ป้ายรถประจำทางเพื่อผู้พิการทางสายตา

BUS STATION FOR BLIND PEOPLE

โดย

นายพลากร พรหมจันทร์

นายอดิศักดิ์ เมืองใหญ่

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ตรวจครอบแล้ว

13 มี.ค. 51

ป้ายรถประจำทางเพื่อผู้พิการทางสายตา

BUS STATION FOR BLIND PEOPLE

โดย

นายพลากร พรหมจันทร์ 48015069

นายอดิศักดิ์ เมืองใหญ่ 48015087

อาจารย์ที่ปรึกษา

ผศ.ดร. พิพัฒน์ พรหมมี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ปีการศึกษา 2550

ภาควิชาวิศวกรรมไทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **ป้ายรถประจำทางเพื่อผู้พิการทางสายตา**

BUS STATION FOR BLIND PEOPLE

ผู้จัดทำ

1. นายพลากร พรหมจันทร์ 48015069

2. นายอดิศักดิ์ เมืองใหญ่ 48015087

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. พิพัฒน์ พรหมมี)

ป้ายรถประจำทางเพื่อผู้พิการทางสายตา

BUS STATION FOR BLIND PEOPLE

โดย นายพลากร พรหมจันทร์ 48015069
นายอดิศักดิ์ เมืองใหญ่ 48015087

อาจารย์ที่ปรึกษา ผศ.ดร. พิพัฒน์ พรหมมี

บทคัดย่อ

โครงการนี้นำเสนอป้ายรถประจำทางเพื่อคนพิการทางสายตา ซึ่งจะอำนวยความสะดวกแก่ผู้พิการที่ใช้บริการรถประทาง โดยป้ายรถประจำทางจะแจ้งให้ทราบว่าสายใดจะมาถึง ซึ่งจะแสดงผลทางเสียง เมื่อผู้พิการต้องการที่จะขึ้นรถก็สามารถป้อนข้อมูลรถประจำทางสายที่ต้องการ และระบบจะทำการติดต่อกับรถสายนั้นๆ เพื่อให้คนขับรถทราบ โดยการสื่อสารทั้งหมดจะใช้ความถี่วิทยุด้วยรหัสต่างๆ กัน และควบคุมการทำงานด้วยไมโครคอนโทรเลอร์

ABSTRACT

This project presents the bus station for blind people. The sign will assisted the blinds who use the bus service by using sound system to inform the blinds about the route of the coming bus. In addition, the blind can also put information of the bus they are waiting for to the system so that the system will directly contact the bus driver, about the waiting passenger. In this project, radio frequency at different signals are used and controlled by microcontroller.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้จัดทำขึ้นเป็นผลสำเร็จได้ เนื่องด้วยได้รับความอนุเคราะห์และได้รับคำปรึกษาด้วยดีจาก อาจารย์ที่ปรึกษา ผศ.ดร. พิพัฒน์ พรหมมี ที่ช่วยให้คำแนะนำข้อคิดเห็นที่ดีตลอดมา ขอขอบคุณเพื่อนๆ พี่ๆ ทุกคนที่ให้ความช่วยเหลือ ให้กำลังใจคอยให้ข้อคิดเห็นมาตลอด ขอขอบคุณปริญญานิพนธ์ของรุ่นพี่ ที่เป็นพื้นฐานอ้างอิงในการออกแบบ และการประยุกต์ที่เกี่ยวกับผลงานชิ้นนี้ และที่สำคัญต้องขอขอบคุณภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ท้ายสุดขอขอบคุณอาจารย์ทุกท่านที่ไม่ได้กล่าวถึงในที่นี้ที่กรุณา ประสิทธ์ประสาทวิชาความรู้ รวมถึงแนวทางความคิดและแนวทางการปฏิบัติให้แก่ผู้จัดทำ จนทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จผลตามเป้าหมาย

นายพลากร พรหมจันทร์
นายอดิศักดิ์ เมืองใหญ่

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของโครงงาน

ในปัจจุบันการเดินทางเข้ามามีความสำคัญกับชีวิตประจำวันของเราทุกคนเป็นอย่างมาก โดยเฉพาะในตอนนี้ที่ราคาน้ำมันมีราคาแพงมากขึ้น ดังนั้นการเดินทางโดยรถประจำทางจึงเข้ามามีความสำคัญกับเราทุกคนมากขึ้น ซึ่งรวมไปถึงผู้ที่พิการทางสายตาด้วย เนื่องจากว่าการเดินทางด้วยรถประจำทางเป็นปัญหาอย่างมากสำหรับผู้พิการทางสายตา ในกรณีที่ไม่มีใครอยู่ที่ป้ายรถประจำทาง ผู้พิการจะไม่รู้เลยว่ารถประจำทางสายที่มาถึงนั้นเป็นสายอะไร หรือถ้าอยู่บนรถก็จะไม่รู้ว่าจะถึงป้ายที่จะลงแล้วหรือยัง ดังนั้นจึงมีแนวความคิดที่จะจัดทำป้ายรถประจำทางที่สามารถให้บริการ และอำนวยความสะดวกในการใช้บริการแก่ผู้พิการทางสายตา และบุคคลทั่วไปขึ้มา

1.2 วัตถุประสงค์ของโครงงาน

ในโครงงานนี้จะกล่าวถึงการออกแบบฮาร์ดแวร์และซอฟต์แวร์ เพื่อใช้งานร่วมกันในโครงงาน โดยมีวัตถุประสงค์ในการทำโครงงานดังนี้

- เพื่อให้ผู้พิการทางสายตาสามารถ รู้ได้ว่ารถประจำทางสายใดกำลังจะเข้าป้ายรถประจำทาง
- เพื่อให้ผู้พิการทางสายตาสามารถ รู้ได้ว่าป้ายรถประจำทางต่อไปที่กำลังจะถึงเป็นป้ายอะไร
- เพื่อให้ผู้พิการทางสายตาสามารถ เรียกบริการของรถประจำทางได้

1.3 ส่วนประกอบโครงงาน

ในโครงงานนี้ประกอบด้วยกัน 2 ส่วน ดังนี้

- ส่วนของรถประจำทาง
- ส่วนของป้ายรถประจำทาง

โดยการสื่อสารข้อมูลทั้งหมดของโครงงานนี้ จะใช้โมดูล TRW 2.4 GHz และควบคุมการทำงานของโมดูลด้วยไมโครคอนโทรเลอร์ MCS-51 ในส่วนของวงจรเสียง จะใช้ IC บันทึกเสียงเบอร์ ISD25120

1.4 ขอบเขตของโครงงาน

โดยทั่วไปของโครงงานนี้ คือที่ป้ายรถประจำทางจะทำการบันทึกหมายเลขของป้ายเอาไว้ ซึ่งป้ายแต่ละป้ายในเส้นทางที่รถประจำทางนั้น ๆ วิ่งผ่าน จะมีหมายเลขที่ไม่ซ้ำกันตลอดเส้นทาง และที่รถประจำทางก็จะบันทึกหมายเลขของรถ และหมายเลขของป้ายรถประจำทางแต่ละป้ายในเส้นทางที่รถประจำทางนั้นวิ่งผ่าน

โดยจะเริ่มจากป้ายรถประจำทางจะวนส่งข้อมูลของหมายเลขป้ายออกไป และเมื่อมีรถประจำทางวิ่งเข้ามาในรัศมีของเครื่องส่ง รถประจำทางก็จะรับข้อมูลนั้นเข้ามาแล้วตรวจสอบดูว่าตรงกับข้อมูล

ของหมายเลขป้ายในเส้นทางที่รถวิ่งอยู่หรือไม่ ถ้าตรงก็จะทำการแสดงผลเป็นข้อความเสียงของชื่อป้ายนั้นออกมา จากนั้นก็จะส่งข้อมูลของหมายเลขรถตอบกลับไปยังป้าย

เมื่อป้ายได้รับข้อมูล ก็จะแสดงผลเป็นข้อความเสียงของหมายเลขรถตามข้อมูลที่รับเข้ามา และถ้าหากที่ป้ายรถประจำทางมีผู้ต้องการที่จะขึ้นรถประจำทางสายที่แสดงผลอยู่ก็สามารถทำการติดต่อกับรถเพื่อเรียกรถคันดังกล่าวได้ โดยทำการกดปุ่มเรียกรถที่อยู่บนตัวกล่องอุปกรณ์ และเมื่อรถประจำทางได้รับข้อมูลของการเรียกรถก็จะแสดงผลให้คนขับรถรู้ว่ามีคนต้องการจะขึ้นรถที่ป้ายหน้า

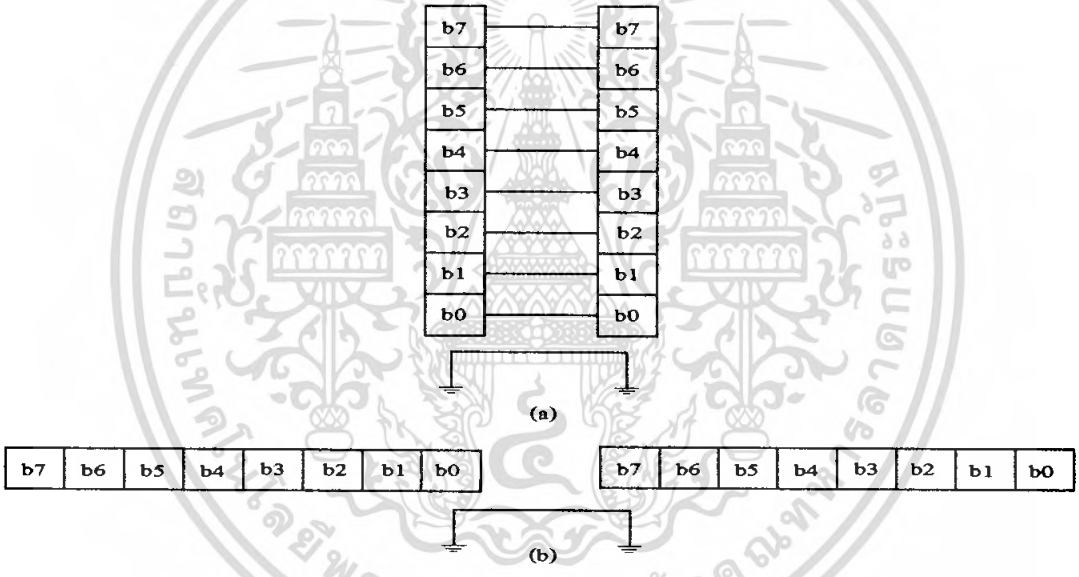
บทที่ 2
ทฤษฎีและหลักการ

2.1 การสื่อสารข้อมูล

การสื่อสารข้อมูลคือ ขบวนการในการแลกเปลี่ยนข้อมูลหรือข่าวสาร ซึ่งประกอบด้วย ผู้ส่ง (Sender) ผู้รับ (Receiver) และตัวกลางในการส่งข้อมูล (Medium) โดยที่ข้อมูลที่ทำการสื่อสารกันจะอยู่ในรูปของสัญญาณดิจิตอล (Digital) คืออยู่ในรูปของเลขฐานสอง ซึ่งอาจอยู่ในรูปรหัสตัวอักษร ตัวเลข หรือเครื่องหมายรหัส ASCII (American Standard Code For Information Interchange)

2.1.1 ประเภทของการสื่อสารข้อมูล

การสื่อสารข้อมูลแบ่งเป็นการสื่อสารข้อมูลแบบอนุกรม และการสื่อสารข้อมูลแบบขนาน



รูปที่ 2.1 รูปแบบการสื่อสารข้อมูล

(a) การสื่อสารข้อมูลแบบขนาน (b) การสื่อสารข้อมูลแบบอนุกรม

2.1.1.1 การสื่อสารข้อมูลแบบขนาน

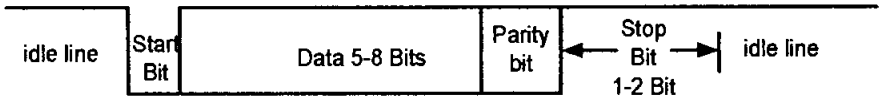
ลักษณะของการสื่อสารข้อมูลแบบขนาน จะเป็นการรับส่งข้อมูลแบบทีละไบต์ (1ไบต์เท่ากับ 8 บิต) ข้อมูลทั้งหมด 8 บิตจะถูกส่งออกจากอุปกรณ์ส่งไปยังอุปกรณ์รับพร้อมๆ กัน และช่องสัญญาณที่ใช้ในการรับส่งจะต้องมีอย่างน้อย 8 ช่องสัญญาณ สำหรับสัญญาณแต่ละบิตพร้อมกับมีสัญญาณควบคุมอีกหลายเส้น ในการส่งจะใช้สายเคเบิล (Cable) แบบที่มีตัวนำหลายสาย โดยที่ระยะทางระหว่างเครื่องทั้งสองไม่ควรมากเกินไปเนื่องจากสภาพความเป็นตัวเก็บประจุภายในสาย สภาพความไม่สมบูรณ์ของตัวนำภายในสาย และการที่ระดับของกราวด์ (ground) ทางไฟฟ้าที่อุปกรณ์รับผิดไปจากอุปกรณ์ส่งสาเหตุ

เหล่านี้ทำให้เกิดการผิดพลาดของข้อมูลได้ ข้อดีของการสื่อสารข้อมูลแบบขนานคือสามารถรับส่งข้อมูลได้รวดเร็วและเป็นจำนวนมาก ข้อเสียคือไม่เหมาะที่จะนำไปใช้ในการสื่อสารข้อมูลระยะไกล เนื่องจากค่าใช้จ่ายของสายนำสัญญาณมีราคาแพง

2.1.1.2 การสื่อสารข้อมูลแบบอนุกรม

ลักษณะของการสื่อสารแบบอนุกรม ด้านส่งจะส่งข้อมูลออกจากพอร์ต (Port) เรียงกันออกไปทีละบิตและด้านรับจะรับข้อมูลเข้ามาทีละบิตและตรวจสอบบิตที่รับเข้ามาว่าบิตใดเป็นเริ่มต้น และบิตสิ้นสุดการตรวจสอบขึ้นอยู่กับรูปแบบของรหัสของบิตที่ใช้การสื่อสารแบบอนุกรมมี 2 แบบดังนี้

- **การสื่อสารข้อมูลแบบอะซิงโครนัส (Asynchronous Transmission)** ในการสื่อสารแบบอะซิงโครนัส การส่งข้อมูลแต่ละตัวอักษรไม่มีกำหนดเวลาที่แน่นอนคือ แต่ละตัวอักษรห่างกันเท่าไรก็ได้หรือจะส่งติดต่อกันไปตลอดก็ได้ ดังนั้นเพื่อให้ผู้รับแยกออกได้ว่าข้อมูลแต่ละตัวเริ่มต้นเมื่อใด ในการส่งข้อมูลแต่ละตัวหรือแต่ละไบต์นั้นจะมีสัญญาณสำหรับตรวจสอบบิตแรกภายในตัวมันเอง โดยแต่ละไบต์จะถูกเพิ่มโดยบิตเริ่มต้น (Start Bit) ก่อนสิ้นสุดบิตก็ได้ ดังนั้นระยะเวลาระหว่างข้อมูลแต่ละไบต์ก็ไม่จำเป็นต้องแน่นอนเพราะอุปกรณ์รับจะตรวจสอบทีละไบต์เท่านั้น โดยขณะไม่มีการส่งข้อมูลสภาพลอจิก (logic) จะเป็น " 1 " อุปกรณ์รับจะคอยตรวจสอบการเปลี่ยนลอจิกจาก " 1 " เป็น " 0 " เมื่อมีการกำหนดให้บิตเริ่มต้นมีลอจิกเป็น " 0 " ซึ่งหมายถึงบิตที่ตามมาเป็นบิตแรกของไบต์นั้น รูปแบบของการจัดเรียงบิตในการสื่อสารแบบอะซิงโครนัสแสดงดังรูปที่ 2.2 (a)
- **การสื่อสารข้อมูลแบบซิงโครนัส (Synchronous Transmission)** การสื่อสารข้อมูลแบบซิงโครนัส หมายถึง การสื่อสารแบบอนุกรมที่มีการกำหนดจำนวนของอักขระที่จะส่งในแต่ละครั้งเป็นจำนวนที่แน่นอนเรียกว่า เฟรมข้อมูล (Data Frame) การส่งข้อมูลแบบนี้จะต้องมีการส่งสัญญาณนาฬิกา (clock) ไปพร้อมๆ กับสัญญาณข้อมูล ในการส่งข้อมูลระยะสั้นๆ สัญญาณนาฬิกาซึ่งใช้เป็นสัญญาณซิงค์อาจจะส่งแยกไปในสายส่งข้อมูลก็ได้ แต่ถ้าเป็นการส่งข้อมูลระยะไกลๆ แล้วสัญญาณนาฬิกาจะถูกเข้ารหัสส่งรวมไปกับสัญญาณข้อมูลในสายส่งเดียวกัน การส่งแบบซิงโครนัสข้อมูลจะเรียงติดกันไปโดยไม่มีบิตเริ่มต้นและบิตของข้อมูลบล็อกหนึ่งๆ(ในแต่ละบล็อกจะประกอบด้วยข้อมูลหลายชุด) จะแสดงจุดเริ่มต้นและจุดสิ้นสุดของข้อมูลเท่านั้น เพราะฉะนั้นถ้ามีการส่งข้อมูลเราจะเพิ่ม Framing Character เข้ารวมในแต่ละบล็อก (Block) ข้อมูลซึ่งแสดงในรูปที่ 2.2 (b)



(a)

Openning Flag 01111110	Address 8bits	Control 8bits	Data Field	CRC1	CRC2	Closing Flag 01111110

(b)

รูปที่ 2.2 การสื่อข้อมูลแบบอนุกรม

(a) การส่งข้อมูลอนุกรมแบบอะซิงโครนัส (b) การส่งข้อมูลอนุกรมแบบซิงโครนัส

เปรียบเทียบระหว่างการสื่อสารข้อมูลแบบอนุกรมกับการสื่อสารข้อมูลแบบขนาน

- ระยะทาง การสื่อสารข้อมูลแบบขนาน ปกติจะน้อยกว่า 100 ฟุต ส่วนในการสื่อสารแบบอนุกรมมากกว่า 100 ฟุต

- ความเร็ว การสื่อสารข้อมูลแบบขนานจะมีอัตราความเร็วสูงมากในระยะทางที่ไม่ไกลมากนัก ส่วนในการสื่อสารแบบอนุกรมจะมีอัตราความเร็วของข้อมูลอยู่ในช่วง 0 – 2 ล้านบิตต่อวินาที

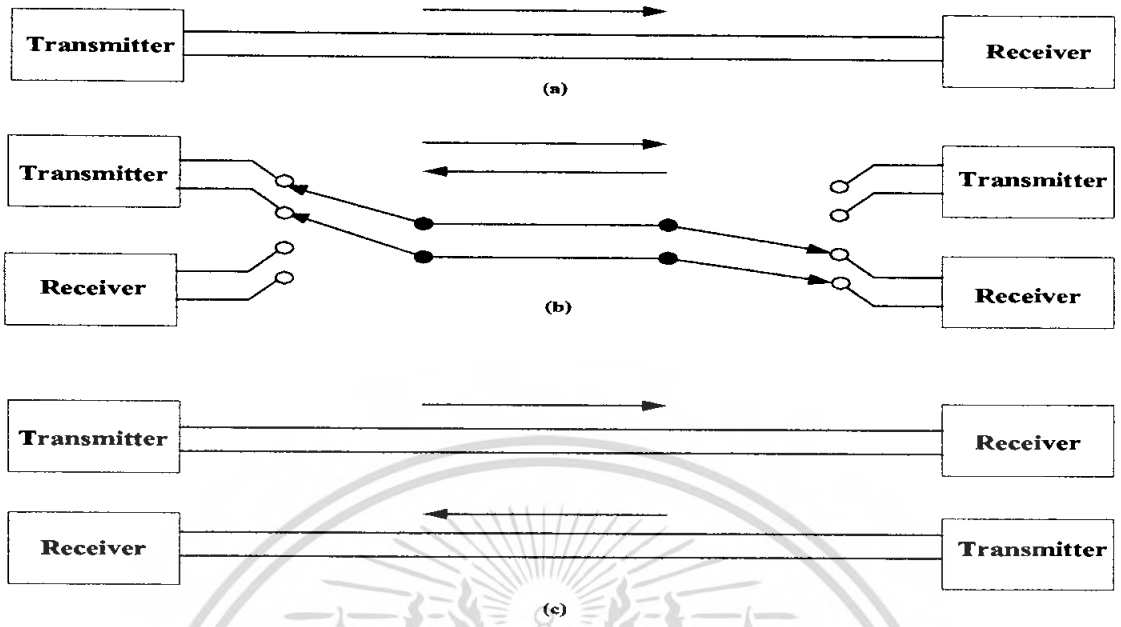
- ระดับของสัญญาณ การสื่อสารแบบขนาน การอินเตอร์เฟส (Interface) จะใช้ระดับของสัญญาณที่ใช้กับอุปกรณ์ TTL คือสัญญาณลอจิก 1 และ 0 จะแทนด้วยระดับแรงดัน +5 โวลท์ และ 0 โวลท์ ตามลำดับ ส่วนการสื่อสารแบบอนุกรมจะใช้มาตรฐาน EIA-RS232C คือมีระดับสัญญาณไฟฟ้าขนาด $\pm$ 12 โวลท์ หรืออาจจะใช้มาตรฐาน 20 mA Current Loop

- ความผิดเพี้ยนของสัญญาณ การสื่อสารข้อมูลแบบขนานถ้ามีระยะทางไกลๆ จะมีข้อมูลผิดเพี้ยนได้ง่าย ส่วนการสื่อสารข้อมูลแบบอนุกรมการผิดเพี้ยนของข้อมูลจะมีน้อยกว่า

- ค่าใช้จ่าย การสื่อสารข้อมูลแบบขนานถ้าส่งในระยะทางไกลๆ จะสิ้นเปลืองค่าใช้จ่ายมาก ส่วนการสื่อสารแบบอนุกรมจะสิ้นเปลืองน้อยกว่า แม้ว่าจะใช้อุปกรณ์เปลี่ยนสัญญาณจากข้อมูลแบบขนานไปเป็นอนุกรม และจากข้อมูลแบบอนุกรมไปเป็นขนานในการสื่อสารข้อมูล เพราะใช้จำนวนสายน้อยกว่าจึงทำให้มีราคาลงทุนต่ำกว่า

2.2 ช่องทางการสื่อสาร (Communication Channeling)

รูปแบบในการสื่อสารข้อมูลมี 3 แบบ ที่สามารถใช้เป็นช่องทางการสื่อสารที่อยู่ระหว่างสองอุปกรณ์ที่เรียกว่า ซิมเพล็กซ์ (Simplex) ฮาร์ฟดูเพล็กซ์ (Half-Duplex) และฟูลดูเพล็กซ์ (Full-Duplex) ซึ่งอธิบายในส่วนย่อยดังนี้



รูปที่ 2.3 การสื่อสารข้อมูลแบบต่างๆ

(a) การสื่อสารแบบซิมเพล็กซ์ (b) การสื่อสารแบบฮาล์ฟดูเพล็กซ์ (c) การสื่อสารแบบฟูลดูเพล็กซ์

2.2.1 ซิมเพล็กซ์

ที่เรียกกันว่าการสื่อสารทางเดียว การสื่อสารแบบซิมเพล็กซ์เป็นการทำงานในทิศทางเดียวเท่านั้น และก็ต้องการช่องทางการสื่อสาร พื้นฐานการสื่อสารแบบซิมเพล็กซ์ คือ ธุรกิจวิทยุกระจายเสียงทั่วไป ข้อมูลข่าวสารจะไหลไปในทิศทางเดียวจากผู้ประกาศไปยังผู้ฟัง ผู้ฟังจะไม่สามารถใช้เครื่องรับวิทยุเพื่อตอบสนองไปยังผู้ประกาศได้ และตัวอย่างการสื่อสารข้อมูลแบบซิมเพล็กซ์คือการอินเตอร์เฟสระหว่างคอมพิวเตอร์และเครื่องพิมพ์ (Printer) ข้อมูลจะส่งจากคอมพิวเตอร์ไปยังเครื่องพิมพ์เท่านั้น เครื่องพิมพ์ไม่สามารถส่งข้อมูลกลับมายังคอมพิวเตอร์ได้

2.2.2 ฮาร์ฟดูเพล็กซ์

การสื่อสารแบบฮาร์ฟดูเพล็กซ์สามารถทำการรับส่งได้ในแต่ละทิศทาง แต่จะทำในทิศทางเดียวที่เวลานั้น สามารถทำการสื่อสารสองช่องทางสลับกัน การสื่อสารแบบฮาร์ฟดูเพล็กซ์ต้องการช่องทางที่สามารถทำการสวิตช์ (Switch) เพื่อเปลี่ยนทิศทาง ตัวอย่างของการสื่อสารแบบฮาร์ฟดูเพล็กซ์คือระบบวิทยุสองทางเช่น เมื่อคนหนึ่งส่งอีกคนก็ทำการรับ เมื่อต้องการเปลี่ยนทิศทางของการสื่อสาร คนที่ทำการส่งต้องสวิตช์ที่โหมดรับ (Receive Mode) และคนซึ่งทำการรับก็ต้องสวิตช์ที่โหมดส่ง (Transmit Mode)

2.2.3 ฟูลดูเพล็กซ์

การสื่อสารแบบฟูลดูเพล็กซ์จะทำได้ในสองทิศทางในเวลาเดียวกัน ฟูลดูเพล็กซ์ต้องการช่องทางการสื่อสารสองช่องทางเพื่อนำพาข้อมูลข่าวสารในแต่ละทิศทาง การสื่อสารแบบฟูลดูเพล็กซ์เป็นพื้นฐานระหว่างคอมพิวเตอร์ รูปที่ 2.3 (c) แสดงถึงหลักการช่องทางการสื่อสารทั้งสองยอมให้แต่ละอุปกรณ์ปลายทางสามารถส่งและรับในเวลาเดียวกัน

2.3 โครงสร้างของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS – 51 ที่ใช้ในโครงงานนี้คือ ไมโครคอนโทรลเลอร์ตระกูล MCS – 51 ซึ่งมีหน่วยความจำภายในเป็นแบบแฟลช (Flash memory) ของ Atmel Corporation มีเบอร์ขึ้นต้นด้วย AT89 เหตุผลที่ใช้ไมโครคอนโทรลเลอร์แบบนี้ในการเรียนรู้เพื่อใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS – 51 มีด้วยกันหลายประการดังนี้

- หน่วยความจำโปรแกรมภายในตัวไมโครคอนโทรลเลอร์เป็นแบบแฟลช ทำให้สามารถลบและเขียนใหม่ได้นับพันครั้ง จึงสามารถใช้งานในรูปแบบของไมโครคอนโทรลเลอร์ชิปเดี่ยวไม่ต้องใช้หน่วยความจำนอกส่งผลให้สามารถใช้งานพอร์ตอินพุตเอาต์พุตของไมโครคอนโทรลเลอร์ได้อย่างเต็มประสิทธิภาพ
- ต้นทุนและเวลาในการพัฒนาระบบไมโครคอนโทรลเลอร์ลดลงอย่างมาก เนื่องจากไม่ต้องใช้เครื่องมือพัฒนาจำพวกอีมูเลตอร์ (Emulator) และเครื่องโปรแกรมอีพรอม (Eprom)
- บริษัทผู้ผลิตได้ทำการผลิตไมโครคอนโทรลเลอร์ตระกูลนี้ออกมาหลายเบอร์และมีความสามารถแตกต่างกันไปทำให้มีทางเลือกในการใช้งานสูง
- ด้วยการใช้หน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ ทำให้สามารถป้องกันการคัดลอกข้อมูลของหน่วยความจำโปรแกรมได้เป็นอย่างดี
- ในบางเบอร์ของไมโครคอนโทรลเลอร์ ที่ผลิตโดย Atmel สามารถทำการโปรแกรมข้อมูลในหน่วยความจำโปรแกรมได้โดยที่ไม่ต้องถอดตัวไมโครคอนโทรลเลอร์ออกมาทำการโปรแกรมใหม่หรือเรียกว่า การโปรแกรมในวงจร หรือ ในระบบ (IN – system programming) ทำให้การพัฒนาหรือการซ่อมบำรุงตลอดจนการปรับปรุงหรือ อัปเกรด (Up grade) ข้อมูลในหน่วยความจำโปรแกรมทำได้อย่างสะดวกภายใต้งบประมาณที่ไม่สูงมากนัก
- ชุดคำสั่งและสถาปัตยกรรมพื้นฐานเหมือนกับไมโครคอนโทรลเลอร์ตระกูล MCS – 51 ของผู้ผลิตอื่น ไม่ว่าจะเป็นอินเทล (Intel) ซีเมนส์ (Siemens) หรือ ดัลลัส (Dallas)

2.3.1 คุณสมบัติของไมโครคอนโทรลเลอร์ตระกูล MCS – 51 อนุกรม AT89xx

- เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียู (CPU) ขนาด 8 บิต
- ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนใหม่ได้พันครั้ง
- หน่วยความจำข้อมูลพื้นฐานเป็นหน่วยความจำแบบแรม (Ram) ในบางเบอร์จะมีหน่วยความจำแบบอีอีพรอม (EEPROM) เพิ่มเติม

- ขาพอร์ตเป็นแบบสองทิศทาง สามารถใช้งานเป็นได้ทั้งอินพุตและเอาต์พุต
- มีวงจรสื่อสารอนุกรมแบบฟูลดูเพล็กซ์
- ไทม์เมอร์ (Timer) /เคาน์เตอร์ขนาด 16 บิตอย่างน้อย 2 ตัว
- สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ (Interrupt) ได้ 6 ประเภท
- สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้สูงสุด 64 กิโลไบต์
- มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป
- มีวงจรสื่อสารอนุกรมแบบ SPI สำหรับในอนุกรม AT89Sx
- มีวอตช์ด็อกไทม์เมอร์ (Watch Dog Timer) ในตัว สำหรับในอนุกรม AT89Sxx

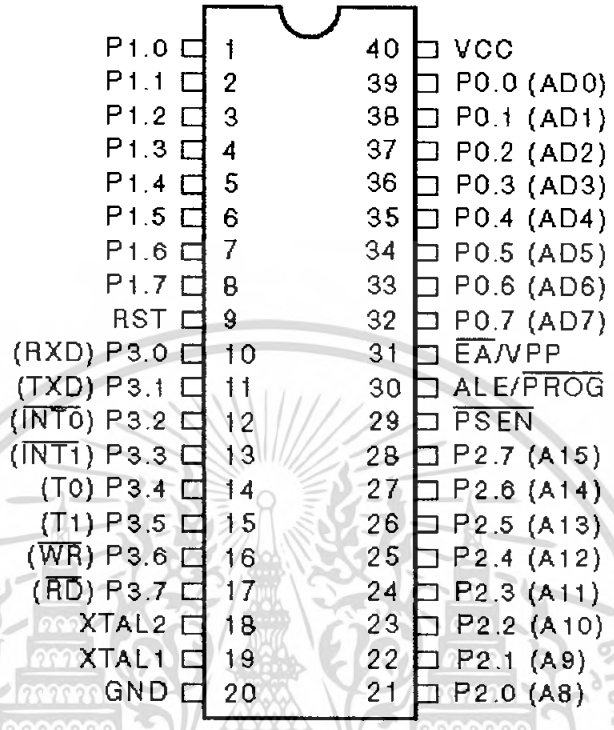
ในตารางที่ 2.1 แสดงรายละเอียดบางส่วนของไมโครคอนโทรลเลอร์ตระกูล MCS-51 แต่ละเบอร์ที่ Atmel ผลิตขึ้นและมีใช้งานอยู่ในปัจจุบัน

ตารางที่ 2.1 รายละเอียดโดยสรุปบางส่วนของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลชที่ Atmel ผลิตขึ้นและใช้ในการอ้างอิงในรายงานนี้

เบอร์ของไมโครคอนโทรลเลอร์	หน่วยความจำโปรแกรม	หน่วยความจำข้อมูล	จำนวนไทม์เมอร์/เคาน์เตอร์ 16 ชนิด
AT89C1051	แบบแฟลช ขนาด 1 กิโลไบต์	แรม 64 ไบต์	1
AT89C2051	แบบแฟลช ขนาด 2 กิโลไบต์	แรม 128 ไบต์	2
AT89C51	แบบแฟลช ขนาด 4 กิโลไบต์	แรม 128 ไบต์	2
AT89C52	แบบแฟลช ขนาด 8 กิโลไบต์	แรม 256 ไบต์	3
AT89C55	แบบแฟลช ขนาด 20 กิโลไบต์	แรม 256 ไบต์	3
AT89S8252	แบบแฟลช ขนาด 8 กิโลไบต์	แรม 256 ไบต์ อีอีพรอม 2 กิโลไบต์	3
AT89S53	แบบแฟลช ขนาด 12 กิโลไบต์	แรม 256 ไบต์	3

2.3.2 การจัดขาของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS–51 ทุกเบอร์จะมีขาใช้งานพื้นฐานเหมือนกันโดยมีรายละเอียด ดังนี้



รูปที่ 2.4 การจัดขามาตรฐานของไมโครคอนโทรเลอร์ MCS-51 AT89C5x

ขา Vcc ใช้สำหรับต่อไฟเลี้ยง +5 โวลท์

ขา GND เป็นขากราวด์ สำหรับต่อกับกราวด์ของระบบ

ขาพอร์ต 0 (P0.0 – P0.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ต 0 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย จึงมีอินพุตอิมพีแดนซ์ (Impedance) สูงสามารถใช้เป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้กระบวนการมัลติเพล็กซ์ (Multiplex) เข้าช่วย เพื่อสลับการทำงานเป็นได้ทั้งขาติดต่อแอดเดรสและขาข้อมูล

ขาพอร์ต 1 (P1.0 – P1.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไปถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย นอกจากนั้นในอนุกรม AT89Sxx จะใช้ขา P1.0 เป็นขาอินพุตสำหรับนับค่าของไทม์เมอร์ 2 และ P1.1 เป็นขาอินพุตทริกเกอร์ของไทม์เมอร์ 2 ในขณะที่ขา P1.4 ถึง P1.7 เป็นขาสำหรับเชื่อมต่อแบบ SPI เพื่อทำการโปรแกรมข้อมูลในระบบ

ขาพอร์ต 2 (P2.0 – P2.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไปถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก (A8-A15)

ขาพอร์ต 3 (P3.0 – P3.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ต 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษดังมีรายละเอียดขั้นต้นต่อไปนี้

P3.0 ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

P3.1 ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

P3.2 ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัพต์จากภายนอกช่อง 0 หรือขา $\overline{\text{INT0}}$

P3.3 ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัพต์จากภายนอกช่อง 1 หรือขา $\overline{\text{INT1}}$

P3.4 ใช้เป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0 หรือขา T0

P3.5 ใช้เป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 1 หรือขา T1

P3.6 ใช้เป็นขาสัญญาณ $\overline{\text{WR}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

P3.7 ใช้เป็นขาสัญญาณ $\overline{\text{RD}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

ขารีเซ็ต (Reset) ใช้ในการรีเซ็ทการทำงานของไมโครคอนโทรลเลอร์ โดยใช้การป้อนสัญญาณเพื่อ รีเซ็ทสถานะที่ขานี้ต้องอยู่ในระดับรีเซ็ทอย่างน้อย 2 แมชชีนไซเคิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างเป็นปกติ

ขา $\overline{\text{ALE}}$ /PROG (Address Latch Enable/Program pulse input) เป็นขาที่ใช้ในการควบคุมการแลตช์ของขาพอร์ต 0 เมื่อมีการใช้งานหน่วยความจำภายนอก นอกจากนั้นขานี้ยังใช้เป็นขาสำหรับรับพัลส์ของการโปรแกรมสำหรับโปรแกรมข้อมูลลงในไมโครคอนโทรลเลอร์ MCS-51 ในรุ่นที่มีหน่วยความจำโปรแกรมเป็นแบบอีพรอม

ขา $\overline{\text{PSEN}}$ (Program Store Enable) ขานี้ใช้ในการส่งสัญญาณเพื่อร้องขอติดต่อกับหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอก ตัวไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอกตัวไมโครคอนโทรลเลอร์จะส่งสัญญาณออกมาที่ขานี้ 2 ครั้ง ในแต่ละแมชชีนไซเคิล แต่ถ้าหากติดต่อกับหน่วยความจำข้อมูลภายนอกขานี้จะไม่มีการส่งสัญญาณใดๆออกมา

ขา $\overline{\text{EA}}$ /Vpp (External Access enable/Programming voltage input) ใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมจากภายนอกหรือภายในตัวไมโครคอนโทรลเลอร์ ถ้าหาขานี้เป็น "0" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมภายนอก แต่ถ้าหากขานี้เป็น "1" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ นอกจากนี้ที่

ขานี้ยังใช้เป็นขาอินพุตสำหรับรับแรงดันไฟสูงสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์ สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชต้องการแรงดันสำหรับการโปรแกรม +5V

ขา XTAL1 และ XTAL2 เป็นขาสำหรับต่อคริสตัลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์

2.3.3 การอ่านค่าลอจิกจากพอร์ต

ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถอ่านค่าลอจิกจากพอร์ตได้ 2 ลักษณะคือ อ่านจากขาพอร์ตโดยตรงและอ่านจากวงจรแลตช์ของแต่ละพอร์ต ในกรณีที่พอร์ตต่อกับขาเบสทรานซิสเตอร์ (Transistor) ชนิด NPN และขาอิมิตเตอร์ (Emitter) ของทรานซิสเตอร์ตัวนั้นต่อลงกราวด์ หากมีการส่งข้อมูล “ 1 ” ไปยังทรานซิสเตอร์จะทำให้ทรานซิสเตอร์ทำงานสถานะลอจิกที่ขาพอร์ตจะเป็น “ 0 ” เนื่องจากเมื่อทราน ซิสเตอร์ทำงาน จะเสมือนว่าขาพอร์ตนั้นถูกต่อลงกราวด์ ทำให้หากอ่านค่าลอจิกที่ขาพอร์ตจะได้ผลตรงข้ามกับที่ส่งออกมา แต่ถ้าหากทำการอ่านค่าลอจิกที่วงจรแลตช์ จะได้ค่าที่ตรงกับค่าที่ต้องการส่งจริงดังนั้นในการอ่านค่าลอจิกจากพอร์ตจึงต้องเลือกวิธีการให้เหมาะสมกับอุปกรณ์ที่นำมาต่อด้วย

2.3.4 จังหวะการทำงานของไมโครคอนโทรลเลอร์ MCS-51

ในการใช้งานไมโครคอนโทรลเลอร์ MCS-51 จะต้องทำความเข้าใจถึงจังหวะการทำงานของซีพียูและลำดับขั้นตอนการประมวลผลคำสั่ง ในการประมวลคำสั่งของซีพียูจะมีขั้นตอนหลัก ๆ 2 ขั้นตอนคือ

- **กระบวนการเฟตช์ (Fetch)** เป็นการเรียกคำสั่งออกจากหน่วยความจำโปรแกรมแล้วทำการแปลงรหัสคำสั่งนั้นเป็นภาษาเครื่องเพื่อเตรียมการประมวลผล
- **กระบวนการเอ็กซิคิวต์ (Execute)** เป็นการกระทำตามคำสั่งที่กำหนดหรือตามที่เฟตช์ขึ้นมาโดยกระบวนการก่อนหน้านี้ เมื่อทำการเอ็กซิคิวต์คำสั่งเรียบร้อยแล้วก็จะไปเริ่มกระบวนการเฟตช์คำสั่งใหม่ต่อไป

เมื่อเริ่มจ่ายไฟให้แก่ไมโครคอนโทรลเลอร์จะเกิดการรีเซตในลักษณะที่เรียกว่า เพาเวอร์ออนรี - เซต (Power-on reset) ซีพียูเริ่มต้นการทำงานที่แอดเดรส 0000H ของหน่วยความจำโปรแกรมจังหวะการทำงานของซีพียูจะเป็นไปตามรูปแบบ โดยได้รับการกำหนดมาจากรอบการทำงานหรือแมชชีนไซเกิล (Machine cycle)

2.3.5 โครงสร้างและการทำงานของพอร์ต

ไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลชมีพอร์ตใช้ในงานทั้งสิ้น 4 พอร์ตคือ พอร์ต 0 ถึง พอร์ต 3 แต่ละพอร์ตมีขนาด 8 บิต เป็นพอร์ตแบบ 2 ทิศทาง กล่าวคือ สามารถเป็นได้ทั้งอินพุตสำหรับรับสัญญาณข้อมูลเข้าและเอาต์พุตสำหรับส่งสัญญาณข้อมูลออก ทุกพอร์ตของไมโครคอนโทรลเลอร์

MCS – 51 แบบแฟลชมีวงจรแลตช์และวงจรขับตลอดจนบัฟเฟอร์อินพุต ที่พอร์ต 0 และพอร์ต 2 จะใช้งานเป็นพอร์ตอินพุตและเอาต์พุตสำหรับทั่วไป และใช้ในการติดต่อกับหน่วยความจำภายนอก สำหรับพอร์ต 3 ทั้งพอร์ตและพอร์ต 1 บางขานอกจากจะใช้เป็นขาพอร์ตอินพุตเอาต์พุตตามปกติแล้วยังสามารถใช้งานในหน้าที่พิเศษได้อีก ขึ้นอยู่กับว่าเป็นไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ใด ดังสรุปได้ในตารางที่ 2.2

ตารางที่ 2 .2 หน้าที่พิเศษของพอร์ต 1 ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชของ Atmel

ขา	เบอร์ของ ไมโครคอนโทรลเลอร์	หน้าที่พิเศษ
P1.0	AT89C52/AT89Sxx	ขา T2 เป็นขาอินพุตนับค่าของไทม์เมอร์/เคาน์เตอร์ 2 และเป็นขา
P1.1	AT89C52/AT89Sxx	และควบคุมทิศทางของสัญญาณ
P1.4	AT89Sxx	ขา $\overline{SS}$ (Slave Select) เป็นขาเลือกการติดต่อในกรณีที่ ไมโครคอนโทรลเลอร์เป็นอุปกรณ์สเลฟ (Slave) ในระบบการ ติดต่อแบบ SPI
P1.5	AT89Sxx	ขา MOSI (Master data output, Slave data input) ใช้ในการ ติดต่อกับพอร์ต SPI
P1.6	AT89Sxx	ขา MISO (Master data input, Slave data output) ใช้ในการ ติดต่อกับพอร์ต SPI
P1.7	AT89Sxx	ขา SCK (Master clock output) เป็นขาสัญญาณนาฬิกาของการ ติดต่อกับพอร์ต SPI

2.3.6 การใช้งานเป็นพอร์ตอินพุต

เนื่องจากพอร์ตทั้งหมดของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถเป็นได้ทั้งอินพุต และเอาต์พุต ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะการทำงานให้แก่พอร์ตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช ในการกำหนดให้เป็นพอร์ตอินพุต ต้องเริ่มต้นด้วยการเขียนข้อมูล “1” มาแต่ละบิตของพอร์ตที่ต้องการใช้งานเป็นอินพุต เพื่อหยุดการทำงานของเฟต (FET) ที่ใช้ในการขับสัญญาณเอาต์พุตของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ตเชื่อมต่อเข้ากับวงจร พูลอัป (Pull - up) ภายในโดยตรงส่งผลให้ขาพอร์ตนั้นมีลอจิกเป็น “1” สามารถรับสัญญาณลอจิก “0” จากอุปกรณ์ภายนอกได้ง่ายสัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ตแล้วรอให้ซีพียูมาอ่านค่าเข้าไป เมื่อเป็นเช่นนี้อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ตอินพุตของไมโครคอนโทรลเลอร์ MCS - 51 แบบแฟลชควรกำหนดให้ทำงานในสภาวะลอจิก “0” จะดีและสะดวกที่สุด (ซึ่งในปัจจุบันอุปกรณ์อินพุตที่ใช้ต่อไมโครคอนโทรลเลอร์ แทบทั้งหมดทำงานที่ลอจิก “0” แล้ว)

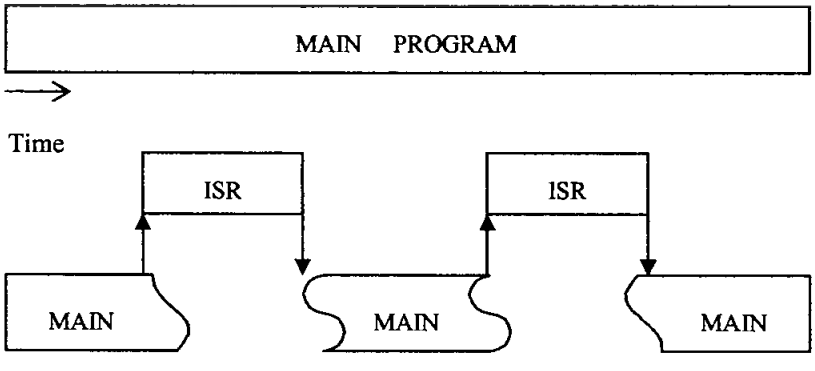
2.3.7 การใช้งานเป็นพอร์ตเอาต์พุต

โดยปกติแล้ว ขาพอร์ตจะกำหนดให้มีลักษณะเป็นเอาต์พุตอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปได้อย่างง่ายดายและตรงไปตรงมา กล่าวคือ เมื่อต้องการส่งข้อมูล "0" ออกไปทางเอาต์พุตก็ให้เขียนข้อมูล "0" ไปยังวงจรแลตช์ ซึ่งก็จะส่งต่อไปขับเฟตทำให้เฟตทำงานที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก "0" ขึ้นในทางตรงข้ามต้องการส่งข้อมูล "1" ออกไปก็ให้เขียนข้อมูล "1" ไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงานทำให้ที่ขาพอร์ตเชื่อมต่อกับวงจรพูลอัปภายในเกิดเป็นลอจิก "1" ที่ขาพอร์ตนั้น ซึ่งจะคล้ายกับการกำหนดให้เป็นขาอินพุตมาก เพียงแต่แตกต่างกันที่กระบวนการในการเคลื่อนย้ายข้อมูล โดยถ้าเป็นอินพุตจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาต์พุตจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใด เว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาต์พุตเมื่อใช้งานเป็นพอร์ตเอาต์พุต แต่ละขา (หรือแต่ละบิต) ของแต่ละพอร์ตมีความสามารถในการจ่ายกระแสหรือที่เรียกว่า กระแสซอร์ส (Source current) ได้สูงสุด 10 mA และทุกขารวมกันในแต่ละพอร์ต (ทั้ง 8 บิต) สูงสุด 26 mA สำหรับพอร์ต 0 และ 15 mA สำหรับพอร์ต 1-3 ในกรณีที่ใช้งานทุกพอร์ตเอาต์พุตจะสามารถจ่ายกระแสได้รวมกันสูงสุด 71 mA ดังนั้นในการใช้งานเป็นพอร์ตเอาต์พุตเพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัฟเฟอร์ทางเอาต์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง

2.4 การเขียนโปรแกรมอินเตอร์รัพต์

2.4.1 ความรู้เบื้องต้นเกี่ยวกับการอินเตอร์รัพต์

การอินเตอร์รัพต์คือการขัดจังหวะการทำงานของโปรแกรมหลัก โดยให้โปรแกรมหลักหยุดชั่วคราวแล้วกระโดดไปทำงานในส่วนโปรแกรมบริการอินเตอร์รัพต์ (Interrupt Service Routine) เมื่อทำงานในโปรแกรมบริการเสร็จเรียบร้อยโปรแกรมจะกลับไปทำงานโปรแกรมหลักต่อไป เหตุผลสำคัญที่ต้องมีการใช้การอินเตอร์รัพต์ เนื่องจากขณะที่มีการใช้อุปกรณ์ต่อพ่วง (Peripheral Device) หลายอุปกรณ์เข้ากับซีพียู จนซีพียูต้องทำการวนรอเพื่อตรวจสอบการร้องขอการบริการจากอุปกรณ์ต่อพ่วงเพื่อร้องขอการทำงาน จะทำให้เกิดการเสียเวลา ถ้าอุปกรณ์ต่อพ่วงมีจำนวนมาก เทคนิคการใช้วิธีการอินเตอร์รัพต์จึงช่วยลดเวลา การทำงานแบบวนรอได้โดยเปลี่ยนมาเป็นการบริการเมื่อมีการร้องขอมาแบบขัดจังหวะมาเท่านั้น



รูปที่ 2.5 การอินเตอร์รัพต์

เมื่อเกิดการอินเตอร์รัพต์ที่เวลาใด ๆ จะทำให้การทำงานของโปรแกรมหลักหยุดไป แล้วกระโดดไปทำงานของโปรแกรมบริการอินเตอร์รัพต์ (ISR) สำหรับ MCS - 51 จะมีแหล่งของอินเตอร์รัพต์อยู่ 5 แหล่งดังนี้

จากภายใน 1. Timer 0
2. Timer 1
3. Serial

จากภายนอก 1. INT0
2. INT1

สำหรับการอินเตอร์รัพต์จากภายในจะเป็นไทม์เมอร์ 0 , ไทม์เมอร์ 1 และพอร์ทอนุกรม โดยจะตรวจสอบการเกิดอินเตอร์รัพต์จากแฟลกโอเวอร์โฟลว์คือ TF1,TF0 ส่วนพอร์ทอนุกรมจะตรวจสอบจาก TI หรือ RI

การเกิดอินเตอร์รัพต์จากภายนอกจะใช้ขาสัญญาณ INT0 และ INT1 ขาสัญญาณนี้จะทำงานที่ลอจิก "0"

2.4.2 การโปรแกรมใช้งานอินเตอร์รัพต์

ในตัวไมโครคอนโทรลเลอร์ MCS-51 จะมีรีจิสเตอร์ สำหรับโปรแกรมการทำงานอินเตอร์รัพต์อยู่ 3 ตัว คือ IE (Interrupt Enable), IP (Interrupt Priority), TCON (Timer Control) ซึ่งรายละเอียดจะเป็นดังนี้

- **รีจิสเตอร์ IE (Interrupt Enable)** อยู่ตำแหน่งแอดเดรส 0A8H สามารถอ้างตำแหน่งแบบบิตได้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

รูปที่ 2.6 แสดงบิตภายใน IE

EA ถ้าเป็นลอจิก "1" หมายความว่าให้อินเตอร์รัพต์ได้

ET2 ถ้าเป็นลอจิก "1" จะเอ็นนาเบิล Timer 2 (ใช้กับเบอร์ที่มี Timer 2)

ES ถ้าเป็นลอจิก "1" จะเอ็นนาเบิล อินเตอร์รัพต์จากพอร์ทอนุกรม

ET1 ถ้าเป็นลอจิก "1" จะเอ็นนาเบิล Timer 1

EX1 ถ้าเป็นลอจิก "1" จะเอ็นนาเบิล สัญญาณอินเตอร์รัพต์จากภายนอกที่เข้ามาทางขา INT1

ET0 ถ้าเป็นลอจิก "1" จะเอ็นนาเบิล Timer 0

EX0 ถ้าเป็นลอจิก "1" จะเอ็นนาเบิล สัญญาณอินเตอร์รัพต์จากภายนอกที่เข้ามาทางขา INT0

● **รีจิสเตอร์ IP (Interrupt Priority)** อยู่ตำแหน่งแอดเดรส 0B8H สามารถอ้างตำแหน่งแบบบิตได้ ใช้กำหนดลำดับของการอินเตอร์รัพต์ กรณีที่เกิดการอินเตอร์รัพต์จากหลายแหล่งพร้อม ๆ กัน

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
-	-	PT2	PS	PT1	PX1	PT0	PX0

รูปที่ 2.7 แสดงบิตภายใน IP

PT2 สงวนไว้ใช้งานภายหน้า

PS การอินเตอร์รัพต์จาก พอร์ทอนุกรม

PT1 การอินเตอร์รัพต์จาก Timer1

PX1 การอินเตอร์รัพต์จาก INT1

PT0 การอินเตอร์รัพต์จาก Timer0

PX0 การอินเตอร์รัพต์จาก INT 0

ในกรณีที่มี การกำหนดลำดับความสำคัญของแหล่งการเกิดอินเตอร์รัพต์สูงสุดเท่ากัน ซีพียูจะเรียงลำดับความสำคัญ จากการโพล (Polling) ภายในโครงสร้างโดยมีการเรียงลำดับดังนี้

IE0 - ลำดับความสำคัญสูงสุด

TF0 - ลำดับความสำคัญสูงสุด

IE1 - ลำดับความสำคัญสูงสุด

TF1 - ลำดับความสำคัญสูงสุด

Serial - ลำดับความสำคัญสูงสุด

การเขียนโปรแกรมด้วยภาษาซีนั้นโปรแกรมตอบสนองการอินเตอร์รัพต์จะเขียนเหมือนกับฟังก์ชันทั่วไป แต่จะใช้คำว่า อินเตอร์รัพต์ และหมายเลขอินเตอร์รัพต์ต่อท้ายฟังก์ชันนั้น โดยหมายเลขอินเตอร์รัพต์ของอินเตอร์รัพต์แต่ละตัวจะเป็นดังนี้

ตารางที่ 2.3 หมายเลขอินเตอร์รัพต์ของอินเตอร์รัพต์

แหล่งกำเนิดสัญญาณอินเตอร์รัพต์	หมายเลขอินเตอร์รัพต์
อินเตอร์รัพต์จากภายนอก (IE0)	0
อินเตอร์รัพต์ไทม์เมอร์ (TF0)	1
อินเตอร์รัพต์จากภายนอก (IE1)	2
อินเตอร์รัพต์ไทม์เมอร์ (TF1)	3
อินเตอร์รัพต์พอร์ทอนุกรม	4

● **รีจิสเตอร์ TCON (Timer Control)** เป็นรีจิสเตอร์ขนาด 8 บิตอยู่ตำแหน่งที่ 88H สามารถอ้างตำแหน่งแบบบิตได้ ใช้ในการควบคุมการทำงานของไทม์เมอร์และควบคุมการทำงานอินเตอร์รัพต์ บิตต่างๆเป็นดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

รูปที่ 2.8 แสดงบิตภายใน TCON

TF1 เป็นบิตโอเวอร์โฟลว์ของไทม์เมอร์ 1 จะเป็นลอจิก "1" เมื่อไทม์เมอร์เกิดโอเวอร์โฟลว์ และบิตนี้สามารถอินเตอร์รัพต์ไมโครคอนโทรลเลอร์ได้ เมื่อทำโปรแกรมตอบสนองการอินเตอร์รัพต์จบ บิต TF1 นี้จะกลับมาเป็นลอจิก "0"

TR1 ใช้ปิดเปิดไทม์เมอร์ 1

TF0 เหมือนกับ TF1 แต่ใช้กับไทม์เมอร์ 0

TR0 ใช้ปิดเปิดไทม์เมอร์ 0

IE1 เป็นบิตแสดงการอินเตอร์รัพต์ทางฮาร์ดแวร์ที่เข้ามาทางขา INT1 ว่ามีลักษณะใด ถ้าเป็น "1" หมายความว่าจะเกิดการอินเตอร์รัพต์เมื่อมีสัญญาณขอบขาลงเข้ามา ถ้าเป็นลอจิก "0" หมายความว่าจะเกิดการอินเตอร์รัพต์เมื่อมีระดับลอจิก "0"

IE0 ใช้งานเหมือน IE1 แต่จะใช้กับ INT0

IT0 ใช้งานเหมือนกับ IT1 แต่จะใช้กับ INT0

2.5 หลักการของระบบบัส I^2C

2.5.1 ความรู้เบื้องต้นเกี่ยวกับ I^2C

I^2C ย่อมาจาก Inter-IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซีโดยบัส I^2C ได้รับการพัฒนาขึ้นโดยฟิลิปส์ (Philps) ด้วยจุดมุ่งหมายหลัก คือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงาน และควบคุมภายใต้สัญญาณเพียง 2 เส้น เส้นหนึ่ง คือ สายข้อมูล อีกเส้นหนึ่ง คือ สายสัญญาณนาฬิกา ที่ใช้ในการกำหนดจังหวะการทำงาน การต่อรวมกันของอุปกรณ์บนบัส I^2Cทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data Line) ส่วนสายสัญญาณนาฬิกาอนุกรม หรือ SCL (Serial Clock Line)

อุปกรณ์ที่ทำการเชื่อมต่อบนบัส I^2C มีหลายชนิด ไม่ว่าจะเป็นไอซีขยายพอร์ตอินพุตเอาท์พุต (I/O Expander), ไอซีแปลงสัญญาณอนาลอกเป็นดิจิตอล (ADC) และแปลงดิจิตอลเป็นอนาล็อก (DAC), ไอซีรีสไทม์คล็อก (RTC), ไอซีขับโมดูล LCD, หน่วยความจำอีอีพรอมและไมโครคอนโทรลเลอร์

2.5.2 คุณสมบัติโดยทั่วไปของบัส I^2C

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพลูอัพกับแรงดัน + 5v ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสองวงจรเอาท์พุทของอุปกรณ์ที่อยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (open-drain) หรือคอลเล็กเตอร์เปิด (open-collector)

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติ (standard mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (fast mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลการเข้าถึง 2 ค่า คือ 7 บิต (7-bit addressing) หรือ 10 บิต (10-bit addressing)

ข้อเด่นอีกประการหนึ่งของบัส I^2C คือ สามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้สามารถติดต่อสื่อสารกันได้ โดยอุปกรณ์บนบัส I^2C ตัวหนึ่งอาจใช้ไฟเลี้ยง +5V ในขณะที่อีกตัวหนึ่งใช้ไฟเลี้ยง +12V การต่อรวมกันบนบัส I^2C สามารถกระทำได้ในลักษณะเดียวกันกับกรณีที่อุปกรณ์ทั้งสองใช้ไฟเลี้ยงเท่ากัน กล่าวคือ ให้ต่อสาย SDA และ SCL ของอุปกรณ์แต่ละตัวเข้าด้วยกัน และต้องตัวต้านทานพลูอัพ (R_p) เข้ากับแรงดัน +5V ไว้ด้วยเสมอ

ในกรณีที่อาจมีแรงดันไฟกระชากขนาดใหญ่ปะปนเข้ามาในบัส I^2C ที่ขา SDA และ SCL ของอุปกรณ์แต่ละตัว ต้องต่อตัวต้านทานอนุกรมกับขา SDA และ SCL เรียกว่า R_s ก่อนต่อเข้าสู่บัส I^2C

2.5.3 หลักการของบัส I^2C

บัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้น ดังที่ได้กล่าวมาแล้ว คือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้น จึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส หรือเรียกว่า โปรโตคอล (protocal) เพื่อให้ผู้ใช้สามารถทราบได้ว่า ขณะนี้อุปกรณ์ใดติดต่อกันอยู่ และอุปกรณ์ใดเป็นตัวรับตัวส่ง ต่อไปนี้จะอธิบายลักษณะ หน้าที่ และนิยามของอุปกรณ์ที่ต่ออยู่บนบัส I^2C เพื่อเป็นข้อตกลงพื้นฐานก่อนที่จะอธิบายการทำงานของบัส I^2C ต่อไป

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง (transmitter)

อุปกรณ์ที่เป็นผู้สร้างข้อมูล เรียกว่า ตัวรับ (receiver) ในอุปกรณ์บนบัส I^2C สามารถเป็นได้ทั้งตัวรับและตัวส่ง บางอุปกรณ์ทำหน้าที่เป็นตัวรับเพียงอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I^2C ที่ทำหน้าที่เป็นตัวส่งเพียงอย่างเดียว

อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I^2C เรียกว่า มาสเตอร์ (master)

อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (slave)

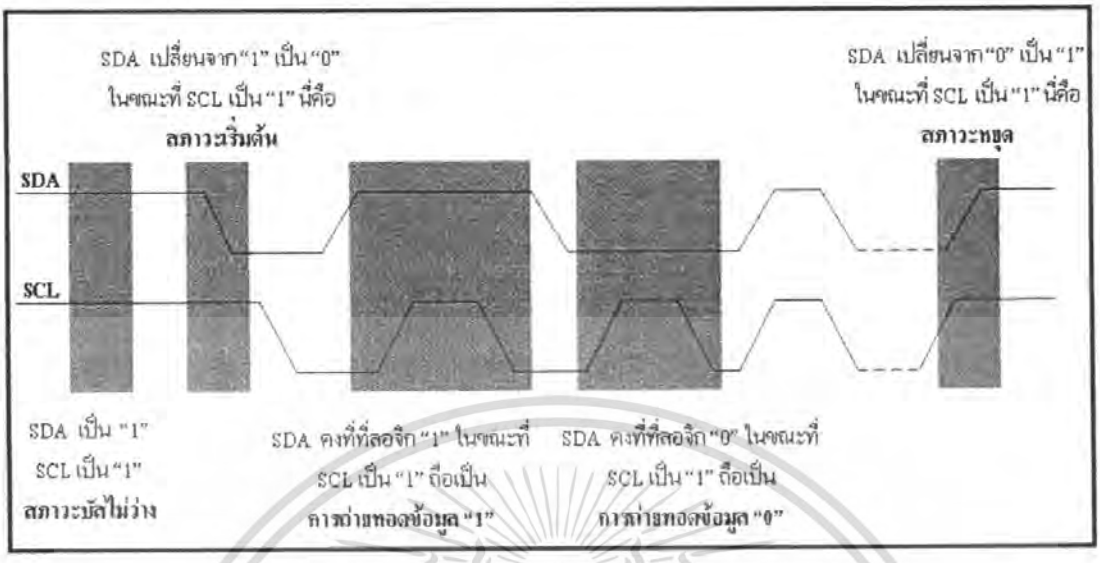
ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ

- การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น
- ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะเป็นลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

2.5.4 สภาวะที่เกิดขึ้นบนบัส I^2C

มีด้วยกัน 5 สภาวะ ดังนี้

- **บัสว่าง (BUS not busy)** สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้ง คู่ นั่นหมายความว่าการถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้
- **เริ่มต้นการถ่ายทอดข้อมูล (Start data transfer)** เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับ ลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะเริ่มต้น (START)
- **หยุดการถ่ายทอดข้อมูล (Stop data transfer)** เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิก จากต่ำไปสูง ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะหยุด (STOP)
- **ข้อมูลดำรงอยู่บนบัส (Data valid)** สภาวะนี้เกดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA คือ ข้อมูลที่ทำการถ่ายทอดเมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่าเป็น 0 หรือ 1 ข้อมูลอาจเกิดการเปลี่ยนแปลงงได้ ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการ ให้เกิดการถ่ายทอดข้อมูลอย่างสมบรูณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ช่วงเวลาที่สาย SCL มีสถานะลอจิกสูงหากมีการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุด หรือสภาวะเริ่มต้นก็ได้ ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น
- **รับรู้ข้อมูล (Acknowledge)** เกิดขึ้นหลังจากการถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบรูณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิต เรียกว่า บิตรับรู้ (Acknowledge) สถานะเป็นลอจิกสูงหลังจากส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษ ซึ่งสัมพันธ์กับสัญญาณนาฬิกา เพื่อตอบสนองบิตรับรู้ที่ส่งมาจากตัวส่ง ทางด้านตัวรับจัส่งบิตรับรู้ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อ หรือกำลังติดต่ออยู่ในขณะนั้น ก็จะกำเนิดบิตรับรู้เพื่อตอบสนองให้ทราบว่าได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว

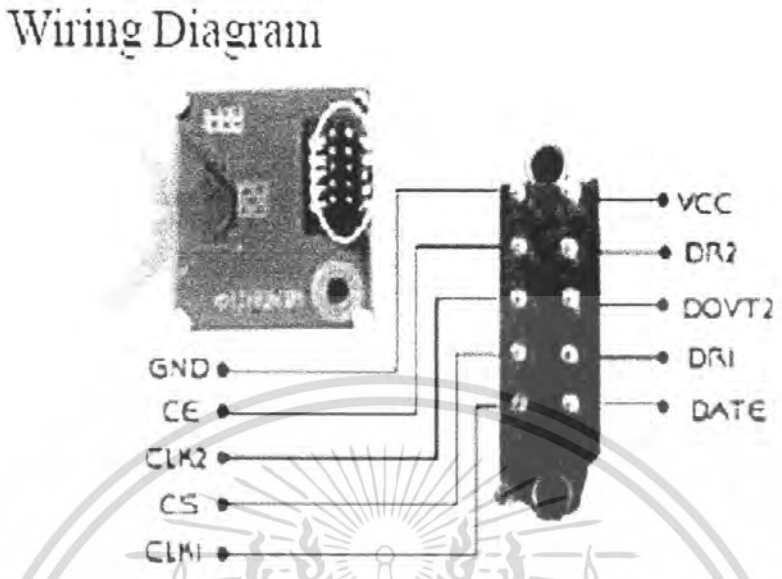


รูปที่ 2.9 ไดอะแกรมแสดงสถานะต่าง ๆ บนระบบบัส I2C

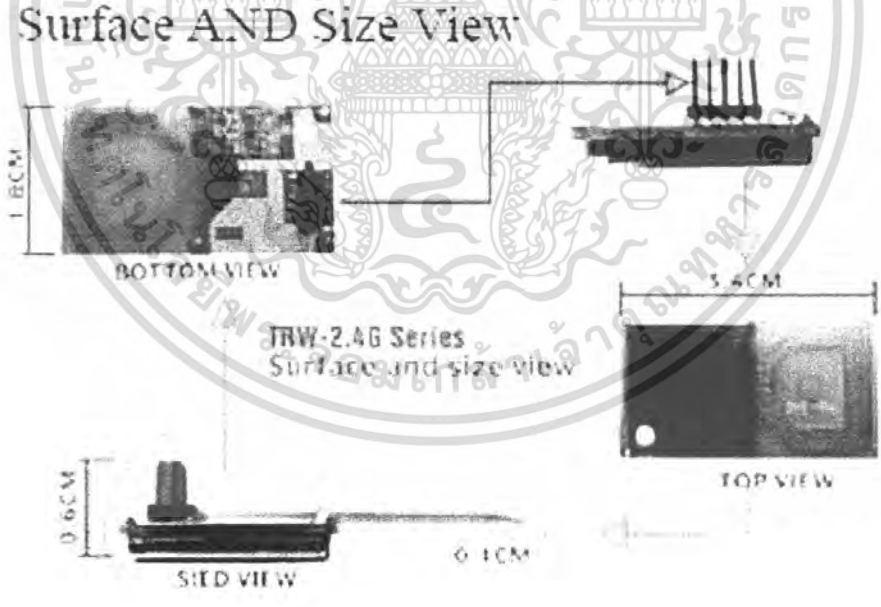
2.6 โมดูลความถี่วิทยุ 2.4 GHz

เป็นโมดูลสำเร็จรูปที่ใช้ รับ - ส่ง ข้อมูล Data ในแบบอนุกรม ใช้กับความถี่ 2.4 GHz ปรับแต่งสำเร็จรูป พร้อมมีเสาอากาศในตัว ซึ่งสามารถใช้งานได้ในระยะไกล 280 m (ความเร็วข้อมูล 250 kbps) หรือระยะ 150 m (ความเร็ว 1M bps) ในพื้นที่โล่งแจ้ง

- ความถี่ในการใช้งานที่ 2.4 ~ 2.524 GHz
- มีรูปแบบและความเร็วในการรับ-ส่งข้อมูลโดยส่งแบบ GFSK (Gaussian Frequency Shift Keying)
- ทำงานที่ความต่างศักย์ทางไฟฟ้า 3 โวลท์
- กำลังงานเอาต์พุต +4 dBm
- ความเร็วในการรับส่งข้อมูลถึง 250 Kbps : 1Mbps
- ขนาด 20.0 * 36.7 * 2.4 mm
- ทำงานที่อุณหภูมิ : -40~ +85 Centigrade
- ระยะทางในการรับส่งสัญญาณ 280 m (250Kbps) : 150m (1Mbps)
- มีเสารับส่งสัญญาณในตัวโมดูลความถี่วิทยุ



รูปที่ 2.10 รูปแสดงลักษณะขาของโมดูลความถี่วิทยุ (TRW – 2.4 GHz)



รูปที่ 2.11 แสดงรายละเอียดทางด้านบน ด้านข้างและด้านหน้าของตัวโมดูลความถี่วิทยุ

2.6.1 การจัดขาของโมดูลความถี่วิทยุ (TRW- 2.4 GHz)

ตารางที่ 2.4 แสดงลักษณะการทำงานของขาของตัวโมดูลความถี่วิทยุ

ขา	ชื่อ	ลักษณะการทำงานของขา	รายละเอียด
1	GND	Power	Gound (0V)
2	CE	Input	Chip Enable activates RX or TX mode
3	CLK2	I/O	Clock output/input for RX data channel 2
4	CS	Input	Chip Select activates Configuration mode
5	CLK1	I/O	Clock Input(TX)&I/O(RX) for data channel 1 3-wire interface
6	DATA	I/O	RX data channel 1/TX data input /3-wire interface
7	DR1	Output	RX data ready at data channel 1 (ShockBurst only)
8	DOUT2	Output	RX data channel 2
9	DR2	Output	RX data ready at data channel 2 (ShockBurst only)
10	VCC	Power	Power Supply (+3V DC)

2.6.2 โหมดการทำงานของ TRW-2.4 GHz

โหมดในการใช้งานของ TRW-2.4 GHz มีอยู่ 2 โหมดคือ

- ShockBurst Mode
- Direct Mode

โดยในโครงงานนี้ได้กำหนดให้โมดูลความถี่วิทยุมีการทำงานในโหมด ShockBurst ซึ่งมีรายละเอียดดังต่อไปนี้

2.6.2.1 โหมด Shock Burst

โหมด ShockBurst เป็นการใช้เทคโนโลยีรับ/ส่งข้อมูลบนชิป (Chip) แบบเข้าก่อน-ออกก่อน (First in – First out) โดยในการส่งข้อมูลมีทั้งระดับอัตราในการส่งบิตข้อมูลมีทั้งความเร็วต่ำและระดับความเร็วสูง เมื่อโมดูลความถี่วิทยุทำงานในโหมด ShockBurst สามารถเพิ่มการเข้าถึงระดับข้อมูลได้สูง (1 Mbps) โดยใช้ย่านความถี่ 2.4 GHz และต้องใช้ไมโครคอนโทรลเลอร์ความเร็วสูงในการประมวลผล โดยการจัดกระบวนการประมวลผลให้เหมาะสมกับโพโตคอลบนชิพจะทำให้ได้รับประโยชน์จากโมดูลความถี่วิทยุที่ตามมาดังนี้

- ประหยัดกระแส
- ระบบมีราคาต่ำ (เนื่องจากไมโครคอนโทรลเลอร์มีราคาถูก)
- ลดการชนกันของข้อมูลเมื่อใช้เวลาในการส่งระยะสั้นๆ

2.6.2.2 หลักการทำงานในโหมด ShockBurst

เมื่อทำการกำหนดค่าให้โมดูลทำงานในโหมด ShockBurst แล้ว การทำงานของโมดูลในการรับ/ส่ง ข้อมูลมีหลักการทำงานดังนี้

- **การส่งข้อมูลในโหมด ShockBurst**

โดยทำการเชื่อมต่อไมโครคอนโทรลเลอร์กับขา CE, CLK1, DATA ของตัวโมดูล

- เมื่อไมโครคอนโทรลเลอร์ต้องการส่งข้อมูลให้กับโมดูลต้องทำการเซตขา CE ให้อยู่ในสถานะ " high " เพื่อกระตุ้นให้โมดูลทำการนำข้อมูลมาเก็บไว้ภายในตัวโมดูล
- เมื่อต้องการส่งข้อมูลออกจากตัวโมดูล ทำการตั้งค่าที่ขา CE ของโมดูลที่เชื่อมต่ออยู่กับไมโครคอนโทรลเลอร์ให้อยู่ในสถานะ " low " เพื่อกระตุ้นให้โมดูลทำการส่งข้อมูล

- **การรับข้อมูลในโหมด ShockBurst**

โดยไมโครคอนโทรลเลอร์ทำการเชื่อมต่อกับขา CE, CLK1, DR1 และ DATA (กรณีที่ใช้ช่องสัญญาณเพียงช่องเดียว)

- เมื่อ RF package มีแอดเดรสที่ถูกต้องและขนาดของข้อมูลที่เข้ามา ตัวโมดูลจะทำการเซตค่าให้ขา CE อยู่ในสถานะ " high "
- เมื่อข้อมูลที่รับเข้ามาถูกต้อง(แอดเดรสและ CRC ถูกต้อง) โมดูลจะทำการย้าย preamble, address และ CRC โดยจะแจ้งไปยังไมโครคอนโทรลเลอร์ให้ทำการเซตค่า DR1 ให้อยู่ในสถานะ " high " และเซตค่าขา CE ให้อยู่ในสถานะ " low " เพื่อบอกว่าขณะนี้ทำการรับข้อมูลอยู่
- ไมโครคอนโทรลเลอร์จะทำการเซตค่าเพื่อให้รับข้อมูลได้เหมาะสมและเมื่อทำการรับข้อมูลเสร็จเรียบร้อยจะทำการเซตค่าให้ขา DR1 ให้อยู่ในสถานะ " low " เพื่อเตรียมพร้อมที่จะรับข้อมูลที่เข้ามาใหม่

2.6.2.3 ส่วนประกอบของชุดข้อมูล

Preamble	Address	Payload	CRC

รูปที่ 2.12 แสดงส่วนประกอบของเฟรมข้อมูล

ส่วนประกอบของชุดข้อมูลแบ่งได้เป็น 4 ส่วนคือ

- **Preamble** เป็นส่วนแรกของชุดข้อมูล เพื่อแสดงจุดเริ่มต้นของเฟรมข้อมูล
- **Address** เป็นส่วนที่ระบุแอดเดรสของตัวรับข้อมูล
- **Payload** เป็นส่วนที่เก็บข้อมูล
- **CRC** เป็นส่วนที่ใช้ตรวจสอบความผิดพลาดบิตของชุดข้อมูล

ตารางที่ 2.5 รายละเอียดของ Data package

1. PREMBLE	- ส่วน preamble ต้องมีใน Shock Burst mode และ Direct mode - Preamble มีความยาว 8 บิตและ ขึ้นกับบิตแรกของแอดเดรส PREAMBLE 1st ADDR_BIT 01010101 0 10101010 1 - Preamble จะใส่ไปในชุดข้อมูลโดยอัตโนมัติและดังนั้นจึงมีที่ว่างพิเศษสำหรับ payload ใน Direct mode MCU ต้องควบคุม preamble ใน ShockBurst mode Rx preamble จะถูกย้ายจากรับข้อมูลเอาท์พุต ใน Direct mode preamble จะเห็นได้ชัดเจนสำหรับข้อมูลเอาท์พุต
2. ADDRESS	- ส่วนของแอดเดรสต้องมีใน ShockBurst โหมด และ Direct โหมด - ความยาว 80 – 40 บิต - แอดเดรสจะถูกย้ายโดยอัตโนมัติจากชุดรับใน ShockBurst mode ใน Direct mode MCU ต้องควบคุมแอดเดรส
3. PAYLOAD	- ข้อมูลถูกส่ง - ใน ShockBurst โหมด payload มีขนาดน้อยที่สุด 256 บิต (แอดเดรส 8-40 บิต +CRC 8หรือ16 บิต) - ใน Direct mode ความยาวมากสุดสำหรับ 1 Mbps คือ 4000 บิต
4. CRC	- CRC จะเลือกได้ใน ShockBurst โหมด และไม่ใช้ใน Direct โหมด - ความยาว 8 หรือ 16 บิต - ShockBurst Rx CRC จะถูกย้ายจากรับข้อมูลเอาท์พุต

2.6.2.4 ตำแหน่งบิตข้อมูลของตัวโมดูลความถี่วิทยุ

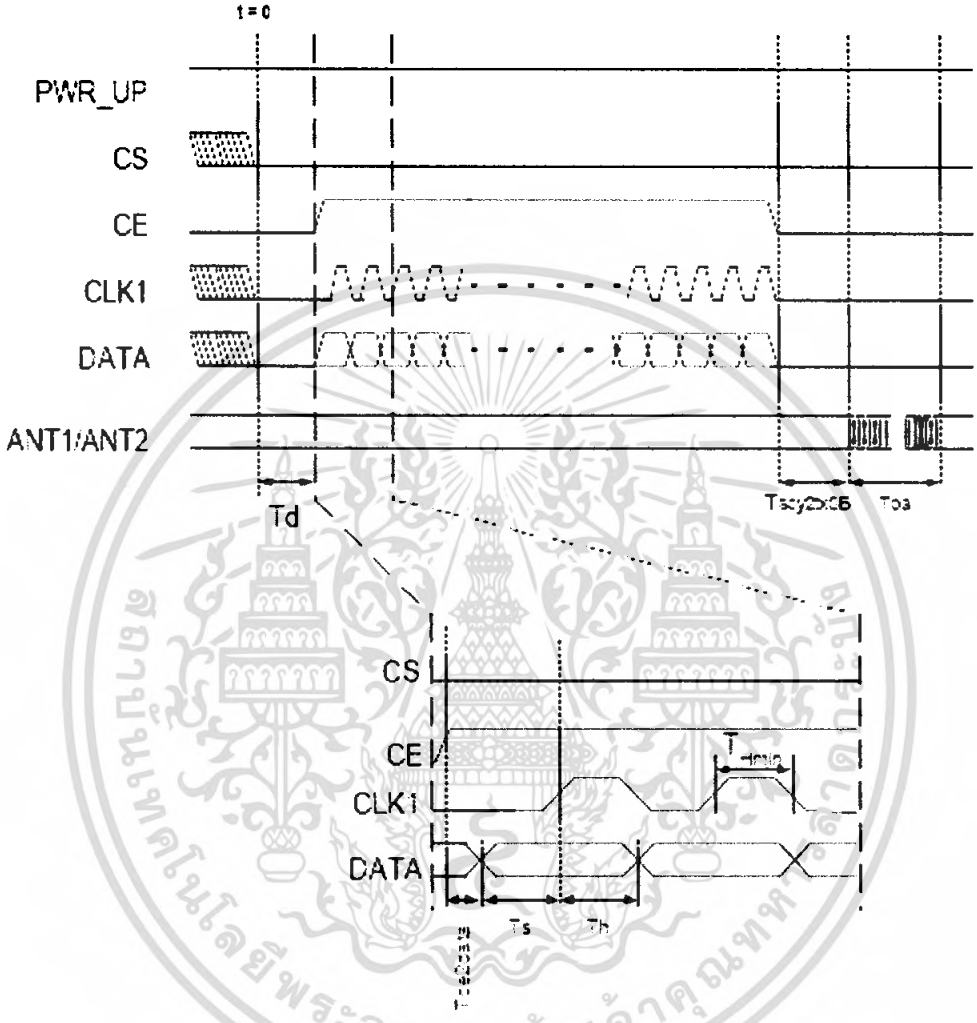
ในการที่จะกำหนดค่าให้กับตัวโมดูลความถี่วิทยุเพื่อให้โมดูลความถี่วิทยุทำงานในโหมดที่เราต้องการนั้นเราต้องทราบก่อนว่าต้องกำหนดค่าอะไรบ้างลงในตำแหน่งบิตที่เท่าไร ดังนั้นตารางข้างล่างนี้จะแสดงตำแหน่งของบิตภายในตัวโมดูลความถี่วิทยุที่ใช้ในการกำหนดค่าให้กับตัวโมดูลความถี่วิทยุ

ตารางที่ 2.6 แสดงรายละเอียดของตำแหน่งบิตภายในตัวโมดูลความถี่วิทยุ

ตำแหน่งบิต	จำนวนบิต	ชื่อ	ความหมาย
143 : 120	24	TEST	จองไว้สำหรับทดสอบข้อมูล
119 : 112	8	DATA2_W	จำนวนความกว้างของบิตข้อมูล channel 2
111 : 104	8	DATA1_W	จำนวนความกว้างของบิตข้อมูล channel 1
103 : 64	40		แอดเดรสของตัวรับ channel 2
63 : 24	40	ADDR1	แอดเดรสของตัวรับ channel 1
23 : 18	6	ADDR_W	จำนวนบิตที่จองไว้สำหรับแอดเดรสตัวรับ
17	1	CRC_L	จำนวนความกว้างของบิต CRC
16	1	CRC_EN	ยอมให้สร้าง CRC สำหรับ TX และมีการตรวจสอบบิตCRC สำหรับ RX
15	1	RX2_EN	กำหนดจำนวนสัญญาณ RX
14	1	CM	เลือกโหมด (Direct หรือ ShockBurst)
13	1	REDR_SB	เลือกอัตราการส่งผ่านข้อมูล (1 Mbps และ 250 Kbps)
12 : 10	3	XO_F	ความถี่คริสตอล (Crystal)
9 : 8	2	RF_PWR	เลือกค่ากำลังของตัวส่ง
7 : 1	7	RF_CH#	เลือกความถี่
0	1	RXEN	เลือกโหมดรับ/ส่ง

2.6.2.5 ShockBurst Mode Timing

ShockBurst TX:

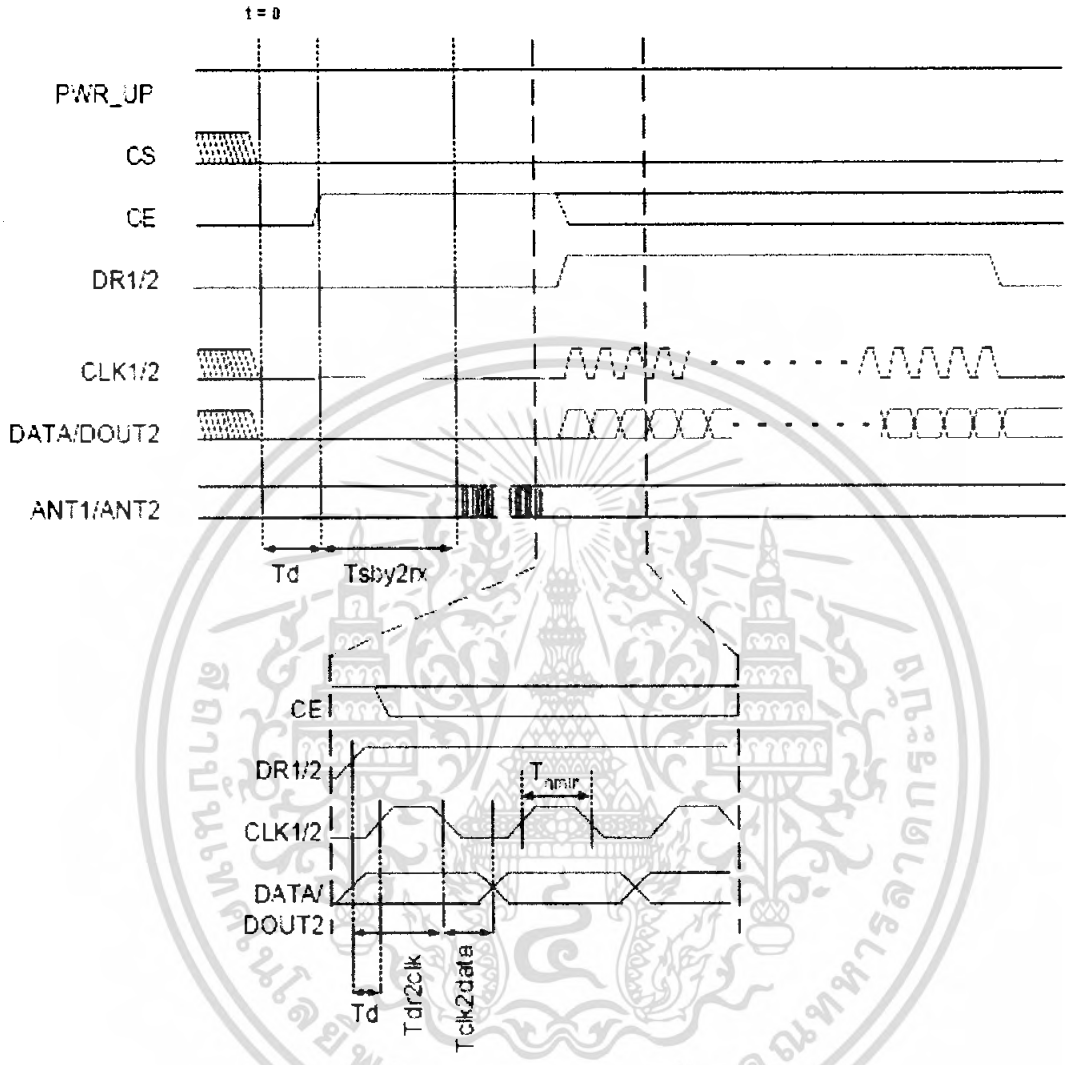


รูปที่ 2.13 Timing ของ ShockBurst ใน TX

ความยาวชุดข้อมูลและอัตราข้อมูลทำให้ delay Toa (Time on air : เวลาในอากาศ) ซึ่งแสดงในสมการ บิตข้อมูลจะเป็นผลรวมของบิตที่รวมทุกๆบิต CRC และบิต preamble ซึ่งอาจเพิ่มขึ้น

Toa = 1/data rate*(#databits+1)

ShockBurst Rx:



รูปที่ 2.14 Timing ของ ShockBurst ใน Rx

CE อาจยังคงสถานะ high ช่วงที่คาวน์โหลดข้อมูล แต่จะทำให้ค่าการใช้กระแสสูงกว่า (19mA) และผลประโยชน์คือ เวลา start-up น้อย (200us) เมื่อ DR1 ขาลง

ตำแหน่งนั้น สิ่งที่จะได้มาอย่างแรกคือ ค่าตำแหน่งของคีย์นั้น จากนั้นก็จะนำค่าตำแหน่งนั้นไปเปิดตารางข้อมูล เพื่อที่จะได้ค่าที่ต้องการนำไปแสดงผลที่แท้จริง

2.7 การขยายจำนวนพอร์ตอินพุตเอาต์พุตด้วยไอซี PCF8574A

2.7.1 ข้อมูลเบื้องต้นของ PCF8574A

ไอซี PCF8574A มีคุณสมบัติดังนี้

- ทำงานที่ระดับแรงดันตั้งแต่ 2.5V ถึง 6V
- กินกระแสในสภาวะสแตนด์บายต่ำเพียง 10 μA
- ใช้การเชื่อมต่อแบบบัส I2C
- มีเอาต์พุตอินเตอร์รัปต์แบบเดรนเปิด
- เอาต์พุตสามารถต่อกระแสได้สูง โดยสามารถนำไปขับ LED ได้โดยตรง และสามารถแลตช์ค่าได้
- สามารถกำหนดตำแหน่งแอดเดรสของไอซีแต่ละตัวได้ทางฮาร์ดแวร์ ด้วยขา A0-A2 ทำให้สามารถต่อพ่วงกันได้ถึง 8 ตัว

การจัดขาของไอซี PCF8574A การทำงานของแต่ละขาแสดงในตารางที่ P18-1 ขาพอร์ตทั้ง 8 ขาของ PCF8574A สามารถกำหนดให้เป็นอินพุตหรือเอาต์พุตได้โดยอิสระโดยไม่จำเป็นต้องใช้คำสั่งควบคุมเพื่อเลือกให้เป็นขาเอาต์พุตหรือขาอินพุต เมื่อจ่ายไฟให้กับ PCF8574A ครั้งแรก ขาพอร์ตทั้ง 8 ขาจะมีลอจิกเป็น “1” ซึ่งจะเป็นการจ่ายกระแสมาจากแหล่งจ่ายกระแสที่ภายในตัวไอซี ทำให้มีกระแสในขณะลอจิกเป็น “1” นี้ เพียง 100 μA เท่านั้น ในกรณีที่ต้องการให้มีการจ่ายกระแสสูง ๆ จะเป็นต้องต่อตัวต้านทานพูลอัปเอาไว้ที่ขาพอร์ตเหล่านี้ด้วย

เมื่อต้องการให้ขาพอร์ตเหล่านี้ทำหน้าที่เป็นอินพุตจะต้องส่งสัญญาณให้ขาเหล่านี้มีลอจิก “1” เสียก่อน เมื่อขาอินพุตได้รับสัญญาณจากภายนอกป้อนเข้ามา ไอซี PCF8574A จะสร้างสัญญาณอินเตอร์รัป (INT) ป้อนให้ไมโครคอนโทรลเลอร์หรือคอมพิวเตอร์รับรู้แทนการต้องคอยตรวจสอบขาอินพุตอยู่ตลอดเวลา สัญญาณอินเตอร์รัปต์นี้จะถูกรีเซตเมื่อมีการอ่านค่าข้อมูลหรือมีการเปลี่ยนค่าของอินพุตไปสู่ค่าเดิม

2.7.2 การเขียนโปรแกรมควบคุม PCF8574A

เนื่องจาก PCF8574A มีการเชื่อมต่อเป็นแบบบัส I2C ดังนั้นการติดต่อจึงสามารถใช้โปรแกรมย่อยการติดต่อกับอุปกรณ์สเลฟ, การสร้างสภาวะเริ่มต้น, สภาวะหยุด ส่วนที่ต้องเปลี่ยนแปลงสำหรับ PCF8574A คือ ข้อมูลกำหนดแอดเดรส โดยข้อมูลของ PCF8574A มีรูปแบบดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	1	1	1	A2	A1	A0	$R/\overline{W}$

รูปที่ 2.15 แสดงบิตข้อมูลของ PCF8574A

บิต A0,A1,A2 ใช้ในการระบุ PCF8574A ที่ใช้บนบอร์ดในกรณีที่มีการต่อ PCF8574A มากกว่า 1 ตัว โดยค่าของ A0-A2 จะมีความแตกต่างกันไปในแต่ละตัว สามารถกำหนดได้ทางฮาร์ดแวร์ โดยการต่อขา A0-A2 เข้ากับไฟเลี้ยง +5V เพื่อกำหนดเป็นลอจิก "1" หรือกราวด์เพื่อกำหนดเป็นลอจิก "0" ดังนั้นค่าแอดเดรสของ PCF8574A 1 ตัวจึงต้องนำสถานะที่กำหนดทางฮาร์ดแวร์ที่ขา A0-A2 มารวมด้วยจึงจะสมบูรณ์ ส่วนบิต $R/\overline{W}$ ใช้กำหนดว่าต้องการอ่านหรือเขียนข้อมูลกับไอซี PCF8574A ยกตัวอย่าง ถ้าหากกำหนดขา A0-A2 ลงกราวด์ทั้งหมด และต้องการอ่านข้อมูลจาก PCF8574A ข้อมูลกำหนดแอดเดรสที่ต้องส่งให้แก่ PCF8574A คือ 01110001B เป็นต้น

ไอซี PCF8574A ยังมีอีกเบอร์หนึ่งในอนุกรมเดียวกัน นั่นคือ PCF8574 ซึ่งก็มีข้อมูลกำหนดแอดเดรสที่แตกต่างกับ PCF8574A แต่ฟังก์ชันการทำงานเหมือนกันทุกประการ โดยข้อมูลกำหนดแอดเดรสของ PCF8574 มีดังนี้

ตารางที่ 2.7 รายละเอียดหน้าที่การทำงานของแต่ละขาของไอซี PCF8574/PCF8574A

ชื่อ	ตำแหน่งขา	หน้าที่
A_0	1	อินพุตแอดเดรสตัวที่ 1
A_1	2	อินพุตแอดเดรสตัวที่ 2
A_2	3	อินพุตแอดเดรสตัวที่ 3
P_0	4	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 0
P_1	5	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 1
P_2	6	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 2
P_3	7	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 3
V_{ss}	8	กราวด์
P_4	9	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 4
P_5	10	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 5
P_6	11	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 6
P_7	12	พอร์ตอินพุตเอาต์พุต 2 ทิศทางบิต 7
INT	13	ขาเอาต์พุตอินเตอร์รัปต์(ทำงานที่ลอจิก0)
SCL	14	ขาสัญญาณนาฬิกาสำหรับ I2C บัส
SDA	15	ขาข้อมูลสำหรับ I2C บัส
V_{DD}	16	ไฟเลี้ยง

ดังนั้น จึงสามารถต่อพ่วงไอซีในอนุกรม PCF8574x ได้สูงถึง 16 ตัว การส่งหรือเขียนข้อมูลไปยัง PCF8574A โดยมีลำดับขั้นตอนดังนี้

- เตรียมข้อมูลกำหนดแอดเดรสของ PCF8574A
- เรียกโปรแกรมย่อยการติดต่อกับอุปกรณ์สเลฟ
- รอรับการตอบกลับจาก PCF8574A
- ส่งข้อมูลไปยัง PCF8574A
- เรียกโปรแกรมย่อยสภาวะหยุด

2.8 การเชื่อมต่อกับโมดูลแอลซีดี (LCD Module)

ในโมดูลแอลซีดีนั้นจะมีส่วนประกอบ 3 ส่วนหลักดังนี้

- ตัวแสดงผล(Display) ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็นได้โดยอาศัยแสงจากภายนอก ดังนั้นจึงต้องมีมุมในการมองข้อมูลที่แสดงบนจอแอลซีดี

- ตัวควบคุม(Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอก มาควบคุมการทำงานของโมดูลแอลซีดี เช่น ลบจอภาพ แสดงตัวอักษรหรือเลื่อนเคอร์เซอร์เป็นต้น ตัวควบคุมนี้ใช้ชิบควบคุมโดยเฉพาะชิบ ที่นิยมใช้คือเบอร์ HD44780 และ HD61830 โดย HD44780 จะใช้ควบคุม LCD แบบอักขระ และ HD61830 ใช้ควบคุม LCD แบบกราฟฟิก

- ตัวขับ(Driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงข้อมูลตามที่กำหนด ชิบที่ใช้ทำงานเป็นตัวขับนี้ได้แก่ เบอร์ HD44100H และ MSM5259 เป็นต้น

2.8.1 โครงสร้างภายในของตัวควบคุมโมดูล LCD

ในการใช้งานโมดูล LCD จำเป็นต้องทำความเข้าใจเกี่ยวกับโครงสร้างและคำสั่งในการควบคุมให้ดีเสียก่อน โดยในที่นี้จะทำการแสดงตัวอย่าง โมดูล LCD แบบอักขระ เพราะสามารถเข้าใจง่าย โดยบล็อกไดอะแกรมภายในของชิบควบคุม LCD เบอร์ HD44780H ซึ่งใช้ในโมดูล LCD แบบอักขระประกอบด้วย

บัฟเฟอร์อินพุตเอาท์พุต เป็นส่วนที่ใช้ในการติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอก เพื่อที่จะถ่ายทอดข้อมูลเข้าออกภายในตัวควบคุม

รีจีสเตอร์คำสั่ง (Instruction Register:IR) เป็นรีจิสเตอร์ที่ใช้รับข้อมูลจากอุปกรณ์ภายนอก เพื่อถ่ายทอดต่อไปยังหน่วยความจำที่ทำหน้าที่เก็บข้อมูลแสดงผล หรือนำข้อมูลไปสร้างตัวอักษรเพิ่มเติมในแรมเก็บตัวอักษร

แรมเก็บข้อมูลแสดงผล(Display Data RAM:DDRAM) เป็นหน่วยความจำแบบแรมทำหน้าที่เก็บข้อมูลที่มาจากรีจิสเตอร์ DR ตัวควบคุมจะนำข้อมูลใน DDRAM นี้ไปเปิดตาราง (Look up table) ของตัวอักษรที่เก็บไว้ในหน่วยความจำรอมและแรมเก็บตัวอักษร เพื่อนำไปแสดงที่ตัวแสดงผล

รอมเก็บตัวอักษร (Character Generator ROM:CGROM)เป็นหน่วยความจำแบบรอมที่ใช้เก็บข้อมูลตัวอักษรหรือสัญลักษณ์ที่สามารถอ่านออกไปแสดงที่ตัวแสดงผลได้ มีขนาด 7200 บิต โดยจะถูกอ่านด้วยของข้อมูลใน DRAM

แรมเก็บตัวอักษร (Character Generator RAM:CGRAM) เป็นหน่วยความจำแรมที่ใช้เก็บอักษรที่มีการสร้างเพิ่มเติมขึ้นใหม่ ในกรณีที่ตัวอักษรใน CGROM ไม่เพียงพอ มีขนาด 512 บิต การเขียนและอ่านค่าไปใช้นั้นทำได้เช่นเดียวกับ CGROM คือ เขียนข้อมูลลงใน DDRAM แล้วตัวควบคุมจะมาอ่านค่าจาก CGROM เอง

แฟลกบิซซี(Busy Flag) เป็นส่วนที่ทำหน้าที่แจ้งสถานะการทำงานของตัวควบคุมให้อุปกรณ์ภายนอกทราบว่าตัวควบคุมพร้อมที่จะรับข้อมูลหรือคำสั่งหรือไม่

สำหรับโมดูล LCD ที่ใช้ในโครงงาน จะประกอบด้วยขาทั้งหมด 14 ขา แต่ละขามีหน้าที่ดังต่อไปนี้

Vss(ขา1) : ต่อกราวน์

Vdd(ขา2) : ต่อไฟเลี้ยง +5 v

Vo(ขา4) : เป็นอินพุตรับแรงดันเพื่อปรับความเข้มของการแสดงผล

RS(ขา4) : เป็นขาอินพุตใช้ในการแยกชนิดของข้อมูลที่ทำการประมวลผลในขณะนั้นว่าเป็นคำสั่งสำหรับรีจิสเตอร์ IR หรือเป็นข้อมูลสำหรับรีจิสเตอร์ DR โดยขานี้เป็น “0” ข้อมูลที่ส่งมาจะเป็นคำสั่งแต่ถ้าขานี้เป็น “1” ข้อมูลที่ส่งมาจะเป็นข้อมูลสำหรับการแสดงผล

R/W(ขา15) : เป็นขาที่ใช้เลือกการอ่านหรือเขียนข้อมูลจากโมดูล LCD ถ้าเป็น “0” เป็นการกำหนดให้เขียนข้อมูล แต่ถ้าเป็น “1” จะเป็นการอ่านข้อมูล

E(ขา6) : เป็นขาสำหรับรับสัญญาณพัลส์เอ็นเอเบิลโมดูล LCD ให้ทำงาน

D0-D7 : เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่าง LCD กับอุปกรณ์ภายนอกขนาด 8 บิต

(ขา 7-14) : อนึ่งขา RS,R/W และ E จะทำงานร่วมกัน โดยมีความสัมพันธ์แสดงดังตาราง

2.8.2 คำสั่งควบคุมโมดูล LCD

ในการเขียนคำสั่งลงในตัวควบคุม แน่นอนว่าต้องกำหนดให้ขา RS และ R/W เป็น “0” แล้วเขียนคำสั่งตามไป คำสั่งควบคุมโมดูล LCD ของชิปควบคุม HD44780 ที่สำคัญมี 10 คำสั่งดังนี้

- **คำสั่งเคลียร์ตัวแสดงผล (Clear display)** มีข้อมูลคำสั่งเป็น 01H เป็นคำสั่งที่ใช้เขียนข้อมูลช่องว่าง หรือ space เข้าไปใน DDRAM ทั้งหมด เมื่อตัวควบคุมเอ็กซิคิวต์คำสั่งนี้ จะทำการกำหนดแอดเดรสของ DDRAM เป็น 0 เคอร์เซอร์จะกลับไปอยู่ที่ตำแหน่งซ้ายมือสุด ของจอแสดงผล แล้วเซ็ทบิต I/D ให้เป็น “1”

- **คำสั่ง(Return home)** ต้องกำหนดให้บิต 1 ของข้อมูลเป็น “1” เป็นคำสั่งให้เคอร์เซอร์เคลื่อนที่กลับไปยังตำแหน่งซ้ายสุดของจอแสดงผล แต่ข้อมูลบนจอแสดงผลไม่เปลี่ยนแปลง นั่นคือข้อมูลคำสั่งของคำสั่งนี้จะเป็น 02H หรือ 03H ก็ได้

- **คำสั่งเลือกโหมดการป้อนข้อมูล (Entry mode set)** มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	0	0	1	I/D	S

รูปที่ 2.16 แสดงบิตข้อมูลของคำสั่งเลือกโหมดการป้อนข้อมูล

- บิต S เป็นบิตที่ใช้ในการกำหนดลักษณะของการแสดงผล เมื่อมีการป้อนข้อมูล ถ้าหากบิต S เป็น "1" เมื่อเกิดข้อมูลใหม่บนจอแสดงผล ตัวเคอร์เซอร์จะอยู่กับที่ แต่ตัวอักษรข้อมูลเดิมจะถูกดันไปทางซ้าย แต่ถ้าหากบิตนี้เป็น "0" เมื่อเกิดข้อมูลใหม่ตัวเคอร์เซอร์จะเลื่อนไปทางขวามือ

- บิต I/D เป็นบิตที่ใช้กำหนดว่า เมื่อเขียนหรืออ่านข้อมูลแล้ว แอดเดรสของ DDRAM เพิ่มขึ้นหรือลดลงหนึ่งแอดเดรส โดยถ้าบิตนี้เป็น "1" แอดเดรสของ DDRAM จะเพิ่มขึ้น แต่ถ้าเป็น "0" แอดเดรสจะลดลง ดังนั้นข้อมูลคำสั่งที่เกิดขึ้นสำหรับคำสั่งนี้ได้แก่ 04H-07H และที่ใช้บ่อยคือ 06H หมายถึง กำหนดให้เมื่อเกิดข้อมูลใหม่ เคอร์เซอร์จะเลื่อนไปทางขวามือ และแอดเดรสของ DDRAM เพิ่มขึ้น

2.8.3 คำสั่งควบคุมการแสดงผล

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	0	0	D	C	B

รูปที่ 2.17 แสดงบิตข้อมูลของคำสั่งควบคุมการแสดงผล

- บิต D ใช้ควบคุมการเปิดปิดจอแสดงผล ถ้าบิตนี้เป็น "1" จะเป็นการเปิดจอแสดงผล ถ้าเป็น "0" จะเป็นการปิดจอแสดงผล

- บิต C ใชควบคุมการแสดงตัวเคอร์เซอร์บนจอแสดงผล ถ้าต้องการให้มีเคอร์เซอร์แสดงผลบนจอแสดงผล ต้องกำหนดให้บิตนี้เป็น "1" ถ้ากำหนดให้เป็น "0" จะเป็นการปิดเคอร์เซอร์ หรือไม่แสดงเคอร์เซอร์

- บิต B ใช้ควบคุมการกระพริบของเคอร์เซอร์ ถ้าบิตนี้เป็น "1" เคอร์เซอร์จะกระพริบดังนั้นจะมีข้อมูลคำสั่งได้ตั้งแต่ 08H-0FH ที่ใช้บ่อยคือ 0CH เป็นการสั่งให้เปิดจอแสดงผล แต่ไม่แสดงเคอร์เซอร์ และ 0FH เป็นการสั่งให้เปิดจอแสดงผล แสดงเคอร์เซอร์ และสั่งให้เคอร์เซอร์กระพริบ

2.8.4 คำสั่งควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	0	1	S/C	R/L	*	*

รูปที่ 2.18 แสดงบิตข้อมูลของคำสั่งควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร

การควบคุมการเลื่อนเคอร์เซอร์และตัวอักษรบนจอแสดงผล ขึ้นอยู่กับการกำหนดบิต S/C และ R/L ซึ่งสามารถสรุปได้ดังนี้

ตารางที่ 2.8 แสดงการกำหนดบิต S/C และ R/L

S/C	R/L	ลักษณะการเลื่อน	ข้อมูลคำสั่ง
0	0	เลื่อนเคอร์เซอร์ไปทางซ้าย	10H-13H
0	1	เลื่อนเคอร์เซอร์ไปทางขวา	14H-17H
1	0	เลื่อนตัวอักษรใหม่ไปทางซ้าย	18H-1BH
1	1	เลื่อนตัวอักษรใหม่ไปทางขวา	1CH-1FH

2.8.5 คำสั่งกำหนดฟังก์ชั่นการทำงาน

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
0	0	1	DL	N	F	*	*

รูปที่ 2.19 แสดงบิตข้อมูลของคำสั่งกำหนดฟังก์ชั่นการทำงาน

- บิต DL ใช้กำหนดจำนวนบิตที่ใช้ติดต่อส่งผ่านข้อมูล ถ้าบิตนี้เป็น “0” จะเป็นการติดต่อแบบ 4 บิต แต่ถ้าบิตนี้เป็น “1” จะเป็นแบบ 8 บิต

- บิต N ใช้กำหนดจำนวนบรรทัดของการแสดงผล ถ้าเป็น “0” จะแสดงผล 1 บรรทัด ถ้าเป็น “1” จะแสดงผล 2 บรรทัด ในกรณี ในกรณีที่จอแสดงผลสามารถแสดงได้มากกว่า 2 บรรทัด และต้องการให้แสดงผลมากกว่า 2 บรรทัด ก็กำหนดบิต N นี้ให้เป็น “1” จุดที่น่าสังเกตคือ โมดูล LCD แบบ 16

ตัวอักษร 1 บรรทัด แม้จะมีบรรทัดการแสดงผลเพียง 1 บรรทัด แต่จะต้องกำหนด N ให้เป็น "1" เนื่องจากแอดเดรสของ DDRAM แบ่งเป็น 2 ช่อง คือ 00H และ 40H

- บิต F ใช้เลือกความละเอียดของตัวอักษรในการแสดงผล ถ้าบิตนี้เป็น "0" จะเป็นการแสดงผลแบบ 5x7 และถ้าเป็น "1" จะแสดงเป็นแบบ 5x10 จุด ข้อมูลคำสั่งที่ใช้บ่อยคือ 38H เป็นการกำหนดให้โมดูล LCD ทำงานในแบบ 8 บิต แสดงผล 4 บรรทัด และเลือกความละเอียดเป็น 5x7 จุด

คำสั่งเลือกแอดเดรสของ CGRAM ต้องกำหนดให้บิต 7 เป็น "0" บิต 6 เป็น "1" ส่วนอีก 6 บิตที่เหลือจะแทนด้วยค่าแอดเดรสของ CGRAM จะต้องทากรกำหนดแอดเดรสด้วยคำสั่งนี้ ก่อนจะอ่านหรือเขียนข้อมูลให้ CGRAM โดยแอดเดรสของ CGRAM อยู่ระหว่าง 00H-3FH

2.8.6 คำสั่งเลือกแอดเดรสของ DDRAM

ใช้ในการเลือกแอดเดรสของ DDRAM ก่อนที่จะทำการอ่านหรือเขียนข้อมูล โดยบิต 7 ต้องเป็น "1" และข้อมูลอีก 7 บิตที่เหลือจะเป็นคู่แอดเดรสของ DDRAM ซึ่งแอดเดรสของ DDRAM จะอยู่ระหว่าง 8CH-0FFH ทั้งนี้จำนวนแอดเดรสยังขึ้นกับการกำหนดสถานะบิตที่ N ด้วย เป็น "0" แอดเดรสของ DDRAM จะอยู่ระหว่าง 80H-0CFH และถ้าบิต N เป็น "1" แอดเดรสของ DDRAM จะมีสองช่วงคือ 8CH-87H และ 0C0H-0C7H

2.8.7 คำสั่งอ่านบิตซีแฟลกและแอดเดรส

มีรายละเอียดของรูปแบบข้อมูลคำสั่งดังนี้

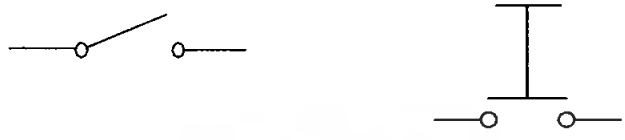
บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
BF	A	A	A	A	A	A	A

รูปที่ 2.20 แสดงบิตข้อมูลของคำสั่งอ่านบิตซีแฟลกและแอดเดรส

เป็นคำสั่งที่ใช้อ่านบิซซีแฟลก(BF) โดยแฟลกนี้จะเป็นตัวบอกสถานะของตัวควบคุม LCD ว่าพร้อมจะรับข้อมูลอยู่หรือไม่ ถ้าหากบิต BF เป็น "0" แสดงว่าพร้อมรับข้อมูลหรือคำสั่ง แต่ถ้าเป็น "1" แสดงว่า ขณะนี้ตัวควบคุม LCD ยังอยู่ในกระบวนการทำงานภายในหรือกำลังประมวลผลข้อมูลอยู่ ยังไม่พร้อมรับข้อมูลหรือคำสั่ง เมื่อต้องการอ่านค่าแฟลกต้องกำหนดให้ขา R/W เป็น "1" ด้วย แต่สัญญาณที่ RS ยังต้องเป็น "0" อยู่เพราะข้อมูลนี้เป็นข้อมูลคำสั่ง นอกจากนี้ ยังใช้เป็นคำสั่งอ่านข้อมูล แอดเดรสของ CGRAM และ DDRAM ด้วย

2.9 คีย์แพด (Keypad)

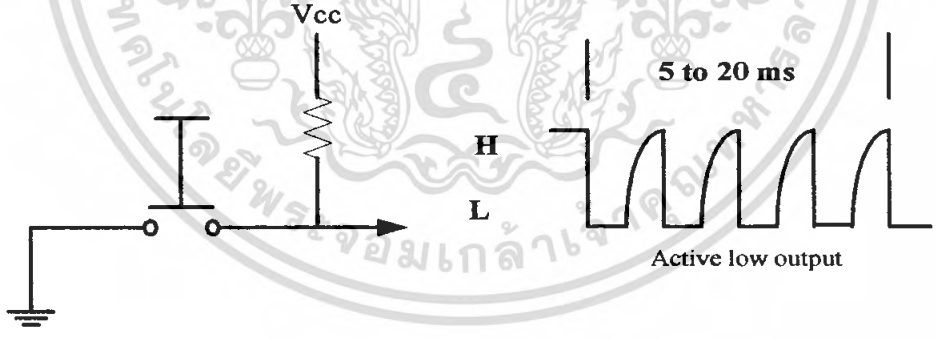
คีย์แพด (Keypad) คืออุปกรณ์อินพุตที่เกิดจากการนำสวิตช์หลายๆ ตัวมาต่อด้วยกันแบบเมทริกซ์ เสมือนเป็นคีย์บอร์ดขนาดเล็กอันหนึ่ง เช่น ต้องการคีย์แพดขนาด 4×4 จะใช้สวิตช์ทั้งหมด 16 ตัว โดยกลไกการทำงานของสวิตช์เมื่อกดปุ่มสวิตช์ จะทำให้เส้นลวดทองแดงสัมผัสกันเกิดการครบวงจรขึ้นดังรูปที่ 2.21



(1) Push-botton Switch (2) Toggle Switch

รูปที่ 2.21 การทำงานของสวิตช์

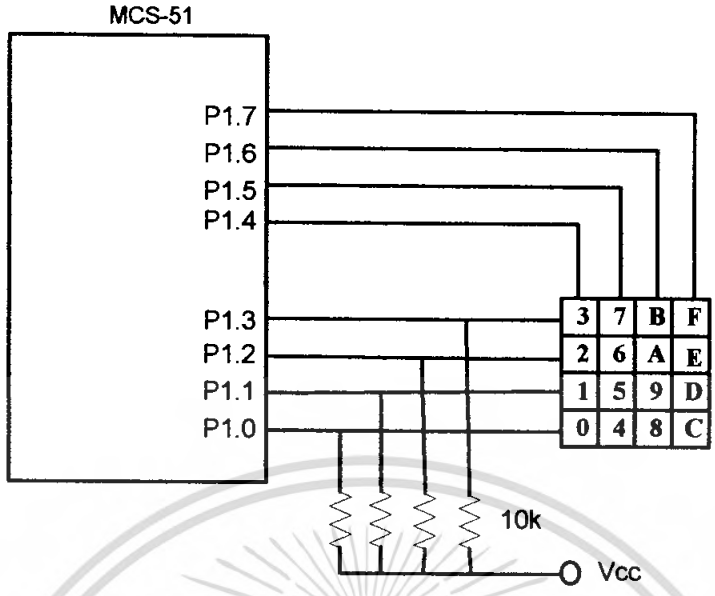
จากโครงสร้างการทำงานของสวิตช์จะมีปัญหาที่เรียกว่า Contact Bounce หรือการกดสัมผัส นั่นคือเมื่อเรากดสวิตช์ เอาต์พุตที่ได้จะเกิดเป็นพัลส์ขึ้นมาในช่วงแรกๆ ที่เป็นเช่นนี้เพราะว่าเวลาเรากดสวิตช์หน้าสัมผัสของสวิตช์ไม่ได้เชื่อมต่อทันทีทันใด แสดงดังรูปที่ 2.22



รูปที่ 2.22 แสดงการเกิดพัลส์เมื่อทำการกดสวิตช์

เพื่อให้ได้เอาต์พุตที่ถูกต้องจากการกดสวิตช์ ก่อนส่งไปยังไมโครคอนโทรลเลอร์เราต้องหน่วงเวลาไว้ระยะหนึ่งก่อนเพื่อแน่ใจว่าสวิตช์สัมผัสกันดีแล้ว กระบวนการนี้เรียกว่า Debouncing ดังนั้นในการใช้งานคีย์แพดจึงประกอบไปด้วย 3 ขั้นตอน

- **Keypad Scanning** สำหรับตรวจสอบว่าสวิตช์ไหนถูกกด เนื่องจากคีย์แพดประกอบด้วยสวิตช์จำนวนหลายตัวมาต่อกันเป็นเมทริกซ์ และใช้เทคนิคการถอดรหัส และการเลือกว่าคีย์ไหนถูกกดให้ดูคีย์ในแนวที่แถวและคอลัมน์ตรงกัน



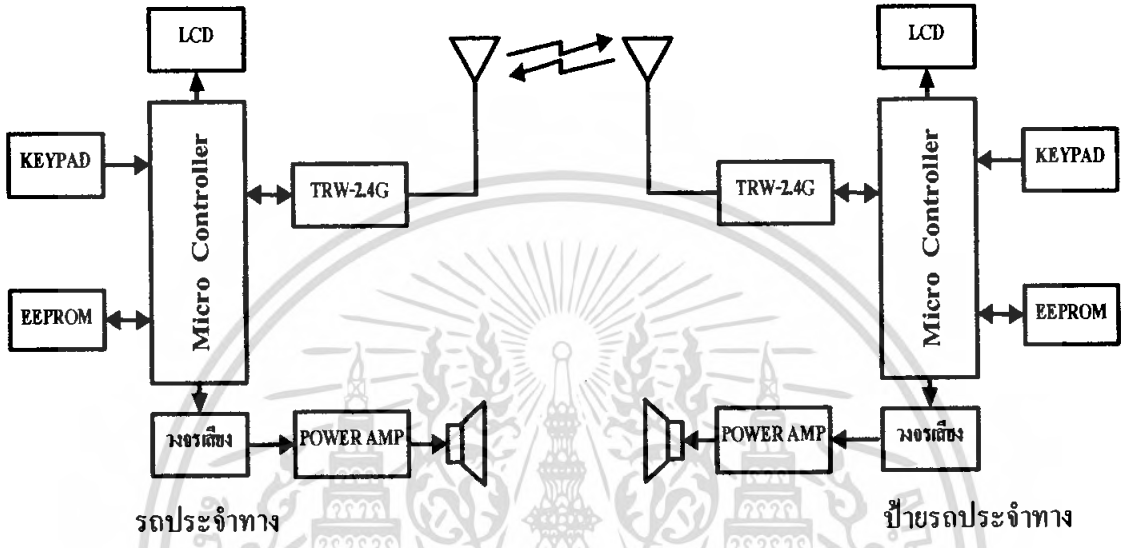
รูปที่ 2.23 รูปแสดงการเชื่อมต่อคีย์แพดเข้ากับตัวไมโครคอนโทรลเลอร์

- **Keypad Debouncing** เพื่อความแน่ใจว่าสวิตช์นั้นถูกกดจริง เมื่อเรากดสวิตช์เอาต์พุตที่ได้จะเกิดเป็นพัลส์ขึ้นมาในช่วง 5 ms แรก และเกิดพัลส์เมื่อไม่กดหรือปล่อยสวิตช์ภายใน 20 ms ดังนั้นจึงให้คิดว่าสวิตช์ถูกกดหลังจากแรงดันเป็น low ประมาณ 10 ms และสวิตช์ถูกปล่อยหลังจากแรงดันเป็น high ประมาณ 10 ms

- **Table Lookup** สำหรับหารหัสแอสกี้ (ASCII Code) ของคีย์ที่กดนั้น เพื่อส่งให้ไมโครคอนโทรลเลอร์ หลังจากคีย์ถูกกดโปรแกรมก็จะหารหัสแอสกี้ (ASCII) ของคีย์ที่เรากดในตารางรหัสแอสกี้ (ASCII Code Table) จากนั้นจะส่งรหัสที่ได้ไปยังซีพียู

บทที่ 3
การออกแบบและการสร้าง

3.1 ส่วนประกอบของโครงงาน



รูปที่ 3.1 แสดง Block diagram ในส่วยของรถประจำทาง และป้ายรถประจำทาง

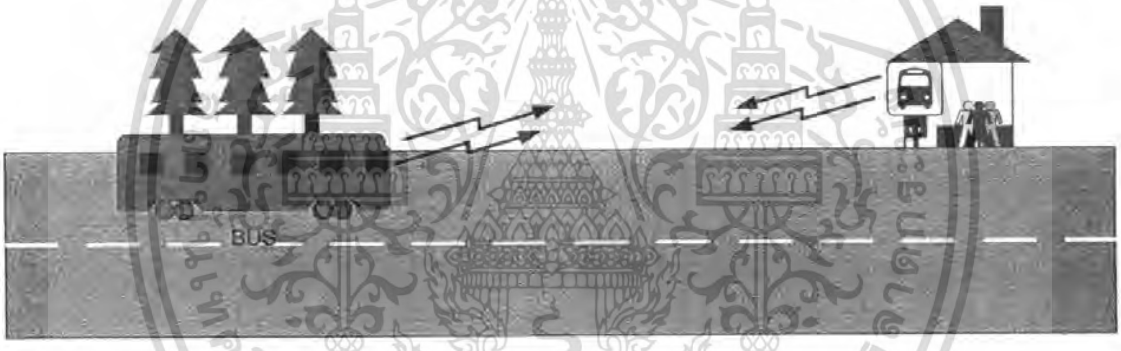
จากรูปที่ 3.1 จะแสดงบล็อกไดอะแกรม (Block diagram) ในส่วนของรถประจำทาง และป้ายรถประจำทาง ซึ่งจะมีไมโครคอนโทรเลอร์เป็นหน่อยประมวลผลกลาง (CPU) ซึ่งจะทำหน้าที่เป็นตัวควบคุมอุปกรณ์ต่าง ๆ ที่นำมาต่อร่วมกัน คือ คีย์แพด (Key pad) ,จอแสดงข้อความ (LCD) ,ส่วนเก็บข้อมูล (EEPROM) , RF Module (TRW-2.4GHz) , วงจรบันทึกเสียง และ วงจรขยายเสียง ซึ่งหลักการทำงานของวงจรต่าง ๆ สามารถอธิบายได้ดังนี้

- ส่วนเก็บข้อมูล (EEPROM) จะเอาไว้สำหรับเก็บข้อมูล โดยในส่วนของป้ายรถประจำทางจะใช้ในการเก็บรหัสหมายเลขของป้ายรถประจำทาง และในส่วนของรถประจำทางจะใช้ในการเก็บรหัสหมายเลขของรถประจำทาง
- ส่วนคีย์แพด (Key pad) จะเป็นส่วนที่ใช้ในการป้อนข้อมูลเข้าไปตามที่ต้องการ โดยเมื่อทำการป้อนค่าเข้าไปจะไปทำการกำหนดค่าต่าง ๆ โดยดึงเอาโปรแกรมต่าง ๆ ที่ถูกเขียนอยู่ในไมโครคอนโทรเลอร์ออกมาใช้ กล่าวคือจะใช้ในการเลือกเมนูการทำงานต่าง ๆ ที่ถูกเขียนอยู่ในไมโครคอนโทรเลอร์ออกมาใช้ การใส่ข้อมูลต่าง ๆ ลงในส่วนเก็บข้อมูล
- ส่วนของจอแสดงผล (LCD) จะเป็นส่วนในการแสดงข้อความต่าง ๆ ให้สามารถดำเนินการในแต่ละส่วนได้อย่างถูกต้อง

- ส่วนของวงจรเสียง ในวงจรนี้ได้ทำการบันทึกเสียงตามรสัสต่าง ๆ ไว้แล้ว โดยเมื่อรถประจำทางสายใดวิ่งเข้ามาที่ป้าย ป้ายก็จะแสดงเสียงของหมายเลขรถคันนั้นตามรหัสที่ได้บันทึกไว้แล้วออกมา โดยนำแต่ละเสียงมาเรียงต่อกันเพื่อให้ได้เป็นหมายเลขของรถประจำทาง และในส่วนของรถประจำทางก็จะแสดงผลเป็นเสียงของชื่อป้ายที่อยู่ข้างหน้าตามรหัสที่รับเข้ามา

- ส่วนของ RF Module (TRW-2.4GHz) มีไว้เพื่อทำการ รับ-ส่ง ข้อมูล ในภาคส่งจะทำการส่งข้อมูลออกไปยังอากาศ โดยข้อมูลที่ส่งจะได้จากไมโครคอนโทรเลอร์ ซึ่งข้อมูลที่มาจากไมโครคอนโทรเลอร์จะเป็นข้อมูลต่าง ๆ ที่ได้เก็บบันทึกไว้ในส่วนเก็บข้อมูลหรือ EEPROM ในส่วนของภาครับจะทำการรับข้อมูลที่ภาคส่งส่งมาทางอากาศ แล้วจะทำการส่งข้อมูลต่อไปยังไมโครคอนโทรเลอร์เพื่อทำการประมวลผลต่อไป

3.2 หลักการทำงาน



รูปที่ 3.2 แสดงหลักการทำงานของรถประจำทางและป้ายรถประจำทาง

โดยหลักการทำงานของโครงงานนี้คือ

- จะเริ่มต้นจากที่ป้ายรถประจำทางจะทำการวนส่ง ข้อมูลรหัสที่ใช้ในการส่งให้รถ และข้อมูลของหมายเลขป้ายรถประจำทางออกไป

- เมื่อรถประจำทางวิ่งเข้าใกล้ยังป้ายรถประจำทางในรัศมีของเครื่องส่ง รถประจำทางก็จะทำการรับข้อมูลที่ส่งมาจากป้ายเข้ามาตรวจสอบดูว่าตรงกับ ข้อมูลรหัสที่ใช้ในการส่งให้รถหรือไม่ ถ้าไม่ตรงก็จะกลับไปรอรับข้อมูลใหม่ แต่ถ้าตรงก็จะนำเอาข้อมูลของหมายเลขป้ายไปตรวจสอบว่าตรงกับหมายเลขของป้ายในเส้นทางที่รถวิ่งอยู่หรือไม่ ถ้าไม่ตรงก็จะกลับไปรอรับข้อมูลใหม่ แต่ถ้าตรงก็จะทำการแสดงผลเป็นข้อความเสียงของชื่อป้ายตามรหัสที่รับเข้ามาที่ได้บันทึกไว้ใน IC บันทึกเสียง

- เมื่อรถประจำทางแสดงผลเสร็จแล้ว ก็จะส่งข้อมูลกลับไปยังป้าย โดยข้อมูลที่ส่งไปนั้นจะประกอบไปด้วย ข้อมูลรหัสที่ใช้ในการส่งให้ป้าย ข้อมูลของหมายเลขป้ายนั้น และข้อมูลรหัสหมายเลขรถของตัวเอง แล้วก็รอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถช่วงเวลาหนึ่ง ซึ่งเมื่อเลยช่วงเวลาที่กำหนดไว้แล้ว

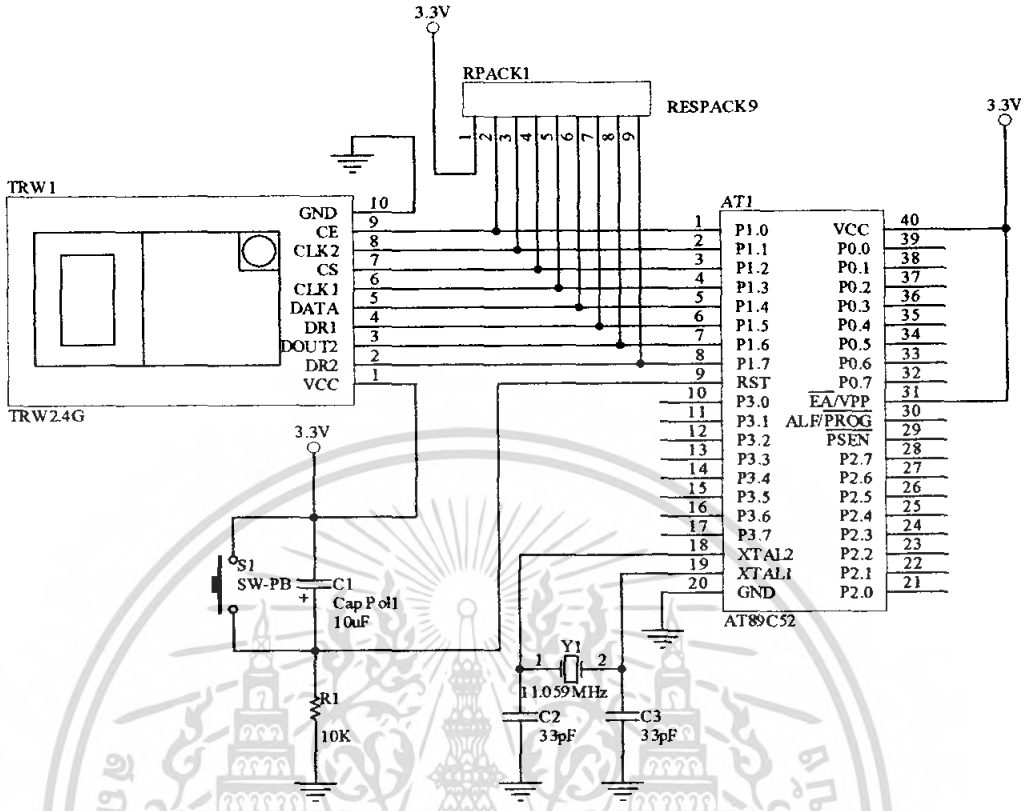
จึงจะกลับไปเริ่มต้นรอรับข้อมูลที่เป็นรหัสที่ใช้ส่งให้รถใหม่ โดยรถประจำทางจะไม่รับข้อมูลของป้ายที่ทำการแสดงผลไปแล้วอีก

- เมื่อป้ายได้รับข้อมูลก็จะตรวจสอบดูว่าตรงกับข้อมูลรหัสที่ใช้ในการส่งให้ป้ายหรือไม่ ถ้าไม่ตรงก็จะกลับไปวนส่งข้อมูลใหม่ แต่ถ้าตรงก็จะตรวจสอบดูว่าตรงกับรหัสของป้ายตัวเองหรือไม่ ถ้าไม่ตรงก็จะกลับไปวนส่งข้อมูลใหม่ แต่ถ้าตรงก็จะนำเอาข้อมูลของหมายเลขรถไปทำการแสดงผลเป็นข้อความเสียงของหมายเลขรถ ตามรหัสที่รับเข้ามาที่ได้บันทึกไว้ใน IC บันทึกเสียง แล้วก็กลับไปวนส่งข้อมูลใหม่

- กรณีที่ป้ายรถประจำทางมีผู้ต้องการที่จะขึ้นรถประจำทางสายที่แสดงผลอยู่ก็สามารถทำการติดต่อกับรถคันนั้นได้ โดยทำการกดปุ่มเรียกรถที่อยู่บนตัวกล่องอุปกรณ์ตามหมายเลขปุ่มที่ป้ายกำหนด ซึ่งที่ปุ่มกดนั้นจะมีตัวอักษรเบรลล์ไว้ เพื่ออำนวยความสะดวกแก่ผู้พิการทางสายตา ซึ่งป้ายจะทำการส่งข้อมูลที่เป็นข้อมูลในการเรียกรถคันนั้นออกไป โดยข้อมูลที่ส่งไปนั้นจะประกอบไปด้วย ข้อมูลรหัสที่ใช้ในการเรียกรถ ข้อมูลของหมายเลขรถคันนั้น และข้อมูลของหมายเลขป้ายที่ทำการติดต่อ

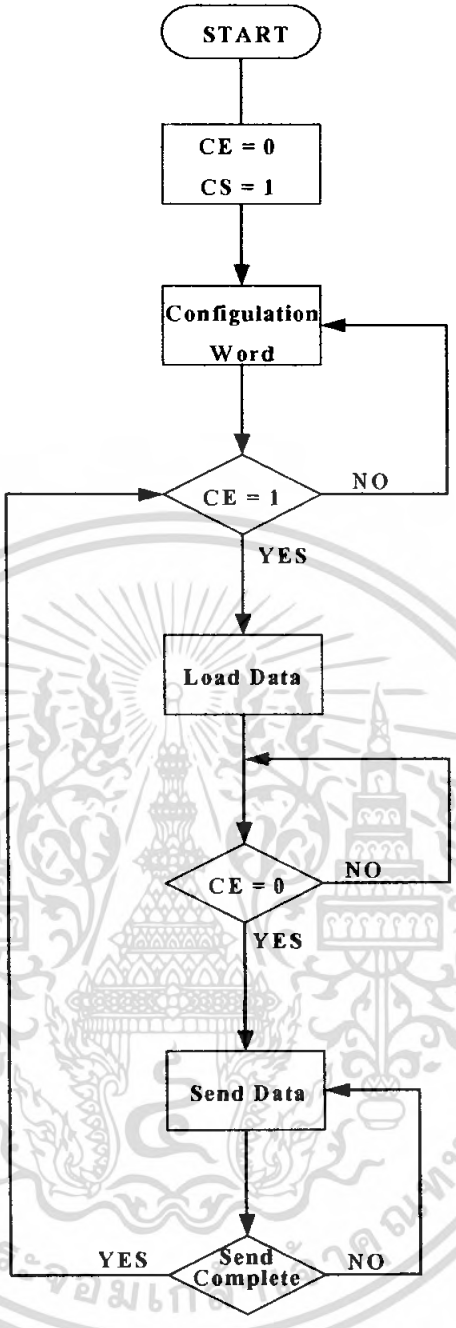
- เมื่อทางด้านรถประจำทางได้รับข้อมูลที่เป็นรหัสที่ใช้เรียกรถเข้ามา ก็จะนำข้อมูลไปตรวจสอบดูว่าตรงกับข้อมูลของหมายเลขรถตัวเองหรือไม่ ถ้าไม่ตรงก็กลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถใหม่ แต่ถ้าตรงก็จะตรวจสอบดูว่าข้อมูลที่รับเข้ามาตรงกับหมายเลขป้ายในเส้นทางที่รถวิ่งอยู่หรือไม่ ถ้าไม่ตรงก็กลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถใหม่ แต่ถ้าตรงก็จะแสดงผลให้คนขับรถรู้ว่ามีคนต้องการจะขึ้นรถที่ป้ายหน้า แล้วก็กลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถใหม่จนกว่าจะเลยเวลาที่กำหนด เมื่อเลยเวลาที่กำหนดไว้รถประจำทางก็จะกลับไปเริ่มต้น รอรับข้อมูลที่เป็นรหัสที่ใช้ส่งให้รถใหม่ โดยรถประจำทางจะไม่รับข้อมูลของป้ายที่ทำการแสดงผลไปแล้วอีก

3.3 การออกแบบภาคส่ง



รูปที่ 3.3 แสดงวงจรของภาคส่งและภาครับ

จากรูปที่ 3.3 เป็นวงจรทางด้านภาคส่ง โดยใช้พอร์ต 1 ของไมโครคอนโทรเลอร์ทั้งหมดต่ออยู่กับโมดูลความถี่วิทยุ ซึ่งโมดูลความถี่วิทยุเป็นตัวส่งข้อมูลโดยใช้การมอดูเลตข้อมูลแบบ GFSK (Gaussian Frequency Shift Keying) ซึ่งสามารถกำหนดอัตราการส่งผ่านข้อมูลได้โดยต้องใช้การเขียนโปรแกรมให้กับตัวไมโครคอนโทรลเลอร์ โดยลักษณะของโปรแกรมที่ใช้เขียนควบคุมตัวโมดูลความถี่วิทยุทางด้านภาคส่ง สามารถเขียนออกมาเป็นรูปแบบของผังงานได้ดัง รูปที่ 3.4



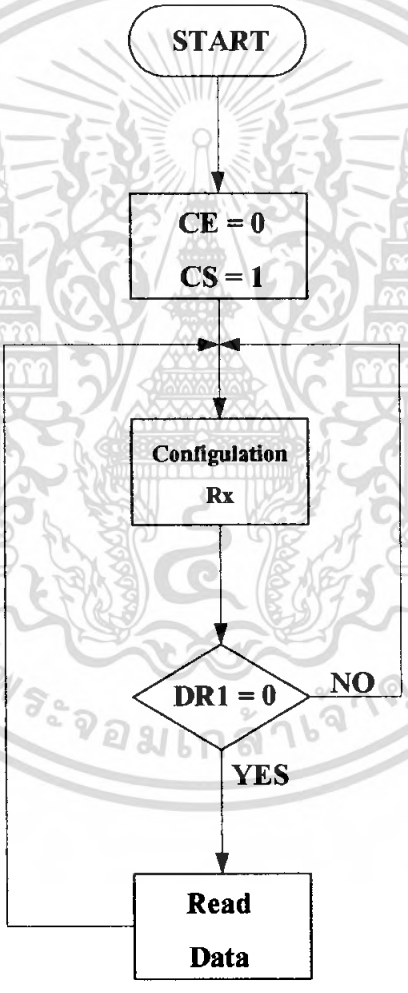
รูปที่ 3.4 รูปแบบผังงานแสดงการเขียนโปรแกรมควบคุมโมดูลความถี่วิทยุทางด้านภาคส่ง

จากรูปที่ 3.4 เป็นผังงานในส่วนควบคุมโมดูลความถี่วิทยุทางด้านภาคส่งซึ่งมีลักษณะการทำงานคือต้องทำการกำหนดค่าให้กับโมดูลความถี่วิทยุโดยทำการกำหนดค่าลงในเฟรมข้อมูล เพื่อให้ตัวโมดูลความถี่วิทยุทราบก่อนว่าต้องการให้โมดูลความถี่วิทยุทำงานเป็นตัวส่ง ซึ่งมีจำนวนทั้งหมด 144 บิต ในการที่จะทำการส่งเฟรมข้อมูลต้องทำการกำหนดให้ขา CE มีสถานะ " low " และ CS มีสถานะ " high " ก่อน เมื่อทำการกำหนดเรียบร้อยแล้ว เมื่อต้องการที่จะส่งข้อมูลต้องทำการป้อนข้อมูลให้กับโมดูลความถี่วิทยุก่อน โดยต้องตั้งค่าให้ CE มีสถานะ " high " เพื่อโหลดข้อมูลไปเก็บไว้ภายในตัวโมดูลความถี่วิทยุ

หลังจากนั้นเมื่อส่งข้อมูลให้กับโมดูลความถี่วิทยุเรียบร้อยแล้วให้ตั้งค่า CE ให้อยู่ในสถานะ “ low ” เพื่อกระตุ้นให้โมดูลความถี่วิทยุทำการส่งข้อมูลออกไป

3.4 การออกแบบภาครับ

จากรูปที่ 3.3 เป็นวงจรทางด้านภาครับ โดยใช้พอร์ต 1 ของไมโครคอนโทรลเลอร์ต่อกับขาของตัวโมดูลความถี่วิทยุ เมื่อมีสัญญาณจากตัวส่งเข้ามาโมดูลความถี่วิทยุจะทำหน้าที่รับสัญญาณดังกล่าวเข้ามา เพื่อให้ทางด้านตัวไมโครคอนโทรลเลอร์ทำการเก็บข้อมูลที่ส่งมาจากด้านตัวส่งแล้วนำข้อมูลมาทำการประมวลผลต่อไป ซึ่งทางภาครับต้องมีการโปรแกรมให้กับตัวไมโครคอนโทรลเลอร์เหมือนกับทางด้านตัวส่งซึ่งสามารถเขียนออกมาเป็นผังงานได้ดังรูปที่ 3.5



รูปที่ 3.5 รูปแบบผังงานแสดงการเขียนโปรแกรมควบคุมโมดูลความถี่วิทยุทางด้านภาครับ

จากรูปที่ 3.5 เป็นผังงานในส่วนควบคุมโมดูลความถี่วิทยุทางด้านภาครับซึ่งมีลักษณะการทำงานคือต้องทำการกำหนดค่าให้กับโมดูลความถี่วิทยุโดยทำการกำหนดค่าลงในเฟรมข้อมูล เพื่อให้ตัวโมดูลความถี่วิทยุทราบก่อนว่าต้องการให้โมดูลความถี่วิทยุทำงานเป็นตัวรับ ซึ่งมีจำนวนทั้งหมด 144 บิต

ในการที่จะทำการกำหนดเฟรมข้อมูลต้องทำการกำหนดให้ขา CE มีสถานะ " low " และ CS มีสถานะ " high " ก่อน เมื่อทำการกำหนดเรียบร้อยแล้ว ทางด้านฝ่ายรับจะทำการรอรับข้อมูลโดยการตรวจเช็คขา DR1 ซึ่งขา DR1 เป็นขาที่แสดงความพร้อมที่จะรับข้อมูลเมื่ออยู่ในสถานะ " low " ดังนั้นเมื่อขา DR1 ของโมดูลความถี่วิทยุอยู่ในสถานะ " low " เมื่อมีข้อมูลเข้ามาจึงสามารถรับข้อมูลได้ทันทีในขณะทำการรับข้อมูล DR1 จะอยู่ในสถานะ " high " เมื่อรับข้อมูลเรียบร้อยแล้ว DR1 จะกลับมาอยู่ในสถานะ " low " อีกครั้งเพื่อเตรียมพร้อมที่จะรับข้อมูลที่จะมีเข้ามาใหม่ โดยข้อมูลที่รับมาจะทำการป้อนให้กับไมโครคอนโทรลเลอร์เพื่อให้ไมโครคอนโทรลเลอร์ทำการประมวลผลและทำการแสดงผลต่อไป

3.5 การตั้งค่าโมดูลความถี่วิทยุ (TRW-2.4)

3.5.1 RF_CH# กับ RXEN

ตารางที่ 3.1 แสดงตำแหน่งบิตของช่องสัญญาณความถี่และโหมดการทำงาน

ชื่อ	RF_CH#							RXEN
ตำแหน่งบิต	7	6	5	4	3	2	1	0

จากตารางที่ 3.1 บิต RXEN เป็นบิตอยู่ที่บิตศูนย์เป็นบิตที่ให้กำหนดค่าให้ตัวโมดูลความถี่วิทยุให้ทำงานในภาครับหรือทำงานในภาคส่ง ถ้าต้องการให้โมดูลทำหน้าที่เป็นตัวส่งต้องกำหนดให้บิต RXEN เป็น "0" แต่ถ้าต้องการให้โมดูความถี่วิทยุทำหน้าที่เป็นตัวรับต้องกำหนดให้บิต RXEN เป็น "1"

ส่วน RF_CH# เป็นตำแหน่งบิตที่ 1 ถึงบิต 7 จะเป็นบิตที่กำหนดความถี่ที่ใช้ รับ/ส่ง ซึ่งมีความถี่ในช่วง 2400 MHz – 2524 MHz โดยคำนวณได้จากสูตรดังนี้

ช่องสัญญาณความถี่ = 2400 MHz + [(RF_CH#) (1.0 MHz)]

ในโครงงานนี้ในตัวส่งได้กำหนดค่าไว้ดังนี้

ตารางที่ 3.2 แสดงการกำหนดค่าให้กับตัวส่ง

RF_CH#							RXEN
0	0	0	1	0	1	0	0

ซึ่งเมื่อนำค่าใน RF_CH# ซึ่งก็คือค่าในบิต 1–7 มาทำการแปลงเป็นเลขฐานสิบ ในโครงงานนี้ได้ค่าเป็นเลขฐานสิบเท่ากับ 10 แล้วนำเลขฐานสิบไปคูณกับ 1.0 MHz แล้วนำไปบวกกับ 2400 MHz จะได้เป็นค่าความถี่ที่ใช้ส่งสัญญาณ ซึ่งถ้าต้องการที่จะไม่ใช้ความถี่นี้ก็สามารถทำการคำนวณได้ดังตัวอย่างแต่ค่าความถี่ต้องอยู่ในช่วง 2400 MHz ถึง 2524 MHz และถ้าต้องการที่จะทำการส่งสองช่องสัญญาณสามารถคำนวณหาความถี่ที่ใช้ได้จากสูตรดังนี้

ช่องสัญญาณความถี่ = 2400 MHz + [(RF_CH#) (1.0 MHz)] + 8 MHz

ส่วนทางด้านภาครับของโครงงานนี้ได้กำหนดค่าดังนี้

ตารางที่ 3.3 แสดงการกำหนดค่าให้กับตัวรับ

RF_CH#							RXEN
0	0	0	1	0	1	0	1

3.5.2 PF_PWR

ตารางที่ 3.4 แสดงการกำหนดค่ากำลังงานของภาคส่ง

กำลังงานของภาคส่ง		
D9	D8	P(dBm)
0	0	-20
0	1	-10
1	0	-5
1	1	0

ในโครงงานนี้ได้ทำการเลือกค่ากำลังงานของภาคส่งเท่ากับ 0 dBm บิตที่ 8 และ 9 จึงได้รหัสคือ 11

3.5.3 XO_F

ตารางที่ 3.5 แสดงการกำหนดค่าสถานะของตำแหน่งบิตที่ 10 ถึง 12

ตำแหน่งบิต	12	11	10
สถานะ	0	1	1

3.5.4 RFDR_SB

จะใช้ตำแหน่งบิตที่ 13 โดยเป็นบิตที่ใช้เลือกค่าอัตราการส่งผ่านข้อมูลของโมดูลความถี่วิทยุ

สถานะ 0 : อัตราการส่งผ่านข้อมูลเท่ากับ 250 Kbps

สถานะ 1 : อัตราการส่งผ่านข้อมูลเท่ากับ 1 Mbps

ในโครงงานนี้ได้กำหนดอัตราการส่งผ่านข้อมูลไว้ 1 Mbps

3.5.5 CM

ตำแหน่งบิตที่ 14 เป็นบิตที่เลือกโหมดการส่งข้อมูลกำหนดให้มีสถานะเป็น 1 เนื่องจากส่งข้อมูลในโหมด ShockBurst

3.5.6 RX2_EN

ตำแหน่งบิตที่ 15 เป็นบิตที่เลือกจำนวนช่องสัญญาณของภาครับ

สถานะ 0 : จำนวน 1 ช่องสัญญาณ

สถานะ 1 : จำนวน 2 ช่องสัญญาณ

ในโครงงานนี้ได้มีการกำหนดใช้ช่องสัญญาณเพียง 1 ช่องสัญญาณดังนั้นในตำแหน่งนี้จึงได้กำหนดไว้เป็น 0 ดังนั้นบิตที่ 8 – 15 จะได้ค่าที่กำหนดคือดังตารางที่ 3.6

ตารางที่ 3.6 แสดงกำหนดสถานะบิตตำแหน่งที่ 8 – 15

ชื่อ	RX2_EN	CM	REDR_SB	OX_F			RF_RWR	
ตำแหน่งบิต	15	14	13	12	11	10	9	8
สถานะ	0	1	1	0	1	1	1	1

3.5.7 CRC_EN

เป็นตำแหน่งบิตที่ 16 ใช้ในกำหนดให้มีการสร้าง CRC สำหรับภาคส่ง และให้มีการตรวจสอบบิต CRC สำหรับภาครับ ถ้ากำหนดให้สถานะบิตเป็น 1

3.5.8 CRC_L

เป็นตำแหน่งบิตที่ 17 ในการกำหนดความกว้างของบิต CRC

สถานะ 0 : ให้ CRC มีความกว้าง 8 บิต

สถานะ 1 : ให้ CRC มีความกว้าง 16 บิต

ในโครงงานนี้ได้มีการกำหนดให้มีสถานะเป็น 1

3.5.9 ADDR_W

เป็นตำแหน่งที่ 18 – 23 ใช้ในการกำหนดจำนวนที่จองไว้สำหรับแอดเดรสภาครับโดยที่จองไว้สูงสุดถึง 40 บิต (5 ไบต์) กำหนดให้มีจำนวนการจองบิตสูงสุดจะต้องกำหนดค่าดังแสดงในตารางดังต่อไปนี้

ตารางที่ 3.7 แสดงตารางการกำหนดค่า ADDR_W

ชื่อ	ADDR_W					
ตำแหน่งบิต	23	22	21	20	19	18
สถานะ	1	0	1	0	0	0

และการกำหนดค่าตั้งแต่ตำแหน่งบิตที่ 16–23 กำหนดได้ดังนี้

ตารางที่ 3.8 ตารางการกำหนดค่าในตำแหน่งบิตที่ 16–23

ชื่อ	ADDR_W						CRC_L	CRC_EN
ตำแหน่งบิต	23	22	21	20	19	18	17	16
สถานะ	1	0	1	0	0	0	1	1

3.5.10 DDRx

เป็นตำแหน่งบิตที่ 24–63 ขนาด 40 บิต (5 ไบต์) ใช้ในการกำหนดแอดเดรสให้กับตัวรับในช่องสัญญาณที่ 1 คือ ADDR1 และตำแหน่งบิตที่ 64–103 (5 ไบต์) ใช้ในการกำหนดแอดเดรสตัวรับในช่องสัญญาณที่ 2 คือ ADDR2 ได้มีการกำหนดให้มีแอดเดรสดังนี้

ตารางที่ 3.9 ตารางแสดงการกำหนดค่าในตำแหน่งบิตที่ 24-63

ชื่อ	ADDRx							
ตำแหน่งบิต	63	62	61	60	59	58	57	56
สถานะ	0	0	0	0	0	0	0	0
ตำแหน่งบิต	55	54	53	52	51	50	49	48
สถานะ	0	0	0	0	0	0	0	0
ตำแหน่งบิต	47	46	45	44	43	42	41	40
สถานะ	0	0	0	0	0	0	0	0
ตำแหน่งบิต	39	38	37	36	35	34	33	32
สถานะ	0	0	0	0	0	0	0	0
ตำแหน่งบิต	31	30	29	28	27	26	25	24
สถานะ	0	0	0	0	0	0	0	1

3.5.11 DATAx_W

เป็นตำแหน่งบิตที่ 104-111 ขนาด 8 บิตในการกำหนดความกว้างของบิตข้อมูลช่อง สัญญาณที่ 1 ซึ่งก็คือ DATA1_W และใช้ตำแหน่งบิตที่ 112–119 ขนาด 8 บิตในการกำหนดความกว้างของข้อมูลของช่องสัญญาณที่ 2 โดยในโครงงานนี้ได้มีการกำหนดค่าดังนี้

ตารางที่ 3.10 ตารางแสดงจำนวนความกว้างของบิตข้อมูลในช่องสัญญาณที่ 1

DATA1_W								
ตำแหน่งบิต	111	110	109	108	107	106	105	104
สถานะ	0	0	0	0	1	0	0	0

ตารางที่ 3.11 ตารางแสดงจำนวนความกว้างของบิตข้อมูลในช่องสัญญาณที่ 2

DATA2_W								
ตำแหน่งบิต	119	118	117	116	115	114	113	112
สถานะ	0	0	0	0	1	0	0	0

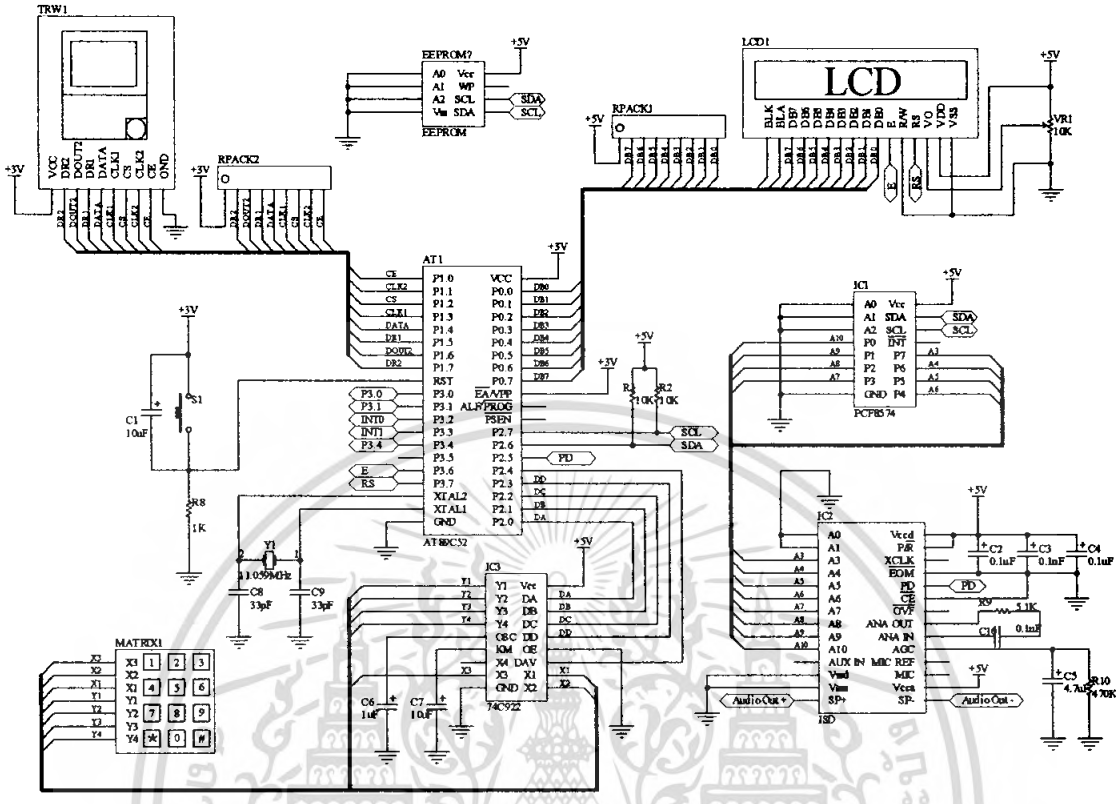
3.5.12 TEST

เป็นตำแหน่งบิตที่ 120–143 ขนาด 3 ไบต์โดยเป็นจำนวนบิตที่จองไว้สำรอง โดยได้กำหนดค่าดังนี้

ตารางที่ 3.12 ตารางแสดงการกำหนดค่าให้กับบิตที่ 120–143

ชื่อ	TEST							
ตำแหน่งบิต	143	142	141	140	139	138	137	136
สถานะ	1	0	0	0	1	1	1	0
ตำแหน่งบิต	135	134	133	132	131	130	129	128
สถานะ	0	0	0	0	1	0	0	0
ตำแหน่งบิต	127	126	125	124	123	122	121	120
สถานะ	0	0	0	1	1	1	0	0

3.6 การเชื่อมต่ออุปกรณ์ต่าง ๆ กับไมโครคอนโทรเลอร์



รูปที่ 3.6 การเชื่อมต่ออุปกรณ์ต่าง ๆ เข้ากับไมโครคอนโทรเลอร์

จากรูปที่ 3.6 แสดงการเชื่อมต่อคีย์แพด 4 x 3 กับไมโครคอนโทรเลอร์ โดยจะมี IC Scankey เบอร์ 74C922 ซึ่งเป็น 16 K encoder เป็นตัวเชื่อมต่อระหว่างคีย์แพด 4 x 3 กับไมโครคอนโทรลเลอร์ โดยที่เอาท์พุทของไอซีนี้จะมี 4 เส้น เพื่อต่อกับไมโครคอนโทรลเลอร์ดังรูปที่ 3.6 ข้อดีของ IC Scankey นี้คือจะช่วยลดความยุ่งยากในการเขียนโปรแกรม ทำให้โปรแกรมมีขนาดสั้นลง ไม่เปลือง Flash Memory ของไมโครคอนโทรลเลอร์

การเชื่อมต่อ LCD ในโครงงานนี้เราจะใช้ LCD แบบ 16 ตัวอักษร 2 บรรทัด แบบ 8 บิต ในการเลือกการต่อร่วมกับไมโครคอนโทรลเลอร์ มีทั้งให้เลือกแบบ 8 บิต และ 4 บิต ซึ่งจะมีข้อดีและข้อเสียคือ

- ความเร็วในการแสดงผลแบบ 8 บิตจะเร็วกว่า แบบ 4 บิต
- การใช้สายสัญญาณติดต่อกับไมโครคอนโทรลเลอร์ แบบ 4 บิต จะใช้น้อยกว่าแบบ 8 บิต

ในส่วนของหน่วยความจำ ในโครงงานนี้เราจะใช้สำหรับเก็บข้อมูลต่าง ๆ ที่คีย์เข้าไป โดยในส่วนของป้ายจะใช้เก็บข้อมูลของหมายเลขป้าย และในส่วนของตัวรถจะใช้เก็บข้อมูลของหมายเลขรถ โดยในโครงงานนี้เราจะใช้ IC เบอร์ 24LC16 ซึ่งเป็นหน่วยความจำแบบนอน - โวลาไทล์ (non - volatile) คือสามารถเก็บข้อมูลอยู่ได้โดยไม่ต้องจ่ายไฟเลี้ยงสามารถเขียน - อ่าน - ลบ ได้ด้วยสัญญาณไฟฟ้า ซึ่งจะใช้การเชื่อมต่อแบบระบบบัส I^2C มีขนาดของหน่วยความจำ เท่ากับ 2 กิโลไบท์

ในส่วนของการแสดงผลทางเสียงจะใช้ IC บันทึกเสียงเบอร์ ISD25120 ซึ่งสามารถบันทึกเสียงได้นาน 120 วินาที ต่อกับ IC ขยายพอร์ตเบอร์ PCF8574A ซึ่งจะใช้การเชื่อมต่อกับไมโครคอนโทรเลอร์แบบระบบบัส I^2C โดย IC บันทึกเสียงได้กำหนดตำแหน่ง address ที่ทำการบันทึกคำพูดไว้ดังนี้

ตารางที่ 3.13 ตารางแสดงคำพูดที่บันทึกเสียงในแต่ละแอดเดรส ของ IC บันทึกเสียงที่ป้ายรถประจำทาง

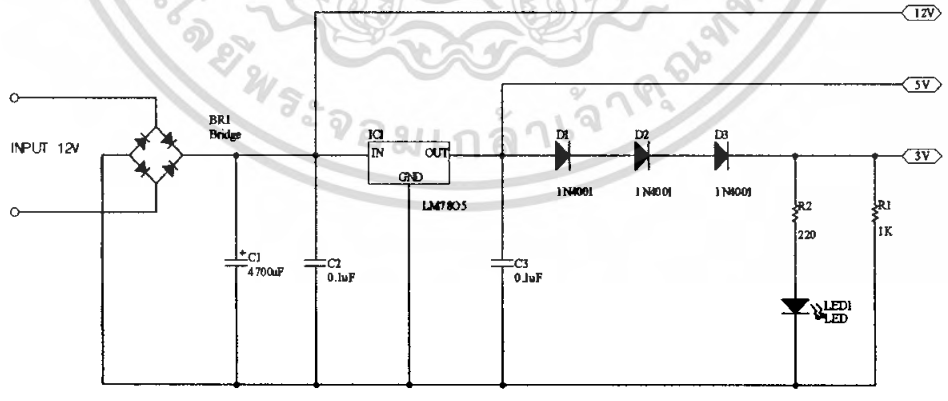
ตำแหน่ง address ที่ทำการบันทึก										คำพูดที่ทำการบันทึก
1	2	3	4	5	6	7	8	9	10	
0	0	0	0	0	1	0	0	0	0	หนึ่ง
0	0	0	0	0	0	1	0	0	0	สอง
0	0	0	0	0	1	1	0	0	0	สาม
0	0	0	0	0	0	0	1	0	0	สี่
0	0	0	0	0	1	0	1	0	0	ห้า
0	0	0	0	0	0	1	1	0	0	หก
0	0	0	0	0	1	1	1	0	0	เจ็ด
0	0	0	0	0	0	0	0	1	0	แปด
0	0	0	0	0	1	0	0	1	0	เก้า
0	0	0	0	0	0	1	0	1	0	ศูนย์
0	0	0	0	0	1	1	0	1	0	สาย
0	0	0	0	0	0	0	1	1	0	กำลังจะเข้าป้าย
0	0	0	0	0	1	0	1	1	0	ถ้าต้องการขึ้นรถ
0	0	0	0	0	0	1	1	1	0	กดปุ่ม 1
0	0	0	0	0	1	1	1	1	0	กดปุ่ม 2
0	0	0	0	1	1	0	0	0	0	เสียงเตือน

โดยในส่วนวงจรของรถประจำทางจะมีการเชื่อมต่ออุปกรณ์ต่างๆ กับไมโครคอนโทรลเลอร์ที่เหมือนกับ ในส่วนวงจรของป้ายรถประจำทางทุกประการแต่จะต่างกันเพียงในส่วนของ LED แสดงผล

ตารางที่ 3.14 ตารางแสดงคำพูดที่บันทึกเสียงในแต่ละแอดเดรส ของ IC บันทึกเสียงที่รถประจำทาง

ตำแหน่ง address ที่ทำการบันทึก										คำพูดที่ทำการบันทึก	รหัส
1	2	3	4	5	6	7	8	9	10		
0	0	0	0	0	0	0	0	1	0	เสียงเตือน	-
0	0	0	0	0	0	0	1	0	0	ป้ายต่อไป	-
0	0	0	0	0	0	0	1	1	0	พระจอมเกล้า	2001,3007
0	0	0	0	0	0	1	0	0	0	ตลาดหัวตะเข้	2002,3006
0	0	0	0	0	0	1	0	1	0	ซอยไปรษณีย์	2003,3005
0	0	0	0	0	0	1	1	0	0	แยกกิ่งแก้ว	2004,3004
0	0	0	0	0	0	1	1	1	0	หมู่บ้านพิบูลณ์สิน	2005,3003
0	0	0	0	0	1	0	0	1	0	คลองตัน	2006,3002
0	0	0	0	0	1	0	1	0	0	ซีคอน	2007,3001

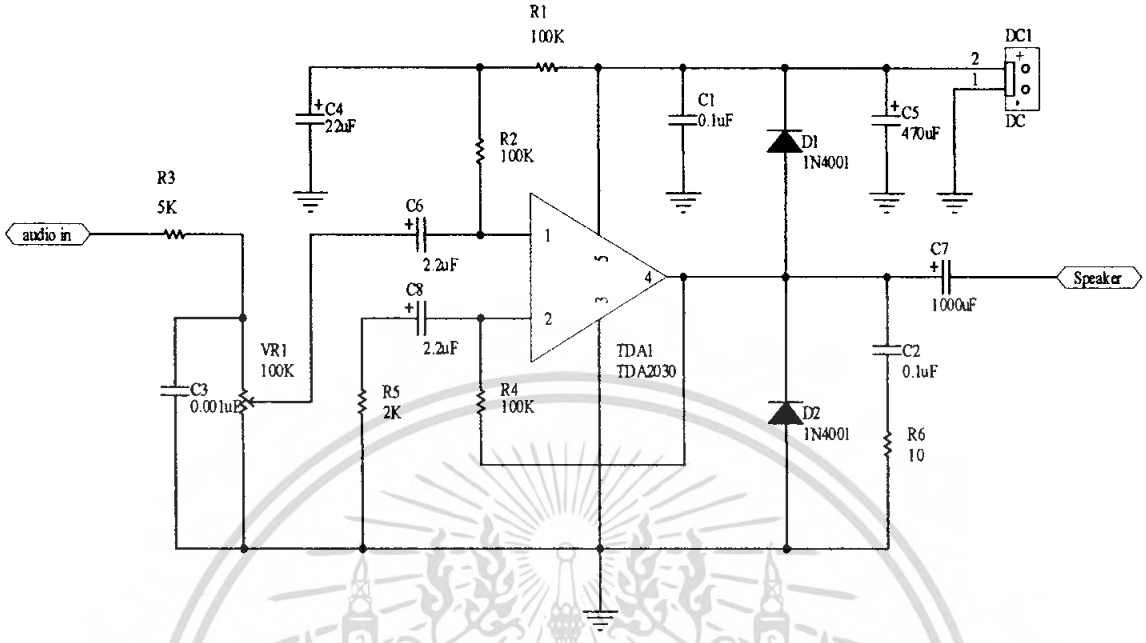
3.7 วงจรจ่ายไฟ



รูปที่ 3.7 วงจรจ่ายไฟ

จากรูปที่ 3.7 เป็นวงจรจ่ายไฟโดยในโครงงานนี้เราต้องการใช้ไฟประมาณ 3V , 5V และ 12V จ่ายให้แก่อุปกรณ์ต่างๆ

3.8 วงจรขยายเสียง

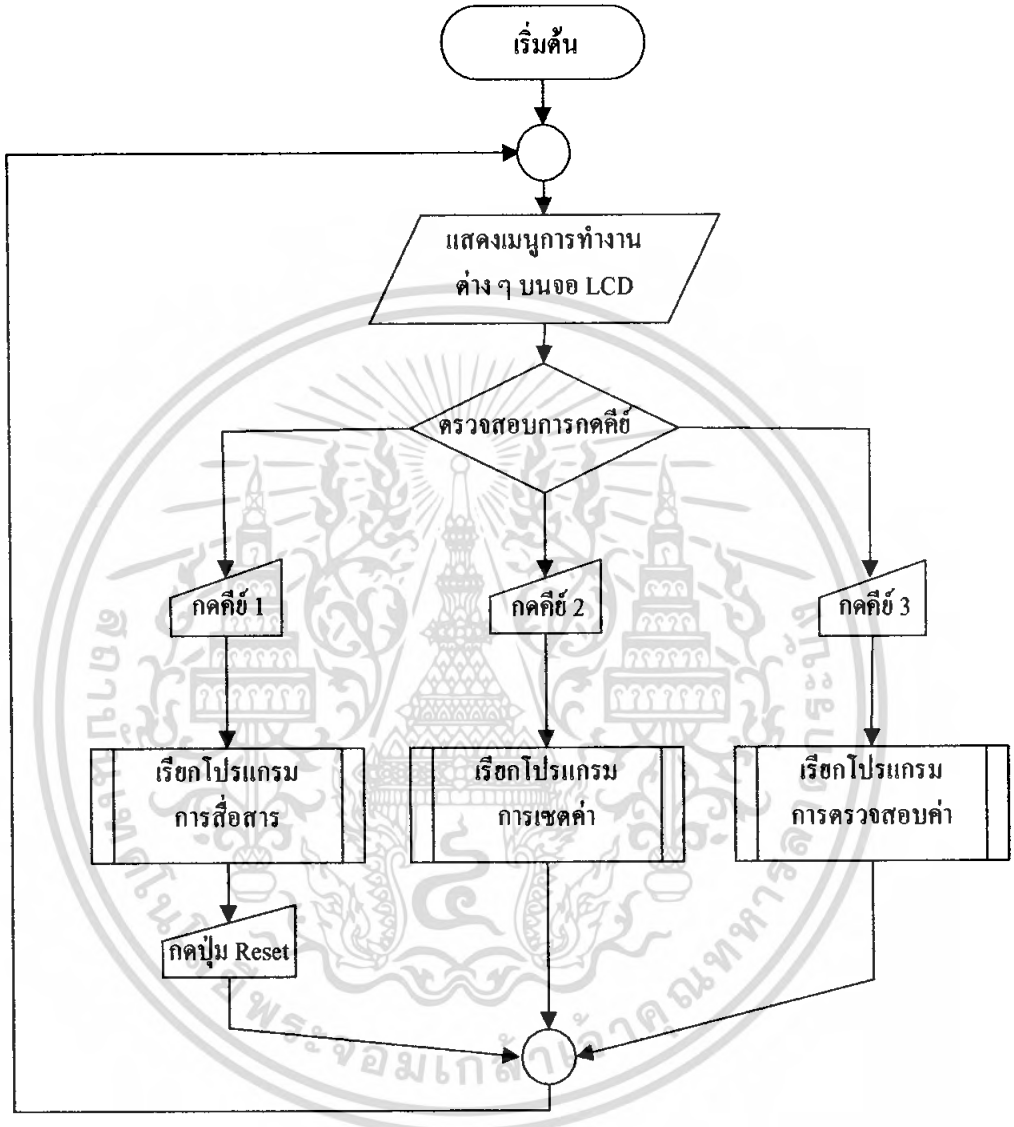


รูปที่ 3.8 วงจรขยายเสียง

จากรูปที่ 3.8 เป็นวงจรขยาย โดยในโครงงานนี้จะใช้ IC เบอร์ TDA2030 เป็นตัวขยายกำลังสัญญาณเสียง ซึ่งอินพุตของวงจร จะได้มาจาก เอาท์พุตของ IC บันทึกเสียง เบอร์ ISD25120

3.9 การออกแบบโปรแกรม

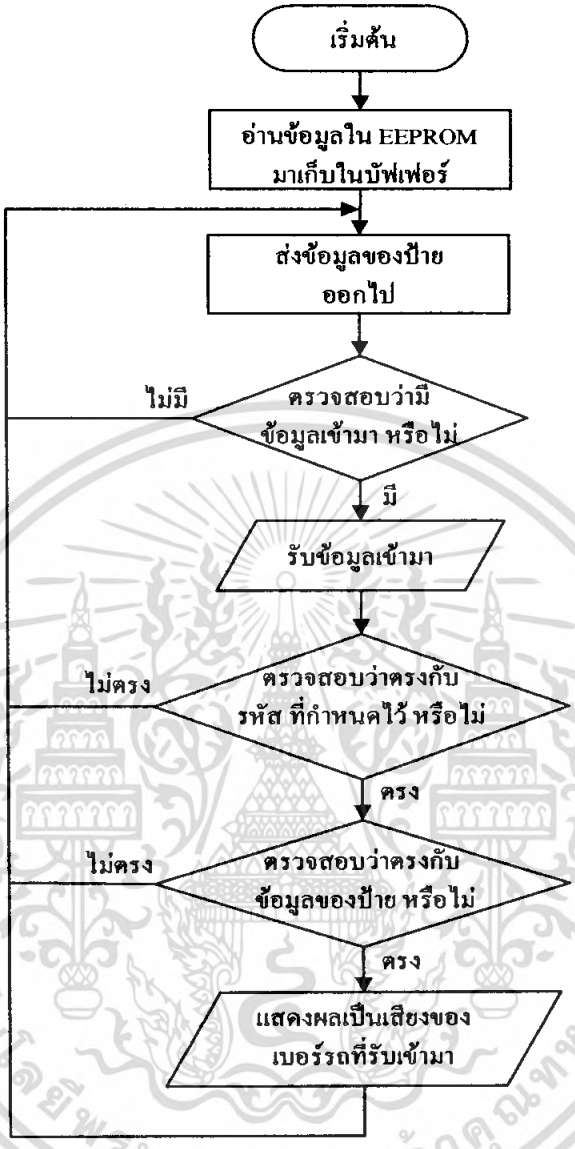
3.9.1 การทำงานในส่วนของเมนูหลักของป้ายรถประจำทาง



รูปที่ 3.9 Flowchart แสดงการทำงานในส่วนของเมนูหลักของป้ายรถประจำทาง

รูปที่ 3.9 เป็นการแสดงการทำงานในส่วนของเมนูหลักของป้ายรถประจำทาง โดยจะแบ่งการทำงานออกเป็น 3 ส่วน ซึ่งหากกดคีย์ 1 จะเป็นการเรียกโปรแกรมหลักในการสื่อสารระหว่างป้ายรถประจำทางกับรถประจำทาง ในรูปแบบที่ทำการเริ่มต้นการสื่อสารจากป้ายรถประจำทางก่อน หากกดคีย์ 2 จะเป็นการเรียกโปรแกรมย่อยในการตั้งค่าหมายเลขของป้ายรถประจำทาง และหากกดคีย์ 3 จะเป็นการเรียกโปรแกรมย่อยในการตรวจสอบค่าหมายเลขของป้ายรถประจำทางที่ได้ทำการบันทึกไว้

3.9.1.1 โปรแกรมย่อยในการสื่อสารของป้ายรถประจำทาง



รูปที่ 3.10 Flowchart แสดงโปรแกรมย่อยในการสื่อสารของป้ายรถประจำทาง

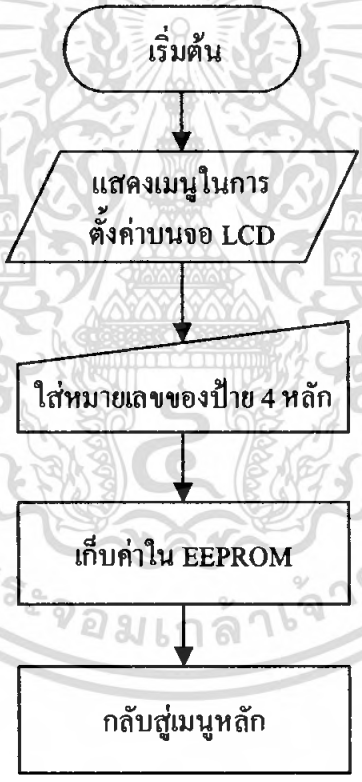
จากรูปที่ 3.10 แสดงการทำงานของโปรแกรมย่อยในการสื่อสารของป้ายรถประจำทาง โดยมีขั้นตอนการทำงานดังนี้

- เริ่มต้นจากการอ่านข้อมูลของหมายเลขป้ายที่ได้บันทึกไว้จาก EEPROM มาเก็บในบัฟเฟอร์ เซตค่าเริ่มต้นให้กับ RF Module (TRW-2.4GHz)

- ทำการส่งข้อมูลของป้ายรถประจำทางออกไป โดยจะส่งข้อมูลไปจำนวน 3 ไบต์ ซึ่งไบต์แรกจะเป็นรหัสของการส่ง (ส่งให้รถประจำทางอย่างเดียว) ไบต์ที่ 2 จะเป็นข้อมูลของหมายเลขป้ายรถประจำทางในหลักพัน กับหลักร้อย และไบต์ที่ 3 จะเป็นข้อมูลของหมายเลขป้ายรถประจำทาง ในหลักสิบ กับหลักหน่วย

- ตรวจสอบดูว่ามีข้อมูลเข้ามาหรือไม่ ถ้าไม่มีข้อมูลเข้ามาก็จะกลับไปเริ่มส่งข้อมูลของป้ายออกไปใหม่ แต่ถ้ามีข้อมูลเข้ามาก็จะรับข้อมูลเข้ามา 1 ไบต์ แล้วตรวจสอบดูว่าตรงกับรหัสที่ใช้ส่งให้ป้ายหรือไม่ (เพื่อป้องกันการรับข้อมูลจากป้ายรถตรงข้าม) ถ้าไม่ตรงก็จะกลับไปเริ่มส่งข้อมูลของป้ายออกไปใหม่ แต่ถ้าตรงก็จะรับข้อมูลเข้ามาอีก 2 ไบต์ แล้วตรวจสอบดูว่าตรงกับหมายเลขของป้ายตัวเองหรือไม่ (เพื่อป้องกันการรับข้อมูลจากรถที่วิ่งอยู่ในเส้นทางตรงข้าม) ถ้าไม่ตรงก็จะกลับไปเริ่มส่งข้อมูลของป้ายออกไปใหม่ แต่ถ้าตรงก็จะรับข้อมูลเข้ามาอีก 2 ไบต์ ซึ่งข้อมูลที่รับเข้ามานี้จะเป็นข้อมูลของหมายเลขรถประจำทาง จากนั้นก็จะนำหมายเลขรถประจำทางที่ได้ไปแสดงผลเป็นเสียงออกมาว่ามีรถสายที่รับข้อมูลเข้ามากำลังจะเข้าป้าย แล้วจึงกลับไปวนส่งข้อมูลของป้ายรถประจำทางออกไปใหม่

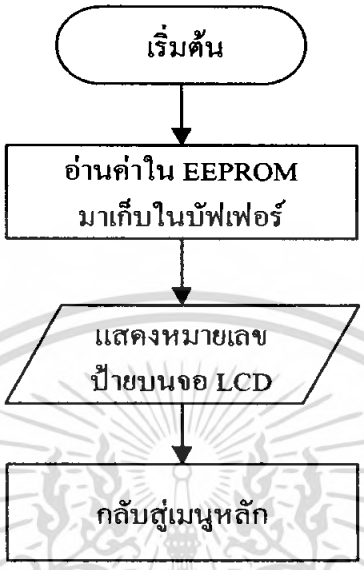
3.9.1.2 โปรแกรมย่อยในการเซตค่าหมายเลขให้กับป้ายรถประจำทาง



รูปที่ 3.11 Flowchart แสดงโปรแกรมย่อยในการเซตค่าหมายเลขให้กับป้ายรถประจำทาง

จากรูปที่ 3.11 แสดงการทำงานในการเซตค่าหมายเลขให้กับป้ายรถประจำทาง โดยเริ่มต้นจากการแสดงเมนูของการตั้งค้าบนหน้าจอ LCD จากนั้นให้ทำการป้อนค่าหมายเลขของป้ายรถประจำทางจำนวน 4 หลัก ซึ่งป้ายแต่ละป้ายในเส้นทางของรถจะต้องมีหมายเลขที่ไม่ซ้ำกัน จากนั้นทำการเก็บค่าลงใน EEPROM แล้วจึงกลับสู่เมนูหลัก

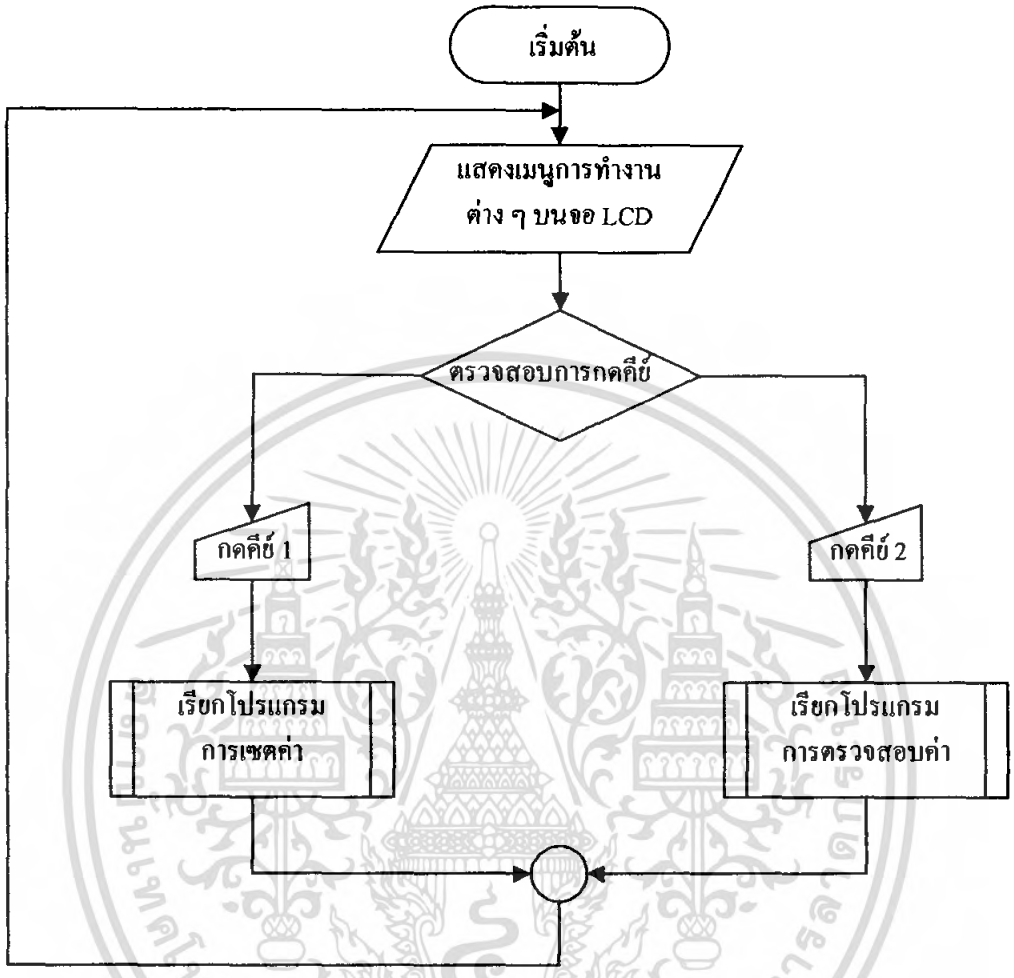
3.9.1.3 โปรแกรมย่อยในการตรวจสอบค่าหมายเลขป้ายรถประจำทางที่ได้บันทึกไว้



รูปที่ 3.12 Flowchart แสดงโปรแกรมย่อยในการตรวจสอบค่าหมายเลขป้ายรถประจำทาง

จากรูปที่ 3.12 แสดงการทำงานในการตรวจสอบค่าหมายเลขป้ายรถประจำทาง โดยเริ่มต้นจากการอ่านค่าใน EEPROM มาเก็บไว้ในบัฟเฟอร์ จากนั้นก็แสดงค่าหมายเลขของป้ายบนจอ LCD แล้วก็กลับไปสู่เมนูหลัก

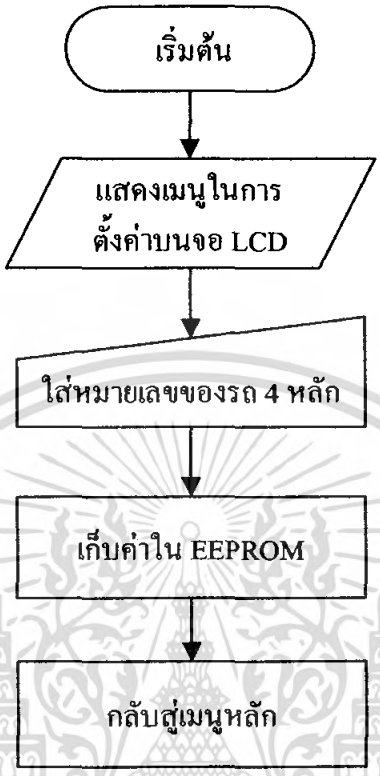
3.9.2 การทำงานในส่วนของเมนูหลักของรถประจำทาง



รูปที่ 3.13 Flowchart แสดงการทำงานในส่วนของเมนูหลักของรถประจำทาง

รูปที่ 3.13 เป็นการแสดงการทำงานในส่วนของเมนูหลักของรถประจำทาง โดยจะแบ่งการทำงานออกเป็น 2 ส่วน ซึ่งหากกดคีย์ 1 จะเป็นการเรียกโปรแกรมย่อยในการตั้งค่าหมายเลขรถประจำทาง และหากกดคีย์ 2 จะเป็นการเรียกโปรแกรมย่อยในการตรวจสอบค่าหมายเลขของรถประจำทางที่ได้ทำการบันทึกไว้

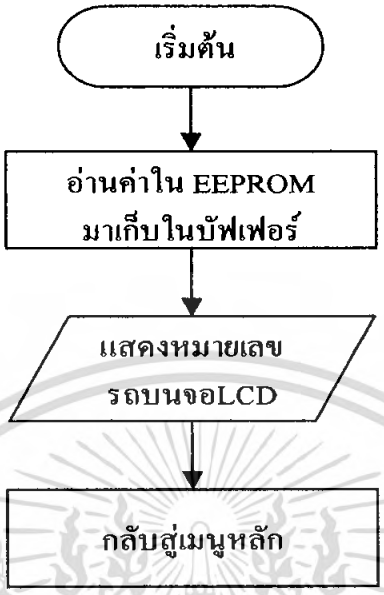
3.9.2.1 โปรแกรมย่อยในการเซตค่าหมายเลขให้กับรถประจำทาง



รูปที่ 3.14 Flowchart แสดงโปรแกรมย่อยในการเซตค่าหมายเลขให้กับรถประจำทาง

จากรูปที่ 3.14 แสดงการทำงานในการเซตค่าหมายเลขให้กับรถประจำทาง โดยเริ่มจากการแสดงเมนูในการตั้งค่าบนจอ LCD จากนั้นให้ทำการป้อนค่าหมายเลขของรถประจำทางจำนวน 4 หลัก เมื่อครบ 4 หลัก ก็ทำการเก็บค่าลงใน EEPROM แล้วจึงกลับสู่เมนูหลัก

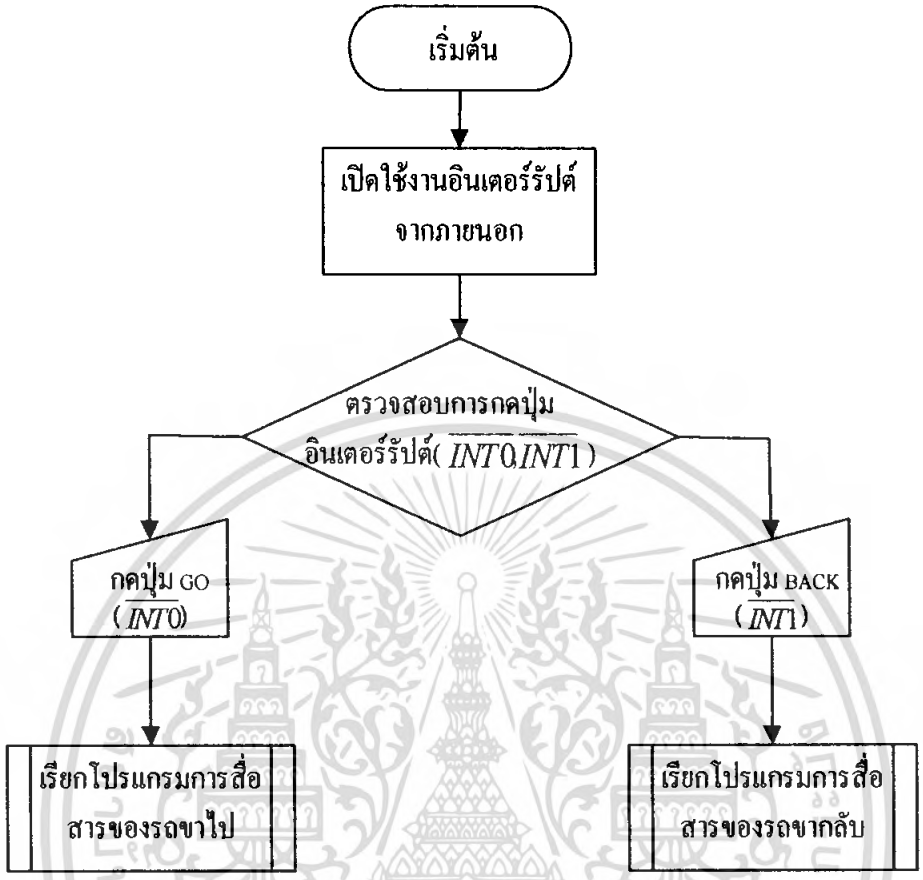
3.9.2.2 โปรแกรมย่อยในการตรวจสอบค่าหมายเลขรถประจำทางที่ได้บันทึกไว้



รูปที่ 3.15 Flowchart แสดงโปรแกรมย่อยในการตรวจสอบค่าหมายเลขรถประจำทาง

จากรูปที่ 3.15 แสดงการทำงานในการตรวจสอบค่าหมายเลขของรถประจำทาง โดยเริ่มต้นจากการอ่านค่าใน EEPROM มาเก็บไว้ในบัฟเฟอร์ จากนั้นก็แสดงค่าหมายเลขของรถประจำทางบนจอ LCD แล้วก็กลับไปสู่เมนูหลัก

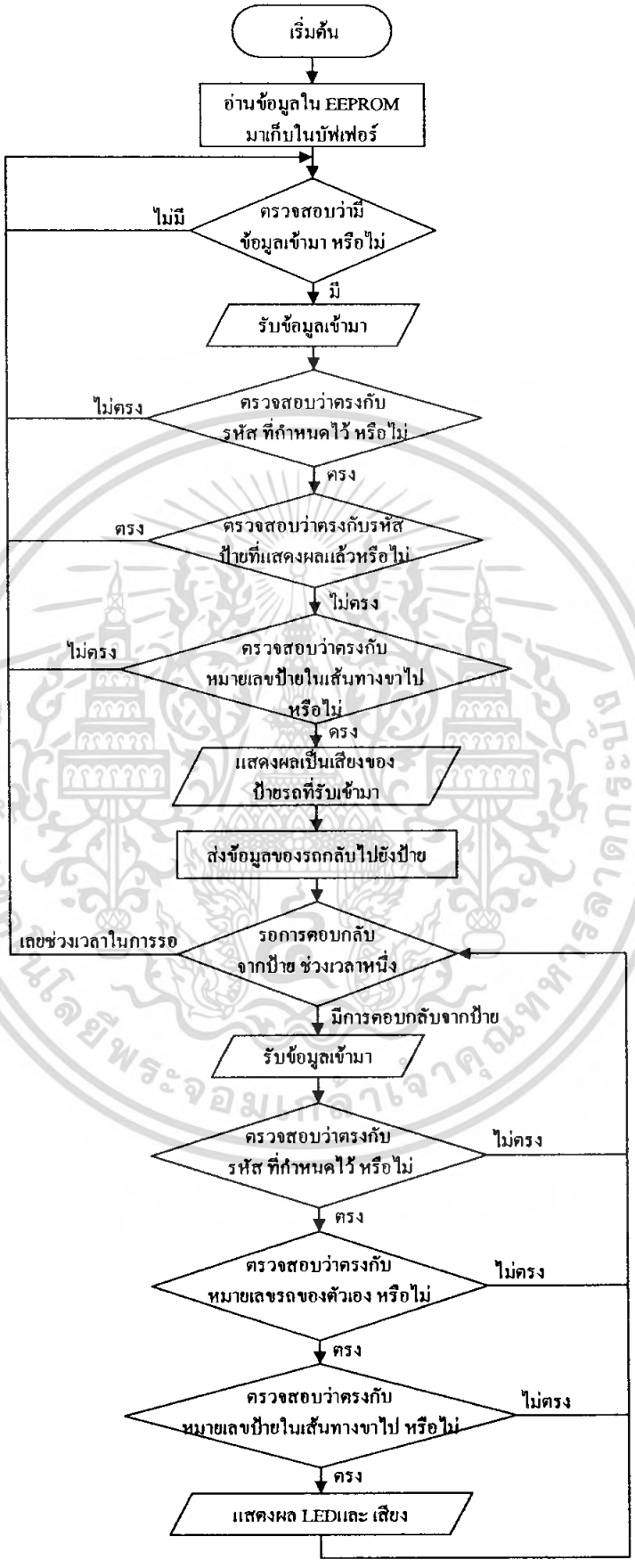
3.9.3 โปรแกรมการสื่อสารของรถประจำทาง



รูปที่ 3.16 Flowchart แสดงโปรแกรมการสื่อสารของรถประจำทาง

จากรูปที่ 3.16 แสดงการทำงานของโปรแกรมการสื่อสารของรถประจำทาง โดยเริ่มต้นจากการเปิดใช้งานบริการอินเตอร์รัปต์จากภายนอก แล้วตรวจสอบการกดปุ่มอินเตอร์รัปต์ ($\overline{INT0}$,$\overline{INT1}$) ซึ่งหากมีการกดปุ่มอินเตอร์รัปต์ $\overline{INT0}$ จะเป็นการเรียกโปรแกรมหลักในการสื่อสารของรถประจำทางเส้นทางขาไป และหากมีการกดปุ่มอินเตอร์รัปต์ $\overline{INT1}$ จะเป็นการเรียกโปรแกรมหลักในการสื่อสารของรถประจำทางเส้นทางขากลับ

3.9.3.1 โปรแกรมย่อยการสื่อสารของรถประจำทางเส้นทางขาไป



รูปที่ 3.17 Flowchart แสดงโปรแกรมย่อยการสื่อสารของรถประจำทางเส้นทางขาไป

จากรูปที่ 3.17 แสดงการทำงานของโปรแกรมย่อยการสื่อสารของรถประจำทางเส้นทางขาไป โดยมีขั้นตอนการทำงานดังนี้

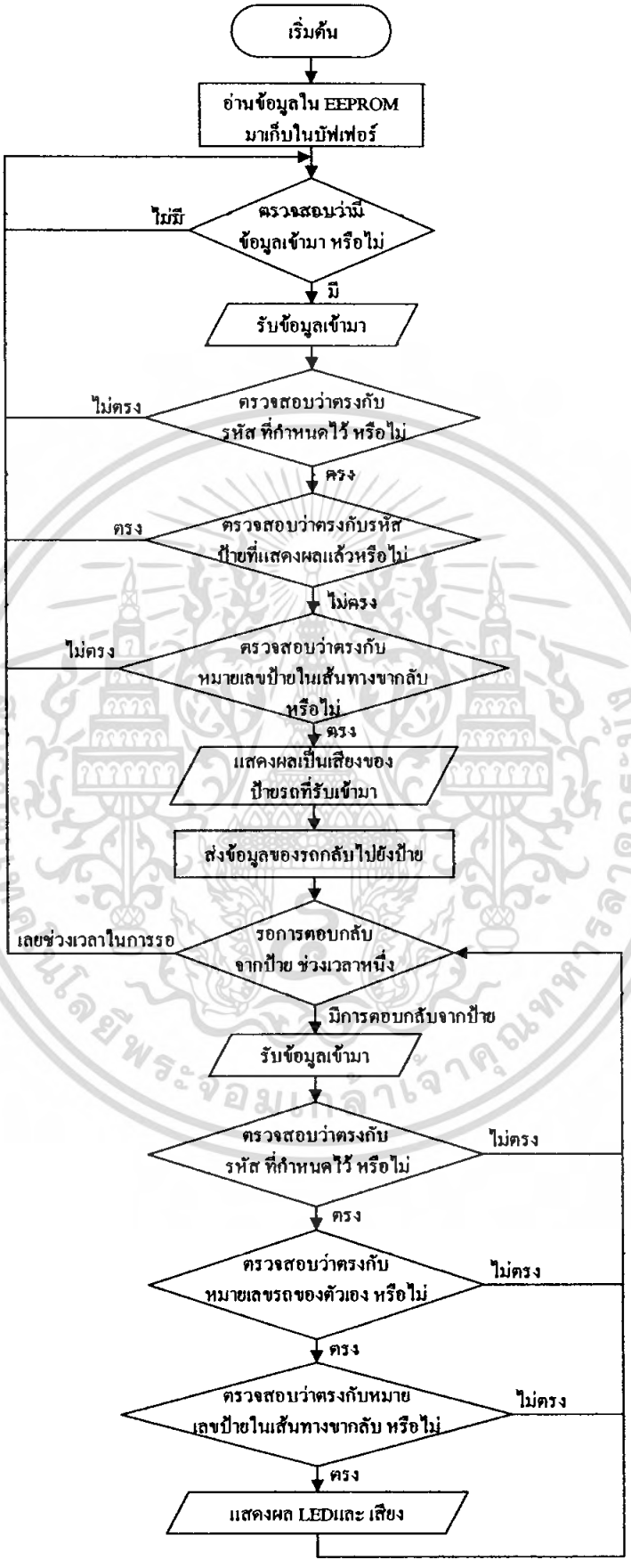
- เริ่มต้นจากการอ่านข้อมูลของหมายเลขรถประจำทางที่ได้บันทึกไว้จาก EEPROM มาเก็บในบัฟเฟอร์ เซตค่าเริ่มต้นให้กับ RF Module (TRW-2.4GHz)

- คอยตรวจสอบว่ามีข้อมูลเข้ามาหรือไม่ ถ้ามีข้อมูลเข้ามาก็จะรับข้อมูลเข้ามา 1 ไบต์ แล้วตรวจสอบดูว่าตรงกับรหัสที่ใช้ส่งให้รถประจำทางหรือไม่ (เพื่อป้องกันการรับข้อมูลจากรถประจำทางที่วิ่งในเส้นทางเดียวกัน) ถ้าไม่ตรงก็จะกลับไปรอรับข้อมูลใหม่ แต่ถ้าตรงก็จะรับข้อมูลเข้ามาอีก 2 ไบต์ แล้วตรวจสอบดูว่าตรงกับหมายเลขของป้ายรถประจำทาง ที่เคยแสดงผลไปแล้วหรือไม่ (เพื่อป้องกันการรับข้อมูลจากป้ายที่เคยแสดงผลไปแล้วเข้ามาอีก) ถ้าตรงก็จะกลับไปรอรับข้อมูลใหม่ แต่ถ้าไม่ตรงก็จะนำข้อมูลนั้นไปตรวจสอบดูว่า ตรงกับหมายเลขของป้ายรถประจำทางในเส้นทางขาไปที่ได้กำหนดไว้ในโปรแกรมหรือไม่ (เพื่อป้องกันการรับข้อมูลจากป้ายรถประจำทางที่อยู่ในเส้นทางขากลับ) ถ้าไม่ตรงก็จะกลับไปรอรับข้อมูลใหม่อีกครั้ง แต่ถ้าตรงก็จะรับข้อมูลเข้ามาอีก 2 ไบต์ ซึ่งข้อมูลที่รับเข้ามานี้จะเป็นข้อมูลของหมายเลขป้ายรถประจำทาง จากนั้นก็จะนำหมายเลขป้ายรถประจำทางที่ได้ไปแสดงผลเป็นเสียงของชื่อป้ายตามรหัสหมายเลขที่ได้บันทึกไว้ใน IC บันทึกเสียง

- เมื่อแสดงผลเสร็จแล้ว รถประจำทางก็จะส่งข้อมูลของรถกลับไปยังป้าย โดยจะส่งข้อมูลกลับไปจำนวน 5 ไบต์ ซึ่งไบต์แรกจะเป็นรหัสของการส่ง (ส่งให้ป้ายรถประจำทางอย่างเดียว) และไบต์ที่ 2 กับ 3 จะเป็นข้อมูลของหมายเลขป้ายที่รับเข้ามา และไบต์ที่ 4 จะเป็นข้อมูลของหมายเลขรถประจำทางในหลักพัน กับหลักร้อย และไบต์ที่ 5 จะเป็นข้อมูลของหมายเลขป้ายรถประจำทางในหลักสิบ กับหลักหน่วย

- รอการตอบกลับจากป้ายรถประจำทางในช่วงเวลาหนึ่ง (รอการเรียกรถเมื่อมีคนต้องการจะขึ้นรถ) ซึ่งในช่วงเวลานั้นรถจะรับแต่ข้อมูลที่เป็นรหัสที่ใช้เรียกรถเท่านั้นจนกว่าจะเลยช่วงเวลาที่กำหนด เมื่อมีข้อมูลเข้ามาแล้วข้อมูลนั้นตรงกับรหัสที่ใช้เรียกรถ ก็จะรับข้อมูลเข้ามาอีก 2 ไบต์ แล้วตรวจสอบว่าตรงกับหมายเลขของรถตัวเองหรือไม่ (เพื่อป้องกันการรับข้อมูลการเรียกรถจากป้ายในเส้นทางขากลับ) ถ้าไม่ตรงก็จะกลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถใหม่ แต่ถ้าตรงก็จะรับข้อมูลเข้ามาอีก 2 ไบต์ แล้วตรวจสอบว่าข้อมูลที่รับเข้ามานั้นตรงกับหมายเลขของป้ายในเส้นทางขาไปหรือไม่ (เพื่อป้องกันการรับข้อมูลการเรียกรถเบอร์เดียวกันจากป้ายในเส้นทางขากลับ) ถ้าไม่ตรงก็จะกลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถใหม่ แต่ถ้าตรงก็จะแสดงผลเป็นเสียงเพื่อบอกให้คนขับรถรู้ว่ามีคนต้องการจะขึ้นรถที่ป้ายข้างหน้า แล้วก็กลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้เรียกรถจนกว่าจะเลยเวลาที่กำหนด เมื่อเลยเวลาที่กำหนดก็จะกลับไปรอรับข้อมูลที่เป็นรหัสที่ใช้ตรวจสอบรถใหม่ แล้วก็วนทำซ้ำแบบนี้ไปเรื่อยๆ

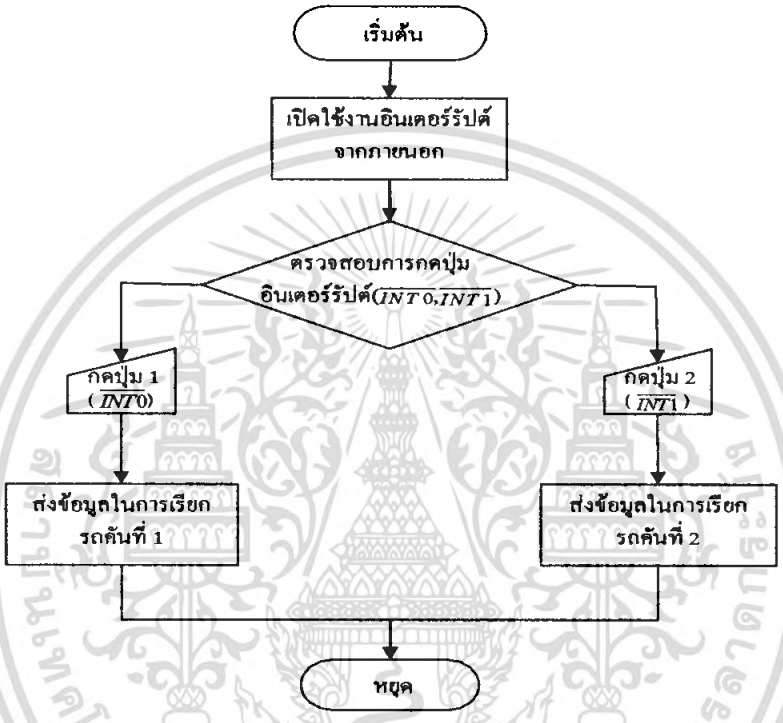
3.9.3.2 โปรแกรมย่อยการสื่อสารของรถประจำทางเส้นทางขากลับ



รูปที่ 3.18 Flowchart แสดงโปรแกรมย่อยการสื่อสารของรถประจำทางเส้นทางขากลับ

จากรูปที่ 3.18 แสดงการทำงานของโปรแกรมย่อยการสื่อสารของรถประจำทางเส้นทางขากลับ ซึ่งจะมีหลักการทำงานคล้ายกับหลักการทำงานของโปรแกรมหลักการสื่อสารของรถประจำทางเส้นทางขาไป แต่จะแตกต่างกันตรงที่หมายเลขป้ายในเส้นทางที่รถวิ่งผ่านที่ได้กำหนดไว้ในโปรแกรม และหมายเลขป้ายที่ใช้ในการตรวจสอบ จะเป็นหมายเลขป้ายในเส้นทางขากลับ

3.9.4 โปรแกรมย่อยการสื่อสารเพื่อเรียกรถของป้ายรถประจำทาง



รูปที่ 3.19 Flowchart โปรแกรมย่อยการสื่อสารเพื่อเรียกรถของป้ายรถประจำทาง

จากรูปที่ 3.19 แสดงการทำงานของโปรแกรมย่อยการสื่อสารเพื่อเรียกรถของป้ายรถประจำทาง ซึ่งมีขั้นตอนการทำงาน โดยเริ่มต้นจากการเปิดใช้งานบริการอินเตอร์รัปต์จากภายนอก แล้วตรวจสอบการกดปุ่มอินเตอร์รัปต์ ($\overline{INT0}$,$\overline{INT1}$)

- มีการกดปุ่มอินเตอร์รัปต์ $\overline{INT0}$ (กดปุ่ม 1) ระบบก็จะทำการส่งข้อมูลในการเรียกรถคันที่หนึ่งออกไป โดยจะส่งข้อมูลไปจำนวน 5 ไบต์ ซึ่งไบต์แรกจะเป็นรหัสของการส่ง (ส่งให้รถเพื่อเรียกรถ) และไบต์ที่ 2 กับ 3 จะเป็นข้อมูลของหมายเลขรถประจำทางของคันที่ 1 และไบต์ที่ 4 กับ 5 จะเป็นข้อมูลของหมายเลขป้าย

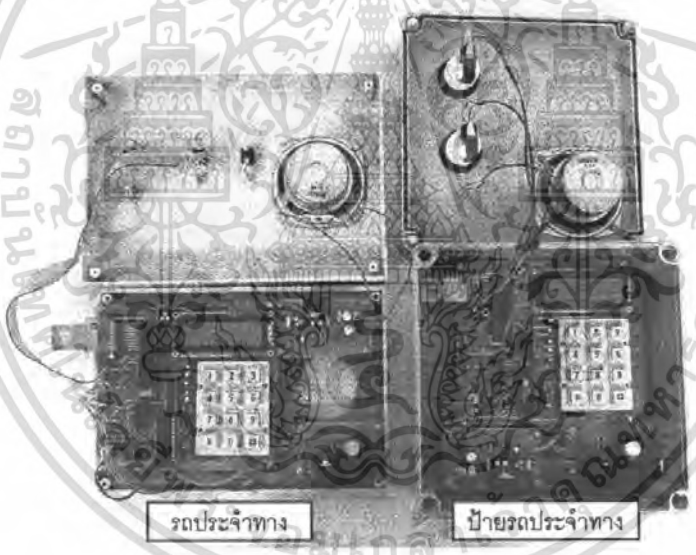
- มีการกดปุ่มอินเตอร์รัปต์ $\overline{INT1}$ (กดปุ่ม 2) ระบบก็จะทำการส่งข้อมูลในการเรียกรถคันที่สองออกไป โดยจะส่งข้อมูลไปจำนวน 5 ไบต์ ซึ่งไบต์แรกจะเป็นรหัสของการส่ง (ส่งให้รถเพื่อเรียกรถ) และไบต์ที่ 2 กับ 3 จะเป็นข้อมูลของหมายเลขรถประจำทางของคันที่ 2 และไบต์ที่ 4 กับ 5 จะเป็นข้อมูลของหมายเลขป้าย ซึ่งข้อมูลของหมายเลขรถประจำทางของคันที่ 1 กับ 2 นั้นจะเปลี่ยนไปเรื่อยๆ โดยจะเปลี่ยนไปตามหมายเลขรถที่เข้ามา

บทที่ 4
การทดลองและผลการทดลอง

ในการสื่อสารระหว่างป้ายรถประจำทางกับรถประจำทางจำเป็นต้องใช้อุปกรณ์ในการสื่อสาร โดยในการสื่อสารในโครงงานนี้จะใช้โมดูล TRW 2.4 GHz เพื่อให้เกิดความเข้าใจในตัวโมดูลก่อนที่จะนำไปใช้งาน เราก็จะทำการทดลอง และทดสอบการทำงาน ของตัวโมดูลก่อน โดยในการทดลองจะให้โมดูล ตัวที่ 1 ทำการส่งข้อมูลอย่างเดียว และโมดูล ตัวที่ 2 รับข้อมูลอย่างเดียว

4.1 การทดลองโปรแกรม รับ/ส่ง ข้อมูลโดยใช้โมดูล TRW 2.4GHz

ในการทดลองนี้ มีวัตถุประสงค์เพื่อทดสอบการทำงานของโปรแกรมที่ใช้ในการ รับ/ส่ง ข้อมูลระหว่างภาครับ และภาคส่งของป้ายรถประจำทาง และรถประจำทางว่าสามารถ รับ/ส่งข้อมูลระหว่างกันได้หรือไม่

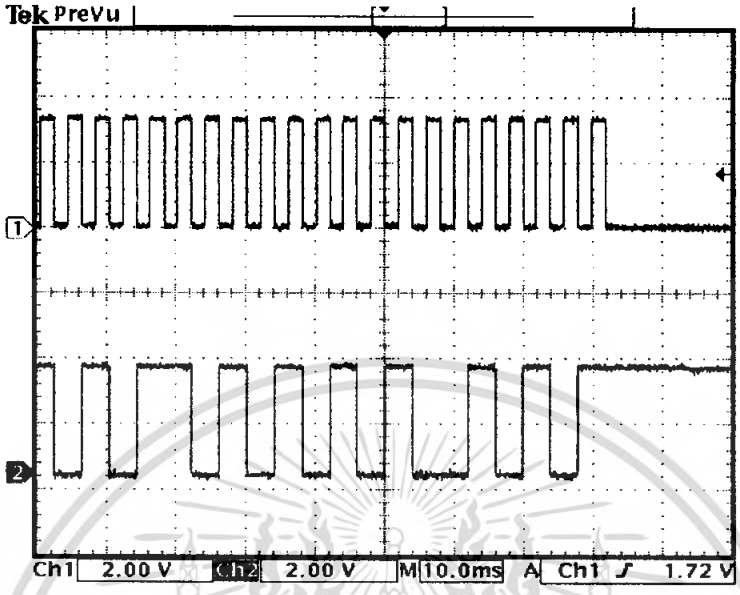


รูปที่ 4.1 แสดงอุปกรณ์สำหรับการทดลองโปรแกรม รับ/ส่ง ข้อมูลระหว่างป้าย และรถประจำทาง

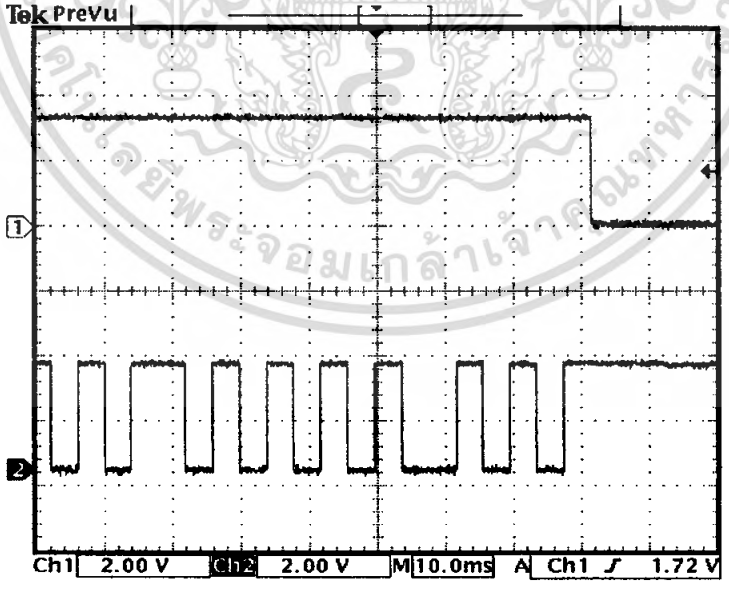
4.1.1 การทดลองในส่วนของภาคส่ง

วิธีการทดลอง โดยการทดสอบการ รับ/ส่ง ข้อมูลของภาคส่ง และภาครับสัญญาณ โดยข้อมูลที่ส่งก็คือ 95H ส่งไปแสดงผลที่ภาครับสัญญาณ ในส่วนของโหมดความถี่วิทยุทางด้านภาคส่ง เมื่อต้องการที่จะส่งข้อมูลต้องทำการป้อนข้อมูลการตั้งค่าให้กับโมดูล TRW 2.4GHz ก่อนโดยต้องตั้งค่าให้ขา CE มีสถานะ " high " เพื่อโหลดข้อมูลไปเก็บไว้ภายในตัวโมดูลความถี่วิทยุ และตั้งค่าให้ CS มีสถานะ " low " หลังจากที่ส่งข้อมูลให้กับโมดูลความถี่วิทยุเรียบร้อยแล้วให้ตั้งค่า CE ให้อยู่ในสถานะ "low" เพื่อกระตุ้นให้โมดูลความถี่วิทยุทำการส่งข้อมูลออกไป

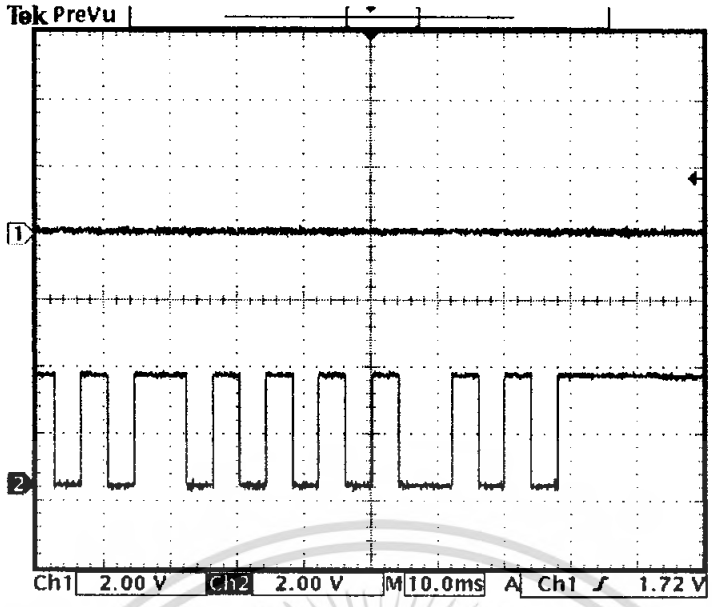
ผลการทดลอง หลังจากที่ได้ทดลองตามขั้นตอนข้างต้นแล้ว โดยทำการวัดสัญญาณที่ขา CE, CS, DATA, CLK1 ของโมดูลความถี่วิทยุ ซึ่งได้ผลการทดลองดังรูปที่แสดงดังต่อไปนี้



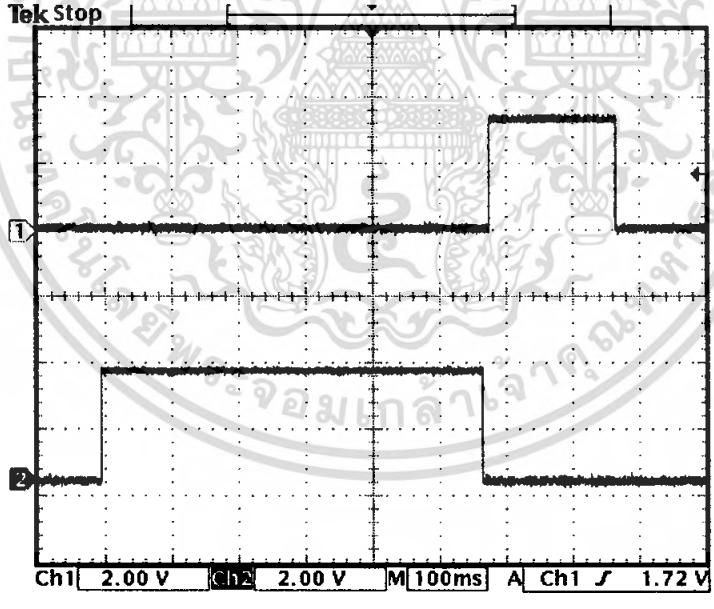
รูปที่ 4.2 แสดงความสัมพันธ์ระหว่างสัญญาณนาฬิกากับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณนาฬิกาขณะทำการวัดสัญญาณที่ขา CLK1
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA



รูปที่ 4.3 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา CE กับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CE
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA



รูปที่ 4.4 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา CS กับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CS
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA

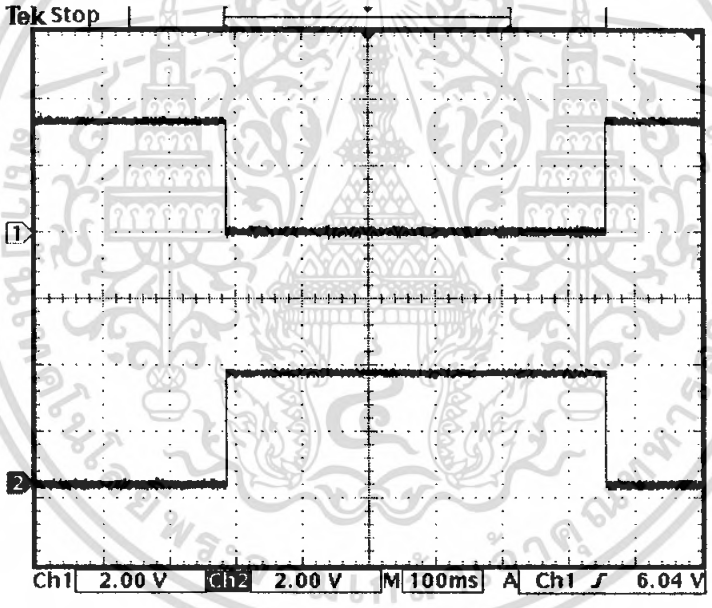


รูปที่ 4.5 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา CE กับสัญญาณที่ขา CS
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CE
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CS

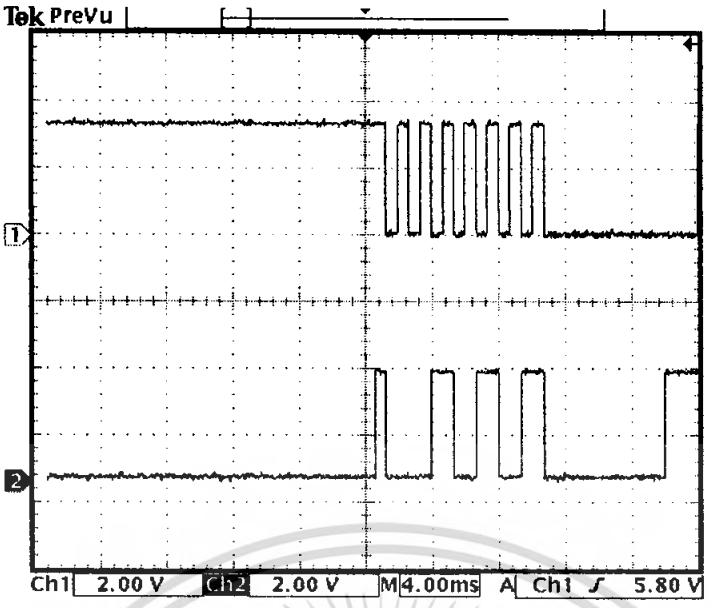
4.1.2 การทดลองในส่วนของภาครับ

วิธีการทดลอง สำหรับในกรณีของภาครับ เมื่อต้องการที่จะรับข้อมูลต้องทำการป้อนข้อมูลการตั้งค่าให้กับโมดูล TRW 2.4GHz โดยที่ตั้งค่าให้ค่า CE มีสถานะ "low" ขา CS มีสถานะ "high" และขา DR1 มีสถานะ "low" เพื่อแสดงความพร้อมที่จะรับข้อมูล ซึ่งขณะทำการรับข้อมูล DR1 จะอยู่ในสถานะ "high " เมื่อรับข้อมูลเรียบร้อยแล้ว DR1 จะกลับมาอยู่ในสถานะ "low" อีกครั้งเพื่อเตรียมพร้อมที่จะรับข้อมูลที่เข้ามาใหม่

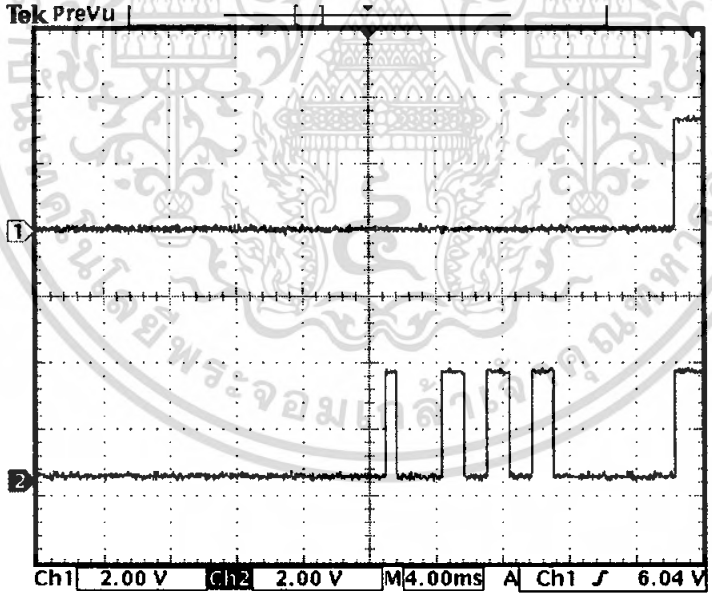
ผลการทดลอง หลังจากที่ได้ทดลองตามขั้นตอนข้างต้นแล้ว โดยทำการวัดสัญญาณที่ขา CE, CS, DR1, DATA ของโมดูล TRW 2.4GHz ดังรูป และทำการสังเกต ปรากฏว่าข้อมูลที่ได้เป็นสัญญาณข้อมูลที่ภาคส่งได้ทำการส่งมายังภาครับ ซึ่งค่าที่ได้ส่งมาคือ 95H แสดงว่าโปรแกรมในการ รับ/ส่ง ข้อมูลนี้สามารถทำงานได้ตามวัตถุประสงค์ของการทดลอง



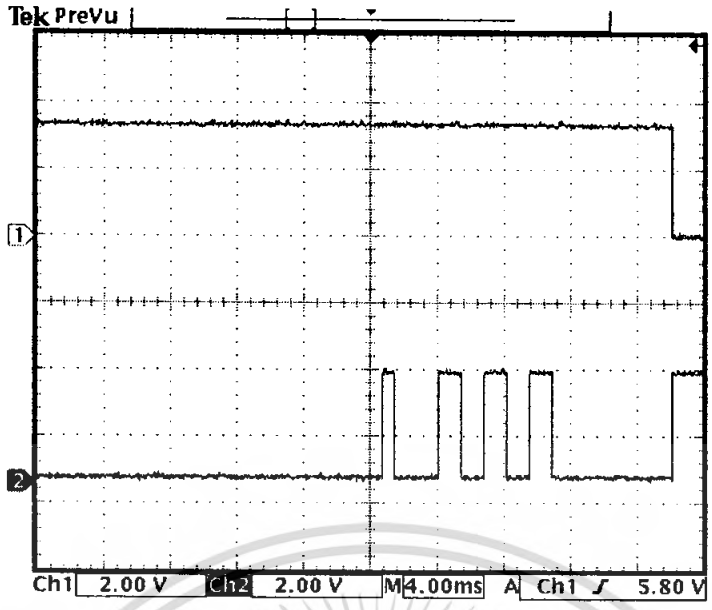
รูปที่ 4.6 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา CE กับสัญญาณที่ขา CS
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CE
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CS



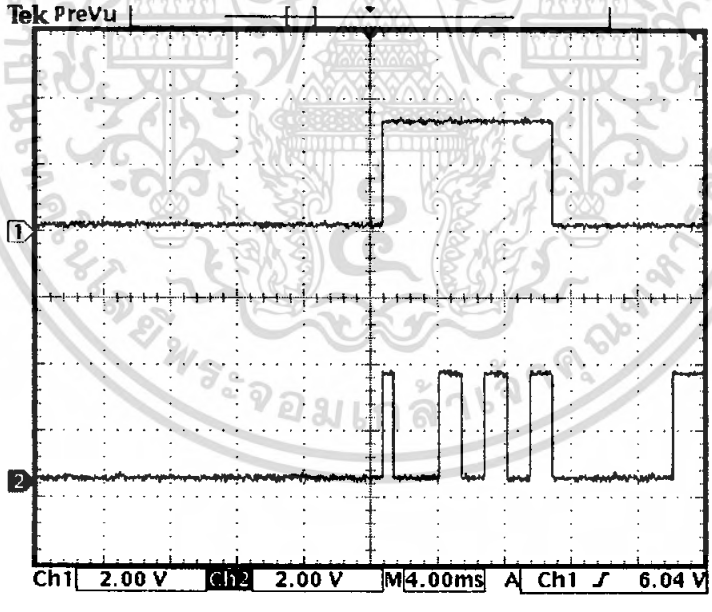
รูปที่ 4.7 แสดงความสัมพันธ์ระหว่างสัญญาณนาฬิกา กับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณนาฬิกาขณะทำการวัดสัญญาณที่ขา CLK1
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA



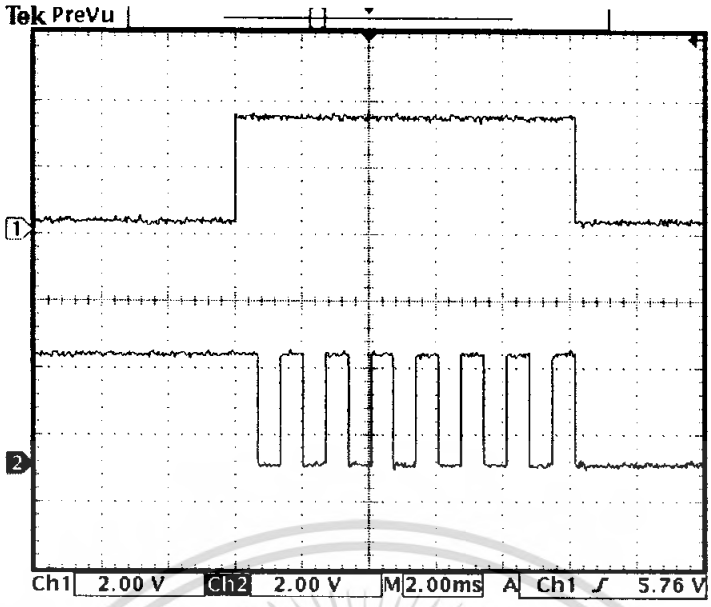
รูปที่ 4.8 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา CE กับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CE
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA



รูปที่ 4.9 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา CS กับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา CS
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA



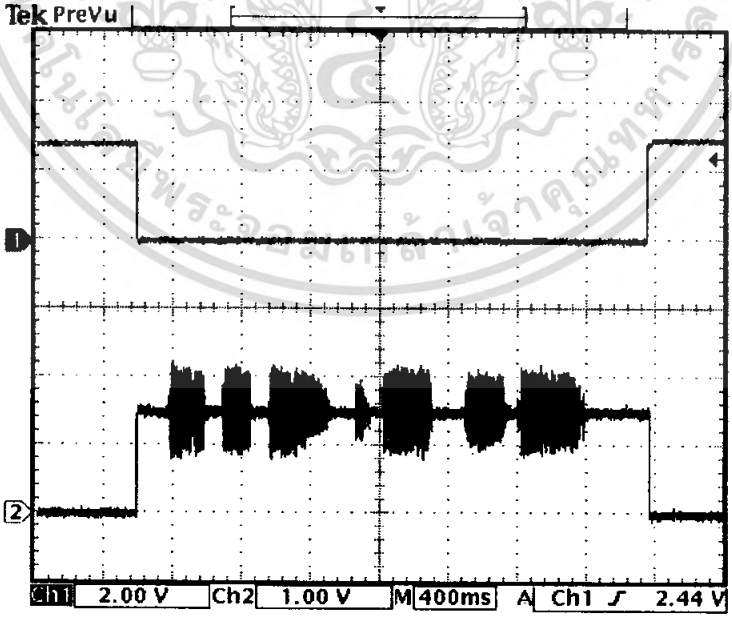
รูปที่ 4.10 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา DR1 กับสัญญาณที่ขา DATA
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DR1
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DATA



รูปที่ 4.11 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา DR1 กับสัญญาณที่ขา CLK1
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา DR1
(Channel 2) แสดงลักษณะของสัญญาณนาฬิกาขณะทำการวัดสัญญาณที่ขา CLK1

4.2 การทดลองวัดสัญญาณวงจรบันทึกเสียง

จากการทดลอง เมื่อป้อน address ให้กับ ขา A3-A10 แล้ว ต้องเซต ขา PD ให้เป็นค่า 0 จะทำให้ IC บันทึกเสียงแสดงผลออกมา



รูปที่ 4.12 แสดงความสัมพันธ์ระหว่างสัญญาณที่ขา PD กับสัญญาณเสียง SP+/-
(Channel 1) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา PD
(Channel 2) แสดงลักษณะของสัญญาณขณะทำการวัดสัญญาณที่ขา SP+/-

4.3 การทดลองวัดระยะทางที่สามารถติดต่อระหว่างป้ายกับรถ

จากการทดลองวัดระยะทางที่รถกับป้ายสามารถติดต่อกันได้พบว่าในสถานที่โล่ง ไม่มีสิ่งกีดขวางสามารถทำการติดต่อได้ไกลประมาณ 80 เมตร แต่ในการทำงานจริงในท้องถนนนั้นจะมีสิ่งกีดขวางที่เป็นอุปสรรคในการส่งข้อมูลจึงทำให้การใช้งานจริงได้ระยะไม่ถึง 80 เมตร

4.4 ขั้นตอนการทำงานในส่วนของป้ายรถประจำทาง

เมื่อเริ่มต้นทำการเปิดเครื่องที่หน้าจอ LCD จะแสดงข้อความของเมนูต่าง ๆ ที่สามารถเลือกได้ดังรูปที่ 4.13

รูปที่ 4.13 หน้าจอแสดงเมนูของการทำงานต่าง ๆ ที่สามารถเลือกได้

จากนั้นเมื่อแสดงข้อความต่าง ๆ ครบแล้วก็จะรอการกดคีย์ว่าต้องการเข้าสู่เมนูใด ซึ่งที่หน้าจอ LCD ก็จะแสดงข้อความดังรูปที่ 4. 14

รูปที่ 4.14 หน้าจอแสดงข้อความเมื่อรอการกดคีย์

หากต้องการทำการเซตค่าหมายเลขให้กับป้ายรถประจำทางก็ให้ กดคีย์ 2 ที่หน้าจอ LCD ก็จะแสดงข้อความเพื่อให้ป้อนหมายเลขของป้ายรถประจำทาง ดังรูปที่ 4.15 จากนั้นให้ทำการป้อนค่าหมายเลขของป้ายรถประจำทางจำนวน 4 หลัก

เมื่อป้อนหมายเลขครบ 4 หลักแล้วก็จะทำการบันทึกหมายเลขของป้ายรถประจำทางไว้ แล้วก็จะออกจากเมนูนี้โดยกลับไปยังเมนูหลักอีกครั้ง ซึ่งที่หน้าจอ LCD ก็จะแสดงข้อความเหมือนตอนเริ่มต้นใหม่อีกครั้ง

รูปที่ 4.15 หน้าจอแสดงข้อความให้บันทึกหมายเลขของป้าย

หากต้องการตรวจสอบดูว่าข้อมูลที่ได้บันทึกไว้นั้นเป็นหมายเลขอะไรบ้างก็ให้ กดคีย์ 3 ที่หน้าจอ LCD ก็จะแสดงหมายเลขของป้ายรถประจำทางที่ได้บันทึกไว้ดัง รูปที่ 4.16 แล้วก็จะออกจากเมนูนี้โดยกลับไปยังเมนูหลักอีกครั้ง ซึ่งที่หน้าจอ LCD ก็จะแสดงข้อความเหมือนตอนเริ่มต้นใหม่อีกครั้ง

รูปที่ 4.16 หน้าจอแสดงข้อมูลหมายเลขของป้ายที่ได้บันทึกไว้

หากต้องการเข้าสู่โปรแกรมหลักการสื่อสารของป้ายรถประจำทางก็ให้ กดคีย์ 1 ที่หน้าจอ LCD ก็จะแสดงข้อความดังรูปที่ 4.17 อุปกรณ์ก็จะเริ่มทำการรับส่งข้อมูล โดยจะเริ่มทำการวนส่งสัญญาณข้อมูลของป้ายรถประจำทางออกไป แล้วก็วนรอรับสัญญาณข้อมูลตอบกลับจากรถประจำทาง เมื่อมีข้อมูลตอบกลับจากรถประจำทาง ก็จะนำข้อมูลนั้นไปตรวจสอบถ้าตรงตามที่กำหนดไว้ ก็จะแสดงผลเป็นข้อความเสียงว่าจะมีรถสายอะไรที่กำลังจะเข้าป้าย

รูปที่ 4.17 หน้าจอแสดงข้อความเมื่อเข้าสู่โปรแกรมหลักการสื่อสารของป้ายรถประจำทาง

4.5 ขั้นตอนการทำงานในส่วนของรถประจำทาง

เมื่อเริ่มต้นทำการเปิดเครื่องที่หน้าจอ LCD จะแสดงข้อความของเมนูต่าง ๆ ที่สามารถเลือกได้ดังรูปที่ 4.18

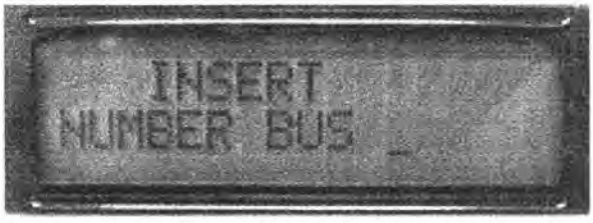
รูปที่ 4.18 หน้าจอแสดงเมนูของการทำงานต่าง ๆ ที่สามารถเลือกได้

จากนั้นเมื่อแสดงข้อความต่าง ๆ ครบแล้วก็จะรอการกดคีย์ว่าต้องการเข้าสู่เมนูใด ซึ่งที่หน้าจอ LCD ก็จะแสดงข้อความดังรูปที่ 4. 19

รูปที่ 4.19 หน้าจอแสดงข้อความเมื่อรอการกดคีย์

หากต้องการทำการเซตค่าหมายเลขให้กับรถประจำทางก็ให้ กดคีย์ 1 ที่หน้าจอ LCD ก็จะแสดงข้อความเพื่อให้ป้อนหมายเลขของรถประจำทาง ดังรูปที่ 4.20 จากนั้นให้ทำการป้อนค่าหมายเลขของรถประจำทาง จำนวน 4 หลัก โดยถ้าหากต้องการป้อนหมายเลขของรถประจำทางที่มีจำนวนน้อยกว่า 4 หลัก ก็ให้ทำการป้อนหมายเลข 0 ไว้ก่อนหน้าหมายเลขที่ต้องการตั้งค่า เช่น หากต้องการตั้งค่าหมายเลขรถเบอร์ 517 ก็ให้ป้อนเป็น 0517 หรือ ต้องการตั้งค่าหมายเลขรถเบอร์ 97 ก็ให้ป้อนเป็น 0097

เมื่อป้อนหมายเลขครบ 4 หลักแล้วก็จะทำการบันทึกหมายเลขของรถประจำทางไว้ แล้วก็จะออกจากเมนูนี้โดยกลับไปยังเมนูหลักอีกครั้ง ซึ่งที่หน้าจอ LCD ก็จะแสดงข้อความเหมือนตอนเริ่มต้นใหม่อีกครั้ง

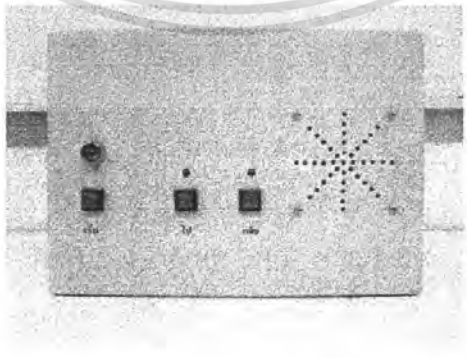


รูปที่ 4.20 หน้าจอแสดงข้อความให้บันทึกหมายเลขของรถ

หากต้องการตรวจสอบดูว่าข้อมูลที่ได้บันทึกไว้นั้นเป็นหมายเลขอะไรบ้างก็ให้ กดคีย์ 2 ที่หน้าจอ LCD ก็จะแสดงหมายเลขของรถประจำทางที่ได้บันทึกไว้ดัง รูปที่ 4.21 แล้วก็จะออกจากเมนูนี้ โดยกลับไปยังเมนูหลักอีกครั้ง ซึ่งที่หน้าจอ LCD ก็จะแสดงข้อความเหมือนตอนเริ่มต้นใหม่อีกครั้ง

รูปที่ 4.21 หน้าจอแสดงข้อมูลหมายเลขของรถที่ได้บันทึกไว้

หากต้องการเข้าสู่โปรแกรมหลักการสื่อสารของรถประจำทาง ก็ให้ กดปุ่ม GO หรือ ปุ่ม BACK ที่อยู่บนกล่องอุปกรณ์ของรถประจำทาง เพื่อเลือกเส้นทางของการสื่อสาร ดังรูปที่ 4.22 ซึ่งเมื่อ กดปุ่ม GO ก็จะเข้าสู่โปรแกรมหลักการสื่อสารของรถประจำทางในเส้นทางขาไปของรถประจำทาง และเมื่อ กดปุ่ม BACK ก็จะเข้าสู่โปรแกรมหลักการสื่อสารของรถประจำทางในเส้นทางขากลับของรถประจำทาง



รูปที่ 4.22 แสดงปุ่มกดเพื่อเลือกเส้นทางการสื่อสาร

เมื่อกดปุ่ม GO หรือ ปุ่ม BACK ที่หน้าจอ LCD ก็จะแสดงข้อความดังรูปที่ 4.23 อุปกรณ์ก็จะเริ่มทำการรับส่งข้อมูลตามเส้นทางที่กำหนด โดยจะเริ่มทำการรอรับสัญญาณข้อมูลที่ส่งมาจากป้ายรถประจำทาง แล้วนำไปตรวจสอบถ้าตรงตามที่กำหนดไว้ ก็จะแสดงผลเป็นข้อความเสียงว่าป้ายต่อไปชื่อว่าป้ายอะไร แล้วก็ส่งสัญญาณข้อมูลตอบกลับไปยังป้ายรถประจำทาง หรือหากมีข้อมูลของการเรียกรถเข้ามาก็จะแสดงผลเป็นเสียงเพื่อบอกให้คนขับรถรู้ว่ามีคนต้องการจะขึ้นรถที่ป้ายข้างหน้า

รูปที่ 4.23 หน้าจอแสดงข้อความเมื่อเข้าสู่โปรแกรมหลักการสื่อสารของรถประจำทาง

บทที่ 5
บทสรุปและวิจารณ์

5.1 สรุปผลการทดลอง

ในโครงงานป้ายรถประจำทางเพื่อผู้พิการทางสายตา จากการทดลองในส่วนของโปรแกรมย่อยต่าง ๆ เช่นการบันทึกข้อมูล การอ่านข้อมูลที่บันทึกไว้สามารถทำงานได้ดี เมื่อทดลองการรับส่งข้อมูลระหว่างป้ายรถประจำทางกับรถประจำทางด้วย RF Module (TRW-2.4GHz) ก็สามารถทำงานได้ดีโดยไม่มีการผิดพลาด และในส่วนของการแสดงผลทางเสียง ก็สามารถทำงานได้ดีเช่นกัน

5.2 ปัญหาที่พบระหว่างการดำเนินงาน

- ขาซ็อกเก็ต ของตัวโมดูล TRW 2.4 GHz มีขนาดเล็กทำให้ ขาหักง่าย ทำให้บางครั้ง พิจารณาอาการเสียของวงจรผิดพลาด

- ในการบันทึกเสียงให้กับ IC ISD25120 จะมีสัญญาณรบกวนสูง ต้องหาห้องเก็บเสียงและอุปกรณ์บันทึกเสียงที่ดี ในการบันทึกเสียง

5.3 แนวทางในการพัฒนาต่อ

สำหรับแนวทางในการพัฒนาต่อคือ ควรเขียนโปรแกรมให้มีความกระชับขึ้น และ การเก็บข้อมูลการให้บริการ เช่น การเก็บข้อมูลการใช้ความเร็ว ของรถแต่ละคัน โดยการเก็บค่าเวลา ว่ารถแต่ละคัน ผ่านป้ายแต่ละป้าย ในเวลาเท่าไร เพื่อไปคำนวณหาค่าความเร็ว เพื่อนำมาปรับปรุงการให้บริการเป็นต้น

หนังสืออ้างอิง

1. นคร ภักดีชาติ "ปฏิบัติการไมโครคอนโทรเลอร์ MCS-51 ด้วยโปรแกรมภาษาซี" Innovative Expeiment Press Thailand , กรุงเทพมหานคร
2. รศ.สมยศ จุณณะปิยะ "การประยุกต์ใช้งานไมโครคอนโทรเลอร์" สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , กรุงเทพมหานคร , 2546
3. ณรงค์ จิรวงศ์บุญรอด , ศักดิ์สิทธิ์ มณีกรณ์ และ อัจฉริยเทพ ตติยนันทพร " ป้ายรถประจำทางสำหรับผู้พิการทางสายตา " ภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2548
4. สุริยา เหลากลาง และ ปัทมา แสงมณี " ระบบสัญญาณไฟจราจรไร้สาย " ภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2549

ภาคผนวก

โปรแกรมที่ใช้ในการทดลองในส่วนของภาคส่ง

```
/************************************************/
/*                  include                     */
/************************************************/
#include <reg52.h>
sbit CE     = P1^0;  // Port Enable Shift
sbit CS     = P1^2;  // Port chip select
sbit CLK1   = P1^3;  // Port Clock befor send data chanel 1
sbit Data   = P1^4;  // Port send data to module 2.4G chane 1 1
sbit DR1    = P1^5;  // port select data to chanel1
/************************************************/
/*               Delay time                     */
/************************************************/
void dmsec (unsigned int count)   // Delay mSec Xtal= 11.0592 Mhz
{
    unsigned int i;
    while (count)
    {
        i = 225; while (i>0) i--;
        count--;
    }
}
void Wait(unsigned int x)
{
     unsigned int i;
     for (i=0;i<x;i++)
     {}
}
/************************************************/
/*            Initial TRW-2.4G                  */
/************************************************/
void Init_TRW24(void)
{
    CE    =  0;
    CS    =  0;
    CLK1  =  0;
    Data  =  0;
    DR1   =  0;
}
/************************************************/
/*               CLK TRW-2.4G                   */
/************************************************/
void CLK_1(void)
{
     CLK1 = 0;
     dmsec(1);
     CLK1 = 1;
     dmsec(1);
}
/************************************************/
/*           Write TRW-2.4G  CH1                */
/************************************************/
void Write_CH1(unsigned char Dat_1)
{
     unsigned char i;
     bit Out;
     for (i=0;i<8;i++)
     {
             Out = Dat_1 & 0x80;
             Data = Out;
```

```
            CLK_1();
            Dat_1 = Dat_1 << 1;
        }
    }
    /************************************************/
    /*              Set Mode TRW-2.4G               */
    /************************************************/
    void SetMode_TRW24(unsigned char Mode)
    {
        Wait(500);
        CE = 0;
        CS = 1;
        Write_CH1(0x8E);      // Reserved for testing
        Write_CH1(0x08);      // Reserved for testing
        Write_CH1(0x1C);      // Reserved for testing
        Write_CH1(0x08);      // Length of Bit Ch 2
        Write_CH1(0x08);      // Length of Bit Ch 1
        Write_CH1(0xC0);      // Address 5 Byte Ch 2
        Write_CH1(0xAA);
        Write_CH1(0x55);
        Write_CH1(0xAA);
        Write_CH1(0x55);
        Write_CH1(0xAA);      // Address 5 Byte Ch 1
        Write_CH1(0x55);
        Write_CH1(0xAA);
        Write_CH1(0x55);
        Write_CH1(0xAA);
        Write_CH1(0xA3);      // Number of Address bit + CRC
        Write_CH1(0x6F);      // 1 CH  250Kbps
        if (Mode == 1)              // Tx Mode CH1#1
        {
            Write_CH1(0x14);      // Tx Mode 2410MHz
        }
        else                        // Rx_Mode CH1#1
        {
            Write_CH1(0x15);      // Rx Mode 2410MHz
            Data=1; DR1=1; CE=1;
        }
        CS = 0;
        Wait(200);
    }
    /************************************************/
    /*                  Send data                   */
    /************************************************/
    void send_CH1(unsigned char dat)
    {
        Wait(500);
        CS = 0;
        CE = 1;
        Write_CH1(0xAA);       // Address 5 Byte Ch 1
        Write_CH1(0x55);
        Write_CH1(0xAA);
        Write_CH1(0x55);
        Write_CH1(0xAA);
        Write_CH1(dat);        // send data
        Wait(250);
        CLK1= 0;
        CE = 0;
        Wait(250);
        dmsec(500);
    }
```

```
/************************************************/
/*                Mian Program                  */
/************************************************/
void main (void)
{
     Init_TRW24();
     SetMode_TRW24(1);                  // set Config mode TX
     while(1)
     {
               send_CH1(0x95);
               dmsec(500);
               send_CH1(0xFF);
               dmsec(500);

               SetMode_TRW24(1);        // set Config mode TX
      }
}
```

โปรแกรมที่ใช้ในการทดลองในส่วนของภาครับ

```
/************************************************/
/*                  include                     */
/************************************************/
#include <reg52.h>
sbit CE     = P1^0;     // Port Enable Shift
sbit CS     = P1^2;     // Port chip select
sbit CLK1   = P1^3;     // Port Clock befor send data chanel 1
sbit Data   = P1^4;     // Port send data to module 2.4G chane 1 1
sbit DR1    = P1^5;     // port select data to chanel1
/************************************************/
/*                Delay time                    */
/************************************************/
void dmsec (unsigned int count)   //  Delay mSec Xtal= 11.0592 Mhz
{
    unsigned int i;
    while (count)
    {
        i = 225;while (i>0) i--;
        count--;
    }
}
void Wait(unsigned int x)
{
     unsigned int i;
     for (i=0;i<x;i++)
     {}
}
/************************************************/
/*              Initial TRW-2.4G                */
/************************************************/
void Init_TRW24(void)
{
    CE    =  0;
    CS    =  0;
    CLK1  =  0;
    Data  =  0;
    DR1   =  0;
}
```

```
/************************************************/
/*                 CLK TRW-2.4G                 */
/************************************************/
void CLK_1(void)
{
     CLK1 = 0;
     dmsec(1);
     CLK1 = 1;
     dmsec(1);
}
/************************************************/
/*            Write TRW-2.4G  CH1               */
/************************************************/
void Write_CH1(unsigned char Dat_1)
{
     unsigned char i;
     bit Out;
     for (i=0;i<8;i++){
            Out = Dat_1 & 0x80;
            Data = Out;
            CLK_1();
            Dat_1 = Dat_1 << 1;
     }
}
/************************************************/
/*             Set Mode TRW-2.4G                */
/************************************************/
void SetMode_TRW24( unsigned char Mode)
{
     Wait(500);
     CE = 0;
     CS = 1;
     Write_CH1(0x8E);      // Reserved for testing
     Write_CH1(0x08);      // Reserved for testing
     Write_CH1(0x1C);      // Reserved for testing
     Write_CH1(0x08);      // Length of Bit Ch 2
     Write_CH1(0x08);      // Length of Bit Ch 1
     Write_CH1(0xC0);      // Address 5 Byte Ch 2
     Write_CH1(0xAA);
     Write_CH1(0x55);
     Write_CH1(0xAA);
     Write_CH1(0x55);
     Write_CH1(0xAA);      // Address 5 Byte Ch 1
     Write_CH1(0x55);
     Write_CH1(0xAA);
     Write_CH1(0x55);
     Write_CH1(0xAA);
     Write_CH1(0xA3);      // Number of Address bit + CRC
     Write_CH1(0x6F);      // 1 CH  250Kbps
     if (Mode == 1)                  // Tx Mode CH1#1
     {
            Write_CH1(0x14);         // Tx Mode 2410MHz
     }
     else                            // Rx_Mode CH1#1
     {
            Write_CH1(0x15);         // Rx Mode 2410MHz
            Data=1; DR1=1; CE=1;
     }
     CS = 0;
     Wait(200);
}
```

```
/************************************************/
/*                Read TRW-2.4G                 */
/************************************************/
unsigned char Read_CH1(void)
{
     unsigned char i,dat1=0;
     bit Out1;
     Out1=0x00;
     Data = 1;                    // set P1.0 input channel 1
     for (i=0;i<8;i++)
     {
            dat1 = dat1 << 1;     // set dat = 0x00
            CLK1 = 1;
             Wait(50);
            Out1 = Data;          // send data out
            if (Out1)
            {
                 dat1 = dat1 + 0x01;
            }
                 CLK1 = 0;
                 Wait(50);
     }return(dat1);
}
/************************************************/
/*                Mian Program                  */
/************************************************/
void main (void)
{
     unsigned char DATA=0;
     P3=0xff;
     Init_TRW24();
     SetMode_TRW24(0);                  // set Config mode RX
      while(1)
      {
               while (DR1 == 0);
               DATA = Read_CH1();
               P3 = DATA;
               SetMode_TRW24(0);       // set Config mode RX
      }
}
```

โปรแกรมในส่วนของป้ายรถประจำทาง

```
/*****************************************************/
/*                      include                      */
/*****************************************************/
#include <reg52.h>
#include <string.h>
#include <intrins.h>
sbit  E    = P3^6;
sbit  RS   = P3^7;
sbit  PD   = P2^5;
sbit  SDA  = P2^6;
sbit  SCL  = P2^7;
sbit  led  = P3^4;
sbit  key_press = P2^4;
sbit CE     = P1^0;       // Port Enable Shift
sbit CS     = P1^2;       // Port chip select
sbit CLK1   = P1^3;       // Port Clock befor send data chanel 1
sbit Data   = P1^4;       // Port send data to module 2.4G chanel 1
sbit DR1    = P1^5;       // port select data to chanel1
unsigned char x1,x2,x3,MEM[3],NUM[3],RXin[6];
unsigned char x = 0;
/*****************************************************/
/*                   Delay time                      */
/*****************************************************/
void dmsec (unsigned int count)    /* Delay mSec Xtal= 11.0592 MHz */
{
      unsigned int i;
      while (count)
      {
          i = 225; while (i>0) i--;
          count--;
      }
}
void Wait(unsigned int x)
{
      unsigned int i;
      for (i=0;i<x;i++)
      {}
}
/*****************************************************/
/*                Initial TRW-2.4G                   */
/*****************************************************/
void Init_TRW24(void)
{
      CE    =  0;
      CS    =  0;
      CLK1  =  0;
      Data  =  0;
      DR1   =  0;
}
/*****************************************************/
/*                  CLK TRW-2.4G                     */
/*****************************************************/
void CLK_1(void)
{
      CLK1 = 0;
      dmsec(1);
      CLK1 = 1;
      dmsec(1);
}
```

```
/*****************************************************/
/*                Write TRW-2.4G  CH1                */
/*****************************************************/
void Write_CH1(unsigned char Dat_1)
{
      unsigned char i;
      bit Out;
      for (i=0;i<8;i++)
      {
            Out = Dat_1 & 0x80;
            Data = Out;
            CLK_1();
            Dat_1 = Dat_1 << 1;
      }
}
/*****************************************************/
/*                 Set Mode TRW-2.4G                 */
/*****************************************************/
void SetMode_TRW24( unsigned char Mode)
{
      Wait(500);
      CE = 0;
      CS = 1;
      Write_CH1(0x8E);      /* Reserved for testing */
      Write_CH1(0x08);      /* Reserved for testing */
      Write_CH1(0x1C);      /* Reserved for testing */
      Write_CH1(0x08);      /* Length of Bit Ch 2   */
      Write_CH1(0x08);      /* Length of Bit Ch 1   */
      Write_CH1(0xC0);      /* Address 5 Byte Ch 2  */
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);      /* Address 5 Byte Ch 1  */
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0xA3);      /* Number of Address bit + CRC */
      Write_CH1(0x6F);      /* 1 CH  250Kbps               */
      if (Mode == 1)                    /* Tx Mode CH1#1    */
      {
            Write_CH1(0x14);            /* Tx Mode 2410MHz  */
      }
      else                              /* Rx_Mode CH1#1    */
      {
            Write_CH1(0x15);            /* Rx Mode 2410MHz */
            Data=1; DR1=1; CE=1;
      }
      CS = 0;
      Wait(200);
}
/*****************************************************/
/*                  Read TRW-2.4G                    */
/*****************************************************/
unsigned char Read_CH1(void)
{
      unsigned char i,dat1=0;
      bit Out1;
      Out1 = 0x00;
      Data = 1;                 /* set P1.0 input channel 1  */
```

```
      for (i=0;i<8;i++)
      {
            dat1 = dat1 << 1;        /* set dat = 0x00 */
            CLK1 = 1;
            Wait(50);
            Out1 = Data;              /* send data out */
            if (Out1)
            {
                  dat1 = dat1 + 0x01;
            }
                  CLK1 = 0;
                  Wait(50);
      }return(dat1);
}
/*******************************************************/
/*                     Send data                       */
/*******************************************************/
void send_CH1(unsigned char dat)
{
      Wait(500);
      CS = 0;
      CE = 1;
      Write_CH1(0xAA);       /* Address 5 Byte Ch 1 */
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(dat);        /* send data */
      Wait(250);
      CLK1= 0;
      CE = 0;
      Wait(250);
}
/*******************************************************/
/*            Function Delay time                      */
/*******************************************************/
void delayfifty(int fiftyms)
{
      int  x;
      for (x = 0;x<fiftyms;x++)
      {
            TR0 = 0x4c;
            TL0 = 0x00;
            TF0 = 0;
            TR0 = 1;
            while(TF0 == 0);
            TR0 = 0;
      }
}
/*******************************************************/
/*                   Function  LCD                     */
/*******************************************************/
void lcd_delay(int tick)
{
      int i,j ;
      for(i=0;i<tick;i++)
      for(j=0;j<250;j++);
}
void lcd_command(unsigned char com)
{
      RS = 0;
```

```
        E  = 1;
        P0 = com;
        lcd_delay(5);
        E  = 0;
        lcd_delay(5);
}
void lcd_text(unsigned char text)
{
        RS = 1;
        E  = 1;
        P0 = text;
        lcd_delay(5);
        E  = 0;
        lcd_delay(5);
}
void lcd_init()
{
        lcd_delay(500);
        lcd_command(0x38);
        lcd_command(0x0c);
        lcd_command(0x01);
}
void lcd_show(unsigned char text[],bit row)
{
        unsigned char a;
        if(row == 0)
                lcd_command(0x80);
        else
                lcd_command(0xC0);
        for(a = 0;a < strlen(text);a++)
                lcd_text(text[a]);
}
void key2lcd(unsigned char pad)
{
        pad += 0x30;
        lcd_text(pad);
}
void lcd_clear(void)
{
        lcd_command(0x01);
}
void lcd_blink()
{
        lcd_command(0x0F);
}
/*******************************************************/
/*                  Function i2c bus                   */
/*******************************************************/
void i2c_delay(void)
{
        unsigned char i;
        for(i=0;i<20;i++)
        _nop_ ();
}
void i2c_clk(void)
{
        i2c_delay();
        SCL = 1;
        i2c_delay();
        SCL = 0;
}
```

```
void i2c_start(void)
{
      if(SCL)
      SCL = 0;
      SDA = 1;
      SCL = 1;
      i2c_delay();
      SDA = 0;
      i2c_delay();
      SCL = 0;
}
void i2c_stop(void)
{
      if(SCL)
      SCL = 0;
      SDA = 0;
      i2c_delay();
      SCL = 1;
      i2c_delay();
      SDA = 1;
}
void i2c_NACK(void)
{
      SDA = 1;
      i2c_delay();
      i2c_clk();
      SCL = 1;
}
bit i2c_wrdata(unsigned char dat)
{
      bit data_bit;
      unsigned char i;
      for(i=0;i<8;i++)
      {
            data_bit = dat & 0x80;
            SDA = data_bit;
            i2c_clk();
            dat = dat<<1;
      }
      SDA = 1;
      i2c_delay();
      SCL = 1;
      i2c_delay();
      data_bit = SDA;
      SCL = 0;
      i2c_delay();
      return(data_bit);
}
unsigned char i2c_rddata(void)
{
      bit rd_bit;
      unsigned char i,dat;
      dat = 0x00;
      for(i=0;i<8;i++)
      {
          i2c_delay();
          SCL =1;
          i2c_delay();
          rd_bit = SDA;
          dat = dat << 1;
          dat = dat | rd_bit;
```

```
            SCL =  0;
        }
        SDA = 1;
        i2c_delay();
        i2c_clk();
        SCL = 1;
        return(dat);
}
/*******************************************************/
/*       Function Read/Write data in EEPROM            */
/*******************************************************/
void Wr_EEPROM(unsigned int Addr,unsigned char Data)
{
        i2c_start();
        i2c_wrdata(0xA0);
        i2c_wrdata(Addr >> 8);
        i2c_wrdata(Addr & 0x00FF);
        i2c_wrdata(Data);
        i2c_stop();
        lcd_delay(50);
}
unsigned char Rd_EEPROM(unsigned int Addr)
{
        unsigned char Temp;
        i2c_start();
        i2c_wrdata(0xA0);
        i2c_wrdata(Addr >> 8);
        i2c_wrdata(Addr & 0x00ff);
        i2c_start();
        i2c_wrdata(0xA1);
        Temp = i2c_rddata();
        i2c_NACK();
        i2c_stop();
        return(Temp);
}
/*******************************************************/
/*            Function Scan keypad 4x3                 */
/*******************************************************/
unsigned char scankey(void)
{
   unsigned char d_key,key_data;
             delayfifty(20);
           while(~key_press);
             d_key = P2;
             d_key &= 0x0F;
                   if(d_key == 0x00)
                         key_data = 0x01;
                   else if(d_key == 0x01)
                         key_data = 0x02;
                   else if(d_key == 0x02)
                         key_data = 0x03;
                   else if(d_key == 0x04)
                         key_data = 0x04;
                   else if(d_key == 0x05)
                         key_data = 0x05;
                   else if(d_key == 0x06)
                         key_data = 0x06;
                   else if(d_key == 0x08)
                         key_data = 0x07;
                   else if(d_key == 0x09)
                         key_data = 0x08;
```

```
                else if(d_key == 0x0A)
                        key_data = 0x09;
                else if(d_key == 0x0C)
                        key_data = 0x18;
                else if(d_key == 0x0D)
                        key_data = 0x00;
                else if(d_key == 0x0E)
                        key_data = 0x19;
                else
                        key_data = 0x1A;
                return(key_data);
}
/*******************************************************/
/*******************************************************/
unsigned char buff_1(unsigned char buff)
{
      buff &= 0x0F;
      return (buff);
}
unsigned char buff_2(unsigned char buff)
{
      buff &= 0xF0;
      buff  = buff >> 4;
      return(buff);
}
unsigned char buffer(unsigned char bufa,bufb)
{
      bufa = bufa << 4;
      bufa = bufa | bufb;
      return(bufa);
}
void two2lcd(unsigned char st)
{
      unsigned char a;
      a = buff_2(st);
      key2lcd(a);
      a = buff_1(st);
      key2lcd(a);
}
void two2lcd2(unsigned char st,st2)
{
      unsigned char a;
      a = buff_2(st);
      key2lcd(a);
      a = buff_1(st);
      key2lcd(a);
      a = buff_2(st2);
      key2lcd(a);
      a = buff_1(st2);
      key2lcd(a);
}
/*******************************************************/
/*              Function show data                     */
/*******************************************************/
void show_1(void)
{
      lcd_clear();
      lcd_show(" INSERT NUMBER ",0);
      lcd_show(" BUS STOP ",1);
      lcd_blink();
}
```

```
void show_2(unsigned char stop1,stop2)
{
      lcd_clear();
      lcd_show("   BUS STOP ",0);
      lcd_show(" NUMBER ",1);
      two2lcd2(stop1,stop2);
}
/*******************************************************/
/*                Function Show Sound                  */
/*******************************************************/
void soundi2c(unsigned char m)
{
      i2c_start();
      Wait(3);
      i2c_wrdata(0x70);
      i2c_wrdata(m);
      Wait(3);
      i2c_stop();
}
void show_sound1(void)
{
      soundi2c(0x1A);
      PD = 0;
      delayfifty(100);
      PD = 1;
      delayfifty(2);
}
void show_sound2(void)
{
      soundi2c(0x06);
      PD = 0;
      delayfifty(200);
      PD = 1;
}
void show_sound3(void)
{
      PD = 1;
      delayfifty(2);
      soundi2c(0x16);
      PD = 0;
      delayfifty(200);
      PD = 1;
      delayfifty(1);
}
void show_sound4(void)
{
      PD = 1;
      delayfifty(2);
      soundi2c(0x0E);
      PD = 0;
      delayfifty(200);
      PD = 1;
}
void show_sound5(void){
      PD = 1;
      delayfifty(2);
      soundi2c(0x1E);
      PD = 0;
      delayfifty(200);
      PD = 1;
}
```

```
void sound(unsigned char s)
{
      if (s == 0x00)
         soundi2c(0x0A);
      else if (s == 0x01)
         soundi2c(0x10);
      else if (s == 0x02)
         soundi2c(0x08);
      else if (s == 0x03)
         soundi2c(0x18);
      else if (s == 0x04)
         soundi2c(0x04);
      else if (s == 0x05)
         soundi2c(0x14);
      else if (s == 0x06)
         soundi2c(0x0C);
      else if (s == 0x07)
         soundi2c(0x1C);
      else if (s == 0x08)
         soundi2c(0x02);
      else
         soundi2c(0x12);
      PD = 0;
      delayfifty(152);
      PD = 1;
}
void check_sound(unsigned char y)
{
      x1 = buff_2(y);
      if(x1 != 0x00)
        {
            PD = 1;
            sound(x1);
            delayfifty(1);
            PD = 1;
        }
      x2 = buff_1(y);
      if( x2 != 0x00)
      {
            PD = 1;
            sound(x2);
            delayfifty(1);
            PD = 1;
      }
      else if(x1 != 0x00)
      {
            PD = 1;
            sound(x2);
            delayfifty(1);
            PD = 1;
      }
}
void check_sound2(unsigned char y)
{
      unsigned char s,t;
      s = buff_2(y);
      if(s != 0x00)
      {
            PD = 1;
            sound(s);
            delayfifty(1);
```

```
            PD = 1;
            t = buff_1(y);
            sound(t);
            delayfifty(1);
            PD = 1;
      }
      else
      {
            if(x2 != 0x00)
            {
                  PD = 1;
                  sound(s);
                  delayfifty(1);
                  PD = 1;
                  t = buff_1(y);
                  sound(t);
                  delayfifty(1);
                  PD = 1;
            }
            else
            {
                  if(x1 != 0x00)
                  {
                        PD = 1;
                        sound(s);
                        delayfifty(1);
                        PD = 1;
                        t = buff_1(y);
                        sound(t);
                        delayfifty(1);
                        PD = 1;
                  }
                  else
                  {
                        t = buff_1(y);
                        PD = 1;
                        sound(t);
                        delayfifty(1);
                        PD = 1;
                  }
            }
      }
}
/*****************************************************/
/*                Function Interrupt                 */
/*****************************************************/
void service_0(void) interrupt 0
{
      PD = 1;
      soundi2c(x3);
      PD = 0;
      delayfifty(100);
      PD = 1;
      SetMode_TRW24(1);
      send_CH1(0xCC);
      send_CH1(RXin[0]);
      send_CH1(RXin[1]);
      send_CH1(MEM[1]);
      send_CH1(MEM[2]);
      x3 = 0xFF;
}
```

```
void service_1(void) interrupt 2
{
     PD = 1;
     soundi2c(x3);
     PD = 0;
     delayfifty(100);
     PD = 1;
     SetMode_TRW24(1);
     send_CH1(0xCC);
     send_CH1(RXin[2]);
     send_CH1(RXin[3]);
     send_CH1(MEM[1]);
     send_CH1(MEM[2]);
     x3 = 0xFF;
}
/*******************************************************/
/*                  Function receive                   */
/*******************************************************/
void receive_1(void)
{
      PD = 1;
      show_sound1();
      check_sound(RXin[0]);
      check_sound2(RXin[1]);
      show_sound2();
      delayfifty(1);
      PD = 1;
      show_sound3();
      delayfifty(1);
      PD = 1;
      show_sound4();
      delayfifty(1);
      PD = 1;
}
void receive_2(void)
{
      PD = 1;
      show_sound1();
      check_sound(RXin[2]);
      check_sound2(RXin[3]);
      show_sound2();
      delayfifty(1);
      PD = 1;
      show_sound3();
      delayfifty(1);
      PD = 1;
      show_sound5();
      delayfifty(1);
      PD = 1;
}
/*******************************************************/
/*                  Function Set_num                   */
/*******************************************************/
void set_num(void)
{
      unsigned char keya,keyb,keyc,keyd;
      show_1();
      keya = scankey();
      delayfifty(20);
      key2lcd(keya);
      lcd_blink();
```

```
      keyb = scankey();
      key2lcd(keyb);
      MEM[1] = buffer(keya,keyb);
      delayfifty(20);
      keyc = scankey();
      delayfifty(20);
      key2lcd(keyc);
      lcd_blink();
      keyd = scankey();
      key2lcd(keyd);
      delayfifty(20);
      MEM[2] = buffer(keyc,keyd);
      Wr_EEPROM(0x02FF,MEM[1]);
      Wr_EEPROM(0x01FF,MEM[2]);
}
/*******************************************************/
/*                Function check_num                   */
/*******************************************************/
void check_num(void)
{
      MEM[1] = Rd_EEPROM(0x02FF);
      MEM[2] = Rd_EEPROM(0x01FF);
      lcd_init();
      show_2(MEM[1],MEM[2]);
      lcd_delay(1000);
}
/*******************************************************/
/*                Function send_num                    */
/*******************************************************/
void send_num(void)
{
      unsigned char DATA;
      MEM[1] = Rd_EEPROM(0x02FF);
      MEM[2] = Rd_EEPROM(0x01FF);
      Init_TRW24();
      IT0 = 1;
      IT1 = 1;
      EA  = 1;
      EX0 = 1;
      EX1 = 1;
      while(1)
      {
        do
          {
           SetMode_TRW24(1);
           lcd_clear();
           lcd_show("NOW SEND DATA",0);
           send_CH1(0xAA);
           send_CH1(MEM[1]);
           send_CH1(MEM[2]);
           SetMode_TRW24(0);
           x3 = 0x30;
           dmsec(1680);
           }while(DR1 == 0);
                  DATA = Read_CH1();
                  NUM[0] = DATA;
                  if(NUM[0] == 0xBB)
                  {
                         while(DR1 == 0);
                         DATA = Read_CH1();
                         NUM[1] = DATA;
```

```
                        while(DR1 == 0);
                        DATA = Read_CH1();
                        NUM[2] = DATA;
                    if(NUM[1]==MEM[1] & NUM[2]==MEM[2])
                    {
                         while(DR1 == 0);
                         DATA = Read_CH1();
                         RXin[x] = DATA;
                         x++;
                         while(DR1 == 0);
                         DATA = Read_CH1();
                         RXin[x] = DATA;
                         x++;
                         if(x == 2)
                         {
                               receive_1();
                         }
                         else if(x == 4)
                         {
                               x = 0;
                               receive_2();
                         }
                    }
               }
          }
}
/*****************************************************/
/*                  Mian Program                     */
/*****************************************************/
void main (void)
{
      unsigned char key;
      PD = 1;
      led = 0;
      lcd_init();
      lcd_show("     WELCOME",0);
      delayfifty(40);
      while(1)
      {
            lcd_clear();
            lcd_show("MAIN PROGRAM",0);
            lcd_show("PRESS 1 ",1);
            delayfifty(100);
            lcd_clear();
            lcd_show("SET STATION",0);
            lcd_show("PRESS 2 ",1);
            delayfifty(100);
            lcd_clear();
            lcd_show("CHECK STATION",0);
            lcd_show("PRESS 3 ",1);
            delayfifty(100);
            lcd_clear();
            lcd_show("CHOOSE NUMBER ",0);
            lcd_blink();
            key = scankey();
      if(key == 0x01)
      {
            key2lcd(key);
            delayfifty(30);
            send_num();
      }
```

```
      else if(key == 0x02)
      {
            key2lcd(key);
            delayfifty(30);
            set_num();
      }
      else if(key == 0x03)
      {
            key2lcd(key);
            delayfifty(30);
            check_num();
      }
      else delayfifty(10);
   }
}
```

โปรแกรมในส่วนของรถประจำทาง

```
/*****************************************************/
/*                     include                       */
/*****************************************************/
#include <reg52.h>
#include <string.h>
#include <intrins.h>
sbit  E     = P3^6;
sbit  RS    = P3^7;
sbit  PD    = P2^5;
sbit  SDA   = P2^6;
sbit  SCL   = P2^7;
sbit  key_press = P2^4;
sbit  way_go    = P3^0;
sbit  way_back  = P3^1;
sbit  led_show  = P3^4;
sbit  CE      = P1^0;       // Port Enable Shift
sbit  CS      = P1^2;       // Port chip select
sbit  CLK1    = P1^3;       // Port Clock befor send data chanel 1
sbit  Data    = P1^4;       // Port send data to module 2.4G chanel 1
sbit  DR1     = P1^5;       // port select data to chanel1
unsigned char MEM[3],NUM[3],RXin[5],data_in1,data_in2;
/*****************************************************/
/*                  Delay time                       */
/*****************************************************/
void dmsec (unsigned int count)   /*  Delay mSec Xtal= 11.0592 MHz */
{
      unsigned int i;
      while (count){
          i = 225; while (i>0) i--;
          count--;
      }
}
void Wait(unsigned int x)
{
      unsigned int i;
      for (i=0;i<x;i++)
      {}
}
/*****************************************************/
/*               Initial TRW-2.4G                    */
/*****************************************************/
void Init_TRW24(void)
{
```

```
      CE    =  0;
      CS    =  0;
      CLK1  =  0;
      Data  =  0;
      DR1   =  0;
}
/*******************************************************/
/*                  CLK TRW-2.4G                       */
/*******************************************************/
void CLK_1(void)
{
      CLK1 = 0;
      dmsec(1);
      CLK1 = 1;
      dmsec(1);
}
/*******************************************************/
/*                Write TRW-2.4G  CH1                  */
/*******************************************************/
void Write_CH1(unsigned char Dat_1)
{
      unsigned char i;
      bit Out;
      for (i=0;i<8;i++)
      {
            Out = Dat_1 & 0x80;
            Data = Out;
            CLK_1();
            Dat_1 = Dat_1 << 1;
      }
}
/*******************************************************/
/*                Set Mode TRW-2.4G                    */
/*******************************************************/
void SetMode_TRW24( unsigned char Mode)
{
      Wait(500);
      CE = 0;
      CS = 1;
      Write_CH1(0x8E);     /* Reserved for testing */
      Write_CH1(0x08);     /* Reserved for testing */
      Write_CH1(0x1C);     /* Reserved for testing */
      Write_CH1(0x08);     /* Length of Bit Ch 2   */
      Write_CH1(0x08);     /* Length of Bit Ch 1   */
      Write_CH1(0xC0);     /* Address 5 Byte Ch 2  */
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);     /* Address 5 Byte Ch 1  */
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0xA3);     /* Number of Address bit + CRC */
      Write_CH1(0x6F);     /* 1 CH  250Kbps               */
      if (Mode == 1)                  /* Tx Mode CH1#1  */
      {
            Write_CH1(0x14);          /* Tx Mode 2410MHz*/
      }
      else                            /* Rx_Mode CH1#1  */
```

```
      {
            Write_CH1(0x15);          /* Rx Mode 2410MHz*/
            Data=1; DR1=1; CE=1;
      }
      CS = 0;
      Wait(200);
}
/*****************************************************/
/*                  Read TRW-2.4G                    */
/*****************************************************/
unsigned char Read_CH1(void)
{
      unsigned char i,dat1=0;
      bit Out1;
      Out1=0x00;
      Data = 1;                     /* set P1.0 input channel 1 */
      for (i=0;i<8;i++)
      {
            dat1 = dat1 << 1;       /* set dat = 0x00 */
            CLK1 = 1;
            Wait(50);
            Out1 = Data;            /* send data out */
            if (Out1)
            {
               dat1 = dat1 + 0x01;
            }
            CLK1 = 0;
            Wait(50);
      }return(dat1);
}
/*****************************************************/
/*                  Send data                        */
/*****************************************************/
void send_CH1(unsigned char dat)
{
      Wait(500);
      CS = 0;
      CE = 1;
      Write_CH1(0xAA);        /* Address 5 Byte Ch 1 */
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(0x55);
      Write_CH1(0xAA);
      Write_CH1(dat);         /* send data */
      Wait(250);
      CLK1= 0;
      CE = 0;
      Wait(250);
}
/*****************************************************/
/*             Function Delay time                   */
/*****************************************************/
void delayfifty(int fiftyms)
{
      int  x;
      for (x = 0;x<fiftyms;x++)
      {
            TR0 = 0x4c;
            TL0 = 0x00;
            TF0 = 0;
            TR0 = 1;
```

```
            while(TF0 == 0);
            TR0 = 0;
      }
}
/*****************************************************/
/*                  Function  LCD                    */
/*****************************************************/
void lcd_delay(int tick)
{
      int i,j ;
      for(i=0;i<tick;i++)
      for(j=0;j<250;j++);
}
void lcd_command(unsigned char com)
{
      RS = 0;
      E  = 1;
      P0 = com;
      lcd_delay(5);
      E  = 0;
      lcd_delay(5);
}
void lcd_text(unsigned char text)
{
      RS = 1;
      E  = 1;
      P0 = text;
      lcd_delay(5);
      E  = 0;
      lcd_delay(5);
}
void lcd_init()
{
      lcd_delay(500);
      lcd_command(0x38);
      lcd_command(0x0c);
      lcd_command(0x01);
}
void lcd_show(unsigned char text[],bit row)
{
      unsigned char a;
      if(row == 0)
          lcd_command(0x80);
      else
          lcd_command(0xC0);
      for(a = 0;a < strlen(text);a++)
          lcd_text(text[a]);
}
void key2lcd(unsigned char pad)
{
      pad += 0x30;
      lcd_text(pad);
}
void lcd_clear(void)
{
      lcd_command(0x01);
}
void lcd_blink()
{
      lcd_command(0x0F);
}
```

```
/*******************************************************/
/*                  Function i2c bus                   */
/*******************************************************/
void i2c_delay(void)
{
      unsigned char i;
      for(i=0;i<20;i++)
      _nop_ ();
}
void i2c_clk(void)
{
      i2c_delay();
      SCL = 1;
      i2c_delay();
      SCL = 0;
}
void i2c_start(void)
{
      if(SCL)
      SCL = 0;
      SDA = 1;
      SCL = 1;
      i2c_delay();
      SDA = 0;
      i2c_delay();
      SCL = 0;
}
void i2c_stop(void)
{
      if(SCL)
      SCL = 0;
      SDA = 0;
      i2c_delay();
      SCL = 1;
      i2c_delay();
      SDA = 1;
}
void i2c_NACK(void)
{
      SDA = 1;
      i2c_delay();
      i2c_clk();
      SCL = 1;
}
bit i2c_wrdata(unsigned char dat)
{
      bit data_bit;
      unsigned char i;
      for(i=0;i<8;i++)
      {
            data_bit = dat & 0x80;
            SDA = data_bit;
            i2c_clk();
            dat = dat<<1;
      }
      SDA = 1;
      i2c_delay();
      SCL = 1;
      i2c_delay();
      data_bit = SDA;
      SCL = 0;
```

```
      i2c_delay();
      return(data_bit);
}
unsigned char i2c_rddata(void)
{
      bit rd_bit;
      unsigned char i,dat;
      dat = 0x00;
      for(i=0;i<8;i++)
      {
          i2c_delay();
          SCL =1;
          i2c_delay();
          rd_bit = SDA;
          dat = dat << 1;
          dat = dat | rd_bit;
          SCL =  0;
      }
      SDA = 1;
      i2c_delay();
      i2c_clk();
      SCL = 1;
      return(dat);
}
/*******************************************************/
/*       Function Read/Write data in EEPROM            */
/*******************************************************/
void Wr_EEPROM(unsigned int Addr,unsigned char Data)
{
      i2c_start();
      i2c_wrdata(0xA0);
      i2c_wrdata(Addr >> 8);
      i2c_wrdata(Addr & 0x00FF);
      i2c_wrdata(Data);
      i2c_stop();
      lcd_delay(50);
}
unsigned char Rd_EEPROM(unsigned int Addr)
{
      unsigned char Temp;
      i2c_start();
      i2c_wrdata(0xA0);
      i2c_wrdata(Addr >> 8);
      i2c_wrdata(Addr & 0x00ff);
      i2c_start();
      i2c_wrdata(0xA1);
      Temp = i2c_rddata();
      i2c_NACK();
      i2c_stop();
      return(Temp);
}
/*******************************************************/
/*            Function Scan keypad 4x3                 */
/*******************************************************/
unsigned char scankey(void)
{
      unsigned char d_key,key_data;
        while(~key_press);
            delayfifty(20);
            d_key = P2;
            d_key &= 0x0F;
```

```
                    if(d_key == 0x00)
                          key_data = 0x01;
                    else if(d_key == 0x01)
                          key_data = 0x02;
                    else if(d_key == 0x02)
                          key_data = 0x03;
                    else if(d_key == 0x04)
                          key_data = 0x04;
                    else if(d_key == 0x05)
                          key_data = 0x05;
                    else if(d_key == 0x06)
                          key_data = 0x06;
                    else if(d_key == 0x08)
                          key_data = 0x07;
                    else if(d_key == 0x09)
                          key_data = 0x08;
                    else if(d_key == 0x0A)
                          key_data = 0x09;
                    else if(d_key == 0x0C)
                          key_data = 0x18;
                    else if(d_key == 0x0D)
                          key_data = 0x00;
                    else if(d_key == 0x0E)
                          key_data = 0x19;
                    else
                          key_data = 0x1A;
                    return(key_data);
}
/*****************************************************/
/*****************************************************/
unsigned char buff_1(unsigned char buff)
{
      buff &= 0x0F;
      return (buff);
}
unsigned char buff_2(unsigned char buff)
{
      buff &= 0xF0;
      buff = buff >> 4;
      return(buff);
}
unsigned char bufferl(unsigned char bufa,bufb)
{
      bufa = bufa << 4;
      bufa = bufa | bufb;
      return(bufa);
}
unsigned char buffer2(unsigned char bufc,bufd)
{
      bufc = bufc << 4;
      bufc = bufc | bufd;
      return(bufc);
}
void two2lcd2(unsigned char st,st2)
{
      unsigned char a;
      a = buff_2(st);
      key2lcd(a);
      a = buff_1(st);
      key2lcd(a);
      a = buff_2(st2);
```

```
      key2lcd(a);
      a = buff_1(st2);
      key2lcd(a);
}
/*****************************************************/
/*               Function show data                  */
/*****************************************************/
void show_1(void)
{
      lcd_clear();
      lcd_show("   INSERT",0);
      lcd_show("NUMBER BUS ",1);
      lcd_blink();
}
void show_2(unsigned char num1,num2)
{
      lcd_clear();
      lcd_show("NUMBER BUS ",0);
      two2lcd2(num1,num2);
}
/*****************************************************/
/*               Function Show Sound                 */
/*****************************************************/
void soundi2c(unsigned char m)
{
      i2c_start();
      i2c_wrdata(0x70);
      i2c_wrdata(m);
      i2c_stop();
}
void show_sound1(void)
{
      soundi2c(0x02);
      PD = 0;
      delayfifty(100);
      PD = 1;
}
void show_sound2(void)
{
      soundi2c(0x04);
      PD = 0;
      delayfifty(150);
      PD = 1;
}
void show_sound(unsigned char s,y)
{
      if(s==0x20 & y==0x01)
           soundi2c(0x06);
      else if(s==0x20 & y==0x02)
           soundi2c(0x08);
      else if(s==0x20 & y==0x03)
           soundi2c(0x0A);
      else if(s==0x20 & y==0x04)
           soundi2c(0x0C);
      else if(s==0x20 & y==0x05)
           soundi2c(0x0E);
      else if(s==0x20 & y==0x06)
           soundi2c(0x12);
      else if(s==0x20 & y==0x07)
           soundi2c(0x14);
      else if(s==0x30 & y==0x07)
```

```
            soundi2c(0x06);
      else if(s==0x30 & y==0x06)
            soundi2c(0x08);
      else if(s==0x30 & y==0x05)
            soundi2c(0x0A);
      else if(s==0x30 & y==0x04)
            soundi2c(0x0C);
      else if(s==0x30 & y==0x03)
            soundi2c(0x0E);
      else if(s==0x30 & y==0x02)
            soundi2c(0x12);
      else if(s==0x30 & y==0x01)
            soundi2c(0x14);
      PD = 0;
      delayfifty(180);
      PD = 1;
}
void show_request(void)
{
      PD = 1;
      led_show = 0;
      show_sound1();
      delayfifty(1500);
      led_show = 1;
}
void check_request_go(unsigned char s,y)
{
      if(s==0x20 & y==0x01)
        show_request();
      else if(s==0x20 & y==0x02)
        show_request();
      else if(s==0x20 & y==0x03)
        show_request();
      else if(s==0x20 & y==0x04)
        show_request();
      else if(s==0x20 & y==0x05)
        show_request();
      else if(s==0x20 & y==0x06)
        show_request();
      else if(s==0x20 & y==0x07)
        show_request();
}
void check_request_back(unsigned char s,y)
{
      if(s==0x30 & y==0x07)
        show_request();
      else if(s==0x30 & y==0x06)
        show_request();
      else if(s==0x30 & y==0x05)
        show_request();
      else if(s==0x30 & y==0x04)
        show_request();
      else if(s==0x30 & y==0x03)
        show_request();
      else if(s==0x30 & y==0x02)
        show_request();
      else if(s==0x30 & y==0x01)
        show_request();
}
```

```
/*******************************************************/
/*                  Function send_data                 */
/*******************************************************/
void send_data_go(void)
{
     unsigned char DATA,i=0;
     MEM[1]=Rd_EEPROM(0x02FF);
     MEM[2]=Rd_EEPROM(0x01FF);
     Init_TRW24();
     SetMode_TRW24(1);
     led_show = 1;
     send_CH1(0xBB);
     send_CH1(NUM[1]);
     send_CH1(NUM[2]);
     send_CH1(MEM[1]);
     send_CH1(MEM[2]);
     data_in1 = NUM[1];
     data_in2 = NUM[2];
     for(i=0;i<20;i++)
     {
          SetMode_TRW24(0);
          while (DR1 == 0);
             DATA   = Read_CH1();
             RXin[0] = DATA;
          if(RXin[0] == 0xCC)
          {
             while(DR1 == 0);
                DATA   = Read_CH1();
                RXin[1] = DATA;
             while (DR1 == 0);
                DATA   = Read_CH1();
                RXin[2] = DATA;
             if(RXin[1]==MEM[1] & RXin[2]==MEM[2])
             {
                while(DR1 == 0);
                   DATA   = Read_CH1();
                   RXin[3] = DATA;
                while (DR1 == 0);
                   DATA   = Read_CH1();
                   RXin[4] = DATA;
                   check_request_go(RXin[3],RXin[4]);
             }
          }
      }
}
void send_data_back(void)
{
     unsigned char DATA , i = 0;
     MEM[1] = Rd_EEPROM(0x02FF);
     MEM[2] = Rd_EEPROM(0x01FF);
     Init_TRW24();
     SetMode_TRW24(1);
     led_show = 1;
     send_CH1(0xBB);
     send_CH1(NUM[1]);
     send_CH1(NUM[2]);
     send_CH1(MEM[1]);
     send_CH1(MEM[2]);
     data_in1 = NUM[1];
     data_in2 = NUM[2];
```

```
      for(i=0;i<20;i++)
      {
             SetMode_TRW24(0);
          while (DR1 == 0);
             DATA   = Read_CH1();
             RXin[0] = DATA;
          if(RXin[0] == 0xCC)
          {
             while(DR1 == 0);
                  DATA   = Read_CH1();
                  RXin[1] = DATA;
              while (DR1 == 0);
                  DATA   = Read_CH1();
                  RXin[2] = DATA;
              if(RXin[1]==MEM[1] & RXin[2]==MEM[2])
              {
                  while(DR1 == 0);
                       DATA   = Read_CH1();
                       RXin[3] = DATA;
                  while (DR1 == 0);
                       DATA   = Read_CH1();
                       RXin[4] = DATA;
                       check_request_back(RXin[3],RXin[4]);
              }
          }
      }
}
/*******************************************************/
/*                    check_data                       */
/*******************************************************/
void check_data_go(void)
{
    if(MEM[1]==0x01 & MEM[2]==0x43)
      {
          if(NUM[1]==0x20 & NUM[2]==0x01)
          {
              PD = 1;
              show_sound2();
              delayfifty(1);
              PD = 1;
              show_sound(NUM[1],NUM[2]);
              delayfifty(1);
              PD = 1;
              send_data_go();
          }
          else if(NUM[1]==0x20 & NUM[2]==0x03)
          {
              PD = 1;
              show_sound2();
              delayfifty(1);
              PD = 1;
              show_sound(NUM[1],NUM[2]);
              delayfifty(1);
              PD = 1;
              send_data_go();
          }
          else if(NUM[1]==0x20 & NUM[2]==0x04)
          {
              PD = 1;
              show_sound2();
              delayfifty(1);
```

```
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
      }
      else if(NUM[1]==0x20 & NUM[2]==0x05)
      {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
      }
      else if(NUM[1]==0x20 & NUM[2]==0x06)
      {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
      }
}
else if(MEM[1]==0x05 & MEM[2]==0x17)
{
      if(NUM[1]==0x20 & NUM[2]==0x01)
      {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
      }
      else if(NUM[1]==0x20 & NUM[2]==0x03)
      {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
      }
      else if(NUM[1]==0x20 & NUM[2]==0x04)
      {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
```

```
            PD = 1;
            send_data_go();
        }
        else if(NUM[1]==0x20 & NUM[2]==0x05)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
        }
        else if(NUM[1]==0x20 & NUM[2]==0x06)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
        }
    }
    else if(MEM[1]==0x10 & MEM[2]==0x13)
    {
        if(NUM[1]==0x20 & NUM[2]==0x02)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
        }
        else if(NUM[1]==0x20 & NUM[2]==0x03)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
        }
        else if(NUM[1]==0x20 & NUM[2]==0x04)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_go();
        }
```

```
            else if(NUM[1]==0x20 & NUM[2]==0x05)
            {
                  PD = 1;
                  show_sound2();
                  delayfifty(1);
                  PD = 1;
                  show_sound(NUM[1],NUM[2]);
                  delayfifty(1);
                  PD = 1;

                  send_data_go();
            }
            else if(NUM[1]==0x20 & NUM[2]==0x07)
            {
                  PD = 1;
                  show_sound2();
                  delayfifty(1);
                  PD = 1;
                  show_sound(NUM[1],NUM[2]);
                  delayfifty(1);
                  PD = 1;
                  send_data_go();
            }
      }
}
void check_data_back(void)
{
    if(MEM[1]==0x01 & MEM[2]==0x43)
      {
            if(NUM[1]==0x30 & NUM[2]==0x02)
            {
                  PD = 1;
                  show_sound2();
                  delayfifty(1);
                  PD = 1;
                  show_sound(NUM[1],NUM[2]);
                  delayfifty(1);
                  PD = 1;
                  send_data_back();
            }
            else if(NUM[1]==0x30 & NUM[2]==0x03)
            {
                  PD = 1;
                  show_sound2();
                  delayfifty(1);
                  PD = 1;
                  show_sound(NUM[1],NUM[2]);
                  delayfifty(1);
                  PD = 1;
                  send_data_back();
            }
            else if(NUM[1]==0x30 & NUM[2]==0x04)
            {
                  PD = 1;
                  show_sound2();
                  delayfifty(1);
                  PD = 1;
                  show_sound(NUM[1],NUM[2]);
                  delayfifty(1);
                  PD = 1;
                  send_data_back();
```

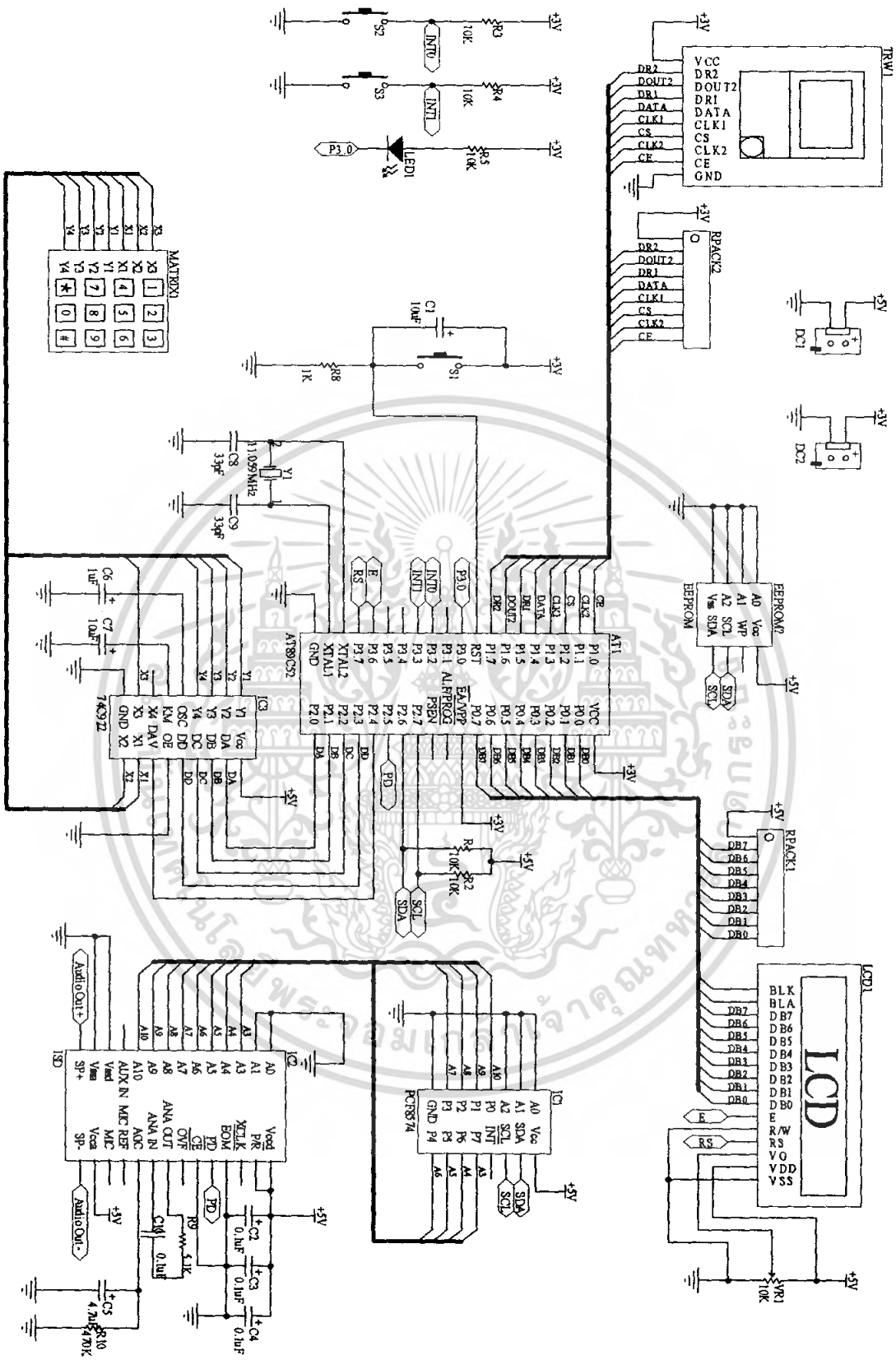
```
    }
    else if(NUM[1]==0x30 & NUM[2]==0x05)
    {
         PD = 1;
         show_sound2();
         delayfifty(1);
         PD = 1;
         show_sound(NUM[1],NUM[2]);
         delayfifty(1);
         PD = 1;
        send_data_back();
    }
    else if(NUM[1]==0x30 & NUM[2]==0x07)
    {
         PD = 1;
         show_sound2();
         delayfifty(1);
         PD = 1;
         show_sound(NUM[1],NUM[2]);
         delayfifty(1);
         PD = 1;
         send_data_back();
    }
}
else if(MEM[1]==0x05 & MEM[2]==0x17)
{
    if(NUM[1]==0x30 & NUM[2]==0x02)
    {
         PD = 1;
         show_sound2();
         delayfifty(1);
         PD = 1;
         show_sound(NUM[1],NUM[2]);
         delayfifty(1);
         PD = 1;
         send_data_back();
    }
    else if(NUM[1]==0x30 & NUM[2]==0x03)
    {
         PD = 1;
         show_sound2();
         delayfifty(1);
         PD = 1;
         show_sound(NUM[1],NUM[2]);
         delayfifty(1);
         PD = 1;
         send_data_back();
    }
    else if(NUM[1]==0x30 & NUM[2]==0x04)
    {
         PD = 1;
         show_sound2();
         delayfifty(1);
         PD = 1;
         show_sound(NUM[1],NUM[2]);
         delayfifty(1);
         PD = 1;
         send_data_back();
    }
    else if(NUM[1]==0x30 & NUM[2]==0x05)
    {
```

```
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_back();
        }
        else if(NUM[1]==0x30 & NUM[2]==0x07)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
           send_data_back();
        }
    }
    else if(MEM[1]==0x10 & MEM[2]==0x13)
    {
        if(NUM[1]==0x30 & NUM[2]==0x01)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_back();
        }
        else if(NUM[1]==0x30 & NUM[2]==0x03)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_back();
        }
        else if(NUM[1]==0x30 & NUM[2]==0x04)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
            PD = 1;
            show_sound(NUM[1],NUM[2]);
            delayfifty(1);
            PD = 1;
            send_data_back();
        }
        else if(NUM[1]==0x30 & NUM[2]==0x05)
        {
            PD = 1;
            show_sound2();
            delayfifty(1);
```

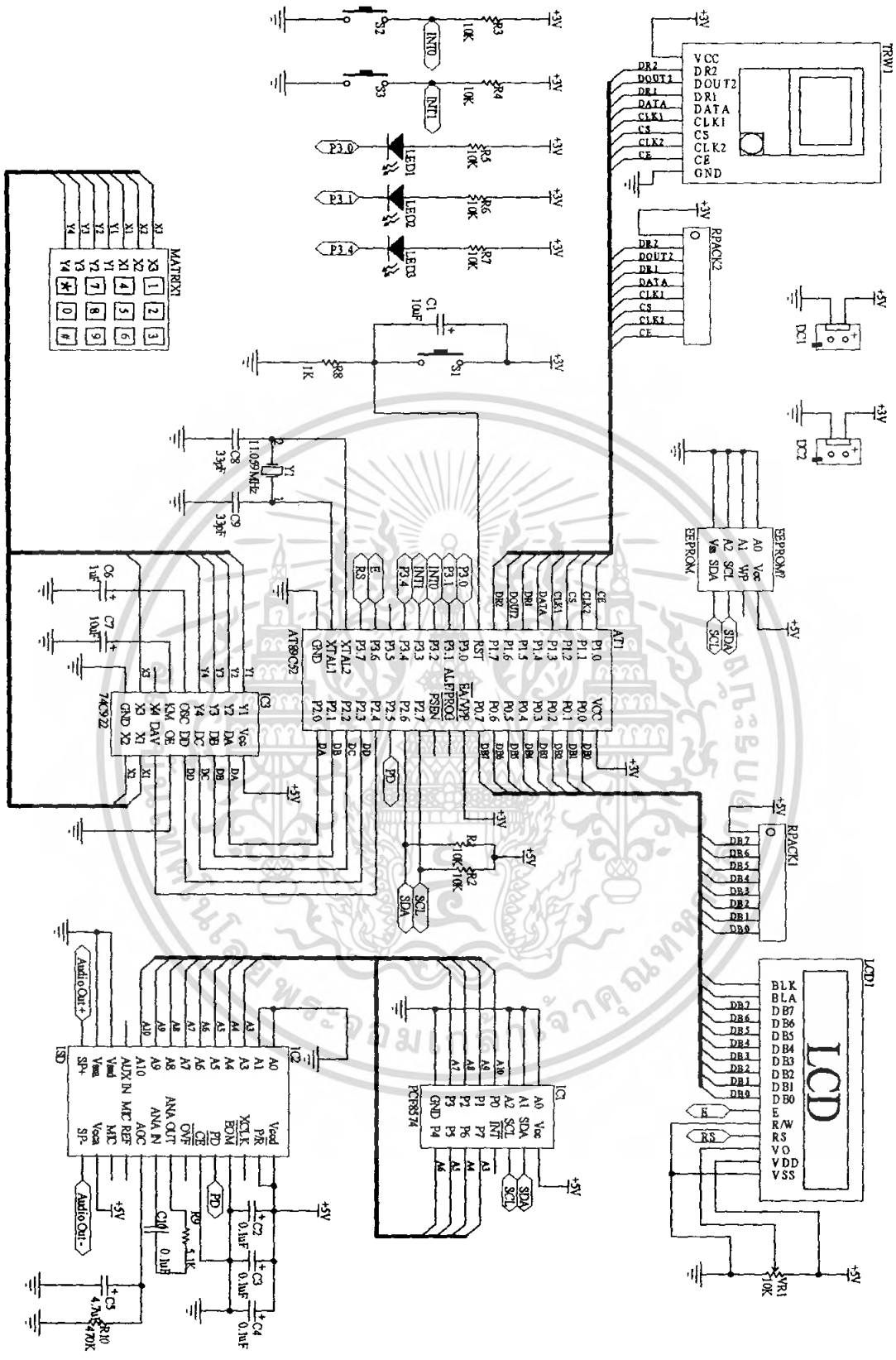
```
                PD = 1;
                show_sound(NUM[1],NUM[2]);
                delayfifty(1);
                PD = 1;
                send_data_back();
            }
            else if(NUM[1]==0x30 & NUM[2]==0x06)
            {
                PD = 1;
                show_sound2();
                delayfifty(1);
                PD = 1;
                show_sound(NUM[1],NUM[2]);
                delayfifty(1);
                PD = 1;
                send_data_back();
            }
        }
}
/*******************************************************/
/*                Function receive_num                 */
/*******************************************************/
void receive_go(void)
{
      unsigned char  DATA;
      MEM[1] = Rd_EEPROM(0x02FF);
      MEM[2] = Rd_EEPROM(0x01FF);
      Init_TRW24();
      led_show = 1;
      way_go   = 0;
      way_back = 1;
      while(1)
      {
          SetMode_TRW24(0);
          lcd_clear();
          lcd_show("NOW RECEIVE DATA",0);
        while (DR1 == 0);
          DATA   = Read_CH1();
          NUM[0] = DATA;

        if(NUM[0] == 0xAA)
        {
              while(DR1 == 0);
                  DATA   = Read_CH1();
                  NUM[1] = DATA;
              while (DR1 == 0);
                  DATA   = Read_CH1();
                  NUM[2] = DATA;
                  if(NUM[1]!=data_in1 | NUM[2]!=data_in2)
                  {
                       check_data_go();
                  }
        }
      }
}
void receive_back(void)
{
      unsigned char DATA;
      MEM[1] = Rd_EEPROM(0x02FF);
      MEM[2] = Rd_EEPROM(0x01FF);
      Init_TRW24();
```

```
        led_show = 1;
        way_go   = 1;
        way_back = 0;
        while(1)
        {
               SetMode_TRW24(0);
               lcd_clear();
               lcd_show("NOW RECEIVE DATA",0);
             while (DR1 == 0);
                 DATA   = Read_CH1();
                 NUM[0] = DATA;
            if(NUM[0] == 0xAA)
            {
                   while(DR1 == 0);
                       DATA   = Read_CH1();
                       NUM[1] = DATA;
                   while (DR1 == 0);
                       DATA   = Read_CH1();
                       NUM[2] = DATA;
                       if(NUM[1]!=data_in1 | NUM[2]!=data_in2)
                       {
                            check_data_back();
                       }
                 }
      }
}
/*****************************************************/
/*                  Function Set_num                 */
/*****************************************************/
void set_num(void)
{
        unsigned char keya,keyb,keyc,keyd;
        show_1();
        keya = scankey();
        key2lcd(keya);
        lcd_blink();
        keyb = scankey();
        key2lcd(keyb);
        lcd_blink();
        keyc = scankey();
        key2lcd(keyc);
        lcd_blink();
        keyd = scankey();
        key2lcd(keyd);
        MEM[1] = buffer1(keya,keyb);
        MEM[2] = buffer2(keyc,keyd);
        Wr_EEPROM(0x02FF,MEM[1]);
        Wr_EEPROM(0x01FF,MEM[2]);
        delayfifty(100);
}
/*****************************************************/
/*                 Function check_num                */
/*****************************************************/
void check_num(void)
{
      MEM[1] = Rd_EEPROM(0x02FF);
      MEM[2] = Rd_EEPROM(0x01FF);
      lcd_init();
      show_2(MEM[1],MEM[2]);
      lcd_delay(1000);
}
```

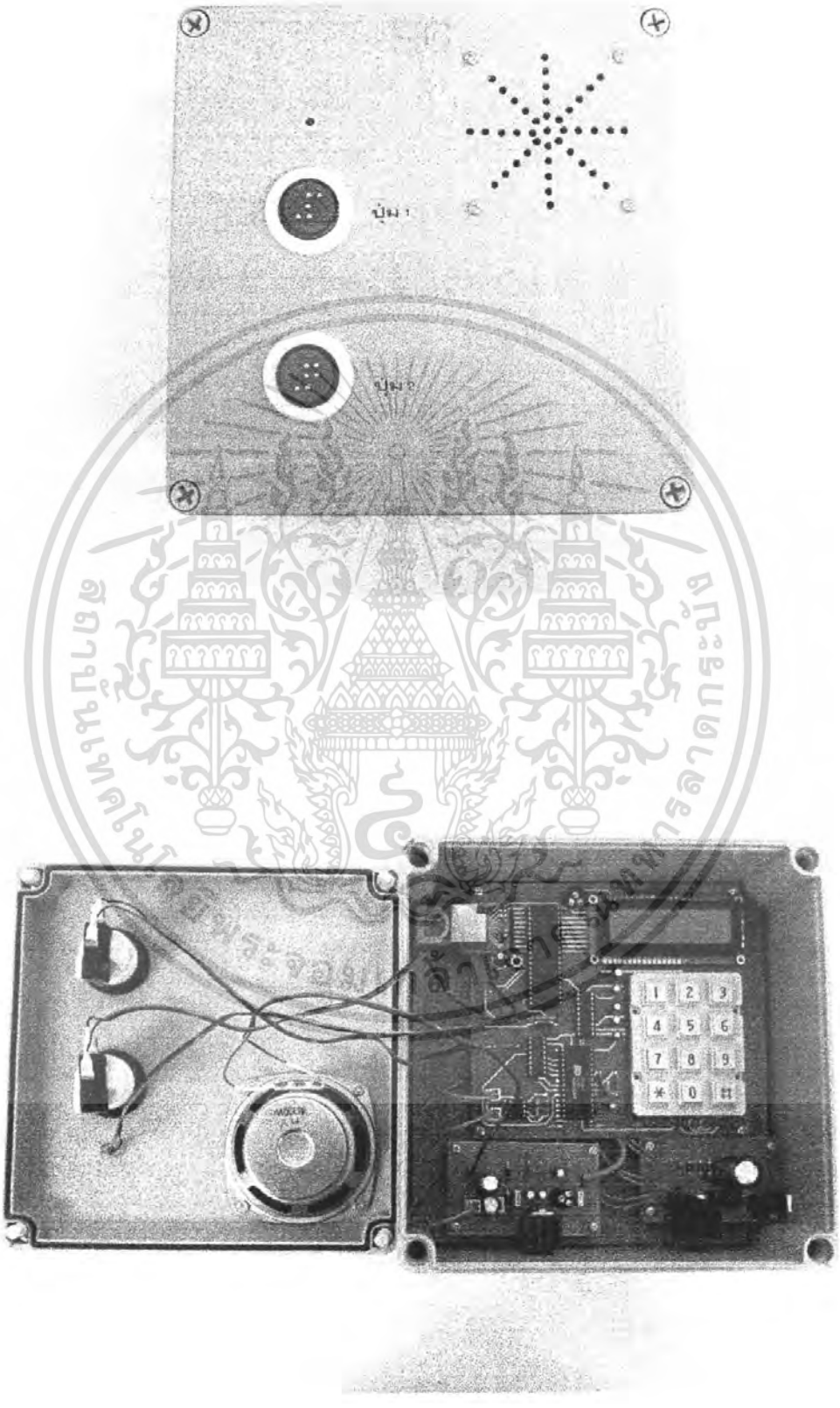
```
/*******************************************************/
/*                Function Interrupt                   */
/*******************************************************/
void service_0(void) interrupt 0
{
      receive_go();
}
void service_1(void) interrupt 2
{
    receive_back();
}
/*******************************************************/
/*                  Mian Program                       */
/*******************************************************/
void main (void)
{
     unsigned char key;
     PD       = 1;
     IT0      = 1;
     IT1      = 1;
     EA       = 1;
     EX0      = 1;
     EX1      = 1;
     way_go   = 1;
     way_back = 1;
     led_show = 1;
     lcd_init();
     lcd_show("     WELCOME",0);
     delayfifty(40);
     while(1)
     {
          lcd_clear();
          lcd_show("SET NUMBER BUS",0);
          lcd_show("PRESS 1 ",1);
          delayfifty(100);
          lcd_clear();
          lcd_show("CHECK NUMBER BUS",0);
          lcd_show("PRESS 2 ",1);
          delayfifty(100);
          lcd_clear();
          lcd_show("CHOOSE NUMBER ",0);
          lcd_blink();
          key = scankey();
       if(key == 0x01)
       {
          key2lcd(key);
          delayfifty(30);
          set_num();
       }
       else if(key == 0x02)
       {
          key2lcd(key);
          delayfifty(30);
          check_num();
       }
       else delayfifty(10);
    }
}
```



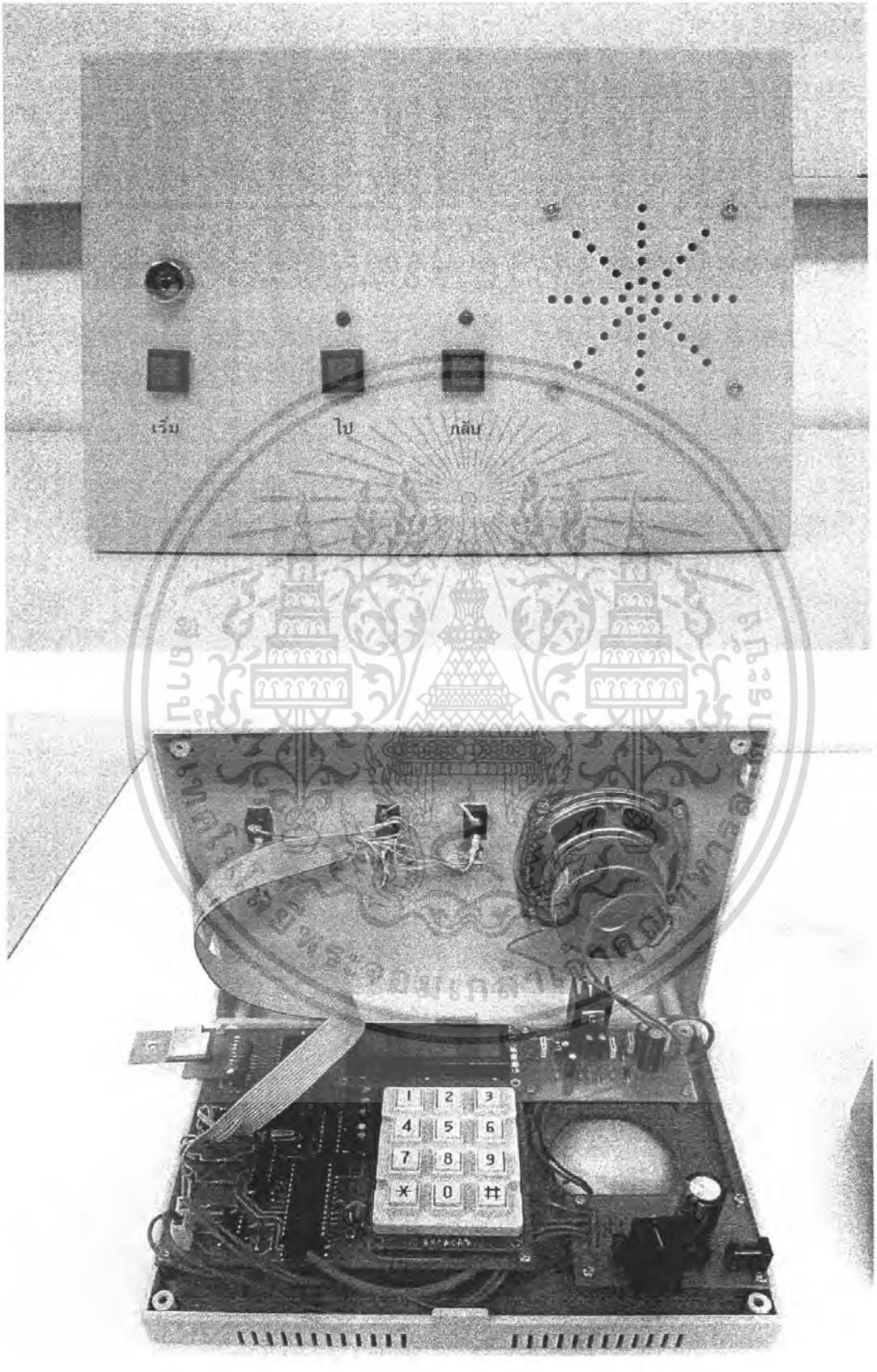
รูปแสดงวงจรรวมที่ป้ายรถประจำทาง



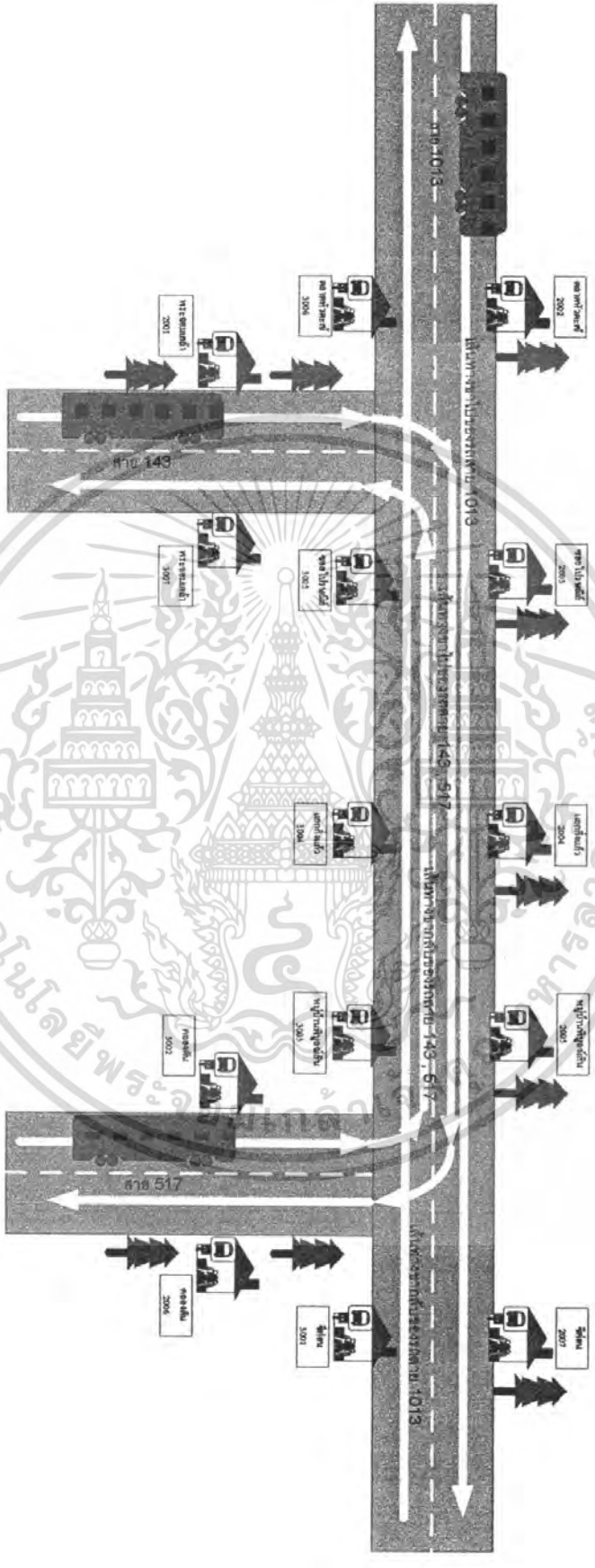
รูปแสดงวงจรรวมที่รถประจำทาง



รูปแสดงอุปกรณ์ที่ป้ายรถประจำทาง



รูปแสดงอุปกรณ์ที่รถประจำทาง



รูปแผนที่จำลองเส้นทางการเดินรถ