รถสำรวจ

SURVEY CAR

โดย

นาย กัมปนาท กัปโก

นาย ปรีชา ขาวสะอาด

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

รถสำรวจ

SURVEY CAR

โดย

นาย กัมปนาท กัปโก รหัส 46010030

นาย ปรีชา ขาวสะอาด รหัส 46010431

อาจารย์ที่ปรึกษา

รศ.ดร. สุริภณ สมควรพาณิชย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ ปีการศึกษา 2549

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง รถสำรวจ

ผู้จัดทำ

1. นาย กัมปนาท กัปโก รหัส 46010030
2. นาย ปรีชา ขาวสะอาด รหัส 46010431

.....อาจารย์ที่ปรึกษา

(รศ.ดร. สุริภณ สมควรพาณิชย์)

รถสำรวจ

นาย กัมปนาท กัปโก รหัส 46010030
นาย ปรีชา ขาวสะอาด รหัส 46010431
รศ.ดร.สุริภณ สมควรพาณิชย์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

โครงงานนี้เป็นการออกแบบการควบคุมรถสำรวจแบบไร้สายจากเครื่องคอมพิวเตอร์ โดยใช้โปรแกรมควบคุมรถที่เขียนจากภาษาวิชวลเบสิก6 คำสั่งจะถูกส่งออกไปทางพอร์ตอนุกรม และแปลงแรงดันเป็นระดับ TTL ด้วยไอซีMAX232 จากนั้นส่งออกอากาศแบบสัญญาณวิทยุด้วย Transmitter ที่ตัวรถมี Receiver รับสัญญาณ RF แล้วส่งสัญญาณ TTL ไปยังไมโครคอนโทรลเลอร์ซึ่งเป็นตัวควบคุมรถให้ทำตามคำสั่ง การควบคุมทิศทางของรถมาจากมอเตอร์กระแสตรง 2 ตัว ที่ล้อหน้า การควบคุมกล้องใช้มอเตอร์สเตปเพื่อให้กล้องหมุนขึ้น ลงได้ ตัวกล้องมีตัวส่งสัญญาณภาพอยู่ภายใน ที่เครื่องคอมพิวเตอร์จะมีส่วนรับสัญญาณภาพทำให้สามารถดูภาพได้จากโปรแกรมควบคุมแบบ realtime และสามารถบันทึกภาพเป็นไฟล์ .avi ได้ และมี Optical encoder เพื่อใช้รอบการหมุนของล้อ เซนเซอร์สนามแม่เหล็กโลกใช้วัดทิศทางของรถ เซนเซอร์อุณหภูมิ ที่ตัวรถ ซึ่งใช้ไมโครคอนโทรลเลอร์ในการติดต่อ และส่งข้อมูลกลับมาที่คอมพิวเตอร์ เพื่อแสดงเส้นทางการวิ่งของรถ และอุณหภูมิรอบตัวรถ

SURVEY CAR

Mr. Kampanath Kappago ID.46010030

Mr. Presha kawsaard ID.46010431

Assoc. Prof. Dr. Suripon Somkuarnpanit Advisor

Education Year 2006

Abstract

This project is wireless control design for survey car from computer .Control program builds from visual basic language. It transmits control data from serial port.IC max232 change data to TTL level. Data is modulated and transmit in air. At car receiver change RF signal to TTL data to microcontroller for control survey car. Direction control of car from two DCmotor at front wheel. Direction control of camera from step motor.Inside of camea has AV signal transmitter. At computer has AV signal receiver. Control program has realtime video monitor from camera. That can record to .avi file.There are optical encoders,compass sensor,temperature sensor.Microcontroller receives data from sensor and tranducer and send data to computer

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีโดยได้รับความอนุเคราะห์ ช่วยเหลือ และกำลังใจ จากบุคคลหลายๆท่าน จึงขอขอบพระคุณและระลึกถึงไว้ ณ ที่นี้

รศ.ดร. สุริภณ สมควรพาณิชย์ ผู้ให้โอกาสในการทำรายงานครั้งนี้ ให้คำปรึกษาและช่วยเหลือในการทำโครงงาน

ท่านอาจารย์ทุกท่านที่สั่งสอนวิชาต่างๆ ซึ่งนำมาใช้ในโครงงานนี้ พ่อแม่ ที่คอยให้กำลังใจและการสนับสนุน เพื่อนทุกคนที่ให้คำแนะนำ

นาย กัมปนาท กัปโก

นาย ปรีชา ขาวสะอาด

สารบัญ

สารบัญรูป

รูป **หน้า**

สารบัญตาราง

บทที่1
บทนำ

ในปัจจุบันมีหุ่นยนต์จำนวนมากได้ใช้งานอยู่ในอุตสาหกรรม ซึ่งมันจะช่วยหลีกเลี่ยงการเสี่ยงภัยของมนุษย์ต่องานที่อาจเป็นอันตราย เช่น พื้นที่ที่มีความดันหรือความร้อนสูง ในทีที่ไม่มีอากาศ พื้นที่แคบๆ หุ่นยนต์เหล่านี้อาจจะบังคับโดยคนจากคอมพิวเตอร์ หรือมีการโปรแกรมให้มันทำงานแบบอัตโนมัติได้ เช่น หุ่นยนต์ขนถ่ายสินค้า ซึ่งสามารถเคลื่อนจากที่หนึ่งไปอีกที่หนึ่งในเส้นทางเดิมๆและขนถ่ายสินค้าอย่างอัตโนมัติ ด้วยการโปรแกรมที่แม่นยำ

หุ่นยนต์สำรวจดาวอังคารของนาซ่าเป็นตัวอย่างที่ดีเยี่ยมของหุ่นยนต์สำรวจ การเก็บข้อมูลภาพและเสียง อุณหภูมิ ความดัน ตัวอย่างพื้นผิว การเคลื่อนที่ที่มีความยืดหยุ่นสูงต่อพื้นผิวที่ขรุขระ การใช้พลังงานแบบโซลาเซลล์ ทำให้ทำงานได้ยาวนาน ความทนทานของตัวถัง เหล่านี้เป็นตัวอย่างของคุณสมบัติที่ดี

โครงงานในเทอมนี้จะมีคุณสมบัติเป็นรถสำรวจ ที่สามารถเคลื่อนที่ได้จากการสั่งงานจากคอมพิวเตอร์แบบไร้สาย สามารถดูภาพจากกล้อง และบันทึกไว้ได้ ซึ่งจะต้องใช้ความรู้ในหลายๆด้านมาทำงานเข้าด้วยกัน เช่น การเขียนโปรแกรมคอมพิวเตอร์ การใช้งานไมโครคอนโทรลเลอร์ การสื่อสารไร้สาย พื้นฐานแมคานิกส์ ซึ่งจะเป็นพื้นฐานในการพัฒนาต่อไป

บทที่ 2
หลักทฤษฎี

2.1 ไมโครคอนโทรลเลอร์

2.1.1 โครงสร้างและสถาปัตยกรรมของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

ไมโครคอนโทรเลอร์ MCS-51 มีหน่วยความจำแบบแฟลช (flash memory) ของ Atmel Corporationมีเบอร์ขึ้นต้นด้วย AT89 เหตุผลที่ใช้ไมโครคอนโทรลเลอร์แบบนี้ในการเรียนรู้เพื่อใช้งานไมโครคอนโทรลเลอร์ MCS-51 มีด้วยกันหลายประการดังนี้

1.หน่วยความจำโปรแกรมภายในตัวไมโครคอนโทรลเลอร์แบบแฟลช ทำให้สามารถลบและเขียนใหม่ นับพันครั้ง จึงสามารถใช้งานในรูปแบบไมโครคอนโทรลเลอร์ชิปเดี่ยวไม่ต้องใช้หน่วยความจำภายนอกส่งผลให้ใช้งาน port input output ของ microcontroller ได้อย่างมีประสิทธิภาพ

2.ต้นทุนในการพัฒนาระบบ microcontroller ลดลงอย่างมากเนื่องจากไม่ต้องใช้เครื่องมือพัฒนาจำพวก emulator และเครื่องโปรแกรม eprom

3.บริษัทผู้ผลิตได้ทำการผลิต microcontroller ตระกูลนี้ออกมาหลายเบอร์ และมีความสามารถแตกต่างกัน

4.ด้วยการใช้หน่วยความจำภายใน microcontroller ทำให้สามารถป้องกันการคัดลอกข้อมูลของหน่วยความจำได้อย่างดี

5.ในบางเบอร์ของ microcontroller ที่ผลิตโดย Atmel สามารถทำงานโปรแกรมข้อมูลในหน่วยความจำได้โดยที่ไม่ต้องถอดตัว microcontroller ออก มาทำการโปรแกรมใหม่ หรือเรียกว่าการโปรแกรมในวงจร หรือในระบบ(In-system Programming) ทำให้การพัฒนาหรือการซ่อมบำรุง ตลอดจนการปรับปรุงหรือการ up-grade ข้อมูลในหน่วยความจำโปรแกรมทำได้อย่างสะดวก ภายใต้งบประมาณไม่สูงนัก

6.ชุดคำสั่งและสถาปัตยกรรมพื้นฐาน เหมือนกับmicrocontroller mcs-51 ของผู้ผลิตอื่น

คุณสมบัติของไมโครคอนโทรลเลอร์ MCS-51 อนุกรม AT89XX

1.เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียูขนาด 8 บิต

2.ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนได้พันครั้ง

3.หน่วยความจำข้องมูลพื้นฐานเป็นหน่วยความจำแบบแรม ในบางเบอร์จะมีหน่วยความจำแบบอีพรอมเพิ่มเติม

4.ขาพอร์ตเป็นแบบสองทิศทางสามารถใช้งานเป็นได้ทั้งอินพุตและเอาท์พุต

5.มีวงจรสื่อสารแบบอนุกรมแบบฟูลดูเพล็กซ์

6.ไทเมอร์/เคาท์เตอร์ขนาด 16 บิตอย่าน้อง 2 ตัว

7.สามารรองรับแหล่งกำเนิดอินเตอร์รัปต์ได้ 6 ประเภท

8.สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้สูงสุด 64 กิโลไบต์

9.มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป

10.มีวงจรสื่อสารแบบอนุกรม SPI สำหรับอนุกรม AT89Sxx

11.มีวอตช์ด็อกไทเมอร์ในตัวสำหรับในอนุกรม AT89Sxx

ในรูป 2.1 เป็นโครงสร้างพื้นฐานของไมโครคอนโทรเลอร์ MCS-51 ในอนุกรม AT89Cxx จะเห็นได้ว่าโครงสร้างของ AT89Cxx จะเหมือนกับ ไมโครคอนโทรลเลอร์ตระกูล MCS-51 พื้นฐาน หากแต่แตกต่างกันเฉพาะหน่วยความจำแบบแฟลชที่เพิ่มเติมเข้ามา หากเป็นไมโครคอนโทรลเลอร์ในตระกูล 87xx หน่วยความจำภายในจะเป็นแบบอีพรอม และบางเบอร์สามารถโปรแกรมได้เพียงครั้งเดียว

สำหรับในรูป 2.2 เป็นโครงสร้างพื้นฐานของอนุกรม AT89Sxx จะเห็นได้ว่ามีส่วนประกอบที่เพิ่มเติมแตกต่างจาก AT89Cxx อยู่หลายส่วน อาทิ วงจรเชื่อมต่ออนุกรมแบบ SPI ซึ่งในไมโครคอนโทรลเลอร์อนุกรมนี้ใช้ในการเขียนข้อมูลลงในหน่วยความจำอนุกรมโดยไม่ต้องถอดตัวชิปออกไปจากระบบหรือเรียกว่าการโปรแกรมในวงจร ไทเมอร์ / เคาน์เตอร์ขนาด 16 บิตที่เพิ่มเติมเข้ามาอีก 1 ตัว เป็นไทเมอร์ 2 และวงจร วอตช์ด็อกที่ใช้ในการตรวจสอบการทำงานผิดพลาดของซีพียู

ในตารางที่ 2.1 แสดงรายละเอียดบางส่วนของไมโครคอนโทรลเลอร์ MCS-51 แต่ละเบอร์ที่ Atmel ผลิตขึ้น และมีใช้งานอยู่ในปัจจุบัน

2.1.2 การจัดขาของไมโครคอนโทรเลอร์ MCS-51

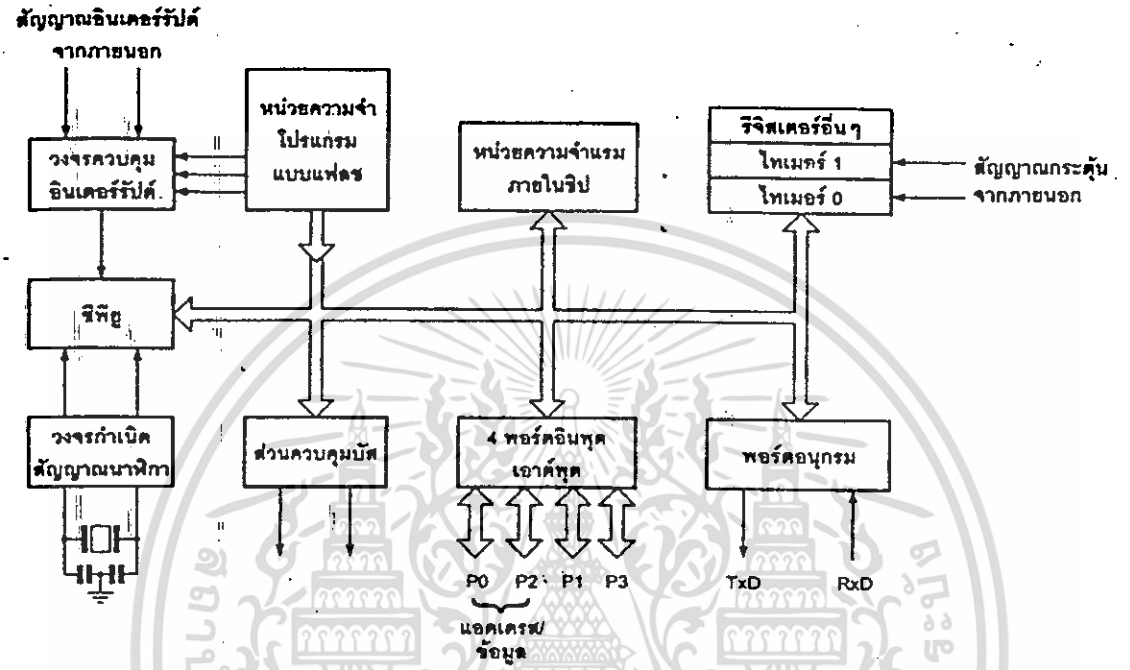
ไมโครคอนโทรลเลอร์ MCS-51 ทุกเบอร์จะมีสถาปัตยกรรมและขาใช้งานพื้นฐานเหมือนกันดังแสดงในรูป.....และ..... โดยมีรายละเอียดขั้นต้นดังนี้

ขา VCC ใช้สำหรับต่อไฟเลี้ยง +5V

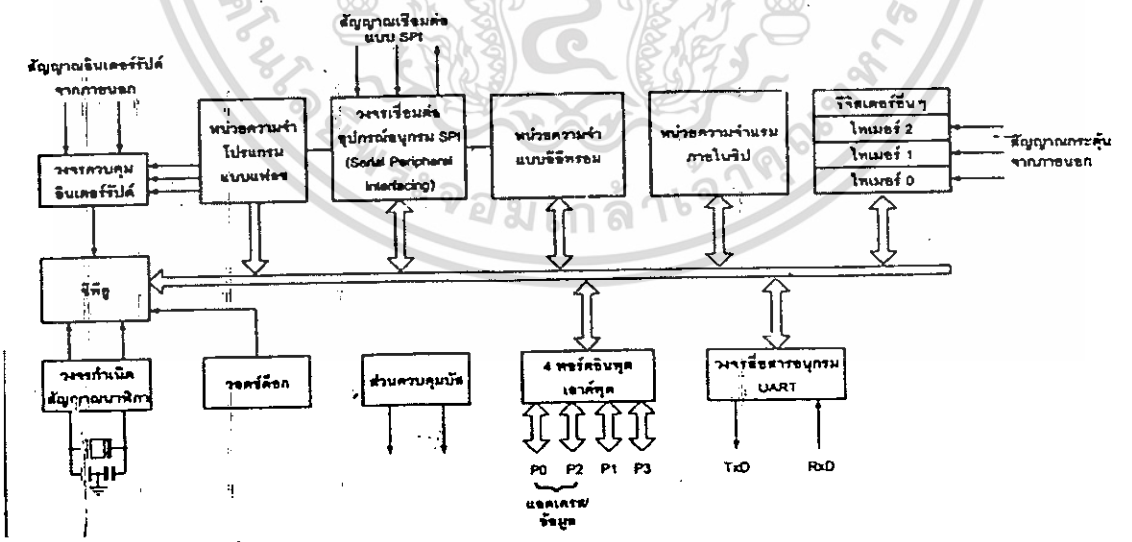
ขา GND สำหรับต่อกราวด์ของระบบ

ขาพอร์ต 0 (P0.0-P0.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาท์พุต สำหรับใช้งานทั่วไปถ้าหากต้องการให้ขาพอร์ต 0 ขาใดขาหนึ่งสามารถทำได้โดยกาเขียนข้อมูล 1 ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วยส่งผลให้ขานั้นปล่อยลอยจึงมีอินพุต

อิมพีแดนซ์สูงสามารถใช้งานเป็นขาอินพุตได้ นอกจากนั้นขาอินพุตยังถูกใช้งานติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอกและขาข้อมูลโดยใช้มัลติเพล็กซ์เข้าช่วยเพื่อสลับการทำงานเป็นได้ทั้งขาติดต่อแอดเดรสและขาข้อมูล



รูปที่ 2.1 ส่วนประกอบของไมโครคอนโทรลเลอร์ AT89CXX



รูปที่ 2.2 ส่วนประกอบของไมโครคอนโทรลเลอร์ AT89SXX

ขาพอร์ต 1 (P1.0-P1.7) มี 8 ขา แต่ละสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตใช้สำหรับงานทั่วไปหากต้องการให้ขาพอร์ตใดเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล 1 ไปยังแต่ละบิตที่ต้องการติดต่อด้วยนอกจากนั้นในอนุกรม AT98Sxx จะใช้ขา P1.0 เป็นขาอินพุตสำหรับนับค่าของไทเมอร์ 2 ในขณะที่ขา P1.4 ถึง 1.7 เป็นขาสำหรับเชื่อมต่อแบบ SPI เพื่อทำการโปรแกรมข้อมูลในระบบ

ขาพอร์ต 2 (P2.0-P2.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตใช้สำหรับงานทั่วไปหากต้องการให้ขาพอร์ตใดเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล 1 ไปยัง แต่ละบิตของพอร์ตส่งผลให้ขานั้นปล่อยลอยจึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาอินพุตได้นอกจากนั้นขาพอร์ตนี้ยังถูกใช้งานในการติดต่อ กับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก

ขาพอร์ต 3 (P3.0-P3.7) มี 8 ขาแต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตใช้สำหรับงานทั่วไปหากต้องหารให้ขาพอร์ตใดเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล 1 ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วยส่งผลให้ขานั้นปล่อยลอยจึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาอินพุตได้นอกจากนั้นขาพอร์ต 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษดังมีรายละเอียดขั้นต้นต่อไปนี้

P3.0 ใช้งานเป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RXD

P3.1 ใช้งานเป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรมหรือ TXD

P3.2 ใช้งานเป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 0 หรือขา INTO

P3.3 ใช้งานเป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขา INT1

P3.4 ใช้งานเป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0 หรือขา T0

P3.5 ใช้งานเป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 1 หรือขา T1

P3.6 ใช้งานปืนขาสัญญาณ WR ในกรณีที่ใช้เชื่อมต่อความจำภายนอก

P3.7 ใช้งานปืนขาสัญญาณ RD ในกรณีที่ใช้เชื่อมต่อความจำภายนอก

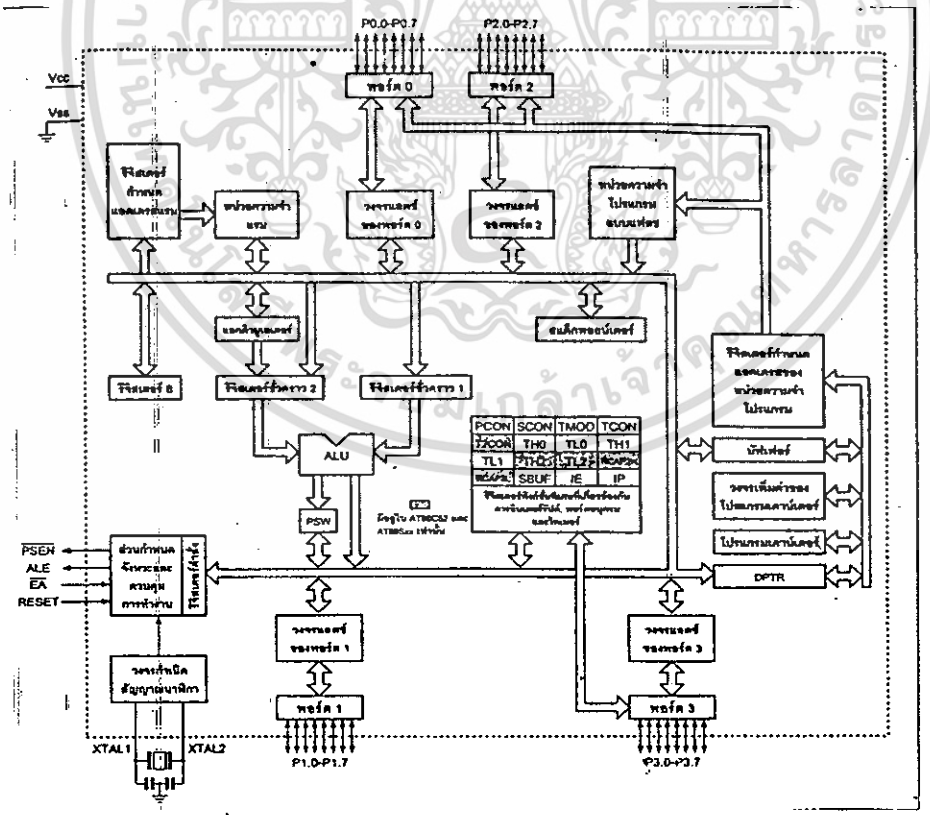
สัญญาณเพื่อการรีเซตสถานะที่ขานี้ต้องอยู่ในสถานะอย่างน้อย 2 แมทซีนไซเกิลโดยที่วงจรกำหนดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างปกติ

ขา ALE/PROG (Address Latch Enabel/Program pulse input) เป็นขาที่ใช้ในการควบคุมการแลตซ์ของขาพอร์ต 0 เมื่อมีการใช้หน่วยความจำภายนอก นอกจากนั้นขาที่ยังใช้เป็นขาสำหรับพัลส์ของการโปรแกรมสำหรับโปรแกรมข้อมูลลงในไมโครคอนโทรลเลอร์ในรุนนี้มีหน่วยความจำเป็นแบบอีพรอม

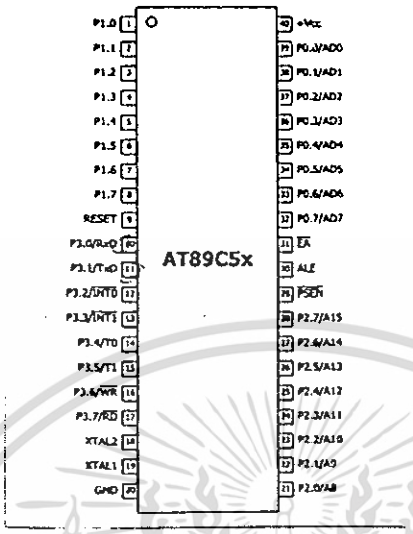
ขา RSEN (Program Store Enabel)ขานี้ใช้ในการส่งสัญญาณร้องขอติดต่อกับหน่วยความจำภายนอกเทฯมื่อไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลความจำจากโปรแกรมภายนอกตัวไมโครคอนโทรลเลอร์ จะส่งสัญญาณมาที่ขานี้ 2 ครั้ง ในแต่ละแมทชีนไซเกิล แต่ถ้าหากติดต่อกับหน่วยความจำข้อมูลภายนอก ขานี้จะไม่มีการส่งข้อมูลใดๆออกมา

ขา EA/Vpp (External Accessenable/Programming voltage input) ใช้เลือกการติดต่อหน่วยความจำโปรแกรมจากภายนอกหรือภายในตัวไมโครคอนโทรลเลอร์ถ้าหากขานี้เป็น 0 เป็นการเลือกให้ไมโครคอลโทรลเลอร์ติดต่อกับหน่วยความจำภายนอก แต่ถ้าหากขานี้เป็น1 เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกัยหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ นอกจากนี้ขานี้ยังใช้เป็นขาอินพุตสำหรับแรงดันไฟสูงสำหรับโปรแกรมหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชต้องการแรงดันสำหรับการโปรแกรม +12V

ขา XTAL1และ XTAL2 เป็นขาสำหรับการติดต่อคริสตอลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของตัวไมโครคอนโทรลเลอร์



รูปที่ 2.3 สถาปัตยกรรมของไมโครคอนโทรลเลอร์



รูปที่ 2.4 แสดงพอร์ตของไมโครคอนโทรลเลอร์

2.1.3 โครงสร้างและการทำงานของพอร์ต

ไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีพอร์ตให้ใช้งานได้ทั้งสิ้น 4 พอร์ตคือพอร์ต 0 ถึงพอร์ต 3 แต่ละพอรืตมีขนาด 8 บิต เป็นพอร์ตแบบ 2ทิศทาง กล่าวคือเป็นได้ทั้งอินพุตสำหรับสำหรับข้อมูลเข้าและเอาท์พุตสำหรับส่งสัญญาณข้อมูลออก ทุกพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีช่วงแลตช์แลวงจรขับตลอดจนบัฟเฟอร์อินพุต ดังแสดงให้เห็นในสถาปัตยกรรมรูปที่ 2.3

ที่พอร์ต 0 และพอร์ต 2 จะใช้งานปันทั้งอินพุตและเอาท์พุตสำหรับใช้งานทั่วไปและใช้ในการติดต่อกับหน่วยความจำภายนอก สำหรับพอร์ต 3 ทั่งพอร์ตและพอร์ต 1 บางขา นอกจากจะใช้เป็นขาอินพุตเอาท์พุตปกติแล้วยังสามารถใช้วานในหน้าที่พิเศษขึ้นอยู่กับว่าเป็นไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ใด ดังสรุปในตารางที่ 2-2

ในรูปที่ 2.5 แสดงวงจรภายในของแต่ละพอรืตไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช โดยในรูปที่ 2.5 (ก) เป็นวงจรของพอร์ต 0 วงจรแลตช์ของแต่ละบิตในแต่ละพอร์ตก็คือวงจรดีฟลิฟลอฟนั่นเอง การอ่านค่าสถานะของพอร์ตและสถานะของวงจรแลตช์สามารถกระทำได้อย่างอิสระด้วยสัญญาณที่แยกจากกัน นั่นคือสัญญาณข้อมูลที่แยกจากขาพอร์ต และสัญญาณอ่านข้อมูลจากวงจรแลตช์ ส่วนการเขียนข้อมูลมายังพอร์ตต้องส่งสัญญาณมาจากขา CLK ของดีฟลิฟลอฟในขณะที่ข้อมูลจะผ่านมายังขาบัสข้อมูลภายในสู่ขา D ของดีฟลิฟลอฟ

ที่พอร์ตนี้มีวงจรมัลติเพล็กซ์สำหรับกำหนดการทำงานของพอร์ตว่า ต้องการใช้งานเป็นพอร์ตปกติหรือใช้ในการติดต่อกับหน่วยความจำภายนอกไมโครคอนโทรลเลอร์

เนื่องจากที่ขาพอร์ต 0 ไม่มีวงจร พูลอัปภายในหากมีการนำพอร์ต 0ไปใช้งานเป็นพอร์ตอินพุตจะต้องต่อตัวต้านทานพูลอัปภายนอกที่ขาพอร์ต 0 ทุกขาด้วย

ในรูปที่ 2.5(ข) เป็นวงจรของพอร์ต 1 ซึ่งมีลักษณะโดยทั่วไปคล้ายพอร์ต 0 แต่ไม่มีวงจรมัลติแพลกซ์ เนื่องจากพอร์ตนี้จะไม่ใช้ในการติดต่อกับหน่วยความจำภายนอก แต่วงจรพูลอัปภายในที่แต่ละบิตของพอร์ตนี้แทน สำหรับรายละเอียดของวงจรพูลอัปแสดงในรูปที่ 2-6

ในรูปที่ 2-56(ค) เป็นวงจรภายในของพอร์ต 2 จะคล้ายพอร์ต 0 มากต่างกันเพียงมีวงจรพูลอัปเพิ่มเติมเข้ามาส่วนในรูปที่ 2-5 (ง) เป็นวงจรภายในของพอร์ต3 จะเห็นได้ว่าคล้ายพอร์ต 01 มีการเพิ่มวงจรบัฟเฟอร์และวงจรอินพุตเอาท์พุตเมื่อทำงานในฟังก์ชั่นพิเศษเข้ามา เนื่องจากพอร์ต 3 สามารถทำงานในฟังก์ชั่นพิเศษได้ทุกขา

2.1.3.1 การใช้งานเป็นพอร์ตอินพุต

เนื่องจากขาพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช สามารถเป็นได้ทั้งอินพุตและเอาทีพุตดังนั้นจึงความจำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจการกำหนดการทำงานของพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

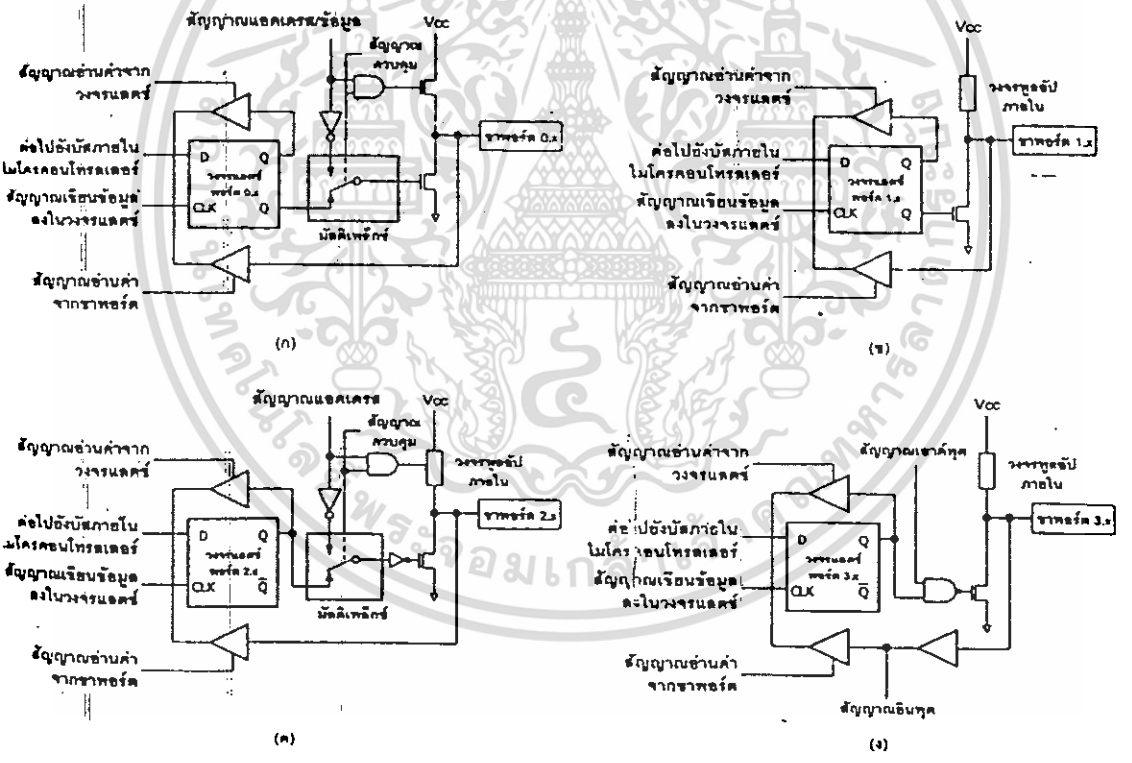
ในการกำหนดเป็นพอรืตอินพุตต้องเริ่มต้นด้วยการเขียนข้อมูล 1 มาแต่ละบิตที่ต้องใช้งานเป็นพอร์ตอินพุต เพื่อหยุดการทำงานของแฟลชที่ใช้ในการขับสัญญาณเอาท์พุตของบิตนั่นๆทำให้ขาสัญญาณของพอร์ตเชื่อมต่อเข้ากับวงจรพูลอัปภายในโดยตรง ส่งให้ขาพอร์ตนั่นมีลอจิกเป็น 1 สามารถรับสัญญาณลอจิก 0 จากอุปกรณ์ภายนอกได้ง่ายสัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไงว้ในวงจรบัฟเฟอร์ภายในพอร์ต แล้วรอให้ซีพียูอ่านค่าเข้าไป เมื่อเป็นเช่นนี้อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชควรกำหนดให้ทำงานในสถานะลอจิก 0 จะดีและสะดวกที่สุด

2.1.3.2 การใช้งานเป็นพอร์ตเอาท์พุต

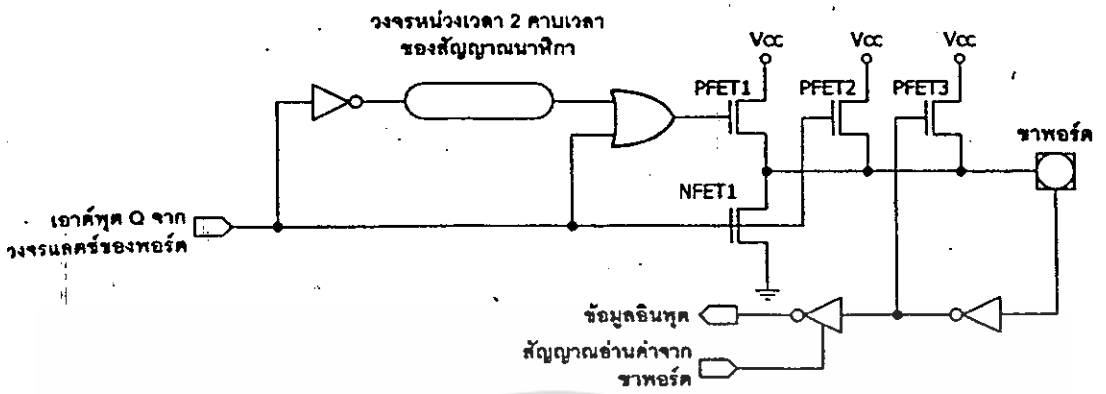
โดยปกติขาพอร์ตจะถูกกำหนดให้เป็นเอาต์พุตอยู่แล้วดังนั้นจึงสามารถส่งข้อมูลออกไปอย่างง่ายดายและตรงไปตรงมากล่าวคือเมื่อต้องการส่งข้อมูล 0 ออกไปทางเอาต์พุตก็ให้เขียนข้อมูล 0 ออกไปยังวงจรแลตช์ ซึ่งจะส่งต่อไปขับเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก 0 ขึ้น ในทางตรงกันข้ามหากต้องการส่งข้อมูล 1 ออกไป ก็ให้เขียนข้อมูล 1 ออกไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ตเชื่อมต่อวงจรพูลอัพภายใน

เกิดเป็นลอจิก 1 ที่ขาพอร์ตนั้นซึ่งคล้ายกับการกำหนดให้เป็นขาอินพุตมาก เพียงแต่แตกต่างกันที่ขบวนการในการเคลื่อนย้ายข้อมูลโดยถ้าเป็นอินพุตจะส่งสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาต์พุตจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใด เว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาต์พุต

เมื่อใช้งานเป็นพอร์ตเอาต์พุต แต่ละขาของแต่ละพอร์ตมีความสามารถในการจ่ายกระแสหรือที่เรียกว่า กระแสซอร์สได้สูงสุด 10 mA และทุกขารวมกันในแต่ละพอร์ตสูงสุด 26 mA สำหรับ พอร์ต 0 และ 15 mA สำหรับพอร์ต 1-3 ในกรณีที่ใช้งานทุกพอร์ตเอาต์พุตสามารถจ่ายกระแสได้รวมกันสูงสุด 71 mA ดังนั้นการใช้งานเป็นพอร์ตเอาต์พุตเพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัฟเฟอร์ทางเอาต์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง



รูปที่ 2.5 แสดงวงจรภายในของพอร์ตต่างๆ



วงจรพูลอัปประกอบด้วยเฟตรนิคพีแชนเนล 3 ตัวคือ PFET1-PFET3 โดย NFET1 จะทำงานเมื่อได้รับลอจิก "1" จากขา Q และหยุดทำงานเมื่อได้รับลอจิก "0" วงจรพูลอัปจะเริ่มต้นทำงานเมื่อ NFET1 ได้รับลอจิก "1" PFET1 จะทำงานนานประมาณ 2 คาบเวลาของสัญญาณนาฬิกาภายใน หลังจากที่เกิดการเปลี่ยนแปลงจากลอจิก "0" เป็นลอจิก "1" ในขณะที่ PFET1 ทำงาน จะทำให้ PFET3 ทำงานตามไปด้วย ทำให้เกิดการพูลอัปขาพอร์ต

รูปที่ 2.6 วงจรพูลอัพ

2.1.3.3 การอ่านค่าลอจิกทางพอร์ต

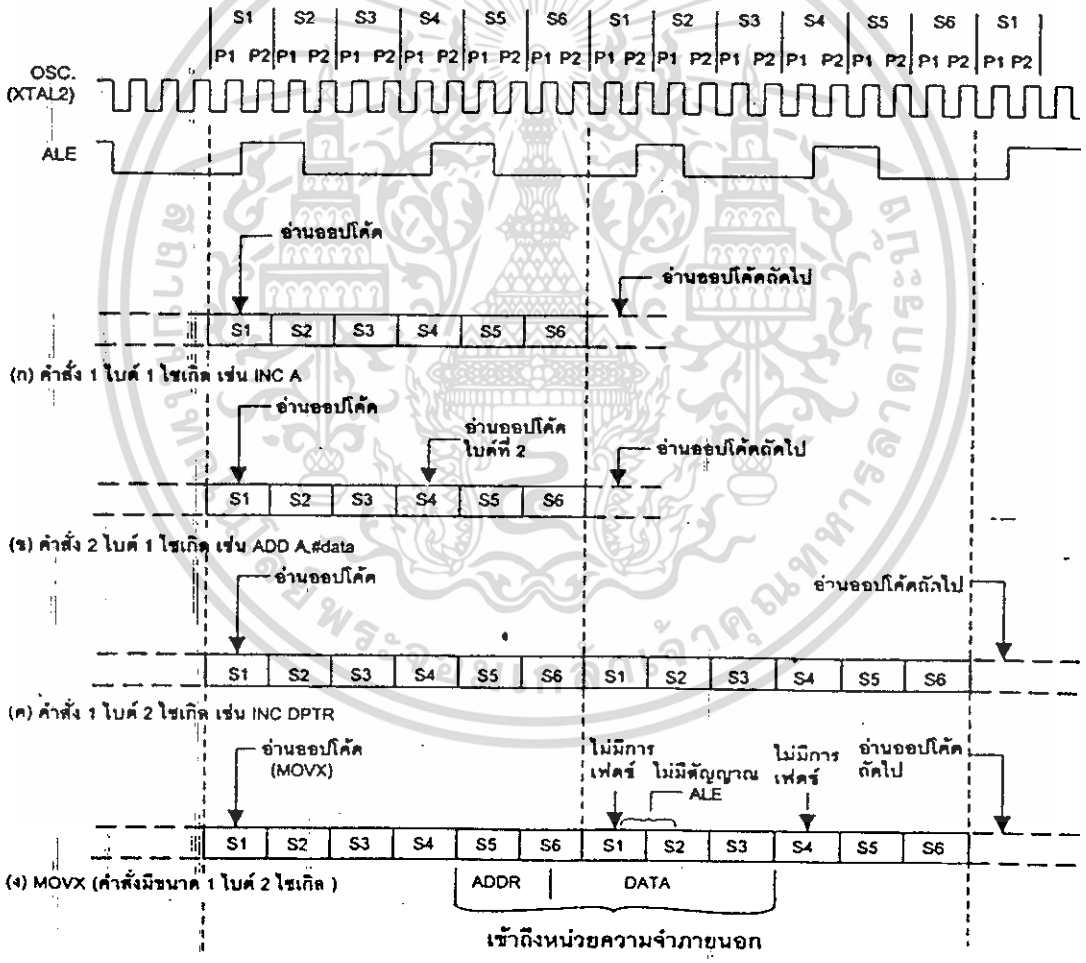
ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถอ่านค่าโลจิกทางพอร์ตได้ 2 ลักษณะคือ อ่านจากขาพอร์ตโดยตรงและอ่านจากวงจรแลตช์ของแต่ละพอร์ต

ในกรณีที่ขาของพอร์ตต่อกับขาเบสทรานซิสเตอร์ NPN และขา อิมิตเตอร์ ของทรานซิสเตอร์ตัวนั้นต่อลงกราวด์ หากมีการส่งข้อมูล 1 ไปยังทรานซิสเตอร์ จะทำให้ทรานซิสเตอร์ทำงานที่สถานะลอจิกที่ขาพอร์ตจะเป็น 0 เนื่องจากทรานซิสเอตร์ทำงานเหมือนกับว่าขาพอร์ตนั้นต่อลงกราวด์ทำให้หากอ่านค่าลอจิกที่ขาพอร์ตจะได้ผลตรงข้ามกับที่ส่งออกมาแต่ถ้าหากทำการอ่านค่าลอจิกที่วงจรแลตช์จะได้ค่าตรงกับค่าที่ทำการส่งจริง ดังนั้นในการอ่านค่าลอจิกจากพอร์ตจึงต้องเลือกวิธีการให้เหมาะสมกับอุปกรณ์ที่นำมาต่อด้วย

2.1.4 จังหวะในการทำงานของไมโครคอนโทรลเลอร์ MCS-51

ในการใช้งานไมโครคอนโทรลเลอร์ MCS-51 จะต้องทำความเข้าใจจังหวะการทำงานของซีพียูและลำดับขั้นตอนการประมวลผลคำสั่งในการประมวลผล ผลของคำสั่งของซีพียูจะมีขั้นตอนหลัก 2 ขั้นตอน คือ กระบวนการเฟต เป็นการเรียกคำสั่งออกมาจากหน่วยความจำโปรแกรมแล้วทำการแปลงหรัสคำสั่งนั้นเป็นภาษาเครื่องเพื่อเตรียมการประมวลผล ขั้นตอนต่อมาคือ กระบวนการเอ็กซิคิวต์ เป็นการทำตามคำสั่งที่กำหนดหรือตามที่เฟตช์ขึ้นมาโดยกระบวนการก่อนหน้านี้เมื่อทำการเอ็กซิคิวต์คำสั่งเรียบร้อยแล้วก็จะไปเริ่มกระบวนการเฟตช์คำสั่งใหม่ต่อไป

เมื่อเริ่มจ่ายไฟให้แก่ไมโครคอนโทรลเลอร์จะเกิดการรีเซตในลักษณะที่เรียกว่าพาวเวอร์ออนรีเซตซีพียูเริ่มทำงานที่แอดเดรส 000Hของหน่วยความจำโปรแกรมจังหวะการทำงานของซีพียูจะเป็นไปตามรูปแบบโดยได้รับการกำหนดมาจากรอบการทำงานหรือแมชชีนไซเคิล ในรูปที่ 2.7 เป็นไดอะแกรมเวลาแสดงจังหวะการทำงานของไมโครคอนโทรเลอร์ MCS-51 โดยใน 1รอบการทำงานหรือ 1 แมชชีนไซเกิลจะแบ่งย่อยออกเป็น 6 สเตต กำหนดชื่อเป็น S1-S6 ในแต่ละสเตตมีเวลาเท่ากับ 2 คาบเวลา ของสัญญาณนาฬิกา ถ้าสัญญาณนาฬิกามีความถี่ 12MHZ จะมีคาบเวลาเท่ากับ 1ms คาบเวลาทั้งสองภายในหนึ่งสเตตจะเรียกว่า เฟส 1 และ เฟส 2



รูปที่ 2.7 แสดงแมชชีนไซเคิลการทำงานของไมโครคอนโทรลเลอร์

ในรูปที่ 2.8 จะเป็นการเอ็กซิคิวต์คำสั่งที่ใช้เวลา 1 ไซเกิล เริ่มต้นที่สเตต 1 จะเป็นการอ่านค่าออปโค้ด อันเป็นกระบวนการแลตช์ค่าโค้ดส่งไปยังไปให้รีจิสเตอร์คำสั่ง การแฟตช์ครั้งที่ 2 จะเกิดที่สเตต 4 ภายในแมชชีนไซเกิลเดียวกัน

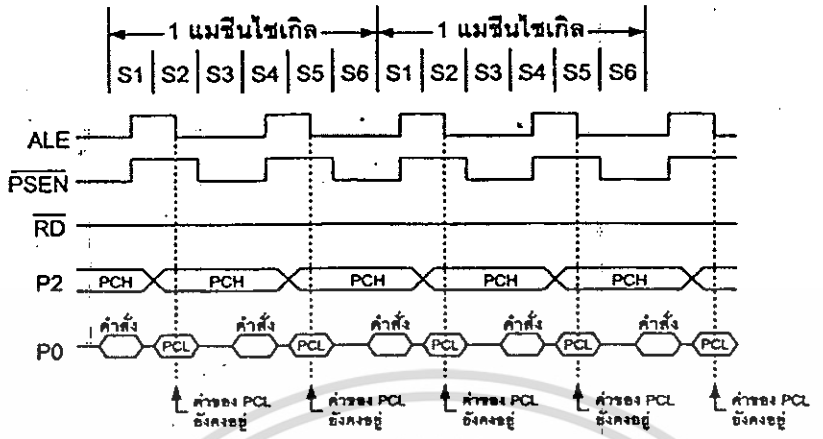
ในกรณีคำสั่งใช้ 2 ไซเคิล การทำงานของคำสั่งจะสิ้นสุดลงในสเตต 6 ของแมทชีนไซเกิลที่ 2 ดังนั้นไดอะแกรมในรูป 2.8 สำหรับในการทำคำสั่งMOVX ซึ่งเป็นคำสั่งขนาด 1 ไบต์ 2 ไซเกิล จะไม่มีการเฟตช์เกิดขึ้นในไซเกิลที่ 2 ของคำสั่ง MOVX เนื่องจากซีพียูจะไปทำการติดต่อกับหน่วยความจำภายนอกดังแสดงในไดอะแกรมรูปที่ 2.8 จะเห็นได้ว่าเวลาในการเอ็กซิคิวต์จะไม่ได้ขึ้นอยู่กับว่าทำการติดต่อหน่วยความจำโปรแกรม
ภายในหรือภายนอก

ในรูปที่ 2.8 แสดงสัญญาณและไดอะแกรมเวลาของการเข้าถึงหน่วยความจำโปรแกรมภายนอกโดยในรูป 2.8 เป็นไดอะแกรมเวลาในขณะที่ยังไม่มีการทำคำสั่ง MOVX สัญญาณที่ขา ALE และ PSEN จะเกิดการแอกตีฟ 2 ครั้งภายในหนึ่งแมทชีนไซเกิล ในทุกครั้งที่ ALE เกิดการแอกตีฟที่พอร์ต 0 (P0) จะมีค่าของรีจิสเตอร์ของPC ในไบต์ต่ำออกมา ในขณะที่พอร์ต 2 (P2) ก็จะมีค่าของ PC ในไบต์สูงสูงเพื่อชี้ไปยังแอดเดรสต่อไปที่ต้องไปดำเนินการ สำหรับขา PSEN ก็จะแอคตีฟเมื่อมีการติดต่อกับหน่วยความจำโปรแกรมภายนอก ในกรณีที่กระทำคำสั่ง MOVX เพื่อเข้าถึงหน่วยความจำโปรแกรมภายนอก ที่ขา PSEN จะไม่เกิดการแอตีฟ 2ครั้งภายใน 1แมทชีนไซเกิล เนื่องจากบัสแอดเดรสและบัสข้อมูลจะถูกใช้ในการติดต่อหน่วยความจำข้อมูลภายนอกแทนแต่สำหรับสัญญาณ ALE ยังคงแอกตีฟตามจังหวะการทำงานเหมือนเดิม

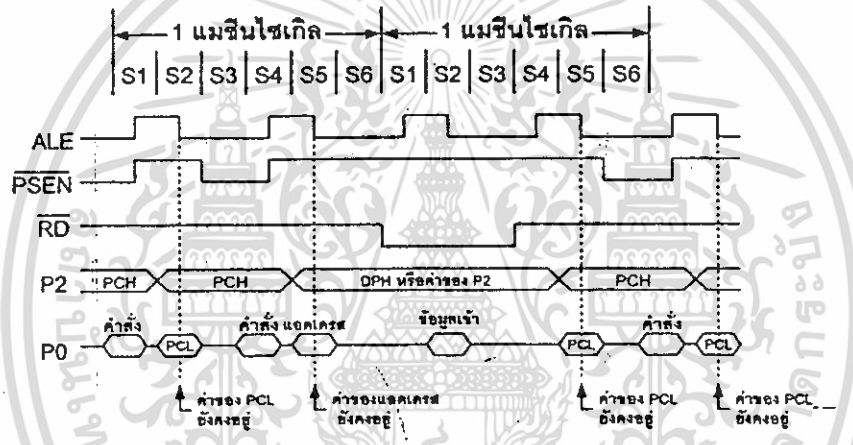
จากไดอะแกรมเวลาสามารถสรุปได้ว่า ในการทำงานภายใน 1 รอบ หรือ 1 แมทชีนไซเกิล ซีพียูในไมโครคอนโทรลเลอร์ MCS-51 จะใช้เวลา 12 คาบ เวลาของสัญญาณนาฬิกา นั่นคือเวลาในการทำงาน 1 แมทชีนไซเกิลมีค่าเท่ากับ 1 ms หรือมีความเร็วในการทำงานภายใน 1MHZ ในกรณีที่ใช้ความถี่ สัญญาณนาฬิกา 12 MHZ

ดังนั้นถ้าต้องการทราบความเร็ว ในการทำงานของไมโครคอนโทรลลเอร์ MCS-51 สามารถหาได้จากค่าความถี่สัญญาณนาฬิกาหารด้วย 12 และถ้าต้องการหาค่าเวลาของ 1 รอบการทำงาน หรือ 1 แมทชีนไซเกิล สามารถทำได้โดยการหาส่วนกลับของความเร็ว ในการทำงานภายในของไมโครคอนโทรลเลอร์ MCS-51 สามารถสรุปเป็นสูตรคณิตศาสตร์ได้ดังนี้

ความเร็วของการทำงานภายในของไมโครคอนโทรลเลอร์MCS-51 เท่ากับ
ความถี่ของสัญญาณนาฬิกา(ของคริสตอลที่ต่ออยู่ที่ขา XTAL1 และ XTAL 2) /12
เวลา 1 แมทชีนไซเกิล = 1/ความเร็วในการทำงานภายในของไมโครคอนโทรลเลอร์



(ก) เมื่อไม่กระทำคำสั่ง MOVX



(ข) เมื่อกระทำคำสั่ง MOVX

รูปที่ 2.8 แสดงไดอะแกรมของหน่วยความจำภายนอก

2.2 ความรู้เบื้องต้นเกี่ยวกับวิชวลเบสิก

Visual Basic ถือเป็นเครื่องมือพัฒนาแอพพลิเคชั่นที่ถูกพัฒนามาอย่างต่อเนื่องทั้งจากไมโครซอฟท์ซึ่งปันผู้ให้กำเนิดและบริษัทผู้ผลิตซอฟแวร์ต่างๆ ทั่วโลก จนถึงทุกวันนี้ Visual Basic 6.0 ได้ถูกเพิ่มความสามารถในการทำงานอย่างมากมาย ซึ่งเป็นเครื่องมือพัฒนาแอพพลิเคชั่นที่คนทั่วโลกยอมรับในความง่ายและ ความสามารถ

2.2.1 ลักษณะของ Visual Basic

Visual Basic มีจุดเด่นที่แตกต่างจากเครื่องมือชุดอื่นๆ คือ ความเรียบง่ายในการพัฒนาแอพพลิเคชั่น ทำให้ลดเวลาในการพัฒนาแอพพลิเคชั่นลงมาก ซึ่งเป็นผลมาจาก Visual Basic

สนับสนุนการพัฒนาแอพพลิเคชั่นแบบ Component ซึ่งคือการนำเอาส่วนประกอบ (Component) ด้านซอฟแวร์ ที่ได้สร้างและทดสอบเป็นอย่างดีแล้ว นำมาประกอบกัน แล้วเขียนคำสั่งกำกับการทำงานให้เป็นแอพพลิเคชั่นที่ใช้งานได้จริง

- การพัฒนาแอพพลิเคชั่นโดยใช้ ActiveX Control เพราะความสำเร็จของ Visual Basic เกิดจากการนำเอาองค์ประกอบด้านซอฟต์แวร์ต่างๆ มาร่วมกันเป็นแอพพลิเคชั่น องค์ประกอบด้านซอฟต์แวร์เหล่านั้นรู้จักกันในนามของ ActiveX Control (หรือ OLE Control, หรือ VBX สำหรับคนที่คุ้นเคยกับ Visual Basic 3.0)

ActiveX Control นั้นถูกสร้างขึ้นมาอย่างมากมาย จากบริษัทผู้ผลิตซอฟต์แวร์ทั่วโลก หรือแม้แต่ถูกสร้างขึ้นมาจากผู้ใช้งาน Visual Basic เอง ซึ่งทำให้แอพพลิเคชั่นที่สร้างขึ้น ตรงกับความต้องการของผู้ใช้มากที่สุด

- ความสามารถในการพัมนาแบบ Object Oriented Programming เมื่อการทำงานในชีวิตจริงทวีความซับซ้อนมากยิ่งขึ้น การพัฒนาแอพพลิเคชั่นเพื่อช่วยในการทำงานแบบเดิมๆที่เคยใช้ได้ดีกับ Visual Basic อาจทำให้เวลาในการพัมนายาวนานกว่าปกติ หรือเพิ่มความยุ่งยากในการบำรุงรักษาทำให้แนวคิดในการพัฒนาแอพพลิเคชั่นในเชิงวัตถุถูกคิดขึ้นมาและถูกเติมเข้าไปใน Visual Basicอย่างสมบูรณ์จริงๆแล้วแนวความคิดแบบ Object Oriented Programming นี้มีมานานแล้วและซ่อนอยู่เบื้องหลังการทำงานของ Visual Basic ทำให้ผู้ใช้งาน Visual Basic แทบไม่รู้สึกตัวว่ากำลังใช้การพัฒนาแบบ Object Oriented-Programming อยู่เลย และตั้งแต่ เวอร์ชั่น 4.0 จนถึงเวอร์ชั่นปัจจุบัน Visual Basic ก็เผยความสามารถด้านObject Oriented Programming ให้ผู้ใช้ Visual Basic ได้ใช้กัน

- ความเหนือชั้นในงานด้านฐานข้อมูล สิ่งที่โดดเด่นอื่นๆนอกจากความเรียบง่าย แล้วก็เห็นจะได้แก่ความสามารถในการใช้งานและสร้างแอพพลิเคชันสำหรับฐานข้อมูล ซึ่ง Visual Basic สนับสนุนการใช้งานและสร้างแอพพลิเคชันสำหรับฐานข้อมูล ซึ่ว Visual Basic สนับสนุนการใช้งานกับระบบจัดการฐานข้อมูล (DBMS) หลากหลายไม่ว่าจะเป็นค่ายของไมโครซอฟท์ เช่น Access , FoxPro , SQLSever หรือต่างค่ายไม่ว่าจะเป็น dBase , Oracle , Sybase ฯลฯ จึงทำให้ Visual Basic เป็นตัวเลือกอันดับต้น ๆ ของการสร้างแอพพลิเคชันเพื่องานร่วมกับฐานข้อมูล

- MSDN: คัมภีร์สำหรับนักพัฒนาแอพพลิเคชัน เนื่องจากเครื่องมือพัฒนาแอพพลิเคชันรุ่นใหม่ ๆ ต่างมีความสามารถหลากหลายชิ้น การที่เราจะนำมาศึกษาก่อนนำมาใช้งาน อ้างอิงขณะทำงานคงต้องใช้เอกสารจำนวนมาก ซึ่งคงไม่สะดวกมากนัก ดังนั้นไมโครซอฟท์จึงตั้งเว็บไซต์ (Microsoft Developer Network) ขึ้นมาเพื่อเก็บเอกสารต่าง ๆ ที่จำเป็นมารวบรวม

ไว้ และอัพเดตเป็นระยะ ๆ นอกจากจะอยู่ในอินเตอร์เน็ตแล้ว MSDN เก็บอยู่ในรูปของ CD – ROM สามารถติดตั้งลงในเครื่องPC หรือเซิร์ฟเวอร์ สำหรับการศึกษาและใช้งานได้ตลอดเวลา

- MCSD: หนทางก้าวหน้าของผู้สนใจในการพัฒนาซอร์ฟแวร์ MCSD (Microsoft Certified Software Development) เป็นหนึ่งในประกาศนียบัตรรับรองจากไมโครซอฟท์ที่มอบให้แก่ผู้ที่พัฒนาซอฟต์แวร์แอพพลิเคชันที่ใช้ผลิตภัณฑ์ของไมโครซอฟท์ เช่น Visual Studio , Microsoft Backoffice ซึ่งมีความเข้าใจอย่างจริงจังกับตัวเครื่องมือเหล่านั้น รวมถึงเข้าใจในหลักการทำงานขั้นลึกซึ้งของระบบปฏิบัติการของไมโครซอฟท์ ผู้ที่จะได้ MCSD นั้นจะต้องทำการสอบ ตามข้อสอบที่ไมโครซอฟท์กำหนด ส่วนหนึ่งต้องใช้ความรู้ Visual Basic ด้วย นอกจากนี้ยังต้องเข้าใจในการทำงานภายในของระบบปฏิบัติการ Windows 95/98 รวมทั้งสามารถใช้งานระบบจักการฐานข้อมูล MS SQLSever

2.2.2 การติดตั้ง Visual Basic

เมื่อเราตัดสินใจจะใช้ Visual Basic เป็นเครื่องมือสำหรับพัฒนาแอพพลิเคชัน เราก็คงต้องเตรียมตัวก่อนการติดตั้ง Visual Basic

- ความต้องการของระบบ (System Requirement)

ด้านฮาร์ดแวร์ เราคงต้องเตรียมดังรายละเอียดในตาราง 2.1

ฮาร์ดแวร์	ข้อกำหนดขั้นต่ำ	คำแนะนำ
CPU	Pentium 90 MHz ขึ้นไป	CPU ยิ่งเร็วเท่าไหร่ยิ่งดี
ฮาร์ดดิสก์	Standard Edition : ประหยัด 48 MB, เต็มที่ 80 MB Professional Edition : ประหยัด 48 MB, เต็มที่ 80 MB Enterprise Edition : ประหยัด 128 MB, เต็มที่ 147 MB สำหรับ MSDN : 67 MB, สำหรับ IE 4.X อีก 66 MB	แนะนำให้ติดตั้งด้วย Enterprise Edition และควรจะมีเนื้อหาที่สำหรับ MSDN ให้มากรวมทั้งเนื้อที่สำหรับ Active Control ใหม่ๆ
RAM	24 MB สำหรับ Windows 95/98 32 MB สำหรับ Windows NT	ควรจะเป็น 32 MB ขึ้นไป
CD-ROM	ต้องมี	ขาดไม่ได้
Sound Card	ไม่จำเป็น	กรณีที่ต้องสร้างแอพพลิเคชั่นมัลติมีเดีย อาจจำเป็นต้องใช้งาน

ตารางที่ 2.1 แสดงการติดตั้ง VB1

ด้านซอฟต์แวร์ ต้องมีซอฟต์แวร์ที่จำเป็นต่อการใช้งาน Visual Basic ดังนี้

ฮาร์ดแวร์	ข้อกำหนดขั้นต่ำ	คำแนะนำ
ระบบปฏิบัติการ	Windows 95/98 Windows NT	Windows 98 จะเหมาะสมกว่า
ระบบจัดการฐานข้อมูล	ไม่จำเป็นต้องมีก็ได้	สามารถใช้งานร่วมกับ ROBMS ได้แทบทุกตัวเช่น Access, SQL Server, Oracle เป็นต้น
เว็บบราวเซอร์	จำเป็นต้องมี Internet Explorer 4.01 ขึ้นไป	
เว็บเซิร์ฟเวอร์	หากต้องพัฒนาแอพพลิเคชั่น Internet ก็ต้องมี	สำหรับทดสอบควรใช้ PWS 4.0 สำหรับใช้งานจริงควรใช้ IIS 4.0

ตารางที่ 2.2 แสดงการติดตั้ง VB2

2.2.3 สภาพแวดล้อมการทำงาน

ลักษณะการทำงานของ Visual Basic 6.0 จะทำงานในลักษณะ IDE (Integrated Development Environment) คือ การรวบรวมเครื่องไม้เครื่องมือ, ข้อมูลที่ใช้งานต่างๆ ไว้ในหน้าจอเดียว ทำให้เรียกใช้งานได้ง่าย

Menu Bar เมนูบาร์เป็นส่วนที่รับคำสั่งในแบบเมนู เมื่อเราทำการสร้างแอพพลิเคชั่นด้วย Visual Basicเป็นเหมือนศูนย์กลางที่ควบคุมการสร้าง แอพพลิเคชั่น

Tool Bar ในการใช้งานเมนูบาร์สั่งงานอาจจะมีขั้นตอนที่ยุ่งยาก เพื่อลดขั้นตอนลงเราจะคลิกที่ทูลบาร์เพียงครั้งเดียวก็สามารถสั่งงานที่เราต้องการได้ (เป็นเหมือนคีย์ลัดในการทำงาน)

Project Window เป็นเครื่องมือที่ใช้ควบคุมการทำงานของ Project เพราะ Visual Basic เองสนับสนุนการสร้างแอพพลิเคชั่นหลายแบบ

Properties Window เป็นส่วนที่กำหนดพรอพเพอร์ตี้ให้กับออบเจ็กต์ต่าง ในแอพพลิเคชั่น

From Layout ฟอร์มเลย์เอาท์ เป็นหน้าตาคร่าวๆ ของฟอร์มที่ได้จากการรันแอพพลิเคชั่น

From Designer ฟอร์มดีไซเนอร์ เป็นส่วนที่เรามองเห็นได้ในขณะออกแบบแอพพลิเคชั่นของ Visual Basic เป็นส่วนที่เราจะใช้ติดต่อกับผู้ใช้งาน โดยการนำ ActiveX Control ต่างๆมาวางไว้ข้างบน

Code Window เป็นส่วนที่เราเขียนโปรแกรมเพื่อควบคุมการทำงานของแอพพลิเคชั่น

2.2.4 ชนิดของข้อมูล (Data Type)

ที่ผ่านมาเราได้เขียนโปรแกรมที่มีการใช้ข้อมูลนำไปขับเคลื่อนให้แอพพลิเคชันทำงานได้สำหรับข้อมูลชนิดต่าง ๆ ได้ ที่ Visual Basic อณุญาตให้เราสามารถใช้งานในการเขียนโปรแกรมได้แก่

ชนิด	คำอธิบาย	ขนาด หน่วยความจำ
Byte	เป็นข้อมูลตัวเลขจำนวนเต็มตั้งแต่ 0 ถึง 225	1 ไบต์
Boolean	เป็นข้อมูลทางตรรก : จริง (TRUE), เท็จ (FALSE)	2 ไบต์
Integer	เป็นจำนวนเต็มระหว่าง -32768 ถึง 32767	2 ไบต์
Long	เป็นจำนวนเต็มระหว่าง -2147483648 ถึง 2147483647	4 ไบต์
Singer	เป็นเลขทศนิยมระหว่าง -3402823E38 ถึง -1401298E-4 สำหรับค่าลบ และ -1401298E-45 ถึง 3402823E38 สำหรับค่าบวก	4 ไบต์
Double	เป็นเลขทศนิยมระหว่าง -1.79769313486232E308 ถึง -4940656458 – 41247E – 324 สำหรับค่าลบ และ 494065645841247E – 324 ถึง 1.79769313486232E308 สำหรับค่าบวก	8 ไบต์
Currency	เป็นเลขที่มีค่าตั้งแต่ -922337203685477.5808 ถึง 922337203685477.5807	8 ไบต์
Date	เป็นวันที่ตั้งแต่ 1 มกราคม ค.ศ. 100 ถึง 31 ธันวาคม ค.ศ.9999	8 ไบต์
Object	เป็นข้อมูลที่อ้างอิงออบเจ็กต์จึงเก็บแอดเดรสของออบเจ็กต์ไว้	4 ไบต์
String	เก็บสตริง หรือข้อความที่เรียงต่อกัน	64 KB หรือ 2 MB
Variant	เป็นข้อมูลชนิดพิเศษที่เก็บค่าได้ทุกรูปแบบ (รวมไปถึงค่าพิเศษ 16 ไบต์ ต่าง ๆ (ที่มีตัวเลข) เช่น EMPTY , NULL เป็นต้น)	

จากข้างต้นจะเห็นว่าข้อมูลชนิด Variant เป็นข้อมูลที่สามารถแทนชนิดข้อมูลอื่นๆ ได้ทุกรูปแบบ โดย Visual Basic จะตัดสินใจเองว่าควรจะเก็บข้อมูลที่เป็น Variant แบบใดโดยยึดสภาวะรอบข้างในการตัดสินใจ (Context Decision)

ข้อมูลชนิด Variant แม้จะเก็บค่าได้หลายรูปแบบ แต่ว่าต้องใช้เนื้อที่มากกว่าข้อมูลชนิดอื่นๆ เพราะฉะนั้นการเก็บข้อมูลด้วย Variant จึงต้องใช้ทรัพยากรของระบบมากกว่าปกติ เพราะนอกจากจะเสียเนื้อที่มากยังต้องเสียเวลาให้ Visual Basic เลือดว่าข้อมูล Variant ที่เก็บนั้นควรเป็นข้อมูลแบบใด

2.2.5 ตัวแปร และการประกาศค่า (Variable and Variable Declaration)

ในการใช้ข้อมูลชนิดต่างๆ เราต้องกำหนดตัวแปรเพื่อให้เข้ากับชนิดข้อมูลที่เราต้องการ โดยเราจะต้องประกาศชื่อตัวแปรให้ Visual Basic ได้รู้จักโดยใช้คำสั่ง Dim

Dim ชื่อตัวแปร As ชนิดตัวแปร, ชื่อตัวแปร As ชนิดตัวแปร,...

หากเราไม่มีการระบุชนิดของตัวแปร Visual Basic จะกำหนดให้ชนิดตัวแปรนั้นเป็น Varianในบางครั้งการประกาศค่าตัวแปร อาจจะใช้ตัวอักษรพิเศษต่อท้ายชื่อตัวแปร (Type-declaration character) เพื่อระบุชนิดของตัวแปรได้ เช่นให้ % ต่อท้ายตัวแปรที่มีชนิดเป็นจำนวนเต็มหรือ $ ต่อท้ายตัวแปรที่มีชนิดเป็นสตริง

```
Dim myVar%, yourVar$
```

จะมีค่าหมือนกับ

```
Dim myVar  As Integer,  yourVar As String
```

แต่วิธีดังกล่าวไม่ขอแนะนำให้ใช้ เพราะอาจเกิดความสับสนในการใช้งานจริง

User-Define Type

นอกจากตัวแปรที่ Visual Basic มีให้เราใช้งาน เราสามารถสร้างตัวแปรชนิดพิเศษของเราขึ้นมาเองได้เช่นเดียวกัน แต่ชนิดตัวแปรพิเศษสร้างขึ้นใหม่ก็ยังต้องประกอบด้วยชนิดตัวแปรดั้งเดิมที่มี โดยมีขั้นตอนดังนี้

1. ประกาศตัวแปรที่เรากำหนดเองในโมดูลพิเศษ โดยเลือกเมนู Project > Module เพื่อเพิ่มโมดูล
2. กำหนดชนิดตัวแปรดดยใช้คำสั่ง Type

ค่าคงที่ (Constant)

ในกรณีที่เราต้องการใช้ข้อมูลที่มีค่าไม่เปลี่ยนแปลงตลอดเวลา เราจะใช้การกำหนดให้เป็นค่าคงที่ ซึ่งค่าคงที่ แบ่งได้เป็น 2 ประเภท ได้แก่

User Define Constant

เป็นค่าคงที่ที่เราเป็นผู้กำหนดเอง โดยใช้คำสั่ง Const เป็นการประกาศค่าคงที

Const ชื่อค่าคงที่ As ชนิดข้อมูล = ค่าคงที่นั้น

Pre-Defined Constant

เป็นค่าคงที่ๆ Visual Basic ได้กำหนดค่าไว้แล้ว เราสามารถนำมาใช้ได้ทันทีโดยเราไม่ต้องประกาศค่าคงที่เหมือนกับ Visual Basic เวอร์ชั่นเก่าๆ ที่ต้อง Add Fine Constant.txt เข้ามาด้วย) ในระหว่างการเขียนโค้ด Visual Basic จะมีการตรวจสอบว่ามี Pre-Defined Constant ใดที่เกี่ยวข้อง ซึ่ง Visual Basic จะนำค่าคงที่ๆเหมาะสมมาแสดงให้เรา เพื่อเลือกในขณะที่เขียนโปรแกรมเอง ซึ่งเราเรียกความสามารถนี้ว่า Auto Line Member

Auto Line Member จะใช้งานได้ เราต้องกำหนดใน Option โดยการเลือกเมนู Tool > Options แล้วเลือกเก็บแท็บ Editor ดังเช้น การกำหนด Auto List Member ใน Option

2.2.6 การตัดสินใจ(Decision)

ในการเขียนโปรแกรม บางครั้งจำเป็นต้องตัดสินใจให้แอพพลิเคชั่นทำงานอย่างใดอย่างหนึ่งจากทางเลือกที่มีจะมีให้เลือกมากกว่า 1 ทางเลือก โดยการตัดสินใจในโปรแกรม (ไม่ว่าจะใช้ภาษาคอมพิวเตอร์ภาษาใดก็ตาม) จะมี 2 ประเภท

1. ตัดสินใจเลือกจากทางเลือก 2 ทางเลือก
2. ตัดสินใจเลือกมากกว่า 2 ทางเลือก

If...Then...Else : ตัดสินใจเลือกจาก 2 ทางเลือก

เราจะใช้ If...Then...Else ในการตัดสินใจเมื่อมีทางเลือกให้เลือก 2 ทาง เป็นการตัดสินเลือกจากทางเลือก 2ทางเลือก

```
If < ทดสอบเงื่อนไขว่าจริงหรือเท็จ > Then
        ถ้าเป็นจริงให้ทำงานหลังคำว่า Then
Else
        ถ้าเป็นเท็จให้ทำงานหลังคำว่า Else
End If
```

ในบางครั้งเราก็อาจจะใช้ If...Then...Else เพื่อตัดสินใจเลือก จากทางเลือกมากกว่า 2 ทางเลือก

```
If < ทดสอบเงื่อนไขว่าจริงหรือเท็จ > Then
        ถ้าเป็นจริงให้ทำงานหลังคำว่า Then
Else If < ทดสอบเงื่อนไขว่าจริง หรือ เท็จ > Then
        ถ้าเป็นจริงให้ทำงานหลังคำว่า Then เสมอ
End If
        :
        :
        ถ้าเป็นเท็จให้ทำงานหลังคำว่า Else
End If
```

Select...Case : ตัดสิใจเลือกมากกว่า 2 ทางเลือก

```
Select Case < ทดสอบเงื่อนไข>
        Case เงื่อนไขแรก      : < ทำตามเงื่อนไขแรก>
        Case เงื่อนไขที่สอง   : < ทำตามเงื่อนไขที่สอง>
        :
        :
        Case สุดท้าย          : < ทำงานตามเงื่อนไขสุดท้าย>
        Case Else             : <เมื่อไม่ตรงกับเงื่อนไขใดๆเลย ทำงานหลังคำว่า Else >
End Select
```

การวนซ้ำ (Iteration)

การเขียนโปรแกรมเราอาจจำเป็นต้องสั่งให้แอพพลิเคชั่นทำงานวนซ้ำตามจำนวนครั้งที่ต้องการได้ ซึ่งรูปแบบของการวนซ้ำมี 2 รูปแบบได้แก่

1. การวนซ้ำด้วยจำนวนรอบที่แน่นอน
2. การวนซ้ำด้วยจำนวนรอบที่ไม่แน่นอน

For ...Next : วนซ้ำด้วยจำนวนรอบที่แน่นอน

```
เป็นรูปแบบการวนซ้ำที่เราสามารถกำหนดรอบของการวนซ้ำได้แน่นอน
For ตัวแปรใช้นับจำนวนรอบ = จำนวนรอบเริ่มต้น To จำนวนรอบสุดท้าย <ทำตามคำสั่ง>
```

Next ตัวแปรที่ใช้นับจำนวนรอบ

จะเห็นได้ว่าเราต้องใช้ตัวแปร 1 ตัว ทำหน้าที่เป็นตัวนับรอบในการวนซ้ำ ซึ่งตัวนับนี้สามารถนับได้ทั้งแบบเดินหน้าและถอยหลัง

While...Wend

เป็นรูปแบบการวนซ้ำที่ไม่สามารถกำหนดเงื่อนไขการวนซ้ำได้ โดยอาศัยการตรวจสอบเงื่อนไขก่อนการวนซ้ำว่าต้องวนซ้ำอีกหรือไม่

```
While < ทดสอบเงื่อนไขจริงหรือเท็จ >
                <ถ้าเงื่อนไขเป็นจริง ให้ทำงานตามคำสั่ง>
Wend
```

สำหรับเงื่อนไขการหลุดจากการวนซ้ำนั้นจะต้องทำให้เงื่อนไขเป็นเท็จเท่านั้นจึงจะหลุด

Do/While...Until/Loop : วนซ้ำด้วยจำนวนรอบที่ไม่แน่นอน

นอกเหนือจากการ while...Wend ในการวนซ้ำแล้ว เรายังใช้การวนซ้ำแบบตรวจสอบเงื่อนไขอีกหลายๆแบบได้แก่

```
Do While < ทดสอบเงื่อนไขจริงหรือเท็จ >
        <ถ้าเงื่อนไขเป็นจริง ให้ทำงานตามคำสั่ง>
Loop
```

```
Do Until    < ทดสอบเงื่อนไขจริงหรือเท็จ >
        <ถ้าเงื่อนไขยังเป็นเท็จอยู่ให้ทำงานตามคำสั่ง>
Loop
```

```
    Do
        <ทำงานตามคำสั่ง>
Loop While <ทดสอบเงื่อนไขจริงหรือเท็จ  ถ้าเป็นจริงให้กลับไปทำงานอีกรอบ>
```

```
    Do
        <ทำงานตามคำสั้ง ก่อน 1 รอบ >
```

```
Loop Until <ทดสอบเงื่อนไขจริงหรือเท็จ ถ้าเป็นเท็จให้กลับไปทำงานอีกรอบ>
```

ข้อมูลตัวที่ 1
ข้อมูลตัวที่ 2
ข้อมูลตัวที่ 3
ข้อมูลตัวที่ 4
ข้อมูลตัวที่ 5

จะเห็นว่าเราสามารถตรวจสอบเงื่อนไขได้ 2 แบบ

*While จะหลุดจากการวนซ้ำได้เมื่อทดสอบเงื่อนไขแล้วเป็นเท็จ (Fasle)

*Until จะหลุดจากการวนซ้ำได้เมื่อทดสอบเงื่อนไขแล้วเป็นจริง (True)

แล้วเป็นจริง (True) นอกจากนี้แล้วยังสามารถวางตำแหน่งการตรวจสอบเงื่อนไขได้ว่าจะตรวจสอบก่อนการวนซ้ำหรือหลังจากการวนซ้ำ ควรตรวจสอบเงื่อนไขของการวนซ้ำให้ดี เพราะมิฉะนั้นจะเกิดการวนซ้ำแบบไม่มีที่สิ้นสุดและเกิดข้อผิดหลาดได้

กระโดดจากการวนซ้ำ

ในบางครั้งเราจำเป็นต้องกระโดดออกจากการวนซ้ำ ด้วยเงื่อนไขบางอย่างก็สามารถทำได้โดยใช้คำสั่ง Exit For, Exit Loop

อาร์เรย์ และไดนามิกอาร์เรย์ (Array and Dynamic Array)

ถ้าเราจำเป็นต้องเก็บข้อมูลที่มีชนิดเดียวกันไว้เป็นชุด โดยการเก็บไว้ในตัวแปรชื่อเดียว เราจะใช้อาร์เรย์ในการเก็บ

ในการประกาศใช้อาร์เรย์เรากำหนดได้ดังนี้

```
Dim ชื่อ อาร์เรย์ (ขอบเขตล่าง TO ขอบเขตบน,...) As ชนิดของข้อมูลที่เก็บในอาร์เรย์
```

ตัวอย่างการประกาศอาร์เรย์ในรูปแบบต่างๆ

```
Dim myArray (1 To 5) As String
Dim yourArray (-4 To 5) As String
Dim ourArray (1 To 5,-4 To 5) as String
```

ในการประกาศใช้อาร์เรย์ เราจะต้องกำหนดขนาดของอาร์เรย์ว่าจะเก็บข้อมูลได้กี่ค่า (Array Size) และจะเห็นว่าอาร์เรย์ที่เราใช้งานกันนั้นสามารถเก็บได้หลายมิติ (Dimension) เช่น อาร์เรย์ 2 มิติ, อาร์เรย์ 3 มิติ เป็นต้น

สำหรับการอ้างอิงถึงค่าในอาร์เรย์จะทำได้โดยการระบุอินเด็กซ์ของอาร์เรย์ (Index of Array)

แม้ว่า Visual Basic จะไม่ได้จำกัดมิติของอาร์เรย์ที่ประกาศไว้ แต่ส่วนใหญ่อาร์เรย์ที่เราใช้งานมีมิติไม่มากนัก ซึ่งทำให้การอ้างอิงและการทำความเข้าใจเป็นไปได้ไม่ลำบาก

ไดนามิกส์อาร์เรย์ (Dynamic Array)

ในบางครั้งของการทำงานโดยใช้อาร์เรย์เราอาจประสบปัญหาต่อไปนี้

1. ไม่สามารถจะบอกขนาดของอาร์เรยีได้อย่างชัดเจน

2. การประกาศขนาดของอาร์เรย์ที่มีขนาดใหญ่ แต่เวลาใช้งานจริงกลับใช้ได้ไม่เต็มที่ซึ่งจะเกิดการสูญเสียทรัพยากรระบบโดยไม่จำเป็น และอาจทำให้แอพพลิเคชั่นของระบบทำงานช้าลงด้วย
3. ต้องประกาศอาร์เรย์ขนาดใหญ่แต่ยังไม่ถูกใช้งานทันที ทำให้หน่วยความจำถูกนำไปครอบครองโดยยังไม่มีการใช้งานดังนั้นไดนามิกส์อาร์เรย์จึงเกิดขึ้น โดยไดนามิกส์อาร์เรย์จะยอมให้เรากำหนดขนาดของอาร์เรย์ในขณะรันแอพพลิเคชั่น โดยเราจะเขียนโปรแกรมเพื่อกำหนดขนาดที่เหมาะสมให้กับอาร์เรย์

2.2.7 โปรแกรมย่อย (Procedure)

ในการเขียนโปรแกรม ย่อมเกิดการทำงานที่ซ้ำๆกัน เราสามารถลดการทำงานที่ซ้ำๆนั้นโดยการเขียนโปรแกรมย่อยเก็บไว้ เมื่อเราต้องการทำงานแบบเดิมอีก เราก็เพียงแต่เรียกโปรแกรมย่อยนั้นมาใช้งานทำให้ลดเวลาการเขียนโปรแกรม และลดความยุ่งยากของโปรแกรมที่เขียนด้วยโปรแกรมย่อยแบ่งออกเป็น 2 ประเภท ตามลักษณะของการคืนค่ากลับมาหลังจากจบโปรแกรมย่อย

Sub (Sub routine)

ซับรูทีน เป็นโปรแกรมย่อยที่เราเรียกขึ้นมา เมื่อโปรแกรมย่อยทำงานเสร็จแล้วจะไม่มีการคืนค่าใดๆกลับมายังผู้ที่เรียกใช้งาน ซึ่งสับรูทีนมีโครงสร้างดังนี้

```
Sub ชื่อสับรูทีน (รายการอาร์กิวเมนต์มีหรือไม่มีก็ได้๗)

        <คำสั่งใน Visual Basic>

END Sub
```

เมื่อประกาศสับรูทีนแล้วเราก็สามารถเรียกใช้งานสับรูทีนได้

Function

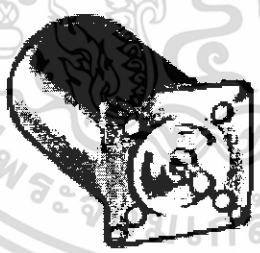
ฟังก์ชัน เป็นโปรแกรมย่อยที่ต้องคืนค่ากลับคืนมาหาผู้เรียกใช้ หลังจากจบโปรแกรมการทำงานเสร็จ ซึ่งฟังก์ชันมีโครงสร้างดังนี้

```
Function ชื่อฟังก์ชัน (รายการอาร์กิวเมนต์ไม่ต้องมีก็ได้)

        <คำสั่งใน Visual Basic>

        ชื่อฟังก์ชัน = ค่าที่คืนกลับ
End Function
```

ฟังก์ชันนั้นจะต้องมีการคืนค่ากลับให้ผู้เรียกใช้งาน โดยเราจะระบุชื่อฟังก์ชันไว้ในบรรทัดสุดท้ายก่อนจบการทำงานของฟังก์ชันเสมอ และกำหนดค่าเพื่อให้เป็นค่าที่คืนกลับผู้เรียกใช้

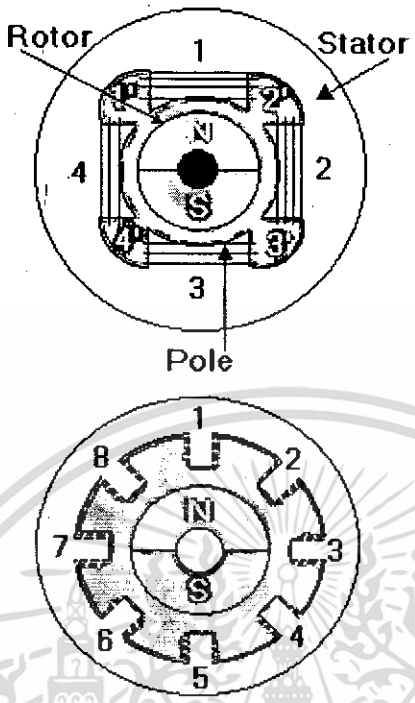
2.3 สเต็ปเปอร์มอเตอร์

Step Moter เป็นมอเตอร์ที่มีลักษณะเมื่อเราป้อนไฟฟ้าให้กับมอเตอร์ทำให้หมุนเพียงเล็กน้อยตามเส้นรอบวงและหยุด ซึ่งต่าง จากมอเตอร์ ทั่วไปที่จะหมุนทันทีและตลอดเวลาเมื่อป้อนแรงดันไฟฟ้าข้อดีของสเต็ปมอเตอร์ สามารถกำหนด ตำแหน่งของการหมุนด้วยตัวเลข(องศาหรือระยะทาง) ได้อย่างละเอียดโดย ใช้คอมพิวเตอร์หรือ ไมโครคอนโทรลเลอร์เป็น เครื่องกำหนดและจัดเก็บตัวเลข



รูปที่ 2.9 สเตปมอเตอร์

โครงสร้างของขั้วแม่เหล็กบนสเตเตอร์ทำมาจากแผ่นเหล็กวงแหวนที่มีซี่ยื่นออกมาประกอบกันเป็นชั้นๆ โดยที่แต่ละซี่นั้นจะมีคอยล์(ขดลวด)พันสวมอยู่ เมื่อมีการป้อนกระแสผ่านคอยล์ทำให้เกิดสนามแม่ เหล็กไฟฟ้า(Electromagnetic) ดูดังรูปด้านล่างนี้ จะแสดงถึงองค์ประกอบที่กล่าวมา

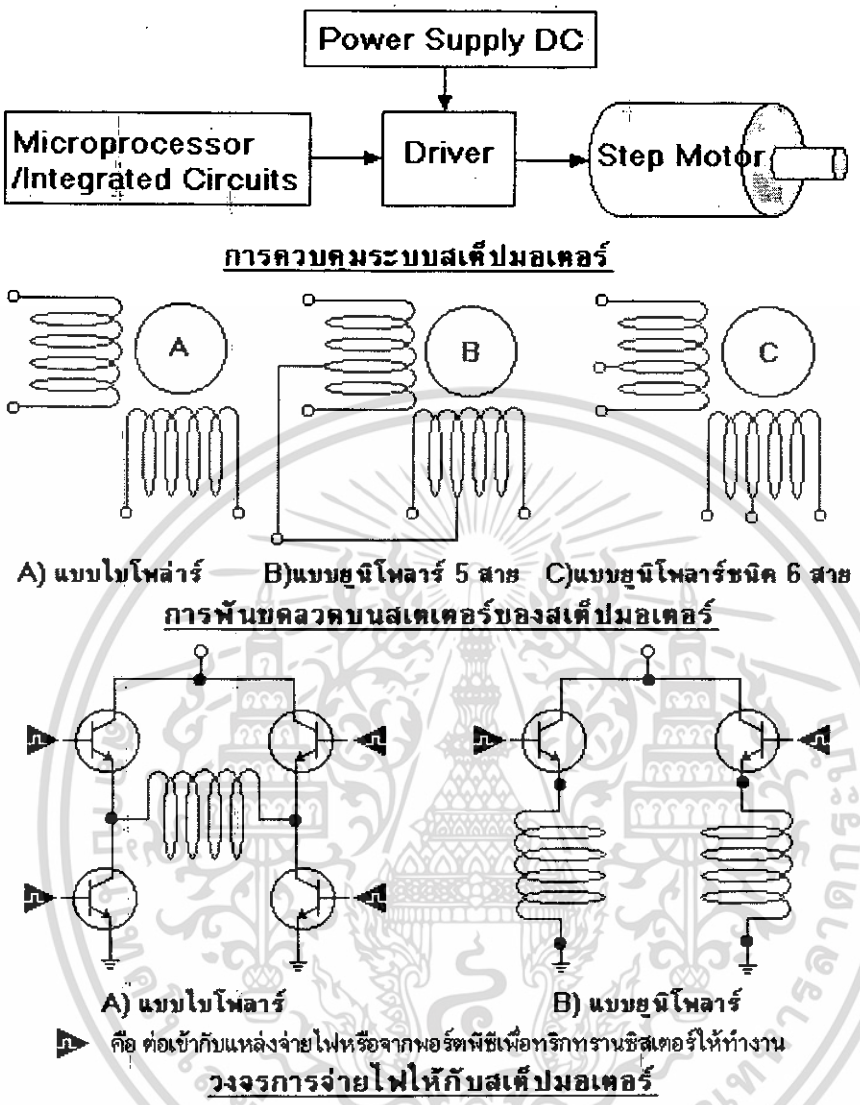


รูปที่ 2.10 โครงสร้างของขั้วแม่เหล็ก

ในที่นี้ซึ่งถ้าเราเพิ่มจำนวนของขั้วแม่เหล็กมากขึ้นจะเพิ่มจำนวนของสเต็ปต่อวงจรรอบมากขึ้นตามด้วย ลองดูตามรูปด้านบน

ลักษณะการนำไปใช้งาน สเต็ปมอเตอร์ ใช้งานลักษณะ Open Loop System แปลเป็นภาษาไทย ระบบเปิด คือ สเต็ปมอเตอร์สามาทำงานได้โดยไม่ต้องมีการ ป้อนค่าพารามิเตอร์กลับมา (Feed back) แต่ทุกวิธีที่ต้องการกำหนดตำแหน่งที่แน่นอนนั้นล่ะ จะต้องการป้อนกลับไปยังระบบและตัวบอก ตำแหน่งว่าถูกต้องหรือผิดผลาดให้รับทราบ

ดังเช่นวิธีที่ใช้กับสเต็ปมอเตอร์ คือเรานำลิมิทสวิตช์ติดตามตำแหน่งที่จะตรวจจับ เมื่อสเต็ปมอเตอร์ เริ่มหมุนแล้วหมุนไปจนถึงตำแหน่งของสวิตช์ตรวจจับสัญญาณ สวิตช์ทำงานก็จะป้อนกลับไปสู่ระบบ ซึ่งก็จะทำให้รู้การทำงานของสเต็ปมอเตอร์ตลอด ตัววงจรไมโครคอนโทรลเลอร์เองจะมีจุดอ้างอิง ไว้ให้เริ่มต้นการทำงานและอ้างอิงตำแหน่งได้ถูกต้อง



รูปที่ 2.11 วงจรการจ่ายไฟให้กับสเตปมอเตอร์

โดยแนวทางสเต็ปมอเตอร์เป็นอุปกรณ์จำพวกเชิงกลทางไฟฟ้า โดยมีกรุ๊ปของไบนารี่โวลท์เตท เป็นอินพุตและการเคลื่อนที่ แบบเชิงมุมเป็นเอาท์พุต หรือว่าหมุนทีละสเต็ปซึ่งอยู่ระหว่าง 0.1 - 30 องศา อยู่ที่โครงสร้างของสเต็ปมอเตอร์ โดยตามสัญญาณ พัลซ์ที่จ่ายให้กับขดสเตเตอร์ทำให้เกิดแรงผลักแก่โรเตอร์หมุนไป สเต็ปมอเตอร์มีขดลวดหลายชุดในที่นี้เราเรียกว่า Phase (เฟส) ดังนั้นสัญญาณที่ต่อเนื่องเป็น Sequence(ซีเควน) ลักษณะของBinary(ไบนารี่) ซึ่งจะต้องไปผ่านวงจร Driver(ไดรเวอร์) ก็จะ ทำให้โรเตอร์หมุนไปอย่างต่อเนื่อง ที่กล่าวมาสามารถดูได้จากรูปด้านบนชื่อ การควบคุมสเต็ปมอเตอร์

คราวนี้ให้ดูที่รูปชื่อการพันขดลวดบนสเตเตอร์ของสเต็ปมอเตอร์ จะเห็นว่าการพันมีด้วยกัน 2 วิธี คือ แบบ Bipolar(ไบโพล่าร์กับ แบบ Unipolar(ยูนิโพล่าร์)

แบบ Bipolar

จะมีการพันขดลวดหนึ่งขด(จะกี่รอบก็แล้วแต่ สเป็กใช้งานนะครับ)ในแต่ขั้วแม่เหล็กของสแตอร์ โดยขั้วแม่เหล็กที่เกิดขึ้น ที่สเตเตอร์จะถูกกำหนดโดยทิศทางของการไหลของกระแสไฟฟ้า ซึ่งสามารถทำให้เกิดขั้วแม่เหล็กในทิศทางตรงกันข้ามได้เพียง การกลับทิศทางของการไหลในกระแสไฟฟ้า โดยมาจากการควบคุมของวงจรสวิทชิ่งให้กลับขั้วไฟฟ้า

แบบ Unipolar

แบบนี้มี 2 ขด บนแต่ละขั้วแม่เหล็กของสเตเตอร์ ทำให้แต่ละขดลวดเกิดขั้วแม่เหล็กในทิศทางตรงกันข้าม เช่นกันครับการกลับทิศทางขั้วแม่เหล็กทำได้โดยใช้วงจรสวิทชิ่งให้สลับหนึ่งไปยังอีกขั้วหนึ่งแทนกัน

พื้นฐานการสวิทชิ่งดูรูปด้านบนที่ชื่อวงจรการจ่ายไฟให้กับสเต็ปมอเตอร์ครับ ท่านคงจะถามในใจแล้วว่าการพันขดลวดทั้ง 2 แบบที่กล่าวมา มันต่างกันอย่างไร สั้นๆครับ แบบยูนิโพล่าร์จะทำให้เกิดแรงบิดน้อยกว่าแบบไบโพล่าร์ แล้วก็จะต้องมีคำถามตามมาอีกว่าแล้วถ้าไปซื้อหรือหามาใช้งานจะรู้ได้จากตรงไหน ก็สังเกตุจาก สายไฟที่ต่อมาจากตัวสเต็ปมอเตอร์ซึ่งแบบไบโพลาร์จะมี 4 สาย ส่วนเป็นแบบยูนิโพล่าร์จะมี 5 สายหรือ 6 สาย

การสั่งงานควบคุมการหมุนของสเต็ปมอเตอร์

การควบคุมและสั่งงานให้สเต็ปมอเตอร์ทำงาน ไปทีละสเต็ปสามารถทำได้โดยการจ่ายกำลังไฟไปยังขดลวด ในแต่ละขอบนสเตเตอร์ โดยการป้อนจะทำในลักษณะเป็นลำดับหรือเรียกว่า ซีเควนเวียลในลูปที่ถูกต้อง ซึ่งจะแบบ ได้เป็น 3 รูปแแบ คือ แบบเวฟ(wave) แบบ 2 เฟส(2 phase) และแบบครึ่งสเต็ป (half step) ทั้ง 3 แบบนี้ก็จะมีข้อดีและข้อเสียต่างกันออกไป

แบบเวฟ (wave)

จะเป็นการกระตุ้นแบบที่ง่ายที่สุด ซึ่งจะทำการกระตุ้นขดลวดทีละขดในเวลาหนึ่งๆเรียงกันไป ตัวอย่างเช่น ขดที่ 1 , 2 , 3 , 4 ,1, 2 , 3 , 4 เป็นลำดับอย่างนี้ หรือ ขด 1 , 4 , 3 , 2 , 1 , 4 , 3 , 2 เป็นลำดับกันไป ทั้งนี้ขึ้นอยู่กับทิศทางที่เราต้องให้มอเตอร์หมุนไป วงจรที่นำมากระตุ้นนั้นจะมีราคาค่อนข้างจะถูกกว่าและง่ายกว่า ดังในรูปของวงจรการจ่ายไฟ ที่อยู่ด้านบนนั้น

เราสามารถเขียนขั้นตอนการทำงานเป็นตารางออกมาได้ดังนี้

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

รูป 2.12 การขับสเตปมอเตอร์แบบเวฟ

แบบ 2 เฟส(2 Phase)

แบบนี้ก็จะคล้ายกับการกระตุ้นในแบบเวฟแต่จะต่างกันตรงที่ แบบ 2 เฟส จะกระตุ้นทีละ 2 ขด ที่อยู่ใกล้กันใน เวลาเดียวกัน และจะเรียงลำดับกันไป ดังเช่นแบบเดียวกับแบบเวฟครับ จะยกตัวอย่างการกระตุ้นขดลวดในลักษณะ ซีเควนให้เดูดังนี้ 12,23,34.41,12,23,34,41 เรียงลำดับกันไปเรื่อยๆ หรือจะเป็น 14,43,32,21,14,43,32,21 เรียงกันไปเรื่อยๆเช่นกัน ถ้าจะมากล่าวถึงข้อดีข้อเสียของแบบ 2 เฟส แล้วมีดังนี้

ข้อดี การที่เราจะเพิ่มจำนวนขดลวดที่ถูกกระตุ้นจะทำให้แรงบิดได้มากกว่า แบบเวฟ ซึ่งโรเตอร์จะหมุนด้วยแรง ดึงแบเต็มๆแรงจาก ทั้ง 2 ขดลวดที่กระตุ้นพร้อมกัน

ข้อเสีย แบบ 2 เฟส จะกระตุ้นขดลวดนั้นต้องใช้กำลังไฟมากขึ้นเป็น 2 เท่าของแบบเวฟ ก็เป็นไปตามธรรมชาติ ได้อย่างก็ต้องเสียอย่าง เราสามารถเขียนลำดับการกระตุ้นของขดลวดแบบ 2 เฟส ได้ดังในตารางต่อไปนี้

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

รูป 2.13 การขับสเตปมอเตอร์แบบ 2 เฟส

แบบครึ่งสเต็ป

แบบนี้แบบรูปแบบผสมผสานของการกระตุ้นระหว่าง แบบเวฟ กับ แบบ 2 เฟส เพื่อให้จำนวนรอบของสเต็ปให้ มากขึ้นเป็น 2 เท่า ซึ่งในระบบนี้จะทำการกระตุ้นขดลวดเรียงกันไปเรื่อยๆเป็นลำดับ ดังจะยกตัวอย่างต่อไปนี้ 1,12,2,23,3,34,4,41,1,12,2,23,3,34,4,41,1 เป็นลำดับอยู่อย่างนี้เรื่อยไปครับ ถ้าเราจะกลับทิศทางการหมุนก็จะไดเป็นดังนี้ครับ
1,41,4,43,3,32,2,21,1,41,4,43,3,32,2,21,1 เป็นลำดับ กันไปเราจะมาพูดกันถึงข้อดีและข้อเสียของการกระตุ้นแบบครึ่งสเต็ปกัน

ข้อดี การกระตุ้นแบบนี้จะให้แรงบิดที่เพิ่มมากขึ้น เนื่องจากช่วงสเต็ปที่มีระยะสั้นลงออีกประการหนึ่งแต่ละ สเต็ปเกิดแรงดึงจากขดลวด 2 ขดที่ถูกกระตุ้นพร้อมกันเป็นผลให้ค่าตำแหน่งความถูกต้องมากขึ้นไปด้วย

ข้อเสีย ก็คงจะเช่นเดียวกับแบบ 2 เฟสล่ะครับ ที่ต้องจ่ายกำลังไฟเป็น 2 เท่าของแบบเวฟหรือจะใช้เท่ากับแบบ 2เฟส นั้นเอง

ดังนั้นเราสามารถนำลำดับการทำงานของ แบบครึ่งเฟส ในรูปของตารางได้ดังนี้

Step No.	Phase 1	Phase 2	Phase 3	Phase 4
1	ON			
2	ON	ON		
3		ON		
4		ON	ON	
5			ON	
6			ON	ON
7				ON
8	ON			ON
9	ON			
10	ON	ON		

รูป 2.14 การขับสเตปมอเตอร์แบบครึ่งเฟส

2.4 มาตรฐาน RS -232

RS-232 เป็นมาตรฐานการเชื่อมต่อข้อมูลแบบอนุกรมที่มีคนนิยมใช้มากที่สุด กำหนดโดย EIA (Electronics Industry Association) หรือสมาคมผู้ประกอบการอุตสาหกรรมอิเล็กทรอนิกส์ ของอเมริกา ตั้งแต่ปี 1969 โดยมีจุดเริ่มต้นจากความต้องการที่จะกำหนดมาตรฐานการเชื่อมต่อระหว่างคอมพิวเตอร์กับโมเด็มในสมัยนั้น ตัวมาตรฐานจะกำหนดสิ่งที่เกี่ยวข้องกับการเชื่อมต่อนี้ด้วยกันทั้งหมด 4 หัวข้อหลักๆ ด้วยกันคือ

1. คุณสมบัติทางไฟฟ้าของสัญญาณ
2. คุณสมบัติทางกลของการเชื่อมต่อ ซึ่งหมายถึงตัวคอนเน็กเตอร์นั่นเอง
3. หน้าที่การทำงานของวงจรสำหรับแลกเปลี่ยนข้อมูล
4. มาตรฐานการเชื่อมต่อสำหรับระบบสื่อสารเฉพาะอย่าง

RS-232-C เป็นมาตรฐาน RS-232 ที่มีการปรับปรุงแก้ไขจากมาตรฐานเดิม ซึ่งเราอาจคุ้นเคยกับชื่อนี้มากกว่า RS-232-A หรือ RS-232-B อันที่จริงแล้วยังมีมาตรฐาน RS-232-D ที่ใหม่กว่า RS-232-C โดยที่มีการเพิ่มข้อกำหนดของคอนเน็กเตอร์แบบ DB เข้าไปด้วย เช่น DB-25 ซึ่งในขณะนั้นสิทธิบัตรของตัวคอนเน็กเตอร์แบบนี้ได้หมดอายุลงพอดี จึงสามารถรวมข้อกำหนดเข้าไว้ได้ ลักษณะโดยทั่วไปของการเชื่อมต่อข้อมูลแบบอนุกรมตามมาตรฐาน RS-232 คือเป็นการสื่อสารข้อมูลแบบจุดต่อจุด ซึ่งเดิมทีเป็นการสื่อสารข้อมูลระหว่างคอมพิวเตอร์กับโมเด็ม ซึ่งจริงๆแล้วทั้งสองฝั่งจะเป็นอะไรก็ได้ การสื่อสารเป็นแบบสองทางพร้อมกัน (Full-duplex) โดยอาจใช้สายสัญญาณอื่นร่วมเพื่อทำแฮนด์เชค (Hand-shake) หรือไม่ก็ได้ มาตรฐาน RS-232 จำกัดความยาวสายไว้ที่ 50 ฟุต (หรือประมาณ 15 เมตร) สำหรับการส่งสัญญาณที่ความเร็ว 19,200 บิทต่อวินาที โดยที่ความยาวสายจะต้องสั้นลงถ้าต้องการสื่อสารที่ความเร็วสูงขึ้น และถ้ามีสัญญาณรบกวนมากๆ เช่นในโรงงาน หรือบริเวณใกล้เครื่องจักรที่เป็นแบบมีการสวิทช์สัญญาณไฟฟ้าที่กระแสสูงๆ ก็จะทำให้ต้องมีการลดความเร็วในการส่งสัญญาณลงหรือใช้สายที่สั้นลง

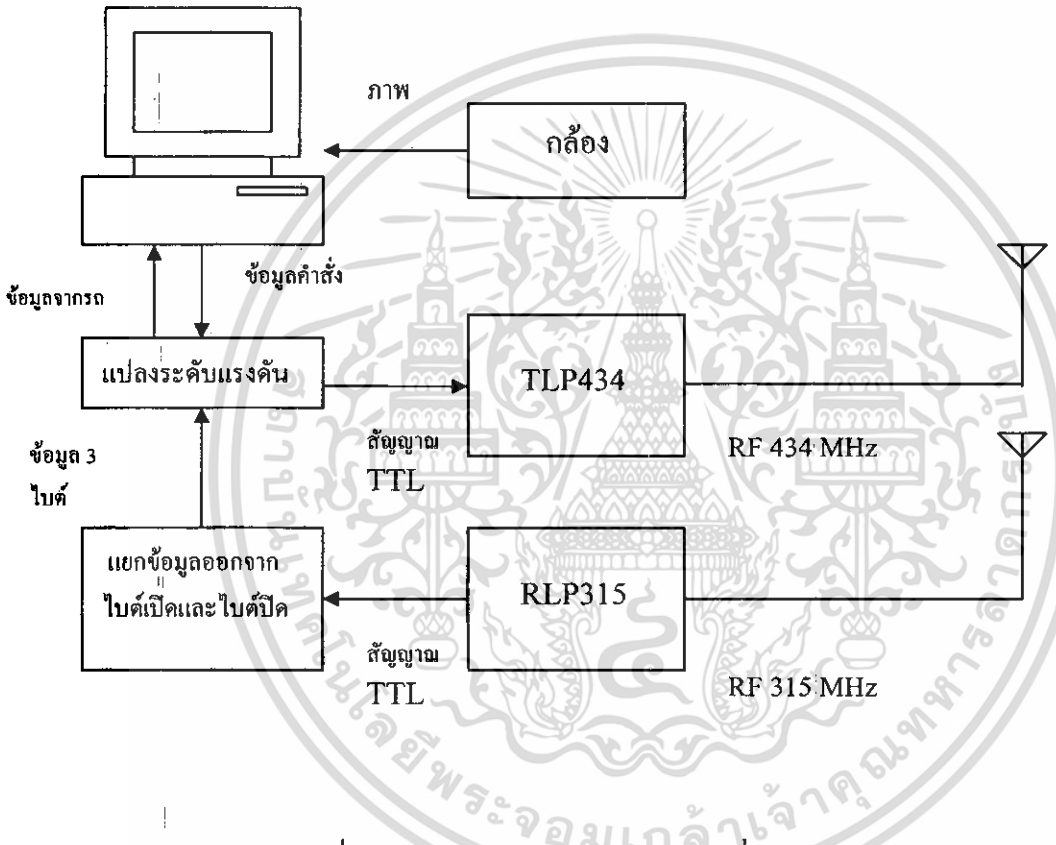
มาตรฐาน RS-422 หรือ RS-422-A ถูกกำหนดขึ้นโดยสมาคมผู้ประกอบการอุตสาหกรรมอิเล็กทรอนิกส์หรือ EIA เช่นเดียวกันกับมาตรฐาน RS-232 โดยมีจุดมุ่งหมายที่จะแก้ปัญหาเรื่องความยาวของสายสื่อสารโดยใช้การส่งสัญญาณแบบผลต่าง (Differential) แทนที่จะใช้การส่งสัญญาณแบบอ้างอิงกับจุดกราวนด์ (หรือสายดิน) เช่นเดียวกันกับ RS-232 การส่งสัญญาณแบบ Differential นี้ช่วยลดปัญหาสัญญาณรบกวนจาก 2 ปัจจัยด้วยกัน ได้แก่ ปัญหาแรงดันกราวนด์ 2 ฝั่งสายไม่เท่ากัน อันเกิดจากกระแสไฟฟ้าที่ไหลในสายกราวนด์ที่ยาวมากๆ ก่อให้เกิดความต่างศักดิ์ และปัญหาสัญญาณรบกวนที่เกิดจากแม่เหล็กไฟฟ้าเหนี่ยวนำในสาย โดยหากสายไฟที่ใช้ถูกตี

เกลียวและวางไว้ใกล้กัน เมื่อมีแรงดันเหนี่ยวนำจะปรากฏแรงดันรบกวนบนสายทั้งสองเท่าๆ กัน เป็นผลให้ ตัวรับที่อ่านความต่างศักดิ์ระหว่างสายอ่านข้อมูลได้เช่นเดิม ทั้งสองปัจจัยนี้เองเป็นสาเหตุที่ทำให้ความต้านทานต่อสัญญาณรบกวนของการสื่อสารแบบ RS-232 ด้อยกว่า RS-422) ตามมาตรฐาน RS-422 นี้จะใช้สายสัญญาณทั้งหมด 4 เส้น (2 เส้นสำหรับการส่งสัญญาณ และอีก 2 เส้นสำหรับรับสัญญาณ) และสามารถใช้ความยาวสายสัญญาณได้ถึง 4,000 ฟุต (หรือ 1.2 กม.) ที่ความเร็ว 100,000 บิทต่อวินาที และการสื่อสารเป็นแบบ 2 ทางพร้อมกัน (Full Duplex)

มาตรฐาน RS-485 กำหนดโดยสมาคมผู้ประกอบการอุตสาหกรรมอิเล็กทรอนิกส์หรือ EIA เป็นมาตรฐานการเชื่อมต่อสัญญาณแบบอนุกรม (Serial Communication) มีลักษณะการเชื่อมต่อเป็นแบบหลายจุด (Multi-point) หรือ Multi-drop สายสัญญาณที่ใช้มีทั้งแบบที่เป็น 2 สายและแบบที่เป็น 4 สาย การต่อแบบหลายจุดนี้ทำให้สามารถมองสายสัญญาณเป็นบัสนำสัญญาณได้ (Signal Bus) จำนวนคอมพิวเตอร์หรืออุปกรณ์ที่สามารถอยู่บน RS-485 บัสหนึ่งถูกกำหนดไว้ที่ 32 ตัว ในกรณีที่ต้องการเพิ่มจะต้องมีตัวทวนสัญญาณ (Signal Repeater) หรือใช้ตัวส่ง-รับสัญญาณที่มีอิมพิแดนซ์ (ความต้านทานเสมือน) สูงขึ้น ซึ่งเราอาจเพิ่มจำนวนจุดเชื่อมต่อขึ้นได้ถึง 128 จุด ความยาวของสายสัญญาณตามมาตรฐาน RS-485 นี้สามารถยาวได้ถึง 1.2 กม เช่นเดียวกับมาตรฐาน RS-422 แต่การสื่อสารจะเป็นแบบสองทางไม่พร้อมกัน (Half Duplex) มีเพียงคอมพิวเตอร์หรืออุปกรณ์ตัวเดียวเท่านั้นที่สามารถส่งสัญญาณออกได้ ณ เวลาหนึ่งๆ ส่วนที่เหลือจะเป็นผู้รับสัญญาณ หรือผู้ฟัง

บทที่3
การออกแบบวงจร

3.1 การออกแบบภาคส่ง



รูป 3.1 Block Diagram ของวงจรภาคส่ง

การควบคุมรถสำรวจจากเครื่องคอมพิวเตอร์ใช้โปรแกรมที่เขียนจากภาษาวิชวลเบสิก6 ส่งรหัสคำสั่งเป็นข้อมูลขนาด 1 ไบต์ พร้อม ไบต์เปิดและไบต์ปิด เมื่อมีการกดปุ่มคำสั่ง 1 ครั้งจะมีการส่ง 15 รอบ เพื่อให้แน่ใจว่าตัวรับสามารถรับคำสั่งได้ข้อมูลที่ส่งเป็นลำดับดังนี้

ไบต์เปิด	คำสั่ง	ไบต์ปิด (เช็คerror)

ไบต์เปิดทำให้ไมโครคอนโทรเลอร์ตัวรับรู้ว่าถัดจากไบต์เปิดเป็นคำสั่ง ในโค้ดไบต์เปิดคือ 250D และ 230 D ถัดจาก 230D คือคำสั่ง คำสั่งนั้นใช้ค่าดังนี้

ข้อมูล 3 H เป็นคำสั่งให้รถเดินหน้า ข้อมูล 4 H เป็นคำสั่งให้รถหมุนซ้าย

ข้อมูล 1 H เป็นคำสั่งให้รถถอยหลัง ข้อมูล 2 H เป็นคำสั่งให้รถหมุนขวา

ข้อมูล 5 H เป็นคำสั่งให้รถหยุด ข้อมูล 6 H เป็นคำสั่งให้กล้องหมุนขึ้น

ข้อมูล 7 H เป็นคำสั่งให้กล้องหมุนลงข้อมูล 8 H เป็นคำสั่งให้หมุนซ้าย

ข้อมูล 9 H เป็นคำสั่งให้หมุนขวา

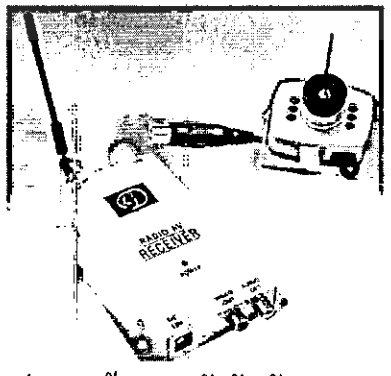
การ Xor ไบต์เปิดกับคำสั่ง เพื่อดูว่าเท่ากับ ไบต์ปิด หรือไม่ ถ้าไม่แสดงว่ามีสัญญาณรบกวน ตัวรับจะไปรอรับข้อมูลรอบใหม่ซึ่งการกดปุ่มคำสั่ง 1 ครั้ง จะส่ง 15 รอบ

ข้อมูลถูกส่งทางพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ โดยใช้สาย DB9 ระดับแรงดันของสัญญาณจากสาย DB9 สำหรับลอจิก 1 ประมาณ -12 V ลอจิก 0 ประมาณ +12 V ต้องเปลี่ยนเป็นระดับ TTL เพื่อนำไปใช้กับ AT89C52 โดยใช้ไอซี MAX232 มาแปลงสัญญาณ -12 Vเป็น 5 V และแปลงสัญญาณ +12 V เป็น 0 V จะได้สัญญาณ TTL จากนั้น IC TLP434A จะทำการมอดูเลตทางขนาดกับสัญญาณวิทยุเพื่อส่งออกอากาศทาง Antenna

ส่วนของข้อมูลที่ส่งมาจากรถจะถูกรับโดย ไอซี RLP315 จากนั้นข้อมูลจะถูกไมโครคอนโทรลเลอร์เช็ค error และตัดไบต์เปิดและปิดออก ส่ง ข้อมูล 4 ไบต์ คือ จำนวนรอบของล้อ อุณหภูมิ ทิศทางของรถจากทิศอ้างอิง ไปยังIC Max232 เพื่อแปลงระดับแรงดันและเข้าสู่พอร์ตอนุกรมของคอมพิวเตอร์เพื่อนำไปแสดงผล

3.1.1 กล้องและการ์ดจับภาพ

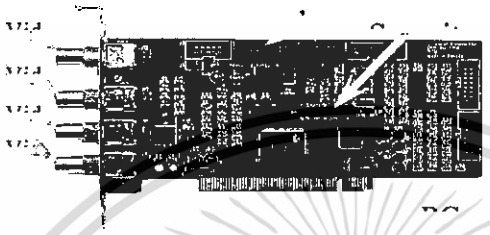
ในส่วนของกล้องและตัวรับสัญญาณใช้ของบริษัท TCTCOM รุ่น 208CWA และRC100A ดังรูป



รูป 3.2 กล้องและตัวรับสัญญาณภาพ

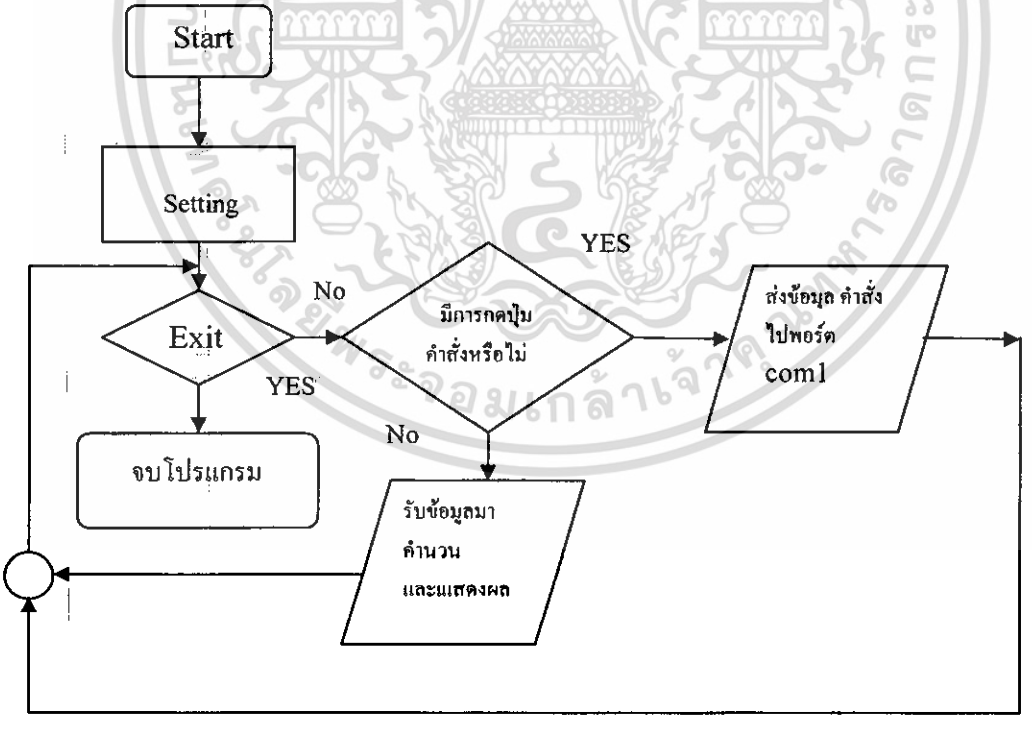
กล้องเป็นแบบ CMOS ใช้แรงดัน 8 V 200 mA มีตัวส่งสัญญาณอยู่ภายใน ตัวรับสัญญาณ ใช้แรงดัน 12 V 500 mA ในการใช้งานจะต่อสัญญาณ Video Out เข้ากับ อินพุตของการ์ดจับภาพ

การ์ดจับภาพ ใช้ PCI CARD PICO2000 25FR บริษัทเดียวกัน การ์ดสามารถรับกล้องได้ 4 ตัว ที่ 25 fr ในการใช้งานต้อง Download Driver มาติดตั้งด้วย

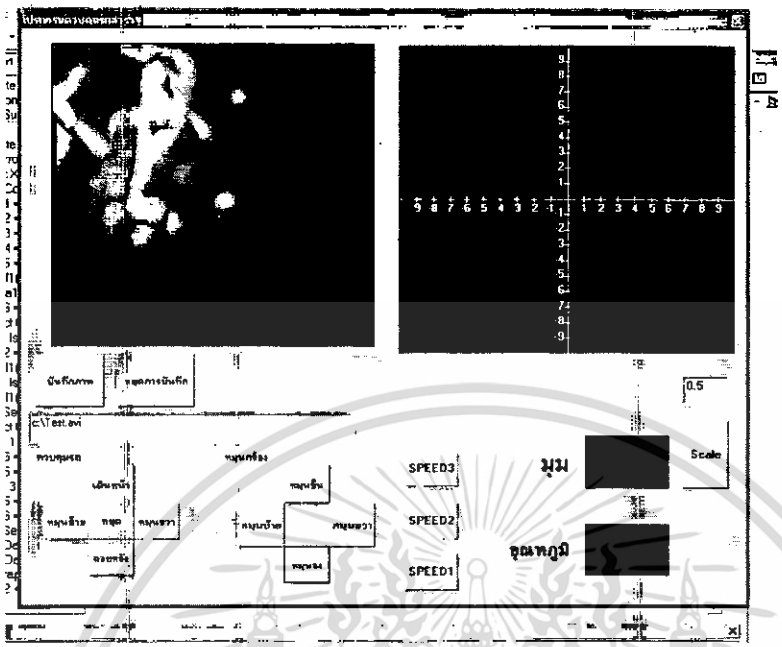


รูป 3.3 การ์ดจับภาพ

3.1.2 โปรแกรมวิชวลเบสิกควบคุมรถสำรวจ



รูป 3.4 Flow Chart ของโปรแกรมวิชวลเบสิก6ที่ใช้ควบคุมรถ

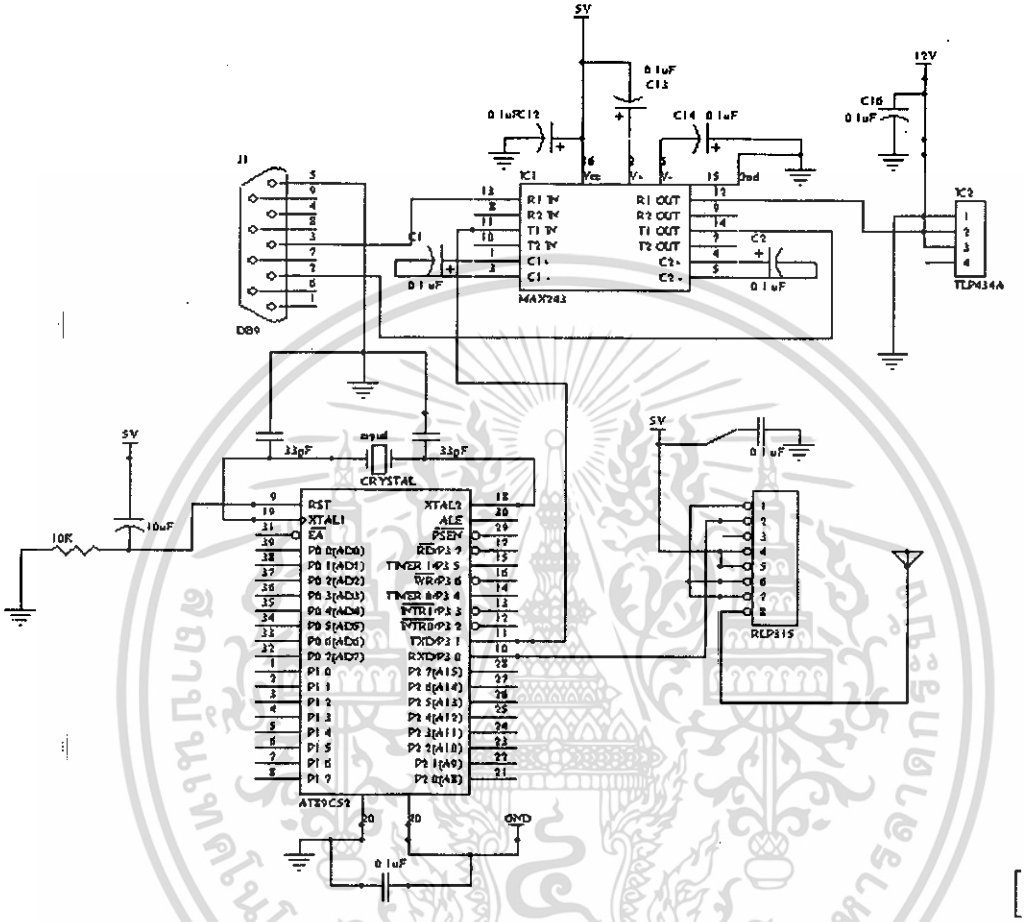


รูป 3.5 โปรแกรมควบคุมรถสำรวจ

จาก FlowChart ในส่วนของการ Setting สามารถเลือกได้ว่าจะติดต่อกับพอร์ตไหน เลือกอัตราการส่งข้อมูลซึ่งจะต้องเป็นอัตราเดียวกันทั้ง วิชวลเบสิกและ MCS-51 จึงจะสามารติดต่อกันได้ เลือกการเช็คบิตพาริตี้จากโปรแกรมเลือกไม่เช็ค เลือกจำนวนบิตของข้อมูล จากโปรแกรม เลือก 8 บิต และเลือกบิตปิดท้าย 1 บิต และเซตให้สามารถใช้พอร์ตอนุกรมได้ จากนั้นเมื่อมีการกดปุ่มต่างๆ ก็จะมีการส่งข้อมูลของปุ่มนั้นออกไปและวนกลับไปรอรับค่าใหม่ และรับข้อมูลมาเพื่อแสดงผล

โปรแกรมควบคุมรถสำรวจสามารถบันทึกภาพเป็นไฟล์ .avi และเลือกปลายทางที่จะเซฟได้จาก Textbox ในส่วนของโปรแกรมได้แสดงไว้ที่ภาคผนวก

3.1.3 วงจรภาคส่ง

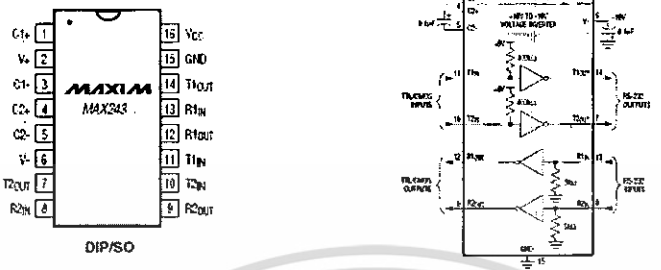


รูป 3.6 วงจรภาคส่ง

จากรูป หัวต่อของ DB9 มี 9 ขา แต่จะใช้เพียง 2 ขาคือ ขา 5 ต่อกราวด์ ขา 3 เป็น Output สัญญาณอนุกรมของ DB9 ซึ่งระดับแรงดันไม่เป็น TTL เมื่อมีการกดปุ่มคำสั่งจากโปรแกรมควบคุม สัญญาณลอจิก 1 จะมีค่า -9 โวลต์ และสัญญาณลอจิก 0 มีค่า +9 โวลต์ จากการวัด เพื่อให้ MCS-51 สามารถรับสัญญาณได้ต้องนำมาแปลงเป็นสัญญาณ TTL ก่อน โดยใช้ IC MAX 243

รูปที่ 3.7 แสดงถึงตัวถัง DIP 16 ขา และลักษณะภายในของ MAX243 โดยมีอินพุต TTL และ RS232 อย่างละ 2 ชุด ซึ่งสามารถแปลงแรงดัน TTL ไปเป็น แรงดัน RS232 และ แปลงแรงดันจาก

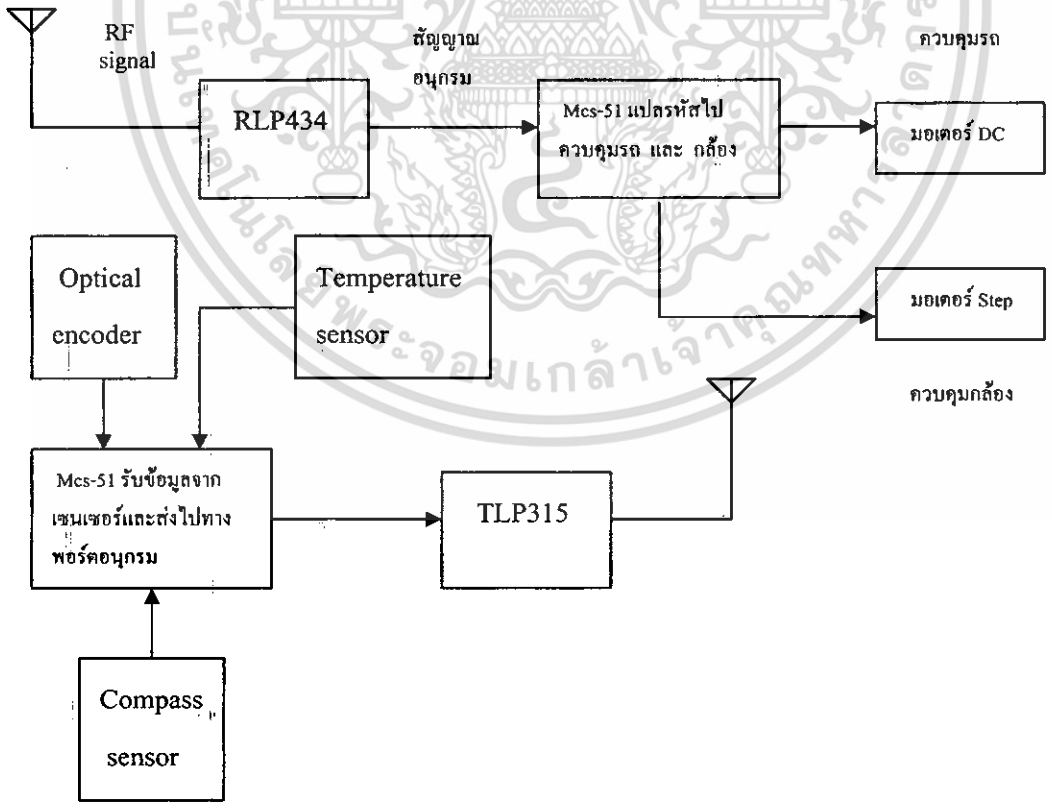
RS232 ไปเป็นระดับ TTL ได้ตามลำดับ IC ใช้แรงดัน 5 V และกระแส ประมาณ 15 mA



รูป 3.7 IC MAX243

จากนั้นส่งสัญญาณไปมอดูเลตกับสัญญาณ RF ที่ IC TLP434A ซึ่งใช้ไฟเลี้ยงได้ตั้งแต่ 2-12 V ที่ 12 V จะใช้กระแสสูงสุด 19.4 mA มี Data Rate ตั้งแต่ 512 – 200K bps และใช้การมอดูเลตแบบ ASK ออกอากาศไปทาง Antenna

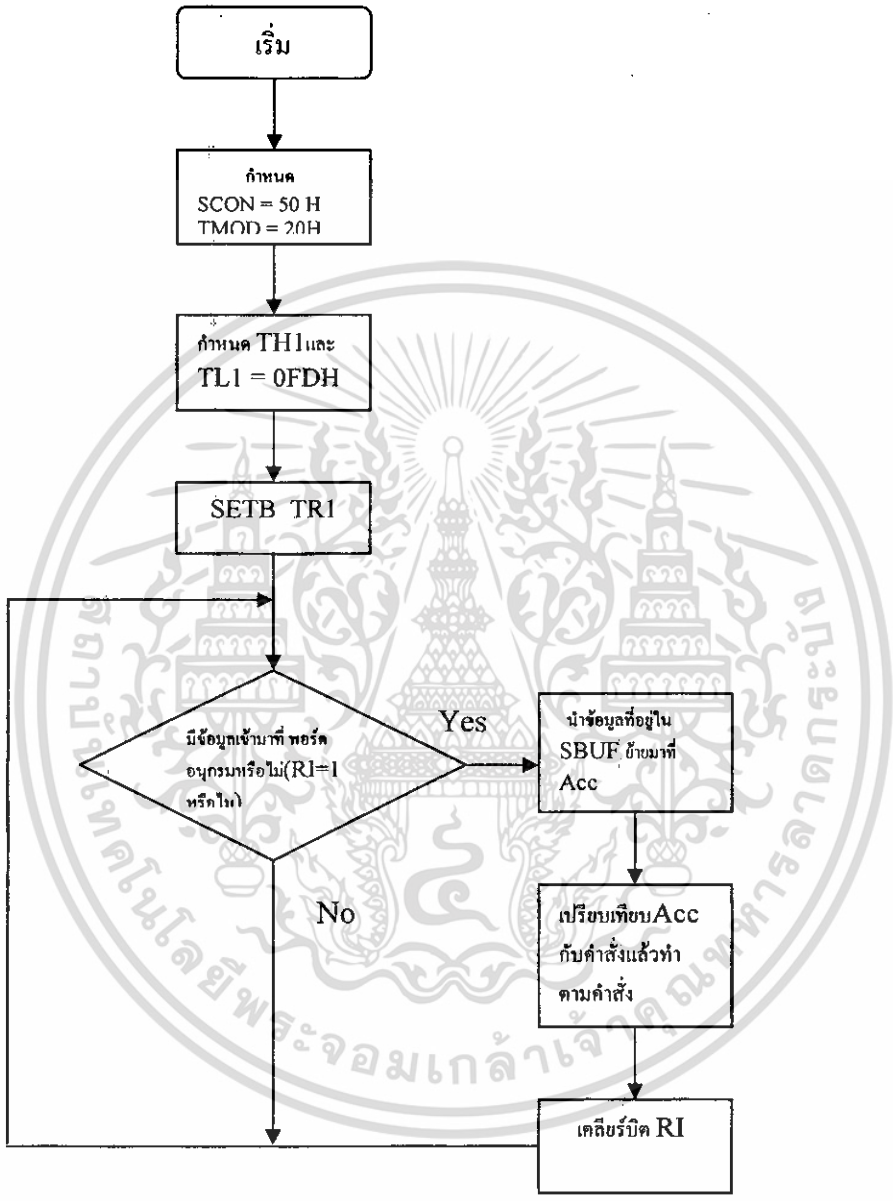
3.2 การออกแบบภาครับ



รูป 3.8 Block Diagram ของภาครับ

จากรูป 3.8ในส่วนควบคุมนั้น IC RLP434A จะดีมอดูเลตสัญญาณ ASK โดยตัดสัญญาณพาหะทิ้ง เหลือแต่สัญญาณอนุกรมและส่งไปที่ P3.0 ที่ AT89C52 ซึ่งจะทำการแปลรหัสแล้วควบคุมมอเตอร์ DC และ มอเตอร์ Step ตามคำสั่ง ที่กล้อง ซึ่งมีตัวส่งสัญญาณภาพอยู่ภายใน จะส่งสัญญาณภาพไปที่ Radio AV Receiver ที่ฝั่งส่ง

การที่จะให้ไมโครคอนโทรลเลอร์สามารถรับข้อมูลที่เข้ามาทางพอร์ตการสื่อสารอนุกรมได้นั้น ต้องกำหนดค่าของรีจิสเตอร์บางตัว การสื่อสารแบบนี้เป็นแบบซิงโครนัสจึงต้องการสัญญาณนาฬิกามานับเวลา AT89C52 มีรีจิสเตอร์ที่ทำหน้าที่เป็นสัญญาณนาฬิกา 3 ตัว คือ ไทเมอร์0(T0) ไทเมอร์1(T1) และไทเมอร์2(T2) เลือกใช้ T1 โดยโหลดค่า 20H เข้า TMOD ซึ่งเป็นรีจิสเตอร์เลือกการทำงานของไทเมอร์0 และ 1 หมายความว่าให้ T1 ทำงานในโหมด2 เป็นไทเมอร์ 8 บิต แบบตั้งค่าอัตโนมัติ จากนั้นเลือกค่าอัตราบอดเรตและต้องเป็นอัตราเดียวกับที่กำหนดไว้ที่โปรแกรมวิชวลเบสิก6 คือ 1200 โดยโหลดค่า 0E8H เข้ารีจิสเตอร์ TH1 และ TL1 จากนั้นกำหนดรีจิสเตอร์ควบคุมพอร์ตอนุกรม SCON โดยโหลดค่า 50H เข้าไปเพื่อให้ทำงานในโหมด 1 ส่งข้อมูลแบบ 8 บิต มีบิตเริ่มต้นและบิตหยุด จากนั้นเซตบิต TR1 เพื่อให้ไทเมอร์1 ทำงาน



รูป 3.9 Flow Chart การทำงานของไมโครคอนโทรลเลอร์ของภาครับในส่วนควบคุม

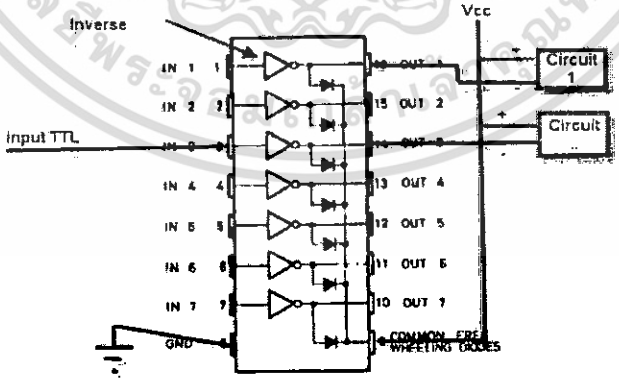
เมื่อมีข้อมูลเข้ามาที่พอร์ตอนุกรม ข้อมูลจะอยู่ที่รีจิสเตอร์ SBUF และจะทำให้ บิต RI เป็น1 จากนั้นย้ายข้อมูลจาก SBUF มาที่ Acc และเคลียร์บิต RI ให้เป็น 0 นำ Acc ไปเปรียบเทียบกับคำสั่งต่างๆเพื่อทำตามคำสั่งนั้น จากนั้นวนไปรอรับข้อมูลใหม่

ในส่วนของการรับค่าจากเซนเซอร์ ไมโครคอนโทรลเลอร์จะติดต่อกับ Compass sensor แบบ I2C ซึ่งจะรับข้อมูลมา 2 ไบต์ ซึ่งเมื่อนำไปแปลงค่าที่คอมพิวเตอร์แล้วจะได้ค่า 0-360 ซึ่งเป็นทิศทางที่รถทำกับทิศอ้างอิง ส่วนเซนเซอร์อุณหภูมิจะติดต่อแบบ ONE-WIRE และรับข้อมูลมา 1 ไบต์ ซึ่งจะมีความละเอียด 0.5 องศาเซลเซียส ส่วน Optical encoder นั้นจะติดอยู่กับล้อ โดยไมโครจะนับว่าทุกๆ 0.3 S ล้อได้หมุนไปเท่าไหร่ ซึ่งจะเป็นข้อมูล 1 ไบต์ จากนั้นก็จะส่งค่าข้อมูลที่ได้ทั้งหมด 4 ไบต์ โดยส่งไปทีละไบต์ พร้อมไบต์เปิดและปิด

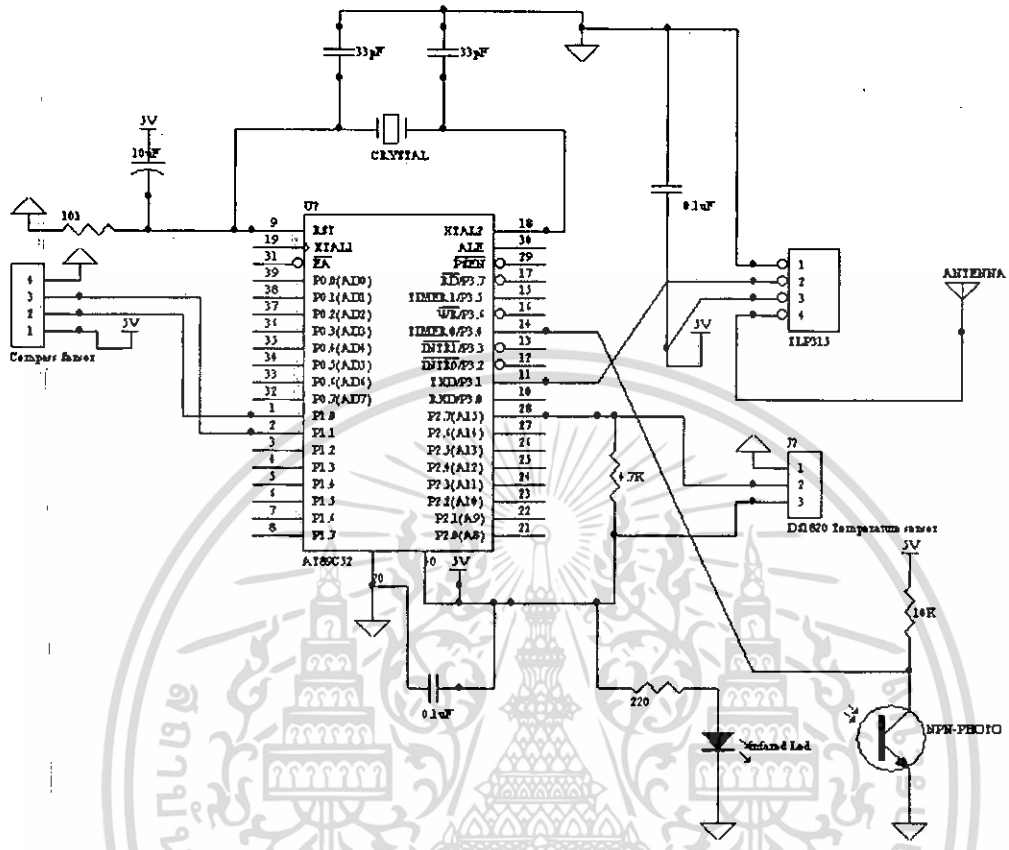
ไอซีที่นำมาใช้ขับสเตปมอเตอร์คือ ULN2003 ลักษณะการทำงานแสดงดังรูป 3.10 จะเห็นว่าทำหน้าที่คล้ายอินเวอเตอร์ ในการต่อสายนั้นสายสัญญาณ 4 สาย ของมอเตอร์จะต่อกับเอาต์พุตของ ULN2003 สายไฟเลี้ยงต่อกับไฟ 12 V จากนั้นเขียนโปรแกรมให้ไล่จ่ายไฟ 5 V ไปที่อินพุต 1-4 ของ ULN2003 โดยเว้นช่วงประมาณ 100 mS อินพุตที่มีไฟเข้า 5V จะทำให้เอาต์พุต เป็นกราวด์ ทำให้กระแสไหลได้ครบวงจรของมอเตอร์สเตป ซึ่งด้านเอาต์พุตสามารถใช้แรงดันได้ถึง 50 V และแต่ละขา สามารถ ทนกระแสได้ 500 mA ยังมีไดโอดที่แต่ละขาไว้สำหรับเมื่อใช้กับโหลดที่เป็น inductive load เพื่อป้องกันไฟย้อนกลับ อินเวอเตอร์ภายในก็คือ ทรานซิสเตอร์ต่อดาลิงตัน 2 ตัว แบบคอมมอนอีมิตเตอร์

การหาขาไฟเลี้ยงนั้นทำได้โดยนำโอห์มมิเตอร์มาวัดความต้านทานระหว่าง 2 ขา จะมี 1ขาที่วัดแล้วได้ความต้านทานต่างจากขาอื่นๆ ขานั้นจะเป็นขาไฟเลี้ยง

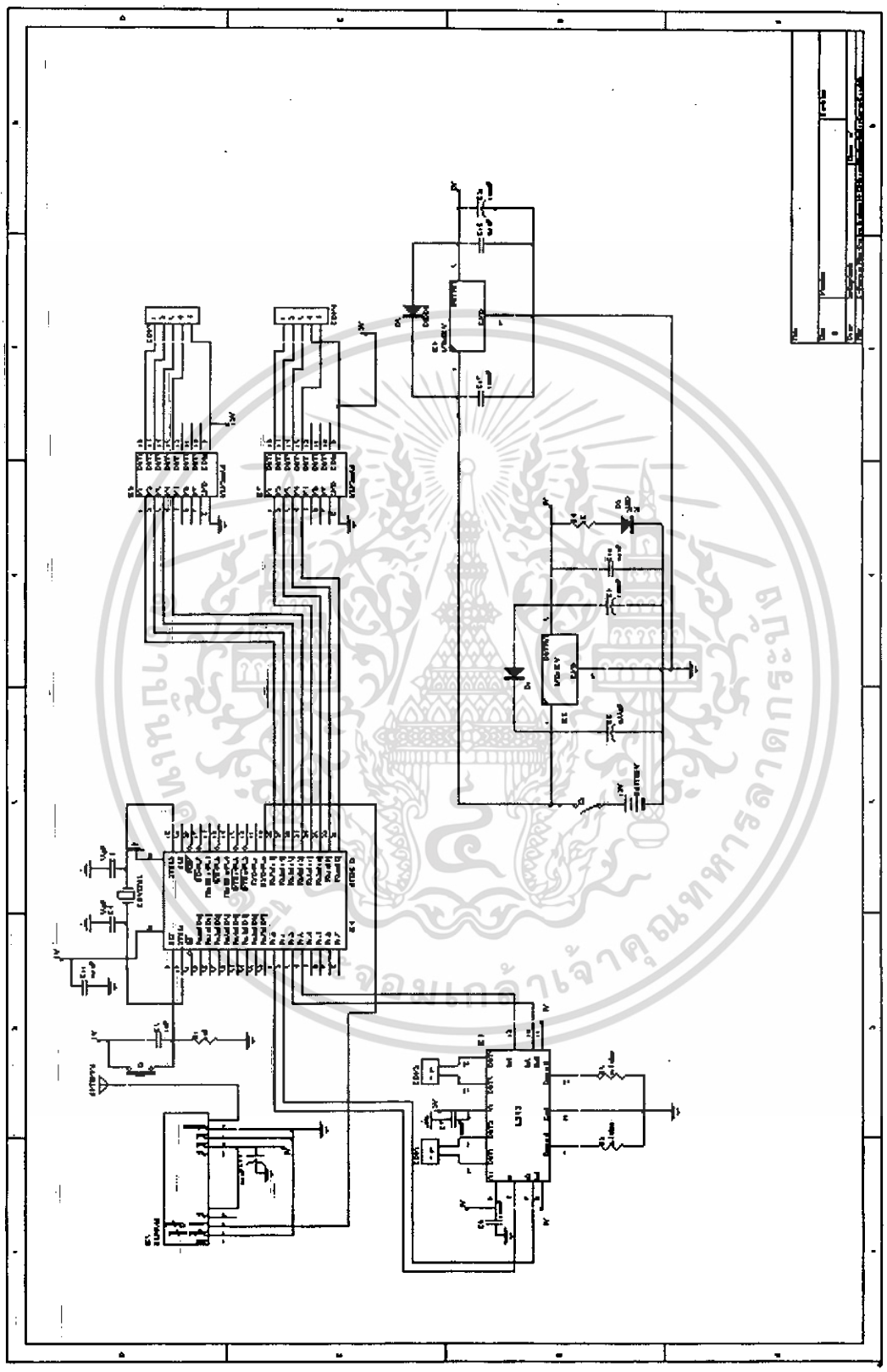
ส่วนการขับ DC มอเตอร์นั้นใช้ไอซี L298 ซึ่งทนแรงดันเอาต์พุตได้ 50 V และกระแส แต่ละเอาต์พุต 2 A มอเตอร์ที่ใช้ใช้ไฟ เลี้ยง 12 V และกระแส 0.5 A



รูป 3.10 ลักษณะภายในของ ULN2003



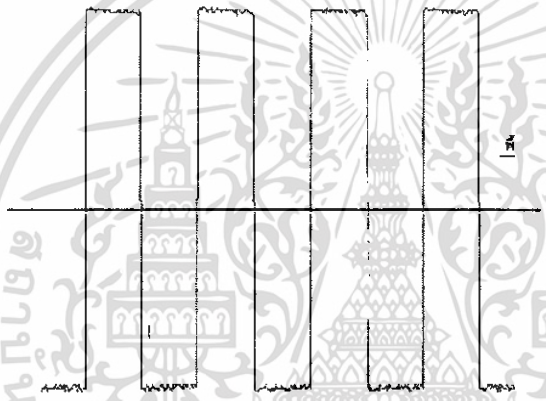
รูปที่ 3.11วงจรภาครับส่วนรับค่าเซนเซอร์



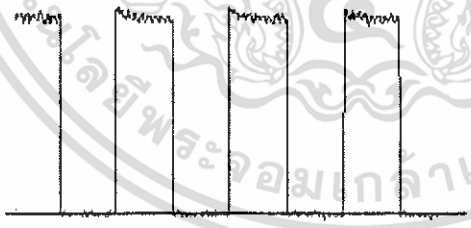
รูป 3.11 วงจรภาครับส่วนควบคุม

บทที่4
การวัดค่าสัญญาณ

4.1 ทดสอบการแปลงค่าสัญญาณของไอซี MAX243 โดยการป้อนอินพุตที่ขา 13 เป็นคลื่นสี่เหลี่ยม ความถี่ 2 K Hz ขนาด 10 Vp-p ดังรูปที่ 4.1 นำโอสซิโลสโคปวัดที่ขา 12 ได้ผลดังรูป 4.2 โดยกราฟทุกรูปใช้สเกล 2V/DIV 250uS/DIV



รูปที่ 4.1 อินพุตของMAX243

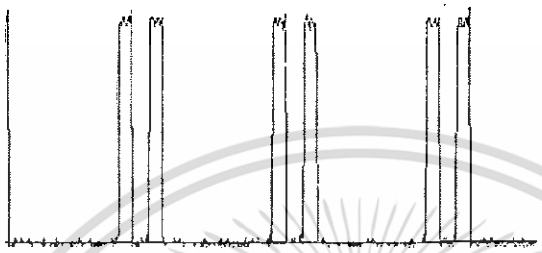


รูปที่ 4.2 เอาต์พุตของMAX243

จากการทดลองพบว่าได้ผลที่ถูกต้องคือ MAX243 จะแปลงแรงดันอินพุตที่มากกว่า +3 V เป็น 0 Vที่เอาต์พุต และแปลงแรงดันต่ำกว่า -3 V ไปเป็น +5 V ที่เอาต์พุตโดยความถี่ยังมีค่าคงที่ 2K Hz

4.2 การวัดไบต์คำสั่งเมื่อกดปุ่มควบคุมที่โปรแกรม

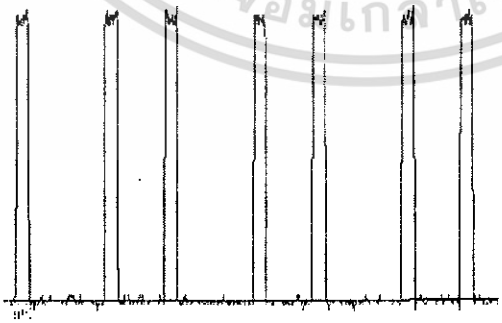
เมื่อกดปุ่มเดินหน้า จะส่งข้อมูล 03 H ออกทางพอร์ตอนุกรม จากนั้นแปลงเป็นระดับ TTL ด้วย MAX 232 วัดขาเอาต์พุต ได้ดังรูป 4.3 กราฟทุกรูปใช้ 2 V/DIV 2.5mS/DIV



รูปที่ 4.3 เอาต์พุตของ MAX232 เมื่อกดปุ่มเดินหน้า

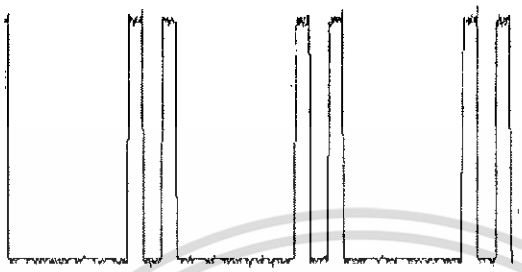
อธิบายได้ดังนี้ ข้อมูลที่ถูกส่งออกมามีขนาด 8 บิต มีบิตปิดท้าย 1 บิต จากที่กำหนดไว้ที่โปรแกรมวิชวลเบสิก มีการเว้นระยะระหว่างข้อมูลประมาณ 2 บิต โดยจะส่งบิตที่ 1 ออกมาก่อนจากกราฟคือบิตที่เป็น 1 ตามด้วยบิตที่เป็น 0 อีก 7 บิต จากนั้นมีบิตปิดท้ายที่เป็น 1 อีก 1 บิต และเว้นระยะห่างระหว่างข้อมูล 2 บิตซึ่งเป็น0 จากนั้นก็จะตามด้วยบิตที่ 1 ของข้อมูลตัวต่อไป

เมื่อกดปุ่มเลี้ยวซ้าย จะส่งข้อมูล 04 H ออกทางพอร์ตอนุกรม จากนั้นแปลงเป็นระดับ TTL ด้วย MAX 232 วัดขาเอาต์พุต ได้ดังรูป 4.4



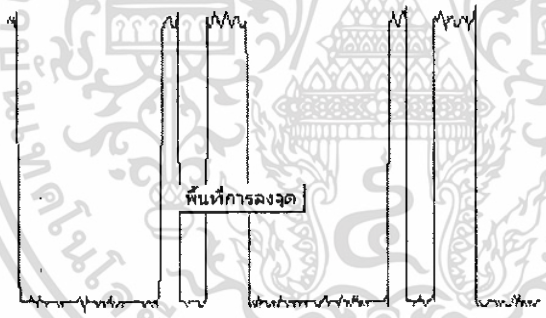
รูปที่ 4.4 เอาต์พุตของ MAX232 เมื่อกดปุ่มเลี้ยวซ้าย

เมื่อกดปุ่มเลี้ยวขวา จะส่งข้อมูล 02 H ออกทางพอร์ตอนุกรม จากนั้นแปลงเป็นระดับ TTL ด้วย MAX 232 วัดขาเอาต์พุต ได้ดังรูป 4.5



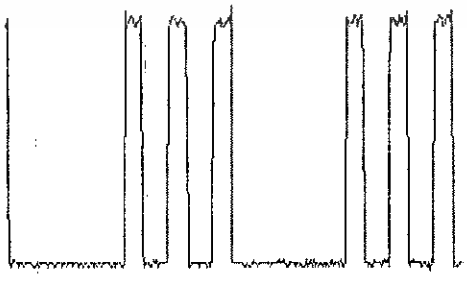
รูปที่ 4.5 เอาต์พุตของ MAX232 เมื่อกดปุ่มเลี้ยวขวา

เมื่อกดปุ่มถอยหลัง จะส่งข้อมูล 01 H ออกทางพอร์ตอนุกรม จากนั้นแปลงเป็นระดับ TTL ด้วย MAX 232 วัดขาเอาต์พุต ได้ดังรูป 4.6



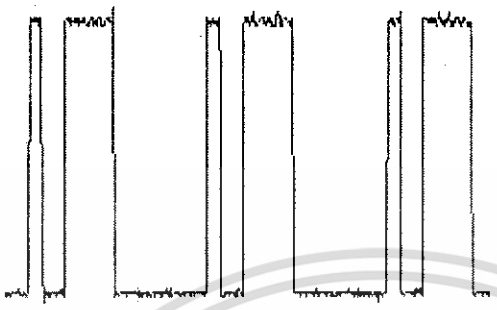
รูป 4.6 เอาต์พุตของ MAX232 เมื่อกดปุ่มถอยหลัง

เมื่อกดปุ่มหมุนกล้องขึ้น จะส่งข้อมูล 05 H ออกทางพอร์ตอนุกรม จากนั้นแปลงเป็นระดับ TTL ด้วย MAX 232 วัดขาเอาต์พุต ได้ดังรูป 4.7



รูป 4.7 เอาต์พุตของ MAX232 เมื่อกดปุ่มหมุนกล้องขึ้น

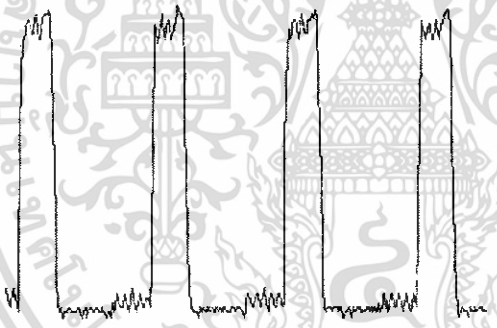
เมื่อกดปุ่มหมุนกล้องขึ้น จะส่งข้อมูล 06H ออกทางพอร์ตอนุกรม จากนั้นแปลงเป็นระดับ TTL ด้วย MAX 232 วัดขาเอาต์พุต ได้ดังรูป 4.8



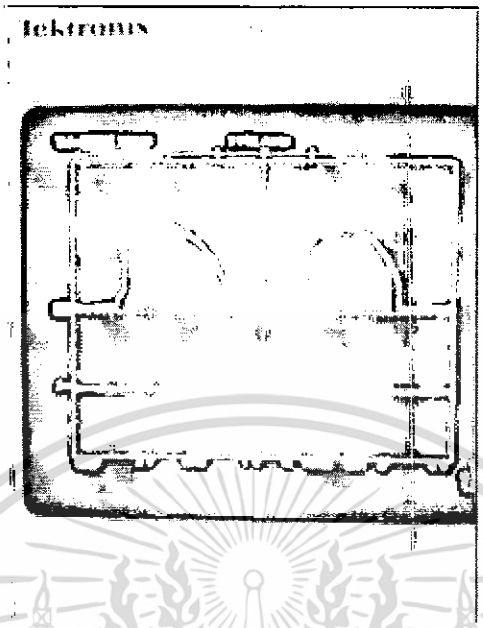
รูป 4.8 เอาต์พุตของ MAX232 เมื่อกดปุ่มหมุนกล้องขึ้น

4.3 วัดสัญญาณ TLP434A กับ RLP 434 A

เมื่อป้อนอินพุตแก่ ไอซี TLP434A ดังรูปที่ 4.2 แล้ววัดสัญญาณจากขา 2 ของ RLP ซึ่งเป็นเอาต์พุตที่จะไปเข้าพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ได้ผลดังรูป 4.9



รูป 4.9 เอ้าต์พุตของ RLP434A เอาต์พุตของ RLP มีลักษณะเหมือนกับอินพุตของ TLP แต่จะมีสัญญาณรบกวนอยู่บ้าง



รูปที่ 4.10 ลักษณะสัญญาณจากการวัดที่ขาC ของโฟโต้ทรานซิสเตอร์

จากรูปใช้ 2 V/div และ 10ms/div เมื่อโฟโต้TR เจอกับ encoder สีดำ จะให้สัญญาณ High มีค่าประมาณ 4.5 V และเมื่อโฟโต้TR เจอกับ encoder สีขาว จะให้สัญญาณ Low มีค่าประมาณ 0.2 V

ตารางที่ 4.1 เปรียบเทียบระยะที่รถวิ่งได้จริงกับระยะที่คำนวณจากencoder

ครั้งที่	ระยะที่รถวิ่งได้จริง (cm)	ระยะจากจอแสดงผล	%ความผิดพลาด
1	94	90	4.4
2	100	101	1
3	90	88	2.2
4	84	89	5.95
5	106	115	8.49
6	88	84	4.54
7	95	102	7.4
8	98	96	2
9	99	104	5
10	105	112	6.7

รูปที่ 4.11 encoder ที่ติดกับล้อ และโฟโต้ทรานซิสเตอร์

ตารางที่ 4.2 ผลการเปรียบเทียบค่ามุมที่วัดได้จากโมดูลเข็มทิศกับค่ามุมจริง

มุม	0	45	90	135	180	225	270	315	360
มุมที่อ่านได้จากโมดูล	0	48.9	98.3	144.8	193.3	238	275	315	359
%ความผิดพลาด	0	8.7	9.2	7.3	7.4	5.8	1.8	0	0.3

บทที่ 5
สรุปผลและวิจารณ์

การทำโครงงานนี้ ต้องใช้ความรู้ในหลายๆด้านเข้ามารวมกัน เช่น โปรแกรมวิชวลเบสิก การควบคุมโดยใช้ไมโครคอนโทรลเลอร์ การรับ-ส่งข้อมูลแบบไร้สาย การควบคุมมอเตอร์ การออกแบบโครงสร้างของรถ การรับสัญญาณภาพวีดีโอ ซึ่งในช่วงของการหาข้อมูลใช้เวลามากเกินไป จนเหลือเวลาในการทำโครงงานจริงๆน้อย ทำให้ผลงานออกมาอาจไม่ตรงกับที่คิดไว้ เนื่องจากไม่เหลือเวลาในการปรับปรุง ควรปรับปรุงโดยการวางแผนเวลาล่วงหน้า

การส่งและรับข้อมูลในระยะใกล้ประมาณ 20 เมตรเป็นไปอย่างถูกต้องแต่เมื่อไกลจากนี้จะเริ่มมีสัญญาณรบกวนมากขึ้นทำให้มีความผิดพลาดมากยิ่งขึ้นและไม่สามารถสั่งงานรถได้ที่ระยะ40m

โครงสร้างของรถยังไม่อาจทำให้เคลื่อนที่ในพื้นที่ขรุขระมากๆได้ อาจแก้ไขให้ยกตัวรถสูงขึ้นเพิ่มเข้าไปอีก หรือทำล้อตีนตะขาบ เพื่อเพิ่มความยืดหยุ่นในการเคลื่อนที่ การวัดระยะทางด้วย encoder มีความผิดพลาดเป็นบางครั้งเนื่องจากการที่ระยะห่างของ encoder กับโฟโตทรานซิสเตอร์เปลี่ยนไปจากการกระแทก และอาจมีการรบกวนของแสงด้วย การวัดมุมด้วยโมดูลเข็มทิศดิจิตอลยังมีความผิดพลาดอยู่ในบางมุมมีค่าสูงเกือบ 10 % การควบคุมการทำงานและการรับส่งข้อมูลของไมโครคอนโทรลเลอร์ไม่มีข้อบกพร่อง การวัดค่าของอุณหภูมิทำได้ถูกต้องมีค่าแตกต่างกับตัววัดตัวอื่น 1- 2 องศา

หนังสืออ้างอิง

1. ธีรวัฒน์ ประกอบผล ; "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์" ; สมาคมส่งเสริมเทคโลยี (ไทย-ญี่ปุ่น),2542

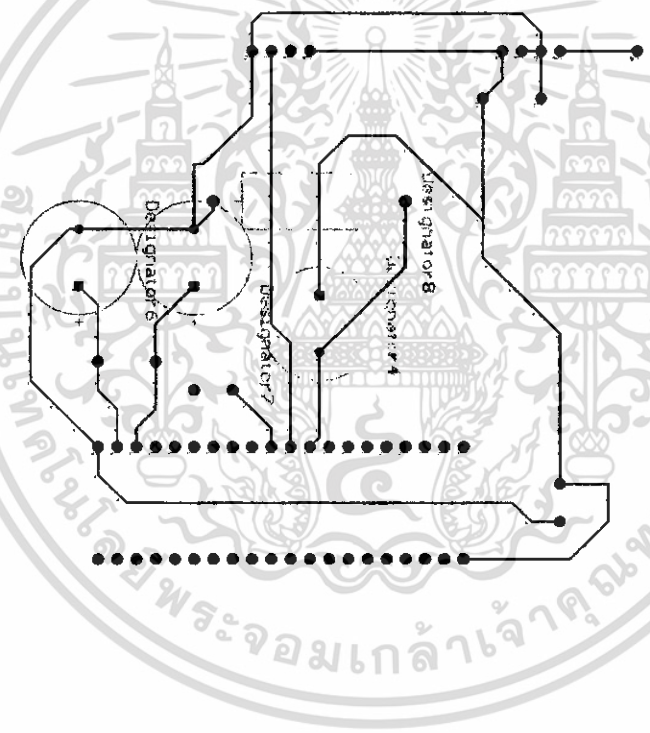
2.วรพจน์ กรแก้ววัฒนกุลม,ชัยวัฒน์ลิ้มพรจิตรวิไล ; "เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช "; Inex

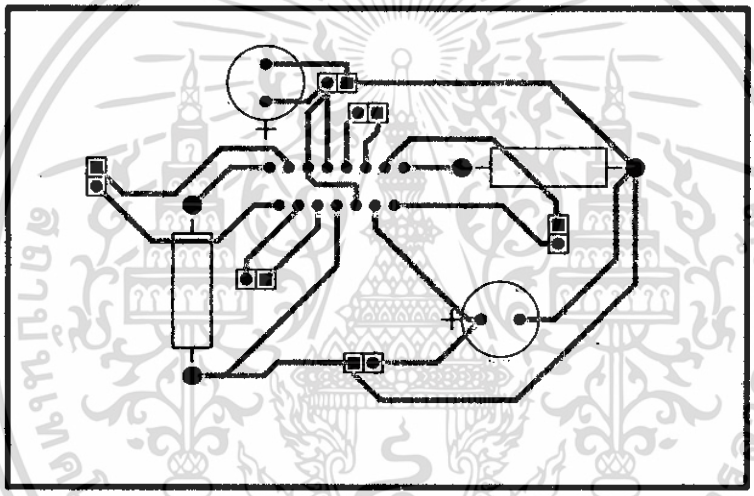
3. อภิชาต ภู่พลับ ; "การติดต่อกับฮาร์ดแวร์ด้วยวิชวลเบสิก"; Infopress Develop Book,2546

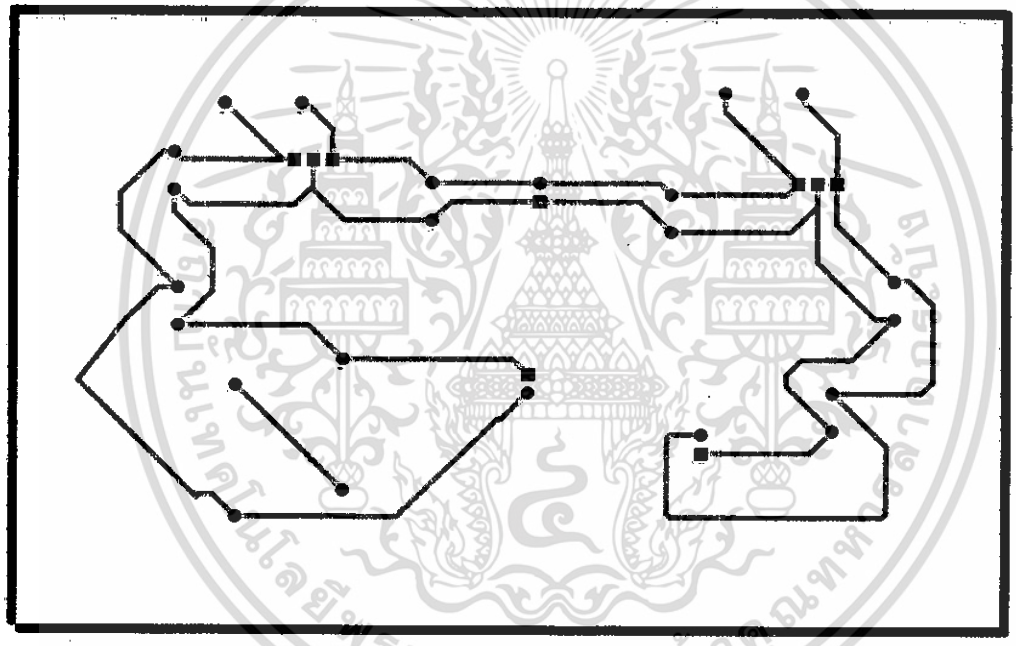
4.www.thaiio.com

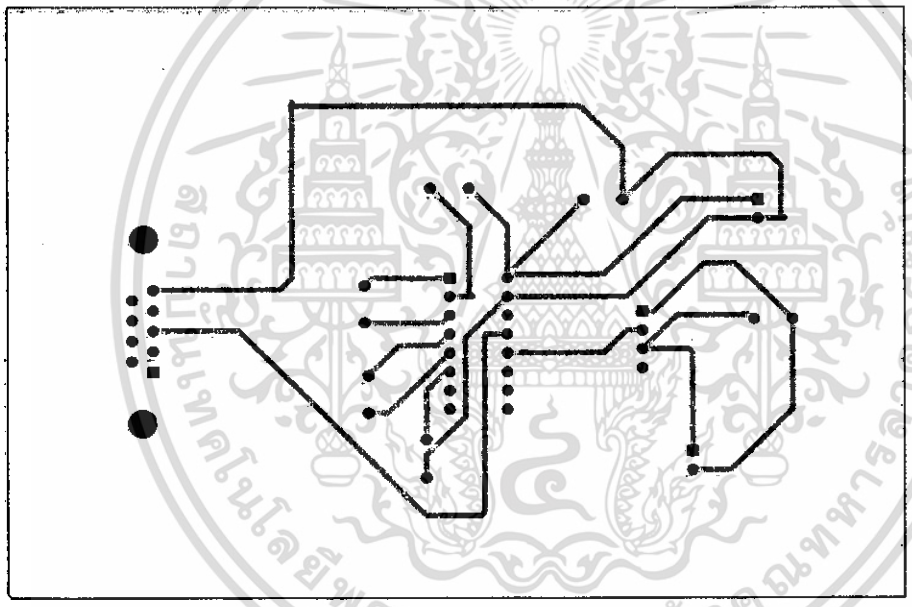
5.www.wara.com

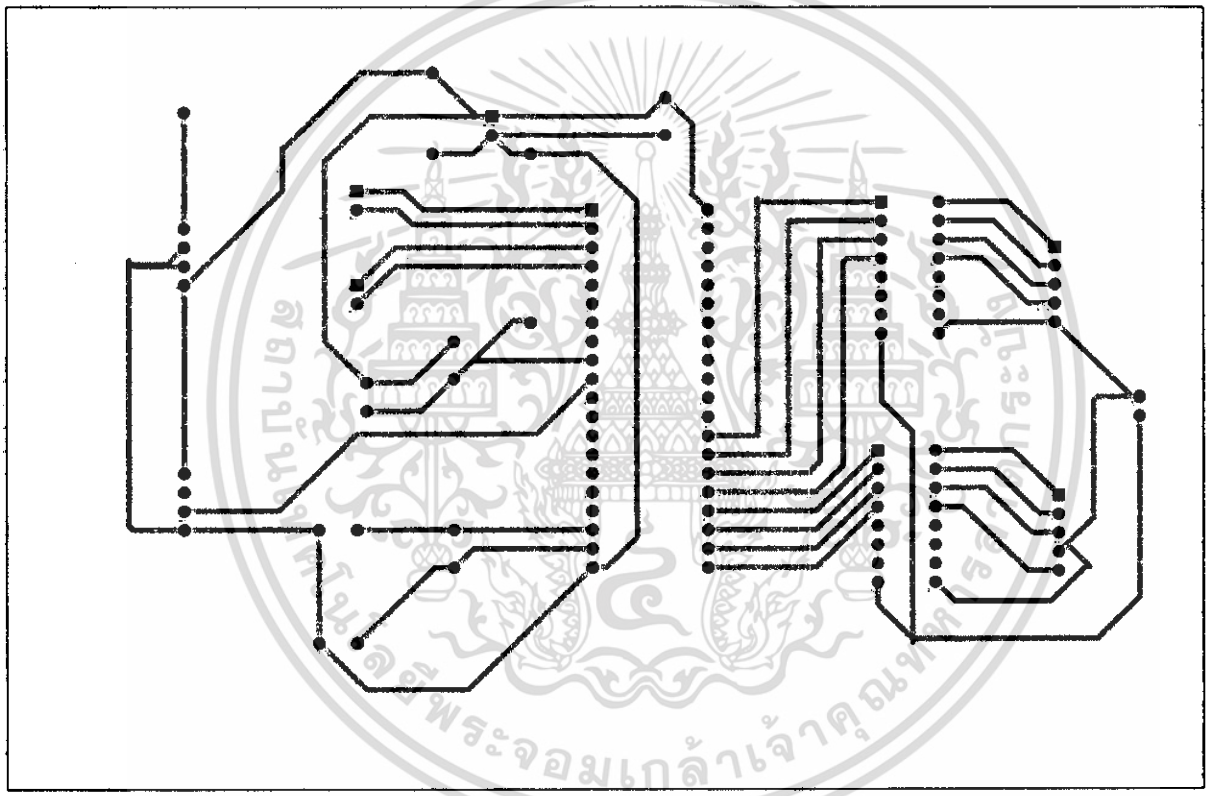
ภาคผนวก

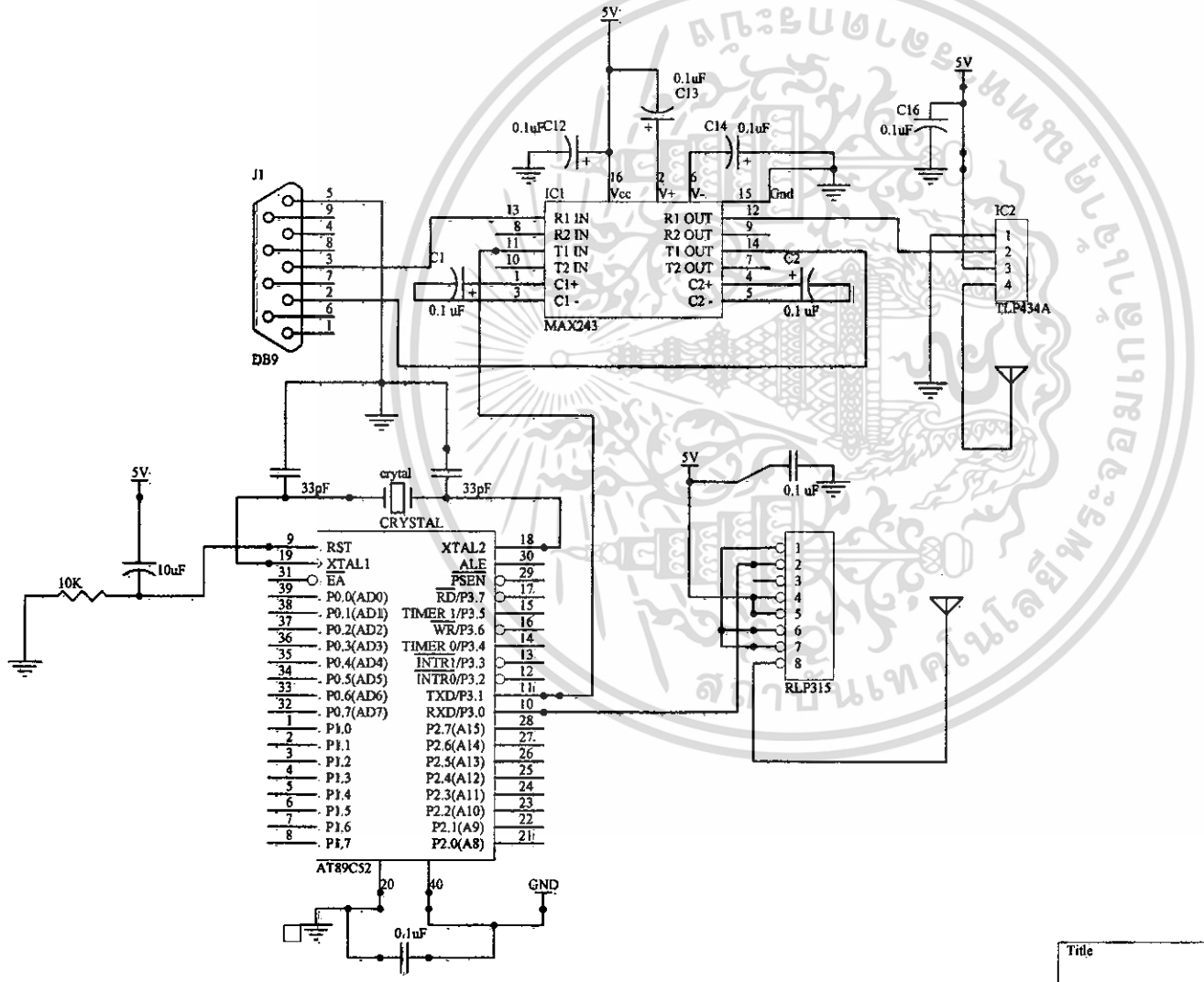




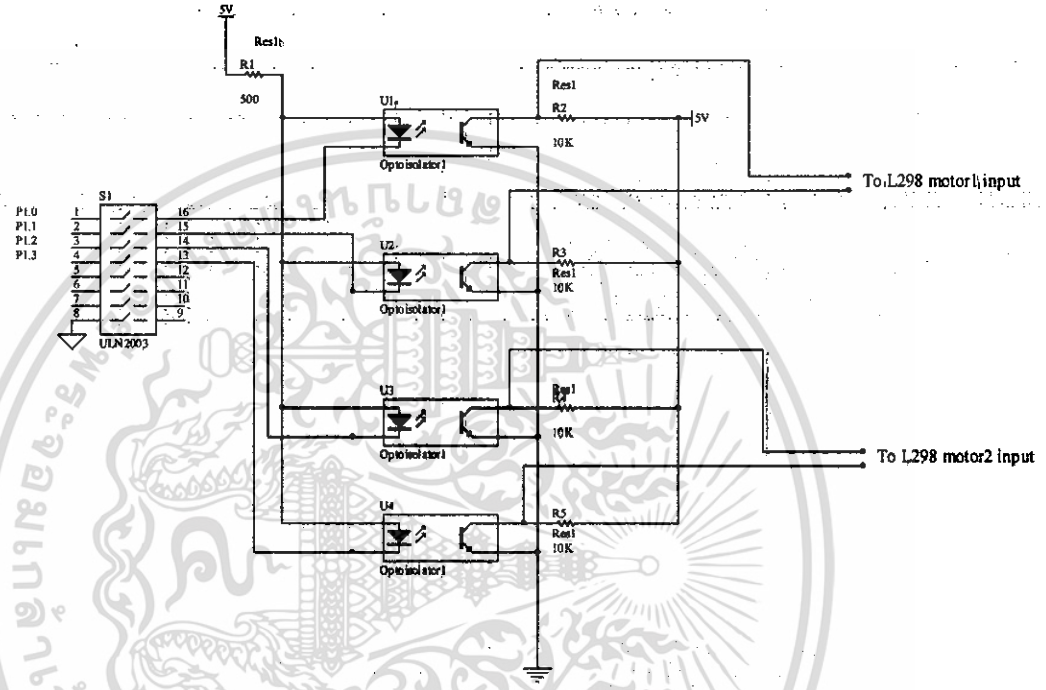




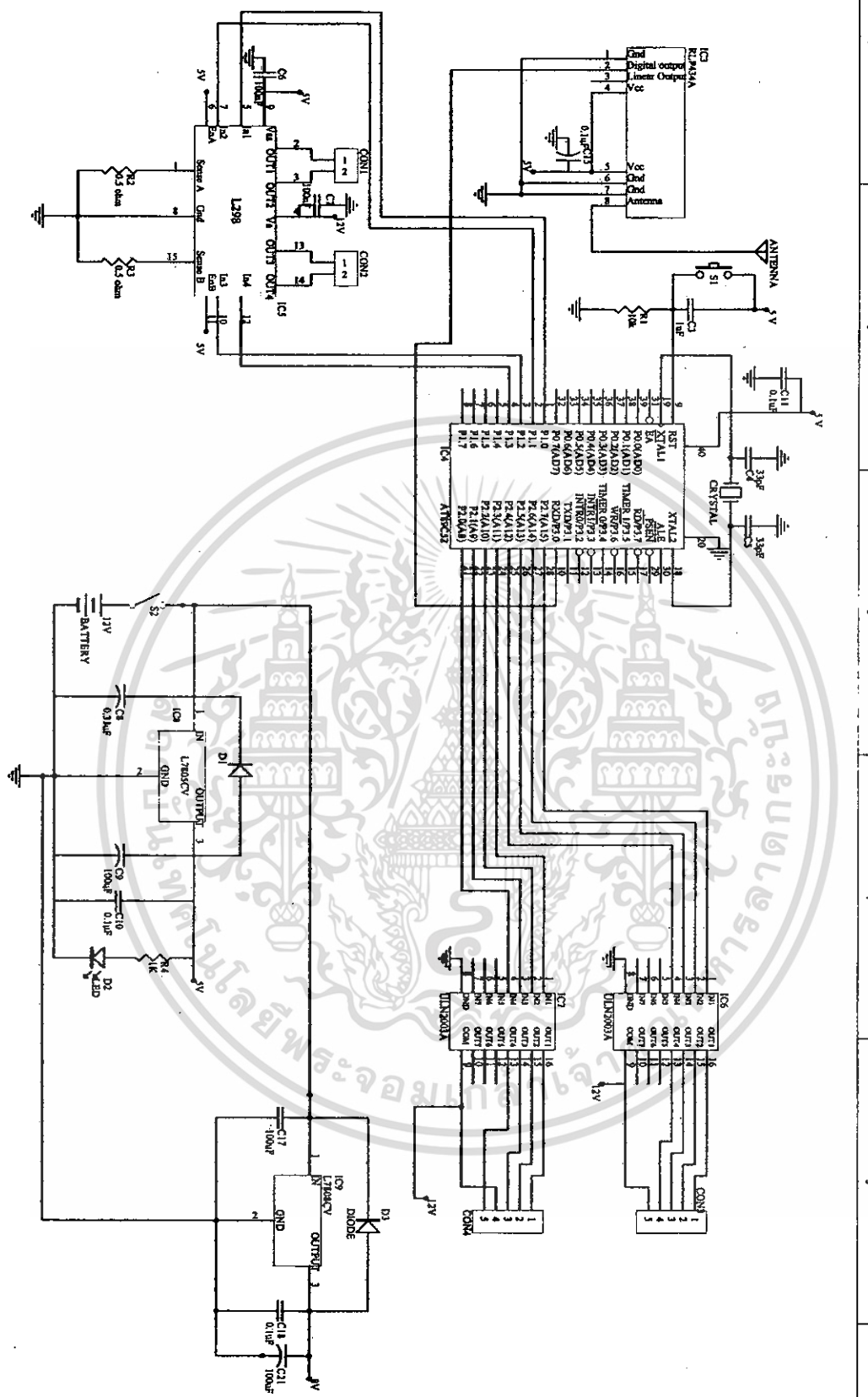




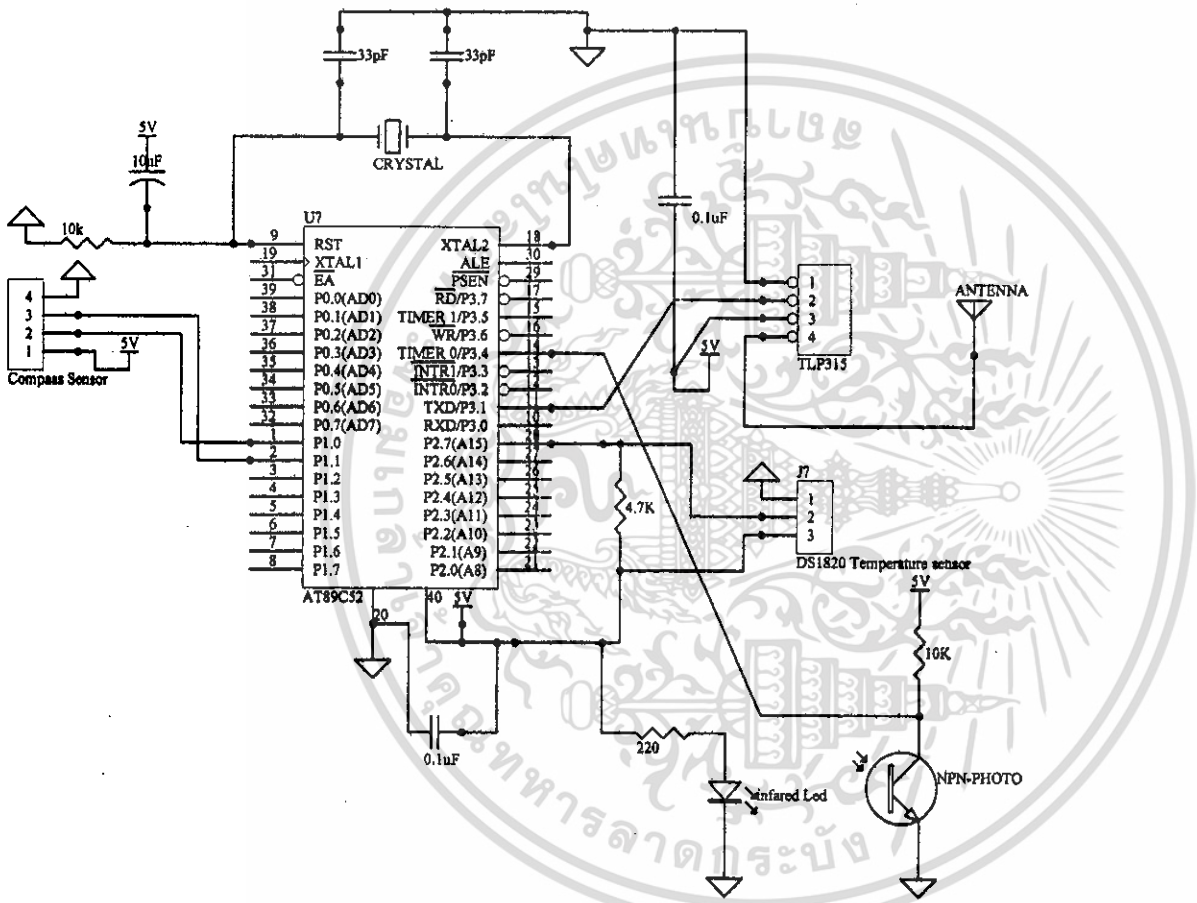
Title		
Size	Number	Revision
A4		



Title		
Size: B	Number	Revision
Date: 3/8/2007		Sheet of
File: C:\Documents and Settings\...\op1.SCHDOC		Drawn By:



ภาพที่ 3.12 วงจรภาคส่งส่วนควบคุม



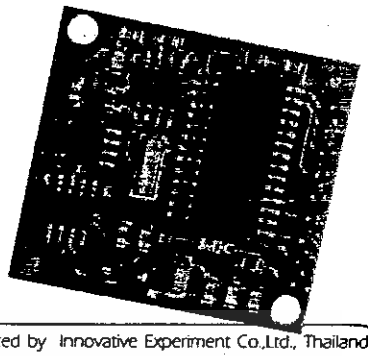
รูปที่ 3.11 วงจรรับค่าจากเซนเซอร์

Title		
Size A4	Number	Revision
Date: 9-Feb-2007		Sheet of
File: C:\Program Files\Design Explorer 99 SE\Examples\Pgs.DDB		

CMPS03

Digital Compass Module

โมดูลเข็มทิศดิจิตอล

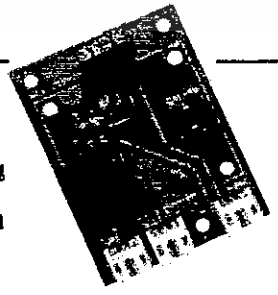


Distributed by Innovative Experiment Co.,Ltd., Thailand

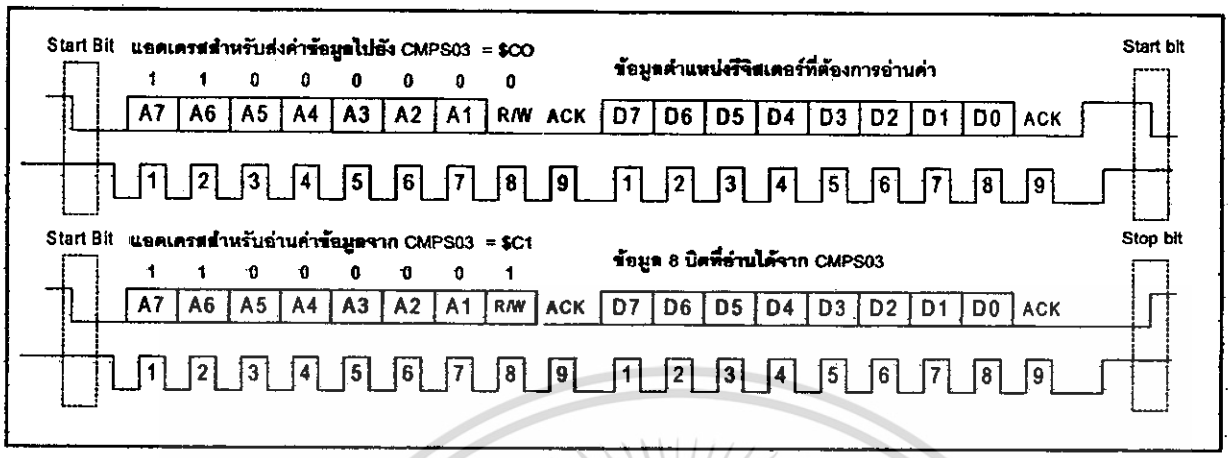
คุณสมบัติ

- ใช้ไฟเลี้ยง +5V ต้องการกระแสไฟฟ้า 20mA
- ใช้ตัวตรวจจับสนามแม่เหล็กเบอร์ KMZ51 ของ Philips จำนวน 2 ตัว เพื่อให้สามารถตรวจจับสนามแม่เหล็กโลกได้อย่างสมบูรณ์และมีความละเอียดมากเพียงพอ
- ความละเอียดของมุม 0.1 องศา
- ค่าความผิดพลาด 3-4 องศา โดยประมาณ หลังจากการปรับแต่ง
- เอาต์พุตแบบสัญญาณพัลส์ ความกว้าง 1 ถึง 37 มิลลิวินาที โดยมีอัตราเพิ่มครั้งละ 0.1 มิลลิวินาที
- เอาต์พุตข้อมูลดิจิตอลผ่านการติดต่อระบบบัส I^2C รองรับสัญญาณนาฬิกาความถี่สูงถึง 1MHz โดยให้ข้อมูล 2 รูปแบบคือ 0-255 และ 0-3599
- ขนาดเล็กเพียง 32 x 35 มิลลิเมตร
- สื่อสารกับไมโครคอนโทรลเลอร์ยอดนิยมได้ทุกตระกูล อาทิ เบสิกแสตมป์ 2SX/2P, PIC, MCS-51, PSoC, 68HC11 ทั้งผ่านระบบบัส I^2C และด้วยการวัดสัญญาณพัลส์

อุปกรณ์เสริม



- บอร์ด ADX-CMPS03 ซึ่งเป็นบอร์ดอะแดปเตอร์สำหรับอำนวยสะดวกในการเชื่อมต่อกับบอร์ดไมโครคอนโทรลเลอร์และแผงต่อวงจรหรือเบรดบอร์ด
- สาย PCB3A สำหรับเชื่อมต่อกับบอร์ดควบคุมหุ่นยนต์ของ i-nex



รูปที่ 3 แสดงไทมิ่งไดอะแกรมของการติดต่อสื่อสารกับโมดูล CMPS03 ผ่านระบบบัส I^2C

1. ส่งบิตเริ่มต้นหรือ Start bit เพื่อแจ้งให้ระบบบัส I^2C เตรียมพร้อมรับข้อมูล

2. ส่งค่าแอดเดรส $C0 เพื่อระบุว่าต้องการติดต่อเพื่อเขียนข้อมูลไปยังกับโมดูล CMPS03

3. ส่งค่าตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03 ที่ต้องการอ่านค่า ซึ่งมีรายละเอียดแสดงในตารางที่ 1

4. ส่งค่าแอดเดรส $C1 เพื่อระบุว่าต้องการอ่านค่าข้อมูลจากโมดูล CMPS03

5. อ่านค่าข้อมูลจากโมดูล CMPS03 มาเก็บไว้ในหน่วยความจำ

6. ส่งบิตหยุดหรือ Stop เพื่อหยุดการสื่อสารข้อมูล และกำหนดให้บัสอยู่ในสภาวะบัสว่าง

จากลำดับขั้นการติดต่อสื่อสารข้างต้น สามารถนำมาเขียนเป็นโปรแกรมตัวอย่างเพื่ออ่านข้อมูลจากโมดูล CMPS03 โดยใช้เบสิกแสตมป์ 2SX หรือ i-Stamp ได้ดังแสดงในโปรแกรมที่ 2

ตำแหน่งรีจิสเตอร์	รายละเอียด
0	ตัวเลขแสดงรุ่นของบอร์ด CMP03
1	ส่งค่าตำแหน่งแบบหยาบ (0-255)
2,3	ส่งค่าตำแหน่งแบบละเอียดด้วยตัวเลข 16 บิต (0-3599) สามารถแปลงค่าเพื่อแสดงองศา 0-359.9 องศาได้โดยตรง
4,5	สำหรับตรวจสอบค่าภายใน โดยจะแสดงค่าความต่างของ Sensor1 เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
6,7	สำหรับตรวจสอบค่าภายใน โดยจะแสดงค่าความต่างของ Sensor2 เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
8,9	แสดงค่าตัวเลขการปรับแต่งภายใน (calibration value1) เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
10,11	แสดงค่าตัวเลขการปรับแต่งภายใน (calibration value2) เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
12,13	ไม่ใช้งานอ่านค่าได้เป็น 0
14	ไม่ใช้งาน ไม่ได้กำหนดค่าไว้
15	คำสั่งสำหรับการปรับแต่งค่า โดยเมื่อต้องการปรับแต่งค่า ต้องเขียนข้อมูล 255 เข้าที่รีจิสเตอร์ตำแหน่งนี้

ตารางที่ 1 แสดงตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03

3.2 การอ่านค่าทิศทางเป็นข้อมูลดิจิตอลผ่านระบบบัส I^2C

การอ่านค่าจากโมดูล CMPS03 ให้ได้ค่าที่มีความแม่นยำสูงควรเลือกเอาต์พุตข้อมูลดิจิตอลผ่านระบบบัส I^2C โดยโมดูล CMPS03 สามารถส่งข้อมูลของตำแหน่งออกมาที่ความละเอียดสูงสุด 0.1 องศาโดยไม่จำเป็นต้องมีการคำนวณหรือแปลงค่าใด ๆ อีก

3.2.1 รูปแบบการสื่อสารข้อมูลบัส I^2C

บัส I^2C เชื่อมต่อกับไมโครคอนโทรลเลอร์โดยใช้สายสัญญาณ 2 เส้นได้แก่ขา SDA (รับและส่งข้อมูล) และ SCL (ขาสัญญาณนาฬิกา) โดยขาสัญญาณทั้งสองจะต้องต่อตัวต้านทานพูลอัปต่อไว้เพื่อกำหนดสถานะลอจิก "1" ให้กับระบบบัส

3.2.2 ลำดับขั้นการติดต่อ

ค่าแอดเดรสของโมดูล CMPS03 คือ $C0 สำหรับการส่งข้อมูล และ $C1 สำหรับการอ่านค่าข้อมูล โดยขั้นตอนการติดต่อกับโมดูล CMPS03 เพื่ออ่านข้อมูลมีดังนี้

3. การอ่านค่าสัญญาณเอาต์พุตของโมดูล CMPS03

3.1 การอ่านค่าทิศทางจากเอาต์พุตสัญญาณพัลส์

การอ่านค่าสัญญาณในโหมดนี้ เป็นการนำค่าความกว้างพัลส์ที่ได้จากเอาต์พุตสัญญาณพัลส์ของโมดูล CMPS03 มาระบุตำแหน่งองศา จาก 0 ถึง 359.9 องศา โดยมีย่านของค่าความกว้างสัญญาณพัลส์จาก 1 มิลลิวินาทีไปจนถึง 36.99 มิลลิวินาที มีความละเอียด 0.1 มิลลิวินาทีต่อองศา ในสัญญาณพัลส์แต่ละไซเกิล มีช่วงลอจิก "0" กว้าง 65 มิลลิวินาที

ดังนั้นในการนำสัญญาณพัลส์มาประมวลผลเป็นค่ามุม จึงต้องใช้การนับความกว้างของสัญญาณพัลส์เป็นหลักในการคำนวณหาค่ามุมที่โมดูล CMPS03 วัดได้

3.1.1 ตัวอย่างโปรแกรมที่ใช้ควบคุมสำหรับเบสิกแสตมป์ 2SX และ i-Stamp

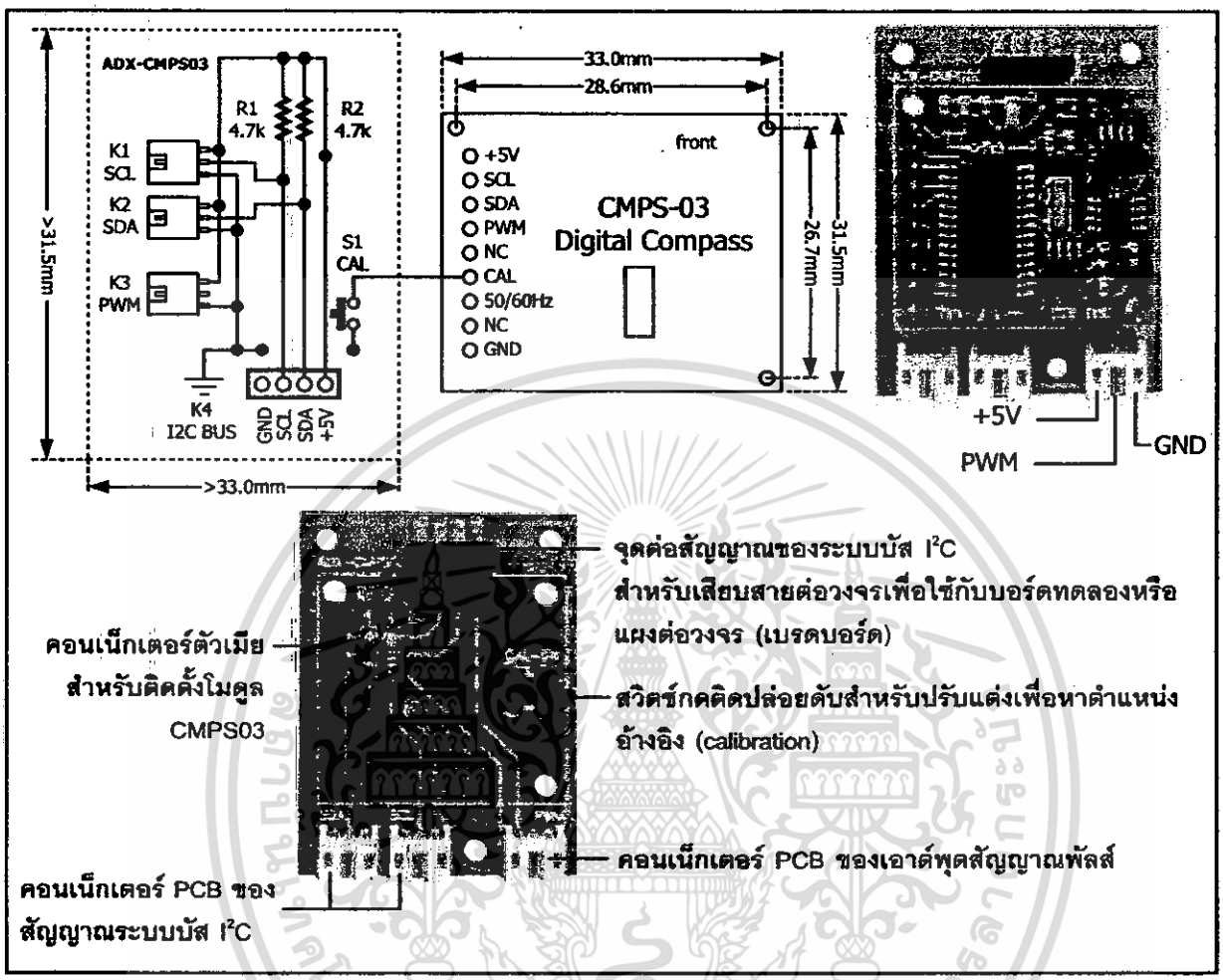
การใช้งานร่วมกับเบสิกแสตมป์ 2SX และ i-Stamp นั้น จะใช้คำสั่ง PULSIN ในการนับสัญญาณพัลส์ โดยจะเพิ่มค่าการนับขึ้นทุกๆ 0.8 ไมโครวินาที ดังนั้นที่ความกว้างของพัลส์ที่ 1 มิลลิวินาทีสำหรับตำแหน่ง 0 องศา เบสิกแสตมป์ 2SX และ i-Stamp จะนับค่าได้เท่ากับ 1,250 จึงสามารถใช้ค่านี้เป็นจุดอ้างอิงที่ 0 องศา *เมื่อต้องการทราบค่ามุมที่แม้จริง ให้นำค่ามุมที่นับได้ลบด้วย 1,250 แล้วหารด้วย 125 ก็จะได้ค่ามุมในหน่วยองศาที่ต้องการ* รายละเอียดของโปรแกรมแสดงในโปรแกรมที่ 1

ที่ความกว้างพัลส์สูงสุดคือ 36.99 มิลลิวินาที ค่าที่นับได้จากคำสั่ง PULSIN เท่ากับ 46,237 เมื่อลบด้วย 1,250 แล้วหารด้วย 125 เพื่อแปลงเป็นองศา ค่าสูงสุดที่แสดงเป็นผลลัพธ์ได้คือ 359 เป็นค่าหน่วยองศาสูงสุดนั่นเอง

หมายเหตุ *ในบางกรณี ค่าที่อ่านได้สูงสุดจากคำสั่ง PULSIN อาจไม่ถึง 46,237 ผู้ใช้งานสามารถปรับเปลี่ยนการคำนวณใหม่ให้ได้มุมเป็น 359.9 องศาได้*

```
'{$STAMP BS2sx}
'{$PBASIC 2.5}
bearing  VAR  WORD
main:
     PULSIN 4, 1, bearing                ' Get reading
     bearing = (bearing-1250)/125        ' BS2sx  - Calculate Bearing
                                         ' in degrees
     DEBUG "Compass Bearing  ", DEC3 bearing ,CR
                                         ' Display Compass Bearing
     GOTO main
```

โปรแกรมที่ 1 แสดงการอ่านค่าสัญญาณพัลส์จากโมดูล CMPS03



รูปที่ 2 แสดงวงจรของบอร์ด ADX-CMPS03 และการเชื่อมต่อกับโมดูล CMPS03

2. การปรับแต่งค่าทิศทางอ้างอิงแก่โมดูล CMPS03

เพื่อให้การวัดทิศทางของโมดูล CMPS03 มีความแม่นยำมากที่สุด จึงมีอินพุตสำหรับปรับแต่งค่าทิทางอ้างอิง ทั้งนี้เพื่อประโยชน์ในการกำหนดทิศทางอ้างอิงเฉพาะสำหรับผู้ใช้งาน โดยต้องป้อนสัญญาณลอจิก "0" เข้าที่ขาอินพุตสำหรับปรับแต่งโมดูล CMPS03 ซึ่งก็คือขา 6 หากใช้บอร์ด ADX-CMPS03 กับโมดูล CMPS03 จะมีสวิตช์กดติดปล่อยดับติดตั้งไว้ให้แล้ว การปรับแต่งมีขั้นตอนดังนี้

(1) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศเหนือ กดสวิตช์ 1 ครั้ง

(2) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันออก กดสวิตช์

(3) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศใต้ กดสวิตช์ 1 ครั้ง

(4) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันตก กดสวิตช์

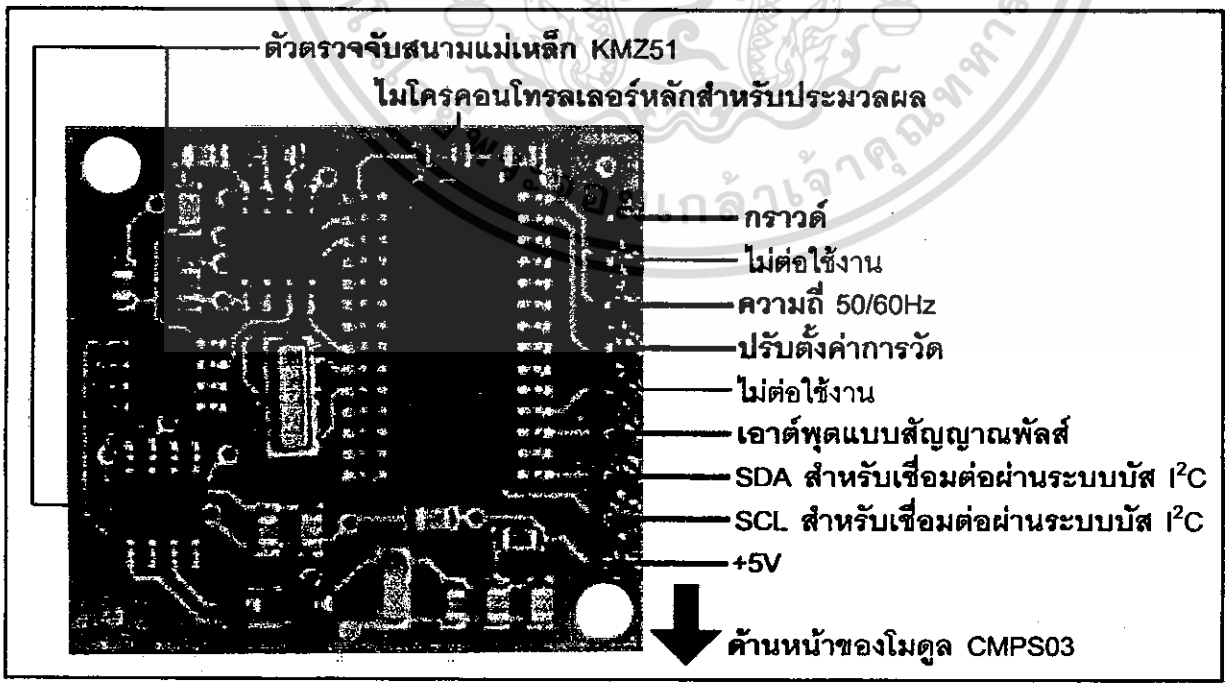
เป็นอันสิ้นสุดการปรับตั้งค่าทิศทางอ้างอิงของโมดูล CMPS03 โดยโมดูลจะเก็บค่าอ้างอิงนี้ไว้ในหน่วยความจำอีอีพรอมและไม่ต้องปรับตั้งค่าใหม่อีกเมื่อจ่ายไฟเลี้ยงครั้งใหม่

โมดูลเข็มทิศดิจิตอล CMPS03 เป็นผลงานของ Devantech (www.radio-electronics.co.uk) ออกแบบมาเพื่อช่วยในการกำหนดทิศทางการเคลื่อนที่ของหุ่นยนต์อัตโนมัติ และนำมาใช้ในการสร้างเครื่องมือวัดและตรวจสอบทิศระบบอิเล็กทรอนิกส์ โดยหัวใจสำคัญของโมดูล CMPS03 คือ ตัวตรวจจับสนามแม่เหล็กเบอร์ KMZ51 ของ Philips จำนวน 2 ตัว เพื่อให้มีความไวเพียงพอในการตรวจจับสนามแม่เหล็กโลก (Earth magnetic field) และไมโครคอนโทรลเลอร์เพื่อรับสัญญาณจากตัวตรวจจับมาประมวลผลเป็นข้อมูลดิจิตอลและสัญญาณพัลส์สำหรับแจ้งผลการวัดทิศทาง

1. ตำแหน่งขาและการต่อใช้งาน

ในรูปที่ 1 แสดงรูปร่างหน้าตาและการจัดขาของ CMPS03 โมดูลเข็มทิศดิจิตอล จะเห็นว่าเป็นแผงวงจรที่มีคอนเน็กเตอร์ต่อออกมาเพื่อให้เชื่อมต่อไปใช้งาน อย่างไรก็ตามเพื่ออำนวยความสะดวกแก่ผู้ใช้งานกับบอร์ดควบคุมหุ่นยนต์ของบริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด (i-nex : เป็นตัวแทนจำหน่ายสินค้าของ Devantech ในประเทศไทยอย่างเป็นทางการ) จึงได้พัฒนาบอร์ดอะแดปเตอร์รุ่น ADX-CMPS03 เพื่อให้นำโมดูล CMPS03 มาติดตั้ง (โดยบอร์ด ADX-CMPS03 ต้องจัดซื้อแยก)

บนบอร์ด ADX-CMPS03 ได้จัดเตรียมคอนเน็กเตอร์ PCB 3 ขาตัวผู้สำหรับเชื่อมต่อกับบอร์ดควบคุมหุ่นยนต์ และคอนเน็กเตอร์ IDC ตัวเมียแถวเดี่ยว 4 ขาสำหรับเสียบสายต่อวงจรเบอร์ AWG#22 เพื่อต่อกับแผงต่อวงจรหรือเบรดบอร์ด นอกจากนั้นยังมีสวิตช์กดสำหรับปรับตั้งค่า (calibration) เพื่อกำหนดตำแหน่งทิศอ้างอิง โดยวงจรของบอร์ด ADX-CMPS03 แสดงในรูปที่ 2



รูปที่ 1 แสดงรูปร่างและตำแหน่งขาสำหรับการต่อใช้งาน

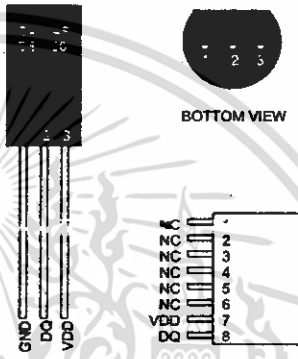
DALLAS SEMICONDUCTOR

DS1820
1–Wire™ Digital Thermometer

FEATURES

- Unique 1–Wire™ interface requires only one port pin for communication
- Multidrop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line
- Zero standby power required
- Measures temperatures from –55°C to +125°C in 0.5°C increments. Fahrenheit equivalent is –67°F to +257°F in 0.9°F increments
- Temperature is read as a 9–bit digital value.
- Converts temperature to digital word in 200 ms (typ.)
- User–definable, nonvolatile temperature alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



DS1820
PR35 PACKAGE
See Mech. Drawings Section

DS1820S
16–PIN SSOP
See Mech. Drawings Section

PIN DESCRIPTION

GND	–	Ground
DQ	–	Data In/Out
V_{DD}	–	Optional V_{DD}
NC	–	No Connect

DESCRIPTION

The DS1820 Digital Thermometer provides 9–bit temperature readings which indicate the temperature of the device.

Information is sent to/from the DS1820 over a 1–Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS1820. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS1820 contains a unique silicon serial number, multiple DS1820s can exist on the same 1–Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls, sensing temperatures inside buildings, equipment or machinery, and in process monitoring and control.

DETAILED PIN DESCRIPTION

PIN 16–PIN SSOP	PIN PR35	SYMBOL	DESCRIPTION
9	1	GND	**Ground.**
8	2	DQ	**Data Input/Output pin.** For 1–Wire operation: Open drain. (See "Parasite Power" section.)
7	3	V_{DD}	**Optional V_{DD} pin.** See "Parasite Power" section for details of connection.

DS1820S (16–pin SSOP): All pins not specified in this table are not to be connected.

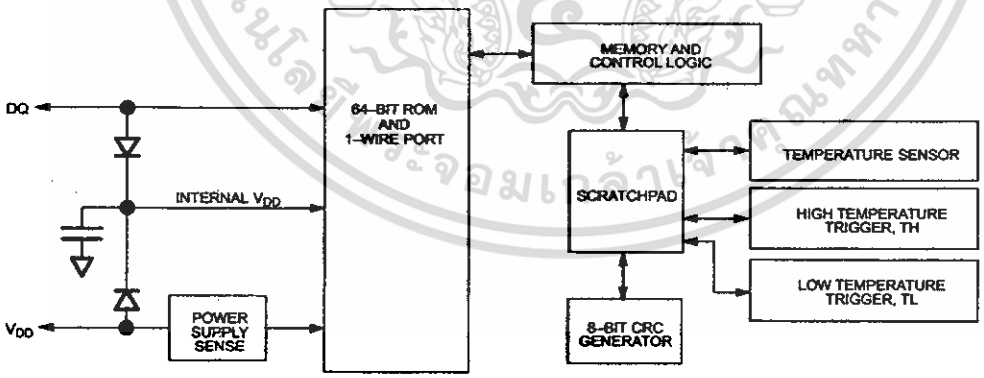
OVERVIEW

The block diagram of Figure 1 shows the major components of the DS1820. The DS1820 has three main data components: 1) 64–bit lasered ROM, 2) temperature sensor, and 3) nonvolatile temperature alarm triggers TH and TL. The device derives its power from the 1–Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1–Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS1820 may also be powered from an external 5 volts supply.

Communication to the DS1820 is via a 1–Wire port. With the 1–Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64–bit lasered ROM portion of each device and can single out a specific device if many are present on the 1–Wire line as well as indicate to the Bus Master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands.

One control function command instructs the DS1820 to perform a temperature measurement. The result of this measurement will be placed in the DS1820's scratchpad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of one byte EEPROM each. If the alarm search command is not applied to the DS1820, these registers may be used as general purpose user memory. Writing TH and TL is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.

DS1820 BLOCK DIAGRAM Figure 1



PARASITE POWER

The block diagram (Figure 1) shows the parasite powered circuitry. This circuitry "steals" power whenever the I/O or V_{DD} pins are high. I/O will provide sufficient power as long as the specified timing and voltage requirements are met (see the section titled "1–Wire Bus System"). The advantages of parasite power are two–fold: 1) by parasiting off this pin, no local power source is needed for remote sensing of temperature, and 2) the ROM may be read in absence of normal power.

In order for the DS1820 to be able to perform accurate temperature conversions, sufficient power must be provided over the I/O line when a temperature conversion is taking place. Since the operating current of the DS1820 is up to 1 mA, the I/O line will not have sufficient drive due to the 5K pull–up resistor. This problem is particularly acute if several DS1820's are on the same I/O and attempting to convert simultaneously.

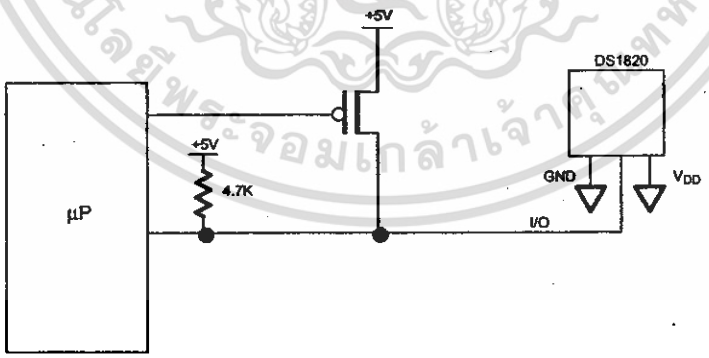
There are two ways to assure that the DS1820 has sufficient supply current during its active conversion cycle. The first is to provide a strong pull–up on the I/O line whenever temperature conversions or copies to the E^2 memory are taking place. This may be accomplished by using a MOSFET to pull the I/O line directly to the power supply as shown in Figure 2. The I/O line must be switched over to the strong pull–up within 10 µs maximum after issuing any protocol that involves copying to the E^2 memory or initiates temperature conversions. When using the parasite power mode, the V_{DD} pin must be tied to ground.

Another method of supplying current to the DS1820 is through the use of an external power supply tied to the V_{DD} pin, as shown in Figure 3. The advantage to this is that the strong pull–up is not required on the I/O line, and the bus master need not be tied up holding that line high during temperature conversions. This allows other data traffic on the 1–Wire bus during the conversion time. In addition, any number of DS1820's may be placed on the 1–Wire bus, and if they all use external power, they may all simultaneously perform temperature conversions by issuing the Skip ROM command and then issuing the Convert T command. Note that as long as the external power supply is active, the GND pin may not be floating.

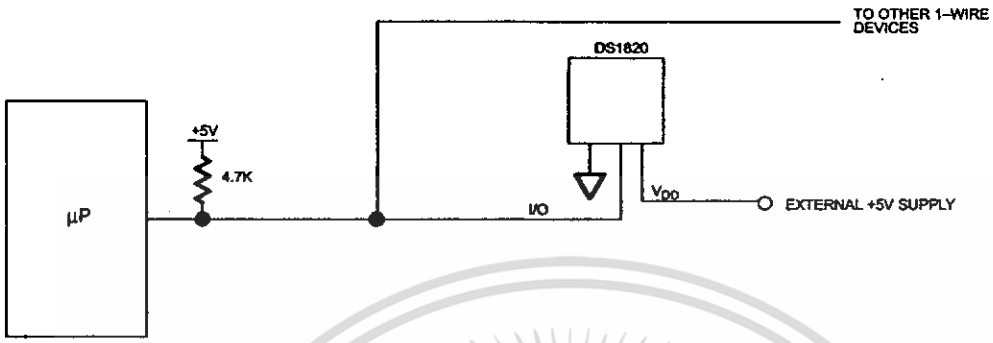
The use of parasite power is not recommended above 100°C, since it may not be able to sustain communications given the higher leakage currents the DS1820 exhibits at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that V_{DD} be applied to the DS1820.

For situations where the bus master does not know whether the DS1820's on the bus are parasite powered or supplied with external V_{DD}, a provision is made in the DS1820 to signal the power supply scheme used. The bus master can determine if any DS1820's are on the bus which require the strong pull–up by sending a Skip ROM protocol, then issuing the read power supply command. After this command is issued, the master then issues read time slots. The DS1820 will send back "0" on the 1–Wire bus if it is parasite powered; it will send back a "1" if it is powered from the V_{DD} pin. If the master receives a "0", it knows that it must supply the strong pull–up on the I/O line during temperature conversions. See "Memory Command Functions" section for more detail on this command protocol.

STRONG PULL–UP FOR SUPPLYING DS1820 DURING TEMPERATURE CONVERSION Figure 2



USING V_{DD} TO SUPPLY TEMPERATURE CONVERSION CURRENT Figure 3



OPERATION – MEASURING TEMPERATURE

The DS1820 measures temperature through the use of an on–board proprietary temperature measurement technique. A block diagram of the temperature measurement circuitry is shown in Figure 4.

The DS1820 measures temperature by counting the number of clock cycles that an oscillator with a low temperature coefficient goes through during a gate period determined by a high temperature coefficient oscillator. The counter is preset with a base count that corresponds to –55°C. If the counter reaches zero before the gate period is over, the temperature register, which is also preset to the –55°C value, is incremented, indicating that the temperature is higher than –55°C.

At the same time, the counter is then preset with a value determined by the slope accumulator circuitry. This circuitry is needed to compensate for the parabolic behavior of the oscillators over temperature. The counter is then clocked again until it reaches zero. If the gate period is still not finished, then this process repeats.

The slope accumulator is used to compensate for the non–linear behavior of the oscillators over temperature, yielding a high resolution temperature measurement. This is done by changing the number of counts necessary for the counter to go through for each incremental degree in temperature. To obtain the desired resolution, therefore, both the value of the counter and the number of counts per degree C (the value of the slope accumulator) at a given temperature must be known.

Internally, this calculation is done inside the DS1820 to provide 0.5°C resolution. The temperature reading is provided in a 16–bit, sign–extended two's complement reading. Table 1 describes the exact relationship of output data to measured temperature. The data is transmitted serially over the 1–Wire interface. The DS1820 can measure temperature over the range of –55°C to +125°C in 0.5°C increments. For Fahrenheit usage, a lookup table or conversion factor must be used.

Note that temperature is represented in the DS1820 in terms of a $^1/_2$°C LSB, yielding the following 9–bit format:

MSB								LSB
1	1	1	0	0	1	1	1	0

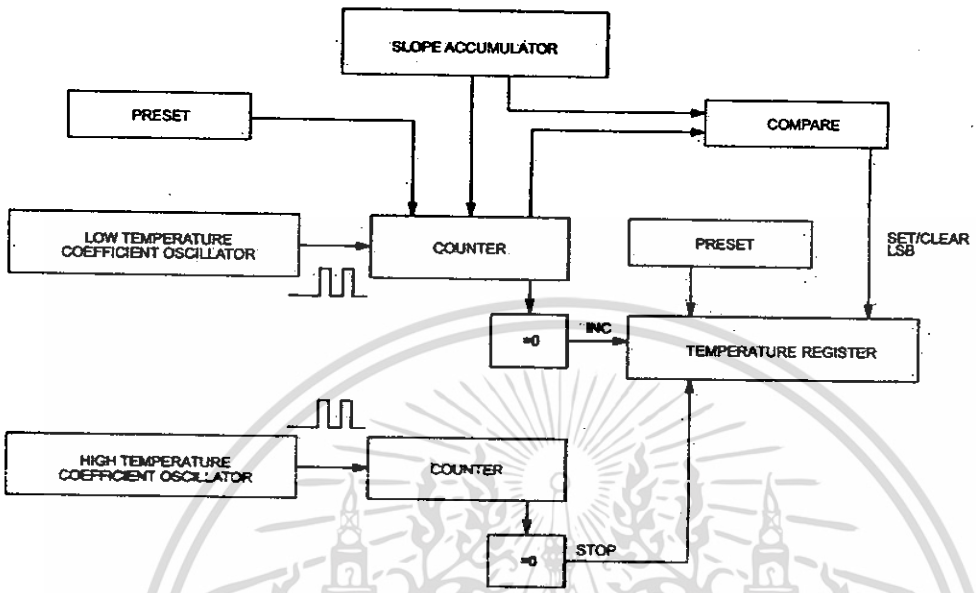
= –25°C

The most significant (sign) bit is duplicated into all of the bits in the upper MSB of the two–byte temperature register in memory. This "sign–extension" yields the 16–bit temperature readings as shown in Table 1.

Higher resolutions may be obtained by the following procedure. First, read the temperature, and truncate the 0.5°C bit (the LSB) from the read value. This value is TEMP_READ. The value left in the counter may then be read. This value is the count remaining (COUNT_REMAIN) after the gate period has ceased. The last value needed is the number of counts per degree C (COUNT_PER_C) at that temperature. The actual temperature may be then be calculated by the user using the following:

$$\text{TEMPERATURE} = \text{TEMP\_READ} - 0.25 + \frac{(\text{COUNT\_PER\_C} - \text{COUNT\_REMAIN})}{\text{COUNT\_PER\_C}}$$

TEMPERATURE MEASURING CIRCUITRY Figure 4



TEMPERATURE/DATA RELATIONSHIPS Table 1

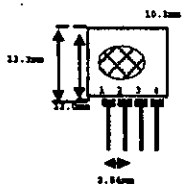
TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	00000000 11111010	00FA
+25°C	00000000 00110010	0032h
$+^1/_2$°C	00000000 00000001	0001h
+0°C	00000000 00000000	0000h
$-^1/_2$°C	11111111 11111111	FFFFh
-25°C	11111111 11001110	FFCEh
-55°C	11111111 10010010	FF92h

OPERATION – ALARM SIGNALING

After the DS1820 has performed a temperature conversion, the temperature value is compared to the trigger values stored in TH and TL. Since these registers are 8–bit only, the 0.5°C bit is ignored for comparison. The most significant bit of TH or TL directly corresponds to the sign bit of the 16–bit temperature register. If the result of a temperature measurement is higher than TH or lower than TL, an alarm flag inside the device is set. This flag is updated with every temperature measurement. As long as the alarm flag is set, the DS1820 will respond to the alarm search command. This allows many DS1820s to be connected in parallel doing simultaneous temperature measurements. If somewhere the temperature exceeds the limits, the alarming device(s) can be identified and read immediately without having to read non–alarming devices.

TLP434A & RLP434A RF ASK Hybrid Modules for Radio Control (New Version)

TLP434A Ultra Small Transmitter



pin 1 : GND
pin 2 : Data in
pin 3 : Vcc
pin 4 : Antenna (RF output)

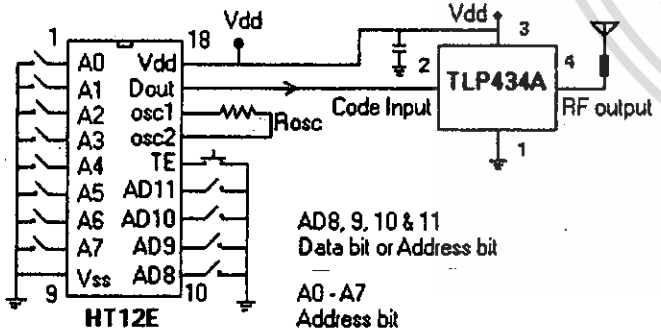
Frequency 315, 418 and 433.92 Mhz

Modulation : ASK
Operation Voltage : 2 - 12 VDC

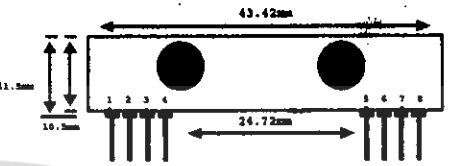
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Vcc	Operating supply voltage		2.0	-	12.0	V
Icc 1	Peak Current (2V)		-	-	1.64	mA
Icc 2	Peak Current (12V)		-	-	19.4	mA
Vh	Input High Voltage	Idata= 100uA (High)	Vcc-0.5	Vcc	Vcc+0.5	V
Vl	Input Low Voltage	Idata= 0 uA (Low)	-	-	0.3	V
FO	Absolute Frequency	315Mhz module	314.8	315	315.2	MHz
PO	RF Output Power- 50ohm	Vcc = 9V-12V	-	16	-	dBm
		Vcc = 5V-6V	-	14	-	dBm
DR	Data Rate	External Encoding	512	4.8K	200K	bps

Notes : (Case Temperature = 25°C +- 2°C , Test Load Impedance = 50 ohm)

Application Circuit :
Typical Key-chain Transmitter using HT12E-18DIP, a Binary 12 bit Encoder from Holtek Semiconductor Inc.



RLP434A SAW Based Receiver



pin 1 : Gnd
pin 2 : Digital Data Output
pin 3 : Linear Output /Test
pin 4 : Vcc
pin 5 : Vcc
pin 6 : Gnd
pin 7 : Gnd
pin 8 : Antenna

Frequency 315, 418 and 433.92 Mhz

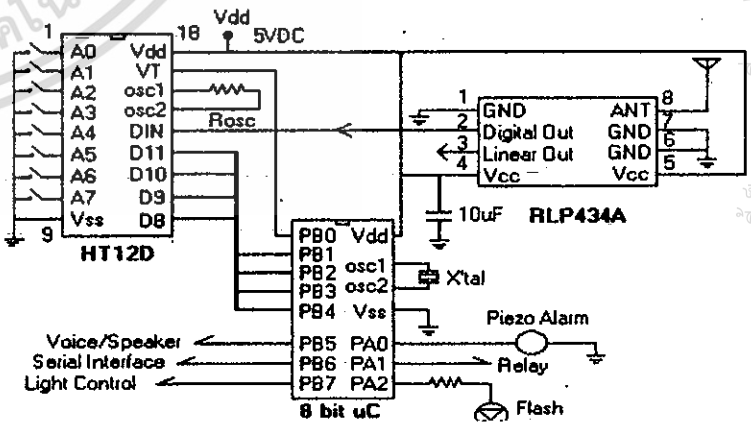
Modulation : ASK
Supply Voltage : 3.3 - 6.0 VDC
Output : Digital & Linear

Symbol	Parameter	Conditions	Min	Typ	Max	
Vcc	Operating supply voltage		3.3	5.0V	6.0	V
Itot	Operating Current		-	4.5		mA
Vdata	Data Out	Idata = +200 uA (High)	Vcc-0.5	-	Vcc	V
		Idata = -10 uA (Low)	-	-	0.3	V

Electrical Characteristics

Characteristics	SYM	Min	Typ	Max	Unit
Operation Radio Frequency	FC		315, 418 and 433.92		MHz
Sensitivity	Pref		-110		dBm
Channel Width			+-500		Khz
Noise Equivalent BW			4		Khz
Receiver Turn On Time			5		ms
Operation Temperature	Top	-20	-	80	C
Baseboard Data Rate			4.8		KHz

Application Circuit :
Typical RF Receiver using HT12D-18DIP, a Binary 12 bit Decoder with 8 bit uC HT48RXX from Holtek Semiconductor Inc.



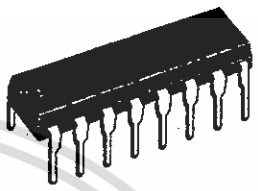
Laipac Technology, Inc.
105 West Beaver Creek Rd. Unit 207 Richmond Hill Ontario L4B 1C6 Canada

LAIPAC

ULN2001A-ULN2002A
ULN2003A-ULN2004A

SEVEN DARLINGTON ARRAYS

- SEVEN DARLINGTONS PER PACKAGE
- OUTPUT CURRENT 500mA PER DRIVER (600mA PEAK)
- OUTPUT VOLTAGE 50V
- INTEGRATED SUPPRESSION DIODES FOR INDUCTIVE LOADS
- OUTPUTS CAN BE PARALLELED FOR HIGHER CURRENT
- TTL/CMOS/PMOS/DTL COMPATIBLE INPUTS
- INPUTS PINNED OPPOSITE OUTPUTS TO SIMPLIFY LAYOUT



DIP16

ORDERING NUMBERS: ULN2001A/2A/3A/4A

SO16

ORDERING NUMBERS: ULN2001D/2D/3D/4D

DESCRIPTION

The ULN2001A, ULN2002A, ULN2003 and ULN2004A are high voltage, high current darlington arrays each containing seven open collector darlington pairs with common emitters. Each channel rated at 500mA and can withstand peak currents of 600mA. Suppression diodes are included for inductive load driving and the inputs are pinned opposite the outputs to simplify board layout.

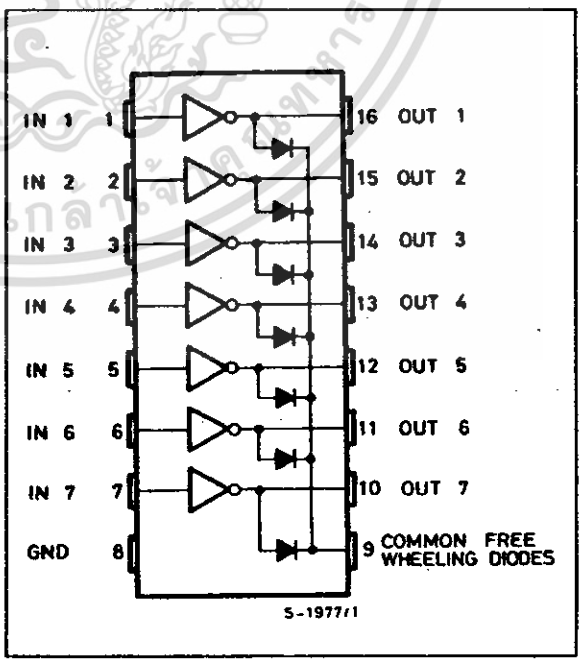
The four versions interface to all common logic families :

ULN2001A	General Purpose, DTL, TTL, PMOS, CMOS
ULN2002A	14-25V PMOS
ULN2003A	5V TTL, CMOS
ULN2004A	6–15V CMOS, PMOS

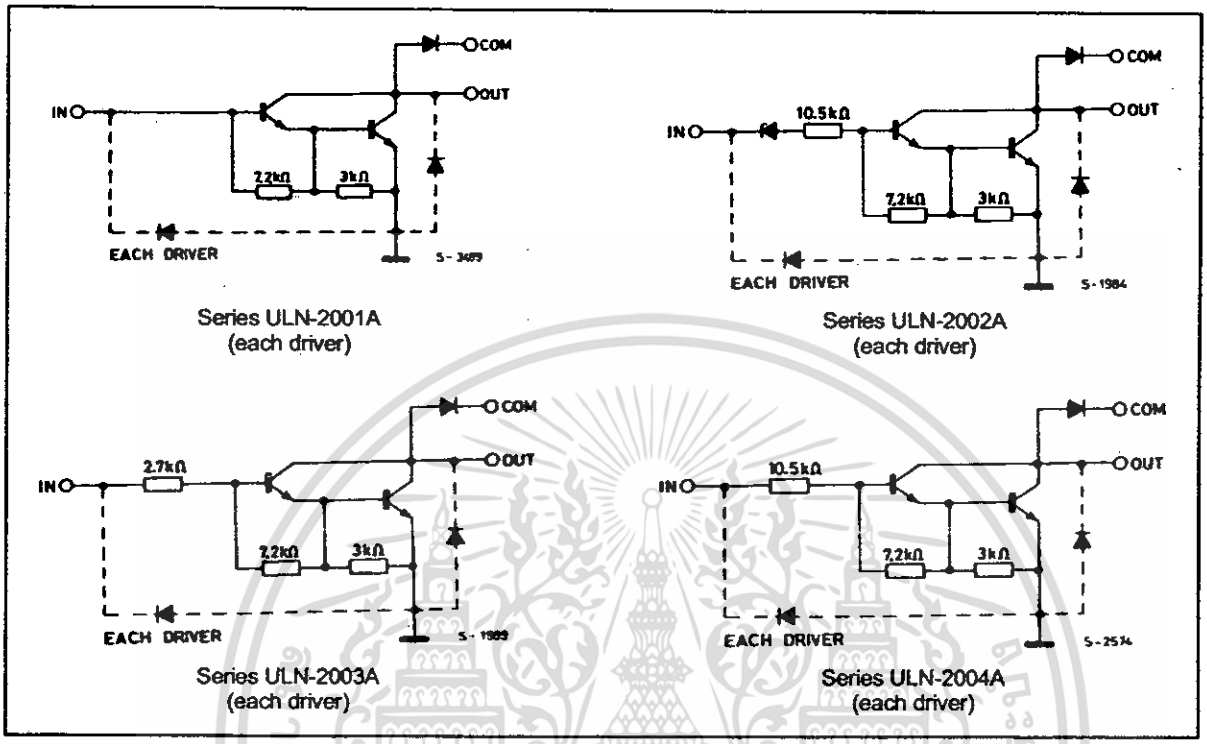
These versatile devices are useful for driving a wide range of loads including solenoids, relays DC motors, LED displays filament lamps, thermal printheads and high power buffers.

The ULN2001A/2002A/2003A and 2004A are supplied in 16 pin plastic DIP packages with a copper leadframe to reduce thermal resistance. They are available also in small outline package (SO-16) as ULN2001D/2002D/2003D/2004D.

PIN CONNECTION



SCHEMATIC DIAGRAM



Series ULN-2001A (each driver)

Series ULN-2002A (each driver)

Series ULN-2003A (each driver)

Series ULN-2004A (each driver)

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_o	Output Voltage	50	V
V_{in}	Input Voltage (for ULN2002A/D - 2003A/D - 2004A/D)	30	V
I_c	Continuous Collector Current	500	mA
I_b	Continuous Base Current	25	mA
T_{amb}	Operating Ambient Temperature Range	– 20 to 85	°C
T_{stg}	Storage Temperature Range	– 55 to 150	°C
T_j	Junction Temperature	150	°C

THERMAL DATA

Symbol	Parameter		DIP16	SO16	Unit
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	70	120	°C/W

ELECTRICAL CHARACTERISTICS (T_{amb} = 25°C unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit	Fig.
I_{CEX}	Output Leakage Current	V_{CE} = 50V			50	µA	1a
		T_{amb} = 70°C, V_{CE} = 50V			100	µA	1a
		T_{amb} = 70°C for ULN2002A V_{CE} = 50V, V_i = 6V			500	µA	1b
		for ULN2004A V_{CE} = 50V, V_i = 1V			500	µA	1b
$V_{CE(sat)}$	Collector-emitter Saturation Voltage	I_C = 100mA, I_B = 250µA		0.9	1.1	V	2
		I_C = 200 mA, I_B = 350µA		1.1	1.3	V	2
		I_C = 350mA, I_B = 500µA		1.3	1.6	V	2
$I_{i(on)}$	Input Current	for ULN2002A, V_i = 17V		0.82	1.25	mA	3
		for ULN2003A, V_i = 3.85V		0.93	1.35	mA	3
		for ULN2004A, V_i = 5V		0.35	0.5	mA	3
		V_i = 12V		1	1.45	mA	3
$I_{i(off)}$	Input Current	T_{amb} = 70°C, I_C = 500µA	50	65		µA	4
$V_{i(on)}$	Input Voltage	V_{CE} = 2V				V	5
		for ULN2002A I_C = 300mA			13		
		for ULN2003A I_C = 200mA			2.4		
		I_C = 250mA			2.7		
		I_C = 300mA			3		
		for ULN2004A I_C = 125mA			5		
		I_C = 200mA			6		
		I_C = 275mA			7		
		I_C = 350mA			8		
h_{FE}	DC Forward Current Gain	for ULN2001A V_{CE} = 2V, I_C = 350mA	1000				2
C_i	Input Capacitance			15	25	pF	
t_{PLH}	Turn-on Delay Time	0.5 V_i to 0.5 V_o		0.25	1	µs	
t_{PHL}	Turn-off Delay Time	0.5 V_i to 0.5 V_o		0.25	1	µs	
I_R	Clamp Diode Leakage Current	V_R = 50V			50	µA	6
		T_{amb} = 70°C, V_R = 50V			100	µA	6
V_F	Clamp Diode Forward Voltage	I_F = 350mA		1.7	2	V	7

19-4323; Rev 11; 2/03

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multidrop RS-232 Networks

Features

Superior to Bipolar

- **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- **Meet All EIA/TIA-232E and V.28 Specifications**
- **Multiple Drivers and Receivers**
- **3-State Driver and Receiver Outputs**
- **Open-Line Detection (MAX243)**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.1	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package

For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.

ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (V_{CC}) ..... -0.3V to +6V
Input Voltages
T_{IN} ..... -0.3V to (V_{CC} - 0.3V)
R_{IN} (Except MAX220) ..... ±30V
R_{IN} (MAX220) ..... ±25V
T_{OUT} (Except MAX220) (Note 1) ..... ±15V
T_{OUT} (MAX220) ..... ±13.2V
Output Voltages
T_{OUT} ..... ±15V
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Driver/Receiver Output Short Circuited to GND ..... Continuous
Continuous Power Dissipation (T_A = +70°C)
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C) 842mW
18-Pin Plastic DIP (derate 11.11mW/°C above +70°C) 889mW
20-Pin Plastic DIP (derate 8.00mW/°C above +70°C) 440mW
16-Pin Narrow SO (derate 8.70mW/°C above +70°C) 696mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
18-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
20-Pin Wide SO (derate 10.00mW/°C above +70°C) 800mW
20-Pin SSOP (derate 8.00mW/°C above +70°C) 640mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C) 800mW
18-Pin CERDIP (derate 10.53mW/°C above +70°C) 842mW
Operating Temperature Ranges
MAX2_ _AC_ _, MAX2_ _C_ _ ..... 0°C to +70°C
MAX2_ _AE_ _, MAX2_ _E_ _ ..... -40°C to +85°C
MAX2_ _AM_ _, MAX2_ _M_ _ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10s) ..... +300°C

Note 1: Input voltage measured with T_{OUT} in high-impedance state, $\overline{SHDN}$ or V_{CC} = 0V.
Note 2: For the MAX220, V+ and V- can have a maximum magnitude of 7V, but their absolute difference cannot exceed 13V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High	All devices except MAX220		2	1.4		V
	MAX220: V_{CC} = 5.0V		2.4			
Logic Pull-Up/Input Current	All except MAX220, normal operation			5	40	µA
	$\overline{SHDN}$ = 0V, MAX222/242, shutdown, MAX220			±0.01	±1	
Output Leakage Current	V_{CC} = 5.5V, $\overline{SHDN}$ = 0V, V_{OUT} = ±15V, MAX222/242			±0.01	±10	µA
	V_{CC} = $\overline{SHDN}$ = 0V, V_{OUT} = ±15V			±0.01	±10	
Data Rate				200	116	kbps
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V			±7	±22	mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
RS-232 Input Threshold Low	V_{CC} = 5V	All except MAX243 $R2_{IN}$	0.8	1.3		V
		MAX243 $R2_{IN}$ (Note 2)	-3			
RS-232 Input Threshold High	V_{CC} = 5V	All except MAX243 $R2_{IN}$		1.8	2.4	V
		MAX243 $R2_{IN}$ (Note 2)		-0.5	-0.1	
RS-232 Input Hysteresis	All except MAX243, V_{CC} = 5V, no hysteresis in shdn.		0.2	0.5	1	V
	MAX243			1		
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243 (continued)

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
TTL/CMOS Output Leakage Current	$\overline{SHDN}$ = V_{CC} or $\overline{EN}$ = V_{CC} ($\overline{SHDN}$ = 0V for MAX222), 0V ≤ V_{OUT} ≤ V_{CC}			±0.05	±10	µA
$\overline{EN}$ Input Threshold Low	MAX242			1.4	0.8	V
$\overline{EN}$ Input Threshold High	MAX242		2.0	1.4		V
Operating Supply Voltage			4.5		5.5	V
V_{CC} Supply Current ($\overline{SHDN}$ = V_{CC}), Figures 5, 6, 11, 19	No load	MAX220		0.5	2	mA
		MAX222/232A/233A/242/243		4	10	
	3kΩ load both inputs	MAX220		12		
		MAX222/232A/233A/242/243		15		
Shutdown Supply Current	MAX222/242	T_A = +25°C		0.1	10	µA
		T_A = 0°C to +70°C		2	50	
		T_A = -40°C to +85°C		2	50	
		T_A = -55°C to +125°C		35	100	
$\overline{SHDN}$ Input Leakage Current	MAX222/242				±1	µA
$\overline{SHDN}$ Threshold Low	MAX222/242			1.4	0.8	V
$\overline{SHDN}$ Threshold High	MAX222/242		2.0	1.4		V
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V	MAX222/232A/233A/242/243	6	12	30	V/µs
		MAX220	1.5	3	30	
Transmitter Propagation Delay TLL to RS-232 (Normal Operation), Figure 1	t_{PHLT}	MAX222/232A/233A/242/243		1.3	3.5	µs
		MAX220		4	10	
	t_{PLHT}	MAX222/232A/233A/242/243		1.5	3.5	
		MAX220		5	10	
Receiver Propagation Delay RS-232 to TLL (Normal Operation), Figure 2	t_{PHLR}	MAX222/232A/233A/242/243		0.5	1	µs
		MAX220		0.6	3	
	t_{PLHR}	MAX222/232A/233A/242/243		0.6	1	
		MAX220		0.8	3	
Receiver Propagation Delay RS-232 to TLL (Shutdown), Figure 2	t_{PHLS}	MAX242		0.5	10	µs
	t_{PLHS}	MAX242		2.5	10	
Receiver-Output Enable Time, Figure 3	t_{ER}	MAX242		125	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}	MAX242		160	500	ns
Transmitter-Output Enable Time ($\overline{SHDN}$ Goes High), Figure 4	t_{ET}	MAX222/242, 0.1µF caps (includes charge-pump start-up)		250		µs
Transmitter-Output Disable Time ($\overline{SHDN}$ Goes Low), Figure 4	t_{DT}	MAX222/242, 0.1µF caps		600		ns
Transmitter + to - Propagation Delay Difference (Normal Operation)	t_{PHLT} - t_{PLHT}	MAX222/232A/233A/242/243		300		ns
		MAX220		2000		
Receiver + to - Propagation Delay Difference (Normal Operation)	t_{PHLR} - t_{PLHR}	MAX222/232A/233A/242/243		100		ns
		MAX220		225		

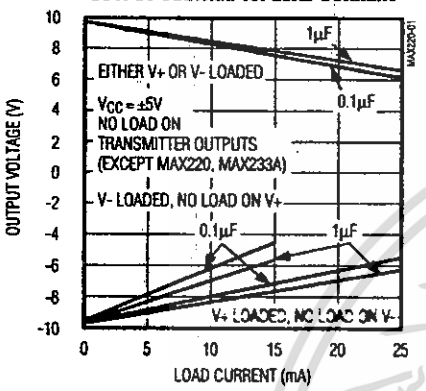
Note 3: MAX243 $R2_{OUT}$ is guaranteed to be low when $R2_{IN}$ is ≥ 0V or is floating.

+5V-Powered, Multichannel RS-232 Drivers/Receivers

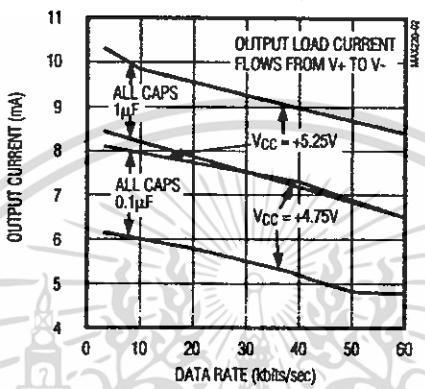
Typical Operating Characteristics

MAX220/MAX222/MAX232A/MAX233A/MAX242/MAX243

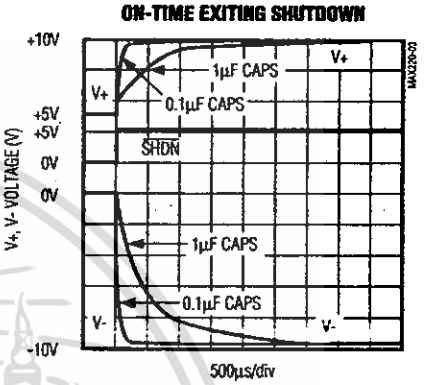
OUTPUT VOLTAGE vs. LOAD CURRENT

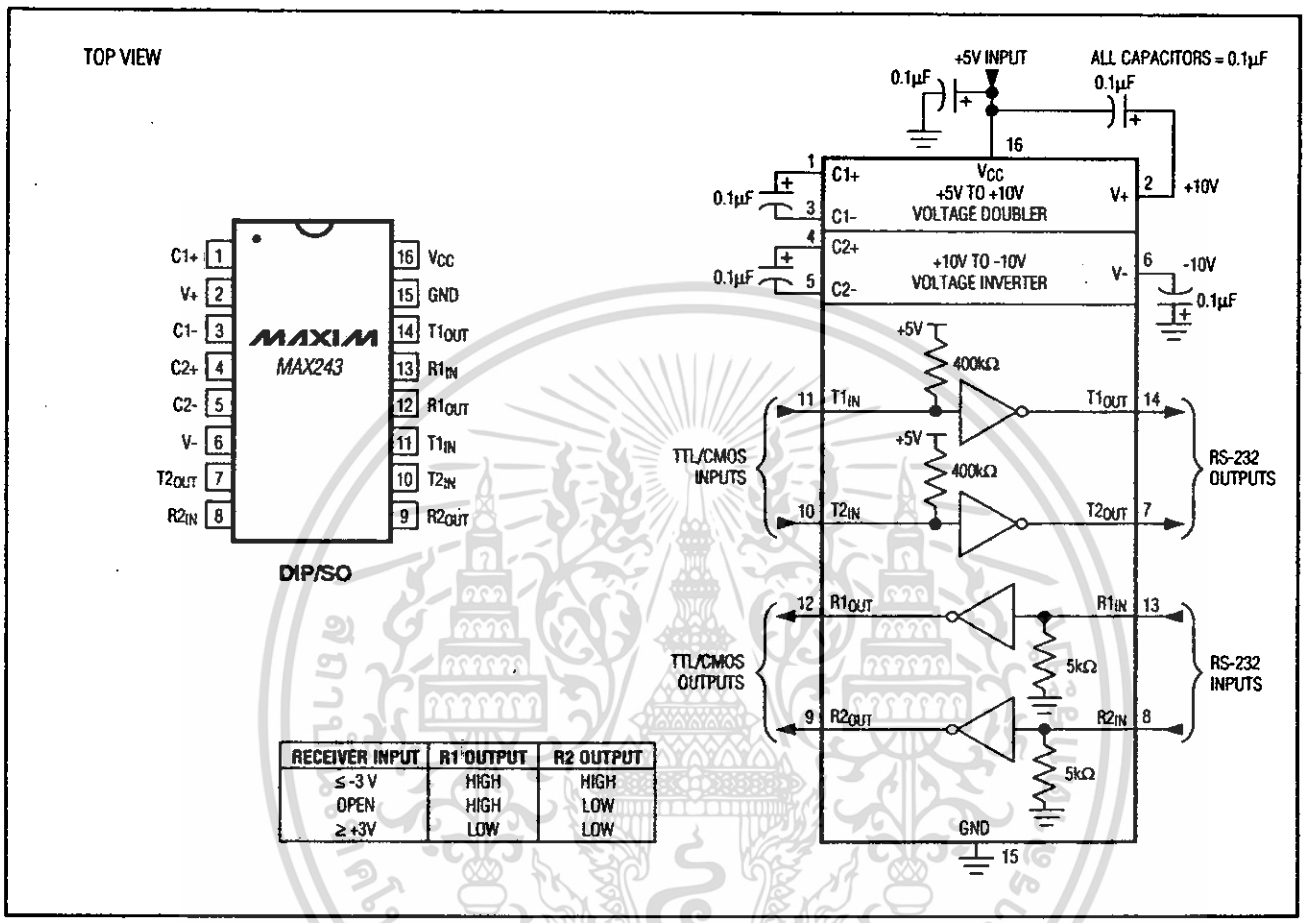


AVAILABLE OUTPUT CURRENT vs. DATA RATE



MAX222/MAX242 ON-TIME EXITING SHUTDOWN





RECEIVER INPUT	R1 OUTPUT	R2 OUTPUT
≤-3 V	HIGH	HIGH
OPEN	HIGH	LOW
≥+3V	LOW	LOW

Figure 19. MAX243 Pin Configuration and Typical Operating Circuit

Features

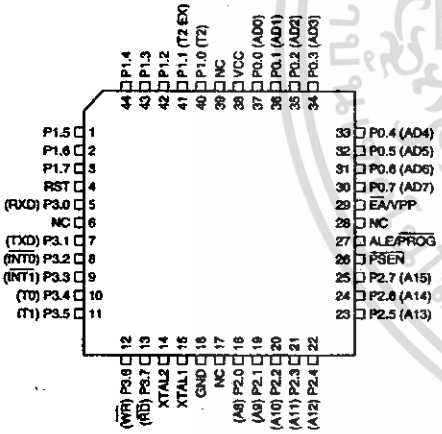
- Compatible with MCS-51™ Products
- 8K Bytes of In-System Reprogrammable Flash Memory
- Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Eight Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

Description

The AT89C52 is a low-power, high-performance CMOS 8-bit microcomputer with 8K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 80C51 and 80C52 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C52 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

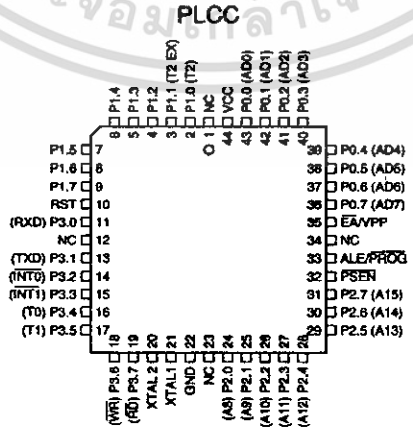
Pin Configurations

PQFP/TQFP



PDIP

PLCC

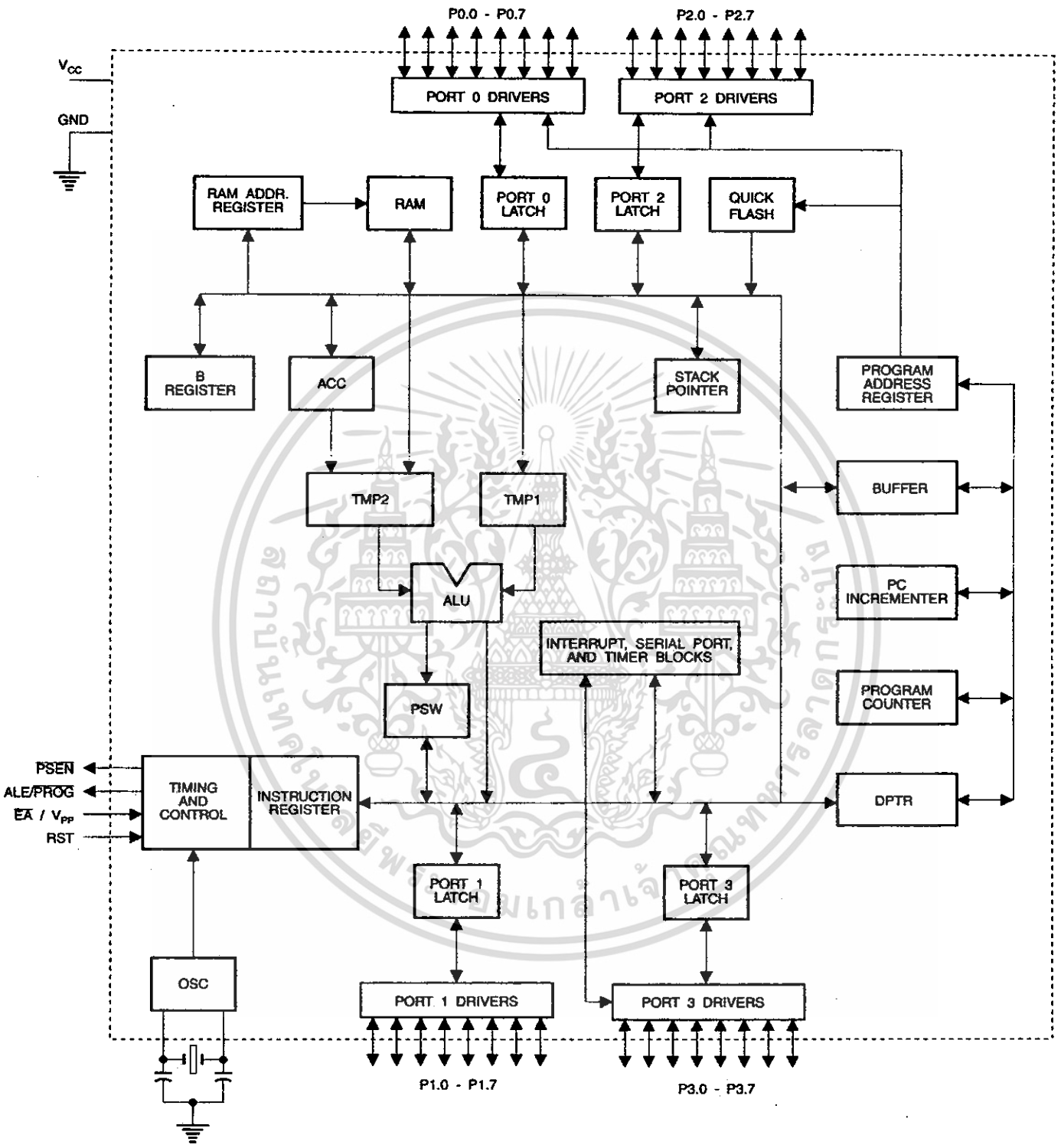


8-bit Microcontroller with 8K Bytes Flash

AT89C52

Rev. 0313H–02/00

Block Diagram



The AT89C52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89C52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next hardware reset.

Pin Description

VCC
Supply voltage.

GND
Ground.

Port 0
Port 0 is an 8-bit open drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1
Port 1 is an 8-bit bi-directional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

In addition, P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively, as shown in the following table.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port Pin	Alternate Functions
P1.0	T2 (external count input to Timer/Counter 2), clock-out
P1.1	T2EX (Timer/Counter 2 capture/reload trigger and direction control)

Port 2
Port 2 is an 8-bit bi-directional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3
Port 3 is an 8-bit bi-directional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51, as shown in the following table.

Port 3 also receives some control signals for Flash programming and verification.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST
Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$
Address Latch Enable is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external

timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external data memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

$\overline{\text{PSEN}}$

Program Store Enable is the read strobe to external program memory.

When the AT89C52 is executing code from external program memory, $\overline{\text{PSEN}}$ is activated twice each machine cycle, except that two $\overline{\text{PSEN}}$ activations are skipped during each access to external data memory.

$\overline{\text{EA}}$/VPP

External Access Enable. $\overline{\text{EA}}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{\text{EA}}$ will be internally latched on reset.

$\overline{\text{EA}}$ should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming when 12-volt programming is selected.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Table 1. AT89C52 SFR Map and Reset Values

0F8H								0FFH
0F0H	B 00000000							0F7H
0E8H								0EFH
0E0H	ACC 00000000							0E7H
0D8H								0DFH
0D0H	PSW 00000000							0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000		0CFH
0C0H								0C7H
0B8H	IP XX000000							0BFH
0B0H	P3 11111111							0B7H
0A8H	IE 0X000000							0AFH
0A0H	P2 11111111							0A7H
98H	SCON 00000000	SBUF XXXXXXXX						9FH
90H	P1 11111111							97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000		8FH
80H	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000				PCON 0XXX0000 87H

Special Function Registers

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in Table 1.

Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect.

User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

Timer 2 Registers Control and status bits are contained in registers T2CON (shown in Table 2) and T2MOD (shown in Table 4) for Timer 2. The register pair (RCAP2H, RCAP2L) are the Capture/Reload registers for Timer 2 in 16-bit capture mode or 16-bit auto-reload mode.

Interrupt Registers The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the six interrupt sources in the IP register.r

Table 2. T2CON – Timer/Counter 2 Control Registe

T2CON Address = 0C8H — Reset Value = 0000 0000B

Bit Addressable

Bit	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$
	7	6	5	4	3	2	1	0

Symbol	Function
TF2	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software. EXF2 does not cause an interrupt in up/down counter mode (DCEN = 1).
RCLK	Receive clock enable. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in serial port Modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	Transmit clock enable. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in serial port Modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	Timer 2 external enable. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	Start/Stop control for Timer 2. TR2 = 1 starts the timer.
C/$\overline{T2}$	Timer or counter select for Timer 2. C/$\overline{T2}$ = 0 for timer function. C/$\overline{T2}$ = 1 for external event counter (falling edge triggered).
CP/$\overline{RL2}$	Capture/Reload select. CP/$\overline{RL2}$ = 1 causes captures to occur on negative transitions at T2EX if EXEN2 = 1. CP/$\overline{RL2}$ = 0 causes automatic reloads to occur when Timer 2 overflows or negative transitions occur at T2EX when EXEN2 = 1. When either RCLK or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Data Memory

The AT89C52 implements 256 bytes of on-chip RAM. The upper 128 bytes occupy a parallel address space to the Special Function Registers. That means the upper 128 bytes have the same addresses as the SFR space but are physically separate from SFR space.

When an instruction accesses an internal location above address 7FH, the address mode used in the instruction specifies whether the CPU accesses the upper 128 bytes of RAM or the SFR space. Instructions that use direct addressing access SFR space.

For example, the following direct addressing instruction accesses the SFR at location 0A0H (which is P2).

```
MOV 0A0H, #data
```

โค้ดวิชวลเบสิก

```
Dim dataa(2)
 Dim dat(4)
Private Sub Command10_Click()
ezVidCap1.CaptureFile = txtAVI.Text
ezVidCap1.CaptureVideo
End Sub

Private Sub Command10_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
Command10.BackColor = &HC0&
End Sub

Private Sub Command11_Click()
ezVidCap1.CaptureEnd
End Sub


Private Sub Command1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = vbLeftButton Then
Command1.BackColor = &HC0&
Dim re1 As Integer
For re1 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H3)
MSComm1.Output = Chr(&H1F)
dataa(1) = 2
Next re1
End If
End Sub

Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Command1.BackColor = &HE0E0E0
End If
End Sub

Private Sub Command11_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
Command10.BackColor = &HE0E0E0
End Sub
```

```
Private Sub Command12_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim reA As Integer
For reA = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&HA)
MSComm1.Output = Chr(&H22)
Next reA
End If
End Sub

Private Sub Command13_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim reB As Integer
For reB = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&HB)
MSComm1.Output = Chr(&H23)
Next reB
End If
End Sub

Private Sub Command14_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim reC As Integer
For reC = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&HC)
MSComm1.Output = Chr(&H16)
Next reC
End If
End Sub
```

```
Private Sub Command15_Click()
dataa(2) = Text1.Text
End Sub

Private Sub Command2_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re2 As Integer
For re2 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H4)
MSComm1.Output = Chr(&H18)
dataa(1) = 1
Next re2
End If
End Sub

Private Sub Command2_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Command2.BackColor = &HE0E0E0
End If
End Sub

Private Sub Command3_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re3 As Integer
For re3 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H2)
MSComm1.Output = Chr(&H1E)
dataa(1) = 1
Next re3
End If
End Sub

Private Sub Command3_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Command3.BackColor = &HE0E0E0
```

```
End If
End Sub

Private Sub Command4_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re4 As Integer
For re4 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H1)
MSComm1.Output = Chr(&H1D)
dataa(1) = 3
Next re4
End If
End Sub

Private Sub Command4_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
MSComm1.Output = Chr(&H3)
Command4.BackColor = &HE0E0E0
End If
End Sub

Private Sub Command5_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re6 As Integer
For re6 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H6)
MSComm1.Output = Chr(&H1A)
Next re6
End If
End Sub

Private Sub Command5_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
MSComm1.Output = Chr(&H0)
End If
```

```
End Sub

Private Sub Command6_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re5 As Integer
For re5 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H5)
MSComm1.Output = Chr(&H19)
Next re5
End If
End Sub

Private Sub Command6_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
MSComm1.Output = Chr(&H9)
End If
End Sub

Private Sub Command7_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re8 As Integer
For re8 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H8)
MSComm1.Output = Chr(&H14)
Next re8
End If
End Sub

Private Sub Command7_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
MSComm1.Output = Chr(&H0)
End If
End Sub

Private Sub Command8_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
If Button = 1 Then
Dim re7 As Integer
For re7 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H7)
MSComm1.Output = Chr(&H1B)
Next re7
End If
End Sub

Private Sub Command8_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
MSComm1.Output = Chr(&H0)
End If
End Sub

Private Sub Command9_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Dim re9 As Integer
For re9 = 1 To 15
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&H22)
MSComm1.Output = Chr(&HFA)
MSComm1.Output = Chr(&HE6)
MSComm1.Output = Chr(&H9)
MSComm1.Output = Chr(&H15)
Next re9
End If
End Sub

Private Sub Form_Activate()
Dim intX As Integer, intY As Integer
picGraph.BackColor = vbBlack
picGraph.ForeColor = vbYellow
picGraph.DrawWidth = 1
picGraph.Scale (-10, 10)-(10, -10)
picGraph.Line (-10, 0)-(10, 0)
picGraph.Line (0, -10)-(0, 10)
For intX = -9 To 9
  picGraph.Line (intX, -0.2)-(intX, 0.2)
```

```
    If intX <> 0 Then
    picGraph.CurrentX = intX - 0.4
    picGraph.CurrentY = -0.2
    picGraph.Print (intX)
    End If
  Next intX
  For intY = -9 To 9
    picGraph.Line (-0.2, intY)-(0.2, intY)
    If intY <> 0 Then
    picGraph.CurrentX = -0.8
    picGraph.CurrentY = intY + 0.5
    picGraph.Print (intY)
    End If
  Next intY
  X = 0
  Y = 0
  End Sub

  Private Sub Form_Load()
  MSComm1.CommPort = 1
  MSComm1.Settings = "1200,n,8,1"
  MSComm1.PortOpen = True
  MSComm1.RThreshold = 1
  MSComm1.InputLen = 1
  End Sub

  Private Sub Form_Unload(Cancel As Integer)
  MSComm1.PortOpen = False
  End Sub

  Private Sub MSComm1_OnComm()
  On Error Resume Next
  Static X, Y, Data6 As Single
  If MSComm1.CommEvent = comEvReceive Then
  Data1 = Asc(MSComm1.Input)
  Data2 = Asc(MSComm1.Input)
  Data3 = Asc(MSComm1.Input)
  Data4 = Asc(MSComm1.Input)
  Data5 = ((Data3 * 256) + Data4) / 10
  Label1(0).Caption = Data5
  If Data1 > 127 Then Data1 = Data1 - 128
  Data6 = Data1 / 5
  Select Case Data2
```

```
Case Is > 128
Data2 = Data2 - 128
Label1(1).Caption = Data2 & ".5 C"
Case Is < 128
Label1(1).Caption = Data2 & " C"
End Select
Select Case dataa(1)
Case 1
Data6 = Data6 - Data6
Data5 = Data5 / Text1.Text
Case 3
Data5 = Data5 + 180
Data6 = Data6 / 1
End Select
X2 = Data6 * Cos(Data5 * 0.0174603)
Y2 = Data6 * Sin(Data5 * 0.0174603)
picGraph.Line (X, Y)-(X2 + X, Y2 + Y)
X = X2 + X
Y = Y2 + Y
Debug.Print Data1
Debug.Print Data6
Debug.Print "END"
End If
End Sub
```

โค้ดตัวรับที่รถ

```
FLAG EQU 20H
        ORG 0000H
        MOV TMOD,#20H
        MOV TH1,#0E8H
        MOV TL1,#0E8H
        MOV SCON,#01010000B
        SETB TR1
        MOV R4,#00010000B
        MOV R5,#00000001B
        MOV P0,#00H
        MOV P1,#00H
        SETB P1.4

WAIT:   MOV R1,#10
        JNB RI,$
        MOV A,SBUF
        CLR RI

    CJNE A,#250,WAIT
        JNB RI,$
        MOV A,SBUF
        CLR RI

        CJNE A,#230,WAIT
        JNB RI,$
        MOV A,SBUF
        CLR RI

        MOV R0,A
        JNB RI,$
        MOV A,SBUF
        CLR RI

    MOV FLAG,A
        MOV A,#250
        XRL A,#230
        XRL A,R0
        CJNE A,FLAG,WAIT
        MOV A,R0

STOP:   CJNE A,#05H,FORWARD
        MOV P1,#00H
        AJMP WAIT

FORWARD: CJNE A,#01H,RIGHT
```

```
        MOV P1,#00010101B
        MOV A,#01H
        MOV SBUF,A
        JNB TI,$
        CLR TI
        AJMP WAIT

RIGHT:  CJNE A,#02H,BACK
        MOV P1,#00011001B
        MOV A,#02H
        MOV SBUF,A
        JNB TI,$
        CLR TI
        AJMP WAIT

BACK:   CJNE A,#03H,LEFT
        MOV P1,#00011010B
        MOV A,#03H
        MOV SBUF,A
        JNB TI,$
        CLR TI
        AJMP WAIT

LEFT:   CJNE A,#04H,CAMERA_UP
        MOV P1,#00010110B
        MOV A,#04H
        MOV SBUF,A
        JNB TI,$
        CLR TI
        AJMP WAIT

CAMERA_UP:      CJNE A,#06H,CAMERA_DOWN
                DJNZ R1,CONTI1
                LJMP WAIT
CONTI1:         MOV A,R4
                CLR C
                RLC A
                JNC NEXT1
                MOV A,#00010000B
NEXT1:          MOV R4,A
                MOV P2,R4
                ACALL DELAY_100ms
                AJMP CAMERA_UP

CAMERA_DOWN:    CJNE A,#07H,CAMERA_LEFT
                DJNZ R1,CONTI2
```

```
                LJMP WAIT
CONTI2:         MOV A,R4
                SWAP A
                CLR C
                RRC A
                JNC NEXT3
                MOV A,#00001000B
NEXT3:          SWAP A
                MOV R4,A
                MOV P2,R4
                ACALL DELAY_100ms
                AJMP CAMERA_DOWN


CAMERA_LEFT:    CJNE A,#08H,CAMERA_RIGHT
                DJNZ R1,CONTI3
                LJMP WAIT
CONTI3:         MOV A,R5
                SWAP A
                CLR C
                RLC A
                JNC NEXT2
                MOV A,#00010000B
NEXT2:          SWAP A
                MOV R5,A
                MOV P2,R5
                ACALL DELAY_100ms
                AJMP CAMERA_LEFT

CAMERA_RIGHT:   CJNE A,#09H,HOME
                DJNZ R1,CONTI4
HOME:           LJMP WAIT
CONTI4:         MOV A,R5
                CLR C
                RRC A
                JNC NEXT4
                MOV A,#00001000B
NEXT4:          MOV R5,A
                MOV P2,R5
                ACALL DELAY_100ms
                AJMP CAMERA_RIGHT

DELAY_100ms:    MOV R7,#26
DELAY_100ms_1:  MOV R6,#0E6H
DELAY_100ms_2:  NOP
                NOP
```

```
DJNZ R6,DELAY_100ms_2
DJNZ R7,DELAY_100ms_1
RET
END
```

```
SDA             BIT     P1.0    ; SDA I2C Bus
SCL             BIT     P1.1    ; SCL I2C Bus
;-------------------------------------------------------------
; Define User Register
;-------------------------------------------------------------
FLAG            EQU     02FH    ; User FLAG
I2C_ACK         BIT     FLAG.0  ; Define I2C Acknowledge as bit

;-------------------------------------------------------------
; Define User Register
;-------------------------------------------------------------
I2C_ADDR EQU    033H    ; For keep I2C Address
I2C_DATA EQU    034H    ; For keep I2C Data
IO_DATA1 EQU    035H    ; For keep I2C 8 bit I/O Data
OUT_DATA        EQU     036H    ; For keep I2C 4 bit Output LED Data
IO_DATA2 EQU    037H

;DEFINE PORT
ONEWIRE BIT P2.7
BUSY BIT FLAG.0
ONEWIRE_DATA EQU 032H
TEMP EQU 038H
ENCO EQU 02DH
PREEN EQU 02EH
;-------------------------------------------------------------
; Main Program.
;-------------------------------------------------------------

                ORG 0000H
                MOV TMOD,#00100110B
                MOV TH1,#0E8H
                MOV TL1,#0E8H
                MOV TH0,#00H
                MOV TL0,#00H
                MOV SCON,#01010000B
                SETB P1.0
                SETB P1.1
                SETB P3.4
                SETB TR1        ; Reset Vector
                SETB TR0
                SETB ONEWIRE
                MOV PREEN,#00H

GATE_EN:  MOV R7,#200
GATE_EN_1: MOV R6,#2
GATE_EN_2: MOV R5,#0E6H
```

```
GATE_EN_3: NOP
           NOP
           DJNZ R5,GATE_EN_3
           DJNZ R6,GATE_EN_2
           DJNZ R7,GATE_EN_1
           CLR TR0


                ACALL DS1820_RST
           ACALL DS1820_PRES
           MOV ONEWIRE_DATA,#0CCH
           ACALL DS1820_WR
           MOV ONEWIRE_DATA,#044H
           ACALL DS1820_WR
           SETB BUSY

LOOP:      ACALL DS1820_RST
           ACALL DS1820_PRES
           JB BUSY,LOOP
           NOP
           NOP
           NOP
           NOP
           ACALL DS1820_RST
           ACALL DS1820_PRES
           MOV ONEWIRE_DATA,#0CCH
           ACALL DS1820_WR
           MOV ONEWIRE_DATA,#0BEH
           ACALL DS1820_WR
           ACALL DS1820_RD
           MOV TEMP,ONEWIRE_DATA
           ACALL DS1820_RST
           ACALL DS1820_PRES
           MOV A,TEMP
           CLR C
           RRC A
           JNC NOTONE
           SETB ACC.7
NOTONE:    MOV TEMP,A

                MOV     I2C_ADDR,#0C0H
                ACALL   I2C_SLAVE
                MOV     I2C_DATA,#02H
                ACALL   I2C_DATA_WR
                ACALL   I2C_START
                MOV     I2C_DATA,#0C1H
```

```
                ACALL   I2C_DATA_WR

                ACALL   I2C_DATA_RD
                MOV     IO_DATA1,I2C_DATA    ;OUTPUT
                ACALL   I2C_NACK_BIT

                MOV     I2C_ADDR,#0C0H
                ACALL   I2C_SLAVE
                MOV     I2C_DATA,#03H
                ACALL   I2C_DATA_WR
                ACALL   I2C_START
                MOV     I2C_DATA,#0C1H
                ACALL   I2C_DATA_WR

                ACALL   I2C_DATA_RD
                MOV     IO_DATA2,I2C_DATA    ;OUTPUT
                ACALL   I2C_NACK_BIT


                ACALL   I2C_STOP
                CPL PREEN.7
                MOV R1,#5
                MOV A,#128
                CJNE A,PREEN,LUUP
                MOV ENCO,TL0
                SETB ENCO.7
                LJMP FIVETIME
LUUP:           MOV ENCO,TL0
FIVETIME:MOV A,#250
                MOV SBUF,A
                JNB TI,$
                CLR TI
                MOV A,#230
                MOV SBUF,A
                JNB TI,$
                CLR TI
                MOV A,ENCO
                MOV SBUF,A
                JNB TI,$
                CLR TI
                MOV A,TEMP
                MOV SBUF,A
                JNB TI,$
                CLR TI
                MOV A,IO_DATA1
                MOV SBUF,A
```

```
JNB TI,$
CLR TI
MOV A,IO_DATA2
MOV SBUF,A
JNB TI,$
CLR TI
MOV A,#250
XRL A,#230
XRL A,ENCO
XRL A,TEMP
XRL A,IO_DATA1
XRL A,IO_DATA2
MOV SBUF,A
JNB TI,$
CLR TI
DJNZ R1,FIVETIME
MOV TH0,#00H
MOV TL0,#00H
SETB TR0
LJMP GATE_EN
```

```
;-------------------------------------------------------------------
; I2C Data Write
; I/P:          I2C_DATA
; Reserve:  R5
;-------------------------------------------------------------------
I2C_DATA_WR:    PUSH    ACC             ; Push ACC.
                SETB    I2C_ACK         ; Set ACK. bit
                MOV     A,I2C_DATA      ; Get Data
                MOV     R5,#008         ; Set loop 8 times
I2C_DATA_WR_1:  RLC     A               ; Rotate ACC. to Left with Carry
                MOV     SDA,C           ; Move Carry Flag to SDA
                ACALL   I2C_CLK         ; Pulse I2C Clock
                DJNZ    R5,I2C_DATA_WR_1; Do until 8 times
                SETB    SDA             ; Set SDA
                ACALL   I2C_DELAY       ; Delay
```

```
                SETB    SCL             ; Set SCL
                ACALL   I2C_DELAY       ; Delay
                JB      SDA,I2C_DATA_WR_2       ; Check Acknowledge from Slave
                CLR     I2C_ACK         ; Clear ACK. bit
I2C_DATA_WR_2:  CLR     SCL             ; Clear SCL
                POP     ACC             ; Pop ACC.
                RET                     ; Return

;-------------------------------------------------------------
; I2C Data Read
; O/P:          I2C_DATA
; Reserve:  R5
;-------------------------------------------------------------
I2C_DATA_RD:    PUSH    ACC             ; Push ACC.
                CLR     A               ; Clear ACC.
                MOV     R5,#008         ; Set loop 8 times
I2C_DATA_RD_1:  ACALL   I2C_DELAY       ; Delay
                SETB    SCL             ; Set SCL
                ACALL   I2C_DELAY       ; Delay
                MOV     C,SDA           ; Get SDA to Carry Flag
                RLC     A               ; Rotate ACC. to Left with Carry
                CLR     SCL             ; Clear SCL
                DJNZ    R5,I2C_DATA_RD_1; Do until 8 times
                MOV     I2C_DATA,A      ; Move Data to I2C_DATA
                POP     ACC             ; Pop ACC.
                RET                     ; Return

;-------------------------------------------------------------
; I2C Slave Connect
; I/P:          I2C_ADDR
; O/P Flag:  I2C_ACK
; Reserve:  R5
;-------------------------------------------------------------
I2C_SLAVE:      PUSH    ACC             ; Push ACC.
                SETB    I2C_ACK         ; Set ACK. bit
                MOV     A,I2C_ADDR      ; Get Slave Address
                ACALL   I2C_START       ; Send Start Condition
                MOV     R5,#008         ; Set loop 8 times
I2C_SLAVE_1:    RLC     A               ; Rotate ACC. to Left with Carry
                MOV     SDA,C           ; Move Carry Flag to SDA
                ACALL   I2C_CLK         ; Pulse I2C Clock
                DJNZ    R5,I2C_SLAVE_1  ; Do until 8 times

                SETB    SDA             ; Set SDA
                ACALL   I2C_DELAY       ; Delay
                SETB    SCL             ; Set SCL
```

```
                ACALL   I2C_DELAY           ; Delay
                JB      SDA,I2C_SLAVE_2 ; Check Acknowledge from Slave
                CLR     I2C_ACK             ; Clear ACK.
I2C_SLAVE_2:    CLR     SCL                 ; Clear SCL
                POP     ACC                 ; Pop ACC.
                RET                         ; Return

;-------------------------------------------------------------
; I2C Start Condition
;-------------------------------------------------------------
I2C_START:      JNB     SCL,I2C_START_1     ; Check current SCL set?
                CLR     SCL                 ; Clear SCL

I2C_START_1:    SETB    SDA                 ; Set SDA
                SETB    SCL                 ; Set SCL

                ACALL   I2C_DELAY           ; Delay
                CLR     SDA                 ; Clear SDA during SCL set
                ACALL   I2C_DELAY           ; Delay
                CLR     SCL                 ; Clear SCL
                RET                         ; Return

;-------------------------------------------------------------
; I2C Stop Condition
;-------------------------------------------------------------
I2C_STOP: JNB   SCL,I2C_STOP_1  ; Check current SCL set?
                CLR     SCL                 ; Clear SCL

I2C_STOP_1:     CLR     SDA                 ; Clear SDA
                ACALL   I2C_DELAY           ; Delay
                SETB    SCL                 ; Set SCL
                ACALL   I2C_DELAY           ; Delay
                SETB    SDA                 ; Set SDA during SCL set
                RET                         ; Return

;-------------------------------------------------------------
; I2C Clock
;-------------------------------------------------------------
I2C_CLK:  ACALL I2C_DELAY       ; Pulse SCL
                SETB    SCL                 ;
                ACALL   I2C_DELAY           ;
                CLR     SCL                 ;
                RET                         ; Return

;-------------------------------------------------------------
; I2C Acknowledge
```

```
;-------------------------------------------------------------
I2C_ACK_BIT:    CLR     SDA             ; Clear SDA
                ACALL   I2C_DELAY       ; Delay
                ACALL   I2C_CLK         ; Pulse I2C Clock
                SETB    SDA
                RET                     ; Return

;-------------------------------------------------------------
; I2C Not Acknowledge
;-------------------------------------------------------------
I2C_NACK_BIT:   SETB    SDA             ; Set SDA
                ACALL   I2C_DELAY       ; Delay
                ACALL   I2C_CLK         ; Pulse I2C Clock
                SETB    SCL
                RET                     ; Return

;-------------------------------------------------------------
; Dummy Delay time I2C_DELAY, 100m
;-------------------------------------------------------------
I2C_DELAY:      MOV     R6,#00CH  ; Each loop = 50 us
I2C_DELAY_1:    NOP
                NOP
                DJNZ    R6,I2C_DELAY_1
                RET

DELAY_100ms:    MOV     R7,#100         ; Do 100 times
DELAY_100ms_1:  MOV     R6,#0E6H  ; Each loop = 1 ms
DELAY_100ms_2:  NOP
                NOP
                DJNZ    R6,DELAY_100ms_2
                DJNZ    R7,DELAY_100ms_1
                RET
DS1820_RD: MOV R4,#8
           CLR A
DS1820_RD_LOOP: CLR ONEWIRE
                NOP
                NOP
                SETB ONEWIRE
                NOP
                NOP
                NOP
                NOP
                MOV C,ONEWIRE
                ACALL ONEWIRE_DELAY
                RRC A
                DJNZ R4,DS1820_RD_LOOP
```

```
                MOV ONEWIRE_DATA,A
                RET

DS1820_WR: MOV R4,#8
           MOV A,ONEWIRE_DATA
DS1820_WR_LOOP: RRC A
                JNC DS1820_WR_L
                CLR ONEWIRE
                NOP
                NOP
                NOP
                NOP
                SETB ONEWIRE
                ACALL ONEWIRE_DELAY
                AJMP DS1820_WR_NX
DS1820_WR_L:    CLR ONEWIRE
                ACALL ONEWIRE_DELAY
                SETB ONEWIRE
                NOP
                NOP
                NOP
                NOP
DS1820_WR_NX:   DJNZ R4,DS1820_WR_LOOP
                RET

DS1820_RST:     CLR ONEWIRE
                ACALL DELAY_1ms
                SETB ONEWIRE
                MOV R4,#8
                DJNZ R4,$
                RET

DS1820_PRES:  MOV R4,#8
DS1820_PRES_1: MOV R3,#0
DS1820_PRES_2: JNB ONEWIRE,DS1820_PRES_3
               DJNZ R3,DS1820_PRES_2
               DJNZ R4,DS1820_PRES_1
               RET
DS1820_PRES_3: JNB ONEWIRE,$
               MOV R4,#8
               DJNZ R4,$
               CLR BUSY
               RET

ONEWIRE_DELAY: MOV R6,#012H
ONEWIRE_DELAY_1: NOP
```

```
                NOP
                DJNZ R6,ONEWIRE_DELAY_1
                RET

DELAY_1ms:  MOV R6,#0E6H
DELAY_1ms_1:  NOP
                NOP
                DJNZ R6,DELAY_1ms_1
                RET
                END
; This is a Systronix generated Main file
;  Put your include statements in this file
```

โค้ดภาคส่งคัวรับข้อมูลจาก rtp315

```
FLAG EQU 020H
TEMP EQU 021H
DATA1 EQU 022H
DATA2 EQU 023H
ENDCO EQU 024H
SUFCO EQU 025H
        ORG 0000H
        MOV P1,#00H
        MOV P0,#00H
        MOV P2,#00H
        MOV TMOD,#20H
        MOV TH1,#0E8H
        MOV TL1,#0E8H
        MOV SCON,#01010000B
        SETB TR1
        MOV SUFCO,#00H

WAIT:   JNB RI,$
        MOV A,SBUF
        CLR RI

    CJNE A,#250,WAIT
        JNB RI,$
        MOV A,SBUF
        CLR RI

        CJNE A,#230,WAIT
        JNB RI,$
        MOV A,SBUF
        CLR RI
        MOV ENDCO,A
        JNB RI,$
        MOV A,SBUF
        CLR RI
        MOV TEMP,A
        JNB RI,$
        MOV A,SBUF
        CLR RI
        MOV DATA1,A
        JNB RI,$
        MOV A,SBUF
        CLR RI
        MOV DATA2,A
        JNB RI,$
```

```
        MOV A,SBUF
        CLR RI
        MOV FLAG,A

        MOV A,#250
        XRL A,#230
        XRL A,ENDCO
        XRL A,TEMP
        XRL A,DATA1
        XRL A,DATA2
        CJNE A,FLAG,WAIT
        LJMP SEND_DATA


SEND_DATA: MOV A,ENDCO
        XRL A,SUFCO
        JZ WAIT
        MOV A,ENDCO
        MOV SBUF,A
        JNB TI,$
        CLR TI
        MOV A,TEMP
        MOV SBUF,A
        JNB TI,$
        CLR TI
        MOV A,DATA1
        MOV SBUF,A
        JNB TI,$
        CLR TI
        MOV A,DATA2
        MOV SBUF,A
        JNB TI,$
        CLR TI
        MOV SUFCO,ENDCO
        LJMP WAIT
```