ชุดควบคุมสำหรับหุ่นยนต์ขนาดเล็ก

Controller for Microrobot

โดย

นายตรีทรรศ ตรีพัฒนาสุวรรณ

นางสาวตวงทอง สุขสุวานนท์

อาจารย์ที่ปรึกษา

รศ. สมศักดิ์ มิตะถา

อาจารย์ นเรศ มาลัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ชุดควบคุมสำหรับหุ่นยนต์ขนาดเล็ก

Controller for Microrobot

ผู้จัดทำ

1. นายตรีทรรศ ตรีพัฒนาสุวรรณ รหัสประจำตัว 41014149
2. นางสาวตวงทอง สุขสุวานนท์ รหัสประจำตัว 41014487 41014150

_____ อาจารย์ที่ปรึกษา

(รศ. สมศักดิ์ มิตะถา)

_____ อาจารย์ที่ปรึกษา

(อ. นเรศ มาลัย)

ชุดควบคุมสำหรับหุ่นยนต์ขนาดเล็ก

ตรีทรรศ ตรีพัฒนาสุวรรณ
ตวงทอง สุขสุวานนท์
รศ.สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษา
อาจารย์ นเรศ มาลัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

โครงงานชุดควบคุมสำหรับหุ่นยนต์ขนาดเล็ก ได้จัดทำขึ้นเพื่อสร้างชุดควบคุมหุ่นยนต์ขนาดเล็กที่ใช้งานได้อย่างอเนกประสงค์ สามารถรองรับการทำงานของอุปกรณ์อินพุต/เอาท์พุตพื้นฐานต่าง ๆ ได้ เพื่อประโยชน์ทางด้านความสะดวกต่อการใช้งาน และ ความประหยัดโดยมีไมโครคอนโทรลเลอร์ 8051 เป็นหน่วยประมวลผลกลาง และใช้โปรแกรมภาษา C ในการเขียนโปรแกรมควบคุมฟังก์ชันการทำงาน โดยให้ตัวคอมพายเลอร์เป็นตัวแปลงภาษาเครื่อง ส่งคำสั่งที่ได้ไปยังชุดควบคุมหุ่นยนต์ขนาดเล็กโดยผ่านทางพอร์ทอนุกรมของเครื่องคอมพิวเตอร์

ผลที่ได้จากโครงงานนี้คือ จะทำให้เราทราบถึงวิธีการเขียนโปรแกรมภาษา ASM-51 สำหรับไมโครคอนโทรลเลอร์ 8051 และได้เรียนรู้การทำงานของตัวคอมไพเลอร์อีกทั้งได้รู้ถึงการทำงานของฟังก์ชันต่าง ๆ ของซอฟท์แวร์และฮาร์ดแวร์

Controller for Microrobot

Treethad	Treepattanasuwan	
Tuangtong	Suksuwanon	
Assoc. Prof. Somsak	Mitatha	**Advisor**
Nareth	Malai	**Advisor**
2001		

Abstract

This project ,Controller for Microrobot, is aim to build a controller which can support many kind of basic I/O devices. There is Microcontroller 8051,used as Central Processing Unit (CPU) and C language is selected to control the working functions. It can be transformed to machine language by using compiler and sent to Microrobot via serial port.

By doing this project, we have learnt about Microcontroller 8051 and Learnt to operate with C language programming.

สารบัญ

สารบัญ (ต่อ)

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หัวเรื่อง หน้า

บทที่ 1
บทนำ

ในปัจจุบันนี้ การนำไมโครคอนโทรลเลอร์มาใช้ในการควบคุมหุ่นยนต์ขนาดเล็กนั้น มีข้อจำกัดในด้านของความยืดหยุ่นในการใช้งานเป็นอย่างมาก ส่วนมากมักจะนำไปใช้งานเฉพาะทาง ทำให้ไม่สามารถใช้งานคอนโทรลเลอร์ได้อย่างเต็มประสิทธิภาพ ดังนั้นโครงงานนี้จึงสร้างคอนโทรลเลอร์เพื่อใช้ในการควบคุมหุ่นยนต์ขนาดเล็กที่สามารถทำงานได้อย่างอเนกประสงค์ ซึ่งรองรับการทำงานของอุปกรณ์อินพุต/เอาท์พุตพื้นฐานต่าง ๆ เพื่อประโยชน์ทางด้านความสะดวกต่อการใช้งาน และความประหยัด

1.1 จุดประสงค์ของโครงงาน

1. เพื่อสร้างชุดควบคุมสำหรับหุ่นยนต์ขนาดเล็กที่สามารถใช้งานได้อย่างอเนกประสงค์
2. เพื่อศึกษาการทำงานของไมโครคอนโทรลเลอร์ 8051
3. เพื่อให้สามารถเขียนโปรแกรม สำหรับไมโครคอนโทรลเลอร์ 8051 ได้
4. เพื่อให้สามารถออกแบบและสร้างวงจรที่ใช้ควบคุมได้

1.2 ขอบข่ายของโครงงาน

1. สามารถออกแบบและสร้างวงจรควบคุมการทำงานของชุดควบคุมหุ่นยนต์ขนาดเล็กโดยใช้ไมโครคอนโทรลเลอร์ 8051 ได้
2. สามารถส่งคำสั่งที่เขียนแล้วไปยังชุดควบคุมหุ่นยนต์ขนาดเล็กโดยผ่านพอร์ทอนุกรมได้
3. มีฟังก์ชั่นพื้นฐานต่าง ๆ เพื่อสะดวกในการเขียนโปรแกรม
4. มีพอร์ทสำหรับต่อกับอุปกรณ์อินพุต (เซ็นเซอร์ต่าง ๆ) ขนาด 8 บิต จำนวน 10 พอร์ท และมีพอร์ทสำหรับต่อกับอุปกรณ์เอาท์พุต (มอเตอร์ต่าง ๆ) ขนาด 8 บิต จำนวน 10 พอร์ท รองรับการทำงานของอุปกรณ์อินพุต/เอาท์พุตพื้นฐานต่าง ๆ ได้แก่
   - เซ็นเซอร์อินฟราเรด
   - เซ็นเซอร์อัลตร้าโซนิค
   - เซ็นเซอร์โฟโต้ไดโอด
   - สเตปปิ้งมอเตอร์
   - DC มอเตอร์
   - เซอร์โวมอเตอร์

1.3 ลักษณะการใช้งาน

ผู้ใช้สามารถนำอุปกรณ์ Input/Output ที่ออกแบบให้ใช้งานได้ต่อเข้ากับตัวควบคุมที่ตำแหน่งใดก็ได้ (แยก Ground, VCC และ Input/Output) และควบคุมอุปกรณ์ที่ต่อนั้นด้วย ความเปลี่ยนแปลงของสภาพแวดล้อมที่ เซนเซอร์ ได้รับ โดยการเขียนโปรแกรมและ download ผ่านทางพอร์ทอนุกรมลงไปที่ ตัวควบคุม

ประโยชน์ที่ได้จากโครงงานนี้ ทำให้นักศึกษาได้เรียนรู้ และสามารถเขียนโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ 8051 ได้ นอกจากนั้นยังสามารถออกแบบและสร้างวงจรทางฮาร์ดแวร์ได้อีกด้วย

บทที่ 2
อุปกรณ์ที่ใช้กับโครงงาน

เนื่องจากโครงงานนี้มีจุดมุ่งหมายเพื่อออกแบบหุ่นยนต์ขนาดเล็กให้มีความยืดหยุ่นสามารถใช้ได้กับอุปกรณ์อินพุตและเอาต์พุตหลายชนิดตามความต้องการของผู้ใช้จึงได้มีการศึกษาอุปกรณ์ต่าง ๆ ที่หุ่นยนต์ทั่วไปใช้ ซึ่งในที่นี้จะแบ่งออกเป็น 2 ชนิด คือ อุปกรณ์ตรวจจับ และมอเตอร์

2.1 อุปกรณ์ตรวจจับ (Sensor)

ในวงจรอิเล็กทรอนิกส์ในส่วนนำสัญญาณเข้า ที่ทำหน้าที่เป็นส่วนรับรู้ความรู้สึกต่าง ๆ เราเรียกว่า ตัวตรวจจับ (Sensor) ซึ่งจะทำการเปลี่ยนแปลงความรู้สึกต่าง ๆ ที่ได้รับเป็นสัญญาณทางไฟฟ้าซึ่งอาจจะเป็นแรงดันหรือกระแสก็ได้ และส่งให้กับวงจรอิเล็กทรอนิกส์เพื่อตีความหมาย และเอาผลดังกล่าวไปใช้งานได้ตามต้องการ

ตัวตรวจจับแบบพื้นฐานที่เราคุ้นเคยกันอย่างดี เช่น สวิตช์กลไก, สวิตช์แม่เหล็ก, เซลล์รับแสง, โฟโตทรานซิสเตอร์, ออปโต้คัปเปลอร์, ตัวตรวจจับตำแหน่ง, ตัวตรวจจับแรงดัน, ตัวตรวจจับอุณหภูมิ, ตัวตรวจจับเสียง เป็นต้น ตัวตรวจจับต่าง ๆ เหล่านี้ จะทำหน้าที่เปลี่ยนสถานภาพทางฟิสิกส์ให้เป็นสัญญาณทางไฟฟ้า เพื่อนำไปประยุกต์ใช้งานในวงจรอิเล็กทรอนิกส์ให้สามารถทำงานได้ตามต้องการ

ในชุดควบคุมหุ่นยนต์ขนาดเล็กนี้ไม่ได้มีอุปกรณ์ตรวจจับรวมอยู่ในชุดควบคุม แต่ชุดควบคุมถูกออกแบบมาให้สามารถรองรับการทำงานของตัวตรวจจับพื้นฐาน ได้แก่ เซ็นเซอร์อินฟราเรดและ เซนเซอร์อัลตร้าโซนิค โดยการต่อผ่านพอร์ทอินพุตของชุดควบคุมหุ่นยนต์ขนาดเล็ก

2.2 มอเตอร์ (Motor)

2.2.1 สเตปปิ้งมอเตอร์

รูปที่ 2.1 ตัวอย่างสเตปปิ้งมอเตอร์

สเตปปิ้งมอเตอร์ทำหน้าที่เปลี่ยนสัญญาณข้อมูลแบบดิจิตอล ไปสู่การเคลื่อนที่ทางกล อย่างได้สัดส่วนกัน โดยที่แกนหรือโรเตอร์ของมอเตอร์ชนิดนี้มักถูกควบคุมให้หมุนเป็นลำดับขั้น

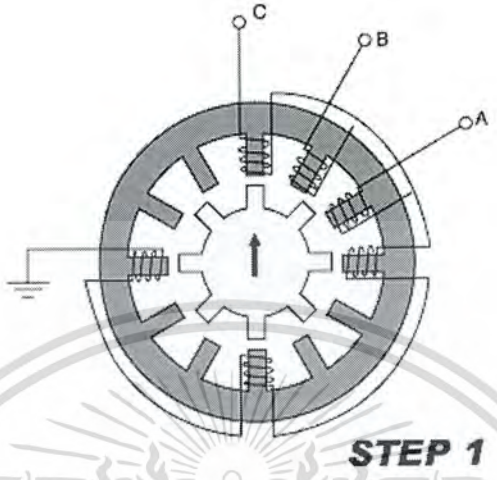
ประโยชน์จากการใช้งานสเตปปิ้งมอเตอร์ที่มีการควบคุมโดยใช้สัญญาณดิจิตอลคือ มีความถูกต้องเที่ยงตรงและสามารถเปลี่ยนตำแหน่งของโหลด ได้อย่างรวดเร็ว เนื่องจากแต่ละอินพุตพัลซ์ จะทำให้สเตปปิ้งมอเตอร์เคลื่อนที่ไปหนึ่งสเตป อย่างเที่ยงตรง

เมื่องพิจารณาในแง่ทางกลจะพบว่าสเตปปิ้งมอเตอร์มีความเที่ยงตรงที่ยอมรับได้ ซึ่งในการเปลี่ยนตำแหน่งอย่างง่ายอาจใช้สวิตช์ ในการควบคุมมอเตอร์ นั่นคือ สร้างวงจรควบคุมมอเตอร์ได้ง่าย ถ้าต้องการเพิ่มประสิทธิภาพของส่วนควบคุม ให้ดีขึ้น อาจใช้ไอซีที่มีความเร็วสูงเนื่องจากประกอบด้วยมอสเฟสอยู่ภายใน จึงได้กำลังงานสูง และต้นทุนต่ำ ความสะดวกในการใช้งานนี้เองจึงทำให้นำไปสู่การใช้งานอย่างแพร่หลาย

การได้รับประโยชน์จากการใช้สเตปปิ้งมอเตอร์อย่างเต็มที่ นั้นขึ้นอยู่กับการขับอย่างถูกต้องเหมาะสม ซึ่งภาคขับนี้จะต้องประกอบไปด้วย แหล่งจ่ายไฟฟ้ากระแสตรง อิเล็กทรอนิกส์สวิตช์ โดยอาจจะเป็นทรานซิสเตอร์หรือมอสเฟส และแหล่งจ่ายสัญญาณข้อมูลแบบดิจิตอล ซึ่งมีลักษณะเป็นพัลซ์ เพื่อใช้ควบคุมทิศทางการหมุนของมอเตอร์

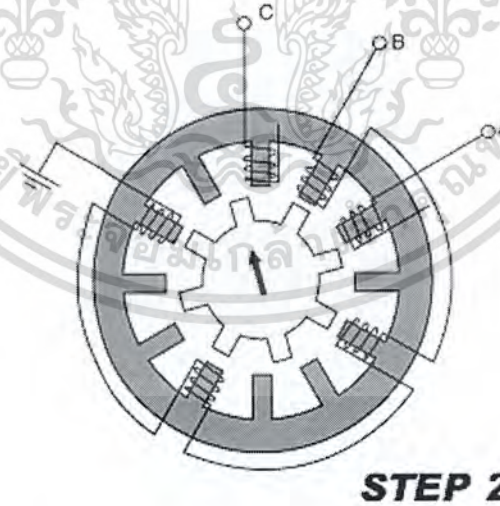
ทิศทางของกระแสไฟฟ้าที่ป้อนเข้าสู่สเตปปิ้งมอเตอร์ ถูกควบคุมโดยการทำงานของอิเล็กทรอนิกส์สวิตช์ ดังนั้นสเตปปิ้งมอเตอร์จะหมุนไปหนึ่งช่วงในแต่ละอินพุตพัลซ์ที่ป้อนเข้าสู่วงจรอิเล็กทรอนิกส์สวิตช์ ซึ่งขนาดมุมที่หมุนไปจะขึ้นอยู่กับการออกแบบสเตปปิ้งมอเตอร์ โดยมีค่าตั้งแต่ 1.8 องศา ถึง 15 องศา ถ้าส่งสัญญาณไปยังวงจรสวิตช์ซึ่ง 24 พัลซ์ โดยมีค่าของหนึ่งช่วงการหมุนเท่ากับ 15 องศา สเตปปิ้งมอเตอร์ก็จะหมุนไปครบ 1 รอบพอดี เวลาที่ใช้ในการหมุนครบ 1 รอบ ขึ้นอยู่กับอัตราการจ่ายสัญญาณพัลซ์ควบคุมโดยสัญญาณนี้อาจถูกผลิตโดยวงจรกำเนิดสัญญาณที่ปรับความถี่ได้หรือจากไอซีควบคุม

การทำงานของสเตปปิ้งมอเตอร์



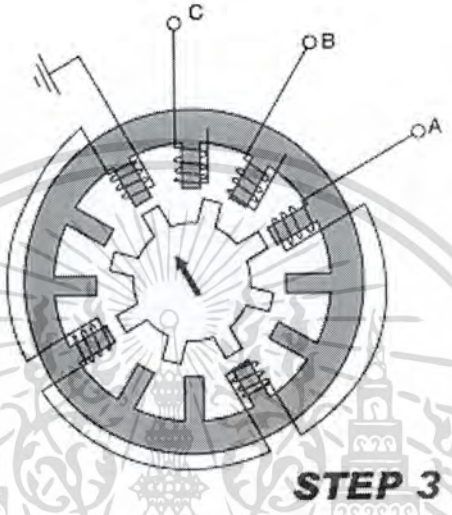
***รูปที่ 2.2** การทำงานของสเตปปิ้งมอเตอร์ขั้นที่ 1*

ตัวสเตเตอร์และโรเตอร์จะถูกสร้างด้วยเหล็กผสมซิลิคอน โดยที่ตัวโรเตอร์จะไม่ถูกพันด้วยคอล์ยแต่ส่วนส่วนที่ถูกพันด้วยคอล์ย จะเป็นส่วนของสเตเตอร์เท่านั้น ดังนั้นเมื่อเราจ่ายกระแสไฟฟ้าผ่านคอล์ย จึงทำให้สเตเตอร์มีคุณสมบัติเป็นแม่เหล็ก



***รูปที่ 2.3** การทำงานของสเตปปิ้งมอเตอร์ขั้นที่ 2*

เราจะจ่ายไฟให้แก่เฟส C ดังนั้นซี่ของโรเตอร์ก็จะถูกดูดเข้า หาเฟส C จากนั้นเราก็หยุดจ่ายไฟให้แก่เฟส C แล้วจ่ายไฟให้แก่ เฟส B ซี่ของสเตเตอร์อันถัดไปที่อยู่ใกล้เฟส B มากที่สุดก็จะถูกดูดเข้ามาแทน และเมื่อเราหยุดจ่ายไฟให้เฟส B และจ่ายไฟที่เฟส A แทนโรเตอร์ที่อยู่ใกล้ A ที่สุดก็จะถูกดูดเข้ามาแทน



รูปที่ 2.4 การทำงานของสเตปปิ้งมอเตอร์ขั้นที่ 3

จากหลักการที่ได้ให้ไปการที่เราจะทำให้มอเตอร์ชนิดนี้หมุนเรา เราก็ต้องจ่ายไฟเรียงลำดับ A B C ซึ่งเราอาศัยไมโครคอนโทรลเลอร์เข้ามาช่วย โดยส่งมาเป็นข้อมูลขนานในการส่งสัญญาน ไปควบคุม Step Motor โดยเราจะส่งค่า 1 2 และ 4 ไปที่พอร์ตตามลำดับวนไปเรื่อยๆ ซึ่งจะเป็นค่าในเลขฐานสอง

```
00000001
00000010
00000100
```

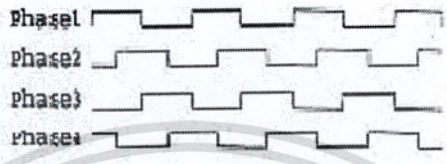
รูปที่ 2.5 ลักษณะสัญญาณที่ส่งไปควบคุมสเตปปิ้งมอเตอร์

สเตปปิ้งมอเตอร์จะหมุนได้นั้นต้องมีการป้อนพัลส์กระตุ้นเข้าที่ลวดแต่ละเฟสของมอเตอร์ ซึ่งวิธีการป้อนพัลส์ที่ต่างกันนั้นสามารถทำได้ดังนี้

1. การป้อนพัลส์กระตุ้นเฟสเดียว (Single phase Excitation) ซึ่งเป็นการป้อนพัลส์ให้กับขดลวดทีละขด ดังรูป

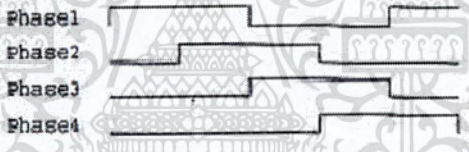
สเตปปิ้งมอเตอร์จะหมุนได้นั้นต้องมีการป้อนพัลส์กระตุ้นเข้าที่ลวดแต่ละเฟสของมอเตอร์ ซึ่งวิธีการป้อนพัลส์ที่ต่างกันนั้นสามารถทำได้ดังนี้

1. การป้อนพัลส์กระตุ้นเฟสเดียว (Single phase Excitation) ซึ่งเป็นการป้อนพัลส์ให้กับขดลวดทีละขด ดังรูป



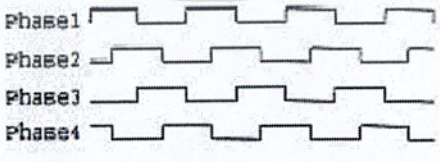
รูปที่ 2.6 การป้อนพัลส์กระตุ้นเฟสเดียว

2. การป้อนพัลส์กระตุ้นสองเฟส (Two phase Excitation) เป็นการป้อนพัลส์ให้กับขดลวดทีละสองขด ดังรูป



รูปที่ 2.7 การป้อนพัลส์กระตุ้นสองเฟส

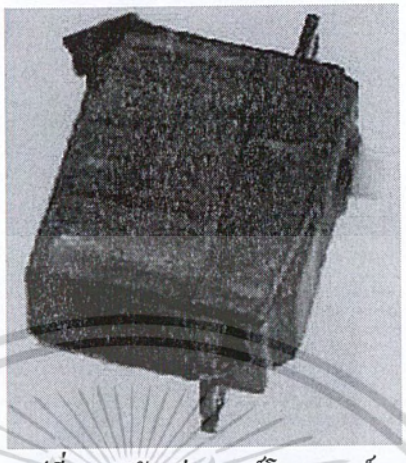
3. การป้อนพัลส์กระตุ้นแบบฮาล์ฟสเตป (Half Step Excitation) การป้อนพัลส์แบบนี้จะทำให้มอเตอร์มีสเตปการหมุนที่ละเอียดขึ้น ดังรูป



รูปที่ 2.8 การป้อนพัลส์กระตุ้นแบบฮาล์ฟสเตป

การป้อนพัลส์กระตุ้นแบบนี้มีความสำคัญมาก เพราะสามารถทำให้สเตปปิ้งมอเตอร์ที่มีความละเอียดในการหมุนต่ำสามารถมีความละเอียดในการหมุนเพิ่มขึ้นได้

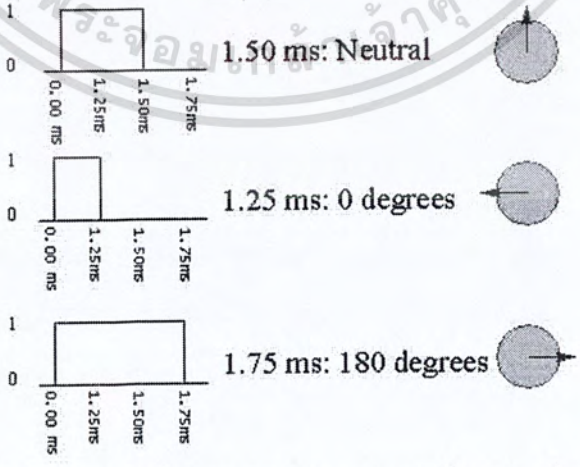
2.2.2 เซอร์โวมอเตอร์



รูปที่ 2.9 ตัวอย่างเซอร์โวมอเตอร์

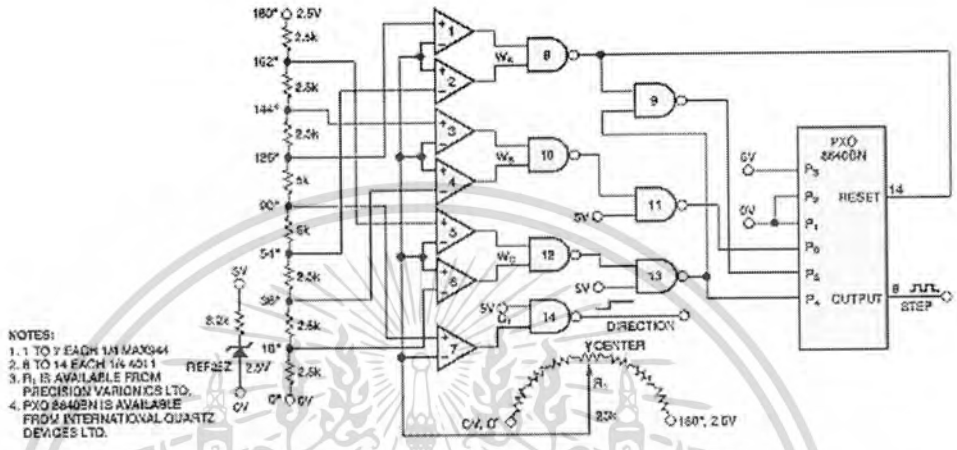
การทำงานของเซอร์โวมอเตอร์ เซอร์โวมอเตอร์ เป็นมอเตอร์ทดเฟืองขนาดเล็กโดยมีแกนส่งกำลัง 1 อัน โดยปกติเซอร์โวมอเตอร์จะหมุนได้เพียง 180 องศา เท่านั้น หรือบางตัวอาจจะถึง 210 องศาขึ้นอยู่กับบริษัทผู้ผลิตและแกนส่งกำลังนี้สามารถที่จะควบคุมทิศทางที่หันไปหรือตำแหน่งที่ต้องการได้ โดยการส่งรหัสควบคุมให้กับตัวเซอร์โวมอเตอร์ ตำแหน่งของแกนส่งสัญญาณดังกล่าวเข้าที่ขาอินพุตตลอดเวลา เซอร์โวมอเตอร์สามารถนำไปประยุกต์ใช้งานกับอุปกรณ์ได้หลากหลาย

ถ้าแกนอยู่ในมุมที่ถูกต้องมอเตอร์ก็จะปิดเองอัตโนมัติ แต่ถ้าวงจรตรวจสอบพบว่ามุมยังไม่ถูกต้องมอเตอร์ก็จะหมุนไปในทางที่ถูกต้องจนกระทั่งได้มุมที่ถูกต้อง และจะหยุดอยู่จนกว่าการป้อนข้อมูลเข้ามาใหม่ การกำหนดองศาของการหมุนทำโดยการส่งสัญญาณ Pulse ค่าระหว่าง 1.5 มิลลิวินาที ถึง 2.0มิลลิวินาทีดังรูป



รูปที่ 2.10 แสดงการควบคุมการทำงานของเซอร์โวมอเตอร์มอเตอร์

เป็นการแสดงลักษณะความกว้างของ Pulse กับทิศทางที่เซอร์โวมอเตอร์หมุนไปในการควบคุมการหมุนของเซอร์โวมอเตอร์นั้นทำโดยการสร้างสัญญาณพัลส์จากไมโครคอนโทรลเลอร์ ไปเข้าสู่เซอร์โวมอเตอร์



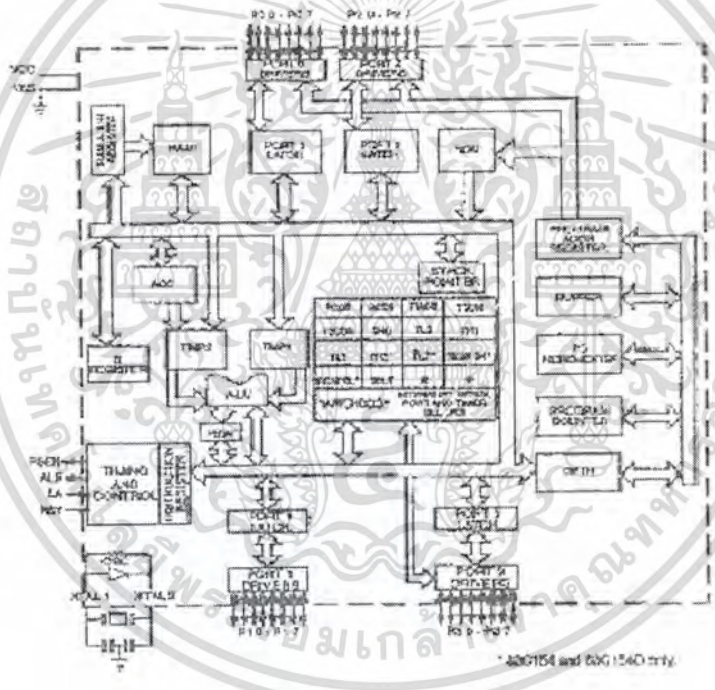
รูปที่ 2.11 แสดงโครงสร้างภายในของเซอร์โวมอเตอร์

บทที่ 3
โครงสร้างของ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 เป็น ไมโครคอนโทรลเลอร์ แบบชิพเดี่ยวที่ผลิตโดยบริษัทอินเทลมีอยู่ด้วยกันหลายเบอร์ด้วยกันซึ่งแต่ละเบอร์นั้นมีคุณสมบัติที่แตกต่างกันตามแต่จะเลือกใช้งานให้เหมาะสมโดยทั่วไปแล้วคุณสมบัติที่แตกต่างกันในแต่ละเบอร์ก็คือ จำนวน Memory ROM RAM, bit I/O

3.1 โครงสร้างภายในของ 8051

ไมโครคอนโทรลเลอร์ MCS-51 ใช้เทคโนโลยีในการผลิตเป็นแบบ NMOS และ CMOS เบอร์ 8032 และ 8052 จะมี ROM BASIC อยู่ภายในจึงสะดวกสำหรับโปรแกรมเมอร์ที่จะเขียนโปรแกรมด้วยภาษาเบสิก โครงสร้างภายในสำหรับเบอร์ 8051 เป็นดังรูปโครงสร้างภายในของ 8051



รูปที่ 3.1 บล็อกไดอะแกรมของ MCS 8051

3.2 คุณสมบัติของไมโครคอนโทรลเลอร์ MCS-51

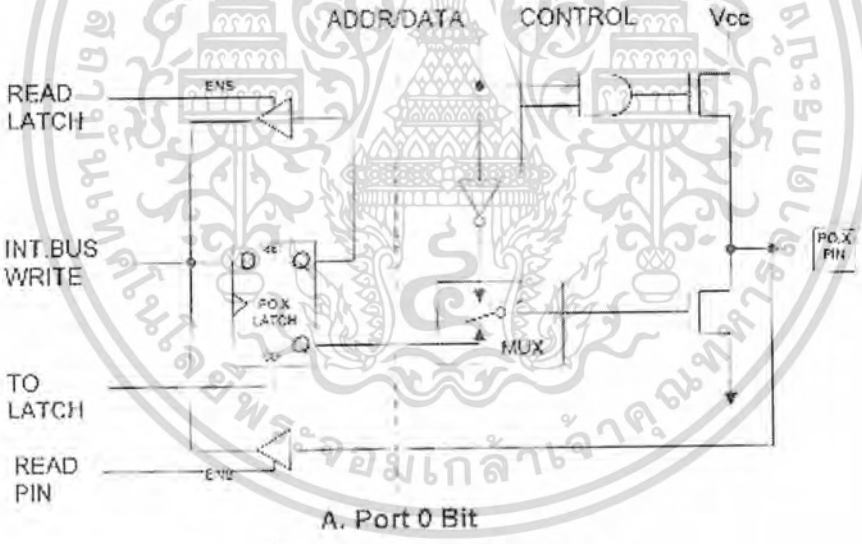
- ต้องการแหล่งจ่ายไฟ +5 โวลต์ ชุดเดียว
- มีหน่วยความจำโปรแกรม (Program Memory) ขนาด 4 กิโลไบต์สำหรับเบอร์ 8051 และ 8031 สำหรับเบอร์ 8052 มีหน่วยความจำถึง 8 กิโลไบต์
- มีหน่วยความจำสำหรับเก็บข้อมูล (Data Memory) ขนาด 128 ไบต์ สำหรับเบอร์ 8052 ขึ้นไปมีถึง 256 ไบต์
- หน่วยวามจำสำหรับเก็บโปรแกรมและข้อมูลแยกจากกันอย่างละ 64 กิโลไบต์

- มีไทม์เมอร์ เคาน์เตอร์ ขนาด 16 บิต 2 ชุด (สำหรับ 8052 มี 3 ชุด) ทำงานได้ 4 โหมด
- รับอินเตอร์รัพท์ได้ 6 แหล่ง 5 เวคเตอร์ สำหรับเบอร์ 8052 ขึ้นไปมี 8 แหล่ง 6 เวคเตอร์
- มีพอร์ทรับส่งข้อมูลอนุกรม (UART) 2 พอร์ทแบบ Full Duplex เลือกรูปแบบได้ 4 โหมด
- มีคำสั่งในการทำ AND,OR หรือ Complement ได้ทั้งแบบ 8 บิตและ 1 บิต

3.3 Port ต่าง ๆ ภายใน 8051

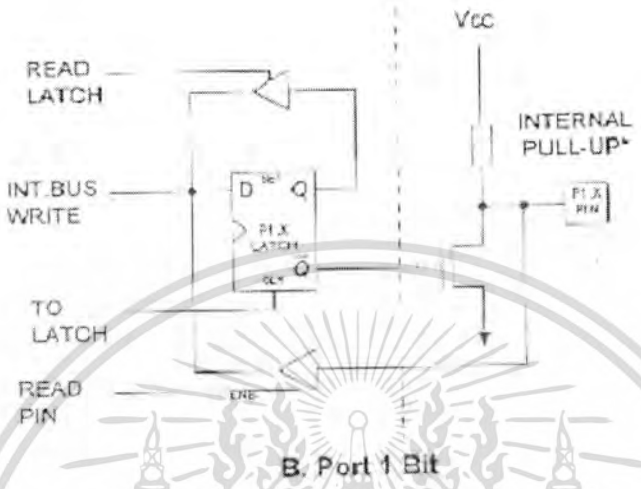
พอร์ทของ 8051 เป็น ไมโครคอนโทรลเลอร์ ขนาด 40 ขา ซึ่งมีขาต่าง ๆดังนี้

- Vcc (ขา 40) ต่อกับ +5V
- Vss (ขา 20)เป็นขา GND
- พอร์ท 0 (ขา 32-39) มีทั้งหมด 8 บิต คือ P0.7- P0.0 ใช้งานได้ 2 หน้าที่คือ แอดเดรสบัส และ ดาต้าบัส เมื่อต้องการให้ทำงานเป็นอินพุตพอร์ทต้องส่งลอจิก "1" ไปยังพอร์ทนี้ จะมีผลให้ Q bar ของ D-FF เป็น "0" ทำให้ FETตัวล่างมีสถานะเป็น OFF สัญญาณที่ใช้อ่านอินพุตพอร์ทแลทช์ โดยส่งสัญญาณ READ LATCH ไปกระตุ้นที่ Tri State Buffer ตัวบน และการอ่าน Port (pin) จะใช้สัญญาณ Read(pin)



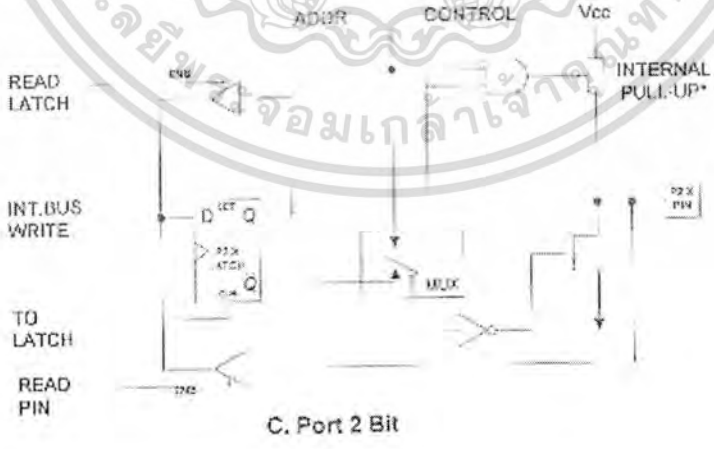
รูปที่ **3.2** *แสดงโครงสร้าง port 0 (bit)*

- พอร์ท 1 (ขา 1-8) มีทั้งหมด 8 บิตคือ (P1.0- P1.7) มีโครงสร้างคล้าย พอร์ท 0 แต่จะใช้ความต้านทานภายในพูลอัพแทน Internal Pull up Register มีโครงสร้างดังรูป



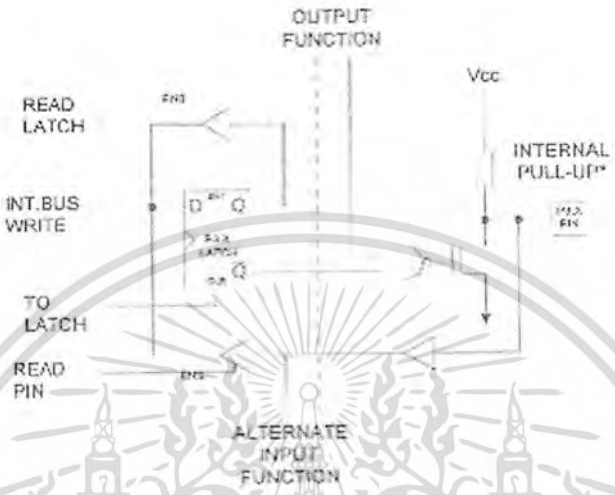
รูปที่ 3.3 แสดงโครงสร้าง port 1 (bit)

- พอร์ท 2 (ขา 21-28) มีทั้งหมด 8 บิต คือขา (P2.7- P2.0)มีโครงสร้างคล้าย พอร์ท 0 โดยมี FET ตัวล่างตัวเดียวส่วนด้านบนใช้ความต้านทานพูลอัพแทน(Internal Pull up) พอร์ทนี้
- ทำงาน 2 หน้าที่ คือสามารถใช้เป็น แอดเดรสบัสขนาด 8 บิต และเป็น ไอโอพอร์ทใช้งานทั่วไป เมื่อจะใช้งานเป็นอินพุตพอร์ทต้องส่งลอจิก "1" มาที่พอร์ทนี้ก่อนเพื่อบังคับให้ FETอยู่ในสภาวะ off ดังรูป



รูปที่ 3.4 แสดงโครงสร้าง port 2 (bit)

- พอร์ท 3 (ขา 10-17) มีทั้งหมด 8 บิต คือ ขา P3.7-P3.0 มีโครงสร้างคล้าย พอร์ท 1 ทำงานได้ 2 หน้าที่คือ เป็นไอโอพอร์ท ถ้าจะโปรแกรมให้เป็น อินพุตพอร์ทต้องส่ง ลอจิก "1" มาที่พอร์ทนี้ก่อน



รูปที่ 3.5 แสดงโครงสร้าง port 2 (bit)

อีกหน้าที่หนึ่งก็คือใช้ส่งสัญญาณควบคุมออกมาและรับสัญญาณเข้าไปสัญญาณต่าง ๆ ดังนี้

P3.0/RXD (Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม (UART)

P3.1/TXD (Serial Output Port)เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม (UART)

P3.2/INT0 (External Interrupt 0)ใช้รับสัญญาณการขัดจังหวะจากภายนอกเบอร์ 0

P3.3/INT1 (External Interrupt 1) ใช้รับสัญญาณการขัดจังหวะจากภายนอกเบอร์ 1

P3.4/T0 (Counter 0 External Input) ขารับสัญญาณ Pulse input เข้าไปยังวงจร counter เป็น input โหมด counter

P3.5/T1 (Counter 1 External Input) ขารับสัญญาณ Pulse input เข้าไปยังวงจร counter 1 เป็นโหมด counter

P3.6/WR (External Data Memory Write Strobe) ขาสัญญาณควบคุมการเขียนข้อมูลลงหน่วยความจำข้อมูลภายนอก

P3.7/RD (External Data Memory Read Strobe) ขาสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำข้อมูลภายนอก

ALE (ขา 30) เป็นขาส่ง Strobe สำหรับใช้ในการ latch address byte ต่ำ (A7-A0) ที่ส่งออกมาจาก port 0 สัญญาณนี้จะ active ทุก ๆ 2 ครั้งใน 1 machine cycle

PSEN (ขา 29) เป็นขา Strobe สำหรับใช้อ่านข้อมูลจาก Program Memory ภายนอกสัญญาณนี้จะส่งออกมา 2 ครั้งในแต่ละ Machine cycle แต่ถ้าเป็นการอ่าน Internal Program Memory จะไม่มีสัญญาณออกที่ขานี้

EA (ขา 31) ใช้เลือกหน่วยความจำโปรแกรมภายนอก

ป้อน " 0 " จะอ่านโปรแกรมจากภายนอกชิพ

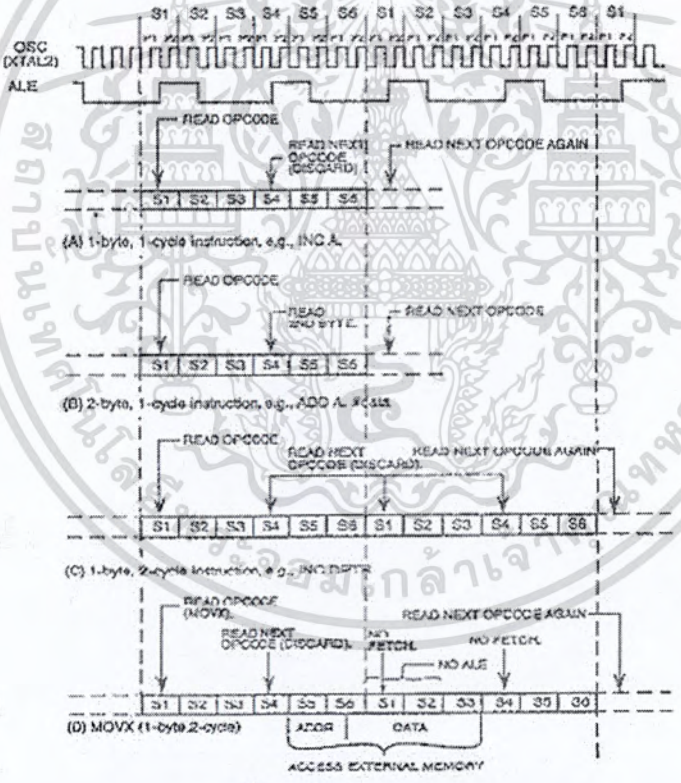
ป้อน " 1 " จะอ่านโปรแกรมจากภายในชิพ

RST (ขา 9) ขา reset จะ reset ได้ก็ต่อเมื่อป้อนลอจิก " 1 " เข้าที่ขานี้นานอย่างน้อย 2 machine cycle

XTAL1 (ขา 19) ใช้ต่อคริสตอลภายนอกโดยเป็นอินพุตเข้าสู่วงจรออสซิสเลเตอร์ภายใน

XTAL2 (ขา 18) ใช้ต่อคริสตอลภายนอกโดยเป็นเอาท์พุตของวงจรออสซิสเลเตอร์ภายใน

3.4 ผังเวลาของ CPU (CPU Timing)



รูปที่ 3.6 ผังเวลาของ CPU

การทำงานใน 1 คำสั่งต่ำสุดจะกินเวลาเพียง 1 ไมโครวินาที เช่นคำสั่ง INC A ซึ่งเป็นคำสั่ง 1 ไบต์ 1 machine cycle ซึ่งจะใช้ clock ไปเท่ากับ 12 ลูก โดย clock ลูกที่ 1 และ 2 จะอยู่ในช่วง S1 P1 และ S1 P2 และ clock ลูกที่ 12 ก็จะอยู่ในช่วง S6P2 นั่งเอง (ปรกติแล้ว CPU จะ RUN ด้วยความเร็วเท่ากับ 12 MHz ดังนั้น clock 12 ลูกจะกินเวลาเท่ากับ 1 ไมโครวินาที)

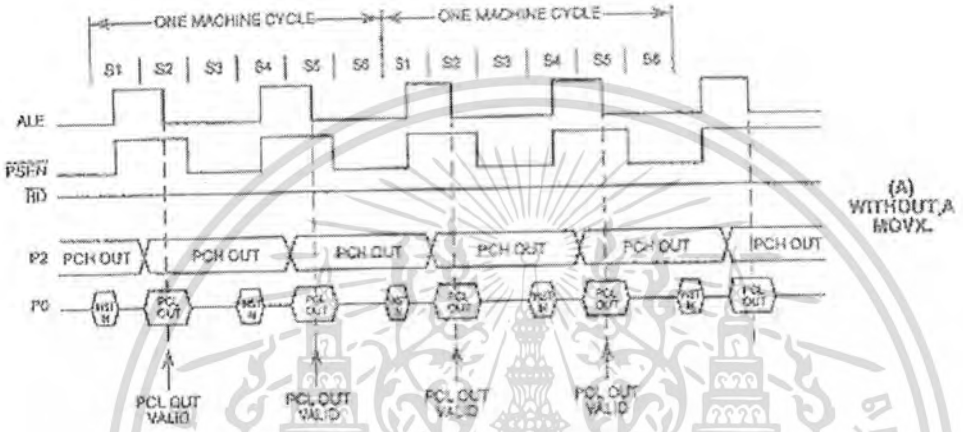
* คำว่า 1 machine cycle คือช่วงการทำงานตั้งแต่ S1 จนถึง S6 *

รูป (a) แสดงการทำงานของคำสั่ง INC A ซึ่งเป็นคำสั่ง 1 ไบต์ทำงานเสร็จภายใน 1 machine cycle

รูป (b) แสดงการทำงานของคำสั่ง ADD A.#Data ซึ่งเป็นคำสั่ง 2 ไบต์ทำงานเสร็จภายใน 1 machine cycle

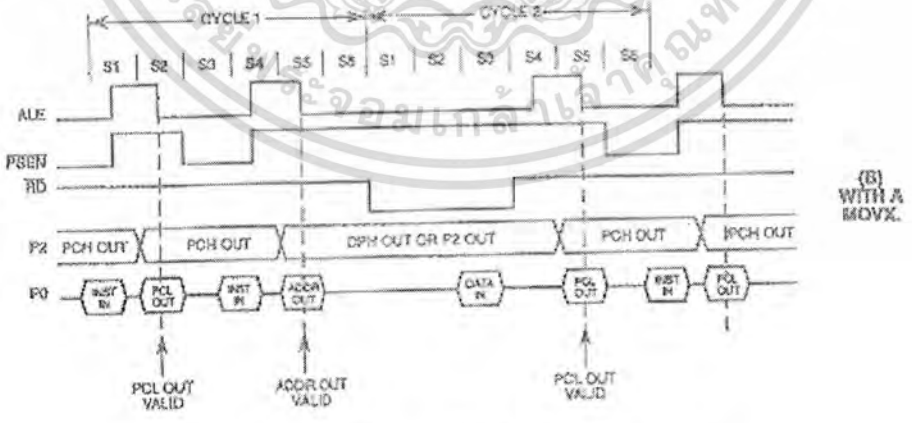
รูป (c) แสดงการทำงานของคำสั่ง INC DPTR ซึ่งเป็นคำสั่ง 1 ไบต์ แต่ทำงานเสร็จใน 2 machine cycle

รูป (d) แสดงการทำงานของคำสั่ง MOVX ซึ่งเป็นคำสั่ง 1 ไบต์ แต่ทำงานเสร็จใน 2 machine cycle



รูปที่ 3.7 ผังเวลาของสัญญาณซึ่งเกี่ยวข้องกับ Fetch

รูปที่ 3.7 เป็นผังเวลาของสัญญาณซึ่งเกี่ยวข้องกับ Fetch เมื่อส่วนของ Program Memory อยู่ภายนอก ดังนั้น สัญญาณที่จำนำไปใช้อ่านออปโค้ด จาก Program Memory ก็คือ PSEN ซึ่งจะ active 2 ครั้งใน 1 machine cycle ดังนั้น สัญญาณที่ใช้อ่านข้อมูลจาก Program Memory จะใช้สัญญาณ PSEN



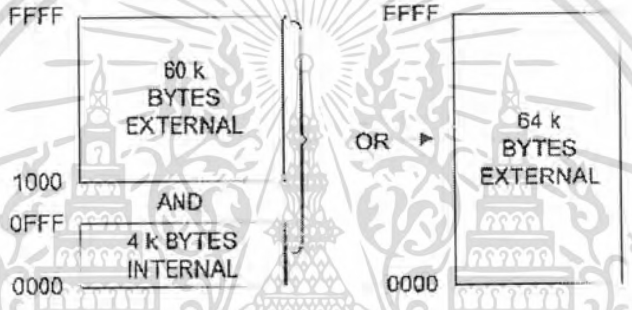
รูปที่ 3.8 ผังเวลาของสัญญาณที่ใช้อ่านข้อมูลจาก Data Memory

รูปที่ 3.8 เป็นผังเวลาของสัญญาณที่ใช้อ่านข้อมูลจาก Data Memory สัญญาณ PSEN จะมีเพียง 1 ลูก เพราะช่วงเวลาถัดมาจะเป็นช่วงเวลาในการอ่านข้อมูลจาก Data Memory โดยใช้สัญญาณ RD

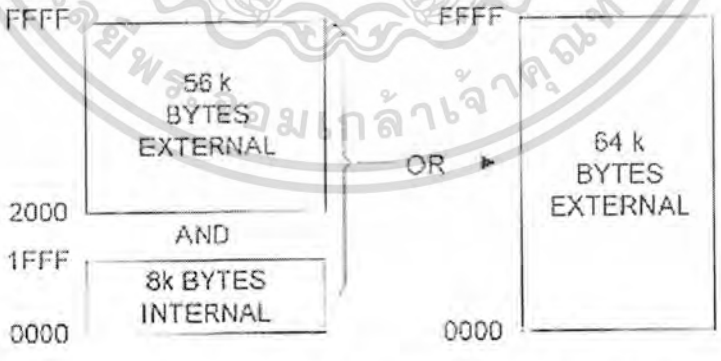
3.5 การแบ่งประเภทของหน่วยความจำ

หน่วยความจำที่ใช้กับ MCS –51 มีอยู่ด้วยกัน 2 ชนิด คือ หน่วยความจำสำหรับเก็บโปรแกรม (Program Memory) และหน่วยความจำสำหรับเก็บข้อมูล (Data Memory)

Program Memory หน่วยความจำสำหรับเก็บโปรแกรม เป็นหน่วยความจำที่ใช้เก็บโปรแกรมสั่งงานบรรจุอยู่ในชิพ 8051 ส่วนที่เป็น Program Memory ก็คือ ROM ขนาด 4 กิโลไบต์นั่นเอง แต่ถ้าเป็นเบอร์ 8052 จะมี ROM ขนาด 8 กิโลไบต์ ดังแสดงในรูปที่ 3.9 และ 3.10

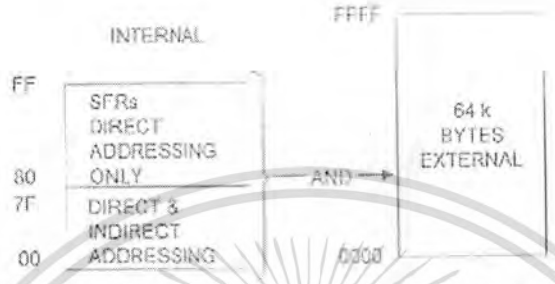


รูปที่ 3.9 ผังหน่วยความจำสำหรับโปรแกรมสำหรับเบอร์ 8051



รูปที่ 3.10 หน่วยความจำสำหรับเก็บโปรแกรมสำหรับเบอร์ 8052

Data Memory (RAM) แบ่งเป็น 2 ส่วน คือ หน่วยความจำข้อมูลภายในชิพ มีเพียง 128 ไบต์ สำหรับเบอร์ 8051 และ 256 ไบต์ สำหรับเบอร์ 8052 ขึ้นไป และหน่วยความจำข้อมูลภายนอกชิพมีความจุ 64 กิโลไบต์ ดังแสดงในรูปที่ 3.11 และ 3.12



รูปที่ 3.11 ผังหน่วยความจำสำหรับ Data Memory เบอร์ 8051

รูปที่ 3.12 ผังหน่วยความจำสำหรับ Program Memory ของ 8052

พื้นที่หน่วยความจำที่เข้าถึงข้อมูลทางอ้อมเท่านั้น (Indirect Addreses Area)

พื้นที่หน่วนความจำบริเวณ (80h – FFh) เป็นพื้นที่ซ้อนกันอยู่อย่างละ 128 ไบต์ โดยส่วนแรกจะเป็น SFR address และ Indirect Address Area ดังนั้นถ้าจะติดต่อกับ SFR จะต้องใช้คำสั่งแบบเข้าถึงข้อมูลโดยตรงเท่านั้น (Direct Address Area) ส่วนพื้นที่อีกส่วนหนึ่งจะเข้าถึงข้อมูลแบบทางอ้อมเท่านั้น (Indirect Address Area) ส่วนตำแหน่ง (00h – 7Fh) จะเข้าถึงข้อมูลได้ทั้ง 2 แบบ

พื้นที่หน่วยความจำที่เข้าถึงข้อมูลโดยตรงและทางอ้อม (Direct and Indirect Address Area)

พื้นที่ 128 ไบต์ ล่างสุดจะแบ่งเป็น 3 ส่วน

1. Register Bank (Register Banks 0-3)

ตั้งแต่ตำแหน่ง (00-Fh) จะเป็นส่วนของรีจิสเตอร์แบงค์ (0-3 ๙ โดยแบ่งเป็นแบงค์ละ 8 ไบต์รวมแล้วได้ 32 ไบต์ (แต่ละแบงค์จะมีรีจิสเตอร์ R0,R1,R2,R3,R4,R5,R6,R7) ถ้า CPU ทำงานอยู่ที่แบงค์ 3 เมื่อถูก reset ก็จะกลับมาทำงานที่แบงค์ 0 เสมอ และ SP จะมาเริ่มต้นที่ตำแหน่ง 07h ทันที

2. บริเวณหน่วยความจำที่ใช้คำสั่งอ่านเขียนทีละกับบิตได้ (Bit Addressable Area)

พื้นที่ตั้ง address (20h-7Fh) จำนวน 16 ไบต์หรือแบ่งเป็นบิตจะได้เท่ากับ 128 บิตซึ่งจะมีตำแหน่งบิตดังนี้ 00,01,02,03,04,05,06,07 จนถึง 7Fh

บริเวณหน่วยความจำที่ใช้งานทั่วไป (Scratch Pad Area)

พื้นที่ตั้งแต่ (30h -7Fh) จะเขียนข้อมูลได้ทีละไบต์เท่านั้น ไม่สามารถใช้คำสั่งเกี่ยวกับบิตได้ถ้าย้ายเนื้อที่สแตกมาบริเวณนี้

Special Function Register (SFR) มีรายละเอียดดังตาราง

Symbol	Name	Address	Symbol	Name	Address
*ACC	Accumulator	0E0H	TMOD	Timer/Counter Mode	89H
*B	B Register	0F0H	*TCON	Timer/Counter Control	88H
*PSW	Program Status Word	0D0H	*+T2CON	Timer/Counter 2 Control	0C8H
SP	Stack Pointer	81H	TH0	Timer/Counter 0 Hight Byte	8CH
DPTR	Data Pointer 2 Bytes		TL0	Timer/Counter 0 Low Byte	8AH
DPL	Low Byte	82H	TH1	Timer/Counter 1 Hight Byte	8DH
DPH	High Byte	83H	TL1	Timer/Counter 1 Low Byte	8BH
*P0	Port 0	80H	+TH2	Timer/Counter 2 Hight Byte	0CDH
*P1	Port 1	90H	+TL2	Timer/Counter 2 Low Byte	0CCH
*P2	Port 2	0A0H	+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
*P3	Port 3	0B0H	+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*IP	Interupt Priority Control	0B8H	*SCON	Serial Control	98H
*IE	Interupt Enable Control	0A8H	SBUF	Serial Data Buffer	99H
			PCON	Power Control	87H

* = Bit Addressable + = 8052 only

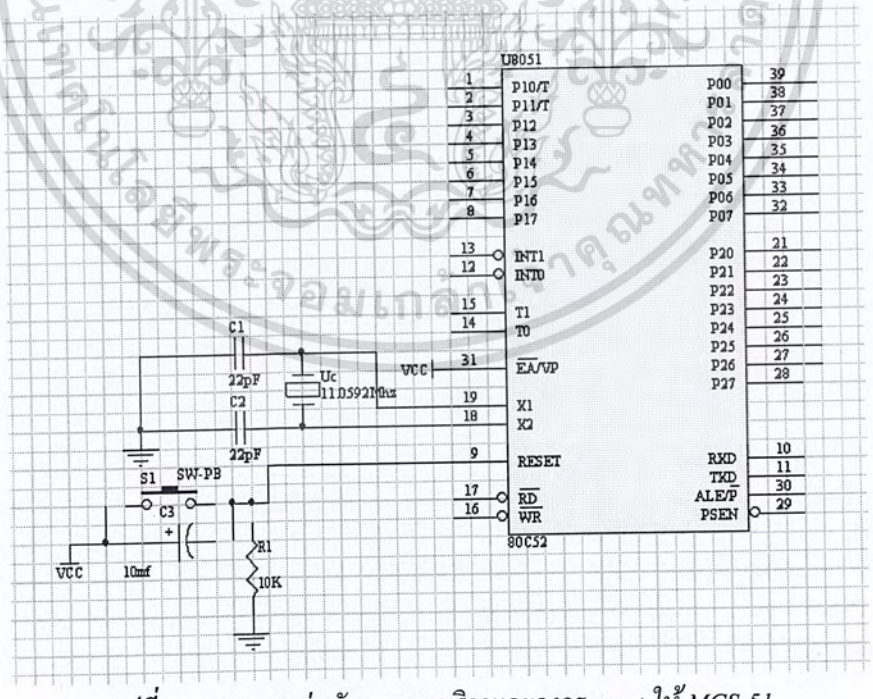
ตารางที่ 3.1 *แสดงรายละเอียดของ Special Function Register*

บทที่ 4
การสร้างและการออกแบบในส่วนฮาร์ดแวร์

จุดมุ่งหมายหลักของการออกแบบวงจรเพื่อออกแบบให้วงจรนี้สามารถดาวน์โหลดโปรแกรมจากผู้ใช้มาเก็บไว้และทำงานตามที่ผู้ใช้โปรแกรมไว้ได้และในส่วนของวงจรอินพุตและเอาต์พุตนั้นก็ออกแบบไว้เพื่อให้สามารถทำงานกับอุปกรณ์อินพุตและเอาต์พุตที่ออกแบบไว้ได้ในโครงงานชุดควบคุมหุ่นยนต์ขนาดเล็กนี้ จะแบ่งการออกแบบออกเป็น 5 ส่วน ได้แก่ การจ่ายสัญญาณนาฬิกาและการออกแบบปุ่ม reset ให้ MCS-51, การออกแบบการใช้หน่วยความจำภายนอก, การออกแบบวงจรเพื่อติดต่อกับคอมพิวเตอร์ผ่านพอร์ตอนุกรม, การออกแบบวงจรสำหรับเอาต์พุตพอร์ต และการออกแบบวงจรสำหรับอินพุตพอร์ต

4.1 การจ่ายสัญญาณนาฬิกาและการออกแบบปุ่ม reset ให้ MCS-51

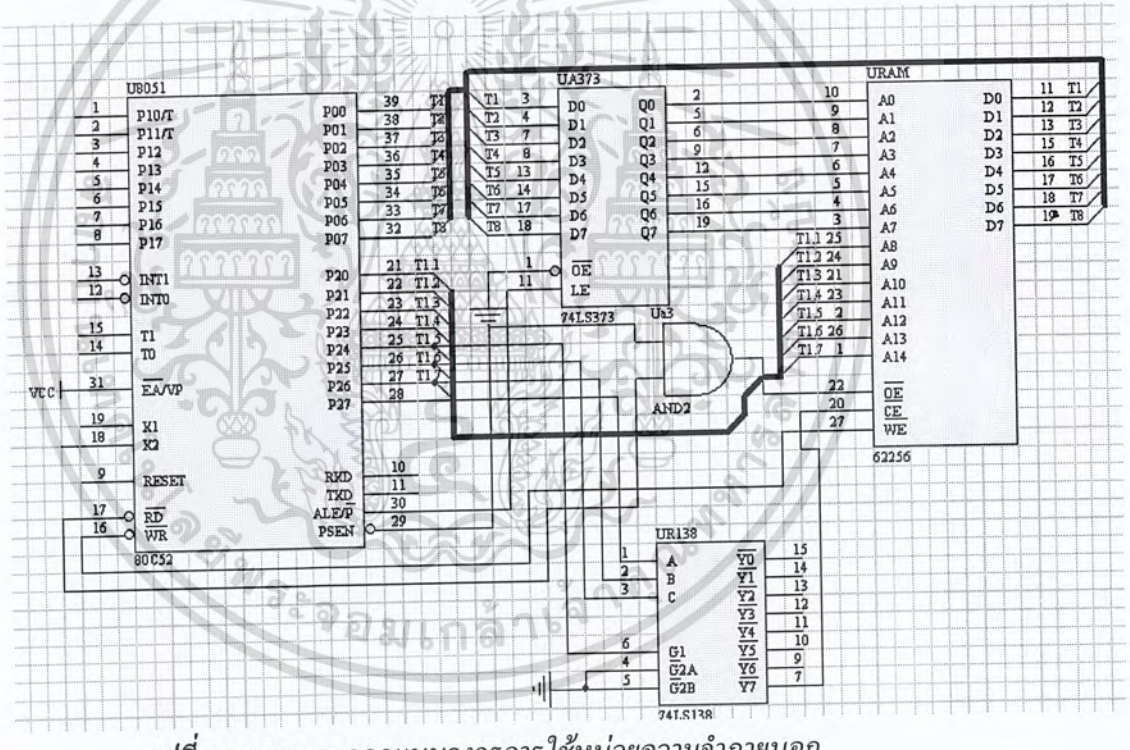
เนื่องจาก MCS-51 จะทำงานได้ต้องมีการจ่ายสัญญาณนาฬิกาให้ดังนั้นเราจึงต่อคริสตัลให้กับ MCS-51 โดยต่อเข้าที่ขา 19,18 โดยสัญญาณนาฬิกาที่ MCS-51 ได้รับนี้จะใช้ในการกำหนดจังหวะการทำงานของ MCS-51 ในส่วนของ วงจร Reset นั้นจะเป็นการต่อสวิตช์เพื่อควบคุมการ Reset เพื่อให้ MCS-51 ไปทำงานที่ตำแหน่งเริ่มต้นอีกครั้งโดยทั้งการ Reset และการให้สัญญาณนาฬิกาแก่ MCS-51 สามารถทำได้ ดังรูป



รูปที่ 4.1 แสดงการต่อสัญญาณนาฬิกาและวงจร reset ให้ MCS-51

4.2 การออกแบบการใช้หน่วยความจำภายนอก

เนื่องจากต้องการใช้งานหน่วยความจำภายนอกเป็นหน่วยความจำข้อมูล (เก็บไว้ชั่วคราวในลักษณะ ram) แต่ต้องการให้เก็บโปรแกรมจึงออกแบบให้ใช้ ram ในการเก็บแต่ใช้งานเหมือนกับหน่วยความจำโปรแกรมจึงต้องใช้ขา RD (ขา 17)ของ MCS-51 นำไป and กับขา PSEN (ขา 29) ของ MCS-51 แล้วจึงนำไปต่อกับขา OE (ขา 22) ของ 62256 เพื่อให้สามารถอ่านข้อมูลได้เมื่อใช้ ram ในลักษณะของ หน่วยความจำโปรแกรม และในส่วนของ 74LS138 จะทำหน้าที่กำหนดตำแหน่งข้อมูลที่ 62256 เก็บอยู่โดยในการออกแบบจะให้เริ่มตั้งแต่ตำแหน่ง F000H สังเกตุได้จากเมื่อขา 25,26,27,28 ของ MCS-51 เป็น "1" ทั้งหมด 74LS138 จะส่งสัญญาณ "0" ออกมาจากขา 7 เพื่อไปทำการ enable ให้ 62256 ทำงาน และใช้พอร์ต 0 ทำหน้าที่เป็นทั้งขาข้อมูลและขาบอกตำแหน่ง address โดยตำแหน่ง address นั้นจะถูกแลตช์ค่าไว้ที่ 74LS373 ดังรูป



รูปที่ 4.2 แสดงการออกแบบวงจรการใช้หน่วยความจำภายนอก

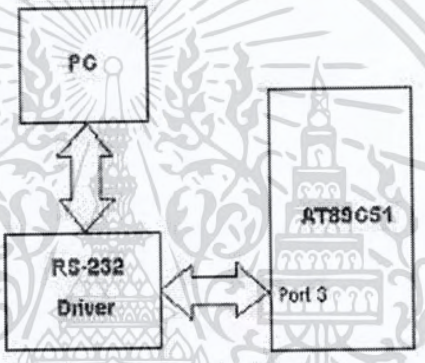
4.3 การออกแบบวงจรเพื่อติดต่อกับคอมพิวเตอร์ผ่านพอร์ตอนุกรม

เนื่องจากต้องการออกแบบให้ผู้ใช้สามารถดาวน์โหลดโปรแกรมที่ผู้ใช้เขียนขึ้นเองลงบนหน่วยความจำและให้ไปทำงานตามที่ผู้ใช้ได้โปรแกรมไว้จึงต้องมีการติดต่อกับเครื่องคอมพิวเตอร์โดยออกแบบให้ติดต่อผ่านพอร์ตอนุกรมและเนื่องจากแรงดันไฟฟ้าภายในพอร์ตอนุกรมนั้นไม่เท่ากับแรงดันไฟฟ้าของระบบทีทีแอลซึ่งใช้ใน MCS-51 ดังนั้นจึงต้องทำการปรับแรงดันให้สามารถใช้ได้กับ MCS-

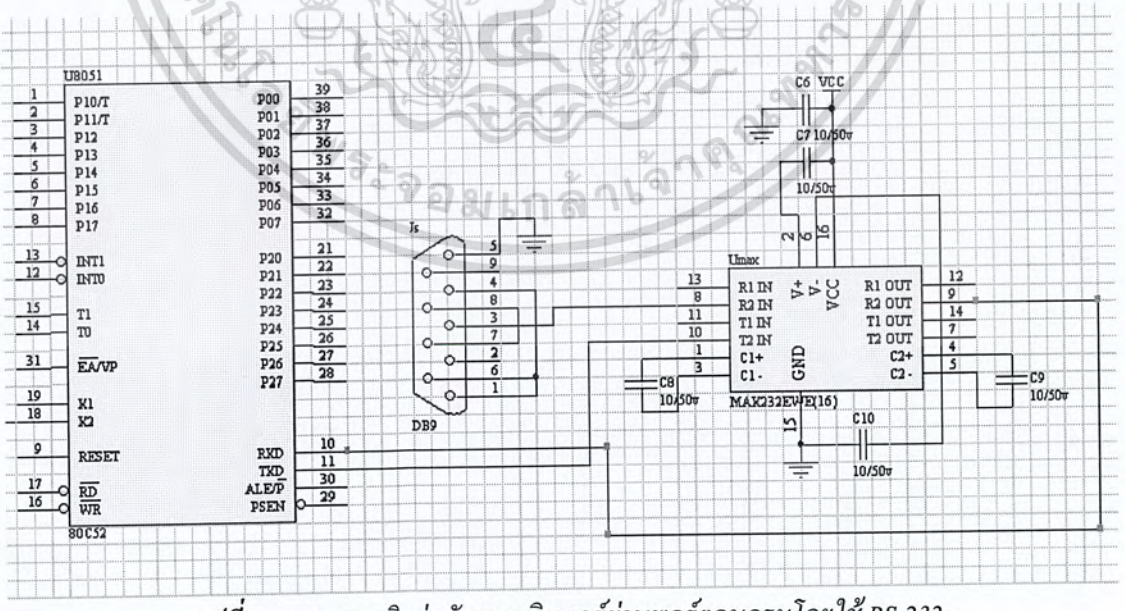
51 ในการเชื่อมต่อวงจรที่ทำงานในระดับแรงดันแบบทีทีแอล (TTL) เข้ากับพอร์ท RS-232 ของเครื่องคอมพิวเตอร์ ซึ่งมีระดับแรงดัน –15 ถึง +15 นั้น จะต้องมีวงจรพิเศษเพื่อทำการแปลงระดับแรงดันให้เหมาะสม ซึ่งในที่นี้เราได้เลือกใช้ไอซีเบอร์ MAX 232 ซึ่งเป็นไอซีที่ใช้อุปกรณ์ประกอบจากภายนอกน้อย คือใช้ตัวเก็บประจุ (C) ชนิดอีเล็คโทรไลท์เพียง 5 ตัวเท่านั้น

หลักการทำงาน

จากวงจรจะใช้ C ที่ต่อระหว่างขา 1 กับ 3 ,ระหว่างขา 4 กับ 5 , ขา 2 และขา 6 เป็นตัวกำหนดระดับแรงดันในการเชื่อมต่อ โดยขา R1I และ R2I จะเป็นขาที่รับระดับแรงดัน –15 ถึง +15 และแปลงออกเป็นแรงดัน 0V , +5V ออกที่ขา R1O และ R2O ส่วนขา T1I และ T2I ก็จะรับแรงดันที่เป็น 0V, +5V แปลงเป็นระดับแรงดัน –10 ถึง +10 ออกที่ขา T1O และ T2O



รูปที่ 4.3 โครงสร้างการเชื่อมต่อของโมดูลการรับส่งข้อมูลแบบอนุกรม



รูปที่ 4.4 แสดงการติดต่อกับคอมพิวเตอร์ผ่านพอร์ตอนุกรมโดยใช้ RS-232

4.4 การออกแบบวงจรสำหรับเอาต์พุตพอร์ต

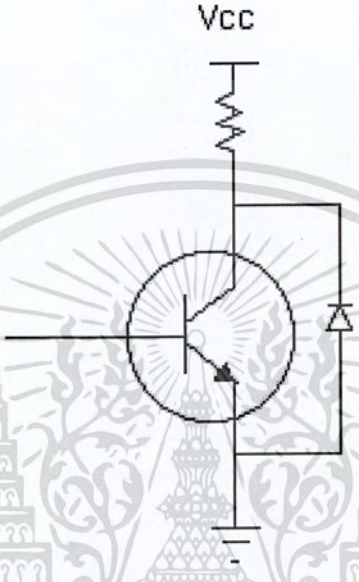
โมดูล เอาท์พุต นี้จะเป็นส่วนที่ให้ผู้ใช้สามารถติดอุปกรณ์ เอาท์พุต ซึ่งส่วนมากแล้วจะเป็น มอเตอร์ เพื่อตอบสนองความต้องการของผู้ใช้โดยทั่วไปแล้วจะสัมพันธ์กับค่าตามที่ เซนเซอร์ ได้รับมาและจากการประมวลผลจาก ไมโครโรบอท แล้วทำให้ผู้ใช้สามารถควบคุมการทำงานของ เอาท์พุต เหล่านี้ได้ จากการวิเคราะห์ลักษณะการควบคุมของ เอาท์พุต ชนิดต่างๆ แล้วสามารถจำแนกชนิดของ เอาท์พุต ได้ 4 แบบด้วยกัน ดังนี้

1. เอาท์พุต ที่มีลักษณะการควบคุมแบบเปิดปิด กล่าวคือ เอาท์พุต เหล่านี้จะทำงานในลักษณะเปิดปิดคือมีเพียงแค่ 2 สถานะเช่น LED เมื่อมีกระแสไฟไหลผ่านก็จะส่องสว่างออกมา เมื่อไม่มีกระแสไฟไหลผ่านก็จะไม่ส่องสว่าง (แรงดันที่เปลี่ยนไปก็มีผลแต่มักไม่นำมาใช้ในการสื่อความหมาย)
2. เอาท์พุตที่มีลักษณะการควบคุมโดยใช้แรงดันกล่าวคือเอาท์พุตเหล่านี้จะมีลักษณะการทำงานเปลี่ยนแปลงไปตามแรงดันที่ได้รับ ยกตัวอย่างเช่น ดีซีมอเตอร์ จะมีการเปลี่ยนแปลงความเร็วตามแรงดันที่ได้รับ
3. เอาท์พุตที่มีลักษณะการควบคุมโดยใช้ขนาดความยาวของพัลส์กล่าวคือ เอาท์พุตเหล่านี้จะมีลักษณะการทำงานเปลี่ยนไปตามช่วงเวลาที่ได้รับพัลส์เข้ามาหรือเรียกอีกอย่างว่าเปลี่ยนแปลงตามขนาดของพัลส์หรือเปลี่ยนแปลงตามระยะเวลา ยกตัวอย่างเช่น เซอร์โวมอเตอร์ เมื่อมีการจ่ายพัลส์ขนาด 1.25 มิลลิวินาทีตำแหน่งของโรเตอร์จะอยู่ที่ 0 องศา
4. เอาท์พุตที่มีลักษณะการควบคุมโดยใช้บิตหลาย ๆ บิตควบคุมกล่าวคือ เอาท์พุตเหล่านี้จะต้องใช้บิตในการควบคุมหลายบิตเพื่อให้เอาท์พุตเหล่านี้ทำงานตามที่ต้องการ ยกตัวอย่างเช่น สเตปปิ้งมอเตอร์ จะเป็นการส่งบิตควบคุม 4 บิต เพื่อควบคุมการหมุนของโรเตอร์ภายในมอเตอร์

ในส่วนของโมดูล เอาท์พุตจึงแบ่งได้เป็น 2 ส่วนย่อย ๆ คือ

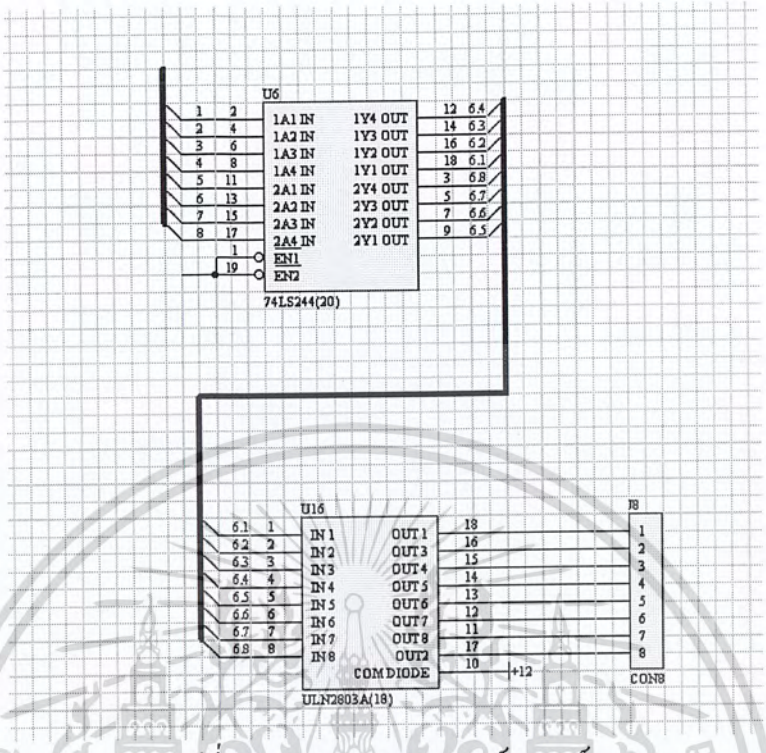
1. ส่วนที่ใช้สัญญาณบิตทั่วไปในการควบคุมเอาท์พุต ซึ่งจะสามารถควบคุมการทำงานของอุปกรณ์เอาท์พุตได้ถึง 3 ลักษณะคือ ส่วนที่ควบคุมแบบลักษณะปิดเปิด,ส่วนควบคุมด้วยสัญญาณควบคุมหลายบิต และส่วนควบคุมด้วยขนาดของพัลส์โดยจะใช้สัญญาณที่ได้จาก ไมโครโรบอท โดยตรงหรือผ่านส่วนที่ปรับแรงดันเพื่อให้ตรงกับความต้องการของอุปกรณ์ที่ต่อ
2. ส่วนที่ใช้สัญญาณแรงดันในการควบคุมเอาท์พุต ซึ่งจะสามารถควบคุมการทำงานของอุปกรณ์เอาท์พุตแบบควบคุมด้วยแรงดันได้ สัญญาณควบคุมด้วยแรงดันนี้จะได้มาจากการแปรงสัญญาณดิจิตอลที่ออกมาจากตัวไมโครโรบอตจากที่มีลักษณะเป็นหลายบิต (8 บิต) ให้ออกมาเป็นแรงดันค่าต่าง ๆ ซึ่งแปรผันตามค่าดิจิตอลที่ออกมาจากโมโครโรบอทเราเรียกโมดูลนี้ว่า ดิจิตอลทูอนาล็อกโมดูล

เนื่องจากในส่วนโมดูลเอาท์พุตเหล่านี้เมื่อได้รับค่าจากไมโครคอนโทรลเลอร์แล้วกระแสที่ไหลออกมานั้นไม่พอสำหรับการควบคุมอุปกรณ์เอาท์พุตต่างๆ ดังนั้นจึงได้มีการใส่วงจรเพิ่มกระแสเข้าไปเพื่อสามารถใช้งานกับเอาท์พุตต่างๆ ได้โดยมีวงจรเพิ่มกระแสดังรูป



รูปที่ 4.5 วงจรเพิ่มกระแส

เนื่องจากอุปกรณ์ที่ได้ออกแบบไว้ให้สามารถใช้ได้เป็นมอเตอร์ดังนั้นจึงไม่สามารถต่อเข้าตรง ๆ กับพอร์ตที่เป็นเอาต์พุตได้เนื่องจากว่าอุปกรณ์เหล่านี้ต้องการกระแสไฟฟ้าในการทำงานมากกว่าที่บัฟเฟอร์ (74LS244) สามารถให้ได้ดังนั้นจึงจำเป็นต้องต่อวงจรขยายกระแสไฟฟ้าเพิ่มให้กับพอร์ตเอาต์พุต โดยโครงงานนี้ได้เลือกไอซีเบอร์ ULN2803 ในการช่วยขยายกระแสไฟฟ้าโดยไอซี ULN 2803 นี้สามารถรองรับการใช้งานกระแสได้ 0.5 แอมป์ซึ่งเพียงพอสำหรับขับมอเตอร์ที่เลือกใช้กับโครงงานนี้ได้ ดังนั้นแต่ละพอร์ตเอาต์พุตของโครงงานนี้จึงสามารถใช้กระแสไฟฟ้าได้ 0.5 แอมป์ต่อหนึ่งพอร์ตการออกแบบสามารถทำได้ดังรูปโดยในโครงงานนี้มีพอร์ตเอาต์พุตทั้งหมด 10 พอร์ต



รูปที่ 4.6 แสดงการออกแบบเอาต์พุตพอร์ต

4.5 การออกแบบวงจร อินพุตพอร์ต

โมดูลอินพุตนี้จะเป็นส่วนที่ให้ผู้ใช้ต่ออุปกรณ์เซนเซอร์ชนิดต่าง ๆ เข้ามาเพื่อรับค่าจากสภาพแวดล้อมภายนอกอาทิเช่น อุณหภูมิ แสง แรงดัน และสามารถนำเข้ามาประมวลผลภายในไมโครโรบอท และแสดงค่าออกทางส่วนเอาท์พุตตามที่ผู้ใช้ได้กำหนดไว้จากโปรแกรมอินพุต ส่วนใหญ่นั้นจะเป็นเซนเซอร์ชนิดต่าง ๆ เช่น เซนเซอร์อินฟราเรด (Infrared sensor), เซนเซอร์อุลตร้าโซนิค (Ultrasonic sensor) หรือ เซนเซอร์เทอร์โม (Thermo sensor) จากการวิเคราะห์แล้วพบว่าตัวเซนเซอร์แบ่งออกเป็น 2 แบบด้วยกัน คือ

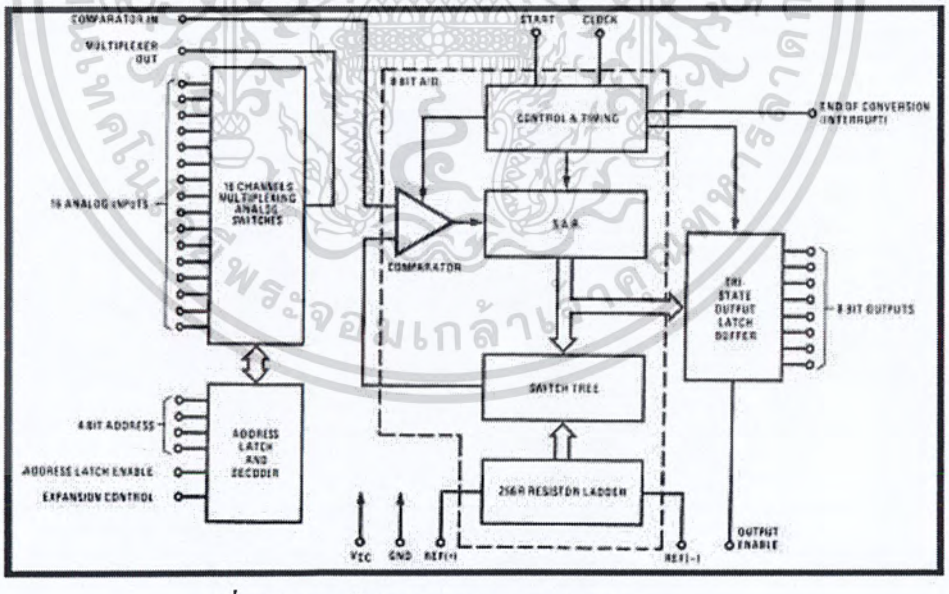
1. เซนเซอร์ที่มีลักษณะเป็น เซนเซอร์เปิดปิด กล่าวคือ เซนเซอร์ จะทำงานในสองสถานะคือ สถานะ active หรือสถานะปกติ จึงมีการส่งสัญญาณออกมาเป็นลักษณะ logic “0” หรือ logic “1” ยกตัวอย่าง เช่น infrared sensor
2. เซนเซอร์ที่มีลักษณะที่เปลี่ยนแปลงแรงดันตามสภาวะที่ตัว เซนเซอร์ ได้รับ คือมีแรงดันเปลี่ยนไป เช่น thermo sensor กล่าวคือ เมื่อมีอุณหภูมิสูงขึ้นตัว เซนเซอร์ ก็จะส่งค่าแรงดันเพิ่มขึ้นตามอุณหภูมิเหล่านั้น

ในส่วนของโมดูล เซนเซอร์ นี้จึงได้แบ่งออกเป็น 2 ส่วนย่อย ๆ คือ

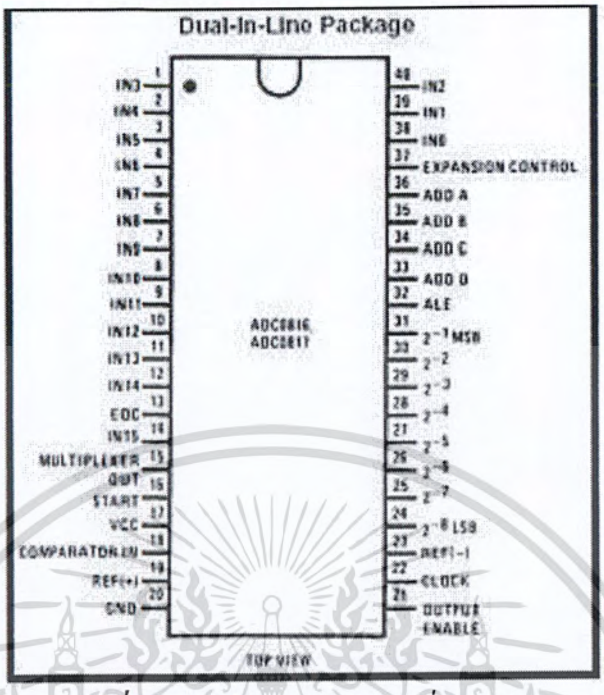
4.5.1 ส่วนที่ต้องผ่าน ตัวแปลงแรงดันไฟฟ้าเป็นค่า Digital (Analog to Digital converter Module) โดย (ในส่วนของตัวแปลงแรงดันไฟฟ้านี้ไม่สามารถใช้งานได้ในทุก อินพุต คอนเนคเตอร์ แต่จะกำหนดไว้ในส่วนที่สามารถใช้งานได้เฉพาะ)

การแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

เนื่องจากในการใช้งานอุปกรณ์อินพุตบางชนิดจะมีการส่งค่าที่เปลี่ยนไปของสภาพแวดล้อมออกมาเป็นลักษณะของแรงดันไฟฟ้าที่แตกต่างกัน ดังนั้นในวงจรจึงต้องมีโมดูลที่แปลงแรงดันไฟฟ้าที่แตกต่างนี้ออกมาเป็นข้อมูลในลักษณะดิจิตอลเพื่อนำข้อมูลนี้ไปใช้ในการประมวลผลต่อไปโดยโมดูลนี้จะสามารถใช้ได้เฉพาะบางส่วนของส่วนอินพุตเท่านั้นคือมีเพียง 8 ช่องสัญญาณสำหรับรับค่าที่เป็นอนาล็อค และแปลงมาเป็นดิจิตอล และเนื่องจากการทำงานที่เน้นความอเนกประสงค์ดังนั้นส่วนของไมโครโรบอทจะไม่สามารถรู้มาก่อนได้เลยว่าจะถูกใช้โมดูลนี้ที่ตำแหน่งใดของช่องสัญญาณทั้ง 8 ดังนั้นจึงได้เลือกชิพที่มี Multiplex 16 Channel อยู่ด้วยเพื่อใช้ในการเลือกช่องสัญญาณว่าช่องใดจะใช้โมดูลนี้โดยช่องสัญญาณนี้สามารถถูกใช้ในลักษณะรับสัญญาณแบบเปิดปิดได้อีกด้วย ADC0816 เป็นไอซีแปลงสัญญาณอนาล็อกไปเป็นสัญญาณดิจิตอลขนาด 8 บิต (256 ระดับ) ที่มี Multiplex 16 Channel รวมอยู่ด้วย ADC0816 มีบล็อกไดอะแกรมดังรูปที่ 4.7



รูปที่ 4.7 บลอกไดอะแกรมของ ADC0816



รูปที่ 4.8 รูปไอซี ADC0816 และชื่อขาต่างๆ

ADC0816 มีการทำงานของขาต่างๆ ดังนี้

- ขา IN 0 - 15 (1 – 12, 14, 39, 40) เป็นขาที่ใช้ในการส่งค่าสัญญาณอนาล็อกที่เราจะใช้ทั้งหมด (ต่อมาจากความต้านทานปรับค่าได้) เข้า โดยที่ค่าที่ได้จะออกมาทางขา Multiplex Out
- ขา Multiplex Out (15) เป็นค่าที่ทำหน้าส่งค่าสัญญาณอนาล็อกที่ต้องการใช้ออกมา
- ขา ADDRESS A, B, C, D (33 – 36) เป็นขาที่ทำหน้าที่เลือกสัญญาณว่าจะให้สัญญาณที่เข้ามาจาก IN 0 – 15 ว่า สัญญาณขาไหนที่จะถูกเลือกออกมาทางขา Multiplex Out
- ขา Expansion Control เป็นขาที่ทำหน้าที่คัดว่า จะให้ขา ADD A, B, C, D ทำงาน โดยมีตารางการทำงานดังตารางที่ 4.1
- ขา Comparator IN (18) เป็นขาที่ทำหน้าที่ รับค่าสัญญาณอนาล็อกที่ถูกเลือกมาจากขา IN (โดยจะรับค่าจากขา Multiplex Out) แล้วนำเข้าไปสู่กระบวนการแปลงสัญญาณเป็นสัญญาณดิจิตอล
- ขา Data Out (24 - 31) เป็นขาที่ให้ค่าที่แปลงแล้วจากสัญญาณอนาล็อก เป็นสัญญาณดิจิตอลออกมาทั้งหมด 8 บิต
- ขา START (16) เป็นขาที่รับค่าพัลล์ (Pulse) เริ่มต้น เพื่อทำการสั่งให้ไอซีทำการแปลงสัญญาณ โดยที่จะทำการแปลงสัญญาณ 1 ครั้งต่อ 1 พัลล์ที่เข้ามา
- ขา EOC (13) คือขาที่จะให้สัญญาณ 1 พัลล์ทุกครั้ง ที่แปลงสัญญาณเสร็จสิ้น
- ขา Clock (22) คือขาที่รับคล็อกไป เพื่อเป็นตัวให้จังหวะการทำงานแก่ไอซีตัวนี้

- ขา REF (+), REF (-) เป็นขาที่รับค่าแรงดันไฟฟ้าอ้างอิง เพื่อที่จะนำสัญญาณอนาล็อกที่ต้องการแปลงเป็นสัญญาณดิจิตอลมาเปรียบเทียบค่าแรงดันกับแรงดันอ้างอิงนี้ โดยจะนำค่าแรงดันสูงสุดกับค่าแรงดันต่ำสุดไปคำนวณหาค่าที่จะแปลงออกมา

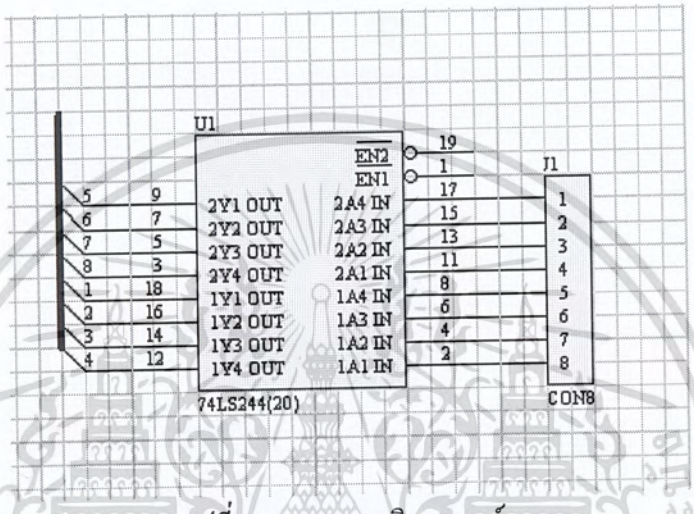
Selected Analog Channel	Address Line				Expansion Control
	A	B	C	D	
IN 0	L	L	L	L	H
IN 1	L	L	L	H	H
IN 2	L	L	H	L	H
IN 3	L	L	H	H	H
IN 4	L	H	L	L	H
IN 5	L	H	L	H	H
IN 6	L	H	H	L	H
IN 7	L	H	H	H	H
IN 8	H	L	L	L	H
IN 9	H	L	L	H	H
IN 10	H	L	H	L	H
IN 11	H	L	H	H	H
IN 12	H	H	L	L	H
IN 13	H	H	L	H	H
IN 14	H	H	H	L	H
IN 15	H	H	H	H	H
All Channels OFF	X	X	X	X	L

ตารางที่ 4.1 แสดงขาของ ADC0816

4.5.2 ส่วนที่สามารถนำค่าไปใช้ได้เลยหรือมีการปรับค่าแรงดันให้ตรงกับความต้องการเท่านั้นไม่ต้องทำการแปลงค่าจากอนาลอกเป็น ดิจิตอล

เนื่องจากอุปกรณ์อินพุตที่จะนำมาต่อกับโครงงานนี้จำเป็นต้องมีอุปกรณ์พิเศษที่ใช้จึงไม่นำอุปกรณ์พิเศษเหล่านี้เข้ามาอยู่ในวงจรเพื่อความสะดวกในการออกแบบและเพื่อลดขนาดของวงจรและการใช้งานจะไม่ซ้ำซ้อนจึงได้กำหนดให้ผู้ใช้จำเป็นจะต้องต่อวงจรพิเศษนี้ขึ้นเองเพื่อใช้งานอุปกรณ์อินพุตเหล่านั้นกับ

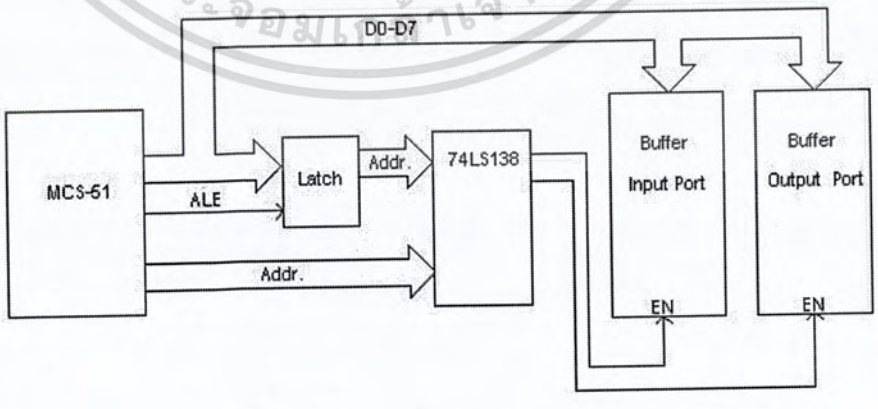
วงจร ยกตัวอย่างเช่น ในการใช้งานอินฟราเรดเซนเซอร์ก็ต้องต่อวงจรเปรียบเทียบระดับแรงดันเองแล้วส่งสัญญาณที่มีแรงดันอยู่ในช่วง ทีทีแอลกลับมาให้กับวงจร หรือในส่วนของอุปกรณ์อินพุตชนิดอุลตร้าโซนิก ซึ่งตัวส่งจะต้องได้รับความถี่ขนาด 40 เมกกะเฮิร์ซผู้ใช้ก็จะต้องต่อวงจรในส่วนนี้เองจึงจะสามารถนำมาใช้กับโครงงานนี้ได้ ในส่วนของวงจรของโครงงานนี้จะมีวงจรบัฟเฟอร์เพื่อรับค่าของอินพุตที่มีสัญญาณอยู่ในระดับ ทีทีแอลแล้วเท่านั้นโดยจะมีพอร์ตอินพุตให้ทั้งหมด 10 พอร์ต



รูปที่ 4.9 แสดงวงจรอินพุตพอร์ต

ส่วนของการเชื่อมต่อกับอุปกรณ์อินพุต/เอาท์พุตภายนอก (Device Module)

การเชื่อมต่อกับอุปกรณ์ภายนอก จะเชื่อมต่อผ่านไอซีบัฟเฟอร์เบอร์ 74LS373 โดยจะมีการถอดรหัสแอดเดรส โดยใช้ไอซี 74LS138 ซึ่งจะใช้สัญญาณที่ได้จากการถอดรหัส เป็นตัวเอนนาเบิลไอซีบัฟเฟอร์เพื่อให้บัฟเฟอร์ตัวนั้นทำงาน โมดูลการเชื่อมต่อสามารถแสดงได้ดังรูป 4.9



รูปที่ 4.10 แสดงการเชื่อมต่อของโมดูลการเชื่อมต่อกับอุปกรณ์อินพุต/เอาท์พุตภายนอก

บทที่5
การสร้างและการออกแบบในส่วนซอฟท์แวร์

ซอฟแวร์ในที่นี้หมายถึงส่วนของโปรแกรมที่ใช้ควบคุมการทำงานของไมโครโรบอทซึ่งเป็นหัวใจของการทำงานของไมโครโรบอท ซอฟแวร์ภายในโครงงานนี้สามารถแบ่งออกได้เป็น 2 ส่วนหลักด้วยกันคือ ซอฟแวร์ที่ผู้ใช้ไมโครโรบอทเขียนขึ้น และซอฟแวร์ที่เป็นโปรแกรมสำเร็จรูป

5.1 ซอฟแวร์ที่ผู้ใช้ไมโครโรบอทเขียนขึ้น

ซอฟแวร์ที่ผู้ใช้เขียนขึ้นนี้ใช้เพื่อกำหนดการทำงานของตัวไมโครโรบอทโดยขึ้นอยู่กับ เซนเซอร์ ที่ได้รับเข้ามาว่าตรงตามความต้องการหรือยังและจะให้ออกเอาท์พุตเป็นค่าอะไรและที่เอาท์พุตตัวใดในส่วนซอฟแวร์นี้จะเป็นซอฟแวร์ที่เขียนขึ้นบนส่วนของโปรแกรมที่อยู่บนคอมพิวเตอร์ส่วนบุคคลซึ่งทางผู้สร้างโครงงานเขียนขึ้นเพื่ออำนวยความสะดวกให้ผู้ใช้ ตัวโปรแกรมที่อยู่บนคอมพิวเตอร์ส่วนบุคคลจะแปลงข้อมูลในลักษณะที่สามารถใช้งานกับไมโครโรบอทได้และจะทำการดาวน์โหลดข้อมูลเหล่านี้ลงบนไมโครโรบอทเพื่อใช้ในการควบคุมต่อไป

5.2 ซอฟแวร์ที่เป็นโปรแกรมสำเร็จรูป

ซอฟแวร์โปรแกรมสำเร็จรูปมีหน้าที่อำนวยความสะดวกให้ผู้ใช้โดยลดภาระการเขียนโปรแกรมของผู้ใช้ทั้งในส่วนที่ต้องเขียนบนคอมพิวเตอร์ส่วนบุคคลและในส่วนที่จะต้องเขียนลงไมโครโรบอทโดยซอฟแวร์ที่เป็นโปรแกรมสำเร็จสำเร็จรูปนี้จะแบ่งออกเป็น 2 ส่วนคือ

5.2.1 ซอฟแวร์สำเร็จรูปที่อยู่บนคอมพิวเตอร์ส่วนบุคคล

ซอฟแวร์สำเร็จรูปที่อยู่บนคอมพิวเตอร์ส่วนบุคคลมีหน้าที่ติดต่อกับผู้ใช้โดย ผู้ใช้ต้องทำการเขียนโปรแกรมเป็นภาษา ASM-51 คอมไพล์และ ดาวน์โหลด ผ่านทางซอฟแวร์ตัวนี้ ซึ่งก็คือโปรแกรม CFM

โปรแกรม CFM นี้ได้ใช้โปรแกรม Delphi6 พัฒนาขึ้นมา จะเป็นส่วนของการเชื่อมต่อกับผู้ใช้การทำงานสามารถแบ่งได้เป็น 3 ส่วนหลัก ๆ คือ

1. Write ASM Code จะเป็นการเขียน Assembly code ส่วนนี้จะให้ User เป็นคนเขียนโปรแกรมควบคุมการทำงานของไมโครโรบอท ตามที่ต้องการ โดยที่ User ไม่จำเป็นต้องเขียน Code ขึ้นมาเองทั้งหมด แต่สามารถเรียกใช้ Procedure ที่มีไว้ให้ได้ แต่ต้องกำหนดค่าต่าง ๆ ตามที่โปรแกรมกำหนดไว้
2. Compile ในส่วนของโปรแกรมที่จะนำมาคอมไพล์ Assembly code นั้นจะใช้ โปรแกรม SXA51 แต่ User ไม่ต้องไปเรียกใช้โปรแกรม SXA51 โดยตรง สามารถเรียกผ่านโปรแกรม CFM ได้

3. Download จะเป็นการดาวน์โหลดไฟล์ .HEX ที่ได้จากการ Compile ไปใส่ไว้ในตัวไมโครโรบอท เพื่อให้ไมโครโรบอททำงานตามที่ต้องการ ในการดาวน์โหลดนี้จะต้องมีการกำหนดค่าพอร์ทที่จะใช้ , Baudrate และ ค่าต่าง ๆ นอกจากนั้นจะต้องมีการเปิดไฟล์ที่จะใช้ดาวน์โหลดลงไปอีกด้วย

Source Code

- โปรแกรม CFM

  เป็นส่วนหลักของโปรแกรมทั้งหมด จะเป็นตัวเชื่อมต่อการทำงานของแต่ละส่วน

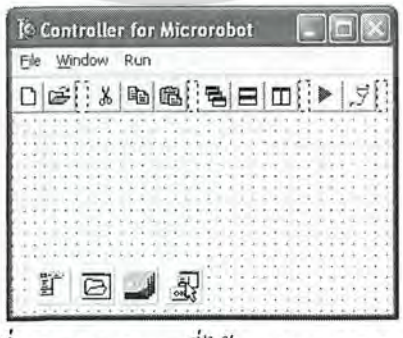
```
program CFM;
uses
  Forms,
  MDIFrame in 'C:\CFM\Delphi\MDIFrame.pas' {FrameForm},
  MDIEdit in 'C:\CFM\Delphi\MDIEdit.pas' {EditForm},
  Open_ASM in 'C:\CFM\Delphi\Open_ASM.pas' {Form1},
  MnForm in 'C:\CFM\Delphi\MnForm.pas' {DlForm},
  SettingsDlg in 'C:\CFM\Delphi\SettingsDlg.pas' {SettingsForm},
  FmxUtils in 'C:\CFM\Delphi\FmxUtils.pas',
  AboutTTY in 'C:\CFM\Delphi\AboutTTY.pas' {AboutBoxForm};
{$R *.RES}
```

จากโค้ดด้านบนจะเห็นว่า ส่วนแรกจะเป็นการกำหนดว่าจะให้ไปเรียกแต่ละส่วนของโปรแกรมมาจากไดเรคทอรี่ไหน

```
begin
  Application.Initialize;
  Application.CreateForm(TFrameForm, FrameForm);
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TDlForm, DlForm);
  Application.CreateForm(TAboutBoxForm, AboutBoxForm);
  Application.Run;
end.
```

หลังจากนั้นจะเป็นการสั่งให้สร้าง (Create) แต่ละส่วนขึ้นมา

- Unit MDIFrame จะเป็นหน้าต่างแรกที่เปิดขึ้นมาเมื่อเราทำการรันโปรแกรม CFM



รูปที่ 5.1 แสดง Form ที่ได้จาก Unit MDIFrame

```
unit MDIFrame;
interface
uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ToolWin, ComCtrls, StdActns, ActnList, ImgList;
```

โค้ดด้านบนจะเป็นการกำหนดว่า Unit นี้เรียกใช้ Component ใดบ้าง

```
type
  TFrameForm = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    Window1: TMenuItem;
    Tile1: TMenuItem;
    Cascade1: TMenuItem;
    Arrangeicons1: TMenuItem;
    OpenFileDialog: TOpenDialog;
    ImageList1: TImageList;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ActionList1: TActionList;
    FileNew1: TAction;
    FileOpen1: TAction;
    FileClose1: TWindowClose;
    FileSave1: TAction;
    FileSaveAs1: TAction;
    FileExit1: TAction;
    EditCut1: TEditCut;
    EditCopy1: TEditCopy;
    EditPaste1: TEditPaste;
    WindowCascade1: TWindowCascade;
    WindowTileHorizontal1: TWindowTileHorizontal;
    WindowTileVertical1: TWindowTileVertical;
    WindowMinimizeAll1: TWindowMinimizeAll;
    WindowArrangeAll1: TWindowArrange;
    HelpAbout1: TAction;
    ToolButton2: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
    ToolButton7: TToolButton;
    ToolButton8: TToolButton;
    ToolButton9: TToolButton;
    ToolButton10: TToolButton;
    ToolButton11: TToolButton;
    ToolButton12: TToolButton;
    ToolButton13: TToolButton;
    ToolButton14: TToolButton;
    Run1: TMenuItem;
    Compile1: TMenuItem;
    Download1: TMenuItem;
    ToolButton15: TToolButton;
    procedure Exit1Click(Sender: TObject);
    procedure New1Click(Sender: TObject);
    procedure Tile1Click(Sender: TObject);
    procedure Cascade1Click(Sender: TObject);
    procedure Arrangeicons1Click(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure ToolButton13Click(Sender: TObject);
    procedure ToolButton14Click(Sender: TObject);
    procedure Compile1Click(Sender: TObject);
    procedure Download1Click(Sender: TObject);
  private
    { Private declarations }
```

```
 public
   { Public declarations }
 end;
```

โค้ดด้านบนจะเป็นการกำหนดลักษณะของแต่ละไอเทมที่อยู่บนฟอร์ม รวมถึงกำหนด Procedure ต่าง ๆ ที่มีใน Unit นี้

```
var
  FrameForm: TFrameForm;
implementation
uses MDIEdit, Open_ASM, MnForm;
{$R *.dfm}
procedure TFrameForm.Exit1Click(Sender: TObject);
begin
  Close;
end;
```

เมื่อมีการกดปุ่ม  ให้ทำการปิดหน้าต่าง CFM

```
procedure TFrameForm.New1Click(Sender: TObject);
begin
  TEditForm.Create(Self);
end;
```

เมื่อมีการกดปุ่ม ให้ทำการสร้าง EditForm ขึ้นมา

```
procedure TFrameForm.Tile1Click(Sender: TObject);
begin
  Tile;
end;
procedure TFrameForm.Cascade1Click(Sender: TObject);
begin
  Cascade;
end;
procedure TFrameForm.Arrangeicons1Click(Sender: TObject);
begin
  ArrangeIcons;
end;
```

เมื่อมีการกดปุ่ม Arrange หน้าต่าง ให้ทำการเรียงหน้าต่างตามปุ่มที่กด

```
procedure TFrameForm.Open1Click(Sender: TObject);
begin
  if OpenFileDialog.Execute then
  with TEditForm.Create(Self) do
    Open(OpenFileDialog.FileName);
end;
```

เมื่อมีการกดปุ่ม  ให้ทำการเปิด OpenDialog

```
procedure TFrameForm.ToolButton13Click(Sender: TObject);
begin
      Form1.showmodal;
end;
```

เมื่อมีการกดปุ่ม  ให้ทำการเปิดหน้าต่าง Form1

```
procedure TFrameForm.ToolButton14Click(Sender: TObject);
begin
      Dlform.showmodal;
end;
```

เมื่อมีการกดปุ่ม  ให้ทำการเปิดหน้าต่าง Dlform

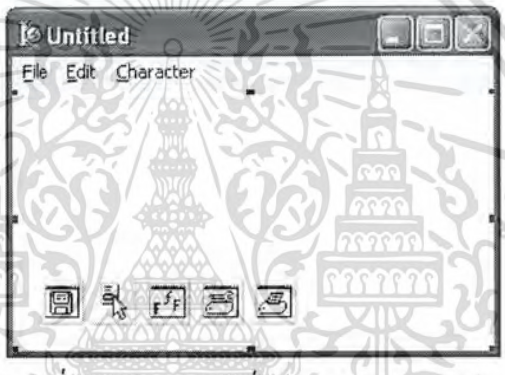
```
procedure TFrameForm.Compile1Click(Sender: TObject);
begin
     form1.ShowModal;
end;
```

เมื่อมีการกดปุ่ม Compile (ที่เมนู Run) ให้ทำการเปิดหน้าต่าง Form1

```
procedure TFrameForm.Download1Click(Sender: TObject);
begin
      dlform.ShowModal;
end;
```

เมื่อมีการกดปุ่ม Download (ที่เมนู Run) ให้ทำการเปิดหน้าต่าง Dlform

- Unit MDIEdit เป็นหน้าต่างที่ถูกเปิดขึ้นเมื่อมีการสร้างไฟล์ใหม่ หรือเปิดไฟล์ .ASM



รูปที่ 5.2 แสดง Form ที่ได้จาก Unit MDIEdit

```
unit MDIEdit;
interface
uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ComCtrls;
```

โค้ดด้านบนจะเป็นการกำหนดว่า Unit นี้เรียกใช้ Component ใดบ้าง

```
type
  TEditForm = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    Saveas1: TMenuItem;
    Print1: TMenuItem;
    N2: TMenuItem;
    Exit1: TMenuItem;
    Edit1: TMenuItem;
    Cut1: TMenuItem;
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    Delete1: TMenuItem;
    N3: TMenuItem;
    Selectall1: TMenuItem;
    Character1: TMenuItem;
    Left1: TMenuItem;
    Right1: TMenuItem;
```

```
    Center1: TMenuItem;
    N4: TMenuItem;
    Wordwrap1: TMenuItem;
    N5: TMenuItem;
    Font1: TMenuItem;
    Editor: TRichEdit;
    PopupMenu1: TPopupMenu;
    Cut2: TMenuItem;
    Copy2: TMenuItem;
    Paste2: TMenuItem;
    SaveFileDialog: TSaveDialog;
    FontDialog1: TFontDialog;
    Printersetup1: TMenuItem;
    Close1: TMenuItem;
    PrinterSetupDialog1: TPrinterSetupDialog;
    PrintDialog1: TPrintDialog;
    procedure Exit1Click(Sender: TObject);
    procedure New1Click(Sender: TObject);
    procedure AlignClick(Sender: TObject);
    procedure Wordwrap1Click(Sender: TObject);
    procedure Cut1Click(Sender: TObject);
    procedure Copy1Click(Sender: TObject);
    procedure Paste1Click(Sender: TObject);
    procedure SelectAll1Click(Sender: TObject);
    procedure Delete1Click(Sender: TObject);
    procedure Edit1Click(Sender: TObject);
    procedure Saveas1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Font1Click(Sender: TObject);
    procedure Close1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure FormCreate(Sender: TObject);
    procedure Printersetup1Click(Sender: TObject);
    procedure Print1Click(Sender: TObject);
    procedure Open1Click(Sender: TObject);
  private
    { Private declarations }
    PathName: string;
  public
    { Public declarations }
    procedure Open(const AFileName: string);
  end;
```

โค้ดด้านบนจะเป็นการกำหนดลักษณะของแต่ละไอเทมที่อยู่บนฟอร์ม รวมถึงกำหนด Procedure ต่าง ๆ ที่มีใน Unit นี้

```
var
  EditForm: TEditForm;
const
  DefaultFileName = 'Untitled';
implementation
uses Clipbrd, Printers, MDIFrame;
{$R *.dfm}
procedure TEditForm.Exit1Click(Sender: TObject);
begin
  FrameForm.Exit1Click(Sender);
end;
```

เมื่อมีการกดปุ่ม  ให้ทำการปิดหน้าต่าง CFM

```
procedure TEditForm.New1Click(Sender: TObject);
begin
  FrameForm.New1Click(Sender);
end;
```

เมื่อมีการกดปุ่ม  ให้ทำการสร้าง EditForm ขึ้นมา

```
procedure TEditForm.Open1Click(Sender: TObject);
begin
  FrameForm.Open1Click(Sender);
end;
```

เมื่อมีการกดปุ่ม 🗁 ให้ทำการเปิด OpenDialog

```
procedure TEditForm.AlignClick(Sender: TObject);
begin
  Left1.Checked := False;
  Right1.Checked := False;
  Center1.Checked := False;
  with Sender as TMenuItem do Checked := True;
  with Editor.Paragraph do
    if Left1.Checked then
      Alignment := taLeftJustify
    else if Right1.Checked then
      Alignment := taRightJustify
    else if Center1.Checked then
      Alignment := taCenter;
end;
```

เมื่อมีการกดปุ่ม Left, Right, Center บนเมนู Character ให้ทำการจัดตัวหนังสือตามคำสั่ง เช่น เมื่อคลิ๊ก Left ให้จัดตัวอักษรให้ชิดซ้าย เป็นต้น

```
procedure TEditForm.Wordwrap1Click(Sender: TObject);
begin
  with Editor do
  begin
    WordWrap := not WordWrap; { toggle word wrapping }
    if WordWrap then
      ScrollBars := ssVertical
    else
      ScrollBars := ssBoth;
    WordWrap1.Checked := WordWrap; { set menu item check }
  end;
end;
```

เมื่อมีการกดปุ่ม WordWrap บนเมนู Character ให้ทำการ WordWrap ไฟล์นั้น

```
procedure TEditForm.Cut1Click(Sender: TObject);
begin
  Editor.CutToClipboard;
end;
```

เมื่อมีการกดปุ่ม Cut ให้ทำการตัดข้อมูลที่ Highlight ไว้

```
procedure TEditForm.Copy1Click(Sender: TObject);
begin
  Editor.CopyToClipboard;
end;
```

เมื่อมีการกดปุ่ม Copy ให้ทำการคัดลอกข้อมูลที่ Highlight ไว้

```
procedure TEditForm.Paste1Click(Sender: TObject);
begin
  Editor.PasteFromClipboard;
end;
```

เมื่อมีการกดปุ่ม Paste ให้ทำการวางข้อมูลที่ Highlight ไว้

```
procedure TEditForm.Selectall1Click(Sender: TObject);
```

```
begin
  Editor.SelectAll;
end;
```

เมื่อมีการกดปุ่ม SelectAll ให้ทำการ Highlight ข้อมูลทั้งหมด

```
procedure TEditForm.Delete1Click(Sender: TObject);
begin
  Editor.ClearSelection;
end;
```

เมื่อมีการกดปุ่ม Delete ให้ทำการลบข้อมูลที่ Highlight ไว้

```
procedure TEditForm.Edit1Click(Sender: TObject);
var
  HasSelection: Boolean;
begin
  Paste1.Enabled := Clipboard.HasFormat(CF_TEXT);
  Paste2.Enabled := Paste1.Enabled;
  HasSelection := Editor.SelLength > 0;
  Cut1.Enabled := HasSelection;
  Cut2.Enabled := HasSelection;
  Copy1.Enabled := HasSelection;
  Copy2.Enabled := HasSelection;
  Delete1.Enabled := HasSelection;
end;
```

จะเป็นการกำหนดค่าเริ่มต้นให้ตัวแปรต่าง ๆ เมื่อมีการกดปุ่ม Edit

```
procedure TEditForm.Open(const AFileName: string);
begin
  PathName := AFileName;
  Caption := ExtractFileName(AFileName);
  with Editor do
  begin
    Lines.LoadFromFile(PathName);
    SelStart := 0;
    Modified := False;
  end;
end;
```

Procedure นี้จะทำการเปิดไฟล์โดยส่ง AfileName และ PathName ไปเป็นพารามิเตอร์

```
procedure TEditForm.Saveas1Click(Sender: TObject);
begin
  SaveFileDialog.FileName := PathName;
  if SaveFileDialog.Execute then
  begin
    PathName := SaveFileDialog.FileName;
    Caption := ExtractFileName(PathName);
    Save1Click(Sender);
  end;
end;
```

เมื่อมีการกดปุ่ม Saveas จะทำการเปิด SaveDialog และนำข้อมูลไปเก็บไว้ที่ไฟล์ที่ถูกเลือก

```
procedure TEditForm.Save1Click(Sender: TObject);
begin
  if PathName = DefaultFileName then
    SaveAs1Click(Sender)
  else
  begin
    Editor.Lines.SaveToFile(PathName);
    Editor.Modified := False;
  end;
end;
```

เมื่อมีการกดปุ่ม Save จะทำการบันทึกข้อมูล

```
procedure TEditForm.Font1Click(Sender: TObject);
begin
  FontDialog1.Font := Editor.Font;
  if FontDialog1.Execute then
    Editor.SelAttributes.Assign(FontDialog1.Font);
end;
```

เมื่อมีการกดปุ่ม Font จะทำการเปิด FontDialog เพื่อให้ผู้ใช้แก้ไขฟอนท์ตามที่ต้องการ

```
procedure TEditForm.Close1Click(Sender: TObject);
begin
  Close;
end;
```

เมื่อมีการกดปุ่ม Close จะทำการปิดหน้าต่างนี้

```
procedure TEditForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
end;
procedure TEditForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
const
  SWarningText = 'Save changes to %s?';
begin
  if Editor.Modified then
  begin
    case MessageDlg(Format(SWarningText, [PathName]), mtConfirmation,
      [mbYes, mbNo, mbCancel], 0) of
      idYes: Save1Click(Self);
      idCancel: CanClose := False;
    end;
  end;
end;
```

เมื่อมีการกดปุ่ม Close จะทำการถามว่าต้องการจะ Save Change รึเปล่า

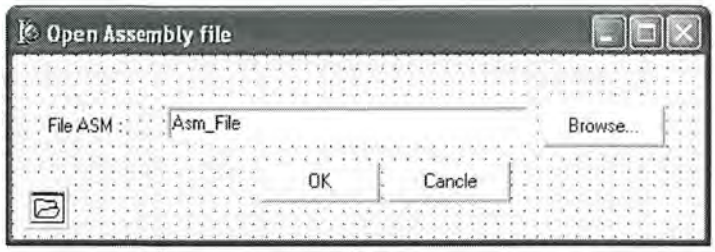
```
procedure TEditForm.Printersetup1Click(Sender: TObject);
begin
  PrinterSetupDialog1.Execute;
end;
```

เมื่อมีการกดปุ่ม Printersetup จะทำการเปิดหน้าต่าง PrinterSetupDialog

```
procedure TEditForm.Print1Click(Sender: TObject);
begin
  if PrintDialog1.Execute then
    Editor.Print(PathName);
end;
end.
```

เมื่อมีการกดปุ่ม Print จะทำการเปิดหน้าต่าง PrintDialog

- Unit Open_ASM เป็นหน้าต่างที่ถูกเปิดขึ้นเมื่อต้องการจะ Compile ไฟล์ .ASM



รูปที่ 5.3 แสดง Form ที่ได้จาก Unit Open_ASM

```
unit Open_ASM;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls,Fmxutils;
```

โค้ดด้านบนจะเป็นการกำหนดว่า Unit นี้เรียกใช้ Component ใดบ้าง

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Asm_File: TEdit;
    OpenDialog1: TOpenDialog;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

โค้ดด้านบนจะเป็นการกำหนดลักษณะของแต่ละไอเทมที่อยู่บนฟอร์ม

```
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
      Opendialog1.execute;
      Form1.Asm_File.text := Opendialog1.FileName;
end;
```

เมื่อมีการกดปุ่ม Browse จะทำการเปิดหน้าต่าง OpenDialog

```
procedure TForm1.Button2Click(Sender: TObject);
begin
      fmxutils.ExecuteFile('sxa51.exe',Form1.Asm_File.text,'C:\CFM\sxa51',1);
      close;
end;
```

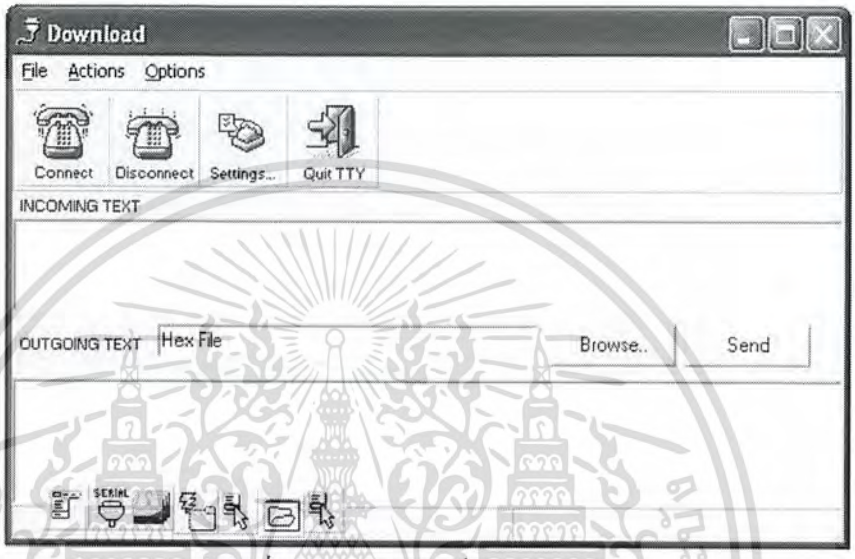
เมื่อมีการกดปุ่ม OK จะทำการส่งไฟล์ที่เลือกไป Compile โดยจะเรียกใช้โปรแกรม SXA51

```
procedure TForm1.Button3Click(Sender: TObject);
begin
      close;
```

```
end;
end.
```

เมื่อมีการกดปุ่ม Close จะทำการปิดหน้าต่างนี้

- Unit MnForm จะเป็นหน้าต่างที่ใช้ในการดาวน์โหลดไฟล์ .HEX ไปยังส่วน Hardware



รูปที่ 5.4 แสดง Form ที่ได้จาก Unit MnForm

```
unit MnForm;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ComCtrls, ExtCtrls, StdCtrls, ToolWin, ImgList, ClipBrd,
  CPDrv, AboutTTY, SettingsDlg, Psock, NMFtp;
```

โค้ดด้านบนจะเป็นการกำหนดว่า Unit นี้เรียกใช้ Component ใดบ้าง

```
type
  TDlForm = class(TForm)
    cpDrv: TCommPortDriver;
    MainMenu: TMainMenu;
    FileMenu: TMenuItem;
    OptionsMenu: TMenuItem;
    Splitter1: TSplitter;
    RXPanel: TPanel;
    IncomingRichEdit: TRichEdit;
    Panel2: TPanel;
    TXPanel: TPanel;
    Panel3: TPanel;
    ToolBar1: TToolBar;
    ConnectToolButton: TToolButton;
    DisconnectToolButton: TToolButton;
    SettingsToolButton: TToolButton;
    E_ImageList: TImageList;
    QuitTTYToolButton: TToolButton;
    SerialIOSettingsCmd: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    IT_PopupMenu: TPopupMenu;
```

```
    OT_PopupMenu: TPopupMenu;
    IT_ClearCmd: TMenuItem;
    OT_ClearCmd: TMenuItem;
    N1: TMenuItem;
    OT_CutCmd: TMenuItem;
    OT_CopyCmd: TMenuItem;
    N3: TMenuItem;
    IT_CopyCmd: TMenuItem;
    ActionsMenu: TMenuItem;
    ActionsConnectCmd: TMenuItem;
    ActionsDisconnectCmd: TMenuItem;
    FileQuitCmd: TMenuItem;
    OT_PasteCmd: TMenuItem;
    NMFTP1: TNMFTP;
    Panel1: TPanel;
    StatusPanel: TPanel;
    FrameSettingsPanel: TPanel;
    FlowSettingsPanel: TPanel;
    Button1: TButton;
    Edit1: TEdit;
    Button2: TButton;
    OpenDialog1: TOpenDialog;
    Memo1: TMemo;
    procedure SettingsToolButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ConnectToolButtonClick(Sender: TObject);
    procedure DisconnectToolButtonClick(Sender: TObject);
    procedure OutgoingRichEditKeyDown(Sender: TObject; var Key: Word;Shift:
TShiftState);
    procedure OT_ClearCmdClick(Sender: TObject);
    procedure OutgoingRichEditKeyPress(Sender: TObject; var Key: Char);
    procedure cpDrvReceiveData(Sender: TObject; DataPtr: Pointer;DataSize: Cardinal);
    procedure IT_ClearCmdClick(Sender: TObject);
    procedure QuitTTYToolButtonClick(Sender: TObject);
    procedure HelpAboutCmdClick(Sender: TObject);
    procedure OT_CutCmdClick(Sender: TObject);
    procedure OT_CopyCmdClick(Sender: TObject);
    procedure OT_PasteCmdClick(Sender: TObject);
    procedure IT_CopyCmdClick(Sender: TObject);
    procedure OT_PopupMenuPopup(Sender: TObject);
    procedure IT_PopupMenuPopup(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    // Startup about-box (splash screen)
    FAboutBox: TAboutBoxForm;
    FAboutBoxShownTime: DWORD;
    // Called when the message queue gets empty.
    procedure IdleProc( Sender: TObject; var Done: boolean );
    // Updates the panels on bottom of this window.
    procedure UpdateStatusPanels;
    // Displays an error box informing the user we can't send data
    procedure CannotSendError;
  public
  end;
```

โค้ดด้านบนจะเป็นการกำหนดลักษณะของแต่ละไอเทมที่อยู่บนฟอร์ม รวมถึงกำหนด Procedure ต่าง ๆ ที่มีใน Unit นี้

```
var
  DlForm: TDlForm;
implementation
{$R *.DFM}
// Lets the user to customize I/O settings
procedure TDlForm.SettingsToolButtonClick(Sender: TObject);
var dlg: TSettingsForm;
begin
  // Tell the user we cannot change settings while a connection is active
  if cpDrv.Connected then
```

```
 begin
   if Application.MessageBox( 'Could not change settings while a connection is
active.'#13#10+
                      'Close the connection and continue ?',
                      'Confirm',
                      MB_OKCANCEL or MB_ICONQUESTION ) <> ID_OK then
     exit;
   cpDrv.Disconnect;
 end;
 // Let the user to customize settings
 dlg := nil;
 try
   dlg := TSettingsForm.Create( Self, cpDrv );
   dlg.ShowModal;
 finally
   dlg.Free;
 end;
end;
```

เมื่อมีการกดปุ่ม Setting จะทำการตรวจสอบว่าขณะนั้นได้ทำการเชื่อมต่ออยู่หรือไม่ ถ้าทำการเชื่อมต่ออยู่ก็จะแสดงหน้าจอว่าไม่สามารถทำการ Setting ได้ ต้องทำการ Disconnect ก่อน แต่ถ้าไม่ได้ทำการเชื่อมต่ออยู่ก็จะเป็นหน้าต่าง SettingsForm

```
// Disconnect
procedure TDlForm.DisconnectToolButtonClick(Sender: TObject);
begin
  // Do nothing if not connected
  if not cpDrv.Connected then
    exit;
  // Disconnect
  cpDrv.Disconnect;
end;
```

เมื่อมีการกดปุ่ม Disconnect จะทำการปิดช่องทางการเชื่อมต่อ

```
procedure TDlForm.OT_ClearCmdClick(Sender: TObject);
begin
     Memo1.Lines.BeginUpdate;
     Memo1.Lines.Clear;
     Memo1.Lines.EndUpdate;
end;
```

เมื่อมีการกดปุ่ม ClearCmd จะทำการ Clear ค่าต่าง ๆ บนหน้าต่าง

```
// Displays an error box informing the user we can't send data
procedure TDlForm.CannotSendError;
begin
  if cpDrv.CheckLineStatus then
    Application.MessageBox( 'Could not send data.'#13#10+
                      'No device connected to serial port or device is off. Please, turn it
on.'#13#10 +
                      'Try setting Device Check to Off in the settings dialog box.'#13#10 +
                      'Also, replace your two wires serial cable with a full wires cable.',
                      'Warning',
                      MB_OK or MB_ICONINFORMATION )
  else
    Application.MessageBox( 'Could not send data.'#13#10+
                      'Please, check connections and try again.'#13#10 +
                      'Turn serial device on or, try setting Device Check to On in the settings
dialog box.'#13#10 +
                      'Also, disable Hardware Flow Control if you are using a two wires cable.',
                      'Warning',
                      MB_OK or MB_ICONINFORMATION )
end;
```

Procedure นี้จะทำการแสดง Error เมื่อไม่สามารถส่งข้อมูลออกไปได้

```
procedure TDlForm.OutgoingRichEditKeyPress(Sender: TObject;
  var Key: Char);
begin
  // Do nothing if not connected
  if not cpDrv.Connected then
    exit;
  // Send the character
  if not cpDrv.SendChar( Key ) then
    CannotSendError;
end;
// Handles incoming data
procedure TDlForm.cpDrvReceiveData(Sender: TObject; DataPtr: Pointer;
  DataSize: Cardinal);
var iLastLine, i: integer;
    s, ss: string;
begin
  // Convert incoming data into a string
  s := StringOfChar( ' ', DataSize );
  move( DataPtr^, pchar(s)^, DataSize );
  // Exit if s is empty. This usually occurs when one or more NULL characters
  // (chr(0)) are received.
  while pos( #0, s ) > 0 do
    delete( s, pos( #0, s ), 1 );
  if s = '' then
    exit;
  // Remove line feeds
  i := pos( #10, s );
  while i <> 0 do
  begin
    delete( s, i, 1 );
    i := pos( #10, s );
  end;
  // Don't redraw the rich edit control until we finished updating it
  //IncomingRichEdit.Lines.BeginUpdate;
  // Get last line index
  iLastLine := IncomingRichEdit.Lines.Count-1;
  // If the rich edit is empty...
  if iLastLine = -1 then
  begin
    // Remove line feeds from the string
    i := pos( #10, s );
    while i <> 0 do
    begin
      delete( s, i, 1 );
      i := pos( #10, s );
    end;
    // Remove carriage returns from the string (break lines)
    i := pos( #13, s );
    while i <> 0 do
    begin
      ss := copy( s, 1, i-1 );
      delete( s, 1, i );
      IncomingRichEdit.Lines.Append( ss );
      i := pos( #13, s );
    end;
    IncomingRichEdit.Lines.Append( s );
  end
  else
  begin
    // String to add is : last line added + new one
    s := IncomingRichEdit.Lines[iLastLine] + s;
    // Remove carriage returns (break lines)
    i := pos( #13, s );
    while i <> 0 do
    begin
      ss := copy( s, 1, i-1 );
      delete( s, 1, i );
      if iLastLine <> -1 then
```

```
    begin
      IncomingRichEdit.Lines[iLastLine] := ss;
      iLastLine := -1;
    end
    else
      IncomingRichEdit.Lines.Append( ss );
    i := pos( #13, s );
  end;
  if iLastLine <> -1 then
    IncomingRichEdit.Lines[iLastLine] := s
  else
    IncomingRichEdit.Lines.Append( s );
 end;
 //IncomingRichEdit.Lines.EndUpdate;
 // Scroll incoming text rich edit
 SendMessage( IncomingRichEdit.Handle, EM_SCROLLCARET, 0, 0 );
end;
```

Procedure นี้จะทำการรับข้อมูลมาจาก Controller และนำมาแสดงผลที่ Incoming Text

```
procedure TDlForm.IT_ClearCmdClick(Sender: TObject);
begin
 IncomingRichEdit.Lines.BeginUpdate;
 IncomingRichEdit.Lines.Clear;
 IncomingRichEdit.Lines.EndUpdate;
end;
```

เมื่อมีการกดปุ่ม ClearCmd จะทำการ Clear ค่าต่าง ๆ บนหน้าต่าง

```
// Quits TTY
procedure TDlForm.QuitTTYToolButtonClick(Sender: TObject);
begin
 PostQuitMessage( Handle );
end;
procedure TDlForm.HelpAboutCmdClick(Sender: TObject);
var dlg: TAboutBoxForm;
begin
 dlg := nil;
 try
  dlg := TAboutBoxForm.Create( Self, true );
  dlg.ShowModal;
 finally
  dlg.Free;
 end;
end;
```

เมื่อมีการกดปุ่ม Quit จะทำการปิดหน้าต่างนี้

```
procedure TDlForm.OT_CutCmdClick(Sender: TObject);
begin
 Memo1.CutToClipboard;
end;
```

เมื่อมีการกดปุ่ม Cut จะทำการตัดค่าที่ทำ Highlight ไว้ใน Memo1

```
procedure TDlForm.OT_CopyCmdClick(Sender: TObject);
begin
 Memo1.CopyToClipboard;
end;
```

เมื่อมีการกดปุ่ม Copy จะทำการคัดลอกค่าที่ทำ Highlight ไว้ใน Memo1

```
procedure TDlForm.OT_PasteCmdClick(Sender: TObject);
var clp: TClipboard;
  s, ss: string;
```

```
  iLastLine, i: integer;
begin
 // Get the clipboard object
 clp := Clipboard;
 // If the clipboard contains some text...
 if clp.HasFormat( CF_TEXT ) then
 begin
  // Automatically connect
  if not cpDrv.Connected then
  begin
   ConnectToolButtonClick( nil );
   if not cpDrv.Connected then
    exit;
  end;
  // Get the text
  s := clp.AsText;
  // Remove line feeds
  i := pos( #10, s );
  while i <> 0 do
  begin
   delete( s, i, 1 );
   i := pos( #10, s );
  end;
  // Add the text to the rich edit and send it
  iLastLine := Memo1.Lines.Count-1;
  i := pos( #13, s );
  while i <> 0 do
  begin
   ss := copy( s, 1, i-1 );
   delete( s, 1, i );
   if iLastLine <> -1 then
   begin
    Memo1.Lines[iLastLine] := Memo1.Lines[iLastLine] + ss;
    iLastLine := -1;
   end
   else
    Memo1.Lines.Append( ss );
   if not cpDrv.SendString( ss + #13 ) then
   begin
    CannotSendError;
    exit;
   end;
   i := pos( #13, s );
  end;
  if iLastLine <> -1 then
   Memo1.Lines[iLastLine] := Memo1.Lines[iLastLine] + s
  else
   Memo1.Lines.Append( s );
  if not cpDrv.SendString( s ) then
   CannotSendError;
 end;
end;
```

เมื่อมีการกดปุ่ม Paste จะทำการวางค่าที่ทำ Highlight ไว้ใน Memo1

```
procedure TDlForm.IT_CopyCmdClick(Sender: TObject);
begin
 IncomingRichEdit.CopyToClipboard;
end;
```

เมื่อมีการกดปุ่ม Copy จะทำการคัดลอกค่าที่ทำ Highlight ไว้ใน IncomingRichEdit

```
procedure TDlForm.OT_PopupMenuPopup(Sender: TObject);
begin
 OT_ClearCmd.Enabled := Memo1.Lines.Count > 0;
 OT_CutCmd.Enabled := Memo1.SelLength > 0;
 OT_CopyCmd.Enabled := Memo1.SelLength > 0;
 OT_PasteCmd.Enabled := Clipboard.HasFormat( CF_TEXT );
end;
procedure TDlForm.IT_PopupMenuPopup(Sender: TObject);
```

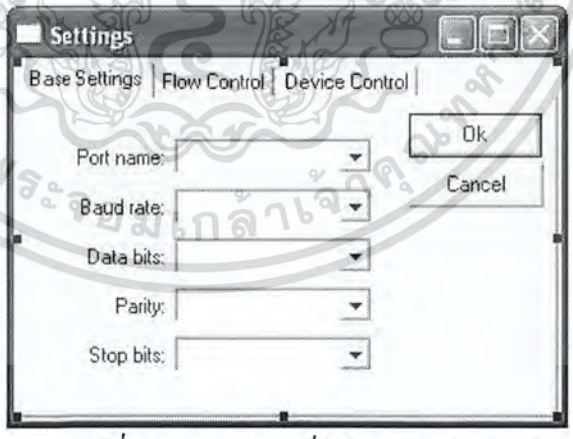
```
begin
  IT_ClearCmd.Enabled := IncomingRichEdit.Lines.Count > 0;
  IT_CopyCmd.Enabled := IncomingRichEdit.SelLength > 0;
end;
procedure TDlForm.Button1Click(Sender: TObject);
var
    Hex_file : file;
    buf : string;
    NumRead: integer;
begin
     OpenDialog1.Execute;
     Edit1.Text := OpenDialog1.FileName;
     DlForm.Memo1.Lines.LoadFromFile(OpenDialog1.Filename);
end;
```

เมื่อมีการกดปุ่ม Browse จะทำการเปิดหน้าต่าง OpenDialog เพื่อเปิดไฟล์ที่ต้องการดาวน์โหลด

```
procedure TDlForm.Button2Click(Sender: TObject);
begin
  //Do nothing if not connected
  if not cpDrv.Connected then
    exit;
  // Send the character
  if not cpDrv.SendString(Memo1.Lines.text) then
    CannotSendError;
end;
end.
```

เมื่อมีการกดปุ่ม Send จะทำการส่งข้อมูลที่อยู่ใน Memo1

- Unit SettingsDlg เป็นหน้าต่างที่จะเปิดขึ้นเมื่อต้องการจะกำหนดค่าต่าง ๆ ให้กับการรับส่งข้อมูลระหว่าง PC และ Hardware



รูปที่ 5.5 *แสดง Form ที่ได้จาก Unit SettinsDlg*

```
unit SettingsDlg;
interface
uses
  // Delphi units
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls, ComCtrls, ExtCtrls,
  // ComDrv32 units
```

```
  CPDrv;
type
  TSettingsForm = class( TForm )
    Panel1: TPanel;
    SettingsPageControl: TPageControl;
    BaseSettingsTabSheet: TTabSheet;
    OkButton: TButton;
    CancelButton: TButton;
    PortComboBox: TComboBox;
    Label1: TLabel;
    BaudRateComboBox: TComboBox;
    Label2: TLabel;
    Label3: TLabel;
    DataBitsComboBox: TComboBox;
    Label4: TLabel;
    ParityComboBox: TComboBox;
    Label5: TLabel;
    StopBitsComboBox: TComboBox;
    FlowControlTabSheet: TTabSheet;
    Label6: TLabel;
    HwFlowComboBox: TComboBox;
    Label7: TLabel;
    SwFlowComboBox: TComboBox;
    Label8: TLabel;
    DTRControlComboBox: TComboBox;
    TabSheet1: TTabSheet;
    Label9: TLabel;
    DevCheckComboBox: TComboBox;
    procedure OkButtonClick(Sender: TObject);
  private
    // TCommPortDriver whose settings must be customized
    FCPDrv: TCommPortDriver;
    // Shows current settings
    procedure UpdateControls;
  public
    // Constructor
    constructor Create( AOwner: TComponent; ACPDrv: TCommPortDriver ); reintroduce;
virtual;
  end;
implementation
{$R *.DFM}
// Constructor
constructor TSettingsForm.Create( AOwner: TComponent; ACPDrv: TCommPortDriver );
begin
  // Call inherited constructor
  inherited Create( AOwner );
  // Save the cpDrv reference
  FCPDrv := ACPDrv;
  // Show current settings
  UpdateControls;
  // Be sure the "Base Settings" tab control is the one selected
  SettingsPageControl.ActivePage := BaseSettingsTabSheet;
end;
// Shows current settings
procedure TSettingsForm.UpdateControls;
begin
  // Base Settings page
  PortComboBox.Text := FCPDrv.PortName;
  BaudRateComboBox.Text := IntToStr( FCPDrv.BaudRateValue );
  DataBitsComboBox.ItemIndex := ord( FCPDrv.DataBits );
  ParityComboBox.ItemIndex := ord( FCPDrv.Parity );
  StopBitsComboBox.ItemIndex := ord( FCPDrv.StopBits );
  // Flow Control page
  HwFlowComboBox.ItemIndex := ord( FCPDrv.HwFlow );
  SwFlowComboBox.ItemIndex := ord( FCPDrv.SwFlow );
  DTRControlComboBox.ItemIndex := ord( not FCPDrv.EnableDTROnOpen );
  // Device Control page
  DevCheckComboBox.ItemIndex := ord( not FCPDrv.CheckLineStatus );
end;
procedure TSettingsForm.OkButtonClick(Sender: TObject);
var newPortName: string;
    newBaudRate: DWORD;
```

```
begin
 // Validate settings
 newPortName := Trim( PortComboBox.Text );
 if newPortName = '' then
 begin
  Application.MessageBox( 'Please, enter a valid port name.',
                'Error',
                MB_OK or MB_ICONERROR );
  exit;
 end;
 newBaudRate := DWORD( StrToIntDef( BaudRateComboBox.Text, -1 ) );
 if (newBaudRate < 110) or (newBaudRate > 921600) then
 begin
  Application.MessageBox( 'Please, enter a valid baud rate.',
                'Error',
                MB_OK or MB_ICONERROR );
  exit;
 end;
 // Apply new settings
 FCPDrv.PortName := newPortName;
 FCPDrv.BaudRateValue := newBaudRate;
 FCPDrv.DataBits := TDataBits( DataBitsComboBox.ItemIndex );
 FCPDrv.Parity := TParity( ParityComboBox.ItemIndex );
 FCPDrv.StopBits := TStopBits( StopBitsComboBox.ItemIndex );
 FCPDrv.HwFlow := THwFlowControl( HwFlowComboBox.ItemIndex );
 FCPDrv.SwFlow := TSwFlowControl( SwFlowComboBox.ItemIndex );
 FCPDrv.EnableDTROnOpen := DTRControlComboBox.ItemIndex = 0;
 FCPDrv.CheckLineStatus := DevCheckComboBox.ItemIndex = 0;
 // Done
 ModalResult := mrOk;
end;
end.
```

โค้ดด้านบนนี้จะเป็นการกำหนดค่าต่าง ๆ ที่จะใช้ในการเชื่อมต่อ เช่น พอร์ท, ความเร็ว ฯลฯ

5.2.2 ซอฟแวร์สำเร็จรูปที่อยู่บนไมโครโรบอท

ซอฟแวร์สำเร็จรูปที่อยู่บนไมโครโรบอททำหน้าที่รับค่าจากคอมพิวเตอร์ส่วนบุคคลผ่านทางการดาวน์โหลด โดยค่าที่ได้รับนี้จะเป็นโปรแกรมที่ผู้ใช้ได้ทำการแปลงจากโปรแกรม ASM-51 ให้อยู่ในภาษาที่สามารถใช้คอบคุมไมโครโรบอทได้ (ไฟล์ .HEX) และนำไปเก็บไว้ในหน่วยความจำภายนอก หลังจากนั้นก็ส่งค่าที่นำไปเก็บกลับมายังตัวคอมพิวเตอร์ส่วนบุคคล เพื่อให้ผู้ใช้สามารถตรวจสอบได้ว่าค่าที่นำไปเก็บนั้นถูกต้องหรือไม่ สุดท้ายจึงโดดไปทำคำสั่งที่เก็บไว้

โปรแกรมนี้ได้พัฒนาโดยใช้ภาษา ASM-51 ,คอมไพล์ด้วยโปรแกรม Systronix RAD51 และถูกเบิร์นอยู่ในไมโครคอนโทรลเลอร์ AT89C51

บทที่6
ผลการทดลอง

ในบทนี้จะกล่าวถึงการนำชุดควบคุมหุ่นยนต์ขนาดเล็กไปใช้งาน โดยจะแบ่งการทำงานออกเป็น 2 ส่วนหลัก ๆ 2 ส่วน ได้แก่ การเชื่อมต่อส่วนฮาร์ดแวร์ และการใช้งานโปรแกรม CFM

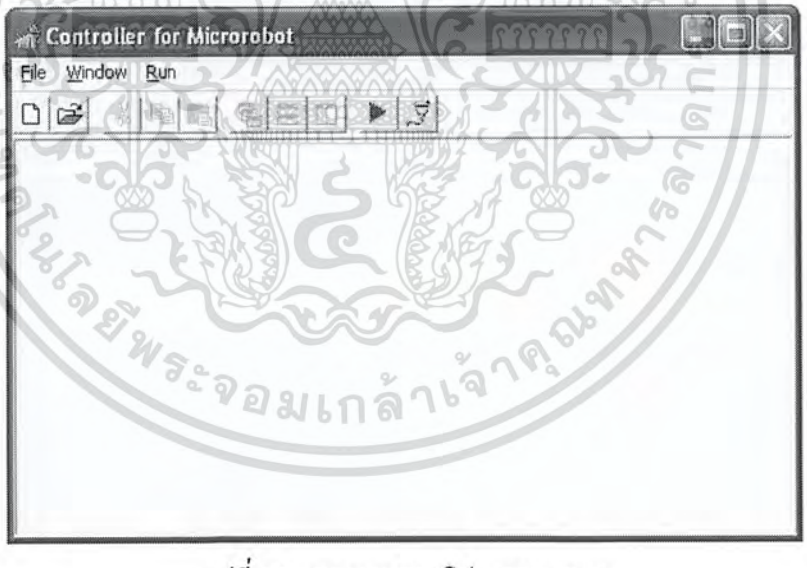
6.1 การเชื่อมต่อส่วนฮาร์ดแวร์

1. เชื่อมต่ออุปกรณ์ต่าง ๆ เข้ากับพอร์ท ซึ่งจะต้องพิจารณาด้วยว่าเป็นอุปกรณ์อินพุต หรืออุปกรณ์เอาท์พุต
2. เชื่อมต่อพอร์ทอนุกรมเข้ากับคอนโทรลเลอร์

6.2 การใช้งานโปรแกรม CFM

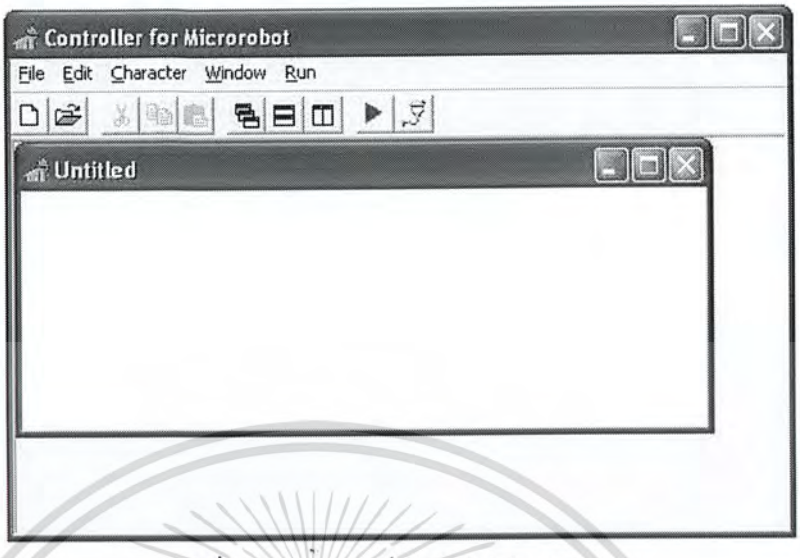
ในการใช้งานครั้งแรก ผู้ใช้ต้องทำการก๊อปปี้ไดเรคทอรี่ CFM ที่อยู่ในแผ่นดิสก์ ไปเก็บไว้ในไดเรคทอรี่ C:\ หลังจากนั้นให้ปฏิบัติตามนี้

1. เรียกโปรแกรม CFM.exe จากไดเรคทอรี่ C:\CFM จะได้โปรแกรมดังรูปที่ 6.1



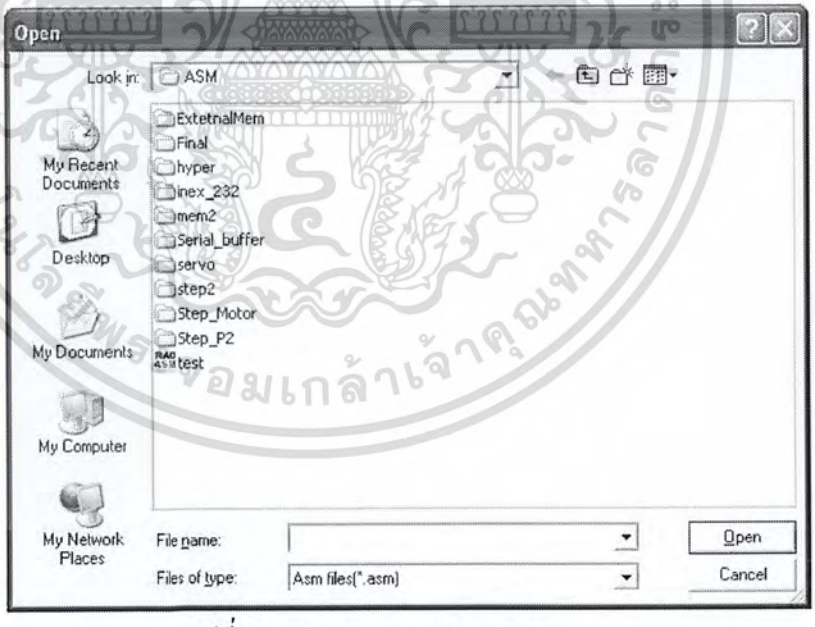
รูปที่ 6.1 แสดงภาพของโปรแกรม CFM

2. ในการเขียนโปรแกรมเพื่อควบคุมการทำงานของไมโครโรบอท ให้คลิกที่ปุ่ม 🗋 หรือเลือก New ที่เมนู File เพื่อสร้างไฟล์ใหม่ เมื่อคลิ๊กแล้วจะได้ไฟล์ untitled ดังรูปที่ 6.2



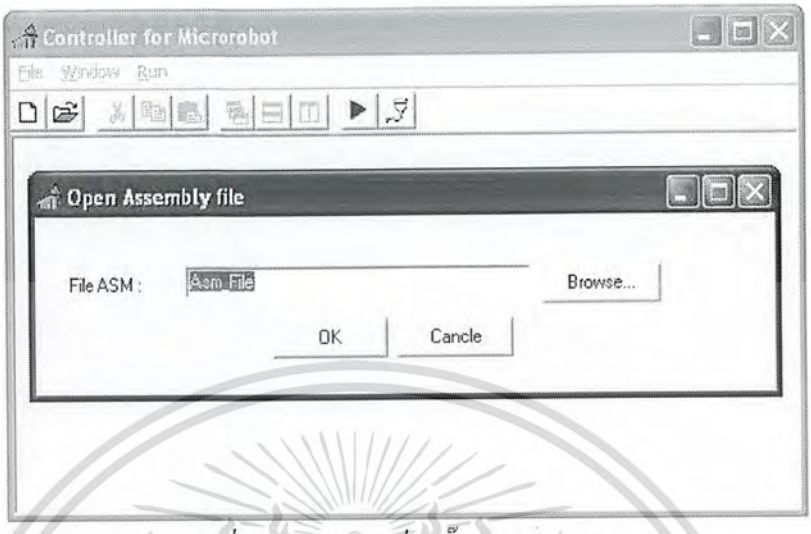
รูปที่ 6.2 แสดงภาพเมื่อสร้างไฟล์ใหม่

หรือคลิ๊กที่ปุ่ม เพื่อเปิดไฟล์เก่า เมื่อคลิ๊กแล้วจะได้ Open Dialog ให้เลือกไฟล์ที่ต้องการจะเปิด ดังรูปที่ 6.3



***รูปที่ 6.3** แสดงภาพของ OpenDialog*

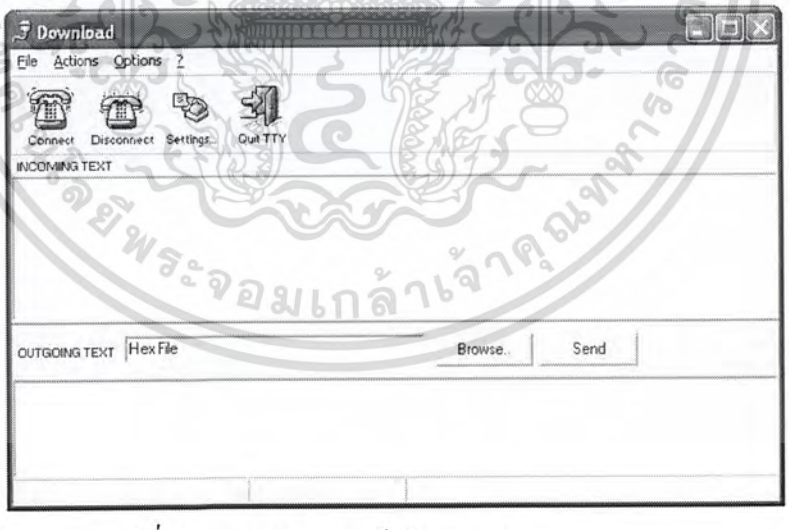
3. เมื่อเขียนโปรแกรมเสร็จแล้วให้คลิกที่ปุ่ม หรือเลือก Compile ที่เมนู Run จะเปิดหน้าต่างสำหรับเปิดไฟล์ที่ต้องการ Compile ขึ้นมาดังรูปที่ 6.4



รูปที่ 6.4 แสดงภาพเมื่อคลิ๊กปุ่ม Compile

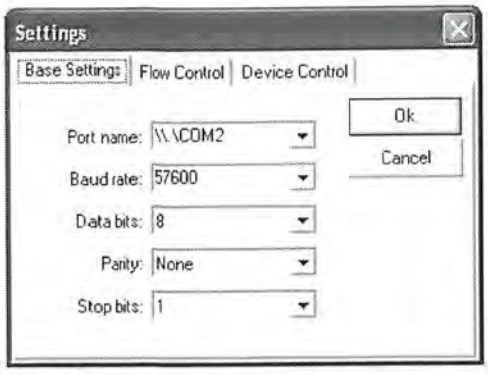
เมื่อใส่ชื่อไฟล์ แล้วคลิ๊กปุ่ม OK จะได้ไฟล์ .HEX อยู่ในไดเรคทอรี่เดียวกับไฟล์ .ASM

4. จากนั้นคลิ๊กปุ่ม หรือเลือก Download จากเมนู Run เพื่อทำการดาวน์โหลด ไฟล์ .HEX เมื่อคลิ๊กแล้วจะได้หน้าต่างดังรูปที่ 6.5



รูปที่ 6.5 แสดงภาพของหน้าต่าง Download

5. คลิ๊กปุ่ม Setting เพื่อทำการกำหนดค่าต่าง ๆ ในการดาวน์โหลด เมื่อคลิ๊กแล้วจะได้หน้าต่างดังรูป 6.6 เมื่อกำหนดค่าต่าง ๆ เสร็จแล้วให้คลิ๊กปุ่ม Ok



รูปที่ 6.6 แสดงภาพของหน้าต่าง Settings

6. คลิ๊กปุ่ม Browse ที่หน้าต่าง Download เพื่อเลือกไฟล์ที่ต้องการจะดาวน์โหลด (ต้องเป็นไฟล์ .HEX)
7. คลิ๊กปุ่ม Connect เพื่อทำการเชื่อมต่อกับส่วน Hardware
8. กดปุ่ม Send เพื่อทำการส่งข้อมูลไปยังคอนโทรลเลอร์

เมื่อทดลองใช้งานแล้วได้ผลดังนี้

ความสามารถในการส่งและรับข้อมูลผ่านพอร์ทอนุกรม

จำนวนข้อมูลที่ส่ง (ตัว)	ความผิดพลาดเมื่อทดลอง 100 ครั้ง	เปอร์เซ็นต์ความผิดพลาด
10	0	0
50	0	0
100	1	1
500	1	1

ความสามารถในการใช้งานอย่างต่อเนื่อง ทดลองโดยใช้ DC MOTOR 6 Volt

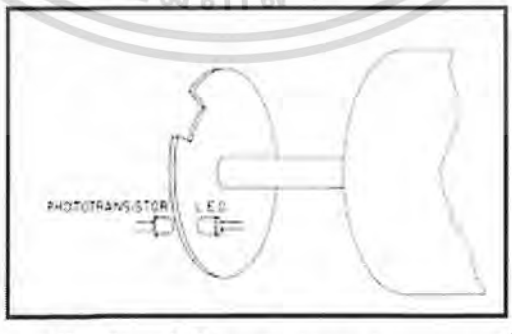
จำนวนอุปกรณ์ที่ใช้	ระยะเวลาการใช้งานอย่างต่อเนื่อง (นาที)
1	73
2	68
3	51
4	44

บทที่ 7
การประยุกต์ไปใช้งาน

เนื่องจากโครงงานไมโครคอนโทรลเลอร์สำหรับไมโครโรบอทนี้มีจุดประสงค์หลักเพื่อเพิ่มความอเนกประสงค์ให้สามารถใช้งานได้ในสถานะและกับอุปกรณ์หลาย ๆ ชนิดดังนั้นหลักที่จะทำให้โครงงานนี้มีประโยชน์สูงสุดก็คือการนำเอาไมโครโรบอทนี้ไปประยุกต์ใช้งานกับการทำงานจริง ซึ่งการทำงานได้จริงนี้จะต้องประกอบด้วยทั้งอุปกรณ์ที่ใช้ในการทำงานคือเอาท์พุตที่เป็นมอเตอร์ และอุปกรณ์ที่เป็นอินพุตคือเซนเซอร์ต่าง ๆ

7.1 ลักษณะการควบคุมโดยมี feedback

การทำงานของอุปกรณ์เอาท์พุตต่างๆ นั้นเมื่อใช้งานจริงสภาวะแวดล้อมขณะนั้นอาจจะมีผลทำให้การทำงานของอุปกรณ์ไม่ตรงกับความต้องการของผู้ใช้อย่างแท้จริง จึงควรที่จะมีการตรวจสอบการทำงานของอุปกรณ์เหล่านั้นยกตัวอย่างเช่น เมื่อเรานำเอา ดีซีมอเตอร์ต่อเข้ากับวงจรที่ควบคุมโดยแรงดันก่อนที่เราจะ take load ให้กับดีซีมอเตอร์ตัวนั้นเมื่อเราจ่ายไฟ 5 โวลต์ให้กับมอเตอร์มอเตอร์อาจหมุนด้วยความเร็ม 500 รอบต่อวินาทีเมื่อเรานำไปใช้งานจริง เราต้องการให้มอเตอร์หมุนด้วยความเร็ว 500 รอบต่อวินาทีเราจึงคาดการณ์ว่าต้องจ่ายไฟ 5 โวลต์ให้กับมอเตอร์ตัวนั้นแต่เมื่อนำมอเตอร์ตัวนี้ไปใช้งานจริงจะมีแรงหน่วงจากอุปกรณ์ที่ต่อเข้ากับมอเตอร์ทำให้มอเตอร์หมุนช้ากว่าที่กำหนดไว้เราจึงจำเป็นที่จะต้องมีอุปกรณ์ตรวจสอบว่ามอเตอร์หมุนด้วยความเร็วตามที่เราต้องการหรือไม่และทำอย่างไรมอเตอร์จึงจะหมุนด้วยความเร็วตามที่เราต้องการจึงอยู่ที่การประยุกต์นำเอาลักษณะวงจร feedback มาใช้ตัวอย่างเช่น เมื่อเราต้องการให้มอเตอร์หมุนด้วยความเร็ว 500 รอบต่อวินาทีเราก็จะใช้ตัวเซนเซอร์ infrared มาตรวจสอบความเร็วของมอเตอร์หากมอเตอร์หมุนไม่ได้ตามความเร็วที่เรากำหนดโปรแกรมก็จะทำการเพิ่มหรือลดแรงดันไฟฟ้าที่จ่ายให้มอเตอร์จนกว่ามอเตอร์จะหมุนด้วยความเร็ว 500 รอบต่อวินาที



รูปที่ 7.1 ตัวอย่างการประยุกต์โดยมีการส่งค่า feedback จาก photosensor เพื่อควบคุมความเร็วมอเตอร์

7.2 การประยุกต์การใช้งานโดยไม่มีค่า feedback

อุปกรณ์เอาท์พุตบางชนิดไม่จำเป็นต้องมีการรับค่า feedback เพื่อตรวจสอบการทำงานของอุปกรณ์ อาจจะเนื่องมาจากลักษณะของอุปกรณ์เองหรือเป็นเพราะไม่ต้องการความละเอียดในการทำงานซึ่งต้องขึ้นอยู่กับวิจารณญาณของผู้ใช้ตัวอย่างของการทำงานของอุปกรณ์ที่ไม่ต้องการรับค่า feedback เช่น การทำงานของ LED ซึ่งถ้า LED ทำงานในลักษณะเอาท์พุตเพียงอย่างเดียวส่วนมากเราก็จะไม่คำนึงถึงค่าความสว่างเท่าไหร่นัก เนื่องจากไม่ได้มีการสื่อความหมายแต่อย่างไร

การประยุกต์นำไปใช้งานทั้งสองลักษณะเมื่อนำมารวมกันแล้วสามารถทำให้อุปกรณ์มีประโยชน์อย่างมากโดยตัวอย่างการนำประยุกไปใช้งานเช่น การประยุกต์ใช้งานเป็นไมโครโรบอท หรือการประยุกต์ใช้งานในลักษณะที่คล้ายคลึงกับการทำงานของ PLC ควบคุมการทำงานของฝ่ายการผลิต

บรรณานุกรม

[1] ธีรวัฒน์ ประกอบผล, "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์," สำนักพิมพ์ ส.ส.ท.,2543.

[2] สมชาย อังศุโชติกุล และ นายอดิศักดิ์ โหตระภวานนท์ , "ไมโครเมาส์," ปริญญานิพนธ์ วศ.บ. สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2540. อัดสำเนา.