วงจรควบคุมการสลับช่องสัญญาณจราจร

REVERSABLE LANES

โดย

นาย ไมตรี สงวนไชยกฤษณ์

นาย พิสิฐ บุญศรีเมือง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

วงจรควบคุมการสลับช่องสัญญาณจราจร

REVERSABLE LANES

โดย

นาย ไมตรี สงวนไชยกฤษณ์ 38013022

นาย พิสิฐ บุญศรีเมือง 38013065

อาจารย์ที่ปรึกษา

รศ.ดร. ถวิล พึ่งมา

ศ. มนูญ สุขเกษม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง วงจรควบคุมการสลับช่องจราจร

REVERSABLE LANES

ผู้จัดทำ นาย ไมตรี สงวนไชยกฤษณ์ 38013022

นาย พิสิฐ บุญศรีเมือง 38013065

.....

(รศ.ดร. ถวิล ทึ่งมา)

.....

(ศ. มนูญ สุขเกษม)

วงจรควบคุมการสลับช่องจราจร

REVERSABLE LANES

โดย นาย ไมตรี สงวนไชยกฤษณ์ 38013022
นาย พิสิฐ บุญศรีเมือง 38013065

อาจารย์ที่ปรึกษา รศ.ดร. ถวิล พึ่งมา
ศ. มนูญ สุขเกษม

บทคัดย่อ

วงจรควบคุมการสลับช่องจราจร (Reversable Lanes Controller) สามารถควบคุมการสลับช่องจราจรได้โดยการใช้สัญญาณไฟเป็นตัวบอกเส้นทางจราจรซึ่งสามารถควบคุมเส้นทางจราจรได้โดยการตั้งโปรแกรมตามเวลาจริงได้ 256 ระดับ หรือใช้สวิตช์ในการควบคุมการเปลี่ยนช่องจราจรก็ได้ โดยที่อุปกรณ์นี้จะสามารถควบคุมช่องทางจราจรได้ 8 ช่องทาง พร้อมทั้งมีสวิตช์ในการโปรแกรมหรือควบคุมสัญญาณไฟ มีหน้าปัทม์แสดงเวลาจริงในปัจจุบัน เพื่อจะได้ทราบเวลาจริงในขณะนั้นว่า เวลาใดควรจะควบคุมช่องจราจรอย่างไร สามารถต่อถึงกันได้ถึง 64 ชุด โดยกำหนดให้ มีเครื่อง Master เป็นตัวควบคุมเครื่อง Slave และมีหน้าปัทม์สัญญาณไฟ เพื่อแสดงสถานะของไฟสัญญานในเวลานั้นว่าเป็นเช่นใด และมีสัญญานเตือน พร้อมทั้งบอกตำแหน่งของไฟสัญญาน เมื่อไฟสัญญานช่องใดช่องหนึ่งเกิดชำรุดขึ้น ซึ่งหน้าที่การทำงานต่างๆ ของอุปกรณ์นี้ถูกควบคุมจาก Microcontroller

Abstract

The Reversable Lanes can control traffic line by indicated light traffic line with 256 levels real timeprograming or direct control for switching line exchange. It can control 8 lines, connect to 64 stages with master control to slave with lamp alarm for inoperative. It has LCD display for all function, and all operation is controlled by Microcontroller.

สารบัญ

สารบัญ รูปภาพ

บทที่ 1. บทนำ

1. บทนำ

เนื่องจากในสภาวะปัจจุบันปัญหาการจราจรมีปัญหามากอย่างยิ่งโดยเฉพาะในเมืองใหญ่ๆ ผู้คนส่วนใหญ่ จะเดินทางเข้ามาทำงานในเมืองหลวง ทำให้เกิดสภาพปัญหาการจราจรติดขัด การแก้ปัญหาในการจราจรโดยใช้วงจรอีเล็คทรอนิคส์ (Electronics) เข้ามาเป็นเครื่องมือช่วยในการแก้ปัญหาซึ่งจะทำหน้าที่ในการจัดการจราจร การจราจรที่หนาแน่นนี้ส่วนใหญ่จะอยู่ในเมืองใหญ่ๆ และโดยเฉพาะอย่างยิ่งถนนที่เข้าเมืองหลวงเพื่อเดินทางต่อไปยังจังหวัดต่างๆทั่วประเทศ ซึ่งความหนาแน่นการจราจรบนถนนในแต่ละสายจะไม่เท่ากันปัจจัยส่วนหนึ่งที่ทำให้ความหนาแน่นของการจราจรบนถนนต่างกันก็คือ " เวลา " ทางผู้จัดทำจึงใช้แนวคิดที่กล่าวมานี้เป็นแนวทางสร้างเครื่องควบคุมการเปลี่ยนช่องทางจราจร (REVERSABLE LANES) ซึ่งจะเป็นเครื่องควบคุมการเปลี่ยนช่องทางจราจรบนฐานเวลาจริง ให้การจราจรในแต่ละวัน และในแต่ละเวลาแปรเปลี่ยนต่างกัน การเปลี่ยนช่องทางการจราจรของถนนแต่ละสาย สามารถที่จะเปลี่ยนแปลงได้อย่างเหมาะสมนั้น สามารถที่จะเปลี่ยนแปลงได้สองวิธีด้วยกันคือ

1.1 การเปลี่ยนแปลงแบบอัตโนมัติ (AUTOMATIC) โดยการเปลี่ยนแปลงจะเปลี่ยนแปลงตามขั้นตอนที่กำหนดไว้ตามโปรแกรม (Program) โดยการเปลี่ยนแปลงตามโปรแกรมนี้จะเปลี่ยนแปลงตามฐานเวลาจริง ซึ่งในวงจรนี้จะมีวงจรนาฬิกาโดยวงจรนาฬิกานี้จะมีความผิดพลาดของเวลาน้อยมาก และสามารถที่จะเปลี่ยนแปลงช่องทางการจราจรได้แม้ว่าจะไม่มีผู้ควบคุม หรือตำรวจจราจรอยู่ในขณะนั้น

1.2 การควบคุมจากผู้ควบคุมได้โดยตรง (MANAUL) ซึ่งจะมีสวิตช์ (Switch) โดยสามารถที่จะควบคุมการเปลี่ยนได้โดยตรง ตามการเปลี่ยนแปลงที่นอกเหนือจากที่โปรแกรมเอาไว้ การเปลี่ยนแปลงนี้จะเปลี่ยนแปลงทันทีที่กดสวิตช์ จากตู้ควบคุมที่จะมีสวิตช์เท่ากับจำนวนดวงไฟที่แสดงในช่องทางการจราจร การเปลี่ยนช่องทางการจราจรนี้จะแสดงให้ผู้ขับรถสามารถเห็นได้ด้วยสัญลักษณ์ไฟ โดยให้ช่องสัญญาณที่สามารถที่จะเดินรถได้นั้นแทนสัญลักษณ์ด้วยสัญลักษณ์ลูกศรสีเขียว และช่องทางจราจรที่ไม่สามารถที่จะเดินรถได้นั้นจะแทนด้วยสัญลักษณ์กากบาทสีแดง ซึ่งสัญลักษณ์ที่แสดงให้เห็นนั้น จะใช้หลอดไฟฟ้าไฮดรอเจน (Hydrogen) นำมาเรียงต่อกันเป็นสัญลักษณ์ลูกศร และสัญลักษณ์กากบาทตามลำดับ การแสดงสัญลักษณ์ไฟนี้จะแสดงแบบเป็นจุดๆ ตามความยาวของถนนเรียงต่อกัน โดยการเรียงต่อกันนี้ ระยะห่างจะแตกต่างกันออกไปตามแต่ละสภาพของถนน จะอย่างไรก็ตามจะต้องให้ผู้ขับขี่รถสามารถที่จะสังเกตุเห็นได้อย่างสะดวก ในการควบคุมการเปลี่ยนช่องทางจราจรนี้ จะควบคุมการทำงานโดยใช้ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นตัวที่ทำหน้าที่ควบคุมการทำงานของเครื่องทั้งหมด เหตุผลที่เลือกใช้ไมโครคอนโทรลเลอร์ ในการควบคุมการทำงานนั้น เพราะว่ามีข้อดีหลายประการด้วยกัน คือทำให้ขนาดของวงจรมีขนาดเล็ก มีน้ำหนักเบา มีราคาถูก การทำงานมีความเชื่อมั่นได้สูง และยังมีความยืดหยุ่นในการทำงานสูงอีกด้วย การทำงานจะเป็นลำดับขั้นตอนของโปรแกรม ซึ่งเมื่อต้องการที่จะเปลี่ยนแปลงขั้นตอนการทำงาน หรือต้องการที่จะทำการพัฒนาก็สามารถที่จะทำได้สะดวก เนื่องจากสามารถที่จะเปลี่ยนแปลงได้ทันทีจากซอฟท์แวร์ (Software) ไม่ยุ่งยากเหมือนการทำงานประเภทวงจร ที่เมื่อต้องการที่จะเปลี่ยนแปลงอะไรนั้นจะต้องทำการเปลี่ยนแปลงทางฮาร์ดแวร์ (Hardware) ซึ่งจะเป็นการ

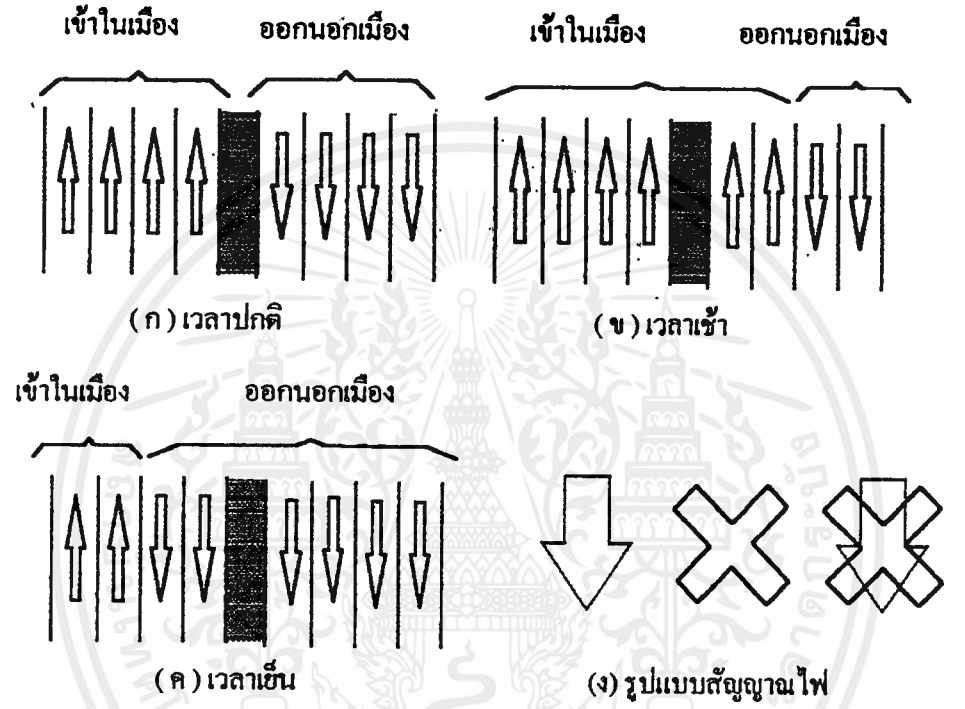
ประหยัดค่าใช้จ่ายเวลาที่ต้องการจะพัฒนาการทำงานของวงจรให้มีความสามารถเพิ่มมากขึ้น เครื่องควบคุมการเปลี่ยนช่องทางการจราจรนี้ สามารถที่ต่อเชื่อมการทำงานได้ถึงกัน โดยที่การแบ่งออกเป็นจุดๆ ซึ่งเวลาในแต่ละจุดจะต้องสัมพันธ์กัน คือเวลาในแต่ละจุดนั้น จะมีค่าเดียวกันโดยแต่ละจุดจะส่งข้อมูลเกี่ยวกับเวลา เพื่อที่จะใช้ในการอ้างอิงเวลา ซึ่งจะมีผลทำให้ทุกเครื่องมีเวลาเท่ากันทุกเครื่อง

นอกจากนี้ โครงงานนี้ยังสามารถที่จะพัฒนาจนถึงขั้นที่เป็นเนตเวอร์ก (Network) ขนาดใหญ่ที่สมบูรณ์แบบ คือสามารถที่จะต่อเชื่อมกับแยกทุกๆแยก และถนนสายต่างๆ ซึ่งจะทำให้การแก้ปัญหาการจราจรมีประสิทธิภาพมากยิ่งขึ้น ในการที่จะสามารถทำได้ขนาดนั้นต้องมีขั้นตอนการเขียนโปรแกรมที่สลับซับซ้อนมากยิ่งขึ้น ทางผู้จัดทำหวังว่าในอนาคตประเทศไทยจะสามารถทำได้เช่นนั้น

บทที่ 2. ทฤษฎี และหลักการ

2. ทฤษฎี และหลักการ

จากหลักการในการจัดช่องการจราจรให้มีประสิทธิภาพ เพื่อให้การจราจรมีความคล่องตัว โดยเฉพาะบนถนนสายหลักๆ ที่ใช้ในการจราจรที่เชื่อมต่อกับตัวเมืองใหญ่ๆ นั้น สามารถจัดช่องจราจรได้ตามตัวอย่างดังรูป



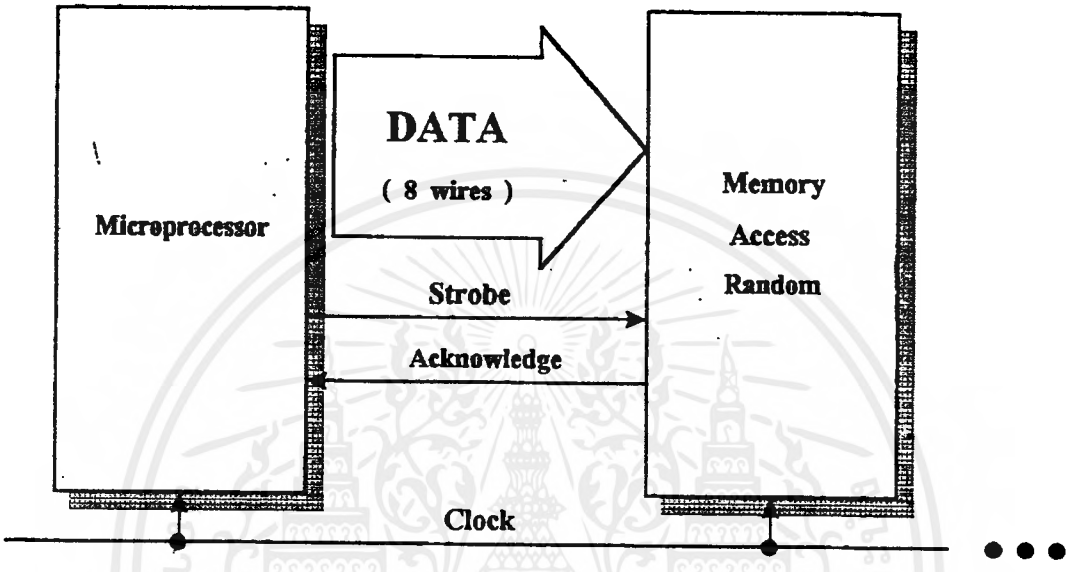
รูปที่ 2.1 แสดงการจัดช่องการจราจรตามหลักการของ Reversable Lanes

หลักการที่สำคัญของโครงงานนี้ ซึ่งเราแยกออกเป็น 2 ส่วนที่สำคัญ คือส่วนของซอฟท์แวร์ ซึ่งเป็นโปรโตคอล (Protocol) ในการติดต่อกันระหว่างสถานีควบคุมแต่ละแห่ง ในโครงงานนี้เรามีมาตรฐาน เค โอ เค โปรโตรคอล (K O K Protocol) แต่เราจะไม่ขอกล่าวถึงซอฟท์แวร์ ที่ใช้เขียนควบคุม ซึ่งทางผู้จัดทำใช้ภาษาแอสเซมบลี (Assembly) และอีกส่วนเป็นฮาร์ดแวร์ โดยที่ทั้งสองส่วนจะต้องทำงานสัมพันธ์กันซึ่งมีรายละเอียดดังนี้

รูปแบบของการสื่อสารข้อมูล และ เค ไอ เค โปรโตคอล (K O K Protocol)

2.1 หลักการ และโครงสร้างการสื่อสารข้อมูลแบบขนาน

หลักการ และโครงสร้างการสื่อสารข้อมูลแบบขนานเป็นการสื่อสารข้อมูล โดยการรับ-ส่งผ่านสายสัญญาณ หรือช่องสัญญาณพร้อมกันหลายๆเส้น ดังรูป



รูปที่ 2.2 แสดงโครงสร้างของการสื่อสารข้อมูลแบบขนาน

ซึ่งจำนวนของสัญญาณจะมีจำนวนที่ไม่แน่นอน ขึ้นอยู่กับโครงสร้างของการประมวลผลของระบบนั้นๆ โดยข้อดีของการสื่อสารแบบนี้ คือสามารถที่จะสื่อสารข้อมูลได้อย่างรวดเร็ว ในเวลาอันสั้นๆ แต่ข้อเสียของวิธีนี้ ก็คือจะสิ้นเปลืองสายสัญญาณเป็นจำนวนมาก และถ้ายิ่งเป็นการสื่อสารในระยะทางไกลๆ นอกจากจะสิ้นเปลืองค่าใช้จ่ายจำนวนมาก และยังเกิดการลดทอนของสัญญาณอีกด้วย ดังนั้นการสื่อสารข้อมูลแบบขนานนี้นั้นจะนิยมนำมาใช้ในการสื่อสารข้อมูลในระยะสั้นๆ ที่ต้องการสื่อสารข้อมูลด้วยอัตราการส่งข้อมูลที่สูง เช่น การสื่อสารข้อมูลเชื่อมต่อของสัญญาณระหว่างหน่วยประมวลผลกลางของคอมพิวเตอร์กับอุปกรณ์รอบข้าง

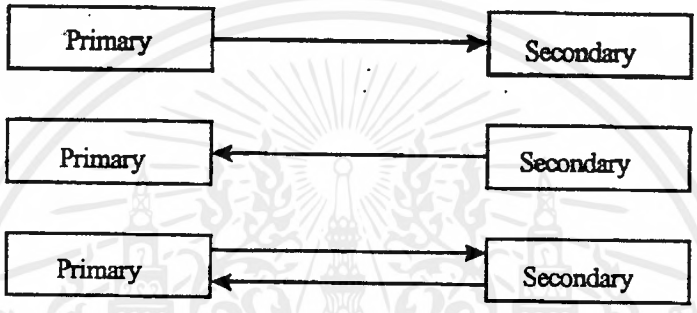
2.2 หลักการ และโครงสร้างการสื่อสารข้อมูลแบบอนุกรม

หลักการ และโครงสร้างการสื่อสารข้อมูลแบบอนุกรมที่การรับ และการส่งข้อมูลเป็นการใช้สายสัญญาณในการส่งข้อมูลจำนวนน้อย ซึ่งโดยปกติจะใช้เพียง 1 คู่ คือสายสัญญาณที่เป็นข้อมูล และสายกราวด์ (Ground) เปรียบเทียบกัน โดยข้อมูลที่ส่งออกหรือรับเข้าจะเป็นลักษณะที่เป็นบิท (Bit) ต่อบิท ซึ่งเมื่อเราทำการเปรียบเทียบกับการสื่อสารข้อมูลแบบขนานที่จำนวนข้อมูล และอัตราการส่งข้อมูลที่เท่ากันแล้ว จะพบว่าการสื่อสารข้อมูลแบบอนุกรมนี้จะต้องใช้เวลาในการรับ-ส่งข้อมูลมากกว่าแน่นอน แต่ข้อดีของการสื่อสารข้อมูลแบบอนุกรม คือการใช้สายสัญญาณน้อยกว่า และสามารถที่จะส่งข้อมูลได้ไกลกว่า แม้ว่าอัตราการลดทอน และ

การเพี้ยนของสัญญาณที่มีผลจากความยาวของสายสัญญาณจะมีค่าเท่ากับการสื่อสารข้อมูลแบบขนาน แต่การสื่อสารข้อมูลแบบอนุกรมมีวิธีที่ใช้ในการที่จะลดผลของการลดทอนของสัญญาณนี้ โดยการรับ-ส่งสัญญาณแบบดิฟเฟอร์เรนเชียล (Differential) ดังนั้นการสื่อสารข้อมูลแบบอนุกรมจึงเหมาะสมกับการสื่อสารข้อมูลในระยะทางไกลๆ การสื่อสารข้อมูลที่ต้องใช้จำนวนสาย หรือช่องสัญญาณในการรับ-ส่งข้อมูลจำนวนน้อย เช่น การสื่อสารข้อมูลท้องถิ่น (Local Area Netwoek : LAN)

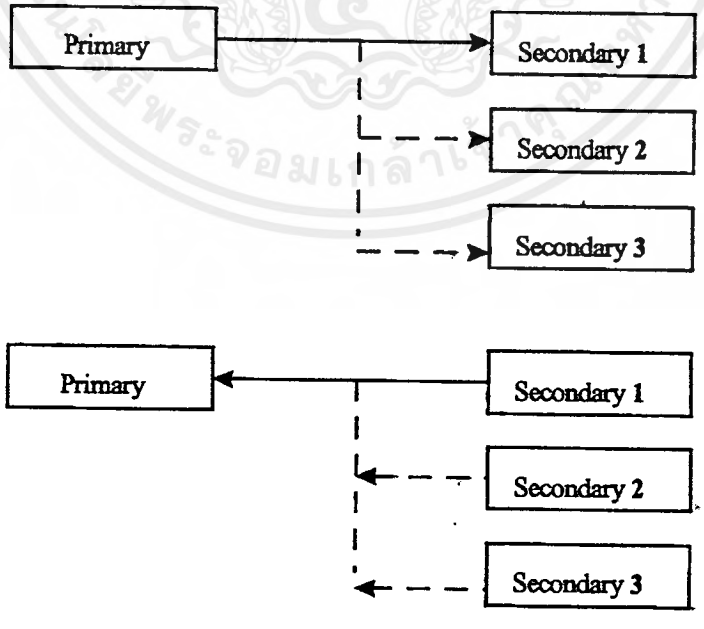
จากที่ได้กล่าวมาแล้ว การสื่อสารข้อมูลแบบอนุกรม ยังสามารถที่จะแบ่งตามหลักลักษณะของทิศทางในการสื่อสารข้อมูล ตามโครงสร้าง และความต้องการของระบบ ได้ดังนี้คือ

1.การสื่อสารข้อมูลในทิศทางเดียวตลอดเวลา : ซิมเพล็ก (Simplex)



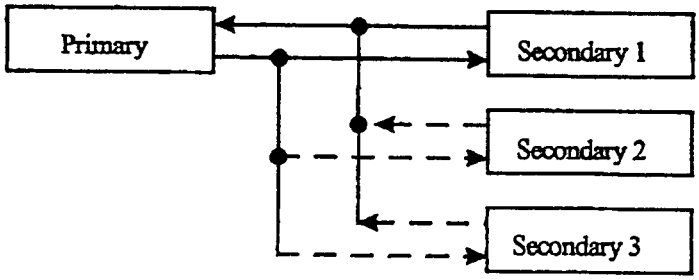
รูปที่ 2.3 แสดงโครงสร้างการสื่อสารข้อมูลแบบซิมเพล็ก

2.การสื่อสารข้อมูลใน 2 ทิศทาง แต่สลับเวลากัน : ฮาล์ฟ-ดูเพล็ก (Half-Duplex)



รูปที่ 2.4 แสดงโครงสร้างการสื่อสารข้อมูลแบบฮาล์ฟ-ดูเพล็ก

3.การสื่อสารข้อมูลใน 2 ทิศทางตลอดเวลา : ฟูลล์-ดูเพล็ก (Full-Duplex)



รูปที่ 2.5 แสดงโครงสร้างการสื่อสารข้อมูลแบบฟูลล์-ดูเพล็ก

ซิมเพล็ก เป็นการสื่อสารข้อมูลในทิศทางใดก็จะใช้ทิศทางนั้นตลอดไป ไม่มีการเปลี่ยนแปลงทิศทาง เช่น การส่งกระจายเสียงของสถานีวิทยุไปยังเครื่องรับ หรือการส่งข้อมูลไปยังวิทยุติดตามตัว

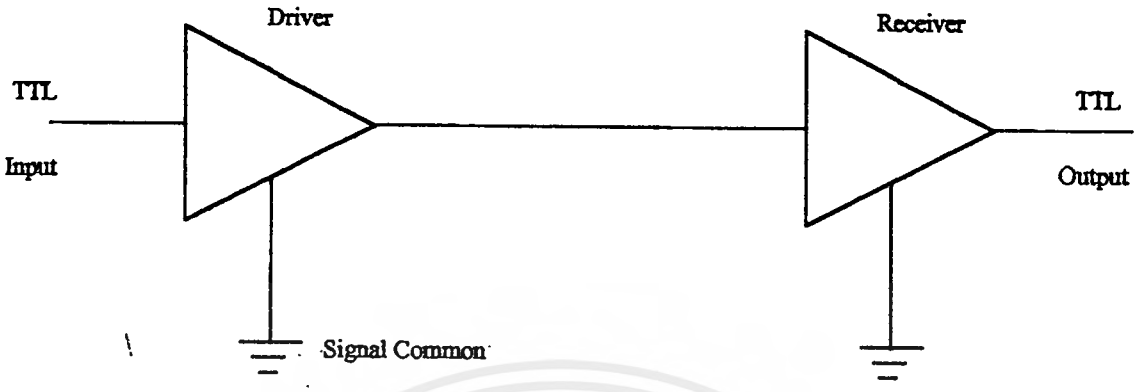
ฮาล์ฟ-ดูเพล็ก เป็นการสื่อสารข้อมูลใน 2 ทิศทาง แต่ขณะเวลาหนึ่งนั้น สัญญาณจะไปได้ในทิศทางเดียวเท่านั้น ดังนั้นอุปกรณ์แต่ละตัวที่จะเชื่อมต่อ หรือการสื่อสารข้อมูลในลักษณะนี้ จะต้องได้ทั้งตัวรับและตัวส่ง ซึ่งมีชื่อว่า ทรานส์ซิฟเวอร์ (Transceiver) และจะต้องมีวงจรที่เลือก ณ. เวลานั้นจะทำงานเป็นตัวรับ หรือตัวส่ง

ฟูลล์-ดูเพล็ก เป็นการสื่อสารข้อมูลที่คล้ายกับการสื่อสารแบบฮาล์ฟ-ดูเพล็ก เพียงแต่ในขณะเวลาเดียวกัน สามารถที่จะทำหน้าที่ได้ทั้งรับ และส่งในเวลาเดียวกันได้

2.2.1 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232C

การสื่อสารข้อมูลแบบอนุกรมที่ใช้งานอยู่ในปัจจุบันนี้ ได้มีการกำหนดมาตรฐานการรับ-ส่งข้อมูลไว้หลายแบบด้วยกัน แต่ที่ได้รับความนิยมนำไปใช้กันอย่างแพร่หลาย คือการสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232C เหตุที่เป็นที่นิยม เนื่องจากเป็นการสื่อสารข้อมูลที่ใช้ในเครื่องคอมพิวเตอร์ (Computer) IBM-PC ซึ่งเป็นเครื่องคอมพิวเตอร์ที่ใช้อย่างแพร่หลายจากอดีตมาถึงปัจจุบัน

มาตรฐาน RS-232C โครงสร้างการสื่อสารข้อมูลเป็นแบบจุดต่อจุดเท่านั้น โดยมีลักษณะสมบัติทางไฟฟ้า และทางกายภาพ แสดงได้ดังรูป



รูปที่ 2.6 แสดงโครงสร้างลักษณะคุณสมบัติทางไฟฟ้าแบบไม่สมดุลย์

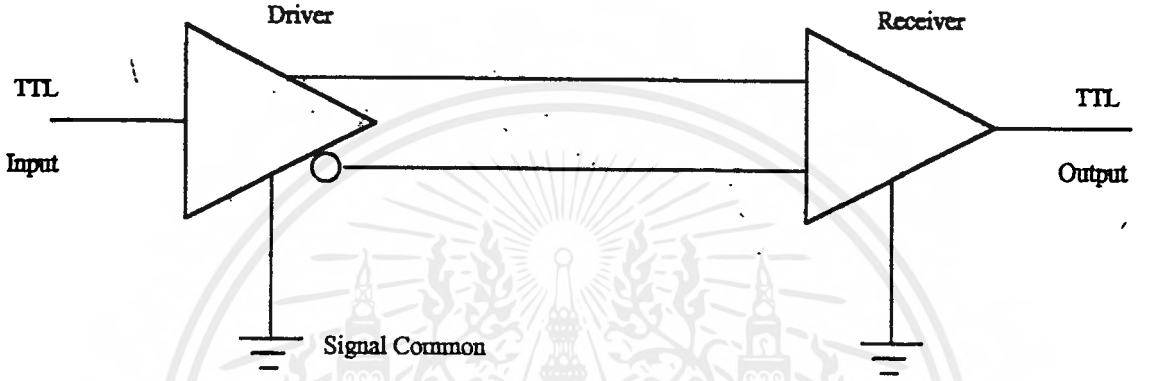
ข้อจำกัดของการสื่อสารข้อมูลแบบอนุกรมมาตรฐาน RS-232C

ข้อจำกัดนี้ประกอบกันด้วย 3 อย่างด้วยกัน คือ

- ระยะทางในเคเบิล (Cable) RS-232C ค่าความจุไฟฟ้าระหว่างสายสัญญาณต่างๆ กับกราวด์ เป็นพารามิเตอร์หลักที่จำกัดระยะทางการใช้เคเบิลนั้น ตามข้อกำหนดมีว่า ค่าความจุไฟฟ้าที่มองไปจากวงจรขับ (Driver) จะต้องไม่เกิน 2500 pF การเพิ่มระยะทางของเคเบิล ก็จะเพิ่มค่าความจุไฟฟ้าที่มาเป็นภาระของวงจรขับธรรมดาที่ใช้เคเบิลที่ใช้อยู่จะมีความจุไฟฟ้า 2500 pF ที่ความยาวประมาณ 15 เมตร
- อัตราการส่งข้อมูล จะต้องไม่เกิน 20,000 บิท/วินาที ปัญหานี้เกี่ยวข้องกับความต้านทาน กับค่าความจุของสายเคเบิลเช่นกัน RS-232C นั้น กำหนดค่าความต้านทานอินพุท (Input) ของวงจรภาครับค่อนข้างสูง 3000 - 7000 Ω
- สัญญาณรบกวน สามารถทำให้เกิดข้อมูลผิดพลาด ซึ่งเกิดจากแหล่งกำเนิดภายนอก เช่น มอเตอร์ไฟฟ้า สถานีวิทยุ และจากแหล่งกำเนิดภายใน เช่น ความต้านทานไม่สมบูรณ์ การกระจายของกระแสจากสายสัญญาณต่างๆ

2.2.2 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-422A

ในการออกแบบการสื่อสารข้อมูลที่กล่าวมา ได้มีการพยายามที่จะออกแบบให้การสื่อสารข้อมูลได้รวดเร็วยิ่งขึ้น และมีระยะทางในการสื่อสารข้องมูลได้มากขึ้นด้วย ซึ่งที่ผ่านมา การสื่อสารข้อมูลมาตรฐาน RS-232C ได้ออกแบบมา เพื่อใช้เชื่อมโยง กับโมเด็ม (Modem) เท่านั้น จึงไม่ได้คำนึงถึงความเร็ว และระยะทางในการสื่อสาร แต่ในปัจจุบันได้มีการออกแบบมา เพื่อรองรับความต้องการของการใช้งานที่ต้องการให้รับ-ส่งข้อมูลได้ไกลยิ่งขึ้น คือมาตรฐาน RS-422A ซึ่งจะใช้สัญญาณแบบดิฟเฟอร์เรนเชียลแสดงดังรูป



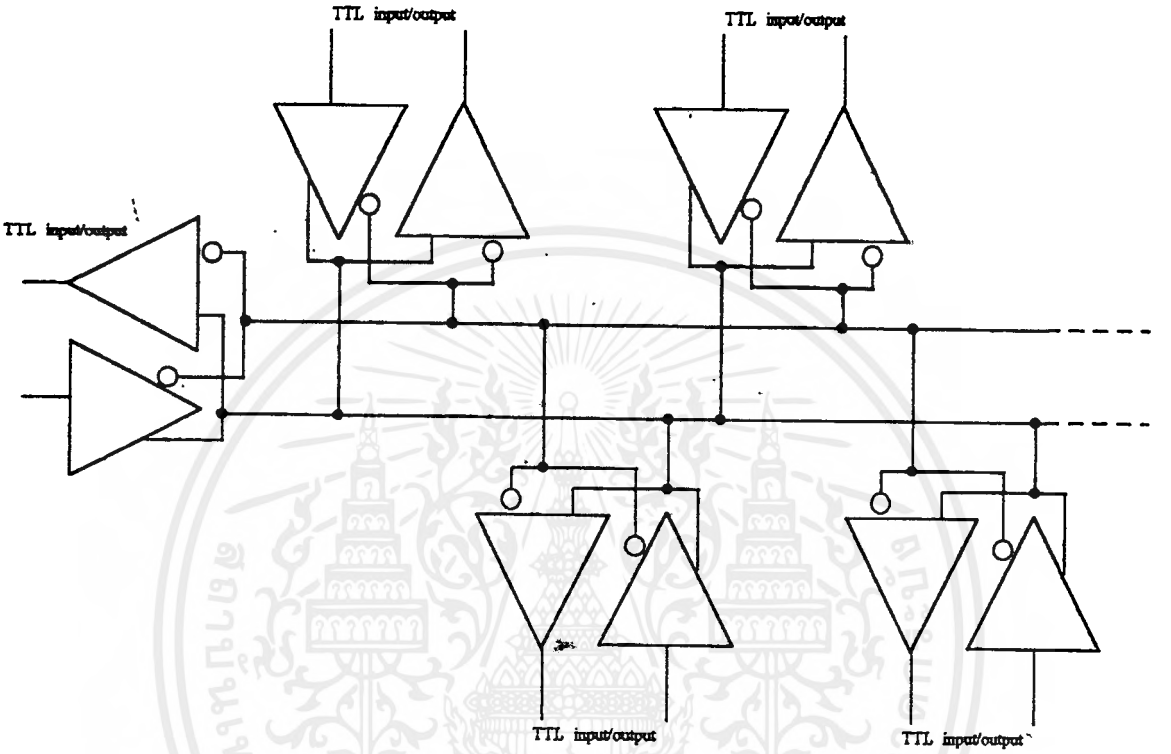
รูปที่ 2.7 แสดงโครงสร้างลักษณะคุณสมบัติทางไฟฟ้าแบบสมดุลย์

หลักการก็คือ สัญญาณที่จะรับ-ส่งจะเป็นการเปรียบเทียบระหว่างสัญญาณ 2 เส้นเปรียบเทียบ กับมาตรฐาน RS-232C ที่สัญญาณทุกสัญญาณจะเทียบกราวด์ ซึ่งในการสื่อสารในระยะทางไกลๆ แล้วสัญญาณจะถูกลดทอนไปถึงจุดๆหนึ่งสัญญาณนั้นก็จะผิดพลาดไปจากความเป็นจริง ก็จะทำให้การรับ-ส่งผิดพลาดขึ้น แต่สำหรับสัญญาณดิฟเฟอร์เรนเชียลแล้ว การลดทอนของสัญญาณจะไปลดทอนทั้งสองสายด้วยค่าที่เท่ากัน หรือใกล้เคียงกัน และความแตกต่างของสัญญาณทั้งสองเส้น จากตัวส่งไปยังตัวรับก็ยังคงมีค่าเท่าเดิม หรือแตกต่างกันเล็กน้อย จึงทำให้ผลการลดลงของสัญญาณที่ระยะทางการสื่อสารไกล มีผลต่อสัญญาณดิฟเฟอร์เรนเชียล มีค่าน้อยกว่าการสื่อสารข้อมูลแบบนี้ จึงสามารถส่งข้อมูลได้ไกลกว่า และอัตราการสื่อสารข้อมูลสูงกว่า

2.2.3 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-485

การสื่อสารข้อมูลตามมาตรฐานที่กล่าวมาข้างต้นนั้น เป็นมาตรฐานการสื่อสารข้อมูลในแบบที่ใช้กันระหว่างอุปกรณ์ หรือจุดต่อจุด (Point-to-Point) ส่วน RS-422A นั้นเป็นมาตรฐานที่พัฒนามาจาก RS-232C ให้ได้ระยะทางไกลขึ้น และอัตราการสื่อสารมากขึ้น แต่ยังเป็นการสื่อสารข้อมูลระหว่างอุปกรณ์หนึ่งไปยังอุปกรณ์อื่นๆ ได้อัตราสูงสุด 10 ตัว ไม่สามารถส่งย้อนกลับจากอุปกรณ์ 10 ตัวได้ หรือกล่าวได้ว่าอุปกรณ์การสื่อสารข้อมูลตามมาตรฐาน RS-422A นั้น เป็นการสื่อสารข้อมูลแบบซิมเพล็ก คือทิศทางการสื่อสารข้อมูลเป็นทางเดียวตลอดเวลา ดังนั้น ถ้าต้องการออกแบบระบบให้เป็นโครงข่ายข้อมูลก็ไม่สามารถที่จะออกแบบได้ จึงได้มีการพัฒนามาตรฐานการสื่อสารข้อมูลแบบใหม่ เพื่อรองรับการต้องการนี้ คือมาตรฐาน RS-485 ซึ่งเป็นมาตรฐานที่

อาศัยหลักการของสัญญาณแบบดิฟเฟอร์เรนเชียล เช่นเดียวกับมาตรฐาน RS-422A จากผลการทดลองการใช้สัญญาณในลักษณะดิฟเฟอร์เรนเชียล นี้จะทำให้ระยะทาง และความเร็วในการสื่อสารข้อมูล เช่นเดียวกับมาตรฐาน RS-422A แต่มาตรฐาน RS-485 สามารถที่จะสื่อสารกันระหว่างอุปกรณ์ ทั้งการรับ และการส่งได้หลายจุดที่เรียกว่า มัลติพ้อยท์คอมมิวนิเคชั่น (Multipoint Communication) โครงสร้างในการสื่อสารข้อมูลแบบ RS-485 ดังแสดงดังรูป



รูปที่ 2.8 แสดงโครงสร้างการสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-485

ตารางเปรียบมาตรฐานการสื่อสารข้อมูลของ EIA

พารามิเตอร์ (Parameter)	RS-232C	RS-423-A	RS-422A	RS-485
โหมดการทำงาน	Single-ended	Single-ended	Differential	Differential
ความยาวของคู่สาย และตัวส่งที่รับได้	1 ตัวส่ง 1 ตัวรับ.	1 ตัวส่ง 10 ตัวรับ	1 ตัวส่ง 10 ตัวรับ	32 ตัวส่ง 32 ตัวรับ
ความยาวของคู่สาย สัญญาณรับ-ส่งข้อมูล	50 ฟุต	4,000 ฟุต	4,000 ฟุต	4,000 ฟุต
อัตราการส่งข้อมูล	20 K	100 K	10 M	10 M

สูงสุด (บิทต่อวินาที)				
แรงดันไฟฟ้าโหมดร่วมสูงสุด	±2.5 V	±6 V	6 V -2.5 V	12 V -7 V
Driver Load	±5 V ต่ำสุด ±15 V สูงสุด	±3.6 V ต่ำสุด ±6.0 V สูงสุด	±2 V ต่ำสุด	±1.5 V ต่ำสุด
Driver Load Ω	3 K ถึง 7 K	450 ต่ำสุด	100 ต่ำสุด	60 ต่ำสุด
Drive Slew Rate	3 K/uS สูงสุด		NA	NA
กระแสลิมิต เมื่อกระแสลัดวงจร	500 mA ลัดวงจรกับ VCC หรือ GND	150 mA ลัดวงจรกลับกับ GND	150 mA ลัดวงจรกลับกับ GND	150 mA ลัดวงจรกลับกับ GND , 150 mA ลัดวงจรกลับกับ 8 V กับ 12 V
ความต้านทานเอาท์พุท (Output) ของตัวส่ง Ω	NA - Power On 300 - Power Off	NA – Power On 60 K – Power Off	NA – Power On 60 K – Power Off	120 K Power On , Off
ความต้านทานอินพุท ของตัวรับ Ω	3 K ถึง 7 K	4 K	4 K	12 K
ความไวของตัวรับ	±3 V	±200 mA	±200 mA	±200 mA

2.3 คุณสมบัติของการสื่อสารข้อมูลแบบ RS-485

คุณสมบัติของการสื่อสารข้อมูลแบบ RS-485 ที่แตกต่างจาก RS-422A

คุณลักษณะเฉพาะของตัวส่ง RS485

- ตัวส่ง 1 ตัว สามารถที่จะขับโหลด (Load) ได้สูงสุด 32 ตัว โดยที่โหลด 1 ชุด ประกอบด้วยตัวส่ง 1 ตัว และตัวรับหนึ่งตัว และค่าของความต้านทานที่ตกคร่อมระหว่างคู่สายสัญญาณมีค่า 60 Ω
- เอาท์พุท ของตัวส่งในสภาวะออฟ (Off) มีกระแสรั่วไหลไม่เกิน 100 uA ในช่วงแรงดันไฟฟ้าโหมดร่วมระหว่าง - 7 V ถึง + 12 V
- เอาท์พุท ของตัวส่งให้แรงดันไฟฟ้าเอาท์พุท 1.5 V ถึง 5 V ในช่วงแรงดันไฟฟ้าร่วมระหว่าง - 7 V ถึง + 12 V
- ตัวส่งมีวงจรป้องกันตัวเองที่ส่วนเอาท์พุท ในกรณีที่ตัวส่งหลายๆตัว ส่งข้อมูลออกมาพร้อมๆกัน
- คุณสมบัติของตัวรับ RS-485

- ค่าความต้านทานอินพุท มีค่าสูง โดยมีค่าไม่น้อยกว่า 12 กิโลโอห์ม (Kilo-Ohm)
- ตัวรับมีค่าแรงดันไฟฟ้าอินพุทโหมดร่วมระหว่างค่า – 7 V ถึง + 12 V
- ตัวรับสามารถตอบสนองต่อสัญญาณที่แตกต่างกันจากสัญญาณที่โหมดร่วมได้ ± 200 mA (น้อยที่สุด)

คุณสมบัติของคู่สายสัญญาณ RS-485

- คู่สายสัญญาณรับ-ส่งควรจะพันเป็นเกลียว เพื่อลดสัญญาณรบกวน

ความหมายของยูนิตโหลด (Unit Load)

เป็นจำนวนที่มากที่สุดของตัวรับ และตัวส่ง ที่สามารถที่จะใช้รับ-ส่งสัญญาณคู่หนึ่ง โดยจะขึ้นกับค่ายูนิตโหลด ซึ่ง RS-485 ยอมรับได้ที่ 32 ยูนิตโหลดต่อสายสัญญาณหนึ่งคู่สาย

นิยาม เป็นโหลดที่ใช้กระแส 1 มิลลิแอมป์ (milli-Amp) ที่แรงดันไฟฟ้าโหมดร่วมระหว่าง 12 V ซึ่งโหลดนี้ ประกอบด้วย ตัวส่ง และตัวรับ แต่ไม่รวมค่าความต้านทานที่ตกคร่อมคู่สายสัญญาณรับ-ส่ง

คุณสมบัติของคู่ตัวรับ-ส่ง

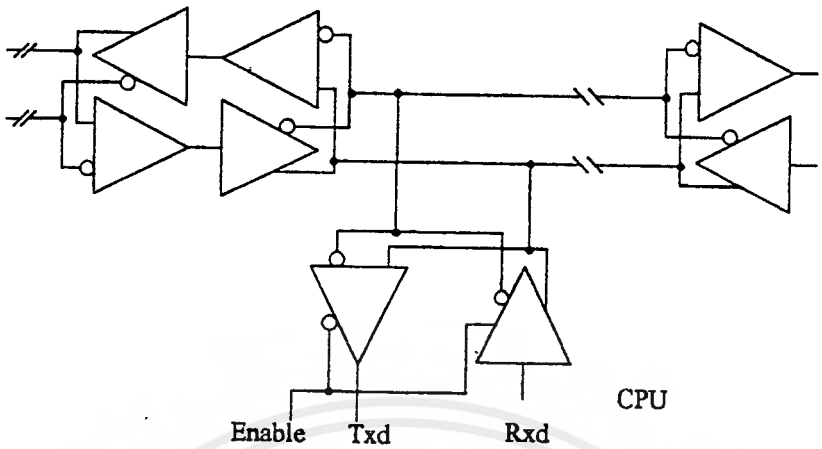
คู่ตัวรับ-ส่ง เป็นอุปกรณ์ที่มีทั้งตัวรับ และตัวส่งในอุปกรณ์ตัวเดียวกัน เพื่อให้สะดวกในการใช้งาน และทำให้ระบบมีขนาดเล็ก ในที่นี้จะใช้ไอซี (IC) DS75176

คุณสมบัติเฉพาะของคู่ตัวรับ-ส่ง

ตามมาตรฐาน RS-485 , RS-422A , CCITT , v.11 และ X.27

- เอาท์พุท ของตัวส่งเป็นแบบ 3 - State
- เอาท์พุท ตัวส่งสามารถขับกระแสได้สูง 60 mA
- Thermal Shutdown Protection
- คู่ตัวรับมีค่า Input Sensitivity ± 200 mA
- คู่ตัวรับมีค่า In Hysteresis 50 mA
- ใช้ไฟเลี้ยง 5 V

2.4 การประยุกต์ใช้งานแบบพื้นฐานของคู่ตัวรับ-ส่ง DS 75176



รูปที่ 2.9 แสดงการต่อการรับ-ส่งข้อมูล DS75176 กับ CPU และการทวนสัญญาณ

จากรูปแต่ละเทอร์มินอล (Terminal) อาจจะเป็นตัวส่ง หรือตัวรับข้อมูลก็ได้ โดยจะมีเทอร์มินอลอยู่หนึ่งชุดเป็นตัวควบคุมระบบการสื่อสารข้อมูลทั้งหมด และกำหนดรูปแบบของโปรโตคอล ซึ่งเทอร์มินอล ดังกล่าว จะอยู่บนเมนบอร์ด (Mainboard) ของตัวมาสเตอร์ (Master) และจะมีเมนบอร์ดของตัวสเลฟ (Slave) ซึ่งจะรับข้อมูลจากตัวมาสเตอร์ และยังทำหน้าที่ทวนสัญญาณ เพื่อส่งต่อให้ไปยังตัวสเลฟตัวต่อไป หากไม่ใช่ข้อมูลของตนเอง จนถึงสเลฟตัวสุดท้าย ซึ่งลักษณะการต่อจะเป็นการต่อแบบซีเรียล (Seriel) กันไปเรื่อยๆ

2.5 เค โอ เค โปรโตคอล (K O K Protocol)

เค โอ เค โปรโตคอล เป็นเงื่อนไขการรับ-ส่งข้อมูลที่พัฒนาขึ้นมา เพื่อใช้ในระบบการสื่อสารข้อมูลแบบ RS-485 โดยได้ประยุกต์มาจาก DDCMP Protocol , BSC Protocol , HDLC Protocol และ Xedec Protocol เพื่อสร้างชุดการรับ-ส่งของข้อมูล และการประยุกต์ใช้งานต่อไป โดยมีรายละเอียดดังต่อไปนี้

เฟรม (Frame) ของข้อมูล ประกอบด้วย HEADER PORTION และ DATA PORTION

SYN	SYN	SYN	HEADER	DATA PORTION	SYN	SYN	SYN

HEADER PORTION ในส่วนของ เค โอ เค โปรโตคอล ได้ประยุกต์มาจากบางส่วนของ

DDCPM (Digital Data Communication Message Protocol)

HDLC (High Level Data Link Control)

Xedec Protocol

โดยมีรายละเอียดดังนี้

SOH	TPD	EXC	CDAT	FSEQTF	Fserr	ADDRESS	LRC#1
8 Bit	2 Bit	4 Bit	10 Bit	8 Bit	8 Bit	24 Bit	8 Bit

SOH : Start of Header [8 Bit]

เป็นส่วนที่แสดงการเริ่มต้นของข้อความ HEADER PORTION มีขนาด 8 บิท (Bit)

TPD : Type of Data [2 Bit]

เป็นส่วนที่ใช้แสดงถึงชนิดของข้อมูลที่บรรจุอยู่ในส่วนของดาต้า (DATA PORTION)

TPD	ลักษณะของข้อมูล
00	ข้อความเป็นข้อมูล
01	ข้อความเป็นคำ
10	ข้อความเป็นคำสั่ง และพารามิเตอร์
11	ข้อความเป็นข้อความตอบรับจากตัวตอบรับ

EXC : Expantion of Count [4 Bit]

เป็นส่วนแสดงของ CDAT มีขนาด 4 บิท

บิทแรก = 0 แสดงว่า CDAT มีขนาด 10 บิท

= 1 แสดงว่า CDAT มีขนาด 13 บิท (เพิ่มจาก 3 บิท ของ EXC)

CDAT : Count of Data [10 Bit]

เป็นส่วนแสดงของข้อมูลที่บรรจุอยู่ใน Data Field ของ DATA PORTION โดยการนับข้อมูลแบบไบนารี (Binary) ได้สูงสุด 1 กิโลไบท์/เฟรม (Kilo-Byte/Frame) (เราสามารถที่จะขยายได้ 8 กิโลไบท์)

FSEQTF : File Sequence Transfer [8 Bit]

เป็นส่วนแสดงการรับ-ส่งข้อมูลอย่างต่อเนื่อง ครั้งละหลายๆเฟรม มีขนาด 8 บิท ประกอบด้วย

FSQ	NFs	P/F	NFr
1 Bit	3 Bit	1 Bit	3 Bit

FSQ : Frame Sequence Bit [1 Bit]

0 : ส่งข้อความทีละเฟรม ไม่ต้องพิจารณาอีก 7 บิท ของ FSEQTF และ FSerr

1 : ส่งข้อความทีละหลายๆเฟรม พิจารณาอีก 7 บิท ของ FSEQTF และ Fserr (ส่งต่อเนื่องไม่เกิน 8 เฟรม ต่อครั้ง)

NFs : Number of Frame Sequence Begin to Send [3 Bit]

แสดงลำดับของเฟรมที่จะส่ง (0-7) มีขนาด 3 บิท

P/F : Poll / Final Bit [1 Bit]

กรณีที่เซ็นเตอร์ (Center) ส่งข้อความให้เทอร์มินอลบิท P/F เป็น F : Final

F = 1 ยินดีให้เทอร์มินอลตอบรับได้

F = 0 ยังไม่ยินดีให้เทอร์มินอลตอบรับได้

กรณีเซ็นเตอร์ส่งข้อความให้เซ็นเตอร์บิท P/F เป็น F : Final

F = 1 ยินดีให้เซ็นเตอร์ตอบรับได้

F = 0 ยังไม่ยินดีให้เซ็นเตอร์ตอบรับได้

FSerr : Frame Sequence Error

แสดงข้อความที่ผิดพลาด มีขนาด 8 บิท FSerr ประกอบด้วย

F7	F6	F5	F4	F3	F2	F1	F0
1 Bit	1 Bit	1 Bit	1 Bit	1 Bit	1 Bit	1 Bit	1 Bit

ตัวอย่าง ถ้าเราส่งข้อมูล 5 Frame (Frame 0 - 4) ผลของ Frame 1 และ Frame 3 เกิดผิดพลาด

FSerr = 00001010 b

ADDRESS : Address of Node [24 Bit]

เป็นส่วนแสดงค่าตำแหน่งของตัวรับที่ตัวส่งต้องการติดต่อด้วย มีขนาด 24 บิท เมื่อตัวรับเป็นเซ็นเตอร์

ADDRESS = CXX : C = 43 h (ASCII)

: XX = 01-32 (ASCII 2 byte)

เมื่อตัวรับเป็นตัวเทอร์มินอล

TERMINAL = TXX : T = 54 h (ASCII)

LRC#1 : Longitudinal Redundancy Check

เป็นส่วนที่แสดงความผิดพลาด และเป็นรหัสเช็ค (Check) ความผิดพลาด HEADER มีขนาดเป็น 8 บิท

DATA PORTION

ในส่วนหนึ่งของ DATA PORTION ของ K O K Protocol ได้ประยุกต์มาจากบางส่วนของ

HDLC Protocal : High Level Data Link Control Protocal

BSC Potocol : Binary Synchronous Communication Protocol

Xedec Protocol

โดยมีรายละเอียดของ DATA PORTION ดังนี้

STX	DATA FIELD	ETX	LRC#2
8 Bit	1 - 8 Bit	8 Bit	8 Bit

STX : Start of Text [8 Bit]

เป็นการแสดงการเริ่มต้นของข้อความส่วน DATA PORTION มีขนาด 8 บิท

ETX : End of Text [8 Bit]

เป็นส่วนแสดงการสิ้นสุดของข้อความสิ้นสุดของข้อความส่วน DATA PORTION มีขนาดบิท

LRC#2 : Longitudinal Redundancy Check [8 Bit]

เป็นส่วนหนึ่งแสดงรหัสการสิ้นสุดของข้อความส่วน DATA PORTION มีขนาด 8 บิท

DATA FIELD [1 - 8 Bit]

เป็นส่วนแสดงข้อมูลที่ใช้ในการติดต่อ มีขนาดไม่เกิน 1024 ตัวอักษรต่อหนึ่งเฟรม (ข้อความขยายได้ 8192 ตัวอักษร) โดยมีรูปแบบดังนี้

COMMAND	SEP	SDB	DATA	EDB
8 Bit	8 Bit	8 Bit	ASCII CODE	8 Bit

COMMAND [8 Bit]

คือ ชุดคำสั่งของ K O K Protocol

SEP : Seperator [8 Bit]

เป็นตัวกั้นระหว่างคำสั่ง กับข้อมูล หรือพารามิเตอร์

SDB : Start of Data Block [8 Bit]

เป็นส่วนการแสดงการเริ่มต้นของข้อมูล หรือพารามิเตอร์

DATA : Data of Parameter (ASCII)

เป็นข้อมูล หรือพารามิเตอร์ รูปแบบเป็นรหัสแอสกี

ชุดคำสั่งของ เค โอ เค โปรโตคอล

ชุดคำสั่ง หรือฟังก์ชั่น (Function) การทำงานของ เค โอ เค โปรโตคอล ได้พัฒนาเพื่อการประยุกต์ด้านการสื่อสารข้อมูลในระบบ RS-485 โดยมีรายระเอียดของคำสั่งดังต่อไปนี้

TEST : Test [8 Bit]

รูปแบบ : | STX | TEST | ETX |

: สำหรับตรวจสอบความพร้อมของตัวรับ เพื่อติดต่อส่งข้อมูล (TPD = 01 b)

: สำหรับแสดงความพร้อมของตัวรับ เพื่อติดต่อส่งข้อมูล (TPD = 11 b)

RESET : Reset [8 Bit]

รูปแบบ : | STX | RESET | ETX |

: สำหรับรีเซต (Reset) ระบบของตัวรับ

: สำหรับตอบรับการรีเซตระบบ

ACK : Positive Acknowledgement [8 Bit]

รูปแบบ : | STX | ACK | ETX |

: สำหรับข้อความที่รับได้ถูกต้อง

NAK : Negative Acknowledgement [8 Bit]

รูปแบบ : | STX | NAK | ETX |

: สำหรับข้อมูลที่รับได้ เกิดการผิดพลาด และร้องขอให้ส่งข้อความซ้ำ (TPD = 01 b)

2.5.1 รูปแบบการโพลข้อมูล เค โอ เค โปรโตคอล

โพลข้อมูลจากเทอร์มินอล T01 (แบบธรรมดา), T02 (แบบต่อเนื่อง)

ตัวส่ง		ตัวรับ
SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0) Fserr (0), T01 , LRC#1 , STX , TEST , ETX , LRC#2	→	T01 ?
T01 ready	⇐⇐⇐⇐	SOH , TPD (11 b), EXC (0), CDAT (1), FSEQTF (0), Fserr (0), T01 , LRC#1 , STX , TEST , ETX , LRC#2
SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0), Fserr (0), T01 , LRC#1 , STX , ACK , ETX , LRC#2	→	ACK
SOH , TPD (01), EXC (0), CDAT (1), FSEQTF (0), Fserr (0), T01 , LRC#1 , STX , DATA , ETX , LRC#2	→	POLL
transfer data	←	SOH , TPD (11 b), EXC (0), CDAT (1028), FSEQTF (0), Fserr (0), T01 , LRC#1 , STX , DATA [1 K], ETX , LRC#2
SOH , TPD (01), EXC (0), CDAT (1), FSEQTF (0) Fserr (0),T01 , LRC#1 , STX , DATA , ETX , LRC#2	→	ACK

SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0), Fserr (0), T02 , LRC#1 , STX , TEST , ETX , LRC#2	→	T02 ?

T02 ready ← SOH, TPD(11b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T02, LRC#1, STX, TEST, ETX, LRC#2

SOH, TPD(01b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T02, LRC#2, STX, ACK, ETX, LRC#2 → ACK

SOH, TPD(01b), EXC(0), CDAT(1), FSEQTF(0), Fserr(01b), T02, LRC#1, STX, DATA, ETX, LRC#2 → POLL

เทอร์มินอล ส่งข้อมูลต่อเนื่องจำนวนเฟรมให้เซ็นเตอร์

F0 ← SOH, TPD(11b), EXC(0), CDAT(1028), FSEQTF(1,0,0,0), Fserr(0), T02, LRC#1, STX, DATA[1K], ETX, LRC#1

F1 ← SOH, TPD(11b), EXC(0), CDAT(1028), FSEQTF(1,1,0,0), Fserr(0), T02, LRC#1, STX, DATA[1K], ETX, LRC#2

F2 ← SOH, TPD(11b), EXC(0), CDAT(1028), FSEQTF(1,2,0,0), Fserr(0), T02, LRC#1, STX, DATA[1K], ETX, LRC#2

F3 ← SOH, TPD(11b), EXC(0), CDAT(1028), FSEQTF(1,3,0,0), Fserr(0), T02, LRC#1, STX, DATA[1K], ETX, LRC#2 เหลือ 1 เฟรม P/F = F = 1 ยินยอมให้เซ็นเตอร์ตอบรับได้

SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (1 , 0 , 1 , 3),
Fserr (88 h), T02 , LRC#1 , STX , DATA [1 K], ETX , LRC#2 → Frame 3 error
< FSerr = 88 h = 10001000 b >

เทอร์มินอลส่งข้อมูล Frame 4 และ Frame 3

F4 ← SOH , TPD (11 b), EXC (0), CDAT (1028),
FSEQTF (1 , 4 , 0 , 0), Fserr (0), T02 , LRC#1 ,
STX , DATA [1 K], ETX , LRC#2

F3 ← SOH , TPD (11 b), EXC (0), CDAT (1028),
FSEQTF (0 , 3 , 0 , 0), Fserr (0), T02 , LRC#1 ,
STX , DATA [1 K], EDTX , LRC#2

SOH , TPD (01 b), EXC (0), CDAT (0), FSEQTF (0),
Fserr (0), T02 , LRC#2 , STX , ACK , ETX , LRC#2 → ACK

รีเซตพารามิเตอร์การรับ-ส่งข้อมูล

SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0),
Fserr (0), T02 , LRC#1 , STX , ACK , ETX , LRC#2 → reset FSEQTF

T02 RSTFS ← SOH , TPD (11 b), EXC (0), CDAT (1),
FSEQTF (0), Fserr , T01 , LRC#1 ,
RFTFS , ETX , LRC#2

Fserr (0), T02 , LRC#1 , STX , ETX , LRC#2 → ACK

รูปแบบการติดต่อโดยการใช้คำสั่งของ K O K Protocol

ตัวส่ง		ตัวรับ
SOH, TPD(01b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T01, LRC#1, STX, TEST, CLR, ETX, LRC#2	→	T01 ?
T01 ready	←	SOH, TPD(11b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T01, LRC#1, STX, TEST, ETX, LRC#2
SOH, TPD(01b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T01, LRC#1, STX, TEST, ETX, LRC#2	→	ACK
SOH, TPD(01b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T01, LRC#1, STX, TIME[time], ETX, LRC#2	→	set time
time set	←	SOH, TPD(11b), EXC(0), CDAT(1), FSEQTF(0), Fserr, T01, LRC#1, STX, TIME, ETX, LRC#2
SOH, TPD(01b), EXC(0), CDAT(0), FSEQTF(0), Fserr(0), T01, LRC#1, STX, ACK, ETX, LRC#2	→	ACK

การเคลียร์ (Clear) ข้อมูลในบัฟเฟอร์ (Buffer)

SOH, TPD(01b), EXC(0), CDAT(1), FSEQTF(0), Fserr(0), T01, LRC#1, STX, TEST, CLR, ETX, LRC#2	→	T01 ?
T01 ready	←	SOH, TPD(11b), EXC(0), CDAT(1), FSEQTF(0), Fserr, T01, LRC#1, STX, TEST, ETX, LRC#2

SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0),
Fserr (0), T01 , LRC#1 , STX , ACK , ETX , LRC#2 → ACK

SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0),
Fserr (0), T01 , LRC#1 , STX , CLR , [07 h], ETX , LRC#2 → clear page 1 , 2 , 3
= 00000111 b

clear ok ← SOH , TPD (11 b), EXC (0), CDAT (1),
FSEQTF (0), Fserr , T01 , LRC#1 ,
STX , CLR [00], ETX , LRC#2

SOH , TPD (01 b), EXC (0), CDAT (1), FSEQTF (0),
Fserr (0), T01 , LRC#1 , STX , ACK , ETX , LRC#2 → ACK

2.6 การติดตั้งระบบผ่าน K O K Protocol

การติดตั้งระบบสามารถทำได้ทั้งแบบบางส่วนของระบบ หรือแบบทั้งระบบก็ได้ โดยใช้ชุดติดตั้งระบบพอร์ทเทเบิล (Portable Setup) เชื่อมต่อกับส่วนประมวลผลนั้นๆ แล้วติดตั้งระบบ

2.6.1 การติดตั้งระบบแบบทั้งระบบ

สำหรับเทอร์มินอล และเซ็นเตอร์ มีลำดับการเซตพารามิเตอร์ต่างๆ ดังนี้

- รีเซท (Reset) ระบบของตัวรับ คำสั่ง RESET
- ติดตั้งประเภทของเทอร์มินอล คำสั่ง PORT
- ติดตั้งค่าฐานเวลา คำสั่ง TIME

2.6.2 การติดตั้งระบบแบบบางส่วนของระบบ

การติดตั้งแบบบางส่วนของระบบของส่วนประมวลผล คือเทอร์มินอล และเซ็นเตอร์ สามารถเลือกคำสั่งในการติดตั้งได้ และไม่ต้องทำตามลำดับขั้นตอน โดยเลือกเอาเฉพาะฟังก์ชั่นที่ต้องการใช้งานเท่านั้น

2.6.3 Timeout Procedure of K O K Protocol

ในการติดตั้งระบบเริ่มแรก เซ็นเตอร์ สามารถกำหนดค่า Timeout แบบอัตโนมัติ ได้ดังนี้

- Receive Timeout :

เมื่อเซ็นเตอร์ทำการโพล (Poll) ตรวจสอบความพร้อมของทุกเทอร์มินอล เพื่อทำการติดต่อรับ-ส่งข้อมูลโดยใช้คำสั่ง TEST เซ็นเตอร์ จะเริ่มสั่งไทม์เมอร์ (Timer) ภายใน MCS-51 ให้หยุดทำงาน เมื่อเทอร์มินอล ณ. ตำแหน่งที่ทำการตรวจสอบอยู่ตอบรับกลับมา จากนั้นเซ็นเตอร์จะนำค่าเวลาที่ใช้ในการตรวจสอบแต่ละเทอร์มินอลมาคูณด้วยสอง และเก็บเป็นค่าไทม์เมอร์ สำหรับการติดตั้งแต่ละเทอร์มินอล ไว้ เมื่อเซ็นเตอร์ทำการติดต่อรับ-ส่งข้อมูลในรูปต่อไปแล้ว ไม่ได้รับการตอบรับจากเทอร์มินอล ภายในเวลาไทม์เอาท์ (Timeout) ที่เซ็นเตอร์ กำหนดไว้ เซ็นเตอร์จะทำการติดต่ออีกครั้งหนึ่ง (Retransmission)

- Disconnect Timeout :

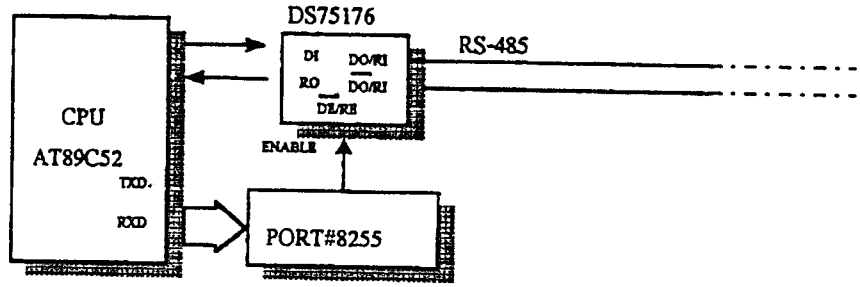
ถ้าเซ็นเตอร์ไม่ได้รับการตอบรับใดๆ จากเทอร์มินอลภายใน 10 วินาที เซ็นเตอร์จะถือว่าฟังก์ชั่นของการรับ-ส่งข้อมูลของเทอร์มินอล ณ. ตำแหน่งนั้นมีปัญหา เซ็นเตอร์จะทำการตัดเทอร์มินอลออกจากระบบโครงข่าย และแจ้งให้ทราบ

- Continue Timeout :

เมื่อเซ็นเตอร์ต้องการจะติดต่อกับเทอร์มินอล แต่เทอร์มินอลไม่พร้อมที่จะติดต่อกับเซ็นเตอร์ในขณะนั้น เนื่องจากกำลังทำงานในส่วนอื่นที่มีความสำคัญสูงกว่าการรับ-ส่งข้อมูล เทอร์มินอลสามารถส่งข้อความ TTD แจ้งให้เซ็นเตอร์ทราบได้ภายในเวลา Timeout หลังจากที่ได้ข้อความจากเซ็นเตอร์

2.7 การประยุกต์ใช้งาน MCS-51 ร่วมกับ RS-485 และ K O K Protocol

MCS-51 เป็นไมโครคอนโทรลเลอร์ที่ถูกเลือกนำมาใช้งานเป็นหน่วยประมวลผลกลางของเทอร์มินอลโมดูล (Terminal Module) และเซ็นเตอร์โมดูล (Center Module) สำหรับการประยุกต์ใช้งาน เราจะให้แต่ละโมดูล (Module) สามารถรับ-ส่งข้อมูลแบบอนุกรมผ่านทาง RS-485 และ K O K Protocol ได้ โดยการเชื่อมต่อ MCS-51 เข้ากับคู่ตัวรับ-ส่ง DS 75176 ดังแสดงในรูป



รูปที่ 2.10 แสดงการเชื่อมต่อระหว่าง MCS-51 กับ DS 75176

MCS-51 มีฟังก์ชั่นสำหรับการสื่อสารข้อมูลอนุกรมกับคอมพิวเตอร์ หรืออุปกรณ์ที่ใช้ในการสื่อสารข้อมูลได้ โดยใช้รีจิสเตอร์ (Register) SBUF เป็นที่พักข้อมูลที่ต้องการรับ หรือส่ง และใช้รีจิสเตอร์ SCON (Serial Port Control Register) เป็นส่วนควบคุมการสื่อสารอนุกรม นอกจากนี้ MCS-51 สามารถใช้ไทม์เมอร์ 1 ทำงานร่วมกับรีจิสเตอร์ TCON (Timer/Counter Control Register) และรีจิสเตอร์ TMOD (Timer/Counter Mode Control Register) เพื่อกำหนดอัตราการรับ-ส่งข้อมูล

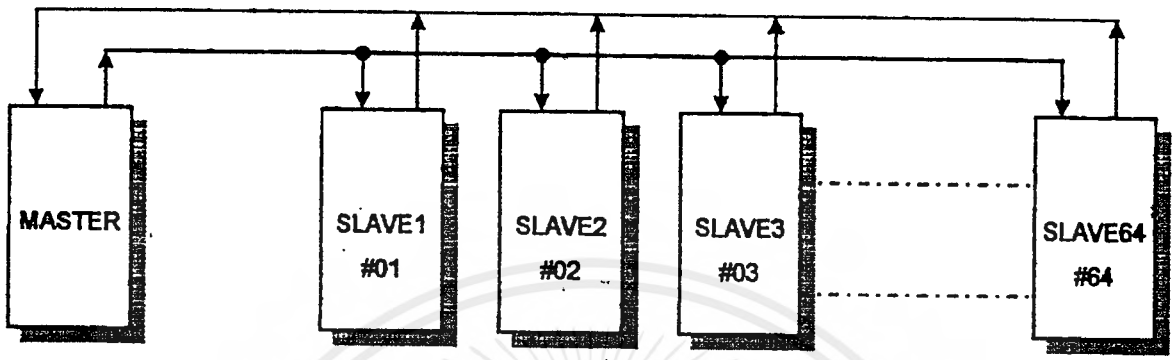
โหมดของการสื่อสารข้อมูลอนุกรมของ MCS-51 มีอยู่ด้วยกัน 4 โหมด คือ

โหมด (Mode) 0 : Shift Register Mode
โหมด 1 : Standard UART Mode
โหมด 2 : Multiprocessor Fixed Mode
โหมด 3 : Multiprocessor Variable Mode

การใช้งานในระบบการสื่อสารข้อมูลแบบ RS-485 เราเลือกใช้โหมด 3 และใช้อัตราการรับ-ส่งข้อมูล 1200 บิทต่อวินาที

2.7.1 การรับ-ส่งข้อมูลแบบอนุกรมโครงข่าย RS-485 ด้วย MCS-51

การใช้งานเราจะให้ MCS-51 สามารถรับ-ส่งข้อมูลอนุกรมกับ MCS-51 ตัวอื่นๆ ที่เชื่อมต่อกันเป็นโครงข่ายดังรูป



รูปที่ 2.11 แสดงการเชื่อมต่อการรับ-ส่งข้อมูลแบบอนุกรมแบบมัลติโปรเซสเซอร์ (Multiprocessor) ของ MCS-51

จากรูปที่ 2.11 การทำงานแบบโครงข่ายเริ่มต้นด้วย

MCS-51 ตัวแม่ ส่งข้อความให้กับ MCS-51 ตัวลูกที่ต้องการติดต่อด้วย โดยระบุตำแหน่งของตัวลูกที่ต้องการติดต่อด้วย

MCS-51 ตัวลูกทุกตัวที่อยู่ภายในโครงข่าย ได้รับข้อความจากตัวแม่ พร้อมอ่านค่าตำแหน่งตัวลูกจากข้อความดังกล่าว มาตรวจสอบ

MCS-51 ตัวลูกทุกตัว ตรวจสอบค่าตำแหน่งที่ตัวแม่ส่งมา ว่าเป็นค่าตำแหน่งของตัวเองหรือไม่

ถ้าใช่ ให้ส่งข้อความตอบรับกลับไปยัง MCS-51 ตัวแม่

ถ้าไม่ใช่ ไม่ต้องส่งข้อความใดๆ กลับไปยัง MCS -51 ตัวแม่

จากการทำงานของระบบแบบโครงข่ายที่ตัวแม่ และตัวลูก ใช้ DS 75176 เป็นชุดรับ-ส่งข้อมูล จะประสบปัญหา ในกรณีที่ตำแหน่งของตัวลูกที่ตัวแม่ต้องการติดต่อด้วย ส่งข้อมูลตอบรับออกมาที่สายสัญญาณรับ-ส่งข้อมูล นอกจากตัวแม่จะได้ข้อความตอบรับแล้ว ตัวลูกตำแหน่งอื่นๆ ที่อยู่ในโครงข่ายก็ได้รับข้อความดังกล่าวด้วย ซึ่งข้อความตอบรับนี้ ตัวแม่เท่านั้นที่ควรได้รับ การแก้ไขในส่วนนี้ คือโดยการพิจารณารีจิสเตอร์ SCON

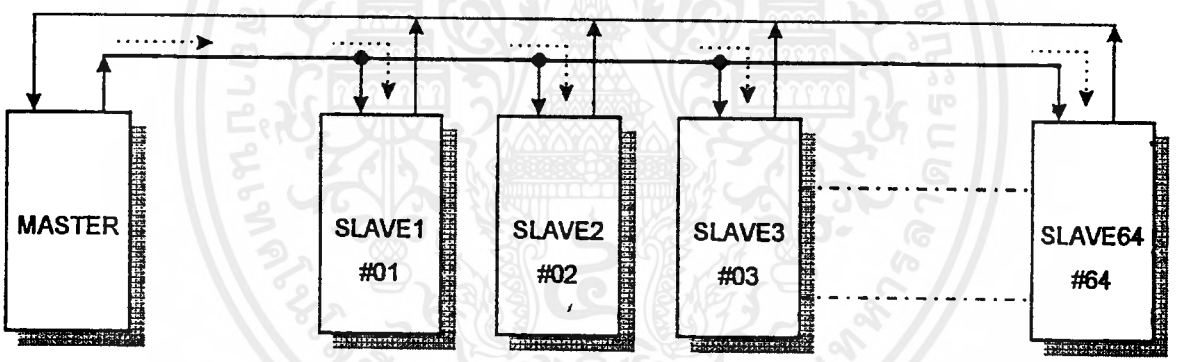
จากรายละเอียดของบิท SM2 จะพบว่า ถ้าตัวรับตั้งค่า SM2 = 1 การอินเทอร์รัพท์ (Interupt) การรับข้อมูลอนุกรม โดยข้อมูลอนุกรมจากตัวส่ง (RI) จะเกิดขึ้นโดยตัวส่งจะต้องตั้งบิท TB8 = 1 ซึ่งค่านี้เมื่อตัวรับ รับมาแล้วจะถูกเก็บไว้ในบิท RB8 ของตัวรับ หรือถ้าตัวรับตั้งค่า SM2 = 0 การอินเทอร์รัพท์ การรับข้อมูลอนุกรมโดยข้อมูลอนุกรมจากตัวส่ง (RI) จะเกิดขึ้น โดยตัวส่งจะต้องตั้งบิท TB8 = 0 ซึ่งค่านี้เมื่อตัวรับ รับมาแล้วจะถูกเก็บเอาไว้ในบิท RB8 ของตัวรับ

ดังนั้นลำดับการทำงานของการตั้งค่ารีจิสเตอร์ SCON ของตัวแม่และตัวลูก เมื่อใช้ฟังชั่น TEST ของ เค ไอ เค ไปรโตคอล ในการทำงานแบบมัลติโปรเซสเซอร์ แสดงดังรูป

เริ่มต้นด้วย

MCS-51 ตัวแม่ส่งข้อความ TEST ให้กับ MCS-51 ตัวลูก ตำแหน่ง T01 แสดงดังรูป

SOH , TPD (01 b) , EXC (0) , CDAT (1) , FSEQTF (0) ,
Fserr (0) , T01 , LRC#1 , STX , TEST , CLR , ETX , LRC#2 ⟶ T01 ?

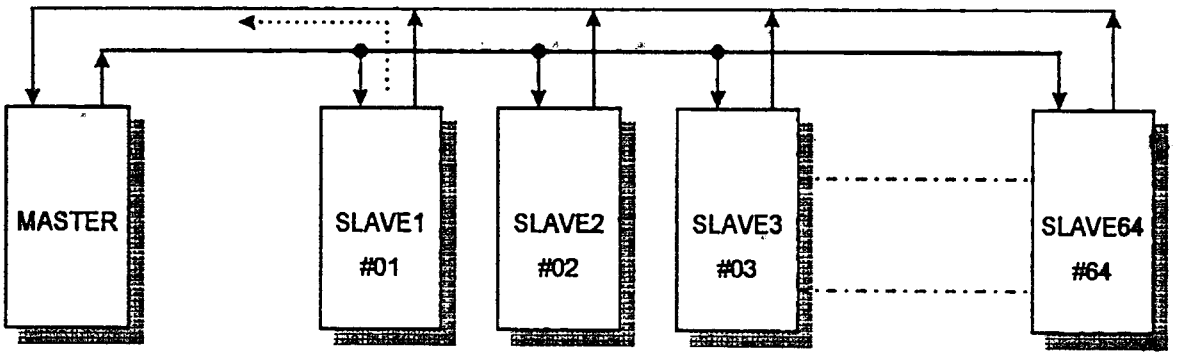


รูปที่ 2.12 แสดงพารามิเตอร์ SCON ของตัวแม่ และตัวลูก เมื่อตัวแม่ส่งข้อความ TEST

ตัวลูกทุกตัวที่อยู่ในโครงข่าย ได้รับข้อความจากตัวแม่ พร้อมกับอ่านค่าตำแหน่งตัวลูก จากข้อความดังกล่าวมาตรวจสอบ ซึ่งตำแหน่งตัวลูกที่ตัวแม่ต้องการติดต่อด้วยคือ T01

T01 ส่งข้อความตอบรับ TEST ตัวแม่ ดังแสดงดังรูป

T01 ready ⟵ SOH , TPD (11 b) , EXC (0) , CDAT (1) ,
FSEQTF (0) , Fserr , T01 , LRC#1 ,
STX , TEST , ETX , LRC#2

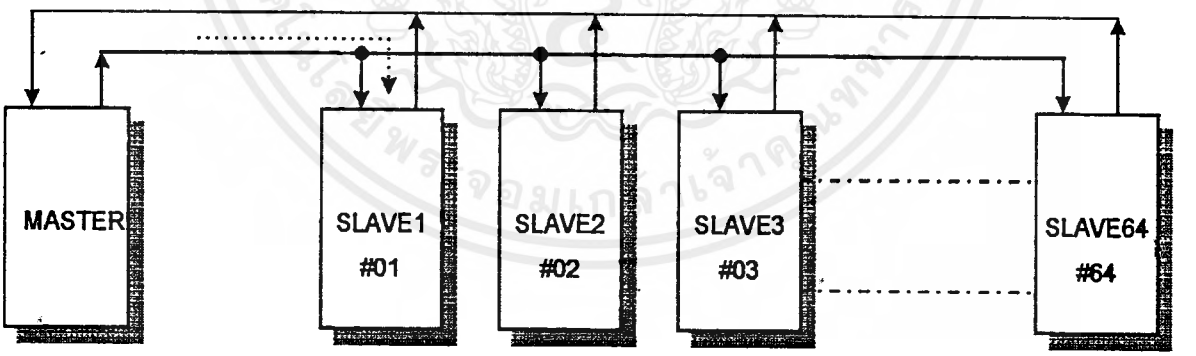


รูปที่ 2.13 แสดงพารามิเตอร์ SCON ของตัวแม่ และตัวลูก เมื่อ T01 ส่งข้อความตอบรับ TEST

จากรูป เมื่อตัวแม่ส่งข้อความ TEST ให้ตัวลูกในโครงข่ายแล้ว ตัวแม่จะเปลี่ยนค่า SM2 จาก 1 เป็น 0 เพื่อรอรับค่าตำแหน่งที่ถูกต้องแล้ว จะทำการตั้งค่า SM2 และ TB2 มีลอจิก (Logic) เป็น " 0 " เพื่อให้ข้อความส่งออกไป ซึ่งข้อความตอบรับนี้มีเพียงตัวแม่เท่านั้นที่สามารถรับได้ ส่วนตัวลูกอื่นๆ ไม่สามารถรับได้

ตัวแม่ส่งข้อความ แสดงความถูกต้องของการตอบรับของข้อความ TEST ให้ T01

SOH , TPD (01 b) , EXC (0) , CDAT (1) , FSEQTF (0) ,
Fserr (0) , T01 , LRC#1 , STX , ACK , ETX , LRC#2 → ACK.



รูปที่ 2.14 แสดงพารามิเตอร์ SCON ของตัวแม่ และตัวลูก เมื่อตัวลูกส่งข้อความ ACK

จากรูป ตัวแม่ต้องการที่จะส่งข้อความ ACK ให้กับ T01 ซึ่งข้อความดังกล่าว มีเพียงตัว T01 ที่สามารถรับได้ ส่วนตัวลูกอื่นๆ ไม่สามารถที่จะรับได้ และเมื่อตัวลูกส่งข้อความ ACK ให้กับ T01 แล้ว พารามิเตอร์ SCON ของตัวแม่ และ T01 จะกลับไปมีค่าเริ่มต้นใหม่

ในส่วนของไมโครคอนโทรลเลอร์นั้น จะมีหน้าที่ในการควบคุมการทำงานทั้งหมดของวงจร ซึ่งจะเป็นหัวใจสำคัญที่จะทำให้เครื่องควบคุมการเปลี่ยนช่องการจราจรนั้นมีความสลับซับซ้อนเพียงใดนั้น ก็ขึ้นอยู่กับการเขียนโปรแกรม (Program) ทางผู้จัดทำนั้น เลือกไมโครคอนโทรลเลอร์ เบอร์ AT89C52 ซึ่งการทำงานต่างๆ และโครงสร้างต่างๆ คล้ายคลึงกับไมโครคอนโทรลเลอร์ เบอร์ 8031 ของอินเทล (Intel) มาก แต่มีข้อที่เหนือกว่าอยู่ คือมีแฟลชเมโมรี (Flash Memory) อยู่ภายใน คือสามารถที่จะทำการบันทึกโปรแกรมเข้าไปไว้ภายในตัวของมันได้เลย เป็นผลทำให้มันสามารถที่จะรัน (Run) แบบซิงเกิลชิฟ (Single Ship) ลำพัง โดยที่ไม่ต้องทำการต่ออีพรอม (EPROM) เพิ่มเติม ซึ่งมีรายละเอียดดังต่อไปนี้

ส่วนของฮาร์ดแวร์ ในทฤษฎี จะขอกล่าวเฉพาะบางส่วนเท่านั้น โดยจะกล่าวถึงหน่วยประมวลผลกลาง และส่วนแสดงผลจอแอลซีดี (LCD)

2.8 คุณสมบัติของไมโครคอนโทรลเลอร์ เบอร์ AT89C52

1. เป็นไมโครคอนโทรลเลอร์ขนาด 8 Bit
2. มีวงจรออสซิลเลเตอร์ (Oscillator) และวงจรผลิตสัญญาณนาฬิกา (Clock Oscillator) ภายในไอซี (IC)
3. มีขาสัญญาณอินพุต-เอาต์พุต จำนวน 32 Bit
4. สามารถเชื่อมต่อหน่วยความจำข้อมูลภายนอก (External Data Memory) โดยอ้างแอดเดรส (Address) ได้ถึง 64 Kbyte
5. มีหน่วยความจำโปรแกรมภายในตัวขนาด 8 Kbyte
6. มีหน่วยความจำข้อมูลภายในตัวขนาด 256 Byte
7. หน่วยความจำข้อมูลภายในบางส่วนสามารถเข้าถึงข้อมูลระดับบิตได้ด้วย ส่งผลให้การเขียนโปรแกรมทำได้ง่ายมากขึ้น
8. มีไทม์เมอร์/เคาน์เตอร์ (Timer/Counters) ขนาด 16 Bit จำนวน 3 ตัว
9. การอินเตอร์รัปต์สามารถทำได้จาก 6 แหล่งกำเนิด โดยการอินเตอร์รัปต์ยังสามารถจัดระดับความสำคัญได้เป็น 2 ระดับ
10. มีพอร์ตสื่อสารอนุกรมภายในตัวเองซึ่งทำงานเป็นแบบฟูลล์-ดูเพล็กซ์
11. มีชุดคำสั่งในการคำนวณทางคณิตศาสตร์ และทางตรรกศาสตร์ และยังใช้ชุดคำสั่งเดียวกันกับไมโครคอนโทรลเลอร์ ตระกูล MCS-51
12. มีคำสั่ง โดยส่วนใหญ่ ใช้เวลาการทำงานเพียง 1 ไมโครวินาที (uSec) เมื่อใช้คริสตอล (Crystal) ความถี่ 12 เมกะเฮิร์ต (MHz)
13. สามารถใช้เครื่องคอมพิวเตอร์ PC ทำการลบ และเขียนโปรแกรมได้โดยตรง
14. สามารถโปรแกรมทับได้นับพันครั้ง เหมือนกับการโปรแกรมในลักษณะเดียวกันกับ EEPROM โดยการโปรแกรมสามารถโปรแกรมได้ในลักษณะ " In - System Programming " บนการ์ดคอนโทรล

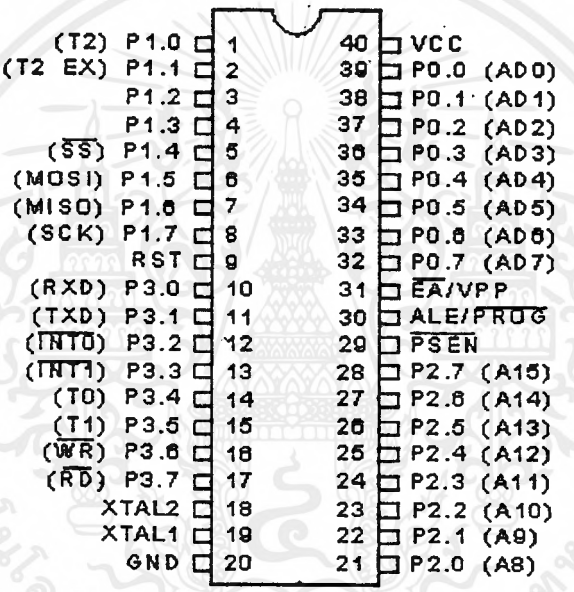
(Controller Card) เราออกแบบขึ้นมาได้เลย หรือจะโปรแกมโดยใช้เครื่องโปรแกรมทั่วไป เช่น Pro-100 เป็นต้น

15. ในการแก้ไขปรับปรุงโปรแกรมเดิม หรือการโปรแกรมใหม่ สามารถทำได้ โดยการโปรแกรม ทับลงไป โดยไม่ต้องลบ ด้วยแสงอัลตร้าไวโอเลต (Ultraviolet)

16. ต้องการแหล่งจ่ายไฟ 5 Volt เพียงชุดเดียว

2.8.1 โครงสร้างภายนอกของไมโครคอนโทรลเลอร์ เบอร์ AT89C52

ไมโครคอนโทรลเลอร์ เบอร์ AT89C52 จะมีตำแหน่งขาพื้นฐานที่เหมือนกันกับไมโครคอนโทรลเลอร์ เบอร์ 80C52 และไมโครคอนโทรลเลอร์ทั่วไป ดังแสดงในรูปที่ 2.14



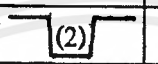
รูปที่ 2.15 แสดงการจัดตำแหน่งขาต่างๆของไมโครคอนโทรลเลอร์ เบอร์ AT89C52

2.8.2 การโปรแกรม AT89C52

โดยปกติแล้ว ไมโครคอนโทรลเลอร์นี้จะอยู่ในสภาวะว่าง (Contect = FF h) และพร้อมที่จะได้รับการโปรแกรมซึ่งสามารถโปรแกรมได้ ทั้งแบบใช้แรงดันค่าสูง (+ 12 V) และในแบบแรงดันค่าต่ำ (ที่ระดับแรงดัน Vcc นั่นคือ + 5 V) กรณีโปรแกรมในโหมดแรงดันไฟค่าสูง 12 V สามารถทำได้ โดยการใช้เครื่องโปรแกรมทั่วๆไป ที่สามารถจะโปรแกรมลงใน EEPROM ได้ โดยใช้ซอฟต์แวร์ ที่สามารถโปรแกรม AT89C52 ส่วนการโปรแกรมแบบค่าแรงดันต่ำนั้น ทำให้สามารถโปรแกรมได้ในแบบ In - System Programming ได้

2.8.3 ขั้นตอนการโปรแกรม

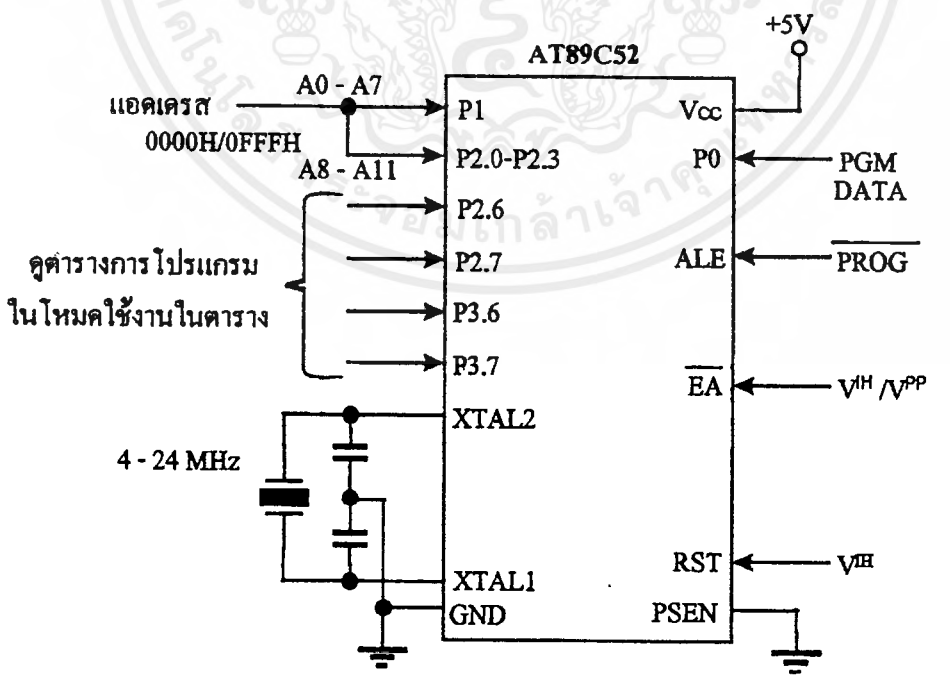
ก่อนโปรแกรม AT89C52 ไลน์แอดเดรส (Line Address) ข้อมูล และสัญญาณควบคุม จะต้องกำหนดขึ้นตามตาราง และจะต่อขาใช้งาน และสัญญาณเข้าไปยัง AT89C52 ตามรูปที่ 2.15 และการตรวจสอบการโปรแกรม ต่อตามรูปที่ 2.3

ขาต่อใช้งาน / โหมดการทำงาน	RST	$\overline{\text{PSEN}}$	ALE/ $\overline{\text{PROG}}$	$\overline{\text{EA}}$/ Vpp	P2.6	P2.7	P3.6	P3.7
เขียนรหัสข้อมูล	H	L		H/12V(1)	L	H	H	H
อ่านรหัสข้อมูล	H	L	H	H	L	L	H	H
(2)	H	L		H/12V	H	L	L	L
กำหนดไบต์การอ่าน	H	L		H	L	L	L	L

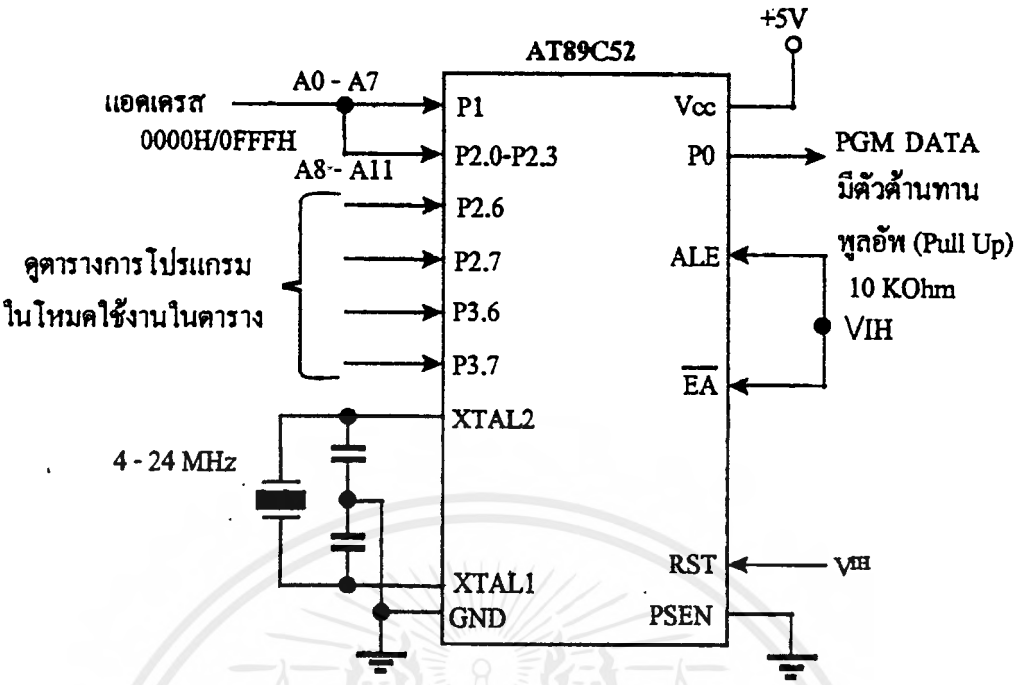
หมายเหตุ (1) ระดับแรงดันในการเขียน และลบข้อมูลที่ระดับแรงดันค่าสูง (+ 12 V)

(2) คาบเวลาของพัลส์ (Pulse) ในการลบข้อมูลในชิบ (Chip) เท่ากับ 10 มิลลิวินาที (mSec)

ตารางแสดงการเซต (Set) ค่าสัญญาณขณะทำการโปรแกรม



รูปที่ 2.16 แสดงการต่อใช้งานขณะโปรแกรมเข้าไปใน AT89C52



รูปที่ 2.17 แสดงการทดสอบการโปรแกรมใน AT89C52

จากนั้นจะเริ่มกระทำตามขั้นตอนดังนี้

1. ป้อนค่าตำแหน่งแอดเดรสบนไลน์แอดเดรส
2. ป้อนไบต์ (Byte) ข้อมูลเข้าทางไลน์ (Line) ข้อมูล
3. กำหนดค่าสัญญาณควบคุมที่ถูกต้อง เข้าทางขา P2.6 , P2.7 ,P3.6 , P3.7
4. กำหนดค่าแรงดันป้อนให้ขา $\overline{EA}$/Vpp ไว้ที่ + 12 V ในกรณีที่ค่าแรงดันสูง
5. ป้อนพัลส์ที่ขา ALE/PROG เมื่อจะโปรแกรมหนึ่งไบต์เข้าไปในหน่วยความจำแฟลช จากนั้น วงรอบการเขียนข้อมูลจะเกิดขึ้นเองตามมา โดยในระยะเวลาใน 1 รอบ จะไม่เกิน 1.5 มิลลิวินาที เสร็จแล้ว เริ่มทำขั้นตอนที่ 1 ถึง 5 โดยการเปลี่ยนแอดเดรส และข้อมูลที่อยู่ถัดไปจนครบทั้งหมด หรือได้รับ Object File ที่ต้องการแล้ว

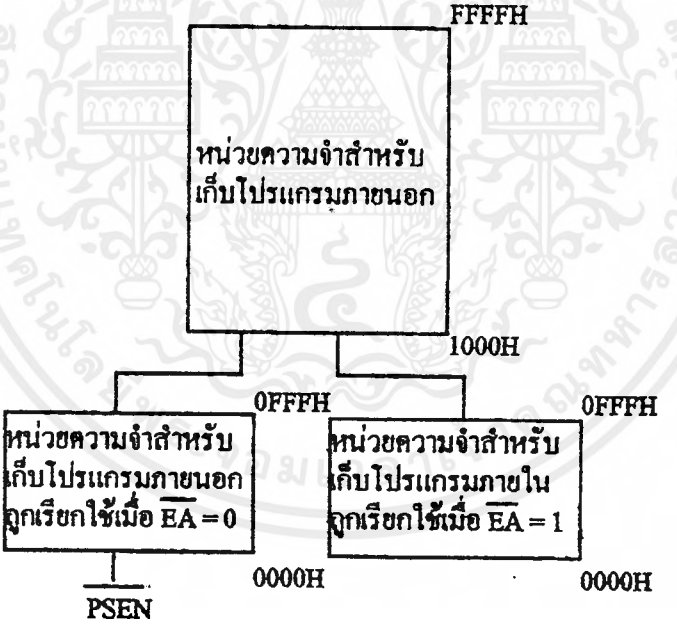
2.8.4 หน่วยความจำภายในของไมโครคอนโทรลเลอร์ เบอร์ AT89C52

ในไมโครคอนโทรลเลอร์ เบอร์ AT89C52 แบ่งชนิด หรือหน้าที่ของหน่วยความจำออก เป็น 2 ส่วน คือหน่วยความโปรแกรม (Program Memory) และหน่วยความจำข้อมูล (Data Memory)

หน่วยความจำโปรแกรมจะใช้สำหรับเก็บโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ ซึ่งจะเป็นหน่วยความจำแบบแฟลช สำหรับหน่วยความจำข้อมูลจะใช้สำหรับเก็บข้อมูล หรือค่าตัวแปรต่างๆ จากการทำงานของโปรแกรม ซึ่งในเบอร์ AT89C52 จะมีหน่วยความจำในส่วนนี้เท่ากับ 256 ไบต์ ซึ่งจะเป็นหน่วยความจำแบบ RAM

2.8.5 หน่วยความจำโปรแกรม

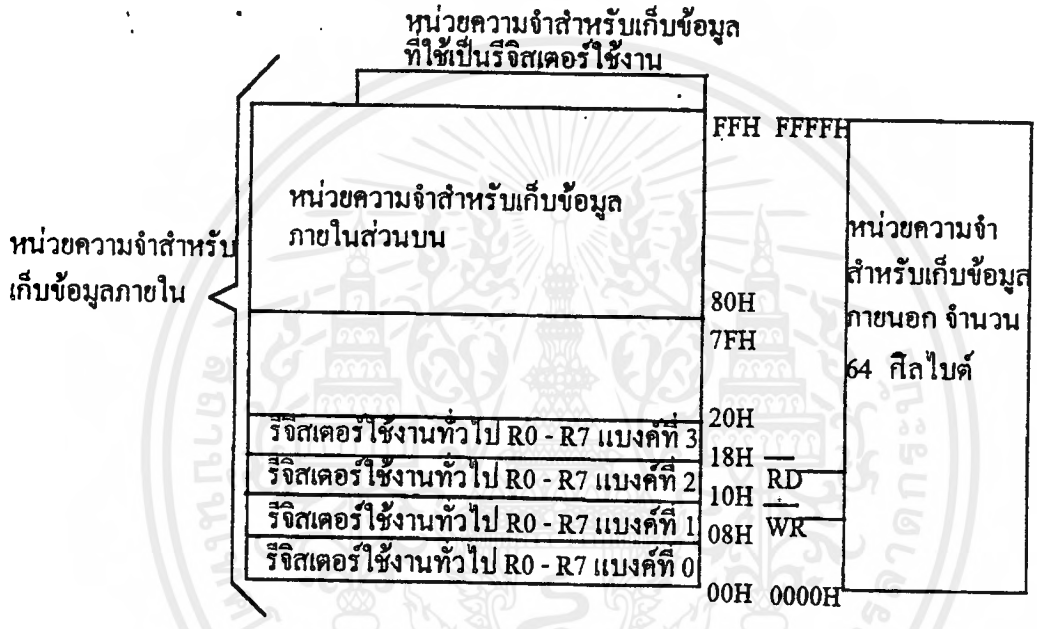
หน่วยความจำโปรแกรมสามารถแบ่งออกได้เป็น 2 ส่วน คือหน่วยความจำโปรแกรมภายใน (Internal Program Memory) และหน่วยความจำโปรแกรมภายนอก (External Program Memory) หน่วยความจำโปรแกรมภายในถูกเลือกใช้งาน ถ้าขาสัญญาณ $\overline{EA}$ มีค่าเป็น " 1 " โดยจะถูกใช้งานในช่วงแอดเดรส 0 - 1FFF h นอกเหนือจากช่วงแอดเดรสนี้ จะใช้หน่วยความจำโปรแกรมภายนอกทั้งหมด ในกรณีตรงกันข้าม ถ้าขาสัญญาณ $\overline{EA}$ มีค่าเป็น " 0 " ในช่วงแอดเดรส 0 - 1FFF h จะถูกใช้จากหน่วยความจำภายนอก หรือกล่าวได้ว่า ถ้าขาสัญญาณ $\overline{EA}$ มีค่าเป็น " 0 " จะเป็นการเลือกใช้หน่วยความจำโปรแกรมภายนอกทั้งหมดตลอดช่วงแอดเดรสดังแสดงในรูปที่ 2.17



รูปที่ 2.18 หน่วยความจำสำหรับเก็บโปรแกรม

2.8.6 หน่วยความจำข้อมูล

หน่วยความจำข้อมูลสามารถแบ่งออกได้เป็น 2 ส่วน คือ หน่วยความจำข้อมูลภายใน และหน่วยความจำข้อมูลภายนอก สำหรับหน่วยความจำข้อมูลภาย ในยังแบ่งออกได้เป็น 2 ส่วนย่อย คือ ส่วนที่ใช้เก็บข้อมูลทั่วไป จะถูกใช้สำหรับเก็บข้อมูล หรือค่าตัวแปรต่างๆ จากการทำงานของโปรแกรม และส่วนที่ใช้เป็นรีจิสเตอร์ หรือที่เรียกว่า " SFR " (Special Function Register) หน้าที่พิเศษ โดยที่ส่วนนี้จะใช้ควบคุมการทำงาน และบอกสถานะการทำงานของไมโครคอนโทรลเลอร์ ไมโครคอนโทรลเลอร์ เบอร์ AT89C52 จะมีหน่วยความจำข้อมูลภายในขนาด 256 ไบต์ ดังแสดงในรูปที่ 2.18



รูปที่ 2.19 หน่วยความจำสำหรับเก็บข้อมูล

2.8.7 รีจิสเตอร์หน้าที่พิเศษ (SFR)

รีจิสเตอร์หน้าที่พิเศษมีบทบาทอย่างมาก ในการควบคุมการทำงานของไมโครคอนโทรลเลอร์ และทำให้การเขียนโปรแกรมสามารถทำได้สะดวกมากขึ้น รีจิสเตอร์หน้าที่พิเศษทำหน้าที่สำคัญ คือควบคุมการทำงานในส่วนต่างๆ ภายในไมโครคอนโทรลเลอร์ และทำหน้าที่แสดงสถานะการทำงาน ซึ่งในรีจิสเตอร์หน้าที่พิเศษบางตัว ยังสามารถเข้าถึงได้ในระดับบิต (Bit Addressable) ด้วย ดังแสดงรูปที่ 2.19 การจัดหน่วยความจำ และตำแหน่งของรีจิสเตอร์หน้าที่พิเศษต่างๆ

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAPL	RCAPH	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

BIT ADDRESSABLE

รูปที่ 2.20 แสดงการจัดหน่วยความจำ และตำแหน่งของรีจิสเตอร์หน้าที่พิเศษต่างๆ

2.8.8 รีจิสเตอร์ใช้งานทั่วไป

รีจิสเตอร์ใช้งานทั่วไปมีไว้สำหรับให้ผู้เขียนโปรแกรมสามารถนำข้อมูลไปพักไว้ชั่วคราว หรือใช้งานทั่วไปได้ตามต้องการ ซึ่งรีจิสเตอร์ใช้งานทั่วไปนี้ มีอยู่ด้วยกัน 8 ตัว ถูกจัดให้อยู่รวมกัน และมีให้เลือกใช้ถึง 4 แบงค์ (Bank) นั่นคือ มีรีจิสเตอร์ใช้งานทั่วไปถึง 32 ตัวให้ใช้งาน เพียงแต่ การเลือกใช้รีจิสเตอร์ R0 - R7 ในแบงค์ใดแบงค์หนึ่งจะถูกกำหนดจากบิต RS0 , RS1 ในรีจิสเตอร์หน้าที่พิเศษ PSW ดังนั้น การเลือกใช้จึงเลือกได้เพียงแบงค์เดียวในขณะใดขณะหนึ่ง อย่างไรก็ตาม ค่าข้อมูลที่เก็บไว้ในรีจิสเตอร์แบงค์ใดก็ตามที่มีชื่อเดียวกันแต่อยู่คนละแบงค์จะไม่มีผลซึ่งกันและกันเลย ทำให้ผู้เขียนโปรแกรมใช้งานรีจิสเตอร์ทั่วไปนี้ ได้ทั้ง 32 ตัว อย่างเต็มที่ และไม่ยุ่งยากในการเขียนโปรแกรม

2.9 จอแอลซีดี (LCD)

จอแอลซีดีมีหน้าที่แสดงค่าของฟังก์ชั่นต่างๆของเครื่อง เช่น วัน เดือน ปี และ เวลา นอกจากนั้นยังใช้ในโปรแกรมของเครื่อง ซึ่งทำหน้าที่แสดงผลต่างๆของการกดคีย์ (Key)

การอินเตอร์เฟส (Interface) กับจอแสดงผล LCD โมดูล (Liquid Crystal Display Module) ในระบบที่ใช้งานไมโครคอนโทรลเลอร์ทำงานแบบระบบโดดเดี่ยว (Stand Alone Microcontroller Applications) บ่อยครั้งที่จำเป็นต้องติดต่อกับอุปกรณ์แสดงผลด้วย เพื่อแสดงข้อความ ตัวเลข ผลการวัดค่า หรือข้อมูลต่างๆ ในการสื่อสารกับผู้ใช้งานได้ และอุปกรณ์หนึ่งที่ถูกนำมาใช้งานกันอย่างกว้างขวาง ก็คือจอแสดงผล LCD โมดูล ตัวอย่างที่เห็นกันได้บ่อยก็คือ นำมาใช้ในการแสดงผลแบบเมนู (Menu) เพื่อให้ผู้ใช้งานกดคีย์บอร์ด (Keyboard) ในการเลือกรูปแบบการทำงานต่อไป เป็นต้น

การส่งผ่านข้อมูลทั้งตัวอักขระ และคำสั่งควบคุมถูกป้อนเข้ากับบัสข้อมูล 2 ทิศทางของระบบจริงๆ แล้วการใช้งานจอแสดงผล LCD โมดูล สามารถทำงานได้ในทั้งโหมด 4 บิต (Mode 4 Bit) (D_0 - D_3) หรือโหมด 8 บิต (D_0 - D_7) ก็ได้ แต่เนื่องจากในระบบที่เราศึกษาอยู่เป็นไมโครคอนโทรลเลอร์แบบ 8 บิต ดังนั้น การเชื่อมต่อ LCD โมดูล จึงต่อใช้งานแบบ 8 บิต กับบัส (Bus) ข้อมูลได้พอดี และสามารถเขียนโปรแกรม ให้ทำงานในโหมดส่งผ่านข้อมูล 8 บิต กับจอ LCD ได้ทันที ซึ่งเป็นการต่อแบบเมโมรีแมพ (Memory Mapping)

คำสั่งควบคุมการทำงานของ LCD โมดูล

คำสั่งควบคุมการทำงานของ LCD โมดูล ที่ใช้ไมโครคอนโทรลเลอร์ของฮิตาชิ (HITACHI) ตามคาต้าบุ๊ค (Data Book) นั้น มีความยาวประมาณ 30 หน้า สำหรับคำอธิบายครบทุกคุณสมบัติของ LCD โมดูล ดังนั้นในที่นี้คงจะกล่าวถึงคำสั่ง และการใช้งานทั้งหมดคงไม่ไหวแน่ การนำเสนอคงทำได้เฉพาะคำสั่งที่มีความสำคัญ และควรทราบครอบคลุมการใช้งานทั่วไปได้เพียงเท่านั้น ถ้าผู้อ่านท่านใดต้องการศึกษารายละเอียดของ LCD โมดูลรุ่นใดก็คงต้องหาคาต้าบุ๊คจากผู้ผลิต หรือผู้แทนจำหน่ายมาศึกษา

ความสามารถหลักที่ LCD โมดูลนิยมถูกหยิบยกมาใช้งาน ก็เพราะมันสามารถแสดงผลตัวอักขระแอสกีได้ (ASCII Character) ซึ่งมีอยู่ในไมโครคอนโทรลเลอร์ของ LCD โมดูล และใน LCD โมดูล แต่ละรุ่นก็มักใช้ไมโครคอนโทรลเลอร์ เบอร์เดียวกันเสมอ นั่นคือ คำสั่งควบคุมต่างๆ จึงใช้เหมือนกันใน LCD โมดูลแต่ละรุ่น เช่น ตารางชุดคำสั่งทั่วไป ดังแสดงในตาราง ซึ่งใช้งานได้กับ LCD โมดูลทั่วไป เช่น รุ่น H2570 , LM016L , LM1612A เป็นต้น

ก่อนที่จะเริ่มต้นศึกษารูปแบบคำสั่งควบคุมต่างๆ ผู้อ่านควรทราบถึงคำศัพท์บางคำเกี่ยวกับองค์ประกอบ และการทำงานพื้นฐานของไมโครคอนโทรลเลอร์ใน LCD โมดูลกันก่อน การทำงานภายใน LCD โมดูลจะมีบัฟเฟอร์อยู่ภายใน ซึ่งมีความจุประมาณ 80 ตัวอักขระมีชื่อเรียกว่า DD-RAM (Display Data RAM) มีตำแหน่งแอดเดรสอยู่ระหว่าง 00 h ถึง 4F h ยกตัวอย่าง เช่น ถ้าเป็น LCD โมดูล ขนาด 16 ตัวอักษรต่อ 1 บรรทัด DD-RAM ที่จัดเก็บข้อมูลตัวอักขระที่จะแสดงผลทั้ง 16 ตัวอักษรจะถูกเก็บอยู่ที่ตำแหน่งแอดเดรส 00 h ถึง 0F h โดยเริ่มต้นจากด้านซ้ายของจอแสดงผล

การทำให้เกิดช่องว่าง (Window) ใน DD-RAM สามารถทำได้โดยการใช้คำสั่งชิฟต์ (Shift) ตัวอักขระในการแสดงผล หรืออีกวิธีหนึ่งที่ง่ายมากขึ้น ก็คือขณะที่มีข้อมูลตัวอักขระเก็บไว้ในแอดเดรสค่าหนึ่ง การแสดงผลตัวอักขระต่อไป หรือทำให้เกิดช่องว่างอาจทำได้โดยการกำหนดจุดเริ่มต้นของแอดเดรสใหม่แตกต่างจากเดิมไป 1 แอดเดรสของ DD-RAM ซึ่งก็ทำให้ได้ผลลัพธ์เช่นเดียวกัน

ในกรณีที่เป็นจอแสดงผล LCD โมดูลแบบ 2 บรรทัด เช่น ในรุ่น LM16255 เป็นต้น บรรทัดแรกจะใช้ตำแหน่งแอดเดรสเริ่มต้นที่ 00 h สำหรับตัวอักขระเริ่มต้น และในบรรทัดที่ 2 จะเริ่มต้นที่ตำแหน่งแอดเดรส 40 h การเขียนโปรแกรมสำหรับ LCD โมดูลแบบ 2 บรรทัดย่อมมีความซับซ้อนมากกว่า LCD โมดูล แบบ 1 บรรทัด แต่นั่นก็ไม่ใช่ปัญหาใหญ่ หากผู้อ่านเข้าใจการควบคุมการทำงานได้ดีพอแล้ว

ผู้เขียนโปรแกรม หรือผู้ใช้งานสามารถทราบถึงการแสดงผลด้วยเคอร์เซอร์ (Cursor) ซึ่งเป็นตัวชี้ตำแหน่งต่อไปของตัวอักขระที่จะแสดงผลใน DD-RAM และตำแหน่งนี้ถูกเรียกว่า แอดเดรสเคาน์เตอร์ (Address Counter) เคอร์เซอร์สามารถกำหนดให้มีการแสดงผล หรือไม่ก็ได้ขึ้นอยู่กับการเขียนโปรแกรม และกำหนดให้มีการแสดงผลแบบกระพริบได้ด้วย

โดยทั่วไปการเขียนโปรแกรม เพื่อควบคุมการแสดงผลของ LCD เมื่อมีการส่งข้อมูลตัวอักขระไปยัง LCD คือให้ทำการเลื่อนตัวอักขระเดิมไป และเลื่อนเคอร์เซอร์ตามไปด้วย เพื่อชี้ตำแหน่งของตัวอักษรต่อไป การเขียนโปรแกรมให้แสดงผลแบบนี้ ทำให้ง่ายต่อการอ่าน และเข้าใจของผู้ใช้งานได้มากที่สุด

ใน LCD โมดูลมีหน่วยความจำส่วนหนึ่งที่สำคัญก็คือ CG-RAM ซึ่งทำหน้าที่เก็บข้อมูลรายละเอียดโครงร่างของตัวอักษรที่สามารถนำมาแสดงผลได้ หน่วยความจำของตัวอักษรที่สามารถนำมาแสดงผลได้ หน่วยความจำในส่วนนี้ผู้ใช้งานสามารถที่จะสร้างตัวอักขระใดๆ ได้ตามต้องการแล้วเก็บลงในหน่วยความจำส่วนนี้ สำหรับเรียกมาใช้งานได้ด้วย ถ้าหากตัวอักขระที่มีอยู่แล้วนั้น ไม่เพียงพอต่อการใช้งาน การใช้งานในส่วนของ CG-RAM นี้สามารถดูได้จากดาต้าชีต (Data Sheet) ของ LCD โมดูลรุ่นนั้นๆ ที่ซื้อมาใช้งานได้

การส่งคำสั่งควบคุมไปยัง LCD โมดูลสามารถกำหนดเป็นรหัสคำสั่งต่างๆ ซึ่งจะกล่าวต่อไปนี้ออกที่ตำแหน่งแอดเดรส 0E001 H โดยกำหนดให้ขาสัญญาณ RS มีค่าเป็น " 0 " และเซตโหมดให้เป็นการอ่าน หรือเขียนได้ด้วยขาสัญญาณ R/W ต่อไปจะกล่าวถึงคำสั่งพื้นฐานที่ใช้งานบ่อยครั้งในการควบคุมการทำงานของ LCD โมดูลสำหรับเครื่องหมายดอกจัน (*) แทนสถานะของบิตที่จะเป็นสถานะใดก็ได้ไม่สนใจ และไม่มีผลกับคำสั่งนั้นๆ

คำสั่งเคลียร์จอแสดงผล (Clear Display)

D_7							D_0
0	0	0	0	0	0	0	1

เมื่อไมโครคอนโทรลเลอร์ประมวลผลคำสั่งนี้จะทำให้ข้อมูลใน DD-RAM ทั้งหมด ถูกแทนที่ด้วยค่า 20H = Space ในรหัสแอสกี ทำให้จอแสดงผลไม่ปรากฏตัวอักขระใดๆ บนจอภาพ เคอร์เซอร์จะถูกเซตให้อยู่ที่ตำแหน่งเริ่มต้นใหม่อีกครั้ง และยกเลิกผลการใช้คำสั่งเลื่อนข้อมูล (Display Shift) ที่ผ่านมาแล้ว

คำสั่งเลื่อนเคอร์เซอร์ไปยังตำแหน่งเริ่มต้น (Return Home)

D_7							D_0
0	0	0	0	0	0	1	★

คำสั่งนี้มีผลทำให้รีเซตเคอร์เซอร์ให้กับไปอยู่ที่ตำแหน่งเริ่มต้นใหม่ และรีเซตคำสั่งเลื่อนข้อมูลที่ผ่านมาแล้ว โดยที่ข้อมูลตัวอักขระใน DD-RAM ไม่เกิดการเปลี่ยนแปลง นั่นคือตัวอักขระบนจอแสดงผลจะยังคงเหมือนเดิมไม่เปลี่ยนแปลง

คำสั่งกำหนดโหมดป้อนข้อมูล (Entry Mode Set)

D_7							D_0
0	0	0	0	0	1	I/D	S

คำสั่งนี้ถูกใช้ประโยชน์ในการกำหนดการทำงานของ LCD โมดูล หลังจากเกิดการส่งไบต์ ข้อมูลไปยังจอแสดงผลบิต I/D (Increase/Decrease Bit) ทำหน้าที่กำหนดการเพิ่ม (I/D = 1) หรือลด (I/D = 0) ค่าตำแหน่งแอดเดรสใน DD-RAM 1 แอดเดรสอัตโนมัติ เมื่อเกิดการอ่าน หรือเขียนตัวอักขระ ค่าของตำแหน่งแอดเดรสนี้ถูกเก็บอยู่ในแอดเดรสเคาน์เตอร์

บิต S (Shift Bit) เป็นบิตที่ใช้กำลัง โดยถ้าบิต S = 1 เมื่อมีการส่งไบต์ข้อมูลใหม่เกิดขึ้นตัวเคอร์เซอร์จะอยู่กับที่ แต่ตัวอักขระข้อมูลเดิมจะถูกดันไปทางซ้าย แต่ถ้าหากบิต S = 0 เมื่อเกิดข้อมูลใหม่ตัวเคอร์เซอร์ จะเลื่อนไปทางขวา

คำสั่งควบคุมการแสดงผล (Display ON/OFF Control)

D_7							D_0
0	0	0	0	1	D	C	B

คำสั่งนี้ใช้สำหรับควบคุมการแสดงผล ในการปิด หรือเปิดการแสดงผลหน้าจอ และเคอร์เซอร์ โดยปราศจากการเปลี่ยนแปลงข้อมูลใน DD-RAM ผู้เขียนโปรแกรมสามารถกำหนดหน้าจอแสดงผลให้ปิด หรือเปิดได้ด้วยบิต D โดยกำหนดให้บิต D = 1 เป็นการเปิดจอแสดงผล และถ้า D = 0 เป็นการปิดจอแสดงผล เช่นเดียวกัน สำหรับบิต C = 0 และบิต C = 1 เป็นการควบคุมให้เคอร์เซอร์เปิด หรือปิดตามลำดับ และบิต B ซึ่งเป็นบิตกำหนดว่า จะให้เคอร์เซอร์กระพริบหรือไม่

คำสั่งควบคุมการเลื่อนเคอร์เซอร์ และตัวอักขระ (Cursor or Display Shift)

D_7							D_0
0	0	0	1	S/C	R/L	*	*

คำสั่งนี้ใช้สำหรับควบคุมการเลื่อนของเคอร์เซอร์ และตัวอักขระที่แสดงผล ซึ่งมีความสำคัญ และใช้งานบ่อย เนื่องจากผู้เขียนโปรแกรมต้องแสดงผลข้อความบนจอแสดงผลตามแนวนอน

คำสั่งเซตฟังก์ชั่น (Function Set)

D_7							D_0
0	0	1	DL	N	F	*	*

คำสั่งนี้ใช้สำหรับเซตโหมดการทำงานของ LCD โมดูล หลังจากรีเซตการทำงาน หรือเริ่มต้นทำงานระบบทุกครั้ง ในที่นี้เราจะใช้ LCD โมดูลในโหมด 8 บิต และใช้งานเพียง 1 บรรทัดเท่านั้น ความหมายของบิตต่างๆ มีดังนี้

บิต DL = 1 หมายถึง ทำงานในโหมดอินเตอร์เฟสแบบ 8 บิต

บิต DL = 0 หมายถึง ทำงานในโหมดอินเตอร์เฟสแบบ 4 บิต

บิต N = 1 หมายถึง ทำงานแบบ 1 บรรทัด

บิต N = 0 หมายถึง ทำงานแบบ 2 บรรทัด หรือ มากกว่า

บิต F = 0 หมายถึง ทำงานในโหมดความละเอียด 5 * 7 จุด

บิต F = 1 หมายถึงทำงานในโหมดความละเอียด 5*10 จุด

คำสั่งเซตตำแหน่งแอดเดรสใน CG-RAM (Set CG-RAM Address)

D_7							D_0
0	1	a5	A4	a3	a2	a1	a0

คำสั่งนี้ใช้สำหรับกำหนดค่าตำแหน่งแอดเดรสใน CG-RAM ภายใน LCD โมดูล เพื่อทำการส่งผ่านข้อมูลโดยกำหนดตำแหน่งแอดเดรสใน CG-RAM แน่นอนค่าหนึ่ง เพื่อถ่ายทอดไบต์ข้อมูลต่อไป บิต a0 - a5 เป็นบิตกำหนดตำแหน่งใน CG-RAM ซึ่งจะถูกโหลดไปเก็บไว้ในแอดเดรสเคาน์เตอร์

คำสั่งเซตตำแหน่งแอดเดรสใน DD-RAM (Set DD-RAM Address)

D_7							D_0
1	A6	a5	a4	a3	a2	a1	a0

คำสั่งนี้ใช้สำหรับกำหนดค่าตำแหน่งแอดเดรสใน DD-RAM ภายใน LCD โมดูล เพื่อทำการส่งผ่านข้อมูล โดยกำหนดตำแหน่งแอดเดรสใน DD-RAM แน่นอนค่าหนึ่งเพื่อถ่ายทอดไบต์ข้อมูลต่อไป บิต a0 - a6 เป็นบิตกำหนดตำแหน่งแอดเดรสใน DD-RAM ซึ่งจะถูกโหลดไปเก็บไว้ในแอดเดรสเคาน์เตอร์

คำสั่งอ่านแฟลกบิวซี (Read Busy Flag)

D_7							D_0
BF	A6	a5	a4	a3	a2	a1	a0

เมื่อต้องการอ่านสถานะของแฟลกบิวซีต้องกำหนดขาสัญญาณ R/W เป็น " 1 " เสมอ แฟลกบิวซีเป็นตัวบอกสถานะการทำงานของ LCD โมดูลว่าอยู่ในสถานะพร้อมรับข้อมูลหรือไม่ โดยถ้า BF = 1 หมายถึงขณะนี้ LCD โมดูลยังไม่พร้อมรับข้อมูลต่อไป อันเนื่องจากกระบวนการประมวลผลจากคำสั่ง หรือไบต์ข้อมูลที่ผ่านมายังไม่เสร็จสิ้น แต่ถ้า BF = 0 หมายถึงขณะนี้ LCD โมดูล พร้อมที่จะรับคำสั่งใหม่ หรือข้อมูลใหม่ได้แล้ว นอกจากคำสั่งนี้จะทำให้ทราบสถานะของแฟลกบิวซีแล้ว ในขณะเดียวกันที่บิต a0 – a6 จะถูกอ่านด้วย ซึ่งเป็นค่าที่เก็บอยู่ในแอดเดรสเคาน์เตอร์

คำสั่งเขียนข้อมูลไปยัง CG หรือ DD-RAM (Write Data to CG or DD-RAM)

D_7							D_0
D7	D6	d5	d4	d3	d2	d1	d0

เมื่อต้องการใช้คำสั่งนี้ต้องกำหนดให้ขาสัญญาณ R/W = 0 และ RS = 1 เสมอ คำสั่งนี้ ใช้สำหรับเขียนข้อมูลไปยัง CG หรือ DD-RAM ในตำแหน่งแอดเดรสที่ต้องการ ซึ่งขึ้นอยู่กับการกำหนดตำแหน่งแอดเดรสใน CG หรือ DD-RAM ก่อนหน้าที่จะส่งข้อมูลนี้แล้ว คำสั่งนี้จะมีผลเกี่ยวเนื่องถึงการกำหนดโหมดการทำงานก่อนหน้านี้ด้วยว่า จะให้ทำการเพิ่ม หรือลดค่าใน AC หลังจากส่งผ่านไบต์ข้อมูลแล้ว ซึ่งเป็นผลมาจากคำสั่งกำหนดโหมดป้อนข้อมูล (Entry Mode Command)

คำสั่งอ่านข้อมูลจาก CG หรือ DD-RAM (Read Data from CG or DD-RAM)

D_7							D_0
D7	D6	d5	d4	d3	d2	d1	d0

เมื่อต้องการใช้คำสั่งนี้ต้องกำหนดให้ขาสัญญาณ R/W = 1 และ RS = 1 เสมอ คำสั่งนี้ ใช้สำหรับอ่านข้อมูลจาก CG หรือ DD-RAM ในตำแหน่งแอดเดรสที่ต้องการ ซึ่งขึ้นอยู่กับการกำหนดตำแหน่งแอดเดรสใน CG หรือ DD-RAM ก่อนหน้าที่จะใช้คำสั่งอ่านข้อมูลนี้แล้ว

2.9 วัอทด็อก (Watchdog)

วัอทด็อกของเครื่องควบคุมการเปลี่ยนช่องทางการจราจร ทางผู้ออกแแบบเลือกไอซีเบอร์ MAX691 ซึ่งจะทำหน้าที่ในการป้องกันการทำงานที่ผิดพลาดของไมโครคอนโทรลเลอร์ ซึ่งจะมีหน้าที่ในการทำงานอยู่สองอย่างด้วยกัน คือ

2.9.1 หน้าที่ป้องกันแหล่งจ่ายไฟเลี้ยงเครื่องตก

ซึ่งเมื่อเกิดการไฟเลี้ยงเครื่องตก จะมีผลทำให้การทำงานของเครื่องเกิดการทำงานที่ผิดพลาดได้ และข้อมูลที่ใช้เก็บอยู่ในแรม (RAM) อาจจะถูกทำลาย หรือเกิดการสูญหายได้ ดังนั้น เพื่อป้องกันเหตุที่จะเกิดขึ้นดังกล่าวจึงต้องมีตัวที่ทำหน้าป้องกันเหตุการณ์นี้

หลักการทำงานของวอทด็อก คือเราจะต่อแบตเตอรี่แบ็คอัพ (Battery Backup) เข้ากับขา V-BATT ของไอซีเบอร์ MAX691 ซึ่งแบตเตอรี่แบ็คอัพแรม หรือส่วนอื่นๆ ที่ต้องการไฟเลี้ยงตลอดเวลา โดยมีหลักการอยู่ที่ว่า เมื่อแหล่งจ่ายไฟนั้นเกิดตกขึ้นมาวอทด็อกนี้ จะทำการต่อ แบตเตอรี่แบ็คอัพเข้าไปแทนแหล่งจ่ายเดิม ทันทีที่แหล่งจ่ายมีค่าแรงดันต่ำกว่าแรงดันเทรสโฮลด์ (Threshold) หรือแรงดันอ้างอิง ซึ่งแรงดันอ้างอิงนี้ จะมีค่าอยู่ที่ 4.65 V แรงดันอ้างอิงนี้ จะได้มาจากแบตเตอรี่ที่อยู่ภายในของตัวไอซี ซึ่งสามารถที่ดูโครงสร้างภายในของไอซีได้จากบล็อกไดอะแกรม (Block Diagram)

2.9.2 ป้องกันการทำงานขั้นตอนของโปรแกรมผิดพลาด

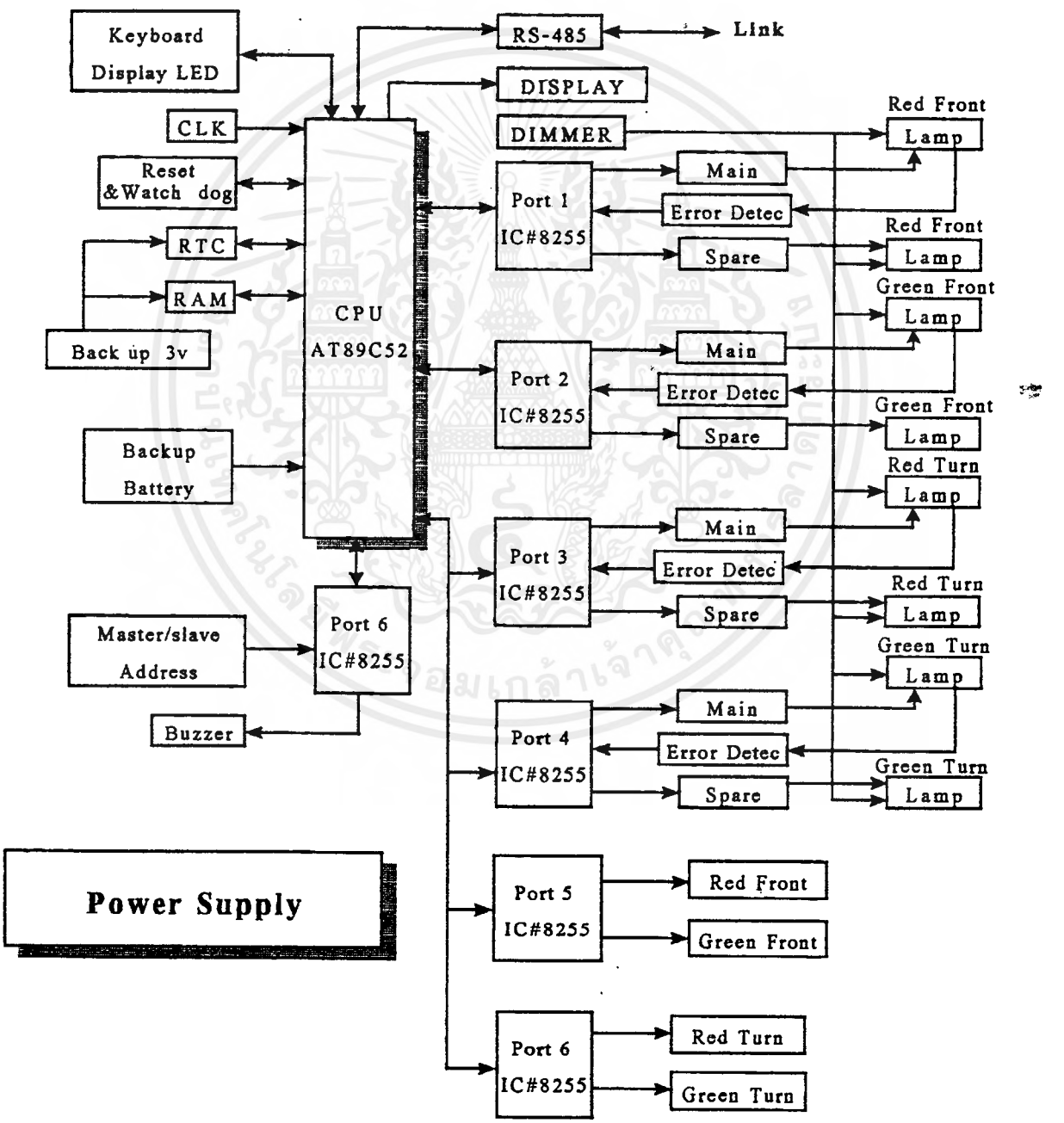
การทำงานขั้นตอนของโปรแกรมผิดพลาด อาจจะเกิดจากการที่ผู้เขียนนั้น เขียนขั้นตอนการทำงานของโปรแกรมไม่รอบคอบพอ หรือที่เรียกว่า การเกิดบัก (Bug) เนื่องจากเครื่องควบคุมการเปลี่ยนช่องทางการจราจรนั้น จะต้องการความปลอดภัยสูงจึงจำเป็นที่จะต้องป้องกันการเกิดบักไว้ล่วงหน้า

หลักการป้องกันการเกิดบัก และการทำคำสั่งผิด มีอยู่ว่าไมโครคอนโทรลเลอร์นั้น จะทำงานเป็นรอบการทำงาน หรือที่เรียกกันว่า ลูป (Loop) ซึ่งเป็นเหตุการณ์ในกรณีที่ทำงานตามปกติ แต่ถ้าหากว่าเกิดมีบักขึ้นมาในโปรแกรมก็จะเกิดการหลุดลูปจากกรณีต่างๆ นอกเหนือจากที่กล่าวมาขั้นต้น อาศัยหลักการที่กล่าวมาขั้นต้นว่า โปรแกรมจะทำงานเป็นลูปนั้น เราจะต้องสร้างพัลซ์ 1 ลูกขึ้นมา เพื่อที่จะส่งไปกระตุ้นวอทด็อกก่อนทุกๆ เวลา 1.6 วินาที มิฉะนั้นแล้ววอทด็อกนั้นจะส่งสัญญาณรีเซตให้กับไมโครคอนโทรลเลอร์ ซึ่งเมื่อเกิดเหตุการณ์หลุดลูปขึ้นมานั้นไมโครคอนโทรลเลอร์นั้น จะไม่สามารถที่จะส่งสัญญาณมากระตุ้นวอทด็อกได้ก่อนเวลา 1.6 วินาที เป็นผลให้ไมโครคอนโทรลเลอร์ถูกรีเซต เมื่อนั้นไมโครคอนโทรลเลอร์ก็จะเริ่มการทำงานใหม่หมด โดยจะเริ่มที่แอดเดรส 0000 h และจะเข้าไปอยู่ในลูปการทำงานปกติ

บทที่ 3. การออกแบบ และการสร้าง

3.1 การออกแบบให้อยู่ในของบล็อกไดอะแกรม

การออกแบบนั้นขั้นแรกของการออกแบบคือ เราจะต้องตั้งวัตถุประสงค์ของเครื่องก่อนว่าเราต้องการที่ทำอะไรแล้วจากนั้นแปลงความต้องการของวัตถุประสงค์นั้นๆออกมาอยู่ในรูปของบล็อกไดอะแกรม และจากบล็อกไดอะแกรมนี้สามารถที่จะเขียนวงจรซึ่งแสดงไว้ตอนท้ายของเล่ม โดยวัตถุประสงค์หรือหน้าที่ที่ต้องการทั้งหมดเป็นดังนี้

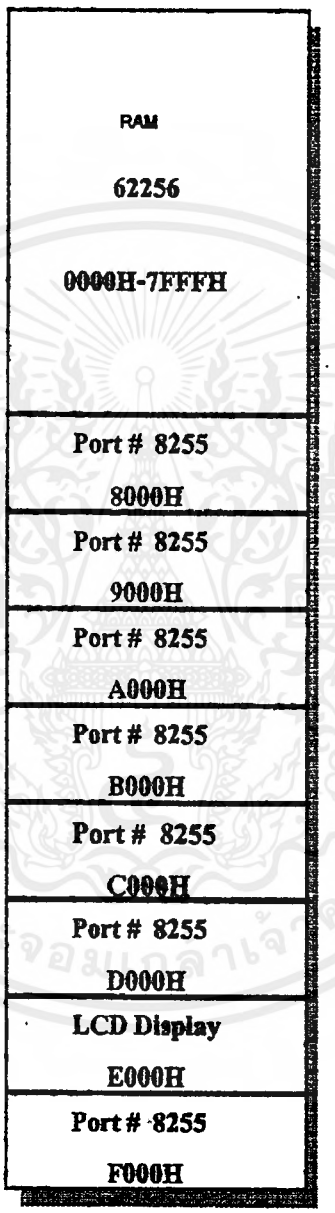


รูปที่ 3.1 แสดงบล็อกไดอะแกรมของเครื่องควบคุมการสลับช่องจราจรทั้งหมด

3.2 หลักการอินเตอร์เฟสภาคต่างๆ กับหน่วยประมวลผล

3.2.1 การแบ่งการอินเตอร์เฟสแบบเมโมรี

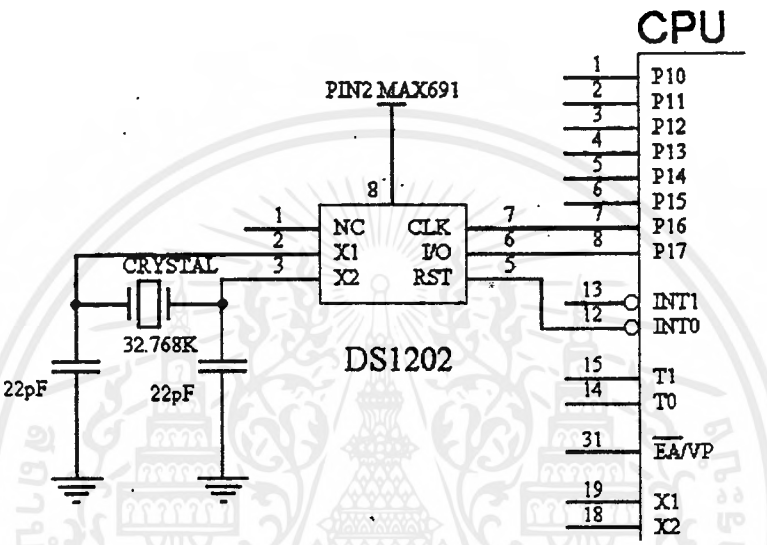
ซึ่งใน MCS-51 นั้นเราสามารถที่จะอ้างอิงหน่วยความจำได้ทั้งหมด 64 กิโลไบท์ (64Kbyte) โดยเราสามารถใช้ 74LS138 ในการดีโค้ท (Decode) แอดเดรสของหน่วยความจำ ซึ่งเราสามารถแบ่งได้ดังรูป



รูปที่ 3.2 แสดงพื้นที่หน่วยความจำทั้งหมดที่ใช้งาน

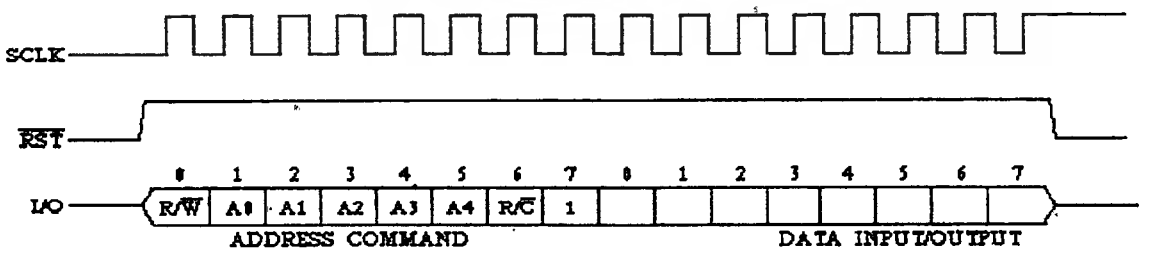
3.2.2 การอินเตอร์เฟส กับอุปกรณ์อาร์ทีซี (Real Time Clock)

อาร์ทีซี (RTC) เป็นอุปกรณ์แสดงเวลา , วัน , เดือน , ปี และสามารถที่จะอ่านเขียนได้แบบอนุกรมการอินเตอร์เฟส จะใช้สายสัญญาณในการเชื่อมต่อเพียง 3 เส้นเท่านั้น ดังที่ได้แสดงไว้ในรูปที่ 3.3 สำหรับการอ่านหรือเขียนข้อมูลกับอุปกรณ์อาร์ทีซี เราจะต้องส่งข้อมูลคำสั่ง (Command Byte) ไปยังอาร์ทีซี 8 บิต แล้วหลังจากนั้นก็เป็นการรับข้อมูลเข้ามาอีก 8 บิต ในกรณีของการอ่าน และหากเป็นกรณีของการเขียนข้อมูลลงไปก็จะเป็นการเขียนข้อมูลตามหลังข้อมูลคำสั่งอีก 8 บิต เช่นกัน ซึ่งลักษณะของสัญญาณที่ส่งไปจะเป็นสัญญาณแบบอนุกรมดังแสดงในรูปที่ 3.7



รูปที่ 3.3 แสดงการอินเตอร์เฟสกับอุปกรณ์อาร์ทีซี

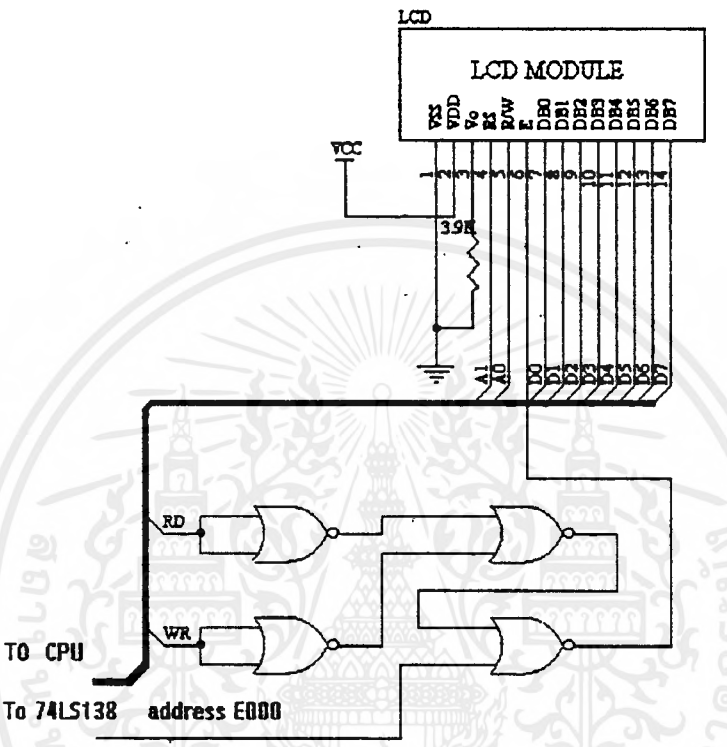
ลักษณะการส่งข้อมูลจะต้องเป็นไปตามข้อกำหนดของสัญญาณที่แสดงไว้ในรูปที่ 3.4 ไม่เช่นนั้นจะไม่สามารถที่จะติดต่อกับอาร์ทีซีได้



รูปที่ 3.4 รูปแบบของการเขียน/อ่านอุปกรณ์อาร์ทีซี

3.2.3 การอินเตอร์เฟสจอแอลซีดี

การต่อของแอลซีดี เช่นเดียวกับการต่อพอร์ท 8255 (Port 8255) คือการใช้เมโมรีแมพ (Memory Mapping) โดยทางผู้สร้างกำหนดให้ตำแหน่งของเมโมรีไว้ที่ตำแหน่ง 0E000h ซึ่งลักษณะการต่อแบบนี้จะเป็นการเขียนข้อมูลครั้งละ 8 บิท และสามารถที่จะอ่านค่าจากจอ LCD ได้เหมือนกับแรม ซึ่งมีลักษณะการต่อดังรูป

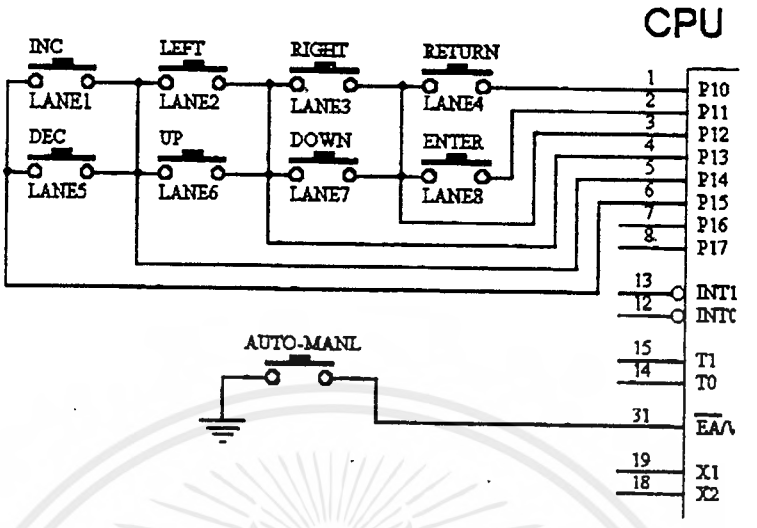


รูปที่ 3.5 แสดงการต่อจอแอลซีดีโมดูล

3.3.4 การอินเตอร์เฟสคีย์บอร์ด (Keyboard)

คีย์จะมีหน้าที่ในการเซท หรือกำหนดค่าควบคุมต่างๆของเครื่อง ซึ่งจะทำให้ผู้ที่จะใช้เครื่องสามารถที่จะทำการติดต่อกับเครื่อง และตั้งค่าควบคุมต่างๆ โดยการป้อนค่านั้นผ่านทางคีย์บอร์ด ซึ่งในเครื่องนี้จะแบ่งการต่อคีย์บอร์ดออกเป็นสองลักษณะด้วยกันคือการต่อแบบแมทริก (Matrix) และการโดยตรงแบบต่อระหว่างพอร์ทกับกราวน์ เหตุที่ต้องทำการต่อแบบนี้ก็คือ เพื่อที่ประหยัดซึ่งจะสามารถที่จะประหยัดได้ทั้งตัวสวิทช์ และพอร์ทของไมโครคอนโทรลเลอร์ ซึ่งสามารถที่จะเอาไว้ใช้ประโยชน์เพื่อการต่อกับอุปกรณ์อื่นๆ โดยที่ทั้งสองลักษณะการต่อและวัตถุประสงค์ต่างกันคือ

● การต่อแบบแมทติก ซึ่งเราเลือกที่จะกำหนดขนาดของแมทติกที่ขนาด 4×2 ซึ่งก็เพียงพอสำหรับการใช้งานแล้ว โดยที่คีย์บอร์ดจะแยกความหมายของการกดแบ่งออกเป็นสองช่วงซึ่งสามารถที่จะแสดงให้เห็นดังรูป



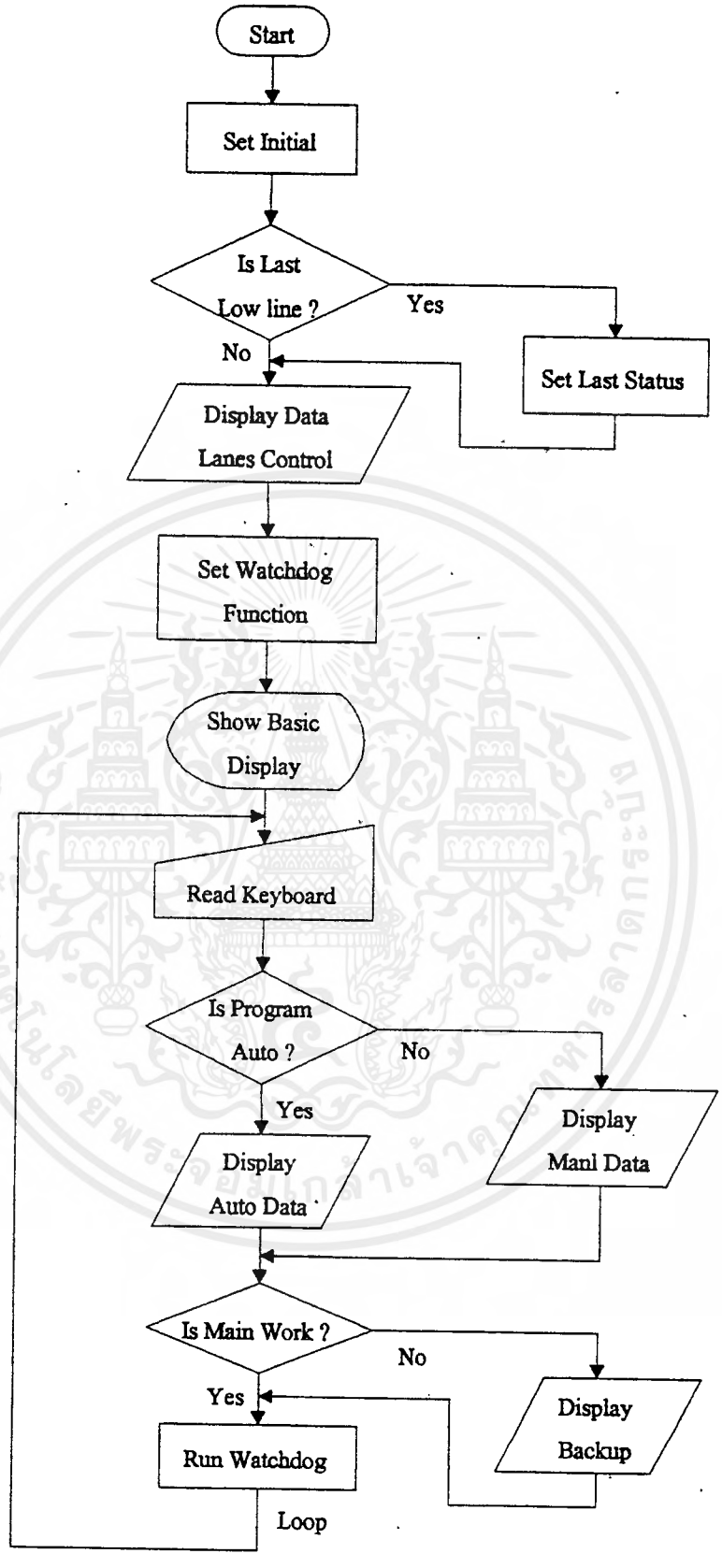
รูปที่ 3.6 แสดงการแมพคีย์บอร์ด (Keyboard Mapping)

● สภาวะการทำงานคีย์บอร์ดอยู่ในช่วงการทำงานปกติ ซึ่งคีย์บอร์ดแบบแมทตริกนี้ จะทำหน้าที่กำหนดฟังก์ชั่นต่างๆเช่น เอ็นเตอร์ (Enter) , รีเทอร์น (Return) เป็นต้น โดยในช่วงนี้ไมโครคอนโทรลเลอร์จะทำงานในสภาวะปกติ

● สภาวะการทำงานของคีย์บอร์ดอยู่ในช่วงอินเตอร์รัพท์ ซึ่งในช่วงนี้ คีย์บอร์ดแบบแมทตริกนี้จะทำหน้าที่ในการปิด , เปิดสัญญาณไฟจราจรโดยตรงจากตู้คอนโทรลโดยตรง ซึ่งการทำงานของไมโครคอนโทรลเลอร์นี้อยู่ในช่วงโปรแกรมอินเตอร์รัพ จะเข้าไปเซตค่าในการตีความหมายของค่าที่อ่านได้จากการกดคีย์ใหม่

3.3 การเขียนโปรแกรมในการควบคุม

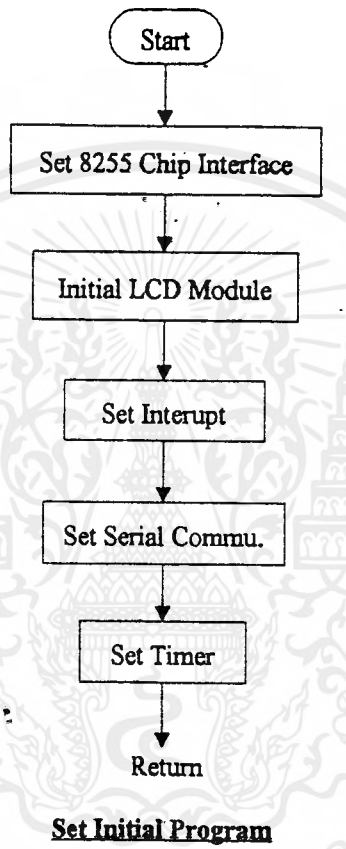
ในการเขียนใช้โปรแกรม SXA-51 เป็นคอมไพเลอร์ (Compiler) แล้วทำการกอปปี้ (Copy) โปรแกรมผ่านพอร์ทอนุกรมของคอมพิวเตอร์ไปที่ EEPROM ซึ่งเมนโปรแกรม (Main Program) เราจะแสดงไว้ในตอนท้ายของเล่ม แต่จะแสดงให้เห็นผังการทำงานของโปรแกรมส่วนต่างๆ ดังรูป



Main Program

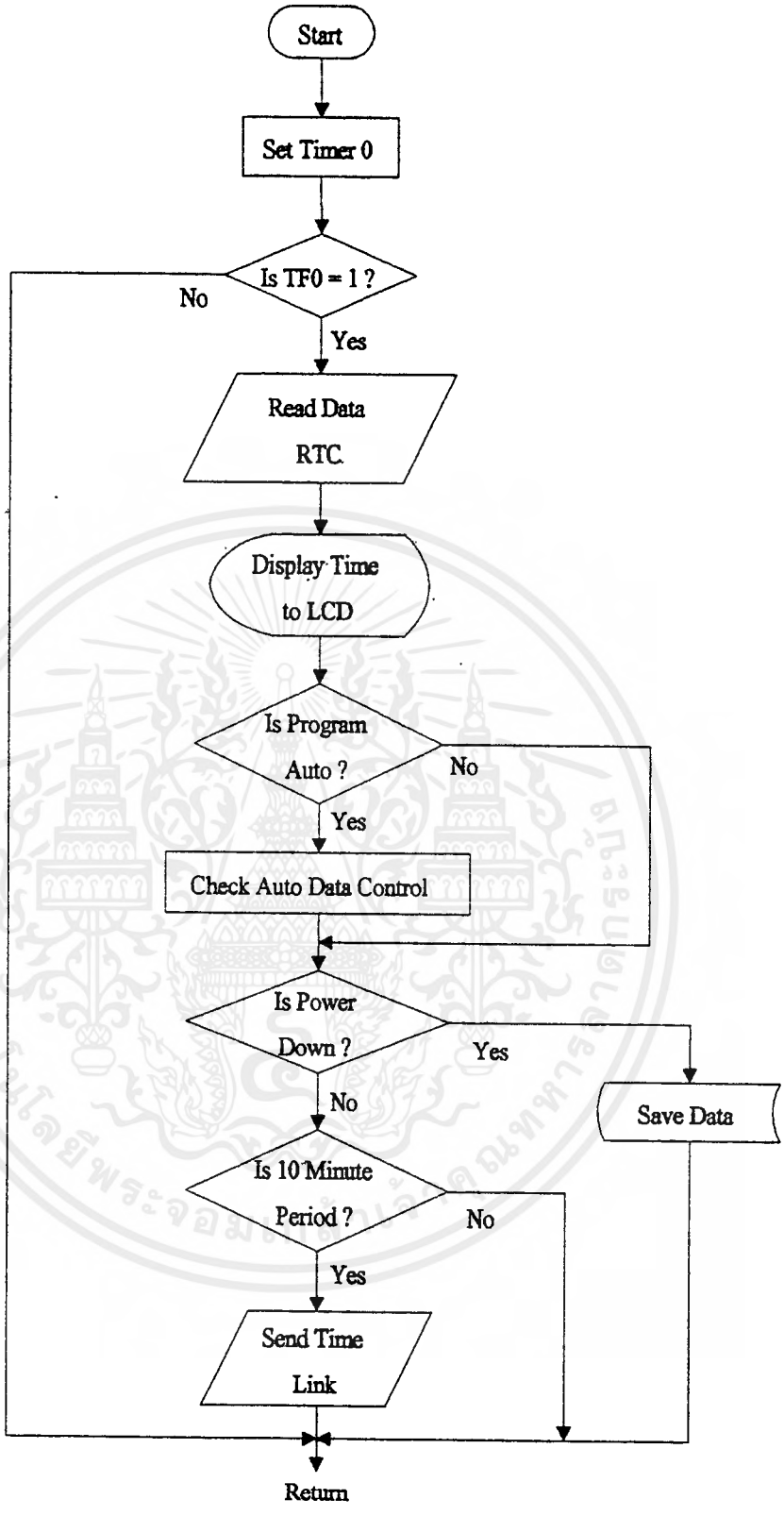
รูปที่ 3.7 แสดงไฟลวชาร์ท (Flowchart) แสดงการทำงานของเมนโปรแกรม

จากรูป เมื่อเริ่มต้นโปรแกรม เครื่องจะทำการตั้งค่าเริ่มต้นจากนั้นจะตรวจสอบสภาวะของกระแสไฟฟ้าที่สภาวะที่ผ่านมาว่าอยู่ในสภาวะปกติหรือไม่ ถ้าไม่ปกติ เครื่องจะทำการดึงข้อมูลในสภาวะที่ผ่านมากับคืนมา แต่ถ้าสภาวะของกระแสไฟฟ้าปกติดี เครื่องจะทำการแสดงสัญญาณไฟจราจรในสภาวะนั้น จากนั้นจะเซตวอท์ชด็อก จากนั้นจะเข้าสู่เบสิกดีสเพล (Basic Display) คือเป็นการแสดงสภาวะพื้นฐานของเครื่อง จากนั้นจะไปอ่านคีย์บอร์ดแล้วเช็คสภาวะนั้น ว่าเป็นโปรแกรมออโตเมติก หรือว่าเป็นโปรแกรมแมนน่วล พร้อมกันนั้นจะเช็คสภาวะของหลอดด้วย จากนั้นจะเข้าสู่ลูป การอ่านค่าคีย์บอร์ด และรันวอท์ชด็อกฟังก์ชั่นเป็นอย่างนี้เรื่อยไป



รูปที่ 3.8 แสดงโฟลวชาร์ทแสดงการทำงานของอินนิเชี่ยลโปรแกรม (Initial Program)

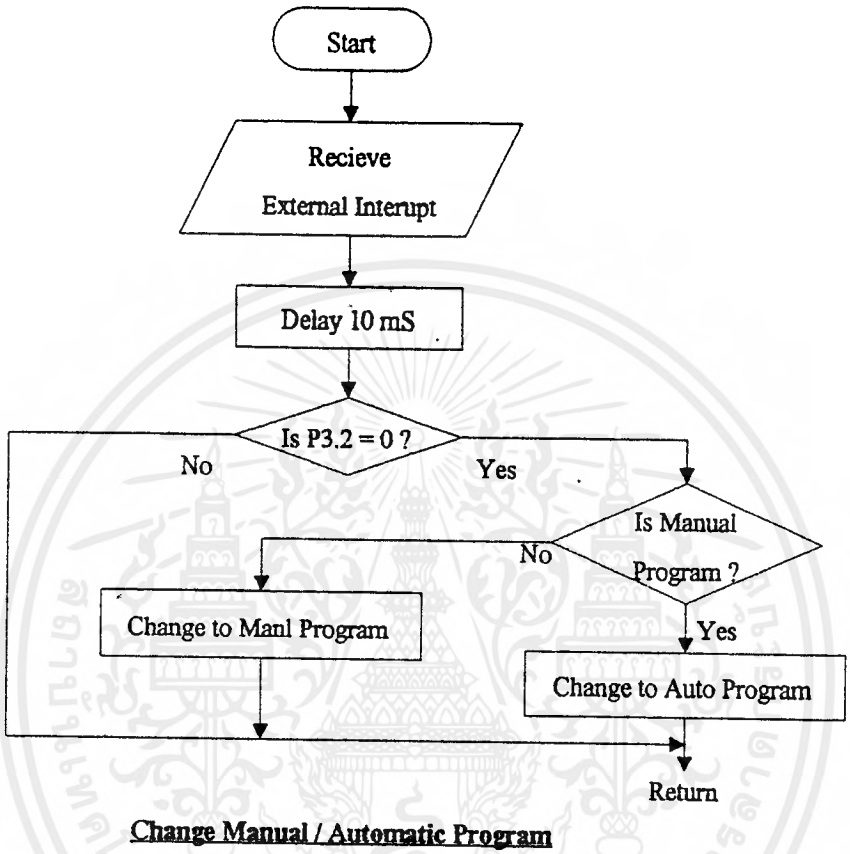
จากรูป จะเป็นการเซตพอร์ท 8255 และเซตค่าเริ่มต้นของจอแอลซีดี จากนั้นจะเซตค่าเริ่มต้น ได้แก่ การตั้งค่าอินเทอร์รัพ ระบบการสื่อสารอนุกรมและไทม์เมอร์



Check Automatic Data and Run Basic Display

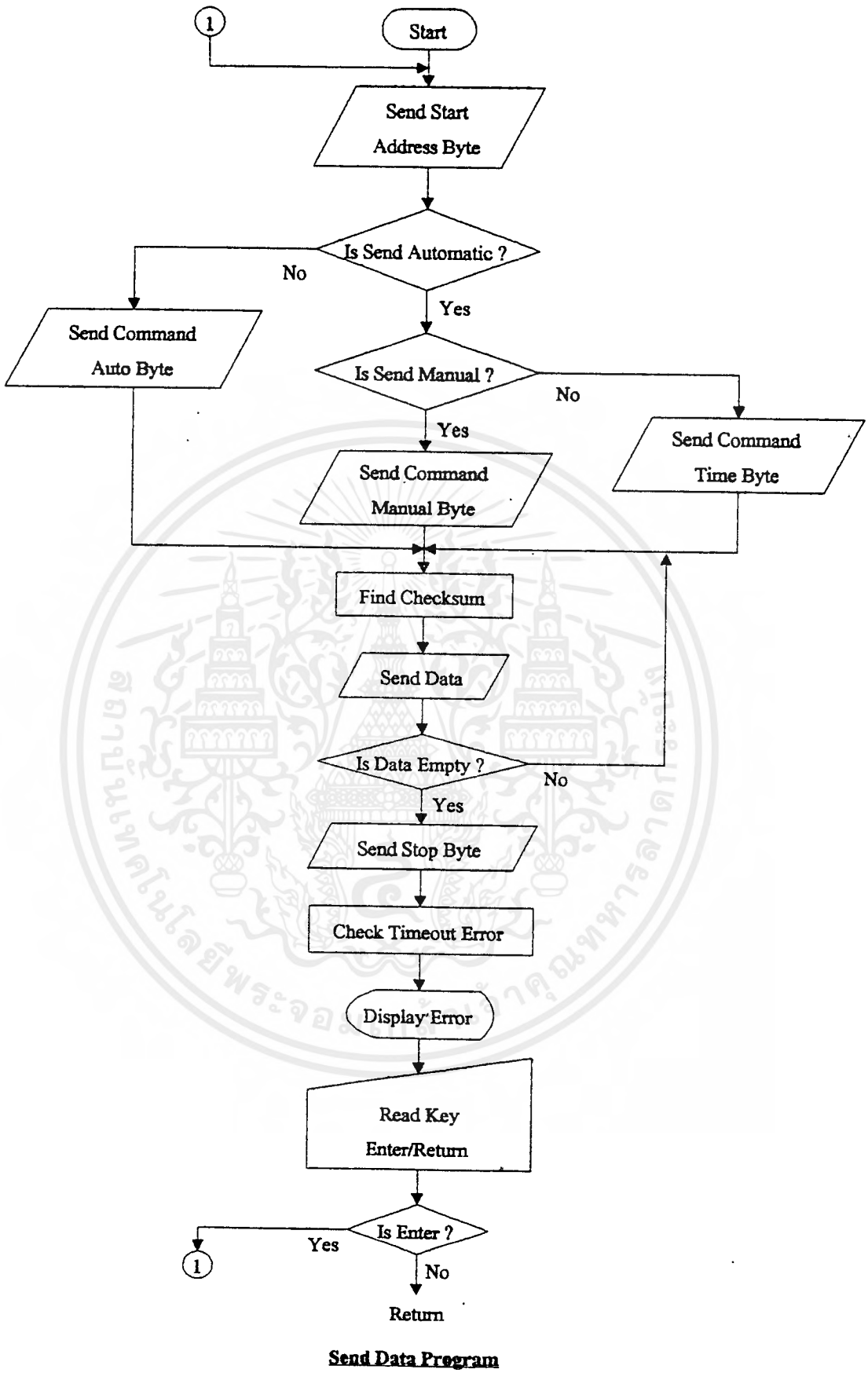
รูปที่ 3.9 แสดงไฟลวชาร์ทแสดงการทำงานการอ่านค่าเวลาและการตรวจสอบข้อมูลอัตโนมัติ

จากรูป จะทำการเช็คค่าไทม์เมอร์ 0 ที่ได้ตั้งไว้ เมื่อได้ตามเวลาที่กำหนดจะอ่านค่าเวลาจากอาร์ทีซี และทำการตรวจสอบโปรแกรมว่าเป็นโปรแกรมอัตโนมัติหรือไม่ ถ้าใช่จะทำการตรวจสอบข้อมูลที่ได้ตั้งเอาไว้ และตรวจสอบสภาวะของกระแสไฟด้วย ถ้ากระแสไฟตกลงจะทำการเก็บข้อมูลที่จะเป็นไว้ชั่วคราว เพื่อป้องกันข้อมูลที่จำเป็นจะสูญหายไป จากนั้นก็จะตรวจสอบเวลา เพื่อทำการอ้างอิงเวลา



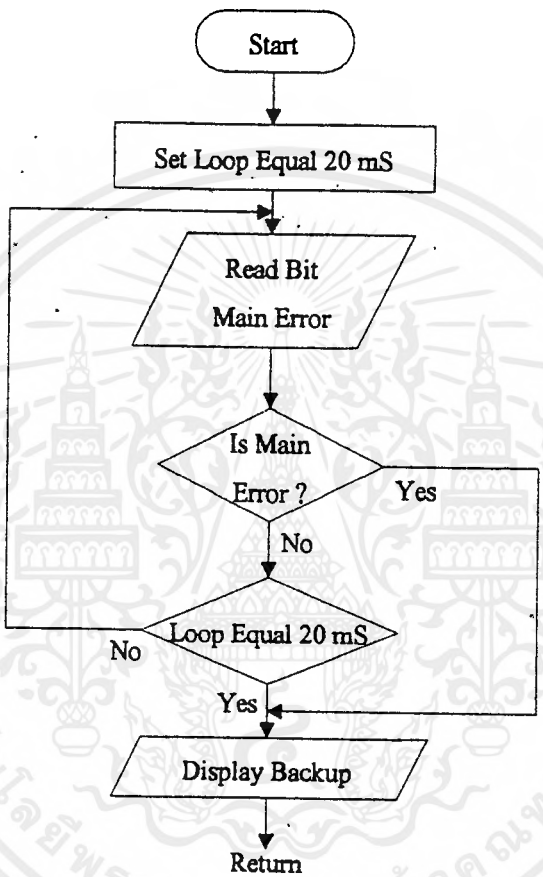
รูปที่ 3.10 แสดงไฟลวชาร์ทแสดงการทำงานของโปรแกรมการเปลี่ยน Manual - Automatic

จากรูป เครื่องจะตอบสนองต่อการอินเตอร์รัพจากภายนอก (External Interupt 0) แล้ว ทำการหน่วงเวลา เพื่อตรวจสอบว่าเป็นการอินเตอร์รัพจริงหรือไม่ ถ้ามีการอินเตอร์รัพ เครื่องจะเปลี่ยนสภาวะจากโปรแกรมที่เป็นอยู่ไปสู่อีกโปรแกรมหนึ่ง เช่น อยู่ในโปรแกรมแมนน่วล ก็จะเปลี่ยนเป็นโปรแกรมออโตเมติก แต่ถ้าอยู่ที่โปรแกรมออโตเมติก ก็จะเปลี่ยนเป็นโปรแกรมแมนน่วล



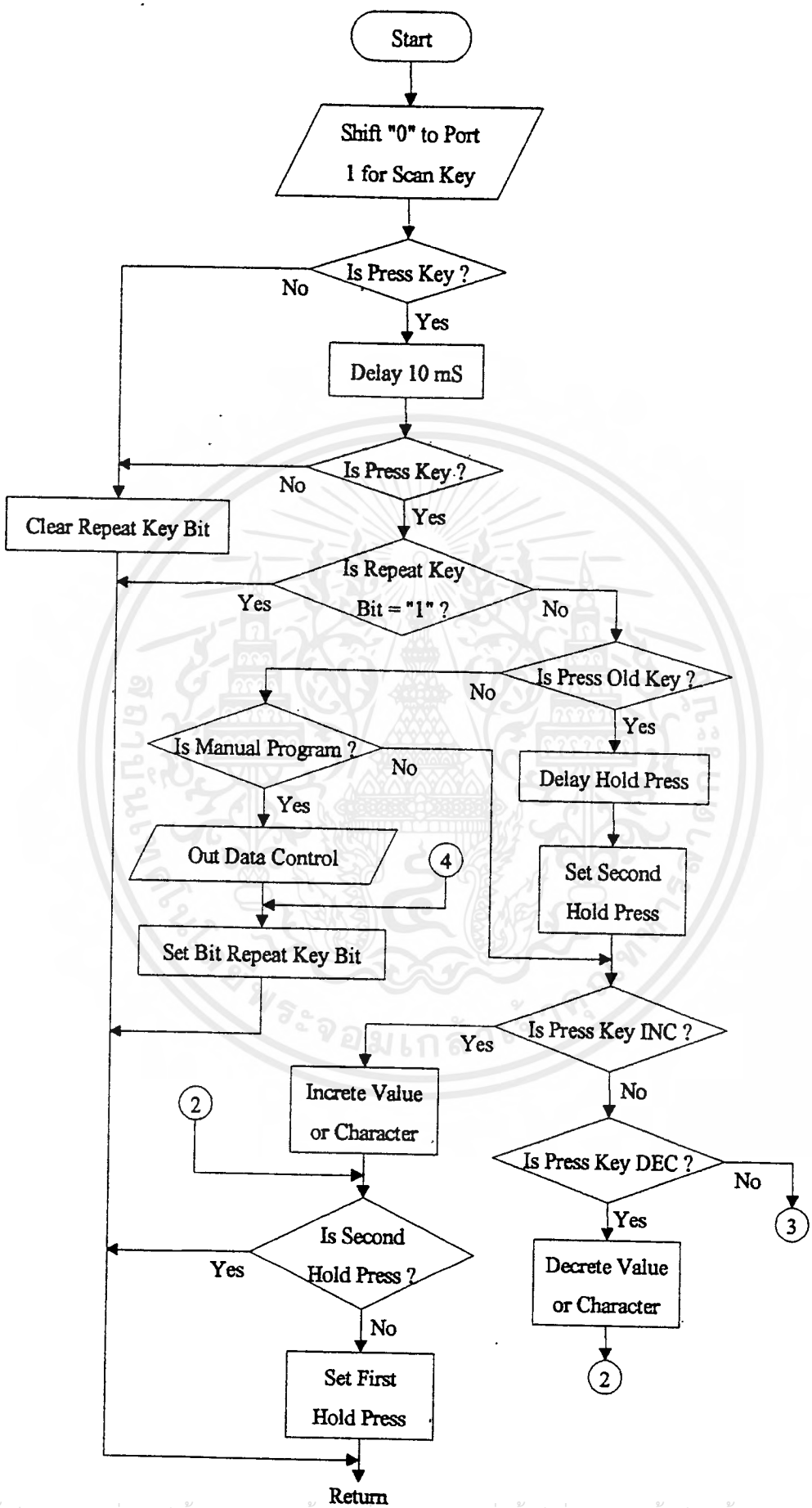
รูปที่ 3.11 แสดงไฟลวชาร์ทแสดงการทำงานของโปรแกรมการส่งข้อมูล

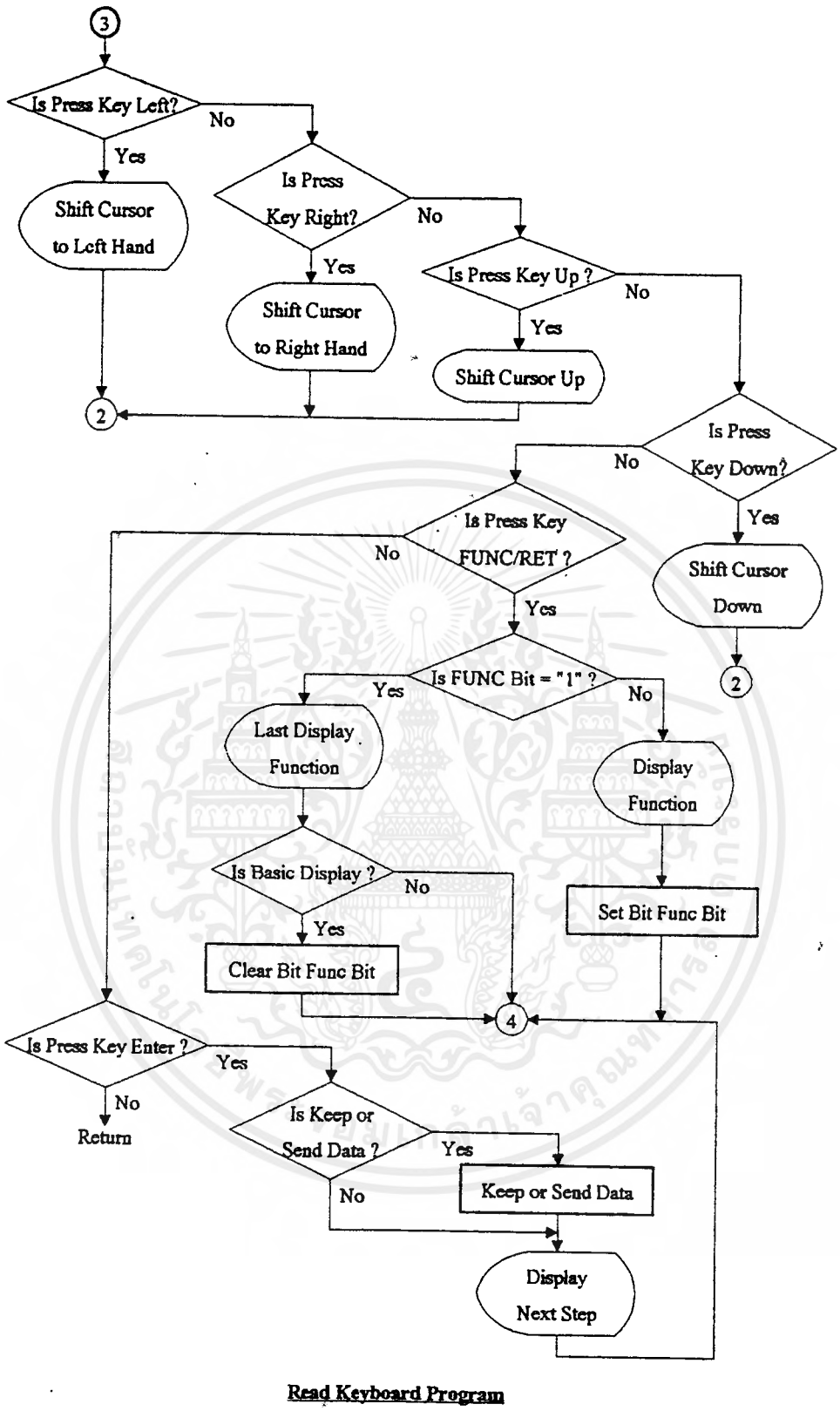
จากรูป จะทำการส่งตำแหน่งของตัวลูกออกไปก่อน จากนั้นจะตรวจสอบว่าเป็นข้อมูลชนิดใด เพื่อที่จะทำการส่งข้อมูลที่เป็นคำสั่งออกไปได้ถูกต้อง แล้วนำข้อมูลที่ต้องการจะส่งมาทำการหารหัส เพื่อใช้ในการตรวจสอบข้อมูลที่ต้องการส่งออกไป เพื่อให้ตัวรับสามารถจะรับข้อมูลได้อย่างถูกต้อง จากนั้นจะส่งข้อมูลที่บ่งบอกถึงการสิ้นสุดของข้อมูลที่ต้องการจะส่ง แล้วคอยรับสัญญาณตอบกลับจากตัวลูก พร้อมกับใช้ไทม์เอาท์เออร์เลอร์ (Timeout Error) เมื่อไม่ได้รับสัญญาณตอบรับจากตัวรับ และถึงเวลาไทม์เอาท์ เครื่องจะแสดงผลการส่งข้อมูลผิดพลาด แล้วจะคอยรับคำสั่งว่าต้องการให้ส่งข้อมูลใหม่หรือไม่



รูปที่ 3.12 แสดงไฟลวชาร์ทแสดงการทำงานของโปรแกรมตรวจสอบการทำงานของส่วนแสดงผล

จากรูป จะทำการอ่านค่าของข้อมูลที่ทำการตรวจสอบหลอดไฟฟ้า โดยจะทำการอ่านค่าเป็นค่าเวลาสูงสุดเท่ากับ 20 mS หรือเท่ากับความถี่ 50 Hz ของสัญญาณไฟฟ้ากระแสสลับ ซึ่งค่าเวลาที่ตรวจสอบหลอดไฟฟ้าเป็นช่วงเวลาที่มากพอที่จะพบหลอดไฟฟ้าที่ชำรุดได้อย่างแน่นอน จากนั้นเมื่ออ่านค่าของหลอดไฟฟ้าที่ชำรุดได้แล้วจะทำการขับหลอดสำรองที่เตรียมไว้ขึ้นมาทันที



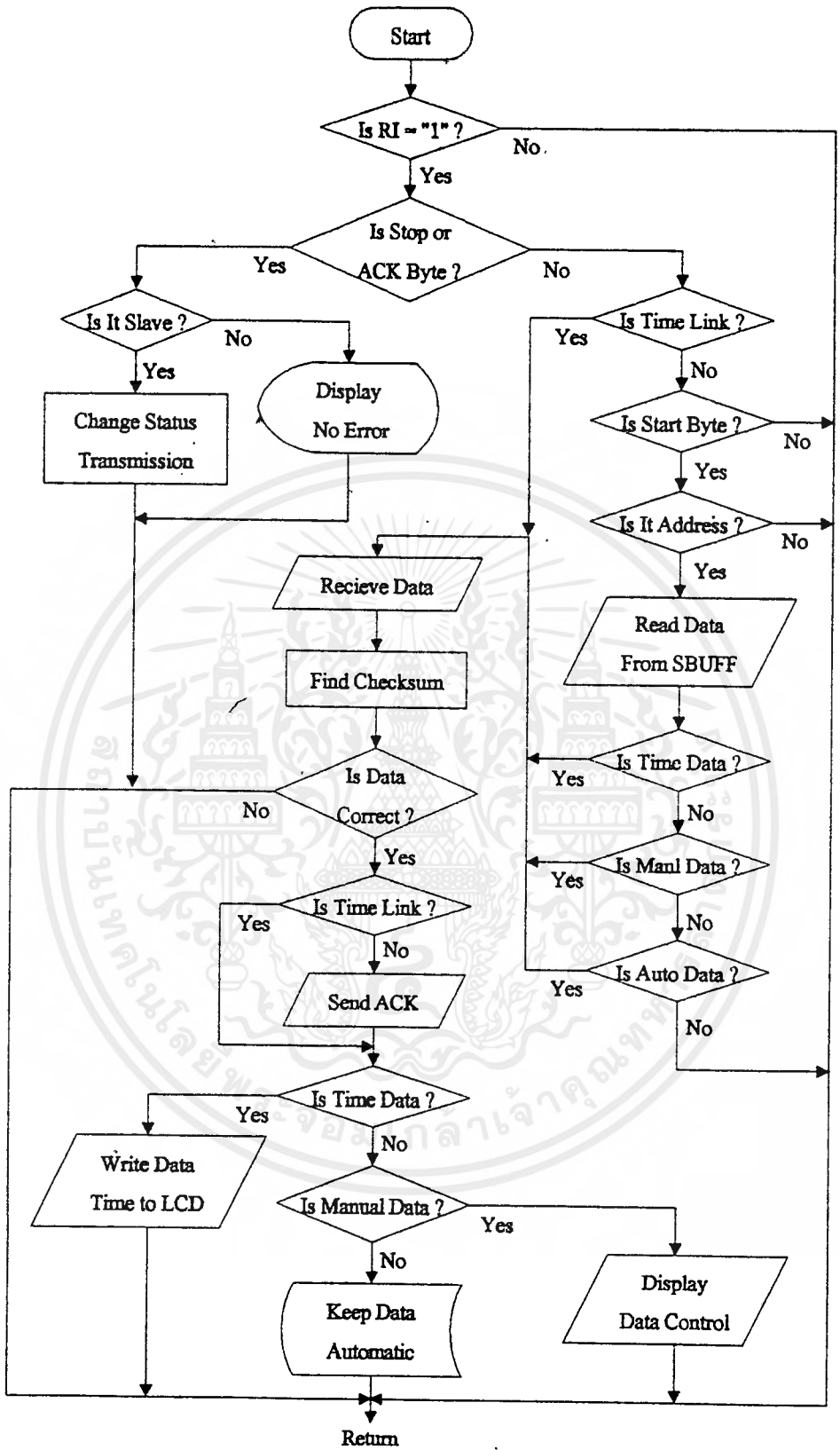


Read Keyboard Program

รูปที่ 3.13 แสดงโฟลวชาร์ทแสดงการทำงานของโปรแกรมอ่านค่าคีย์บอร์ด

จากรูปจะทำการเลื่อนลอจิกเข้าสู่พอร์ทหนึ่ง เพื่อใช้ในการอ่านค่าของคีย์บอร์ด โดยที่ ถ้ามีการกดคีย์จะทำหน่วงเวลาประมาณ 10 mS เพื่อตรวจสอบว่า มีการกดคีย์บอร์ดจริงหรือไม่ ถ้าไม่มีการกดคีย์บอร์ด เครื่องจะกลับเข้าสู่ลูป การอ่านค่าคีย์บอร์ดต่อไป ถ้ามีการกดคีย์จริง ก็จะตรวจสอบต่อไปว่าเป็นการกดคีย์ค้างหรือไม่ โดยถ้าอยู่ในโปรแกรมแมนน่วลก็จะตอบสนองสำหรับการกดคีย์ครั้งใหม่เท่านั้น แต่ถ้าอยู่ในโปรแกรมออโตเมติกจะตอบสนองทั้งสองอย่าง ถ้ามีการกดคีย์ค้างก็จะหน่วงเวลาไว้ค่าๆหนึ่งที่เหมาะสมแล้ว จึงทำงานตามรหัสค่าคีย์บอร์ดนั้นที่อ่านค่าเข้ามาได้ แต่ถ้าไม่ได้เป็นการกดคีย์ค้างก็จะทำงานตามรหัสค่าคีย์บอร์ดนั้นที่อ่านค่าเข้ามาได้นั้นทันที ซึ่งการทำงานตามรหัสต่างๆ นั้นแบ่งออกได้เป็น 8 หน้าที่ด้วยกัน ดังนี้

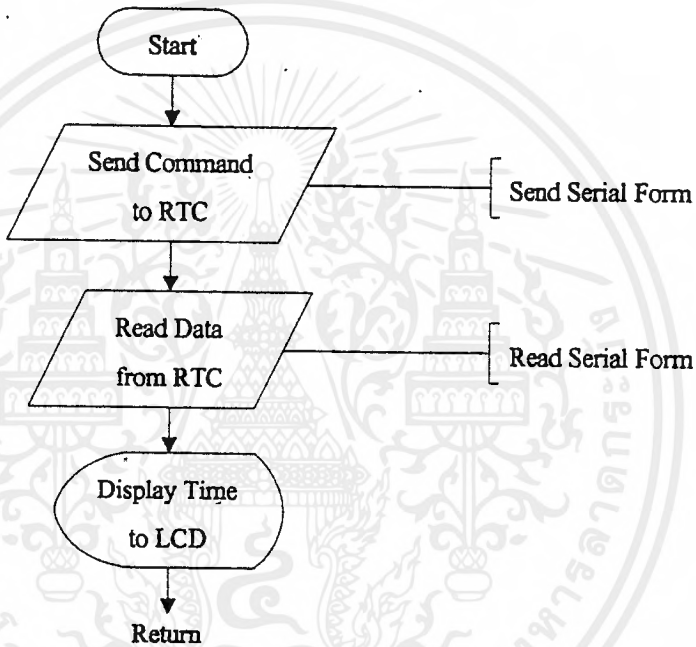
- INC (Increase) เป็นการเพิ่มค่าของรหัส หรืออักขระขึ้น หนึ่งค่า หรือหนึ่งตัวอักษร เช่น จาก 1 เป็น 2 , จาก A เป็น B
- DEC (Decrease) เป็นการลดค่าของรหัส หรืออักขระขึ้นหนึ่งค่า หรือหนึ่งตัวอักษร เช่น จาก 2 เป็น 1 , จาก B เป็น A
- LEFT เป็นการเลื่อนเคอร์เซอร์ไปทางซ้ายหนึ่งตำแหน่ง หรือไปยังตำแหน่งที่กำหนดไว้ทางด้านซ้ายมือ
- RIGTH เป็นการเลื่อนเคอร์เซอร์ไปทางขวาหนึ่งตำแหน่ง หรือไปยังตำแหน่งที่กำหนดไว้ทางด้านขวามือ
- UP เป็นการเลื่อนเคอร์เซอร์ขึ้นไปทางด้านบนหนึ่งบรรทัด หรือหนึ่งตำแหน่ง
- DOWN เป็นการเลื่อนเคอร์เซอร์ลงไปทางด้านล่างหนึ่งบรรทัด หรือหนึ่งตำแหน่ง
- FUNC/RET (Function/Return) เป็นการเข้าสู่เมนูฟังก์ชั่นเซต เพื่อทำการกำหนดค่าต่างๆ และการกลับออกจากเมนูของฟังก์ชั่นต่างๆ รวมถึงการกลับสู่เบสิกดีสเพล
- ENT (Enter) เป็นการเข้าไปเซตฟังก์ชั่นรวมถึงการนำข้อมูลเข้าไปเก็บและการส่งข้อมูลไปให้ตัวลูกด้วย



Recieve Data Program

รูปที่ 3.14 แสดงโฟลวชาร์ทแสดงการทำงานของโปรแกรมรับข้อมูล

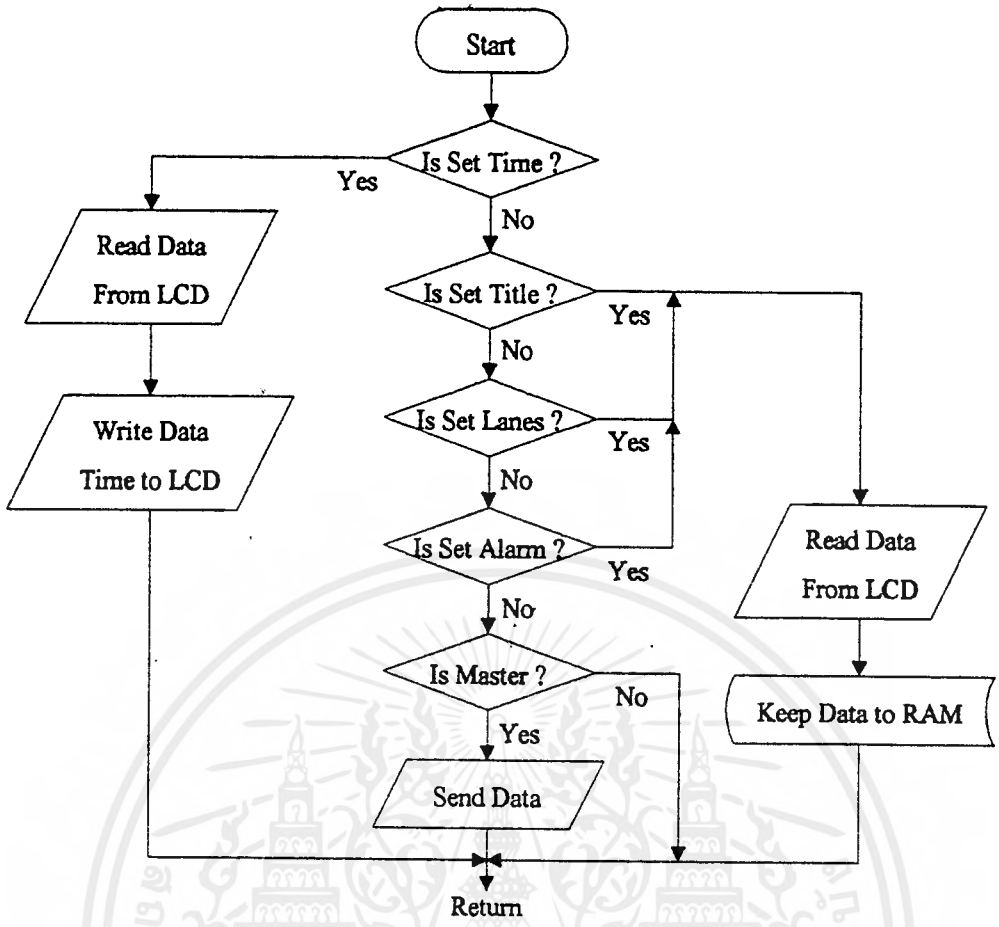
จากรูป เป็นการรับข้อมูลที่ถูกส่งมาจากตัวส่ง จากนั้นจะทำการตรวจสอบว่ามันเป็นข้อมูลของการตอบรับ หรือการสิ้นสุดของข้อมูลหรือไม่ ถ้าใช่จะทำการตรวจสอบว่าสภาพของตัวรับเป็นตัวแม่ หรือตัวลูก ถ้าเป็นตัวลูกจะทำการเปลี่ยนสภาวะของตัวส่งนั้น เพื่อให้ข้อมูลสามารถจะเคลื่อนที่ได้ทั้งไปและกลับ แต่ถ้าเป็นตัวแม่แสดงว่า ตัวส่งนั้นเป็นตัวลูก และได้ส่งสัญญาณตอบรับกลับมา ก็แสดงว่าตัวลูกนั้นรับข้อมูลได้ถูกต้อง แต่ถ้าไม่ใช่ข้อมูลตอบรับ หรือข้อมูลสิ้นสุด จะทำการรับข้อมูลที่ได้ส่งมา โดยจะตรวจสอบตำแหน่งของตัวเองก่อนว่าถูกต้องหรือไม่ และตรวจสอบว่าคำสั่งของข้อมูลนั้นคืออะไรด้วย ถ้าข้อมูลนั้นเป็นข้อมูลของเวลาจะทำการตั้งค่าเวลาของผู้รับนั้นทันที ถ้าเป็นแมนน่วลดาต้า (Manual Data) จะทำการตั้งสภาวะของช่องการจราจรในขณะนั้น แต่ถ้าเป็นออโตเมติกดาต้า (Automatic Data) ก็จะเอาข้อมูลไปเก็บไว้ในหน่วยความจำ เพื่อใช้ในโปรแกรมออโตเมติกต่อไป



รูปที่ 3.15 แสดงไฟลวชาร์ทแสดงการทำงานของโปรแกรมอ่านค่าเวลาจากอาร์ทีซีชิพ (Real Time Clock Chip)

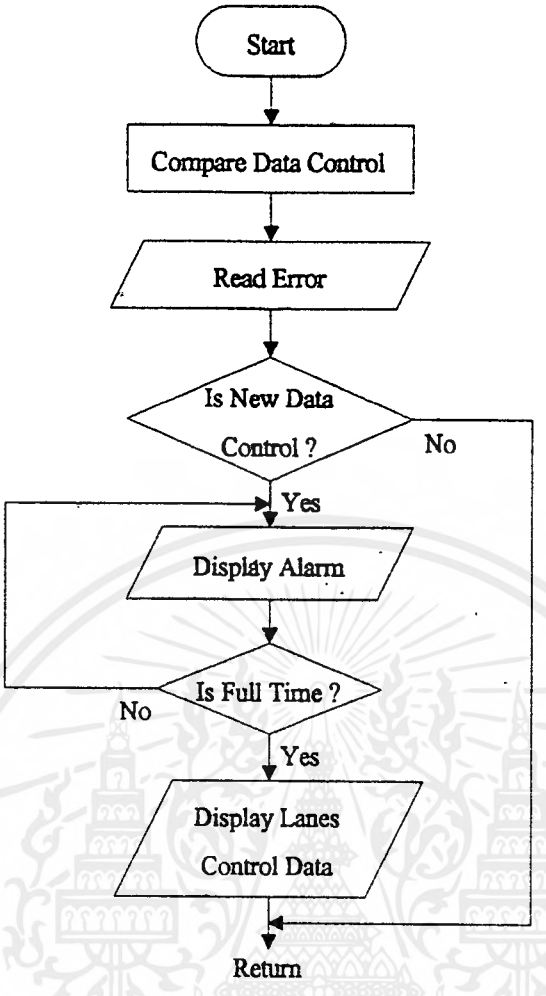
จากรูปจะทำการส่งคำสั่งที่ต้องการอ่านข้อมูลจากอาร์ทีซีไปยังอาร์ทีซี จากนั้นอาร์ทีซีก็จะส่งค่าข้อมูลที่ต้องการทราบออกมา โดยจะส่งออกมาเป็นรหัสบีซีดี (BCD Code) ซึ่งข้อมูลทั้งคำสั่งในการอ่านค่า และข้อมูลของเวลาที่ส่งออกมานั้นจะเป็นแบบข้อมูลอนุกรม เมื่ออ่านค่าได้แล้วก็จะแสดงผลที่จอแอลซีดี

การเขียนข้อมูลไปยังอาร์ทีซีก็ทำด้วยวิธีที่คล้ายกับการอ่านค่าข้อมูลเช่นกัน



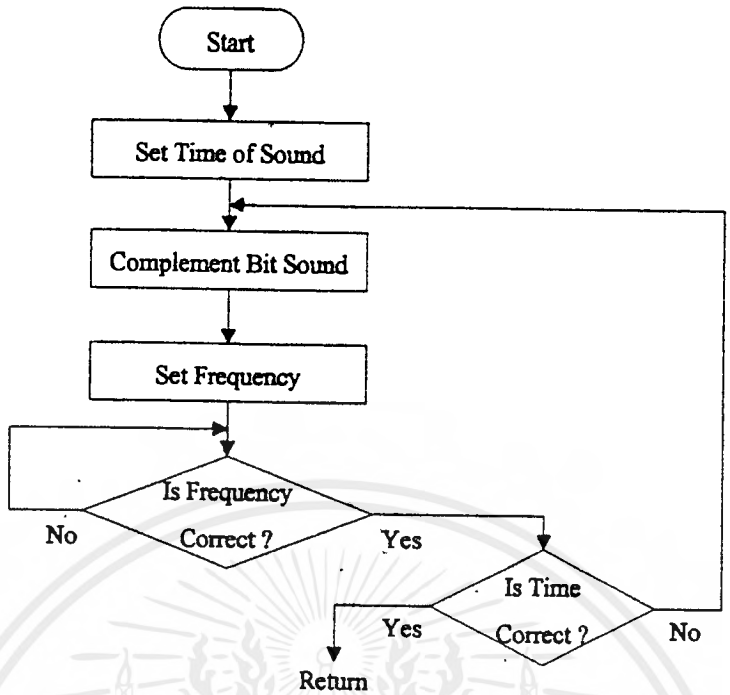
รูปที่ 3.16 แสดงโฟลวชาร์ทแสดงการทำงานของโปรแกรมเลือกเก็บข้อมูล หรือส่งข้อมูล

จากรูป แสดงถึงการกดคีย์เอ็นเทอร์ที่อยู่ภายในฟังก์ชั่นต่างๆ ซึ่งถ้าเป็นฟังก์ชั่นตั้งเวลาก็จะไปตั้งค่าที่อาร์ทีซีทันที ถ้าเป็นฟังก์ชั่นตั้งชื่อถนน ตั้งโปรแกรมอัตโนมัติหรือตั้งค่าเวลาในการกระพริบ ก็จะเอาข้อมูลนั้นไปเก็บไว้ภายในหน่วยความจำเพื่อใช้ต่อไป แต่ถ้าเป็นฟังก์ชั่นการส่งข้อมูลของตัวส่งก็จะทำการทำการส่งข้อมูลที่ต้องการนั้นไปยังตัวรับ



รูปที่ 3.17 แสดงโฟลวชาร์ทแสดงการทำงานของโปรแกรมการแสดงสัญญาณไฟจราจร

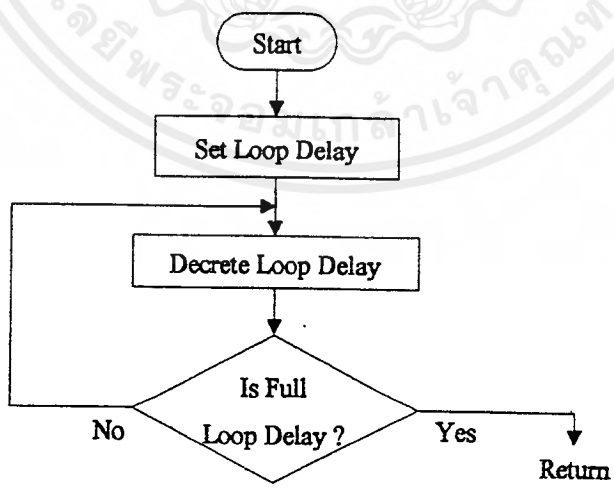
จากรูปเป็นการเปรียบเทียบข้อมูลในแต่ละช่องการจราจรระหว่างข้อมูลในปัจจุบันและข้อมูลที่ได้รับเข้ามาใหม่ ถ้าข้อมูลทั้งสองแตกต่างกัน แสดงว่ามีการสลับช่องจราจรเกิดขึ้นช่องการจราจรที่เกิดการเปลี่ยนสภาวะนั้นจะกระพริบ เพื่อบอกการเปลี่ยนสถานะนั้น ด้วยระยะเวลาตามที่ได้กำหนดไว้ เมื่อครบตามกำหนดเวลาแล้วก็จะแสดงสภาวะของช่องการจราจรที่ได้รับข้อมูลเข้ามาใหม่



Generate Sound Program

รูปที่ 3.18 แสดงโฟลวชาร์ทแสดงการทำงานของโปรแกรมกำเนิดเสียง

จากรูป จะทำการตั้งระยะเวลาของเสียงแล้วทำการสร้างเสียงโดยการใช้การเปลี่ยนแปลงของสัญญาณพัลซ์ซึ่งมีค่าของคาบเวลาแปรผกผันกับความถี่ที่ต้องการ



Delay Program

รูปที่ 3.19 แสดงโฟลวชาร์ทแสดงการทำงานโปรแกรมดีเลย์ (Delay Program)

การทำงานของโปรแกรมดีเลย์จะใช้การลดค่าลงของรีจิสเตอร์ เพื่อใช้ในการหน่วงการทำงานของไมโครคอนโทรลเลอร์

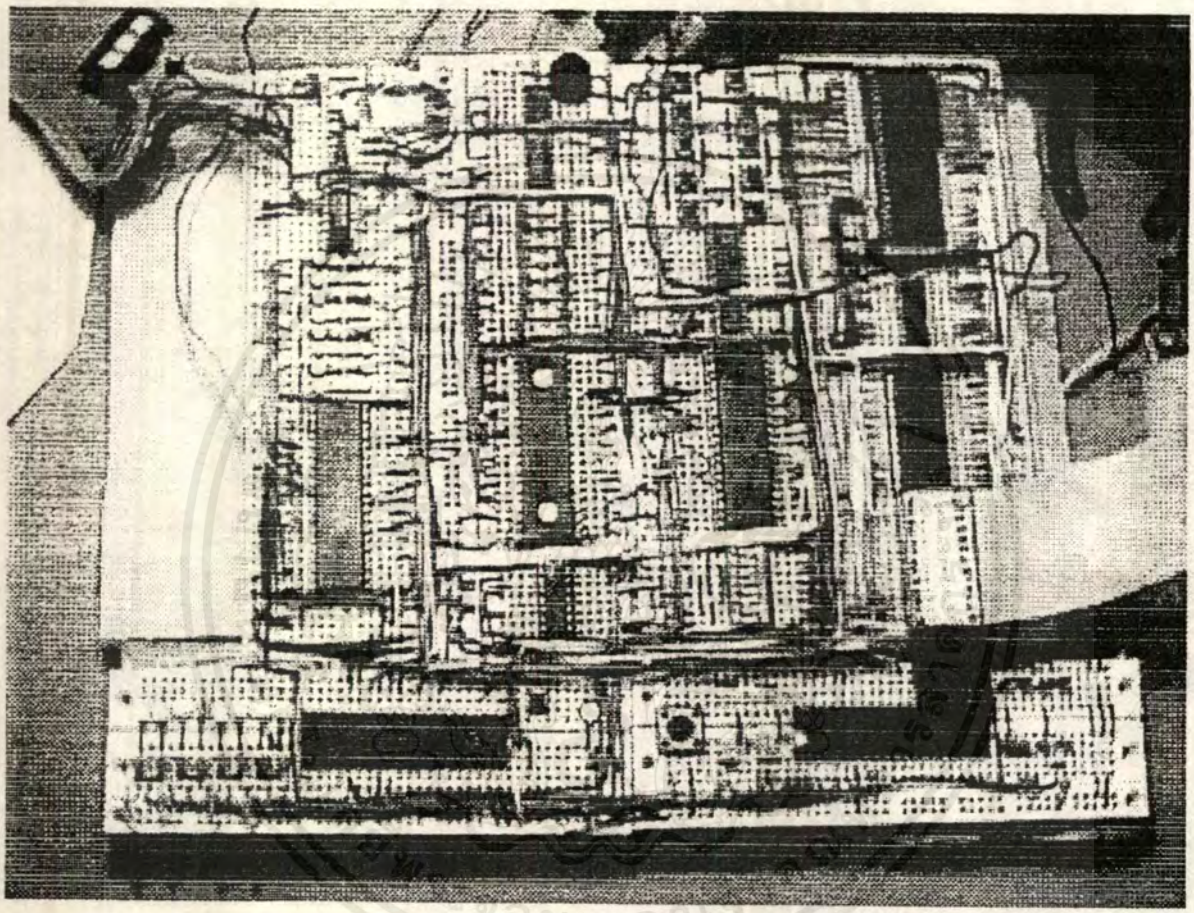
3.4 ขั้นตอนการออกแบบลายวงจร

หลังจากที่ทำการทดลองจะสำเร็จสมบูรณ์ทุกส่วนขั้นตอนสุดท้ายของการทำโครงงานนี้ก็คือการออกแบบลายวงจรซึ่งเป็นขั้นตอนที่ยุ่งยาก โดยผู้จัดทำได้ใช้ซอฟแวร์ โพรเทล (Protel) .1.5 ฟอร์วินโดว์ (1.5 for Window) ในการออกแบบลายวงจรซึ่งแสดงให้เห็นลายวงจรที่ออกแบบซึ่งจะมีส่วนที่เป็นลายทองแดงและส่วนที่มีลายทองแดงรวมอยู่กับเลย์เอาท์ (Lay Out) อุปกรณ์ซึ่งแสดงไว้ในตอนท้ายของเล่ม

บทที่ 4. การทดลอง และผลการทดลอง

4.1 ทดลองการทำงานบอร์ดคอนโทรล (Control Board)

ผลการทดลองของผู้สร้างได้ทำการทดลองต่างๆ ในโปรโตบอร์ด (Proto Board) และทดลองรัน การทำงานของซีพียู (CPU) โดยการเขียนโปรแกรมในคอมพิวเตอร์ และใช้โปรแกรม SXA-51 คอมไพล์ แล้วทำการกอบปี้โปรแกรมผ่านพอร์ทอนุกรมไปที่ EEPROM เพื่อทดลองรันโปรแกรม ซึ่งบอร์ดที่ต่อมีลักษณะดังรูป



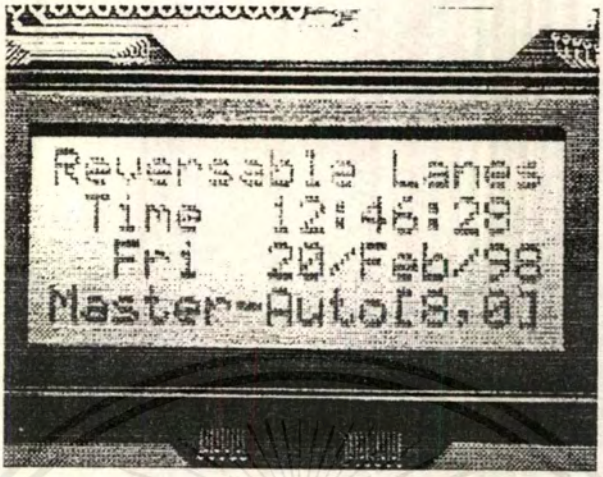
รูปที่ 4.1 แสดงบอร์ดที่ใช้ในการทดลอง

4.2 ทดลองการทำงาน และผลการทดลองของคีย์บอร์ด และแอลซีดี

เมื่อเขียนโปรแกรมในส่วนของการสแกนคีย์ (Scan Key) และการสั่งงานจอแอลซีดีได้แล้วเราก็สามารถที่เขียนฟังก์ชั่นต่างๆที่เราต้องการจะใช้ในการควบคุมการทำงานต่างๆได้ซึ่งจะแสดงให้เห็นในผลการทดลองการ

ผลการทดลองที่เกิดขึ้นในการสั่งงานจากคีย์บอร์ด สามารถที่จะแสดงสภาวะการทำงานต่างๆได้คือ

●เมื่อตอนที่เปิดเครื่องออกมาเป็นครั้งแรกจะเห็นหน้าจอแอลซีดีแสดงดังรูป

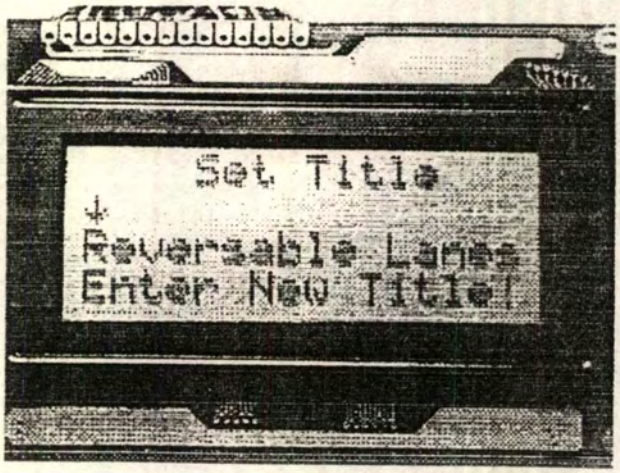


รูปที่ 4.2 แสดงเบสิกดิสเพลย์

●โดยบรรทัดแรกของหน้าจอสามารถที่จะโปรแกรมเป็นชื่อต่างๆตามถนนที่นำไปติดตั้งได้ แต่ในขณะนี้ตั้งค่าให้ออกหน้าจอว่า " Reversable lanes "

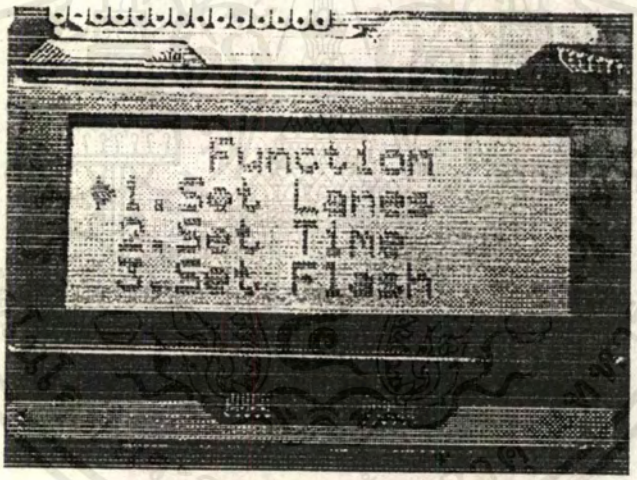
รูปที่ 4.3 แสดงการเลือกฟังก์ชั่นการตั้งชื่อถนน

● หน้าจอนี้แสดงฟังก์ชั่นในการตั้งชื่อถนน โดยใช้เคอร์เซอร์ชี้ที่ฟังก์ชั่นนั้น แล้วกดเอ็นเตอร์



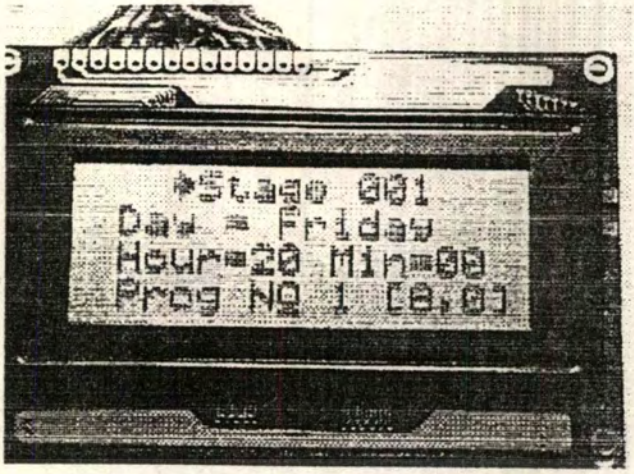
รูปที่ 4.4 แสดงการตั้งชื่อถนน

- เริ่มต้นเซตตัวอักษรตัวแรกโดยการให้เคอร์เซอร์ชี้ที่ตำแหน่งที่ตัวอักษรนั้นแล้วกด INC หรือ DEC เพื่อเปลี่ยนตัวอักษร



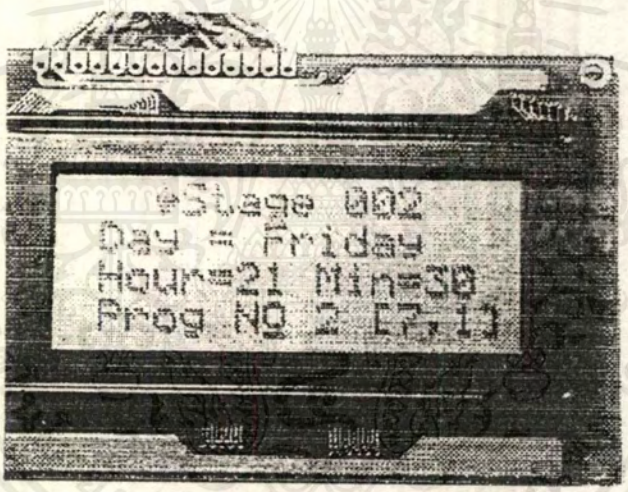
รูปที่ 4.5 แสดงการเลือกฟังก์ชั่นการตั้งโปรแกรมอัตโนมัติ

- เป็นฟังก์ชั่นในการเซตโปรแกรมให้การสลับช่องการจราจรเป็นแบบอัตโนมัติ



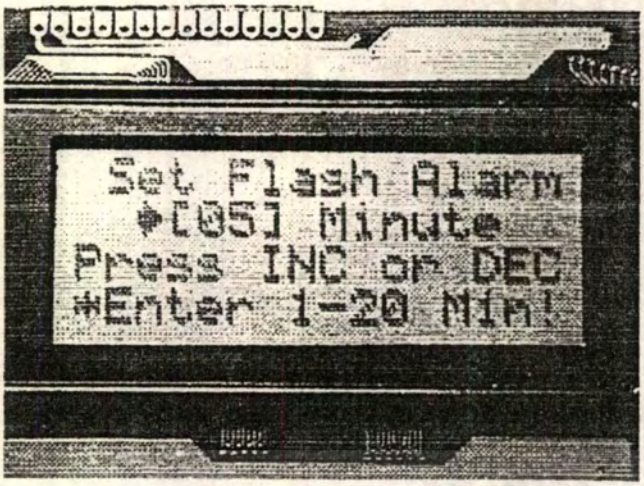
รูปที่ 4.6 แสดงการตั้งโปรแกรมอัตโนมัติโปรแกรมที่ 1

- เป็นการตั้งค่าโปรแกรมที่ 1 โดยเลือก วัน-เวลาและรูปแบบการสลับช่องทางจราจร



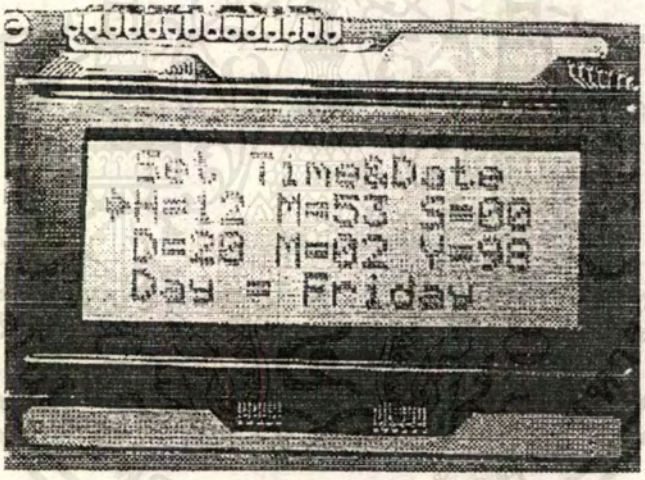
รูปที่ 4.7 แสดงการตั้งโปรแกรมอัตโนมัติโปรแกรมที่ 2

- เป็นการตั้งค่าโปรแกรมที่ 2 โดยเลือก วัน-เวลาและรูปแบบการสลับช่องทางจราจร



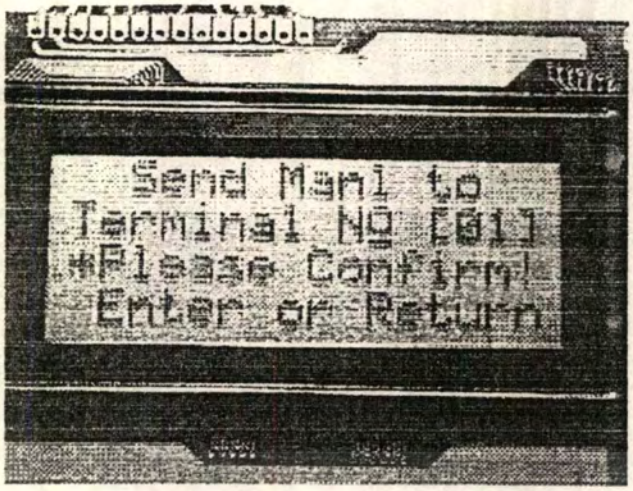
รูปที่ 4.8 แสดงการตั้งค่าเวลาของการกระพริบ เมื่อมีการเปลี่ยนช่องการจราจร

- เป็นฟังก์ชั่นในการเซตเวลาในการกระพริบสัญญาณไฟก่อนที่จะที่การสลับช่องการจราจร



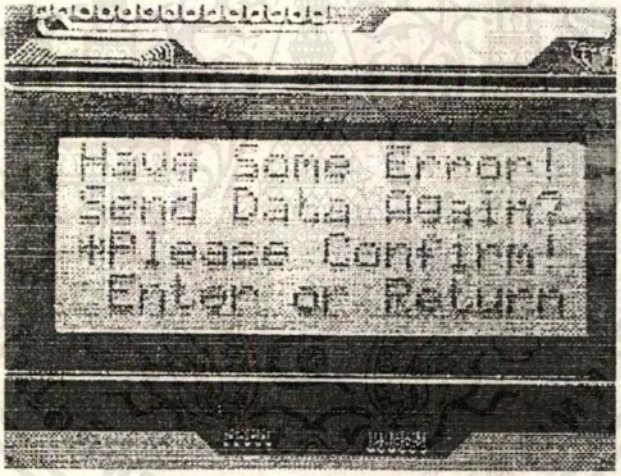
รูปที่ 4.9 แสดงการตั้งเวลาจริงของเครื่อง

- เป็นฟังก์ชั่นในการเซตเวลาที่บอร์ดคอนโทรล



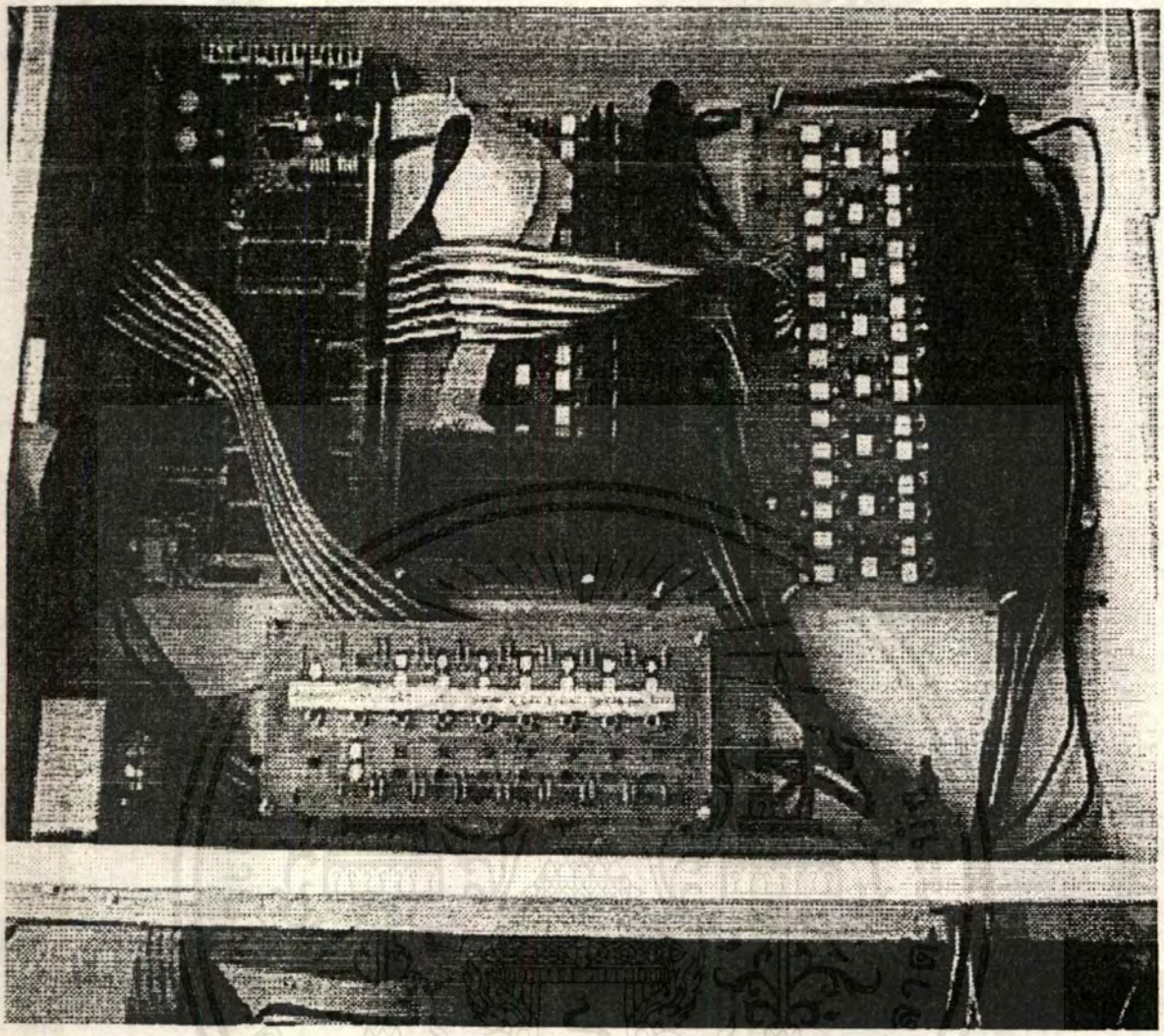
รูปที่ 4.10 แสดงการยืนยัน เพื่อที่จะส่งข้อมูล

- เป็นการส่งโปรแกรมที่เซตจากตัวมาสเตอร์ไปยังตัวสเลฟตำแหน่งต่างๆ

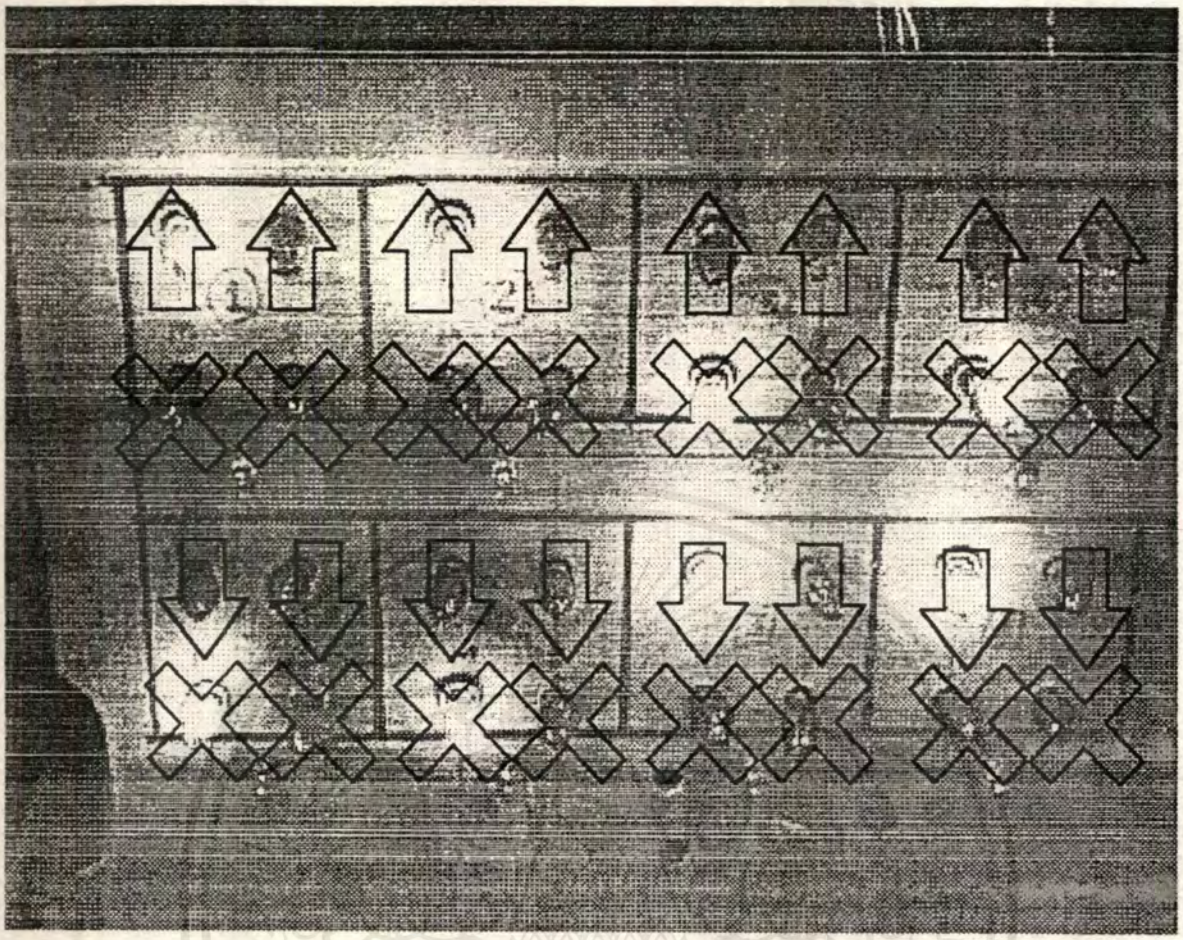


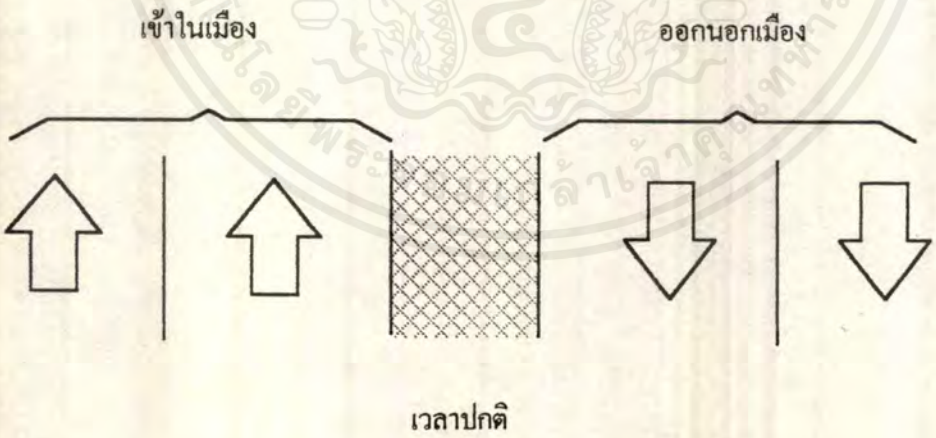
รูปที่ 4.11 แสดงความผิดพลาดของการส่งข้อมูล และยืนยันการส่งข้อมูลอีกครั้ง

- เป็นการเกิดเออร์เลอร์ (Error) ที่เกิดจากการส่งข้อมูลไปยังที่ตัวสเลฟ และสเลฟไม่ตอบ ACK ให้ตัวมาสเตอร์

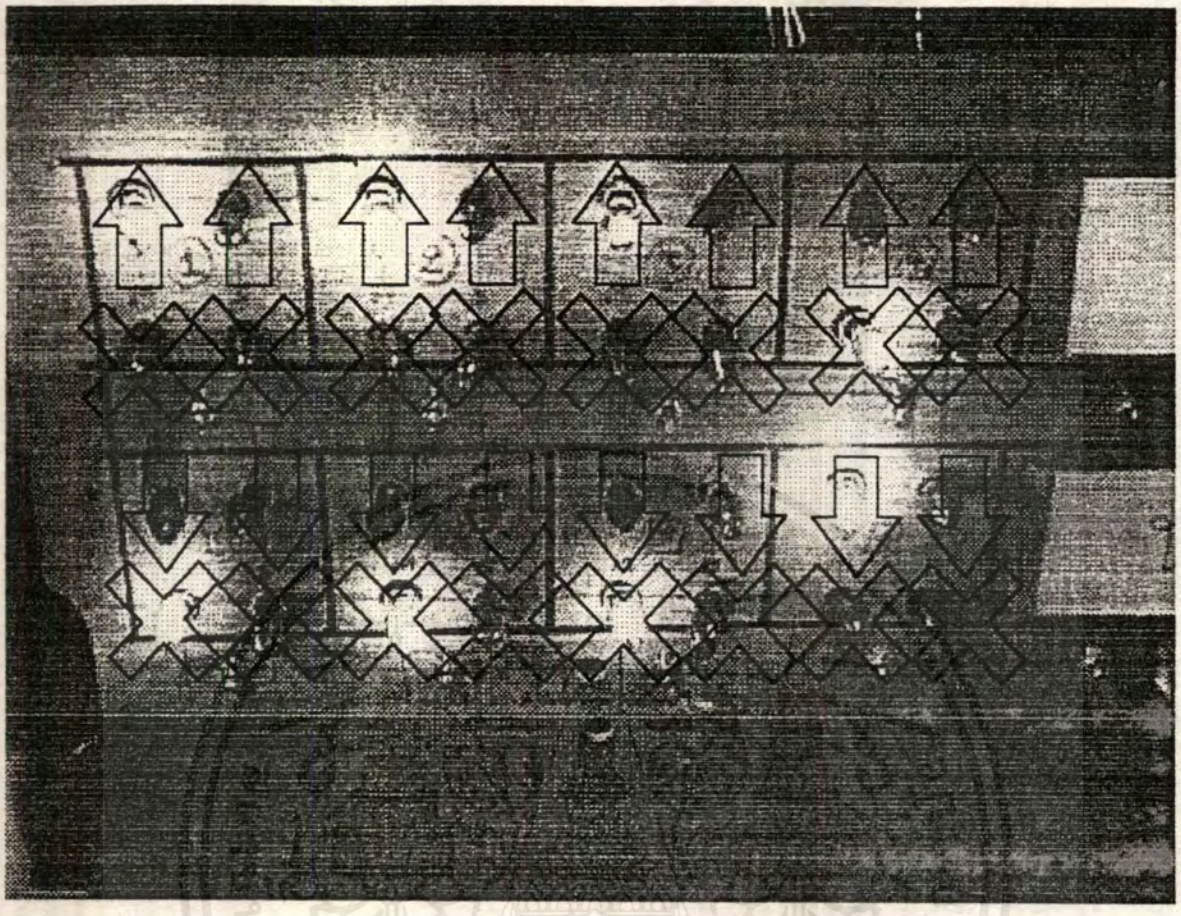


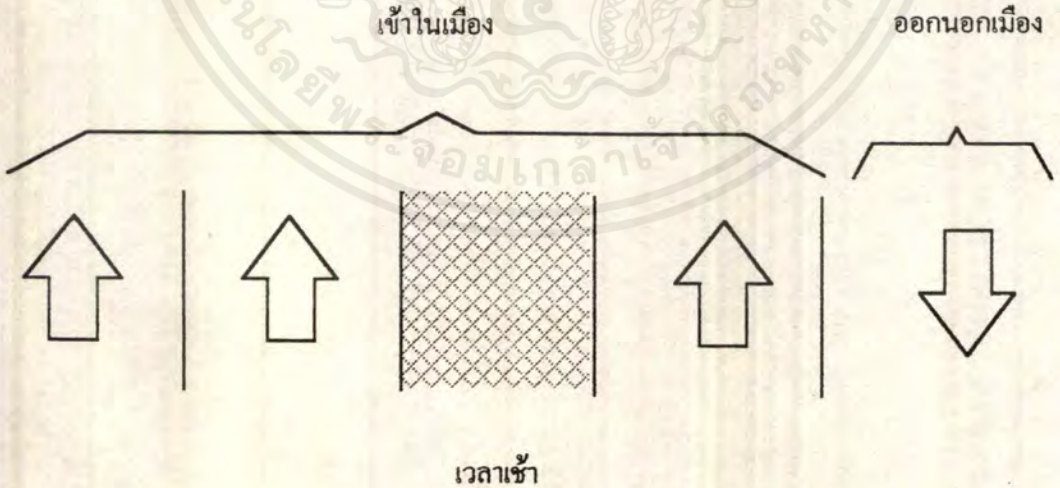
รูปที่ 4.12 แสดงเครื่องสลับช่องการจราจรที่ประกอบลงบนแผ่นวงจรพิมพ์ และลงกล่องเรียบร้อยแล้ว



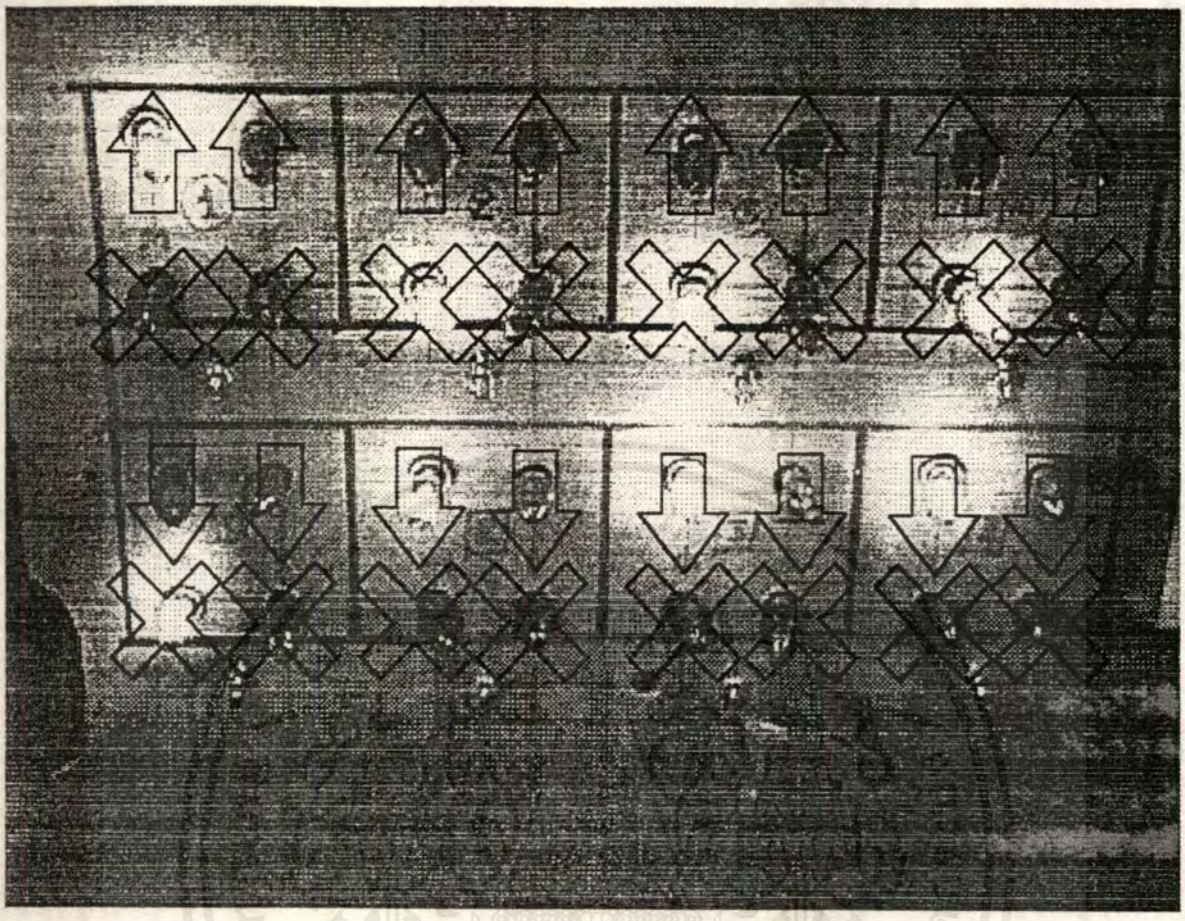


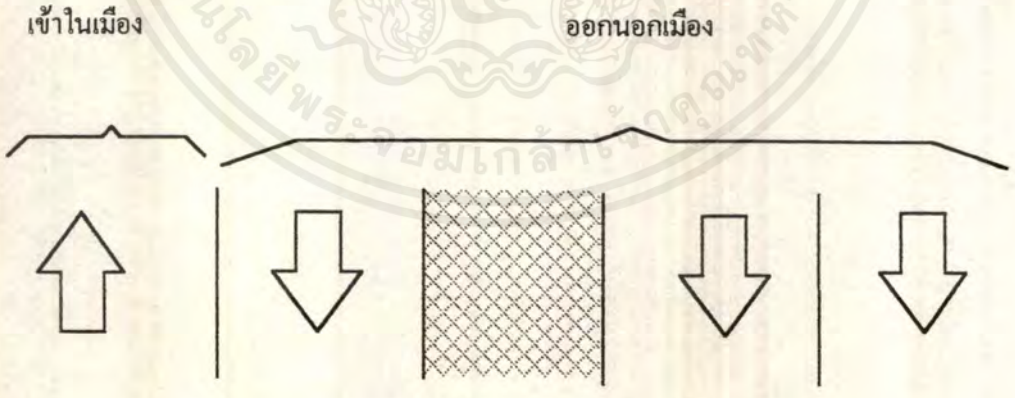
รูปที่ 4.13 แสดงผลการทดลองสัญญาณไฟจราจร เมื่อมีการจัดช่องการจราจรตามปกติ



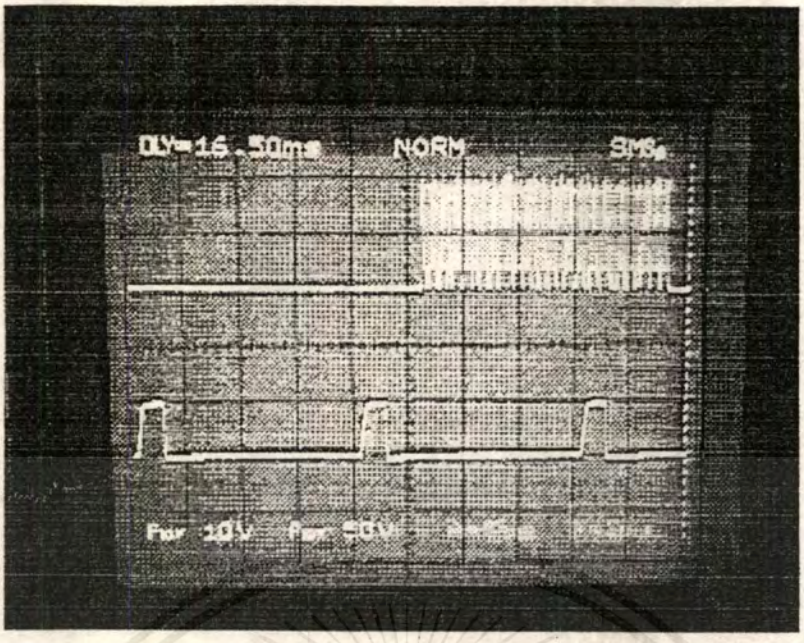


รูปที่ 4.14 แสดงผลการทดลองสัญญาณไฟจราจร เมื่อมีการสลับช่องการจราจรในช่องเวลาเช้า



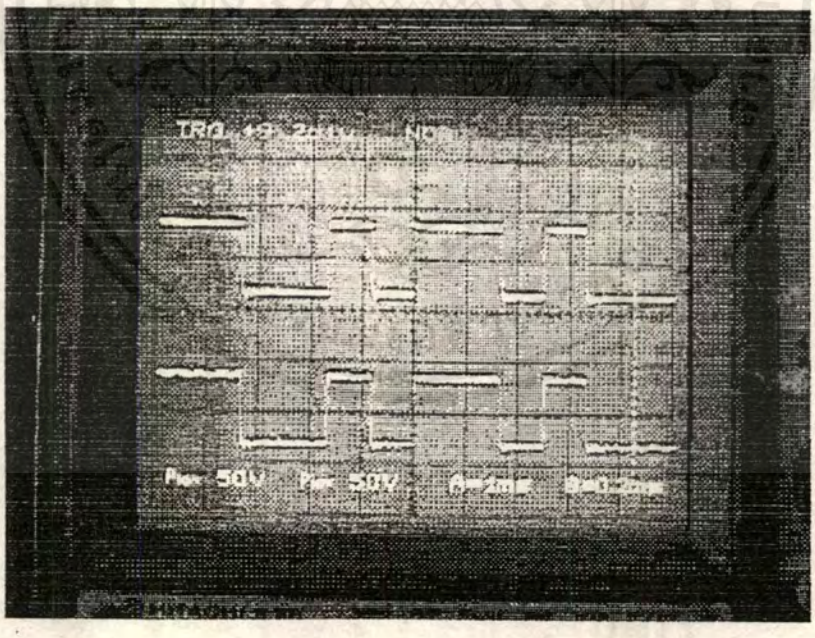


รูปที่ 4.15 แสดงผลการทดลองสัญญาณไฟจราจร เมื่อมีการสลับช่องการจราจรในช่องเวลาเย็น



รูปที่ 4.16 แสดงสัญญาณในตรวจสอบการเสียหายของหลอดไฟฟ้า

- เป็นภาพแสดงการดีเทค (Detection) สัญญาณเพื่อแสดงว่าขณะนี้หลอดดีหรือเสียอยู่ ซึ่งถ้าหากเกิดหลอดเสียสัญญาณพัลซ์ด้านล่างจะไม่มี



รูปที่ 4.1 แสดงสัญญาณของข้อมูลที่ส่งออกไป

- เป็นภาพแสดงสัญญาณที่ส่งจากต้นทางไปถึงปลายทางว่ามีการสูญเสียเท่าไหร่ จากการวัดที่ระยะทางระหว่าง Master-Slave ห่างกัน 500 m ความเร็วในอัตรา 1200 Baud

บทที่ 5. บทสรุป และวิจารณ์

บทสรุป และวิจารณ์

สรุปจากการทดลองที่ผ่านมานั้นในขั้นตอนการทำงานหลังจากที่ทางผู้จัดได้ทำการศึกษาเนื้อหาที่เกี่ยวกับไมโครคอนโทรลเลอร์ จนมีความรู้ที่สามารถที่จะออกแบบวงจรซึ่งในทางทฤษฎีสามารถที่จะทำงานได้แล้ว แต่เมื่อนำมาทำการปฏิบัติปัญหาส่วนใหญ่

●ไม่สามารถที่จะควบคุมเวลาในการทำโครงงานให้เป็นไปตามที่วางแผนไว้ได้ เนื่องทางผู้จัดทำอาจจะขาดประสบการณ์และความรับผิดชอบ

●ในโครงงานนี้ เมื่อนำไปใช้งานจริงจะต้องมีสัญลักษณ์แสดงให้เห็นถึงช่องทางการจราจรที่ชัดเจน แต่ในส่วนนั้นจะต้องลงทุนสูง เนื่องจากต้องใช้ตู้แสดงสัญลักษณ์ สายเคเบิลใยแก้ว (Optic Fiber) ดังนั้นจึงไม่สามารถทำในส่วนนั้นได้

●เวลาที่แสดงการเปลี่ยนช่องการจราจรนั้นเมื่อสัญญาณไฟกำลังกระพริบอยู่เครื่องจะไม่สามารถทำงานในส่วนอื่นๆได้เลย

●โครงงานนี้จะต้องมีแหล่งจ่ายไฟ (Power Supply) ที่มีสเถียรภาพพอสมควร เนื่องจาก MAX691 นั้นไวต่อระดับแรงดัน (Threshold Level) ที่ 4.65 V. ดังนั้นในสภาวะปกติ แรงดันที่จ่ายให้เครื่องจะต้องสม่ำเสมอ

●ส่วนการติดต่อสื่อสารนั้นทางผู้จัดทำไม่ได้มีการจัดรูปแบบการสื่อสารข้อมูลที่เป็นมาตรฐาน (Protocal) ซึ่งทางผู้จัดทำอาศัยหลักการที่เรียนมา มาทำการประยุกต์ใช้ดังนั้นอาจจะขาดความรอบครอบทางด้านนี้ไปบ้าง

●ในการต่ออุปกรณ์อินเตอร์เฟส ไอซี 8255 (IC#8255) ทางผู้จัดทำมิได้ลดหรือใช้งานอย่างคุ้มค่าเท่าใดนัก เนื่องจากสามารถที่จะออกแบบได้ง่าย ทางผู้จัดทำมีความประสงค์เพื่อให้สามารถเพิ่มศักยภาพของโครงงานนี้ได้ในอนาคตด้วย

บทวิจารณ์โครงงานเครื่องควบคุมการสลับช่องการจราจรส่วนที่สำคัญที่สุดคือความปลอดภัยของชีวิตคน ทุกๆคนที่ใช้ท้องถนนซึ่งเครื่องควบคุมการเปลี่ยนช่องการจราจรจะต้องมีหน้าที่เสมือนกับว่าจะต้องมีส่วนร่วมในการรับผิดชอบชีวิตคนด้วย ซึ่งในการออกแบบจะมีระบบป้องกันการผิดพลาดที่อาจจะเกิดขึ้นได้ นอกจากนี้การติดต่อสื่อสารของเครื่องควบคุมการเปลี่ยนช่องการจราจรในแต่ละจุดถ้าหากสามารถที่จะขยายเพื่อที่จะทำให้การแก้ปัญหาการจราจรมีประสิทธิภาพเพิ่มขึ้นอย่างมาก ซึ่งในต่างประเทศสามารถที่จะทำได้และนั้นก็เป็นความหวังทางผู้จัดทำเช่นกันถ้าประเทศไทยของเราจะเป็นเช่นที่กล่าวมา.

ภาคผนวก

```
                    ORG  0000H

SHOW_ERROR_A_RED    EQU  08000H
SHOW_ERROR_B_RED    EQU  09001H
SHOW_ERROR_C_RED    EQU  08002H
SHOW_ERROR_D_RED    EQU  08003H
SHOW_ERROR_A_GRE    EQU  08001H
SHOW_ERROR_B_GRE    EQU  09000H
SHOW_ERROR_C_GRE    EQU  09002H
SHOW_ERROR_D_GRE    EQU  09003H
COMPLEMT_A_RED      EQU  0A000H
COMPLEMT_B_RED      EQU  0A001H
COMPLEMT_C_RED      EQU  0A002H
COMPLEMT_D_RED      EQU  0A003H
COMPLEMT_A_GRE      EQU  0B000H
COMPLEMT_B_GRE      EQU  0B001H
COMPLEMT_C_GRE      EQU  0B002H
COMPLEMT_D_GRE      EQU  0B003H
PORT_A_RED          EQU  0C000H
PORT_B_RED          EQU  0C001H
PORT_C_RED          EQU  0C002H
PORT_D_RED          EQU  0C003H
PORT_A_GRE          EQU  0D000H
PORT_B_GRE          EQU  0D001H
PORT_C_GRE          EQU  0D002H
PORT_D_GRE          EQU  0D003H
PORT_A_CONTROL      EQU  0F000H
PORT_B_CONTROL      EQU  0F001H
PORT_C_CONTROL      EQU  0F002H
PORT_D_CONTROL      EQU  0F003H
COMMAND             EQU  0E000H
BUSY                EQU  0E001H
WRITE_DATA          EQU  0E002H
READ_DATA           EQU  0E003H

;-------------------------------------------------------------------------

                    LJMP START

IE0_ADDRESS:        LJMP MANUAL_CHECK          ;EXTERNAL INTERUPT VECTOR
                    NOP
                    NOP
                    NOP
                    NOP
                    NOP

TF0_ADDRESS:        LJMP CHECK_TIME_0          ;TIMER 0 INTERUPT VECTOR
                    NOP
                    NOP
                    NOP
                    NOP
                    NOP

IE1_ADDRESS:        NOP
                    NOP
                    NOP
                    NOP
                    NOP
                    NOP
                    NOP
                    NOP
```

```
TF1_ADDRESS:    LJMP CHECK_TIME_1          ;TIMER 1 INTERUPT VECTOR
                NOP
                NOP
                NOP
                NOP
                NOP

RI_TI_ADDRESS:  LJMP CHECK_RECEIVE         ;SERIAL COMMU. VECTOR
                NOP
                NOP
                NOP
                NOP
                NOP

;-------------------------------------------------------------------------

START:          MOV  R2,#5                 ;START PROGRAM
LOOP_SET_PORT:  CALL SET_PORT_8255         ;SET 8255 CHIP INTERFACE
                DJNZ R2,LOOP_SET_PORT
                MOV  DPTR,#25FFH           ;CHECK LAST POWER
                MOVX A,@DPTR
                JNE  A,#55H,START_2
                JMP  START_3
START_2:        CJNE A,#0AAH,START_5
                JMP  START_4
START_5:        MOV  A,#1
                MOV  DPTR,#ALARM_DATA_USED
                MOVX @DPTR,A
START_4:        MOV  TIMER_CHECK,#2        ;SET INITIAL
                MOV  COMMU_NUMBER,#2
                MOV  LOOP_CHECK_SPAIR,#17H
                CLR  RETURN_MAIN_BIT
                CLR  SET_TITLE_BIT
                CLR  ENTER_LANE_BIT
                CLR  FUNCTION_SET_BIT
                SETB P3.3                  ;SET WATCHDOG FUNCTION
                CALL OUT_DATA_CONTRL
                CALL FUNCTION_SET          ;SET LCD MODULE
                CALL ENTRY_MODE_SET
                CALL DISPLAY_OFF_OFF
                CALL MAP_CHARACTER
                MOV  IE,#9BH
                MOV  IP,#10H
                MOV  TMOD,#11H
                MOV  SCON,#0F0H
                MOV  T2CON,#34H
                MOV  RCAP2H,#0FEH
                MOV  RCAP2L,#0E0H
                CALL SHOW_TIME             ;BASIC DISPLAY
                CALL SET_TIMER_0
                CPL  P3.3
LOOP:           CALL START_SCAN_KEY        ;READ KEY
                CALL CHECK_AUTO_DATA
                CALL LOOP_CHECK            ;CHECK ERROR & DISPLAY LANES
                CPL  P3.3
                JMP  LOOP

START_3:        MOV  DPTR,#SECURITY_AREA   ;SET LAST OPERATE
                MOV  R0,#20H
```

```
START_3_1:        MOVX A,@DPTR
                  MOV  @R0,A
                  INC  DPTR
                  INC  R0
                  CJNE R0,#80H,START_3_1
                  JMP  START_4

CHECK_AUTO_DATA: JB   MANUAL_SET_BIT,OUT_CHECK_AUTO    ;CHECK AUTO PROGRAM
                  PUSH DPH
                  PUSH DPL
                  MOV  DPTR,#AUTO_DATA_CONTRL  ;DISPLAY AUTO LANES CONTROL
                  MOVX A,@DPTR
                  MOV  MANUAL_DATA_CONTRL,A
                  MOV  DPTR,#SUB_DATA_CONTRL_RED
                  MOVX A,@DPTR
                  POP  DPL
                  POP  DPH
                  XRL  A,MANUAL_DATA_CONTRL
                  JZ   OUT_CHECK_AUTO
                  CALL OUT_DATA_CONTRL
OUT_CHECK_AUTO:   RET

;-------------------------------------------------------------------------

LOOP_CHECK:       DJNZ LOOP_CHECK_SPAIR,OUT_CHECK_AUTO  ;CHECK ERROR
                  MOV  LOOP_CHECK_SPAIR,#13H
                  MOV  A,MANUAL_DATA_CONTRL
                  CALL DATA_SPAIR_RED
                  CPL  A
                  CALL DATA_SPAIR_GRE
                  RET

;-------------------------------------------------------------------------

STAND_BY:         MOV  DPTR,#25FFH                ;STORE DATA WHEN POWER DOWN
                  MOV  A,#55H
                  MOVX @DPTR,A
                  MOV  DPTR,#SECURITY_AREA
                  MOV  R0,#20H
STAND_BY_2:       MOV  A,@R0
                  MOVX @DPTR,A
                  INC  DPTR
                  INC  R0
                  CJNE R0,#80H,STAND_BY_2
                  JMP  CHECK_TIME_0_3

;-------------------------------------------------------------------------

CHECK_TIME_0:     PUSH DPH
                  PUSH DPL
                  PUSH ACC
                  MOV  C,P3.5                     ;CHECK POWER DOWN
                  JNC  STAND_BY
                  MOV  DPTR,#25FFH
                  MOV  A,#0AAH
                  MOVX @DPTR,A
CHECK_TIME_0_3:   DJNZ TIMER_CHECK,CHECK_TIME_0_1 ;LOOP READ TIME FROM RTC
                  MOV  TIMER_CHECK,#2
                  CALL READ_DATA_RTC
                  CALL TIME_TO_LCD
                  MOV  DPTR,#AUTOMATIC
                  MOV  R1,#0
CHECK_TIME_0_2:   MOVX A,@DPTR                    ;COMPARE TIME BETWEEN AUTO
                  CJNE A,DATA_FR_RTC_DAY,SHIFT_NEXT_1
```

```
                INC  DPTR
                MOVX A,@DPTR
                CJNE A,DATA_FR_RTC_HOUR,SHIFT_NEXT_2
                INC  DPTR
                MOVX A,@DPTR
                CJNE A,DATA_FR_RTC_MIN,SHIFT_NEXT_3   ;COMPARE DATA
                INC  DPTR
                MOVX A,@DPTR

CHECK_TIME_0_1: CALL SET_TIMER_0
                JB   SET_SLAVE_BIT,CHECK_TIME_0_4     ;COMPARE TIME LINK
                MOV  A,DATA_FR_RTC_MIN
                ANL  A,#0F0H
                CJNE A,DATA_RTC_MIN_OLD,TIME_LINK
CHECK_TIME_0_4: POP  ACC
                POP  DPL
                POP  DPH
                RETI

;-------------------------------------------------------------------------

MANUAL_CHECK:   ANL  IE,#0FEH                 ;CHECK PRESS KEY AUTO/MANUAL
                CALL DELAY_10_mS
                MOV  C,P3.2
                JC   OUT_MANUAL_1
                JB   MANUAL_SET_BIT,MANUAL_CHECK_1    ;CHANGE MANUAL/AUTO
                SETB MANUAL_SET_BIT
                CALL SHOW_MANUAL
OUT_MANUAL:     PUSH 4
                PUSH 5
                PUSH 6
                PUSH 0
                CALL KEY_SOUND_2
                CALL SHOW_TIME
                CALL SET_TIMER_0
                POP  0
                POP  6
                POP  5
                POP  4
OUT_MANUAL_1:   ORL  IE,#1
                RETI
MANUAL_CHECK_1: CLR  RETURN_MAIN_BIT          ;CHANGE AUTO/MANUAL
                CLR  MANUAL_SET_BIT
                CALL SHOW_AUTOMATIC
                JMP  OUT_MANUAL

;-------------------------------------------------------------------------

CHECK_RECIEVE:  PUSH DPH
                PUSH DPL
                PUSH ACC
                PUSH 0
                PUSH 1
                PUSH 2
                PUSH 3
                CLR  EA
                JNB  RI,$
                MOV  SCON,#0D0H
                MOV  A,SBUF
                MOV  R1,A
                ANL  1,#0C0H
                ANL  A,#3FH
                CJNE R1,#0C0H,CHECK_RECIEVE_2   ;CHECK ACK
                CJNE A,TRANSFER_ADDRESS,CHECK_TRANSMISS
```

```
                    CLR  SET_NO_ERROR_BIT
                    CALL SHOW_NO_ERROR
                    CALL KEY_SOUND
                    SETB SET_NO_ERROR_BIT
                    JMP  OUT_RECIEVE
CHECK_RECIEVE_2:    CJNE R1,#40H,CHECK_RECIEVE_3    ;CHECK TIME LINK
                    JMP  RECV_TIME_LINK
CHECK_RECIEVE_3:    CJNE R1,#0,OUT_RECEIVE          ;CHECK START BYTE
                    CJNE A,TRANSFER_ADDRESS,OUT_RECIEVE
                    MOV  PARITY_DATA_BYTE,#0
                    JNB  RI,$
                    CLR  RI
                    MOV  A,SBUF
                    CJNE A,#0AAH,CHECK_COMMAND_2    ;CHECK COMMAND
                    JMP  RECV_AUTO_DATA
CHECK_COMMAND_2:    CJNE A,#033H,CHECK_COMMAND_3
                    JMP  RECV_MANL_DATA
CHECK_COMMAND_3:    CJNE A,#0CCH,OUT_RECIEVE
                    JMP  RECV_TIME_DATA

RECV_MANL_DATA:     CALL LOOP_RECEIVE               ;RECEIVE MANUAL DATA
                    MOV  DPTR,#KEEP_DATA_SENDED
                    MOVX A,@DPTR
                    CALL FIND_PARITY_1              ;CHECK DATA
                    INC  DPTR
                    MOVX A,@DPTR
                    CJNE A,PARITY_DATA_BYTE,OUT_RECIEVE
                    CALL SEND_ACK
                    MOV  DPTR,#KEEP_DATA_SENDED
                    MOVX A,@DPTR
                    MOV  MANUAL_DATA_CONTRL,A       ;DISPLAY LANES CONTROL DATA
                    JB   MANUAL_SET_BIT,RECIEVE_MANL_1
                    MOV  DPTR,#AUTO_DATA_CONTRL
                    MOVX @DPTR,A
OUT_RECIEVE:        POP  3
                    POP  2
                    POP  1
                    POP  0
                    POP  ACC
                    POP  DPL
                    POP  DPH
                    MOV  SCON,#0F0H
                    MOV  IE,#9BH
                    RETI
RECIEVE_MANL_1:     CALL OUT_DATA_CONTRL
                    JMP  OUT_RECIEVE

RECV_AUTO_DATA:     CALL LOOP_RECEIVE               ;RECEIVE AUTOMATIC DATA
                    MOV  R0,#80H
                    MOV  DPTR,#KEEP_DATA_SENDED
                    CALL RECIEVE_AUTO_1
                    CALL SEND_ACK
                    MOV  R0,#80H                    ;KEEP AUTOMATIC DATA
                    MOV  DPTR,#AUTOMATIC
                    MOV  R2,DPH
                    MOV  R3,DPL
                    MOV  DPTR,#KEEP_DATA_SENDED
RECIEVE_AUTO_3:     MOV  R1,#8
RECIEVE_AUTO_2:     MOVX A,@DPTR
                    INC  DPTR
                    PUSH DPH
                    PUSH DPL
                    MOV  DPL,R3
                    MOV  DPH,R2
```

```
                MOVX @DPTR,A
                INC  DPTR
                MOV  R2,DPH
                MOV  R3,DPL
                POP  DPL
                POP  DPH
                DJNZ R1,RECIEVE_AUTO_2
                INC  DPTR
                DJNZ R0,RECIEVE_AUTO_3
                JMP  OUT_RECIEVE

RECV_TIME_DATA: CALL LOOP_RECEIVE           ;RECEIVE TIME DATA
                MOV  R0,#1
                MOV  DPTR,#KEEP_DATA_SENDED
                CALL RECIEVE_AUTO_1
                CALL SEND_ACK
RECIEVE_TIME_4: MOV  DPTR,#KEEP_DATA_SENDED ;WRITE TIME DATA TO RTC
                MOV  R1,#3
                MOV  R0,#DATA_TO_RTC_SEC
RECIEVE_TIME_2: MOVX A,@DPTR
                MOV  @R0,A
                INC  DPTR
                DEC  R0
                DJNZ R1,RECIEVE_TIME_2
                MOV  R1,#4
                MOV  R0,#DATA_TO_RTC_DATE
RECIEVE_TIME_3: MOVX A,@DPTR
                MOV  @R0,A
                INC  DPTR
                INC  R0
                DJNZ R1,RECIEVE_TIME_3
                CALL WRITE_DATA_RTC
RECIEVE_TIME_5: JMP  OUT_RECIEVE

RECV_TIME_LINK: JNB  RI,$                   ;RECEIVE TIME LINK
                CLR  RI
                MOV  A,SBUF
                CJNE A,#0CCH,RECIEVE_TIME_5
                MOV  DPTR,#KEEP_DATA_SENDED
RECV_TIME_LINK1:JNB  RI,$
                CLR  RI
                MOV  A,SBUF
                MOVX @DPTR,A
                INC  DPTR
                CPL  P3.3
                JNB  RB8,RECV_TIME_LINK1
                MOV  A,SBUF
                CJNE A,#80H,RECIEVE_TIME_5
                MOV  R0,#1
                MOV  DPTR,#KEEP_DATA_SENDED
                CALL RECIEVE_AUTO_1
                JMP  RECIEVE_TIME_4

LOOP_RECIEVE:   MOV  DPTR,#KEEP_DATA_SENDED ;STORE DATA SENDED
LOOP_RECIEVE_1: JNB  RI,$
                CLR  RI
                MOV  A,SBUF
                MOVX @DPTR,A
                INC  DPTR
                CPL  P3.3
                JNB  RB8,LOOP_RECIEVE_1
                MOV  A,SBUF
                MOV  R1,A
                ANL  1,#0C0H
```

```
                CJNE R1,#0C0H,OUT_RECIEVE_1
                ANL  A,#3FH
                CJNE A,TRANSFER_ADDRESS,OUT_RECIEVE_1
                MOV  SCON,#0E0H
                RET

;-------------------------------------------------------------------------

START_SCAN_KEY: MOV  A,#3EH                     ;SCAN KEY
                PUSH ACC
                ANL  A,#3
                MOV  R3,A
                POP  ACC
SCAN_KEY_LOOP:  MOV  P1,A
                MOV  A,P1                       ;READ PRESS KEY
                MOV  R2,A
                CALL DELAY_10_mS
                MOV  A,P1
                CJNE A,2,SCAN_KEY_LOOP
                ANL  A,#3CH
                CJNE A,#3CH,KEY_CODE            ;CHECK PRESS KEY
                MOV  A,#3DH
                PUSH ACC
                ANL  A,#3
                MOV  R3,A
                POP  ACC
SCAN_KEY_LOOP_1: MOV P1,A
                MOV  A,P1
                MOV  R2,A
                CALL DELAY_10_mS
                MOV  A,P1
                CJNE A,2,SCAN_KEY_LOOP_1
                ANL  A,#3CH
                CJNE A,#3CH,KEY_CODE
                CLR  CHECK_KEY_BIT_1
                CLR  REPEAT_ORDER_BIT
OUT_SCAN_KEY:   RET

REPEAT_KEY_2:   DJNZ REPEAT_ADDRESS,START_SCAN_KEY    ;LOOP REPEAT KEY
                MOV  REPEAT_ADDRESS,#FOUTH_REPEAT
                JMP  REPEAT_KEY_3

REPEAT_KEY:     DJNZ REPEAT_ADDRESS,START_SCAN_KEY    ;LOOP REPEAT KEY
                MOV  REPEAT_ADDRESS,#SECOND_REPEAT
REPEAT_KEY_3:   CLR  CHECK_KEY_BIT_1
                SETB REPEAT_ORDER_BIT
KEY_CODE:       CALL SET_TIMER_1                ;SET TIME PRESS KEY
                ORL  A,R3
                JB   MANUAL_SET_BIT,CHECK_REPT_KEY
KEY_CODE_1:     CJNE A,#2EH,KEY_CODE_2          ;PRESS SHIFT RIGHT KEY
                JB   STATUS_SET_BIT,OUT_KEY_CODE
                JB   SET_ALARM_BIT,OUT_KEY_CODE
                JB   FUNCTION_SET_BIT,OUT_KEY_CODE
                JB   CHECK_KEY_BIT_1,REPEAT_KEY
                CALL SHIFT_RIGHT_KEY
                CALL KEY_SOUND_1
                JMP  CHECK_ORDER

KEY_CODE_2:     CJNE A,#35H,KEY_CODE_3          ;PRESS SHIFT LEFT KEY
                JB   STATUS_SET_BIT,OUT_KEY_CODE
                JB   SET_ALARM_BIT,OUT_KEY_CODE
                JB   FUNCTION_SET_BIT,OUT_KEY_CODE
                JB   CHECK_KEY_BIT_1,REPEAT_KEY_2
                CALL SHIFT_LEFT_KEY
```

```
                    CALL KEY_SOUND_1
                    JMP  CHECK_ORDER_2

KEY_CODE_3:         CJNE A,#36H,KEY_CODE_4        ;PRESS SHIFT UP KEY
                    JB   STATUS_SET_BIT,OUT_KEY_CODE
                    JB   SET_TITLE_BIT,OUT_KEY_CODE
                    JB   CHECK_KEY_BIT_1,REPEAT_KEY
                    CALL SHIFT_UP_KEY
                    JMP  CHECK_ORDER

KEY_CODE_4:         CJNE A,#2DH,KEY_CODE_5        ;PRESS SHIFT DOWN KEY
                    JB   STATUS_SET_BIT,OUT_KEY_CODE
                    JB   SET_TITLE_BIT,OUT_KEY_CODE
                    JB   CHECK_KEY_BIT_1,REPEAT_KEY_2
                    CALL SHIFT_DOWN_KEY
                    JMP  CHECK_ORDER_2

CHECK_ORDER:        SETB CHECK_KEY_BIT_1          ;PRESS KEY COMPLETE
                    JB   REPEAT_ORDER_BIT,OUT_KEY_CODE
                    MOV  REPEAT_ADDRESS,#FIRST_REPEAT     ;SET HOLD PRESS KEY
OUT_KEY_CODE:       RET

CHECK_ORDER_2:      SETB CHECK_KEY_BIT_1
                    JB   REPEAT_ORDER_BIT,OUT_KEY_CODE
                    MOV  REPEAT_ADDRESS,#THIRD_REPEAT

CHECK_REPT_KEY:     JB   CHECK_KEY_BIT_1,OUT_KEY_CODE     ;MANUAL KEY
                    CJNE A,#3AH,CONT_MANUAL_1     ;CHANGE STATUS IN LANE 1
                    CPL  68H
CONT_MANUAL_0:      SETB CHECK_KEY_BIT_1
                    JMP  OUT_DATA_CONTRL
CONT_MANUAL_1:      CJNE A,#36H,CONT_MANUAL_2     ;CHANGE STATUS IN LANE 2
                    CPL  69H
                    JMP  CONT_MANUAL_0
CONT_MANUAL_2:      CJNE A,#2EH,CONT_MANUAL_3     ;CHANGE STATUS IN LANE 3
                    CPL  6AH
                    JMP  CONT_MANUAL_0
CONT_MANUAL_3:      CJNE A,#1EH,CONT_MANUAL_4     ;CHANGE STATUS IN LANE 4
                    CPL  6BH
                    JMP  CONT_MANUAL_0
CONT_MANUAL_4:      CJNE A,#39H,CONT_MANUAL_5     ;CHANGE STATUS IN LANE 5
                    CPL  6CH
                    JMP  CONT_MANUAL_0
CONT_MANUAL_5:      CJNE A,#35H,CONT_MANUAL_6     ;CHANGE STATUS IN LANE 6
                    CPL  6DH
                    JMP  CONT_MANUAL_0
CONT_MANUAL_6:      CJNE A,#2DH,CONT_MANUAL_7     ;CHANGE STATUS IN LANE 7
                    CPL  6EH
                    JMP  CONT_MANUAL_0
CONT_MANUAL_7:      CJNE A,#1DH,CONT_MANUAL_8     ;CHANGE STATUS IN LANE 8
                    CPL  6FH
                    JMP  CONT_MANUAL_0
CONT_MANUAL_8:      RET

REPEAT_KEY_1:       JMP  REPEAT_KEY

KEY_CODE_5:         CJNE A,#3AH,KEY_CODE_6        ;PRESS INCRETE KEY
                    JB   STATUS_SET_BIT,OUT_KEY_CODE_2
                    JB   FUNCTION_SET_BIT,OUT_KEY_CODE_2
                    JB   CHECK_KEY_BIT_1,REPEAT_KEY_1
                    CLR  ENTER_LANE_BIT
                    CALL SHIFT_CHAR_UP
                    JMP  CHECK_ORDER
```

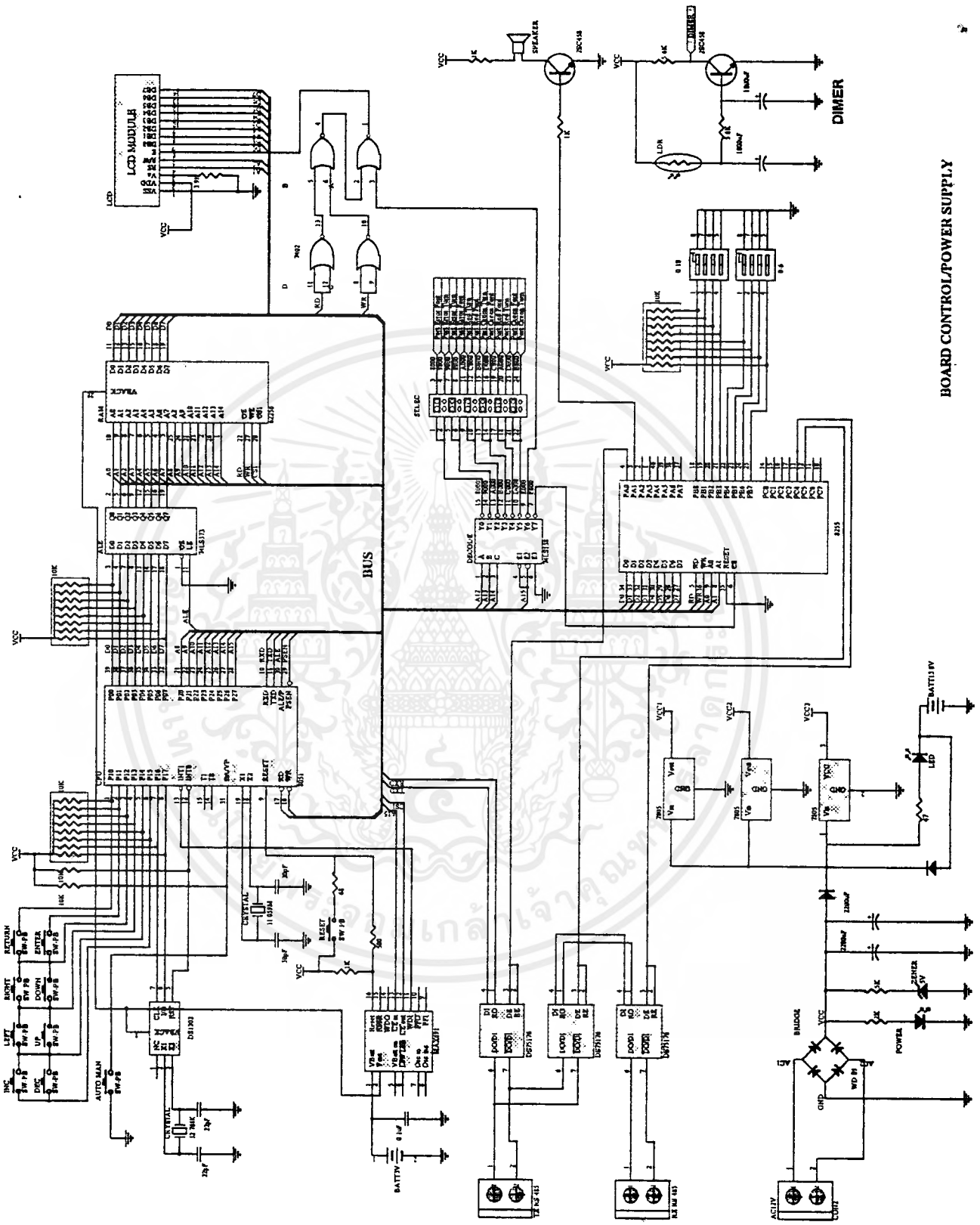
```
KEY_CODE_6:      CJNE A,#39H,KEY_CODE_7       ;PRESS DECRETE KEY
                 JB   STATUS_SET_BIT,OUT_KEY_CODE_2
                 JB   FUNCTION_SET_BIT,OUT_KEY_CODE_2
                 JB   CHECK_KEY_BIT_1,REPEAT_KEY_4
                 CLR  ENTER_LANE_BIT
                 CALL SHIFT_CHAR_DOWN
                 JMP  CHECK_ORDER_2

REPEAT_KEY_4:    JMP  REPEAT_KEY_2

KEY_CODE_7:      CJNE A,#1DH,KEY_CODE_8       ;PRESS FUNCTION/RETURN KEY
                 JB   CHECK_KEY_BIT_1,OUT_KEY_CODE_2
                 SETB CHECK_KEY_BIT_1
                 JB   RETURN_MAIN_BIT,KEY_CODE_7_1
                 MOV  TCON,#45H
                 MOV  FUNCTION_WINDOW,#0
KEY_CODE_7_3:    CALL SHOW_FUNCTION
                 CALL KEY_SOUND_4
KEY_CODE_7_4:    MOV  DATA_MODE,#0
                 CLR  SET_ALARM_BIT
                 CLR  SET_TITLE_BIT
                 CLR  STATUS_SET_BIT
                 SETB RETURN_MAIN_BIT
OUT_KEY_CODE_2:  RET

KEY_CODE_8:      CJNE A,#1EH,OUT_KEY_CODE_1   ;PRESS ENTER KEY
                 JB   STATUS_SET_BIT,OUT_KEY_CODE_1
                 JB   CHECK_KEY_BIT_1,OUT_KEY_CODE_1
                 CLR  ENTER_LANE_BIT
                 MOV  A,DATA_MODE
                 CJNE A,#2,KEY_CODE_8_2       ;STORE DATA TIME
                 JMP  KEEP_DATA_TIME
KEY_CODE_8_2:    CJNE A,#3,KEY_CODE_8_3       ;STORE DATA TITLE
                 JMP  KEEP_DATA_TITLE
KEY_CODE_8_3:    CJNE A,#1,KEY_CODE_8_5       ;STORE DATA LANE
                 JMP  KEEP_DATA_LANE
KEY_CODE_8_5:    CJNE A,#4,KEY_CODE_8_6       ;STORE DATA ALARM
                 JMP  KEEP_DATA_ALARM
KEY_CODE_8_6:    CJNE A,#5,KEY_CODE_8_7       ;SEND DATA TO SLAVE
                 JMP  SEND_DATA_TO_SL
KEY_CODE_8_7:    CJNE A,#6,OUT_KEY_CODE_8     ;CONFIRM TO SEND DATA
                 JMP  SEND_DATA_SOUR
OUT_KEY_CODE_8:  CALL SET_FUNCTION            ;ENTER TO SET FUNCTION
KEY_CODE_8_4:    CALL KEY_SOUND_4
                 CLR  ENTER_LANE_BIT
                 SETB CHECK_KEY_BIT_1
                 RET

OUT_KEY_CODE_1:  RET

;-------------------------------------------------------------------------

SEND_DATA_SOUR:  CALL KEY_SOUND_4
                 CALL SET_SEND
                 MOV  PARITY_DATA_BYTE,#0     ;SET INITIAL CHECK ERROR
                 MOV  SCON,#0F8H
                 MOV  IE,#0
                 MOV  A,TRANSFER_ADDRESS      ;SET SLAVE ADDRESS
                 ANL  A,#3FH
                 CALL START_SEND_DATA         ;SEND START BYTE
                 CLR  TB8
                 CALL SHOW_PLEAS_WAIT
                 JB   SEND_AUTO_BIT,SEND_AUTO_0
                 JB   SEND_MANL_BIT,SEND_MANUAL
```

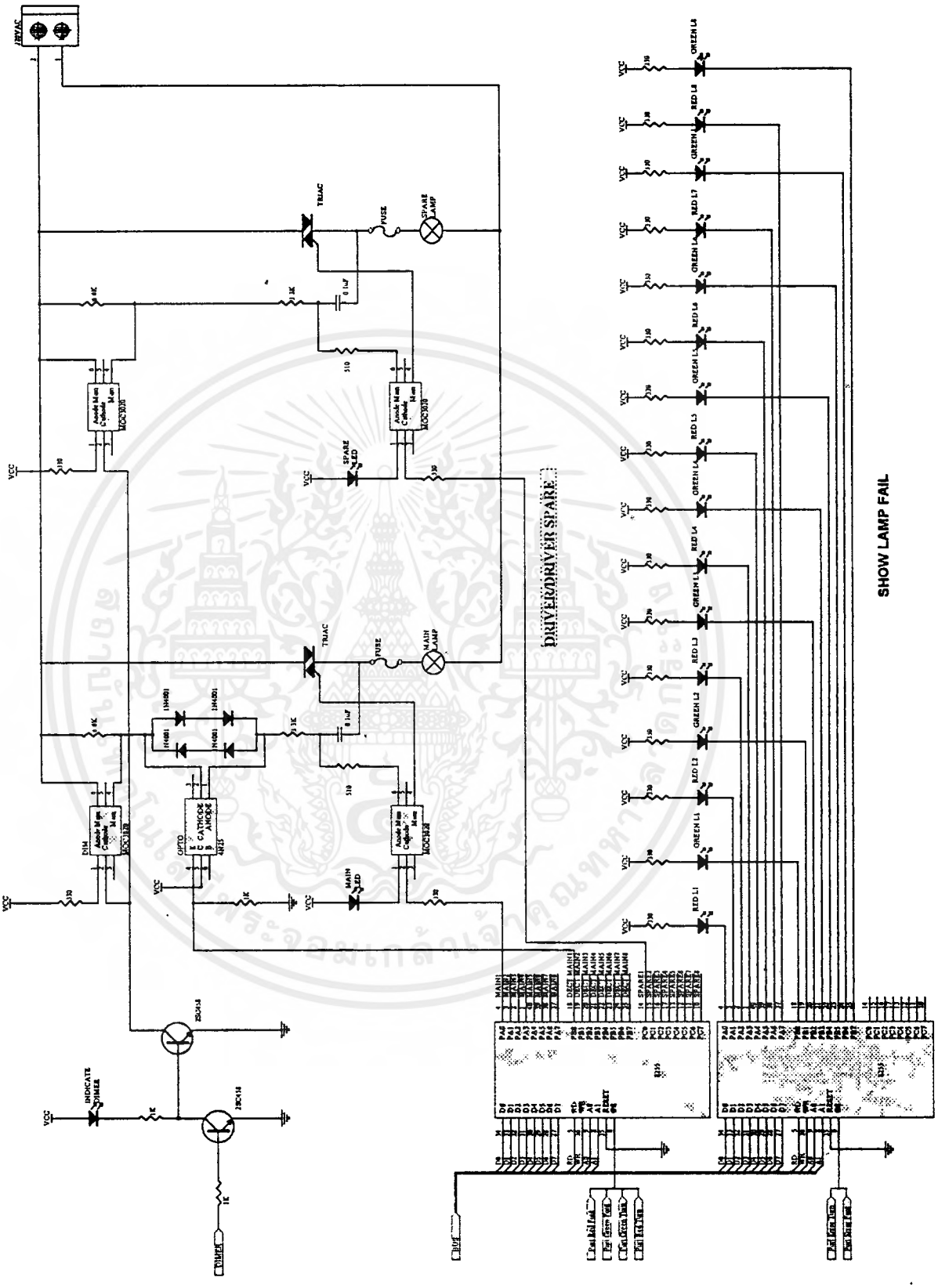
```
                JB   SEND_TIME_BIT,SEND_TIME_0
                RET

SEND_MANUAL:    MOV  A,#033H                    ;SEND COMMAND MANUAL BYTE
                CALL START_SEND_DATA
                CALL SEND_MANL                  ;SEND MANUAL DATA
SEND_MANUAL_2:  MOV  SCON,#0F8H
                MOV  A,TRANSFER_ADDRESS         ;SEND STOP BYTE
                ORL  A,#0C0H
                CALL START_SEND_DATA
                CLR  SET_NO_ERROR_BIT
                MOV  SCON,#0F0H
                MOV  IE,#90H
                CALL SET_RECIEVE
                CALL DELAY_2_S                  ;WAIT RECEIVE ACK
                CALL SHOW_ERROR                 ;DISPLAY ERROR
                MOV  DATA_MODE,#6
                MOV  IE,#9BH
                RET

SEND_AUTO_0:    MOV  A,#0AAH                    ;SEND COMMAND AUTO BYTE
                CALL START_SEND_DATA
                CALL SEND_AUTO
                JMP  SEND_MANUAL_2

SEND_TIME_0:    MOV  A,#0CCH                    ;SEND COMMAND TIME BYTE
                CALL START_SEND_DATA
                CALL SEND_TIME
                JMP  SEND_MANUAL_2

SEND_AUTO:      MOV  R7,#80H                    ;LOOP SEND AUTOMATIC
                MOV  DPTR,#AUTOMATIC            ;SEND AUTOMATIC DATA
SEND_AUTO_2:    MOVX A,@DPTR
                CALL FIND_PARITY_1
                CALL SEND_AUTO_3
                CALL FIND_PARITY_2
                CALL SEND_AUTO_3
                CALL FIND_PARITY_3
                CALL SEND_AUTO_3
                CALL FIND_PARITY_4
                CALL SEND_AUTO_3
                CALL FIND_PARITY_5
                CALL SEND_AUTO_3
                CALL FIND_PARITY_6
                CALL SEND_AUTO_3
                CALL FIND_PARITY_7
                CALL SEND_AUTO_3
                CALL FIND_PARITY_8
                CALL SEND_AUTO_3
                MOV  SBUF,PARITY_DATA_BYTE
                JNB  TI,$
                CLR  TI
                CPL  P3.3
                DJNZ R7,SEND_AUTO_2
                RET

SEND_AUTO_3:    CALL START_SEND_DATA
                INC  DPTR
                MOVX A,@DPTR
                RET

SEND_MANL:      MOV  A,MANUAL_DATA_CONTRL       ;SEND MANUAL DATA
                CALL FIND_PARITY_1
                CALL START_SEND_DATA
```

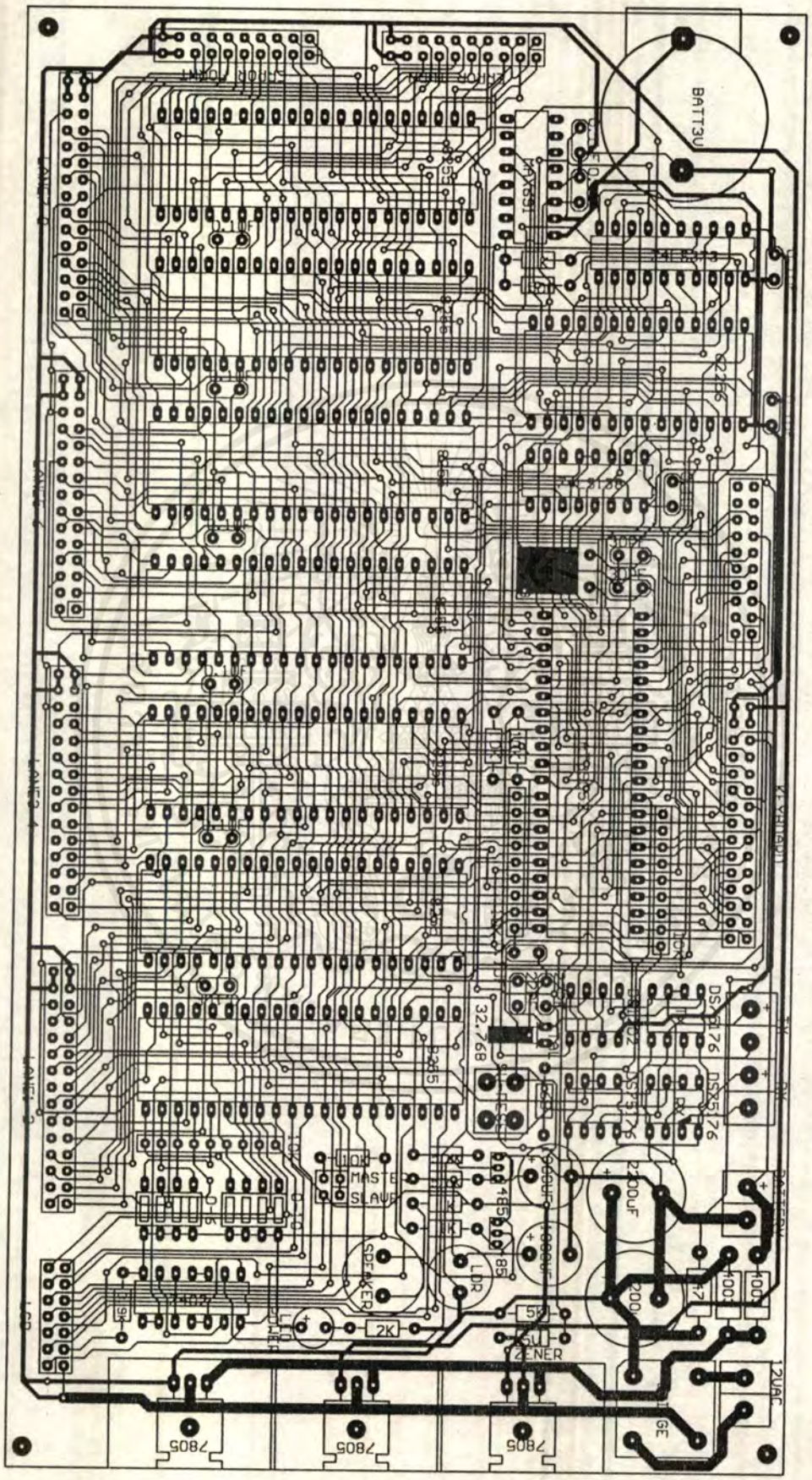
```
                MOV  SBUF,PARITY_DATA_BYTE
                JNB  TI,$
                RET

SEND_TIME:      CALL READ_DATA_RTC          ;READ DATA FROM RTC
                MOV  R0,#DATA_FR_RTC_SEC    ;SEND TIME DATA
                MOV  A,@R0
                CALL FIND_PARITY_1
                CALL START_SEND_DATA
                DEC  R0
                MOV  A,@R0
                CALL FIND_PARITY_2
                CALL START_SEND_DATA
                DEC  R0
                MOV  A,@R0
                CALL FIND_PARITY_3
                CALL START_SEND_DATA
                MOV  R0,#DATA_FR_RTC_DATE
                MOV  A,@R0
                CALL FIND_PARITY_4
                CALL SEND_TIME_2
                CALL FIND_PARITY_5
                CALL SEND_TIME_2
                CALL FIND_PARITY_6
                CALL SEND_TIME_2
                CALL FIND_PARITY_7
                CALL START_SEND_DATA
                CLR  A
                CALL FIND_PARITY_8
                CALL START_SEND_DATA
                MOV  SBUF,PARITY_DATA_BYTE
                JNB  TI,$
                CPL  P3.3
                RET

SEND_TIME_2:    CALL START_SEND_DATA
                INC  R0
                MOV  A,@R0
                RET

START_SEND_DATA: MOV  SBUF,A
                JNB  TI,$
                CLR  TI
                RET

;-------------------------------------------------------------------------

JUMP_OUT_FLASH:  JMP  OUT_FLASH

OUT_DATA_CONTRL: PUSH DPH
                PUSH DPL
                PUSH ACC
                CALL SET_PORT_8255          ;SET 8255 CHIP INTERFACE
                CALL OUT_CODE_LANE          ;FIND CODE LANE
                MOV  DPTR,#SUB_DATA_CONTRL_RED  ;COMPARE BETWEEN NEW&OLD
                MOVX A,@DPTR
                XRL  A,MANUAL_DATA_CONTRL
                JZ   JUMP_OUT_FLASH
OUT_CONTRL_2:   ANL  IE,#0
                CALL KEY_SOUND_1
                MOV  IE,#82H
                MOV  DPTR,#SUB_DATA_CONTRL_RED
                MOVX A,@DPTR
                XRL  A,MANUAL_DATA_CONTRL
```

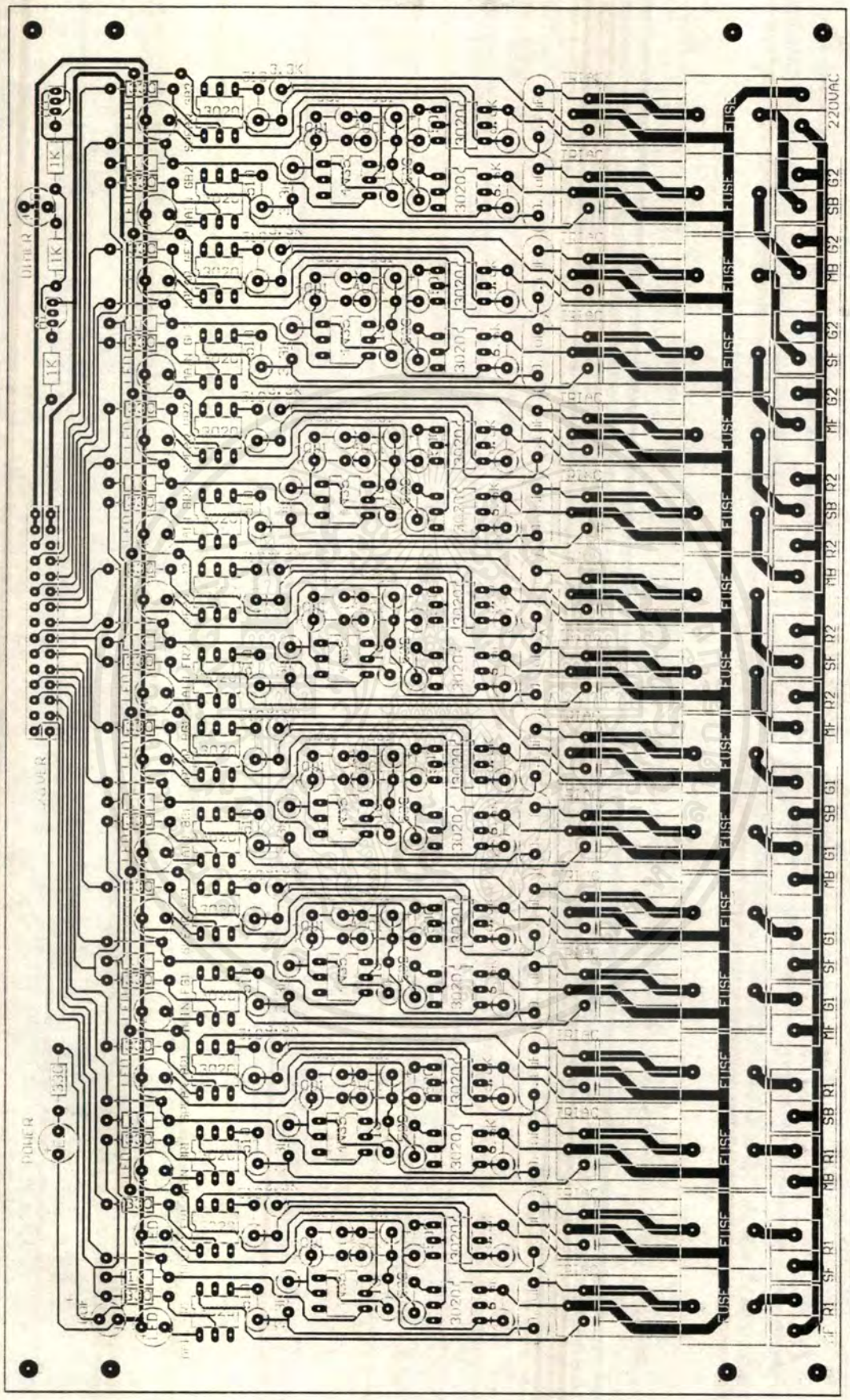
```
                    MOV  KEEP_LANE_COMP_1,A
                    CPL  A
                    MOV  KEEP_LANE_COMP_2,A
                    MOV  DPTR,#ALARM_DATA_USED   ;DISPLAY ALARM LANES CONTROL
                    MOVX A,@DPTR
                    MOV  DATA_TO_RTC_SEC,A
FLASH_CONTRL_3:     MOV  R7,#1EH
FLASH_CONTRL_2:     MOV  DPTR,#SUB_DATA_CONTRL_RED
                    MOVX A,@DPTR
                    ANL  A,KEEP_LANE_COMP_2
                    MOV  DPTR,#PORT_A_RED
                    MOVX @DPTR,A
                    MOV  DPTR,#COMPLEMT_A_RED
                    MOVX @DPTR,A
                    CALL DATA_SPAIR_RED
                    MOV  DPTR,#SUB_DATA_CONTRL_RED
                    MOVX A,@DPTR
                    XRL  A,KEEP_LANE_COMP_2
                    ANL  A,KEEP_LANE_COMP_2
                    MOV  DPTR,#PORT_A_GRE
                    MOVX @DPTR,A
                    MOV  DPTR,#COMPLEMT_A_GRE
                    MOVX @DPTR,A
                    CALL DATA_SPAIR_GRE
                    CALL DELAY_09_S
                    MOV  DPTR,#SUB_DATA_CONTRL_RED
                    MOVX A,@DPTR
                    MOV  DPTR,#PORT_A_RED
                    MOVX @DPTR,A
                    MOV  DPTR,#COMPLEMT_A_RED
                    MOVX @DPTR,A
                    CALL DATA_SPAIR_RED
                    CPL  A
                    MOV  DPTR,#PORT_A_GRE
                    MOVX @DPTR,A
                    MOV  DPTR,#COMPLEMT_A_GRE
                    MOVX @DPTR,A
                    CALL DATA_SPAIR_GRE
                    CALL DELAY_09_S
                    DJNZ R7,FLASH_CONTRL_2
                    DJNZ DATA_TO_RTC_SEC,FLASH_CONTRL_3
                    MOV  A,MANUAL_DATA_CONTRL    ;DISPLAY LANES CONTROL DATA
                    MOV  DPTR,#PORT_A_RED
                    MOVX @DPTR,A
                    MOV  DPTR,#COMPLEMT_A_RED
                    MOVX @DPTR,A
                    CALL DATA_SPAIR_RED
                    MOV  DPTR,#SUB_DATA_CONTRL_RED
                    MOVX @DPTR,A
                    CPL  A
                    MOV  DPTR,#PORT_A_GRE
                    MOVX @DPTR,A
                    MOV  DPTR,#COMPLEMT_A_GRE
                    MOVX @DPTR,A
                    CALL DATA_SPAIR_GRE
                    MOV  IE,#9BH
                    POP  ACC
                    POP  DPL
                    POP  DPH
                    RET

DATA_SPAIR_RED:     MOV  B,A
                    CLR  A
                    MOV  DPTR,#PORT_A_RED
```

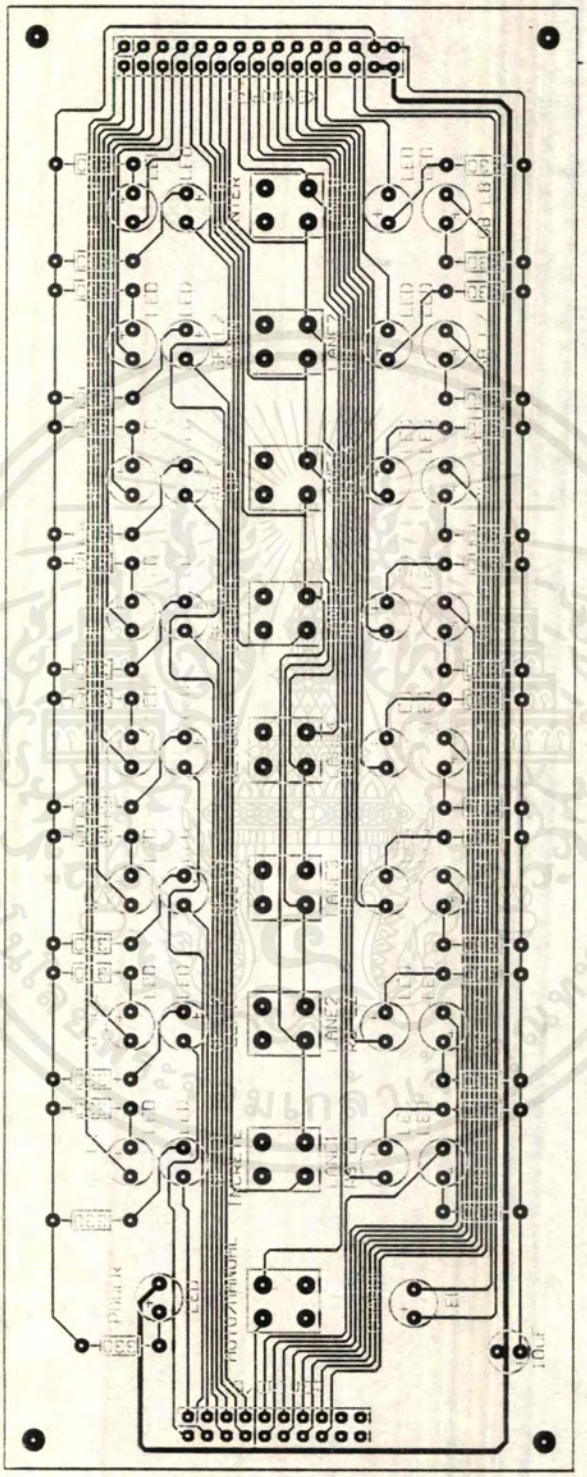
```
                MOVX @DPTR,A
                MOV  DPTR,#COMPLEMT_A_RED
                MOVX @DPTR,A
                MOV  DPTR,#PORT_B_RED          ;READ ERROR
                MOVX A,@DPTR
                MOV  DPTR,#SHOW_ERROR_A_RED
                MOVX @DPTR,A                   ;DISPLAY ERROR
                ANL  A,B
                MOV  DPTR,#PORT_C_RED          ;DISPLAY BACKUP
                MOVX @DPTR,A
                MOV  DPTR,#COMPLEMT_B_RED
                MOVX A,@DPTR
                MOV  DPTR,#SHOW_ERROR_B_RED
                MOVX @DPTR,A
                ANL  A,B
                MOV  DPTR,#COMPLEMT_C_RED
                MOVX @DPTR,A
                MOV  A,B
                MOV  DPTR,#PORT_A_RED
                MOVX @DPTR,A
                MOV  DPTR,#COMPLEMT_A_RED
                MOVX @DPTR,A
                RET

DATA_SPAIR_GRE: MOV  B,A
                CLR  A
                MOV  DPTR,#PORT_A_GRE
                MOVX @DPTR,A
                MOV  DPTR,#COMPLEMT_A_GRE
                MOVX @DPTR,A
                MOV  DPTR,#PORT_B_GRE          ;READ ERROR
                MOVX A,@DPTR
                MOV  DPTR,#SHOW_ERROR_A_GRE    ;DISPLAY ERROR
                MOVX @DPTR,A
                ANL  A,B
                MOV  DPTR,#PORT_C_GRE          ;DISPLAY BACKUP
                MOVX @DPTR,A
                MOV  DPTR,#COMPLEMT_B_GRE
                MOVX A,@DPTR
                MOV  DPTR,#SHOW_ERROR_B_GRE
                MOVX @DPTR,A
                ANL  A,B
                MOV  DPTR,#COMPLEMT_C_GRE
                MOVX @DPTR,A
                MOV  A,B
                MOV  DPTR,#PORT_A_GRE
                MOVX @DPTR,A
                MOV  DPTR,#COMPLEMT_A_GRE
                MOVX @DPTR,A
                RET

;-------------------------------------------------------------------------------

END
```

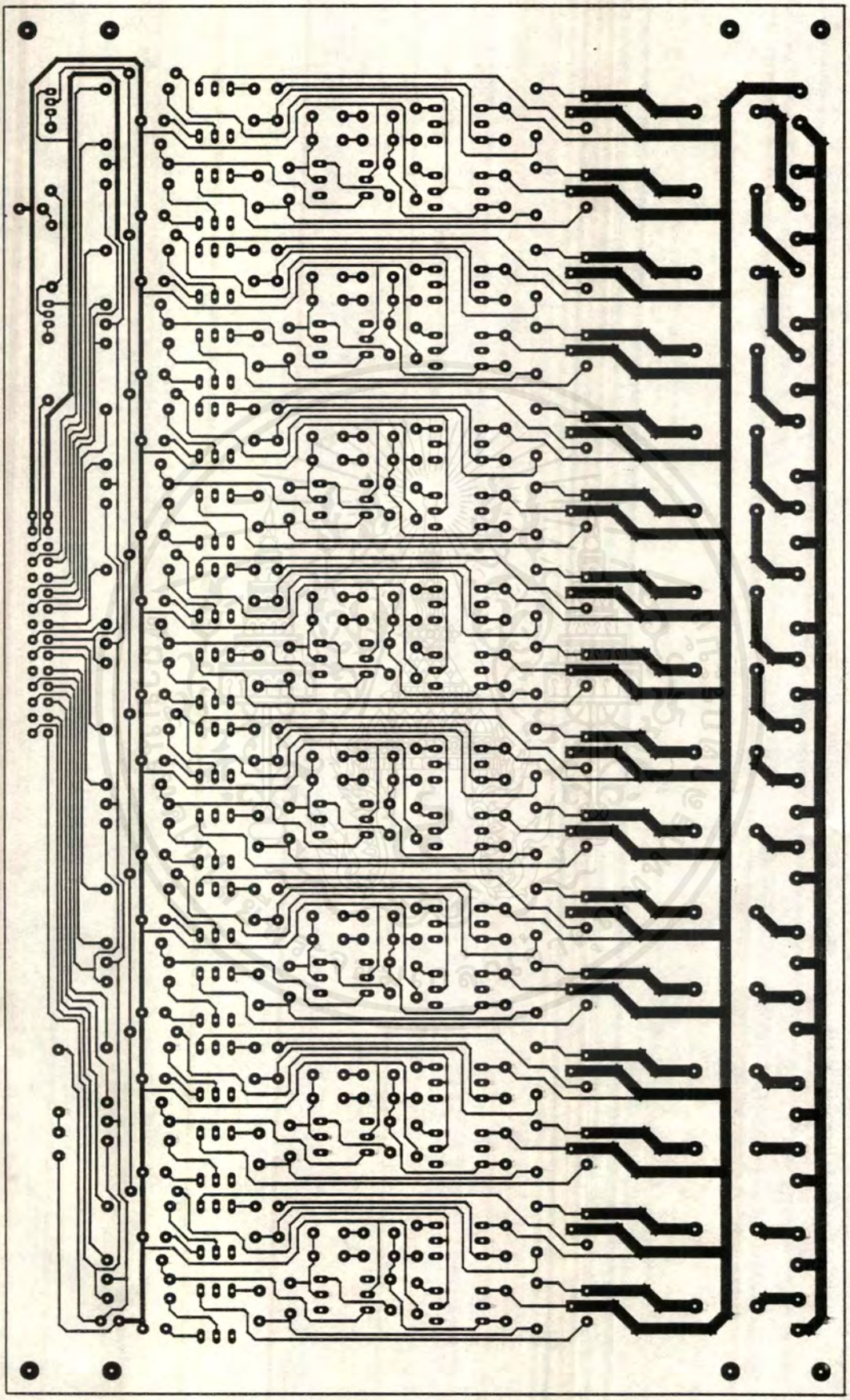


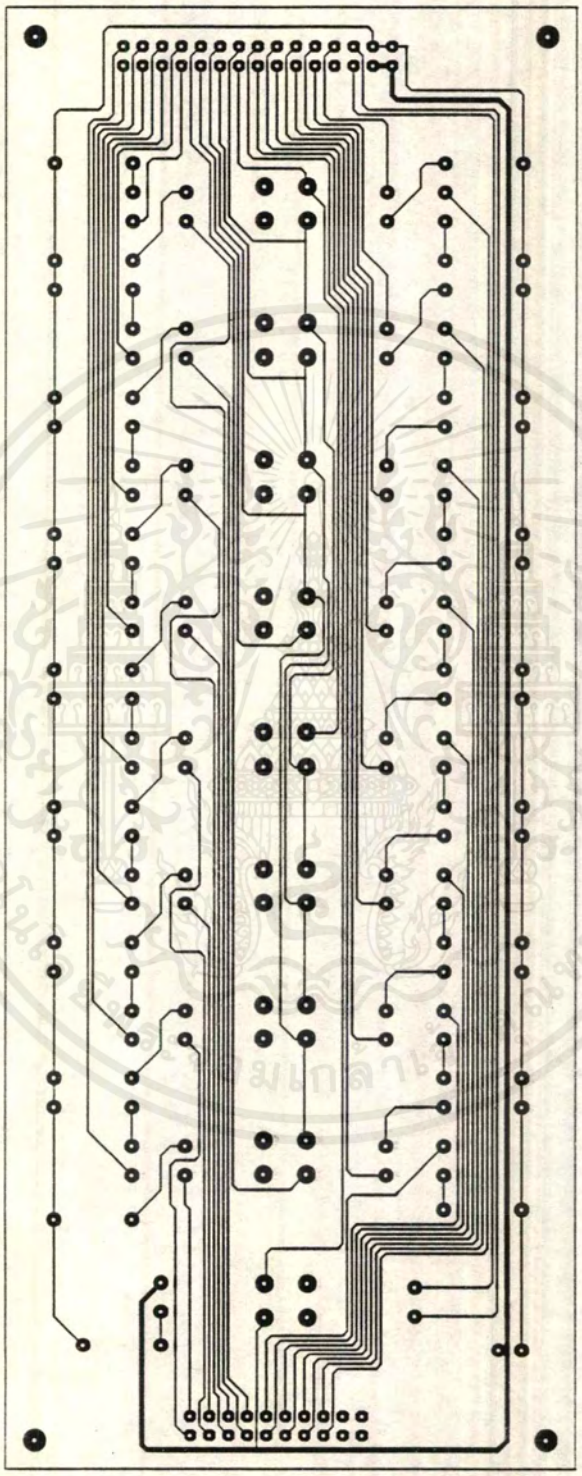












กิตติกรรมประกาศ

ปริญญานิพนธ์ชิ้นนี้สามารถที่จะสำเร็จลงได้ด้วยความช่วยเหลือของผู้มีอุปการะคุณหลายฝ่าย โดยเฉพาะอย่างยิ่ง พระคุณของบิดา มารดาผู้ซึ่งให้โอกาส และทุกสิ่งทุกอย่างแก่พวกเราคณะผู้จัดทำ ตลอดจนพระคุณท่านอาจารย์ที่ปรึกษา ท่านอาจารย์ถวิล พึ่งมาและอาจารย์มนูญ สุขเกษม ที่คอยให้คำปรึกษา คำแนะนำและเครื่องมือเครื่องใช้ที่ใช้ในการดำเนินการการทดลอง และสุดท้ายก็คงจะเป็นกลุ่มเพื่อนๆที่คอยให้กำลังใจและแนวความคิดบางสิ่งบางอย่างแก่เรา ทางคณะผู้จัดทำจึงขอขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้ด้วย

คณะผู้จัดทำ
ไมตรี สงวนไชยกฤษณ์
พิสิฐ บุญศีรเมือง

บรรณานุกรม

ไกรวุฒิ ไรจน์ประเสริฐสุด. ไมโครโปรเซสเซอร์ 2. กรุงเทพมหานคร:เม็ดทรายพริ้นติ้ง, 2539

สุเจตน์ จันทรังษ์. ไมโครคอนโทรลเลอร์ชิพเดี่ยว 8051. กรุงเทพมหานคร:วิทยาลัยมหานคร, 2535

วิวัฒน์ กิรานนท์. การสื่อสารข้อมูลดิจิตอล. กรุงเทพมหานคร:สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2535