง่ายกว่า

ทั้งหมดคือเหตุผลส่วนหนึ่งจากอีกหลายเหตุผลที่ทำให้ภาษา C เป็นตัวเลือกในการพัฒนาโปรแกรมสำหรับไมโครคอนโทรลเลอร์ในปัจจุบันการเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ PIC ด้วยภาษา C จะมีรูปแบบและโครงสร้างทางภาษาเช่นเดียวกับภาษา C มาตรฐานและยังมีส่วนเพิ่มเติมพิเศษเฉพาะสำหรับใช้งานกับไมโครคอนโทรลเลอร์ PIC ซึ่งได้แก่ชุดคำสั่งพิเศษและฟังก์ชั่นพร้อมใช้งานที่มีมากับตัวคอมไพล์เลอร์สำหรับการเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ด้วยภาษา C

2.4.1 โปรแกรมภาษา C

โปรแกรมภาษา C จะประกอบไปด้วย 3 ส่วนหลัก ๆ คือ

1. ฟังก์ชัน
2. ตัวแปร
3. ชุดคำสั่งการทำงาน

โดยฟังก์ชันจะมีอยู่ 2 ส่วนคือ ฟังก์ชันที่สร้างขึ้นมากับฟังก์ชันที่มีมาพร้อมกับตัวคอมไพล์เลอร์โดยฟังก์ชันที่มากับตัวคอมไพล์เลอร์จะเรียกว่า Built – in function นอกจากนี้ใน CCS C คอมไพล์เลอร์ยังได้เพิ่มฟังก์ชันสำเร็จรูปแบบโมดูลฟังก์ชันที่ CCS เขียนขึ้นมาเพื่ออำนวยความสะดวกให้กับผู้ใช้งานสามารถนำโค้ดไปใช้งานได้ทันที เช่น LCD, ระบบบัส 1 สาย, ระบบ I2C และอ่านเขียนหน่วยความจำข้อมูลอีอีพรอม รวมทั้งโค้ดโปรแกรมตัวอย่างใช้งานที่มีมากับ CCS C คอมไพล์เลอร์ที่ครอบคลุมการทำงานติดต่อกับอุปกรณ์ต่าง ๆ

2.4.2 CCS C คอมไพล์เลอร์

เป็นซอฟต์แวร์สำหรับแปลโปรแกรมภาษา C ของไมโครคอนโทรลเลอร์ PIC เป็นรหัสเครื่องหรือแมชชีนโค้ดผลิตโดย Custom Computer Service สหรัฐอเมริกา สามารถดูรายละเอียดผ่านทางเวปไซต์

CCS C คอมไพเลอร์มีหลายรุ่นเพื่อรองรับความต้องการของผู้ใช้งานการติดตั้งโปรแกรมทำได้ง่ายมากด้วยขั้นตอนที่เหมือนกับซอร์ฟแวร์ประยุกต์ทั่วไปที่ทำงานบนระบบวินโดวส์ CCS C คอมไพล์เลอร์แบ่งออกเป็น 2 รุ่นใหญ่ๆ คือ

1. รุ่นทำงานบนวินโดวส์
2. รุ่นบรรทัดคำสั่ง

ถ้าเป็นการใช้งานรุ่น IDE สามารถที่จะเขียนโค้ดโปรแกรมผ่าน Windows IDE ได้โดยตรงและสามารถที่จะคอมไพล์โปรแกรมได้ทันทีรวมทั้งมีคอมไพล์เลอร์แบบบรรทัดคำสั่งหรือ Command Line ให้ใช้งานด้วย แต่ถ้าเป็นรุ่นบรรทัดคำสั่งจะไม่มี Windows IDE ต้องจัดหาโปรแกรมเท็กซ์เอดิเตอร์มาใช้งานในการเขียนโค้ดโปรแกรมเอง ไม่ว่าจะเป็น Notepad ของวินโดวส์, Editor ของ MPLAB หรือจะเป็น Edit Plus ที่เป็นที่นิยมในบ้านเรา ซึ่งก็แล้วแต่ความถนัดของผู้ใช้งานเป็นหลัก

2.4.3 พื้นฐานภาษา C กับ CCS C คอมไพล์เลอร์

โครงสร้างของภาษา C ในรูปแบบมาตรฐาน (ANSI Standard C) จะประกอบไปด้วยรายละเอียดดังนี้

- **พรีโปรเซสเซอร์ไดเร็กตีฟ (PreProcessor directives)**

ชุดคำสั่งที่ใช้ในการจัดการเตรียมข้อมูลสำหรับการประมวลผล โดยก่อนที่จะมีการคอมไพล์หรือแปลตัวโค้ดโปรแกรมให้เป็นภาษาเครื่องนั้นจะมาทำงานในส่วนนี้ก่อนจึงเรียกวา พรีโปรเซสเซอร์สำหรับ CCS C คอมไพล์เลอร์ จะมีไดเร็กตีฟทั้งที่เป็นมาตรฐานเดียวกับ ANSI C และที่เพิ่มเติมเฉพาะสำหรับ CCS C คอมไพล์เลอร์

- **การประกาศ (Declarations)**

ก่อนใช้งานตัวแปรหรือฟังก์ชันต้องมีการประกาศและสร้างตัวแปรหรือฟังก์ชันขึ้นมาก่อน

- **การกำหนดค่า (Definitions)**

การประกาศและจองหน่วยความจำหรือกำหนดค่าให้กับตัวแปรหรือฟังก์ชัน

- **นิพจน์ (Expressions)**

นิพจน์คือการกระทำระหว่างตัวดำเนินการ (Operators) กับตัวถูกดำเนินการหรือโอเปอร์แรนด์ (Operands) เพื่อให้เกิดค่าใดค่าหนึ่ง

- **สเตตเมนต์ (Statements)**

คำสั่งการทำงานคือคำสั่งที่ใช้ในการทำงานตามความต้องการของผู้เขียนโปรแกรม

- **ฟังก์ชัน (Functions)**

ฟังก์ชันคือส่วนประกอบของโปรแกรมที่กำหนดให้ทำงานอย่างใดอย่างหนึ่งจนเสร็จสิ้น โดยที่ฟังก์ชันจะประกอบไปด้วยการประกาศใช้งานตัวแปรการกำหนดค่าให้กับตัวแปร นิพจน์และคำสั่งการทำงาน

- **ฟังก์ชัน main() (Main Function)**

เป็นฟังก์ชันที่จะต้องมีการประกาศทุกครั้งเมื่อมีการเขียนโปรแกรมด้วยภาษา C เพราะการทำงานของโปรแกรมจะเริ่มต้นที่ฟังก์ชันนี้และเป็นฟังก์ชั่นที่ใช้ในการเรียกฟังก์ชันอื่นๆในการทำงาน

2.4.4 ตัวแปรและชนิดข้อมูลในโปรแกรมภาษา C ของไมโครคอนโทรลเลอร์ PIC

นอกจากรายละเอียดโครงสร้างของภาษา C ที่ต้องทำความเข้าใจแล้ว ตัวแปรและชนิดข้อมูลที่ใช้ในการสร้างตัวแปร ก็เป็นส่วนหนึ่งที่สำคัญที่ต้องทำความเข้าใจ ก่อนที่จะเริ่มต้นเขียนโปรแกรมด้วยภาษา C โดยเฉพาะการเลือกใช้ชนิดของข้อมูล เนื่องจากการใช้งานชนิดของข้อมูลก็คือการเลือกใช้งานหน่วยความจำแรมของไมโครคอนโทรลเลอร์เช่นเดียวกัน จึงต้องเลือกใช้ให้เหมาะสมเพื่อนำมาสร้างตัวแปรไม่ว่าจะเป็นทศนิยม, จำนวนเต็ม, ตัวอักขระ เพราะหน่วยความจำข้อมูลแรมของไมโครคอนโทรลเลอร์มีขีดจำกัด

2.4.5 ชนิดของข้อมูล

ในการสร้างตัวแปรขึ้นมาใช้งานจะต้องมีการระบุชนิดของข้อมูลให้กับตัวแปร โดยชนิดของข้อมูลก็จะแบ่งออกเป็น 2 ชนิดใหญ่คือ ชนิดข้อมูลจำนวนตัวเลขและชนิดข้อมูลตัวอักษร

ชนิดข้อมูล	ขนาด	ค่าของข้อมูล
Int1	ตัวเลข 1 บิต	0 หรือ 1
Int8	ตัวเลข 8 บิต	0 ถึง 255
Int16	ตัวเลข 16 บิต	0 ถึง 65,536
Int32	ตัวเลข 32 บิต	0 ถึง 4,294,967,295
char	ตัวอักขระ 8 บิต	ตัวอักขระรหัสแอสกี้
float	ตัวเลขทศนิยม 32 บิต	$3.4x10^{-38}$ ถึง $3.4x10^{38}$
short	ตัวเลข 1 บิต	0 หรือ 1
int	ตัวเลขจำนวนเต็ม 8 บิต	0 ถึง 255
long	ตัวเลข 16 บิต	0 ถึง 65,525
void	ไม่กำหนด	-

ตารางที่ 2.1 การกำหนดตัวแปร

2.4.6 ประเภทของการเก็บข้อมูลตัวแปร (Variable Storage Class)

ตัวแปรและฟังก์ชันที่ใช้งานในภาษา C จะมีรายละเอียดอยู่ 2 ส่วนคือ ชนิดของข้อมูล (Datatype) และการเก็บข้อมูล (Storage Class)

โดยชนิดของข้อมูลจะเป็นการระบุว่าตัวแปรที่สร้างขึ้นมานั้น มีชนิดข้อมูลเป็นอย่างไร เช่น Char, int, float เป็นต้น และอีกส่วนซึ่งโดยปกติแล้วจะมีค่าเริ่มต้นเป็น auto (นั่นคือสามารถปรับเปลี่ยนได้อัตโนมัติ) จึงไม่เขียนระบุไว้ตอนเริ่มต้นสร้างตัวแปร แต่เมื่อต้องการระบุการจัดเก็บและใช้งานหน่วยความจำว่าต้องการใช้งานบนหน่วยความจำโปรแกรมหรือบนรีจิส จึงต้องมีการระบุลงไปในตอนสร้างตัวแปรใช้งาน

คุณสมบัติ	รายละเอียด
Auto(automatic)	เป็นการระบุให้ตัวแปรเป็นประเภทหน่วยความจำอัตโนมัติและสามารถใช้ได้ภายในฟังก์ชั่น
Static	ตัวแปรที่ประกาศจะเป็นแบบทั่วไป (โกบอล : global) มีค่าเริ่มต้นเท่ากับ 0 จะใช้ภายในหรือภายนอกฟังก์ชั่นก็ได้เป็นตัวแปรแบบสถิต
Extern(external)	เป็นการบอกให้รู้ว่าตัวแปรที่ถูกประกาศมีการใช้งานแล้วในฟังก์ชันอื่น(นิยมใช้ในการประกาศข้ามไฟล์ของซอร์สโปรแกรม)
Register	จะเหมือนกับการประกาศแบบ auto แต่จะใช้กับตัวแปรที่มีการใช้งานบ่อยๆ นิยมใช้กับตัวแปรที่เกี่ยวข้องกับการวนลูปเนื่องจากเรียกใช้งานได้อย่างรวดเร็วเพราะเป็นการประกาศในหน่วยความจำความเร็วสูง

ตารางที่ 2.2 คุณสมบัติของตัวแปร

2.4.6.1 ตัวแปร (Variable)

ตัวแปรคือชื่อที่ผู้พัฒนาโปรแกรมประกาศขึ้นมาเพื่อใช้เก็บข้อมูลชั่วคราว การประกาศใช้งานตัวแปรก็คือการประกาศใช้งานหน่วยความจำ การประกาศใช้งานตัวแปรต้องเลือกชนิดของข้อมูลให้เหมาะสมกับข้อมูลที่จะใช้และไม่ควรจะประกาศใช้งานตัวแปรมากเกินความจำเป็น เนื่องจากเมื่อมีการประกาศใช้งานตัวแปร หน่วยความจำแรมจะถูกจองไว้ใช้งานสำหรับตัวแปรนั้นทันที

ในการประกาศใช้งานตัวแปรในภาษา C จะต้องกำหนดชนิดของตัวแปรนั้นว่าเป็นชนิดใดเช่น ตัวแปรชนิด Integer, Character หรือ Floatin เป็นต้น นอกจากนี้ยังใช้คีย์เวิร์ด typedef (ตั้งชื่อใหม่ใช้อ้างอิงแทนชื่อเก่า) เพื่อใช้ในการอ้างอิงถึงชนิดของข้อมูลที่กำหนดมา เช่น ถ้าต้องกานำค่าตัวเลข 2 ตัว มาบวกกันแล้วให้ผลลัพธ์ไปเก็บไว้จะต้องกระทำตามขั้นตอนดังนี้

1. สร้างตัวแปรขึ้นมารองรับ 3 ตัวสำหรับเก็บค่า 3 ค่า
2. นำค่าที่ต้องการบอกกันมาเก็บไว้ในตัวแปรทั้งที่สร้างขึ้น
3. นำค่าตัวแปรทั้งสองตัวที่ได้เก็บค่าไว้มากระทำกันด้วยเครื่องหมายบวก แล้วเก็บไว้ในอีกตัวแปร ก็จะได้ค่า 2 ค่าบวกกันและสามารถนำไปใช้งานได้

2.4.6.2 กฎในการตั้งชื่อตัวแปร

ในการสร้างตัวแปรขึ้นมาใช้งานจะต้องมีกฎที่เกี่ยวข้องกับการตั้งชื่อ เพื่อป้องกันไม่ให้ไปซ้ำกับค่าที่ถูกกำหนดไว้ในตัวคอมไพเลอร์แล้ว หรือสร้างชื่อตัวแปรที่คอมไพเลอร์ไม่สามารถตีความหมายได้ กฎของการตั้งชื่อตัวแปรมีดังนี้

1. ไม่มีตัวอักษรพิเศษประกอบอยู่ในตัวแปร เช่น @, !, #, $, % เป็นต้น
2. ชื่อตัวแปรต้องไม่ขึ้นต้นด้วยตัวเลข
3. ไม่มีช่องว่างระหว่างชื่อของตัวแปร
4. ใช้อักษรตัวเล็กใหญ่ได้ แต่จะให้ความหมายเป็นคนละตัวแปร เช่น ตัวแปร Abc กับ abc เป็นคนละตัวแปรกัน เนื่องจากภาษา C จะคำนึงถึง case sensitive
5. ห้ามตั้งชื่อตัวแปรซ้ำกับคำสงวนหรือคำสั่งที่มีในภาษา C และ Build – in function เช่น ไม่สามารถตั้งชื่อตัวแปรดังนี้ Char, float, int, output_b, printf เป็นต้น

2.4.7 นิพจน์และตัวดำเนินการในโปรแกรมภาษา C ของไมโครคอนโทรลเลอร์ PIC

การเขียนโปรแกรมไม่ว่าจะเป็นภาษา C, BASIC, หรือ PASCAL จะต้องเกี่ยวข้องกับนิพจน์และตัวดำเนินการซึ่งจะอยู่ในรูปแบบของชุดคำสั่ง การเขียนโปรแกรมจึงหนีไม่พ้นที่จะต้องทำความเข้าใจกับนิพจน์และตัวดำเนินการ โดยเฉพาะกับการเขียนโปรแกรมด้วยภาษา C แล้วการใช้งานนิพจน์และตัวดำเนินการใน CCS C คอมไพล์เลอร์ จะมีรูปแบบและลักษณะการใช้งานเช่นเดียวกับ ANSI C จึงสามารถศึกษารายละเอียดเพิ่มเติมได้จากหนังสือภาษา C ที่เป็น ANSI Standard C

2.4.7.1 นิพจน์ (Expression)

นิพจน์คือการนำตัวแปร ค่าคงที่ หรือตัวเลข ที่มากระทำกันด้วยเครื่องหมายดำเนินการ ด้วยเงื่อนไขหนึ่ง เช่น

```
A+B;          //นำค่าในตัวแปร A และ B มาบวกกัน
A==B;         //นำค่าในตัวแปร A และ B มาเปรียบเทียบกันด้วยเครื่องหมายเท่ากับ
A+B/(A*B);    //นำค่าในตัวแปร B หารด้วย A คูณกับ B และนำค่าที่ได้ไปบวกกับ A
```

โดยการเขียนนิพจน์จะต้องมีเครื่องหมายตัวดำเนินการเข้ามาเกี่ยวข้องด้วย และมีระดับความสำคัญของการกระทำในตัวดำเนินการตามหลักระดับความสำคัญของตัวดำเนินการ

2.4.7.2 ตัวดำเนินการ (Operators)

เครื่องหมายดำเนินการคือ ตัวกระทำหรือเครื่องหมายการทำที่มีผลต่อข้อมูลหรือตัวแปร ในภาษา C จะแบ่งออกเป็น 5 ประเภทใหญ่คือ

1. เครื่องหมายดำเนินการทางคณิตศาสตร์ (Arithmetic & Unary Operators)
2. เครื่องหมายดำเนินการสัมพันธ์ทางลอจิก (Relational and Logical Operators)
3. เครื่องหมายดำเนินการกำหนดค่าหรือให้ค่า (Assignment Operators)
4. เครื่องหมายดำเนินการแก้ไข (Conditional Operator)
5. เครื่องหมายดำเนินการทางบิต (Bitwise Operator)

2.4.7.3 เครื่องหมายดำเนินการทางคณิตศาสตร์

ในภาษา C จะมีเครื่องหมายที่ใช้ดำเนินการเกี่ยวกับคณิตศาสตร์ดังนี้

เครื่องหมายดำเนินการ	การกระทำ	ตัวอย่าง
+	การบวกค่า	A+B
–	การลบค่า	A-B
*	การคูณค่า	A*B
/	การหารค่า	A/B
%	การหารค่าเอาเฉพาะเศษ	A%B
-	จำนวนลบ	-A

--	ลดค่า	--A,A--
++	เพิ่มค่า	++A,A++

ตารางที่ 2.3 เครื่องหมายดำเนินการทางคณิตศาสตร์

2.4.7.4 เครื่องหมายดำเนินการสัมพันธ์และลอจิก

เครื่องหมายดำเนินการในลักษณะนี้จะเกี่ยวข้องกับเรื่องของการเปรียบเทียบค่าเพื่อแสดงผลลัพธ์ของการเปรียบเทียบค่าว่าเป็นจริงหรือเป็นเท็จโดยที่เครื่องหมาย &&, // และ ! จะเกี่ยวข้องกับตัวดำเนินการทางลอจิกดังมีรายการต่อไปนี้

เครื่องหมายดำเนินการ	การกระทำ	ตัวอย่าง
<	น้อยกว่า	A<B
<=	น้อยกว่าหรือเท่ากับ	A<=B
>	มากกว่า	A>B
>=	มากกว่าหรือเท่ากับ	A>=B
==	เท่ากับ	A==B
!=	ไม่เท่ากับ	A!=B
&&	และ	A&&B
||	หรือ	A||B
!	ไม่ใช่	!A

ตารางที่ 2.4 เครื่องหมายดำเนินการสัมพันธ์และลอจิก

2.4.7.5 เครื่องหมายดำเนินการกำหนดค่าหรือให้ค่า

เครื่องหมายดำเนินการที่ใช้ในการกำหนดค่าหรือให้ค่าของข้อมูลหรือตัวแปรหรือการกระทำด้วยนิพจน์จากทางขวาของเครื่องหมาย = ให้กับตัวแปรที่อยู่ทางซ้ายเช่น

```
A=20;                    //กำหนดค่า 20 ให้กับตัวแปร A
```

A=A+B; //นำค่าใน A บวกกับค่าใน B แล้วกำหนดให้กับ A

2.4.7.6 เครื่องหมายดำเนินการเงื่อนไข

เครื่องหมายดำเนินการเงื่อนไข ? : นี้เป็นเครื่องหมายดำเนินการเลือกอย่างใดอย่างหนึ่งโดยดูจากเงื่อนไขของการดำเนินการลอจิกตามตัวอย่าง

X=(A>B)? 0 : 150; //ค่าของ X จะเท่ากับ 150

Y=(A<B)? 0 : 150; //ค่าของ Y จะเท่ากับ 0

2.4.7.7 ความสัมพันธ์ของเครื่องหมายดำเนินการ

เครื่องหมายดำเนินการทั้งหมดมีการกำหนดระดับความสำคัญทั้งนี้เพื่อประโยชน์ในการลำดับความสำคัญในการทำงานในกรณีที่มีเครื่องหมายดำเนินการหลายตัวในเงื่อนไข ดังนี้

เครื่องหมายดำเนินการ	ระดับความสำคัญในการกระทำ
-,++,--,!,sizeof, (type)	ขวา ไป ซ้าย
*,/,%	ซ้าย ไป ขวา
+,-	ซ้าย ไป ขวา
<,<=,>,>=	ซ้าย ไป ขวา
==,!=	ซ้าย ไป ขวา
&&	ซ้าย ไป ขวา
//	ซ้าย ไป ขวา
?:	ขวา ไป ซ้าย
=,+=,-=,*=,/=,%=	ขวา ไป ซ้าย

ตารางที่ 2.5 ความสัมพันธ์ของเครื่องหมายดำเนินการ

2.4.7.8 เครื่องหมายดำเนินการทางบิต

เครื่องหมายดำเนินการทางบิตข้อมูลจัดเป็นเครื่องหมายที่มีความสำคัญมากโดยเฉพาะกับการเขียนโปรแกรมด้วยภาษา C กับงานไมโครคอนโทรลเลอร์ซึ่งหนีไม่พ้นเรื่องของบิตและไบต์ข้อมูล รายละเอียดของตัวดำเนินการมีดังนี้

เครื่องหมาย	การกระทำ	ตัวอย่าง
&	การแอนด์ข้อมูล	A&B
|	การออร์ข้อมูล	A|B
^	การเอ็กคลูซีพออร์ข้อมูล	A^B
<<	เลื่อนบิตไปทางซ้าย	A<<B
>>	เลื่อนบิตไปทางขวา	A>>B
~	การกลับค่าบิต	~A

ตารางที่ 2.6 เครื่องหมายดำเนินการทางบิต

2.4.8 การใช้งานพอร์ตอนุกรม USART ในไมโครคอนโทรลเลอร์ PIC ด้วยโปรแกรมภาษา C

พอร์ตอนุกรมจัดเป็นพอร์ตสื่อสารข้อมูลที่นิยมใช้งานทั้งบนเครื่องคอมพิวเตอร์และไมโครคอนโทรลเลอร์โดยเฉพาะนำมาใช้งานในการสื่อสารระหว่างเครื่องคอมพิวเตอร์กับอุปกรณ์ไมโครคอนโทรลเลอร์สำหรับไมโครคอนโทรลเลอร์ PIC หลายๆ เบอร์ได้เตรียมโมดูลการเชื่อมต่อพอร์ตอนุกรมไว้ให้แล้วที่เรียกว่า USART (Universal Synchronous Asynchronous Receiver Transmitter) ที่มีคุณสมบัติในการทำงานเป็นตัวรับและตัวส่งข้อมูลในแบบอะซิงโครนัสหรือซิงโครนัสก็ได้ นอกจากนี้ยังสามารถกำหนดความเร็วในการรับส่งข้อมูลได้ด้วย

ใน CCS C คอมไพล์เลอร์ ได้เตรียมใส่วนติดต่อและใช้งานพอร์อนุกรมไว้ให้แล้ว ทั้งในส่วนของการประกาศไดเร็กตีฟเพื่อเรียกใช้งานพอร์ตอนุกรม และฟังก์ชันพร้อมใช้งานเพื่อใช้ในการจัดการข้อมูลอนุกรมซึ่งมีการทำงานในลักษณะโพลลิ่ง คือ การเขียนโปรแกรมตักจับข้อมูลที่ผ่านเข้ามาทางพอร์ตอนุกรมตลอดเวลาในลูปของโปรแกรมกลับ และแบบที่สองคือการอินเตอร์รัปต์ของพอร์อนุกรม โดยอาศัยฟังก์ชันที่เกี่ยวข้องกับการอินเตอร์รัปต์วิธีที่สองนี้เป็นการใช้อินเตอร์รัปต์ในการดักจับ

สัญญาณข้อมูลที่ผ่านเข้ามาทางพอร์ตอนุกรมแทน ทำให้ไม่ต้องเขียนโปรแกรมจับข้อมูลทางพอร์ตตลอดเวลาเหมือนวิธีแรก

2.4.9 ไมโครคอนโทรลเลอร์ตระกูล PIC

PIC คือไมโครคอนโทรลเลอร์ตระกูลหนึ่ง ผลิตโดยบริษัทไมโครชิพโดย PIC ย่อมาจากคำว่า Peripheral Interface Controller ซึ่งภายใน PIC ประกอบด้วยหน่วยความจำ โปรแกรมหน่วยความจำข้อมูลพอร์ตอินพุต พอร์ตเอาต์พุตทำให้ PIC เหมือนไมโครคอมพิวเตอร์ตัวหนึ่ง นอกจากนี้ภายใน PIC ยังมี I^2C, PWM, A/D ซึ่งถือว่าเป็นคุณสมบัติพิเศษของ PIC ที่แตกต่างจากไมโครคอนโทรลเลอร์ตัวอื่นๆ การรวมทุกสิ่งทุกอย่างไว้ในตัว PIC ทำให้นำมาใช้งานได้ง่ายและสะดวก เพียงต่อไฟเลี้ยงป้อนสัญญาณนาฬิกาและเขียนโปรแกรมควบคุม PIC ก็สามารถควบคุมอุปกรณ์ภายนอกผ่านพอร์ตอินพุตและพอร์ตเอาต์พุตได้

2.4.9.1 ชนิดของไมโครคอนโทรลเลอร์ตระกูล PIC

ไมโครคอนโทรลเลอร์ตระกูล PIC ที่นิยมใช้กันในปัจจุบันมีด้วยกันหลายเบอร์โดยแยกเป็นกลุ่มได้ดังนี้ PIC10, PIC12, PIC14, PIC16, PIC17, PIC18 ซึ่งแต่ละกลุ่มยังแยกเป็นเบอร์ต่างๆอีกหลายเบอร์แต่กลุ่มที่ได้รับความนิยมมี 3 กลุ่มคือ PIC16, PIC17, PIC18 ทั้ง 3 กลุ่มนี้จะมีคุณสมบัติต่างกันออกไปเช่น ขนาดของหน่วยความจำ จำนวนพอร์ต สถาปัตยกรรม อาจแบ่งตามชนิดของหน่วยความจำโปรแกรมได้ 3 กลุ่ม คือ

1. มีหน่วยความจำโปรแกรมได้ครั้งเดียว
   หน่วยความจำโปรแกรมกลุ่มนี้เรียกว่า OTP (One Time Programmable) เป็นชิพที่ราคาถูกที่สุดเพราะสามารถโปรแกรมได้ครั้งเดียวไม่สามารถแก้ไขได้อีกจึงเหมาะกับงานที่ตรวจสอบแล้วว่าไม่มีข้อผิดพลาดใดๆ แล้วชิปแบบ OTP จะมีตัวอักษร C แสดงบนตัวชิป
2. มีหน่วยความจำโปรแกรมได้หลายครั้งแบบอีพรอม
   หน่วยความจำโปรแกรมกลุ่มนี้ เรียกว่า EPROM (Erasable Programmable ROM) เป็นชิปที่หน่วยความจำโปรแกรมสามารถเขียนและลบได้โดยใช้แสงอัลตราไวโอเลตหรือยูวี ซึ่งด้านบนของชิปจะมีกรอบกระจกเพื่อให้แสงยูวีส่องผ่านเข้าไปในตัวชิปโดยใช้เวลาประมาณ 5-10 นาที การลบโปรแกรมย่อยจะทำให้เกิดอาการด้านไม่สามารถโปรแกรมได้ ชิปแบบนี้จะมีอักษร JW แสดงบนตัวชิป

3. ที่มีหน่วยความจำโปรแกรมได้หลายครั้งแบบแฟลชหรืออีอีพรอม
หน่วยความจำโปรแกรมนี้เรียกว่า Flash หรือ EEPROM เป็นแบบที่นิยมมากที่สุด เนื่องจากหน่วยความจำโปรแกรมสามารถอ่านและลบได้หลายพันครั้งทำให้สะดวกในการแก้ไขปรับปรุง ชิปแบบแฟลชจะมีอักษร F แสดงบนชิป

2.4.9.2 คุณสมบัติของไมโครคอนโทรลเลอร์ตระกูล PIC ในแต่ละกลุ่ม

การแบ่งกลุ่มของ PIC จะแบ่งตามตัวเลขที่ขึ้นต้น โดยแต่ละกลุ่มจะมีเบอร์ต่างๆ แยกออกไปอีกหลายเบอร์ซึ่งแบ่งได้เป็น 6 กลุ่ม คือ

1. ไมโครคอนโทรลเลอร์ตระกูล PIC16C5X
เป็นชิปที่ผลิตออกมาในยุคแรกๆ เหมาะกับงานที่ไม่ซับซ้อน มีคำสั่งใช้งานที่เป็นภาษาแอสแซมบลี 33 คำสั่ง มี I/O, Timer, Watchdog ไม่มี I^2C หรือ Serial ซึ่งต้องเขียนโปรแกรมขึ้นเอง ชิปในกลุ่มนี้มีหลายเบอร์แต่ละเบอร์จำนวนขาและขนาดของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลต่างกัน
2. ไมโครคอนโทรลเลอร์ตระกูล PIC12CXXX
บริษัทไมโครชิปได้พัฒนาและปรับปรุงจนเป็นตระกูล PIC16XXX โดยเป็นชิปแบบ 18 ขา มีคำสั่งภาษาแอสเซมบลี 35 คำสั่ง มี Timer เพิ่มขึ้น บางเบอร์มีมากกว่า 1 ตัว มีพอร์ต 12C และ USART ทำให้สามารถติดต่อกับอุปกรณ์ภายนอกได้สะดวกขึ้นเขียนโปรแกรมควบคุมง่ายขึ้น นอกจากนี้ยังได้ทำการเพิ่มขนาดของหน่วยความจำให้มีขนาดใหญ่ขึ้นด้วย
3. ไมโครคอนโทรลเลอร์ตระกูล PIC12CXXX และ PIC12FXXX
เป็นชิปขนาดเล็กที่มีเพียง 8 ขาเท่านั้นเหมาะกับงานเล็กๆ มีคำสั่งภาษาแอสเซมบลี 33 และ 35 คำสั่ง สำหรับจุดเด่นของ PIC กลุ่มนี้ คือ มีสัญญาณนาฬิกาขนาด 4MHz อยู่ภายในชิป ทำให้ไม่ต้องต่อภายนอก และมีหน่วยความจำข้อมูลภายในเป็นแบบ EEPROM แต่ส่วนของหน่วยความจำโปรแกรมยังเป็นแบบ OTP และ EPROM หลังจากนั้นได้ผลิต PIC16FXXX แล้วทางบริษัทได้ผลิต PIC12FXXX ขึ้นมาซึ่งเป็นแบบแฟลช ทำให้สามารถเขียนและลบได้หลายครั้ง
4. ไมโครคอนโทรลเลอร์ตระกูล PIC17CXXX

เป็นชิปที่ผลิตออกมาพร้อมกับ PIC16CXXX แต่ต่างกันที่ PIC17CXXXจะมีความสามารถสูงกว่ามีจำนวนขามากกว่ามีคำสั่งภาษาแอสเซมบลี 58 คำสั่ง การคูณ หาร รวมทั้งขนาดของหน่วยความจำโปรแกรมหน่วยความจำข้อมูลที่ขนาดใหญ่ขึ้น และยังสามารถต่อกับหน่วยความจำภายนอกได้อีกด้วย

5. ไมโครคอนโทรลเลอร์ตระกูล PICFXXX
   เป็นชิปที่ได้รับความนิยมมากเพราะเป็นชิปที่ความจำโปรแกรมเป็นแบบแฟลช แล้วมีหน่วยความจำข้อมูลแบบ EEPROM ทำให้สามารถพัฒนาโปรแกรมได้ง่าย ซึ่ง PIC16FXXX สนับสนุนการทำงานแบบอินเซอร์กิตตีบักเกอร์ทำให้ไม่ต้องซื้ออีกพรอมอีมูเลเตอร์ที่มีราคาแพง มีคำสั่ง 35 คำสั่ง และมีวงจรแปลงสัญญาณอนาล็อกเป็นดิจิตอลขนาด 10 บิตอยู่ภายใน
6. ไมโครคอนโทรลเลอร์ตระกูล PIC18CXXX และ 18FXXX
   เป็นอีกกลุ่มที่ได้รับความนิยมเนื่องจากมีความสามารถใหญ่กว่า PIC16FXXX ไม่ว่าจะเป็นคำสั่งภาษาแอสเซมบลีที่มีถึง 77 คำสั่ง หรือหน่วยความจำโปรแกรมที่มีขนาดใหญ่ทำให้สามารถรองรับการเขียนโปรแกรมภาษาซีได้

2.5 การใช้งานไมโครคอนโทรลเลอร์กับพอร์ตอนุกรม

การติดต่อกับพอร์ตอนุกรมนั้นจะยากกว่าการติดต่อกับพอร์ตขนาน การสื่อสารของอุปกรณ์ที่ต่อกับพอร์ตอนุกรมจะถูกเปลี่ยน (Convert) เป็นสัญญาณแบบขนานแล้วจึงนำไปประมวลต่อ ซึ่งจะใช้ Universal Asynchronous Receive/Transmitter (UART) เป็นตัวทำหน้าที่ ส่วนทางด้านโปรแกรมก็มีรีจิสเตอร์ที่ต้องจัดการมากกว่า Standard Parallel Port (SPP) อีกหลายตัว

ข้อดีของพอร์ตอนุกรม

1. ระยะของสายอนุกรมสามารถมีความยาวได้มากกว่าสายของขนานมาก ทั้งนี้เพราะสัญญาณของพอร์ตอนุกรม ซึ่งส่วนใหญ่ใช้มาตรฐาน RS-232C จะมีค่า -3 Volt ถึง -15 Volt สำหรับ Logic “1” หรือ “Mark” และมีค่า +3 Volt ถึง +15 Volt สำหรับ Logic “0” หรือ “Space” (สำหรับช่วง +3 Volt ถึง +3 Volt เป็นช่วง Underfined) ส่วนสัญญาณของสายขนานนั้น Logic “1” จะมีค่า +5 Volt และ logic “0” จะมีค่า 0 Volt ทำให้สัญญาณของสายอนุกรมสามารถรับการสูญเสียของสาย (Cable Loss) ได้มากว่าสัญญาณของสายขนาน

ปกติสายขนานจะไปได้เพียง 5 ฟุต ส่วนสาย RS-232 จะไปได้ถึง 50 ฟุตที่ความเร็วสูงสุดของมัน

2. สายอนุกรมจะใช้จำนวนสายไฟน้อยกว่าสายขนานถ้าต่อในลักษณะ Null Modem จะใช้สายเพียง 3 เส้น ขณะที่แบบขนานจะต้องใช้สาย 19 ถึง 25 เส้น
3. การสื่อสารแบบไร้สายเช่นการใช้ Infra Red การส่งพร้อมกันทีละ 8 บิต แบบขนาน จะทำให้ไม่สามารถแยกแยะได้ว่า Bit ใดเป็น Bit0 หรือ Bit.... เป็นต้น ปัจจุบันอุปกรณ์ IrDA มีความเร็วไม่ต่ำกว่า 115.2K Baud แต่มี Pulse length เพียง 3/16 ของ RS-232 เพื่อประหยัดพลังงาน เพราะส่วนมากใช้ในอุปกรณ์แบบพกพาเช่น Laptop หรือ Palmtop
4. ปัจจุบันได้แก้ไขให้รองรับกับเทคโนโลยีใหม่ได้ จึงมีการปรับปรุงถึง RS-232E ซึ่งมีรายละเอียดอีกหลายอย่าง

รูปที่ 2.9 พอร์ตอนุกรม

Pin	Name	RS232	V.24	Dir	Description
1	CD	CF	109	←	Carrier Detect
2	RXD	BB	104	←	Receive Data
3	TXD	BA	103	→	Transmit Data
4	DTR	CD	108.2	→	Data Terminal Ready
5	GND	AB	102	—	Signal Ground
6	DSR	CC	107	←	Data Set Ready
7	RTS	CA	105	→	Request to Send
8	CTS	CB	106	←	Clear to Send
9	RI	CE	125	←	Ring Indicator

ตารางที่ 2.7 แสดงถึงหน้าที่ของขาต่าง ๆ

Abbreviation	Full Name	Originator	Function
TD	Transmit Data	DTE	Serial data output (TXD) from DTE.
RD	Receive Data	DCE	Serial data input (RXD) to DTE.
CTS	Clear To Send	DCE	Tell DTE that DCE is ready to exchange data.
(D)CD	(Data) Carrier Detect	DCE	Carrier from remote DCE is detected.
DSR	Data Set Ready	DCE	Tell DTE that DCE is ready to establish a link.
DTR	Data Terminal Ready	DTE	Tell DCE that DTE is ready to establish a link.
RTS	Ready To Send	DTE	Tell DCE that DTE is ready to exchange data.
RI	Ring Indicator	DCE	Ringing signal from the phone line is detected.

ตารางที่ 2.8 แสดงสัญญาณต่างๆ ที่ส่งในรูปแบบอนุกรม

2.5.1 หน้าที่ของสัญญาณต่างๆ มีดังนี้

Protective ground เป็นจุดที่ต่อกับตัวเปลือกของอุปกรณ์ และไม่ต่อกับสัญญาณใดๆ ในระบบ เพื่อใช้ต่อลงดิน เป็นการป้องกันอันตรายจากไฟลัดวงจร และใช้เป็น Shield ป้องกันการรบกวน

Signal ground หรือ Common return เป็นจุดสำคัญที่สุดที่ต้องมีในระบบ RS-232 เพราะเป็นจุดอ้างอิงของทุกสัญญาณ ยกเว้น Protective ground

Transmit Data เป็นสัญญาณที่ส่งจาก DTE ไปยัง DCE ในขณะที่ไม่ได้ส่งข้อมูลมันจะมีสถานะเป็น Logic "1" หรือ "Mark" หรือ "Off" หรือ -5 V ถึง -15 V ที่ด้านส่ง (DTE) หรือ -3 V ถึง -15

V ที่ด้านรับ (DCE) การส่งข้อมูลจะเกิดขึ้นได้ต้องมีสัญญาณควบคุมที่เกี่ยวข้อง “On” ก่อนคือสัญญาณ RTS, CTS, (D)CD, DTR และ DSR

Receive Data เป็นสัญญาณข้อมูลจาก DCE มายัง DTE ในขณะที่ไม่ได้ส่งข้อมูลมันจะมีสถานะเป็น Logic “1” หรือ “Mark” หรือ “Off” หรือ -5 V ถึง -15 V ที่ด้านส่งหรือ -3 V ถึง -15 V ที่ด้านรับในขณะที่ไม่ได้ส่งข้อมูลมันจะมีสถานะเป็น Logic “1” หรือ “Mark” หรือ “Off” หรือ -5 V ถึง -15 V ที่ด้านส่ง (DEC) หรือ -3 V ถึง -15 V ที่ด้านรับ (DTE) กรณีที่เป็นการสื่อสารแบบ Half-duplex สัญญาณ RD จะอยู่สถานะ “Off” ขณะที่สัญญาณ RTS อยู่ในสถานะ “On” และสัญญาณ RD จะยังคงอยู่ในสถานะ “Off” อีกชั่วระยะหนึ่งหลังจากที่สัญญาณ RTS เปลี่ยนจากสถานะ “On” มาเป็น “Off” แล้วเพื่อให้การรับ – ส่งข้อมูลเสร็จสมบรูณ์

Request to Send เป็นสัญญาณจาก DTE ส่งไปให้ DCE เพื่อขอส่งข้อมูลไป ปกติจะอยู่ในสถานะ “Off” เมื่อต้องการส่งข้อมูลจะเปลี่ยนเป็นสถานะ “On” จนกว่าการส่งเสร็จสิ้นจึงเปลี่ยนกลับมาที่สถานะ “Off” ตามเดิม ทั้งนี้ทาง DTE ต้องได้รับสัญญาณ CTS จึงจะสามารถส่งข้อมูล TD ไปยัง DCE ได้ และสัญญาณ RTS ที่กลับสู่สถานะ “Off” จะไม่สามารถเปลี่ยนเป็น “On” ใหม่ ขณะที่สัญญาณ CTS อยู่ในสถานะ “On” ต้องรอจนกว่าสัญญาณ CTS เปลี่ยนมาอยู่ในสถานะ “Off” ก่อนเพื่อป้องกันการเกิด Overrun

Clear to Send เป็นสัญญาณที่ DCE ส่งให้ DTE เพื่อแจ้งว่าพร้อมรับการส่งข้อมูลจาก DTE

Data set ready เป็นสัญญาณที่ DCE ส่งให้ DTE เพื่อแจ้งว่า DCE สามารถเชื่อมโยงไปยังปลายทางและพร้อมที่จะติดต่อแล้ว

Data Terminal ready เป็นสัญญาณที่ DTE ส่งให้ DCE เพื่อแจ้งว่า DTE พร้อมหรือต้องการจะติดต่อสื่อสาร ซึ่งสัญญาณ DTR นี้ต้องเกิดก่อนทาง DCE จึงจะทำการติดต่อไปยังปลายทางและเมื่อติดต่อได้แล้วจึงส่งสัญญาณ DSR มายัง DTE เพื่อแจ้งให้รู้ว่าพร้อมรับการสื่อสารแล้ว และถ้า DTR เปลี่ยนเป็น Off แปลว่า DTE ไม่ต้องการติดต่อสื่อสารแล้วทาง DCE ก็จะปิดช่องสื่อสารและเปลี่ยนสัญญาณ DSR เป็น Off ทันที คู่สัญญาณระหว่าง RTS กับ CTS เป็นเรื่องของความพร้อมเกี่ยวกับช่องสื่อสารระหว่าง DTE กับ DCE ส่วนคู่สัญญาณ DTR กับ DSR เป็นเรื่องของความพร้อมเกี่ยวกับตัวอุปกรณ์ DTE กับ DCE

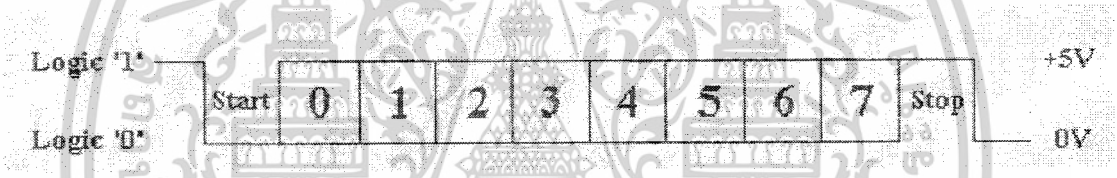
Data carrier detect เป็นสัญญาณที่ DCE ส่งให้ DTE เพื่อแจ้งว่าได้รับสัญญาณพาหะจาก DCE ที่อยู่อีกด้านหนึ่งของการสื่อสาร ซึ่งหมายความว่า ช่องการสื่อสารระหว่าง DCE ทั้ง 2 ไม่ขาดตอน

พร้อมที่จะทำการสื่อสารได้ ซึ่งอุปกรณ์ DTE หรือโปรแกรมที่ควบคุมการสื่อสารมักจะตรวจ สัญญาณนี้ถ้าไม่อยู่ที่ on แสดงว่าช่องการสื่อสารขาด ก็จะไม่ทำการรับหรือส่งข้อมูล

Ring Indicator เป็นสัญญาณจาก DCE แจ้งให้ DTE รู้ว่ามีการเรียกจาก DCE ที่อยู่อีกด้านหนึ่งของการสื่อสาร ซึ่งมักจะใช้ในระบบ automatic answering

2.5.2 รูปคลื่น สัญญาณ RS-232

การสื่อสารโดย RS-232 เป็นการสื่อสารแบบ asynchronous หมายความว่าสัญญาณ clock ที่ใช้ควบคุมจังหวะไม่ได้ส่งไปพร้อมกับ Data แต่จะใช้ start bit เป็นตัว sync. ในแต่ละ word ของการสื่อสารและใช้สัญญาณ clock ภายในของแต่ละด้านเป็นตัวให้จังหวะเอง



รูปที่ 2.10 รูปคลื่นของสัญญาณที่ส่ง

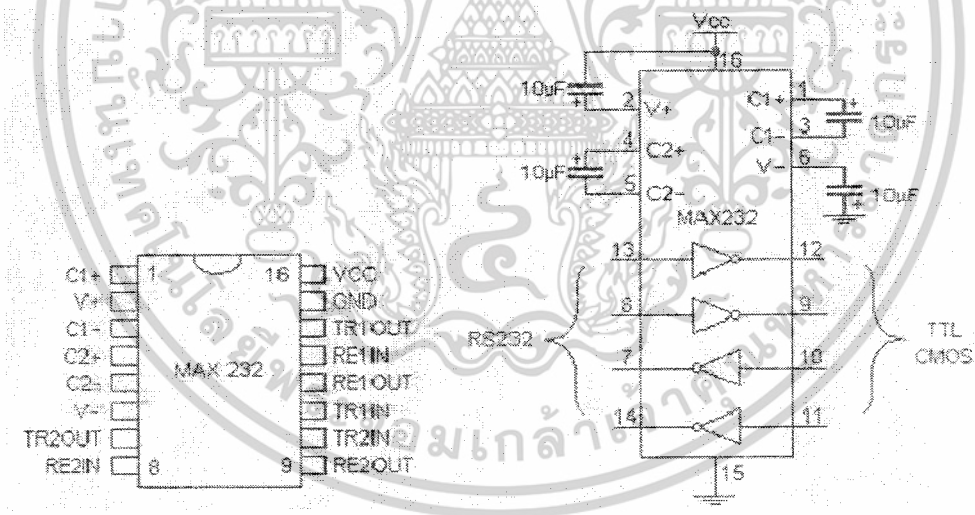
แสดงลักษณะของสัญญาณจาก UART เมื่อใช้ format แบบ 8N1 คือ 8 data bits ไม่มี parity bit และมี 1 stop bit ขณะที่ idle จะอยู่ในสถานะ “Mark” หรือ logic “1” การส่งจะเริ่มจากการส่ง start bit คือ logic “0” และตามด้วย LSB bit จนหมด data bits และถ้ามี parity bit ก็จะส่งที่จุดนี้แล้วลงท้ายด้วย stop bit ซึ่งมีค่าเป็น logic “1” ในรูปได้แสดง bit ที่ต่อถัดจาก stop bit ซึ่งมีค่าเป็น logic “0” หมายความว่าเป็น start bit การส่ง word ถัดไป แต่ถ้ายังไม่มีการส่ง word ถัดไปก็ต้องอยู่ในสถานะของ logic “1” ซึ่งเป็นสถานะของ idle และถ้าสายอยู่ในสถานะของ logic “0” นานกว่าเวลาของการส่ง 1 full word ระบบจะถือว่าเป็นสัญญาณ “Break” เพื่อหยุดการสื่อสารดังนั้นต้องไม่ลืมที่จะสั่งในสายกลับสู่สถานะ idle เมื่อสิ้นสุดการส่ง การรับ – ส่งข้อมูลในลักษณะนี้เรียกว่าแบบ frame คือมีกรอบปิดล้อมข้อมูลไว้ด้วย start bit และ stop bit

2.5.3 ตัวแปลงสัญญาณ RS-232

สัญญาณ RS-232 มีค่าแรงไฟต่างจากที่ใช้ใน UART ดังนั้นจึงต้องมีตัวแปลงสัญญาณเพื่อแปลงระดับสัญญาณให้เหมาะสมก่อนที่ จะเชื่อมต่อกับพอร์ตอนุกรม หรือ RS-232 port ของคอมพิวเตอร์

สัญญาณ RS-232 นั้น logic "0" จะมีค่า +3 V ถึง +25 V และ logic "1" จะมีค่า -3 V ถึง -25 V ส่วนค่าระหว่าง -3 V ถึง +3 V เป็นค่า underfined ระดับสัญญาณนี้ใช้กับทุกสัญญาณไม่ใช่เฉพาะสัญญาณรับ – ส่งข้อมูลเท่านั้นแต่ยังรวมถึงสัญญาณควบคุมต่าง ๆ เช่น DTR, RTS, CTS เป็นต้น

IC ที่ใช้มักจะเป็นเบอร์ 1488 (RS-232 Driver) และ 1489 (RS-232 Receiver) โดยภายในแต่ละตัวจะประกอบด้วย inverter 4 ตัวและต้องการไฟเลี้ยง 2 ชุดคือ +7.5 V ถึง +15 V และ -7.5 V ถึง -15 V ซึ่งอาจจะมีปัญหาในเครื่องที่มีไฟเลี่ยง +5 V เพียงชุดเดียว แต่ก็ยังมี IC อีกตัวหนึ่งคือเบอร์ MAX-232 ซึ่งมีวงจร charge pump สามารถสร้างไฟ +10 V และ -10 V จากไฟ +5 V ได้พร้อมทั้งมี 2 Tx และ 2 Rx อยู่ใน package เดียวกัน และรองรับ band rate ได้ถึง 120 Kbps จึงสะดวกมากเพราะใช้ IC เพียงตัวเดียว รูปข้างล่างคือ MAX-232



รูปที่ 2.11 แสดงโครงสร้างภายในและตำแหน่งขาต่างๆ ของ MAX-232

ส่วนการที่เราจะนำข้อมูลมาใช้งานก็ต้องแปลงเป็น Parallel ก่อนซึ่งเป็นหน้าที่ของ UART ซึ่งปัจจุบันไมโครคอนโทรลเลอร์มักจะมี serial communication interface (SCI) อยู่ในตัว แต่อาจจะมีงานบางอย่างที่ไม่ได้ใช้ไมโครคอนโทรลเลอร์และต้องการประมวลผลข้อมูลกับพอร์ตอนุกรมเช่น ต่อ ADC เข้ากับ UART หรือต่อ LCD display เข้ากับ Serial comm. ก็ต้องใช้ UART ช่วย เช่น เบอร์ 8550 หรือ

16550A หรือเบอร์อื่นๆ ที่ได้กล่าวมาแล้ว แต่มี UART อีกพวกหนึ่งที่แยก Tx bus กับ Rx bus ออกจากกันทำให้มีความคล่องตัวมากขึ้น

2.6 การสื่อสารข้อมูล

การสื่อสารข้อมูลมีจุดหมายในการส่ง หรือการถ่ายทอดข่าวสารจากจุดหนึ่งไปยังอีกจุดหนึ่งอย่างถูกต้องเหมือนกับการที่ด้านส่งออกมา ในการสื่อสารรูปแบบใดๆ จะต้องประกอบไปด้วยส่วนประกอบหลักเบื้องต้น 3 ส่วนดังนี้

1. แหล่งต้นกำเนิดข่าวสาร (Soure)
2. ตัวกลางในการสื่อสาร (Media)
3. แหล่งรับข่าวสาร (Receiver)

2.6.1 ประเภทของการสื่อสารข้อมูล

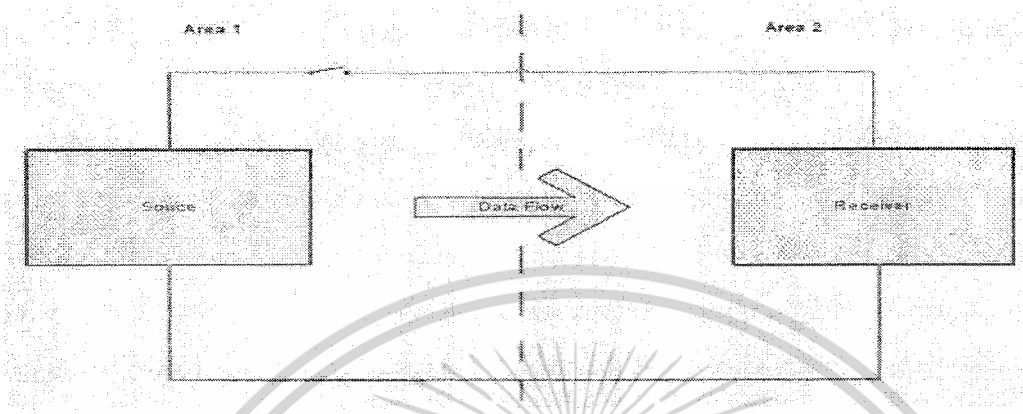
ข้อมูลในระบบของการสื่อสารเขียนแสดงได้ด้วยค่าในระบบเลขฐานสอง โดยใช้ค่าตัวเลข 0 หรือ 1 มาประกอบของกันเป็นรหัส แต่ในการส่งเราอาศัยการส่งทางไฟฟ้า ข้อมูลจะถูกเปลี่ยน ให้อยู่ในรูปแบบทางไฟฟ้าโดยใช้ค่าสัญญาณไฟฟ้า 2 ระดับคือสูงและต่ำ ในการวัดอัตราเร็วในการส่งได้จากจำนวนบิตที่ส่งไปในหน่วยเวลาโดยทั่วไปใช้หน่วย Bit per second (bps) ซึ่งในระบบการสื่อสารข้อมูลนั้นอาจจะมีขนาดใดก็ได้แต่ในการส่งนั้นจะต้องมีการกำหนดจุดเริ่มต้น และจุดสิ้นสุดของการแต่ละการส่ง ตามปกติจะจัดแบ่งข่าวสารที่จะส่งเป็นบล็อกๆ ซึ่งหนึ่งบล็อกคือกลุ่มของบิตจำนวนหนึ่งที่ถูกส่งออกไปเป็นหน่วยเดียวกัน โดยมีการนำกลุ่มบิตนั้นไปผ่านกระบวนการบางอย่าง เพื่อใช้ในการควบคุมข้อผิดพลาดที่อาจจะเกิดขึ้นเราสามารถจำแนกวิธีการส่งข้อมูลได้หลายแบบตามคุณสมบัติต่างๆ ดังนี้

2.6.1.1 การจำแนกตามทิศทางในการส่งข้อมูล

สามารถแบ่งการส่งข้อมูลออกได้เป็น 3 ชนิด ดังนี้

2.6.1.1.1 การรับส่งข้อมูลทางเดียว (Simplex)

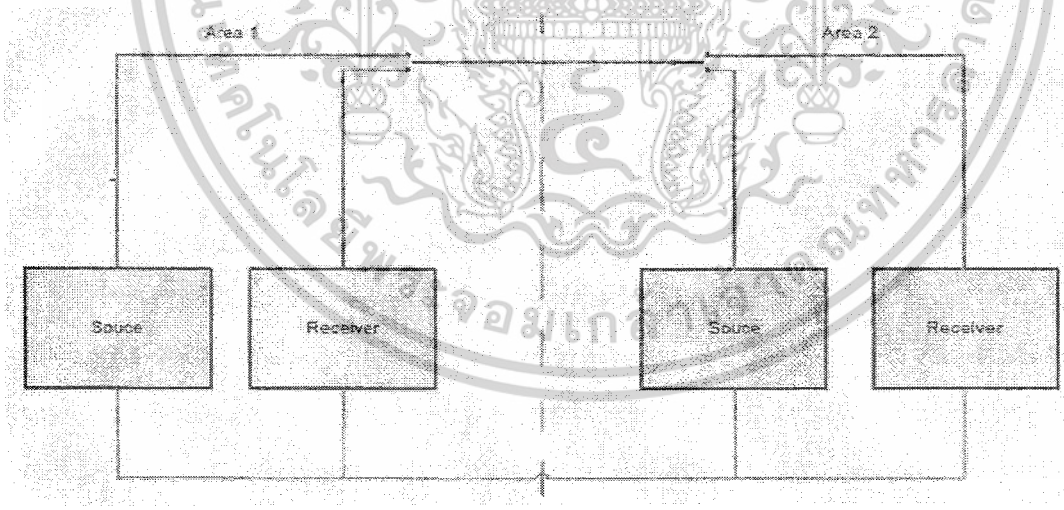
เป็นการสื่อสารทางเดียว ที่เห็นได้ชัดก็คือ การรับส่งโทรทัศน์ และวิทยุกระจายเสียงนั่นเอง สถานีโทรทัศน์จะเป็นตัวส่งและเครื่องรับทำหน้าที่รับเพียงอย่างเดียว จะส่งข่าวสารมายังสถานีส่งไม่ได้



รูปที่ 2.12 แสดงการรับส่งข้อมูลแบบเดียว (Simplex)

2.6.1.1.2 การรับส่งแบบผลัดกันส่ง (Half Duplex)

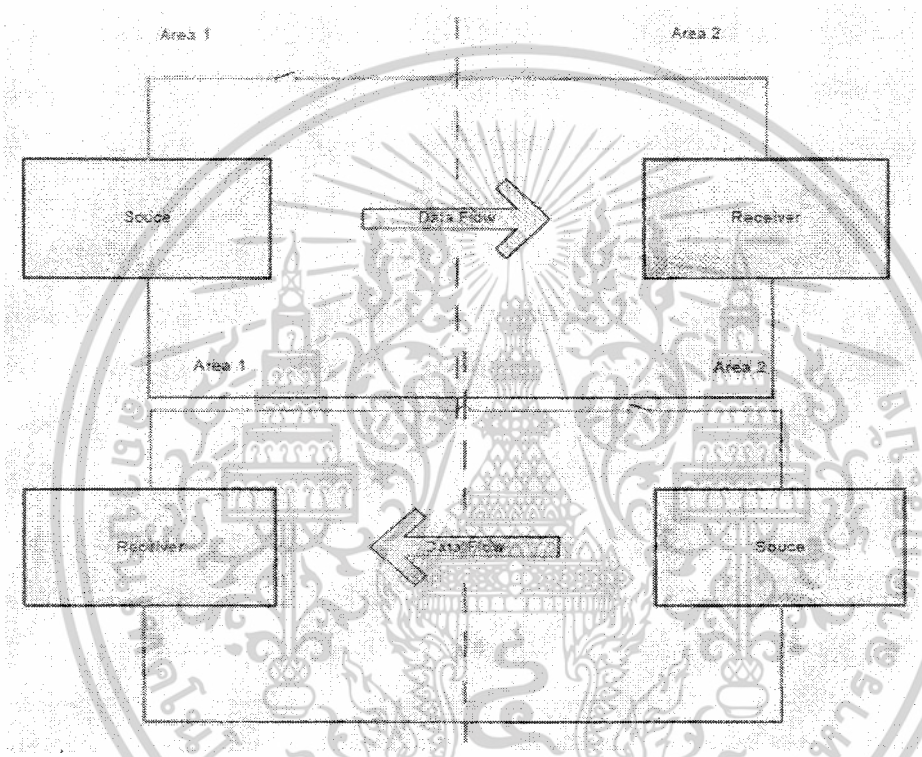
มีคุณสมบัติสามารถรับและส่งข้อมูลได้ แต่ว่าต้องสลับการส่งโดยจะส่งพร้อมกันทั้งสองด้านไม่ได้ อุปกรณ์ที่ใช้ในการติดต่อในแบบนี้ ได้แก่ วิทยุสื่อสาร และอินเตอร์คอม



รูปที่ 2.13 แสดงการรับส่งข้อมูลสวนทางกันได้แบบผลัดการส่ง (Half Duplex)

2.6.1.1.3 การรับส่งสวนทางได้พร้อมกัน (Full Duplex)

การรับส่งแบบนี้ผู้รับและผู้ส่งสามารถรับและส่งพร้อมๆ กันได้ ในเวลาเดียวกันโดยไม่จำเป็นต้องรอให้อีกฝ่ายหนึ่งส่งจบเสียก่อน เช่น การพูดโทรศัพท์ของเรา



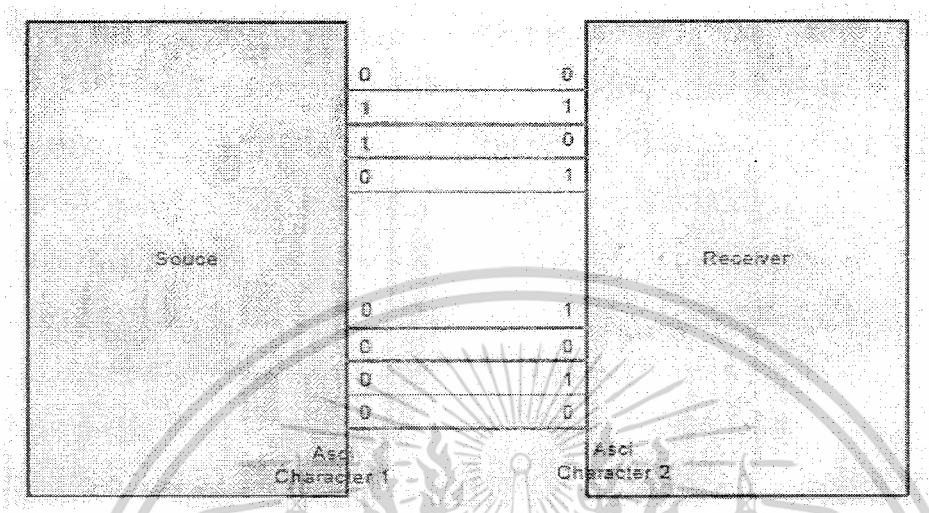
รูปที่ 2.14 แสดงการรับส่งข้อมูลแบบสวนทางกันได้พร้อมกัน (Full Duplex)

2.6.1.2 การจำแนกตามลักษณะการจัดข้อมูล

สามารถแบ่งการส่งข้อมูลออกได้เป็น 2 ชนิดดังนี้

2.6.1.2.1 การส่งข้อมูลแบบขนาน (Parallel Transmission)

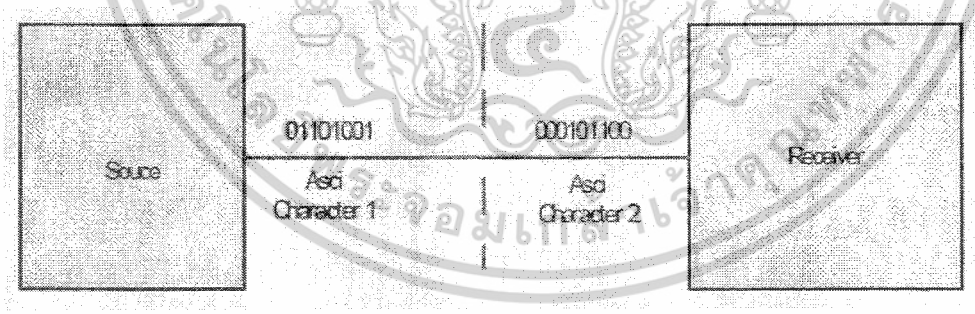
เป็นการส่งข้อมูลทีละไบท์ คือส่งทุกบิตของรหัสที่ประกอบขึ้นเป็นอักขระไปพร้อมๆ กันในเวลาเดียวกัน มีข้อดีคือ ความสามารถในการส่งข้อมูลที่สูงขึ้น แต่จำนวนช่องทางการสื่อสารที่จะต้องมีจำนวนเท่ากับจำนวนบิตที่ประกอบขึ้นเป็นอักขระ ซึ่งต้องใช้การมัลติเพล็กข้อมูลต่างๆ



รูปที่ 2.15 แสดงการส่งข้อมูลแบบขนาน

2.6.1.2.2 การส่งข้อมูลแบบอนุกรม (Series Transmission)

การส่งข้อมูลแบบนี้จะกระทำทีละบิตด้วยแชนแนลเดียว ทางด้านรับจะมีอุปกรณ์สำหรับจัดการจัดข้อมูลดังกล่าวเป็นชุดอักขระ ตามข้อตกลงระหว่างอุปกรณ์ปลายทั้งสองที่สื่อสารกัน



รูปที่ 2.16 แสดงการส่งข้อมูลแบบอนุกรม

2.6.1.3 การจำแนกตามความสัมพันธ์ของข้อมูล

ในการสื่อสารข้อมูลนั้นจะต้องพิจารณาถึงความสัมพันธ์ระหว่างข้อมูล 2 ชนิด คือ

- ความสัมพันธ์ระหว่างบิต (Bit Synchronization)

- ความสัมพันธ์ของอักขระ (Character Synchronization)

ในการสื่อสารข้อมูลเราจำแนกวิธีการส่งข้อมูลตามการกำหนดความสัมพันธ์ของข้อมูลได้ 2 แบบ คือ

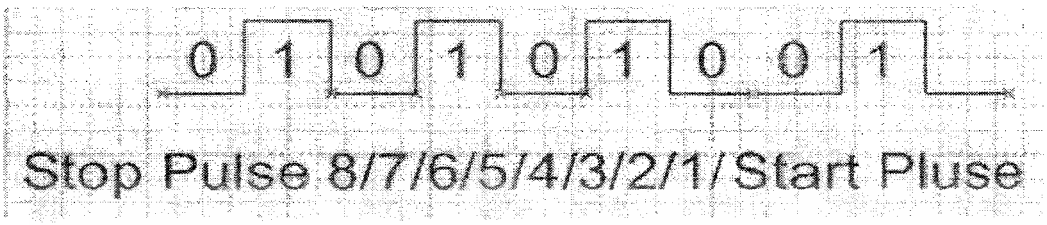
2.6.1.3.1 การส่งแบบสัมพันธ์ (Synchronous Transmission)

การส่งแบบนี้ใช้สำหรับการส่งข้อมูลทั้งชุดไปในครั้งเดียวดังรูป 2.17 เทคนิคการส่งแบบนี้ช่วงเวลาระหว่างบิตต่อจะมีค่าเท่ากัน การส่งมีลักษณะคล้ายกับการส่งข่าวสารในรูปของเลขฐานสองที่มีจำนวนติดต่อกันไป โดยไม่ได้แยกว่าความยาวใดเป็นช่วงอักษรใดในระบบเช่นนี้บิตแต่ละบิตจะมีความยาวเท่ากัน ตัวอักษรแต่ละตัวมีช่วงเวลาห่างกันเท่ากับศูนย์ ทางด้านรับนั้นเพียงหาว่าบิตแรกของตัวอักษรตัวแรกคือบิตใด และทราบขนาดหรือจำนวนบิตในหนึ่งตัวอักษรพร้อมทั้งความเร็วในการส่งก็สามารถแยกข่าวสารของแต่ละอักษรออกมาได้

รูปที่ 2.17 แสดงการส่งข้อมูลแบบสัมพันธ์

2.6.1.3.2 การส่งแบบไม่สัมพันธ์ (Asynchronous Transmission)

การส่งแบบนี้ตัวอักขระจะถูกส่งออกไปที่เวลาใดๆ ก็ได้โดยไม่จำเป็นต้องมีความสัมพันธ์ระหว่างอักษรว่าต้องมีเวลาที่แน่นอนอย่างไร ทางด้านรับจะต้องทราบถึงบิตเริ่มต้นของรหัสแต่ละตัวว่าเริ่มต้นเมื่อใดซึ่งสามารถกระทำได้โดยการเพิ่มบิตที่เรียกว่า พัลส์เริ่มต้น (Start Pulse) โดยเติมเข้าไปที่ข้างหน้าชุดของทุกอักขระ และเมื่อการส่งครบทุกบิตของตัวอักษรแล้วจะต้องมีบิตสำหรับบอกถึงการสิ้นสุดที่เรียกว่า พัลส์การสิ้นสุด (Stop Pulse) ส่งมาให้ทางด้านรับมีเวลาสำหรับการเตรียมข้อมูลของตัวอักษรตัวต่อไปดังรูป 2.20 บางครั้งจึงเรียกระบบการส่งแบบนี้ว่า ระบบการส่งแบบเริ่มหยุด (Star-Stop Transmission)



รูปที่ 2.18 แสดงการส่งข้อมูลแบบไม่สัมพันธ์

2.7 โปรแกรมซี – พลัส – พลัส บิลเดอร์ (C++Builder)

ในโครงงานนี้ได้มีการใช้งานโปรแกรมซี – พลัส – พลัส บิลเดอร์ทำการติดต่อกับผู้ใช้งานให้สามารถควบคุมรูปแบบการเปิดและปิดของหัวของน้ำพุตามที่ผู้ใช้งานต้องการได้สะดวกและรวดเร็ว เพื่อให้ผู้ใช้งานระบบน้ำพุที่มีความรู้ทางด้านอิเล็กทรอนิกส์หรือไม่ก็จะสามารถใช้งานระบบได้โดยศึกษาวิธีการใช้งานเพียงเล็กน้อยเท่านั้นการเขียนโปรแกรมโดยโปรแกรมซี – พลัส – พลัส บิลเดอร์มีรูปแบบของโครงสร้างที่สำคัญต่อไปนี้

2.7.1 การเขียนโปรแกรมแบบวิชวล

วิชวล หมายถึง ภาพหรือสิ่งที่เรามองเห็นการเขียนโปรแกรมแบบวิชวลหรือวิชวลโปรแกรมมิ่งจึงหมายถึงการเขียนโปรแกรมแบบวิชวลมี 4 รายการดังต่อไปนี้

1. ฟอร์ม (form) เป็นหน้าต่างว่างใช้สำหรับออกแบบโปรแกรม การทำงานต่างๆ ของโปรแกรมจะปรากฏอยู่บนฟอร์ม
2. คอมโพเนนต์ (component) เป็นส่วนประกอบต่างๆ ที่จะต้องใช้ในโปรแกรม เช่น เมนู (menu) ปุ่มหรือ บัตทอม (button – ปุ่มสำหรับให้โปรแกรมทำงานอย่างหนึ่ง) เมสเสจบ็อกซ์ (message box – กรอบแสดงข้อความ) ไดอะล็อกบ็อกซ์ (dialog – กรอบสำหรับโต้ตอบกับโปรแกรม)
3. คำสั่งสำหรับจัดลักษณะและการทำงานของคอมโพเนนต์
4. คำสั่งสำหรับควบคุมการทำงานของโปรแกรม

2.7.2 C++และC++ บิลเดอร์

C++ (ซี – พลัส – พลัส) และ C++Builder (ซี – พลัส – พลัส บิลเดอร์) ต่างเป็นภาษาคอมพิวเตอร์ซึ่งใช้สำหรับเขียนโปรแกรมเพื่อส่งให้คอมพิวเตอร์ทำงานต่าง ๆ ตามที่เราต้องการ C++ และ C++ Builder มีคำและกฎเกณฑ์ในการเขียนโปรแกรมเหมือนกันทุกประการแต่วิธีเขียนโปรแกรมต่างกันกล่าวคือ C++ ใช้วิธีเขียนโปรแกรมด้วยอักษรและเครื่องหมายแต่ C++Builder ใช้วิธีเขียนโปรแกรมแบบวิชวลดังนั้นผู้เขียนโปรแกรมด้วย C++ จึงสามารถเปลี่ยนมาเขียนโปรแกรมด้วย C++Builder ได้อย่างรวดเร็วซึ่งวิธีการของ C++ Builder จะทำให้เราสามารถเขียนโปรแกรมได้ง่ายและรวดเร็วกว่าวิธีการของ C++

C++Builder เป็น ANSI C++ (แอนซี ซี – พลัส – พลัส) ซึ่งหมายความว่า C++Builder เป็นภาษา C++ ที่เป็นไปตามมาตรฐาน ANSI (American National Standards Institue – สถาบันมาตรฐานสหรัฐอเมริกาทำหน้าที่กำหนดมาตรฐานต่าง ๆ รวมทั้งภาษา C++ ด้วย) ดังนั้นผู้ที่เคยเขียนโปรแกรมด้วย C++Builder ได้ทันที

C++Builder ผลิตโดยบริษัทอินไพรซ์ (Inprise Corporation) ซึ่งเปลี่ยนชื่อมาจากบริษัทบอร์แลนด์อินเตอร์เนชันนัล (Borland International,Inc.) ประเทศสหรัฐอเมริกา C++Builder มีชื่อเต็มว่า บอร์แลนด์ C++Builder (Borland C++Builder) และนิยมเรียกว่า C++Builder C++Builder เวอร์ชั่นแรกมีจำหน่ายในตอนต้นปี 1997 สำหรับ C++Builder เวอร์ชั่น 5 มีจำหน่ายในตอนต้นปี 2000 โดยแบ่งการจำหน่ายเป็น 3 ชุดคือ 1) Borlan C++ Builder 5 Standard 2) Borland C++Builder 5 Professional 3) Borland C++ Builder 5 Enterprise ซึ่งแต่ละชุดจะมีส่วนประกอบเท่ากันคือชุด Standard จะมีส่วนประกอบน้อยที่สุดและชุด Enterprise จะมีส่วนประกอบมากที่สุด

วิธีเขียนโปรแกรมด้วย C++Builder ทำได้โดยการนำคอมโพเนนต์ที่เรามองเห็นมาว่าในฟอร์แล้วกำหนดลักษณะการทำงานให้กับคอมโพเนนต์นั้น ในบางคอมโพเนนต์ในบางโปรแกรมอาจจะต้องเขียนคำสั่งควบคุมการทำงานของโปรแกรมเพิ่มเติมต่อจากนั้นจึงคอมไพล์ (compile) และรัน (run) โปรแกรมนั้น ผลจากการคอมไพล์โปรแกรมจะได้ไฟล์ชนิด .EXE ซึ่งสามารถนำไปรันในคอนพิว-เตอร์เครื่องอื่นได้อย่างอิสระโดยไม่ต้องเกี่ยวข้องกับ C++Builder อีกเลย

C++Builder เป็นภาษา C++ ประเภทที่สามารถสร้างโปรแกรมใช้งานได้อย่างรวดเร็ว ภาษาคอมพิวเตอร์ประเภทนี้มีชื่อย่อว่า (RAD – Rapid Application Development) ซึ่งหมายความว่าด้วย C++ Builder เราสามารถเขียนโปรแกรมภาษา C++ เพื่อรันในวินโดวส์ได้เร็วกว่าการเขียนโปรแกรมแบบธรรมดา

2.7.2.1 การติดต่อระหว่างวินโดวส์กับแอปพลิเคชั่น

มีคำ 2 คำที่เกี่ยวข้องกับการอธิบายเรื่องการติดต่อระหว่างวินโดวส์กับแอปพลิเคชั่นคือ อีเวนต์ (event) และ เมสเสจ (message)

อีเวนต์

อีเวนต์หมายถึงทุกสิ่งทุกอย่างที่เกิดขึ้นบนระบบคอมพิวเตอร์ เช่น การเลื่อนเมาส์ การคลิกการกดคีย์ การเลื่อนวินโดว์ การเปลี่ยนขนาดของวินโดว์ เมื่อเกิดอีเวนต์ขึ้นวินโดวส์จะเก็บอีเวนต์ทั้งหมดไว้ในคิว(queue) ที่มีชื่อเรียกว่า โกลบอลอีเวนต์คิว(global event queue – คิวของทั้งหมด ซึ่งบางทีเรียกว่า system – wide queue หรือ hardware event queue) โกลบอลอีเวนต์คิวใช้สำหรับเก็บอีเวนต์ของทุกแอปพลิเคชั่นที่รันในวินโดวส์

เมสเสจ

เมสเสจหมายถึงชุดของข่าวสารที่เป็นรายละเอียดของแต่ละอีเวนต์ วินโดวส์จะสร้างเมสเสจของอีเวนต์โดยการส่งอีเวนต์ที่เกิดขึ้นให้แก่ไดรเวอร์ (driver – โปรแกรมที่คอมพิวเตอร์ใช้ติดต่อกับอุปกรณ์ชนิดนั้น) ที่สอดคล้องกับอีเวนต์นั้น เช่น เมื่อเราคลิกจะเกิดอีเวนต์คลิก วินโดว์จะส่งอีเวนต์นี้ให้แก่ไดรเวอร์ของเมาส์และไดรเวอร์ของเมาส์จะสร้างเมสเสจคลิกให้แก่วินโดวส์โดยไดรเวอร์จึงส่งเมสเสจนี้ไว้ในซิสเต็มคิว (system queue – คิวของระบบ) ต่อจากนั้นวินโดวส์จึงส่งเมสเสจจากซิสเต็มคิวไปที่แอปพลิเคชั่นคิว (application queue – คิวเก็บเมสเสจของแอปพลิเคชั่น) ของแอปพลิเคชั่นต่างๆ ที่กำลังทำงานอยู่ในขณะนั้นซึ่งเป็นผลให้แอปพลิเคชั่นได้รับเมสเสจจากวินโดวส์เนื่องจากวินโดวส์สามารถรันแอปพลิเคชั่นคิวสำหรับแอปพลิเคชั่นนั้นโดยเฉพาะความสัมพันธ์ระหว่างอีเวนต์กับเมสเสจ

หลักการทำงานของแอปพลิเคชั่นที่รันบนวินโดวส์ก็คือ การดำเนินการกับเมสเสจการรับส่งเมสเสจวินโดวส์หรือกับแอปพลิเคชั่นอื่นเมสเสจทั้งหมดของวินโดวส์กำหนดไว้เป็นคอนสแตนต์ (constant – ค่าคงที่ไม่เปลี่ยนแปลง) เช่น จากเมนูที่ปรากฏบนจอภาพเมื่อเราเลือกรายการหนึ่งจะมีเมสเสจ WM_COMMAND เกิดขึ้นในบรรดาเมสเสจเหล่านี้มีหลายเมสเสจที่เกิดขึ้นพร้อมกับแวลิเอเบิล (variable – ค่าที่เปลี่ยนแปลงได้) จำนวนหนึ่งซึ่งนำมาใช้ในแอปพลิเคชั่นได้ เช่น เมื่อเรากดปุ่มซ้ายของเมาส์ วินโดวส์จะสร้างเมสเสจ WM_LBUTTONDOWN ขึ้นมาเมสเสจนี้ประกอบด้วยแวริเอเบิลซึ่งเป็นโคออร์ดิเนต (coordinate – ตำแหน่ง) ของเมาส์พอยน์เตอร์เราสามารถนำค่าดังกล่าวนี้มาใช้ได้ เช่น ใช้สำหรับแสดงผลบนจอภาพใช้ในการตรวจสอบตำแหน่งของเมาส์พอยน์เตอร์ เมสเสจในวินโดวส์มีประมาณ 200 เมสเสจแต่เมสเสจที่ใช้บ่อยมีประมาณ 20 เมสเสเท่านั้น

2.7.3 โปรแกรมที่ทำงานด้วยอีเวนต์

แอปพลิเคชั่นที่รันในวินโดวส์จะใช้เวลาส่วนใหญ่เพื่อการรับและส่งเมสเสจชนิดต่างๆ จึงนิยมเรียกแอปพลิเคชั่น (หรือโปรแกรม) ที่ทำงานในลักษณะนี้ว่าโปรแกรมที่ทำงานด้วยอีเวนต์ (event – drive program) ซึ่งเป็นโปรแกรมที่มีหลักการทำงานดังต่อไปนี้คือในโปรแกรมที่ทำงานด้วยอีเวนต์ประกอบด้วยฟังก์ชัน (function – สเตตเมนต์ชุดหนึ่งซึ่งทำงานอย่างหนึ่งและมีชื่อเฉพาะสำหรับให้ส่วนอื่นของโปรแกรมเรียกใช้งานได้) จำนวนหนึ่งซึ่งเรียกว่าฟังก์ชันตอบสนองอีเวนต์หรือเมตธอด (method) ซึ่งแต่ละฟังก์ชันจะมีหน้าที่ตอบสนองกับแต่ละอีเวนต์โดยเฉพาะและเป็นอิสระต่อกันทั้งนี้เนื่องจากในวินโดวส์จะมีอีเวนต์เกิดขึ้นเมื่อใดก็ได้เราจึงไม่ทราบว่าจะเรียกใช้ฟังก์ชันอะไรเมื่อใด

สมมุติว่าเราจะเขียนโปรแกรมให้ทำงานพร้อมกัน 2 งานคือ 1) แสดงเวลาปัจจุบันซึ่งตัวเลขเปลี่ยนแปลงเวลาทุกวินาทีที่มุมล่างขวาของจอภาพ 2) แสดงเมนูซึ่งมีหลายรายการเมนูให้เลือกที่กลางจอภาพขณะที่โปรแกรมทำงานในบริเวณแสดงเวลาเราจะเห็นเวลาเปลี่ยนแปลงไปเรื่อยๆ และในบริเวณที่แสดงเมนูจะเห็นรายการเมนูซึ่งสามารถเลือกรายการหนึ่งได้ทันที โปรแกรมจะมีส่วนประกอบและหลักการทำงานดังนี้ 1) ในส่วนแสดงเวลาจะมีสเตตเมนต์เกี่ยวกับการแสดงเวลา 2) ในส่วนเมนูจะมีสเตตเมนต์แสดงเมนูและมีฟังก์ชันตอบสนองอีเวนต์การเลือกรายการเมนูแต่ละรายการโดยแต่ละส่วนของโปรแกรมและแต่ละฟังก์ชันที่มีอยู่ต่างทำงานแยกเป็นอิสระไม่เกี่ยวข้องกันในการทำงานเมื่อแอปพลิเคชั่นได้รับเมสเสจมาจากวินโดวส์แอปพลิเคชั่นจะดำเนินการเลือกฟังก์ชันที่สอดคล้องกันมาทำงาน เช่น ถ้าการคลิกเลือกรายการหนึ่งจากเมนูฟังก์ชั่นตอบสนองอีเวนต์การเลือกเมนูรายการนั้นจะทำงาน ด้วยกลไกการทำงานดังกล่าวนี้เราจึงสามารถสร้างแอปพลิเคชั่นที่มีความซับซ้อนได้ง่ายขึ้นเพราะเราเพียงแต่สร้างฟังก์ชันตอบสนองอีเวนต์ซึ่งได้รับมาจากวินโดวส์เท่านั้นโดยไม่จำเป็นต้องกังวลใจว่าเมสเสจนั้นจะเกิดขึ้นเมื่อไร เราสนใจแต่เพียงว่าเมื่อเมสเสจนั้นเกิดขึ้นโปรแกรมจะต้องทำอะไร

2.7.4 หลักการเขียนโปรแกรม

C++ บิลเดอร์เป็นโปรแกรมภาษาคอมพิวเตอร์ชนิดคอมไพเลอร์ (Compiler- โปรแกรมแปลภาษา) ทำหน้าที่ คอมไพล์ (Object code- รหัสที่สั่งให้คอมพิวเตอร์ทำงาน) โดยมีการลิงค์ (link – รวมส่วนประกอบต่างๆ ของโปรแกรมเข้าด้วยกัน) แบบอัตโนมัติ
วิธีเขียนโปรแกรมมีลำดับขั้นดังนี้

1. สร้างซอร์สโค้ดเป็นภาษาอังกฤษที่เขียนขึ้นตามกฎเกณฑ์ของภาษา C++ ผู้ที่รู้ภาษา C++ จะเข้าใจความหมายของซอร์สโค้ดว่าเป็นการสั่งให้คอมพิวเตอร์ทำอะไรทำอย่างไรและทำ

เมื่อไรการสร้างซอร์สโค้ดใน C++ บิลเดอร์สามารถทำได้ง่ายและรวดเร็วเพราะ C++ บิลเดอร์จะสร้างซอร์สโค้ดส่วนหนึ่งอัตโนมัติ ซึ่งส่วนหนึ่งที่ทำให้โปรแกรมสามารถทำงานได้ในระดับหนึ่ง ส่วนของซอร์สโค้ดที่จะทำให้โปรแกรมทำงานได้สมบรูณ์เราอาจป้อนเติมเข้าทางคีย์บอร์ดหรือโดยการอ่านจากดิสก์ก็ได้

2. คอมไพล์ เมื่อมีซอร์สโค้ดอยู่ในคอมพิวเตอร์เรียบร้อยแล้ว ขั้นต่อไปก็คือการคอมไพล์ซึ่งเป็นการตรวจสอบว่าโปรแกรมเขียนถูกต้องตามกฎหรือไม่ถ้าผิดคอมไพล์เลอร์จะแสดงข้อความสาเหตุที่ผิดเราจะต้องแก้ไขและคอมไพล์ใหม่จนถูกต้อง เมื่อไม่พบความผิดพลาดแล้วก็ถือว่าโปรแกรมนั้นผ่านการคอมไพล์
3. ลิงค์ สำหรับโปรแกรมที่ผ่านการคอมไพล์แล้ว C++ บิลเดอร์จะนำส่วนประกอบอื่นๆ เช่น โปรแกรมย่อย ข้อมูล ซึ่งโปรแกรมของเราจะต้องใช้เข้ามารวมเพื่อให้โปรแกรมสามารถทำงานได้ตามต้องการ ขั้นตอนนี้เรียกว่าการลิงค์ ซึ่ง C++ บิลเดอร์จะดำเนินการให้แบบอัตโนมัติทันทีที่คอมไพล์ผ่าน เมื่อผ่านการลิงค์แล้วเราจะได้ออปเจ็กต์โค้ดอยู่ในไฟล์ชนิด .EXE
4. รันเมื่อเป็นออปเจ็กต์โค้ดแล้วโปรแกรมนั้นก็สามารถทำงานได้ การสั่งให้โปรแกรมทำงานเรียกว่าการ รัน (run) ขณะรันโปรแกรมอาจจะมีความผิดพลาดเกิดขึ้นได้อีก เราเรียกความผิดพลาดกลุ่มนี้ว่ารันไทม์เออเรอร์ (runtime eror) ซึ่งโปรแกรมจะหยุดทำงานทันทีและแสดงสาเหตุของความผิดพลาดนั้น

2.7.5 C++ บิลเดอร์ IDE

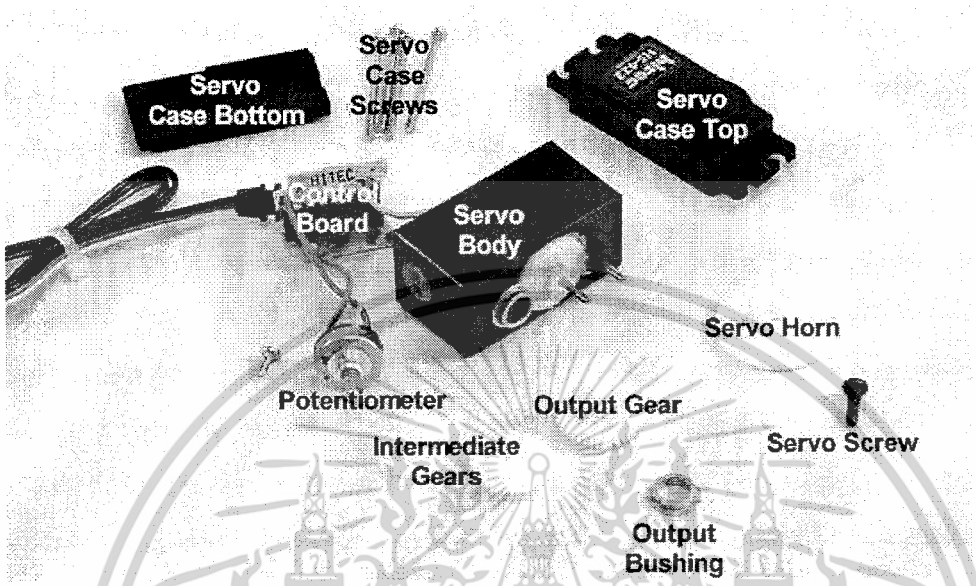
C++ บิลเดอร์ได้รวมส่วนประกอบต่างๆ ที่เกี่ยวข้องกับการเขียนโปรแกรมไว้ด้วยกันเพื่อเป็นการอำนวยความสะดวกในการเขียนโปรแกรมให้แก่ผู้เขียนโปรแกรม ส่วนประกอบเหล่านี้มีชื่อย่อว่า IDE (Integrated Development Environment – ไอดีอี) ความสามารถของ IDE มีตัวอย่างดังนี้

1. มีส่วนประกอบต่างๆ คือ ฟอร์ม คอมโพเนนต์คำสั่งดำเนินการกับคอมโพเนนต์เพื่อสำหรับใช้ออกแบบโปรแกรมแบบวิชวล
2. มีโค้ดเอดิเตอร์ (code editor) ซึ่งช่วยอำนวยความสะดวกในการป้อนซอร์สโค้ดและปรับปรุงซอร์สโค้ด

3. ถ้าพบความผิดพลาดในการคอมไพล์ IDE จะแสดงสาเหตุความผิดพลาดนั้นและเลื่อนเคอร์เซอร์ไปอยู่ ณ บริเวณที่มีความผิดพลาดซึ่งช่วยให้เราทราบสาเหตุของความผิดพลาดและสามารถหาตำแหน่งที่ผิดพลาดได้เร็วขึ้น
4. อำนวยความสะดวกในการรันโปรแกรมโดยไม่ต้องออกจาก IDE ทำให้เราสามารถทดสอบโปรแกรมได้อย่างสะดวกและรวดเร็ว
5. มีระบบ Help ซึ่งสามารถแสดงคำอธิบายต่างๆเกี่ยวกับ C++ บิลเดอร์และครบถ้วน

2.8 เซอร์โวมอเตอร์(Servo motor)

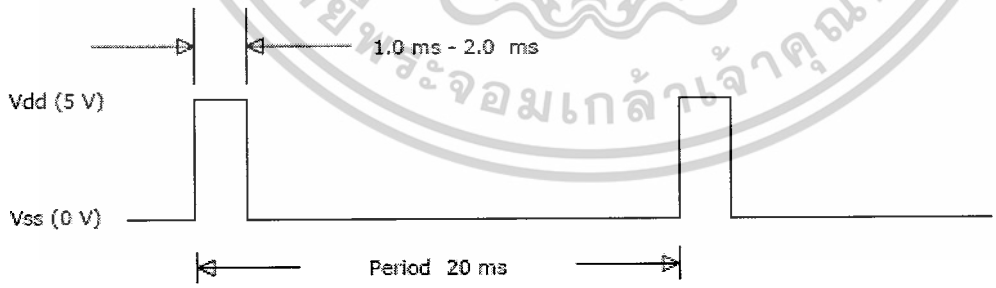
Servo motor คือ มอเตอร์ไฟฟ้ากระแสตรง (DC motor) ที่ถูกประกอบรวมกับ ชุดเกียร์ และส่วนควบคุมต่างๆไว้ในโมดูลเดียวกัน หรือภายในกล่องพาสติกเดียวกัน โดยมอเตอร์ชนิดนี้จะมีสายต่อใช้งานเพียง 3 เส้นเท่านั้น คือ VCC,GND และสายสัญญาณควบคุม(Control Line) ซึ่งสามารถควบคุมให้มอเตอร์หมุนซ้าย หรือ ขวาได้จากสายสัญญาณเพียงเส้นเดียว โดยสัญญาณที่ใช้ควบคุมนี้จะเป็นสัญญาณพัลส์วิดมอด (PWM) แบบ TTL Level ระดับแรงดันที่จ่ายให้มอเตอร์นี้จะอยู่ในช่วงประมาณ 4 ถึง 6 โวลท์ ขึ้นอยู่กับคุณสมบัติของมอเตอร์แต่ละตัว ข้อดีของมอเตอร์ชนิดนี้ก็คือ จะมีขนาดเล็กน้ำหนักเบาให้แรงบิตสูงกินพลังงานน้อย และ สามารถควบคุม ด้วยแรงดันลอจิกที่เป็น TTL ได้โดยตรงไม่จำเป็นต้องต่อวงจรขับ(Driver) อื่นๆ เพราะมอเตอร์ชนิดนี้จะมีวงจรควบคุมบรรจุไว้ภายในอยู่แล้ว ซึ่งมอเตอร์ชนิดนี้สามารถควบคุมให้หมุนไปในตำแหน่ง หรือทิศทางองศาที่ต้องการได้ โดยอาศัยสัญญาณความกว้างพัลส์ที่ป้อนให้มอเตอร์ แต่เซอร์โวมอเตอร์นี้จะหมุนได้แค่เพียงในช่วงประมาณ 180° หรือ ครึ่งรอบเท่านั้น หรือบางรุ่นอาจหมุนได้ถึง 210° แต่จะไม่สามารถหมุนเป็นวงรอบได้ เนื่องจากโครงสร้างภายในจะประกอบด้วย ตัวต้านทานชนิดปรับค่าได้ (VR) ที่ทำหน้าที่ตรวจสอบตำแหน่งการหมุนของมอเตอร์ และ ตัวต้านทานนี้จะถูกยึดติดกับแกนหมุนของมอเตอร์ ซึ่งจากการที่ตัวต้านทานปรับค่านี้ไม่สามารถหมุนเป็นวงรอบได้ ดังนั้นเซอร์โวมอเตอร์จึงถูกออกแบบให้หมุนได้เพียงแค่ประมาณ 180 องศา หรือครึ่งรอบเท่านั้น เพื่อป้องกันความเสียหายที่จะเกิดกับตัวต้านทานปรับค่าได้ แต่ถ้าหากเราต้องการให้มอเตอร์หมุนเป็นวงรอบ (360°) นั้นก็สามารถทำได้ โดยจะต้องทำการปรับแต่ง (Modify) ดัดแปลงชิ้นส่วนบางอย่างของมอเตอร์

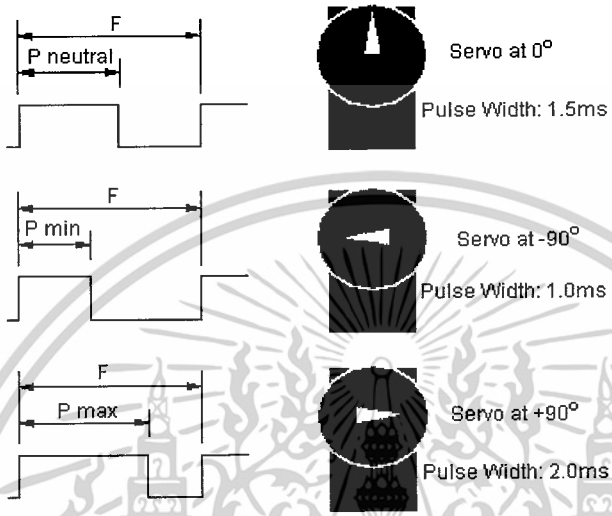


รูปที่ 2.19 การแยกชิ้นส่วนของเซอร์โว

2.8.1 หลักการทำงานของ Servo motor

การควบคุมการทำงานของ เซอร์โวมอเตอร์ ทำได้โดยการป้อนสัญญาณความกว้างพัลส์ให้กับมอเตอร์ ซึ่งตำแหน่งและทิศทางการหมุนของมอเตอร์นี้จะขึ้นอยู่กับขนาดของความกว้างของพัลส์นั้นๆ โดยทั่วไปแล้วความกว้างของสัญญาณพัลส์จะมีจุดให้อ้างอิง 3 จุด ดังรูป คือ





รูปที่ 2.20 การทำงานของเซอร์โวมอเตอร์

สัญญาณความกว้างพัลส์ขนาด 1.5 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม 0 องศา หรือ จุดกึ่งกลางของมอเตอร์

สัญญาณความกว้างพัลส์ขนาด 1 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม - 90 องศา หรือในทิศทางทวนเข็มนาฬิกา

สัญญาณความกว้างพัลส์ขนาด 2 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม + 90 องศา หรือในทิศทางตามเข็มนาฬิกา

ส่วนการที่จะควบคุมให้มอเตอร์หมุนเป็นมุมอื่นๆนั้นก็สามารถทำได้โดยการป้อนสัญญาณพัลส์เป็นระดับความกว้างต่างๆ โดยอ้างอิงจากจุดทั้ง 3 จุดที่กล่าวมานี้ ตัวอย่างเช่น ถ้าต้องการให้มอเตอร์หมุนไปที่มุม - 45 องศา เราก็จะต้องป้อนสัญญาณพัลส์ที่มีความกว้าง 1.25 ms เป็นต้น และสัญญาณพัลส์นี้จะต้องจ่ายให้มอเตอร์ทุกๆ 20 ms (Period) เพื่อรักษาสภาพตำแหน่งของมอเตอร์ไว้โดยหลักการก็คือ จะอาศัยการเปรียบเทียบช่วงเวลาของความกว้างพัลส์ที่จ่ายให้กับมอเตอร์ทางขาสัญญาณควบคุมกับค่าเวลาของวงจร RC ภายในบอร์ดควบคุมในตัวของมอเตอร์ ซึ่งค่าเวลาของวงจร RC นี้จะมีการเปลี่ยนแปลงตามการหมุนของมอเตอร์ เนื่องจากตัวต้านทานปรับค่าจะถูกยึดติดอยู่กับแกนหมุนของมอเตอร์ ซึ่งการหมุนของมอเตอร์จะทำให้ค่าความต้านทานของตัวต้านทานปรับค่า (VR) เปลี่ยนแปลง

ไป เป็นผลทำให้ค่าเวลาของวงจร RC เปลี่ยนแปลงตามไปด้วย โดยในขณะที่เราป้อนสัญญาณความกว้างพัลส์ให้กับมอเตอร์ทางขาสัญญาณควบคุม สัญญาณนี้จะถูกนำไปเปรียบเทียบกับค่าเวลาของวงจร RC หากค่าทั้ง 2 ไม่เท่ากันมอเตอร์ก็จะหมุนทำให้ค่าเวลาของวงจร RC เปลี่ยนแปลงจนกระทั่งค่าเวลาความกว้างพัลส์ของวงจร RC เปลี่ยนแปลงจนเท่ากับสัญญาณพัลส์ทางขาควบคุม (Control line) มอเตอร์จึงจะหยุดหมุน

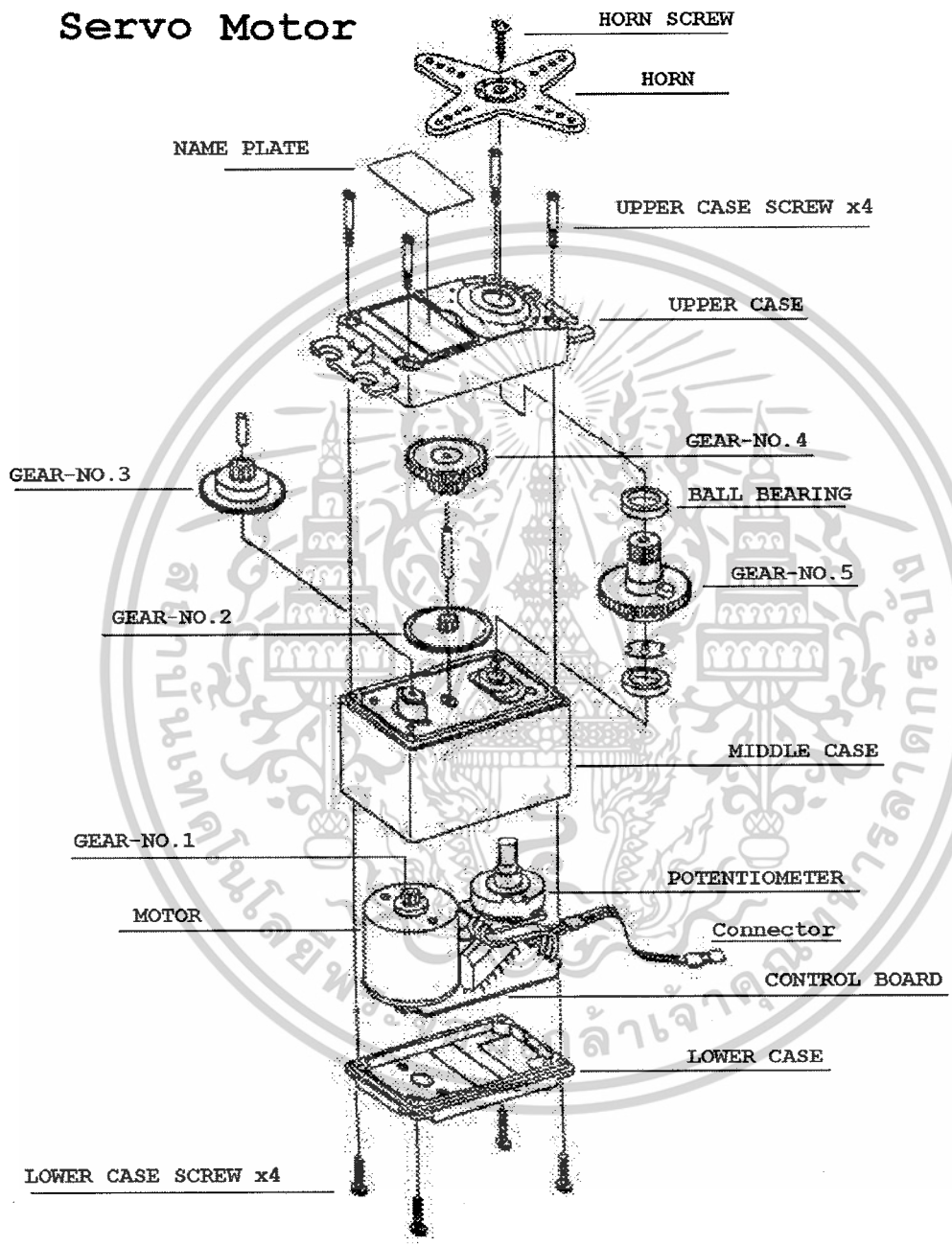
2.8.2 การปรับแต่งเซอร์โวมอเตอร์

จากคุณสมบัติของ Servo motor ที่ผลิตออกมาจากโรงงานจะสามารถหมุนได้แค่เพียงประมาณ 180 องศา หรือประมาณครึ่งรอบเท่านั้น หากเราต้องการนำเอา Servo motor ไปใช้งานในลักษณะที่หมุนเป็นวงรอบนั้นก็สามารถทำได้ แต่ก็จะสูญเสียการควบคุมในเรื่องของการสั่งให้มอเตอร์หมุนไปในตำแหน่ง หรือมุมที่ต้องการไปด้วย จะทำได้ก็เพียงในเรื่องของการสั่งให้หมุนซ้าย ขวา และหยุด เท่านั้น โดยการทำให้มอเตอร์สามารถหมุนเป็นวงรอบได้นั้นจะต้องทำการปรับแต่ง หรือแก้ไขโครงสร้างภายในบางส่วนของมอเตอร์ ซึ่งได้แก่

- การต่อตัวต้านทานคงที่ 2 ตัวอนุกรม แทนตัวต้านทานปรับค่าได้
- ตัดชิ้นส่วนของแกนเฟืองที่ทำหน้าที่หยุดมอเตอร์ (TAB STOP) ออก
- การดัดแปลงตัวต้านทานปรับค่าได้(VR) ให้สามารถหมุนได้รอบทิศทาง (360°)

มีขั้นตอนดังต่อไปนี้

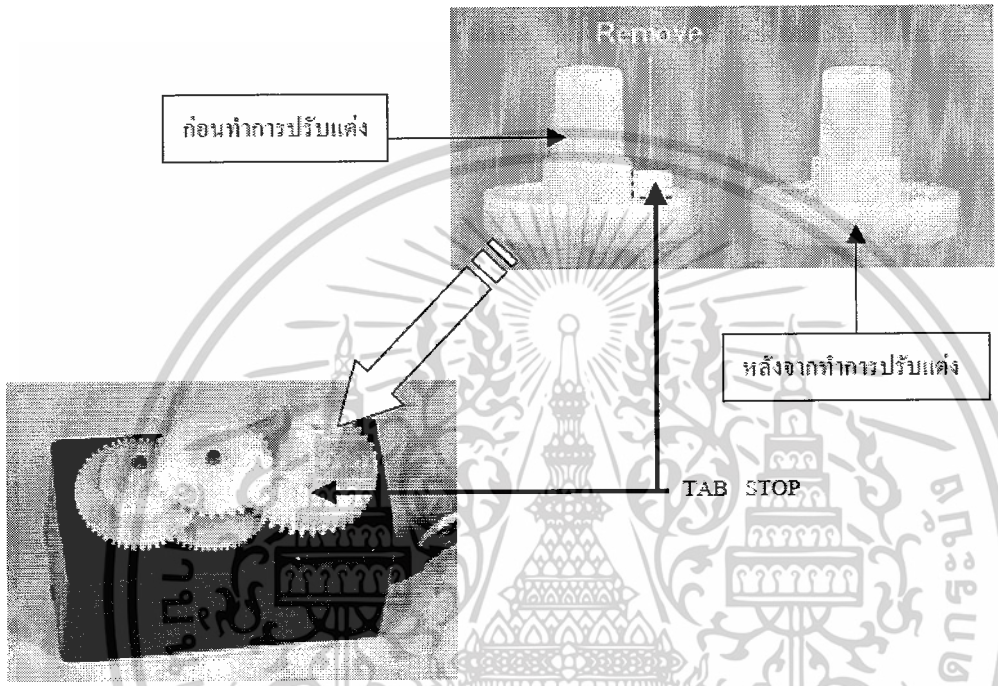
1. ถอดชิ้นส่วนของ Servo motor ออกเป็นส่วนๆ



รูปที่ 2.21 ส่วนประกอบของเซอร์โว

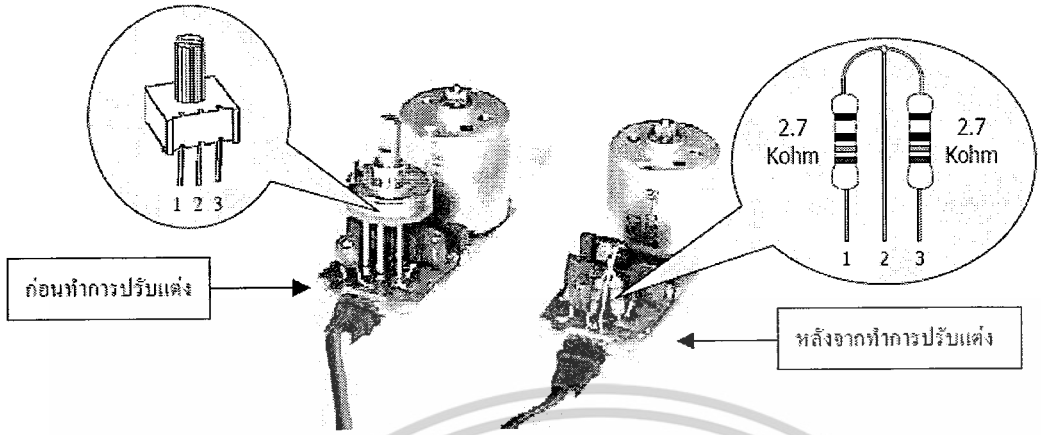
2. ตัดแกนที่ติดกับเฟือง (TAB STOP) ออก โดยแกนนี้มีหน้าที่ป้องกันไม่ให้มอเตอร์หมุนเกินมุม 180 องศา ทั้งนี้เพื่อป้องกันความเสียหายที่จะเกิดขึ้นกับตัวต้านทานปรับค่าได้ เนื่องจากตัวต้านทานชนิด

ปรับค่าได้ไม่สามารถหมุนเป็นวงรอบได้ ดังนั้นเพื่อให้มอเตอร์หมุนเป็นวงรอบได้จึงต้องตัด TAB STOP ในส่วนนี้ออกดังรูป



รูปที่ 2.22 แกนที่ติดกับเฟือง (TAB STOP)

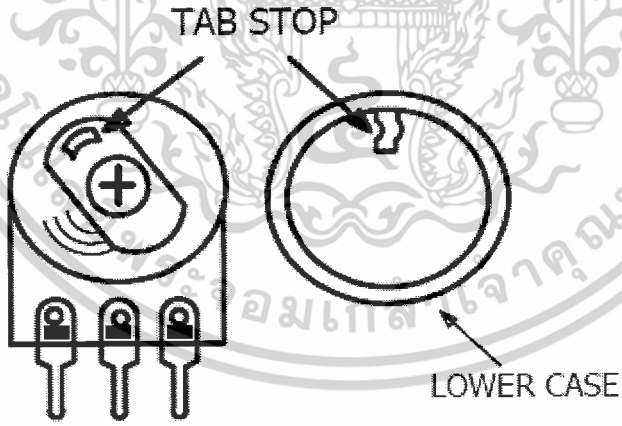
3. ถอดตัวต้านทานปรับค่าได้ (VR) ออก แล้วใส่ตัวต้านทานชนิดค่าคงที่ 2 ตัวที่ต่ออนุกรมกันเข้าไปแทนในตำแหน่งของตัวต้านทานปรับค่าได้ โดยตัวต้านทานชนิดค่าคงที่ที่นำมาต่อนี้จะต้องมีค่าอยู่ในช่วง 2.2 k ถึง 3.3 k ทั้งนี้เนื่องจากตัวต้านทานชนิดปรับค่าได้ที่อยู่ในบอร์ดควบคุมของ Servo motor นั้นจะมีค่าความต้านทาน 5 k ดังนั้น จึงต้องนำตัวต้านทานค่าคงที่มาต่ออนุกรมกันเพื่อให้ได้ค่าความต้านทานใกล้เคียงกับของเดิม ดังรูปต่อไปนี้



รูปที่ 2.23 ตัวต้านทานปรับค่าได้ (VR)

4. ถึงแม้ว่าเราจะถอดตัวต้านทานปรับค่า(VR) ออกจากวงจรแล้วก็ตาม แต่เนื่องจากเรายังคงต้องใช้ตัวต้านทานปรับค่าได้นี้ไปเป็นแกนหมุนของมอเตอร์อยู่ ซึ่งตัวตานทานปรับค่านี้ จะไม่สามารถหมุนเป็นวงรอบได้ ทำให้เราต้องแก้ไขเปลี่ยนแปลงบางส่วนของตัวต้านทานเพื่อให้ตัวต้านทานสามารถหมุนรอบตัวเองได้ เพื่อที่จะได้ไม่ไปขัดขวางการหมุนของมอเตอร์ซึ่งทำได้โดย

- ถอดชิ้นส่วนของตัวต้านทานปรับค่าออก



รูปที่ 2.24 ภายในตัวต้านทานปรับค่า (VR)

- ตัวต้านทานปรับค่าในมอเตอร์แต่ละรุ่นนั้น อาจจะใช้ไม่เหมือนกัน แต่จะมีหลักการเดียวกันโดยจะมีแท็ป ที่ทำหน้าที่หยุดการหมุนของตัวต้านทานอยู่ ให้เราทำการตัดส่วนนี้ออกแล้วทดลองหมุนแกนของตัวต้านทานปรับค่า ถ้าสามารถหมุนรอบตัวเองได้ ก็ทำการประกอบตัวต้านทานเข้าไว้เหมือนเดิม แต่ถ้า

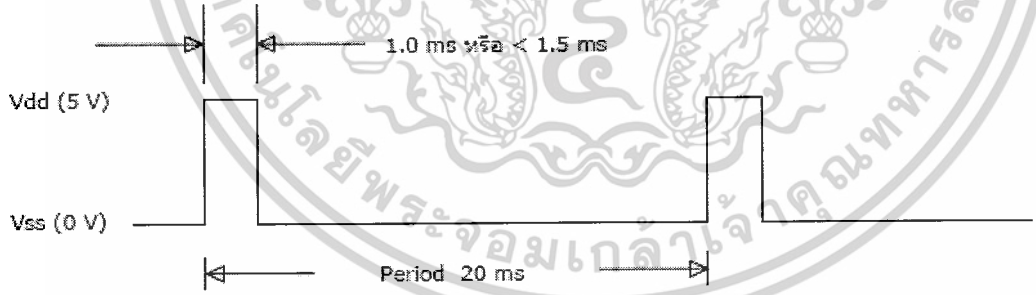
ยังหมุนเป็นวงรอบไม่ได้ก็ให้พิจารณาดูว่ามีชิ้นส่วนใด ที่ยังขัดขวางการหมุนของตัวต้านทานอยู่ เมื่อพบก็ให้เอาออก หรือ ทำลายได้เลยโดยไม่ต้องสนใจว่าจะทำให้ตัวต้านทานนี้พัง เพราะเราไม่ได้ใช้ประโยชน์จากการเปลี่ยนแปลงค่าความต้านทานนี้อีกแล้ว นอกจากใช้เป็นแกนหมุนของเฟืองเท่านั้น

- จากนั้นตัดหรือพับขาของตัวต้านทานปรับค่า (VR) เพื่อป้องกันไม่ให้ขาของตัวต้านทานดังกล่าวไปช็อตกับแผงวงจรควบคุม

5. ประกอบชิ้นส่วนต่างๆ เข้าที่เดิม และ เพื่อความปลอดภัยในการประกอบตัวต้านทานปรับค่า (VR) ลงในกล่องของ Servo motor ควรหาฉนวนรองตรงส่วนของขาที่เป็นโลหะของตัวต้านทานด้วยเพื่อไม่ให้ไปช็อตกับส่วนอื่นๆ ในแผงวงจรควบคุม เพียงเท่านี้มอเตอร์ของเราก็จะสามารถหมุนเป็นวงรอบ 360 องศาได้แล้ว และ ในการนำไปใช้งานจะต้องระวังเรื่องของโหลดที่นำมาต่อกับมอเตอร์ เพราะหากนำมอเตอร์ไปขับ หรือ ยกโหลดที่มีน้ำหนักมากเกินไป อาจจะทำให้เกิดความเสียหายกับ เฟือง หรือ เกียร์ต่างๆ ของมอเตอร์ได้

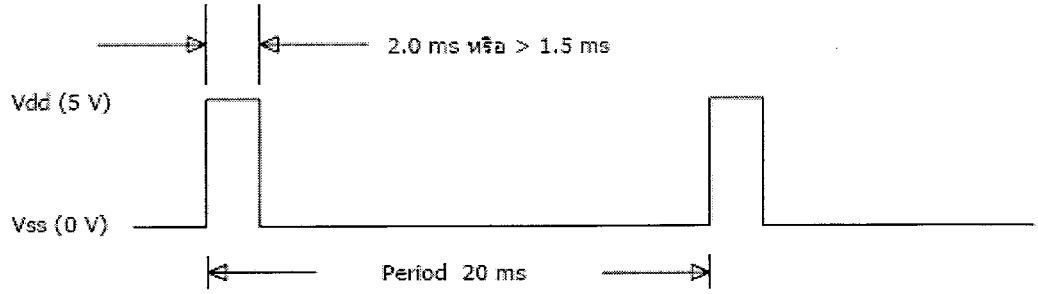
หลังจากเราได้ทำการปรับแต่งการทำงานของเซอร์โวมอเตอร์ให้สามารถหมุนเป็นวงรอบได้แล้ว วิธีในการควบคุมให้มอเตอร์หมุน จะมีลักษณะดังนี้

- การควบคุมให้มอเตอร์หมุนทางด้านซ้ายจะต้องป้อนสัญญาณพัลส์ที่มีขนาดความกว้างพัลส์ 1 ms หรือให้น้อยกว่า 1.5 ms โดยจะต้องป้อนสัญญาณพัลส์นี้ทุกๆ 20 ms (หรือในช่วงประมาณ 20ms – 30ms)



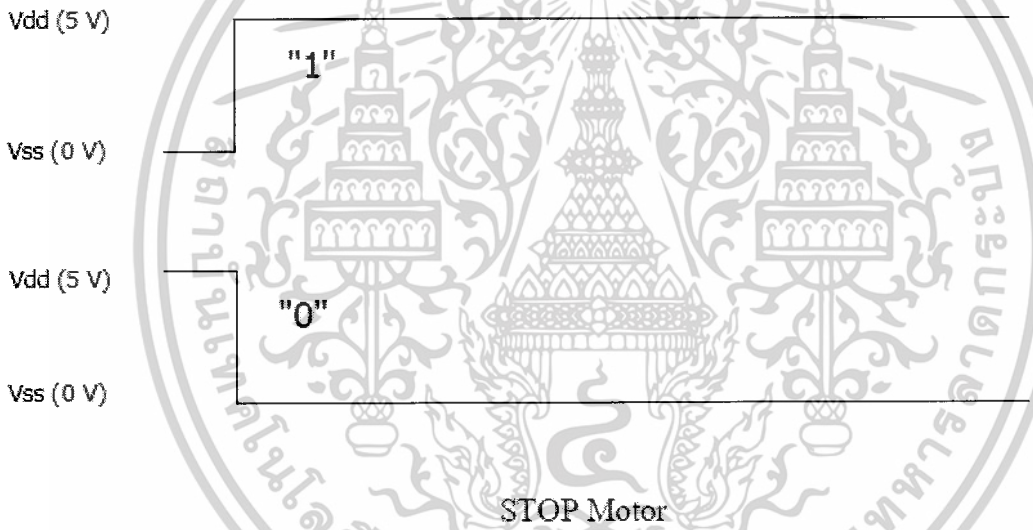
รูปที่ 2.25 การควบคุมให้มอเตอร์หมุนทางด้านซ้าย

- การควบคุมให้มอเตอร์หมุนทางด้านขวาจะต้องป้อนสัญญาณพัลส์ที่มีขนาดความกว้างพัลส์ 2 ms หรือไม่ต่ำกว่า 1.5 ms และจะต้องป้อนสัญญาณพัลส์นี้ทุกๆ 20 ms (หรือในช่วงประมาณ 20ms – 30ms) เช่นกัน



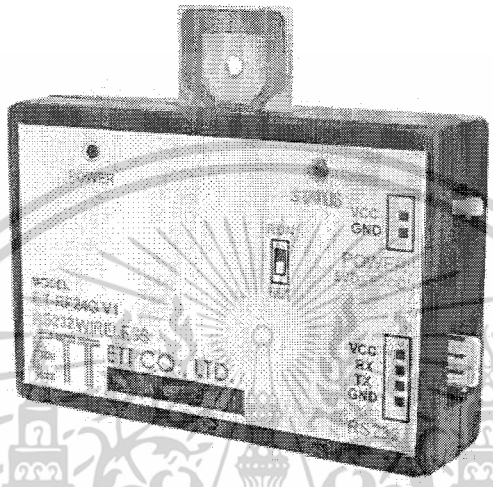
รูปที่ 2.26 การควบคุมให้มอเตอร์หมุนทางด้านขวา

การควบคุมให้มอเตอร์หยุดหมุน ทำได้โดยการส่งลอจิก “0” หรือ “1” ให้กับมอเตอร์ หรือ ก็คือการไม่จ่ายสัญญาณพัลส์ให้กับมอเตอร์นั่นเอง



รูปที่ 2.27 การควบคุมให้มอเตอร์หยุดหมุน

2.9 RS232 to RF-Wireless (RF 2.4 GHz) CONVERTER รุ่น ET-RF24G V1.0



รูปที่ 2.28 RS232 to RF-Wireless (RF2.4GHz) Converter

2.9.1 ลักษณะโดยทั่วไป

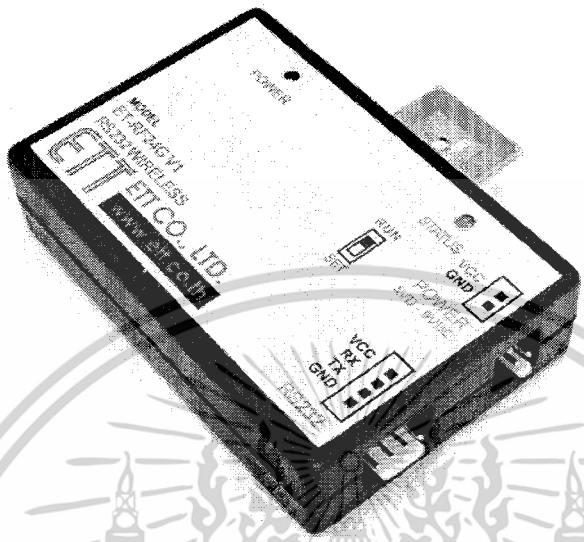
ET-RF24G V1.0 เป็นชุด Signal Converter สำหรับใช้แปลงสัญญาณระหว่าง RS232 และ RF-Wireless โดยในโหมดการทำงานของการส่งข้อมูล (Transmitter) จะทำหน้าที่รอรับข้อมูลจากพอร์ตสื่อสารอนุกรม RS232 จากขา RX แล้วแปลงเป็นสัญญาณความถี่ (GFSK) ส่งออกไปในอากาศ และในทางกลับกันในโหมดการทำงานแบบรับ (Receiver) ชุด ET-RF24G V1.0 ก็จะทำหน้าที่คอยตรวจจับข้อมูลที่อยู่ในรูปของสัญญาณความถี่ (GFSK) จากด้าน RF เพื่อแปลงกลับเป็นข้อมูลแบบ RS232 ส่งออกไปทางขา TX ได้ด้วย

ซึ่งจะเห็นได้ว่าชุดแปลงสัญญาณ ET-RF24G V1.0 นั้นสามารถนำไปต่อใช้งานร่วมกับพอร์ตสื่อสารอนุกรมแบบ RS232 เพื่อใช้งานในลักษณะของการสื่อสารอนุกรมแบบไร้สาย (Transceiver) ได้โดยตรง โดยจะมีข้อดีกว่าคือ สามารถรับส่งข้อมูลกันได้ในระยะที่ไกลกว่า RS232 หลายเท่าตัวและประการสำคัญคือไม่จำเป็นต้องใช้สายสัญญาณที่เป็นตัวนำสัญญาณทางไฟฟ้าในการสื่อสารข้อมูลกัน ทำให้สามารถเปลี่ยนแปลงหรือเคลื่อนย้ายจุดรับส่งข้อมูลได้ตลอดเวลา ซึ่งถ้าเป็นการรับส่งข้อมูลด้วยระบบ RS232 แบบที่ใช้สายสัญญาณนั้นจะเกิดความยุ่งยากในการติดตั้งสายสัญญาณเป็นอย่างมาก

แต่อย่างไรก็ตามการรับส่งข้อมูลโดยใช้อากาศเป็นตัวกลางในการสื่อสารนั้น ก็มีข้อจำกัดบางประการเหมือนกัน โดยเฉพาะอย่างยิ่งเรื่องความน่าเชื่อถือของข้อมูลที่รับส่งกัน ซึ่งมีโอกาศผิดพลาดหรือสูญหายได้เหมือนกันเนื่องจากในการลำเลียงข้อมูลนั้นไม่ได้ใช้สายสัญญาณเป็นตัวกลางในการรับส่งข้อมูล แต่ใช้อากาศเป็นตัวกลางในการรับส่งข้อมูลแทน ซึ่งมีโอกาสที่ข้อมูลจะเกิดการรบกวนจากสัญญาณอื่นๆ ที่มีย่านความถี่ใกล้เคียงกันแล้วทำให้ข้อมูลผิดเพี้ยนไปได้บ้างเหมือนกัน ซึ่งระบบการจัดการข้อมูลของเครื่อง ET-RF24G V1.0 นั้นมีระบบการเข้ารหัสและถอดรหัสข้อมูลที่มีความน่าเชื่อถืออยู่ในเกณฑ์ที่จัดว่าดี โดยข้อมูลแต่ละ Byte ที่มีการรับส่งกันนั้นจะมีการตรวจสอบความถูกต้องของข้อมูลให้ด้วยแล้ว โดยข้อมูลที่รับได้จากด้าน RF นั้นรับประกันได้ว่าเป็นข้อมูลที่มีความถูกต้องแน่นอน แต่อย่างไรก็ตามการรับส่งข้อมูลนั้นมีโอกาสผิดพลาดในเรื่องของการสูญหายของข้อมูลบ้างเหมือนกัน เนื่องจากกลไกในการรับส่งข้อมูลของเครื่อง ET-RF24G V1.0 นั้น จะมีการตรวจสอบข้อมูลทุก Byte ที่รับได้จาก RF เสมอ ซึ่งถ้าพบว่ามีความผิดพลาดเกิดขึ้นจะทิ้งข้อมูล Byte นั้นไป ซึ่งผู้ใช้ควรมีกลไกในการตรวจสอบข้อมูลที่รับส่งกันว่าครบถ้วนหรือไม่ด้วยซึ่งหากพบว่ามีการสูญหายของข้อมูลเกิดขึ้นก็ให้ร้องขอให้มีการส่งข้อมูลนั้นซ้ำใหม่อีกครั้งหนึ่งก็จะสามารถแก้ไขปัญหาดังกล่าวได้

2.9.2 Power Supply

สำหรับการต่อแหล่งจ่ายไฟให้กับเครื่อง ET-RF24G V1.0 นั้นจะสามารถเลือกต่อแหล่งจ่ายไฟให้กับตัวเครื่องได้ 2 ทางด้วยกัน โดยเครื่อง ET-RF24G V1.0 นั้นต้องการไฟเลี้ยงวงจรซึ่งเป็นแหล่งจากกระแสตรง ขนาดประมาณ +5VDC ถึง +9VDC โดยจุดเชื่อมต่อแหล่งจ่ายไฟของเครื่อง ET-RF24G V1.0 นี้ สามารถเชื่องต่อได้ 2 จุดด้วยกันโดยผู้ใช้สามารถเลือกต่อแหล่งจ่ายไฟให้กับเครื่อง ET-RF24G V1.0 จุดใดจุดหนึ่งก็ได้



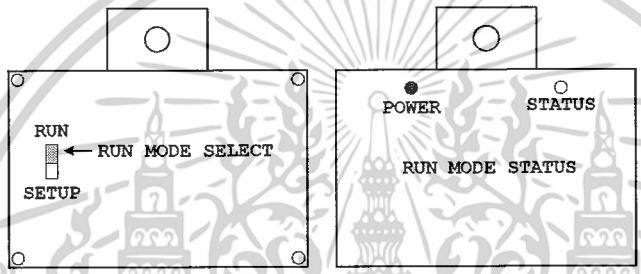
รูปที่ 2.29 แสดงการต่อแหล่งจ่ายไฟภายนอก

2.9.3 โหมดการทำงาน

สำหรับโหมดการทำงานของ ET-RF24G V1.0 นั้นจะแบ่งออกเป็น 2 โหมดด้วยกันโดยการกำหนดโหมดการทำงานของ ET-RF24G V1.0 นั้นจะกระทำผ่าน switch เลือกโหมดที่อยู่ด้านใต้กล่องโดยการเลือกโหมดการทำงานนั้นจะต้องทำให้เสร็จเรียบร้อยก่อนการจ่ายไฟให้กับ ET-RF24G V1.0 ด้วยเสมอ เนื่องจากการทำงานของเครื่อง ET-RF24G V1.0 นั้นจะทำการตรวจสอบโหมดการทำงานของเครื่องจาก Switch เลือกโหมด เฉพาะในช่วงของการจ่ายไฟเลี้ยงให้เครื่องเริ่มต้นทำงานครั้งแรก (Power-ON) เท่านั้น ซึ่งการเปลี่ยนแปลงตำแหน่งการทำงานของ switch เลือกโหมด หลังจากทำการจ่ายไฟให้กับ ET-RF24G V1.0 ไปแล้วจะไม่มีผลต่อการทำงานของเครื่องแต่อย่างใด โดยการทำงานของเครื่อง ET-RF24G V1.0 นั้นจะมี LED แสดงสถานะการทำงานของเครื่องจำนวน 2 หลอดคือ LED POWER ซึ่งเป็น LED สีแดง โดยที่ LED POWER นี้จะติดสว่างให้เห็นตลอดเวลาที่มีการจ่ายไฟเลี้ยงให้เครื่องทำงานอยู่ ส่วน LED อีกดวงนั้นจะเป็น LED สีเขียวใช้แสดงสถานการณ์ทำงานของเครื่องซึ่งเรียกว่า LED STATUS โดย LED STATUS นี้จะเกิดการกระพริบตามจังหวะของการรับส่งข้อมูลกันในแต่ละครั้ง โดยในสภาวะปกตินั้นถ้าเครื่องทำงานอยู่ใน RUN MODE หลอด LED STATUS จะดับอยู่ตลอดเวลาถ้าไม่มีการรับส่งข้อมูลแต่ถ้าตัวเครื่องทำงานอยู่ใน SETUP MODE หลอด LED STATUS จะติดอยู่ตลอดเวลาถ้าไม่มีการรับส่งข้อมูล โดยโหมดการทำงานของ ET-RF24G V1.0 จะมีอยู่ด้วยกัน 2 โหมดคือ

2.9.3.1 การใช้งานเครื่อง ET-RF24G V1.0 ใน RUN MODE

การใช้งานใน RUN MODE ซึ่งเป็นโหมดการใช้งานตามปกติของเครื่อง โดยเมื่อเครื่อง ET-RF24G V1.0 เข้าทำงานในโหมดนี้แล้ว จะสังเกตเห็นหลอดไฟแสดงสถานการณ์ทำงาน หรือ LED STATUS ดับอยู่ แต่เมื่อมีการรับหรือส่งข้อมูลเกิดขึ้น สถานการณ์ทำงานของ LED STATUS จึงจะกระพริบตามจังหวะของการรับส่งข้อมูลนั้นๆ แต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะดับอยู่ตลอดเวลา



รูปที่ 2.30 แสดงการเลือกโหมดการทำงานสำหรับใช้งานปกติ (RUN MODE)

สำหรบการทำงานใน Run Mode นั้นจะแบ่งลักษณะการทำงานออกเป็น 3 แบบด้วยกันโดยลักษณะการทำงานนี้จะถูกกำหนดไว้แล้วใน Configuration ของเครื่องใน Setup Mode ดังนั้นก่อนการใช้งานเครื่องในครั้งแรกจะต้องทำการกำหนดค่า Configuration ต่างๆ ให้เรียบร้อยเสียก่อนโดยเมื่อเครื่อง EF-RF24G V1.0 เริ่มต้นเข้าทำงานใน Run Mode แล้วมันจะทำการอ่านค่า Configuration ที่เก็บไว้ออกมาเพื่อใช้เป็นเงื่อนไขในการทำงานตามค่าที่ได้กำหนดไว้ โดยลักษณะการทำงานใน Run Mode แบ่งออกเป็นดังนี้

2.9.3.1.1 การทำงานแบบ RF Receive Only

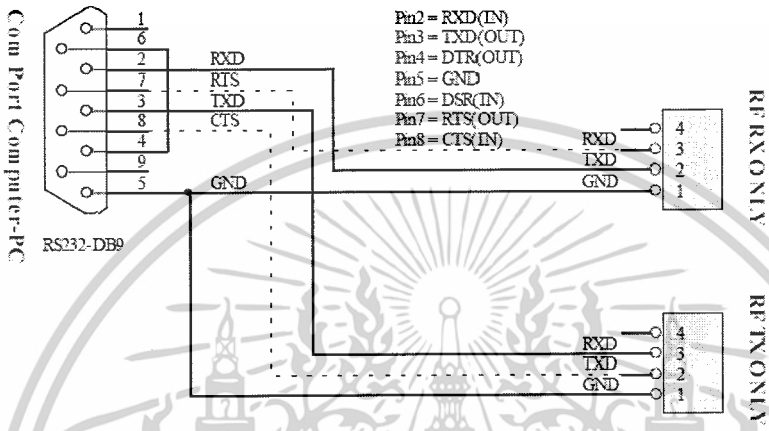
เป็นการทำงานแบบทิศทางเดียว โดยการทำงานในโหมดนี้จะเป็นการรอรับข้อมูลความถี่แบบ GFSK จากด้าน RF แล้วเปลี่ยนเป็นข้อมูลอนุกรมส่งออกไปทางขา TX (Transmit) ของ RS232 โดยการทำงานจะวนรอบอยู่เช่นนี้ไปตลอด ซึ่งในการใช้งานเครื่อง ET-RF24G V1.0 ในโหมดนี้จะต้องนำสัญญาณ TX (Transmit) ไปต่อขาสัญญาณ RX (Receive) ของอุปกรณ์ด้านตรงข้าม (RS232 ของคอมพิวเตอร์ PC) โดยในโหมดนี้การทำงานของขาสัญญาณ RX ด้าน RS232 ของเครื่อง ET-RF24G V1.0 จะถูกเปลี่ยนหน้าที่เป็นสัญญาณ CTS (Clear To Send) สำหรับใช้ตรวจสอบความพร้อมในการส่ง

ข้อมูลไปให้อุปกรณ์ด้านตรงข้ามแทน ซึ่งในการใช้งานจะต้องนำสัญญาณนี้ไปต่อเข้ากับสัญญาณ RTS (Ready To Send) ของอุปกรณ์ด้านตรงข้าม โดยเครื่อง ET-RF24G V1.0 จะทำการตรวจสอบสถานะของสัญญาณ RX ซึ่งในโหมดนี้เสมือน CTS ว่ามีค่าเป็น "0" หรือไม่ โดยถ้าพบว่าเป็น "0" จึงจะส่งข้อมูลออกไปให้ทางขา TX แต่ถ้าพบว่าสถานะของขาสัญญาณนี้เป็น "1" แสดงว่าอุปกรณ์ด้านตรงข้ามยังไม่พร้อมรับข้อมูลก็จะรอจนกว่าจะพบว่าสถานะของสัญญาณดังกล่าวมีค่าเป็น "0" จึงจะส่งข้อมูลออกไปให้ โดยเครื่อง ET-RF24G V1.0 จะสามารถจัดเก็บข้อมูลไว้ใน Buffer เพื่อรอการส่งได้สูงสุด 64 Byte เท่านั้น ซึ่งถ้าในระหว่างที่รอความพร้อมอยู่นั้นมีข้อมูลด้าน RF ส่งเข้ามาเกินกว่า 64 Byte จะทำให้ข้อมูลที่เกินมานั้นสูญหายไป

2.9.3.1.2 การทำงานแบบ RF Transmit Only

เป็นการทำงานแบบทิศทางเดียว โดยการทำงานในโหมดนี้จะมีลักษณะตรงกันข้ามกับ RF Receive Only กล่าวคือเครื่อง ET-RF24G V1.0 จะทำหน้าที่รอรับข้อมูลจากขา Rx(Receive) ด้าน RS232 แล้วเปลี่ยนเป็นข้อมูลแบบ GFSK ส่งออกไปทางด้าน RF โดยการใช้งานเครื่องในโหมดนี้จะต้องนำสัญญาณ TX(Transmit) ซึ่งเป็นขาส่งข้อมูลจาก RS232 ของอุปกรณ์ด้านตรงข้ามมาต่อเข้ากับขา RX(Receive) ของเครื่อง ET-RF24G V1.0 ส่วนขาสัญญาณ TX จะถูกเปลี่ยนหน้าที่เป็น RTS(Ready To Send) เพื่อใช้แสดงสถานะความพร้อมในการรับข้อมูลจากด้าน RS232 ซึ่งในการใช้งานจะต้องนำสัญญาณ Tx ซึ่งในขณะนี้เปรียบเสมือนกับ RTS นำไปต่อเข้ากับสัญญาณ CTS (Clear To Send) ของอุปกรณ์ด้านตรงข้ามเพื่อใช้ในการตรวจสอบความพร้อมในการรับข้อมูลโดยอุปกรณ์ด้านตรงข้ามจะต้องทำการตรวจสอบสถานะของสัญญาณ RTS นี้ เพื่อตรวจสอบความพร้อมในการรับข้อมูลของ ET-RF24G V1.0 ด้วย โดยถ้าเครื่อง ET-RF24G V1.0 พร้อมรับข้อมูลจาก RS232 มันจะส่งสัญญาณจาก RTS ให้มีค่าเป็น "0" รอไว้ และเมื่อใดก็ตามที่การรับข้อมูลทางด้านของ RS232 มีจำนวนข้อมูลที่ยังไม่สามารถเปลี่ยนเป็น GFSK เพื่อส่งออกไปทางด้าน RF ได้ทันจนเกือบจะเต็ม Buffer แล้วเครื่อง EF-RF24G V1.0 จะทำการส่งสัญญาณ RTS ให้มีค่าเป็น "1" ออกไปบอกให้อุปกรณ์ด้านตรงข้ามทราบเพื่อจะได้หยุดการส่งข้อมูลออกมา โดยอุปกรณ์ตรงข้ามจะต้องหยุดการส่งข้อมูลและรอจนกว่าสถานะของสัญญาณ RTS จะกลับเป็น "0" จึงจะเริ่มต้นส่งข้อมูลออกมาใหม่ ซึ่งหลังจากที่เครื่อง ET-RF24G V1.0 ส่งสัญญาณ RTS ด้วยค่า "1" ออกไปแล้วจะยังคงสามารถรับข้อมูลได้เพิ่มเติมอีกไม่เกิน 16 Byte เท่านั้น ซึ่งถ้าอุปกรณ์ด้านตรงข้ามยังส่งข้อมูลต่อเนื่องมาอีกจนเกินขนาดของ Buffer แบบ Full Duplex ที่เครื่อง ET-RF24G V1.0 จำนวน 4 ชุด มาต่อใช้งานร่วมกันเพื่อใช้งานในการรับส่งข้อมูลกันแบบ Full Duplex

โดยแบบการใช้งานออกเป็น 2 ด้าน คือ ต้นทาง และ ปลายทาง ด้านละ 2 ชุด โดยแต่ละด้านให้กำหนดหน้าที่การทำงานเป็น RF Receive Only 1 ชุด และ RF Transmit Only อีก 1 ชุด



รูปที่ 2.31 แสดงสายสัญญาณ RS232 เพื่อใช้กับ ET-RF24G ในโหมด RF Receive และ RF Transmit Only

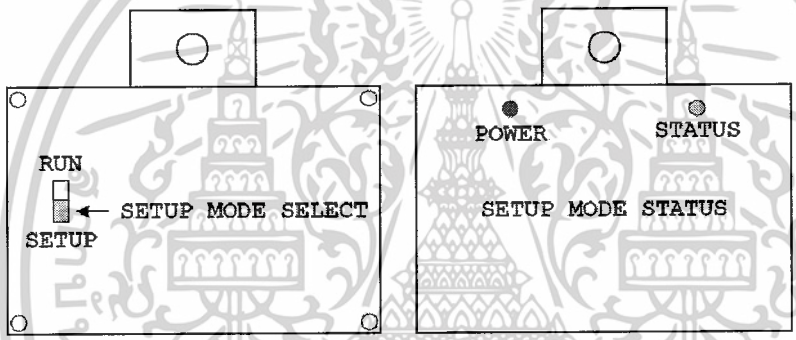
2.9.3.1.3 การทำงานแบบ RF Auto Direction

เป็นการทำงานชนิด 2 ทิศทาง แบบ Half Duplex หรือ ผลัดกันรับผลัดกันส่ง ซึ่งสามารถใช้รับส่งข้อมูลระหว่างต้นทาง และ ปลายทาง ได้โดยใช้เครื่อง ET-RF24G V1.0 ด้านละ 1 ชุด เท่านั้น เพียงแต่การรับส่งข้อมูลแบบนี้จะไม่สามารถส่งข้อมูลสวนทางกันได้เหมือนกับแบบ Full Duplex แต่จะต้องใช้วิธีการผลัดกันรับข้อมูลและส่งข้อมูลแทน โดยเมื่อฝ่ายรับทำการรับข้อมูลได้จนครบแล้วจึงจะสลับหน้าที่เป็นฝ่ายส่งเพื่อส่งข้อมูลย้อนกลับไป โดยในโหมดนี้ เครื่อง ET-RF24G V1.0 จะทำหน้าที่เป็นทั้ง ฝ่ายรับ และฝ่ายส่ง ข้อมูล แบบอัตโนมัติ โดยในสภาวะปกติจะอยู่ในสภาวะของการรอรับข้อมูลทั้งด้าน RF และ RS232 ซึ่งถ้าพบว่ามีข้อมูลส่งเข้ามาทางด้านของ RF ก็จะนำข้อมูลนั้นส่งออกไปทางด้านขา TX ของ RS232 ทันที และในทำนองเดียวกัน ถ้าพบว่ามีข้อมูลส่งเข้ามาทางด้าน RX ของ RS232 มันก็จะทำการรับข้อมูลนั้นจาก RS2323 พร้อมกับเปลี่ยนทิศทางของอุปกรณ์ RF จากการรอรับข้อมูลให้ทำหน้าที่เป็นตัวส่งข้อมูลแทน เพื่อทำการส่งข้อมูลที่รับได้จาก RS232 ออกไปทาง RF ในทันที ซึ่งหลังจากที่เครื่อง ET-RF24G V1.0 ทำการสลับโหมดการทำงานของอุปกรณ์ด้าน RF จากการรอรับเป็นการส่งและทำการเริ่มต้นส่งข้อมูลออกไปทางด้าน RF เรียบร้อยแล้ว มันจะวนกลับไปตรวจสอบการรับข้อมูลจากด้าน RS232 อีกว่ายังมีข้อมูลส่งเข้ามาอีกหรือไม่ ถ้าพบว่ายังมีข้อมูลส่งเข้า

อีกก็จะทำการแปลงข้อมูลนั้นเพื่อส่งออกไปยังด้าน RF ต่อไปอีกจนกว่าการส่งข้อมูลด้าน RS232 จะสิ้นสุดลง ซึ่งข้อมูลด้าน RS232 ที่ส่งเข้ามานั้นควรส่งอย่างต่อเนื่อง โดยเมื่อเครื่อง ET-RF24G V1.0 ทำการส่งข้อมูลแต่ละ Byte ออกไปทางด้าน RF เรียบร้อยแล้วมันจะวนรอบรอรับข้อมูล Byte ถัดไปจาก RS232 ภายในเวลา 2.5 mS ถ้าไม่พบข้อมูลส่งเข้ามาอีกภายในระยะเวลาดังกล่าวมันจึงจะทำการเปลี่ยนหน้าที่ของอุปกรณ์ด้าน RF ให้กลับมาทำกน้าที่เป็นการรอรับข้อมูลตามเดิม โดยในขณะที่อุปกรณ์ด้าน RF ถูกกำหนดให้เป็นฝ่ายส่งข้อมูลอยู่นั้นจะไม่สามารถทำการรับข้อมูลจาก RF ได้ ซึ่งถ้ามีการส่งข้อมูลเข้ามาในขณะนั้นก็จะไม่สามารถรับได้ โดยค่าเวลาที่จะใช้ในการสลับโหมดการทำงานของ RF จากฝ่ายส่งข้อมูลให้เป็นฝ่ายรับข้อมูลนั้นจะมีค่าเป็น 2.5 mS ดังนั้นเมื่อฝ่ายรับสามารถรับข้อมูลได้ครบหมดแล้วก่อนที่จะทำการส่งข้อมูลเพื่อตอบกลับไปยังฝ่ายตรงข้ามนั้นควรทำการหน่วงเวลาไว้ไม่น้อยกว่า 3 mS นับจากรับข้อมูล Byte สุดท้ายได้เรียบร้อยแล้วจึงเริ่มต้นส่งข้อมูล Byte แรกย้อนกลับไป ซึ่งถ้าฝ่ายรับทำการส่งข้อมูลตอบกลับไปยังฝ่ายตรงข้ามเร็วกว่านี้อาจทำให้ฝ่ายตรงข้ามไม่สามารถรับข้อมูล Byte แรกได้ทัน สำหรับการใช้งานเครื่อง ET-RF24G V1.0 ในโหมด RF Auto Direction นี้ การรับและส่งข้อมูลด้าน RS232 จะไม่มีการตรวจสอบความพร้อมของฝ่ายรับและส่งด้วยสัญญาณทางไฟฟ้า (CTS/RTS) เหมือนกับการใช้งานใน 2 โหมดที่ผ่านมาแล้ว โดยเมื่อมันสามารถรับข้อมูลจาก RF ได้ ก็จะทำการส่งข้อมูลนั้นออกไปทางขา TX (Transmit) ของ RS232 ในทันที โดยไม่สนใจว่า อุปกรณ์ที่ต่อไว้ด้าน RS232 จะพร้อมรับข้อมูลหรือไม่ ซึ่งถ้าด้าน RS232 ไม่พร้อมรับข้อมูลก็จะทำให้ข้อมูล Byte นั้นสูญหายไปทันที ซึ่งในการใช้งานนั้น ผู้ใช้ควรกำหนดค่าความเร็วในการรับส่งข้อมูลด้าน RS232 ที่จะใช้กับเครื่อง ET-RF24G V1.0 ทุกๆ ตัวด้วยค่าความเร็วที่เท่ากันด้วย เพื่อให้การรับและส่งข้อมูลเกิดความสัมพันธ์กันอย่างเหมาะสม สำหรับความสามารถในการรอรับข้อมูลจาก RS232 ของเครื่อง ET-RF24G V1.0 ในโหมดนี้ จะสามารถรับข้อมูลได้อย่างต่อเนื่องสูงสุดไม่เกิน 64 Byte ดังนั้นในกรณีที่มีการส่งข้อมูลจากด้าน RS232 ด้วยข้อมูลจำนวนมากกว่า 64 Byte ต่อเนื่องกันนั้น ควรทำการแบ่งข้อมูลออกเป็นชุดๆ โดยให้มีขนาดชุดละไม่เกิน 64 Byte ซึ่งหลังจากทำการส่งข้อมูลอย่างต่อเนื่องไปได้ 1 ชุด (64 Byte) แล้วควรทำการหน่วงเวลาไว้ชั่วขณะหนึ่งอย่างน้อย 1 mS แล้วจึงเริ่มส่งข้อมูลชุดถัดไป สลับกับการหน่วงเวลาอย่างนี้เรื่อยๆ เพื่อให้เครื่อง ET-RF24G V1.0 สามารถนำข้อมูลที่รับได้จากด้าน RS232 ส่งออกไปทางด้าน RF ได้ทัน ซึ่งถ้าทำการส่งข้อมูลอย่างต่อเนื่องโดยไม่มีการหน่วงเวลาเลยอาจทำให้ข้อมูลบาง Byte เกิดการสูญหายไปได้

2.9.3.2 การใช้งานเครื่อง ET-RF24G V1.0 ใน Setup Mode

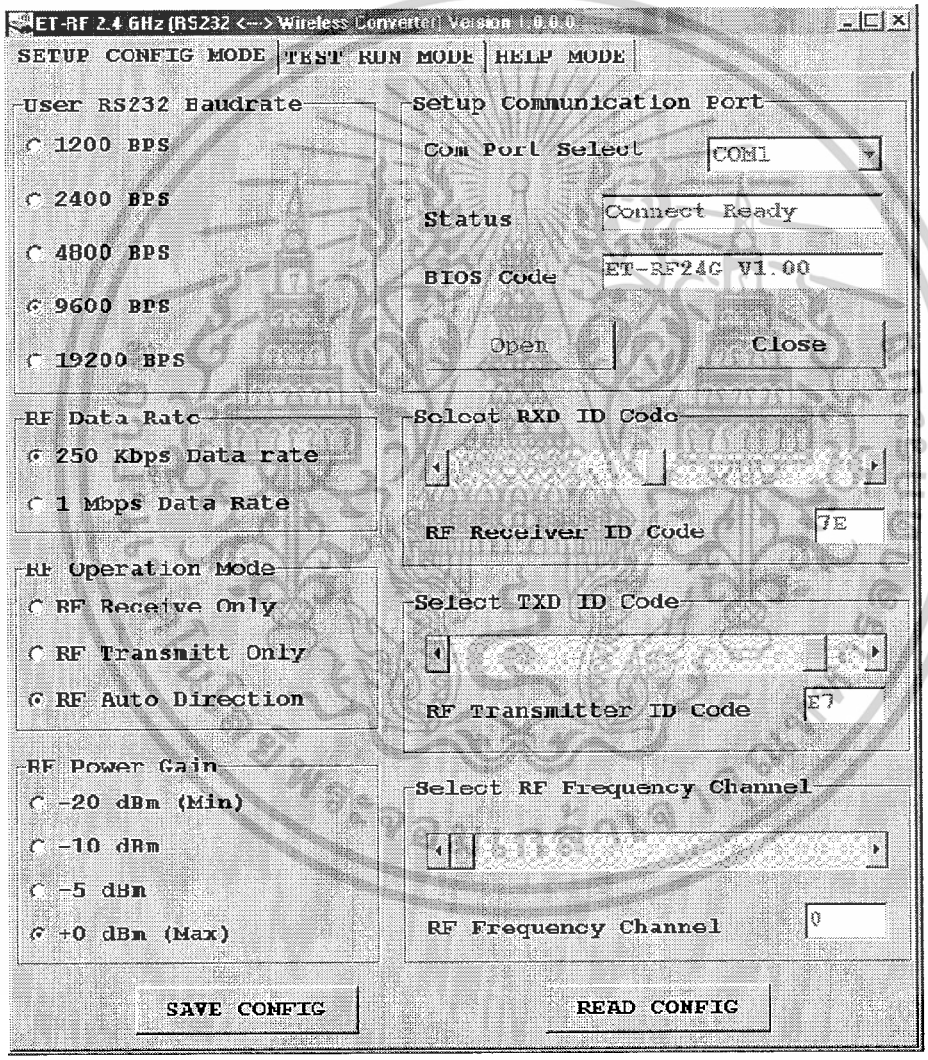
การใช้งานเครื่อง ET-RF24G V1.0 ใน Setup Mode ซึ่งเป็นโหมดสำหรับใช้กำหนดค่า Configuration ต่างๆ สำหรับควบคุมการทำงานของเครื่อง ET-RF24G V1.0 ที่จะใช้ในขณะที่เครื่องทำงานอยู่ใน Run Mode โดยในการ Setup ค่า Configuration ต่างๆ นั้นจะกระทำร่วมกับโปรแกรม "ET_RF24G_v1.EXE" ของอีทีที ซึ่งเมื่อเครื่อง ET-RF24G V1.0 เข้าทำงานในโหมด Setup แล้วจะสังเกตเห็นหลอดไฟแสดงสถานะการทำงาน หรือ LED STATUS ติดสว่างค้างอยู่ตลอดเวลา แต่เมื่อมีการสั่งอ่านหรือเขียนข้อมูลกับบอร์ด สถานการณ์ทำงานของ LED STATUS จึงจะกระพริบตามจังหวะของการรับส่งข้อมูล แต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะติดค้างอยู่ตลอดเวลา



รูปที่ 2.32 แสดงการเลือกโหมดการทำงานสำหรับกำหนดค่า Configuration (Setup Mode)

ซึ่งการกำหนดค่า Configuration ให้กับ ET-RF24G V1.0 นั้นจะต้องกระทำในขณะที่ตัวเครื่องทำงานอยู่ใน Setup Mode เท่านั้น (เลือก Switch กำหนดโหมดไว้ทางด้าน Setup แล้วจ่ายไฟให้เครื่องเริ่มต้นทำงาน) โดยค่าของ Configuration ต่างๆ นั้นจะถูกใช้สำหรับเป็นเงื่อนไขในการทำงานของ ET-RF24G V1.0 ในขณะที่อยู่ใน Run Mode ดันนั้นก่อนการเริ่มต้นใช้งานเครื่องในครั้งแรกนั้นจึงจำเป็นอย่างยิ่งที่จะต้องทำการกำหนดค่าของ Configuration ต่างๆ ให้ถูกต้องและตรงกับความต้องการที่จะใช้งานเสียก่อน โดยเมื่อทำการกำหนดค่าตัวเลือกต่างๆ ของ Configuration เรียบร้อยแล้ว ก็สามารถเปลี่ยนโหมดการทำงานของตัวเครื่องกลับเป็น Run Mode พร้อมกับการปิดไฟที่จ่ายให้กับตัวเครื่อง (Power-OFF) ชั่งขณะหนึ่ง จากนั้นจึงเริ่มต้นจ่ายไฟให้กับตัวเครื่องใหม่ (Power-ON) ก็สามารถใช้งาน ET-RF24G V1.0 ตามค่าของ Configuration ที่กำหนดไว้แล้วได้ทันทีโดยค่าตัวเลือกต่างๆ ของ Configuration ที่ได้กำหนดไว้แล้วจะถูกเก็บไว้ภายในตัวเครื่องอย่างถาวรถึงแม้ว่าจะไม่ได้ทำการจ่ายไฟให้กับตัวเครื่องแล้วก็ตาม ดังนั้นเมื่อทำการกำหนดค่า Configuration ต่างๆ เรียบร้อยแล้ว ถ้าไม่มีการ

เปลี่ยนแปลงเงื่อนไขการทำงานของตัวเครื่องต่างไปจากเงื่อนไขเดิมที่ได้กำหนดไว้แล้ว ก็ไม่จำเป็นต้องทำการกำหนดค่า Configuration ใหม่อีกแต่อย่างใด โดยทุกๆ ครั้งที่เริ่มต้นจ่ายไฟเข้าเครื่องในครั้งแรกนั้น การทำงานของ ET-RF24G V1.0 จะเป็นไปตามเงื่อนไขที่กำหนดไว้ใน Configuration เสมอทุกๆ ครั้ง โดยคุณสมบัติของ Configuration ต่างๆ นั้นมีดังนี้



รูปที่ 2.33 แสดงรูปโปรแกรมที่ใช้สำหรับกำหนดค่า Configuration ของ ET-RF24G V1.0

- **User RS232 Baudrate** ใช้สำหรับกำหนดค่าความเร็วในการรับส่งข้อมูลทางด้าน RS232 ของตัวเครื่อง ในขณะที่ทำงานอยู่ใน Run Mode ซึ่งสามารถกำหนดได้ 5 ค่าคือ

  - 1200 BPS
  - 2400 BPS
  - 4800 BPS
  - 9600 BPS
  - 19200 BPS

- **RF Data Rate** ใช้สำหรับกำหนดความเร็วในการรับส่งข้อมูลทางด้าน RF ของ ET-RF24G V1.0 ซึ่งจะต้องกำหนดให้เครื่อง ET-RF24G V1.0 ทุกๆ ตัว ที่จะนำมาใช้ติดต่อสื่อสารกัน มีค่าอัตราความเร็วต่างกันจะไม่สามารถรับส่งข้อมูลกันได้ ซึ่งค่าอัตราความเร็วในการส่งข้อมูลนี้จะมีผลต่อระยะทางการรับส่งข้อมูลด้วย ซึ่งถ้าใช้ความเร็วในการส่งสูง (1 Mbps) จะทำให้รัศมีการรับส่งข้อมูลได้ระยะสั้นลง แต่ถ้าใช้ความเร็วในการรับส่งข้อมูลที่ช้าลง (250 Kbps) จะทำให้ได้รัศมีการรับส่งไกลขึ้น โดยค่า RF Data Rate สามารถกำหนดได้ 2 ค่า คือ

  - 250 Kbps
  - 1 Mbps

- **RF Operation Mode** ใช้สำหรับกำหนดโหมดการทำงานของ ET-RF24G V1.0 ซึ่งสามารถกำหนดหน้าที่การทำงานได้ 3 แบบ ด้วยกันคือ

- RF Receive Only เป็นการกำหนดให้ ET-RF24G V1.0 ทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RF เพื่อเปลี่ยนเป็นข้อมูลแบบ RS232 และส่งออกไปทางด้านขา TX ของ RS232 ตลอดเวลา
- RF Transmit Only เป็นการกำหนดให้ ET-RF24G V1.0 ทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RS232 จากขา RX เพื่อเปลี่ยนเป็นข้อมูลแบบ GFSK และส่งออกไปทางด้าน RF ตลอด เวลา
- RF Auto Direction เป็นการกำหนดโหมดการทำงานแบบ Half Duplex 2 ทิศทาง ซึ่งสามารถสลับโหมดการทำงานระหว่างการรับและส่งข้อมูลได้เองโดยอัตโนมัติ โดยในโหมดการทำงานนี้ เครื่อง ET-RF24G V1.0 รอตรวจสอบข้อมูลทั้งจากด้าน RS232 แล้วด้าน RF อยู่ตลอดเวลาโดยถ้าได้รับข้อมูลจากด้าน RS232 ก็จะทำการแปลงแล้วส่งออกทางด้าน RF จากนั้นก็จะกำหนดให้ด้าน RF กลับมาเป็นฝ่ายรอรับข้อมูลตามเดิมและเมื่อได้รับข้อมูลจากด้าน RF ก็จะแปลเป็นข้อมูลแล้วส่งออกไปทางด้าน RS232 โดยอัตโนมัติ

- RF Power Gain เป็นการกำหนดกำลังส่งของวงจร RF Power ที่ใช้ในการส่งข้อมูล โดยค่า +0 dBm เป็นค่ากำลังส่งสูงสุด ส่วน -20 dBm เป็นค่ากำลังส่งต่ำสุด โดยสามารถกำหนดได้ 4 ระดับคือ

- -20 dBm (กำลังส่งต่ำสุด)
- -10 dBm
- -5 dBm
- +0 dBm (กำลังส่งสูงสุด)

- RXD ID Code เป็นรหัส ID Code ของเครื่อง ET-RF24G V1.0 ในโหมดของการรับข้อมูลจาก RF โดยเมื่อเครื่อง ET-RF24G V1.0 ด้านส่งจะทำการส่งข้อมูลออกไปทาง RF นั้นจะมีการระบุหมายเลข ID Code ของด้านรับรวมไปกับชุดข้อมูลด้วยเสมอ โดยเมื่อเครื่อง ET-RF24G V1.0 ที่อยู่ทางด้านรับทำการรับข้อมูลจากด้าน RF ได้ อันดับแรกมันจะทำการเปรียบเทียบรหัส ID Code ที่รวมมากับข้อมูลที่รับมาได้ว่าตรงกับรหัสของRXD ID Code ที่กำหนดไว้ในตัวมันหรือไม่ ซึ่งถ้าถูกต้องก็จะแยกเอาเฉพาะส่วนของข้อมูลที่รับเข้ามาได้เพื่อเปลี่ยนเป็นข้อมูลแบบ RS232 แล้วส่งออกไปทางด้าน TX ของ RS232 แต่ถ้ารหัส ID Code ที่รับมาได้ไม่ตรงกับรหัส RXD ID Code ที่กำหนดไว้ เครื่อง ET-RF24G V1.0 จะทิ้งข้อมูลชุดนั้นไปทันที โดยคา RXD ID Code นั้นสามารถกำหนดได้ 256 ค่าในรูปแบบของเลขฐานสิบหก (00H-FFH)

- TXD ID Code เป็นรหัส ID Code ปลายทางที่จะส่งข้อมูลไปหาโดยที่เครื่อง ET-RF24G V1.0 ที่ถูกกำหนดให้ทำหน้าที่เป็นฝ่ายส่งข้อมูลนั้น เมื่อมันสามารถรับข้อมูลจาก RS232 ได้แล้วมันจะทำการนำเอาข้อมูลนั้นไปเข้ารหัสรวมกับ TXD ID Code ที่กำหนดไว้ แล้วส่งออกไปทางด้าน RF โดยรหัสของ TXD ID Code นี้หมายถึงรหัส RXD ID Code ของฝ่ายรับที่ต้องการส่งข้อมูลไปหานั่นเองโดยค่า TXD ID Code นั้นสามารถกำหนดได้ 256 ค่าในรูปแบบของเลขฐานสิบหก (00H-FFH)

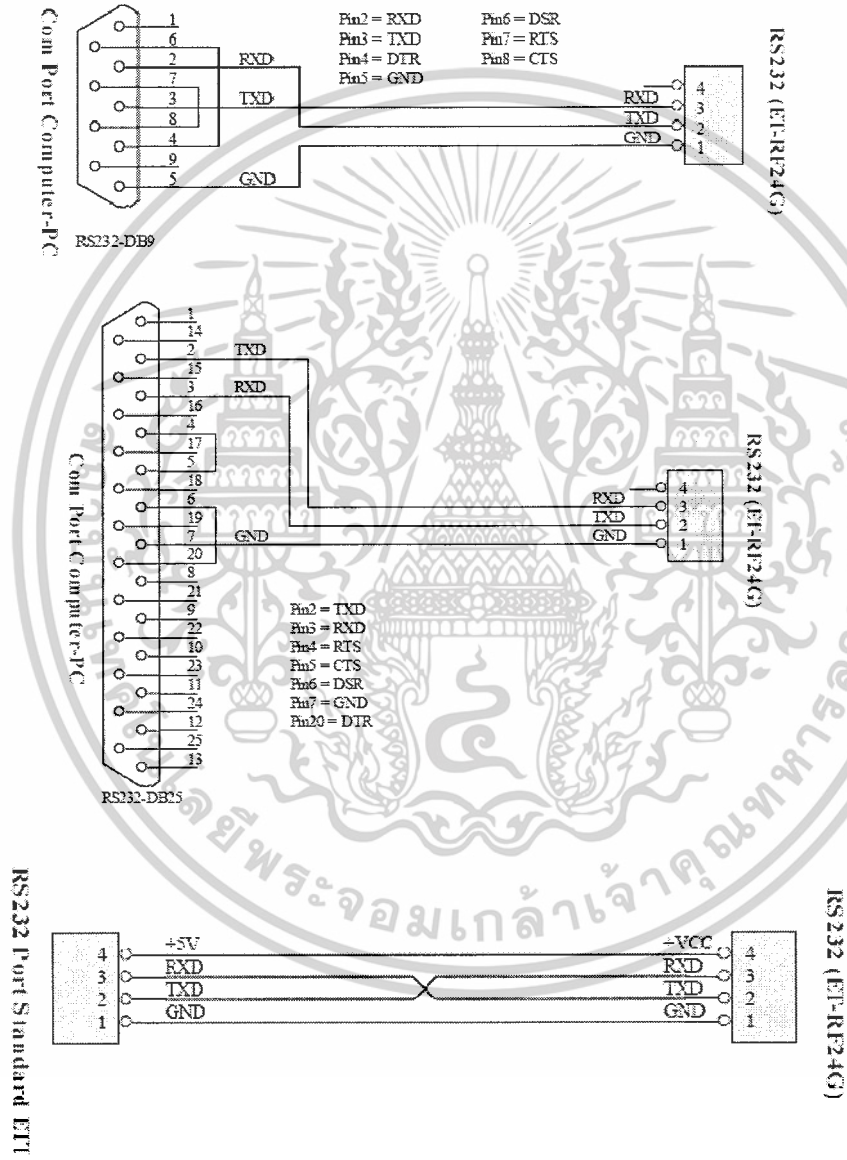
- RF Frequency Channel เป็นการกำหนดค่าของช่องความถี่ที่จะใช้ในการรับส่งข้อมูลกัน โดยสามารถเลือกกำหนดช่องความถี่ได้สูงสุดมากถึง 125 ช่อง (0-124) โดยการที่เครื่อง ET-RF24G V1.0 จะทำการรับส่งข้อมูลกันได้นั้นจะต้องกำหนดช่องความถี่ที่ตรงกันและใช้อัตราความเร็ว RF Data Rate ที่เท่ากันด้วย ซึ่งที่สามารถเลือกกำหนดช่องความถี่ RF Frequency Channel ได้นั้น จะมีประโยชน์เป็นอย่างมากในกรณีที่มีการใช้งานเครื่อง ET-RF24G V1.0 จำนวนหลายๆ กลุ่มในบริเวณพื้นที่ใกล้เคียงกัน โดยให้กำหนดช่องความถี่ของ ET-RF24G V1.0 กลุ่มที่จะสื่อสารข้อมูลร่วมกันไว้ที่ช่องความถี่เดียวกัน ส่วนกลุ่มอื่นๆ ก็ให้เลือกกำหนดช่องความถี่ที่แตกต่างกันออกไป เพื่อลดปัญหาการรบกวนกัน

ข้อแนะนำในการกำหนดค่า Configuration

การกำหนดค่า Configuration ให้กับเครื่อง ET-RF24G V1.0 นั้นสามารถเลือกกำหนดได้ตามความต้องการและจุดประสงค์ของการใช้งานโดยแต่ละโหมดของการใช้งานนั้นจะมีค่า Configuration ที่เหมาะสมต่างกันซึ่งขอแนะนำวิธีการกำหนดค่า Configuration ดังแนวทางต่อไปนี้

- ความเร็วในการรับส่งข้อมูลด้าน RS232 หรือ User RS232 Baudrate ที่ความเร็ว 19200 Bps นั้นเหมาะกับการใช้งาน ET-RF24G V1.0 แบบ Receive Only หรือ Transmit Only ซึ่งมีการตรวจสอบความพร้อมของสัญญาณในการรับส่งข้อมูลกันด้วย แต่ถ้าต้องการใช้งานเครื่อง ET-RF24G V1.0 ในโหมด Auto Direction นั้น ควรกำหนดค่า User RS232 Baudrate ไว้ที่ความเร็วไม่เกิน 9600 Bps จะดีที่สุดและกำหนดค่า Baudrate ของทั้งสองฝ่ายให้มีค่าเท่ากันด้วย
- ค่าความเร็วของการรับส่งข้อมูลด้าน RF หรือ RF Data Rate ที่สามารถรับส่งข้อมูลกันได้ระยะทางไกลมากที่สุดและมีโอกาศผิดพลาดน้อยที่สุด คือ 250 Kbps
- ค่า RF Power Gain ที่ดีที่สุดคือ 0 dBm ซึ่งเป็นค่ากำลังส่งสูงสุดซึ่งจะทำให้สามารถส่งข้อมูลได้ระยะทางไกลที่สุดแต่ถ้าระยะการรับส่งข้อมูลไม่ไกลกันมากและมีการใช้งานเครื่อง ET-RF24G V1.0 จำนวนหลายๆ กลุ่มในพื้นที่ใกล้เคียงกันก็อาจทำการลดกำลังส่งให้ต่ำลงเพื่อลดปัญหาการรับกวนกันหรือกำหนดช่องความถี่ RF Frequency Channel ให้ห่างกันมากๆ
- ในกรณีที่มีการใช้เครื่อง ET-RF24G V1.0 หลายๆ กลุ่มในพื้นที่ใกล้เคียงกันควรกำหนดช่องความถี่ในการใช้งาน หรือ RF Frequency Channel ให้ห่างกันด้วยเพื่อป้องกันการรบกวนกัน
- การใช้งานเครื่อง ET-RF24G V1.0 แบบ Auto Direction นั้น ถ้ามีการส่งข้อมูลจำนวนมากๆ ควรจัดแบ่งข้อมูลออกเป็นชุดๆ โดยให้มีขนาดข้อมูลชุดละไม่เกิน 64 Byte โดยในการส่งข้อมูลแต่ละชุดนั้นให้ทำการส่งข้อมูลอย่างต่อเนื่องโดยให้ข้อมูลแต่ละ Byte มีระยะเวลาห่างกันไม่เกิน 2.5 mS เนื่องจากถ้าข้อมูลขาดหายไปนานกว่านี้ เครื่อง ET-RF24G V1.0 จะทำการเปลี่ยนโหมดของการส่งข้อมูลกลับเป็นโหมดของการรับข้อมูลแทน ซึ่งเมื่อมีการส่งข้อมูล Byte ถัดไปมาอีกก็จะต้องเสียเวลาในการสลับโหมดจากฝ่ายรอรับข้อมูลให้เป็นฝ่ายส่งข้อมูลอีก ซึ่งจะทำให้ประสิทธิภาพในการจัดส่งข้อมูลลดลงเนื่องจากต้องเสียเวลาในการสลับโหมดการทำงานของวงจรภาค RF อยู่ตลอดเวลา โดยที่เมื่อทำการจัดส่งข้อมูลครบ 64 Byte แล้ว ให้ทำการหน่วงเวลาไว้ชั่วขณะหนึ่งประมาณ 1 mS-2 mS แล้วจึงส่งข้อมูลชุดถัดไปอีกอย่างนี้เรื่อยๆ จะทำให้การรับส่งข้อมูลมีประสิทธิภาพสูงสุด

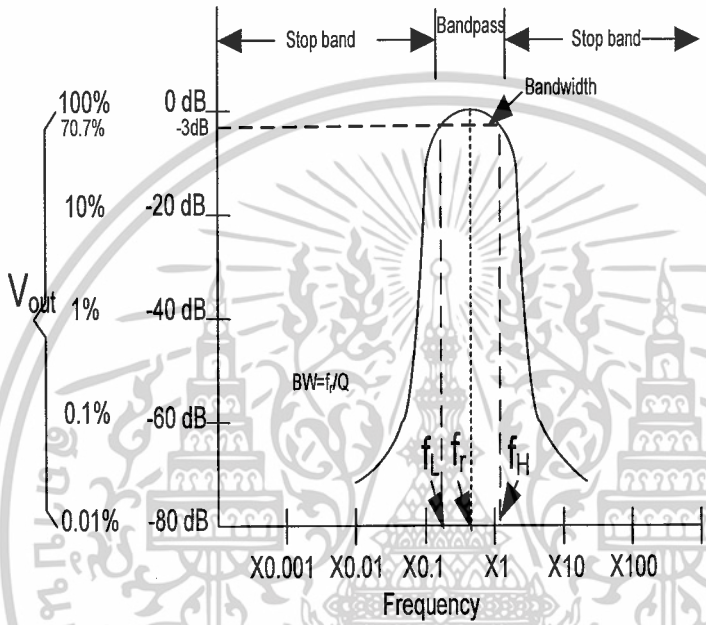
- การใช้งานเครื่อง ET-RF24G V1.0 แบบ Auto Direction นั้นควรหน่วงเวลาในการสลับโหมดจากฝ่ายของการรอรับข้อมูลเป็นฝ่ายส่งข้อมูล อย่างน้อยที่สุด 3 mS-5 mS ซึ่งถ้าส่งข้อมูลย้อนกลับด้วยเวลาที่เร็วกว่านี้อาจทำให้ฝ่ายตรงข้ามไม่สามารถรับข้อมูล Byte แรกได้ทัน



รูปที่ 2.34 แสดงแผนผังการต่อสาย RS232 เพื่อใช้งานกับ ET-RF24G V1.0 ในโหมด Auto Direction

2.10 วงจรกรองแถบความถี่ผ่าน (Band Pass Filter)

วงจรกรองความถี่ผ่านเป็นช่วงหรือแบนด์พาสฟิลเตอร์ คือวงจรที่ยอมให้สัญญาณบางความถี่ผ่านได้เท่านั้น รูปที่ 5.1 แสดงคุณสมบัติการตอบสนองต่อความถี่ของวงจรกรองความถี่เป็นช่วง



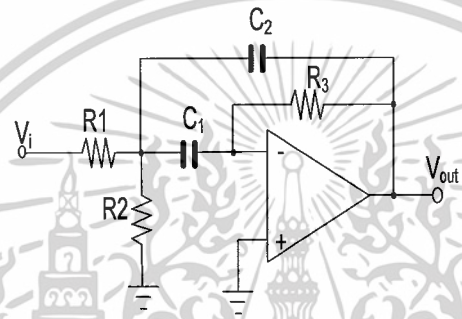
รูปที่ 2.35 ผลตอบสนองความถี่ของแบนด์พาสฟิลเตอร์

จากรูป จะพบว่า ณ ความถี่ที่เอาต์พุตมีขนาดสูงสุดเราเรียกว่าความถี่รีโซแนนท์ และที่ความถี่ซึ่งแรงดันเอาต์พุตลดลงเหลือ 70.7% ทั้งด้านที่ความถี่สูงขึ้นและที่ความถี่ลดลงเรียกว่า f_H และ f_L ตามลำดับ โดยที่ผลต่างของความถี่ทั้งสองนี้ $(f_H - f_L)$ แสดงแบนด์วิด (BW) ของวงจรถ้า BW มีค่าต่ำกว่า 10% ของความถี่รีโซแนนท์ (f_r) จะเรียกวงจรนี้ว่าฟิลเตอร์ช่วงแคบ แต่จะเรียกเป็นวงจรฟิลเตอร์ช่วงกว้างหากแบนด์วิดมีค่าสูงกว่า 10% ของ f_r นอกจากนี้ยังให้นิยามสำหรับค่า Q (quality factor) ว่าเป็นสัดส่วนของความถี่รีโซแนนท์และแบนด์วิด ดังสมการ

$$Q = \frac{f_r}{BW} \tag{1.1}$$

วงจรที่มีค่า Q สูงมากเท่าใด แบนด์วิดก็จะยิ่งแคบเท่านั้น (เข้าใกล้วงจรในอุดมคติซึ่งต้องการเลือกความถี่ที่ผ่านวงจรกรองได้เพียงค่าเดียว)และเอาต์พุตที่มีขนาดสูงขึ้นด้วย

วงจรในรูปที่ 2.36 แสดงวงจรกรองความถี่เป็นช่วง ซึ่งใช้วงจรกรองความถี่สูงและวงจรความถี่ต่ำเข้าด้วยกัน(โดยที่วงจรใดจะมาก่อนกันก็ได้) R_1 และ C_2 คือ อุปกรณ์ในการกรองความถี่ต่ำ ส่วน C_1 และ R_2 ใช้ในวงจรกรองความถี่สูงและสามารถหาความถี่รีโซแนนท์ f_r จากสมการ



รูปที่ 2.36 วงจรกรองแถบความถี่ผ่าน

และหาค่า Q ได้จากสมการ

Centre Freuency (fr)

$$f_r = \frac{1}{2\pi C}\sqrt{\frac{R1+R2}{(R1.R2.R3)}}$$

$$R_p = \frac{R_1 R_2}{R_1 + R_2}$$

Input resistnce (R1)

$$R1 = Q/(2\pi GfC)$$

Attenuator resistance (R2)

$$R2 = Q/(2Q^2 - G)2\pi fC$$

Feedback resistance (R3)

$$R3 = Q/\pi fC$$

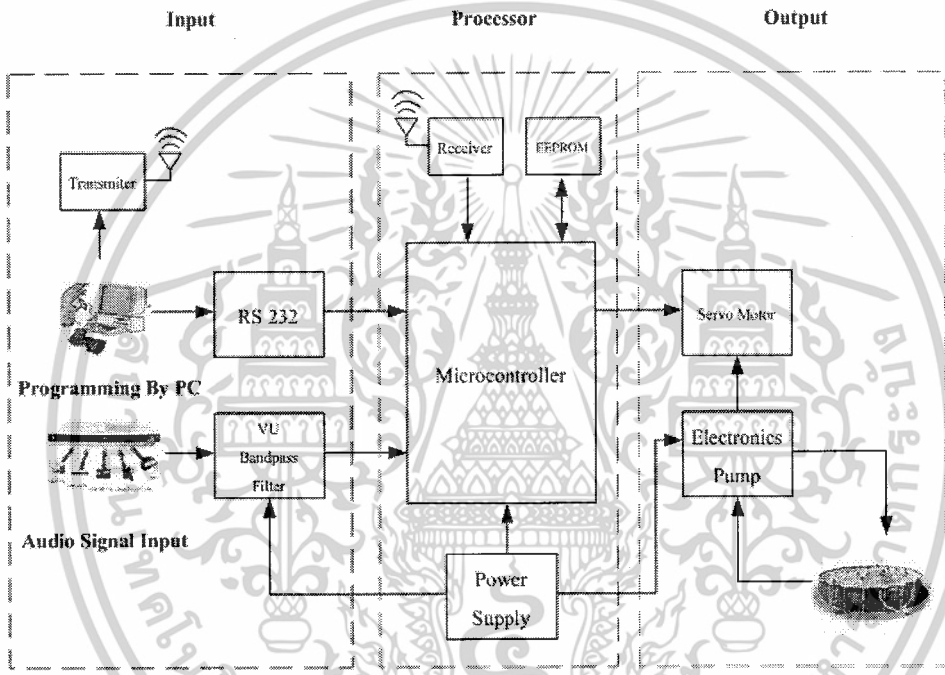
Passband gain (G)

$$G = 1/(2(R1/R2))$$

บทที่ 3
การออกแบบและการสร้าง

3.1 โครงสร้างการทำงานของน้ำพุ

โครงสร้างการทำงานของน้ำพุ สามารถแบ่งได้ 3 ส่วนใหญ่ ดังรูป



รูปที่ 3.1 แสดงโครงสร้างการทำงานของน้ำพุดนตรี

จากรูปที่ 3.1 จะเป็นแสดงการทำงาน โดยการป้อนอินพุทจากคอมพิวเตอร์และสัญญาณจากเสียงดนตรี โดยการป้อนอินพุทจากคอมพิวเตอร์จะแบ่งออกได้สองลักษณะคือ ส่งข้อมูลผ่านพอร์ต RS232 และการส่งข้อมูลผ่านอุปกรณ์ Wireless และเมื่อทำการส่งข้อมูลไปยังไมโครคอนโทรลเลอร์จะทำการประมวลผลข้อมูลเพื่อส่งไปที่วงจรคอนโทรลเซอร์โวมอเตอร์

3.2 วงจร 5 แบนด์มิวสิกไลน์

เสียงดนตรี เป็นสัญญาณอิเล็กทรอนิกส์หนึ่งซึ่งประกอบด้วยสัญญาณหลายๆ ความถี่มารวมกันทำให้เกิดความไพเราะ แต่ถ้าต้องการความถี่ใดความถี่หนึ่งที่นำไปทำการควบคุมอุปกรณ์ไฟฟ้าที่มีขนาดกำลังวัตต์สูงหรือจะนำไปทำการตรวจวัดระดับสัญญาณของความถี่นั้น เราคงนึกถึงวงจรแบนพาสฟิลเตอร์หรือวงจรแยกความถี่ ตัวอย่างของงานลักษณะดังกล่าวได้แก่ วงจรควบคุมไฟเวที, วงจรแสดงระดับสัญญาณหรือที่เรียกว่า VU เราจึงขอเสนอวงจร 5 แบนด์มิวสิกไลน์ที่เป็นแยกความถี่สัญญาณเสียงดนตรี 5 ช่องความถี่ที่นำไปควบคุมอุปกรณ์ไฟฟ้าได้

3.2.1 ลักษณะของวงจรและการใช้งาน

วงจร 5 แบนด์มิวสิกไลน์ เป็นวงจรรับสัญญาณเสียงดนตรีของเครื่องเล่นเทปซีดีเพลง หรือ ดีวีดี ที่ส่งออกตัวเครื่องเล่นมาเข้าที่จุด J1 และ J2 ของวงจร เพื่อนำเสียงดนตรีไปแยกตามความถี่ที่กำหนดไว้ โดยแบ่งเป็น 5 ช่อง คือ 60Hz, 250Hz, 1kHz, 3.5kHz และ 10kHz เอาต์พุตแต่ละช่องจะต่อออกไปที่คอนเน็คเตอร์ J3 ซึ่งเป็นคอนเน็คเตอร์ 8 ขา เพื่อสะดวกในการนำไปต่อใช้ควบคุมไฟเวที ของ 5 ไตรแอดสวิตช์การ์ด หรือจะนำไปต่อกับวงจร 5 แบนด์วียูมิเตอร์ โดยวงจรได้ออกแบบให้มีการปรับเพิ่มอัตราขยายอัตโนมัติในกรณีที่สัญญาณเสียงดนตรีที่ส่งมามีขนาดเบา ทำให้สัญญาณควบคุมมีอัตราขยายที่คงที่ วงจรใช้แหล่งจ่ายไฟกระแสตรงขนาด 5 Vdc

3.2.2 การทำงานของวงจร

วงจร 5 แบนด์มิวสิกไลน์แบ่งการทำงานเป็น 3 ส่วนคือ ปรีแอมป์, ควบคุมอัตราขยายอัตโนมัติและแบนด์พาสฟิลเตอร์

เมื่อจ่ายไฟกระแสตรงขนาด 5 โวลท์ให้กับวงจร และนำสัญญาณเสียงดนตรีมาต่อเข้าที่ J1 สัญญาณเสียงจะผ่านเข้า R1, R7, R9, C2 เข้าขา 5 ของ U1B ซึ่งเป็นออฟแอมป์ขยายเสียง ต่อวงจรขยายแบบนอนอินเวอร์ติ้งแอมป์ โดยมีอัตราขยายประมาณ 48 เท่าโดยกำหนดจากอัตราขยายเท่า 1+(R3/R2) เอาต์พุตที่ขา 7 ของ U1B จะแยกเป็น 2 ทาง ทางแรกจะผ่าน C4 เข้าวงจรควบคุมอัตราขยายอัตโนมัติซึ่งประกอบด้วย R20, D2, D1, C5, R18, R19, Q1 และ Q2 ซึ่งวงจรจะควบคุมให้เอาต์พุตของวงจรปรีแอมป์ให้มีขนาดสัญญาณคงที่ ส่วนทางที่ 2 ผ่าน C6 ส่งเข้าวงจรแบนด์พาสฟิลเตอร์หรือวงจรขยายเฉพาะความถี่ โดยวงจรจะยอมให้เฉพาะความถี่ที่ตรงกับความถี่ที่กำหนดไว้ผ่านไปได้เท่านั้น ซึ่งวงจรกำหนดไว้ 5 ช่อง ซึ่งช่วงความถี่กลางของแต่ละวงจรจะมีความถี่ดังนี้

- ช่องที่ 1 ความถี่กลาง 60 Hz ประกอบด้วย R4, R5, C7, C8, R6, U1A และ C17 โดยเอาต์พุตจะต่อเข้าที่ขา 6 ของ J3

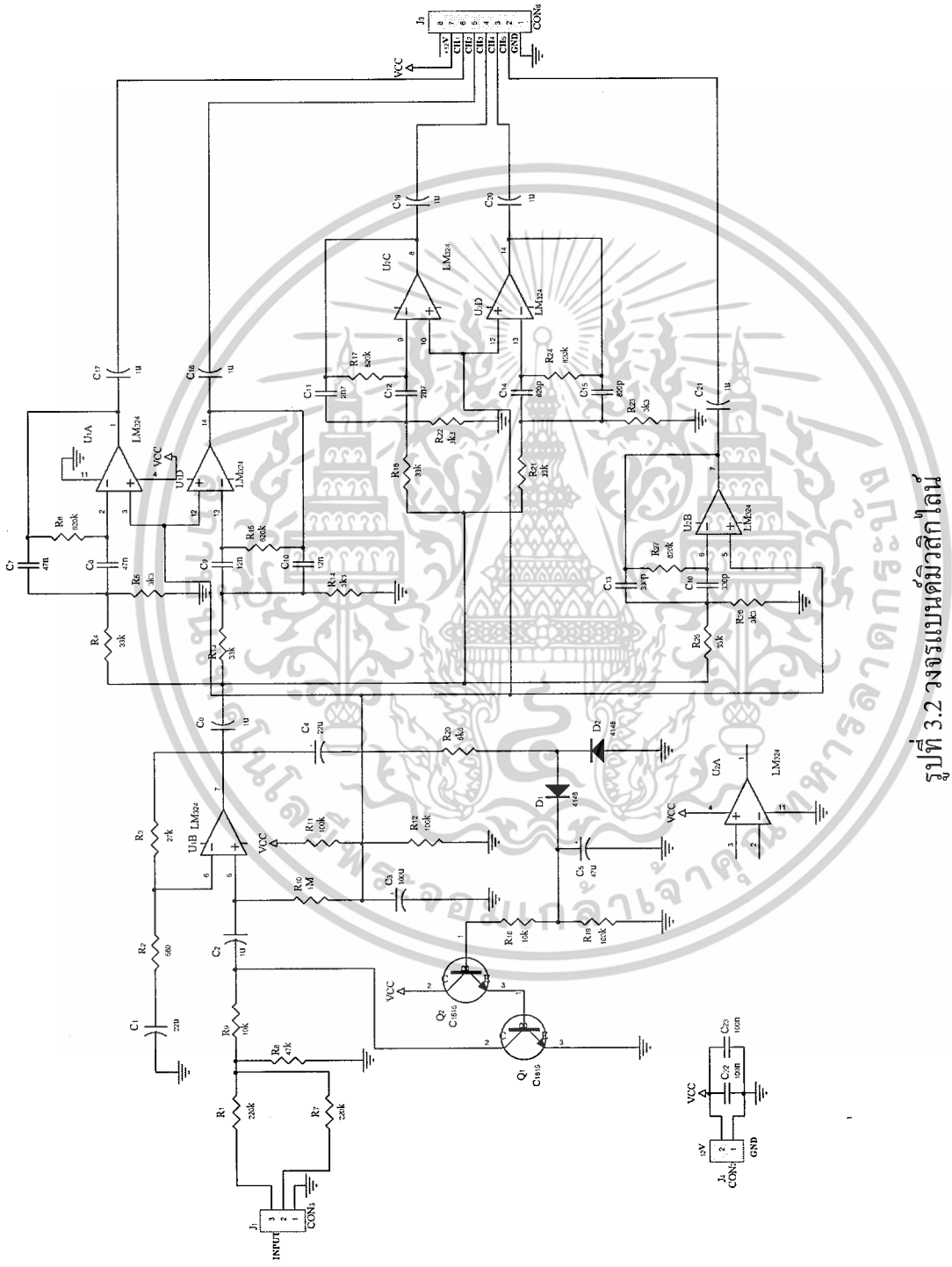
- ช่องที่ 2 ความถี่กลาง 250 Hz ประกอบด้วย R13, R14, C9, C10, R15, U1C และ C18 โดยเอาต์พุตจะต่อเข้าที่ขา 5 ของ J3

- ช่องที่ 3 ความถี่กลาง 1 kHz ประกอบด้วย R16, R17, C11, C12, R22, U2D และ C19 โดยเอาต์พุตจะต่อเข้าที่ขา 4 ของ J3

- ช่องที่ 4 ความถี่กลาง 3.5 kHz ประกอบด้วย R21, R23, C14, C15, R24, U2C และ C20 โดยเอาต์พุตจะต่อเข้าที่ขา 3 ของ J3

- ช่องที่ 5 ความถี่กลาง 10 kHz ประกอบด้วย R25, R27, C16, C13, R26, U2B และ C21 โดยเอาต์พุตจะต่อเข้าที่ขา 2 ของ J3

ส่วนภาคจ่ายไฟที่ใช้กับวงจรใช้ไฟจากวงจรเรกติไฟร์ขนาด 5 Vdc และแหล่งจ่ายไฟจะต่อเข้าที่ขา 7 ของ J3 ส่วนขากราวด์จะเข้าที่ขา 1 ของ J3 เพื่อความสะดวกในการต่อใช้งาน เมื่อออกสนามจริง อินพุตรับสัญญาณเสียงจึงใช้ RCA แบบยึดลงปริ๊นซ์ ส่วนเอาต์พุตออกแบบให้เป็นคอนเน็คเตอร์แบบตั้ง 8 ขา ซึ่งสามารถเชื่อมต่อกับวงจรควบคุมกำลังไฟฟ้า หรือวงจรแสดงตำแหน่งขาและชื่อสัญญาณไว้แล้ว การทดสอบและการต่อใช้งานก็คงจะต้องทำให้ไปด้วยกัน เพราะการทดสอบแบบง่ายก็คือ การทดลองต่อใช้งานจริง การต่อใช้งานก็เพียงแต่นำเอาต์พุตของวงจรไปเชื่อมเข้ากับวงจรไตรแอคสวิทช์การ์ด ในการควบคุมไฟเวทีใช้กระพริบตามความถี่ของเสียงดนตรี หรือจะใช้กับ 5 แบนด์วียูมิเตอร์ ใช้แสดงผลระดับสัญญาณตามความถี่ของเสียงดนตรี ซึ่งตำแหน่งขาของคอนเน็คเตอร์ได้ออกแบบให้ตรงกันทั้ง 8 ขา สามารถใช้งานง่ายเหมือนกับการต่อสายแพในเครื่องคอมพิวเตอร์



รูปที่ 3.2 วงจรแบนด์พาสฟิลเตอร์

3.3 วงจร 5 วียูมิเตอร์

วียูมิเตอร์เป็นวงจรใช้วัดระดับสัญญาณเสียงดนตรีที่ติดตามเครื่องเสียงไม่ว่าจะเป็นเพาเวอร์แอมป์ มิกเซอร์ หรือพวกเครื่องมือวัดที่มีจุดประสงค์เพื่อบอกระดับความแรงของสัญญาณเสียง นอกจากนี้วียูมิเตอร์ยังทำให้เกิดความสวยงาม โดยทั่วไปวียูที่ใช้ในเครื่องเสียงจะแสดงบอกระดับความดังของการขยายเสียง ในเครื่องขยายเสียงที่เป็นระบบโมโนจะมีวียูเพียง 1 วียู คือมีวงจรวียูแสดงผล 5 ช่องแบ่งตามความถี่ของสัญญาณเสียงดนตรี สามารถนำไปใช้ติดตั้งในเครื่องเสียงเพื่อแสดงระดับของสัญญาณตามความถี่ หรือสามารถไปประยุกต์ ใช้งานเป็นเครื่องทดสอบระดับความถี่ของสัญญาณ(สเปกตรัมขนาดเล็กๆ) สามารถนำไปประยุกต์ใช้งานได้มากมาย

3.3.1 ลักษณะของวงจรและการใช้งาน

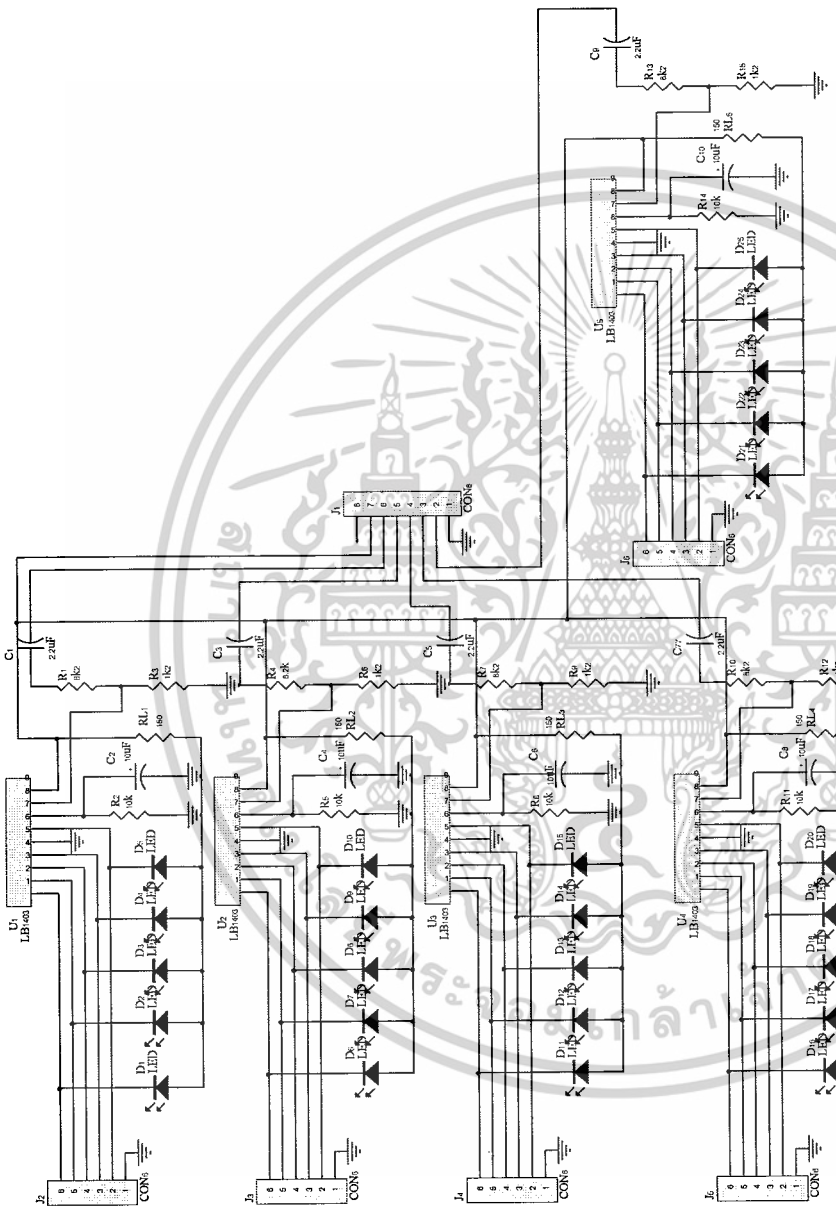
วงจร 5 วียูมิเตอร์ใช้วัดระดับสัญญาณเสียงแสดงผลแบบ Bar-Graph LED 5 ระดับ คือ -10dB, -5dB, 0dB, +3dB และ +6dB แบ่งเป็น 5 ช่องตามความถี่ของสัญญาณเสียงดนตรีคือช่องที่ 1 ความถี่ 50 Hz, ความถี่ที่ 2 ความถี่ 250 Hz, ความถี่ที่ 3 ความถี่ 1 KHz, ความถี่ที่ 4 ความถี่ 3.5 KHz และ ความถี่ที่ 5 ความถี่ 10 KHz การต่อใช้วงจร 5 วียูมิเตอร์จะใช้งานร่วมกับวงจร 5 แบนด์มิวสิกไลน์ ซึ่งทำหน้าที่แยกสัญญาณเสียงดนตรีที่เข้ามาที่เป็นความถี่ทั้งหมด 5 ความถี่เพื่อส่งต่อให้กับวงจร 5 วียูมิเตอร์เพื่อแสดงผลระดับความดังตามความถี่ทั้ง 5 ความถี่วงจรอาศัยไฟเลี้ยงวงจรจากวงจร 5 แบนด์มิวสิกไลน์ ผ่านการเชื่อมต่อของสายแพ 8 หรือ 16 ขาของ J1 จึงไม่ต่อไฟเลี้ยงให้กับวงจร

3.3.2 การทำงานของวงจร

หลักการสำคัญของวงจร 5 วียูมิเตอร์อยู่ที่ไอซี LB1403 ซึ่งทำหน้าที่ IC VU มิเตอร์ที่แสดงระดับสัญญาณได้ 5 ระดับด้วย LED ใช้วัดระดับสัญญาณเสียงแสดงผลแบบ Bar-Graph LED 5 ระดับตัว LB1403 ใช้กับไฟได้ตั้งแต่ 5-12 Vdc วงจร 5 วียูมิเตอร์จะประกอบด้วยวงจรที่ใช้ LB1403 ทั้งหมด 5 ตัว ซึ่งการทำงานของวงจรทั้ง 5 ตัวจะเหมือนกัน ต่างกันตรงที่ระดับของสัญญาณความถี่แต่ละช่องที่ส่งเข้ามาที่อินพุทของแต่ละช่องผ่าน J1

เริ่มต้นเมื่อนำวงจร 5 แบนด์มิวสิกไลด์มาต่อสายแพขนาด 8 หรือ 16 ขาเข้าที่จุด J1 ซึ่งคอนเน็คเตอร์ขา 8 ขา ซึ่งขาที่ 1 GND, ขาที่ 2 อินพุทของความถี่ที่ 5 10 KHz, ขาที่ 3 อินพุทของความถี่ที่ 4 3.5 KHz, ขาที่ 4 อินพุทของความถี่ที่ 3 1 KHz, ขาที่ 5 อินพุทของความถี่ที่ 2 250 Hz, ขาที่ 6 อินพุทของความถี่ที่ 1 50 Hz และ ขาที่ 7 ไฟเลี้ยงวงจร เมื่อต่อวงจรทั้งสองเข้าด้วยกันไฟเลี้ยงจะไหลผ่านเข้า J1 ไปเลี้ยงวงจร เมื่อต่อวงจรทั้งสองเข้าด้วยกันไฟเลี้ยงจะไหลผ่านเข้า J1 ไปเลี้ยงวงจร

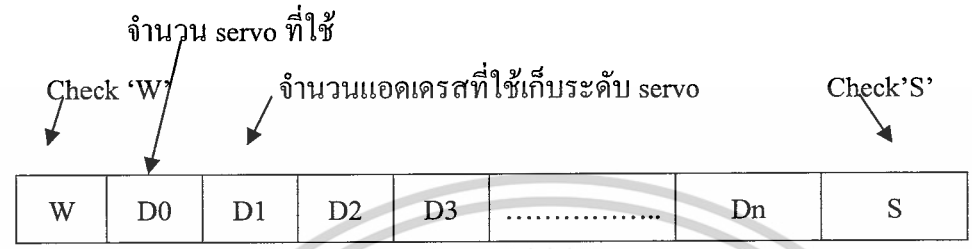
5 วียูมิเตอร์ สัญญาณเสียงความถี่ 5 10 KHz จะถูกต่อผ่านจากวงจร 5 แบนด์มิวสิกไลด์เข้าที่ J1 ขาที่ 6 ผ่าน C1 Coupling และลดทอนระดับสัญญาณด้วย R1, R3 แล้วจึงส่งให้ U1 ซึ่งเป็น IC VU สำเร็จรูปภายในประกอบด้วย IC ทำหน้าที่ขยายสัญญาณเข้าและทำการ Filter ด้วย R2, C2 ก่อนป้อนให้ IC Comparator 5 ระดับคือ +60dB,+3dB, 0dB, -5dB, -10dB ไปเปรียบเทียบกับค่าแรงดันอ้างอิงภายใน 85 m Vrms ที่ 0dB แล้วส่งผลที่วัดได้ไปเข้าวงจรขับกระแสผ่าน RL1 จำกัดกระแสให้ D1-D5 ติดสว่าง ส่วนการทำงานในความถี่ที่เหลือคือ 3.5 KHz, 1 KHz, 250 Hz และ 50 Hz จะมีการทำงานลักษณะเดียวกัน



รูปที่ 3.3 วงจรคอมพิวเตอร์

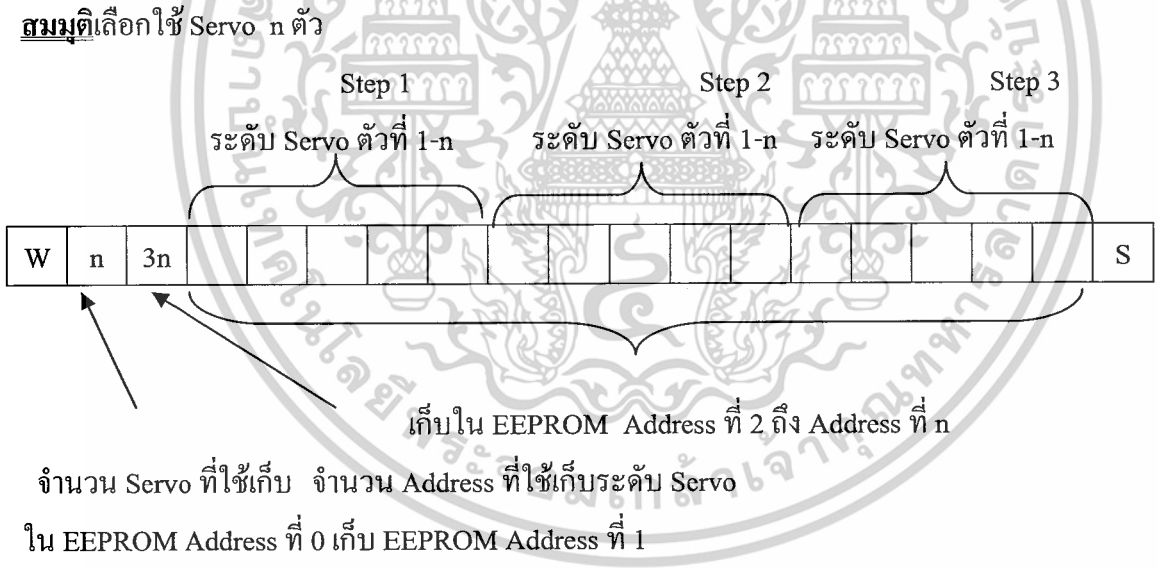
3.4 การทำงานในส่วนของ PIC16F275

การรับข้อมูล RS232 จาก C-Builder



หมายเหตุ D2-Dn เป็นระดับของ Servo

รูปที่ 3.4 การส่งข้อมูลจากคอมพิวเตอร์



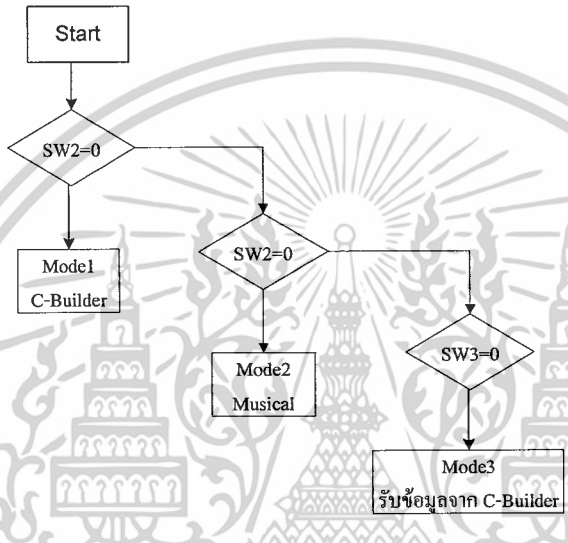
รูปที่ 3.5 การเลือกใช้ Servo

การทำงานของวงจร

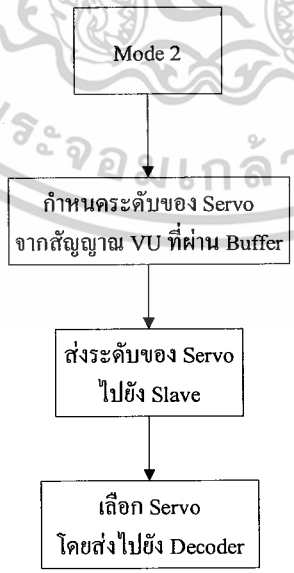
สัญญาณของวียูมิเตอร์ 5 Chanel จะถูกส่งมายัง IC 741LS245 เพื่อทำการเลือก การทำงานของสัญญาณแต่ละ Chanel โดยใช้ PIC 16F887 ในการควบคุม IC 74LS245 แต่ละตัวเพื่อเลือกสัญญาณ แต่ละ Chanel จากนั้นนำสัญญาณที่ได้มาทำการกำหนดระดับของ Servo ด้วย PIC

16F887 แล้วส่งระดับที่กำหนดไปยัง PIC 12F675 แต่ละตัว จากนั้น PIC 16F887 ก็จะทำการเลือก Servo โดยใช้ IC Decoder 74HC154 แล้วส่งไปยัง PIC 12F675 เพื่อให้ Servo ตัวนั้นทำงาน

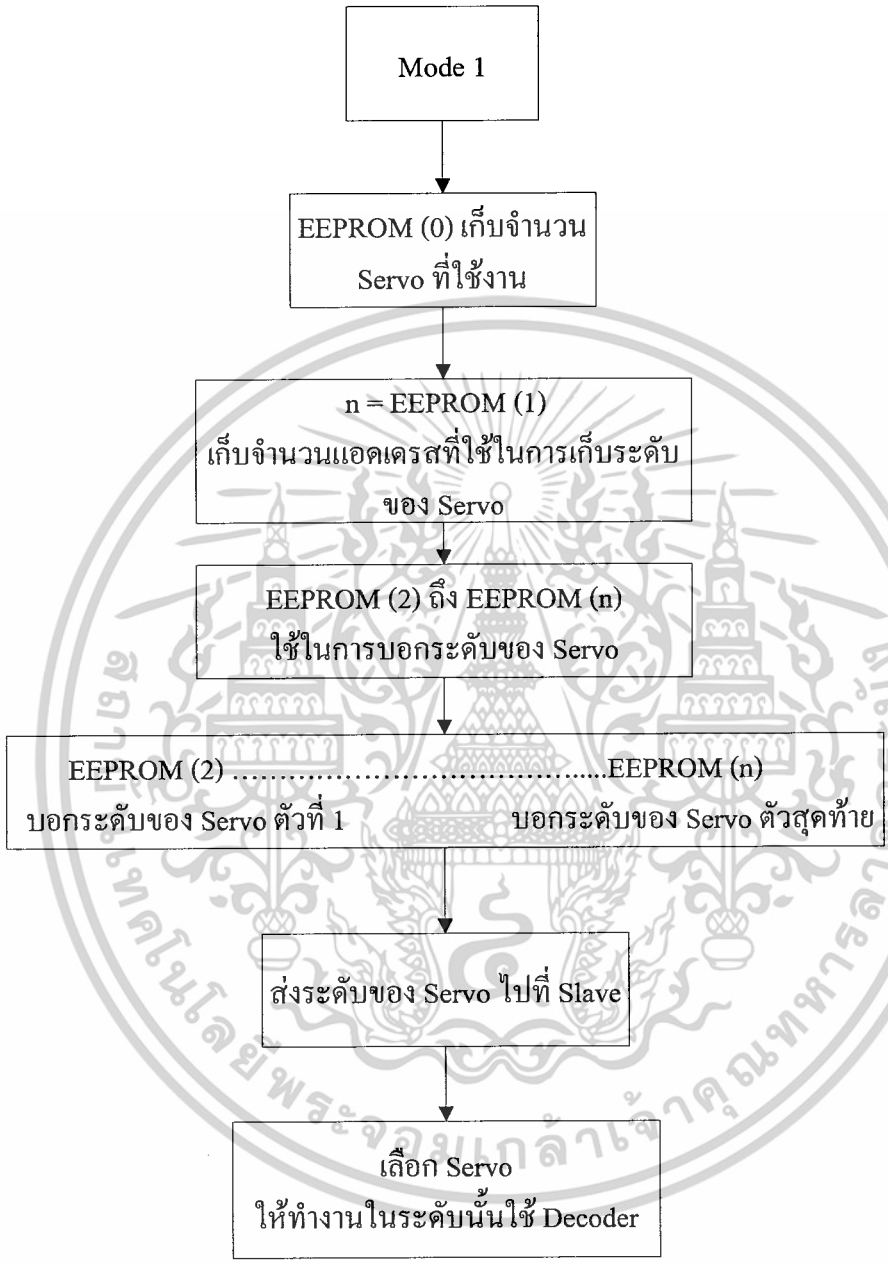
3.4.1 สวิทช์เลือกโหมดการทำงานของ Servo



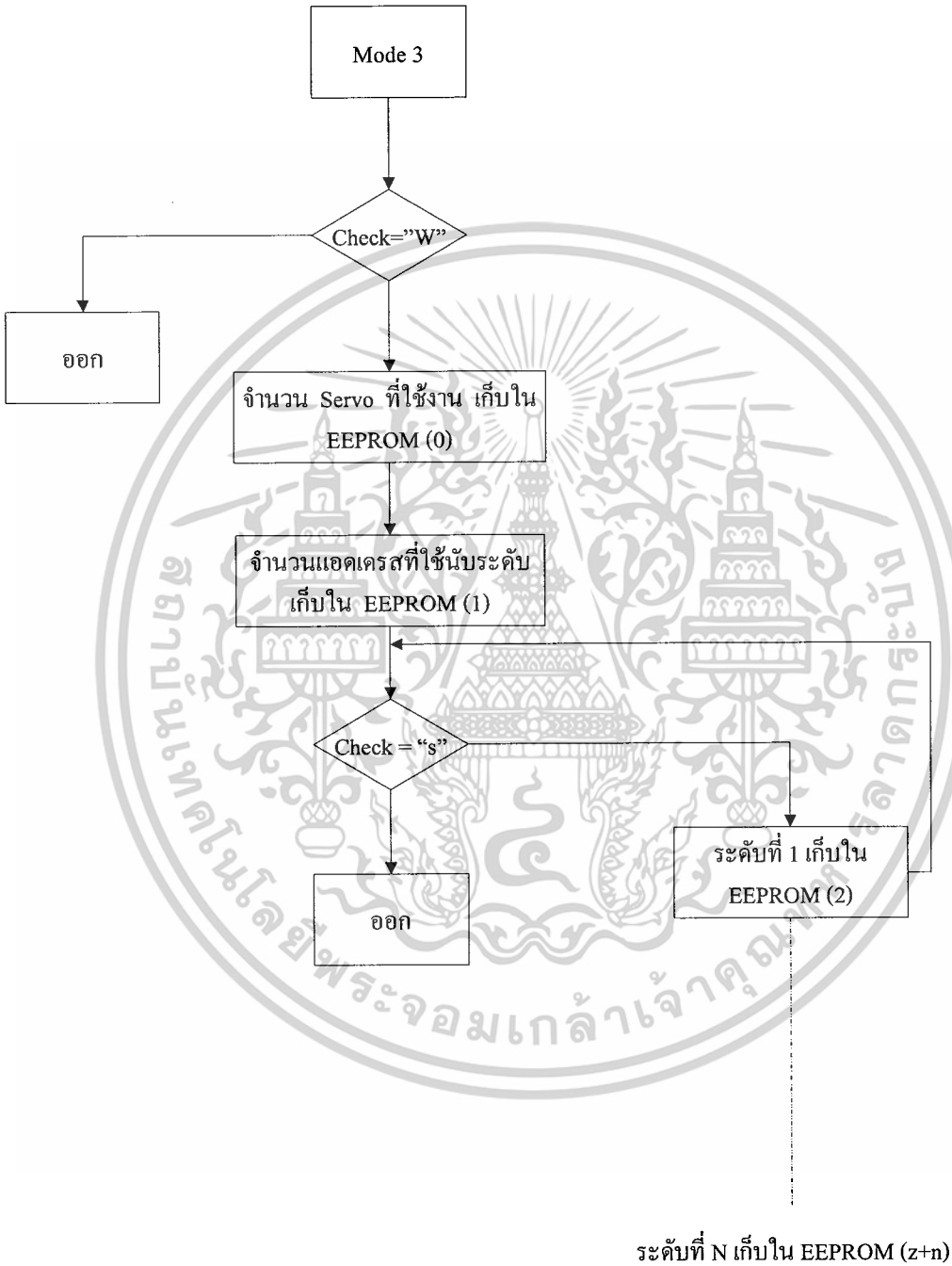
รูปที่ 3.6 การเลือกโหมดการทำงาน



รูปที่ 3.7 การทำงานของโหมดที่ 2 Musical

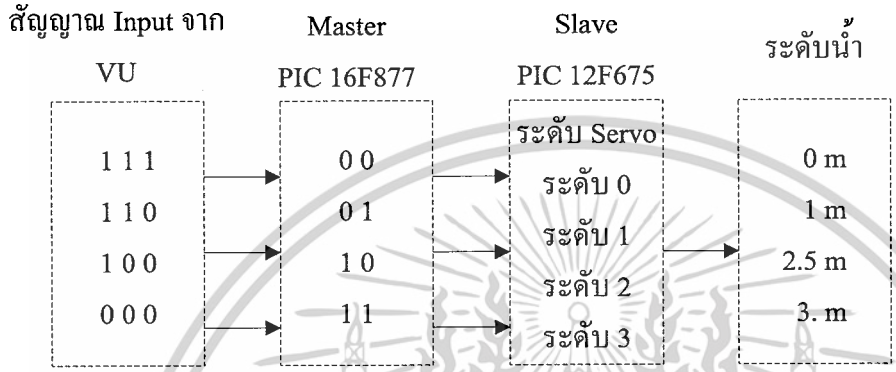


รูปที่ 3.8 การทำงานของโหมดที่ 1 C++ Builder



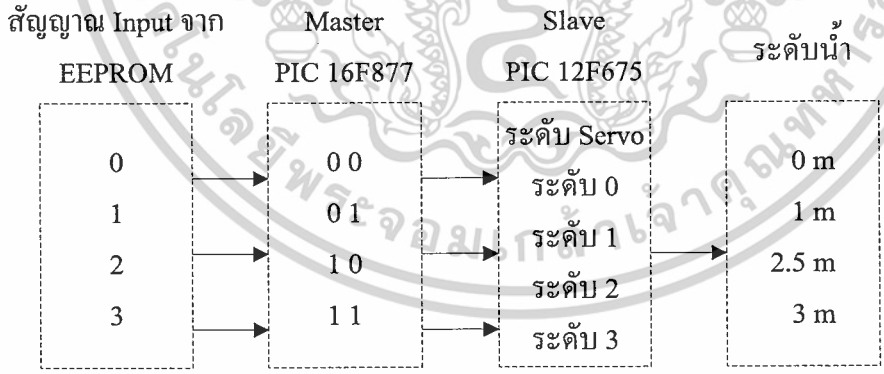
รูปที่ 3.9 การทำงานของโหมดที่ 3

3.4.2 การกำหนดระดับ Servo จาก วียูมิเตอร์



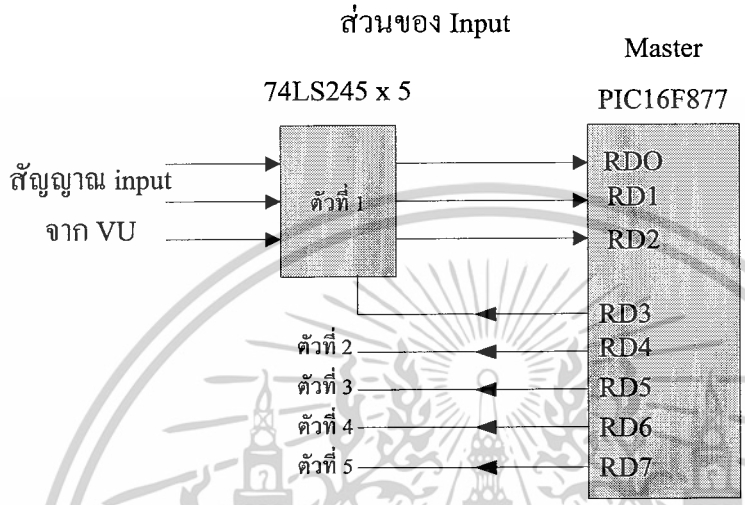
รูปที่ 3.10 การกำหนดระดับ Servo จากวียูมิเตอร์

3.4.3 การกำหนดระดับ Servo จาก EEPROM

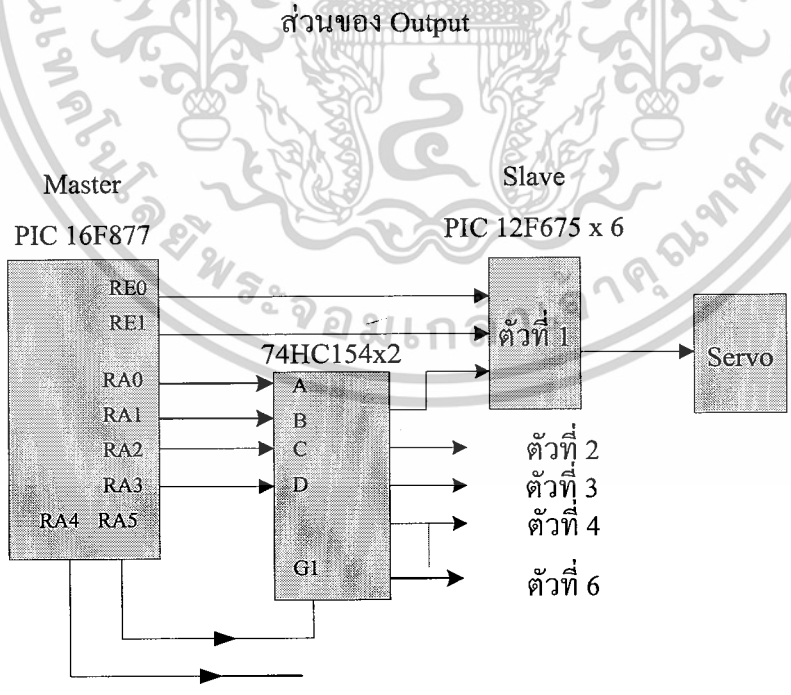


รูปที่ 3.11 การกำหนดระดับ Servo จาก EEPROM

3.4.4 ส่วนของ Input และ Output

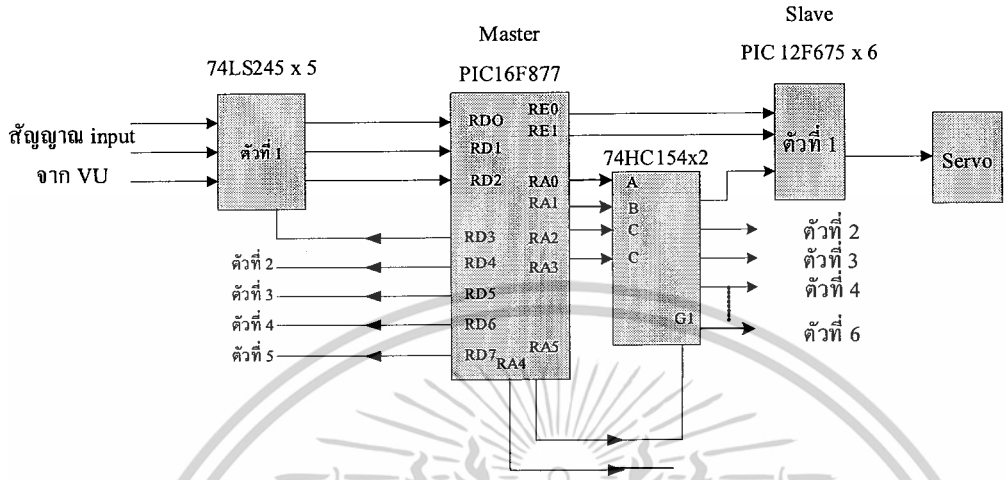


รูปที่ 3.12 การทำงานในส่วนของ Input

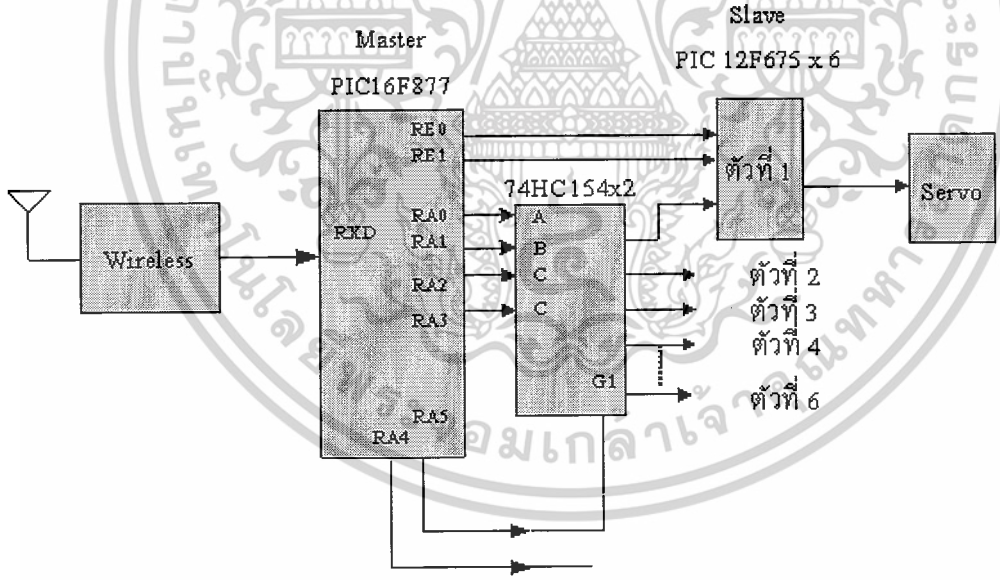


รูปที่ 3.13 การทำงานส่วนของ Output

3.4.5 การทำงานในส่วนของไมโครคอนโทรเลอร์



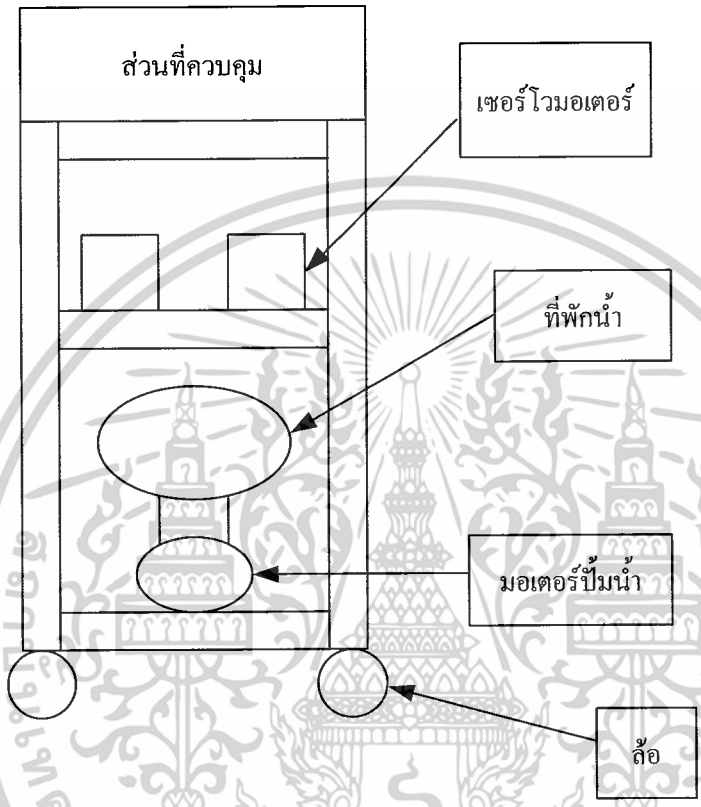
รูปที่ 3.14 การทำงานของส่วนไมโครคอนโทรเลอร์



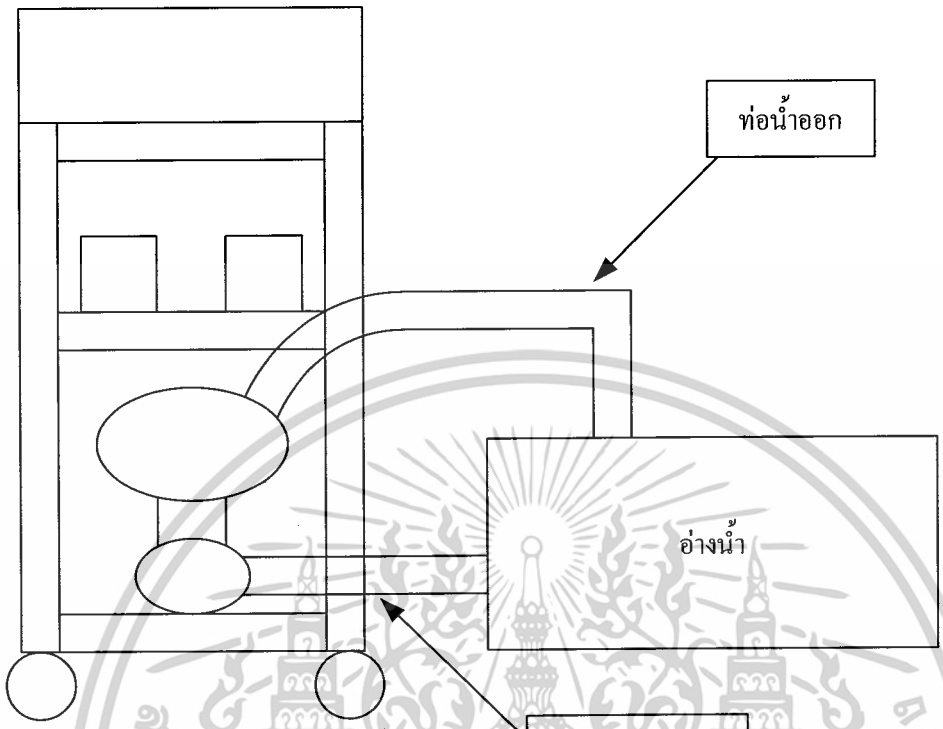
รูปที่ 3.15 การทำงานของส่วนไมโครคอนโทรเลอร์ร่วมกับ Wireless

3.5 การออกแบบน้ำพุและโครงสร้าง

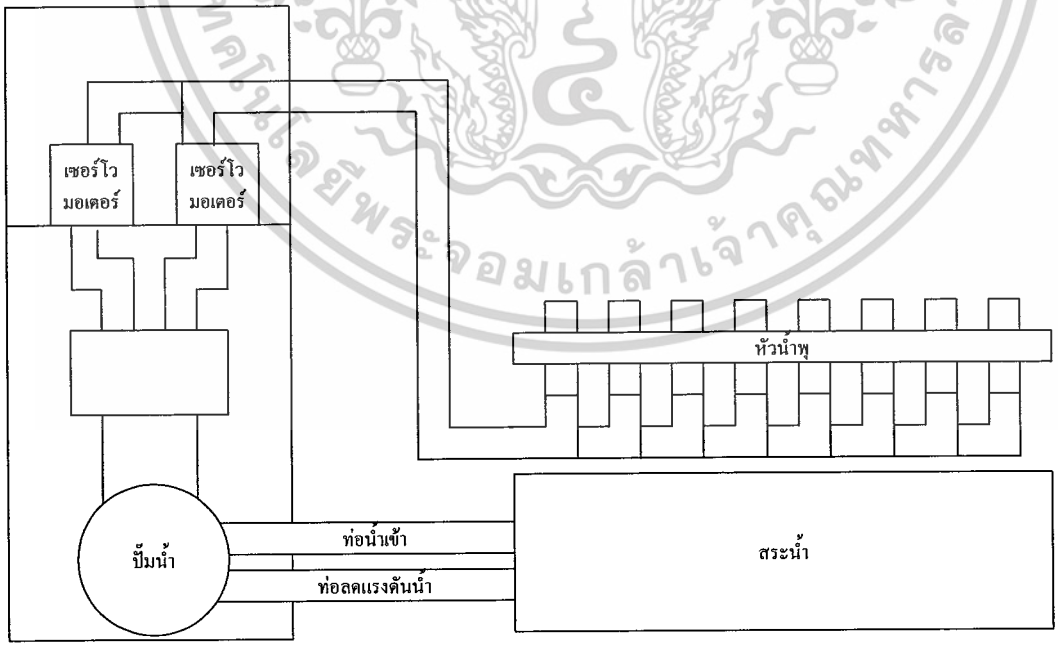
โครงสร้างของตัวแท่นน้ำพุ



รูปที่ 3.16 มุมมองด้านหน้า



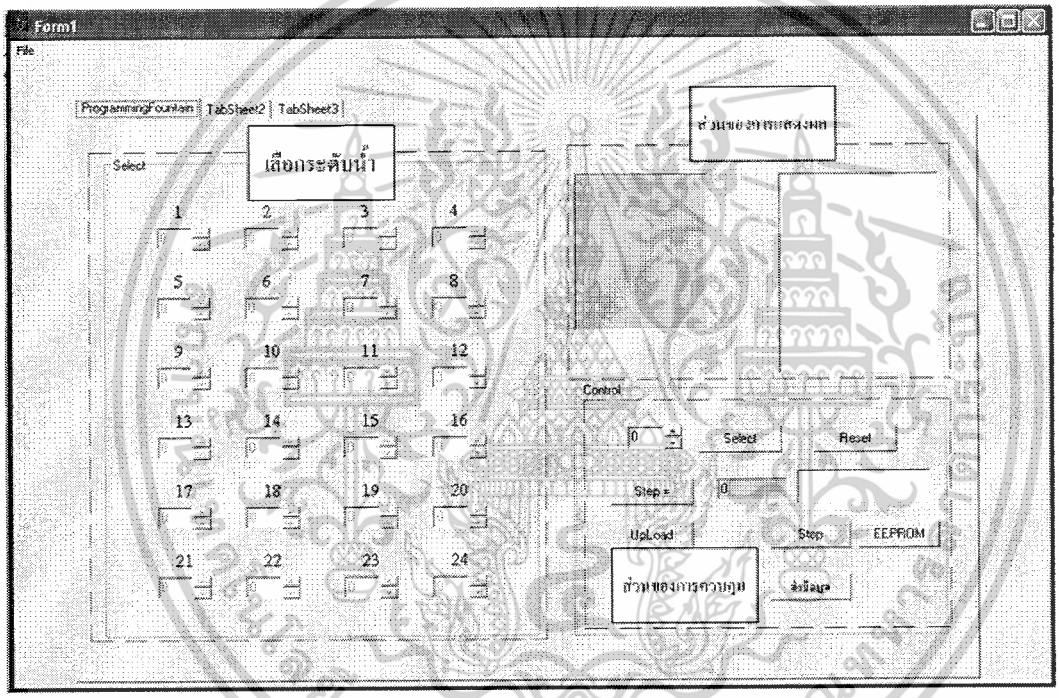
รูปที่ 3.17 การส่งน้ำ



รูปที่ 3.18 การเปิด-ปิดของเซอร์โวมอเตอร์เพื่อส่งน้ำไปยังหัวน้ำพุ

การทำงานเมื่อเปิดปั้มน้ำเข้าปั้ม จะอัดแรงดันน้ำขึ้นไปที่พักน้ำจะมีท่อระบายน้ำจากที่พักน้ำมายังอ่างน้ำ เพื่อที่จะระบายน้ำเมื่อวาล์วทุกวาล์วปิดหมดแรงดันน้ำจากปั้มจะส่งไปที่วาล์วและเซอร์โวมอเตอร์จะควบคุมการเปิด-ปิดวาล์วน้ำ ที่จะออกทางหัวน้ำพุมีอยู่ 4 ระดับ คือ ระดับ 1 คือ ปิดวาล์ว, ระดับ 2 คือเปิดวาล์ว 25%, ระดับ 3 คือ เปิดวาล์ว 75%, ระดับ 4 คือ เปิดวาล์ว 100%

3.6 การทำงานของโปรแกรม C++ Builder



รูปที่ 3.19 โปรแกรมควบคุมจากคอมพิวเตอร์

การทำงานของโปรแกรมมีทั้งหมด 3 ส่วน ได้แก่ 1) ส่วนของการควบคุมหรือ Control มีหน้าที่ในการกำหนดเลือกหัวของน้ำพุที่จะใช้งานแล้วเป็นตัวที่ส่งข้อมูลไปให้แก่ไมโครคอนโทรเลอร์ 2) ส่วนของการเลือกระดับของน้ำพุ หรือ Select เป็นส่วนที่รับคำสั่งจากส่วนของการควบคุม โดยจะมีระดับในการปล่อยน้ำด้วยกัน 4 ระดับ และในส่วนสุดท้าย 3) เป็นส่วนของการแสดงผลข้อมูลที่ส่งออกไปให้แก่ไมโครคอนโทรเลอร์

3.7 การทำงานในส่วนของบอร์ดคอนโทรลเลอร์

ประกอบด้วย IC BUFFER 74LS245, IC MASTER PIC 16F877, IC DECODER 74HC154, IC SLAVE PIC 12F675, IC MAX 232 และ SERVO MOTOR FUTABA S3003

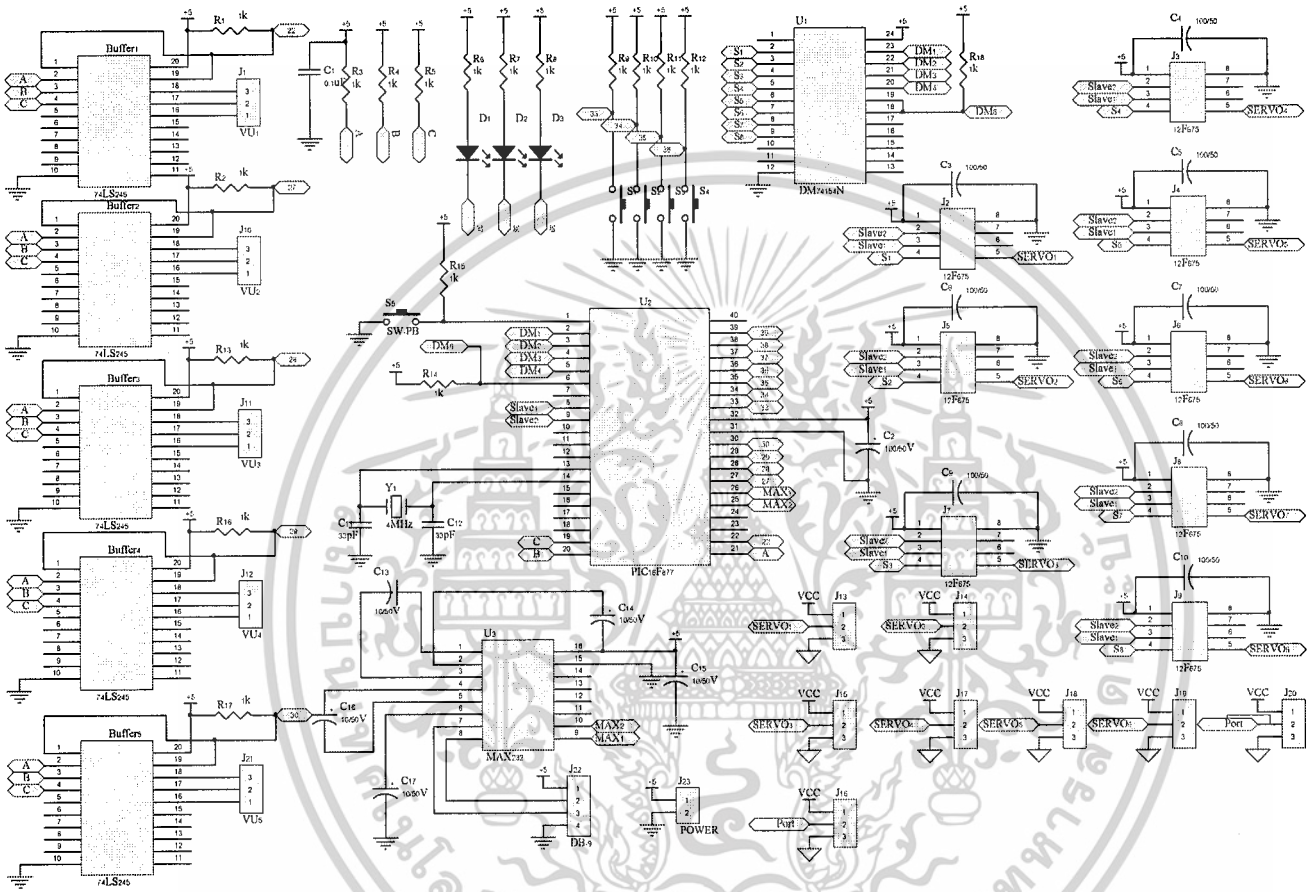
การทำงานของ S1 เป็นสวิตซ์ที่ใช้ในการหยุดการทำงาน S2 เป็นสวิตซ์ที่เลือกใช้สัญญาณข้อมูลจาก EEPROM S3 เป็นสวิตซ์ที่เลือกใช้สัญญาณเสียงจาก AUDIO INPUT ภายนอก S4 เป็นสวิตซ์ที่ใช้ในการรอรับข้อมูลจากหน้าจอคอมพิวเตอร์ โดย ซี พลัส พลัส บิลเดอร์ มาเก็บไว้ที่ EEPROM ของ PIC 16F877

การรับค่าข้อมูลแบ่งออกเป็น 2 ส่วน คือ ส่วนของข้อมูลจากสัญญาณเสียง AUDIO และส่วนของการนำค่าข้อมูลจาก EEPROM ไปกำหนดการใช้งาน

ในส่วนของการรับค่าข้อมูลจากสัญญาณเสียง AUDIO เป็นการนำสัญญาณเสียงป้อนเข้าสู่ AUDIO INPUTผ่านวงจรแอมพลิฟายน์เออร์และวงจรแยกช่องสัญญาณออกเป็น 5 ช่องสัญญาณ สัญญาณที่ได้จาก วียูมิเตอร์จะถูกส่งผ่านไปยัง IC BUFFER 74LS245 ซึ่งสัญญาณวียูมิเตอร์ จะมี 5 ช่อง การทำงานในส่วนของ IC BUFFER 74LS245 จะถูกควบคุมด้วย IC MASTER PIC 16F877 ซึ่ง IC MASTER PIC 16F877จะทำการกำหนดในการเลือกช่องสัญญาณทั้ง 5 ช่อง แล้วนำสัญญาณที่ได้ไปกำหนดระดับในการควบคุมเซอร์โวมอเตอร์ เพื่อที่จะส่งต่อไปยัง IC SLAVE PIC 12F675 แล้วใช้ IC DECODER 74HC154ในการเลือกการทำงานของ IC SLAVE PIC 12F675 เพื่อที่จะเลือกการใช้งานของเซอร์โวมอเตอร์

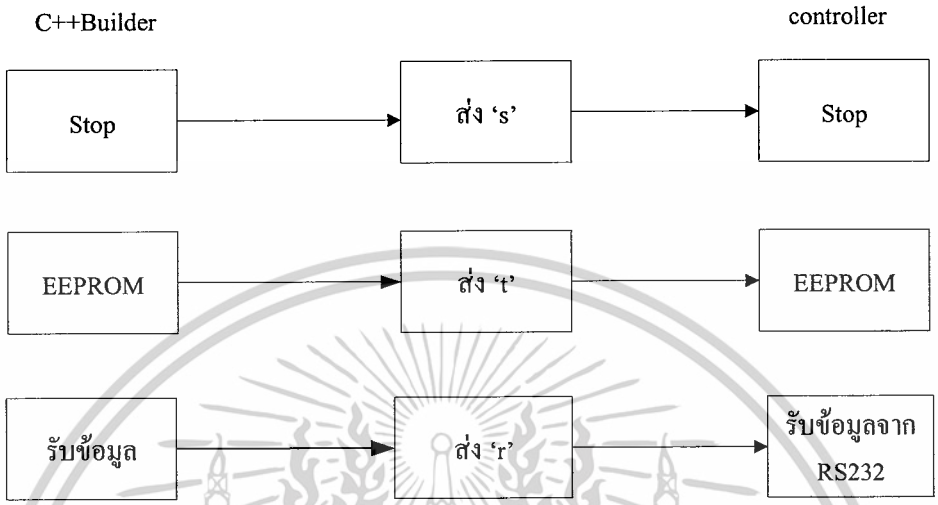
ในส่วนของการรับค่าข้อมูลจาก EEPROM ไปใช้งานนั้น IC MASTER PIC 16F877จะทำการนำข้อมูลที่เก็บไว้ใน EEPROM มาใช้ในการกำหนด ระดับในการควบคุมเซอร์โวมอเตอร์ เพื่อที่จะส่งต่อไปยัง IC SLAVE PIC 12F675 แล้วใช้ IC DECODER 74HC154ในการเลือกการทำงานของ IC SLAVE PIC 12F675 เพื่อที่จะเลือกการใช้งานของเซอร์โวมอเตอร์

การทำงานในส่วนของ IC MAX 232 จะทำหน้าที่ในการแปลงสัญญาณข้อมูลเพื่อที่จะทำการรับส่งข้อมูลระหว่างบอร์คอนโทรลเลอร์กับเครื่องคอมพิวเตอร์



รูปที่ 3.20 วงจรในส่วนของการควบคุมโดยรับข้อมูลจาก EEPROM และสัญญาณเสียงจากภายนอก

3.8 การทำงานในส่วนของ Wireless



รูปที่ 3.21 การเลือกคำสั่งการใช้งาน

ในส่วนของการส่งข้อมูลผ่านตัว Wireless โดยทำการเลือกผ่านโปรแกรม C++ Builder และประกอบไปด้วย ตัวอักษร 's' เป็นตัวอักษรที่ทำหน้าที่ระบบหยุดการทำงาน ตัวอักษร 't' เป็นตัวอักษรที่ทำหน้าที่ดึงข้อมูลจากไมโครคอนโทรเลอร์มาใช้งานใน EEPROM ตัวอักษร 'r' เป็นตัวอักษรที่ทำหน้าตรวจรับเข้ามาของข้อมูลจากไมโครคอนโทรเลอร์

3.9 การทำงานในบอร์ดคอนโทรลเลอร์แบบไร้สาย

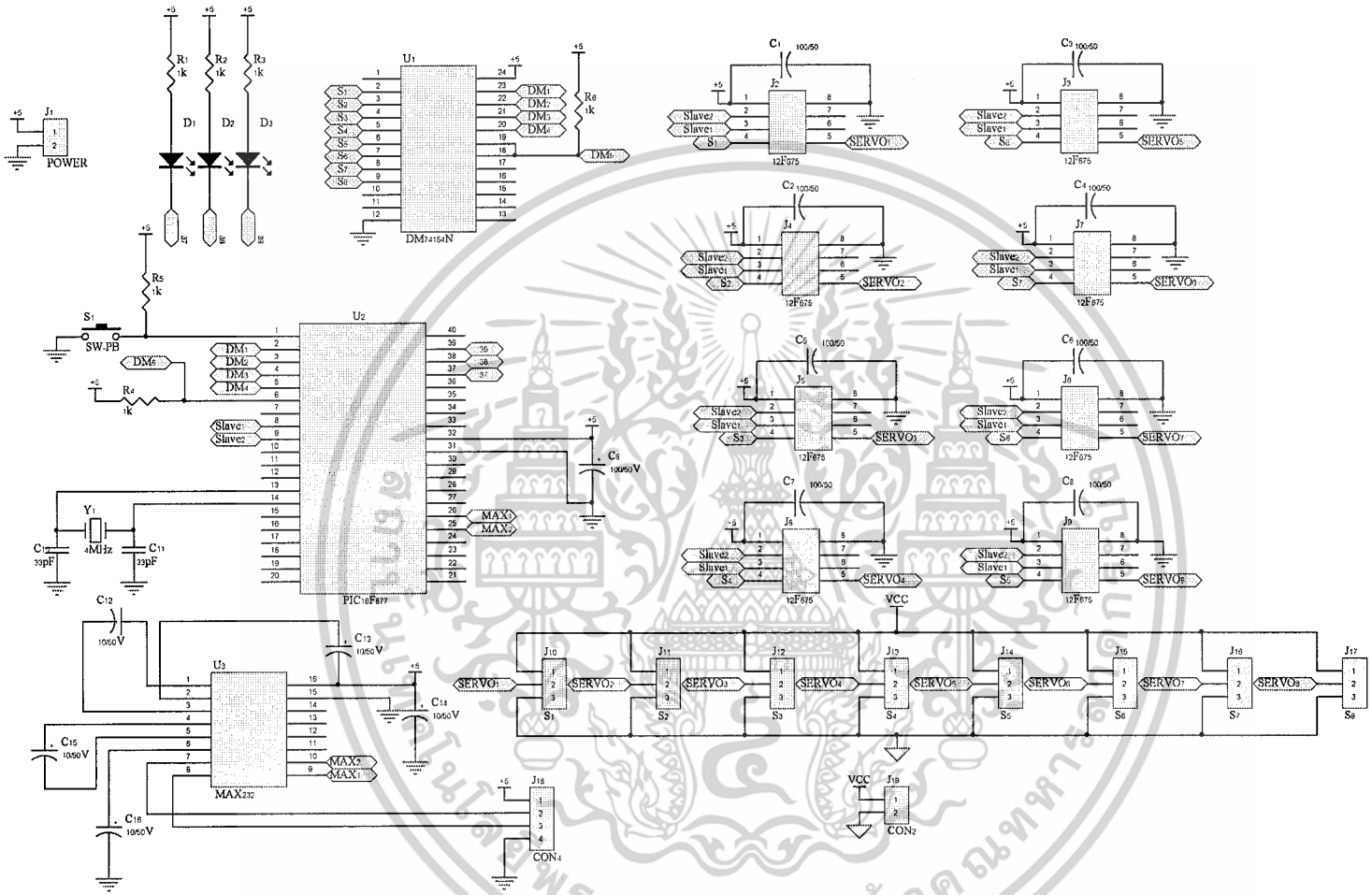
ประกอบด้วย IC MASTER PIC 16F877, IC DECODER 74HC154, IC SLAVE PIC 12F675, IC MAX 232และ SERVO MOTOR FUTABA S3003

การทำงานของบอร์ดคอนโทรลเลอร์แบบไร้สายจะใช้การควบคุมผ่านทางหน้าจอคอมพิวเตอร์โดย ซี พลัส พลัส บิลเดอร์แบบไร้สาย ซึ่งปุ่ม STOP เป็นสวิตซ์ที่ใช้ในการหยุดการทำงาน ปุ่ม EEPROR เป็นสวิตซ์ที่เลือกใช้สัญญาณข้อมูลจาก EEPROM ปุ่ม ส่งข้อมูล เป็นสวิตซ์ที่ใช้ในการรอรับข้อมูลจากหน้าจอคอมพิวเตอร์ โดย ซี พลัส พลัส บิลเดอร์ มาเก็บไว้ที่ EEPROM ของ PIC 16F877

การรับค่าข้อมูลจะเป็นการรับค่าข้อมูลจาก EEPROM ไปกำหนดการใช้งาน

การรับค่าข้อมูลจาก EEPROM ไปใช้งานนั้น IC MASTER PIC 16F877จะทำการนำข้อมูลที่เก็บไว้ใน EEPROM มาใช้ในการกำหนด ระดับในการควบคุมเซอร์โวมอเตอร์ เพื่อที่จะส่งต่อไปยัง IC SLAVE PIC 12F675 แล้วใช้ IC DECODER 74HC154ในการเลือกการทำงานของ IC SLAVE PIC 12F675 เพื่อที่จะเลือกการใช้งานของเซอร์โวมอเตอร์

การทำงานในส่วนของ IC MAX 232 จะทำหน้าที่ในการแปลงสัญญาณข้อมูลเพื่อที่จะทำการรับส่งข้อมูลระหว่างบอร์คอนโทรลเลอร์กับเครื่องคอมพิวเตอร์



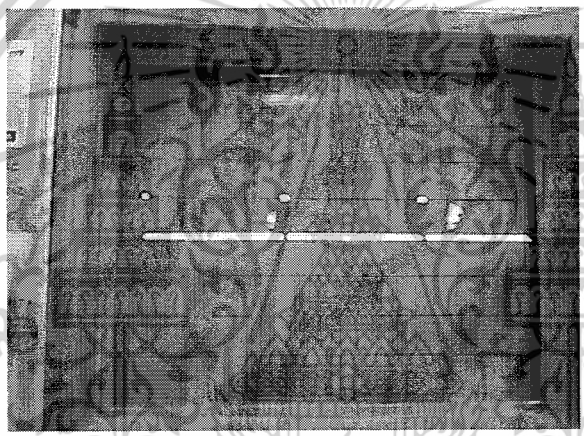
รูปที่ 3.22 วงจรในส่วนของการควบคุมโดยผ่านการสื่อสารทาง Wireless

บทที่ 4

ผลการทดลอง

4.1 สัญญาณควบคุมองศาของเซอร์โวมอเตอร์

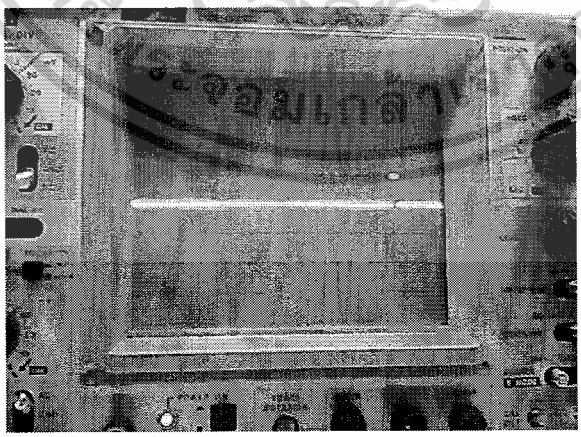
จากการทดลองในการควบคุมให้เซอร์โวมอเตอร์หมุนซ้ายหรือขวาเป็นการควบคุมช่วงของพัสส์ซีกบวกโดยเริ่มต้นที่ 0 องศง โดยจะต้องสร้างสัญญาณที่มีช่วงของพัลส์บวกขนาด 0.5 mS และเมื่อเพิ่มความกว้างของพัลส์ไปเรื่อยๆ จนถึง 1.5 mS เซอร์โวมอเตอร์ก็จะหมุนไปที่ตำแหน่ง 90 องคา พอดี



Time/Div = 5mS

Volt/Div = 5V

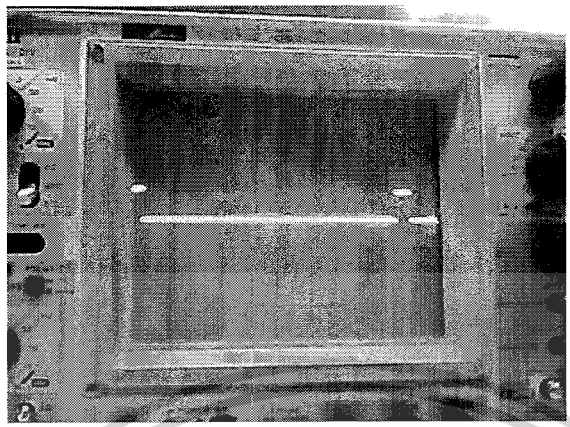
รูปที่ 4.1 สัญญาณที่มีความกว้างของช่วงบวก 0.5 mS



Time/Div = 2mS

Volt/Div = 5V

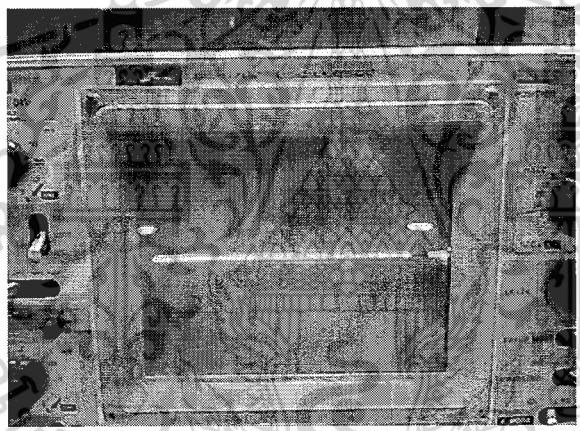
รูปที่ 4.2 สัญญาณที่มีความกว้างของช่วงบวก 1.5 mS



Time/Div = 2mS

Volt/Div = 5V

รูปที่ 4.3 สัญญาณที่มีความกว้างของช่วงบวก 1.1 mS



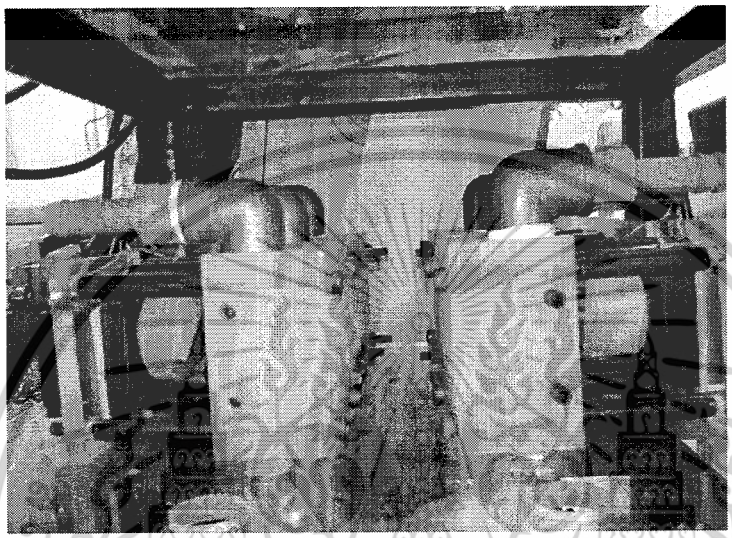
Time/Div = 2mS

Volt/Div = 5V

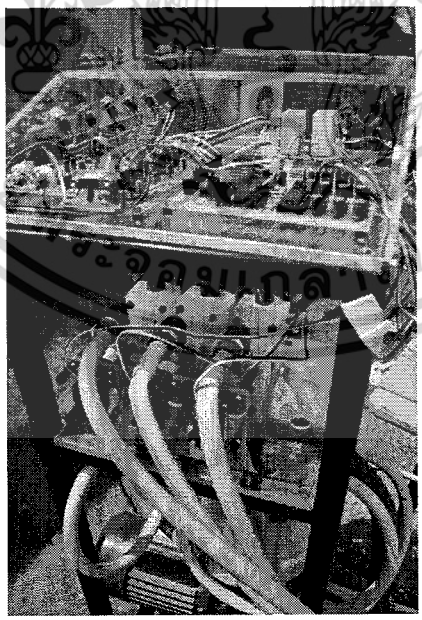
รูปที่ 4.4 สัญญาณที่มีความกว้างของช่วงบวก 1.5 mS

4.2 การทดลองระดับของน้ำที่เกิดจากการทำงานของ Servo

จากการทดลองระดับของน้ำจากปั้มน้ำ โดยระบบส่วนของตัวควบคุมการงานยังสามารถปรับระดับของความสูงขอน้ำพุได้ด้วยการปรับระดับการเปิดปิดวาล์วน้ำ



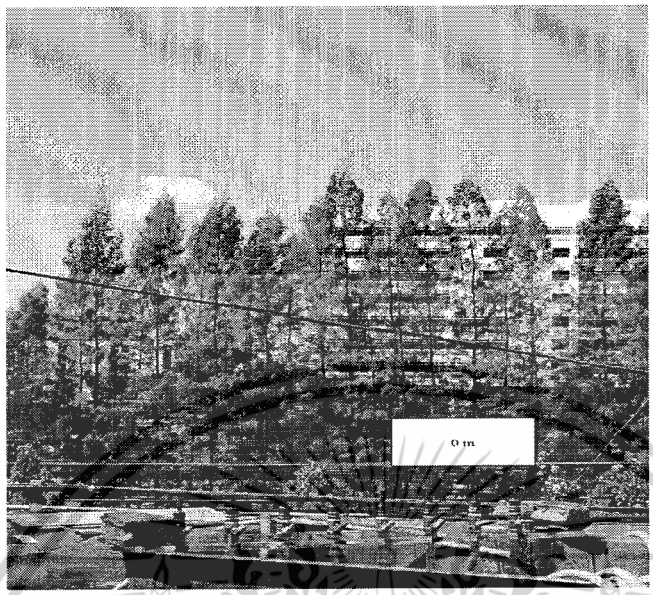
รูปที่ 4.5 ชุด Servo ควบคุมน้ำพุ



รูปที่ 4.6 การต่อชุดควบคุมกับ Servo

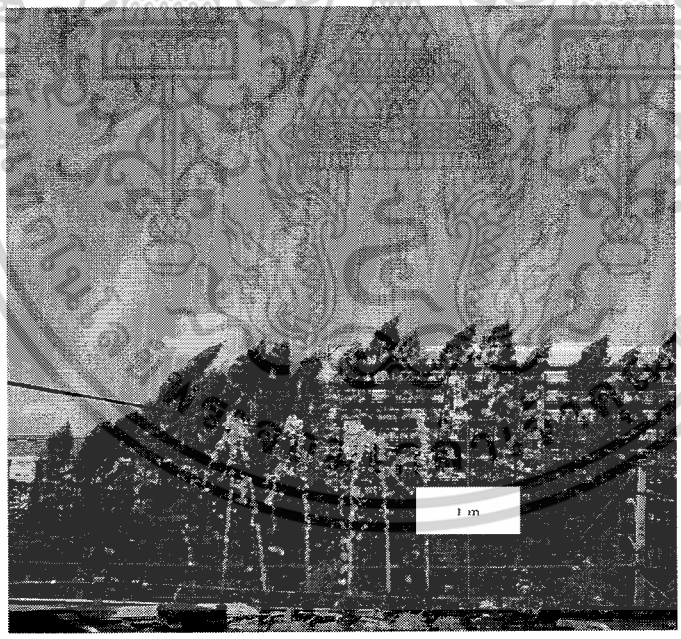
รูปที่ 4.7 การต่อชุดควบคุม Servo กับหัวน้ำพุ

Step ที่ 0

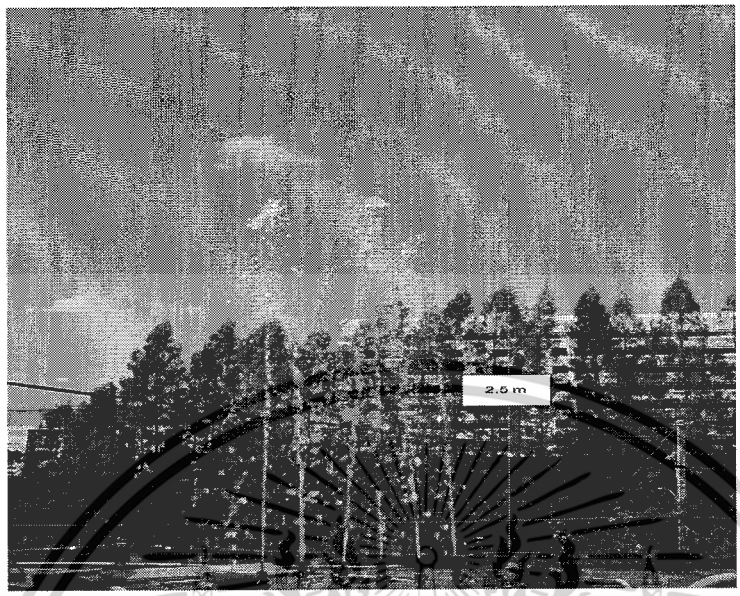


รูปที่ 4.8 การทำงานใน Step ที่ 0

Step ที่ 1

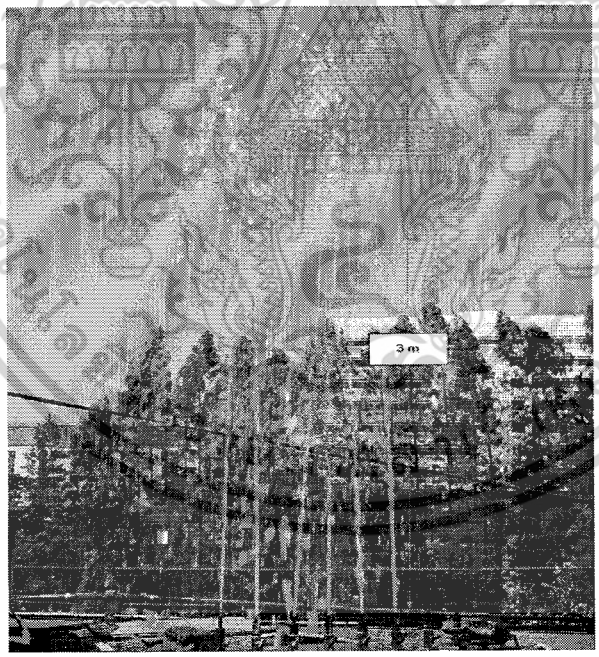


รูปที่ 4.9 การทำงานใน Step ที่ 1



Step ที่ 2

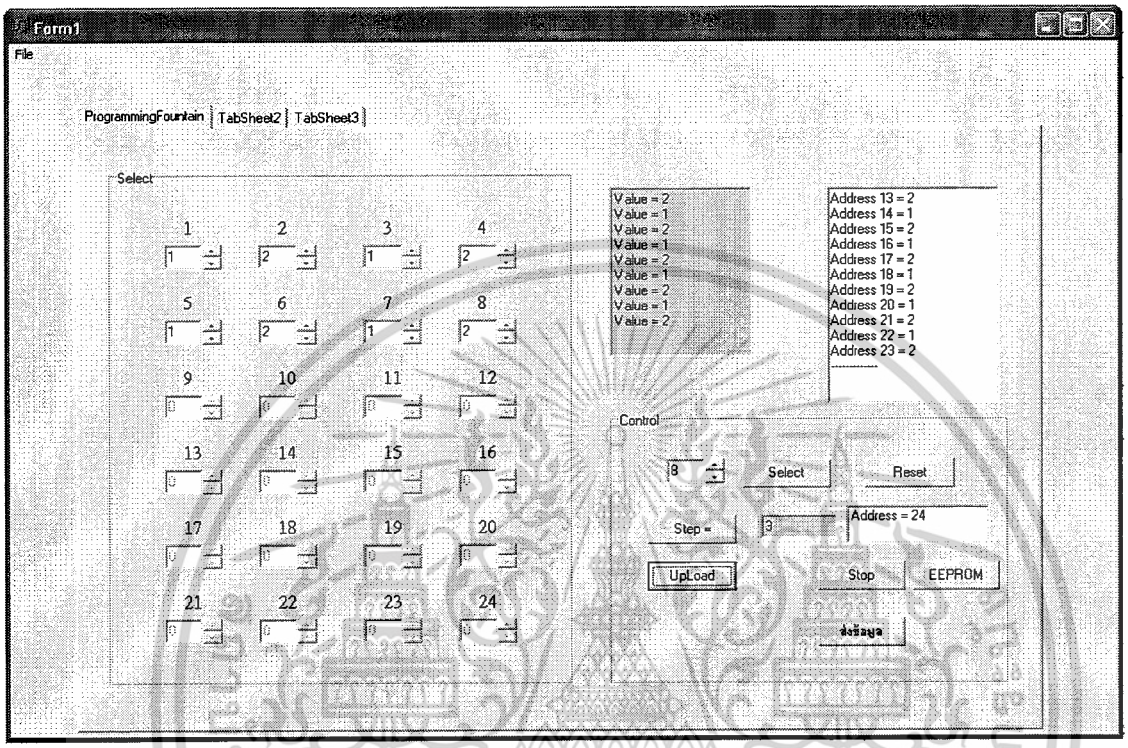
รูปที่ 4.10 การทำงานใน Step ที่ 2



Step ที่ 3

รูปที่ 4.11 การทำงานใน Step ที่ 3

4.3 การส่งข้อมูลของโปรแกรม C++ Builder



รูปที่ 4.12 การส่งข้อมูล

รูปที่ 4.13 Servo ควบคุมหัวน้ำพุผ่านพอร์ตอนุกรม RS232

4.4 ผลการทดลองในส่วนของ Musical

จากผลการทดลองพบว่าในส่วนของวงจรมีการตอบสนองในช่องความถี่ทั้งหมด 5 ช่วงความถี่ด้วยกันซึ่งประกอบไปด้วย ความถี่ 60 Hz ความถี่ 250 Hz ความถี่ 1 kHz ความถี่ 3.5 kHz และ ความถี่ 10 kHz

ความถี่ 60Hz																		
ความถี่(Hz)	57.25	60	62.5	62.7	63	63.8	64	64.5	65	65.5	68	70.25	72.25	75	77.5	80	85	90
แรงดัน(V)	0.704	0.95	1.27	1.32	1.32	1.5	1.56	1.62	1.72	1.76	1.8	1.48	1.2	0.96	0.78	0.66	0.52	0.46

ตารางที่ 4.1 ความถี่ 60 Hz

ความถี่ 250Hz																		
ความถี่(Hz)	210	220	230	240	245	250	255	260	265	270	275	280	290	300	310	320	330	340
แรงดัน(V)	0.44	0.52	0.64	0.8	0.9	1.06	1.2	1.4	1.6	1.8	1.84	1.74	1.42	1.1	0.9	0.76	0.64	0.56

ตารางที่ 4.2 ความถี่ 250 Hz

ความถี่ 1KHz																		
ความถี่(Hz)	1050	1060	1070	1090	1120	1140	1160	1170	1180	1200	1221	1230	1250	1280	1310	1330	1350	1370
แรงดัน(V)	0.87	0.93	0.99	1.12	1.35	1.5	1.7	1.78	1.8	1.8	1.74	1.6	1.46	1.2	1.05	0.94	0.86	0.8

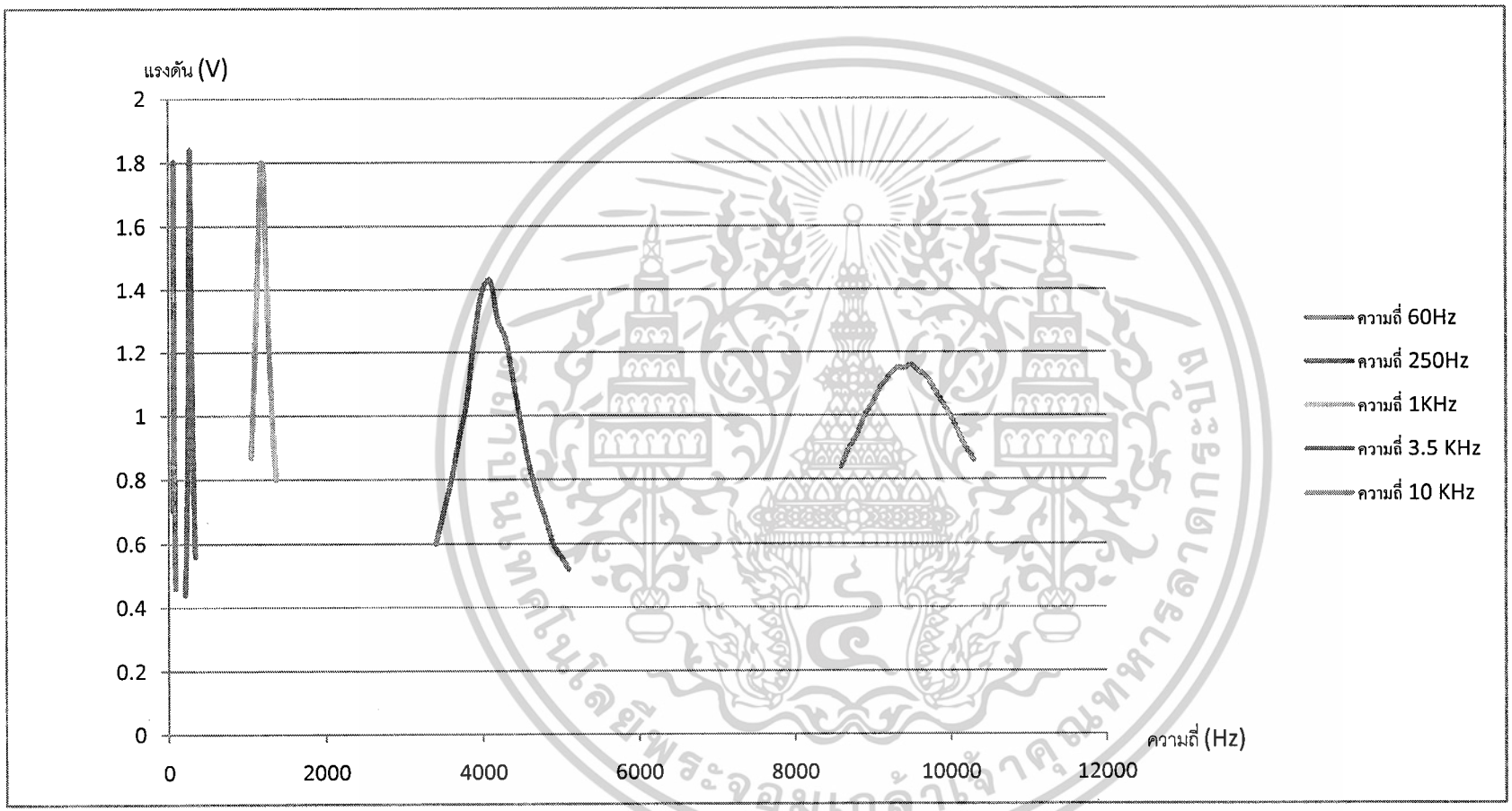
ตารางที่ 4.3 ความถี่ 1 kHz

ความถี่ 3.5 KHz																		
ความถี่(Hz)	3400	3500	3600	3700	3800	3900	4000	4100	4200	4300	4400	4500	4600	4700	4800	4900	5000	5100
แรงดัน(V)	0.6	0.7	0.8	0.93	1.05	1.26	1.4	1.43	1.3	1.24	1.1	0.96	0.84	0.75	0.68	0.6	0.56	0.52

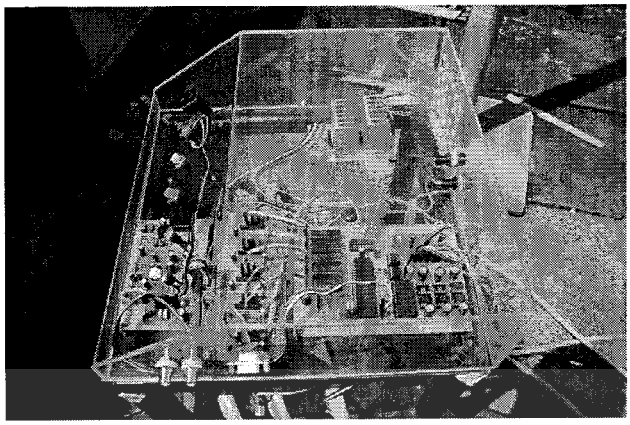
ตารางที่ 4.4 ความถี่ 3.5 kHz

ความถี่ 10 KHz																		
ความถี่(Hz)	8600	8700	8800	8900	9000	9100	9200	9300	9400	9500	9600	9700	9800	9900	10000	10100	10200	10300
แรงดัน(V)	0.84	0.9	0.94	1	1.04	1.09	1.12	1.15	1.15	1.16	1.14	1.12	1.08	1.04	1	0.95	0.9	0.86

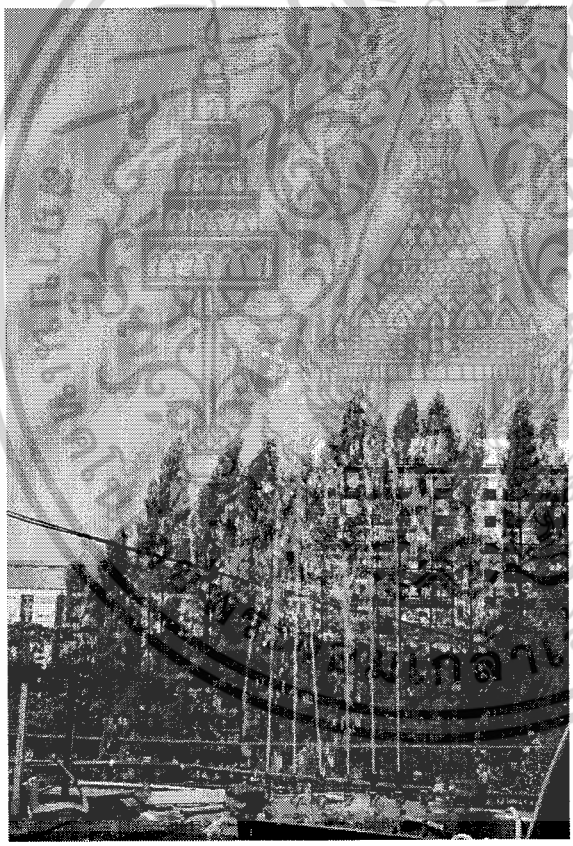
ตารางที่ 4.5 ความถี่ 10 kHz



รูปที่ 4.14 กราฟการทดลอง Musical

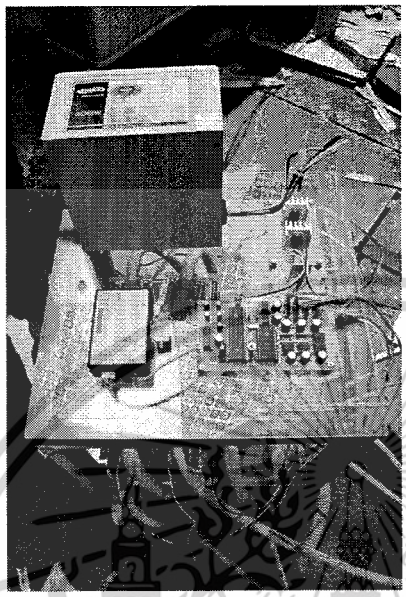


รูปที่ 4.15 วงจรควบคุม Servo ในโหมด Musical



รูปที่ 4.16 การทำงานของหัวน้ำพุตามการควบคุมในโหมด Musical

4.5 ผลการทดลองในส่วนของ Wireless



รูปที่ 4.17 Servo ควบคุมหัวน้ำพุผ่าน Wireless

รูปที่ 4.18 การทำงานของหัวน้ำพุตามการควบคุมในโหมดของ Wireless

บทที่ 5
สรุปและแนวทางในการพัฒนา

ไมโครคอนโทรเลอร์สามารถรับข้อมูลอินพุทจากโปรแกรม C++Builder และสัญญาณเสียงเพลงจากวงจรภายนอก เพื่อนำมากำหนดเซอร์โวมอเตอร์ให้หมุนวาล์วให้มีระดับน้ำออกที่หัวน้ำพุตามระดับต่างๆได้ อีกทั้งยังสามารถส่งผ่านข้อมูลจากเครื่องคอมพิวเตอร์ไปควบคุมการเปิด-ปิดเซอร์โวมอเตอร์ด้วยระบบการส่งแบบไร้สาย (Wireless) ได้

ปัญหาที่พบ : คือการตอบสนองของเซอร์โวมอเตอร์ยังไม่เป็นเชิงเส้นและการตอบสนองต่อสัญญาณเสียงจาก VU ยังไม่ดีพอ

แนวทางในการพัฒนา : ทำการปรับปรุงให้เซอร์โวมอเตอร์สามารถตอบสนองให้เป็นเชิงเส้นมากยิ่งขึ้น, เพิ่มจำนวนเซอร์โวมอเตอร์ให้สามารถควบคุมหัวน้ำพุได้มากขึ้น และพัฒนาให้มีการตอบสนองต่อสัญญาณเสียงจาก VU ได้ดีขึ้น

บรรณานุกรม

1. ดอนสัน ปงผาบ, ทิพย์วัลย์ คำน้ำนอง, ไมโครคอนโทรลเลอร์ PIC และการประยุกต์ใช้งาน, กรุงเทพ, สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2550
2. ประจิน พลังสันติสกุล, เรียนรู้และใช้งาน CCS C คอมไพย์เลอร์เขียนโปรแกรมภาษา C ควบคุมไมโครคอนโทรลเลอร์ PIC , กรุงเทพฯ : บริษัทอินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด
3. นุกูล กระจาย, การเขียนโปรแกรมแบบวิชวลด้วย C++ Builder 5, สุวีริยาสาสน์, กรุงเทพฯ, 2544

ภาคผนวก

Code VU

```
#include <16f877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, stream=PC_COM)
#include <math.h>
char enable,data;
int address=0,count=0,H,C1,n,m,D2,D1=0,D3;
int number=0,mode;
int j=0;
int head[1][24]= {0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,
                  0x1B,0x1C,0x1D,0x1E,0x1F,0x21,0x22,0x23,0x24,0x25,
                  0x26,0x27,0x28,0x29
                 };

#INT_EXT
void IntEXT_isr(void)
{  output_high(pin_B5);
   output_high(pin_B6);
   output_low(pin_B4);
   delay_ms(200);
   output_high(pin_B4);
   delay_ms(200);
   output_low(pin_B4);
   while(TRUE)
   {  if(!input(pin_B1))
      {  mode=1;
         goto outmode;
      }
```

```
    if(!input(pin_B2))
    {  mode=2;
       goto outmode;
    }
    if(!input(pin_B3))
    {  mode=3;
       goto outmode;
    }
  }
  outmode:
  output_high(pin_B4);
  delay_ms(200);
  output_low(pin_B4);
  delay_ms(200);
  output_high(pin_B4);
}

void servoCH1(void);
void servoCH2(void);
void servoCH3(void);
void servoCH4(void);
void servoCH5(void);

void main()
{  enable_interrupts(GLOBAL);
  enable_interrupts(INT_EXT);
  set_tris_B(0x0F);
  set_tris_D(0x07); //B7-B3 output select chanel ,B2-B0 input signal
  set_tris_A(0x00); //D5-D4 output select decoder ,D3-D0 output select servo by decoder
  set_tris_E(0x00); //output level servo
```

```
output_high(pin_B4);
output_high(pin_B5);
output_high(pin_B6);
mode=0;
H=2;
C1=read_eeprom(0);
n=read_eeprom(1);
while(TRUE)
{
  ////////////////////////////start eeprom/////////////////////////////////
  if(mode==1)
  { output_low(pin_B5);
        for(D2=0;D2<C1;D2++)
        { if(H<2||H>n)
          { H=2;
          }
          if(H>=2&&H<=n)
          { m=read_eeprom(H);
            if(m==0)
            { output_E(0x00);  //output servo level 0
              output_A(head[D1][D2]);  //select servo by decoder
            }
            if(m==1)
            { output_E(0x01);  //output servo level 1
              output_A(head[D1][D2]);  //select servo by decoder
            }
            if(m==2)
            { output_E(0x02);  //output servo level 2
              output_A(head[D1][D2]);  //select servo by decoder
            }
```

```
            if(m==3)
            { output_E(0x03);  //output servo level 3
              output_A(head[D1][D2]);  //select servo by decoder
            }
          }
          H++;
        }
        delay_ms(500);
    //}
    output_high(pin_B5);
}
////////////////////////////stop eeprom/////////////////////////////////

////////////////////////////start musical/////////////////////////////////
if(mode==2)
{ output_low(pin_B6);
  output_D(0xF0);  //chanel 1
  if(input(pin_D2)&input(pin_D1)&input(pin_D0))  //check input signal
  { output_E(0x00);  //output servo level 1
      D3=0;
      output_A(head[D1][D3]);
  }
  if(input(pin_D2)&input(pin_D1)&!input(pin_D0))
  { output_E(0x01);  //output servo level 2
      D3=0;
      output_A(head[D1][D3]);
  }
  if(input(pin_D2)&!input(pin_D1)&!input(pin_D0))
  { output_E(0x02);  //output servo level 3
      D3=0;
```

```
    output_A(head[D1][D3]);
}
if(!input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x03);   //output servo level 4
    D3=0;
    output_A(head[D1][D3]);
}

  output_D(0xE8);  //chanel 2
if(input(pin_D2)&input(pin_D1)&input(pin_D0))
{  output_E(0x00);
    D3=1;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&input(pin_D1)&!input(pin_D0))
{  output_E(0x01);
    D3=1;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x02);
    D3=1;
    output_A(head[D1][D3]);
}
if(!input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x03);
    D3=1;
    output_A(head[D1][D3]);
}
```

```
output_D(0xD8);  //chanel 3
if(input(pin_D2)&input(pin_D1)&input(pin_D0))
{  output_E(0x00);
    D3=2;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&input(pin_D1)&!input(pin_D0))
{  output_E(0x01);
    D3=2;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x02);
    D3=2;
    output_A(head[D1][D3]);
}
if(!input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x03);
    D3=2;
    output_A(head[D1][D3]);
}

output_D(0xB8);  //chanel 4
if(input(pin_D2)&input(pin_D1)&input(pin_D0))
{  output_E(0x00);
    D3=3;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&input(pin_D1)&!input(pin_D0))
{  output_E(0x01);
```

```
    D3=3;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x02);
    D3=3;
    output_A(head[D1][D3]);
}
if(!input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x03);
    D3=3;
    output_A(head[D1][D3]);
}

output_D(0x78);  //chanel 5
if(input(pin_D2)&input(pin_D1)&input(pin_D0))
{  output_E(0x00);
    D3=4;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&input(pin_D1)&!input(pin_D0))
{  output_E(0x01);
    D3=4;
    output_A(head[D1][D3]);
}
if(input(pin_D2)&!input(pin_D1)&!input(pin_D0))
{  output_E(0x02);
    D3=4;
    output_A(head[D1][D3]);
}
```

```
  if(!input(pin_D2)&!input(pin_D1)&!input(pin_D0))
  {  output_E(0x03);
      D3=4;
      output_A(head[D1][D3]);
  }
   delay_ms(100);
   output_high(pin_B6);
}
////////////////////////////////stop musical/////////////////////////////////

////////////////////////////////start RS232/////////////////////////////////
if(mode==3)
{  output_low(pin_B5);
   delay_ms(200);
   output_high(pin_B5);
   delay_ms(200);
   output_low(pin_B5);
   delay_ms(200);
   output_high(pin_B5);
   delay_ms(200);
   output_low(pin_B5);
   delay_ms(200);
   j=0;
   H=2;
   enable=getchar();            //CHECK w
   if(enable==0x77)
   {  number=getchar();
     if(0x00<number<=0x08)
     {  write_eeprom(0,number);
        j++;
```

```
      }
      count=2;
      for(address=2;address<=count;address++)
      { data=getchar();
        if(data!=0x73)  //CHECK S
        { write_eeprom(address,data);
          count++;
          j++;              //NUMBER OF ADDRESS
        }
      }
      write_eeprom(1,j);
    }
    C1=read_eeprom(0);
    n=read_eeprom(1);
    mode=1;
    output_high(pin_B5);
    delay_ms(200);
    output_low(pin_B5);
    delay_ms(200);
    output_high(pin_B5);
    delay_ms(200);
    output_low(pin_B5);
    delay_ms(200);
    output_high(pin_B5);
  }

  ////////////////////////////////stop RS232/////////////////////////////////
 }
}
```

Code Wireless

```
#include <16f877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, stream=PC_COM)
#include <math.h>
char enable,data,select;
int address=0,count=0,H,C1,n,m,D2,D1=0,D3;
int number=0,mode;
int j=0;
int head[1][24]= {0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,
                  0x1B,0x1C,0x1D,0x1E,0x1F,0x21,0x22,0x23,0x24,0x25,
                  0x26,0x27,0x28,0x29
                 };
#INT_RDA
void rs232_isr(void)
{  select=getchar();
  if(select==0x74)  //CHECK t    stop
  {
     output_high(pin_B5);
     output_high(pin_B6);
     output_low(pin_B4);
     delay_ms(200);
     output_high(pin_B4);
     delay_ms(200);
     output_low(pin_B4);

      select=getchar();
```

```
if(select==0x65)  //CHECK e   eeprom
{  mode=1;
   output_high(pin_B4);
   delay_ms(200);
   output_low(pin_B4);
   delay_ms(200);
   output_high(pin_B4);
}
else if(select==0x72)  //CHECK r   recieve
{  output_high(pin_B4);
   delay_ms(200);
   output_low(pin_B4);
   delay_ms(200);
   output_high(pin_B4);
   output_low(pin_B5);
   delay_ms(200);
   output_high(pin_B5);
   delay_ms(200);
   output_low(pin_B5);
   delay_ms(200);
   output_high(pin_B5);
   delay_ms(200);
   output_low(pin_B5);
   delay_ms(200);
j=0;
H=2;
enable=getchar(); //recieve
if(enable==0x77)  //CHECK w
{  number=getchar();
   if(0x00<number<=0x08)
```

```
{  write_eeprom(0,number);
  j++;
}
count=2;
for(address=2;address<=count;address++)
{  data=getchar();
  if(data!=0x73)   //CHECK s
  {  write_eeprom(address,data);
    count++;
    j++;              //NUMBER OF ADDRESS
  }
}
```

Code Slave

```
#include<12F675.h>
#fuses INTRC_IO,NOPROTECT,NOMCLR,NOWDT,NOPUT,NOBROWNOUT
#use delay(clock=4000000)
#define GP0 PIN_A0
#define GP1 PIN_A1
#define GP2 PIN_A2
#define GP3 PIN_A3
#define GP4 PIN_A4
#define GP5 PIN_A5
int i;
void main()
{
  set_tris_a(0b11111011);  //set input output for GPO  bit 5,4,3 input 5,4=data 3 enble GP2 output
pulse to servo
  setup_adc_ports(NO_ANALOGS);  //disable adc
  setup_adc(ADC_OFF);
  setup_timer_0(RTCC_INTERNAL);  //disable timer
  setup_timer_1(T1_DISABLED);
  setup_comparator(NC_NC_NC_NC);//disable comparator
  setup_vref(FALSE);//disable vref
  while(TRUE)
  {
      while(!input(GP5)&!input(GP4)&!input(GP3))    //step1
      {
        for(i=0;i<=18;i++)
        {
          output_high (GP2);
          delay_us(460);
```

```
    output_low (GP2);
    delay_us(19540);
  }
}
while(!input(GP5)&input(GP4)&!input(GP3))    //step2
{
  for(i=0;i<=18;i++)
  {
    output_high (GP2);
    delay_us(800);
    output_low (GP2);
    delay_us(19200);
  }
}
while(input(GP5)&!input(GP4)&!input(GP3))    //step3
{
  for(i=0;i<=18;i++)
  {
    output_high (GP2);
    delay_us(1050);
    output_low (GP2);
    delay_us(18950);
  }
}
while(input(GP5)&input(GP4)&!input(GP3))    //step4
{
  for(i=0;i<=18;i++)
  {
    output_high (GP2);
    delay_us(1200);
```

```
  output_low (GP2);
  delay_us(18800);
}
```