รถสำรวจควบคุมแบบไร้สายโดยคอมพิวเตอร์

Computerized wireless controlled explorer car

โดย

นายฐานันดร์ ยันตะกนก

นายอนรรฆพงษ์ ทองทัต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

รถสำรวจควบคุมแบบไร้สายโดยคอมพิวเตอร์

Computerized wireless controlled explorer car

โดย

นายฐานันดร์ ยันตะกนก 46012008

นายอนรรถพงษ์ ทองทัต 46012038

อาจารย์ที่ปรึกษา

รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **รถสำรวจควบคุมแบบไร้สายโดยคอมพิวเตอร์**

Computerized wireless controlled explorer car

ผู้จัดทำ

1. นายฐานันดร์ ยันตะกนก 46012008

2. นายอนรรถพงษ์ ทองทัต 46012038

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี)

รถสำรวจควบคุมแบบไร้สายโดยคอมพิวเตอร์

Computerized wireless controlled explorer car

โดย นาย ฐานันดร์ ยันตะกนก 46012008
นาย อนรรคพงษ์ ทองทัต 46012038

อาจารย์ที่ปรึกษา รศ.ดร. ยุทธพงษ์ รังสรรค์เสรี

บทคัดย่อ

โครงงานนี้ทำการออกแบบสร้างรถสำรวจควบคุมแบบไร้สายโดยคอมพิวเตอร์ควบคุมการทำงาน ซึ่งรถสำรวจได้ทำการติดตั้งกล้องวีดีทัศน์โดยการส่งสัญญาณด้วยคลื่น RF แบบไร้สายประมวลผลทางคอมพิวเตอร์

ABSTRACT

The purpose of this project is to design a computerized control car for a site exploration. The attached video camera signaling via RF (radio frequency) is also controlled by computer system.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลงได้ด้วยดีโดยความช่วยเหลือจากบุคคลหลายๆฝ่ายขอกราบขอบพระคุณ รศ.ดร.ยุทธพงษ์ รังสรรค์เสรี ที่ให้คำแนะนำตลอดระยะเวลาที่ทำโครงงาน ขอขอบคุณ นางสาวเอวิกา งามเถื่อน ที่ให้คำแนะนำในการทำรูปเล่มโปรเจค ขอขอบคุณอาจารย์ทุกท่านที่ให้คำสั่งสอนอันมีค่า และขอขอบคุณเพื่อนๆ ที่มีน้ำใจช่วยเหลือทำให้โครงงานชิ้นนี้สำเร็จลงได้ด้วยดี

สารบัญ

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

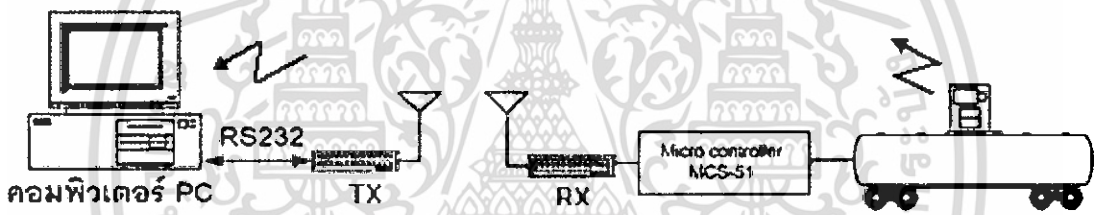
สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันหุ่นยนต์ได้เข้ามามีบทบาทในชีวิตประจำวันของมนุษย์มากขึ้นทุกวันโดยเทคโนโลยีมนุษย์คิดค้นและผลิตขึ้นไม่มีวันหยุดยั้งเมื่อมนุษย์ต้องการความสะดวกสบายมากขึ้นจึงเป็นที่มาของการพัฒนารูปแบบหุ่นยนต์ให้ใช้งานได้หลายๆด้าน เช่น ใช้เป็นเครื่องทุนแรง เครื่องอำนวยความสะดวก หรือแม้แต่ในด้านการสำรวจ

ในการสำรวจนั้นสถานที่เป็นพื้นที่อันตรายซึ่งมนุษย์ไม่อาจเข้าไปสำรวจหรือเก็บข้อมูลได้โดยตรง จึงจำเป็นอย่างยิ่งที่จะต้องมีอุปกรณ์หรือเครื่องมือที่ทำหน้าที่แทนมนุษย์ดังนั้นรถสำรวจจึงเข้ามามีบทบาทในด้านวิศวกรรม และทางการทหาร เช่น ในการสำรวจเมืองแร่ สำรวจพื้นที่ที่มีวัตถุระเบิดหรือพื้นที่ที่มีสารเคมีที่เป็นพิษต่อร่างกายมนุษย์ ฯลฯ โดยมีการควบคุม จากระยะไกล ซึ่งโครงงานนี้เป็นการสร้างรถสำรวจควบคุมโดยคอมพิวเตอร์แบบไร้สาย โดยมีกล้องวิดีโอติดอยู่ที่ตัวรถ และส่งข้อมูลภาพกลับมาเพื่อแสดงผลยังคอมพิวเตอร์



รูปที่ 1.1 ภาพรวมของโครงงาน

รูปที่ 1.1 แสดงให้เห็นถึงหลักการทำงานของรถสำรวจซึ่งสามารถแบ่งออกได้เป็น 2 ส่วน คือ ส่วนที่ทำหน้าที่ในการควบคุมโดยใช้คอมพิวเตอร์ และส่วนของตัวรถสำรวจ โดยส่วนที่ทำหน้าที่ในการควบคุมจะส่งสัญญาณควบคุมต่างๆ ออกไปยังรถสำรวจ ที่ตัวรถสำรวจจะมีไมโครคอนโทรลเลอร์ทำหน้าที่ในการตัดสินใจว่าสัญญาณควบคุมที่รับได้นั้นเป็นสัญญาณที่ต้องการให้รถสำรวจทำงานอย่างไร เช่น เดินหน้าหรือถอยหลัง เลี้ยวซ้ายหรือเลี้ยวขวา หรือควบคุมตำแหน่งของกล้อง และกล้องที่ติดอยู่กับตัวรถสำรวจก็จะส่งสัญญาณภาพกลับมายังส่วนที่ทำหน้าที่ในการควบคุมเพื่อแสดงผล

ส่วนของการส่งสัญญาณนั้นเราเลือกใช้กล้องและเครื่องส่งสัญญาณภาพที่เป็นแบบสำเร็จรูป โดยที่ตัวคอมพิวเตอร์จะมี CARD สำเร็จรูปในการรับสัญญาณ TV/VDO อยู่เพื่อทำหน้าที่ในการรับสัญญาณที่ส่งมาจากกล้อง

บทที่ 2
ทฤษฎีและหลักการ

2.1 ไมโครคอนโทรลเลอร์ MCS-51

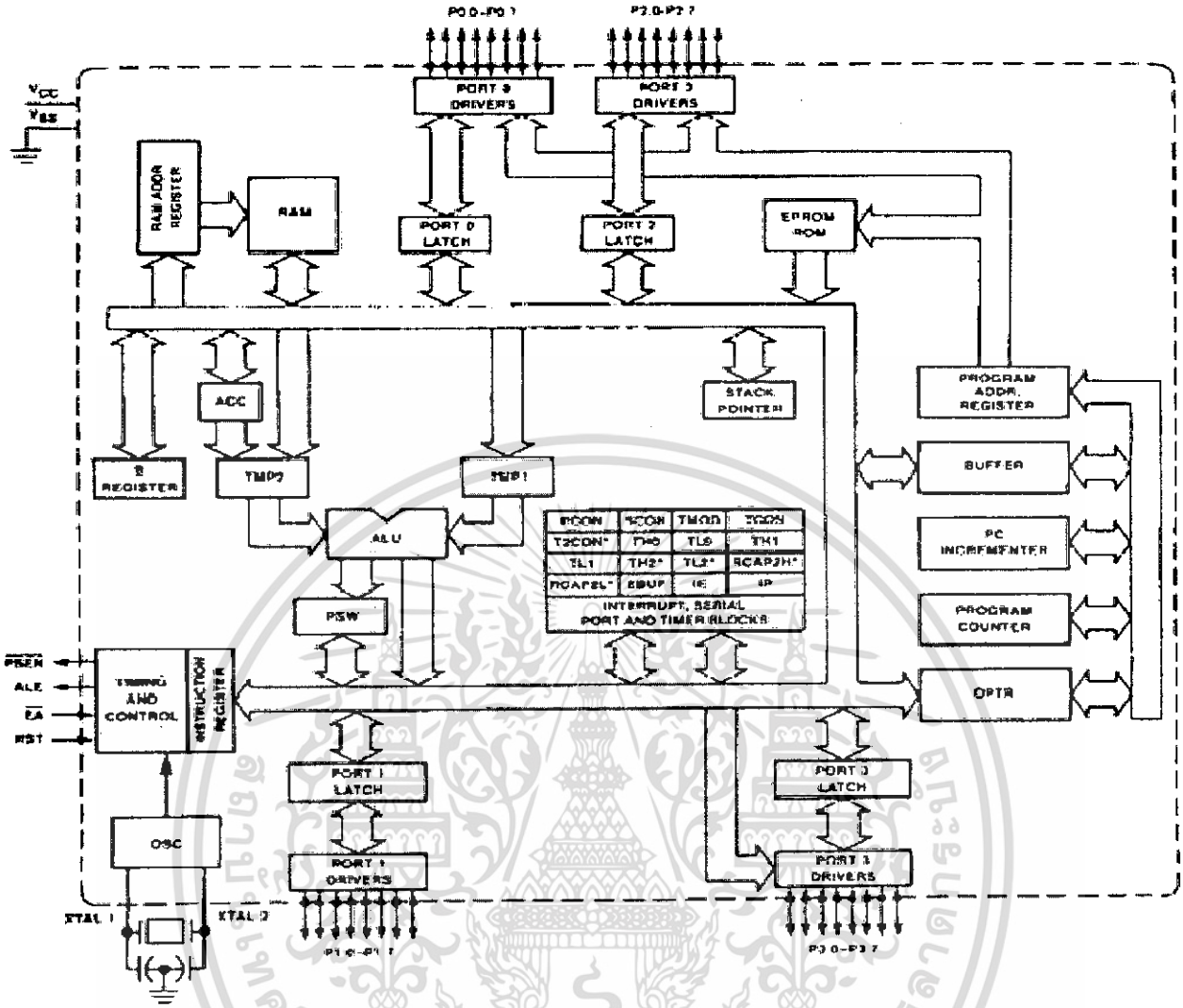
ไมโครคอนโทรลเลอร์หรืออาจเรียกว่าชิงเกิลชิพไมโครคอนโทรลเลอร์ (Single Chip - Microcontroller) เป็นอุปกรณ์ที่นำเอาไมโครโพรเซสเซอร์มารวมกับหน่วยความจำและอุปกรณ์ไอโอต่างๆเอาไว้ในตัวเดียวกัน สามารถทำงานได้ทันทีเมื่อป้อนไฟเลี้ยงและสัญญาณนาฬิกาให้ตัวไมโครคอนโทรลเลอร์ จึงทำให้การออกแบบวงจรง่ายขึ้นและมีขนาดเล็กลงเป็นอย่างมาก ในที่นี้จะขอยกตัวอย่างไมโครคอนโทรลเลอร์ตระกูล MCS-51 ซึ่งมีบล็อกไดอะแกรมแสดงในรูปที่ 2.1

การพัฒนาขีดความสามารถของไมโครคอนโทรลเลอร์ได้เป็นมาอย่างต่อเนื่อง ซึ่งมีการพัฒนาทั้งในด้านฮาร์ดแวร์(Hardware) และซอฟต์แวร์(Software) ในการพัฒนาด้านฮาร์ดแวร์นั้นสิ่งที่เห็นได้ชัดเจนคือการพัฒนาเกี่ยวกับหน่วยความจำ ซึ่งได้เริ่มมีการพัฒนามาจาก PROM EPROM EEPROM และในปัจจุบัน ได้พัฒนามาจนถึงหน่วยความจำที่เรียกว่า FLASH MEMORY

ในส่วนของซอฟต์แวร์นั้นได้พัฒนาการเขียนโปรแกรมมาจาก การเขียนโปรแกรมในระดับบิต (เลขฐาน2) , ระดับไบต์ (เลขฐาน 16), แอสแซมบลีและภาษาในระดับสูง เช่น ภาษาเบสิก และภาษาซี เป็นต้น

2.1.1 โครงสร้างภายในของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ใช้เทคโนโลยีในการผลิตแบบ NMOS และ CMOS ซึ่งภายในได้รวมวงจรต่างๆไว้อย่างครบถ้วนพร้อมที่จะทำงานได้เมื่อจ่ายไฟเลี้ยงและสัญญาณนาฬิกา ไมโครคอนโทรลเลอร์ตระกูล MSC-51 ได้ถูกผลิตออกมามากมายหลายเบอร์โดยบริษัทต่างๆ เช่น บริษัท Atmel, Philips, Dallas, Infinion และบริษัทอื่นๆซึ่งไม่ว่าจะเป็นเบอร์อะไรก็ตาม ถ้าเป็นไมโครคอนโทรลเลอร์ตระกูล MSC-51 และจะมีโครงสร้างต่างๆที่คล้ายกันจะแตกต่างกันออกไปในส่วนของความสามารถพิเศษของแต่ละเบอร์ ยกตัวอย่างเช่น เบอร์ AT89C51 มีไทเมอร์ 2 ตัว ในขณะที่เบอร์ AT89C52 มีไทเมอร์ 3 ตัว เป็นต้น



รูปที่ 2.1 บล็อกไดอะแกรมของไมโครคอนโทรลเลอร์ตระกูล MCS-51

2.1.2 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ MCS-51

เนื่องจากคุณสมบัติของไมโครคอนโทรลเลอร์แต่ละเบอร์นั้นจะมีแตกต่างกันในรายละเอียดปลีกย่อยดังได้กล่าวแล้วข้างต้น ดังนั้นในที่นี้จะขออ้างอิงถึงเบอร์ AT89C52 ของบริษัท Atmel ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ซึ่งจะใช้อ้างอิงตลอด และสามารถใช้อ้างอิงร่วมกับชิพเบอร์อื่นๆในตระกูลเดียวกันได้ เช่น AT89C51, AT89S51/52/53 และ AT89S8252 เป็นต้น สำหรับในการทดลองจะเน้นไปที่เบอร์ AT89S5252 ซึ่งมีคุณสมบัติดังนี้

* มีหน่วยความจำโปรมแกรมแบบแฟรช (Flash Memory) ขนาด 8 กิโลไบต์
  - โปรแกรมผ่านพอร์ตอนุกรมมาตรฐาน SPI (SPI Serial Interface)
  - สามารถโปรแกรมและลบซ้ำได้นับ 1,000 ครั้ง
* มีหน่วยความจำแบบ EEPROM ขนาด 2 กิโลไบต์
  - สามารถโปรแกรมและลบซ้ำได้นับ 100,000 ครั้ง

* ใช้แหล่งจ่ายไฟฟ้ากระแสตรงขนาด 5 โวลท์ (ทำงานได้ในช่วง 4-6 โวลท์)
* ทำงานได้ด้วยสัญญาณนาฬิกาตั้งแต่ 0-24 MHz
* สามารถป้องกันหน่วยความจำได้ 3 ระดับ
* มีหน่วยความจำข้อมูล(RAM) ขนาด 256 ไบต์
* มีพอร์ต 32 พอร์ต อิสระสามารถเข้าถึงในระดับบิตได้
* มีไทเมอร์/เคาเตอร์ขนาด 16 bit ทั้งหมด 3 ตัว
* รองรับการอินเตอร์รัปต์ได้ 8 แหล่ง
* สามารถสื่อสารข้อมูลแบบอนุกรมได้ด้วย UART Channel
* มีโหมดการทำงานแบบ Low Power Idle และPower Down สำหรับการประหยัดพลังงาน
* มี Watchdog Timer เพื่อเพิ่มเสถียรภาพการทำงานของระบบ

2.1.3 การจัดการของไมโครคอนโทรลเลอร์ตระกูล MCS-51(เบอร์ 89C2051)

VCC ต่อไฟเลี้ยง (supply voltage)

GND ต่อกราวด์ (ground)

Port1 (P1.0-P1.7) เป็นพอร์ตแบบสองทิศทางขนาด 8 บิต มีการต่อความต้านทานพูลอัพ(pull-up resistor) ไว้ภายในทำหน้าที่เป็นพอร์ตอินพุตเอาท์พุตทั่วไป นอกจากนี้ยังใช้งานเป็นขาอินพุตเอาต์พุตของไทเมอร์ 2

Port3 เป็นพอร์ตแบบสองทิศทางขนาด 8 บิต มีการต่อความต้านทานพูลอัพ(pull-up resistor) ไว้ภายในทำหน้าที่เป็นพอร์ตอินพุตเอาท์พุตทั่วไป นอกจากนี้ยังใช้งานเป็นขาสัญญาณควบคุมการติดต่อกับหน่วยความจำการอินเตอร์รัปต์ และอื่นๆ

RST เป็นขาอินพุตที่ใช้รับสัญญาณสำหรับรีเซตซีพียู ซีพียูจะถูกรีเซตเมื่อขานี้เป็นลอจิก "1" นาน 2 แมชชินไซเคิล หรือ 24 ไซเคิลของสัญญาณนาฬิกา

ALE/PROG ทำหน้าที่เป็นขาเอาต์พุตเมื่อซีพียูต้องการติดต่อกับหน่วยความจำภายนอก คือจะทำการส่งสัญญาณพัลซ์ออกมาที่ขานี้เพื่อทำการแลตแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก และขานี้จะเป็นอินพุตเมื่ออยู่ในระหว่างโปรแกรมแฟรช

PSEN เป็นขาเอาต์พุตใช้ในการติดต่อกับหน่วยความจำโปรแกรมภายนอก คือเมื่อซีพียูทำการประมวลผลกับหน่วยความจำโปรแกรมภายนอกขานี้จะแอคทีฟสองครั้งในแต่ละแมชชินไซเคิล

EA/VPP เป็นขาอินพุตและต้องการลอจิก "0" เพื่อยอมให้ซีพียูสามารถเข้าถึงหน่วยความจำโปรแกรมภายนอกได้ ซึ่งอยู่ที่ตำแหน่ง 0000H ถึง FFFFH นอกจากนี้แล้วขานี้ยังใช้รับไฟ 12 โวลท์เพื่อใช้ในระหว่างที่ทำการโปรแกรมแฟรช

XTAL1 เป็นขาอินพุตของวงจรออสซิลเลเตอร์แอมปลิไฟเออร์ และยังเป็นอินพุตของวงจรกำเนิดสัญญาณนาฬิกาภายใน

XTAL2 เป็นขาเอาต์พุตของวงจรออสซิลเลเลอร์แอมปลิไฟเออร์

2.2 การใช้งานพอร์ตต่างๆ

พอร์ตเป็นเส้นทางในการรับและส่งสัญญาณระหว่างอุ ปกรณ์ต่างๆจากภายนอก ไม่ว่าจะเป็นอุปกรณ์เอาต์พุต(output)หรืออุปกรณ์อินพุต(input) โดยทั่วไปพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 มีทั้งหมด 4 พอร์ต คือ พอร์ต0 , พอร์ต1, พอร์ต2 และพอร์ต 3 (ไมโครคอนโทรลเลอร์บางเบอร์อาจมีพอร์ตมากหรือน้อยกว่า 4 พอร์ต) การใช้งานพอร์ตต่างๆ ต้องเลือกใช้งานให้เหมาะสมกับงานและความสามารถของพอร์ตนั้น

2.2.1 พอร์ต 0 (PO.0-PO.7)

พอร์ต 0 เป็นพอร์ตแบบสองทิศทางขนาด 8 บิต พอร์ต 0 สามารถทำงานได้ 2 หน้าที่คือเป็นพอร์ตอินพุตเอาต์พุตสำหรับใช้งานทั่วไปแต่ในกรณีที่ใช้งานเป็นพอร์ตอินพุต เอาต์พุตพอร์ต0 จะไม่สามารถจ่ายแรงดันและกระแสได้ แต่สามารถรับกระแสได้ตามปกติดังนั้นพอร์ตนี้ไม่สามารถใช้ขับอุปกรณ์ที่รับกระแสได้ เช่นหากนำ LEDไปต่อโดยเอาขา A ต่อที่ขาพอร์ต และขาK ต่อกราวด์ LED จะไม่สว่างเพราะพอร์ตนี้ไม่สามารถจ่ายไฟให้LEDได้ แต่อย่างไรก็ตามสามารถดัดแปลงให้พอร์ต0 จ่ายกระแสได้ โดยการต่อตัวต้านทานพูลอัพ(pull-up resistor) ซึ่งกระแสที่จ่ายออกมานั้นขึ้นอยู่กับตัวต้านทานพูลอัพ

ในกรณีที่มีการต่อหน่วยความจำภายนอก พอร์ต0 จะทำหน้าที่เป็นพอร์ตซึ่งกำหนดตำแหน่ง(address)ไบต์ต่ำ และเป็นพอร์ตสำหรับรับส่งข้อมูลกับหน่วยความจำภายนอก ซึ่งจะเรียกการทำงานของพอร์ตลักษณะนี้ว่า "multiplexed low-order address/data bus"

วงจรภายในของพอร์ต 0 มีอยู่ 2 ส่วน ส่วนแรกคือ วงจรมัลติเพล็ก (Multiplex) ทำหน้าที่เลือกการทำงานของพอร์ตว่าจะให้เป็นพอร์ตอินพุตเอาพุตทั่วไปหรือจะให้เป็นพอร์ตเพื่อใช้ติดต่อกันกับหน่วยความจำภายนอก ส่วนที่สองคือวงจร แลตซ์(Latch) ซึ่งเป็น ดี-ฟลิบฟลอบ(D-Flip Flop) ทำหน้าที่คงสภาวะข้อมูล

2.2.2 พอร์ต1 (P1.0-P1.7)

พอร์ต 1 เป็นพอร์ตแบบสองทิศทางขนาด 8 บิต มีการต่อความต้านทานชนิดพูลอัพ(pull-up resister)ไว้ภายใน ทำหน้าที่เป็นพอร์ตอินพุตเอาท์พุตทั่วไป นอกจากนี้ยังใช้งานเป็นขาอินพุต/เอาต์พุตของไทเมอร์2 ซึ่งรายละเอียดแสดงในตารางที่ 2.1ในรูปที่จะเห็นได้ชัดว่าพอร์ต1 จะไม่มีส่วนของวงจรมัลติเพล็ก(Multiplex) เพราะพอร์ต1 ไม่ได้ใช้ในการติดต่อกับหน่วยความจำภายนอก แต่มีวงจรแลตซ์(Latch) ซึ่งเป็น ดี-ฟลิบฟลอบ(D-Flip Flop) ทำหน้าที่คงสภาวะข้อมูล

ตารางที่ 2.1 หน้าที่พิเศษของพอร์ต 1

Port Pin	Alternate Functions
P1.0	T2 (external count input Timer/Counter), clock-out
P2.0	T2EX (Timer/Counter 2 capture/reload trigger and direction control)

2.2.3 พอร์ต 3 (P3.0-P3.7)

พอร์ต 3 เป็นพอร์ตแบบสองทิศทางขนาด 8 บิต มีการต่อความต้านทานพูลอัพ (pull-up resister) ไว้ภายใน ทำหน้าที่เป็นพอร์ตอินพุตเอาท์พุตทั่วไป นอกจากนี้พอร์ต3 ยังทำหน้าที่พิเศษต่างๆ เช่นการสื่อสารข้อมูลแบบอนุกรม , การอินเตอร์รัปต์จากภายนอก , รับ/ส่งสัญญาณสำหรับไทเมอร์1 และสัญญาณควบคุมการอ่านและเขียนหน่วยความจำ ซึ่งรายละเอียดต่างๆ ของพอร์ต 3 แสดงในตารางที่ 2.2

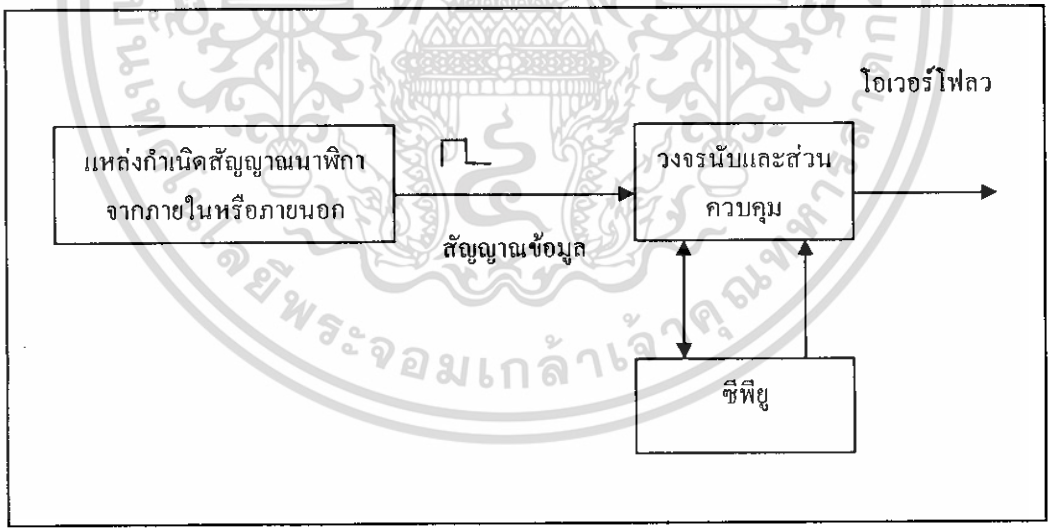
ตารางที่ 2.2 หน้าที่พิเศษของพอร์ต 3

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

2.3 ไทเมอร์ / เคาเตอร์ (Timer/Counter)

ไทเมอร์/เคาร์เตอร์ เป็นวงจรภายในที่ช่วยเพิ่มความสามารถให้ไมโครคอนโทรเลอร์ เนื่องจาก ไทเมอร์/ เคาเตอร์จะมีลักษณะการทำงานเป็นตัวนับเวลาหรือนับสัญญาณนาฬิกา ซึ่งสามารถทำงานเกี่ยวกับการสร้างฐานเวลา, การสร้างสัญญาณพัลซ์, การเปรียบเทียบค่าของเวลาหรือค่าการนับสัญญาณพัลซ์ที่รับเข้ามาจากภายนอก และรวมไปถึงการควบคุมอัตราการรับส่งข้อมูลของพอร์ตอนุกรมด้วย ไทเมอร์/เคาเตอร์เป็นวงจรที่สามารถทำงานได้เองโดยเป็นอิสระจากซีพียู แต่ซีพียูสามารถควบคุมการทำงาน, การอ่านและเขียนข้อมูลต่าง ๆ ได้

ไมโครคอนโทรลเลอร์แต่ละเบอร์อาจจะมีไทเมอร์ / เคาเตอร์จำนวนต่างกัน หรือมีความสามารถที่แตกต่างกัน เช่น เบอร์AT89C51 มีไทเมอร์ / เคาเตอร์จำนวน 2 ตัว(T0 และ T1) ในขณะที่เบอร์ AT89C52 หรือเบอร์ AT89S8252 มีไทเมอร์ / เคาเตอร์จำนวน 3 ตัว(T1 และ T2)เป็นต้น ไทเมอร์ / เคาเตอร์แต่ละตัวอาจจะมีความสามารถที่แตกต่างกันในรายละเอียดทางฮาร์ดแวร์และการทำงาน แต่อย่างไรก็ตามจะมีหลักการทำงานที่เหมือนกันคือ ถูกควบคุมจากซีพียูและใช้สัญญาณนาฬิกามาเป็นสัญญาณอินพุต ในกรณีใช้งานเป็นไทเมอร์หรือรับสัญญาณอินพุตจากภายนอกในกรณีใช้งานเป็นเคาเตอร์นอกจากนี้แล้ว ไทเมอร์/เคาเตอร์ทุกตัวยังมีการทำงานที่เป็นอิสระต่อกันอีกด้วย ในที่นี้จะขอกล่าวเพียงสองตัวเท่านั้น คือ ไทเมอร์/เคาเตอร์ 0 และไทเมอร์/เคาเตอร์ 1



รูปที่ 2.2 โครงสร้างโดยรวมของไทเมอร์/เคาเตอร์

2.3.1 การทำงานในโหมดไทเมอร์ (Timer Mode) ของไทเมอร์/เคาเตอร์ 0 และ 1

เมื่อกำหนดให้มีการทำงานเป็นโหมดไทเมอร์หรือตัวตั้งเวลา วงจรนับหรือเคาเตอร์จะทำการนับขึ้นโดยใช้สัญญาณนาฬิกาภายในซึ่งสัญญาณนาฬิกาตัวนี้จะเกิดขึ้นทุกๆ 1 แมชชีนไซเคิล (Machine cycle) นั่นหมายความว่าจะเกิดการนับขึ้น 1 ค่าในแต่ละแมชชีนไซเคิลนั่นเอง โดนในไมโครคอนโทรลเลอร์โดยทั่วๆไป 1 แมชชีนไซเคิลจะใช้เวลา 12 คาบของสัญญาณนาฬิกาที่ป้อนให้กับตัวไมโครคอนโทรลเลอร์ นั่นหมายความว่าจะเกิดการนับขึ้นทุกๆ 1/12 ของความถี่สัญญาณนาฬิกา(บางเบอร์อาจใช้แค่ 4 คาบ เช่น เบอร์ DS80C320 หรือ ใช้ 6 คาบ เช่นเบอร์ P89V51RD2 เป็นต้น)

2.3.2 การทำงานในโหมดเคาเตอร์ (Counter Mode) ของไทเมอร์/เคาเตอร์ 0 และ 1

เมื่อกำหนดให้มีการทำงานในโหมดเคาเตอร์หรือตัวนับ วงจรนับหรือเคาเตอร์ภายในจะทำการนับขึ้นโดยใช้สัญญาณขอบขาลง (เปลี่ยน 1 เป็น 0) จากภายนอกที่เข้ามาทางขา T0(P3.4),T1(P3.5) กระบวนการอ่านสัญญาณอินพุตเหล่านี้จะใช้เวลาทั้งสิ้น 2 แมชชีนไซเคิล นั่นหมายความว่าอัตราการนับจะมีค่าเท่ากับ 1/24 (กรณี 1 แมชชีนไซเคิลใช้เวลา 12 คาบ) ของความถี่สัญญาณนาฬิกา ดังนั้นสามารถใช้การทำงานในโหมดเคาเตอร์เพื่อนับสัญญาณนาฬิกาได้สูงสุดเท่ากับ ความถี่สัญญาณนาฬิกา/24 (กรณี 1 แมชชีนไซเคิลใช้เวลา 12 คาบ) เช่นใช้สัญญาณนาฬิกา 12 MHz จะสามารถรับสัญญาณอินพุตที่มีความถี่สูงได้ 12 MHz/24 = 500 kHz (กรณี 1 แมชชีนไซเคิลใช้เวลา 12 คาบ)

2.3.3 รีจิสเตอร์ของไทเมอร์/ เคาเตอร์ 0 และ 1 (Timer / Counter0,1 Register)

การกำหนดให้ไทเมอร์ / เคาเตอร์ ทำงานตามความต้องการนั้นจะต้องมีการกำหนดค่าต่างๆ ให้กับรีจิสเตอร์ที่เกี่ยวข้องกับการทำงานอย่างเหมาะสม ซึ่งรีจิสเตอร์เหล่านี้เป็นรีจิสเตอร์ฟังก์ชันพิเศษ (SFR) ซึ่งมีรายละเอียดดังนี้

TL0 เป็นรีจิสเตอร์ไบต์ต่ำของไทเมอร์ 0 มีแอดเดรสอยู่ที่ 8AH

TH0 เป็นรีจิสเตอร์ไบต์สูงของไทเมอร์ 0 มีแอดเดรสอยู่ที่ 8BH

TL1 เป็นรีจิสเตอร์ไบต์ต่ำของไทเมอร์ 1 มีแอดเดรสอยู่ที่ 8CH

TH1 เป็นรีจิสเตอร์ไบต์สูงของไทเมอร์ 1 มีแอดเดรสอยู่ที่ 8DH

รีจิสเตอร์ของไทเมอร์ทั้ง 4 ตัวนี้เป็นรีจิสเตอร์ขนาด 8 บิต ดังนั้นเมื่อนำ TL0 มารวมกับ TH0 จะทำให้ได้รีจิสเตอร์ของไทเมอร์ 0 ขนาด 16 บิต ในทำนองเดียวกันเมื่อนำ TL1 มารวมกับ TH1 จะทำให้ได้รีจิสเตอร์ของไทเมอร์ 1 ขนาด 16 บิต ซึ่งสามารถเก็บค่าได้สูงสุดถึง 65,536 ค่า (0-65,535 หรือ 0000H ถึง FFFFH)

2.3.4 รีจิสเตอร์ควบคุมการทำงานของไทเมอร์/เคาเตอร์ 0 และ1

TCON เป็นรีจิสเตอร์ที่ทำงานของไทเมอร์/เคาเตอร์ 0 และ1 มีขนาด 8 บิตและมีแอดเดรสอยู่ที่ 88H และสามารถเข้าถึงในระดับบิตได้ ในแต่ละบิตมีหน้าที่แตกต่างกันออกไปดังนี้

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

รูปที่ 2.3 บิตต่างๆในรีจิสเตอร์ TCON

***TF 1** (Timer1 overflow flag) จะเกิดการเซตด้วยกระบวนการฮาร์ดแวร์ คือเมื่อการนับเพิ่มขึ้นจนเกิดการโอเวอร์โฟลว์ (เปลี่ยนจาก FFH ไปเป็น 00H ในกรณีที่เป็น 8บิต และเปลี่ยนจาก FFFFH ไปเป็น 0000H กรณีที่เป็น 16บิต) บิตนี้จะถูกเคลียร์โดยกระบวนการทางฮาร์ดแวร์เช่นกันโดยจะเคลียร์เมื่อซีพียูกระโดดไปทำงานในโปรแกรมบริการอินเตอร์รัปต์ของไทเมอร์ 1 โอเวอร์โฟลว์

***TR 1** (Timer 1 run control bit) ใช้ควบคุมการเปิด/ปิด (Enable/Disable) การทำงานของไทเมอร์1 การเซตและการเคลียร์สามารถทำได้โดยกระบวนการทางซอฟแวร์

"0" ปิดการทำงานของไทเมอร์ 0 (disable)

"1" เปิดการทำงานของไทเมอร์ 0 (enable)

***TF 0** (Timer 0 overflow flag) จะเกิดการเซตด้วยกระบวนการทางฮาร์ดแวร์ คือเมื่อการนับเพิ่มขึ้นจนถึงโอเวอร์โฟลว์ (เปลี่ยนจาก FFH ไปเป็น 00H ในกรณีที่เป็นแบบ 8 บิต และเปลี่ยนจาก FFFFH ไปเป็น 0000H กรณีที่เป็น 16บิต) บิตนี้จะถูกเคลียร์โดยกระบวนการทางฮาร์ดแวร์เช่นกันโดยจะเคลียร์เมื่อ ซีพียูกระโดดไปทำงานในโปรแกรมบริการการอินเตอร์รัปต์ของไทเมอร์ 0โอเวอร์โฟลว์

***TR 0** (Timer 0 run control bit) ใช้ควบคุมการเปิด/ปิด (Enable/Disable) การทำงานของไทเมอร์ 1การเซตและการเคลียร์สามารถทำได้โดยกระบวนการทางซอฟแวร์

"0" ปิดการทำงานของไทเมอร์ 0 (disable)

"1" เปิดการทำงานของไทเมอร์ 0 (enable)

***IE 1** (External Interrupt 1 edge flag) ใช้ในขบวนการอินเตอร์รัปต์ คือจะเกิดการเซตโดยกระบวนการทางฮาร์ดแวร์ นั่นคือเมื่อตรวจพบว่ามีสัญญานอินเตอร์รัปต์เข้ามาที่ขาอินพุตของอินเตอร์รัปต์ 1 (INT1 : P3.3) บิตนี้จะเซต และจะเคลียร์เมื่อ ซีพียูจะกระโดดไปทำงานในโปรแกรมบริการอินเตอร์รัปต์ของ external interrupt 1

***IT1** (interrupt 1 type control bit) ใช้ในกระบวนการอินเตอร์รัปต์ของเอ็กเทอร์นอลอินเตอร์รัปต์ 1 โดยเป็นตัวกำหนดว่าจะให้อินเตอร์รัปต์เกิดขึ้นเมื่อตรวจพบสัญญาณที่เข้ามาทางขาอินพุตของ

อินเตอร์รัปต์ 1 (INT1 : P3.3) เป็นการตรวจสอบสัญญานขอบขาลงหรือระดับลอจิก" 0 " การเซตและเคลียร์บิตนี้สามารถทำได้โดยกระบวนการทางซอฟแวร์

***IE 0** (External Interrupt 0 edge flag) **ใช้ในกระบวนการอินเตอร์รัปต์ คือจะเกิดการเซตโดย** กระบวนการทางฮาร์ดแวร์ นั่นคือเมื่อตรวจพบว่ามีสัญญานอินเตอร์รัปต์เข้ามาที่ขาอินพุตของอินเตอร์รัปต์ (INT1 : P3.2)บิตนี้จะเซต และจะเคลียร์เมื่อ ซีพียูจะกระโดดไปทำงานในโปรแกรมบริการอินเตอร์รัปต์ ของ external interrupt 0

***IT0** (interrupt 0 type control bit) **ใช้ในกระบวนการอินเตอร์รัปต์ของเอกเทอร์นอลอินเตอร์** รัปต์ 0โดยเป็นตัวกำหนดว่าจะให้อินเตอร์รัปต์เกิดขึ้นเมื่อตรวจพบสัญญานที่เข้ามาทางขาอินพุตของ อินเตอร์รัปต์ 0(INT1 : P3.2 เป็นการตรวจสอบสัญญานขอบขาลงหรือระดับลอจิก" 0 " การเซตและเคลียร์บิตนี้สามารถทำได้โดยกระบวนการทางซอฟแวร์

"0" จะเกิดการอินเตอร์รัปต์เมื่อตรวจพบสัญญานขอบขาลง (falling edge)

"1" จะเกิดการอินเตอร์รัปต์เมื่อตรวจพบสัญญาณเป็นลอจิก" 0" (low Level triggered)

2.3.5 รีจิสเตอร์เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1

TMOD รีจิสเตอร์ที่ทำงานของไทเมอร์/เคาเตอร์ 0 และ1 มีขนาด 8 บิตและมีแอดเดรสอยู่ที่ 89H ไม่สามารถเข้าถึงในระดับบิตได้ รีจิสเตอร์ตัวนี้จะแยกออกเป็นสองส่วน คือ 4 บิตบนจะใช้เลือกโหมดของไทเมอร์/เคาเตอร์1 และ 4 บิตล่างจะใช้เลือกโหมดของไทเมอร์/เคาเตอร์ 0 โดยในแต่ละบิตจะมีหน้าที่แตกต่างกันออกไปดังนี้

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	
Gate	C/T'	M1	M0	GATE	C/T'	M1	M0

รูปที่ 2.4 บิตต่างๆในรีจิสเตอร์ TMOD

***GATE** ทำหน้าที่เป็นตัวเลือกรูปแบบการทำงานของไทเมอร์/เคาเตอร์ ว่าจะให้มีการควบคุมการทำงานแบบซอฟแวร์หรือฮาร์ดแวร์ดังนี้

"0" เลือกรูปแบบการควบคุมการทำงานเป็นแบบซอฟแวร์ คือ ไทเมอร์/เคาเตอร์ x จะทำงานเมื่อ TRx ในรีจิสเตอร์ TCON เป็น "1"

"1" เลือกรูปแบบการควบคุมการทำงานเป็นแบบฮาร์ดแวร์ คือ ไทเมอร์/เคาเตอร์ x จะทำงานเมื่อ TRx ในรีจิสเตอร์ TCON เป็น "1" และที่ขาอินพุตของอินเตอร์รัปต์ INTx เป็นสถานะลอจิก "1"

***M1 , M0** (Mode Selector Bit) เป็นบิตที่ใช้เลือกโหมดการทำงานของไทเมอร์ซึ่งมีดังนี้

"00" เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์เป็นแบบ 13 บิต

"01" เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์เป็นแบบ 16 บิต

"10" เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์เป็นแบบ 8 บิต รีโหลดค่าอัตโนมัติ

"11"เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์0 (ไทเมอร์/เคาเตอร์ 1ไม่ทำงาน) ให้ทำงานแบบแยกส่วน โดยแยกออกเป็นไทเมอร์/เคาเตอร์สองตัว โดยตัวแรกจะใช้รีจิสเตอร์ TL0 จะถูกควบคุมการทำงานโดย TR0 ส่วนตัวที่สองจะใช้รีจิสเตอร์ TH0 จะถูกควบคุมการทำงานโดย TR1 (TR0 และTR1 อยู่ในรีจิสเตอร์ TCON)

2.4 กระบวนการอินเตอร์รัปต์

การอินเตอร์รัปต์คือการขัดจังหวะการทำงานของ CPU เพื่อ CPU หยุดการประมวลผลในปัจจุบันเอาไว้ชั่วคราวก่อน จากนั้นจะกระโดดไปทำงานในฟังก์ชันก์หรือชุดคำสั่งของการอินเตอร์รัปต์ฟังก์ชันหรือชุดคำสั่งที่ CPU กระโดดไปประมวลผลเมื่อเกิดการอินเตอร์รัปต์ จะเรียกว่าโปรแกรมสำหรับบริการอินเตอร์รัปต์ (Interrupt Service Routine : ISR) เมื่อ CPU ประมวลผลชุดคำสั่งในโปรแกรมการสำหรับบริการอินเตอร์รัปต์เสร็จสิ้น CPU จะกลับไปประมวลผลโปรแกรมที่ได้หยุดไว้ก่อนหน้าที่จะมีการอินเตอร์รัปต์ การเกิดอินเตอร์รัปต์ของไมโครคอลโทรลเลอร์ตระกูล MCS-51 ที่นิยมใช้กันทั่วไป (40 ขา) จะเกิดขึ้นได้จาก 6 แหล่งกำเนิดสัญญาณอินเตอร์รัปต์ แต่มีเวกเตอร์ (Vector) ของการอินเตอร์รัปต์เพียง 5 เวกเตอร์ เนื่องจากการอินเตอร์ที่เกิดจากพอร์ตอนุกรม (Serial port) ทั้งการรับข้อมูลและการส่งข้อมูลจะใช้เวกเตอร์ร่วมกัน การอินเตอร์รัปต์เกิดขึ้นได้ 2 ลักษณะ คือการอินเตอร์รัปต์จากภายนอกและการอินเตอร์รัปต์จากภายใน รายละเอียดของการอินเตอร์รัปต์ต่างๆ มีดังต่อไปนี้

2.4.1 รีจิสเตอร์ที่เกี่ยวข้องกับกระบวนการอินเตอร์รัปต์

กระบวนการอินเตอร์รัปต์มีรีจิสเตอร์ที่คอยควบคุมการทำงานอยู่ทั้งหมด 4 ตัว แต่ละตัวมีหน้าที่การทำงานที่แตกต่างกันออกไป ดังนี้

(ก) รีจีสเตอร์ควบคุมการปิด/เปิดการอินเตอร์รัปต์ (Interrupt Enable register ; IE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	
EA	-	ET2	ES	ET1	EX1	ET0	EX0

รูปที่ 2.4 (ก) บิตต่างๆของรีจิสเตอร์ IE

รีจิสเตอร์ IE เป็นรีจิสเตอร์ฟังก์ชันพิเศษมีขนาด 8 บิตอยู่ที่แอดเดรส 0×A8 สามารถเข้าถึงในระดับบิตได้ ใช้ในการควบคุมการปิด/เปิด หรือการตอบสนองของการอินเตอร์รัปต์ต่างๆรายละเอียดของแต่ละบิตมีดังต่อไปนี้

EA (Global enable/disable interrupt) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์ทั้งหมด บิตนี้เป็นบิตที่มีลำดับความสำคัญสูงสุด คือการอินเตอร์รัปต์ต่างๆจะไม่สามารถเกิดขึ้นได้เลย ถ้ามีการปิดการอินเตอร์รัปต์ที่บิตนี้เอาไว้ บิต EA สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์เพื่อให้การอินเตอร์รัปต์ทั้งหมดไม่สามารถเกิดขึ้นได้

"1" เปิดการอินเตอร์รัปต์เพื่อให้การอินเตอร์รัปต์ทั้งหมดสามารถเกิดขึ้นได้

EA2 (Timer2 interrupt enable) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์จากไทเมอร์ 2 อันเนื่องจากไทเมอร์ 2 มีการนับขึ้นจนเกิดการโอเวอร์โฟลว์ หรือมีการแคบเจอร์เกิดขึ้น บิตนี้สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์จากไทเมอร์ 2

"1" เปิดการอินเตอร์รัปต์จากไทเมอร์ 2

ES (Serial port interrupt enable) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์จากพอร์ตอนุกรมอันเนื่องมาจากกระบวนการรับหรือส่งข้อมูลบิตนี้สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์จากพอร์ตอนุกรม

"1" เปิดการอินเตอร์รัปต์จากพอร์ตอนุกรม

ET1 (Timer1 interrupt enable) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์จากไทเมอร์ 1 อันเนื่องจากการเกิดโอเวอร์โฟลว์ของไทเมอร์ 1 บิตนี้สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์จากไทเมอร์ 1

"1" เปิดการอินเตอร์รัปต์จากไทเมอร์ 1

EX1 (External interrupt1 enable) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์จากจากภายนอกหมายเลข 1 (INT1 ; P3.3) อันเนื่องมาจากมีสัญญาณลอจิก "0" หรือสัญญาณขอบขาลงเข้ามาที่ขา INT1 บิตนี้สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์จากภายนอกหมายเลข 1

"1" เปิดการอินเตอร์รัปต์จากภายนอกหมายเลข 1

ET0 (Timer0 interrupt enable) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์จากไทเมอร์ 0 อันเนื่องมาจากการเกิดโอเวอร์โฟลว์ของไทเมอร์ 0 บิตนี้สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์จากไทเมอร์ 0

"1" เปิดการอินเตอร์รัปต์จากไทเมอร์ 0

EX0 (External interrupt0 enable) : ใช้ในการปิด/เปิดการอินเตอร์รัปต์จากจากภายนอกหมายเลข 0 (INT0 ; P3.2) อันเนื่องมาจากมีสัญญาณลอจิก "0" หรือสัญญาณขอบขาลงเข้ามาที่ขา INT0 บิตนี้สามารถเข้าถึงเพื่อเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ปิดการอินเตอร์รัปต์จากภายนอกหมายเลข 0

"1" เปิดการอินเตอร์รัปต์จากภายนอกหมายเลข 0

(ข) รีจิสเตอร์จัดลำดับความสำคัญของการอินเตอร์รัปต์ (Interrupt Priority register ; IP)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	
-	-	PT2	PS	PT1	PX1	PT0	PX0

รูปที่ 2.4 (ข) บิตต่างๆของรีจิสเตอร์ IP

รีจิสเตอร์ IP เป็นรีจิสเตอร์ฟังก์ชันพิเศษ มีขนาด 8 บิตอยู่ที่แอดเดรส 0×88 สามารถเข้าถึงในระดับบิตได้ ใช้ในการควบคุมลำดับความสำคัญของการอินเตอร์รัปต์ต่างๆ เนื่องจากในบางครั้งการอินเตอร์รัปต์จะเกิดขึ้นพร้อมกันหรือซ้อนกัน ดังนั้นจึงต้องมีการจัดลำดับความสำคัญของการอินเตอร์รัปต์ เพราะเมื่อมีการอินเตอร์รัปต์เกิดขึ้นซ้อนกัน CPU จะไปประมวลผลในโปรแกรมบริการ การอินเตอร์รัปต์ที่มีความสำคัญสูงกว่าก่อน หากต้องการให้การอินเตอร์รัปต์จากแหล่งกำเนิดไหนสูงกว่าสามารถทำได้โดยการให้บิตประจำตำแหน่งของการอินเตอร์รัปต์เป็น "1" รายละเอียดของแต่ละบิตมีดังต่อไปนี้

PT2 (Timer2 interrupt priority) : ใช้ในการกำหนดลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 2 (ไทเมอร์ 2 เกิดการนับขึ้นจึงเกิดการโอเวอร์โฟลว์ หรือมีการแคปเจอร์เกิดขึ้น) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟแวร์เท่านั้น

"0" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 2 มีความสำคัญต่ำ

"1" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 2 มีความสำคัญสูง

PS (Serial port interrupt priority) : ใช้ในการกำหนดลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากพอร์ตอนุกรม (การรับหรือส่งข้อมูลเสร็จสิ้น) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากพอร์ตอนุกรม มีความสำคัญต่ำ

"1" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากพอร์ตอนุกรม มีความสำคัญสูง

PT1 (Timer1 interrupt priority) : ใช้ในการกำหนดลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 1 (ไทเมอร์ 1 เกิดการนับขึ้นจนเกิดการโอเวอร์โฟลว์) บิตนี้สามารถเซตหรือเคลียร์ได้โดยการกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 1 มีความสำคัญต่ำ

"1" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 1 มีความสำคัญสูง

PX1 (External interrupt1 priority) : ใช้ในการกำหนดลำดับความสำคัญของการอินเตอร์รัปต์จากภายนอกหมายเลข 1 (มีสัญญาณที่เป็นลอจิก "0" หรือสัญญาณขอบขาลงเข้ามาที่ขา INT1) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ลำดับความสำคัญของการอินเตอร์รัปต์จากภายนอกหมายเลข 1 มีความสำคัญต่ำ

"1" ลำดับความสำคัญของการอินเตอร์รัปต์จากภายนอกหมายเลข 1 มีความสำคัญสูง

PT0 (Timer0 interrupt priority) : ใช้ในการลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 0 (ไทเมอร์ 0 เกิดการนับขึ้นจนเกิดการโอเวอร์โฟลว์) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 0 มีความสำคัญต่ำ

"1" ลำดับความสำคัญของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 0 มีความสำคัญสูง

PX0 (External interrupt0 priority) : ใช้ในการกำหนดลำดับความสำคัญของการอินเตอร์รัปต์จากภายนอกหมายเลข 0 (มีสัญญาณที่เป็นลอจิก "0" หรือสัญญาณขอบขาลงเข้ามาที่ขา INT0) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ลำดับความสำคัญของการอินเตอร์รัปต์จากภายนอกหมายเลข 0 มีความสำคัญต่ำ

"1" ลำดับความสำคัญของการอินเตอร์รัปต์จากภายนอกหมายเลข 0 มีความสำคัญสูง

(ค) รีจิสเตอร์ควบคุมการทำงานของไทเมอร์ 0 และ 1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

รูปที่ 2.4 (ค) บิตต่างๆของรีจิสเตอร์ TCON

รีจิสเตอร์ TCON เป็นรีจิสเตอร์ฟังก์ชันพิเศษ มีขนาด 8 บิตมีแอดเดรส 0×88 สามารถเข้าถึงในระดับบิตได้ บิต 6 หรือ บิต 8 ใช้ในการควบคุมการปิด/เปิดการทำงานของไทเมอร์ 0 และไทเมอร์ 0 ส่วนบิตที่เหลือใช้สำหรับกระบวนการอินเตอร์รัปต์ รายละเอียดของแต่ละบิตดังต่อไปนี้

TF1 (Timer1 overflow flag) : ใช้ในการบอกสถานะของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 1 บิตนี้จะเซตเป็น "1" เมื่อไทเมอร์ 1 นับขึ้นจนเกิดการโอเวอร์โฟลว์ นอกจากนี้ยังสามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์

"0" ไทเมอร์ 1 ยังไม่เกิดการโอเวอร์โฟลว์

"1" ไทเมอร์ 1 เกิดการโอเวอร์โฟลว์ ถ้ามีการอีนาเบิ้ลการอินเตอร์รัปต์ของไทเมอร์ 1 เอาไว้จะมีการอินเตอร์รัปต์จากไทเมอร์ 1 เกิดขึ้น

TF0 (Timer0 overflow flag) : ใช้ในการบอกสถานะของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 0 บิตนี้จะเซตเป็น "1" เมื่อไทเมอร์ 0 นับขึ้นจนเกิดการโอเวอร์โฟลว์ นอกจากนี้ยังสามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์

"0" ไทเมอร์ 0 ยังไม่เกิดการโอเวอร์โฟลว์

"1" ไทเมอร์ 0 เกิดการโอเวอร์โฟลว์ ถ้ามีการอีนาเบิ้ลการอินเตอร์รัปต์ของไทเมอร์1 เอาไว้จะมีการอินเตอร์รัปต์จากไทเมอร์ 0 เกิดขึ้น

IE1 (External interrupt1 edge flag) : ใช้ในการบอกสถานะของการอินเตอร์รัปต์ที่เกิดขึ้นจากการอินเตอร์รัปต์จากภายนอกหมายเลข 1 บิตนี้จะเซตเป็น "1" เมื่อมีสัญญาณลอจิก "0" หรือสัญญาณขอบขาลงเข้ามาที่ขา INT1(P3.3) นอกจากนี้ยังสามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์

"0" ยังไม่มีสัญญาณอินเตอร์รัปต์เข้ามาที่ขา INT1

"1" มีสัญญาณอินเตอร์รับเข้ามาที่ขา INT1 ถ้ามีการอีนาเบิ้ลการอินเตอร์รัปต์เอาไว้จะมีการอินเตอร์รัปต์จากภายนอกหมายเลข 1 เกิดขึ้น

IT1 (Intertup1 type control) : ใช้ในการกำหนดรูปแบบของการอินเตอร์รัปต์จากภายนอกหมายเลข 1 ว่าจะให้มีการอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" หรือมีสัญญาณขอบขาลงเข้ามาที่ขา INT0(P3.3) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ให้การอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" เข้ามาที่ขา INT1(P3.3)

"1" ให้การอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณขอบขาลงเข้ามาที่ขา INT1(P3.3)

IE0 (External interrupt0 edge flag) : ใช้ในการบอกสถานะของการอินเตอร์รัปต์ที่เกิดขึ้นจากการอินเตอร์รัปต์จากภายนอกหมายเลข 0 บิตนี้จะเซตเป็น "1" เมื่อมีสัญญาณลอจิก "0" หรือสัญญาณขอบขาลงเข้ามาที่ขา INT1(P3.2) นอกจากนี้ยังสามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์

"0" ยังไม่มีสัญญาณอินเตอร์รัปต์เข้ามาที่ขา INT0

"1" มีสัญญาณอินเตอร์รับเข้ามาที่ขา INT0 ถ้ามีการอินาเบิ้ลการอินเตอร์รัปต์เอาไว้จะมีการอินเตอร์รัปต์จากภายนอกหมายเลข 0 เกิดขึ้น

IT0 (Intertup0 type control) : ใช้ในการกำหนดรูปแบบของการอินเตอร์รัปต์จากภายนอกหมายเลข 0 ว่าจะให้มีการอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" หรือมีสัญญาณขอบขาลงเข้ามาที่ขา INT0(P3.2) บิตนี้สามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์เท่านั้น

"0" ให้การอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" เข้ามาที่ขา INT1(P3.2)

"1" ให้การอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณขอบขาลงเข้ามาที่ขา INT1(P3.2)

(ง) รีจิสเตอร์ควบคุมการทำงานของไทเมอร์ 2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2'	CP/RL2'

รูปที่ 2.4 (ง) บิตต่างๆของรีจิสเตอร์ T2CON

รีจิสเตอร์ T2CON เป็นรีจิสเตอร์ฟังก์ชันพิเศษ มีขนาด 8 บิตอยู่ที่แอดเดรส 0×CB สามารถเข้าถึงในระดับบิตได้ บิตที่ใช้สำหรับกระบวนการอินเตอร์รัปต์มีบิตเดียวคือบิต 7 รายละเอียดมีดังต่อไปนี้

TF2 (Timer2 overflow flag) : ใช้ในการบอกสถานะของการอินเตอร์รัปต์ที่เกิดขึ้นจากไทเมอร์ 2 บิตนี้จะเซตเป็น "1" เมื่อไทเมอร์ 2 นับขึ้นจนเกิดการโอเวอร์โฟลว์ หรือมีการแคปเจอร์เกิดขึ้นนอกจากนี้ยังสามารถเซตหรือเคลียร์ได้โดยกระบวนการทางซอฟต์แวร์

"0" ไทเมอร์ 2 ยังไม่มีการเกิดการโอเวอร์โฟลว์ และไม่มีการแคปเจอร์เกิดขึ้น

"1" ไทเมอร์ 2 มีการเกิดการโอเวอร์โฟลว์ หรือมีการแคปเจอร์เกิดขึ้น ถ้ามีการอีนาเบิ้ลการอินเตอร์รัปต์ของไทเมอร์ 2 เอาไว้จะมีการอินเตอร์รัปต์จากไทเมอร์ 2 เกิดขึ้น

2.4.2 การอินเตอร์รัปต์จากภายนอก

การอินเตอร์รัปต์จากภายนอกจะเกิดขึ้นเมื่อได้รับจากสัญญาณอินเตอร์รัปต์ที่ส่งเข้ามาจากภายนอกผ่านทางขารับสัญญาณอินเตอร์รัปต์ การอินเตอร์รัปต์จากภายนอกสามารถเกิดขึ้นได้ 2 แหล่งกำเนิด คืออินเตอร์รัปต์จากภายนอกหมายเลข 0 (External Interrupt 0 : INT0) และอินเตอร์รัปต์จากภายนอกหมายเลข 1 (External Interrupt 1 : INT1) รายละเอียดของการอินเตอร์รัปต์จากภายนอกทั้งสองแหล่งมีดังนี้

การเกิดอินเตอรัปต์จากภายนอกหมายเลข 0 (External Interrupt 0 : INT0)

การอินเตอร์รัปต์จากภายนอกหมายเลข 0 จะเกิดขึ้นเมื่อมีสัญญาณเข้ามาที่ขา INT0(P3.2) การอินเตอร์รัปต์ตัวนี้จะเกิดขึ้นได้ 2 ลักษณะคือ เมื่อขา INT0 ได้รับสัญญาณลอจิก "0" และเมื่อขา INT0 ได้รับสัญญาณขอบขาลง อินเตอร์รัปต์จากภายนอกหมายเลข 0 มีเวกเตอร์ของการอินเตอร์รัปต์ต่ออยู่ที่หมายเลข 0 (interrupt 0)

การเกิดอินเตอรัปต์จากภายนอกหมายเลข 1 (External Interrupt 1 : INT0)

การอินเตอร์รัปต์จากภายนอกหมายเลข 1 จะเกิดขึ้นเมื่อมีสัญญาณเข้ามาที่ขา INT0(P3.3) การอินเตอร์รัปต์ตัวนี้จะเกิดขึ้นได้ 2 ลักษณะคือ เมื่อขา INT1 ได้รับสัญญาณลอจิก "0" และเมื่อขา INT1 ได้รับสัญญาณขอบขาลง อินเตอร์รัปต์จากภายนอกหมายเลข 1 มีเวกเตอร์ของการอินเตอร์รัปต์ต่ออยู่ที่หมายเลข 2 (interrupt 2)

การเลือกรูปแบบของการเกิดอินเตอร์รัปต์จากภายนอก (INTO & INT1)

ดังที่ได้กล่าวมาแล้วจากข้างต้นมา การอินเตอร์รัปต์จากภายนอกสามารถเกิดขึ้นได้ 2 รูปแบบ คือ จะเกิดการอินเตอร์รัปต์เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" เข้ามาที่ขา INTx และมีสัญญาณขอบขาลงเข้ามาที่ขา INTx การเลือกรูปแบบของการอินเตอร์รัปต์ดังกล่าวสามารถทำได้โดยการกำหนดที่ ITx ดังนี้

***IT0 = "0"** ให้การอินเตอร์รัปต์หมายเลข 0 เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" เข้ามาที่ขา INT0

***IT0 = "1"** ให้การอินเตอร์รัปต์หมายเลข 0 เกิดขึ้นเมื่อมีสัญญาณขอบขาลงเข้ามาที่ขา INT0

***IT1 = "0"** ให้การอินเตอร์รัปต์หมายเลข 1 เกิดขึ้นเมื่อมีสัญญาณลอจิก "0" เข้ามาที่ขา INT1

***IT1 = "1"** ให้การอินเตอร์รัปต์หมายเลข 1 เกิดขึ้นเมื่อมีสัญญาณขอบขาลงเข้ามาที่ขา INT1

2.4.3 การอินเตอร์รัปต์จากภายใน

การอินเตอร์รัปต์จากภายในจะเกิดขึ้นจากสัญญาณอินเตอร์รัปต์ของวงจรภายในที่มีอยู่ในตัวไมโครคอนโทลเลอร์ เช่น ไทเมอร์ และพอร์ตอนุกรม เป็นต้น

การอินเตอร์รัปต์จากไทเมอร์ 0

ไทเมอร์ 0 สามารถสร้างสัญญาณอินเตอร์รัปต์ให้เกิดขึ้นได้เมื่อมีการนับขึ้นจนเกิดการโอเวอร์โฟลว์ เมื่อไทเมอร์ 0 นับขึ้นจนเกิดการโอเวอร์โฟลว์ จะมีผลทำให้บิต TF0 ซึ่งเป็นบิตที่ 5 ของรีจิสเตอร์ TCON เกิดการเซตเป็น "1" และในเวลาเดียวกันนี้เองหากมีการอีนาเบิ้ลการอินเตอร์รัปต์ (EA=1; ET0=1;) CPU จะกระโดดไปประมวลผลยังโปรแกรมบริการ การอินเตอร์รัปต์ของไทเมอร์ 0 โดยการอินเตอร์รัปต์จากไทเมอร์ 0 มีเว็กเตอร์ของการอินเตอร์รัปต์อยู่ที่หมายเลข 1 (interrupt 1)

การเกิดโอเวอร์โฟลว์ของไทเมอร์ จะเกิดขึ้นในจังหวะของการเปลี่ยนแปลงจากค่าสูงสุดไปยังค่า 0 หรือเกิดขึ้นในจังหวะของการเริ่มนับในรอบใหม่นั้นเอง

การอินเตอร์รัปต์จากไทเมอร์ 1

การอินเตอร์รัปต์จากไทเมอร์ 1 ไม่ได้มีความแตกต่างไปจากการอินเตอร์รัปต์จากไทเมอร์ 0 นั้น คือ ไทเมอร์ 1 สามารถสร้างสัญญาณอินเตอร์รัปต์ให้เกิดขึ้นได้เมื่อมีการนับขึ้นจนเกิดการโอเวอร์โฟลว์ เมื่อไทเมอร์ 1 นับขึ้นจนเกิดการโอเวอร์โฟลว์ จะมีผลทำให้บิต TF1 ซึ่งเป็นบิตที่ 7 ของรีจิสเตอร์ TCON เกิดการเซตเป็น "1" และในเวลาเดียวกันนี้เองหากมีการอีนาเบิ้ลการอินเตอร์รัปต์ไว้ (EA=1; ET=1;) CPU จะกระโดดไปประมวลผลยังโปรแกรมบริการ การอินเตอร์รัปต์ของไทเมอร์ 1 โดยการอินเตอร์รัปต์จากไทเมอร์ 1 มีเว็กเตอร์ของการอินเตอร์รัปต์อยู่ที่หมายเลข 3 (interrupt 3)

การอินเตอร์รัปต์จากไทเมอร์ 2

ไทเมอร์ 2 เป็นไทเมอร์ที่มีความสามารถมากกว่า 0 และ 1 อย่างเห็นได้ชัด เช่น สามารถทำงานในโหมด 16 บิต รีโหลดค่าอัตโนมัติ (16 bit auto-reload) และโหมดแคปเจอร์ (Capture) ได้ด้วย ความสามารถดังกล่าวจำทำให้ไทเมอร์ 2 สามารถสร้างสัญญาณอินเตอร์รัปต์ของไทเมอร์ได้ 2 แหล่งกำเนิดนั้น คือการอินเตอร์รัปต์อันเนื่องมาจากเกิดการโอเวอร์โฟลว์ และการอินเตอร์รัปต์อันเนื่องมาจากการแคปเจอร์การเกิดการอินเตอร์รัปต์ทั้ง 2 กรณีสามารถแยกอธิบายได้ดังต่อไปนี้

การอินเตอร์รัปต์จากไทเมอร์ 2 เมื่อเกิดการโอเวอร์โฟลว์

การอินเตอร์รัปต์จากการโอเวอร์โฟลว์ของเมื่อไมเทอร์ 2 ไม่ได้มีความแตกต่างจากการอินเตอร์รัปต์จากไทเมอร์ 0 และ 1 คือเมื่อไมเทอร์ 2 นับขึ้นจนเกิดการโอเวอร์โฟลว์จะทำให้บิต TF2 ซึ่งเป็นบิตที่ 7 ในรีจิสเตอร์ T2CON เซตเป็น "1" และในเวลาเดียวกันนี้เองหากมีการอีนาเบิ้ลการอินเตอร์รัปต์ไว้ (EA=1; ET=1;) CPU จะกระโดดไปประมวลผลยังโปรแกรมบริการ การอินเตอร์รัปต์ของไทเมอร์ 2

การอินเตอร์รัปต์จากไทเมอร์ 2 เมื่อมีการแคปเจอร์

การอินเตอร์รัปต์จากไทเมอร์ 2 เมื่อมีการแคปเจอร์ เป็นอินเตอร์รัปต์ที่เกิดขึ้นอันเนื่องมาจากไมโครคอนโทรลเลอร์ตรวจจับสัญญาณขอบขาลง (เปลี่ยนจาก "1" เป็น "0") ที่ขา T2EX(P1.1) ได้ การตรวจจับสัญญาณนี้ได้จะมีผลทำให้บิต EXF2 ซึ่งเป็นบิตที่ 6 ในรีจิสเตอร์ T2CON เซตเป็น "1" และในเวลาเดียวกันเองหากมีการอีนาเบิ้ลการอินเตอร์รัปต์ไว้ (EA=1; ET=1; EXEN2=1;) CPU จะกระโดดไปประมวลผลยังโปรแกรมบริการ การอินเตอร์รัปต์ของไทเมอร์ 2 โดยการอินเตอร์รัปต์จากไทเมอร์ 2 มีเวกเตอร์ของการอินเตอร์รัปต์อยู่ที่หมายเลข 5 (interrupt 5)

การอินเตอร์รัปต์ที่เกิดจากไทเมอร์ 2 ทั้ง 2 กรณีจะเวกเตอร์ในการอินเตอร์รัปต์ตัวเดียวกันคือเวกเตอร์อินเตอร์รัปต์หมายเลขที่ 5 (interrupt 5) ดังนั้นในการใช้งานจะต้องใช้บิต EXEN2 เป็นตัวเลือกว่าจะให้เกิดการอินเตอร์รัปต์จากกรณีใด ดังนี้

EXEN2 = "0" ตอบสนองการอินเตอร์รัปต์จากไทเมอร์ 2 เมื่อมีการนับขึ้นจนเกิดโอเวอร์โฟลว์

EXEN2 = "1" ตอบสนองการอินเตอร์รัปต์จากไทเมอร์ 2 เมื่อมีสัญญาณของขาลงเข้ามาที่ขา T2EX(P1.1) หรือมีการแคปเจอร์เกิดขึ้นนั้นเอง

2.4.4 การคำนวณความเร็วการรับและส่งข้อมูลแบบอนุกรม (Generating Baud Rate)

การกำหนดความเร็วการรับส่งข้อมูลแบบอนุกรมสามารถแบ่งออกตาม Mode การทำงานดังนี้

Mode 0

ความเร็วในการรับส่งข้อมูลแบบอนุกรมใน Mode นี้จะกำหนดอัตราการรับส่งตายตัวเท่ากับ 1/12 ของความถี่ของชุดกำเนิดความอ้างอิงของ 8051 และจะไม่ใช้ Timer/Counter ดังนั้นเพียงกำหนดที่รีจิสเตอร์ SCON ก็เพียงพอ จะได้

Baud Rate = Osc.Freq/12

Mode 1

ในการกำหนดความเร็วการรับส่งข้อมูลแบบอนุกรมใน Mode 1 นี้จะใช้ Timer 1 เป็นฐานเวลาของการทำงานของ Timer 1 ใน Mode 2 (Auto-Reload) โดยสามารถคำนวณได้ดังนี้

Baud Rate = [K*Osc.Freq] / [32*12*(256-(TH1))]

K = 1 เมื่อ SMOD ในรีจิสเตอร์ PCON = 0

K = 2 เมื่อ SMOD ในรีจิสเตอร์ PCON = 1

ส่วนมากแล้ว ผู้ใช้จะทราบค่าของ Baud Rate ที่จะส่งได้ค่านั้น จะได้ค่าของ Time 1 สำหรับ Reload ได้เป็น

TH1 = 256 – [(k*Osc.Freq) / (384 * Baud Rate)]

จากตารางต่อไปนี้ แสดง Baud Rate ต่างๆ และค่า Reload ของ Time 1

ตารางที่ 2.3 Baud Rate ต่างๆ และค่า Reload ของ Time1

Baud Rate	Freq osc	SMOD	Timer1		
			C/T	Mode Value	Reload
Mode 0 Max : 1MHz	12 MHz	X	X	X	X
Mode 2 Max : 375 K	12 MHz	1	X	X	X
Modes 1.3 : 62.5 K	12 MHz	1	0	2	FFH
19.2 K	11.059 MHz	1	0	2	FDH
9.6 K	11.059 MHz	0	0	2	FDH
4.8 K	11.059 MHz	0	0	2	FAH
2.4 K	11.059 MHz	0	0	2	F4H
1.2 K	11.059 MHz	0	0	2	E8H
137.5	11.986 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

Mode 2

ความเร็วการรับส่งใน Mode นี้จะเป็นค่าคงที่ซึ่งมี 2 ค่า ขึ้นกับค่า SMOD ในรีจิสเตอร์ PCON ดังนี้

เมื่อ SMOD = 1 Baud Rate = 1/32 Osc.Freq

เมื่อ SMOD = 0 Baud Rate = 1/64 Osc.Freq

Mode 3

การกำหนดความเร็วการรับส่งใน Mode 3 จะคิดเช่นเดียวกับการคิดใน Mode 1 สำหรับค่าเริ่มต้นของรีจิสเตอร์ SFR เมื่อ 8051 ถูกรีเซ็ต จะมีค่าดังแสดงในตารางต่อไปนี้

ตารางที่ 2.4 ค่าเริ่มต้นของรีจิสเตอร์ SFR เมื่อ 8051 ถูกรีเซ็ต

SFR Name	Reset Value
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0 – P3	FFH
IP (8051)	XXX0000B
IP (8052)	XX000000B
IE(8051)	0XX00000B
IE (8052)	0X000000B
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2(8052)	00H
TL2(8052)	00H
RCAP2H (8052)	00H
RCAP2L(8052)	00H
SCON	00H
SBUF	Indeteminate
PCON(HMOS)	0XXXXXXXXB
PCON(CHMOS)	0XXX0000B

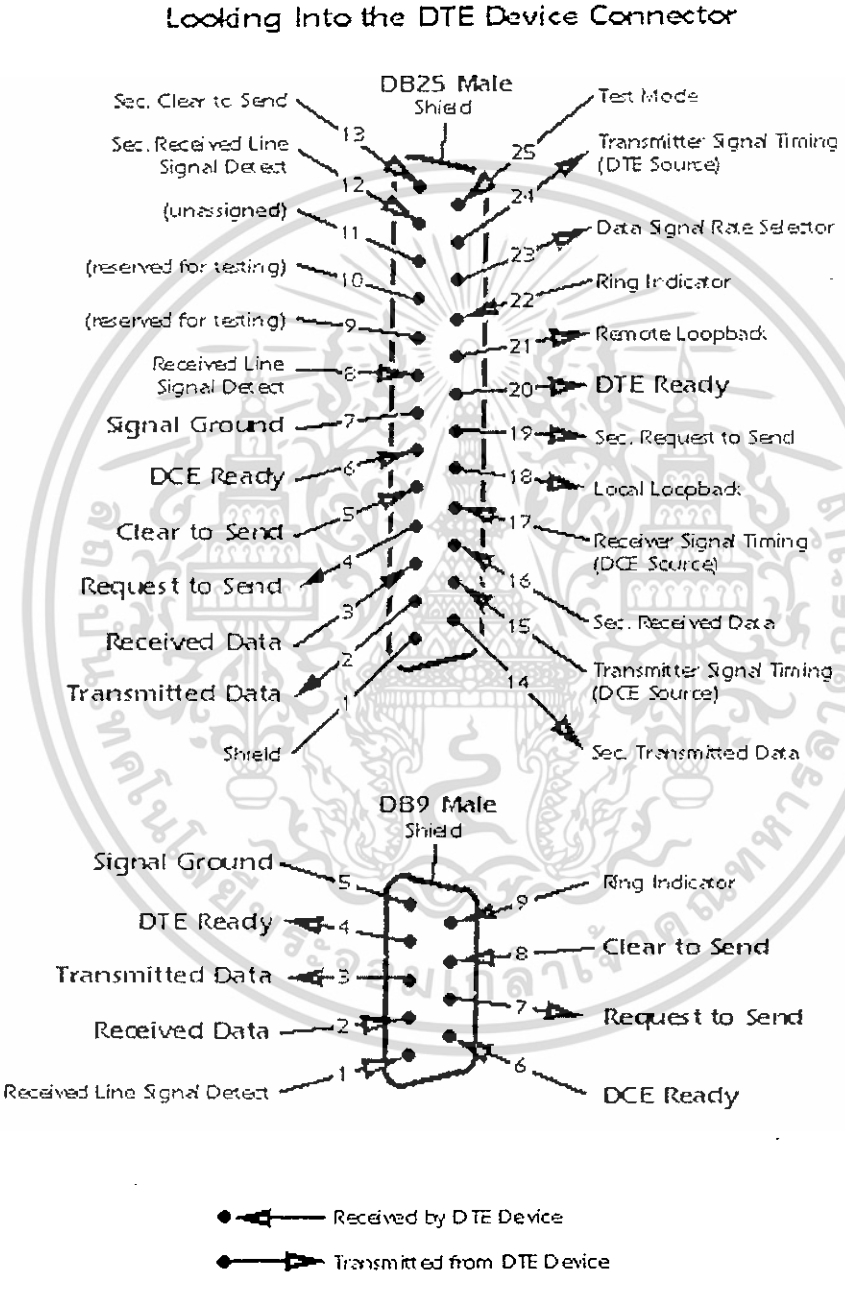
2.5 RS-232

เป็นมาตรฐานที่ออกแบบมาสำหรับการติดต่อสื่อสารระหว่างอุปกรณ์สื่อสารข้อมูลต่างๆโดยมีรายละเอียดทั้งในเรื่องระดับสัญญาณ เวลาของสัญญาณ รูปแบบโปรโตคอลที่ใช้และตลอดจนลักษณะทางกายภาพของอุปกรณ์ โดยมีรายละเอียดดังนี้

2.5.1. การจัดขาของอุปกรณ์

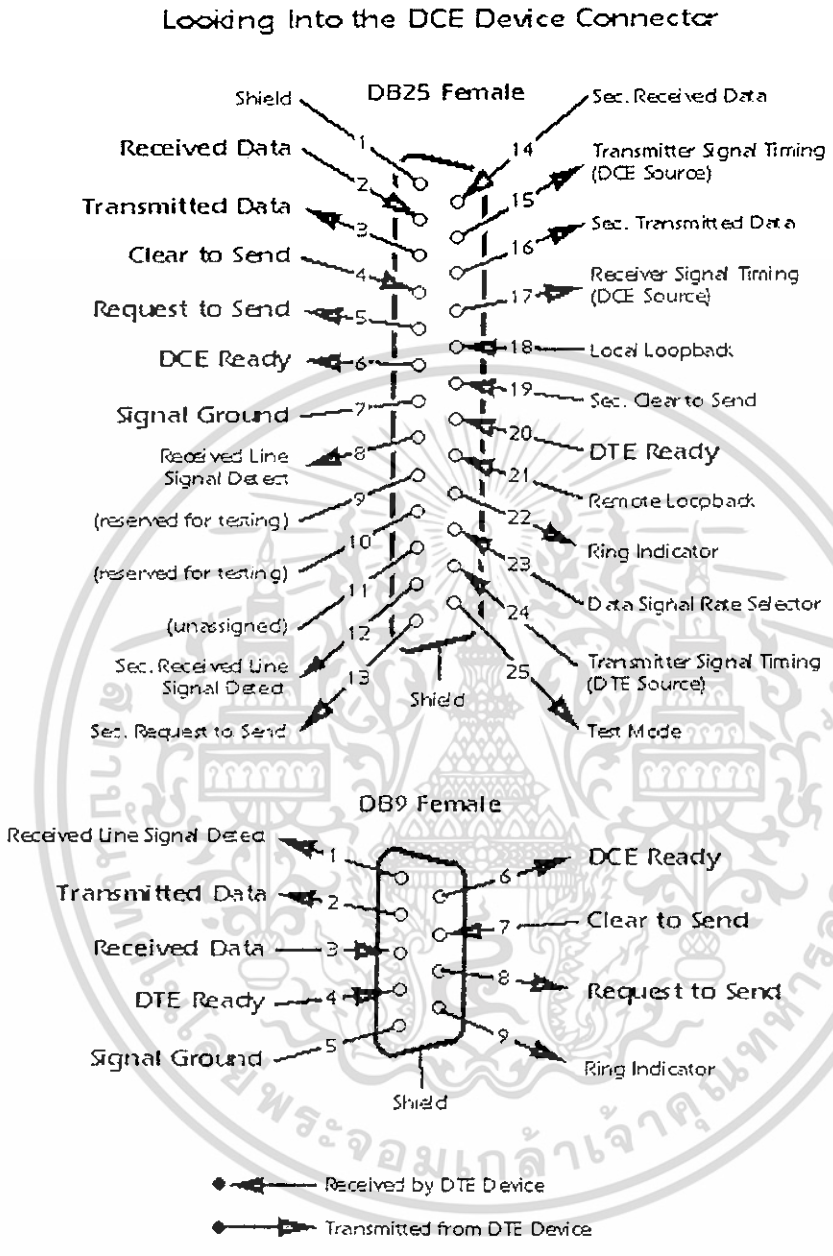
แบ่งเป็น 2 ลักษณะ คือการจัดขาทางด้าน DTE : Data Terminal Equipment(computer) และการจัดขาทางด้าน DCE : Data Circuit-terminating Equipment (อุปกรณ์ต่อพ่วง เช่น Moderm) ดังนี้

2.5.1.1 การจัดขาทางด้าน DTE



รูปที่ 2.5 การจัดขาทางด้าน DTE

2.5.1.2 การจัดขาทางด้าน DCE



รูปที่ 2.6 การจัดขาทางด้าน DCE

ต่อไปเราจะกล่าวเฉพาะการจัดขาแบบ 9 pin เนื่องจากได้นำมาใช้ในโครงงานนี้ ความหมายของชื่อ pin ต่างๆ(แบบ 9 pin)

1. Signal Ground เป็น Ground ของสัญญาณ

2. Transmitted Data (TxD) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรมออกไปยังอุปกรณ์อื่นๆ

3. Received Data(RxD) เป็นขาที่ใช้รับข้อมูลอนุกรมจากอุปกรณ์ภายนอกเข้ามา

4. Request to Send(RTS) เป็นขาที่ใช้ส่งสัญญาณจากอุปกรณ์ DTE ให้อุปกรณ์ DCE เตรียมตัวรับข้อมูลได้

5. Clear to Send (CTS) เป็นขาที่อุปกรณ์ DCE จะส่งสัญญาณตอบกลับอุปกรณ์ DTE ว่าการส่งข้อมูลจะเกิดขึ้นได้

6. DCE Ready(DSR) เป็นขาที่ใช้บอกว่าอุปกรณ์ DCE ต่ออยู่อย่างถูกต้อง

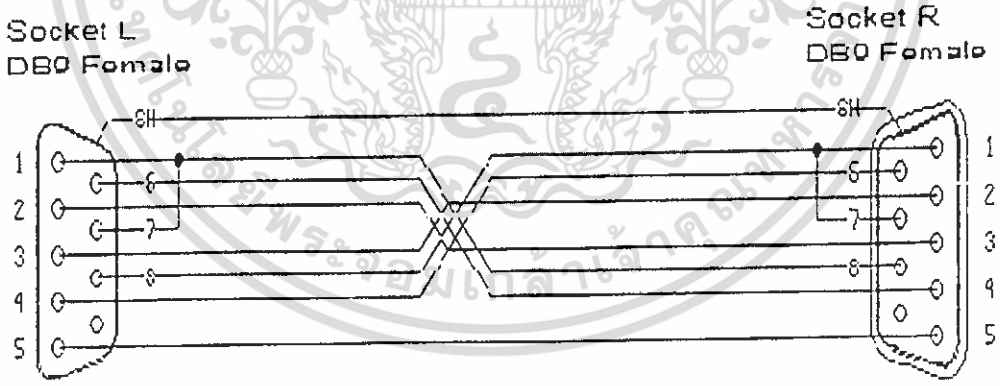
7. DTE Ready (DTR) เป็นขาที่ใช้บอกว่าอุปกรณ์ DTE ต้องการเปิดช่องสัญญาณในการสื่อสาร

8. Received line Signal Detector (CD) บางทีก็เรียกว่า Carrier Detector เป็นขาที่ใช้ในการรติดต่อ ในกรณีที่อุปกรณ์ DCE เป็นโมเด็ม ขานี้จะรับสัญญาณจากการตรวจสอบ Answer tone ของโมเด็ม

9. Ring Indicator (RI) เป็นขาที่ใช้ในการตรวจสอบสัญญาณกระดิ่ง เมื่ออุปกรณ์ DCE เป็นโมเด็ม

2.5.2 การเชื่อมต่อแบบ Null moderm

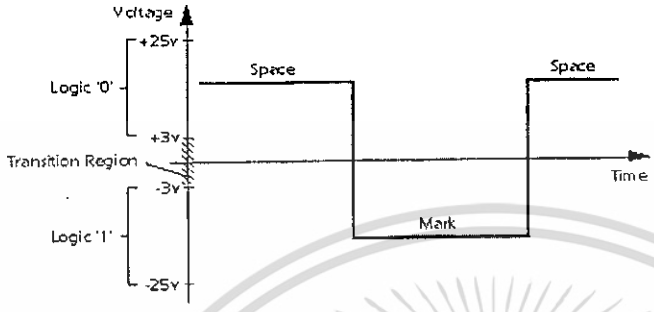
เป็นการเชื่อมต่อคอมพิวเตอร์ด้วยกันโดยใช้ Port อนุกรม โดยไม่ต้องใช้โมเด็ม การนำไปใช้งานต้องมีการต่อสายให้ถูกต้องดังนี้



รูปที่ 2.7 การต่อสาย null modem

2.5.3 ระดับแรงดัน

ระดับแรงดันตามมาตรฐาน RS232 จะไม่เหมือนกับ logic แบบ TTL เนื่องจากในการส่งผ่านสายสัญญาณ จะมีสัญญาณรบกวนเกิดขึ้นและรบกวนข้อมูลทางของเราได้ มาตรฐานระดับแรงดัน logic 0 และ 1 แสดงในรูปที่ 2.8



รูปที่ 2.8 ระดับแรงดันในมาตรฐาน RS - 232

จากรูปที่ 2.8 จะเห็นได้ว่าช่วง logic 0 จะมีระดับแรงดันเป็น +3 ถึง +25 V ส่วนช่วง logic 1 จะมีระดับแรงดันเป็น -3 ถึง -25 แต่โดยทั่วไปช่วง logic 0 จะมีระดับแรงดันเป็น +8 ถึง +14 ส่วนช่วง logic 1 จะมีระดับแรงดันเป็น -8 ถึง -14

2.6 ช่วงเวลาของสัญญาณ

ในมาตรฐาน RS232 กำหนดช่วงเวลาของสัญญาณไว้ดังนี้

ในการใช้งานปกติมักจะใช้ความเร็วบอร์ด 300,1200,2400,9600,19200 แต่ไม่ได้กำหนดความเร็วของบอร์ดไว้แน่นอน

ในการเปลี่ยนแปลงของสัญญาณจากระดับ logic หนึ่งไปยังอีกระดับ logic หนึ่งมีข้อกำหนดดังนี้

1. สัญญาณที่ผ่านช่วง transition region จะต้องเปลี่ยนไปยังอีกช่วงระดับหนึ่งของ logic หนึ่งโดยไม่มีการย้อนกลับ

2. สัญญาณควบคุมต้องผ่านช่วง transition region โดยใช้เวลาน้อยกว่า 1 ms

3. สัญญาณข้อมูล หรือ สัญญาณ timing ถ้าช่วงกว้างมากกว่า 25 ms ให้ใช้เวลาเปลี่ยนผ่านช่วง transition region น้อยกว่า 1 ms แต่ถ้าช่วงกว้างบิตอยู่ระหว่าง 125 μs จนถึง 25 ms ให้ใช้เวลาเปลี่ยน 4% ของช่วงกว้างบิต และถ้าช่วงกว้างบิตน้อยกว่า 125 μs ให้ใช้น้อยกว่า 5 μs

4. Slope ควรน้อยกว่า 30 V/μs

2.7 Delphi 7

Delphi 7 คือซอฟต์แวร์ที่สามารถนำมาใช้ในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชั่น หรือซอฟต์แวร์ โดยประกอบไปด้วยเครื่องมือชนิดต่างๆ ที่ใช้ให้การเขียนโปรแกรมทำได้อย่างสะดวก

Delphi 7 เป็นเครื่องมือเขียนโปรแกรมชนิด Visual Programming เช่นเดียวกับ Visual Basic หรือ Visual C++ โดยมีข้อดีคือ สามารถเขียนโปรแกรมได้ง่าย และให้ผลงานออกมาอย่างรวดเร็ว ซึ่งจะแตกต่างจากเครื่องมือเขียนโปรแกรมรุ่นเดิมๆ เช่น Turbo Pascal หรือ Borland C ที่มีความยุ่งยากในการทำงานและการเรียนรู้ในการเขียนโปรแกรม ดังนั้นจึงจัดให้ Delphi 7 เป็นซอฟต์แวร์ประเภท RAD หรือ Rapid Application Development ซึ่งแปลว่าสามารถสร้างแอพพลิเคชันได้อย่างรวดเร็ว

องค์ประกอบของ Delphi 7

หลังจากติดตั้งแล้วเสร็จ เราสามารถเรียกใช้ Delphi 7 ได้โดยคลิกที่ Start > All Programs > Borland Delphi 7 > Delphi7

องค์ประกอบที่สำคัญต่างๆ สามารถอธิบายโดยย่อได้ดังนี้

เมนูบาร์	เป็นส่วนที่เก็บคำสั่งเพื่อสั่งงาน Delphi 7 โดยมีลักษณะเป็นเมนูให้เลือกใช้งานเหมือนกับโปรแกรมอื่นๆ ทั่วไป
Form Designer	เป็นส่วนที่ใช้ในการออกแบบโปรแกรมที่สร้างขึ้นมาด้วย Delphi7 ที่เราเรียกว่าฟอร์ม (Form) ซึ่งมีลักษณะเหมือนกับวินโดว์ทั่วไป
Component Palette	เป็นส่วนที่เก็บคอมโพเนนต์ (Component) ซึ่งจะเป็นส่วนสำคัญที่จะเป็นองค์ประกอบภายในฟอร์ม เช่น ปุ่มคลิก,ปุ่มเลือก,รายการตัวเลือกเป็นต้น
Object tree View	เป็นส่วนที่ใช้อ้างอิงถึงออบเจ็กต์ และคอมโพเนนต์ต่างๆ ที่นำมาใช้ในระหว่างการพัฒนาแอพพลิเคชั่น
Object Inspector	เป็นส่วนที่ใช้กำหนดค่าให้กับฟอร์ม หรือคอมโพเนนต์ที่อยู่บนฟอร์มค่าที่กำหนดนี้จะทำให้หน้าตา หรือความสามารถของฟอร์มคอมโพเนนต์เปลี่ยนไป เช่น เปลี่ยนสีพื้น, เปลี่ยนข้อความบนปุ่ม, ปรับเปลี่ยนขนาดกว้างยาวของฟอร์ม เป็นต้น

นอกจากนี้ยังมีองค์ประกอบอื่นอีก ซึ่งจะพบเมื่อเราเริ่มเขียนโปรแกรมกับ Delphi 7 ดังนี้

Code Editor	เป็นส่วนที่เราจะใช้เขียนคำสั่งเพื่อให้โปรแกรม หรือแอพพลิเคชั่นที่สร้างขึ้นมานั้นทำงานได้ตามที่เราตั้งใจไว้ ซึ่งเราจะเขียนคำสั่งต่างๆด้วยภาษา Object Pascal (คล้ายกับภาษา Pascal มาก)

เรียกองค์ประกอบต่างๆ ที่กำลังจะใช้งานเพื่อเขียนโปรแกรมนี้ว่า IDE (Integrate Development Environment) ซึ่งก็คือ สภาพแวดล้อมที่ประกอบต่างๆ ข้างต้น

2.7.1 ขั้นตอนการเขียนโปรแกรมกับ Delphi 7

Delphi 7 นั้นได้รับการพัฒนาให้สามารถเขียนโปรแกรมเพื่อใช้งานกับ Windows ได้เป็นอย่างดี ขณะเดียวกันก็สามารถนำโปรแกรมที่เขียนได้นี้ไปปรับปรุงพัฒนาเพิ่มเติมเพื่อใช้งานกับระบบปฏิบัติการอื่นๆอย่างลีนุกซ์(Linux) ได้

สำหรับการเขียนโปรแกรมกับ Delphi นั้นจะใช้แนวทางการเขียนโปรแกรมแบบ Event Driven ซึ่งเป็นการเขียนโปรแกรมในลักษณะที่ว่าเมื่อมีเหตุการณ์อย่างหนึ่งเกิดขึ้นกับตัวโปรแกรม เราจะมีวิธีจัดการกับเหตุการณ์นั้น (ด้วยการเขียนคำสั่งต่างๆ เพื่อรองรับเหตุการณ์ที่จะเกิดขึ้น) ได้อย่างไร

2.7.2 ออบเจ็กต์

ประกอบไปด้วยวินโดว์, ปุ่มกด, ตัวเลือก, รูปภาพ สิ่งต่างๆที่ประกอบกันเป็นแอพพลิเคชันนี้เราเรียกว่า ออบเจ็กต์(Object)

การเขียนโปรแกรมกับ Delphi7 จะต้องเขียนภาษา object Pascal ในการเขียนโปรแกรม ซึ่ง Object Pascal เป็นภาษาปาสคาลที่ได้รับการปรับปรุงให้มีคุณสมบัติในการเขียนโปรแกรมแบบ OOP (Object Oriented Programming) ดังนั้นสิ่งแรกที่ต้องรู้นั่นคือ ออบเจ็กต์

2.7.3 กำหนดความสามารถด้วยเมธอด (Method)

เมธอด คือความสามารถในการทำงานอย่างหนึ่งของออบเจ็กต์ ซึ่งจะถูกเรียกใช้งานตอนที่เราเขียนโปรแกรม ซึ่งออบเจ็กต์คนละชนิดกันก็จะมีเมธอดที่แตกต่างกันไป

2.7.4 อีเวนต์ (Event)

Delphi นั้นเป็นการเขียนโปรแกรม Event Driven คือใช้เหตุการณ์ต่างๆที่เกิดขึ้นมาทำการขับเคลื่อนการทำงานแอพพลิเคชัน ซึ่งสิ่งที่เราจะต้องพบเจอในขณะที่เขียนโปรแกรมก็คือการจัดการกับเหตุการณ์แต่ละแบบ ซึ่งเราจะเรียกแต่ละเหตุการณ์ว่า อีเวนต์

2.8 ขั้นตอนการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชั่นกับ Delphi 7

2.8.1 ออกแบบโปรแกรม

ขั้นตอนแรกของการเขียนโปรแกรมที่ควรต้องทำ โดยจะต้องออกแบบเสียก่อนว่าจะให้โปรแกรมมีหน้าตาอย่างไร มีขั้นตอนการทำงานเป็นอย่างไร

ในที่นี้เราจะออกแบบโปรแกรมโดยวาดคร่าวๆ โดยให้โปรแกรมของเรามีหน้าต่างเดียวซึ่งจะมีปุ่มตัวเลือกอยู่ด้านซ้ายมือ มีส่วนแสดงรูปภาพอยู่ทางด้านขวามือ มีปุ่มจบการทำงานอยู่ด้านล่าง ซึ่งปุ่มจบการทำงานนี้จะใส่รูปภาพที่สื่อให้เห็นได้อย่างชัดเจน

2.8.2 ฟอร์ม และคอมโพเนนต์

ในการเขียนโปรแกรมกับ Delphi เราะเห็นว่าสิ่งแรกที่เราต้องเจอคือ ฟอร์ม (Form) ซึ่งเป็นหน้าต่างหลัก ซึ่งเราจะใช้ในการติดต่อกับผู้ใช้งาน การรับข้อมูล หรือการแสดงผลการทำงานให้กับผู้ใช้งานได้เข้าใจ

สิ่งที่จะทำหน้าที่ติดต่อกับผู้ใช้งานโดยตรง โดยทั้งหมดจะวางอยู่บนฟอร์ม

พร้อพเพอร์ตี้ที่สนใจของฟอร์ม

เมื่อสังเกตที่ฟอร์มเราจะพบว่ามันมีพร้อพเพอร์ตี้อยู่หลายตัว ซึ่งพร็อมเพอร์หลายตัวสามารถมองเห็น

Name	เป็นชื่อที่ใช้เรียกฟอร์ม ปกติ Delphi 7 จะตั้งให้เป็น Form1แต่เรามักจะเปลี่ยนชื่อให้จดจำได้ง่าย และสื่อความหมาย
Caption	เป็นข้อความที่แสดงบนแถบด้านบนของฟอร์ม
Cursor	เป็นลักษณะของตัวชี้ของเมาเมื่อเราเลื่อนวางเหนือฟอร์ม ปกติเป็นรูปลูกศรแต่เราสามารถเปลี่ยนเป็นรูปอื่นๆ เช่น นาฬิกาทราย ได้ตามต้องการ
Color	เป็นการกำหนดสีพื้นของฟอร์มตามที่ต้องการ
Position	เป็นตำแหน่งที่ฟอร์มจะถูกแสดงผลครั้งแรกเมื่อเรียกขึ้นมาใช้งาน เราสามารถกำหนดตำแหน่งได้ จะกลางหน้าจอ มุมใดของหน้าจอ หรือแม้แต่ตำแหน่งเดียวกับตอนที่เขียนโปรแกรมก็ได้
Windowstate	เป็นลักษณะของฟอร์มเมื่อแสดงผลว่าจะเป็นฟอร์มตามปกติ ย่อให้เล็กเป็นไอคอน (Minimize) หรือแสดงเต็มหน้าจอ (Maximize)
Icon	เป็นการกำหนดรูปภาพที่แสดงเป็นไอคอนเมื่อฟอร์มถูกยุบ(Minimize) ลงปกติ Delphi 7 จะกำหนดให้อยู่แล้ว แต่เราสามารถปรับเปลี่ยนได้ตามต้องการ
BorderStyle	เป็นลักษณะของขอบของฟอร์ม ซึ่งเราสามารถกำหนดได้ว่าจะยอมให้มีการปรับขนาดของฟอร์ม(ค่าปกติ) หรือเป็นฟอร์มที่มีขนาดคงที่ หรือจะเป็นหน้าจอแบบไดอะล็อกเป็นต้น.
Left	เป็นพิกัดในแกน X ตำแหน่งมุมบนซ้ายของหน้าจอ
Top	เป็นพิกัดในแกน Y ตำแหน่งมุมบนซ้ายของหน้าจอ
Width	เป็นขนาดความกว้างของฟอร์ม (วัดจากมุมบนซ้ายมาทางมุมบนขวา)ในหน่วยพิกเซล
Height	เป็นขนาดความสูงของฟอร์ม(วัดจากมุมบนซ้ายลงมามุมล่างซ้าย) ในหน่วยพิกเซล
ClientWidth	เป็นขนาดความกว้างของ Client Area มีหน่วยเป็นพิกเซล
ClientHeight	เป็นขนาดความสูงของ Client Area
AutoScroll	ถ้ากำหนดค่าเป็น True จะทำให้ Delphi เติมสกรอลบาร์ให้โดยอัตโนมัติทุกครั้งที่มีการปรับขนาดของฟอร์มให้เล็กลงจนไม่สามารถเห็นคอมโพเนนต์ต่างๆ ได้ครบทั้งหมด
FormStyle	เป็นการกำหนดลักษณะของฟอร์ม เช่น เป็นฟอร์มเดี่ยว (SDI) หรือฟอร์มแม่ที่มีฟอร์มลูกๆอยู่ภายในได้อีก (MDI)

2.8.3 เมธอดที่สำคัญของฟอร์ม

Show	เป็นเมธอดที่สั่งให้ฟอร์มแสดงขึ้นมา
ShowModal	เป็นเมธอดที่สั่งให้ฟอร์มนั้นแสดงขึ้นมาแบบ Modal (ฟอร์มแบบ Modal คือฟอร์มที่จะไม่ยอมให้มีฟอร์มอื่นๆ แสดงขึ้นมาเหนือตัวเอง ดังนันจึงต้องปิดฟอร์มแบบ Modal เสียก่อนจึงจะสามารถแสดงฟอร์มอื่นขึ้นมาได้)
Close	เป็นเมธอดที่สั่งให้ปิดฟอร์ม
Hide	เป็นเมธอดที่ซ่อนฟอร์มไว้ไม่ให้มองเห็น แต่ถือว่ายังไม่ปิดฟอร์ม

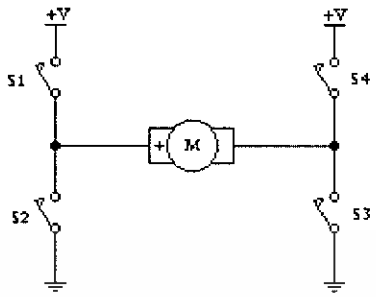
2.8.4 อีเวนต์ที่สำคัญของฟอร์ม

ฟอร์มนั้นมีอีเวนต์ต่างๆ ที่จะเกิดขึ้นได้มากมาย แต่เราใช้งานบ่อยๆ นั้นมีดังนี้

OnCreate	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการขอหน่วยความจำจาก Windows เพื่อเริ่มสร้างฟอร์ม
Onshow	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการแสดงฟอร์ม (ฟอร์มนั้นได้เปิดไว้แล้ว)
OnResize	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการปรับขนาดของฟอร์ม
OnActivate	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อย้ายการทำงานกลับเข้ามายังฟอร์ม(ที่ได้เปิดฟอร์มไว้แล้ว)
OnDeactivate	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อย้ายการทำงานออกไปจากฟอร์ม(แต่ฟอร์มยังไม่ปิด)
ONCloseQuery	เป็นเหตุการณ์ที่เกิดขึ้นก่อนจะเริ่มปิดฟอร์ม อาจจะมีการถามยืนยันก่อนการปิดฟอร์ม
OnClose	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อปิดฟอร์ม
OnDestroy	เป็นเหตุการณ์ที่เกิดขึ้นเมื่อปิดฟอร์มเสร็จแล้ว และได้คืนหน่วยความจำที่ใช้สร้างฟอร์มให้กับ Windows แล้ว

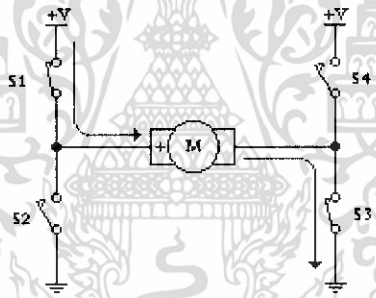
2.9 หลักการทำงานของวงจร H-Bridge Switching

เริ่มจากหลักการของวงจรนั้น จะประกอบไปด้วย สวิทช์ 4 ตัว นั่นก็คือ S1 ,S2 ,S3 และ S4 ซึ่งในรูปตัวอย่าง จะใช้ DC-Motor เป็น Load ของวงจร



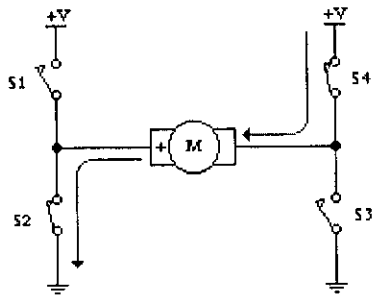
รูปที่ 2.9 H-Bridge Switching

ในสภาวะเริ่มต้น สวิทช์ ทุกตัว Off อยู่ ก็จะไม่มีอะไรเกิดขึ้นทั้งสิ้น เพราะไม่มีกระแสไฟฟ้าไหลเข้าสู่มอเตอร์ (รูปที่2.9)



รูปที่ 2.10 H-Bridge Switching

และเมื่อเราทำการ On สวิทช์ S1 และ S3 พร้อมกัน (รูปที่2.10) จะเป็นการเชื่อมวงจร ทำให้มีกระแสไฟฟ้า ไหลผ่านมอเตอร์ จากขั้วบวกของมอเตอร์ ไปยังขั้วลบของมอเตอร์ จึงทำให้มอเตอร์สามารถหมุนได้ ในทิศทาง Forward (จะหมุนแบบตามเข็มนาฬิกา หรือทวนเข็มนาฬิกานั้น ขึ้นอยู่กับลักษณะของการพันขดลวดภายในมอเตอร์)



รูปที่ 2.11 H-Bridge Switching

และในทางกลับกัน ถ้าหากเราทำการ On สวิทช์ S2 และ S4 พร้อมกัน (รูปล่าง) ก็จะเป็นการเชื่อมวงจร และทำให้เกิดกระแสไฟฟ้า ไหลผ่านมอเตอร์ จากขั้วลบของมอเตอร์ ไปยังขั้วบวกของมอเตอร์ จึงทำให้มอเตอร์สามารถหมุนได้ และเป็นการหมุนในทิศทาง Reverse (กลับทิศทางกับกรณีแรก)

วงจรนี้จะอาศัยสวิทช์ 4 ตัว เพื่อบังคับทิศทางการไหล ของกระแสไฟฟ้า ที่ไหลผ่านมอเตอร์ เพื่อควบคุมให้มอเตอร์หมุนตามทิศทางที่เราต้องการ โดยการผลัดกัน On และ Off สวิทช์พร้อมกัน 2 ตัวนั่นเอง

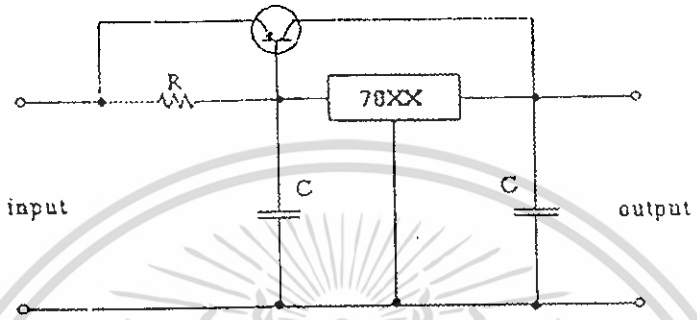
2.10 โมดูล RS232 to RF-Wireless (RF2.4GHz) CONVERTER รุ่น ET-RF24G V1.0

ลักษณะโดยทั่วไป

ET-RF24G V1.0 เป็นชุด Signal Converter สำหรับใช้แปลงสัญญาณระหว่าง RS232 และ RF-Wirelessโดยในโหมดการทำงานของการส่งข้อมูล (Transmitter) จะทำหน้าที่รอรับข้อมูลจากพอร์ตสื่อสารอนุกรม RS232จากขา RX แล้วแปลงเป็นสัญญาณความถี่ (GFSK) ส่งออกไปในอากาศ และในทางกลับกันในโหมดการทำงานแบบรับ (Receiver) โมดูล ET-RF24G ก็จะทำหน้าที่คอยตรวจจับข้อมูลที่อยู่ในรูปของสัญญาณความถี่ (GFSK)จากด้าน RF เพื่อแปลงกลับเป็นข้อมูลแบบ RS232 ส่งออกไปทางขา TX ด้วย ซึ่งเราได้นำเอาชุดแปลงสัญญาณ ET-RF24G V1.0 ไปต่อใช้งานร่วมกับพอร์ตสื่อสารอนุกรมแบบ RS232 เพื่อใช้งานในลักษณะของการสื่อสารอนุกรมแบบไร้สาย(Wireless Transceiver) ได้ ซึ่งระบบการจัดการข้อมูลของเครื่อง ET-RF24G V1.0 นั้น มีระบบการเข้ารหัสและถอดรหัสข้อมูลที่มีความน่าเชื่อถืออยู่ในเกณฑ์ที่จัดว่าดี โดยข้อมูลแต่ละ Byte ที่มีการรับส่งกันนั้น จะมีการตรวจสอบความถูกต้องของข้อมูลให้ด้วยแล้ว โดยข้อมูลที่รับได้จากด้าน RF นั้นเป็นข้อมูลที่มีความถูกต้องแน่นอน แต่อย่างไรก็ตามการรับส่งข้อมูลนั้นมีโอกาสผิดพลาดในเรื่องของการสูญหายของข้อมูลบ้างเหมือนกัน เนื่องจากกลไกในการรับส่งข้อมูลของเครื่อง ET-RF24G V1.0 นั้น จะมีการตรวจสอบข้อมูลทุก Byte ที่รับได้จาก RF เสมอ ซึ่งถ้าพบว่ามีความผิดพลาดเกิดขึ้นจะทิ้ง ข้อมูล Byte นั้น

2.11 วงจรรักษาแรงดันใช้ไอซีรักษาแรงดัน(IC Regulator)

ข้อดีของวงจรแบบนี้คือ ราคาถูก มีขนาดเล็ก และรูปแบบวงจรที่ง่าย สามารถจ่ายกระแสเอาท์พุทได้ตั้งแต่ 3 mA ถึง 100 mA ตามเบอร์ที่เราเลือกใช้ ยังมีวงจรป้องกันกระแสเกินภายในและวงจรป้องกันอุณหภูมิเกินภายในด้วย โดย IC เบอร์ต่างๆ จะมีคุณสมบัติด้านกระแสเอาท์พุทสูงสุดแรงดันอินพุทไลน์เรคกูเลชั่น โหลดเรคชัน และช่วงอุณหภูมิที่ทำงานให้เราเลือกตามต้องการวงจรรักษาแรงดับแรงดันแบบ IC

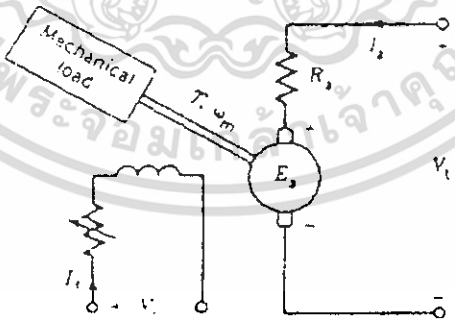


รูปที่ 2.12 วงจรรักษาแรงดันแบบแรงดันคงที่ใช้ MC 78xx

2.12 DC Motor

โครงสร้างของมอเตอร์กระแสตรง (DC Motor Structure)

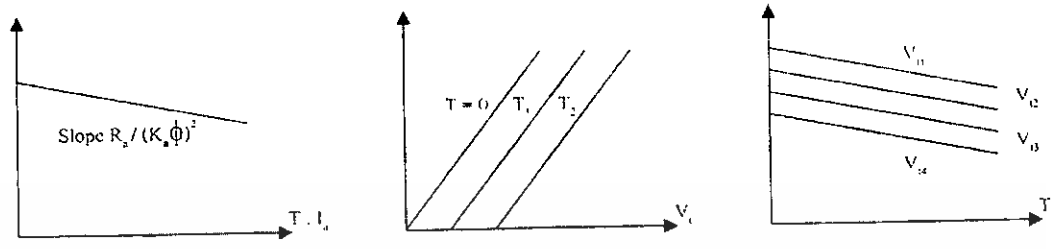
โครงสร้างของมอเตอร์กระแสตรงโดยพื้นฐานจะประกอบด้วยขดลวดสร้างสนามแม่เหล็ก(Field Winding) และขดลวด Armature โดยขดสร้างสนามแม่เหล็กที่ตัว Stator และขด Armature จะอยู่ที่ตัว Rotor ของตัวมอเตอร์ ดังรูปที่ 2.13



รูปที่ 2.13 โครงสร้างพื้นฐานของมอเตอร์กระแสตรง

แต่โครงสร้างของมอเตอร์ที่ใช้จริงในโครงงานนี้ ขดลวดที่สร้างสนามจะถูกเปลี่ยนเป็นแม่เหล็กถาวร โดยจะสร้างสนามแม่เหล็กที่มีค่าคงที่ตลอดเวลา ดังนั้นการควบคุมมอเตอร์ชนิดนี้จึงสามารถควบคุม

ค่าแรงดัน(V_t) และกระแส (I_a) ได้เท่านั้น โดยความเร็วของมอเตอร์สามารถควบคุมได้จากแรงดัน (V_t) ของมอเตอร์ โดยความสัมพันธ์ระหว่างความเร็วรอบ(ω_m), ค่าแรงดัน (V_t), กระแส (I_a) แสดงดังรูปที่ 2.14



รูปที่ 2.14 กราฟความสัมพันธ์ระหว่าง ω_m, I_a และ T

ซึ่งความสัมพันธ์ระหว่าง ω_m, I_a และ T สามารถเขียนได้คือ

$$E_a = K_a\phi\omega_m = V_t - I_aR_a$$

$$T = K_a\phi I_a$$

$$\omega_m = (\mathrm{Vt} - \mathrm{I}_a R_a)/K_a\phi = [V_t/Ka\phi] - [R_aT/(K_a\phi^2]$$

โดยที่

E_a คือ แรงดันเหนี่ยวนำย้อนกลับ (Back Electromotive Force (emf)

K_a คือ ค่าคงที่ขึ้นอยู่กับโครงสร้างแต่ละตัว

ϕ คือ เส้นแรงแม่เหล็กที่เกิดจากสนามแม่เหล็ก

ω_m คือ ความเร็วรอบของมอเตอร์

V_t คือ กระแสที่ไหลเข้ามอเตอร์

I_a คือ แรงดันที่ขั้วมอเตอร์

R_a คือความต้านทานของขดลวด armature

T คือ ค่าแรงบิดของมอเตอร์

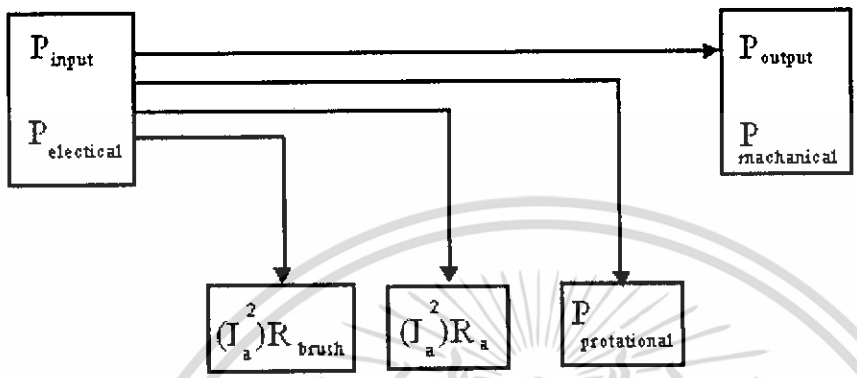
ซึ่งจะเห็นได้ชัดว่า ถ้าแรงดันที่ขั้วมอเตอร์(Vt) เพิ่มขึ้น จะทำให้ความเร็วเพิ่มขึ้นและถ้าหากโหลดเพิ่มขึ้น(แรงบิดเพิ่มขึ้น) มอเตอร์จะกินกระแสมากขึ้น ความเร็วมอเตอร์จะลดลง

ประสิทธิภาพของมอเตอร์ที่ใช้ สามารถวัดได้จากกำลังทางกลที่ได้รับ และกินกำลังงานของไฟฟ้าที่ให้โดย

กำลังกลที่ได้รับ (P_m): $P_m = T\omega_m$

กำลังไฟฟ้าที่ให้ (P_e): V_tI_a

การสูญเสียที่เกิดขึ้น จะเกิดขึ้นที่ความต้านทานของขด Armature ซึ่งมีค่าเท่ากับ $(I_a)^2 R_a$ และการสูญเสียที่เกิดจากความต้านทานการแปลงถ่าน(Brush) ซึ่งเท่ากับ $(I_a)^2 R_a$ และการสูญเสียที่เกิดจากแรงเสียดทานที่ตัว Bearing(Protational) และสามารถแสดงได้ดังรูปที่ 2.15



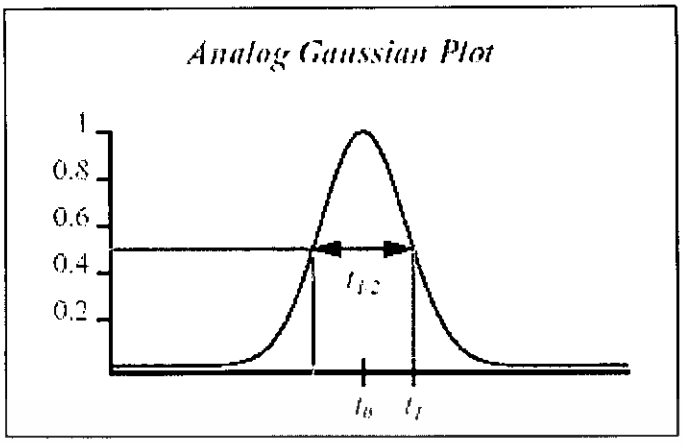
รูปที่ 2.15 กำลังงานที่สูญเสียในมอเตอร์กระแสตรง

ประสิทธิภาพของมอเตอร์คือ

$$\text{Efficiency} = (P_{out}/P_{in}) \times 100\%$$
$$= (P_e) \times 100\%$$
$$= [(V_t I_a - \text{Losses}) / V_t I_a] \times 100\ \%$$

2.13 GFSK (Gaussian Frequency Shift Keying)

การมอดูเลชั่น แบบGFSK เหมือน FSK ทุกอย่าง แต่จะทำการเพิ่มวงจร pulse shaper เพื่อลดเหลี่ยมของ pulse ข้อมูล ก่อน modulate เท่านั้น ที่ทำเช่นนี้ก็เพราะ ถ้าข้อมูลเป็นเหลี่ยม เมื่อมอดูเลชั่น แล้วจะเกิดแถบความถี่ที่กว้าง(bandwidth กว้าง) ทำให้เปลืองแถบความถี่ ที่ใช้งาน การ modulation แบบ FSK ก็คือ ในขณะที่ข้อมูลเปลี่ยน จาก 0 เป็น 1 หรือ 1 เป็น 0 จะเกิดการเปลี่ยนเฟสของ carrier อย่างรวดเร็ว อาจจะสูงขึ้นหรือต่ำลง ซึ่งก็มีผลให้ความถี่ carrier จริงสูงกว่า หรือ ต่ำกว่า f0 หรือ f1 ที่กำหนดไว้ นั่นก็คือ bandwidth จะกว้างขึ้น ดังนั้นเขาจึงหาวิธีลดปัญหาดังกล่าว โดยให้การเปลี่ยนแปลงของสัญญาณข้อมูลเป็นแบบค่อยๆ ขึ้น หรือ ค่อยๆ ลง โดยมีความโค้งเป็นแบบ Gaussian pulse ซึ่งเป็นแบบที่ได้คัดเลือกแล้วว่า ทำให้มี แบนวิดแคบใกล้เคียงที่ต้องการ และได้ symbol rate สูง

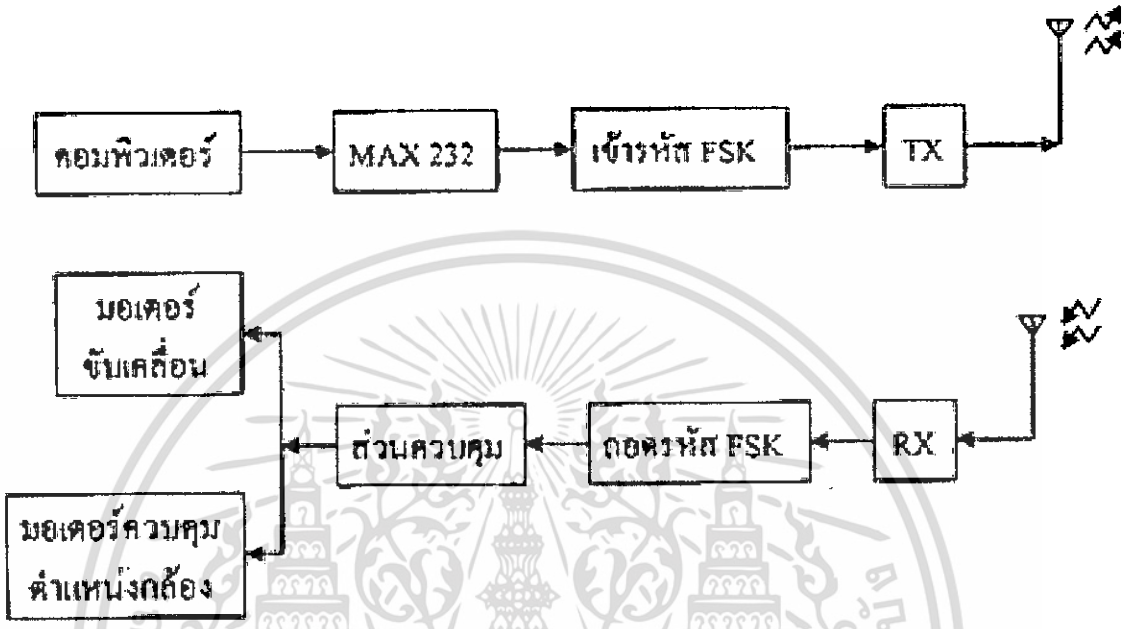


รูปที่ 2.16 Analog Gaussian Plot

บทที่ 3

การคำนวณและ การสร้าง

ภาพรวมของวงจรพร้อมทั้งอธิบายการทำงาน



รูปที่ 3.1 ภาพรวมของวงจร

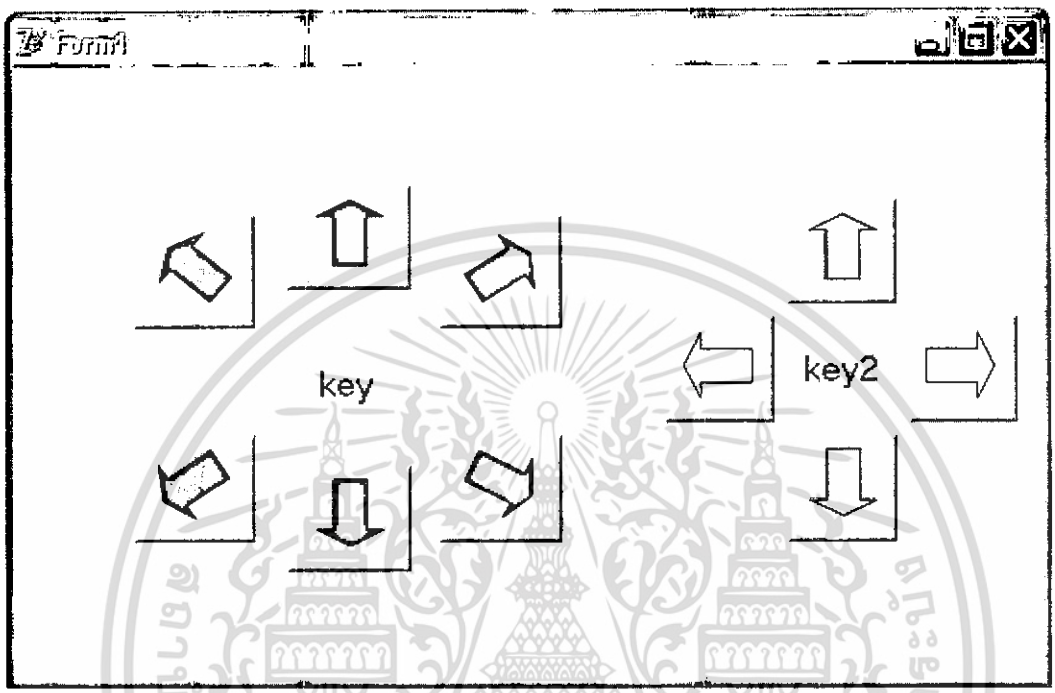
สำหรับภาพรวมของโครงงานทั้งระบบ สามารถอธิบายคร่าวๆ ได้ดังนี้ เริ่มจากคอมพิวเตอร์ทำการส่งข้อมูล ASCII ส่งออก RS -232 แล้วทำการเข้ารหัส แล้วส่งข้อมูลไปยังโมดูล TX เพื่อทำการส่งข้อมูลไปยังด้านรับโดยมีวิธีการส่งแบบ GFSK ไปยังโมดูล ด้านรับ เมื่อด้านรับ RX ได้รับข้อมูลก็จะทำการส่ง DATA ต่อไปยัง ไมโคร คอนโทรลเลอร์เพื่อประมวลผล และสั่งงานให้วงจรไฮบริดของมอเตอร์ขับเคลื่อน รถ และมอเตอร์กล้องทำงาน

โดย รายละเอียดการทำงานในแต่ละส่วนสามารถอธิบายการทำงานแบบละเอียดได้ดังนี้

3.1 การออกแบบและการทำงานในการควบคุมและส่งคำสั่งโดยใช้ภาษาเดลไฟ

การออกแบบและการทำงานในการควบคุมและส่งคำสั่ง โดยใช้ภาษาเดลไฟ

1.กำหนดฟอร์มและออพเจ็กต่างๆ โดยเลือกปุ่มกดแบบ Click และ mousedown และทำการออกแบบ ได้ผลตามที่ต้องการดังรูปที่ 3.2



รูปที่ 3.2 รูปแบบฟอร์มควบคุมรถและกล้องของ โปรแกรมเดลไฟล์

2.เขียนคำสั่งให้ พอร์ต COM2 ทำงาน เมื่อรันโปรแกรมและเรียกฟอร์ม และ สั่งให้หยุดการทำงานเมื่อปิดฟอร์ม

3.ทำการเซตค่า component ที่ทำการเพิ่มเข้ามา โดย component จะทำหน้าที่ ติดต่อสื่อสารระหว่างโปรแกรม เดลไฟ และส่งข้อมูลออก ทาง COM1 ได้

4.ทำการเซต แต่ละไอคอน ในฟอร์ม ให้มีค่า ดังนี้

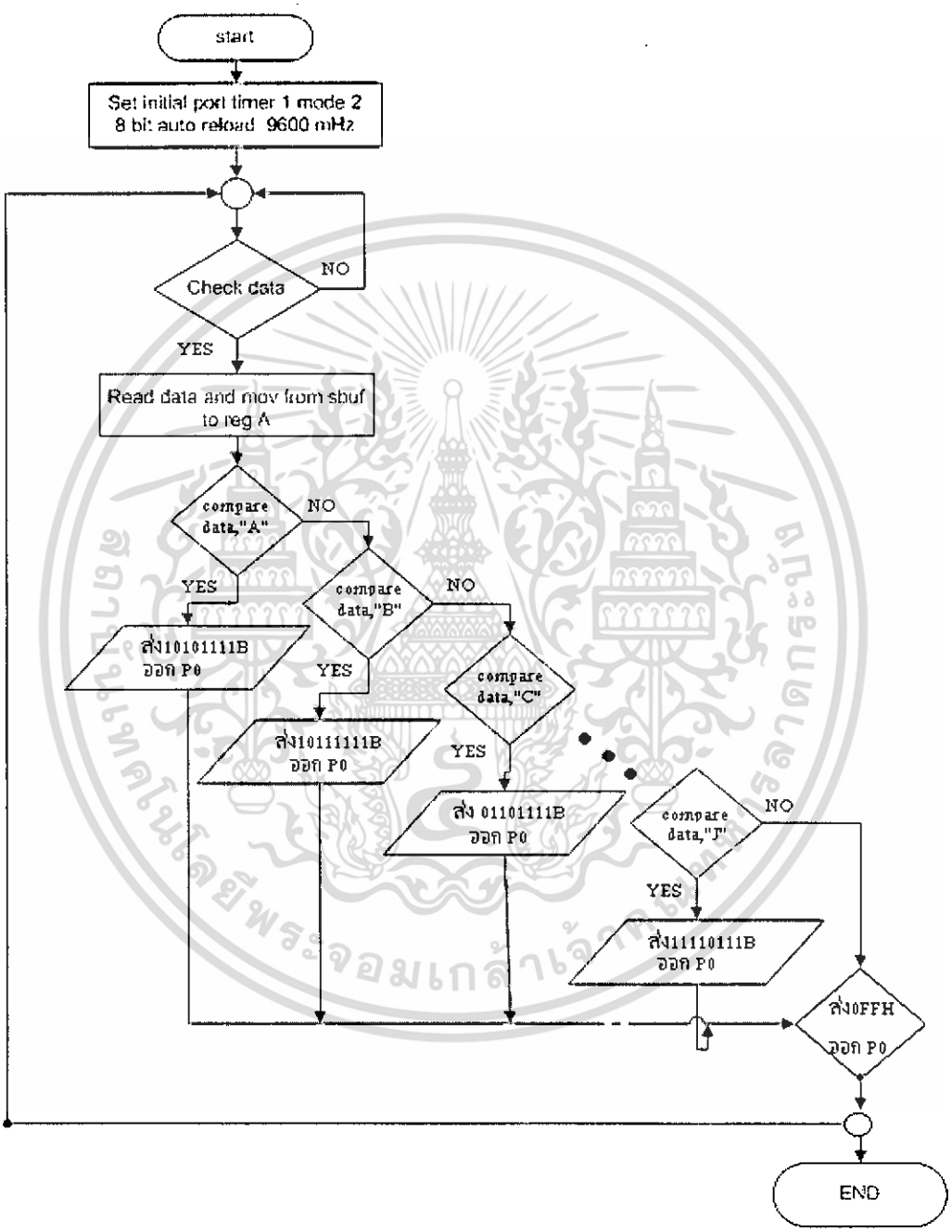
ตารางที่ 2.5 แสดงค่าการเซตไอคอน

ค่า ASCII (ซึ่งจะเป็นข้อมูลที่ทำการส่งออกเพื่อไปประมวลผลในขั้นตอนต่อไป)	**ทิศทาง ของรถ หรือกล้อง**
A	รถเลี้ยวซ้าย และเดินหน้า
B	รถเดินหน้า
C	รถเลี้ยวขวา และเดินหน้า
D	รถเลี้ยวซ้าย และถอยหลัง
E	รถถอยหลัง
F	รถเลี้ยวขวา และถอยหลัง
G	กล้อง เลื่อนขึ้น
H	กล้องเลื่อนลง
I	กล้องหมุนไปทางซ้าย
J	กล้องหมุนไปทางขวา

5.สั่งให้ปุ่มทุกปุ่มที่ ออกแบบไว้ รวมถึงแป้นพิมพ์ ของคอมพิวเตอร์ ทำงาน โดย เมื่อกดค่า ต่างๆที่แป้นพิมพ์หรือที่ตัวโปรแกรมเคลไฟโปรแกรมก็จะทำการส่งข้อมูลออกไปยัง COM1 เพื่อนำไปประมวลผลในขั้นตอนต่อไป

3.2 การออกแบบการควบคุมการทำงาน ของ MCS-51

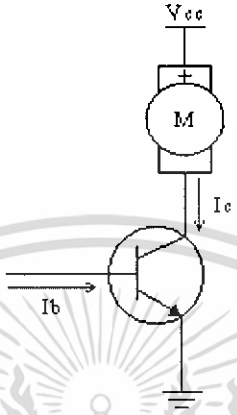
ได้ทำการออกแบบโปรแกรม เพื่อที่จะให้ ไมโครคอลโทรลเลอร์ ทำการควบคุม วงจรไฮบริด ของมอเตอร์ขับเคลื่อนรถสำรวจ และมอเตอร์ควบคุมกล้อง โดยมีการออกแบบโปรแกรมแสดงเป็น แผนภาพการทำงาน (flowchart) ได้ดังรูปที่ 3.3 ดังนี้



รูปที่ 3.3 การออกแบบการควบคุมการทำงาน ของ MCS-51

3.3 อธิบายการทำงาน ของ วงจร H-Bridge Switching ควบคุมมอเตอร์รถ

จากรูปที่ 3.4 เป็นวงจรสาธิตแบบง่าย ๆ โดยการนำทรานซิสเตอร์ มาเป็นสวิทช์ควบคุมมอเตอร์ หลักการคิดง่าย ๆ ก็คือ เมื่อเราป้อนกระแส Ib ด้วยปริมาณที่มากพอ ก็จะทำให้ทรานซิสเตอร์ทำงาน (On) จะทำให้กระแส Ic ไหล แปลว่ามีกระแสไหลผ่านมอเตอร์ได้ (กระแส Ib จะต้องมากเพียงพอที่จะทำให้ทรานซิสเตอร์อยู่ในสภาวะ "อิ่มตัว" ได้)

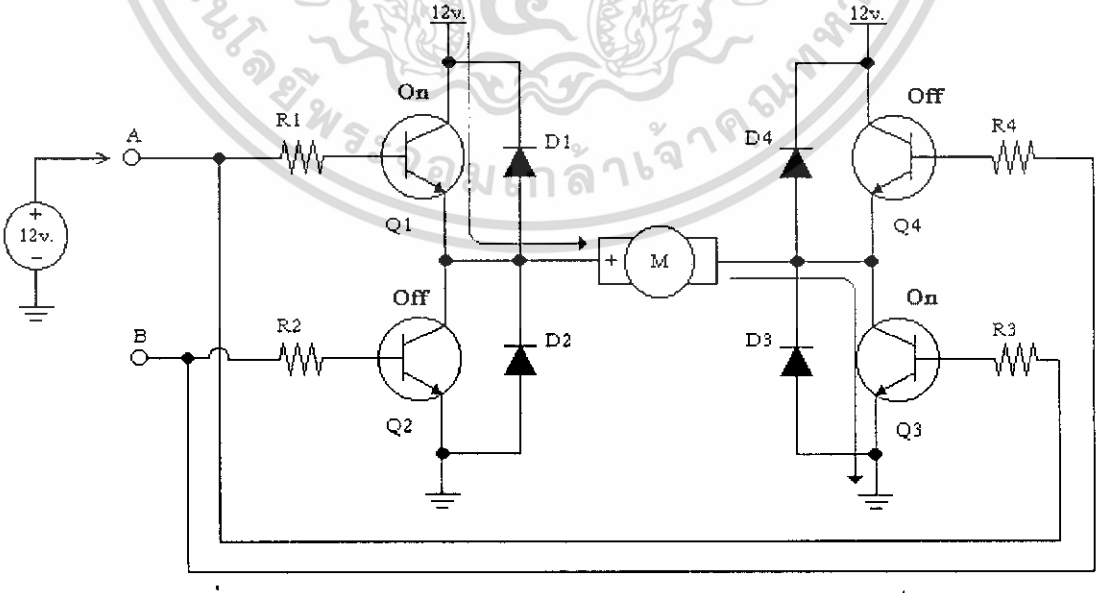


รูปที่ 3.4 การทำงาน ของ วงจร H-Bridge Switching ควบคุมมอเตอร์รถ

ในสภาวะ อิ่มตัว (Saturation mode) นี้ ทรานซิสเตอร์จะทำงานเหมือนกับสวิทช์ปิดวงจร ค่าความต้านทานระหว่างขา C และขา E จะมีค่าเข้าใกล้ศูนย์ กระแส Ic ที่ไหลจะมีค่าเข้าใกล้ Ic(max)

ในสภาวะ คัตออฟ (Cutoff mode) นี้ จะเกิดขึ้นเมื่อเราหยุดจ่ายกระแส Ib (Ib = 0) ทรานซิสเตอร์จะทำงานเหมือนกับสวิทช์เปิดวงจร ค่าความต้านทานระหว่างขา C และขา E จะมีค่าเป็นอนันต์ กระแส Ic จะมีค่าเข้าใกล้ศูนย์

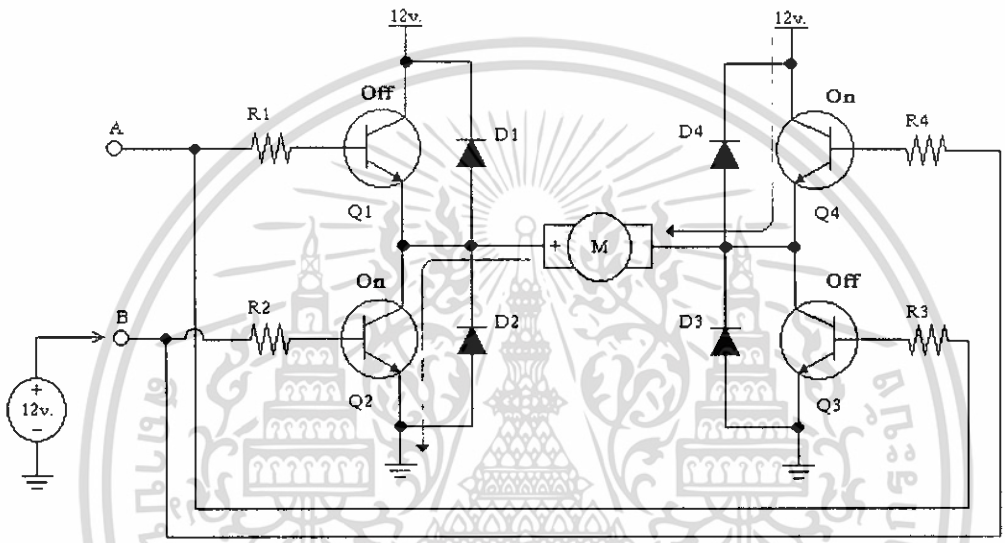
กรณีที่ Q1 และ Q3 ทำงาน



รูปที่ 3.5 การทำงาน ของ วงจร H-Bridge Switching ควบคุมมอเตอร์รถ

เมื่อมีการจ่ายแรงดัน 12v. เข้าที่จุด A ทำให้มีกระแสไหลผ่าน R1 เข้าสู่ขา base ของ Q1 และมีกระแสไหลผ่าน R3 เข้าสู่ขา base ของ Q3 ทำให้ Q1 และ Q3 ทำงาน (On) เปรียบเสมือนสวิทช์ปิดวงจร ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (12v.) ผ่านขา Collecter และEmitter ของ Q1 ผ่านเข้าสู่ขั้วบวก (+) ของมอเตอร์ ผ่านไปยังขา Collecter และEmitter ของ Q3 ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางบวก และครบวงจร จึงทำให้มอเตอร์สามารถหมุน ในทิศทาง Forward ได้

กรณีที่ Q2 และ Q4 ทำงาน



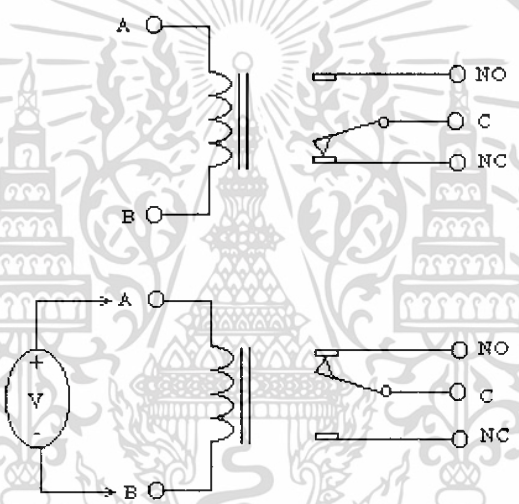
รูปที่ 3.6 การทำงาน ของ วงจร H-Bridge Switching ควบคุมมอเตอร์รถ

เมื่อมีการจ่ายแรงดัน 12v. เข้าที่จุด B ทำให้มีกระแสไหลผ่าน R2 เข้าสู่ขา base ของ Q2 และมีกระแสไหลผ่าน R4 เข้าสู่ขา base ของ Q4 ทำให้ Q2 และ Q4 ทำงาน (On) เปรียบเสมือนสวิทช์ปิดวงจร ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (12v.) ผ่านขา Collecter และEmitter ของ Q4 ผ่านเข้าสู่ขั้วลบ (-) ของมอเตอร์ ผ่านไปยังขา Collecter และEmitter ของ Q2 ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางลบ และครบวงจร จึงทำให้มอเตอร์สามารถหมุน ในทิศทาง Reward ได้

3.4 หลักการทำงาน วงจร H-Bridge Switching จาก Relay ที่นำมาประยุกต์ใช้ควบคุมกล้อง

ภายในโครงสร้างของ รีเลย์ จะประกอบไปด้วยขดลวด (Coil) 1 ชุด และ หน้าสัมผัส (Contactor) ซึ่งในหน้าสัมผัส 1 ชุด จะประกอบไปด้วย

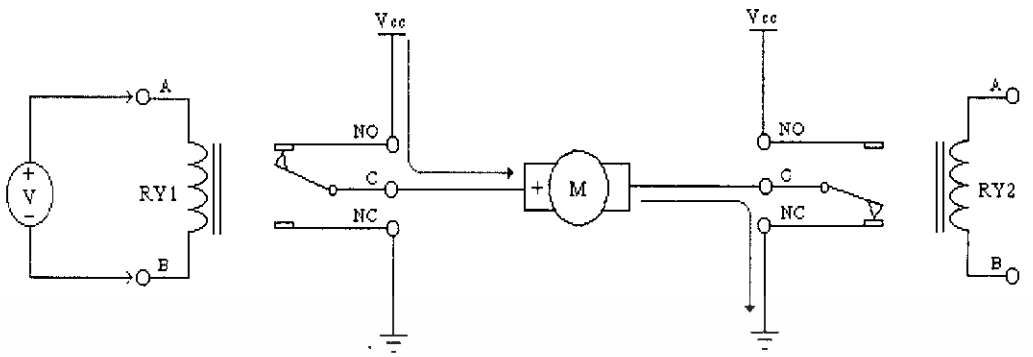
- หน้าสัมผัสแบบปกติปิด (Normally Close หรือ NC.) ซึ่งในสภาวะปกติ ขานี้จะต่ออยู่กับขาร่วม (Common)
- หน้าสัมผัสแบบปกติเปิด (Normally Open หรือ NO.) ขานี้จะต่อเข้ากับขาร่วม (Common) เมื่อขดลวดมีแรงดันตกคร่อม หรือกระแสไหลผ่าน (ในปริมาณที่เพียงพอ) ใน รีเลย์ 1 ตัว อาจมีหน้าสัมผัสมากกว่า 1 ชุด เช่น 2 ชุด, 4 ชุด เป็นต้น เมื่อขดลวดได้รับแรงดันตกคร่อม (ขา A และ B) จะทำให้มีกระแสไหลผ่านขดลวด ซึ่งจะทำให้เกิดอำนาจสนามแม่เหล็ก ดึงดูดให้หน้าสัมผัส NO และ C ติดกัน.



รูปที่ 3.7 การทำงาน วงจร H-Bridge Switching จาก Relay ที่นำมาประยุกต์ใช้ควบคุมกล้อง

ภายในส่วนประกอบของวงจรที่ควบคุมมอเตอร์กล้องจะประกอบไปด้วย รีเลย์ 2 ตัว คือ RY1 และ RY2 ซึ่ง Load ก็คือ DC-Motor ซึ่งต่ออยู่กับขาร่วม (C.) ของ RY1. และ RY2. โดยขั้วบวก (+) ของมอเตอร์ ต่ออยู่ที่ขา C. ของ RY1 และขั้วลบ (-) ของมอเตอร์ ต่ออยู่ที่ขา C. ของ RY2 โดยที่ขา NO. ของ RY1 และ RY2 จะต่ออยู่กับขั้วบวก ของแหล่งจ่ายไฟ ที่จะจ่ายให้มอเตอร์ (Vcc) และขา NC. ของ RY1 และ RY2 จะต่อลงกราวด์

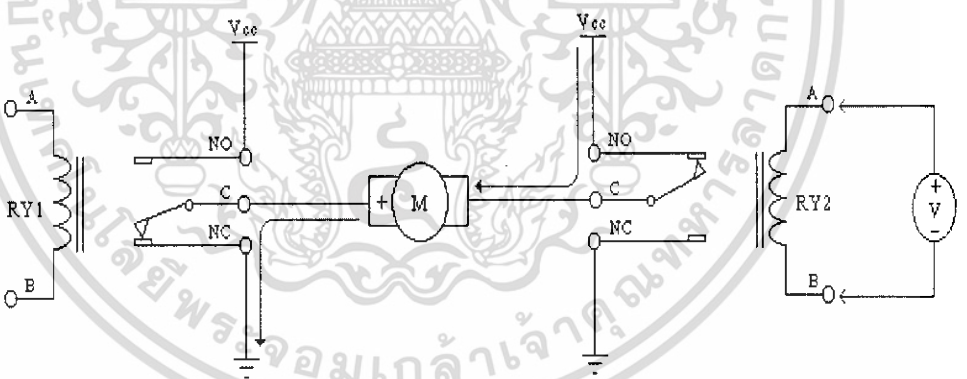
กรณีที่ RY1 ทำงาน



รูปที่ 3.8 การทำงาน วงจร H-Bridge Switching จาก Relay ที่นำมาประยุกต์ใช้ควบคุมกล้อง

เมื่อ RY1 ทำงาน (มีกระแสไหลผ่านขดลวดในปริมาณที่เพียงพอ) จะทำให้เกิดอำนาจสนามแม่เหล็กไฟฟ้าดึงดูดให้ขา NO และขา C ของ RY1 ติดกัน ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (Vcc) ผ่านเข้าสู่ขั้วบวก (+) ของมอเตอร์ ผ่านไปยังขา C ของ RY2 ซึ่งต่ออยู่ที่ NC และลงกราวด์ ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางบวก และครบวงจร จึงทำให้มอเตอร์สามารถหมุน ในทิศทาง Forward ได้

กรณีที่ RY2 ทำงาน

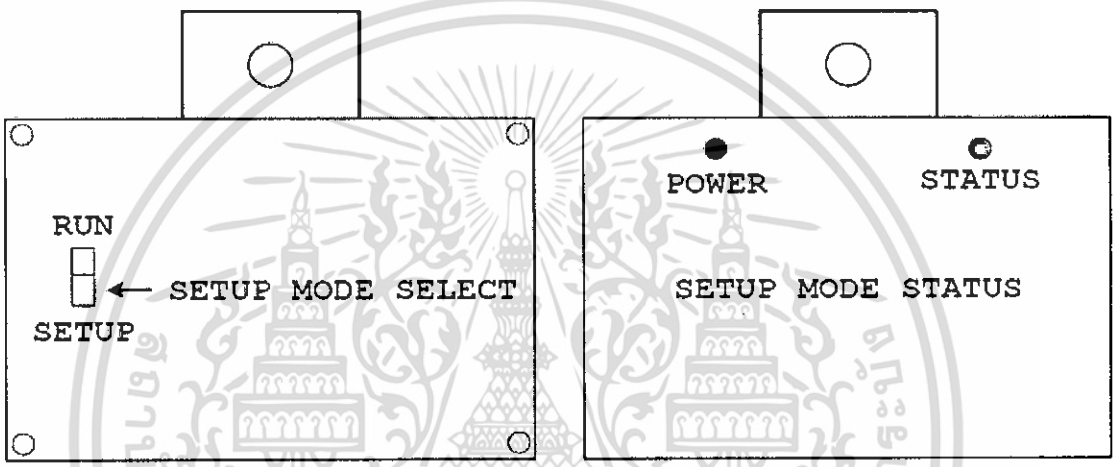


รูปที่ 3.9 การทำงาน วงจร H-Bridge Switching จาก Relay ที่นำมาประยุกต์ใช้ควบคุมกล้องกรณีที่ RY2 ทำงาน

เมื่อ RY2 ทำงาน (มีกระแสไหลผ่านขดลวดในปริมาณที่เพียงพอ) จะทำให้เกิดอำนาจสนามแม่เหล็กไฟฟ้า ดึงดูดให้ขา NO และขา C ของ RY2 ติดกัน ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (Vcc) ผ่านเข้าสู่ขั้วลบ (-) ของมอเตอร์ ผ่านไปยังขา C ของ RY1 ซึ่งต่ออยู่ที่ NC และลงกราวด์ ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางลบ และครบวงจร จึงทำให้มอเตอร์สามารถหมุน ในทิศทาง Reward ได้

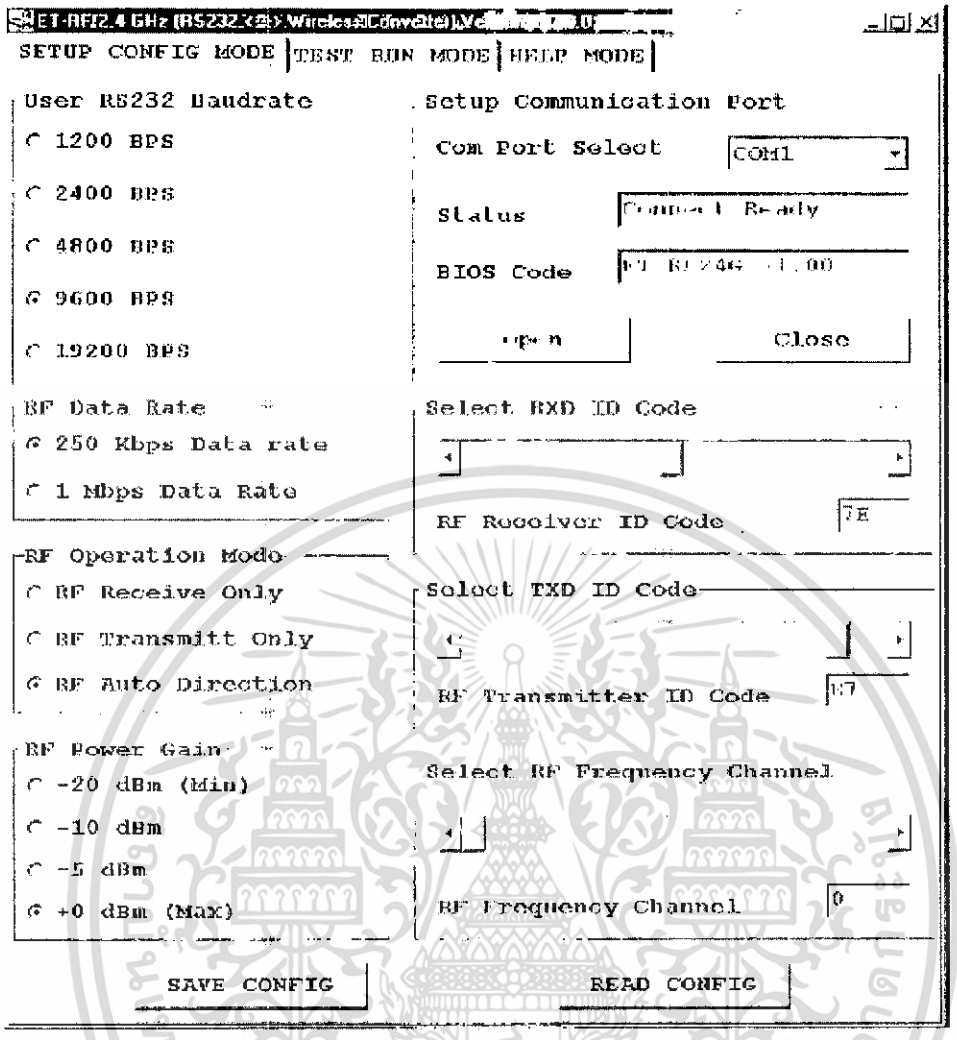
3.5 การใช้ติดตั้งและ กำหนดค่าต่างๆ ของเครื่อง ET-RF24G V1.0

การใช้งานเครื่อง ET-RF24G V1.0 ใน Setup Mode ซึ่งเป็นโหมดสำหรับใช้กำหนดค่า Configurationต่างๆ สำหรับควบคุมการทำงานของเครื่อง ET-RF24G V1.0 ที่จะใช้ในขณะที่เครื่องทำงานอยู่ใน Run Mode โดยในการ Setup ค่า Configuration ต่างๆนั้นจะกระทำร่วมกับโปรแกรม **"ET_RF24G_V1.EXE"** ของ อีทีที ซึ่งเมื่อเครื่อง ET-RF24G V1.0 เข้าทำงานในโหมด Setup แล้ว จะสังเกตุเห็นหลอดไฟแสดงสถานะการทำงาน หรือ LEDSTATUS ติดสว่างค้างอยู่ตลอดเวลา แต่เมื่อมีการสั่งอ่านหรือเขียนข้อมูลกับบอร์ด สถานะการทำงานของ LEDSTATUS จึงจะกระพริบตามจังหวะของการรับส่งข้อมูล แต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะติดค้างอยู่ตลอดเวลา



รูปที่ 3.10 การเลือกโหมดการทำงานสำหรับกำหนดค่า Configuration (Setup Mode)ของเครื่อง ET-RF24G V1.0

ซึ่งการกำหนดค่า Configuration ให้กับ ET-RF24G V1.0 นั้น จะต้องกระทำในขณะที่ตัวเครื่องทำงานอยู่ในSetup Modeเท่านั้น (เลือกSwitch กำหนดโหมดไว้ทางด้าน Setup แล้วจ่ายไฟให้เครื่องเริ่มต้นทำงาน) โดยค่าของConfigurationต่างๆนั้นจะถูกใช้สำหรับเป็นเงื่อนไขในการทำงานของ ET-RF24G V1.0 ในขณะที่อยู่ใน Run Modeโดยเมื่อทำการกำหนดค่าตัวเลือกต่างๆของConfiguration เรียบร้อยแล้ว ก็สามารถเปลี่ยนโหมดการทำงานของตัวเครื่องกลับเป็น Run Mode พร้อมกับการปิดไฟที่จ่ายให้กับตัวเครื่อง (Power-OFF) ชั่วขณะหนึ่ง จากนั้นจึงเริ่มต้นจ่ายไฟให้กับตัวเครื่องใหม่ (Power-ON) ก็สามารถใช้งาน ET-RF24G V1.0 ตามค่าของ Configuration ที่กำหนดไว้แล้วได้ โดยค่าตัวเลือกต่างๆของ Configuration ที่ได้กำหนดไว้แล้วจะถูกเก็บไว้ภายในตัวเครื่อง ตามเงื่อนไขที่กำหนดไว้ใน Configuration เสมอทุกๆครั้ง โดยคุณสมบัติของ Configuration ต่างๆ ที่ได้ทำการ กำหนด นั้นๆมีดังนี้



รูปที่ 3.11 รูปแสดง การเลือกโหมดการทำงาน สำหรับกำหนดค่า Configuration (Setup Mode)

• **User RS232 Baudrate** ใช้สำหรับกำหนดค่าความเร็วในการรับส่งข้อมูลทางด้าน RS232 ของตัวเครื่องในขณะที่ทำงานอยู่ใน Run Mode ซึ่งกำหนดให้ใช้งานที่ 9600 BPS

• **RF Data Rate** ใช้สำหรับกำหนดความเร็วในการรับส่งข้อมูลทางด้าน RF ของ ET-RF24G V1.0 โดยค่า RF Data Rate ได้กำหนดค่าไว้ คือ 250 Kbps

• **RF Operation Mode** ใช้สำหรับกำหนดโหมดการทำงานของ ET-RF24G V1.0 ซึ่งสามารถกำหนดหน้าที่การทำงานได้ 3 แบบ ด้วยกันแต่ที่ติดตั้งใช้งานสำหรับโหมดนี้นั้นใช้งานอยู่ 2 แบบ คือ

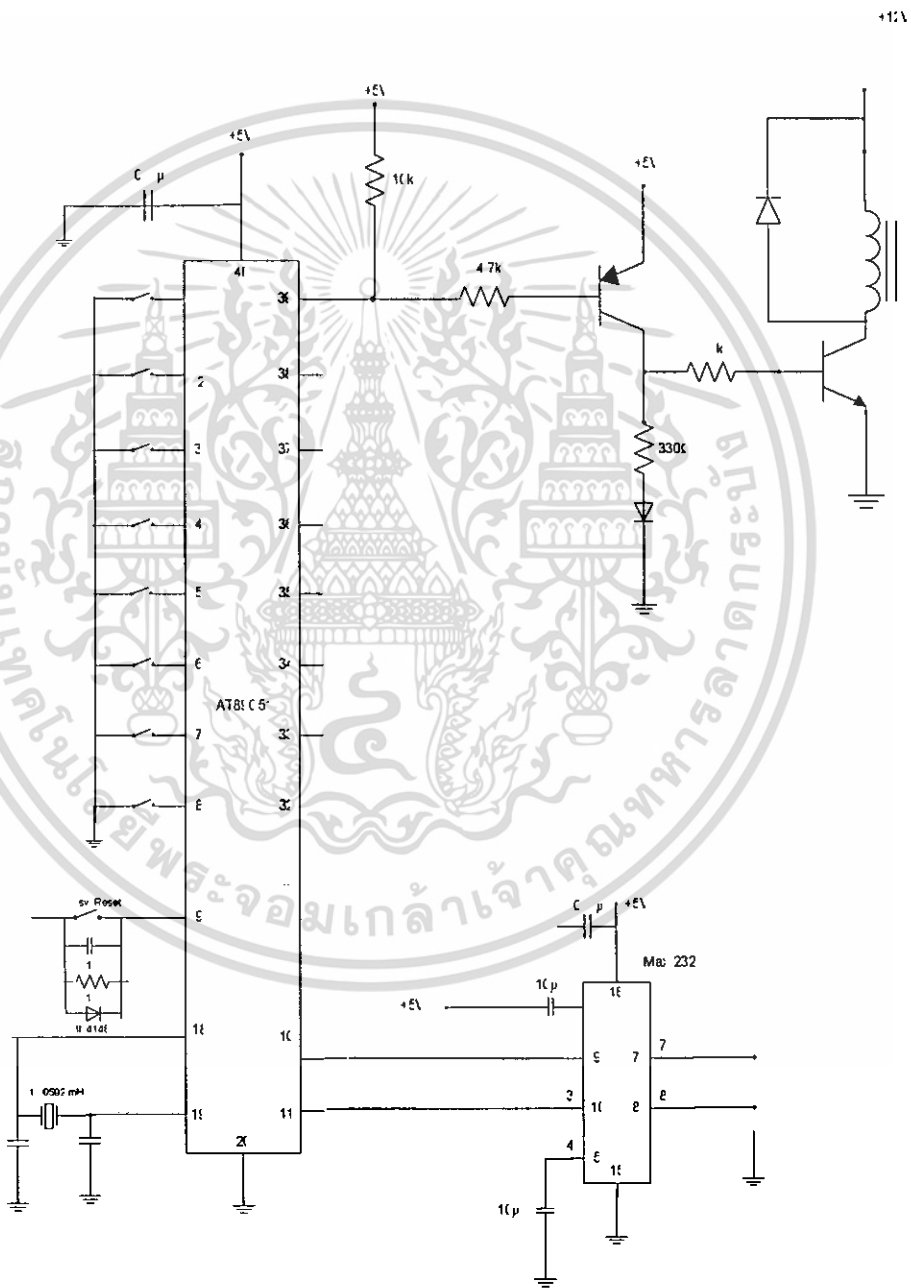
- RF Receive Only คือ การกำหนดให้ ET-RF24G V1.0 ทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RF เพื่อเปลี่ยนเป็นข้อมูลแบบ RS232 และส่งออกไปทางด้านขา TX ของ RS232 ตลอดเวลา

- RF Transmit Only คือ การกำหนดให้ ET-RF24G V1.0 ทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RS232 จากขา RX เพื่อเปลี่ยนเป็นข้อมูลแบบ GFSK และส่งออกไปทางด้าน RF ตลอดเวลา

• **RF Power Gain** คือการกำหนดกำลังส่งของวงจร RF Power ที่ใช้ในการส่งข้อมูล โดยตั้งค่าไว้ที่ +0dBm ซึ่ง เป็นค่ากำลังส่งสูงสุด

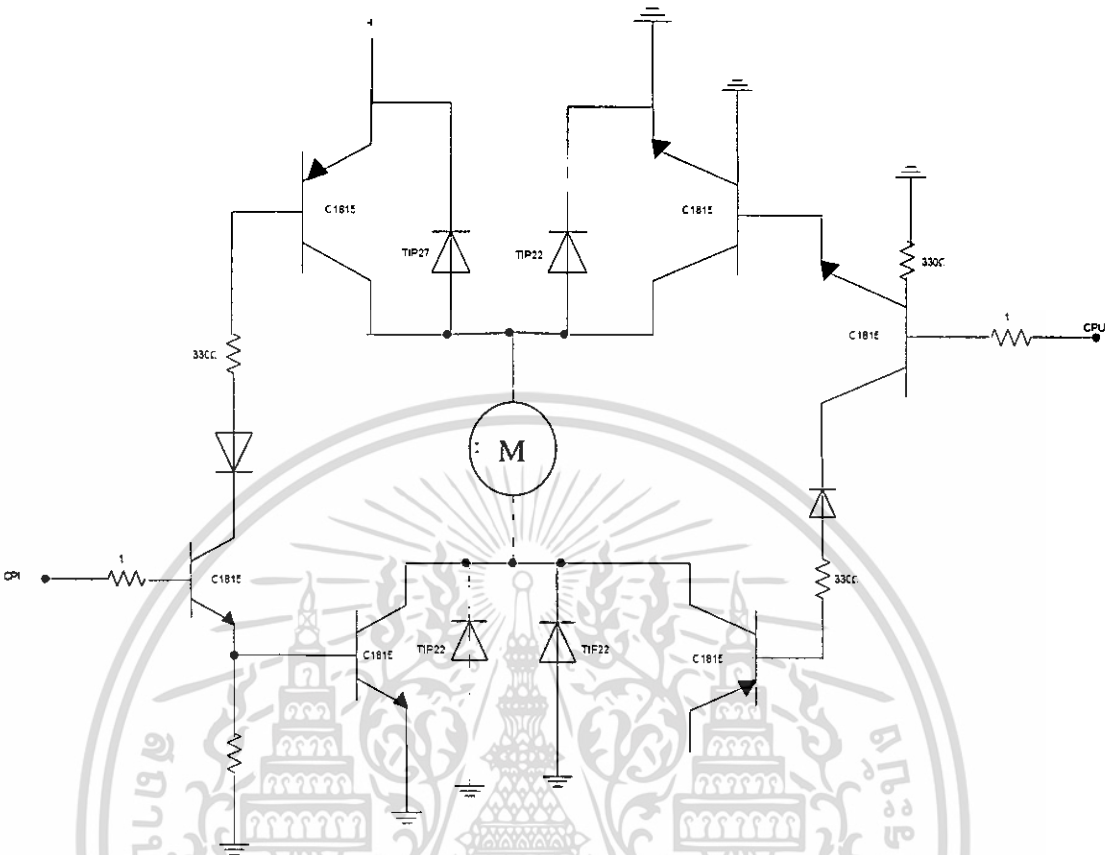
3.6 วงจรรวมของภาคควบคุมมอเตอร์ขับเคลื่อนรถและควบคุมกล้อง

ภาคควบคุมมอเตอร์ขับเคลื่อนรถและควบคุมกล้องหลังจากที่รับสัญญาณข้อมูลมาจาก RS-232 เพื่อที่จะให้ไมโครคอลโทรลเลอร์ทำงานในขั้นตอนต่อไป โดยมีวงจรรวมดังรูปที่ 3.12



รูปที่ 3.12 วงจรรวมของภาคควบคุมมอเตอร์ขับเคลื่อนรถและควบคุมกล้อง

และวงจร h-bridge ซึ่งมีหน้าที่ขับเคลื่อนมอเตอร์รถ ดังรูปที่ 3.13



รูปที่ 3.13 แสดง บล็อคไดอะแกรม พร้อมทั้งวงจร

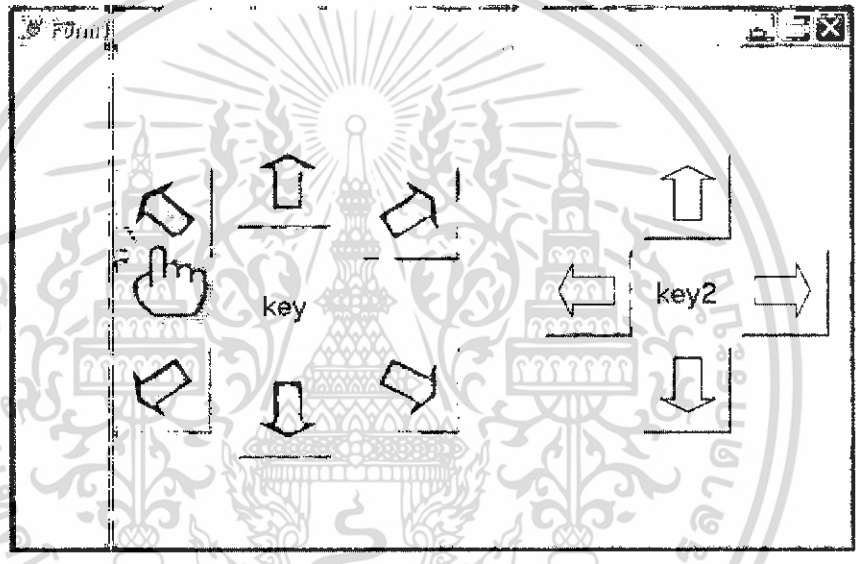
บทที่ 4

การทดลองและผลการทดลอง

4.1 ผลการทดลองของภาครับและภาคส่ง ของรถสำรวจ

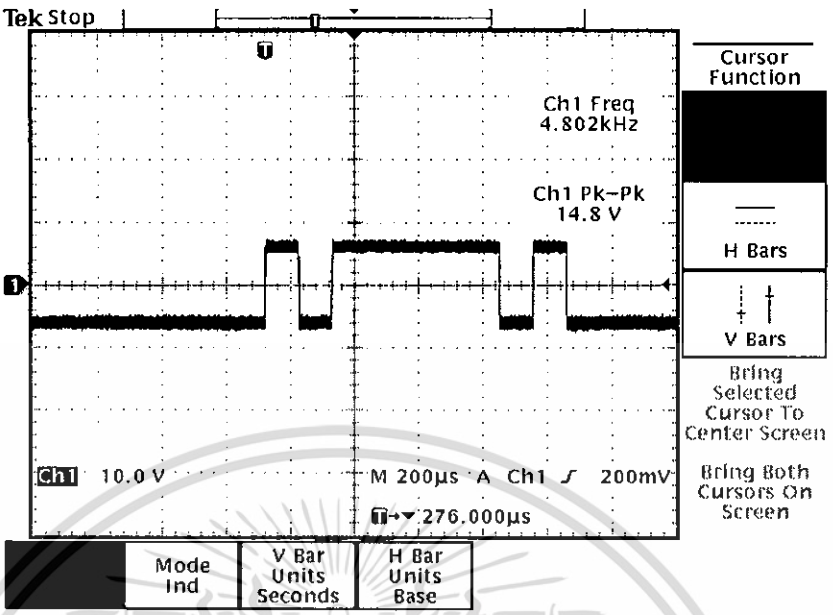
ทำการวัดสัญญาณข้อมูลที่ภาคส่งและภาครับสัญญาณ โดยในภาคส่งนั้นได้ทำการวัดสัญญาณที่ ขา RX ก่อนที่จะส่งสัญญาณข้อมูลต่อไปยังโมดูลส่งและหลังจากที่ทำการส่งข้อมูลจากคอมพิวเตอร์แล้วนั้น ได้ทำการวัดสัญญาณภาครับที่ขา 9 ของ RS -232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 เพื่อให้ MCS-51 ส่งสัญญาณไปควบคุมการเคลื่อนที่ของรถและกล้องต่อไป โดยมีรายละเอียดดังต่อไปนี้

4.1.1 กดปุ่มเดินหน้าและเลี้ยวซ้าย



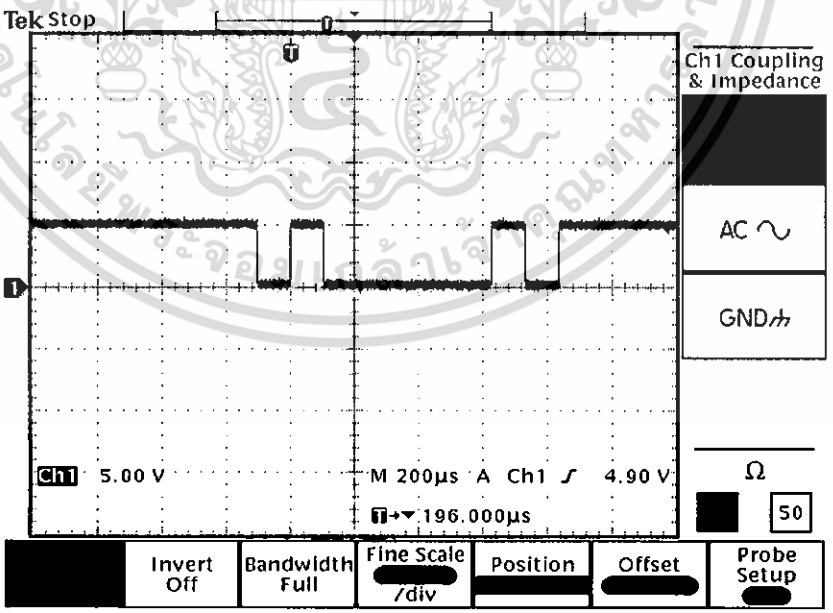
รูปที่ 4.1 ผลการทดลองภาครับของรถสำรวจเมื่อกดปุ่มเดินหน้าและเลี้ยวซ้าย

ในขณะที่กดปุ่มส่งข้อมูลเดินหน้าและเลี้ยวซ้ายค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่ง ข้อมูล "A" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "A" ที่ถูกส่งมานั้น เมื่อแปลงเป็น Binary Code จะมีค่า เท่ากับ 0100 0001 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณจะมีผลการทดลองดังนี้



รูปที่ 4.2 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้รถสำรวจเดินหน้าและเลี้ยวซ้าย

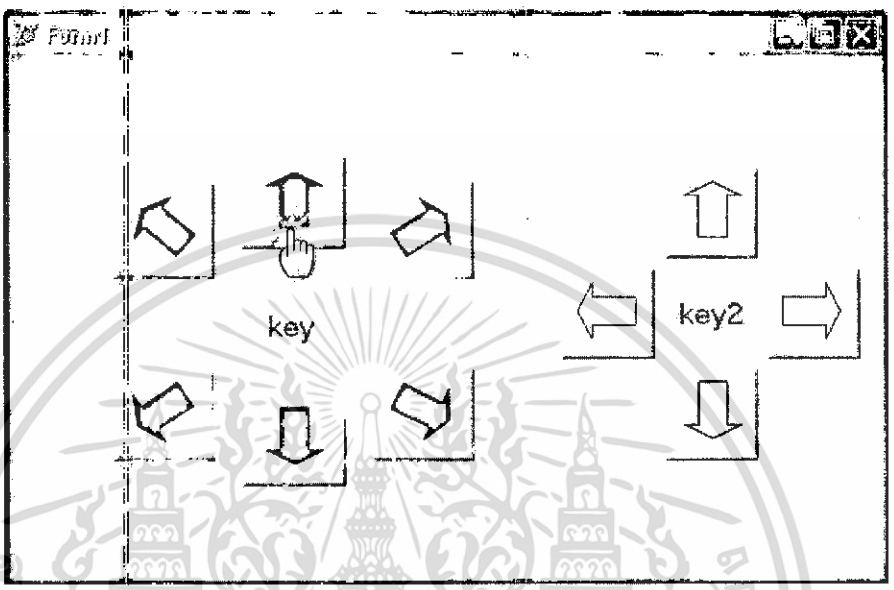
และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



รูปที่ 4.3 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้รถสำรวจเดินหน้าและเลี้ยวซ้าย

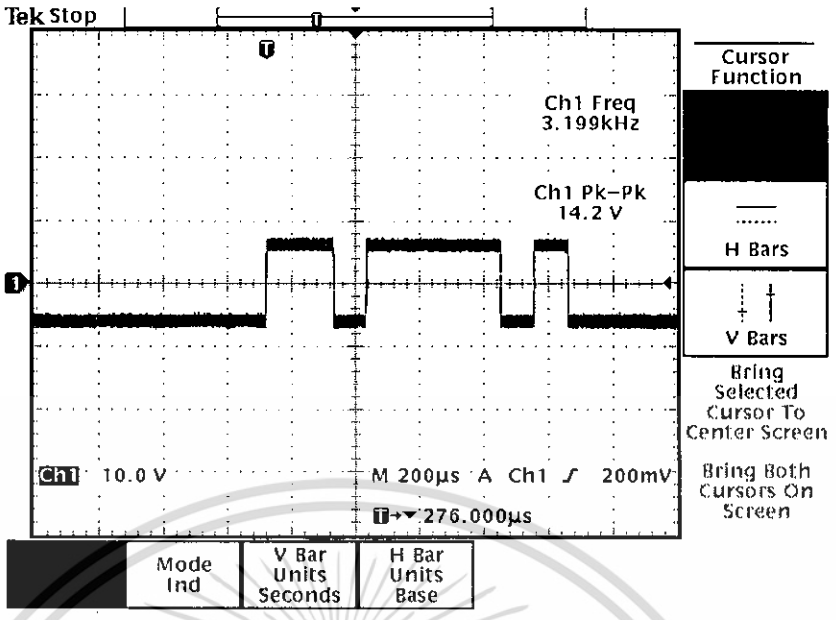
เมื่อนำผลการวัดสัญญาณของทั้งสองจุดจากภาครับและภาคส่งมาเปรียบเทียบกัน จะพบว่าสัญญาณที่ได้รับมานั้นมีความถูกต้อง สามารถส่งสัญญาณเพื่อเข้าสู่กระบวนการอื่นๆ ในขั้นตอนต่อๆไป ได้

4.1.2 กดปุ่มเดินหน้า



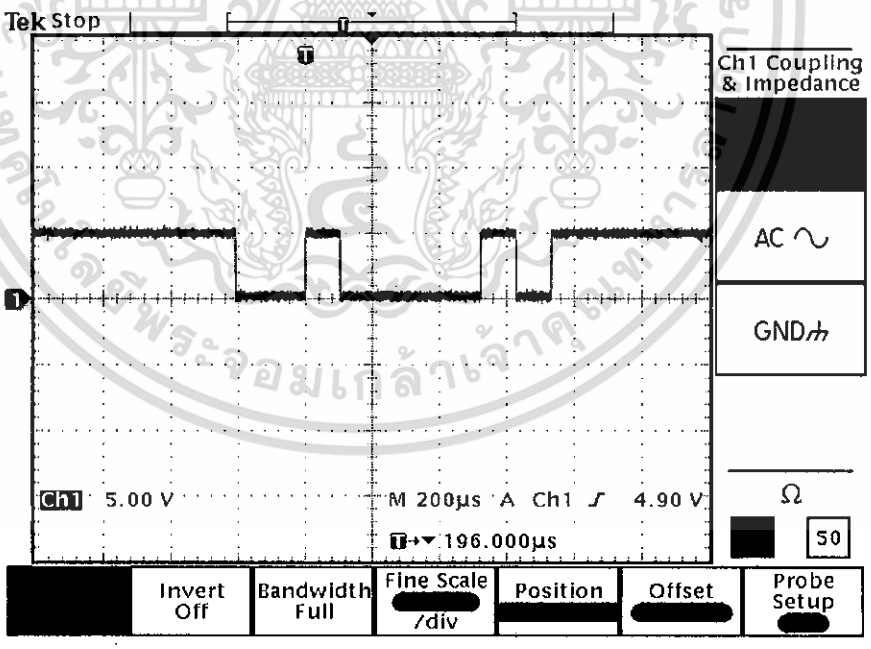
รูปที่ 4.4 ปุ่มควบคุมในขณะทำการสั่งให้รถสำรวจเดินหน้า

ในขณะที่กดปุ่มส่งข้อมูลเดินหน้าค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล "B" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "B" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 0010 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



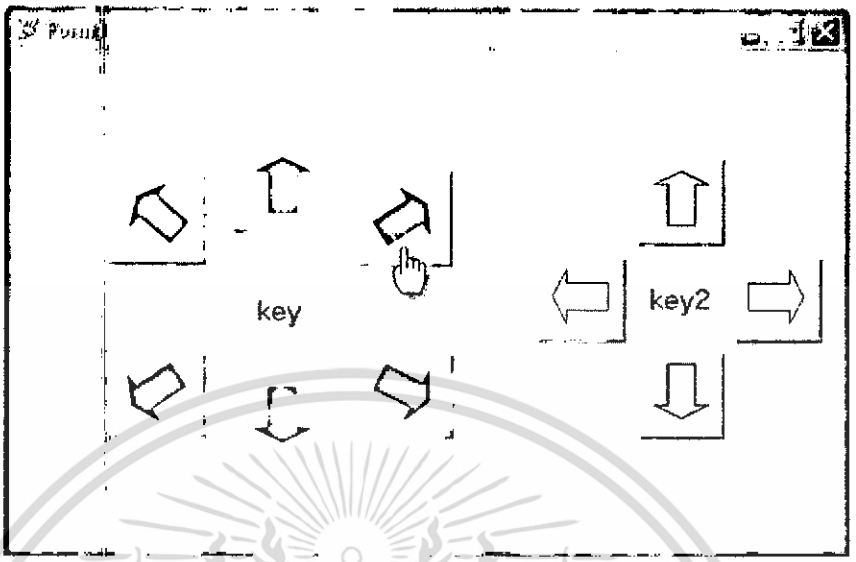
รูปที่ 4.5 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้รถสำรวจเดินหน้า

และภาครับจะรับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



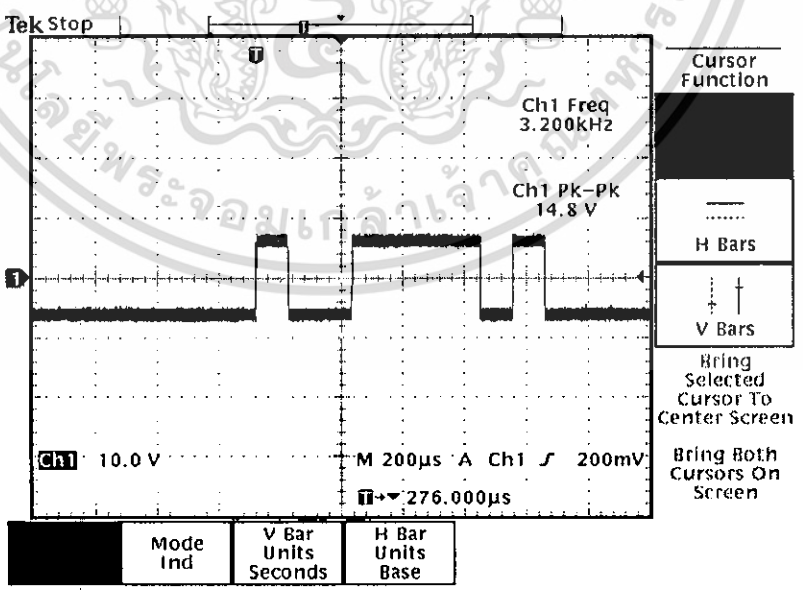
รูปที่ 4.6 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้รถสำรวจเดินหน้า

4.1.3 กดปุ่มเดินหน้าและเลี้ยวขวา



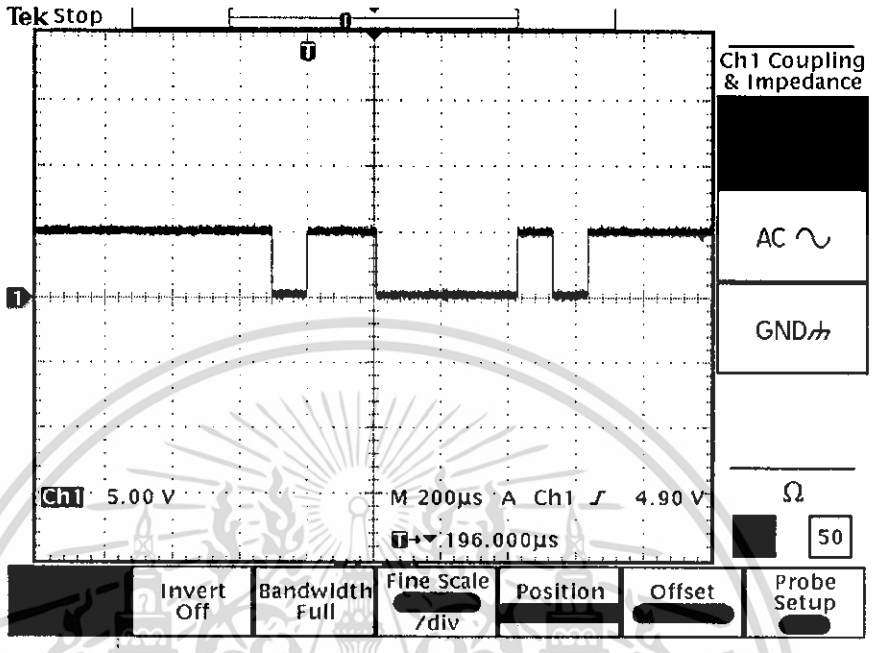
รูปที่ 4.7 ปุ่มควบคุมในขณะกดปุ่มเดินหน้าและเลี้ยวขวา

ในขณะที่กดปุ่มส่งข้อมูลเดินหน้าค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล "C" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "C" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 0011 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



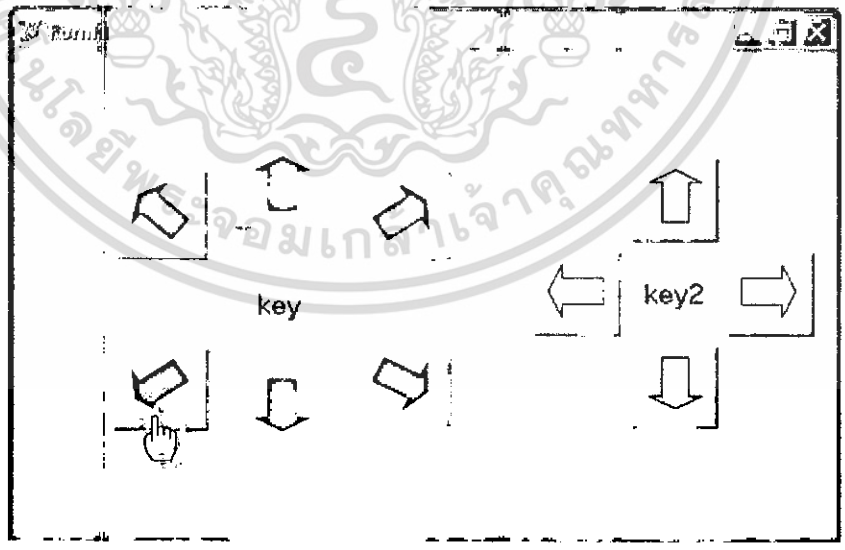
รูปที่ 4.8 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้รถสำรวจเดินหน้าและเลี้ยวขวา

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



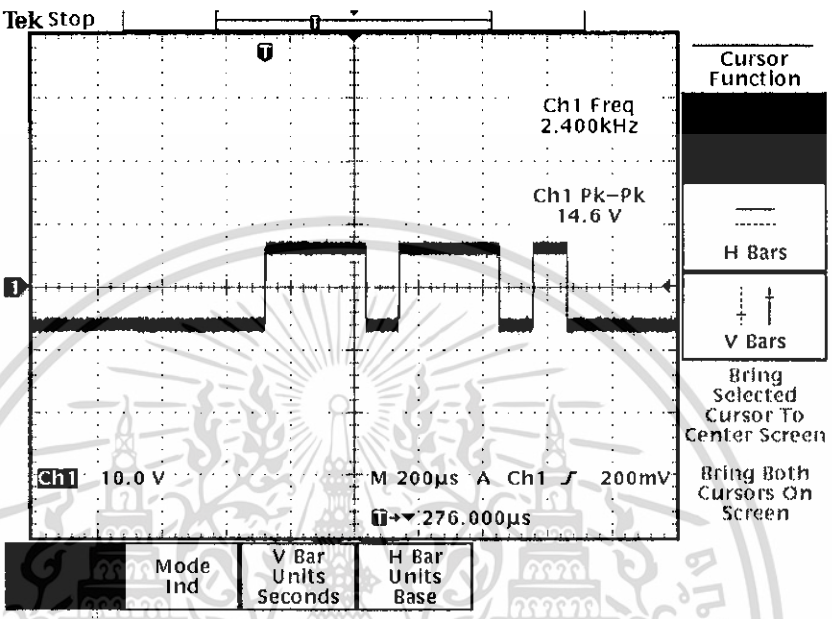
รูปที่ 4.9 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้รถสำรวจเดินหน้าและเลี้ยวขวา

4.1.4 กดปุ่มถอยหลังและเลี้ยวซ้าย



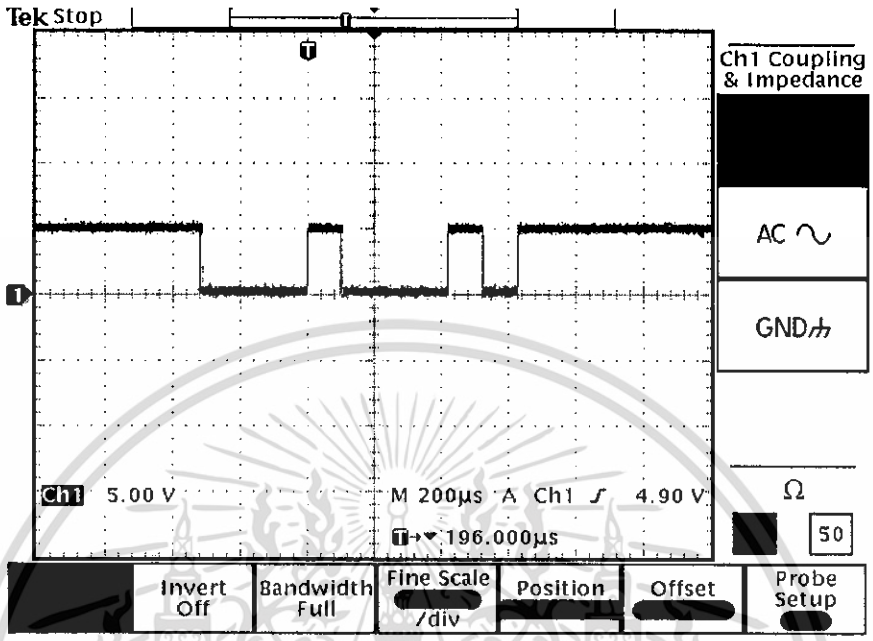
รูปที่ 4.10 ปุ่มควบคุมในเมื่อกดปุ่มถอยหลังและเลี้ยวซ้าย

ในขณะที่กดปุ่มส่งข้อมูลเดินหน้าค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล "D" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "D" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 0100 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



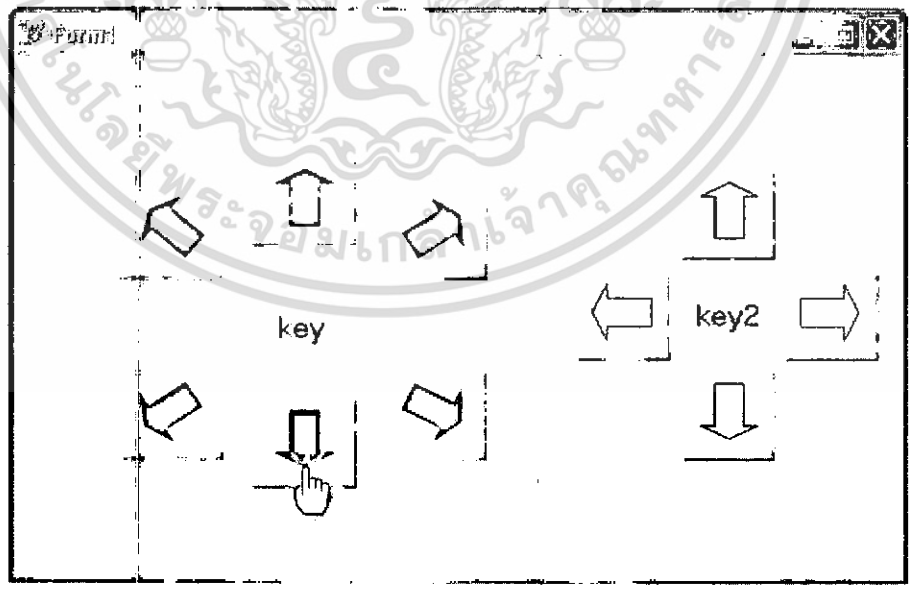
รูปที่ 4.11 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้รถสำรวจถอยหลังและเลี้ยวซ้าย

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



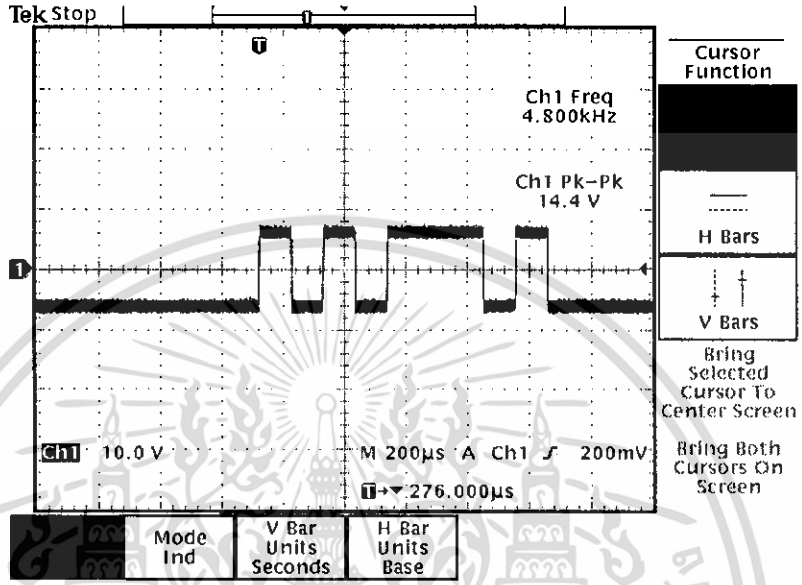
รูปที่ 4.12 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้รถสำรวจถอยหลังและเลี้ยวซ้าย

4.1.5 กดปุ่มถอยหลัง



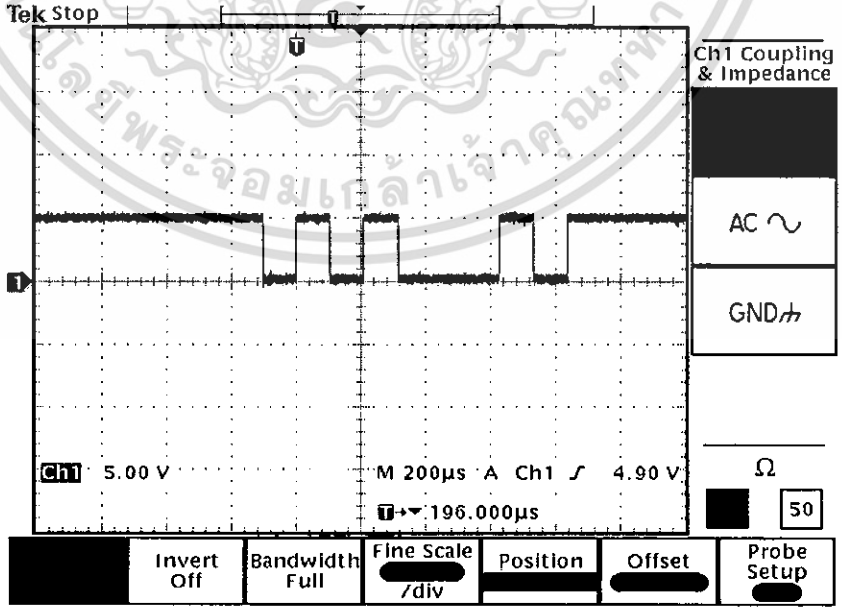
รูปที่ 4.13 ปุ่มควบคุมเมื่อกดปุ่มถอยหลัง

ในขณะที่กดปุ่มส่งข้อมูลเดินหน้าค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล "E" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "E" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 0101 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



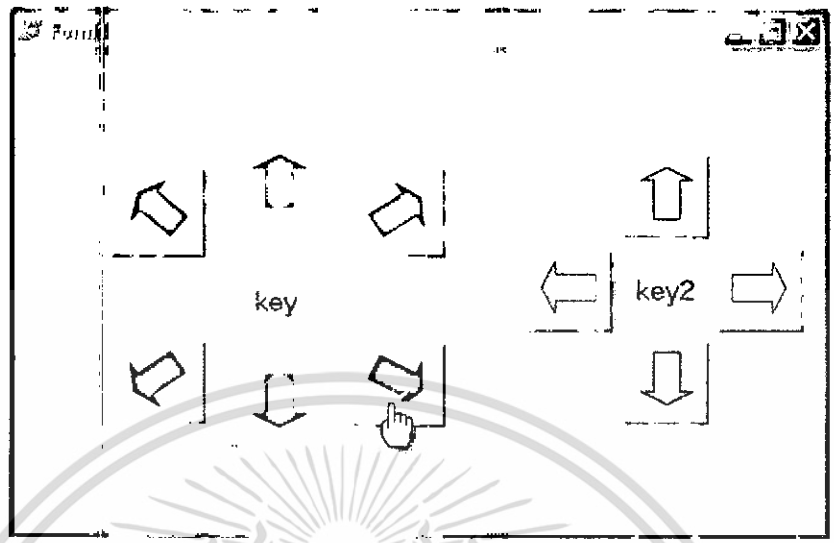
รูปที่ 4.14 สัญญาณข้อมูลไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้รถสำรวจถอยหลัง

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



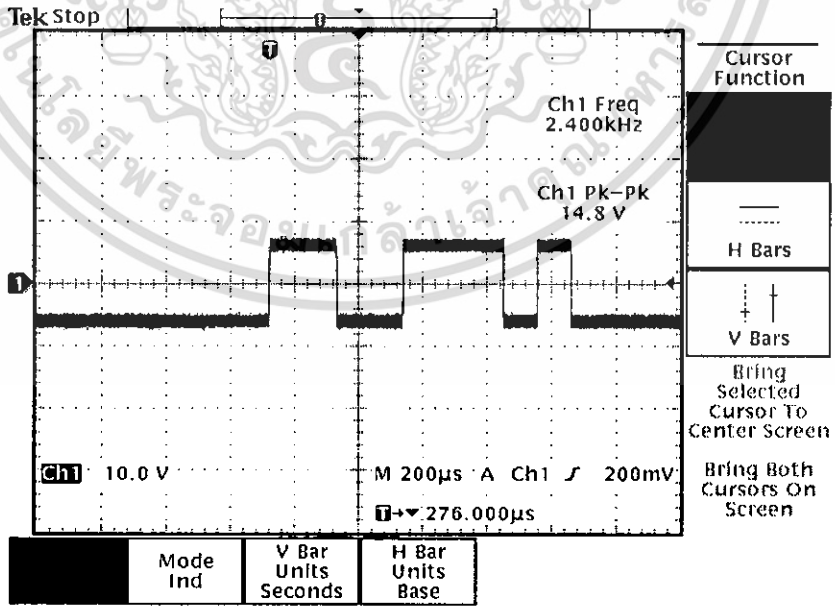
รูปที่ 4.15 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้รถสำรวจถอยหลัง

4.1.6 กดปุ่มถอยหลังและเลี้ยวขวา



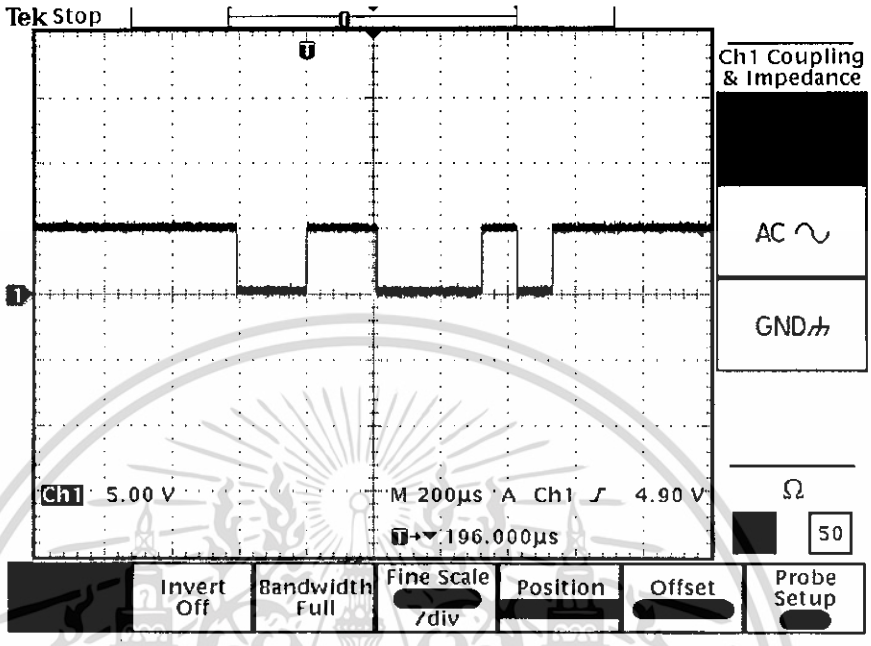
รูปที่ 4.16 ปุ่มควบคุมในขณะทำการสั่งให้รถสำรวจถอยหลังและเลี้ยวขวา

ในขณะที่กดปุ่มส่งข้อมูลเดินหน้าค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล "F" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "F" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 0110 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



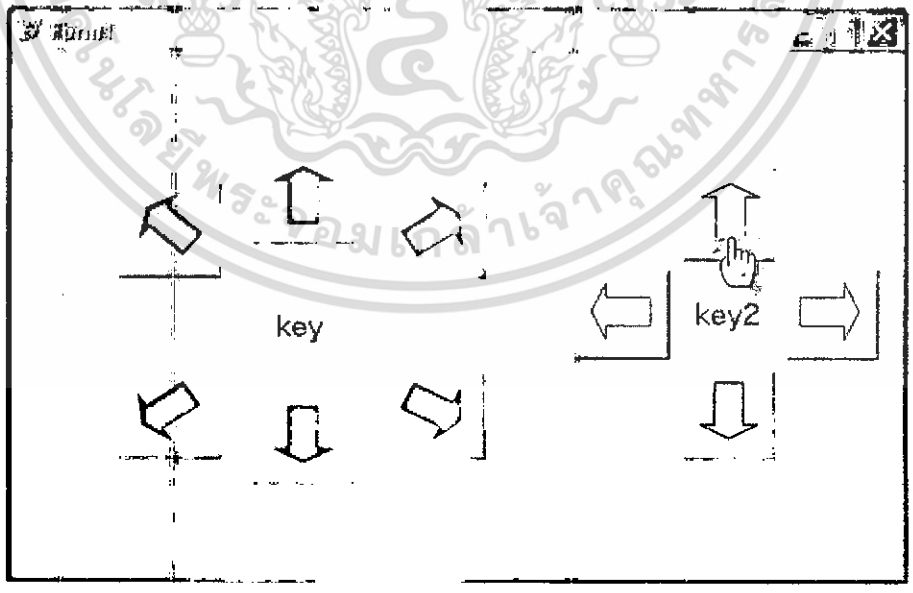
รูปที่ 4.17 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้รถสำรวจถอยหลังและเลี้ยวขวา

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



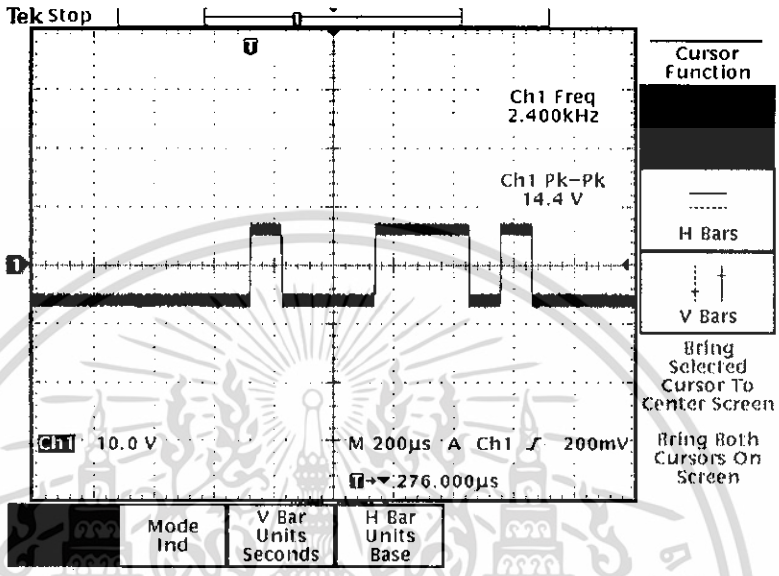
รูปที่ 4.18 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้รถสำรวจถอยหลังและเลี้ยวขวา

4.1.7 กดปุ่มควบคุมกล้องเลื่อนลง



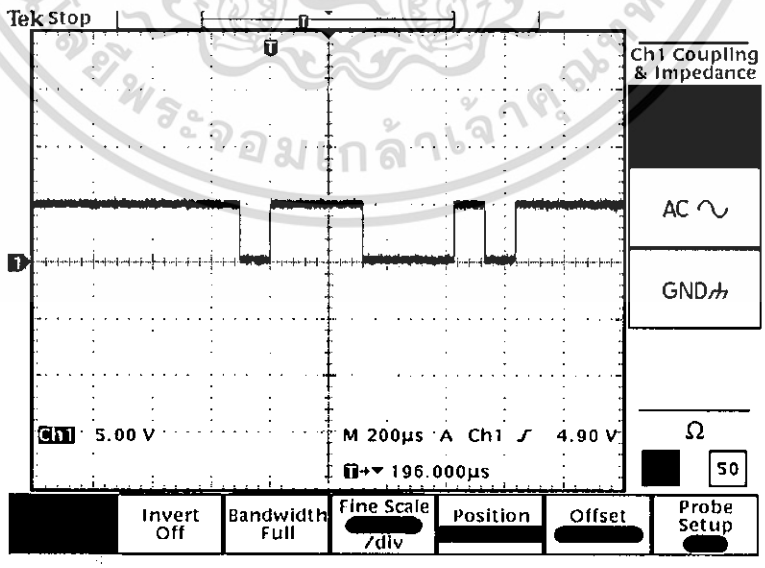
รูปที่ 4.19 ปุ่มควบคุมเมื่อกดปุ่มควบคุมกล้องเลื่อนลง

ในขณะที่กดปุ่มส่งข้อมูลเลื่อนกล้องลง ค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล “G” ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII “G” ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 0111 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



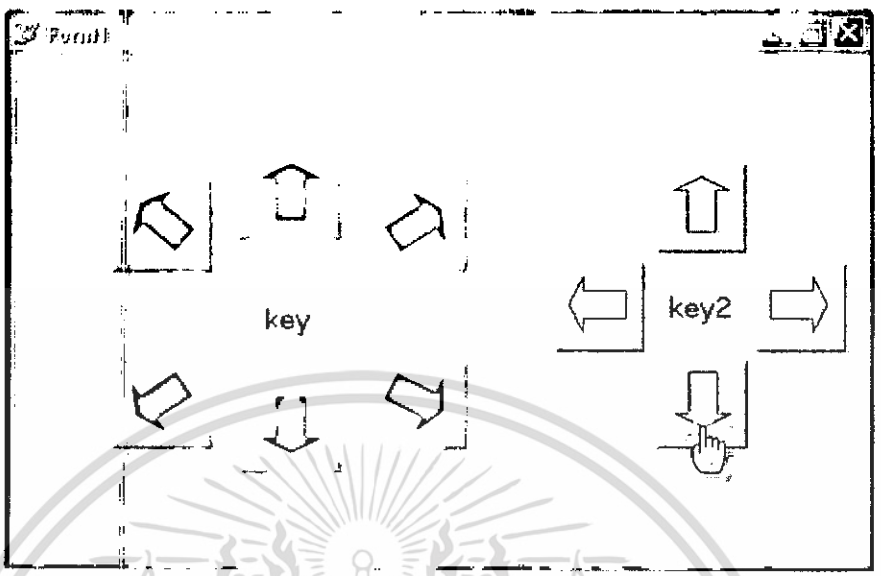
รูปที่ 4.20 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้กล้องเลื่อนลง

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



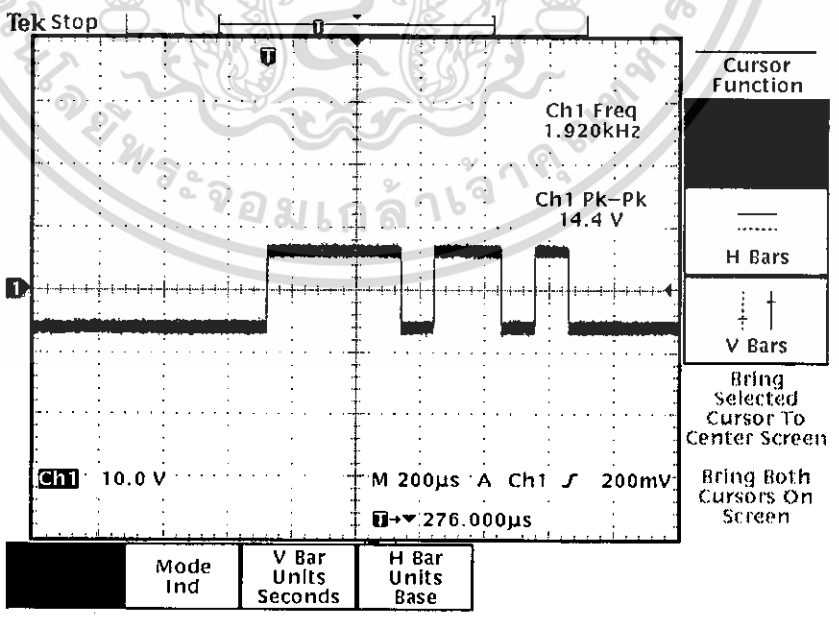
รูปที่ 4.21 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้กล้องเลื่อนลง

4.1.8 กดปุ่มควบคุมกล้องเลื่อนขึ้น



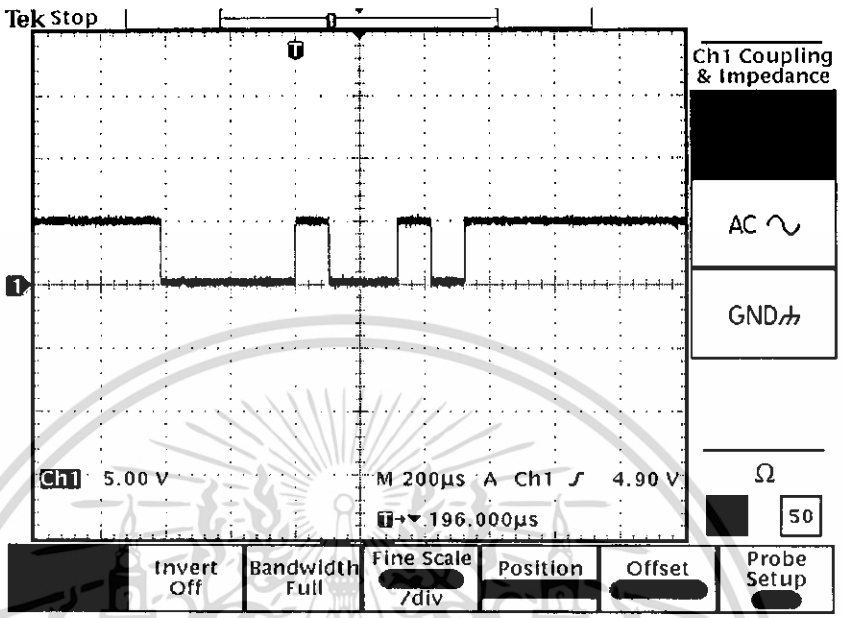
รูปที่ 4.22 ปุ่มควบคุมในขณะทำการสั่งให้กล้องเลื่อนขึ้น

ในขณะที่กดปุ่มส่งข้อมูลเลื่อนกล้องขึ้น ค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่งข้อมูล "H" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "H" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 1000 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



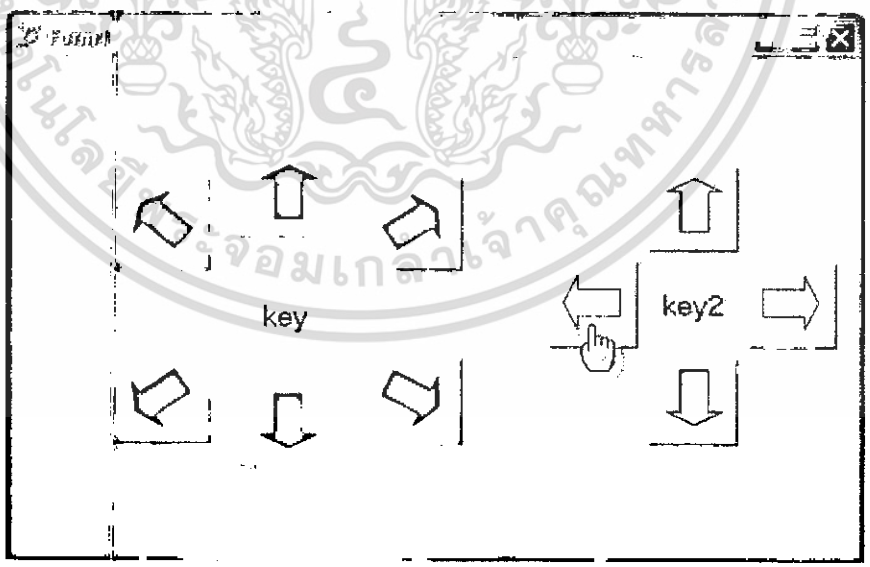
รูปที่ 4.23 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้กล้องเลื่อนขึ้น

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



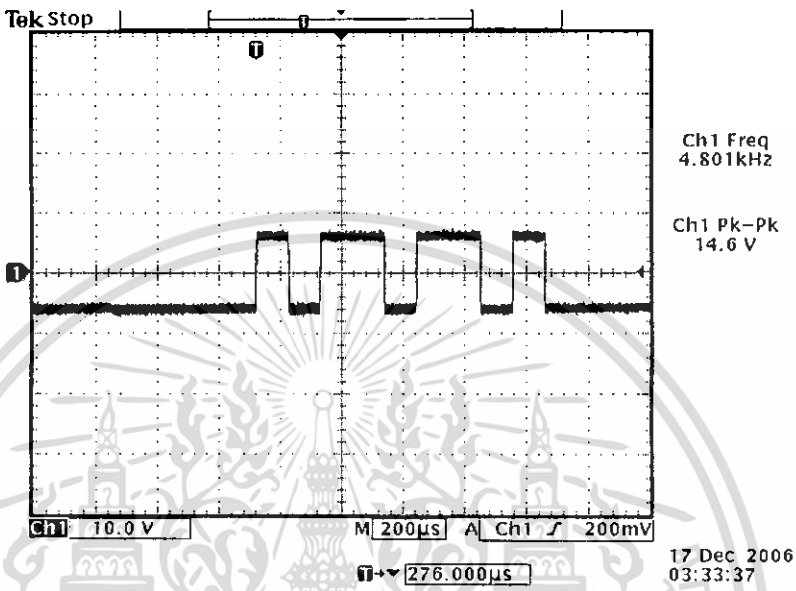
รูปที่ 4.24 สัญญาณข้อมูลไบนารี่ด้านรับ ในขณะทำการสั่งให้กล้องเลื่อนขึ้น

4.1.9 กดปุ่มควบคุมกล้องเลื่อนซ้าย



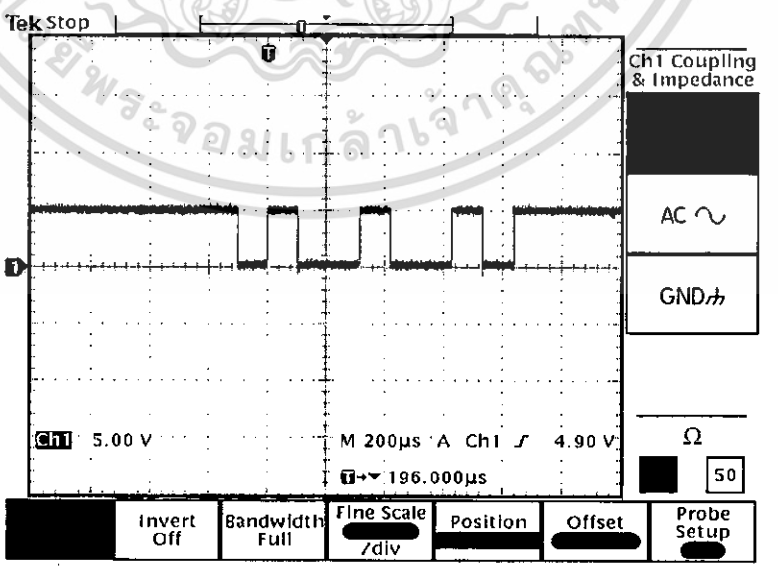
รูปที่ 4.25 ปุ่มควบคุมในขณะทำการสั่งให้กล้องเลื่อนซ้าย

ในขณะที่กดปุ่มส่งข้อมูลเลื่อนกล้องไปทางซ้าย ค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่ง ข้อมูล "I" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "I" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 1001 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



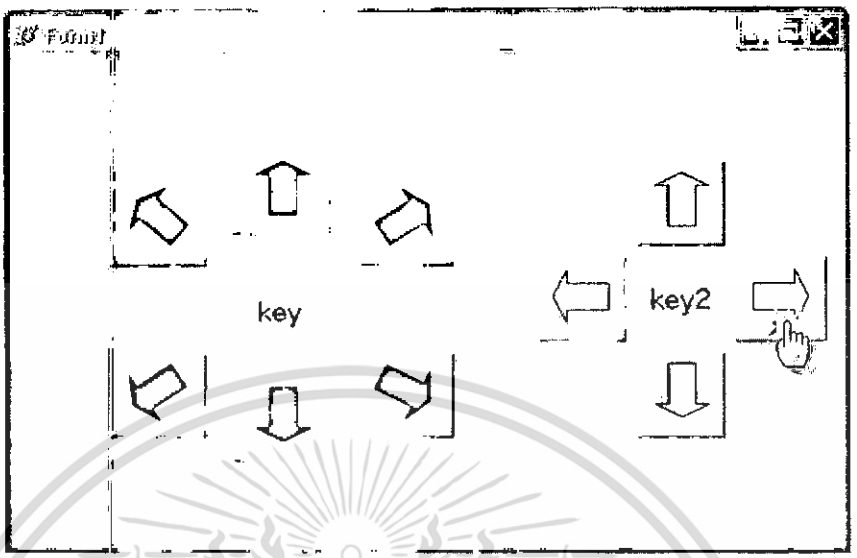
รูปที่ 4.26 สัญญาณข้อมูลไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้กล้องเลื่อนซ้าย

และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



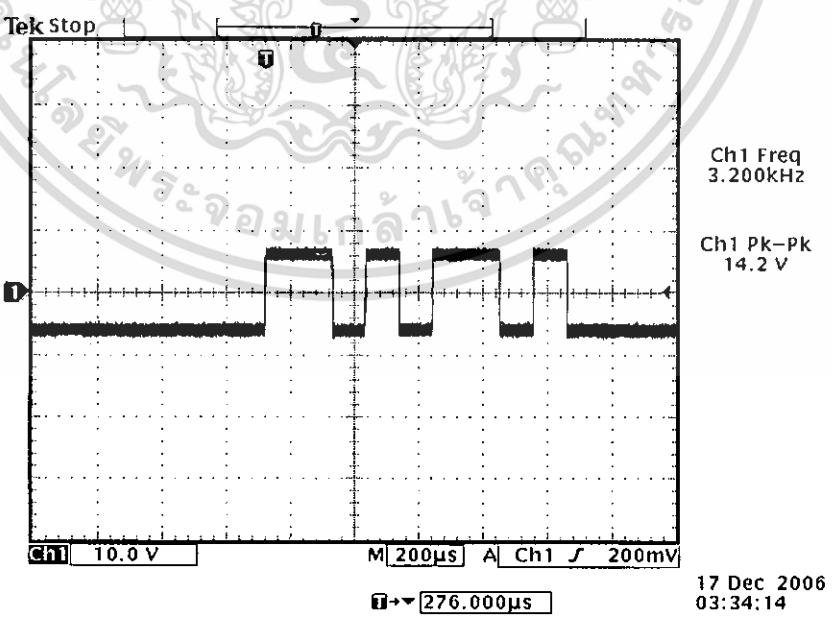
รูปที่ 4.27 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้กล้องเลื่อนซ้าย

4.1.10 กดปุ่มควบคุมกล้องเลื่อนขวา



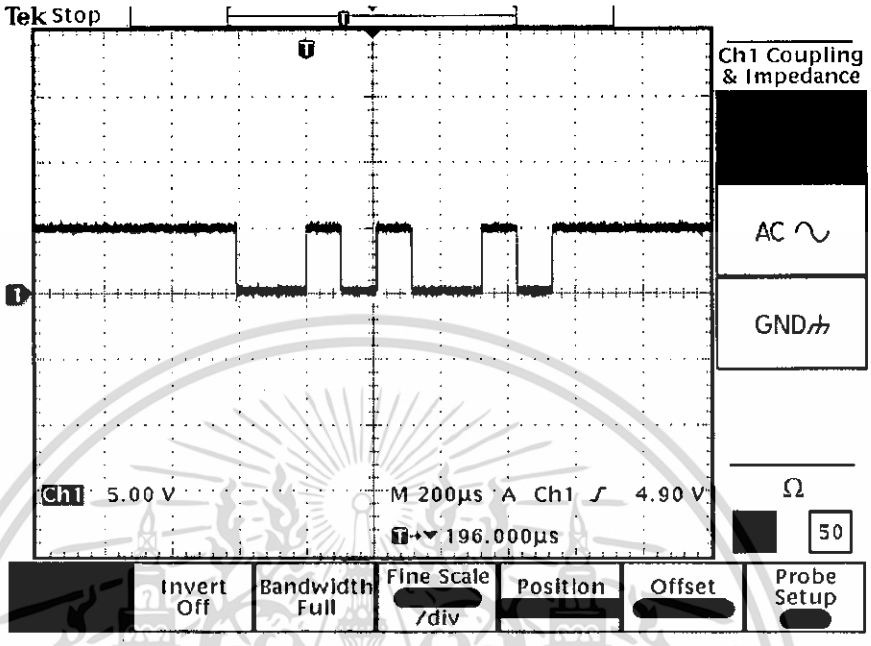
รูปที่ 4.28 ปุ่มควบคุมในขณะทำการสั่งให้กล้องเลื่อนขวา

ในขณะที่กดปุ่มส่งข้อมูลเลื่อนกล้องไปทางซ้าย ค้างไว้จากโปรแกรมในคอมพิวเตอร์นั้น โปรแกรมจะทำการส่ง ข้อมูล "J" ออก มาทางภาคส่ง ด้วยวิธีการส่งแบบ GFSK ซึ่ง ข้อมูล ASCII "J" ที่ถูกส่งมานั้น เมื่อแปลง เป็น Binary Code จะมีค่า เท่ากับ 0100 1010 ซึ่งเมื่อทำการวัดสัญญาณที่สาย RX ก่อนที่สัญญาณจะถูกส่งไปยังโมดูลส่งสัญญาณ จะมีผลการทดลองดังนี้



รูปที่ 4.29 สัญญาณข้อมูล ไบนารี่ที่ทำการวัดที่สาย RX ในขณะทำการสั่งให้กล้องเลื่อนขวา

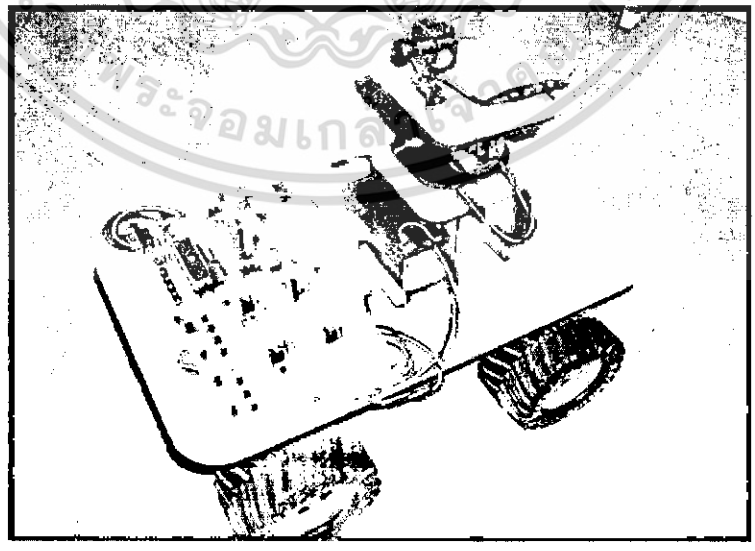
และ ภาครับ รับสัญญาณมาประมวลผล และได้ทำการวัดสัญญาณที่ขา 9 ของ RS-232 ซึ่งเป็นขาที่ส่งข้อมูลไปยัง MCS-51 ซึ่งมีรูปสัญญาณที่ส่งมาดังนี้



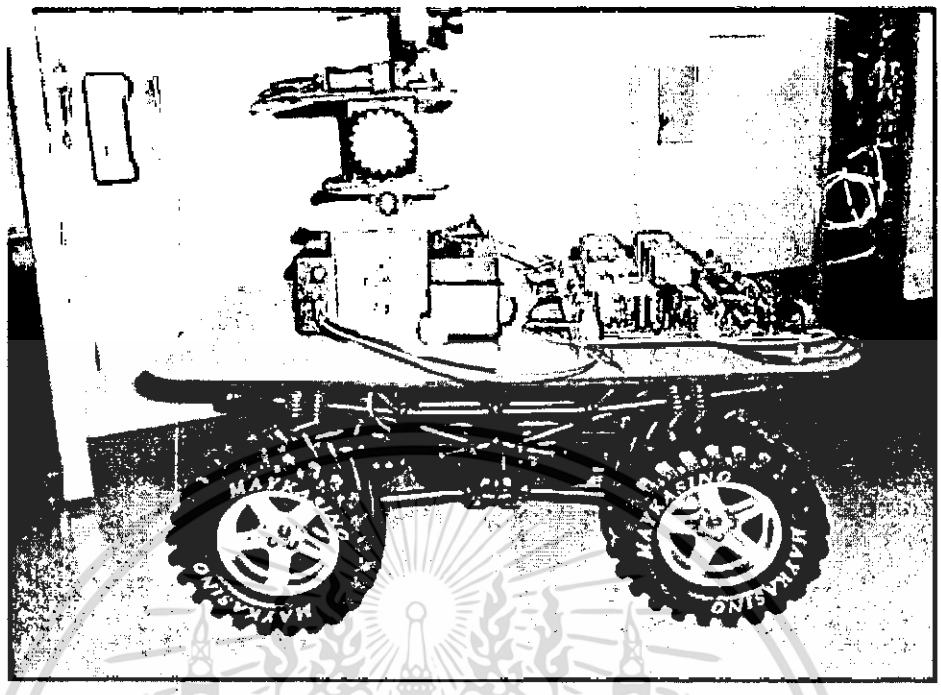
รูปที่ 4.30 สัญญาณข้อมูลไบนารี่ในขณะทำการสั่งให้กล้องเลื่อนขวา

4.2 รูปภาพรถสำรวจ ในมุมมองต่างๆ

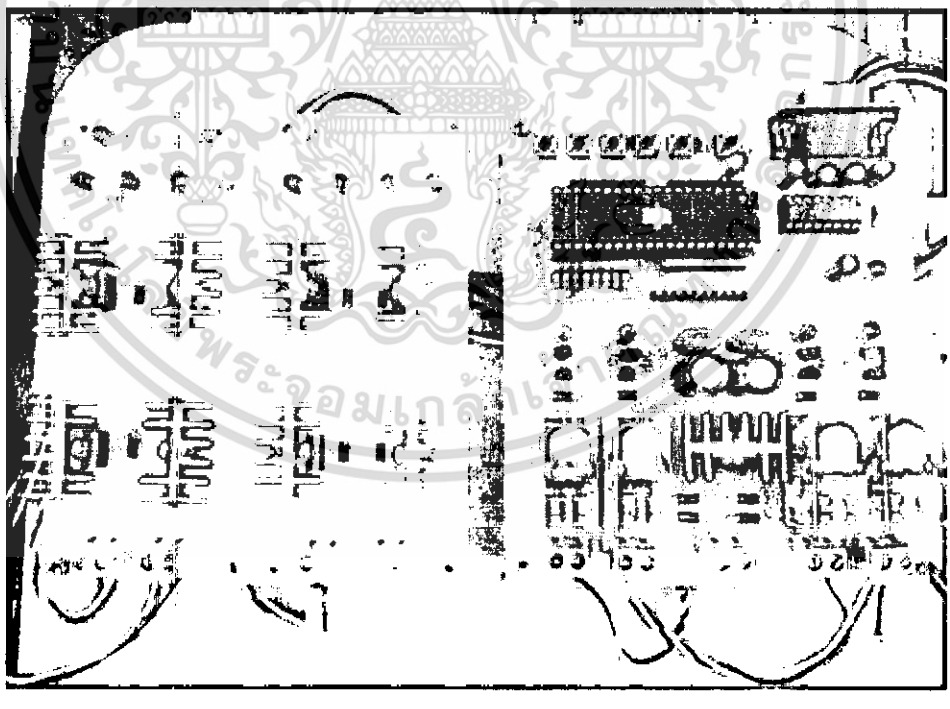
รถสำรวจหลังจากที่ได้ทำการออกแบบและ ทดสอบจนเสร็จสมบูรณ์โดยมี รูปร่างของรถสำรวจ และรูปวงจรหลังจากที่ทำการ ประกอบส่วนประกอบทั้งหมด ดัง รูปที่ 4.31,4.32 และ 4.33 ดังนี้



รูปที่ 4.31 ภาพถ่ายรถสำรวจในสภาพที่สมบูรณ์ ในแนวเฉียง



รูปที่ 4.32 ภาพถ่ายรถสำรวจในสภาพที่สมบูรณ์ ในมุมมองด้านข้าง



รูปที่ 4.33 ภาพถ่ายวงจรที่อยู่ติดกับรถสำรวจ

บทที่ 5
สรุปผลการทดลอง และวิจารณ์ผลการทดลอง

สรุปผลการทดลอง

โครงการนี้ได้ทำการออกแบบทดลองและสร้างรถสำรวจควบคุมไร้สายโดยใช้คอมพิวเตอร์โดยได้มีการทดสอบโปรแกรมและวงจรในส่วนต่างๆ ซึ่งมีผลการทดลองของส่วนต่างๆ สามารถทำงานได้และสามารถควบคุมรถและกล้องให้เคลื่อนที่ตามที่ออกแบบไว้ได้

ปัญหาที่พบ

ปัญหาที่พบ คือ กำลังขับของมอเตอร์นั้นมีสมรรถนะไม่มากพอที่จะเคลื่อนที่ในสภาวะที่ยากลำบากจึงทำให้การสำรวจในบางเส้นทางไม่สามารถกระทำได้

แนวทางการพัฒนา

สำหรับแนวทางการพัฒนา นั้น ควรจะเพิ่มกำลังขับของมอเตอร์ให้รถมีกำลังมากขึ้น มีการส่งสัญญาณควบคุมระหว่างรถกับส่วนควบคุมให้มีระยะทางมากขึ้นและปรับขนาดรถให้มีขนาดที่เหมาะสมเพื่อที่สามารถเดินทางในสภาวะที่ยากลำบากต่อเส้นทางในการสำรวจมากขึ้นที่มนุษย์ไม่อาจเข้าถึงได้ และควรเพิ่มส่วนของการควบคุมรถสำรวจที่ ขาดการติดต่อ ให้มีทำงานและตัดสินใจได้เองในกรณีที่ขาดการติดต่อ

ความสามารถและประโยชน์ของโครงงาน

1.สามารถควบคุมรถและกล้องที่ติดกับรถสำรวจได้แบบไร้สายโดยควบคุมผ่านคอมพิวเตอร์

2.รถสำรวจสามารถเคลื่อนที่ได้ถึง 6 ทิศทาง

3.สามารถควบคุมกล้องให้หมุนได้ 300 องศา แบบสามมิติ

4.ความสามารถของการ์ดที่รับภาพเข้ามาแสดงผลยังคอมพิวเตอร์นั้น สามารถบันทึกภาพแบบเคลื่อนไหวได้ ซึ่งสามารถนำภาพที่บันทึกได้ มาใช้ประโยชน์ในเชิงประยุกต์ต่อไป

หนังสืออ้างอิง

1. สัจจะ จรัสรุ่งรวี, จักรพงษ์ สุขประเสริฐ," เริ่มต้นมืออาชีพด้วย DELPHI 7 ฉบับสมบูรณ์", นนทบุรี : Infopress Developer Book, พ.ศ. 2547.

2. รองศาสตรจารย์ สมยศ จุณณะปิยะ, "การประยุกต์ใช้งาน ไมโครคอนโทรลเลอร์", สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2546.

3. วรลักษณ์ อินทร์มี, วิทยา กลีบเมฆ, "รถสำรวจเคลื่อนที่ควบคุมโดยคอมพิวเตอร์", ปริญญานิพนธ์ คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง, 2546.

4. ปิยรัตน์ ม่วงพานิช, "รถสำรวจเคลื่อนที่ควบคุมโดยคอมพิวเตอร์", ปริญญานิพนธ์ คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง,2542.

ภาคผนวก

คำสั่งของโปรแกรมต่างๆที่ใช้ในโครงงาน

1. คำสั่งภาษาเดลไฟ

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls, VaClasses, VaComm, Buttons, Mask, AppEvnts,
ExtCtrls;
type
  TForm1 = class(TForm)
    VaComm1: TVaComm;
    Label1: TLabel;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    SpeedButton8: TSpeedButton;
    Timer1: TTimer;
    SpeedButton9: TSpeedButton;
    SpeedButton10: TSpeedButton;
    Label2: TLabel;
    SpeedButton11: TSpeedButton;
    SpeedButton12: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton1MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton2MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton3MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton4MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton5MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton6MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
```

```
    procedure SpeedButton7Click(Sender: TObject);
    procedure SpeedButton7MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton8MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);

    procedure FormKeyUp(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure Timer1Timer(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton9MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton10MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);

    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton11MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton12MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);

     private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  St : string;
  Form1: TForm1;

implementation

{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);

begin
       VaComm1.Open;
       form1.KeyPreview:=true;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
       VaComm1.Close;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
      TIMER1.Enabled:= FALSE;

end;
```

```
procedure TForm1.SpeedButton1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
Begin
         St := 'G';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label2.Caption:= 'G' ;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
      TIMER1.Enabled:= FALSE;
end;

procedure TForm1.SpeedButton2MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    St := 'H';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label2.Caption:= 'H' ;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
         TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton3MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
         St := 'B';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label1.Caption:= 'B' ;
end;
procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
      TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton4MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
        St := 'E';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label1.Caption:= 'E' ;
end;

procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
         TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton5MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

```
begin
      St := 'A';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label1.Caption:= 'A' ;
end;

procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
      TIMER1.Enabled:= FALSE;
end;

procedure TForm1.SpeedButton6MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
      St := 'D';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label1.Caption:= 'D' ;
end;

procedure TForm1.SpeedButton7Click(Sender: TObject);
begin
      TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton7MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
      St := 'C';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label1.Caption:= 'C' ;
end;

procedure TForm1.SpeedButton8Click(Sender: TObject);
begin
      TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton8MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
       St := 'F';
        Vacomm1.WriteText(St);
        TIMER1.Enabled:= TRUE;
        label1.Caption:= 'F' ;
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
     case Key of

          'A'  :
              BEGIN
                St := 'A';
                label1.Caption:= 'A' ;
                Vacomm1.WriteText('st') ;
```

```
      TIMER1.Enabled:= TRUE;

    END;
'B'  :
    BEGIN
      St := 'B';
      label1.Caption:= 'B' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
'C'  :
    BEGIN
      St := 'C';
      label1.Caption:= 'C' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
'D'  :
    BEGIN
      St := 'D';
      label1.Caption:= 'D' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
    'E'  :
    BEGIN
      St := 'E';
      label1.Caption:= 'E' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
    'F'  :
    BEGIN
      St := 'F';
      label1.Caption:= 'F' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
    'G'  :
    BEGIN
      St := 'G';
      label2.Caption:= 'G' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
    'H'  :
    BEGIN
      St := 'H';
      label2.Caption:= 'H' ;
      Vacomm1.WriteText('st') ;
      TIMER1.Enabled:= TRUE;

    END;
    'I'  :
    BEGIN
```

```
                St := 'I';
                label2.Caption:= 'I' ;
                Vacomm1.WriteText('st') ;
                TIMER1.Enabled:= TRUE;

              END;
              'J'  :
              BEGIN
                St := 'J';
                label2.Caption:= 'J' ;
                Vacomm1.WriteText('st') ;
                TIMER1.Enabled:= TRUE;

              END;

          else
         label1.Caption:= 'not use' ;
         label2.Caption:= 'not use' ;

          end;
end;


procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
       TIMER1.Enabled:= FALSE;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Vacomm1.WriteText(St);
end;

procedure TForm1.SpeedButton9Click(Sender: TObject);
begin

         TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton10Click(Sender: TObject);
begin
          TIMER1.Enabled:= FALSE;
end;

procedure TForm1.SpeedButton9MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
         St := 'I';
         Vacomm1.WriteText(St);
         TIMER1.Enabled:= TRUE;
         label2.Caption:= 'I' ;
end;

procedure TForm1.SpeedButton10MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
          St := 'J';
         Vacomm1.WriteText(St);
```

```
        TIMER1.Enabled:= TRUE;
        label2.Caption:= 'J' ;
end;


procedure TForm1.SpeedButton11Click(Sender: TObject);
begin

        TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton11MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
              St := 'G';
        Vacomm1.WriteText(St);
        TIMER1.Enabled:= TRUE;
        label2.Caption:= 'G' ;
end;

procedure TForm1.SpeedButton12Click(Sender: TObject);
begin

        TIMER1.Enabled:= FALSE;

end;

procedure TForm1.SpeedButton12MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
        St := 'H';
        Vacomm1.WriteText(St);
        TIMER1.Enabled:= TRUE;
        label2.Caption:= 'H' ;
end;

end.
```

2. คำสั่งภาษาแอสแซมบลี ของไมโครคอนโทรลเลอร์

```
;
;       REMOTE CAR
;
SPAC:   DS      48

BUF1:   DS      1
BUF2:   DS      1
SW01:   DS      1
SW02:   DS      1

KLOC    BIT     00H
MLC1    BIT     01H
MLC2    BIT     02H
MLC3    BIT     03H
MLC4    BIT     04H

        ORG     0000H
        LJMP    INIT

        ORG     0030H
INIT:   MOV     P0,#0FFH
        MOV     P2,#0FFH
        MOV     P1,#0FFH
        MOV     P3,#0FFH

        MOV     R0,#00                  ; GP
        MOV     R1,#00                  ; GP
        MOV     R2,#00                  ; Count
        MOV     R3,#00                  ;
        MOV     R4,#00                  ;
        MOV     R5,#00                  ; Disp Buff
        MOV     R6,#00                  ; Delay
        MOV     R7,#00                  ; Delay

        MOV     SCON,#50H               ; Serial Mode 1
        MOV     TMOD,#22H               ; Timer 1 8 Bit Auto Reload
        MOV     TH1,#0FDH               ; 9600 Bps
        MOV     TL1,#0FDH
        SETB    TR1                     ; Start Timer
        CLR     RI                      ; Clear Receive Bit
        CLR     TI                      ; Clear Transmitt Bit

        MOV     SBUF,#'A'
        JNB     TI,$
        CLR     TI


;--->   Main Loop  ------------------------------------------------
---

MAIN:   JNB     RI,$                    ; Data In
        CLR     RI
        MOV     A,SBUF

        MOV     SBUF,A
        JNB     TI,$
        CLR     TI

MN10:   CJNE    A,#'A',MN11             ; A
        MOV     P0,#10101111B
        LJMP    MN20

MN11:   CJNE    A,#'B',MN12             ; B
        MOV     P0,#10111111B
        LJMP    MN20
```

```
MN12:   CJNE    A,#'C',MN13             ; C
        MOV     P0,#10011111B
        LJMP    MN20

MN13:   CJNE    A,#'D',MN14             ; D
        MOV     P0,#01101111B
        LJMP    MN20

MN14:   CJNE    A,#'E',MN15             ; E
        MOV     P0,#01111111B
        LJMP    MN20

MN15:   CJNE    A,#'F',MN16             ; F
        MOV     P0,#01011111B
        LJMP    MN20

MN16:   CJNE    A,#'G',MN17             ; G
        MOV     P0,#11111110B
        LJMP    MN20

MN17:   CJNE    A,#'H',MN18             ; H
        MOV     P0,#11111101B
        LJMP    MN20

MN18:   CJNE    A,#'I',MN19             ; I
        MOV     P0,#11111011B
        LJMP    MN20

MN19:   CJNE    A,#'J',MN1A             ; J
        MOV     P0,#11110111B
        LJMP    MN20

MN1A:   LJMP    MN30

MN20:   LCALL   HLAY
        MOV     P0,#0FFH

MN30:   NOP

        LJMP    MAIN


HLAY:   MOV     R6,#00
HL10:   MOV     R7,#00
        DJNZ    R7,$
        DJNZ    R6,HL10
        RET

        END
;
;       REMOTE CAR
;

SPAC:   DS      48

BUF1:   DS      1
BUF2:   DS      1
SW01:   DS      1
SW02:   DS      1

KLOC    BIT     00H
MLC1    BIT     01H
MLC2    BIT     02H
MLC3    BIT     03H
MLC4    BIT     04H

        ORG     0000H
        LJMP    INIT
```

```
        ORG     0030H
INIT:   MOV     P0,#0FFH
        MOV     P2,#0FFH
        MOV     P1,#0FFH
        MOV     P3,#0FFH

        MOV     R0,#00                  ; GP
        MOV     R1,#00                  ; GP
        MOV     R2,#00                  ; Count
        MOV     R3,#00                  ;
        MOV     R4,#00                  ;
        MOV     R5,#00                  ; Disp Buff
        MOV     R6,#00                  ; Delay
        MOV     R7,#00                  ; Delay

        MOV     SCON,#50H               ; Serial Mode 1
        MOV     TMOD,#22H               ; Timer 1 8 Bit Auto Reload
        MOV     TH1,#0FDH               ; 9600 Bps
        MOV     TL1,#0FDH
        SETB    TR1                     ; Start Timer
        CLR     RI                      ; Clear Receive Bit
        CLR     TI                      ; Clear Transmitt Bit

        MOV     SBUF,#'A'
        JNB     TI,$
        CLR     TI

;--->   Main Loop  ------------------------------------------------
---

MAIN:   JNB     RI,$                    ; Data In
        CLR     RI
        MOV     A,SBUF

        MOV     SBUF,A
        JNB     TI,$
        CLR     TI

MN10:   CJNE    A,#'A',MN11             ; A
        MOV     P0,#10101111B
        LJMP    MN20

MN11:   CJNE    A,#'B',MN12             ; B
        MOV     P0,#10111111B
        LJMP    MN20

MN12:   CJNE    A,#'C',MN13             ; C
        MOV     P0,#10011111B
        LJMP    MN20

MN13:   CJNE    A,#'D',MN14             ; D
        MOV     P0,#01101111B
        LJMP    MN20

MN14:   CJNE    A,#'E',MN15             ; E
        MOV     P0,#01111111B
        LJMP    MN20

MN15:   CJNE    A,#'F',MN16             ; F
        MOV     P0,#01011111B
        LJMP    MN20

MN16:   CJNE    A,#'G',MN17             ; G
        MOV     P0,#11111110B
        LJMP    MN20

MN17:   CJNE    A,#'H',MN18             ; H
```

```
        MOV     P0,#11111101B
        LJMP    MN20

MN18:   CJNE    A,#'I',MN19             ; I
        MOV     P0,#11111011B
        LJMP    MN20

MN19:   CJNE    A,#'J',MN1A             ; J
        MOV     P0,#11110111B
        LJMP    MN20

MN1A:   LJMP    MN30

MN20:   LCALL   HLAY
        MOV     P0,#0FFH

MN30:   NOP

        LJMP    MAIN


HLAY:   MOV     R6,#00
HL10:   MOV     R7,#00
        DJNZ    R7,$
        DJNZ    R6,HL10
        RET

        END
```