# การพัฒนาชุดคำสั่งและโปรแกรมจัดการ สำหรับการนำคอม86 ไปประยุกต์ใช้ในงานหุ่นยนต์

# COM86 ROBOT FRAMEWORK



นายพงศกร ศรีราชญา
นายพลวัต เหลืองศุภบูลย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2546

# การพัฒนาชุดคำสั่งและโปรแกรมจัดการ
# สำหรับการนำคอม86 ไปประยุกต์ใช้ในงานหุ่นยนต์
# COM86 ROBOT FRAMEWORK


โดย

นายพงศกร ศรีราชญา

นายพลวัต เหลืองศุภบูลย์

อาจารย์ที่ปรึกษา

ผศ. อภินตร อูนากูล


ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ปีการศึกษา 2546

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาชุดคำสั่งและโปรแกรมจัดการสำหรับการนำคอม86 ไปประยุกต์ใช้ในงานหุ่นยนต์ Com86 Robot Framework

ผู้จัดทำ

1. นาย พงศกร ศรีราชญา รหัส 43010275
2. นาย พลวัต เหลืองศุภบูลย์ รหัส 43010294

อาจารย์ที่ปรึกษา

( ผศ. อภิเนตร อูนากูล )

# การพัฒนาชุดคำสั่งและโปรแกรมจัดการสำหรับการนำคอม86 ไปประยุกต์ใช้ในงานหุ่นยนต์

นาย พงศกร ศรีราชญา 43010275
นาย พลวัต เหลืองศุภบูลย์ 43010294
ผศ. อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
ปีการศึกษา 2546

**บทคัดย่อ**

การนำหุ่นยนต์มาใช้งานในด้านต่าง ๆ นั้นเป็นเรื่องที่ได้รับความสนใจอย่างมากในปัจจุบัน แต่ถึงอย่างไร การที่จะสร้างหุ่นยนต์ขึ้นมาใช้งาน ยังมีข้อจำกัดอยู่หลายประการ ส่วนมากมักจะเกิดจากการขาดความรู้และความเข้าใจ เนื่องจากสร้างหุ่นยนต์จำเป็นต้องมีความรู้ทั้งทางด้านฮาร์ดแวร์ และซอร์ฟแวร์ที่ค่อนข้างสูงพอสมควร อีกทั้งขีดความสามารถของหุ่นยนต์ในปัจจุบันนั้น พัฒนาไปมาก การใช้งานไมโครคอนโทลเลอร์ ที่สามารถรองรับการประมวลผลที่สูงขึ้นมาใช้งาน ยิ่งทำให้ต้องทำการศึกษาการทำงานของอุปกรณ์เพิ่มมากขึ้นอีก จึงทำให้การพัฒนาหุ่นยนต์ ยังถูกมองว่าเป็นเรื่องที่ยากอยู่

โดยเหตุนี้ คณะผู้จัดทำโครงงานนี้จึงมีความคิด ที่จะพัฒนาต้นแบบของโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์ และชุดคำสั่งควบคุมและสั่งงานอุปกรณ์ สำหรับหุ่นยนต์ที่ใช้ชุดพัฒนาคอม86เป็นตัวประมวลผลหลัก โดยการออกแบบแพลตฟอร์มที่เหมาะสมและสะดวกต่อการใช้งานขึ้น ซึ่งจะช่วยทำให้การพัฒนางานด้านหุ่นยนต์ทำได้ง่าย และสะดวกที่จะนำไปประยุกต์ใช้งานต่อไป

# Com86 Robot Framework


| | | |
|---|---|---|
| Mr. Pongsakorn | Sriradya | 43010275 |
| Mr. Polwat | Lungsupabul | 43010294 |
| Mr. Aplinetr | Unakul | Advisor |



**Abstract**

At this present time, the use of Robot is widely use in many areas and it is very interesting. However, there are some limits for building a robot, mostly are the lack of knowledge and understanding. Building a robot, the high knowledge of hardware and software is necessary. And as the ability of the robot nowadays is much created and developed, to support it, the micro-controller, which can support the higher evolution, is introduced to use in building the robot. Then the higher ability is relevant to the harder study of working of the devices of the robot also. Thus, the development of the robot is assumed that it is still difficult.

So, the idea of this project is created to develop the assistant program for using the robot, and the order set for controlling and directing the devices for the Framework and Engine for com86 Robot. It will help smooth the complicated of the set up and program writing of the robot which is easier to build a robot and it is convenient to adapt in other work in the future.

# กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และความร่วมมือจากหลาย ๆ ฝ่าย ด้วยกัน บุคคลท่านแรกที่ทางคณะผู้จัดทำต้องกล่าวถึง เพราะเป็นส่วนสำคัญ ที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ ก็คือ ผศ. อภิเนตร อูนากูล ซึ่งท่านเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์นี้ ที่ให้ความเอาใจใส่ แนะนำ และให้ความช่วยเหลืออย่างดีเสมอมา นอกจากนั้นยังได้ให้ความรู้และแนวคิดในการทำงาน รวมถึงการใช้ชีวิตอันมีค่ายิ่งให้แก่ผู้เขียน ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ขอขอบคุณพี่ ๆ เพื่อน ๆ ห้อง ESL และเพื่อน ๆ ในห้องโปรเจ็กต์อื่นทุก ๆ ห้อง ที่คอยช่วยเป็นกำลังใจ โดยเฉพาะห้องฮาร์ดแวร์ที่ให้ความรู้และข้อมูลในการทำโครงงาน ขอบคุณภาควิชาและมหาวิทยาลัยแห่งนี้ ที่มอบประสบการณ์ที่ประทับใจต่อการใช้ชีวิตในมหาวิทยาลัย และขอขอบคุณคู่ร่วมงานของข้าพเจ้า ที่นับว่าเป็นสหายที่ร่วมอุปสรรคเคยเรียน เล่น ทำงานด้วยกันมาตลอด ซึ่งต้องเผชิญกับปัญหานานับประการ แต่ตลอดเวลาที่ทำงานร่วมกันมานั้นคอยช่วยเหลือและเป็นกำลังใจในการทำวิทยานิพนธ์มาโดยตลอด น้ำใจของเหล่าเพื่อนไม่เคยแห้งแล้งในถิ่นแดนนี้

ขอขอบคุณ บริษัทเน็ตเกตเจ็ตที่เอื้อเฟื้อบอร์ดคอม86 ขอขอบคุณพี่โอที่เอื้อเฟื้อซอร์ฟแวร์และให้ความรู้ในการเขียนโปรแกรม ขอขอบคุณพี่ภาคที่สอนการใช้งานบอร์ดคอม86 และขอขอบคุณทุกคนที่ให้กำลังใจอย่างดีเสมอมา

สุดท้ายนี้ ต้องขอขอบพระคุณบุคคลที่ทำให้ข้าพเจ้ามีวันนี้ คือ บิดา มารดา รวมทั้งญาติพี่น้องอันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่และยังให้กำลังใจเอาใจใส่ในตัวข้าพเจ้าเสมอมา ในทุก ๆ ด้านอันหาที่เปรียบไม่ได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี่

นาย พลวัต เหลืองศุภบูลย์
นาย พงศกร ศรีราชญา

# สารบัญ

หน้าที่

หน้าที่

หน้าที่

หน้าที่

# สารบัญภาพ

หน้าที่



หน้าที่

# บทที่ 1
# บทนำ

## 1.1 ที่มาและความสำคัญ

ปัจจุบันการพัฒนาด้านหุ่นยนต์ได้รับความสนใจมากขึ้น แต่ก็ยังมีปัญหาในการพัฒนาเพราะผู้ที่จะพัฒนาหุ่นยนต์จะต้องมีความรู้ทั้งทางด้านฮาร์ดแวร์และซอร์ฟแวร์ที่ค่อนข้างสูงพอสมควร จึงทำให้การพัฒนาหุ่นยนต์ยังถูกมองว่าเป็นเรื่องที่ยากอยู่

โดยเหตุนี้ คณะผู้จัดทำโครงงานนี้จึงมีความคิด ที่จะพัฒนา โปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์ และ ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ สำหรับหุ่นยนต์ที่ใช้ชุดพัฒนาคอม86เป็นตัวประมวลผลหลักขึ้น เพื่อให้การพัฒนาหุ่นยนต์ง่ายขึ้นมา โดยช่วยลดปัญหาทางด้านฮาร์ดแวร์ด้วยการสร้างพอร์ตสำหรับเชื่อมต่อโมดูลของอุปกรณ์ต่าง ๆ และให้ผู้พัฒนาหุ่นยนต์สั่งงานอุปกรณ์ต่างๆผ่านทางพอร์ตที่เชื่อมต่อไว้ โดยที่ผู้พัฒนาหุ่นยนต์ไม่จำเป็นต้องทราบถึงแอดเดรสบัสที่ใช้เชื่อมต่อกับโมดูลของอุปกรณ์ที่แท้จริง ส่วนการลดปัญหาทางด้านการออกแบบซอร์ฟแวร์นั้น ทางคณะผู้จัดทำได้สร้างชุดคำสั่ง ขึ้นมาเพื่อรองรับการควบคุมและสั่งงานอุปกรณ์ต่าง ๆ โดยชุดคำสั่งที่เตรียมไว้ให้จะเป็นตัวช่วยเหลือให้ผู้พัฒนาหุ่นยนต์สร้างซอร์ฟแวร์ควบคุมหุ่นยนต์ได้ง่ายขึ้น

อีกทั้งคณะผู้จัดทำ ได้สร้างส่วนการรายงานสถานะของอุปกรณ์กลับมายังผู้พัฒนาหุ่นยนต์เพื่อให้ผู้พัฒนาหุ่นยนต์สามารถตรวจสอบความถูกต้องและนำไปแก้ไขซอร์ฟแวร์ให้ตรงตามที่ต้องการได้

## 1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อสร้างแพลตฟอร์มสำหรับหุ่นยนต์(Robot Platform) สำหรับนำไปใช้ในการพัฒนาหุ่นยนต์ต่อไป
2. เพื่อสร้างชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ที่ใช้ในการควบคุมหุ่นยนต์
3. เพื่อพัฒนาแอปพลิเคชั่นโปรแกรมสำหรับช่วยเหลือในการเรียกใช้งานชุดคำสั่ง
4. เพื่อพัฒนาส่วนรับส่งข้อมูลระหว่างหุ่นยนต์และผู้ใช้งานผ่านทางคอมพิวเตอร์ เพื่อตรวจสอบการทำงานและควบคุมหุ่นยนต์จากระยะไกล

## 1.3 ขอบเขตของโครงงาน

1. จัดทำชุดคำสั่งพื้นฐานที่ผู้ใช้งานสามารถเรียกใช้เป็นตัวอย่างโดยแบ่งออกเป็น 2 ประเภทด้วยกันคือ ส่วนควบคุมการเคลื่อนที่ของหุ่นยนต์ และ ส่วนรับข้อมูลจากภายนอก โดย 2 ส่วนประกอบด้วยตัวอย่าง ดังนี้
   - มอเตอร์ 2 ประเภท คือ สเต็ปปิ้งมอเตอร์ และ ดีซีมอเตอร์
   - เซนเซอร์ 3 ประเภท คือ อินฟราเรด , อัลตราโซนิก ,สวิทช์สัมผัส
2. โปรแกรมควบคุมหลักที่สามารถช่วยเหลือผู้ใช้งานในการจัดการกับหุ่นยนต์ในด้าน ต่าง ๆ ดังนี้
   - โหลดโปรแกรมที่เขียนจากคอมพิวเตอร์ไปยังคอม86
   - แสดงผลการทำงานของคอม86ไปยังคอมพิวเตอร์
   - ช่วยเหลือผู้ใช้งานในการติดตั้งอุปกรณ์และควบคุมการทำงาน

- ส่วนที่จะควบคุมปรับเปลี่ยนการทำงานและตั้งค่าการทำงานของหุ่นยนต์รวมถึงการแก้ไขโปรแกรมของหุ่นยนต์ ผ่านทางคอมพิวเตอร์ในแบบเรียลไทม์

3. หุ่นยนต์ต้นแบบที่นำโปรแกรมที่คณะจัดทำได้ออกแบบขึ้นไปเป็นส่วนควบคุมหลักและมีการเรียกใช้ชุดคำสั่งในการเขียนโปรแกรมควบคุมตัวหุ่นยนต์ โดยตัวอย่างหุ่นยนต์ที่กำหนดไว้คือ หุ่นยนต์ทำความสะอาด โดยมีรายละเอียด คือ เป็นหุ่นยนต์ใช้ อินฟราเรดเซนเซอร์ จำนวนมากในการตรวจจับข้อมูลสภาวะแวดล้อมรอบหุ่น และใช้วงจรอัลตราโซนิกในการหาระยะและคำนวณขนาดห้อง และเขียนโปรแกรมควบคุมการเคลื่อนที่ให้หุ่นยนต์เคลื่อนที่และตรวจสอบสิ่งกีดขวางโดยอัตโนมัติพร้อมกับสามารถควบคุมการทำงานโดยตรงจากผู้ใช้งานได้

**1.4 วิธีการดำเนินงาน**

1. ค้นหาข้อมูลการเขียนโปรแกรมช่วยเหลือสำหรับทำหุ่นยนต์ ที่มีอยู่แล้วในปัจจุบัน เพื่อวิเคราะห์โครงสร้างความสามารถ การทำงานของโปรแกรมช่วยเหลือสำหรับทำหุ่นยนต์ เหล่านั้น
2. ออกแบบโครงสร้างและรายละเอียดการทำงานของโปรแกรมควบคุมหลัก
3. ออกแบบโครงสร้างและรายละเอียดชุดคำสั่งของโปรแกรมช่วยเหลือสำหรับทำหุ่นยนต์ และลักษณะการทำงานของโปรแกรมควบคุมหลัก
4. ศึกษาและออกแบบรูปแบบ โครงสร้าง และการทำงานของหุ่นยนต์ที่นำมาเป็นต้นแบบ
5. ออกแบบซอร์ฟแวร์ ในส่วน แผนภาพยูสเคส และ โครงสร้างของโปรแกรม
6. ออกแบบฮาร์ดแวร์ ออกแบบวงจรสำหรับใช้งานอุปกรณ์ต่าง ๆ
7. ศึกษาการทำงานและการนำ ไมโครคอนโทรลเลอร์ โอเปอร์เรทติ้ง ซิสเต็มเวอร์ชัน 2 ( μC/OS-II ) มาใช้งานกับคอม86
8. เขียนโปรแกรมช่วยเหลือสำหรับทำหุ่นยนต์ ควบคุมอุปกรณ์ และทดสอบการทำงาน
9. สร้างแอพพลิเคชั่นบนคอมพิวเตอร์ เพื่อเรียกใช้งานโปรแกรมควบคุมหลัก
10. ทดสอบการทำงานของโปรแกรมบนคอมพิวเตอร์ ก่อนทำการใช้งานจริง
11. ทดสอบซอฟต์แวร์ที่สร้างขึ้นมา จากการใช้งานบนหุ่นยนต์ต้นแบบที่สร้างขึ้น รวมถึงการตรวจสอบการทำงานและวิเคราะห์ผลการทำงานของหุ่นยนต์
12. สรุปผลการทำงาน ผลที่ได้รับจากงานวิจัยชิ้นนี้ และแนวทางในการพัฒนางานวิจัยนี้เพิ่มเติมและแนวทางการนำไปประยุกต์ใช้ต่อไป

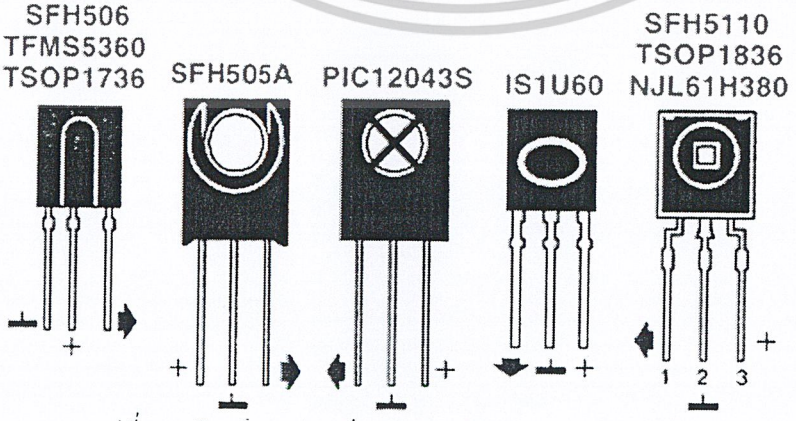
# บทที่ 2
# ทฤษฎีและหลักการของอุปกรณ์ที่เกี่ยวข้อง

เนื่องจากโครงงานนี้ มีจุดมุ่งหมายหลักเพื่อนำชุดพัฒนาคอม86 มาเป็นตัวประมวลผลหลัก และทำการช่วยเหลือผู้ใช้งานที่ต้องการสร้างหุ่นยนต์นั้นสามารถใช้งานคอม86ได้โดยง่าย ดังนั้นเพื่อออกแบบหุ่นยนต์ให้มีความยืดหยุ่นสามารถใช้ได้กับ อุปกรณ์อินพุตและเอาต์พุตหลายชนิด ตามความต้องการของผู้ใช้งานส่วนใหญ่ การจุดประสงค์หลัก จึงได้มีการศึกษาวิธีการใช้งานอุปกรณ์ต่าง ๆ ที่ตามที่หุ่นยนต์ทั่วไปนิยมใช้ ซึ่งในที่นี้จะแบ่งออกได้เป็น 2 ประเภทหลัก ๆ ด้วยกัน คือ อุปกรณ์ที่ทำหน้าที่ตรวจจับและอุปกรณ์ที่ใช้ในการเคลื่อนที่ ทั้งนี้รวมถึงส่วนการเชื่อมต่อระหว่างตัวไมโครคอนโทรลเลอร์ที่เป็นตัวประมวลผลของหุ่นยนต์ ( คอม86 ) และคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม

## 2.1 อุปกรณ์ตรวจจับ ( Sensor )

ในวงจรอิเล็กทรอนิกส์ในส่วนนำสัญญาณเข้า ที่ทำหน้าที่เป็นส่วนรับรู้ความรู้สึกต่าง ๆ เราเรียกว่าตัวตรวจจับ ( Sensor ) ซึ่งจะทำการเปลี่ยนแปลงความรู้สึกต่าง ๆ ที่ได้รับเป็นสัญญาณทางไฟฟ้าซึ่งอาจจะเป็นแรงดันหรือกระแสก็ได้ และส่งให้กับวงจรอิเล็กทรอนิกส์เพื่อตีความหมาย และเอาผลดังกล่าวไปใช้งานได้ตามต้องการ

ตัวตรวจจับแบบพื้นฐานที่เราคุ้นเคยกันอย่างดี เช่น สวิตช์กลไก, สวิตช์แม่เหล็ก, เซลล์รับแสง, โฟโตทรานซิสเตอร์, ออปโต้คัปเปลอร์, ตัวตรวจจับตำแหน่ง, ตัวตรวจจับแรงดัน, ตัวตรวจจับอุณหภูมิ, ตัวตรวจจับเสียง เป็นต้น ตัวตรวจจับต่าง ๆ เหล่านี้ จะทำหน้าที่เปลี่ยนสถานภาพทางฟิสิกส์ให้เป็นสัญญาณทางไฟฟ้า เพื่อนำไปประยุกต์ใช้งานมในวงจรอิเล็กทรอนิกส์ให้สามารถทำงานได้ตามต้องการในชุดควบคุมหุ่นยนต์ขนาดเล็กนี้ไม่ได้มีอุปกรณ์ตรวจจับรวมอยู่ในชุดควบคุม แต่ชุดควบคุมถูกออกแบบมาให้สามารถรองรับการทำงานของตัวตรวจจับพื้นฐาน ได้แก่ เซ็นเซอร์อินฟราเรดและ เซนเซอร์อัลตร้าโซนิก โดยการต่อผ่านพอร์ทอินพุตของชุดควบคุมหุ่นยนต์ขนาดเล็ก

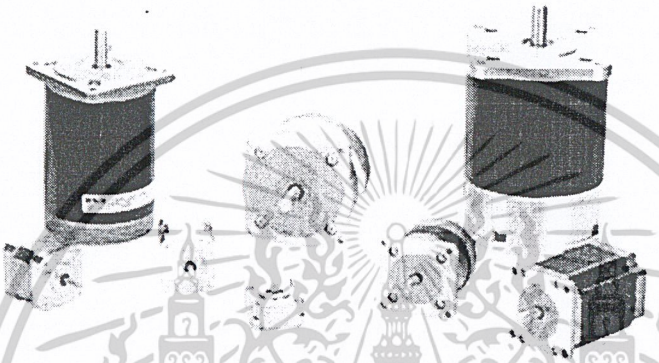



*รูปที่ 2-1 ตัวอย่างอุปกรณ์ตรวจจับ ( อินฟราเรดเซนเซอร์ )*

## 2.2 มอเตอร์ ( Motor )

ในการสร้างหุ่นยนต์นั้นอุปกรณ์ที่จะขาดไม่ได้ ในการสร้างหุ่นยนต์นั้นก็คือ ส่วนในการเคลื่อนที่ รวมถึงการเคลื่อนไหวต่าง ๆ ในทางกลไกของหุ่นยนต์แล้วจะต้องประกอบด้วยมอเตอร์ในแบบต่าง ๆ โดยอุปกรณ์มอเตอร์นั้นจะเป็นส่วนหลักในการจัดการและทำให้เกิดการเคลื่อนที่โดยมอเตอร์นั้นได้แบ่งออกเป็นอีกหลายประเภท โดยทางคณะผู้จัดทำโครงงานได้เลือกที่จะอธิบายในส่วนของ สเต็ปปิ้งมอเตอร์ และ เซอร์โวมอเตอร์

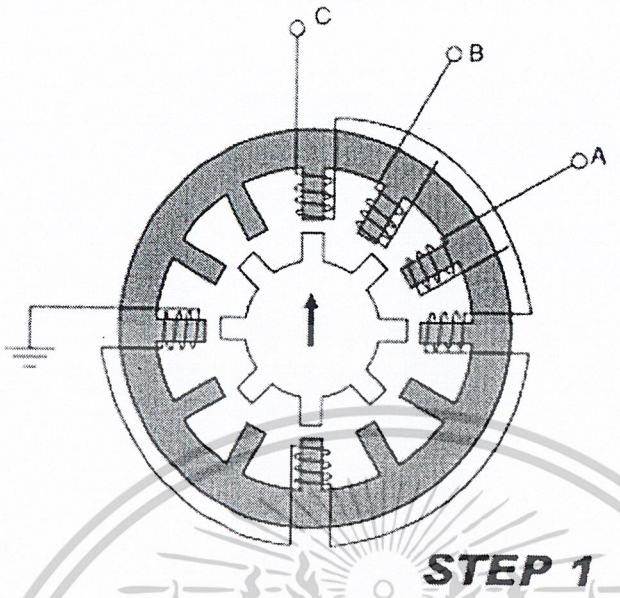
### 2.2.1 สเต็ปปิ้งมอเตอร์



*รูปที่ 2-2 ตัวอย่างสเต็ปปิ้งมอเตอร์*

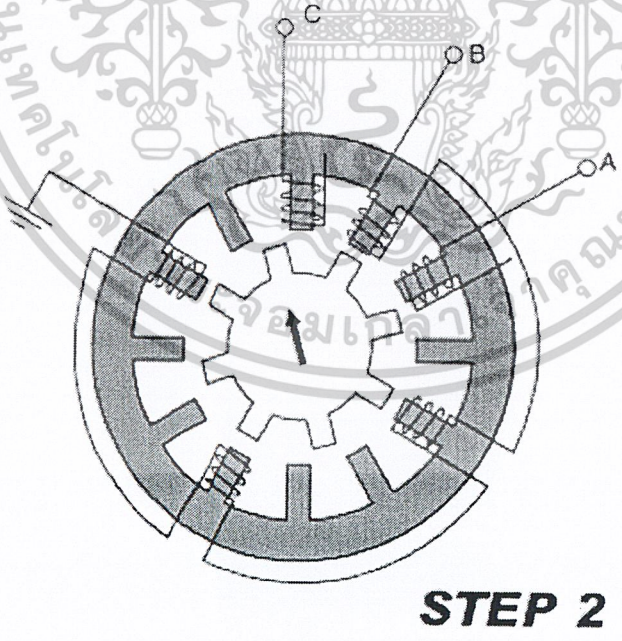
สเต็ปปิ้งมอเตอร์ ทำหน้าที่เปลี่ยนสัญญาณข้อมูลแบบดิจิตอล ไปสู่การเคลื่อนที่ทางกล อย่างได้สัดส่วนกัน โดยที่แกนหรือโรเตอร์ของมอเตอร์ชนิดนี้มักถูกควบคุมให้หมุนเป็นลำดับขั้นประโยชน์จากการใช้งาน สเต็ปปิ้งมอเตอร์ที่มีการควบคุมโดยใช้สัญญาณดิจิตอลคือ มีความถูกต้องเที่ยงตรงและสามารถเปลี่ยนตำแหน่งของโหลดได้อย่างรวดเร็ว เนื่องจากแต่ละอินพุตพัลซ์ จะทำให้ สเต็ปปิ้งมอเตอร์เคลื่อนที่ไปหนึ่งสเต็ป อย่างเที่ยงตรงเมื่อพิจารณาในแง่ทางกลจะพบว่า สเต็ปปิ้งมอเตอร์มีความเที่ยงตรงที่ ยอมรับได้ ซึ่งในการเปลี่ยนตำแหน่งอย่างง่ายอาจใช้สวิตซ์ ในการควบคุมมอเตอร์ นั่นคือ สร้างวงจรควบคุมมอเตอร์ได้ง่าย ถ้าต้องการเพิ่มประสิทธิภาพของส่วนควบคุม ให้ดีขึ้น อาจใช้ไอซีที่มีความเร็วสูงเนื่องจากประกอบด้วยมอสเฟสอยู่ภายใน จึงได้กำลังงานสูง และต้นทุนต่ำ ความสะดวกในการใช้งานนี้เองจึงทำให้นำไปสู่การใช้งานอย่างแพร่หลาย การได้รับประโยชน์จากการใช้สเต็ปปิ้งมอเตอร์อย่างเต็มที่นั้นขึ้นอยู่กับการขับอย่างถูกต้องเหมาะสม ซึ่งภาคขับนี้จะต้องประกอบไปด้วยแหล่งจ่ายไฟฟ้ากระแสตรง อิเล็กทรอนิกส์สวิตซ์โดยอาจจะเป็นทรานซิสเตอร์หรือมอสเฟส และแหล่งจ่ายสัญญาณข้อมูลแบบดิจิตอล ซึ่งมีลักษณะเป็นพัลซ์ เพื่อใช้ควบคุมทิศทางการหมุนของมอเตอร์ทิศทางของกระแสไฟฟ้าที่ป้อนเข้าสู่สเต็ปปิ้งมอเตอร์ ถูกควบคุมโดยการทำงานของอิเล็กทรอนิกส์สวิตซ์ ดังนั้น สเต็ปปิ้งมอเตอร์ จะหมุนไปหนึ่งช่วงในแต่ละอินพุตพัลซ์ ที่ป้อนเข้าสู่วงจรอิเล็กทรอนิกส์สวิตซ์ ซึ่งขนาดมุมที่หมุนไปจะขึ้นอยู่กับการออกแบบสเต็ปปิ้งมอเตอร์

**การทำงานของสเต็ปปิ้งมอเตอร์**



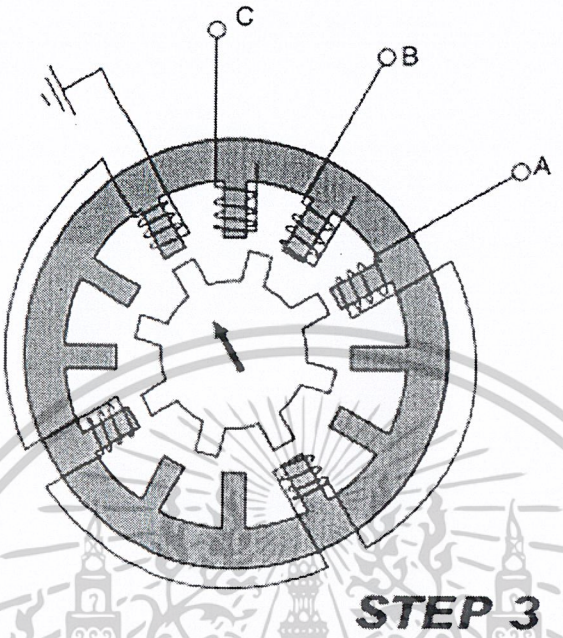

*รูปที่ 2-3 การทำงานของสเต็ปปิ้งมอเตอร์ขั้นที่ 1*

ตัวสเตอร์และโรเตอร์จะถูกสร้างด้วยเหล็กผสมซิลิคอน โดยที่ตัวโรเตอร์จะไม่ถูกพันด้วยคอล์ย แต่ส่วนที่ถูกพันด้วยคอล์ย จะเป็นส่วนของสเตเตอร์เท่านั้น ดังนั้นเมื่อเราจ่ายกระแสไฟฟ้าผ่านคอล์ย จึงทำให้สเตเตอร์มีคุณสมบัติเป็นแม่เหล็ก




*รูปที่ 2-4 การทำงานของสเต็ปปิ้งมอเตอร์ขั้นที่ 2*

เราจะจ่ายไฟให้แก่เฟส C ดังนั้นซี่ของโรเตอร์ก็จะถูกดูดเข้า หาเฟส C จากนั้นเราก็หยุดจ่ายไฟให้แก่เฟส C แล้วจะจ่ายไฟให้แก่ เฟส B ซี่ของสเตเตอร์อันถัดไปที่อยู่ใกล้เฟส B มากที่สุดก็จะถูกดูดเข้ามาแทน และเมื่อเราหยุดจ่ายไฟให้เฟส B และจ่ายไฟที่เฟส Aแทนโรเตอร์ที่อยู่ใกล้ A ที่สุดก็จะถูกดูดเข้ามาแทน




*รูปที่ 2-5 การทำงานของสเต็ปปิ้งมอเตอร์ขั้นที่ 3*

สเต็ปปิ้งมอเตอร์คือมอเตอร์ชนิดหนึ่งที่ทำการหมุนเป็นขั้นๆทีละน้อย ตามเส้นรอบวงและหยุดแตกต่างกับมอเตอร์ทั่ว ๆ ไปซึ่งจะหมุนไปเรื่อย ๆ ตลอดเวลา ซึ่งเราจะสามารถควบคุมการหมุนสเต็ปปิ้งมอเตอร์ได้อย่างละเอียดโดยคอมพิวเตอร์หรือไมโครคอนโทลเลอร์ สเต็ปปิ้งมอเตอร์สามารถ ใช้ในงานละเอียดต่าง ๆ เช่น ควบคุมการหมุนกระดาษในพรินเตอร์หรือใช้ในการเคลื่อนที่หัวอ่านและเขียนของ ฟลอปปี้ ดิสก์ เป็นต้น สเต็ปปิ้งมอเตอร์ใช้งานลักษณะระบบเปิด (Open Loop System) คือ สเต็ปปิ้งมอเตอร์สามารถทำงานได้โดยไม่ต้องมีการ ป้อนค่าพารามิเตอร์กลับมา (Feed back) แต่ทุกวิธีที่ต้องการกำหนดตำแหน่งที่แน่นอนนั้นล่ะ จะต้องป้อนกลับไปยังระบบและตัวบอก ตำแหน่งว่าถูกต้องหรือผิดพลาดให้รับทราบ

การพันขดลวดบนสเตเตอร์ ของสเต็ปปิ้งมอเตอร์ จะเห็นว่าการพันมีด้วยกัน 2 วิธี คือ แบบไบโพลาร์ (Bipolar) กับแบบยูนิโพลาร์ (Unipolar)

**การสั่งงานควบคุมการหมุนของสเต็ปปิ้งมอเตอร์**

ความยาวของขั้นการหมุนของสเต็ปปิ้งมอเตอร์ทั่วๆไปนั้นจะมีระยะตั้งแต่ 0.9 องศา ถึง 30 องศาโดยสเต็ปปิ้งมอเตอร์จะหมุนเป็นขั้นๆ โดยใช้การเปลี่ยนแปลงของกระแสในสนามแม่เหล็กของสเต็ปปิ้งมอเตอร์ ที่ใช้กันทั่วไปนั้นจะมี 3 แบบคือ แบบเวฟ, 2 เฟส และ 4 เฟส

- **แบบเวฟ (wave)**

จะเป็นการกระตุ้นแบบที่ง่ายที่สุด ซึ่งจะทำการกระตุ้นขดลวดทีละขดในเวลาหนึ่ง ๆ เรียงกันไปตัวอย่าง เช่น ขดที่ 1, 2 , 3 , 4 , 1 , 2 , 3 , 4 เป็นลำดับอย่างนี้ หรือ ขด 1 , 4 , 3 , 2 , 1 , 4 , 3 , 2 เป็นลำดับกันไป ทั้งนี้

ขึ้นอยู่กับทิศทางที่เราต้องให้มอเตอร์หมุนไป วงจรที่นำมากระตุ้นนั้นจะมีราคาค่อนข้างจะถูกกว่าและง่ายกว่า ดังในรูปของวงจรการจ่ายไฟ ที่อยู่ด้านบนนั้น เราสามารถเขียนขั้นตอนการทำงานเป็นตารางออกมาได้ดังนี้

| Step No. | Phase 1 | Phase 2 | Phase 3 | Phase 4 |
|---|---|---|---|---|
| 1 | ON | - | - | - |
| 2 | - | ON | - | - |
| 3 | - | - | ON | - |
| 4 | - | - | - | ON |
| 5 | ON | - | - | - |
| 6 | - | ON | - | - |

*รูปที่ 2-6 ขั้นตอนการหมุนของสเต็ปปิ้งมอเตอร์แบบเวฟ*

- **แบบ 2 เฟส (2 Phase)**

เป็นการกระตุ้นขดลวดทีละ 2 ขด ที่อยู่ใกล้กันใน เวลาเดียวกัน และจะเรียงลำดับกันไป จะยกตัวอย่างการกระตุ้นขดลวดในลักษณะเรียงกันไป ให้ดูดังนี้ 12, 23, 34, 41, 12, 23, 34, 41 เรียงลำดับกันไปเรื่อยๆ หรือจะเป็น 14, 43, 32, 21, 14, 43, 32, 21 เรียงกันไปเรื่อยๆเช่นกัน

**ข้อดี** ของการกระตุ้นแบบ 2 เฟส การที่เราจะเพิ่มจำนวนขดลวดที่ถูกกระตุ้นจะทำให้แรงบิดได้มากกว่าแบบเวฟ ซึ่งโรเตอร์จะหมุนด้วยแรง ดึงแบบเต็มๆแรงจาก ทั้ง 2 ขดลวดที่กระตุ้นพร้อมกัน

**ข้อเสีย** แบบ 2 เฟส จะกระตุ้นขดลวดนั้นต้องใช้กำลังไฟมากขึ้นเป็น 2 เท่าของแบบเวฟ ก็เป็นไปตามธรรมชาติ ได้อย่างก็ต้องเสียอย่าง นั้นละครับ

เราสามารถเขียนลำดับการกระตุ้นของขดลวดแบบ 2 เฟส ได้ดังในภาพต่อไปนี้

| Step No. | Phase 1 | Phase 2 | Phase 3 | Phase 4 |
|---|---|---|---|---|
| 1 | ON | ON | - | - |
| 2 | - | ON | ON | - |
| 3 | - | - | ON | ON |
| 4 | - | - | - | ON |
| 5 | ON | ON | - | - |
| 6 | - | ON | ON | - |

*รูปที่ 2-7 ขั้นตอนการหมุนของสเต็ปปิ้งมอเตอร์แบบ 2 เฟส*

- **แบบ 4 เฟส (4 phase)**

แบบ 4-เฟส ในการที่เราจะทำให้สเต็ปปิ้งมอเตอร์แบบ 4-เฟสหมุนตามเข็มหรือทวนเข็มนาฬิกาไปได้ในแต่ละขั้นนั้น เราจะต้องใช้สวิตซ์ 4 ตัว ในการควบคุมในที่นี้จะแทนด้วย SW1, SW2, SW3, และ SW4 สมมุติว่า SW1, SW2 เปิดอยู่ ถ้าเราต้องการที่จะหมุนสเต็ปปิ้งมอเตอร์ไปตามเข็มนาฬิกายังขั้นต่อไป เราก็ต้อง ปิด SW2 และเปิด SW4 และถ้าเราเปลี่ยนมาเปิด SW3 และ SW2 ก็จะทำให้สเต็ปปิ้งมอเตอร์ หมุนไปอีกขั้นหนึ่ง ถ้าต้องการให้สเต็ปปิ้งมอเตอร์หมุนทวนเข็มนาฬิกา ไปหนึ่งขั้นเราก็ต้องทำย้อนขั้นตอนที่กล่าวมาแล้ว ตารางการหมุนของ 4-เฟส สเต็ปปิ้งมอเตอร์ทั้งตาม และทวนเข็มนาฬิกา

| Step No. | Phase 1 | Phase 2 | Phase 3 | Phase 4 |
|---|---|---|---|---|
| 1 | - | - | ON | ON |
| 2 | - | - | - | ON |
| 3 | ON | - | - | - |
| 4 | ON | - | - | - |
| 5 | ON | ON | - | - |
| 6 | - | ON | - | - |
| 7 | - | ON | - | - |
| 8 | - | ON | ON | - |
| 1 | - | - | ON | - |

*รูปที่ 2-8 ขั้นตอนการหมุนของสเต็ปปิ้งมอเตอร์แบบ 4 เฟส*

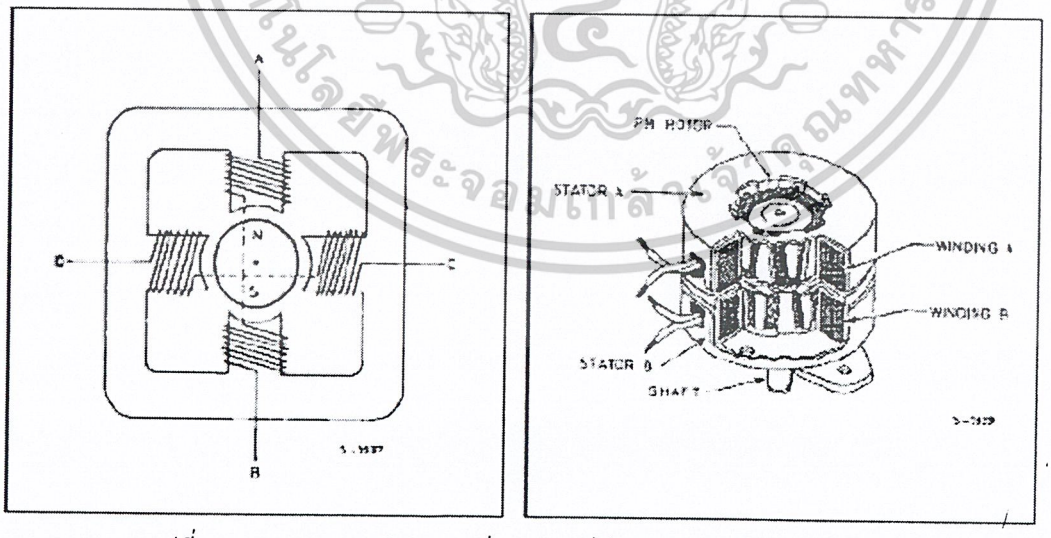
หลังจากที่เราใส่ค่าสวิตช์ต่างๆไปครั้งหนึ่งแล้ว เราจะต้องรออีก 2-3 มิลลิวินาทีก่อนที่จะใส่ค่าสวิตช์ในขั้นตอนต่อไป โดยที่เราสามารถย้อนทำตามขั้นตอนที่แสดงไว้ในตารางข้างต้น จนกว่าจะถึงขั้นที่ต้องการสำหรับสเต็ปปิ้งมอเตอร์ ที่นำมาใช้ในโครงงานนี้เป็นแบบ 4-เฟส จะต้องทำการเคลื่อนที่ 48 step จึงจะครบรอบ 360 องศา หรือ มีการเคลื่อนที่ 7.5 องศาต่อ 1 step

**ข้อดี** การกระตุ้นแบบนี้จะให้แรงบิดที่เพิ่มมากขึ้น เนื่องจากช่วงสเต็ปที่มีระยะสั้นลงอีกประการหนึ่งแต่ละสเต็ปเกิดแรงดึงจากขดลวด 2 ขดที่ถูกกระตุ้นพร้อมกันเป็นผลให้ค่าตำแหน่งความถูกต้องมากขึ้นไปด้วย

**ข้อเสีย** ของการหมุนแบบ 4 เพสก็คงจะเช่นเดียวกับแบบ 2 เฟส ที่ต้องจ่ายกำลังไฟฟ้าเป็น 2 เท่าของแบบเวฟหรือจะใช้เท่ากับแบบ 2เฟส นั้นเอง

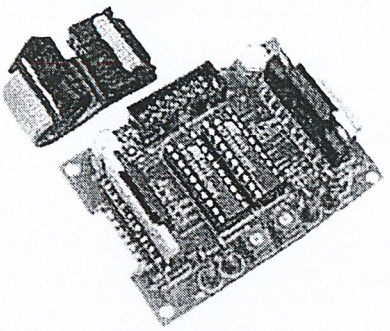
ปัจจุบันสเต็ปมอเตอร์ที่นิยมใช้กันมีอยู่ 2 ชนิดด้วยกัน คือ แบบแม่เหล็กถาวร และแบบที่สามารถเปลี่ยนแปลงความเข้มสนามแม่เหล็กได้ ซึ่งแบบแม่เหล็กถาวรจะใช้งานได้ง่ายกว่า แบ่งออกเป็น 2 ประเภท

- ยูนิโพลาร์ ( Unipolar )
- ไบโพลาร์ ( Bipolar )




*รูปที่ 2-9 ภาพแสดงชนิดของสเต็ปมอเตอร์ซ้ายไบโพล่าร์ ขวายูนิโพล่าร์*

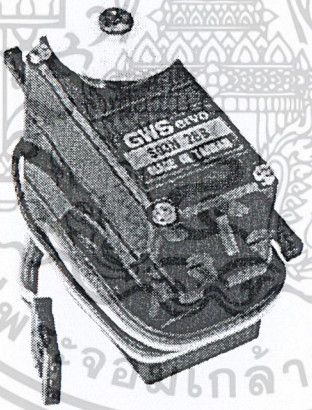
ทั้งสองชนิดมีความแตกต่างที่ขั้วของมอเตอร์ แบบยูนิโพล่าร์จะทำให้เกิดแรงบิดของมอเตอร์น้อยกว่าแบบไบโพล่าร์ ความแตกต่างทางกายภาพคือสายไฟจากมอเตอร์แบบไบโพล่าร์จะมี 4 เส้น และยูนิจะมี 5-6 เส้น



*รูปที่ 2-10 ตัวอย่างวงจรขับเคลื่อนสเต็ปปิ้งมอเตอร์*

โดยจะพบว่าในปัจจุบัน มีการนำสเต็ปมอเตอร์ ไปใช้งานอย่างแพร่หลายอย่างมาก ในหลายด้านด้วยกัน ทั้งนี้เนื่องจากความสะดวกในการควบคุม รวมถึงการขับเคลื่อน สเต็ปมอเตอร์ในปัจจุบันนั้น สามารถใช้ควบคุมมอเตอร์โดยผ่านไอซีได้เลย โดยวงจะขับเคลื่อน ตัวสเต็ปมอเตอร์คู่กับวงจรเพิ่มกระแสเพื่อเพิ่มแรงขับตัวอย่างเช่น ชุดวงจร L297 – L298N โดยสามารถกำหนดแหล่งจ่ายได้ถึง 36 โวลต์

**2.2.2 เซอร์โวมอเตอร์**



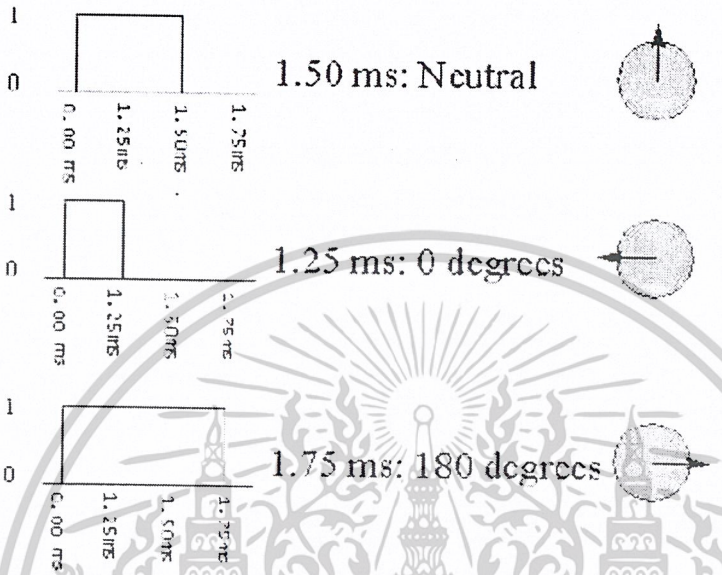

*รูปที่ 2-11 ตัวอย่างเซอร์โวมอเตอร์*

**การทำงานของเซอร์โวมอเตอร์**

เซอร์โวมอเตอร์ เป็นมอเตอร์ทดเฟืองขนาดเล็ก โดยมีแกนส่งกำลัง 1 อัน โดยปกติเซอร์โวมอเตอร์จะหมุนได้เพียง 180 องศา เท่านั้น หรือบางตัวอาจจะถึง 210 องศาขึ้นอยู่กับบริษัทผู้ผลิต และแกนส่งกำลังนี้สามารถที่จะควบคุมทิศทางที่หันไป หรือสามารถระบุตำแหน่งที่ต้องการได้ โดยการส่งรหัสให้กับตัวเซอร์โวมอเตอร์ ทำได้โดยการส่งค่าตำแหน่งของแกนส่งสัญญาณ ดังกล่าวเข้าที่ขาอินพุตตลอดเวลา

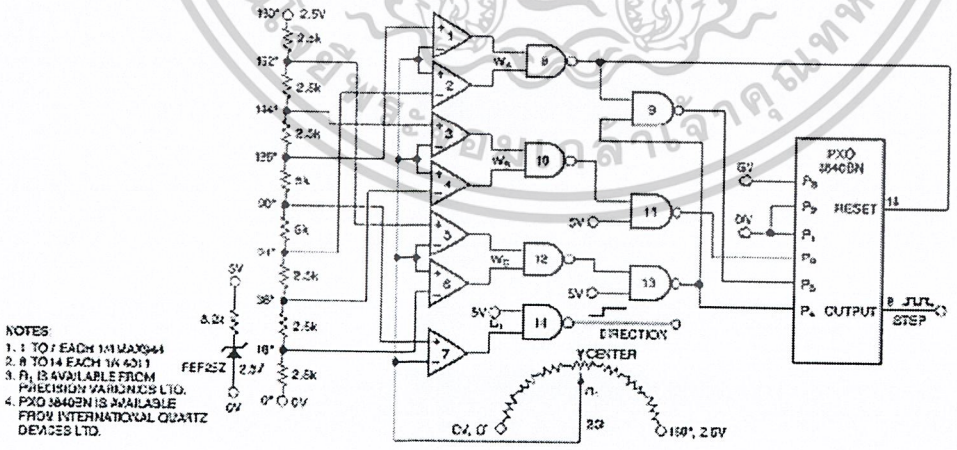
ดังนั้นจะพบได้ว่า เซอร์โวมอเตอร์สามารถนำไปประยุกต์ใช้งานกับอุปกรณ์ได้หลากหลายมาก เพราะการควบคุมมุมและค่าของการหมุนทำได้โดยง่ายและสะดวก เพียงจ่ายค่าสัญญาณ พัลส์เข้าที่ขารับข้อมูล โดยตั้งค่าที่

ตรงตามตำแหน่งที่ต้องการไปเรื่อย ๆ ถ้าแกนอยู่ในมุมที่ถูกต้องมอเตอร์ก็จะปิดเองโดยอัตโนมัติ แต่ถ้าวงจรของมอเตอร์ ตรวจสอบพบว่ามุมยังไม่ถูกต้องมอเตอร์ก็จะหมุนไปในทางที่ถูกต้องจนกระทั่งได้มุมที่ถูกต้อง และจะหยุดรอจนกว่าจะมีการป้อนข้อมูลเข้ามาใหม่ การกำหนดองศาของการหมุนทำโดยการส่งสัญญาณ พัลส์ ที่มีค่าระหว่าง 1.5 มิลลิวินาที ถึง 2.0 มิลลิวินาที ดังรูป




*รูปที่ 2-12 แสดงการควบคุมการทำงานของเซอร์โวมอเตอร์*

จากภาพเป็นการแสดงลักษณะความกว้างของ Pulse สัมพันธ์กับทิศทางที่เซอร์โวมอเตอร์หมุนไปในการควบคุมการหมุนของเซอร์โวมอเตอร์นั้น สามารถทำโดยการสร้างสัญญาณพัลส์ จากไมโครคอนโทรลเลอร์ ไปเข้าสู่เซอร์โวมอเตอร์ ตามตำแหน่งที่ต้องการ



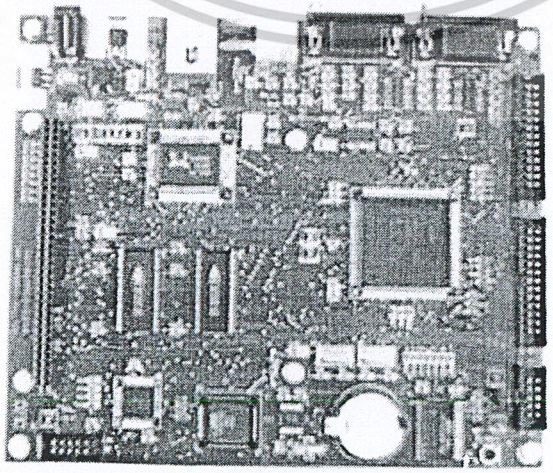

*รูปที่ 2-13 แสดงโครงสร้างภายในของเซอร์โวมอเตอร์*

**2.3 ชุดพัฒนาคอม 86**

ชุดพัฒนาคอม86 เป็นชุดเครื่องมือสำหรับใช้ในการพัฒนาอุปกรณ์ทางด้านระบบฝังตัวที่อยู่บนพื้นฐานการพัฒนาโดยการใช้แพลตฟอร์มที่มีไมโครโพรเซสเซอร์ตระกูล x86 ซึ่งเป็นที่คุ้นเคยกับผู้ใช้งานในปัจจุบันเป็นส่วนประกอบหลักในการทำงาน

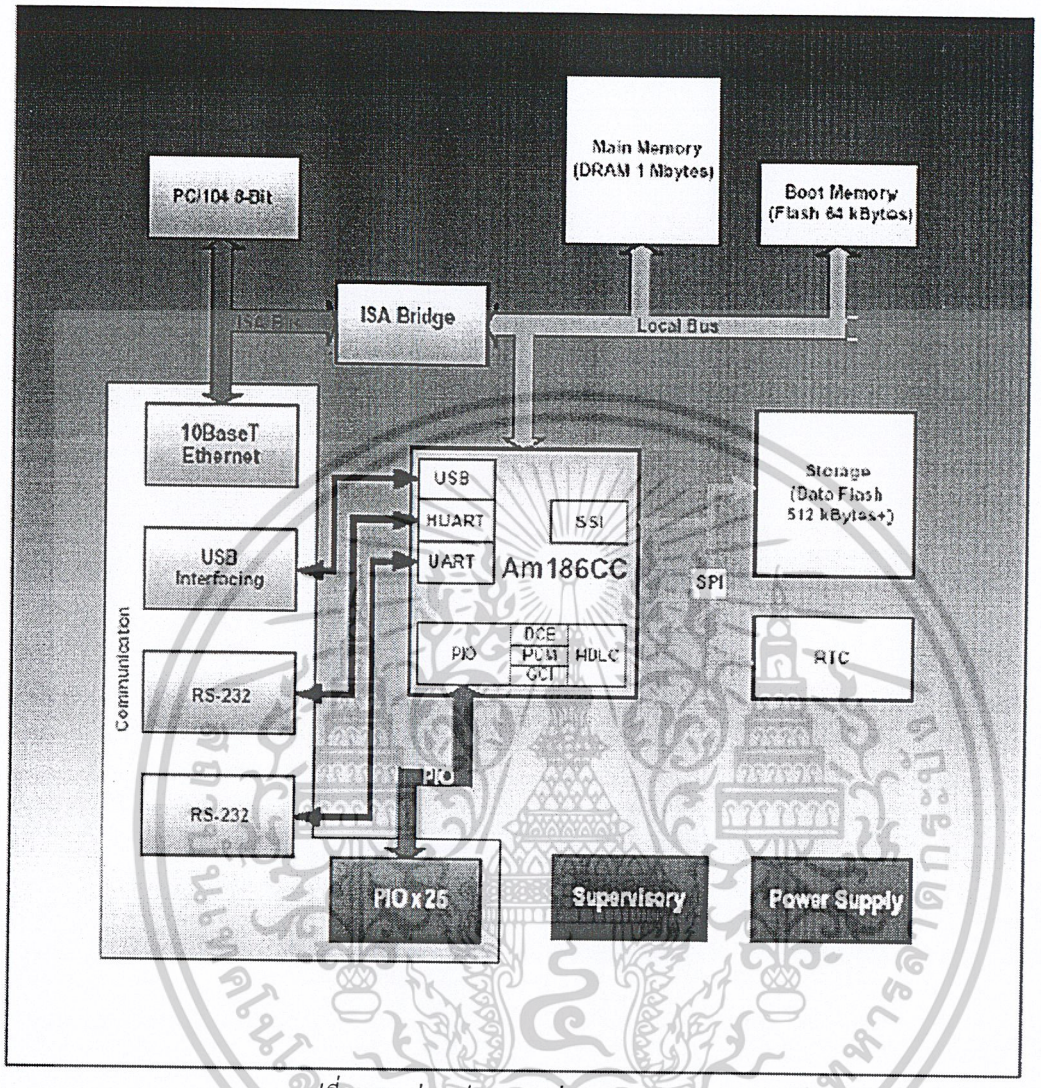
**2.3.1 คุณสมบัติของบอร์ดคอม86** มีคุณสมบัติดังนี้

- ซีพียู เอเอ็มหนึ่งแปดหกซีซี (CPU Am186CC)
- ดีแรม 1 เมกกะไบต์ (1 MBytes DRAM)
- แฟลตไบออส 64 กิโลไบต์ (64 kBytes Flash BIOS)
- หน่วยความจำ 512 กิโลไบต์
- ฟูลดูเพล็ก IEEE 802.3 บนบอร์ดอีเทอร์เน็ตคอนโทรลเลอร์ด้วย RJ45 คอนเน็คเตอร์
- ยูเอสบี คอนเน็คเตอร์
- พอร์ตอนุกรมคู่ อาร์เอส232 ด้วย ดีบี-9 คอนเน็คเตอร์
- 4 ช่องสัญญาณเอชดีเอลซี (HDLC : High level Data Link Control)
- สี่ไทม์สล๊อต (TSAs : Time slot Assigner)
- จีซีไอ คอนโทรลเลอร์
- 3 ไทม์เมอร์ที่สามารถโปรแกรมได้
- นาฬิกา เรียลไทม์
- วอชด็อก ไทม์เมอร์ และ รีเซ็ท คอนโทรลเลอร์
- อินเทอร์รัพ คอนโทรลเลอร์
- 25 อินพุต/เอาท์พุต ที่สามารถโปรแกรมได้
- ไอซ่า บัส ขนาด 8 บิต
- เพาเวอร์ซัพพลาย 3.3 โวลต์ และ 5 โวลต์
- เอลอีดี แสดงสถานะของเพาเวอร์ซัพพลาย และ อีเทอร์เน็ต



*รูปที่ 2-14 บอร์ดคอม 86*

### 2.3.2 สถาปัตยกรรมทางด้านฮาร์ดแวร์



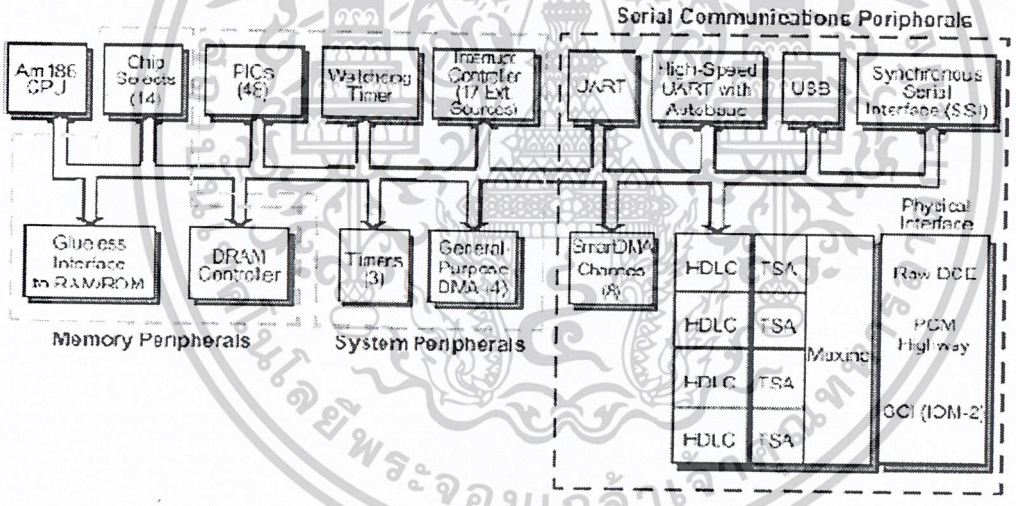

*รูปที่ 2-15 ส่วนประกอบต่างๆของคอม 86*

#### 2.3.2.1 ไมโครคอนโทรลเลอร์ Am186CC

ไมโครคอนโทรลเลอร์ AM186CC ถือว่าเป็นหัวใจหลักที่ควบคุมการทำงานส่วนต่างๆของบอร์ดคอม 86 ซึ่งไมโครคอนโทรลเลอร์ตัวนี้เป็นไมโครคอนโทรลเลอร์ขนาด 16 บิต ในตระกูล อี86 ของบริษัทเอเอ็ม ดีชื่อว่า Am186CC ซึ่งเป็นคอมมิวนิเคชันไมโครคอนโทรลเลอร์ (Communication Microcontroller) ที่ใช้ แกนหลัก (core) ของ 80186 โดยได้เพิ่มความสามารถทางด้านการสื่อสาร และส่วนประกอบอื่นๆ โดยมี ส่วนสำคัญที่ได้เพิ่มเข้าไปดังนี้

- ส่วนเอชดีแอลซี (HDLC) หรือ ไฮเลเวลดาต้าลิ้งค์คอนโทรล (High–level Data Link Control) สำหรับเป็นส่วนจัดการเรื่องการติดต่อสื่อสารแบบ เอชดีแอลซี ที่สามารถเชื่อมต่อกับ ดีซีอี, พีซีเอ็ม ไฮเวย์ และ จีซีไอ ผ่านทางพีไอโอโดยมีทีซ่า (TSAs) หรือ ไทม์สลอตแอสซายเนอร์ (Time Slot Assigners) สำหรับการทำงานในแบบมัลติเพลกซ์ (Multiplex)
- ยูเอสบีเพอร์ริพารอลคอนโทรล (USB Peripheral Controller) สำหรับการทำเป็นอุปกรณ์ยูเอสบี

- เอชยูอาร์ต (HUART) หรือ ไฮสปีดยูอาร์ต (High-speed UART) สำหรับใช้งานในการเชื่อมต่อแบบอนุกรมความเร็วสูง โดยสามารถสื่อสารได้ในอัตราเร็วสูงสุด 460 กิโลบิตต่อวินาที (Kbit /s) สามารถตรวจสอบอัตราเร็วในการเชื่อมต่อได้อัตโนมัติ (Automatic Baud Rate Detection) และมีฮาร์ดแวร์แฮนชาร์คกิ้ง(Handshaking)
- ยูอาร์ต (UART) หรือ ยูนิเวอร์ซัลดซิงโคนัสรีซีพเวอร์/ทรานส์มิทเทอร์ (Universal Asynchronous Receiver/Transmitter) สำหรับการเชื่อมต่อแบบอนุกรมโดยสามารถสื่อสารได้ในอัตราเร็วสูงสุด 115.2 กิโลบิตต่อวินาที และมีฮาร์ดแวร์แฮนชาร์คกิ้ง
- เอสเอสไอ (SSI) หรือ ซิงโคนัสซีเรียลอินเทอร์เฟส (Synchronous Serial Interface) สำหรับการเชื่อมต่อ สื่อสารแบบอนุกรมความเร็วสูง 25 เมกกะบิตต่อวินาที (Mbit/s) ซึ่งสามารถสื่อสารแบบ
- โปรแกรมเมเบิลไทม์เมอร์ (Programmable Timer) ขนาด 16 บิต 3 ตัว
- ฮาร์ดแวร์วอชด็อกไทม์ (Hardware Watchdog Time)
- ดีแรมคอนโทรลเลอร์ (DRAM Controller)
- อินเทอร์รัพคอนโทรลเลอร์ (Interrupt Controller) 36 มาสก์เอเบิลอินเทอร์รัพ
- โปรแกรมเมเบิลอินพุทเอาท์พุทซิกแนล (Programmable I/O Signal) ซึ่งมีจำนวน 48 แชนแนล




*รูปที่ 2-16 องค์ประกอบของไมโครคอนโทรลเอลร์AM186CC*

ไมโครคอนโทรลเลอร์ Am186CC นี้ เป็นไมโครคอนโทรลเลอร์ที่ที่มีสถาปัตยกรรมแบบ x86 มีชุดคำสั่งที่เข้ากันได้กับชุดคำสั่งของ 80286 (Real Mode) สนับสนุนการอ้างอิงหน่วยความจำหลักได้ถึง 1 เมกกะไบต์

**2.3.2.2 หน่วยความจำ (Memory)**

ภายในบอร์ดคอม86 ประกอบด้วยหน่วยความจำหลายส่วนประกอบกันซึ่งมีรายละเอียดต่าง ๆ ดังนี้

**1. แรม (RAM)**

หน่วยความจำส่วนนี้เป็นส่วนที่ใช้ในการทำงานของโปรแกรมต่างๆ ของบอร์ดคอม86 ซึ่งมีขนาด 1 เมกกะไบต์ อันเนื่องมาจากจากสถาปัตยกรรมของไมโครคอนโทรลเลอร์หน่วยความจำในส่วนนี้จะใช้ เป็นพื้นที่ใน

การใช้งานสำหรับโปแกรมต่างๆทั้งระบบปฏิบัติการและโปรแกรมของนักพัฒนา โดยหน่วย ความจำนี้เป็นหน่วยความจำชนิดอีดีโอ (EDO) ขนาด 1 เมกกะไบต์โดยมีการควบคุมการทำงานจากส่วนของดีแรมคอนโทรลเลอร์ภายในไมโครคอนโทรลเลอร์

**2. แฟลชไบออส**

หน่วยความจำส่วนนี้เป็นหน่วยความจำแบบแฟลชขนาด 64 กิโลไบต์สำหรับใช้ในการเก็บโปรแกรมไบออสสำหรับควบคุมการทำงานพื้นฐานของระบบ หน่วยความจำในส่วนนี้จะถูกเรียกใช้งานเป็นอันดับแรกเมื่อระบบเริ่มทำงาน

**3. ซีเรียลดาต้าแฟลช (Serial DataFlash)**

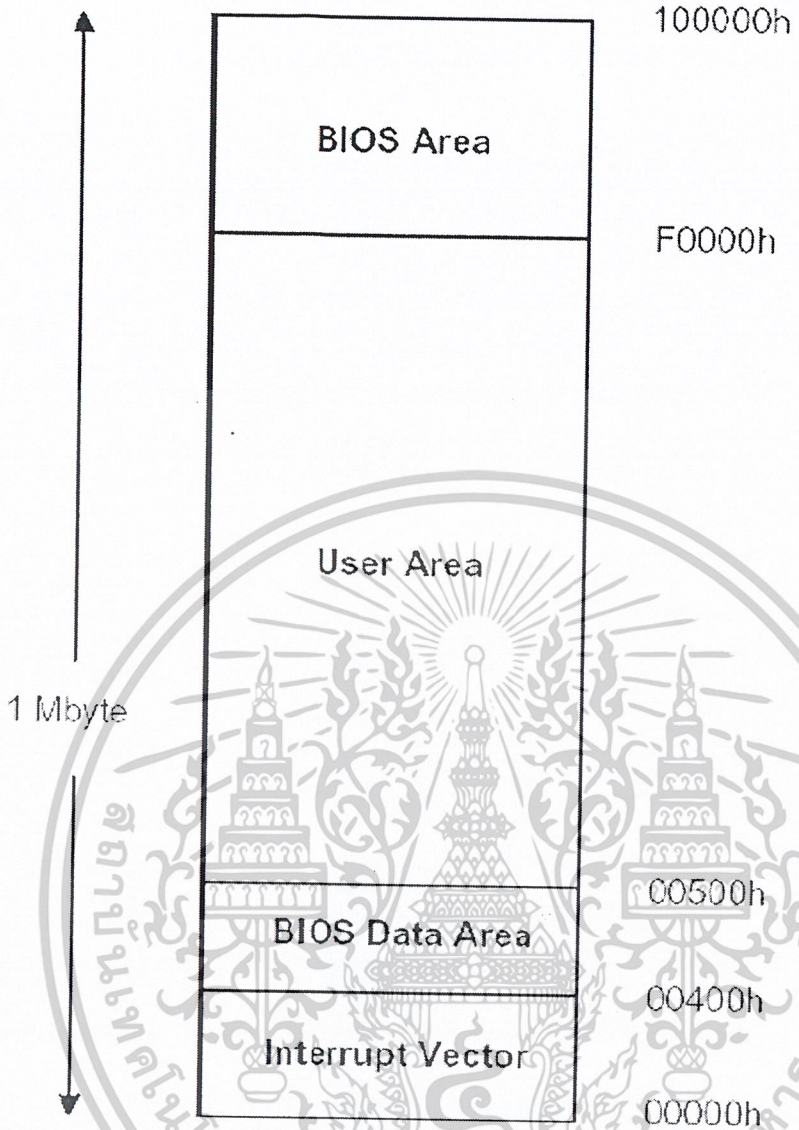
หน่วยความจำแบบแฟลชตัวนี้จะทำงานทำหน้าที่เป็นสตอเรท (Storage) สำหรับใช้ในการเก็บข้อมูลต่างๆ ของระบบที่ไม่ต้องการใช้เกิดการสูญหายในขณะที่บอร์ดคอม86 ขาดกระแสไฟฟ้าหล่อเลี้ยงเช่น ข้อมูลจากการตรวจวัดต่างๆ, โปรแกรมแอพพลิเคชั่นต่างๆ รวมไปถึงระบบปฏิบัติการด้วย ซึ่งบนบอร์ด คอม86 นี้ได้ติดตั้ง ดาต้าแฟลช (IC U12) เบอร์ เอที45ดีบี041ขนาด 512 กิโลไบต์สำหรับใช้งานในส่วนนี้ และได้ออกแบบให้สามารถขยายขนาดได้ตามความต้องการ ซึ่งในการใช้งานนักพัฒนาจะมองหน่วยความจำในส่วนนี้เปรียบเสมือนดิสก์ไดรฟ์อันหนึ่ง (ไดรฟ์ A) ของเครื่องคอมพิวเตอร์

**4. แผนผังหน่วยความจำ**

จากหน่วยจำทั้งหมดที่กล่าวมาในข้างต้นสามารถแสดงเป็นภาพการจัดวางของหน่วยความจำทั้งหมดได้ดังนี้คือ

- หน่วยความจำหลัก

หน่วยความจำในส่วนนี้มีขนาด 1 เมกกะไบต์ สำหรับเป็นพื้นที่ใช้งานของโปรแกรมต่างๆซึ่งมีการจัดแบ่งดังนี้

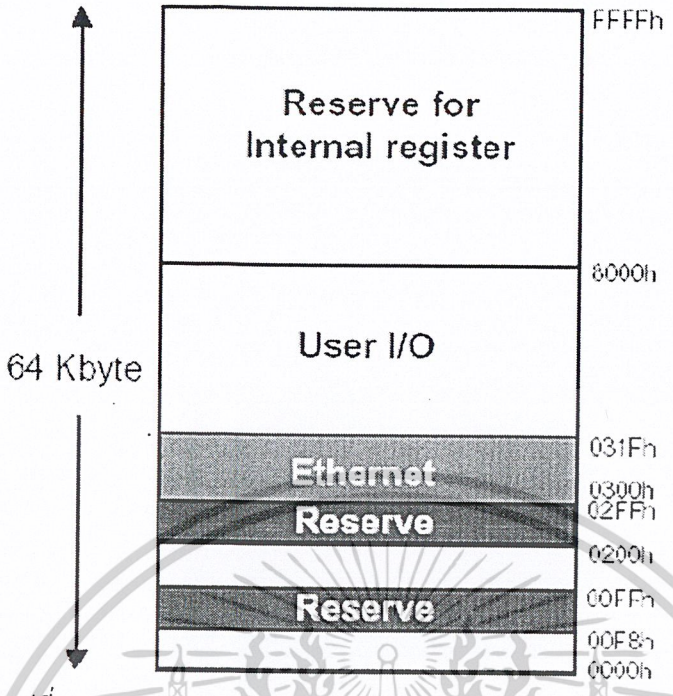



*รูปที่ 2-17 การแบ่งการใช้งานหน่วยความจำหลัก*

จากภาพหน่วยความจำช่วงแรกคือจากตำแหน่ง 00000h จนถึงตำแหน่ง 003FFh เป็นส่วนของอินเทอร์รัพเวกเตอร์เทเบิล (Interrupt Vector Table) สำหรับรองรับการทำงานของการอินเทอร์รัพ (Interrupt) ที่เกิดขึ้น ส่วนต่อมาคือช่วงตำแหน่ง 00400h ถึง 004FFh หน่วยความจำ ส่วนนี้ใช้สำหรับเก็บข้อมูลของไบออส (BIOS) ส่วนถัดมาคือช่วงตำแหน่ง 00500h ถึง F0000h ส่วนนี้จะเป็นพื้นที่ สำหรับใช้งาน โดยโปรแกรมทั่วไปทั้งระบบปฏิบัติการและโปรแกรมของนักพัฒนาที่สร้างขึ้นมา ส่วนที่เหลือในช่วงตำแหน่ง F0000h ถึง FFFFFh ส่วนนี้จะเป็นพื้นที่ที่ถูกสงวนไว้สำหรับไบออสของบอร์ดคอม86

- หน่วยความจำที่เป็นส่วนอินพุท/เอาท์พุท (I/O)

ส่วนนี้คือพื้นที่สำหรับใช้ในการติดต่อกับอุปกรณ์อินพุท/เอาท์พุทต่างๆ และใช้ในการติดต่อกับรีจิสเตอร์ภายในของไมโครคอนโทรลเลอร์ ซึ่งมีการจัดวางดังภาพ




*รูปที่ 2-18 การจัดวางหน่วยความจำในส่วนของอินพุตท์เอาท์พุต*

จากภาพจะแบ่งตำแหน่งของหน่วยความจำได้ออกเป็น 2 ส่วนหลักคือส่วนบนตั้งแต่ตำแหน่งที่ 8000h ถึงตำแหน่ง FFFFh ส่วนนี้จะเป็นพื้นที่สำหรับการใช้งานโดยตัวบอร์ดคอม86 และเป็นส่วนที่ใช้ในการติดต่อรีจิสเตอร์ต่างๆของไมโครคอนโทรลเลอร์ซึ่งบริเวณนี้จะถูกสงวนไว้ไม่สามารถนำมาใช้งานได้ ส่วนที่สองคือส่วนของหน่วยความจำส่วนล่างคือ ตั้งแต่ตำแหน่ง 0000h ถึงตำแหน่ง7FFFh บริเวณนี้จะเป็นพื้นที่สำหรับให้นักพัฒนานำส่วนที่ว่างไปใช้งานโดยเว้นส่วนที่สงวนไว้ใช้งานสำหรับตัวไมโครคอนโทรลเลอร์คือ ตำแหน่ง 00F8h ถึงตำแหน่ง 00FFh และส่วนที่สงวนไว้สำหรับการใช้งานของวงจรอีเทอร์เน็ตคือในตำแหน่ง 0300h จนถึง 031Fh บริเวณที่เหลือนอกจากที่กล่าวมานักพัฒนาสามารถนำไปใช้งานได้ตามต้องการ

**2.3.2.3 พาวเวอร์ซัพพลาย (Power supply)**

ระบบจ่ายไฟของบอร์ดคอม86 แบ่งได้ออกเป็น 2 ส่วนหลักด้วยกันคือ ในส่วนแรกจะเป็นระบบจ่ายไฟแรงดัน 5 โวลท์ (Volt) ซึ่งจะมีไอซีแอลดีโอ LDO (U14) ทำหน้าที่จ่ายกระแสไฟฟ้าสำหรับเลี้ยงอุปกรณ์บนบอร์ดที่ต้องการแรงดันไฟฟ้าในการทำงานที่ 5 โวลท์ เช่น ดีแรม, แฟลชไบออส, เรียลไทม์คล็อก เป็นต้น และสำหรับอุปกรณ์ที่ต้องการแรงดันไฟฟ้า 3.3 โวลท์ ในการทำงานนั้นจะมีไอซีแอลดีโอ LDO (U16) ทำหน้าที่จ่ายกระแสไฟฟ้าให้ซึ่งอุปกรณ์เหล่านี้ได้แก่ตัวไมโครคอนโทรลเลอร์, แฟลชดิสก์เป็นต้น ซึ่งอินพุทของภาคพาวเวอร์ซัพพลายของบอร์ดคอม86 นั้นจะรับเป็นไฟฟ้ากระแสตรงที่แรงดันประมาณ 8 โวลท์ และทำการลดระดับแรงดันลงเป็น 5 โวลท์และ 3.3 โวลท์ตามลำดับ โดยภายในชุดพัฒนาจะมีอแดปเตอร์ให้ 1 ตัวสำหรับเป็นส่วนจ่ายกระแสไฟฟ้าให้กับวงจรทั้งหมด

โดยทั่วไปส่วนวงจรจ่ายกระแสไฟฟ้าแรงดัน 5 โวลท์และ 3.3 โวลท์จะสามารถจ่ายกระแสไฟฟ้าให้กับวงจรที่ต่อเพิ่มเติมได้อีก 300 มิลลิแอมป์ (mA) หากมีการต่อวงจรส่วนขยายที่ต้องการใช้งานกระแสไฟฟ้ามากกว่า

นี้ แนะนำให้ควรมีการต่อวงจรพาวเวอร์ซัพพลายจากภายนอกเพิ่มเติมเพื่อป้องการเสียหายของ ส่วนวงจรภาคจ่ายไฟของบอร์ดคอม86

**2.3.2.4 แอลอีดีอินดิเคเตอร์ (LED Indicator)**

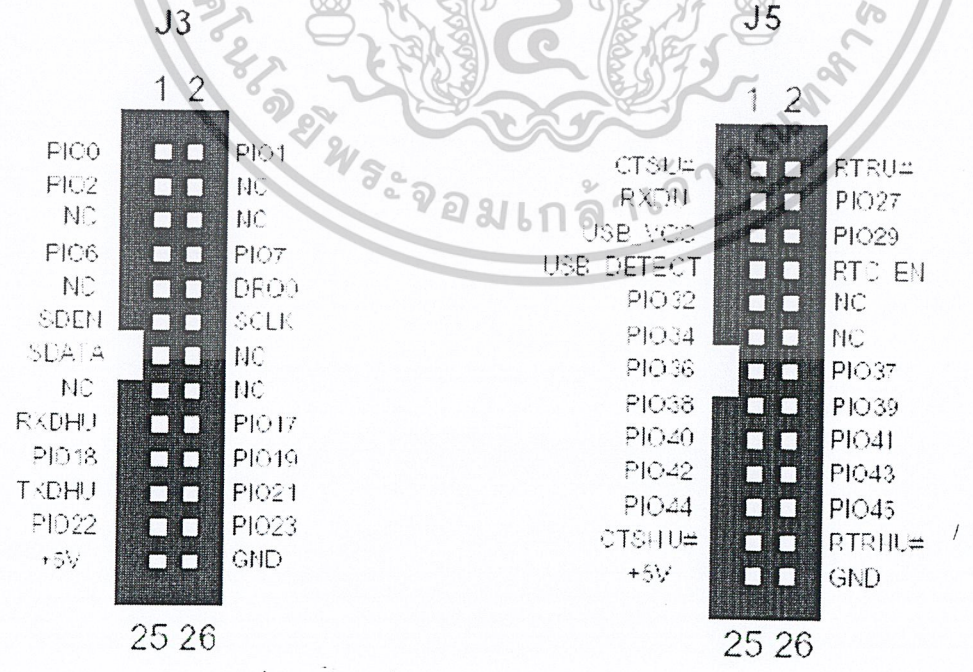
บนบอร์ดคอม86 จะมีแอลอีดี (LED) สำหรับแสดงสถานะต่างๆดังนี้คือ ดวงแรกเป็นแอลอีดีของพาวเวอร์ซัพพลาย (D4) มีสีแดงโดยจะติดสว่าง เมื่อบอร์ดคอม86 ได้รับการจ่ายกระแสไฟฟ้าจากภายนอก และวงจรพาวเวอร์ซัพพลาย 5 โวลต์ และ 3 โวลต์ สามารถทำงานได้อย่างถูกต้อง ดวงที่ 2 คือ LED สีเขียวของส่วนวงจรอีเทอร์เน็ต(D2) ซึ่งดวงนี้จะติดสว่างขึ้นเพื่อแสดงว่าการมีการเชื่อมต่อสายอีเทอร์เน็ตและพร้อมที่จะรับส่ง ข้อมูล ดวงที่ 3 คือ Lแอลอีดีดวงสีเขียวของส่วนวงจรอีเทอร์เน็ตที่เหลืออีกดวงหนึ่ง (D1) ซึ่งดวงนี้จะติดกระพริบเมื่อมีการรับส่งข้อมูลผ่านทางอีเทอร์เน็ตพอร์ต .

**2.3.2.5 เอ๊กแพลนชั่นพอร์ต (Expansion Port)**

ในการใช้งานชุดพัฒนาต่างๆสำหรับพัฒนาอุปกรณ์ทางด้านระบบฝังตัวนั้น เป็นธรรมดาที่นักพัฒนา ย่อมอาจเกิดความต้องการที่จะออกแบบวงจรส่วนขยายต่ออุปกรณ์ต่างๆ เพิ่มเติมจากชุดพัฒนาพื้นฐานที่มีใช้งานอยู่ ซึ่งสำหรับบอร์ดคอม86 นี้ได้จัดเตรียมออกแบบการเชื่อมต่อใช้งานอุปกรณ์ภายนอกในรูปแบบต่างๆ ดังนี้

**1. พีไอโอ (PIO)**

ในส่วนของพีไอโอ นี้จะเป็นลักษณะการต่อใช้งานเหมือนกับที่ใช้งานภายในไมโครคอนโทรลเลอร์โดยทั่วไปซึ่งสามารถโปรแกรมบังคับควบคุมการใช้งานได้เป็นอิสระจากซึ่งกันและกันโดยบนบอร์ดคอม 86 นี้ มีพีไอโอให้นักพัฒนาได้ใช้งานทั้งสิ้นจำนวน 25 แชนแนล (Channel) จากทั้งหมด 48 แชนแนลที่มีในไมโครคอนโทรลเลอร์เนื่องจากการที่สัญญาณพีไอโอบางสัญญาณถูกนำไปใช้งานในลักษณะของการมัลติเพล็กกับสัญญาณอื่นของบอร์ดคอม86 ซึ่งพีไอโอที่ใช้งานได้บนบอร์ดคอม86 และการเชื่อมต่อมายังคอนเน็คเตอร์ของพีไอโอต่างๆ เป็นไป ดังภาพ

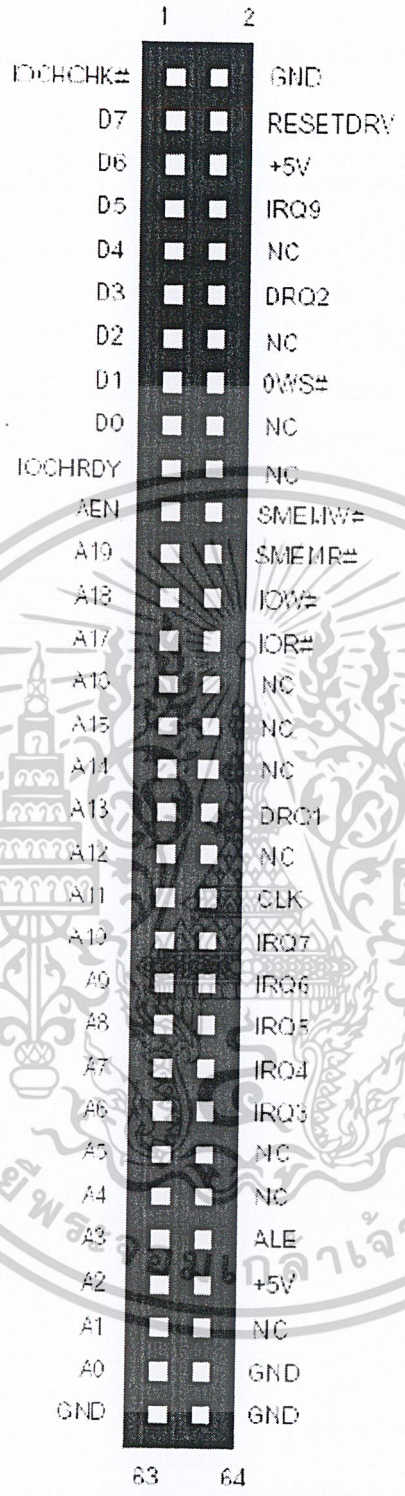



*รูปที่ 2-20 การเชื่อมต่อ PIO ของบอร์ดคอม 86*

จากภาพในส่วนที่เป็นชื่อของสัญญาณอื่นที่มิใช่พีไอโอหมายเลขต่างๆ นั้นคือ พีไอโอ ที่ถูกใช้งานในรูปแบบของสัญญานอื่นที่มัลติเพล็กซ์ (Multiplex) อยู่และสามารถนำมาใช้งานเป็นพีไอโอได้ในกรณีที่ไม่ต้องการใช้งาน เช่นขา CTSHU# ถ้าไม่ต้องการใช้งานส่วนของ โฟลว์คอนโทรล (Flow Control) ของพอร์ต คอม1 ก็สามารถนำมาใช้งานเป็น พีไอโอ46 ได้ เอ็นซี คือขาพีไอโอที่ถูกนำไปใช้ในลักษณะของสัญญาณอื่นที่มัลติเพล็กซ์อยู่และไม่สามารถนำมาใช้งานได้ ซึ่งไม่มีการนำมาเชื่อมต่อให้ไว้ที่คอนเน็คเตอร์ส่วนขาที่เหลือจะเป็นพีไอโอที่สามารถใช้งานได้

**2. ไอซ่า (ISA)**

เป็นรูปแบบของการเชื่อมต่อกับอุปกรณ์แบบขนาน โดยได้ออกแบบให้มีลักษณะการเชื่อมต่อใช้งานคล้ายกับการเชื่อมต่อใช้งานของไอซ่าบัส (ISA Bus) ของคอมพิวเตอร์โดยทั่วไปแตกต่างกันเพียงรูปแบบการจัดวางอุปกรณ์ต่างๆ ที่พยามให้จัดวางในรูปลักษณะของมาตรฐาน พีซี/104 ซึ่งการออกแบบวงจรต่างๆ ที่จะนำมาเชื่อมต่อนั้นสามารถออกแบบในลักษณะเดียวกันกับการออกแบบการ์ดไอซ่า ของเครื่องคอมพิวเตอร์โดยทั่วไป จะมีคอนเนคเตอร์ เจ4 สำหรับใช้ในการเชื่อมต่อกับการ์ดขยายภายนอกที่จะนำมาเชื่อมต่อ ซึ่งการเชื่อมต่อของสัญญาณต่างๆ ของคอนเน็คเตอร์ เจ4 จะเป็นไปดังภาพ




*รูปที่ 2-21 ไอซ่าคอนเนคเตอร์ (ISA Connector)*

**2.3.3 สถาปัตยกรรมทางด้านซอฟต์แวร์**

- BIOS เป็นโปรแกรมที่คอยควบคุมการทำงานของฮาร์ดแวร์ต่างๆบนบอร์ดคอม86 เป็นส่วนที่จัดการการสื่อสารระหว่างฮาร์ดแวร์กับแอพพลิเคชันซอฟท์แวร์ และระบบปฏิบัติการ
- Driver เป็นโปรแกรมขนาดเล็กที่ทำหน้าที่ติดต่อสื่อสารกับฮาร์ดแวร์ อื่นๆที่ไบออสไม่ได้รองรับการทำงานไว้ เช่น ฮาร์ดแวร์ที่สร้างขึ้นมาเพิ่มเติมเพื่อเชื่อมต่อเข้ากับบอร์ดคอม86
- Operating System ระบบปฏิบัติการเป็นโปรแกรมที่ควบคุมการทำงานของโปรแกรมต่างๆในระบบทั้งในส่วนของการจัดการหน่วยความจำ เป็นส่วนสื่อสารระหว่างโปรแกรมแอพพลิเคชันของผู้ใช้และ ส่วนของฮาร์ดแวร์ระบบต่าง ๆ
- Packet Driver เป็นโปรแกรมจัดการเกี่ยวกับแพ็กเก็ตของการส่งข้อมูล สำหรับแอพพลิเคชันทางด้านการสื่อสาร
- TCP/IP เป็นโปรแกรมจัดการทางด้านโพรโตคอลทีซีพี/ไอพีสำหรับแอพลิเคชันทางด้านการสื่อสาร
- Network Application แอพพลิเคชั่นทางด้านการสื่อสาร เป็นโปรแกรมแอพพลิเคชันทางด้านการสื่อสาร ต่างๆ เช่น โปรแกรมเว็บเซิร์ฟเวอร์ในระบบฝังตัว (Embedded web server) เป็นต้น
- Application เป็นโปรแกรมแอพลิเคชันอื่นๆ ที่ไม่ได้ใช้ความสามารถทางด้านการสื่อสาร เช่นโปรแกรมควบคุมการทำงานของหุ่นยนต์ หรือเครื่องจักร เป็นต้น

**2.3.4 จุดเด่นของคอม86**

**2.3.4.1 ใช้ไมโครคอนโทรลเลอร์ที่มีความสามารถสูง**

ภายในชุดพัฒนาคอม86 มีบอร์ดคอม86 ซึ่งได้เลือกใช้ไมโครคอนโทรลเลอร์ Am186CC ของบริษัทเอเอ็มดี (AMD) ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 16 บิตทำงานอยู่ที่ความเร็ว 24 เมกกะเฮิร์ต (MHz) ทำให้มีความสามารถในการประมวลผลต่างๆเพิ่มมากขึ้นเมื่อเทียบกับการพัฒนาโดยใช้ไมโครคอนโทรลเลอร์ขนาด 8 บิตที่นิยมใช้กันอยู่ในปัจจุบัน นอกจากนี้ภายในตัวไมโครคอนโทรลเลอร์เองยังได้รวมเอาอุปกรณ์พื้นฐานต่างๆที่จำเป็นในการใช้งาน เช่น ดีแรมคอมโทรลเลอร์ (DRAM controller), อินเทอร์รัพต์คอนโทรลเลอร์ (Interrupt controller), ยูอาร์ต (UART) เข้าไว้ภายในตัวไมโครคอนโทรลเลอร์ซึ่งทำให้ง่ายต่อการออกแบบอุปกรณ์เพื่อทำเป็นสินค้าภายหลังการพัฒนาตัวอย่างเช่น การนำบอร์ดคอม86 ไปพัฒนาเป็นอุปกรณ์อย่างหนึ่งที่ต้องการควบคุมการทำงานผ่านทางยูเอสบีพอร์ต (USB port) จะสามารถทำได้โดยง่ายเนื่องจากไม่ต้องการออกแบบวงจรควบคุมการทำงานในส่วนยูเอสบี นี้เพิ่มอีก เพียงแต่ออกแบบวงจรการทำงานส่วนอื่นที่ต้องการเพิ่มเติมทำให้ทุ่นเวลาในการพัฒนา เป็นต้น นอกจากนี้ภายในไมโครคอนโทรลเลอร์ยังเพิ่มส่วนโมดูล (Module) ทางด้านการสื่อสารต่างๆเข้าไปภายในตัว ทำให้เป็นไมโครคอนโทรลเลอร์ที่มีความสามารถเด่นทางด้านการสื่อสารเหมาะกับการพัฒนาอุปกรณ์ที่ต้องใช้คุณสมบัติการสื่อสารต่างๆ ซึ่งจะสามารถทำได้ง่ายเมื่อเทียบกับการใช้ไมโครคอนโทรลเลอร์ขนาดเล็กแบบเดิมซึ่งมีข้อจำกัดต่างๆ อยู่มากมาย

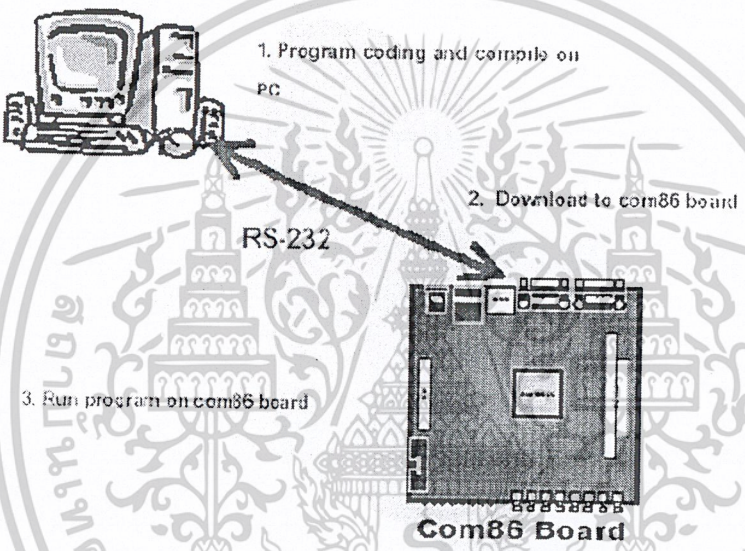
**2.3.4.2 มีเครื่องมือในการพัฒนาโปรแกรมมาก**

เนื่องจากการพัฒนาโปรแกรมโดยใช้ชุดพัฒนาคอม86 นั้นเป็นเสมือนกับการพัฒนาโปรแกรมบนของคอมพิวเตอร์ส่วนบุคคลทั่วไป ดังนั้นจึงสามารถนำโปรแกรมเครื่องมือต่างๆที่ใช้พัฒนาโปรแกรมสำหรับ

คอมพิวเตอร์ส่วนบุคคลมาใช้งานได้มากมาย และโปรแกรมเครื่องมือเหล่านี้ก็เป็นที่คุ้นเคยของนักพัฒนาจึงทำให้สามารถพัฒนาอุปกรณ์ได้อย่างรวดเร็ว ลดต้นทุนในการพัฒนาต่างๆได้เป็นจำนวนมากนอกจากนี้ภายในชุดพัฒนายังมีชุดซอฟต์แวร์เครื่องมือต่างๆที่จำเป็นสำหรับช่วยในการพัฒนาโปรแกรมด้วยอีกทางหนึ่ง

**2.3.4.3 ง่ายในการเรียนรู้และพัฒนา**

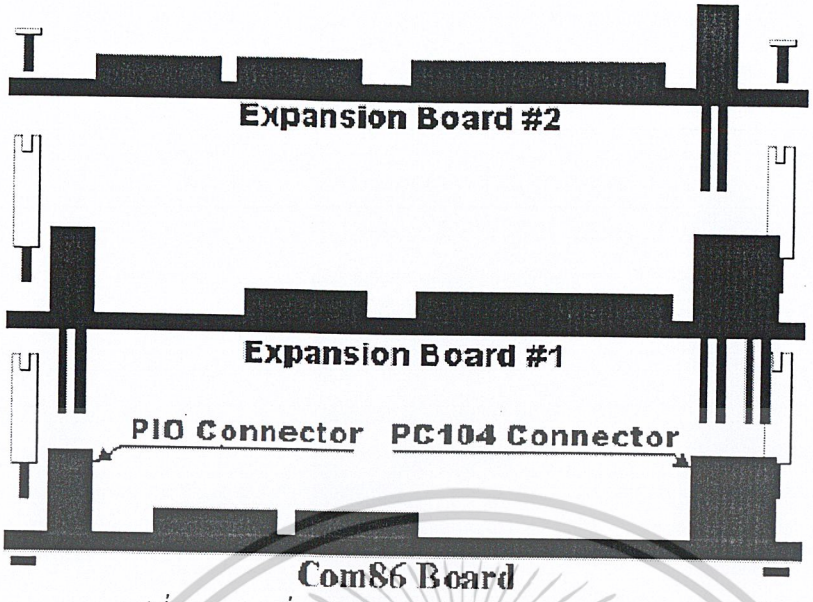
บอร์ดคอม86 ภายในชุดพัฒนาคอม86 นี้มีสถาปัตยกรรมของไมโครคอนโทรลเลอร์แบบ x86 ซึ่งเป็นสถาปัตยกรรมเดียวกับคอมพิวเตอร์ส่วนบุคคลทั่วไป โดยมีชุดคำสั่งที่เข้ากันได้กับชุดคำสั่งของ 80286 (Real mode) ทำให้ง่ายต่อการเรียนรู้ในการพัฒนาโปรแกรมต่างๆ และยังสามารถพัฒนาและทดสอบซอฟต์แวร์ต่างๆได้บนคอมพิวเตอร์ ก่อนที่จะทดสอบกับบอร์ดคอม86 ทำให้สามารถพัฒนาอุปกรณ์ทางด้านระบบฝังตัวต่างๆได้อย่างรวดเร็ว ซึ่งรูปแบบการพัฒนามีลักษณะเป็นไป ดังภาพ




*รูปที่ 2-23 รูปแบบการพัฒนาโปรแกรมโดยใช้คอม 86*

**2.3.4.4 มีความสามารถในการเพิ่มขยายได้**

ในส่วนของการเพิ่มขยายของฮาร์ดแวร์ (Hardware) สามารถทำได้โดยผ่านทางคอนเนคเตอร์ (Connector) ของบอร์ดในรูปแบบของมาตรฐาน (พีซีหนึ่งศูนย์สี่) PC/104 ซึ่งจะมีการเชื่อมต่อแบบบัสอินเทอร์เฟส (BUS Interface) แบบไอเอสเอบัส (ISA BUS) ขนาด 8 บิต คล้ายกับไอเอสเอบัส ที่มีอยู่ในคอมพิวเตอร์ส่วนบุคคลทั่วไป ซึ่งผู้พัฒนาสามารถออกแบบฮาร์ดแวร์ให้เป็นลักษณะของบอร์ดขยายและนำมาเชื่อมต่อกับบอร์ดคอม86 และเพียงเขียนซอฟต์แวร์ไดร์เวอร์ (Software Driver) สำหรับควบคุมการทำงานของส่วนเพิ่มเข้ามาใหม่ซึ่งจะทำงานร่วมกับระบบปฏิบัติการในการรองรับการทำงานของแอพพลิเคชันต่างๆที่เพิ่มเข้ามา นอกจากนี้ตัวบอร์ดคอม86 ยังสามารถเชื่อมต่อกับอุปกรณ์ภายนอกในลักษณะของพีไอโอ (PIO) หรือ โปรแกรมเมเบิลอินพุท/เอาท์พุท (Programmable Input/Output) เหมือนกันการใช้งานไมโครคอนโทลเลอร์รุ่นอื่นๆได้ด้วยเช่นกัน




*รูปที่ 2-24 การเชื่อมต่อของบอร์ดขยายกับบอร์ดคอม 86*

**2.3.4.5 แอพพลิเคชันเป้าหมายของชุดพัฒนา (Target Application)**

จากความสามารถของไมโครคอนโทรลเลอร์ของบอร์ดคอม86 ทำให้การประยุกต์ใช้งานกับแอพลิเคชั่นต่างๆนั้น สามารถทำได้มากมาย โดยตัวอย่างแอพพลิเคชันต่างๆที่เหมาะแก่การพัฒนาโดยบอร์ดคอม 86 มีดังนี้

1. เกี่ยวกับอุตสาหกรรม ( Industrial control and Automation)

ตัวอย่างของอุปกรณ์ควบคุมภายในโรงงาน เช่น การควบคุมเครื่องจักรจากระยะไกล การตรวจวัดสภาวะต่างๆ ภายในโรงงานและส่งกลับไปยังศูนย์ควบคุมโดยการใช้บอร์ดคอม86 เป็นอุปกรณ์ที่ช่วยทำหน้าที่ในการสื่อสาร ควบคุมสภาวะแวดล้อม หรือเครื่องจักรต่างๆภายในโรงงานและรายงานกลับไปยังศูนย์กลางตามกำหนดเวลา เป็นต้น

2. อุปกรณ์ยูเอสบี (USB Peripheral)

อุปกรณ์ยูเอสบีต่างๆ สามารถพัฒนาได้โดยการใช้บอร์ดคอม86 ซึ่งภายในบอร์ดคอม86 นี้มียูเอสบีคอนโทรลเลอร์ภายในตัวไมโครคอนโทรลเลอร์ Am186CC สามารถเชื่อมต่อเข้ากับคอมพิวเตอร์ได้ที่ความเร็ว 12 เมกกะบิตต่อวินาที (Mbps) ซึ่งการพัฒนาเป็นอุปกรณ์ ยูเอสบี ต่างๆ โดยบอร์ดคอม86 นี้สามารถทำได้โดยไม่ต้องมีการต่อวงจรทางด้านยูเอสบีใดๆ เพิ่มเติมอีก

3. เว็บเซิร์ฟเวอร์ (Embedded web server)

เอ็มเบดเดดเว็บเซิร์ฟเวอร์ (Embedded web server) สามารถสร้างได้บนบอร์ดคอม86 โดยไม่จำเป็นต้องเชื่อมต่อวงจรอื่นๆ เพิ่มเติม ซึ่งบอร์ดคอม86 นี้สามารถสื่อสารกับภายนอกผ่านทางอีเทอร์เน็ตพอร์ตที่อยู่บนบอร์ดที่ความเร็ว 10 เมกกะบิตเปอร์เซ็ก ทำให้สามารถนำไปประยุกต์ใช้งานในกาออกแบบอุปกรณ์ควบคุมต่างๆ ผ่านเอ็มเบดเดดเว็บเซิร์ฟเวอร์ ที่ทำงานอยู่บนบอร์ดคอม86 ได้โดยง่าย

4. โรโบติก (Robotics)

บอร์ดคอม86 มีไมโครคอนโทรลเลอร์ที่มีประสิทธิภาพสูงในการทำงาน การพัฒนาซอฟต์แวร์สามารถพัฒนาได้ซับซ้อนมากยิ่งขึ้น การเชื่อมต่อขยายทางด้านฮาร์ดแวร์ต่างๆสามารถทำได้โดยง่าย เหมาะสำหรับการพัฒนาหุ่นยนต์ที่มีการทำงานที่ซับซ้อน

5. การควบคุมบ้านหรือสิ่งก่อสร้าง (Home and building control)

อุปกรณ์สำหรับควบคุมระบบต่างๆภายในบ้าน หรืออาคารสามารถนำบอร์ดคอม86 ไปประยุกต์ใช้งานได้เช่นกัน โดยใช้เป็นส่วนในการควบคุมการทำงานของอุปกรณ์ต่างๆ หรือคอยตรวจสอบการทำงานของเซนเซอร์ (Sensor) ต่างๆ ภายในบ้าน และแจ้งไปยังเจ้าของบ้านเมื่อเกิดเหตุร้ายขึ้น นอกจากนี้ตัวไมโครคอนโทรลเลอร์ยังรองรับการนำไปสร้างเป็นอุปกรณ์ต่างๆ ที่เกี่ยวข้องกับงานทางด้านการสื่อสารต่างๆเช่น ไอเอสดีเอ็นโมเด็มแอนด์เทอมินัลอแดบเตอร์ (ISDN Modem and Terminal Adaptor), โลเอนด์เราเตอร์ (Low-end Router), ไลน์การ์ดแอพพลิเคชั่น (Line card application), เอ็กซ์ดีเอสแอลแอพพลิเคชั่น (xDSL application), ดิจิตอลคอร์ดโฟน (Digital corded phone) เป็นต้น ซึ่งสามารถนำบอร์ดคอม86 ไปประยุกต์ใช้งานในการออกแบบพัฒนาได้เช่นกัน

# บทที่ 3
# ทฤษฎีและหลักการของโปรแกรมที่เกี่ยวข้อง

## 3.1 การเขียนโปรแกรมให้ทำงานเป็นมัลติทาสก์

เป็นโหมดการทำงานที่อนุญาตให้สามารถทำงานพร้อมๆกันได้ หรือ ประมวลผลแบบสอดแทรกกันของสองทาสก์ หรือมากกว่า คล้ายกับมัลติโปรแกรมมิ่งที่มีรูปแบบแตกต่างกัน

ในระบบปฏิบัติการดอสมีข้อจำกัดคือไม่อนุญาตโปรแกรมหลายๆ โปรแกรมทำงานพร้อมๆกันต้องทำงานให้เสร็จทีละงานเป็นลำดับไป ดังนั้น งานในการจัดการให้แต่ละโปรแกรมหรือหลายๆงานทำไปพร้อมกันจึงเป็นขององผู้สร้างโปรแกรม

ในปัจจุบันมีไลบรารีที่มาช่วยจัดการให้โปรแกรมหนึ่งๆ สามารถรันหลายๆฟังก์ชันสลับกันไปมาได้หลายตัวด้วยกัน อาทิเช่น ไมโครซีโอเอส (อ่านว่า MicroC/OS), Multi-C library เป็นต้น ซึ่งในโครงงานนี้ผู้พัฒนาเลือกใช้เนื่องจากไม่ขึ้นกับหน่วยประมวลผล และสนับสนุนการทำงานแบบเรียลไทม์ (real time)

## 3.2 ไมโครคอนโทรลเลอร์ โอเปอร์เรทติ้ง ซิสเต็มเวอร์ชัน 2

ไมโครคอนโทรลเลอร์ โอเปอร์เรทติ้ง ซิสเต็มเวอร์ชัน 2 หรือไมโครซีโอเอส2 (µC/OS-II) ถูกสร้างขึ้นบนพื้นฐานของเรียลไทม์เคอร์เนล (Real-time kernel) ซึ่งนิยมนำไปใช้ในอุปกรณ์หลายๆอย่างด้วยกัน เช่น กล้องถ่ายรูป เน็ตเวอร์คอะเดปเตอร์ อุตสาหกรรม เป็นต้น

### 3.2.1 จุดเด่นของไมโครคอนโทรลเลอร์ โอเปอร์เรทติ้ง ซิสเต็มเวอร์ชัน 2

1. ซอร์ดโค้ด (Source Code) มีการจัดระเบียบไว้เป็นอย่างดีและเข้าใจง่าย

2. พอร์ตเทเบิล (Portable) โดยส่วนใหญ่แล้วไมโครซีโอเอส2 จะถูกเขียนโดย ANSI C ที่มีความสามารถในการพอร์ตเทเบิลสูง ด้วยการเขียนด้วยเอเอสเอ็ม (ASM) ที่ทำให้ไมโครซีโอเอส2 มีขนาดเล็กและสามารถพอร์ตไปยังชิพประมวลผลตัวอื่น ๆ ได้เช่นเดียวกันกับ ไมโครซีโอเอส ไมโครซีโอเอส2 สามารถถูกพอร์ตลงไปในชิพประมวลผลตัวที่ใหญ่กว่าได้ ตราบใดที่ชิพประมวลผลนั้น ๆ มีแสต็กพอยเตอร์ (stack pointer) และซีพียู (CPU register) ที่สามารถถูกพุช (push) และป็อป (pop) ลงไปในแสต็กได้

พอร์ตทั้งหมดที่มีอยู่ในไมโครซีโอเอสสามารถถูกแปลงมาลงในไมโครซีโอเอส2 ภายในเวลาประมาณ 1 ชม. เพราะไมโครซีโอเอส2 สามารถเข้ากันได้กับไมโครซีโอเอส ทำให้โปรแกรมประยุกต์ของไมโครซีโอเอสสามารถรันบนไมโครซีโอเอส2 ได้ด้วยการเปลี่ยนแปลงที่เล็กน้อย หรือว่าไม่เปลี่ยนแปลงเลย สามารถตรวจสอบพอร์ตที่มีอยู่ในปัจจุบันของไมโครซีโอเอส2 ได้จาก เวปไซต์ www.uCOS-II.com

3. รอมเมเบิล (ROMable) ไมโครซีโอเอส2ได้ถูกออกแบบสำหรับโปรแกรมประยุกต์ฝังตัวหมายความว่าถ้าเรามี proper tool chain (เช่น C computer, assembler, และ linker/locator) คุณสามารถฝัง µC/OS-II ลงไปในส่วนของชิ้นงานของคุณได้

4. แสกลเอเบิล (Scalable) ผู้สร้างได้ออกแบบไมโครซีโอเอส2ในแบบที่เราสามารถใช้เฉพาะบริการที่เราต้องการในงานได้ หมายถึงงานของเราสามารถใช้งานบริการบนไมโครซีโอเอส2บางส่วน จนถึงใช้งานบริการ

ทุกส่วน และยอมให้คุณลดขนาดของหน่วยความจำ (ทั้ง ROM และ RAM) ที่ถูกใช้งานโดยไมโครซีโอเอส2 บนแต่ละงานได้

5. พรีเอ็มทีพ (Preemptive) ไมโครซีโอเอส2เป็น “fully preemptive real-time kernel” ซึ่งหมายความว่า ไมโครซีโอเอส2จะทำงานงานที่มีความสำคัญสูงท่าสุดที่พร้อมทำงานก่อนเสมอ ซึ่งโดยเคอร์เนลตามท้องตลาดส่วนใหญ่แล้วจะเป็นแบบพรีเอ็มทีพและไมโครซีโอเอส2เทียบเคียงได้กับเอคร์เนลเหล่านั้น

6. มัลติทาสก์กิ้ง (Multitasking)ไมโครซีโอเอส2สามารถจัดการทาสก์ได้ถึง 64 ทาสก์พร้อมกัน อย่างไรก็ตามซอฟแวร์รุ่นปัจจุบันได้จองทาสก์ ไว้ 8 ทาสก์เพื่อให้สำหรับระบบใช้งาน ทำให้เหลือเพียง 56 ทาสก์เท่านั้น แต่ละทาสก์จะมีเลขกำกับที่ไม่ซ้ำกัน ซึ่งหมายความว่าไมโครซีโอเอส2ไม่สามารถที่จะทำ round-robin scheduling ได้เหตุนี้ทำให้มี 64 ระดับความสำคัญ (priority level)

7. ดีเทอร์มินิสติก (Deterministic) เวลาในการประมวลผลของไมโครซีโอเอส2สามารถวัดได้ นั่นหมายถึงเราสามารถวัด รู้และคำนวณเวลาที่ไมโครซีโอเอส2ใช้ในการคำนวณฟังชันก์และบริการ แต่อย่างไรก็ตามเวลาในการประมวลผลของไมโครซีโอเอส2เซอร์วิสทั้งหมดไม่ขึ้นอยู่กับจำนวนของทาสก์ในงานของเรา

8. ทาสก์แสต็ก (Task Stacks) แต่ละทาสก์ต้องการแสต็กของตัวเอง อย่างไรก็ตามไมโครซีโอเอส2ยอมให้แต่ละทาสก์มีขนาดของแสต็กที่แตกต่างกันได้ ซึ่งสามารถช่วยให้เราลดขนาดการใช้งานบนแรมบนงานของเราได้ด้วย stack-checking ของไมโครซีโอเอส2 เราสามารถประมาณค่าของแสต็กที่เราต้องการใช้บนแต่ละทาสก์ของเราได้อย่างเท่าที่เราใช้งานจริง ๆ

9. บริการ (Services)ไมโครซีโอเอส2 ได้เตรียมบริการต่าง ๆ เช่น mailboxes queues semaphores partition เป็นต้น

10. การจัดการอินเทอร์รัพท์ (Interrupt Management) อินเทอร์รัพท์สามารถหยุดการทำงานของทาสก์ไว้ชั่วคราวได้ โดยที่ทาสก์ที่มีความสำคัญที่สูงกว่าจะสามารถตื่นขึ้นมาทำงานระหว่างอินเทอร์รัพท์ของทาสก์ที่มีความสำคัญต่ำกว่าได้ โดยที่อินเทอร์รัพท์สามารถถูกใช้ได้ถึง 255 ระดับซ้อนกัน

11. มีความสามารถมากและเชื่อถือได้ (Robust and Reliable) ไมโครซีโอเอส2มีพื้นฐานอยู่บน ไมโครซีโอเอส ซึ่งถูกใช้ในหลายร้อยงานทั่วโลกมาตั้งแต่ปี 1992 ซึ่งไมโครซีโอเอส2ใช้คอร์ (core) เดิมและฟังก์ชันที่ยังคงอยู่เหมือนเดิมเป็นส่วนใหญ่

### 3.2.2 แนวคิดแบบเรียลไทม์ซิสเต็ม

เรียลไทม์ซิสเต็มได้ถูกกำหนดลักษณะโดยผลที่ตามมาซึ่งถูกบังคับอันเนื่องมาจากทางตรรกะเช่นเดียวกันกับคุณสมบัติความถูกต้องของเวลาของระบบที่ไม่ได้เป็นดังนั้น แบ่งออกเป็น 2 ชนิดคือ ซอฟท์ (SOFT) และฮาร์ด (HARD) สำหรับ SOFT real-time system นั้นทาสก์จะถูกกระทำโดยระบบเร็วที่สุดตราบเท่าที่จะทำได้ แต่ทาสก์อาจจะไม่เสร็จได้ทันตามเวลาที่ได้กำหนดไว้ สำหรับแบบ Hard real-time system จะถูกกำหนดให้งานออกมาถูกต้องและเสร็จตามเวลาที่กำหนด เรียลไทม์ซิสเต็มโดยส่วนใหญ่แล้ว จะประกอบไปด้วยทั้งแบบ SOFT และ HARD real-time application นั้นมีครอบคลุมกว้างมาก แต่เรียลไทม์ซิสเต็มส่วนใหญ่นั้นจะเป็นแบบฝังตัว หมายความว่าคอมพิวเตอร์ที่ถูกสร้างในระบบจะไม่ถูกผู้ใช้เห็น โดยที่เรียลไทม์ซิสเต็มนั้นออกแบบได้ยากกว่าระบบที่ไม่ รียลไทม์ซิสเต็มมาก

### 3.2.3 มิวเซอร์ลเอ็กซ์คูลชัน (Mutual Exclusion)

วิธีการที่ง่ายที่สุดสำหรับทาสก์ที่จะติดต่อสื่อสารกันผ่านข้อมูลที่ใช้ร่วมกัน เมื่อมีการแลกเปลี่ยนข้อมูลระหว่างกันบนข้อมูลที่ใช้ร่วมนั้น เราต้องมั่นใจว่าแต่ละทาสก์จะมีการเข้าถึงข้อมูลร่วมนั้นโดยหลีกเลี่ยงการแย่งชิงข้อมูล โดยวิธีการโดยทั่วไปที่จะเข้าถึงข้อมูลร่วมมีดังนี้

- ไม่ให้ใช้อินเทอร์รัพท์ (Disable interrupts)
- ทำเทสท์และเซ็ทโอเปอร์เรชัน (Performing test-and-set operations)
- ไม่ใช้มีการใช้การจัดลำดับ (Disabling scheduling)
- ใช้เซมาฟอร์ (Using semaphores)

### 3.2.4 เซมาฟอร์ (Semaphores)

เป็นโปรโตคอลสำหรับมัลติทาสก์กิ้งเคอร์เนล (Multitasking kernels) โดยส่วนใหญ่ มีจุดประสงค์เพื่อ

- ควบคุมการเข้าถึงทรัพยากรร่วม
- ส่งสัญญาณของเหตุการณ์ที่เกิดขึ้น
- ยอมให้ 2 ทาสก์ทำการซิ้งค์ (Sync) งานกันได้

เซมาฟอร์เป็นกุญแจที่จะทำให้โค้ดของเราเรียงลำดับการทำงาน ถ้าเซมาฟอร์กำลังถูกใช้งานอยู่ทาสก์ที่เรียกเข้ามาจะถูกพักไว้ชั่วคราวจนกระทั่งเซมาฟอร์ถูกปล่อยจากเจ้าของปัจจุบัน โดยแบ่งเซมาฟอร์ได้เป็น 2 ชนิด คือ Binary semaphore และ counting semaphore ดังเช่นที่ชื่อได้แสดงไว้ binary semaphore สามารถมีค่าได้เพียงแค่ 0 หรือ 1 ส่วน counting semaphore สามารถมีค่าได้ตั้งแต่ 0 ถึง 255, 65535, 4294967295 แล้วแต่โครงสร้างของแต่ละเซมาฟอร์ที่เขียนโดยใช้ 8, 16, 32 บิต แต่ขนาดจริงๆ จะขึ้นอยู่กับที่เคอร์เนลที่ใช้ เมื่อมีการนำเซมาฟอร์มาใช้งาน ตัวระบบเองต้องมีการเก็บเส้นทางของทาสก์ที่ทำการรอเพื่อให้เซมาฟอร์ดำเนินการได้

โดยปกติแล้ว มีเพียงแค่ 3 โอเปร์เรชัน ที่ถูกเตรียมไว้ใช้ในเซมาฟอร์ โดยประกอบด้วย INITIALIZE (หรือเรียกว่า CREATE), WAIT (หรือเรียกว่า PEND), SIGNAL (หรือเรียกว่า POST) สำหรับค่าเริ่มต้นของเซมาฟอร์ต้องถูกเตรียมในช่วงที่เซมาฟอร์ถูก initialized และทาสก์ที่รออยู่ของเซมาฟอร์จะต้องถูกทำให้ว่าง

ทาสก์ที่ต้องการเซมาฟอร์จะเรียก WAIT แล้วถ้าเซมาฟอร์นั้นสามารถใช้งานได้ (ค่ามีค่าเกิน 0) ค่าของเซมาฟอร์จะถูกลดค่าลงและทาสก์จะสามารถทำงานได้ต่อไป แต่ถ้าค่าของเซมาฟอร์มีค่าเป็น 0 แล้วทาสก์มีการเรียก WAIT จะถูกทำการถูกนำไปไว้ใน waiting list โดยเคอร์เนล ส่วนใหญ่จะยอมให้กำหนด timeout เพื่อกำหนดเวลาที่รอเซมาฟอร์ โดยจะมีการเรียกทาสก์ขึ้นมารันและแสดงข้อผิดพลาดกลับไปยังผู้ที่เรียกทาสก์นั้น ทาสก์ที่ปล่อย เซมาฟอร์ออกมาจะไปเรียก SIGNAL ถ้าไม่มีทาสก์อื่นรออยู่ใน waiting list ค่าของ เซมาฟอร์ก็จะเพิ่มขึ้น แต่ถ้ามีทาสก์เหลืออยู่ในลิสท์จะทำการเรียกทาสก์ในลิสท์ขึ้นมารันโดยไม่มีกี่เพิ่มค่าเซมาฟอร์ สำหรับเงื่อนไขในการหยิบทาสก์ขึ้นมาทำงานนั้น ขึ้นอยู่กับเคอร์เนลดังเช่น

- ทาสก์ที่มีระดับความสำคัญสูงสุดที่รออยู่
- ทาสก์แรกที่เข้ามารอในลิสท์ (First In First Out หรือ FIFO)

### 3.2.5 การสื่อสารระหว่างทาสก์ (Intertask Communication)

บางครั้งก็เป็นความจำเป็นของทาสก์หรือไอเอสอาร์ (ISR) ที่จะติดต่อข้อมูลกับทาสก์อื่น ๆ ข้อมูลที่แลกเปลี่ยนเรียกการสื่อสารระหว่างทาสก์ โดยข้อมูลที่ถูกใช้ในการสื่อสารระหว่างทาสก์ทำได้ 2 ทาง คือ ผ่านตัวแปรกลาง หรือ ส่งข้อความ

เมื่อใช้การผ่านตัวแปรกลาง แต่ละทาสก์หรือไอเอสอาร์ต้องแน่ใจว่ามีการเข้าถึงข้อมูลเพียงทีละหนึ่งทาสก์เท่านั้น ถ้าไอเอสอาร์ได้ถูกเรียกแล้วมีทางเดียวที่จะเข้าถึงข้อมูลได้อย่างปลอดภัยคือ การไม่ให้ใช้อินเทอร์รัพท์ (disabled interrupt) แต่ถ้ามี 2 ทาสก์ที่ขอใช้งานตัวแปรร่วมตัวเดียวกันสามารถทำได้โดยไม่ให้ใช้อินเทอร์รัพท์หรือเรียกใช้เซมาฟอร์

ทาสก์สามารถติดต่อกับไอเอสอาร์ได้เพียงเดียวเท่านั้นคือผ่านตัวแปรกลาง ซึ่งโดยที่จริงแล้วทาสก์ไม่จำเป็นต้องกลัวว่าไอเอสอาร์จะไปเปลี่ยนค่าในตัวแปรกลาง จึงไม่มีความจำเป็นที่ไอเอสอาร์จะต้องส่งสัญญาณไปยังทาสก์ โดยใช้เซทาฟอร์หรือโพล์ (Poll) เพื่อให้เหมาะกับสถานการณ์เช่นนี้ คุณควรเลือกใช้แมสเซสเมล์บ็อกซ์ (message mailbox)

### 3.2.6 แมสเซสเมล์บ็อกซ์

ข้อความสามารถถูกส่งไปยังทาสก์ผ่าน Kernel services คือแมสเซสเมล์บ็อกซ์หรือเรียกว่า Message exchange โดยเก็บผ่านตัวแปรพอยเตอร์ โดยทาสก์หรือไอเอสอาร์สามารถฝากแมสเซสลงในเมล์บอกซ์และ 1 หรือ หลายทาสก์สามารถรับแมสเซสผ่านบริการที่เคอร์เนลเตรียมไว้ โดยการส่งและรับข้อความจะเป็นการส่งข้อมูลที่พอยเตอร์นั้น ๆ ชี้อยู่

Waiting list ที่สัมพันธ์แต่ละเมล์บอกซ์ ในกรณีที่มี 1 หรือ หลาย ๆทาสก์ต้องการรับข้อความผ่าน เมล์บอกซ์ โดยทาสก์ที่ต้องการข้อความจากเมล์บอกซ์ที่ว่าง จะถูกทำให้พักการทำงานและถูกทำให้รออยู่ใน Waiting list จนกระทั่งมีข้อความเข้ามาในเมล์บอกซ์ โดยเคอร์เนลอนุญาตให้รอได้จนถึง timeout ถ้าข้อความยังมาไม่ถึงก่อนเวลาที่กำหนดไว้ทาสก์ที่รอรับข้อความจะถูกทำให้ ready และทำงาน โดยส่ง error code กลับไป เมื่อข้อความได้ถูกนำออกจากเมล์บอกซ์ทาสก์ที่มีระดับความสำคัญสูงที่สุดจะทำการรับข้อความ (แบบ priority based) หรือทาสก์แรกที่เข้ามารอเป็นผู้รับข้อความ (แบบ FIFO based)

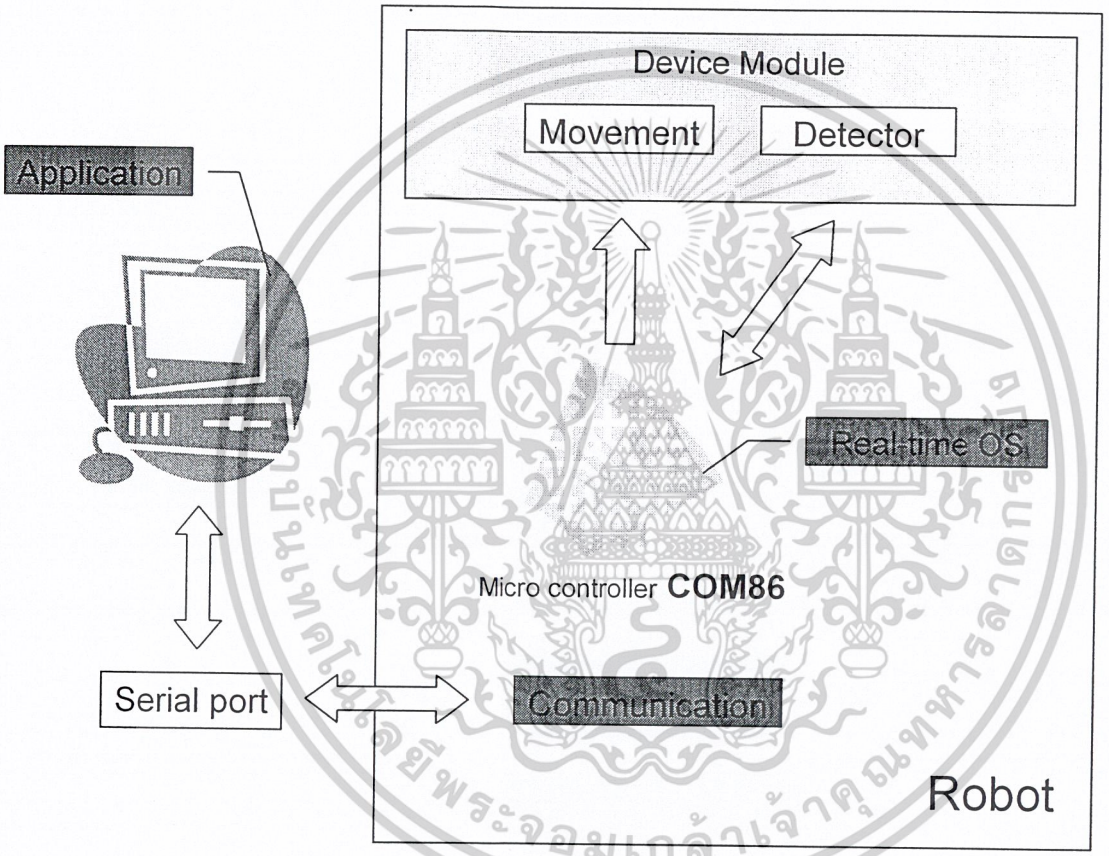
เคอร์เนลได้เตรียมบริการสำหรับเมล์บอกซ์ดังนี้

- กำหนดค่าเริ่มต้นของเนื้อหาในเมล์บอกซ์ในตอนแรกเริ่มเมล์บอกซ์สามารถกำหนดให้มี หรือ ไม่มีแมสเซสได้
- ฝากแมสเซสลงไปในเมล์บอกซ์ (POST)
- รอรับแมสเซสที่ได้ทำการฝากลงมาในเมล์บอกซ์ (PEND)
- รับแมสเซสจากเมล์บอกซ์ ถ้ามีข้อความอยู่ แต่จะไม่พักการทำงานของผู้ที่เรียกถ้าเมล์บอกซ์ ว่างเปล่า (ACCEPT) ถ้าเมล์บอกซ์มีข้อความอยู่ ข้อความจะถูกนำออกจากเมล์บอกซ์ และส่งโค้ดกลับจะถูกใช้นำไปใช้ในการเตือนผู้ที่เรียกเกี่ยวกับผลลัพธ์ของการเรียก

# บทที่ 4
# การออกแบบและพัฒนา

## 4.1 ขอบเขตของโครงงาน

โครงงานมีจุดประสงค์หลักคือ นำคอม86 มาประยุกต์ใช้ในงานหุ่นยนต์ และออกแบบโปรแกรมที่จะช่วยเหลือผู้ใช้งานในการสร้างหุ่นยนต์รวมถึงออกแบบชุดคำสั่งที่สามารถเรียกใช้ควบคุมอุปกรณ์ได้โดยง่าย รวมถึงโปรแกรมที่จะช่วยแสดงผลการทำงานและควบคุมหุ่นยนต์ผ่านทางคอมพิวเตอร์




*รูปที่ 4-1 คอนเท็กซ์ไดอะแกรม ของโครงงาน*

จากภาพประกอบด้วยส่วนประกอบ 2 ส่วนด้วยกัน คือ

1. ส่วนของหุ่นยนต์ที่ใช้คอม86เป็นตัวประมวลผลหลักจากบอร์ดคอม86จะต่อเข้ากับ บอร์ดที่ทำการ ขยายพอร์ตแอดเดสและเชื่อมต่อกับ วงจรของอุปกรณ์ที่ออกแบบเป็นโมดูลมาตรฐานสำหรับเชื่อมต่อกับ ตัว บอร์ด ขยายพอร์ตแอดเดสและใช้ ระบบปฏิบัติการแบบเรียลไทม์ในการจัดการทำงานของคอม86โดยมีการสื่อสารระหว่างหุ่นยนต์กับคอมพิวเตอร์โดยการส่งข้อมูลผ่านพอร์ตอนุกรม

2. ส่วนของคอมพิวเตอร์ที่จะมีโปรแกรมหลักสำหรับควบคุมการทำงาน ส่วนคอมพิวเตอร์นั้นใช้สำหรับการเขียนโปรแกรมควบคุมหุ่นยนต์ด้วยภาษาซี ภายใต้การควบคุมโดยโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์ ที่คณะผู้จัดทำสร้างขึ้นเพื่ออำนวยความสะดวกและจัดการการทำงานของหุ่นยนต์รวมถึงส่วนของชุดหรับควบคุมและ

สั่งงานอุปกรณ์ ที่เป็นชุดคำสั่งสำหรับเรียกใช้งานเพื่อสะดวกและง่ายต่อการควบคุมอุปกรณ์บนตัวหุ่นยนต์ รวมถึงโปรแกรมที่ติดต่อส่งผ่านข้อมูลระหว่างหุ่นยนต์กับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม

จากส่วนประกอบทั้งสองด้านนั้นจำแนกตัวโครงงานออกเป็น 4 ส่วนหลัก ดังนี้

**4.1.1 ส่วนของการเชื่อมต่ออุปกรณ์กับคอม86**

โดยในส่วนนี้ทางคณะผู้จัดทำจะทำการออกแบบวงจรที่จะใช้งานตัวอุปกรณ์เช่น วงจรไดร์ฟมอเตอร์ วงจรเซนเซอร์ เป็นโมดูลที่มีรูปแบบมาตรฐาน ที่ทำให้ผู้ใช้งานสามารถเรียกใช้งาน ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ ได้อย่างถูกต้อง

**4.1.2 ส่วนโปรแกรมควบคุมหลัก**

โดยในส่วนนี้เป็นส่วนที่จะช่วยจัดการและดูแลการทำงานของหุ่นยนต์ประกอบด้วยส่วนต่าง ๆ ดังนี้

- ส่วนการจัดการอุปกรณ์ต่าง ๆ ที่นำมาเชื่อมต่อกับหุ่นยนต์
- ส่วนช่วยเหลือผู้ใช้งานในการเขียนโปรแกรมและโหลดโปรแกรมไปยังคอม86
- ส่วนควบคุมและจัดการเมมโมรี่การใช้งานโปรแกรมต่าง ๆ บนคอม86
- ส่วนประมวลผลและส่งข้อมูลจากหุ่นยนต์ไปแสดงผลการทำงานที่คอมพิวเตอร์

**4.1.3. ส่วนของคำสั่งสำหรับผู้ใช้ทำให้สามารถควบคุมอุปกรณ์ต่าง ๆ**

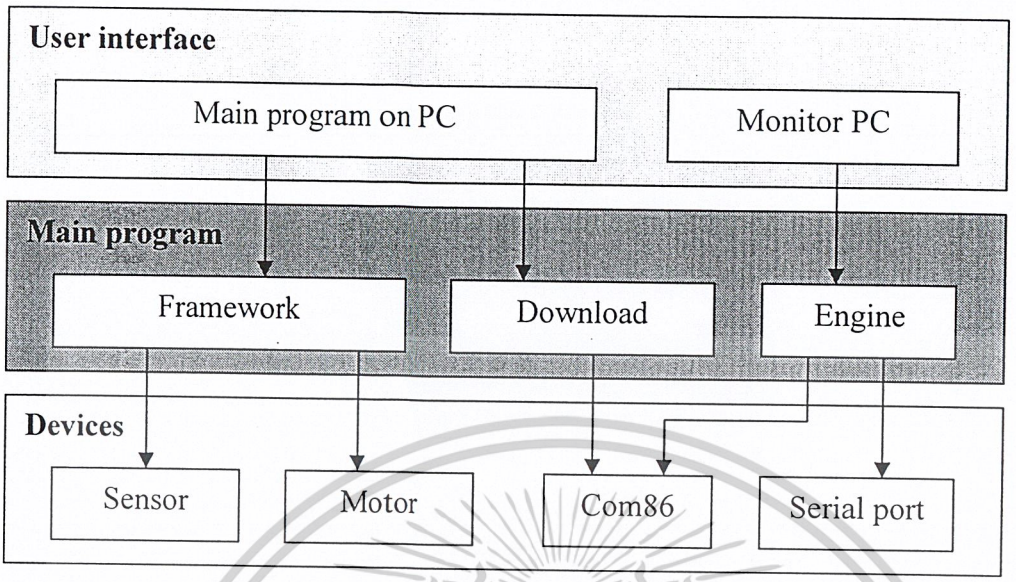
โดยในส่วนของ ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ นั้นทางคณะผู้จัดทำจะทำการเขียนขึ้น เป็นเหมือนชุดคำสั่งสำหรับตัวอุปกรณ์นั้นๆ โดยเมื่อผู้ใช้งานทำการเพิ่มอุปกรณ์นั้น ๆ เข้ามาทางโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์ จะทำการโหลด .h ของอุปกรณ์ตัวนั้นมาลงในโปรแกรมหลัก (ตัวอย่างเช่น Include motor.h) และจะมีคำสั่งต่าง ๆ ที่ให้ผู้ใช้งานสามารถเรียกใช้ควบคุมอุปกรณ์ได้เลยโดยไม่จำเป็นต้องเขียนโปรแกรมติดต่อกับอุปกรณ์โดยตรง

**4.1.4 ส่วนของโปรแกรมบนคอมพิวเตอร์**

(ในขั้นต้นนี้จะทำการพัฒนาโปรแกรมที่ติดต่อกับหุ่นยนต์บนคอมพิวเตอร์ ก่อนแล้วอาจจะทำการเพิ่มเติมไปใช้งานบน Pocket PC หรือ PDA ในภายหลัง) โดยมีส่วนประกอบของโปรแกรมหลัก ๆ ดังนี้

- ส่วนหลักของโปรแกรม เป็นส่วน ติดต่อกับผู้ใช้งานหลักที่จะทำให้ผู้ใช้งาน สามารถเพิ่ม/ลดอุปกรณ์, เขียนโปรแกรม, ส่งข้อมูลไปยังคอม86 รวมถึงส่วนแสดงผลการทำงาน โดยคำสั่งต่าง ๆ ผู้ใช้งานสามารถใช้งานได้ตลอดเวลาเพราะบนหุ่นยนต์นั้นสามารถแก้ไขข้อมูลหรือเปลี่ยนแปลงโปรแกรมในขณะใดก็ได้
- ส่วนแสดงผลการทำงาน เป็นส่วนที่จะแสดงผลการทำงานของหุ่นยนต์ในขณะนั้นโดยจะมีโปรแกรมบนคอม86 จะทำการส่งข้อมูลต่าง ๆ มายังคอมพิวเตอร์ โดยบนคอมพิวเตอร์ จะมีโปรแกรมแสดงผลข้อมูลนั้น ๆ ให้กับผู้ใช้งานทำให้เราสามารถทราบถึงการทำงานและผลของการทำงานของหุ่นยนต์ตลอดเวลา

**4.2 โครงสร้างทางสถาปัตยกรรมของโครงงาน**



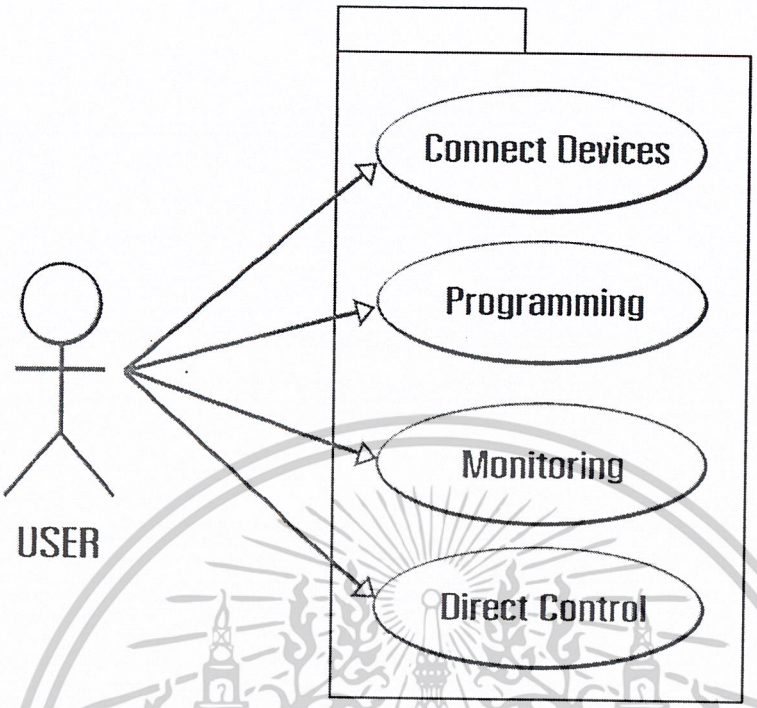

*รูปที่ 4-2 โครงสร้างทางสถาปัตยกรรมของโครงงาน*

แบ่งโครงสร้างของโปรแกรมออกเป็น 3 ชั้น

- ส่วนแอปพลิเคชั่นบนคอมพิวเตอร์ที่ติดต่อกับผู้ใช้งาน ( User Interface )
- ส่วนโปรแกรมควบคุมการทำงานหลัก ( Main Program )
- ส่วนโปรแกรมที่ทำงานบนคอม86 ทำหน้าที่ควบคุมอุปกรณ์ ( Devices layer)

โครงสร้างของโปรแกรมจำแนกลำดับชั้นของโครงสร้างโปรแกรมออกเป็น 3 ส่วนหลัก ดังภาพโดยส่วนที่ติดต่อกับผู้ใช้งานนั้นคือส่วนที่แอปพลิเคชั่นที่ทำงานอยู่บนคอมพิวเตอร์ โดยหน้าที่หลักของส่วนนี้คือช่วยในการสื่อสารระหว่างผู้ใช้งานกับหุ่นยนต์ อีกส่วนหลักก็คือ ส่วนโปรแกรมที่ทำงานติดต่อกับอุปกรณ์ซึ่งเป็นโปรแกรมที่ทำงานในระดับแอดเดสที่เชื่อมอยู่กับอุปกรณ์โดยส่วนนี้จะเป็นเหมือนส่วนชุดคำสั่งในระดับล่างที่ผู้ใช้งานสามารถเรียกใช้ได้โดยไม่ต้องทราบถึงการเชื่อมต่อในระดับฮาร์ดแวร์ ส่วนตรงกลางตามโครงสร้างที่วางไว้คือ เป็นส่วนของโปรแกรมควบคุมการทำงาน เป็นส่วนที่คอยจัดการและทำการช่วยเหลือผู้ใช้งาน ในส่วนของทำการเชื่อมต่อคอมพิวเตอร์ กับหุ่นยนต์นั่นเองโดยการทำงานส่วนใหญ่นั้นเป็นโปรแกรมที่ทำงานอยู่บนคอมพิวเตอร์

## 4.3 ยูสเคสไดอะแกรม (Use Case Diagram) ของระบบ




*รูปที่ 4-3 ยูสเคสไดอะแกรมของผู้ใช้งาน*

### 4.3.1 ผู้ใช้งาน (User)

ความต้องการจากผู้ใช้งานนับเป็นจุดสำคัญของโครงงานเนื่องจากโปรแกรมมีจุดประสงค์หลัก มาจากการออกแบบโปรแกรมที่จะช่วยเหลือการสร้างหุ่นยนต์ ให้สามารถทำได้โดยง่ายสะดวก เข้าใจได้ง่ายและแก้ปัญหาการขาดทักษะรวมถึงขาดประสบการณ์ในการออกแบบ และทำงานกับอุปกรณ์ต่าง ๆ ของหุ่นยนต์

#### 4.3.1.1 ติดตั้งอุปกรณ์กับบอร์ด คอม86

ผู้ใช้งานสามารถต่อโมดูลของอุปกรณ์ที่สนับสนุนการทำงานของฟังก์ชัน ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ กับชุดพัฒนา คอม86 โดย ผู้ใช้งานจะต่อโมดูลของอุปกรณ์ กับบอร์ดขยายพรอร์ตแอดเดส ที่ขยายพอร์ตจากชุดพัฒนา คอม86 ที่คณะผู้จัดทำกำหนดไว้ให้ และ ผู้ใช้งานจะสามารถปรับแต่ง บนแอพพลิเคชั่น ด้วยว่าได้ต่ออุปกรณ์ใดไว้ที่ พอร์ต ใดบ้าง

#### 4.3.1.2 เขียนโปรแกรมเพื่อใช้ควบคุมอุปกรณ์

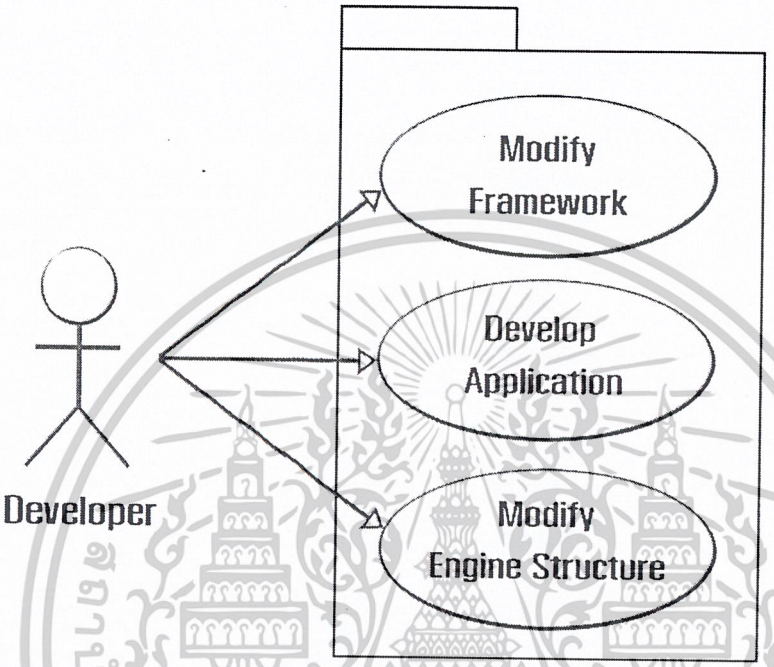
ผู้ใช้งาน สามารถเขียนโปรแกรม และดาวโหลดโปรแกรม ลงโดยใช้ฟังก์ชั่นจากชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ ผ่าน แอพพลิเคชั่น ที่คณะผู้จัดทำสร้างขึ้นมา เพื่อให้หุ่นยนต์สามารถทำงานตามโปรแกรม ใน การทำงานแบบอัตโนมัติได้

#### 4.3.1.3 ดูสถานะของอุปกรณ์และการทำงานของหุ่นยนต์

ผู้ใช้งาน สามารถดูสถานะของอุปกรณ์ต่างๆ ที่มาเชื่อมต่อกับชุดพัฒนา คอม86 ได้ตลอดเวลาผ่านทางพอร์ตอนุกรม โดยจะทราบทันทีว่าว่าอุปกรณ์แต่ละตัวมีการรับค่าหรือส่งค่าอะไรอยู่ เพื่อที่ ผู้ใช้งาน สามารถทราบถึงการทำงานของอุปกรณ์ และทำการตรวจสอบและแก้ไขโปรแกรม ให้เป็นไปอย่างถูกต้องได้

**4.3.1.4 ควบคุมการทำงานของหุ่นยนต์โดยตรง**

ผู้ใช้งาน สามารถควบคุมอุปกรณ์ที่เชื่อมต่อกับชุดพัฒนา คอม86 ได้โดยตรงในลักษณะของการทำงานโดยตรง และผู้ใช้งานจะควบคุมอุปกรณ์ต่างๆผ่านทาง คีย์บอร์ด โดยตรง หรือ ผู้ใช้งาน ควบคุมอุปกรณ์โดยใช้ฟังก์ชั่น ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ ที่มีอยู่มาสั่งงานผ่านทางคอมพิวเตอร์โดยตรงได้




*รูปที่ 4-4 ยูสเคสไดอะแกรมของผู้พัฒนา*

**4.3.2 ผู้นำไปพัฒนา (Developer)**

**4.3.2.1 แก้ไขและพัฒนา ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์**

ผู้พัฒนา สามารถแก้ไขและเพิ่มเติมฟังก์ชั่น ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ ที่คณะผู้จัดทำสร้างขึ้นเพื่อสามารถนำฟังก์ชั่น ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ ไปใช้ตามความต้องการที่หลากหลายขึ้น และเพื่อให้สามารถรองรับโมดูลของอุปกรณ์ชนิดใหม่ที่เพิ่มขึ้นมาได้ในอนาคต

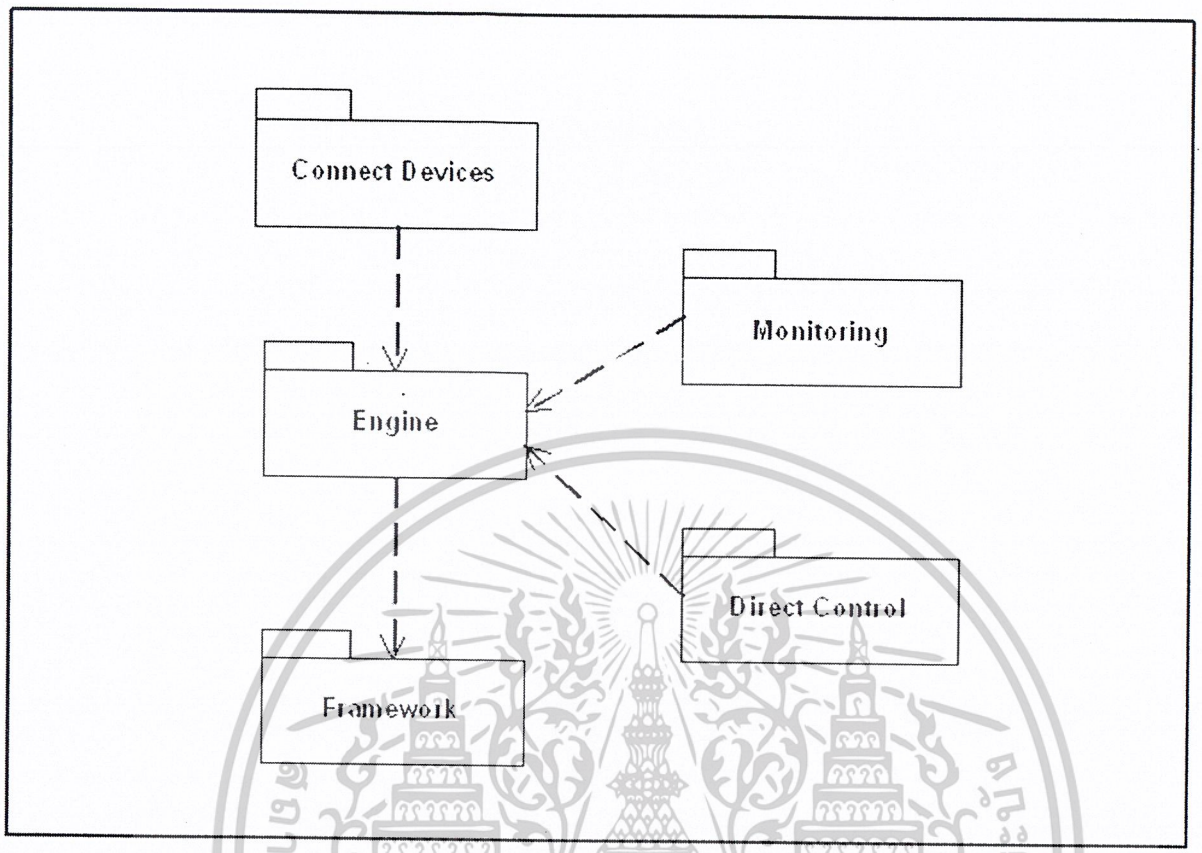
**4.3.2.2 พัฒนาและเปลี่ยนแปลง GUI**

ผู้พัฒนา สามารถแก้ไขและพัฒนา แอพพลิเคชั่น ที่ใช้งานฟังก์ชั่น ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ที่คณะผู้จัดทำสร้างขึ้น ซึ่งอาจแก้ไขรูปแบบของ GUI หรือการทำเป็น แอพพลิเคชั่น ใน pocket PC เพื่อควบคุมอุปกรณ์ที่เชื่อมต่อกับชุดพัฒนา คอม86 ได้

**4.3.2.4 แก้ไขและปรับเปลี่ยนค่าของโปรแกรมควบคุมหลัก**

ผู้พัฒนา สามารถแก้ไขโครงสร้างทาง Hardware บางอย่างของชุดพัฒนา คอม86ได้ เช่น เมมโมรี่ที่จองไว้สำหรับ ผู้ใช้งาน โปรแกรม การแก้ไขตำแหน่งการเก็บค่าในรีจีสเตอร์ต่างๆ หรือ การปรับปรุงแก้ไขแอดเดส เพื่อรองรับโมดูลของอุปกรณ์ชนิดใหม่ที่เพิ่มขึ้นได้

## 4.4 คอมโพแนนท์ไดอะแกรม (Component Diagram)




*รูปที่ 4-5 แผนภาพคอมโพแนนท์ของโครงงาน*

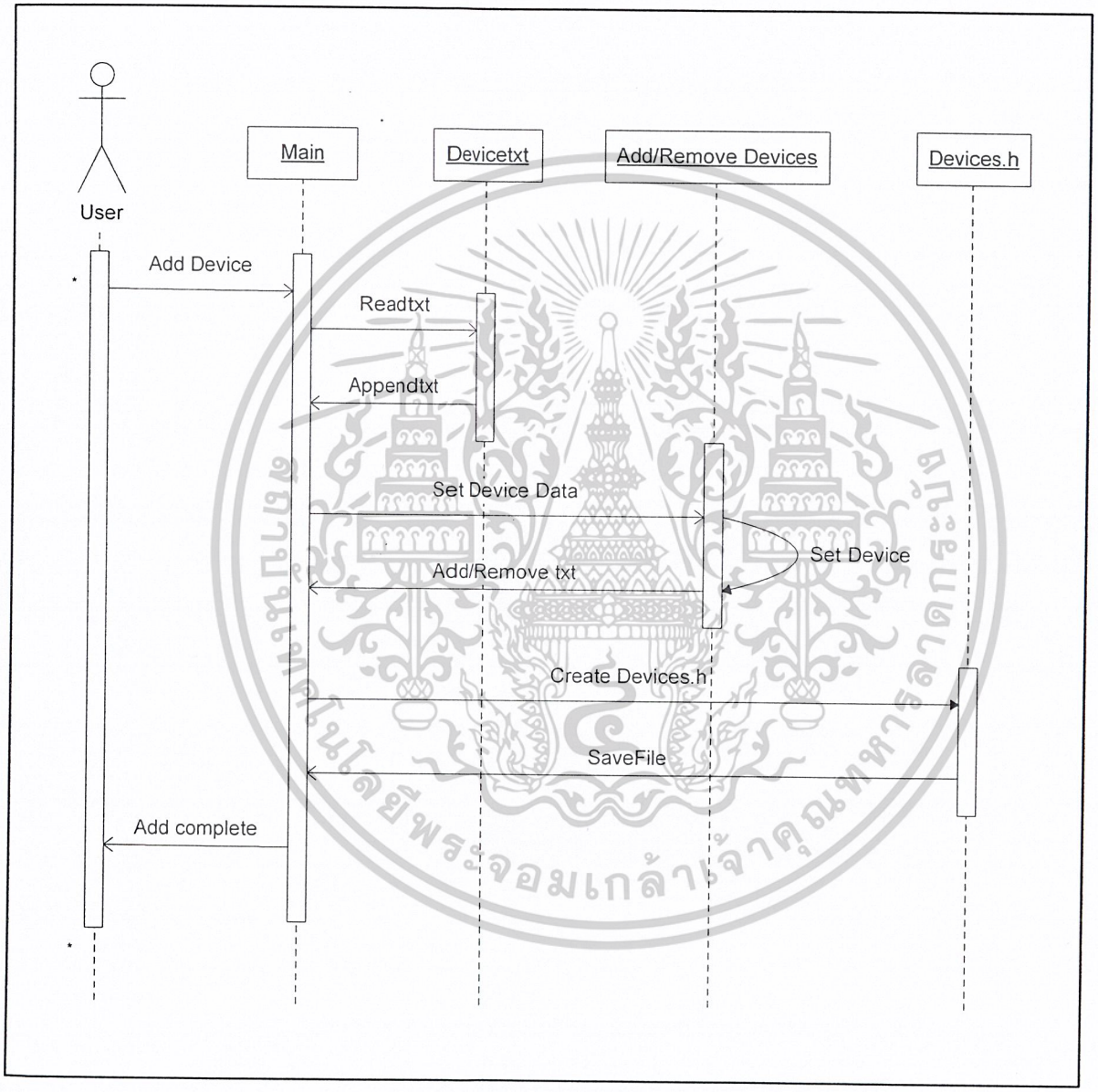
จากแผนภาพคอมโพแนนท์ ที่กำหนดไว้จะทำการแบ่งหมวดหมู่ของโปรแกรมออกเป็นส่วน ๆ โดยแต่ละส่วนจะมีการเรียกใช้อีกส่วนต่างกันออกไปตามลำดับ จากคอมโพแนนท์ไดอะแกรมที่ออกแบบขึ้นนั้น โปรแกรมหลักจะถูกแบ่งออกเป็น 5 ส่วนหลัก ดังนี้

- ส่วนของชุดคำสั่งที่เตรียมไว้สำหรับเรียกใช้ควบคุมอุปกรณ์ ( Framework )
- ส่วนของโปรแกรมช่วยเหลือการทำงานโดยอัตโนมัติ ( Engine )
- ส่วนของโปรแกรมที่ช่วยในการติดตั้งอุปกรณ์ ( Connect Devices )
- ส่วนควบคุมหุ่นยนต์โดยตรง ( Direct Control )
- ส่วนแสดงผลการทำงานจากหุ่นยนต์มายังคอมพิวเตอร์ ( Monitoring )

## 4.5 ซีเควนส์ไดอะแกรม (Sequence Diagram) ของระบบ

ในส่วนนี้จะทำการแสดงถึงขั้นตอนการทำงานของโปรแกรมแอปพลิเคชั่นบนคอมพิวเตอร์ ในส่วนต่าง ๆ ตาม ความต้องการ ที่ได้กำหนดในขั้นต้น โดยแผนภาพจะแสดงรายละเอียดขั้นตอนการใช้งานโปรแกรมควบคุมผ่านทางแอปพลิเคชั่นบนคอมพิวเตอร์รวมถึงขั้นตอนการเรียกใช้งานออปเจ็กต่าง ๆ ที่อยู่ในระบบ

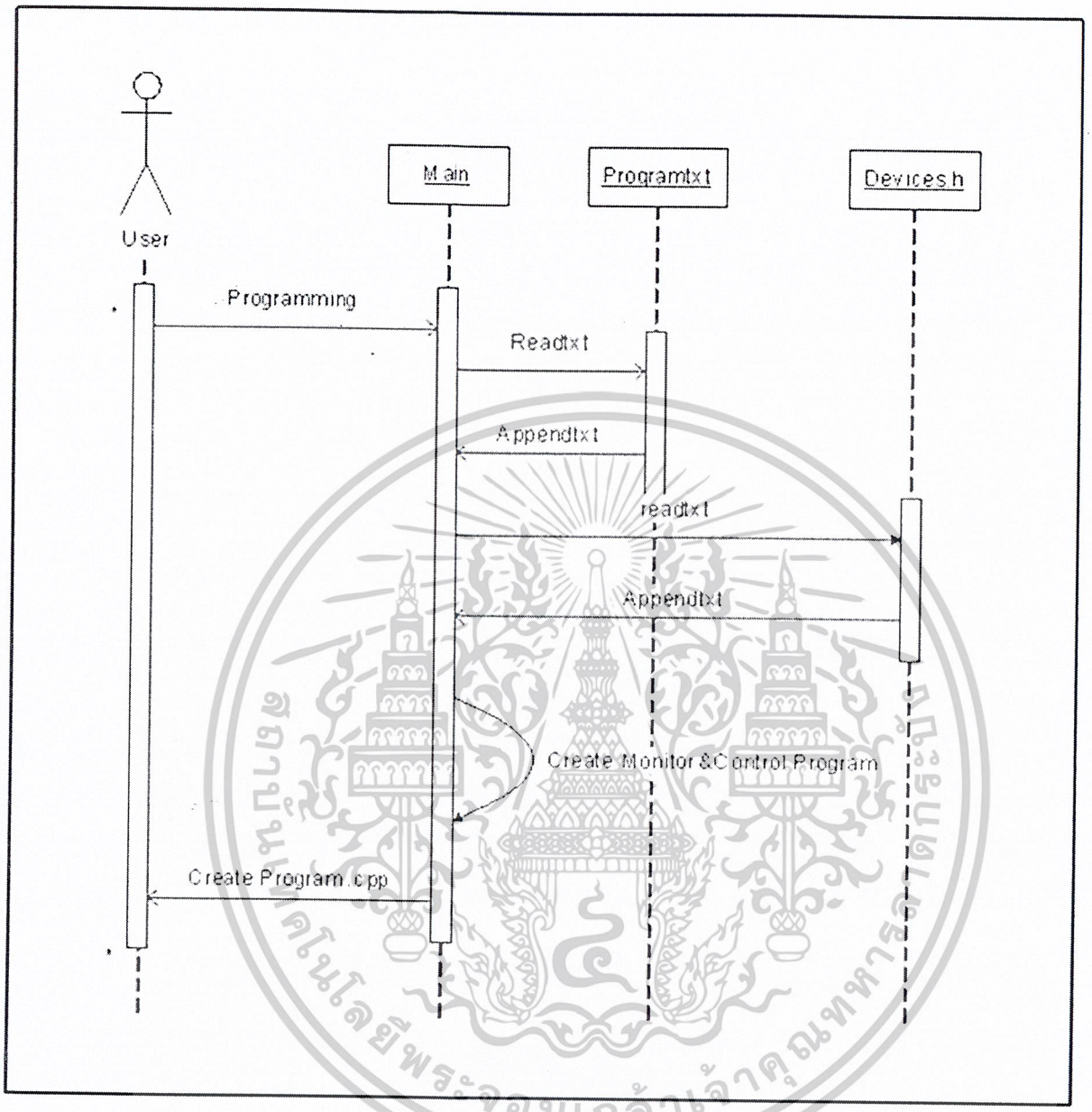
### 4.5.1 ติดตั้งอุปกรณ์กับบอร์ด คอม86 (Connect Devices)




*รูปที่ 4-6 ซีเควนส์ไดอะแกรมของการติดตั้งอุปกรณ์ของผู้ใช้งาน*

แผนภาพอธิบายการทำงานของโปรแกรมในส่วนการเชื่อมต่ออุปกรณ์กับคอม86 โดยผู้ใช้งานเริ่มจากเลือกที่จะติดตั้งอุปกรณ์โปรแกรมหลักจะทำการอ่านค่าไฟล์ข้อมูลอุปกรณ์ มาเก็บไว้เพื่อเตรียมทำการสร้างเฮดเดอร์ไฟล์ และหลังจากได้เลือกพอร์ตและอุปกรณ์จนครบแล้วโปรแกรมหลักจะทำการรวบรวมข้อมูลชนิดของอุปกรณ์พอร์ตที่ใช้แล้วทำการสร้างไฟล์ Devices.h ที่เป็นไลบารี่ของชุดคำสั่งที่เตรียมไว้สำหรับควบคุมอุปกรณ์ที่ติดตั้งไป

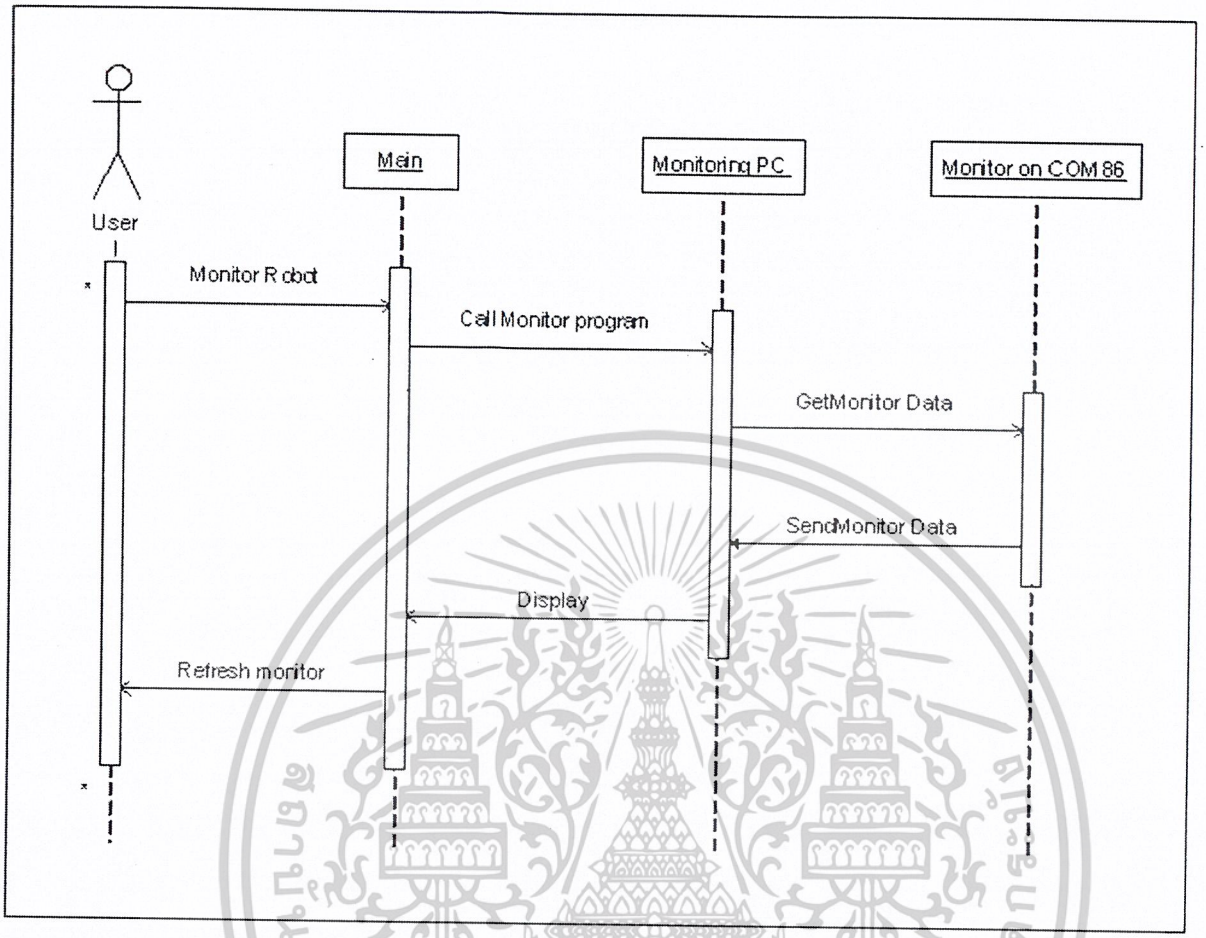
**4.5.2 เขียนโปรแกรมเพื่อใช้ควบคุมอุปกรณ์ ( Programming )**




*รูปที่ 4-7 ซีเควนส์ไดอะแกรมของการเขียนโปรแกรมสั่งงานอุปกรณ์ของผู้ใช้งาน*

แผนภาพอธิบายการทำงานของผู้ใช้งานเรียกใช้โปรแกรมหลักเพื่อทำการเขียนโปรแกรม โดยมีขั้นตอนคือ ผู้ใช้งานจะทำการเรียกแอปพลิเคชั่นว่าจะทำการเขียนโปรแกรม โดยโปรแกรมหลักจะเตรียมข้อมูลเพื่อทำการสร้างไฟล์ Program.cpp โดยการเรียกไฟล์ Devices.h แล้วทำการสร้างโครงสร้างโปรแกรมที่สนับสนุนการใช้งานระบบปฏิบัติการไมโครซีรุ่นที่2 พร้อมทั้งจะทำการสร้างออปเจคของอุปกรณ์ตามข้อมูลที่ได้เก็บบันทึกไว้ จากนั้นโปรแกรมหลักก็จะทำการเพิ่มโปรแกรมในส่วนของการส่งค่าควบคุมหุ่นยนต์ และส่วนโปรแกรมแสดงผลการทำงานไว้ในส่วนท้ายของโปรแกรมแล้วทำการเซฟไฟล์เพื่อให้ผู้ใช้งานนำไฟล์ที่ได้ไปสร้างไฟล์ Program.exe ด้วยการคอมไฟล์บน โปรแกรมบอร์แลนซี (Borland C)

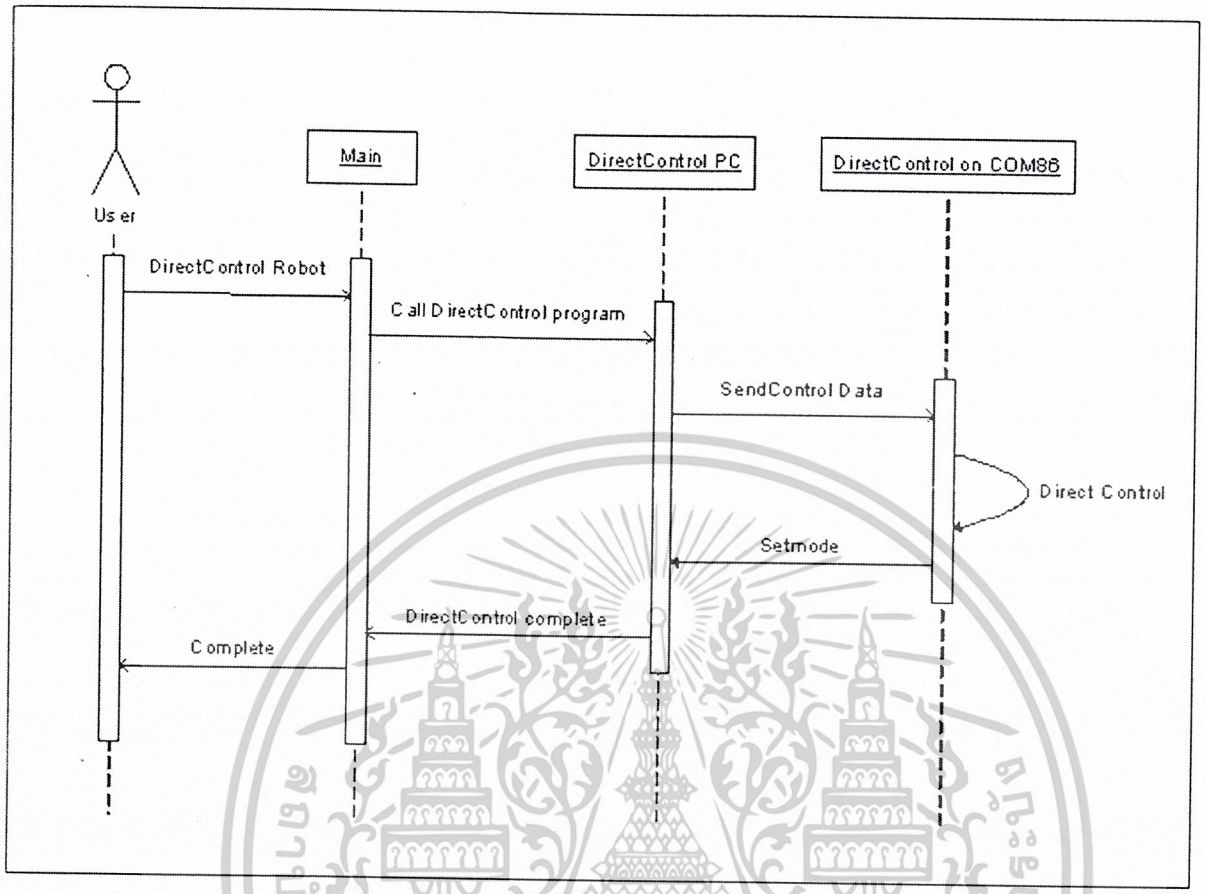
### 4.5.3 ดูสถานะของอุปกรณ์และการทำงานของหุ่นยนต์ (Monitoring)




*รูปที่ 4-8 ซีเควนส์ไดอะแกรมของการดูสถานะของอุปกรณ์และการทำงานของหุ่นยนต์*

แผนภาพอธิบายการทำงานของผู้ใช้งานเรียกดูสถานะ การทำงานของอุปกรณ์ที่ติดตั้งอยู่บนหุ่นยนต์ โดยการเรียกดูค่าสถานะของหุ่นยนต์นั้นสามารถเรียกดูค่าผ่านทางแอปพลิเคชั่นโปรแกรมบนคอมพิวเตอร์ได้ โดยการเรียกดูสถานะการงานของหุ่นยนต์นั้น เริ่มจากโปรแกรมหลักจะทำการเรียกใช้งานโปรแกรมแสดงการทำงานที่ทำงานอยู่บนคอม86 ผ่านทางพอร์ตอนุกรม โดยโปรแกรมแสดงผลที่ทำงานอยู่บนคอม86 จะทำการดึงข้อมูลสถานะของอุปกรณ์บนคอม86 ส่งไปยังคอมพิวเตอร์ผ่านทางพอร์ตเดียวกัน หลังจากโปรแกรมหลักทำการรับค่าจากคอม86 แล้วจะทำการตีความจากข้อมูลที่รับมาได้แล้วทำการแสดงผลการทำงานออกทางมอนิเตอร์ โดยรายละเอียดของการแสดงผลนั้นสามารถดัดแปลงได้ตามความต้องการ โดยโปรแกรมออกแบบมาช่วยเหลือในการช่วยรับค่าและส่งผ่านข้อมูล จากคอม86 มายังคอมพิวเตอร์เท่านั้น

### 4.5.4 ควบคุมการทำงานของหุ่นยนต์โดยตรง (Direct control)




*รูปที่ 4-9 ซีเควนส์ไดอะแกรมของการควบคุมการทำงานของหุ่นยนต์โดยตรง*

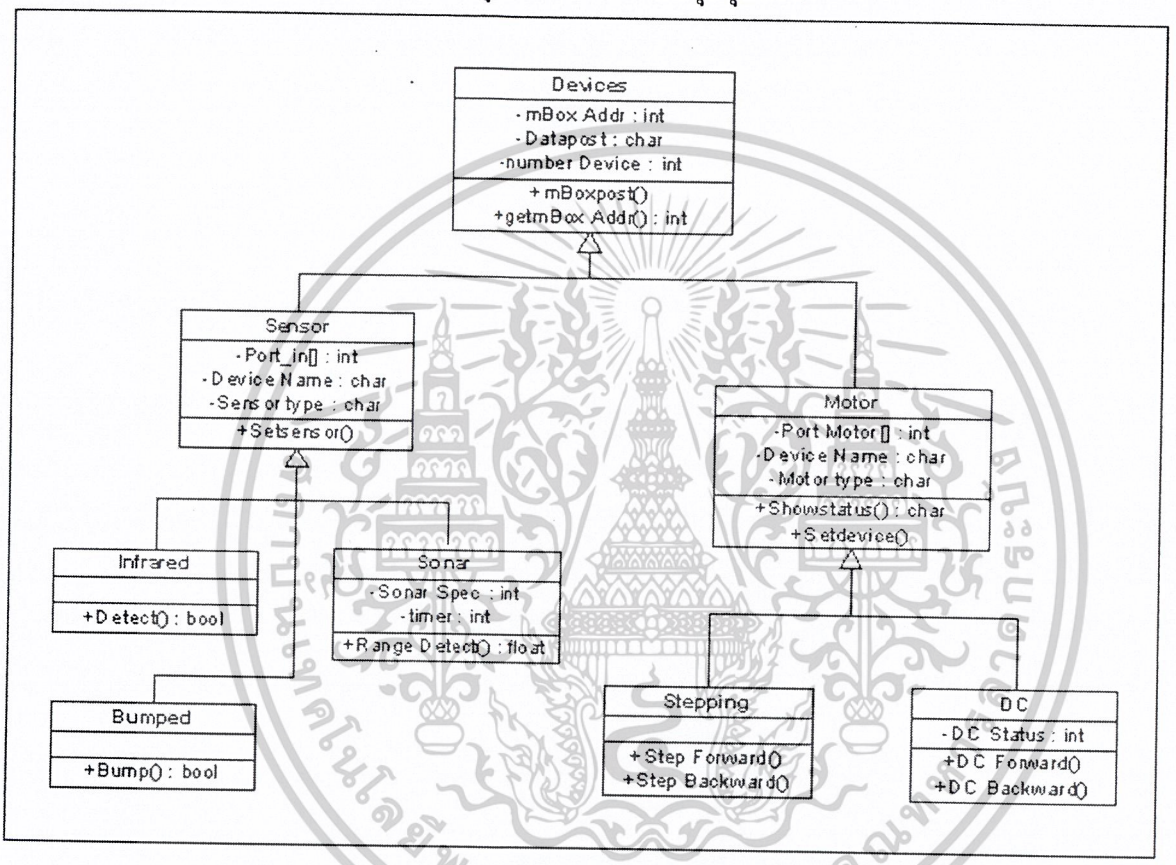
แผนภาพอธิบายการควบคุมหุ่นยนต์ โดยตรงจากคอมพิวเตอร์ ในที่นี้ผู้ใช้งานสามารถส่งค่าไปยังคอม86 เพื่อทำการแก้ไขควบคุมหรือเปลี่ยนแปลงการทำงานของหุ่นยนต์ ได้ตลอดเวลา โดยรายละเอียดขั้นตอนการทำงานของโปรแกรมเริ่มจากผู้ใช้งานเรียกใช้โปรแกรมหลักโดยต้องการที่จะทำการควบคุมอุปกรณ์บนตัวหุ่นยนต์ จากนั้นโปรแกรมหลักจะทำการเรียกโปรแกรมควบคุมการทำงานของอุปกรณ์ที่ทำงานอยู่บนคอม86 โดยการส่งค่าไปบอก หลังจากคอม86ได้เปลี่ยนโหมดการทำงานแล้ว ผู้ใช้งานก็จะสามารถควบคุมหุ่นยนต์จากคอมพิวเตอร์โดยตรงผ่านทางพอร์ตอนุกรมได้

## 4.6 คลาสไดอะแกรม (Class Diagram) ของระบบ

คลาสไดอะแกรมของระบบจะแบ่งออกเป็น 2 ส่วนใหญ่ๆคือ

- คลาสไดอะแกรมของระบบในส่วนของชุดคำสั่งสำหรับควบคุมอุปกรณ์
- คลาสไดอะแกรมของระบบในส่วนของโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์

### 4.6.1 คลาสไดอะแกรมของระบบในส่วนของชุดคำสั่งสำหรับควบคุมอุปกรณ์




*รูปที่ 4-10 คลาสไดอะแกรมในส่วนของชุดคำสั่งสำหรับควบคุมอุปกรณ์ (Framework)*

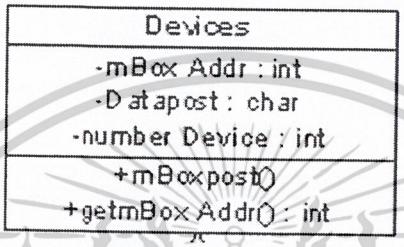
จากภาพเป็นการแสดงรายละเอียดของคลาสภายในโปรแกรมที่เป็นส่วนของชุดคำสั่งที่ใช้สำหรับควบคุมและใช้งานอุปกรณ์ ที่ผู้ใช้งานได้ติดตั้งไว้ ทางคณะผู้จัดทำได้ออกแบบชุดคำสั่งเป็น 2 ส่วนหลัก ด้วยกันคือส่วนของอุปกรณ์ที่ทำการรับค่าเข้ามายังตัวประมวลผล กับส่วนของอุปกรณ์ที่ทำการส่งข้อมูลไปสั่งการ จากภาพจะเห็นส่วนของมอเตอร์ และส่วนของเซนเซอร์แล้วรวมถึงรายละเอียดของคลาสที่ย่อยลงมาโดยในแต่ละคลาสที่ออกแบบนั้น มีรายละเอียดภายใน ดังต่อไปนี้

- คลาสมอเตอร์ ประกอบด้วยคลาสสเต็ปปิ้งมอเตอร์ และ ดีซีมอเตอร์
- คลาสเซนเซอร์ ประกอบด้วยคลาส อินฟราเรด , อัลตราโซนิด และสวิทซ์แบบสัมผัส

โดยคลาสแม่ของโปรแกรมจะมีการเก็บค่าข้อมูลที่จำเป็นต่อการเรียกใช้งาน เช่น ชื่อของออปเจคของอุปกรณ์ พอร์ตที่กำหนดทางลอจิกคอลโดยผู้ใช้งาน และค่าแอดเดสที่กำหนด สำหรับการส่งค่าไปยังเมลบล็อกของแต่ละทาสก์ของชุดคำสั่งที่จะทำการส่งค่าภายในแต่ละทาสก์ของระบบปฏิบัติการไมโครซีรุ่นที่2

ส่วนคลาสลูกที่ทำการอินเฮอร์ลิดแทนส์ ( Inhabitant) มาจะประกอบด้วยฟังก์ชั่น ที่เตรียมไว้สำหรับเรียกใช้เพื่อทำการควบคุมอุปกรณ์ โดยประกอบด้วยชุดคำสั่งสำหรับ ดีซีมอเตอร์, สเต็ปปิ้งมอเตอร์, อินฟราเรดเซนเซอร์, อัลตราโซนิก และสวิทซ์สัมผัส ที่กำหนดเป็นอุปกรณ์พื้นฐาน

#### 4.6.1.1 ลักษณะของคลาส Devices

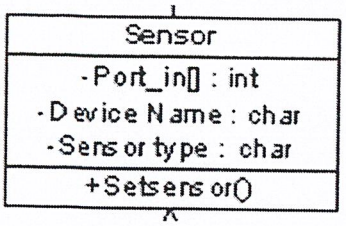



*รูปที่ 4-11 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Devices*

คลาส Devices เป็นคลาสพื้นฐานสำหรับอุปกรณ์ทุกชนิด โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ mBoxAddr เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าตำแหน่งของเมล์บ็อก ที่ใช้ในการส่งค่าข้อมูลควบคุมอุปกรณ์ระหว่างทาสก์ต่างในโปรแกรม
- แอตทริบิวต์ Datapost เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้เก็บค่าข้อมูลที่ส่งไปควบคุมอุปกรณ์แต่ละชนิด
- แอตทริบิวต์ number Device เป็นแอตทริบิวต์ชนิดตัวเลข ที่ใช้เก็บข้อมูลจำนวนอุปกรณ์แต่ละชนิดว่าอุปกรณ์ชนิดนั้นๆมีจำนวนเท่าไร
- เมทธอด mBoxpost() เป็นเมทธอดที่ใช้สำหรับส่งข้อมูลระหว่างกันในแต่ละทาสก์เพื่อนำข้อมูลเหล่านั้นไปใช้ควบคุมอุปกรณ์
- เมทธอด getmBoxAddr() เป็นเมทธอดที่ใช้สำหรับเรียกให้ส่งค่าแอตทริบิวต์ mBoxAddr กลับมา โดยมีการส่งค่ากลับมาเป็นตัวเลข

**4.6.1.2 ลักษณะของคลาส Sensor**

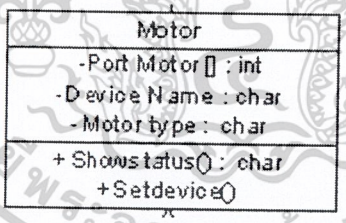



*รูปที่ 4-12 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Sensor*

คลาส Sensor เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิดที่เป็นอินพุตต่างๆ ที่สืบทอดมาจากคลาส Device โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ Port_in[] เป็นแอตทริบิวต์ชนิดชุดของตัวเลข ที่มีไว้สำหรับเก็บค่าตำแหน่งของอุปกรณ์โดยตำแหน่งของอุปกรณ์นี้จะใช้ในการอ้างถึงสำหรับควบคุม(รับข้อมูล)อุปกรณ์
- แอตทริบิวต์ DviceName เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้เก็บค่าชื่อที่ผู้ใช้ตั้งให้กับอุปกรณ์ต่างๆ
- แอตทริบิวต์ Sensortype เป็นแอตทริบิวต์ชนิดตัวอักษร ที่ใช้เก็บข้อมูลระบุชนิดของอุปกรณ์เซนเซอร์ว่าเป็นเซนเซอร์ชนิดใด
- เมทธอด Setsensor() เป็นเมทธอดที่ใช้สำหรับกำหนดค่าเริ่มต้นและกำหนดค่าแอตทริบิวต์ต่างๆของอุปกรณ์เซนเซอร์แต่ละชนิด

**4.6.1.3 ลักษณะของคลาส Motor**



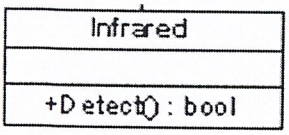

*รูปที่ 4-13 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Motor*

คลาส Motor เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิดที่เป็นเอาท์พุตต่างๆ ที่สืบทอดมาจากคลาส Device โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ PortMotor[] เป็นแอตทริบิวต์ชนิดชุดของตัวเลข ที่มีไว้สำหรับเก็บค่าตำแหน่งของอุปกรณ์โดยตำแหน่งของอุปกรณ์นี้จะใช้ในการอ้างถึงสำหรับควบคุม(ส่งข้อมูล)อุปกรณ์
- แอตทริบิวต์ DeviceName เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้เก็บค่าชื่อที่ผู้ใช้ตั้งให้กับอุปกรณ์ต่างๆ
- แอตทริบิวต์ Motortype เป็นแอตทริบิวต์ชนิดตัวอักษร ที่ใช้เก็บข้อมูลระบุชนิดของอุปกรณ์มอเตอร์ว่าเป็นมอเตอร์ชนิดใด
- เมทธอด Showstatus() เป็นเมทธอดที่ใช้ส่งค่าการทำงานปัจจุบันของมอเตอร์มาให้ผู้ใช้ทราบ

- เมทธอด Setdevice() เป็นเมทธอดที่ใช้สำหรับกำหนดค่าเริ่มต้นและกำหนดค่าแอตทริบิวต์ต่างๆของอุปกรณ์มอเตอร์แต่ละชนิด

#### 4.6.1.4 ลักษณะของคลาส **Infrared**

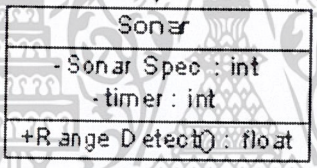



*รูปที่ 4-14 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Infrared*

คลาส Infrared เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิดอินฟราเรดเซนเซอร์ ที่สืบทอดมาจากคลาส Sensorโดยมีส่วนประกอบดังนี้

- เมทธอด Detect() เป็นเมทธอดที่ใช้ส่งค่าการทำงานปัจจุบันของอินฟราเรดเซนเซอร์มาให้ผู้ใช้นำไปใช้งาน โดยจะส่งค่ากลับมาเป็นค่าชนิดตรรกะ

#### 4.6.1.5 ลักษณะของคลาส **Sonar**

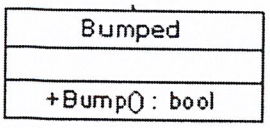



*รูปที่ 4-15 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Sonar*

คลาส Sonar เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิด อัลตราโซนิกเซนเซอร์ ที่สืบทอดมาจากคลาส Sensorโดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ SonarSpec เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลการเปิด-ปิดการใช้งาน อัลตราโซนิกเซนเซอร์
- แอตทริบิวต์ timer เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้เก็บค่าเวลาที่ใช้นับหลังจากมีการส่งค่าข้อมูลของ อัลตราโซนิกเซนเซอร์ เพื่อนำใช้มาคำนวณระยะทาง
- เมทธอด RangeDetect() เป็นเมทธอดที่ใช้คำนวณและส่งค่าระยะทางที่ อัลตราโซนิกเซนเซอร์ วัดได้ไปให้ผู้ใช้นำไปใช้งาน โดยจะส่งค่ากลับมาเป็นค่าชนิดตัวเลขทศนิยม

**4.6.1.6 ลักษณะของคลาส Bumped**

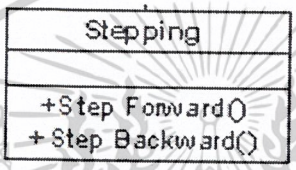



*รูปที่ 4-16 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Bumped*

คลาส Bumped เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิด บัมพ์เซนเซอร์ ที่สืบทอดมาจากคลาส Sensor โดยมีส่วนประกอบดังนี้

- เมทธอด Bump() เป็นเมทธอดที่ใช้ส่งค่าการทำงานปัจจุบันของ บัมพ์เซนเซอร์ มาให้ผู้ใช้นำไปใช้งาน โดยจะส่งค่ากลับมาเป็นค่าชนิดตรรกะ

**4.6.1.7 ลักษณะของคลาส Stepping**

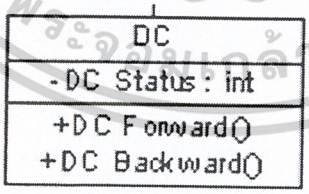



*รูปที่ 4-17 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Stepping*

คลาส Stepping เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิด สเต็ปมอเตอร์ ที่สืบทอดมาจากคลาส Motor โดยมีส่วนประกอบดังนี้

- เมทธอด StepForward() เป็นเมทธอดที่ใช้ส่งค่าสั่งงานให้ สเต็ปมอเตอร์ เคลื่อนที่ไปข้างหน้า
- เมทธอด StepBackward() เป็นเมทธอดที่ใช้ส่งค่าสั่งงานให้ สเต็ปมอเตอร์ เคลื่อนที่กลับไปข้างหลัง

**4.6.1.8 ลักษณะของคลาส DC**

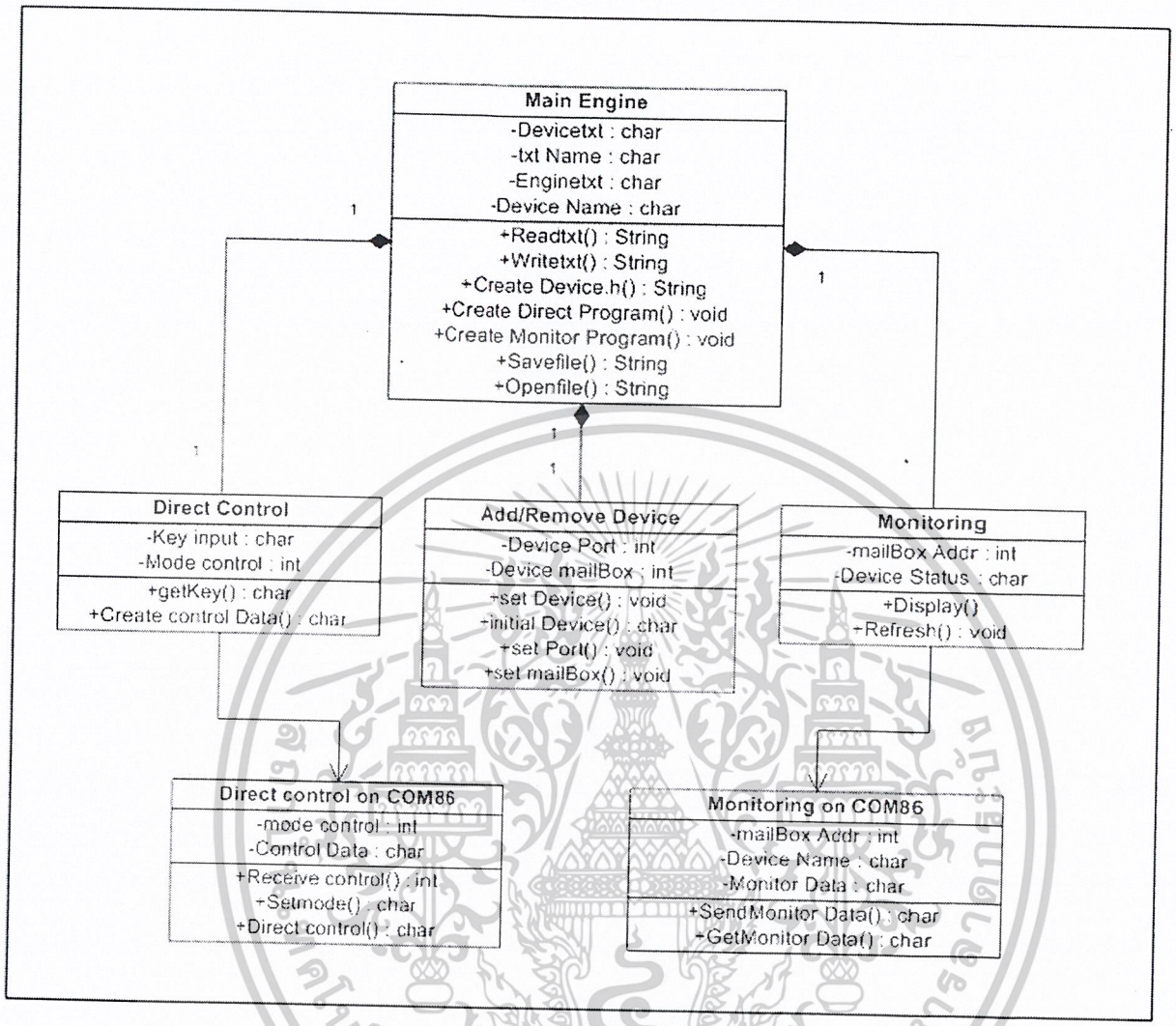



*รูปที่ 4-18 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส DC*

คลาส DC เป็นคลาสพื้นฐานสำหรับอุปกรณ์ชนิด ดีซีมอเตอร์ ที่สืบทอดมาจากคลาส Motorโดยมี. ส่วนประกอบดังนี้

- แอตทริบิวต์ DCstatus เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลสถานะการทำงานของ ดีซีมอเตอร์
- เมทธอด DCForward() เป็นเมทธอดที่ใช้ส่งค่าสั่งงานให้ ดีซีมอเตอร์ เคลื่อนที่ไปข้างหน้า
- เมทธอด DCBackward() เป็นเมทธอดที่ใช้ส่งค่าสั่งงานให้ ดีซีมอเตอร์ เคลื่อนที่กลับหลัง

### 4.6.2 คลาสไดอะแกรมของระบบในส่วนของโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์



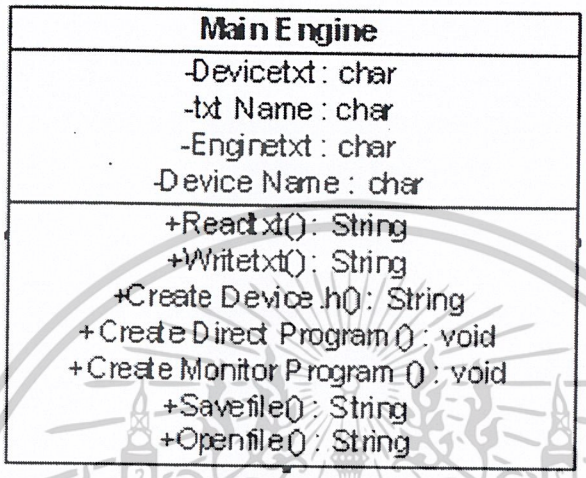

*รูปที่ 4-19 คลาสไดอะแกรมในส่วน โปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์ (Engine)*

จากภาพเป็นการแสดงรายละเอียดของคลาสที่เป็นส่วนประกอบภายในโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์ ที่จะเป็นโปรแกรมที่จะทำการช่วยเหลือผู้ใช้งานให้สามารถจัดการเกี่ยวกับการติดตั้งอุปกรณ์ การเขียนโปรแกรม การสร้างโปรแกรมที่สามารถทำงานในแบบมัลติทาสก์ รวมถึงการเพิ่มโปรแกรมในส่วนการแสดงผลการทำงานและส่งค่าควบคุมการทำงานของหุ่นยนต์ผ่านทางแอปพลิเคชั่นบนคอมพิวเตอร์ โดยในแต่ละคลาสจะมีการทำงานดังต่อไปนี้

- คลาสของส่วนโปรแกรมหลัก
- คลาสของการติดตั้งอุปกรณ์
- คลาสของการรับค่าการทำงานมาแสดงผลบนคอมพิวเตอร์
- คลาสของการส่งข้อมูลควบคุมการทำงานของหุ่นยนต์โดยตรง

โดยในส่วนของคลาสแสดงผลการทำงาน และคลาสควบคุมการทำงานนั้น จะแบ่งออกเป็นคลาสย่อยที่ทำงานอยู่บนคอม86 ด้วยอีกส่วนหนึ่งเพื่อให้โปรแกรมหลักนั้น สามารถเรียกใช้ให้หุ่นยนต์สามารถส่งข้อมูลและรับข้อมูลมายังคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม

#### 4.6.2.1 ลักษณะของคลาส MainEngine



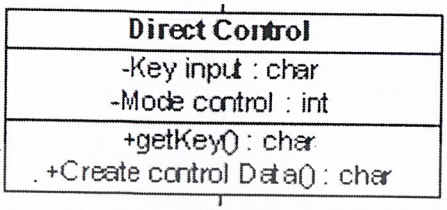

*รูปที่ 4-20 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส MainEngine*

คลาส MainEngine เป็นคลาสหลักของส่วนโปรแกรมช่วยเหลือผู้ใช้งาน โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ Devicetxt เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าชื่อของแฟ้มข้อมูลของอุปกรณ์ต่างๆ
- แอตทริบิวต์ txtName เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าชื่อของแฟ้มข้อมูลโปรแกรมหลักที่สร้างจากโปรแกรมช่วยเหลือผู้ใช้
- แอตทริบิวต์ Enginetxt เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าชื่อของแฟ้มข้อมูลของส่วนประกอบของโปรแกรมช่วยเหลือผู้ใช้
- แอตทริบิวต์ DeviceName เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าชื่อของอุปกรณ์ต่างๆ
- เมทธอด Readtxt() เป็นเมทธอดที่ใช้อ่านแฟ้มข้อมูลเพื่อนำมาประมวลผลสร้างโปรแกรมหลัก
- เมทธอด Writetxt() เป็นเมทธอดที่ใช้เขียนค่าลงไปในแฟ้มข้อมูลโปรแกรมหลัก
- เมทธอด CreateDevice.h() เป็นเมทธอดที่ใช้สร้างแฟ้มข้อมูลแบบเฮดเดอร์ ของอุปกรณ์ที่ผู้ใช้เลือก
- เมทธอด CreateDirectProgram() เป็นเมทธอดที่ใช้สร้างส่วนประกอบแฟ้มข้อมูลโปรแกรมหลักในส่วนทาสก์ที่ใช้ควบคุมอุปกรณ์โดยตรงจากผู้ใช้
- เมทธอด CreateMonitorProgram() เป็นเมทธอดที่ใช้สร้างส่วนประกอบแฟ้มข้อมูลโปรแกรมหลักในส่วนทาสก์ที่ใช้แสดงผลสถานะการทำงานของอุปกรณ์ต่างๆ

- เมทธอด Savefile() เป็นเมทธอดที่ใช้บันทึกแฟ้มข้อมูลโปรแกรมหลักที่โปรแกรมช่วยเหลือสร้างขึ้นมาแล้วผู้ใช้นำไปเขียนโปรแกรมในส่วนของทาสก์ผู้ใช้
- เมทธอด Openfile() เป็นเมทธอดที่ใช้เปิดแฟ้มข้อมูลโปรแกรมหลักที่ผู้ใช้เคยบันทึกไว้แล้ว

**4.6.2.2 ลักษณะของคลาส DirectControl**

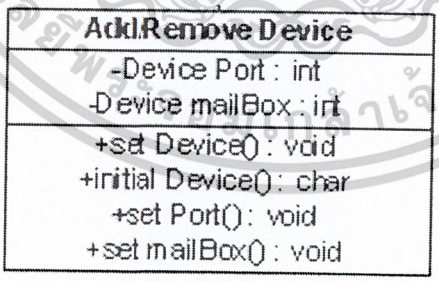



*รูปที่ 4-21 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส DirectControl*

คลาส DirectControl เป็นคลาสที่เป็นส่วนประกอบของคลาส MainEngine โดยจะทำงานในส่วนการควบคุมอุปกรณ์โดยตรงของโปรแกรมช่วยเหลือผู้ใช้งาน โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ Keyinput เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าอินพุตที่รับจากผู้ใช้โดยวิธีการกดคีย์บอร์ด
- แอตทริบิวต์ Modecontrol เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลสถานะการทำงานของการควบคุมอุปกรณ์
- เมทธอด getKey() เป็นเมทธอดที่ใช้รับค่าข้อมูลจากผู้ใช้ผ่านทางคีย์บอร์ด
- เมทธอด CreatecontrolData() เป็นเมทธอดที่ใช้สร้างข้อมูลที่ส่งไปควบคุมอุปกรณ์แต่ละชนิดโดยตรง

**4.6.2.3 ลักษณะของคลาส Add/Remove Device**




*รูปที่ 4-22 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Add/Remove Device*

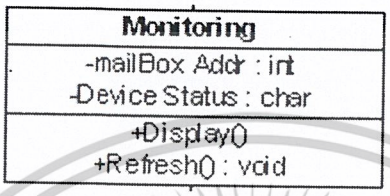
คลาส Add/Remove Device เป็นคลาสที่เป็นส่วนประกอบของคลาส MainEngine โดยจะทำงานในส่วนการติดตั้งอุปกรณ์ของโปรแกรมช่วยเหลือผู้ใช้งาน โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ DevicePort เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลตำแหน่งที่ใช้อ้างถึงอุปกรณ์ต่างๆที่ผู้ใช้เลือกไว้
- แอตทริบิวต์ DevicemailBox เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูล

เมล์บ็อกที่ใช้ส่งข้อมูลของแต่ละอุปกรณ์

- เมทธอด setDevice() เป็นเมทธอดที่ใช้กำหนดชนิดอุปกรณ์ที่ผู้ใช้เลือก
- เมทธอด initialDevice() เป็นเมทธอดที่ใช้กำหนดค่าเริ่มต้นของอุปกรณ์ที่ผู้ใช้เลือก
- เมทธอด setPort() เป็นเมทธอดที่ใช้กำหนดค่าตำแหน่งที่ใช้อ้างถึงอุปกรณ์ที่ผู้ใช้เลือก
- เมทธอด setmailBox() เป็นเมทธอดที่ใช้กำหนดค่าเมล์บ็อกให้กับอุปกรณ์ทุกตัวที่ผู้ใช้เลือก

#### 4.6.2.4 ลักษณะของคลาส Monitoring

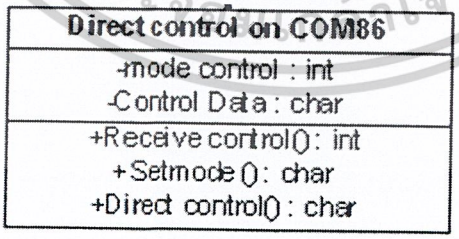



*รูปที่ 4-23 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Monitoring*

คลาส Monitoring เป็นคลาสที่เป็นส่วนประกอบของคลาส MainEngine โดยจะทำงานในส่วนการแสดงสถานะของอุปกรณ์ต่างให้ผู้ใช้ทราบทางจอแสดงผลของโปรแกรมช่วยเหลือผู้ใช้งาน โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ mailBoxAddr เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลตำแหน่งเมล์บ็อกที่ใช้ส่งข้อมูลของแต่ละอุปกรณ์
- แอตทริบิวต์ DeviceStatus เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าข้อมูลสถานะการทำงานของอุปกรณ์แต่ละตัว
- เมทธอด Display() เป็นเมทธอดที่ใช้แสดงสถานะของอุปกรณ์ต่างทางจอแสดงผล
- เมทธอด Refresh() เป็นเมทธอดที่ใช้เปลี่ยนค่าที่แสดงผลใหม่

#### 4.6.2.5 ลักษณะของคลาส Direct control on Com86



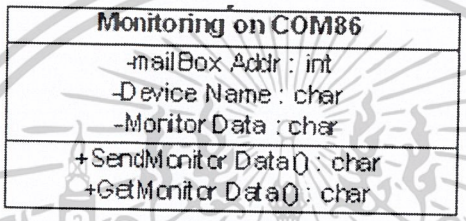

*รูปที่ 4-24 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Direct control on Com86*

คลาส Direct control on Com86 เป็นคลาสย่อยที่เกี่ยวข้องกับคลาส DirectControl โดยจะทำงานในส่วนการควบคุมอุปกรณ์โดยตรงที่จะทำงานภายในคอม86 ของโปรแกรมช่วยเหลือผู้ใช้งาน โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ modecontrol เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลสถานะการควบคุมอุปกรณ์ของโปรแกรมช่วยเหลือผู้ใช้งาน

- แอตทริบิวต์ ControlData เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าข้อมูลที่ส่งไปสั่งงานอุปกรณ์ต่างๆโดยตรง
- เมทธอด Receivecontrol() เป็นเมทธอดที่ใช้รับค่าข้อมูลคำสั่งควบคุมอุปกรณ์จากโปรแกรมช่วยเหลือผู้ใช้งานที่อยู่บนคอมพิวเตอร์
- เมทธอด Setmode() เป็นเมทธอดที่ใช้รับค่าข้อมูลการกำหนดสถานะการควบคุมโปรแกรมช่วยเหลือผู้ใช้งานที่อยู่บนคอมพิวเตอร์มากำหนดให้กับโปรแกรมที่กำลังทำงานอยู่ที่คอม86
- เมทธอด Directcontrol() เป็นเมทธอดที่ใช้ส่งค่าข้อมูลคำสั่งควบคุมอุปกรณ์จากคอม86ไปยังตัวอุปกรณ์นั้นๆ

**4.6.2.6 ลักษณะของคลาส Monitoring on Com86**

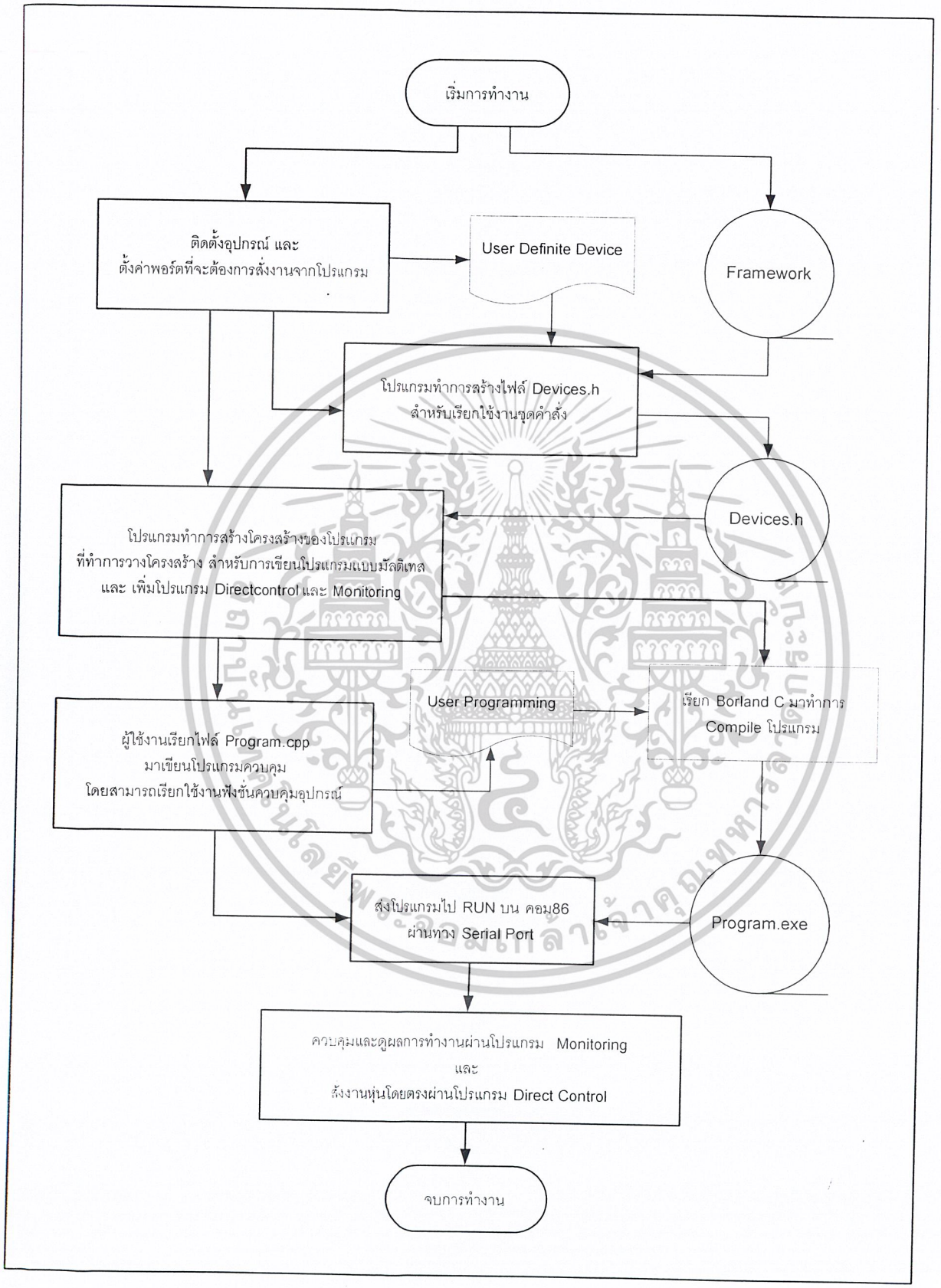



*รูปที่ 4-25 รายละเอียดแอตทริบิวต์และเมทธอดของคลาส Monitoring on Com86*

คลาส Monitoring on Com86 เป็นคลาสย่อยที่เกี่ยวข้องกับคลาส Monitoring โดยจะทำงานในส่วนการแสดงผลสถานะการทำงานของอุปกรณ์ที่จะทำงานภายในคอม86 ของโปรแกรมช่วยเหลือผู้ใช้งาน โดยจะส่งข้อมูลไปยังส่วนของคอมพิวเตอร์ โดยมีส่วนประกอบดังนี้

- แอตทริบิวต์ mailBoxAddr เป็นแอตทริบิวต์ชนิดตัวเลข ที่มีไว้สำหรับเก็บค่าข้อมูลตำแหน่งเมล์บ็อกที่ใช้ส่งข้อมูลของแต่ละอุปกรณ์
- แอตทริบิวต์ DeviceName เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าชื่อของอุปกรณ์ชนิดต่างๆที่ผู้ใช้ตั้งให้
- แอตทริบิวต์ MonitorData เป็นแอตทริบิวต์ชนิดตัวอักษร ที่มีไว้สำหรับเก็บค่าข้อมูลสถานะการทำงานของอุปกรณ์แต่ละตัวที่จะนำไปแสดงผล
- เมทธอด SendMonitorData() เป็นเมทธอดที่ใช้ส่งค่าข้อมูลสถานะการทำงานของอุปกรณ์แต่ละตัวไปยังคอมพิวเตอร์เพื่อแสดงผลต่อไป
- เมทธอด GetMonitorData() เป็นเมทธอดที่ใช้รับค่าข้อมูลสถานะการทำงานของอุปกรณ์แต่ละตัวที่ทำงานอยู่ ณ.ปัจจุบัน

## 4.7 แผนภาพแสดงการทำงานของโปรแกรมจัดการระบบ (Design Flow)

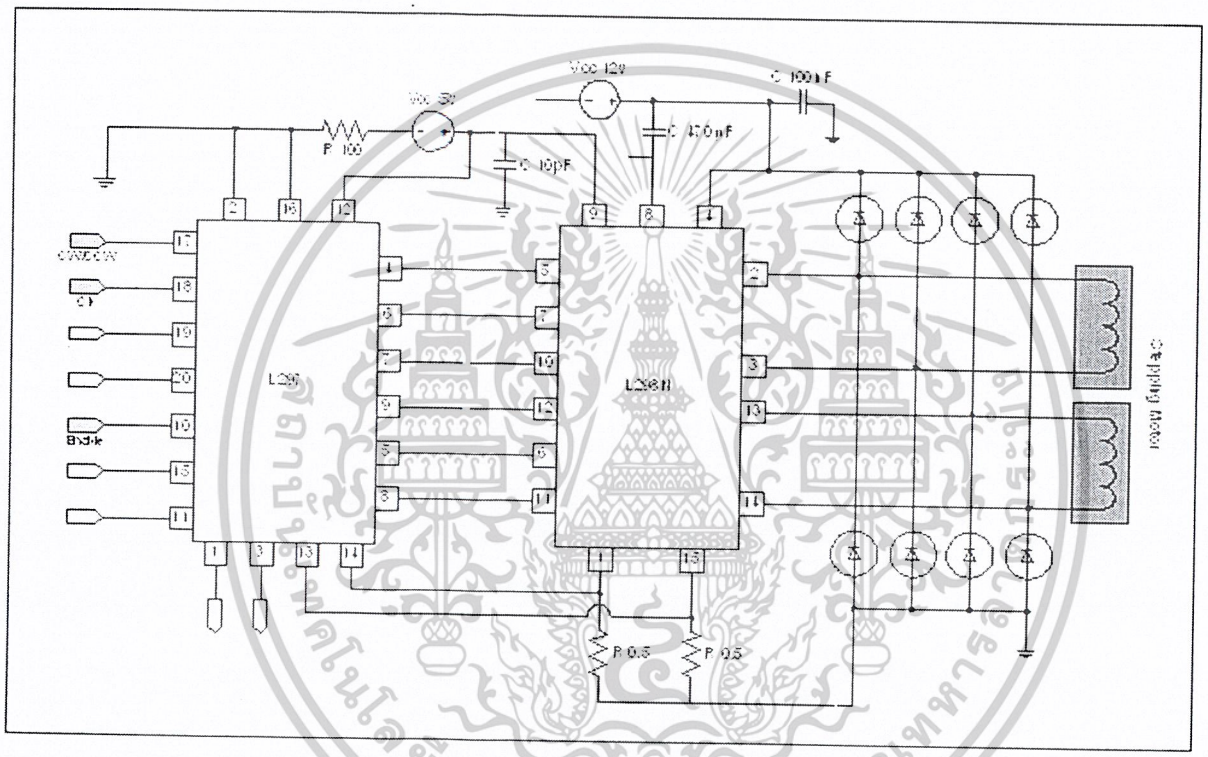



*รูปที่ 4-26 แผนภาพแสดงการทำงานของโปรแกรมจัดการระบบ (Design Flow)*

## 4.8 วงจรโมดูลของอุปกรณ์ (Devices Module)

เนื่องจากการเรียกใช้ชุดคำสั่งที่เตรียมไว้นั้น จะต้องมีการกำหนดค่าพอร์ตของอุปกรณ์ ที่จะต้องติดต่อกับไมโครคอนโทรลเลอร์อย่างถูกต้อง และเพื่อให้เข้ากันกับโปรแกรมช่วยเหลือผู้ใช้งานบนคอมพิวเตอร์ ทางคณะผู้จัดทำโครงงานจึงได้ทำการออกแบบวงจรสำหรับใช้งานอุปกรณ์ และเป็นวงจรที่สามารถเรียกใช้งานชุดคำสั่งที่เตรียมการไว้อย่างถูกต้อง โดยไม่มีปัญหาแต่อย่างไร

### 4.8.1 สเต็ปปิ้งมอเตอร์

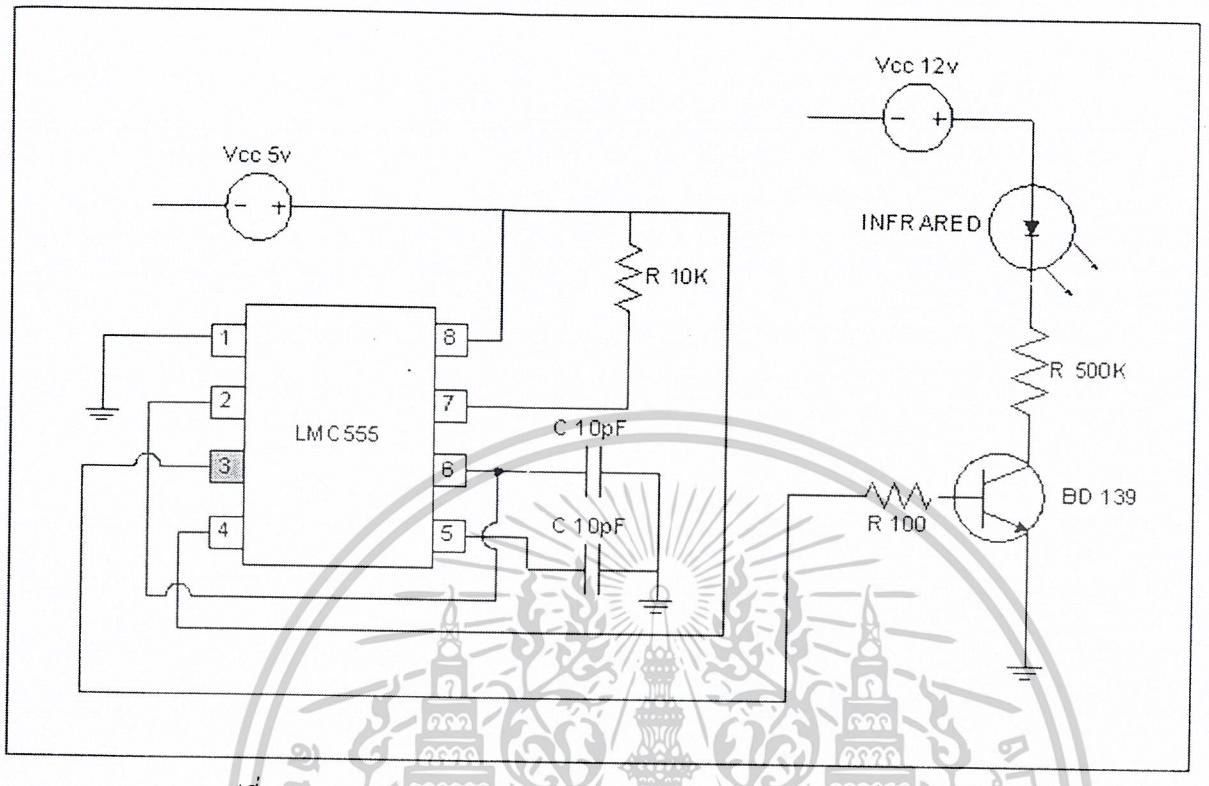



*รูปที่ 4-27 แผนภาพแสดงโมดูลของการขับสเต็ปมอเตอร์ด้วย L297-L298N*

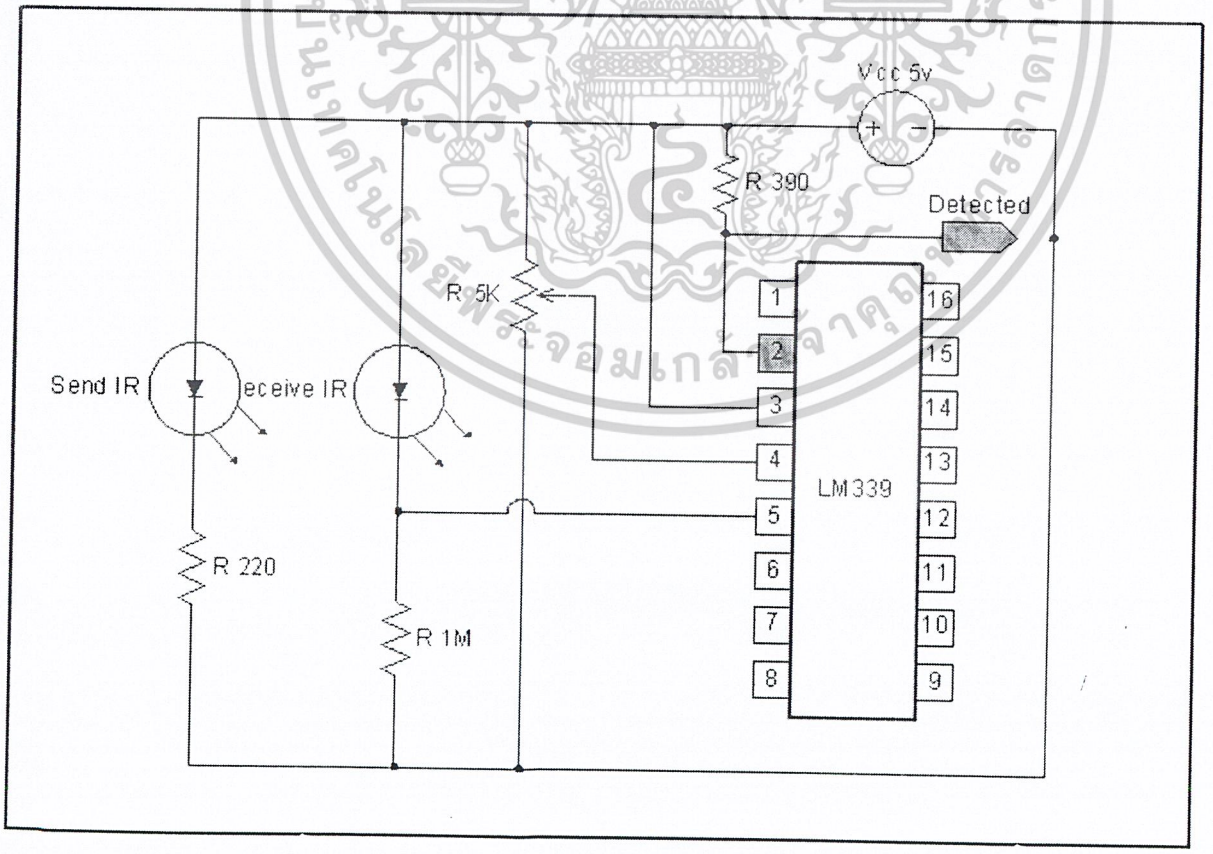
โมดูลทำการขับเคลื่อนอุปกรณ์สเต็ปมอเตอร์ ตัวนี้ประกอบด้วยไอซี L297 ที่ทำหน้าที่เลื่อนบิตและทำการกำหนดทิศทางการเลื่อนบิต รวมถึงการเพิ่มบิตที่ใช้งาน ซึ่งเป็นไอซีที่เหมาะกับการใช้งานควบคุมสเต็ปมอเตอร์อย่างมาก โดยขั้นตอนต่อไป ก็คือการเพิ่มกระแสในการขับเคลื่อนมอเตอร์ โดยทางคณะผู้จัดทำได้เลือก L298N ที่เป็นไอซีสำหรับการเพิ่มกระแสในการขับมอเตอร์ที่รอรับการจ่ายไฟถึง 36 โวลต์ ซึ่งค่อนข้างจะครอบคลุมการใช้งานมอเตอร์เป็นอย่างมาก

โดยในส่วนการเพิ่มกระแส เพื่อขับมอเตอร์ของไอซี L298N นั้นสามารถประยุกต์ใช้งานกับการขับมอเตอร์ชนิดอื่น ๆ อีกด้วย โดยการเลื่อกขาอินพุตไปยัง ดีซีมอเตอร์ หรือรวมถึงการสร้างวงจรขับมอเตอร์แบบเฮดบริดซ์ (H-Bridge) สำหรับการควบคุมมอเตอร์ดีซี โดยจะไม่ยกตัวอย่างในที่นี่โดยจะสามารถหาวิธีการจากภาคผนวกท้ายเล่มได้

### 4.8.2 อินฟราเรดเซนเซอร์




*รูปที่ 4-28 แผนภาพแสดงโมดูลของการส่งอินฟราเรดด้วยความถี่ 38 KHz*




*รูปที่ 4-29 แผนภาพแสดงโมดูลของวงจรตรวจสอบด้วยอินฟราเรดเซนเซอร์*

# บทที่ 5
# การทดสอบและวิเคราะห์ผล

## 5.1 วัตถุประสงค์การทดลอง

- ทำการศึกษาวิธีการนำโปรแกรมต่างๆไปใช้งานกับคอม86
- ทำการศึกษาวิธีการนำไมโครซีโอเอส2 มาใช้ทำงานร่วมกับคอม86
- ทำการศึกษาและทดสอบนำโปรแกรมที่สร้างขึ้นมา ควบคุมอุปกรณ์ต่างๆที่จะนำมาสร้างเป็นหุ่นยนต์

โดยวัตถุประสงค์หลักที่ต้องการในการทดสอบนั้น คือการนำไมโครซีโอเอส2 ที่เป็นระบบหลักในการทำงานแบบมัลติทาสก์บนคอม86 เพื่อให้สามารถทำงานหลายโปรแกรมในเวลาเดียวกัน และทำให้คอม86 มีความสามารถในการรองรับการทำงานที่โครงงานนี้ต้องการ

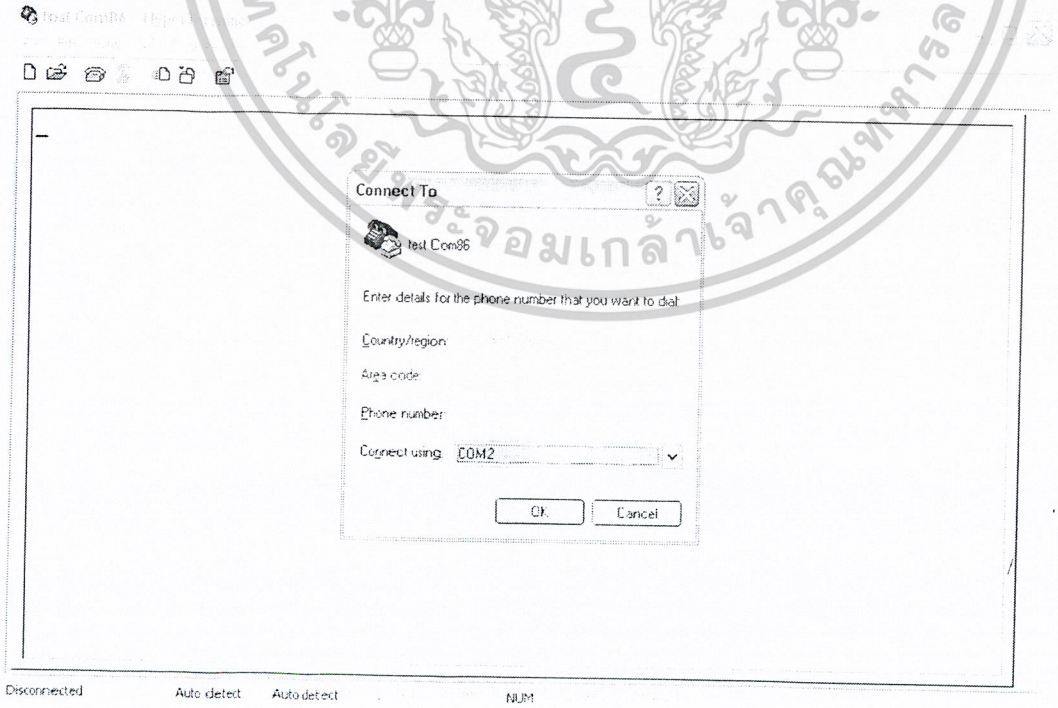
### 5.1.1 ทำการศึกษาวิธีการนำโปรแกรมต่างๆไปใช้งานกับคอม86

วิธีการทดลอง

- เปิดการใช้งาน ไฮเปอร์เทอร์มินอล (Hyper terminal)
- ทำการปรับแต่งค่าเริ่มต้นในการติดต่อกับคอม86 แล้วเปิดการใช้งานคอม86
- พิมพ์คำสั่งส่งแฟ้มข้อมูล แล้วเลือกแฟ้มข้อมูลที่จะใช้งานส่งจาก ไฮเปอร์เทอร์มินอล ไปยัง คอม86
- สั่งให้โปรแกรมทำงานบนคอม86ผ่านทาง ไฮเปอร์เทอร์มินอล แล้วสังเกตการทำงานของโปรแกรม

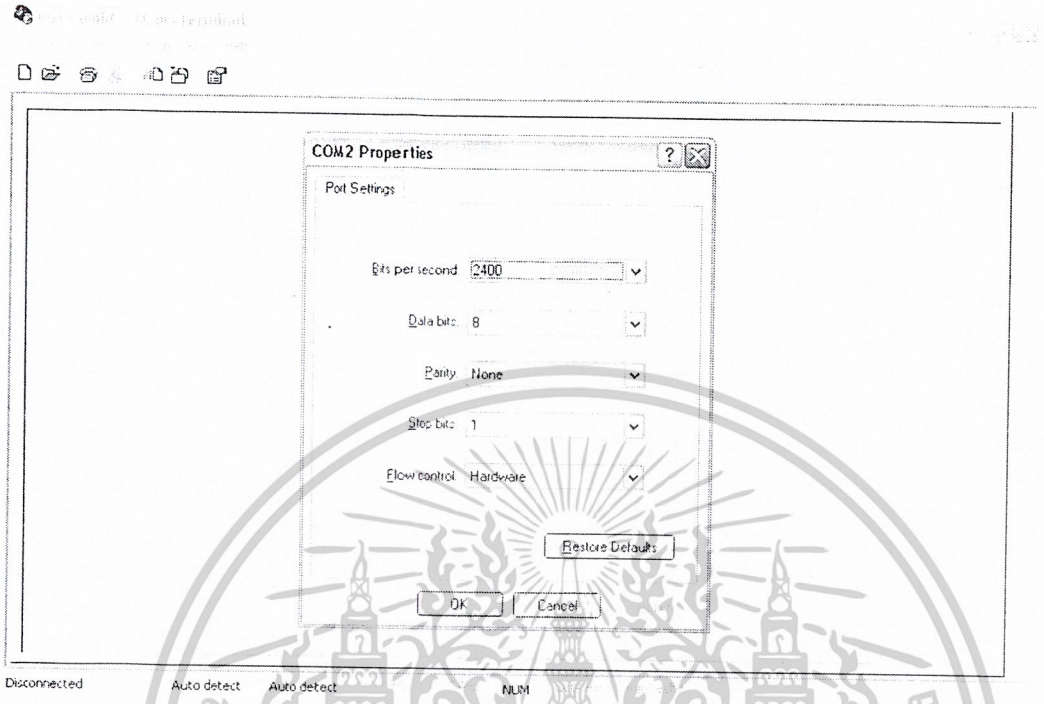
ผลการทดลอง

กำหนดให้ติดต่อผ่านพอร์ตอนุกรม



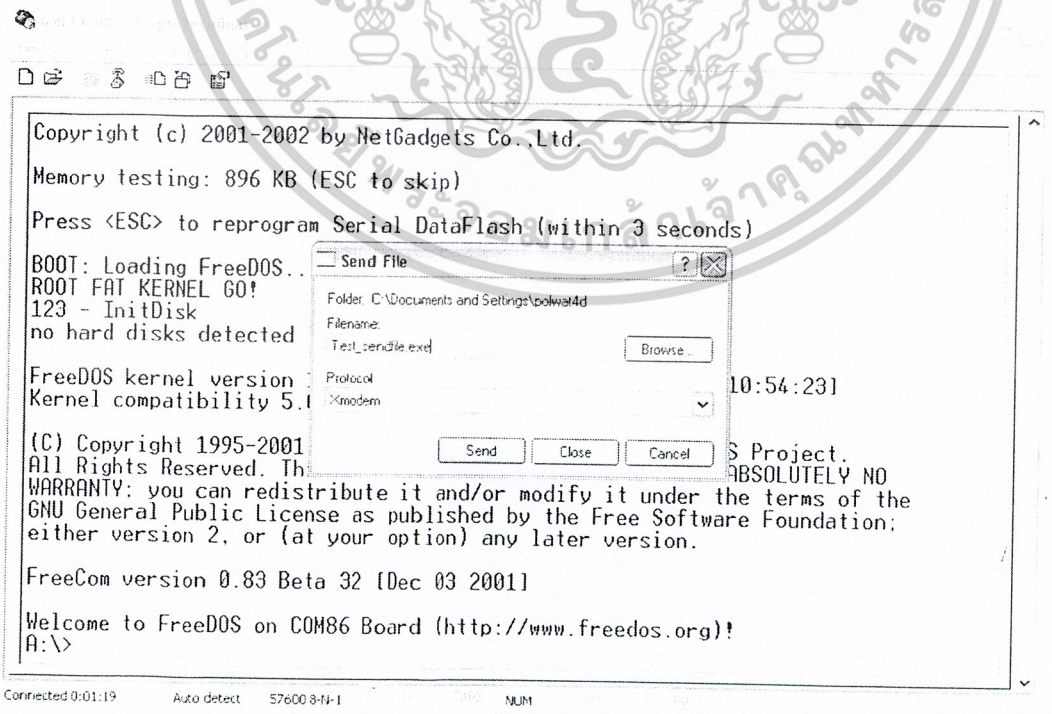

*รูปที่ 5-1 แสดงการตั้งพอร์ตที่ติดต่อระหว่างคอมพิวเตอร์กับคอม86*

กำหนดค่าการความเร็วในการส่งข้อมูล จำนวนข้อมูลในการส่งแต่ละครั้ง การตรวจสอบความถูกต้องของข้อมูล และการควบคุมการส่ง ดังรูป




*รูปที่ 5-2 แสดงการตั้งค่าไฮเปอร์เทอร์มินอล*

ใช้ไฮเปอร์เทอร์มินอลส่งแฟ้มข้อมูลไปยังคอม86 โดยใช้คำสั่ง transfer /r <filename> แล้วเลือกแฟ้มข้อมูลที่จะส่ง อาจจะเลือกโปรโตคอล Xmodem ดังตัวอย่างได้




*รูปที่ 5-3 การส่งไฟล์ไปยังคอม86ด้วยโปรแกรมไฮเปอร์เทอร์มินอล*

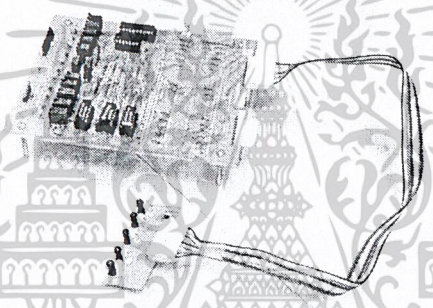
**5.1.2 ทำการทดสอบนำโปรแกรมทดสอบการทำงาน ไอซ่าบัส (ISA bus) ของคอม86**

วิธีการทดลอง

- นำอุปกรณ์ทดสอบไอซ่าบัสติดตั้งกับคอม86
- ส่งโปรแกรมทดสอบไอซ่าบัสไปยังคอม86
- สั่งให้โปรแกรมทำงาน แล้วสังเกตผลการทดลอง
- ปรับแต่งโปรแกรมทดสอบไอซ่าบัส แล้วทำการทดลองใหม่

ผลการทดลอง

จากโปรแกรมการทดสอบ ไอซ่าบัส สามารถทำให้คอม86 สั่งงานและรับค่าข้อมูลจาก ไอซ่าบัสได้โดย จากการทดลองสามารถทำให้คอม86 รับค่าจากสวิตช์แล้วสามารถแสดงผลสั่งงานหลอดแสดงผล ดังรูป



*รูปที่ 5-4 แสดงตัวอย่างการใช้งานไอซ่าบัส*

**5.1.3 ทำการทดสอบนำโปรแกรมทดสอบการทำงาน อินเทอร์รัพต์ (Interrupt) ของคอม86**

วิธีการทดลอง

- ทำการสร้างอุปกรณ์ทดสอบอินเทอร์รัพต์ แล้วนำไปติดตั้งกับคอม86
- ส่งโปรแกรมทดสอบอินเทอร์รัพต์ ไปยังคอม86
- สั่งให้โปรแกรมทำงาน แล้วสังเกตผลการทดลอง
- ปรับแต่งโปรแกรมทดสอบอินเทอร์รัพต์ แล้วทำการทดลองใหม่

ผลการทดลอง

จากโปรแกรมทดสอบการใช้งานอินเทอร์รัพต์ สามารถนำคอม86 ไปทำงานที่ต้องการใช้งานแบบอินเทอร์รัพต์ได้โดยเรียกใช้งานได้จาก อินเทอร์รัพต์รีเควส (IRQ) หมายเลขต่างๆในพอร์ตไอซ่าบัสและพอร์ตพีไอโอ บนคอม86

จากการทดสอบการทำงานของคอม86 สามารถแสดงให้เห็นได้ว่าสามารถนำคอม86 มาพัฒนาให้ทำงานเป็นหุ่นยนต์โดยใช้ส่วนประกอบต่างๆที่คอม86 มีไว้ภายในได้ แต่จากการทดลองยังพบว่าต้องทดสอบส่วนของโมดูลอุปกรณ์ที่นำมาติดต่อกับไอซ่าบัสให้เสถียรก่อน มิฉะนั้นคอม86 อาจเกิดปัญหา ในการทำงานได้

**5.2 ทำการศึกษาวิธีการนำระบบปฏิบัติการไมโครซีรุ่นที่2 มาใช้ทำงานร่วมกับคอม86**

จากการที่โครงงานต้องการให้คอม86 สามารถทำงานแบบมัลติทาสก์ และสามารถเรียกใช้งานเมลบล๊อกในการส่งค่าภายในทาสก์ โดยจากการศึกษาพบว่า ไมโครซีโอเอส2 นั้นสนับสนุนการทำงานในรูปแบบที่ต้องการและยังเปฝ็นระบบปฏิบัติการที่เสถียรและมีการทำงานที่โดดเด่น และใช้งานอย่างแพร่หลายในปัจจุบัน

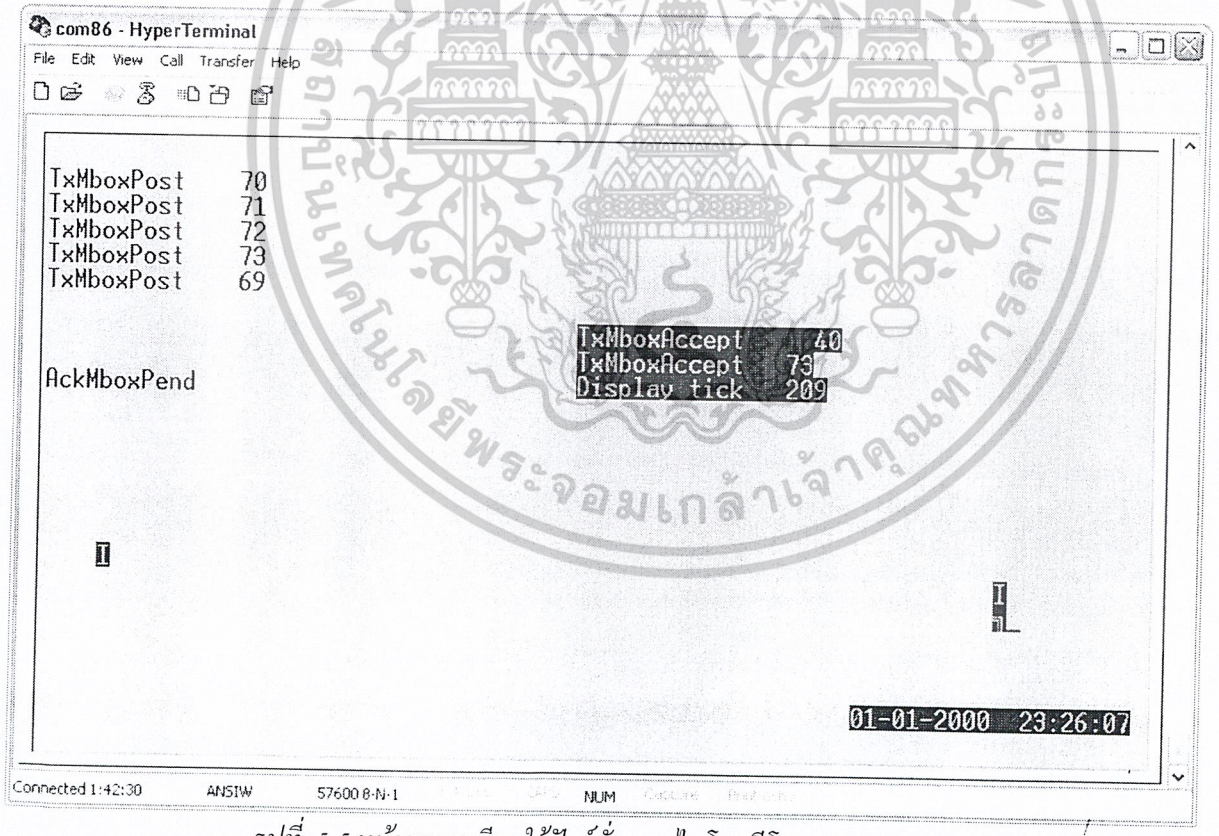
**5.2.1 ทำการทดสอบการทำงานฟังก์ชันต่างๆ ของระบบปฏิบัติการไมโครซีรุ่นที่2บนคอม86**

วิธีการทดลอง

- ส่งโปรแกรมที่ใช้งานฟังก์ชันที่ต้องการทดสอบไปยังคอม86
- สั่งให้คอม86 เรียกโปรแกรมนั้นมาทำงาน แล้วสังเกตการทำงานของโปรแกรม
- เปลี่ยนโปรแกรมทดสอบที่ใช้ทดสอบแล้วทำการทดลองใหม่

ผลการทดลอง

คอม86 สามารถเรียกใช้งานฟังก์ชันของระบบปฏิบัติการไมโครซีรุ่นที่2ได้ ซึ่งฟังก์ชั่นเหล่านี้สามารถนำมาประยุกต์ให้ในการทำชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ได้ เช่นฟังก์ชันการส่งข้อมูลให้กันของทาสก์ต่างๆ ฟังค์ชันการนับค่าเวลา เป็นต้น




*รูปที่ 5-5 หน้าจอการเรียกใช้ฟังก์ชั่นของไมโครซีโอเอสบนคอม86*

จากภาพเป็นการเรียกใช้ฟังก์ชั่น OSMboxPost และ OSMboxPend ในการส่งผ่านข้อมูลระหว่างทาสก์ของโปรแกรมที่ทำงานอยู่บนคอม86

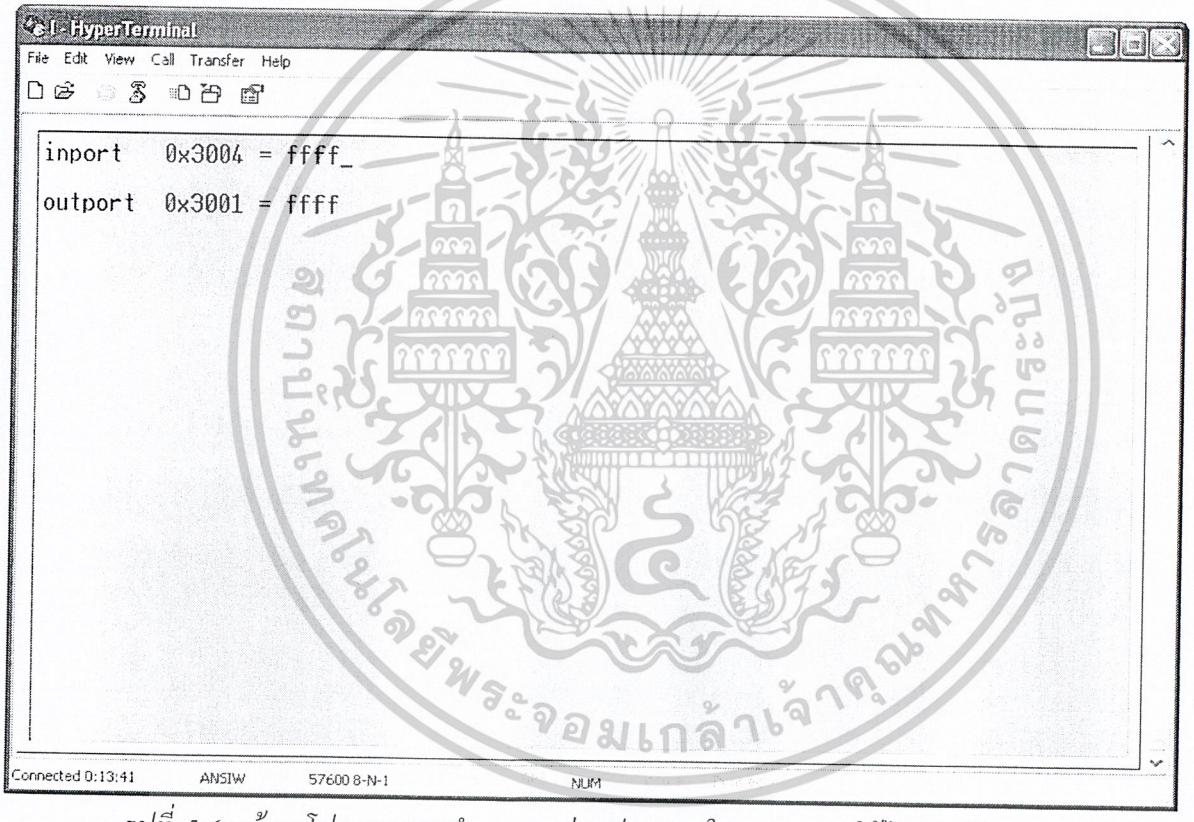
**5.2.2 ทำการทดสอบการทำงานฟังก์ชันต่าง ๆ ของไมโครซีโอเอส2 ร่วมกับส่วนประกอบภายในของคอม86**

วิธีการทดลอง

- ทำการติดตั้งอุปกรณ์ทดสอบ เช่นอุกรณ์ทดสอบไอซ่าบัส กับคอม86
- ส่งโปรแกรมที่ใช้ทดสอบอุปกรณ์ที่มีการใช้งานไมโครซีโอเอส2
- สั่งให้คอม86 ให้เรียกโปรแกรมนั้นมาทำงาน แล้วสังเกตการทำงานของโปรแกรม
- ปรับเปลี่ยนโปรแกรมที่นำมาทดสอบ และปรับเปลี่ยนอุปกรณ์ที่ใช้ทดสอบ แล้วทำการทดลองใหม่

ผลการทดลอง

ฟังก์ชั่นของไมโครซีโอเอส2 นั้นสามารถนำมาประยุกต์ใช้งานกับการทำงานการสั่งงานอุปกรณ์จากคอม86ได้ และสามารถนำประยุกต์ใช้งานกับการทำงานแบบใช้การเรียกฟังก์ชันจากแฟ้มข้อมูลแบบเฮดเดอร์ได้ โดยการปรับแต่งตัวคอมไพล์ของระบบปฏิบัติการไมโครซีรุ่นที่ 2บ้างเล็กน้อย




*รูปที่ 5-6 หน้าจอโปรแกรมการทำงานของส่วนประกอบในคอม86ภายใต้ไมโครซีโอเอส2*

จากภาพเป็นการนำคำสั่งที่ใช้งานกับคอม86 คือคำสั่งรับ-ส่งข้อมูลทางไอซ่าบัสมาใช้งานร่วมกับฟังก์ชั่นของไมโครซีโอเอส2 คือ การส่งค่าระหว่างทาสก์ผ่านทางเมล์บ็อก โดยคำสั่งรับและส่งข้อมูลจะทำงานอยู่ในทาสก์คนละทาสก์กัน และส่งค่าให้กันผ่านทางเมล์บ็อก

การทดลองนำระบบปฏิบัติการไมโครซีรุ่นที่ 2 มาใช้กับคอม86 ทำให้คอม86 สามารถทำงานแบบมัลติทาสก์ได้ เมื่อจะนำมาประยุกต์ใช้ต้องทดสอบการทำงานแบบใช้การเรียกฟังก์ชันจากแฟ้มข้อมูลแบบเฮดเดอร์และการทำงานแบบคลาสจากแฟ้มข้อมูลแบบเฮดเดอร์ซึ่งจะต้องมีการปรับแต่งค่าให้กับตัวคอมไพล์เพื่อรองรับการทำงานเหล่านี้

**5.3 ทำการศึกษาและทดสอบนำโปรแกรมที่สร้างขึ้นมา ควบคุมอุปกรณ์ต่างๆที่จะนำมาสร้างเป็นหุ่นยนต์**

หลักจากการทดสอบโปรแกรมบนคอม86แล้ว ต้องทำการทดสอบอุปกรณ์ที่ใช้จริง โดยจากการที่โครงงานได้จัดทำชุดคำสั่งเพื่อให้ผู้ใช้งานสามารถเรียกใช้งาน ควบคุมและสั่งงานอุปกรณ์ที่ออกแบบตามโมดูลที่กำหนดได้ทันที ดังนั้นทางคณะผู้จัดทำโครงงานจึงได้ทำการทดสอบโมดูลแต่ละโมดูลการชุดคำสั่งที่เตรียมไว้เพื่อทดลองการเรียกใช้งานรวมถึงตรวจสอบปัญหาที่จะเกิดขึ้นตามมา

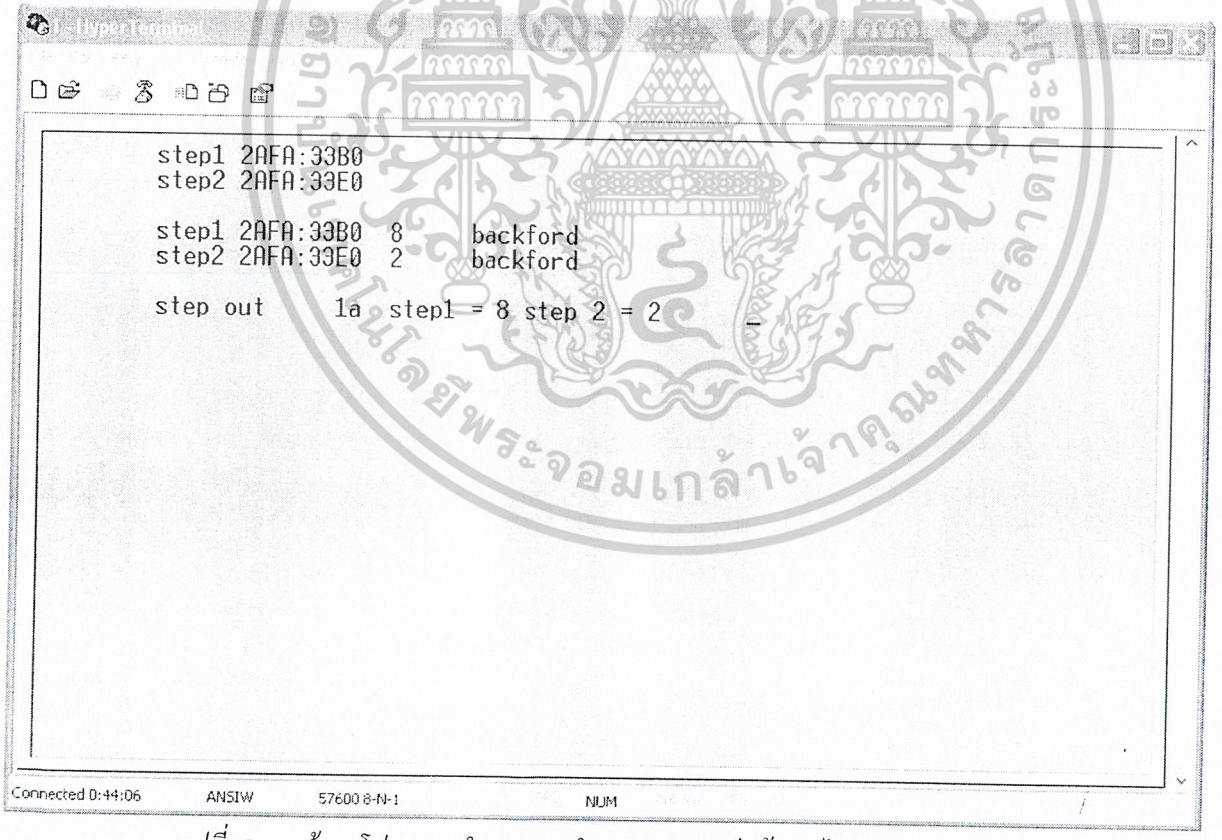
**5.3.1 ทดลองสร้างโปรแกรมต้นแบบจำลองการทำงานการควบคุมอุปกรณ์สำหรับหุ่นยนต์ที่ทำงานบนคอม86**

วิธีการทดลอง

- สร้างโปรแกรมต้นแบบ
- นำโปรแกรมต้นแบบส่งไปยังคอม86 แล้วสั่งให้คอม86ให้เรียกโปรแกรมนั้นมาทำงาน
- สังเกตผล แล้วนำมาปรับแก้โปรแกรม

ผลการทดลอง

จากการทดสอบโปรแกรมสามารถนำโปรแกรมที่สร้างขึ้นมาจำลองการทำงานในการสั่งงานอุปกรณ์ต่างของหุ่นยนต์และสามารถจำลองการทำงานของอุปกรณ์ที่เชื่อมต่อให้ทำงานร่วมกันได้




*รูปที่ 5-7 หน้าจอโปรแกรมจำลองการทำงานของการส่งข้อมูลไปควบคุมอุปกรณ์*

จากภาพเป็นหน้าจอแสดงผลโปรแกรมจำลองการสั่งงาน สเต็ปมอเตอร์ 2 ตัว และแสดงค่าที่โปรแกรมจะส่งออกไปทางไอซ่าบัสของคอม86

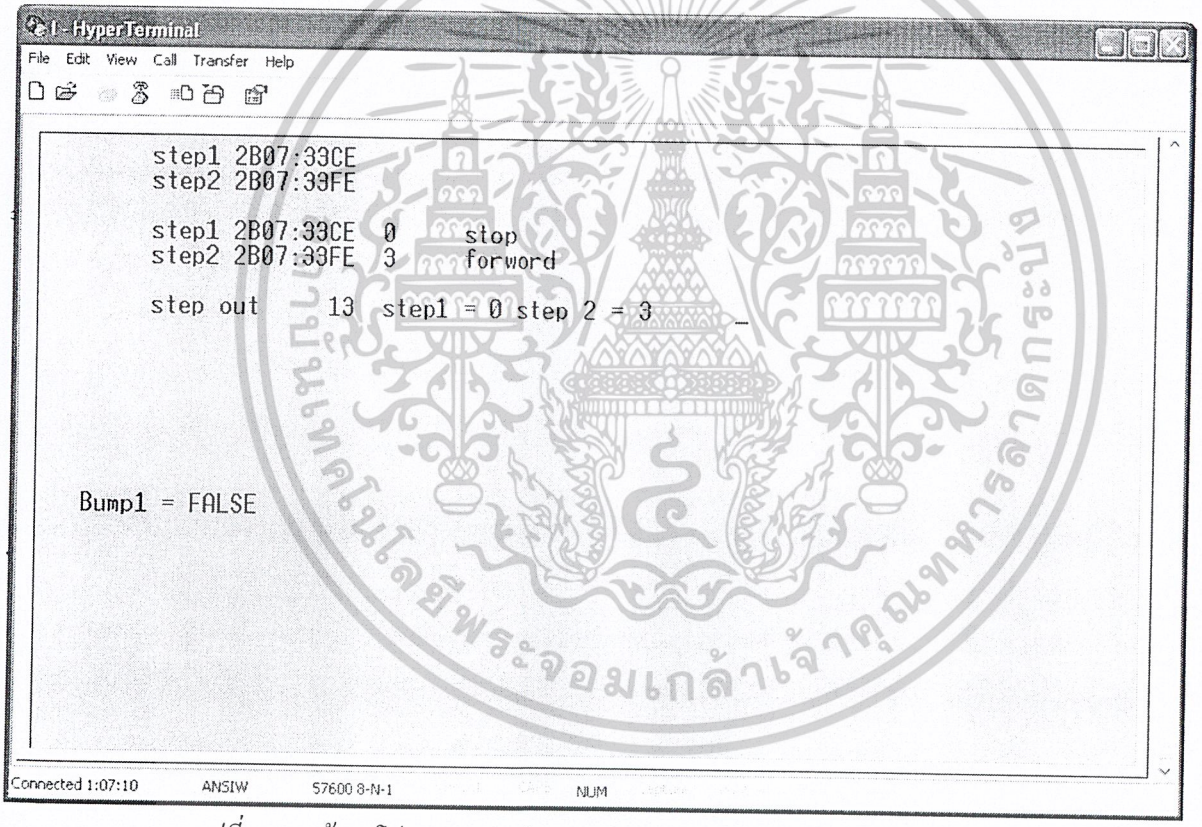
### 5.3.2 ทดลองนำโปรแกรมต้นแบบมาควบคุมอุปกรณ์ที่ใช้ทดสอบบนคอม86

วิธีการทดลอง

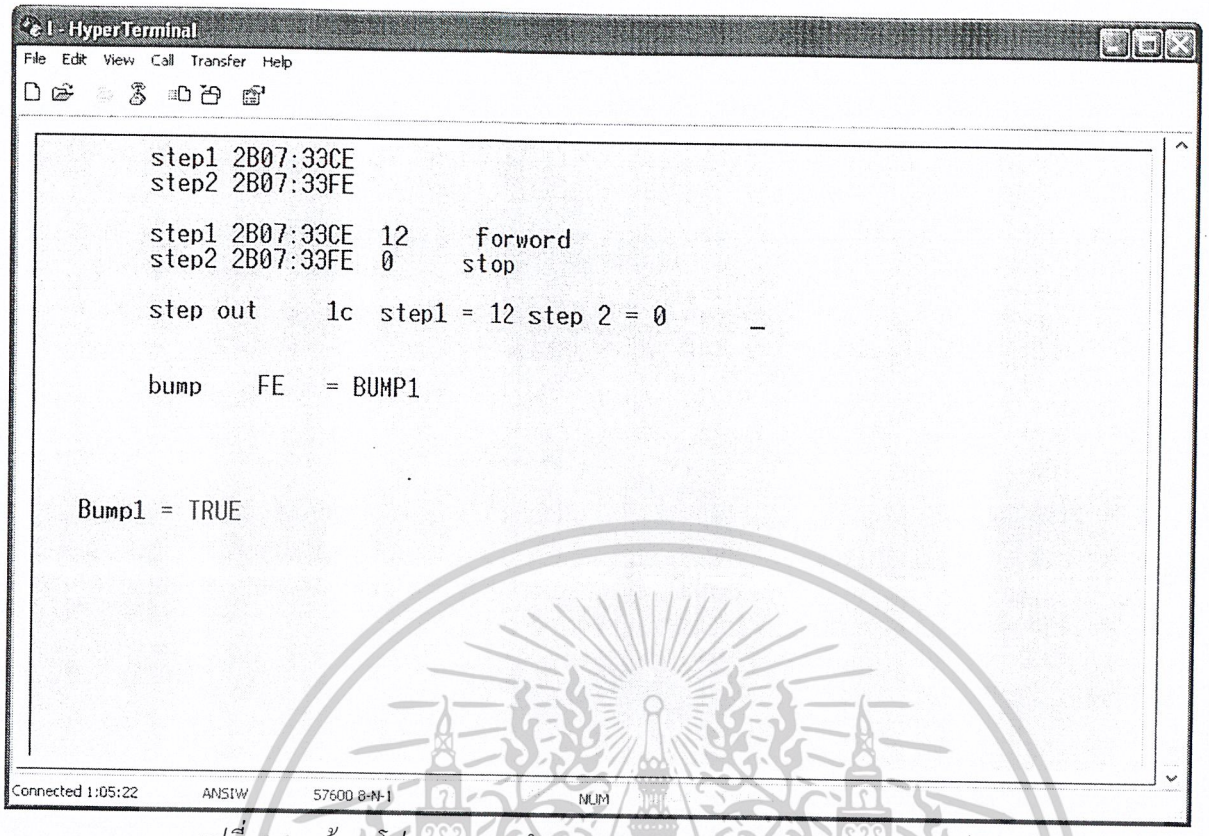
- ดัดแปลงโปรแกรมต้นแบบให้สามารถสั่งงานออกทาง พอร์ตไอซ่าบัสได้จริง
- ติดตั้งอุปกรณ์ทดสอบโดยจะเป็นอินพุตจากสวิตช์ และเอาท์พุตไปยังหลอดแสดงผล
- นำโปรแกรมต้นแบบส่งไปยังคอม86 แล้วสั่งให้คอม86ให้เรียกโปรแกรมนั้นมาทำงาน
- สังเกตผล แล้วนำมาปรับแก้โปรแกรม

ผลการทดลอง

จากการทดลองสามารถสั่งงานอุปกรณ์ทดสอบให้ทำงานได้ตามที่ต้องการ และสามารถทำงานร่วมกันได้แต่ยังมีปัญหาเรื่องการเชื่อมต่อคอม86 กับอุปกรณ์ทดสอบบ้างทำให้บ้างครั้งคอม86เกิดปัญหาขัดข้อง




*รูปที่ 5-8 หน้าจอโปรแกรมการทำงานของการทดสอบการควบคุมอุปกรณ์(1)*




*รูปที่ 5-9 หน้าจอโปรแกรมการทำงานของการทดสอบการควบคุมอุปกรณ์(2)*

จากภาพเป็นการแสดงผลของโปรแกรมทดสอบที่สมมติให้ บัมพ์สวิตช์สั่งงานสเต็ปมอเตอร์ โดยใช้สวิตช์แทนบัมพ์สวิตช์ และใช้หลอดแสดงผลแทนสเต็ปมอเตอร์ ซึ่งถ้ามีการกดบัมพ์สวิตช์ จะให้สเต็ปมอเตอร์ตัวที่ 1 ทำงาน (รูปที่5-9) แต่ถ้าไม่มีการกดบัมพ์สวิตช์ จะให้สเต็ปมอเตอร์ตัวที่ 2 ทำงาน (รูปที่5-8)

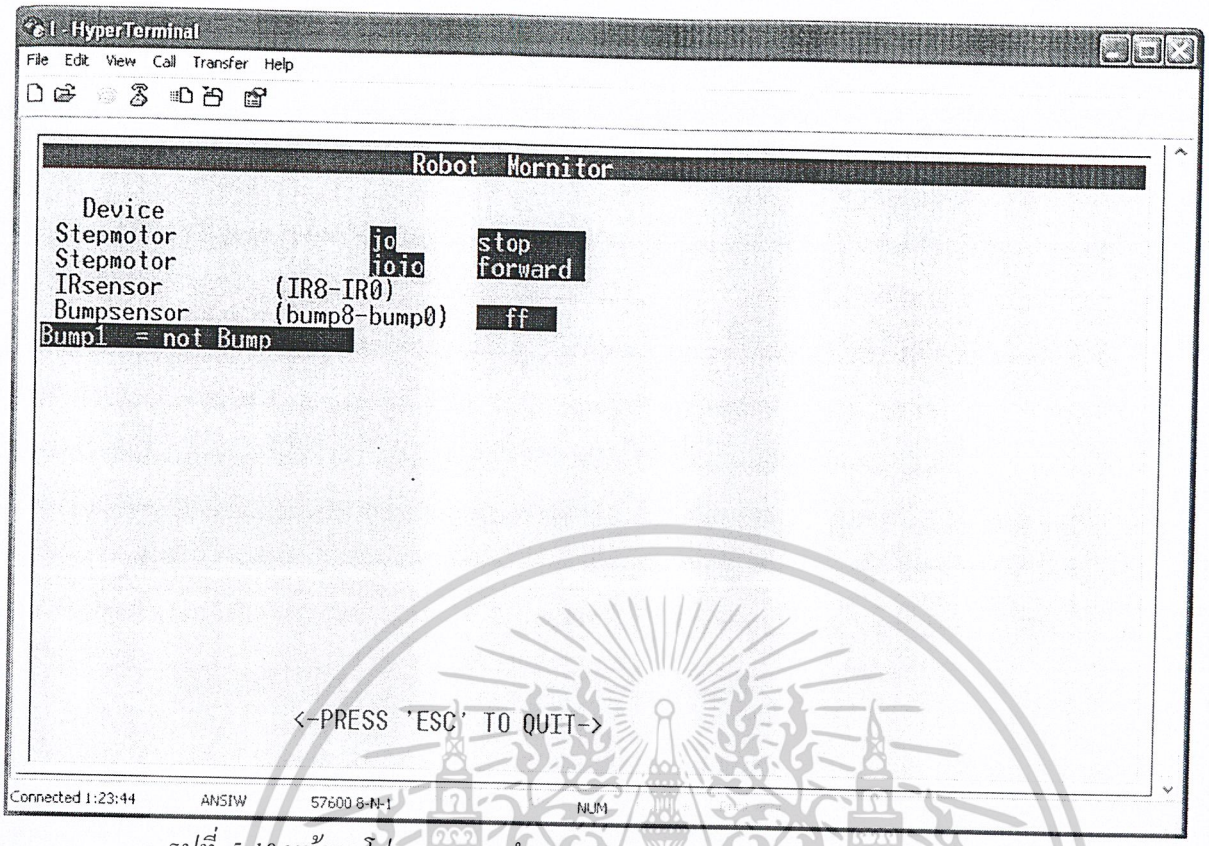
**5.3.3 ทดสอบนำโปรแกรมต้นแบบ มาควบคุมอุปกรณ์จริงที่จะนำไปใช้เป็นส่วนของหุ่นยนต์**

วิธีการทดลอง

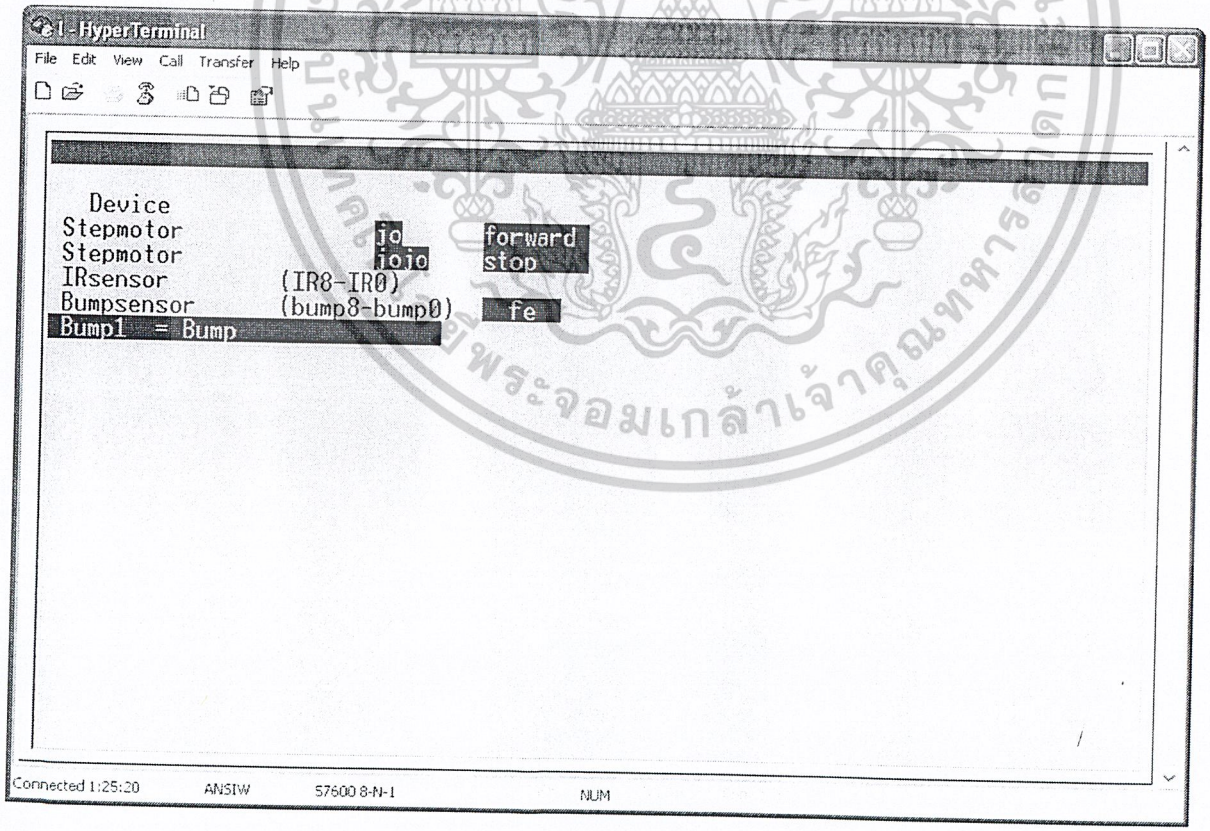
- สร้างโมดูลอุปกรณ์ส่วนต่างๆของหุ่นยนต์
- ดัดแปลงการทำงานของโปรแกรมต้นแบบให้สามารถทำงานร่วมกับหุ่นยนต์ได้
- ติดตั้งอุปกรณ์ทดสอบโดยจะเป็นอินพุตจากสวิตช์ และเอาท์พุตไปยังหลอดแสดงผล
- นำโปรแกรมต้นแบบส่งไปยังคอม86 แล้วสั่งให้คอม86 ให้เรียกโปรแกรมนั้นมาทำงาน
- สังเกตผล แล้วนำมาปรับแก้โปรแกรม

ผลการทดลอง

จากการทดลองสามารถสั่งงานโมดูลอุปกรณ์ให้ทำงานได้ตามที่ต้องการ และสามารถทำงานร่วมกันได้แต่ยังมีปัญหาเรื่องการเชื่อมต่อคอม86 กับโมดูลอุปกรณ์บ้างทำให้บ้างครั้งคอม86 เกิดปัญหาขัดข้อง




*รูปที่ 5-10 หน้าจอโปรแกรมการทำงานของการทดสอบการควบคุมอุปกรณ์จริง(1)*




*รูปที่ 5-11 หน้าจอโปรแกรมการทำงานของการทดสอบการควบคุมอุปกรณ์จริง(2)*

จากภาพเป็นการแสดงผลของโปรแกรมทดสอบที่ให้ บัมพ์สวิตช์สั่งงานสเต็ปมอเตอร์ ซึ่งถ้ามีการกดบัมพ์สวิตช์ จะให้สเต็ปมอเตอร์ตัวที่ 1 ทำงาน (รูปที่5-11) แต่ถ้าไม่มีการกดบัมพ์สวิตช์ จะให้สเต็ปมอเตอร์ตัวที่ 2 ทำงาน (รูปที่5-10)

การทดลองสร้างโปรแกรมมาควบคุมโมดูลอุปกรณ์สามารถสั่งให้โมดูลอุปกรณ์ทำงานได้ แต่มีปัญหาที่โมดูลของอุปกรณ์อาจมีความผิดพลาดทางฮาร์ดแวร์และส่งผลกระทบต่อการทำงานของคอม86 ทำให้ คอม86ไม่สามารถทำต่อได้ ซึ่งอาจต้องปรับแต่งโมดูลอุปกรณ์ให้ให้ทำงานให้เสถียรขึ้น

ยกตัวอย่างเช่น จากการทดสอบบอร์ดขยายขาแอดเดสของไอซ่าบัสนั้น บางครั้งบอร์ดคอม86 ไม่สามารถเริ่มการทำงานได้ อาจจะเกิดจากขาสัญญาณขัดข้องและจากการแก้ไขนั้นบางครั้งพบว่าเกิดจากเส้นแอดเดส ไม่เข้ากับบอร์ด โดยทั่วไปแล้วสรุปได้ว่า การนำโมดูลที่ออกแบบไปใช้งานนั้น สามารถใช้งานได้และสามารถเรียกใช้งานชุดคำสั่งที่เตรียมไว้ได้อย่างถูกต้อง แต่ในบางครั้งยังไม่เสถียรเท่าที่ควร ซึ่งจะทำการแก้ไขและปรับแต่งต่อไป

# บทที่ 6
# การประยุกต์นำโปรแกรมไปใช้งาน

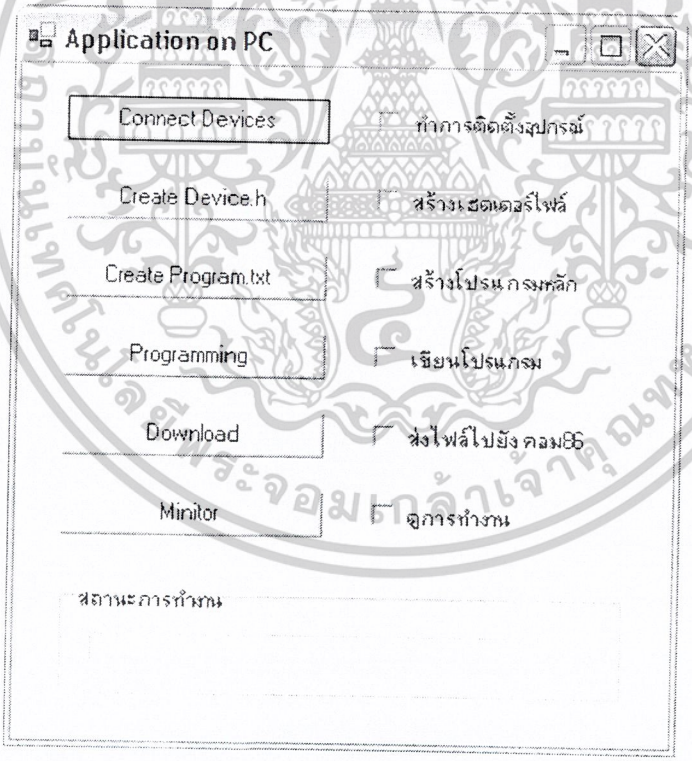
ลักษณะการใช้งาน โปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์และ ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ แบ่งออกได้ 2 ส่วนหลัก ๆ ดังนี้

- โปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์
- โมดูลของอุปกรณ์ที่เหมาะสมกับการเรียกใช้งานชุดคำสั่ง

โดยทั้ง 2 ส่วนนั้นจัดได้ว่าเป็นส่วนที่จะเป็นตัวช่วย ทำให้ผู้ใช้งานสามารถเรียกใช้งาน ชุดคำสั่งที่เตรียมไว้ได้อย่างถูกต้องและสะดวก โดยโปรแกรมช่วยเหลือนั้นจะมีการติดตั้งอยู่ทั้งบนคอมพิวเตอร์ และบนบอร์ดคอม86 โดยโปรแกรมที่รันอยู่บนคอม86นั้น จะมีการสร้างทาสก์ในส่วนที่ทำการส่งค่าข้อมูลการทำงานมายังคอมพิวเตอร์เพื่อทำการแสดงผลการทำงาน และส่วนที่ทำการรอรับค่าจากคอมพิวเตอร์ เพื่อรองรับการควบคุมการผู้ใช้งานโดยตรง

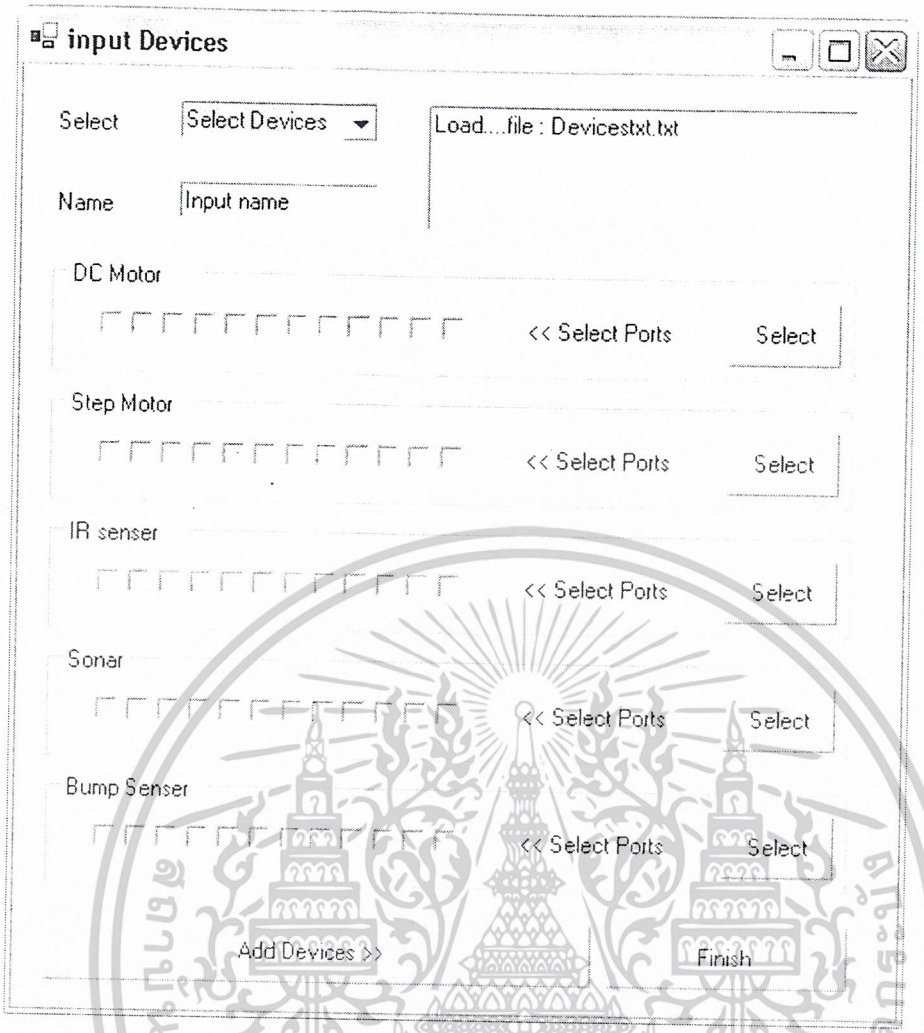
ซึ่งการทำงานหลัก ๆ ของแอพพลิเคชั่นบนคอมพิวเตอร์มีดังต่อไปนี้

## 6.1 การใช้โปรแกรมหลักบนคอมพิวเตอร์




*รูปที่ 6-1 หน้าหลักของโปรแกรมช่วยเหลือการใช้งานหุ่นยนต์บนคอมพิวเตอร์*

โปรแกรมที่สร้างขึ้นจะมีส่วนประกอบหลักอยู่ 6 ส่วน ดังภาพ โดยการใช้งานคือผู้ใช้งานต้องเลือกลำดับการทำงานตามขั้นตอนจากบนลงล่าง โดยเริ่มติดตั้งอุปกรณ์ โดยการกำหนดชนิดของอุปกรณ์และตั้งค่าเพื่อเรียกใช้งานชุดคำสั่ง โดยการเลือก Connect Devices

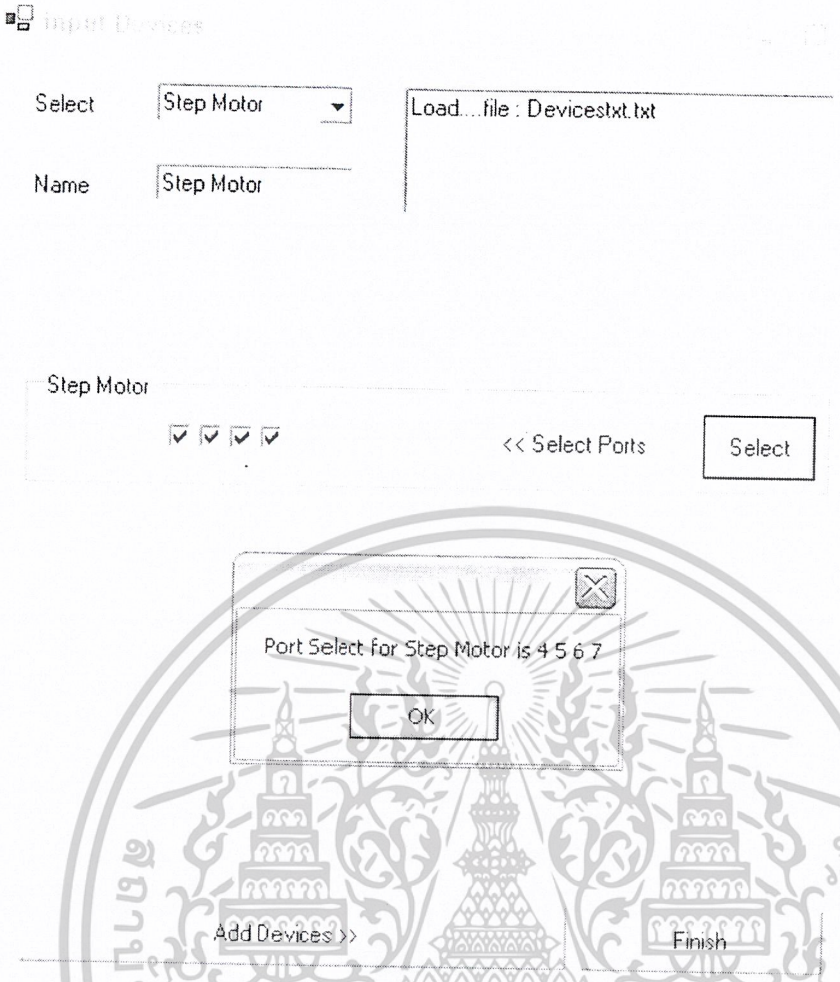



*รูปที่ 6-2 โปรแกรมช่วยเหลือการติดตั้งอุปกรณ์*

โดยผู้ใช้งานต้องกำหนดชนิดของอุปกรณ์ และชื่อของอุปกรณ์ที่จะเรียกใช้ในโปรแกรม ( โปรแกรมจะทำการสร้างออปเจค ของอุปกรณ์ขึ้นในโปรแกรมเพื่อให้ผู้ใช้งานสามารถเรียกใช้ชุดคำสั่งผ่านทางออปเจคนั้น ) โดยหลังจากกำหนดชื่อแล้วขึ้นตอนต่อไปคือการเลือกพอร์ตที่ต้องการ โดยในส่วนนี้โปรแกรมจะทำการเช็คค่าพอร์ตที่ติดกับคอม86 ให้โดยอัตโนมัติ

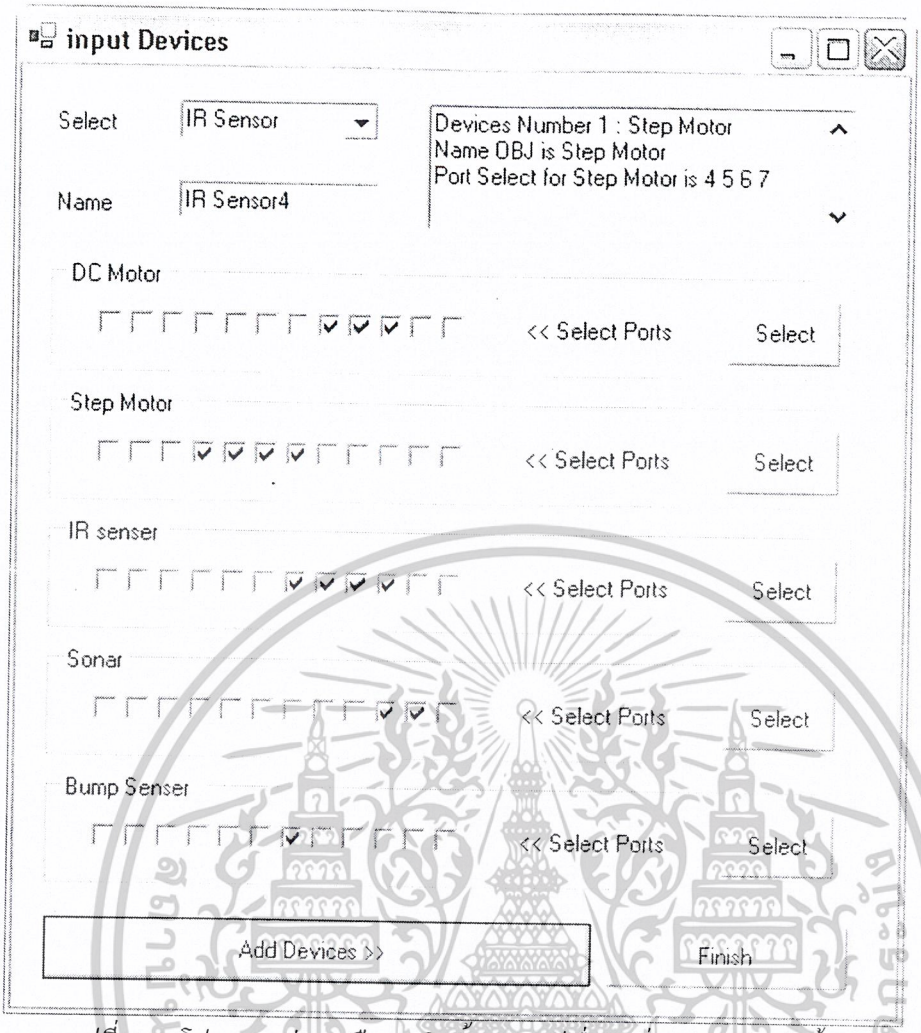
โดยขั้นตอนการใช้งานในส่วนของการติดตั้งอุปกรณ์มีดังต่อไปนี้

- เลือกชนิดของอุปกรณ์
- ตั้งชื่อของออปเจคของอุปกรณ์ที่นำมาติดตั้ง
- เลือกพอร์ตทางลอจิกคอลของอุปกรณ์ โดยการเลือกพอร์ตจากโปรแกรมหลัก
- เช็คความถูกต้องของพอร์ตที่กำหนดและค่าต่าง ๆ ที่เลือกไว้
- ทำการเก็บค่าที่ติดตั้งอุปกรณ์ เพื่อเตรียมไปทำการสร้างไลบารี่สำหรับเรียกใช้ต่อไป



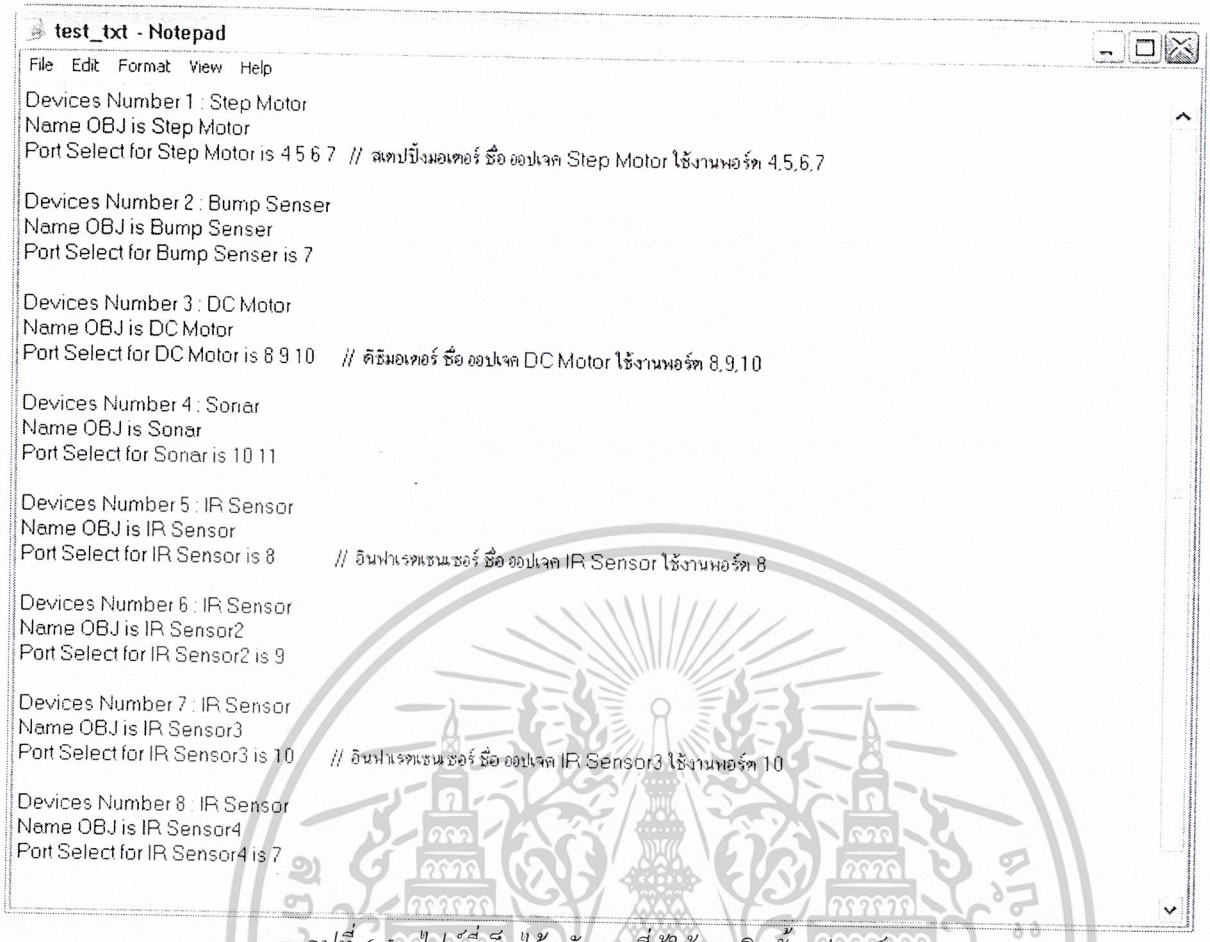

*รูปที่ 6-3 โปรแกรมช่วยเหลือการติดตั้งอุปกรณ์หลังการกำหนดพอร์ตแล้ว*

หลังจากทำการกำหนดค่าของอุปกรณ์ตามขั้นตอนอย่างถูกต้องแล้ว ขั้นตอนต่อไปคือการเก็บค่าอุปกรณ์ไว้ยังไฟล์ที่จะเรียกใช้เพื่อทำการสร้างไฟล์ Devices.h เพื่อใช้ในการเขียนโปรแกรมโดยการเลือก Add Devices



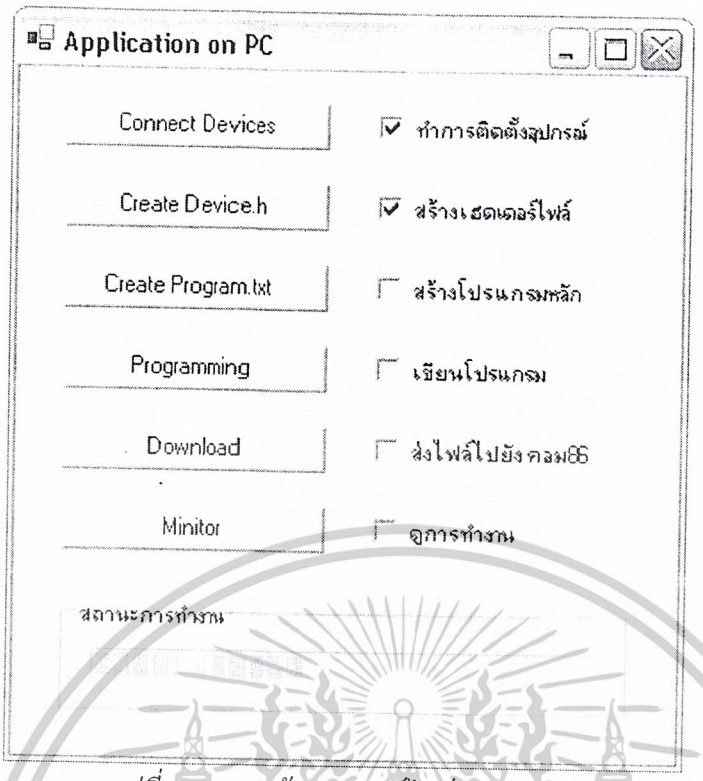

*รูปที่ 6-4 โปรแกรมช่วยเหลือการติดตั้งอุปกรณ์เมื่อติดตั้งอุปกรณ์เสร็จสิ้น*

หลังการทำการเลือกติดตั้งอุปกรณ์ครบถ้วนตามที่ต้องการแล้ว โปรแกรมจะทำการเก็บข้อมูลที่ผู้ใช้งานต้องการเก็บไว้ยังเทกไฟล์ เพื่อเรียกมาคำนวณและสร้างไฟล์สำหรับ include ไปในโปรแกรมหลักเพื่อเรียกใช้งานชุดคำสั่งและสร้างออปเจคของคลาสอุปกรณ์นั้น ๆ



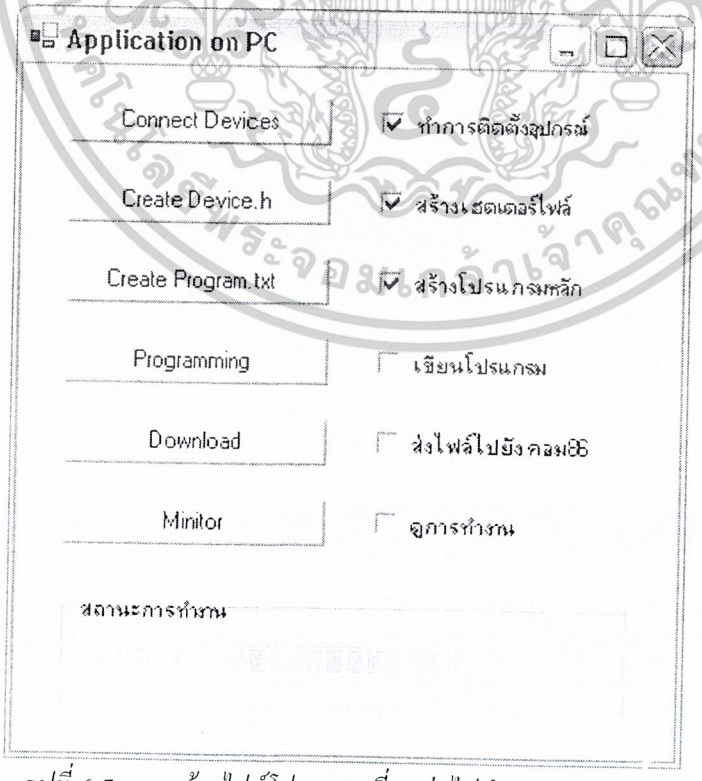

*รูปที่ 6-5 ไฟล์ที่เก็บไว้หลังจากที่ผู้ใช้งานติดตั้งอุปกรณ์*

ขั้นตอนต่อไปหลังการติดตั้งอุปกรณ์คือการสร้างเฮดเดอร์ไฟล์ สำหรับเรียกใช้งานฟังก์ชั่น ควบคุมอุปกรณ์ โดยขั้นตอนนี้โปรแกรมหลักจะทำการเขียน โครงสร้างของ ทากส์และประการค่าต่าง ๆ ให้กับโปรแกรมที่ผู้ใช้งาน จะต้องเขียนให้สามารถทำงานในแบบบัลติทาสก์ ให้โดยอัตโนมัติ



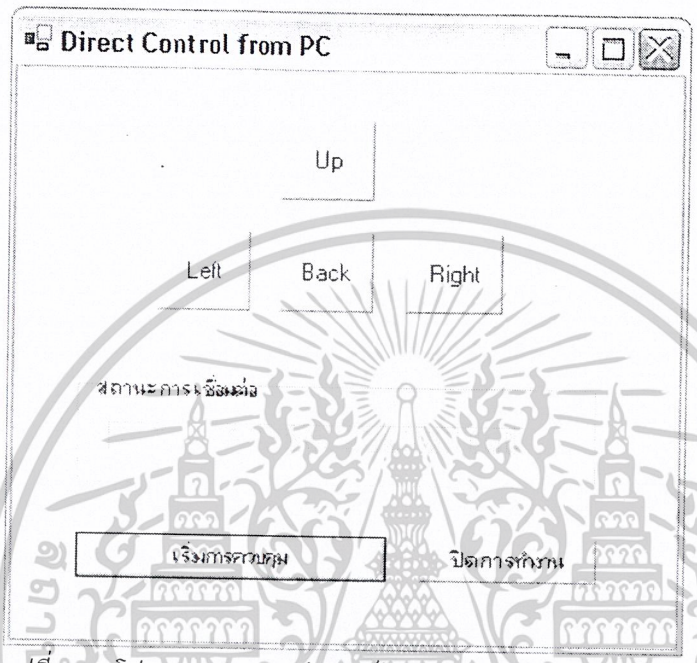

*รูปที่ 6-6 การสร้างเฮดเดอร์ไฟล์ Device.h*

หลังจากการสร้างเฮดเดอร์ไฟล์ ต่อไปจะทำการวางโครงสร้างโปรแกรมที่จะส่งไปทำงานบนคอม86 โดยโปรแกรม จะสร้างเทกไฟล์ ที่จะรองรับการใช้งานระบบปฏิบัติการไมโครซี รวมถึงแทรกโปรแกรมที่จะใช้สำหรับดูผลการทำงานและโปรแกรมสั่งงานโดยตรงไว้อีกด้วย เพื่อให้ผู้ใช้งานเรียกไปใช้ในการเขียนโปรแกรมต่อไป



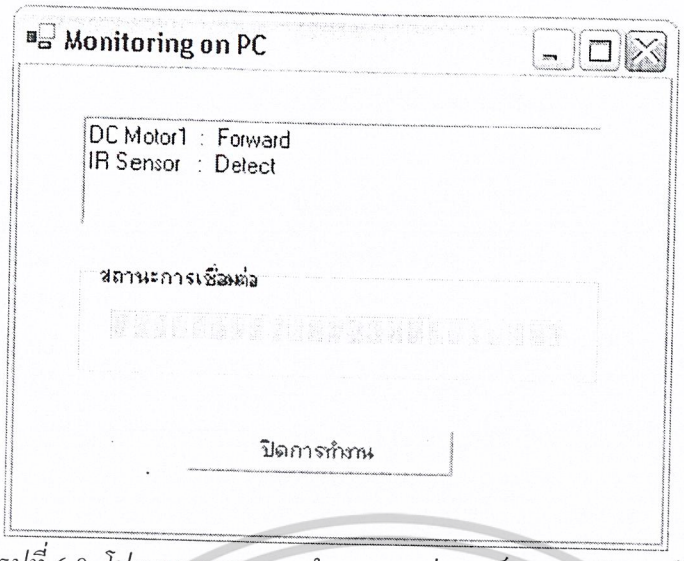

*รูปที่ 6-7 การสร้างไฟล์โปรแกรมที่จะส่งไปทำงานบนคอม86*

ในส่วนทำการเขียนโปรแกรมนั้น ผู้ใช้งานจะสามารถเลือกเขียนโปรแกรมบนโปรแกรมหลัก หรือทำการเปิดโปรแกรมเขียนภาษาซี อื่น ๆ มาใช้งานได้โดยโครงสร้างโปรแกรมที่สร้างขึ้นมานั้นจะอยู่บนไดร์พซี ชื่อ Program.txt หลังจากนั้นก็ทำการคอมไพล์ด้วย โปรแกรมบอแลนซี เพราะคอม86 นั้นเป็นไมโครคอนโทลเลอร์ขนาด 16 บิต ต้องระวังในข้อนี้ด้วย




*รูปที่ 6-8 โปรแกรมควบคุมหุ่นยนต์โดยตรงผ่านทางคอมพิวเตอร์*

ในส่วนโปรแกรมควบคุมโดยตรงนั้น มีหลักการคือต้องการออกแบบ ให้ผู้ใช้งานสามารถควบคุมหุ่นยนต์ผ่านทางพอร์ตอนุกรม โดยโปรแกรมที่ทำการส่งไปทำงานบนคอม86 จริง ๆ นั้นจะช่วยทำให้คอม86สนับสนุนการส่งค่าผ่านคีย์บอร์ดระหว่างทำงานโปรแกรมอื่นได้ ทำให้เมื่อมีการส่งค่าจากคอมพิวเตอร์แล้ว คอม86สามารถรับค่าไปประมวลผลได้ ส่วนการควบคุมนั้นผู้ใช้งานสามารถเลือกออกแบบโปรแกรมได้เองตามความต้องการ โดยในที่นี้โปรแกรมควบคุมโดยตรงที่สร้างขึ้นเป็นเพียงตัวอย่าง เท่านั้น




*รูปที่ 6-9 โปรแกรมแสดงการทำงานของหุ่นยนต์บนคอมพิวเตอร์*

ส่วนโปรแกรมแสดงผลการทำงานของหุ่นยนต์ ในส่วนนี้ก็เช่นกันโปรแกรมที่แทรกไปยังคอม86 นั้นมีหน้าที่ทำการส่งค่าข้อมูลสถานะของอุปกรณ์ ( ออปเจค ) ที่ทำงานอยู่บนคอม86 มายังคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม โดยโปรแกรมที่ออกแบบนั้นเป็นเพียงตัวอย่างการใช้งานค่าที่รับมาแสดงผล อาจจะนำไปประยุกต์ใช้ในการจำลองการทำงานบนคอมพิวเตอร์ได้ตามความต้องการ

**6.2 โมดูลของอุปกรณ์ที่เหมาะสมกับการเรียกใช้งานชุดคำสั่ง**

ในส่วนของวงจรสำหรับใช้งานอุปกรณ์ก็นับว่าเป็นส่วนสำคัญ ในการที่จะเรียกใช้งาน ชุดคำสั่งที่เตรียมไว้โดยทางคณะผู้จัดทำได้ทำการออกแบบ วงจรพื้นฐานที่นิยมใช้ในการสร้างหุ่นยนต์เป็น โมดูลต้นแบบที่สามารถเรียกใช้งานชุดคำสั่งได้ทันที โดยที่ผู้ใช้งานสามารถออกแบบ แก้ไขชุดคำสั่งให้เหมาะสมกับการใช้งานเพิ่มเติมได้

โดยรูปแบบวงจรนั้นได้แนบลายวงจรไว้ในภาคผนวกแล้ว โดยการใช้งานนั้นสามารถใช้งานร่วมกับโปรแกรมช่วยเหลือการทำงานข้างต้นโดยการทำโมดูลของอุปกรณ์แล้วทำการเลือกติดตั้งอุปกรณ์ตามขั้นตอนที่ได้กำหนดไว้ โดยการแก้ไขไฟล์ชุดคำสั่งนั้น สามารถแก้ไขจากไฟล์ Device.txt ได้

# บทที่ 7
# สรุปและวิจารณ์

## 7.1 วิเคราะห์ผลจากการออกแบบและสร้างระบบ

ผลของการสร้างโปรแกรมช่วยเหลือสำหรับการใช้งานหุ่นยนต์และ ชุดคำสั่งสำหรับควบคุมและสั่งงานอุปกรณ์ สำหรับหุ่นยนต์ที่ใช้ชุดพัฒนาคอม86 ของโครงงานนี้ จากจุดประสงค์ของโครงงานนี้ที่ว่าด้วยการแสดงให้เห็นถึงการสร้างหุ่นยนต์โดยใช้โปรแกรมที่ออกแบบมานั้นสะดวกและง่ายกว่า รวมถึงการควบคุมอุปกรณ์ของหุ่นยนต์ โดยการเรียกใช้ชุดคำสั่งที่ทำขึ้นนั้นต้องช่วยให้การควบคุม ติดต่อและใช้งานหุ่นยนต์ทำได้สะดวกมากยิ่งขึ้น โดยจากจุดประสงค์ทำให้คณะผู้จัดทำออกแบบการทำงานของแอปพลิเคชั่นให้สามารถช่วยเหลือผู้ใช้งานในการ ติดตั้งอุปกรณ์ ช่วยเหลือในการวางโครงสร้างของโปรแกรม ส่งข้อมูลไปประมวลผล แสดงผลการทำงาน และส่วนของการควบคุมหุ่นยนต์ และได้ออกแบบโครงสร้างการทำงานของโปรแกรมบนหุ่นยนต์ให้สามารถทำงานแบบมัลติโปรแกรมมิ่ง โดยจะมีการออกแบบชุดคำสั่งให้ผู้ใช้งานสามารถนำไปควบคุมอุปกรณ์จากการเรียกใช้งานไลบารี่ที่จัดทำขึ้นรวมทั้งมีการเรียกโปรแกรมที่ช่วยในการวางโครงสร้างโปรแกรมที่จะนำไปทำงานบนหุ่นยนต์ ให้สนับสนุนการทำงานแบบมัลติทาสก์ เพราะจะต้องมีการส่งโปรแกรมในการรับส่งข้อมูล ระหว่างคอมพิวเตอร์และคอม86ที่อยู่บนหุ่นยนต์

ในส่วนของหุ่นยนต์นั้นทางคณะผู้จัดทำต้องทำการออกวงจรเพื่อทำการขยายสัญญาณแอดเดส จากคอม86 เพื่อนำมาใช้ในการติดตั้งโมดูลอุปกรณ์ โดยโมดูลของอุปกรณ์ในที่นี้ได้ทำการออกแบบเพื่อให้สามารถเรียกใช้ชุดคำสั่งควบคุมจากโปรแกรมทำเตรียมไว้ โดยโมดูลที่เลือกมาเป็นอุปกรณ์พื้นฐานนั้นก็นับว่าเป็นอุปกรณ์ที่นิยมใช้งานในการสร้างหุ่นยนต์ โดยจากการที่ทดสอบโมดูลด้วยชุดคำสั่งที่เตรียมไว้สามารถทำงานได้ด้วยดี

จากการทดสอบระบบโดยการทดลองนำโปรแกรมไปใช้กับหุ่นยนต์ต้นแบบนั้นพบว่า สามารถใช้งานได้ในระดับหนึ่ง โดยการช่วยเหลือในการติดตั้งระหว่างโมดูลกับบอร์ดขยายสัญญาณที่เป็นตัวช่วยในการเชื่อมต่ออุปกรณ์กับคอม86นั้น อาจมีปัญหาในการใช้งานบ้าง การควบคุมโปรแกรมที่ทำงานบนหุ่นยนต์โดยตรง ผ่านทางคอมพิวเตอร์และโปรแกรมแสดงผลการทำงานของหุ่นยนต์สามารถทำงานได้ดี

## 7.2 ปัญหาและอุปสรรค

สามารถสรุปปัญหาที่เกิดจากการทำโครงงานได้ ดังนี้

- การสร้างบอร์ดขยายขาแอดเดสจากไอซ่าบัสของคอม86 ยังพบปัญหาในการใช้งานไอซ่าบัสอยู่พบสมควร โดยปัญหาที่พบบ่อยในการทำงานคือเมื่อทำการเชื่อมต่อบอร์ดดีโค๊ดแอดเดสการไอซ่าบัสของคอม86แล้ว ทำให้บอร์โคอม86ไม่สามารถทำงานได้ จากการศึกษาพบว่าเกิดจากโครงสร้างการทำงานของไอซ่าบัสของคอม86 ต่างจากการทำงานของไอซ่าบัสของคอมพิวเตอร์พอสมควร
- การเชื่อมต่อวงจรร่วมกันระหว่างบอร์ดขับเคลื่อนมอเตอร์ กับบอร์ดขยายแอดเดส พบว่าการเชื่อมต่อวงจรที่ใช้กระแสมากนั้นสามารถทำการรบกวนการทำงานของคอม86 ได้ ต้องทำการป้องกันปัญหาของ

กระแสจากบอร์ดมอเตอร์ที่อาจจะทำให้บอร์ดคอม86 ทำงานผิดพลาดได้ อาจจะทำได้โดยการแยกแหล่งจ่ายไฟของทั้งสองบอร์ดออกจากกันแต่ต้องระวังเรื่องกราวด์ของบอร์ดที่ต้องเชื่อมกันด้วย

- การสร้างหุ่นยนต์และการทดสอบโมดูล อุปกรณ์แต่ละชิ้นนั้นทีราคาแพง เนื่องจากงบประมาณที่จำกัด ทำให้ไม่สามารถทดสอบโมดูลของอุปกรณ์ได้มากนัก รวมถึงการสร้างต้นแบบของหุ่นยนต์จึงจัดทำเพียงหุ่นยนต์ที่สามารถแสดงการเรียกใช้งานของคำสั่งพื้นฐานที่ครบถ้วนเท่านั้น
- ในส่วนของโปรแกรมที่จะช่วยผู้ใช้งานเขียนโปรแกรม ทำได้เพียงวางโครงสร้างและช่วยเหลือในส่วนของการกำหนดพอร์ตที่เชื่อมต่อจริง โดยในส่วนของการคอมไพล์ยังต้องเรียกใช้งานจากคอมไพล์เลอร์ภายนอก และในส่วนของการส่งข้อมูลก็เรียกใช้งานผ่านโปรแกรมไฮเปอร์เทอร์มินอล
- การติดตั้งอุปกรณ์จากโมดูลที่ออกแบบนั้นยังเป็นรู)แบบที่ค่อนข้างตายตัว ขาดความยืดหยุ่นในการใช้งานการเพิ่มเติมขาสัญญาณนั้นจะต้องทำการเปลี่ยนแปลงข้อมูลหลายส่วน ตั้งแต่โปรแกรมตั้งขาเชื่อมต่อบนคอมพิวเตอร์ การแก้ไขการใช้งานชุดคำสั่ง การแก้โปรแกรมแสดงผลการทำงาน ซึ่งไม่สะดวกนัก

## 7.3 ข้อดีและข้อเสียของระบบ

**ข้อดี**

- ระบบนี้ออกแบบด้วยออบเจ็กต์โอเรียนเต็ด ทำให้สามารถนำส่วนต่างมาใช้ใหม่ รวมถึงการแก้ไขโปรแกรมและสามารถทำการพัฒนาต่อได้ง่าย และจากการวางโครงสร้างออกเป็นคลาสต่าง ๆ ทำให้ได้ระบบที่มีประสิทธิภาพในการทำงานและค่อนข้างเหมาะสมในการใช้งาน
- การเชื่อมต่ออุปกรณ์ผ่านทางโปรแกรมช่วยเหลือทำให้การชื่องต่ออุปกรณ์ทำได้โดยง่าย
- การเพิ่มประสิทธิภาพการทำงานของโปรแกรมของหุ่นยนต์ให้ทำงานแบบมัลติทาสก์นั้น ทำให้การเรียกใช้งานอุปกรณ์ รวมถึงการสร้างโปรแกรมเพิ่มเติม การประมวลผลดีขึ้นอย่างมาก ซึ่งจากการที่วางโครงสร้างโปรแกรมไว้ให้ก่อนเป็นการสะดวกต่อการเขียนโปรแกรมอย่างมาก
- การเพิ่มโปรแกรมในส่วน รับส่งข้อมูลไปทำงานบนหุ่นยนต์โดยที่แปรแกรมที่ทำงานอยู่นั้นทำงานอยู่ต่างทาสก์กัน และทำการส่งผ่านข้อมูลภายในด้วยการส่งเมลบล็อกนั้นทำให้การทำงานเสถียรมากขึ้น
- ผู้ใช้งานไม่จำเป็นต้องเข้าใจในส่วนของการใช้งานอุปกรณ์มากนักเนื่องจากได้จัดทำ ส่วนโมดูลของวงจรสำหรับเรียกใช้งานอุปกรณ์ไว้ให้แล้ว ผู้ใช้งานสามารถนำโมดูลที่เตรียมไว้ไปใช้งานได้เลย

**ข้อเสีย**

- การทำบอร์ดขยายขาแอดเดสด้วยไอซีมีความแสถียรน้อยกว่าการใช้ ชิพที่โปรแกรมได้ (PAL)
- โมดูลที่ทำการออกแบบมาสนับสนุนการใช้งานอุปกรณ์พื้นฐานเท่านั้น
- การเพิ่มเติมและพัฒนาโปรแกรมที่จัดทำขึ้นอาศัยความรู้พอสมควร การแก้ไขและดัดแปลงค่าต่าง ๆ ยังทำได้ยากอยู่ รวมถึงการแก้ไขชุดคำสั่งที่เตรียมไว้ใช้งานนั้นต้องทำการแก้ไขในระดับโค๊ดของโปรแกรม ซึ่งถ้าต้องทำการปรับเปลี่ยนผู้ใช้งานอาจเกิดปัญหาและยุ่งยากพอสมควร

## 7.4 แนวทางในการพัฒนาต่อไป

- สามารถนำไปพัฒนาเพิ่มเติมในส่วนของแอปพลิเคชั่นที่ยังขาดอยู่ เช่น โปรแกรมที่ทำการจำลองการทำงานโดยการประมวลผลจากข้อมูลที่รับมา การควบคุมอุปกรณ์เพิ่มเติม การเพิ่มโปรแกรมในส่วนของการคอมไพล์โปรแกรม โปรแกรมส่งผ่านไฟล์จากคอมพิวเตอร์ไปยังคอม86 (ไฮเปอร์เทอร์มินอล) รวมถึงโปรแกรมที่ช่วยเหลือในการแก้ไขชุดคำสั่งที่ทำให้ผู้ใช้งานสามารถแก้ไข หรือเพิ่มเติมชุดคำสั่งจากโมดูลที่ออกแบบขึ้นเองโดยไม่ต้องแก้ไขจากโค๊ดโดยตรง
- สามารถพัฒนาการสื่อสารระหว่างคอมพิวเตอร์กับหุ่นยนต์เป็นแบบไร้สาย อาจทำได้โดยการติดตั้ง บลูทูส (Bluetooth) เพิ่มเติม
- สามารถพัฒนาอุปกรณ์ที่ติดตั้งกับหุ่นยนต์ให้มีมากขึ้น รวมถึงการเพิ่มความสามารถในการใช้งานอุปกรณ์ให้หลากหลายและมีความฉลาดและนำไปประยุกต์ใช้งานให้เหมาะสมและสะดวกมากขึ้น ซึ่งจะเป็นการดีต่อผู้ใช้งานอย่างมากถ้าสามารถออกแบบโมดูลของอุปกรณ์ที่ต้องการใช้งานได้เอง

## 7.5 สรุป

ถึงแม้ว่าเทคโนโลยีการใช้งานหุ่นยนต์ นั้นจะเป็นเทคโนโลยีที่มีรูปแบบการใช้งานที่หลากหลาย และได้รับความสนใจมากขึ้นในปัจจุบัน แต่หากดูในรายละเอียดของพื้นฐานการใช้งานแล้ว จะพบว่าในการใช้งานจริงนั้น อุปกรณ์ของหุ่นยนต์ที่ใช้นั้นส่วนใหญ่แล้วเป็นอุปกรณ์พื้นฐานทั้งสิ้น แล้วแต่ความสามารถในการออกแบบและการนำไปประยุกต์ใช้งาน การนำอุปกรณ์พื้นฐานต่าง ๆ มาใช้งานร่วมกันให้เหมาะสม และใช้งานให้เกิดประโยชน์เหมาะสมกับความต้องการนับว่าเป็นหลักการสำคัญในการออกแบบหุ่นยนต์ เพราะการสร้างหุ่นยนต์ยังมีส่วนประกอบสำคัญอื่น ๆ ทางด้านฮาร์ดแวร์อีก มากอย่างเช่นความเสถียรของวงจรที่ใช้งาน ปัญหาในการจ่ายกระแสไฟในวงจร ปัญหาเกี่ยวกับการรบกวนกันภานในวงจร ซึ่งจะพบว่าในส่วนของการเขียนโปรแกรมควบคุมและส่วนของการเพิ่มความสามารถในการประมวลผล เป็นเพียงส่วนประกอบหนึ่งเท่านั้น

# ภาคผนวก ก.

## การสร้างแฟ้มข้อมูลสำหรับการคอมไพล์แฟ้มข้อมูลที่ใช้งานไมโครซีโอเอส2

ในการคอมไพล์จะต้องมีการสร้างแฟ้มข้อมูล ขึ้นมาเพื่อเรียกใช้งานโปรแกรมที่ใช้งานซึ่งภายในแฟ้มข้อมูลที่สร้างขึ้นมาจะประกอบด้วย (ใช้แฟ้มข้อมูลนามสกุล .MAK)

**1. การกำหนดเครื่องมือที่ใช้มาในการคอมไพล์**

```
BORLAND=c:\borlandc
CC=$(BORLAND)\BIN\BCC
ASM=$(BORLAND)\BIN\TASM
LINK=$(BORLAND)\BIN\TLINK
TOUCH=$(BORLAND)\BIN\TOUCH
```

**2.ใส่ตำแหน่งไดเรคทอรี่ที่จะไปคอมไพล์**

```
LST=..\LST
OBJ=..\OBJ
SOURCE=..\SOURCE
TARGET=..\TEST
WORK=..\WORK

OS=\SOFTWARE\uCOS-II\SOURCE
COM86=\SOFTWARE\BLOCKS\COM86\BC31
PORT=\SOFTWARE\uCOS-II\C86L\BC31
```

**3. การกำหนดค่าการคอมไพล์**

```
###############################################################################
#                    COMPILER FLAGS
#
# -1          Generate 80186 code
# -B          Compile and call assembler
# -c          Compiler to .OBJ
# -G          Select code for speed
# -I          Path to include   directory
# -k-         Standard stack frame
# -L          Path to libraries directory
# -ml         Large memory model
# -N-         Do not check for stack overflow
# -n          Path to object directory
# -O          Optimize jumps
# -Ob         Dead code elimination
# -Oe         Global register allocation
# -Og         Optimize globally
# -Ol         Loop optimization
# -Om         Invariant code motion
# -Op         Copy propagation
# -Ov         Induction variable
# -v          Source debugging ON
# -vi         Turn inline expansion ON
# -wpro       Error reporting: call to functions with no prototype
# -Z          Suppress redundant loads
###############################################################################
```

```
C_FLAGS=-c -ml -1 -G -O -Ogemvlbpi -Z -d -n..\obj -k- -v -vi- -wpro -I$(BORLAND)\INCLUDE -
L$(BORLAND)\LIB

###############################################################################
#                       ASSEMBLER FLAGS
#
# /MX           Case sensitive on globals
# /ZI           Full debug info
# /O            Generate overlay code
###############################################################################
#

ASM_FLAGS=/MX /ZI /O
```

**4.กำหนดแฟ้มข้อมูลเอดเดอร์ที่เกี่ยวข้อง**

```
INCLUDES=               $(SOURCE)\INCLUDES.H \
                $(SOURCE)\devices.H \
                        $(SOURCE)\OS_CFG.H   \
                $(PORT)\OS_CPU.H     \
                        $(COM86)\COM86.H     \
                $(COM86)\AM186CC.H   \
                $(OS)\uCOS_II.H
```

**5. กำหนดแฟ้มข้อมูลที่ใช้ในการสร้างแฟ้มข้อมูลที่จะนำไปใช้งาน**

```
$(TARGET)\TEST.EXE:             \
        $(WORK)\INCLUDES.H   \
        $(OBJ)\OS_CPU_A.OBJ  \
        $(OBJ)\OS_CPU_C.OBJ  \
        $(OBJ)\OS_DEBUG.OBJ  \
        $(OBJ)\COM86.OBJ     \
        $(OBJ)\TEST.OBJ      \
        $(OBJ)\devices.OBJ   \
        $(OBJ)\uCOS_II.OBJ   \
        $(SOURCE)\TEST.LNK
        COPY   $(SOURCE)\TEST.LNK
        $(LINK) $(LINK_FLAGS)    @TEST.LNK
        COPY   $(OBJ)\TEST.EXE  $(TARGET)\TEST.EXE
        COPY   $(OBJ)\TEST.MAP  $(TARGET)\TEST.MAP
        DEL    TEST.MAK
```

**6.ทำการสร้างแฟ้มข้อมูลที่จะนำมาประกอบเป็นแฟ้มข้อมูลที่ทำงานได้**

```
$(WORK)\INCLUDES.H:             \
        $(INCLUDES)
        COPY  $(SOURCE)\INCLUDES.H INCLUDES.H
        COPY  $(SOURCE)\devices.H  devices.H
        COPY  $(SOURCE)\OS_CFG.H   OS_CFG.H
        COPY  $(COM86)\COM86.H     COM86.H
        COPY  $(PORT)\OS_CPU.H     OS_CPU.H
        COPY  $(OS)\uCOS_II.H      uCOS_II.H

$(OBJ)\OS_CPU_A.OBJ:            \
        $(PORT)\OS_CPU_A.ASM
        COPY  $(PORT)\OS_CPU_A.ASM OS_CPU_A.ASM
        $(ASM) $(ASM_FLAGS) $(PORT)\OS_CPU_A.ASM,$(OBJ)\OS_CPU_A.OBJ
```

```
$(OBJ)\OS_CPU_C.OBJ:            \
        $(PORT)\OS_CPU_C.C      \
        COPY  $(PORT)\OS_CPU_C.C   OS_CPU_C.C
        $(CC) $(C_FLAGS)           OS_CPU_C.C

$(OBJ)\OS_DEBUG.OBJ:            \
        $(PORT)\OS_DEBUG.C      \
        COPY  $(PORT)\OS_DEBUG.C   OS_DEBUG.C
        $(CC) $(C_FLAGS)           OS_DEBUG.C

$(OBJ)\COM86.OBJ:               \
        $(COM86)\COM86.C        \
        $(INCLUDES)
        COPY  $(COM86)\COM86.C     COM86.C
        $(CC) $(C_FLAGS)        . COM86.C

$(OBJ)\devices.OBJ:             \
        $(SOURCE)\devices.CPP   \
        $(INCLUDES)
        COPY  $(SOURCE)\devices.CPP    devices.CPP
        $(CC) $(C_FLAGS)           devices.CPP


$(OBJ)\TEST.OBJ:                \
        $(SOURCE)\TEST.CPP      \
        $(INCLUDES)
        COPY  $(SOURCE)\TEST.CPP    TEST.CPP
        $(CC) $(C_FLAGS)           TEST.CPP


$(OBJ)\uCOS_II.OBJ:             \
        $(OS)\uCOS_II.C         \
        $(INCLUDES)
        COPY  $(OS)\uCOS_II.C     uCOS_II.C
        $(CC) $(C_FLAGS)          uCOS_II.C
```

หลังจากสร้างแฟ้มข้อมูลในการสร้างแฟ้มข้อมูลที่ใช้งานแล้วต้องสร้างแฟ้มข้อมูลที่เก็บส่วนที่เกี่ยวข้องกับแฟ้มข้อมูลที่ทำงานด้วย (ใช้แฟ้มข้อมูลนามสกุล .LNK) ซึ่งจะกำหนดไว้ในส่วนการกำหนดแฟ้มข้อมูลที่ใช้ในการสร้างแฟ้มข้อมูลที่จะนำไปใช้งาน

```
/v /s /c /P-          +
c:\borlandc\LIB\C0L.OBJ  +
..\OBJ\TEST.OBJ       +
..\OBJ\OS_CPU_A.OBJ    +
..\OBJ\OS_CPU_C.OBJ    +
..\OBJ\OS_DEBUG.OBJ    +
..\OBJ\devices.OBJ              +
..\OBJ\COM86.OBJ      +
..\OBJ\uCOS_II.OBJ
..\OBJ\TEST,..\OBJ\TEST
c:\borlandc\LIB\EMU.LIB  +
c:\borlandc\LIB\MATHL.LIB +
c:\borlandc\LIB\CL.LIB
```

หลังจากนั้นก็ทำการเขียนแฟ้มข้อมูลที่สั่งให้ทำการสร้างแฟ้มข้อมูลที่เรียกใช้งานขึ้นมา (ใช้แฟ้มข้อมูลนามสกุล .BAT)

```
MD    ..\WORK
MD    ..\OBJ
MD    ..\LST
CD    ..\WORK
COPY  ..\TEST\TEST.MAK   TEST.MAK
c:\borlandc\BIN\MAKE -f TEST.MAK
CD    ..\TEST
```

# ภาคผนวก ข.
# ชุดคำสั่งควบคุมอุปกรณ์สำหรับหุ่นยนต์และการนำไปใช้

ชุดคำสั่งควบคุมอุปกรณ์สำหรับหุ่นยนต์แบ่งเป็น 2 ส่วน

1. ส่วนของแฟ้มข้อมูลแบบไลเบอร์รี่ ที่เก็บคลาสของอุปกรณ์ชนิดต่างไว้
2. ส่วนของทาสก์ที่ใช้ควบคุมอุปกรณ์แต่ละชนิด

- **ส่วนของแฟ้มข้อมูลแบบไลเบอร์รี่**

จะประกาศคลาสต่างๆไว้ในแฟ้มข้อมูล devices.h และจะเก็บโปรแกรมแสดงการทำงานของเมทธอดในแต่คลาสไว้ในแฟ้มข้อมูล devices.cpp เพื่อให้ผู้ใช้งานเรียกใช้ได้งาน

แฟ้มข้อมูล devices.h

```
//-------------------------------- DC MOTOR ----------------------------------
class DCmotor
        {
                private :
                        OS_EVENT *Mbox;
                        INT16U port_out1;
                        char port_status;
                public :
                        char name[12];
                        void setDCmotor(OS_EVENT *M,INT16U p1,int p2,char p3,char N[]);
                        void forward();
                        void backward();
                        void stop();
        };

//-------------------------------- STEP MOTOR ----------------------------------

class STEPmotor
        {
                private :
                        OS_EVENT *Mbox;
                        int port_out;
                        char port_status;
                public :
                        char name[12];
                        void setSTEPmotor(OS_EVENT *M,INT16U p1,char p3,char N[]);
                        void forward ();
                        void backward();
                        void stop();
        };
```

```
//-------------------------------- IR SENSOR ------------------------------------

class IRsensor
        {
                private :
                        OS_EVENT *Mbox;
                        INT16U port_in;
                public :
                        char name[12];
                        void setIRsensor(OS_EVENT *M,INT16U p1,char N[]);
                        int detect();
        };

//-------------------------------- BUMPED SENSOR ---------------------------------

class BUMPsensor
        {
                private :
                        OS_EVENT *Mbox;
                        INT16U port_in;
                public :
                        char name[12];
                        void setBUMPsensor(OS_EVENT *M,INT16U p1,char N[]);
                        int bump();
        };

//------------------------------------ End all Class of devices.h ------------------------------
```

จากแฟ้มข้อมูล devices.h จะแสดงค่าแอตทริบิวต์และเมทธอด ของคลาสต่างๆโดยจะเห็นว่าแต่ละคลาสจะประกอบด้วยส่วนหลักๆดังนี้

- แอตทริบิวต์ Mbox เป็นตัวแปรชนิด พ้อยเตอร์ของ OS_EVENT ซึ่งจะมีไว้สำหรับอ้างถึงเมล์บ็อกที่ใช้ในการสื่อสารระหว่างอุปกรณ์ กับทาสก์ที่ควบคุมอุปกรณ์ชนิดนั้นๆ โดยอุปกรณ์ 1 ตัวจะมี 1 เมล์บ็อก
- แอตทริบิวต์ port_in,port_out เป็นตัวแปรชนิด int ซึ่งจะใช้เก็บว่าเป็นอุปกรณ์ตัวใดในจำนวนอุปกรณ์ที่เป็นชนิดเดียวกันทั้งหมด หรือก็คือการกำหนดหมายเลขให้กับอุปกรณ์เพื่อสามารถสั่งงานอุปกรณ์ได้ตรงตามที่ต้องการ
- แอตทริบิวต์ name เป็นตัวแปรชนิด อาร์เรย์ของ char ซึ่งจะใช้เก็บชื่อของอุปกรณ์ตามที่ผู้ใช้กำหนดไว้
- เมทธอด ที่ขึ้นต้นด้วยคำว่า set...(...) จะเป็นเมทธอดที่ใช้ในการกำหนดค่าแอตทริบิวต์ต่างๆให้กับคลาส

ส่วนของเมทธอดนอกเหนือจากนั้นจะเป็นเมทธอดที่ใช้ในการสั่งงานควบคุมอุปกรณ์ของแต่ละชนิด

แฟ้มข้อมูล devices.cpp

```
#include "includes.h"

//--------------------------------- DC MOTOR ------------------------------------
void DCmotor ::setDCmotor(OS_EVENT *M,INT16U p1,char p3,char N[])
        {
                Mbox = M;
                port_out1 = p1;
                port_status    = p3;
                strcpy(name,N);
        }
void DCmotor::forward()
        {
        static char jms[3];

         port_status = jms[port_out1-1];


        jms[port_out1-1] = 0x01<<((port_out1-1)*2);

        OSMboxAccept(Mbox);
        OSMboxPost(Mbox,(void *)&jms[port_out1-1]);

        }
void DCmotor::backward()
        {
        static char jms[3];

         port_status = jms[port_out1-1];


        jms[port_out1-1] = 0x02<<((port_out1-1)*2);

        OSMboxAccept(Mbox);
        OSMboxPost(Mbox,(void *)&jms[port_out1-1]);

        }

void DCmotor::stop()
        {
        static char jms;

        port_status = jms;

        jms = 0;
        OSMboxAccept(Mbox);
        OSMboxPost(Mbox,(void *)&jms);


        }
```

ในส่วนของแฟ้มข้อมูล devices.cpp ที่ใช้ในการควบคุมดีซีมอเตอร์ โดยทั่วไปจะเป็นการคำนวณค่าของดีซีมอเตอร์ แต่ละตัวเพื่อส่งให้ทาสก์ดีซีมอเตอร์ทางเมล์บ็อก ประมวลผลส่งไปควบคุมดีซีมอเตอร์จริงๆ

```
//-------------------------------- STEP MOTOR ----------------------------------

void STEPmotor::setSTEPmotor(OS_EVENT *M,INT16U p1,char p3,char N[])
        {

                char s[80];
                INT8U err;
                Mbox = M;
                port_out   = p1;
                port_status    = p3;
                strcpy(name,N);
        }

void STEPmotor::forward()
        {
        static char jms[2];

        port_status = jms[port_out-1];

         if (port_out==1)
         {
           jms[0] =12;
        }
        else
        {
          jms[1] =3;
         }

        OSMboxAccept(Mbox);
        OSMboxPost(Mbox,(void *)&jms[port_out-1]);
        }

void STEPmotor::backward()
        {
        static char jms[2];

        port_status = jms[port_out-1];

         if (port_out==1)
        {
          jms[0] = 8;
        }
        else
        {
          jms[1] =2;
        }

        OSMboxAccept(Mbox);
        OSMboxPost(Mbox,(void *)&jms[port_out-1]);
        }

void STEPmotor::stop()
        {
        static char jms;

        port_status = jms;
        jms = 0;

        OSMboxAccept(Mbox);
        OSMboxPost(Mbox,(void *)&jms);
        }
```

ในส่วนของแฟ้มข้อมูล devices.cpp ที่ใช้ในการควบคุม สเต็ปมอเตอร์ โดยทั่วไปจะเป็นการคำนวณค่าของสเต็ปมอเตอร์ แต่ละตัวในที่นี้ใช้ควบคุม สเต็ปมอเตอร์ 2 ตัวเพื่อส่งให้ทาสก์สเต็ปมอเตอร์ผ่านทางเมล์บ็อก ประมวลผลส่งไปควบคุมสเต็ปมอเตอร์จริงๆ

```
//---------------------------------- IR SENSOR ----------------------------------

void IRsensor::setIRsensor(OS_EVENT *M,INT16U p1,char N[])
        {

        Mbox = M;
        port_in = p1;
        strcpy(name,N);

        }
int IRsensor::detect()
        {
        char  *IRin;


        IRin= (char *) OSMboxAccept(Mbox);
        OSMboxPost(Mbox,IRin);


                if( *IRin== 0x10 )
                {
                return 1;
                }
                else return 0;
        }
```

ในส่วนของแฟ้มข้อมูล devices.cpp ที่ใช้ในการควบคุม อินฟราเรดเซนเซอร์ จะเป็นการรับข้อมูลมาจากเมล์บ็อกที่ส่งมาจากทาสก์ของ อินฟราเรดเซนเซอร์ โดยทาสก์ อินฟราเรดเซนเซอร์ จะส่งค่ามาบอกว่า อินฟราเรดเซนเซอร์ตัวนั้น มีค่าเป็นอะไร( ทำงานอยู่หรือไม่ทำงาน) แล้วเมทธอด detect() จะส่งค่ากลับไปให้ผู้เรียกว่าอินฟราเรดเซนเซอร์ตัวนั้น ทำงานอยู่หรือไม่ ถ้าทำงานจะส่งค่า 1 ถ้าไม่ทำงานจะส่งค่า 0

```
//---------------------------------- BUMPED SENSOR ----------------------------------

void BUMPsensor::setBUMPsensor(OS_EVENT *M,INT16U p1,char N[])
        {       Mbox = M;
                port_in = p1;
                strcpy(name,N);
        }

int BUMPsensor::bump()
        {
        char  *Bump;

        Bump= (char *) OSMboxAccept(Mbox);
        OSMboxPost(Mbox,Bump);
         if( *Bump ==1)
                { return 1;}
       else { return 0;}
       }
```

ในส่วนของแฟ้มข้อมูล devices.cpp ที่ใช้ในการควบคุม บัมเซนเซอร์ จะเป็นการรับข้อมูลมาจากเมล์บ็อก ที่ส่งมาจากทาสก์ของ บัมเซนเซอร์ โดยทาสก์ อินฟราเรดเซนเซอร์ จะส่งค่ามาบอกว่า บัมเซนเซอร์ ตัวนั้น มีค่าเป็นอะไร( ถูกกระทบหรือไม่ถูกกระทบ) แล้วเมทธอด bump() จะส่งค่ากลับไปให้ผู้เรียกว่า บัมเซนเซอร์ตัวนั้น ถูกกระทบอยู่หรือไม่ ถ้าถูกกระทบจะส่งค่า 1 ถ้าไม่ถูกกระทบจะส่งค่า 0

**ตัวอย่างวิธีการเรียกใช้ชุดคำสั่งที่เป็นไลเบอร์รี่**

```
---------- ส่วนของการประกาศออบเจ็ก ----------
        BUMPsensor B1;
        STEPmotor step1;
        STEPmotor step2;


---------- ส่วนของการเรียกใช้งานเมทธอด ----------
        if (B1.bump ())
        {
                step1.backward ();
                step2.stop ();

                OSTimeDlyHMSM (0, 0, 1, 0);

                step1.forward ();

                OSTimeDlyHMSM (0, 0, 1, 0);

                step1.stop ();

        }
        else
        {
                step1.stop ();
                step2.backward ();

                OSTimeDlyHMSM (0, 0, 1, 0);

                step2.forward ();

                OSTimeDlyHMSM (0, 0, 1, 0);

                step2.stop ();


        }
```

* ส่วนการประกาศออบเจ็ก โดยปกติโปรแกรมช่วยเหลือจะประกาศให้ ผู้ใช้สามารถเรียกใช้ได้เลย

- **ส่วนของทาสก์ที่ใช้ควบคุมอุปกรณ์**

ซึ่งจะแยกเก็บเป็นแฟ้มข้อมูลที่มีชื่อเหมือนอุปกรณ์นั้นๆ และจะถูกเรียกมารวมกับแฟ้มข้อมูลหลักเมื่อผู้ใช้เลือกอุปกรณ์ชนิดนั้น โดยจะมีหน้าที่ควบคุมอุปกรณ์นั้นๆ โดยตรง แบ่งเป็น

*ทาสก์ดีซีมอเตอร์*

```
void  Task2 (void *data)
{
  char  *rxmsg;
  int sh,oldsh;
  data = data;

        for (;;){
                oldsh=sh;
                rxmsg = (char *) OSMboxAccept (DCMbox);
                 sh=*rxmsg;

                if ((sh != 116) &&( sh!= oldsh))
                {
                        outportb(0x3020,sh);
                }

                 OSTimeDlyHMSM (0, 0, 1, 0);                    // Wait 1 second
                }
}
```

ทาสก์ดีซีมอเตอร์จะรับข้อมูลมาจากเมล์บ็อกที่ชื่อ DCMbox แล้วนำมาตรวจสอบค่าก่อนส่งข้อมูลออกไปทาง ไอซ่าบัสตำแหน่งที่ 3020 เพื่อควบคุมดีซีมอเตอร์ จากโปรแกรมข้างบนยังควบคุมดีซีมอเตอร์ 1 ตัวหากต้องการควบคุมเพิ่มจะต้องเพิ่ม เมล์บ็อกที่ส่งค่ามาให้สำหรับควบคุมดีซีมอเตอร์ และเพิ่มส่วนประมวลผลจัดการสำหรับส่งข้อมูลออกไอซ่าบัส โดยไอซ่าบัส 1 ตำแหน่งควบคุมดีซีมอเตอร์ได้สูงสุด 4 ตัว ถ้าต้องการควบคุมดีซีมอเตอร์มากกว่านั้นต้องเพิ่มการอ้างตำแหน่งของไอซ่าบัส ส่วนของการหน่วงเวลาช่วงท้ายเป็นการกระทำเพื่อให้ทาสก์อื่นๆ สามารถแย่งชิงตัวประมวลผลไปทำงานได้ทำให้ไม่มีทาสก์ใดควบคุมตัวประมวลผลเพียงตัวเดียว

*ทาสก์สเต็ปมอเตอร์*

```
void  Task3 (void *data)
{   char  *step1;
   char  *step2;
    int sh,oldsh,sh1,sh2;

   sh=0;
   *step1=0;
   *step2=0;
   data = data;

for (;;) {
        sh1=*step1;
        sh2=*step2;
        oldsh=sh;

        step1 = (char *) OSMboxAccept(Step1Mbox);          OSMboxPost(Step1Mbox,step1);
        step2 = (char *)OSMboxAccept(Step2Mbox);           OSMboxPost(Step2Mbox,step2);
```

```
        if (*step1==116)
                { *step1=sh1;}
        if (*step2==116)
                {*step2=sh2;}
        sh=*step1|*step2|0x10;

        if ((sh != 116) &&( sh!= oldsh))
                { outportb(0x3001,sh); }
         OSTimeDlyHMSM(0, 0, 0,70);
    }
}
```

ทาสก์สเต็ปมอเตอร์จะรับข้อมูลมาจากเมล์บ็อกที่ชื่อ Step1Mbox และ Step2Mbox สำหรับควบคุมสเต็ปมอเตอร์ 2 ตัว แล้วนำมาประมวลผลรวมค่าที่ได้จากเมล์บ็อกทั้งสอง ส่งข้อมูลอออกไปทาง ไอซ่าบัสตำแหน่งที่ 3001 เพื่อควบคุมสเต็ปมอเตอร์ จากโปรแกรมข้างบนยังควบคุมสเต็ปมอเตอร์ 2 ตัวหากต้องการควบคุมเพิ่มจะต้องเพิ่ม เมล์บ็อกที่ส่งค่ามาให้สำหรับควบคุมสเต็ปมอเตอร์และเพิ่มส่วนประมวลผลจัดการสำหรับส่งข้อมูลออกไอซ่าบัส โดยไอซ่าบัส 1 ตำแหน่งควบคุมสเต็ปมอเตอร์ได้สูงสุด 3 ตัว (สามารถดัดแปลงให้ควบคุมถึง 4 ตัวได้) ถ้าต้องการควบคุมสเต็ปมอเตอร์มากกว่านั้นต้องเพิ่มการอ้างตำแหน่งของไอซ่าบัส

*ทาสก์อินฟราเรดเซนเซอร์*

```
void  Task4 (void *data)
{
        INT16U pt;
        char inIR,inIR1;
        char true,false;

        data  = data;
        true=0x10;
        false=0x11;

       for (;;) {
                pt=0x3004;
                inIR = inportb(pt);


                inIR1=inIR&(0x01<<0);
                if (inIR1==0x01)
                {
                OSMboxAccept(IR1Mbox);
                 OSMboxPost(IR1Mbox, (void *)true);
                }
                else
                {
                 OSMboxAccept(IR1Mbox);
                 OSMboxPost(IR1Mbox, (void *)false);
                }

                inIR1=inIR&(0x01<<1);
                if (inIR1==0x02)
                {
                 OSMboxAccept(IR2Mbox);
                 OSMboxPost(IR2Mbox, (void *)true);
                 }
```

```
    else
    {
     OSMboxAccept(IR2Mbox);
     OSMboxPost(IR2Mbox, (void *)false);
     }

   inIR1=inIR&(0x01<<2);
   if (inIR1==0x04)
   {
     OSMboxAccept(IR3Mbox);
     OSMboxPost(IR3Mbox, (void *)true);
   }
    else
    {
    OSMboxAccept(IR3Mbox);
    OSMboxPost(IR3Mbox, (void *)false);
    }

    inIR1=inIR&(0x01<<3);
   if (inIR1==0x08)
   {
     OSMboxAccept(IR4Mbox);
     OSMboxPost(IR4Mbox, (void *)true);
    }
    else
    {
    OSMboxAccept(IR4Mbox);
    OSMboxPost(IR4Mbox, (void *)false);
    }

  inIR1=inIR&(0x01<<4);
  if (inIR1==0x010)
  {
    OSMboxAccept(IR5Mbox);
    OSMboxPost(IR5Mbox, (void *)true);
   }
   else
   {
   OSMboxAccept(IR5Mbox);
   OSMboxPost(IR5Mbox, (void *)false);
   }

 inIR1=inIR&(0x01<<5);
 if (inIR1==0x20)
 {
   OSMboxAccept(IR6Mbox);
   OSMboxPost(IR6Mbox, (void *)true);
  }
  else
  {
  OSMboxAccept(IR6Mbox);
  OSMboxPost(IR6Mbox, (void *)false);
  }

inIR1=inIR&(0x01<<6);
if (inIR1==0x40)
{
  OSMboxAccept(IR7Mbox);
  OSMboxPost(IR7Mbox, (void *)true);
 }
```

```
                    else
                    {
                    OSMboxAccept(IR7Mbox);
                    OSMboxPost(IR7Mbox, (void *)false);
                    }

                   inIR1=inIR&(0x01<<7);
                   if (inIR1==0x80)
                   {
                     OSMboxAccept(IR8Mbox);
                     OSMboxPost(IR8Mbox, (void *)true);
                   }
                   else
                    {
                    OSMboxAccept(IR8Mbox);
                    OSMboxPost(IR8Mbox, (void *)false);
                    }
          OSTimeDlyHMSM(0, 0, 0,30);
    }
}
```

ทาสก์อินฟราเรดเซนเซอร์จะรับข้อมูลมาทางตำแหน่งที่ 3004 ของไอซ่าบัสแล้วนำมาประมวลผลว่าอินฟราเรดเซนเซอร์แต่ละตัวมีค่าเป็นอะไรแล้วส่งค่าไปเก็บไว้ในเมล์บ็อกของอินฟราเรดเซนเซอร์แต่ละตัว เพื่อให้ผู้ใช้งานสามารถเรียกค่าของอินฟราเรดเซนเซอร์แต่ละตัวมาใช้งานได้ทันที โดยโปรแกรมข้างบนจะจัดการกับอินฟราเรดเซนเซอร์จำนวน 8 ตัว ถ้าใช้น้อยกว่าจำนวนที่กำหนดให้สามารถแก้ไขลบส่วนที่ไม่ใช้ออกไปได้ แต่ถ้าต้องการเพิ่มจำนวนต้องมีการเพิ่มการอ้างตำแหน่งของไอซ่าบัสในการรับค่า เนื่องจาก ไอซ่าบัส 1 ตำแหน่งควบคุมอินฟราเรดเซนเซอร์ได้จำนวน 8 ตัว

*ทาสก์บัมเซนเซอร์*

```
void  Task5 (void *data)
{
        char true,false;
        INT8U inBump,tempbump,tem;

        data = data;
         true=0x01;
        false=0x11;

        for (;;) {
                 inBump = inportb(0x3010);

                 tempbump=inBump|0xfe;
                 if (tempbump==0xfe)
                 {
                 OSMboxAccept(Bump1Mbox);
                 OSMboxPost(Bump1Mbox, (void *)&true);
                 }
                 else
                 {
                  OSMboxAccept(Bump1Mbox);
                  OSMboxPost(Bump1Mbox, (void *)&false);
                 }
```

```
 tempbump=inBump|0xfd;
 if (tempbump ==0xfd)
 {
 OSMboxAccept(Bump2Mbox);
  OSMboxPost(Bump2Mbox, (void *)&true);
 }
 else
 {
  OSMboxAccept(Bump2Mbox);
  OSMboxPost(Bump2Mbox, (void *)&false);
 }

 tempbump=inBump|0xfb;
 if (tempbump==0xfb)
 {
 OSMboxAccept(Bump3Mbox);
  OSMboxPost(Bump3Mbox, (void *)&true);
 }
 else
 {
  OSMboxAccept(Bump3Mbox);
  OSMboxPost(Bump3Mbox, (void *)&false);
 }

 tempbump=inBump|0xf7;
 if (tempbump==0xf7)
 {
 OSMboxAccept(Bump4Mbox);
 OSMboxPost(Bump4Mbox, (void *)&true);
 }
 else
 {
  OSMboxAccept(Bump4Mbox);
  OSMboxPost(Bump4Mbox, (void *)&false);
 }

 tempbump=inBump|0xef;
 if (tempbump==0xef)
 {
 OSMboxAccept(Bump5Mbox);
 OSMboxPost(Bump5Mbox, (void *)&true);
 }
 else
 {
  OSMboxAccept(Bump5Mbox);
  OSMboxPost(Bump5Mbox, (void *)&false);
 }

 tempbump=inBump|0xdf;
 if (tempbump==0xdf)
 {  tem=1;
 OSMboxAccept(Bump6Mbox);
 OSMboxPost(Bump6Mbox, (void *)&true);
 }
 else
 {
  OSMboxAccept(Bump6Mbox);
  OSMboxPost(Bump6Mbox, (void *)&false);
 }
```

```
                tempbump=inBump|0xbf;
                if (tempbump==0xbf)
                {
                OSMboxAccept(Bump7Mbox);
                OSMboxPost(Bump7Mbox, (void *)&true);
                }
                else
                {
                 OSMboxAccept(Bump7Mbox);
                 OSMboxPost(Bump7Mbox, (void *)&false);
                }

                tempbump=inBump|0x7f;
                if (tempbump==0x7f)
                {
                OSMboxAccept(Bump8Mbox);
                OSMboxPost(Bump8Mbox, (void *)&true);
                }
                else
                {
                 OSMboxAccept(Bump8Mbox);
                 OSMboxPost(Bump8Mbox, (void *)&false);
                }

                OSTimeDlyHMSM(0, 0,1,0);
        }
}
```

ทาสก์บัมเซนเซอร์จะทำงานคล้ายกับ ทาสก์อินฟราเรดเซนเซอร์ แต่ต่างกันที่ ทาสก์บัมเซนเซอร์ ทำงานที่ค่าเป็น 0 และค่าปกติเป็น 1

โดยทาสก์ต่างจะถูกรวมไว้ภายใต้โครงสร้างการทำงานของไมโครซีโอเอส2 ซึ่งมีรูปแบบดังนี้

```
#include "includes.h"

#define     TASK_STK_SIZE   512         /// กำหนด ขนาด สแต็ก ของทาสก์

////////////////////////////////////////////// กำหนด ID ////////////////////////////////////////////////
#define     TASK_START_ID     0         /* Application tasks IDs      */
#define     TASK_CLK_ID       1
#define     TASK_1_ID         2
#define     TASK_2_ID         3
#define     TASK_3_ID         4
#define     TASK_4_ID         5
#define     TASK_5_ID         6

////////////////////////////////////////////// กำหนด Priority ////////////////////////////////////////////////
#define     TASK_START_PRIO   10          /* Application tasks priorities          */
#define     TASK_CLK_PRIO     17
#define     TASK_1_PRIO       12
#define     TASK_2_PRIO       13
#define     TASK_3_PRIO       14
#define     TASK_4_PRIO       15
#define     TASK_5_PRIO       16

////////////////////////////////////////////// กำหนด task stack ////////////////////////////////////////////////
OS_STK      TaskStartStk[TASK_STK_SIZE];        /* Startup  task stack                  */
OS_STK      TaskClkStk[TASK_STK_SIZE];          /* Clock    task stack                  */
OS_STK      Task1Stk[TASK_STK_SIZE];            /* Task #1  task stack                  */
```

```
OS_STK      Task2Stk[TASK_STK_SIZE];        /* Task #2   task stack          */
OS_STK      Task3Stk[TASK_STK_SIZE];        /* Task #3   task stack          */
OS_STK      Task4Stk[TASK_STK_SIZE];        /* Task #4   task stack          */
OS_STK      Task5Stk[TASK_STK_SIZE];        /* Task #5   task stack          */

////////////////////////////////////////////// กำหนด task ////////////////////////////////////////////////////////////////
     void  TaskStart(void *data);           /* Function prototypes of tasks  */
static  void  TaskStartCreateTasks(void);
static  void  TaskStartDispInit(void);
static  void  TaskStartDispInit2(int);
static  void  TaskStartDisp(void);
              void  TaskClk(void *data);
              void  Task1(void *data);
     void  Task2(void *data);
     void  Task3(void *data);
     void  Task4(void *data);
     void  Task5(void *data);

OS_EVENT    *DispSem  //// SEMAPHORES สำหรับแสดงผล /////
///  create os event for mailbox for device  ///////////////////////

OS_EVENT    *DCMbox;
OS_EVENT    *Step1Mbox;
OS_EVENT    *Step2Mbox;
OS_EVENT    *IR1Mbox;
OS_EVENT    *IR2Mbox;
OS_EVENT    *IR3Mbox;
OS_EVENT    *Bump1Mbox;
OS_EVENT    *Bump2Mbox;
OS_EVENT    *Bump3Mbox;

OS_EVENT    *IR4Mbox;
OS_EVENT    *IR5Mbox;
OS_EVENT    *IR6Mbox;
OS_EVENT    *IR7Mbox;
OS_EVENT    *IR8Mbox;

OS_EVENT    *Bump4Mbox;
OS_EVENT    *Bump5Mbox;
OS_EVENT    *Bump6Mbox;
OS_EVENT    *Bump7Mbox;
OS_EVENT    *Bump8Mbox;
OS_EVENT    *SonarMbox;
OS_EVENT    *Senddis1;
OS_EVENT    *Senddis2;
////////////////////////////// จบ create os event for mailbox for device//////////////////////////////////

//////////////////////////     ประกาศ object ตามชื่อ : ชื่อคลาส ชื่อออปเจค(ที่กำหนด) ////////////////////////////

//// ตัวอย่าง ////////
BUMPsensor B1;
BUMPsensor B2;
STEPmotor step1;
STEPmotor step2;


////////////////////////////////////////////////// จบ ประกาศ object//////////////////////////////////////////////////////


void ma n (void)  ////////////    main uCOS ////////////////////////////////////////////////////////////////
{
   OS_STK *ptos;
```

```
    OS_STK *pbos;
    INT32U  size;

    COM86_DispInit(); ///กำหนดค่าเริ่มต้นของหน้าจอแสดงผล/////
    COM86_DispClrScr(DISP_BLACK, DISP_WHITE);          /* Clear the screen            */

        OSInit();                                      /* Initialize uC/OS-II         */
    COM86_DOSSaveReturn();                             /* Save environment to return to DOS       */
    COM86_VectSet(uCOS, OSCtxSw);                      /* Install uC/OS-II's context switch vector */
    DispSem    = OSSemCreate(1);                       /* Console display semaphore               */
    COM86_ElapsedInit();                               /* Initialized elapsed time measurement    */

        ptos       = &TaskStartStk[TASK_STK_SIZE - 1];     /* TaskStart() will use Floating-Point
*/
    pbos       = &TaskStartStk[0];
    size       = TASK_STK_SIZE;

        OSTaskStkInit_FPE_x86(&ptos, &pbos, &size);


        ////////// สร้าง taskstart ////////////////////////////////////////

        OSTaskCreateExt(TaskStart,
            (void *)0,
            ptos,
            TASK_START_PRIO,
            TASK_START_ID,
            pbos,
            size,
            (void *)0,
            OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
    OSStart();                                         /* Start multitasking                      */
        ////////////////////////////////////// จบ สร้าง taskstart ////////////////////////
}
/*
*********************************************************************************************************
*                                          STARTUP TASK
*********************************************************************************************************
*/

void  TaskStart (void *pdata)
{
#if OS_CRITICAL_METHOD == 3                            /* Allocate storage for CPU status register */
    OS_CPU_SR  cpu_sr;
#endif
  char s[80];
        INT16S    key;
        INT8U   err;
        pdata = pdata;                                 /* Prevent compiler warning                */

  TaskStartDispInit();                                 /* Setup the display                       */

  OS_ENTER_CRITICAL();                                 /* Install uC/OS-II's clock tick ISR       */
  COM86_VectSet(0x08, OSTickISR);
  COM86_SetTickRate(OS_TICKS_PER_SEC);                 /* Reprogram tick rate                     */
  OS_EXIT_CRITICAL();

  OSStatInit();                                        /* Initialize uC/OS-II's statistics        */

////////////////////////////////////// create mail box ////////////////////////////////////////////////////////
  /* Create message mailboxes                          */
```

```
        DCMbox = OSMboxCreate((void *)0);
        Step1Mbox = OSMboxCreate((void *)0);
        Step2Mbox = OSMboxCreate((void *)0);
        IR1Mbox = OSMboxCreate((void *)0);
        IR2Mbox = OSMboxCreate((void *)0);
        IR3Mbox = OSMboxCreate((void *)0);
        Bump1Mbox = OSMboxCreate((void *)0);
        Bump2Mbox = OSMboxCreate((void *)0);
        Bump3Mbox = OSMboxCreate((void *)0);


        IR4Mbox = OSMboxCreate((void *)0);
        IR5Mbox = OSMboxCreate((void *)0);
        IR6Mbox = OSMboxCreate((void *)0);
        IR7Mbox = OSMboxCreate((void *)0);
        IR8Mbox = OSMboxCreate((void *)0);
        Bump4Mbox = OSMboxCreate((void *)0);
        Bump5Mbox = OSMboxCreate((void *)0);
        Bump6Mbox = OSMboxCreate((void *)0);
        Bump7Mbox = OSMboxCreate((void *)0);
        Bump8Mbox = OSMboxCreate((void *)0);

        Senddis1= OSMboxCreate((void *)0);
        Senddis2= OSMboxCreate((void *)0);
        SonarMbox = OSMboxCreate((void *)0);


        ////////////////////////////  call init variable  ////////////////////////////////////
        /////// เรียกเมทธอดในการ กำหนดค่าเริ่มต้นของ แอตทริบิวต์ ////////
        step1.setSTEPmotor(Step1Mbox,1,0,"jo");
        step2.setSTEPmotor(Step2Mbox,2,0,"jojo");
        B1.setBUMPsensor(Bump1Mbox,1,"jo1");
        B2.setBUMPsensor(Bump2Mbox,2,"jo2");
  ////////////////////////////จบ กำหนดค่าเริ่มต้น ////////////////////////////

    TaskStartCreateTasks();                 /* Create all other tasks               */
        for (;;) {
      TaskStartDisp();                      /* Update the display                   */
     if (COM86_GetKey(&key)) {              /* See if key has been pressed          */
        if (key == 0x1B) {                  /* Yes, see if it's the ESCAPE key      */
           COM86_DOSReturn();               /* Yes, return to DOS                   */
        }
     }
     OSCtxSwCtr = 0;                        /* Clear context switch counter         */
     OSTimeDly(OS_TICKS_PER_SEC);           /* Wait one second                      */
        }
}
////////////////////////////////////////////////// จบ  task main uCOS-II//////////////////////////////////////////////////////////////////////

//////////////////////////// task กำหนดค่าเริ่มต้นในการแสดงผล ////////////////////////////////////////////////
static void TaskStartDispInit (void)
{
   COM86_DispStr( 1,  1, "                    Robot  Mornitor                                                    ",
DISP_WHITE + DISP_BLINK, DISP_RED);
   COM86_DispStr( 1,  2, "
", DISP_BLACK, DISP_WHITE);
   COM86_DispStr( 1,  3, "  Device                                                                              ",
DISP_BLACK, DISP_WHITE);
```

```
    COM86_DispStr( 1,  4, " Stepmotor                                                                    ",
 DISP_BLACK, DISP_WHITE);
    COM86_DispStr( 1,  5, " Stepmotor                                                                    ",
 DISP_BLACK, DISP_WHITE);
    COM86_DispStr( 1,  6, " IRsensor       (IR8-IR0)
 ", DISP_BLACK, DISP_WHITE);
    COM86_DispStr( 1,  7, " Bumpsensor     (bump8-bump0)
 ", DISP_BLACK, DISP_WHITE);
    COM86_DispStr( 1, 23,"                <-PRESS 'ESC' TO QUIT->                                  ",
 DISP_BLACK + DISP_BLINK, DISP_WHITE);
 }
 /////////////////////////// จบ task กำหนดค่าเริ่มต้นในการแสดงผล ////////////////////////////////////////////////
 /////////////////////////// task กำหนดค่าการแสดงผล ////////////////////////////////////////////////
 static  void  TaskStartDisp (void)
 {
    char   s[80];
    INT8U  err;
           char *temp;
           char temp1;

           temp = (char *)OSMboxAccept(Step1Mbox);
           OSMboxPost(Step1Mbox, temp);

           if (*temp ==12)
 {
   sprintf(s, "%s\tforward ",step1.name);
           OSSemPend(DispSem, 0, &err);
           COM86_DispStr(25, 4, s, DISP_YELLOW, DISP_BLUE);
   OSSemPost(DispSem);
 }
           else if (*temp ==8)
 {
            sprintf(s, "%s\tbackward",step1.name);
           OSSemPend(DispSem, 0, &err);
           COM86_DispStr(25, 4, s, DISP_YELLOW, DISP_BLUE);
   OSSemPost(DispSem);
 }
 else if (*temp ==0)
           {
           sprintf(s, "%s\tstop    ",step1.name);
           OSSemPend(DispSem, 0, &err);
           COM86_DispStr(25, 4, s, DISP_YELLOW, DISP_BLUE);
   OSSemPost(DispSem);
           }

          temp = (char *)OSMboxAccept(Step2Mbox);
          OSMboxPost(Step2Mbox, temp);

          if (*temp ==3)
{
  sprintf(s, "%s\tforward " ,step2.name);
          OSSemPend(DispSem, 0, &err);
          COM86_DispStr(25, 5, s, DISP_YELLOW, DISP_BLUE);
   OSSemPost(DispSem);
}
 else if (*temp ==2)
{
                   sprintf(s, "%s\tbackward",step2.name);
          OSSemPend(DispSem, 0, &err);
          COM86_DispStr(25, 5, s, DISP_YELLOW, DISP_BLUE);
  OSSemPost(DispSem);
```

```
    }
    else if (*temp ==0)
             {
             sprintf(s, "%s\tstop    ",step2.name);
             OSSemPend(DispSem, 0, &err);
             COM86_DispStr(25, 5, s, DISP_YELLOW, DISP_BLUE);
      OSSemPost(DispSem);
             }
             temp1=inportb(0x3004);
             sprintf(s, "\t%x",temp1);
             OSSemPend(DispSem, 0, &err);
             COM86_DispStr(25, 6, s, DISP_YELLOW, DISP_BLUE);
      OSSemPost(DispSem);

             sprintf(s, " \t  ");
             OSSemPend(DispSem, 0, &err);
             COM86_DispStr(27, 6, s, DISP_YELLOW, DISP_BLUE);
      OSSemPost(DispSem);*/

            temp1=inportb(0x3010);
            sprintf(s, "\t%x  ",temp1);
            OSSemPend(DispSem, 0, &err);
            COM86_DispStr(25, 7, s, DISP_YELLOW, DISP_BLUE);
     OSSemPost(DispSem);

             sprintf(s, "\t  ");
            OSSemPend(DispSem, 0, &err);
             COM86_DispStr(27, 7, s, DISP_YELLOW, DISP_BLUE);
     OSSemPost(DispSem);
 }
 ////////////////////////// จบ task กำหนดค่าการแสดงผล ////////////////////////////////////////////////
 /*
 *********************************************************************************************************
 *                                      CREATE TASKS
 *********************************************************************************************************
 */
 ////////////////////////////////////////////// มีไว้สร้าง task ////////////////////////////////////////////
 static  void  TaskStartCreateTasks (void)
 {

         ////////////////////////////////////////////////////////// สร้างไว้ตามจำนวนที่ ประกาศไว้ข้างบน ////////////////////////////
   OSTaskCreateExt(TaskClk,//// ชื่อ/////////////////////
             (void *)0, ////////////ตามมัน/////////////////
             &TaskClkStk[TASK_STK_SIZE - 1],/////////// stack  ตาม top ////////////////////
                                    TASK_CLK_PRIO,/////////////// task prio /////////////////
             TASK_CLK_ID,////////////////task id /////////
             &TaskClkStk[0],////////////////////////  stack  ตาม bottum ////////////////////
             TASK_STK_SIZE,  ////////////////////////// fix ////////////////////
             (void *)0,  ////////////////////////// fix ////////////////////
             OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);  /////////////////////////// fix
//////////////////////

   OSTaskCreateExt(Task1,
             (void *)0,
             &Task1Stk[TASK_STK_SIZE - 1],
             TASK_1_PRIO,
             TASK_1_ID,
             &Task1Stk[0],
             TASK_STK_SIZE,
             (void *)0,
             OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
```

```
    OSTaskCreateExt(Task2,
                    (void *)0,
                    &Task2Stk[TASK_STK_SIZE - 1],
                    TASK_2_PRIO,
                    TASK_2_ID,
                    &Task2Stk[0],
                    TASK_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

    OSTaskCreateExt(Task3,
                    (void *)0,
                    &Task3Stk[TASK_STK_SIZE - 1],
                    TASK_3_PRIO,
                    TASK_3_ID,
                    &Task3Stk[0],
                    TASK_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

    OSTaskCreateExt(Task4,
                    (void *)0,
                    &Task4Stk[TASK_STK_SIZE-1],
                    TASK_4_PRIO,
                    TASK_4_ID,
                    &Task4Stk[0],
                    TASK_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

    OSTaskCreateExt(Task5,
                    (void *)0,
                    &Task5Stk[TASK_STK_SIZE-1],
                    TASK_5_PRIO,
                    TASK_5_ID,
                    &Task5Stk[0],
                    TASK_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
}
//////////////////////////////////////////////////////จบ มีไว้สร้าง task ////////////////////////////////////////
//////////////////////// main user ////////////////////////////////////////////////////////////////////////////
void  Task1 (void *pdata)
{
/////////////////////////// ผู้ใช้กำหนดตัวแปล ////////////////////////////////////////////////////

        pdata=pdata;


/////////////////////////////////// ผู้ใช้กำหนด โปรแกรม ////////////////////////////////////////////////
        for (;;)
        {

        }
}
/////////////////////////////////////////// จบ main user ////////////////////////////////////////////

/////////////////////////////////////////// task ต่างๆที่กล่าวมาข้างต้น ////////////////////
```

*ทาสก์ดีซีมอเตอร์*

*ทาสก์สเต็ปมอเตอร์*

*ทาสก์อินฟราเรดเซนเซอร์*

*ทาสก์บัมเซนเซอร์*

////////////////////////////////////////// จบ task ต่างๆที่กล่าวมาข้างต้น //////////////////////

********** *จบโครงสร้าง uCOS-II* **********

# บรรณานุกรรม

เอกสารอ้างอิง


[1] **Jean J. Labrosse : MicroC/OS-II The Real-Time Kernel** : Miller Freeman,1999

[2] **อภิเนตร อูนากูล : Object-Oriented Analysis and Design** : โครงการผลิตตำราของคณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2543

[3] **นุกูล กระจาย : การเขียนโปรแกรมในดอสและวินโดวส์ด้วยบอร์แลน C++5.0** : ซีเอ็ดยูเคชั่น , 2540

[4] **William Stalling,Ph.D. : Operating System Internals and Design Principles** : Prentice-Hall International, Inc. , 2001

[5] **เทพพิทักษ์ การุญบุญญานันท์ : กลเม็ดและเทคนิคในภาษา C/C++** : ซีเอ็ดยูเคชั่น , 2539

[6] **อภิชาติ ภู่พลับ : เริ่มต้นเขียนโปรแกรมติดต่อและควบคุมฮาร์ดแวร์ด้วย VB** : อินโฟเพรส , 2546

[7] **Muhammad Ali Mazidi, Janice Gillispie Mazidi : THE 80x86 IBM PC and Compatible Computers volume II** : Prentice-Hall International, Inc. , 1995


เว็บไซต์ที่เกี่ยวข้อง


[1] **http://www.ucos-ii.com/ : "Micro C/OS-II"**

[2] **http://www.datasheetlocator.com/ : "Datasheet"**

[3] **http://micromouse.cannock.ac.uk/ : "Micromouse Information Centre"**

[4] **http://thaiesf.org/ : "TESA Thai Embedded System Association"**

[5] **http://www.freertos/ : "Free Real Time OS"**

[6] **http://moss.csc.ncsu.edu/~mueller//rt/mindstrom/www.crynwr.com/lego-robotics/ : "Lego Mindstroms Internals"**