เครื่องแสดงผลแบบหลายฟังก์ชันและระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์

MULTIFUNCTION DISPLAY AND SECURITY SYSTEM FOR MOTORCYCLE

จัดทำโดย

นายศุกะรสุต เมฆศิริ รหัส 45010774

นายศุภชัย เขื่อนแก้ว รหัส 45010777

นายอาทิตย์ ภูเดช รหัส 45010962

อาจารย์ที่ปรึกษา

ผศ.พลผดุง ผดุงกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ ปีการศึกษา 2548

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องแสดงผลแบบหลายฟังก์ชั่นและระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์

MULTIFUNCTION DISPLAY AND SECURITY SYSTEM FOR MOTORCYCLE

ผู้จัดทำ

1. นายศุกะรสุต เมฆศิริ 45010774
2. นายศุภชัย เขื่อนแก้ว 45010777
3. นายอาทิตย์ ภูเดช 45010962

..... อาจารย์ที่ปรึกษา

(ผศ.พลผดุง ผดุงกุล)

เครื่องแสดงผลแบบหลายฟังก์ชั่น และ ระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์

นายศุกะรสุต เมฆศิริ
นายศุภชัย เชื่อนแก้ว
นายอาทิตย์ ภูเดช
ผศ.พลผดุง ผดุงกุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

โครงงานนี้เป็นการสร้างเครื่องแสดงผลแบบหลายฟังก์ชั่น และระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์ โดยในการแสดงผลจะแสดง ความเร็วรอบเครื่อง, ความเร็วรถ,ระดับน้ำมันเชื้อเพลิง, ระดับแรงดันแบตเตอรี่ ส่วนแสดงสถานะการทำงานต่างๆ ของรถ เช่น เปิด-ปิดไฟเลี้ยว ไฟหน้ารถ เป็นต้น พร้อมทั้งมีระบบรักษาความปลอดภัยและมีการแสดง เวลา, วัน, เดือน, ปี ซึ่งโครงงานนี้เป็นงานทางด้านฮาร์ดแวร์ และซอฟแวร์ร่วมกันโดยใช้ไมโครคอนโทรลเลอร์ MCS-51 เป็นตัวควบคุมระบบทั้งหมด ทั้งทางด้านการแสดงผล และการรักษาความปลอดภัย

Multifunction display and security system for motorcycle

Sukarasut Maksiri

Supachai Keankaew

Arthit Phudej

Asst.Prof.Polpadung Phadungkul Advisor

2005

Abstract

This project is to construct Multifunction display and security for motorcycle. It is able to display revolution per minute of engine(R.P.M.), velocity , fuel level , battery level show other status of motorcycle for example turn-left, turn-right, beam and security system together with show time/date/month/year . This project is works of hardware and software that use Microcontroller MCS-51 to control all system.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดีเพราะได้รับคำปรึกษา และข้อเสนอแนะจากอาจารย์ที่ปรึกษา ผศ. พลผดุง ผดุลกุล อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่าง ๆ ตลอดจนแก้ไขเรียบเรียงทำให้ปริญญานิพนธ์ฉบับนี้ให้สำเร็จมีความสมบูรณ์มากยิ่งขึ้น ผู้จัดทำรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณคณาจารย์ และเพื่อน ๆ นักศึกษาภาควิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ได้ให้ความช่วยเหลือในการทำปริญญานิพนธ์ฉบับนี้

สุดท้ายขอขอบพระคุณ คุณพ่อ คุณแม่ ผู้มีพระคุณอย่างสูงที่ได้สละกำลังทรัพย์ กำลังใจและความห่วงใยตลอดการทำงานอันทำให้ผู้จัดทำฝ่าฟันอุปสรรคต่าง ๆ จนสำเร็จ

คุณประโยชน์ และความดีอันเกิดจากปริญญานิพนธ์ฉบับนี้ข้าพเจ้าขอมอบแด่ คุณพ่อ คุณแม่ และคณาจารย์ผู้ประสิทธิประศาสตร์วิชาทุกท่าน

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่1
บทนำ

1.1 แนวคิดริเริ่มในการทำโครงงาน

การดำเนินชีวิตประจำวันของเราต้องมีการคมนาคมเข้ามาเกี่ยวข้องเสมอ โดยเฉพาะการคมนาคมทางบก ไม่ว่าจะเป็น รถโดยสารประจำทาง รถยนต์ รถจักรยานยนต์ หรือรถจักรยาน โดยเฉพาะการใช้รถจักรยานยนต์ซึ่งมีการใช้กันอย่างแพร่หลายในปัจจุบัน เนื่องจากเป็นพาหนะที่ใช้ง่าย สะดวก และมีความคล่องตัวสูง

รถจักรยานยนต์ที่ใช้กันอยู่ในปัจจุบันนี้จะมีมาตรวัดต่าง ๆ เป็นแบบเข็มอยู่ ไม่ว่าจะเป็นมาตรวัดความเร็ว หรือ มาตรวัดน้ำมันเชื้อเพลิง ซึ่งค่อนข้างยากต่อการมอง และมีความละเอียดน้อย อีกทั้งยังขาดมาตรวัดหลาย ๆ อย่างอยู่ เช่น มาตรวัดความเร็วรอบเครื่องยนต์ มาตรวัดอุณหภูมิเครื่องยนต์ และมาตรวัดระดับแบตเตอรี และนอกจากนี้ยังรวมถึงระบบรักษาความปลอดภัย (ระบบกันขโมย) ซึ่งล้วนแล้วแต่มีความจำเป็นต่อการใช้รถจักรยานยนต์ดังนั้นผู้จัดทำจึงคิดทำโครงงานนี้ขึ้นเพื่อความสะดวก และความปลอดภัยของรถ และผู้ขับขี่รถจักรยานยนต์

1.2 ชื่อโครงงาน

เครื่องแสดงผลแบบหลายฟังก์ชั่น และระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์

1.3 วัตถุประสงค์ของการจัดทำโครงงาน

1. สร้างเครื่องแสดงผลบนจอ LCD ที่แสดงผลได้เป็นตัวเลข และสามารถแสดงผลได้หลายฟังก์ชั่น ดังนี้
    - ความเร็วรถ
    - ความเร็วรอบเครื่องยนต์
    - อุณหภูมิเครื่องยนต์
    - ระดับน้ำมันเชื้อเพลิง
    - ระดับแรงดันแบตเตอรี่
    - แสดงสถานะต่าง ๆของรถ เช่น ไฟเลี้ยว, ไฟหน้า
    - แสดงวันเวลา

    เพื่อความสะดวกต่อการใช้งานของผู้ขับขี่รถจักรยานยนต์

2. สร้างระบบเตือนเมื่อรถเกิดความผิดปกติ ได้แก่
   - น้ำมันเชื้อเพลิงใกล้จะหมด
   - ระดับแบตเตอรี่อ่อน

   เพื่อความปลอดภัยแก่รถจักรยานยนต์ และผู้ขับขี่รถจักรยานยนต์
3. สร้างระบบรักษาความปลอดภัย (ระบบกันขโมย) เพื่อความปลอดภัยต่อรถจักรยานยนต์ ซึ่งเป็นทรัพย์สินของผู้ใช้รถจักรยานยนต์

1.4 องค์ประกอบหลักของโครงงาน

1. ส่วนแสดงผลเป็นตัวเลขบนจอ LCD หลายฟังก์ชั่น ได้แก่
   - ความเร็วรถ
   - ความเร็วรอบเครื่องยนต์
   - อุณหภูมิเครื่องยนต์
   - ระดับน้ำมันเชื้อเพลิง
   - ระดับแรงดันแบตเตอรี
   - แสดงสถานะต่าง ๆของรถ เช่น ไฟเลี้ยว, ไฟหน้า
   - แสดงวันเวลา
2. ส่วนของระบบเตือนเมื่อรถเกิดความผิดปกติ ได้แก่
   - น้ำมันเชื้อเพลิงใกล้จะหมด
   - ระดับแบตเตอรีอ่อน
3. ส่วนของระบบรักษาความปลอดภัย (ระบบกันขโมย)

1.5 ขอบเขตของโครงงาน

1. สร้างเครื่องแสดงผลบนจอ LCD ที่แสดงผลได้เป็นตัวเลข และสามารถแสดงผลได้หลายฟังก์ชั่น ดังนี้
   - ความเร็วรถ
   - ความเร็วรอบเครื่องยนต์
   - อุณหภูมิเครื่องยนต์
   - ระดับน้ำมันเชื้อเพลิง
   - ระดับแรงดันแบตเตอรี
2. สร้างระบบเตือนเมื่อรถเกิดความผิดปกติ ได้แก่
   - น้ำมันเชื้อเพลิงใกล้จะหมด
   - ระดับแรงดันแบตเตอรีเมื่อมีค่าต่ำกว่าที่ตั้งไว้

3. สร้างระบบรักษาความปลอดภัย (ระบบกันขโมย)

เป็นการสร้างระบบความปลอดภัยให้กับรถจักรยานยนต์ที่มีการป้องกันทางด้านต่างๆ กล่าวคือ มีการตรวจจับการเปลี่ยนแปลงที่คอบังคับเลี้ยว โดยจะมีการติดตั้งเซนเซอร์ไว้ที่คอบังคับเลี้ยวของรถจักรยานยนต์ หลังจากที่มีการล็อคที่คอบังคับเลี้ยวแล้ว เมื่อมีการเปลี่ยนแปลงที่คอบังคับเลี้ยวหรือเมื่อมีการหมุนที่คอบังคับเลี้ยวจนถึงระดับหนึ่งจะทำให้เซนเซอร์ทำงานแล้ววงจรกันขโมยจะทำงาน นอกจากนี้ยังมีระบบการป้องกันการใช้งานรถโดยการป้อนรหัสผ่านเพื่อเปิดสวิตช์รถจักยานยนต์ โดยการป้อนรหัสผ่านนี้สามารถป้อนผ่านทางหน้าปัทม์ของรถจักรยานยนต์ได้โดยตรง ซึ่งการแสดงผลก็จะแสดงข้อมูลผ่านทางจอแสดงผลแบบผลึกเหลวหรือจอแสดงผลแบบแอลซีดี(LCD) ขนาด 20x4 ตัวอักษร

บทที่ 2 ทฤษฎีและหลักการ

โครงสร้างทางสถาปัตยกรรมของไมโครคอนโทรลเลอร์ MCS-51 ไมโครคคอนโทรลเลอร์ซึ่งมีหน่วยความจำแบบแฟลช (flash memory) ของ Atmel Corporation มีเบอร์ขึ้นต้นด้วย AT89 มีคุณสมบัติที่สำคัญคือ

1.หน่วยความจำโปรแกรมภายในตัวไมโครคอนโทรลเลอร์เป็นแบบแฟลช ทำให้สามารถลบและเขียนใหม่ได้นับพันครั้งจึงสามารถใช้งานในรูปของชิปเดี่ยวไม่ต้องใช้หน่วยความจำภายในส่งผลให้สามารถใช้งานพอร์ตอินพุตเอาท์พุทได้อย่างเต็มประสิทธิภาพ

2.ต้นทุนในการพัฒนาระบบไมโครคอนโทรลเลอร์ลดลงอย่างมาก เนื่องจากไม่ต้องใช้เครื่องมือพัฒนาจำพวกอีมูเลเตอร์และเครื่องโปรแกรมอีพรอม

3.บริษัทผู้ผลิตได้ทำการผลิตไมโครคอนโทรลเลอร์ตระกูลนี้ออกมาหลายเบอร์และมีความสามารถแตกต่างกัน ทำให้มีทางเลือกในการใช้งานสูง

4.ด้วยการใช้งานหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ ทำให้สามารถป้องกันการคัดลอกข้อมูลในหน่วยความจำโปรแกรมได้เป็นอย่างดี

5.ในบางเบอร์ของไมโครคอนโทรลเลอร์ที่ผลิตโดย Atmel สามารถทำการโปรแกรมข้อมูลในหน่วยความจำโปรแกรมได้โดยที่ไม่ต้องถอดตัวไมโครคอนโทรลเลอร์ออกมาทำการโปรแกรม หรือเรียกว่า การโปรแกรมในวงจร หรือในระบบ (In-system programming) ทำให้การพัฒนาหรือการซ่อมบำรุง ตลอดจนการปรับปรุงหรืออัปเกรดข้อมูลในหน่วยความจำโปรแกรมทำได้อย่างสะดวก ภายใต้งบประมาณที่ไม่สูงมากนัก

6. ชุดคำสั่งและสถาปัตยกรรมพื้นฐานเหมือนกับไมโครคอนโทรลเลอร์ MCS-51 ของผู้ผลิตอื่น ไม่ว่าจะเป็นอินเทล, ซีเมนส์ หรือดัลลัส

2.1 คุณสมบัติของไมโครคอนโทรลเลอร์ ตระกูล MCS-51

- เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียูขนาด 8 บิต
- ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนใหม่ได้พันครั้ง หน่วยความจำข้อมูลพื้นฐานเป็นหน่วยความจำแบบแรม ในบางเบอร์จะมีหน่วยความจำอีอีพรอมเพิ่มเติม
- ขาพอร์ตเป็นแบบสองทิศทาง สามารถใช้งานเป็นได้ทั้งอินพุตและเอาร์พุต
- มีวงจรสื่อสารอนุกรมแบบฟุลดูเพล็กซ์
- ไทเมอร์/เคาน์เตอร์ขนาด 16 บิตอย่างน้อย 2 ตัว
- สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ได้ 6 ประเภท
- สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้สูงสุด 64 กิโลไบต์

ไมโครคอนโทรลเลอร์ MCS -51 ทุกเบอร์จะมีสถาปัตยกรรมและขาการใช้งานเหมือนกัน

- มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป
- มีวงจรสื่อสารอนุกรมแบบ SPI สำหรับในอนุกรม AT89Sxx
- มีวอตช์ด็อกไทเมอร์ในตัว สำหรับในอนุกรม AT89xx

ในรูปที่2.1 เป็นโครงสร้างพื้นฐานของของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89xx จะเห็นได้ว่าโครงสร้างของ AT89xx จะเหมือนกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 พื้นฐานหากแต่แตกต่างกันเฉพาะหน่วยความจำโปรแกรมแบบเฟลชที่เพิ่มเติมเข้ามา หากเป็นไมโครคอนโทรลเลอร์ในอนุกรม 87xx หน่วยความจำโปรแกรมภายในจะเป็นแบบอีพรอม และบางเบอร์สามารถโปรแกรมได้เพียงครั้งเดียว

สำหรับในรูปที่ 2.2 เป็นโครงสร้างพื้นฐานของอนุกรม AT89xx จะเห็นได้ว่า มีส่วนประกอบเพิ่มเติมแตกต่างจาก AT89xx อยู่หลายส่วน อาทิ วงจรเชื่อมต่ออนุกรมแบบ SPI ซึ่งในไมโครคอนโทรลเลอร์ อนุกรมใช้ในการเขียนข้อมูลลงในหน่วยความจำโปรแกรมโดยไม่ต้องถอดตัวชิปออกไปจากระบบหรือเรียกว่า การโปรแกรมในวงจร ไทเมอร์/เคาน์เตอร์ขนาด 16 บิตที่เพิ่มเติมเข้ามาอีกหนึ่งตัวเป็น ไทเมอร์ 2 และวงจรวอตช์ด็อกที่ใช้ในการตรวจสอบการทำงานผิดพลาดของซีพียู

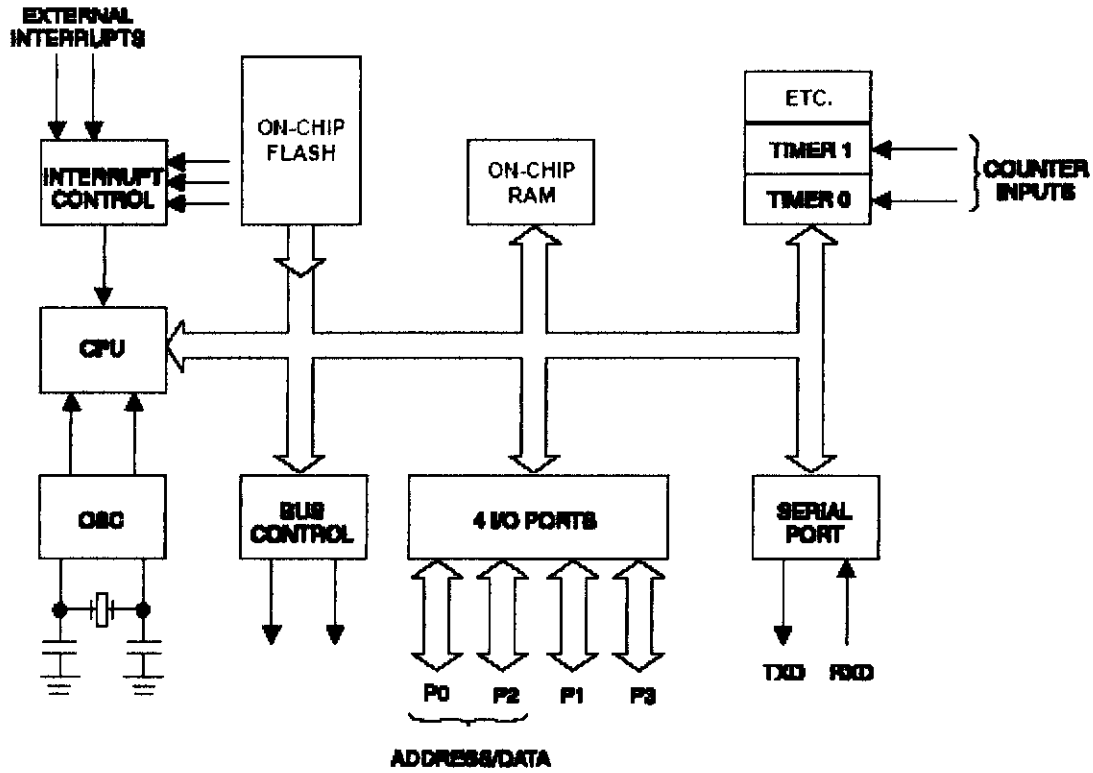
2.2 การจัดขาของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS -51 ทุกเบอร์จะมีสถาปัตยกรรมและขาการใช้งานเหมือนกัน โดยมีรายละเอียดของขาการใช้งานขั้นต้น ดังนี้

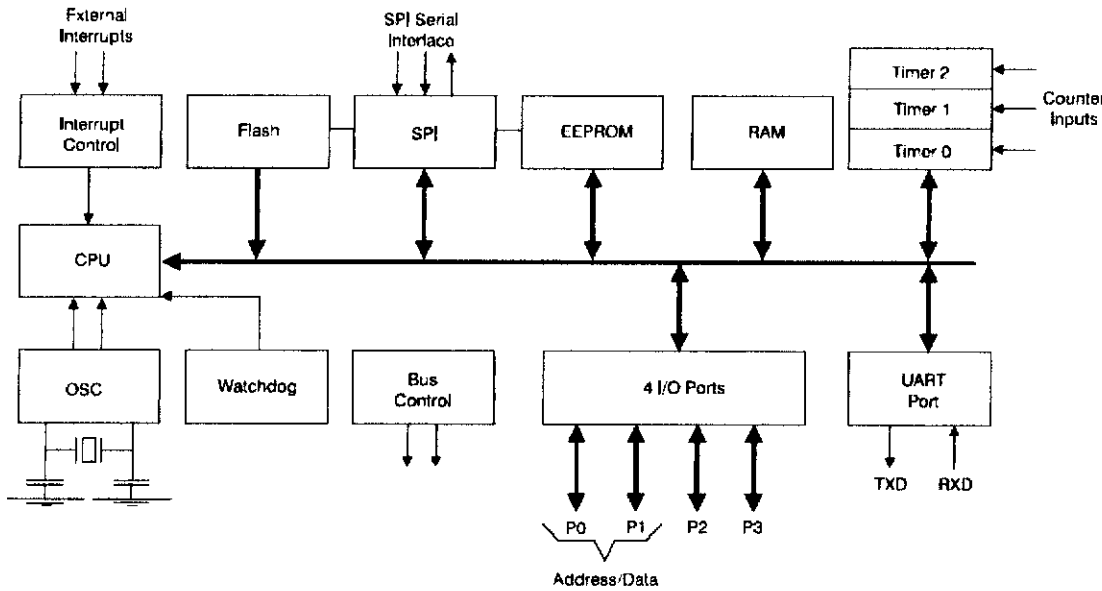
ขา Vcc ใช้สำหรับการต่อไฟเลี้ยง +5 โวลท์

ขา GND เป็นขากราวด์ สำหรับการต่อกราวด์ของระบบ

ขาพอร์ต 0 (P0.0-P0.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาท์พุต สำหรับใช้งานทั่วๆไป ถ้าหากต้องการให้ขาพอร์ต 0 ขาใดขาหนึ่งเป็นอินพุตสามารถทำโดยการเขียนข้อมูล "1" ไปยังแต่บิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนี้ขาพอร์ตนี้ยังถูกใช้งานติดต่อกับขาแอดเดรสไบต์ของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้กระบวนการมัลติเพล็กซ์เข้าช่วย เพื่อสลับการทำงานเป็นได้ทั้งขาติดต่อแอดเดรสและขาข้อมูล



รูปที่ 2.1 แสดงโครงสร้างของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช



รูปที่ 2.2 โครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

FLASH MCU FEATURE SUMMARY

FEATURES	AT89C1051	AT89C2051	AT89C51	AT89LV51	AT89C52	AT89LV52	AT89C55	AT89S8252
FLASH (BYTES)	1K	2K	4K	4K	8K	8K	20K	8K
RAM (BYTES)	64	128	128	128	256	256	256	256
ON-CHIP EEPROM	0	0	0	0	0	0	0	2K
SPI INTERFACE	NO	NO	NO	NO	NO	NO	NO	YES
IN-SYSTEM PROGRAMMING	YES	YES	YES	YES	YES	YES	YES	YES
TIMERS/COUNTERS	1	2	2	2	3	3	3	4
SERIAL DOWN-LOADING	NO	NO	NO	NO	NO	NO	NO	YES
DATA POINTER	1	1	1	1	1	1	1	2
SECURITY LOCK BITS	2	2	3	3	3	3	3	3
WATCHDOG TIMER	-	-	-	-	-	-	-	YES
SERIAL UART	-	YES	YES	YES	YES	YES	YES	YES

รูปที่ 2.3 แสดงรายละเอียดบางส่วนของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

ขาพอร์ต1 (P1.0-P1.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้อินพุตและเอาท์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการให้ขาพอร์ต 1 ขาใดขาหนึ่งเป็นอินพุตสามารถทำโดยการเขียนข้อมูล "1" ไปยังแต่บิตของพอร์ตที่ต้องการติดต่อด้วย นอกจากนั้นในอนุกรม AT98SXX จะใช้ขาของ P1.0 เป็นขาของอินพุตสำหรับนับค่าของไทเมอร์ 2 และ P1.1 เป็นขาอินพุตทริกเกอร์ของไทเมอร์ 2 ในขณะที่ขา P1.4 –P1.7 เป็นขาสำหรับเชื่อมต่อแบบ SPI เพื่อทำการโปรแกรมข้อมูลในระบบ

ขาพอร์ต 2 (P2.0-P2.7) มี 8 ขา แต่ละขาสามารถให้เป็นได้ทั้งอินพุตและเอาท์พุต ถ้าหากต้องการให้ขาพอร์ต 2 ขาใดขาหนึ่งเป็นอินพุตสามารถทำโดยการเขียนข้อมูล "1" ไปยังบิตของพอร์ตที่ต้องการติดต่อด้วยส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ตอินพุตได้

ขาพอร์ต 3 (P3.0-P3.7) มี 8 ขา แต่ละขาสามารถให้เป็นได้ทั้งอินพุตและเอาท์พุต ถ้าหากต้องการให้ขาพอร์ต 2 ขาใดขาหนึ่งเป็นอินพุตสามารถทำโดยการเขียนข้อมูล "1" ไปยังแต่บิตของพอร์ตที่ต้องการติดต่อด้วยงผลให้ขาพอร์ตนั้นีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ต 3 ยังมีหน้าที่การใช้งานพิเศษ ดังนี้

P3.0 ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

P3.1 ใช้เป็นขาอินพุตสำหรับส่งข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

P3.2 ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 0 หรือขา INTO

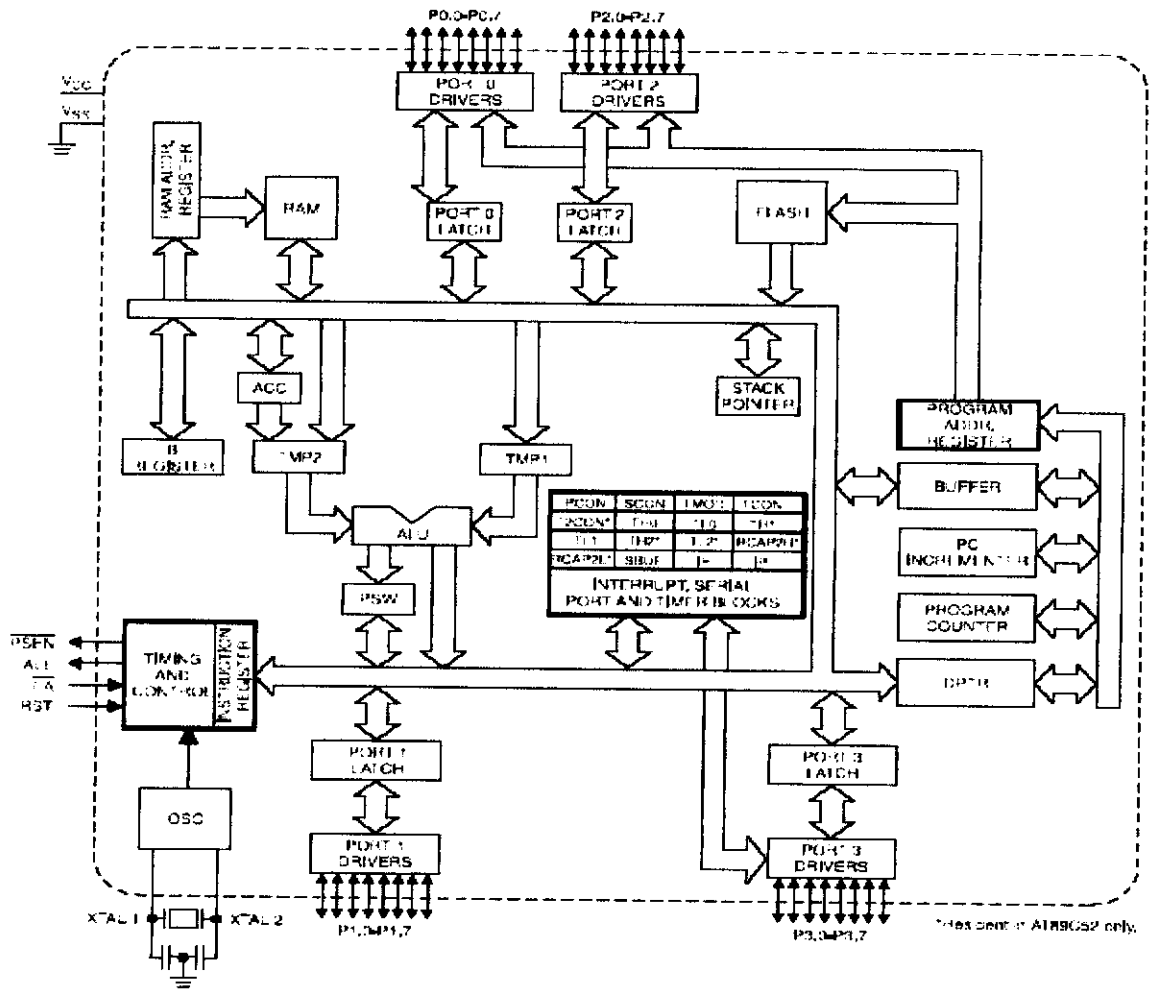
P3.3 ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขา INTI

P3.4 ใช้เป็นขาอินพุตสำหรับรับสัญญาณ ไทเมอร์จากภายนอกช่อง 0 หรือขา TO

P3.5 ใช้เป็นขาอนพุตสำหรับรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขาTI

P3.6 ใช้เป็นขาสัญญาณ WR.ในกรณีที่เชื่อมต่อกับหน่วยความจำภายนอก

P3.7 ใช้เป็นขาสัญญาณ RD ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก



รูปที่ 2.4 แสดงรายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์ MCS-51 ของ ATMEL

ขารีเซต (Reset) ใช้ในการรีเซตการทำงานไมโครคอนโทรลเลอร์ โดยในการป้อนสัญญาณเพื่อรีเซตสถานะที่ขานี้ต้องอยู่ในระดับรีเซตอย่างน้อย 2 แมชีนไซเกิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างปกติ

ขา ALE/PROG (Address Latch Enable/Program pulse input) เป็นขาที่ใช้ในการควบคุมการแลตซ์ของขาพอร์ต 0 เมื่อมีการใช้งานหน่วยความจำภายนอก นอกจากนั้นขานี้ยังใช้เป็นขาสำหรับรับพัลส์ของการโปรแกรมสำหรับโปรแกรมข้อมูลลงใน ไมโครคอนโทรเลอร์ MCS-51 ในรุ่นที่มีหน่วยความจำโปรแกรมเป็นแบบอีพรอม

ขา PSEN (Program Store Enable) ขานี้ใช้ในการส่งสัญญาณเพื่อร้องขอติดต่อกับหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอก ตัวไมโครคอนโทรลเลอร์จะส่งสัญญาณออกมาที่ขานี้ 2 ครั้ง ในแต่ละแมชชีนไซเกิล แต่ถ้าหากติดต่อกับหน่วยความจำข้อมูลภายนอก ขานี้จะไม่มีการส่งสัญญาณใดๆออกมา

ขา EA/ Vpp (External Access enable/ Programming voltage input) ใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมภายนอก หรือภายในตัวไมโครคอนโทรลเลอร์ ถ้าหากขานี้เป็น "0"เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมแต่หากขานี่เป็น "1" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายในตัวมโครคอนโทรลเลอร์ นอกจากนี้ ที่ขานี้ยังใช้เป็นขาอินพุตสำหรับรับแรงดันไฟสูงสำหรับการ โปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์ สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชต้องการแรงดันสำหรับการโปรแกรม +12V

ขา XTAL1 และ XTAL2 เป็นขาสำหรับต่อคริสตอลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์

โครงสร้างของการทำงานของพอร์ต

ไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีพอร์ตให้ใช้งานทั้งสิ้น 4 พอร์ตคือ พอร์ต 0 ถึง พอร์ต 3 แต่ละพอร์ตมีขนาด 8 บิต เป็นพอร์ตแบบ 2 ทิศทาง กล่าวคือ สามารถเป็นได้ทั้งอินพุตสำหรับรับสัญญาณข้อมูลเข้าและเอาต์พุตสำหรับส่งสัญญาณข้อมูลออก ทุกพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีวงแลตซ์และวงจรขับตลอดจนบัฟเฟอร์อินพุต ดังแสดงให้เห็นในสถาปัตยกรรมรูปที่ 2-3

ที่พอร์ต 0 และพอร์ต 2 จะใช้งานเป็นพอร์ตอินพุตและเอาต์พุตสำหรับงานทั่วไป และใช้ในการติดต่อกับหน่วยความจำภายนอก สำหรับพอร์ต 3 ทั้งพอร์ตและพอร์ต 1 บางขานอกจากจะใช้เป็นขาพอร์ตอินพุตเอาต์พุตตามปกติแล้วยังสามารถใช้งานในหน้าที่พิเศษได้อีก ขึ้นอยู่กับว่าเป็นไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีวงจรแลตซ์เบอร์ใด ดังสรุปได้ในตารางที่ 2.1

ในรูปที่ 2.5 แสดงวงจรภายในของแต่ละพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช โดยในรูปที่ 2.5 (A) เป็นวงจรของพอร์ต 0 วงจรแฟลชแต่ละบิตในแต่ละพอร์ตก็คือวงจร

ดีฟลิฟลอปนั่นเอง การอ่านค่าสถานะของวงจรแลตช์สามารถกระทำได้อย่างอิสระด้วยสัญญาณที่แยกจากกันนั่นคือสัญญาณอ่านข้อมูลจากขาพอร์ต และสัญญาณอ่านข้อมูลจากวงจรแลตช์ ส่วนการเขียนข้อมูลมายังพอร์ตต้องส่งสัญญาณมายังขา CLK ของดีฟลิปฟลอบในขณะที่ข้อมูลจะผ่านมาทางขาบัสข้อมูลภายในเข้าสู่ขา D ของดีฟลิฟลอป

ที่พอร์ตนี้มีวงจรมัลติเพล็กซ์สำหรับกำหนดลักษณะการทำงานของพอร์ตว่าต้องการใช้งานเป็นขาพอร์ตอินพุตเอาต์ปกติหรือใช้ในการติดต่อกับหน่วยความจำภายนอกไมโครคอนโทรลเลอร์

เนื่องจากที่ขาพอร์ต 0 ไม่มีวงจรพูลอัปภายในหากมีการนำพอร์ต 0 ไปใช้งานพอร์ตอินพุตจะต้องต่อตัวต้านทานพูตอัปภายนอกเข้าที่ขาพอร์ต 0 ทุกขาด้วย

ในรูปที่ 2-5 (B) เป็นวงจรของพอร์ต 1 ซึ่งมีลักษณะโดยทั่วไปคล้ายกับพอร์ต 0 หากแต่ไม่มีวงจรมัลติเพล็กซ์ เนื่องจากพอร์ตนี้จะไม่ใช้ในการติดต่อกับหน่วยความจำภายนอก แต่จะมีวงจรพูตอัปภายในที่แต่ละบิตของพอร์ตนี้แทน สำหรับรายละเอียดของวงจรพุตอัปแสดงในรูปที่ 2-6

ในรูปที่2-5 (C) เป็นวงจรภายในของพอร์ต 2 จะคล้ายกับพอร์ต 0 มากต่างกันเพียงมีวงจรพุร์ตอัปเพิ่มเติมเข้ามา ส่วนในรูปที่ 2-5 (D) เป็นวงจรภายในของพอร์ต 3 จะเห็นได้ว่าคล้ายกับพอร์ต 1 มีการเพิ่มเติมวงจรบัฟเฟอร์และวงจรอินพุตเอาต์พุตเมื่อทำงานในฟังชั่นพิเศษเข้ามา เนื่องจากพอต 3 สามารถนำไปใช้งานในหน้าที่พิเศษได้ทุกขา

2.3 การใช้งานเป็นพอร์ตอินพุต

เนื่องจากพอร์ตทั้งหมดของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถเป็นได้ทั้งอินพุตและเอาต์พุต ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะการทำงานให้แก่พอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

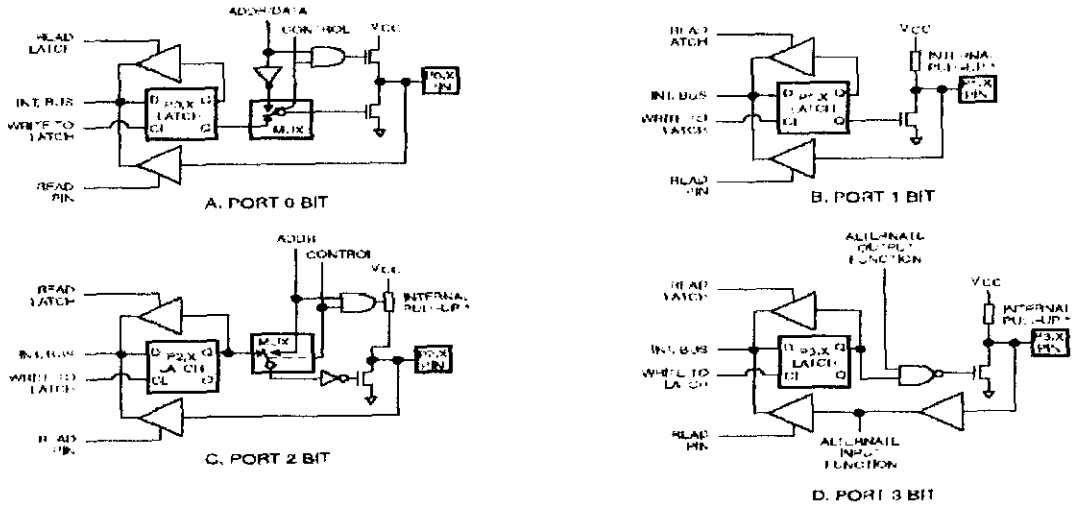
ในการกำหนดให้เป็นพอร์ตอินพุต ต้องเริ่มต้นด้วยการเขียนข้อมูล "1" มาที่แต่ละบิตที่ต้องการใช้งานเป็นอินพุต เพื่อหยุดการทำงานของเฟตที่ใช้ในการขับสัญญาณเอาต์พุตของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ตเชื่อมต่อเข้ากับวงจรพุตอัปภายในโดยตรง ส่งผลให้ขาพอร์ตนั้นมีลอจิกเป็น "1" สามารับสัญญาณลอจิก "0" จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ต แล้วรอซีพียูมาอ่านค่าเข้าไป เมื่อเป็นเช่นนี้ อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ตอินพุตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชควรกำหนดให้ทำงานในภาวะลอจิก "0" และสะดวกที่สุด (ซึ่งในปัจจุบันอุปกรณ์อินพุตที่เชื่อมต่อ ไมโครคอนโทรลเลอร์แทบทั้งหมดทำงานที่ลอจิก "0" แล้ว)

2.4 การใช้งานเป็นพอร์ตเอาต์พุต

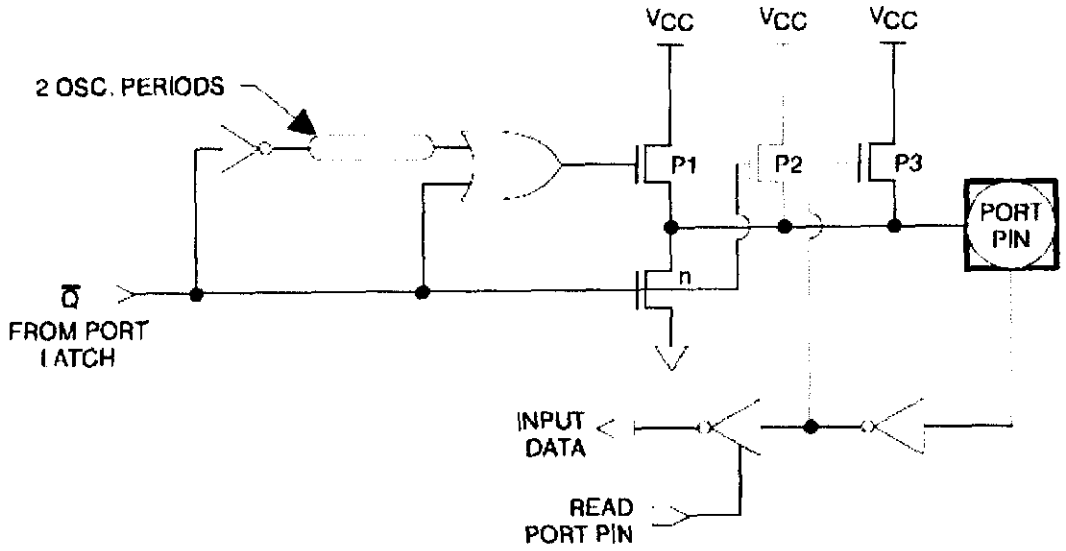
โดยปกติแล้ว ขาพอร์ตจะกำหนดให้มีลักษณะเป็นเอาต์พุตอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปอย่างง่ายดายและตรงไปตรงมา กล่าวคือ เมื่อต้องการส่งข้อมูล "0" ออกไปก็สามารถส่งออกไปได้เลยซึ่งก็จะส่งผลต่อไปขับเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่ทำงานก็จะเกิดลอจิก "0" ขึ้น ในทางตรงกันข้ามหากต้องการส่งข้อมูล "1" ออกไปก็สามารถเขียนข้อมูล "1" ออกไปได้เช่นกัน ทำให้ที่ขาพอร์ตนั้นเชื่อมต่อกับวงจรพูลอัพที่อยู่ภายในเกิดเป็นลอจิก "1" ขึ้น โดยถ้าเป็นอินพุตจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่เป็นเอาต์พุตจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์ เว้นแต่ต้องการให้มาตรวจที่ส่งออกมาที่อินพุต

ขา	เบอร์ของไมโครคอนโทรลเลอร์	หน้าที่พิเศษ
P1.0	AT89C52/AT89Sxx	ขา T2 เป็นขาอินพุตนับค่าของไทเมอร์/เคาร์เตอร์ 2 และเป็นขา
P1.1	AT89C52/AT89Sxx	และควบคุมทิศทางของสัญญาณ
P1.4	AT89Sxx	ขา SS (Slave Select) เป็นขาเลือกติดต่อในกรณีที่ไมโครคอนโทรลเลอร์เป็นอุปกรณ์สเลฟ ในระบบการติดต่อแบบ SPI
P1.5	AT89Sxx	ขา MOSI (Master data output, Slave data input) ใช้ในการติดต่อกับพอร์ต SPI
P1.6	AT89Sxx	ขา MISO (Master data input, Slave data output) ใช้ในการติดต่อกับพอร์ต SPI
P1.7	A T89Sxx	ขา SCK (Master clock output) เป็นขาสัญญาณนาฬิกาของการติดต่อกับพอร์ต SPI

ตารางที่ 2.1 หน้าที่พิเศษของพอร์ต 1 ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชของ Atmel



รูปที่ 2.5 แสดงวงจรภายในของพอร์ตทุกพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช



รูปที่ 2.6 แสดงวงจรพูลอัพภายในพอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

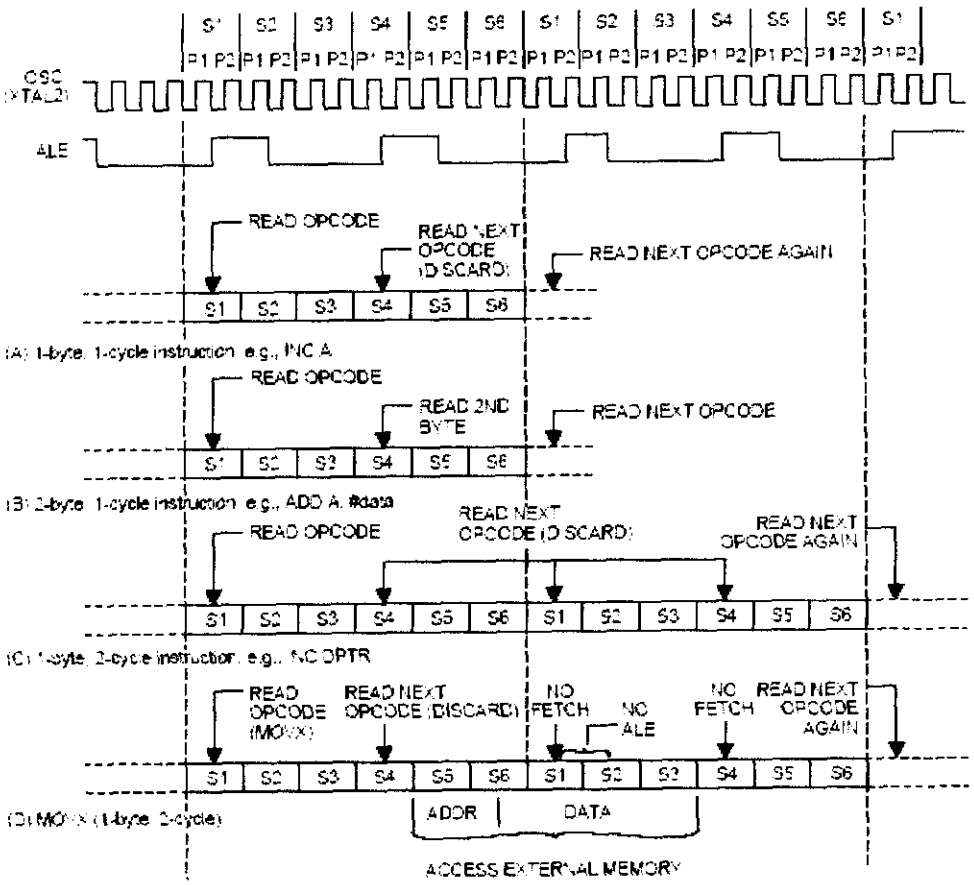
เมื่อมีการใช้งานพอร์ตเอาต์พุตแต่ละขาสามารถจ่ายกระแสได้สูงสุด (Source current) ประมาณ 10 มิลลิแอมป์ และทุกขารวมกันได้สูงสุดไม่เกิน 26 มิลลิแอมป์ ที่พอร์ต 0 และ 15 มิลลิแอมป์ที่พอร์ต 1-3 ในกรณีที่ใช้งานทุกพอรีตร่วมกันสามารถจ่ายกระแสรวมกันได้สูงสุด 71 มิลลิแอมป์ ดังนั้นการใช้งานเป็นพอร์ตเอาต์พุต เพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัฟเฟอร์ทางเอาต์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง

2.5 การอ่านค่าทางลอจิก

ในไมโครคอนโทรเลอร์ MCS-51 แบบเฟลชสามารถอ่านค่าลอจิกจากพอร์ตได้ 2 ลักษณะ คือ ในกรณีที่ต่อขาเบสทรานซิสเตอร์ชนิด NPN และขาอิมิตเตอร์ต่อลงกราวด์ หากมีการส่งข้อมูล "1" ไปยังทรานซิสเตอร์ จะทำให้ลอจิกที่เอาต์พุตเป็น "0" เป็นต้น ดังนั้นการเลือกใช้งานอุปกรณ์ที่มาต่อต้องเลือกให้เหมาะสมด้วย

2.6 จังหวะการทำงานของไมโครคอนโทรเลอร์ MCS-51

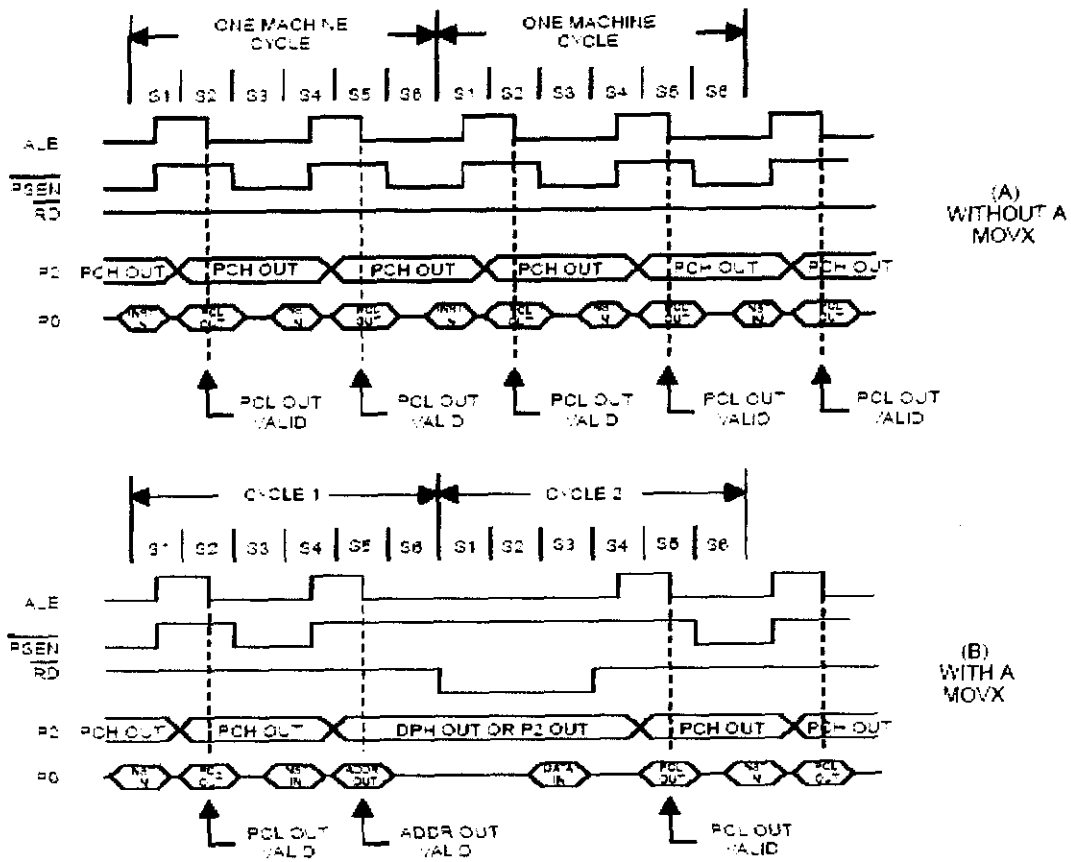
ในการใช้งานไมโครคอนโทรเลอร์ MCS-51 จะต้องทำความเข้าใจถึงจังหวะการทำงานของซีพียูและลำดับการขั้นตอนการประมวลผลทางคำสั่ง ในการประมวลผลคำสั่งของซีพียูจะมีขั้นตอนหลัก ๆ อยู่ 2 ขั้นตอนคือ **กระบวนการเฟตช์ (fetch)** เป็นการเรียกคำสั่งออกจากหน่วยความจำโปรแกรมแล้วทำการแปลงรหัสคำสั่งนั้นเป็นภาษาเครื่องเพื่อเตรียมการประมวล ขั้นต่อมาคือ **กระบวนการเอ็กซิคิวต์ (Execute)** เป็นกระบวนการที่ทำตามคำสั่งที่กำหนดหรือตามที่เฟตช์ขึ้นมาโดยกระบวนการก่อนหน้านี้ เมื่อทำการเอ็กซิคิวต์คำสั่งเรียบร้อยแล้วก็จะไปเริ่มกระบวนการเฟตช์คำสั่งใหม่ต่อไป



รูปที่ 2.7 แสดงไซเกิลการทำงานของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

เมื่อเริ่มจ่ายไฟให้แก่ไมโครคอนโทรเลอร์จะเกิดการรีเซตในลักษณะที่เรียกว่า **เพาเวอร์-ออนรีเซต** (Power on reset) ซีพียูเริ่มต้นทำงานที่แอดเดรส 0000H ของหน่วยความจำโปรแกรม จังหวะการทำงานของซีพียูจะเป็นไปตามรูปแบบ โดยได้รับการกำหนดมาจากการทำงานหรือแมชชีนไซเกิล (machine cycle) ในรูปที่ 2.7 เป็นไดอะแกรมการทำงานของไมโครคอนโทรเลอร์ MCS-51 โดยใน 1 รอบการทำงานหรือแมชชีนไซเกิลจะแบ่งย่อยออกเป็น 6 สเตต ในแต่ละสเตตมีค่าคาบเวลาเท่ากับ 2 คาบเวลา ถ้าสัญญาณที่ความถี่ 12 MHz จะมีคาบเวลาเท่ากับ 1 ms คาบเวลาทั้งสองภายในหนึ่งเตตจะเรียกว่า เฟส 1 และ เฟส 2

ในรูปที่ 2.7 จะเป็นการเอ็กซิคิวต์คำสั่งที่ใช้เวลา 1 แมชชีนไซเกิล เริ่มต้นที่สเตต 1 จะเป็นการอ่านค่าออปโค้ด อันเป็นกระบวนการแลตช์ค่าของออปโค้ดส่งไปให้รีจิสเตอร์คำสั่ง (instruction register: IR) การเฟตช์จะเกิดขึ้นในสเตตที่ 4 ภายในแมชชีนไซเกิลเดียวกัน ในกรณีที่เป็นเพียงคำสั่งไบต์เดียว การเฟตช์ครั้งที่2 ภายในแมชชีนไซเกิลจะถูกตัดทิ้งไปในคำสั่งที่มีการใช้เวลา 1 ไซเกิล จะสิ้นสุดการทำงานลงที่สเตตที่ 6 ของแมชชีนไซเกิลเดียวกัน



รูปที่ 2.8 แสดงเวลาการติดต่อและเข้าถึงหน่วยความจำโปรแกรมภายนอกของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

ในกรณีที่คำสั่งใช้เวลา 2 แมชชีนไซเกิล การทำงานของคำสั่งนั้นจะสิ้นสุดลงในสเตตที่ 6 ของแมชชีนไซเกิลที่สอง ดังรูป 2.8 สำหรับการกระทำคำสั่ง MOVX ซึ่งคำสั่งที่มีขนาด 1 ไบต์ 2 ไซเกิลจะไม่มีการเฟตช์เกิดขึ้นในไซเกิลที่สองของคำสั่ง MOVX นี้ เนื่องจากซีพียูจะไปทำการติดต่อกับหน่วยความจำภายนอก ซึ่งจะเห็นว่าในการเอ็กซิคิวต์จะไม่ได้ขึ้นอยู่กับว่าทำการติดต่อหน่วยความจำโปรแกรมภายในหรือภายนอก

ในรูปที่ 2.8แสดงสัญญาณและเวลาการเข้าถึงหน่วยความจำโปรแกรมภายนอก โดยในรูปที่เป็นไดอะแกรมเวลาที่ยังไม่มีการกระทำคำสั่ง MOVX สัญญาณที่ขา ALE และ PSEN จะเกิดการแอคทีฟ 2 ครั้งภายในหนึ่งแมชชีนไซเกิล ในทุกครั้งที่ ALE เกิดการแอคทีฟที่พอร์ต 0 (P0) ค่าของรีจิเตอร์ PC ในไบต์ต่ำออกมา ในขณะที่พอร์ต 2 ก็จะมีค่าของไบต์สูงเพื่อชี้ไปยังแอดเดรสต่อไปดำเนินการ สำหรับขา PSEN ก็จะเกิดการแอคทีฟเมื่อมีการติดต่อกับหน่วยความจำภายนอก ในกรณีที่กระทำคำสั่ง MOVX เพื่อติดต่อหน่วยความจำภายนอก ที่ขา PSEN จะไม่เกิดการแอคทีฟ 2 ครั้งภายในเวลา 1 แมชชีนไซเกิล เนื่องจากบัสแอดเดรสและบัสข้อมูลจะถูกใช้ในการติดต่อหน่วยความจำภายนอกแทน แต่สำหรับสัญญาณ ALE ยังคงแอคทีฟตามจังหวะการทำงานเหมือนเดิม

จากไดอะแกรมเวลาสามารถสรุปได้ว่า ในการทำงานหนึ่งแมชชีนไซเกิล ซีพียูในไมโครคอนโทรเลอร์ MCS-51 จะใช้เวลาในการทำงาน 12 คาบเวลาของสัญญาณนาฬิกา นั่นคือเวลาในการทำงาน 1 ไซเกิลมีค่าเท่ากับ 1 ms หรือมีความเร็วในการทำงานเท่ากับ 1 MHz ในกรณีที่ใช้ความถี่สัญญาณนาฬิกาเท่ากับ 12 MHz ดังนั้นถ้าต้องการทราบความเร็วของการทำงานของไมโครคอนโทรเลอร์ สามารถหาจากค่าความถี่ของสัญญาณนาฬิกาหารด้วย 12 และถ้าต้องการหาค่าเวลาของ 1 รอบการทำงาน สามารถหาได้จาก

ความเร็วในการทำงานของไมโครคอนโทรลเลอร์ = ความถี่ของสัญญาณนาฬิกา/12

เวลา 1 แมชชีนไซเกิล = 1/ความเร็วในการทำงานของไมโครคอนโทรลเลอร์

2.7 การเข้าถึงข้อมูลของไมโครคอนโทรลเลอร์ MCS-51

การเข้าถึงข้อมูล(Addressing) เป็นวิธีการที่ไมโครคอนโทรลเลอร์ใช้ในการระบุตำแหน่งเพื่อเข้าถึงข้อมูลที่ต้องการดำเนินการ สำหรับไมโครคอนโทรลเลอร์ MCS-51 มีวิธีการเข้าถึงข้อมูลอยู่ 5 แบบ คือ

1.แบบทันทีทันใด (Immediate addressing Mode)

2.แบบโดยตรง (Direct Addressing Mode)

3.แบบผ่านรีจิสเตอร์ (Register Addressing Mode)

4.แบบโดยอ้อม (Indirect Addressing Mode)

5.แบบอินเด็กซ์หรือแบบโดยอ้อมผ่านค่าดัชนี (Index Addressing Mode)

2.7.1การเข้าถึงข้อมูลแบบทันทีทันใด

เป็นวิธีการเข้าถึงข้อมูลที่มีความซับซ้อนน้อยที่สุด สามารถทำความเข้าใจได้ง่าย เนื่องจากสามารถโอนย้ายหรือเข้าถึงค่าข้อมูลตรงๆเช่น

```
MOV R0, #10H          ; เป็นการนำค่า 10 ไปเก็บไว้ในรีจิสเตอร์ R0
```

2.7.2 การเข้าถึงข้อมูลแบบโดยตรง

เป็นการเข้าถึงข้อมูลโดยใช้หมายเลขตำแหน่งของหน่วยความจำภายในต้นทางเพื่อเรียกหรือโอนย้ายข้อมูลได้โดยตรง แล้วนำข้อมูลจากหน่วยความจำภายในต้นทางไปเก็บยังรีจิสเตอร์หรือหน่วยความจำปลายทาง การเข้าถึงแบบนี้จะใช้กับหน่วยความจำข้อมูลภายในของไมโครคอนโทรลเลอร์ MCS-51 เท่านั้น สามารถอ้างแอดเดรสได้สูงสุด 256 ตำแหน่ง ซึ่งก็คือแอดเดรสของหน่วยความจำภายในนั่นเองซึ่งมีวิธีการดังนี้

```
MOV R0, 30H           ; เป็นการนำค่าที่อยู่ในแอดเดรส 30H มาเก็บไว้ที่รีจิสเตอร์ R0
```

2.7.3 การเข้าถึงข้อมูลแบบผ่านรีจิสเตอร์

การเข้าถึงข้อมูลแบบนี้จะคล้ายกับแบบโดยตรง ต่างกันตรงที่ข้อมูลต้นทางจะถูกเก็บไว้ที่รีจิสเตอร์แบงก์ R0-R7 ส่วนรีจิสเตอร์ปลายทางต้องเป็นรีจิสเตอร์ A หรือแอคคิวมูเลเตอร์ เช่น

```
MOV A,Rn              ;Rn เป็นรีจิสเตอร์แบงก์ใด ๆ
```

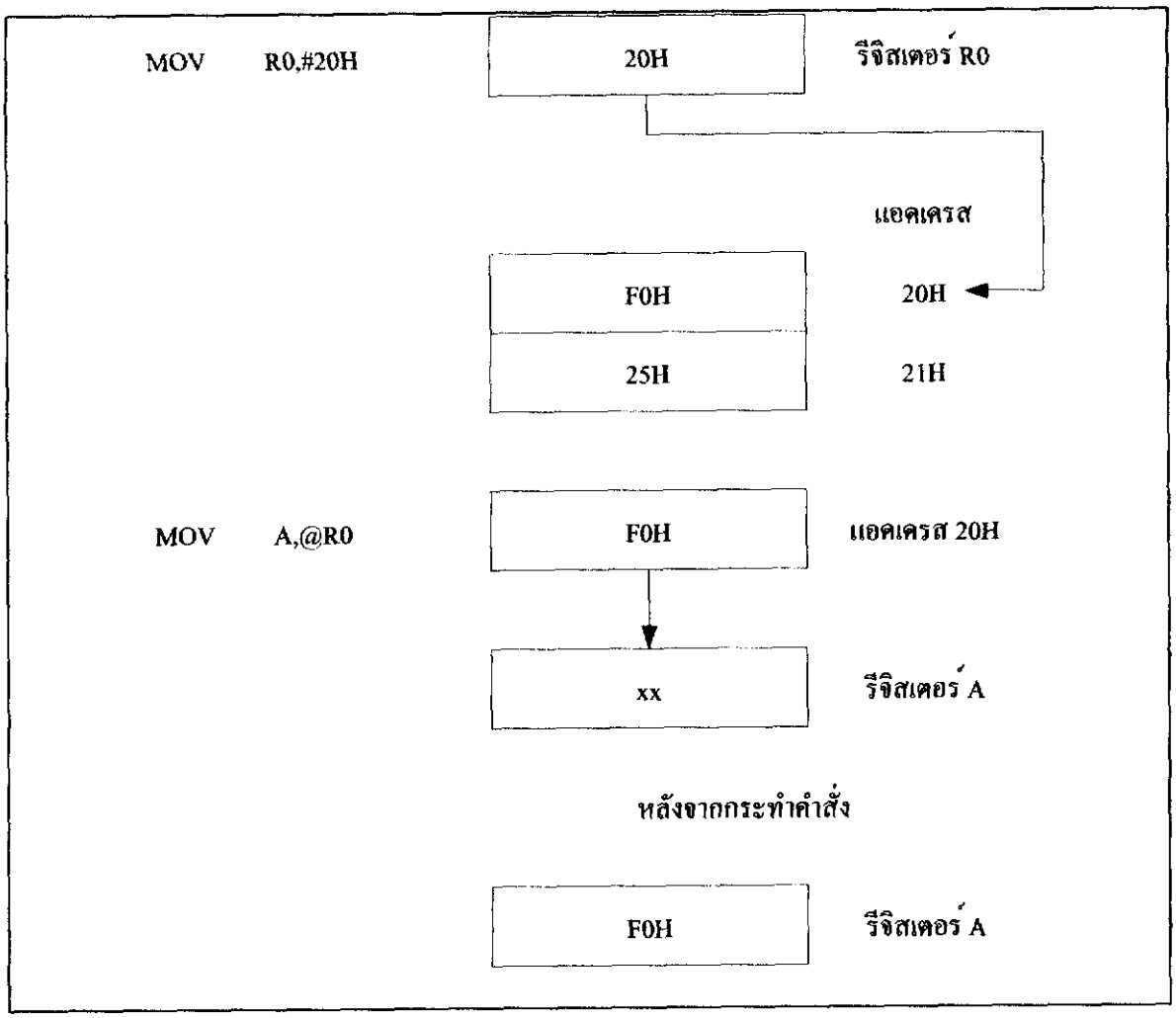
2.7.4 การเข้าถึงข้อมูลแบบโดยอ้อม

การเข้าถึงแบบนี้มีความซับซ้อนกว่าแบบโดยตรง โดยค่าข้อมูลของแอดเดรสต้นทางจะถูกเก็บไว้ในรีจิสเตอร์ R0 หรือ R1 หรือ DPTR หรือใช้รีจิเตอร์ R0, R1 และ DPTR เป็นตัวชี้แอดเดรสของหน่วยความจำแทนและต้องใส่สัญลักษณ์ @ หน้ารีจิสเตอร์ที่เป็นตัวชี้แอดเดรสต้นทางด้วย มีตัวอย่างดังนี้

```
MOV A,@R0
```

หรือ

```
MOV A,@DPTR
```



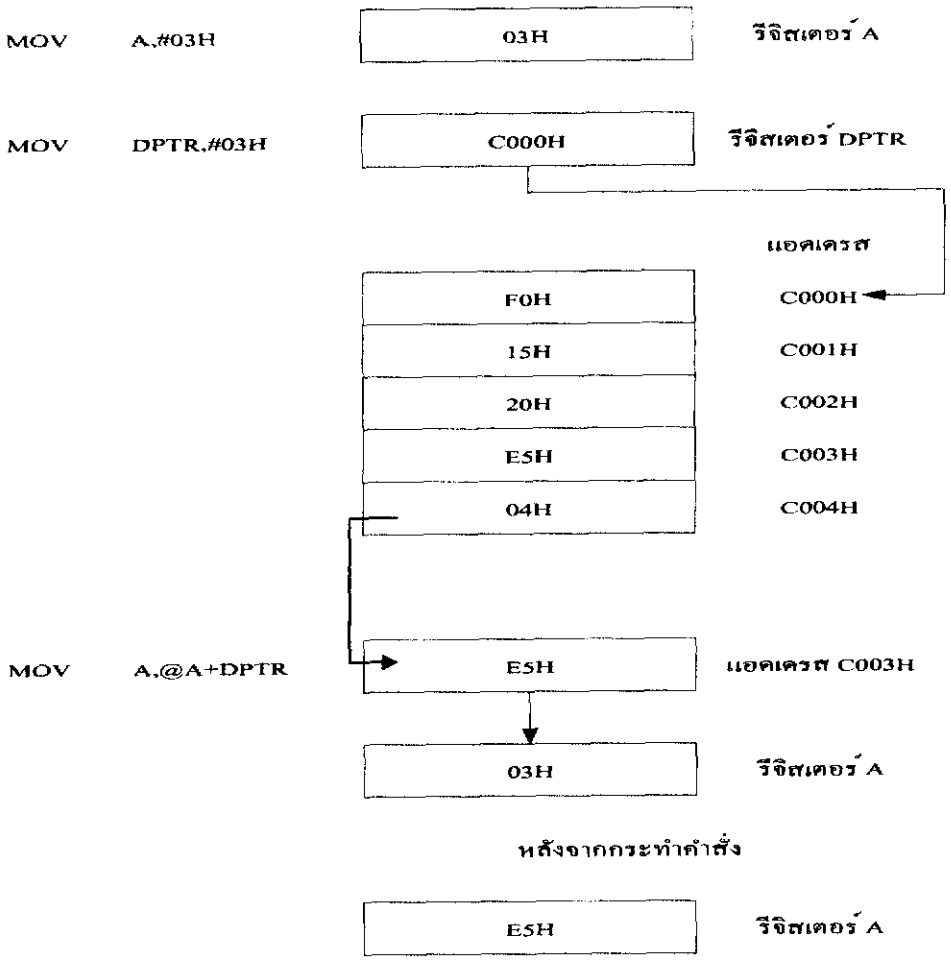
รูปที่ 2.9 แสดงกระบวนการเข้าถึงข้อมูลแบบโดยอ้อม

สำหรับการใช้รีจสเตอร์ R0 และ R1 ในการเป็นตัวชี้แอดเดรส ทำให้สามารถอ้างแอดเดรสได้เพียง 256 ตำแหน่งเท่านั้น เนื่องจากรีจิสเตอร์ R0 และ R1 มีขนาดเพียง 8 บิต ถ้าหากต้องการให้สามารถอ้างแอดเดรสให้ได้มากกว่านี้ก็สามารถอ้างได้โดยรีจิสเตอร์ DPTR ที่สามารถอ้างแอดเดรสได้สูงถึง 65,536 ตำแหน่ง โดยปกติแล้วจใช้รีจิสเตอร์ DPTR ในการชี้ตำแหน่งแอดเดรสเมื่อต้องการติดต่อกับหน่วยความจำภายนอก

2.7.5 การเข้าถึงข้อมูลแบบอินเด็กซ์

วิธีการเข้าถึงแบบนี้มีความซับซ้อนมากที่สุด เนื่องจากต้องใช้รีจิสเตอร์ 2 ตัวทำงานร่วมกันแล้วใช้คำสั่งการโอนย้ายข้อมูล MOV A,@A+DPTR หรือ MOV A,@A+PC มาจัดการโอนย้ายข้อมูล รีจิสเตอร์ DPTR และ PC ถูกนำมาใช้เป็นแอดเดรสเริ่มต้น และใช้ค่าที่เก็บอยู่ในรีจิสเตอร์ A เป็นตัวเลื่อนแอดเดรสต้นทาง จากนั้นทำการโอนย้ายข้อมูลมาจากแอดเดรสที่ถูกชี้ด้วยผลรวมของ

ค่ารีจิสเตอร์ A ในรูปที่ แสดงกระบวนการถึงข้อมูลแบบนี้ การเข้าถึงข้อมูลแบบอินเด็กซ์หรือแบบโดยอ้อมผ่านค่าดัชนี ถึงแม้ว่าจะมีความซับซ้อนที่มากกว่าทุกวิธีที่กล่าวมาก่อนหน้านี้แต่ก็เป็นประโยชน์อย่างมากในการเปิดตารางข้อมูลหรือ Look Up Table



รูปที่ 2.10 แสดงการเข้าถึงข้อมูลแบบอินเด็กซ์

2.8 ไทเมอร์/เคาเตอร์ของไมโครคอนโทรลเลอร์ MCS-51

ไทเมอร์/เคาเตอร์ (Time/counter) เป็นอีกหนึ่งส่วนประกอบที่สำคัญของไมโครคอนโทรลเลอร์ เนื่องจากการทำงานของไมโครคอนโทรลเลอร์จะต้องมีการเก็บและตรวจสอบค่าของเวลาและจำนวนของสัญญาณนาฬิกาอยู่ตลอดเวลา ทั้งนี้เพื่อประโยชน์ในการสร้างฐานเวลา สร้างสัญญาณพัลส์ เปรียบเทียบค่าเวลาหรือเปรียบเทียบค่าของการนับ รวมไปถึงการกำหนดอัตราเร็วในการสื่อสารข้อมูลของพอร์ตอนุกรมด้วย

ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ AT89C51 มีวงจรไทเมอร์/เคาเตอร์ขนาด 16 บิต 2 ตัว โดยค่าของวงจรไทเมอร์/เคาเตอร์นี้จะเก็บไว้ในรีจิสเตอร์ขนาด 16 บิต ที่มีชื่อ ไท

เมอร์ 0 (Timer 0) และ ไทเมอร์ 1 (timer 1) เรียกสั้นๆว่า T0 และ T1 ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ AT89C52/55 และอนุกรม AT89Sxx จะมีไทเมอร์/เคาเตอร์ถึง 3 ตัวคือ มีไทเมอร์ 2(Time 2 : T2) เพิ่มเติม โดยรีจีสเตอร์ไทเมอร์/เคาเตอร์ทั้งสามตัวสามารถกำหนดให้ทำงานเป็นตัวตั้งเวลาหรือไทเมอร์และตัวนับหรือเคาเตอร์ได้อย่างอิสระต่อกัน

2.8.1 การทำงานเป็นไทเมอร์

เมื่อกำหนดให้ทำงานเป็นตัวตั้งเวลาหรือไทเมอร์ ค่าของรีจีสเตอร์จะเพิ่มขึ้นในทุกๆ แมชีนไซเกิล ดังนั้นเมื่อทำงานเป็นไทเมอร์ รีจีสเตอร์จะทำการนับค่าของแมชีนไซเกิลนั่นเอง และเนื่องจากแมชีลไซเกิลประกอบด้วยคาบเวลาของวงจรกำเนิดสัญญาณนาฬิกา 12 คาบเวลา ดังนั้นอัตราในการนับของรีจีสเตอร์จึงเท่ากับ 1/12 ของความถี่สัญญาณนาฬิกา

2.8.2 การทำงานเป็นเคาเตอร์

เมื่อทำงานเป็นตัวนับหรือเคาเตอร์ ค่าของรีจีสเตอร์จะเพิ่มขึ้นก็ต่อเมื่อมีการเปลี่ยนแปลงของระดับลอจิกจาก "1" เป็น "0" เกิดขึ้นที่ขาอินพุตทางฮาร์ดแวร์ของวงจรไทเมอร์/เคาเตอร์ ซึ่งก็คือขา T0 (P3.4) และขา T1 (P3.5) สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ AT89C51 รวมทั้งขา T2 (P1.0) ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ AT89C52 และอนุกรม AT89Sxx โดยจะมีการสุ่มรับสัญญาณจากขาอินพุตในทุกๆ คาบเวลาที่ 2 ของสเตตที่ 5 (S5P2) ในแต่ละแมชีนไซเกิล

เมื่อสัญญาณอินพุตเปลี่ยนแปลงจาก "1" เป็น "0" เป็นเวลาหนึ่งไซเกิล ในไซเกิลต่อมาค่าของการนับจะเพิ่มขึ้นหนึ่งค่า และจะไปปรากฏในรีจีสเตอร์ภายในคาบเวลาที่ 1 ของสเตตที่ 3 (S3P1) ของแมชีนไซเกิลต่อไปหลังจากที่ตรวจจับพบการเปลี่ยนแปลงที่ขาไทเมอร์อินพุตแล้ว เมื่อเป็นเช่นนี้ในกระบวนการตรวจจับการเปลี่ยนแปลงของสัญญาณอินพุตที่ขาไทเมอร์จะต้องใช้ 2 แมชีนไซเกิล อัตราการนับของเคาเตอร์จึงเท่ากับ 1/24 ของความถี่สัญญาณนาฬิกา ดังนั้น **ความถี่**สูงสุดของสัญญาณอินพุตท่ไทเมอร์/เคาเตอร์ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถตรวจจับได้จึงเท่ากับความถี่ของสัญญาณนาฬิกาหารด้วย 24 ยกตัวอย่าง ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ AT89C51 สามารถใช้สัญญาณนาฬิกาได้สูงสุด 24 MHz ดังนั้นความถี่สูงสุดของสัญญาณอินพุตที่ไทเมอร์/เคาเตอร์สามารถตรวจจับได้คือ 1 MHz

2.8.3 รีจีสเตอร์ที่เกี่ยวข้องกับการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1

ในการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1 ในไมโครคอนโทรลเลอร์ MCS-51 8CH , TL1 มีแอดเดรสอยู่ที่ 8BH และ TH1 มีแอดเดรสอยู่ที่ 8DH รีจีสเตอร์ทั้ง 4 ตัวจะอยู่ในพื้นที่ของรีจีสเตอร์ฟังก์ชั่นพิเศษหรือ SFR รีจีสเตอร์แต่ละตัวมีขนาด 8 บิต แต่การใช้งานโดยทั่วไปมักใช้

ร่วมกันโดยจัดเป็นคู่คือ TL0 กับ TH0 รวมกันเป็นรีจีสเตอร์ Time 0 ขนาด 16 บิต และ TL1 กับ TH1 รวมกันเป็นรีจีสเตอร์ Time 1 ขนาด 16บิต โดยใน TL0 และ TL1 เก็บข้อมูล 8 บิตล่าง ส่วน TH0 และ TH1 เก็บข้อมูล 8 บิตบน รีจีสเตอร์ทั้ง 2 คู่เมื่อนำไปใช้รวมกันจะสามารถเก็บค่าของการนับได้สูงสุด 65,563 หรือ FFFFH เมื่อนับถึงค่านี้แล้วจะวนไปเริ่มนับ 0000H ใหม่และเมื่อเกิดการนับรอบใหม่ บิต TF0 หรือ TF1 ใน
รีจีสเตอร์ TCON ที่ใช้ควบคุมการทำงานของไทเมอร์จะเกิดการเซต เพื่อแจ้งให้ทราบว่า นับเกินค่าสูงสุดแล้ว การเซตบิต TF0 และ TF1 ขึ้นอยู่กับว่าเลือกใช้งานรีจีสเตอร์ไทเมอร์ตัวใด

2.8.4 รีจีสเตอร์ควบคุมการทำงานของไทเมอร์/เคาเตอร์หรือ TCON (Timer/Counter Control Register)

เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 88H ในพื้นที่ของรีจีสเตอร์ SFR สามารถเข้าได้ถึงระดับบิตมีรายละเอียดการทำงานดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 (Timer 1 overflow flag) : เซตด้วยกระบวนการทางฮาร์ดแวร์ เมื่อค่าของรีจีสเตอร์ Timer 1 เกิดการนับเกินหรือเกิดโอเวอร์โฟลว การเคลียร์บิตนี้ทำได้ด้วยกระบวนการทางฮาร์ดแวร์เช่นกันโดยบิตนี้จะเคลียร์เมื่อมีการอินเตอรัปต์เกิดขึ้น

TR1 (timer 1 run control bit) : ใช้ในการเปิดปิดการทำงานของไทเมอร์ 1 (เอ็นเอเบิลหรือดิสเอเบิล) ทำการเซตและเคลียร์ด้วยกระบวนการทางซอร์ฟแวร์ ถ้าต้องการให้ไทเมอร์ 1 ทำงานต้องเซตบิตนี้ให้เป็น "1"

TF0 (Timer 0 overflow flag) : เซตด้วยกระบวนการทางฮาร์ดแวร์ เมื่อค่าของรีจีสเตอร์ Timer 0 เกิดการนับเกินหรือเกิดโอเวอร์โฟลว การเคลียร์บิตนี้ทำได้ด้วยกระบวนการทางฮาร์ดแวร์เช่นกันโดยบิตนี้จะเคลียร์เมื่อมีการอินเตอร์รัปต์เกิดขึ้น

TR0: (timer 0 run control bit): ใช้ในการปิดการทำงานของไทเมอร์ 0 (เอ็นเอเบิลหรือดิสเอเบิล) ทำการเซ็ตและเคลียร์ด้วยกระบวนการซอร์ฟแวร์ ถ้าต้องการให้ไทเมอร์ 0 ทำงานต้องเซตบิตนี้ให้เป็น "1"

IE1 (External Interrupt 1 edge flag) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อสามารถตรวจจับขอบขาของสัญญาณอินเตอร์รัปต์จากภายนอกที่ขาของอินพุตอินเตอร์รัปต์ 1 (INT1) ได้ และจะทำการเคลียร์เมื่อมีการบริการอินเตอร์รัปต์เกิดขึ้น

IT1 (Interrupt 1 type control bit) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์โดยใช้ในการเลือกลักษณะของสัญญาณอินเตอร์รัปต์จากภายนอกที่ต้องการให้ทำการตอบสนองสำหรับขาอินพุตอินเตอร์รัปต์ 1 (INT1) การเซตและเคลียร์ทำได้ด้วยกระบวนการซอฟต์แวร์

"0" เลือกขอบขาลงของสัญญาณ (falling edge)

"1" เลือกระดับลอจิกต่ำ (low level triggered)

IE0 (External Interrupt 0 edge flag) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์ สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อสามารถตรวจจับขอบขาของสัญญาณอินเตอร์รัปต์จากภายนอกที่ขาอินพุตอินเตอร์รัปต์ 0 (INT0) ได้ และจะทำการเคลียร์เมื่อมีการบริการอินเตอร์รัปต์เกิดขึ้น

IT0 (Interrupt 0 type control bit) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์โดยใช้ในการเลือกลักษณะของสัญญาณอินเตอร์รัปต์จากภายนอกที่ต้องการให้ทำการตอบสนองสำหรับขาอินพุตอินเตอร์รัปต์ 0 (INT0) การเซตและเคลียร์ทำได้ด้วยกระบวนการทางซอฟต์แวร์

"0 " เลือกขอบขาลงของสัญญาณ (falling edge)

"1" เลือกระดับลอจิกต่ำ (law level triggered)

2.8.5 รีจีสเตอร์เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์หรือ TMOD (Timer/Counter Mode Control Register)

เป็นรีจีสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 89H ในพื้นที่ของรีจีสเตอร์ SFR ไม่สามารถเข้าถึงได้ในระดับบิต แบ่งการทำงานออกเป็น 2 ส่วนคือ 4 บิตล่าง ใช้ในการเลือกโหมดการ อธิบายการทำงานจะขออธิบายเพียงส่วนเดียวดังนี้

GATE: ใช้เลือกลักษณะการควบคุมการทำงานของไทเมอร์/เคาเตอร์

"0" ไทเมอร์/เคาเตอร์จะทำงานเมื่อบิต TRx ในรีจีสเตอร์ TCON เป็น "1" เรียกการควบคุมแบบนี้ว่าการควบคุมทางซอฟต์แวร์

"1" ไทเมอร์/เคาเตอร์จะทำงานเมื่อบิต TRx ในรีจีสเตอร์ TCON เป็น "1" และสถานะลอจิกที่ขาอินพุตอินเตอร์รัปต์ INT0 และ INT1 เป็น "1" เรียกการควบคุมแบบนี้ว่า การควบคุมทางฮาร์ดแวร์

C/T (Timer or Counter selector) : ใช้เลือกลักษณะการทำงานของไทเมอร์/เคาเตอร์

"0" เลือกให้ทำงานเป็นไทเมอร์ โดยใช้สัญญาณอินพุตจากสัญญาณนาฬิกาภายในไมโครคอนโทรลเลอร์

"1" เลือกให้ทำงานเป็นเคาเตอร์ โดยรับสัญญาณอินพุตทางขา T0 หรือ T1

M1 , M0 (Mode selector bit) : ใช้เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์

"00" เลือกให้ทำงานในโหมดไทเมอร์/เคาเตอร์ 13 บิต

"01" เลือกให้ทำงานในโหมดไทเมอร์/เคาเตอร์ 16 บิต

"10" เลือกให้ทำงานในโหมดไทเมอร์/เคาเตอร์ขนาด 8 บิต แบบตั้งค่าอัตโนมัติ

"11" สำหรับไทเมอร์ 0 เลือกให้ทำงานในโหมดไทเมอร์/เคาเตอร์แยกส่วน โดยแยกเป็นไทเมอร์/เคาเตอร์ 8 บิต 2 ตัว รีจีสเตอร์ TL0 จะได้รับการควบคุมการเปิดปิดจากบิต TR0 ในรีจีสเตอร์ TCON และรีจีสเตอร์ TH0 ซึ่งเป็นไทเมอร์/เคาเตอร์ 8 บิต อีกตัวหนึ่ง จะได้รับการควบคุมจากบิต TR1 ในรีจีสเตอร์ TCON ในกรณีของไทเมอร์ 1 เป็นการสั่งให้ไทเมอร์/เคาเตอร์ 1 หยุดทำงาน (ดิสเอเบิล)

2.8.6 โหมดการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1

ไทเมอร์ 0 และ ไทเมอร์ 1 สามารถเลือกโหมดการทำงานได้ 4 โหมดคือ โหมด0 : ไทเมอร์/เคาเตอร์ 13 บิต (13 bit timer/counter) ,โหมด 1: ไทเมอร์/เคาเตอร์ 16 บิต (16 bit timer/counter) , โหมด 2: ตั้งค่าอัตโนมัติ ขนาด 8 บิต (8 bit auto-reload timer/counter) และโหมด 3: ไทเมอร์/เคาเตอร์แยกส่วน (split timer/counter) หรืออาจเรียกว่า ไทเมอร์/เคาเตอร์ 8 บิต ก็ได้ในขณะที่ไทเมอร์ 2 มีโหมดการทำงาน 3 โหมดคือ โหมดแคปเจอร์หรือตรวจจับสัญญาณ (capture) , โหมดตั้งค่าอัตโนมัติ (auto-reload) และโหมดกำเนิดอัตราเร็วในการสื่อสารข้อมูลอนุกรมหรืออัตราบอด (baud rate generator)

การเลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1 สามารถทำได้ที่รีจีสเตอร์ TCON และ TMOD ร่วมกันโดย TCON ใช้ในการเอ็นเอเบิลไทเมอร์/เคาเตอร์ ส่วน TMOD ใช้ในการเลือกโหมดและลักษณะการทำงาน ในขณะที่การทำงานของไทเมอร์ 2 จะอธิบายแยกต่างหาก

การทำงานในโหมด 0 : ไทเมอร์/เคาเตอร์ 13 บิต

มีไดอะแกรมการทำงานแสดงในรูปที่ 6-1 ในที่นี้จะใช้ไทเมอร์/เคาเตอร์ 1 ในการอธิบายโหมดนี้ขะเป็นการกำหนดให้ใช้งานรีจีสเตอร์ TL1 เพียง 5 บิต และ TH1 ครบ 8 บิต โดย TL0 จะทำหน้าที่คล้ายกับเปรสเกลเลอร์หาร 32 สัญญาณอินพุตสำหรับการนับจะเลือกจากสัญญาณนาฬิกาภายในหรือภายนอกผ่านทางขา T1 ขึ้นอยู่กับการควบคุมของบิต C/T และ GATE ในรีจีสเตอร์ TMOD, บิต TR1 ในรีจีสเตอร์ TCON และสถานะของลจิกที่ขาอินพุต INT1 เมื่อ TL1 นับครบ32คือจาก 0-31 ก็จะส่งสัญญาณไปยัง TH1 เพื่อทำการเพิ่มค่า ดังนั้นในโหมดนี้ค่าของการนับจะมีขนาด 13 บิต เมื่อทำการนับครบรอบ ก็จะทำการเซตบิต TF1 ในรีจีสเตอร์ TCON

ส่วนการทำงานในโหมดนี้ของไทเมอร์/เคาเตอร์ 0 มีลักษณะเหมือนกันทุกประการเพียงแต่เปลี่ยนรีจีสเตอร์และขาสัญญาณที่เกี่ยวข้องให้เป็นของไทเมอร์/เคาเตอร์ 0

การทำงานในโหมด 1 : ไทเมอร์/เคาเตอร์ 16 บิต

มีไดอะแกรมการทำงานแสดงในรูปที่ 6-2 ในที่นี้จะใช้ไทเมอร์ 1 ในการอธิบายการทำงานในโหมดนี้จะคล้ายกับโหมด 0 แต่จะใช้งานรีจีสเตอร์ TL1 และ TH1 ครบ 8 บิต ดังนั้นในโหมดนี้ค่าของการนับจะมีขนาด 16 บิต คือ 0000H-FFFFH เมื่อทำการนับครบรอบ ค่าของการนับเปลี่ยน

จาก FFFFH เป็น 0000H ก็จะเซตบิต TF1 ในรีจีสเตอร์ TCON ส่วนการทำงานในโหมดนี้ของไทเมอร์ 0 มีลักษณะเหมือนกันทุกประการ เพียงแต่เปลี่ยนรีจีสเตอร์และขาสัญญาณที่เกี่ยวข้องให้เป็นของไทเมอร์ 0

การทำงานในโหมด 2 : ไทเมอร์/เคาเตอร์ 8 บิต แบบตั้งค่าอัตโนมัติ

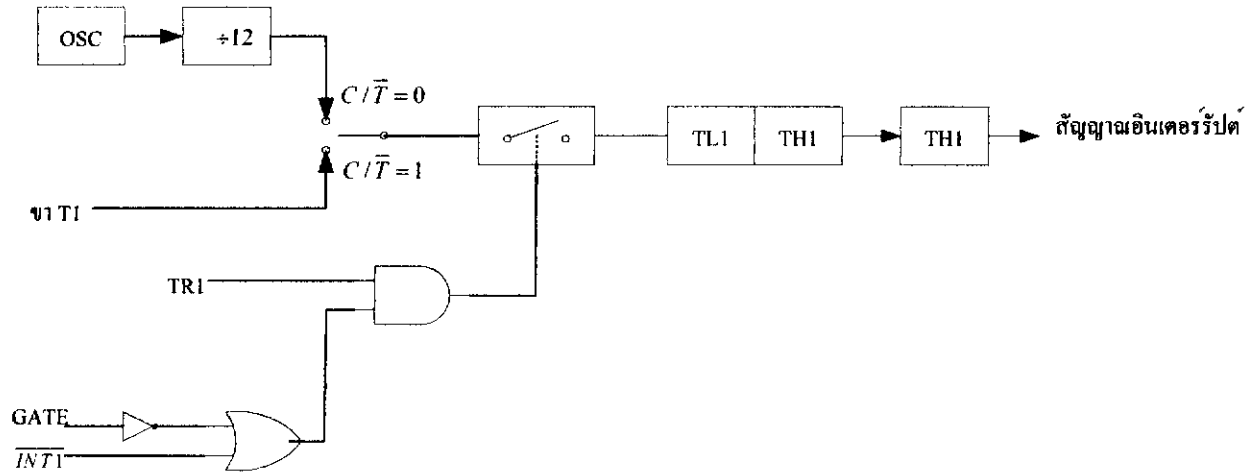
มีไดอะแกรมการทำงานแสดงในรูปที่ 6-3 ในที่นี้จะใช้ไทเมอร์/เคาเตอร์ 1 ในการอธิบายการทำงานในโหมดนี้จะแยกรีจีสเตอร์ไทเมอร์ออกเป็น 2 ตัว ตัวละ 8 บิต โดยรีจีสเตอร์ TL1 ทำหน้าที่เป็นตัวนับค่า ส่วน TH1 ใช้ในการเก็บค่าเริ่มต้นของการนับเมื่อเริ่มต้นการทำงาน ค่าของรีจีสเตอร์ TH1 จะถูกส่งไปยังรีจีสเตอร์ TL1 ทำให้เมื่อเริ่มต้นการทำงานค่าของรีจีสเตอร์ TL1 และ TH1 จะเหมือนกัน เมื่อ TL1 นับถึง FFH และจะเริ่มต้นการนับรอบใหม่ จะทำการเซตบิตTF1 พร้อมๆ กับทำการรับค่าการนับเริ่มต้นจาก TH1 ใหม่โดยอัตโนมัติ หรือเรียกกระบวนการนี้ว่า รีโหลด (reload) แม้ว่าจะมีการส่งค่าเริ่มต้นไปยัง TL1 แล้วก็ตาม ค่าของข้อมูลในรีจีสเตอร์ TH1 ก็ยังคงเป็นค่าเดิม ไม่มีการเปลี่ยนแปลง จนกว่าจะมีการกำหนดค่าใหม่ด้วยกระบวนการทางซอฟต์แวร์

ส่วนการทำงานในโหมดนี้ของไทเมอร์/เคาเตอร์ 0 มีลักษณะเหมือนกันทุกประการ เพียงแต่เปลี่ยนรีจีสเตอร์และขาสัญญาณที่เกี่ยวข้องให้เป็นของไทเมอร์/เคาเตอร์

การทำงานในโหมด 3 : ไมเมอร์/เคาเตอร์แยกส่วนหรือไทเมอร์/เคาเตอร์ 8 บิต

ในโหมดนี้เป็นโหมดเดียวที่การทำงานของไทเมอร์ 0 และ ไทเมอร์ 1 ไม่เหมือนกัน ขออธิบายในส่วนของไทเมอร์ 1 ก่อน เมื่อเข้าสู่โหมดนี้ จะเป็นการสั่งให้ไทเมอร์/เคาเตอร์หยุดนับ ค่าของการนับก่อนหน้านี้จะถูกเก็บไว้ในรีจีสเตอร์ไทเมอร์1 มีลักษณะการทำงานเหมือนกับการดิสเอเบิลไทเมอร์/เคาเตอร์ 1ด้วยการเคลียร์บิต TR1 ในรีจีสเตอร์ TCON

ส่วนการทำงานของไทเมอร์ 0 ในโหมดนี้มีไดอะแกรมการทำงานแสดงในรูปที่ 6-4 การทำงานในโหมดนี้จะแยกรีจีสเตอร์ไทเมอร์ 0 ออกเป็น 2 ตัว ตัวละ 8 บิต คือรีจีสเตอร์ TL0 และ TH1 โดยแยกการทำงานออกจากกันรีจีสเตอร์ TL0 สามารถเลือกการทำงานได้เหมือนกับไทเมอร์/เคาเตอร์ตามปกติ ส่วนรีจีสเตอร์ TH0 สามารถทำงานในโหมดไทเมอร์ได้เพียงอย่างเดียว กล่าวคือสามารถรับสัญญาณอินพุตจากสัญญาณนาฬิกาหภายในเพียงทางเดียวเท่านั้น แต่การแจ้งการนับเกินยังคงเหมือนเดิม หากแต่ TL0 แจ้งผ่านบิต TF0 ในขณะที่ TH0 จะแจ้งผ่านทางบิต TF1



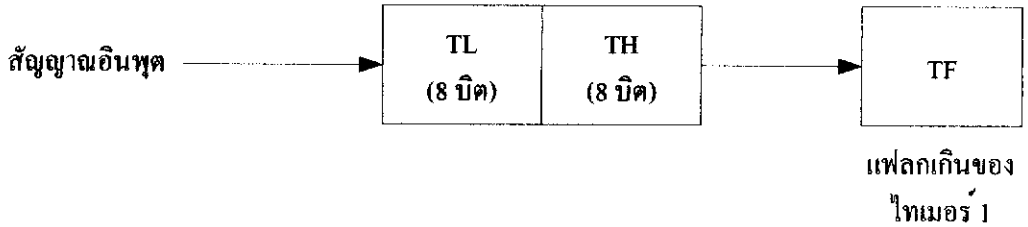
รูปที่ 2.11 ไดอะแกรมการทำงานในโหมด 0 ของไทเมอร์/เคาเตอร์ 1

2.8.7 ข้อมูลที่ใช้ในการเลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1

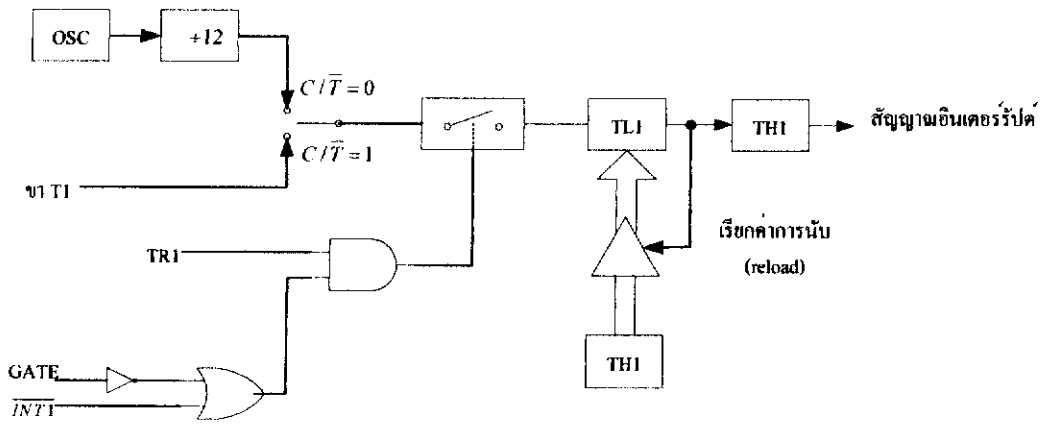
เนื่องจากมีตัวแปรอยู่หลายตัวที่ใช้ในการควบคุมและเลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1 เพื่อให้เกิดความสะดวกในการใช้งานจึงได้ทำการสรุปข้อมูลที่ใช้ในการกำหนดรูปแบบการทำงานของไทเมอร์/เคาเตอร์ 0 และ 1 ไว้ในตารางที่ 6- 1ถึง 6-4
อย่างไรก็ตาม ข้อมูลที่นำมายกเป็นตัวอย่างขั้นต้นนี้ สามารถเปลี่ยนแปลงได้ตามความต้องการของผู้ใช้งานไม่จำเป็นต้องยึดค่าเหล่านี้ไว้เสมอไป แต่สำหรับผู้เริ่มต้นใช้งานควรใช้ค่าตัวอย่างที่ให้ไว้ในการเขียนโปรแกรมควบคุมก่อน จนกว่ามีความชำนาญจึงค่อยปรับเปลี่ยนต่อไป

2.9 ไทเมอร์/เคาเตอร์ 2 ในไมโครคอนโทรลเลอร์ MCS-51

ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ AT89C52 และในอนุกรม AT89Sxx ยังมีไทเมอร์/เคาเตอร์ขนาด 16 บิต ให้ใช้งานเพิ่มเติมอีกหนึ่งตัวคือ ไทเมอร์เคาเตอร์ 2 (timer/counter2) รีจีสเตอร์ฟังก์ชั่นพิเศษที่เกี่ยวข้องกับไทเมอร์/เคาเตอร์ 2 มีเพิ่มเติมอีก 5 ตัวคือ รีจีสเตอร์ไทเมอร์/เคาเตอร์ 2 ซึ่งประกอบด้วย TL2 และ TH2 , รีจีสเตอร์ T2CON , รีจีสเตอร์ T2MOD มีแอดเดรสอยู่ที่ C9H สุดท้ายคือรีจีสเตอร์แคปเจอร์ ซึ่งประกอบด้วย RCAP2L และ RCAP2H มีแอดเดรสอยู่ที่ CAH และ CBH



รูปที่ 2.12 ไดอะแกรมการทำงานในโหมด 1 ของไทเมอร์/เคาเตอร์ 1



รูปที่ 2.13 ไดอะแกรมการทำงานในโหมด 2 ของไทเมอร์/เคาเตอร์ 1

ไทเมอร์/เคาเตอร์ 2 สามารถทำงานเป็นไทเมอร์หรือตัวตั้งเวลาและเคาเตอร์หรือตัวนับได้ เช่นเดียวกับไทเมอร์/เคาเตอร์ 0 และ 1 โดยการกำหนดที่บิต C/T ในรีจีสเตอร์ T2CON เมื่อเทียบกับไทเมอร์/เคาเตอร์ 0 และ1 ก็คือรีจีสเตอร์ TCON สำหรับโหมดการทำงานของไทเมอร์/เคาเตอร์ 2 มีด้วยกัน 3 โหมด คือ โหมดแคปเจอร์ หรือตรวจจับสัญญาณ (capture) , โหมดตั้งค่าอัตโนมัติ (auto-reload) และโหมดกำหนดอัตราเร็วในการสื่อสารข้อมูลอนุกรมหรืออัตราบอด(baud rate generator)

2.9.1 รีจีสเตอร์ที่เกี่ยวข้องกับการทำงานของไทเมอร์/เคาเตอร์ 2

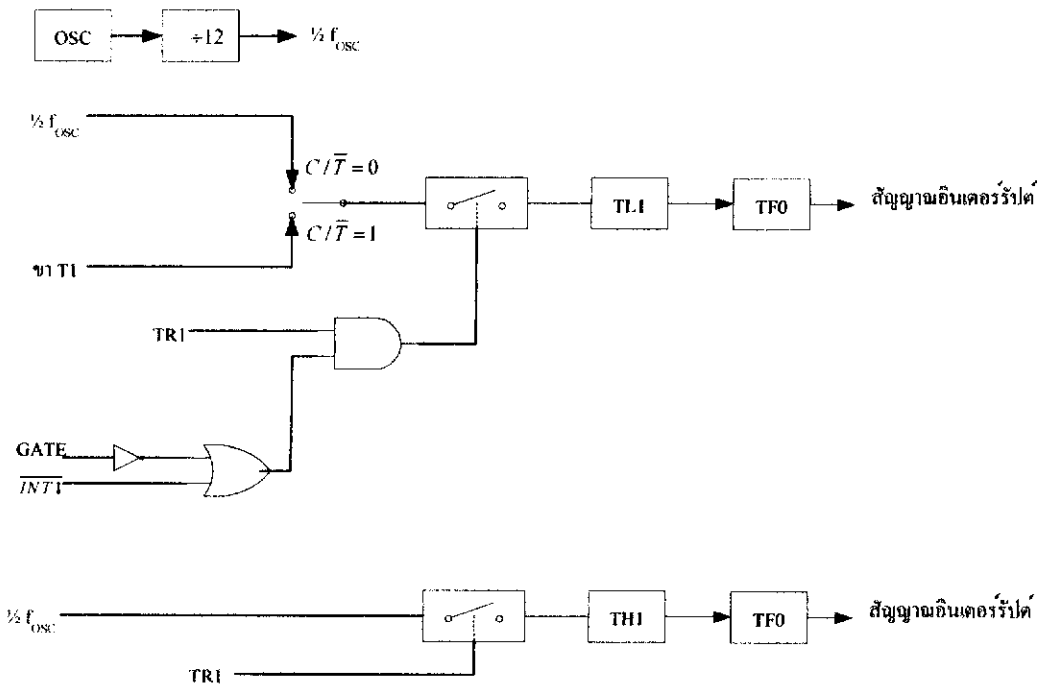
ในการทำงานของไทเมอร์/เคาเตอร์ 2 ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช มีรีจีสเตอร์ที่ต้องเกี่ยวข้องเป็นพื้นฐานอยู่ 5 ตัวดังรายละเอียดต่อไปนี้

รีจีสเตอร์ไทเมอร์

มีด้วยกัน 2 ตัวคือ TL2 มีแอดเดรสอยู่ที่ CCH และ TH2 มีแอดเดรสอยู่ที่ CDH รีจีสเตอร์ทั้ง 2 ตัวจะอยู่ในพื้นที่รีจีสเตอร์ของฟังก์ชั่นพิเศษหรือ SFR รีจีสเตอร์แต่ละตัวมีขนาด 8 บิต แต่ในการใช้งานทั่วไปมักใช้รวมกันเป็นรีจีสเตอร์ไทเมอร์ 2 ขนาด 16 บิต โดยใน TL2 จะเก็บข้อมูล 8 บิตล่าง ส่วน TH2 เก็บข้อมูล 8 บิตบน รีจีสเตอร์ทั้งคู่เมื่อนำมาใช้งานร่วมกันจะสามารถเก็บค่าของการนับได้สูงสุด 65,536 ค่าหรือ FFFFH เมื่อนับถึงค่านี้แล้วก็จะวนไปเริ่มนับ 0000H ใหม่ และเมื่อเกิดการนับรอบใหม่ จะมีการเซตบิต TF2 ในรีจีสเตอร์ T2CON ที่ใช้ควบคุมการทำงานของไทเมอร์ เพื่อแจ้งให้ทราบว่าเกิดการนับเกินค่าสูงสุด

2.9.2 รีจีสเตอร์ควบคุมการทำงานของไทเมอร์/เคาเตอร์ 2 หรือ T2CON (Timer/counter 2 Control Register)

เป็นรีจีสเตอร์ขนาด 8 บิตมีแอดเดรสอยู่ที่ C8H ในพื้นที่ของรีจีสเตอร์ SFR สามารถเข้าถึงได้ในระดับบิต มีรายละเอียดการทำงานดังนี้



รูปที่ 2.14 ไดอะแกรมการทำงานในโหมด 3 ของไทเมอร์/เคาเตอร์

บิต 7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

TF2 (Timer 2 overflow flag): จะเซตเมื่อค่าของรีจีสเตอร์ Timer 2 เกิดการนับเกินหรือเกิดโอเวอร์โฟลว การเคลียร์บิตนี้ทำได้ด้วยกระบวนการทางซอฟต์แวร์ และจะไม่เกิดการเซตถ้าบิต RCLK หรือบิต TCLK เป็น "1"

EXF2 (Timer 2 external flag) : จะเกิดการเซตเมื่อมีการแคปเจอร์หรือรีโหลดขึ้น ซึ่งเกิดจากการเปลี่ยนแปลงระดับลอจิกจาก "1" เป็น "0" ที่ขาอินพุต T2EX (ขา P1.1) และบิต EXEN2 เป็น "1" ในกรณีที่อินเตอร์รัปต์ในไทเมอร์ 2 ได้รับการเอ็นเอเบิล และบิต EXF2 นี้เกิดเซตจะมีผลทำให้ซีพียูไปอ่านค่าเวกเตอร์ของการบริการอินเตอร์รัปต์ บิตนี้สามารถเคลียร์ด้วยกระบวนการทางซอฟต์แวร์เท่านั้น

RCLK (Receive clock flag) : ถ้าบิตนี้เกิดการเซต ทำให้วงจรพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ใช้พัลส์โอเวอร์โฟลวของไทเมอร์ 2 เป็นสัญญาณนาฬิกาในการรับข้อมูลในโหมด 1 และ 3 ถ้าหากบิตนี้เป็น "0" จะใช้พัลส์จากไทเมอร์ 1

TCLK (transmit clock flag): ถ้าบิตนี้เกิดการเซต ทำให้วงจรพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ใช้พัลส์โอเวอร์โฟลวของไทเมอร์ 2 เป็นสัญญาณนาฬิกาในการส่งข้อมูลในโหมด 1 และ 3 ถ้าหากบิตนี้เป็น "0" จะใช้พัลส์จากไทเมอร์ 1

EXEN2 (Timer 2 external enable flag): เมื่อบิตนี้เซตเป็น "1" จะเป็นการเอ็นเอเบิลการแคปเจอร์หรือรีโหลดที่ขา T2EX ให้เกิดขึ้นได้ ภายใต้เงื่อนไขว่า ไทเมอร์ 2 ต้องไม่ถูกนำไปใช้ในการสื่อสารพอร์ตอนุกรม หรือบิต RCLK หรือ TCLK เป็น "1" ถ้าบิตนี้เป็น "0" ไทเมอร์ 2 จะไม่สนใจเหตุการณ์ขา T2EX

โหมด	ฟังก์ชั่นของไทเมอร์ 0	TMOD	
		การควบคุมจากภายใน	การควบคุมจากภายนอก
0	ไทเมอร์/เคาเตอร์ 13 บิต	00H	08H
1	ไทเมอร์/เคาเตอร์ 16 บิต	01H	09H
2	8 บิตตั้งค่าอัตโนมัติ	02H	0AH
3	ไทเมอร์/เคาเตอร์แยกส่วน	03H	0BH

ตารางที่ 2.2 แสดงข้อมูลของรีจีสเตอร์ TMOD เพื่อกำหนดให้ไทเมอร์/เคาเตอร์ 0 ทำงานเป็นไทเมอร์

*หมายเหตุ

การควบคุมจากภายใน : ควบคุมให้ไทเมอร์เปิดปิดด้วยการเซตและเคลียร์บิต TR0 โดยกระบวนการทางซอฟต์แวร์

การควบคุมจากภายนอก : ควบคุมให้ไทเมอร์เปิดปิดด้วยการตรวจการเปลี่ยนแปลงจาก "1" เป็น "0" ที่ขา $\overline{INT0}$ (P3.2) เมื่อ TR0 = "1"

โหมด	ฟังก์ชั่นของไทเมอร์ 0	TMOD	
		การควบคุมจากภายใน	การควบคุมจากภายนอก
0	ไทเมอร์/เคาเตอร์ 13 บิต	04H	0CH
1	ไทเมอร์/เคาเตอร์ 16 บิต	05H	0DH
2	8 บิตตั้งค่าอัตโนมัติ	06H	0EH
3	ไทเมอร์/เคาเตอร์แยกส่วน	07H	0FH

ตารางที่ 2.3 แสดงข้อมูลของรีจีสเตอร์ TMOD เพื่อกำหนดให้ไทเมอร์/เคาเตอร์ 0 ทำงานเป็นเคาเตอร์

*หมายเหตุ

การควบคุมจากภายใน : ควบคุมให้ไทเมอร์เปิดปิดด้วยการเซตและเคลียร์บิต TR0 โดยกระบวนการทางซอฟต์แวร์

การควบคุมจากภายนอก : ควบคุมให้ไทเมอร์เปิดปิดด้วยการตรวจการเปลี่ยนแปลงจาก "1" เป็น "0" ที่ขา $\overline{INTO}$ (P3.2) เมื่อ TR0 = "1"

โหมด	ฟังก์ชั่นของไทเมอร์ 0	TMOD	
		การควบคุมจากภายใน	การควบคุมจากภายนอก
0	ไทเมอร์/เคาเตอร์ 13 บิต	00H	80H
1	ไทเมอร์/เคาเตอร์ 16 บิต	10H	90H
2	8 บิตตั้งค่าอัตโนมัติ	20H	A0H
3	หยุดการทำงาน	30H	B0H

ตารางที่ 2.4 แสดงข้อมูลของรีจีสเตอร์ TMOD เพื่อกำหนดให้ไทเมอร์/เคาเตอร์ 1 ทำงานเป็นไทเมอร์

*หมายเหตุ

การควบคุมจากภายใน : ควบคุมให้ไทเมอร์เปิดปิดด้วยการเซตและเคลียร์บิต TR1 โดยกระบวนการทางซอฟต์แวร์

การควบคุมจากภายนอก : ควบคุมให้ไทเมอร์เปิดปิดด้วยการตรวจการเปลี่ยนแปลงจาก "1" เป็น "0" ที่ขา $\overline{INT1}$ (P3.3) เมื่อ TR1 = "1"

โหมด	ฟังก์ชั่นของไทเมอร์ 0	TMOD	
		การควบคุมจากภายใน	การควบคุมจากภายนอก
0	ไทเมอร์/เคาเตอร์ 13 บิต	40H	C0H
1	ไทเมอร์/เคาเตอร์ 16 บิต	50H	D0H
2	8 บิตตั้งค่าอัตโนมัติ	60H	E0H
3	-	-	-

ตารางที่ 2.5 แสดงข้อมูลของรีจีสเตอร์ TMOD เพื่อกำหนดให้ไทเมอร์/เคาเตอร์ 1 ทำงานเป็นไทเมอร์

*หมายเหตุ

การควบคุมจากภายใน : ควบคุมให้ไทเมอร์เปิดปิดด้วยการเซตและเคลียร์บิต TR1 โดยกระบวนการทางซอฟต์แวร์

การควบคุมจากภายนอก : ควบคุมให้ไทเมอร์เปิดปิดด้วยการตรวจการเปลี่ยนแปลงจาก "1" เป็น "0" ที่ขา $\overline{INT1}$ (P3.3) เมื่อ TR1 = "1"

TR2: บิตนี้ใช้ในการควบคุมการเริ่มต้นและหยุดการทำงานของไทเมอร์ 2 เป็นการควบคุมทางซอฟต์แวร์ "0" หยุดการทำงานของไทเมอร์ 2 (STOP)

"1" เริ่มต้นการทำงานของไทเมอร์ 2 (START)

C/T (Timer or Counter selector): ใช้เลือกลักษณะการทำงานของไทเมอร์/เคาเตอร์ 2

"0" เลือกให้ทำงานเป็นไทเมอร์ โดยใช้สัญญาณอินพุตจากสัญญาณนาฬิกาภายในไมโครคอนโทรลเลอร์

"1" เลือกให้ทำงานเป็นเคาเตอร์ โดนรับสัญญาณอินพุตทางขา T2 (P1.0)

CP/RL2 (Capture /Reload flag): เป็นบิตแสดงสถานะการทำงานของไทเมอร์ 2

"0" มีการรีโหลดค่าของการนับแบบอัตโนมัติเกิดขึ้น เมื่อไทเมอร์ 2 เกิดโอเวอร์โฟลวหรือเกิดการเปลี่ยนระดับลอจิกแบบลบ (เปลี่ยนจาก "1" เป็น "0") ที่ขา T2EX ในกรณีที่บิต EXEN2 เป็น "1" แต่ถ้าหากบิต RCLK หรือ TCLK เป็น "1" จะยอมให้การรีโหลดแบบอัตโนมัติเกิดขึ้นเมื่อไทเมอร์ 2 เกิดโอเวอร์โฟลวเท่านั้น

"1" แสดงว่ามีการแคปเจอร์เกิดขึ้น เมื่อเกิดการเปลี่ยนแปลงระดับลอจิกแบบลบ (เปลี่ยนจาก "1" เป็น "0") ที่ขา T2EX ในกรณีที่บิต EXEN2 เป็น "1"

2.9.3 รีจีสเตอร์เลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 2 หรือ T2MOD (Timer/Counter 2 Mode Control Register)

เป็นรีจีสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ C9H ในพื้นที่ของรีจีสเตอร์ SFR ไม่สามารถเข้าถึงได้ในระดับบิต มีบิตสำหรับกำหนดการทำงานเพียง 2 บิต คือ

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
-	-	-	-	-	-	T2OE	DCEN

T2OE (Timer 2 Output Enable bit): ใช้เอ็นเอเบิลเอาต์พุตของไทเมอร์ 2

"0" ดิสเอเบิล

"1" เอ็นเอเบิล

DCEN (Timer 2 Counter Enable bit): กำหนดให้ไทเมอร์ 2 ทำงานเป็นตัวนับหรือเคาเตอร์แบบขึ้นและลง

"0" ดิสเอเบิล

"1" เอ็นเอเบิลให้ทำงานเป็นตัวนับหรือเคาเตอร์แบบขึ้นและลง

2.9.4 โหมดการทำงานของไทเมอร์/เคาเตอร์ 2

ไทเมอร์/เคาเตอร์ 2 สามารถเลือกโหมดการทำงานได้ 3 โหมด โดยการกำหนดที่บิต RCLK+TCLK, CP/RL2 และ TR2 ในรีจีสเตอร์ T2CON ดังแสดงรายละเอียดในตารางที่ 6-5 โหมดการทำงานของไทเมอร์/เคาเตอร์ 2 ได้แก่ โหมดแคปเจอร์หรือตรวจจับสัญญาณ (capture) ,โหมดการตั้งค่าการนับอัตโนมัติ (auto-reload) และโหมดกำเนิดอัตราเร็วในการสื่อสารข้อมูลอนุกรมหรืออัตราบอด (baud rate generator) ต่อไปนี้จะเป็นการอธิบายการทำงานในแต่ละโหมดของไทเมอร์/เคาเตอร์ 2

RCLK+TCLK	CP/$\overline{RL2}$	TR2	โหมด
0	0	1	16บิตแบบตั้งค่าการนับอัตโนมัติหรือรีโหลด
0	1	1	แคปเจอร์หรือตรวจจับสัญญาณ
1	X	1	กำเนิดอัตราบอด
X	X	0	ดิสเอเบิล

ตารางที่ 2.6 การเลือกโหมดทำงานของไทเมอร์/เคาเตอร์ 2

2.9.5 การทำงานในโหมดแคปเจอร์หรือตรวจจับสัญญาณ (Capture)

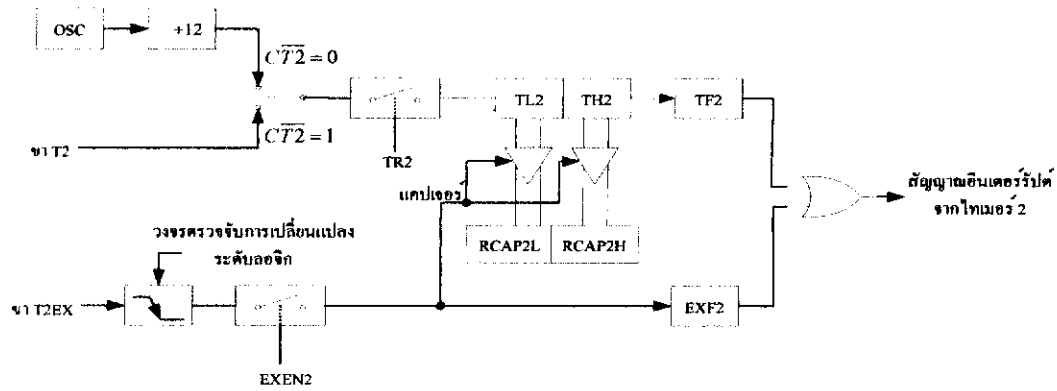
การทำงานในโหมดนี้จะเกิดขึ้นได้อย่างสมบูรณ์เมื่อกำหนดให้บิต EXEN2 เป็น "1" หลังจากที่เลือกให้ไทเมอร์ 2 ทำงานในโหมดแคปเจอร์แล้วเมื่อเลือกให้ทำงานในโหมดนี้ขาอินพุตที่ทำการตรวจจับคือขาอินพุต T2EX ซึ่งตรงกับขา P1.1 ที่ขา T2EX จะมีวงจรตรวจจับการเปลี่ยนแปลงของระดับลอจิกจาก "1" เป็น "0" รีจีสเตอร์TL2 และ TH2 จะทำการนับค่าเพิ่มขึ้นไปเรื่อยๆ จนกว่าที่ขา T2EX จะตรวจจับการเปลี่ยนแปลงได้ เมื่อตรวจจับได้ ค่าของรีจีสเตอร์ TL2 และ TH2 จะถูกส่งต่อไปยังรีจีสเตอร์ RCAP2L และ RCAP2H พร้อมกันนั้นยังทำการเซตบิต EXF2 ให้เป็น "1" เพื่อสร้างสัญญาณอินเตอร์รัปต์ให้เกิดขึ้น ในรูปที่ 6-5 แสดงการทำงานของไทเมอร์ 2 ในโหมดแคปเจอร์นี้

2.9.6 การทำงานในโหมดการตั้งค่าการนับอัตโนมัติ

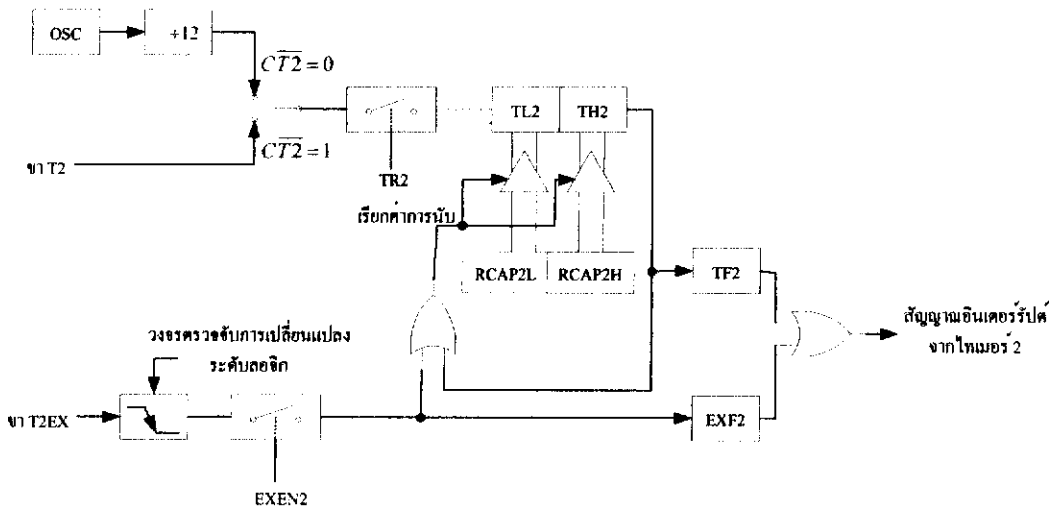
การทำงานในโหมดนี้ การกำหนดสถานะที่บิต EXEN2 จะทำให้เกิดการทำงานใน 2 ลักษณะคือ ถ้าบิต EXEN2 เป็น "0" เมื่อไทเมอร์ 2 ทำการนับเกินรอบแล้ว จะเกิดการเซตบิต TF2 และจะทำการเรียกค่าการนับใหม่จากรีจีสเตอร์ RCAP2L และ RCAP2H โดยอัตโนมัติ การกำหนดค่าของรีจีสเตอร์ RCAP2L และ RCAP2H สามารถกระทำได้ด้วยกระบวนการทางซอฟต์แวร์ ในกรณีที่บิตEXEN2 เป็น "1" หลังจากที่เลือกให้ไทเมอร์ 2 จะทำการนับเหมือนเดิม แต่เงื่อนไขของการตั้งค่าการนับใหม่โดยอัตโนมัติจากรีจีสเตอร์ RCAP2L และ RCAP2H จะใช้การเปลี่ยนแปลงระดับลอจิกจาก "1" เป็น "0" ที่ขา T2EX เป็นตัวกำหนด และทำการเซตบิต EXF2 เพื่อสร้างสัญญาณอินเตอร์รัปต์แทนบิตTF2 ในรูปที่ 6-6 แสดงไดอะแกรมการทำงานของไทเมอร์ 2 ในโหมดการตั้งค่าอัตโนมัติ

2.9.7 การทำงานในโหมดกำเนิดอัตราเร็วในการสื่อสารข้อมูลอนุกรมหรืออัตราบอด

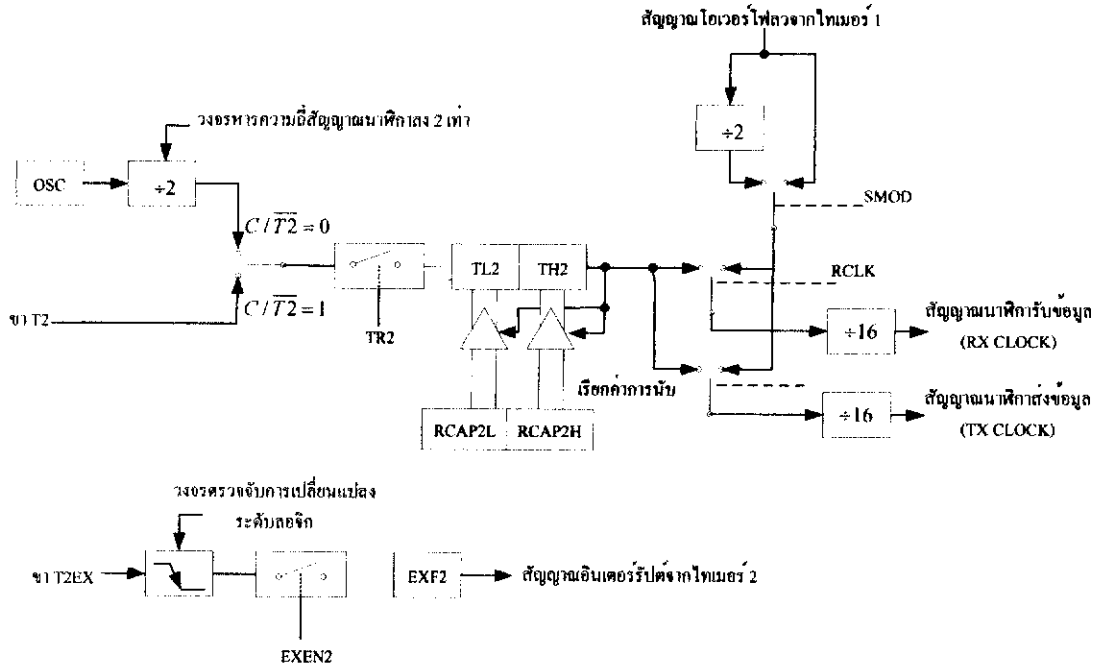
การทำงานในโหมดนี้จะเกิดขึ้นเมื่อทำการเซตบิต RCLK และ/หรือ TCLK ในรีจีสเตอร์ T2CON ให้เป็น "1" เมื่อเข้าสู่การทำงานในโหมดนี้ บิต TF2 จะไม่เกิดการเซต และไม่มีการสร้างสัญญาณอินเตอร์รัปต์ แต่การอินเตอร์รัปต์ในไทเมอร์ 2 ก็ไม่ได้ถูกดิสเอเบิล เมื่ออยู่ในโหมดนี้ บิต EXEN2 ถูกเซต การเปลี่ยนแปลงระดับลอจิกจาก "1" เป็น "0" ที่ขา T2EX จะส่งผลให้บิต EXF2 เกิดการเซต แต่จะไม่มีการเรียกค่าจากรีจีสเตอร์ RCAP2L และ RCAP2H เมื่อเป็นนี้ขา T2EX จึงสามารถใช้เป็นขาอินพุตสำหรับการอินเตอร์รัปต์ได้เป็นกรณีพิเศษ



รูปที่ 2.15 ไดอะแกรมการทำงานในโหมดแคปเจอร์ของไทเมอร์/เคาเตอร์ 2



รูปที่ 2.16ไดอะแกรมการทำงานในโหมดตั้งค่าการนับอัตโนมัติของไทเมอร์/เคาเตอร์ 2



รูปที่ 2.17 ไดอะแกรมการทำงานในโหมดกำเนิดอัตราบอดในการสื่อสารข้อมูลผ่านพอร์ตอนุกรมของไทเมอร์/เคาเตอร์ 2

โหมดการทำงานของไทเมอร์/เคาเตอร์ 2	T2CON	
	การควบคุมจากภายใน	การควบคุมจากภายนอก
16 บิตแบบตั้งค่าการนับอัตโนมัติหรือรีโหลด	00H	08H
แคปเจอร์หรือตรวจจับสัญญาณ	01H	09H
กำเนิดอัตราบอด โดยอัตรารับและส่งเท่ากัน	34H	36H
รับข้อมูลเท่านั้น	24H	26H
ส่งข้อมูลเท่านั้น	14H	16H

ตารางที่ 2.7 แสดงข้อมูลของรีจีสเตอร์ T2CON เพื่อกำหนดให้ไทเมอร์/เคาเตอร์ 2 ทำงานเป็นไทเมอร์

*หมายเหตุ

การควบคุมจากภายใน : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์

การควบคุมจากภายนอก : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์และการเปลี่ยนแปลงระดับลอจิก "1" เป็น "0" ที่ขา T2EX ยกเว้นกรณีที่กำหนดให้ทำงานในการกำเนิดอัตราบอดสำหรับพอร์ตอนุกรมในไมโครคอนโทรลเลอร์

โหมดการทำงานของไทเมอร์/เคาเตอร์ 2	T2CON	
	การควบคุมจากภายใน	การควบคุมจากภายนอก
16 บิตแบบตั้งค่าการนับอัตโนมัติหรือรีโหลด	02H	0AH
แคปเจอร์หรือตรวจจับสัญญาณแบบ 16บิต	03H	0BH

ตารางที่ 2.8 แสดงข้อมูลของรีจีสเตอร์ T2CON เพื่อกำหนดให้ไทเมอร์/เคาเตอร์ 2 ทำงานเป็นเคาเตอร์

*หมายเหตุ

การควบคุมจากภายใน : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์

การควบคุมจากภายนอก : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์และการเปลี่ยนแปลงระดับลอจิก "1" เป็น "0" ที่ขา T2EX ยกเว้นกรณีที่กำหนดให้ทำงานในการกำเนิดอัตราบอดสำหรับพอร์ตอนุกรมในไมโครคอนโทรลเลอร์

เมื่อไทเมอร์ 2ทำงาน (บิต TR2 เป็น "1") จะถูกกำหนดให้ทำงานเป็นไทเมอร์จึงไม่สามารถอ่านหรือเขียนค่ากับรีจีสเตอร์ TL2 และ TH2 ได้ ค่าของรีจีสเตอร์ไทเมอร์จะเพิ่มขึ้นทุกๆ สเตต (1 สเตตเท่ากับ 2คาบเวลาของสัญญาณนาฬิกา) ค่าการนับของรีจีสเตอร์นำมาใช้สร้างสัญญาณนาฬิกาของการสื่อสารข้อมูลผ่านพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ ในรูปที่ 6-7 แสดงการทำงานของไทเมอร์ 2 ในการกำเนิดอัตราบอด

2.9.8 ข้อมูลที่ใช้ในการเลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 2

เนื่องจากมีตัวแปรอยู่หลายตัวที่ใช้ในการควบคุมและเลือกโหมดการทำงานของไทเมอร์/เคาเตอร์ 2 เช่นเดียวกับไทเมอร์/เคาเตอร์ 0 และ 1 โดยเฉพาะอย่างยิ่งโหมดของการทำงานที่แตกต่างกันมากและมีความซับซ้อนเพิ่มมากขึ้น เพื่อให้เกิดความสะดวกในการใช้งานจึงได้ทำการสรุปข้อมูลที่ใช้ ในการกำหนดรูปแบบการทำงานของไทเมอร์/เคาเตอร์ 2 ไว้ในตารางที่ 2.7 ถึง 2.8

2.10 วอตช์ด็อกไทเมอร์ (Watchdog Timer: WDT)

นอกเหนือไปจากไทเมอร์/เคาเตอร์ทั้งสามแล้ว สำหรับในอนุกรม AT89Sxx ไม่ว่าจะเป็นเบอร์AT89S8252, AT89LS8252 และเบอร์ AT89S53 ยังมีวงจรไทเมอร์อีกแบบหนึ่งเพิ่มเติมเพื่อให้สามารถนำไปใช้สร้างสัญญาณรีเซตในกรณีที่ซีพียูอยู่ภายในไมโครคอนโทรลเลอร์เกิดภาวะหยุดทำงานกระทันหันหรือแฮงก์ (hang) วอตช์ด็อกไทเมอร์ (Watchdog Timer: WDT) โดยวอตช์

ด็อกไทเมอร์ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชอนุกรม AT89Sxx จะเป็นวอตช์ด็อกไทเมอร์แบบโปรแกรมได้ โดยการกำหนดค่าของบิต PS0-PS2 รีจีสเตอร์ WMCON สำหรับเบอร์ AT89S8252 และรีจีสเตอร์ WCON สำหรับเบอร์ AT89S53

วอตช์ด็อกไทเมอร์จะได้รับการดิสเอเบิลโดยอัตโนมัติเมื่อเริ่มต้นจ่ายไฟหรือเมื่อเกิดพาวเวอร์ออนรีเซต และเมื่อไมโครคอนโทรลเลอร์เข้าสู่การทำงานในโหมดประหยัดพลังงานแบบลดพลังงานหรือพาวเวอร์ดาวน์ สำหรับการเอ็นเอเบิลสามารถกระทำได้โดยกระบวนการทางซอฟต์แวร์ โดยการเซตบิต WDTEN ในรีจีสเตอร์ WMCON สำหรับเบอร์ AT89S8252 และรีจีสเตอร์ WCON สำหรับเบอร์ AT89S53

การทำงานของวอตช์ด็อกไทเมอร์จะเริ่มต้นขึ้นเมื่อมีการกำหนดคาบเวลาในการทำงานของวอตช์ด็อกไทเมอร์และมีการเอ็นเอเบิลให้วอตช์ด็อกไทเมอร์เริ่มทำงาน วงจรนับภายในวอตช์ด็อกไทเมอร์ จะนับค่าเวลาไปตามปกติ หากซีพียูทำงานอยู่เป็นปกติจะมีการส่งสัญญาณมารีเซตหรือเคลียร์ค่าในวอตช์ด็อกไทเมอร์ อยู่เป็นระยะเพื่อไม่ให้วอตช์ด็อกไทเมอร์ทำงานจนถึงค่าเวลาที่กำหนดหรือเกิดการไทม์เอาต์ (time out) แต่ถ้าซีพียูเกิดภาวะหยุดทำงานขึ้นกระทันหัน ไม่ว่าจะด้วยสาเหตุใด ยกเว้นการรีเซตและการเข้าสู่โหมดประหยัดพลังงานแบบลดพลังงานหรือพาวเวอร์ดาวน์ ก็จะไม่มีสัญญาณมารีเซตหรือเคลียร์ค่าในวอตช์ด็อกไทเมอร์ทำให้ในที่สุดวอตช์ด็อกไทเมอร์ก็จะทำงานถึงค่าเวลาที่กำหนดเกิดการไทม์เอาต์ วอตช์ด็อกไทเมอร์จะทำการสร้างสัญญาณรีเซตส่งไปยังซีพียู เพื่อทำการรีเซตซีพียู ส่งให้ซีพียูสามารถกลับมาเริ่มต้นทำงานใหม่ได้อีกครั้ง โดยที่ไม่จำเป็นต้องทำการรีเซตซีพียูจากภายนอก หรือจ่ายไฟเลี้ยงใหม่ให้แก่ไมโครคอนโทรลเลอร์

ในไมโครคอนโทรลเลอร์ MCS-51 เบอร์พื้นฐาน เช่น 8031,8032,8051 หรือแบบแฟลชเบอร์ AT89C51 , AT89C52 , AT89C55 จะไม่มีการวอตช์ด็อกไทเมอร์อยู่ภายใน เมื่อมีความจำเป็นต้องการตรวจสอบการทำงานของซีพียู จึงต้องใช้เทคนิคในการเขียนโปรแกรมเข้าช่วย โดยการใช้ไทเมอร์ที่มีอยู่ในไมโครคอนโทรลเลอร์ เบอร์พื้นฐานเหล่านั้น หรือใช้ไปซีที่ทำหน้าที่เป็นวงจรวอตช์ด็อกไทเมอร์โดยเฉพาะต่อเข้ากับขา RST ของไมโครคอนโทรลเลอร์ เพื่อสร้างสัญญาณรีเซตในกรณีที่ไมโครคอนโทรลเลอร์เกิดภาวะหยุดทำงานกระทันหันโดยไม่ทราบสาเหตุ

2.10.1 รีจีสเตอร์ควบคุมการทำงานของ วอตช์ด็อกไทเมอร์ (Watchdog & Memory Control register: WMCON และ Watchdog control register: WCON)

ในการทำงานของวอตช์ด็อกไทเมอร์ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชอนุกรม AT89Sxx มีรีจีสเตอร์ที่ใช้ในการควบคุมการทำงาน 1 ตัว คือ WMCON สำหรับเบอร์ AT89S8252 และรีจีสเตอร์ WCON สำหรับเบอร์ AT89S53 โดยมีหน้าที่ในการทำงานหลักเหมือนกันทุกประการ ส่วนแตกต่างมีอยู่จุดเดียวคือ ในเบอร์ AT89S8252 มีหน่วยความจำข้อมูลแบบอีอีพรอมอยู่ จึงต้องมีการควบคุมการทำงานเพิ่มเติมเข้ามา จึงใช้บิต 3 และ 4 ในรีจีสเตอร์

WMCON มาจัดการหน่วยความจำข้อมูลส่วนนี้ สำหรับบิตอื่นๆ จะเหมือนกันทุกประการดังนั้นในการอธิบายรายละเอียดของรีจีสเตอร์ควบคุมการทำงานของวอตช์ด็อกไทเมอร์จะอธิบายรวมไปในคราวเดียวกัน โดยจะเพิ่มเติมรายละเอียดของบิตที่ใช้ควบคุมหน่วยความจำข้อมูลแบบอีอีพรอมเข้าไปด้วย

รีจีสเตอร์ควบคุมการทำงานของวอตช์ด็อกไทเมอร์ทั้งWMCON และ WCON เป็นรีจีสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 96H ในพื้นที่ของรีจีสเตอร์ SFR สามารถเข้าถึงได้ในระดับบิตมีรายละเอียดดังนี้

AT89S8252

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
PS2	**PS1**	**PS0**	**EEMWE**	**EEMEN**	**DPS**	**WDTRST**	**WDTEN**

AT89S53

PS2	**PS1**	**PS0**			**DPS**	**WDTRST**	**WDTEN**

PS2-PS0 (Prescaler bits): ใช้ในการเลือกค่าเวลาให้แก่วอตช์ด็อกไทเมอร์ มีรายละเอียดดังนี้

บิตปรีสเกลเลอร์			ค่าเวลาของวอตช์ด็อกไทเมอร์
PS2	PS1	PS0	
0	0	0	16 ms
0	0	1	32 ms
0	1	0	64 ms
0	1	1	128 ms
1	0	0	256 ms
1	0	1	512 ms
1	1	0	1024 ms
1	1	1	2048 ms

ตารางที่ 2.9 ใช้ในการเลือกค่าเวลาให้แก่วอตช์ด็อกไทเมอร์

EEMWE (EEROM Data Memory Write Enable bit): บิตนี้เป็นบิต 4 ของรีจีสเตอร์ WMCON ใช้ในการเอ็นเอเบิลการเขียนข้อมูลลงในหน่วยความจำข้อมูลแบบอีอีพรอมภายในไมโครคอนโทรลเลอร์ AT89S8252 เมื่อต้องการเขียนข้อมูลต้องทำการเซตบิตนี้ก่อน หลังจากการเขียนข้อมูลเสร็จสิ้นลง ต้องการทำการเคลียร์บิตนี้เสมอสำหรับในรีจีสเตอร์ WCON ในไมโครคอนโทรลเลอร์ AT89S53 บิตนี้จะไม่มีการใช้งานต้องกำหนดให้เป็น "0"

EEMEN (Internal EEPROM Access Enable bit) : บิตนี้เป็นบิต 3 ของรีจีสเตอร์ WMCON ใช้ในการเอ็นเอเบิลการเข้าถึงหน่วยความจำข้อมูลแบบอีอีพรอมภายใน AT89S8252 แทนที่จะใช้หน่วยความจำข้อมูลภายนอก เมื่อต้องการเข้าถึงข้อมูลในหน่วยความจำส่วนนี้ โดยใช้คำสั่ง MOVX และรีจีสเตอร์ DPTR ต้องทำการเซตบิตนี้ก่อน ถ้าหากบิตนี้เป็น "0" จะเป็นการกำหนดให้ไมโครคอนโทรลเลอร์ใช้คำสั่ง MOVXและรีจีสเตอร์ DPTRติดต่อกับหน่วยความจำข้อมูลภายนอกไมโครคอนโทรลเลอร์ สำหรับในรีจีสเตอร์ WCON ของ AT89S53 บิตนี้ไม่ใช้งาน กำหนดให้เป็น "0"

DPS (Data Pointer Register Select): เนื่องจากในไมโครคอนโทรลเลอร์อนุกรม AT89Sxx จะมีรีจีสเตอร์สำหรับชี้ข้อมูลหรือ DPTR 2 ตัว คือ DPTR0, DPTR1 บิตนี้จึงมีหน้าที่ใช้ในการเลือกรีจีสเตอร์ DPTR ถ้าเป็น "0" จะเป็นการเลือกรีจีสเตอร์ DPTR0 (ซึ่งก็คือ DP0L, DP0H) ในกรณีที่บิตนี้เป็น "1" จะเป็นการเลือกรีจีสเตอร์ DPTR1 (ซึ่งก็คือ DP1L, DP1H)

WDTRST/RDY/BSY (Watchdog Timer Reset and EEPROM Ready/Busy flag):

บิตนี้มีความเกี่ยวข้องกับการทำงาน2 ส่วนคือ การรีเซตของวอตช์ด็อกไทเมอร์และใช้เป็นบิตแสดงสถานะของหน่วยความจำข้อมูลอีอีพรอมภายในไมโครคอนโทรลเลอร์ AT89S8252 ในกรณีของวอตช์ด็อกไทเมอร์ บิตนี้จะเซตโดยผู้งานด้วยกระบนการทางซอฟต์แวร์ เป็นการสั่งให้วอตช์ด็อกไทเมอร์สร้างสัญญาณรีเซตส่งออกไป ในกรณีที่วอตช์ด็อกไทเมอร์เกิดไทม์เอาต์ หลังจากที่ส่งสัญญาณรีเซตออกไปแล้วบิตนี้จะเคลียร์ตัวเองเป็น "0" โดยอัตโนมัติ ในไซเกิลของการทำงานถัดไป ดังนั้นบิตนี้จึงเขียนได้เพียงอย่างเดียว

ในกรณีของหน่วยความจำข้อมูลอีอีพรอม จะใช้บิตนี้ในการป้องกันการเขียนข้อมูลลงในหน่วยความจำถ้าบิตนี้เป็น "1" หน่วยความจำข้อมูลอีอีพรอมจะสามารถอ่านได้เพียงอย่างเดียว เมื่อมีความต้องการเขียนข้อมูลหลังจากเอ็นเอเบิลการเขียนที่บิต EEMWE แล้ว ให้ทำการเคลียร์บิตนี้เป็น "0" หลังจากที่การเขียนข้อมูลเสร็จสิ้นลง บิตนี้จะเซตเป็น "1" โดยอัตโนมัติ เป็นการแจ้งให้ทราบว่า การเขียนข้อมูลเสร็จสิ้นลงแล้ว

สำหรับ AT89S53 ใช้งานบิตนี้ในการสร้างสัญญาณรีเซตของวอตช์ด็อกไทเมอร์เท่านั้น

WDTEN (Watchdog Timer Enable bit): ใช้ในเอ็นเอเบิลการทำงานของวอตช์ด็อกไทเมอร์

"0" ดิสเอเบิลวอตช์ด็อกไทเมอร์

"1" เอ็นเอเบิลวอตช์ด็อกไทเมอร์

จากรายละเอียดของไทเมอร์/เคาเตอร์ ทั้งหมด จะเห็นได้ว่า ในการสร้างระบบควบคุมที่มี ไทเมอร์/เคาเตอร์เป็นสิ่งที่สำคัญและจำเป็นต้องมี ไม่ว่าจะเป็นการสร้างอัตราบอดสำหรับการสื่อสารข้อมูลผ่านพอร์ตอนุกรมเพื่อเชื่อมต่อกับคอมพิวเตอร์ เพื่อสร้างฐานเวลา และยังสามารถใช้ในการวัดคาบเวลาและความถี่ของสัญญาณได้อีกด้วย นอกจากนั้นไมโครคอนโทรลเลอร์อนุกรม 89Sxx ที่มีวอตช์ด็อกไทเมอร์ในตัวนั้นยังช่วยให้ระบบควบคุมมีเสถียรภาพมากขึ้น การมีวอตช์ด็อกไทเมอร์ในตัวจัดได้ว่าเป็นคุณสมบัติขั้นพื้นฐานของไมโครคอนโทรลเลอร์สมัยใหม่ไปแล้วจึงส่งผลให้ไมโครคอนโทรลเลอร์ MCS-51 มีขีดความสามารถที่ทัดเทียมกับไมโครคอนโทรลเลอร์ยุคใหม่ และยังเป็นที่นิยมใช้งานต่อไป

2.11 พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51

พอร์ตสื่อสารอนุกรมของไมโครคอนโทรลเลอร์ มีโครงสร้างเป็นแบบฟูลดูเพล็กซ์ ซึ่งรับและส่งข้อมูลในเวลาเดียวกันได้ มีรีจีสเตอร์ SBUF (Serial Data Buffer) เป็นบัฟเฟอร์สำหรับการรับส่งข้อมูลอนุกรม โดยเริ่มต้นเมื่อมีการเขียนข้อมูลเก็บไว้ในรีจีสเตอร์ SBUF หลังจากนั้นข้อมูลจะถูกจัดการโดยวิธีทางฮาร์ดแวร์ในการเลื่อนบิตเพื่อส่งสัญญาณออกไปภายนอก หลังจากมีการส่งข้อมูลออกไปจนครบแล้ว จึงจะเซตบิตโดยกำหนดค่าของแฟลก TI ในรีจีสเตอร์ SCON ให้เป็นสถานะ "1" เพื่อแจ้งว่ารีจีสเตอร์ SBUF ว่างแล้วและพร้อมที่จะส่งข้อมูลไบต์ต่อไปได้

การรับส่งข้อมูลจากพอร์ตอนุกรมจะต้องเริ่มต้นโดยกำหนดค่าของบิต REN ที่อยู่ในรีจีสเตอร์ SCON ให้มีค่าเป็นสถานะ "1" หลังจากนั้นเมื่อมีการรับข้อมูลเข้ามาจากภายนอกก็จะเลื่อนข้อมูลไปโดยอัตโนมัติ และเมื่อบิตสุดท้ายถูกเลื่อนบิตเข้ามาเรียบร้อยแล้ว ข้อมูลจะถูกย้ายมาเก็บไว้ที่รีจีสเตอร์ SBUF และจะเซตที่บิต RI ให้เป็นสถานะ "1" ซึ่งส่งผลให้เกิดการอินเตอร์รัปต์โปรแกรมขึ้น

โหมดการติดต่อทางพอร์ตอนุกรม

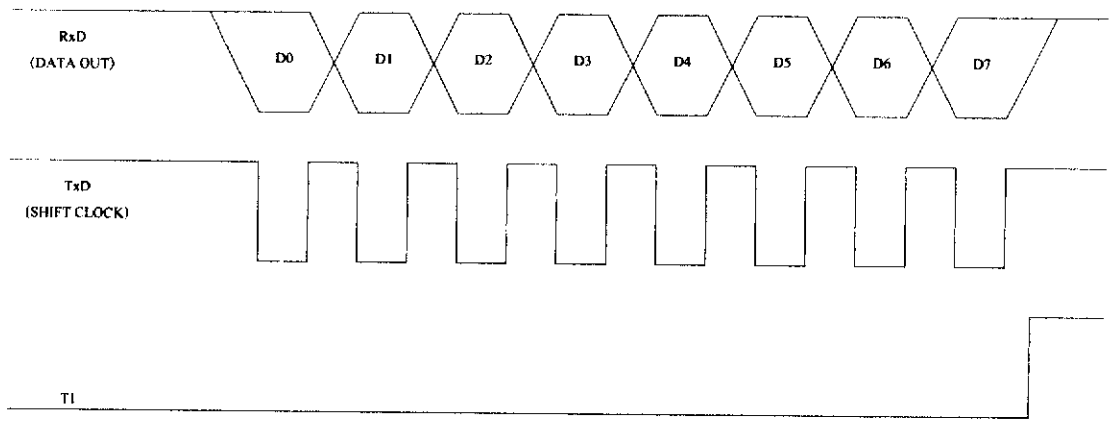
การสื่อสารอนุกรมของไมโครคอนโทรลเลอร์จะแบ่งออกได้เป็น 4 โหมดด้วยกัน และในแต่ละโหมดจะสามารถสรุปหน้าที่ได้ดังนี้

โหมด 0 จะเป็นการรับส่งข้อมูลขนาด 8 บิต แบบอนุกรมการส่งข้อมูลจะเลื่อนออกไปทีละบิต โดยจะใช้งานของขา RxD เพียงขาเดียวและไม่มีการส่งบิตเริ่มต้น (Start bit) ส่วนขา TxD จะใช้เป็นขาของสัญญาณนาฬิกาในการให้จังหวะการเลื่อนข้อมูลกับวงจรภายนอก (Shift clock) อัตราการรับส่งข้อมูล (Baud rate) จะเป็น 1/12 เท่าของสัญญาณนาฬิกา

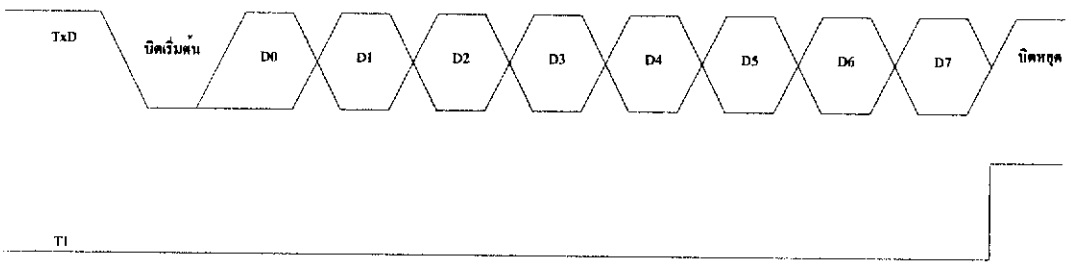
การรับและส่งข้อมูลจะเริ่มจากบิตต่ำ (LSB) ก่อน ใช้สำหรับเป็นชิฟต์รีจีสเตอร์ (Shift Register) จุดประสงค์เพื่อใช้ในการขยายพอร์ตอินพุตหรือพอร์ตเอาต์พุตให้มีจำนวนมากกต่ในโหมด 0 เรามักจะไม่ค่อยนิยมนำมาใช้งาน เพราะไอซีไมโครลคอนโทรลเลอร์ในปัจจุบันที่เราใช้อยู่

นั้นมีจำนวนพอร์ตที่มากพอและมีไอซีเบอร์อื่น ๆ ในตระกูลเดียวกันให้เลือกจำนวนพอร์ตใช้งานมากมายอยู่แล้ว

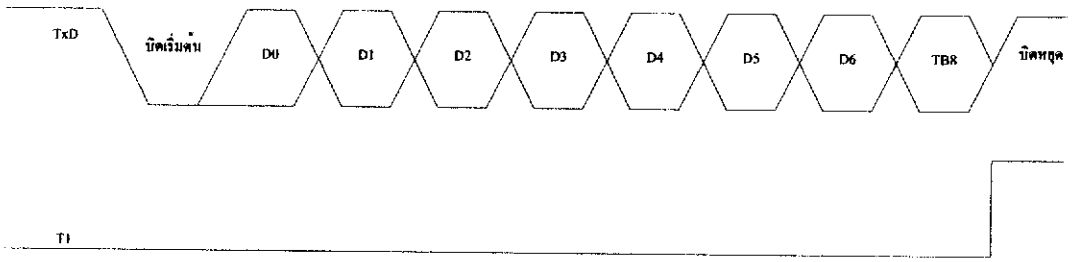
โหมด 1 จะเป็นการรับและส่งข้อมูลขนาด 10 บิตแบบ UART (Universal Asynchronous Receiver Transmitters) สามารถใช้ติดต่าสื่อสารอนุกรมกับมาตราฐานของ RS-232C ของไมโครคอมพิวเตอร์ได้ ซึ่งข้อมูลอนุกรม 10 บิต จะเข้ามาทางขา RxD และส่งข้อมูลออกแบบอนุกรมทางขา TxD โดยจะประกอบด้วย 1 บิตแรกเป็นบิตเริ่มต้น (Start bit ค่า 0) ,8 บิตต่อมาจะเป็นบิตของข้อมูล(การับ/ส่งจะเริ่มจากบิตต่ำก่อน)และบิตหยุดอีก 1 บิต (Stop bit ค่า 1)



รูปที่ 2.18 แสดงสัญญาณการรับและส่งข้อมูลในโหมด 0



รูปที่ 2.19 แสดงสัญญาณการส่งข้อมูลในโหมด 1



รูปที่ 2.20 แสดงสัญญาณการส่งข้อมูลในโหมด 2

ส่วนทางด้านรับข้อมูลจะนำค่าบิตหยุดที่รับเข้ามาได้นำไปเก็บไว้ในบิต RB8 ที่อยู่ในรีจีสเตอร์ SCON และความเร็วของการส่งข้อมูลในโหมด 1 จะขึ้นอยู่กับบิต SMOD ที่อยู่ในรีจีสเตอร์ PCON และอัตราโดเวอร์โฟลว์ของไทม์เมอร์ 1 ซึ่งอัตราการรับส่งข้อมูลในโหมดนี้สามารถกำหนดได้ตามต้องการ

โหมด 2 จะเป็นการรับส่งข้อมูลขนาด 11 บิตแบบ UART ข้อมูลแบบอนุกรมจะถูกรับเข้ามาทางขา RxD และส่งข้อมูลออกไปทางขา TxD ซึ่งข้อมูล 11 บิต ประกอบด้วยบิตแรกจะเป็นบิตเริ่มต้น 9 บิตต่อมาจะเป็นบิตของข้อมูลและบิตสุดท้ายจะเป็นบิตหยุด 1 บิต

สำหรับข้อมูลในบิตที่ 9 จะกำหนดไว้ใน TB8 ที่อยู่ในรีจีสเตอร์ SCON ซึ่งสามารถกำหนดเป็น 0 หรือ 1 ก็ได้ นิยมนำมาใช้ในการส่งบิตเพื่อตรวจสอบการส่งข้อมูล (Parity bit)

โหมด 3 เป็นการรับส่งข้อมลแบบ 11 บิตแบบ UART เหมือนกับโหมด 2 แต่ในโหมด 3 สามารถกำหนดอัตราความเร็วในการรับส่งข้อมูลได้ตามต้องการ

หมายเหตุ UART เป็นการส่งข้อมูลแบบอนุกรม โดยขึ้นอยู่กับความพร้อมของทางด้านส่งและด้านรับ เป็นการส่งข้อมูลโดยเพิ่มเติมข้อมูลบางอย่างเข้าไป (Start bit, Stop bit, Parity bit) เพื่อให้การรับและการส่งข้อมูลสามารถจะทำงานให้มีความถูกต้องของข้อมูลมากยิ่งขึ้น

การสื่อสารเพื่อเชื่อมต่อไมโครคอนโทรลเลอร์ในการรับและส่งข้อมูลทางอนุกรมมีอยู่ด้วยกัน 2 ระบบคือ

1.Single Processor System คือระบบการสื่อสารโดยใช้ไมโครคอนโทรลเลอร์ 2 ตัวเชื่อมต่อหากัน

2. Multi Processors System คือระบบการสื่อสารแบบมัลติโปรเซสเซอร์ โดยใช้ไมโครคอนโทรลเลอร์ 1 ตัว เป็นตัวแม่ (Master) และสามารถที่จะเชื่อมต่อกับไมโครคอนโทรลเลอร์ที่เป็นตัวลูก (Slave) ได้อีกเป็นจำนวนหลาย ๆ ตัว

รีจีสเตอร์ควบคุมและรับส่งข้อมูลของพอร์ตอนุกรม

รีจีสเตอร์ที่ใช้ในการติดต่าสื่อสารทางพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ประกอบด้วยรีจีสเตอร์ SCON ทำหน้าที่การควบคุมการทำงานรีจีสเตอร์ SBUF จะใช้เก็บข้อมูลที่รับหรือส่งและรีจีสเตอร์ PCON จะใช้กำหนดอัตรารับส่ง โดยรีจีสเตอร์แต่ละตัวจะมีหน้าที่และการทำงานในแต่ละบิตดังต่อไปนี้

รีจีสเตอร์ SCON (Serial Port Control Register) เป็นรีจีสเตอร์ขนาด 8 บิต อยู่ในส่วนของรีจีสเตอร์พิเศษ (Special Function Register) ในตำแหน่งแอดเดรสที่ 98H และสามารถเข้าถึงข้อมูลแบบไบต์และแบบบิตได้ โดยจะทำหน้าที่ควบคุมการทำงานของพอร์ตอนุกรม การเลือกโหมดการทำงานและเก็บข้อมูลในบิตที่ 9 (ซึ่งโดยปกติข้อมูลจะมี 8 บิต อยู่ในรีจีสเตอร์ SBUF) ของการรับข้อมูล (RB8) และส่งข้อมูล (TB8) รายละเอียดของแต่ละบิตมีดังต่อไปนี้

SM0, SM1 (Serial port mode bit 0-1) เป็นบิตที่ใช้ในการกำหนดโหมดการทำงานของพอร์ตอนุกรมจำนวน 4 โหมด

SM2 เป็นบิตทำหน้าที่ควบคุมการทำงานและเลือกลักษณะการเชื่อมต่อสื่อสารระหว่างไมโครคอนโทรลเลอร์แบบ

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

ตารางที่ 2.10 แสดงบิตต่าง ๆ ของรีจีสเตอร์ SCON

SM0	SM1	โหมด	การทำงาน	อัตรารับ-ส่ง
0	0	0	Shift register	Fosc/12
0	1	1	8bit UART	Variable
1	0	2	9bit UART	Fosc/32 หรือ Fosc/64
1	1	3	9bit UART	Variable

ตารางที่ 2.11 แสดงการกำหนดบิต SM0 และ SM1 เพื่อกำหนดโหมดการทำงาน

Single Processor System หรือ Multi Processors System โดยกำหนดให้

SM2 = 1 เป็นการเลือกแบบ Multi Processors System คือ ระบบการสื่อสารแบบใช้ซีพียูหลาย ๆ ตัว ร่วมกันทำงานจะใช้งานในโหมด 2 หรือ โหมด 3

SM2 = 0 เป็นการเลือกแบบ Single Processor System โดยสามารถใช้ได้กับทุกโหมด (การใช้งานในโหมด 0 ต้องกำหนดให้ SM2 = 0)

ในกรณีที่เลือกให้ SM2 = 1 แบบ Multi Processors System ถ้าข้อมูลที่รับเข้ามาบิตที่ 9 (อยู่ในบิต RB8) มีค่าเป็น "1" ทำให้แฟลกอินเตอร์รัปต์ทางด้านรับจะถูกเซตให้เป็น 1 (RI=1) แต่ถ้าข้อมูลในบิตที่ 9 รับเข้ามามีค่าเป็น "0" จะทำให้แฟลกอินเตอร์รัปต์ทางด้านรับเป็น 0 (RI = 0)

การทำงานในโหมด 1 ถ้าให้ SM2 = 1 แฟลกอินเตอร์รัปต์ทางด้านรับ (แฟลก RI) จะไม่ถูกเซตหารข้อมูลที่รับเข้ามาไม่มีบิตหยุด

REN (Enable Serial Reception) เป็นบิตที่ควบคุมการรับข้อมูลของพอร์ตอนุกรม กำหนดสถานะของบิตได้โดยซอร์ฟแวร์

1 = ให้มีการรับข้อมูล

0 = ไม่ให้มีการรับข้อมูล

TB8 (Transmit bit D8)เป็นบิตของข้อมูลบิตที่ 9 ในการส่งข้อมูลใช้งานโหมด2 และโหมด 3 กำหนดสถานะของบิตได้โดยซอร์ฟแวร์

RB8 (Receive bit D8)เป็นบิตข้อมูลบิตที่ 9 ในการรับข้อมูล โดยใช้งานโหมด 2 และโหมด 3 หากใช้งานในโหมด 1 ถ้ากำหนดให้ SM2 = 0 บิตนี้จะเป็นค่าของบิตหยุดที่รับเข้ามา สำหรับโหมด 0 จะไม่ใช้งานบิตนี้

TI (Transmit Interrupt Flag) เป็นบิตที่ใช้งานในการอินเตอร์รัปต์ด้านส่งข้อมูลและจะถูกเซตทางฮาร์ดแวร์เมื่อมีการส่งข้อมูลเสร็จสิ้นลงในบิตที่ 3 ของโหมด ของโหมด 0 (Shift register) หรือเมื่อเริ่มต้นส่งบิตหยุดในโหมด 1,2 หรือ 3 และจะต้องเคลียร์บิตนี้ด้วยวอร์ฟแวร์ทุกครั้ง เมื่อโปรแกรมตอบสนองอินเตอร์รัปต์ของการส่งข้อมูลเรียบร้อยแล้ว

RI (Receive Interrupt Flag) เป็นบิตที่ใช้งานในการอินเตอร์รัปต์ทางด้านรับข้อมูลจะถูกเซตทางฮาร์ดแวร์เมื่อมีการรับข้อมูลเสร็จสิ้นลงในบิตที่ 8 ในโหมด 0 (Shift register) และจะต้องเคลียร์บิตนี้ด้วยซอร์ฟแวร์ทุกครั้งเมื่อโปรแกรมตอบสนองการอินเตอร์รัปต์ ของการรับข้อมูลเรียบร้อยแล้วหรืออาจกล่าวได้ว่าถ้าบิต RI ถูกเซตเมื่อใด หมายถึงข้อมูลได้เข้ามาเก็บไว้ที่รีจีสเตอร์ SBUF จนครบทั้ง 8 บิตแล้ว สามารถที่จะอ่านข้อมูลจากรีจีสเตอร์ SBUF ได้

รีจีสเตอร์ SBUF (Serial data buffer register) เป็นรีจีสเตอร์ขนาด 8 บิตหรือ 1 ไบต์ มีแอดเดรสอยู่ตำแหน่งที่ 99H และเข้าถึงข้อมูลแบบไบต์ได้อย่างเดียว จะทำหน้าที่รับและส่งออกไปยังพอร์ตอนุกรมของไมโครคอนโทรลเลอร์

ในการอ่านข้อมูลจากภายนอกที่รับเข้ามาทางพอร์ตอนุกรมจะต้องอ่านค่ารีจีสเตอร์ SBUF ซึ่งเป็นบัฟเฟอร์เก็บข้อมูลที่รับเข้ามาได้จากภายนอก ในทำนองเดียวกันขณะที่ต้องการส่งข้อมูล เราก็จะนำเอาค่าข้อมูลที่ส่งออกไปไว้ที่ในรีจีสเตอร์ SBUF ก่อน หลังจากนั้นจึงส่งออกไป โดยใช้คำสั่งการโอนย้ายข้อมูลแบบไบต์ เช่น MOV SBUF,#20H หรือ MOV SBUF,@R1 ก็ได้

การรับข้อมูลในโหมด 0 จะเริ่มต้นรับ เมื่อค่าของบิต RI = 0 และ REN = 1 ส่วนในโหมดอื่น ๆ การรับข้อมูลจะเริ่มต้นเมื่อกำหนดบิต REN = 1 และมี Start bit เข้ามาที่ขา RxD

รีจีสเตอร์ PCON (Power control) เป็นรีจีสเตอร์ขนาด 1 ไบต์ มีแอดเดรสอยู่ตำแหน่งที่ 87H เข้าถึงข้อมูลได้แบบไบต์อย่างเดียวเท่านั้น โดยจะประกอบด้วยบิตดังต่อไปนี้

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SMOD	-	-	-	GF1	GF0	PD	IDL

ตารางที่ 2.12 แสดงรีจีสเตอร์ PCON เพื่อกำหนดอัตราการรับส่งข้อมูล

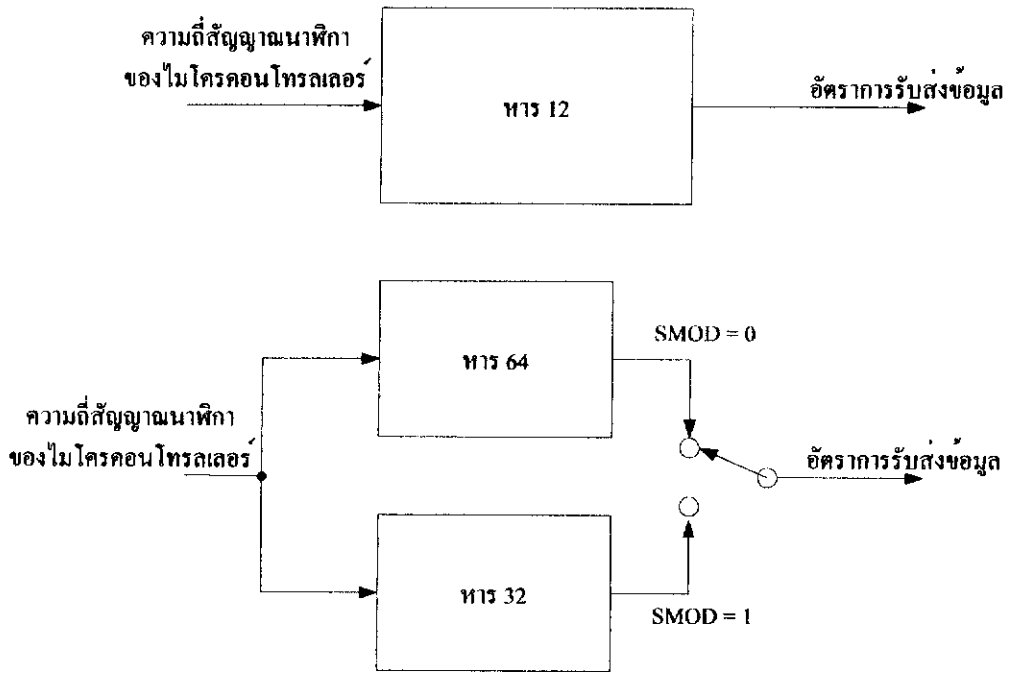
PCON.7 SMOD ในกรณีที่ใช้ไทเมอร์ 1 เป็นตัวกำหนดอัตรารับส่ง (BAUD rate) และหากกำหนดให้บิตนี้มีค่าเป็น "0" ในการใช้งานกับพอร์ตสื่อสารอนุกรมโหมด 1,2 และโหมด 3 ค่าอัตรารับส่งจะเพิ่มขึ้นเป็นสองเท่า

รีจีสเตอร์ PCON ไม่สามารถอ้างตำแหน่งของบิตได้ แต่จะใช้คำสั่งทางลอจิกของการ OR เช่น ORL PCON,#80H จะเป็นการเซตบิตที่ 7 ของรีจีสเตอร์ PCON และการกำหนดให้บิตมีสถานะเป็น "0" หรือเคลียร์บิตจะใช้การ AND เช่น ANL PCON,#01111111B จะเป็นการเคลียร์บิตที่ 7 ของรีจีสเตอร์ PCON

การกำหนดอัตรารับและส่งข้อมูล

โหมด 0 อัตรารับส่งในโหมด 0 ไม่สามารถกำหนดอัตรารับส่งเองได้แต่จะมีค่าเท่ากับความถี่สัญญาณนาฬิกาไมโครคอนโทรลเลอร์หารด้วย 12 หรืออาจกล่าวได้ว่าขึ้นกับค่าความถี่ของคริสตอลที่นำมาต่อใช้งานแล้วหารด้วย 12 นั่นเอง

อัตรารับส่งโหมด 0 = 1/12 เท่าของความถี่สัญญาณนาฬิกา



รูปที่ 2.21 ตัวอย่างอัตรารับส่งข้อมูลที่กำหนดจากการใช้ไทเมอร์ 1
เมื่อใช้บอดเรตค่ามาตราฐานต่าง ๆ

โหมด 2 อัตรารับส่งในโหมด 2 เลือกได้ 2 อัตราความเร็วในการรับส่งข้อมูล โดยหาได้จากความถี่ของสัญญาณนาฬิกาของไมโครคอนโทรลเลอร์หารด้วย 32 หรือหารด้วย 64 เรียกว่า SMOD0 และ SMOD1 ซึ่งขึ้นอยู่กับการกำหนดค่าสถานะของบิต SMOD ที่อยู่ใน
รีจีสเตอร์ PCON เป็นตัวเลือก

ถ้าบิต SMOD = 0อัตรารับส่งโหมด 2 = 1/64 เท่าของความถี่สัญญาณนาฬิกา

ถ้าบิต SMOD = 1 อัตรารับส่งโหมด 2 = 1/32 เท่าของความถี่สัญญาณนาฬิกา

หลังจากที่เรารีเซตระบบของไมโครคอนโทรลเลอร์ค่าข้อมูลในบิต SMOD จะเป็นสถานะ "0" เสมอ ดังนั้นเราสามารถเขียนเป็นสูตรสำหรับการคำนวณหาค่าอัตรารับส่งหรือ
บอดเรต(Baud rate) ได้ดังสมการต่อไปนี้

อัตรารับส่ง (Baud rate)

$$\text{โหมด 2} = \frac{[2^{SMOD} \times (\text{ความถี่สัญญาณนาฬิกาที่ใช้})}{64}$$

ในกรณีที่เราใช้คริสตอลค่า 11.0592 MHz จะได้ค่าอัตรารับส่งสูงสุด = 345.6 k

ในกรณีที่เราใช้คริสตอลค่า 12 MHz จะได้ค่าอัตรารับส่งสูงสุด = 375 K

โหมด 1 และโหมด 3 จะมีอัตราการรับส่งข้อมูลโดยถูกกำหนดได้ตามต้องการ โดยใช้อัตราการเกิดโอเวอร์โฟลว์ของไทเมอร์ 1 หรือไทเมอร์ 2 เป็นตัวกำหนด

การใช้ไทเมอร์ 1 กำหนดอัตรารับส่ง

เมื่อใช้ไทเมอร์ 1 เป็นตัวสร้างอัตราการรับส่งข้อมูลของพอร์ตอนุกรมในโหมด 1 หรือโหมด 3 สังเกตได้ว่าเมื่อเกิดไอเวอร์โฟลว์ในไทเมอร์ตัวใดจะทำให้เกิดสัญญาณอินเตอร์รัปต์เพื่อบอกให้ซีพียูรับทราบ ดังนั้นในขณะใช้ไทเมอร์ 1 เพื่อสร้างอัตรารับและส่งข้อมูล จะต้องไม่ให้มีการร้องขออินเตอร์รัปต์ที่เกิดจากไทเมอร์ 1 ในกรณีใด ๆ ในระหว่างนั้นอีก (โดยการควบคุมที่รีจีสเตอร์ IE) อัตราการรับและส่งข้อมูลมาจากอัตราการเกิดโอเวอร์โฟลว์ของไทเมอร์ 1 และค่าของบิต SMOD ที่อยู่ในรีจีสเตอร์ PCON สามารถที่จะเขียนเป็นสูตรสำหรับการคำนวณหาค่าอัตรารับส่งได้ดังสมการต่อไปนี้

$$\text{อัตรารับส่งในโหมด 1 และ 3} = \frac{2^{SMOD}}{32} \times (\text{อัตราการเกิดโอเวอร์ของไทเมอร์ 1})$$

ในการใช้งานสื่อสารข้อมูลแบบอนุกรมนั้น เราจะนิยมใช้งานไทเมอร์ 1 ในลักษณะของโหมด 2 (Auto-reload กำหนดค่าการควบคุมที่รีจีสเตอร์ TMOD = 0010xxxx) การคำนวณหาค่าอัตรารับส่งจะได้สมการดังต่อไปนี้

$$\text{อัตรารับส่งในโหมด 1 และ 3} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator}}{12 \times \left[256 - (TH1)\right]}$$

เราสามารถที่จะให้ไทเมอร์ 1 มีอัตราการรับส่งต่ำ ๆ ได้ โดยใช้ไทเมอร์ 1 ในโหมด 1 ทำงานลักษณะของตัวจับเวลาแบบ 16 บิต (มีค่าของการควบคุมที่รีจีสเตอร์ TMOD = 0001xxxx) และมีการอินเตอร์รัปต์จากไทเมอร์ 1 โดยให้โปรแกรมตอบสนองการอินเตอร์รัปต์ของไทเมอร์ 1 แล้วจะกำหนดค่าเริ่มต้นใหม่ (Reload) ให้กับตัวจับเวลา ซึ่งเป็นการทำงานแบบ 16 บิต ทางซอร์แวร์ (Software reload) เนื่องจากการทำงานในโหมด 1 ของไทเมอร์ 1 ไม่สามารถโหลดค่าใหม่เองด้วยฮาร์ดแวร์ได้

ตารางในรูปที่ 7 เป็นอัตรารับส่งข้อมูลที่กำหนดจากการใช้ไทเมอร์ 1 เมื่อใช้บอดเรตค่ามาตราฐานต่าง ๆ และการกำหนดค่าของรีจีสเตอร์ TMOD ที่บิต C/T และรีจีสเตอร์ PCON ที่บิต SMOD เพื่อใช้งาน

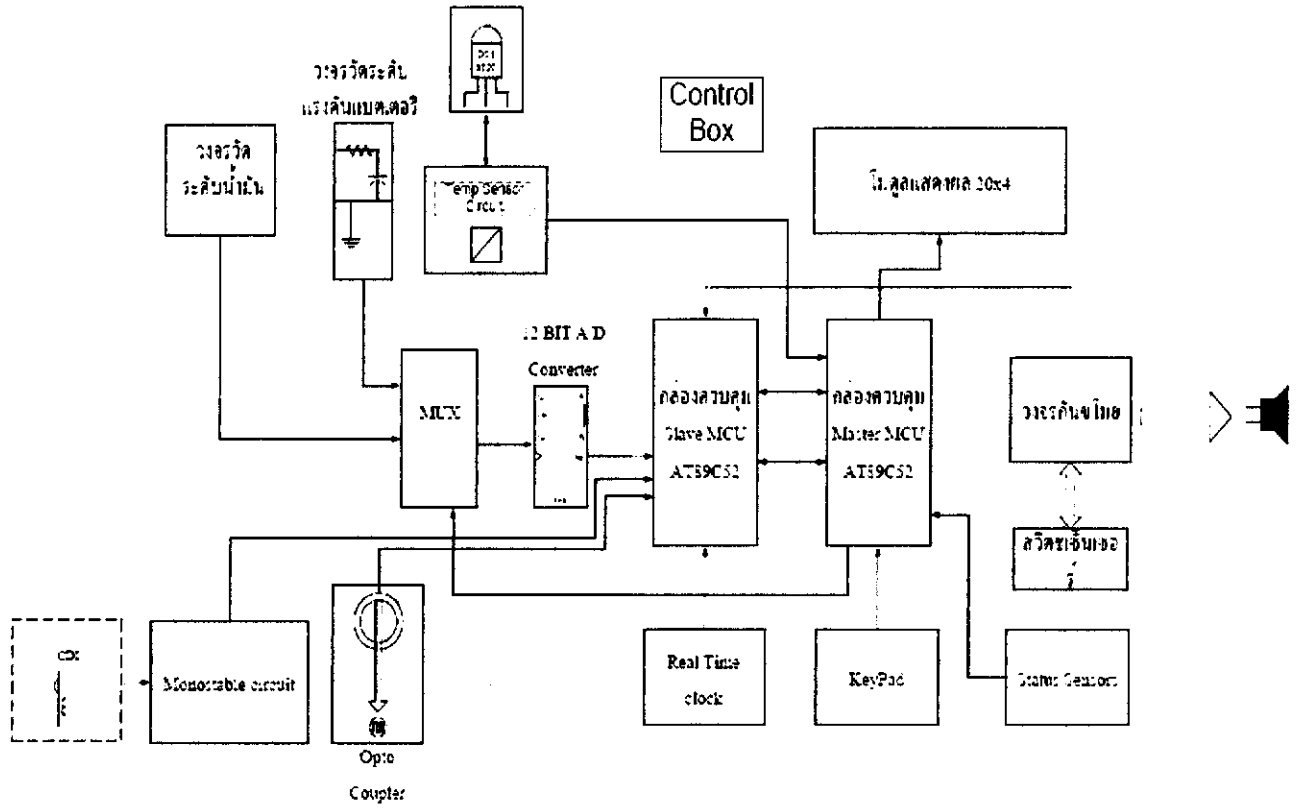
Baud rate	Fosc (MHz)	บิต SMOD	Timer 1		
			บิต C/T	MODE	Reload Value
62.5 K	12.00	1	0	2	FFH
19.25K	11.059	1	0	2	FDH
9600	11.059	0	0	2	FDH
4800	11.059	0	0	2	FAH
2400	11.059	0	0	2	F4H
1200	11.059	0	0	2	E8H
137.5	11.059	0	0	2	1DH
110	6	0	0	2	72H
110	12	0	0	1	FEEBH

ตารางที่ 2.13 รับส่งข้อมูลที่กำหนดจากการใช้ไทเมอร์ 1 เมื่อใช้บอดเรตค่ามาตราฐานต่าง ๆ และการกำหนดค่าของรีจีสเตอร์ TMOD ที่บิต C/T และรีจีสเตอร์ PCON ที่บิต SMOD เพื่อใช้งาน

จะสังเกตได้ว่าการใช้งานจากความถี่คริสตอลเรามักจะนิยมะเลือกใช้ค่าความถี่ 11.0592MHz เนื่องจากการกำหนดค่าอัตราการรับส่งข้อมูลจะสามารถกำหนดค่าบอดเรตในโหมด 1 และโหมด 3 ได้ และเป็นค่าที่มาตราฐานเช่น 1200, 2400, 4800, 9600, 19200 ดังนั้นจึงเป็นเหตุผลที่เราเลือกใช้คริสตอลค่าความถี่ 11.0592 MHz มากกว่าค่า 12 MHz

บทที่ 3 ขั้นตอนและหลักการทำงาน

ในด้านโครงสร้างของเครื่องแสดงผลแบบหลายฟังก์ชันและระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์นั้นสามารถแยกส่วนประกอบและการทำงานของส่วนต่างๆได้ดังนี้



รูปที่ 3.1 แสดงไดอะแกรมการทำงานของเครื่องแสดงผลแบบหลายฟังก์ชันและระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์

3.1 วงจรวัดอุณหภูมิ

ในวงจรส่วนนี้เป็นการใช้ไอซีวัดอุณหภูมิของดัลดัสเบอร์ DS18B20 เป็นตัวตรวจจับอุณหภูมิ ซึ่งเป็นการส่งข้อมูลแบบ ONE-WIRE® คือมีการรับ-ส่งข้อมูลผ่านสายสัญญาณเพียงเส้นเดียว แล้วส่งข้อมูลไปให้ไมโครคอนโทรลเลอร์ประมวลผลเพื่อไปแสดงผลที่จอแสดงผลที่หน้าจอแอลซีดี ซึ่งมีความละเอียดของอุณหภูมิที่ 0.5 องศาเซลเซียสโดยที่มีรายละเอียดและการทำงานต่างๆ ดังนี้

ไอซีตรวจจับอุณหภูมิ DS1820

เป็นไอซีตรวจจับอุณหภูมิที่ใช้การติดต่อแบบระบบบัสหนึ่งสาย มีขาต่อใช้งานเพียง 3 ขา คือ DQ ซึ่งเป็นขาเชื่อมต่อกับระบบบัส, ขาต่อไฟเลี้ยงภายนอก และขากราวด์ ดังแสดงการจัดขาของไอซี DS1820 ในรูปที่ 3.2

รูปที่ 3.2 การจัดขาของ DS1820

หัวใจหลักสำคัญของ DS1820 อยู่ที่ตัวตรวจจับอุณหภูมิและหน่วยความจำความเร็วสูงที่เรียกว่า สแครตช์แพด (scratchpad) ซึ่งมีขนาด 9 ไบต์ มีการจัดสรรหน่วยความจำส่วนนี้แสดงดังในรูปที่ 3.3

	ไบต์
ข้อมูลอุณหภูมิไบต์ต่ำ (TL)	0
ข้อมูลอุณหภูมิไบต์สูง	1
ข้อมูลอุณหภูมิไบต์ค่าสูง	2
ข้อมูลอุณหภูมิค่าต่ำ (TL)	3
สำรองไว้	4
สำรองไว้	5
รีจิสเตอร์เก็บค่าการนับ	6
รีจิสเตอร์เก็บค่าการนับต่อ °C	7
CRC	8

รูปที่ 3.3 การจัดสรรพื้นที่ของสแครตช์แพดใน DS1820

เมื่อวัดอุณหภูมิได้ก็จะนำค่าที่วัดได้นี้มาเก็บไว้ในสแครตแพดซ์ที่ไบต์ 0 และ 1 ทั้งนี้เนื่องจากไอซี DS1820 สามารถให้ข้อมูลของอุณหภูมิได้ละเอียดถึง 16 บิต เมื่อนำมาแปลงเป็นข้อมูลเลขฐานสิบจึงสามารถแสดงความละเอียดของค่าอุณหภูมิได้ถึง 0.5 องศาเซลเซียสและ 0.9 องศาฟาเรนไฮต์ โดยมีย่านวัดอุณหภูมิ -55 ถึง +125 องศาเซลเซียสหรือ -67 ถึง +257 องศาฟาเรนไฮต์ โดยค่าขององศาฟาเรนไฮต์ต้องใช้การแปลงหน่วยเข้ามาช่วย ใช้เวลาในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลประมาณ 200 มิลลิวินาที สามารถกำหนดขอบเขตของอุณหภูมิที่ทำการวัดได้ และให้แจ้งเตือนเมื่อค่าของอุณหภูมิสูงขึ้นหรือลดต่ำลงถึงค่าที่กำหนด โดยค่าอุณหภูมิที่กำหนดนี้จะเก็บไว้ในสแครตซ์แพดในไบต์ 2 และ 3

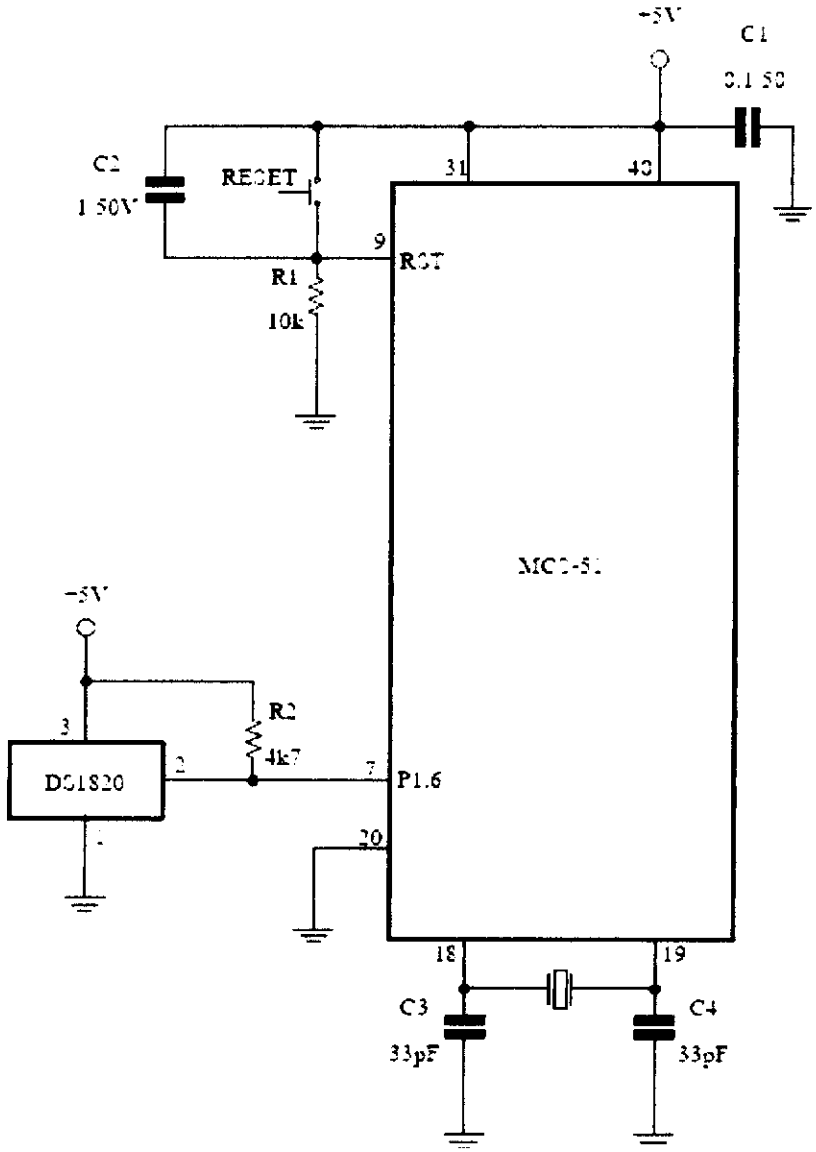
คำสั่งเพื่อควบคุมการทำงานของ DS1820

ในการติดต่อกับไอซี DS1820 จะมีคำสั่งที่ต้องส่งให้แก่ DS1820 เพื่อกำหนดรูปแบบการทำงาน คำสั่งที่ใช้มากที่สุดมีด้วยกัน 3 คำสั่งดังนี้

1. คำสั่งไม่ติดต่อกับหน่วยความจำรอมหรือสคิปรอม (Skip ROM) เนื่องจากในการใช้งาน DS1820 โดยปกติแล้วจะมี DS1820 อยู่บนสายสัญญาณเพียงตัวเดียว จึงไม่จำเป็นต้องใช้ข้อมูลกำหนดแอดเดรส ดังนั้นจึงไม่ต้องติดต่อกับหน่วยความจำรอมเพื่ออ่านข้อมูล ข้อมูลของคำสั่งสคิปรอมที่ต้องส่งให้ DS1820 คือ 0CCH
2. คำสั่งแปลงอุณหภูมิ (Convert T) มีค่าเท่ากับ 44H เมื่อคำสั่งนี้ให้ DS1820 จะต้องทำการวนลูปรออย่างน้อย 200 มิลลิวินาที เพื่อให้ DS1820 ได้ใช้เวลานี้ในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลมาเก็บไว้ในสแครตซ์แพด
3. คำสั่งอ่านข้อมูลจากสแครตซ์แพด (Read Scratchpad) มีค่าเท่ากับ 0BEH เมื่อส่งคำสั่งนี้ DS1820 จะทยอยส่งข้อมูลค่าอุณหภูมิออกมาทั้งหมด 9 ไบต์

การเชื่อมต่อกับไมโครคอนโทรลเลอร์

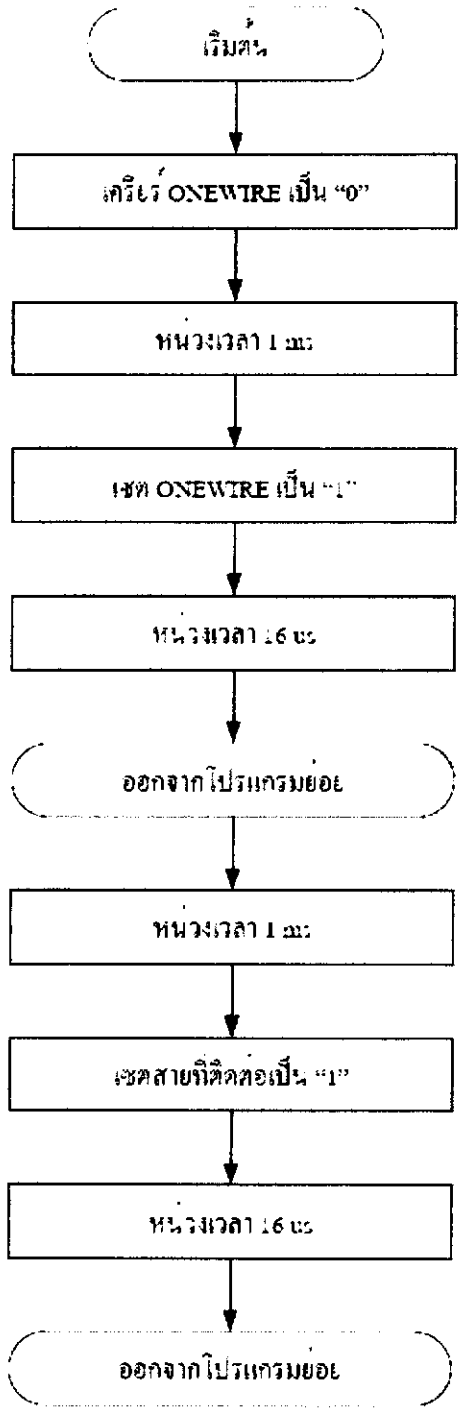
วงจรการเชื่อมต่อแสดงดังในรูปที่ 3.4 ใช้ขาพอร์ตเพียง 1 ขาเท่านั้นสำหรับการเชื่อมต่อกับ DS1820 โดยต้องมีตัวต้านทาน 4.7 กิโลโอห์ม ต่อพูลอัปกับไฟเลี้ยง +5V จากนั้นจึงทำการเขียนโปรแกรมเพื่อติดต่อกัน โดยใช้รูปแบบการติดต่อตามมาตรฐานระบบบัสหนึ่งสายของดัลลัส



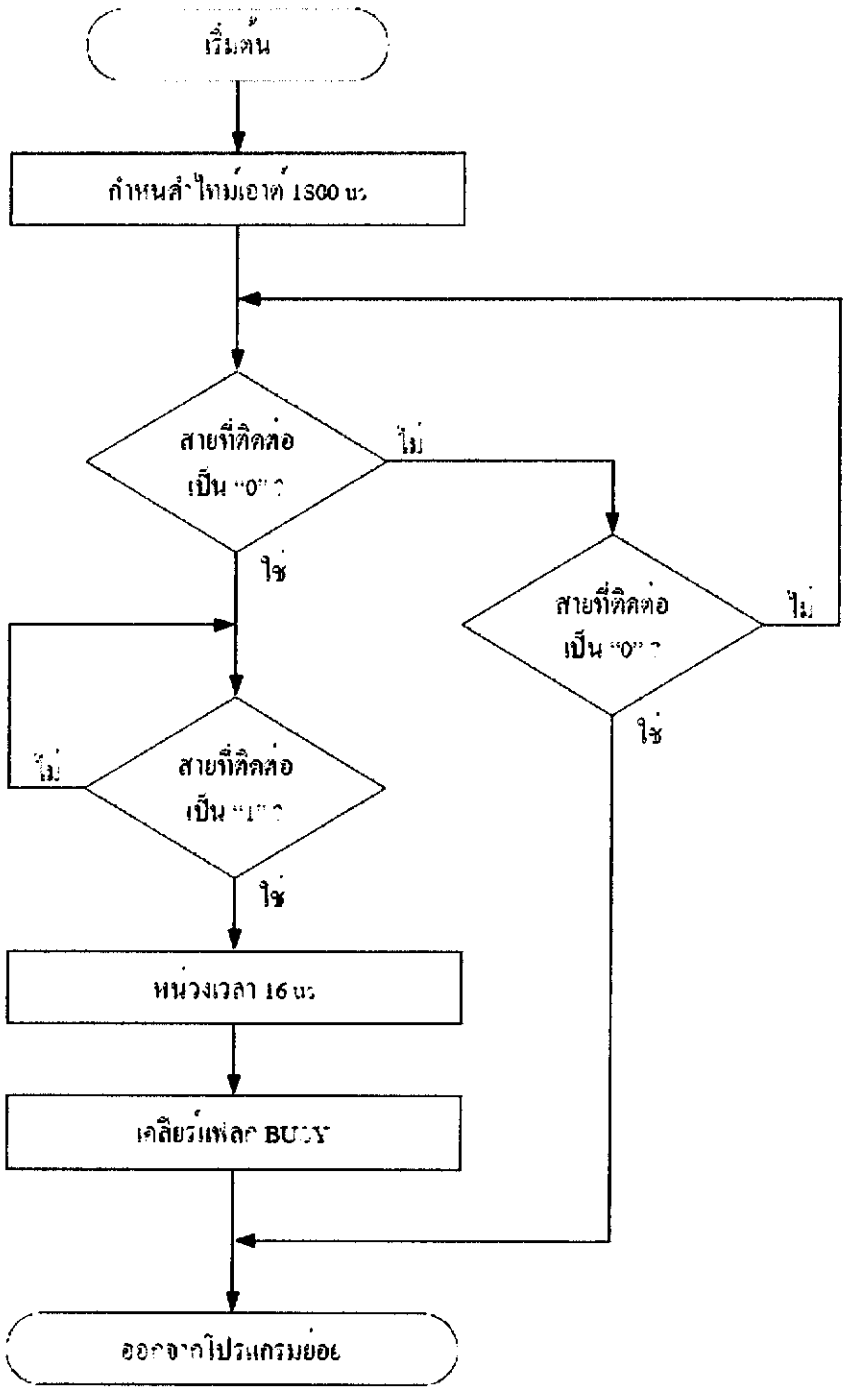
รูปที่ 3.4 การเชื่อมต่อ DS1820 กับไมโครคอนโทรลเลอร์ MCS-51

การเขียนโปรแกรมเพื่อติดต่อกับ DS1820

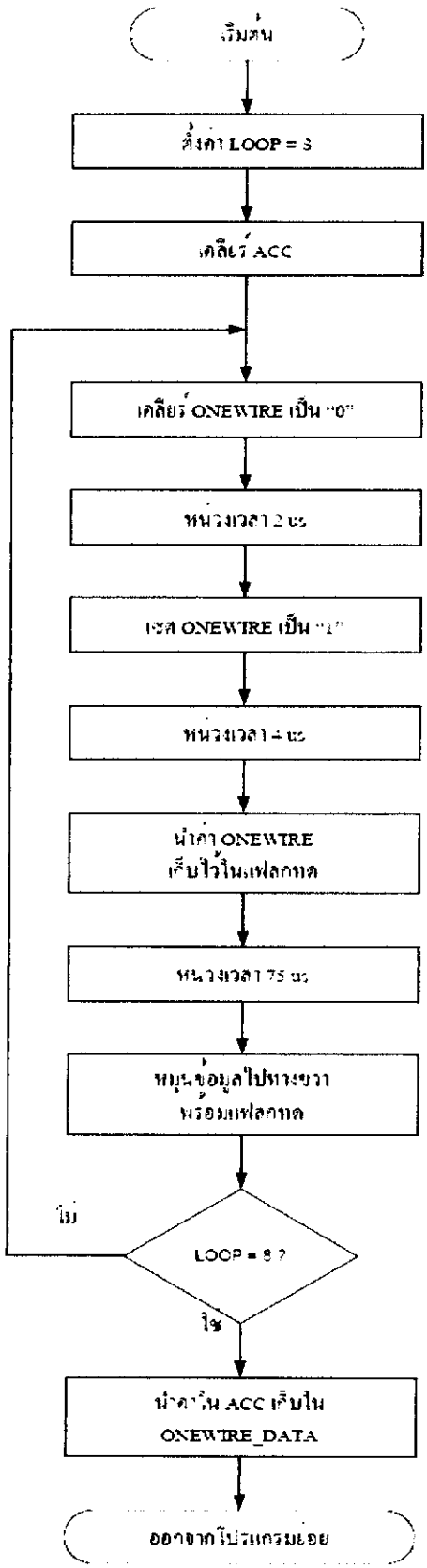
จากรายละเอียดของรูปแบบการสื่อสารในระบบบัสหนึ่งสายที่กล่าวมาในบทที่แล้ว สามารถนำมาใช้เพื่อเป็นข้อมูลในการเขียนโปรแกรมติดต่อ โดยจะต้องเขียนโปรแกรมย่อยเพื่อสร้างไทม์สล็อตของฟังก์ชั่นต่างๆ ดังแสดงรายละเอียดของโฟลวชาร์ตและโปรแกรมย่อยในรูปที่ 3.5 ถึง รูปที่ 3.8 เริ่มจากโปรแกรมย่อยการรีเซต (RESET), โปรแกรมย่อยรอการตอบรับจาก DS1820 (PRESENCE), โปรแกรมย่อยการอ่านและการเขียนข้อมูลกับ DS1820



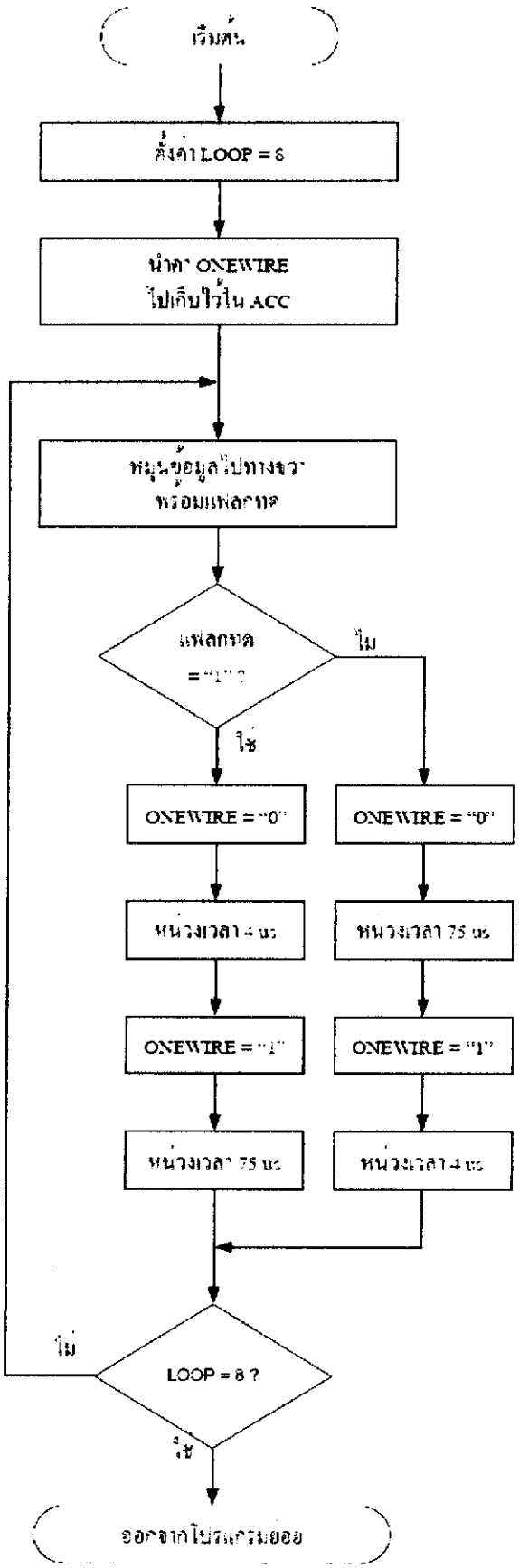
รูปที่3.5 โฟลวชาร์ตการรีเซต DS1820



รูปที่ 3.6 โฟลวชาร์ตการตอบรับจาก DS1820



รูปที่ 3.7 โฟลวชาร์ตการอ่านข้อมูลกับ DS1820



รูปที่ 3.8 โฟลวชาร์ตการเขียนข้อมูลกับ DS1820

ในโครงงานนี้การติดต่อกับ DS1820 จะมีเพียง DS1820 ตัวเดียวจึงไม่จำเป็นต้องใช้คำสั่งเพื่อติดต่อกับหน่วยความจำรอมภายใน DS1820 นั่นคือ จะติดต่อแบบสคิปรอม (Skip ROM) ดังมีรูปแบบการติดต่อสรุปอย่างเห็นได้ชัดในตารางที่ 3.1

ขั้นตอนที่	การทำงานของอุปกรณ์มาสเตอร์	ข้อมูลหรือสภาวะ	รายละเอียด
1	ตัวส่ง	รีเซต	สร้างสัญญาณรีเซต
2	ตัวรับ	ตอบรับ	รอการตอบรับจาก DS1820
3	ตัวส่ง	0CCH	คำสั่ง Skip ROM
4	ตัวส่ง	44H	คำสั่งแปลงอุณหภูมิ (Convert T)
5	ตัวรับ	ข้อมูล 1 ไบต์	อ่านแฟลก Busy 8 ครั้ง
6	ตัวส่ง	รีเซต	สร้างสัญญาณรีเซต
7	ตัวรับ	ตอบรับ	รอการตอบรับจาก DS1820
8	ตัวส่ง	0CCH	คำสั่ง Skip ROM
9	ตัวส่ง	0BEH	คำสั่งอ่านค่าจากสแครตช์แพด
10	ตัวรับ	ข้อมูล 9 ไบต์	อ่านค่าของอุณหภูมิจากสแครตช์แพด
11	ตัวส่ง	รีเซต	สร้างสัญญาณรีเซต
12	ตัวส่ง	ตอบรับ	รอการตอบรับจาก DS1820
13	-	-	ทำการคำนวณค่าที่ได้จาก DS1820 เป็นเลขฐานสิบแล้วนำไปแสดงผลหรือใช้งานอื่นต่อไป

ตารางที่ 3.1 สรุปขั้นตอนการติดต่อกับ DS1820 โดยอุปกรณ์มาสเตอร์ คือ ไมโครคอนโทรลเลอร์ MCS-51

การใช้งานไอซีตรวจจับอุณหภูมิ DS18B20 ในโครงงานนี้

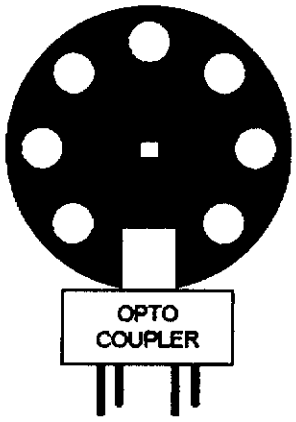
ในโครงงานนี้จะนำ DS18B20 มาทำการตรวจจับอุณหภูมิเครื่องยนต์โดยจะนำไปติดที่เสื้อสูบของรถจักรยานยนต์โดยการใช้ไอซีตัวนี้ไปยึดกับแผ่นเหล็กแล้วทำการขันสกรูยึดเข้ากับเสื้อสูบแล้วนำค่าอุณหภูมิของเครื่องยนต์ที่ตรวจจับได้ไปแสดงผลบนจอ LCD และเมื่อพบว่าอุณหภูมิของเครื่องยนต์ที่ตรวจจับได้มีค่าเกินกว่าระดับอุณหภูมิปกติ ซึ่งมีค่าประมาณ 80-90 องศาเซลเซียส

ไมโครคอนโทรลเลอร์ MCS-51 จะทำการส่งสัญญาณเตือนแก่ผู้ขับขี่รถจักรยานยนต์ทราบว่าในขณะนั้นมีอุณหภูมิที่สูงเกินไปแสดงที่หน้าจอ

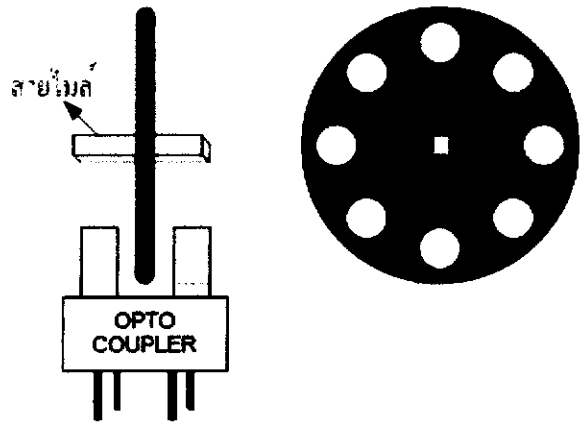
3.2 ส่วนตรวจจับความเร็วรถโดยการใช้ OPTO

เป็นการใช้ OPTO เพื่อเป็นเซนเซอร์วัดความเร็วรถโดยจะติดตั้งอุปกรณ์ คือล้อเจาะรู 8 รู ดังรูป 3.9 ไว้ที่สายไมล์ เพื่อเป็นตัวให้ OPTO ตรวจจับแสงซึ่งจะสามารถตรวจนับพัลส์ขอบขาลงได้ โดยใช้อินเตอร์รัปต์ของไมโครคอนโทรลเลอร์ MCS-51 แล้วนำค่าที่ได้ไปคำนวณแล้วนำไปแสดงผล

เมื่อมีการเคลื่อนที่ของรถสายไมล์รถจะหมุนทำให้ล้อหมุนซึ่งเป็นตัวทำให้เซนเซอร์ OPTO ตรวจจับการเคลื่อนที่ของรถ โดยการหมุนของล้อรถ 1 รอบสายไมล์จะหมุนไป 2.5 รอบ ซึ่งข้อมูลที่ออกมาจะเป็นที่มีค่าเป็นลอจิก 0 และลอจิก 1 คือ เมื่อ OPTO เจอช่องที่ล้อจะให้ข้อมูลออกมาเป็นลอจิก 1 แต่ถ้าเป็นส่วนอื่นจะให้ข้อมูลออกมาเป็นลอจิก 0 ซึ่งข้อมูลที่ได้นั้นเราจะใช้อินเตอร์รัปต์ของไมโครคอนโทรลเลอร์ MCS-51 ตรวจนับพัลส์ขอบขาลงแล้วนำข้อมูลที่ได้ไปคำนวณใน MCS-51 แล้วส่งแสดงผลบนหน้าจอ LCD



รูปที่3.9 แสดงการใช้ OPTO COUPLER ตรวจจับที่จานหมุน



รูปที่3.10 แสดงการใช้ OPTO COUPLING วัดความเร็วรอบเพื่อนำค่าที่ได้ไปคำนวณความเร็ว รถจักรยานยนต์โดยการติดจานหมุนไว้ที่ปลายสายไมล์ของรถจักรยานยนต์

3.3 ส่วนวงจรการตรวจจับความเร็วรอบเครื่องยนต์

ในด้านการการวัดความเร็วรอบของเครื่องยนต์จะเป็นการตรวจจับสัญญาณจาก CDI ของรถจักรยานยนต์แล้วนำไปเข้าวงจร Monostable เพื่อปรับสัญญาณให้ไมโครคอนโทรลเลอร์ MCS-51 สามารถตรวจจับได้ ซึ่งเราจะใช้อินเตอร์รัปต์ของไมโครคอนโทรลเลอร์ MCS-51 ตรวจนับพัลส์ขอบขาลงแล้วนำข้อมูลที่ได้ไปคำนวณใน MCS-51 แล้วส่งไปแสดงผลบนหน้าจอ LCD

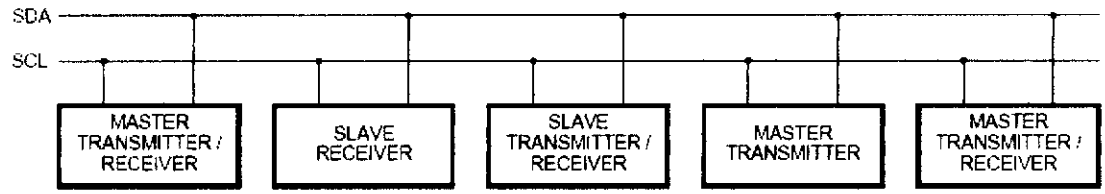
Analogue to Digital Converter

การแปลงสัญญาณอนาลอกเป็นดิจิตอลโดยใช้ IC เบอร์ 3201 ซึ่งเป็นการแปลงสัญญาณอนาลอกเป็นดิจิตอลแบบซัคเซสซีฟแอปพรีอกซิเมชั่น หรือ เป็นการแปลงแบบประมาณค่าใกล้เคียงโดย ใช้การเชื่อมต่อกับไมโครคอนโทรลเลอร์ MCS-51 แบบบัส I^2C ซึ่งข้อมูลที่ได้นั้นเป็นข้อมูลที่ได้จากการแปลง voltage ของ input ซึ่งเปลี่ยนแปลงแบบเป็นเชิงเส้นกับ output

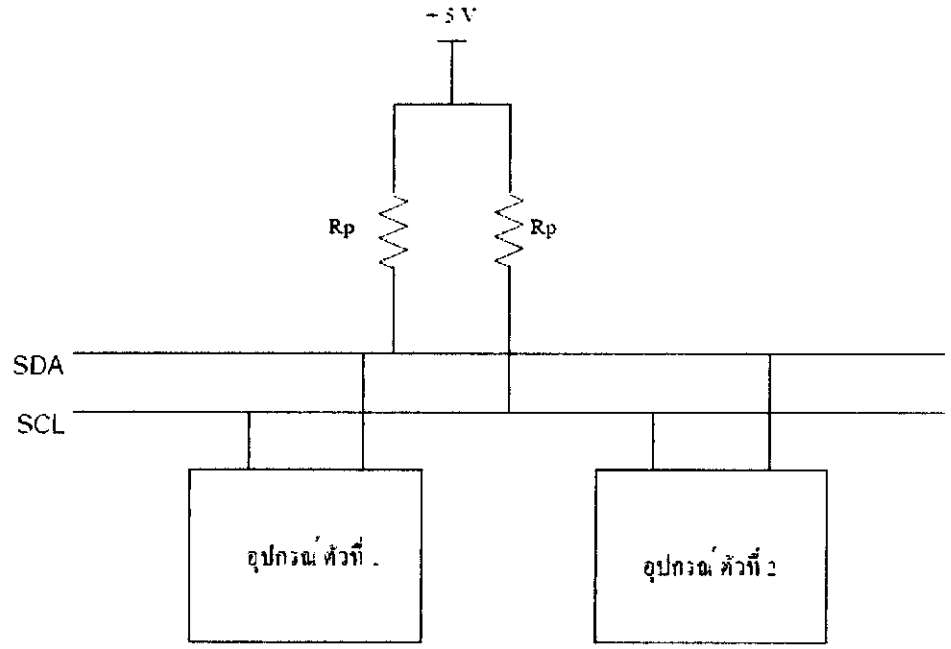
A/D Converter

ความรู้เบื้องต้นเกี่ยวกับ I^2C

I^2C ย่อมาจาก Inter-IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซี โดยบัส I^2C ได้รับการพัฒนาขึ้นโดยฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติต่อ สั่งงาน และควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นหนึ่งคือ สายข้อมูลอีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน การติดต่อร่วมกันของอุปกรณ์บนบัส I^2C ทำได้ง่ายมาก เพียงแต่ต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัว ขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว



รูปที่ 3.11 ผังแสดงการเชื่อมต่อของอุปกรณ์ต่างๆ บนระบบบัส I^2C



รูปที่ 3.12 แสดงวงจรเอาต์พุตของอุปกรณ์ในระบบบัส I^2C

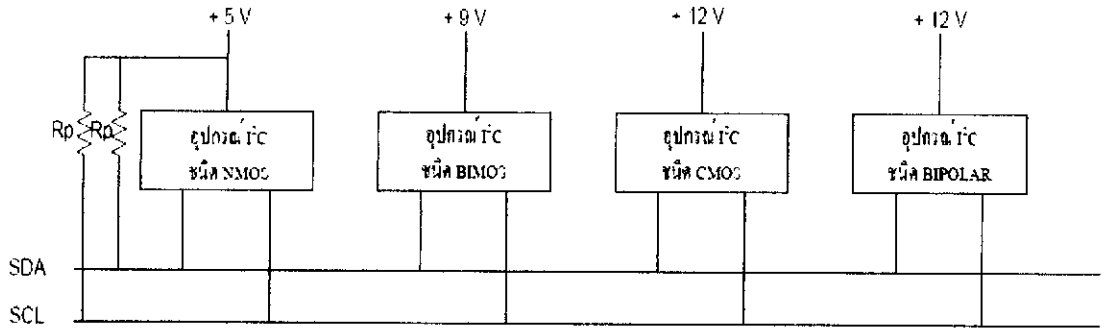
สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock line) ในการอธิบายต่อไปนี้จะเรียกสายสัญญาณทั้งสองว่า สาย SDL และ SCL

ในรูปที่ P18-1 แสดงผังของการเชื่อมต่ออุปกรณ์ต่างๆ บนบัส I^2C จะเห็นได้ว่า อุปกรณ์ที่ทำการเชื่อมต่อบนบัส I^2C มีหลากหลาย ไม่ว่าจะเป็นไอซีพอร์ตอินพุตเอาต์พุต (I/O Expander) , ไอซีแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC) และแปลงสัญญาณดิจิตอลเป็นอะนาลอก (DAC) , ไอซีรีไทม์คล็อก (RTC) , ไอซีขับโมดูล LCD ,หน่วยความจำอีอีพรอม และไมโครคอนโทรลเลอร์

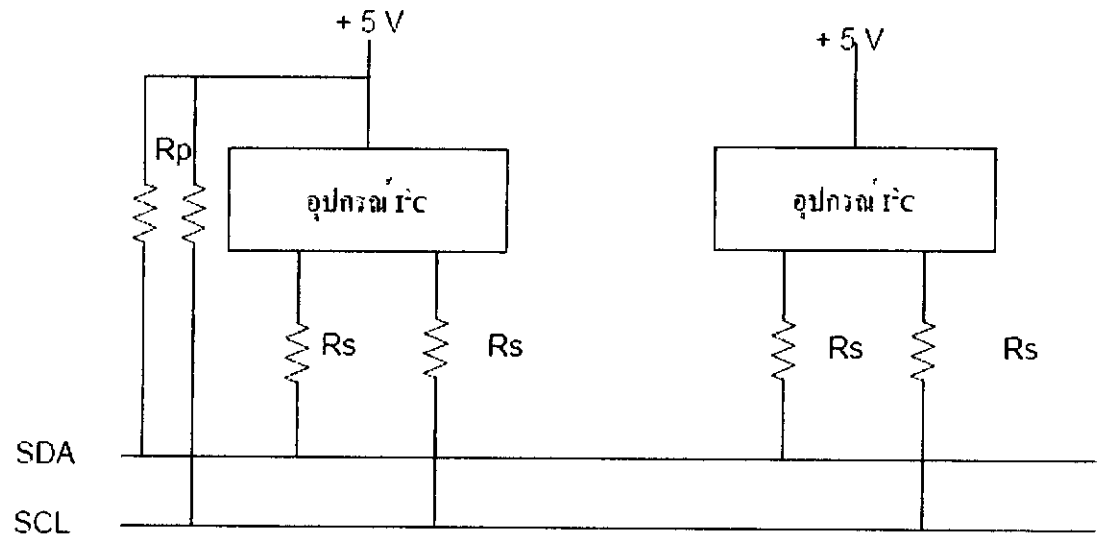
คุณสมบัติโดยทั่วไปของบัส I^2C

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพูลอัปกับแรงดัน +5V ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาต์พุตของอุปกรณ์ที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (open-drain) หรือคอลเล็กเตอร์เปิด (open-collector) ดังรายละเอียดในรูปที่ 3.13

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาที ในโหมดปกติ (standard mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (fast mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDL และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่าคือ 7 บิต (7-bit addressing) หรือ 10บิต (10-bit addressing)



รูปที่3.13 การต่อตัวต้านทานพูลอัปบนสายสัญญาณในระบบบัส I^2C



รูปที่ 3.14 การต่อตัวต้านทาน Rs เพื่อลดสัญญาณรบกวนขนาดใหญ่ที่อาจเข้ามาในบัส I^2C

ข้อเด่นอีกประการหนึ่งของบัส I^2C คือสามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้สามารถติดต่อสื่อสารกันได้ โดยอุปกรณ์บนบัส I^2C ตัวหนึ่งอาจใช้ไฟเลี้ยง +5v ในขณะที่ตัวหนึ่งใช้ไฟเลี้ยง +12v การต่อร่วมกันบนบัส I I^2C สามารถกระทำได้ในลักษณะเดียวกับกรณีที่อุปกรณ์ทั้งสองใช้ไฟเลี้ยงเท่ากัน กล่าวคือ ให้ต่อสาย SDA และ SCL ของอุปกรณ์แต่ละตัวเข้าด้วยกัน และต้องต่อตัวต้านทานพูลอัป (Rp) เข้ากับแรงดัน +5v ไว้ด้วยเสมอ ดังแสดงในรูปที่ 3.13

ในกรณีที่อาจมีแรงดันกระชากขนาดใหญ่ปะปนเข้ามาในบัส I^2C ที่ขา SDA และ SCL ของอุปกรณ์แต่ละตัวต้องต่อตัวต้านทานอนุกรมกับขา SDA และ SCL เรียกว่า Rs ก่อนต่อเข้าสู่บัส I^2C ดังรูปที่ 3.14

หลักการของบัส I^2C

บัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้น ดังที่ได้กล่าวมาแล้วคือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดั้งนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส หรือเรียกว่า โปรโตคอล (protocol) เพื่อให้ผู้ใช้งานทราบว่า ขณะนี้อุปกรณ์ใดติดต่อกันอยู่ และอุปกรณ์ตัวใดเป็นตัวรับหรือตัวส่ง ต่อไปนี้จะขออธิบายลักษณะ หน้าที่ และนิยามของอุปกรณ์ที่ต่ออยู่บนบัส I^2C เพื่อเป็นข้อตกลงพื้นฐานก่อนที่จะอธิบายการทำงานของบัส I^2C ต่อไป

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง (Transmitter)

อุปกรณ์ที่เป็นผู้รับข้อมูล เรียกว่า ตัวรับ (Receiver) อุปกรณ์บนบัส I^2C สามารถเป็นได้ทั้งตัวรับและตัวส่ง บางอุปกรณ์มีหน้าที่เป็นตัวรับอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I^2C ที่ทำหน้าที่เป็นตัวส่งเพียงตัวเดียว

อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I^2C เรียกว่า มาสเตอร์ (master)

อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่พ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ

(1)การถ่ายทอดข้อมูลขณะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น

(2)ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะที่เป็นลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

สภาวะที่เกิดขึ้นบนบัส I^2C

มีด้วยกัน 5 สภาวะดังนี้

(1) **บัสว่าง** (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้งคู่ นั่นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้

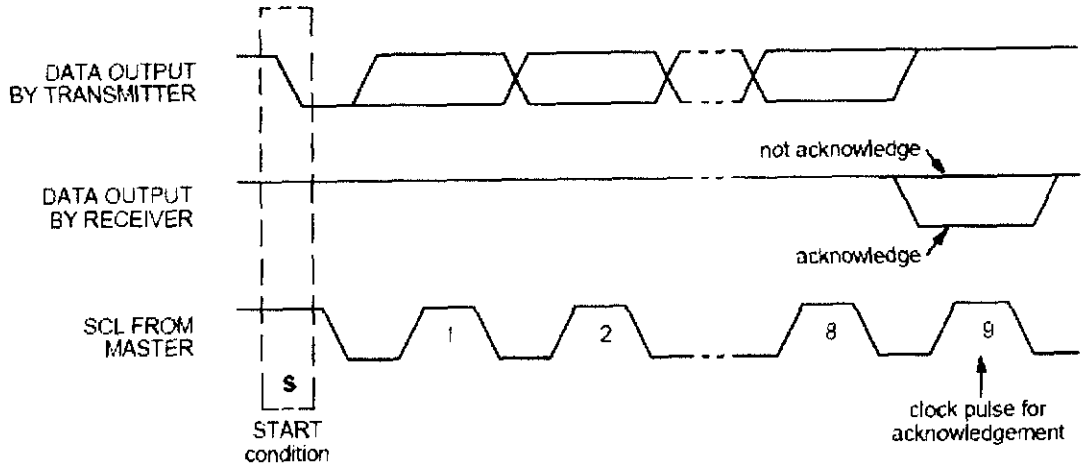
(2) **เริ่มต้นการถ่ายทอดข้อมูล**(Start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า **สภาวะเริ่มต้น** (start)

(3) **หยุดการถ่ายทอดข้อมูล** (Stop data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูง ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า **สภาวะหยุด** (stop)

(4) **ข้อมูลที่ดำรงอยู่บนบัส** (Data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดบนสายSDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่า เป็น "0"หรือ "1"
ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดได้

(5) **รับรู้ข้อมูล**(acknowledge)เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะส่งข้อมูลมา 1 บิตเรียกว่า **บิตรับรู้** (acknowledge bit) มีสถานะเป็นลอจิกสูง หลังจากส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกาเพื่อตอบสนองบิตรับรู้ที่ส่งมาจากตัวส่ง ทางด้านตัวรับส่งบิตรับรู้ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้เพื่อตอบสนองได้ทราบว่าได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว

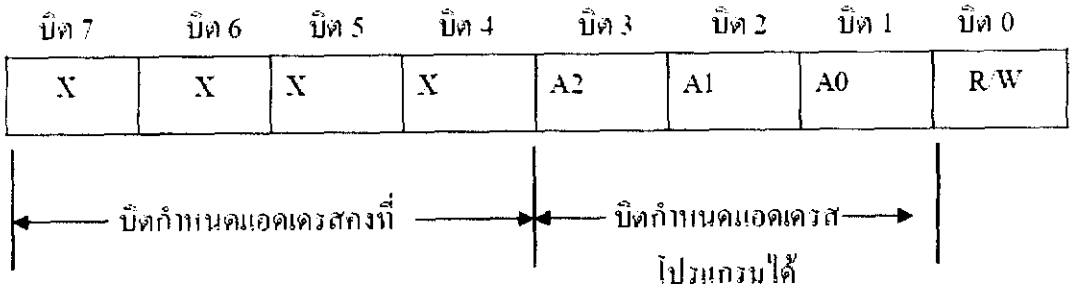
ในรูปที่ 3.15 เป็นไดอะแกรมที่แสดงถึงการเกิดสภาวะต่างๆ บนบัส I^2C ไม่ว่าจะเป็นสภาวะบัสว่าง, เริ่มต้น, ถ่ายทอดข้อมูล, รับรู้, และหยุดการถ่ายทอดข้อมูล



รูปที่ 3.15 ไดอะแกรมเวลาแสดงสถานะต่างๆ ในบัส I^2C

การทำงานบนบัส I^2C ก่อนที่จะเริ่มต้นการถ่ายทอดข้อมูลระหว่างอุปกรณ์ต่างๆ ที่ต่ออยู่บนบัส ต้องมีการอ้างถึงเสียก่อน โดยอ้างถึงอุปกรณ์บนบัส I^2C นั้นจะใช้การอ้างถึงแบบ 7 บิตหรือ 10บิต ในกรณีที่มีอุปกรณ์ต่ออยู่บนบัสไม่มากใช้การอ้างถึงแบบ 7 บิตก็เพียงพอ แต่ถ้ามีอุปกรณ์ที่ต่ออยู่บนบัสมากกว่า 127 แอดเดรสจำเป็นต้องใช้การอ้างถึงแบบ 10 บิต หลังจากที่ติดต่ออุปกรณ์แต่ละตัวได้เรียบร้อยแล้ว ก็จะเริ่มต้นการถ่ายทอดข้อมูลกันต่อไป

ดังนั้นหัวใจสำคัญในอันดับแรกของการทำงานบนบัส I^2C คือการอ้างถึงอุปกรณ์แต่ละตัว ซึ่งในที่นี้จะอธิบายรายละเอียดการอ้างถึงทั้ง 2 รูปแบบ



รูปที่ 3.16 รูปแบบของข้อมูลกำหนดแอดเดรสที่ใช้ในการอ้างถึงแบบ 7 บิต

การอ้างถึงแบบ 7 บิต (7-bit addressing)

ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้นคือ ข้อมูลที่ใช้ในการอ้างถึงอุปกรณ์ที่ต้องการติดต่อ หรือ ข้อมมูลกำหนดแอดเดรส โดยมีรูปแบบแสดงในรูปที่ 3.16 ใน 7 บิตบนรวมทั้งบิต MSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดนแบ่งเป็น บิตกำหนดแอดเดรสคงที่ (fixed address bit)จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิต ไม่สามารถเปลี่ยนแปลงแก้ไขได้ ถัดมาอีก 3 บิตเป็นบิตกำหนดแอดเดรสที่สามารถโปรแกรมได้(programmable address bit)โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0-A2ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I^2C ส่วนในบิต LSBเป็นบิตที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอปุกรณ์สเลฟตัวนั้น หากบิตLSB เป็น "0" หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ

ข้อมูลในไบต์ต่อมาคือ **ข้อมูลควบคุม** (Control byte) ในอุปกรณ์แต่ละตัวมีการกำหนดข้อมูลควบคุมที่แตกต่างกันไป ยกตัวอย่าง ไอซีขยายพอร์ตมีข้อมูลควบคุมที่ใช้กำหนดว่าบิตใดเป็นอินพุต บิตใดเป็นเอาต์พุต ในขณะที่ไอซี ADC/DAC ต้องการข้อมูลเพื่อกำหนดควบคุมเพื่อกำหนดให้ทำงานเป็นวงจร ADC หรือ DAC เป็นต้น

ข้อมูลไบต์ต่อมาคือ ข้อมูลที่ทำการถ่ายทอดจริง (Data)

หลังจากที่มีการถ่ายทอดข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง เพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้ ในรูปที่ 3.16 รูปแบบของข้อมูลอนุกรมที่ใช้ติดต่อกับอุปกรณ์บัส I^2C เมื่อใช้การอ้างถึงแบบ 7 บิต

การอ้างถึงแบบ 10บิต

ในการอ้างถึงแบบนี้ ยังคงใช้รูปแบบข้อมูลอนุกรมที่เหมือนกับแบบ 7 บิต หากแต่จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อย โดยในข้อมูลไบต์แรกหลังจากเกิดสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของของอุปกรณ์ที่ต้องการติดต่อ ในบิต LSB ของข้อมูลไบต์แรกยังเป็นการกำหนดว่าต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบต์ต่อมาเป็นข้อมูลแอดเดรสในไบต์ที่ 2 ของอุปกรณ์ที่ต้องการติดต่อด้วย ข้อมูลไบต์ถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนั้นก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อ

เช่นเดียวกับการอ้างถึงแบบ 7 บิต หลังจากการถ่ายทอดข้อมูลครบทุกไบต์ ต้องมีสภาวะรับรู้เกิดขึ้น เพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้

การต่ออุปกรณ์ระบบบัส I^2C กับไมโครคอนโทรลเลอร์ MCS-51

สามารถทำได้ง่ายมาก เพียงใช้ขาพอร์ต 2 ขา โดยกำหนดให้ขาหนึ่งเป็น SDA อีกขาหนึ่งเป็น SCL และต่อตัวต้านทานค่าประมาณ 4.7 k พูลอัปที่ขาพอร์ตทั้ง 2 ขา เพียงเท่านี้ก็สามารถติดต่อกับอุปกรณ์ระบบบัส I^2C ได้แล้ว

การเขียนโปรแกรมติดต่อบัส I^2C

เริ่มต้นด้วยการสร้างสถาวะมาตรฐานของบัส I^2C อันประกอบด้วย สภาวะเริ่มต้น, สภาวะสิ้นสุดการส่งข้อมูล, สภาวะหยุด, สัญญาณนาฬิกาบนขา SCL, การเขียนและอ่านข้อมูลกับอุปกรณ์บนระบบบัส I^2C

การสร้างสภาวะเริ่มต้น

1. เมื่อต้องการติดต่อกับบัส I^2C ที่แรกที่ต้องทำสำหรับไมโครคอนโทรลเลอร์ซึ่งถือว่าเป็นอุปกรณ์มาสเตอร์คือ การทำให้บัสว่างด้วยการกำหนดให้ขา SCL และ SDA มีลอจิกเป็น "1" ทั้งคู่
2. จากนั้นทำให้ขา SDA มีลอจิกเป็น "0"โดยที่ขา SCL ยังคงเป็นลอจิก "1"อยู่
3. กำหนดให้ขา SCL มีลอจิกเป็น "0"ถึงตอนนี้ทั้ง SCL และ SDA จะมีลอจิกเป็น "0" ทั้งคู่ พร้อมที่จะติดต่อได้แล้ว

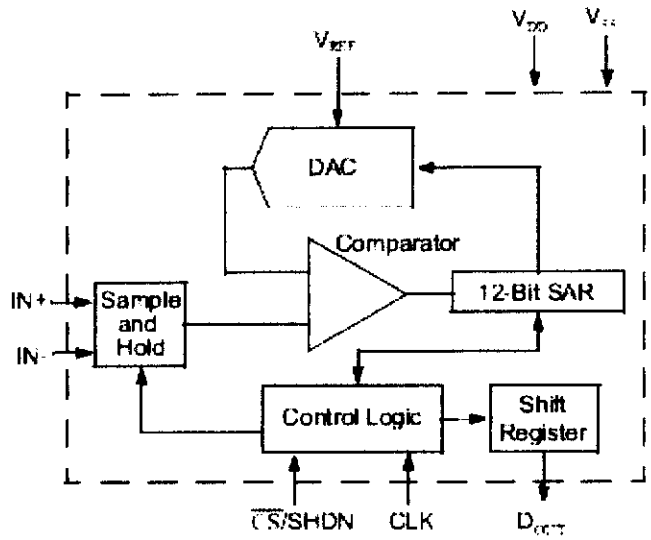
การสร้างสภาวะหยุด

1. เมื่อต้องการหยุดส่งข้อมูลจะต้องส่งสภาวะหยุดออกไป โดยในตอนแรกต้องกำหนดให้ขา SCL และ SDA เป็นลอจิก "0" ทั้งคู่ก่อน
2. กำหนดให้ขา SCL มีลอจิกเป็น "1" โดย SDA ยังคงมีลอจิกเป็น "0"
3. จากนั้นทำให้ขา SDA มีลอจิกเป็น "1" ซึ่งจะทำให้ระบบบัสกลับเข้าสู่บัสว่างอีกครั้ง พร้อมที่จะรับหรือส่งข้อมูลต่อไป

การแปลงสัญญาณอะนาล็อกดิจิตอลแบบซักเซสซีพแอปพร็อกซิเมชั่น (SUCCESSIVE APPROXIMATION)

การแปลงสัญญาณอะนาล็อกเป็นดิจิตอล(ADC) ที่ได้รับความนิยมสูงสุดคือการแปลงแบบซักเซสซีพแอปพร็อกซิเมชั่น ไอซีที่นำมาใช้ในการใช้งานนี้คือ ไอซี ADC เบอร์ MCP3201 ซึ่งมีความละเอียดของข้อมูลถึง 12 บิต ดังนั้นจึงต้องมาทำความเข้าใจการทำงานของ ไอซีเบอร์นี้ก่อน

โดยทั่วไปการแปลงแบบซักเซสซีพแอปพร็อกซิเมชั่น อาจเรียกได้ว่า เป็นการแปลงแบบค่าประมาณใกล้เคียง ส่วนสำคัญหลักคือวงจรเปรียบเทียบแรงดัน วงจรสัญญาณนาฬิกา ส่วนควบคุมลอจิก



รูปที่ 3.17 แสดงวงจรหลักของการแปลงแบบซักเซสซีพแอปพร็อกซิเมชั่น

ความเที่ยงตรงของวงจร ADC

เป็นการเปรียบเทียบแรงดันอะนาล็อกกับแรงดันที่น่าจะเกิดขึ้นจริง ยกตัวอย่างที่ข้อมูลดิจิตอลสุงสุดที่ควรเป็นของวงจร ADC ขนาด 12 บิตเมื่อเทียบกับแรงดันอะนาล็อกควรมีค่าเท่ากับ 5 โวลท์ แต่จะค่าของความผิดพลาดบอกมาด้วยนั่นคือบอกค่ามาเป็นค่าของ LSB นั่นเอง ซึ่งค่าความเที่ยงตรงของวงจรนี้คือที่ ±2 LSB

ค่าเวลาในการแปลงสัญญาณ (CONVERTION TIME)

เป็นค่าของเวลาทั้งหมดที่วงจร ADC แบบวงจรนับแรมป์และแบบซักเซสซีพแอปพร็อกซิเมชั่น ใช้ในการแปลงสัญญาณอะนาล็อกเป็นดิจิตอลจนเสร็จสิ้น พารามิเตอร์ตัวนี้มักจะปรากฏในคุณสมบัติของไอซีที่ทำงานเป็นวงจร ADC เมื่อไอซีแปลงสัญญาณจนเสร็จสิ้นลง จะส่งสัญญาณที่เรียกว่า EOC (End of Conversion)

ค่าเวลาในการแปลงสัญญาณ ของวงจร ADC จะขึ้นอยู่กับจำนวนบิตของวงจร ค่าความถี่ของสัญญาณนาฬิกาที่ใช้ในการแปลงและขนาดของสัญญาณอะนาล็อกอินพุต

ข้อมูลเบื้องต้นของ ไอซี ADC MCP3201

ในตัวไอซีเบอร์ MCP3201 มีวงจรแบบซักเซสซีพแอปพรีอกซิเมชั่น ขนาด 8 บิต ระบบการเชื่อมต่อเป็นแบบบัส I^2C ทำให้สายสัญญาณเพียงสองเส้น สามารถนำไปประยุกต์งานได้อย่างกว้างขวาง ซึ่งมีรายละเอียดทางเทคนิคดังนี้

- ทำงานโดยใช้แหล่งจ่ายไฟเพียงชุดเดียว
- ทำงานที่แรงดัน 2.7 – 5.5 โวลท์
- มีวงจรแซมเปิลแอนด์โฮลอยู่ภายใน
- มีอัตราการแซมปลิ้ง 100 KSPS ที่แรงดัน V_{DD} = 5 โวลท์
- มีอัตราการแซมปลิ้ง 50 KSPS ที่แรงดัน V_{DD} = 2.7 โวลท์
- กระแสขณะสแตนด์บายสูงสุด 2 ไมโครแอมป์
- กระแสใช้งานสูงสุดที่ 400 ไมโครแอมป์
- การอ่านค่าสามารถเลื่อนบิตได้โดยอัตโนมัติ
- สัญญาณอนาล็อกมีระดับตั้งแต่ V_{SS} - V_{DD}

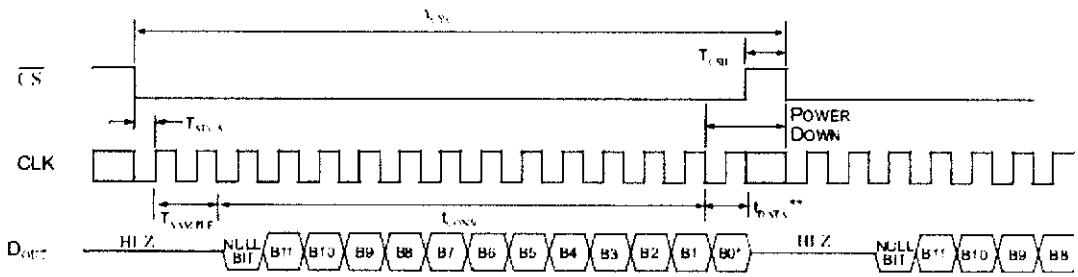
รายละเอียดฟังก์ชันต่างๆ ของ MCP3201

ตำแหน่งแอดเดรส

ในระบบบัส I2C การติดต่อกับอุปกรีแต่ละตัวต้องระบุแอดเดรสของอุปกรณ์เหล่านั้น เช่น ถ้าเป็นการอ้างถึงแบบ 7 บิต แอดเดรส 4 บิตบนจะเป็นตำแหน่งของแอดเดรสเฉพาะของอุปกรณ์ตัวนั้นๆ มาจากผู้ผลิต เป็นต้น

การอ่านค่าข้อมูลของ MCP3201

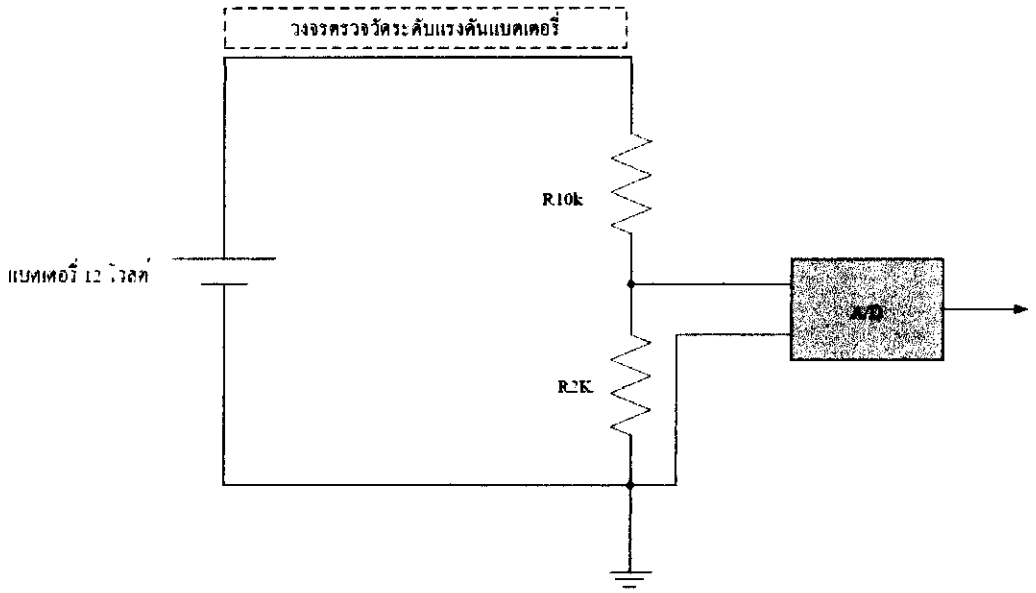
1. เตรียมข้อมูลการกำหนดแอดเดรส
2. เรียกโปรแกรมย่อยการติดต่อแบบ I2C
3. ส่งข้อมูลควบคุมไปยังอุปกรณ์นั้นๆ
4. ส่งสัญญาณ stop
5. เรียกโปรแกรมย่อยการติดต่อแบบ I2C อีกครั้ง
6. อ่านค่าจากตัวอุปกรณ์ออกมา



รูปที่ 3.18 แสดงการติดต่อใช้งาน MCP3201

3.4 วงจรวัดระดับแรงดันแบตเตอรี่

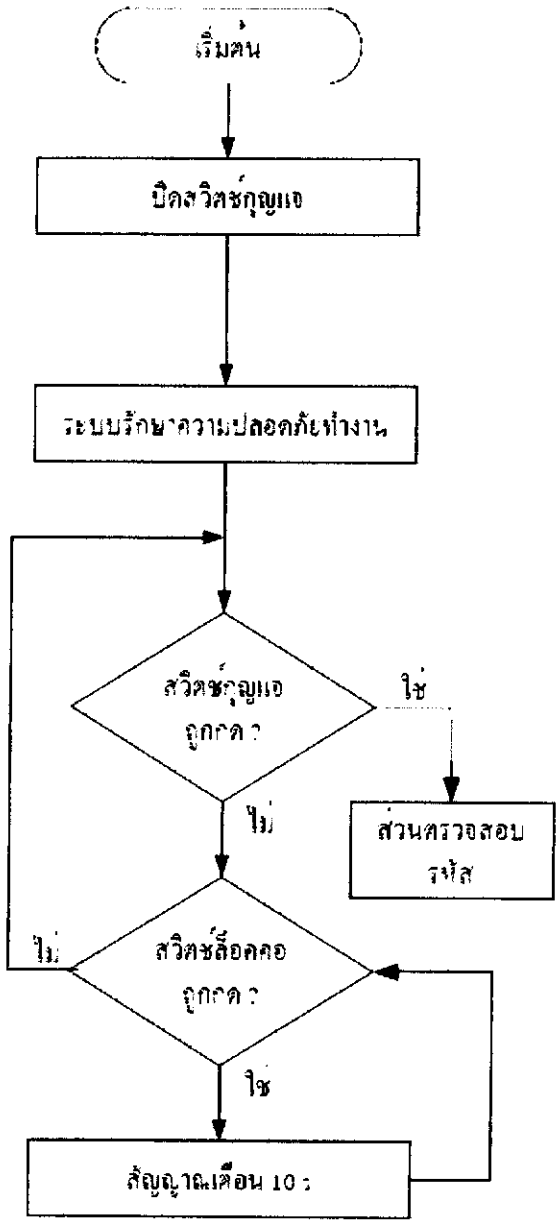
การวัดเบตเตอรี่โดยการนำค่าแรงดันจากแบตเตอรี่(12 V) มาทำการแบ่งแรงดัน (voltage divider) ให้มีค่ามากที่สุดเท่ากับ 5 V แล้วต่ำสุดเท่ากับ 0 V แล้วผ่านเข้าสู่ส่วนแปลงสัญญาณจากอนาลอกเป็นดิจิตอล เพื่อนำไปคำนวณค่าแรงดันแล้วแสดงผลซึ่งมีลักษณะของวงจรดังรูป



รูปที่ 3.19 แสดงวงจรตรวจวัดระดับแรงดันแบตเตอรี่

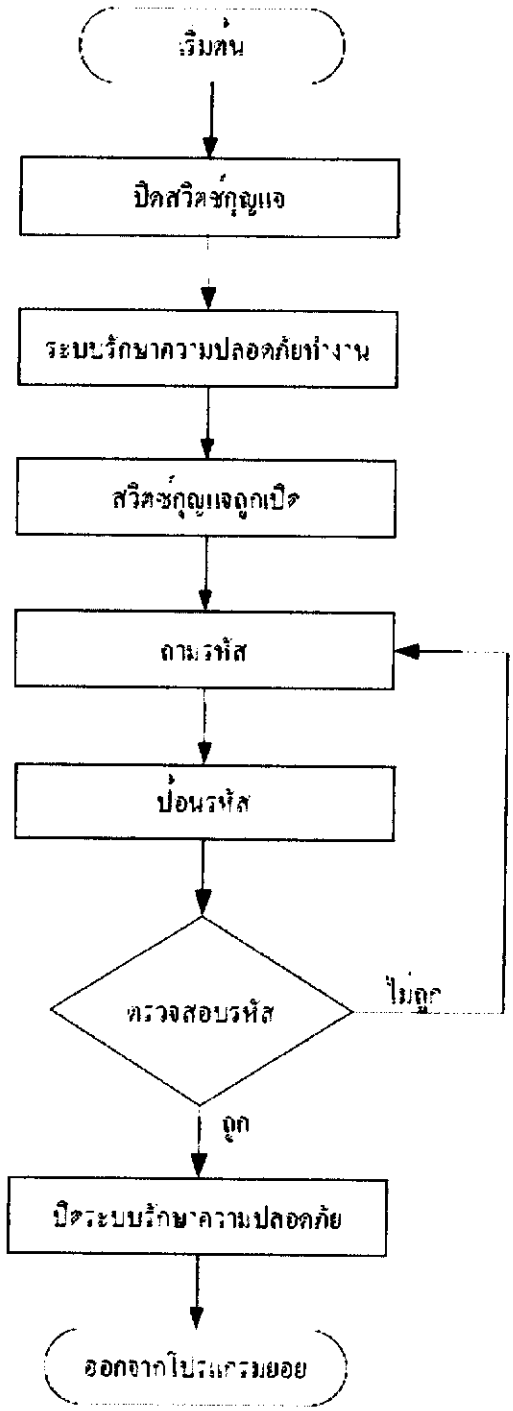
3.5 ระบบรักษาความปลอดภัย (กันขโมย)

ระบบรักษาความปลอดภัยจะเริ่มทำงานโดยอัตโนมัติทุกครั้งที่มีการปิดสวิตช์กุญแจรถจักรยานยนต์ ซึ่งระบบจะทำการตัดระบบ start ของรถจักรยานยนต์ แล้วคอยตรวจจับการเปิดสวิตช์ 2 ตำแหน่ง คือ สวิตช์กุญแจ และ สวิตช์ที่ล็อคคอรถจักรยานยนต์ เมื่อมีการเปิดสวิตช์ตัวใดตัวหนึ่งระบบสัญญาณเตือนจะทำงาน 10 วินาที แล้วจะย้อนกลับไปตรวจสอบสวิตช์ 2 ตำแหน่งเดิมอีกครั้ง โฟลวชาร์ตแสดงการทำงานของส่วนตรวจตรวจสอบสวิตช์ 2 ตำแหน่งของระบบรักษาความปลอดภัยแสดงดังรูปที่ 3.20



รูปที่ 3.20 โฟลวชาร์ตแสดงการทำงานของส่วนตรวจตรวจสอบสวิตช์ 2 ตำแหน่งของระบบรักษาความปลอดภัย

เมื่อต้องการใช้งานรถจักรยานยนต์ผู้ใช้งานต้องทำการป้อนรหัสเพื่อทำการปิดระบบรักษาความปลอดภัยก่อน โดยจะต้องทำการป้อนรหัสเป็นตัวเลข 0-9 ให้ตรงกับรหัสที่ได้ทำการตั้งไว้แต่แรกโดยระบบจะนำรหัสที่ได้รับไปเปรียบเทียบกับรหัสที่ตั้งไว้ถ้ารหัสตรงกันระบบจะทำการปิดระบบรักษาความปลอดภัยและสามารถ start รถได้และจะย้อนกลับไปถามหารหัสใหม่อีก โฟลวชาร์ตแสดงการทำงานของส่วนป้อนรหัสเพื่อปิดระบบรักษาความปลอดภัยแสดงดังรูปที่ 3.21



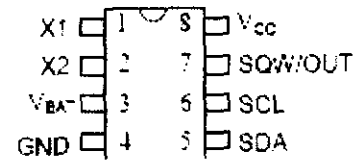
รูปที่ 3.21 โฟลวชาร์ตแสดงการทำงานของส่วนป้อนรหัสเพื่อปิดระบบรักษาความปลอดภัยระบบรักษาความปลอดภัย

3.6 การใช้งานไอซีสร้างฐานเวลาจริงหรือรีลไทม์คล็อก (RTC)

ผู้ผลิตคือ ดัลลัสเซมิคอนดักเตอร์(Dallas semiconductor) มีหน้าที่สร้างฐานเวลาจริงให้แก่ระบบไมโครคอนโทรลเลอร์ โดย DS1307 จะให้ข้อมูลเกี่ยวกับเวลาทั้งหมด ไม่ว่าจะเป็นค่าของเวลาที่ละเอียดถึงหลักวินาที, นาที, ชั่วโมง, วันที่(date), วันในสัปดาห์(day), เดือน และปี โดยสามารถปรับวันเดือนปีให้ตรงตามปฏิทินได้อย่างถูกต้อง รวมถึงการกำหนดวันในปีอธิกสุรทินด้วย คุณสมบัติทางเทคนิคที่สำคัญมีดังนี้

- เป็นไอซีรีลไทม์คล็อกให้ข้อมูลตั้งแต่วินาทีถึงปี รวมถึงการปรับวันในปีอธิกสุรทินด้วย สามารถให้ข้อมูลเวลาได้อย่างเที่ยงตรงถึงปีคริสตศักราช 2100
- มีหน่วยความจำนอนโวลาไทล์แรม 56 ไบต์อยู่ภายใน สามารถใช้เก็บข้อมูลทั่วไปได้
- ใช้การเชื่อมต่อแบบระบบบัส I^2C
- มีวงจรตรวจจับไฟเลี้ยงต่ำหรือหายไปอย่างอัตโนมัติ และสามารถรักษาข้อมูลเวลาไว้ได้แม้ไม่มีไฟเลี้ยงไอซี

รายละเอียดขาต่อใช้งานของ DS1307



DS1307 8-Pin DIP (300-mil)

รูปที่ 3.22 การจัดขาของไอซี DS1307 ไอซีสร้างฐานเวลาจริง (RTC)

ในรูปที่ 3.22 แสดงการจัดขาของ DS1307 แต่ละขามีหน้าที่และการใช้งานดังนี้

V_{CC}, GND (ขา 8, 4) ต่อกับไฟเลี้ยง +5V

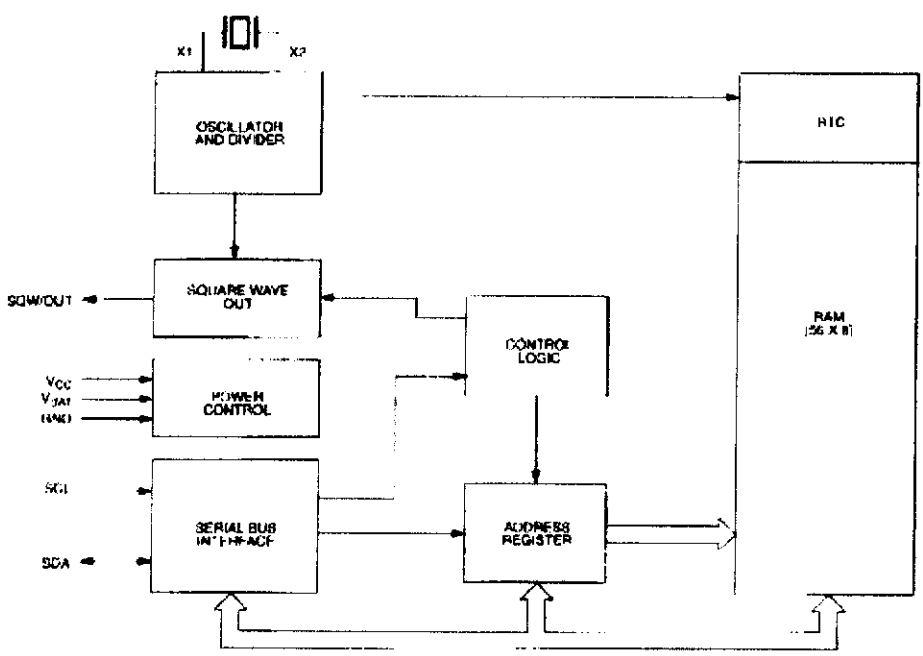
V_{BAT} (ขา 3) ใช้ต่อกับแบตเตอรี่ 3V เพื่อรักษาการทำงานของวงจรสร้างฐานเวลาของ DS1307 ให้คงอยู่ต่อไป แม้ว่าไม่มีไฟเลี้ยงจ่ายให้แก่ DS1307 ชนิดของแบตเตอรี่ที่เหมาะสมคือแบตเตอรี่แบบลิเธียม ซึ่งมีความจุ 40 mAhr หรือมากกว่า จะสามารถรักษาข้อมูลได้นาน 10 ปี ที่อุณหภูมิ 25 องศาเซลเซียส

SDA, SCL (ขา 5 และ 6) เป็นขาสำหรับเชื่อมต่อกับไมโครคอนโทรลเลอร์บนระบบบัส I^2C

SQW OUT (ขา 7) ที่ขานี้จะมีสัญญาณรูปสี่เหลี่ยมส่งออกมา โดยสามารถเลือกความถี่ได้ 1 Hz, 4.096kHz, 8.192kHz และ 32kHz ในการใช้ลานต้องต่อตัวต้านทาน 1k พูลอัพที่ขานี้ด้วย

X1, X2 (ขา 1 และ 2) ใช้ต่อกับคริสตอลความถี่มาตรฐาน 32.768kHz เพื่อใช้เป็นฐานเวลาในการสร้างค่าความจริง ในการใช้งานต้องต่อคริสตอลเข้ากับขาทั้งสองนี้และที่แต่ละขาต้องต่อตัวเก็บประจุค่าต่ำๆ ประมาณ 15pF ครอบกับขากราวด์ด้วย

การทำงานของ DS1307



รูปที่ 3.23 โครงสร้างภายในของไอซี DS1307

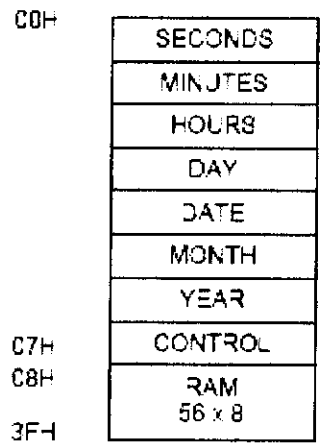
ไอซี DS1307 จัดการเชื่อมต่อในแบบบัส I^2C โดยจะทำงานเป็นอุปกรณ์สเลฟเสมอ ดังนั้นการติดต่อเพื่อใช้งานจึงต้องกำหนดรูปแบบตามที่กำหนดไว้ในการติดต่อแบบ I^2C ในรูปที่ 3.23 แสดงส่วนประกอบหลักที่สำคัญและไดอะแกรมการทำงานของ DS1307 วงจรออสซิลเลเตอร์ถือเป็นหัวใจหลักของไอซี เนื่องจากเป็นจุดเริ่มต้นของการสร้างข้อมูลเวลาจริง ในขณะที่ DS1307 ทำงานที่ขา SQW/OUT จะมรสัญญาณพัลส์สี่เหลี่ยมส่งออกมาตลอดเวลาในกรณีที่มีการอินาเบิลวงจรกำเนิดสัญญาณพัลส์ที่รีจิสเตอร์ควบคุม ค่าความถี่ของสัญญาณนี้สามารถเลือกได้ 4 ค่า คือ 1 Hz, 4.096kHz, 8.192kHz และ 32kHz พร้อมกันนั้นก็จมีการเก็บค่าของเวลาไว้ในหน่วยความจำคอลโวลาไทล์แรม ซึ่งมีขนาดรวม 64 ไบต์ และเป็นหน่วยความจำสำหรับเก็บข้อมูลทั่วไปสำหรับผู้ใช้งานอีก 56 ไบต์

วงจรควบคุมพลังงานไฟฟ้าจะคอยตรวจสอบสภาวะของไฟเลี้ยงไอซี หากไฟเลี้ยงต่ำกว่า $1.25xV_{BAT}$ ก็จะควบคุมให้ DS1307 หยุดการทำงานรีเซตค่าตัวนับแอดเดรสภายในทำให้ไม่สมารถติดต่อกับ DS1307 ได้ ดังนั้นในการใช้งาน DS1307 ต้องระมัดระวังอย่าให้ไฟเลี้ยงตกต่ำกว่า $1.25xV_{BAT}$ หรือประมาณ 3.75V ในกรณีที่ใช้ V_{BAT} เท่ากับ 3V ถ้าหากไฟเลี้ยงมีค่าต่ำกว่า V_{BAT} ไอซี DS1307 จะเข้าสู่โหมดสำรองข้อมูลกระแสต่ำทันที จะไม่มีการส่งสัญญาณพัลส์ออกมาที่ขา SQW/OUT แต่วงจรสร้างฐานเวลายังคงทำงานเพื่อให้ค่าของเวลาเดินไปอย่างไม่ผิดพลาด เมื่อมีไฟเลี้ยงปรากฏขึ้นอีกครั้ง DS1307 ก็จะสามารถให้ค่าของเวลาที่เป็นจริงแก่ผู้ใช้งานได้ต่อไป

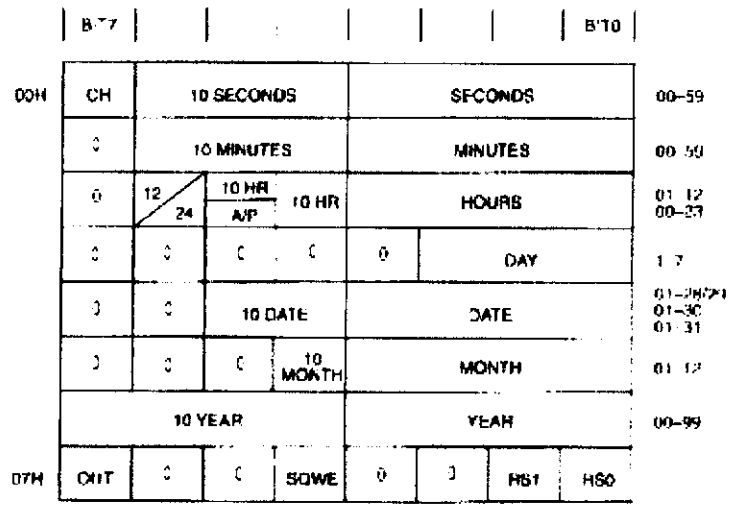
วงจรสื่อสารอนุกรมภายใน DS1307 ได้รับการกำหนดให้ทำงานตามรูปแบบของบัส I^2C เป็นช่องทางการสื่อสารระหว่าง DS1307 กับอุปกรณ์มาสเตอร์ ผู้ใช้งานสามารถเข้าถึงหน่วยความจำที่ใช้เก็บค่าเวลาและหน่วยความจำใช้งานทั่วไปได้โดยการเขียนข้อมูลตามรูปแบบที่กำหนดในระบบบัส I^2C

การจัดสรรหน่วยความจำใน DS1307

ในรูปที่ 3.24 แสดงการจัดสรรพื้นที่ของหน่วยความจำภายใน DS1307 พื้นที่ 7 ไบต์แรกตั้งแต่แอดเดรส 00H-06H เป็นพื้นที่ของรีจิสเตอร์ค่าเวลาใช้ในการเก็บข้อมูลเกี่ยวกับเวลา ไบต์ต่อมาที่แอดเดรส 07H เป็นพื้นที่ของรีจิสเตอร์ควบคุมการทำงานของ DS1307 ในรูปที่ 3.25 แสดงรายละเอียดของรีจิสเตอร์ควบคุมของ DS1307



รูปที่ 3.24 การจัดการหน่วยความจำแรมภายใน DS1307



รูปที่ 3.25 รายละเอียดของรีจิสเตอร์เก็บค่าเวลาและรีจิสเตอร์ควบคุมของ DS1307

ด้วยการจัดสรรพื้นที่แบบนี้ทำให้ผู้ใช้สามารเรียกข้อมูลเวลาออกมาได้ตามที่ต้องการ โดยไม่จำเป็นต้องอ่านออกมาทั้งหมดก็ได้ ค่าของเวลาทั้งหมดจะอยู่ในรูปของเลขฐานสิบ สำหรับการแสดงเวลาในรูปของชั่วโมงสามารถเลือกได้ว่าต้องการแบบ 12 หรือ 24 ชั่วโมง โดยกำหนดที่บิต 6 ของแอดเดรส 02H และเมื่อเลือกแบบ 12 ชั่วโมง ที่บิต 5 ในแอดเดรสเดียวกันจะใช้ในการแสดงค่า AM/PM โดยถ้าบิตนี้เป็น "1" หมายถึง ค่าชั่วโมงในขณะนี้เป็นช่วงเวลาหลังเที่ยงวัน ในกรณีที่เป็นแบบ 24 ชั่วโมง บิตนี้จะใช้ในการแสดงค่า 2 ของหลักสิบในหน่วยชั่วโมง

รีจิสเตอร์ควบคุม

มีแอดเดรสอยู่ที่ 07H มีรายละเอียดดังนี้

OUT (Output Control) : ใช้ในการควบคุมระดับลอจิกที่ขา SQW OUT ในกรณีที่ดิสเอเบิลการกำเนิดสัญญาณสี่เหลี่ยม โดยถ้าบิตนี้เป็น "1" ที่ขา SQW OUT ก็จะเป็น "1" ถ้าบิตนี้เป็น "0" ที่ขา SQW OUT ก็จะเป็น "0"

SQWE (Square Wave Enable) : ใช้ในการเอ็นเนเบิลวงจรกำเนิดสัญญาณสี่เหลี่ยมที่ขา SQW OUT ถ้าต้องการให้มีสัญญาณสี่เหลี่ยมออกให้กำหนดบิตนี้เป็น "1"

RS1, RS0 (Rate Select) : ใช้ในการเลือกความถี่ของสัญญาณสี่เหลี่ยมที่ออกจากขา SQW OUT ดังมีรายละเอียดต่อไปนี้

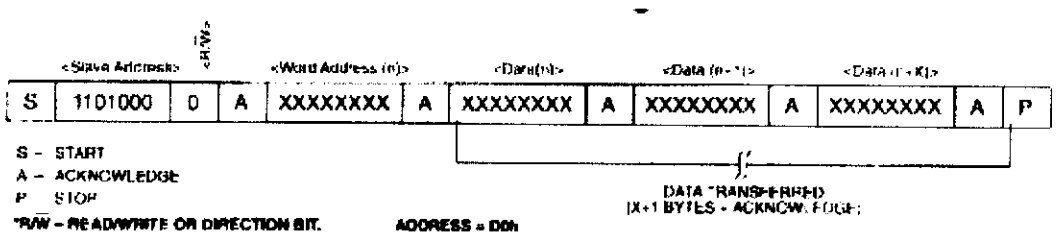
RS1	RS0	ค่าความถี่ของสัญญาณสี่เหลี่ยม
0	0	1Hz
0	1	4.096kHz
1	0	8.192kHz
1	1	32.768kHz

โหมดการทำงานของ DS1307

มีด้วยกัน 2 โหมด คือ โหมดเขียนข้อมูลและโหมดอ่านข้อมูล ในการใช้งาน DS1307 ตามปกติจะใช้งานเฉพาะโหมดอ่านข้อมูลเท่านั้น เนื่องจากไมโครคอนโทรลเลอร์จะติดต่อกับ DS1307 เพื่ออ่านข้อมูลของเวลาไปใช้งาน โหมดการเขียนข้อมูลจะถูกใช้งานก็ต่อเมื่อต้องการตั้งเวลาใหม่และต้องการเขียนข้อมูลในหน่วยความจำใช้งานทั่วไป อย่างไรก็ตามเมื่อเริ่มต้นติดต่อกับ DS1307 จำเป็นอย่างยิ่งที่จะต้องเข้าสู่โหมดการเขียนข้อมูลก่อนเพื่อกำหนดแอดเดรสเพื่อกำหนดแอดเดรสที่ต้องการอ่านข้อมูล จากนั้นจึงเปลี่ยนโหมดการทำงานมาเป็นโหมดการอ่านข้อมูล

โหมดการเขียนข้อมูล

มีรูปแบบดังรูปที่ 3.26 เริ่มต้นเมื่อไมโครคอนโทรลเลอร์ทำการกำหนดสภาวะเริ่มต้น (START :S) จากนั้นส่งข้อมูลกำหนดแอดเดรส 1101000 ตามด้วยข้อมูลเลือกการเขียน นั่นคือ ค่า 0 จากนั้นจะรอการตอบรับจาก DS1307 ขั้นตอนต่อมาคือ ส่งข้อมูลเพื่อเลือกแอดเดรสที่ต้องการเขียน จากนั้นรอการตอบรับจาก DS1307 เมื่อมีการตอบรับมาเรียบร้อย ก็เริ่มทยอยเขียนข้อมูลลงไปครั้งละแอดเดรส หลังจากเขียนข้อมูลในแต่ละแอดเดรส จะต้องหยุดรอการตอบรับจาก DS1307 ทุกครั้ง จึงจะสามารถเขียนข้อมูลต่อไปได้ เมื่อเขียนเรียบร้อยแล้วให้ส่งสภาวะหยุด (STOP :P) อันสิ้นสุดกระบวนการเขียนข้อมูล

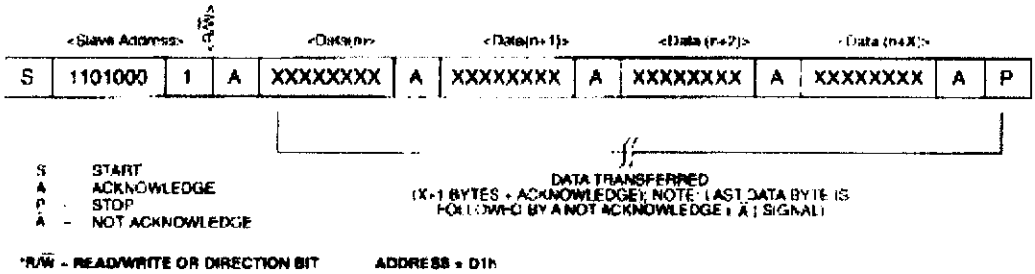


รูปที่ 3.26 รูปแบบของข้อมูลสำหรับติดต่อกับ DS1307 ในโหมดการเขียนข้อมูล

โหมดการอ่านข้อมูล

มีรูปแบบแสดงในรูปที่ 3.27 เริ่มต้นการทำงานเหมือนกับโหมดเขียนข้อมูลคือ ไมโครคอนโทรลเลอร์กำหนดสภาวะเริ่มต้นแล้วส่งข้อมูลกำหนดแอดเดรสตามด้วยข้อมูลเลือกการอ่าน ซึ่งเท่ากับ 1 จากนั้นรอการตอบรับจาก DS1307 เมื่อตอบรับเรียบร้อย DS1307 จะทยอยส่งข้อมูลออกมาให้ไมโครคอนโทรลเลอร์คราวละ 1 แอดเดรสหรือ 1 ไบต์ โดยแอดเดรสที่เลือกอ่าน

ข้อมูลจะต้องมีการกำหนดมาก่อนล่วงหน้าด้วยโหมดการเขียนข้อมูลวิธีง่ายๆ คือ เข้าสู่โหมดการเขียนข้อมูลก่อน เมื่อถึงจังหวะที่ต้องการเขียนข้อมูล ให้ทำการสร้างสภาวะเริ่มต้นและส่งข้อมูลกำหนดแอดเดรสใหม่อีกครั้ง ตามด้วยเลือกโหมดการอ่านข้อมูล ข้อมูลที่ออกมาจาก DS1307 ก็จะเป็นข้อมูลจากแอดเดรสที่กำหนดไว้ก่อนหน้านี้

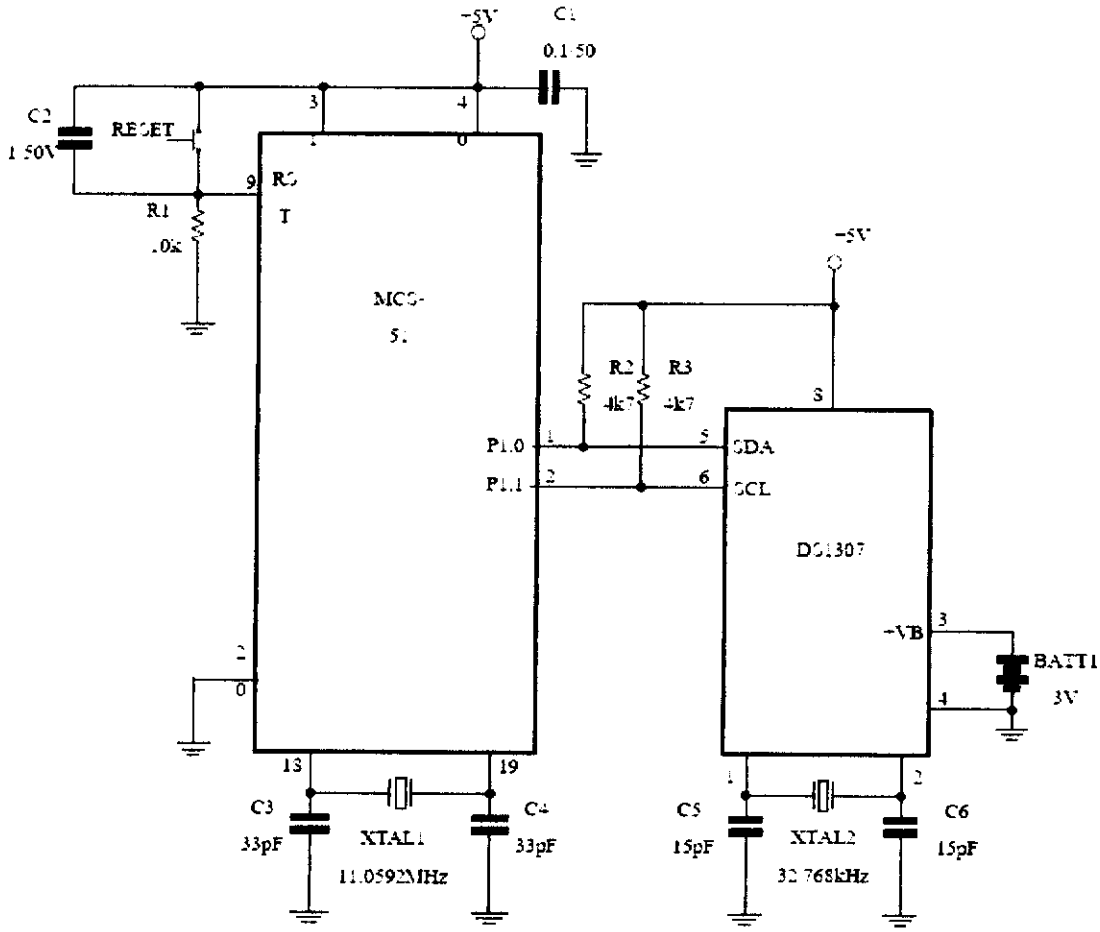


รูปที่ 3.27 รูปแบบของข้อมูลสำหรับติดต่อกับ DS1307 ในโหมดการอ่านข้อมูล

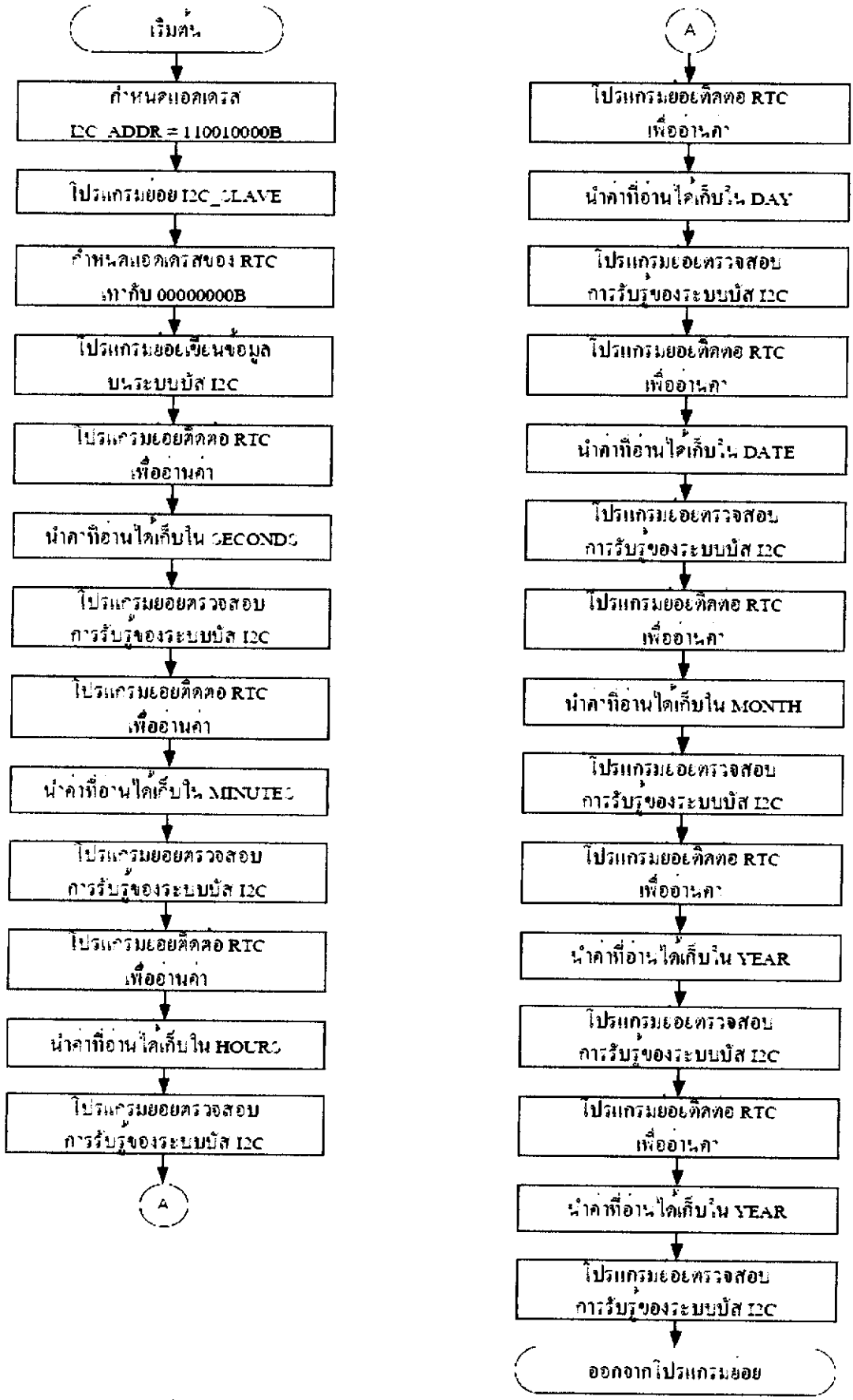
การเชื่อมต่อ DS13071 กับไมโครคอนโทรลเลอร์ MCS-51

มีตัวอย่างแสดงวงจรในรูปที่ 3.28 จะเห็นได้ว่ามีลักษณะการต่อเหมือนกับอุปกรณ์บนระบบบัส I^2C ตัวอื่นๆ ทุกประการ และสามารถที่จะต่อไอซีทั้งหมดร่วมกันบนสาย SDA และ SCL ได้ เป็นการย้ำให้เห็นถึงความสามารถพิเศษของระบบบัส I^2C ที่ผู้ใช้งานสามารถเชื่อมต่ออุปกรณ์ที่มีความต่างกันในหน้าที่การทำงานบนสายสัญญาณเดียวกันได้ ถึงการทดลองนี้ผู้ใช้งานสามารถเชื่อมต่ออุปกรณ์ระบบบัส I^2C ได้ถึง 3 ตัว 3 ลักษณะการทำงาน โดยใช้สายสัญญาณเพียง 2 เส้น

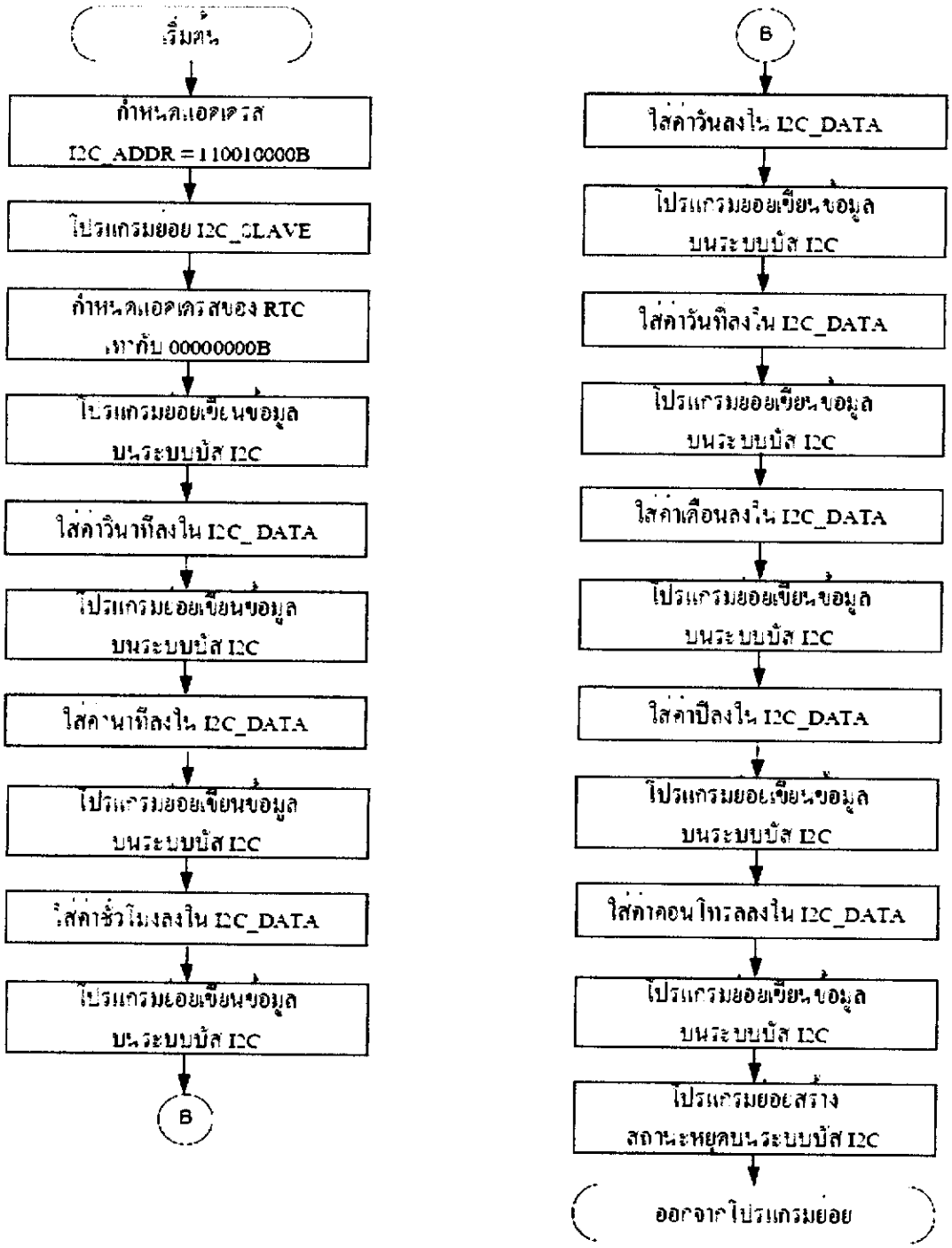
จากวงจรในรูปที่ 3.28 ไอซี DS1307 จำเป็นจะต้องต่อแบตเตอรี่ไว้ตลอดเวลาไม่ว่าจะใช้งานหรือไม่ ทั้งนี้เพื่อรักษาการทำงานของวงจรภายใน DS1307 ให้ยังคงทำงานต่อเนื่องไป เมื่อใดที่ไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูล ก็จะได้ข้อมูลที่เป็นจริงตลอดเวลา



รูปที่ 3.28 แสดงการเชื่อมต่อไมโครคอนโทรลเลอร์ MCS-51 กับไอซีรีลไทม์คล็อก DS1307



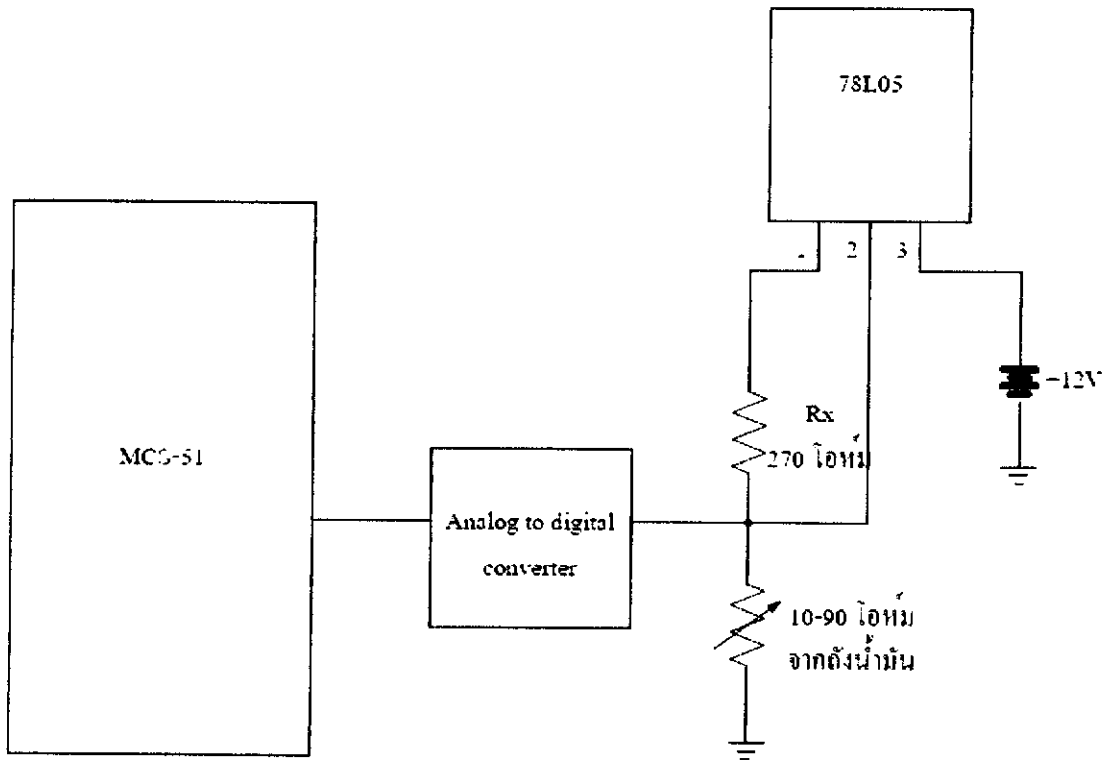
รูปที่ 3.29 โฟล์วชาร์ตการอ่านค่าจากไอซีรีลไทม์คล็อก DS1307



รูปที่ 3.30 โฟล์วชาร์ตการเขียนข้อมูลไปยังไอซีรีลไทม์คลีอก DS1307

3.7 วงจรวัดระดับน้ำมันเชื้อเพลิง

ในส่วนวัดระดับน้ำมันเชื้อเพลิงใช้ไอซี 78L05 มาสร้างเป็นวงกระแสคงที่ (constant current) เพื่อจ่ายกระแสให้กับวงจรวัดระดับน้ำมันที่มีค่าความต้านทานเปลี่ยนแปลงในช่วง 10 โอห์ม – 90 โอห์มซึ่งจะได้ค่าแรงดันเปลี่ยนแปลงในช่วง 0.2 โวลต์ – 1.8 โวลต์ แล้วนำค่าที่ได้ไปเข้าวงจร Analog to digital converter แล้วนำค่าที่ได้ไปคำนวณ และแสดงผลต่อไป



รูปที่ 3.31 แสดงวงจรวัดระดับน้ำมันเชื้อเพลิง

3.8 ส่วนแสดงสถานะต่าง ๆ

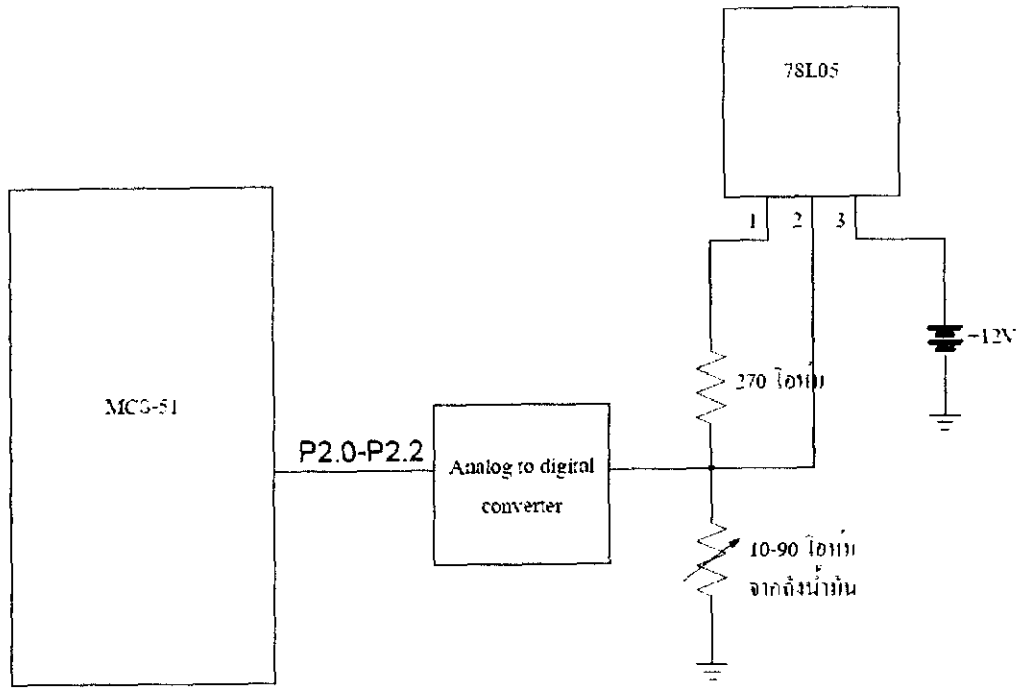
ในส่วนของการแสดงสถานะต่าง ๆ ทำได้โดยการใช้ไมโครโทรลเลอร์ MCS-51 รับค่าสถานะต่าง ๆ ของรถจักรยานยนต์แล้วนำไปแสดงผลที่จอแสดงผล ซึ่งสถานะต่าง ๆ ได้แก่

- เปิดไฟสูง
- เปิดไฟเลี้ยว (ซ้าย – ขวา)
- แสดงสถานะฉุกเฉินต่าง ๆ ได้แก่ Battery Low และ Fuel Low
- บอกเกียร์ว่างและเกียร์สูงสุด

บทที่ 4 การดำเนินการและสรุปผล

4.1 การทดลองวงจรจ่ายกระแสคงที่

การทดลองของวงจรนี้เป็นการทดลองเพื่อทดลองความคงที่ของวงจรว่าในขณะที่โหลดมีการเปลี่ยนแปลงค่าต่าง ๆ กระแสที่ได้จะมีค่าเปลี่ยนแปลงอย่างไร ซึ่งได้ทำการทดลอง ดังนี้



รูปที่ 4.1 แสดงวงจรที่ใช้ในการทดลองวงจรจ่ายกระแสคงที่

ซึ่งในการทดลองได้กำหนดให้ค่ากระแสคงที่ประมาณ 22 mA ซึ่งคำนวนจาก

$$I_{cons} = \frac{5}{Rx} + I_Q ; I_Q = 3.8mA, Rx = 270\Omega$$

$$= 18.5mA + 3.8mA$$

$$\approx 22mA$$

ซึ่งจะได้ผลการทดลองดังนี้

ค่าความต้านทาน (Ω)	ทดลอง (Volt)	คำนวณ (Volt)	ค่าผิดพลาด (%)
5	0.09	0.10	10
10	0.19	0.20	5
50	0.98	1.0	2
100	1.98	2.0	1
120	2.3	2.4	4

ตารางที่ 4.1 ผลการทดลองวงจรจ่ายกระแสคงที่

โดยที่จากการทดลองค่าที่ได้จาการทดลองอยู่ในช่วงที่ยอมรับได้ นั่นแสดงว่าวงจรจ่ายกระแสคงที่มีเสถียรภาพในการจ่ายกระแสซึ่งวงจรที่ได้จะไปจ่ายกระแสให้กับวงจรวัดระดับน้ำมันที่มีค่าความต้านทานเปลี่ยนแปลงในช่วง 10 โอห์ม – 90 โอห์มซึ่งจะได้ค่าแรงดันเปลี่ยนแปลงในช่วง 0.2 โวลต์ – 2 โวลต์ แล้วนำค่าที่ได้ไปประมวลผลต่อไป

4.2 วงจร A/D

คือ วงจรที่แปลงแรงดันให้กลายเป็นข้อมูลดิจิตอล

ซึ่งการทดลองนี้เป็นการทดสอบค่าที่ออกมาจาก A/D เบอร์ MCP3201 ซึ่งได้จากการป้อนแรงดันจากแหล่งจ่ายไฟ

แรงดันอินพุต(โวลต์)	ค่าลอจิกเอาต์พุต	ค่าทางเอาต์พุต	ค่าแรงดันเอาต์พุต (โวลต์)
1	0011 0010 0001	801	0.978
2	0110 0010 1111	1583	1.932
2.5	0111 1111 1010	2042	2.493
3	1001 1110 0101	2533	3.092
4	1100 1110 1110	3310	4.041
5	1111 1111 0111	4087	4.989

ตารางที่ 4.2 แสดงค่าทางเอาต์พุตที่ได้จากวงจร Analogue to Digital Converter

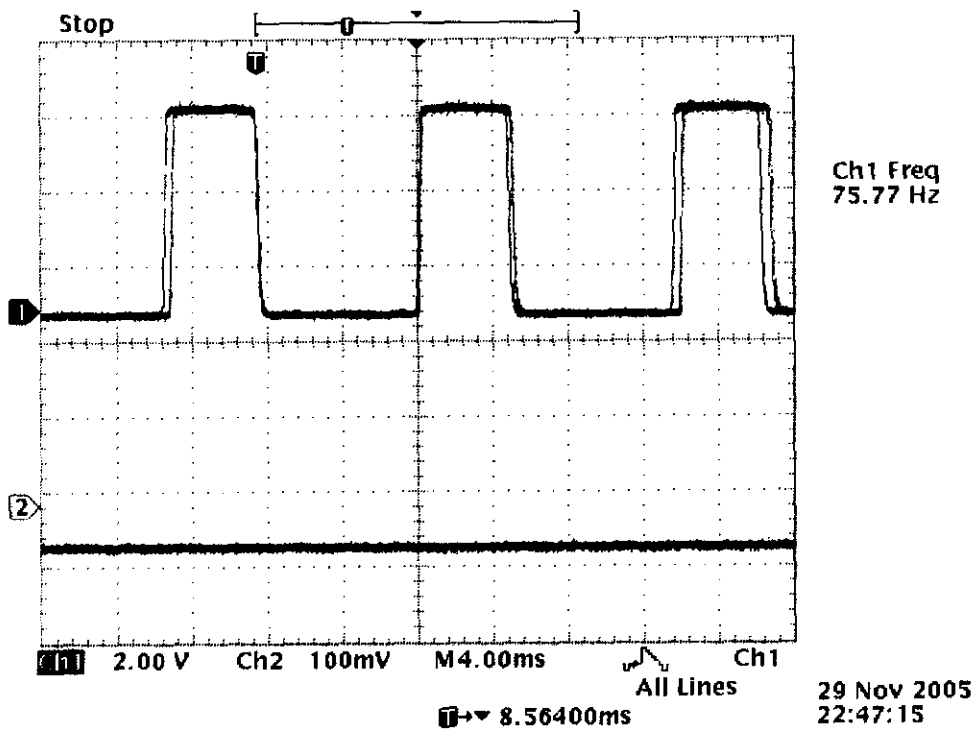
*หมายเหตุ ค่าทางเอาต์พุต หมายถึงการนำค่าลอจิกเอาต์พุตมาทำให้เป็นเลขฐานสิบ, V_{ref} = 5 โวลต์ โดยที่ค่าทางเอาต์พุตคำนวณจาก

$$\text{Digital output code} = (V_{in} \times 4096) / V_{ref}$$

4.3 การทดลองวัดรูปสัญญาณ

- สัญญาณจาก Opto couple

การทดลองทำโดยต่อวงจร Opto couple แล้วใช้ล้อเจาะรูติดกับสายไมล์ของรถจักรยานยนต์ แล้วหมุนผ่าน Opto couple แล้วจับสัญญาณ output ที่ออกจาก Opto couple ได้ output ดังรูปที่ 4.2

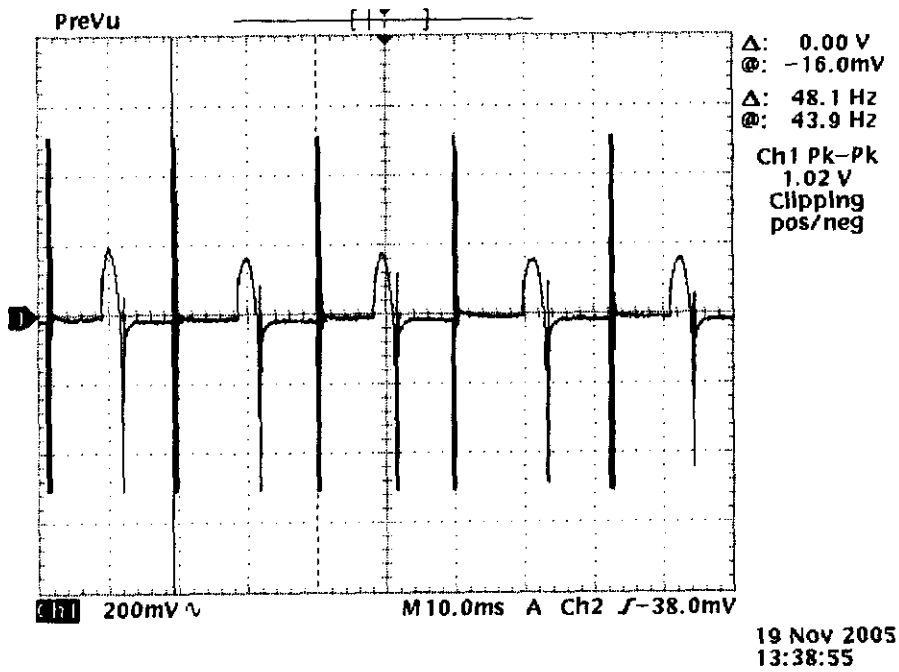


รูปที่ 4.2 สัญญาณที่ตรวจจับได้จาก Opto couple

จากรูปสัญญาณดังรูปที่ 4.2 เป็นสัญญาณที่ไมโครคอนโทรนเลอร์ (MCS-51) ตรวจจับได้

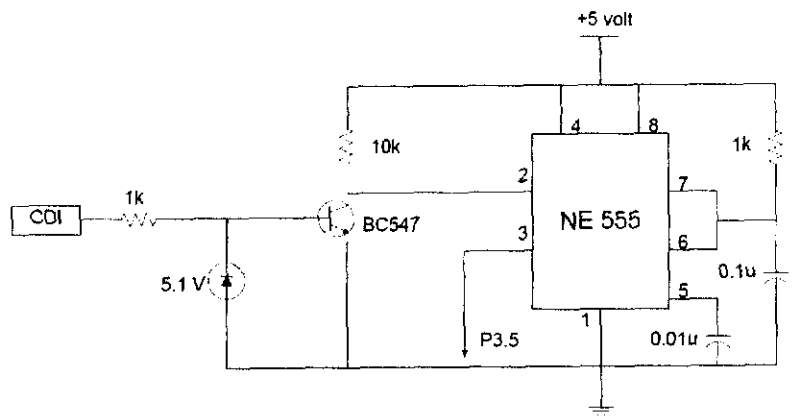
- สัญญาณจาก CDI ของรถจักรยานยนต์

ทำการตรวจจับสัญญาณที่ได้จากการจ่ายสัญญาณจากหน้าทองขาวก่อนที่จะเข้าสู่วงจร CDI โดยได้รูปสัญญาณดังรูปที่ 4.3



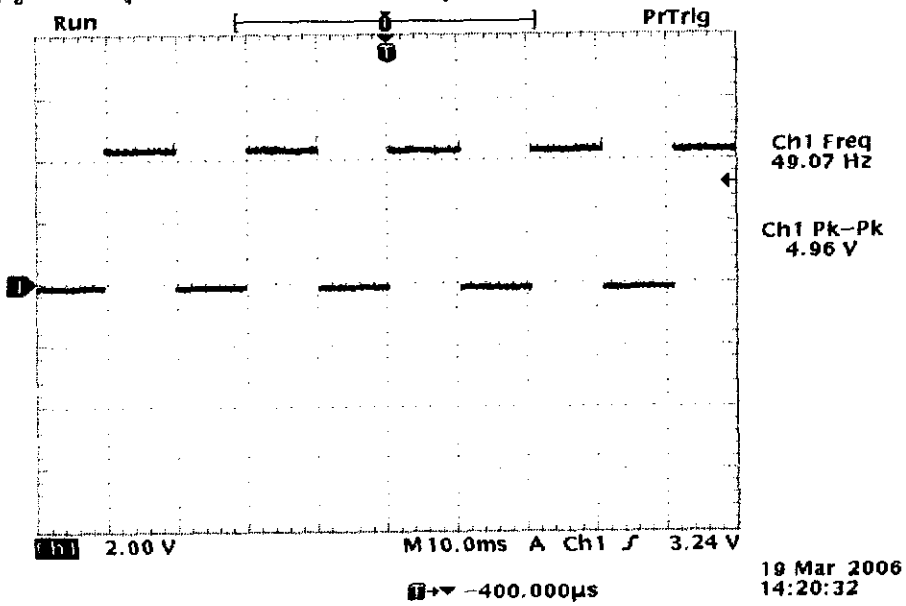
รูปที่ 4.3 สัญญาณที่ตรวจจับได้จาก CDI ของรถจักรยานยนต์

จากรูปสัญญาณดังรูปที่ 4.3 เป็นสัญญาณที่ไมโครคอนโทรนเลอร์ (MCS-51) ไม่สามารถตรวจจับได้จึงจำเป็นต้องมีวงจรโมโนสเตเบิ้ล เพื่อให้ไมโครคอนโทรนเลอร์ (MCS-51) ตรวจจับได้ ดังนั้นจึงต้องนำมาผ่านวงจรโมโนสเตเบิล ดังรูป



รูปที่ 4.4 แสดงวงจรโมโนสเตเบิลที่ใช้ในการทริกสัญญาณที่ได้จาก CDI

ซึ่งได้สัญญาณเอาต์พุตจากวงจรโมโนสเตเบิลดังรูป

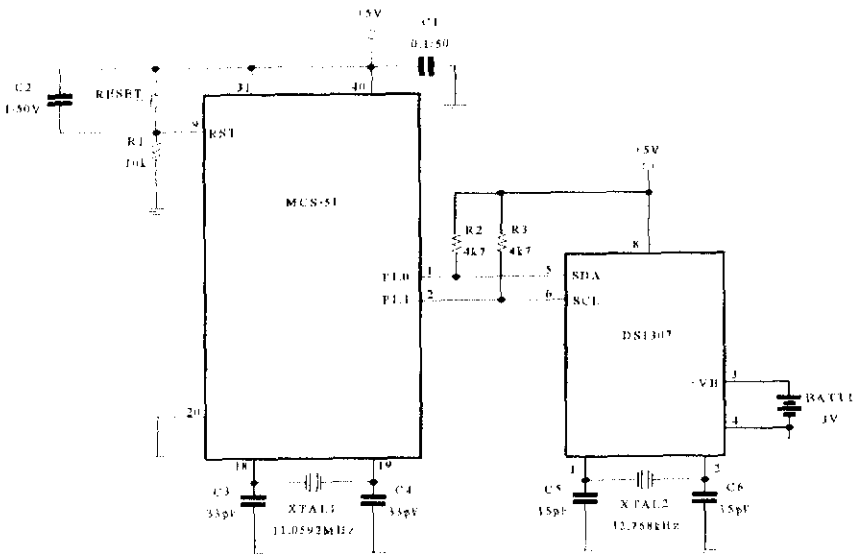


รูปที่ 4.5 แสดงสัญญาณจากการทริกของวงจรโมโนสเตเบิล

4.4 การแสดงค่าที่จอแสดงผล

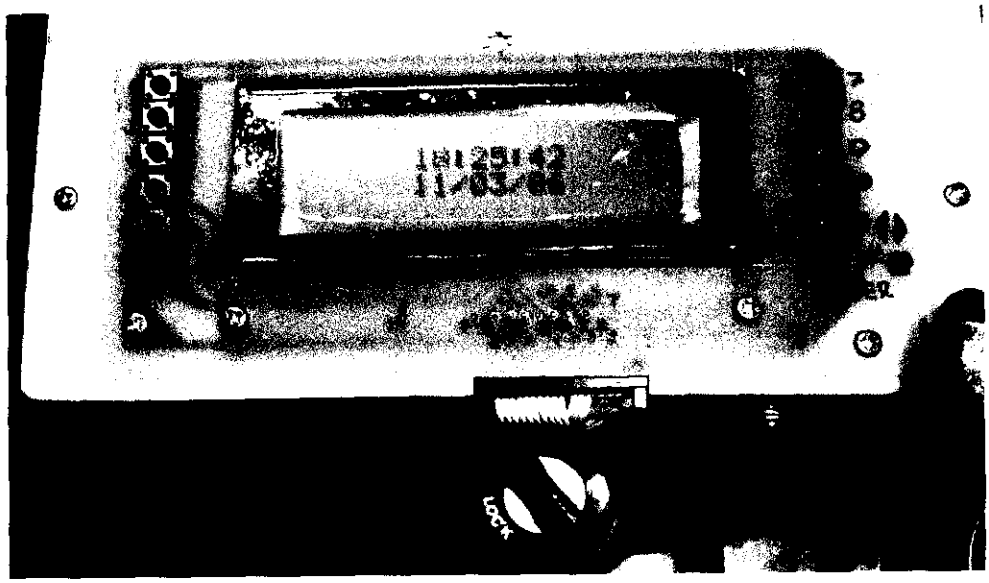
4.4.1 ส่วนแสดงวัน เวลา

เป็นการใช้ IC ไอซีรีลไทม์คล็อก DS1307 ซึ่งให้ข้อมูลตั้งแต่วินาทีถึงปี สามารถให้ข้อมูลเวลาได้อย่างเที่ยงตรงถึงปีคริสตศักราช 2100 มีหน่วยความจำนอนโวลาไทล์แรม 56 ไบต์อยู่ภายในสามารถใช้เก็บข้อมูลทั่วไปได้ ใช้การเชื่อมต่อแบบระบบบัส I2C มีวงจรตรวจจับไฟเลี้ยงต่ำหรือหายไป อย่างอัตโนมัติ และสามารถรักษาข้อมูลเวลาไว้ได้แม้ไม่มีไฟเลี้ยงไอซีโดยต่อเข้ากับไมโครคอนโทรลเลอร์ดังรูป

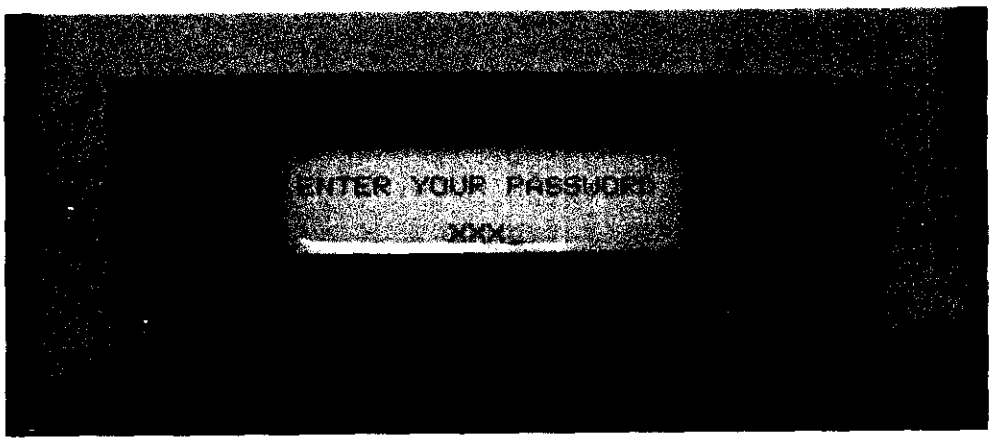


รูปที่ 4.6 แสดงการเชื่อมต่อไมโครคอนโทรลเลอร์ MCS-51 กับไอซีรีลไทม์คล็อก DS1307

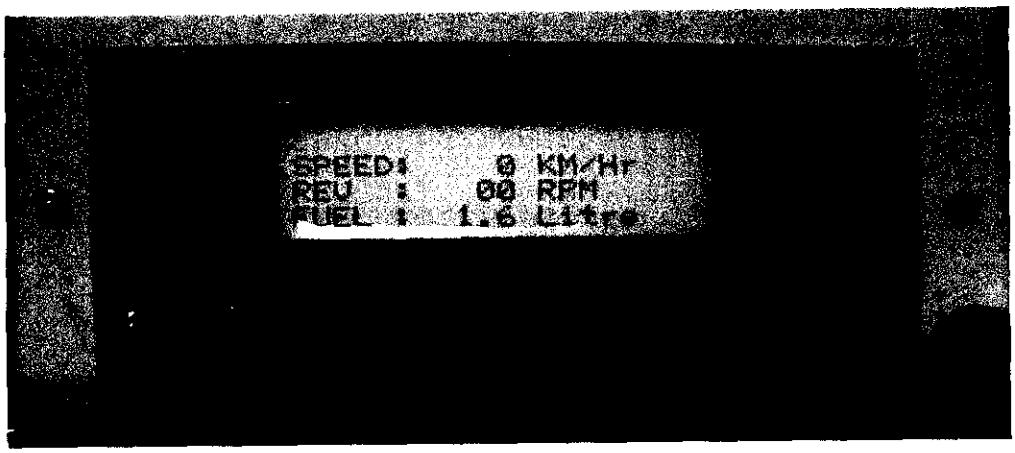
โดยการแสดงเวลาของเครื่องจะแสดงในโหมดที่สวิตช์กุญแจมีการปิดเท่านั้น เมื่อสวิช์กุญแจมีการเปิดก็จะเข้าสู่โหมดการถามรหัสต่อไป ในรูปที่ เป็นรูปที่แสดงวัน เวลา ของเครื่องควบคุมรถจักรยานยนต์โดยที่เมื่อมีการเปิดสวิต์กุญแจจะมีการถามรหัสผ่าน จึงต้องมีการป้อนรหัสผ่าน 4 หลัก เมื่อมีการป้อนรหัสผ่านถูกต้องแล้วจึงเข้าสู่หน้าแสดงความเร็วรถ ระดับน้ำมัน และแสดงค่าต่างๆ ต่อไป



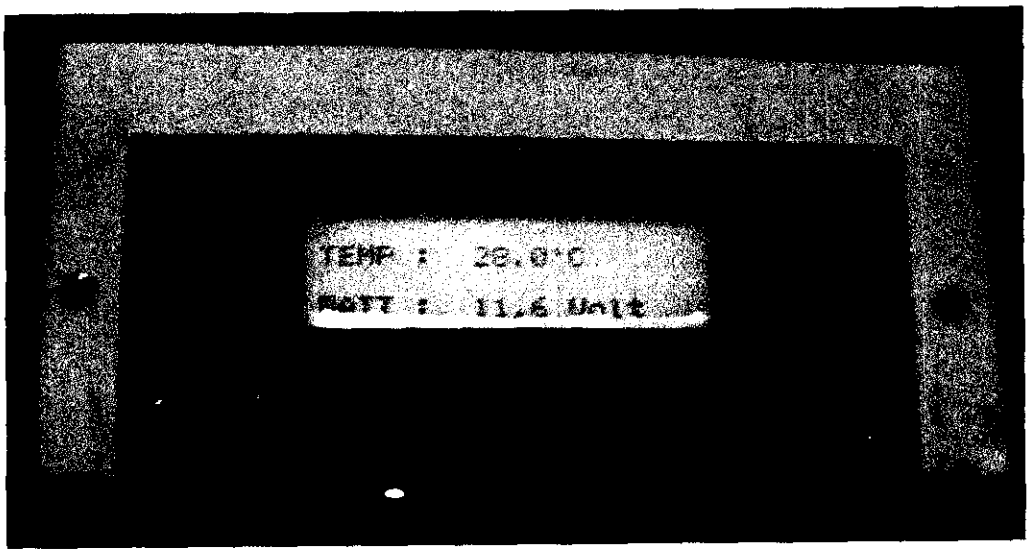
รูปที่ 4.7 การแสดงเวลาของวงจรนาฬิกาในขณะที่สวิตช์กุญแจยังปิดอยู่



รูปที่ 4.8 การแสดงการถามรหัสผ่านเมื่อเปิดสวิตช์กุญแจ



รูปที่ 4.9 การแสดงค่าความเร็วรถ ความเร็วรอบ และแสดงระดับน้ำมัน ในหน้าแรก



รูปที่ 4.10 การแสดงอุณหภูมิและ ระดับน้ำมันแบตเตอรี่ในหน้าที่สองของการแสดงผล

4.5 สรุปผลการดำเนินการ

ในการดำเนินการในโครงงานครั้งนี้ได้ทำการทดลองสร้างวงจรที่สำคัญหลายๆวงจร กล่าวคือมีวงจรที่ได้สร้างแล้วดังนี้

1. วงจรวัดความเร็วรถ
2. วงจรวัดความเร็วรถ
3. วงจรวัดอุณหภูมิ
4. วงจรวัดระดับแรงดันแบตเตอรี่
5. วงจรวัดระดับน้ำมันเชื้อเพลิง
6. วงจรกันขโมย
7. แสดงสถานะการทำงานต่างๆของรถ เช่น เปิด-ปิดไฟเลี้ยว ไฟหน้ารถ เป็นต้น

โดยมีการเขียนซอร์ฟแวร์รองรับวงจรต่างๆเหล่านี้ซึ่งมีการควบคุมผ่านไมโครคอนโทรลเลอร์ MCS-51 เบอร์ AT89C52จากผลการดำเนินการที่ผ่านมาเมื่อนำเครื่องแสดงผลแบบหลายฟังก์ชันและระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์ที่ได้ไปใช้งาน พบว่าสามารถนำไปใช้งานได้ตามที่ได้ออกแบบไว้ โดยมีปัญหาบ้างในบางส่วนการทำงาน

บรรณานุกรม

1. วรพจน์ กรแก้ววัฒนกุล , ชัยวัฒน์ ลิ้มพรจิตรวิลัย , "**เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51**" , บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด , 2521
2. นคร ภักดีชาติ, ชัยวัฒน์ ลิ้มพรจิตรวิลัย, "**ทดลองและใช้งานไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษาซี**", บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด, 2521
3. ธีรบูลย์ หล่อวิเชียรรุ่งและคณะ "**ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษา C**", บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด, 2521

ภาคผนวก

ภาคผนวก ก.

ชุดคำสั่ง
สำหรับไมโครคอนทรลเลอร์ตัวหลัก
(SOUCE CODE FOR MASTER MCU)

```
// SUKARASUT MEKSIRI  45010774
// SUPACHAI KERNKEAW 45010777
// ARTHIT PHUDEJ          45010962
// KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
// ELCECTRONICS ENGINEERING
//************************** stdio file ******************************//

#ifndef __STDIO_H__
#define __STDIO_H__

#ifndef EOF
 #define EOF -1
#endif

#ifndef NULL
 #define NULL ((void *) 0)
#endif

#ifndef _SIZE_T
 #define _SIZE_T
 typedef unsigned int size_t;
#endif

#pragma SAVE
#pragma REGPARMS
extern char _getkey (void);
extern char getchar (void);
extern char ungetchar (char);
extern char putchar (char);
extern int printf  (const char *, ...);
extern int sprintf  (char *, const char *, ...);
extern int vprintf  (const char *, char *);
extern int vsprintf (char *, const char *, char *);
extern char *gets (char *, int n);
extern int scanf (const char *, ...);
extern int sscanf (char *, const char *, ...);
```

```
extern int puts (const char *);

#pragma RESTORE

#endif

//****************************** intrins ********************************//
#ifndef __INTRINS_H__
#define __INTRINS_H__

extern void          _nop_   (void);
extern bit           _testbit_ (bit);
extern unsigned char _cror_   (unsigned char, unsigned char);
extern unsigned int  _iror_   (unsigned int,  unsigned char);
extern unsigned long _lror_   (unsigned long, unsigned char);
extern unsigned char _crol_   (unsigned char, unsigned char);
extern unsigned int  _irol_   (unsigned int,  unsigned char);
extern unsigned long _lrol_   (unsigned long, unsigned char);
extern unsigned char _chkfloat_(float);
extern void          _push_   (unsigned char _sfr);
extern void          _pop_    (unsigned char _sfr);

#endif
//*****************************************************************************
*************
#ifndef __STDIO_H__
#define __STDIO_H__

#ifndef EOF
 #define EOF -1
#endif

#ifndef NULL
 #define NULL ((void *) 0)
#endif
```

```
#ifndef _SIZE_T
 #define _SIZE_T
 typedef unsigned int size_t;
#endif
//***************************************************************************//
#pragma SAVE
#pragma REGPARMS
extern char _getkey (void);
extern char getchar (void);
extern char ungetchar (char);
extern char putchar (char);
extern int printf  (const char *, ...);
extern int sprintf  (char *, const char *, ...);
extern int vprintf  (const char *, char *);
extern int vsprintf (char *, const char *, char *);
extern char *gets (char *, int n);
extern int scanf (const char *, ...);
extern int sscanf (char *, const char *, ...);
extern int puts (const char *);

#pragma RESTORE

#endif

#ifndef __REG52_H__
#define __REG52_H__

/*  BYTE Registers  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
```

```
sfr DPL   = 0x82;
sfr DPH   = 0x83;
sfr PCON  = 0x87;
sfr TCON  = 0x88;
sfr TMOD  = 0x89;
sfr TL0   = 0x8A;
sfr TL1   = 0x8B;
sfr TH0   = 0x8C;
sfr TH1   = 0x8D;
sfr IE    = 0xA8;
sfr IP    = 0xB8;
sfr SCON  = 0x98;
sfr SBUF  = 0x99;

/*  8052 Extensions  */
sfr T2CON  = 0xC8;
sfr RCAP2L = 0xCA;
sfr RCAP2H = 0xCB;
sfr TL2   = 0xCC;
sfr TH2   = 0xCD;


/*  BIT Registers  */
/*  PSW  */
sbit CY   = PSW^7;
sbit AC   = PSW^6;
sbit F0   = PSW^5;
sbit RS1  = PSW^4;
sbit RS0  = PSW^3;
sbit OV   = PSW^2;
sbit P    = PSW^0; //8052 only

/*  TCON  */
sbit TF1  = TCON^7;
sbit TR1  = TCON^6;
sbit TF0  = TCON^5;
```

```
sbit TR0  = TCON^4;
sbit IE1  = TCON^3;
sbit IT1  = TCON^2;
sbit IE0  = TCON^1;
sbit IT0  = TCON^0;

/* IE */
sbit EA   = IE^7;
sbit ET2  = IE^5; //8052 only
sbit ES   = IE^4;
sbit ET1  = IE^3;
sbit EX1  = IE^2;
sbit ET0  = IE^1;
sbit EX0  = IE^0;

/* IP */
sbit PT2  = IP^5;
sbit PS   = IP^4;
sbit PT1  = IP^3;
sbit PX1  = IP^2;
sbit PT0  = IP^1;
sbit PX0  = IP^0;

/* P3 */
sbit RD   = P3^7;
sbit WR   = P3^6;
sbit T1   = P3^5;
sbit T0   = P3^4;
sbit INT1 = P3^3;
sbit INT0 = P3^2;
sbit TXD  = P3^1;
sbit RXD  = P3^0;

/* SCON */
sbit SM0  = SCON^7;
sbit SM1  = SCON^6;
```

```
sbit SM2  = SCON^5;
sbit REN  = SCON^4;
sbit TB8  = SCON^3;
sbit RB8  = SCON^2;
sbit TI   = SCON^1;
sbit RI   = SCON^0;

/* P1 */
sbit T2EX  = P1^1; // 8052 only
sbit T2   = P1^0; // 8052 only

/* T2CON */
sbit TF2   = T2CON^7;
sbit EXF2  = T2CON^6;
sbit RCLK  = T2CON^5;
sbit TCLK  = T2CON^4;
sbit EXEN2 = T2CON^3;
sbit TR2   = T2CON^2;
sbit C_T2  = T2CON^1;
sbit CP_RL2 = T2CON^0;
//
#endif
//*****************************************************************************//
unsigned char dat = 0;
unsigned char count = 0;
code unsigned char text1[]="ENTER YOUR PASSWORD";
code unsigned char text4[]="WRONG PASSWORD";
code unsigned char text5[]="WELCOME";
signed char pass[6];
unsigned char d_morey[7];
code unsigned char display[]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B};
sbit c1 = P2^6;
sbit c2 = P2^5;
sbit c3 = P2^4;
sbit r1 = P2^0;
sbit r2 = P2^1;
```

```
sbit r3 = P2^2;
sbit r4 = P2^3;
sbit e = P3^6;
sbit rs = P3^7;
sbit start = P1^7;
sbit light = P2^7;
sbit turnon =P3^2;


//*****************************************************************************//
void delay (int tick)
{
        unsigned int i,j;
        for(i=0;i<tick;i++)
        for(j=0;j<250;j++);
}
//*****************************************************************************//
void lcd_command(unsigned char com)
{
        rs = 0;
        e = 1;
        P0 = com;
        delay(10);
        e = 0;
        delay(10);
}
void lcd_text(unsigned char text)
{
        rs = 1;
        e = 1;
        P0 = text;
        delay(10);
        e = 0;
        delay(10);
}
```

```
void lcd_clear()
{
        lcd_command(0x01);
}
void lcd_jumporigin()
{
        lcd_command(0x02);
}
void lcd_init()
{       delay(250);
        delay(250);
        lcd_command(0x38);
        lcd_command(0x0C);
        lcd_command(0x01);
}
void delay_db(int time)
{
        do
        {
        time--;
        }while(time>0);
}
//*****************************************************************************//
unsigned char scankey(void)
{
        unsigned char ret = 0xFF;
        c1 = 0;
        if(r1==0)
        {
    delay_db(30000);
                ret = 0x01;
        }
        if(r2==0)
        {
          delay_db(30000);
                ret = 0x04;
```

```
}
if(r3==0)
{
        delay_db(30000);
        ret = 0x07;
}
if(r4==0)
{
        delay_db(30000);
        ret = 0x0A;
}
c1 = 1;

c2 = 0;
if(r1==0)
{
        delay_db(30000);
        ret = 0x02;
}
if(r2==0)
{
        delay_db(30000);
        ret = 0x05;
}
if(r3==0)
{
        delay_db(30000);
        ret = 0x08;
}
if(r4==0)
{
        delay_db(30000);
        ret = 0x00;
}
```

```
        c2 = 1;
        c3 = 0;
        if(r1==0)
        {
                delay_db(30000);
                ret = 0x03;
        }
        if(r2==0)
        {
                delay_db(30000);
                ret = 0x06;
        }
        if(r3==0)
        {
                delay_db(30000);
                ret = 0x09;
        }
        if(r4==0)
        {
                delay_db(30000);
                ret = 0x0B;
        }
        c3 = 1;
        return(ret);
}
//*****************************************************************************//
void start_display()
{
        unsigned char m;
        P2 = 0xff;
        lcd_init();
        lcd_command(0xC0);
        for(m=0;m<19;m++)
        lcd_text(text1[m]);
                //
  }
```

```
//*****************************************************************************//
void wrong pass()
{
        unsigned char m;
                                P2 = 0xff;
              lcd_init();
                                lcd_command(0x01);
                                lcd_command(0xC3);
                                for(m=0;m<14;m++)
                          lcd_text(text4[m]);
}
void welcome()
{
        unsigned char m;
                    lcd_command(0x01);
                                lcd_command(0xC7);
                                for(m=0;m<7;m++)
                          lcd_text(text5[m]);

}
//*****************************************************************************//
void pass check(void)
{

  start = 1;
        while(start==1);
  {
  unsigned char key = 0;
        unsigned char x_key = 0;
  start display();
  lcd_command(0xDC);
  lcd_command(0x0F);
        while(x_key < 5)
   {
            key = scankey();
        if(key != 0xff)
```

```
{
        x_key++;
        pass[x_key] = display[key];
 if(x_key < 5)
                                {
                                lcd_command(0xDB+x_key);
        lcd_text('X');
                                }
                                else
                                {
                                lcd_command(0xE0);
                                lcd_text(' ');
                          }

                          }
}

 if(x_key == 5)

 {
          if(pass[5] == 11)
          {
          if(pass[4] == 1)
           {
           if(pass[3] == 2)
                    {
                    if(pass[2] == 3)
                    {
                     if(pass[1] == 4)
                       pass[6] = 0x0D;

                    }
                    else wrong_pass();
                    }
            else wrong_pass();
                    }
```

```
                else wrong_pass();
                                }
              else wrong_pass();
                                }




    }


          }
//*****************************************************************************************//
void idle_state()
{
   code unsigned char text6[]="SPEED:";
          code unsigned char text20[]="Km/Hr";
          code unsigned char text7[]="REV  :";
          code unsigned char text17[]="RPM";
          code unsigned char text8[]="FUEL :";
          code unsigned char text18[]="Litre";

   unsigned char m;
//
          lcd_command(0xC0);
          for(m=0;m<6;m++)
          lcd_text(text6[m]);


          lcd_command(0xCC);
          for(m=0;m<5;m++)
          lcd_text(text20[m]);
  //
  lcd_command(0x94);
          for(m=0;m<6;m++)
  lcd_text(text7[m]);
          //
  lcd_command(0xA0);
          for(m=0;m<3;m++)
```

```
    lcd_text(text17[m]);
            //
            lcd_command(0xD4);
            for(m=0;m<6;m++)
    lcd_text(text8[m]);
            //
            lcd_command(0xE0);
            for(m=0;m<5;m++)
    lcd_text(text18[m]);
            //
}
//*****************************************************************************************//
void page_2 ()
{
            code unsigned char text9[]="BATT :";
            code unsigned char text19[]="Volt";
    code unsigned char text15[]="TEMP : ";
    unsigned char m;
            lcd_command(0xC0);
            for(m=0;m<7;m++)
            lcd_text(text15[m]);
            //

            //
            lcd_command(0xCA);
    lcd_text('.');
            //

            //
            lcd_command(0xCC);
    lcd_text(0xB2);
    lcd_text('C');
//
            lcd_command(0xd4);
            for(m=0;m<6;m++)
            lcd_text(text9[m]);
```

```
        //
        lcd_command(0xE1);
        for(m=0;m<4;m++)
        lcd_text(text19[m]);
        //
}

void gear_check()
{
code unsigned char text10[]="[ TOP ]";
code unsigned char text11[]="[  N  ]";
unsigned char key = P1;
unsigned char m;
unsigned char n = 0;
P1 = 0x80;

switch(key)
  {
   case 0x02 : while(n < 2)
                {
                                        n++;
                                        lcd_command(0x87);
                for(m=0;m<7;m++)
          lcd_text(tex110[m]);
                                        }
   case 0x20 : while(n < 2)
                {
                                        n++;
                                        lcd_command(0x87);
                for(m=0;m<7;m++)
          lcd_text(text11[m]);
                                        }
   default :   while(n<2)
          {
                                        n++;
                                        lcd_command(0x87);
```

```
                for(m=0;m<7;m++)
            lcd_text(' ');
                                        }
 }
}
//*****************************************************************************************//
void status()
{
code unsigned char text12[]="<<<<";
code unsigned char text13[]=">>>>";
code unsigned char text14[]="[H-BEAM]";
unsigned char key = P1;
unsigned char m;
unsigned char n = 0;
P1 = 0x80;
gear_check();
switch(key)
  {
   case 0x04 : while(n < 2)
                {
                                        n++;
                                        lcd_command(0x81);
                for(m=0;m<4;m++)
           lcd_text(text12[m]);
                                        lcd_command(0x81);
                                        lcd_text(' ');
                                        lcd_text(' ');
                                        lcd_text(' ');
                                        lcd_text(' ');
                                        }
   case 0x40 : while(n < 2)
                {
                                        n++;
                                        lcd_command(0x8F);
                for(m=0;m<4;m++)
           lcd_text(text13[m]);
```

```
                                        lcd_command(0x8F);
                                        lcd_text(' ');
                                        lcd_text(' ');
                                        lcd_text(' ');
                                        lcd_text(' ');
                                        }
            case 0x08 : while(n < 2)
                  {
                                        n++;
                                        lcd_command(0x87);
                  for(m=0;m<8;m++)
            lcd_text(text14[m]);
                                        lcd_command(0x87);
                                        }
                              }
}
//***************************** Temp ************************************//
sbit onewire = P1^6;
bit ans;
unsigned char i;
void ds1820_reset()
{
            onewire = 0;
            for(i=0;i<100;i++) _nop_();
            onewire = 1;
            for(i=0;i<2;i++) _nop_();
}
void ds1820_ans(void)
{
            ans = 0;
            while(onewire);
            while(~onewire);
            for(i=0;i<2;i++) _nop_();
}
bit readbit (void)
{
```

```
        bit dat;
        onewire = 0;
        _nop_();
        _nop_();
        onewire = 1;
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        dat = onewire;
  for(i=0;i<8;i++) _nop_();
        return(dat);
}
unsigned char ds1820_read()
{
        unsigned char i,j,dat;
        dat = 0;
        for(i=0;i<11;i++)
        {
                j = readbit();
                dat = (j<<7)|(dat>>1);
        }
        return(dat);
}
/**************    Function Write data to 1-wire bus  ***********************/
void ds1820_write(unsigned char com)
{
        unsigned char j;
        bit send;
        for(j=0;j<8;j++)
        {
                send = com & 0x01;
                com = com>>1;
                if(send)
                {
                        onewire = 0;
```

```
                    _nop_();
                    _nop_();
                    _nop_();
                    _nop_();
                    onewire = 1;
                    for(i=0;i<8;i++) _nop_();
                }
                else
{
                    onewire = 0;
                    for(i=0;i<8;i++) _nop_();
                    onewire = 1;
                    _nop_();
                    _nop_();
                    _nop_();
                    _nop_();
                }
        }
}
void delay_convert(unsigned int count)
{
        do
        {
                count--;
        }while(count>0);
}
//********************************* VELOCITY OF MOTORCYCLE ************************//
void velocity  (void)
{
  unsigned char buf;
        unsigned int value,value_0,value_1;
        TMOD =0x06;
        EA = 1;
        TH0 = 0x00;
        TL0 = 0x00;
        TR0 = 1;
```

```
        ET0 = 1;
        delay(114);    //gate delay 0.25 sec normal 114
        TR0 = 0;
        ET0 = 0;
        value = TL0*2;
        value_0 = value/100;
  if(value_0 == 0)
        {
        lcd_command(0xC8);
        lcd_text(' ');
        }
else
{
        lcd_command(0xC8);
  buf = value_0 | 0x30;
  lcd_text(buf);
}
  value_1 = value%100;
        value = value_1/10;
{
        if(value_0 == 0)
        {
                if(value == 0)
                {
                        lcd_command(0xC9);
                        lcd_text(' ');
                }

  else
        {
                        lcd_command(0xC9);
                        buf = value | 0x30;
                lcd_text(buf);
        }
        }
 else
```

```
            {
                        lcd_command(0xC9);
                        buf = value | 0x30;
                lcd_text(buf);
            }
}
   value = value_1%10;
            lcd_command(0xCA);
            buf = value | 0x30;
   lcd_text(buf);
            TMOD =0x00;
            EA = 0;
}
//******************************** RESOLUTION  ************************************//
void revolution (void)
{
            unsigned char buf;
   unsigned int value,value_0,value_1,value_2,value_3,value_4;
            TMOD =0x60;
            EA = 1;
            TH1 = 0x00;
            TL1 = 0x00;
            TR1 = 1;
            ET1 = 1;
            delay(114);    //gate delay 0.25 sec [normal 114]
            TR1 = 0;
            ET1 = 0;
            value = TL1*240;
            value_0 = value/1000;
  if(value_0 == 0)
            {
            lcd_command(0x9B);
            lcd_text(' ');
            }
            else
            {
```

```
        lcd_command(0x9B);
  buf = value_0 | 0x30;
  lcd_text(buf);
        }
  value_1 = value%1000;
        value = value_1/100;

        if(value_0 == 0)
        {
                if(value == 0)
                {
                        lcd_command(0x9C);
                        lcd_text(' ');
                }
                else
                {
                        lcd_command(0x9C);
                        buf = value | 0x30;
                lcd_text(buf);
                }
        }
else
        {
                        lcd_command(0x9C);
                        buf = value | 0x30;
                lcd_text(buf);
        }

 value_2 = value_1%100;
        value_3 = value_2/10;
        lcd_command(0x9D);
        buf = value_3 | 0x30;
lcd_text(buf);
        //////
        value_4 = value_2%10;
        lcd_command(0x9E);
```

```
        buf = value_4 | 0x30;
   lcd_text(buf);
        TMOD =0x00;
        EA = 0;
}
//****************************** TEMPERATURE ************************************//
void temperature (void)
{
        unsigned char temp,temp_2 = 0,temp_1 = 0,temp_h = 0,temp_hh = 0,half = 0;
                ds1820_reset();
                ds1820_ans();
                ds1820_write(0xCC);
                ds1820_write(0x44);
                delay_convert(50000);
                ds1820_reset();
                ds1820_ans();
                ds1820_write(0xCC);
                ds1820_write(0xBE);
                temp = ds1820_read();
                half = temp & 0x01;
                if(half == 0x01)
                        {
                        half = 5 | 0x30;
                        }
                else
                        {
                        half = 0 | 0x30;
                        }
                        temp_2 = temp>>1;
                        temp = temp_2/100;
                        if(temp == 0)
                                {
                                  lcd_command(0xC7);
                                  lcd_text(' ');
                                 }
```

```
                else
                {
                lcd_command(0xC7);
                temp_hh = temp | 0x30;
                lcd_text(temp_hh);
                }
        temp = (temp_2%100);
        lcd_command(0xC8);
        temp_h = (temp/10)|0x30;
        lcd_text(temp_h);
        lcd_command(0xC9);
        temp_l = (temp%10)|0x30;
        lcd_text(temp_l);


                /*-------------------------Send Thermal to LCD display-----------------------------*/
        lcd_command(0xCA);
        lcd_text('.');
        lcd_text(half);
}
                                        //*****************************//
unsigned char buf;
void oil (void)
{
        lcd_command(0xDC);
        buf = ((d_morcy[1] & 0xF0)>>4)|(0x30);
        lcd_text(buf);
  if(buf<1)
        {
        lcd_command(0x86);
        lcd_text('L');
        lcd_text('o');
        lcd_text('w');
        lcd_text(' ');
        lcd_text('f');
        lcd_text('u');
```

```
            lcd_text('e');
            lcd_text('l');
            }
            lcd_text('.');
//
            buf = (d_morcy[1] & 0x0F) | 0x30;
            lcd_text(buf);
            //
}
void bat (void)
{
  unsigned char buf_1,buf_2;
            buf_1 = (d_morcy[0] & 0xF0)>>4;
            if(buf_1<0x08)
            {
            lcd_command(0x86);
            lcd_text('L');
            lcd_text('o');
            lcd_text('w');
            lcd_text(' ');
            lcd_text('B');
            lcd_text('a');
            lcd_text('t');
            lcd_text('t');
            lcd_text('e');
            lcd_text('r');
            lcd_text('y');
            }
  lcd_command(0xDC);
            buf = buf_1/10;
            lcd_text(buf | 0x30);
            buf_2 = buf_1%10;
            lcd_text(buf_2 | 0x30);
            lcd_command(0xDE);
            lcd_text('.');
            //
```

```
   lcd_command(0xDF);
        buf = (d_morcy[0] & 0x0F)| 0x30;
   lcd_text(buf);
        lcd_text(' ');
        //
}
//************************** main program ************************************//

void receiver (void)
{
   unsigned char i,j;
        PCON = 0x00;
        SCON = 0x50;
        TMOD = 0x20;
        TH1 = 0xFD;
        SM2 = 0;
        TR1 = 1;

        for(i=0;i<8;i++)
        {
        for(j=0;j<8;j++)
        {
                while(~RI);
                RI = 0;
        }
        d_morcy[i]=SBUF;
        }

}
void s_clk()
{  unsigned char buf = 0;

        //
        lcd_command(0xC6);
        buf = ((d_morcy[4] & 0xF0)>>4)|(0x30);
        lcd_text(buf);
```

```
//
buf = (d_morcy[4] & 0x0F) | 0x30;
lcd_text(buf);
lcd_text(':');
//
buf = ((d_morcy[3] & 0xF0)>>4)|(0x30);
lcd_text(buf);
//
buf = (d_morcy[3] & 0x0F) | 0x30;
lcd_text(buf);
lcd_text(':');
///
buf = ((d_morcy[2] & 0xF0)>>4)|(0x30);
lcd_text(buf);
//
buf = (d_morcy[2] & 0x0F) | 0x30;
lcd_text(buf);



//****************** write hr min sec ******************************///
lcd_command(0x9A);
buf = ((d_morcy[5] & 0xF0)>>4)|(0x30);
lcd_text(buf);
//
buf = (d_morcy[5] & 0x0F) | 0x30;
lcd_text(buf);
lcd_text('/');
//
buf = ((d_morcy[6] & 0xF0)>>4)|(0x30);
lcd_text(buf);
//
buf = (d_morcy[6] & 0x0F) | 0x30;
lcd_text(buf);
lcd_text('/');
//
```

```
        buf = ((d_morcy[7] & 0xF0)>>4)|(0x30);
        lcd_text(buf);
        //
        buf = (d_morcy[7] & 0x0F) | 0x30;
        lcd_text(buf);
}
void main(void)
{
        turnon = 0;
        if(start == 0)
  {
        light = 1;
        pass_check();
        if(pass[6] == 0x0D)
                { lcd_init();
                        turnon = 1;
                        welcome();

                        while(1)
                        {
                        unsigned char key;
                bit A1 = 1;
                        A1 = 1;
                        lcd_init();
                        idle_state();
//************************* part 1 ***************************************//
      while(A1)
                {
                idle_state();
                gear_check();
                key = scankey();
                status();
                velocity();
                 revolution();
                receiver();
                oil();
```

```
            if(start == 1)
        {
            light = 0;
            lcd_init();
            while(start)
        {
            turnon = 0 ;
            receiver();
            s_clk();
        }
        }
//********************** part 2 *********************************//

        key = scankey();
        if(key == 0x0A )
            {
            cd_init();
            hile(A1)
              {     page_2();
                    while(A1)
                    {
                    page_2();
                    temperature();
                    receiver();
                    bat();
                    if(start == 1)
                            {
                            light = 0;
                            lcd_init();
                            while(start)
                                    {
                                    turnon = 0 ;
                                    receiver();
                                    s_clk();
                                    }
                            }
```

```
/*-------------------------------------------------------------------------------------------*/
        key = scankey();
                if(key == 0x0A )
                        {
                        A1 = 0;
                        }

                  }

                }
              }
//********************** end of part 2 ************************************//
        }
     }
  }

        light = 0;
        lcd_init();
        while(start)
                {
                turnon = 0 ;
                receiver();
                s_clk();
                }

}
```

ชุดคำสั่ง
สำหรับไมโครคอนทรลเลอร์ตัวรอง
(SOUCE CODE FOR SLAVE MCU)

```
// ********** SOUCE CODE FOR SLAVE MCU ********* //
#include<reg52.h>
#include<stdio.h>
#include<math.h>
#include<lcd.h>
#include<i2c.h>

unsigned char motor_data[7];
unsigned char i,j ;
sbit clk = P2^0;
sbit D_a2d = P2^1;
sbit sl_a2d = P2^2;
sbit fuel = P0^1;
sbit batt = P0^0;
sbit sw1 = P0^2;
sbit siren = P0^4;
sbit start = P1^0;
unsigned int DataA2d_f,DataA2d_b;
//*****************************************************************//
void A2ddata (void)
{
        bit d;
        unsigned char i;
        fuel = 1;
        batt = 0;
        clk = 1;
        sl_a2d = 1;
        sl_a2d = 0;
        for(i=0;i<16;i++)
                {
                clk = 0;
                d = D_a2d;
                DataA2d_f = DataA2d_f | d;
                DataA2d_f = DataA2d_f << 1;
                DataA2d_f = DataA2d_f & 0x0FFF;
                clk = 1;
```

```
                delay(0.2);
                }
                sl_a2d = 1;
        delay(1);


        fuel = 0;
        batt = 1;
        clk = 1;
        sl_a2d = 1;
        sl_a2d = 0;
        for(i=0;i<16;i++)
                {
                clk = 0;
                d = D_a2d;
                DataA2d_b = DataA2d_b | d;
                DataA2d_b = DataA2d_b << 1;
                DataA2d_b = DataA2d_b & 0x0FFF;
                clk = 1;
                delay(0.2);
                }
                sl_a2d = 1;
        delay(1);
}
//*****************************************************************************//
void oil_batt (void)
{
float  oil,batt;
unsigned char oil_1,sub_oil_1,sub_oil_2,batt_1,sub_batt_1,sub_batt_2;
oil = (DataA2d_f*5.6)/4096;
sub_oil_1= oil;
oil_1 = oil*10;
sub_oil_2= oil_1%10;
//******************//
sub_oil_1 = sub_oil_1<<4;
motor_data[1] = sub_oil_1 | sub_oil_2;
```

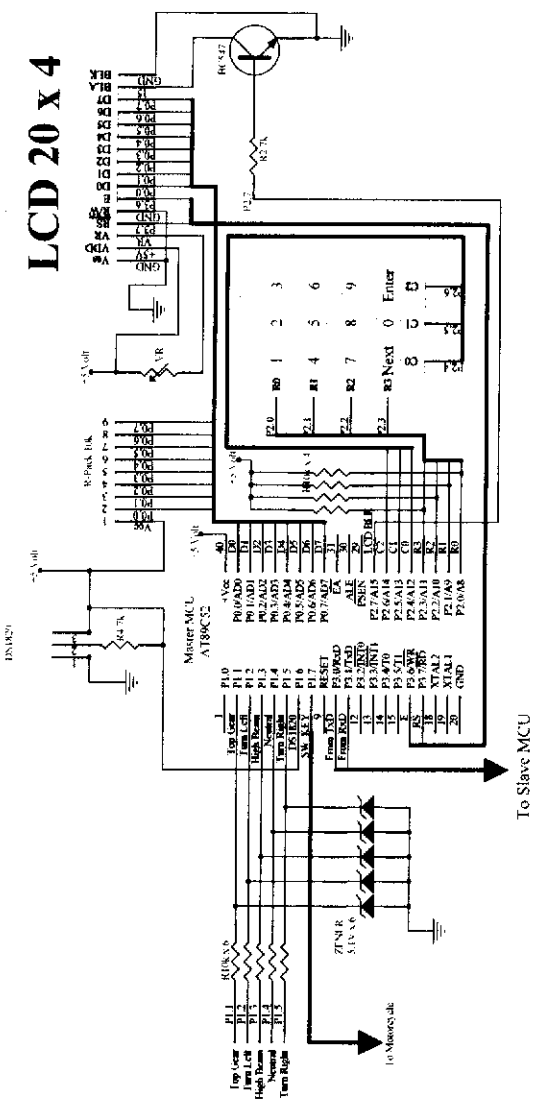
```
//*******************//
batt = (DataA2d_b*15)/4096;
sub_batt_1= batt;
batt_1 = batt*10;
sub_batt_2= batt_1%10;
//*******************//
sub_batt_1 = sub_batt_1<<4;
motor_data[0] = sub_batt_1 | sub_batt_2;
//*******************//
}
//*****************************************************************************//

#define DS1307_ID 0xD0
unsigned char sec,min,hour,day,date,month,year,control;

/********************** Function read data from DS1307 **********************************/

unsigned char DS1307_rd(unsigned char addr)
{
        unsigned char ret;
        i2c_start();
        i2c_wrdata(DS1307_ID);
        i2c_wrdata(addr);
        i2c_start();
        i2c_wrdata(DS1307_ID+1);
        ret = i2c_rddata();
        i2c_stop();
        return(ret);
}

//********************* Function write hour/min/sec on chip DS1307 ***************************/
void DS1307_wrtime(unsigned char hh,unsigned char mm,unsigned char ss)
{
        i2c_start();
        i2c_wrdata(DS1307_ID);
        i2c_wrdata(0x00);
```

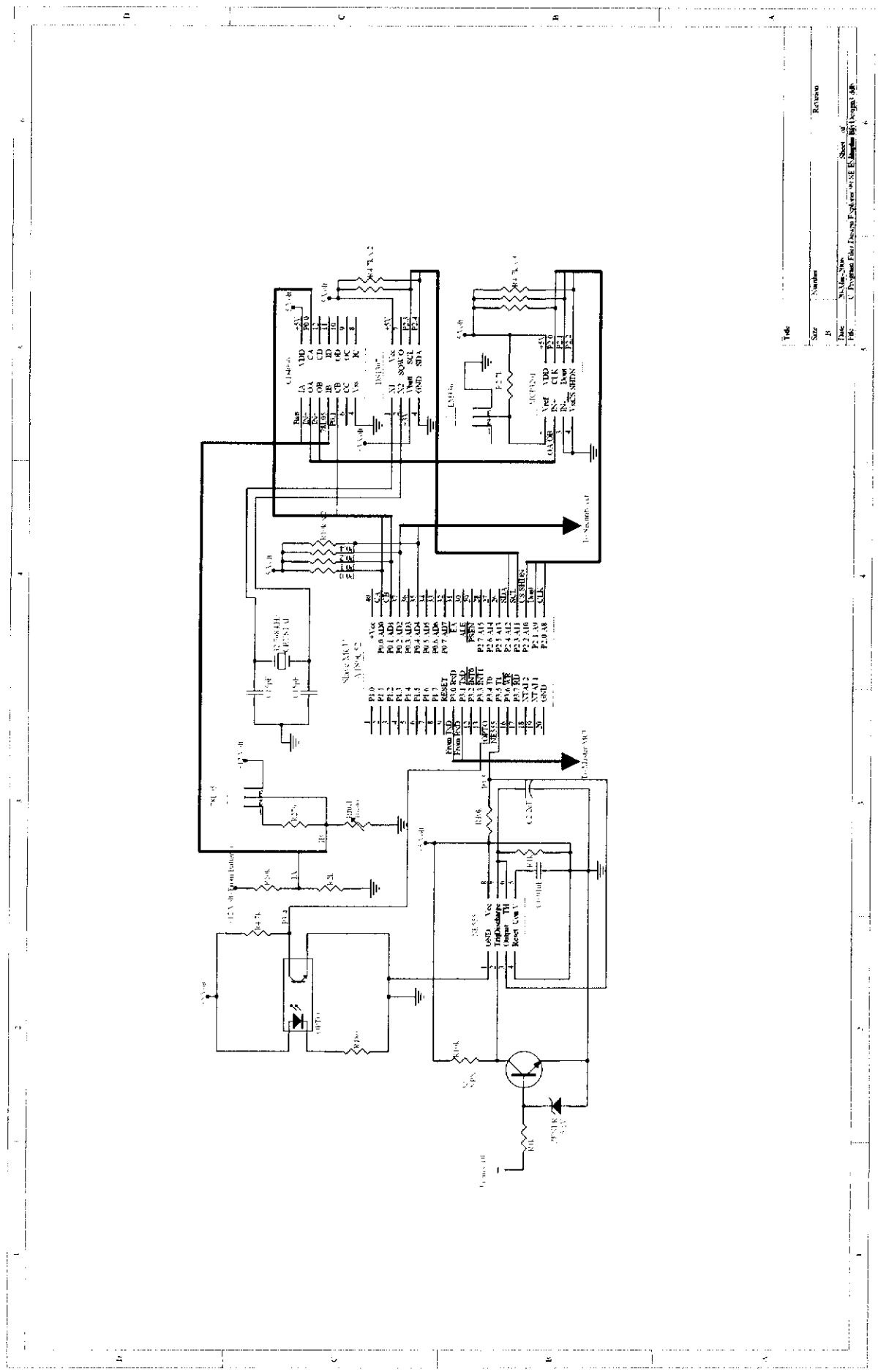
```
        i2c_wrdata(ss);
        i2c_wrdata(mm);
        i2c_wrdata(hh);
        i2c_stop();
}

void datetime(void)
{
                motor_data[2] = DS1307_rd(0x00);
                motor_data[3] = DS1307_rd(0x01);
                motor_data[4] = DS1307_rd(0x02);
                motor_data[5] = DS1307_rd(0x04);
                motor_data[6] = DS1307_rd(0x05);
                motor_data[7] = DS1307_rd(0x06);
}
//*********************************************************************//
void transmitter (void)
{
        TMOD =0x20;
        PCON = 0x00;
        SCON = 0x50;
        TH1 = 0xFD;
        SM2 = 0;
        TR1 = 1;
        for(i=0;i<8;i++)
        {
        for(j=0;j<8;j++)
        {
                SBUF = motor_data[i];
                while(~TI);
                TI = 0;
        }
        }

}
```

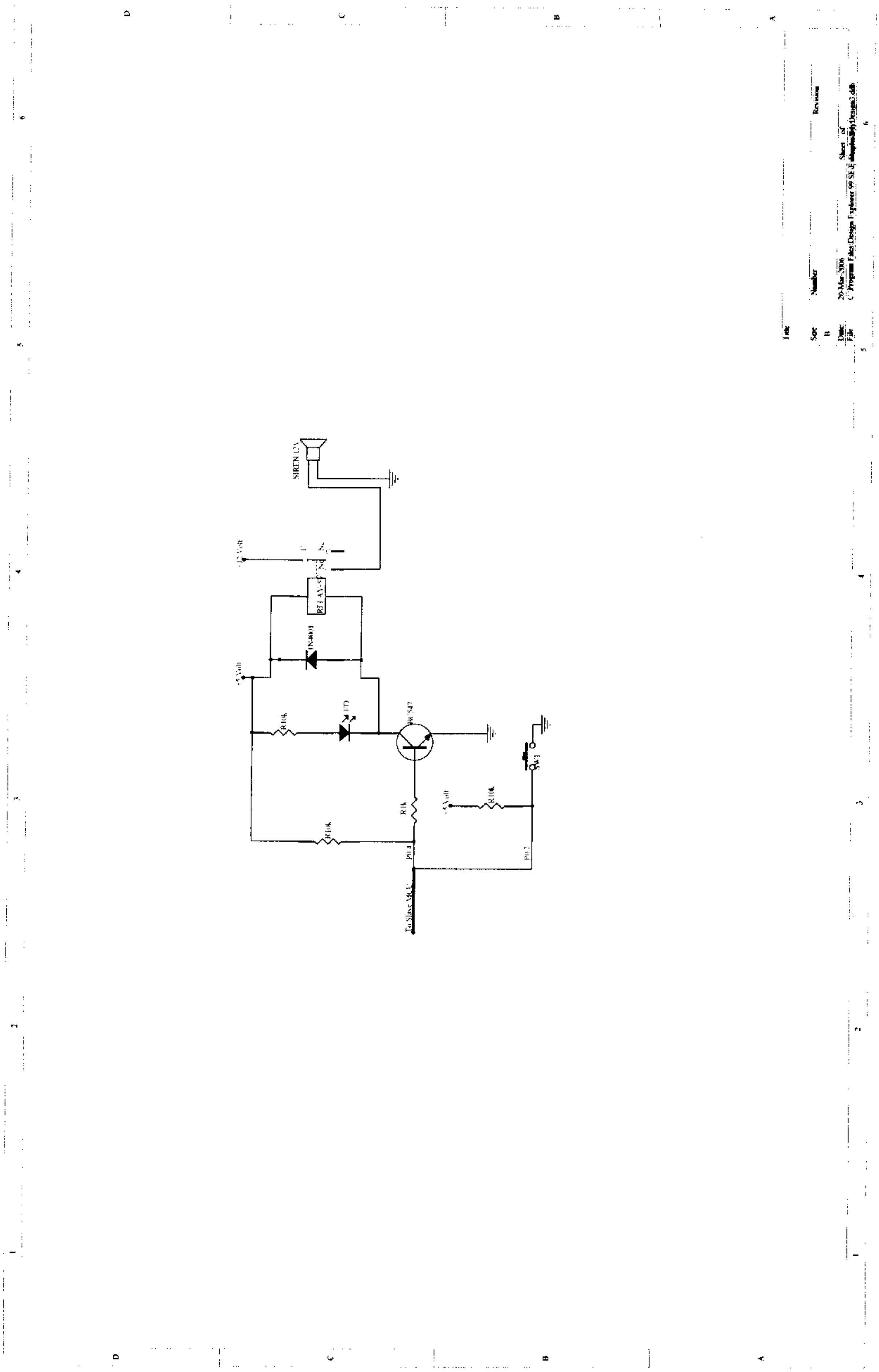
```
void security (void)
{
if(sw1 == 0)
{
siren = 1;
delay(4560);
siren = 0;
}
}

//************************************ MAIN PROGRAM ************************//
void main (void)
{
start = 1;
sw1 = 1;
siren = 0;
while(1)
{
A2ddata();
oil_batt();
datetime();
transmitter();
        if (start == 1)
        {
        security();
        }
}
}
```

ภาคผนวก ข.

วงจรเครื่องแสดงผลแบบหลายฟังก์ชัน
และระบบรักษาความปลอดภัยสำหรับรถจักรยานยนต์


LCD 20 x 4
Master MCU AT89C52
To Slave MCU
To Motorcycle
Enter
Next




SIREN 12V
RELAY-5V
1N4001
BC547
R10k
R1k
R10k
SW1
P0.4
P0.2
To Slave MCU