ระบบควบคุมกล้องอัตโนมัติ

โดย

นาย นเรศ เปาะทองคำ 41012011
นาย จำลักษณ์ วารีชล 41012047

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตรบัณฑิต
สาขาเทคโนโลยีอิเล็กทรอนิกส์ ภาควิชาเทคนิคอุตสาหกรรม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

Automatic Control System for Room Monitoring Camera

By

Mr. Naress	Pothongkam	41012011
Mr. Jumlux	Wareechol	41012047

Project Report Submitted in Partial Fulfillment of the Requirement

For the Bachelor's Degree

Department of Industrial Technology

Faculty of Engineer

King Mongkut's Institute of Technology Ladkrabang

1999

หัวข้อปริญญานิพนธ์	ระบบควบคุมกล้องอัตโนมัติ		
ชื่อนักศึกษา	นาย นเรศ	เปาะทองคำ	41012011
	นาย จำลักษณ์	วารีชล	41012047
อาจารย์ที่ปรึกษา	ผศ.ดร. ปิติเขต	สู้รักษา	
	อาจารย์ มยุรี	เลิศเวชกุล	
ภาควิชา	เทคนิคอุตสาหกรรม		
ปีการศึกษา	2542		

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตรบัณฑิต

.....หัวหน้าภาควิชา

( )

คณะกรรมการสอบปริญญานิพนธ์

.....อาจารย์ที่ปรึกษา

( )

.....อาจารย์ที่ปรึกษา

( )

.....กรรมการ

( )

.....กรรมการ

( )

.....กรรมการ

( )

Project Report	Automatic Control System for Room Monitoring Camera		
By	Mr. Naress	Pothongkam	41012011
	Mr. Jumlux	Wareechol	41012047
Department	Industrial Technology		
Project Report Advisor	Asst.Prof.Dr. Pitikhate	Sooraksa	
	Ms. Mayuree	Lertwatechakul	

Accepted by the Faculty of Engineer, King Mongkut's Institute of Technology Ladkrabang in Partial Fulfillment of the Requirement for the Bachelor's Degree

.....Chairman

( )

.....Chairman

( )

Project Report Committee

.....Member

( )

.....Member

( )

.....Member

( )

หัวข้อปริญญานิพนธ์	ระบบควบคุมกล้องอัตโนมัติ		
ชื่อนักศึกษา	นาย นเรศ	เปาะทองคำ	41012011
	นาย จำลักษณ์	วารีชล	41012047
อาจารย์ที่ปรึกษา	ผศ.ดร. ปิติเขต	สู้รักษา	
	อาจารย์ มยุรี	เลิศเวชกุล	
ภาควิชา	เทคนิคอุตสาหกรรม		
ปีการศึกษา	2542		

บทคัดย่อ

จุดมุ่งหมายของปริญญานิพนธ์นี้เพื่อออกแบบและสร้างระบบควบคุมกล้องอัตโนมัติ โดยอาศัยการทำงานของไมโครคอนโทรลเลอร์ BASIC STAMP2 และโปรแกรม VISUAL BASIC 6.0 ซึ่งใช้สร้างส่วนสื่อสารระหว่างผู้ใช้กับคอมพิวเตอร์รวมถึงการเชื่อมต่อกับไมโครคอนโทรลเลอร์

ผลที่ได้จากปริญญานิพนธ์นี้สามารถนำไปใช้เพื่อพัฒนาระบบรักษาความปลอดภัยที่สามารถผลิตได้เองในประเทศ

Project Report	Automatic Control System for Room Monitoring Camera
By	Mr. Naress Pothongkam 41012011
	Mr. Jumlux Wareechol 41012047
Department	Industrial Technology
Project Report Advisor	Asst.Prof.Dr. Pitikhate Sooraksa
	Ms. Mayuree Lertwatechakul
Academic year	1999

Abstract

The objective of this thesis is to design and implement an automatic control system for room monitoring camera. A "Basic Stamp2" microcontroller is used as a CPU and the program is written in VISUAL BASIC 6.0 in order to build up communication among the users , computers and the microcontroller.

The results from this thesis can be applied to the room-monitoring security system produced domestically in Thailand.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้มิอาจลุลวงไปได้หากขาดความช่วยเหลือจากบุคคลต่างๆในหลายฝ่าย ดังนั้นคณะผู้จัดทำจึงใคร่ขอขอบคุณบุคคลต่างๆดังต่อไปนี้

ขอขอบคุณ ผศ.ดร ปิติเขต สู้รักษา ที่กรุณาให้คำชี้แนะและเป็นที่ปรึกษาในโครงงานนี้ด้วยดีตลอดมาพร้อมกันนี้ขอขอบคุณ อาจารย์ มยุรี เลิศเวชกุล อาจารย์ พิทักษ์ ธรรมวาริน และ คณาจารย์ภาควิชาเทคนิคอุตสาหกรรมที่ให้คำปรึกษาเกี่ยวกับข้อมูลต่างๆ

คณะผู้จัดทำ

สารบัญ

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและสาเหตุจูงใจของการทำโครงงาน

ในปัจจุบันได้มีระบบรักษาความปลอดภัยโดยใช้กล้องโทรทัศน์วงจรปิดหลายรูปแบบที่ใช้อยู่ตามห้างสรรพสินค้าร้านค้าหรืออาคารต่างๆซึ่งมีอยู่จำนวนมากมายหลายชนิด ซึ่งแต่ละชนิดก็ขึ้นอยู่กับความเหมาะสมในการเลือกที่จะนำไปใช้งาน และโดยทั่วไปที่ใช้อยู่จะเป็นเพียงจับภาพคนที่เข้ามาในบริเวณที่ติดกล้องอยู่เท่านั้น แต่ไม่สามารถที่จะเตือนหรือป้องกันไม่ให้ผู้ที่เข้ามาโดยมีจุดประสงค์เชิงโจรกรรมและอาชญากรรมเลิกล้มจุดประสงค์นั้นหรือออกไปจากสถานที่ที่เราไม่ต้องการให้เข้ามาโดยเฉพาะอย่างยิ่งบริเวณที่รักษาสมบัติล้ำค่าและสำคัญของชาติ เช่น พิพิธภัณฑ์สถาน โรงงานกษาปณ์และบริเวณคลังสรรพวุฒ

ดังนั้นในโครงงานนี้จึงมีการสร้างระบบรักษาความปลอดภัยที่มีความสามารถในการเตือนให้ผู้ที่มีจุดประสงค์ไม่ดี ที่เข้ามาในบริเวณที่ติดกล้องเลิกล้มความตั้งใจนั้น และสามารถที่จะจับภาพคนที่เข้าม่าในห้องได้อัตโนมัติด้วย อีกทั้งยังมีราคาที่ประหยัดกว่าระบบรักษาความปลอดภัยในปัจจุบันและสามารถใช้งานได้ง่ายกว่ารวมถึงเพื่อรองรับเทคโนโลยีการโจรกรรมที่ทันสมัยและก้าวหน้าในปัจจุบัน

1.2 วัตถุประสงค์ของการทำโครงการมีดังนี้

1.2.1 เพื่อศึกษาถึงการใช้งาน Microcontroller Basic Stamp ที่จะใช้ควบคุมฐานกล้อง

1.2.2 เพื่อศึกษาถึงการสร้าง Software ควบคุมการทำงานโดยใช้ Pbasic และ Visual Basic 6.0

1.2.3 เพื่อให้ผู้ใช้สามารถควบคุมการทำงานของระบบรักษาความปลอดภัยได้อย่างง่าย

1.2.4 เพื่อศึกษาการออกแบบระบบรักษาความปลอดภัยที่ประหยัดและสะดวกในการใช้งาน

1.3 ขอบเขตโครงการมีดังนี้

1.3.1 สามารถควบคุมการทำงานได้ 2 Modes คือ

1.3.1.1 Manual Mode คือการควบคุมตำแหน่งของกล้องโดย User ผ่าน Computer ซึ่งควบคุมได้ 4 ทิศทาง

1.3.1.2 Automatic Mode คือการหาตำแหน่งของผู้บุกรุกโดยใช้ระบบ Infrared Tracking

1.3.2 สามารถที่จะชี้ตำแหน่งผู้บุกรุกได้

1.3.3 สามารถแบ่งแยกผู้ใช้ออกเป็น 2 กลุ่ม

1.3.3.1 กลุ่มแรกนี้สามารถควบคุมตำแหน่งกล้องได้เพียงอย่างเดียว

1.3.3.2 กลุ่มที่สองนี้สามารถควบคุมตำแหน่งกล้องและใช้ Option ต่างๆที่มีอยู่ของระบบได้

1.4 ขั้นตอนการดำเนินงานมีดังนี้

1.4.1 ศึกษาการใช้งาน Microcontroller Basic Stamp2

1.4.2 ศึกษาการใช้ Program Pbasic เพื่อใช้ควบคุม Microcontroller

1.4.3 ศึกษาถึงการเขียน Program Visual Basic 6.0 เพื่อใช้สร้าง Interface ใช้งานและควบคุมการทำงานของระบบรักษาความปลอดภัย

1.4.4 ออกแบบโครงสร้างของ Hardware

1.4.5 ทดสอบวงจรโดยใช้ Photo Broad

1.4.6 สร้างวงจรโดยสมบูรณ์

1.4.7 ทดสอบการทำงานทั้งหมดของระบบและทดสอบการควบคุมระบบทั้งหมด

1.4.8 วิเคราะห์ถึงข้อบกพร่องของโครงงานและตรวจสอบความถูกต้องในการใช้งาน

1.4.9 สรุปผลและเขียนรายงานจัดทำโครงงานพร้อมทั้งนำเสนอผลงาน

บทที่ 2

ทฤษฎีหลักการและความรู้เบื้องต้นเกี่ยวกับ Microcontroller Basic Stamp

2.1 แนะนำเบสิกแสตมป์

ย้อนหลังไปราวปี 1996 วงการไมโครคอนโทรลเลอร์เมืองไทยมีโอกาสต้อนรับบอร์ดไมโครคอนโทรลเลอร์สายพันธุ์ใหม่ที่ชื่อว่า เบสิกแสตมป์ 1 (BASIC Stamp I) อันเป็นผลงานการพัฒนาของ Parallax Inc. จากสหรัฐอเมริกา ด้วยแนวคิดที่ว่า ผู้ใช้งานไม่จำเป็นต้องมีความรู้เกี่ยวกับภาษาแอสแซมบลีของไมโครคอนโทรลเลอร์เบอร์ใด ๆ เพียงสามารถเขียนโปรแกรมด้วยภาษาเบสิก อันเป็นภาษาที่มีความซับซ้อนน้อยที่สุดก็สามารถเขียนโปรแกรมเพื่อควบคุมและใช้งานไมโครคอนโทรลเลอร์ได้

ในขณะเดียวกันต้องไม่ใช้เครื่องมือในการพัฒนาที่มีความซับซ้อน กล่าวคือ เพียงต่อสายจากพอร์ตของคอมพิวเตอร์ ส่งข้อมูลผ่านโปรแกรมสื่อสารลงสู่หน่วยความจำบนตัวบอร์ด ปลดสายออก ทำการรีเซตระบบ เพียงเท่านี้บอร์ดไมโครคอนโทรลเลอร์นั้นก็สามารถทำงานได้แล้ว

แนวคิดนี้ได้รับการต้อนรับเป็นอย่างดีจากนักทดลองไมโครคอนโทรลเลอร์ โดยเฉพาะอย่างยิ่งนักพัฒนาระบบควบคุมของหุ่นยนต์ เนื่องจากขนาดของบอร์ดเล็กมาก กินไฟฟ้าต่ำเขียนโปรแกรมง่าย ไม่ต้องใช้เครื่องมือพัฒนาราคาแพง หากแต่ในเบสิกแสตมป์ 1 มีจำนวนของขาอินพุตเอาต์พุต และขนาดของหน่วยความจำน้อย ไม่เพียงพอต่อการใช้งาน

นั่นจึงเป็นที่มาของการพัฒนาเบสิกแสตมป์ในรุ่นต่อมาคือ เบสิกแสตมป์ 2 (BASIC Stamp II : BS2 – IC) ซึ่งมีจำนวนขาพอร์ตอินพุตเอาต์พุตมากขึ้นคือมีมากถึง 16 ขา และขาพอร์ตสำหรับสื่อสารข้อมูลอนุกรมอีก 2 ขา ในขณะที่ขนาดของหน่วยความจำโปรแกรมและข้อมูลก็เพิ่มสูงขึ้นอีกหลายเท่าตัว และที่สำคัญคือ ความเร็วในการทำงานสูงขึ้นถึง 4,000 คำสั่งภาษาเบสิกต่อวินาที

2.2 เบสิกแสตมป์คืออะไร

สำหรับนักทดลองไมโครคอนโทรลเลอร์และผู้สนใจที่ยังไม่รู้จักเบสิกแสตมป์ จึงขออธิบายความรู้เบื้องต้นเพื่อเป็นการแนะนำบอร์ดไมโครคอนโทรลเลอร์ตระกูลนี้ที่กำลังได้รับความนิยมสูงมากในวงการนักทดลองไมโครคอนโทรลเลอร์และหุ่นยนต์ขนาดเล็กทั่วโลก

เบสิกแสตมป์ (BASIC Stamp) คือแผงวงจรไมโครคอนโทรลเลอร์สำเร็จรูปที่บรรจุตัวแปลงภาษาเบสิกหรือเบสิกอินเตอร์พรีตเตอร์ (BASIC Interpreter) รวมไว้ด้วยกัน สามารถใช้การเขียนโปรแกรมด้วยภาษาเบสิกควบคุมการทำงานได้ ในการพัฒนาระบบด้วยเบสิกแสตมป์ไม่จำเป็นต้องใช้เครื่องโปรแกรม (programmer) หรือเครื่องเลียนแบบ (emulator) ไมโครคอนโทรลเลอร์และหน่วยความจำแต่อย่างใด เพียงต่อสายจากคอมพิวเตอร์เข้ากับเบสิกแสตมป์เท่านั้น ก็สามารถพัฒนา

โปรแกรมได้แล้วและเหตุผลที่มีคำว่าแสตมป์ต่อท้ายก็เพื่อต้องการให้ทราบว่าบอร์ดไมโครคอนโทรลเลอร์ตัวนี้มีขนาดเล็กเท่ากับแสตมป์หรือตราไปรษณียากรนั่นเอง

การเรียนรู้เพื่อสร้างระบบหรือชิ้นงานด้วยเบสิกแสตมป์จะใช้งบประมาณเริ่มต้นต่ำ เมื่อเทียบกับไมโครคอนโทรลเลอร์ตระกูลอื่น ๆ เนื่องจากเบสิกแสตมป์ไม่ต้องการเครื่องมือพิเศษอื่นใด ไม่ว่าจะเป็นเครื่องโปรแกรมไมโครคอนโทรลเลอร์ซึ่งมีราคาตั้งแต่ 3,000 บาทขึ้นไป จนถึงหลายหมื่นบาท หรือเครื่องโปรแกรมหน่วยความจำ ซึ่งก็มีราคาตั้งแต่ 2,000 บาทขึ้นไป หรือกระทั่งเครื่องเลียนแบบไมโครคอนโทรลเลอร์ที่เรียกว่า อีมูเลเตอร์ (emulator) ซึ่งก็มีราคาไม่ต่ำกว่า 3,000 บาทขึ้นไป รวมแล้วต้องใช้งบประมาณเป็นหลักหมื่นบาทสำหรับการพัฒนาไมโครคอนโทรลเลอร์เบอร์หนึ่ง ๆ แต่สำหรับเบสิกแสตมป์ใช้งบประมาณเพียง 2,000 บาท (ที่อัตราแลกเปลี่ยน 1 เหรียญดอลล่าสหรัฐ เท่ากับ 40 บาทไทยโดยประมาณ) ซอฟต์แวร์ที่ใช้งานร่วมกับเบสิกแสตมป์ ทางผู้ผลิตคือ Parallax Inc. ก็แจกฟรี สามารถดาวน์โหลดได้จากเวปไซต์ของ Parallax Inc. ที่ http://www.parallaxinc.com

ทางด้านการเขียนโปรแกรมด้วยชุดคำสั่งภาษาเบสิกที่เรียกว่า พีเบสิก (PBASIC) ซึ่งมีด้วยกัน 36 คำสั่ง ทำให้การเรียนรู้ง่ายและรวดเร็ว แต่ละคำสั่งสามารถนำไปใช้ได้ทันที ไม่ต้องเขียนโปรแกรมย่อยมากมาย ดังเช่นการเขียนด้วยภาษาแอสแซมบลี กอปรกับความเร็วในการกระทำคำสั่งของเบสิก โดยเฉพาะเบสิกแสตมป์ 2 (BS2 – IC) ซึ่งสูงถึง 4,000 คำสั่งภาษาเบสิกต่อวินาที ทำให้เบสิกแสตมป์ทำงานได้รวดเร็วทัดเทียมกับไมโครคอนโทรลเลอร์เบอร์อื่น ๆ อย่างสบาย

2.3 ตระกูลของเบสิกแสตมป์

เบสิกแสตมป์ผลิตโดย Parallax Inc. ในสหรัฐอเมริกานับถึงปัจจุบัน (ปี 1999) มีการผลิตเบสิกแสตมป์ออกมาแล้ว 4 รุ่น คือ

BS1 – REV.D หรือ เบสิกแสตมป์ 1 รีวิชั่นดี มีขาพอร์ตอินพุตเอาต์พุต 8 ขา หน่วยความจำ 256 ไบต์ บรรจุคำสั่งภาษา PBASIC – 1 ได้ 75 คำสั่ง ความเร็วในการกระทำคำสั่งภาษาเบสิก 2,000 คำสั่งต่อวินาที

BS1 – IC หรือ เบสิกแสตมป์ 1 รุ่นไอซี มีรูปร่างเป็นไอซีแบบแถวเดี่ยว มีจำนวนขาอินพุตเอาต์พุต, ขนาดหน่วยความจำ และความเร็วเท่ากับรุ่น REV.D

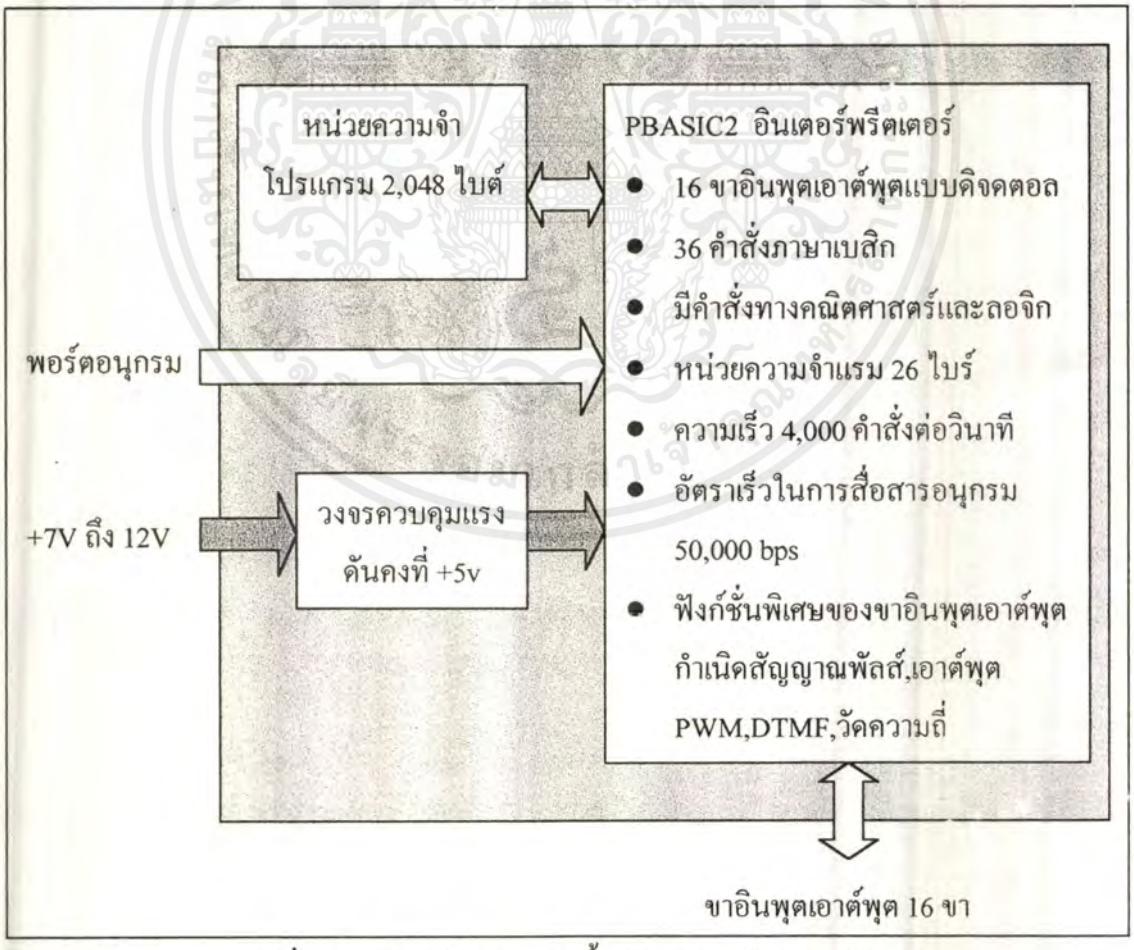
BS2 – IC หรือ เบสิกแสตมป์ 2 รุ่นไอซี มีรูปร่างและขนาดเท่ากับไอซีตัวถัง DIP ขนาด 4 ขา มีขาพอร์ตอินพุตเอาต์พุต 16 ขา ขาสำหรับเชื่อมต่อกับคอมพิวเตอร์ 3 ขา ขนาดหน่วยความจำ 2 กิโลไบต์ บรรจุคำสั่งภาษา PBASIC – 2 ได้ 500 คำสั่ง มีความเร็วในการประมวลผลคำสั่งภาษาเบสิก 4,000 คำสั่งต่อวินาที

BS3SX – IC หรือ เบสิกแสตมป์ 2 รุ่นเอสเอ็กซ์ เป็นรุ่นที่ใช้ไมโครคอนโทรลเลอร์ตระกูล SX ซึ่งมีความเร็วในการทำงานสูงกว่า BS2 – IC มีจำนวนขาพอร์ตอินพุตเอาต์พุตและขนาดของตัวถังเหมือนกับ BS2 – IC ขนาดของหน่วยความจำสูงถึง 16 กิโลไบต์ สามารถบรรจุคำสั่งภาษา PBASIC –2

ได้ 2,000 คำสั่ง มีความเร็วในการประมวลผลคำสั่งภาษาเบสิก 10,000 คำสั่งต่อวินาที เหมาะสำหรับงานที่ต้องการความเร็วและขนาดของหน่วยความจำสูง

2.4 วงจรภายในของเบสิกแสตมป์ 2

เนื่องจากข้อจำกัดด้านความเร็วและขนาดของหน่วยความจำโปรแกรมใน BS1 จึงได้มีการพัฒนาเบสิกแสตมป์ในรุ่นถัดมาเป็น เบสิกแสตมป์ 2 หรือ BS2 – IC ในรูปที่ 2 – 1 เป็นบล็อกไดอะแกรมแสดงคุณสมบัติหลัก ๆ ของ BS2 จะเห็นได้ว่าส่วนประกอบหลักของ BS2 จะมี 3 ส่วนคือ ตัวแปรภาษาPBASIC,หน่วยความจำโปรแกรมและวงจรเรกูเลเตอร์ BS2 มีจำนวนขาอินพุตเอาต์พุตมากถึง 16 ขา (P0 – P15) มีชุดคำสั่ง PBASIC – 2 36 คำสั่ง และสามารถประมวลผลทางคณิตศาสตร์และลอจิกได้ดีเพิ่มขึ้น มีหน่วยความจำแรมภายใน 26 ไบต์ มีความเร็วในการทำงานภาษา PBASIC – 2 ได้สูงถึง 4,000 คำสั่งต่อวินาที มีอัตราเร็วในการถ่ายทอดข้อมูลกับคอมพิวเตอร์ผ่านพอร์ตอนุกรมสูงถึง 50 กิโลบิตต่อวินาที และขาพอร์ตอินพุตเอาต์พุตสามารถใช้ในการกำเนิดสัญญาณพัลส์, สัญญาณ DTMF, สัญญาณ PWM และสามารถใช้ในการวัดความถี่ของสัญญาณไฟฟ้าโดยตรง



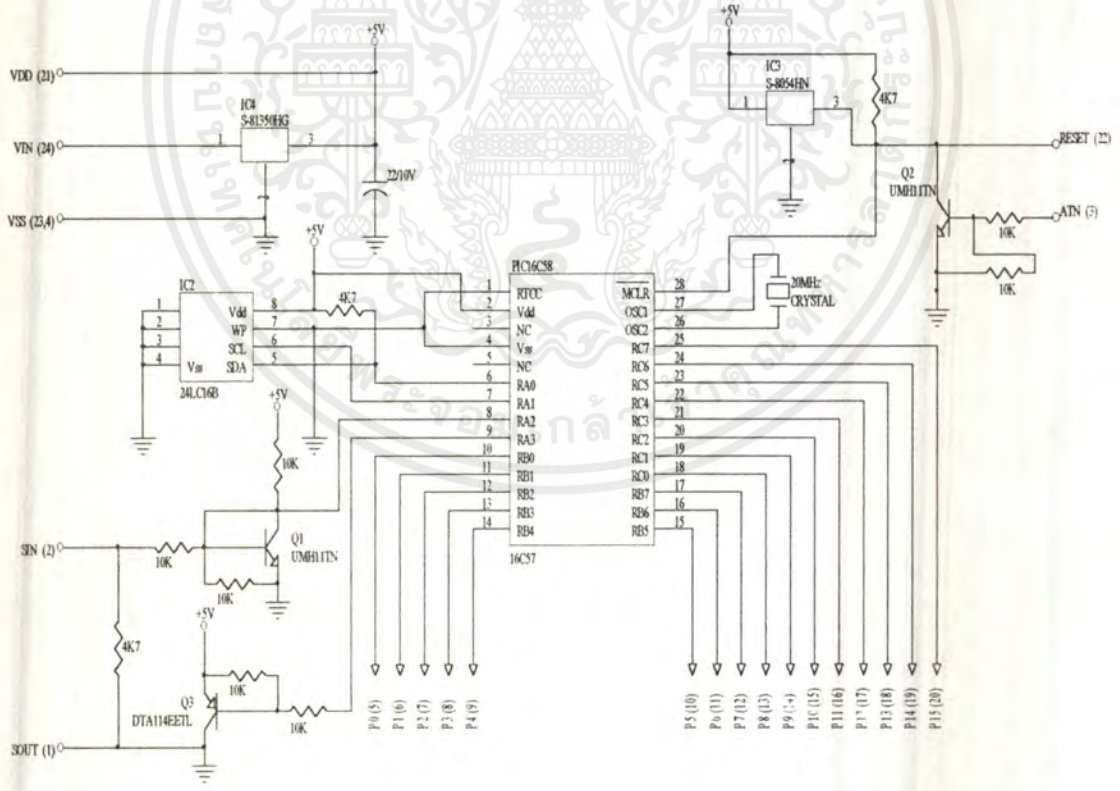
รูปที่ 2 – 1 บล็อกไดอะแกรมเบื้องต้นของเบสิกแสตมป์ 2

วงจรสมบูรณ์ของ BS2 แสดงในรูปที่ 2 – 2 แบ่งออกเป็น 4 ส่วนเช่นเดียวกับ BS1 เริ่มจากตัว

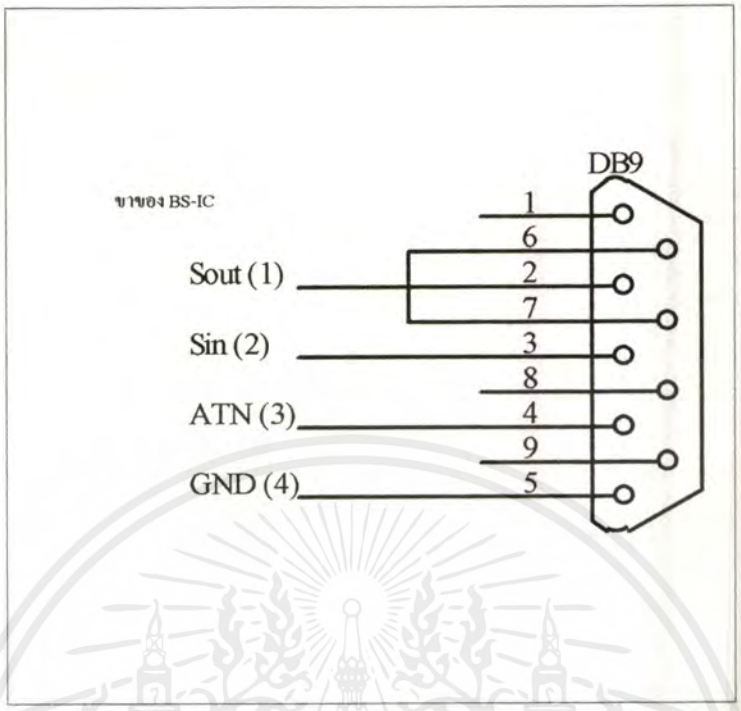
แปลภาษา PBASIC – 2 สำหรับ BS2 ใช้ไมโครคอนโทรลเลอร์ PIC16C57 ซึ่งมีจำนวนของพอร์ตอินพุตเอาต์พุตมากกว่า PIC16C56 ที่ใช้ใน BS1 ส่วนความสามารถในการจ่ายกระแส ยังคงเป็น 25 mA สำหรับกระแสซิงก์ และ 20mA สำหรับกระแสซอร์ซ และทุกขาพอร์ตสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต

ทางด้านหน่วยความจำโปรแกรม IC2 ใช้ไอซีหน่วยความจำอีอีพรอมอนุกรมเบอร์ 24LC16 มีความจุ 2 กิโลไบต์ (2 KB) สามารถบรรจุคำสั่งของ PBASIC – 2 ที่ใช้ในการรันโปรแกรมของ BS2 ได้ 500 คำสั่ง การเก็บรักษาข้อมูลทำได้ยาวนานและมีรอบของโปรแกรมเหมือนกับ BS1 คือเก็บได้นาน 10 ปี โปรแกรมใหม่ได้ 10 ล้านรอบ

ส่วนการโปรแกรมข้อมูล ยังคงใช้การโปรแกรมในลักษณะอนุกรม แต่จะติดต่อกับพอร์ตอนุกรมคอมพิวเตอร์แทน โดยต่อสายจากขา TxD, RxD, DTR และกราวด์ของพอร์ตอนุกรมเข้าสู่ขา Sin, Sout, ATN และกราวด์ของ BS2 นอกจากนั้นที่ขา 6 และ 7 ของพอร์ตอนุกรมต้องต่อถึงกันด้วย เพื่อให้สามารถโปรแกรมข้อมูลแบน BS2 ได้ตลอดเวลาอย่างอัตโนมัติ ดังแสดงในรูปที่ 2 – 3



รูปที่ 2 – 2 วงจรสมบูรณ์ของเบสิกแสตมป์ 2



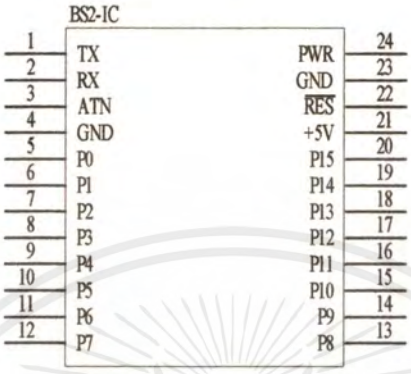
รูปที่ 2 – 3 การเชื่อมต่อเบสิกแสตมป์ 2 กับพอร์ตอนุกรมของคอมพิวเตอร์

สัญญาณจากพอร์ตอนุกรมของคอมพิวเตอร์โดยทั่วไป มีระดับแรงดันสูงกว่าระดับทีทีแอล กล่าวคือ มีแรงดันประมาณ ± 12V สำหรับคอมพิวเตอร์ตั้งโต๊ะ และ ± 6V ถึง ± 8V สำหรับคอมพิวเตอร์ให้เป็นระดับทีทีแอล (0 – 5V) เพื่อให้สามารถติดต่อกับ PIC16C57 ได้ขา SIN เป็นขาสำหรับข้อมูลจากคอมพิวเตอร์ เมื่อขาคอมพิวเตอร์ส่งข้อมูล "1" จะทำให้ SIN เป็นลบ ทรานซิสเตอร์ Q1 จะไม่ทำงาน ทำให้ที่ขา RA2 ของ PIC16C57 (หรือ IC1) มีแรงดัน +5V เกิดเป็นลอจิก "1" ทั้งนี้เนื่องจากในมาตรฐาน RS-232 แล้วลอจิก "1" คือระดับแรงดันตั้งแต่ –3 ถึง –12V ในขณะที่ลอจิก "0" คือระดับแรงดัน + 3 ถึง + 12 V เมื่อ BS2 ส่งข้อมูล "0" ทรานซิสเตอร์ Q3 ทำงาน ที่ขา SOUT จึงเกิดแรงดัน + 5 V ทำให้คอมพิวเตอร์อ่านข้อมูลได้เป็น "0"

เมื่อเป็นเช่นนี้ในการติดต่อระหว่าง BS2 กับคอมพิวเตอร์จะต้องสลับกันรับและส่งข้อมูลกล่าวคือ เมื่อคอมพิวเตอร์ส่งข้อมูลมา BS2 ต้องทำหน้าที่รับข้อมูลอย่างเดียว ไม่สามารถที่จะส่งข้อมูลกลับไปยังคอมพิวเตอร์ในเวลาเดียวกันได้

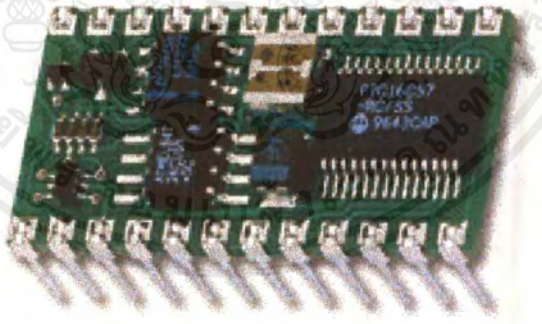
ขา ATN ซึ่งต่อเข้ากับขา DTR (Data Terminal Ready) ใช้ในการแฮนด์เชก หรือตรวจสอบความพร้อมในการรับส่งข้อมูลของพอร์ตอนุกรม จะมีลักษณะทำงานคล้ายกับขา SIN ถ้าหากคอมพิวเตอร์ส่งข้อมูล "0" หรือทำให้ขาอ DTR มีแรงดันเป็น ±12V ทรานซิสเตอร์ Q2 ทำงาน ทำให้ขาคอลเล็กเตอร์ของ Q2 เสมือนต่อลงกราวด์เท่ากับว่าขา MCLR ของ PIC16C57 ถูกต่อลงกราวด์ด้วย อันเป็นการสร้างสัญญาณรีเซตให้แก่ PIC16C57 ดังนั้นในขณะที่ทำการโปรแกรมข้อมูลลงบน BS2 ซอฟต์แวร์ที่ใช้ในการโปรแกรม ซึ่งเรียกว่า STAMP2 โฮสต์โปรแกรม จะส่งพัลส์มายังขา ATN เพื่อรี

เซต ICI แล้วตามด้วยการส่งข้อมูลมายังขา SIN เพื่อแจ้งให้ทราบว่าต้องการเขียนโปรแกรมใหม่ลงบน BS2 แต่ถ้าหาก STAMP2 โฮสต์โปรแกรม ทำให้ขา ATN เป็น –12V หรือเป็นลอจิก "1" ก็จะหมายความว่าขณะนี้ BS2 อยู่ในโหมดรัน นั่นคืออยู่ในโหมดทำงานปรกติ



รูปที่ 2 - 4 การจัดขาของเบสิกแสตมป์ 2

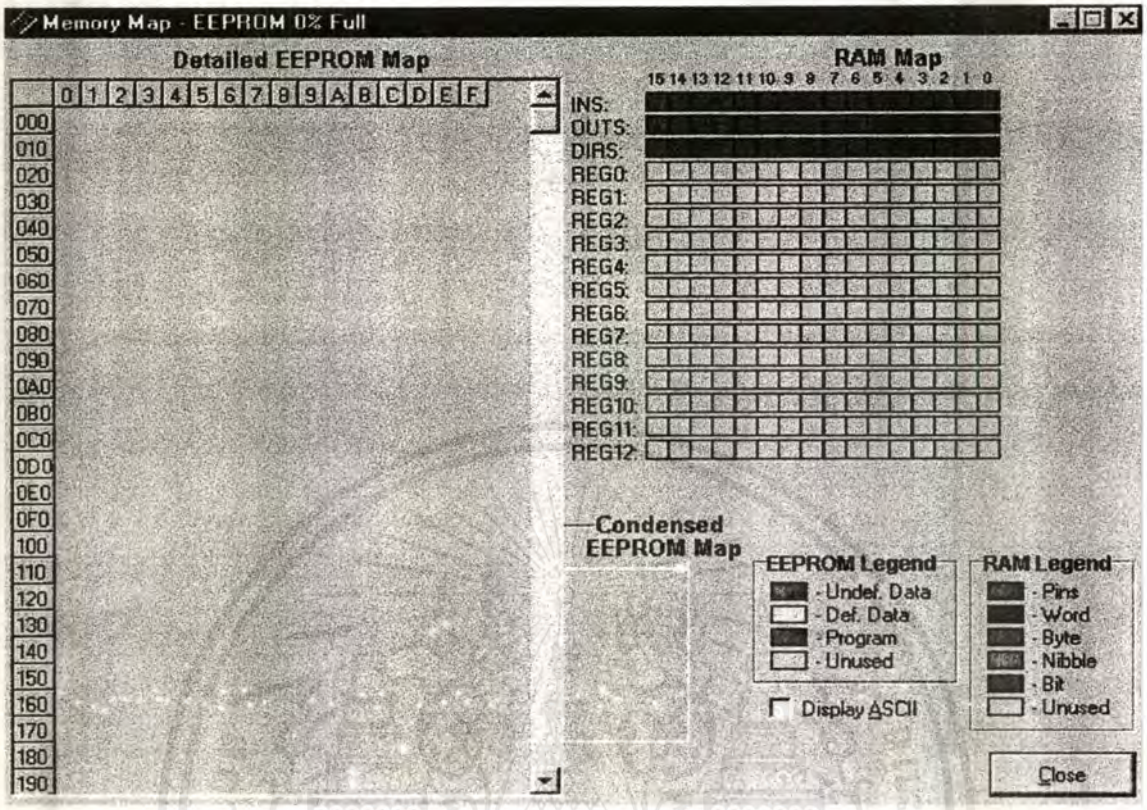
ในรูปที่ 2 – 4 แสดงการจัดขาของ BS2 พร้อมทั้งรายละเอียดเบื้องต้น BS2 มีขนาดเท่ากับไอซีตัวถัง DIL 24 ขา สามารถติดตั้งลงบนซ็อกเก็ตไอซี 24 ขาแบบธรรมดาได้ทันที สำหรับภาคจ่ายไฟและวงจรรีเซตมีการทำงานเหมือนกับ BS1 ทุกประการ



รูปที่ 2 –5 ไมโครคอนโทรลเลอร์ BASIC STAMP2

2.5 การจัดหน่วยความจำภายในเบสิกแสตมป์ 2

เบสิกแสตมป์ 2 มีหน่วยความจำอยู่ 2 ชนิดอยู่บนตัวมัน ชนิดแรกคือ หน่วยความจำแรม (RAM : Random Acess Memory) ซึ่งใช้สำหรับกำหนดค่าตัวแปรในการเขียนโปรแกรม และชนิดที่ 2 คือ หน่วยความจำอีอีพรอม (EEPROM : Electrically Erasable Programmable Read – Only Memory)



รูปที่ 2 – 6 แสดงการจัดหน่วยความจำภายในของเบสิกแสตมป์ 2

ใช้สำหรับการเก็บโปรแกรมและข้อมูลที่ต้องการคงอยู่เป็นระยะเวลายาวนาน เปรียบเทียบกับคอมพิวเตอร์แล้ว หน่วยความอีอีพรอมก็เหมือนกับฮาร์ดดิสก์นั่นเอง ซึ่งใช้ในการเก็บทั้งโปรแกรมและแฟ้มข้อมูล

ข้อแตกต่างระหว่างหน่วยความจำแรมกับหน่วยความจำอีอีพรอม คือ

1. ข้อมูลในหน่วยความจำแรมจะสูญหายไปเมื่อไม่มีไฟจ่ายให้เบสิกแสตมป์ 2 และเมื่อจ่ายไฟให้กับเบสิกแสตมป์ 2 อีกครั้งค่าต่าง ๆ ภายในหน่วยความจำแรมจะกลายเป็น 0

2. หน่วยความจำอีอีพรอมจะเก็บรักษาข้อมูลเอาไว้แม้ไม่ได้จ่ายไฟเลี้ยงและข้อมูลภายในหน่วยความจำอีอีพรอมนี้จะคงอยู่ไปตลอดจนกว่าจะมีการเขียนข้อมูลใหม่ หรือมีการโหลดโปรแกรมจากคอมพิวเตอร์ลงไปยังเบสิกแสตมป์ 2 ครั้งใหม่

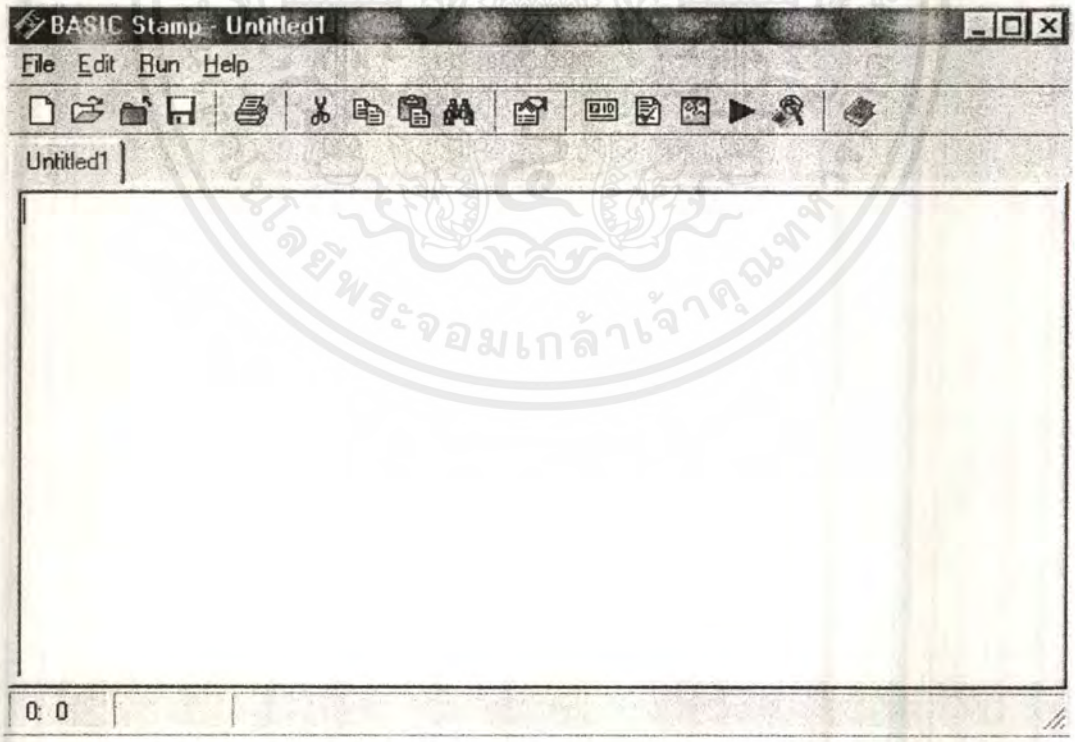
2.6 หน่วยความจำข้อมูลของเบสิกแสตมป์ 2

เบสิกแสตมป์ 2 มีหน่วยความจำข้อมูล 32 ไบต์ โดย 6 ไบต์สงวนไว้สำหรับเก็บข้อมูลของอินพุตเอาต์พุตและควบคุมทิศทางของขาอินพุตเอาต์พุต ดังนั้นจึงเหลือหน่วยความจำ 26 ไบต์สำหรับการใช้งานทั่วไป รูปที่ 2 – 6 แสดงให้เห็นถึงการจัดการหน่วยความจำของเบสิกแสตมป์ 2

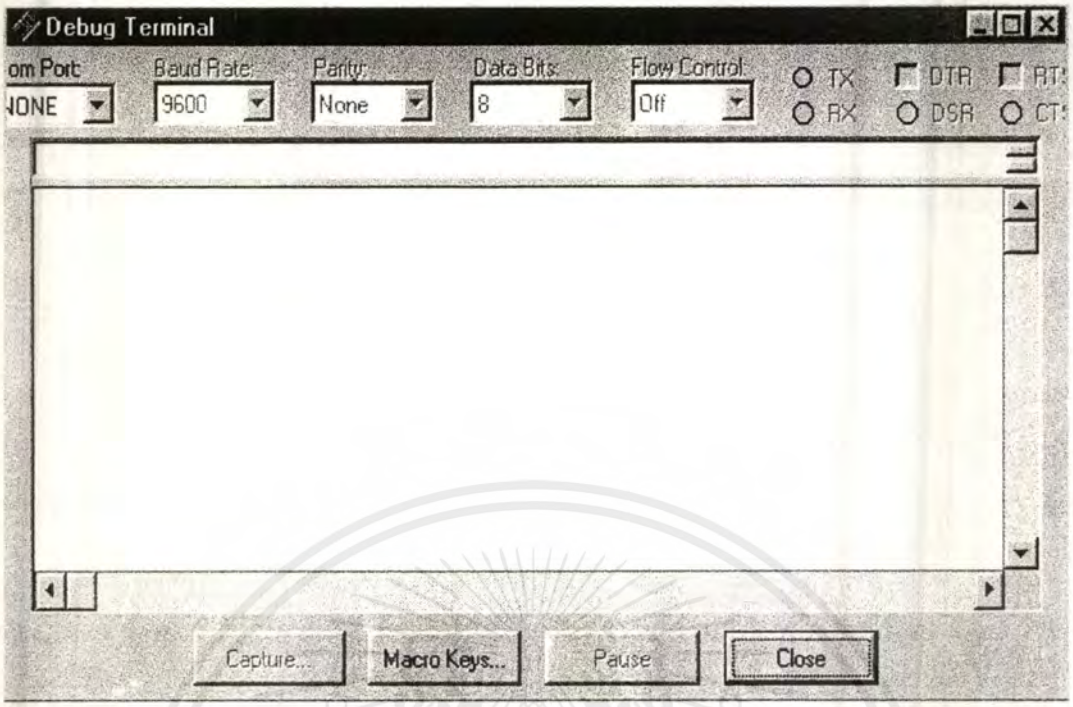
2.7 ตัวแปรอินพุต/เอาต์พุต

จากรูปที่ 2 – 6 รีจิสเตอร์ 3 ตัวแรกจะใช้ในการติดต่อกับบิตอินพุตเอาต์พุต ทั้ง 16 บิตซึ่งก็คือขาพอร์ต P0 – P 15 ของเบสิกแสตมป์ 2 นั่นเอง โดยรีจิสเตอร์ตัวแรกคือ INS ใช้สำหรับอ่านค่าอินพุตจากขา P0 ถึง P15 โดยรีจิสเตอร์ INS จะสามารถอ่านค่าได้อย่างเดียว ไม่สามารถเขียนค่าไปยังตัวมันได้ ในขณะที่รีจิสเตอร์ OUTS จะใช้สำหรับค่าออกไปยังขา P0 – P15 เมื่อมีการกำหนดให้ขาพอร์ตนี้เป็นขาเอาต์พุต พร้อมทั้งสามารถแลตช์ค่าเอาไว้ได้ด้วย ส่วนรีจิสเตอร์ DIRS ใช้เพื่อควบคุมทิศทางของขาพอร์ต P0 – P15 หรือใช้ในการกำหนดว่าขาใดเป็นอินพุตหรือเอาต์พุต และเขียนข้อมูล "I" เมื่อต้องการให้ขาพอร์นั้น ๆ เป็นเอาต์พุต

เมื่อจ่ายไฟให้กับเบสิกแสตมป์ 2 ครั้งแรก ค่าของหน่วยความจำทุกไบต์จะเคลียร์เป็น 0 ไม่เว้นแม้แต่รีจิสเตอร์ DIRS (DIRS = %0000000000000000) ดังนั้นเมื่อจ่ายไฟครั้งแรก ขาพอร์ตทั้งหมดจึงเป็นอินพุต เมื่อต้องการให้ขาพอร์ตใดเป็นเอาต์พุตให้เขียนข้อมูล "I" ไปยังรีจิสเตอร์ DIRS ในตำแหน่งของแต่ละขาพอร์ตนั้น เมื่อกำหนดให้เป็นขาเอาต์พุตแล้ว จะสามารถส่งค่าออกไปได้ด้วยการเขียนข้อมูลไปยังรีจิสเตอร์ OUTS ถ้าหากต้องการเก็บค่าที่ขาเอาต์พุตนั้นก็สามารถทำได้ โดยการอ่านค่าจากรีจิสเตอร์ด้วยคำสั่ง INS



รูปที่ 2 – 7 แสดงหน้าจอภาพหลังจากเริ่มต้นเรียกโปรแกรมเบสิกแสตมป์เอดิเตอร์ขึ้นมาใช้งาน



รูปที่ 2 – 8 แสดงหน้าต่าง DEBUG TERMINAL

2.8 การใช้งานโปรแกรม BASIC EDITOR

ในการเขียนโปรแกรมเพื่อควบคุมการทำงานของเบสิกแสตมป์ 2 นั้นจะใช้ภาษาเบสิกที่เรียกว่า พีเบสิก (PBASIC) โดยผ่านโปรแกรมที่ชื่อว่า เบสิกแสตมป์เอดิเตอร์ (BASIC Stamp Editor) โปรแกรมนี้จะถูกใช้ในการติดต่อคอมพิวเตอร์กับเบสิกแสตมป์ 2 ผ่านทางพอร์ตอนุกรม ใช้ในการเขียนและแก้ไขโปรแกรมภาษาเบสิกหรือทำหน้าที่เป็นเอดิเตอร์นั่นเอง โดยโปรแกรมจะทำการตรวจสอบการเขียนโปรแกรมในทุกบรรทัดว่าผิดพลาดหรือไม่ และใช้ในการโหลดหรือเขียนโปรแกรมลงบนตัวเบสิกแสตมป์เพื่อทำการรัน อาจกล่าวได้ว่า โปรแกรมเบสิกแสตมป์เอดิเตอร์ เพียงตัวเดียวทำงานได้ครบวงจร โดยไม่ต้องใช้โปรแกรมในการพัฒนาหลายตัวเหมือนกับไมโครคอนโทรลเลอร์ตัวอื่น ๆ ทำให้การเรียนรู้ไมโครคอนโทรลเลอร์เบสิกแสตมป์ง่ายและมีขั้นตอนน้อยที่สุด

2.9 รายละเอียดของโปรแกรมเบสิกแสตมป์เอดีเตอร์

เมนูเครื่องมือจัดการแฟ้มข้อมูลมาตรฐานและการพิมพ์

ชื่อของเมนูนี้คือ File มีด้วยกัน 5 เครื่องมือย่อยคือ New Document, Open file, Save, Close File และ Print แต่ละเครื่องมือย่อยมีการทำงานดังนี้

New Document ใช้สำหรับสร้างแฟ้มข้อมูลใหม่

Open File ใช้สำหรับเปิดแฟ้มข้อมูล นามสกุล BS2 สามารถใช้คีย์ลัดคือ Ctrl+O

Save ใช้สำหรับบันทึกแฟ้มข้อมูลในชื่อเดิม สามารถใช้คีย์ลัดคือ Ctrl+S

Save As ใช้สำหรับบันทึกข้อมูล เมื่อต้องการเปลี่ยนชื่อแฟ้มข้อมูล

Print ใช้สำหรับพิมพ์ซอร์สไฟล์ (แฟ้มข้อมูลภาษาเบสิกที่เขียนขึ้น) สามารถใช้คีย์ลัดคือ Ctrl+P

Close File ใช้สำหรับปิดแฟ้มข้อมูล

Exit ใช้สำหรับออกจากโปรแกรม

2.10 เมนูแก้ไขแฟ้มข้อมูล

ชื่อของเมนูนี้คือ Edit เป็นเมนูเครื่องมือที่มีลักษณะเหมือนกับโปรแกรมเท็กซ์เอดิเตอร์ (text editor) หรือโปรแกรมแก้ไขข้อความทั่วไป ดังมีรายละเอียดของเครื่องมือย่อยในเมนู Exit ต่อไปนี้

Undo ใช้สำหรับยกเลิกการใช้งานเครื่องมือแก้ไขข้อมูลก่อนหน้านี้ สามารถใช้คีย์ลัดคือ Ctrl+Z

Cut ใช้ตัดเพื่อย้ายตำแหน่งของข้อความที่เลือก สามารถใช้คีย์ลัดคือ Ctrl+X

Copy ใช้คัดลอกข้อความ สามารถใช้คีย์ลัดคือ Ctrl+C

Paste ใช้วางข้อความที่ตัดหรือคัดลอกมา สามารถใช้คีย์ลัดคือ Ctrl+V

Select All ใช้เลือกข้อความทั้งหมดในแฟ้มข้อมูล สามารถใช้คีย์ลัดคือ Ctrl+A

Find/Replace ใช้สำหรับค้นหาและแทนที่ข้อความ สามารถใช้คีย์ลัดคือ Ctrl+F

Find Next ใช้สำหรับให้ทำการค้นหาข้อความต่อไป สามารถใช้คีย์ลัดคือ ปุ่ม F3

Preference ใช้สำหรับปรับเปลี่ยนแก้ไขพารามิเตอร์ต่าง ๆ ของโปรแกรมเบสิกแสตมป์เอดิเตอร์ ซึ่งมีด้วยกัน 5 ส่วนคือ

1. ปรับเปลี่ยนการแสดงผลของหน้าต่างเอดิเตอร์ (Editor Appearance) ทั้งสีพื้นและสีของตัวอักษรและข้อความต่าง ๆ

2. ปรับเปลี่ยนการแสดงผลของหน้าต่าง Debug Terminal (Debug Appearance) ทั้งสีพื้นและสีของตัวอักษรและข้อความต่าง ๆ

3. กำหนดการทำงานของโปรแกรมเบสิกแสตมป์เอดิเตอร์ (Editor Operation)

4. กำหนดรูปแบบการทำงานของดีบัก (Debug Function)

5. กำหนดรายละเอียดพอร์ตของคอมพิวเตอร์ที่ใช้ในการดีบัก (Debug Port)

ถึงแม้ว่าที่หน้าต่าง Preference จะสามารถใช้ในการปรับเปลี่ยนการทำงานของโปรแกรมเบสิกแสตมป์เอดีเตอร์ได้อย่างมากมาย แต่ในการใช้งานจริง โปรแกรมจะทำการปรับเปลี่ยนและเลือกค่าที่เหมาะสมให้เองโดยอัตโนมัติผู้ใช้งานแทบจะไม่มีความจำเป็นต้องเข้ามาปรับเปลี่ยนหรือปรับแต่งใดๆ

2.11 เมนูแสดงการทำงาน

ชื่อของเมนูนี้คือ Run เป็นเมนูเครื่องมือที่มีบทบาทมากที่สุดในการพัฒนาและใช้งานเบสิกแสตมป์ มีเครื่องมือย่อยทั้งสิ้น 5 ตัวคือ

RUN ใช้สำหรับรันโปรแกรมที่เขียนขึ้น สามารถใช้คีย์ลัดคือ Ctrl+R

Cheek Syntax ใช้สำหรับตรวจสอบไวยากรณ์ของโปรแกรมที่เขียนขึ้นว่า มีการใช้คำสั่งที่ผิดพลาดหรือไม่ สามารถใช้คีย์ลัดคือ Ctrl+T

Memory Map ใช้สำหรับตรวจสอบค่าในหน่วยความจำโปรแกรมที่บรรจุอยู่ในหน่วยความจำอีอีพรอมของเบสิกแสตมป์และหน่วยความจำแรมภายใน สามารถใช้คีย์ลัดคือ Ctrl+M

Debug ใช้สำหรับติดต่อเบสิกแสตมป์กับคอมพิวเตอร์ เมื่อใช้เครื่องมือนี้จะปรากฏหน้าต่าง Debug Terminal ขึ้นสามารถใช้คีย์ลัดคือ Ctrl+D

Identify ใช้สำหรับค้นหาเบสิกแสตมป์ที่ต่อกับพอร์ตอนุกรม สามารถใช้คีย์ลัดคือ Ctrl+I

ที่เมนูเครื่องมือนี้จะมีเพียง Memory Map และ Debug เท่านั้นที่เมื่อเรียกใช้งานแล้วจะปรากฏหน้าต่างแสดงการทำงานขึ้นมาใหม่ นอกนั้นจะแสดงการทำงานผ่านบาร์แสดงสถานะ (status bar) ซึ่งอยู่ด้านล่างของหน้าต่างหลักของโปรแกรม นอกเหนือไปจากนั้นเครื่องมือทั้งหมดในเมนูนี้ยกเว้น Check Syntax จะสามารถใช้งานได้ก็ต่อเมื่อเบสิกแสตมป์เชื่อมต่อเข้ากับคอมพิวเตอร์และสามารถติดต่อสื่อสารกันได้

2.12 รายละเอียดชุดคำสั่งของเบสิกแสตมป์ 2

คำสั่งหลักในโปรแกรมพีเบสิกแสตมป์ 2 มีทั้งสิ้น 36 คำสั่ง สามารถแบ่งออกเป็นกลุ่มตามลักษณะการทำงานได้ 11 กลุ่ม ดังนี้

1. กลุ่มคำสั่งการกระโดด มีทั้งสิ้น 5 คำสั่งคือ

IF...THEN เปรียบเทียบเงื่อนไขก่อนกระโดด

BRANCH กระโดดไปยังตำแหน่งที่กำหนดตามค่าของตัวแปร

GOTO กระโดดไปยังแอดเดรสใด ๆ

GOSUB กระโดดไปยังโปรแกรมย่อย

RETURN กระโดดออกจากโปรแกรมย่อย

2. กลุ่มคำสั่งเกี่ยวกับลูป มี 1 คำสั่งคือ

FOR...NEXT กำหนดจำนวนรอบที่ต้องการวนลูปหรือวนทำงานซ้ำ

3. กลุ่มคำสั่งเลือกข้อมูล มี 3 คำสั่งคือ

LOOKUP เปิดตารางข้อมูล

LOOKDOWN ค้นหาตัวเลขที่เหมือนกัน แล้วเก็บค่าไว้ในตัวแปร

RANDOM สุ่มตัวเลข

4. กลุ่มคำสั่งควบคุมการทำงานของขาพอร์ต มี 13 คำสั่งคือ

INPUT กำหนดให้ทำงานเป็นอินพุต

OUTPUT กำหนดให้ทำงานเป็นเอาต์พุต

REVERSE เปลี่ยนจากขาอินพุตเป็นเอาต์พุตหรือจากเอาต์พุตเป็นอินพุต

LOW ทำให้ขาเอาต์พุตเป็นลอจิก "O"

HIGH ทำให้ขาเอาต์พุตเป็นลอจิก "I"

TOGGLE ทำให้ขาเอาต์พุตกลับสถานะลอจิก

PULSIN วัดสัญญาณพัลส์อินพุต (ความละเอียด 2 ไมโครวินาที)

PULSOUT ส่งสัญญาณพัลส์ออก (ความละเอียด 2 ไมโครวินาที)

BUTTON แก้การเบาซ์ของสวิตซ์

SHIFTIN เลื่อนข้อมูลเข้าจากแบบขนานเป็นแบบอนุกรม

SHIFTOUT ส่งข้อมูลออกจากแบบอนุกรม

COUNT นับจำนวนไซเกิลของสัญญาณอินพุต (มีค่า 0 – 125kHz)

XOUT กำเนิดรหัสควบคุมสำหรับอุปกรณ์ต่อพ่วงอนุกรม X – 10

5. กลุ่มคำสั่งติดต่อพอร์ตอนุกรม มี 2 คำสั่งคือ

SERIN รับข้อมูลอนุกรมเข้า มีรูปแบบข้อมูลแบบ N81 หรือ E71

SEROUT ส่งข้อมูลอนุกรมในรูปแบบ N81 หรือ E71 ออกไปทางขวา SOUT ของเบสิกแสตมป์

6. กลุ่มคำสั่งควบคุมขาพอร์ตอินพุตเอาต์พุตแบบอะนาลอก มี 2 คำสั่งคือ

PWM สร้างสัญญาณ PWM ขนาด 0 – 5 V ออกไปทางขาพอร์ต

RCTIME วัดค่าเวลาประจุและคายประจุของวงจร RC สามารถนำไปใช้ในการวัดค่าของตัวต้านทานปรับค่าได้

7. กลุ่มคำสั่งจัดการด้านเสียง มี 2 คำสั่งคือ

FREQOUT กำเนิดสัญญาณไซน์หนึ่งหรือสองความถี่ ตั้งแต่ 0 – 32.767kHz

DTMFOUT กำเนิดสัญญาณ DTMF ของระบบโทรศัพท์

8. กลุ่มคำสั่งเข้าถึงหน่วยความจำอีอีพรอมภายในเบสิกแสตมป์ มี 3 คำสั่งคือ

DATA เก็บข้อมูลลงในหน่วยความจำอีอีพรอมก่อนทำการโหลดโปรแกรมพีเบสิก

READ อ่านข้อมูลระดับไบต์จากหน่วยความจำอีอีพรอมมาเก็บไว้ในตัวแปร

WRITE เขียนข้อมูลระดับไบต์ไปเก็บไว้ในหน่วยความจำอีอีพรอม

9. กลุ่มคำสั่งจัดการด้านเวลา มี 1 หนึ่งคำสั่งคือ

PAUSE หน่วยเวลา 0 – 65, 536 มิลลิวินาที

10. กลุ่มคำสั่งควบคุมพลังงาน มี 3 คำสั่งคือ

NAP หยุดทำงานในช่วงเวลาสั้น ๆ การกินพลังงานจะลดลง

SLEEP กำหนดให้หยุดทำงานได้นาน 1 – 65, 536 วินาที เมื่อกำหนดให้ทำงานในโหมดสลีปนี้เบสิกแสตมป์ 2 กินกระแสไฟฟ้าลดลงเหลือเพียง 50μA

END ทำงานในโหมดสลีปจนกว่าจะมีการจ่ายไฟหรือต่อเบสิกแสตมป์ 2 เข้ากับคอมพิวเตอร์ เมื่อกระทำคำสั่งนี้เบสิกแสตมป์ 2 จะกินกระแสไฟฟ้าลดลงเหลือ 50μA

11. กลุ่มคำสั่งแก้ไขโปรแกรม มี 1 คำสั่ง

DEBUG แสดงค่าของตัวแปรผ่านทางคอมพิวเตอร์

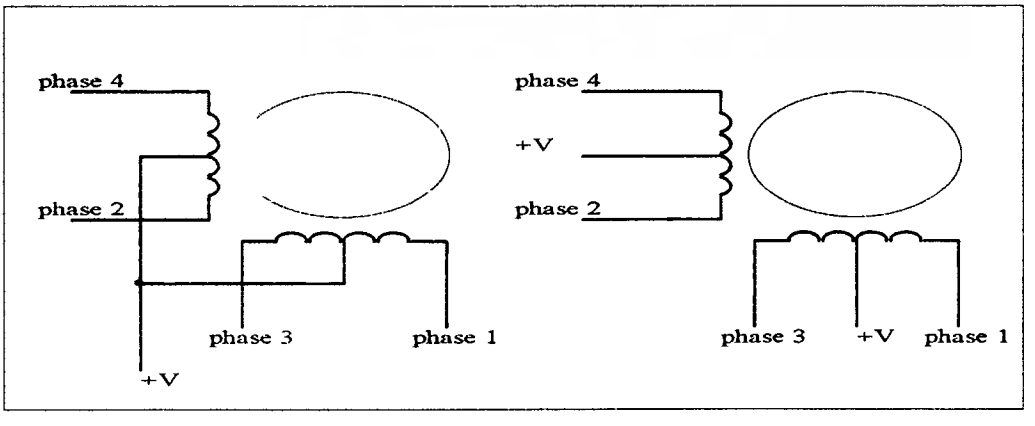
สำหรับรายละเอียดของคำสั่งทั้งหมดของเบสิกแสตมป์ 2 ที่จะทำการอธิบายต่อจากนี้ เรียงลำดับตามตัวอักษรภาษาอังกฤษตั้งแต่ตัว B เป็นต้นไป โดยในแต่ละคำสั่งก็จะมีตัวอย่างการใช้งาน และในบางคำสั่งก็จะมีตัวอย่างโปรแกรมสั้น ๆ ที่สามารถทำการทดลองกับบอร์ดทดลอง NX – BS2 เพื่อตรวจสอบผลการทำงานได้ในทันที อันเป็นการช่วยเพิ่มความเข้าใจในคำสั่งนั้น ๆ แก่ผู้เรียนและผู้ใช้งานได้เป็นอย่างดี

2.13 เรื่องโครงสร้างสเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์ (stepper motor) เป็นมอเตอร์แบบหนึ่งที่มีการหมุนแบบเป็นจังหวะหรือเรียกว่าเป็นสเต็ป กล่าวคือ เมื่อจ่ายสัญญาณให้แก่มอเตอร์อย่างถูกต้อง มอเตอร์จะหมุนไปเป็นจังหวะไม่เหมือนกับมอเตอร์ทั่ว ๆ ไป ที่เมื่อทำการจ่ายไฟแล้ว ก็จะหมุนไปในทันที ส่วนจะหมุนซ้ายหรือขวาขึ้นอยู่กับขั้วของการจ่ายไฟให้แก่ขดลวดภายในมอเตอร์ ความละเอียดของการหมุนของสเต็ปเปอร์มอเตอร์จะขึ้นอยู่กับองค์ประกอบสำคัญของ 2 ประการคือ โครงสร้างของสเต็ปเปอร์มอเตอร์และวงจรขับ โดยวงจรขับสามารถที่จะช่วยให้การหมุนของสเต็ปเปอร์มอเตอร์มีความละเอียดมากขึ้นได้

ตัวอย่างการใช้งานสเต็ปเปอร์มอเตอร์ที่เห็นได้ชัดคือ ใช้ในการขับเคลื่อนหัวพิมพ์ของเครื่องพิมพ์ ไม่ว่าจะเป็นเครื่องพิมพ์แบบหัวเข็มหรือเครื่องพิมพ์ในระบบพ่นหมึก ในเครื่องพล็อตเตอร์และใช้ในฟลอปปี้ดิสก์ไดรฟ์ เหตุผลที่สเต็ปเปอร์มอเตอร์ได้รับความนิยมอย่างมากก็คือ สามารถกำหนดตำแหน่งของการหมุนได้อย่างแม่นยำ โดยไม่ต้องใช้วงจรขับที่มีความซับซ้อนมากนัก ยกตัวอย่าง หากต้องการให้มอเตอร์หมุนขวาไปเป็นระยะทาง 15 องศาจากจุดเริ่มต้น หากใช้มอเตอร์ธรรมดา จะต้องสร้างวงจรป้อนกลับเพื่อตรวจสอบตำแหน่งต้นทางและปลายทาง เพื่อควบคุมให้มอเตอร์หยุดหมุนได้อย่างแม่นยำ รวมถึงต้องหาวิธีการลดพลังงานและสัญญาณรบกวนในขณะที่มอเตอร์หยุดหมุนชั่วขณะด้วย ในขณะที่หากใช้สเต็ปเปอร์มอเตอร์ที่มีความละเอียด 7.5 องศาต่อสเต็ป เพียงป้อนสัญญาณให้มอเตอร์หมุนไป 2 สเต็ป ก็จะได้ตำแหน่งตามต้องการ โดยที่ไม่ต้องใช้วงจรซับซ้อน และเมื่อมอเตอร์ไม่หมุนก็จะไม่มีการใช้พลังงาน จึงไม่เกิดความสูญเสียมากมายดังเช่นมอเตอร์แบบธรรมดา

ไมโครคอนโทรลเลอร์นับเป็นอีกหนึ่งอุปกรณ์ที่มักถูกนำมาใช้ในการควบคุมการทำงานของสเต็ปเปอร์มอเตอร์ ทดแทนวงจรขับที่ใช้ไอซีดิจิตอล เนื่องจากไมโครคอนโทรลเลอร์สามารถกำหนดรูปแบบการขับได้อย่างหลากหลายภายใต้ฮาร์ดแวร์เดียวกัน ดังนั้นจึงเป็นเรื่องจำเป็นอย่างยิ่งที่จะต้องศึกษาวิธีการนำเบสิกแสตมป์ 2 ซึ่งถือได้ว่าเป็นไมโครคอนโทรลเลอร์สมัยใหม่ความสามารถสูงไปใช้ในการควบคุมการทำงานของสเต็ปเปอร์



รูปที่ 2 - 9 โครงสร้างพื้นฐานของสเต็ปเปอร์มอเตอร์แบบยูนิโพล่าร์

2.14 สเต็ปเปอร์มอเตอร์แบบยูนิโพล่าร์

สเต็ปเปอร์มอเตอร์ที่มีใช้งานในปัจจุบัน มีด้วยกัน 2 แบบคือ แบบยูนิโพล่าร์และแบบไบโพล่าร์ สำหรับในที่นี้ขออธิบายเฉพาะแบบยูนิโพล่าร์ เนื่องจากในปัจจุบันสเต็ปเปอร์มอเตอร์แบบไบโพล่าร์แทบจะไม่มีการใช้งานอีกแล้ว เนื่องจากวงจรที่ใช้ขับมีความซับซ้อนและใช้อุปกรณ์จำนวนมากในขณะที่การขับสเต็ปเปอร์มอเตอร์แบบยูนิโพล่าร์ ง่ายกว่า และใช้อุปกรณ์น้อยกว่ามาก

โครงสร้างของสเต็ปเปอร์มอเตอร์แบบยูนิโพล่าร์แสดงดังในรูปที่ 2.9 สเต็ปเปอร์มอเตอร์แบบนี้จะมีการพันขดลวด 2 ขดบนแต่ละขั้วแม่เหล็กของสเตเตอร์ แต่ละขดแบ่งเป็น 2 เฟส รวมมอเตอร์ทั้งตัวมีขดลวดทั้งสิ้น 4 เฟส คือ เฟส 1, 2, 3 และ 4 มีการต่อสายออกมาจากขดลวดแต่ละขดเพื่อจ่ายไฟเลี้ยง ทำให้สเต็ปเปอร์มอเตอร์แบบนี้จึงมีทั้งแบบ 5 และ 6 สาย โดยสายที่ 5 คือ สายจ่ายไฟเลี้ยง ส่วนกรณี 6 สาย จะต้องนำสายไฟเลี้ยงของขดลวดทั้ง 2 ของมอเตอร์มาต่อร่วมกัน แล้วจ่ายไฟจึงจะทำให้สเต็ปเปอร์มอเตอร์ทำงานได้

2.15 การขับสเต็ปเปอร์มอเตอร์

การขับให้สเต็ปเปอร์มอเตอร์หมุนจะต้องป้อนสัญญาณไฟฟ้าไปยังแต่ละเฟสของขดลวดอย่างเหมาะสมและมีรูปแบบที่ถูกต้อง สเต็ปเปอร์มอเตอร์จึงจะหมุนได้ โดยมีรูปแบบในการขับอย่างง่าย 3 รูปแบบคือ แบบหนึ่งเฟส, แบบสองเฟส หรือฟูลสเต็ป (full step) และแบบฮาล์ฟสเต็ป (half step)

แบบหนึ่งเฟสเป็นการขับที่มีรูปแบบง่ายที่สุด โดยทำการป้อนสัญญาณกระตุ้นขดลวดครั้งละเฟสในช่วงเวลาหนึ่งไล่เรียงกันไป เช่น เริ่มต้นจากเฟสที่ 1 ต่อด้วยเฟสที่ 2, 3 และ 4 แล้ววนกลับมาเฟสที่ 1 ใหม่ หรือจะให้เริ่มจากเฟสที่ 1 ไปยังเฟสที่ 4, 3 และ 2 แล้ววนกลับมาเฟสที่ 1 อีกครั้ง ด้วยลำดับการป้อนสัญญาณกระตุ้นที่ต่างกัน ทำให้ทิศทางในการหมุนของสเต็ปเปอร์สวนทางกันด้วย การขับสเต็ปเปอร์มอเตอร์แบบนี้ จะมีขดลวดเพียงเฟสเดียวที่ได้รับสัญญาณกระตุ้น ในตารางที่ 2 – 1 แสดงลำดับการป้อนสัญญาณเพื่อขับสเต็ปเปอร์มอเตอร์แบบหนึ่งเฟส

สเต็ปที่	Phase 1	Phase 2	Phase 3	Phase 4
1	ทำงาน	-	-	-
2	-	ทำงาน	-	-
3	-	-	ทำงาน	-
4	-	-	-	ทำงาน

ตารางที่ 2 – 1 ลำดับการป้อนสัญญาณกระตุ้นของวงจรขับสเต็ปเปอร์มอเตอร์แบบหนึ่งเฟส

สเต็ปที่	Phase 1	Phase 2	Phase 3	Phase 4
1	ทำงาน	ทำงาน	-	-
2	-	ทำงาน	ทำงาน	-
3	-	-	ทำงาน	ทำงาน
4	ทำงาน	-	-	ทำงาน

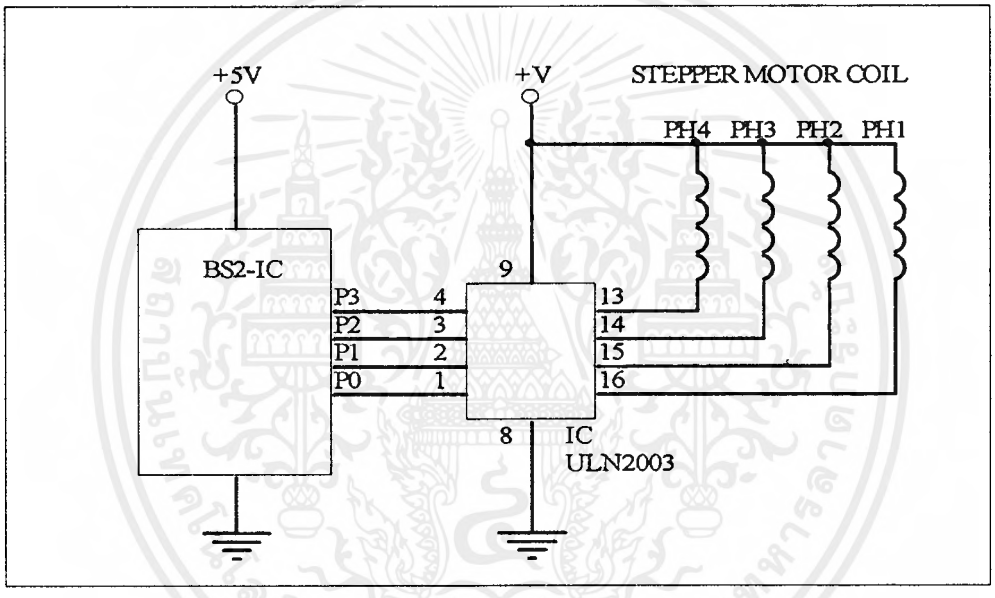
ตารางที่ 2 – 2 ลำดับการป้อนสัญญาณกระตุ้นของวงจรขับสเต็ปเปอร์มอเตอร์แบบสองเฟส

สเต็ปที่	Phase 1	Phase 2	Phase 3	Phase 4
1	ทำงาน	-	-	-
2	ทำงาน	ทำงาน	-	-
3	-	ทำงาน	-	-
4	-	ทำงาน	ทำงาน	-
5	-	-	ทำงาน	-
6	-	-	ทำงาน	ทำงาน
7	-	-	-	ทำงาน
8	ทำงาน	-	-	ทำงาน

ตารางที่ 2 – 3 ลำดับการป้อนสัญญาณกระตุ้นของวงจรขับสเต็ปเปอร์มอเตอร์แบบฮาล์ฟสเต็ป

แบบสองเฟสจะมีลักษณะคล้ายกับแบบหนึ่งเฟส บาทีเรียกการขับแบบนี้ว่า แบบฟูลสเต็ป (full step) แต่แทนที่จะส่งสัญญาณกระตุ้นเพียงเฟสเดียว ในการขับแบบนี้จะป้อนสัญญาณกระตุ้นไปยังเฟสของมอเตอร์ที่อยู่ใกล้กัน ในเวลาเดียวกันและเรียงลำดับกันไปเช่นเดียวกับแบบหนึ่งเฟส ดังแสดงในตารางที่ 2 – 2 คือเริ่มต้นด้วยป้อนสัญญาณกระตุ้นไปยังเฟสที่ 1 และ 2 พร้อมกันในสเต็ปที่ 1 ในสเต็ปที่ 2 จะป้อนสัญญาณไปยังเฟสที่ 2 และ 3 ถัดมาในเฟสที่ 3 จะทำการป้อนสัญญาณกระตุ้นไปที่เฟส 3 และ 4 ในสเต็ปที่ 4 จะป้อนสัญญาณไปยังเฟสที่ 4 และ 1 แล้ววนกลับไปที่เฟสที่ 1 และ 2 อีกครั้ง ด้วยการขับแบบนี้ทำให้ได้แรงบิดหรือทอค (toque) มากกว่าแบบหนึ่งเฟส แต่ข้อเสียคือ ใช้พลังงานในการขับเพิ่มมากขึ้น

แบบฮาล์ฟสเต็ป การขับแบบนี้ได้รับความนิยมมากที่สุด เนื่องจากสามารถช่วยให้สเต็ปเปอร์มอเตอร์สามารถหมุนได้อย่างละเอียดมากขึ้นเป็นสองเท่าของความละเอียดปกติของสเต็ปเปอร์มอเตอร์ โดยมีรูปแบบการขับให้หมุนแสดงในตารางที่ 2 –3 จะเห็นได้ว่า การขับสเต็ปเปอร์มอเตอร์แบบนี้เป็นการผสมผสานระหว่างการขับแบบหนึ่งและสองเฟส กล่าวคือ มีทั้งการป้อนสัญญาณกระตุ้นไปยังขดลวดเพียงเฟสเดียวและแบบพร้อมกันสองเฟสในช่วงเวลาหนึ่ง ด้วยการขับแบบนี้ส่งผลให้แรงบิดที่ได้จากการหมุนเพิ่มขึ้น เพราะระยะทางในการหมุนสั้นลง ความถูกต้องของตำแหน่งที่หมุนเพิ่มมากขึ้น เพียงแต่ว่าในการขับของแต่ละสเต็ปจะให้ผลเพียงครึ่งสเต็ปของการขับปกติ ดังนั้นหากต้องการให้การเคลื่อนที่เป็นไปแบบเต็มสเต็ป จะต้องกำหนดให้ทำการหมุนไป 2 สเต็ปต่อเนื่องกัน



รูป 2 – 10 ตัวอย่างวงจรขับสเต็ปเปอร์มอเตอร์ของเบสิกแสตมป์ 2

2.16 การขับสเต็ปเปอร์มอเตอร์โดยใช้เบสิกแสตมป์ 2

เนื่องจากขาพอร์ตของเบสิกแสตมป์ 2 เมื่อทำงานเป็นขาเอาต์พุต มีความสามารถในการจ่ายกระแสประมาณ 10 – 20 mA ดังนั้นจึงไม่สามารถนำไปขับสเต็ปเปอร์มอเตอร์แบบยูนิโพล่าร์ได้โดยตรง จะต้องส่งสัญญาณที่ใช้ในการขับสเต็ปเปอร์มอเตอร์ผ่านไอซีหรือวงจรขับกระแสสูง ในที่นี้ขอแนะนำไอซีขับโหลดกระแสและแรงดันสูงเบอร์ ULN2003 มีตัวอย่างวงจรแสดงในรูปที่ 2 – 10

ภายในไอซี ULN2003 เป็นวงจรอินเวอร์เตอร์หรือวงจรกลับสถานะลอจิก ดังนั้นในการป้อนสัญญาณเพื่อขับสเต็ปเปอร์มอเตอร์จะต้องป้อนด้วยลอจิก “1” และการต่อเฟสของมอเตอร์เข้ากับ ULN2003 จะต้องต่อในลักษณะคอลเล็กเตอร์เปิด กล่าวคือ ขดลวดด้านหนึ่งต้องต่อกับไฟเลี้ยงมอเตอร์ ส่วนอีกด้านหนึ่งต่อเข้ากับไอซีขับเบอร์ ULN2003 เมื่อ ULN2003 ได้รับสัญญาณลอจิก “1” ก็จะกลับ

เป็นลอจิก "0" ทำให้เสมือนว่า ขดลวดถูกต่อลงกราวด์ทำให้เกิดการครบวงจร มีกระแสไหลผ่านขดลวดในเฟสนั้น ๆ อย่างสมบูรณ์ มอเตอร์ก็จะเคลื่อนที่ไปหนึ่งสเต็ป

การเขียนโปรแกรมควบคุมต้องกำหนดเสียก่อนว่าต้องการขับสเต็ปเปอร์มอเตอร์ในลักษณะใด หนึ่งเฟส สองเฟส หรือแบบฮาล์ฟสเต็ป จากนั้นจึงทำการเขียนข้อมูลส่งออกไปยังขาพอร์ตที่ต่อกับไอซีขับ ULN2003 ทุกครั้งที่ส่งข้อมูลออกไปในแต่ละสเต็ปต้องทำการหน่วงเวลาเล็กน้อย ก่อนส่งข้อมูลของการหมุนในสเต็ปต่อไป ทั้งนี้เพื่อให้ไอซีขับและตัวสเต็ปเปอร์มอเตอร์สามารถตอบสนองกับข้อมูลก่อนหน้านี้ได้ทัน ก่อนที่จะรับข้อมูลใหม่ต่อไป

2.17 ข้อมูลสำหรับขับสเต็ปเปอร์มอเตอร์

ข้อมูลที่เบสิกแสตมป์ 2 จะต้องส่งออกไปยัง ULN2003 เพื่อขับสเต็ปเปอร์มอเตอร์จะขึ้นอยู่กับรูปแบบของการขับ ในกรณีขับแบบหนึ่งเฟส ข้อมูลสำหรับขับสเต็ปเปอร์มอเตอร์จะเป็นดังนี้

สเต็ปที่	ข้อมูล			
	ฐานสอง	ฐานสิบ	ฐานสิบหก	
1	%0001	1	$1	
2	%0010	2	$2	
3	%0100	4	$4	
4	%1000	8	$8	
5	%0001	1	$1	(เริ่มต้นวนรอบใหม่)

ส่วนข้อมูลของการขับสเต็ปเปอร์มอเตอร์แบบสองเฟส มีดังนี้

สเต็ปที่	ข้อมูล			
	ฐานสอง	ฐานสิบ	ฐานสิบหก	
1	%0011	3	$3	
2	%0110	6	$6	
3	%1100	12	$C	
4	%1001	9	$9	
5	%0011	3	$3	(เริ่มต้นวนรอบใหม่)

สุดท้ายข้อมูลของการขับสเต็ปเปอร์มอเตอร์แบบฮาล์ฟสเต็ปซึ่งเป็นแบบที่ใช้ในโครงงานนี้ มีดังนี้

สเต็ปที่	ข้อมูล		
	ฐานสอง	ฐานสิบ	ฐานสิบหก
1	%0001	1	$1
2	%0011	3	$3
3	%0010	2	$2
4	%0110	6	$6
5	%0100	4	$4
6	%1100	12	$C
7	%1000	8	$8
8	%1001	9	$9
9	%0001	1	$1 (เริ่มต้นวนรอบใหม่)

ถ้าหากต้องการขับสเต็ปเปอร์มอเตอร์ให้หมุนไปในทิศทางตรงข้าม สามารถทำได้โดยการส่งข้อมูลย้อนกลับจากที่กำหนดไว้ในโปรแกรมตัวอย่าง ยกตัวอย่าง ในกรณีหนึ่งเฟสให้ทำการส่งข้อมูล %1000 หรือ $8 ออกไปก่อน ตามด้วย %0100, %0010 และ %0001 ในที่สุด จากนั้นวนกลับมาที่ %1000 อีกครั้ง

การขับให้สเต็ปเปอร์มอเตอร์หมุนครบหนึ่งรอบ จะใช้จำนวนพัลส์ที่แตกต่างกันไปขึ้นอยู่กับโครงสร้างของมอเตอร์เองและรูปแบบในการขับ กล่าวคือ หากมอเตอร์มีความละเอียด 7.5 องศาต่อสเต็ป ดังนั้นถ้าต้องการให้หมุนครบรอบต้องส่งพัลส์ทั้งสิ้น 48 ลูก คำนวนได้จากการเคลื่อนที่เป็นวงกลมหนึ่งรอบจะมีมุมรวมทั้งสิ้น 360 องศา นำความละเอียดของสเต็ปเปอร์มอเตอร์มาหาร 360 ก็จะได้ค่าเท่ากับ 48 ในกรณีที่ใช้การขับแบบฮาล์ฟสเต็ป จำนวนพัลส์ที่ใช้ก็จะเพิ่มเป็น 2 เท่า ถ้าหากใช้มอเตอร์ 7.5 องศาต่อสเต็ป ต้องส่งพัลส์ทั้งสิ้น 96 ลูก สำหรับการขับให้มอเตอร์หมุนครบ 1 รอบ

ความเร็วของการหมุนจะขึ้นอยู่กับอัตราการส่งข้อมูลไปยังวงจรขับและการตอบสนองของตัวมอเตอร์เอง ถ้าหากส่งข้อมูลได้เร็วเท่าใด มอเตอร์ก็จะหมุนเร็วขึ้นเท่านั้น อย่างไรก็ตาม ตัวสเต็ปเปอร์มอเตอร์ก็มีข้อจำกัดในการรับสัญญาณพัลส์ที่ส่งมากระตุ้นเช่นกัน หากส่งสัญญาณมาเร็วเกินไปสเต็ปเปอร์มอเตอร์จะหยุดหมุน ทั้งนี้เนื่องจากตัวสเต็ปเปอร์มอเตอร์ไม่สามารถตอบสนองพัลส์กระตุ้นที่ส่งมาได้ทัน

2.18 การจัดหน่วยความจำและการใช้งานหน่วยความจำอีอีพรอมภายในเบสิกแสตมป์ 2

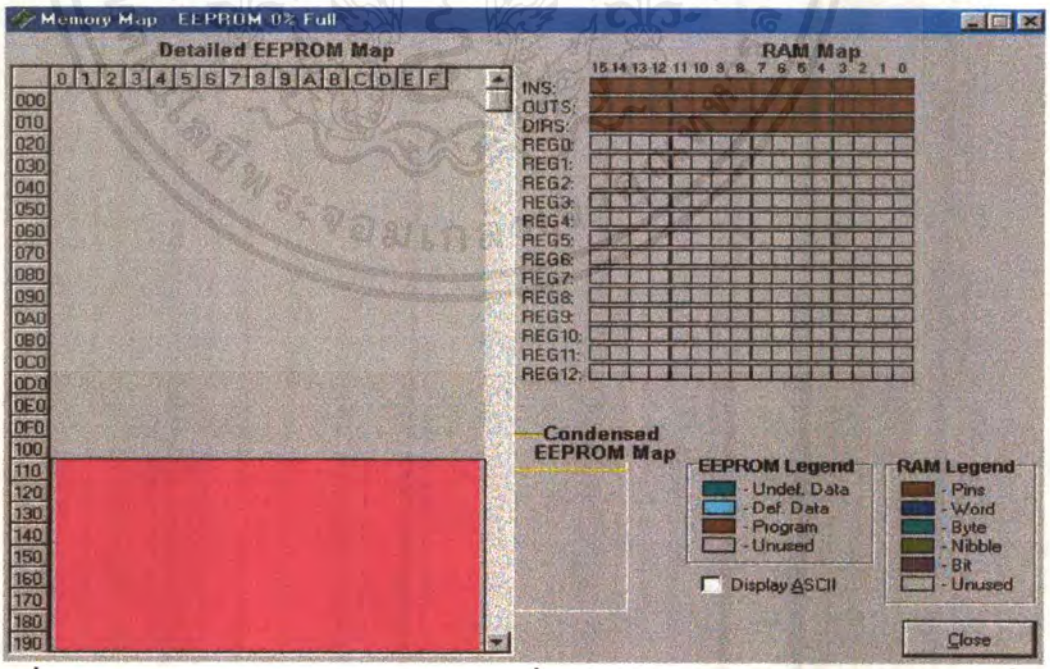
เบสิกแสตมป์ 2 มีคุณสมบัติที่เหมือนกับไมโครคอนโทรลเลอร์ความสามารถสูงสมัยใหม่คือมีหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลอยู่ภายในอย่างสมบูรณ์ โดยหน่วยความจำโปรแกรมจะมีขนาด 2 กิโลไบต์ เป็นหน่วยความจำอีอีพรอมที่สามารถลบและเขียนใหม่ได้ถึง 10 ล้าน

รอบ บรรจุคำสั่งภาษาพีเบสิกได้มากถึง 500 คำสั่ง ส่วนหน่วยความจำข้อมูลมีด้วยกัน 2 ส่วนคือ หน่วยความจำแรมซึ่งบรรจุอยู่ในไมโครคอนโทรลเลอร์ PIC16C57 ซึ่งเป็นเบสิกอินเตอร์พรีตเตอร์ โดยหน่วยความจำในส่วนนี้จะเรียกว่า รีจิสเตอร์ (register) มีขนาด 32 ไบต์ หรือ 16 เวิร์ด (1 เวิร์ด เท่ากับ 16 บิต หรือ 2 ไบต์) หน่วยความจำข้อมูลอีกส่วนหนึ่งคือ หน่วยความจำอีอีพรอม ซึ่งใช้พื้นที่ร่วมกับหน่วยความจำโปรแกรม โดยในการแยกส่วนนั้นจะใช้วิธีการเข้าถึงในทิศทางที่สวนกัน กล่าวคือ ข้อมูลของหน่วยความจำโปรแกรมจะถูกเก็บลงตั้งแต่แอดเดรส $7FF ลดถอยลงมาเรื่อย ๆ ในขณะที่ข้อมูลของหน่วยความจำข้อมูลจะเริ่มเก็บจากแอดเดรส $000 ในรูปที่ 2 – 13 แสดงการจัดสรรพื้นที่ของหน่วยความจำทั้งหมดภายในเบสิกแสตมป์ 2

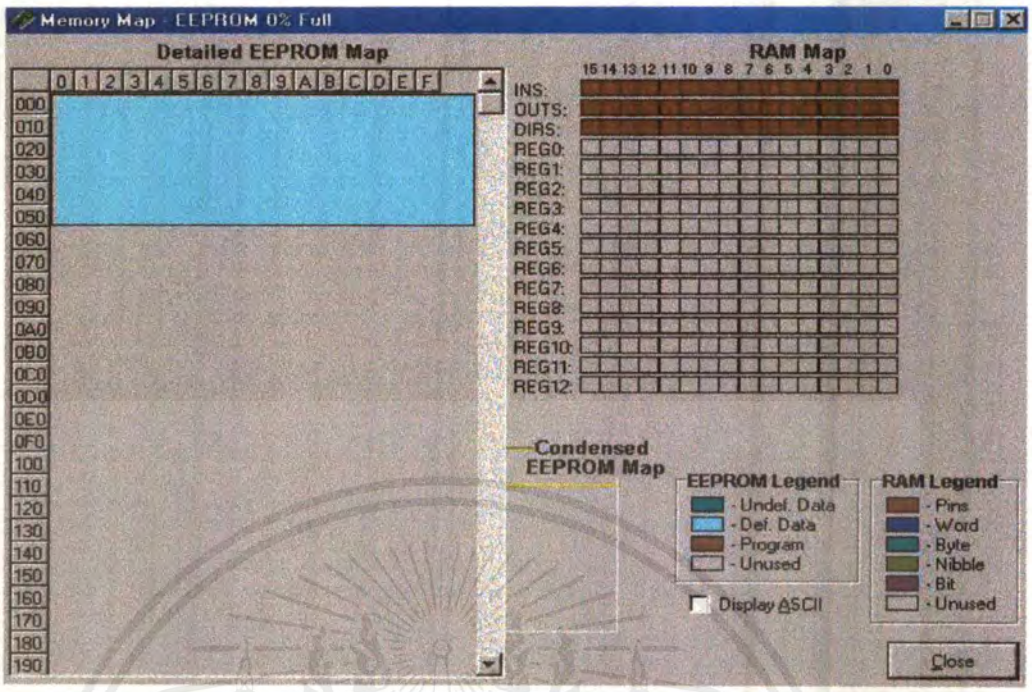
2.19 การดูรายละเอียดของหน่วยความจำทั้งหมดภายในเบสิกแสตมป์ 2

เริ่มต้นด้วยการเชื่อมต่อเบสิกแสตมป์ 2 กับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม จากนั้นทำการเรียกโปรแกรมนามสกุล .BS2 ขึ้นมา ตามด้วยโหลดโปรแกรมลงบนเบสิกแสตมป์ 2 จากนั้นกดปุ่ม Memory map หรือเลือกเมนู RUN ไปที่ Memory Map หรือกดคีย์ Ctrl+M จะปรากฏหน้าต่างของการจัดสรรหน่วยความจำภายในเบสิกแสตมป์ 2 ขึ้นมาตามรูปที่ 2 – 11

รูปที่ 2 – 11(ก,ข) หน้าต่าง Memory Map ของโปรแกรมเบสิกแสตมป์เอดิเตอร์ ซึ่งใช้ในการดูและตรวจสอบข้อมูลภายในหน่วยความจำอีอีพรอม



รูปที่ 2 – 11(ก) แสดงหน่วยความจำโปรแกรม ซึ่งบรรจุอยู่ในตอนล่างของหน่วยความจำ เริ่มตั้งแต่แอดเดรส $7FF การบรรจุข้อมูลจะนับแอดเดรสย้อนกลับมา (ตำแหน่งแอดเดรสลดลง) แสดงด้วยพื้นที่สีแดงเข้ม



รูปที่ 2 – 11(ข) แสดงหน่วยความจำข้อมูล ซึ่งบรรจุอยู่ในตอนบนของหน่วยความจำ เริ่มตั้งแต่แอดเดรส $000 แสดงด้วยพื้นที่สีฟ้าอ่อน

จากในรูปที่ 2 – 11 (ก) แสดงข้อมูลที่บรรจุอยู่ในหน่วยความจำโปรแกรมของเบสิกแสตมป์ 2 จะเห็นได้ว่า ข้อมูลของหน่วยความจำโปรแกรมจะได้รับการเขียนลงในแอดเดรสจากตำแหน่งสุดท้าย คือ $7FF ย้อนขึ้นมา โดยแสดงด้วยพื้นที่สีแดง ส่วนบริเวณสีเทาคือพื้นที่ที่ยังไม่มีการใช้งาน ในพื้นที่สีฟ้าเข้มในส่วนของหน่วยความจำแรมแสดงถึงพื้นที่ของรีจิสเตอร์ที่มีการใช้งานแล้ว โดยในบล็อกย่อยของหน่วยความจำจะมีขนาด 1 เวิร์ดหรือ 2 ไบต์หรือ 16 บิต

ในรูปที่ 2 – 11 (ข) แสดงให้เห็นถึงพื้นที่ของหน่วยความจำข้อมูลที่ได้รับการจัดสรรในพื้นที่ของหน่วยความจำอีอีพรอม โดยใช้สีฟ้าอ่อนแสดงขอบเขตของหน่วยความจำข้อมูลส่วนนี้

2.20 รีจิสเตอร์ภายในเบสิกแสตมป์ 2

รีจิสเตอร์ภายในเบสิกแสตมป์ 2 ก็คือ หน่วยความจำแรมนั่นเอง โดยจะแบ่งออกเป็น 2 ส่วน คือ รีจิสเตอร์ควบคุมทิศทางอินพุตเอาต์พุตและรีจิสเตอร์ใช้งานทั่วไป สำหรับรีจิสเตอร์ควบคุมทิศทางอินพุตเอาต์พุตมี 3 ตัวคือ DIR ใช้กำหนดโหมดการทำงานของขาพอร์ตของเบสิกแสตมป์ 2 ถ้าบิตใดเป็น “0” หมายความว่า บิตนั้นเป็นอินพุต ในขณะที่ถ้าเป็น “1” บิตที่กำหนดนั้นจะเป็นเอาต์พุต รีจิสเตอร์ตัวถัดมาคือ IN ใช้เก็บข้อมูลที่อ่านเข้ามาของขาพอร์ตที่กำหนดให้เป็นอินพุต และสุดท้ายคือ OUT ใช้เก็บข้อมูลที่ต้องการส่งออกไปยังขาพอร์ตของเบสิกแสตมป์ 2

ดังนั้นเบสิกแสตมป์ 2 จึงมีรีจิสเตอร์ที่สามารถใช้งานทั่วไปได้ 13 เวิร์ดหรือ 26 ไบต์ โดยข้อมูลในรีจิสเตอร์สามารถกำหนดให้อยู่ในรูปของบิต, นิบเบิล, ไบต์ และเวิร์ดได้ นั่นหมายความว่าผู้ใช้งานสามารถกำหนดตัวแปรระดับไบต์สำหรับเก็บข้อมูลได้ถึง 26 ตัวคือ b0 – b25 และตัวแปรระดับเวิร์ด 13 ตัวคือ w0 – w12 โดยเมื่อใช้งาน w0 จะเป็นการใช้งานพื้นที่ของ b0 กับ b1 โดย b0 เก็บข้อมูลใน 8 บิตบนหรือไบต์บน ส่วน b1 เก็บข้อมูลของ 8 บิตล่างหรือไบต์ล่าง ในขณะที่ w1 จะใช้ b2 และ b3 ร่วมกัน เป็นต้น

2.21 หน่วยความจำอีอีพรอมภายในเบสิกแสตมป์ 2

เป็นที่ทราบอยู่แล้วว่า ข้อมูลของโปรแกรมทั้งหมดของเบสิกแสตมป์ 2 จะถูกเก็บไว้ในหน่วยความจำโปรแกรมชนิดอีอีพรอม ซึ่งเป็นหน่วยความจำแบบนอนโวลาไทล์ (non – volatile) แม้ว่าไม่มีการจ่ายไฟเลี้ยง ข้อมูลก็จะยังคงอยู่ไม่สูญหาย ในเบสิกแสตมป์ 2 ใช้หน่วยความจำเบอร์ 24LC16 มีขนาด 2 กิโลไบต์ มีแอดเดรสตั้งแต่ $000 - $7FF หรือ 0 – 2,048 ตำแหน่ง สามารถเก็บรักษาข้อมูลได้นาน 10 ปี

หน่วยความจำอีอีพรอมภายในเบสิกแสตมป์ 2 ได้รับการจัดสรรเป็น 2 ส่วน คือ ส่วนหนึ่ง (ซึ่งเป็นส่วนใหญ่ด้วย) ใช้เก็บข้อมูลของโปรแกรมควบคุมการทำงาน และอีกส่วนหนึ่งใช้เก็บข้อมูลทั่วไป ส่วนใหญ่จะเป็นตารางข้อมูล (data table) ข้อมูลทั้งสองส่วนนี้ไม่มีการแบ่งแยกด้วยขอบเขตหรือขนาดแต่อย่างใด เพียงแต่ใช้วิธีการกำหนดแอดเดรสในการบรรจุข้อมูลลงไปในทิศทางที่สวนกัน ดังที่ได้กล่าวมาแล้วในตอนต้น

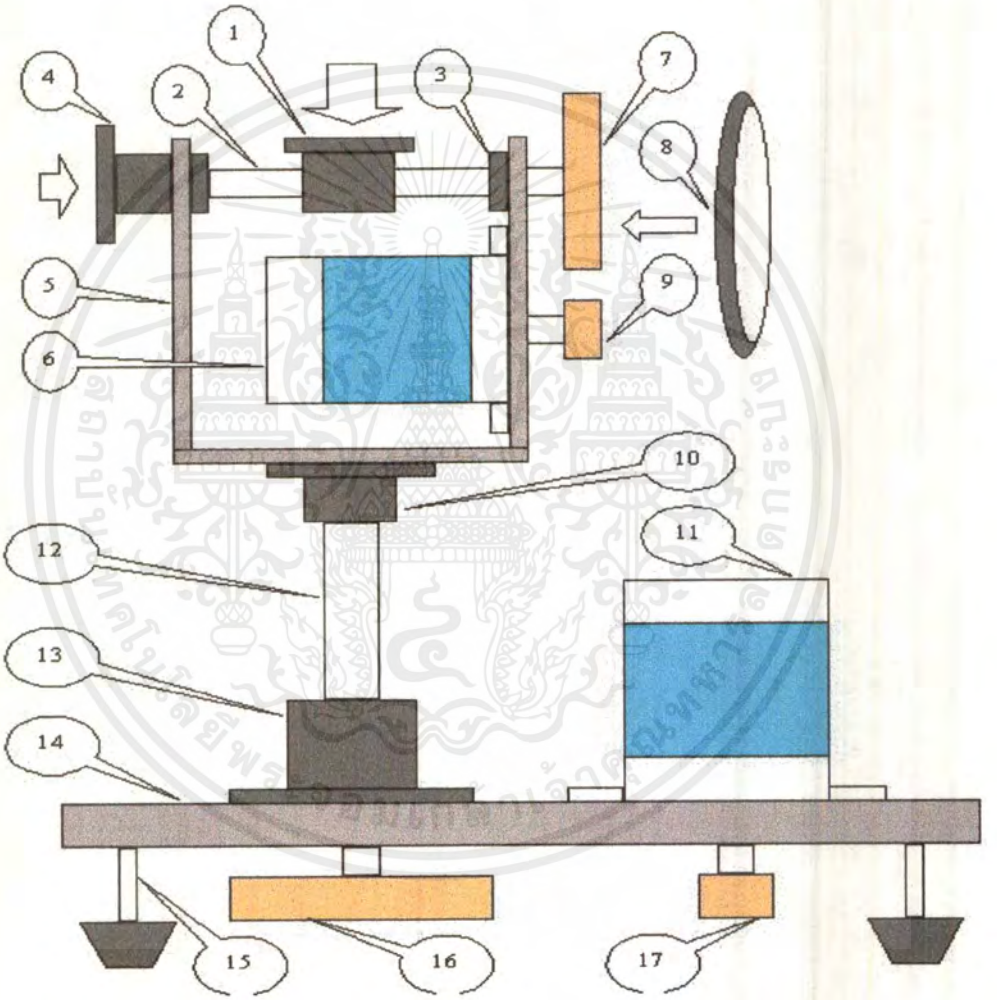
สำหรับการอ่านและเขียนข้อมูลภายในหน่วยความจำอีอีพรอมของเบสิกแสตมป์ 2 สามารถกระทำได้โดยใช้คำสั่งสำหรับอ่านและเขียนโดยเฉพาะ มีด้วยกัน 3 คำสั่งคือ WRITE สำหรับเขียน, READ สำหรับอ่านข้อมูล และ DATA ซึ่งเป็นคำสั่งสำหรับเขียนค่าของข้อมูลลงสู่หน่วยความจำอีอีพรอมโดยตรง เมื่อทำการโหลดโปรแกรมลงบนเบสิกแสตมป์ 2 หรือเป็นคำสั่งสำหรับกำหนดตารางข้อมูลนั่นเอง

บทที่ 3

โครงสร้างและการออกแบบ

3.1 การออกแบบโครงสร้างของฐานกล้อง

ฐานกล้องสามารถเคลื่อนที่ได้ 4 ทิศทาง คือปรับมุมเงยได้ 0-90 องศา และหมุนได้ 0-180 องศา การทำงานจะถูกควบคุมโดยอุปกรณ์อิเล็กทรอนิกส์และโปรแกรมของไมโครคอนโทรลเลอร์เบสิกแสตมป์ 2 โครงสร้างทั้งหมดของฐานกล้องทำจากเหล็ก มีส่วนประกอบต่างๆดังนี้



รูปที่ 3 – 1 โครงสร้างของฐานกล้อง

ส่วนประกอบต่างๆมีดังนี้

1) จุดยึดกล้องดิจิตอล
2) แกนหมุนปรับมุมเงย
3) บูธยึดแกนหมุน

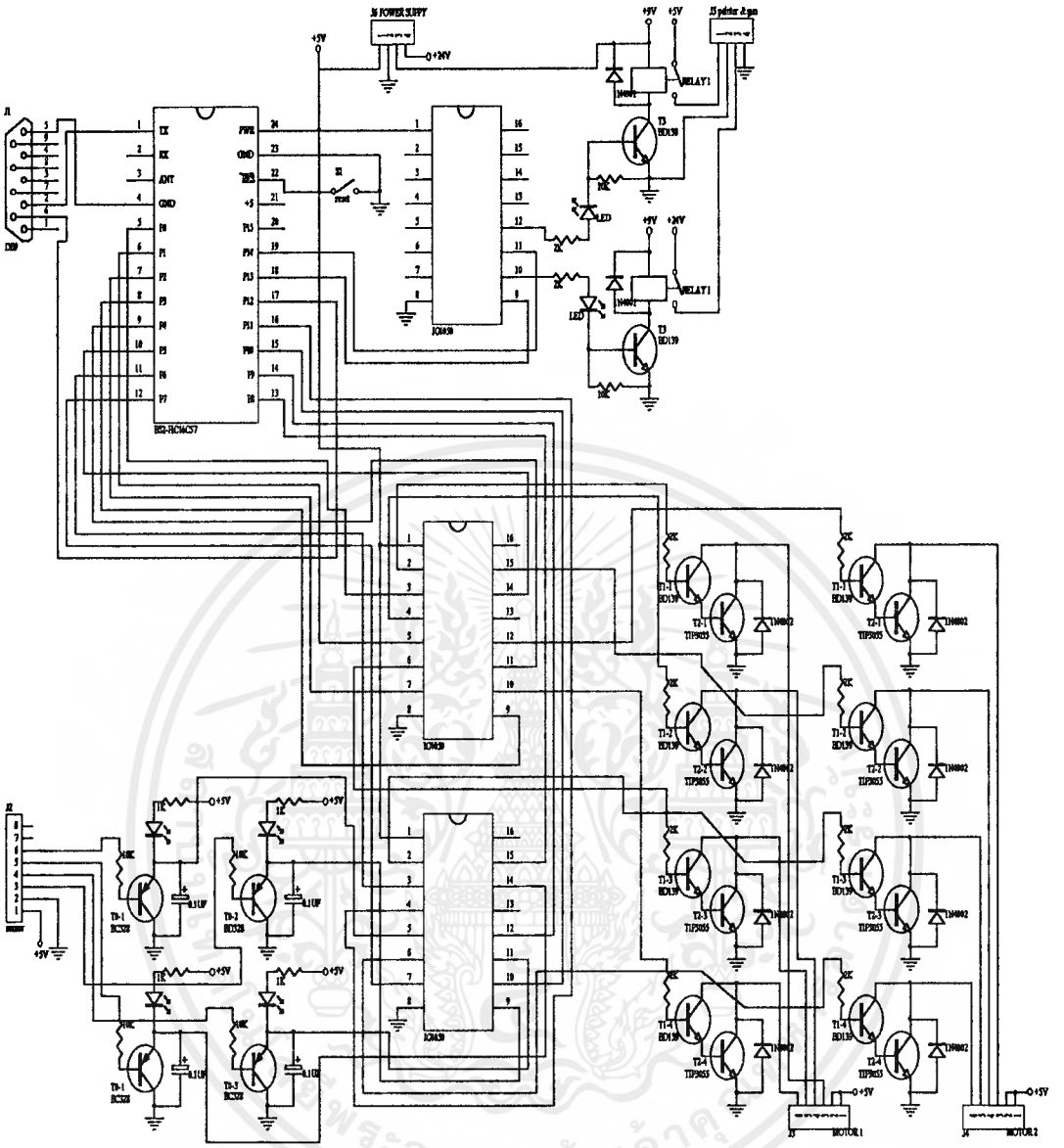
4) จุดล๊อกแกน
5) โครงสร้างด้านบน
6) มอเตอร์ปรับมุมเงย
7) เฟื่องแกนหมุนปรับมุมเงย
8) สายพาน
9) เฟื่องมอเตอร์
10) จุดเชื่อมต่อโครงสร้างด้านบนกับด้านล่าง
11) มอเตอร์ควบคุมการหมุน
12) แกนหมุน
13) ตลับลูกปืน
14) ฐานเหล็ก
15) ขาตั้ง
16) เฟื่องแกนหมุน
17) เฟื่องมอเตอร์

เมื่อทำการประกอบชิ้นส่วนต่างๆ เสร็จแล้วก็จะมีลักษณะดังรูปที่ 3-2 ซึ่งแสดงตำแหน่งของอุปกรณ์ต่างๆ ที่ยึดไว้เรียบร้อยแล้ว อุปกรณ์ที่นำมาติดตั้งนั้นได้แก่ กล้องดิจิตอล วงจรรับส่งอินฟาเรด ในการออกแบบนั้นได้ออกแบบให้การติดตั้งอุปกรณ์ต่างๆมีจุดยึดที่สามารถถอดออกได้สะดวก เพื่อประโยชน์ในการปรับปรุงประสิทธิภาพการทำงานต่อไป

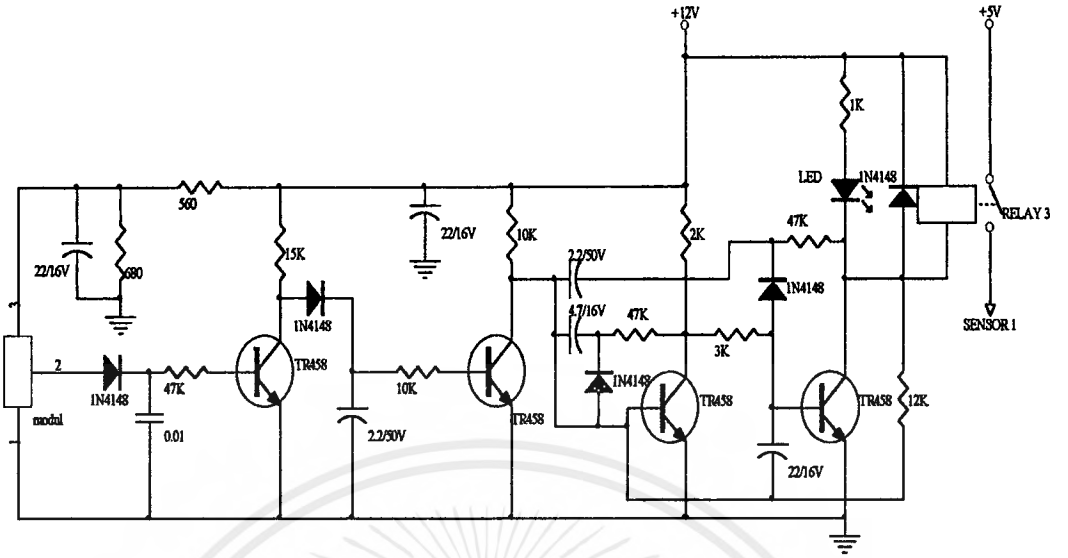
รูปที่ 3- 2 โครงสร้างฐานกล้องพร้อมอุปกรณ์ต่างๆ

3.2 การออกแบบวงจรควบคุมการทำงาน

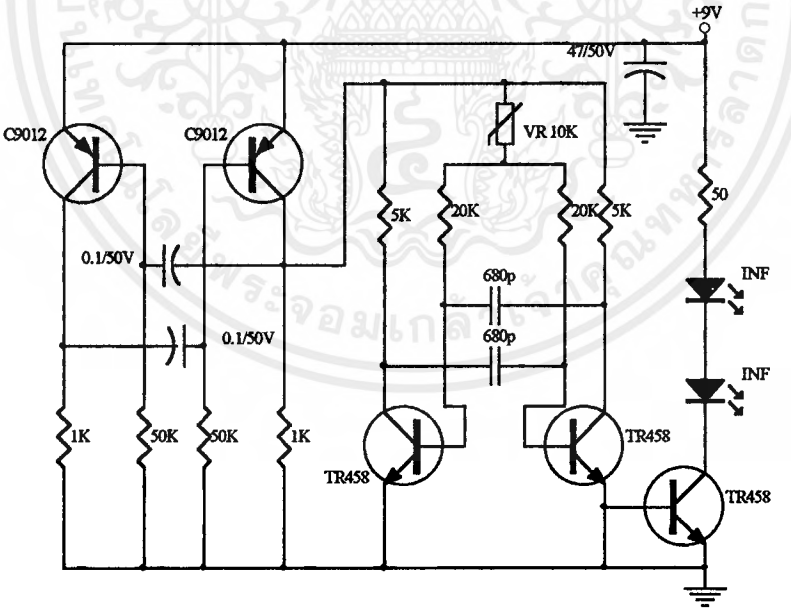
การออกแบบวงจรควบคุมการทำงานนั้นประกอบด้วยวงจรของไมโครคอลโทรลเลอร์เบสิกแสตมป์ 2 ซึ่งเชื่อมต่อกับวงจรบัฟเฟอร์และวงจรไดรเวอร์ ซึ่งทำหน้าที่ขยายกระแสในการควบคุมการทำงานของมอเตอร์ให้มีประสิทธิภาพสูงสุด การควบคุมมอเตอร์นั้นสิ่งสำคัญคือกระแสที่จ่ายให้กับขดลวดของมอเตอร์ต้องคงที่เสมอ เพราะการหมุนแต่ละครั้งนั้นต้องมีความแม่นยำ การออกแบบแหล่งจ่ายที่ไม่มีประสิทธิภาพจะทำให้ค่าตำแหน่งการหมุนผิดพลาดได้ การออกแบบวงจรจ่ายกระแสให้ระบบนั้นจึงจำเป็นต้องมีวงจรขยายกระแสที่มีประสิทธิภาพสูง ในการออกแบบโครงงานนี้ได้ออกแบบวงจรพาวเวอร์ซัพพลายที่ใช้ทรานซิสเตอร์ MJ2955 จำนวน 3 ตัว ในการควบคุมการไหลของกระแสไฟฟ้าให้คงที่ โดยมีไอซีเร็กกูเลทเบอร์ 7805 ควบคุมระดับแรงดันที่จ่ายให้กับมอเตอร์



รูปที่ 3 – 3 วงจรหลักของโครงงาน

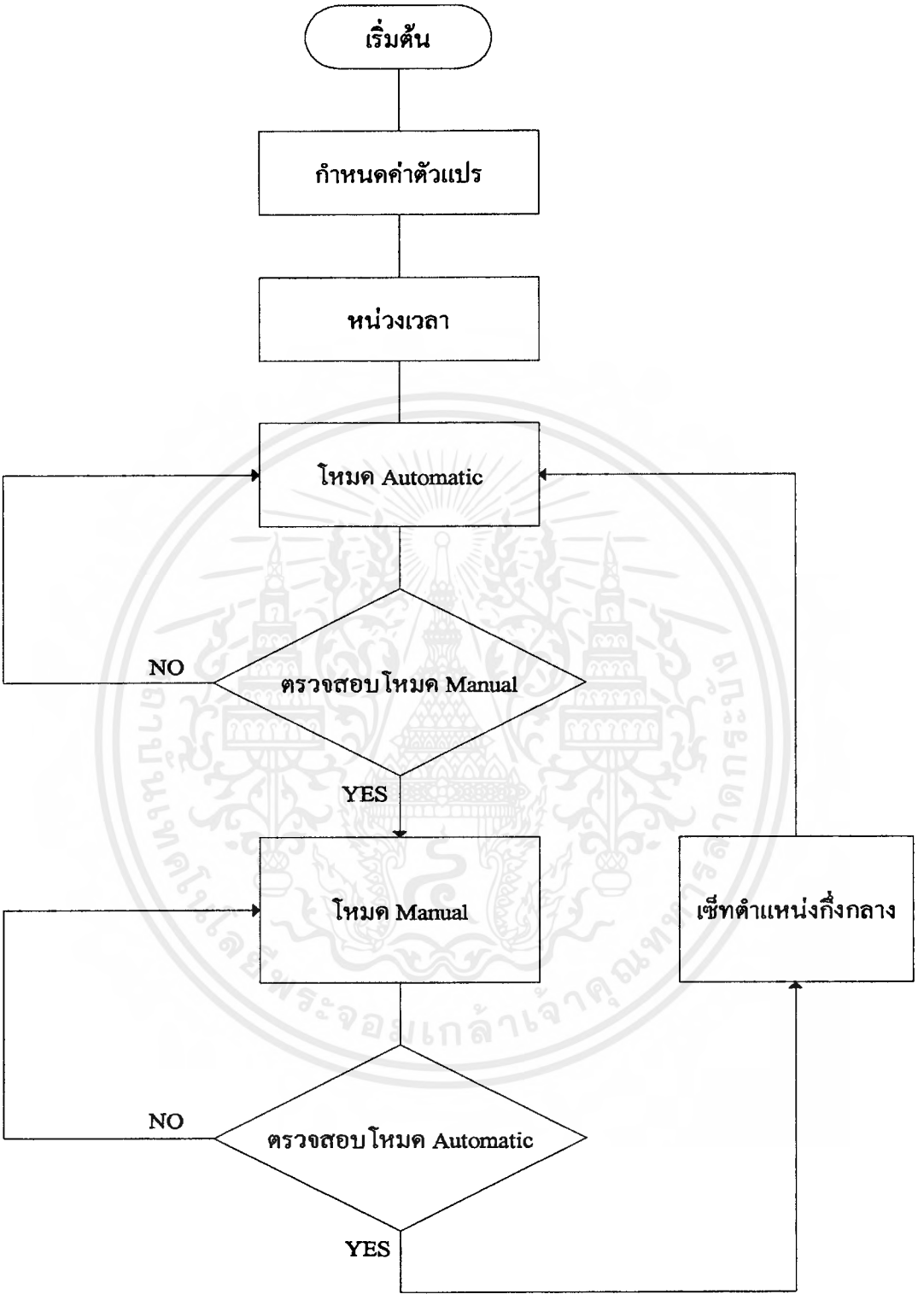


รูปที่ 3 – 4 วงรับอินฟาเรด



รูปที่ 3 – 5 วงจรส่งอินฟาเรด

วงจรรับส่งอินฟาเรดที่ใช้ในโครงงานมีรัศมีในการตรวจจับประมาณ 3 เมตร โดยอาศัยการสะท้อนของแสงอินฟาเรด ซึ่งในการใช้งานจะไม่สามารถแยกความแตกต่างระหว่างคนและวัตถุได้ แต่วงจร sensor ที่ใช้หลักการสะท้อนของแสงอินฟาเรดนั้นมีต้นทุนที่ต่ำมาก จึงมีความเหมาะสมในการออกแบบโครงงานที่เน้นความประหยัด



รูปที่ 3-6 โฟลวชาร์ตแสดงการทำงานของไมโครคอนโทรลเลอร์เบสิกแสตมป์

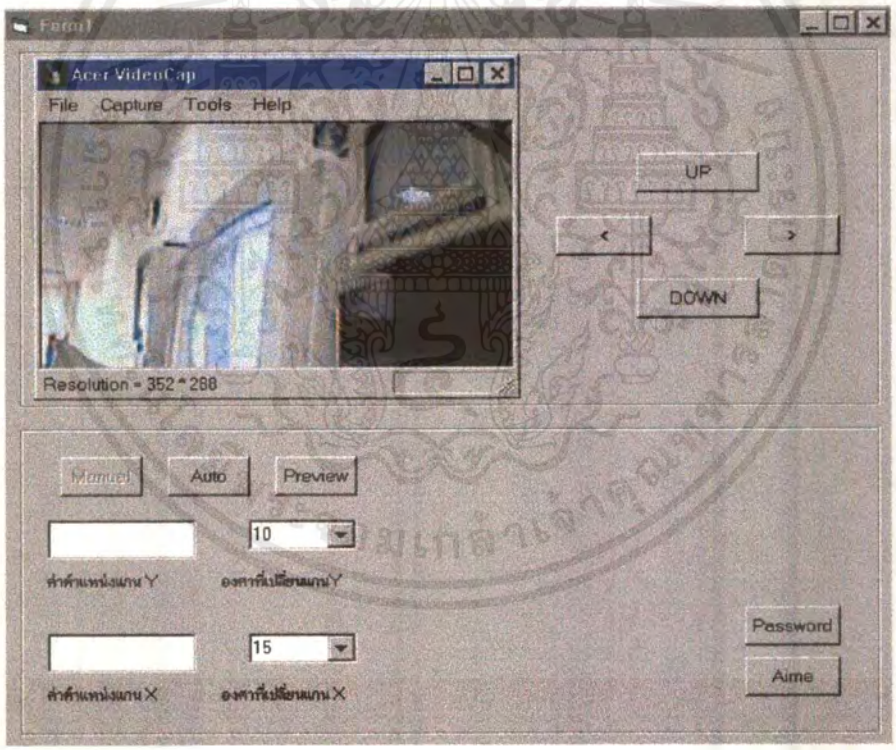
3.3 การทำงาน

3.3.1 การทำงานโหมด Automatic

เมื่อเริ่มต้นการทำงาน ระบบจะทำงานในโหมด Automatic ก่อนเสมอ โดยจะทำการตรวจสอบสภาวะของ sensor ทั้ง 4 ตัว เมื่อมีผู้บุกรุกเข้ามาระบบควบคุมกล้องอัตโนมัติจะหมุนฐานกล้องไปจับภาพผู้บุกรุกไว้ ถ้าผู้บุกรุกเปลี่ยนตำแหน่งฐานกล้องก็จะหมุนตามไปด้วย มุมในการจับภาพของกล้องจะอยู่ในช่วง 0-180 องศา

3.3.2 การทำงานโหมด Manual

เมื่อผู้ใช้เปลี่ยนโหมดการทำงานเป็น manual โดยการใช้ mouse กดปุ่มเลือกโหมดการทำงานบนหน้าจอคอมพิวเตอร์ดังแสดงในรูปที่ 3-6 ระบบควบคุมกล้องอัตโนมัติจะหยุดการทำงานในโหมด Automatic และหลังจากนั้นจะรอรับค่าการทำงานจากคอมพิวเตอร์ผ่านพอร์ตอนุกรม



รูปที่ 3-7 หน้าต่างที่ใช้งานบนเครื่องคอมพิวเตอร์

ปุ่มที่ใช้	ค่าที่กำหนด
ปุ่มขึ้น 2.25 องศา	55
ปุ่มขึ้น 10 องศา	59
ปุ่มขึ้น 30 องศา	67
ปุ่มลง 2.25 องศา	81
ปุ่มลง 10 องศา	85
ปุ่มลง 30 องศา	93
ปุ่มซ้าย 3.6 องศา	29
ปุ่มซ้าย 15 องศา	32
ปุ่มซ้าย 45 องศา	40
ปุ่มซ้าย 90 องศา	53
ปุ่มขวา 3.6 องศา	3
ปุ่มขวา 15 องศา	6
ปุ่มขวา 45 องศา	14
ปุ่มขวา 90 องศา	27
ปุ่ม Manual	100
ปุ่ม Auto	101

ตารางที่ 3-1 แสดงค่าที่ส่งไปยังไมโครคอนโทรลเลอร์เมื่อกดปุ่มต่างๆ

การกดปุ่มต่างๆ บนหน้าจอคอมพิวเตอร์ เมื่อค่าที่ส่งไปให้ไมโครคอลโทรลเลอร์มีค่าตรงกับค่าที่กำหนดไว้ในตาราง ไมโครคอนโทรลเลอร์จะควบคุมให้ตัวฐานกล้องทำงานตามคำสั่งนั้นๆ

บทที่ 4

การสร้างโปรแกรมติดต่อระหว่างคอมพิวเตอร์กับผู้ใช้เพื่อควบคุมการทำงานของระบบรักษาความปลอดภัย

4.1 บทนำ

ในปัจจุบันได้มีโปรแกรมที่ใช้สร้างส่วนติดต่อระหว่างคอมพิวเตอร์กับผู้ใช้หรือUser Interface ออกมาหลายโปรแกรมด้วยกัน และเป็นโปรแกรมที่มีลักษณะการเขียนโปรแกรม Object – Oriented Programming ซึ่งเราสามารถที่จะใช้ความสามารถของโปรแกรมเหล่านี้มาสร้างและพัฒนาโปรแกรมคอมพิวเตอร์ในรูปแบบของกราฟฟิกได้อีกด้วย

สำหรับในโครงงานนี้ได้เลือกใช้ Visual Basic 6.0 มาสร้างส่วนติดต่อระหว่างคอมพิวเตอร์กับผู้ใช้ เพื่อเป็นชุดควบคุมการทำงานของระบบควบคุมกล้องอัตโนมัติ

4.2 สาเหตุที่ต้องใช้ Visual Basic

สามารถใช้ Visual Basic สร้างโปรแกรมบนวินโดวส์โดยอาศัยการออกแบบโปรแกรมในลักษณะ Visualize ซึ่งใช้การกำหนดตำแหน่งของ Object ลงบนจอภาพเพื่อติดต่อกับผู้ใช้โดยตรง Object เหล่านี้จะเปลี่ยนไปตามเหตุการณ์ (event) ต่าง ๆ ที่เกิดขึ้น เช่น การเคลื่อนเมาส์ หรือการรับข้อมูลจากคีย์บอร์ด ในการกำหนดขั้นตอนการทำงานให้กับ Object ภายใต้ Event ใด ๆ จะใช้ภาษา Basic เข้ามาช่วยในการเขียนโปรแกรม ดังนั้นอาจกล่าวได้ว่า การพัฒนาโปรแกรมบนวินโดวส์โดยใช้ Visual Basic มีความง่ายและสะดวกในการใช้งาน รวมทั้งมีขั้นตอนน้อย เพียงแต่เลือก Form และ Control ที่เหมาะสม แล้ววาดลงบนจอภาพเพื่อใช้ติดต่อกับผู้ใช้ จากนั้นจึงทำการเขียนภาษา Basic เพื่อสร้างโปรแกรมด้วยตนเอง ด้วยวิธีที่ง่ายและรวดเร็วกว่าที่คิด จึงทำให้ผู้ใช้เรียนรู้ได้ภายใน 2 – 3 ชั่วโมง และสามารถสร้างโปรแกรมวินโดวส์เป็นโปรแกรมแรกได้

นอกจากนี้ Visual Basic ใช้ได้ตั้งแต่ User ระดับต้น เพื่อจะสร้างโปรแกรมง่าย ๆ บน วินโดวส์ หรือ Programmer ระดับกลางที่จะเรียกใช้ฟังก์ชันการทำงานต่าง ๆ ของ Visual Basic ได้อย่างมีประสิทธิภาพ ตลอดจน Programmer ระดับอาชีพที่จะพัฒนาโปรแกรมในระดับสูงโดยใช้ Object linking and Embedding (OLE) และ Windows Application Programmin Interface (API) มาประกอบในการเขียนโปรแกรม

4.3 ขั้นตอนในการพัฒนาโปรแกรมของ Visual Basic

ขั้นตอนในการพัฒนาโปรแกรม ประกอบด้วยขั้นตอนหลัก 2 ขั้นตอนดังนี้

ขั้นตอนที่ 1. สร้างจอภาพของโปรแกรม

ในขั้นตอนนี้จะทำการออกแบบ Form เพื่อใช้ในการติดต่อกับผู้ใช้ หรือที่เรียกว่า การออกแบบ "User Interface" ในการพัฒนาโปรแกรมแบบเดิม ขั้นตอนนี้จะใช้เวลาและค่าใช้จ่ายสูง เนื่องจากจะ

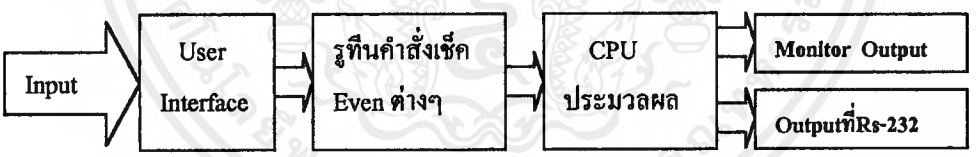
ต้องเขียนโปรแกรมเพื่อสร้างจอภาพต่าง ๆ จากนั้นต้อง Compile โปรแกรมนั้น แล้ว Run จึงจะเห็นจอภาพที่จัดทำขึ้น แต่สำหรับ Visual Basic ปัญหาในลักษณะนี้ได้ถูกแก้ไขโดยใช้เทคนิคของ Visualize ซึ่งเป็นความสามารถส่วนหนึ่งของ Visual Basic ขั้นตอนนี้สามารถทำได้อย่างง่ายดาย เพียงแต่นำเอาคอนโทรลต่าง ๆ ใน Toolbox ที่ต้องการใช้งานมาวางไว้บน Form ซึ่งทำให้ประหยัดเวลา และสามารถเห็นลักษณะจอภาพที่ออกแบบได้ในขณะนั้นเลย

ขั้นตอนที่ 2 เขียนโปรแกรม

เมื่อทำการวาง Control ต่าง ๆ บน Form เป็นที่เรียบร้อยแล้ว (Control ต่าง ๆ เมื่อถูกนำมาวางไว้บน Form จะเรียกว่า "Object") ขั้นตอนต่อมาก็คือ การเขียนโปรแกรมเพื่อกำหนดการทำงานให้กับแต่ละ Object ภายใต้เหตุการณ์ต่าง ๆ (Event) ที่จะเกิดขึ้นกับจอภาพนั้น ๆ

4.4 ความสามารถของโปรแกรมระบบควบคุมกล้องอัตโนมัติที่สร้างขึ้นในโครงการนี้

1. สามารถสั่ง Tracking ได้อัตโนมัติโดยใช้ Sensor
2. สามารถควบคุมการทำงานของระบบควบคุมกล้องอัตโนมัติด้วย User
3. สามารถที่จะปรับมุมในการหมุนของกล้องได้
4. สามารถที่จะส่งภาพคนที่เข้ามาในบริเวณที่ติดกล้องกลับมาที่ User ได้
5. สามารถที่จะสั่งให้ระบบเตือนภัยทำงาน
6. สามารถจำกัดผู้ที่จะเข้ามาใช้ระบบเตือนภัยได้



รูปที่ 4.1 ภาพแสดงการทำงานโดยรวมของระบบ

- ส่วน Input จะเป็นการ Click Mouse หรือการกด Key Broad
- ส่วน User Interface จะมี Option ต่างๆให้เลือกใช้โดยการ Click Mouse หรือการกด Key Broad
- รูทีนคำสั่งรับค่าจาก Even ต่างๆที่มาจากผู้ใช้และส่งให้ CPU ประมวลผล
- Output ที่ได้จะมี 2 ลักษณะคือ Output ที่แสดงผลทาง Monitor และ Output ที่ส่งไปที่ RS – 232 โดยจะมีการแสดงผลที่ตัว Hardware

4.5 การออกแบบโปรแกรม Interface เพื่อควบคุมระบบควบคุมกล้องอัตโนมัติ

โปรแกรม Interface ที่ออกแบบนี้ จะมีส่วนสำคัญอยู่ 2 ส่วนคือ

1. ส่วนหน้าต่างผู้ใช้และส่วนสั่งระบบเตือนภัยให้ทำงาน

2. ส่วนบังคับฐานกล้องโดยการการส่งค่าผ่านพอร์ต RS232 โดยใช้ MSComm Control

4.6 ส่วนหน้าต่างผู้ใช้และส่วนสั่งระบบเตือนภัยให้ทำงาน

ส่วนหน้าต่างผู้ใช้และส่วนสั่งระบบเตือนภัยให้ทำงานจะมีลักษณะการทำงานเป็น 2 Modes คือ Manual Mode และ Automatic Mode โดย Manual Mode นั้นผู้ใช้สามารถที่จะควบคุมการทำงานของระบบรักษาความปลอดภัยได้เองและสามารถที่จะใช้ระบบเตือนภัย โดยในการใช้ระบบเตือนภัยนั้นผู้ใช้จะต้องใส่รหัสผ่านทุกครั้ง ส่วน Automotic Mode นั้นระบบรักษาความปลอดภัยจะทำงานเองอัตโนมัติ Flow Chart ของการทำงานใน Automatic Mode และ Manual Modeสามารถดูได้ในภาคผนวก ก.

4.7 ส่วนบังคับฐานกล้องโดยการส่งค่าผ่านพอร์ต RS232 โดยใช้ MSComm Control

ในการจะบังคับให้ฐานกล้องซึ่งอยู่ในส่วนของ Hardware ให้ทำงานนั้นจะต้องใช้ Function คำสั่ง MSComm ที่มีอยู่ใน Visual Basic มาช่วย

MSComm เป็น Object Controller ที่ใช้ทางด้านการสื่อสารผ่าน Port Series ของ Computer แอปพลิเคชั่น MSComm ไปใช้นั้นมีอยู่หลายประเภทด้วยกัน เช่น การเขียนโปรแกรมสื่อสารกับคอมพิวเตอร์อื่น ๆ หรือศูนย์บริการ BBS หรือแม้กระทั่งโฮสต์ที่ให้บริการอินเทอร์เน็ตโดยผ่านทางโมเด็ม หรือนอกจากนี้ผู้อ่านยังสามารถที่จะใช้คอนโทรล MSComm ในการติดต่อหรือควบคุมบอร์ดต่าง ๆ หรือแม้กระทั่งเครื่องอ่านรหัสบาร์โค้ด (Barcode Reader) ที่ต่อผ่านพอร์ตอนุกรมก็ได้เช่นกัน

ด้วยเหตุนี้คอนโทรล MSComm จึงมีประโยชน์อย่างมากกับผู้อ่านที่ต้องการสร้างแอปพลิเคชั่นด้านการสื่อสารหรือการควบคุมบอร์ดต่าง ๆ ที่มีใช้ตามโรงงานอุตสาหกรรมทั่วไป ซึ่งในปัจจุบันได้มีการนำ Visual Basic มาประยุกต์ใช้งานด้านนี้กันอย่างแพร่หลาย เช่น การใช้ Visual Basic ร่วมกับคอนโทรล MS Comm ในการติดต่อกับเครื่องวัดการใช้กำลังไฟฟ้าของหม้อแปลงไฟฟ้าในโรงงานอุตสาหกรรม เป็นต้น

จะเห็นว่าเราสามารถที่จะนำ Visual Basic มาประยุกต์ใช้กับอุตสาหกรรมที่ต้องการควบคุมบอร์ดหรืออุปกรณ์ด้านอิเล็กทรอนิกส์ผ่านทางพอร์ตอนุกรมแบบอัตโนมัติ (realtime - automatic control) ได้โดยอาศัยเพียงคอนโทรล MSComm เท่านั้น ดังนั้นในบทนี้ผู้เขียนจึงขอเน้นเนื้อหาโดยละเอียดและสามารถที่จะนำไปใช้งานอย่างสมบูรณ์

คอนโทรล MSComm

คอนโทรล MSComm (Communications) เป็นคอนโทรลตัวหนึ่งที่ช่วยในการติดต่อกับพอร์ตอนุกรม (serial port) ซึ่งผู้อ่านสามารถทำการรับ - ส่งข้อมูลผ่านทางพอร์ตอนุกรมได้ด้วยคอนโทรลนี้ เช่น การติดต่อผ่านโมเด็ม (modem) หรือติดต่อโดยตรงกับบอร์ดอิเล็กทรอนิกส์ เป็นต้น ซึ่งคอนโทรล MSComm ที่มากับ Visual Basic จะเป็นคอนโทรลที่ทำงานโดยมีการตอบสนองต่อเหตุการณ์แบบ

Event - Driven นั่นก็คือคอนโทรลจะทำหน้าที่ตรวจสอบการเกิดขึ้นหรือการร้องขอให้เกิดเหตุการณ์ต่าง ๆ กับพอร์ตอนุกรมโดยอัตโนมัติ และจะมีการแจ้งเตือนให้ผู้อ่านได้ทราบโดยผ่านทางโพรซีเยอร์เหตุการณ์ เช่นเดียวกับคอนโทรลทั่วไปของ Visual Basic นั่นเอง ดังนั้นในการเขียนโค้ดผู้อ่านจึงไม่จำเป็นต้องสร้างโพรซีเยอร์ที่ทำหน้าที่คอยตรวจสอบเหตุการณ์ต่าง ๆ ของพอร์ตอนุกรม ซึ่งจำทำให้ง่ายต่อการใช้งานเป็นอย่างมาก

คอนโทรล MSComm จะมีหน้าที่มาตรฐานหลัก ๆ สำหรับการสื่อสารผ่านพอร์ตอนุกรม 3 ประการ ดังต่อไปนี้

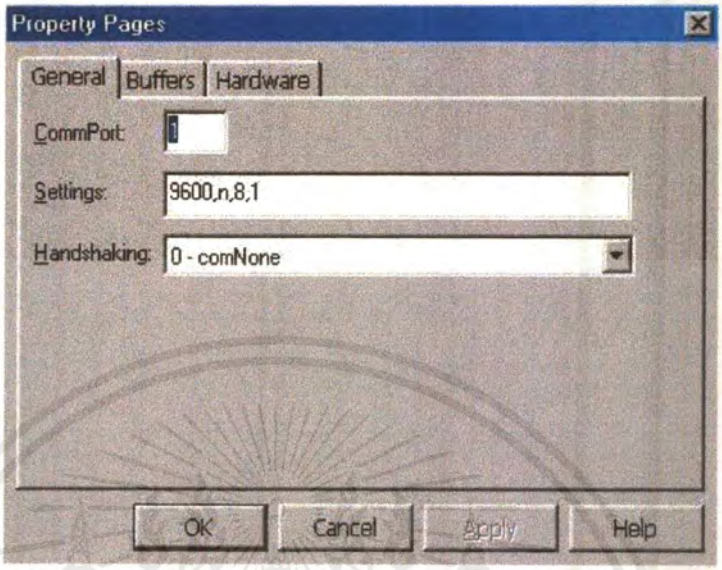
- หมุนหมายเลขติดต่อกับโทรศัพท์ปลายทางที่กำหนด
- ตรวจสอบการเข้ามาของข้อมูลยังพอร์ตอนุกรมโดยอัตโนมัติ
- ส่งข้อมูลตามที่กำหนดจากโปรแกรมไปยังพอร์ตอนุกรม

ในความเป็นจริงคอนโทรล MSComm ไม่ได้ทำหน้าที่ติดต่อกับพอร์ตอนุกรมโดยตรง แต่มันจะทำหน้าที่เรียกใช้ฟังก์ชันวินโดว์ API ซึ่งวินโดวส์จะทำการส่งหรือรับข้อมูลผ่านทางพอร์ตอนุกรมโดยอาศัยไดรเวอร์ Comm.drv อีกทอดหนึ่ง ดังนั้นจึงสามารถสรุปสั้น ๆ ได้ว่าทุกครั้งที่ผู้อ่านมีการเรียกใช้คอนโทรล MSComm ก็หมายถึงเรียกใช้ฟังก์ชันวินโดวส์ API ซึ่งจะถูกตีความอีกทอดหนึ่งโดยไดรเวอร์ Comm.drv จากนั้นก็จะส่งผ่านข้อมูลที่ถูกจัดรูปแบบตามมาตรฐานการสื่อสาร (ทั้งนี้ก็ขึ้นกับอุปกรณ์ที่ต่อเข้ากับพอร์ตอนุกรม) ให้กับดีไวซ์ไดรเวอร์อีกทอดหนึ่งนั่นเอง

การกำหนดคุณสมบัติของคอนโทรล MSComm ในขณะออกแบบ ผู้อ่านสามารถกระทำได้อย่างสะดวกโดยการคลิกที่ปุ่ม ของรายการ (Custom) ในหน้าต่างคุณสมบัติ ซึ่งก็จะปรากฏไดอะล็อกบ็อกซ์ Property Pages เพื่อให้ผู้อ่านได้ปรับแต่งค่าของคุณสมบัติของที่คอนโทรล MSComm สนับสนุน ดังในรูปที่ 4.2 ซึ่งผู้เขียนจะได้อธิบายรายละเอียดของแต่ละแท็ปต่อไป ส่วนปุ่มคำสั่งต่าง ๆ ของไดอะล็อกบ็อกซ์ Property Pages มีความหมายดังนี้

- ปุ่มคำสั่ง OK	ยอมรับการแก้ไขคุณสมบัติของคอนโทรล MSComm
- ปุ่มคำสั่ง Cancel	ยกเลิกการแก้ไขคุณสมบัติของคอนโทรล MSComm
- ปุ่มคำสั่ง Apply	อัปเดตคุณสมบัติที่ถูกแก้ไขของคอนโทรล MSComm
- ปุ่มคำสั่ง Help	แสดงผล Help ของคอนโทรล MSComm
-CommPort	หมายเลขของพอร์ตอนุกรม

- Settings พารามิเตอร์สำหรับการสื่อสาร เช่น baud rate หรือ parity bit เป็นต้น



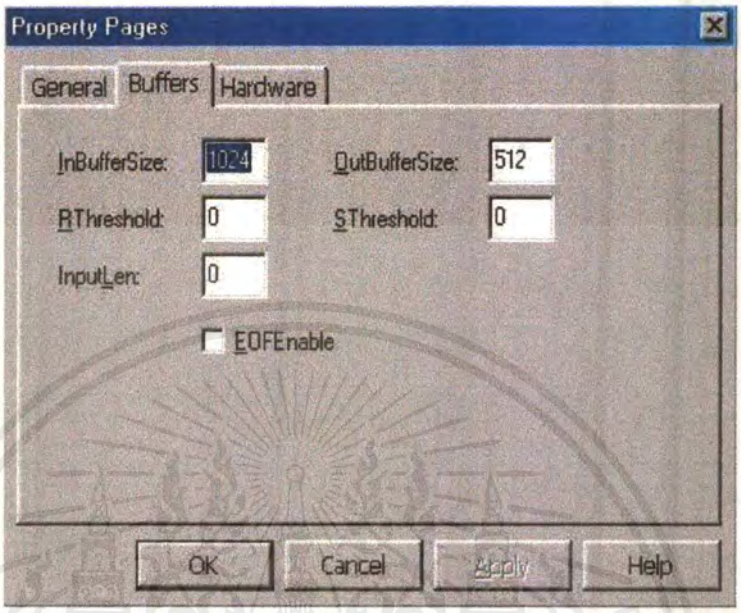
รูปที่ 4.2 แสดงแท็บ Generalในไดอะล็อกบ็อกซ์ Property PagesของคอนโทรลMSComm

- Handshaking การตรวจสอบการตอบรับการสื่อสาร (handshaking)

แท็บ **Buffets** การกำหนดคุณสมบัติเกี่ยวกับบัฟเฟอร์ข้อมูล

- InBufferSize ขนาดของบัฟเฟอร์สำหรับด้านรับเข้าข้อมูล
- Rthreshold จำนวนตัวอักษรที่จะรับเข้า ก่อนที่คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEvReceive และมีการเรียกเหตุการณ์ OnComm
- InputLen จำนวนตัวอักษรที่คุณสมบัติ Input จะอ่านข้อมูลจากบัฟเฟอร์ด้านรับเข้า
- OutBufferSize ขนาดของบัฟเฟอร์ด้านการส่งออกข้อมูล
- Sthreshold จำนวนตัวอักษรที่น้อยที่สุดที่ถูกจัดเก็บในบัฟเฟอร์ด้านส่งออก ก่อนที่คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEvSend และมีการเรียกเหตุการณ์ OnComm

- EOFEnable กำหนดให้คอนโทรล MSComm มีการตรวจสอบตัวอักษรรหัสจุดสิ้นสุดของไฟล์ (EOF) ในระหว่างการรับเข้าของข้อมูล



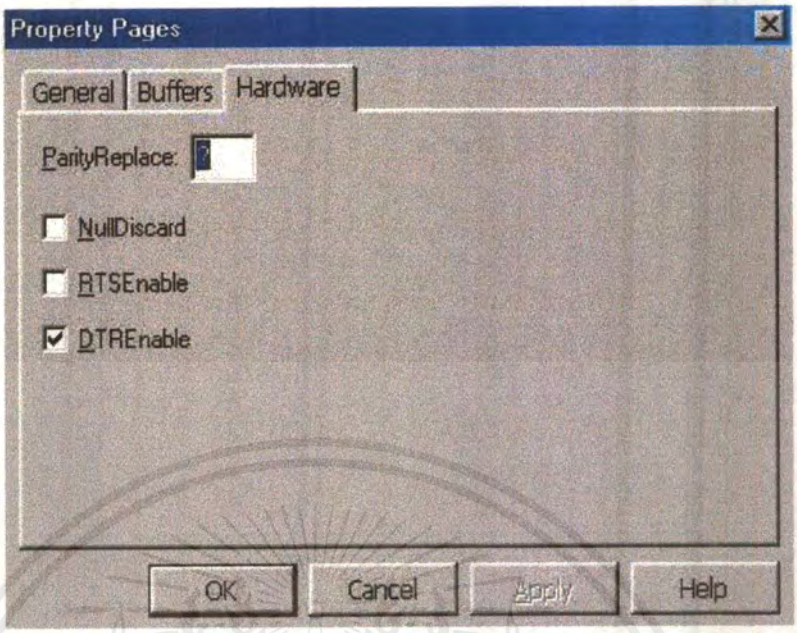
รูปที่ 4.3 แสดงแท็บ Buffers ในไดอะล็อกบ็อกซ์ Property Pages ของคอนโทรล MSComm

แท็บ **Hardware** การกำหนดคุณสมบัติฮาร์ดแวร์ (โมเด็ม)

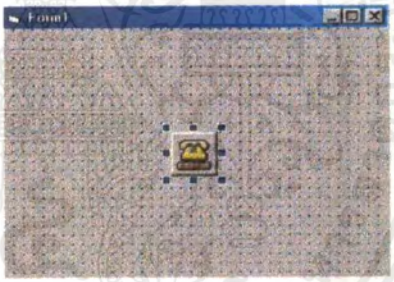
- ParityReplace กำหนดตัวอักษรสำหรับแทนที่ตัวอักษรที่ไม่เป็นจริง ในขณะที่เกิดข้อผิดพลาด parity error
- NullDiscard ตรวจสอบการส่งตัวอักษร Null จากพอร์ตอนุกรามไปยังบัฟเฟอร์ด้านรับเข้า
- RTSEnable มีการใช้งานสาย Request To Send (RTS)
- DTREnable มีการใช้งานสาย Data Terminal Ready (DTR)

สำหรับรายละเอียดทั้งหมดของคุณสมบัติที่สามารถกำหนดในไดอะล็อกบ็อกซ์ Property Pages ของคอนโทรล MSComm ตามที่กล่าวมาแล้วในข้างต้น ผู้อ่านสามารถที่จะดูรายละเอียดได้จากแต่ละคุณสมบัติดังที่จะกล่าวต่อไป

สำหรับฟอร์มหนึ่ง ๆ ผู้อ่านสามารถเพิ่มได้หลาย ๆ คอนโทรล MSComm ทั้งนี้ก็ขึ้นความต้องการของผู้อ่านในการติดต่อกับพอร์ตอนุกรมใดบ้าง สำหรับวินโดวส์ 95 และ NT 4.0 ผู้อ่านสามารถที่จะติดตั้งพอร์ตอนุกรมได้มากกว่า 4 พอร์ต โดยเมื่อผู้อ่านเพิ่มคอนโทรล MSComm ลงในฟอร์ม ก็จะปรากฏดังในรูปที่ 4.5 ซึ่งจะสนับสนุนคุณสมบัติ และโพรซีเยอร์เหตุการณ์ ดังต่อไปนี้



รูปที่ 4.4 แสดงแท็บ Hardwareในไดอะล็อกบ็อกซ์ Property Pages ของคอนโทรลMSComm



รูปที่ 4.5 แสดงคอนโทรล MSComm ในขณะออกแบบ

คุณสมบัติ

Break	CDHolding	CommEvent	CommID
CommPort	CTSHolding	DSRHolding	DTREnable
EOFEnable	Handshaking	InBufferCount	InBufferSize
Index	Input	InputLen	InputMode
Name	NullDiscard	Object	OutBufferCount
OutBufferSize	Output	Parent	ParityReplace
PortOpen	RTHreshold	RTSEnable	Settings
Sthreshold	Tag		

Break

กำหนดหรือยกเลิกสัญญาณการหยุด (Break signal) ซึ่งผู้อ่านสามารถกำหนดได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.Break [= boolean]

boolean หมายถึง ข้อมูลชนิดบูลีน ที่กำหนดสถานะของสัญญาณการหยุด ดังต่อไปนี้

True หมายถึง กำหนดสถานะของสัญญาณการหยุด

False หมายถึง ยกเลิกสถานะของสัญญาณการหยุด

การกำหนดสถานะของสัญญาณการหยุด หมายถึง การกำหนดให้สายสัญญาณ transmission อยู่ในสถานะของการหยุด ซึ่งสัญญาณการหยุดจะทำหน้าที่หยุดการรับเข้าตัวอักษรชั่วคราว จนกว่าสถานะดังกล่าวจะถูกยกเลิก โดยการกำหนดให้คุณสมบัติ Break มีค่าเท่ากับ False อีกครั้ง ซึ่งโดยปกติเราจะกำหนดสถานะของสัญญาณการหยุด ก็ต่อเมื่ออุปกรณ์ที่เราทำการติดต่อด้วยนั้น ต้องการให้มีการกำหนดสถานะของสัญญาณการหยุดเท่านั้น

CDHolding

ตรวจสอบสัญญาณพาหะ (carrier signal) โดยการค้นหาจากสถานะของสายสัญญาณ Carrier Detect (CD) ซึ่งผู้อ่านสามารถกำหนดได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.CDHolding

สัญญาณพาหะจะถูกส่งจากโมเด็มมายังพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ เพื่อแสดงสถานะความพร้อมที่จะติดต่อสื่อสาร (online) ของโมเด็ม ซึ่งค่าที่ได้จากคุณสมบัติ CDHolding จะเป็นค่าบูลีนดังต่อไปนี้

True หมายถึง สายสัญญาณ Carrier Detect อยู่ในสถานะ high (ไม่พร้อมที่จะติดต่อสื่อสารในขณะนั้น)

Fasle หมายถึง สายสัญญาณ Carrier Detect อยู่ในสถานะ low (พร้อมที่จะติดต่อสื่อสารในขณะนั้น)

ถ้าหากสายสัญญาณ Carrier Detect (ในบางครั้งวงการสื่อสารด้วยโมเด็มมักจะเรียกว่า Receive Line Signal Detect, RLSD ก็ได้) อยู่ในสถานะ high ก็แสดงว่าเกิดสถานะ time out ในขณะนั้น ซึ่งในกรณีนี้คอนโทรล MSComm ก็จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEventCDTO และเรียกโพรซีเยอร์เหตุการณ์ OnComm ทันที เพื่อแสดงให้เห็นถึงการเกิดข้อผิดพลาด

ถ้าหากผู้อ่านเขียนโปรแกรมที่มีการติดต่อกับโฮสต์ต่าง ๆ เช่น BBS หรืออินเทอร์เน็ต เป็นต้น ผู้อ่านควรที่จะเขียนโค้ดเพื่อตรวจสอบสถานะของสัญญาณพาหะเป็นช่วง ๆ ทั้งนี้เพราะโฮสต์ปลายทางอาจมีการยกเลิกการสื่อสารโดยการวางสายของโมเด็มได้ตลอดเวลา เพราะโดยปกติโฮสต์ปลายทางเหล่านี้จะมีการติดต่อกับผู้ใช้จำนวนมากในคราวเดียวกัน

CommEvent

รายงานเหตุการณ์ทุกครั้งที่เกิดข้อผิดพลาดหรือมีการสื่อสาร ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.CommEvent

คอนโทรล MSComm จะมีการเรียกโพรซีเยอร์เหตุการณ์ OnComm ทุกครั้งที่เกิดข้อผิดพลาดหรือมีการสื่อสารเกิดขึ้น ซึ่งค่าตัวเลขที่จำนวนเต็ม แสดงถึงข้อผิดพลาดหรือเหตุการณ์ที่มีการสื่อสารดังกล่าว ก็จะถูกจัดเก็บเอาไว้ในคุณสมบัติ CommEvent เสมอ ดังนั้นถ้าหากผู้อ่านต้องการตรวจสอบข้อผิดพลาดหรือเหตุการณ์ที่มีการสื่อสารภายในโพรซีเยอร์เหตุการณ์ OnComm ก็ควรที่จะใช้ค่าตัวเลขจากคุณสมบัติ CommEvent ในการตรวจสอบเสมอ ซึ่งค่าตัวเลขที่รายงานโดยคุณสมบัติ CommEvent มีดังต่อไปนี้

ข้อผิดพลาด

ค่าคงที่	ค่าตัวเลข	รายละเอียด
comEventBreak	1001	ได้รับสัญญาณการหยุด (Break signal)
comEventCTSTO	1002	สายสัญญาณ Clear To Send อยู่ในสถานะ low (timeout) ในขณะที่พยายามจะส่งออกตัวอักษร
comEventDSRTO	1003	สายสัญญาณ Data Set Ready จะอยู่ในสถานะ low (timeout) ในขณะที่พยายามจะส่งออกตัวอักษร
ComEventFrame	1004	เฟรมของข้อมูลไม่ถูกต้อง ซึ่งถูกตรวจพบโดยฮาร์ดแวร์
ComEventOverun	1006	เกิด Potr Overun หมายถึง มีการรับตัวอักษรตัวใหม่เข้ามาในขณะที่ตัวอักษรก่อนหน้านี้ยังไม่ถูกอ่านจากฮาร์ดแวร์ ดังนั้นจึงเกิดการสูญหายของตัวอักษร
ComEventCDTO	1007	สายสัญญาณ Carrier Detect อยู่ในสถานะ low (timeout) ในขณะที่พยายามจะส่งออกตัวอักษร
ComEventRxOver	1008	Receive Buffer Overflow หมายถึง ขนาดของบัฟเฟอร์ด้านรับเข้าข้อมูล (receive buffer) ไม่เพียงพอกับขนาดของข้อมูลที่รับเข้ามา
ComEventRxParity	1009	Parity Error ซึ่งถูกตรวจพบโดยฮาร์ดแวร์

ComEventTxFull 1010 Transmit Buffer Full หมายถึง บัฟเฟอร์ด้านส่งออกข้อมูล (transmit Buffer) เต็ม ในขณะที่พยายามจัดเก็บข้อมูลใหม่ลงในบัฟเฟอร์

ComEventDCB 1011 Unexpected error หมายถึง เกิดข้อผิดพลาดที่ไม่ได้ถูกนิยามเอาไว้ในขณะอ่าน Device Control Block (DCB) จากพอร์ตอนุกรม

เหตุการณ์ (Event)

ค่าคงที่	ค่าตัวเลข	รายละเอียด
comEvSend	1	มีจำนวนตัวอักษรในบัฟเฟอร์ด้านส่งออกข้อมูล น้อยกว่าจำนวนตัวอักษรที่กำหนดในคุณสมบัติ Sthreshold
ComEvReceive	2	การรับเข้าจำนวนตัวอักษรที่ถูกกำหนดในคุณสมบัติ Rthreshold ซึ่งจะเกิดขึ้นอย่างต่อเนื่องจนกว่าผู้อ่านจะใช้คุณสมบัติ Input ในการอ่านข้อมูลจากบัฟเฟอร์สำหรับรับเข้าข้อมูล
ComEvCTS	3	มีการเปลี่ยนแปลงสถานะของสายสัญญาณ Clear To Send
ComEvDSR	4	มีการเปลี่ยนสถานะของสายสัญญาณ Data Set Ready ซึ่งเหตุการณ์นี้จะเกิดขึ้นเมื่อมีการเปลี่ยนสถานะของสายสัญญาณ Data Set Ready จาก 1 เป็น 0 เท่านั้น
ComEvCD	5	มีการเปลี่ยนสถานะของสายสัญญาณ Carrier Detect
ComEvRing	6	มีการตรวจพบการเรียกหมายเลข (สัญญาณเสียงกริ่ง) ซึ่ง URAT บางตัวอาจจะไม่สนับสนุนคุณสมบัตินี้
ComEvEOF	7	มีการรับตัวอักษรรหัสจุดสิ้นสุดของไฟล์ (EOF, ASCII 26)

CommID

รายงานหมายเลข handle ของดีไวซ์ด้านการสื่อสาร ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.CommID

หมายเลข handle ของดีไซ์ด้านการสื่อสารที่ได้จากคุณสมบัติ CommID จะเป็นค่าตัวเลขจำนวนเต็ม long สำหรับนำไปใช้งานร่วมกับฟังก์ชันวินโดวส์ API อื่น ๆ

CommPort

รายงานหรือกำหนดหมายเลขของพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ที่ต้องการติดต่อ

รูปแบบการใช้งาน

object.CommPort [= value]

value หมายถึง ข้อมูลชนิดจำนวนเต็ม ที่กำหนดหมายเลขของพอร์ตอนุกรม

สำหรับหมายเลขของพอร์ตอนุกรมสามารถมีค่าได้ตั้งแต่ 1 ถึง 16 (ค่าปกติจะเท่ากับ 1) ซึ่งก่อนที่ผู้อ่านจะเปิดพอร์ตด้วยคุณสมบัติ PortOpen ผู้อ่านต้องกำหนดหมายเลขของพอร์ตอนุกรมให้กับคุณสมบัติ CommPort เสียก่อน โดยถ้าหากหมายเลขของพอร์ตอนุกรมที่กำหนดให้คุณสมบัติ CommPort ไม่เป็นความจริง ก็จะเกิดข้อผิดพลาด 68 (Device unavailable) ทันที ซึ่งในกรณีนี้ผู้อ่านสามารถแก้ไขได้โดยการกำหนดหมายเลขของพอร์ตอนุกรมที่ถูกต้องเสียใหม่ แล้วจึงทำการเปิดพอร์ตอนุกรมอีกครั้งด้วยคุณสมบิต PortOpen

CTSHolding

ตรวจสอบสถานะของสายสัญญาณ Clear To Send (CTS) ก่อนที่จะส่งข้อมูลไปยังบัฟเฟอร์ของโมเด็ม ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.CTSHolding

ซึ่งค่าที่ได้จากคุณสมบัติ CTSHolding จะเป็นค่าบูลีนดังต่อไปนี้

True หมายถึง สายสัญญาณ Clear To Send อยู่ในสถานะ high

False หมายถึง สายสัญญาณ Clear To Send อยู่ในสถานะ low

โดยปกติสัญญาณ Clear To Send จะถูกส่งจากโมเด็มมายังพอร์ตอนุกรมของคอมพิวเตอร์ เพื่อแสดงให้เห็นถึงสถานะความพร้อมในการส่งข้อมูลมายังโมเด็ม ซึ่งถ้าหากสายสัญญาณ Clear To Send อยู่ในสถานะ low ก็แสดงว่าเกิดสถานะ time out ในขณะนั้น ซึ่งในกรณีนี้คอนโทรล MSComm ก็จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEventCTSTO และเรียกโพรซีเยอร์เหตุการณ์ OnComm ทันทีเพื่อแสดงให้เห็นถึงการเกิดข้อผิดพลาด

สายสัญญาณ Clear To Send จะถูกในการขอติดต่อสื่อสารของฮาร์ดแวร์ (hardware handshaking) โดยการใช้วิธี RTS/CTS (Request To Send/Clear To Send)

DSRHolding

ตรวจสอบสถานะของสายสัญญาณ Data Set Ready (DSR) ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.DSRHolding

ซึ่งค่าที่ได้จากคุณสมบัติ DSRHolding จะเป็นค่าบูลีนดังต่อไปนี้

True หมายถึง สายสัญญาณ Data Set Ready อยู่ในสถานะ high

False หมายถึง สายสัญญาณ Data Set Ready อยู่ในสถานะ low

โดยปกติสัญญาณ Data Set Ready จะถูกส่งจากโมเด็มมายังพอร์ตอนุกรมของคอมพิวเตอร์ เพื่อแสดงให้เห็นถึงสถานะความพร้อมในการทำงานของโมเโม ซึ่งถ้าหากสายสัญญาณ Data Set Ready อยู่ในสถานะ high ก็แสดงว่าเกิดสถานะ time out ในขณะนั้น ซึ่งในกรณีนี้คอนโทรล MSComm ก็จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEventDSRTO และเรียกโพรซีเยอร์เหตุการณ์ OnComm ทันทีเพื่อแสดงให้เห็นถึงการเกิดข้อผิดพลาด

โดยปกติคุณสมบัติ DSRHolding จะถูกใช้ในการเขียนโพรซีเยอร์สำหรับการขอติดต่อสื่อสาร (handshaking) ด้วยวิธี Data Set Ready/Data Terminal Ready สำหรับติดต่อสื่อสารกับเครื่องจักรหรืออุปกรณ์ประเภท Data Terminal Equipment (DTE)

DTREnable

ตรวจสอบหรือกำหนดสถานะของสายสัญญาณ Data Terminal Ready (DTR) ในระหว่างที่มีการสื่อสารข้อมูล ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.DTREnable [= boolean]

boolean หมายถึง ข้อมูลชนิดบูลีน ที่บอกถึงสถานะของสายสัญญาณ Data Terminal Ready ต่อไปนี้

True หมายถึง ยอมให้มีการใช้งานสายสัญญาณ Data Terminal Ready

False หมายถึง ยกเลิกการใช้งานสายสัญญาณ Data Terminal Ready (default)

โดยปกติสัญญาณ Data Terminal Ready จะถูกส่งจากคอมพิวเตอร์มายังโมเด็ม เพื่อแสดงให้เห็นถึงสถานะความพร้อมในการรับข้อมูลจากโมเด็ม โดยถ้าหาก DTR ถูกกำหนดให้เท่ากับ True สายสัญญาณ Data Terminal Ready อยู่ในสถานะ high (on) เมื่อพอร์ตอนุกรมถูกเปิด และ (off) เมื่อพอร์ตอนุกรมถูกปิดแต่ถ้าหาก DTR ถูกกำหนดให้เท่ากับ False สายสัญญาณ DTR อยู่ในสถานะ low (off) ตลอดเวลา ในการสื่อสารผ่านโมเด็มโดยทั่วไปสัญญาณ DTR อยู่ในสถานะ low จะหมายถึงการวางหูโทรศัพท์หรือสิ้นสุดการสื่อสารนั่นเอง

EOFEnable

กำหนดให้คอนโทรล MSComm มีการตรวจสอบตัวอักษรรหัสจุดสิ้นสุดของไฟล์ (EOF) ในระหว่างการรับเข้าของข้อมูล โดยที่การรับเข้าข้อมูลจะสิ้นสุดที่พบตัวอักษร EOF

รูปแบบการใช้งาน

object.EOFEnable [= boolean]

boolean หมายถึง ข้อมูลชนิดบูลีน ที่บอกถึงสถานะของการตรวจสอบตัวอักษร EOF ต่อไปนี้

True หมายถึง เหตุการณ์ OnComm จะถูกเรียกทันทีที่พบตัวอักษร EOF

False หมายถึง เหตุการณ์ On Comm จะไม่ถูกเรียกเมื่อพบตัวอักษร EOF (default)

โดยถ้าหากผู้อ่านกำหนดให้คุณสมบัติ EOFEnable มีค่าเท่ากับ True เมื่อเหตุการณ์ OnComm จะถูกเรียกทันทีที่พบตัวอักษร EOF คุณสมบัติ CommEvent ก็จะถูกกำหนดให้มีค่าเท่ากับ comEvEOF ทันที

Handshaking

รายงานหรือกำหนดการใช้โปรโตคอลการตอบรับการติดต่อสื่อสารฮาร์ดแวร์ (hardware handshaking protocol)

รูปแบบการใช้งาน object.Handshaking [= value]

value หมายถึงข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดโปรโตคอลการตอบรับการติดต่อสื่อสาร ดังต่อไปนี้

ค่าคงที่	ค่าตัวเลข	รายละเอียด
comNone	0	ไม่มีการตอบรับ การติดต่อสื่อสาร (default)
comXOnXOff	1	การตอบรับการติดต่อสื่อสารแบบ XON/XOFF
comRTS	2	การตอบรับการติดต่อสื่อสารแบบ RTS/CTS (Request To Send/Clear To Send)
comTRSXOnXOff	3	การตอบรับการติดต่อสื่อสารแบบทั้ง Request To Send และ XON/XOFF

ในด้านการสื่อสารด้วยโมเด็ม Handshaking หมายถึง โปรโตคอลการสื่อสารภายในที่ซอฟต์แวร์ใช้ในการส่งข้อมูลจากพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ไปยังบัฟเฟอร์ด้านรับเข้าข้อมูล โดยทุกครั้งที่มีการรับข้อมูลมายังพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ มันก็จะถูกจัดส่งต่อมายังบัฟเฟอร์ด้านรับเข้าข้อมูลทันที ทั้งนี้เพื่อให้ซอฟต์แวร์สามารถอ่านข้อมูลดังกล่าวได้ต่อไป ซึ่งโปรโตคอลการสื่อสารภายในดังกล่าวจะเป็นการช่วยในการตรวจสอบเพื่อป้องกันการสูญหายของข้อมูล เมื่อเกิดปัญหาขนาดของบัฟเฟอร์ไม่เพียงพอกับจำนวนข้อมูลที่ส่งมา (buffer overun)

InBufferCount

รายงานจำนวนของตัวอักษรที่รออยู่ในบัฟเฟอร์ด้านรับเข้า ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะวันแอปพลิเคชันเท่านั้น

รูปแบบการใช้งาน

object.InBufferCount [= value]

value หมายถึง ข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดจำนวนของตัวอักษรที่รออยู่ในบัฟเฟอร์ด้านรับเข้าซึ่งจะเป็นจำนวนของตัวอักษรที่ซอฟต์แวร์จะสามารถอ่านได้

ถ้าหากมีการลบตัวอักษรทั้งหมดที่รออยู่ในบัฟเฟอร์ด้านรับเข้าทิ้ง ค่าของคุณสมบัติ InBufferCount ก็จะเท่ากับ 0

InBufferSize

รายงานหรือกำหนดขนาดของบัฟเฟอร์ด้านรับเข้า ซึ่งมีหน่วยเป็นไบต์ (โดยปกติ 1 ไบต์จะเท่ากับ 1 ตัวอักษร)

รูปแบบการใช้งาน

object.InBufferSize [= value]

value หมายถึง ข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดขนาดของบัฟเฟอร์ด้านรับเข้า ซึ่งปกติซอฟต์แวร์จะกำหนดให้มีค่าเท่ากับ 1,024 ไบต์ (1 KB)

ในการเลือกขนาดของบัฟเฟอร์ด้านรับเข้าที่เหมาะสมนั้น ในทางปฏิบัติเป็นสิ่งที่ยากมาก ทั้งนี้ก็ขึ้นอยู่กับความหนาแน่นของข้อมูลที่มีการส่งไป – มา ในแต่ละครั้งและความเร็วของการสื่อสาร (transmision rate) ของโมเด็ม ซึ่งโดยปกติโปรแกรมเมอร์โดยทั่วไปจะกำหนดให้มีค่าเท่ากับ 1,024 ไบต์ (1 KB) โดยถ้าหากเกิดข้อผิดพลาด overflow ในขณะรันแอปพลิเคชัน ก็ให้ทำการเพิ่มขนาดของบัฟเฟอร์ด้านรับเข้าต่อไป

Input

รายงานพร้อมทั้งทำการลบข้อมูลในบัฟเฟอร์ด้านรับเข้าทิ้ง ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.Input

ทุกครั้งที่มีการใช้คุณสมบัติ Input ในการอ่านข้อมูลจากบัฟเฟอร์ด้านรับเข้านั้น จำนวนของตัวอักษรที่อ่านได้จะถูกกำหนดลงในคุณสมบัติ InputLen ทันที ซึ่งถ้าหากผู้อ่านกำหนดให้คุณสมบัติ InputLen มีค่าเท่ากับ 0 ก็จะหมายถึงการกำหนดให้คุณสมบัติ Input อ่านข้อมูลทั้งหมดจากบัฟเฟอร์ด้านรับเข้านั่นเอง ซึ่งชนิดของข้อมูลที่อ่านโดยคุณสมบัติ Input จะเป็นข้อมูลแบบข้อความหรือไบนารีก็ขึ้นกับการกำหนดค่าของคุณสมบัติ InputMode ดังจะได้กล่าวรายละเอียดต่อไป

InputLen

รายงานหรือกำหนดจำนวนของตัวอักษรที่อ่านโดยคุณสมบัติ Input จากบัฟเฟอร์ด้านรับเข้าซึ่งมีหน่วยเป็นไบต์

รูปแบบการใช้งาน

object.InputLen [= value]

value หมายถึงข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดจำนวนของตัวอักษรที่อ่านโดยคุณสมบัติ Input

โดยปกติผู้เขียนมักจะกำหนดให้คุณสมบัติ InputLen มีค่าเท่ากับ 0 เพื่อให้คุณสมบัติ Input มีการอ่านตัวอักษรทั้งหมดจากบัฟเฟอร์ด้านรับเข้าทั้งหมด แต่ถ้าหากผู้เขียนติดต่อกับบอร์ดควบคุมต่าง ๆ (single control board) ก็จะกำหนดค่าคุณสมบัติ InputLen ให้เท่ากับขนาดของเฟรมข้อมูลที่จะมีการรับ – ส่ง ในแต่ละครั้ง ซึ่งจะมีค่าคงที่

InputMode

รายงานหรือกำหนดชนิดของข้อมูลที่จะถูกอ่านโดยคุณสมบัติ Input จากบัฟเฟอร์ด้านรับเข้า

รูปแบบการใช้งาน

object.InputMode [= value]

value หมายถึงข้อมูลชนิดตัวเลขจำนวนเต็ม ที่กำหนดจำนวนชนิดของข้อมูลที่อ่านโดยคุณสมบัติ Input ดังนี้

ค่าคงที่	ค่าตัวเลข	รายละเอียด
comInputModeText	0	ข้อมูลชนิดข้อความทั่วไป (default)
comInputModeBinary	1	ข้อมูลชนิดไบบารี

การกำหนดชนิดของข้อมูลที่จะถูกอ่านโดยคุณสมบัติ Input ก็ขึ้นกับลักษณะของงานที่ผู้อ่านกำลังควบคุม โดยปกติถ้าหากข้อมูลชนิดตัวอักษร ANSI ทั่วไป ก็จะกำหนดให้เป็นข้อมูลชนิดข้อความ (comInputModeText) แต่ถ้าหากข้อมูลประกอบด้วยตัวอักษรควบคุม (แอสกีตั้งแต่ 0 ถึง 31) ก็จะกำหนดให้เป็นข้อมูลชนิดไบบารี (comInputModeBinary)

NullDiscard

รายงานหรือกำหนดตรวจสอบการส่งตัวอักษร Null (แอสกี 0) จากพอร์ตอนุกรมไปยังบัฟเฟอร์ด้านรับเข้า

รูปแบบการใช้งาน

object.NullDiscard [= boolean]

boolean หมายถึง ข้อมูลชนิดบูลีน ที่กำหนดตรวจสอบการส่งตัวอักษร Null จากพอร์ตอนุกรมไปยังบัฟเฟอร์ด้านรับเข้า ต่อไปนี้

True หมายถึง ไม่มีการส่งตัวอักษร Null จากพอร์ตอนุกรมไปยังบัฟเฟอร์ด้านรับเข้า

False หมายถึง มีการส่งตัวอักษร Null จากพอร์ตอนุกรมไปยังบัฟเฟอร์ด้านรับเข้า (default)

OutBufferCount

รายงานจำนวนของตัวอักษรที่รออยู่ในบัฟเฟอร์ด้านส่งออก ซึ่งผู้อ่านสามารถอ่านค่าได้เฉพาะในขณะรันแอปพลิเคชั่นเท่านั้น

รูปแบบการใช้งาน

object.OutBufferCount [= value]

value หมายถึง ข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดจำนวนของตัวอักษรที่รออยู่ในบัฟเฟอร์ด้านส่งออก

ถ้าหากมีการลบตัวอักษรทั้งหมดที่รออยู่ในบัฟเฟอร์ด้านส่งออกทิ้ง ค่าของคุณสมบัติ OutBufferCount ก็จะเท่ากับ 0

OutBufferSize

รายงานหรือกำหนดขนาดของบัฟเฟอร์ด้านส่งออก ซึ่งมีหน่วยเป็นไบต์ (โดยปกติ 1 ไบต์จะเท่ากับ 1 ตัวอักษร)

รูปแบบการใช้งาน

object.OutBufferSize [= value]

value หมายถึง ข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดขนาดของบัฟเฟอร์ด้านส่งออก ซึ่งปกติซอฟต์แวร์จะกำหนดให้มีค่าเท่ากับ 512 ไบต์

ในการเลือกขนาดของบัฟเฟอร์ด้านส่งออกที่เหมาะสมนั้น ในทางปฏิบัติเป็นสิ่งที่ยากมากเช่นกัน ทั้งนี้ก็ขึ้นอยู่กับความหนาแน่นของข้อมูลที่มีการส่งไป – มา ในแต่ละครั้งและความเร็วของการสื่อสารของโมเด็ม ซึ่งโดยปกติโปรแกรมเมอร์โดยทั่วไปจะกำหนดให้มีค่าเท่ากับ 512 ไบต์ โดยถ้าหากเกิดข้อผิดพลาด overflow ในขณะรันแอปพลิเคชัน ก็ให้ทำการเพิ่มขนาดของบัฟเฟอร์ด้านรับเข้าต่อไป

Output

ทำการส่งข้อมูลไปยังบัฟเฟอร์ด้านส่งออก ซึ่งผู้อ่านสามารถกำหนดค่าได้เฉพาะในขณะทำงานเท่านั้น

รูปแบบการใช้งาน

object.Output [= value]

value หมายถึง ข้อมูลชนิดสตริงหรือชนิด variant ที่ต้องการส่งไปยับบัฟเฟอร์ด้านส่งออก

สำหรับชนิดของข้อมูลที่ถูกส่งโดยคุณสมบัติ Output จะเป็นข้อมูลแบบข้อความหรือไบนารี ก็ขึ้นกับการกำหนดค่าของคุณสมบัติ InputMode ดังที่กล่าวมาแล้วข้างต้น

ParityReplace

กำหนดตัวอักษรสำหรับแทนที่ตัวอักษรที่ไม่เป็นจริง ในขณะที่เกิดข้อผิดพลาด parity error

รูปแบบการใช้งาน

object.ParityReplace [= value]

value หมายถึง ข้อมูลชนิดสตริง ที่กำหนดตัวอักษรใด ๆ

โดยปกติผู้เขียนจะกำหนดให้คุณสมบัติ ParityReplace เท่ากับตัวอักษร ? หรือในบางกรณีก็จะกำหนดให้เท่ากับ สตริงว่าง ("") เพื่อไม่ให้มีการแทนที่ตัวอักษรที่ไม่เป็นจริง ในขณะที่เกิดข้อผิดพลาด parity error ซึ่งในกรณีคุณสมบัติ CommEvent ก็จะถูกกำหนดให้มีค่าเท่ากับ comEventRXParity ทันที

ในการสื่อสารแบบใช้ parity bit นั้น เป็นการกำหนดให้มีการเพิ่มบิต (0 หรือ 1) ลงในข้อมูลที่มีการส่งไป – มาในแต่ละครั้ง ทั้งนี้เพื่อใช้บิตดังกล่าวในการตรวจสอบความถูกต้องของข้อมูล โดยการตรวจสอบผลบวกของบิตทั้งหมดว่ามีค่าเท่ากับเลขคี่ (1) หรือเลขคู่ (0)

PortOpen

กำหนดสถานะการเปิด (open) หรือปิด (close) ของพอร์ตอนุกรมของเครื่องคอมพิวเตอร์

รูปแบบการใช้งาน

object.PortOpen [= boolean]

boolean หมายถึง ข้อมูลชนิดบูลีน ที่กำหนดสถานะของพอร์ตอนุกรม ดังต่อไปนี้

True หมายถึง พอร์ตอนุกรมถูกเปิด

False หมายถึง พอร์ตอนุกรมถูกปิด (default)

พอร์ตอนุกรมถูกปิดโดยอัตโนมัติโดยคอนโทรล MSComm เมื่อแอปพลิเคชันสิ้นสุดการทำงาน โดยถ้าหากหมายเลขของพอร์ตอนุกรมที่กำหนดให้เปิดไม่มีการติดตั้งอยู่จริง ก็จะเกิดข้อผิดพลาดหมายเลข 68 (Device unavailable) ทันที

Rthreshold

รายงานหรือกำหนดจำนวนตัวอักษรที่จะรับเข้าก่อนที่คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEvReceive และมีการเรียกโพรซีเยอร์เหตุการณ์ OnComm

รูปแบบการใช้งาน

object.Rthreshold [= value]

value หมายถึง ข้อมูลชนิดเลขจำนวนเต็ม ที่กำหนดจำนวนตัวอักษรที่จะรับเข้าก่อนที่คอนโทรล MSComm ตามรายละเอียดข้างต้น

ถ้าหากคุณสมบัติ Rthreshold มีค่าเท่ากับ 0 (default) ก็จะเป็นการยกเลิกการเรียกโพรซีเยอร์เหตุการณ์ OnComm เมื่อมีการรับเข้าตัวอักษรมายังบัฟเฟอร์ด้านรับเข้า และในทางกลับกันถ้าหากคุณสมบัติ Rthreshold มีค่าเท่ากับ 1 ก็จะมีการเรียกโพรซีเยอร์เหตุการณ์ OnComm ทุกครั้งที่มีการรับเข้าตัวอักษรมายังบัฟเฟอร์ด้านรับเข้า

RTSEnable

กำหนดให้มีการมีการใช้งานสาย Request To Send (RTS) ซึ่งโดยปกติสัญญาณ Request To Send จะถูกส่งจากเครื่องคอมพิวเตอร์ไปยังโมเด็ม เพื่อเป็นการแจ้งขอส่งสัญญาณจากคอมพิวเตอร์

รูปแบบการใช้งาน

object.RTSEnable [= boolean]

boolean หมายถึง ข้อมูลชนิดบูลีน ที่กำหนดให้มีการใช้งานสาย Request To Send ดังต่อไปนี้

True หมายถึง มีการใช้งานสายสัญญาณ RTS (Request To Send)

False หมายถึง ไม่มีการใช้งานสายสัญญาณ RTS (default)

ถ้าหากคุณสมบัติ RTSEnable มีค่าเท่ากับ True สายสัญญาณ Request To Send จะอยู่ในสถานะ high เมื่อพอร์ตอนุกรมถูกเปิด และอยู่ในสถานะ low เมื่อพอร์ตอนุกรมถูกปิด โดยปกติสายสัญญาณ Request To Send จะถูกใช้ในการตอบรับการติดต่อสื่อสารแบบฮาร์ดแวร์ RTS/CTS เท่านั้น

Settings

รายงานหรือกำหนดพารามิเตอร์ในการสื่อสารผ่านพอร์ตอนุกรม

รูปแบบการใช้งาน

object.Settings [= value]

value หมายถึง ข้อมูลชนิดสตริง ที่กำหนดพารามิเตอร์ในการสื่อสารผ่านพอร์ตอนุกรม ดังต่อไปนี้

รูปแบบของการกำหนดลำดับของพารามิเตอร์ในการสื่อสารผ่านพอร์ตอนุกรม สำหรับคุณสมบัติ Settings จะต้องเรียงลำดับดังนี้ "BBBB, P, D, S" (โดยปกติทั่ว ๆ ไป จะมีค่าเท่ากับ "9600 N, 8, 1") เพราะถ้าหากลำดับไม่ถูกต้องก็จะทำให้เกิดข้อผิดพลาดหมายเลข 380 (Invalid property value) ทันที ซึ่งสัญลักษณ์แต่ละตัวมีความหมายดังนี้

BBBB

ความเร็วของการส่งถ่ายข้อมูลในหน่วยของ baud rate ซึ่งในทางปฏิบัติ 1 baud rate อาจจะมีค่าเท่ากับ 1 bps (bits per second) หรือมากกว่าก็ได้ สำหรับค่าของ baud rate ที่คอนโทรล MSComm สามารถรับได้มีค่าดังต่อไปนี้ 110, 300, 600, 1200, 2400, 9600 (default), 14400, 19200, 28800, 38400 (reserved), 56000 (reserved), 128000 (reserved) หรือ 256000 (reserved)

P บิตพาริตี้ (parity bit) สำหรับใช้ในการตรวจสอบความถูกต้องของข้อมูล ซึ่งสามารถมีค่าได้ดังต่อไปนี้

พารามิเตอร์	ความหมาย
E	Even
M	Mark
N	None (default)
O	Odd
S	Space

D ขนาดของบิตข้อมูล ซึ่งสามารถมีค่าได้ดังต่อไปนี้ 1 (default), 1.5 หรือ 2

S ขนาดของบิตหยุด (stop bit) ซึ่งสามารถมีค่าได้ดังต่อไปนี้ 1 (default), 1.5 หรือ 2

Sthreshold

รายงานหรือกำหนดจำนวนตัวอักษรที่น้อยที่สุดที่ถูกจัดเก็บในบัฟเฟอร์ด้านส่งออก ก่อนที่คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเท่ากับ comEvSend และมีการเรียกเหตุการณ์ OnComm

รูปแบบการใช้งาน object.Sthreshold [= value]

value หมายถึง ข้อมูลชนิดตัวเลขจำนวนเต็ม ที่กำหนดจำนวนตัวอักษรที่น้อยที่สุดที่ถูกจัดเก็บในบัฟเฟอร์ด้านส่งออก ตามรายละเอียดข้างต้น

ถ้าหากคุณสมบัติ Sthreshold มีค่าเท่ากับ 0 (default) ก็จะเป็นการยกเลิกการเรียกโพรซีเยอร์เหตุการณ์ OnComm เมื่อมีการส่งออกตัวอักษรไปยังบัฟเฟอร์ด้านส่งออก และในทางกลับกันถ้าหากคุณสมบัติ Sthreshold มีค่าเท่ากับ 1 ก็จะมีการเรียกโพรซีเยอร์เหตุการณ์ OnComm เมื่อบัฟเฟอร์ด้านส่งออกว่าง

โดยถ้าหากจำนวนของตัวอักษรในบัฟเฟอร์ด้านส่งออกน้อยกว่าค่าตัวเลขที่กำหนด คุณสมบัติ CommEvent ก็จะมีค่าเท่ากับ comEvSend และพร้อมทั้งเกิดการโพรซีเยอร์เหตุการณ์ OnComm ทันที เช่น สมมติให้คุณสมบัติ Sthreshold มีค่าเท่ากับ 10 และถ้าหากจำนวนตัวอักษรในบัฟเฟอร์ด้านส่งออกลดลงจาก 10 เป็น 9 ก็จะเกิดเหตุการณ์ OnComm ทันที เป็นต้น

โพรซีเยอร์เหตุการณ์

OnComm

OnComm

เกิดขึ้นเมื่อมีการเปลี่ยนแปลงค่าของคุณสมบัติ CommEvent ซึ่งเป็นการบอกถึงการเกิดข้อผิดพลาดหรือมีการสื่อสารเกิดขึ้นก็ได้

รูปแบบโพรซีเยอร์เหตุการณ์

Private Sub object_OnComm ()

โดยปกติเมื่อคอนโทรล MSComm มีการเรียกโพรซีเยอร์เหตุการณ์ OnComm เรามักจะมีการเขียนโค้ดภายในโพรซีเยอร์เหตุการณ์นี้ เพื่อทำการตรวจสอบค่าของคุณสมบัติ CommEvent ทั้งนี้เพื่อตรวจสอบสถานะของการสื่อสารหรือข้อผิดพลาดที่เกิดขึ้นนั่นเอง

4.8 ขั้นตอนในการสร้าง

ขั้นตอนในการสร้างส่วน Interfaceนั้นประกอบด้วย

1 เปิดโปรแกรม Visual Basic 6

2 เลือก Standread.EXE แล้ว Click Open

3 หลังจากนั้นจะพบกับ Project และ Form

4 จากนั้นให้ดับเบิลคลิก Mouse ที่ Control ที่อยู่ทางด้านขวามือก็จะพบ Control วางอยู่ที่ Form

5 จากนั้นเลือก Add Component แล้วเช็คเครื่องหมายถูกที่ Microsoft Comm Control เพื่อที่จะใช้ Property MSComm

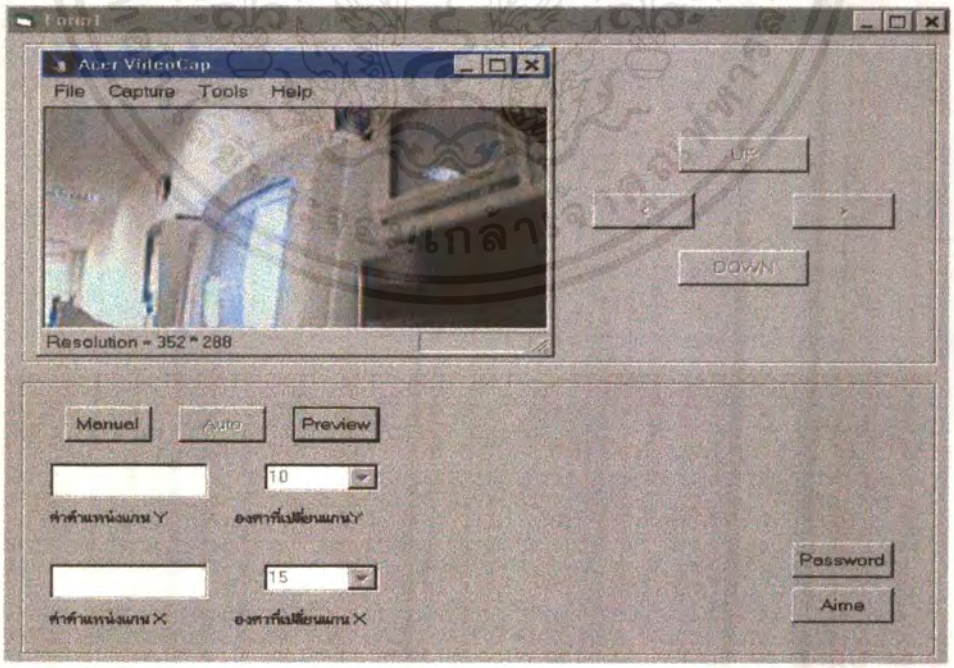
6 เขียน Program ควบคุมการทำงาน

หมายเหตุ ภาพขั้นตอนในการสร้างและ Program ควบคุมการทำงานสามารถดูได้ที่ภาคผนวก ข.

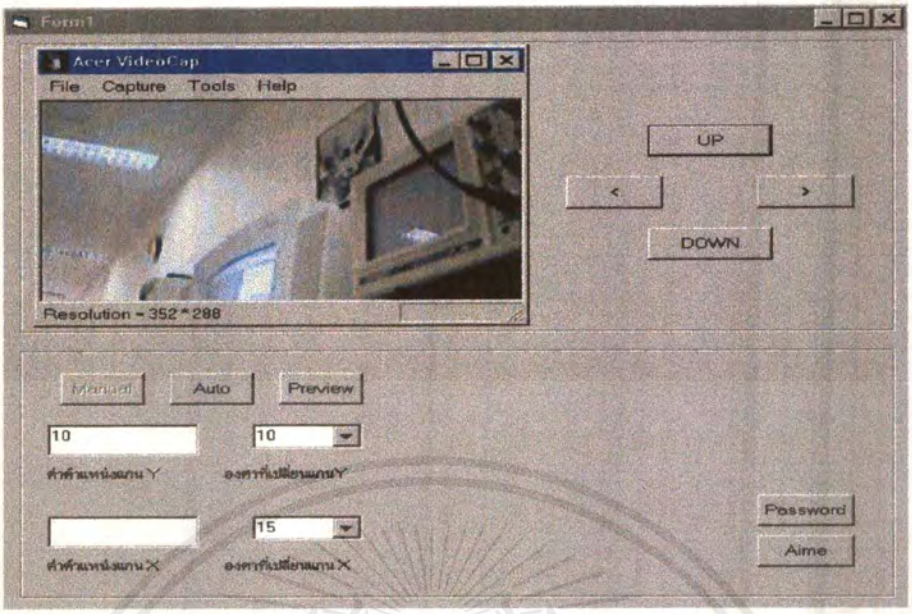
4.9 Software สำเร็จรูปที่สร้างได้ และวิธีการใช้โปรแกรม

หัวข้อที่ผ่านมานั้นเป็นการอธิบายถึงวิธีการสร้างส่วน User Interface ในหัวข้อนี้นั้นจะได้แสดงหน้าต่าง Interface ที่สร้างขึ้นและวิธีการใช้งานเมื่อ User ได้ทำการเปิด Software ที่สร้างขึ้นก็จะได้ภาพดังรูปที่ 4.6 หลังจากนั้น User สามารถเลือกใช้งานใน Modes ที่มีอยู่ใน Program รวมถึง Option ต่างๆได้ การทำงานของ Program นั้นขณะที่ผู้ใช้เปิดมาตอนแรก Program จะทำงานอยู่ใน Modes ของการควบคุมแบบ Automatic ในModes การทำงานนี้นั้นระบบรักษาความปลอดภัยจะทำงานอัตโนมัติ จากนั้นถ้า User เลือกไปที่ Modes การทำงานไปเป็นแบบ Manual ขณะที่อยู่ใน Modes การทำงานนี้ User สามารถที่จะควบคุมการทำงานของระบบรักษาความปลอดภัยได้ด้วยตัว User ถ้า Userกดปุ่ม Preview Userจะสามารถเห็นสิ่งต่างๆในตำแหน่งที่กล้อง Plane อยู่ดังภาพที่ 4.7 จะเป็น

ภาพในขณะที่ User เลือกที่จะควบคุมระบบรักษาความปลอดภัยด้วยตัว User เอง นอกจากนี้ User สามารถที่จะปรับตำแหน่งมุมในการหมุนของกล้องได้โดยการเลือกค่ามุมที่มีให้ในช่องค่าองศา

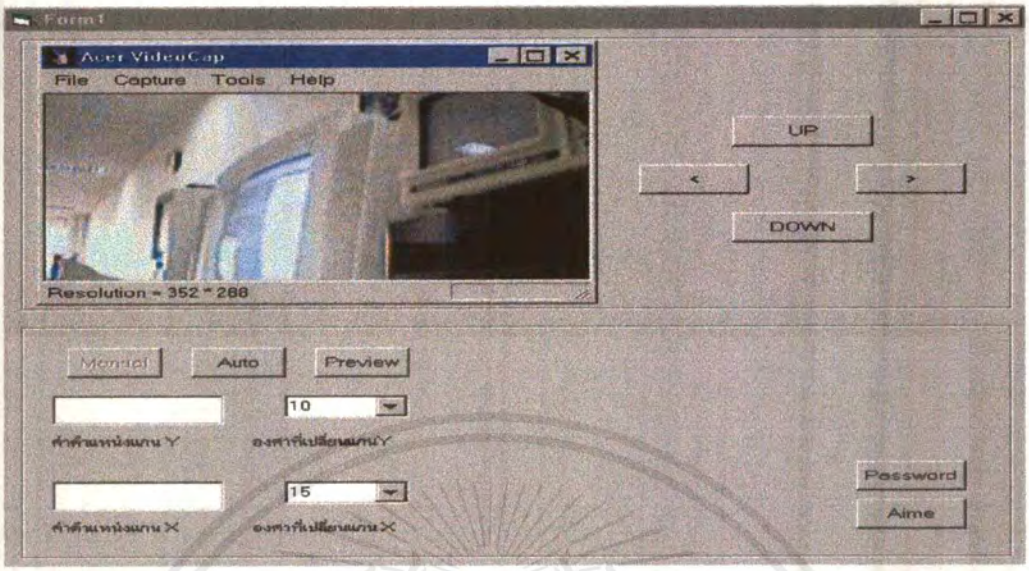


รูปที่ 4.6 ภาพ Interface ที่สร้างได้



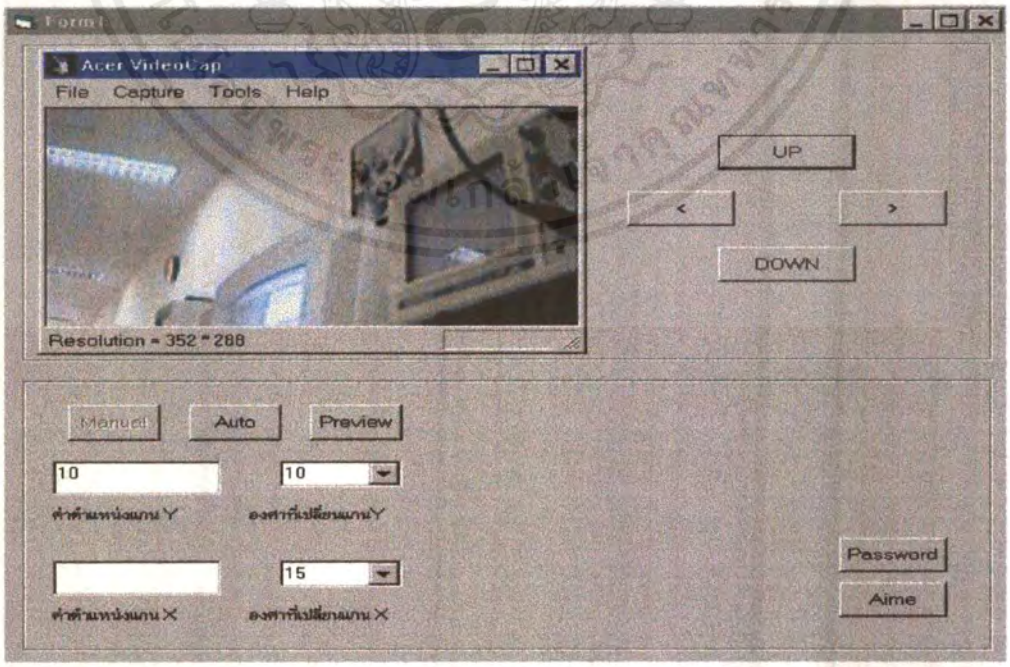
รูปที่ 4.7 ภาพแสดงการทำงานเมื่อผู้ใช้กด Preview

รูปที่ 4.8 ภาพแสดงการทำงานเมื่อผู้ใช้กด Manual



รูปที่ 4.9 ภาพแสดงการทำงานเมื่อผู้ใช้เลือกใช้การกำหนดค่าขององศาที่ต้องการให้เปลี่ยนไป

หลังจากที่Userได้เลือกค่าองศาที่เปลี่ยนแกน Y และ X ดังรูปที่ 4.9 เมื่อ User กดที่ปุ่มไปทางซ้าย (<) หรือที่ปุ่มทางขวา (>), ขึ้นบน (Up), ลงล้าง (Down) กล้องก็จะหมุนไปตามมุมต่างๆค่าตำแหน่งหรือมุมที่กล้องอยู่จะแสดงให้ User เห็นในช่อง ค่าตำแหน่งแกน Y และ X ดังรูปที่ 4.10

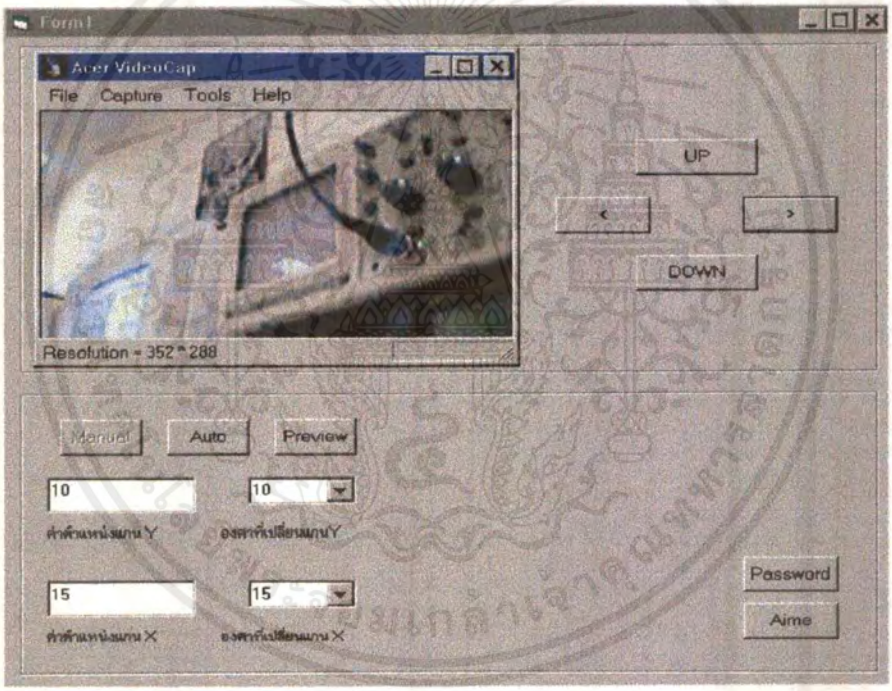


รูปที่4.10 ภาพแสดงเมื่อผู้ใช้กดปุ่ม Up

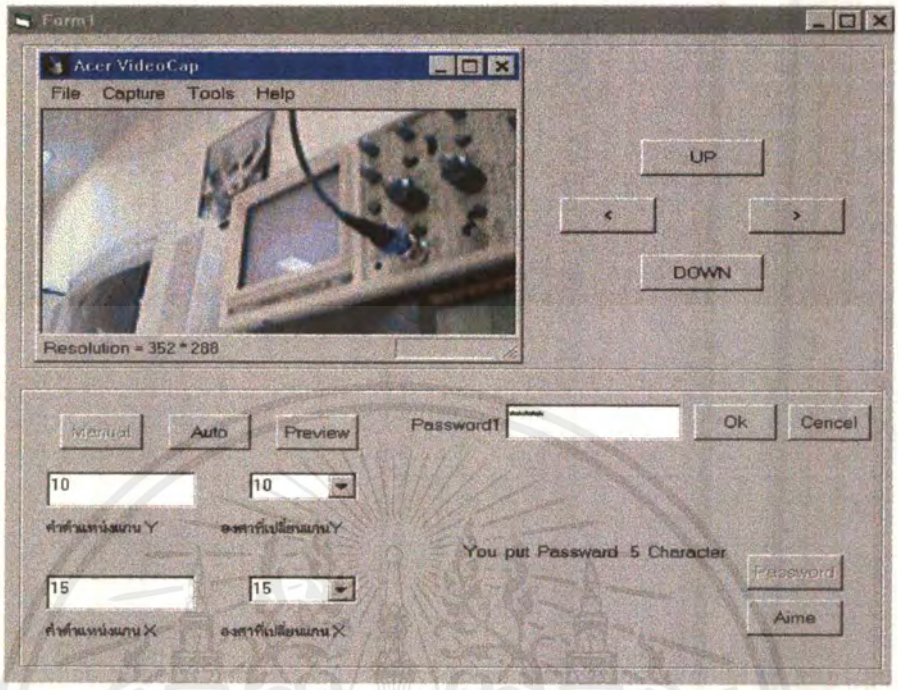
ในรูปที่ 4.10 หลังจากที่ User ใส่ค่าองศาที่เปลี่ยนในแกน Y และแกน X สมมุติว่า User ได้กดไปที่ปุ่ม Up กล้องก็จะหมุนขึ้นไปข้างบน และจะแสดงตำแหน่งมุมที่กล้องหมุนขึ้นไปในช่องค่าตำแหน่งแกน Y ให้ User ทราบ หรือถ้าผู้ใช้กดที่ปุ่ม Down กล้องก็จะหมุนลงและแสดงค่าตำแหน่งให้ทราบเช่นเดียวกัน

ในรูปที่ 4.11 เมื่อ User กดที่ปุ่มขวา (>) กล้องก็จะหมุนไปทางขวาและจะบอกตำแหน่งกล้องในช่องค่าตำแหน่งแกน X เช่นเดียวกัน

ขั้นตอนต่อไปเป็นการสั่งให้ระบบเตือนภัยทำงาน โดยในขั้นตอนใช้ระบบเตือนภัยนี้จะจำกัดจำนวนผู้ที่จะใช้โดยผู้ที่จะเข้ามาใช้จะต้องรู้รหัสผ่าน (Password)จึงจะสามารถใช้งานได้ สมมุติว่าผู้ใช้กดไปที่ปุ่ม Password ก็จะมีการถามรหัสผ่านก่อนดังรูปที่ 4.12

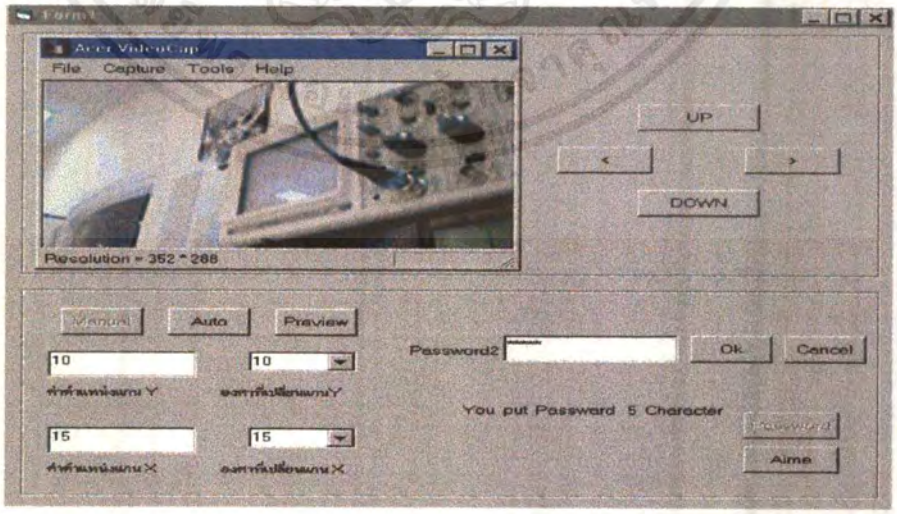


รูปที่ 4.11 ภาพแสดงเมื่อผู้ใช้กดปุ่มขวา (>)



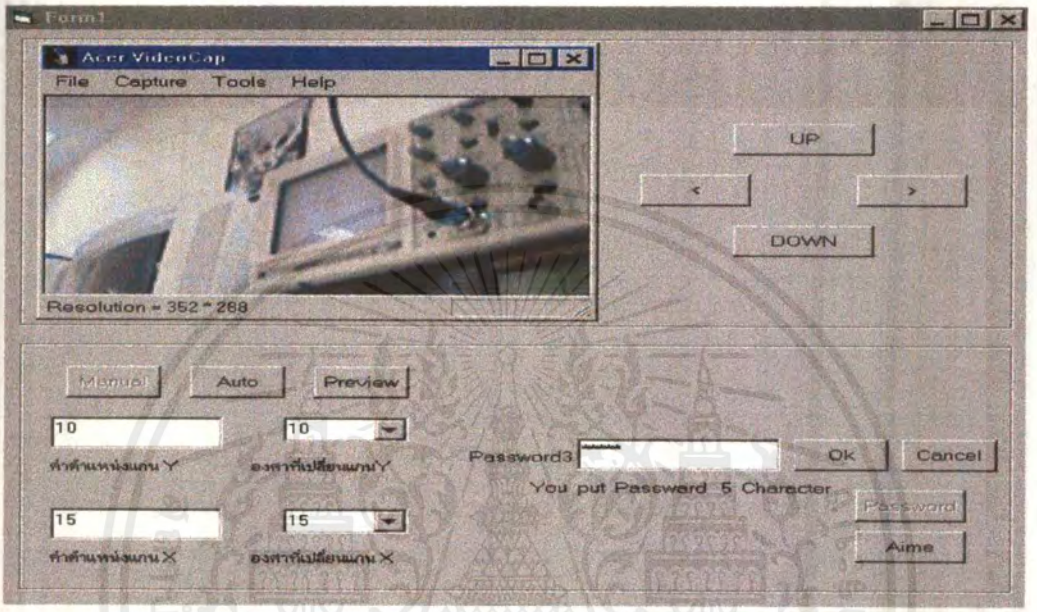
รูปที่ 4.12 ภาพแสดงเมื่อผู้ใช้กดปุ่ม Password

เมื่อผู้ใช้ใส่รหัสผ่านที่ถูกต้องแล้วกดไปที่ปุ่ม OK เมื่อกดไปที่ปุ่ม OK แล้วจะปรากฏหน้าจอเพื่อใช้ผู้ใช้ใส่รหัสผ่านครั้นที่สองลงไปดังรูปที่ 4.13

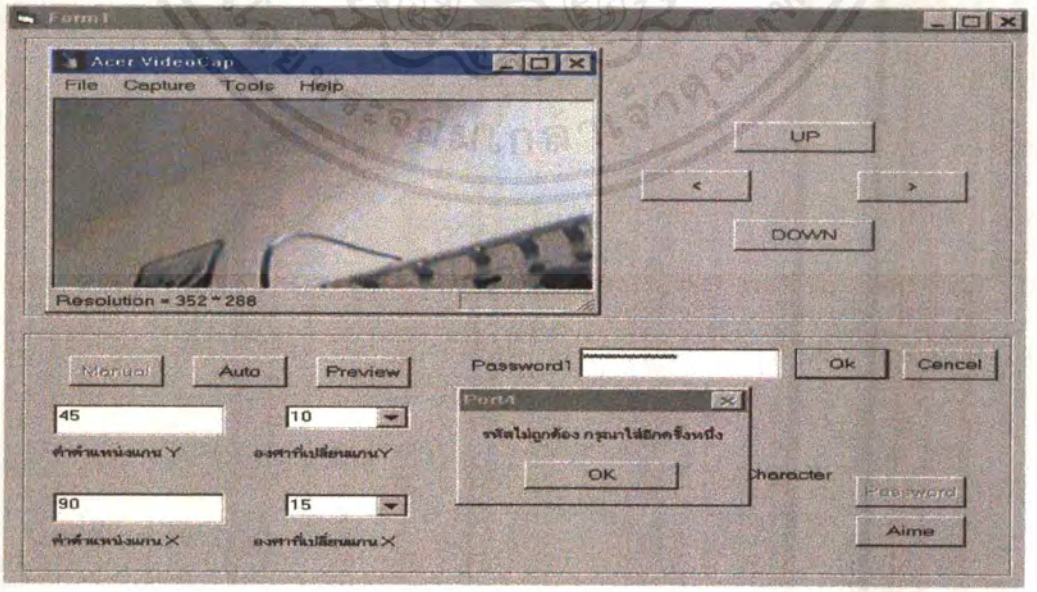


รูปที่ 4.13 ภาพแสดงหน้าต่างให้ผู้ใช้ใส่รหัสผ่านครั้งที่ 2

หลังจากที่ผู้ใช้ใส่รหัสผ่านครั้งที่สองและกดปุ่ม OK ก็จะปรากฏหน้าต่างให้ผู้ใช้ใส่รหัสผ่านครั้งที่สามลงไปดังรูปที่ 4.14 ถ้าผู้ใช้ใส่รหัสผิดผู้ใช้จะต้องใส่รหัสใหม่และจะมีหน้าต่างแจ้งเตือนให้ผู้ใช้ใส่รหัสใหม่ดังรูป 4.15 จะสังเกตุได้ว่าผู้ใช้ต้องใสรหัสผ่านถึงสามครั้งเมื่อผู้ใช้ใส่รหัสผ่านถูกต้องทั้งสามครั้งผู้ใช้สามารถที่จะสั่งให้ระบบเตือนภัยทำงานได้



รูปที่ 4.14 ภาพแสดงให้ผู้ใช้ใส่รหัสผ่านครั้งที่ 3



รูปที่ 4.15 แสดงหน้าต่างแจ้งเตือนให้ผู้ใช้ใส่รหัสใหม่

บทที่ 5
การทดลองและผลการทดลอง

สำหรับในบทนี้จะเป็นการทดลองการใช้งานของโปรแกรมที่ได้ทำการออกแบบมาใช้งานบนเครื่องคอมพิวเตอร์ โดยดูจากภาพที่ได้รับมา และได้มีการสรุปและวิจารณ์ผลการทดลองเพื่อเป็นแนวทางในการออกแบบระบบควบคุมต่อไป

การทดลองที่ 1

การทดสอบการทำงานของปุ่มต่าง ๆบนเครื่องคอมพิวเตอร์

จุดประสงค์

1.ศึกษาผลของการทำงานของปุ่มต่าง ๆ ที่ได้ออกแบบมา

2.ศึกษาเสถียรภาพของโปรแกรมที่ได้ออกแบบมา

ลำดับขั้นการทดลอง

1.ทำการเปิด โปรแกรมขึ้นมา

2.กำหนดค่าขององศาที่ต้องการให้เปลี่ยน

3.ทดลองกดปุ่มบังคับทิศทาง

4.บันทึกผลที่ได้

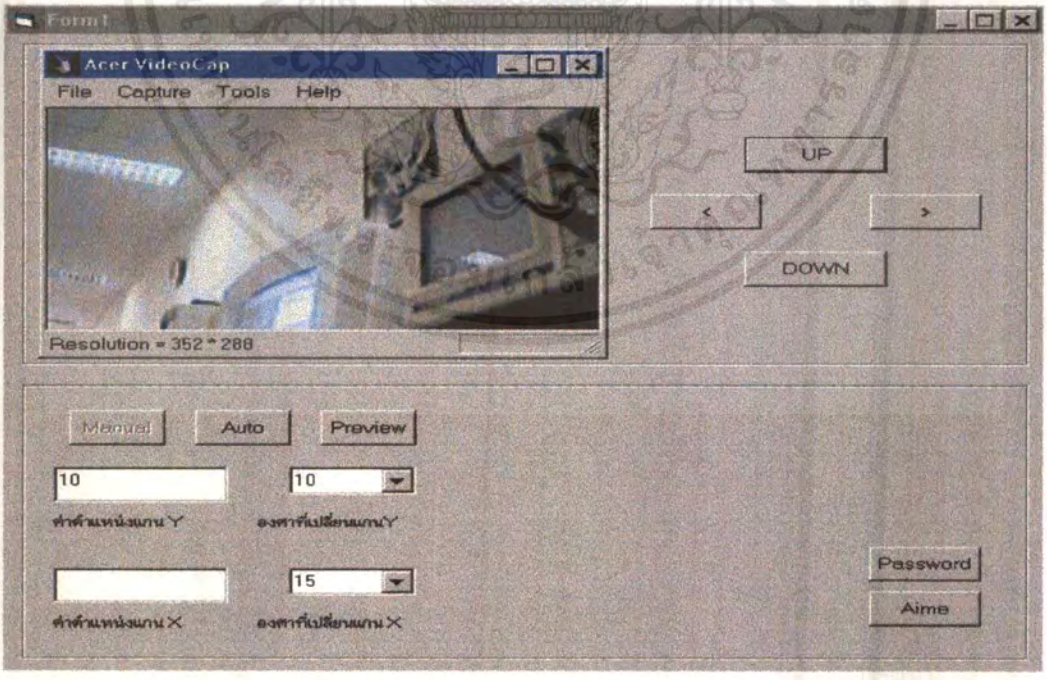
5.ทดลองกดปุ่มอื่นๆ

6.บันทึกผลที่ได้

7.พล็อตกราฟผลการตอบสนองของการกดปุ่ม

8.สรุปผลการทดลอง

ผลการทดลอง

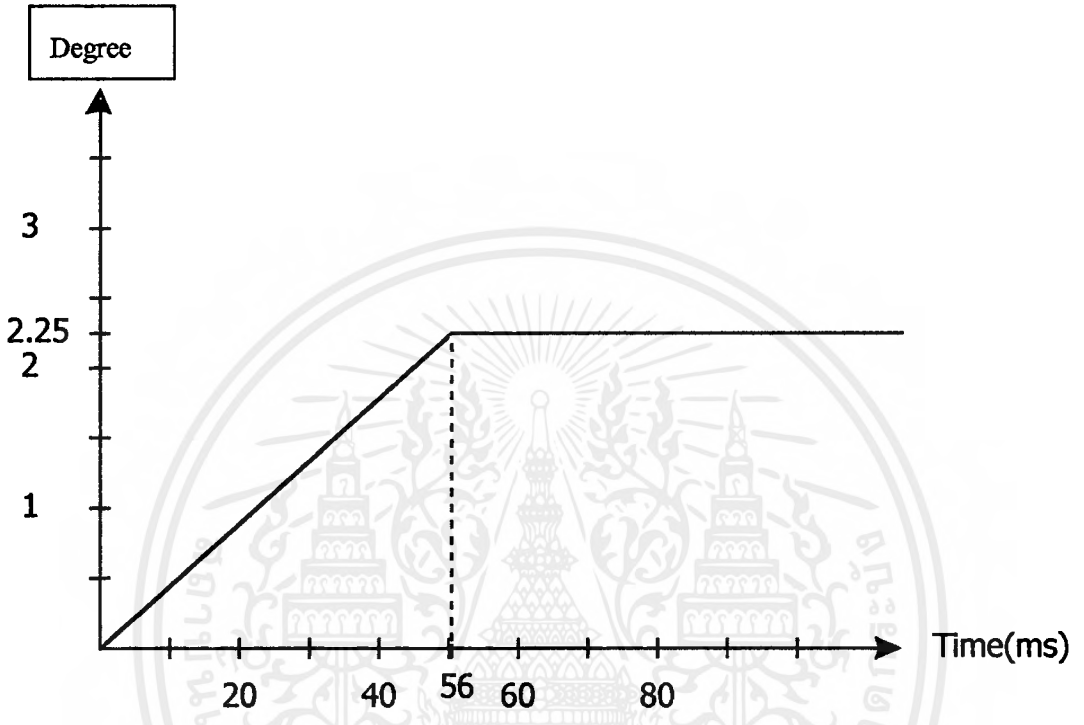


รูปที่ 5-1 หน้าต่างที่ใช้งานบนเครื่องคอมพิวเตอร์

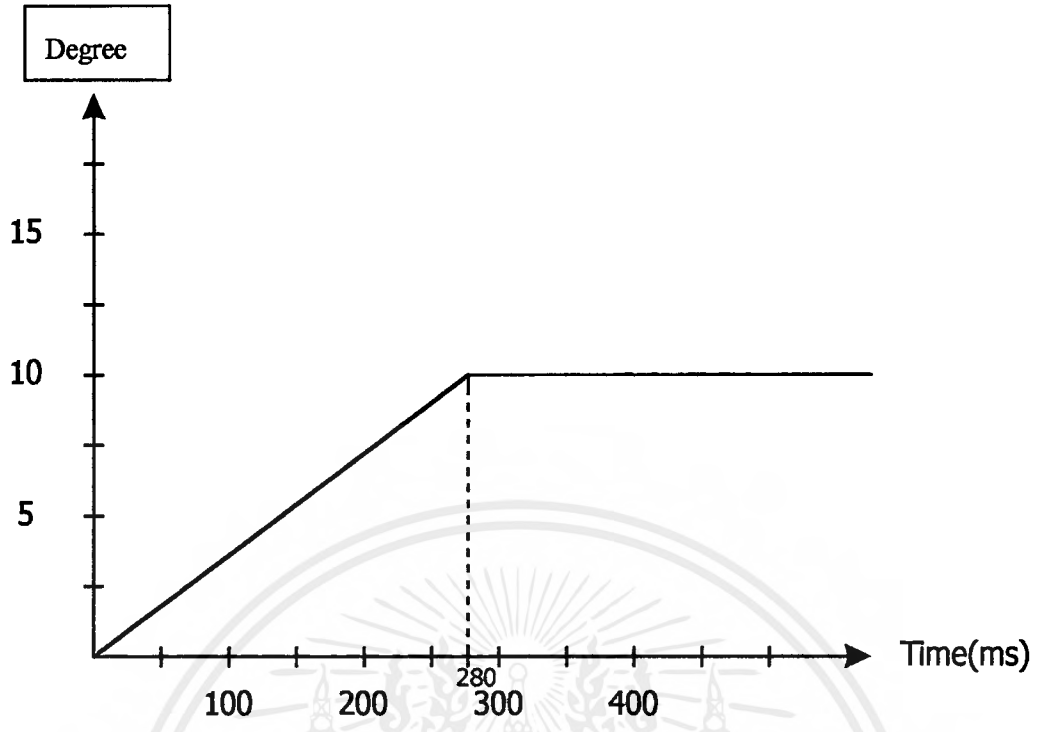
ปุ่มที่ใช้	ค่าที่กำหนด	ค่าวัดได้จาก LED	ค่าเลขฐาน10
ปุ่มขึ้น 2.25 องศา	55	00110111	55
ปุ่มขึ้น 10 องศา	59	00111011	59
ปุ่มขึ้น 30 องศา	67	01000011	67
ปุ่มลง 2.25 องศา	81	01011101	81
ปุ่มลง 10 องศา	85	01010101	85
ปุ่มลง 30 องศา	93	01011101	93
ปุ่มซ้าย 3.6 องศา	29	00011101	29
ปุ่มซ้าย 15 องศา	32	00100000	32
ปุ่มซ้าย 45 องศา	40	00101000	40
ปุ่มซ้าย 90 องศา	53	00110101	53
ปุ่มขวา 3.6 องศา	3	00000011	3
ปุ่มขวา 15 องศา	6	00000110	6
ปุ่มขวา 45 องศา	14	00001110	14
ปุ่มขวา 90 องศา	27	00011011	27
ปุ่ม Automatic	101	01100101	101
ปุ่ม Manual	102	01100110	102

ตารางที่ 5-1 แสดงค่าที่วัดได้เมื่อกดปุ่มต่างๆ

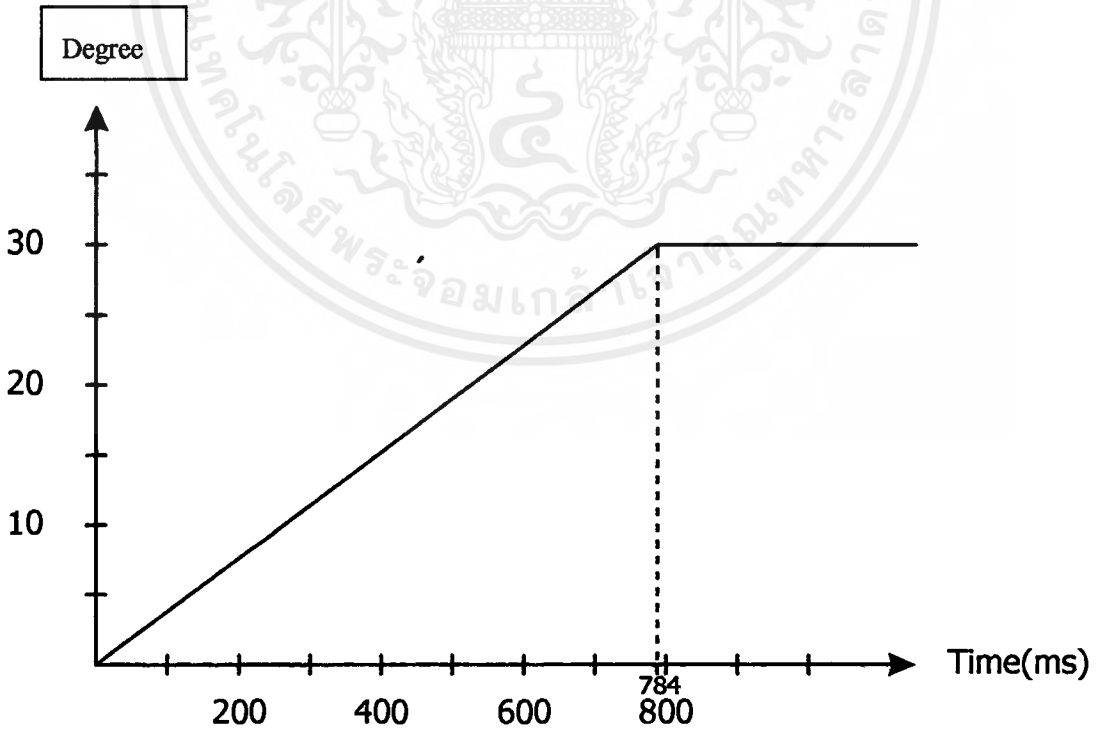
พล็อตกราฟผลการตอบสนองของการกดปุ่ม



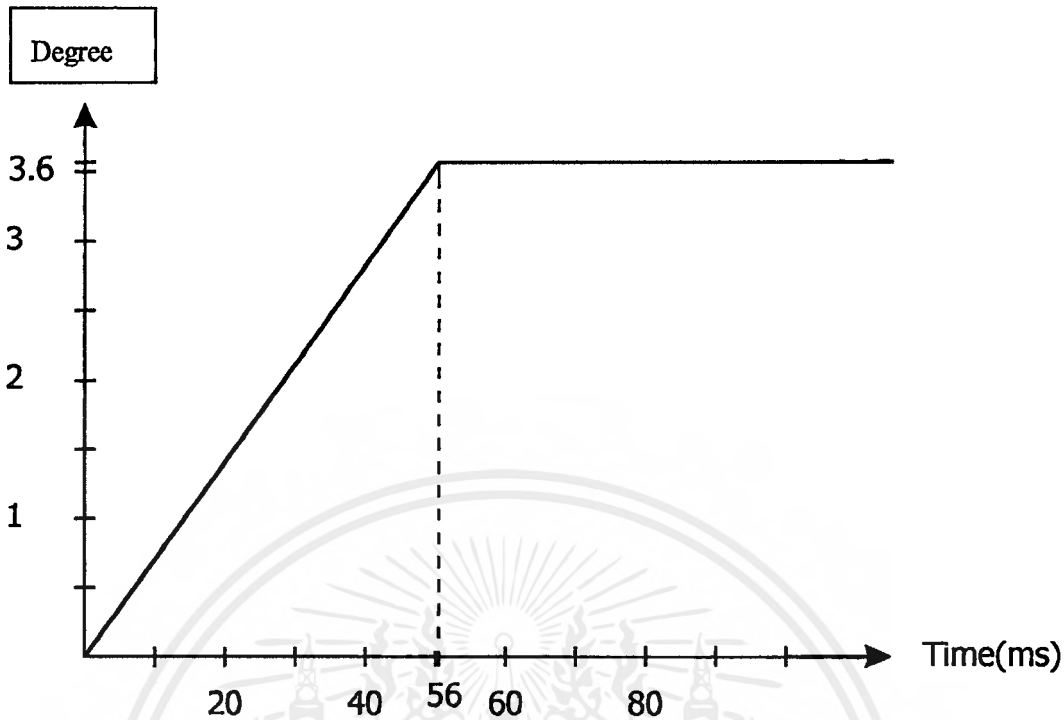
รูปที่ 5-2 กราฟแสดงผลการตอบสนองของการกดปุ่มขึ้น 2.25 องศา



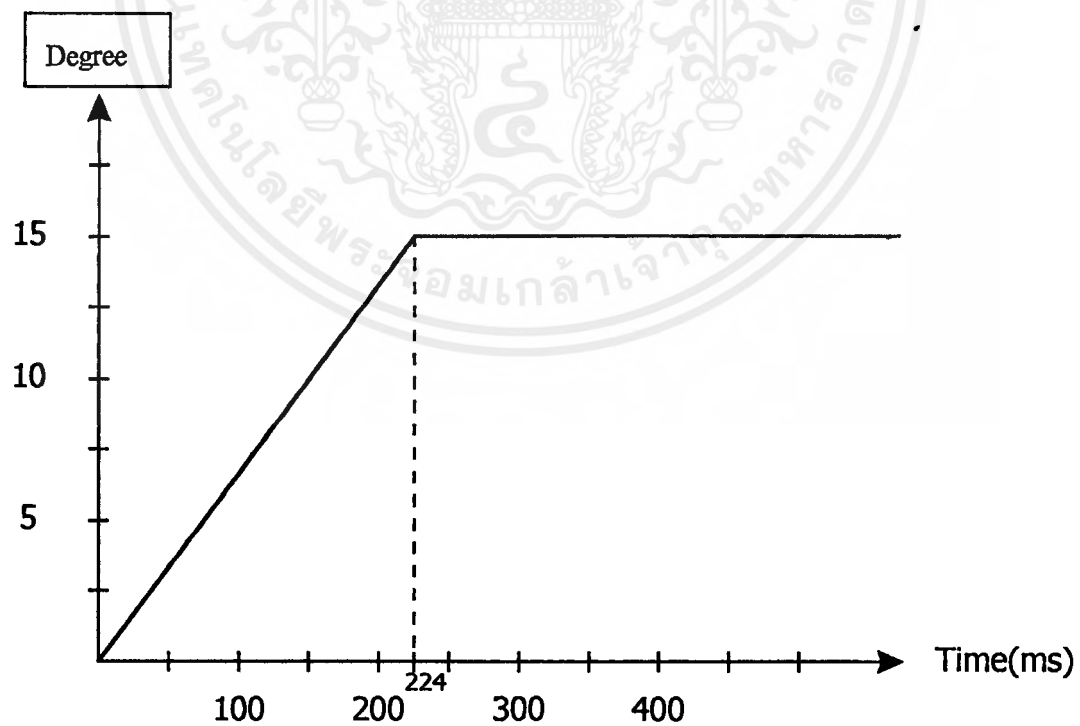
รูปที่ 5-3 กราฟแสดงผลการตอบสนองของการกดปุ่มขึ้น 10 องศา



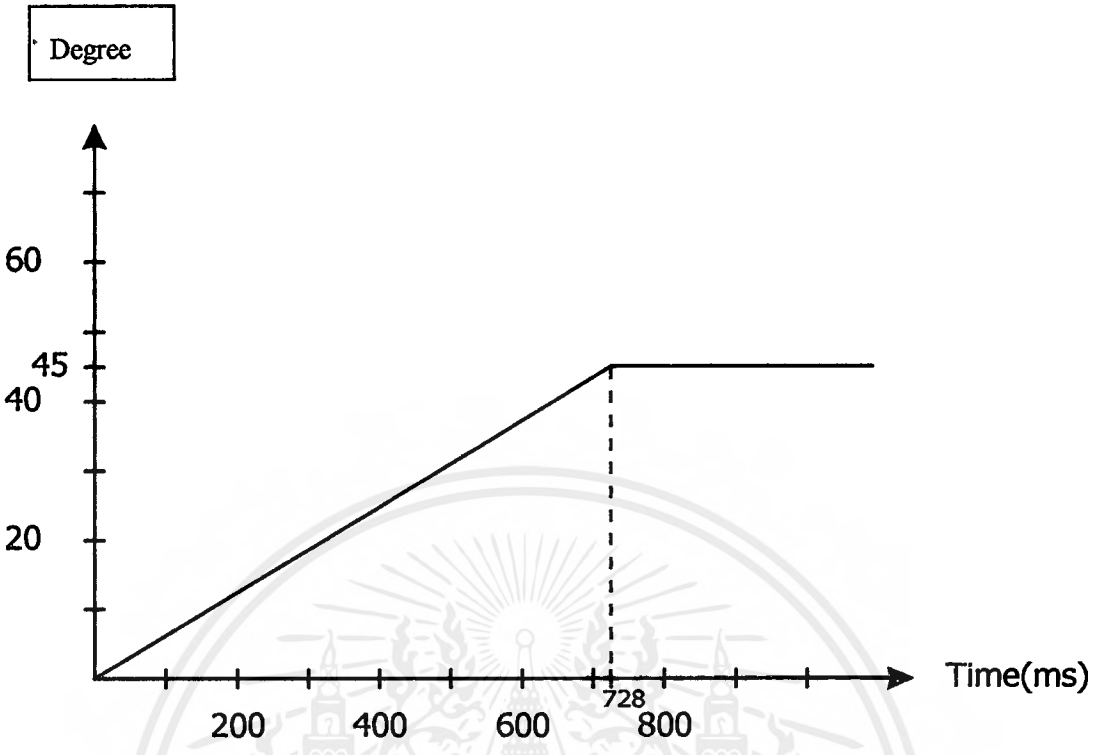
รูปที่ 5-4 กราฟแสดงผลการตอบสนองของการกดปุ่มขึ้น 30 องศา



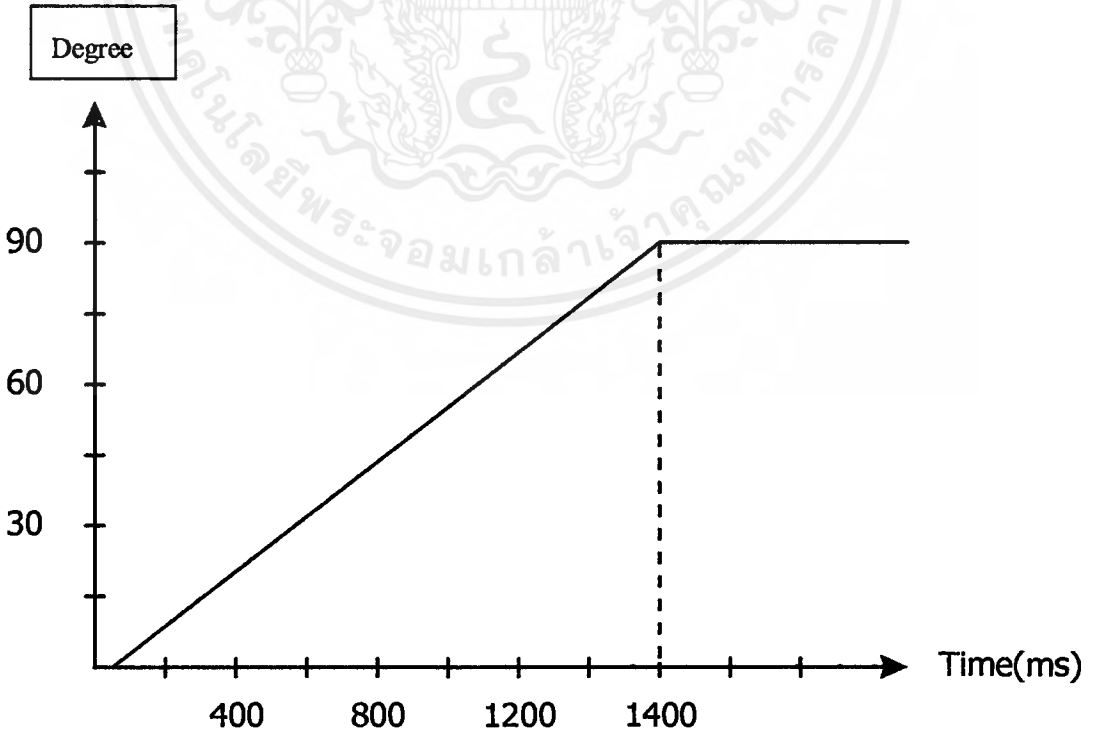
รูปที่ 5-5 กราฟแสดงผลการตอบสนองของการกดปุ่มซ้าย 3.6 องศา



รูปที่ 5-6 กราฟแสดงผลการตอบสนองของการกดปุ่มซ้าย 15 องศา



รูปที่ 5-7 กราฟแสดงผลการตอบสนองของการกดปุ่มซ้าย 45 องศา



รูปที่ 5-8 กราฟแสดงผลการตอบสนองของการกดปุ่มซ้าย 90 องศา

สรุปผลการทดลอง

จากการทดสอบการทำงานของปุ่มต่างๆ บนเครื่องคอมพิวเตอร์ ค่าที่ได้นั้นมีค่าตรงกับค่าที่กำหนดไว้นั้นแสดงว่าจากที่เขียนโปรแกรมมานั้นถูกต้อง และส่งค่าออกมาได้ ตามที่ตัวไมโครคอนโทรลเลอร์นั้น ต้องการ ค่าเหล่านี้จะทำให้ตัวฐานกล้องเคลื่อนไหวตามที่เราสั่งงานได้ ค่าที่ได้นี้จะส่งออกไปยัง RS 232 หรือ พอร์ตอนุกรมนั่นเอง ส่วนกราฟที่ได้นั้นจะเป็นกราฟ แสดงค่าเวลาที่ใช้ในการทำงานของฐานกล้องเมื่อมีการกดปุ่มควบคุมที่คอมพิวเตอร์ซึ่งมีความเร็วในการตอบสนองการทำงานสูงมาก จึงสรุปได้ว่าการทำงานของฐานกล้องเป็นการทำงานแบบเรียลไทม์

การทดลองที่ 2

การตอบสนองของฐานกล้องกับค่าที่กำหนดบนเครื่องคอมพิวเตอร์

จุดประสงค์

1.ศึกษาถึงผลการหมุนของฐานกล้องกับตัวเลขบนหน้าจอ

2.ศึกษาเสถียรภาพของโปรแกรมที่ได้ออกแบบมา

ลำดับขั้นการทดลอง

1.ทำการเปิด โปรแกรมขึ้นมา

2.กำหนดค่าขององศาที่ต้องการให้เปลี่ยน

3.ทดลองกดปุ่มบังคับทิศทาง

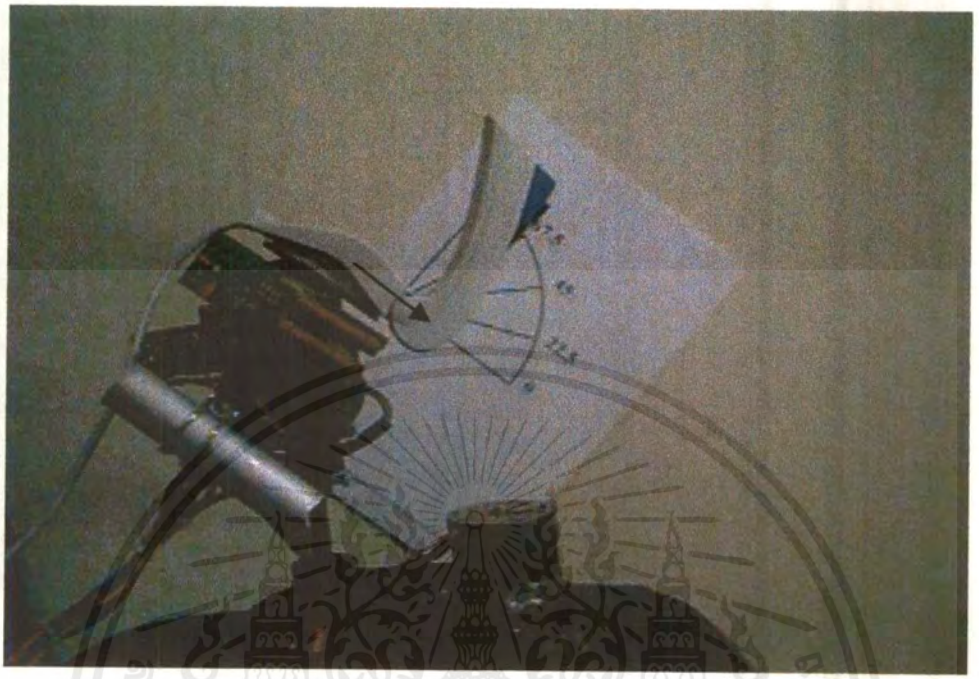
4.บันทึกผลที่ได้

5.เปลี่ยนค่าขององศา

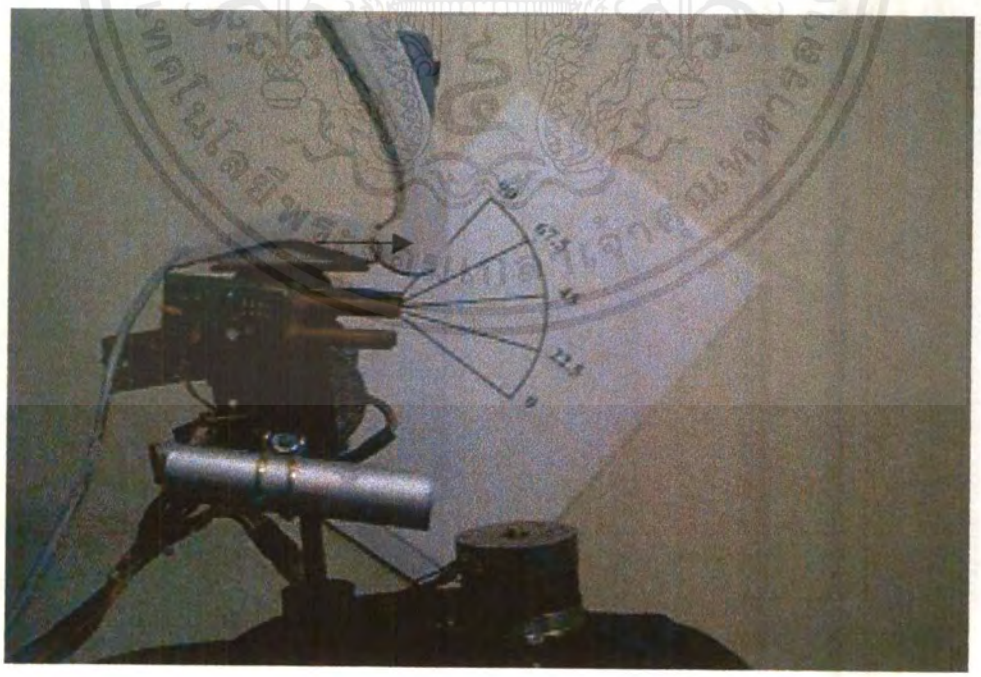
6.กลับไปทำขั้นที่ 3-4-5

7.สรุปผลการทดลอง

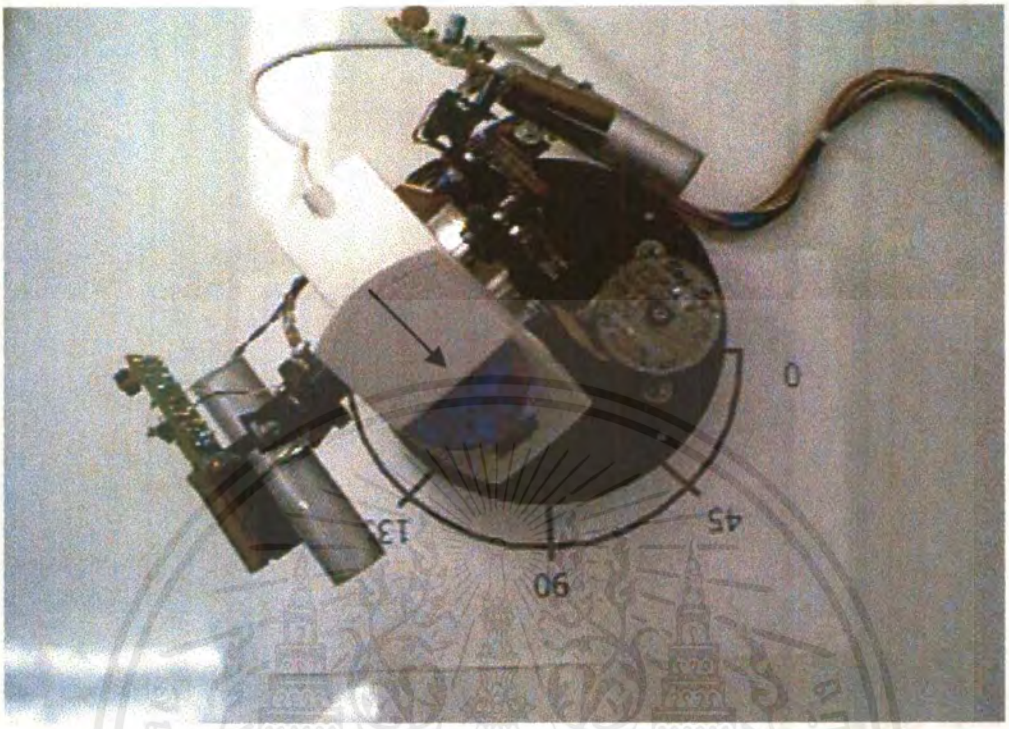
ผลการทดลอง



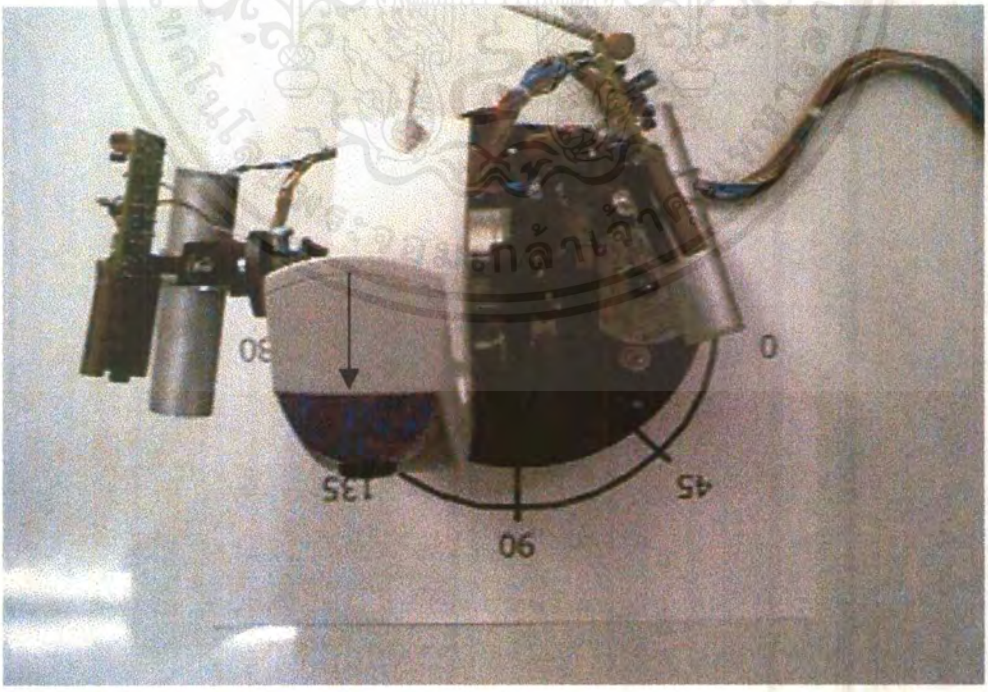
รูปที่ 5-9 แสดงตำแหน่งของกล้องมองจากด้านข้างที่ 0 องศา



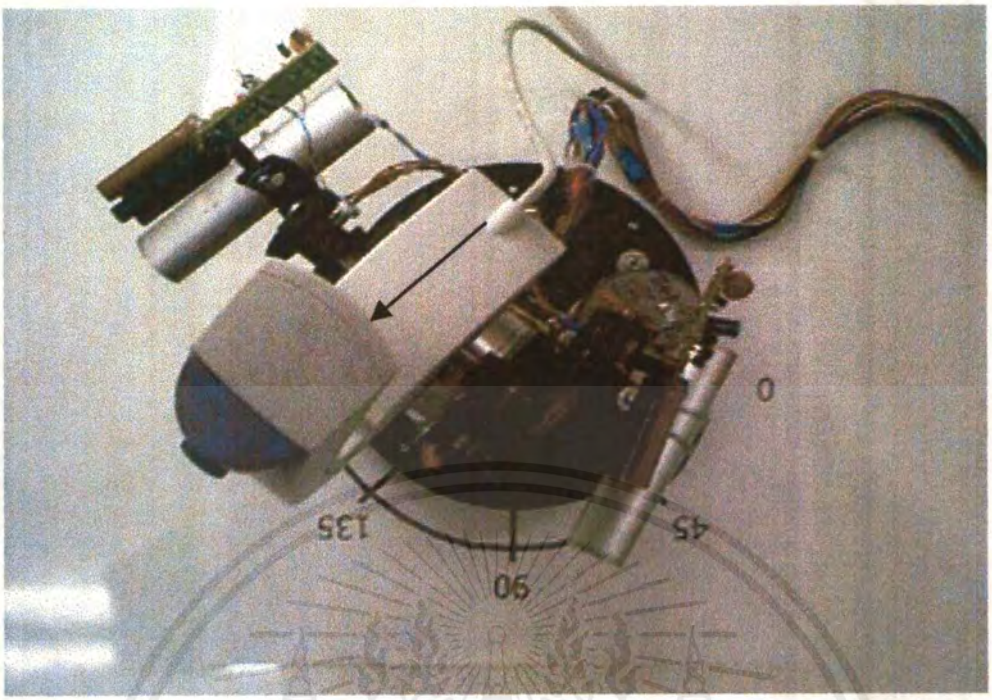
รูปที่ 5-10 แสดงตำแหน่งของกล้องมองจากด้านข้างที่ 45 องศา



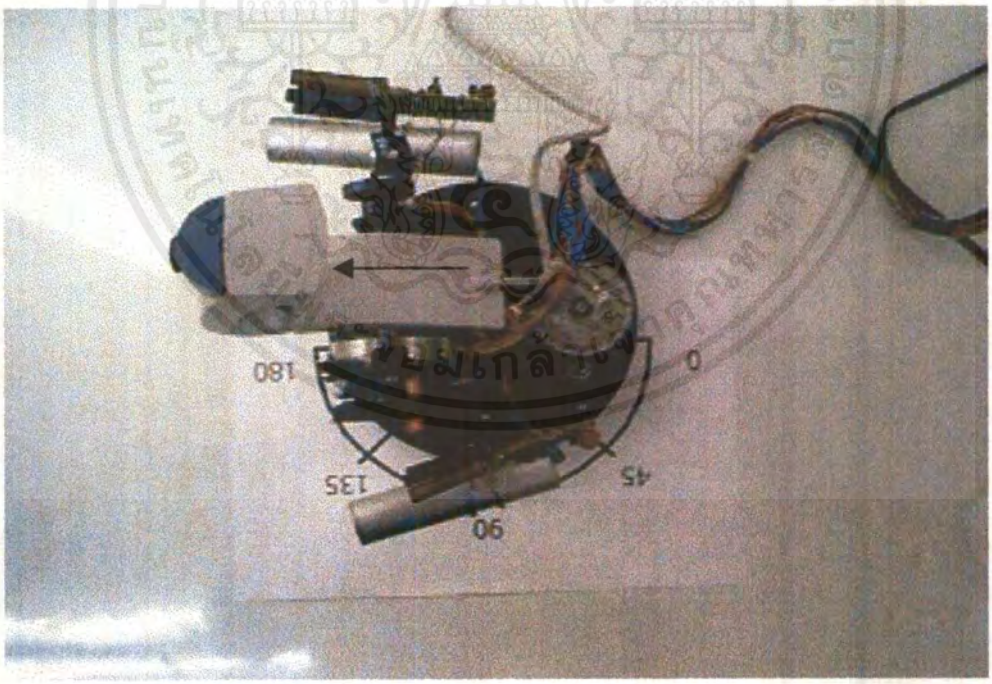
รูปที่ 5-13 แสดงตำแหน่งของกล้องด้านบนที่ 45 องศา



รูปที่ 5-14 แสดงตำแหน่งของกล้องมองจากด้านบนที่ 90 องศา



รูปที่ 5-15 แสดงตำแหน่งของกล้องมองจากด้านบนที่ 135 องศา



รูปที่ 5-16 แสดงตำแหน่งของกล้องมองจากด้านบนที่ 180 องศา

สรุปผลการทดลอง

จากการวัดองศาของฐานกล้องกับค่าตัวเลขบนหน้าจอบนเครื่องคอมพิวเตอร์ นั้นมีค่าเหมือนกับ ตำแหน่งที่กำหนดไว้ แสดงว่าการทำงานของโปรแกรมที่ได้เขียนไว้นั้นทำงานถูกต้อง

บทที่ 6
อภิปรายผลและสรุปผลการทดลอง

การทดสอบการทำงานของปุ่มต่างๆ บนเครื่องคอมพิวเตอร์นั้น ปรากฏว่าค่าที่ได้นั้นมีค่าตรงกับค่าที่กำหนดไว้นั้น แสดงว่าจากที่เขียนโปรแกรมมานั้นถูกต้อง และส่งค่าออกมา ได้ค่าตามที่ตัวไมโครคอนโทรลเลอร์นั้น ต้องการ ค่าเหล่านี้จะทำให้ตัวฐานกล้องเคลื่อนไหวตามที่เราสั่งงานได้ ค่าที่ได้นี้จะส่งออกไปยัง RS 232 หรือ พอร์ตอนุกรมนั่นเอง

การทดสอบการทำงานของปุ่มต่างๆ บนเครื่องคอมพิวเตอร์นั้น ปรากฏว่าค่าที่ได้นั้นมีค่าตรงกับค่าที่กำหนดไว้นั้น ค่านี้จะถูกส่งออกพอร์ตไปที่ไมโครคอรโทรลเลอร์ เพื่อบังคับฐานกล้องให้หมุนไปตามที่เราต้องการ ส่วนกราฟที่ได้นั้นจะเป็นกราฟ แสดงค่าเวลาที่ใช้ในการทำงานของฐานกล้องเมื่อมีการกดปุ่มควบคุมที่คอมพิวเตอร์ซึ่งมีความเร็วในการตอบสนองการทำงานสูงมาก จึงสรุปได้ว่าการทำงานของฐานกล้องเป็นการทำงานแบบเรียลไทม์

การทดลองวัดองศาของฐานกล้องที่คอมพิวเตอร์นั้น ผลปรากฏว่าค่าขององศานั้นกับพิกัดของฐานกล้องที่หมุนไปนั้นมีค่าที่เหมือนกัน จึงสรุปได้ว่าการทำงานของ ไมโครคอนโทรลเลอร์นั้นทำงานสอดคล้องกับค่าที่ส่งออกไป

บรรณานุกรม

1. ธาริน สิทธิธรรมชารี , คู่มือการเขียนโปรแกรม Microsoft Visual Basic 6.0 ฉบับเพื่อใช้งานจริง Success Media Co.,Ltd

2. ธาริน สิทธิธรรมชารี , สุรสิทธิ์ ศิวประสบศักดิ์ , คู่มือการเขียนโปรแกรม Microsoft Visual Basic 6.0 ฉบับเพื่อการประยุกต์ใช้งาน , Success Media Co.,Ltd

3. สัจจะ จรัสรุ่งรวีวร , สมพร จีรสกุล , ASP และแอพพลิเคชั่นฐานข้อมูลสำหรับ อินเตอร์เนต บริษัท ดวงกมลสมัย จำกัด

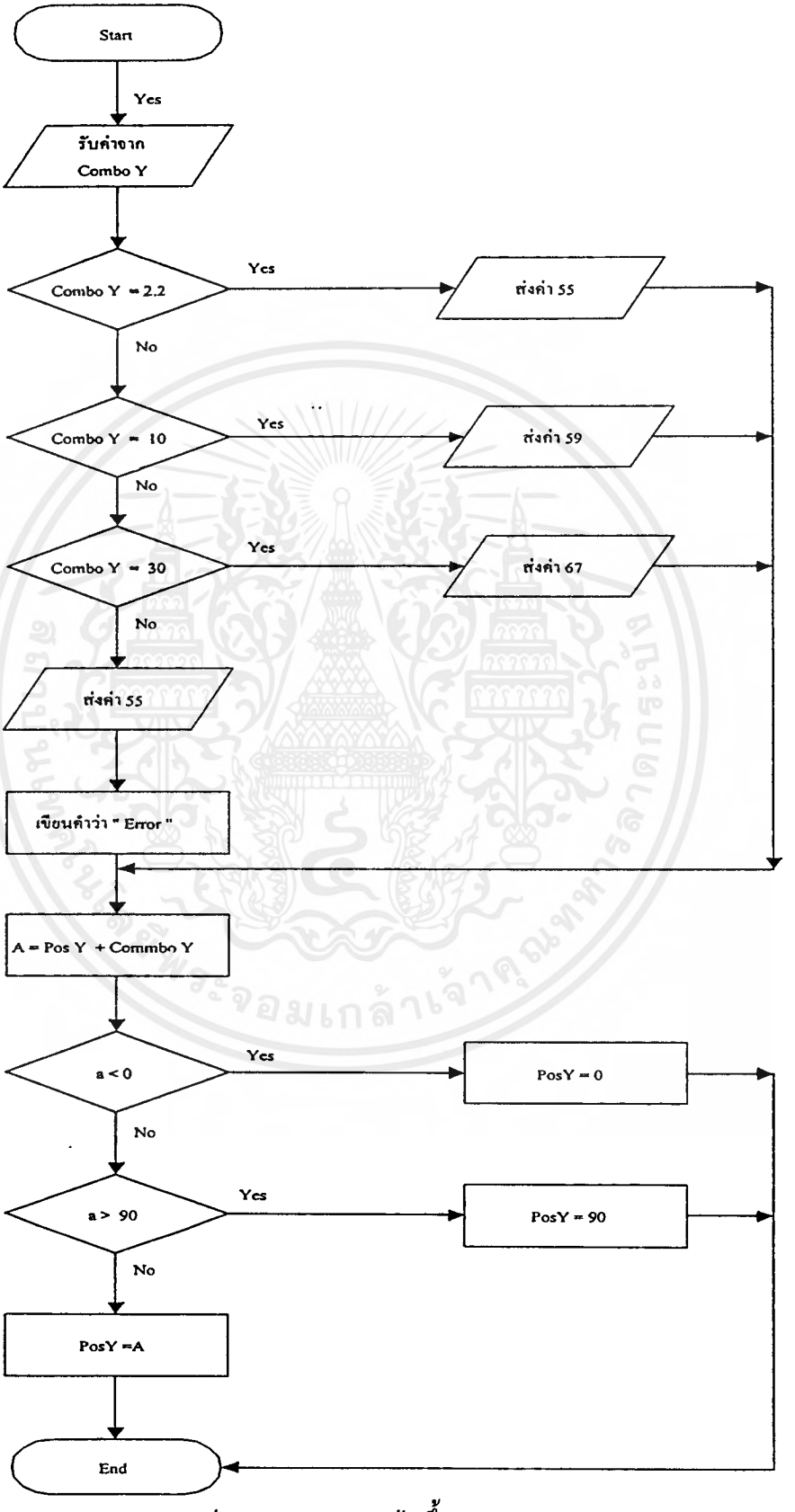
4. กฤดา ใจเย็น , ชัยวัฒน์ ลิ้มพรจิตรวิไล , วรพจน์ กรแก้ววัฒนกุล , เรียนรู้ไมโครคอลโทรลเลอร์ อย่างง่ายกับเบสิกแสตมป์ 2 , (c) Innovative Experiment co.,Ltd

ภาคผนวก

ภาคผนวก ก

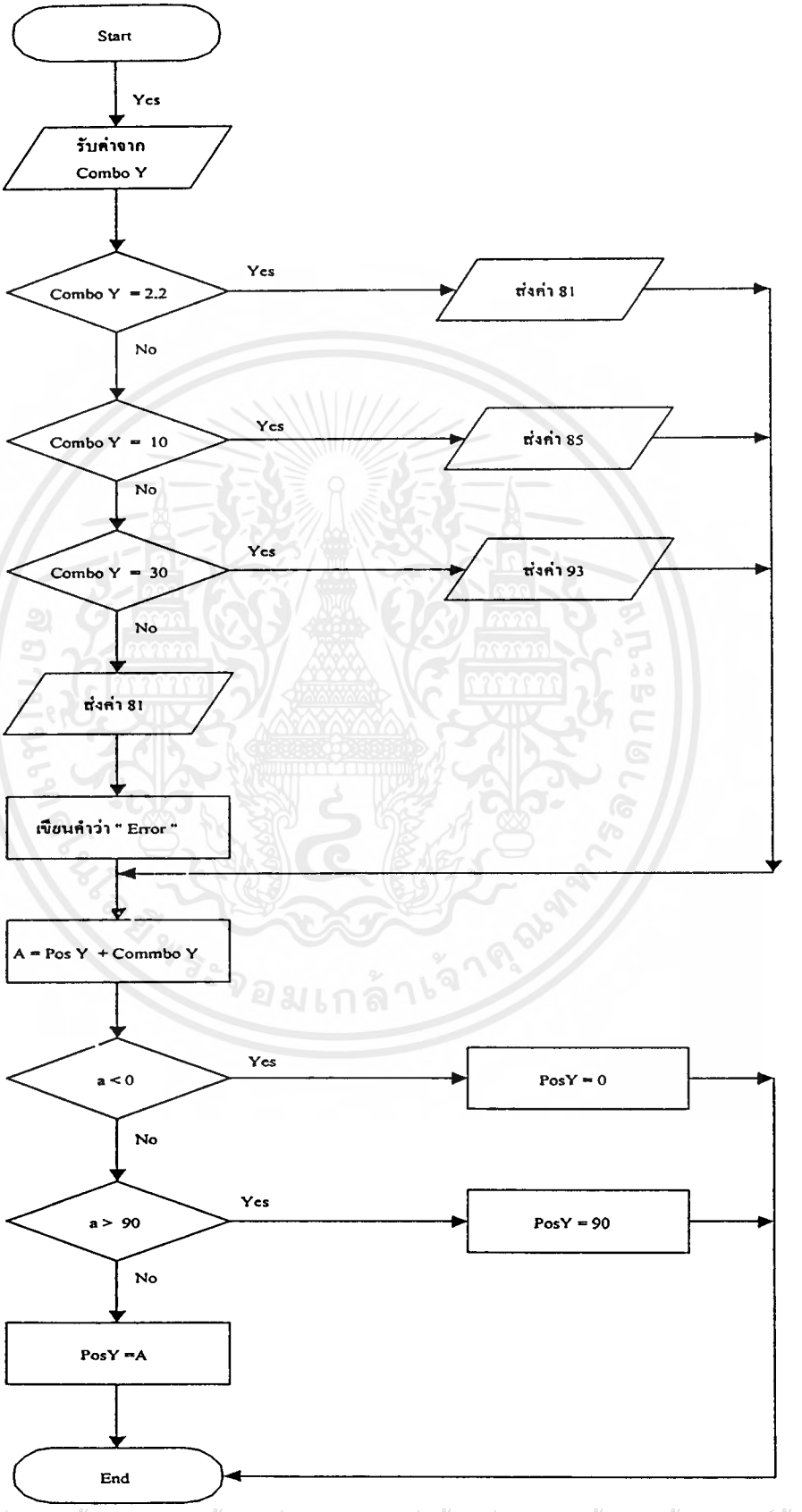
โฟลวชาร์ตแสดงการทำงานของโปรแกรม Visual Basic 6.0

1. Flow Chart ปุ่มบังคับทิศทางขึ้น



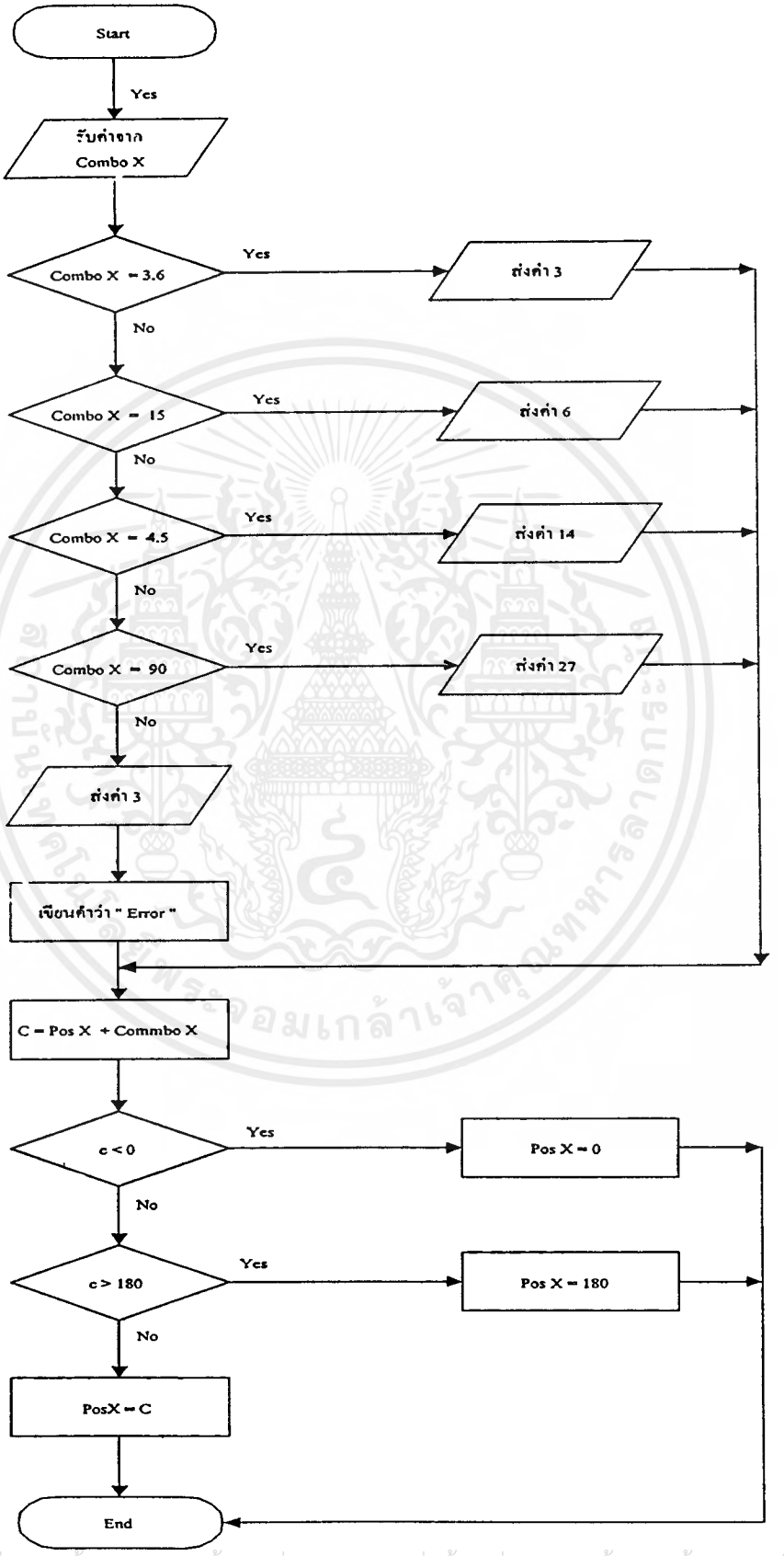
รูป Flow Chart ของปุ่มขึ้น

2 . Flow Chart ปุ่มบังคับทิศทางลง



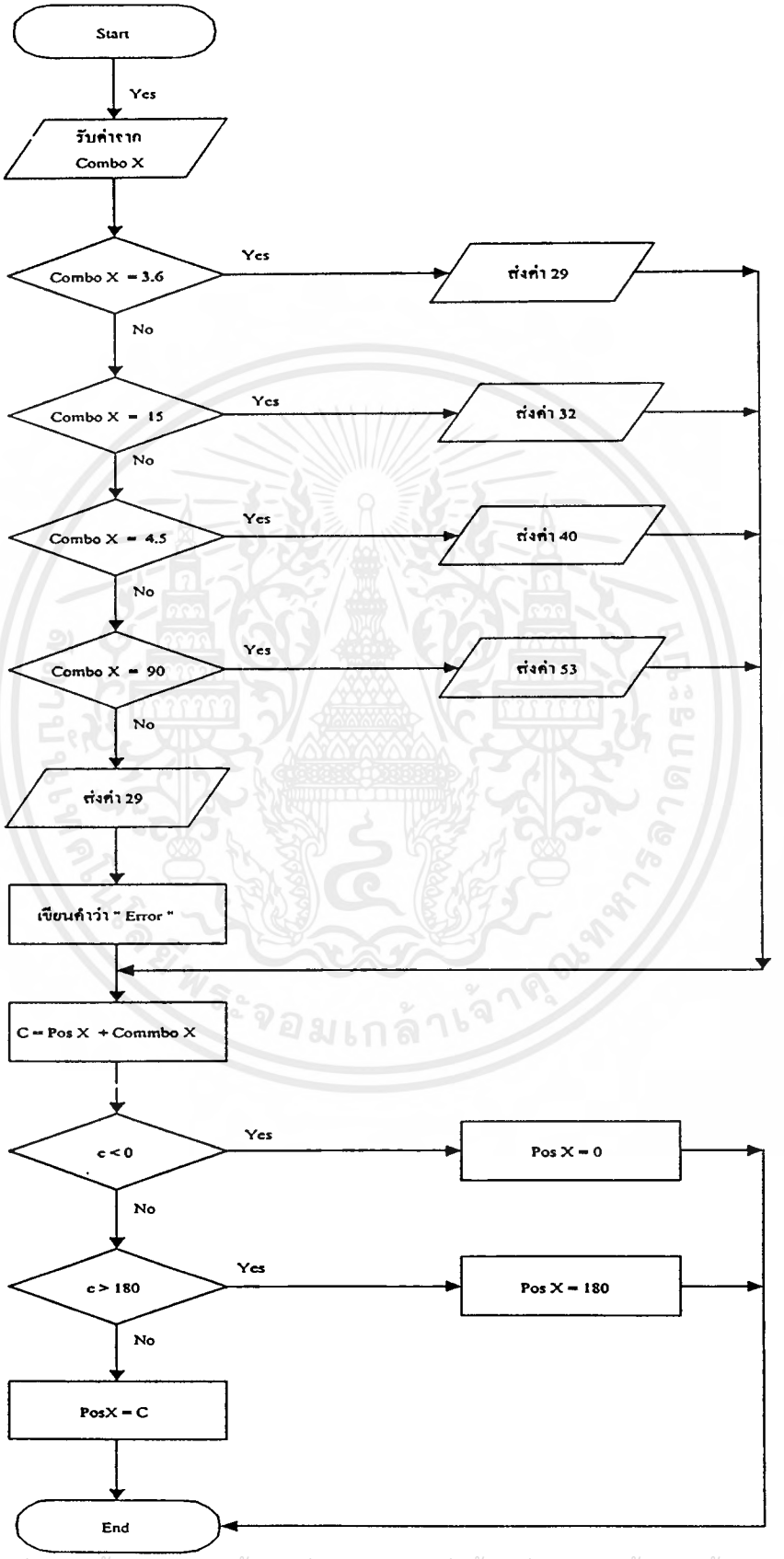
รูป Flow Chart ของปุ่มลง

3 . Flow Chart ปุ่มบังคับทิศทางขวา



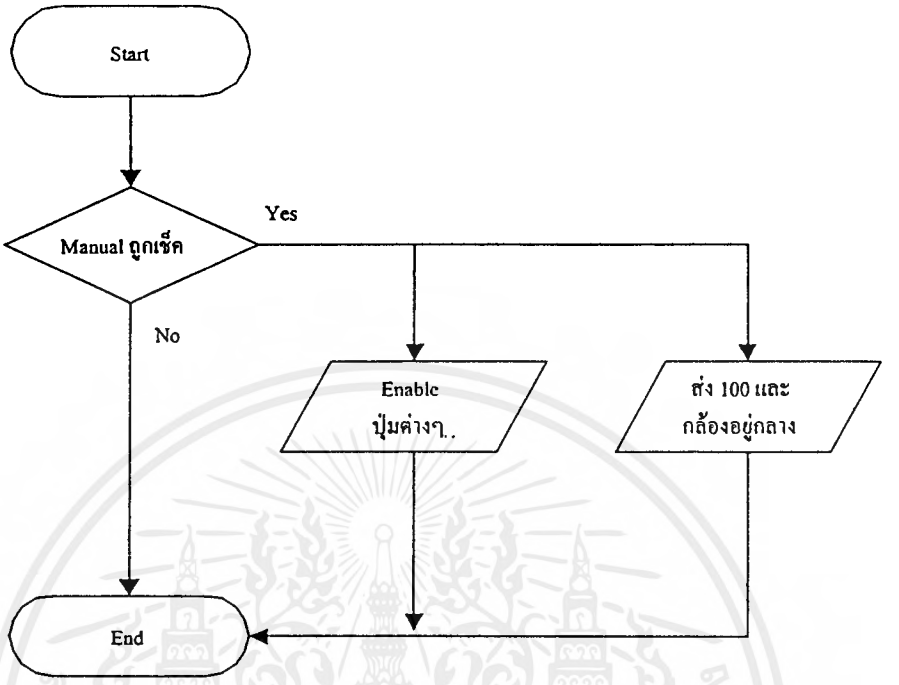
รูป Flow Chart ของปุ่มขวา

4 . Flow Chart ปุ่มบังคับทิศทางซ้าย



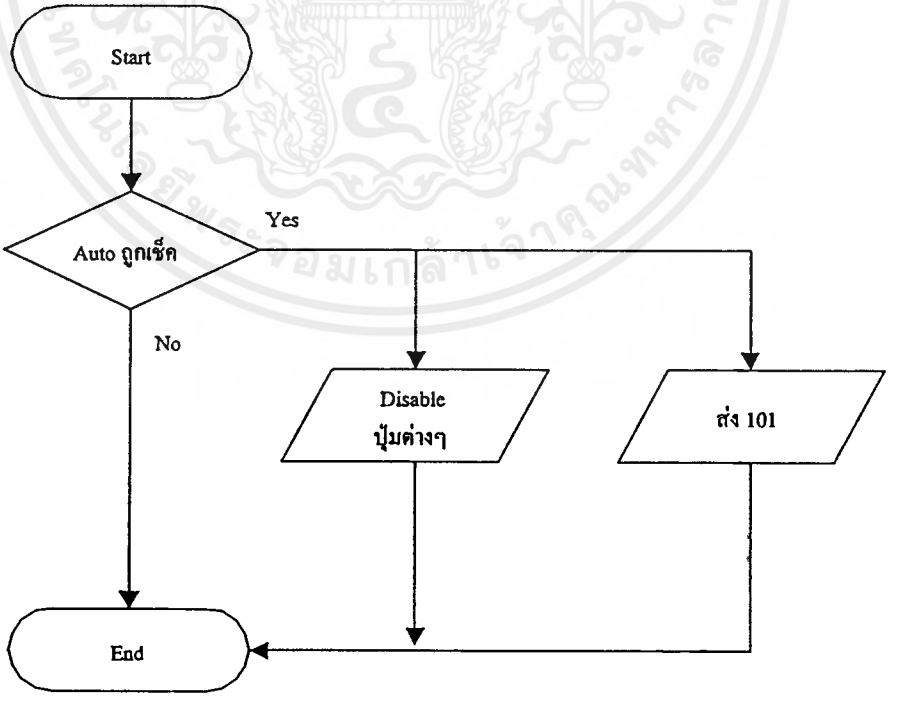
รูป Flow Chart ของปุ่มซ้าย

5 . Flow Chart ปุ่ม Manual



รูป Flow Chart ปุ่ม Manual

6 . Flow Chart ปุ่ม Automatic



รูป Flow Chart ปุ่ม Automatic

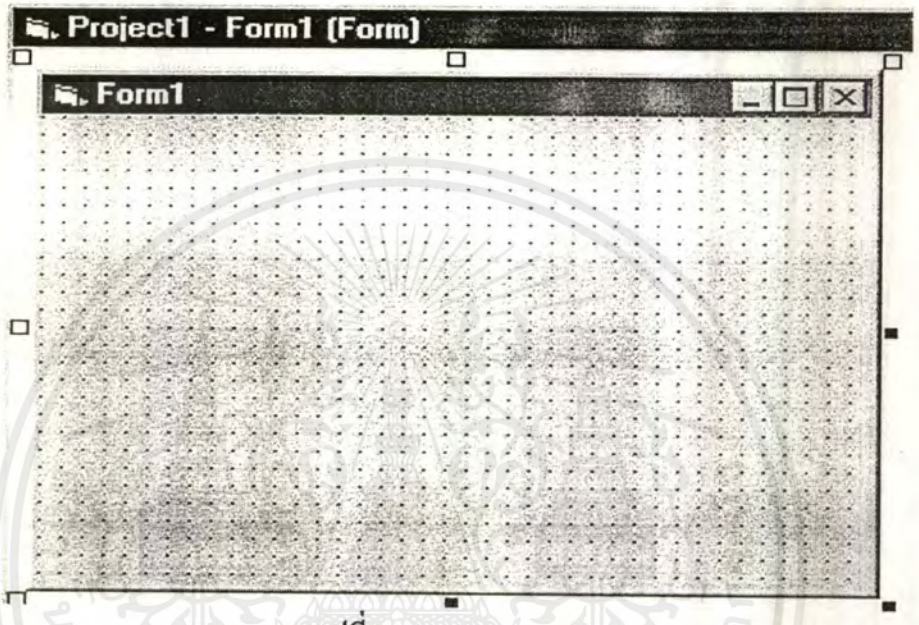
ภาคผนวก ข

ขั้นตอนในการสร้างและการเขียนโปรแกรมควบคุมการทำงาน

ภาคผนวก ข

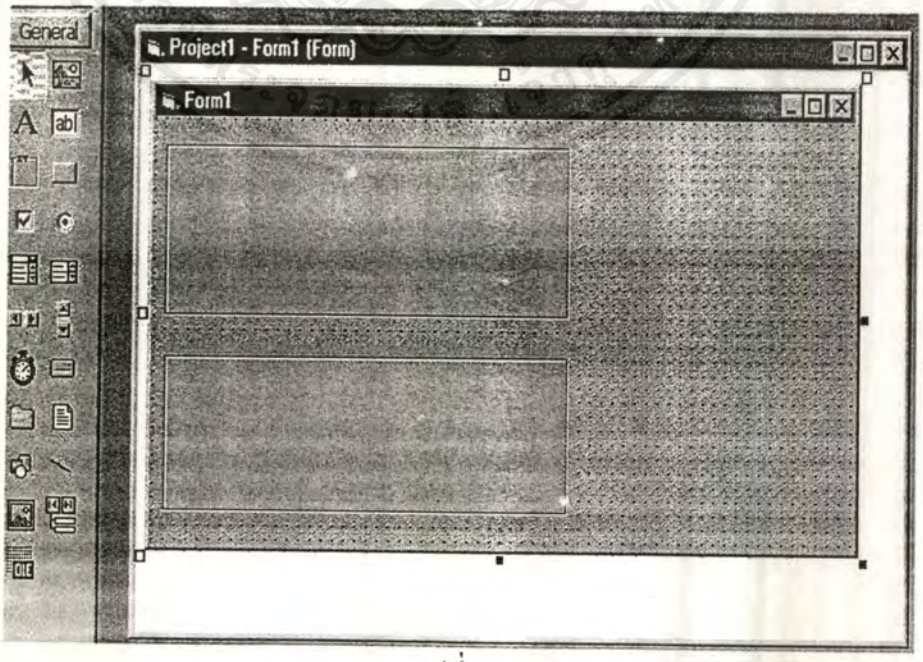
ขั้นตอนในการสร้างและการเขียนโปรแกรมควบคุมการทำงาน

1) หลังจากที่เปิด Program Visual Basic มาแล้วก็จะพบกับ Form ที่จะใช้สร้างดัง รูปที่ 1



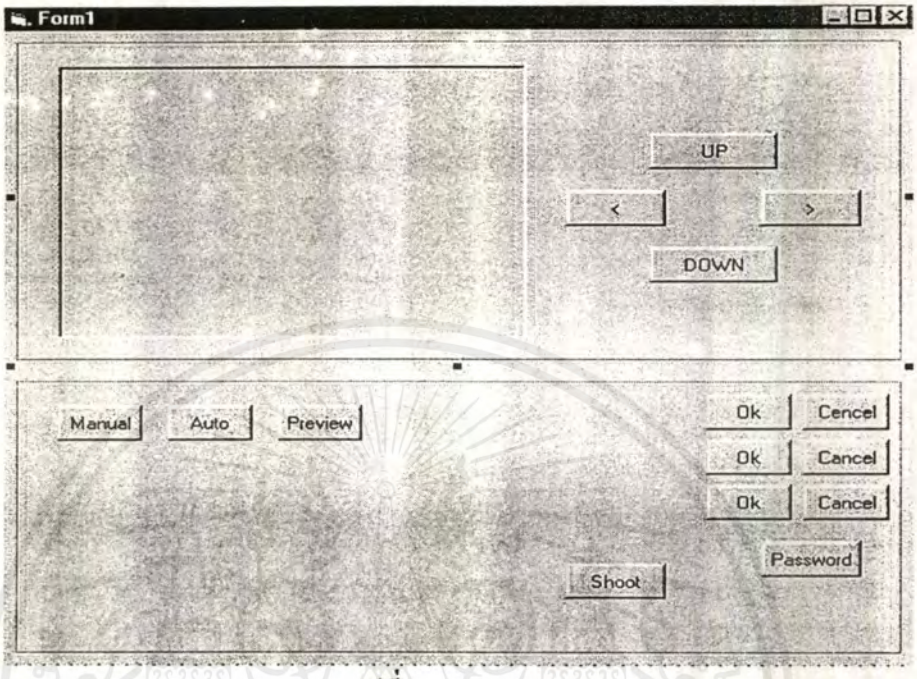
รูปที่ 1

2) หลังจากในขั้นตอนที่ 1 แล้วก็ทำการ Double Object Frame ที่อยู่ใน Tool Box และทำการจัดวาง Fraine ให้ได้ดังรูปที่ 2



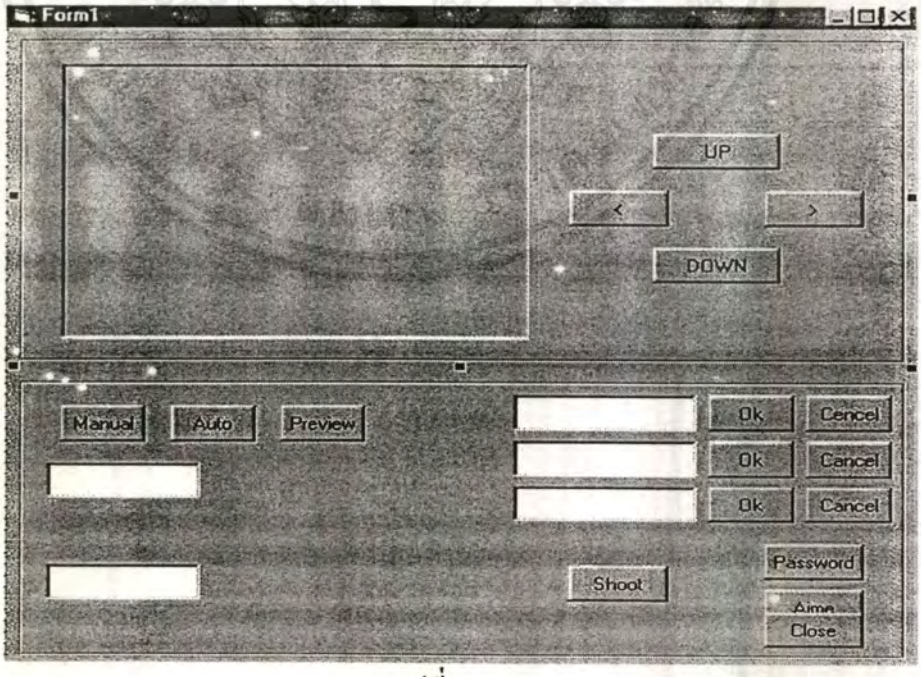
รูปที่ 2

3) ต่อจากขั้นตอนที่ 2 เราจะใช้ Object Command Bottom 17 ตัวกับ Object Picture Box อีกหนึ่งตัว และจะทำการจัดวาง Object ต่างๆ ดังรูปที่ 3 รวมถึงเปลี่ยน Properties Caption ให้ได้ดังรูปที่ 3



รูปที่ 3

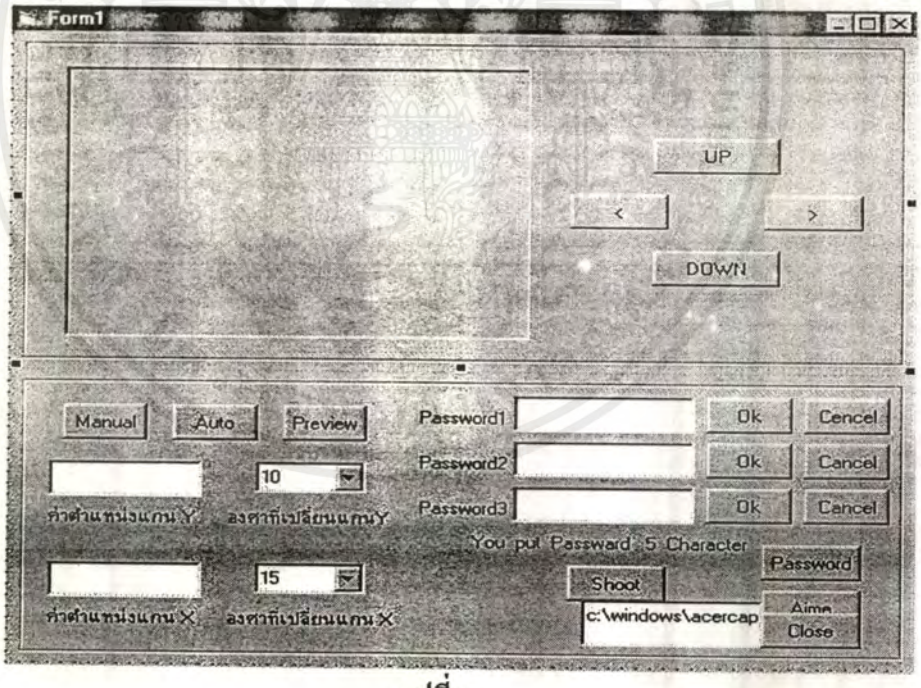
3) จากขั้นตอนที่ 2 ขั้นในตอนที่ 3 เราจะใช้ Object Text Box 5 ตัวพร้อมกับตั้ง Properties Text เป็นที่ว่าง คือไม่ใส่ข้อความใดๆเลยดังรูปที่ 4 และรูปที่ 5



รูปที่ 4

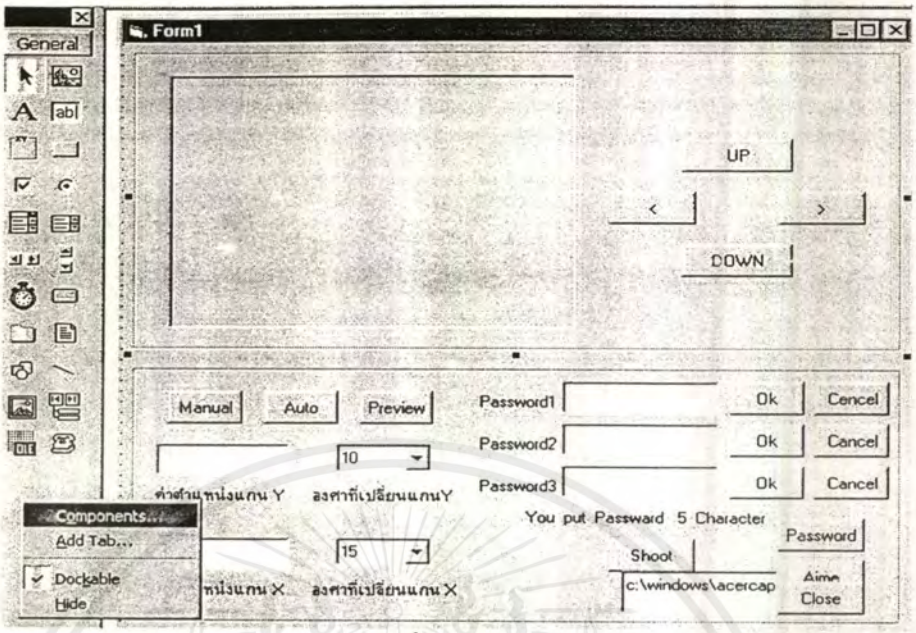
รูปที่ 5

4) จากนั้นในขั้นตอนที่ 4 นี้เราจะใช้ Object Label 7 ตัวและ Object Combobox 2 ตัวและจัดดังรูปที่ 6

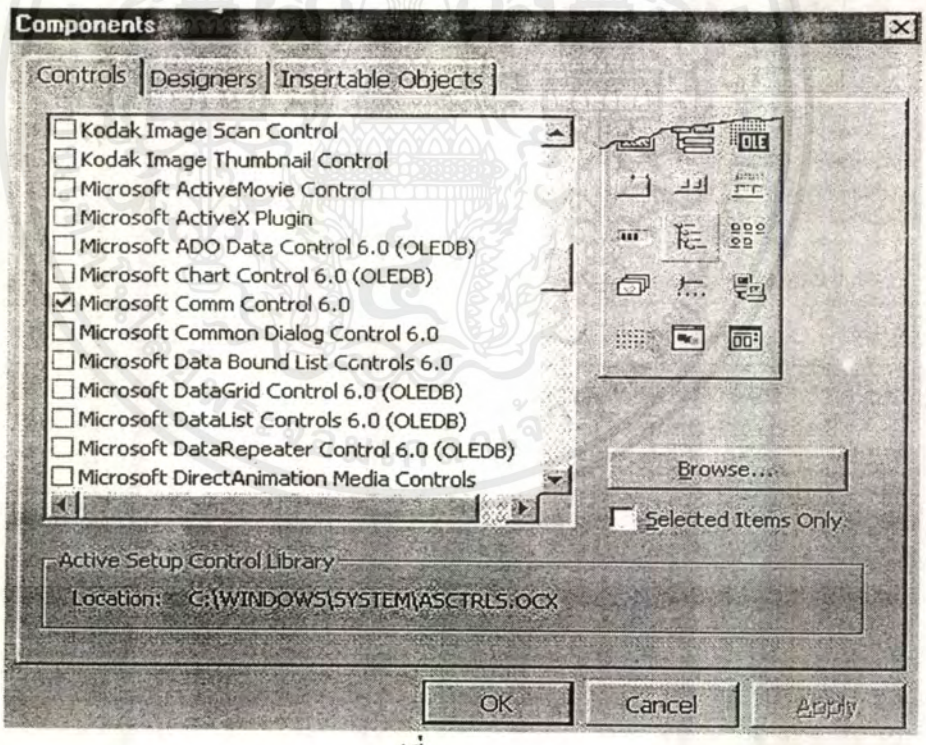


รูปที่ 6

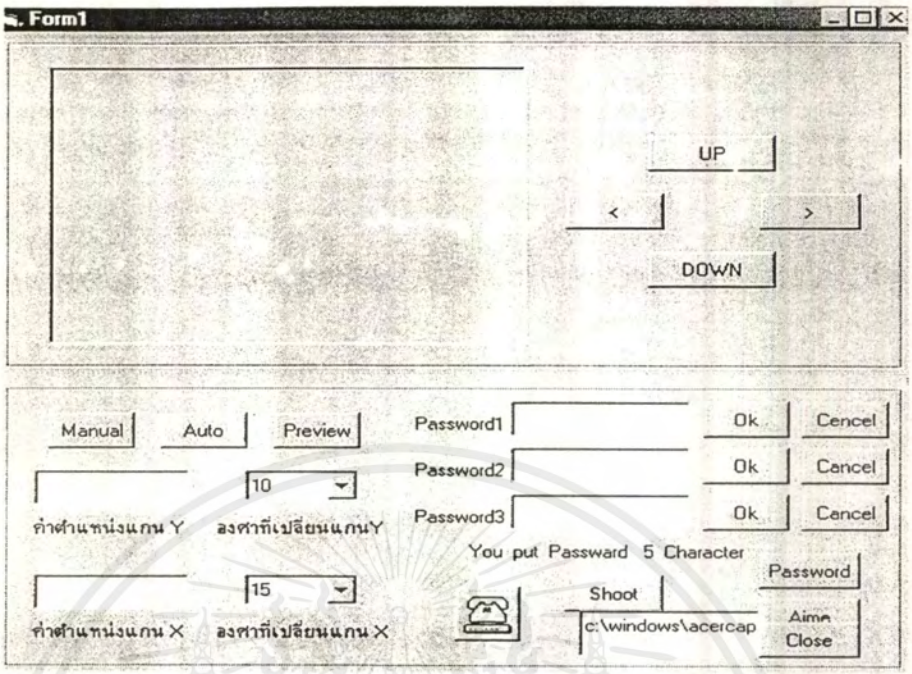
5) ขั้นและตอนสุดท้ายในการสร้างคือ Click ขวาที่บน Tool Box แล้วเลือก Component เพื่อที่จะเลือกใช้ Object Mscomm Control จากนั้น Click ที่ปุ่ม OK ก็จะมีรูปโทรศัพท์หรือ Mscomm Control เพิ่มเข้ามาอยู่บน Tool Box เราก็ทำการเลือกลงบน From เหมือนเดิม ดังรูปที่ 7 , รูปที่ 8 และรูปที่ 9 ในขั้นตอนต่อไปก็จะเขียนโปรแกรมควบคุมการทำงาน



รูปที่ 7



รูปที่ 8



รูปที่ 9

6) ขั้นตอนที่ 6 นี้จะเป็นขั้นตอนในการเขียนโปรแกรมควบคุมการทำงาน

Program ควบคุมการทำงาน

```
Option Explicit

Private Const GW_HWNDNEXT = 2

Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd As Long,
lpdwProcessId As Long) As Long
Private Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As Long,
ByVal lpWindowName As Long) As Long
Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal hwnd As Long,
ByVal lpString As String, ByVal cch As Long) As Long
Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long, ByVal hWndNewParent As
Long) As Long

Private old_parent As Long
Private child_hwnd As Long

' Return the window handle for an instance handle.
Private Function InstanceToWnd(ByVal target_pid As Long) As Long
Dim test_hwnd As Long
Dim test_pid As Long
Dim test_thread_id As Long

  ' Get the first window handle.
  test_hwnd = FindWindow(ByVal 0&, ByVal 0&)

  ' Loop until we find the target or we run out
  ' of windows.
  Do While test_hwnd <> 0
```

```
        ' See if this window has a parent. If not,
        ' it is a top-level window.
        If GetParent(test_hwnd) = 0 Then
            ' This is a top-level window. See if
            ' it has the target instance handle.
            test_thread_id = GetWindowThreadProcessId(test_hwnd, test_pid)

            If test_pid = target_pid Then
                ' This is the target.
                InstanceToWnd = test_hwnd
                Exit Do
            End If
        End If

        ' Examine the next window.
        test_hwnd = GetWindow(test_hwnd, GW_HWNDNEXT)
    Loop
End Function


Private Sub aimedc_Click()                  ' Click ปุ่ม Aime '
Dim aimc As Integer                         ' ประกาศตัวแปร aimc '
    aimc = 105
    MSComm1.Output = Chr$(aimc)             ' แปลง aimc เป็นตัวอักษรแล้วส่งออก Port RS-232 '
aimedc.Visible = False
aimedo.Visible = True
End Sub

Private Sub aimedo_Click()                  ' Click ปุ่ม Close '
Dim aimo As Integer                         ' ประกาศตัวแปร aimo '
    aimo = 104
    MSComm1.Output = Chr$(aimo)             ' แปลง aimo เป็นตัวอักษรแล้วส่งออก Port RS-232 '
    aimedo.Visible = False
    aimedc.Visible = True
```

```
End Sub

Private Sub Comcan1_Click()                'ปุ่มยกเลิกปุ่มที่1'
    Label4.Visible = False
    Label7.Visible = False
    Text4.Visible = False
    Comok1.Visible = False
    Comcan1.Visible = False
    Compwd.Enabled = True
End Sub

Private Sub Comcan2_Click()                'ปุ่มยกเลิกปุ่มที่2'
    Label4.Visible = False
    Label7.Visible = False
    Text4.Visible = False
    Comok1.Visible = False
    Comcan1.Visible = False
    Label5.Visible = False
    Text5.Visible = False
    Comok2.Visible = False
    Comcan2.Visible = False
    Compwd.Enabled = True
End Sub

Private Sub Comcan3_Click()                'ปุ่มยกเลิกปุ่มที่3'
    Label4.Visible = False
    Label7.Visible = False
    Text4.Visible = False
    Comok1.Visible = False
    Comcan1.Visible = False
    Label5.Visible = False
    Text5.Visible = False
    Comok2.Visible = False
    Comcan2.Visible = False
```

```
    Compwd.Enabled = True
    Label6.Visible = False
    Text6.Visible = False
    Comok3.Visible = False
    Comcan3.Visible = False
End Sub

Private Sub ComPrew_Click()                ' ปุ่ม Preview '
Dim pid As Long                            ' ประกาศตัวแปร Pid '
Dim buf As String                          ' ประกาศตัวแปร buf '
Dim buf_len As Long                        ' ประกาศตัวแปร buf_len '
        ' Start the program
pid = Shell(Text1.Text, vbNormalFocus)
   If pid = 0 The
      MsgBox "Error starting program"      ' MsgBox แสดงข้อความ Error '
      Exit Sub
   End If
        ' Get the window handle.
   child_hwnd = InstanceToWnd(pid)
        ' Reparent the program so it lies inside
        ' the PictureBox.
   old_parent = SetParent(child_hwnd, Picchild.hwnd)
End Sub

Private Sub Comok1_Click()                 ' ปุ่มตกลงที่1 '
If Text4.Text = "12345" Then
   Label4.Visible = False
   Text4.Visible = False
   Label5.Visible = True
   Text5.Visible = True
   Comok2.Visible = True
   Comcan2.Visible = True
   Comok1.Visible = False
   Comcan1.Visible = False
```

```
Else
   MsgBox "รหัสไม่ถูกต้อง กรุณาใส่อีกครั้งหนึ่ง", vbOKOnly
End If
End Sub

Private Sub Comok2_Click()                    ' ปุ่มตกลงที่2 '
If Text5.Text = "12345" Then
   Label5.Visible = False
   Label6.Visible = True
   Text5.Visible = False
   Text6.Visible = True
   Comok3.Visible = True
   Comcan3.Visible = True
   Comok2.Visible = False
   Comcan2.Visible = False
Else
   MsgBox "รหัสไม่ถูกต้อง กรุณาใส่อีกครั้งหนึ่ง", vbOKOnly
End If
End Sub

Private Sub Comok3_Click()
Dim t As Integer                              ' ปุ่มตกลงที่3 '
If Text6.Text = "12345" Then
   Comshot.Visible = True
   Comok2.Visible = False
   Comcan2.Visible = False
   Comok3.Visible = False
   Comcan3.Visible = False
   Label4.Visible = False
   Label5.Visible = False
   Label6.Visible = False
   Label7.Visible = False
   Text4.Visible = False
   Text5.Visible = False
```

```
    Text6.Visible = False

t = 102                                    ' ส่งสัญญาณเตือน '
    'MSComm1.Output = Cnr$(t)
Else
    MsgBox "รหัสไม่ถูกต้อง กรุณาใส่อีกครั้งหนึ่ง", vbOKOnly
End If
End Sub

Private Sub Compwd_Click()                 ' ปุ่มใส่ password '
    Text4.Text = ""
    Text5.Text = ""
    Text6.Text = ""
    Compwd.Enabled = False
    Label4.Visible = True
    Label7.Visible = True
    Text4.Visible = True
    Comok1.Visible = True
    Comcan1.Visible = True
End Sub

Private Sub ComUp_Click()                  ' เป็นการส่ง code เมื่อเรากดปุ่มขึ้น '
    Dim a As Single
    Dim aa As Single
ComUp.Enabled = True
Select Case Val(Combo2.Text)               ' เลือกค่าใน ComboBox '
    Case 2.25
        aa = 55
        MSComm1.Output = Chr$(aa)
    Case 10
        aa = 59
        MSComm1.Output = Chr$(aa)
    Case 30
        aa = 67
```

```
        MSComm1.Output = Chr$(aa)
    Case Else
        MsgBox "You sent error. You put again ", vbOKOnly
End Select
    Text2.Text = Val(Text2.Text) + Val(Combo2.Text)          ' ตรวจสอบว่าค่าที่ได้อยู่ในช่วงหรือเปล่า
        a = Text2.Text
    If a < 0 Then
        MsgBox "You Click Again ", vbOKOnly                  ' แสดงข้อความว่า Click อีก ครั้ง '
        Text2.Text = 0
    ElseIf a > 90 Then
        MsgBox "You Click Again ", vbOKOnly
        Text2.Text = 90
    Else
        Text2.Text = a
    End If
End Sub

Private Sub ComDown_Click()                              ' เป็นการส่ง code เมื่อเรากด ComDown '
    Dim aa As Single
    Dim a As Single
    ComDown.Enabled = True
Select Case Val(Combo2.Text)
    Case 2.25
        aa = 81
        MSComm1.Output = Chr$(aa)
    Case 10
        aa = 85
        MSComm1.Output = Chr$(aa)
    Case 30
        aa = 93
        MSComm1.Output = Chr$(aa)
    Case Else
        MsgBox "You sent error. You put again ", vbOKOnly
End Select
```

```
    Text2.Text = Val(Text2.Text) - Val(Combo2.Text)          ' ตรวจสอบว่าค่าที่ได้อยู่ในช่วงหรือเปล่า '
    a = Text2.Text
      If a < 0 Then
        MsgBox "You Click Again", vbOKOnly
        Text2.Text = 0
      ElseIf a > 90 Then
        MsgBox "You Click Again", vbOKOnly
        Text2.Text = 90
      Else
        Text2.Text = a
      End If
End Sub

Private Sub ComRight_Click()                    ' เป็นการส่ง code เมื่อเรากด ComRight '
    Dim c As Single
    Dim aa As Single
    ComRight.Enabled = True
Select Case Val(Combo1.Text)
    Case 3.6
      aa = 3
      MSComm1.Output = Chr$(aa)
    Case 15
      aa = 6
      MSComm1.Output = Chr$(aa)
    Case 45
      aa = 14
      MSComm1.Output = Chr$(aa)
    Case 90
      aa = 27
      MSComm1.Output = Chr$(aa)
    Case Else
      MsgBox "You sent error. You put again ", vbOKOnly
```

```
End Select

Text3.Text = Val(Text3.Text) + Val(Combo1.Text)          ' ตรวจสอบว่าค่าที่ได้อยู่ในช่วงหรือเปล่า '
  c = Text3.Text
     If c < 0 Then
        MsgBox "You Click Again", vbOKOnly
        Text3.Text = 0
     ElseIf c > 180 Then
        MsgBox "You Click Again", vbOKOnly
        Text3.Text = 180
     Else
        Text3.Text = c
     End If
End Sub

Private Sub ComLeft_Click()                              ' เป็นการส่ง code เมื่อเรากด ComLeft '
   Dim c As Single
   Dim aa As Single
   ComLeft.Enabled = True
Select Case Val(Combo1.Text)
   Case 3.6
      aa = 29
      MSComm1.Output = Chr$(aa)
   Case 15
      aa = 32
      MSComm1.Output = Chr$(aa)
   Case 45
      aa = 40
      MSComm1.Output = Chr$(aa)
   Case 90
      aa = 53
      MSComm1.Output = Chr$(aa)
   Case Else
      MsgBox "You sent error. You put again ", vbOKOnly
```

```
End Select

Text3.Text = Val(Text3.Text) - Val(Combo1.Text)          ' ตรวจสอบว่าค่าที่ได้อยู่ในช่วงหรือเปล่า '
  c = Text3.Text
    If c < 0 Then
      MsgBox "You Click Again", vbOKOnly
      Text3.Text = 0
    ElseIf c > 180 Then
      MsgBox "You Click Again", vbOKOnly
      Text3.Text = 180
    Else
      Text3.Text = c
    End If
End Sub

Private Sub ComAuto_Click()                              ' เป็นการส่ง code เมื่อเรากด ComAuto '
  Dim auto As Integer
  auto = 101
  MSComm1.Output = Chr$(auto)
  Text2.Enabled = False                                  ' เป็นการงดการบังคับทั้งหมด '
  Text3.Enabled = False
  Combo1.Enabled = False
  Combo2.Enabled = False
  ComUp.Enabled = False                                  ' เป็นการตั้งปุ่มให้เป็น False '
  ComDown.Enabled = False
  ComRight.Enabled = False
  ComLeft.Enabled = False
  ComMan.Enabled = True
  ComAuto.Enabled = False
  Text2.Text = ""                                        ' ตั้งค่าแกน X และ Y '
  Text3.Text = ""
End Sub

Private Sub ComMan_Click()                               ' เป็นการส่ง code เมื่อเรากด Manual'
```

```
    Dim ma As Integer

    ma = 100
    MSComm1.Output = Chr$(ma)
    Text2.Enabled = True                    ' เป็นการเริ่มใช้ปุ่มบังคับทั้งหมด '
    Text3.Enabled = True
    Label1.Enabled = True
    Combo1.Enabled = True
    Combo2.Enabled = True
     ComUp.Enabled = True                   ' เป็นการตั้งปุ่มให้เป็น True '
    ComDown.Enabled = True
    ComRight.Enabled = True
    ComLeft.Enabled = True
    ComMan.Enabled = False
    ComAuto.Enabled = True
    Text2.Text = 45                         ' ตั้งให้กล้องอยู่ตำแหน่งกลาง '
    Text3.Text = 90
End Sub

Private Sub ComShot_Click()
    Dim shot As Integer
    shot = 103                              ' เป็นการสั่งทำการยิง '
 MSComm1.Output = Chr$(shot)
    Label4.Visible = False
    Label7.Visible = False
    Text4.Visible = False
    Label5.Visible = False
    Text5.Visible = False
    Label6.Visible = False
    Text6.Visible = False
    Comshot.Visible = False
    Compwd.Enabled = True
End Sub
```

```
Private Sub form_load()                          ' เมื่อเปิด Form ขึ้นมาจะทำการโหลดค่าเหล่านี้ '
    Dim a As Byte
'Dim X As Single
'Dim Y As Single
    MSComm1.CommPort = 2
    MSComm1.PortOpen = True
    ComUp.Enabled = True
    ComDown.Enabled = True
    ComRight.Enabled = True
    ComLeft.Enabled = True
    ComAuto.Enabled = True
    ComMan.Enabled = True
    Comshot.Enabled = True

    Combo1.AddItem "3.6"
    Combo1.AddItem "15"
    Combo1.AddItem "45"
    Combo1.AddItem "90"

    Combo2.AddItem "2.25"
    Combo2.AddItem "10"
    Combo2.AddItem "30"
     Text2.Enabled = False                       ' เป็นการงดการบังคับทั้งหมด '
    Text3.Enabled = False
    Combo1.Enabled = False
    Combo2.Enabled = False
    ComUp.Enabled = False                        ' เป็นการตั้งปุ่มให้เป็น False '
    ComDown.Enabled = False
    ComRight.Enabled = False
    ComLeft.Enabled = False
    ComMan.Enabled = True
    ComAuto.Enabled = False
'MSComm1.InBufferCount = 2                       ' รับค่ามาจาก port '
'MSComm1.InputMode = comInputModeBinary
```

```
'a = Val(MSComm1.Input)
 ' Text3.Text = a
'If a >= 64 And a <= 119 Then
 '  X = a - 64
  ' X = X * 3.272727
   ' Text3.Text = Val(X)
'ElseIf a >= 128 And a <= 153 Then
 '  Y = a - 128
  ' Y = Y * 3.6
   ' Text2.Text = Val(Y)
'Else
 '  MsgBox "error", vbOKOnly
'End If
End Sub

Private Sub Picchild_Click()
End Sub
```

ภาคผนวก ค

โปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์

```
dirL = %11111111
dirH = %11110000
ting var byte
ting = 20
ting1 var byte
ting1 = 25
j var byte
j = 0
a1 var byte
a1 = 0
d1 var byte
d1 = 0
d2 var byte
d2 = 0
d3 var byte
d3 = 0
d4 var byte
d4 = 0
out14 = 0

pause 1000
pause 1000
pause 1000
pause 1000
pause 1000
```

```
sent:
a1 = 0
if j=0 then auto1
serin 16\12,16468,19,sent,[j]

if j=101 then auto2
if j>2 and j<28 then left1
if j>28 and j<54 then right1
if j>54 and j<68 then up1
if j>80 and j<94 then down1
if j=103 then shot1
if j=104 then point1
if j=105 then point2
goto sent

left1:
a1 = j-2
left2:
if ting1 = 50 then sent
if a1 = 0 then sent

outa = %1001
pause 7
outa = %1000
pause 7
outa = %1100
pause 7
outa = %0100
pause 7
outa = %0110
pause 7
```

```
outa = %0010
pause 7
outa = %0011
pause 7
outa = %0001
pause 7
ting1 = ting1+1
a1 = a1-1
goto left2

right1:
a1 = j-28
right2:
if ting1 = 0 then sent
if a1 = 0 then sent

outa = %0001
pause 7
outa = %0011
pause 7
outa = %0010
pause 7
outa = %0110
pause 7
outa = %0100
pause 7
outa = %1100
pause 7
outa = %1000
pause 7
outa = %1001
pause 7
```

```
ting1 = ting1-1
a1 = a1-1
goto right2

up1:
a1 = j-54
up2:
if ting = 40 then sent
if a1 = 0 then sent

outb = %1001
pause 7
outb = %1000
pause 7
outb = %1100
pause 7
outb = %0100
pause 7
outb = %0110
pause 7
outb = %0010
pause 7
outb = %0011
pause 7
outb = %0001
pause 7
ting = ting+1
a1 = a1-1

goto up2
```

```
down1:
a1 = j-80
down2:
if ting = 0 then sent
if a1 = 0 then sent

outb = %0001
pause 7
outb = %0011
pause 7
outb = %0010
pause 7
outb = %0110
pause 7
outb = %0100
pause 7
outb = %1100
pause 7
outb = %1000
pause 7
outb = %1001
pause 7

ting = ting-1
a1 = a1-1
goto down2

point1:
out14 = 1
goto sent
```

```
point2:
out14 = 0
goto sent

shot1:
out13 = 1
pause 100
out13 = 0
goto sent



auto1:
serin 16\12,16468,19,sent1,[j]
sent1:
if j=100 then sent2
pause 10
l1:
d1 = 0
button 10,1,0,100,d1,1,s1
pause 10
l2:
d2 = 0
button 11,1,0,100,d2,1,s2
l3:
d3 = 0
l4:
d4 = 0
goto auto1
```

```
s1:
if ting1 = 50 then l1
ting1 = ting1+1

outa = %1001
pause 7
outa = %1000
pause 7
outa = %1100
pause 7
outa = %0100
pause 7
outa = %0110
pause 7
outa = %0010
pause 7
outa = %0011
pause 7
outa = %0001
pause 7
goto l1

s2:
if ting1 = 0 then l2
ting1 = ting1-1
outa = %0001
pause 7
outa = %0011
pause 7
outa = %0010
pause 7
outa = %0110
```

```
pause 7
outa = %0100
pause 7
outa = %1100
pause 7
outa = %1000
pause 7
outa = %1001
pause 7
goto l2

s3:
if ting1 = 50 then l2
ting1 = ting1+1

outa = %1001
pause 7
outa = %1000
pause 7
outa = %1100
pause 7
outa = %0100
pause 7
outa = %0110
pause 7
outa = %0010
pause 7
outa = %0011
pause 7
outa = %0001
pause 7
goto s3
```

```
s4:
if ting1 = 0 then l1
ting1 = ting1-1
outa = %0001
pause 7
outa = %0011
pause 7
outa = %0010
pause 7
outa = %0110
pause 7
outa = %0100
pause 7
outa = %1100
pause 7
outa = %1000
pause 7
outa = %1001
pause 7
goto s4

sent2:
if ting1=25 then sent
if ting1<25 then re1
if ting1>25 then re2
goto sent2

re1:
if ting1 = 25 then sent
ting1 = ting1+1
```

```
outa = %1001
pause 7
outa = %1000
pause 7
outa = %1100
pause 7
outa = %0100
pause 7
outa = %0110
pause 7
outa = %0010
pause 7
outa = %0011
pause 7
outa = %0001
pause 7
goto re1

re2:
if ting1 = 25 then sent
ting1 = ting1-1
outa = %0001
pause 7
outa = %0011
pause 7
outa = %0010
pause 7
outa = %0110
pause 7
outa = %0100
pause 7
outa = %1100
```

```
pause 7
outa = %1000
pause 7
outa = %1001
pause 7
goto re2

auto2:
if ting1=25 then auto3
if ting1<25 then re01
if ting1>25 then re02

re01:
if ting1 = 25 then auto3
ting1 = ting1+1
outa = %1001
pause 7
outa = %1000
pause 7
outa = %1100
pause 7
outa = %0100
pause 7
outa = %0110
pause 7
outa = %0010
pause 7
outa = %0011
pause 7
outa = %0001
pause 7
goto re01
```

```
re02:
if ting1 = 25 then auto3
ting1 = ting1-1
outa = %0001
pause 7
outa = %0011
pause 7
outa = %0010
pause 7
outa = %0110
pause 7
outa = %0100
pause 7
outa = %1100
pause 7
outa = %1000
pause 7
outa = %1001
pause 7
goto re02

auto3:
if ting=20 then auto1
if ting<20 then re001
if ting>20 then re002

re001:
if ting = 20 then auto1
ting = ting+1

outb = %1001
pause 7
```

```
outb = %1000
pause 7
outb = %1100
pause 7
outb = %0100
pause 7
outb = %0110
pause 7
outb = %0010
pause 7
outb = %0011
pause 7
outb = %0001
pause 7
goto re001

re002:
if ting = 20 then auto1
ting = ting-1

outb = %0001
pause 7
outb = %0011
pause 7
outb = %0010
pause 7
outb = %0110
pause 7
outb = %0100
pause 7
outb = %1100
pause 7
```

```
outb = %1000
pause 7
outb = %1001
pause 7
goto re002
```