

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

หุ่นยนต์เตะฟุตบอล
SOCCER ROBOT



ปฏิญานีพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

**หุ่นยนต์เตะฟุตบอล
SOCCER ROBOT**



**ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549**

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชา วิศวกรรมเครื่องกล

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง หุ่นยนต์เตะฟุตบอล

SOCCER ROBOT

ผู้จัดทำ

1. นายวิทยา วงษ์กลาง รหัสประจำตัว 47015372
2. นายสามารถ มุ่งโตกลาง รหัสประจำตัว 47015379
3. นายไชยสาร ทรงประโคน รหัสประจำตัว 47015391



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หุ่นยนต์เตะฟุตบอล

นายวิทยา วงษ์กลาง 47015372
 นายสามารถ มุ่งโตกลาง 47015379
 นายไชยสาร ทรงประโคน 47015391
 ดร. ณัฐวุฒิ เตไพบัว อาจารย์ที่ปรึกษา

บทคัดย่อ

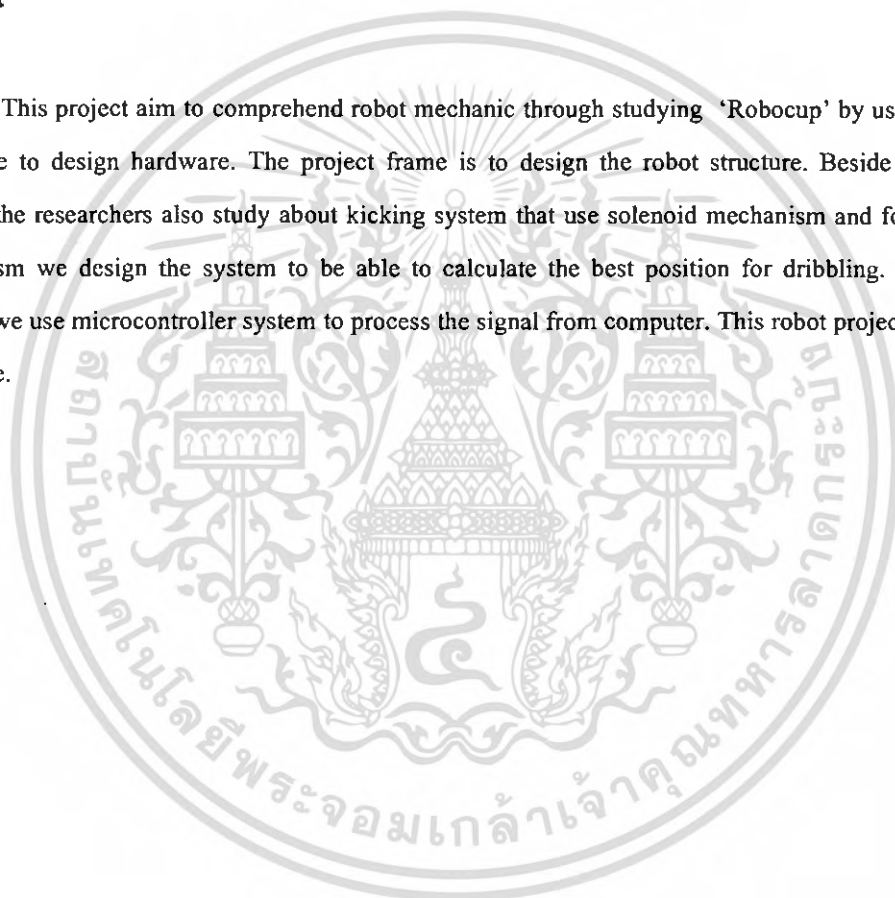
โครงการนี้เป็นการศึกษาถึงการออกแบบสร้างและควบคุมหุ่นยนต์เตะฟุตบอลให้ทำงานได้อย่างอัตโนมัติด้วยไมโครคอนโทรลเลอร์ ซึ่งในการศึกษาถึงการทำงานของหุ่นยนต์เตะฟุตบอลก็เป็นอีกทางหนึ่งที่จะทำให้เข้าใจถึงการควบคุมการทำงานของหุ่นยนต์ โดยในโครงการนี้การออกแบบโครงสร้างของหุ่นยนต์ตัวหุ่นยนต์ ล้อ และอุปกรณ์อื่นๆ ของหุ่นยนต์ โดยจุดประสงค์สำคัญของโครงการนี้ คือ การมุ่งเน้นให้หุ่นยนต์นั้นสามารถเคลื่อนที่เข้าจุดที่ต้องการได้ โดยเป็นไปแบบอัตโนมัติ เริ่มจากการรับภาพจากกล้องไปประมวลผลที่คอมพิวเตอร์ และ ส่งสัญญาณไปยังหุ่นยนต์ ในโครงการนี้จะเป็นการพัฒนาจากหุ่นยนต์ต้นแบบของหุ่นยนต์เตะฟุตบอลในปีการศึกษา 2548

SOCCER ROBOT

Mr.Wittaya Wongklang	47015372
Mr.Samart Mungtoklang	47015379
Mr.Chaisan Songprakhon	47015391
Dr.Nattawut Depaiwa	Advisor

Abstract

This project aim to comprehend robot mechanic through studying 'Robocup' by using mechanical discipline to design hardware. The project frame is to design the robot structure. Beside the movement system, the researchers also study about kicking system that use solenoid mechanism and for the dribbling mechanism we design the system to be able to calculate the best position for dribbling. For processing system, we use microcontroller system to process the signal from computer. This robot project is going to be prototype.



กิตติกรรมประกาศ

ปริญญาานิพนธ์ฉบับนี้จะไม่สามารถเสร็จลุล่วงไปได้ด้วยดี ถ้าหากปราศจากคำแนะนำ และความ
ร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน คือ ดร.ณัฐวุฒิ เตไพบวา ผศ.ดร.อุณหัต พินโสภณ อาจารย์ที่ปรึกษาปริญญา
นิพนธ์ ที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญาานิพนธ์นี้เสร็จลงได้ด้วยดี ที่ให้ความเอาใจใส่ และ
ช่วยเหลือมาโดยตลอด ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ท้ายที่สุด คณะผู้จัดทำขอขอบพระคุณ คณาจารย์และบุคลากร ของภาคเครื่องกล ที่ให้การปรึกษาและ
ความช่วยเหลือมาโดยตลอด ขอขอบพระคุณมา ณ ที่นี้

นายวิชา วงษ์กลาง
นายสามารถ มุ่งโตกลาง
นายไชสาร ทรงประโคน



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

	หน้าที่
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	VII
สารบัญรูปภาพ	VIII
บทที่ 1 บทนำ	1
1.1 ความสำคัญและที่มา	1
1.2 วัตถุประสงค์	1
1.3 ขอบเขตของโครงการ	1
1.4 ขั้นตอนการดำเนินโครงการ	2
บทที่ 2 อุปกรณ์และทฤษฎีที่ใช้ในหุ่นยนต์เตะฟุตบอล	3
2.1 ไมโครคอนโทรลเลอร์	3
2.2 โซลินอยด์ไฟฟ้า	12
2.3 มอเตอร์	13
2.4 อุปกรณ์รับส่งข้อมูล	15
บทที่ 3 การประยุกต์ใช้ไมโครคอนโทรลเลอร์ AVR ด้วยภาษาซี	17
3.1 โครงสร้างภาษาซี	17
3.2 ตัวแปรและค่าคงที่	18
3.3 คำดำเนินการในภาษาซี	19
3.4 ประโยคควบคุมในภาษาซี	21
3.5 การทำซ้ำ	22
3.6 Microcontroller เขียนโปรแกรมควบคุม AVR ผ่าน Serial Port	23
บทที่ 4 โปรแกรมวิซวลเบสิก	25
4.1 ขั้นตอนการสร้างโปรแกรมประยุกต์	25
4.2 คอนโทรลพื้นฐาน	25
4.3 คุณสมบัติร่วม	27
4.4 ทฤษฎีร่วม	30

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

	หน้าที่
4.5 อีเวนต์ร่วม	30
4.6 อีเวนต์ของฟอรัม	31
4.7 โพรซีเจอร์และฟังก์ชัน	31
4.8 ฟังก์ชัน	31
4.9 การประกาศตัวแปร	31
4.10 คำสั่งพื้นฐาน	33
4.11 คำสั่งเกี่ยวกับการจัดการรูปภาพ	35
4.12 คำสั่งเกี่ยวกับการติดต่อพอร์ตอนุกรม	35
4.13 คำสั่งเกี่ยวกับการติดต่อพอร์ต USB	51
บทที่ 5 การออกแบบหุ่นยนต์เตะฟุตบอล	55
5.1 การติดตั้งโครงสร้างของหุ่นยนต์	55
5.2 การติดตั้งล้อของตัวหุ่นยนต์	55
5.3 การติดตั้งตัวถังลูกบอลของตัวหุ่นยนต์	56
5.4 การติดตั้งตัวถังลูกบอลของตัวหุ่นยนต์	57
บทที่ 6 ระบบควบคุมของหุ่นยนต์เตะฟุตบอล	58
6.1 การออกแบบโปรแกรมการทำงาน	58
บทที่ 7 การออกแบบ โปรแกรมคำนวณพิกัดของตัวหุ่นยนต์เตะฟุตบอล	59
7.1 สาเหตุที่เลือกใช้โปรแกรมวิซวลเบสิกในการเขียนโปรแกรม	59
7.2 โปรแกรมคำนวณพิกัดในการเคลื่อนที่	59
7.3 หลักการทำงานของโปรแกรม	59
7.4 แผนผังลำดับการทำงานของโปรแกรม	62
7.5 โปรแกรมการสร้างโค้ด	63
บทที่ 8 รูปแบบและผลการทดลอง	64
8.1 รูปแบบการทดลอง	64
8.2 วิธีการทดลอง	64
8.3 ผลการทดลอง	67

สารบัญ (ต่อ)

	หน้าที่
บทที่ 9 วิเคราะห์และสรุปผลการทดลอง	68
9.1 วิเคราะห์ผลการทดลองและสรุปผลการทดลอง	68
9.2 แนวทางแก้ไข	69
บรรณานุกรม	70
ภาคผนวก ก วิซวลเบสิคซอร์สโค้ด	71
ภาคผนวก ข ซีซอร์สโค้ด	92
ภาคผนวก ค Data Sheet L298N	109



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

	หน้าที่
ตารางที่ 3-1 แสดงการประกาศตัวแปลต่างๆ	19
ตารางที่ 4-1 แสดงถึงคอนโทรลพื้นฐานของวิซวลเบสิกที่ใช้ในการสร้างโปรแกรม	25
ตารางที่ 4-2 แสดงถึงประเภทของข้อมูลพื้นฐานที่ใช้ในการสร้างตัวแปร	32
ตารางที่ 4-3 แสดงลักษณะการทำงานของบิตพาร์ตี	38
ตารางที่ 4-4 แสดงข้อมูลในแอดเดรส 0000:0411H ที่ใช้แจ้งจำนวนพอร์ตอนุกรม	43
ตารางที่ 7-1 แสดงตัวอักษรและสถานะของการเคลื่อนที่ของตัวหุ่นเมื่อเราส่งค่าออกไป	60



สารบัญรูปภาพ

	หน้าที่
รูปที่ 2-1 Block diagram แสดงหลักการทำงานของหุ่นยนต์เตะฟุตบอล	3
รูปที่ 2-2 แสดงถึงโครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์ AVR ATMEGAXX	5
รูปที่ 2-3 แสดง PORTA มีขนาด 8 Bit คือ PA0-PA7	6
รูปที่ 2-4 แสดง PORTB มีขนาด 8 Bit คือ PB0-PB7	6
รูปที่ 2-5 แสดง PORTC มีขนาด 8 Bit คือ PC0-PC7	7
รูปที่ 2-6 แสดง PORTD มีขนาด 8 Bit คือ PD0-PD7	7
รูปที่ 2-7 แสดง PORTE มีขนาด 8 Bit คือ PE0-PE7	7
รูปที่ 2-8 แสดง PORTF มีขนาด 8 Bit คือ PF0-PF7	8
รูปที่ 2-9 แสดงขั้วต่อ ET-CLCD	8
รูปที่ 2-10 แสดงวงจรขั้วต่อ ET-CLCD	8
รูปที่ 2-11 แสดงวงจรขั้วต่อ RS232	9
รูปที่ 2-12 แสดงวงจรและขั้วต่อ ISP LOAD	9
รูปที่ 2-13 แสดงโครงสร้างของบอร์ด ET-AVR ISP	11
รูปที่ 2-14 แสดงถึงทิศทางของสนามแม่เหล็กที่เกิดขึ้นในขดลวดที่มีกระแสไหล	12
รูปที่ 2-15 แสดงถึงการเพิ่มเหล็กอ่อนเข้ามาเพื่อเพิ่มความเข้มของสนามแม่เหล็ก	12
รูปที่ 2-16 แสดงวงจรของมอเตอร์เมื่อทิศทางการหมุนที่ต่างกัน	13
รูปที่ 2-17 แสดงสายสัญญาณ RS232 เพื่อใช้กับ ET-RF24G ในโหมดรับและส่ง	16
รูปที่ 3-1 แสดงการส่งข้อมูลของพอร์ตอนุกรม	23
รูปที่ 4-1 แสดงถึงการกำหนดคุณสมบัติของคอนโทรลที่ใช้ในการสร้างโปรแกรม	29
รูปที่ 4.2 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม	36
รูปที่ 4-3 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส	37
รูปที่ 4-4 การจัดขาของคอนเน็คเตอร์อนุกรมตามมาตรฐาน RS-232 ทั้งแบบ DB-9 และ DB-25	40
รูปที่ 4-5 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่าง ๆ	40
รูปที่ 5-1 แสดงการโครงสร้างของตัวหุ่นยนต์	55
รูปที่ 5-2 แสดงการติดตั้งตัวล้อขับของตัวหุ่นยนต์	56
รูปที่ 5-3 แสดงการติดตั้งตัวเลี้ยงบอลของตัวหุ่นยนต์	56
รูปที่ 5-4 แสดงการติดตั้งตัวยิงลูกบอลของตัวหุ่นยนต์	57
รูปที่ 6-1 แผนผังลำดับการทำงานของโปรแกรมภาษาซีในไมโครคอนโทรลเลอร์	58

สารบัญรูปภาพ(ต่อ)

	หน้าที่
รูปที่ 7-1 แสดงลักษณะการเดินหน้าและดอยหลังตรงของหุ่นยนต์	61
รูปที่ 7-2 แสดงลักษณะการหมุนเลี้ยวของหุ่นยนต์	61
รูปที่ 7-3 แสดงลักษณะการเดินหน้าเฉียงของหุ่นยนต์	61
รูปที่ 7-4 แสดงลักษณะการดอยหลังเฉียงของหุ่นยนต์	61
รูปที่ 7-5 แผนผังลำดับการทำงานของ โปรแกรมวิซวลเบสิกในคอมพิวเตอร์	62
รูปที่ 7-6 แสดงวิธีการคำนวณหามุมเพื่อสร้าง โค้ดต่างๆ	63
รูปที่ 8-1 แสดงแผนการเคลื่อนที่ของหุ่นยนต์ที่ใช้ในการทดลอง	64
รูปที่ 8-2 แสดงถึงลักษณะการเคลื่อนที่แบบเส้นตรงที่ใช้ในการทดลอง	65
รูปที่ 8-3 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ จากผลการทดลองในแกนที่ 1	65
รูปที่ 8-4 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ จากผลการทดลองในแกนที่ 2	66
รูปที่ 8-5 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ จากผลการทดลองในแกนที่ 3	66

บทที่ 1

บทนำ

1.1 ความสำคัญและความเป็นมา

ในปัจจุบันนี้หุ่นยนต์ที่สามารถปฏิบัติงานได้อย่างอัตโนมัติ ได้เริ่มเข้ามามีบทบาทอย่างมากในชีวิตประจำวัน ทั้งในด้านการอำนวยความสะดวกต่าง ๆ และมีประโยชน์อย่างมากในด้านอุตสาหกรรม โดยหุ่นยนต์ทั่วไปจะทำงานได้ก็จำเป็นต้องมีผู้เขียนโปรแกรมการทำงานให้ เมื่อเราต้องการศึกษาและทดลองเกี่ยวกับหุ่นยนต์ เราก็มักจำเป็นต้องมีความรู้ในด้านการเขียนโปรแกรมโดยในโครงการนี้ได้ใช้โปรแกรม Visual Basic ในการเขียนโปรแกรม

การรับค่าจากผู้ใช้ คอมพิวเตอร์นั้นเราสามารถรับค่าได้หลายทาง เช่น เมาส์ แป้นพิมพ์ และอื่นๆ โดยนำค่าเหล่านั้นมาวิเคราะห์อีกทีหนึ่งถึงความต้องการของผู้ใช้จะทำให้การทำงานมีค่าความแม่นยำมากยิ่งขึ้นในการใช้คอมพิวเตอร์แทนคนนั้นในงานที่ซ้ำๆ หรือต้องการความแม่นยำโดยคนเราไม่สามารถทำงานได้แต่คอมพิวเตอร์ทำแทนได้ ทั้งนี้ก็จำเป็นจะต้องมีอีกฝ่ายหนึ่งที่ต้องผลิตโปรแกรมขึ้น ในการผลิตโปรแกรมนี้นี้จะต้องให้ผู้ใช้งานสามารถใช้งานในลักษณะที่ง่ายแต่การใส่ และต้องมีความผิดพลาดน้อยที่สุดจึงจำเป็นจะต้องวิเคราะห์ให้ละเอียดด้วย

1.2 วัตถุประสงค์ของโครงการ

เพื่อศึกษาการทำงาน และ นำไปประยุกต์ใช้ ของหุ่นยนต์อัตโนมัติ โดยใช้ความรู้ทางด้าน การเขียนโปรแกรมประยุกต์ใช้งานด้วย โปรแกรมภาษาวิซวลเบสิก และการประยุกต์ใช้งานของไมโครคอนโทรลเลอร์ (Microcontroller) พร้อมด้วยการเขียนโปรแกรมติดต่อกับ พอร์ต USB และนำค่าจากการควบคุมมาประมวลผลและที่ส่งออกไปยัง พอร์ตอนุกรมเพื่อสั่งให้ตัวหุ่นทำงานตามที่ต้องการได้

1.3 ขอบเขตของโครงการ

เนื้อหาของปริิญาานิพนธ์นี้ จะเริ่มการศึกษากการทำงานของหุ่นยนต์โดยใช้คนบังคับ โดยใช้จอยรส์ติก ซึ่งส่งค่าผ่านพอร์ตยูเอสบี (USB port) และใช้สัญญาณภาพจากกล้องส่งมายังคอมพิวเตอร์แล้วคำนวณหาตำแหน่งพิกัดของตัวหุ่นจากภาพที่ได้รับจากกล้อง แล้วนำค่านั้นไปคำนวณการเคลื่อนที่ของตัวหุ่นแล้วส่งค่าออกไปที่พอร์ตอนุกรม (Serial port) และ Microcontroller จะรับค่าที่ส่งมาจากคอมพิวเตอร์แล้วตีความหมายของค่าที่ส่งมา แล้วสั่งให้มอเตอร์เคลื่อนที่ไปยังตำแหน่งที่ต้องการ และได้ทำการพัฒนาจนสามารถประมวลผลหาตำแหน่งได้อย่างแม่นยำและหุ่นยนต์สามารถเคลื่อนที่ได้โดยอัตโนมัติ

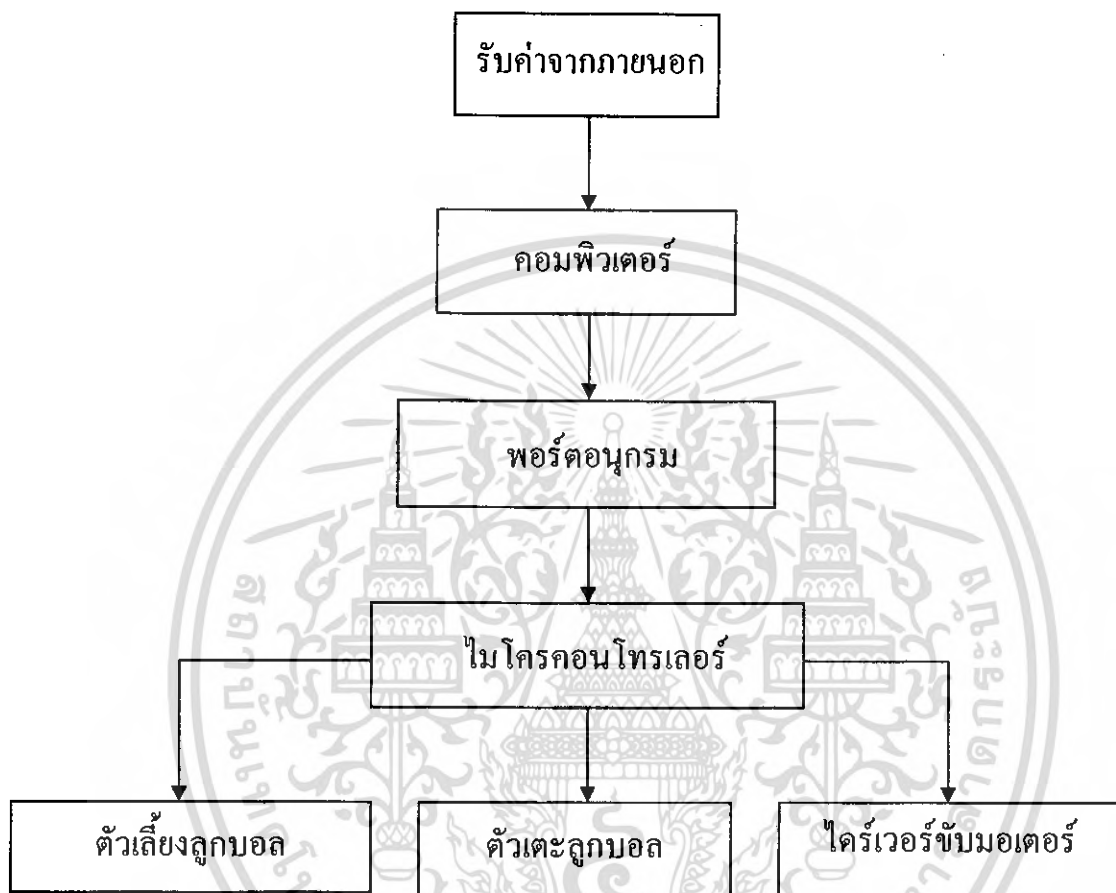
1.4 ขั้นตอนการดำเนินโครงการ

1. ศึกษาหลักการทำงานของ ไมโครคอนโทรลเลอร์
2. ออกแบบและสร้างตัวหุ่นยนต์เตะฟุตบอล
3. ศึกษาโปรแกรมวิชวลเบสิก และสร้างโปรแกรมที่ใช้ในการคำนวณหาตำแหน่งพิกัดจากสัญญาณภาพที่ส่งจากกล้องมายังคอมพิวเตอร์
4. ศึกษาโปรแกรมวิชวลเบสิก และสร้างโปรแกรมที่ใช้ในการคำนวณหาตำแหน่งพิกัดที่ได้จากการใช้จอยล์สติคเพื่อใช้ในโปรแกรมที่ควบคุมด้วยมือ
5. ศึกษาและสร้างไมโครคอนโทรลเลอร์ที่จะใช้ในการควบคุมโปรแกรมที่ใช้ในการกำหนดการเคลื่อนที่ของตัวหุ่นเมื่อได้รับค่าต่างๆ จากคอมพิวเตอร์
6. ติดตั้งและทำการทดลอง
7. สรุปผลการทดลองและข้อผิดพลาด
8. นำเสนอผลงาน



บทที่ 2

อุปกรณ์และทฤษฎีที่ใช้ในหุ่นยนต์เตะฟุตบอล



รูปที่ 2-1 Block diagram แสดงหลักการทำงานของหุ่นยนต์เตะฟุตบอล

ส่วนประกอบของตัวหุ่นยนต์เตะฟุตบอลประกอบไปด้วยส่วนหลักๆ ทั้ง 4 ส่วนดังต่อไปนี้

2.1 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์คือ ไอซีหน่วยประมวลผลกลาง (Central Processing Unit) ประเภทหนึ่งที่สามารถรับสัญญาณข้อมูลจากอุปกรณ์ภายนอก เพื่อนำมาคำนวณ ตัดสินใจ และส่งสัญญาณออกไปยังอุปกรณ์

ภายนอกเพื่อสั่งให้ทำงานตามโปรแกรม และข้อมูลที่ได้รับเข้ามา มีคุณสมบัติความเป็น Single Chip คือ สามารถทำงานได้โดยตัวไอซีเอง ไม่ต้องต่อวงจรเพิ่ม หรือต่อเพิ่มน้อยที่สุด ประโยชน์ที่ได้รับก็คือ การออกแบบวงจรทำได้ง่ายขึ้น ใช้อุปกรณ์ประกอบวงจรน้อยกว่าเดิม พื้นที่วงจรรวมมีขนาดเล็กลงเป็นอย่างมาก กินไฟเลี้ยงวงจรมีน้อยลง ทำให้ต้นทุนการผลิตและต้นทุนการทำงานลดต่ำลง นอกจากนี้ การบันทึกโปรแกรมที่เขียนลงในตัวไมโครคอนโทรลเลอร์ก็สามารถทำได้ง่ายและรวดเร็ว โดยเลือกรุ่นที่มีแฟลชเมโมรี่ในตัวที่สามารถเขียนโปรแกรมลงไปได้ ทำให้ไม่ต้องต่ออพรอมเพื่อเก็บโปรแกรม

ในโครงการนี้ใช้ไมโครคอนโทรลเลอร์ตระกูล AVR ของบริษัท ATMEL เบอร์ ATMEGA128 แบบ TQFP 64 PIN โดยสามารถต่อสัญญาณนาฬิกาได้ 16 MHz มีหน่วยความจำแบบ FLASH 128 kbyte หน่วยความจำโปรแกรม 4 kbyte และ RAM 4 kbyte

2.1.1 ความหมายของไมโครคอนโทรลเลอร์

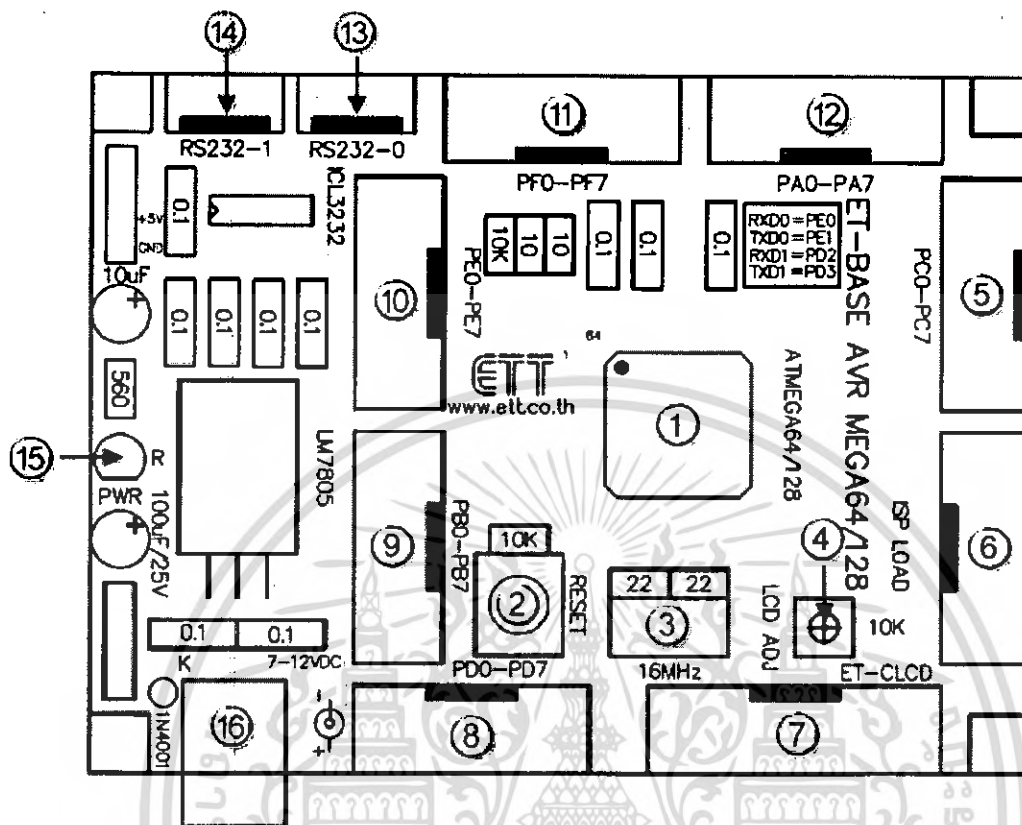
ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นชื่อของอุปกรณ์อิเล็กทรอนิกส์แบบหนึ่งซึ่งรวมเอาหน่วยประมวลผล หน่วยคำนวณทางคณิตศาสตร์และลอจิก วงจรรับสัญญาณอินพุต วงจรขับสัญญาณเอาต์พุต หน่วยความจำ วงจรกำเนิดสัญญาณนาฬิกาไว้ด้วยกัน ทำให้สามารถนำไปใช้งานแทนวงจรรีเลย์อิเล็กทรอนิกส์ที่ซับซ้อนได้เป็นอย่างดี ช่วยลดจำนวนอุปกรณ์และขนาดของระบบ ในขณะที่มีขีดความสามารถสูงขึ้น ภายใต้งบประมาณที่เหมาะสม ดังนั้น ไมโครคอนโทรลเลอร์จึงเป็นอุปกรณ์ที่ใช้ในการควบคุม โดยที่สามารถเขียนโปรแกรมเพื่อกำหนดรูปแบบการควบคุมได้อย่างอิสระ

2.1.2 คุณสมบัติของไมโครคอนโทรลเลอร์ตระกูล AVR อนุกรม ATMEGAxx

ไมโครคอนโทรลเลอร์ตระกูล AVR อนุกรม ATMEGAxx มีคุณสมบัติดังนี้คือ

- เป็นไมโครคอนโทรลเลอร์ที่ใช้แอมป์ขนาด 8 บิต
- ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนใหม่ได้เป็นแสนครั้ง
- หน่วยความจำข้อมูลพื้นฐานเป็นหน่วยความจำแบบแรม และมีหน่วยความจำแบบอพรอม
- ขาพอร์ตเป็นแบบสองทิศทาง สามารถใช้งานเป็นได้ทั้งอินพุต-เอาต์พุต จำนวน 6 พอร์ต
- มีวงจรสื่อสารอนุกรม
- มีไทมเมอร์/เคาน์เตอร์ขนาด 16 บิตจำนวน 2 ช่อง และ 8 บิต จำนวน 2 ช่อง
- สามารถรองรับแหล่งกำเนิดอินเทอร์รัป
- สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้
- มีวงจรถ่ายสัญญาณนาฬิกาอยู่ภายในชิป
- มีวงจรถ่ายอนุกรมแบบ SPI สำหรับในอนุกรม

2.1.3 โครงสร้างบอร์ดไมโครคอนโทรลเลอร์



รูปที่ 2-2 แสดงโครงสร้างบอร์ดไมโครคอนโทรลเลอร์ AVR ATMEGAxx

- หมายเลข 1 คือ เอ็มพิว เบอร์ ATMEGA128 ซึ่งเป็นเอ็มพิวตระกูล AVR จาก ATMEL
- หมายเลข 2 คือ สวิตช์รีเซต ใช้สำหรับรีเซตการทำงานของเอ็มพิว
- หมายเลข 3 คือ คริสตอลสัญญาณนาฬิกา 16 MHz
- หมายเลข 4 คือ ตัวต้านทานสำหรับปรับค่าความสว่างให้ LCD
- หมายเลข 5 คือ พอร์ต C มีขนาด 8 บิต คือ PC0-PC7
- หมายเลข 6 คือ พอร์ต ISP LOAD ใช้สำหรับการดาวน์โหลด Hex File ให้กับเอ็มพิว
- หมายเลข 7 คือ พอร์ต ET-CLCD สำหรับเชื่อมต่อกับ LCD ชนิด Character Type ซึ่งใช้การเชื่อมต่อแบบ 4 บิต
- หมายเลข 8 คือ พอร์ต D มีขนาด 8 บิต คือ PD0-PD7
- หมายเลข 9 คือ พอร์ต B มีขนาด 8 บิต คือ PB0-PB7
- หมายเลข 10 คือ พอร์ต E มีขนาด 8 บิต คือ PE0-PE7

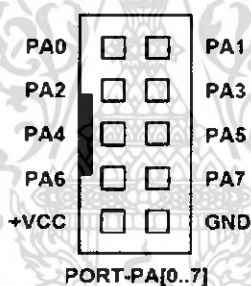
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- **หมายเลข 11** คือ พอร์ต F มีขนาด 8 บิต คือ PF0-PF7
- **หมายเลข 12** คือ พอร์ต A มีขนาด 8 บิต คือ PA0-PA7
- **หมายเลข 13** คือ ขั้วต่อ RS232 สำหรับใช้งานทั่วไป
- **หมายเลข 14** คือ ขั้วต่อ RS232 สำหรับใช้งานทั่วไป
- **หมายเลข 15** คือ LED Power ใช้สำหรับแสดงสถานะของแหล่งจ่ายไฟ +5VDC
- **หมายเลข 16** คือ ขั้วต่อแหล่งจ่ายไฟสำหรับเลี้ยงวงจรของบอร์ด

2.1.4 ขั้วต่อสัญญาณของพอร์ตในไมโครคอนโทรลเลอร์

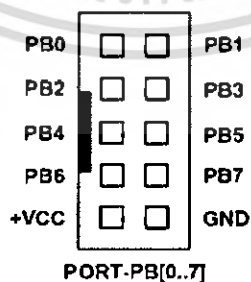
สำหรับขั้วต่อสัญญาณของพอร์ต I/O จาก MCU นั้นจะถูกออกแบบและจัดเตรียมไว้ผ่านทางขั้วต่อแบบ IDC-Header ขนาด 10 Pin (2X5) จำนวน 6 ชุด คือ PA,PB,PC,PD,PE,PF ตามลำดับ โดยที่ขั้วต่อสัญญาณแต่ละชุด จะประกอบไปด้วยสัญญาณของ I/O ที่เชื่อมต่อนมาจากขาสัญญาณของ MCU โดยตรงทั้งหมด โดยจุดเชื่อมต่อกับสัญญาณภายนอกบอร์ดมีดังนี้

- ขั้วต่อแหล่งจ่ายไฟสำหรับเลี้ยงวงจรของบอร์ด
- ขั้วต่อ PORTA มีขนาด 8 Bit คือ PA0-PA7



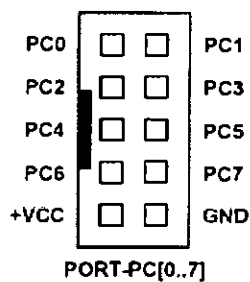
รูปที่ 2-3 แสดง PORTA มีขนาด 8 Bit คือ PA0-PA7

- ขั้วต่อ PORTB มีขนาด 8 Bit คือ PB0-PB7



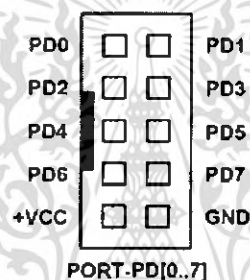
รูปที่ 2-4 แสดง PORTB มีขนาด 8 Bit คือ PB0-PB7

- ขั้วต่อ PORTC มีขนาด 8 Bit คือ PC0-PC7



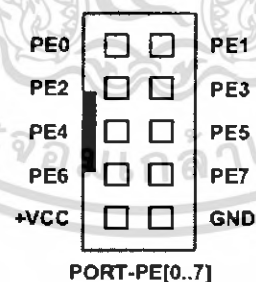
รูปที่ 2-5 แสดง PORTC มีขนาด 8 Bit คือ PC0-PC7

- ขั้วต่อ PORTD มีขนาด 8 Bit คือ PD0-PD7



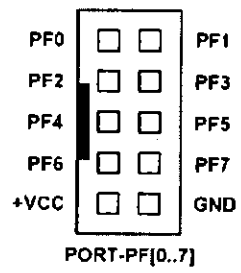
รูปที่ 2-6 แสดง PORTD มีขนาด 8 Bit คือ PD0-PD7

- ขั้วต่อ PORTE มีขนาด 8 Bit คือ PE0-PE7



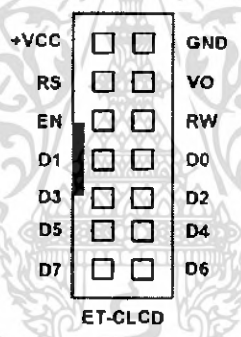
รูปที่ 2-7 แสดง PORTE มีขนาด 8 Bit คือ PE0-PE7

- ขั้วต่อ PORTF มีขนาด 8 Bit คือ PF0-PF7

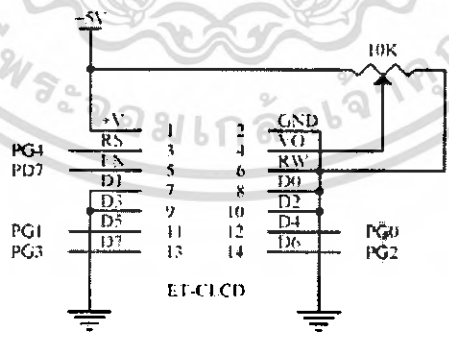


รูปที่ 2-8 แสดง PORTF มีขนาด 8 Bit คือ PF0-PF7

- ขั้วต่อ ET-CLCD สำหรับเชื่อมต่อกับ LCD ชนิด Character Type โดยการใช้การเชื่อมต่อแบบ 4 บิต โดยสัญญาณที่ใช้เชื่อมต่อกับ LCD จะเป็นสัญญาณจากพอร์ต PG และ PD (PD7) โดยในการเชื่อมต่อสายสัญญาณจากขั้วต่อของพอร์ต LCD ไปยังจอแสดงผล LCD นั้นให้ยึดชื่อขาสัญญาณเป็นจุดอ้างอิง โดยให้ต่อสัญญาณที่มีชื่อตรงกันเข้าด้วยกันให้ครบทั้ง 14 เส้น



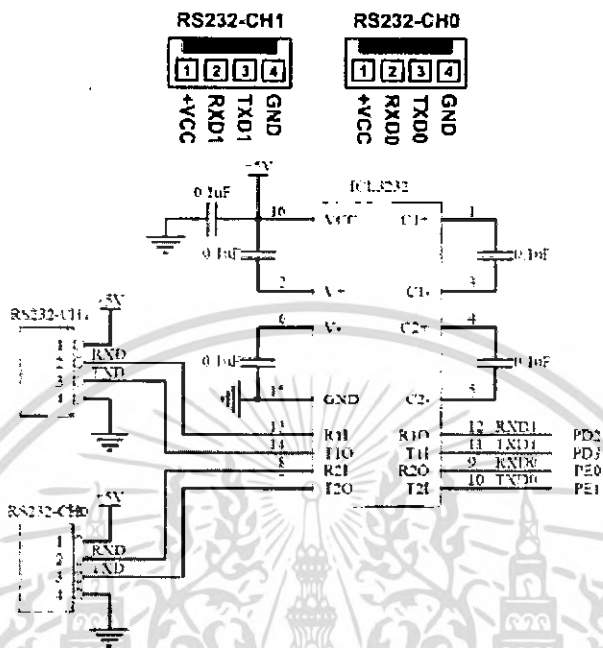
รูปที่ 2-9 แสดงขั้วต่อ ET-CLCD



รูปที่ 2-10 แสดงวงจรขั้วต่อ ET-CLCD

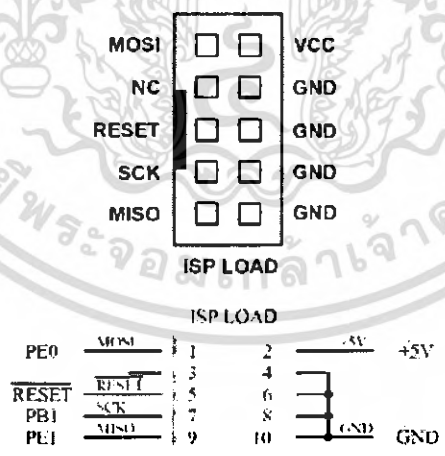
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ขั้วต่อ RS232 จำนวน 2 ช่อง โดยเชื่อมต่อกับสัญญาณ PE0(RXD0) และ PE1(TXD0) จำนวน 1 ช่อง ส่วนที่เหลืออีก 1 ช่อง จะต่อกับสัญญาณ PD2(RXD1) และ PD3(TXD1) เพื่อให้ผู้ใช้สามารถต่อทดลองการติดต่อสื่อสาร RS232



รูปที่ 2-11 แสดงวงจรขั้วต่อ RS232

- ขั้วต่อ ISP LOAD ใช้สำหรับดาวน์โหลด Hex File ให้กับ MCU



รูปที่ 2-12 แสดงวงจรและขั้วต่อ ISP LOAD

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1.5 การทำงานของพอร์ตในไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ AVR มีพอร์ตให้ใช้งานทั้งสิ้น 6 พอร์ต คือ พอร์ต A จนถึง พอร์ต F แต่ละพอร์ตมีขนาด 8 บิต เป็นพอร์ตแบบ 2 ทิศทาง กล่าวคือ สามารถเป็นได้ทั้งอินพุตสำหรับรับสัญญาณข้อมูลเข้า และเอาต์พุตสำหรับส่งสัญญาณข้อมูลออก ทุกพอร์ตของไมโครคอนโทรลเลอร์ AVR

2.1.6 การใช้งานเป็นพอร์ตอินพุต

เนื่องจากพอร์ตทั้งหมดของไมโครคอนโทรลเลอร์ AVR สามารถเป็นได้ทั้งอินพุตและเอาต์พุต ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะการทำงานให้แก่พอร์ตของไมโครคอนโทรลเลอร์ AVR

ในการกำหนดให้เป็นพอร์ตอินพุตต้องเริ่มต้นด้วยการเขียนข้อมูล “1” มาที่แต่ละบิตของพอร์ตที่ต้องการใช้งานเป็นอินพุต เพื่อหยุดการทำงานของเฟตที่ใช้ในการขับสัญญาณเอาต์พุตของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ตเชื่อมต่อเข้ากับวงจรพูลอัพภายในโดยตรง ส่งผลให้ขาพอร์ตนั้นมีลอจิกเป็น “1” สามารถรับสัญญาณลอจิก “0” จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ต แล้วรอให้จิจิพูมาอ่านค่าเข้าไป เมื่อเป็นเช่นนี้ อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ตอินพุตของไมโครคอนโทรลเลอร์ AVR ควรกำหนดให้ทำงานในสภาวะลอจิก “0” จะดีและสะดวกที่สุด ซึ่งในปัจจุบันอุปกรณ์อินพุตที่เชื่อมต่อกับไมโครคอนโทรลเลอร์แทบทั้งหมดทำงานที่ลอจิก “0” แล้ว

2.1.7 การใช้งานเป็นพอร์ตเอาต์พุต

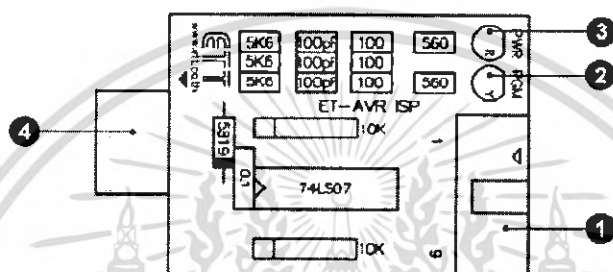
โดยปกติแล้ว ขาพอร์ตจะกำหนดให้มีลักษณะเป็นเอาต์พุตอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปได้อย่างง่ายดายและตรงไปตรงมา กล่าวคือ เมื่อต้องการส่งข้อมูล “0” ออกไปทางเอาต์พุตก็ให้เขียนข้อมูล “0” ไปยังวงจรแลตซ์ ซึ่งก็จะส่งต่อไปยังเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก “0” ขึ้นในทางตรงข้ามหากต้องการส่งข้อมูล “1” ออกไป ก็ให้เขียนข้อมูล “1” ไปยังวงจรแลตซ์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ตเชื่อมต่อกับวงจรพูลอัพภายในเกิดเป็นลอจิก “1” ที่ขาพอร์ตนั้น ซึ่งจะคล้ายกับการกำหนดให้เป็นขาอินพุตมาก เพียงแต่แตกต่างกันที่กระบวนการในการเคลื่อนย้ายข้อมูล โดยถ้าเป็นอินพุตจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาต์พุตจะไม่มีกรอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใด เว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาต์พุต

2.1.8 การดาวน์โหลด Hex File ให้กับ MCU

การดาวน์โหลด Hex File ให้กับ MCU นั้นจำเป็นจะต้องใช้ ET-AVR ISP หรือเครื่องโปรแกรมแบบ ISP อื่นๆ เช่น AVRISP ของ ATMEL เพื่อใช้ในการดาวน์โหลด Hex File ให้กับ MCU ตระกูล AVR ของ Atmel โดยใช้วิธีการแบบ Serial Programming ซึ่งการดาวน์โหลด Hex File ในกรณีที่ใช้ ET-AVR ISP จะ

กระทำผ่านทางพอร์ตขนานของคอมพิวเตอร์ โดยที่จะต้องใช้งานร่วมกับ ET-CAP10P ของอีทีที และ Software ที่ใช้ร่วมกับ ET-AVR ISP ก็คือ PonyProg2000 ซึ่ง PonyProg2000 เป็นโปรแกรม Download ข้อมูลแบบ HEX File ให้กับ CPU ตระกูล AVR โดยใช้วิธีการแบบ Serial Programming ซึ่งสามารถใช้งานกับบอร์ดตระกูล AVR ของ อีทีที ได้เป็นอย่างดี ซึ่งวิธีการใช้งานโปรแกรมโดยทั่วไปนั้น สามารถศึกษาได้จาก Help ของโปรแกรมได้เอง โดยในที่นี้จะขอแนะนำให้ทราบถึงวิธีการ Setup โปรแกรม PonyProg2000 เพื่อใช้งานกับบอร์ดตระกูล AVR ของ อีทีที ซึ่งสามารถใช้งานได้กับบอร์ดตระกูล AVR ทุกรุ่นของ อีทีที

2.1.9 โครงสร้างของบอร์ด ET-AVR ISP

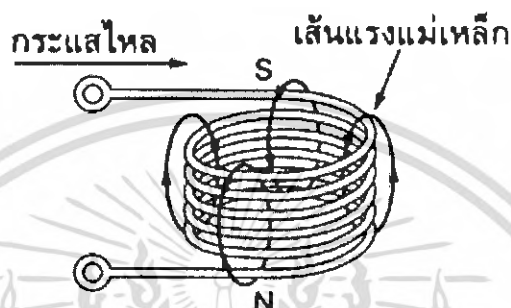


รูปที่ 2-13 แสดงโครงสร้างของบอร์ด ET-AVR ISP

- หมายเลข 1 คือ พอร์ตสำหรับเชื่อมต่อกับ ET-CAP10P ของอีทีที เพื่อ โปรแกรม Hex File ให้กับ MCU
- หมายเลข 2 คือ LED PGM (สีเขียว) แสดงสถานะของการ โปรแกรมหรือดาวน์โหลด Hex File ลง MCU
- หมายเลข 3 คือ LED PWR (สีแดง) แสดงสถานะของ ไฟเลี้ยงบอร์ด
- หมายเลข 4 คือ พอร์ตสำหรับเชื่อมต่อกับบอร์ด Target ซึ่งสามารถใช้โปรแกรม Hex File ให้กับ บอร์ด ET-AVR STAMP ATmega128 โดยเสียบบอร์ด ET-AVR ISP เข้าที่ พอร์ต ET-PSPI ซึ่งมีการ จัดเรียงขาสัญญาณดังรูป

2.2 โซลินอยด์ไฟฟ้า

โซลินอยด์ไฟฟ้ามีหลักการทำงานดังนี้ คือ เมื่อมีกระแสไฟฟ้าไหลในขดลวดตัวนำใดๆก็ตาม จะก่อให้เกิดสนามแม่เหล็กขึ้นรอบๆตัวนำนั้น และหากนำเส้นลวดมาขดเป็นวงๆ หลายๆวง ก็จะเกิดลักษณะของขดลวดขึ้น โดยสนามแม่เหล็กที่เกิดจากขดลวดแต่ละขดจะอยู่ในทิศทางเสริมกัน และก่อกำเนิดเป็นเส้นแรงของสนามแม่เหล็กถาวรแท่งหนึ่ง ซึ่งพร้อมที่จะดูดสารแม่เหล็กทันที แต่เนื่องจากสภาพรอบๆ ขดลวดอาจเป็นอากาศ เส้นแรงแม่เหล็กจึงไม่เข้มข้นมากนัก



รูปที่ 2-14 แสดงถึงทิศทางของสนามแม่เหล็กที่เกิดขึ้นในขดลวดที่มีกระแสไหล

เพื่อไม่ให้สนามแม่เหล็กที่เกิดขึ้นกระจัดกระจาย จึงใส่แกนเหล็กอ่อนรูปตัวซีล้อมรอบๆขดลวดไว้ เพื่อให้สนามแม่เหล็กที่เกิดขึ้นกระจัดกระจายออกไป และเพื่อให้สนามแม่เหล็กมีความเข้มข้นมากขึ้น จากจุดนี้ หากนำเอาแกนกระทุ้ง (Plunger) มาใส่เข้าไปตรงกลางวงของขดลวดในตำแหน่งที่ 1 แกนกระทุ้งจะถูกดูด ให้เหล็กเข้ามาจนสนิทในตำแหน่งที่ 2 ความสัมพันธ์ของแรงกับระยะช่วงชักของโซลินอยด์ ในช่วงชักไกลๆจะมีแรงน้อยมาก แต่ในทางตรงกันข้ามช่วงชักที่มีระยะใกล้ๆก็จะมีแรงมากขึ้นเป็นทวีคูณ



รูปที่ 2-15 แสดงถึงการเพิ่มเหล็กอ่อนเข้ามาเพื่อเพิ่มความเข้มของสนามแม่เหล็ก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.2.1 ประเภทของโซลินอยด์ไฟฟ้า

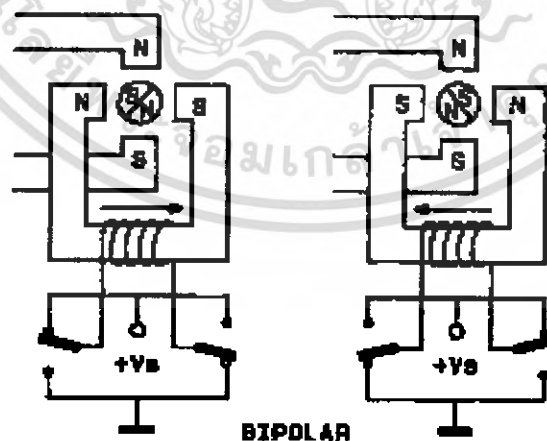
โซลินอยด์ไฟฟ้าสามารถแบ่งได้เป็น 2 ประเภท คือ โซลินอยด์ที่ใช้ไฟฟ้ากระแสตรง และโซลินอยด์ที่ใช้ไฟฟ้ากระแสสลับ ข้อแตกต่างระหว่างโซลินอยด์ที่ใช้ไฟฟ้ากระแสตรง และโซลินอยด์ที่ใช้ไฟฟ้ากระแสสลับ คือ โซลินอยด์ที่ใช้ไฟกระแสตรงจะก่อให้เกิดกระแสที่ไหลในขดลวดค่อนข้างคงที่ไม่เปลี่ยนแปลง ไม่ว่าแกนกระทุ้งจะอยู่ในตำแหน่งใดก็ตาม แต่โซลินอยด์ที่ใช้ไฟกระแสสลับจะมีกระแสในขณะที่แกนกระทุ้งอยู่นอกขดลวดค่อนข้างสูง และเมื่อแกนกระทุ้งถูกดูดเข้ามาจนสุดขดลวด กระแสจะลดต่ำลง ด้วยเหตุนี้เองที่ทำให้ต้องระมัดระวังทำให้เกิดการกระทุ้งในโซลินอยด์ไฟสลับ เพราะจะทำให้เกิดกระแสมากๆไหลค้างอยู่ ทำให้ขดลวดร้อนขึ้น และ อาจจะไหม้เสียหายได้

2.2.2 การเลือกใช้โซลินอยด์ไฟฟ้า

การเลือกใช้โซลินอยด์ไฟฟ้าควรพิจารณาถึงองค์ประกอบต่างๆดังนี้ คือ

1. แรงดันใช้งานซึ่งไม่ว่าจะเป็น ไฟฟ้ากระแสตรง หรือไฟฟ้ากระแสสลับก็ตาม จำเป็นจะต้องดูความถี่ใช้งานให้ตรงตามความต้องการด้วย ซึ่งในการออกแบบและสร้างเครื่องเจาะพลาสติกเพื่อใช้เป็นแบบสำหรับทอพรมนี้ใช้โซลินอยด์ที่มีความถี่ 55 เฮิรซ์
2. ช่วงชักในการใช้งาน (Operating Stroke) ของโซลินอยด์จะต้องเคลื่อนที่เป็นระยะทางเท่าใด มักจะกำหนดเป็นมิลลิเมตร และการสร้างตัวหุ้มนี้อาจใช้โซลินอยด์ขนาดเล็กและมีแรงขับที่พอเหมาะด้วย
- 3 .เป็นการใช้งานอย่างต่อเนื่องหรือไม่ ซึ่งการใช้งานอย่างต่อเนื่องในที่นี้ หมายถึง การใส่แรงดันไฟเข้าขดลวดค้างไว้โดยขดลวดไม่ไหม้ หรือเป็นการใส่แรงดันแบบเป็นจังหวะๆ เพื่อใช้ในการเตะลูกฟุตบอลจะต้องเลือกคือต้องเล็กและมีแรงในการเตะที่แรงด้วย

2.3 มอเตอร์



รูปที่ 2-16 แสดงวงจรของมอเตอร์เมื่อทิศทางการหมุนที่ต่างกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.1 ประเภทของมอเตอร์

มอเตอร์ที่นิยมนำมาใช้ในงานคอนโทรลปัจจุบันจะแบ่งออกได้เป็น 3 ประเภทคือ หนึ่ง สเต็ปเปอร์มอเตอร์ (Stepper Motor) สอง เซอร์โวมอเตอร์ (Servo Motor) และสาม มอเตอร์ไฟฟ้ากระแสตรง (DC Motor) ซึ่งจะกล่าวถึงดังต่อไปนี้

2.3.1.1 สเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์ เป็นมอเตอร์ที่หมุนตามจำนวนองศาหรือตามจำนวนรอบที่ต้องการ ทั้งนี้ความละเอียดของการหมุนจะขึ้นอยู่กับจำนวนองศาต่อหนึ่งจังหวะการหมุน เนื่องจากการหมุนของมอเตอร์ชนิดนี้มีการหมุนเป็นจังหวะ จึงเรียกว่า สเต็ปเปอร์มอเตอร์ และที่นิยมใช้จะเป็นชนิดที่เรียกว่าสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์ (Uni-polar Stepper Motor)

2.3.1.1 เซอร์โวมอเตอร์

เซอร์โวมอเตอร์เป็นมอเตอร์ที่ประกอบไปด้วยชุดเกียร์ (Gear) มอเตอร์กระแสตรง (DC Motor) และส่วนควบคุมอิเล็กทรอนิกส์ ที่รวมอยู่ในตัวมอเตอร์ เซอร์โวมอเตอร์จะทำงานได้ด้วยสัญญาณพัลส์ที่มีความกว้างอยู่ระหว่าง 1 มิลลิวินาที ถึง 2 มิลลิวินาที (พัลส์บวกหรือลوجิก 1) โดยส่งพัลส์ดังกล่าวห่างกันเป็นระยะเวลา 20 มิลลิวินาที (พัลส์ลบหรือ ลอจิก 0) การส่งสัญญาณพัลส์ดังกล่าวมีผลให้มอเตอร์หมุน โดยทิศทางการหมุนนั้นจะขึ้นอยู่กับความกว้างของพัลส์บวก

2.3.1.3 มอเตอร์ไฟฟ้ากระแสตรง

มอเตอร์ไฟฟ้ากระแสตรง เป็นมอเตอร์ที่ขับเคลื่อนด้วยไฟฟ้ากระแสตรง โดยควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรงได้ด้วยการจ่ายกระแสไฟฟ้าเป็นช่วงเวลา แต่ไม่สามารถที่จะควบคุมทิศทางการหมุนของมอเตอร์ได้ การที่จะควบคุมทิศทางการหมุนของมอเตอร์จะต้องใช้ไอซีมาช่วยในการขับเคลื่อน โดยใช้หลักการขับเคลื่อนแบบเฮซบริดจ์ (H-Bridge)

2.3.2 การเลือกใช้อุปกรณ์

จะเห็นได้ว่ามอเตอร์ทั้ง 3 ประเภทนี้ ต่างก็มีข้อดีข้อเสียต่างกันออกไป การที่เราจะตัดสินใจเลือกใช้อุปกรณ์แบบใดนั้นคงต้องพิจารณาจากจุดประสงค์หลักในการใช้งาน สำหรับในหุ่นยนต์แต่ละฟุตบอลนี้จะเลือกใช้อุปกรณ์ชนิดมอเตอร์ไฟฟ้ากระแสตรงเพราะต้องการความเร็วและการหมุนที่ราบเรียบ

2.4 อุปกรณ์รับและส่งข้อมูล

ในการติดต่อสื่อสารและรับส่งข้อมูลกันระหว่างหุ่นยนต์กับคอมพิวเตอร์จะต้องมีสื่อกลางในการรับและส่งข้อมูลซึ่งกันและกัน ดังนั้นจึงต้องมีอุปกรณ์ที่ช่วยในการรับและส่งข้อมูล อุปกรณ์รับและส่งข้อมูลก็จะแบ่งออกเป็น 2 ประเภทใหญ่ ๆ ได้ดังนี้ คือ แบบใช้สายสัญญาณในการรับส่งข้อมูล และแบบไม่ใช้สายสัญญาณ ในการรับส่งข้อมูลหรือก็คือแบบไร้สายนั่นเองในหุ่นยนต์จะใช้การรับส่งข้อมูลแบบไร้สาย ดังนั้นจะพูดถึงเฉพาะแบบไร้สายเท่านั้น

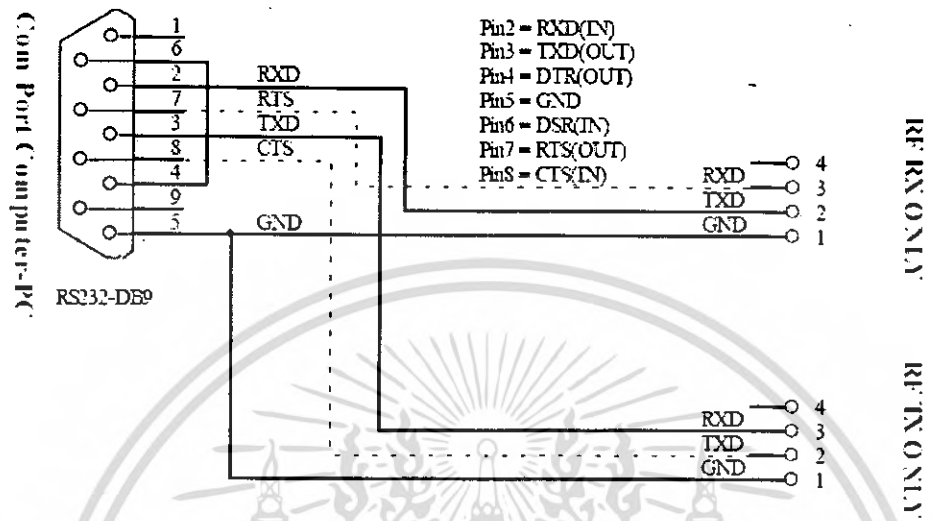
2.4.1 ชุดรับส่งข้อมูล ET-RF24G V1.0

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

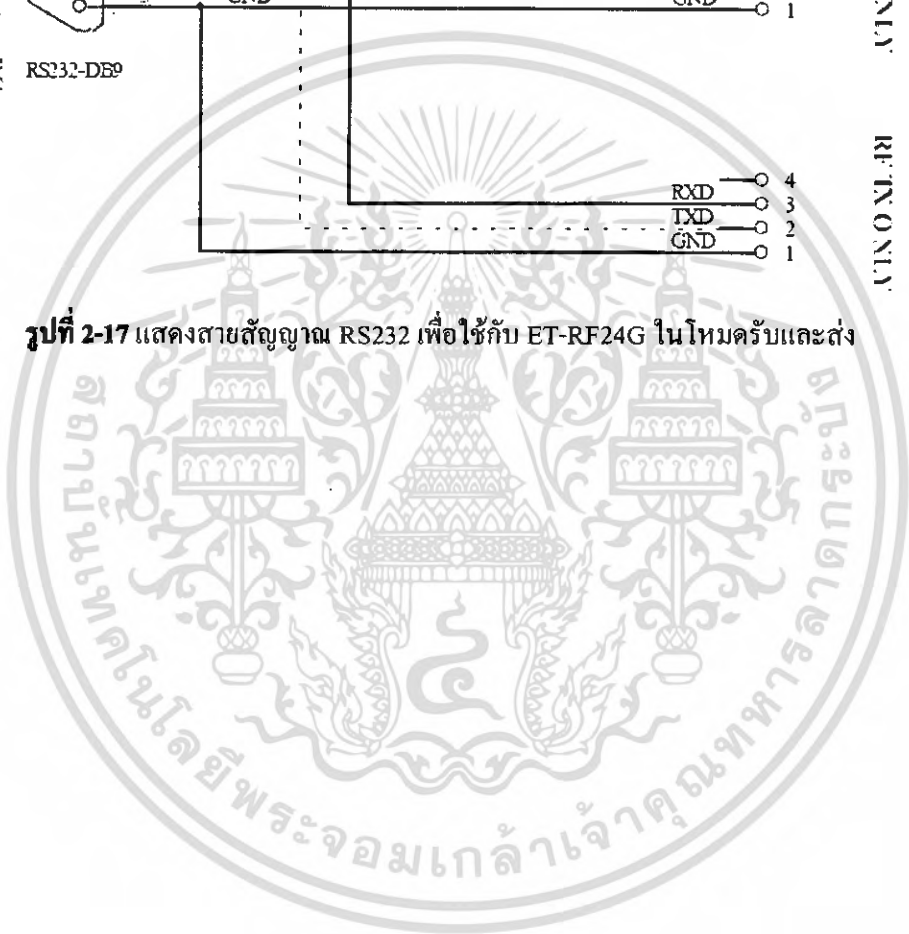
ซึ่งจะเห็นได้ว่าชุดแปลงสัญญาณ ET-RF24G V1.0 นั้น สามารถนำไปต่อใช้งานร่วมกับพอร์ตสื่อสารอนุกรม แบบ RS232 เพื่อใช้งานในลักษณะของการสื่อสารอนุกรมแบบไร้สาย (Wireless Transceiver) ได้โดยตรง โดยจะมีข้อดีกว่า คือ สามารถรับส่งข้อมูลกันได้ในระยะทางที่ไกลกว่า RS232 หลายเท่าตัว และประการสำคัญ คือ ไม่จำเป็นต้องใช้สายสัญญาณที่เป็นตัวนำสัญญาณทางไฟฟ้าในการสื่อสารข้อมูลกันทำให้สามารถเปลี่ยนแปลงหรือเคลื่อนย้ายจุดรับส่งข้อมูลได้ตลอดเวลา

ซึ่งถ้าเป็นการรับส่งข้อมูลด้วยระบบ RS232 แบบที่ใช้สายสัญญาณนั้นจะเกิดความยุ่งยากในการติดตั้งสายสัญญาณเป็นอย่างมากแต่อย่างไรก็ตามการรับส่งข้อมูลโดยใช้อากาศเป็นตัวกลางในการสื่อสารนั้น ก็มีข้อจำกัดบางประการเหมือนกัน โดยเฉพาะอย่างยิ่ง เรื่องความน่าเชื่อถือของข้อมูลที่รับส่งกัน ซึ่งมีโอกาสผิดพลาดหรือสูญหายได้เหมือนกัน เนื่องจากในการลำเลียงข้อมูลนั้นไม่ได้ใช้สายสัญญาณเป็นตัวกลางในการรับส่งข้อมูล แต่ใช้อากาศเป็นตัวกลางในการรับส่งข้อมูลแทน ซึ่งมีโอกาสที่ข้อมูลจะเกิดการรบกวนจากสัญญาณอื่นที่มีย่านความถี่ใกล้เคียงกันแล้วทำให้ข้อมูลผิดเพี้ยนไปได้บ้างเหมือนกัน ซึ่งระบบการจัดการข้อมูลของเครื่อง ET-RF24G V1.0 นั้น มีระบบการเข้ารหัสและถอดรหัสข้อมูลที่มีความน่าเชื่อถืออยู่ในเกณฑ์ที่จัดว่าดี โดยข้อมูลแต่ละ Byte ที่มีการรับส่งกันนั้น จะมีการตรวจสอบความถูกต้องของข้อมูลให้ด้วยแล้ว โดยข้อมูลที่รับได้จากด้าน RF นั้นรับประกันได้ว่าเป็นข้อมูลที่มีความถูกต้องแน่นอน แต่อย่างไรก็ตามการรับส่งข้อมูลนั้นมีโอกาสผิดพลาดในเรื่องของการสูญหายของข้อมูลบ้างเหมือนกัน เนื่องจากกลไกในการรับส่งข้อมูลของเครื่อง ET-RF24G V1.0 นั้น จะมีการตรวจสอบข้อมูลทุก Byte ที่รับได้จาก RF เสมอ ซึ่งถ้าพบว่ามี ความผิดพลาดเกิดขึ้นจะทิ้งข้อมูล Byte นั้นไป ซึ่งผู้ใช้ควรมีกลไกในการตรวจสอบข้อมูลที่รับส่งกันว่า

ครบถ้วนหรือไม่ด้วย ซึ่งหากพบว่ามี การสูญหายของข้อมูลเกิดขึ้นก็ให้ร้องขอให้มีการส่งข้อมูลนั้นซ้ำ ใหม่อีก ครั้งหนึ่ง ก็จะสามารถแก้ไขปัญหาดังกล่าวได้



รูปที่ 2-17 แสดงสายสัญญาณ RS232 เพื่อใช้กับ ET-RF24G ในโหมดรับและส่ง



บทที่ 3

การประยุกต์ใช้ไมโครคอนโทรลเลอร์ AVR ด้วยภาษาซี

การเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ด้วยภาษาแอสเซมบลีนั้นมีโครงสร้างของโปรแกรมจะคล้าย การควบคุมทิศทางของโปรแกรมก็เข้าใจได้ยาก แม้แต่ผู้ที่เขียนโปรแกรมเองเมื่อเวลาผ่านไปแล้วกลับมาดูโปรแกรมใหม่ก็อาจดูงานที่ตัวเองเขียนได้ไม่เข้าใจ แม้ว่าจะเขียนคำอธิบายไว้ทุกบรรทัดก็ตาม ในอดีตการเขียนโปรแกรมด้วยภาษาซีสำหรับไมโครคอนโทรลเลอร์นั้นเมื่อแปลออกมาเป็นรหัสของภาษาเครื่องจะได้รหัสที่มีขนาดใหญ่กว่าการเขียนด้วยภาษาแอสเซมบลีมาก แต่ปัจจุบันคอมพิวเตอร์ส่วนใหญ่จะแปลภาษาออกมาได้รหัสที่มีขนาดเล็กเกือบใกล้เคียงกับภาษาแอสเซมบลี การเขียนโปรแกรมด้วยภาษาซีสามารถนำไปใช้กับงานที่ซับซ้อนได้ เช่น งานควบคุมแบบป้อนกลับ (Close Loop Control) การใช้ไมโครคอนโทรลเลอร์ควบคุมเครื่องมือวัดต่างๆ (Microcontroller Based Instrumentation) และการใช้ไมโครคอนโทรลเลอร์ในการควบคุมหุ่นยนต์ เป็นต้น

3.1 โครงสร้างภาษาซี

ด้วยภาษาซีเป็นภาษาที่สามารถเขียนโปรแกรมเป็นแบบโครงสร้างได้ โดยโปรแกรมจะแบ่งการทำงานต่างๆ ออกเป็นกลุ่ม ๆ หรือ ฟังก์ชัน โดยฟังก์ชันเหล่านั้นสามารถเรียกขึ้นมาใช้ใหม่ได้ ในการเขียนโปรแกรมจะต้องระบุไว้ว่าในโปรแกรมนั้นมีฟังก์ชันใดให้ใช้บ้าง แต่ทุกโปรแกรมจะต้องมีฟังก์ชันหลักที่ชื่อว่า main() เสมอ พิจารณาตัวอย่างโครงสร้างของโปรแกรมต่อไปนี้

```
#include<mega128.h>           /*Preprocessor*/
#include<stdio.h>
void func1 (void);           /*Por to type*/
int func2 (int x);
void main ()                 /* ฟังก์ชันหลัก*/
{
    int a;                   /*ประกาศตัวแปร*/
    PORTA = 0xFF;
    func1();                 /*เรียกใช้ฟังก์ชัน*/
    a = func2 (4);           /*เรียกใช้ฟังก์ชันที่มีการส่งค่า*/
    PORTA = a;
}
```

72174

```

void func1(void)                /*ฟังก์ชันที่ไม่มีการส่งค่า*/
{
    *****
    *****
}

int func2(intx)                /*ฟังก์ชันที่มีการส่งค่า*/

{
    return(x*2);
}

```

จากโปรแกรมพบว่า ส่วนประกาศโปรโตไทป์จะบอกว่า โปรแกรมนี้มีฟังก์ชันชื่อ func1 ให้ใช้งาน และฟังก์ชันนี้จะทำงานเป็น โปรแกรมย่อยเพราะมี void นำหน้า และมีฟังก์ชันชื่อ func2(int x) ซึ่งรับค่าเข้าไปผ่านทางตัวแปร x และคืนค่าออกมาเป็นจำนวนเต็ม

3.2 ตัวแปรและค่าคงที่

การใช้งานตัวแปรและค่าคงที่ต่างๆ จะต้องมีการประกาศชื่อตัวแปรขึ้นมาเสียก่อนเมื่อมีการคอมไพล์ โปรแกรมตัวคอมไพเลอร์จะเตรียมพื้นที่ในหน่วยความจำแรมเอาไว้สำหรับเก็บตัวแปรและค่าคงที่เหล่านั้นใน การประกาศตัวแปรสามารถทำได้ดังนี้

ประเภทของข้อมูล ชื่อตัวแปร[.....];

การประกาศตัวแปรจะต้องเริ่มด้วยชื่อประเภทของข้อมูล และตามด้วยชื่อตัวแปรโดยจะประกาศครั้ง ละกี่ตัวก็ได้ ส่วนชื่อของตัวแปรนั้นจะซ้ำกันไม่ได้และต้องไม่ซ้ำกับชื่อของคำสงวน (Keywords) ของ คอมไพเลอร์ตัวนั้นๆ สำหรับประเภทของข้อมูลในการประกาศตัวแปรดังนี้

ประเภทของข้อมูล	ขนาด (บิต)	ค่าที่เก็บได้
bit	1	0 ถึง 1
char	8	-128 ถึง 127
unsigned char	8	0 ถึง 255
int	16	-32768 ถึง 32767
unsigned int	16	0 ถึง 65535
long	32	-2147483648 ถึง -2147483647
unsigned long	32	0 ถึง 4294967295
float	32	-1.17549e-38 ถึง 3.402823 e+38

ตารางที่ 3-1 แสดงการประกาศตัวแปรต่างๆ

สำหรับค่าสงวนเป็นคำที่คอมไพเลอร์รู้จักและจะถูกใช้งานเฉพาะ เราไม่สามารถนำมาตั้งเป็นชื่อตัวแปรและฟังก์ชันได้ คำสงวนของโปรแกรมเป็นดังต่อไปนี้

at	idata	sfr
alien	interrupt	sfr16
bdata	large	small
bit	pdata	_task_
code	_priority_	using
compact	reentrant	Xdata
data	sbit	

3.3 ตัวดำเนินการในภาษาซี

ตัวดำเนินการจะเป็นตัวที่ใช้กระทำกับตัวแปร ค่าคงที่ต่างๆ ให้รวมเป็นค่าเดียวกัน โดยอาจกระทำทางคณิตศาสตร์หรือกระทำทางลอจิกก็ได้ ในการเขียน โปรแกรมด้วยภาษานั้น ตัวดำเนินการจะแบ่งออกเป็นสองกลุ่มใหญ่ๆ คือตัวดำเนินการที่กระทำกับตัวถูกกระทำตัวเดียว (Single Operand Operators) และตัวดำเนินการที่กระทำกับตัวถูกกระทำสองตัว (Two Operands Operators)

3.3.1 ตัวดำเนินการที่กระทำกับตัวถูกกระทำตัวเดียว

ตัวดำเนินการประเภทนี้จะกระทำกับตัวถูกกระทำเพียงตัวเดียว ประกอบด้วยตัวดำเนินการต่างๆ ดังนี้

- ลบ (negate)

~	กลับค่าลอจิกของบิตข้อมูล (bit wise complement)
!	กลับค่าทางลอจิก (logical complement)
++	เพิ่มค่าขึ้นหนึ่งค่า (increment)
--	ลดค่าลงหนึ่งค่า (decrement)
*	ตัวดำเนินการทางพอยน์เตอร์
&	ตำแหน่งหน่วยความจำของตัวแปร

3.3.2 ตัวดำเนินการที่กระทำกับตัวถูกกระทำสองตัว

ตัวดำเนินการแต่ละชนิดนี้จะกระทำกับตัวถูกกระทำสองตัว ถ้าหากมีตัวถูกกระทำหลายๆ ตัวสามารถนำมาเขียนรวมกันเป็นประโยคได้ ซึ่งประกอบไปด้วยตัวดำเนินการที่ใช้กำหนดค่า ตัวดำเนินการทดสอบค่าซึ่งจะให้ผลลัพธ์เป็นค่าทางลอจิก (จริง, เท็จ) ตัวดำเนินการทางคณิตศาสตร์ และตัวดำเนินการทางลอจิกดังต่อไปนี้

=	กำหนดค่าในประโยค (assignment)
+	บวก
-	ลบ
*	คูณ
/	หาร (division)
%	หารแบบบวก (modulo)
&&	การแอนด์ (logical AND)
	การออร์ (logical OR)
&	การแอนด์แบบบิตต่อบิต (bit wise AND)
	การออร์แบบบิตต่อบิต (bit wise OR)
^	การเอ็กคลูซีฟออร์แบบบิตต่อบิต (bit wise exclusive OR)
<<	เลื่อนบิตไปทางซ้าย
>>	เลื่อนบิตไปทางขวา
==	ทดสอบว่าเท่ากันหรือไม่
!=	ทดสอบว่าไม่เท่ากันหรือไม่
>	ทดสอบว่ามากกว่าหรือไม่
<	ทดสอบว่าน้อยกว่าหรือไม่
>=	ทดสอบว่ามากกว่าหรือเท่ากันหรือไม่
<=	ทดสอบว่าน้อยกว่าหรือเท่ากันหรือไม่

3.4 ประโยคควบคุมในภาษาซี

การทำงานของโปรแกรมนั้นจะทำคำสั่งแต่ละคำสั่งเรียงลำดับกันไป และเราสามารถให้โปรแกรมตัดสินใจในการเลือกทำได้ หรือให้ทำงานใดงานหนึ่งซ้ำๆ ตามเงื่อนไขที่กำหนดได้โดยใช้คำสั่งควบคุมในภาษานั้นจะมีประโยคคำสั่งควบคุมที่ใช้ในการเลือกทำและทำงานซ้ำๆ ดังนี้

3.4.1 ประโยค IF/ELSE

```
if(expression)
    statement;
```

ถ้าหากเป็นการทำงานเลือกทำแบบมีสองทางเลือก และต้องการทำงานเพียงอย่างใดอย่างหนึ่งจะใช้ประโยค if-else ซึ่งมีรูปแบบดังนี้

```
if (expression)
    statement 1;
else
    Statement 2;
```

3.4.2 ประโยค switch

การเลือกทำที่มีทางเลือกหลายๆ ทางเลือกนั้นเราสามารถนำประโยค if-else มาซ้อนกันก็ได้ แต่ทำให้มองดูเข้าใจยาก ในภาษาซีจะมีประโยค switch ที่ใช้ในการเลือกทำอย่างใดอย่างหนึ่งจากหลายๆ ทางเลือก โดยมีรูปแบบของประโยคดังนี้

```
Switch(k)
{
    case 1: statement 1;
           break;
    case2: statement 2;
           Break;
    case3: statement 3;
           break;
```

```

case4: statement 4;
      break;
      : :
      : :
      : :
default: statement n;
}

```

3.5 การทำซ้ำ

คำสั่งให้โปรแกรมทำงานซ้ำถือว่าเป็นประโยคคำสั่งควบคุมอย่างหนึ่ง การทำซ้ำหรือที่เรียกว่าการทำลูปนั้นจะมีประโยคคำสั่งอยู่ 3 ประเภทคือ for, while และ do-while ซึ่งแต่ละแบบจะต่างกันตรงเงื่อนไขของการทำซ้ำ

3.5.1 ประโยค for

ประโยคคำสั่งนี้จะใช้ในกรณีที่มีจำนวนรอบของการทำซ้ำที่แน่นอน โดยมีรูปแบบดังนี้

```

for (initialization : condition : increment)
statement ;

```

โดยที่ initialization เป็นค่ากำหนดเริ่มต้นให้กับตัวแปรของการทำลูป condition เป็นเงื่อนไขที่ใช้ทดสอบการทำซ้ำครั้งต่อไป ซึ่งจะเป็นการกระทำลูปอีก increment เป็นการเพิ่มค่าให้ตัวแปรในการทำซ้ำแต่ละครั้ง สำหรับ statement จะกเป็นสแตตเมนต์ของคำสั่งที่จะทำซ้ำ ซึ่งอาจเป็นสแตตเมนต์รวมก็ได้

3.5.2 ประโยค while

การทำซ้ำแบบนี้จะตรวจสอบเงื่อนไขก่อนการทำซ้ำ ถ้าเงื่อนไขเป็นจริงจะทำสแตตเมนต์ที่กำหนดและทดสอบเงื่อนไขใหม่ ถ้าเงื่อนไขเป็นเท็จจะออกจากการทำซ้ำทันที โดยมีรูปแบบดังนี้

```

while (expression)
statement 1;

```

3.5.3 ประโยค do – while

การทำซ้ำประเภทนี้จะตรวจสอบเงื่อนไขภายหลังการทำสแตตเมนต์แต่ละครั้ง ถ้าหากเงื่อนไขเป็นเท็จจะออกจากการทำซ้ำทันทีโดยมีรูปแบบดังนี้

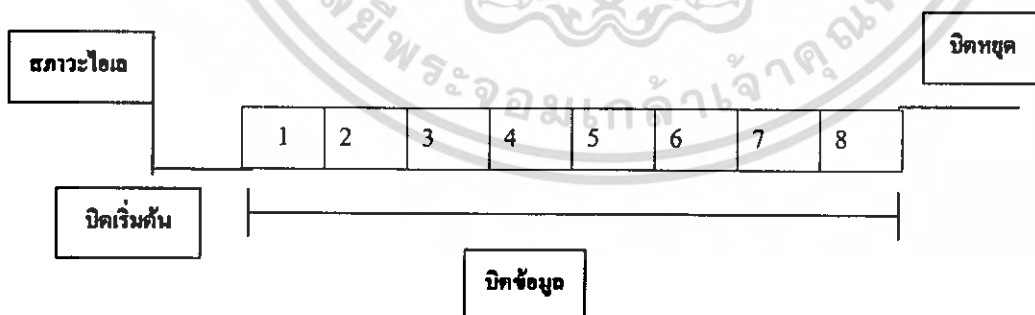
```
do {
    statement ;
}
while (condition);
```

3.6 Microcontroller เขียนโปรแกรมควบคุม AVR ผ่าน Serial Port

โดยปกติแล้ว AVR จะเป็นไมโครคอนโทรลเลอร์ที่มีความสามารถในการ รับข้อมูลจากภายนอกและนำมาประมวลผล พร้อมทั้งสามารถส่งสัญญาณ เพื่อทำการควบคุมการทำงานของอุปกรณ์ต่างๆ ได้อย่างดี และในส่วนของ การติดต่อสื่อสารข้อมูล (Data Communication) กับระบบภายนอกอื่นๆ ก็สามารถกระทำ โดยผ่านทางพอร์ตอนุกรม (Serial Port) ซึ่งพอร์ตอนุกรมนี้ จะเป็นส่วนที่เหมาะสมในการรับ หรือส่งข้อมูลในระยะทางไกลได้ดีกว่าพอร์ขนาน

3.6.1 การส่งข้อมูลแบบอนุกรมในโหมด 1 (Standard UART)

การส่งข้อมูลใน โหมดนี้จะเป็นการส่งข้อมูลแบบอะซิงโครนัส 8 บิตข้อมูล 1 บิตเริ่มต้นและ 1 บิตหยุดคิงรูปที่ 3-1 การทำงานในโหมดนี้จะทำโดยการกำหนดข้อมูลในรีจิสเตอร์ SCON บิต SM0 และบิต SM1 ให้มีค่าเป็น “01” ซึ่ง เป็นการกำหนดให้รีจิสเตอร์ SBUF กลายเป็นตัวรับส่งข้อมูลขนาด 10 บิตแบบฟูลดูเพล็กซ์ (Full Duplex) ซึ่งสามารถรับ และส่งข้อมูล ได้ภายในเวลาเดียวกัน โดยใช้ขา RxD ทำหน้าที่รับสัญญาณอนุกรมที่เข้ามา และขา TxD ทำการส่งข้อมูล แบบอนุกรมไปภายนอก



รูปที่ 3-1 แสดงการส่งข้อมูลของพอร์ตอนุกรม

การส่งข้อมูลจะเริ่มต้นด้วยการส่งบิตเริ่มต้น (Start bit) ออกไปแล้วตามด้วยบิตข้อมูล (โดยส่งบิต 0 ออกไปก่อน) จากนั้นจึงเป็นการส่งบิตหยุด (Stop bit) แฟล็ก TI จะเซ็ทเมื่อส่งข้อมูลครบทั้ง 10 บิต

การรับข้อมูลจะเริ่มจากลอจิกในสายสัญญาณเปลี่ยนสภาวะจาก 1 เป็น 0 (ขอบบวกของบิตเริ่มต้น) บิตเริ่มต้น จะถูกข้ามไปไม่สนใจ จะสุ่มเอาข้อมูลอีก 8 บิตที่เหลือเซารีจิสเตอร์เลื่อนบิตภายในพอร์ทอนุกรม เมื่อครบทั้ง 8 บิตแล้ว สิ่งต่อไปนี้จะเกิดขึ้น

1. บิต 9 (บิตหยุด) จะถูกเก็บเข้าไปในบิต RB8 ของ SCON
2. SBUF จะทำการไหลคข้อมูลทั้ง 8 บิตเข้าตัวเอง
3. แฟล็ก RI เซ็ท

สิ่งที่กล่าวมาทั้งหมดจะเป็นจริงต่อเมื่อ

1. RI = 0
2. SM2 = 1 และบิตหยุดที่รับเข้ามาเป็น 1 หรืออีกกรณีหนึ่งคือ SM2 = 0



บทที่ 4

โปรแกรมวิซวลเบสิก

วิซวลเบสิก (Visual Basic) เป็นโปรแกรมที่ใช้สร้างโปรแกรมประยุกต์ สำหรับระบบปฏิบัติการวินโดวส์ (Windows) เนื่องจากความง่ายของภาษาที่ใช้ในการเขียนและวิซวลเบสิกเป็นโปรแกรมที่เน้นในการสร้างแอปพลิเคชันเพื่อติดต่อกับผู้ใช้งานเป็นหลักทำให้ได้ผลงานออกมารวดเร็วสวยงาม และนอกจากนี้วิซวลเบสิกยังมีเครื่องมือที่ใช้ในการติดต่อกับฮาร์ดแวร์ซึ่งนำไปใช้ในการควบคุมเครื่องอีกด้วย




4.1 ขั้นตอนการสร้างโปรแกรมประยุกต์

การสร้างโปรแกรมประยุกต์ วิซวลเบสิก ประกอบด้วยขั้นตอนหลัก 3 ขั้นตอน คือ

1. การสร้างอินเทอร์เฟซ (Interface) โดยมีฟอร์ม (Form) เป็นอ็อบเจกต์ (Object) พื้นฐานและเป็นที่วางตัวคอนโทรล (Control) สำหรับการติดต่อกับผู้ใช้
2. ตั้งค่าคุณสมบัติ เป็นการกำหนดพฤติกรรมและการทำงานให้กับอ็อบเจกต์ต่างๆ
3. การเขียนคำสั่ง เป็นการควบคุมการประมวลผลผ่านโพรซีเจอร์ (Procedure) ที่กำหนด

4.2 คอนโทรลพื้นฐาน






กลุ่มคอนโทรลพื้นฐานของวิซวลเบสิกประกอบด้วย Text Box, Label, Command Button, Picture Box, Image, Check Box, Frame เป็นต้น

ไอคอน	ชื่อตัว Control	ชื่อ Class	คำอธิบาย
	Check box	CheckBox	ใช้กับการเลือกแบบ ถูก/ผิด(True/False, Yes/No)
	Combo box	ComboBox	เป็นตัว Control เป็นการผสมระหว่าง Text box กับ List box ซึ่งจะปรากฏรายการ เมื่อมีการคลิกลูกศร และ Combo box ไม่สนับสนุนการเลือกแบบหลายค่า
	Command button	CommandButton	ปุ่มคำสั่งเป็นตัว Control ที่ใช้ในทุกรูปแบบ ตามปกติจะเขียนคำสั่งใน Click Event Procedure ของตัว Control

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไอคอน	ชื่อตัว Control	ชื่อ Class	คำอธิบาย
	Data	Data	เป็นตัว Control ที่สามารถรวมข้อมูลกับฐานข้อมูลได้ และเป็นส่วนที่ Visual Basic ให้ผู้ใช้สามารถติดต่อระหว่างตัว Control บนฟอร์มกับฟิลด์ใน Table ของฐานข้อมูล โดย Data จะทำงานกับ Database Jet ของฐานข้อมูล แต่ไม่สามารถทำงานกับ ActiveX Data Object (ADO) ได้
	Directory List box	DirListBox	เป็น List box แบบหนึ่ง ที่แสดงไดเรกทอรีและพาร์ทที่เลือก
	Drive List box	DriveListBox	คล้ายกับ Combo box ที่ใช้เลือกชื่อของไดรฟ์ในระบบ
	File list box	FileListBox	เป็น List box ชนิดพิเศษที่ใช้แสดงชื่อไฟล์ในไดเรกทอรี
	Frame	Frame	สามารถใช้เป็น Container สำหรับตัว control อื่น
	Horizontal และ Vertical Scroll Bar	HScrollBar และ VScrollBar	ใช้เป็นแถบเลื่อนแบบ Stand-alone แต่มักจะไม่ค่อยมีการใช้ เพราะตัว Control อื่น ๆ ส่วนใหญ่ จะมีแถบเลื่อนของตัวเอง แถบเลื่อนแบบ Stand-alone อาจจะใช้ในลักษณะ Slider ได้
	Image	Image	เป็นตัว Control ใช้เก็บภาพคล้ายกับ Picture Box แต่ไม่สามารถทำงานแบบ Container ได้ Image มีข้อดีที่ใช้ทรัพยากรของระบบน้อยกว่า Picture Box
	Label	Label	เป็นตัว Control ที่ใช้แสดงข้อความ หรือป้ายชื่อ
	Line	Line	เป็นตัว Control ใช้สำหรับการตกแต่งด้านกราฟฟิก
	List box	ListBox	เป็นตัว Control ที่เก็บรายการของค่า และให้ผู้ใช้เลือก ซึ่งสามารถเป็นการเลือกค่าเดียวหรือหลายค่า ขึ้นกับการกำหนดคุณสมบัติ MultiSelect
	OLE container	OLE	เป็นตัว Control ที่สามารถเป็น Host Window ให้กับโปรแกรมภายนอก เช่น Microsoft Excel หรืออาจจะกล่าวว่าเป็นการสร้าง Window ให้กับโปรแกรมอื่นบนโปรแกรมประยุกต์ Visual Basic

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไอคอน	ชื่อตัว Control	ชื่อ Class	คำอธิบาย
	Option button	OptionButton	เป็นตัว Control ใช้กับกลุ่มตัว Control โดยให้เลือกได้เพียงตัว Control เดียวต่อครั้งหนึ่ง เมื่อมีการเลือกตัว Control ในกลุ่มแล้ว ตัว Control อื่นในกลุ่มจะเปลี่ยนจากการเลือกโดยอัตโนมัติ ถ้ามีการใช้ Option Button มากกว่า 2 กลุ่ม ต้องวางแต่ละกลุ่มใน Container เช่น Frame
	Picture box	PictureBox	ใช้แสดงภาพในฟอร์แมต BMP, DIB (Bitmap), ไอคอน (Icon), WMF (Metafile), GIF และ JPEG เป็นต้น และสามารถใช้เป็น container สำหรับตัว Control อื่น
	Shape	Shape	เป็นตัว Control ใช้สำหรับการตกแต่งด้านกราฟฟิก
	Text box	TextBox	เป็นตัว Control ที่เป็นฟิลด์ ใช้เก็บตัวอักษรที่สามารถแก้ไขโดยผู้ใช้ได้ และได้รับการใช้งานมาก
	Timer	Timer	เป็นตัว Control พิเศษที่ไม่เห็นเมื่อเวลาเรียกใช้วัตถุประสงค์การใช้คือการสร้าง Event ในฟอร์มแม่ โดยการเขียนคำสั่งใน Procedure ที่เจาะจงสำหรับการทำงานเบื้องหลัง เช่น การตรวจสอบสถานะของอุปกรณ์ต่อพ่วง

ตารางที่ 4-1 แสดงถึงคอนโทรลพื้นฐานของวิซวลเบสิกที่ใช้ในการสร้างโปรแกรม

4.3 คุณสมบัติร่วม

I. Left, Top, Width, Height ใช้จัดตำแหน่งของอ็อบเจกต์ เช่น การวางฟอร์มที่มุมซ้ายบน มีความกว้าง 5000 Twips และสูง 3000 Twips

Form1.Left = 0

Form1.Top = 0

Form1.Width = 5000

Form1.Height = 3000

2. ForeColor และ BackColor ใช้กำหนดสีของข้อความและสีพื้นหลังของอ็อบเจกต์ ซึ่งสามารถกำหนดสีเป็นแบบ Palette และ system เช่น

กำหนดสีแบบ Palette Label1.ForeColor = vbHighlightText

กำหนดสีแบบ System Label1.BackColor = &H800000D

รวมถึงการกำหนดด้วยค่าคงที่, เลขฐานสิบ และเลขฐานสิบหก เช่นตัวอย่างเป็นสีเดียวกัน

ค่าคงที่ Text1.BackColor = vbCyan

เลขฐานสิบ Text1.BackColor = 16776960

เลขฐานสิบหก Text1.BackColor = &HFFFF00

3. Font ใช้กำหนดลักษณะการแสดงผลตัวอักษร การตั้งค่าคุณสมบัติ Size, Bold, Italic, Underline และ Strikethrough เช่น

Text1.Font.Name = "Arial"

Text1.Font.Size = 12

Text1.Font.Bold = True

4. Caption, Text ใช้ในการแสดงข้อความ โดยคุณสมบัติ Caption เป็นข้อความที่ปรากฏในตัวคอนโทรลและไม่สามารถแก้ไขได้โดยผู้ใช้เมื่อเรียกใช้โปรแกรม เช่น

Label1.Caption = "Title"

Text1.Text = "Hello Word"

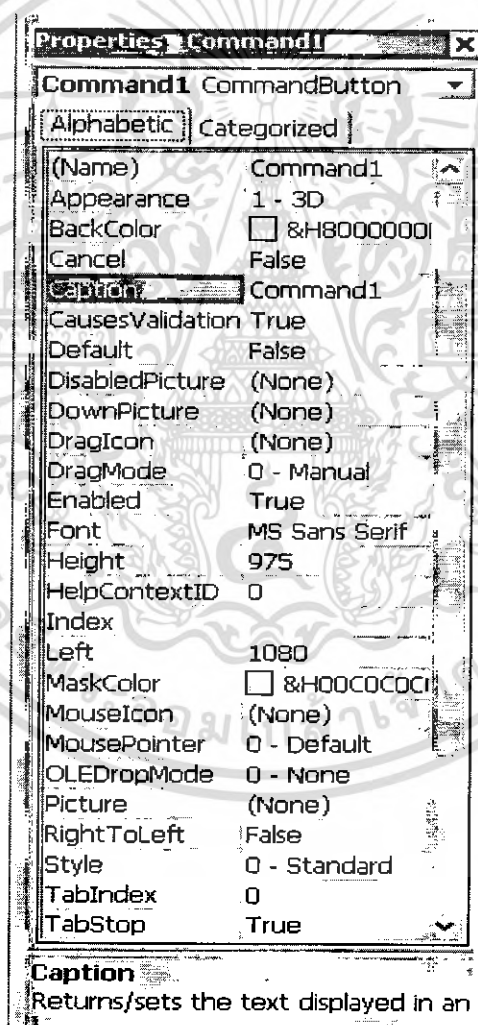
5. Enabled, Locked และ Visible โดยคุณสมบัติ Visible ใช้กำหนดการมองเห็น คุณสมบัติ Enabled และ Locked กำหนดการเข้าถึงตัว Control แต่ต่างกันว่า คุณสมบัติ Enabled ถ้ากำหนดไม่ให้เข้าถึง (Enabled = False) แล้ว ตัว Control ไม่สามารถรับโฟกัสได้ ส่วนคุณสมบัติ Locked ถ้ากำหนดไม่ให้เข้าถึง (Locked = True) แล้ว ตัว Control ยังสามารถรับโฟกัสได้

6. MousePointer และ MouseIcon ใช้กำหนดไอคอนของเมาส์เมื่อเคลื่อนที่ผ่านตัวคอนโทรล เช่น

Text1.MousePointer = vbCrosshair

7. คุณสมบัติอื่นๆ ได้แก่

- คุณสมบัติ Value มีค่าตามชนิดของตัวคอนโทรล
- คุณสมบัติ Index ใช้ในการสร้าง array ของตัวคอนโทรลและมีคุณสมบัติแบบอ่านอย่างเดียว
- คุณสมบัติ Appearance เป็นลักษณะหน้าตาของคอนโทรล ตามปกติจะกำหนดค่าคุณสมบัติเริ่มต้นเป็น 3 มิติ การกำหนดค่าทำได้ในเวลาออกแบบ และเป็นแบบอ่านอย่างเดียว เมื่อมีการเรียกใช้
- คุณสมบัติ Border Style ก็คือคุณสมบัติเส้นขอบ มีคอนโทรลสนับสนุน ได้แก่ Text box, Label, Frame, Picture box, Image และ OLE โดยถ้าตั้งค่าเป็น 0-none จะไม่มีเส้นขอบ ถ้าตั้งค่าเป็น 1-Fixed จะมีเส้นขอบ
- คุณสมบัติ ToolTip เป็นหน้าต่างสี่เหลี่ยมขนาดเล็กแสดงคำอธิบายสั้นๆ ซึ่งปรากฏขึ้นเมื่อเมาส์เคลื่อนที่ไปอยู่เหนือตัว control หรือไอคอน



รูปที่ 4-1 แสดงถึงการกำหนดคุณสมบัติของคอนโทรลที่ใช้ในการสร้างโปรแกรม

4.4 ทฤษฎีร่วม

1. Move ใช้ในการย้ายอ็อบเจกต์ หรือตัวคอนโทรล พร้อมทั้งปรับขนาดในเวลาเดียวกัน มีไวยากรณ์คือ

```
[object.]Move Left [, Top, Width, Height]
```

2. Refresh ใช้ในการวาดตัวคอนโทรลใหม่ ตามปกติไม่ใช้เนื่องจาก Visual Basic มีการปรับค่าโดยอัตโนมัติ แต่สามารถใช้ได้ในกรณีที่ต้องการให้มีการปรับคุณสมบัติทันที เช่น

```
Dim i As Integer
```

```
For i = 1000 to 1 Step - 1
```

```
Label1.Caption = (str(i))
```

```
Label1.Refresh
```

```
Next
```

3. SetFocus ใช้ย้ายโฟกัสไปที่คอนโทรลตัวที่ระบุ เช่น

```
Text1.SetFocus
```

4.5 อีเวนต์ร่วม

1. Click และ DbClick Event โดย Click Event จะเกิดเมื่อผู้ใช้คลิกปุ่มเมาส์ด้านซ้ายบนตัวคอนโทรล และ DbClick Event เกิดเมื่อเป็นการปุ่มเมาส์ด้านซ้ายแบบดับเบิลคลิก

2. Change Event เป็นอีเวนต์พื้นฐานที่เกิดเมื่อข้อมูลของตัวคอนโทรล มีการเปลี่ยนแปลง

3. GotFocus และ LostFocus Event ซึ่ง GotFocus Event เกิดขึ้นเมื่อตัวคอนโทรล ได้รับการโฟกัส เช่นเมื่อเมาส์เลื่อนมาถึง และ LostFocus Event จะเกิดเมื่อการ โฟกัสออกไปจากตัวคอนโทรล

4. KeyPress, KeyDown และ KeyUp Event เป็นอีเวนต์ ที่เกิดเพื่อคีย์แป้นพิมพ์ ขณะที่ตัวคอนโทรลได้รับการ โฟกัส โดยมีลำดับดังนี้ KeyDown (เมื่อผู้ใช้กดปุ่มแป้นพิมพ์) KeyPress (Visual Basic แปลปุ่มนั้นเป็นรหัส ANSI) และ KeyUp (เมื่อปล่อยปุ่มแป้นพิมพ์)

5.MouseDown, MouseUp และ MouseMove Event เป็นอีเวนต์ ที่เกิดขึ้นเมื่อมีการคลิก, ปล่อย หรือ ย้าย ของเมาส์บนตัว Control โดยลำดับการเกิด Click Event มีลำดับ คือ MouseDown -> MouseUp -> Click -> MouseMove ลำดับการเกิด DbClick Event มีลำดับ คือ MouseDown -> MouseUp -> Click -> MouseMove -> DbClick -> MouseUp -> MouseUp

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.6 อีเวนต์ของฟอร์ม

อีเวนต์ของฟอร์มเป็นอีเวนต์ที่ผู้ใช้ในการติดต่อกับผู้ใช้งาน ดังนี้

1. Load Event เป็นอีเวนต์ที่เกิดขึ้นต่อจาก Initialize Event ใช้ในการกำหนดค่าเริ่มต้นให้กับฟอร์ม และอีอปเจกต์
2. Resize Event เกิดขึ้นก่อนที่จะเห็นฟอร์ม โดยอีเวนต์สามารถใช้ในการปรับตัว คอนโทรลให้พอดีกับฟอร์ม นอกจากนี้ Resize event เกิดขึ้นผู้ใช้ปรับขนาดฟอร์มเอง
3. Unload Event เป็นช่วงสุดท้ายก่อนที่โปรแกรมจะถูกปิด อีเวนต์นี้เกิดเมื่อผู้ใช้ปิดโปรแกรม ก่อนที่โปรแกรมจะถูกปิด เราสามารถแจ้งข่าวสารให้ผู้ใช้ได้ เช่น ถ้ามถึงการบันทึกข้อมูลก่อนออกจากโปรแกรม ถ้าไม่มีการยกเลิกการ Unload ขึ้นต่อไป วิศวเบสิกจะทำการยกเลิกตัว คอนโทรล ปิดฟอร์ม และยกเลิกทรัพยากรต่าง ๆ ของวินโดว์

4.7 โพรซีเจอร์และฟังก์ชัน

โพรซีเจอร์หรือฟังก์ชันเป็นส่วนที่ใช้เขียน โปรแกรมลงไป ซึ่งสามารถช่วยลดความซ้ำซ้อนของโปรแกรมแต่ละส่วน สำหรับโพรซีเจอร์สามารถแบ่งได้เป็น 2 ประเภท คือ

1. โพรซีเจอร์เหตุการณ์ จะทำงานเมื่อเกิดเหตุการณ์ต่างๆขึ้นกับอีอปเจกต์ เช่น เมื่อผู้ใช้คลิกปุ่ม "Start" ถ้าหากภายในปุ่มมีโพรซีเจอร์อยู่ ก็จะเกิดการทำงานตามคำสั่งในโพรซีเจอร์ตันที่
2. โพรซีเจอร์ที่กำหนดเอง เป็นโพรซีเจอร์แบบทั่วไป หลังจากสร้างขึ้นแล้วเราสามารถเรียกใช้ได้ทันที เพียงแค่อ้างอิงชื่อของโพรซีเจอร์ตันที่เท่านั้น

4.8 ฟังก์ชัน

ลักษณะของฟังก์ชันจะคล้ายกับโพรซีเจอร์ เพียงแต่ฟังก์ชันจะมีการส่งค่ากลับไปยังโปรแกรมที่เรียกฟังก์ชันนั้นมาใช้

4.9 การประกาศตัวแปร

การประกาศตัวแปรมีคำสั่งแตกต่างกันไปตามชนิดของการใช้งานดังนี้

- Static : ตัวแปรนี้จะใช้ได้เฉพาะภายในโพรซีเจอร์ตันที่นั้นและค่าในตัวแปรจะถูกเก็บไว้ตลอดเวลา แม้ว่าจะออกโพรซีเจอร์ไปแล้วก็ตาม
- Dim : มีลักษณะเป็น Static แต่สามารถประกาศตัวแปรได้ 2 ที่ คือ ในโพรซีเจอร์ และ General Declarations (ส่วนแรกของโปรแกรม ใช้ในการประกาศตัวแปร(หากประกาศในโพรซีเจอร์แล้วเมื่อออกจากโพรซีเจอร์แล้ว ค่าในตัวแปรจะถูกรีเซ็ตใหม่อัตโนมัติ ดังนั้นในแต่ละโพรซีเจอร์อาจมีตัวแปรชื่อซ้ำกันได้ โดยไม่มีผลต่อกัน แต่การประกาศตัวแปรบน General Declarations จะทำให้โพรซี

เจอร์ทุกตัวบนฟอร์มนั้นสามารถเรียกใช้ได้ และหากมีการเปลี่ยนแปลงค่าในโพรซีเจอร์ใดๆแล้ว ก็จะมีผลไปกับโพรซีเจอร์อื่นๆด้วย

- Private : มีรูปแบบเช่นเดียวกับ Dim แต่ประกาศได้ที่ General Declarations เท่านั้น และจะใช้ได้เฉพาะบนฟอร์มนั้นเท่านั้น
- Public : มีรูปแบบเช่นเดียวกับ Private แต่สามารถใช้ได้ในทุกๆฟอร์มนบนโปรแกรมนั้น

ประเภทข้อมูล	ประเภท	ขนาด	การเก็บข้อมูล หรือ ช่วงข้อมูล
Integer	จำนวนเต็ม	2 ไบต์	-32,768 ถึง 32,767
Long	จำนวนเต็ม	4 ไบต์	-2,147,483,648 ถึง 2,147,483,647
Boolean	จำนวนเต็ม	2 ไบต์	เก็บค่า 0 และ -1 ซึ่งแทน False หรือ True
Byte	จำนวนเต็ม	1 ไบต์	เก็บค่าในช่วง 0 ถึง 255
Single	จำนวนทศนิยม	4 ไบต์	ค่าลบ -3.402823E38 ถึง -1.401298E-45 ค่าบวก 1.401298E-45 ถึง 3.402823E38
Double	จำนวนทศนิยม	8 ไบต์	ค่าลบ -1.79769313486232E308 ถึง -4.94065645841247E-324 ค่าบวก 4.94065645841247E-324 ถึง 1.79769313486232E308
Currency	จำนวนทศนิยม (4 ตำแหน่ง)	8 ไบต์	-922,337,203,477.5808 ถึง -922,337,203,477.5807
Decimal	จำนวนทศนิยม	8 ไบต์	ค่าที่ไม่มีทศนิยม +/- 79,228,162,514,264,337,593,543,950,335 ค่าที่มีทศนิยม +/- 7.92281625142643 37593543950335 และมีทศนิยม 28 ตำแหน่ง
String	ข้อความ		
Date	วันที่/เวลา	8 ไบต์	เก็บค่าระหว่าง 1 มกราคม ค.ศ. 100 ถึง 31 ธันวาคม ค.ศ. 9999 และเวลาใดๆ

ตารางที่ 4-2 แสดงถึงประเภทของข้อมูลพื้นฐานที่ใช้ในการสร้างตัวแปร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.10 คำสั่งพื้นฐาน

คำสั่งพื้นฐานที่ใช้ในการสร้างโปรแกรม ได้แก่

- คำสั่งเงื่อนไข If... Then... Else... End If มีรูปแบบดังนี้

IF

< เงื่อนไขที่ทำการเปรียบเทียบ >

Then

< ถ้าเงื่อนไขเป็นจริง เขียนโค้ดที่นี่ >

Else

< ถ้าเงื่อนไขไม่เป็นจริง เขียนโค้ดที่นี่ >

End If

- คำสั่งเงื่อนไข Select Case มีรูปแบบดังนี้

Select Case

< ชื่อตัวแปรที่ต้องการนำไปเปรียบเทียบ >

Case

< ค่าที่กำหนดสำหรับเปรียบเทียบ >

< กรณีค่าตัวแปรเปรียบเทียบตรงกับค่าที่กำหนดไว้ จะทำคำสั่งในส่วนนี้ >

Case

< ค่าอื่นที่ต้องการเปรียบเทียบ >

< กรณีค่าตัวแปรเปรียบเทียบตรงกับค่าที่กำหนดไว้ จะทำคำสั่งในส่วนนี้ >

Case Else

< กรณีนอกเหนือจากข้างต้น จะทำคำสั่งในส่วนนี้ >

End Select

- คำสั่งวนรอบแบบ For...Next มีรูปแบบดังนี้ คือ

For

< ชื่อตัวแปรประเภทตัวเลข > = < คำเริ่มต้น > To < คำสิ้นสุด >

< โปรแกรมที่ต้องการให้ทำงานวนรอบ >

Next

< ชื่อตัวแปร >

- คำสั่งวนรอบแบบ Do Until หรือ While...Loop มีรูปแบบดังนี้ คือ

Do While/Until

< เงื่อนไขการเปรียบเทียบ >

< คำสั่งที่ต้องการให้ทำงานวนรอบ >

Loop

- คำสั่งวนรอบแบบ Do...Until หรือ While...Loop มีรูปแบบดังนี้

Do

< คำสั่งที่ต้องการให้ทำงานวนรอบ >

Loop While/Until

< เงื่อนไขการเปรียบเทียบ >

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.11 คำสั่งเกี่ยวกับการจัดการรูปภาพ

คำสั่งเกี่ยวกับการจัดการรูปภาพ ได้แก่

- LoadPicture : เพื่อทำการโหลดรูปภาพเข้ามาไว้ใน Picture Box หรือ Image มีรูปแบบ คือ

[ชื่อ Picture Box].Picture = LoadPicture(ที่อยู่ของรูปภาพ)

- Point : ใช้อ่านค่าสีจากจุดที่กำหนดบนรูปภาพ ซึ่งจะให้ค่าสีในระบบ Long Color มีรูปแบบ คือ

[ชื่อ Picture Box].Point (พิกัด X, พิกัด Y)

- PSet : ใช้เพื่อวาดจุดลงไปในรูปภาพตามจุดที่กำหนด รูปแบบ คือ

[ชื่อ Picture Box Name].PSet (พิกัด X, พิกัด Y), รหัสสีที่ต้องการ

- PaintPicture : ใช้เพื่อคัดลอกรูปภาพจากที่หนึ่งไปอีกที่ โดยสามารถปรับขนาดได้

[ชื่อกล่องรูปภาพปลายทาง].PaintPicture(ชื่อกล่องรูปภาพต้นทาง, [ตำแหน่งเริ่มต้นของรูปต้นทางแกนX], [แกนY], [ขนาดความกว้างรูป], [ความสูงรูป], [ตำแหน่งเริ่มต้นของรูปต้นทางแกนX], [แกนY], [ขนาดความกว้างรูป], [ความสูงรูป], [รูปแบบการคัดลอกภาพ])

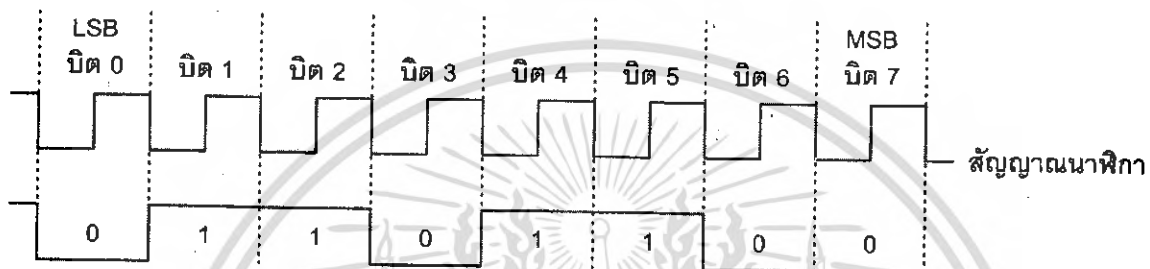
4.12 คำสั่งเกี่ยวกับการติดต่อพอร์ตอนุกรม

4.12.1 ทำความรู้จักกับพอร์ตอนุกรม

การเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพ่วงอื่น ๆ หรือคอมพิวเตอร์ด้วยกันแบบอนุกรมเป็นการรับข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลาย ๆ บิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลมาให้ครบทุกบิตเสียก่อนจึงทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลอนุกรมอาจมีความเร็วต่ำกว่าแบบขนาน ในด้านจำนวนสายสัญญาณการรับส่งข้อมูลแบบอนุกรมจะใช้จำนวนสายที่น้อยกว่ามาก อย่างน้อยที่สุดใช้เพียง 2 – 3 เส้นเท่านั้น แต่อัตราเร็วในการรับส่งข้อมูลอาจต่ำกว่าแบบขนาน อย่างไรก็ตามการรับส่งข้อมูลแบบอนุกรมสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

4.12.2 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นแบ่งออกได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิกา ร่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสคือซีพียูของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นจะเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูลและกราวด์ รูปที่ 1 – 1 แสดงให้เห็นถึงโทมิ่งไคอะแกรมของการส่งข้อมูลแบบซิงโครนัส



รูปที่ 4-2 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

4.12.3 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัสคือการรับและส่งข้อมูลไปในสายโดยไม่จำเป็นต้องมีสัญญาณนาฬิกา ร่วมด้วย เหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอเดรต (Baudrate) มีหน่วยเป็น บิตต่อวินาที (bit per second : bps)

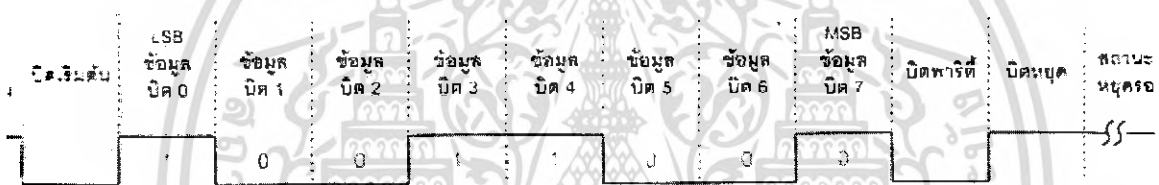
รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกันคือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรมจะมีขนาด 5,6,7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี (Parity Bit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1,1.5 หรือ 2 บิต

รูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส เมื่อไม่มีข้อมูลที่จะส่ง ขา DATA จะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (waiting stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา DATA มีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต เรียกบิตนี้ว่า บิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไป โดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต 5,6,7 หรือ 8 บิตก็ได้ จากนั้นตามด้วยบิตพาริตี ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่ส่งคือ บิตปิดท้าย ซึ่งจะให้

ขาดค่ามีสถานะลอจิก “1” อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver/Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600, และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าบอดเรตได้สูงถึง 11520 บิตต่อวินาที เนื่องจากบอดเรตคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตีความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที



รูปที่ 4.3 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

การตรวจสอบพาริตีสามารถกำหนดให้เป็นแบบคี่ (odd), แบบคู่ (even) หรือไม่มีการตรวจสอบพาริตีก็ได้ การตรวจสอบพาริตีเป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก “1” ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบิตพาริตีเข้าไปด้วย ยกตัวอย่าง ข้อมูลที่จะทำการส่งมีขนาด 8 บิต และมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสองจะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิก “1” จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตีเป็นคู่กว่าในบิตพาริตี จะต้องมีลอจิกเป็น “0” แต่ถ้าพาริตีเป็นคี่ ค่าที่บิตพาริตีจะต้องเป็น “1” เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตีมีจำนวนบิตที่เป็นลอจิก “1” รวมกันเป็นเลขคี่

บิตพาริตีถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART โดยภาครับจะต้องกำหนดคุณสมบัติการตรวจสอบพาริตีให้ตรงกันว่า จะตรวจสอบพาริตีคี่หรือคู่ จากนั้นภาครับของ UART จะตรวจสอบค่าพาริตีที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก “1” ทั้งหมดรวมทั้งบิตพาริตีด้วยถ้ากำหนดพาริตีไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแจ้งข้อผิดพลาดให้ผู้ใช้งาน นับเป็นการตรวจสอบความผิดพลาดของการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิต

เดียวเท่านั้น ถ้าข้อมูลที่ส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตีบิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี

ข้อมูล	บิตพาริตีคู่	บิตพาริตีคี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	1	0
11111111	0	1

ตารางที่ 4-3 แสดงลักษณะการทำงานของบิตพาริตี

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART เบอร์ 8250 UART ชิพเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5V) แต่เพื่อให้มีแรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ในระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิก “0” มีระดับแรงดัน +3V ถึง +12V ในขณะที่ลอจิก “1” มีระดับแรงดัน -3V ถึง -12V

4.12.4 มาตรฐานพอร์ตอนุกรมแบบ RS-232

มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้สื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดอยู่ห่างไกลกัน โดยคณะกรรมการที่เรียกว่า สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ได้วางมาตรฐานที่มีชื่อเรียกกันว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็คเตอร์เป็นแบบ DB-25 โดยกำหนดความสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3 ถึง -12V แสดงว่ามีข้อมูล (Mark) และ +3 ถึง +12V แสดงว่าเป็นช่องว่าง (Space)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

มาตรฐาน RS-232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่าอุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรลเลอร์ หรือ ไมโครคอมพิวเตอร์ ซึ่งมีความสามารถในการสร้างข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่ได้สังเกตเห็นคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ในโมเด็มจะเป็นแบบ DCE

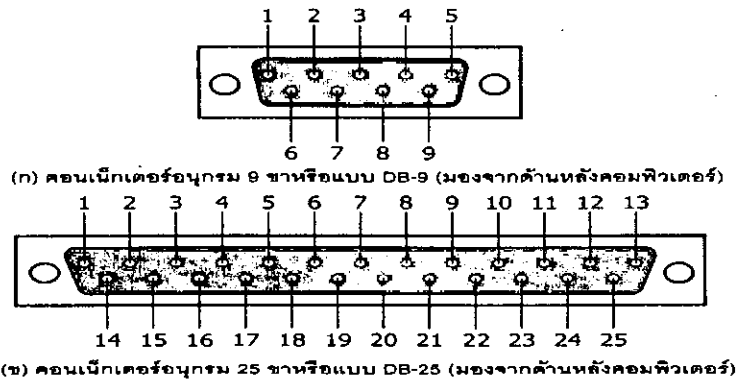
สำหรับการใช้งานบนคอมพิวเตอร์ พอร์ตอนุกรม RS-232 มักถูกใช้เชื่อมต่อกับโมเด็มหรือเมาส์ โดยสามารถรับส่งข้อมูลได้ด้วยความยาวของสายสัญญาณสูงสุดถึง 20 เมตร

4.12.5 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

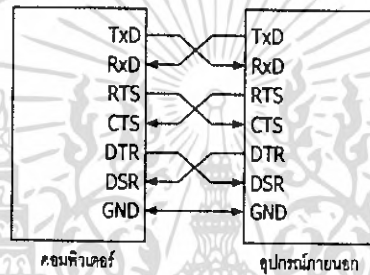
มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ DB-9 ตัวผู้ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่น ๆ ที่เคยใช้งานในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาในรูปแบบที่ 1-3

สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอก ลูกศรในรูปแสดงถึงทิศทางของข้อมูล การเชื่อมต่อแบบ Null modem หรือการเชื่อมต่อโดยตรงโดยไม่ต้องผ่าน โมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็กเต็มรูปแบบ การเชื่อมต่อแบบ Null modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ สำหรับรายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS-232 มีดังนี้

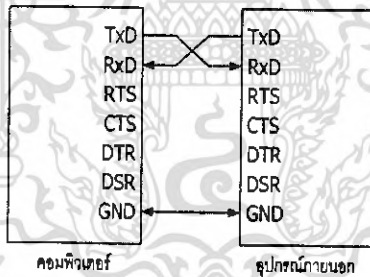
- Data Carrier Detect : DCD หรืออาจเรียกว่า Carrier Detect : CD ขานี้จะแอกติฟเมื่อมีการส่งสัญญาณพาห้จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็ม สำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก
- Receive Data : RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์ บัฟเฟอร์



รูปที่ 4-4 การจัดขาของคอนเน็กเตอร์อนุกรมตามมาตรฐาน RS-232 ทั้งแบบ DB-9 และ DB-25



(ก) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ Null modem



(ข) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ RS-232 โดยใช้สายสัญญาณเพียง 3 เส้น

รูปที่ 4-5 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่าง ๆ

-Transmitted Data : TD หรือ TxD ใช้ส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

-Data Terminal Ready : DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าจะต้อง การติดต่อกับโดยขา DTR นี้ต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทาง และ ขา DTR ของอุปกรณ์ปลายทางต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ ถ้าใช้การเชื่อมต่อเป็นแบบ Null Modem ซึ่งใช้สายในการ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่นิยมนำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DSR ของตัวมันเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วยในกรณีที่ใช้โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

-Signal Ground : GND กราวด์ระบบ

-Data Set Ready : DSR ขานี้จะใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR

-Request To Send : RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null Modem 3 จะต้องเชื่อมต่อขา RTS และ CTS ของตัวมันเองด้วยกัน เพื่อให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

-Clear To Send : CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณ ได้ข้อมูลที่ขา TxD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

-Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่วไปสายนี้จะ ไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มและ โปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

4.12.6 UART

UART มาจากคำว่า Universal Asynchronous Receiver Transmitter ซึ่งหมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสนั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัส แล้วส่งออกไป และทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจ้งข้อมูลอื่น ๆ ให้คอมพิวเตอร์รับทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต), รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพาริตี, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ (programmable buadrate generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารอยู่ในช่วง 1 – 65,535 UART สามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็กซ์ (half duplex) และฟูลดูเพล็กซ์ (full duplex) โดยการส่งแบบฮาล์ฟดูเพล็กซ์เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กซ์นั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

4.12.7 วงจรภายในและรีจิสเตอร์ของพอร์ตอนุกรม RS-232

เครื่องคอมพิวเตอร์โดยทั่วไปสามารถต่อพอร์ตอนุกรมสูงสุดได้ 4 พอร์ต มีชื่อเรียกเป็น com1, com2, com3 และ com4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างก็ใช้งาน UART ภายในคอมพิวเตอร์ในการติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน การทำงานภายในของพอร์ตอนุกรมประกอบด้วยรีจิสเตอร์ 8 บิต 8 ตัวที่ใช้งานร่วมกับ UART แอแดคเรสของรีจิสเตอร์ภายในพอร์ตอนุกรม สามารถคำนวณได้จากค่ารีจิสเตอร์พื้นฐานของพอร์ตอนุกรม ยกตัวอย่าง พอร์ตอนุกรม com1 มีแอดเดรสอยู่ที่ 3F8H ตำแหน่งของรีจิสเตอร์ต่าง ๆ จะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ตอนุกรมมีดังนี้

- 00H เป็นรีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่ได้รับเข้ามาหรือเตรียมข้อมูลก่อนที่จะส่งออกไป
- 01H รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัปต์ ใช้เซตโหมดการอินเตอร์รัปต์ของพอร์ตอนุกรม
- 02H รีจิสเตอร์แสดงโหมดการอินเตอร์รัปต์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัปต์
- 03H รีจิสเตอร์กำหนดรูปแบบของข้อมูล
- 04H รีจิสเตอร์ควบคุมโมเด็ม ใช้ตรวจสอบบิตสำหรับติดต่อกับโมเด็ม เช่น RTS หรือ DTR
- 05H รีจิสเตอร์แสดงสถานะการรับและการส่งข้อมูลแบบอนุกรม
- 06H รีจิสเตอร์แสดงสถานะของโมเด็ม ซึ่งจะแสดงสถานะของขา DCD, RI, DSR และ CTS
- 07H รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว

4.12.8 ลักษณะสัญญาณอินพุตและเอาต์พุตของพอร์ต RS-232

สัญญาณเอาต์พุตที่ใช้ควบคุม (RTS และ DTR) และสัญญาณแสดงสถานะอินพุต (CTS, DSR และ DCD) ของพอร์ตอนุกรม RS-232 จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้งภาคส่งและรับจะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาต์พุตออกมาเป็นแบบทีทีแอลเท่านั้น ดังนั้นเมื่อสัญญาณถูกส่งออกมาจาก UART จึงต้องส่งเข้าสู่วงจรขับเพื่อปรับระดับแรงดันให้ได้ระดับสัญญาณเป็นไปตามมาตรฐาน RS-232 ก่อนส่งออกไปจากคอมพิวเตอร์สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับในลักษณะนี้เช่นเดียวกัน เพื่อให้ได้ระดับสัญญาณในระดับเดียวกัน แต่วงจรขับที่ใช้ทั้งหมดภายในคอมพิวเตอร์และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะถูกกลับสถานะ

แอดเดรสของพอร์ตอนุกรม

แอดเดรสพื้นฐานของพอร์ตอนุกรมมี 4 ตำแหน่งดังนี้คือ

COM1 : 3F8H

COM2 : 2F8H

COM3 : 3E8H

COM4 : 2E8H

บิต 3	บิต 2	บิต 1	จำนวนพอร์ต
0	0	0	ไม่มีพอร์ตอนุกรม
0	0	1	มีพอร์ตอนุกรม 1 พอร์ต
0	1	0	มีพอร์ตอนุกรม 2 พอร์ต
0	1	1	มีพอร์ตอนุกรม 3 พอร์ต
1	0	0	มีพอร์ตอนุกรม 4 พอร์ต

ตารางที่ 4-4 แสดงข้อมูลในแอดเดรส 0000:0411H ที่ใช้แจ้งจำนวนพอร์ตอนุกรม

เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสภายในคอมพิวเตอร์จะทำการตรวจสอบแอดเดรสของพอร์ตอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ตอนุกรม ไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบต์ สำหรับพอร์ตอนุกรม COM1 จะเก็บไว้ที่แอดเดรส 0000:0400H และ 0000:0401H ส่วนตำแหน่งอื่น ๆ มีรายละเอียดดังนี้

COM2 = 0000:0402H – 0000:0403H

COM3 = 0000:0404H – 0000:0405H

COM4 = 0000:0406H – 0000:0407H

นอกจากนี้ที่หน่วยความจำแอดเดรส 0000:0411H ยังใช้สำหรับแสดงจำนวนของพอร์ตอนุกรมที่มีอยู่ในคอมพิวเตอร์อีกด้วย

4.12.9 การเขียนโปรแกรมเพื่อการใช้งานพอร์ตอนุกรม

ก่อนการใช้งานพอร์ตอนุกรมนั้นจะต้องมีการกำหนดค่าเริ่มต้นให้กับตัวมันก่อน ซึ่งก็คือการกำหนดจำนวนบิตข้อมูลที่ต้องการส่ง, จำนวนบิตปิดท้าย, ชนิดของพาริตีที่ใช้ และบอดเรต การกำหนดสามารถทำได้หลายวิธี วิธีแรกเป็นการกำหนดจากคอสพรีอัมพ์ โดยใช้คำสั่ง MODE ซึ่งมีวิธีการใช้งานดังนี้

แต่เมื่อใช้คำสั่งนี้ในขณะที่โปรแกรมทำงานผ่านระบบปฏิบัติการวินโดวส์ จะไม่สามารถใช้งานได้เนื่องจากระบบปฏิบัติการวินโดวส์ได้เข้าฝั่งตัวพอร์ตอนุกรมเข้าเป็นส่วนหนึ่งของระบบปฏิบัติการแล้ว ดังนั้นการเรียกใช้งานจึงจำเป็นต้องเรียกผ่านเครื่องมือที่ติดต่อผ่านระบบปฏิบัติการ เช่นการใช้คอนโทรล MSCOMM32.OCX ของโปรแกรม Visual BASIC

4.12.10 คอนโทรล MSComm

สำหรับการใช้งาน Visual BASIC ตั้งแต่เวอร์ชัน 2 เป็นต้นมา ใน Visual BASIC จะมีก๊อสตอมคอนโทรลสำหรับการสื่อสารอนุกรมผ่านทางพอร์ตอนุกรมของคอมพิวเตอร์มาให้ โดยใน Visual BASIC เวอร์ชัน 2 และเวอร์ชัน 3 จะใช้ชื่อว่า MSCOMM.VBX ส่วนเวอร์ชัน 4 ใช้ชื่อว่า MSCOMM16.OCX สำหรับการดำเนินงานกับระบบปฏิบัติการ 16 บิต และ MSCOMM32.OCX สำหรับการดำเนินงานกับระบบปฏิบัติการ 32 บิต สำหรับใน Visual BASIC เวอร์ชัน 5 จะมีเพียง MSCOMM 32.OCX เท่านั้นเพราะถูกออกแบบมาให้ใช้งานกับระบบปฏิบัติการ 32 บิต

MSComm จัดเตรียมทางเลือกเอาไว้ 2 ทางเพื่อความสะดวกในการสื่อสารข้อมูล ทางแรกคือ การสื่อสารข้อมูลที่กระตุ้นด้วยเหตุการณ์ (event-driven communications) เป็นรูปแบบการใช้งานที่มีประสิทธิภาพมากสำหรับการตอบสนองแบบทันทีทันใด เช่น เมื่อตัวอักษรถูกส่งมาที่พอร์ตอนุกรมหรือเกิดการเปลี่ยนแปลงที่ขา Data Carrier Detect (DCD) หรือขา Request To Send (RTS) เหตุการณ์ Oncomm ของ MSComm จะสามารถตรวจจับสัญญาณนั้นได้ทันที ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อคุณสมบัติ CommEvent ต่อไป ส่วนทางเลือกที่สองเป็นการคอยตรวจสอบค่าเหตุการณ์และความผิดพลาดที่เกิดขึ้นด้วยการดูค่าที่เปลี่ยนแปลงภายในคุณสมบัติ CommEvent หลังจากให้โปรแกรมทำงานในฟังก์ชันต่าง ๆ ไปเรียบร้อยแล้ว ซึ่งวิธีนี้ใช้งานได้ดีในกรณีที่โปรแกรมมีขนาดเล็ก

คอนโทรล MSComm 1 ตัวสามารถควบคุมการทำงานพอร์ตอนุกรมได้ 1 พอร์ต ถ้าในโปรแกรมที่ใช้งานต้องการติดต่อกับพอร์ตอนุกรมมากกว่า 1 พอร์ตจะต้องใช้คอนโทรล MSComm มากกว่า 1 ตัวเพื่อควบคุมพอร์ตอนุกรมในแต่ละพอร์ต แอคเตอรส์ของพอร์ตอนุกรมและแอคเตอรส์ของการเกิดอินเตอร์รัปต์สามารถเปลี่ยนแปลงได้จากการแก้ไขค่าที่ Control Panel

ถึงแม้ว่า คอนโทรล MSComm จะมีคุณสมบัติ (property) มากมายแต่สามารถทำความเข้าใจได้ไม่ยาก

CommPort

ใช้ในการกำหนดและอ่านค่าพอร์ตอนุกรมที่ติดต่อยู่ (COM1, COM2, COM3, COM4)

รูปแบบการใช้งาน

Object.Settings [=value]

ค่า Value มีชนิดข้อมูลเป็นแบบ String มีรูปแบบเป็น "BBBB,P,D,S" โดย BBBB เป็นค่าอัตราบอด, P เป็นค่าพาริตี, D เป็นจำนวนของบิตข้อมูล และ S เป็นจำนวนของบิตปิดท้าย ปกติแล้วค่านี้ถูกกำหนดไว้เป็น "9600, N, 8, 1"

ตัวอย่างการใช้งานคำสั่ง Settings โดยจะเป็นการกำหนดค่าบอดเรตเท่ากับ 9600 ไม่มีพาริตี จำนวนบิตข้อมูล 8 บิต และ บิตปิดท้าย 1 บิต สามารถเขียนโปรแกรมได้ดังนี้

MSComm1.Settings = "9600, N, 8, 1"

PortOpen

ใช้ในการกำหนดและอ่านค่าสถานะของพอร์ตอนุกรม เพื่อเปิดและปิดพอร์ตอนุกรม
รูปแบบการใช้งาน

Object.PortOpen [= value]

ค่า Value มีชนิดข้อมูลเป็นแบบบูลีน คือ True กับ False โดย True หมายถึงการเปิดพอร์ตอนุกรมและ False หมายถึงการปิดพอร์ตอนุกรม สำหรับการปิดพอร์ตนั้นจะมีการเคลียร์บัฟเฟอร์รับข้อมูลและบัฟเฟอร์ส่งข้อมูลด้วย คอนโทรล MSComm จะปิดพอร์ตอนุกรมโดยอัตโนมัติเมื่อออกจากโปรแกรม ก่อนที่จะใช้คุณสมบัติ PortOpen ต้องตรวจสอบให้แน่ใจก่อนว่าคุณสมบัติ CommPort นั้นได้ทำการกำหนดตำแหน่งของพอร์ตอนุกรมไว้ถูกต้องหรือไม่ มิเช่นนั้น MSComm จะแสดงข้อผิดพลาด Error 68 แจ้งแก่ผู้ใช้งาน หรือถ้าพอร์ตอนุกรมนั้นถูกเปิดเอาไว้แล้ว โปรแกรมก็จะแจ้งข้อผิดพลาดออกมาเช่นเดียวกัน

ถ้าคุณสมบัติ DTREnable หรือ RTSEnable ถูกกำหนดให้เป็น True ก่อนที่จะทำการเปิดพอร์ต ค่าคุณสมบัติของ DTREnable หรือ RTSEnable จะถูกเซตเป็น False หลังจากปิดพอร์ต แต่ถ้าเซตเป็น False หลังจากปิดโปรแกรมแล้ว ค่าที่กำหนดไว้จะเป็นค่าเดิม

ตัวอย่างการใช้คำสั่งเปิดพอร์ต เพื่อติดต่อกับพอร์ตอนุกรม COM1 และมีบอดเรต 9600 บิตต่อวินาที ไม่มีพาริตี จำนวนบิตข้อมูล 8 บิต และบิตปิดท้าย 1 บิต มีดังนี้

```
MSComm.Settings = "9600, n, 8, 1"
```

```
MSComm.CommPort = 1
```

```
MSComm.PortOpen = True
```

Input

อ่านค่าและลบค่าขบวนข้อมูลจากบัฟเฟอร์ภาครับ

รูปแบบการใช้งาน

Object.Input

คุณสมบัติ InputLen เป็นตัวกำหนดจำนวนของตัวอักษรที่จะอ่าน โดยคุณสมบัติ Input การกำหนดค่าให้ InputLen เท่ากับ 0 เป็นการกำหนดให้คุณสมบัติ Input ทำการอ่านค่าข้อมูลในบัฟเฟอร์รับข้อมูลทั้งหมด

คุณสมบัติ InputMode เป็นตัวกำหนดชนิดของข้อมูลที่คุณสมบัติ Input รับเข้ามา ถ้า InputMode ถูกกำหนดเป็น comInputModeText คุณสมบัติ Input จะส่งค่าข้อมูลกลับมาในรูปแบบของข้อความชนิดข้อมูลเป็นแบบ Variant ถ้า InputMode กำหนดเป็น comInputModeBinary คุณสมบัติ Input จะส่งข้อมูลกลับมาในรูปแบบของไบนารีและชนิดข้อมูลเป็นแบบ Variant

ตัวอย่างโปรแกรมแสดงให้เห็นถึงวิธีการรับข้อมูลจากบัพเฟอร์รับข้อมูลทั้งหมด

```
Private Sub Command1_Click ()
```

```
Dim InString as String
```

```
MSComm1.InputLen = 0      ' Retrieve all available data.
```

```
If MSComm1.InBufferCount Then ' Check for data.
```

```
InString = MSComm1.Input    ' Read data.
```

```
End If
```

```
End Sub
```

InBufferCount

ส่งค่าจำนวนของตัวอักษรที่อยู่ในบัพเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

```
Object.InBufferCount[ = value]
```

คำสั่ง InBufferCount จะแสดงค่าจำนวนของตัวอักษร ซึ่งรับมาจากภายนอกและยังเก็บอยู่ในบัพเฟอร์ภาครับ เพื่อให้ผู้ใช้งานอ่านค่าออกไป สำหรับการเคลียร์ค่าบัพเฟอร์ภาครับทำได้โดยกำหนดให้ InBufferCount มีค่าเป็น 0

InBufferSize

กำหนดและคืนค่าขนาดของบัพเฟอร์ภาครับในหน่วยเป็นไบต์

รูปแบบการใช้งานคำสั่ง

```
Object.InBufferSize [ = value ]
```

คำสั่ง InBufferSize ใช้เพื่อกำหนดขนาดของบัพเฟอร์ภาครับ ค่าเริ่มต้นกำหนดไว้ที่ 1024 ไบต์

InputLen

กำหนดค่าและคืนค่าจำนวนของตัวอักษรที่อ่านจากบัพเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

```
Object.InputLen [ = value ]
```

ค่าเริ่มต้นของคุณสมบัติ InputLen มีค่าเท่ากับ "0" การกำหนดค่าเท่ากับ "0" จะทำให้ คำสั่ง Input ของ MSComm อ่านค่าข้อมูลที่อยู่ภายในบัพเฟอร์ภาครับทั้งหมด

ถ้าไม่มีข้อมูลอยู่ในบัฟเฟอร์ภาครับมากเท่ากับจำนวน InputLen คำสั่ง Input จะส่งค่าว่าง (“ ”) กลับออกมา ผู้ใช้งานสามารถตรวจสอบข้อมูลในบัฟเฟอร์ภาครับได้โดยใช้คุณสมบัติ InBufferCount โดยกำหนดให้มีข้อมูลอยู่ในบัฟเฟอร์ภาครับก่อนแล้วจึงค่อยอ่านข้อมูลจากบัฟเฟอร์ภาครับ

คุณสมบัตินี้มักใช้กับการอ่านค่าข้อมูลจากเครื่องมือหรือเครื่องจักรที่มีการกำหนดค่าขนาดความยาวของข้อมูลเอาไว้แล้ว

InputMode

กำหนดค่าและคืนค่าชนิดของข้อมูลที่รับโดยคำสั่ง Input

รูปแบบการใช้งานคำสั่ง

Object.InputMode [= value]

คุณสมบัติ InputMode ใช้กำหนดว่าข้อมูลชนิดไหนที่รับเข้ามาผ่านคำสั่ง Input โดยมี 2 ประเภทคือ **comInputModeText** สำหรับข้อมูลที่อยู่ในรูปข้อความตัวอักษรตามมาตรฐาน ANSI โดยจะต้องกำหนดค่าเป็น “0” และค่าเริ่มต้นของการรับค่าข้อมูลก็จะเป็นค่านี้

comInputModeBinary สำหรับข้อมูลอื่น ๆ ซึ่งจะเก็บในรูปแบบไบนารีรวมกันอยู่เป็น ไรต์ข้อมูลและเก็บข้อมูลไว้ในตัวแปรแบบอาร์เรย์ ชนิดข้อมูลเป็นแบบไบต์

Output

ใช้ในการส่งขบวนการของข้อมูลไปยังบัฟเฟอร์ส่งข้อมูล

รูปแบบการใช้งาน

Object.Output [= value]

ค่า value เป็นค่าของตัวอักษรที่เขียนไปยังบัฟเฟอร์ส่งข้อมูล คุณสมบัติ Output สามารถใช้ในการส่งข้อมูลตัวอักษรหรือข้อมูลไบนารีก็ได้ โดยการส่งข้อมูลเป็นรูปแบบตัวอักษรจะต้องกำหนดข้อมูลเป็นแบบ Variant และมีข้อมูลภายในเป็นแบบ String สำหรับการส่งข้อมูลไบนารีจะต้องกำหนดชนิดของข้อมูลเป็นแบบ Variant และมีข้อมูลภายในเป็นแบบ Byte

OutBufferCount

คืนค่าจำนวนของข้อมูลตัวอักษรที่เก็บอยู่ในบัฟเฟอร์ภาคส่ง และสามารถใช้คำสั่งนี้เพื่อเคลียร์บัฟเฟอร์ภาคส่งได้ด้วย

รูปแบบการใช้งานคำสั่ง

Object.OutBufferCount [= value]

ผู้ใช้งานสามารถเคลียร์บัฟเฟอร์ภาคส่งได้โดยการกำหนดค่า OutBufferCount เท่ากับ “0”

OutBufferSize

กำหนดค่าและคืนค่าขนาดของบัฟเฟอร์ภาคส่ง ชนิดตัวแปรเป็นแบบไบต์
รูปแบบการใช้งานคำสั่ง

Object.OutBufferSize [= object]

คุณสมบัติ OutBufferSize ใช้สำหรับกำหนดขนาดของบัฟเฟอร์ภาคส่ง โดยค่าปกติที่ใช้งานจะมีค่าเท่ากับ 512 ไบต์

ParityReplace

กำหนดและคืนค่าตัวอักษรที่ไปวางแทนในตำแหน่งที่เกิดข้อผิดพลาดจากพาริตี
รูปแบบการใช้งานคำสั่ง

Object.ParityReplace [= value]

บิตพาริตี เป็นบิตที่ทางภาคส่งข้อมูลทำการส่งมาพร้อมกับข้อมูล เพื่อตรวจสอบข้อผิดพลาดของข้อมูล โดยเมื่อมีการใช้บิตพาริตี คอนโทรล MSCOM จะทำการบวกบิตทุกบิตที่มีค่าลอจิก “1” ในแต่ละไบต์ และทำการตรวจสอบผลลัพธ์ว่าบิตที่อ่านได้นั้นมีจำนวนลอจิก “1” เป็นเลขคู่หรือคี่ และตรงกับค่าที่กำหนดไว้แต่ต้นหรือไม่ ถ้าค่าที่นำมาบวกแล้วมีพาริตีไม่ตรงแสดงว่าการรับส่งข้อมูลผิดพลาด

การกำหนดค่า เริ่มต้นให้กับ ParityReplace นั้นกำหนดให้ใช้เครื่องหมาย (?) ไปวางไว้ที่ตำแหน่งที่เกิดพาริตีผิดพลาด ถ้ากำหนดค่า ParityReplace ให้เป็นค่าว่าง (“”) จะเป็นการยกเลิกการใช้งาน ParityReplace ใช้ชนิดข้อมูลเป็นแบบสตริง แต่จะทำการกำหนด จะกำหนดได้เพียง ไบต์เดียวเท่านั้น ซึ่งจะสามารถใช้ค่าใด ๆ ก็ได้ที่เป็นโค้ด ANSI มีค่าอยู่ระหว่าง 0-255

DTREnable

ใช้ในการกำหนดสถานะลอจิกของขา Data Terminal Ready (DTR) โดยสัญญาณของขา DTR จะส่งจากคอมพิวเตอร์ไปยังโมเด็มเพื่อแสดงว่าคอมพิวเตอร์พร้อมที่จะรับข้อมูลแล้ว ชนิดของข้อมูลเป็นแบบบูลีน
รูปแบบการใช้งาน

Object.DTREnable [= value]

ค่า Value เป็นสถานะ True หรือ False เพื่อกำหนดลอจิกของขา DTR ให้เป็น “0” หรือ “1” โดย

True หมายถึง ให้ขา DTR มีลอจิก “1”

False หมายถึง ให้ขา DTR มีลอจิก “0”

สำหรับการใช้งานกับโมเด็ม การทำให้ขา DTR เป็นลอจิก “0” จะเป็นการวางหูโทรศัพท์หรือยกเลิกการติดต่อ

RTSEnable

ใช้เพื่อกำหนดสถานะลอจิกให้ขา Request To Send (RTS) โดยขา RTS จะเป็นสัญญาณที่ส่งจากคอมพิวเตอร์ไปยังโมเด็มเพื่อร้องขอส่งข้อมูล ชนิดของข้อมูลเป็นแบบ Boolean

รูปแบบการใช้งาน

Object.RTSEnable [= value]

ค่า Value เป็นสถานะ True หรือ False เพื่อกำหนดลอจิก “0” หรือ “1” ให้ขา RTS โดย

True หมายถึง ให้ขา RTS มีลอจิก “1”

False หมายถึง ให้ขา RTS มีลอจิก “0” (เป็นค่าปกติ)

EOFEnable

เป็นการกำหนดให้ MScComm รอสัญลักษณ์แสดงส่วนท้ายสุดของไฟล์ (End of file : EOF) ระหว่างการรับอินพุตเข้ามา ถ้าพบสัญลักษณ์ EOF ภาคอินพุตจะหยุดรับข้อมูล และเหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงาน คุณสมบัติ CommEvent จะมีค่าเท่ากับ 7 หรือ ComEvEOF

รูปแบบการใช้งาน

Object.EOFEnable [= value]

โดย value เป็นค่าสถานะ True หรือ False เพื่อเอนเอเบิลหรือดิสเอเบิลการทำงานของเหตุการณ์ OnComm เมื่อตรวจพบสัญลักษณ์ EOF โดย

True หมายถึง เหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงานด้วย EOF

False หมายถึง เหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงานด้วย EOF (เป็นค่าปกติ)เมื่อ EOFEnable กำหนดให้เป็น False ส่วนควบคุมจะไม่มีการตรวจสอบสัญลักษณ์ EOF

CTSHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา Clear To Send (CTS) ได้ว่ามีสถานะลอจิก “0” หรือ “1” โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า CTSHolding เป็น True ขา CTS จะมีสถานะลอจิกเป็น “1” ถ้าค่า CTSHolding เป็น False ขา CTS จะมีสถานะลอจิกเป็น “0”

รูปแบบการใช้งาน

Object.CTSHolding

เมื่อขา CTS เป็นลอจิก “0” (CTSHolding = False) และเกิดไทม์เอาต์ คอนโทรล MScComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventCTSTO (Clear To Send Timeout) และกระตุ้นให้เกิดเหตุการณ์ OnComm

CDHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา Data Carrier Detect (DCD) ได้ว่ามีสถานะลอจิกเป็น “1” หรือ “0” โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า CDHolding เป็น True ขา DCD จะมีสถานะลอจิก “1” หรือ “0” โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า DSRHolding เป็น True ขา DSR จะมีสถานะลอจิก “1” ถ้าค่า DSRHolding เป็น False ขา DSR จะมีสถานะลอจิก “0”

รูปแบบการใช้งาน

Object.DSRHolding

เมื่อขา DSR เป็นลอจิก “1” (DSRHolding = True) และเกิดไทม์ คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventDSRTO (Data Set Ready Timeout) และกระตุ้นให้เกิดเหตุการณ์ OnComm

รูปแบบการใช้งานคำสั่ง

Object.Handshaking [= value]

ค่าตัวแปร Value ที่ใช้กำหนดค่ากำหนดได้ 4 รูปแบบด้วยกันคือ

1. comNone ค่าที่กำหนดคือ 0 เป็นการกำหนดให้ไม่มีการแฮนด์เช็ก (เป็นค่าเริ่มต้น)
2. comXOnXOff ค่าที่กำหนดคือ 1 เป็นการกำหนดให้ใช้แฮนด์เช็กแบบ XON/XOFF
3. commRTS ค่าที่กำหนดคือ 2 เป็นการกำหนดให้ใช้ขา RTS/CTS (Request To Send/Clear To Send)
4. comRTSXOnXOff ค่าที่กำหนดคือ 3 เป็นการกำหนดให้ใช้ทั้งแบบ Request To Send และ

XON/XOFF

คุณสมบัติ Handshaking ใช้เพื่อกำหนดรูปแบบการสื่อสารภายใน ระหว่างที่ข้อมูลถูกส่งไปยังบัฟเฟอร์ภาครับ เมื่อข้อมูลตัวอักษรถูกส่งมาถึงพอร์ตอนุกรม อุปกรณ์สื่อสารข้อมูลจะทำการย้ายข้อมูลไปยังบัฟเฟอร์ภาครับ เพื่อที่จะให้โปรแกรมสามารถอ่านค่าไปใช้งานได้ ถ้าไม่มีบัฟเฟอร์ภาครับ โปรแกรมที่ใช้งานจะต้องทำการอ่านค่าข้อมูลโดยตรงจากฮาร์ดแวร์ของพอร์ตอนุกรม ซึ่งผู้ใช้งานจะเกิดปัญหาข้อมูลสูญหายได้ เนื่องจากการเปลี่ยนแปลงของข้อมูลที่ส่งเข้ามามีการเปลี่ยนแปลงอย่างรวดเร็ว

คุณสมบัติ handshaking ช่วยให้ผู้ใช้งานแน่ใจได้ว่าข้อมูลที่ได้รับมานั้นไม่มีการสูญหายเมื่อบัฟเฟอร์ภาครับที่รับข้อมูลนั้นเกิดข้อมูลล้นหรือโอเวอร์โฟลว (overflow) โดยใช้วิธีการตรวจสอบความพร้อมของบัฟเฟอร์ว่าพร้อมรับข้อมูลหรือไม่ก่อนที่จะส่งข้อมูลมาให้

รูปแบบการใช้งาน

Object.Break [= value]

เหตุการณ์ OnComm

เหตุการณ์ OnComm จะถูกสร้างขึ้นเมื่อค่าของคุณสมบัติ CommEvent มีการเปลี่ยนแปลงเพื่อแสดงผลการเปลี่ยนแปลงเหล่านั้นแบบทันทีทันใดหรือ แสดงข้อผิดพลาดที่เกิดขึ้น ตัวอย่าง โปรแกรมย่อย OnComm เพื่อนำเหตุการณ์ CommEvent มาแสดงมีดังนี้

การใช้ MSComm เพื่อการติดต่อฮาร์ดแวร์

จากรายละเอียดของ MSComm ที่กล่าวไปในตอนต้นนั้น จะเห็นได้ว่าวิธีการที่จะอ่านค่าหรือเขียนค่าไปยังขาสถาปนะและควบคุมของพอร์ตอนุกรมสามารถทำได้ง่ายคายนมาก โดยใช้คำสั่งเหล่านี้

DTREnable	สำหรับสั่งให้ขา DTR มีลอจิก “0” หรือ “1”
RTSEnable	สำหรับสั่งให้ขา RTS มีลอจิก “0” หรือ “1”
CTSHolding	สำหรับอ่านค่าสถานะจากขา CTS ว่ามีลอจิก “0” หรือ “1”
CDHolding	สำหรับอ่านค่าสถานะจากขา DCD ว่ามีลอจิก “0” หรือ “1”
DSRHolding	สำหรับอ่านค่าสถานะจากขา DSR ว่ามีลอจิก “0” หรือ “1”
Break	สำหรับการสั่งให้ขา TxD มีลอจิก “0” หรือ “1”

4.13 พอร์ต USB กับวิซวดเบติก

USB เป็นการส่งข้อมูลที่มีรูปแบบการเชื่อมต่อในระบบบัสคือ อุปกรณ์ทุกๆ ตัวจะต้องส่งสัญญาณรวมกันไปในสายส่งสัญญาณเพียงคู่เดียว ดังนั้นอุปกรณ์ทุกๆ ตัวที่เชื่อมต่อกับบัสจะต้องส่งข้อมูลเรียงลำดับกันไปเพื่อไม่ให้เกิดการชนกันของข้อมูล และเนื่องจาก USB เป็นระบบบัสที่ใส่สายส่งสัญญาณเพียงคู่เดียว (2 เส้น) ทำให้ในช่วงเวลาหนึ่งๆ จะมีข้อมูลวิ่งไปได้เพียงทิศทางเดียวเท่านั้น ไม่สามารถเกิดการรับและส่งข้อมูลไปในเวลาเดียวกันได้หรือที่เรียกกันว่า การส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ (half duplex)

เนื่องจากการถ่ายถอดข้อมูลในบัส USB เกิดขึ้นด้วยความเร็วสูง ทำให้ต้องนำเทคนิคการถ่ายถอดสัญญาณรูปแบบต่างๆ มาใช้งานเพื่อให้ข้อมูลส่งไปยังปลายทางได้ถูกต้อง การถ่ายถอดสัญญาณแบบผลต่างเป็นเทคนิคที่นำมาใช้งานเพื่อให้ข้อมูลส่งไปยังปลายทางได้ถูกต้อง การถ่ายถอดสัญญาณแบบผลต่างเป็นเทคนิคที่นำมาใช้เพื่อลดคลื่นรบกวนที่จะแพร่ออกมาจากสายดังที่กล่าวไปแล้ว นอกจากนี้ยังต้องอาศัยการเข้ารหัสแบบ NRZI เพื่อให้ด้านรับทราบถึงจังหวะข้อมูลในแต่ละบิตเพื่อไม่ให้เกิดความผิดพลาด และสามารถตรวจสอบการเชื่อมต่อหรือปลดออกของอุปกรณ์ในบัส จึงต้องมีการกำหนดรูปแบบของสัญญาณต่างๆ เพื่อใช้งานขึ้น

4.13.1 การส่งข้อมูลในสายสัญญาณ

หลังจากข้อมูลถูกเข้ารหัสเรียบร้อยแล้ว ข้อมูลเหล่านั้นก็จะถูกส่งออกไปยังอุปกรณ์ต่างๆที่ต่อกับกับฮับ แต่เนื่องจาก USB ส่งข้อมูลด้วยความเร็วที่สูงมาก โดยใช้สัญญาณที่ยาว (เมื่อเทียบกับสายสัญญาณข้อมูลภายในคอมพิวเตอร์) ทำให้ไม่สามารถใช้การส่งสัญญาณแบบขั้วเดียว (single end) ได้เพราะจะทำให้เกิดการแพร่กระจายของคลื่นแม่เหล็กไฟฟ้าไปกวนอุปกรณ์อื่นๆ และยังทำให้สัญญาณรบกวนจากภายนอกเข้าไปกวนข้อมูลที่ต้องการส่งในสายสัญญาณได้ง่าย

การส่งสัญญาณที่แก้ไขปัญหา 2 ข้อที่กล่าวมาคือการส่งสัญญาณผลต่าง (differential pair signaling) เนื่องจากการส่งสัญญาณด้วยวิธีนี้จะทำให้สนามแม่เหล็กไฟฟ้าที่เกิดขึ้นจากสายสัญญาณทั้งสองเส้นหักล้างกัน เนื่องจากมีศักย์ตรงข้ามกัน จึงไม่เกิดการแพร่กระจายไปรบกวนอุปกรณ์อื่นๆ สัญญาณ 2 เส้นอยู่คู่กัน เมื่อสัญญาณรบกวนเหนี่ยวนำเข้าสู่สายหนึ่งก็จะเหนี่ยวนำเข้าสู่สายอีกเส้นในขนาดที่เท่ากัน เมื่อถึงปลายทางในวงจรภาครับจะใช้วงจรขยายผลต่าง (differential amplifier) เป็นตัวรับสัญญาณ ซึ่งวงจรนี้จะมีคุณสมบัติขยายสัญญาณที่มีขนาดต่างกันและลดทอนสัญญาณที่มีขนาดเท่ากัน ดังนั้นสัญญาณข้อมูลซึ่งมีศักย์ตรงข้ามกัน จะถูกขยายเพื่อส่งต่อไปยังส่วนถัดไป แต่สัญญาณรบกวนซึ่งมีขนาดเท่ากันในสายทั้ง 2 เส้นจะถูกลดทอนหรือตัดทิ้ง

จะเห็นได้ว่าทั้งสองด้านมีวงจรที่เหมือนกันเนื่องจาก USB เป็นบัสที่รับส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ นั่นคือทั้งสองด้านสามารถรับส่งข้อมูลได้ทั้งคู่ แต่คนละช่วงเวลากัน ทำให้ทั้งสองด้านมีชุดวงจรที่เหมือนกัน และเนื่องจากการรับส่งข้อมูลแบบนี้จะมีอุปกรณ์เพียงตัวเดียวที่สามารถรับหรือส่งข้อมูลได้ในช่วงเวลาหนึ่งๆ อุปกรณ์ที่เหลือที่ไม่ได้ใช้งาน บัสจำเป็นจะต้องปล่อยบัสสัญญาณให้มีสภาพอิมพีแดนซ์สูง (high impedance) ดังนั้นวงจรภาครับส่งของอุปกรณ์แต่ละตัวจะต้องสามารถกำหนดสถานะของเอาต์พุตให้เป็นสภาพอิมพีแดนซ์สูงได้

4.13.2 HID มาตรฐานของอุปกรณ์ USB ในระดับเชื่อมต่อกันผู้ใช้งาน

HID Class (Human Interface Device Class) เป็นลักษณะการเชื่อมต่อของอุปกรณ์ USB ที่เน้นไปที่การติดต่อกับมนุษย์ (human) หรือผู้ใช้งานนั่นเอง ซึ่งอุปกรณ์ USB ที่ใช้ระดับการเชื่อมต่อแบบนี้ซึ่งรู้จักกันดีคือ คีย์บอร์ดและเมาส์ ซึ่งจะอธิบายถึงรูปแบบ มาตรฐาน ลักษณะการเชื่อมต่อและถ่ายทอดข้อมูล เนื่องจากในส่วนของการทำงานจะใช้วงจรเชื่อมต่อพอลต์ USB ที่ใช้มาตรฐานการเชื่อมต่อในระดับ HID นี้ ดังนั้นการทำความเข้าใจเกี่ยวกับ HID จึงเป็นสิ่งที่ผู้สนใจเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่านพอร์ต USB ควรทราบ กอปรกับระดับการเชื่อมต่อแบบนี้เป็นระดับการเชื่อมต่อที่สามารถทำความเข้าใจได้ง่ายและเป็นที่ยอมรับนำมาใช้สร้างอุปกรณ์ USB มากที่สุด

4.13.2.1 สิ่งที่ต้องรู้เกี่ยวกับ HID คลาส

1. การเปลี่ยนแปลงของข้อมูลที่เกิดขึ้นในโครงสร้างการทำงานจะเรียกว่า รายงานหรือรีพอร์ตาม ข้อตกลงกำหนดของ HID รีพอร์ท โดยโฮสต์จะทำการรับส่งข้อมูลด้วยการส่งและร้องขอรีพอร์ทในการควบคุมหรือมีลักษณะเป็นการถ่ายถอดสัญญาณแบบอินเตอร์รัปต์ อย่างไรก็ตามรูปแบบของรีพอร์ทนี้ยังไม่มีข้อมูลสรุปชัดเจน แต่ก็มีควมอ่อนค้วสูง ทำให้สามารถรองรับกับข้อมูลได้ทุกรูปแบบ

2. อุปกรณ์ USB สามารถส่งข้อมูลกลับไปยังโฮสต์(ซึ่งส่วนใหญ่คือ คอมพิวเตอร์) ได้ตลอดเวลา โดยไม่สามารถคาดหมายล่วงหน้า ดังนั้นจึงเป็นหน้าที่ของโฮสต์เองที่ต้องคอยตรวจสอบอยู่ตลอดเวลา เพื่อให้สามารถรองรับกับการทำงานของอุปกรณ์ได้ทันทั่วทั้งที่

3. ในการถ่ายถอดสัญญาณหรือข้อมูลบนบัตซ์ของ USB ไม่สามารถทราบถึงอัตราการส่งผ่านข้อมูลได้อย่างแน่นอน สิ่งที่กำหนดได้มีเพียงคาบเวลาในแต่ละทรานแซกชัน และค่าเวลาระหว่างทรานแซกชันนั้นอาจน้อยกว่าก็ได้ อย่างไรก็ตาม ระบบจะพยายามทำให้เกิดอัตราการส่งผ่านข้อมูลเร็วที่สุดเท่าที่จะเป็นไปได้

4. อัตราการถ่ายถอดสัญญาณหรือข้อมูลสูงสุดจะถูกจำกัดไว้ตามประเภทของอุปกรณ์ ซึ่งมีด้วยกัน 3 แบบคือความเร็วต่ำ, ความเร็วเต็มที่ และความเร็วสูง โดยโฮสต์สามารถกำหนดคาบเวลาในการทำงานต่อทรานแซกชันสูงสุดแยกกันไปตามประเภทของอุปกรณ์ โดย

4.1 สำหรับอุปกรณ์ความเร็วต่ำ โฮสต์กำหนดคาบเวลาในการทำงาน 1 ทรานแซกชันสูงสุดไม่เกิน 10 มิลลิวินาที ทำให้อัตราการถ่ายถอดสัญญาณเกิดขึ้นได้สูงสุดไม่เกิน 800 ไบต์ต่อวินาที

4.2 สำหรับอุปกรณ์ความเร็วต่ำ โฮสต์กำหนดคาบเวลาในการทำงาน 1 ทรานแซกชันสูงสุดไม่เกิน 1 มิลลิวินาที ทำให้อัตราการถ่ายถอดสัญญาณเกิดขึ้นได้สูงสุดไม่เกิน 8,000 ไบต์ต่อวินาที

4.2 สำหรับอุปกรณ์ความเร็วต่ำ โฮสต์กำหนดคาบเวลาในการทำงาน 3 ทรานแซกชันสูงสุดไม่เกิน 125 ไมโครวินาที ทำให้อัตราการถ่ายถอดสัญญาณเกิดขึ้นได้สูงสุดไม่เกิน 24.56 เมกะไบต์ต่อวินาที

5. ในระบบปฏิบัติการวินโดวส์ที่ต่ำกว่า 98SE จะไม่รองรับการถ่ายถอดสัญญาณแบบอินเตอร์รัปต์ ดังนั้นการติดต่อจากโฮสต์ไปยังอุปกรณ์จะต้องใช้การถ่ายถอดสัญญาณควบคุมเข้ามาช่วยแทน

4.13.2.2 โครงสร้างคิสทรีปเตอร์

ไม่ว่าจะเป็นอุปกรณ์ USB ในคลาสหรือระดับใด ล้วนแล้วแต่ต้องมีคิสทรีปเตอร์ในการทำงานทั้งสิ้น ใน HID คลาสเองก็เช่นกัน มีคิสทรีปเตอร์เป็นของตัวเองชื่อ HID คิสทรีปเตอร์ ซึ่งภายในคิสทรีปเตอร์นี้จะแบ่งออกเป็น 2 ส่วนคือ คิสทรีปเตอร์รายงานผลการทำงาน หรือ รีพอร์ทคิสทรีปเตอร์ (report descriptor) และกลุ่มคิสทรีปเตอร์ทางกายภาพ หรือ ฟิสิคัลคิสทรีปเตอร์ (physical descriptor) รูปด้านล่างแสดงโครงสร้างคิสทรีปเตอร์ของอุปกรณ์ USB ในระดับ HID จะเห็นได้ว่า HID คิสทรีปเตอร์จะบรรจุอยู่ในอินเตอร์เฟลคิสทรีปเตอร์ร่วมกันเอ็นด์พอยต์คิสทรีปเตอร์

HID คิสกริปเตอร์

- **Length** มีขนาด 1 ไบต์ ใช้ระบุขนาดรวมของคิสกริปเตอร์ในหน่วยไบต์ ใน HID คลาสจะกำหนดให้มีค่าเท่ากับ 09H
- **Descriptor type** มีขนาด 1 ไบต์ ใช้ระบุชนิดของคิสกริปเตอร์ ในกรณีเป็น HID คิสกริปเตอร์จะมีค่าเท่ากับ 21H
- **HID version** มีขนาด 2 ไบต์ กำหนดในลักษณะรหัส BCD ใช้ระบุหมายเลขคุณสมบัติของ HID หรือ อาจกล่าวได้ว่าเป็นการระบุเวอร์ชันก็ได้ โดยค่านี้จะใช้ตัวเลขฐานสิบ 4 ตัว 2 ตัวแรกใช้กำหนดตัวเลขเวอร์ชันหลัก ส่วน 2 ตัวหลังใช้กำหนดตัวเลขเวอร์ชันย่อย คั่นด้วยจุดทศนิยม ยกตัวอย่าง 0100 หมายถึง เวอร์ชัน 1.0 ถ้าเป็นเวอร์ชัน 1.1 ข้อมูลจะเป็น 0110 เป็นต้น
- **CountryCode** มีขนาด 1 ไบต์ ใช้กำหนดรหัสของประเทศที่เกี่ยวข้องกับอุปกรณ์ USB ตัวนั้นๆ ขึ้นตัวอย่างที่เห็นได้ชัดคือ คีย์บอร์ด เนื่องจากในแต่ละประเทศอาจมีการใช้คีย์บอร์ดแตกต่างกัน เช่น ในประเทศไทย, จีน, อังกฤษ เป็นต้น
- **NumDescriptors** มีขนาด 1 ไบต์ ใช้ระบุจำนวนคิสกริปเตอร์ว่าเป็น รีพอร์ต หรือ ฟิสิกส์คิสกริปเตอร์ ถ้าเป็นรีพอร์ตคิสกริปเตอร์จะมีค่าเท่ากับ 22H
- **DescriptorLength** มีขนาด 2 ไบต์ ใช้แจ้งขนาดหรือความยาวของรีพอร์ตคิสกริปเตอร์

4.13.2.3 รีพอร์ตคิสกริปเตอร์ (Report descriptor)

เป็นส่วนประกอบหลักที่อาจกล่าวได้ว่าสำคัญมากที่สุด เนื่องจากข้อมูลที่เก็บอยู่ในคิสกริปเตอร์ตัวนี้เป็นสิ่งที่อธิบายถึงรูปแบบและวิธีการใช้ข้อมูลเพื่อให้บรรจวดูประสงค์ตามที่อยู่อุปกรณ์ USB ตัวนั้นกำหนดไว้ ยกตัวอย่าง หากอุปกรณ์ USB นี้เป็นเมาส์ ข้อมูลในรีพอร์ตคิสกริปเตอร์จะรายงานให้ทราบถึงการเคลื่อนที่ของเมาส์เพื่อระบุตำแหน่ง และสถานะการกำมุขของเมาส์ เป็นต้น

รีพอร์ตคิสกริปเตอร์จะมีความยาวเท่าใดก็ได้ แต่ต้องแจ้งความยาวลงใน DescriptorLength ภายใน HID คิสกริปเตอร์ เพื่อให้โฮสต์ทราบด้วย หลักและเป้าหมายเบื้องต้นของการใช้งานรีพอร์ตคิสกริปเตอร์มีดังนี้

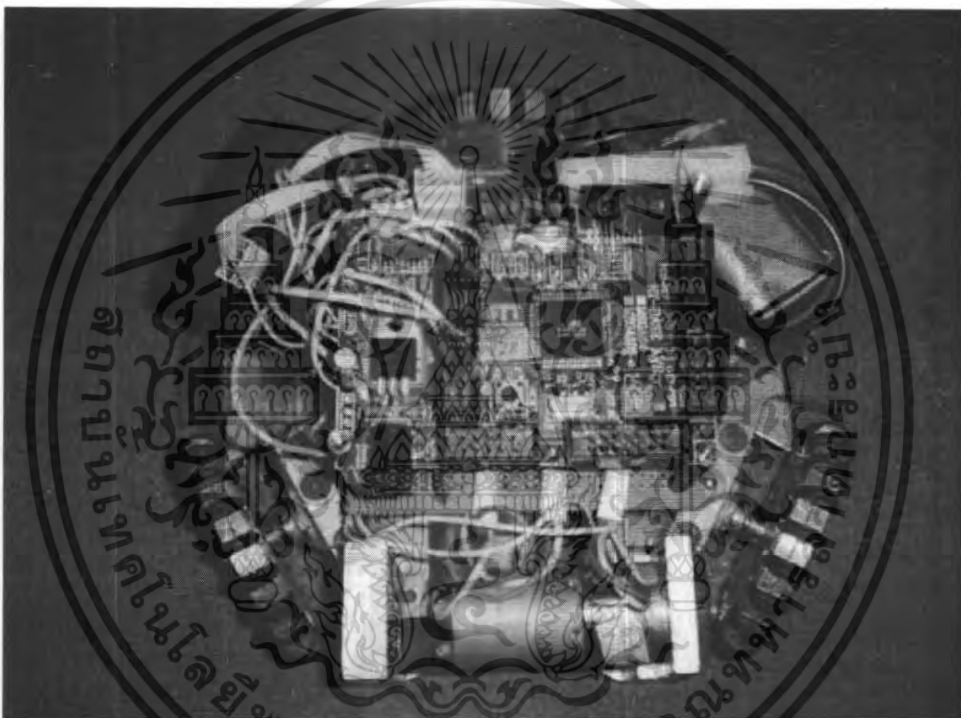
1. ใช้เก็บรายละเอียดข้อมูลของอุปกรณ์ให้ครบถ้วนที่สุดภายใต้พื้นที่ที่เล็กที่สุดเท่าที่เป็นไปได้
2. ขอมให้ซอฟต์แวร์ประยุกต์ที่เกี่ยวข้องสามารถข้ามข้อมูลที่ไม่ชัดเจนไปได้ เพื่อให้ยังคงสามารถดำเนินการต่อไปได้
3. สามารถขยายได้
4. รองรับการจัดเก็บและรวบรวมข้อมูล
5. สามารถอธิบายรายละเอียดด้วยข้อมูลภายในคิสกริปเตอร์เอง เพื่อประโยชน์ในการพัฒนาซอฟต์แวร์ประยุกต์สำหรับใช้งานร่วมกันได้

บทที่ 5

การออกแบบหุ่นยนต์เตะฟุตบอล

5.1 การติดตั้งโครงสร้างของหุ่นยนต์

ส่วนประกอบของหุ่นยนต์จะใช้ล้อ OMNI DIRECTIONAL เพื่อให้รถเคลื่อนที่ได้อิสระรอบทิศทาง ด้วยมอเตอร์ทรงรอบ 12 Volt สามตัวติดตั้งในแนวนอนเพื่อลดพื้นที่ในการติดตั้งและมอเตอร์ขับเคลื่อนโดยตรง พร้อมด้วยชุดปืนลูกบอลให้ติดกับตัวหุ่นด้วย DC MOTOR และชุดยิงลูกบอลด้วย SOLINOID 12 Volt และลักษณะการติดตั้งดังภาพ

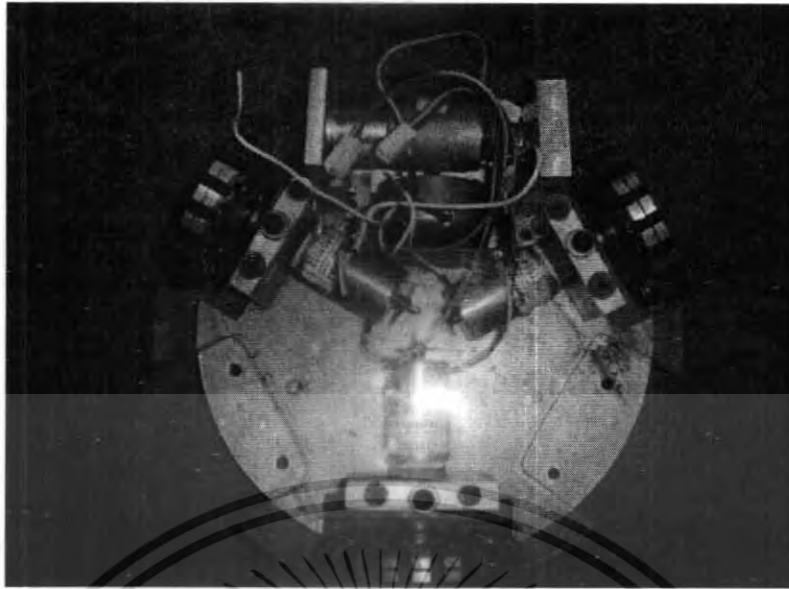


รูปที่ 5-1 แสดงการโครงสร้างของตัวหุ่นยนต์

5.2 การติดตั้งของล้อของตัวหุ่นยนต์

ล้อที่ใช้ในการติดตั้งจะเป็นล้อแบบ OMNI DIRECTIONAL ซึ่งจะเป็นที่สร้างขึ้นเองโดยเฉพาะ เพื่อให้การเคลื่อนที่ได้อิสระและจะใช้ตัวมอเตอร์ขับเคลื่อนโดยตรง โดยที่ไม่ต้องมีชุดเฟืองทดหรือชุดเฟืองเปลี่ยนทิศทาง และทำการติดตั้งแนวนอนเพื่อเป็นการลดความสูงของตัวหุ่นยนต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5-2 แสดงการติดตั้งตัวล๊อซ์ขั้วของตัวหุ่นยนต์

5.3 การติดตั้งตัวเลี้ยงลูกบอลของตัวหุ่นยนต์

การติดตั้งตัวเลี้ยงบอลจะเป็นอย่างไรให้เกิดความหนืดซึ่งเป็นสมบัติของยางในการปั่นบอลเข้ากับตัวหุ่นยนต์ เพื่อให้บอลติดกับตัวหุ่นยนต์ตลอดเวลา โดยปลายด้านหนึ่งของตัวชุดปืนนี้จะต่อเพื่อส่งกำลังเข้ากับมอเตอร์ดังรูปที่ 5-3

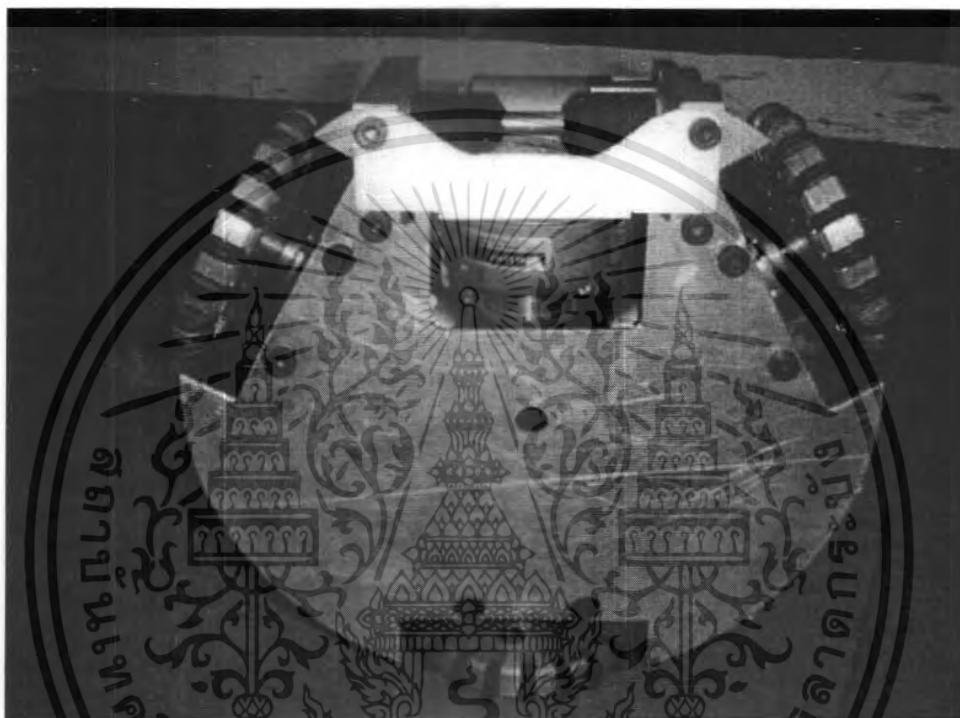


รูปที่ 5-3 แสดงการติดตั้งตัวเลี้ยงบอลของตัวหุ่นยนต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.4 การติดตั้งตัวขึงลูกบอลของตัวหุ่นยนต์

การติดตั้งตัวขึงลูกบอลถือเป็นอุปกรณ์อีกอย่างที่มีความสำคัญในการทำประตู่ ในส่วนอุปกรณ์หลักที่ใช้โซลินอยด์ 12 โวลต์ โดยตำแหน่งที่ติดตั้งนั้นจะอยู่ใต้ตัวถังขึงลูกบอลลงไป และตำแหน่งที่กระทบกับลูกบอลนั้นจะต้องโดนกึ่งกลางของลูกบอลพอดี เพื่อที่การขึงนั้นไม่ให้เกิดการหมุนย้อนกลับ ในส่วนตัวของโซลินอยด์จะต้องมีการติดตัวสปริงให้มีการดึงกลับหลังการขึงแล้วดังรูปที่ 5-4



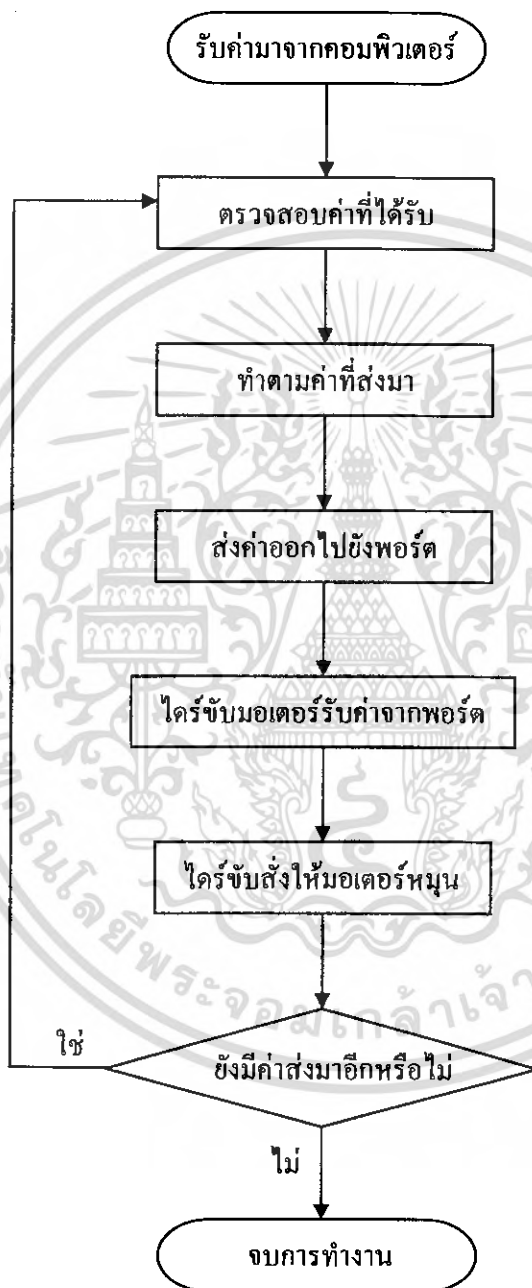
รูปที่ 5-4 แสดงการติดตั้งตัวขึงลูกบอลของตัวหุ่นยนต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6

ระบบควบคุมของหุ่นยนต์เตะฟุตบอล

6.1 การออกแบบโปรแกรมการทำงาน



รูปที่ 6-1 แผนผังลำดับการทำงานของโปรแกรมภาษาซีในไมโครคอนโทรลเลอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 7

การออกแบบโปรแกรมคำนวณพิกัดของตัวหุ่นยนต์เตะฟุตบอล

7.1 สาเหตุที่เลือกใช้โปรแกรมวิซวลเบสิกในการเขียนโปรแกรม

1. เป็นภาษาระดับกลาง มีลักษณะคล้ายภาษาอังกฤษ จึงทำให้ง่ายต่อการทำความเข้าใจและพัฒนาโปรแกรม
2. มีคำสั่งที่ใช้ในการติดต่อสื่อสารกับพอร์ตอนุกรมซึ่งสามารถนำไปควบคุมเครื่องได้
3. มีความสามารถในงานประเภทกราฟฟิกอยู่ในระดับที่น่าพอใจ ทำให้สามารถพัฒนาโปรแกรมได้อย่างสะดวก
4. วิซวลเบสิกใช้พัฒนาโปรแกรมที่ใช้สื่อสารกับผู้ใช้เป็นหลัก ดังนั้นจึงสามารถพัฒนาหน้าตา (Interface) ของโปรแกรมออกมาได้สวยงามน่าใช้

7.2 โปรแกรมคำนวณพิกัดในการเคลื่อนที่

โปรแกรมจึงได้ถูกพัฒนาออกให้ให้ผู้ใช้สามารถใช้งานได้อย่างง่ายดาย โดยโปรแกรมจะแบ่งขั้นตอนต่างๆออกเป็น 3 ส่วน คือส่วนการรับค่าจากกล้อง, ส่วนของการคำนวณค่ามุมของตัวหุ่นและระยะทางที่ทำการลูกบอลและส่วนที่ทำการสร้างค่าของตัวอักษรที่ต้องการจะส่ง หลังจากนั้นก็ส่งออกไปที่พอร์ตอนุกรมได้เลข โดยมีการส่งออกมาที่ระตัวอักษรในระยะเวลาที่สามารถกำหนดได้ส่วนทางด้านไมโครคอนโทรลเลอร์ก็จะรับค่าที่ได้มาแปรเป็นลักษณะการทำงานที่เราต้องการเพื่อให้ตัวหุ่นเคลื่อนที่ ในส่วนของการใช้งานพอร์ตยูเอสบีก็จะเขียนโปรแกรมติดต่อเหมือนกันแต่ว่าจะส่งค่าตัวแปรโดยตรงโดยไม่ต้องคำนวณหาพิกัดที่ได้

ในการเคลื่อนที่ของหุ่นเองยังคงเป็นการเคลื่อนที่แบบแยกส่วนของการเลี้ยวและการเดินตรงออกจากกันจึงทำให้การเคลื่อนที่เข้าหาลูกบอลเป็นไปด้วยความไม่รวดเร็วซึ่งสามารถพัฒนาให้ทำงานคู่กันได้ต่อไป

7.3 หลักการทำงานของโปรแกรม

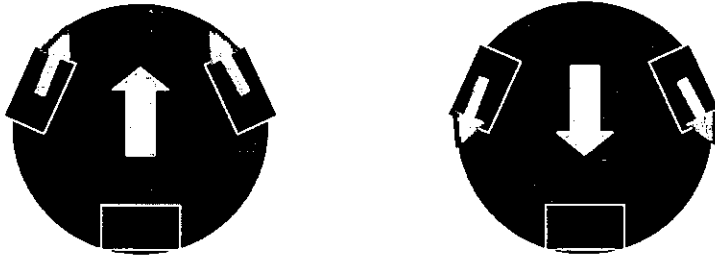
หลังจากที่เราได้ทำการเปิดโปรแกรมจะพบกับหน้าต่างของรูปภาพสนามฟุตบอลซึ่งเราจะเริ่มต้นตั้งตัวหุ่นให้หันหน้าออกไปยังฝั่งตรงข้ามในตำแหน่งประตูของตัวเอง แล้วกล้องจะทำการจับภาพและส่งสัญญาณภาพไปยังคอมพิวเตอร์และคอมพิวเตอร์ก็จะส่งภาพที่ได้มา โปรแกรมจะทำการคำนวณหามุมของตัวหุ่นที่กระทำกับลูกฟุตบอลแล้วเปลี่ยนค่าของมุมเป็นค่าของตัวอักษรซึ่งจะแปรผันตามค่าของมุมนั้นเมื่อทำการเชื่อมมูมเรียบร้อยแล้วก็จะทำการหาระยะทางของตัวหุ่นกับลูกบอล ซึ่งก็จะเปรียบเทียบค่าของตัวเลขระยะทางที่ได้มาเป็นตัวอักษรเหมือนกันกับการหามุม ซึ่งการกระทำเช่นนี้จะเป็นลักษณะของกระทำแบบเวกเตอร์ ก็จะเป็นแบบทางต่อหัวโดยเลื่อนตำแหน่งของการคำนวณไปเรื่อยๆ จนถึงค่าสุดท้ายค่าของตัวอักษร

ที่ต้องการส่งไปหลังจากนั้นคอมพิวเตอร์ก็จะส่งการไปยัง ไมโครคอนโทรลเลอร์ที่ตัวหุ่นจากนั้นหุ่นก็จะเคลื่อนที่ไปยังจุดที่เราต้องการ

ตัวอักษรที่ ต้องการจะส่ง	สถานการณ์เคลื่อนที่ ของหุ่นยนต์	การทำงานของมอเตอร์
8	หุ่นยนต์ 4 เดินหน้า	มอเตอร์หน้าซ้ายหมุนทวนเข็มนาฬิกาและมอเตอร์หน้าขวาหมุนตามเข็มนาฬิกา
2	หุ่นยนต์ 4 ถอยหลัง	มอเตอร์หน้าซ้ายหมุนตามเข็มนาฬิกาและมอเตอร์หน้าขวาหมุนทวนเข็มนาฬิกา
4	หุ่นยนต์ 4 หมุนซ้าย	มอเตอร์หลังหมุนตามเข็มนาฬิกา
6	หุ่นยนต์ 4 หมุนขวา	มอเตอร์หลังหมุนทวนเข็มนาฬิกา
5	หุ่นยนต์ 4 หยุด	มอเตอร์ทั้ง 3 หยุดหมุน
T	หุ่นยนต์ 5 เดินหน้า	มอเตอร์หน้าซ้ายหมุนทวนเข็มนาฬิกาและมอเตอร์หน้าขวาหมุนตามเข็มนาฬิกา
R	หุ่นยนต์ 5 เดินหน้าซ้าย	มอเตอร์หลังหมุนทวนเข็มนาฬิกาและมอเตอร์หน้าขวาหมุนตามเข็มนาฬิกา
Y	หุ่นยนต์ 5 เดินหน้าขวา	มอเตอร์หลังหมุนตามเข็มนาฬิกาและมอเตอร์หน้าซ้ายหมุนทวนเข็มนาฬิกา
B	หุ่นยนต์ 5 ถอยหลัง	มอเตอร์หน้าซ้ายหมุนตามเข็มนาฬิกาและมอเตอร์หน้าขวาหมุนทวนเข็มนาฬิกา
V	หุ่นยนต์ 5 ถอยหลังซ้าย	มอเตอร์หลังหมุนทวนเข็มนาฬิกาและมอเตอร์หน้าซ้ายหมุนตามเข็มนาฬิกา
N	หุ่นยนต์ 5 ถอยหลังขวา	มอเตอร์หลังหมุนตามเข็มนาฬิกาและมอเตอร์หน้าขวาหมุนทวนเข็มนาฬิกา
F	หุ่นยนต์ 5 หมุนซ้าย	มอเตอร์ทั้ง 3 หมุนตามเข็มนาฬิกา
H	หุ่นยนต์ 5 หมุนขวา	มอเตอร์ทั้ง 3 หมุนทวนเข็มนาฬิกา
G	หุ่นยนต์ 5 หยุด	มอเตอร์ทั้ง 3 หยุดหมุน

ตารางที่ 7-1 แสดงตัวอักษรและสถานะของการเคลื่อนที่ของตัวหุ่นเมื่อเราส่งค่าออกไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7-1 แสดงลักษณะการเดินหน้าและถอยหลังตรงของหุ่นยนต์



รูปที่ 7-2 แสดงลักษณะการหมุนเลี้ยวของหุ่นยนต์



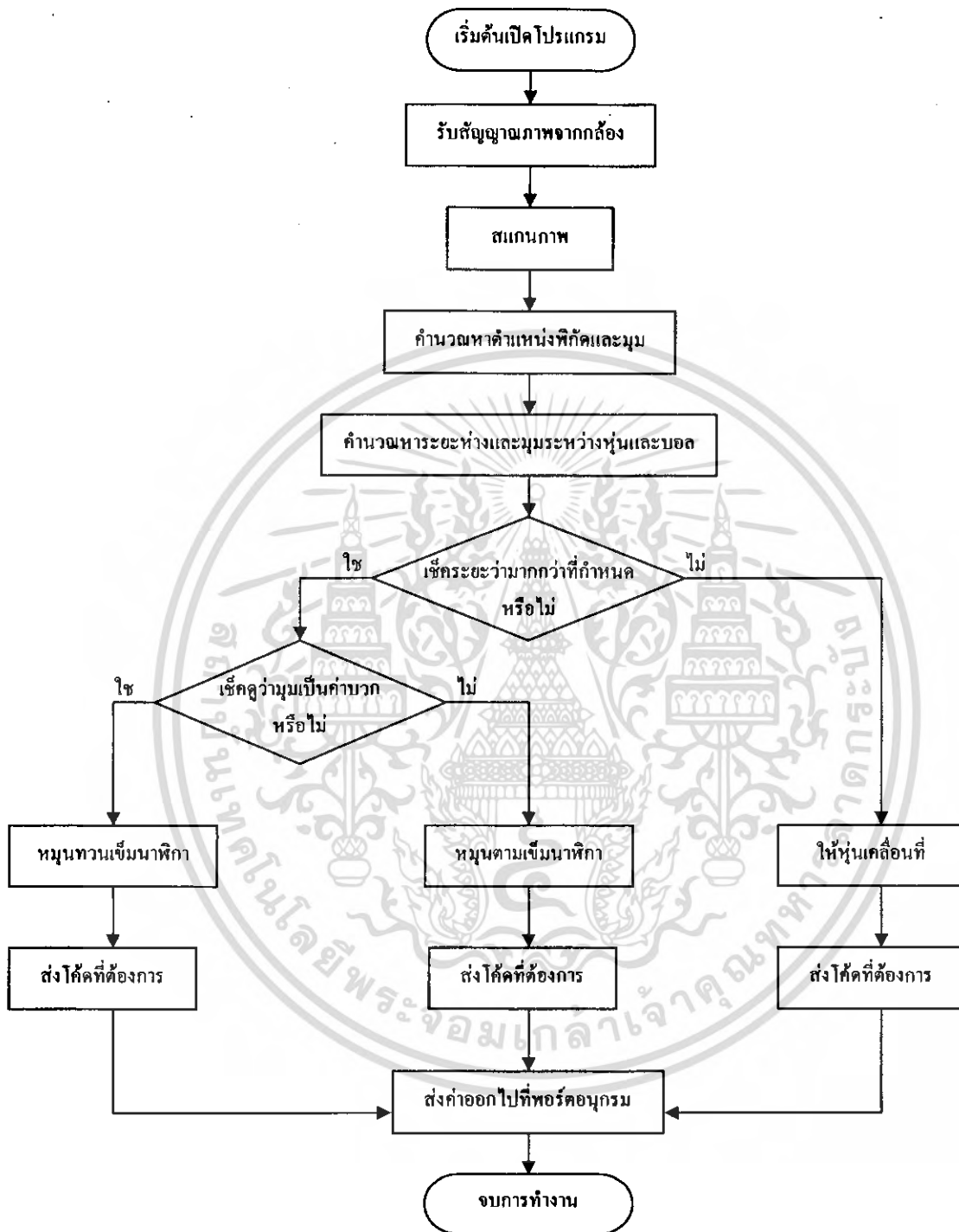
รูปที่ 7-3 แสดงลักษณะการเดินหน้าเฉียงของหุ่นยนต์



รูปที่ 7-4 แสดงลักษณะการถอยหลังเฉียงของหุ่นยนต์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.4 แผนผังลำดับการทำงานของโปรแกรม



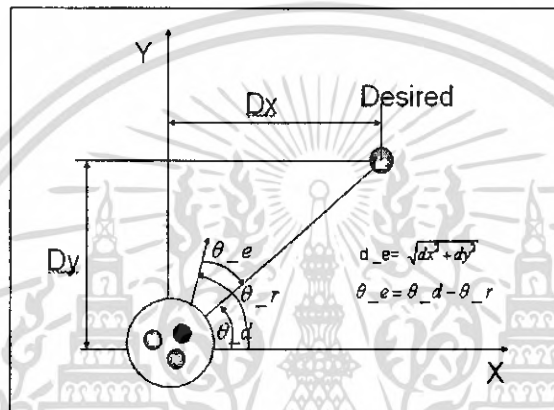
รูปที่ 7-5 แผนผังลำดับการทำงานของโปรแกรมวิชวลเบสิกในคอมพิวเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

7.5 โปรแกรมการสร้างโค้ด(Generat Code Program)

จากการที่รูปภาพในคอมพิวเตอร์เกิดจากการนำจุดสีขนาดเล็กหรือที่เรียกกันว่า Pixels หลายๆจุดมาเรียงต่อกัน ประกอบกันเป็นภาพ เมื่อเราใช้ฟังก์ชันสแกนภาพที่ได้มาจากกล้องจะมีการแสดงค่าของสีซึ่งค่าสีในแต่ละจุดจะถูกจัดเก็บในระบบตัวเลข (Digital) และค่าของพิกัดในแกน x และ y ของตัวโปรแกรมเอง ซึ่งเราจะทำการเก็บค่าเฉพาะพิกัดมาคำนวณค่าของมุมและระยะทางที่เราต้องการได้

โปรแกรมคำนวณตำแหน่งจุดที่จะเคลื่อนที่จะอาศัยกฎของปีทาโกรัสเปรียบเทียบกับค่าที่เราต้องการและนำเอาค่าความแตกต่างของมุมมาสร้างโค้ดที่ต้องการซึ่งจะทำให้หุ่นเคลื่อนที่ไปในตำแหน่งที่เราต้องการได้



รูปที่ 7-6 แสดงวิธีการคำนวณหามุมเพื่อสร้างโค้ดต่างๆ

ตัวอย่างโค้ด FTHBG

ความหมาย

- F : หมุนตัวหุ่นทวนเข็มนาฬิกา
- T : หุ่นเดินหน้าตรง
- H : หมุนตัวหุ่นตามเข็มนาฬิกา
- B : หุ่นถอยหลังตรง
- G : หยุดการเคลื่อนที่ของตัวหุ่น

บทที่ 8

รูปแบบและผลการทดลอง

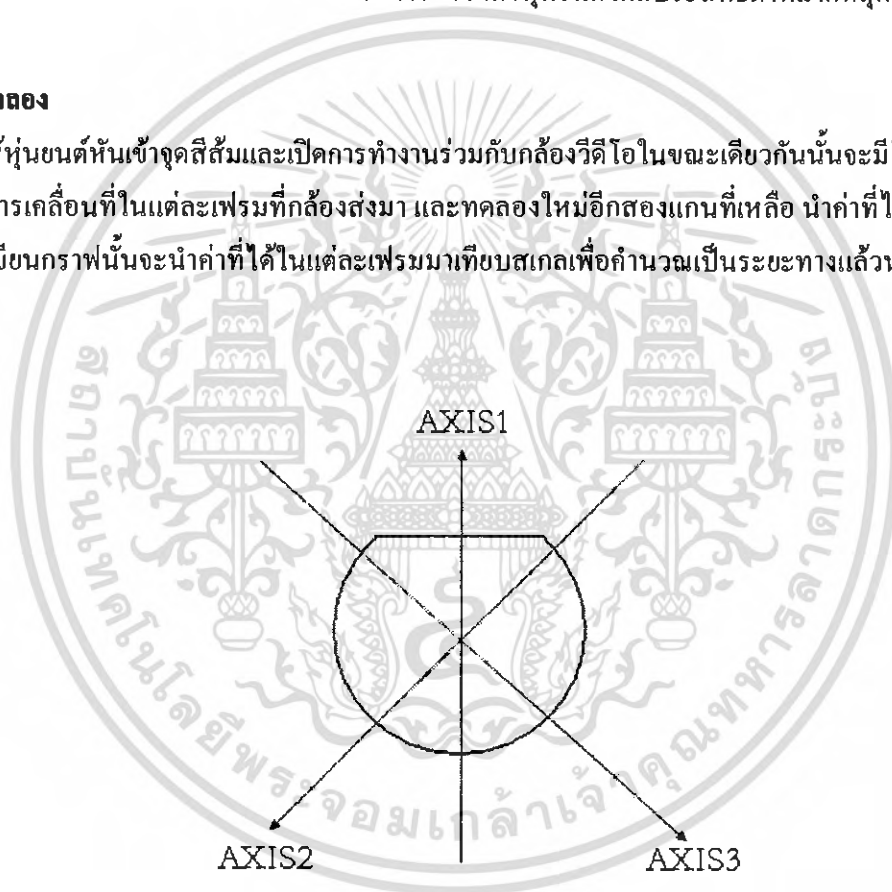
8.1 รูปแบบการทดลอง

รูปแบบของการทดลองเพื่อหาประสิทธิภาพรวมของเครื่องสร้างแบบสำหรับหุ่นยนต์เตะฟุตบอลนี้ คือ ส่วนของการเคลื่อนที่ของตัวหุ่นยนต์ที่เคลื่อนที่เป็นเส้นตรง

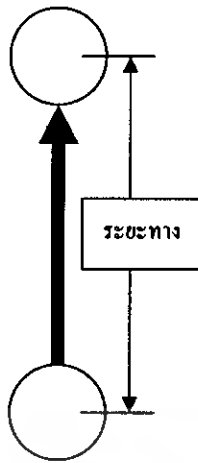
การทดลองนี้จะเป็นการตรวจเช็คว่าการทำงานของตัวหุ่นยนต์ยังมีข้อบกพร่องที่จะต้องแก้ไขบ้างหรือไม่ และจะทำการแก้ไขอย่างไรจึงจะทำให้การทำงานของตัวหุ่นยนต์ได้มีประสิทธิภาพมากที่สุด

8.2 วิธีการทดลอง

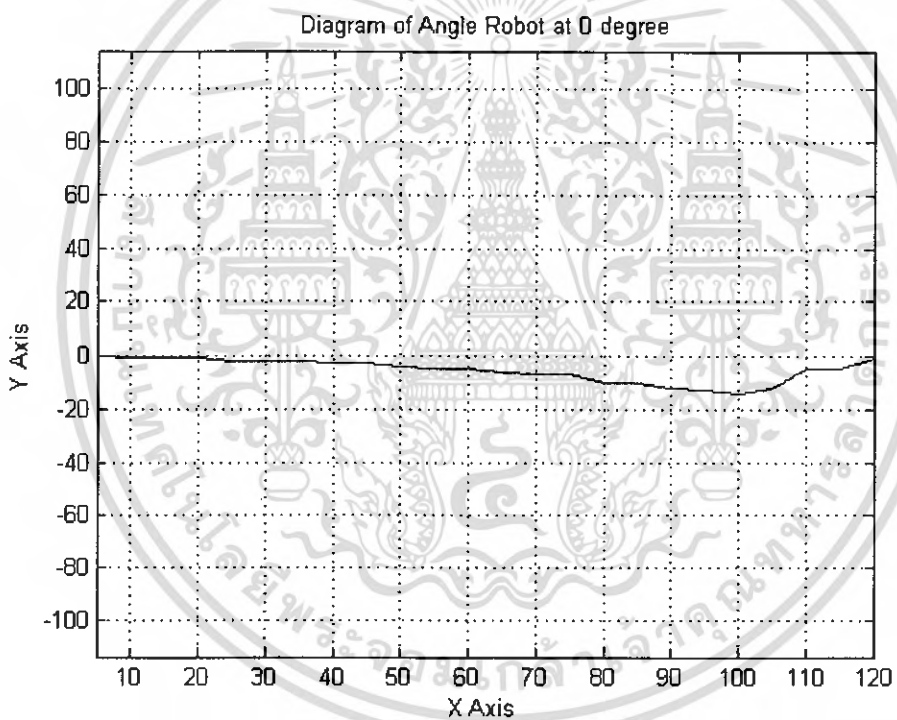
จัดให้หุ่นยนต์หันเข้าจุดสีส้มและเปิดการทำงานร่วมกับกล้องวิดีโอในขณะที่เดียวกันนั้นจะมีโปรแกรมเก็บตำแหน่งการเคลื่อนที่ในแต่ละเฟรมที่กล้องส่งมา และทดลองใหม่อีกสองแกนที่เหลือ นำค่าที่ได้ไปเขียนกราฟ ในการเขียนกราฟนั้นจะนำค่าที่ได้ในแต่ละเฟรมมาเทียบสเกลเพื่อคำนวณเป็นระยะทางแล้วนำมาเขียนกราฟ



รูปที่ 8-1 แสดงแกนการเคลื่อนที่ของหุ่นยนต์ที่ใช้ในการทดลอง

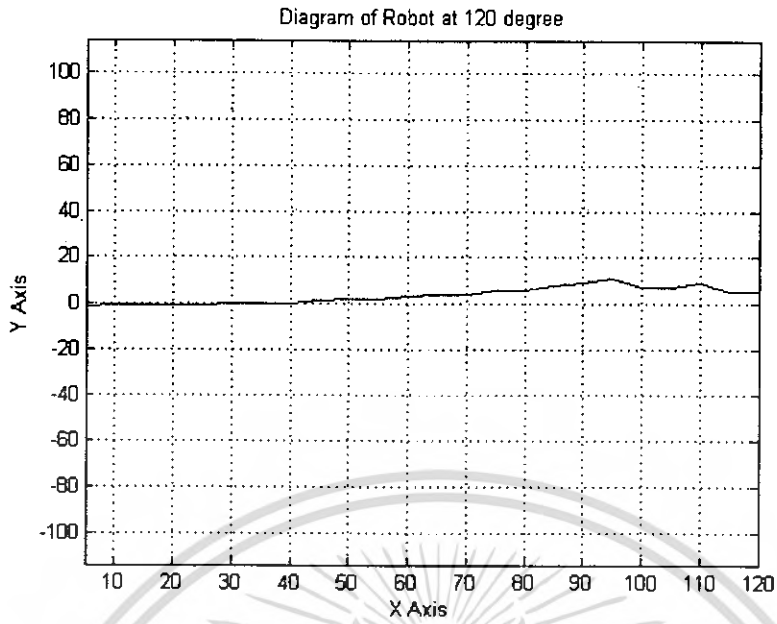


รูปที่ 8-2 แสดงถึงลักษณะการเคลื่อนที่แบบเส้นตรงที่ใช้ในการทดลอง

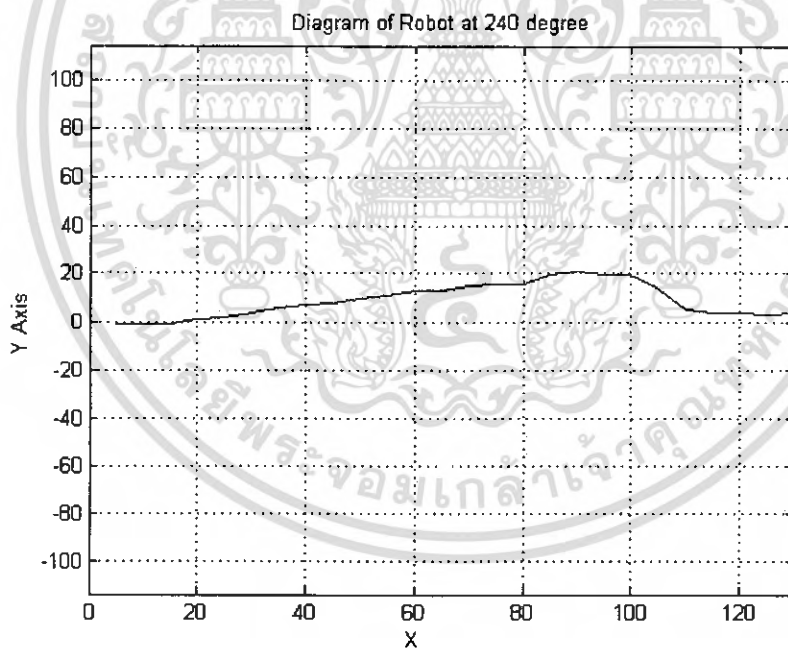


รูปที่ 8-3 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ จากผลการทดลองในแกนที่ 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8-4 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ จากผลการทดลองในแกนที่ 2



รูปที่ 8-5 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ จากผลการทดลองในแกนที่ 3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.3 ผลการทดลอง

การวางตำแหน่งของอุปกรณ์ค่อนข้างลงตัวและเป็นไปตามที่กติกากำหนด หุ่นยนต์สามารถเข้าหาจุดได้โดยไม่ต้องมีการบังคับ และจากการทดลองพบว่าเมื่อหุ่นยนต์เคลื่อนจะมีการวิ่งออกนอกเส้นทางเล็กน้อย อันเนื่องมาจากความผิดพลาดในการขั้นตอนการสร้างและการประกอบ แต่หุ่นยนต์จะสามารถกลับเข้าสู่เส้นทางได้เพราะใช้การควบคุมแบบปิด



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 9

วิเคราะห์และสรุปผลการทดลอง

9.1 การวิเคราะห์ผลการทดลองและสรุปผลการทดลอง

การวิเคราะห์และสรุปผลการทดลองแบ่งออกเป็น 3 ส่วนคือ

9.1.1 ส่วนโปรแกรมควบคุมการเคลื่อนที่ของตัวหุ่น

เนื่องจากการทำงานเป็นแบบทำงานทีละส่วนคือเดินหน้าแล้วหมุนไปเรื่อยๆ จึงทำให้ความเร็วในการเคลื่อนที่ของตัวหุ่นไปยังลูกบอลยังไม่เร็วพอ

คำสั่งที่ใช้ในการเขียนโปรแกรมบางส่วนมีประสิทธิภาพไม่สูงนัก เนื่องด้วยเวลาที่จำกัดผู้จัดเลือกจึงจะใช้คำสั่งพื้นฐานเป็นส่วนใหญ่ แม้ว่าจะให้ผลออกมาใกล้เคียงกันแต่คำสั่งที่เขียนขึ้นนั้นมีสภาพการทำงานที่ไม่เร็วนักหรือมีปัญหาได้ในบางกรณี

คำผิดพลาดในการควบคุมการเคลื่อนที่ของหุ่นยนต์เกิดจากแรงเฉื่อยของหุ่นยนต์เอง

จากโปรแกรมคำนวณตำแหน่งบางครั้งถ้าเราเลือกใช้ค่าในระดับเท่าสนามจริงจะทำให้ค่าของมุมหาค่าไม่ได้เพราะ $\arctan = 0$ หากค่าไม่ได้จึงควรเขียนโปรแกรมป้องกันไว้ก่อน

จากแสงสว่างของห้องและกล้อง ไม่เหมาะสมทำให้สีที่ปรากฏในคอมพิวเตอร์เพี้ยนจากสีจริงทำให้ผลการสแกนสีออกมาผิดพลาดและการระบุตำแหน่งก็ผิดพลาดไปด้วย

9.1.2 ส่วนของแผงควบคุม

เนื่องจากวงจรควบคุมออกแบบมาเพื่อใช้งานง่ายและสะดวก ดังนั้นจึงขนาดใหญ่ นอกจากนี้ยังมีการเชื่อมโยงสายสัญญาณต่างๆระหว่างแผงอีกด้วย ทำให้เกิดการสับสนวุ่นวายเป็นอย่างมาก

แผงวงจรบางส่วนถูกออกแบบตามทฤษฎี ทำให้วงจรบางส่วนไม่สามารถทำงานได้ตามที่ออกแบบไว้เนื่องจากปัจจัยอื่นๆที่ไม่ได้คาดคิดไว้ก่อน

9.1.3 ตัวโครงสร้างของหุ่นยนต์เตะฟุตบอล

เนื่องจากชิ้นส่วนส่วนใหญ่ทำด้วยมือจึงทำให้มีประสิทธิภาพน้อยกว่าที่ต้องการ จึงทำให้ตัวหุ่นยนต์เองไม่ค่อยมีประสิทธิภาพเท่าที่ควร

แม้ว่าชิ้นส่วนแต่ละชิ้นจะถูกออกแบบมาให้ง่ายต่อการสร้างและติดตั้ง แต่ว่าเนื่องจากประสบการณ์และเครื่องมือที่ใช้ยังไม่ค่อยมีประสิทธิภาพเท่าที่ควรจึงทำให้เกิดอุปสรรคบ้าง

เนื่องจากความเสียดทานและภาระของมอเตอร์แต่ละตัวมีค่าไม่เท่ากัน

9.2 แนวทางการแก้ไข

แนวทางการแก้ไขจะแบ่งการวิเคราะห์ออกเป็น 3 ส่วนคือ

9.2.1 ส่วนโปรแกรมควบคุมการเคลื่อนที่

เนื่องจากการทำงานเป็นแบบไล่อ่านสีทีละพิกเซล ดังนั้นควรมีปรับปรุงให้การเคลื่อนที่เป็นการวิ่งตามเส้นแทน อาจจะใช้การคำนวณเชิงตัวเลขเข้ามาช่วยเพื่อลดค่าความผิดพลาดของการเคลื่อนที่ของตัวหุ่นด้วยซึ่งจะส่งผลกับค่าของตัวโค้ดที่ส่งไปแม่นยำมากขึ้น

คำสั่งที่ใช้ในการเขียนโปรแกรมบางส่วนอาจมีประสิทธิภาพไม่สูงนัก เนื่องด้วยเวลาที่จำกัดผู้จัดทำจึงเลือกที่จะใช้คำสั่งพื้นฐานเป็นส่วนใหญ่ แม้ว่าจะให้ผลออกมาใกล้เคียงกันแต่คำสั่งที่เขียนขึ้นนั้นอาจทำงานช้าหรือมีปัญหาได้ในบางกรณี ดังนั้นในการพัฒนาเครื่อง

ในการเขียนโปรแกรมควบคุมหุ่นยนต์ ต้องอาศัยความรู้ประสบการณ์และความชำนาญของผู้เขียน ดังนั้นผู้เขียนโปรแกรมควบคุมหุ่นยนต์จะต้องศึกษาและทำความเข้าใจในตัวโปรแกรมมาเป็นอย่างดีและจะต้องเผื่อค่าผิดพลาดเข้าไปในตัวโปรแกรมด้วย

9.1.2 ส่วนของแผงควบคุม

เนื่องจากวงจรควบคุมต้องควบคุมอุปกรณ์จำนวนมากแต่ต้องอยู่ในพื้นที่ที่จำกัดจึงทำให้การวางตำแหน่งที่ต้องการ ไม่ได้จึงต้องออกแบบวงจรให้เหมาะสมกับตำแหน่งที่สามารถจะวางได้

แผงวงจรบางส่วนถูกออกแบบตามทฤษฎี ทำให้วงจรบางส่วนไม่สามารถทำงานได้ตามที่ออกแบบไว้เนื่องจากปัจจัยอื่นๆที่ไม่ได้คาดคิดไว้ก่อน ดังนั้นแผงวงจรจึงควรได้รับการออกแบบหรือคำแนะนำจากผู้ที่มีประสบการณ์

9.1.3 ส่วนของโครงสร้าง

เนื่องจากงบประมาณและเวลาที่มีจำกัด ทำให้ชิ้นส่วนแต่ละชิ้นถูกออกแบบมาให้สามารถสร้างได้โดยง่าย วัสดุหรืออุปกรณ์ที่เหมาะสมกับตัวหุ่นไม่ก่อกองจะมีตามท้องตลาด และอุปกรณ์บางตัวเป็นชิ้นส่วนที่หาได้รอบตัว ดังนั้นเครื่องอาจดูไม่สวยงามมากนัก แม้ว่าชิ้นส่วนแต่ละชิ้นจะถูกออกแบบมาให้ง่ายต่อการผลิตแต่ในความเป็นจริงแล้วการผลิตด้วยมือจะทำให้เกิดความคลาดเคลื่อนขึ้นทำให้ประสิทธิภาพบางส่วนลดลง ดังนั้นหากต้องการสร้างเพื่อการแข่งขันแล้วอาจจะต้องออกแบบบางส่วนใหม่ให้สามารถทำงานได้ดีขึ้น และชิ้นส่วนควรสร้างด้วยเครื่องจักรอัตโนมัติ เพื่อลดความคลาดเคลื่อนที่เกิดขึ้น กล้องที่ใช้ควรจะมีอัตราเฟรม 40-60 เฟรมต่อวินาที และห้องที่ใช้ทดสอบควรจะเป็นห้องที่ปิดแสงสว่างจากหลอดไฟฟ้าเพียงอย่างเดียว เพื่อให้แสงสว่างคงที่

บรรณานุกรม

- [1] อภิชาติ ภู่อปลับ, “เริ่มต้นเขียนโปรแกรมติดต่อกันและควบคุมฮาร์ดแวร์ด้วย Visual Basic 6”, Infopress Develop Book, 2546
- [2] ธนพล ฉันทวีชัย, “ปฏิบัติการ Visual Basic สำหรับ Common Windows”, ซีเอ็ดยูเคชั่น 2546
- [3] โชติพันธุ์ หล่อเลิศสุนทรและฐิตะพันธุ์ หล่อเลิศสุนทร, “สอนเขียน Visual Basic 6.0 ให้เป็น Project”, Soft Express & Publishing, 2543
- [4] สมศักดิ์ ศรีขจรเกียรติ, “Visual Basic 6. Teach Yourself”, บิบลีโอไฟล์, 2542
- [5] “Visual Basic Graphic Programming”, John Wiley & sons, Inc, 1997
- [6] A.F. Bakker, “Design of a high speed low friction XY-table”, Phillips Centre For industrial Technology (CFT)
- [7] Hassian Z. Tameem, “Design and development of x-y positioning table using stepper motor and belt drive”, Department of Mechanical and Production Engineering Yeshwantrao Chaven college of engineering Wanadongri, Nagpur
- [8] วรพจน์ กรแก้ววัฒนกุล, ชัยวัฒน์ ลิ้มพรจิตรวิไล, “เรียนรู้และปฏิบัติการ ไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชฉบับ AT89C5X ของ Atmel.” Innovative Experiment Co., Ltd.
- [9] Joseph E. Shigley, Charles R. Mischke, Richard G. Budynas, “Mechanical Engineering Design”, Seventh Edition, McGrawHill.
- [10] www.thaiio.com
- [11] www.pantip.com
- [12] www.howstuffwork.com
- [13] www.vbcode.com
- [14] www.download.com
- [15] www.google.co.th

ภาคผนวก ก
วิชาวัดเบสิกซอร์ซโค้ด (Visual Basic Source Code)

***** VISION*****

Option Explicit

Private DataReceived() As VISIONDATA ' Vision data received from clients

Private ReceivedCount As Integer ' Number of data received from clients

Private sendDataTAI As Boolean

Private Sub chkRequest_Click()

 If (chkRequest.Value = vbChecked) Then

 cmdFind_Click

 'findai_Click

 tmrRequest.Enabled = True

 tmrReset.Enabled = True

 tmrFind.Enabled = True

 Else

 tmrReset.Enabled = False

 tmrRequest.Enabled = False

 tmrFind.Enabled = False

 End If

 MSComm1.Settings = "9600,n,8,1"

 If MSComm1.CommPort = 1 Then

 Else

 MSComm1.CommPort = 1

 End If

 'MSComm1.OutBufferSize = 1

 If MSComm1.PortOpen = True Then

```

Else
    MSComm1.PortOpen = True
End If
End Sub

Private Sub cmdFind_Click() ' Broadcast to find clients and find ai
    lstClient.Clear
    sockMain.RemoteHost = frmMain_BROADCAST
    sockMain.sendData "HELLO:" + sockMain.LocalIP
End Sub

Private Sub cmdMonitor_Click()
    frmMonitor.Show
    frmMonitor.Left = Me.Left
    frmMonitor.Top = Me.Top + Me.Height
    frmMonitor.rsMain.Redraw
End Sub

Private Sub cmdReset_Click()
    frmMonitor.rsMain.Reset
End Sub

Private Sub cmdSettings_Click()
    Load frmSettings
    frmSettings.Show vbModal

    ' Update
    tmrRuntime.Interval = frmMain_UPDATEDATA
    tmrRequest.Interval = frmMain_REQUEST
    tmrFind.Interval = frmMain_FINDCLIENT
    lblUnit.Caption = "(In " + UNIT_STRING + ")"
End Sub

Private Sub cmdStart_Click()

```

```

Dim result As Long
If (Text1.Text <> "") Then
    result = startConnection(Text1.hWnd)
Else
    MsgBox "please enter ip"
End If
' Timer1.Enabled = True

```

```

sendDataTAI = True

```

```

End Sub

```

```

Private Sub cmdStop_Click()

```

```

    Dim result As Long

```

```

    'Timer1.Enabled = False

```

```

    sendDataTAI = False

```

```

    result = closeConnection()

```

```

End Sub

```

```

Private Sub Command5_Click()

```

```

    Dim Sent_Code_Close As String

```

```

    chkRequest.Value = 0

```

```

    Sent_Code_Close = 5

```

```

    MSComm1.Output = Sent_Code_Close

```

```

    MSComm1.PortOpen = False

```

```

End Sub

```

```

Private Sub findai_Click()

```

```

    'List1.Clear

```

```

'Winsock1.RemoteHost = frmMain_BROADCAST
'Winsock1.SendData "HIAI:" + Winsock1.LocalIP
End Sub

Private Sub Form_Load()
' Init form & controls
Text1.Text = ""
sendDataTAI = False

sockMain.RemotePort = frmMain_REMOTEPORT
sockMain.RemoteHost = frmMain_BROADCAST
sockMain.Bind frmMain_LOCALPORT

'Winsock1.RemotePort = frmMain_AIREMPORT 'add
'Winsock1.RemoteHost = frmMain_BROADCAST 'add
'Winsock1.Bind frmMain_AIPORT 'add

mnuLoad_Click ' Load settings
tmrRuntime.Interval = frmMain_UPDATEDATA
tmrRequest.Interval = frmMain_REQUEST
tmrFind.Interval = frmMain_FINDCLIENT
lblUnit.Caption = "(In " + UNIT_STRING + ")"

vsbFindClient_Change
vsbRate_Change

StartTime = Now
cmdFind_Click ' Find clients
'findai_Click
End Sub

Private Sub Form_Unload(Cancel As Integer)
sockMain.Close

```

```
'Winsock1.Close 'add
```

```
mnuSave_Click
```

```
'End
```

```
End Sub
```

```
Public Sub UpdateData() ' Update data and controls
```

```
'Write Code Doy na
```

```
' Update data
```

```
RunTime = Now - StartTime
```

```
' Update controls
```

```
lblRunTime.Caption = "Running Time: " + Format$(Hour(RunTime), "0") + ":" +  
Format$(Minute(RunTime), "00") + ":" + Format$(Second(RunTime), "00")
```

```
lblReceive.Caption = "Updated: " + Format$(UpdateCount, "0")
```

```
' Update Robocup Status
```

```
Dim i As Integer
```

```
Dim Robot As ROBOTDATA
```

```
Dim object As OBJECTDATA
```

```
Dim Degree_Ball0 As Integer
```

```
Dim Degree_Robot0 As Integer
```

```
Dim Pos_Robot_X0 As Integer
```

```
Dim Pos_Robot_Y0 As Integer
```

```
Dim Pos_Ball_X As Integer
```

```
Dim Pos_Ball_Y As Integer
```

```
Dim del_x As Integer
```

```
Dim del_y As Integer
```

```
Dim Beta As Double
```

```
Dim Zeta As Double
```

```
Dim Alfa As Double
```

```

Dim del_x0 As Integer
Dim del_y0 As Integer
Dim dis0 As Double
Dim Degree_Ball1 As Integer
Dim Degree_Robot1 As Integer
Dim Pos_Robot_X1 As Integer
Dim Pos_Robot_Y1 As Integer
Dim del_x1 As Integer
Dim del_y1 As Integer
Dim dis1 As Double
Dim Sent_Code As String
Dim X0 As Double, X1 As Double
Dim Y0 As Double, Y1 As Double

'count data
For j = 0 To 4
    ' Team's robots
    frmMonitor.rsMain.GetRobot i, Robot.x, Robot.y, Robot.dir, Robot.valid
    If (Robot.valid) Then
        lblRobot(i).Caption = "Robot " + Format$(i + 1, "0") + ": (" + Format$(Robot.x *
(frmMain_FIELD_MAXX), "0") + ", " + Format$(Robot.y * (frmMain_FIELD_MAXY), "0") + ") @ " +
Format$(Robot.dir * 180 / PI, "0")
    Else
        lblRobot(i).Caption = "Robot " + Format$(i + 1, "0") + ": " + frmMain_NATEXT
    End If

    ' Opponents
    frmMonitor.rsMain.GetOpponent i, object.x, object.y, object.valid
    If (object.valid) Then

```

```

lblOpponent(i).Caption = "Opponent " + Format$(i + 1, "0") + ": (" + Format$(object.x *
(frmMain_FIELD_MAXX), "0") + ", " + Format$(object.y * (frmMain_FIELD_MAXY), "0") + ")"
Else
lblOpponent(i).Caption = "Opponent " + Format$(i + 1, "0") + ": " + frmMain_NATEXT
End If
If i = 3 Then
'Degree Robot Use Compute
Degree_Robot0 = Format$(Robot.dir * 180 / PI)
Pos_Robot_Y0 = Format$(Robot.y * (frmMain_FIELD_MAXY))
Pos_Robot_X0 = Format$(Robot.x * (frmMain_FIELD_MAXX))
End If
If i = 4 Then
'Degree Robot Use Compute
Degree_Robot1 = Format$(Robot.dir * 180 / PI)
Pos_Robot_Y1 = Format$(Robot.y * (frmMain_FIELD_MAXY))
Pos_Robot_X1 = Format$(Robot.x * (frmMain_FIELD_MAXX))
End If
Next i

' Ball
frmMonitor.rsMain.GetBall object.x, object.y, object.valid
If (object.valid) Then
lblBall.Caption = "Ball " + ": (" + Format$(object.x * (frmMain_FIELD_MAXX), "0") + ", " +
Format$(object.y * (frmMain_FIELD_MAXY), "0") + ")"
Else
lblBall.Caption = "Ball " + ": " + frmMain_NATEXT
End If
Pos_Ball_X = Format$(object.x * (frmMain_FIELD_MAXX))
Pos_Ball_Y = Format$(object.y * (frmMain_FIELD_MAXY))

'find Angle Robot

```

```

del_y0 = Pos_Ball_Y - Pos_Robot_Y0
del_x0 = Pos_Ball_X - Pos_Robot_X0
del_y1 = Pos_Ball_Y - Pos_Robot_Y1
del_x1 = Pos_Ball_X - Pos_Robot_X1
Y0 = Abs(del_y0)
X0 = Abs(del_x0)
Y1 = Abs(del_y1)
X1 = Abs(del_x1)

```

```
'find Distance
```

```
dis0 = Sqr((Y0 * Y0) + (X0 * X0))
```

```
Command6.Caption = dis0 \ 1
```

```
dis1 = Sqr((Y1 * Y1) + (X1 * X1))
```

```
Command7.Caption = dis1 \ 1
```

```
If dis0 < dis1 Then
```

```
del_x = del_x0
```

```
del_y = del_y0
```

```
If del_x = 0 Then
```

```
If del_y > 0 Then
```

```
Alfa = 90
```

```
Else
```

```
Alfa = 270
```

```
End If
```

```
Else
```

```
Beta = (Atn(del_y / del_x)) * 180 / PI \ 1
```

```
Command1.Caption = Beta
```

```
' lblBall.Caption = lblBall.Caption + "@" & Beta
```

```
If del_x < 0. Then
```

```
Alfa = Beta + 180
```

```

Else
  If del_y > 0 Then
    Alfa = Beta
    Command1.Caption = Zeta
  Else
    Alfa = Beta + 360
  End If
End If

End If

End If

Zeta = Alfa - Degree_Robot0
Command1.Caption = Zeta
Command2.Caption = Alfa
Command3.Caption = Degree_Robot0
End If

If dis0 > dis1 Then
  del_x = del_x1
  del_y = del_y1

  If del_x = 0 Then
    If del_y > 0 Then
      Alfa = 90
    Else
      Alfa = 270
    End If
  Else
    Beta = (Atn(del_y / del_x)) * 180 / PI \ 1
    Command1.Caption = Beta
    lblBall.Caption = lblBall.Caption + "@" & Beta
    If del_x < 0 Then
      Alfa = Beta + 180
    End If
  End If
End If

```

```

Else
  If del_y > 0 Then
    Alfa = Beta
    Command1.Caption = Zeta
  Else
    Alfa = Beta + 360
  End If
End If

End If

Zeta = Alfa - Degree_Robot1
Command1.Caption = Zeta
Command2.Caption = Alfa
Command3.Caption = Degree_Robot1
End If

If Zeta > 180 Then
  Zeta = Zeta - 360
End If
If Zeta < -180 Then
  Zeta = Zeta + 360
End If
Command4.Caption = Zeta

If (Degree_Robot0 > 180) Then
  Degree_Robot0 = Degree_Robot0 - 360
End If

If (Degree_Robot0 <= 0 And Degree_Robot0 > -30) Then
  Sent_Code = 6
Else
  Sent_Code = 5

```

End If

' code robot 4

If dis0 < dis1 Then

Sent_Code = 5

If dis0 > 15 Then

'Sent_Code = 5

If (Zeta > -30 And Zeta < 30) Then

Sent_Code = 8

End If

If (Zeta > 30 And Zeta < 90) Then

Sent_Code = 7

End If

If (Zeta > 90 And Zeta < 150) Then

Sent_Code = 1

End If

If (Zeta > 150 And Zeta < 180) Or (Zeta > -180 And Zeta < -150) Then

Sent_Code = 2

End If

If (Zeta > -150 And Zeta < -90) Then

Sent_Code = 3

End If

If (Zeta > -90 And Zeta < -30) Then

Sent_Code = 9

End If

End If

If dis0 <= 15 Then

Sent_Code = 5

End If

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

End If

'code Robot 5

If dis0 > dis1 Then

Sent_Code = "G"

If dis1 > 15 Then

If (Zeta > -30 And Zeta < 30) Then

Sent_Code = "T"

End If

If (Zeta > 30 And Zeta < 90) Then

Sent_Code = "R"

End If

If (Zeta > 90 And Zeta < 150) Then

Sent_Code = "V"

End If

If (Zeta > 150 And Zeta < 180) Or (Zeta > -180 And Zeta < -150) Then

Sent_Code = "B"

End If

If (Zeta > -150 And Zeta < -90) Then

Sent_Code = "N"

End If

If (Zeta > -90 And Zeta < -30) Then

Sent_Code = "Y"

End If

End If

If dis1 <= 15 Then

Sent_Code = "G"

End If

End If

'Sent Code to Serial

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

If MSComm1.PortOpen Then
    MSComm1.Output = Sent_Code '
End If
End Sub

```

```

Private Sub mnuAbout_Click()
    frmAbout.Show vbModal
End Sub

```

```

Private Sub mnuExit_Click()
    Unload Me
End Sub

```

```

Private Sub mnuLoad_Click()
    LoadSettings
End Sub

```

```

Private Sub mnuSave_Click()
    SaveSettings
End Sub

```

```

Private Sub sockMain_DataArrival(ByVal bytesTotal As Long)

```

```

    If (bytesTotal <= 1) Then Exit Sub

```

```

    Dim Data As String

```

```

    Dim DataArr() As String

```

```

    Dim i As Integer, Packet As Long

```

```

    Dim FromIP As String

```

```

    Dim tmpflag As Boolean

```

```

    Dim o_ject As OBJECTDATA

```

```

    Dim R_bot As ROBOTDATA

```

```

    sockMain.GetData Data, vbString

```

```

DataArr = Split(Data, ":")
Select Case DataArr(0)
Case "HELLO": ' Reply from client
    FromIP = DataArr(1)
    tmpflag = True ' Assume add
    ' Check existing, if not found then add
    For i = 0 To (lstClient.ListCount - 1)
        If (lstClient.List(i) = FromIP) Then
            tmpflag = False ' Don't add
        Exit For
    End If
Next i
If (tmpflag) Then
    If (lstClient.ListCount <= frmMain_MAXCLIENT) Then ' Limit number of client
        lstClient.AddItem DataArr(1) ' Add client to list
        ReDim DataReceived(0 To (lstClient.ListCount - 1)) As VISIONDATA
    End If
End If
Case "DATA": ' Receive data from a client, update them
    FromIP = DataArr(1)
    Packet = 2
    If (ReceivedCount < lstClient.ListCount) Then
        If (DataArr(2) = "T") Then
            DataReceivcd(ReceivedCount).Ball.valid = True
            DataReceivcd(ReceivedCount).Ball.x = Val(DataArr(3))
            DataReceived(ReceivedCount).Ball.y = Val(DataArr(4))
            Packet = Packet + 3
        Else
            'DataReceived(ReceivedCount).Ball.valid = False
            Packet = Packet + 1
        End If

```

```

For i = 0 To 4
  If (DataArr(Packet) = "T") Then
    DataReceived(ReceivedCount).Opponent(i).valid = True
    DataReceived(ReceivedCount).Opponent(i).x = Val(DataArr(Packet + 1))
    DataReceived(ReceivedCount).Opponent(i).y = Val(DataArr(Packet + 2))
    Packet = Packet + 3
  Else
    'DataReceived(ReceivedCount).Opponent(i).valid = False
    Packet = Packet + 1
  End If
Next i
For i = 0 To 4
  If (DataArr(Packet) = "T") Then
    DataReceived(ReceivedCount).Robot(i).valid = True
    DataReceived(ReceivedCount).Robot(i).x = Val(DataArr(Packet + 1))
    DataReceived(ReceivedCount).Robot(i).y = Val(DataArr(Packet + 2))
    DataReceived(ReceivedCount).Robot(i).dir = Val(DataArr(Packet + 3))
    Packet = Packet + 4
  Else
    'DataReceived(ReceivedCount).Robot(i).valid = False
    Packet = Packet + 1
  End If
Next i
ReceivedCount = ReceivedCount + 1
End If

' Update Data (Average them)
If (ReceivedCount >= lstClient.ListCount) And (lstClient.ListCount > 0) Then
  Dim sumX As Double, sumY As Double
  Dim sumSine As Double, sumCosine As Double
  Dim Count As Integer

```

```

Dim addnew As Boolean
Dim j As Integer, k As Integer
Dim OpponentX(0 To 4) As New clsAvrNumber
Dim OpponentY(0 To 4) As New clsAvrNumber
Dim OppCount As Integer
Dim result As Long

```

```
' Ball
```

```
sumX = 0: sumY = 0
```

```
Count = 0
```

```
For i = 0 To (ReceivedCount - 1)
```

```
    If (DataReceived(i).Ball.valid) Then
```

```
        sumX = sumX + DataReceived(i).Ball.x
```

```
        sumY = sumY + DataReceived(i).Ball.y
```

```
        Count = Count + 1
```

```
    End If
```

```
Next i
```

```
If (Count > 0) Then
```

```
    frmMonitor.rsMain.SetBall sumX / Count / frmMain_FIELD_MAXX, sumY / Count /
frmMain_FIELD_MAXY, True
```

```
If (sendDataTAI = True) Then
```

```
    frmMonitor.rsMain.GetBall o_ject.x, o_ject.y, o_ject.valid
```

```
    'tmpstr = tmpstr + ":T:" + Format$(ob_ject.X * (frmMain_FIELD_MAXX), "0") + ":" +
Format$(ob_ject.Y * (frmMain_FIELD_MAXY), "0")
```

```
    result = sendData(Ball, 0, CLng(o_ject.x * (frmMain_FIELD_MAXX)), CLng(o_ject.y *
(frmMain_FIELD_MAXY)), 360)
```

```
End If
```

```

Else
    frmMonitor.rsMain.SetBall 0, 0, False
End If

' Opponents (Match them)
OppCount = 0
For i = 0 To 4
    OpponentX(i).CreateArray frmMain_MAXCLIENT
    OpponentY(i).CreateArray frmMain_MAXCLIENT
Next i
For i = 0 To (ReceivedCount - 1)
    For j = 0 To 4
        If (DataReceived(i).Opponent(j).valid) Then
            addnew = True
            For k = 0 To (OppCount - 1) ' Find the same point
                If (Distance(DataReceived(i).Opponent(j).x, DataReceived(i).Opponent(j).y,
OpponentX(k).GetAverage, OpponentY(k).GetAverage) <= frmMain_SAMEPOINT) Then
                    ' Add value to existing
                    OpponentX(k).AddValue DataReceived(i).Opponent(j).x
                    OpponentY(k).AddValue DataReceived(i).Opponent(j).y
                    addnew = False
                Exit For
            End If
        Next k
        If (addnew) Then
            If (OppCount < 5) Then
                ' Add new opponent
                OpponentX(OppCount).AddValue DataReceived(i).Opponent(j).x
                OpponentY(OppCount).AddValue DataReceived(i).Opponent(j).y
                OppCount = OppCount + 1
            End If
        End If
    End If
Next j
End For
Next i

```

```

    End If
  End If
  Next j
Next i
' Update opponent data
For j = 0 To (OppCount - 1)
  frmMonitor.rsMain.SetOpponent j, OpponentX(j).GetAverage / frmMain_FIELD_MAXX,
OpponentY(j).GetAverage / frmMain_FIELD_MAXY, True

  If (sendDataTAI = True) Then

    frmMonitor.rsMain.GetOpponent j, o_ject.x, o_ject.y, o_ject.valid
    'tmpstr = tmpstr + ":T:" + Format$(o_ject.X * (frmMain_FIELD_MAXX), "0") + ":" +
Format$(o_ject.Y * (frmMain_FIELD_MAXY), "0")

    result = sendData(DEVIL, j, CLng(o_ject.x * (frmMain_FIELD_MAXX)), CLng(o_ject.y *
(frmMain_FIELD_MAXY)), 360)

  End If

Next j

For j = OppCount To 4 ' Set other opponents to invalid
  frmMonitor.rsMain.SetOpponent j, 0, 0, False
Next j

' Our Team Robots R_bot
For j = 0 To 4
  sumX = 0: sumY = 0
  Count = 0
  sumSine = 0: sumCosine = 0

```

```
For i = 0 To (ReceivedCount - 1)
```

```
  If (DataReceived(i).Robot(j).valid) Then
```

```
    sumX = sumX + DataReceived(i).Robot(j).x
```

```
    sumY = sumY + DataReceived(i).Robot(j).y
```

```
    sumSine = sumSine + Sin(DataReceived(i).Robot(j).dir * PI / 180)
```

```
    sumCosine = sumCosine + Cos(DataReceived(i).Robot(j).dir * PI / 180)
```

```
    Count = Count + 1
```

```
  End If
```

```
Next i
```

```
If (Count > 0) Then
```

```
  Dim radian As Double
```

```
  radian = Atn((sumSine / Count) / (sumCosine / Count))
```

```
  If (sumCosine < 0) Then radian = radian + PI
```

```
  If (radian < 0) Then radian = radian + (2 * PI)
```

```
  frmMonitor.rsMain.SetRobot j, sumX / Count / frmMain_FIELD_MAXX, sumY / Count /  
frmMain_FIELD_MAXY, radian, True
```

```
  If (sendDataTAI = True) Then
```

```
    frmMonitor.rsMain.GetRobot j, R_bot.x, R_bot.y, R_bot.dir, R_bot.valid
```

```
    ' tmpstr = tmpstr + ":T:" + Format$(R_bot.X * (frmMain_FIELD_MAXX), "0") + ":" +  
Format$(R_bot.Y * (frmMain_FIELD_MAXY), "0") + ":" + Format$(R_bot.dir * 180 / PI, "0")
```

```
    result = sendData(ANGEL, 0, CLng(R_bot.x * (frmMain_FIELD_MAXX)),  
CLng(R_bot.y * (frmMain_FIELD_MAXY)), CLng(R_bot.dir * 180 / PI))
```

```
  End If
```

```
Else
```

```
  frmMonitor.rsMain.SetRobot j, 0, 0, 0, False
```

```
End If
```

```
Next j
```

```

    frmMonitor.rsMain.Redraw
    UpdateCount = UpdateCount + 1
    ReceivedCount = 0
End If
End Select
End Sub
Private Sub tmrFind_Timer()
    ' Find client
    cmdFind_Click
    'findai_Click
End Sub
Private Sub tmrRequest_Timer()
    ' Request data from clients
    ReceivedCount = 0
    sockMain.RemoteHost = frmMain_BROADCAST
    sockMain.sendData "REQDAT"

    'Winsock1.RemoteHost = frmMain_BROADCAST
    'Winsock1.SendData "OK_START"
End Sub
Private Sub tmrReset_Timer()
    'Reset data
    Dim i As Long
    DataReceived(ReceivedCount).Ball.valid = False
    For i = 0 To 4
        DataReceived(ReceivedCount).Opponent(i).valid = False
        DataReceived(ReceivedCount).Robot(i).valid = False
    Next i
End Sub

```

```

Private Sub tmrRuntime_Timer()
    UpdateData ' Update form
End Sub

Private Function Distance(X1 As Double, Y1 As Double, x2 As Double, y2 As Double) As Double
Return distance between 2 points
    Dim dx As Double, dy As Double
    dx = x2 - X1
    dy = y2 - Y1
    Distance = Sqr((dx * dx) + (dy * dy))
End Function

Private Sub vsbFindClient_Change()
    txtFindClient.Text = Format$(vsbFindClient.Max + vsbFindClient.Min - vsbFindClient.Value, "0")
    frmMain_FINDCLIENT = Val(txtFindClient.Text) * 1000
    tmrFind.Interval = frmMain_FINDCLIENT
End Sub

Private Sub vsbRate_Change()
    txtRate.Text = Format$(vsbRate.Max + vsbRate.Min - vsbRate.Value, "0")
    frmMain_REQUEST = 1000 / Val(txtRate.Text)
    tmrRequest.Interval = frmMain_REQUEST
End Sub

```

ภาคผนวก ข
ซอร์ซโค้ด(C Source Code)

Robot 1

This program was produced by the
CodeWizardAVR V1.24.7e Evaluation
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
<http://www.hpinfotech.com>
e-mail:office@hpinfotech.com

Project :

Version :

Date : 17/2/2007

Author : Freeware, for evaluation and non-commercial use only

Company :

Comments:

Chip type : ATmega128

Program type : Application

Clock frequency : 16.000650 MHz

Memory model : Small

External SRAM size : 0

Data Stack size : 1024

*****/

#include <mega128.h>

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

// Standard Input/Output functions
#include <stdio.h>

// Declare your global variables here

    unsigned char sent;

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;
```

```
// Port E initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x00;
```

```
// Port F initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTF=0x00;
DDRF=0x00;
```

```
// Port G initialization
// Func4=In Func3=In Func2=In Func1=In Func0=In
// State4=T State3=T State2=T State1=T State0=T
PORTG=0x00;
DDRG=0x00;
```

```
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
```

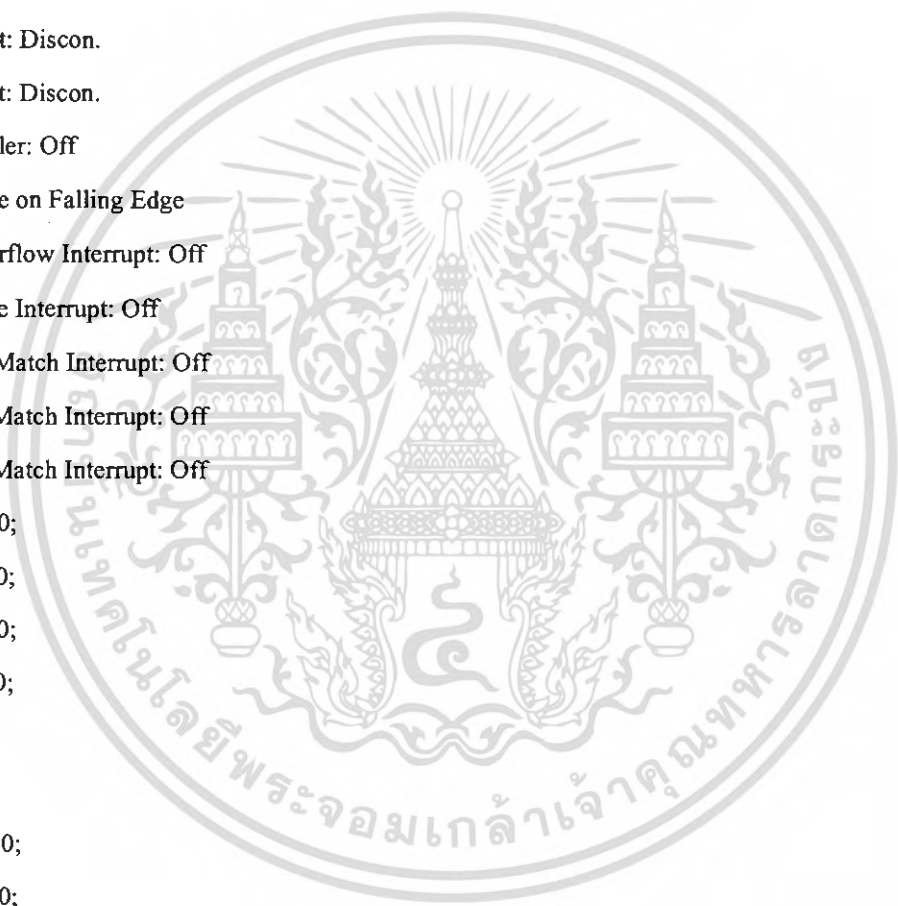
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

```



```
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
```

```
// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
```

```
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;
```

```
// External Interrupt(s) initialization
```

```
// INT0: Off
```

```
// INT1: Off
```

```
// INT2: Off
```

```
// INT3: Off
```

```
// INT4: Off
```

```
// INT5: Off
```

```
// INT6: Off
```

```
// INT7: Off
```

```
EICRA=0x00;
```

```
EICRB=0x00;
```

```
EIMSK=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
```

```
TIMSK=0x00;
```

```
ETIMSK=0x00;
```

```
// USART0 initialization
```

```
// Communication Parameters: 8 Data, 1 Stop, No Parity
```

```
// USART0 Receiver: On
```

```
// USART0 Transmitter: On
```

```
// USART0 Mode: Asynchronous
```

```
// USART0 Baud rate: 9600
```

```
UCSR0A=0x00;
```

```
UCSR0B=0x18;
```

```
UCSR0C=0x06;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
UBRR0H=0x00;
```

```
UBRR0L=0x67;
```

```
// Analog Comparator initialization
```

```
// Analog Comparator: Off
```

```
// Analog Comparator Input Capture by Timer/Counter 1: Off
```

```
ACSR=0x80;
```

```
SFIOR=0x00;
```

```
while (1)
```

```
{
```

```
    sent=getchar();
```

```
    switch (sent)
```

```
    {
```

```
        case '2':
```

```
            PORTA=0x09;
```

```
            putchar(sent);
```

```
        break;
```

```
        case '8':
```

```
            PORTA=0x06;
```

```
            putchar(sent);
```

```
        break;
```

```
        case '4':
```

```
            PORTA=0x10;
```

```
            putchar(sent);
```

```
        break;
```

```
        case '6':
```

```
            PORTA=0x20;
```

```
            putchar(sent);
```



```

break;
case '7' :
    PORTA=0x24;
    putchar(sent);
break;
case '3' :
    PORTA=0x18;
    putchar(sent);
break;
case '9' :
    PORTA=0x12;
    putchar(sent);
break;
case '1' :
    PORTA=0x21;
    putchar(sent);
break;
case 'A' :
    PORTA=0x15;
    putchar(sent);
break;
case 'D' :
    PORTA=0x2A;
    putchar(sent);
break;
case '5' :
    PORTA=0x00;
    putchar(sent);
break;
default :
    PORTA=0x00;

```

```
break;
```

```
}
```

```
// Place your code here
```

```
};
```

```
}
```



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Robot 2

This program was produced by the
CodeWizardAVR V1.24.7e Evaluation
Automatic Program Generator

© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.

<http://www.hpinfotech.com>

e-mail:office@hpinfotech.com

Project :

Version :

Date : 17/2/2007

Author : Freeware, for evaluation and non-commercial use only

Company :

Comments:

Chip type : ATmega128

Program type : Application

Clock frequency : 16.000650 MHz

Memory model : Small

External SRAM size : 0

Data Stack size : 1024

*****/

```
#include <mega128.h>
```

```
// Standard Input/Output functions
```

```
#include <stdio.h>
```

```
// Declare your global variables here
```

```
    unsigned char sent;
```

```
void main(void)
```

```
{
```

```
// Declare your local variables here
```

```
// Input/Output Ports initialization
```

```
// Port A initialization
```

```
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
```

```
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
```

```
PORTA=0x00;
```

```
DDRA=0xFF;
```

```
// Port B initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
```

```
PORTB=0x00;
```

```
DDRB=0x00;
```

```
// Port C initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
```

```
PORTC=0x00;
```

```
DDRC=0x00;
```

```
// Port D initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
```

```

PORTD=0x00;
DDRD=0x00;

// Port E initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x00;

// Port F initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTF=0x00;
DDRF=0x00;

// Port G initialization
// Func4=In Func3=In Func2=In Func1=In Func0=In
// State4=T State3=T State2=T State1=T State0=T
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

```

```

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

```

```

// Timcr/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

```

```
// External Interrupt(s) initialization
```

```
// INT0: Off
```

```
// INT1: Off
```

```
// INT2: Off
```

```
// INT3: Off
```

```
// INT4: Off
```

```
// INT5: Off
```

```
// INT6: Off
```

```
// INT7: Off
```

```
EICRA=0x00;
```

```
EICRB=0x00;
```

```
EIMSK=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
```

```
TIMSK=0x00;
```

```
ETIMSK=0x00;
```

```
// USART0 initialization
```

```
// Communication Parameters: 8 Data, 1 Stop, No Parity
```

```
// USART0 Receiver: On
```

```
// USART0 Transmitter: On
```

```
// USART0 Mode: Asynchronous
```

```
// USART0 Baud rate: 9600
```

```
UCSR0A=0x00;
```

```
UCSR0B=0x18;
```

```
UCSR0C=0x06;
```

```
UBRR0H=0x00;
```

```
UBRR0L=0x67;
```

```
// Analog Comparator initialization
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
```

```
while (1)
```

```
{
```

```
    sent=getchar();
```

```
    switch (sent)
```

```
    {
```

```
        case 'B' :
```

```
            PORTA=0x09;
```

```
            putchar(sent);
```

```
        break;
```

```
        case 'T' :
```

```
            PORTA=0x06;
```

```
            putchar(sent);
```

```
        break;
```

```
        case 'F' :
```

```
            PORTA=0x10;
```

```
            putchar(sent);
```

```
        break;
```

```
        case 'H' :
```

```
            PORTA=0x20;
```

```
            putchar(sent);
```

```
        break;
```

```
        case 'R' :
```

```
            PORTA=0x24;
```

```
            putchar(sent);
```

```
    }
```

```

break;
case 'N' :
    PORTA=0x18;
    putchar(sent);
break;
case 'Y' :
    PORTA=0x12;
    putchar(sent);
break;
case 'V' :
    PORTA=0x21;
    putchar(sent);
break;
case 'G' :
    PORTA=0x00;
    putchar(sent);
break;
default :
    PORTA=0x00;
break;
}

```

// Place your code here

```
};
```

```
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก
Data Sheet L298N



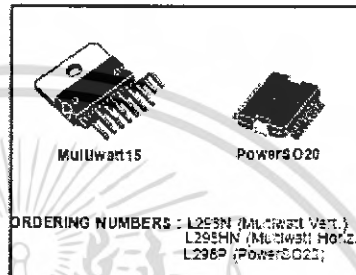
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL '0' INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

DESCRIPTION

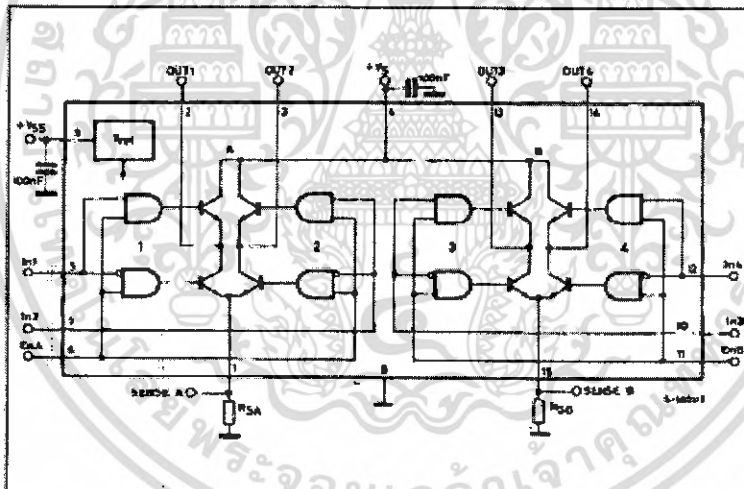
The L298 is an integrated monolithic circuit in a 16-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device, independently of the input signals. The emitters of the power transistors of each bridge are connected together and the corresponding external terminals can be used for the con-



ORDERING NUMBERS : L298N (Multiwatt 15)
L298HN (Multiwatt 16)
L298P (PowerSO20)

nection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

BLOCK DIAGRAM



JefL377 2003

1/13

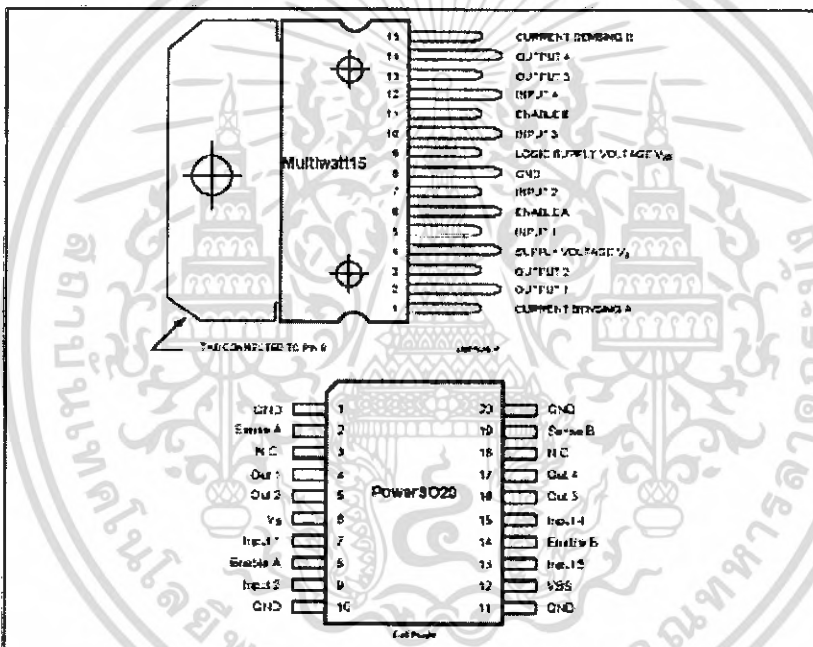
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่นิยมนำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

L298

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{DD}	Power Supply	50	V
V_{DD}	Logic Supply Voltage	7	V
V_{IN}, V_{EN}	Inputs and Enable Voltage	-0.3 to 7	V
I_O	Peak Output Current (each Channel) - Non Repetitive (1 = 100µs) - Repetitive (80% on - 20% off $t_{ON} = 10ms$) - DC Operation	3 2.5 2	A A A
V_{SEN1}	Sensing Voltage	-1 to 2.3	V
P_{tot}	Total Power Dissipation ($T_{amb} = 75^{\circ}C$)	25	W
T_{op}	Junction Operating Temperature	-25 to 130	$^{\circ}C$
T_{stg}, T_j	Storage and Junction Temperature	-40 to 150	$^{\circ}C$

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter	PowerSO20	Multiwatt15	Unit
$R_{th(j-c)}$	Thermal Resistance Junction-case	Max.	3	$^{\circ}C/W$
$R_{th(j-a)}$	Thermal Resistance Junction-ambient	Max.	25	$^{\circ}C/W$

(*) Mounted on aluminum substrate



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่าจะกรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

L298

PIN FUNCTIONS (refer to the block diagram)

MW: 15	PowerSO	Name	Function
1:15	2:19	Sense A; Sense E	Between this pin and ground, connect the sense resistor to control the current of the load.
2:3	4:5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 15.
4	6	V _S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5:7	7:9	Input 1; Input 2	TTL- Compatible Inputs of the Bridge A.
6:11	6:14	Enable A; Enable B	TTL- Compatible Enable input: the L state disables the bridge A (enable A); and/or the bridge B (enable B).
8	11:11:23	GND	Ground.
9	12	V _{SS}	Supply Voltage for the Logic Blocks. A 100nF capacitor must be connected between this pin and ground.
10:12	13:15	Input 3; Input 4	TTL- Compatible Inputs of the Bridge B.
13:14	16:17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
-	3:13	N.C.	Not Connected.

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V; T_J = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V _S	Supply Voltage (pin 4)	Operative Condition	V _{SS} -2.5		46	V
V _{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I _Q	Quiescent Supply Current (pin 4)	V _{SS} = H, I _L = 0 V _I = L V _I = H		13 53	20 70	mA mA
I _{CC}	Quiescent Current from V _{SS} (pin 9)	V _{SS} = H, I _L = 0 V _I = L V _I = H		24 7	36 12	mA mA
V _I	Input Low Voltage (pins 5, 7, 13, 12)		-0.3		1.5	V
V _{IH}	Input High Voltage (pins 5, 7, 13, 12)		2.3		V _{SS}	V
I _L	Low Voltage Input Current (pins 5, 7, 13, 12)	V = L			-10	μA
I _H	High Voltage Input Current (pins 5, 7, 13, 12)	V _I = H ± V _{SS} - 0.6V		30	120	μA
V _{en} = L	Enable Low Voltage (pins 6, 11)		-0.3		1.5	V
V _{en} = H	Enable High Voltage (pins 6, 11)		2.3		V _{SS}	V
I _{en} = L	Low Voltage Enable Current (pins 6, 11)	V _{en} = L			-10	μA
I _{en} = H	High Voltage Enable Current (pins 6, 11)	V _{en} = H ± V _{SS} - 0.6V		30	120	μA
V _{CCsat(H)}	Source Saturation Voltage	I _L = 1A I _L = 2A	0.95	1.35 2	1.7 2.7	V V
V _{CCsat(L)}	Sink Saturation Voltage	I _L = 1A (5) I _L = 2A (5)	0.65	1.2 1.7	1.5 2.3	V V
V _{CCtot}	Total Drop	I _L = 1A (5) I _L = 2A (5)	1.60		3.2 4.5	V V
V _{SENSE}	Sensing Voltage (pins 1, 15)		-1 (1)		2	V



3/13

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

L298

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
T_1 (ns)	Source Current Turn-on Delay	0.5V _{in} to 0.9V _L (2), (4)		1.5		μs
T_2 (ns)	Source Current Fall Time	0.9V _L to 0.1V _L (2), (4)		0.2		μs
T_3 (ns)	Source Current Turn-on Delay	0.5V _{in} to 0.1V _L (2), (4)		2		μs
T_4 (ns)	Source Current Rise Time	0.1V _L to 0.9V _L (2), (4)		0.7		μs
T_5 (ns)	Sink Current Turn-on Delay	0.5V _{in} to 0.9V _L (3), (4)		0.7		μs
T_6 (ns)	Sink Current Fall Time	0.9V _L to 0.1V _L (3), (4)		0.25		μs
T_7 (ns)	Sink Current Turn-on Delay	0.5V _{in} to 0.9V _L (3), (4)		1.6		μs
T_8 (ns)	Sink Current Rise Time	0.1V _L to 0.9V _L (3), (4)		0.2		μs
f_c (kHz)	Commutation Frequency	$I_L = 2A$		25	40	KHz
T_1 (V _{sat})	Source Current Turn-on Delay	0.5V _{in} to 0.9V _L (2), (4)		3		μs
T_2 (V _{sat})	Source Current Fall Time	0.9V _L to 0.1V _L (2), (4)		1		μs
T_3 (V _{sat})	Source Current Turn-on Delay	0.5V _{in} to 0.1V _L (2), (4)		0.3		μs
T_4 (V _{sat})	Source Current Rise Time	0.1V _L to 0.9V _L (2), (4)		0.2		μs
T_5 (V _{sat})	Sink Current Turn-on Delay	0.5V _{in} to 0.9V _L (3), (4)		2.2		μs
T_6 (V _{sat})	Sink Current Fall Time	0.9V _L to 0.1V _L (3), (4)		0.35		μs
T_7 (V _{sat})	Sink Current Turn-on Delay	0.5V _{in} to 0.9V _L (3), (4)		0.25		μs
T_8 (V _{sat})	Sink Current Rise Time	0.1V _L to 0.9V _L (3), (4)		0.1		μs

- 1) Switching voltage can be -1V for $I_L \leq 50$ mA; in steady state $V_{in, min} = -0.5V$.
- 2) See Fig. 2.
- 3) See Fig. 4.
- 4) The load must be a pure resistor.

Figure 1: Typical Saturation Voltage vs. Output Current.

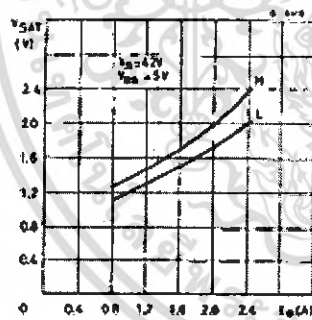
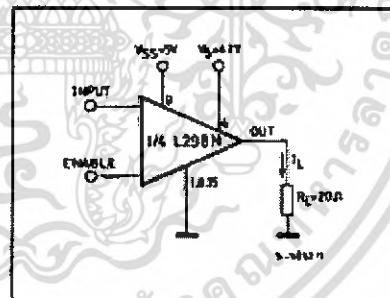


Figure 2: Switching Times Test Circuit.



Note: For INPUT switching, set EN = H
For ENABLE switching, set IN = H

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

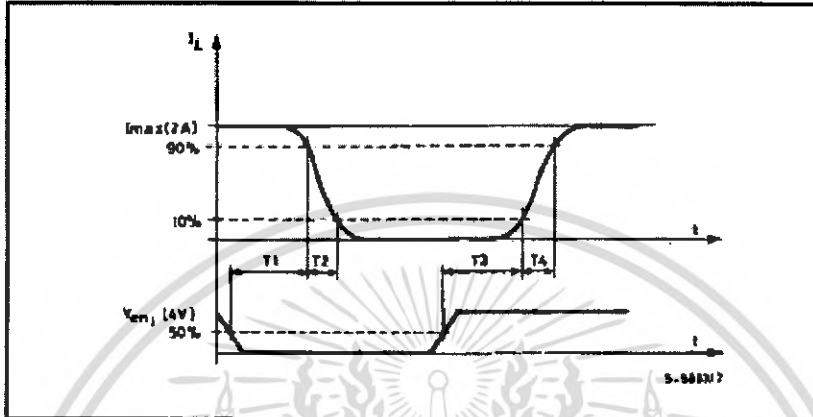
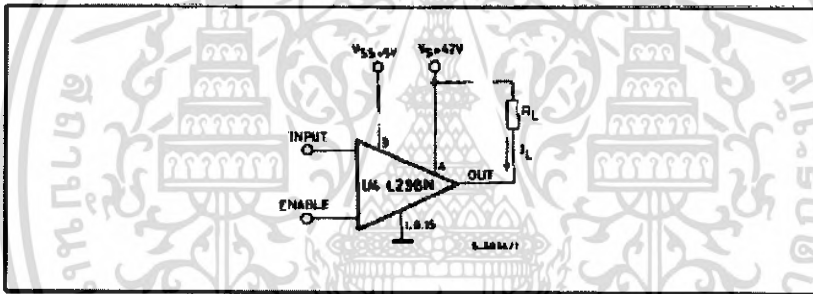


Figure 4 : Switching Times Test Circuits.



Note: For INPUT switching set EN = 1
For ENABLE switching set IN = 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้