เครื่องควบคุมเครื่องปรับอากาศด้วยไมโครโปรเซสเซอร์พร้อมรีโมทคอนโทรลแบบไร้สาย

Microprocessor-Based Air condition Control and Wireless Remote Control

โดย

นายชัยโย นุชท่าโพ

นายพรพรหม สามสวัสดิ์

นายสุชาติ สายมาอินทร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เครื่องควบคุมเครื่องปรับอากาศด้วยไมโครโปรเซสเซอร์พร้อมรีโมทคอนโทรลแบบไร้สาย

Microprocessor-Based Air condition Control and Wireless Remote Control

โดย

นายชัยโย นุชท่าโพ 43015061

นายพรพรหม สามสวัสดิ์ 43015079

นายสุชาติ สายมาอินทร์ 43015095

อาจารย์ที่ปรึกษา

รศ.สมยศ จุณณะปิยะ

ดร.พิเชฐ ม่วงนวล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เครื่องควบคุมเครื่องปรับอากาศด้วยไมโครโปรเซสเซอร์พร้อมรีโมทคอนโทรลแบบไร้สาย

Microprocessor-Based Air condition Control and Wireless Remote Control

โดย นายชัยโย นุชท่าโพ 43015061
นายพรพรหม สามสวัสดิ์ 43015079
นายสุชาติ สายมาอินทร์ 43015095

อาจารย์ที่ปรึกษา รศ.สมยศ จุณณะปิยะ
ดร.พิเชฐ ม่วงนวล

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการศึกษาโครงสร้าง และการใช้งานชิพไมโครคอนโทรลเลอร์ตระกูล MCS-51 และการนำมาประยุกต์ใช้ควบคุมระบบการทำงานของเครื่องปรับอากาศ มาควบคุมเครื่องปรับอากาศได้ โดยที่ควบคุมการเปิด-ปิดคอมเพรสเซอร์ และควบคุมความเร็วของพัดลมได้ มี 3 ระดับ คือ high speed, medium speed และ low speed อุปกรณ์การตัดต่อคอมเพรสเซอร์และพัดลมจะใช้ Solid-State Relay แทน Relay เพื่อแก้ปัญหาของหน้าสัมผัส รวมทั้งมีตัวควบคุมระยะไกล (Remote Control) แบบไร้สาย (ใช้อินฟราเรด) ระยะทางได้ 5-8 เมตร แสดงผลผ่านจอ LCD

ABSTRACT

This thesis is purpose to learning structure and application of "MCS-51" series single chip microcontroller. This application use microcontroller for control start / stop compressor in temperature range and control speed of fan determines current temperature and setting temp. Fan's speed has three levels high, medium and low. The power switching device use solid-state relay instead of magnetic mechanical relay is it has no contact spike. The special of this controller is it can control operation by infrared wireless remote control, in rang of distance 5-8 metric. Temperature is display by LCD MODULE.

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **เครื่องควบคุมเครื่องปรับอากาศด้วยไมโครโปรเซสเซอร์ พร้อมรีโมทคอนโทรลแบบไร้สาย**

Microprocessor-Based Air condition Control and Wireless Remote Control

ผู้จัดทำ

1.	**นายชัยโย**	**นุชท่าโพ**	**43015061**
2.	**นายพรพรหม**	**สามสวัสดิ์**	**43015079**
3.	**นายสุชาติ**	**สายมาอินทร์**	**43015095**

..... อาจารย์ที่ปรึกษา

(รศ.สมยศ จุณณะปิยะ)

..... อาจารย์ที่ปรึกษา

(ดร.พิเชฐ ม่วงนวล)

สารบัญ

สารบัญรูปภาพ

เรื่อง หน้า

บทที่ 2 ทฤษฎีและหลักการ

สารบัญรูปภาพ (ต่อ)

เรื่อง หน้า

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันความเจริญก้าวหน้าทางด้านเทคโนโลยีได้ถูกพัฒนาไปอย่างรวดเร็ว และมีแนวโน้มที่จะพัฒนาก้าวหน้ายิ่งขึ้นอีกอย่างรวดเร็ว ในส่วนของศาสตร์วิศวกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ เราจะมองเห็นได้อย่างชัดเจน เพราะเป็นส่วนใกล้ตัวมนุษย์ปัจจุบันมาก ซึ่งเกี่ยวข้องกับตัวเร่า และบางครั้งไม่อาจสามารถแยกออกได้ โดยมนุษย์ต้องการจัดหาทรัพยากรเพื่อมาตอบสนองแก่ตนเอง เพื่อให้ความเป็นอยู่สะดวกสบายยิ่งขึ้นมากที่สุด แต่การค้นคว้าวิจัยทรัพยากรดังกล่าวจะต้องใช้ความสามารถ สมองสติปัญญาและความอดทน บางครั้งอาจจะประสบผลสำเร็จ แต่บางครั้งอาจประสบความล้มเหลวได้ ฉะนั้นการทำงานทางด้านการค้นคว้าวิจัยพัฒนาจะต้องตั้งมั่นอยู่บนความอดทนเพื่อความสำเร็จของโครงการการศึกษาทางด้านไมโครคอนโทลเลอร์ จึงจำเป็นอย่างยิ่ง และมีแนวโน้มที่จะพัฒนาก้าวหน้ายิ่งขึ้นไปอีก ในทางเทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์ การพัฒนาไมโครคอนโทรลเลอร์ของผู้ผลิตไอซี ทางบริษัทผู้ผลิตต้องทำการค้นคว้าเพิ่มประสิทธิภาพของชิพที่ตนผลิต เพื่อทำการแข่งขันในตลาดโลกได้ เราจะเห็นได้ว่าในปัจจุบันไมโครคอนโทลเลอร์มีประสิทธิภาพและความสามารถสูง สามารถนำมาประยุกต์ใช้งานได้อย่างกว้างขวาง อุปกรณ์เครื่องใช้ไฟฟ้าและอิเล็กทรอนิกส์ที่มีฟังก์ชั่นที่ซับซ้อนนั้นจะมีอุปกรณ์ที่เป็นหัวใจหลักของระบบคือ CPU ส่วนระบบของ CPU นั้นจะรับค่าอินพุตเข้ามาเพื่อประเมินผลและให้ค่าเอาต์พุตต่าง ๆ ออกไปควบคุมระบบตามความต้องการของผู้ออกแบบ จะเห็นได้อย่างชัดเจนว่าตัว CPU คล้ายกับเป็นตัวแทนมันสมองของมนุษย์ แต่ถ้ามันสมองยังเล็กและแคบก็สามารถทำงานได้แค่ระดับหนึ่ง ดังนั้นจึงมีการพัฒนาชิพไมโครคอนโทลเลอร์อยู่ตลอดเวลา จะเห็นได้ว่าชิพไมโครคอนโทลเลอร์จะมีอยู่หลายเบอร์หลายตระกูลด้วยกัน ในโครงการนี้เป็นการประยุกต์ใช้งานของไมโครคอนโทลเลอร์ในการควบคุมเครื่องปรับอากาศ (Air condition) โดยมองความเหมาะสมของตัวไมโครคอนโทลเลอร์กับระบบที่ควบคุมและกับการที่ได้ศึกษาไมโครคอนโทลเลอร์ตระกูล MCS-51 ของ Intel จึงเลือกใช้เบอร์ 89C51

1.1 วัตถุประสงค์ของโครงงาน

1.1.1 เพื่อศึกษาถึงการทำงานของระบบแอร์คอนดิชั่นและรีโมทคอนโทรล

1.1.2 เพื่อให้ระบบแอร์คอนดิชั่นแบบเก่าสามารถควบคุมจากระยะไกลได้

1.1.3 เพื่อเพิ่มความสะดวกและทันสมัยด้านเทคโนโลยี

1.1.4 เป็นการนำเอาความรู้ด้านไมโครคอนโทรลเลอร์ประยุกต์ใช้ให้เกิดประโยชน์ในชีวิตประจำวัน

1.2 ประโยชน์ที่คาดว่าจะได้รับ

1.2.1 สามารถควบคุมค่าอุณหภูมิของแอร์คอนดิชั่นได้ดี

1.2.2 เพิ่มความสะดวกสบายให้ผู้ใช้งาน

1.2.3 ผู้ศึกษาได้ประยุกต์ใช้งานในวิชาที่เรียนมาให้เกิดประโยชน์

1.3 ขอบเขตของโครงงาน

1.3.1 สามารถควบคุมได้ในระยะประมาณ 5-8 เมตร

1.3.2 สามารถควบคุมอุณหภูมิให้ สูงขึ้น / ลดลง ได้ และปรับความแรงของพัดลมได้ 3 ระดับ

1.3.3 มีจอ LCD ในการแสดงผลทั้งอุณหภูมิและเวลา

1.3.4 สามารถปรับมุมของใบกระจายความเย็นได้

1.4 ขั้นตอนการดำเนินงาน

1.4.1 ศึกษาทฤษฎีที่เกี่ยวข้องกับโครงงาน

1.4.2 ออกแบบวงจร

1.4.3 จัดหาเครื่องมือและอุปกรณ์

1.4.4 สร้างวงจรต่าง ๆ แยกเป็นส่วน ๆ

1.4.5 ทดลองการทำงานของวงจรแต่ละส่วน

1.4.6 ประกอบวงจรทั้งหมดเข้าด้วยกัน

1.4.7 เขียนโปรแกรมควบคุมการทำงานของระบบ

1.4.8 ทดลองการทำงาน

1.4.9 แก้ไขข้อบกพร่อง

1.4.10 สรุปผลการทดลอง

บทที่ 2
ทฤษฎีและหลักการ

2.1 พื้นฐานหลักการทำงานของระบบ Air condition

2.1.1 ชนิดของระบบเครื่องปรับอากาศ (Air condition System) เครื่องปรับอากาศที่นิยมใช้แบ่งออกตามชนิดต่าง ๆ ดังนี้

1. แบบกล่อง (Package Unit) เป็นเครื่องปรับอากาศที่อุปกรณ์ต่าง ๆ รวมอยู่ในกล่อง หรือ Package เดียวกัน ทั้งคอนเด็นซิ่งยูนิต และคูลลิ่งยูนิต แยกได้เป็น 2 ชนิดคือ

1.1 แบบติดหน้าต่าง (Window Type) เป็นเครื่องปรับอากาศขนาดเล็กตั้งแต่ประมาณ 6,000 BTU./hr ถึง 30,000 BTU./hr (1 ตัน = 12,000 BTU./hr) ชนิดนี้ง่ายต่อการติดตั้ง แต่มีข้อเสียคือ จะมีเสียงดังและเกิดความรำคาญต่อผู้ใช้

1.2 แบบวางตั้งบนพื้น (Big Package Type) เป็นเครื่องปรับอากาศขนาดใหญ่ ตั้งแต่ 2-15 ตัน เป็นลักษณะที่ต้องวางนอกตัวอาคาร และติดต่อ Duct เข้าไปในห้อง

2. แบบแยกระบบ (Split System) เป็นระบบที่แยกเอาระบบ Condensing Unit ออกไว้ภายนอกอาคาร และเอาระบบความเย็น (Evaporating Unit) และพัดลมไว้ภายในตัวอาคาร ระบบนี้จึงแยกส่วนเป็น 2 ส่วน คือ

2.1 Condensing Unit จะประกอบด้วย Compressor, Condenser และ Condensing Fan ส่วนนี้จะติดตั้งอยู่ภายนอกอาคาร

2.2 Evaporating Unit จะประกอบด้วย Evaporator และพัดลม หรือเราเรียกว่า Cooling Unit จะติดตั้งในห้องที่ต้องการความเย็น

3. แบบเป่าตรง (Direct Expansion system) เป็นเครื่องปรับอากาศขนาดใหญ่ ตั้งแต่ 30 ตัน ขึ้นไป ส่วนมากใช้ Condenser แบบระบายความร้อนด้วยน้ำ (Water Cooled Condenser)

4. แบบชีลเลอร์ (Chiller Water System) เป็นระบบที่ใช้น้ำผ่านเข้าไปใน Evaporator เพื่อให้น้ำมีอุณหภูมิต่ำประมาณ 42-50 F แล้วเอาน้ำเย็นนี้ส่งผ่านท่อออกไปยังห้องที่ต้องการความเย็น แล้วเอาน้ำเย็นนี้ผ่านท่อขดเย็น แล้วใช้พัดลมเป่าให้อากาศภายในห้องผ่านท่อขดเย็นเล็กนี้อีกครั้งหนึ่ง

จากชนิดต่าง ๆ ซึ่งจะถูกแบ่งตามลักษณะของระบบและขนาด (BTU./hr) ซึ่งส่วนใหญ่ขนาดหรือแบบที่นิยมใช้กันมาก คือ แบบ Split Type ซึ่งแบบนี้นิยมเรียกได้อื่น ๆ อีกตามลักษณะระบบการติดตั้ง เช่น ติดข้างฝา , ตั้งพื้น และ แบบขนาน ซึ่งชุดควบคุมของโครงงานนี้จะเหมาะกับชนิดที่กล่าวมา

2.1.2 ส่วนประกอบของ Air condition System

1. คอนเดนซิ่ง ยูนิต (Condensing Unit) หรือ เอ้าดอร์ ยูนิต (Out Door Unit) ประกอบด้วย

- คอมเพรสเซอร์ มอเตอร์ เป็นหัวใจสำคัญของระบบเครื่องปรับอากาศ มีหน้าที่ดูดน้ำยาที่เป็นแก๊สแรงดันต่ำ และอัดให้มีแรงดันสูง

- แฟนมอเตอร์ (Fan Motor) มีหน้าที่ระบายความร้อนของแผงคอยล์เพื่อลดปริมาณความร้อนของ High Presser

- คาร์ปาซิเตอร์ (Capaciter) จะอยู่กับทั้งวงจร แฟน มอเตอร์ และ คอมเพรสเซอร์ มอเตอร์จะต่ออนุกรมกับขดสตาร์ทของมอเตอร์

- อินเทอร์นอล โอเวอร์โหลด โปรเทกเทอร์ จะมีอยู่ภายในของ แฟนมอเตอร์ และ คอมเพรสเซอร์ มอเตอร์ เพื่อป้องกันเกิดการลัดวงจรของขดลวดภายในมอเตอร์

- คอนเดนเซอร์ ส่วนที่ต้องการระบบความร้อน เพื่อลดปริมาณความร้อนลงและทำการเปลี่ยนสถานะน้ำยา ซึ่งเป็นแก๊สของเหลวที่ถูก คอมเพรสเซอร์ มอเตอร์ อัดส่งทำให้น้ำยาที่ออกจากคอนเดนเซอร์เป็นของเหลว

2. คูลลิ่ง ยูนิต (Cooling Unit) หรือเรียกว่า อินดอร์ ยูนิต เป็นส่วนที่อยู่ในห้องหรือในอาคารที่เราควบคุมอุณหภูมิ ประกอบด้วย

- อีวาพอเรเตอร์ (Evaporator) ทำหน้าที่ให้ความเย็น ซึ่งน้ำยาภายในท่อส่วนนี้ถูกเปลี่ยนสถานะจากของเหลวเป็นแก๊ส

- เอกซ์แพนชั่น วาล์ว เป็นตัวควบคุมน้ำยา และขณะเดียวกันน้ำยาภายในท่อถูกเปลี่ยนสถานะตรงจุดนี้ด้วยจากของเหลวเป็นแก๊ส

- แฟน มอเตอร์ จะเป็นตัวเป่าระบายความเย็นที่อีวาพอร์เรเตอร์ เพื่อให้ความเย็นกระจายออกมา

- เทอร์โมสตาร์ท เป็นตัวตัดต่อให้วงจรคอมเพรสเซอร์ มอเตอร์ ทำงานหรือหยุด เมื่ออุณหภูมิภายในห้องเย็นถึงค่าที่สั่งไว้

2.1.3 การทำงานของเครื่องปรับอากาศ

การทำงานเบื้องต้นของเครื่องปรับอากาศ เริ่มต้นเมื่อจ่ายไฟ 220 โวลท์ ให้กับคอมเพรสเซอร์ทำงานดูดน้ำยาที่เป็นแก๊ส และอัดส่งทางออก (ภายในท่อ) ซึ่งมีอุณหภูมิและแรงดันสูงผ่านคอนเดนเซอร์ซึ่งจะมีลมพัดระบายคอนเดนเซอร์ ทำให้น้ำยาที่สถานะเป็นแก๊สกลายเป็นน้ำยาที่เปลี่ยนสถานะเป็นของเหลว (ซึ่งใช้หลักการของการควบแน่น) ส่งผ่านไปเข้าเอ็กแปนชั่นวาล์ว น้ำยาจะลดแรงดันลงฉีดเข้าไปในอีแวปพอร์เรเตอร์ (ใช้หลักการของความร้อนแฝง) เมื่ออีแวปพอเรเตอร์เย็นจะมีพัดลมพัดกระทบผ่านอีแวปพอเรเตอร์ให้ความเย็นกระจายออก

จากข้างต้นที่อธิบายมานั้นจะเห็นว่าเราจะควบคุมระบบการทำงานได้ โดยควบคุมการทำงาน ON/OFF คอมเพรสเซอร์และปรับความเร็วของมอเตอร์ที่ระบายความเย็นที่กล่าวมาเป็นหลักใหญ่ ๆ ของจุดที่จะควบคุม

2.2. รายละเอียดเกี่ยวกับ LCD Module

2.2.1 ส่วนประกอบหลักของ LCD Module

1. ตัวแสดงผล (Display) ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็นโดยอาศัยแสงจากภายนอก ดังนั้นจึงต้องมีมุมมองข้อมูลที่แสดงผลบนจอ

2. ตัวควบคุม (Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาควบคุมการทำงานของ LCD Module เช่น ลบจอภาพ แสดงตัวอักษร หรือเลื่อนเคอร์เซอร์ เป็นต้น ตัวควบคุมนี้ใช้ชิปควบคุม โดยเฉพาะชิปที่นิยมใช้ คือ เบอร์ HD44780 และ HD61830 โดย HD44780 จะใช้ควบคุม LCD แบบอักขระ ส่วน HD61830 ใช้ควบคุม LCD แบบกราฟิก

3. ตัวขับ (Driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงข้อมูลตามที่กำหนด ชิปที่ใช้ทำหน้าที่นี้ได้แก่เบอร์ HD44100H และ MSM5259 เป็นต้น

LCD Module มีอยู่หลายรุ่น และคุณสมบัติแตกต่างกันไป ซึ่งแบ่งได้เป็น 2 แบบคือ แบบ Dot matrix และ Graphic โดยแบบ Dot matrix จะแสดงผลเป็นแบบ 5x7 Dot. หรือ 5x10 Dot. ส่วนใหญ่จะเป็นแบบ 5x10 Dot. มีตั้งแต่ 1 Line, 2 Line และ 4 Line ซึ่งการใช้งานแต่ละแบบจะใกล้เคียงกัน

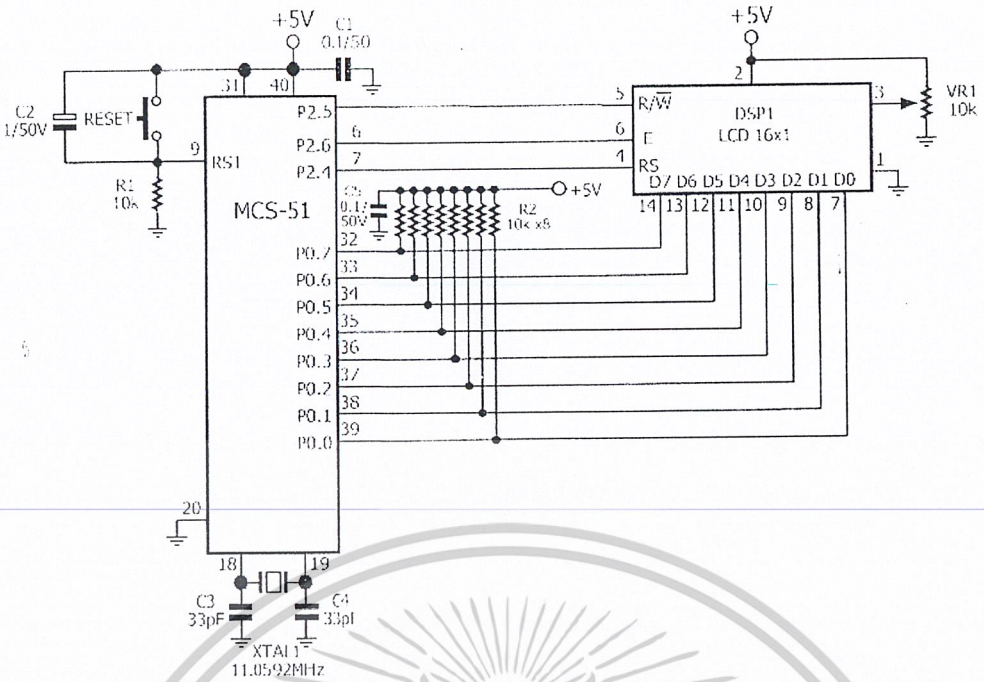
ตัว LCD Module จะมีขาใช้งานทั้งหมด 14 ขา ด้วยกัน หน้าที่ของแต่ละขามีดังนี้คือ

- ขา 1 (GND) เป็น Ground ใช้ต่อกับระบบ Ground ของไมโครคอนโทรลเลอร์
- ขา 2 (VCC) เป็นไฟเลี้ยงวงจรของ LCD มีขนาด +5 VDC
- ขา 3 (Vee) เป็นขาสำหรับปรับความเข้มของจอ LCD โดยที่เมื่อต่อกับ VCC จะมีความเข้มต่ำสุด และเมื่อต่อกับ Ground จะมีความเข้มมากที่สุด โดยปกติจะต่ออยู่กับ Ground เสมอเพื่อความสะดวกในการต่อ
- ขา 4 (RS) ใช้สำหรับบอก LCD ทราบว่าข้อมูลที่ส่งให้นั้นเป็น Instruction หรือ Data โดยเมื่อขานี้ เป็น “0” หมายถึง Instruction เป็น “1” หมายถึง Data
- ขา 5 (R/W) ใช้สำหรับกำหนดว่าเป็นการอ่านหรือเขียนข้อมูลให้กับ LCD โดยเมื่อขานี้ เป็น “0” หมายถึงเป็นการเขียนข้อมูล เป็น “1” หมายถึงเป็นการอ่านข้อมูล
- ขา 6 (E) เป็นขา Enable ขานี้ เป็น “1” ใช้สำหรับบอก LCD ว่าอุปกรณ์ภายนอกต้องการติดต่อด้วย เป็น “0” ตัว LCD จะไม่สนใจในสัญญาณ RS, R/W และ (DB0-DB7) เลย
- ขา 7-14 (DB0-DB7) เป็นขา Data Bus สำหรับอ่านหรือเขียนข้อมูลให้กับตัว LCD

2.2.2 การเชื่อมต่อ LCD Module เข้ากับไมโครคอนโทรลเลอร์

การเชื่อมต่อ LCD Module เข้ากับไมโครคอนโทรลเลอร์สามารถต่อได้โดยตรงกับตัว MCS-51 หรือต่อผ่าน 8255 ก็ได้ ในที่นี้จะต่อโดยผ่าน MCS-51 ดังรูป

- ขาสัญญาณข้อมูล D0-D7 (ขา 32-39) ต่อเข้ากับ MCS-51 พอร์ต 0
- ขา RS (ขา 25) ต่อเข้ากับ MCS-51 พอร์ต 2 บิต 4
- ขา R/W (ขา 26) ต่อเข้ากับ MCS-51 พอร์ต 2 บิต 5
- ขา E (ขา 27) ต่อเข้ากับ MCS-51 พอร์ต 2 บิต 6



รูปที่ 2.1 แสดงการเชื่อมต่อ LCD Module เข้ากับ MCS-51

2.2.3 ชุดคำสั่งของ LCD Module

INSTRUCTION	RS	R/W	DATA BIT								EXECUTE TIME (nS)
			7	6	5	4	3	2	1	0	
CLEAR DISPLAY	0	0	0	0	0	0	0	0	0	1	1640
CURSOR AT HOME	0	0	0	0	0	0	0	0	1	*	1640
ENTRY MODE SET	0	0	0	0	0	0	0	1	I/D	S	40
DISPLAY ON/OFF	0	0	0	0	0	0	1	D	C	B	40
DISPLAY SHIFT	0	0	0	0	0	1	S/C	R/L	*	*	40
FUNCTION SET	0	0	0	0	1	DL	N	F	*	*	40
SET CGRAM ADDR.	0	0	0	1	CGRAM ADDRESS						40
SET DDRAM ADDR.	0	0	1	DDRAM ADDRESS							40
BUSY, ADDR.READ	0	1	BF	ADDRESS							0
CGRAM, DDRAM WR	1	0	WRITE DATA								40
CGRAM, DDRAM RD	1	1	READ DATA								40

ตารางที่ 2.1 แสดงชุดคำสั่งและเวลาที่ LCD Module ใช้ในการทำงานแต่ละคำสั่ง

1. คำสั่งเคลียร์ตัวแสดงผล (CLEAR DISPLAY)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

คำสั่ง CLEAR DISPLAY เป็นคำสั่งที่ใช้เขียนข้อมูลหรือตัวอักษรว่าง (Space) ลงใน DDRAM ทั้งหมด และทำการกำหนดค่า DDRAM Address เป็น 0 และเคอร์เซอร์จะกลับไปอยู่ที่ตำแหน่งบนซ้ายสุดของจอแสดงผล

2. คำสั่ง CURSOR AT HOME

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	*

คำสั่ง CURSOR AT HOME หรือ RETURN HOME เป็นคำสั่งที่ใช้ในการเลื่อนตำแหน่งของเคอร์เซอร์ไปอยู่ที่ตำแหน่งบนซ้ายสุดของจอแสดงผล โดยข้อมูลที่อยู่ใน DDRAM หรือที่หน้าจอแสดงผลจะไม่เปลี่ยนแปลง

3. คำสั่งโหมดในการป้อนข้อมูล (ENTRY MODE SET)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

คำสั่งโหมดในการป้อนข้อมูล (ENTRY MODE SET) ใช้สำหรับกำหนดการเลื่อนเคอร์เซอร์และตำแหน่งแอดเดรสของ DDRAM ดังนี้

- I/D เป็นบิตที่ใช้ในการกำหนดการเลื่อนเคอร์เซอร์และตำแหน่งแอดเดรสของ DDRAM ว่าจะให้เพิ่มหรือลดลงเมื่อเขียนหรืออ่านข้อมูลแล้ว

บิต I/D = 0 แอดเดรสของ DDRAM จะลดลง

บิต I/D = 1 แอดเดรสของ DDRAM จะเพิ่มขึ้น ส่วนเคอร์เซอร์จะเลื่อนตามตำแหน่งแอดเดรสของ DDRAM

- S เป็นบิตที่ใช้กำหนดลักษณะของการแสดงผลเมื่อมีการเขียนข้อมูล

บิต S = 1 เมื่อเขียนข้อมูลใหม่ลงไปแล้วตัวเคอร์เซอร์จะอยู่กับที่แต่ตัวอักษรข้อมูลเดิมจะถูกผลักไปทางซ้าย

บิต S = 0 เมื่อเขียนข้อมูลใหม่ลงไปแล้วตัวเคอร์เซอร์จะเลื่อนไปทางขวามือ

4. คำสั่งควบคุมการแสดงผล (DISPLAY ON/OFF)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

คำสั่งควบคุมการแสดงผล เป็นคำสั่งที่ใช้ในการเปิดปิดจอแสดงผลและเคอร์เซอร์มีลักษณะดังนี้

D = 0 กำหนดให้ปิดจอแสดงผล (Display OFF)

D = 1 กำหนดให้เปิดจอแสดงผล (Display ON)

C = 0 กำหนดให้ปิดเคอร์เซอร์ (Cursor OFF)

C = 1 กำหนดให้เปิดเคอร์เซอร์ (Cursor ON)

B = 0 กำหนดให้ไม่มีการกระพริบที่เคอร์เซอร์

B = 1 กำหนดให้มีการกระพริบที่เคอร์เซอร์ (กระพริบเป็นรูปสี่เหลี่ยมทึบ)

5. คำสั่งควบคุมการเลื่อนเคอร์เซอร์และตัวอักษร (DISPLAY SHIFT)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	*	*

คำสั่งควบคุมการเลื่อนเคอร์เซอร์และตัวอักษร เป็นการควบคุมการเลื่อนของเคอร์เซอร์และตัวอักษรบนจอแสดงผล โดยขึ้นอยู่กับการกำหนดบิต S/C และ R/L โดยมีลักษณะดังนี้

S/C	R/L	ลักษณะการเลื่อน
0	0	เลื่อนเคอร์เซอร์ไปทางซ้าย
0	1	เลื่อนเคอร์เซอร์ไปทางขวา
1	0	เลื่อนตัวอักษรตัวใหม่ไปทางซ้าย
1	1	เลื่อนตัวอักษรตัวใหม่ไปทางขวา

6. คำสั่งการกำหนดฟังก์ชั่นการทำงาน (FUNCTION SET)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	*	*

DL = 0 กำหนดให้ติดต่อกับ LCD Module เป็นแบบ 4 บิต

DL = 1 กำหนดให้ติดต่อกับ LCD Module เป็นแบบ 8 บิต

N = 0 กำหนดการแสดงผลแบบ 1 บรรทัด

N = 1 กำหนดการแสดงผลตั้งแต่ 2 บรรทัดขึ้นไป

F = 0 กำหนดความละเอียดของการแสดงผลเป็น 5x7 Dot.

F = 1 กำหนดความละเอียดของการแสดงผลเป็น 5x10 Dot.

หมายเหตุ

1. LCD Module แบบ 16 ตัวอักษร 1 บรรทัด แม้จะมีการแสดงผลเพียง 1 บรรทัด แต่จะต้องกำหนด N = 1 เนื่องจากแอดเดรสของ DDRAM จะแบ่งออกเป็น 2 ช่วง คือ 8 ตัวอักษรแรกจะเริ่มที่ 00H และอีก 8 ตัว อักษรถัดไปจะเริ่มที่ 40H

2. การกำหนดบิต F สำหรับ LCD Module แบบ 5x7 Dot. จะไม่มีผลอะไรเกิดขึ้น

3. เนื่องจากการกำหนดค่า DL สามารถกระทำได้ที่บิต (DB4-DB7) ถ้ามีการกำหนดให้เป็นแบบ 4 บิต ตั้งแต่ครั้งแรก หลังจากจ่ายไฟเลี้ยงให้กับตัว LCD Module แล้วก็จะทำให้เป็นการติดต่อกับ LCD Module เป็นแบบ 4 บิตทันที

7. คำสั่งเลือกแอดเดรสของ CGRAM (SET CGRAM ADDRESS)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	CGRAM ADDRESS					

คำสั่งนี้ใช้สำหรับกำหนดตำแหน่ง Address ของ Character Generator หรือ CGRAM โดยจะต้องกำหนดค่านี้ทุกครั้งในการเขียนหรืออ่านข้อมูลกับ CGRAM ซึ่งกำหนดที่ (DB0-DB5) ส่วน DB6 ต้องเป็น “1” และ DB7 ต้องเป็น “0” (01XX XXXXB) ซึ่งก็คือ (40H-7FFH)

8. คำสั่งเลือกแอดเดรสของ DDRAM (SET DDRAM ADDRESS)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	DDRAM ADDRESS						

คำสั่งนี้ใช้สำหรับกำหนดตำแหน่ง Address ของ Display Data RAM หรือ DDRAM หรือตำแหน่งของเคอร์เซอร์สำหรับการแสดงผลทางหน้าจอ LCD ซึ่งเมื่อมีการอ่านหรือเขียนค่าตัวอักษรให้กับ LCD ในแต่ละครั้งนั้น ค่าตำแหน่งของ DDRAM Address จะเพิ่มขึ้นหรือลดลง 1 ตำแหน่งโดยอัตโนมัติเสมอ ซึ่งจะเพิ่มหรือลดนั้นกำหนดได้จากบิต I/D ใน ENTRY MODE SET แต่เราก็สามารถที่จะกำหนดตำแหน่งแอดเดรสของ DDRAM ณ ตำแหน่งใด ๆ ก็ได้บนจอ LCD ที่เราต้องการให้แสดงผล ณ จุดนั้น ๆ ได้เอง โดยกำหนดแอดเดรสนี้ก่อนที่จะทำการอ่านหรือเขียนตัวอักษรให้กับ DDRAM Address เสมอ สำหรับแอดเดรสของ DDRAM ของ LCD แต่ละแบบนั้น แสดงดังตารางข้างล่างนี้ ซึ่งจะกำหนดให้บิต DB7 เท่ากับ “1” เสมอ เพื่อความสะดวกในการใช้งาน

8.1 แบบ 16 ตัวอักษร 1 บรรทัด

80	81	82	83	84	85	86	87	C0	C1	C2	C3	C4	C5	C6	C7

8.2 แบบ 16 ตัวอักษร 2 บรรทัด

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

8.3 แบบ 16 ตัวอักษร 4 บรรทัด

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF

8.4 แบบ 20 ตัวอักษร 2 บรรทัด

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	95	93
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3

โดยที่ (1000 0000B = 80H)

(1001 0000B = 90H)

(1100 0000B = C0H)

(1101 0000B = D0H)

9. คำสั่งอ่านแฟลก BUSY และ Address (Read Busy Flag & Address)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	BF	CGRAM/DDRAM ADDRESS						

คำสั่งนี้ใช้สำหรับการอ่านค่าของ Busy Flag (BF) ซึ่งบอกถึงความพร้อมของ LCD ในการรับข้อมูล

ถ้า BF = 0 หมายถึง LCD พร้อมที่จะรับข้อมูลต่อไปได้

ถ้า BF = 1 หมายถึงว่ายังไม่พร้อมที่จะรับข้อมูล

นอกจากนี้แล้วทุกครั้งที่อ่านค่าแฟลก BF เข้ามาแล้วก็จะได้ตำแหน่งของ CGRAM หรือ DDRAM Address ด้วย ในตำแหน่งของ (DB0-DB6)

2.2.4 ตัวอักษรที่ใช้กับ LCD Module

รูปที่ 2.2 แสดงรหัสตัวอักษรที่ใช้กับ LCD Module

สำหรับตัวอักษรที่ใช้ในการแสดงผลของ LCD Module นั้น จะเป็นรหัส ASCII ที่ใช้กันทั่ว ๆ ไป ดังแดงในรูป ตัวอย่างเช่นต้องการให้แสดงเป็นตัวอักษร “A” จะต้องส่งค่า 41H ออกไปให้ที่ตัว LCD หรือ “a” ก็จะต้องส่งค่า 61H ออกไป ส่วนค่า 00H จนถึง 07H หรือ 08H จนถึง 0FH (บิตที่ 4 นั้น จะเป็น “0” หรือ “1” ก็จะได้ตำแหน่งเดียวกัน) นั้นเป็นรหัสตัวอักษรที่เป็นการนำตัวอักษรที่โหลดเอง (CGRAM) ออกไปแสดงผล

2.2.5 การเขียนข้อมูลให้กับ DDRAM ของ LCD Module

1. ในการเขียนและอ่านข้อมูลให้กับ LCD Module จะมี 2 ลักษณะ คือ Instruction หรือ รหัสคำสั่งควบคุม และ Data หรือข้อมูลที่จะให้แสดงผลจะถูกกำหนดโดยขาสัญญาณ RS

RS = 0 หมายถึง Instruction หรืออ่านค่าของแฟลกสถานะการทำงานของ LCD Module

RS = 1 หมายถึง การเขียนหรืออ่านข้อมูลตัวอักษรกับตัว LCD Module

การเขียนข้อมูลตัวอักษรให้กับ LCD จะต้องให้ R/W เป็น “0”

การอ่านข้อมูลจากตัว LCD จะต้องให้ R/W เป็น “1”

2. เมื่อเขียนข้อมูลให้กับ LCD Module ลงไปแล้ว ตัว LCD จะต้องใช้เวลาในการทำงานภายในตัว LCD เองชั่วเวลาหนึ่ง ซึ่งในการเขียนโปรแกรมสามารถตรวจสอบได้ โดยเช็คสถานะได้จาก Busy Flag (BF) ถ้าเรียบร้อยแล้วจึงสามารถส่งตัวอักษรตัวต่อไปได้ หรืออีกวิธีหนึ่งคือใช้วิธีการหน่วงเวลาแทนก็ได้ ซึ่งปกติแล้วจะใช้วิธีการหน่วงเวลาแทน เนื่องจากง่ายในการออกแบบวงจรและเขียนโปรแกรม

3. การเขียนข้อมูลให้กับ LCD Module สามารถทำได้ทั้ง 8 บิต และ 4 บิต โดยการใช้งานแบบ 4 บิต จะใช้ขาสัญญาณเพียง 4 เส้นคือ (DB4-DB7) ซึ่งการเขียนหรืออ่านข้อมูลนั้นกระทำเช่นเดียวกันกับแบบ 8 บิต เพียงแต่ต้องทำ 2 ครั้ง คือ (DB4-DB7) ก่อนแล้วตามด้วย (DB0-DB4) ซึ่งจะต้องกำหนดคุณสมบัติที่บิต DL ใน Function Set ก่อนด้วย

2.2.6 การเขียนข้อมูลให้กับ CGRAM ของ LCD Module

Character Code (DDRAM data)								CGRAM Address						CGRAM Data								Pattern number
D7	D6	D5	D4	D3	D2	D1	D0	A5	A4	A3	A2	A1	A0	P7	P6	P5	P4	P3	P2	P1	P0	
0	0	0	0	×	0	0	0	0	0	0	0	0	0	×	×	×	0	1	1	1	0	pattern 1
											0	0	1				1	0	0	0	1	
											0	1	0				1	0	0	0	1	
											0	1	1				1	1	1	1	1	
											1	0	0				1	0	0	0	1	
											1	0	1				1	0	0	0	1	
											1	1	0				1	0	0	0	1	
											1	1	1				0	0	0	0	0	
0	0	0	0	×	1	1	1	0	0	0	0	0	0	×	×	×	1	0	0	0	1	pattern 8
											0	0	1				1	0	0	0	1	
											0	1	0				1	0	0	0	1	
											0	1	1				1	1	1	1	1	
											1	0	0				1	0	0	0	1	
											1	0	1				1	0	0	0	1	
											1	1	0				1	0	0	0	1	
											1	1	1				0	0	0	0	0	

รูปที่ 2.3 แสดงความสัมพันธ์ระหว่าง DDRAM กับ Pattern ตัวอักษรของ CGRAM

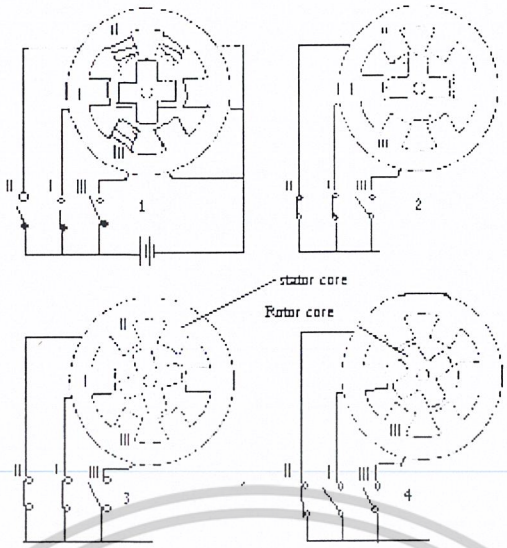
สำหรับการเขียนข้อมูลให้กับ CGRAM นั้นก็คือการโหลดตัวอักษรที่ออกแบบเองลงในหน่วยความจำของ CGRAM นั่นเอง ซึ่งจะทำได้ 8 ตัวอักษร โดยจะใช้แอดเดรสที่อ้างถึงหน่วยความจำของ CGRAM เพียง 6 บิต (A5-A0) สามารถอ้างถึงได้ทั้งหมด $(2^6) = 64$ ไบต์ ซึ่งก็คือ 8 ตัวอักษรคูณกับจำนวนแถว ดังแสดงในรูป จะต้องกำหนดเป็น Pattern แต่ละ Pattern ก็คือ 1 ไบต์ของแต่ละตัวอักษร ซึ่งแต่ละตัวอักษรที่ออกแบบจะต้องใช้ทั้งหมด 8 ไบต์

สำหรับการนำตัวอักษรที่โหลดเองออกไปแสดงผลนั้นจะต้องอ้างถึงรหัตัวอักษรของ DDRAM Data ซึ่งก็คือเมื่อทำการโหลดตัวอักษรที่ออกแบบเข้าไปยัง CGRAM แล้วการที่จะนำออกไปแสดงผลจะกระทำเหมือนกับการเขียนข้อมูลให้กับ DDRAM สำหรับตัวอักษรตัวแรก (Pattern 1) จะต้องส่งค่า 00H หรือ 08H ออกไป ตัวอักษรตัวต่อไป (Pattern ถัดไป) จะต้องส่งค่า 01H จนถึงตัวอักษรตัวที่ 8 ซึ่งเป็นตัวอักษรตัวสุดท้าย (Pattern 8) จะต้องส่งค่า 07H หรือ 0FH ออกไป

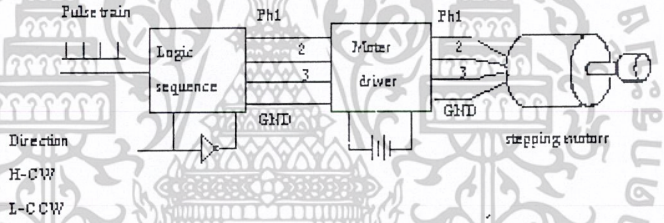
2.3 สเตปปิ้งมอเตอร์

การทำงานของสเตปปิ้งมอเตอร์ ดูจากรูปที่ 2.4 เป็นสเตปปิ้งมอเตอร์ที่มี แกนเหล็กสเตเตอร์ (stator Core) มีซี่ฟัน (Teeth) ซี่ ขณะที่โรเตอร์ (Rotor) มีฟัน 4 ซี่ ทั้งโรเตอร์และสเตเตอร์เป็นเหล็กอ่อน ขดลวด (coil) 3 ชุด ถูกต่ออยู่ดังรูป แต่ละชุดขดลวดมี 2 ขดลวดต่ออนุกรมกัน เรียกแต่ละชุดว่า เฟส (Phase) และผลจากการต่อแบบนี้เรียกว่ามอเตอร์ 3 เฟส (3 Phase Motor) กระแสถูกจ่ายไปยังขดลวดแต่ละขดโดยผ่านสวิทซ์ I , I และ III ในสภาวะ (1) ชุดขดลวดของเฟส 1 ได้รับกระแสโดยผ่านสวิทซ์ I หรือ เฟส 1 ถูกกระตุ้น เส้นแรงแม่เหล็กที่เกิดขึ้นในช่องอากาศ (air gap) เกิดขึ้นเนื่องจากการกระตุ้นซึ่งแสดงด้วยลูกศรในสภาวะนี้สเตเตอร์ 2 ขั้วของเฟส 1 จะอยู่ในแนวเดียวกับ 2 ซี่ที่อยู่ตรงข้ามกันของโรเตอร์ นี่เป็นสภาวะดุลย์ ซึ่งอยู่ในเทอมของไดนามิค (Dynamics) เมื่อสวิทซ์ II ปิด เพื่อกระตุ้นเฟส 2 กับเฟส 1 เส้นแรงแม่เหล็กจะถูกสร้างขึ้นที่ขั้วของสเตเตอร์ของเฟส 2 ในลักษณะซึ่งแสดงในสภาวะ (2) ทอร์ด (Torgue) ทิศทางทวนเข็มนาฬิกาจะถูกสร้างขึ้นจากความเครียด (Tension) ในฟลักซ์แม่เหล็กเอียงไปยังแกนมอเตอร์ที่อยู่ใกล้ หลังจากนั้นโรเตอร์จะมาอยู่ในสภาวะ (3)

ดังนั้นโรเตอร์จะหมุนไปด้วยการเปลี่ยนแปลงคงที่ ซึ่งเรียกว่ามุมสเตป (step angle) ในที่นี้คือ 15 องศา ขณะที่สวิทซ์จะมีการเปลี่ยนแปลงอีกครั้งหนึ่งคือ สวิทซ์ I จะถูกเปิด เพื่อลดพลังงานในเฟส 1 โดยโรเตอร์จะหมุนไป 15 องศา มาอยู่ในสภาวะ (4) ตำแหน่งมุมของโรเตอร์ จะถูกควบคุมโดยการเปิด ปิด สวิทซ์ ถ้าสวิทซ์ถูกปิด เปิด ตามลำดับ โรเตอร์จะหมุนไปในลักษณะที่สเตป ความเร็วเฉลี่ยจะสามารถควบคุมได้ด้วยการเปิด-ปิดสวิทซ์ ดังแผนภาพรูปที่ 2.4



รูปที่ 2.4 แสดงวงจรกระตุ้นสเตปปิ้งมอเตอร์ 3 เฟส



รูปที่ 2.5 แสดงไดอะแกรมของระบบขับสเตปปิ้งมอเตอร์

จากที่กล่าวมาพอที่จะทราบถึงคุณสมบัติของสเตปปิ้งมอเตอร์ได้ว่า

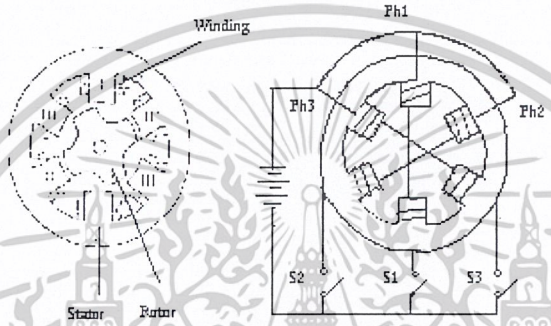
1. การหมุนของมอเตอร์จะเป็นสเตป (เป็นขั้น ๆ) สเตปละกี่องศาขึ้นอยู่กับชนิดของมอเตอร์
2. ความเร็วในการหมุนจะขึ้นอยู่กับสัญญาณพัลซ์ ที่ให้เข้ามาทางอินพุทของมอเตอร์ (ความถี่)
3. ความผิดพลาดในการหมุน 1 สเตปมีค่าน้อยมาก แต่ต้องจำกัดอยู่ในความเร็วที่พอดี
4. คุณสมบัติของการตอบสนองสัญญาณในขณะที่มอเตอร์เริ่มทำงานและหยุดทำงานดีมาก
5. เนื่องจากไม่มีแปลงถ่าน (Commutator) เหมือนมอเตอร์ในระบบดีซี ดังนั้นจึงมีความแน่นอนในการทำงาน
6. แหล่งจ่ายแรงดันไฟที่ใช้ในการขับมอเตอร์มีค่าไม่มาก
7. ไม่ทำให้เกิดเสียงรบกวน แรงขับมอเตอร์มีค่าไม่มาก
8. ทำงานแบบโอเพนลูป (Open loop)

สเตปปิ้งมอเตอร์ที่ใช้กันส่วนมากมี 3 ชนิด

1. วาริเอเบิลริลัคแทนซ์สเตปปิ้งมอเตอร์ (Variable Steppeng Motor)
2. สเตปปิ้งมอเตอร์แบบแม่เหล็กถาวร (Permanent magnetic Stepping Motor)
3. สเตปปิ้งมอเตอร์แบบไฮบริด (Hybride Stepping Motor)

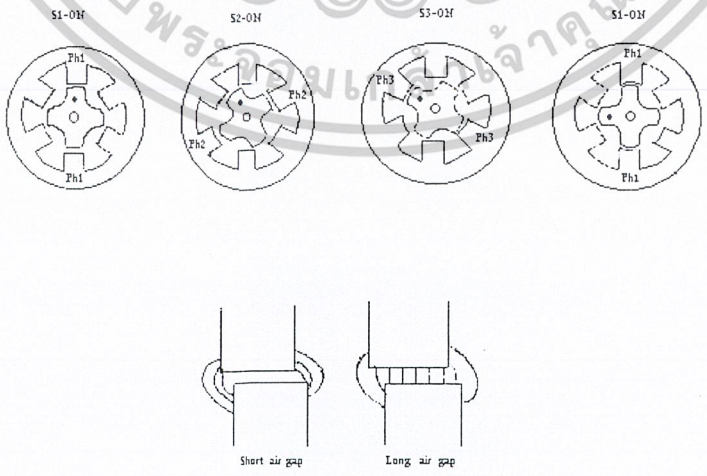
2.3.1 วาริเอเบิลลัคแทนซ์สเตปปิ้งมอเตอร์

มอเตอร์แบบนี้โรเตอร์ทำด้วยเหล็กอ่อน ซึ่งค่าซึมซาบแม่เหล็ก (Permeability) สูงและสามารถให้ ฟลักซ์แม่เหล็กผ่านได้มาก โดยโรเตอร์จะติดอยู่กับแกนมอเตอร์ และสเตเตอร์ติดอยู่กับโครงของตัวมอเตอร์ จากรูป 2.6 เป็นภาพตัดขวางของสเตปปิ้งมอเตอร์แบบนี้ ซึ่งเป็นมอเตอร์ 3 เฟส



รูปที่ 2.6 แสดงโครงสร้างวาริเอเบิลรีลัคแทนซ์มอเตอร์

เราจะเห็นได้ว่าฟันของสเตเตอร์ 2 ชิ้ ที่มีเฟสเดียวกัน จะมีขั้วแม่เหล็กตรงข้ามกันและกันดังแสดงดังรูปที่2.4 สมมุติว่าฟัน I II และ III มีขั้วเป็นขั้วเหนือฟัน I II และ III จะเป็นขั้วใต้เมื่อถูกกระตุ้นกระแสแต่ละเฟสจะถูกกระตุ้นฟลักซ์แม่เหล็กก็จะเกิดดังรูปที่ 2.7 ฟันของโรเตอร์ก็จะมีตำแหน่งในแนวเดียวกันกับฟันของสเตเตอร์ ซึ่งจะมีผลให้แมกเนติกลัคแทนซ์ (Megnetic Teluctance) น้อยที่สุด สภาวะนี้คือตำแหน่งสมดุลย์

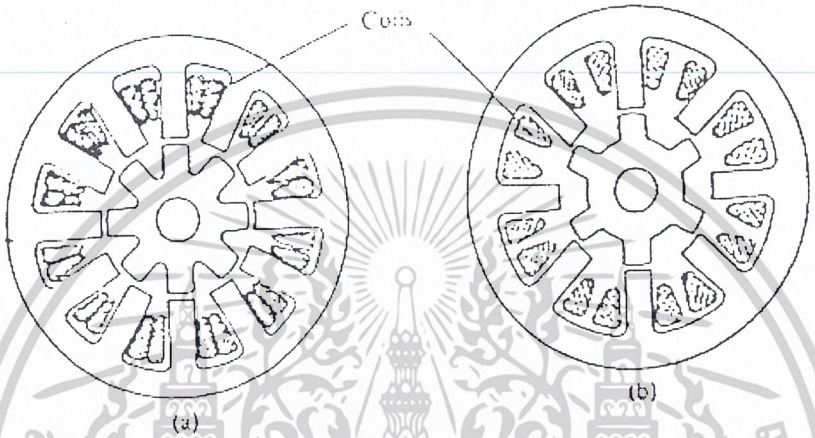


รูปที่ 2.7 แสดงฟลักซ์แม่เหล็กที่เกิดขึ้น

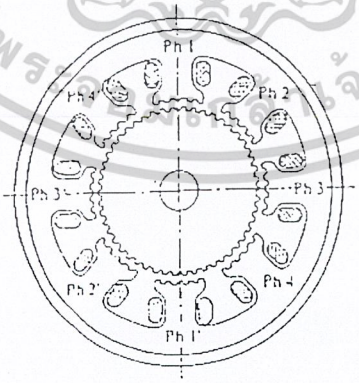
โครงสร้างเบื้องต้น ของมอเตอร์แบบนี้ จะมีลักษณะดังนี้

1. ช่องว่างอากาศควรจะเล็กที่สุด เท่าที่จะเป็นไปได้ ช่องว่างอากาศระหว่างฟันของโรเตอร์กับฟันของสเตเตอร์ควรมีค่าความห่างกันน้อยมาก เพื่อที่จะได้ทอร์คสูง และตำแหน่งที่แน่นอนขึ้น

2. สำหรับมุมสเตปเล็ก ๆ จากรูป 2.8a แสดง 3 เฟสมอเตอร์ที่สเตเตอร์มีฟัน 12 ซี่ และฟันมีโรเตอร์ 8 ซี่ รูป 2.8 b เป็นรูป 4 เฟส มอเตอร์ที่สเตเตอร์มีฟัน 8 ซี่ และโรเตอร์มีฟัน 6 ซี่ ซึ่งทั้งสองรูปนี้มีมุมสเตปเท่ากับ 15 องศา



รูปที่ 2.8 แสดงโรเตอร์และสเตเตอร์



รูป ที่ 2.9 แสดงหน้าตัดของ 4 เฟสมอเตอร์มีฟันโรเตอร์ 50 ซี่ มุมสเตป 1.8 องศา

ความสัมพันธ์ระหว่างมุมสเตป s จำนวนเฟส m โดยจำนวนฟันของโรเตอร์ Nr และจำนวน สเตป ใน 1 รอบ S จะหาได้โดย

$$S = 360 / s = m(Nr)$$

นั่นคือ เราสามารถสดจำนวนสเตปลงได้โดยเพิ่มจำนวนฟันบนโรเตอร์ ดังแสดงในรูป 2.9

2.3.2 สเตปปิ้งมอเตอร์แบบแม่เหล็กถาวร

สเตปปิ้งมอเตอร์แบบนี้ใช้ แม่เหล็กแบบถาวร รูปที่ 2.10 เป็นตัวอย่างของสเตปปิ้งมอเตอร์แบบถาวร แบบ 4 เฟสโรเตอร์ เป็นทรงกระบอก สเตเตอร์มีฟัน 4 ซี่ โดยที่แต่ละซี่มีขดลวดพันรอบ ถ้าจำนวนซี่บนสเตเตอร์และขั้วแม่เหล็กบนโรเตอร์เพิ่มขึ้นเป็น 2 เท่า มุมแต่ละสเตปลดลงจากเดิมครึ่งหนึ่ง ดังนั้นเพื่อที่จะลดมุมสเตปลงไปอีก ในสเตปปิ้งมอเตอร์แบบแม่เหล็กถาวรจะต้องเพิ่มจำนวนแม่เหล็กและซี่ฟัน อย่างไรก็ตามขั้วแม่เหล็กที่จะเพิ่มขึ้นได้นั้นมีจำนวนจำกัด

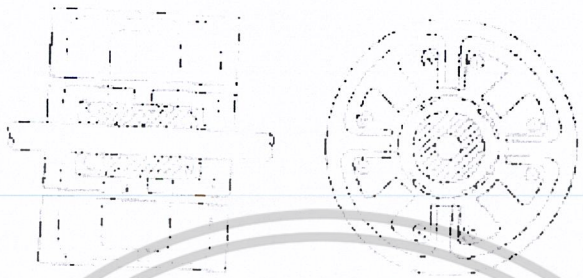
ลักษณะทั่วไปของมอเตอร์แบบนี้คือ โรเตอร์จะถูกยึดติดอยู่กับที่ แม้ว่าไม่มีการกระตุ้นเฟส ลักษณะเช่นนี้เรียกว่า ดีเทนน์ แมคคานิคซิ่ม (DETENT MECANISM)

รูปที่ 2.10 แสดงโครงสร้างของสเตปปิ้งมอเตอร์ 4 เฟส

2.3.3 สเตปปิ้งมอเตอร์แบบไฮบริด

เป็นสเตปปิ้งมอเตอร์แบบหนึ่ง ที่มีโรเตอร์เป็นแบบแม่เหล็กถาวรการใช้ ชื่อไฮบริดได้มาจากการรวมหลักสำคัญของมอเตอร์แบบแม่เหล็กถาวรและแบบวาริเอเบิลรีลัคแทนซ์ โครงสร้างแกนของสเตเตอร์จะคล้ายกับแบบวาริเอรีลัคแทนซ์ แต่การพันและการต่อขดลวดจะต่างจากแบบวาริเอเบิลแทนซ์มอเตอร์ ซึ่งมีเพียง 1 ขดจาก 2 ขดของ 1 เฟส ที่ถูกพันบนขั้วเดียวในขณะที่ 4 เฟส ไฮบริดมอเตอร์ ชุดขดลวด 2 เฟส ที่

แตกต่างกันจะถูกพันเป็นขั้วเดียวกันแบบไบฟิลา (Bifilar Winding) ซึ่งจะทำให้ขั้วแม่เหล็กต่างกันขณะมีการกระตุ้น. ลักษณะที่สำคัญอีกอย่างหนึ่งของไฮบริดมอเตอร์คือโรเตอร์นั้นจะเป็นแม่เหล็กรูปร่างทรงกระบอกอยู่ในแกนเหล็กของโรเตอร์ มันถูกทำให้เป็นแม่เหล็กตามยาวเพื่อสร้างสนามขั้วเดียวดังรูปที่ 2.11 แต่ละขั้วของแม่เหล็กจะถูกล้อมรอบด้วยฟันเหล็กอ่อน ซึ่งฟันของโรเตอร์กับสเตเตอร์อยู่เหลื่อมกันอยู่ครึ่งช่วงฟัน



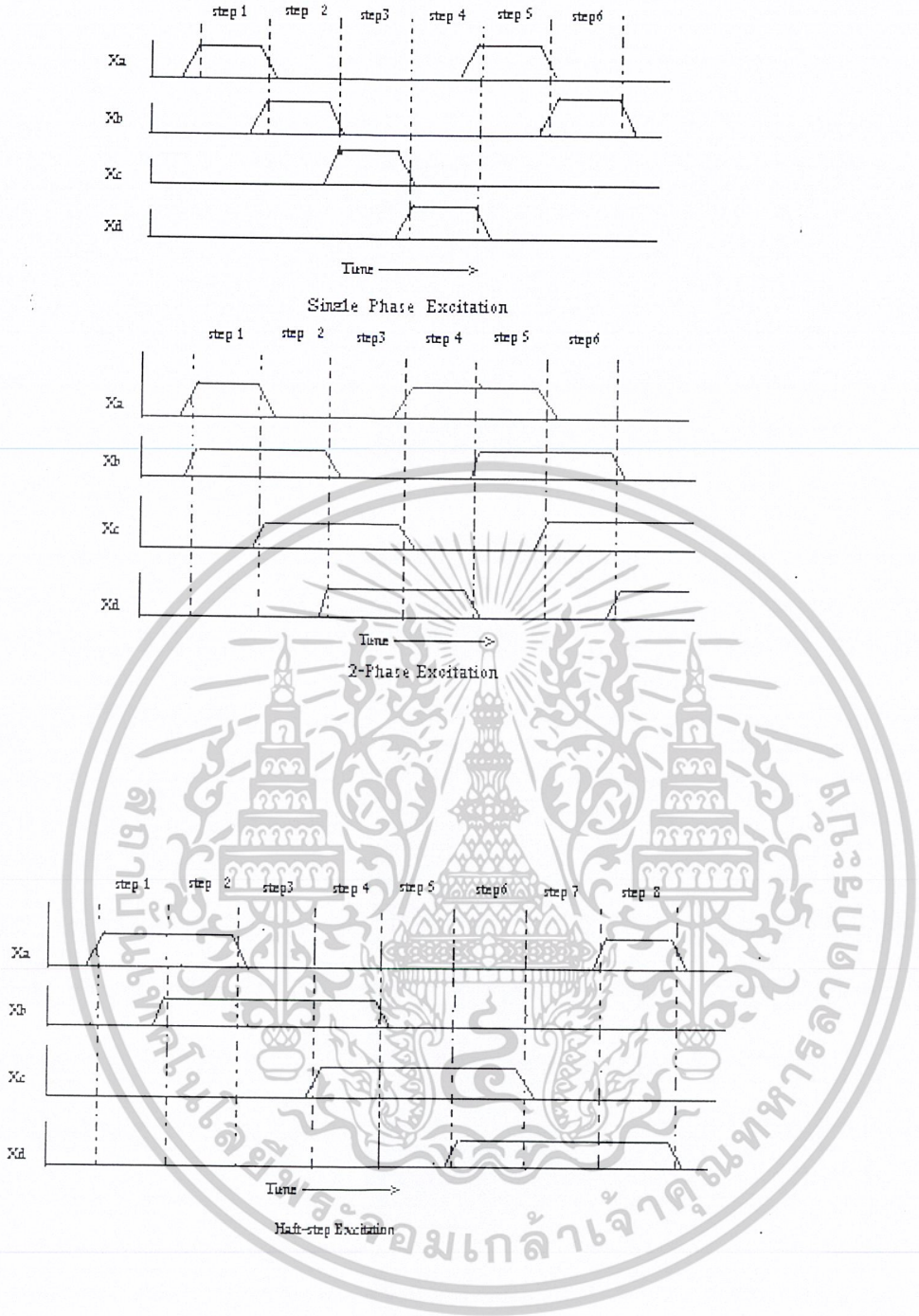
รูปที่ 2.11 แสดงการวางแผ่นแม่เหล็กตามยาวเพื่อสร้างสนามขั้วเดียวกัน

การทำงานพื้นฐาน

สเตปปิ้งมอเตอร์เป็นมอเตอร์ที่ถูกใช้ทำงานโดยสัญญาณอินพุทที่เป็นพัลซ์ ทุก ๆ สัญญาณพัลซ์ที่ให้เข้ามา จะให้การเปลี่ยนแปลงสถานะสนามแม่เหล็กไฟฟ้าที่เกิดขึ้น และให้การหมุนของมอเตอร์เป็นมุมที่คงที่ซึ่งจะแตกต่างกับการหมุนของมอเตอร์แบบธรรมดาในระบบควบคุม ที่ใช้สัญญาณดิจิตอลทั้งหมด จึงทำให้การควบคุมการทำงานของสเตปปิ้งมอเตอร์โดยตรงเป็นไปอย่างดี การทำงานพื้นฐานของสเตปปิ้งมอเตอร์ จะมีการขับสเตปปิ้งมอเตอร์ ทำให้กระแสไฟ ดีซี เรียงลำดับเข้าไปที่มอเตอร์ได้ซึ่งทำได้ด้วยการใช้งานของสวิทซ์และเพื่อให้มอเตอร์หมุนจำเป็นต้องมีวงจรขับสเตปปิ้งมอเตอร์ที่จะควบคุมการไหลของกระแสไฟ คือสวิทซ์ 1 และสวิทซ์ 2 ถ้าให้การทำงานของสวิทซ์ ตามรูปที่แสดงไว้ในรูปที่ 2.12 ถ้าให้การทำงานโดยรีเรย์ (Relay) หรือมือโยก สเตปมอเตอร์ก็จะหมุนเหมือนกันแต่หมุนช้า เนื่องจากรีเลย์หรือมือโยกให้การเปลี่ยนแปลงของสวิทซ์ช้า ซึ่งจะทำได้เร็วเท่ากับความถี่สูงสุดของสัญญาณพัลซ์ที่มอเตอร์ทำงานสามารถทำงานได้ (เป็นจำนวนพัลซ์ต่อวินาที)

วงจรขับมอเตอร์มีหลายชนิด ขึ้นอยู่กับชนิดสเตปปิ้งมอเตอร์ที่มีอยู่ 3 ระบบ คือ

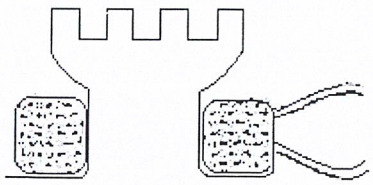
1. ระบบการกระตุ้นสนามแม่เหล็กเฟสเดียว (Single Phase Excitation)
2. ระบบการกระตุ้นสนามแม่เหล็ก 2 เฟส (2-Phase Excitation)
3. ระบบการกระตุ้นสนามแม่เหล็ก 1-2 เฟส (Half –Step Excitation)



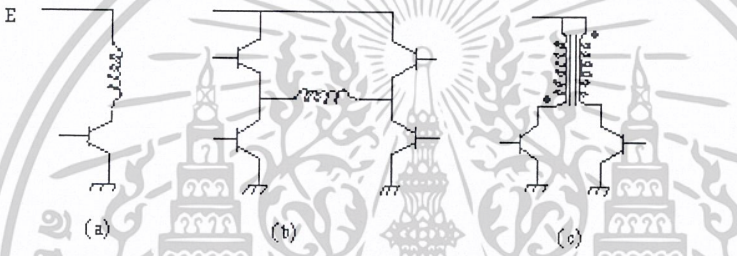
รูปที่ 2.12 แสดงชนิดของการขับสเตปปิ้งมอเตอร์

การพันลวดแบบโมโนฟิลาและไบฟิลา (monofilar and Bifilar Winfing)

แบบโมโนฟิลา มีการพันลวดเส้นเดียว ส่วนแบบไบฟิลา เส้นลวดทั้ง 2 เส้นนี้จะถูกพันเหมือนกับเส้นเดียวกัน ดังรูปที่ 2.13 และขดลวดทั้งสองเส้นนี้จะแยกกันที่ปลายเป็นลักษณะ 2 เส้นแยกกัน ถ้าขดลวดเป็นของเฟส 1 อีกขดหนึ่งเป็นของเฟส 3 และทำนองเดียวกัน ถ้าขดหนึ่งเป็นของเฟส 2 ขดหนึ่งของเฟส 4 (กรณีนี้เป็นมอเตอร์ 4 เฟส)



รูปที่ 2.13 แสดงการพันแบบไบฟิลา



รูปที่ 2.14 แสดงการพันแบบโมโนฟิลา

จุดประสงค์ของการพันแบบไบฟิลาก็คือ เพื่อที่จะให้พลังงานกับขั้วแม่เหล็กสเตเตอร์ โดยการสลับขั้วแม่เหล็ก การกระตุ้นแต่ละเฟสอาจเป็นแบบใดแบบหนึ่งใน 3 แบบในรูป

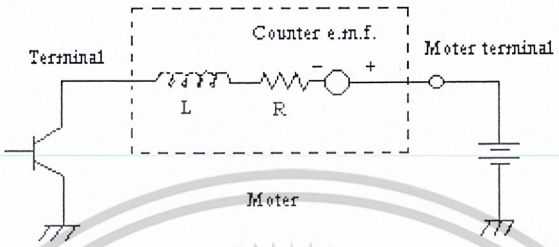
ในวงจรรูปที่ 2.14 เป็นโมโนฟิลา ขั้วแม่เหล็กจะถูกกระตุ้นเป็นขั้วเหนือขั้วใต้เสมอซึ่งแสดงว่าไม่สามารถกลับขั้วแม่เหล็กได้ การกระตุ้นแบบนี้เป็นการกระตุ้นแบบขั้วเดียว (Unipolar Exited) ใน วงจรรูปที่ 2.14 b ทิศทางของกระแสในขดลวดสามารถสลับได้เนื่องจากเป็นวงจรบริดจ์ (Bridge Exited)

อย่างไรก็ตามจะต้องใช้ทรานซิสเตอร์ ถึง 4 ตัวต่อขดลวด 1 ขด แบบนี้จะทำให้ทอร์คที่ความเร็วต่ำมากกว่าแบบไบฟิลา แต่ก็มีโอกาสที่ทรานซิสเตอร์จะพังได้เนื่องจากจัดเวลาผิดพลาด

ในวงจรรูปที่ 2.14 c เกี่ยวกับคู่สายไบฟิลา และทรานซิเตอร์ 2 ตัว โดยทำให้ สเตเตอร์ถูกกระตุ้นเป็นขั้วแบบไบฟิลา ก็จะเกิดสนามแม่เหล็กคัปปลิง (Coupling) เมื่อขั้วใดขั้วหนึ่งถูกกระตุ้น ถ้าแทนการพันแบบไบฟิลาด้วยเส้นลวด 2เส้น ที่แยกจากกันความแตกต่างของอินดัคแตนซ์ (Inductance) จะปรากฏระหว่างขด 2 ขด ทำให้ตำแหน่งผิดไปโดยทั่วไปประสิทธิภาพ ของมอเตอร์แบบแม่เหล็กถาวรกับแบบไฮบริดที่ใช้แบบไบฟิลานี้จะได้ประสิทธิภาพดีกว่าแบบโมโนฟิลา

2.3.4 วงจรขับมอเตอร์ (Motor Drive Circuit)

จุดประสงค์ของวงจรขับ คือ เพื่อที่จะให้ได้โวลท์เตจ และกระแสที่ถูกต้องไปยังมอเตอร์ในช่วงเวลาที่สั้นและลักษณะที่มีประสิทธิภาพทิศทางของกระแสในขดลวดของมอเตอร์แบบแม่เหล็กถาวรแบบไฮบริดมีความสำคัญ เพราะจะต้องใช้ในทิศทางที่เหมาะสมในเวลาต่าง ๆ สำหรับการขับมอเตอร์แบบวาริเอเบิลรีลัคแทนซ์ จะไม่ขึ้นกับทิศทางของกระแสที่ไหลในขดลวดเนื่องจากไม่มีแม่เหล็กอยู่ในตัวมันเลย

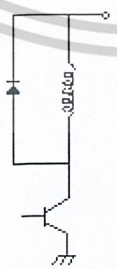


รูปที่ 2.15 แสดงวงจรสมมูลย์ของสเตปปิ้งมอเตอร์

วงจรขับมอเตอร์จะรับสัญญาณควบคุมมาจากวงจรจัดลำดับลอจิก ในการสร้างวงจรขับมอเตอร์จะมีปัญหาเกี่ยวกับอินดัคแทนซ์และรีซิสแทนซ์ อนุกรมกันในวงจรสมมูลย์ของมอเตอร์ ดังรูปที่ 2.15 อีกทั้งในการหมุนยังมีปัญหาเพิ่มขึ้นเกี่ยวกับแรงดันไฟกลับ (Backelectromotive Force : Back EMF.) นอกจากนี้ยังต้องคำนึงถึงแหล่งจ่ายไฟ ดีซีและการใช้การป้องกัน Power ทรานซิสเตอร์ด้วย

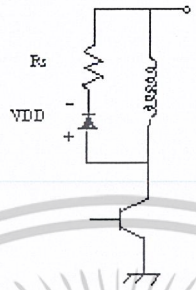
จากรูปที่ 2.15 เมื่อทรานซิสเตอร์หยุดทำงานจะเกิดสไปค์โวลท์เตจ (Spilk Voltage) เนื่องจากขดลวด (ตัวเหนี่ยวนำ) ซึ่งอาจทำให้ทรานซิสเตอร์พังได้ จึงต้องมีวงจรป้องกันและลดลักษณะแบบนี้อย่างรวดเร็วโดยวิธีการดังต่อไปนี้

1. ลดโดยใช้ไดโอด (Diode Suppressor) ใช้ไดโอดต่อคร่อมขนานกับขดลวด ดังรูปที่ 2.15 กระแสจะไหลอยู่ในวงจรขดลวดกับไดโอด หลังจากทรานซิสเตอร์หยุดทำงาน ซึ่งกระแสนี้จะลดลงไปตามเวลาแต่ก็ใช้เวลานานมากจึงยังเกิดทอร์คอยู่



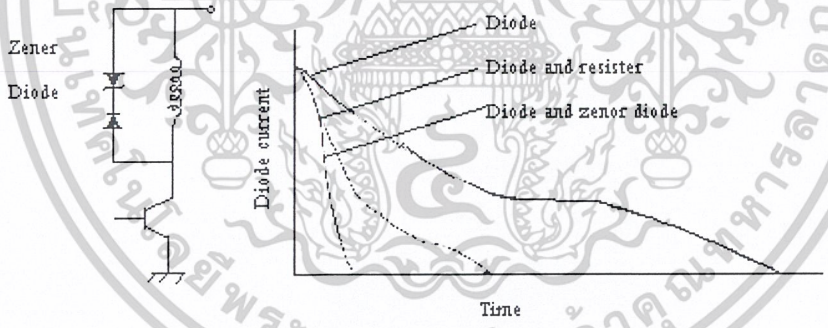
รูปที่ 2.16 แสดงวงจร Diode Suppressor.

2. โดยใช้ไดโอดและตัวต้านทาน (diode and resistor suppressor) โดยการต่อความต้านทานอนุกรมกับไดโอดแล้วต่อขนานคร่อมขดลวด ดังรูปที่ 2.17 วิธีนี้จะช่วยลดเวลาในการที่จะทำให้กระแสหยุดไหลวน ยิ่งใช้ตัวต้านทานค่ามาก กระแสก็จะหยุดไหลเร็วขึ้น แต่ทรานซิสเตอร์ต้องทนโวลท์เตจคร่อมคอลเลคเตอร์ – อิมิเตอร์ เพิ่มขึ้นด้วย



รูปที่ 2.17 แสดงวงจร Diode /Resistor Suppressor

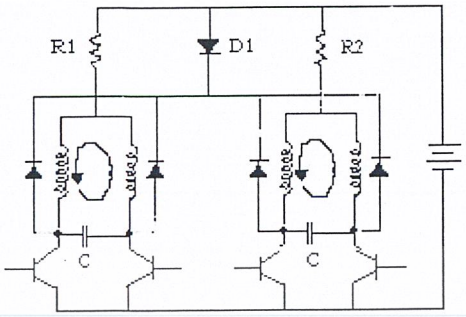
3. โดยใช้ซีเนอร์ไดโอด (Zenor Diode Suppressor) โดยการต่อซีเนอร์ไดโอด ดังรูปที่ 2.18 การต่อแบบนี้เมื่อเปรียบเทียบกับ 2 แบบแรก แล้วจะมีประสิทธิภาพมากกว่าซึ่งสามารถลดกระแสที่เกิดขึ้นได้เร็วกว่า



รูปที่ 2.18 แสดงวงจร Zenor Diode Suppresor

4. โดยใช้ คอนเดนเซอร์ (Condenser Suppressor) วิธีนี้มักจะใช้กับสเตปปิ้งมอเตอร์แบบไบฟิลา การต่อวงจรดังรูป 2.19 ของมอเตอร์ 4 เฟส ตัวคอนแดนเซอร์ที่ต่ออยู่ระหว่างเฟส 1กับเฟส 3 และเฟส 2 กับเฟส 4 จะทำหน้าที่ดังนี้ ในตอนที่ทรานซิสเตอร์ T1 หยุดทำงาน ในการทำงานแบบกระตุ้นเฟสเดียว T2 หรือ T4 เริ่มทำงาน แต่ T3 ยังไม่เริ่มทำงาน เนื่องจากเฟส 1 และ เฟส 3 ต่ออยู่ในลักษณะไบฟิลา กระแสที่เกิดขึ้นหลังจากที่ T1 หยุดทำงานจะไหลวนไปตามเส้นไข่ปลาดังแสดงในรูป ถ้า T3 หรือ T4 เริ่มทำงานแต่ T3 ยังไม่เริ่มทำงานประจุที่ชาร์จ (Charge) เก็บไว้ในคอนเดนเซอร์ จะช่วยเพิ่มทอร์คให้ โดยการปล่อยกระแสไหลผ่านเฟส 1 ทั้งนี้การต่อคอนเดนเซอร์สามารถใช้กับการกระตุ้นแบบ 2 เฟสได้ด้วย สำหรับค่า R1 R2 ใส่ไว้เพื่อปรับ

กระแสให้เหมาะสม นอกจากนั้นคอนเดนเซอร์ยังช่วยลดการสั่นของมอเตอร์ โดยเปลี่ยนพลังงานกลเป็นพลังงานความร้อนแทน



รูปที่ 2.19 แสดงวงจร Condenser Suppressor

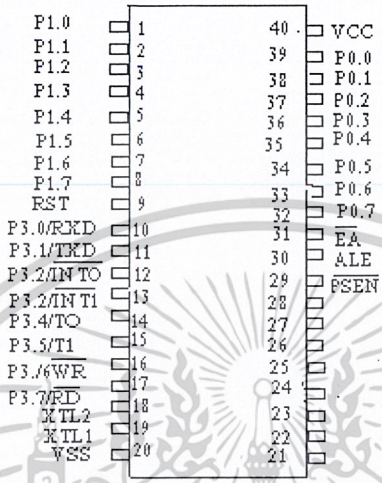
2.4 พื้นฐานทั่วไปของตระกูล MCS-51

CPU เป็นมันสมองของระบบ การอ่านโปรแกรมและทำงานตามคำสั่งโปรแกรมจะกระทำส่วนนี้ โดยการใช้ส่วนคณิตศาสตร์ และตรรกศาสตร์ทำงานร่วมกับ register A, B, PSW (Program Status Word), SP (Stack Pointer) ตัวนับโปรแกรม (PC : Program Counter) ขนาด 16 บิต และตัวชี้ตำแหน่งข้อมูล (DPTR) ส่วนคณิตศาสตร์ และตรรกศาสตร์ (ALU : Arithmatic Logic Unit) นี้ทำงานในฟังก์ชั่นด้วยตัวแปรต่าง ๆ ขนาด 8 บิต ที่มีลักษณะการทำงานเป็น บวก, ลบ, คูณ และหาร รวมทั้งทางตรรก เช่น AND, OR, XOR รวมทั้งการเลื่อนและวนรอบบิต การเคลียค่าและกลับค่า และ ALU ยังสามารถจะตัดสินใจในการให้กระโดดไปทำคำสั่งของโปรแกรมในส่วนอื่น ๆ ตามเงื่อนไขที่ตั้งไว้

ตระกูล MCS-51 นี้เป็นอุปกรณ์ที่ออกแบบมาสนองตามความต้องการของผู้ใช้คือมีสาย อินพุตและเอาท์พุตภายในตัวเอง พอร์ตของอินพุตและเอาท์พุตบัพเพ่อร์อินเตอร์เฟส และสายควบคุมอื่น ๆ ที่ใช้สำหรับแยก Data กับ Address และยังมีคำสั่งเพิ่มขึ้นเป็นพิเศษเพื่อจัดการข้อมูล แถมท้ายด้วยวงจรตั้งเวลากับวงจรนับด้วย (ปกติวงจรนับจะสามารถทำงานเป็นวงจรตั้งเวลาได้ด้วย จึงเรียกควบคู่กันไปคือวงจรตั้งเวลา / วงจรตรวจจับ) จากรูปที่ 2.20 แสดงการจัดวางขาต่าง ๆ ของ MCS-51

หน่วยความจำภายในตัว MCS-51 หน่วยความจำนี้แบ่งได้เป็น 2 กลุ่ม คือ หน่วยความจำสำหรับเก็บโปรแกรม และหน่วยความจำสำหรับเก็บข้อมูล หน่วยความจำแรกมี Address ที่ต่ำกว่า 4 หรือ 6 Kbytes บรรจุอยู่ใน ROM ส่วน MCS-51 ที่ไม่มี ROM ภายในจะใช้หน่วยความจำภายนอกซึ่งอาจเป็น ROM,RAM, หรือ EPROM แทน MCS-51 จะอ่านหน่วยความจำสำหรับเก็บโปรแกรม เข้ามาเป็นภาษาเครื่องตามลำดับ ส่วนหน่วยความจำสำหรับเก็บข้อมูลจะใช้เป็นที่เก็บตัวแปรการคำนวณหาผลลัพธ์ทันที หน่วยความจำสำหรับเก็บข้อมูลใช้ร่วมกับหน่วยความจำภายนอกได้ถึง 64 Kbytes ซึ่งเลือกใช้ ROM ก็ได้ และยังมี register พิเศษ ที่ใช้หน่วยความจำภายนอกของ RAM ได้ 128 หรือ 256 Kbytes รีจีสเตอร์ภายใน MCS-51

MCS-51 มีรีจีสเตอร์ที่อำนวยความสะดวกในการใช้งานตามคำสั่งต่าง ๆ ประกอบด้วย Accumulator, register B ที่ใช้ในการคูณและหาร programstatus word (PSW) stack pointer (SP), data pointer (DPTR), พอร์ต 0-3 รีจีสเตอร์แบบคู่ ซึ่งใช้ส่งและรับข้อมูลชนิดอนุกรม รีจีสเตอร์ 16 bit ที่เป็นวงจร ตั้งเวลา/วงจรนับ รีจีสเตอร์คำสั่ง สำหรับหน้าที่พิเศษ เช่น การ อินเตอร์รัพต์ RTC



รูปที่2.20 แสดงการจัดวางขาต่าง ๆ ของ MCS-51

2.4.1 การจัดขาลักษณะภายนอกของ MCS-51

จากรูป 2.20 แสดงการจัดขาตามลักษณะภายนอก ซึ่งมีรายละเอียดดังนี้

ขา Vss (ขา 20) เป็นขาสำหรับต่อลงดิน

ขา VCC (ขา 40) เป็นขาที่ต่อแรงดันไฟกระแสตรงขนาด 5 V

ขา P0.0-P0.7 / AD0-AD7 (ขา 32-39) เป็นพอร์ตไอโอ 8 บิต แบบ open drain bi-directional การเขียนค่า "1" ที่พอร์ตนี้จะเป็นการปล่อยลอยทำให้พอร์ตอินพุตมีสถานะอิมพีแดนซ์สูง ในการให้พอร์ตนี้บริการแบบไอโอจะทำงานเป็นมัลติเพล็กซ์ ด้วยสัญญาณแอด เดรสไบต์ต่ำกับบัสข้อมูล สำหรับการใช้งานแบบนี้จะใช้ลักษณะภายในเป็นตัวพูลอัพ พอร์ต 0 ยังใช้งานเป็นตัวส่งข้อมูลออกทางพอร์ตนี้ เมื่อใช้ด้านการตรวจสอบโปรแกรม ROM ภายใน และการโปรแกรมนัว EPROM ภายใน ถ้าใช้งานในลักษณะนี้การพูลอัพจากภายนอกจะต้องต่อกับค่าความต้านทานที่มีค่า 10 K

ขา P1.0-P1.7 (ขา 1-8) เป็นพอร์ตไอโอ 8 บิต พร้อมด้วยการพูลอัพภายใน ถ้าให้เป็นพอร์ตเอาท์พุต บัฟเฟอร์สามารถขับ TTL ได้ 4 ตัว เมื่อเขียนค่า "1" ด้วยโปรแกรมการให้สถานะเช่นนี้เป็นการ initial ใช้งานพอร์ตนี้เป็นอินพุต

ขา P2.0-P2.7 / A8-A15 (ขา 21-28) เป็นพอร์ตไอโอ 8 บิต พร้อมด้วยการพูลอัพภายในพอร์ตจะถูกใช้งานเป็นตัวส่งแอดเดรสไบต์สูง เมื่อใช้งานร่วมกับหน่วยความจำภายนอก เพื่อให้ ได้ถึง 16 บิต

ขา P3.0-P3.7 (ขา 10-17) เป็นพอร์ตไอโอ 8 บิต นอกจากนี้ยังใช้เป็นฟังก์ชั่นพิเศษได้ ดังตารางที่ 2.2

ขาพอร์ต	การทำงานฟังก์ชั่นพิเศษ
P3.0	RDX พอร์ตอนุกรม input
P3.1	TXD พอร์ตอนุกรม output
P3.2	INT0 อินเตอร์รัพภายนอกที่ 1
P3.3	INT1 อินเตอร์รัพภายนอกที่ 2
P3.4	T0 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลา/ตัวนับ 0
P3.5	T1 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลา/ตัวนับ 1
P3.6	WR สัญญาณควบคุมการเขียน
P3.7	RD สัญญาณควบคุมการอ่าน

ตารางที่ 2.2 แสดงฟังก์ชั่นพิเศษของพอร์ต

ขา RST (ขา 9) ต้องคงสถานะสูง เป็นเวลาประมาณสองคาบเวลาที่ออสซิลเลเตอร์ทำงานขณะที่ต้องการรีเซททั้งระบบ แต่ทำให้ตัวชิพรีเซทโดยอัตโนมัติ ขณะเปิดไฟจะใช้ คาปาซิเตอร์ต่อคร่อมระหว่าง RST กับ ขา Vcc

ขา ALE / PROG (ขา 30) เป็นขาแอดเดรสแลทซ์อีนาเบิล ด้วยการส่งพัลท์ออกไปใช้สำหรับแลทซ์ค่าแอดเดรสไบต์ต่ำจากพอร์ต 0 ในระหว่างการเข้าถึงข้อมูลจากหน่วยความจำภายใน ALE จะถูกส่งสัญญาณนาฬิกาออกมา ในอัตราความเร็ว 1/8 ของความถี่ออสซิลเลเตอร์ ตลอดเวลาแม้ว่าจะไม่มีการเข้าถึงข้อมูลภายใน ดังนั้นจึงสามารถที่จะใช้สัญญาณจากขานี้ เป็นตัวตั้งเวลาภายนอก หรือเป็นความถี่สัญญาณนาฬิกา แต่ความถี่ของสัญญาณจะช้าลงไปเท่าหนึ่ง ในการทำงานแบบเข้าถึงหน่วยความจำข้อมูลภายนอก ขานี้ยังใช้เป็นสัญญาณพัลส์เข้า สำหรับการควบคุมการโปรแกรม EPROM ภายในชิพ

ขา PSEN (ขา 29) เป็นสโตรปอ่านข้อมูลจากโปรแกรมหน่วยความจำภายนอก ขา PSEN จะสร้างสโตรปต่ำ 2 ครั้งใน 1 machine cycle และกลับเป็นสถานะสูง ถ้า PSEN ไม่มีพัลส์ส่งออก แสดงว่าชิพทำงาน ด้วยโปรแกรมหน่วยความจำภายใน

ขา EA/Vpp (ขา 31) มีสถานะสูงชิพจะทำงานตามโปรแกรมที่อยู่ในหน่วยความจำภายใน (โดยที่โปรแกรมจะต้องไม่ยาวกว่า 4 Kbytes สำหรับเบอร์ 8051 AH และ 8 Kbytes สำหรับ 8052 AH) การทำให้ EA มีสถานะต่ำจะเป็นการควบคุมให้ชิพทำงานตามโปรแกรมหน่วยความจำภายนอกได้ 64 Kbytes ส่วนเบอร์ 8031 และ 8032 ขา EA ต้องต่อลงดินถึงแม้จะไม่มี ROM ภายในก็ตาม

ขา XTAL1 (ขา 19) ใช้เป็น input เข้าสู่ตัวออสซิลเลเตอร์

ขา XTAL2 (ขา 18) ใช้เป็นตัว output จากตัวออสซิลเลเตอร์ขยายแบบ invet ที่กล่าวมาข้างต้นจะเป็นในแง่ของการกำหนดค่าขาต่าง ๆ เพื่อกำหนดค่าสภาวะในการทำงาน ในลักษณะใหม่ ยังมีอีกส่วน

หนึ่งที่มีความสำคัญไม่น้อยที่ควรจะเข้าใจเบื้องต้น ก่อนที่จะทำการเขียนโปรแกรมสั่งงานได้นั้น คือส่วนของโปรแกรม ฉะนั้นจะต้องทำความเข้าใจกับตัวแปรต่าง ๆ และรีจีสเตอร์ที่เป็นคุณสมบัติในการนำไปเขียนโปรแกรม

ACCUMULATOR : ACC

MCS-51 ใช้ ACC ที่มีขนาด 8 บิต คำสั่งส่วนใหญ่จะอ้างถึงรีจีสเตอร์นี้โดยถือค่าภายในเป็นตัวตั้ง และรับค่าผลลัพธ์ที่ได้จากคำสั่งทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร เข้ามาเก็บไว้ ACC ยังสามารถใช้เป็นตัวแหล่งกระทำหรือถูกกระทำในการทำงานตรรก และใช้เป็นตัวกลางในการถ่ายเทข้อมูล ในการติดต่อกับอุปกรณ์ภายนอกไอโอ และหน่วยความจำภายนอก รวมถึงการตรวจสอบตารางข้อมูล

REGISTER B

เป็นรีจีสเตอร์พิเศษที่ใช้งานสำหรับคำสั่งของการ คูณ และ หาร โดยใช้เป็นที่เก็บตัวคูณ หรือ หาร และเป็นที่เก็บผลลัพธ์ตัวที่ 2 หลังการคูณ และเศษหลังการหาร

PROGRAM STATUS WORD : PSW

PSW เป็นรีจีสเตอร์ที่แสดงผลที่ได้หลังจากการใช้คำสั่งต่าง ๆ และใช้เป็นตัวเลือกกลุ่มทำงานของรีจีสเตอร์กลุ่มต่าง ๆ

STACK POINTER : SP

MCS-51 จะรวมเอา stack ของฮาร์ดแวร์ที่ใช้ RAM ภายในสำหรับการเชื่อมต่อระหว่างโปรแกรมหลัก stack การผ่านพารามิเตอร์ระหว่างงานในแต่ละส่วนโปรแกรม และ stack เก็บตัวแปรข้อมูลชั่วคราว หรือ stack การเก็บสถานะระหว่างการบริการอินเตอร์รัพท์ไว้ภายในชิพ โดย SP มีขนาด 8 บิต จะเพิ่มค่าขึ้นโดยอัตโนมัติก่อนที่ข้อมูลจะนำมาเก็บในหน่วยความจำ ระหว่างการใช้คำสั่ง PUSH หรือ CALL และจะลดค่าของ SP ลงหลังจากที่ได้ถ่ายเทข้อมูลออกไปแล้วในคำสั่ง POP หรือ RETURN ในทางปฏิบัติ SP มีเนื้อที่น้อยกว่า 128 Kbytes และจะเริ่มต้นทำตำแหน่ง 07H ฉะนั้น stack จะเริ่มบรรจุข้อมูลที่ตำแหน่ง 08H และ MCS-51 สามารถแปลงค่าใน SP ได้ซึ่งจะเป็นการเปลี่ยนตำแหน่งของ stack ไปยังที่ใด ๆ ของ RAM ภายในชิพ

DATA POINTER : DPTR

DPTR เป็นรีจีสเตอร์ขนาด 16 บิต ที่ประกอบด้วยไบต์สูง (DPH) และ ไบต์ต่ำ (DPL) ที่สามารถแบ่งออกเป็นรีจีสเตอร์ 8 บิต สองตัวที่ใช้ได้อย่างอิสระหรือใช้รวมกันทั้ง 16 บิต ก็ได้ ในการ Increment หรือ Decrement เพื่อประโยชน์ในการใช้เป็นฐานของเลขที่อยู่ใน รีจีสเตอร์ ในการกระโดดในทางอ้อม ในการใช้คำสั่งเกี่ยวกับตารางข้อมูล และชี้ตำแหน่งของหน่วยความจำภายนอก

PORT 0-3

รีจีสเตอร์ P0, P1, P2 และ P3 ของกลุ่มรีจีสเตอร์ฟังก์ชั่นพิเศษ (SFR) จะเป็นตัว รีจีสเตอร์ที่แลทซ์ค่าของพอร์ตในขณะใช้งาน

SERIAL DATA BUFFER : SUBF

บัฟเฟอร์ข้อมูลอนุกรมแบ่งออกเป็นรีจีสเตอร์สองตัว ตัวหนึ่งเป็นบัฟเฟอร์ตัวส่ง และอีกตัวเป็นบัฟเฟอร์ตัวรับ เมื่อข้อมูลถ่ายเทเข้า SBUF มันจะถ่ายเข้าบัฟเฟอร์ส่ง ซึ่งเป็นตัวจัดการส่งข้อมูลอนุกรม วิธีการเคลื่อนย้ายเข้า SBUF ขึ้นอยู่กับการเริ่มแรกการส่งเมื่อข้อมูลย้ายออกจาก SBUF จะเป็นการรับข้อมูลจากบัฟเฟอร์ตัวรับ

CONTROL REGISTER

กลุ่ม SFR ที่เป็น IP, IE, TMOD, TCON, T2CON, SCON, PCON จะประกอบด้วยบิตบิตที่ใช้ควบคุม และแสดงสถานะของการใช้งานในระบบอินเตอร์รัพต์ตัวตั้งเวลา / ตัวนับ และพอร์ตอนุกรม ซึ่งแสดงค่าตำแหน่งดังนี้

		ตำแหน่ง
* ACC	Accumulator	0E0H
* B	B รีจีสเตอร์	0F0H
* PSW	Program Status Word	0D0H
SP	Stack Pointer	081H
DPTR	ตัวชี้ข้อมูลประกอบด้วย DPH และ DPL	083H
* P0	พอร์ต 0	082H
* P1	พอร์ต 1	080H
* P2	พอร์ต 2	090H
* P3	พอร์ต 3	0A0H
* IP	ตัวควบคุมการอินเตอร์รัพต์ตามลำดับ	0B0H
* IE	ตัวควบคุมการอินเตอร์รัพต์อีนาเบิล	0B8H
TMOD	ตัวควบคุมการเลือกโหมดตัวตั้งเวลา / ตัวนับ	0A8H
* T2CON	ตัวควบคุมตัวตั้งเวลา / ตัวนับ 2	089H
TCON	ตัวควบคุมตัวตั้งเวลา / ตัวนับ	088H
TH0	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 0 (ไบต์สูง)	0C8H
TL0	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 0 (ไบต์ต่ำ)	08CH
TH1	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 1 (ไบต์สูง)	08AH
TL1	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 1 (ไบต์ต่ำ)	08DH
+ TH2	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 2 (ไบต์สูง)	08BH

+ TL2	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 2 (ไบต์ต่ำ)	0CDH
+ RLDH	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบต์สูง)	0CBH
+ RLDL	รีจีสเตอร์ตัวตั้งเวลา / ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบต์ต่ำ)	0CAH
* SCON	ควบคุมการส่งข้อมูลอนุกรม 098H	
SBUF	บัฟเฟอร์ข้อมูลการส่งอนุกรม	099H
PCON	ควบคุมการใช้พลังงาน (Power)	097H

* = Bit addressable

+ = 8052 Only

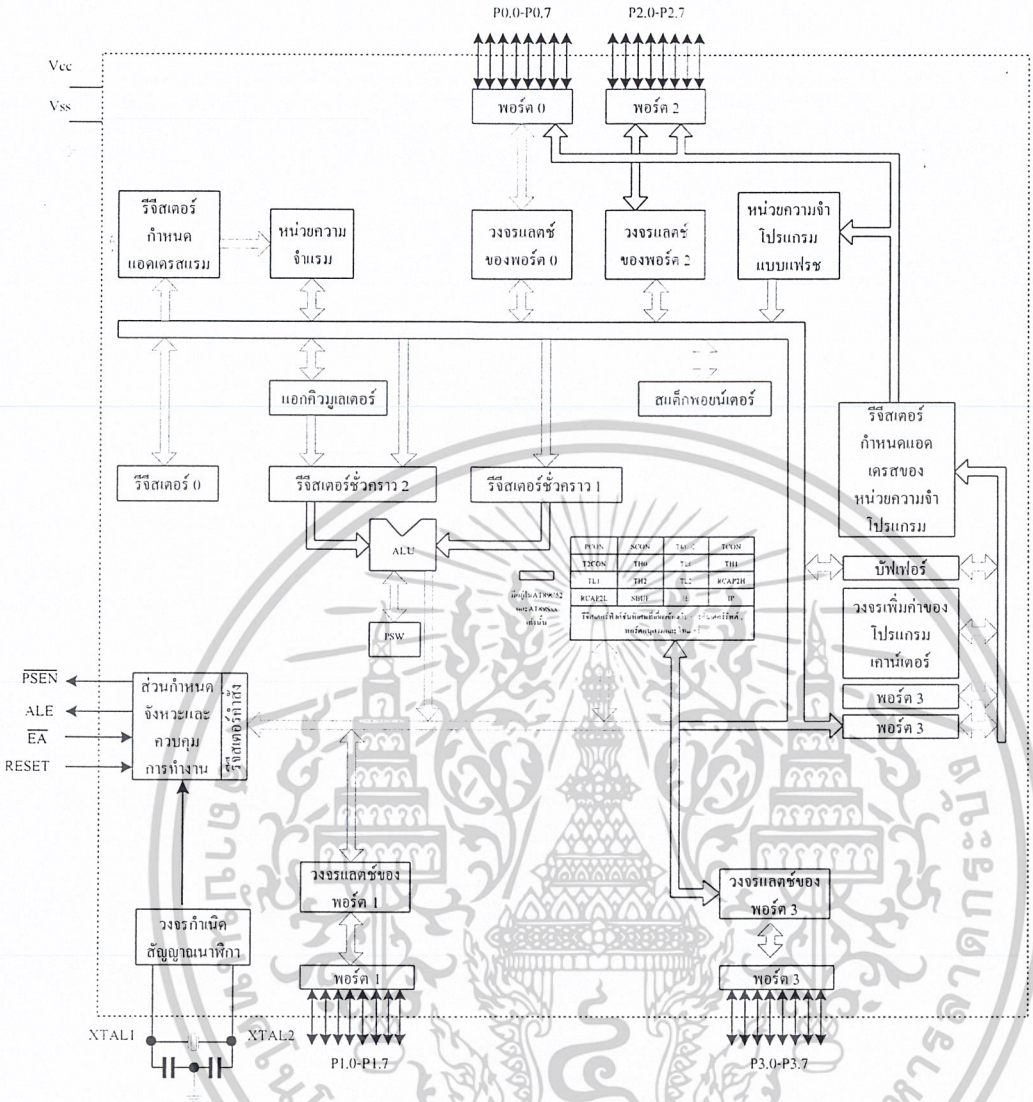
2.4.2 โครงสร้างและการทำงานของพอร์ต

ไมโครคอนโทรเลอร์ MCS-51 แบบแฟลชมีพอร์ตให้ใช้งานทั้งสิ้น 4 พอร์ต คือพอร์ต 0 ถึง พอร์ต 3 แต่ละพอร์ตมีขนาด 8 บิต เป็นพอร์ตแบบ 2 ทิศทาง กล่าวคือสามารถเป็นได้ทั้งอินพุต สำหรับรับสัญญาณข้อมูลเข้าและเอาต์พุตสำหรับส่งสัญญาณออก ทุกพอร์ตของไมโครคอนโทรเลอร์ MCS-51 แบบแฟลชซีมีวงแลตช์และวงจรขับตลอดจนบัฟเฟอร์อินพุต ดังแสดงให้เห็นในสถาปัตยกรรมรูป ที่ 2.21

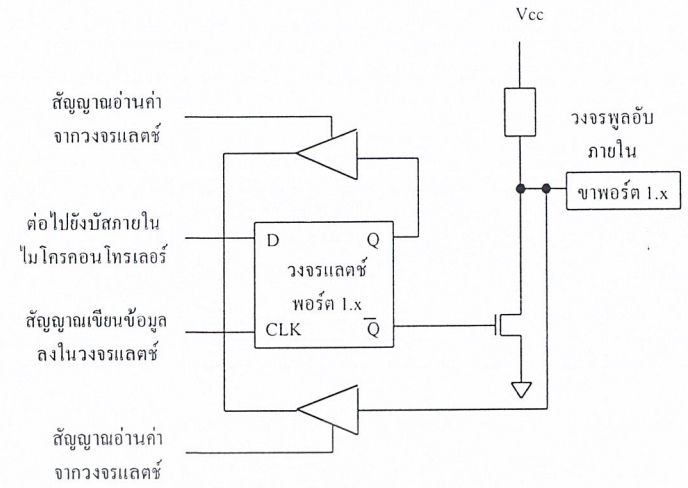
ที่พอร์ต 0 และพอร์ต 2 จะใช้งานเป็นพอร์ตอินพุตสำหรับงานทั่วไปและใช้ในการติดต่อกับหน่วยความจำภายนอก สำหรับพอร์ต 3 ทั้งพอร์ต และพอร์ต 1บางขานอกจากจะใช้เป็นขาพอร์ตอินพุตเอาต์พุตตามปกติแล้วยังสามรถใช้งานในหน้าที่พิเศษได้อีกขึ้นอยู่กับว่าไมโครคอนโทรเลอร์ MCS-51 แบบแฟลชเบอร์ใด ดังสรุปในตารางที่ 2.3

ในรูปที่ 2.22 แสดงวงจรภายในของแต่ละพอร์ตของไมโครคอนโทรเลอร์ MCS-51แบบแฟลชโดยในรูปที่ 2.22 (ก) เป็นวงจรของพอร์ต 0 วงจรแลตช์ของแต่ละบิตในแต่ละพอร์ตก็คือวงจร ดีฟลิฟลอบนั่นเอง การอ่านค่าสถานะของพอร์ตละสถานะของวงจรแลตช์สามารถกระทำได้อย่างอิสระด้วยสัญญาณที่แยกจากกัน นั่นคือสัญญาณอ่านข้อมูลจากขาพอร์ต และสัญญาณอ่านข้อมูลจากขาของแลตช์ ส่วนการเขียนข้อมูลมายังพอร์ตต้องส่งสัญญาณมายังขา CLK ของดีฟลิฟลอบในขณะที่ข้อมูลจะผ่านมาทางขาบัสข้อมูลภายในเข้าสู่ขา D ของดีฟลิฟลอบ

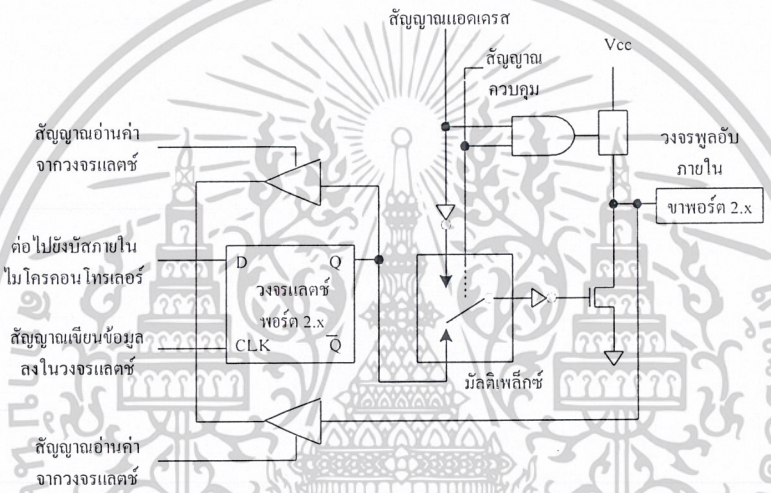
ที่วงจรนี้มีวงจรมัลติเพล็กซ์สำหรับกำหนดลักษณะการทำงานของพอร์ตว่า ต้องการใช้งานเป็นขาพอร์ตอินพุตเอาต์พุตปกติหรือใช้ในการติดต่อกับหน่วยความจำภายนอกไมโครคอนโทรเลอร์ เนื่องจากที่ขาพอร์ต 0 ไม่มีวงจรพูลอัปภายใน หากมีการนำวงจรพอร์ต 0 ไปใช้งานเป็นพอร์ตอินพุตจะต้องต่อตัวต้านทานพูลอัปภายนอก เข้าที่ขาพอร์ต 0 ทุกขาด้วย



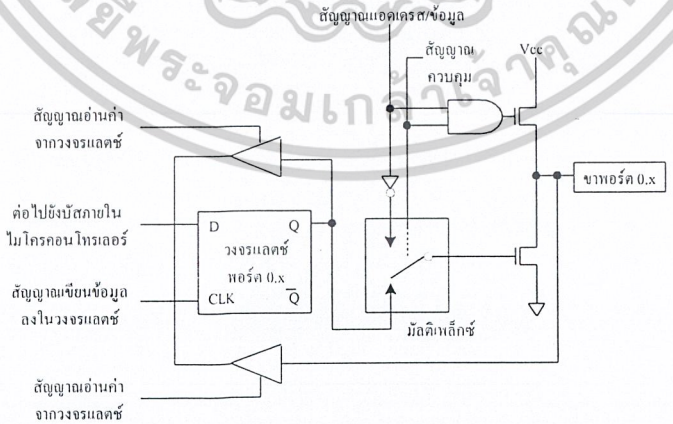
รูปที่ 2.21 แสดงรายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์ MCS-51



(ก)

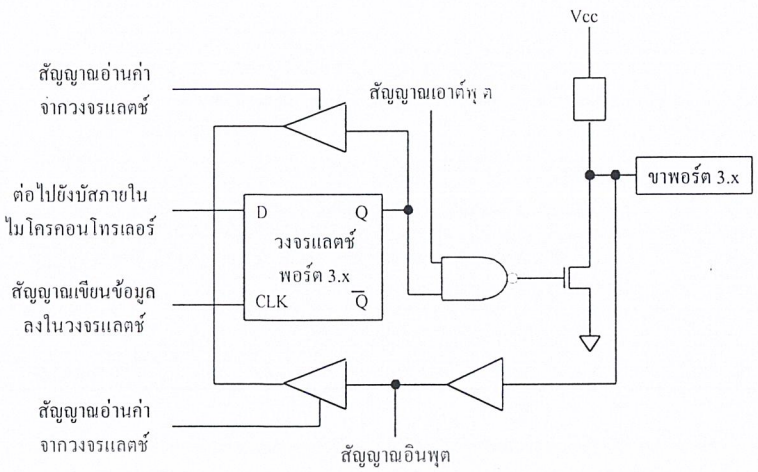


(ข)



(ค)

รูปที่ 2.22 แสดงวงจรภายในของพอร์ตทุกพอร์ตในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟรช



(ง)

รูปที่ 2.22 (ต่อ)วงจรภายในของพอร์ตทุกพอร์ตในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟรช

ในรูปที่ 2.22 (ข) เป็นวงจรของพอร์ต 1 ซึ่งมีลักษณะโดยทั่วไปคล้ายกับพอร์ต 0 หากแต่ไม่มีวงจรมัลติเพล็กซ์ เนื่องจากพอร์ตนี้จะไม่ใช้ในการติดต่อกับหน่วยความจำภายนอก แต่จะมีวงจรพูลอัปภายในที่แต่ละบิตของพอร์ตนี้แทน

ในรูปที่ 2.22 (ค) เป็นวงจรภายในของพอร์ต 2 จะคล้ายกับพอร์ต 0 มาก ต่างกันเพียงมีวงจรพูลอัปเพิ่มเติมเข้ามา ส่วนในรูปที่ 2.22 (ง) เป็นวงจรภายในของพอร์ต 3 มีการเพิ่มเติมวงจรบัฟเฟอร์และวงจรอินพุตเอาต์พุตเมื่อทำงานในฟังก์ชันพิเศษเข้ามา เนื่องจากพอร์ต 3สามารถนำไปใช้งานในหน้าที่พิเศษได้ทุกขา

2.4.3 การใช้งานเป็นพอร์ตอินพุต

เนื่องจากพอร์ตทั้งหมดของไมโครคอนโทรเลอร์ MSC-51 แบบแฟลชสามารถเป็นได้ทั้งอินพุตและเอาต์พุต ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะ การทำงานให้แก่พอร์ตของไมโครคอนโทรเลอร์ MSC-51 แบบแฟลช

ในการกำหนดโดยใช้เป็นพอร์ตอินพุต ต้องเริ่มด้วยการเขียนข้อมูล “1” มาที่แต่ละบิต ของพอร์ตที่ต้องการใช้งานเป็นอินพุต เพื่อหยุดการทำงานของเฟตที่ใช้ในการขับสัญญาณเอาต์พุตของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ตเชื่อมต่อเข้ากับวงจรพูลอัปโดยตรงส่งผลให้ขานั้นมีลอจิกเป็น “1” สามารถรับสัญญาณลอจิก “0” จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ต แล้วรอให้ซีพียูมาอ่านค่าไป เมื่อเป็นเช่นนี้อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ตอินพุตของไมโครคอนโทรเลอร์ MSC-51แบบแฟลชควรกำหนดให้ทำงานในสภาวะลอจิก “0” จะดีและสะดวกที่สุด (ซึ่งในปัจจุบันไมโครคอนโทรเลอร์แทบทั้งหมดทำงานที่ลอจิก “0” แล้ว)

ขา	เบอร์ของ ไมโครคอนโทรลเลอร์	หน้าที่พิเศษ
P1.0	AT89C52 / AT89Sxx	ขา T2 เป็นอินพุตนับค่าของไทเมอร์/เคาน์เตอร์ 2 และเป็นขาควบคุมทิศทางของสัญญาณ
P1.1	AT89C52 / AT89Sxx	ขา T2 เป็นอินพุตนับค่าของไทเมอร์/เคาน์เตอร์ 2 และเป็นขาควบคุมทิศทางของสัญญาณ
P1.4	AT89Sxx	ขา $\overline{SS}$ (Slave Select)เป็นขาเลือกการติดต่อในกรณีที่ไมโครคอนโทรลเลอร์เป็นอุปกรณ์ ในระบบการติดต่อแบบ SPI
P1.5	AT89Sxx	ขา MOSI (Master data output ,slave data input) ใช้ในการติดต่อกับพอร์ต SPI
P1.6	AT89Sxx	ขา MOSI (Master data input ,slave data output) ใช้ในการติดต่อกับพอร์ต SPI
P1.7	AT89Sxx	ขา SCK (Master clock output) เป็นขาสัญญาณนาฬิกาของการติดต่อกับพอร์ต SPI

ตารางที่ 2.3 แสดงหน้าที่พิเศษของพอร์ต 1 ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟรชของ Atmel

2.4.4 การใช้งานเป็นพอร์ตเอาต์พุต.

โดยปกติแล้ว ขาพอร์ตจะกำหนดให้มีลักษณะเป็นเอาท์พุตอยู่แล้ว ดังนั้นสามารถส่งข้อมูลออกไปได้อย่างง่ายดายและตรงไปตรงมา กล่าวคือเมื่อต้องการส่งข้อมูล “0” ออกไปทางเอาท์พุตก็ให้เขียนข้อมูล “0” ไปยังวงจรแลตช์ ซึ่งก็จะส่งต่อไปขับเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก “0” ขึ้น ในทางตรงข้ามหากต้องการส่งข้อมูล “1” ออกไป ก็ให้เขียนข้อมูล “1” ไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ตเชื่อมต่อกับวงจรพูลอัปภายในเกิดเป็นลอจิก“1”ที่ขาพอร์ตนั้น ซึ่งคล้ายกับการกำหนดให้เป็นขาอินพุตมาก เพียงแตกต่างกันที่กระบวนในการเคลื่อนย้ายข้อมูลโดยถ้าเป็นอินพุตจะมีสัญญาณมาอ่านข้อมูลที่บัพเฟอร์ แต่ถ้าเป็นเอาท์พุตจะไม่มีการอ่านข้อมูลที่บัพเฟอร์แต่อย่างใด เว้นแต่กรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาท์พุต

เมื่อใช้งานเป็นพอร์ตเอาท์พุต แต่ละขา (หรือแต่ละบิต) ของแต่ละพอร์ตมีความสามารถในการจ่ายกระแสหรือที่เรียกว่า กระแสซอร์ส (source current) ได้สูงสุด 10 mA และทุกขารวมกันในแต่ละพอร์ต (ทั้ง 8 บิต) สูงสุด 26 mA สำหรับพอร์ต 0 และ 15 mA สำหรับพอร์ต 1-3 ในกรณีที่ใช้งานทุกพอร์ตจะสามารถจ่ายกระแสได้รวมกันสูงสุด 71 mA ดังนั้นในการใช้งานเป็นพอร์ตเอาท์พุตเพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัพเฟอร์ทางเอาท์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง

2.4.5 การอ่านค่าลอจิกจากพอร์ต

ในไมโครลคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถอ่านค่าลอจิกจากพอร์ตได้ 2 ลักษณะ คือ อ่านจากขาพอร์ตโดยตรง และอ่านค่าจากวงจรแลตช์ของแต่ละพอร์ต

ในกรณีที่พอร์ตต่อกับขาเบสทรานซิสเตอร์ชนิด NPN และขาอีมิตเตอร์ของทรานซิสเตอร์ตัวนั้นต่อลงกราวด์ หากมีการส่งข้อมูล "1" ไปยังทรานซิสเตอร์จะทำให้ทรานซิสเตอร์ทำงานสถานะลอจิกที่ขาพอร์ตจะเป็น "0"เนื่องจากเมื่อทรานซิสเตอร์ทำงาน จะเสมือนว่าขาพอร์ตนั้นถูกต่อลงกราวด์ ทำให้อ่านค่าลอจิกที่ขาพอร์ตจะได้ผลตรงข้ามกับที่ส่งออกมาแต่ถ้าหากทำงานอ่านค่าลอจิกที่วงจรแลตซ์ จะได้ค่าที่ตรงกับค่าที่ต้องการส่งจริง ดังนั้น ในการอ่านค่าลอจิกจากพอร์ตจึงต้องเลือกวิธีการให้เหมาะสมกับอุปกรณ์ที่นำมาต่อด้วย

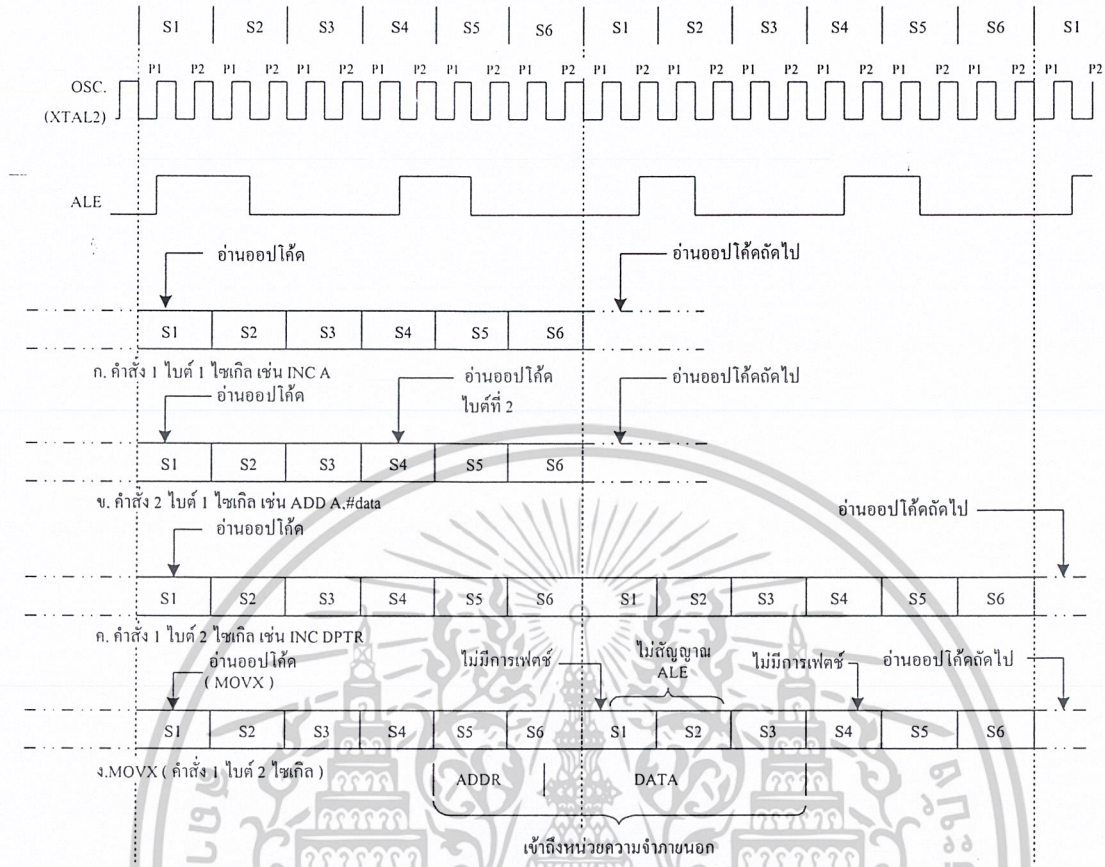
2.4.6 จังหวะการทำงานของไมโครคอนโทรลเลอร์ MCS-51

ในการใช้งานไมโครคอนโทรลเลอร์ MCS-51จะต้องทำความเข้าใจถึงจังหวะการทำงานของซีพียูและลำดับขั้นตอนการประมวลผลคำสั่ง ในการประมวลผลคำสั่งของซีพียูจะมีขั้นตอนหลักๆ 2 ขั้นตอน คือ กระบวนการเฟตช์ (fetch) เป็นการเรียกคำสั่งออกจากหน่วยความจำโปรแกรมแล้วทำการแปลงรหัสคำสั่งนั้นเป็นภาษาเครื่องเพื่อเตรียมการประมวลผล ขั้นตอนต่อมาคือ กระบวนการเอ็กซ์คิวต์ (execute) เป็นการกระทำตามคำสั่งที่กำหนดหรือตามเฟตช์ที่ขึ้นมาโดยกระบวนการก่อนหน้านี้ เมื่อทำการเอ็กซิคิวต์คำสั่งเรียบร้อยแล้วก็จะไปเริ่มกระบวนการเฟตช์คำสั่งใหม่ต่อไป

เมื่อเริ่มจ่ายไฟให้แก่ไมโครคอนโทรเลอร์จะเกิดการรีเซตในลักษณะที่เรียกว่า เพาเวอร์ออนรีเซต (power on reset) ซีพียูเริ่มต้นการทำงานที่แอดเดรส 0000H ของหน่วยความจำโปรแกรม จังหวะการทำงานของซีพียูจะเป็นไปตามรูปแบบ โดยได้รับการกำหนดมาจากรอบการทำงานหรือแมชชีนไซเกิล (machine cycle) ในรูปที่ 2.23 เป็นไดอะแกรมเวลาแสดงจังหวะการทำงานของไมโครคอนโทรลเลอร์ MCS-51 โดยใน 1 รอบการทำงานหรือแมชีนไซเกิลจะแบ่งย่อยออกเป็น 6 สเตต (state) กำหนดชื่อเป็น S1-S6 ในแต่ละสเตตมีค่าเวลาเท่ากับ 2 คาบเวลาของสัญญาณนาฬิกา ถ้าสัญญาณนาฬิกามีความถี่ 12 MHz จะมีคาบเวลาเท่ากับ 1ms คาบเวลาทั้งสองภายในหนึ่งสเตตจะเรียกว่า เฟส 1(phase 1)และเฟส 2 (phase 2)

ในรูปที่ 2.22 (ก) และ 2.22(ข) จะเป็นการเอ็กซิคิวต์คำสั่งที่ใช้เวลา 1 ไซเกิล เริ่มต้นที่สเตต 1 จะเป็นการอ่านค่าออปโค้ด อันเป็นกระบวนการแลตช์ค่าของออปโค้ดส่งไปให้รีจิสเตอร์คำสั่ง (Instruction Register : IR) การเฟตช์ครั้งที่ 2 จะเกิดขึ้นที่สเตต 4 ภายในแมชีนไซเคิลเดียวกัน ในกรณีที่เป็นคำสั่งไบต์เดียว การเฟตช์ครั้งที่ 2 ภายในแมชีนไซเกิลเดียวกันจะถูกตัดทิ้งไป ในคำสั่งที่มีใช้เวลา 1 ไซเกิล จะสิ้นสุดการทำงานลงในสเตต 6 ของแมชีนไซเกิลเดียวกัน

ในกรณีที่คำสั่งใช้เวลา 2 ไซเกิล การทำงานของคำสั่งนั้นจะสิ้นสุดลงในสเตต 6 ของแมชีนไซเกิลสองดังไดอะแกรมรูปที่ 2.22 (ค) สำหรับในการกระทำคำสั่ง MOVX ซึ่งเป็นคำสั่งขนาด 1ไบต์ 2 ไซเกิล จะไม่มีการเฟตช์เกิดขึ้นในไซเกิลที่สองของคำสั่ง MOVX นี้ เนื่องจากซีพียูจะไปทำการติดต่อกับหน่วยความจำภายนอกดังแสดงในไดอะแกรมรูปที่ 2.22 (ง) จะเห็นได้ว่า เวลาในการเอ็กซิคิวต์จะไม่ได้ขึ้นอยู่กับว่าทำการติดต่อกับหน่วยความจำโปรแกรมภายใน



รูปที่ 2.23 แสดงไซเกิลการทำงานของไมโครคอนโทรลเลอร์ MCS-51 แบบเฟรช

ช่วงจังหวะเวลาของ CPU

Machine cycle ประกอบด้วย 6 สถานะ หรือเท่ากับ 12 คาบของออสซิลเลเตอร์ แต่ละสถานะจะแบ่งเป็นเฟส (P1) ดังรูป 2.23 ครึ่งหนึ่งเป็นช่องเฟส 1 แอคทีฟ และ เฟส 2 (P2) เป็นช่องเฟส 2 แอคทีฟ ดังนั้นในแต่ละ Machine cycle จะประกอบด้วย 12 คาบออสซิลเลเตอร์เป็นจำนวน S1P1 คือ สถานะที่ 1 เฟสที่ 1 ถึง S6P2 คือสถานะที่ 6 เฟสที่ 2 โดยปกติการทำงานแบบคณิตศาสตร์ และตรรกศาสตร์จะทำในช่องเฟส 1 และการถ่ายเทข้อมูลภายในระหว่างที่รีจีสเตอร์จะทำในช่องเฟส 2

จากรูปที่ 2.23 แสดงถึงช่วงเวลา fetch และการทำงานที่อ้างถึงลักษณะภายใน และเฟส เนื่องจากสัญญาณนาฬิกาภายใน ผู้ใช้ไม่สามารถที่จะควบคุมการเข้าถึงภายในได้ตามปกติ ALE จะ แอ็คทีฟ 2 ครั้งในแต่ละ machine cycle และจะเกิดขึ้นระหว่าง S1P2 ถึง S2P1 ครั้งหนึ่งระหว่าง S4P2 ถึง S5P1 อีกครั้งหนึ่ง

การทำงานของแต่ละ machine cycle จะเริ่มที่ S1P2 เมื่อ opcode เก็บเข้าในตัวรีจีสเตอร์ ถ้าสั่งหรืออ่าน opcode เข้ามา ถ้าคำสั่งมี 2 ไบต์ ไบต์ที่ 2 จะถูกอ่านในช่วง S4 ภายใน machine cycle เดียว

กัน แต่ถ้าเป็น 1 ไบต์ คำสั่งจะยังคง fetch ที่ S4 แต่ไบต์ที่ถูกอ่าน (ซึ่งควรเป็นไบต์ที่ 2 ของคำสั่งเดียวกัน) จะไม่มีผล และตัว PC จะยังไม่เพิ่มค่าไม่ว่ากรณีใด ๆ

คำสั่ง MCS-51 ส่วนใหญ่จะทำงานในช่วง 1 machine cycle ยกเว้นคำสั่ง MUL (คูณ) Div (หาร) ที่ใช้มากกว่าสอง machine cycle ที่จะทำงานให้สมบูรณ์ได้จะใช้ถึงสี่ machine cycle

2.4. 7 การเขียนไปยังพอร์ต

การทำงานตามคำสั่งที่เปลี่ยนค่าใน latch ของแต่ละ Port ค่าใหม่จะเข้ามาเก็บในช่วงระหว่าง S6P2 ของ machine cycle สุดท้ายของคำสั่งอย่างไรก็ตาม Port จะเก็บค่าใน latch เมื่อมีการใช้ส่งข้อมูลออกที่ buffer output ระหว่าง phase 1 ของคาบบอกเวลาใด ๆ ของสัญญาณนาฬิกา (ส่วนระหว่าง phase 2 buffer output จะยังคงเก็บค่าเริ่มแรกที่ปรากฏใน phase 1 ก่อนหน้านั้น) โดยลำดับค่าใหม่ที่ latch ไว้ จะยังไม่ปรากฏที่ขา port จนกว่าจะถึง phase 1 ตัวใหม่ ซึ่งอยู่ในช่วง S1P1 ของ machine cycle ตัวต่อมา

ที่ได้กล่าวมาเป็นเพียงคุณสมบัติเบื้องต้นที่จำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจเสียก่อนเพื่อเป็นการกำหนดวงจร และแนวทางการที่จะเขียนโปรแกรมสั่งงาน และคุณสมบัติพิเศษที่แตกต่างกัน เฉพาะบางเบอร์ของตระกูล MCS-51

2.5 การเชื่อมโยง 8255 กับ MCS-51

8255A Programmable Peripheral Interface เป็นชิพขนาด 40 ขา มีอยู่ 3 พอร์ท คือ พอร์ท A พอร์ท B และพอร์ท C เป็นพอร์ท 8 บิทที่สามารถโปรแกรมให้เป็นอินพุทหรือเอาท์พุทก็ได้ โดยที่พอร์ท C ยังแบ่งเป็น 4 บิทล่างและ 4 บิทบน โดยมีโครงสร้างตามรูปที่ 2.24

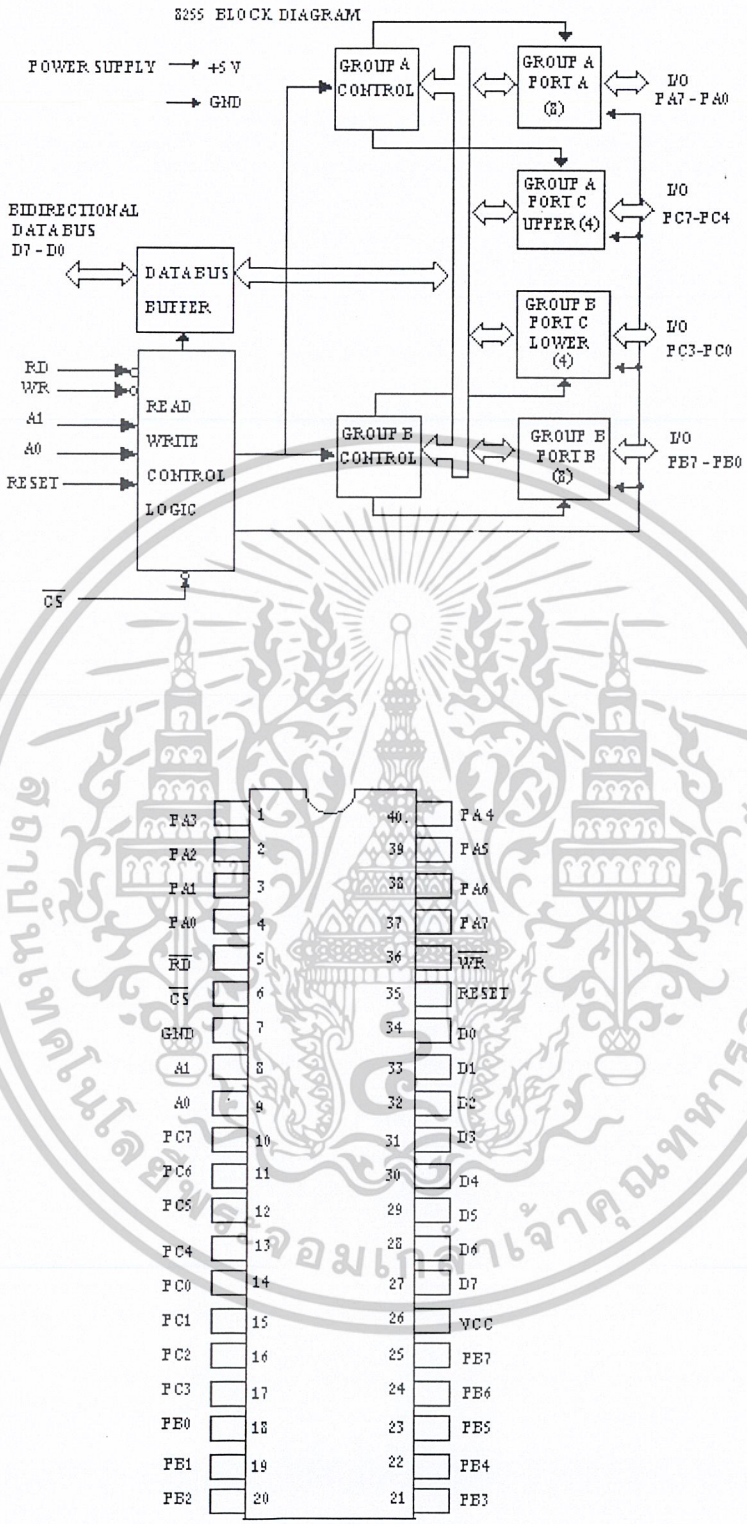
2.5.1 โหมดการทำงานของ 8255

การทำงานมีอยู่ด้วยกัน 3 โหมด ดังตารางที่ 2.4

โหมด 0 มีการทำงานแบบ BASIC I/O ไม่มี handshake

โหมด 1 โหมดนี้ใช้พอร์ท A B ในการรับหรือส่งข้อมูล และใช้พอร์ท C ในการตรวจสอบสัญญาณ (handshake)

- โหมด 2 โหมดนี้ใช้พอร์ท A ในการรับส่งข้อมูล 2 ทิศทางและพอร์ท B ในการรับหรือส่งข้อมูลและใช้พอร์ท C บิต 0 บิต 1 บิต 2 ในการรับส่งข้อมูลบิตและบิต 3,4,5,6,7 เป็นสัญญาณ handshake



รูปที่ 2.24 แสดงการจัดวางขาและโครงสร้างของ 8255

	MODE 0		MODE 1		MODE 2
	IN	OUT	IN		GROUP A ONLY
PA_0	IN	OUT	IN	OUT	⟷
PA_1	IN	OUT	IN	OUT	⟷
PA_2	IN	OUT	IN	OUT	⟷
PA_3	IN	OUT	IN	OUT	⟷
PA_4	IN	OUT	IN	OUT	⟷
PA_5	IN	OUT	IN	OUT	⟷
PA_6	IN	OUT	IN	OUT	⟷
PA_7	IN	OUT	IN	OUT	⟷
PB_0	IN	OUT	IN	OUT	———
PB_1	IN	OUT	IN	OUT	———
PB_2	IN	OUT	IN	OUT	———
PB_3	IN	OUT	IN	OUT	———
PB_4	IN	OUT	IN	OUT	———
PB_5	IN	OUT	IN	OUT	———
PB_6	IN	OUT	IN	OUT	———
PB_7	IN	OUT	IN	OUT	———
PC_0	IN	OUT	$INTR_B$	$\overline{INTR_B}$	I/O
PC_1	IN	OUT	IBF_B	$\overline{OBF_B}$	I/O
PC_2	IN	OUT	$\overline{STB_B}$	$\overline{ACK_B}$	I/O
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$
PC_4	IN	OUT	$\overline{STB_A}$	I/O	$\overline{STB_A}$
PC_5	IN	OUT	IBF_A	I/O	IBF_A
PC_6	IN	OUT	I/O	$\overline{ACK_A}$	$\overline{ACK_A}$
PC_7	IN	OUT	I/O	$\overline{OBF_A}$	$\overline{OBF_A}$

ตารางที่ 2.4 แสดงสรุปโหมดต่าง ๆ ของ 8255

ขาสัญญาณต่าง ๆ ของ 8255

D_7 - D_0 บัสข้อมูลเชื่อมโยงกับ CPU

A_7 - A_0 ใช้เลือกพอร์ท A,B,Cและพอร์ทควบคุม

RESET เมื่อขานี้ได้รับสัญญาณกระตุ้นลอจิก 1 จะทำให้ 8255 ถูกรีเซ็ทมีผลทำให้ทุกพอร์ทเป็นอินพุททันที

PA_7 - PA_0 เป็นพอร์ทขนาน 8 บิท

PB_7 - PB_0 เป็นพอร์ทขนาน 8 บิท

PC_7 - PC_0 เป็นพอร์ทขนาน 8 บิท

$\overline{RD}$ ในการอ่านข้อมูลที่พอร์ทของ 8255 ต้องทำให้ขานี้เป็นลอจิก 0 พร้อมกับ $\overline{CS}$

$\overline{WR}$ ในการเขียนข้อมูลหรือโปรแกรมลงบน 8255 ต้องทำให้ขานี้ เป็นลอจิก 0 พร้อมกับ $\overline{CS}$

$\overline{CS}$ เป็นขาเลือกชิพ 8255 ได้ ขานี้โดยต่อกับ I/O DECODER

เมื่อขา $\overline{WR}$, $\overline{RD}$,A0,A1,$\overline{CS}$ ทำงานทั้ง 5 ขาจะมีฟังก์ชั่นการทำงานดังตารางที่ 2.5

A1	A0	RD	WR	CS	
					Input operation (READ)
0	0	0	1	0	Port A → data bus
0	1	0	1	0	Port B → data bus
1	0	0	1	0	Port C → data bus
					Output operation (WRITE)
0	0	1	0	0	Data bus → Port A
0	1	1	0	0	Data bus → Port B
1	0	1	0	0	Data bus → Port C
1	1	1	0	0	Data bus → Control
					Disable function
X	X	X	X	1	Data bus → 3-state
1	1	0	1	0	Illegal condition
X	X	1	1	0	Data bus → 3-state

ตารางที่ 2.5 แสดงฟังก์ชั่นการทำงานของ 8255

การถอดรหัสตำแหน่งพอร์ทของ 8255 จะได้เบอร์พอร์ทดังนี้

(8000H – 8FFFH)

(9000H – 9FFFH)

(A000H – AFFFH)

(B000H – BFFFH)

(C000H – CFFFH)

(D000H – DFFFH)

(E000H – EFFFH)

(F000H – FFFFH)

และใช้เอาท์พุทที่ตำแหน่ง (F000H – FFFFH) มาถอดรหัสร่วมกับ A8 ,A9, A10 โดยใช้ 74 LS 138 ได้พอร์ท

(F800H – F8FFH)

(F900H – F9FFH)

(FA00H – FAFFH)

(FB00H – FBFFH)

(FC00H – FCFFH)

(FD00H – FDFFH)

(FE00H – FEFFH)

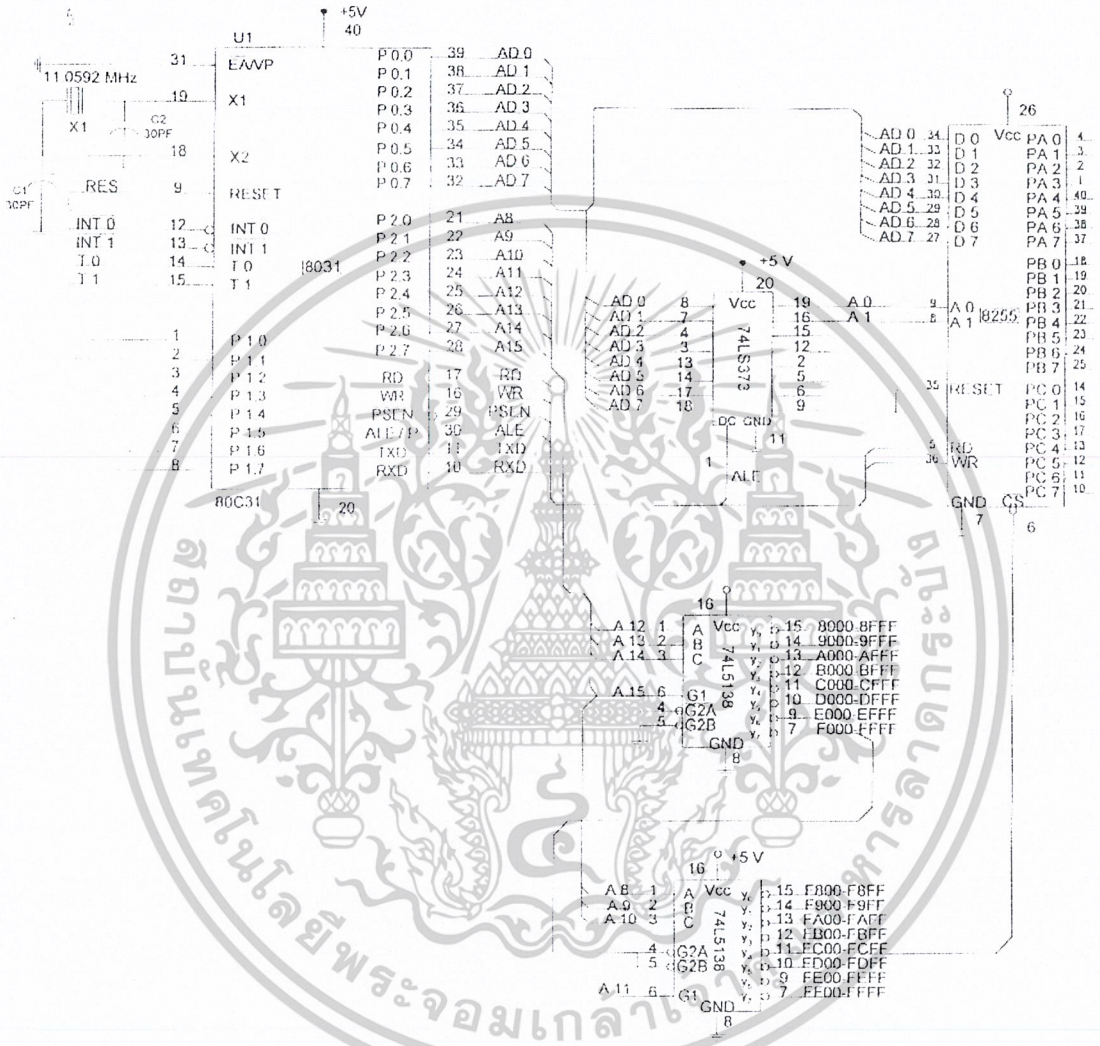
(FF00H – FFFFH)

สัญญาณเลือกชิพ 8255 ใช้เบอร์พอร์ท (FC00H – FCFFH) และ A_0 ,A_1 ต่อเข้า A0, A1 ของ 8255 จะได้เบอร์พอร์ทของ 8255 ดังตารางที่ 2.6

I/O ADDRESS	8255 (PORT)
FC00H	A
FC01H	B
FC02H	C
FC03H	CONTROL

ตารางที่ 2.6 แสดง I/O ADDRESS ของ 8255

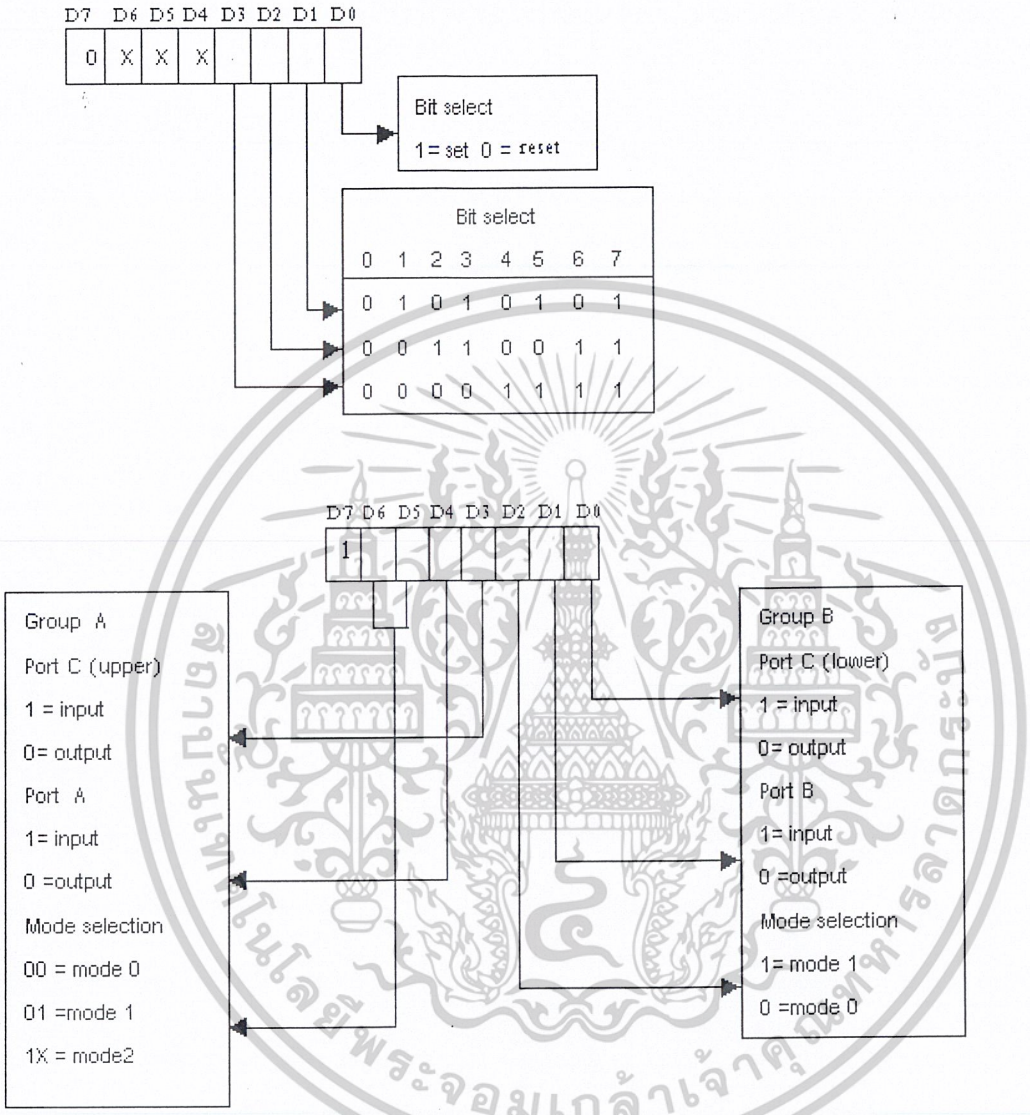
2.5.2 การเชื่อมโยง 8255 เข้ากับ CPU



รูปที่ 2.25 แสดงการเชื่อมโยง 8255 เข้ากับ CPU

2.5.3 การโปรแกรม 8255

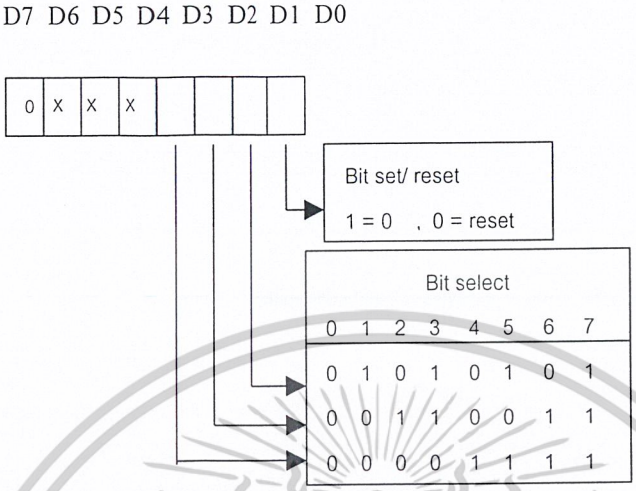
จะใช้ตารางการโปรแกรมดังรูปที่ 2.26



รูปที่ 2.26 แสดง CONTROL WORDS ทั้ง 2 แบบของ MODE และ BIT DEFINITION FORMAT

The Bit Set / Reset Mode

นอกจากเราจะใช้พอร์ท A,B,C ในการโปรแกรมให้เป็นอินพุท/ เอาท์พุทแล้วเรายังสามารถที่จะโปรแกรมพอร์ท C บิต PC0 – PC7 ให้เป็นลอจิก 0 หรือ 1 (ใช้งานเป็นเอาท์พุท) เพื่อใช้เป็นสัญญาณ strobe ได้ วิธีการโปรแกรมพอร์ท C ดังแสดงในรูปที่ 2.27

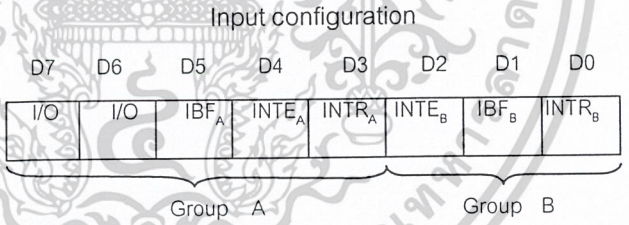


รูปที่ 2.27 แสดงการโปรแกรมบิทของพอร์ท C (ใช้เป็นเอาท์พุทเท่านั้น)

MODE 1 : Strobe I/O

โหมดนี้จะใช้พอร์ท A,B ในการส่งและรับข้อมูลและใช้พอร์ท C ตรวจสอบความพร้อม เมื่อโปรแกรมแล้ว พอร์ท C จะระบุเป็นขาสัญญาณดังต่อไปนี้

เมื่อเป็นอินพุทพอร์ท



$PC_0 = INTR_B$

$PC_1 = IBF_B$

$PC_2 = INTE_B$

$PC_3 = INTR_A$

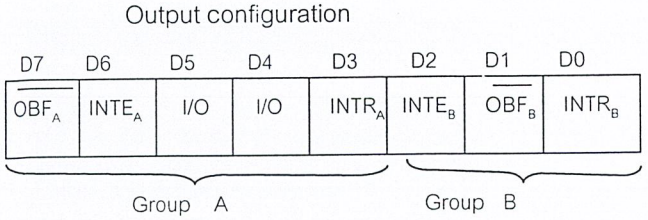
$PC_4 = INTE_A$

$PC_5 = IBF_A$

$PC_6 = I/O$

$PC_7 = I/O$

เมื่อเป็นเอาท์พุทพอร์ท



$PC_0 = INTR_B$

$PC_1 = \overline{OBF_B}$

$PC_2 = INTE_B$

$PC_3 = INTR_A$

$PC_4 = I/O$

PC_5 = I/O

PC_6 = $INTE_A$

PC_7 = OBF_A

โดยที่ INTR คือ Interrupt request (การร้องขออินเตอร์รัพท์)

IBF คือ Input Buffer full (บัฟเฟอร์เต็ม)

$\overline{STB}$ คือ Strobe(สโตรบ)

I/O คือ Input/ Output (อินพุท/เอาท์พุท) โปรแกรมได้

INTE คือ Interrupts Enabled (อินเตอร์รัพท์ อีเนเบิล)

$\overline{ACK}$ คือ Acknowledge

พอร์ท	ชนิด	ชื่อ	คอนโทรลด้วยบิทเซ็ท/รีเซ็ทของ
A	Input	INTEA	PC4
A	Output	INTEA	PC6
B	Input	INTEB	PC2
B	Output	INTEB	PC2

ตารางที่ 2.7 แสดงการโปรแกรม INTE ของพอร์ท A,B

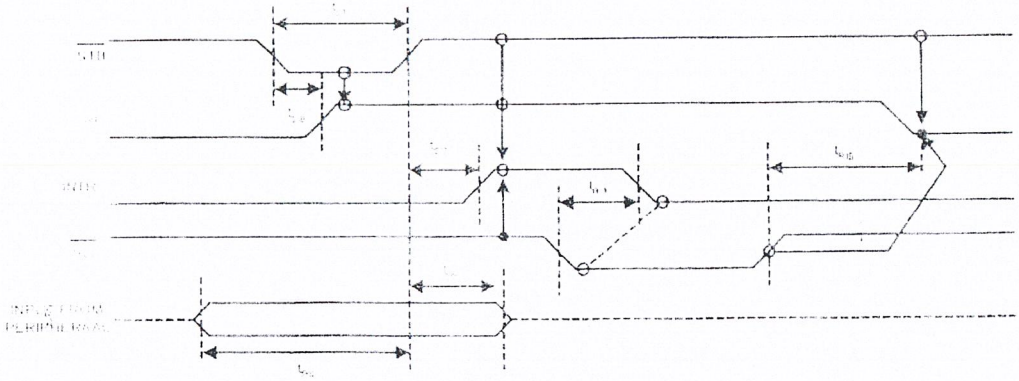
ผังเวลา (โหมด1) อินพุทพอร์ท

เมื่อพอร์ท A, B ได้ถูกโปรแกรมเป็นพอร์ทในการรับส่งข้อมูลแล้ว จะต้องใช้สัญญาณควบคุม 3 สัญญาณคือ IBF, $\overline{STB}$,INTR

IBF เป็นขาเอาท์พุทแอคทีฟที่ 1 แสดงถึงอินพุทบัฟเฟอร์ เต็ม (เมื่อ CPU อ่านข้อมูลจากพอร์ทอินพุทไปแล้วจะทำให้ขานี้เปลี่ยนเป็นลอจิก 0 คือ ว่างพร้อมที่จะรับข้อมูลใหม่

$\overline{STB}$ เป็นขาอินพุทแอคทีฟที่ 0 เป็นขาที่อุปกรณ์ภายนอกส่งมากระตุ้นให้อินพุทพอร์ทแลทช์ข้อมูลไว้

INTR เป็นขาเอาท์พุทแอคทีฟที่ 1 จะแอคทีฟหลังจากที่ได้รับ $\overline{STB}$ แอคทีฟ 0 ช่วงที่ $\overline{STB}$ เปลี่ยนจากระดับต่ำไปสูง ขา INTR จะแอคทีฟทันที (ต้องอีเนเบิล อินเตอร์รัพท์ ฟลิปฟลอปไว้ ก่อน) โดยเซ็ทที่ PC4 ถ้าใช้งานที่พอร์ท A และเซ็ทที่ PC2 ถ้าใช้งานพอร์ท B



รูปที่ 2.28 แสดงผังเวลา (โหมด 1) อินพุทพอร์ท

จากผังเวลาจะเห็นว่า ก่อนที่อุปกรณ์ภายนอกจะส่ง $\overline{\text{STB}}$ มาเข้า 8255 จะต้องตรวจสอบขา IBF ว่าว่างหรือเปล่า (0 คือว่าง) ก็จะส่งข้อมูล 8 บิท มาตามหลังด้วย $\overline{\text{STB}}$ แอคทีฟที่ 0 หลังจากนั้นไม่นาน IBF ก็จะเป็น 1 แสดงว่าข้อมูลถูกแลทซ์ ไว้ได้แล้ว ช่วงที่ $\overline{\text{STB}}$ เปลี่ยนกลับเป็น 1 ช่วงนี้เอง INTR ก็จะแอคทีฟ (มีลอจิก 1) สัญญาณนี้ถูกต่อกับขา INT ของ CPU ก็จะกระโดดไปโปรแกรมบริการอินเตอร์รัพท์ ภายในโปรแกรมบริการอินเตอร์รัพท์ จะมีการอ่านข้อมูลไปเก็บ เมื่ออ่านข้อมูลไปเก็บเสร็จแล้ว IBF ก็จะว่างลงอีกครั้ง (เป็นลอจิก 0)

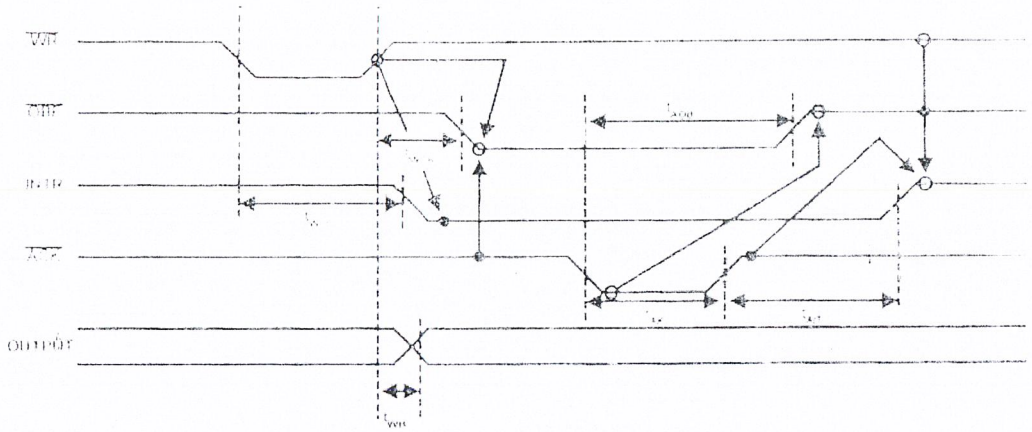
ผังเวลา (โหมด 1) เอาท์พุทพอร์ท

เมื่อพอร์ท A,B ได้ถูกโปรแกรมในโหมด 1 เพื่อใช้ในการส่งข้อมูล 8 บิต พอร์ท C จะถูกกำหนดให้เป็นพอร์ทในการตรวจสอบสัญญาณ (Handshake) สัญญาณที่ใช้มี 3 สัญญาณคือ $\overline{\text{OBF}}$, $\overline{\text{ACK}}$, INTR

$\overline{\text{OBF}}$ เป็นขาเอาท์พุทแอคทีฟ 0 เมื่อ CPU ส่งข้อมูลออกมาที่พอร์ทของ 8255 แล้วขานี้ จะเป็น 0 หลังจากช่วงขอบขาขึ้นของ $\overline{\text{WR}}$ และจะเปลี่ยนเป็น 1ในช่วงขอบขาขึ้นของ $\overline{\text{ACK}}$ (ขา $\overline{\text{OBF}}$ ถ้ามีลอจิก 0 หมายถึงเอาท์พุทบัฟเฟอร์เต็ม ถ้าเป็น 1 หมายถึงว่าง)

$\overline{\text{ACK}}$ เป็นขาอินพุทแอคทีฟที่ 0 เป็นสัญญาณตอบรับอุปกรณ์ภายนอกส่งมายัง 8255 หลังจากที่อินพุทพอร์ทเก็บข้อมูลได้แล้ว

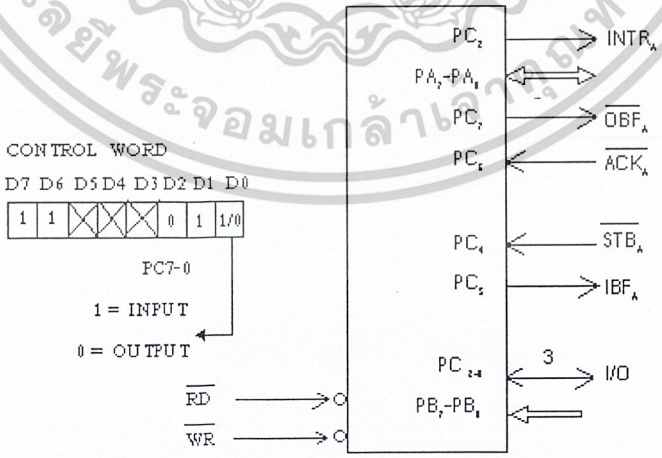
INTR เป็นขาเอาท์พุทแอคทีฟที่ 1 ส่งไปอินเตอร์รัพท์ CPU หลังจากที่ 8255 ได้รับ $\overline{\text{ACK}}$ ช่วงขอบขาขึ้นก็จะส่ง INTR แอคทีฟ 1 ออกไป (ต้องโปรแกรม INTE ไว้ก่อนถ้าเป็นพอร์ท A ต้องเซ็ทที่ PC6 และพอร์ท B ต้องเซ็ทที่ PC2)



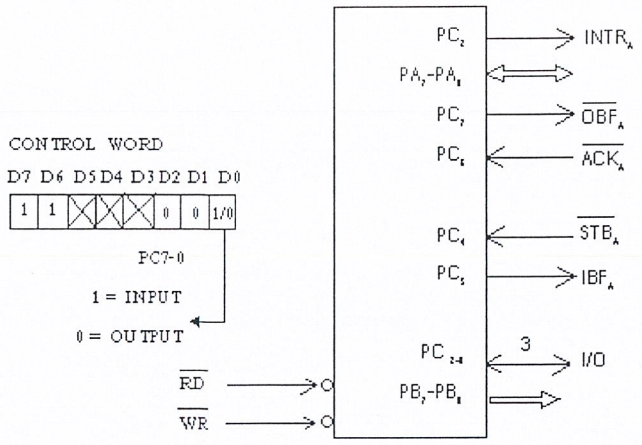
รูปที่ 2.29 แสดงผังเวลา(โหมด 1) เอาท์พุทพอร์ท

จากผังเวลาจะเห็นว่าการทำงานสามารถอธิบายได้ดังนี้ ก่อนที่ CPU จะส่งข้อมูลใหม่มายังเอาท์พอร์ท จะต้องตรวจเช็ค $\overline{OBF}$ ก่อนถ้า $\overline{OBF}$ = 1(คือว่าง) ก็สามารถส่งข้อมูลใหม่มาที่เอาท์พุทพอร์ทได้ขณะการส่งข้อมูลมาสัญญาณ $\overline{WR}$ จะแอคทีฟ 0 และช่วง $\overline{WR}$ เปลี่ยนเป็น 1 ช่วงนี้เองจะทำให้ $\overline{OBF}$ และ INTR เป็น 0 อยู่ช่วงหนึ่งจนกว่าจะมีสัญญาณ $\overline{ACK}$ (สัญญาณตอบรับจากอุปกรณ์ภายนอกบอกว่ารับข้อมูลไว้แล้วหลังจาก $\overline{ACK}$ เป็นสถานะต่ำทำให้ $\overline{OBF}$ เป็น 1 คือเอาท์พุทบัฟเฟอร์จะว่างลงอีกครั้งเมื่อ $\overline{ACK}$ เปลี่ยนเป็น 1 ช่วง leading edge จะทำให้ INTR เปลี่ยนเป็น 1 ด้วยช่วงนี้เองโปรแกรมจะกระโดดไป ISR (Interrupt Service Routine) เพื่อจะได้นำข้อมูลใหม่มาที่เอาท์พุทพอร์ท

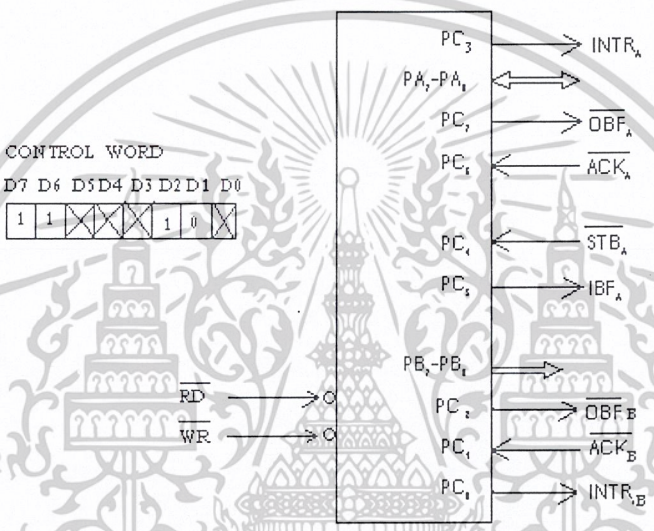
8255 (โหมด 2) ใช้พอร์ท A รับส่งข้อมูลแบบบัส 2 ทิศทาง พอร์ท B เป็น ไอโอพอร์ท และพอร์ท C ใช้สำหรับรับส่งสัญญาณตรวจสอบความพร้อม มีตาราง Control Word และผังเวลาดังรูปที่ 2.30-2.33



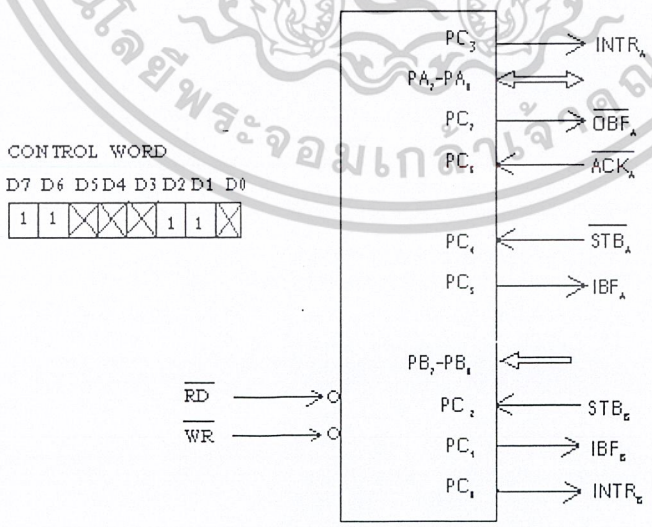
รูปที่ 2.30 แสดงตาราง Control Word ของ mode 2 และ mode 0 (input)



รูปที่ 2.31 แสดงตาราง Control Word mode 2 และ mode 0 (output)



รูปที่ 2.32 แสดงตาราง control word ของ mode 2 และ mode 1 (output)



รูปที่ 2.33 แสดงตาราง control word ของ mode 2 mode 1 (input)

2.6 In-System Programming (ISP)

ถ้าจะกล่าวว่า Flash Microcontroller คือ เรื่องที่เด่นที่สุดสำหรับวงการช่วงก่อนปี 2000 อีกเรื่องที่ถือว่าเด่นไม่แพ้กันในช่วงหลังปี 2000 ก็คือ ISP คำว่า ISP ย่อมาจาก In-System Programming และ IAP ย่อมาจาก In-Application Programming ซึ่งเป็นสิ่งที่เกิดมาในช่วงประมาณไม่กี่ปีมานี้เอง กล่าวคือ ตัวชิพไมโครคอนโทรลเลอร์ที่เป็น Flash Memory ได้ถูกพัฒนาให้ขบวนการอัดโปรแกรม ทำได้ง่ายขึ้นกว่าเดิมอีกแทนที่จะต้องนำมาเข้าเครื่องโปรแกรมทั่ว ๆ ไป (เรียกว่าการโปรแกรมแบบ Parallel) เราสามารถอัดโปรแกรมผ่านทางขาสัญญาณเพียงไม่กี่เส้น และใช้การส่งข้อมูลในแบบ Serial แทน นั้นหมายความว่า ตัวชิพที่ถูกบัดกรีลงไปในบอร์ดแล้ว ก็ยังสามารถถูกนำมาอัดโปรแกรมใหม่ได้ โดยขอให้มีขาสัญญาณดังกล่าวต่อออกมาได้จึงเรียกว่าเป็น ISP นั่นเอง และด้วยคุณสมบัติเดียวกันนี้ บางครั้งโปรแกรมที่พัฒนาอยู่บนบอร์ดเหล่านั้น อาจจะต้องการให้มีการเก็บข้อมูลต่าง ๆ เอาไว้ ซึ่งเราก็สามารถทำโปรแกรมให้เกิดขบวนการรับข้อมูลเพื่อเก็บลงใน Flash Memory ภายในตัวได้ ซึ่งเรียกว่าเป็น IAP นั่นเอง

มีผู้ผลิตชิพมากมายที่พัฒนาชิพไมโครคอนโทรลเลอร์ของตัวเอง ให้มีคุณสมบัติแบบ ISP นี้ ไม่ว่าจะเป็นตระกูล MCS-51 หรือ PIC (Microchip) หรือ อื่น ๆ อีกมากมาย และน่าจะกล่าวได้ว่าถนนทุกสายของการพัฒนาตัวชิพไมโครคอนโทรลเลอร์ กำลังวิ่งเข้าสู่รูปแบบของ ISP กันทั่วหน้า ตัวอย่างเช่น

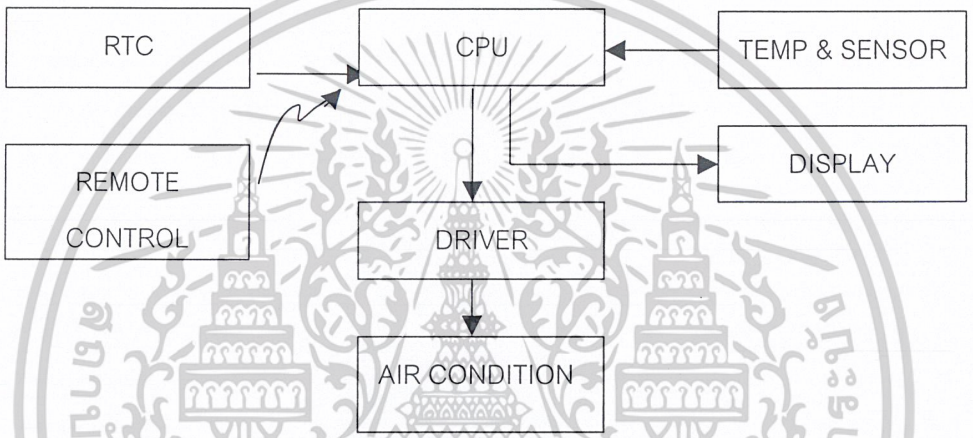
ชิพเบอร์ 89S8252 (ตระกูล MCS-51) ของ Atmel ใช้ขาสัญญาณในการโปรแกรม 4 เส้น เรียกว่า SPI (Serial Peripheral Interface) คือ P1.4(/SS) P1.5 (MOSI) P1.6 (MISO) P1.7 (SCK) การโปรแกรมกระทำด้วยระดับแรงไฟ 5VDC เท่านั้น วิธีการโปรแกรมแบบนี้ช่วยให้สะดวกขึ้นมาก โดยเราอาจจะเขียนโปรแกรมบนเครื่อง PC เพื่อให้เกิดสัญญาณในการโปรแกรมทั้ง 4 เส้น และต่อเข้า Parallel Port ของเครื่อง PC โดยตรงก็ได้ หรืออาจจะทำเป็นตัวช่วยอัดโปรแกรมเล็ก ๆ ต่างหาก โดยรับข้อมูลจากเครื่อง PC ทาง RS232 และนำข้อมูลมาสร้างสัญญาณดังกล่าว เพื่อโปรแกรมลงสู่ตัวชิพอีกที

ชิพเบอร์ 89C51RD+, 89C51RD2 (ตระกูล MCS-51) ของ Phillips ใช้ขาสัญญาณในการโปรแกรม 2 เส้น โดยเป็น Serial Port (RS232) มาตรฐานนั่นเอง คือ P3.0 (RX) P3.1 (TX) การโปรแกรมต้องใช้แรงไฟ 12VDC ด้วย แต่ถ้าเป็นเบอร์ RD2 ก็ใช้เพียงแค่ 5VDC เท่านั้น และเบอร์นี้ยังมีจุดเด่นคือ สามารถทำงานได้เร็วกว่า 1 เท่าด้วย Xtal ค่าเดิม กล่าวคือ 1 คำสั่งของ MCS-51 ทั่วไปจะใช้สัญญาณนาฬิกา 12 Clock แต่เบอร์ RD2 จะใช้เพียง 6 Clock เท่านั้น การโปรแกรมผ่านทาง Serial Port มาตรฐานนี้ จะทำให้ขบวนการต่าง ๆ ทำได้ง่ายมาก เพราะสามารถต่อโดยตรงเข้ากับเครื่อง PC ได้ทันที (ยังคงต้องผ่านชิพปรับแรงไฟ) และผู้ผลิตก็มักจะแจกโปรแกรมเพื่อการติดต่อกับตัวไมโครคอนโทรลเลอร์มาให้อยู่แล้ว หรือไม่ก็สามารถใช้กับโปรแกรมประเภท Terminal Emulator ได้เลย

บทที่ 3
การคำนวณและการสร้าง

3.1 บล็อกไดอะแกรมของโครงงาน

หลักการและการทำงานของส่วนควบคุม เพื่อควบคุมระบบการทำงานของเครื่องปรับอากาศ ในโครงงานนี้จะออกแบบและสร้างส่วนที่ควบคุมเท่านั้น ซึ่งก็สามารถนำไปติดตั้งกับระบบบางระบบได้ และมีข้อจำกัดอยู่เช่นกัน เพื่อเป็นที่เข้าใจง่ายมากยิ่งขึ้น จะขออธิบายโดยแยกส่วนของการทำงานแต่ละภาค โดยสามารถแบ่งแยกเป็นบล็อกไดอะแกรมดัง รูปที่ 3.1



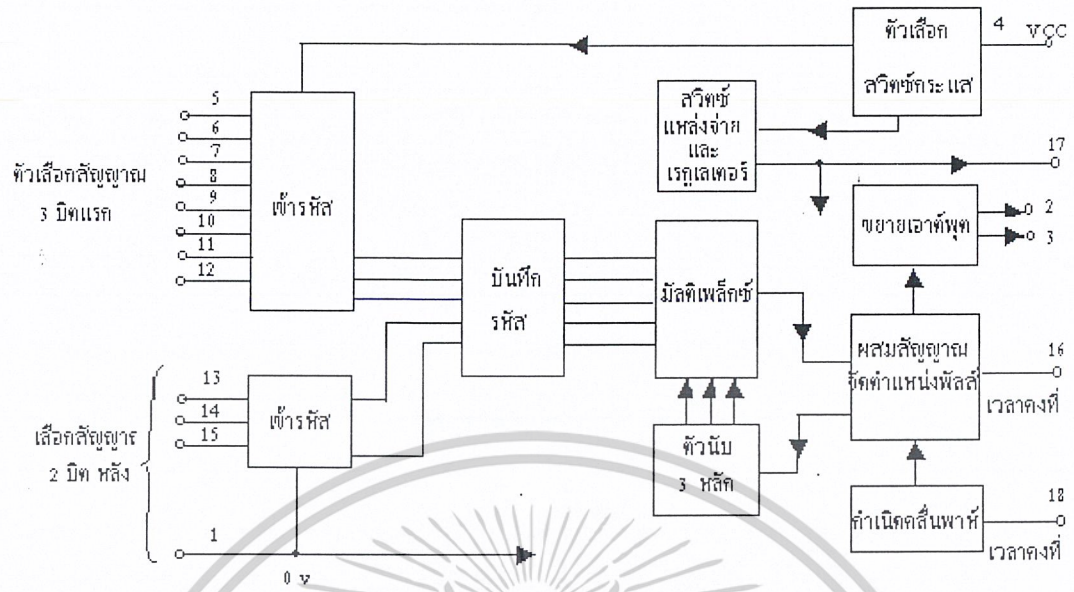
รูปที่ 3.1 แสดงบล็อกไดอะแกรม

จากรูปที่ 3.1 จะเห็นว่า CPU จะเป็นหัวใจในการทำงานของระบบ โดยรับอินพุตมาจาก RTC, REMOTE CONTROL และ ชุดตรวจจับอุณหภูมิ แล้วประมวลผลออกไปควบคุม DRIVER และ DISPLAY ซึ่ง DRIVER จะไปควบคุมเครื่องปรับอากาศในการปรับอุณหภูมิ , ปรับ SPEED ของพัดลม , ควบคุมการ เปิด/ปิด คอมเพรสเซอร์ และ ปรับมุมของใบกระจายความเย็น

3.2 รีโมทคอนโทรล

3.2.1 รายละเอียดเกี่ยวกับไอซี

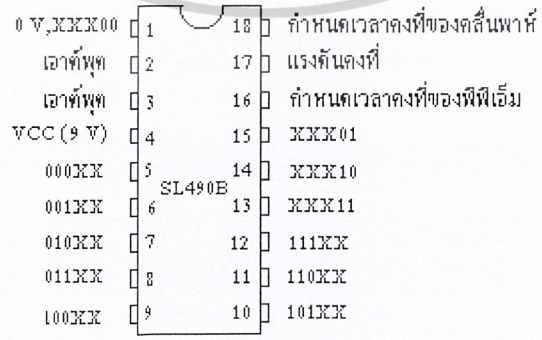
ใช้ไอซี 3 ตัว ซึ่งถูกผลิตมาให้ใช้งานร่วมกันได้คือ SL490B สำหรับภาคส่ง, SL486 และ ML926 เป็นภาครับและขยายสัญญาณ SL490B เป็นไอซีสำหรับเข้ารหัสเมื่อกดคีย์ โดยส่งออกไปเป็นลักษณะของพัลส์โพซิชั่น มอดูเลชั่น หรือพีพีเอ็ม (Pulse Position Modulation: PPM) พีพีเอ็มมีลักษณะเปลี่ยนไปตามโค้ดที่ได้จากโค้ดรีจีสเตอร์ ถ้าเป็นการส่งย่านอัลตราโซนิก จะใช้อัลตราโซนิกทรานดิวเซอร์เป็นตัวปล่อยพัลส์ และใช้ LED แทนเมื่อส่งย่านอินฟราเรด โครงสร้างภายในของ SL490B แสดงในรูปที่ 3.2



รูปที่ 3.2 แสดงบล็อกไดอะแกรมของ SL490B

สัญญาณเอาต์พุตที่ ขา 2, 3 ได้มาจากการผสมสัญญาณระหว่างความถี่ของคลื่นพาห์กับสัญญาณมัลติเพล็กซ์ได้เป็นสัญญาณพีพีเอ็มออกมาดังรูปที่ 3.3 โดย t_1 แสดงสภาวะ "1" ซึ่งแคบกว่า t_0 ที่แสดงสภาวะ "0", ส่วน t_G จะแบ่งแยกสัญญาณพีพีเอ็มแต่ละชุดออกจากกัน รูปนี้เป็นสัญญาณพีพีเอ็มสัญญาณเลข 22 ของฐานสิบ เพราะไอซีตัวนี้สามารถผลิตสัญญาณเลขฐานสองได้ 5 บิต คือ 32 ช่องสัญญาณ และรูปที่ 3.3 แสดงขาใช้งานแต่ละขา

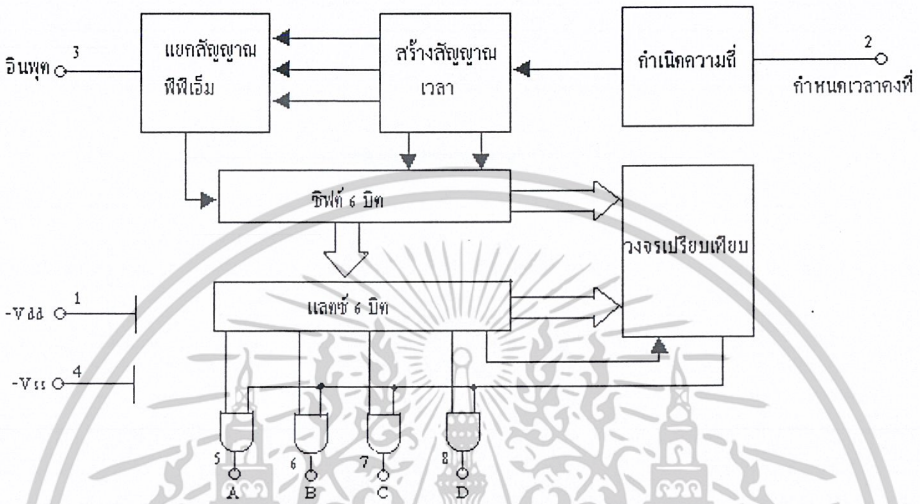
รูปที่ 3.3 แสดงสัญญาณ PPM ที่ส่งออกมา



รูปที่ 3.4 แสดงการใช้งานแต่ละขาของ SL490B

SL486 เป็นภาครับสัญญาณเข้ามาโดยผ่านไดโอดรับคลื่น แล้วทำการขยายสัญญาณพีพีเอ็มที่รับเข้ามาก่อนส่งไป ML926

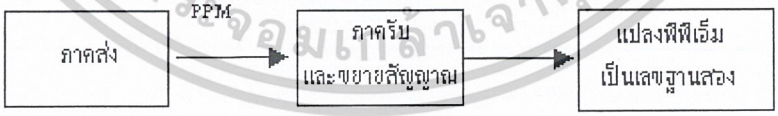
ML926 จะรับสัญญาณพีพีเอ็มเข้ามาแยกแล้วแปลงออกมาเป็นสัญญาณเลขฐานสองออกมาทาง ขา 5, 6, 7 และ 8 โครงสร้างของ ML926 ดูได้จากรูปที่ 3.5



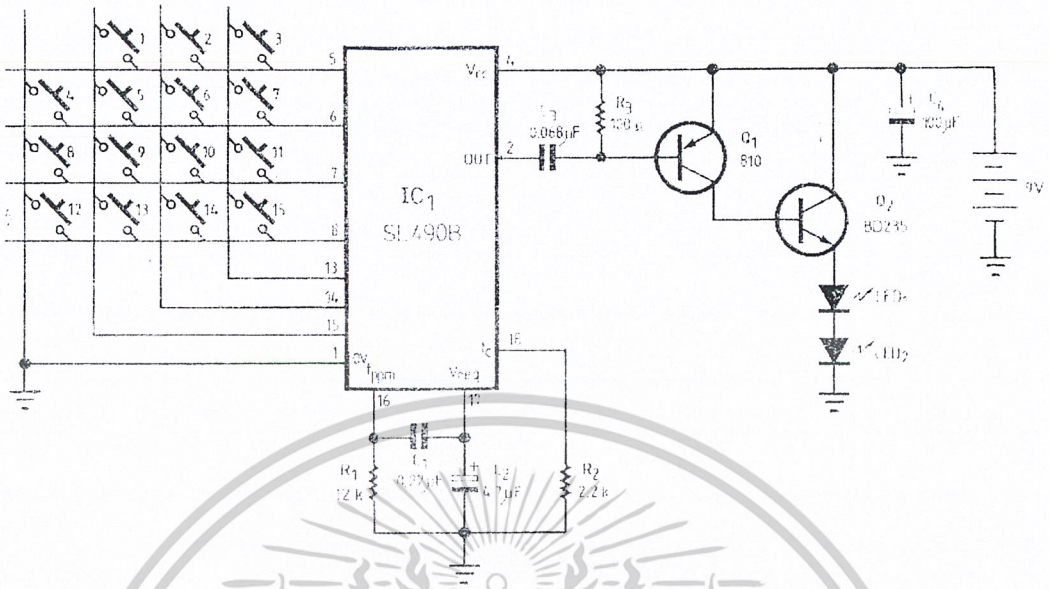
รูปที่ 3.5 แสดงบล็อกไดอะแกรมของ ML926

3.1.2 หลักการทำงาน

รูปที่ 3.6 เป็นบล็อกไดอะแกรมของทั้งระบบ มีภาคส่งสัญญาณพีพีเอ็มออกมาแล้วภาครับจะรับสัญญาณมาขยายก่อนเปลี่ยนเป็นสัญญาณเลขฐานสองเพื่อนำไปใช้งานต่อไป รูปที่ 3.7 เป็นวงจรภาคเครื่องส่ง



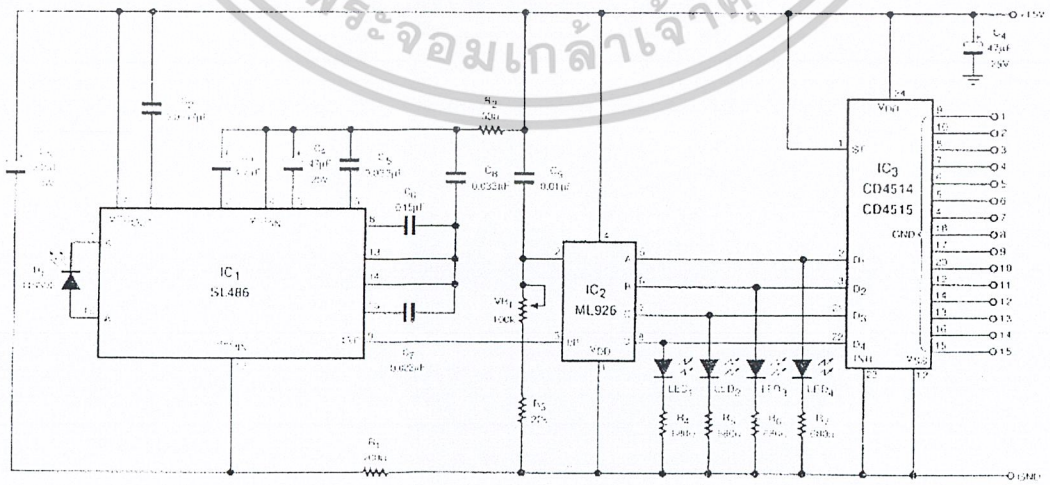
รูปที่ 3.6 แสดงหลักการทำงานเบื้องต้นของรีโมทคอนโทรล



รูปที่ 3.7 แสดงวงจรภาคส่ง 15 ช่อง ระบบอินฟราเรด

วงจรภาคส่งใช้ไฟ 9 โวลต์ ป้อนเข้าขา 4 ของ SL490B และเป็นไฟเลี้ยงให้ Q1, Q2 เมื่อขา 2 ให้สัญญาณเอาต์พุตเป็นไบแอสสำหรับ Q1 เพื่อขับ Q2 ให้ทำงาน LED1, LED2 จะส่งพัลส์อินฟราเรดออกมา โดยมี R1, R2, C1, C2 เป็นตัวกำหนดความถี่คลื่นพาห์

วงจรภาครับในรูปที่ 3.8 ใช้แหล่งจ่ายไฟ 15 โวลต์ เลี้ยง SL486, ML926 และ CD4514 มีไดโอด RS302 เป็นตัวรับแสงอินฟราเรด ซึ่งต่อเข้ากับขา 1, 16 ของ SL486 ซึ่งจะดีโค้ดและขยายสัญญาณพีพีเอ็ม แล้วส่งเข้าอินพุต ขา 3 ของ ML926



รูปที่ 3.8 แสดงวงจรภาครับ

VR, 100 kΩ ปรับค่าได้ใช้สำหรับปรับความถี่ออสซิลเลเตอร์ของ IC2 ให้ตรงกับอินพุตที่รับเข้ามา ส่วนขา 5, 6, 7, 8 ของ IC2 เป็นเอาต์พุตสัญญาณเลขฐานสอง ซึ่งจะแสดงผลผ่าน LED1-LED4 สำหรับ IC3 CD4514 (หรือ CD4515) รับสัญญาณเข้ามาถอดรหัสให้ได้เอาต์พุต 15 ช่อง ในกรณีของ CD4514 เมื่อยังไม่มีการทำงานเอาต์พุตทั้งหมดจะเป็น "0" เวลากดสวิตซ์ช่องใดช่องหนึ่ง ช่องนั้นจะมีเอาต์พุตเป็น "1" ส่วน CD4515 จะเป็นไปในทางตรงข้าม แต่ผลลัพธ์ในการควบคุมเหมือนกัน

ตรงอินพุตของ IC3 นี้ จะมีสภาวะเป็น "0" หมดทั้ง 4 อินพุต เมื่อยังไม่กดสวิตซ์เลือกช่อง ถ้าสมมติว่าเรามีสวิตซ์สำหรับเลือกช่องโดยให้สัญญาณเลขฐานสองเป็น "0000" จะมีสัญญาณพีพีเอ็มมาจากเครื่องส่ง ผ่านภาครับภาคขยาย พอออกจาก ML926 เอาต์พุตที่ได้จะเป็นสภาวะ "0000" ซึ่งเป็นสภาพปกติของอินพุตของ IC3 อยู่แล้ว ดังนั้นจึงไม่มีอะไรเกิดขึ้นทำให้เราไม่สามารถใช้งาน ช่อง 0 เป็นสัญญาณควบคุมได้นั่นเอง

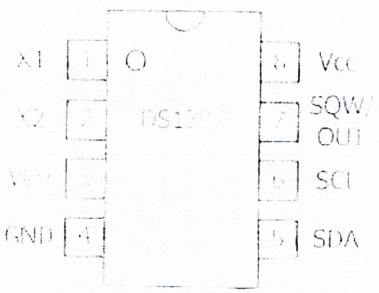
3.3 ชุดการทำงาน Real Time Clock (RTC)

ใช้ไอซีเบอร์ DS1307 ซึ่งจะให้ข้อมูลเกี่ยวกับเวลาทั้งหมด ไม่ว่าจะเป็นค่าของเวลาที่ละเอียดถึงหลักวินาที, นาที, ชั่วโมง, วันที่, วันในสัปดาห์, เดือน และปี โดยสามารถปรับวันเดือนปีให้ตรงตามปฏิทินได้อย่างถูกต้อง รวมถึงการกำหนดวันในปีอธิกสุรทินด้วย คุณสมบัติทางเทคนิคที่สำคัญดังนี้

- เป็นไอซีรีลไทม์คล็อกให้ข้อมูลตั้งแต่วินาทีจนถึงปี รวมถึงการปรับวันในปีอธิกสุรทินด้วย สามารถให้ข้อมูลเวลาได้อย่างเที่ยงตรงถึงปีคริสตศักราช 2100
- มีหน่วยความจำนอนโวลาไทล์แรม 56 ไบต์ อยู่ภายใน สามารถใช้เก็บข้อมูลทั่วไปได้
- ใช้การเชื่อมต่อแบบระบบบัส I²C
- มีวงจรตรวจจับไฟเลี้ยงต่ำหรือหายไปอย่างอัตโนมัติ และสามารถรักษาข้อมูลเวลาไว้ได้แม้ไม่มีไฟเลี้ยงไอซี

3.3.1 รายละเอียดขาต่อใช้งานของ DS1307

ในรูปที่ 3.9 แสดงการจัดขาของ DS1307 แต่ละขามีหน้าที่และการใช้งานดังนี้



รูปที่ 3.9 การจัดขาของไอซี DS1307

- Vcc, GND (ขา 8, 4) ต่อกับไฟเลี้ยง +5V

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

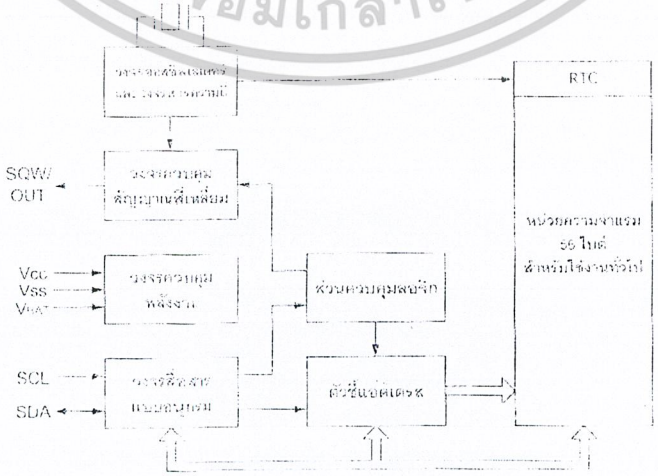
- SDA, SCL (ขา 5 และ 6) เป็นขาสำหรับเชื่อมต่อกับไมโครคอนโทรลเลอร์บนระบบบัส I^2C

- SQW OUT (ขา 7) ที่ขานี้จะมีสัญญาณรูปสี่เหลี่ยมส่งออกมา โดยสามารถเลือกความถี่ได้ 1 Hz , 4.096 kHz , 8.192 kHz และ 32 kHz ในการใช้งานต้องต่อตัวต้านทาน 1k พูลอัปที่ขานี้ด้วย

- X1, X2 (ขา 1 และ 2) ใช้ต่อกับคริสตอลความถี่มาตรฐาน 32.768 kHz เพื่อใช้เป็นฐานเวลาในการสร้างค่าเวลาจริง ในการใช้งานต้องต่อคริสตอลเข้ากับขาทั้งสองนี้ และที่ขาต้องต่อตัวเก็บประจุค่าต่ำ ๆ ประมาณ 15 pF ครอบกับขากราวด์ด้วย

3.3.2 การทำงานของ DS1307

ไอซี DS1307 จัดการเชื่อมต่อในแบบบัส I^2C โดยจะทำงานเป็นอุปกรณ์สเลฟเสมอ ดังนั้นการติดต่อเพื่อใช้งานจึงต้องกำหนดรูปแบบตามที่กำหนดไว้ในการติดต่อแบบ I^2C ในรูปที่ 3.10 แสดงส่วนประกอบหลักที่สำคัญและไดอะแกรมการทำงานของ DS1307 วงจรออสซิลเลเตอร์ถือเป็นหัวใจหลักของไอซี เนื่องจากเป็นจุดเริ่มต้นของการสร้างข้อมูลเวลาจริง ในขณะที่ DS1307 ทำงานที่ขา SQW/OUT จะมีสัญญาณพัลส์สี่เหลี่ยมส่งออกมาตลอดเวลาในกรณีที่มีการอีนาเบิลวงจรกำเนิดสัญญาณพัลส์ที่รีจิสเตอร์ควบคุม ค่าความถี่ของสัญญาณนี้สามารถเลือกได้ 4 ค่า คือ 1Hz, 4.096kHz, 8.192kHz และ 32kHz พร้อมกันนั้นก็จะมีการเก็บค่าของเวลาไว้ในหน่วยความจำนอนโวลาไทล์แรม ซึ่งมีขนาดรวม 64 ไบต์ แต่จัดสรรให้ใช้เก็บข้อมูลเวลา 8 ไบต์ และเป็นหน่วยความจำสำหรับเก็บข้อมูลทั่วไปสำหรับผู้ใช้งานอีก 56 ไบต์



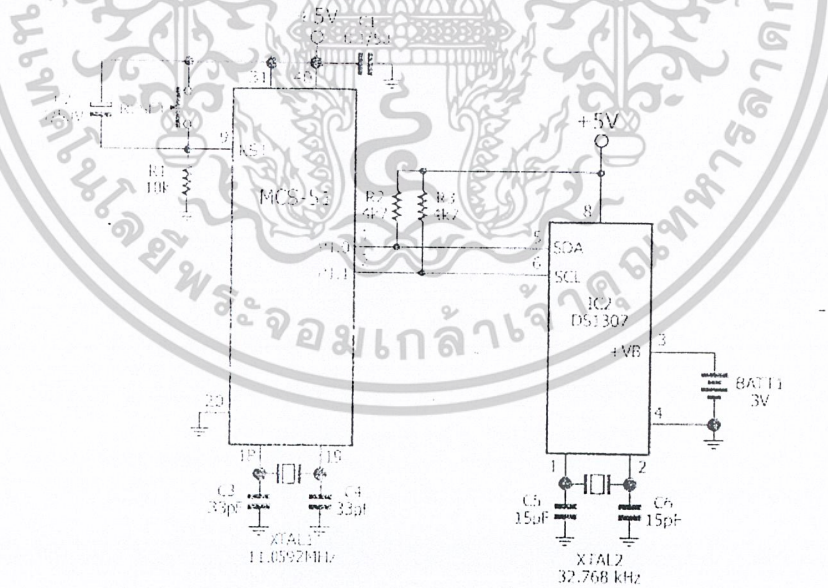
รูปที่ 3.10 แสดงโครงสร้างภายในของไอซีรีลไทม์คล็อกเบอร์ DS1307

วงจรควบคุมพลังงานไฟฟ้าจะคอยตรวจสอบสภาวะของไฟเลี้ยงไอซี หากไฟเลี้ยงต่ำกว่า $1.25xV_{BAT}$ ก็จะควบคุมให้ DS1307 หยุดการทำงาน รีเซตค่าตัวนับแอดเดรสภายในทำให้ไม่สามารถติดต่อกับ 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

3.3.3 การเชื่อมต่อ DS1307 กับไมโครคอนโทรลเลอร์ MCS-51

วงจรแสดงในรูปที่ 3.11 จะเห็นได้ว่ามีลักษณะการต่อเหมือนกับอุปกรณ์บนระบบบัส I^2C ตัวอื่น ๆ ทุกประการ และสามารถที่จะต่อไอซีทั้งหมดร่วมกันบนสาย SDA และ SCL ได้ เป็นการย้ำให้เห็นถึงความสามารถพิเศษของระบบบัส I^2C ที่ผู้ใช้งานสามารถเชื่อมต่ออุปกรณ์ที่มีความต่างกันในหน้าที่การทำงานบนสายสัญญาณเดียวกันได้ ถึงการทดลองนี้ผู้ใช้งานสามารถเชื่อมต่ออุปกรณ์ระบบบัส I^2C ได้ถึง 3 ตัว 3 ลักษณะการทำงาน โดยใช้สัญญาณเพียง 2 เส้น



รูปที่ 3.11 แสดงการเชื่อมต่อไมโครคอนโทรลเลอร์ MCS-51 กับไอซีรีลไทม์คล็อก DS1307

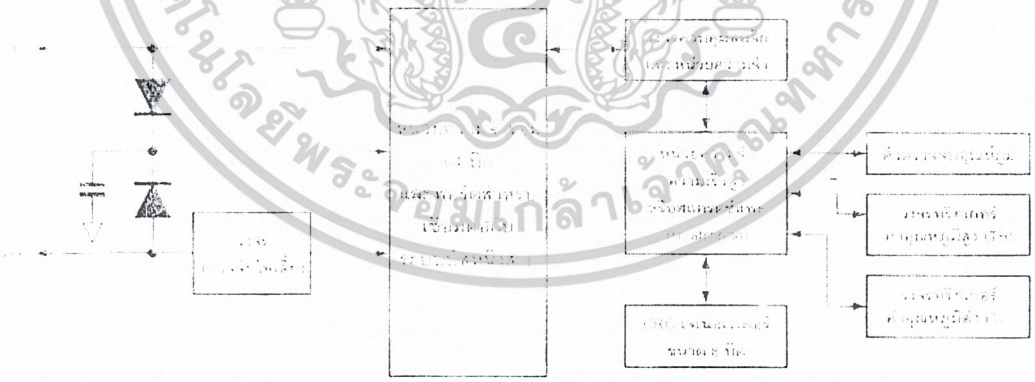
จากรูปที่ 3.11 ไอซี DS1307 จำเป็นจะต้องต่อแบตเตอรี่ไว้ตลอดเวลาไม่ว่าจะใช้งานหรือไม่ ทั้งนี้เพื่อรักษาการทำงานของวงจรภายใน DS1307 ให้ยังคงทำงานต่อเนื่องไป เมื่อใดที่ไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูล ก็จะได้ข้อมูลเวลาที่เป็นจริงตลอดเวลา

3.4 ชุดวงจรตรวจจับอุณหภูมิ

3.4.1 หลักการทำงานและรายละเอียดของไอซีเบอร์ DS1820

ไอซีเบอร์ DS1820 ใช้ในการติดต่อแบบระบบบัสหนึ่งสาย มีขาต่อใช้งานเพียง 3 ขา คือ DQ ซึ่งเป็นขาเชื่อมต่อกับระบบบัส , ขาต่อไฟเลี้ยงภายนอก และขากราวด์ ดังแสดงในรูปที่ 3.12 และมีโครงสร้างการทำงานภายในแสดงดังรูปที่ 3.13

รูปที่ 3.12 แสดงการจัดขาของ DS1820



รูปที่ 3.13 แสดงโครงสร้างการทำงานภายในของไอซีตรวจจับอุณหภูมิ DS1820

หัวใจสำคัญของ DS1820 อยู่ที่ตัวตรวจจับอุณหภูมิและหน่วยความจำความเร็วสูงที่เรียกว่า สแครตซ์แพด (SCRATCHPAD) ซึ่งมีขนาด 9 ไบต์ มีการจัดสรรหน่วยความจำส่วนนี้แสดงดังรูปที่ 3.14

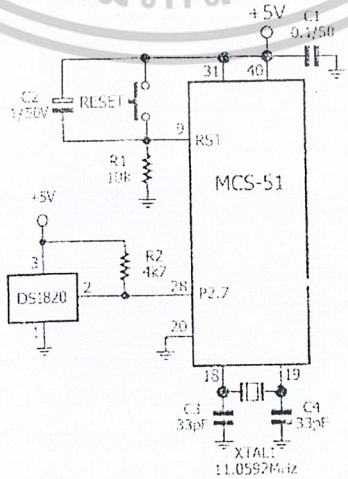
	ไบต์
ข้อมูลอุณหภูมิไบต์ต่ำ (TL)	0
ข้อมูลอุณหภูมิไบต์สูง	1
ข้อมูลอุณหภูมิค่าสูง	2
ข้อมูลอุณหภูมิค่าต่ำ (TL)	3
สำรองไว้	4
สำรองไว้	5
รีจิสเตอร์เก็บค่าการนับ	6
รีจิสเตอร์เก็บค่าการนับต่อ °C	7
CRC	8

รูปที่ 3.14 แสดงการจัดสรรพื้นที่ของสแครตซ์แพดใน DS1820

เมื่อวัดอุณหภูมิได้ก็จะนำค่าที่วัดได้นี้มาเก็บไว้ในสแครตซ์แพดที่ไบต์ 0 และ 1 ทั้งนี้เนื่องจากไอซี DS1820 สามารถให้ข้อมูลของอุณหภูมิได้ละเอียดถึง 16 บิต เมื่อนำมาแปลงเป็นข้อมูลเลขฐานสิบจึงสามารถแสดงความละเอียดของค่าอุณหภูมิได้ถึง 0.5 องศาเซลเซียส และ 0.9 องศาฟาเรนไฮต์ โดยมีย่านวัดอุณหภูมิ -55 ถึง +125 องศาเซลเซียส หรือ -67 ถึง +257 องศาฟาเรนไฮต์ โดยค่าขององศาฟาเรนไฮต์ต้องใช้การแปลงหน่วยเข้ามาช่วยให้เวลาในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลประมาณ 200 มิลลิวินาที สามารถกำหนดขอบเขตของอุณหภูมิที่ทำการวัดได้ และให้แจ้งเตือนเมื่อค่าของอุณหภูมิสูงขึ้นหรือลดต่ำลงถึงค่าที่กำหนด โดยค่าอุณหภูมิที่กำหนดนี้จะเก็บไว้ที่สแครตซ์แพดในไบต์ 2 และ 3

3.4.2 การเชื่อมต่อกับไมโครคอนโทรลเลอร์ MCS-51

แสดงวงจรการเชื่อมต่อในรูปที่ 3.15 ใช้ขาพอร์ตเพียง 1 ขา เท่านั้น สำหรับการเชื่อมต่อกับ DS1820 โดยต้องมีตัวต้านทานค่า 4.7kΩ ต่อพูลอัปกับไฟเลี้ยง +5V จากนั้นจึงทำการเขียนโปรแกรมเพื่อติดต่อกัน โดยใช้รูปแบบการติดต่อตามมาตรฐานระบบบัสหนึ่งสายของดัลลัส

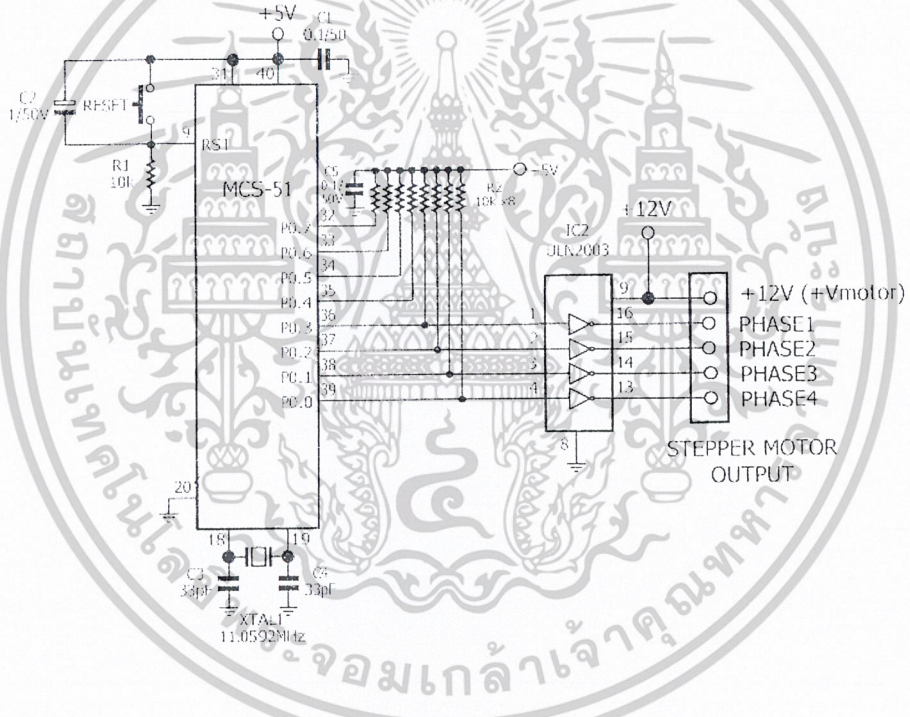


รูปที่ 3.15 แสดงการเชื่อมต่อ DS1820 กับไมโครคอนโทรลเลอร์ MCS-51

3.5 วงจรขับสเต็ปเปอร์มอเตอร์ของไมโครคอนโทรลเลอร์ MCS-51

แสดงในรูปที่ 3.16 จะใช้พอร์ต 0 ส่งข้อมูลไปยังไอซีไดรเวอร์กระแสสูงแบบคอลเล็กเตอร์เปิด เบอร์ ULN2003 ด้วยการใช้ไอซีแบบนี้ทำให้สามารถเลือกแรงดันสำหรับขยับสเต็ปเปอร์มอเตอร์ได้กว้างขวางตั้งแต่ 5-30V โดย ULN2003 มีความสามารถในการจ่ายกระแสได้สูงสุด 500mA ต่อขา ทั้งนี้ต้องเตรียมแหล่งจ่ายไฟให้มีความสามารถในการจ่ายกำลังไฟฟ้าสูงเพียงพอด้วย

เมื่อต้องการให้มอเตอร์หมุนให้ส่งข้อมูล "1" ไล่เรียงไปตามลำดับจาก P0.0 ถึง P0.3 แล้ววนกลับมาที่ P0.0 ใหม่ หากต้องการให้มอเตอร์หมุนกลับทิศทางก็ให้ส่งข้อมูลย้อนกลับ โดยเริ่มจาก P0.3 ก่อนแล้วสิ้นสุดรอบที่ P0.0 แล้ววนกลับไปที่ P0.3 ใหม่ เมื่อ ULN2003 ได้รับข้อมูล "1" ก็จะทำการกลับลอจิก ทำให้เกิดกระแสไฟฟ้าไหลผ่านขดลวดของมอเตอร์ที่ต่ออยู่กับขาเอาต์พุตที่ทำงาน ส่งผลให้เกิดการเคลื่อนที่ของแกนมอเตอร์ขึ้น

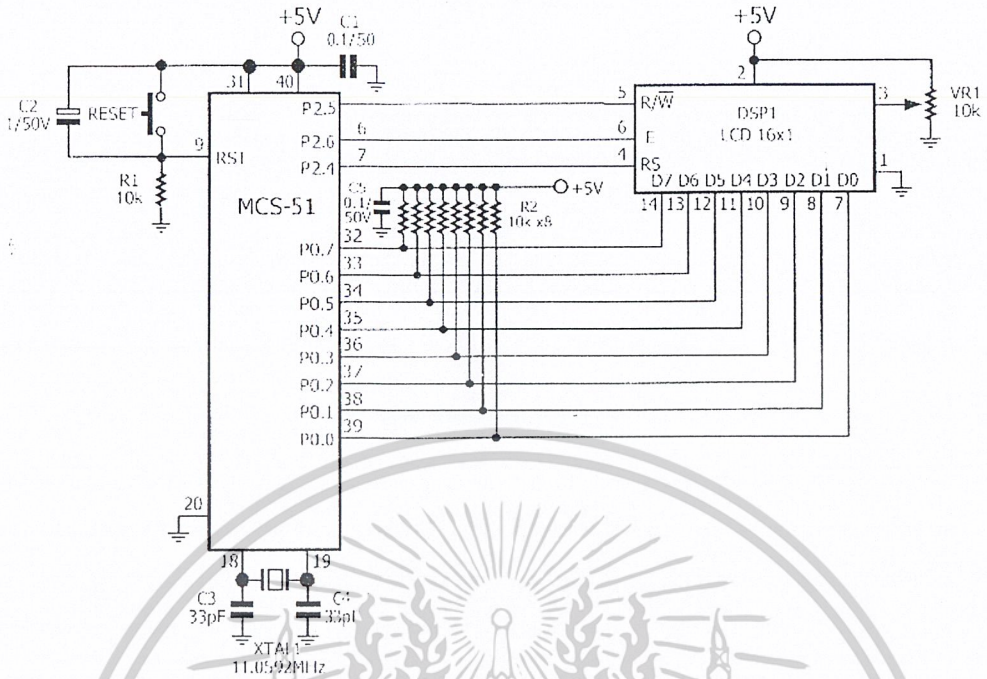


รูปที่ 3.16 แสดงวงจรการขับสเตปเปอร์มอเตอร์ของไมโครคอนโทรลเลอร์ MCS-51

3.6 การเชื่อมต่อ LCD Module เข้ากับไมโครคอนโทรลเลอร์ MCS-51

การเชื่อมต่อ LCD Module เข้ากับไมโครคอนโทรลเลอร์สามารถต่อได้โดยตรงกับตัว MCS-51 ดังรูป 3.17

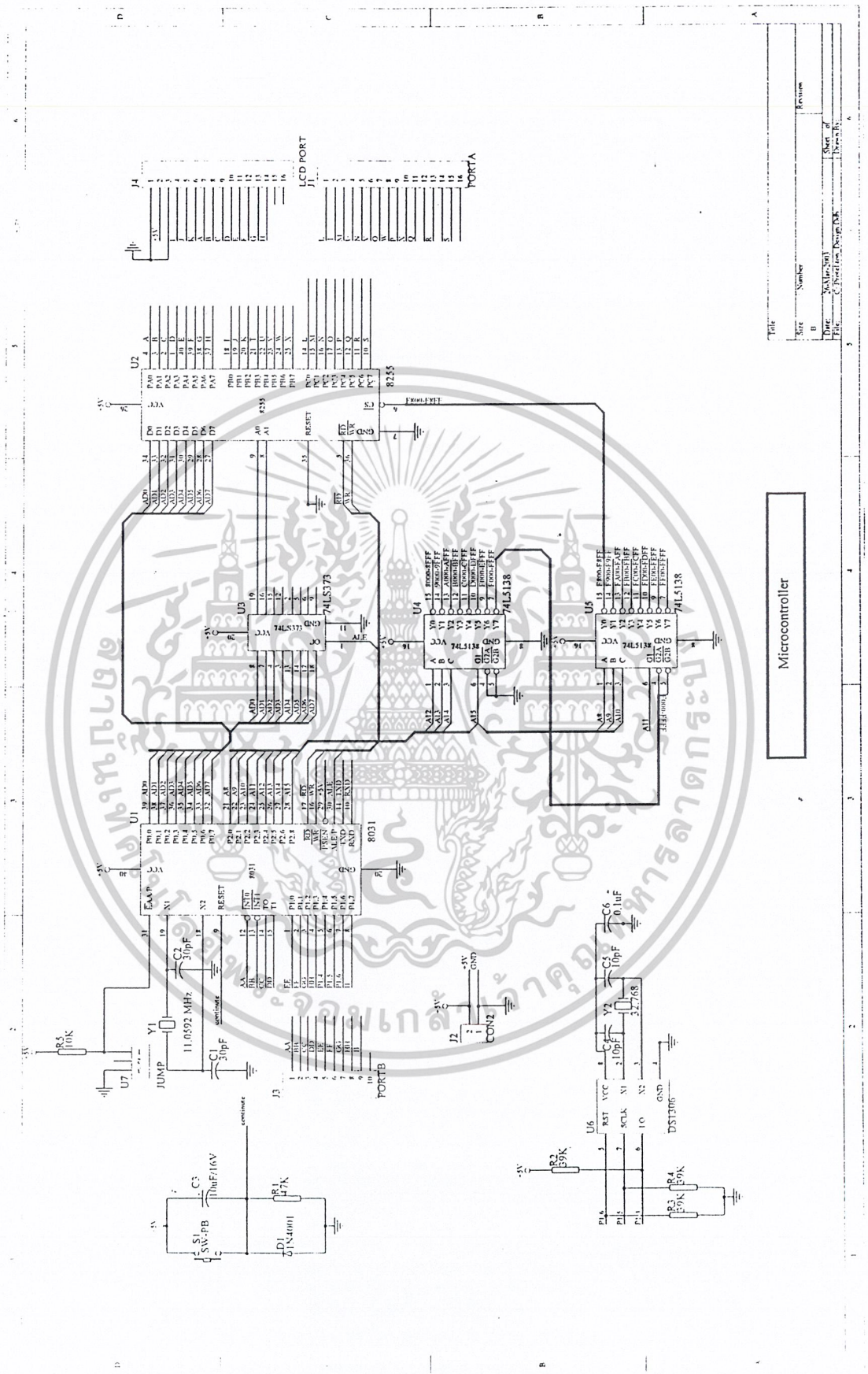
- ขาสัญญาณข้อมูล D0-D7 (ขา 32-39) ต่อเข้ากับ MCS-51 พอร์ต 0
- ขา RS (ขา 25) ต่อเข้ากับ MCS-51 พอร์ต 2 บิต 4
- ขา R/W (ขา 26) ต่อเข้ากับ MCS-51 พอร์ต 2 บิต 5
- ขา E (ขา 27) ต่อเข้ากับ MCS-51 พอร์ต 2 บิต 6



รูปที่ 3.17 แสดงการเชื่อมต่อ LCD Module เข้ากับ MCS-51

3.7 ชุด Controller

ชุด Controller ใช้ชิพตระกูล MCS-51 เบอร์ P89C51RD2 ซึ่งเป็นไอซีที่สามารถโปรแกรมลงไปในตัวเองได้ มีคุณสมบัติเหมือนไอซี DS5000 ทำให้สามารถทำงานได้ โดยสามารถต่อใช้งานกับไอซี 8255 เป็น LCD Port และใช้ Port ที่เหลือเป็น อินพุต/เอาต์พุต พอร์ต ในการควบคุมเครื่องปรับอากาศ และรับอินพุตจากรีโมทและชุดตรวจจับอุณหภูมิ วงจรแสดงดังรูปที่ 3.18



รูปที่ 3.18 แสดงชุด Controller

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลองรับข้อมูลจากเครื่องควบคุมระยะไกล

4.1.1 อุปกรณ์ที่ใช้ในการทดลอง

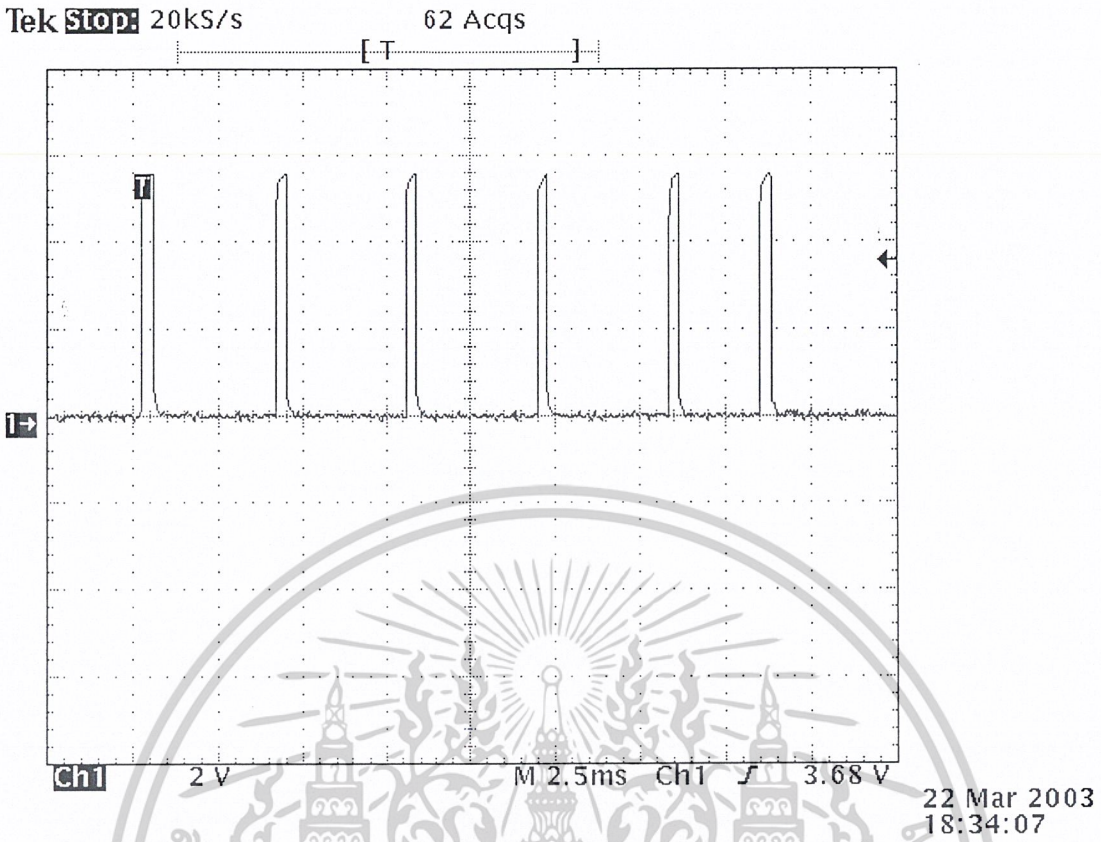
1. Oscilloscope
2. เครื่องควบคุมระยะไกล
3. Control บอร์ด

4.1.2 ขั้นตอนการทดลอง

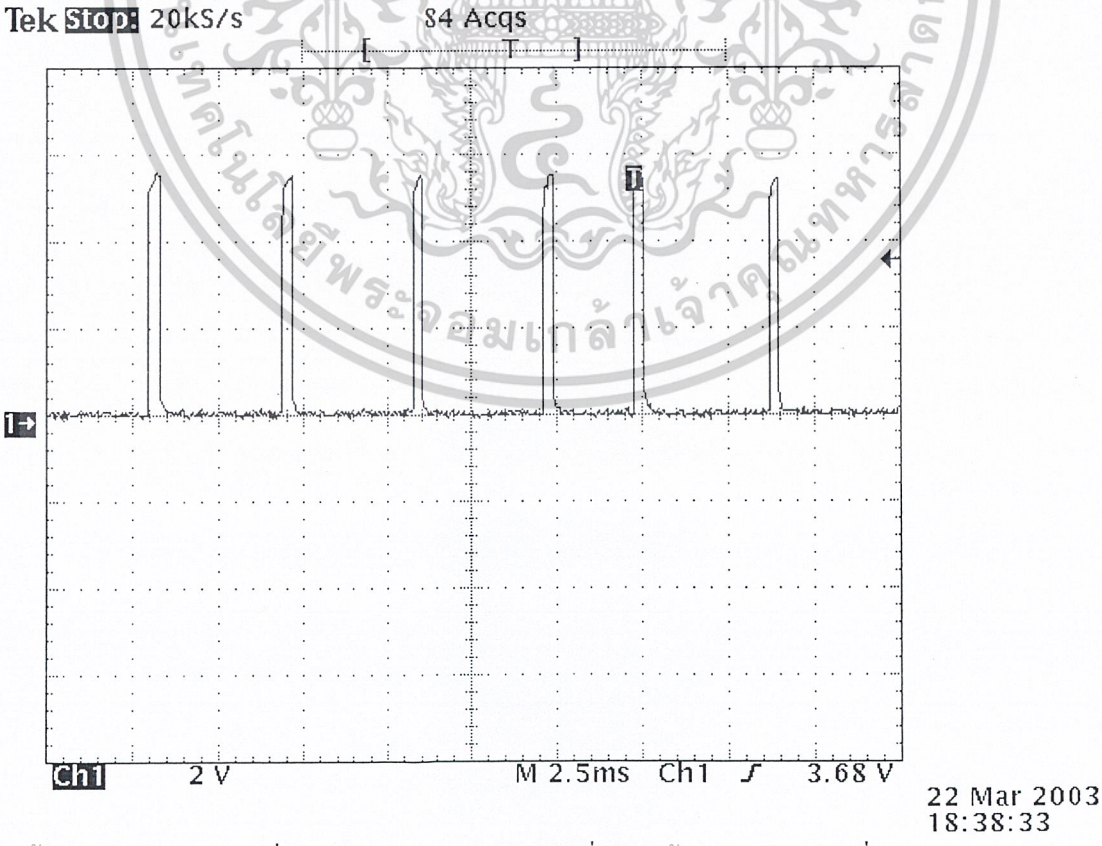
1. ใช้ Oscilloscope วัดรูปสัญญาณในเครื่องควบคุมระยะไกล ที่ขา 3 ซึ่งเป็นขาที่รับสัญญาณจากตัวรับอินฟราเรด
2. กดปุ่มส่งสัญญาณจากเครื่องควบคุมระยะไกลไปยังตัวรับสัญญาณอินฟราเรด
3. พิมพ์รูปสัญญาณที่อ่านได้จาก Oscilloscope
4. อ่านค่า Bit ที่ได้จากเครื่องรับโดยดูที่ LED บนบอร์ดภาครับ
5. ทำการทดลองจาก ข้อ 1-4 ให้ครบทุกปุ่มของเครื่องควบคุมระยะไกล

4.1.3 ผลการทดลอง

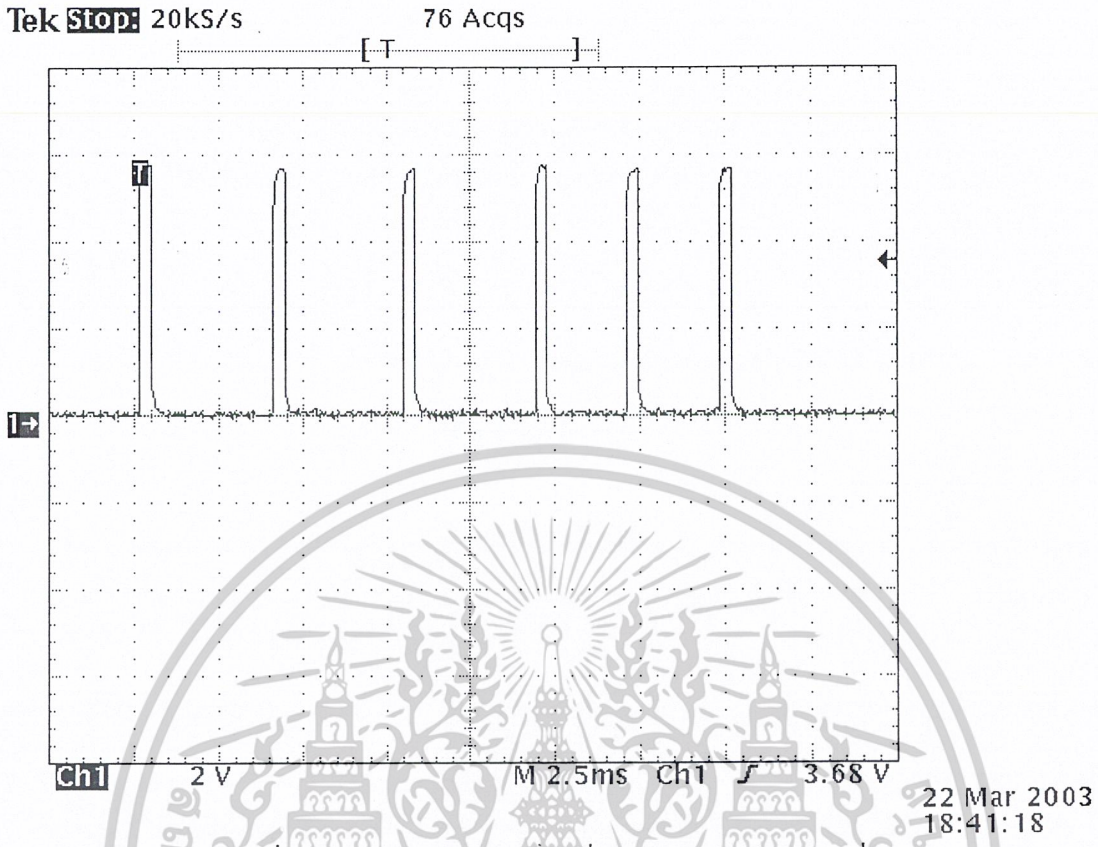
ผลการทดลองที่ได้แสดงดังรูปที่ 4.1-4.15



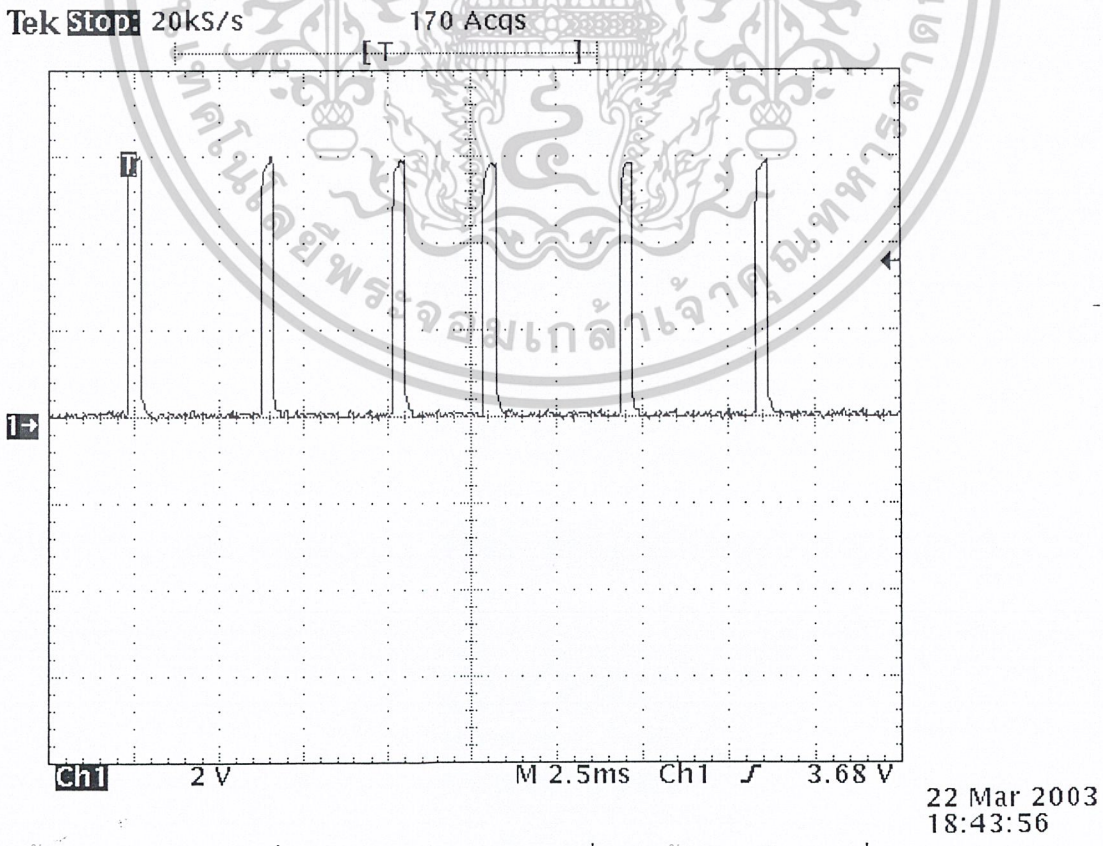
รูปที่ 4.1 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 1



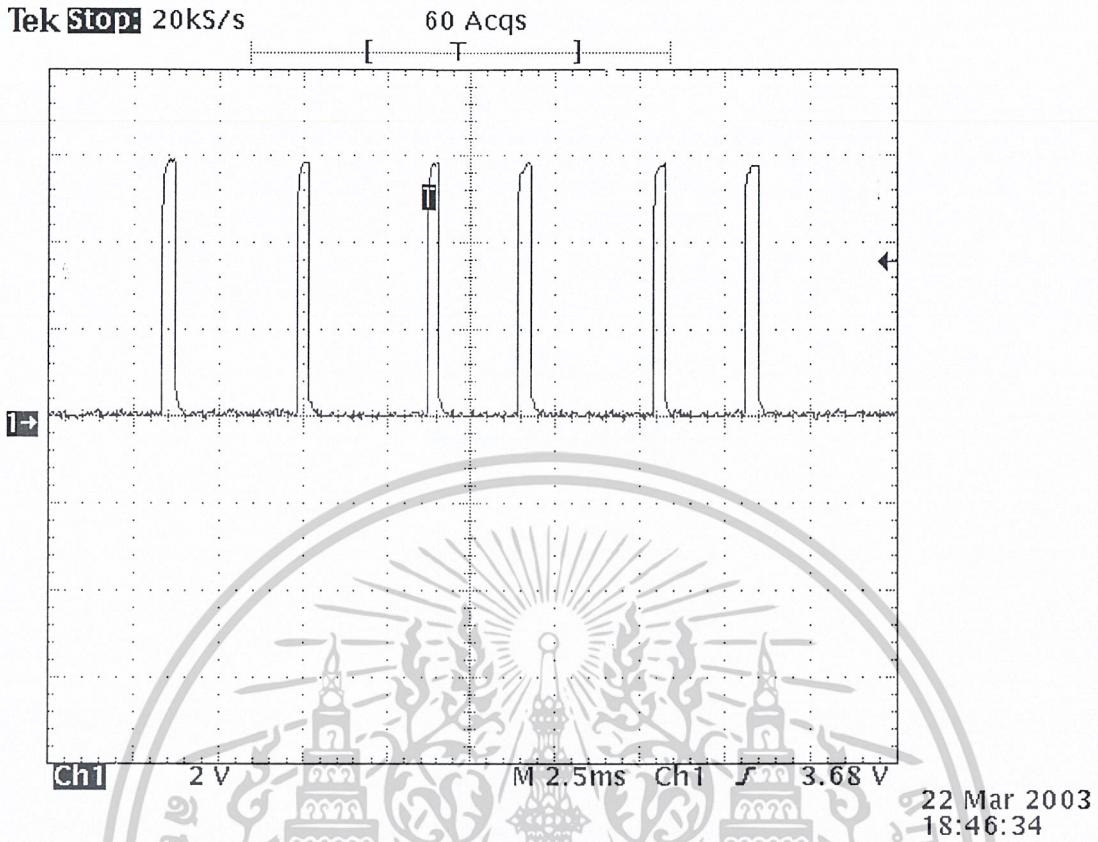
รูปที่ 4.2 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 2



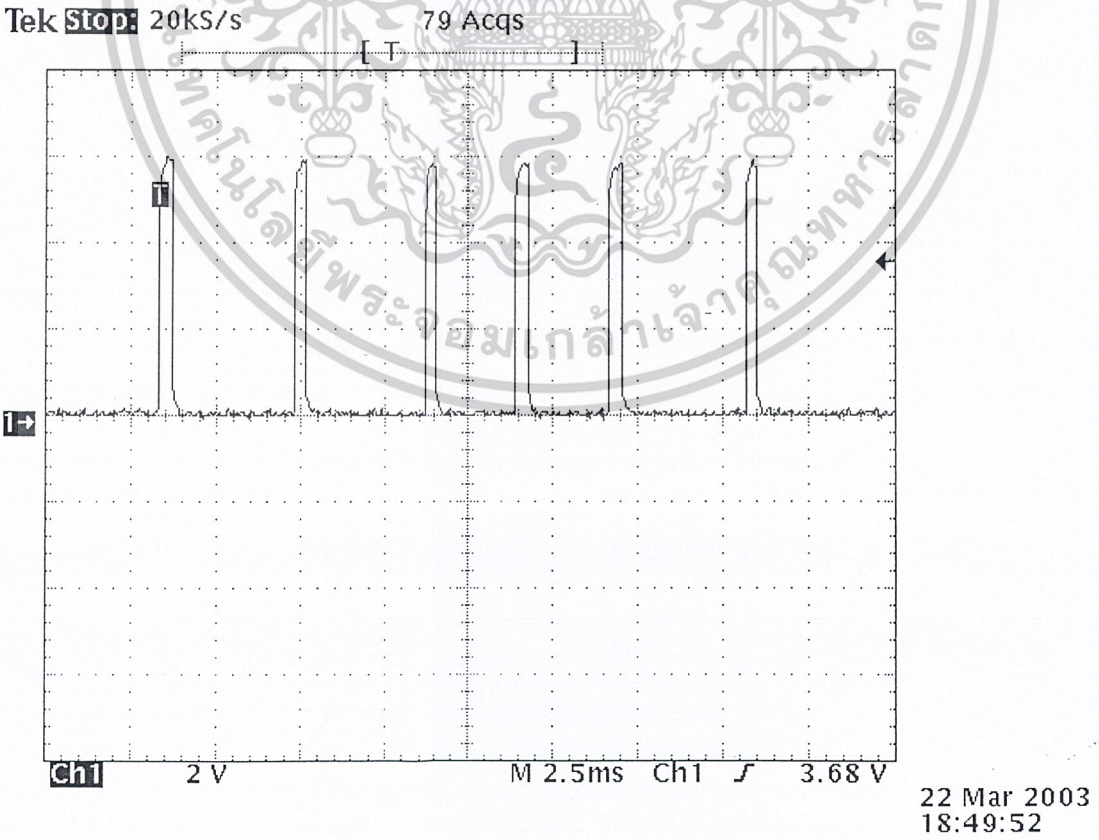
รูปที่ 4.3 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 3



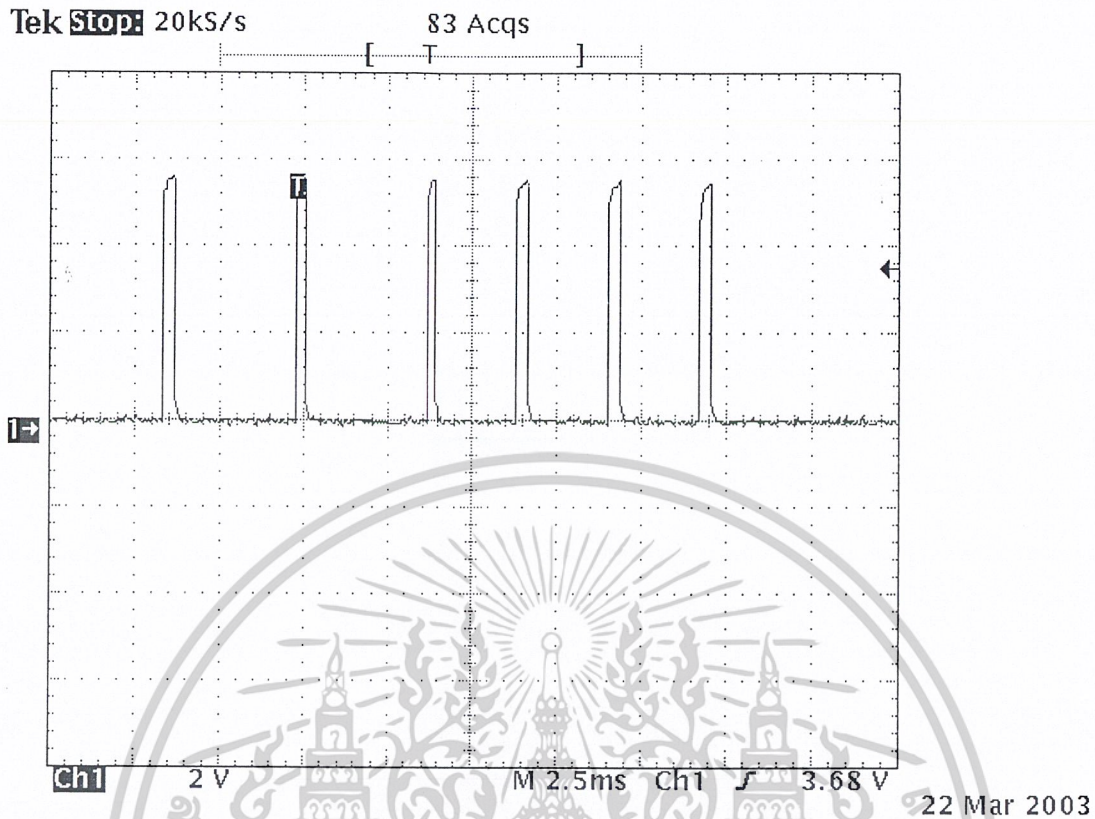
รูปที่ 4.4 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 4



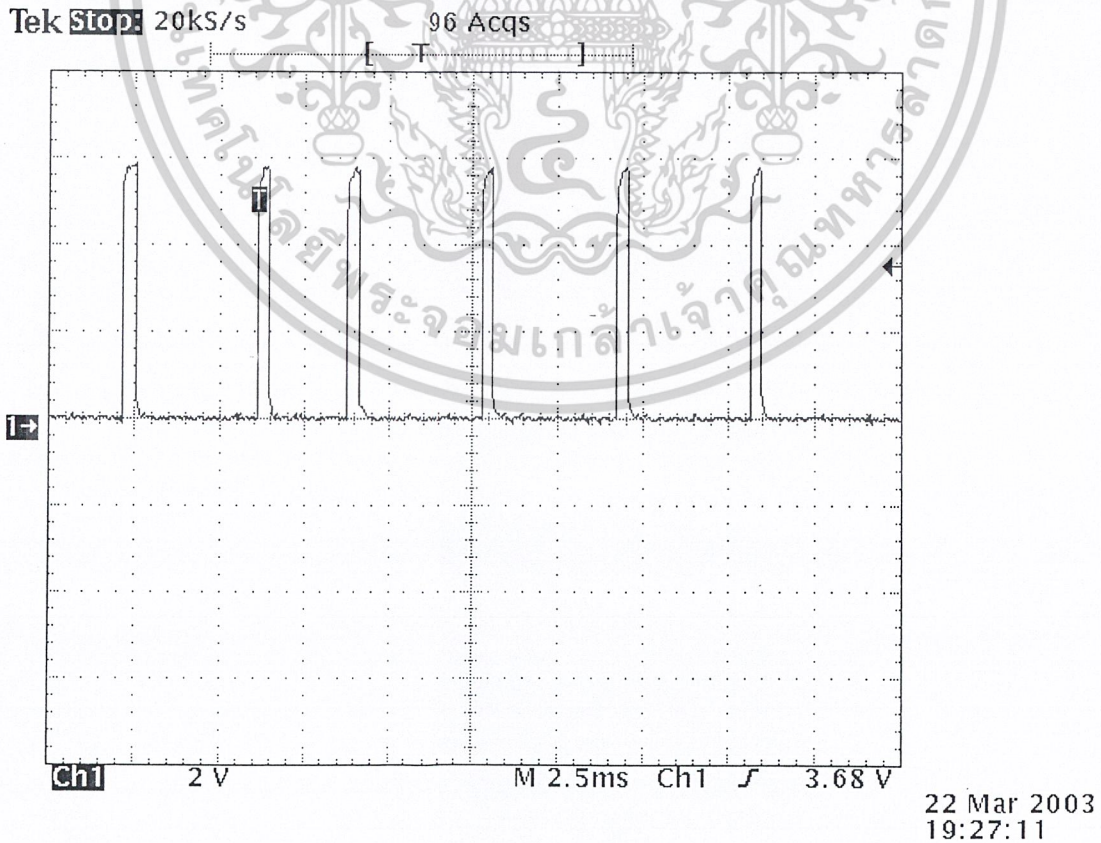
ที่ 4.5 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 5



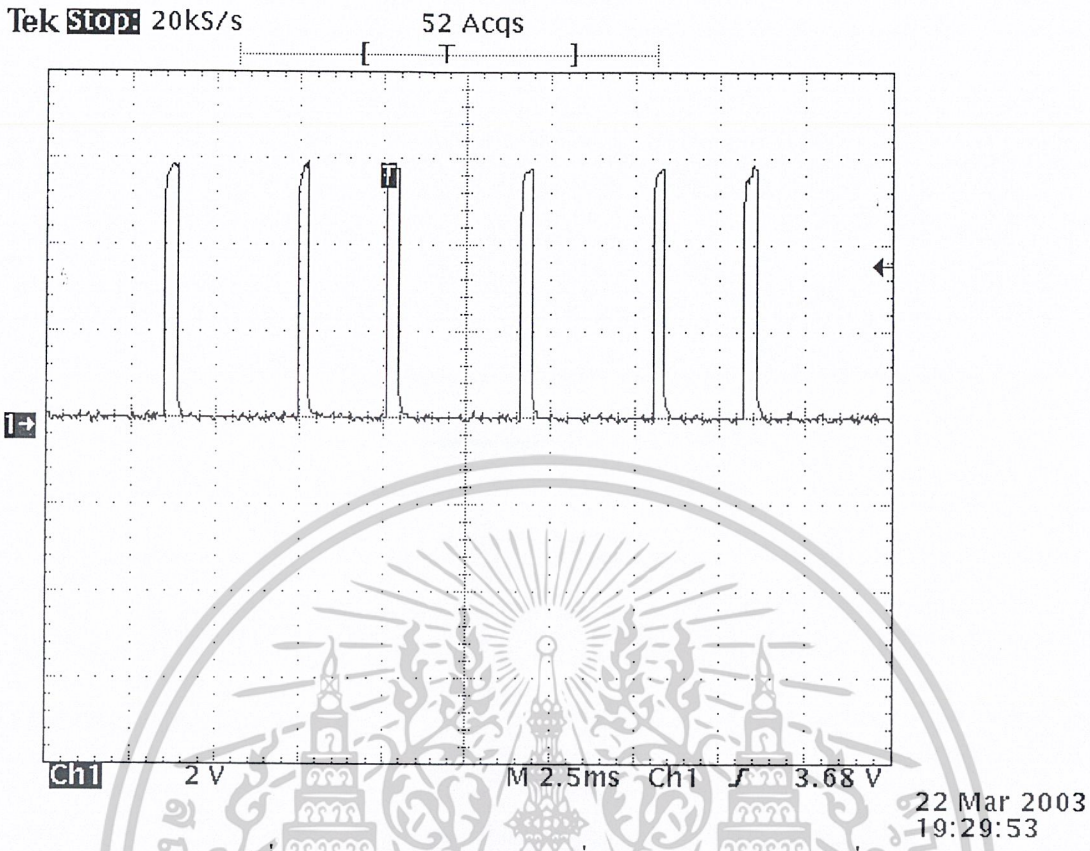
ที่ 4.6 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 6



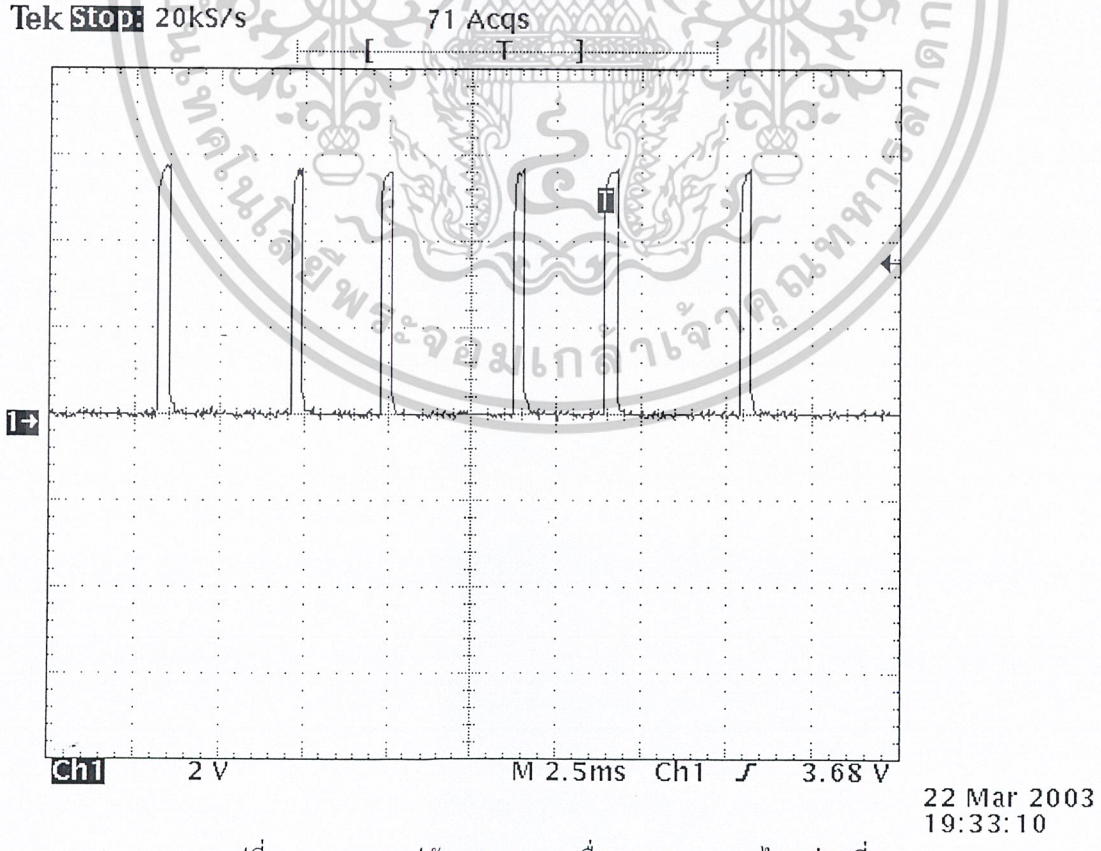
ที่ 4.7 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 7



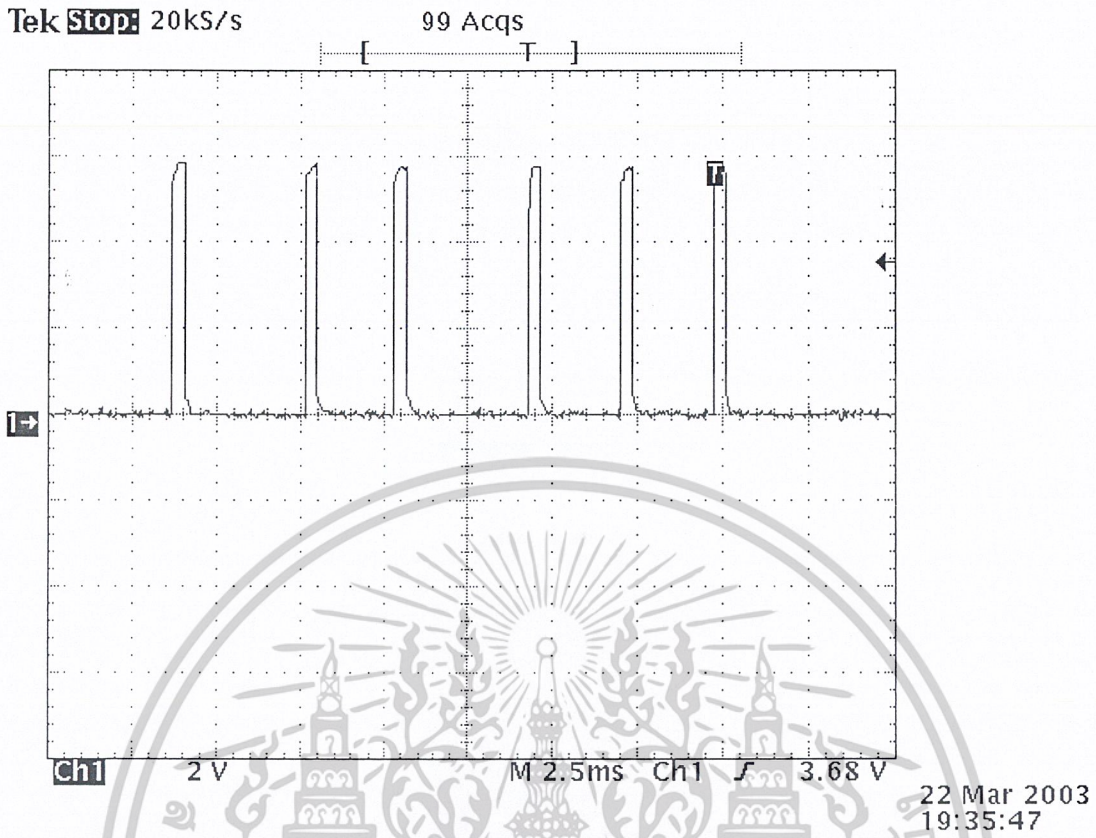
รูปที่ 4.8 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 8



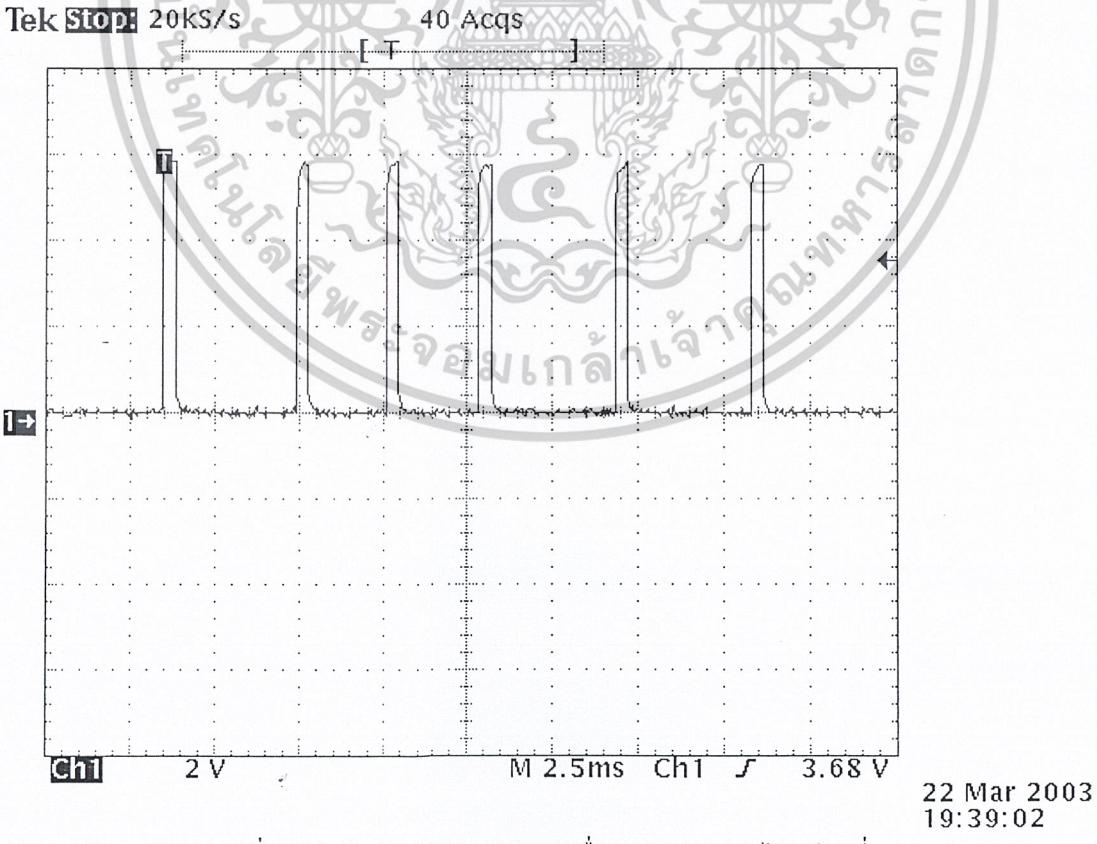
รูปที่ 4.9 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 9



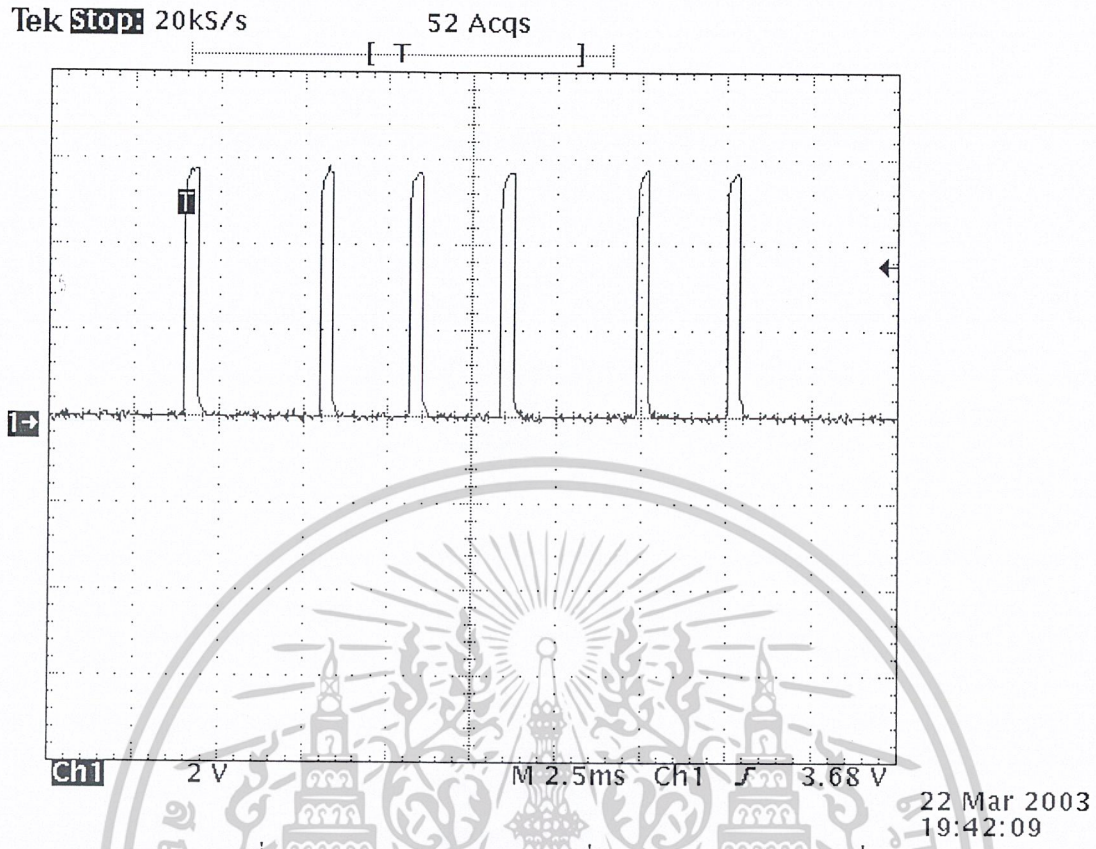
รูปที่ 4.10 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 10



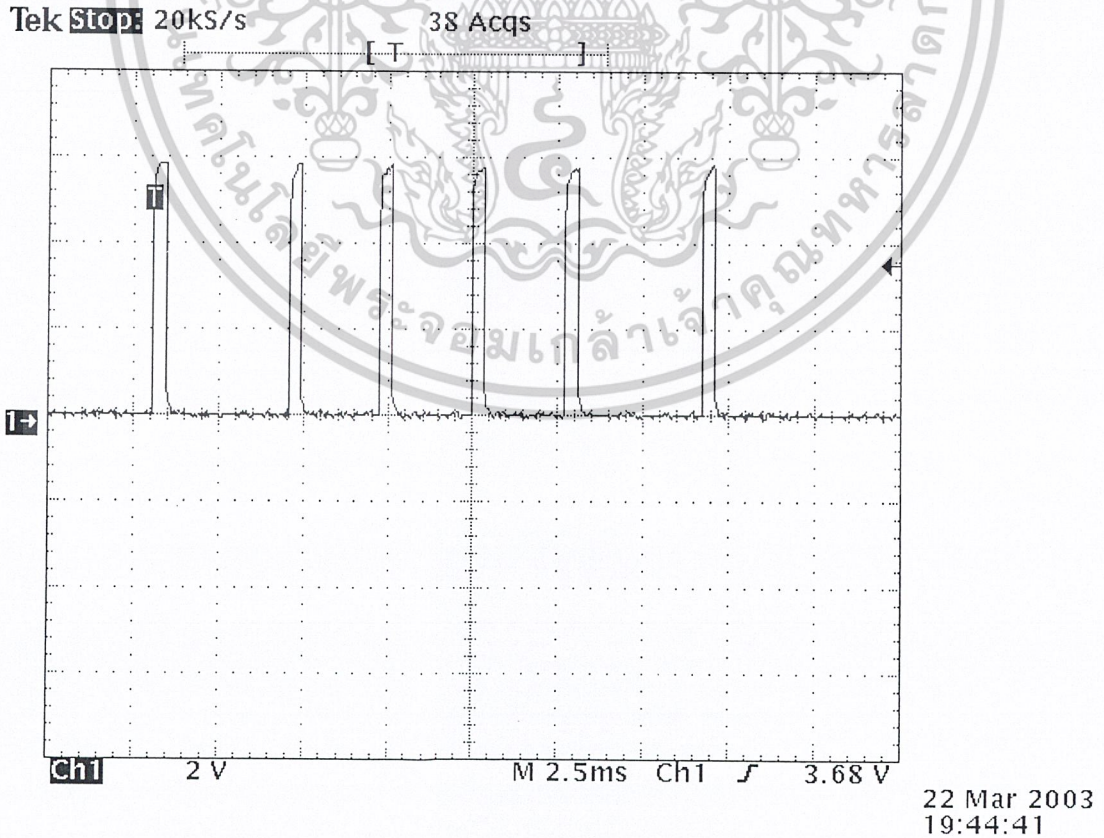
รูปที่ 4.11 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 11



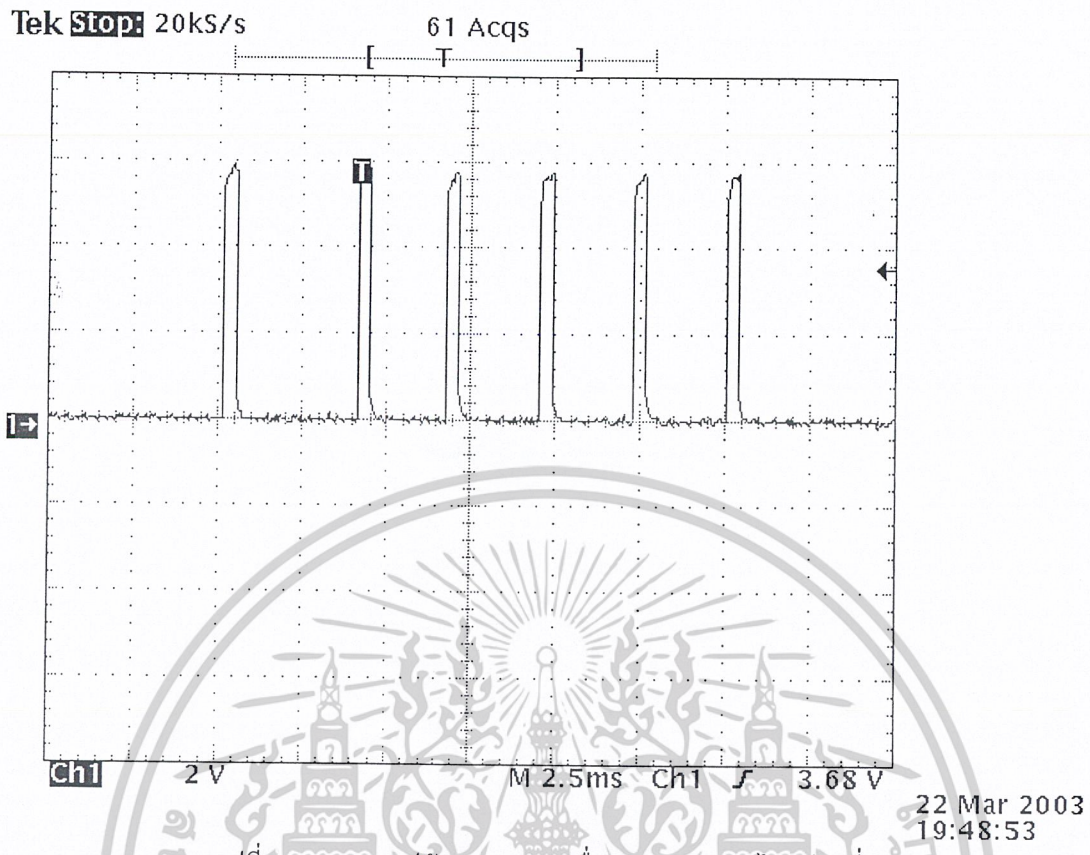
รูปที่ 4.12 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 12



รูปที่ 4.13 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 13



รูปที่ 4.14 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 14



รูปที่ 4.15 แสดงรูปสัญญาณของเครื่องควบคุมระยะไกลช่องที่ 15

4.2 การทดลองการวัดค่า OUTPUT ของตัวตรวจจับอุณหภูมิ DS1820

4.2.1 อุปกรณ์ที่ใช้ในการทดลอง

1. Oscilloscope
2. เครื่องควบคุมระยะไกล
3. Control บอร์ด
4. ตัวตรวจจับอุณหภูมิ DS1820

4.2.2 ขั้นตอนการทดลอง

1. กดปุ่มส่งสัญญาณจากเครื่องควบคุมระยะไกลไปยังตัวรับสัญญาณอินฟราเรด เพื่อตั้งอุณหภูมิตามที่ต้องการ

2. ใช้ Oscilloscope วัดรูปสัญญาณที่ขา 2 ของตัวตรวจจับอุณหภูมิ ซึ่งเป็นขาที่ตรวจจับอุณหภูมิภายในห้องขณะนั้น

3. ทำการทดลองจาก ข้อ 1 และ 2 โดยเริ่มจากอุณหภูมิเริ่มต้นที่ 18 °C จนถึง 31 °C

4.2.3 ผลการทดลอง

อุณหภูมิ (°C)	OUTPUT DIGITAL (HEX)
18	12
19	13
20	14
21	15
22	16
23	17
24	18
25	19
26	1A
27	1B
28	1C
29	1D
30	1E
31	1F

ตารางที่ 4.1 แสดงการวัดค่า OUTPUT ของตัวตรวจจับอุณหภูมิ DS1820

บทที่ 5
บทวิจารณ์และบทสรุป

จากการทดลองในส่วนต่าง ๆ ของปริญญานิพนธ์เรื่องนี้ สามารถควบคุมเครื่องปรับอากาศได้ตามขอบเขตของโครงงานที่ตั้งไว้

ในส่วนของรีโมทคอนโทรล ซึ่งเราได้ทำไป 2 ชุด โดยชุดแรกที่ทำไปนั้น ส่งได้ระยะใกล้ประมาณ 3 เมตร ซึ่งเกิดจากอุปกรณ์ และลายวงจรมีเส้นทางการเดินทางที่ซับซ้อนทำให้เกิดการรบกวนกันทางความถี่ ทางผู้จัดทำได้ทำรีโมทคอนโทรลขึ้นอีก 1 ชุด ซึ่งสามารถส่งได้ตามวัตถุประสงค์ที่ตั้งไว้

ในส่วนของ บอร์ด คอนโทรลเลอร์ ทางผู้จัดทำ ได้ทำการทดลองและ สร้าง แต่ประสบปัญหา คือ ไมโครคอนโทรลเลอร์ ไม่สามารถถอดรหัสตำแหน่งของ 8255 ได้ และทางคณะผู้จัด ได้ทดลองและจัดทำ ชุดคอนโทรลเลอร์ขึ้นมาใหม่ ซึ่งสามารถใช้งานได้ตามวัตถุประสงค์ที่ตั้งไว้เช่นกัน

จากการทดลองที่ผ่านมาจะเห็นว่าได้เกิดข้อผิดพลาดในหลาย ๆ ส่วนของการทำงาน ทั้งนี้เนื่องมาจากข้อผิดพลาดบางประการที่เกิดจากการลงมือปฏิบัติจริงกับทฤษฎีที่เคยศึกษามา แต่ไม่เคยนำมาปฏิบัติ ทำให้ขาดความชำนาญ และขาดความเข้าใจอย่างแท้จริง

ภาคผนวก

```
#pragma code
#include <reg52.h>
#include <absacc.h>
#include <ctype.h>
#include <intrins.h>
#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

sbit  LCD_RS = P2^4;
sbit  LCD_RW = P2^5;
sbit  LCD_E = P2^6;

sbit  KEY0 = P1^0;
sbit  KEY1 = P1^1;
sbit  KEY2 = P1^2;
sbit  KEY3 = P1^3;
sbit  KEY4 = P1^4;
sbit  KEY5 = P1^5;
sbit  KEY6 = P1^6;
sbit  KEY7 = P3^0;
sbit  KEY8 = P3^1;
sbit  KEY9 = P3^2;

sbit   IPSCL  = P2^2;                  // I2C I/O Bit
sbit   IPSDA  = P2^3;
sbit  TMDAT = P2^7;
sbit  P8255A0 = P2^0;
sbit  P8255A1 = P2^1;
sbit  P8255WR = P3^6;
sbit  P8255RD = P3^7;

unsigned char          TIMBUF[8];          // ss,mm,hh,ww,dd,mm,yy,cc
unsigned char          DISBUF[16];
unsigned char          HEXBUF[3];          // hex buffer
unsigned char          TIMEOFFBUF[3];
unsigned char          TEMBUF[2];
unsigned char          temperature,fan_level,step_value;
int step_delay,step_count_L,step_count_R;
bit    timeoff_flag;
bit turn_off;
bit step_flag;
bit step_first_time;


/********** Function **********/

void dmsec(unsigned int count) {  /* Delay mSec */
    unsigned char i;
    while (count) {
        for (i=1;i<=113;i++);
        count--;
    }
}

void wporta(unsigned char xxx){
      P0=xxx;
      P8255A0=0;
      P8255A1=0;
```

```
P8255WR=0;
      P8255WR=1;
}
void stepping_L(){
      wporta(step_value);
      dmsec(350);
      step_value =  step_value << 1;
      if (step_value==0x10){ step_value=0x01;}

}
void stepping_R(){
      wporta(step_value);
      dmsec(350);
      step_value =  step_value >> 1;
      if (step_value==0x0){ step_value=0x08;}

}

void LCD_WI(unsigned char ins){
      LCD_E = 0;
      LCD_RW = 0;
      LCD_RS = 0;
      P0 = ins;
      dmsec(1);
      LCD_E = 1;
      dmsec(1);
      LCD_E = 0;
}

void LCD_WD(unsigned char dd){
      LCD_E = 0;
      LCD_RW = 0;
      LCD_RS = 1;
      P0 = dd;
      dmsec(1);
      LCD_E = 1;
      dmsec(1);
      LCD_E = 0;
}

void INIT_LCD(){
      LCD_WI(0x38);
      LCD_WI(0x0E);
      LCD_WI(0x01);
      LCD_WI(0x06);
}

void display_LCD(unsigned char *str)
{
      unsigned char i;
      LCD_WI(0x80);
      for (i=0;i<8;i++ )
      {
            LCD_WD(str[i]);
      }
      LCD_WI(0xC0);
      for (i=8;i<16;i++ )
      {
            LCD_WD(str[i]);
      }
```

```
}

unsigned char scan_key(){
      unsigned char xx;
      P1=0x7f;

      if (KEY0==1)      { return(0x0);}
      if (KEY1==1)      { return(0x1);}
      if (KEY2==1)      { return(0x2);}
      if (KEY3==1)      { return(0x3);}
      if (KEY4==1)      { return(0x4);}
      if (KEY5==1)      { return(0x5);}
      if (KEY6==1)      { return(0x6);}
      if (KEY7==1)      { return(0x7);}
      if (KEY8==1)      { return(0x8);}
      if (KEY9==1)      { return(0x9);}

      if (step_flag==1){

            if (step_count_L) {
                  stepping_L(); step_count_L--;
                  if (step_count_L==0) {
                        step_value = step_value >> 1;
                        if (step_value==0x0){
                              step_value=0x08;
                        }
                  }
            }
            else  if (step_count_R) {
                  stepping_R(); step_count_R--;
                  if (step_count_R==0){
                        step_value = step_value << 1;
                        if (step_value==0x10){
                              step_value=0x01;
                        }
                  }
            }
            else {step_count_L=8;step_count_R=8; }
      }
      return(0xFF);
}

void ipdel (void) {                        // I2C delay
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
}

void ipchigh (void) {                      // I2C clock high
    IPSCL = 1;
    ipdel ();
}

void ipclow (void) {                       // I2C clock low
    IPSCL = 0;
    ipdel ();
```

```
}

void ipstart (void) {                    // start condition
    IPSDA = 1;
    IPSCL = 1;
    IPSDA = 0;
    ipdel ();
    IPSCL = 0;
    IPSDA = 1;
}

void ipstop (void) {                     // stop condition
    IPSDA = 0;
    IPSCL = 1;
    ipdel ();
    IPSDA = 1;
}

bit irwrbyte (unsigned dat) {            // write one byte for ds1307
    unsigned char i;                     // return 0 = ok
    bit outbit;                          // return 1 = error
    for (i=1;i<=8;i++) {
        outbit = dat & 0x80;
        IPSDA = outbit;
        dat = dat << 1;
        ipchigh ();
        ipclow ();
    }
    IPSDA = 1;
    ipchigh ();
    outbit = IPSDA;
    ipclow ();
    return (outbit);
}

unsigned char irrdbyte () {              // read last byte for ds1307 (not ack)
    unsigned char i,dat;
    bit inbit;
    dat = 0;
    for (i=1;i<=8;i++) {
        ipchigh ();
        inbit = IPSDA;
        dat = dat << 1;
        dat = dat | inbit;
        ipclow ();
    }
    IPSDA = 1;
    ipchigh ();
    ipclow ();
    return (dat);
}

unsigned char irrdbytex () {             // read one byte for DS1307
    unsigned char i,dat;
    bit inbit;
    dat = 0;
    for (i=1;i<=8;i++) {
        ipchigh ();
        inbit = IPSDA;
        dat = dat << 1;
        dat = dat | inbit;
```

```
        ipclow ();
    }
    IPSDA = 0;
    ipchigh ();
    ipclow ();
    IPSDA = 1;
    return (dat);
}
void rtset (void) {                     // set RTC status bit
    TIMBUF[0] = TIMBUF[0] & 0x7f;       // clear CH (clock halt)
    TIMBUF[2] = TIMBUF[2] & 0xbf;       // set 24 hour
    TIMBUF[7] = TIMBUF[7] & 0xec;       // b7=out b4=sqwe b1,b0=rs
}

bit rtwr (void) {                       // ds1307 rtc write
    unsigned char i;                    // return 0=ok 1=error
    bit err;
    ipstart ();
    err = irwrbyte (0xd0);              // address & r/w bit
    if (!err) {
        err = irwrbyte (0x00);          // start register addr=0
        if (!err) {
            for (i=0;i<=7;i++) {
                err = irwrbyte (TIMBUF[i]);
                if (err) break;
            }
        }
    }
    ipstop ();
    return (err);
}

bit rtrd (void) {                       // ds1307 rtc read
    unsigned char i;                    // return 0=ok 1=error
    bit err;
    ipstart ();
    err = irwrbyte (0xd0);              // address & r/w bit
    if (!err) {
        err = irwrbyte (0x00);          // start register addr=0
        if (!err) {
            ipstop ();
            ipdel ();
            ipstart ();
            err = irwrbyte (0xd1);      // address & r/w bit
            if (!err) {
                for (i=0;i<=6;i++) {
                    TIMBUF[i] = irrdbytex ();
                }
                TIMBUF[7] = irrdbyte ();    // last byte
            }
        }
    }
    ipstop ();
    return (err);
}
void htolcd (void) {                        // change hex to segment
    DISBUF[8] = ((HEXBUF[0] & 0xf0) >> 4)+0x30;     // hex[3] to disbuf[6]
    DISBUF[9] = (HEXBUF[0] & 0x0f)+0x30;
      DISBUF[10]=':';
    DISBUF[11] = ((HEXBUF[1] & 0xf0) >> 4)+0x30;
    DISBUF[12] = (HEXBUF[1] & 0x0f)+0x30;
```

```
      DISBUF[13]=':';
    DISBUF[14] = ((HEXBUF[2] & 0xf0) >> 4)+0x30;
    DISBUF[15] = (HEXBUF[2] & 0x0f)+0x30;
}



unsigned char hex2dec(unsigned char xx){
      unsigned char yy;
      yy = xx & 0x0f;
      xx = xx & 0xf0;
      if (yy<0x0A) return (xx+yy);
      else
      switch (yy)
      {
      case 0x0A:        return(xx+0x10);
      case 0x0B:        return(xx+0x11);
      case 0x0C:        return(xx+0x12);
      case 0x0D:        return(xx+0x13);
      case 0x0E:        return(xx+0x14);
      case 0x0F:        return(xx+0x15);
      }
}

/************TEMPERATURE DS1820***********/
void tmreset (void) {                   // Reset TX
    unsigned int i;
    TMDAT = 0;
    i = 103; while (i>0) i--;           // Approx 900 uS
    TMDAT = 1;
    i = 4; while (i>0) i--;
}

void tmpre (void) {                     // Wait for Presence RX
    unsigned int i;
    while (TMDAT);
    while (~TMDAT);
    i = 4; while (i>0) i--;
}

bit tmrbit (void) {                     // read one bit
    unsigned int i;
    bit dat;
    TMDAT = 0; i++;
    TMDAT = 1; i++; i++;
    dat = TMDAT;
    i = 8; while (i>0) i--;             // Approx 65 uS
    return (dat);
}

unsigned char tmrbyte (void) {          // read one byte
    unsigned char i,j,dat;
    dat = 0;
    for (i=1;i<=8;i++) {
        j = tmrbit ();
        dat = (j << 7) | (dat >> 1);
    }
    return (dat);
}

void tmwbyte (unsigned char dat) {                  // write one byte
```

```
    unsigned int i;
    unsigned char j;
    bit testb;
    for (j=1;j<=8;j++) {
        testb = dat & 0x01;
        dat = dat >> 1;
        if (testb) {
            TMDAT = 0;                          // Write 1
            i++; i++;                           // Approx 4 uS
            TMDAT = 1;
            i = 8; while (i>0) i--;             // Approx 65 uS
        }
        else {
            TMDAT = 0;                          // Write 0
            i = 8; while (i>0) i--;             // Approx 65 uS
            TMDAT = 1;
            i++; i++;                           // Approx 4 uS
        }
    }
}

void tmstart (void) {                   // ds1820 start convert
    tmreset ();
    tmpre ();
    dmsec (1);
    tmwbyte (0xcc);                     // skip rom
    tmwbyte (0x44);                     // convert
}

void tmrtemp (void) {                   // read temp
    unsigned char a,b;
    TEMBUF[0] = 0;
    TEMBUF[1] = 0;
    tmreset ();
    tmpre ();
    dmsec (1);
    tmwbyte (0xcc);                     // skip rom
    tmwbyte (0xbe);                     // convert
    a = tmrbyte ();                     // LSB
    b = tmrbyte ();                     // MSB
    if (b==1) return;                   // don't care negative temp
    TEMBUF[1] = a & 0x1;                // 0=x.0 1=x.5
    a = a >> 1;
    TEMBUF[0] = a + 1;                  // adjust for thailand
}


void wportb(unsigned char xxx){
      P0=xxx;
      P8255A0=1;
      P8255A1=0;
      P8255WR=0;
      P8255WR=1;
}
void wportc(unsigned char xxx){
      P0=xxx;
      P8255A0=0;
      P8255A1=1;
      P8255WR=0;
      P8255WR=1;
}
```

```
void wportx(unsigned char xxx){
      P0=xxx;
      P8255A0=1;
      P8255A1=1;
      P8255WR=0;
      P8255WR=1;
}



/********** MAIN **********/

void main(void) {

      bit power_off;
      dmsec(100);
      INIT_LCD();
      wportx(0x80);
      temperature = 25;
      timeoff_flag = 0;
      turn_off=0;
      fan_level=1;        //L=0, M=1, H=2
      wportb(0x02);
      power_off=1;
      step_value=0x01;
      step_flag = 0;
      step_delay = 20;
      step_count_L=8;
      step_count_R=8;


     while (1) {
            unsigned char key,x,toggle_mode;
            bit loop_key;
            bit loop_time_key;
            bit loop_time2;
            bit time_flag1;
            bit first_time;
            bit time_flag_loop;
            bit err;


            if (power_off==1){
                  tmstart();
                  tmrtemp();
                  if (TEMBUF[0]>temperature){wportc(0xff);}
                  else wportc(0x00);
                  DISBUF[1]= (TEMBUF[0]%10)+0x30;
                  DISBUF[0]= (TEMBUF[0]/10)+0x30;
                  DISBUF[2]='.';
                  if (TEMBUF[1]==1)                  // display .0 or .5
                        DISBUF[3] = '0';
                  else
                        DISBUF[3] = '5';
                  DISBUF[4]=0x2F;
                  DISBUF[6]=(temperature%10)+0x30;
                  DISBUF[5]=(temperature/10)+0x30;
                  DISBUF[7]=0x20;
                  switch (fan_level)      {
```

```
                        case 0x0:
                              DISBUF[8]='L';
                              break;
                        case 0x01:
                              DISBUF[8]='M';
                              break;
                        case 0x02:
                              DISBUF[8]='H';
                              break;
                  }
                  DISBUF[9]=0x20;
                  if (timeoff_flag==1){
                        DISBUF[10]=((TIMEOFFBUF[0] & 0xf0) >>4)+0x30;
                        DISBUF[11]=(TIMEOFFBUF[0] & 0x0f) +0x30;
                        DISBUF[12]=':';
                        DISBUF[13]=((TIMEOFFBUF[1] & 0xf0) >>4)+0x30;
                        DISBUF[14]=(TIMEOFFBUF[1] & 0x0f)+0x30;
                        DISBUF[15]=0x20;
                  }
                  else{
                        DISBUF[10]='T';
                        DISBUF[11]='I';
                        DISBUF[12]='=';
                        DISBUF[13]='O';
                        DISBUF[14]='F';
                        DISBUF[15]='F';
                  }
                  display_LCD(DISBUF);
            }
            else {
                  while(1){

                        wportb(0x0);
                        wportc(0x0);
                        step_flag=0;
                        x = 0xff;
                        err = rtrd ();                          // display
hh.mm.ss

                        if (TIMBUF[0]!=x) {                     // check second
change
                              strncpy(DISBUF,"P_off :       ",16);
                           HEXBUF[0] = TIMBUF[2];
                           HEXBUF[1] = TIMBUF[1];
                           HEXBUF[2] = TIMBUF[0];
                           htolcd ();
                             display_LCD(DISBUF);
                        }
                        key = scan_key();
                        if (key==0x04) {
                              power_off=1; dmsec(1500);wportc(0xff);
                              wportb(0x02);fan_level=1; break;}
                  }
            }


            if (timeoff_flag==1){
                  x=0xff;
                  err = rtrd ();                          // display hh.mm.ss
                  if (TIMBUF[0]!=x) {                     // check second
change
```

```
                if (TIMBUF[2]==TIMEOFFBUF[0])
                      if (TIMBUF[1]==TIMEOFFBUF[1]){
                            timeoff_flag=0;
                            power_off=0;

                            wportb(0x0);
                            wportc(0x0);
                      }
        }
}


key = scan_key();
if (key != 0xFF)
{
        switch (key){
        case 0x00:
              if (step_flag==0) {step_flag=1;dmsec(1000);}
              else {step_flag=0;dmsec(1000);}
              break;
        case 0x01:
              wportb(0x01);
              fan_level=2;
              dmsec(1500);
              break;
        case 0x02:
              wportb(0x02);
              fan_level=1;
              dmsec(1500);
              break;
        case 0x03:
              wportb(0x04);
              fan_level=0;
              dmsec(1500);
              break;
        case 0x04:
              if (power_off)
                    power_off=0;
              else
                    power_off=1;
              dmsec(1000);
              break;
        case 0x05:
              toggle_mode=0;
              loop_key=1;
              while(loop_key){
              switch (key)
              {
              case 0x05:
              if (toggle_mode==0)     {
                    toggle_mode=1;
                    strncpy(DISBUF,"Time off :      ",16);
                    display_LCD(DISBUF);
                    dmsec(1500);
                    loop_time2=1;
                    while(loop_time2){
                          key=scan_key();
                                if (key==0x05){
                                      loop_time2=0;
                                      }
```

```
                                                    else if (key==0x09){
                                                            dmsec(1000);
                                                            first_time=0;
                                                            time_flag1=0;
                                                            time_flag_loop=1;

        while(time_flag_loop){
                                                             if
  (first_time==0) {first_time=1;key=0x05;}
                                                             else
        key=scan_key();
                                                             switch (key){
                                                             case 0x05:
  (time_flag1==0){                                           if

        time_flag1=1;

        timeoff_flag=0;

        strncpy(DISBUF,"Disable          ",16);

        display_LCD(DISBUF);

        dmsec(1500);
                                                             }
                                                             else {

        time_flag1=0;

        timeoff_flag=1;

        strncpy(DISBUF,"Enable           ",16);

        display_LCD(DISBUF);

        dmsec(1500);
                                                              }
                                                              break;
                                                    case 0x09:
                                                              if
 (timeoff_flag==0){

        time_flag_loop=0;

        loop_time2 = 0;

        loop_key=0;

        dmsec(1000);

        strncpy(DISBUF,"Time :          ",16);

                                                             }
                                                             else{
                                                             dmsec(1000);
                                                             x = 0xff;
                                                             err = rtrd ();
// display hh.mm.ss
                                                             if (err)
{strncpy(DISBUF,"error  ...      ",16);display_LCD(DISBUF);}
```

```
                                                                    if
(TIMBUF[0]!=x) {                        // check second change
      HEXBUF[0] = TIMBUF[2];
      HEXBUF[1] = TIMBUF[1];
      HEXBUF[2] = TIMBUF[0];
      strncpy(DISBUF,"H:M:S          ",16);
                                                                    htolcd
();
      display_LCD(DISBUF);
      loop_time_key=1;
      while(loop_time_key){
      key=scan_key();
      switch (key){
      case 0x06:
            TIMBUF[2] += 1;
            TIMBUF[2] = hex2dec(TIMBUF[2]);
            if (TIMBUF[2]==0x24) TIMBUF[2]=0;
                  HEXBUF[0] = TIMBUF[2];
                  HEXBUF[1] = TIMBUF[1];
                  HEXBUF[2] = TIMBUF[0];
                  strncpy(DISBUF,"H:M:S          ",16);
                  htolcd ();
                  display_LCD(DISBUF);
                  dmsec(900);
                  break;
            case 0x07:
                  TIMBUF[1] += 1;
                  TIMBUF[1] = hex2dec(TIMBUF[1]);
                  if (TIMBUF[1]==0x60) TIMBUF[1]=0;
                        HEXBUF[0] = TIMBUF[2];
                        HEXBUF[1] = TIMBUF[1];
                        HEXBUF[2] = TIMBUF[0];
```

```
                strncpy(DISBUF,"H:M:S          ",16);
                htolcd ();
                display_LCD(DISBUF);
                dmsec(900);
                break;
case 0x08:
      TIMBUF[0] += 1;
      TIMBUF[0] = hex2dec(TIMBUF[0]);
      if (TIMBUF[0]==0x60) TIMBUF[0]=0;
      HEXBUF[0] = TIMBUF[2];
      HEXBUF[1] = TIMBUF[1];
      HEXBUF[2] = TIMBUF[0];
      strncpy(DISBUF,"H:M:S          ",16);
      htolcd ();
          display_LCD(DISBUF);
          dmsec(900);
          break;
      case 0x09:

          TIMEOFFBUF[0] = TIMBUF[2];
          TIMEOFFBUF[1] = TIMBUF[1];
          TIMEOFFBUF[2] = TIMBUF[0];
          x = 0xff;
          dmsec(900);
          strncpy(DISBUF,"Time off success",16);
          display_LCD(DISBUF);
          dmsec(1500);
          loop_time_key=0;
          loop_time2=0;
          loop_key=0;
```

```
                    time_flag_loop=0;


                    break;
                }
            }
                                                                    }
                                                                }
                                                                break;
                                                            }
                                                        }
                                                    }
                                                }
                                            }///Time off
                                            else if(toggle_mode==1){
                                                toggle_mode=2;
                                                strncpy(DISBUF,"Temperature :    ",16);
                                                display_LCD(DISBUF);
                                                dmsec(1500);
                                                loop_time2=1;
                                                while(loop_time2){
                                                    key=scan_key();
                                                    if (key==0x05){
                                                        loop_time2=0;
                                                    }
                                                    else if (key==0x09){
                                                        dmsec(1000);
                                                        strncpy(DISBUF,"Temp
:          C",16);
                                                        DISBUF[14]=0xDF;
        DISBUF[13]=(temperature%10)+0x30;
        DISBUF[12]=(temperature/10)+0x30;
                                                        display_LCD(DISBUF);
                                                        loop_time_key=1;
        while(loop_time_key){
        key=scan_key();
                                                        switch (key){
                                                        case 0x06:
        temperature += 1;
                                                            if
(temperature==31) {temperature=18;}
        DISBUF[13]=(temperature%10)+0x30;
        DISBUF[12]=(temperature/10)+0x30;
        display_LCD(DISBUF);
        dmsec(900);
                                                            break;
                                                        case 0x07:
```

```
        temperature -= 1;
                                                                    if
  (temperature==17) {temperature=30;}
        DISBUF[13]=(temperature%10)+0x30;
        DISBUF[12]=(temperature/10)+0x30;
        display_LCD(DISBUF);
        dmsec(900);
                                                                    break;
                                                                case 0x09:
        dmsec(900);
        loop_time_key=0;
        loop_time2=0;
        loop_key=0;
        strncpy(DISBUF,"Set temp success",16);
        display_LCD(DISBUF);
        dmsec(1500);
                                                                    break;
                                                                }
                                                              }
                                                            }
                                                          }
                                                        }
                                                      else if(toggle_mode==2){
                                                        toggle_mode=3;
                                                        strncpy(DISBUF,"Set time mode    ",16);
                                                        display_LCD(DISBUF);
                                                        dmsec(1500);
                                                        loop_time2=1;
                                                        while(loop_time2){
                                                          key=scan_key();
                                                            if (key==0x05){
                                                              loop_time2=0;
                                                            }
                                                            else if (key==0x09){
                                                              dmsec(1000);
                                                              x = 0xff;
                                                              err = rtrd ();
// display hh.mm.ss
                                                              if (err)
{strncpy(DISBUF,"error  ...      ",16);display_LCD(DISBUF);}
                                                              if (TIMBUF[0]!=x) {
// check second change
                                                                HEXBUF[0] =
TIMBUF[2];
                                                                HEXBUF[1] =
TIMBUF[1];
```

```
                                                        HEXBUF[2] =
TIMBUF[0];
      strncpy(DISBUF,"H:M:S          ",16);
                                                        htolcd ();
      display_LCD(DISBUF);
      loop_time_key=1;
      while(loop_time_key){
      key=scan_key();
                                                        switch
(key){
      case 0x06:
            TIMBUF[2] += 1;
            TIMBUF[2] = hex2dec(TIMBUF[2]);
            if (TIMBUF[2]==0x24) TIMBUF[2]=0;
            HEXBUF[0] = TIMBUF[2];
            HEXBUF[1] = TIMBUF[1];
            HEXBUF[2] = TIMBUF[0];
            strncpy(DISBUF,"H:M:S          ",16);
            htolcd ();
            display_LCD(DISBUF);
            dmsec(900);
            break;
     case 0x07:
            TIMBUF[1] += 1;
            TIMBUF[1] = hex2dec(TIMBUF[1]);
            if (TIMBUF[1]==0x60) TIMBUF[1]=0;
            HEXBUF[0] = TIMBUF[2];
            HEXBUF[1] = TIMBUF[1];
            HEXBUF[2] = TIMBUF[0];
            strncpy(DISBUF,"H:M:S          ",16);
            htolcd ();
            display_LCD(DISBUF);
            dmsec(900);
```

```
        break;
    case 0x08:
        TIMBUF[0] += 1;
        TIMBUF[0] = hex2dec(TIMBUF[0]);
        if (TIMBUF[0]==0x60) TIMBUF[0]=0;
        HEXBUF[0] = TIMBUF[2];
        HEXBUF[1] = TIMBUF[1];
        HEXBUF[2] = TIMBUF[0];
        strncpy(DISBUF,"H:M:S           ",16);
        htolcd ();
        display_LCD(DISBUF);
        dmsec(900);
        break;
    case 0x09:
        rtset ();
        rtwr();
        x = 0xff;
        dmsec(900);
        strncpy(DISBUF,"Set time success",16);
        display_LCD(DISBUF);
        dmsec(1500);
        loop_time_key=0;
        loop_time2=0;
        loop_key=0;

        break;
                                                    }
                                                  }
                                                }
                                              }// if key==0x09
                                    }
                                }
                                else  {loop_key=0;dmsec(1000);
                                ;
                                }
```

```
                        break;
                    }
                    key=scan_key();
                }
                break;
            }
        }
    }
}
```

80C51 8-bit Flash microcontroller family
16KB/32KB/64KB ISP/IAP Flash with 512B/512B/1KB RAM

P89C51RB2/P89C51RC2/ P89C51RD2Hxx

DESCRIPTION

The P89C51RB2/RC2/RD2Hxx device contains a non-volatile 16KB/32KB/64KB Flash program memory that is both parallel programmable and serial In-System and In-Application Programmable. In-System Programming (ISP) allows the user to download new code while the microcontroller sits in the application. In-Application Programming (IAP) means that the microcontroller fetches new program code and reprograms itself while in the system. This allows for remote programming over a modem link. A default serial loader (boot loader) program in ROM allows serial In-System programming of the Flash memory via the UART without the need for a loader in the Flash code. For In-Application Programming, the user program erases and reprograms the Flash memory by use of standard routines contained in ROM.

This device executes one machine cycle in 6 clock cycles, hence providing twice the speed of a conventional 80C51. An OTP configuration bit lets the user select conventional 12 clock timing if desired.

This device is a Single-Chip 8-Bit Microcontroller manufactured in advanced CMOS process and is a derivative of the 80C51 microcontroller family. The instruction set is 100% compatible with the 80C51 instruction set.

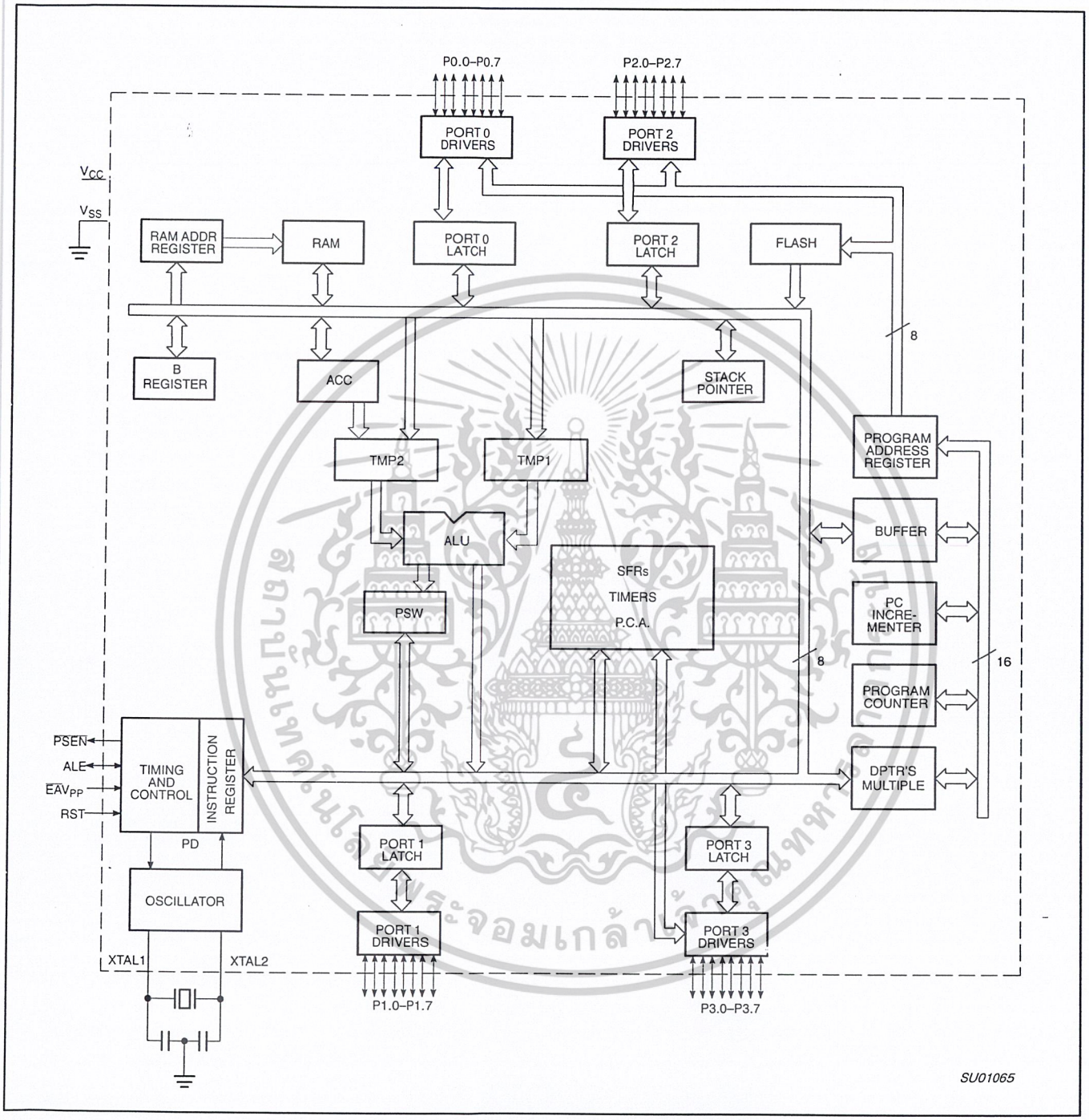
The device also has four 8-bit I/O ports, three 16-bit timer/event counters, a multi-source, four-priority-level, nested interrupt structure, an enhanced UART and on-chip oscillator and timing circuits.

The added features of the P89C51RB2/RC2/RD2Hxx makes it a powerful microcontroller for applications that require pulse width modulation, high-speed I/O and up/down counting capabilities such as motor control.

FEATURES

- 80C51 Central Processing Unit
- On-chip Flash Program Memory with In-System Programming (ISP) and In-Application Programming (IAP) capability
- Boot ROM contains low level Flash programming routines for downloading via the UART
- Can be programmed by the end-user application (IAP)
- Parallel programming with 87C51 compatible hardware interface to programmer
- Six clocks per machine cycle operation (standard)
- 12 clocks per machine cycle operation (optional)
- Speed up to 20 MHz with 6 clock cycles per machine cycle (40 MHz equivalent performance); up to 33 MHz with 12 clocks per machine cycle
- Fully static operation
- RAM expandable externally to 64 kbytes
- Four interrupt priority levels
- Seven interrupt sources
- Four 8-bit I/O ports
- Full-duplex enhanced UART
  - Framing error detection
  - Automatic address recognition
- Power control modes
  - Clock can be stopped and resumed
  - Idle mode
  - Power down mode
- Programmable clock out
- Second DPTR register
- Asynchronous port reset
- Low EMI (inhibit ALE)
- Programmable Counter Array (PCA)
  - PWM
  - Capture/compare

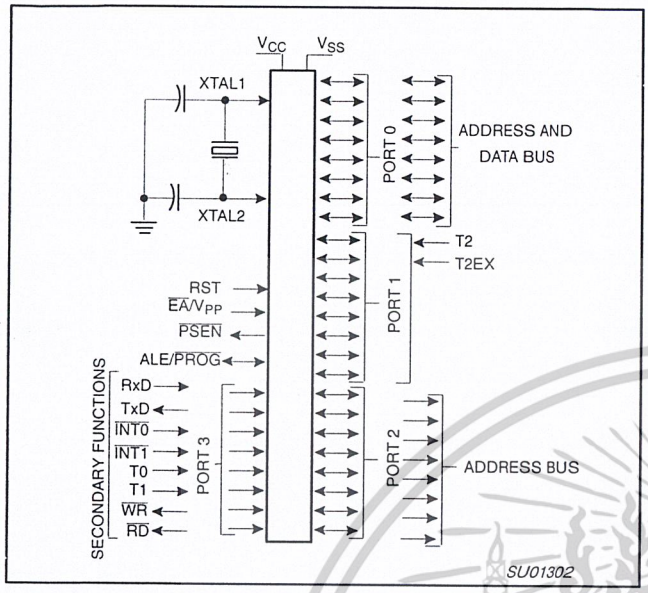
BLOCK DIAGRAM



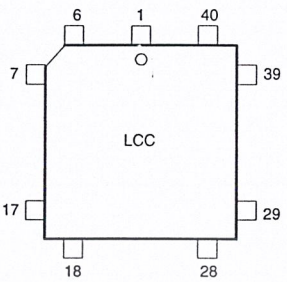
ORDERING INFORMATION

	PART ORDER NUMBER	MEMORY		TEMPERATURE RANGE (°C) AND PACKAGE	VOLTAGE RANGE	FREQUENCY (MHz)		DWG #
		FLASH	RAM			6 CLOCK MODE	12 CLOCK MODE	
1	P89C51RB2HBA	16 kB	512 B	0 to +70, PLCC	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
2	P89C51RB2HBBD	16 kB	512 B	0 to +70, LQFP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT389-1
3	P89C51RC2HBP	32 kB	512 B	0 to +70, PDIP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT129-1
4	P89C51RC2HBA	32 kB	512 B	0 to +70, PLCC	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
5	P89C51RC2HFA	32 kB	512 B	–40 to +85, PLCC	4.75–5.25 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
6	P89C51RC2HBBD	32 kB	512 B	0 to +70, LQFP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT389-1
7	P89C51RC2HFBD	32 kB	512 B	–40 to +85, LQFP	4.75–5.25 V	0 to 20 MHz	0 to 33 MHz	SOT389-1
8	P89C51RD2HBP	64 kB	1 kB	0 to +70, PDIP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT129-1
9	P89C51RD2HBA	64 kB	1 kB	0 to +70, PLCC	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
10	P89C51RD2HBBD	64 kB	1 kB	0 to +70, LQFP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT389-1

LOGIC SYMBOL



Plastic Leaded Chip Carrier



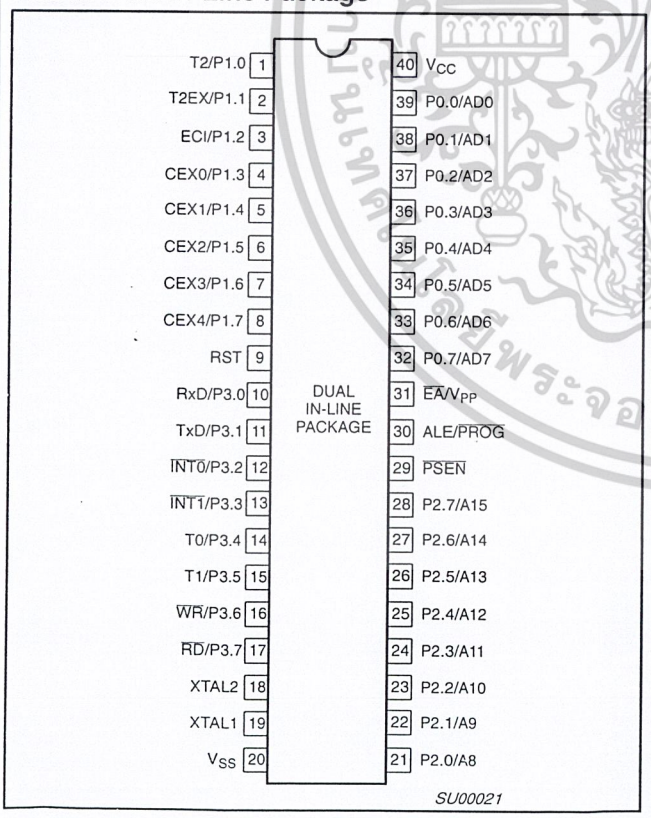
Pin	Function	Pin	Function	Pin	Function
1	NIC*	16	P3.4/T0	31	P2.7/A15
2	P1.0/T2	17	P3.5/T1	32	PSEN
3	P1.1/T2EX	18	P3.6/WR	33	ALE/PROG
4	P1.2/ECI	19	P3.7/RD	34	NIC*
5	P1.3/CEX0	20	XTAL2	35	EA/VPP
6	P1.4/CEX1	21	XTAL1	36	P0.7/AD7
7	P1.5/CEX2	22	VSS	37	P0.6/AD6
8	P1.6/CEX3	23	NIC*	38	P0.5/AD5
9	P1.7/CEX4	24	P2.0/A8	39	P0.4/AD4
10	RST	25	P2.1/A9	40	P0.3/AD3
11	P3.0/RxD	26	P2.2/A10	41	P0.2/AD2
12	NIC*	27	P2.3/A11	42	P0.1/AD1
13	P3.1/TxD	28	P2.4/A12	43	P0.0/AD0
14	P3.2/INT0	29	P2.5/A13	44	VCC
15	P3.3/INT1	30	P2.6/A14		

* NO INTERNAL CONNECTION

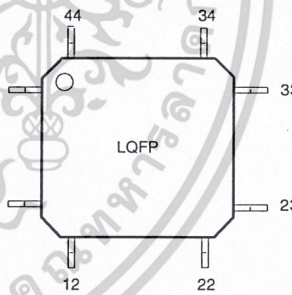
SU00023

PINNING

Plastic Dual In-Line Package



Plastic Quad Flat Pack



Pin	Function	Pin	Function	Pin	Function
1	P1.5/CEX2	16	VSS	31	P0.6/AD6
2	P1.6/CEX3	17	NIC*	32	P0.5/AD5
3	P1.7/CEX4	18	P2.0/A8	33	P0.4/AD4
4	RST	19	P2.1/A9	34	P0.3/AD3
5	P3.0/RxD	20	P2.2/A10	35	P0.2/AD2
6	NIC*	21	P2.3/A11	36	P0.1/AD1
7	P3.1/TxD	22	P2.4/A12	37	P0.0/AD0
8	P3.2/INT0	23	P2.5/A13	38	VCC
9	P3.3/INT1	24	P2.6/A14	39	NIC*
10	P3.4/T0	25	P2.7/A15	40	P1.0/T2
11	P3.5/T1	26	PSEN	41	P1.1/T2EX
12	P3.6/WR	27	ALE/PROG	42	P1.2/ECI
13	P3.7/RD	28	NIC*	43	P1.3/CEX0
14	XTAL2	29	EA/VPP	44	P1.4/CEX1
15	XTAL1	30	P0.7/AD7		

* NO INTERNAL CONNECTION

SU01400

PIN DESCRIPTIONS

MNEMONIC	PIN NUMBER			TYPE	NAME AND FUNCTION
	PDIP	PLCC	LQFP		
V_{SS}	20	22	16	I	**Ground:** 0 V reference.
V_{CC}	40	44	38	I	**Power Supply:** This is the power supply voltage for normal, idle, and power-down operation.
P0.0–0.7	39–32	43–36	37–30	I/O	**Port 0:** Port 0 is an open-drain, bidirectional I/O port. Port 0 pins that have 1s written to them float and can be used as high-impedance inputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external program and data memory. In this application, it uses strong internal pull-ups when emitting 1s.
P1.0–P1.7	1–8	2–9	40–44, 1–3	I/O	**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pull-ups on all pins. Port 1 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 1 pins that are externally pulled low will source current because of the internal pull-ups. (See DC Electrical Characteristics: I_{IL}).
					Alternate functions for P89C51RB2/RC2/RD2Hxx Port 1 include:
	1	2	40	I/O	**T2 (P1.0):** Timer/Counter 2 external count input/Clockout (see Programmable Clock-Out)
	2	3	41	I	**T2EX (P1.1):** Timer/Counter 2 Reload/Capture/Direction Control
	3	4	42	I	**ECI (P1.2):** External Clock Input to the PCA
	4	5	43	I/O	**CEX0 (P1.3):** Capture/Compare External I/O for PCA module 0
	5	6	44	I/O	**CEX1 (P1.4):** Capture/Compare External I/O for PCA module 1
	6	7	1	I/O	**CEX2 (P1.5):** Capture/Compare External I/O for PCA module 2
	7	8	2	I/O	**CEX3 (P1.6):** Capture/Compare External I/O for PCA module 3
	8	9	3	I/O	**CEX4 (P1.7):** Capture/Compare External I/O for PCA module 4
P2.0–P2.7	21–28	24–31	18–25	I/O	**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. Port 2 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 2 pins that are externally being pulled low will source current because of the internal pull-ups. (See DC Electrical Characteristics: I_{IL}). Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @DPTR). In this application, it uses strong internal pull-ups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOV @Ri), port 2 emits the contents of the P2 special function register. P2.7 must be a "1" to program and erase the device.
P3.0–P3.7	10–17	11, 13–19	5, 7–13	I/O	**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pull-ups. Port 3 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 3 pins that are externally being pulled low will source current because of the pull-ups. (See DC Electrical Characteristics: I_{IL}). Port 3 also serves the special features of the P89C51RB2/RC2/RD2Hxx, as listed below:
	10	11	5	I	**RxD (P3.0):** Serial input port
	11	13	7	O	**TxD (P3.1):** Serial output port
	12	14	8	I	**$\overline{\text{INT0}}$ (P3.2):** External interrupt
	13	15	9	I	**$\overline{\text{INT1}}$ (P3.3):** External interrupt
	14	16	10	I	**T0 (P3.4):** Timer 0 external input
	15	17	11	I	**T1 (P3.5):** Timer 1 external input
	16	18	12	O	**$\overline{\text{WR}}$ (P3.6):** External data memory write strobe
	17	19	13	O	**$\overline{\text{RD}}$ (P3.7):** External data memory read strobe
RST	9	10	4	I	**Reset:** A high on this pin for two machine cycles while the oscillator is running, resets the device. An internal resistor to V_{SS} permits a power-on reset using only an external capacitor to V_{CC}.
ALE	30	33	27	O	**Address Latch Enable:** Output pulse for latching the low byte of the address during an access to external memory. In normal operation, ALE is emitted twice every machine cycle, and can be used for external timing or clocking. Note that one ALE pulse is skipped during each access to external data memory. ALE can be disabled by setting SFR auxiliary.0. With this bit set, ALE will be active only during a MOVX instruction.

MNEMONIC	PIN NUMBER			TYPE	NAME AND FUNCTION
	PDIP	PLCC	LQFP		
$\overline{PSEN}$	29	32	26	O	**Program Store Enable:** The read strobe to external program memory. When executing code from the external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory. $\overline{PSEN}$ is not activated during fetches from internal program memory.
$\overline{EA}/V_{PP}$	31	35	29	I	**External Access Enable/Programming Supply Voltage:** $\overline{EA}$ must be externally held low to enable the device to fetch code from external program memory locations. If $\overline{EA}$ is held high, the device executes from internal program memory. The value on the $\overline{EA}$ pin is latched when RST is released and any subsequent changes have no effect. This pin also receives the programming supply voltage (V_{PP}) during Flash programming.
XTAL1	19	21	15	I	**Crystal 1:** Input to the inverting oscillator amplifier and input to the internal clock generator circuits.
XTAL2	18	20	14	O	**Crystal 2:** Output from the inverting oscillator amplifier.

NOTE:
To avoid "latch-up" effect at power-on, the voltage on any pin (other than V_{PP}) must not be higher than V_{CC} + 0.5 V or less than V_{SS} – 0.5 V.

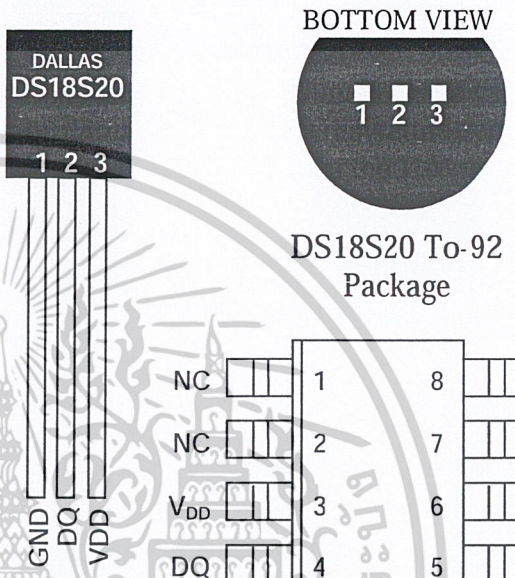
DALLAS SEMICONDUCTOR

DS18S20
High Precision 1-Wire® Digital Thermometer

FEATURES

- Unique 1–Wire interface requires only one port pin for communication
- Multidrop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line. Power supply range is 3.0V to 5.5V
- Zero standby power required
- Measures temperatures from –55°C to +125°C. Fahrenheit equivalent is –67°F to +257°F
- ±0.5°C accuracy from –10°C to +85°C
- Temperature is read as a 9–bit digital value
- Converts temperature to digital word in 750 ms (max.)
- User–definable, nonvolatile temperature alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Functionally compatible with DS1820 1-Wire digital thermometer
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



DS18S20 To-92 Package

DS18S20Z
8-Pin SOIC (150 mil)

PIN DESCRIPTION

GND - Ground
DQ - Data In/Out
V_{DD} - Power Supply Voltage
NC - No Connect

DESCRIPTION

The DS18S20 Digital Thermometer provides 9–bit temperature readings which indicate the temperature of the device.

Information is sent to/from the DS18S20 over a 1–Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS18S20. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS18S20 contains a unique silicon serial number, multiple DS18S20s can exist on the same 1–Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls, sensing temperatures inside buildings, equipment or machinery, and process monitoring and control.

DETAILED PIN DESCRIPTION

PIN 8-PIN SOIC	PIN TO92	SYMBOL	DESCRIPTION
5	1	GND	Ground.
4	2	DQ	**Data Input/Output pin.** For 1-Wire operation: Open drain. (See "Parasite Power" section.)
3	3	V_{DD}	**Optional V_{DD} pin.** See "Parasite Power" section for details of connection. V_{DD} must be grounded for operation in parasite power mode.

DS18S20Z (8-pin SOIC): All pins not specified in this table are not to be connected.

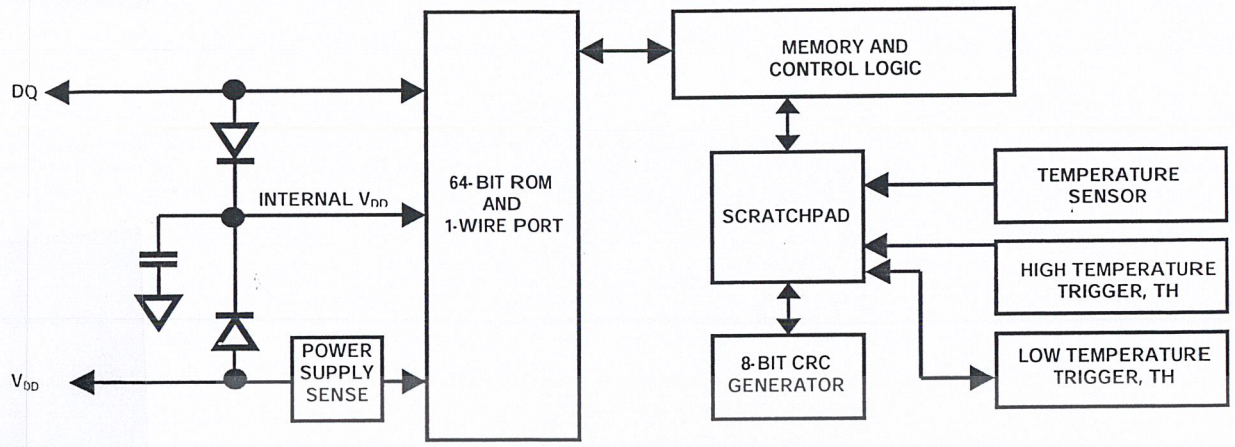
OVERVIEW

The block diagram of Figure 1 shows the major components of the DS18S20. The DS18S20 has three main data components: 1) 64–bit lasered ROM, 2) temperature sensor, 3) nonvolatile temperature alarm triggers TH and TL. The device derives its power from the 1–Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1–Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS18S20 may also be powered from an external 3 volt – 5 volt supply.

Communication to the DS18S20 is via a 1–Wire port. With the 1–Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64–bit lasered ROM portion of each device and can single out a specific device if many are present on the 1–Wire line as well as indicate to the bus master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands.

One control function command instructs the DS18S20 to perform a temperature measurement. The result of this measurement will be placed in the DS18S20's scratch-pad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of 1-byte EEPROM each. If the alarm search command is not applied to the DS18S20, these registers may be used as general purpose user memory. Writing TH and TL is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.

DS18S20 BLOCK DIAGRAM Figure 1



PARASITE POWER

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

In order for the DS18S20 to be able to perform accurate temperature conversions, sufficient power must be provided over the DQ line when a temperature conversion is taking place. Since the operating current of the DS18S20 is up to 1.5 mA, the DQ line will not have sufficient drive due to the 5k pullup resistor. This problem is particularly acute if several DS18S20s are on the same DQ and attempting to convert simultaneously.

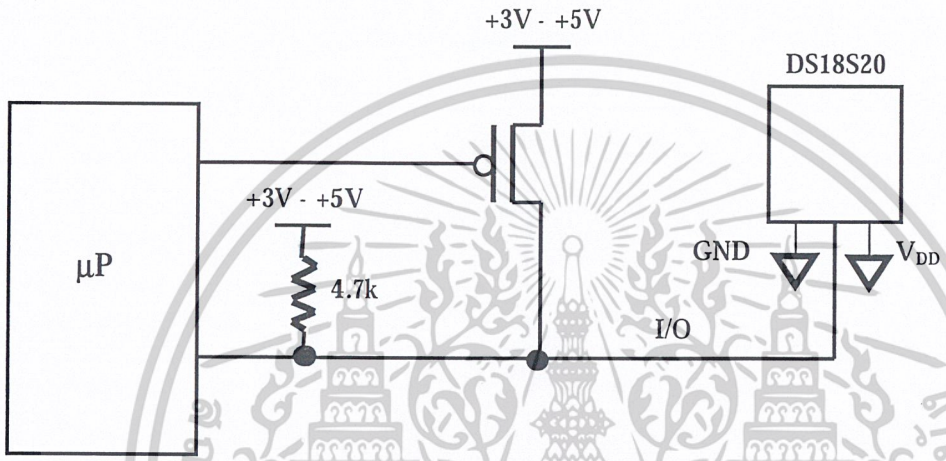
There are two ways to assure that the DS18S20 has sufficient supply current during its active conversion cycle. The first is to provide a strong pullup on the DQ line whenever temperature conversions or copies to the E^2 memory are taking place. This may be accomplished by using a MOSFET to pull the DQ line directly to the power supply as shown in Figure 2. The DQ line must be switched over to the strong pull-up within 10 µs maximum after issuing any protocol that involves copying to the E^2 memory or initiates temperature conversions. When using the parasite power mode, the V_{DD} pin must be tied to ground.

Another method of supplying current to the DS18S20 is through the use of an external power supply tied to the V_{DD} pin, as shown in Figure 3. The advantage to this is that the strong pullup is not required on the DQ line, and the bus master need not be tied up holding that line high during temperature conversions. This allows other data traffic on the 1-Wire bus during the conversion time. In addition, any number of DS18S20s may be placed on the 1-Wire bus, and if they all use external power, they may all simultaneously perform temperature conversions by issuing the Skip ROM command and then issuing the Convert T command. Note that as long as the external power supply is active, the GND pin may not be floating.

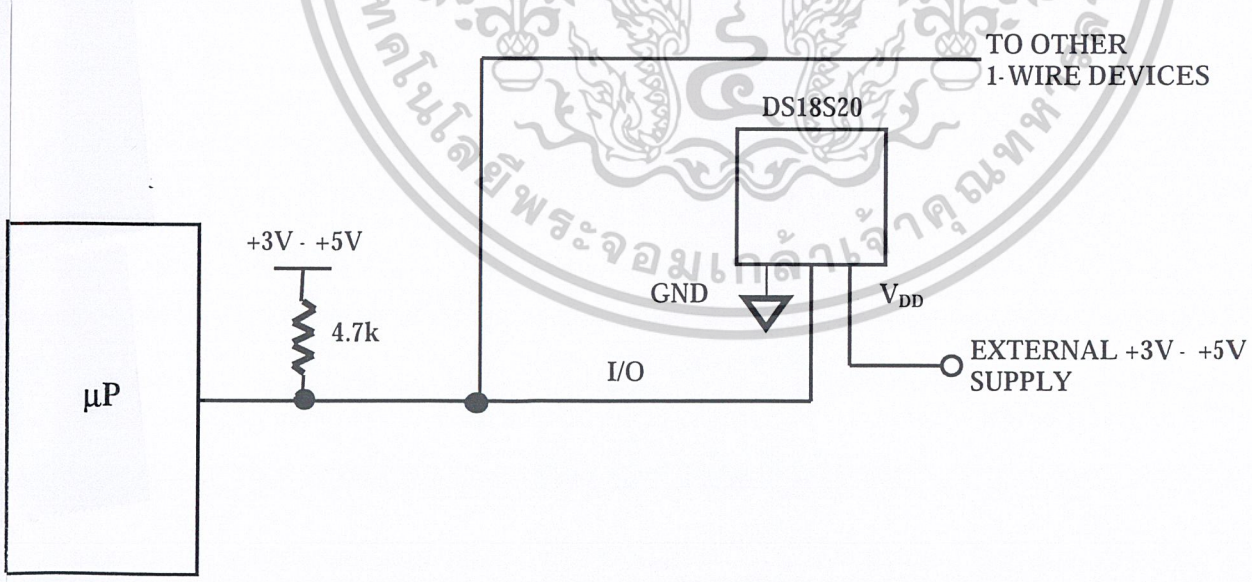
The use of parasite power is not recommended above 100°C, since it may not be able to sustain communications given the higher leakage currents the DS18S20 exhibits at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that V_{DD} be applied to the DS18S20.

For situations where the bus master does not know whether the DS18S20s on the bus are parasite powered or supplied with external V_{DD}, a provision is made in the DS18S20 to signal the power supply scheme used. The bus master can determine if any DS18S20s are on the bus which require the strong pullup by sending a Skip ROM protocol, then issuing the read power supply command. After this command is issued, the master then issues read time slots. The DS18S20 will send back "0" on the 1-Wire bus if it is parasite powered; it will send back a "1" if it is powered from the V_{DD} pin. If the master receives a "0," it knows that it must supply the strong pullup on the DQ line during temperature conversions. See "Memory Command Functions" section for more detail on this command protocol.

STRONG PULL-UP FOR SUPPLYING DS18S20 DURING TEMPERATURE CONVERSION Figure 2



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



DALLAS SEMICONDUCTOR

DS1307
64 X 8 Serial Real Time Clock

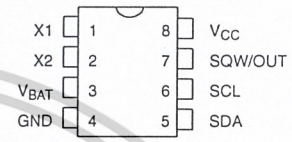
FEATURES

- Real time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap year compensation valid up to 2100
- 56 byte nonvolatile RAM for data storage
- 2–wire serial interface
- Programmable squarewave output signal
- Automatic power fail detect and switch circuitry
- Consumes less than 500 nA in battery backup mode at 25°C
- Optional industrial temperature range –40°C to +85°C (IND)
- Available in 8–pin DIP or SOIC

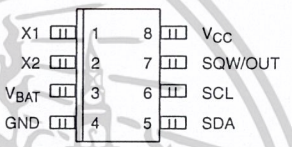
ORDERING INFORMATION

DS1307	Serial Timekeeping Chip; 8–pin DIP
DS1307Z	Serial Timekeeping Chip; 8–pin SOIC (150 mil)
DS1307N	8–pin DIP (IND)
DS1307ZN	8–pin SOIC (IND)

PIN ASSIGNMENT



DS1307 8–PIN DIP (300 MIL)



DS1307Z 8–PIN SOIC (150 MIL)

PIN DESCRIPTION

V_{CC}	– Primary Power Supply
X1, X2	– 32.768 KHz Crystal Connection
V_{BAT}	– +3 Volt Battery Input
GND	– Ground
SDA	– Serial Data
SCL	– Serial Clock
SQW/OUT	– Square wave/Output Driver

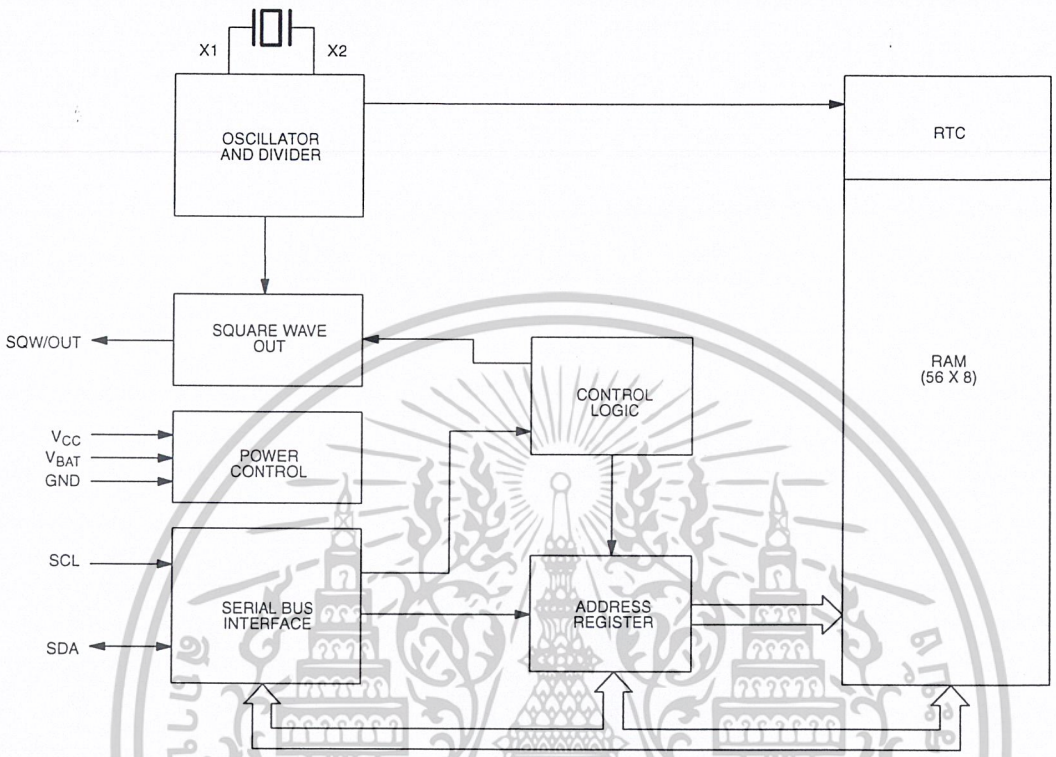
DESCRIPTION

The DS1307 Serial Real Time Clock is a low power full BCD clock calendar plus 56 bytes of nonvolatile SRAM. Address and data are transferred serially via a 2–wire bi–directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24–hour or 12–hour format with AM/PM indicator. The DS1307 has a built–in power sense circuit which detects power failures and automatically switches to the battery supply.

OPERATION

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT} the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When V_{CC} falls below V_{BAT} the device switches into a low current battery backup mode. Upon power up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT}+0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the Serial Real Time Clock. The following paragraphs describe the function of each pin.

DS1307 BLOCK DIAGRAM Figure 1



SIGNAL DESCRIPTIONS

V_{CC}, GND – DC power is provided to the device on these pins. V_{CC} is the +5 volt input. When 5 volts are applied within normal limits, the device is fully accessible and data can be written and read. When a 3 volt battery is connected to the device and V_{CC} is below 1.25 x V_{BAT}, reads and writes are inhibited. However, the Timekeeping function continues unaffected by the lower input voltage. As V_{CC} falls below V_{BAT} the RAM and timekeeper are switched over to the external 3 volt battery.

V_{BAT} – Battery input for any standard 3 volt lithium cell or other energy source. Battery voltage must be held between 2.5 and 3.5 volts for proper operation. The nominal write protect trip point voltage at which access to the real time clock and user RAM is denied is set by the internal circuitry as 1.25 x V_{BAT} nominal. A Lithium battery with 35 mAh or greater will back up the DS1307 for more than 10 years in the absence of power.

SCL (Serial Clock Input) – SCL is used to synchronize data movement on the serial interface.

SDA (Serial Data Input/Output) – SDA is the input/output pin for the 2–wire serial interface. The SDA pin is open drain which requires an external pull–up resistor.

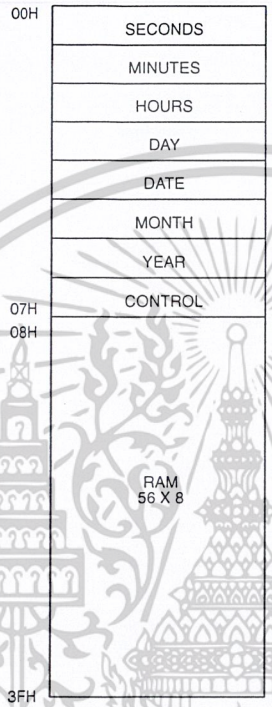
SQW/OUT (Square Wave/ Output Driver) – When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square wave frequencies (1 Hz, 4 KHz, 8 KHz, 32 KHz). The SQW/OUT pin is open drain which requires an external pull–up resistor.

X1, X2 – Connections for a standard 32.768 KHz quartz crystal. The internal oscillator circuitry is designed for operation with a crystal having a specified load capacitance (CL) of 12.5 pF.

RTC AND RAM ADDRESS MAP

The address map for the RTC and RAM registers of the DS1307 is shown in Figure 2. The real time clock registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multibyte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

DS1307 ADDRESS MAP Figure 2



CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. The real time clock registers are illustrated in Figure 3. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the Binary–Coded Decimal (BCD) format. Bit 7 of Register 0 is the Clock Halt (CH) bit. When this bit is set to a one, the oscillator is disabled. When cleared to a zero, the oscillator is enabled.

The DS1307 can be run in either 12–hour or 24–hour mode. Bit 6 of the hours register is defined as the 12– or 24–hour mode select bit. When high, the 12–hour mode is selected. In the 12–hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24–hour mode, bit 5 is the second 10 hour bit (20–23 hours).

DS1307 TIMEKEEPER REGISTERS Figure 3

	BIT7						BIT0	
00H	CH	10 SECONDS			SECONDS			00–59
	X	10 MINUTES			MINUTES			00–59
	X	12/24	10 HR A/P	10 HR	HOURS			01–12 00–23
	X	X	X	X	X	DAY		1–7
	X	X	10 DATE		DATE			01–28/29 01–30 01–31
	X	X	10 MONTH		MONTH			01–12
	10 YEAR				YEAR			00–99
07H	OUT	X	X	SQWE	X	X	RS1	RS0

CONTROL REGISTER

The DS1307 Control Register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	X	X	SQWE	X	X	RS1	RS0

OUT (Output control): This bit controls the output level of the SQW/OUT pin when the square wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0.

SQWE (Square wave Enable): This bit when set to a logic 1 will enable the oscillator output. The frequency of the square wave output depends on the value of the RS0 and RS1 bits.

RS (Rate Select): These bits control the frequency of the square wave output when the square wave output has been enabled. Table 1 lists the square wave frequencies that can be selected with the RS bits.

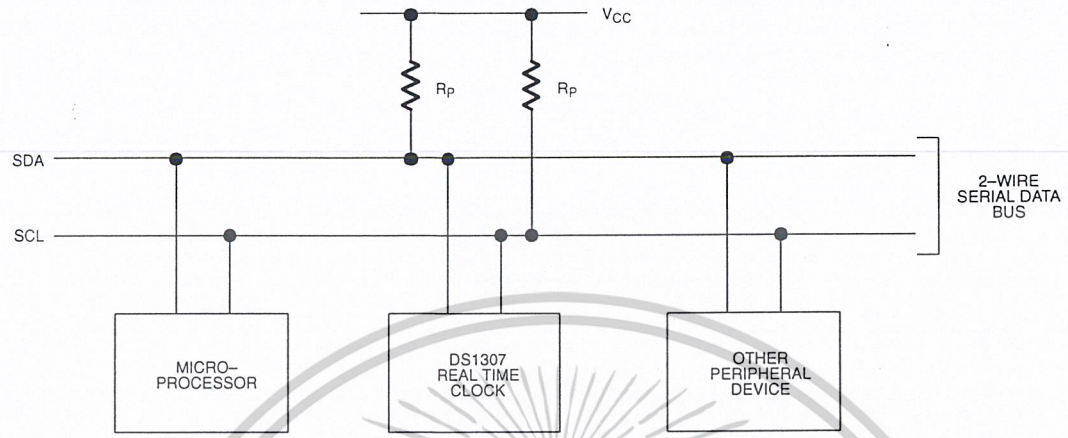
SQUAREWAVE OUTPUT FREQUENCY Table 1

RS1	RS0	SQW OUTPUT FREQUENCY
0	0	1 Hz
0	1	4 KHz
1	0	8 KHz
1	1	32 KHz

2–WIRE SERIAL DATA BUS

The DS1307 supports a bi–directional 2–wire bus and data transmission protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data as a receiver. The device that controls the message is called a master. The devices that are controlled by the master are slaves. The bus must be controlled by a master device which generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a slave on the 2–wire bus. A typical bus configuration using this 2–wire protocol is show in Figure 4.

TYPICAL 2–WIRE BUS CONFIGURATION Figure 4



The following bus protocol has been defined (see Figure 5).

- Data transfer may be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock line is HIGH. Changes in the data line while the clock line is high will be interpreted as control signals.

Accordingly, the following bus conditions have been defined:

Bus not busy: Both data and clock lines remain HIGH.

Start data transfer: A change in the state of the data line from high to low, while the clock line is high, defines a START condition.

Stop data transfer: A change in the state of the data line from low to high, while the clock line is high defines the STOP condition.

Data valid: The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the high period of the clock signal. The data on the line must be changed during the low period of the clock signal. There is one clock pulse per bit of data.

Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between the START and the STOP conditions is not limited, and is determined by the master device. The information is transferred byte–wise and each receiver acknowledges with a ninth bit.

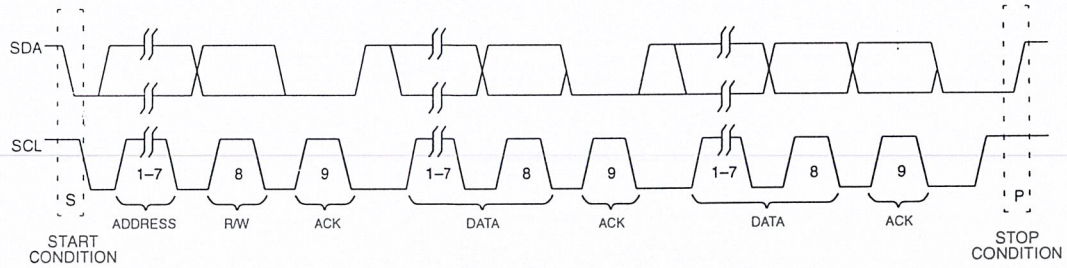
Acknowledge: Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit.

A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable low during the high period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. When receiving data from a slave a master must signal an end of data to the slave by not generating an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line high to enable the master to generate the STOP condition.

DATA TRANSFER

Figures 5, 6, and 7 detail how data transfer is accomplished on the 2–wire bus. Depending on the state of the R/$\overline{W}$ bit in the transmission protocols as shown in Figures 6 and 7, two types of data transfer are possible:

DATA TRANSFER ON 2–WIRE SERIAL BUS Figure 5



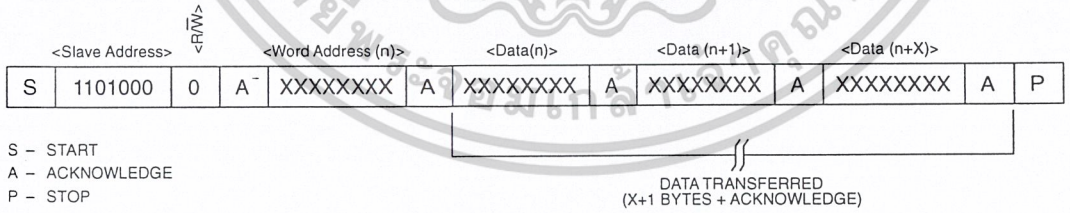
1. Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte. Data is transfered with the most significant bit (MSB) first.
2. Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. This is followed by the slave transmitting a number of data bytes. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a 'not acknowledge' is returned.

The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the bus will not be released. Data is transfered with the most significant bit (MSB) first.

The DS1307 may operate in the following two modes:

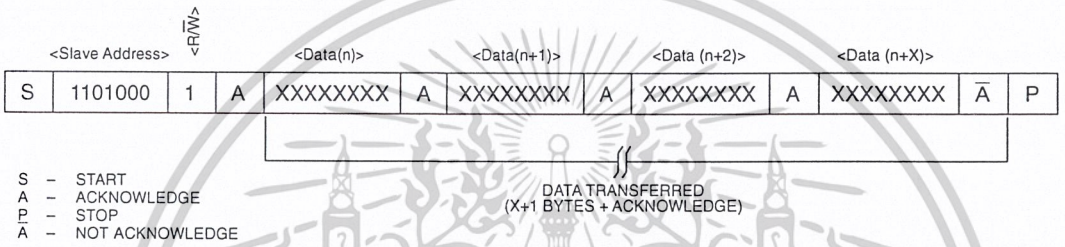
1. Slave receiver mode (DS1307 write mode): Serial data and clock are received through SDA and SCL. After each byte is received an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Address recognition is performed by hardware after reception of the slave address and direction bit (See Figure 6). The address byte is the first byte received after the start condition is generated by the master. The address byte contains the 7 bit DS1307 address, which is 1101000, followed by the direction bit (R/$\overline{W}$) which for a write is a 0. After receiving and decoding the address byte the DS1307 outputs an acknowledge on the SDA line. After the DS1307 acknowledges the slave address + write bit, the master transmits a register address to the DS1307 This will set the register pointer on the DS1307. The master will then begin transmitting each byte of data with the DS1307 acknowledging each byte received. The master will generate a stop condition to terminate the data write.

DATA WRITE – SLAVE RECEIVER MODE Figure 6



2. Slave transmitter mode (DS1307 read mode): The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will indicate that the transfer direction is reversed. Serial data is transmitted on SDA by the DS1307 while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer (See Figure 7). The address byte is the first byte received after the start condition is generated by the master. The address byte contains the 7 bit DS1307 address, which is 1101000, followed by the direction bit (R/W\) which for a read is a 1. After receiving and decoding the address byte the DS1307 inputs an acknowledge on the SDA line. The DS1307 then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode the first address that is read is the last one stored in the register pointer. The DS1307 must receive a Not Acknowledge to end a read.

DATA READ – SLAVE TRANSMITTER MODE Figure 7



S – START
A – ACKNOWLEDGE
P – STOP
Ā – NOT ACKNOWLEDGE

ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to Ground	–0.5V to +7.0V
Operating Temperature	0°C to 70°C
Storage Temperature	–55°C to +125°C
Soldering Temperature	260°C for 10 seconds

* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

The Dallas Semiconductor DS1307 is built to the highest quality standards and manufactured for long term reliability. All Dallas Semiconductor devices are made using the same quality materials and manufacturing methods. However, standard versions of the DS1307 are not exposed to environmental stresses, such as burn–in, that some industrial applications require. Products which have successfully passed through this series of environmental stresses are marked IND or N, denoting their extended operating temperature and reliability rating. For specific reliability information on this product, please contact the factory at (972) 371–4448.

RECOMMENDED DC OPERATING CONDITIONS

(0°C to 70°C)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{CC}	4.5	5.0	5.5	V	1
Logic 1	V_{IH}	2.2		V_{CC}+0.3	V	1
Logic 0	V_{IL}	–0.3		+0.8	V	1
V_{BAT} Battery Voltage	V_{BAT}	2.5		3.5	V	1

DC ELECTRICAL CHARACTERISTICS

(0°C to 70°C; V_{CC}=4.5V to 5.5V)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Input Leakage	I_{LI}			1	µA	10
I/O Leakage	I_{LO}			1	µA	11
Logic 0 Output	V_{OL}			0.4	V	2
Active Supply Current	I_{CCA}			1.5	mA	9
Standby Current	I_{CCS}			200	µA	3
Battery Current (OSC ON); SQW/OUT OFF	I_{BAT1}		300	500	nA	4
Battery Current (OSC ON); SQW/OUT ON (32 KHz)	I_{BAT2}		480	800	nA	4

SL486

INFRA RED REMOTE CONTROL PREAMPLIFIER

(Supersedes version in April 1994 Consumer IC Handbook, HB3120 - 2.0)

The SL486 is a high gain preamplifier designed to form an interface between an infra-red receiving diode and the digital input of remote control receiving circuits. The device contains two other circuit elements, one to provide a stretched output pulse facility and a voltage regulator to allow operation from a wide range of supplies.

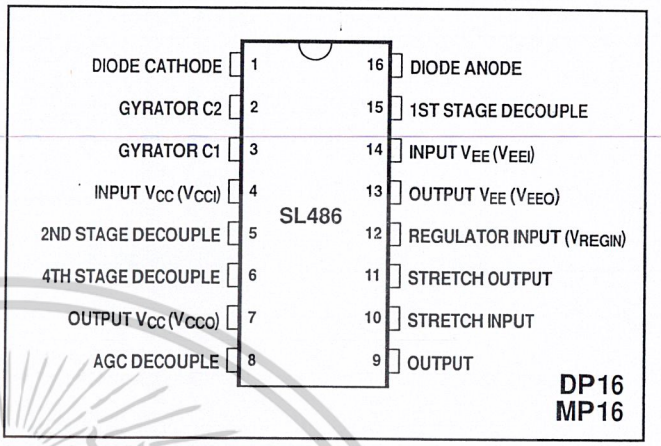


Fig. 1 Pin connections - top view

FEATURES

- Fast Acting AGC Improves Operation In Noisy Environments
- Differential Inputs Reduce Noise Pick-up and Improve Stability
- Gyrator Circuit Allows Operation in Environments with High Brightness Background Light Levels
- Output Pulse Stretcher for use with Microprocessor Decoders
- On-chip Regulator allows Operation from Wide Range of Power Supplies
- Low Noise Output

ORDERING INFORMATION

SL486 NA DP
SL486 NA MP

ABSOLUTE MAXIMUM RATINGS

Supply voltage, V_{CCI}	+10V wrt V_{EEI}
Supply voltage, V_{CCO}	+10V wrt V_{EEO}
Regulator input voltage, V_{REGIN}	−20V wrt V_{CCO}
Output current	5mA
Stretch output current	5mA
Operating temperature range	0°C to +70°C
Storage temperature	−55°C to +150°C

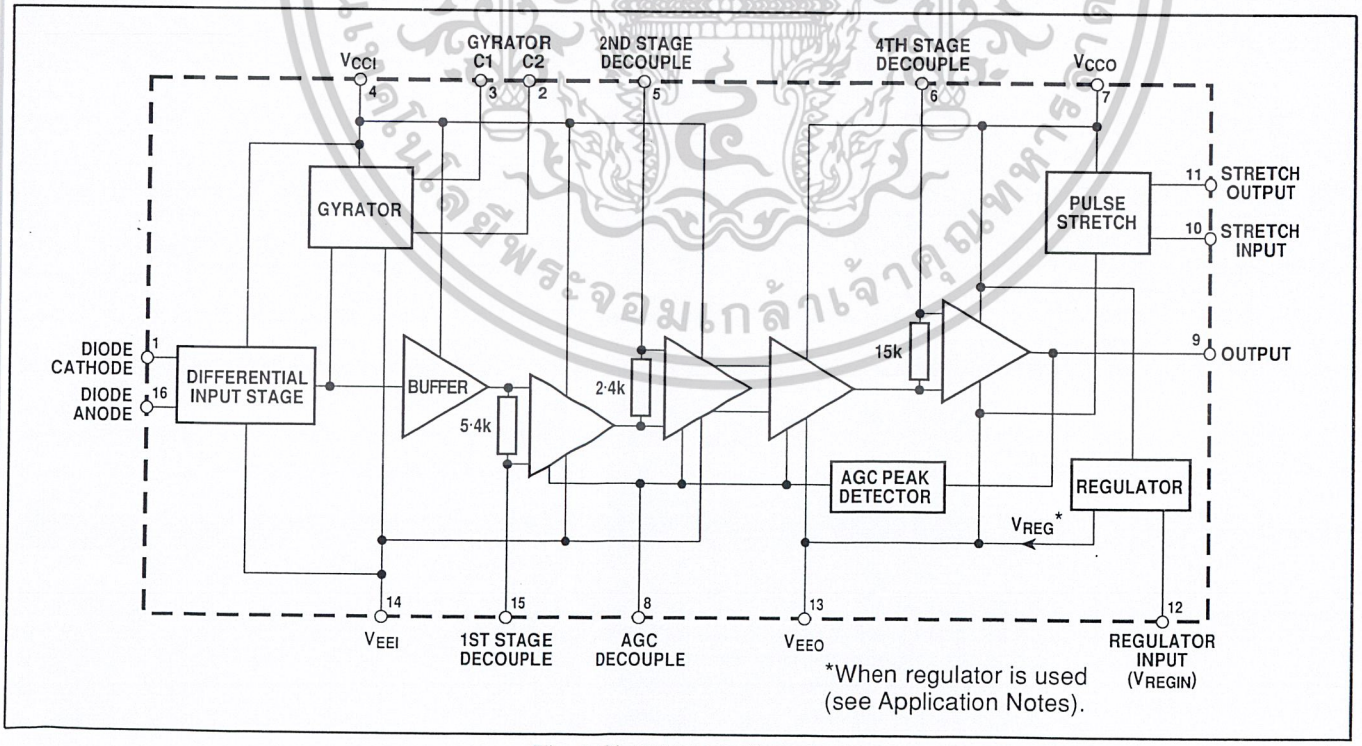


Fig. 2 SL486 block diagram

ELECTRICAL CHARACTERISTICS

These characteristics are guaranteed over the following conditions (unless otherwise stated):

T_{AMB} = +25°C, $V_{CCI} = V_{CCO} = V_{CC}$ = +4·5V to +7·0V, $V_{EEI} = V_{EEO} = V_{EE}$ = 0V

Characteristic	Pin	Value			Units	Conditions
		Min.	Typ.	Max.		
Supply current (see note 1)	4,7		6·5	9·0	mA	V_{CC} = 5·0V, I_D = 1·0µA
	4	$3·5+3I_D$	$4·2+3I_D$	$5+3I_D$	mA	V_{CC} = 4·5V, $I_D \leq 1·5$mA
	4,7		8·5	10	mA	V_{CC} = 18V, I_D = 1·0µA, V_{REGIN} = 0V
Low voltage supply wrt V_{EEI} & V_{EEO}	4,7	4·5		9·5	V	
High voltage supply wrt V_{REGIN}	4,7	8·4		18	V	$V_{EEI} = V_{EEO} = V_{REG}$ (see Figs. 4 & 6)
Int. regulated voltage, V_{REG}, wrt V_{CCO}	13	5·9	6·2	6·5	V	$V_{CCO}+V_{REGIN}$ = +16V
	4,7			1·5	V	
$\lvert V_{CCI}-V_{CCO}\rvert$				1·1	V	T_{AMB} = +70°C
Minimum sensitivity of differential input	1,16	9·0		2·3	nA	I_D = 1·0µA
		74·0		18·5	nA	I_D = 100µA
		168·0		42·0	nA	I_D = 0·5mA
Common mode rejection	1,16		35		dB	
Maximum signal input	1,16	3·0	4·0		mA (pk)	
AGC range			68·0		dB	
Output and Stretch output internal pull-up resistance	9, 11		55·0		kΩ	
Stretch output pulse width, t_P	11		2·4		ms	Capacitance pin 9 to pin 10 (C8 on Figs. 4 and 8) = 10nF; $t_P \approx -R_X C_8 \log_e\left[\frac{1·5}{V_{CC}}\right]$ ms where R_X = 200kΩ ±25% and R_X = internal resistance)
Temperature coefficient of R_X			0·7		%/°C	
Output low voltage	9			$V_{EEO}+0·35$	V	I_{SINK} = 0·2mA max.
Output high voltage	9	$V_{CCO}-0·5$			V	I_{SOURCE} = 5µA
Stretch output low voltage	11			$V_{EEO}+0·5$	V	I_{SINK} = 1·6mA max.
Stretch output high voltage	11	$V_{CCO}-0·1$			V	I_{SOURCE} = 5µA, output open circuit
V_{CCI} supply rejection	4		1·5		V (pk)	Ripple amplitude at 100Hz, V_{REGIN} = 0V
			0·8		V (pk)	Ripple amplitude at 100Hz, V_{EEO} and V_{EEI} = 0V

NOTE 1. I_D = IR diode forward current

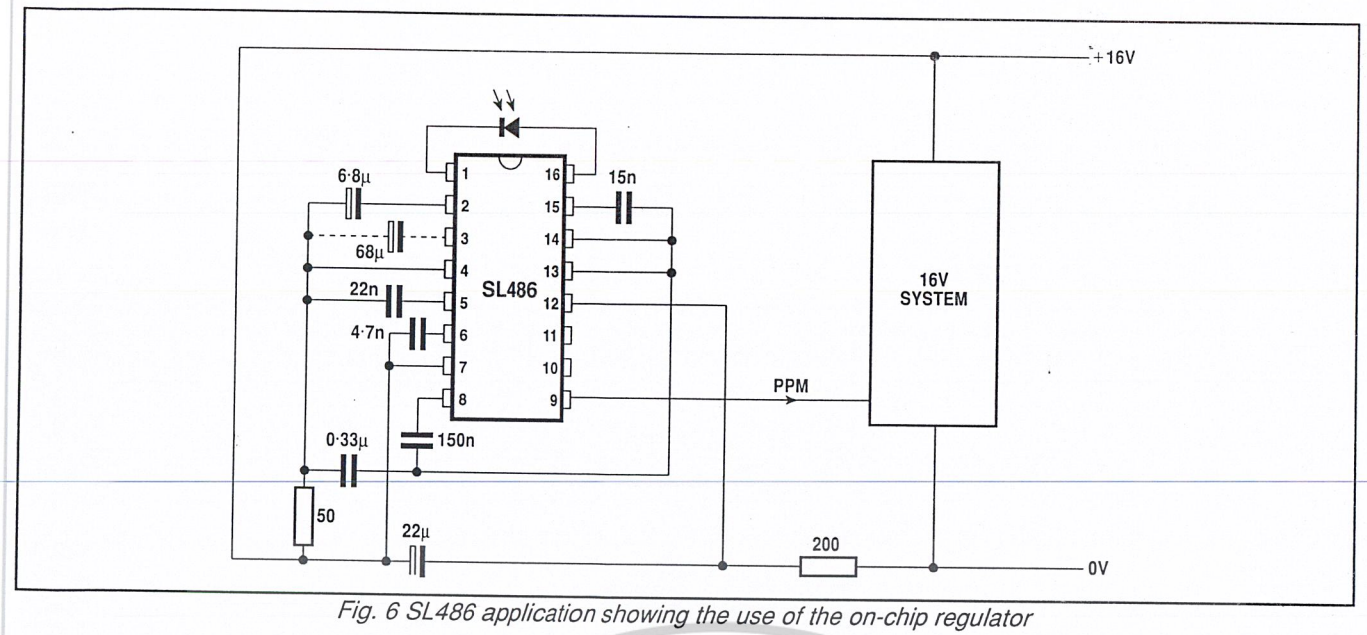


Fig. 6 SL486 application showing the use of the on-chip regulator

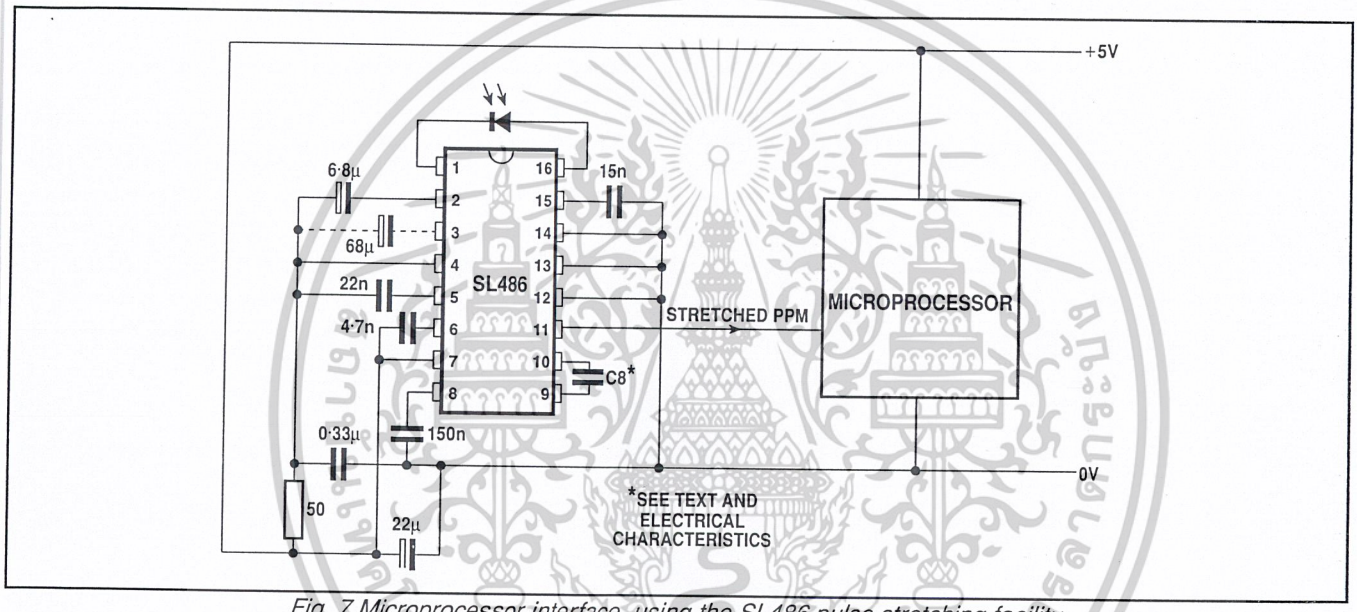


Fig. 7 Microprocessor interface, using the SL486 pulse stretching facility

FAIRCHILD SEMICONDUCTOR™

October 1987
Revised August 2000

CD4514BC• CD4515BC
4-Bit Latched/4-to-16 Line Decoders

General Description

The CD4514BC and CD4515BC are 4-to-16 line decoders with latched inputs implemented with complementary MOS (CMOS) circuits constructed with N- and P-channel enhancement mode transistors. These circuits are primarily used in decoding applications where low power dissipation and/or high noise immunity is required.

The CD4514BC (output active high option) presents a logical "1" at the selected output, whereas the CD4515BC presents a logical "0" at the selected output. The input latches are R–S type flip-flops, which hold the last input data presented prior to the strobe transition from "1" to "0". This input data is decoded and the corresponding output is activated. An output inhibit line is also available.

Features

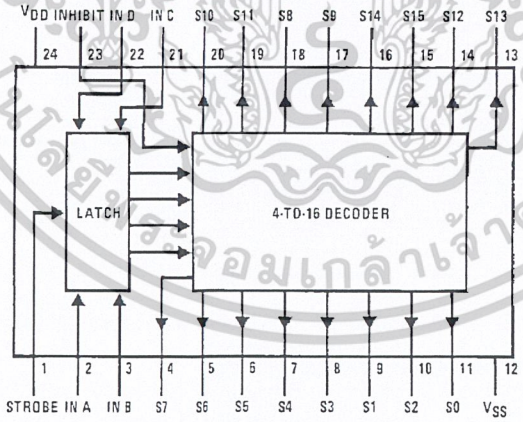
- Wide supply voltage range: 3.0V to 15V
- High noise immunity: 0.45 V_{DD} (typ.)
- Low power TTL: fan out of 2
  compatibility: driving 74L
- Low quiescent power dissipation:
  0.025 µW/package @ 5.0 V_{DC}
- Single supply operation
- Input impedance – $10^{12}\Omega$ typically
- Plug-in replacement for MC14514, MC14515

Ordering Code:

Order Number	Package Number	Package Diagram
CD4514BCWM	M24B	24-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300 Wide
CD4514BCN	N24A	24-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-011, 0.600 Wide
CD4515BCWM	M24B	24-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300 Wide
CD4515BCN	N24A	24-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-011, 0.600 Wide

Devices also available in Tape and Reel. Specify by appending suffix letter "X" to the ordering code.

Connection Diagram



Top View

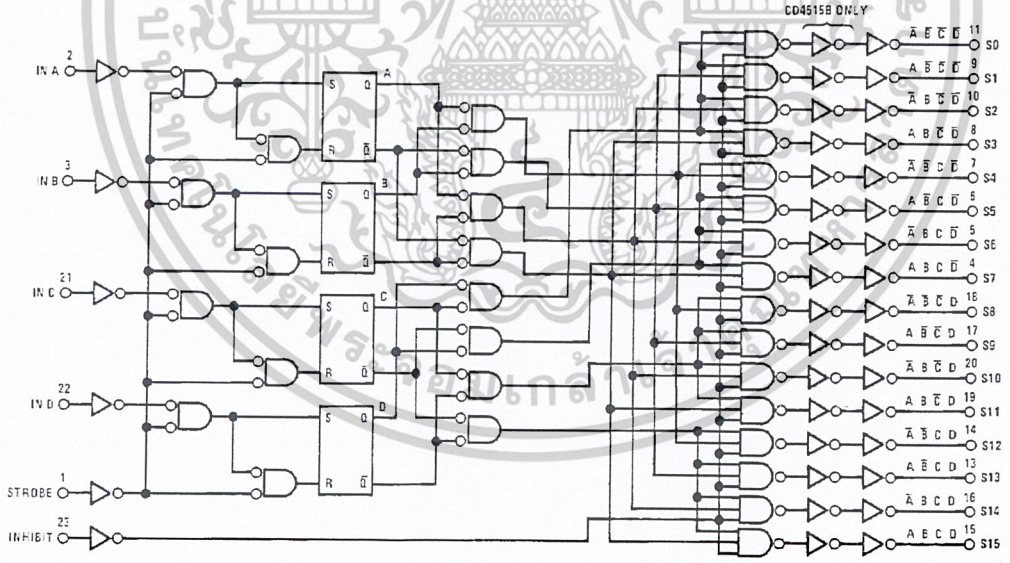
Truth Table

Decode Truth Table (Strobe = 1)

Inhibit	Data Inputs				Selected Output CD4514 = Logic "1" CD4515 = Logic "0"
	D	C	B	A	
0	0	0	0	0	S0
0	0	0	0	1	S1
0	0	0	1	0	S2
0	0	0	1	1	S3
0	0	1	0	0	S4
0	0	1	0	1	S5
0	0	1	1	0	S6
0	0	1	1	1	S7
0	1	0	0	0	S8
0	1	0	0	1	S9
0	1	0	1	0	S10
0	1	0	1	1	S11
0	1	1	0	0	S12
0	1	1	0	1	S13
0	1	1	1	0	S14
0	1	1	1	1	S15
1	X	X	X	X	All Outputs = 0, CD4514 All Outputs = 1, CD4515

X – Don't Care

Logic Diagram



หนังสืออ้างอิง

1. ผศ.พิพัฒน์ เลาหสงคราม, ไมโครคอนโทรลเลอร์ , พิมพ์ครั้งที่ 2 , พฤษภาคม 2537
2. ชัยวัฒน์ ลิ้มพรจิตรวิไล , วรพจน์ กรแก้ววัฒนกุล, เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช ฉบับ AT89C5X ของ Atmel , 2543
3. ปรเมษฐ์ ประนยานันทน์, ปิยพงศ์ เผ่าวณิช, คู่มือและการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS-51 , 2544
4. รีโมต เครื่องควบคุมไร้สาย , ซีเอ็ดยูเคชั่น , 2538
5. สนอง อิ่มเอม , เครื่องทำความเย็นและเครื่องปรับอากาศรถยนต์ , พิมพ์ครั้งที่ 7 , 21 ตุลาคม 2532
6. รศ.สมยศ จุณณะปิยะ, การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS-51 , พิมพ์ครั้งที่ 3 , 2543