ระบบจัดการป้ายรถประจำทาง

BUS STOP MAGNAGEMENT

โดย

นายพีรพันธ์ จอดกิ่ง

นายภูริวัฒน์ ศิริภาพ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ระบบจัดการป้ายรถประจำทาง

BUS STOP MAGNAGEMENT

โดย

นายพีรพันธ์ จอดกิ่ง รหัสประจำตัว 46015186

นายภูริวัฒน์ ศิริภาพ รหัสประจำตัว 46015188

อาจารย์ที่ปรึกษา

อ. ชินภัทร นันทจิวากรชัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ ปีการศึกษา 2548

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบจัดการป้ายรถประจำทาง (BUS STOP MAGNAGEMENT)

ผู้จัดทำ

1. นาย พีรพันธ์ จอดกิ่ง รหัส 46015186
2. นาย ภูริวัฒน์ ศิริภาพ รหัส 46015188

..... อาจารย์ที่ปรึกษา

(อ. ชินภัทร นันทจิวากรชัย)

ปริญญานิพนธ์ ปีการศึกษา 2548

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบจัดการป้ายรถประจำทาง (BUS STOP MAGNAGEMENT)

ผู้จัดทำ

1. นาย พีรพันธ์ จอดกิ่ง รหัส 46015186
2. นาย ภูริวัฒน์ ศิริภาพ รหัส 46015188

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

..... อาจารย์ที่ปรึกษา

(อ. ชินภัทร นันทจิวากรชัย)

ระบบจัดการป้ายรถประจำทาง

นาย พีรพันธ์ จอดกิ่ง รหัส 46015186
นาย ภูริวัฒน์ ศิริกาฬ รหัส 46015188
อ.ชินภัทร นันทจิวากรชัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการออกแบบเครื่องแสดงหมายเลขรถประจำทางล่วงหน้า ก่อนที่รถประจำทางจะเข้ามาจอดที่ป้าย และพร้อมแจ้งให้พนักงานขับรถประจำทางทราบว่ามีผู้โดยสารต้องการจะขึ้นรถในป้ายถัดไปหรือไม่ ซึ่งควบคุมด้วยไมโครคอนโทรลเลอร์ โดยอาศัยเทคนิคการเข้ารหัสข้อมูลที่เป็นรหัสประจำสายของรถประจำทางแต่ละคัน ในรูปแบบของสัญญาณดิจิตอล ทำการมอดดูเลตสัญญาณดิจิตอลแบบเอฟเอสเค (Frequency Shift Keying)

ในการรับส่งข้อมูลระหว่างรถประจำทางกับป้ายแสดงสายรถประจำทาง จะทำให้ผู้โดยสารที่รอรถประจำทางบริเวณป้ายรถประจำทาง ทราบว่าหมายเลขของรถประจำทางที่จะเข้าป้าย และแสดงผลเป็นสัญญาณไฟบอกสถานะที่พนักงานขับรถประจำทาง ว่ามีผู้โดยสารรอขึ้นรถประจำทางคันนี้หรือไม่ เพื่อพนักงานขับรถจะได้ไม่จำเป็นต้องจอดป้ายโดยไม่จำเป็น ซึ่งจะช่วยลดปัญหาจราจรติดขัดได้ระดับหนึ่ง

BUS STOP MANAGEMENT

Mr. Peerapun Jodking ID.46015186

Mr. Puriwat Siripab ID.46015188

Mr. Chinnapat Nantajiwakornchai Advisor

Educational Year 2005

ABSTRACT

This project presents the design of buses line display appliance which shows the number of buses line before they arrive and bus driver teller appliance which shows passenger waiting the bus. The system is controlled by micro controller to encode the bus information in digital format and modulate the signal in FSK (Frequency Shift Keying) form.

They shows type and the number of buses line at bus stop and shows status of passenger wait the bus for bus driver by lamp.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้จะสำเร็จลุล่วงไปได้ด้วยดีไม่ได้เลย ถ้าหากขาดความกรุณาจากคณะอาจารย์ ทุกท่านในภาควิชาอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งอาจารย์ที่ปรึกษาอาจารย์ชินภัทร นันทจิวาภรชัย ที่ให้ความรู้ คำแนะนำ และแนวทางต่างๆ มาโดยตลอด กุณพ่อคุณแม่ ที่คอยห่วงใยมากกว่าใคร ในโลกนี้ จนวันหนึ่งก็เดินทางมาหาจากต่างจังหวัด เพื่อทำอาหารอร่อยๆให้ทาน และเพื่อนๆ ที่ทำร่วมโปรเจคจนลืมวันเกิดตัวเอง และน้องๆในห้องโปรเจคที่น่ารักทุกคน ซื้อของกินมาให้กินเรื่อย

ขอขอบพระคุณทุกๆท่านด้วยความเคารพรัก ไว้ ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันจากผลทางเศรษฐกิจในประเทศของเรา ทำให้มีคนนิยมใช้บริการระบบขนส่งมวลชนกันมากขึ้น ในช่วงเวลาเร่งรีบจึงมีผู้คนมารอขึ้นรถประจำทางเป็นจำนวนมาก ซึ่งอาจจะทำให้ไม่ได้รับความสะดวกในการมองเห็นรถประจำทางที่กำลังเข้าป้ายรถประจำทาง รวมทั้งผู้ที่มีปัญหาทางด้านสายตาอาจจะเห็นหมายเลขของรถประจำทางไม่ชัดเจน และเป็นปัญหาการจราจรที่ติดขัด ซึ่งบางส่วนก็มาจากรถประจำทางเข้าจอดที่ป้ายรถประจำทางมากเกินไป ซึ่งรถประจำทางบางคันไม่มีทั้งผู้โดยสารที่จะขึ้นและลงที่ป้ายรถประจำทางนั้น

1.1 คุณสมบัติของโครงงาน

จากปัญหาดังกล่าวจึงได้เกิดโครงงานนี้ขึ้นมา ซึ่งจะเป็นระบบจัดการเกี่ยวกับการแสดงผลหมายเลขของรถประจำทางก่อนที่จะเข้าป้ายรถประจำทางประมาณ 200 เมตร ซึ่งใช้การสื่อสารของข้อมูลดิจิตอลที่ผ่านการมอดดูเลตสัญญาณแบบ FSK (Frequency Shift Keying Modulation) ใช้คลื่นความถี่วิทยุ (Radio Wave) เป็นตัวกลางในการสื่อสารระหว่างเครื่องรับส่งที่ป้ายรถประจำทางกับเครื่องรับส่งที่อยู่บนรถประจำทาง โดยเครื่องรับส่งที่อยู่บนรถประจำทางจะส่งข้อมูลเพื่อแจ้งให้เครื่องรับส่งที่ป้ายรถประจำทางทราบล่วงหน้าว่ารถประจำทางสายอะไรกำลังจะเข้าไปที่ป้ายรถประจำทาง และเครื่องรับส่งที่ป้ายรถประจำทางจะสามารถส่งข้อมูลแจ้งไปยังเครื่องรับส่งที่อยู่บนรถประจำทาง เพื่อแจ้งให้พนักงานขับรถทราบว่ามีผู้โดยสารกำลังรอรถประจำทางสายที่กำลังจะเข้ามาหรือไม่ ถ้าหากเป็นสายที่มีคนกำลังรออยู่ที่ป้ายรถประจำทาง สัญญาณจากเครื่องรับส่งบนรถประจำทางก็จะแจ้งให้พนักงานขับรถทราบ เพื่อจอดรับผู้โดยสาร แต่ถ้าไม่มีผู้โดยสารรออยู่ ก็จะไม่มีสัญญาณแจ้งให้พนักงานขับรถจอดในป้ายรถประจำทางนั้น

1.2 โครงสร้างของโครงงาน

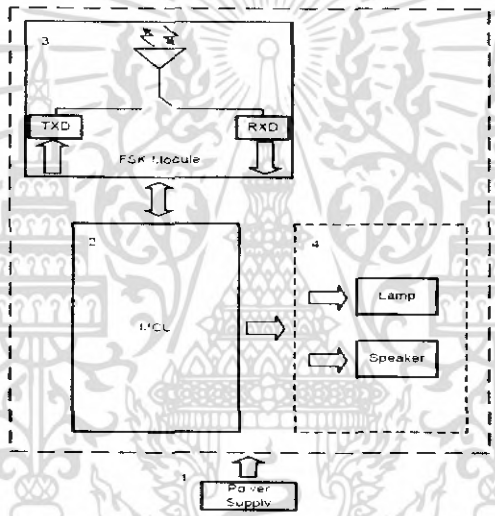
ซึ่งโครงงานนี้จะแบ่งออกเป็น 2 ส่วน ส่วนแรกเป็นส่วนของเครื่องรับส่งที่อยู่บนรถประจำทาง และส่วนที่สองเป็นส่วนเครื่องรับส่งที่ป้ายรถประจำทาง โดยทั้ง 2 ส่วนจะมีส่วนประกอบย่อยแบ่งออกเป็น 4 ส่วน คือ

1. ส่วนจ่ายแรงดันไฟฟ้า คือ วงจรเพาเวอร์ซัพพลาย (Power Supply) เป็นส่วนจ่ายแรงดันไฟฟ้าให้กับวงจรต่างๆ

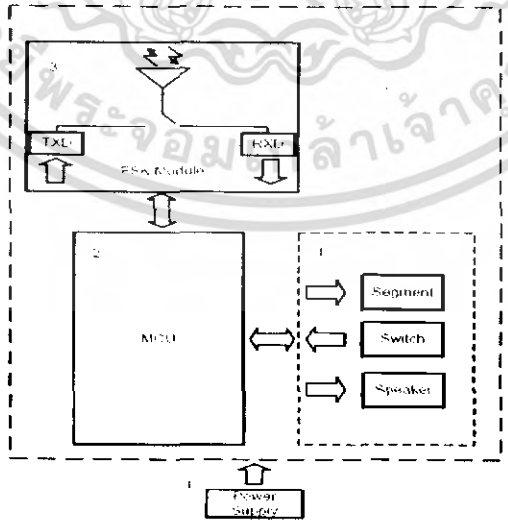
2. ส่วนประมวลผล ได้แก่ วงจรในส่วนของไมโครคอนโทรลเลอร์ ซึ่งทำหน้าที่รับข้อมูลและสัญญาณต่างๆไปประมวลผล และให้ผลออกมาทางส่วนแสดงผลต่อไป

3. ส่วนการรับส่งสัญญาณระหว่างรถประจำทางกับรถประจำทาง ได้แก่ เอฟเอสเคโมดูล (FSK Module) มีหน้าที่นำข้อมูลที่ส่วนประมวลผลส่งมา และทำการแปลงสัญญาณที่รับมา ให้เป็นสัญญาณที่สามารถส่งออกไปยังเครื่องรับได้ ซึ่งในโครงงานนี้จะนำเอาข้อมูลที่มาจากไมโครคอนโทรลเลอร์มามอดูเลตสัญญาณแบบ FSK แล้วส่งออกทางสายอากาศ และมีหน้าที่รับสัญญาณที่ส่งมาจากเครื่องส่งแล้วแปลงเป็นข้อมูล ส่งให้ส่วนประมวลผลต่อไป ซึ่งในโครงงานนี้คือการรับคลื่นวิทยุเข้ามาทางสายอากาศมาดีมอดูเลต (Demodulation) เพื่อให้ได้ข้อมูลเพื่อส่งให้ไมโครคอนโทรลเลอร์นำไปประมวลผลต่อไป

4. ส่วนแสดงผล ได้แก่ หลอดไฟLED จอแสดงผล สวิตช์กดเพื่อเรียกรถประจำทางและลำโพง มีหน้าที่รับสัญญาณจากส่วนประมวลผลมาแสดงในรูปแบบต่างๆที่ได้กำหนดไว้



รูปที่1.1 บล็อกไดอะแกรมของส่วนรถประจำทาง



รูปที่1.2 บล็อกไดอะแกรมของส่วนป้ายรถประจำทาง

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 ความรู้เบื้องต้นเกี่ยวกับการสื่อสารข้อมูล

การสื่อสารข้อมูล เป็นการส่งข่าวสารดิจิตอล (Digital Information) ซึ่งโดยมากอยู่ในรูปของเลขฐานสองจากแหล่งกำเนิดไปยังจุดหมายปลายทาง ข้อมูลจากแหล่งกำเนิด จะอยู่ในลักษณะสัญญาณดิจิตอลและข้อมูลที่รับได้ก็จะอยู่ในลักษณะดิจิตอลเช่นเดียวกันถึงแม้ว่าข้อมูลจะสามารถส่งได้ในลักษณะสัญญาณอนาลอกหรือสัญญาณดิจิตอลก็ตาม ข่าวสารจากแหล่งกำเนิด อาจจะเป็นรหัสของตัวอักษร ตัวเลขหรือเครื่องหมายที่อยู่ในรูปของเลขฐานสอง เช่น รหัส ASCII หรือรหัส EBCDIC รหัสของไมโครโปรเซสเซอร์ (Microprocessor OP-Code) รหัสควบคุม รหัสที่อยู่ของผู้ใช้ (Use Address) โปรแกรมคอมพิวเตอร์ ข่าวสารฐานข้อมูล (Data Base Information)

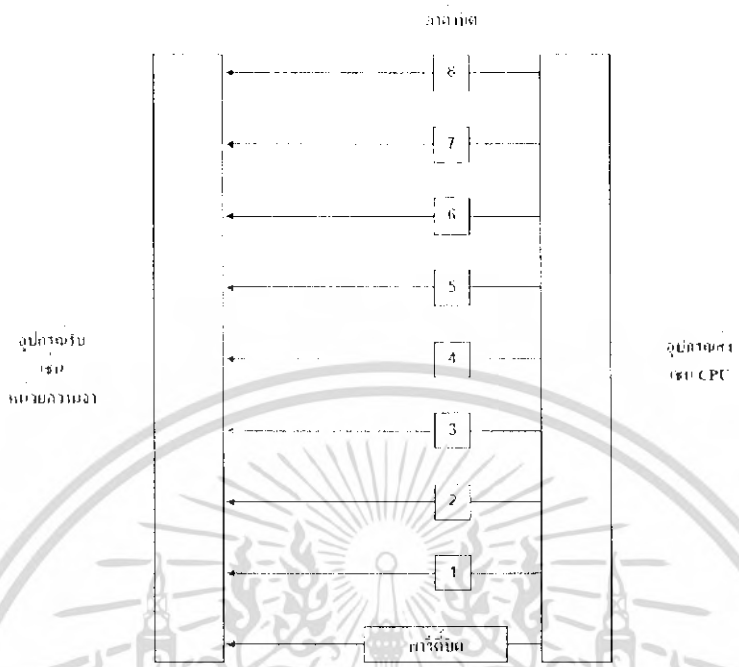
โครงข่ายการสื่อสารข้อมูล (Data Communication Network) แบบง่ายๆ เช่น การเชื่อมโยงคอมพิวเตอร์ส่วนบุคคล หรือไมโครคอมพิวเตอร์สองเครื่องในระยะทางใกล้ๆ ด้วยสายธรรมดาเข้าด้วยกันโดยตรง แต่ถ้าเป็นระยะทางไกลอาจจะต้องใช้สายโทรศัพท์แทน หรือถ้าจะให้เป็นโครงข่ายการสื่อสารข้อมูลที่ยุ่งยาก อาจประกอบด้วยคอมพิวเตอร์เมนเฟรม (Mainframe Computer) หนึ่งเครื่องหรือมากกว่ากับอุปกรณ์ปลายทางข้อมูลเป็นจำนวนร้อยๆเครื่อง

2.1.1 การถ่ายโอนข้อมูลแบบขนาน

การถ่ายโอนข้อมูลแบบขนาน ลักษณะการส่งข้อมูลแบบขนาน ทำได้โดยการส่งข้อมูลออกมาทีละ 1 ไบต์จากอุปกรณ์ส่งไปยังอุปกรณ์รับ ตัวกลางระหว่าง 2 เครื่องจะต้องมีช่องทางให้ข้อมูลเดินทางอย่างน้อย 8 ช่องทาง โดยมากจะเป็นสายขนานให้กระแสไฟฟ้าวิ่งมากกว่าจะเป็นตัวกลางชนิดอื่น เนื่องจากมีการลดทอนของสัญญาณเนื่องจากความต้านทานของสาย ระยะทางระหว่าง 2 เครื่องจำไม่ควรเกิน 100 ฟุต ปัญหาที่เกิดขึ้นหากระยะทางของสายมากกว่านี้ คือ ระดับของกราวด์ในทางไฟฟ้าที่จุดรับผิดไปจากจุดส่งทำให้เกิดการผิดพลาดในการรับสัญญาณลอจิกทางฝ่ายรับ

นอกจากสายที่เป็นทางเดินของข้อมูลแล้ว อาจจะมีทางเดินของสัญญาณอื่นๆอีก เป็นต้นว่า บิตที่บอก Parity ของสัญญาณ เพื่อเป็นการตรวจสอบความผิดพลาดของการรับสัญญาณที่ปลายทาง หรือ สายที่ควบคุมการโต้ตอบ (Hand Shake)

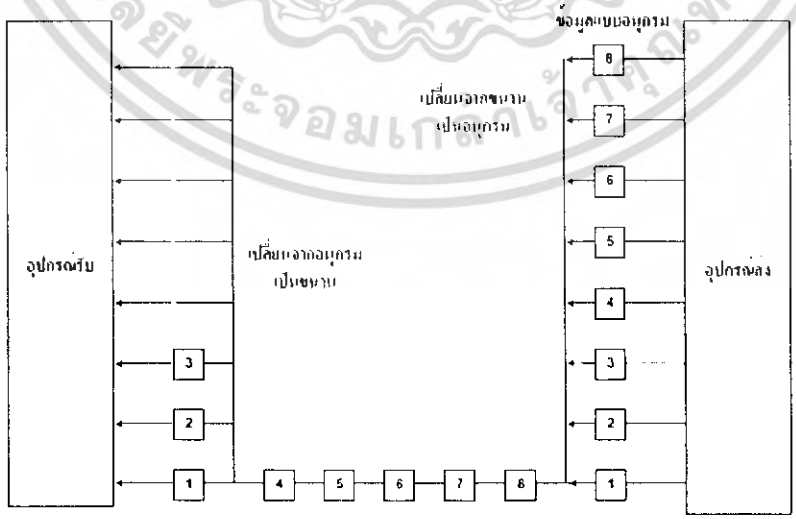
จะเห็นว่าการส่งแบบขนานส่วนมากจะทำในระยะใกล้ๆ เนื่องจากจะต้องมีช่องทางเดินของสัญญาณมากกว่า 8 สาย และอุปกรณ์ที่ติดต่อแบบขนานกับคอมพิวเตอร์ที่เห็นจะได้แก่ เครื่องพิมพ์ เป็นต้น



รูปที่ 2.1 แสดงการส่งข้อมูลแบบขนาน

2.1.2 การถ่ายโอนข้อมูลแบบอนุกรม

การถ่ายโอนข้อมูลแบบอนุกรม ในการถ่ายโอนข้อมูลแบบอนุกรม ข้อมูลถูกส่งออกมาที่ละบิตระหว่างจุดส่งและจุดรับ จะเห็นว่าการส่งข้อมูลแบบนี้จะช้ากว่าแบบขนานที่กล่าวมาแล้วอย่างแน่นอน แต่สายเพียงคู่เดียว ทำให้ค่าใช่จ่ายในเรื่องของสื่อกลางถูกกว่าแบบขนาน สำหรับการส่งระยะทางไกลๆ โดยเฉพาะเมื่อมีระบบสื่อสารทางโทรศัพท์ไว้ใช้งานอย่างแล้วย่อมจะเป็นการสะดวกที่จะนำมาใช้ถ่ายโอนข้อมูลแบบอนุกรมได้



รูปที่ 2.2 แสดงการส่งข้อมูลแบบอนุกรม

จากรูปที่ 2.2 แสดงให้เห็นการส่งข้อมูลแบบอนุกรม ข้อมูลจากจุดส่งจะถูกเปลี่ยนให้เป็นอนุกรมเสียก่อนแล้วค่อยทยอยส่งออกทีละบิตไปยังจุดรับ ณ ที่จุดรับจะต้องมีกลไกในการเปลี่ยนข้อมูลที่ส่งมาที่ละบิต ให้เป็นสัญญาณแบบขนานซึ่งลงตัวพอดีนั่นคือ บิต 1 ลงที่ Data Bus เส้นที่ 1 พอดีการที่จะทำให้การแปลงสัญญาณจากอนุกรมทีละบิต ให้ลงพอดีนั้นจำเป็นที่จะต้องมีกลไกที่เหมาะสม เพื่อป้องกันการผิดพลาดในการรับ กลไกที่ว่านี้แบ่งออกเป็น 2 แบบ คือ

2.1.3 การส่งผ่านข้อมูลแบบซิงค์โครนัส (Synchronous Transmission)

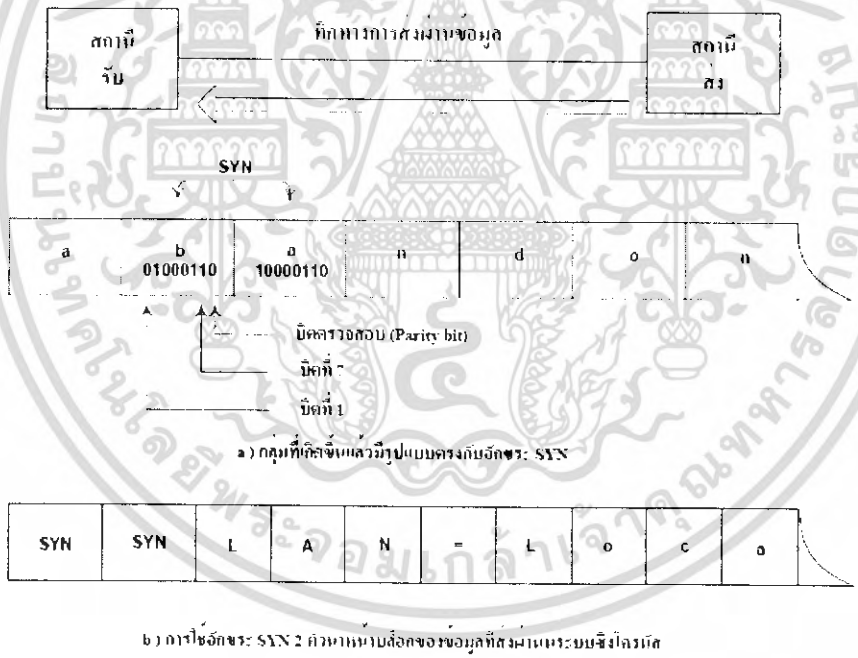
ในการส่งแบบซิงค์โครนัส ข้อมูลจะถูกจัดเป็นกลุ่มๆ และทำการส่งข้อมูลทั้งกลุ่มไปพร้อมกันในทีเดียว เรียกกลุ่มของข้อมูลนี้ว่า "Block of Data" และในการส่งผ่านข้อมูลแบบซิงค์โครนัสนี้ ช่วงเวลาของแต่ละบิตที่ทำการส่งจะใช้เวลาเดียวกัน และในการส่งผ่านทั้งตัวอักษร ตัวอักษรตัวแรกและตัดถัดไปจะไม่มีอะไรมาคั่น (ภายในบล็อกเดียวกัน) ดังนั้นช่วงเวลาระหว่างบิตสุดท้ายของตัวส่งผ่านจะต้องคิดในรูปแบบของระบบส่งผ่านข้อมูลนี้มีแนวโน้มที่จะยึดหลักการส่งผ่านข้อมูลที่อยู่ในรูปของรหัส 1, 0 หรือเลขฐานสองนั่นเอง โดยที่ข้อความที่ต้องการส่งนั้นจะอยู่ในรูปของบิตที่แน่นอน

รูปที่ 2.3 แสดงการต่อเนื่องของข้อความ ที่ถูกส่งผ่านแบบซิงค์โครนัส

จากรูปที่ 2.3 แสดงให้เห็นการเอากลุ่มตัวอักษร (ข้อความที่ต้องการส่งผ่าน) มาเรียงต่อกันเพื่อเตรียมการส่งผ่านข้อมูลแบบซิงค์โครนัส จะเห็นได้ว่าตัวอักษรที่นำมาต่อกันเรียงชิดกันโดยที่ไม่มีช่องว่างของตัวอักษรเลย หรือตามที่กล่าวมาแล้วว่าช่วงเวลาระหว่างบิตสุดท้ายของตัวอักษรตัวแรกกับบิตแรกของตัวอักษรตัวถัดไปเท่ากับศูนย์นั่นเอง ในเมื่อรูปแบบการส่งผ่านข้อมูลเป็นเช่นนี้ อุปกรณ์รับข้อมูลจะต้องทราบอะไรบ้าง จึงจะระบุลงไปส่วนนั้นๆ เป็นกลุ่มบิตของตัวอักษรที่ถูกเข้ารหัสมา สิ่งที่อุปกรณ์ด้านรับจะต้องทราบก็คือ บิตใดเป็นบิตแรกของตัวอักษรตัวแรก ขนาดตัวอักษร (จำนวนบิตที่ใช้แทนในหนึ่งตัวอักษร) และอีกประการหนึ่งก็คือความเร็วในการส่งผ่านข้อมูล อุปกรณ์รับข้อมูลจึงจะจัดกลุ่มของบิตออกเป็นกลุ่มๆ เพื่อแทนค่ากลับเป็นตัวอักษรต่างๆ ที่รับเข้ามา อย่างเช่นกรณีที่ข้อมูลที่ส่งผ่านมาอยู่ในรูปของรหัสแอสกี้ (ASCII) ตัวอักษรแต่ละตัวจะ

ถูกเข้ารหัสในรูป 8 บิต แทนหนึ่งตัวอักษร โดยมีบิตแรกเป็นบิตตรวจสอบ ดังนั้น อุปกรณ์รับข้อมูลจะตัดบิตออกเป็นกลุ่มละ 8 บิต เพื่อนำมาตีความเป็นตัวอักษรแต่ละตัวนั่นเอง

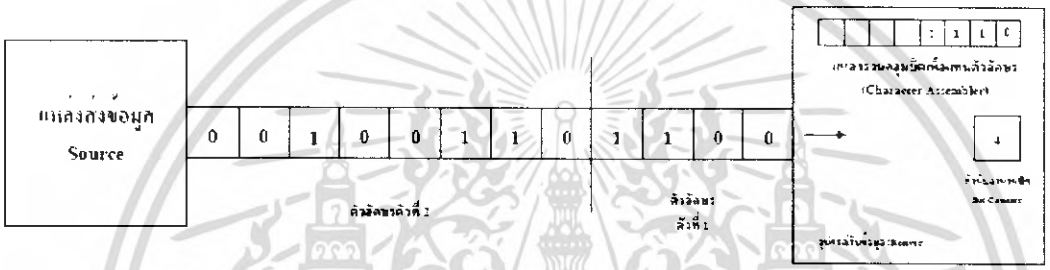
สำหรับวิธีการที่จะระบุไปได้ว่า บิตใดเป็นบิตแรกของตัวอักษรตัวแรกนั้น มีวิธีการดังนี้ ตามที่ได้กล่าวมาแล้วว่าข้อมูลที่ถูกส่งผ่านโดยวิธีการซิงค์โครนัสจะถูกจับมารวมกันเป็นกลุ่มของข้อมูล (Block of data) และที่ส่วนต้นของบล็อก จะใส่ตัวอักขระซิงค์ (SYN Character) ซึ่งเป็นอักขระพิเศษที่ใช้ในการควบคุมการส่งผ่าน ข้อมูลโดยที่อักขระซิงค์มีรูปแบบของบิตคือ 00010110 มีบิตตรวจสอบแบบเลขคี่ : (Odd Parity) แล้วอุปกรณ์รับข้อมูลจะคอยตรวจดูขบวนบิตที่เข้ามาว่ามีส่วนใดตรงกับอักขระ SYN บ้าง เมื่อพบกับอักขระ SYN แล้วอุปกรณ์รับข้อมูลจะทราบได้ทันทีว่าถึงจุดเริ่มต้นที่จะตัดกลุ่มของบิตกลุ่มละ 8 บิต เพื่อแทนตัวอักษรได้และตัวอักษรหลายๆตัวที่ตีความได้ก็คือ ข้อความที่ส่งมาในแต่ละบล็อก แต่การใช้ตัวอักขระ SYN เพียงตัวเดียวใส่ไว้ที่ส่วนต้นของบล็อกยังเป็นวิธีการที่ไม่ถูกต้อง เพราะในบางกรณีขบวนของบิตที่แทนตัวอักษรมีบางช่วงที่ไปตรงกับรูปแบบของบิตอักขระ SYN ได้



รูปที่ 2.4 แสดงรูปแบบการใช้อักขระ SYN นำหน้ากลุ่มตัวอักษร

จากรูปที่ 2.4 จะเห็นได้ว่าถ้าส่งข้อความที่มีตัวอักษร b และ a ติดกัน 4 บิตท้ายของตัวอักษร b ต่อกัน 4 บิตแรกของตัวอักษร a ตรงกับอักขระ SYN พอดีจะทำให้อุปกรณ์รับข้อมูลดีความผิดได้ ดังนั้นวิธีการแก้ไขข้อผิดพลาดที่จะเกิดได้จากกรณีเช่นนี้ โดยการใช้อักขระ SYN 2 ตัวใส่ไว้ที่ที่ตรวจพบอักขระ SYN จะดูอีก 8 บิตถัดไปว่าเป็นอักขระ SYN ต่อไป หรือกล่าวได้ว่าเครื่องจะปรับตัวเข้าสู่ขบวนการจัดกลุ่มบิตกลุ่มละ 8 บิตแทนตัวอักษร หรือข้อมูลที่ได้รับ

รูปที่ 2.5 แสดงการเปรียบเทียบอุปกรณ์รับข้อมูลตรวจหาอักขระในระบบซิงค์โครนัส



รูปที่ 2.6 แสดงการตัดแถวของบิตออก กลุ่มละ 8 บิต เพื่อแทนตัวอักษรของอุปกรณ์รับข้อมูลในระบบการส่งผ่านข้อมูลแบบซิงค์โครนัส

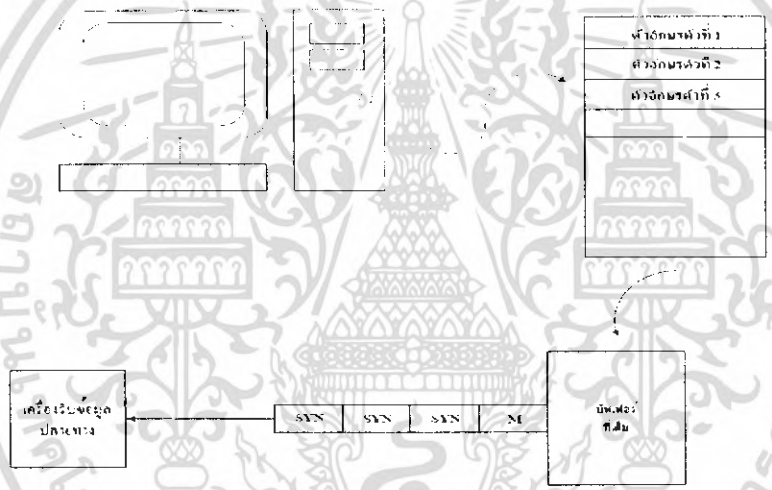
จากรูป 2.5 และ 2.6 คือขบวนการรับข้อมูลที่ส่งผ่านมาในระบบซิงค์โครนัสในบางระบบการใช้อักขระ SYN นำหน้ากลุ่มข้อมูลอาจใช้อักขระ SYN ถึง 3 – 4 ตัวก็ได้ เพื่อความแน่นอนในการส่งผ่านข้อมูลแบบซิงค์โครนัสที่สมบูรณ์แบบยิ่งขึ้น

ถ้าจะกล่าวถึงการส่งผ่านข้อมูลแบบซิงค์โครนัสจะถูกเรียกว่า แฟล็ก (Flag) ความหมายก็คือ มีสัญญาณหรือไม่มี เช่นเดียวกับการใช้สัญญาณการโบกธง แต่ในทุกกรณีจะเห็นได้ว่า หลักของการส่งสัญญาณแบบ ซิงค์โครนัสนั้น จะต้องใช้เครื่องรับข้อมูลที่ปลายทางเข้าใจถึงลักษณะของบิตพิเศษที่ส่งมาเพื่อให้รู้ว่าคือ จุดเริ่มต้นของกลุ่มตัวอักษร (Block) ที่กำลังส่งเรียงกันเข้ามา

บัฟเฟอร์เทอร์มินอล (Buffer Terminal) เมื่อเราทำการส่งตัวอักษรหลายๆ ตัวโดยระบบจัดกลุ่มเข้าเป็น บล็อกตามระบบของการซิงค์โครนัส ดังนั้น สภาวะของการส่งข้อมูลจึงเป็นเรื่องที่ต้องคำนึงถึงเป็นอย่างมาก เราทราบแล้วว่า ผู้ป้อนข้อมูลจะต้องพิมพ์ข้อมูลเข้าทางแป้นพิมพ์ จากนั้นข้อมูลจึงถูกส่งไปตามสายส่งเพื่อไปยังสถานีปลายทางด้วยเหตุนี้จึงมีข้อที่ควรพิจารณา 2 ข้อ คือ

1. ผู้พิมพ์ข้อมูลป้อนเข้าเครื่องไม่สามารถรักษาระดับความห่างระหว่างตัวอักษรได้
2. ความเร็วในการส่งข้อมูลตามสายส่ง จะมีความเร็วสูงกว่าที่ผู้ป้อนข้อมูลจะพิมพ์เข้าเครื่องมาก

ด้วยเหตุผล 2 ข้อนี้ จึงต้องสร้างบัฟเฟอร์เทอร์มินอลขึ้น ซึ่งหมายถึง เครื่องเทอร์มินอลจะต้องมีหน่วยความจำ เพื่อเป็นที่รวบรวมอักขระขึ้น ดังรูปที่ 2.7 ซึ่งแสดงให้เห็นถึงเทอร์มินอลที่มีหน่วยความจำอยู่ด้วย หน่วยความจำประเภทนี้จะมีชื่อเรียกพิเศษว่า บัฟเฟอร์ (Buffer) เมื่อเป็นเช่นนี้ ผู้ป้อนข้อมูลก็จะพิมพ์ตัวอักขระเข้าเครื่องเทอร์มินอลได้ตามสบาย เมื่อพิมพ์ข้อความที่ต้องการส่งเสร็จแล้วจึงกดปุ่มบังคับการส่งข้อมูล ข้อมูลทั้งหมดก็จะถูกบรรจุเข้าสู่สายส่งไปยังสถานีปลายทางโดยทันที หรือในกรณีเมื่อบัฟเฟอร์เต็มก็จะเกิดการส่งโดยอัตโนมัติเสียทีหนึ่งก่อน แล้วล้างบัฟเฟอร์เพื่อรอรับข้อมูลใหม่ต่อไป การสร้างระบบบัฟเฟอร์เทอร์มินอลนี้ จะต้องมีการออกแบบทั้งตัวเครื่อง (Hardware) และตัวโปรแกรมควบคุมระบบการส่ง (Software) เช่น เป็นตัวสร้างอักขระ SYN หน้ากลุ่มข้อมูลว่าจะใช้อักขระ SYN กี่ตัวก่อนที่จะส่งลงไปยังสายส่งสัญญาณและที่ปลายทางของสายส่ง (เครื่องรับข้อมูล) ก็ยังต้องมีบัฟเฟอร์ช่วยในการรับข้อมูลที่มาตามสายส่ง



รูปที่ 2.7 แสดงการใช้บัฟเฟอร์ช่วยในระบบการส่งสัญญาณ แบบซิงค์โครนัส

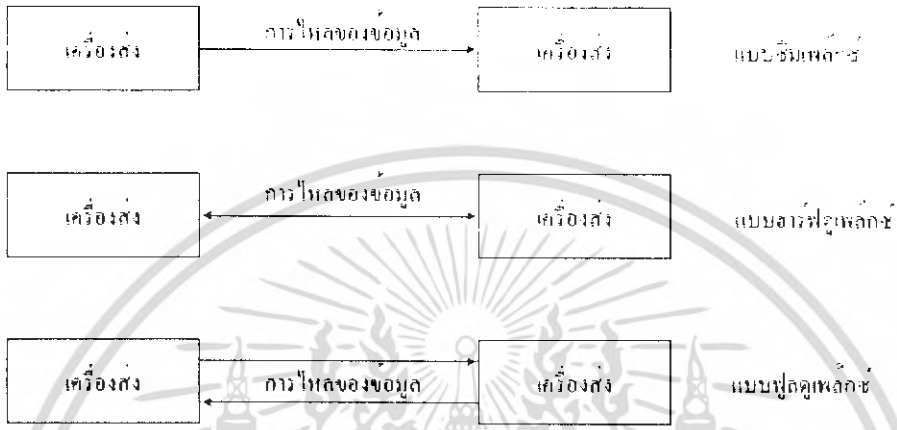
2.1.4 การส่งผ่านข้อมูลแบบอะซิงค์โครนัส (Asynchronous Transmission)

จะมีรูปแบบของการสื่อสารข้อมูลแบบอนุกรม การติดต่อแบบอนุกรมอาจจะแบ่งตามรูปแบบได้ 3 รูปแบบ ได้แก่

แบบซิมเพล็กซ์ (Simplex) ข้อมูลที่ส่งได้ในทางเดียวเท่านั้น บางครั้งก็เรียกว่าการส่งทิศทางเดียว (Unidirectional Data Bus) ในการสื่อสารแบบนี้อุปกรณ์สื่อสารด้านหนึ่ง จะส่งข้อมูลไปในช่องสัญญาณได้เท่านั้น แต่จะรับข้อมูลจากช่องสัญญาณไม่ได้ แต่อุปกรณ์สื่อสารอีกด้านหนึ่งจะรับข้อมูลจากช่องสัญญาณได้เท่านั้น และจะส่งข้อมูลไปในช่องสัญญาณไม่ได้

แบบฮาร์ฟดูเพล็กซ์ (Half Duplex) หมายถึงการสื่อสารข้อมูลใน 2 ทิศทางแต่ในช่วงเวลาหนึ่งได้เพียงทิศทางเดียวเท่านั้น อุปกรณ์สื่อสารทั้ง 2 ด้านจะผลัดกันรับผลัดกันส่ง การสื่อสารแบบนี้ส่วนใหญ่แล้วจะใช้ระบบสาย 2 เส้น (2 Wire)

แบบฟูลดูเพล็กซ์ (Full Duplex) หมายถึงการสื่อสารข้อมูลใน 2 ทิศทางพร้อมกัน การสื่อสารแบบนี้ใช้ได้ทั้งระบบสาย 2 เส้น (2 Wire) และสาย 4 เส้น (4 Wire) แต่ในระบบสาย 2 เส้น จะต้องอาศัยเทคนิคการแบ่งความถี่เข้าช่วย คือจะส่งในความถี่ช่วงหนึ่งและจะรับในความถี่อีกช่วงหนึ่ง



รูปที่ 2.8 แสดงรูปแบบการสื่อสารข้อมูลแบบอนุกรม

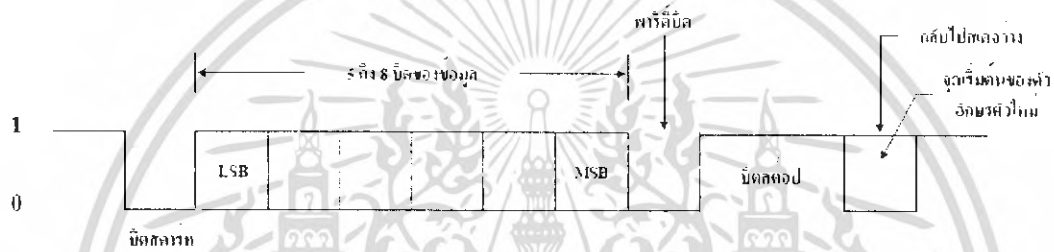
ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม

ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม หน่วยวัดเป็นบิตต่อวินาที (bps) และหน่วยที่บรรยายถึงการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่า บอดเรต (Baud Rate) หรืออัตราบอด ซึ่งแสดงถึงการส่งข้อมูลแบบอนุกรมมากกว่า 1 บิต เขียนในรูปสมการคณิตศาสตร์จะได้

อัตราบิต (Bit rate) = อัตราบอด (Baud rate) บิตใน 1 บอด

การส่งผ่านข้อมูลแบบซิงค์โครนัสนี้ พัฒนาจากการส่งโทรพิมพ์ในสมัยก่อน ลักษณะของสัญญาณแสดงไว้ในรูปที่ 2.9 เพื่อกลไกในการรับส่งอย่างถูกต้อง สัญญาณอะซิงค์โครนัสจะประกอบด้วยบิตเริ่มต้นหรือบิตสตาร์ท (Start Bit) และบิตสิ้นสุดหรือบิตสต็อป (Stop Bit) ขณะที่สถานะของการส่งแบบว่าง (Idle) คือ ยังไม่มีสัญญาณส่งออกมาจะมีสัญญาณหรือมีแรงดัน (หรือกระแส) ตลอดเวลา เพื่อความแน่ใจว่าฝ่ายรับยังติดต่ออยู่กับฝ่ายส่ง เมื่อเริ่มจะส่งข้อมูลสัญญาณของอะซิงค์โครนัสจะเป็น " 0 " หนึ่งช่วงสัญญาณนาฬิกา บิตนี้เรียกว่า สตาร์ทบิตตามหลังของสตาร์ทบิตก็จะเป็นข้อมูลสำหรับ 1 ตัวอักษร ซึ่งอาจจะมีขนาดตั้งแต่ 5 บิตจนถึง 8 บิต โดยบิตที่มีค่าน้อยที่สุด (LSB) จะถูกส่งออกมาก่อนใส่ไปจนบิตที่มีค่ามากที่สุด (MSB) การเข้ารหัสอักขระนี้ส่วนมากจะนิยมให้รหัส ASCII แรกเริ่มที่เดียวในงานของโทรพิมพ์ใช้รหัส Baudot ซึ่งใช้ 5 บิต ในการแทนอักขระ 1 ตัว ตามหลังข้อมูลก็จะเป็นพาริตี้บิต ซึ่งอาจจะใช้หรือไม่ก็ได้ พาริตี้บิตที่ทำหน้าที่เป็นตัวตรวจสอบความถูกต้องของสัญญาณที่ได้รับ พาริตี้อาจเป็นแบบคู่ (Even) หรือเป็น

แบบคี่ (Odd) หมายความว่า ถ้าหากพาริตี้คู่จำนวนบิตที่เป็น 1 ในช่วงบิตข้อมูลกับพาริตี้รวมแล้วจะต้องเป็นจำนวนคู่ ผู้ส่งจะต้องทำหน้าที่ตรวจสอบข้อมูลแล้วใส่พาริตี้เอง ฝ่ายรับเมื่อรับแล้วก็ต้องตรวจสอบดูว่าเป็นจริงดังสถานการณ์ที่ตั้งเอาไว้หรือไม่ หากผิดพลาดก็หมายความว่าสัญญาณที่รับนั้นผิดพลาดไปจากที่ส่งออกมา ทั้งนี้ทั้งนั้นจะต้องผิดเป็นจำนวนคี่เท่านั้น คือผิดไป 1 บิต,3 บิต, หรือ 5 บิต พร้อมกันจึงจะตรวจสอบได้ว่าผิด มองเห็นง่ายๆ ว่าถ้าผิดเป็นจำนวนคู่ ผลรวมของจำนวนหนึ่งก็ยังเป็นคู่อยู่ดี แต่ก็ไม่ได้หมายความว่า พาริตี้คี่ (Odd Parity) จะตรวจสอบการผิดพลาดเป็นจำนวนคี่ ความจริงแล้วตรวจสอบดูว่า สัญญาณที่รับเข้ามาเป็นคู่ก็ตรวจสอบดูว่ามีจำนวนคี่หรือเปล่าอย่างไรก็ตามโอกาสที่จะผิดพลาด 2 บิตพร้อมกันมีน้อยมาก



รูปที่ 2.9 แสดงการเรียงบิตในแต่ละแฟรมของอะซิงค์โครนัส

สำหรับสัญญาณอะซิงค์โครนัส หลังจากบิตพาริตี้แล้วก็จะต้องมีสต็อฟบิตซึ่งมีในความกว้างของสต็อฟบิต อาจจะเป็น 1, 1.5 หรือ 2 บิต แล้วแต่ผู้รับและผู้ส่งจะตกลงกัน การเริ่มใช้พอร์ทอนุกรม จึงจำเป็นจะต้องตั้งค่าต่างๆ สำหรับเป็นการส่งแบบอนุกรม ได้แก่

1. ความเร็วในการส่ง
2. ความยาวรหัส 1 อักขระ
3. บิตตรวจสอบ
4. จำนวนสต็อปบิต

ในการส่งโทรพิมพ์หรือโทรเลข เมื่อก่อนนี้ ใช้ความเร็ว 70 บอดและ 110 บอด สำหรับคอมพิวเตอร์ความเร็วในการส่งมีให้เลือกตั้งแต่ 110, 200, 300, 2400, 4800, 9600, บอด และสูงไปกว่านั้น เนื่องจากมี IC หลายเบอร์ทำหน้าที่รับส่งแบบอะซิงค์โครนัสให้ใช้ การส่งแบบอนุกรมจึงสะดวกสบายสำหรับการออกแบบพอร์ทอนุกรม

จะเห็นได้ว่ากลไกในการซิงค์โครนัสของการสื่อสารอะซิงค์โครนัสมีลักษณะเป็นไปทีละตัวอักขระจำนวนพัลส์ของสัญญาณที่ส่งออกยังมีบางส่วนใช้ในการควบคุมการส่งอยู่ อันได้แก่ สตาร์ทบิต, สต็อปบิต และพาริตี้บิต ทำให้ความเร็วการส่งอักขระต่อวินาทีน้อยลงไป การส่งสัญญาณด้วยความเร็ว 300 บอด สำหรับการเข้ารหัส 7 บิต ไม่ได้หมายความว่าส่งได้ 300 หารด้วย 7 อักขระต่อวินาที

2.2 หลักการส่งสัญญาณแบบ FSK (Frequency Shift Keying Modulation)

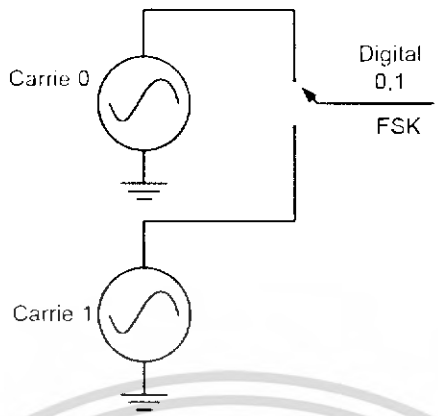
2.2.1 ทฤษฎีการมอดูเลตสัญญาณแบบ FSK

2.2.1.1 การมอดูเลตสัญญาณแบบ FSK

ตัวกำเนิดสัญญาณ FSK มีหลักการทำงานคือ เมื่อสัญญาณข้อมูลดิจิตอลที่เข้ามาเป็นแบบไบนารี่ ซึ่งมีการเปลี่ยนแปลงระดับลอจิกระหว่าง 1 กับ 0 จะทำให้ความถี่ของคลื่นพาห์มีการเปลี่ยนแปลงระหว่างสองความถี่ คือความถี่ที่ลอจิก " 1 " หรือความถี่มาร์ค (Mark Frequency, f_m) และความถี่ที่ลอจิก " 0 " หรือความถี่สเปซ (Space Frequency, f_s)

การเปลี่ยนแปลงของความถี่แต่ละครั้งจะเกิดขึ้นเมื่อมีการเปลี่ยนแปลงของสถานะของลอจิกของสัญญาณข้อมูลทางด้านอินพุตจากลอจิก 1 เป็นลอจิก 0 (หรือในทางกลับกันคือการเปลี่ยนสถานะจากลอจิก 0 เป็นลอจิก 1)นั่นคือการเปลี่ยนแปลของสัญญาณด้านเอาท์พุตจะเท่ากับอัตราการเปลี่ยนแปลงด้านอินพุต ในดิจิตอลมอดูเลชั่นนั้นอัตราการเปลี่ยนแปลงของสัญญาณด้านอินพุตจะเรียกว่า " อัตราบิต " (Bit rate) มีหน่วยเป็นบิตต่อวินาที (bps)ส่วนอัตราการเปลี่ยนแปลงของสัญญาณด้านเอาต์พุตจะเรียกว่า "อัตราการบอด" (Baud rate) ดังนั้นการส่งข้อมูลด้วยวิธีการแบบ FSK อัตราบิตจะเท่ากับอัตราบอดเสมอ Bit rate เป็นหน่วยวัดจำนวนบิตข้อมูลดิจิตอลที่เป็น 0 หรือ 1 ที่ส่งในวินาที ในแต่ละช่องสัญญาณ Baud rate เป็นหน่วยวัดจำนวนชุดข้อมูลที่ถูกส่งเข้าไปในสาย ซึ่งใน 1 ชุดอาจจะมีกี่จำนวนบิตก็ได้ในบางครั้ง Bit rate อาจจะเท่ากับ Baud rate ได้ เช่น ถ้าเป็นการมอดูเลต โดยที่ข้อมูลเป็นแบบไบนารี่ คือ 1 แทนข้อมูล 1 และ 0 แทนข้อมูล 0

การทำงานของตัวกำเนิดสัญญาณดิจิตอลเปลี่ยนแปลงความถี่มีหลักการทำงานดังรูปที่ 2.5 โดยมีแหล่งกำเนิดสัญญาณหรือออสซิเลเตอร์ (Oscillator) 2 ชุด (f_1 และ f_0) และมีสัญญาณข้อมูลดิจิตอลแบบไบนารี่เข้ามาทางอินพุต เพื่อควบคุมตัวกำเนิดสัญญาณเอฟเอสเค ซึ่งการทำงานของตัวกำเนิดสัญญาณดิจิตอลเปลี่ยนแปลงความถี่จะมีลักษณะคล้ายสวิตซ์เลื่อนแบบ 2 ตำแหน่ง จะเลื่อนไปตามสัญญาณควบคุม ดังนั้นสัญญาณดิจิตอลที่เข้ามาเป็นสัญญาณที่เข้ามาควบคุมการเลื่อนตำแหน่งของสวิตซ์ที่จะให้สัญญาณ f_1 หรือ f_0 ออกมาทางด้านเอาต์พุต



รูปที่ 2.10 หลักการทำงานของสัญญาณเอฟเอสเค

สัญญาณดิจิตอลเปลี่ยนแปลงความถี่ที่ได้จะมีสมการดังนี้คือ

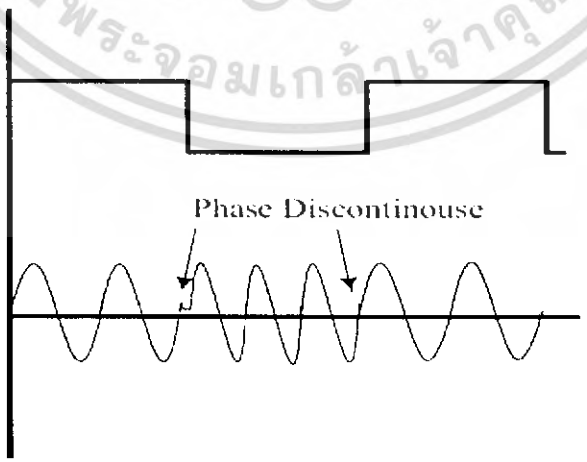
$$v(t) = A\cos(\omega_c \pm \Delta\omega)t$$

โดย A = ขนาดของสัญญาณเอฟเอสเค

ω_c = ความถี่คลื่นพาห์

$\Delta\omega$ = ความถี่เบี่ยงเบน

เครื่องหมาย + และ - จะเปลี่ยนแปลงเป็นอย่างใดอย่างหนึ่งตามสัญญาณดิจิตอล และจะเห็นว่าสัญญาณดิจิตอลเปลี่ยนแปลงความถี่ จะมีขนาดของสัญญาณที่คงที่ แต่มีความถี่ที่เปลี่ยนแปลงไปตาม หรือ สัญญาณดิจิตอลเปลี่ยนแปลงความถี่จะมีลักษณะเฟสที่ไม่ต่อเนื่อง (Phase Discontinuous) ตรงบริเวณที่ตรงกับตำแหน่งการเปลี่ยนแปลงสถานะลอจิกของสัญญาณดิจิตอล เนื่องจากการใช้แหล่งกำเนิดสัญญาณแยกกัน 2 ชุด (f_1 และ f_0) ดังรูปที่ 2.11 และสัญญาณดิจิตอลเปลี่ยนแปลงความถี่ได้นี้จะมีรูปคลื่นมากกว่า 1 คาบเวลา ต่อสัญญาณดิจิตอล 1 บิต



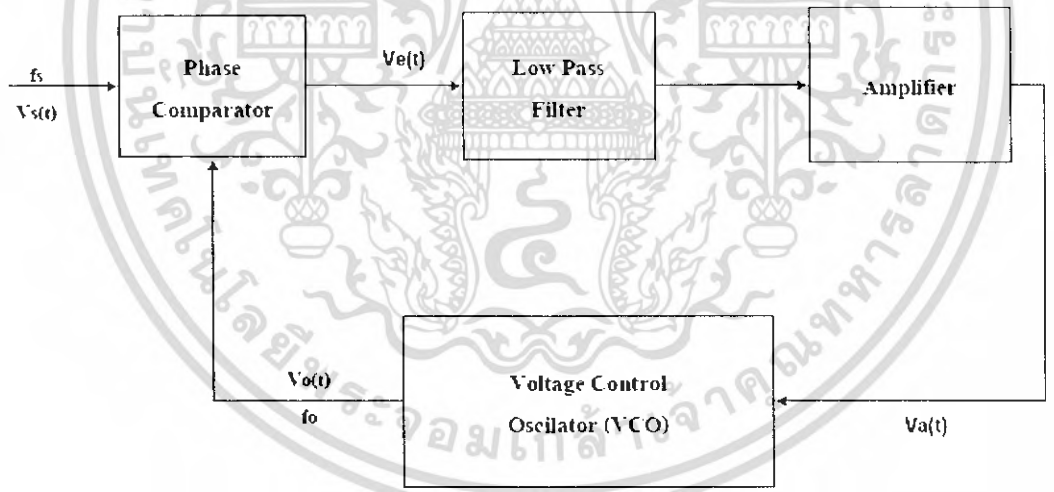
รูปที่ 2.11 รูปสัญญาณเอฟเอสเค

2.2.1.2 การดีมอดูเลตสัญญาณแบบ FSK (Frequency Shift Keying Demodulation)

วงจรที่ใช้โดยทั่วไปของการดีมอดูเลตไบนารี่เอฟเอสเค คือ วงจรเฟสล็อคลูป (Phase Comparator : PD) ซึ่งระบบเฟสล็อคลูปแอฟเอสเคดีมอดูเลเตอร์ (PLL FSK demodulator) สัญญาณอินพุตของเฟสล็อคลูปจะมีการเปลี่ยนแปลงระหว่างมาร์ค (Mark) และสเปซ (Space) ซึ่งจะเกิดค่าโวลท์เตจกระแสตรงผิดพลาดที่เอาต์พุต ซึ่งเกิดจากการเปรียบเทียบเฟสจากความถี่ที่ชิฟไป เพราะที่อินพุตจะมีเพียงสองความถี่เท่านั้น จึงทำให้เกิดค่าโวลท์เตจผิดพลาดสองสถานะที่เอาต์พุต ซึ่งก็คือ ลอจิก 1 และลอจิก 0 ดังนั้น เอาต์พุตของระบบเอฟเอสเคจะมีระดับตามสัญญาณเอฟเอสเคอินพุต ซึ่งโดยทั่วไปความถี่ของเฟสล็อคลูปจะมีค่าเท่ากับความถี่กลางของเอฟเอสเคมอดูเลเตอร์

2.2.1.3 พื้นฐานเฟสล็อคลูป (Phase Lock Loop : PLL)

เฟสล็อคลูป คือ ระบบที่มีการป้อนกลับ ประกอบด้วยวงจรเปรียบเทียบเฟส (Phase Comparator : PD) วงจรกรองความถี่ต่ำผ่าน (Low pass Filter) วงจรขยาย (Amplifier) ซึ่งอยู่ทางด้านที่ทางเดินสัญญาณไปข้างหน้า และออสซิเลเตอร์ควบคุมแรงดัน (Voltage Control Oscillator : VCO) อยู่ทางด้านป้อนกลับบล็อกไดอะแกรมของระบบเฟสล็อคลูป ดังแสดงในรูปที่ 2.12



รูปที่ 2.12 บล็อกไดอะแกรมเฟสล็อคลูป

2.2.1.3.1 เฟสดีเทคเตอร์

เฟสดีเทคเตอร์ ทำหน้าที่เปรียบเทียบเฟสของสัญญาณอินพุต ซึ่งจะมี 2 ค่าของเฟสที่ต่างกัน เรียกว่า ความต่างเฟส (Phase Error) ซึ่งจะมีค่าน้อยที่สุดเป็นศูนย์ และจะมีค่ามากที่สุดเป็น $\pi/2$ เฟสดีเทคเตอร์จะทำการเปลี่ยนค่าความต่างเฟสนี้ให้กลายเป็นแรงดันด้วยค่า Kd (Gain Convertion) ซึ่งมีหน่วยเป็นโวลต์ต่อ เรเดียน

2.2.1.3.2 ลูปฟิลเตอร์

ลูปฟิลเตอร์ ทำหน้าที่กรองสัญญาณความถี่สูงที่ออกจากเฟสดีเทคเตอร์ เนื่องจากเฟสดีเทคเตอร์ให้ เอาพุตเป็นสัญญาณดีซี (DC) ที่มีแรงดันเอซี (AC) มาด้วย สัญญาณความถี่ที่ได้เกิดจากความต่างเฟส ยิ่งต่าง เฟสมากความถี่จะยิ่งสูง ดังนั้นลูปฟิลเตอร์จึงช่วยกรองเอาสัญญาณความถี่สูง ซึ่งแสดงว่าความต่างเฟสออกมาทำให้ระบบสามารถแคปเจอร์ (Capture) สัญญาณได้ช่วงหนึ่ง และช่วยให้รักษาการล็อคไว้ได้อีกด้วย

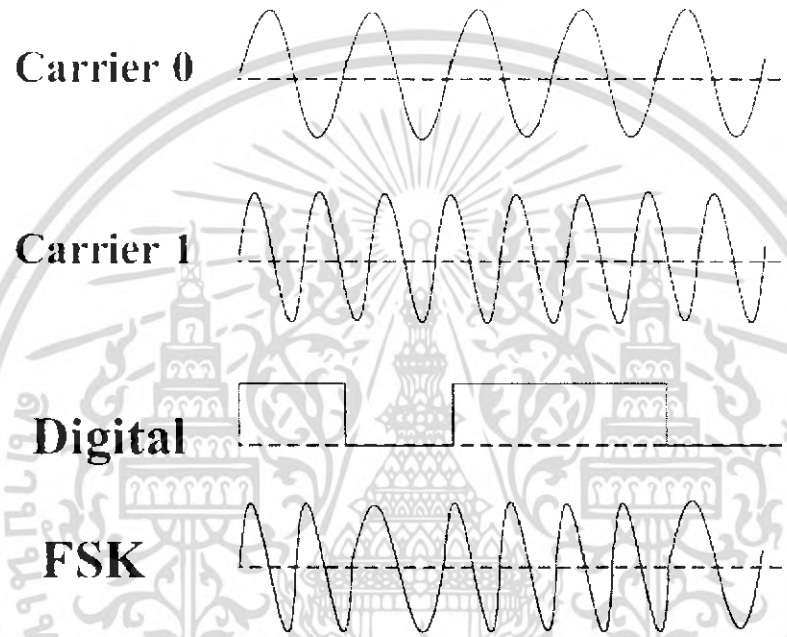
2.2.1.3.3 วีซีโอ (Voltage Control Oscillator)

วีซีโอ ทำหน้าที่ผลิตสัญญาณความถี่โดยการควบคุมแรงดันอินพุตด้วยค่า Kd เท่ากับ Ko มีหน่วยเป็นเรเดียนต่อโวลต์ ระดับแรงดันนี้ได้จากเอาต์พุตของลูปฟิลเตอร์ ความถี่ที่ออกมาจากลูปฟิลเตอร์จะมีผมทำให้เอาต์พุตของวีซีโอเปลี่ยนความถี่ด้วยเช่นกัน ส่งของวงจรขยายสัญญาณซึ่งจะต่อไประหว่างลูปฟิลเตอร์กับวีซีโอ เพื่อเพิ่มพิสัยการล็อคของระบบรวมทั้งเป็นการดัดแปลงการทำงานของเฟสล็อคลูปให้สามารถทำงานได้ตามต้องการ

ขณะที่ยังไม่มีสัญญาณเข้าในระบบ แรงดันควบคุม Vd(t) จะเท่ากับศูนย์วีซีโอ(VCO)จะทำงานโดยตั้งความถี่ไว้ที่ f_0 ซึ่งเรียกว่าความถี่ที่ทำงานเสรี (Free Running Frequency) ถ้าขยายสัญญาณอินพุตเข้าไปในระบบ วงจรเปรียบเทียบเฟสจะทำการเปรียบเทียบเฟสและความถี่ของสัญญาณอินพุตกับวีซีโอ และผลิตแรงดันคลาดเคลื่อน Ve(t) ซึ่งสัมพันธ์กับความแตกต่างของเฟสและความถี่ระหว่างสัญญาณทั้งสองแรงดันคลาดเคลื่อนนี้จะถูกกรองและขยายส่งไปควบคุมวีซีโอ ในการทำงานเช่นนี้แรงดันควบคุมจะไปบังคับความถี่วีซีโอให้เปลี่ยนในทิศทางที่จะลดความถี่ที่แตกต่างระหว่าง f_0 และสัญญาณที่เข้า ถ้าความถี่ของสัญญาณที่เข้าใกล้เคียงกับ f_0 จากการป้อนกลับของเฟสล็อคลูปที่เกิดขึ้นพร้อมๆ กันกับสัญญาณที่เข้ามา ขณะที่ทำการล็อคนั้น ความถี่วีซีโอจะเท่ากับสัญญาณอินพุต แต่เฟสยังต่างกันอยู่ ซึ่งมีความจำเป็นต่อการผลิตแรงดันคลาดเคลื่อนที่จะไปคอยปรับความถี่วีซีโอจากค่าปรับความถี่ที่ทำงานอิสระให้เท่ากับความถี่ที่เข้ามา ดังนั้นเฟส ล็อคลูปจะยังคงรักษาสภาพการล็อค การที่ระบบสามารถปรับตัวได้เอง ทำให้เฟสล็อคลูปสามารถติดตามการ ล็อคกับสัญญาณที่เข้ามา เรียกว่าพิสัยล็อค (Lock Range) ของระบบ ซึ่งจะมีค่าขึ้นอยู่กับแรงดันคลาดเคลื่อน โดยจะไม่ขึ้นกับขอบของแถบความถี่ (Band Edge) ของวงจรกรองช่วงความถี่นี้จะมากกว่าช่วงความถี่เฟสล็อคลูปสามารถทำการล็อคอย่างแท้จริงกับสัญญาณอินพุต ช่วยความถี่หลังนี้เรียกว่า พิสัยแคปเจอร์ (Capture Range) ของระบบทั้งหมด โดยมีผลเกี่ยวกับความคม (Selectivity) ของวงจรเฟสล็อคลูปและยังช่วยเพิ่มความปลอดภัยเกี่ยวกับสัญญาณรบกวนทางด้านเอาต์พุตที่มารบกวนได้เป็นอย่างดี

2.2.2 แบนด์วิทของสัญญาณที่มอดูเลตแบบ FSK (FSK Band)

ในระบบที่สื่อสารข้อมูลด้วยสัญญาณอนาล็อคหรือสัญญาณความถี่นั้น แบนด์วิทเป็นสิ่งที่ต้องพิจารณาเป็นอันดับแรก เนื่องจากวิธีการของเอฟเอสเคอยู่บนพื้นฐานเดียวกับวิธีการของเอฟเอ็ม ดังนั้นการอธิบายถึงสูตรต่างๆ ก็ใช้หลักการของเอฟเอ็มทุกประการ นั่นคือหลักการของ VCO (Voltage Control Oscillator) จะเห็นว่าอัตราการเปลี่ยนแปลงที่เร็วที่สุดของสัญญาณอินพุตเกิดขึ้นเมื่อข้อมูลไบนารี่ลักษณะเป็น 1 และ 0 สลับกัน ซึ่งก็คือสัญญาณ Square Wave นั่นเอง



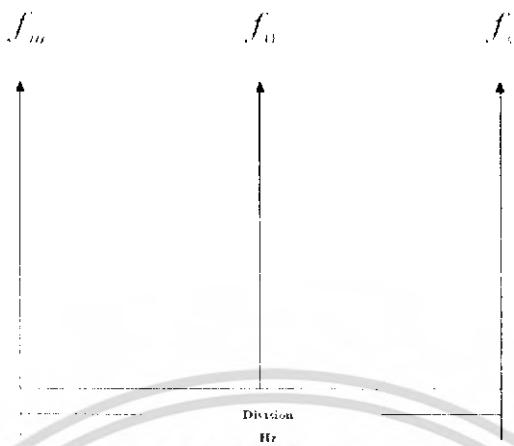
รูปที่ 2.13 สัญญาณอินพุตและเอาต์พุตของสัญญาณเอฟเอสเค

ตามตัวอย่างในรูปที่ 2.13 เป็นสัญญาณในช่วง T1 ความถี่หลักของคลื่นสี่เหลี่ยมจะมีค่าเท่ากับครึ่งหนึ่งของอัตราบิต ดังนั้นถ้าพิจารณาเฉพาะความถี่หลักเพียงอย่างเดียวแล้วความถี่สูงสุดของสัญญาณดิจิตอลที่ต้องการนำมามอดูเลตแบบเอฟเอสเค จะเท่ากับครึ่งหนึ่งของอัตราบิต คือ

$$fa_{\max} = \frac{Bit_{rate}}{2}$$

เมื่อ $fa_{\max}$ = เท่ากับความถี่สูงสุดของสัญญาณดิจิตอลที่จะนำมอดูเลต

ความถี่กลาง (Center Frequency $= f_0$) ของวีซีโอจะอยู่ในตำแหน่งความถี่มาร์คกับความถี่สเปซ ดังรูปที่ 2.14



รูปที่ 2.14 การเบี่ยงเบนความถี่

ลอจิก 1 ด้านอินพุตจะเลื่อนความถี่ VCO จาก f_0 ไปเป็น f_s จะเห็นว่าการเปลี่ยนแปลงข้อมูลไบนารี่ด้านอินพุตจาก 1 ไป 0 หรือจาก 0 ไป 1 จะทำให้ความถี่เอาต์พุตของ VCO เลื่อนหรือเบี่ยงเบนไปมาระหว่างกัน เนื่องจากได้กล่าวมาแล้วว่า เอฟเอสเคนั้นคือการมอดูเลตแบบเอฟเอ็ม ดังนั้นดัชนีมอดูเลต (Modulate index : mi) ในเอฟเอสเคทำได้จากเอฟเอ็มคือ

$$mi = \frac{f}{f_a}$$

เมื่อ mi คือ อัชนีการมอดูเลต

f คือ การเบี่ยงเบนความถี่ใดๆ จากความถี่กลาง

f_a คือ ความถี่ของสัญญาณที่นำมามอดูเลต

f_a mi ที่ยอมรับมีค่าได้สูงสุดคือ 1 miทำให้แบนด์วิทกว้างที่สุดซึ่งเกิดขึ้นเมื่อการเบี่ยงเบนของความถี่ที่ถูกมอดูเลตแล้วและความถี่ของสัญญาณที่นำมามอดูเลตมีค่าสูงสุดในเอฟเอสเค มอดูเลต ค่า f เป็นการเบี่ยงเบนความถี่สูงสุด (Peak Frequency Deviation) ของสัญญาณถูกนำมามอดูเลตแล้ว ซึ่งมีค่าเท่ากับความแตกต่างระหว่าง f_0 กับ f_m หรือ f_0 กับ f_s ซึ่ง ก็คือครึ่งหนึ่งของความแตกต่างระหว่าง f_m กับ f_s นั่นคือ $f = \frac{(f_s - f_m)}{2}$ การเบี่ยงเบนความถี่สูงสุดที่ขึ้นอยู่กับขนาดหรือแอมปลิจูดของสัญญาณที่นำมามอดูเลต เมื่อสถานะลอจิกเป็น 1 จะให้แรงดันตามสถานะเช่น 5 โวลท์ หรือถ้าเป็นลอจิก 0 ก็จะได้แรงดันเป็น 0 โวลต์ ดังนั้นความถี่เบี่ยงเบนของเอฟเอสเคจะเบี่ยงเบนความถี่คงที่ และอยู่ในระดับการเบี่ยงเบนความถี่สูงสุดเสมอ เป็นความถี่หลักของข้อมูลไบนารี่ด้านอินพุต ซึ่งจะทำให้แบนด์วิทกว้างที่สุดเมื่อ $f_a = \frac{Bit_{rate}}{2}$ เท่านั้น เพราะฉะนั้นเราสามารถหาค่า mi ได้จาก

$$mi = \frac{(f_s - f_m)}{2} \times \frac{f_b}{2}$$
$$= \frac{(f_s - f_m)}{f_b}$$

เมื่อ $(f_s - f_m)$ คือ ความถี่เบี่ยงเบนสูงสุด

f_b คือ อัตราการเข้ารหัสของไบนารี่อินพุต

ในการส่งสัญญาณเอฟเอ็มทั่วๆไป ความกว้างของแบนด์วิทจะแปรผันโดยตรงกับค่า *mi* ซึ่งเช่นเดียวกับเอฟเอสเค ที่ต้องมีค่าไม่ต่ำกว่า 1 เพื่อให้เป็นเอฟเอ็มแบนด์แคบ (Narrow Band FM)

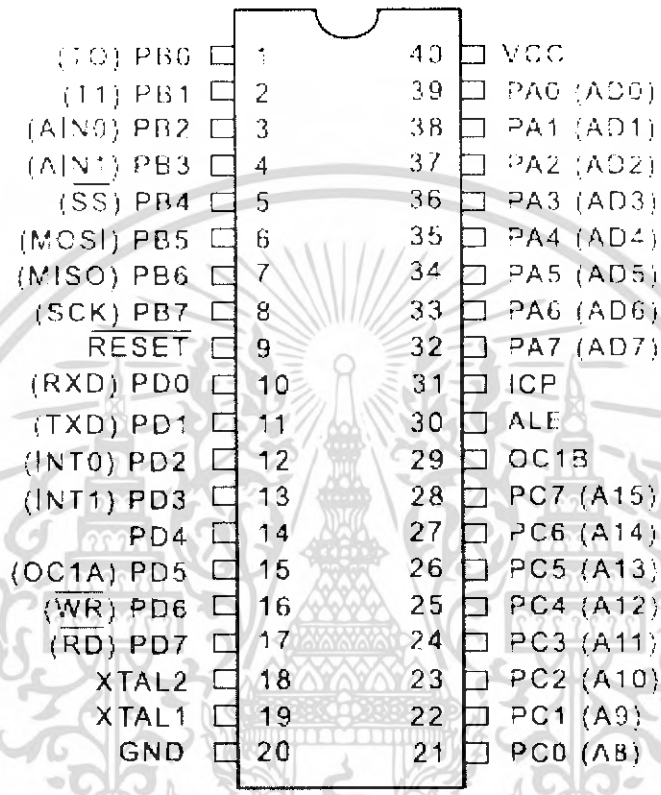
2.3 ทฤษฎีพื้นฐานเกี่ยวกับไมโครคอนโทรลเลอร์ AT90S8515

2.3.1 สถาปัตยกรรม และคุณสมบัติโดยทั่วไปของ AT90S8518

- สถาปัตยกรรมภายในถูกออกแบบให้ใช้สถาปัตยกรรมแบบ RISC (Reduce Instruction Set Computer)
- มีคำสั่งควบคุมการทำงานของไมโครคอนโทรลเลอร์จำนวน 118 คำสั่ง
- หน่วยความจำแบบ Flash สำหรับบันทึก Program Memory ขนาด 8 Kbytes
- หน่วยความจำแบบ EEPROM สำหรับบันทึก Data Memory ขนาด 512 byte
- หน่วยความจำแบบ RAM ขนาด 512 Byte
- กลุ่มรีจีสเตอร์ใช้งานทั่วไป 8 บิต จำนวน 32 ตัว
- ความถี่สัญญาณนาฬิกา 0 – 8 MHz
- ระบบการตรวจจับสัญญาณอนาลอก(Analog Comparator)
- ระบบตรวจจับการทำงานผิดพลาดของ CPU (Watchdog Timer With On – Chip Oscillator)
- ระบบอินเตอร์รัพท์จากภายนอก (External Interrupt)
- Timer/Counter ขนาด 16 บิต 1 Channel
- Timer/Counter ขนาด 8 บิต 1 Channel
- ระบบการสื่อสารผ่านพอร์ทอนุกรม

AT90S8515 เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ที่สถาปัตยกรรมแบบ RISC (Reduce Instruction Set Computer) ซึ่งทำให้การประมวลผลมีความเร็ว 1 คำสั่ง/1สัญญาณนาฬิกา หรือ CPU สามารถประมวลผลคำสั่งได้ 1 MIPS/MHz

PDIP

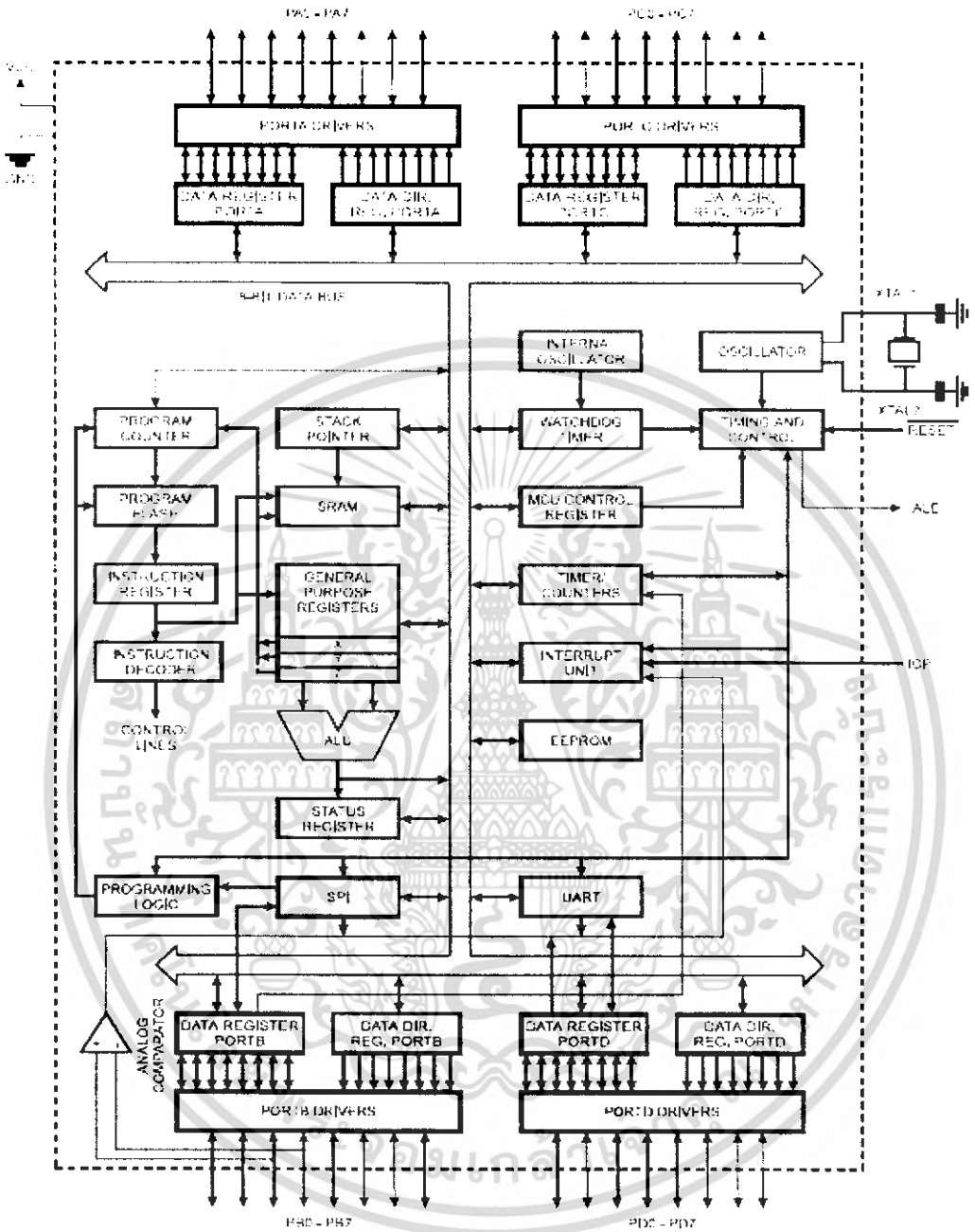


รูปที่ 2.15 โครงสร้างภายนอกและตำแหน่งขาของ AT90S8515

ภายในประกอบด้วยรีจีสเตอร์ใช้งานทั่วไปขนาด 8 บิต จำนวน 32 ตัว ซึ่งแต่ละตัวจะต่อเข้ากับ ALU โดยตรงทำให้การประมวลผลต่อ 1 คำสั่งมีความเร็วกว่า CPU ที่สถาปัตยกรรมแบบ CISC

2.3.2 โครงสร้างภายในของ AT90S8518

AT90S8515 มีหน่วยความจำสำหรับ Program Memory แบบ Flash ขนาด 8 Kbytes หน่วยความจำสำหรับ Data Memory แบบ EEPROM ขนาด 512 Byte และหน่วยความจำแบบ RAM ขนาด 512 Byte มีพอร์ทที่สามารถทำงานได้ 2 ทิศทาง จำนวน 32 เส้นสัญญาณ และ ระบบ Timer/Counter จำนวน 2 ชุดที่มีโหมดการทำงาน



รูปที่ 2.16 บล็อกไดอะแกรมโครงสร้าง ของ AT90S8515

2.3.3 รายละเอียดโครงสร้างภายนอกของ AT90S8518

Vcc ขาจ่ายไฟให้กับ CPU

GND กราวด์

Port A (PA7..PA0) เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA

Port B (PB7..PB0) เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA โดยแต่ละขาสัญญาณจะถูกใช้งานในฟังก์ชั่นอื่น ๆ ได้อีก

Port C (PC7..PC0) เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA โดยแต่ละขาสัญญาณจะถูกใช้งานในฟังก์ชั่นอื่น ๆ ได้อีก

Port D (PD7..PD0) เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA โดยแต่ละขาสัญญาณจะถูกใช้งานในฟังก์ชั่นอื่น ๆ ได้อีก

RESET ใช้เป็นขารีเซ็ทโดยการป้อน 0

XTAL1 เป็นขาอินพุตของ OSC และ รับสัญญาณจาก Clock

XTAL2 เป็นขาเอ้าท์พุทของ OSC

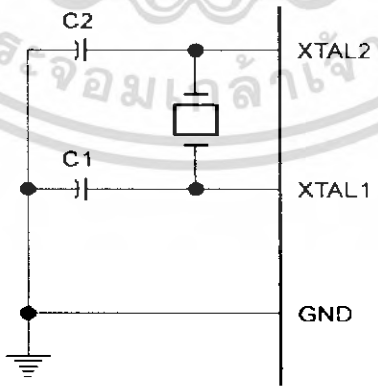
ICP เป็นขาที่ใช้เลือกให้ทำงานเป็น Timer/Counter1

OC1B เป็นเอาท์พุตของสัญญาณนาฬิกา

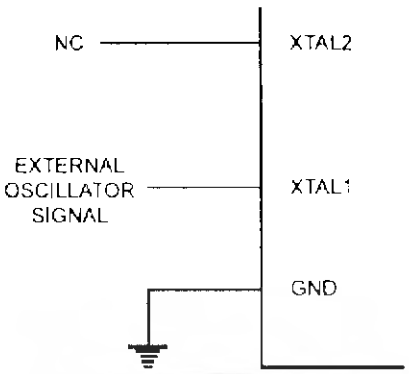
ALE จะต้องเลือกเมื่อต้องการใช้งานหน่วยความจำภายนอก

2.3.4 การใช้งาน CRYSTRAL OSCILLATOR ของ AT90S8518

โดยขา XTAL1 เป็นขาอินพุต และขา XTAL2 เป็นขาเอ้าท์พุท ซึ่งถ้าต้องการใช้ OSC ภายในจะต้องต่อ CRYTAL ค่อมขา XTAL1 และขา XTAL2 โดยมี CAPACITOR ต่อจากขาทั้ง 2 ลงกราวด์ ถ้าต้องการ OSC จากภายนอก ให้ปล่อยขา XTAL2 ลอย และป้อน CLOCK เข้าที่ขา XTAL1 ซึ่งแสดงไว้ในรูปที่ 2.17 และ รูปที่ 2.18



รูปที่ 2.17 แสดงการใช้ OSC ภายใน MCU



รูปที่ 2.18 แสดงการใช้ OSC ภายนอก MCU

2.3.5 สถาปัตยกรรมภายใน และรีจีสเตอร์ใช้งานทั่วไปของ AT90S8518

2.3.5.1 สถาปัตยกรรมภายใน

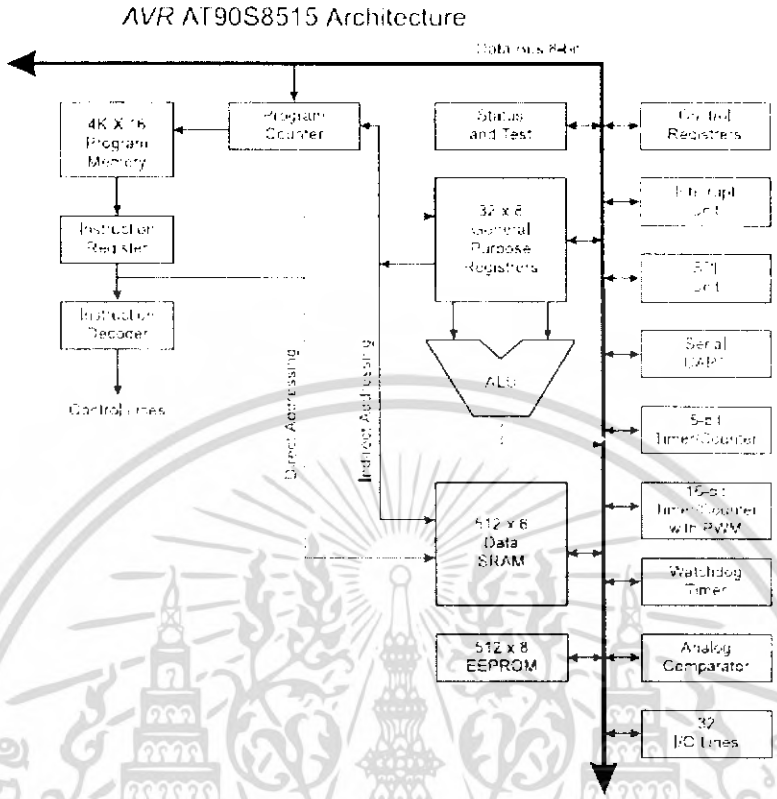
โครงสร้างภายในจะประกอบด้วยรีจีสเตอร์ใช้งานทั่วไปขนาด 8 บิต จำนวน 32 ตัว ที่สามารถเข้าถึงข้อมูลได้ใน 1 Clock ซึ่งหมายความว่า MCU สามารถจัดการข้อมูลภายในรีจีสเตอร์ใช้งานทั่วไปได้เสร็จภายใน 1 Clock ของสัญญาณนาฬิกา

โดยรีจีสเตอร์ R26 – R31 เป็นรีจีสเตอร์ขนาด 8 บิตจำนวน 6 ตัว สามารถจับคู่เพื่อใช้เป็นรีจีสเตอร์ขนาด 16 บิต 3 ตัว โดยใช้ชื่อว่า รีจีสเตอร์ X, Yและ Z

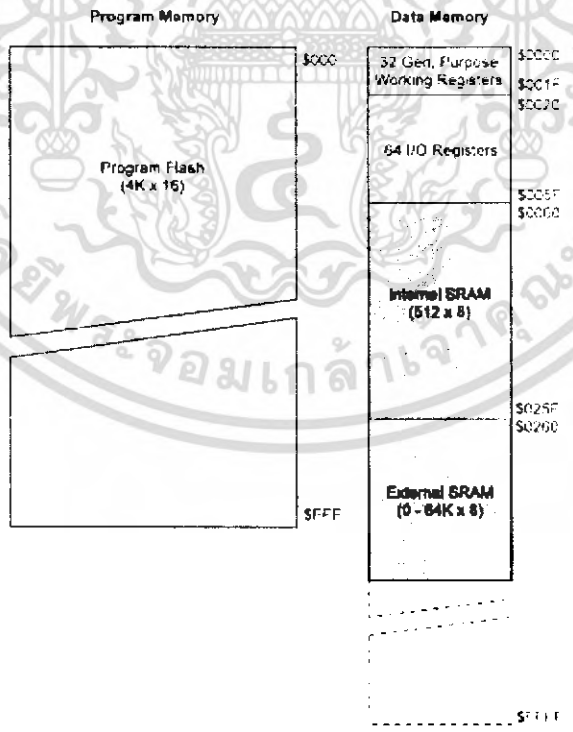
ALU สนับสนุนการกระทำทางคณิตศาสตร์และลอจิก ระหว่าง รีจีสเตอร์กับรีจีสเตอร์ หรือระหว่างรีจีสเตอร์กับค่าคงที่ ซึ่งการเรียกใช้รีจีสเตอร์ใช้งานทั่วไป สามารถกระทำได้โดยการอ้างหน่วยความจำภายใน ที่ตำแหน่ง S00 – S1F จำนวน 32 ตำแหน่ง และใน MCU ได้จัดแบ่งให้มีรีจีสเตอร์ ที่ใช้ควบคุมการทำงานของหน่วยอินพุตและเอาท์พุตต่างๆ อีก 64 ตำแหน่ง โดยสามารถเรียกใช้งานได้โดยการอ้างตำแหน่งหน่วยความจำที่ตำแหน่ง S20 – S1F

ระบบการทำงานของไมโครคอนโทรลเลอร์ใช้หลักการออกแบบของ HAVARD ด้วยการแยกระบบบัสของ PROGRAM และ DATA ออกจากกัน โดยโปรแกรมจะมีการประมวลผลด้วย Single Level Pipelining ซึ่งทำให้ CPU สามารถ Fetch และ Execute คำสั่งได้ภายใน 1 คาบเวลา

ด้วยคำสั่ง JUMP และ CALL แบบ RELATIVE ที่สามารถกระโดดข้ามการทำงานได้ไกลถึง 2K/4K ซึ่งใน 1 คำสั่งจะใช้รหัสการทำงาน 16 Bit หรือ 1 WORD โดยทุกครั้งที่มีการอินเตอร์รัพท์ หรือการข้ามไปทำงานในโปรแกรมย่อยค่าของ PROGRAM COUNTER (PC) จะถูกเก็บลง STACK ซึ่งจะใช้พื้นที่หน่วยความจำใน SRAM บางส่วนเพื่อทำเป็นพื้นที่ของ STACK



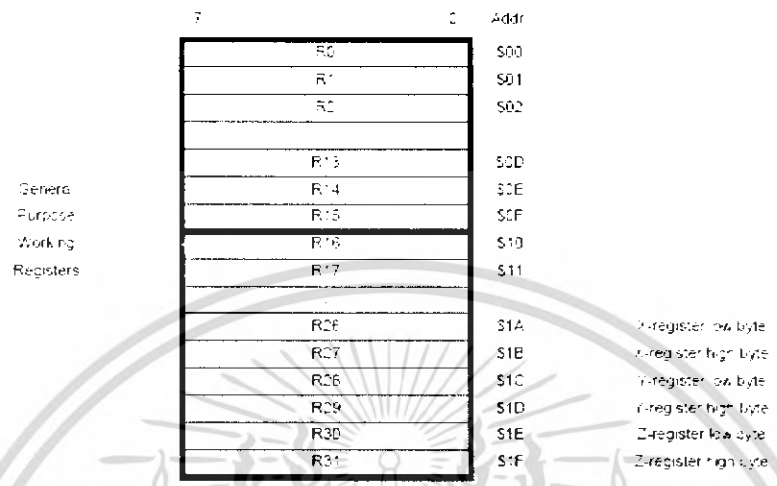
รูปที่ 2.19 แสดงสถาปัตยกรรมแบบ RISC ของ AT90S8515



รูปที่ 2.20 โครงสร้างหน่วยความจำ

2.3.5.2 รีจีสเตอร์ใช้งานทั่วไป

โครงสร้างของรีจีสเตอร์ใช้งานทั่วไปทั้ง 32 ตัว แสดงดังรูปที่ 2.21



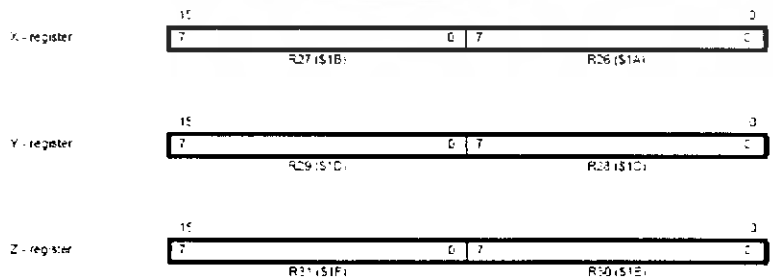
รูปที่ 2.21 แสดงโครงสร้างของรีจีสเตอร์ใช้งานทั่วไป

รีจีสเตอร์ทั้งหมดสามารถใช้ชุดคำสั่ง เพื่อเข้าถึงได้โดยตรงและจะใช้ช่วงเวลาการเข้าถึงเพียง 1 Clock โดยคำสั่ง SBCI, SUBI, CPI, ANDI และ ORI ซึ่งกระทำระหว่างรีจีสเตอร์กับค่าคงที่ และรีจีสเตอร์กับรีจีสเตอร์และคำสั่ง LDI ที่ใช้โหลดค่าคงที่เข้าในรีจีสเตอร์ จะต้องใช้งานกับรีจีสเตอร์ R16 – R31 ส่วนคำสั่ง SBC, SUB, CP, AND และ OR และคำสั่งใช้งานอื่นๆสามารถใช้งานได้ในรีจีสเตอร์ทั่วไป

รูปที่ 2.21 แสดงการจัดวางตำแหน่งของรีจีสเตอร์ใช้งานทั่วไปทั้งหมด โดยรีจีสเตอร์ที่สามารถนำมาใช้งานเป็นรีจีสเตอร์คู่เพื่อทำให้เป็นตัวชี้ข้อมูลที่อยู่ในหน่วยความจำ ซึ่งรีจีสเตอร์ในกลุ่มนี้จะใช้ชื่อว่า X, Yและ Z

2.3.5.2.1 รีจีสเตอร์ X, รีจีสเตอร์ Y และรีจีสเตอร์ Z

รีจีสเตอร์ R26 R31 สามารถนำมาต่อกันเพื่อทำเป็นรีจีสเตอร์คู่ เพื่อใช้งานเป็นตัวชี้ข้อมูลในชื่อของรีจีสเตอร์ X, Yและ Z



รูปที่ 2.22 แสดงรีจีสเตอร์ X,Yและ Z

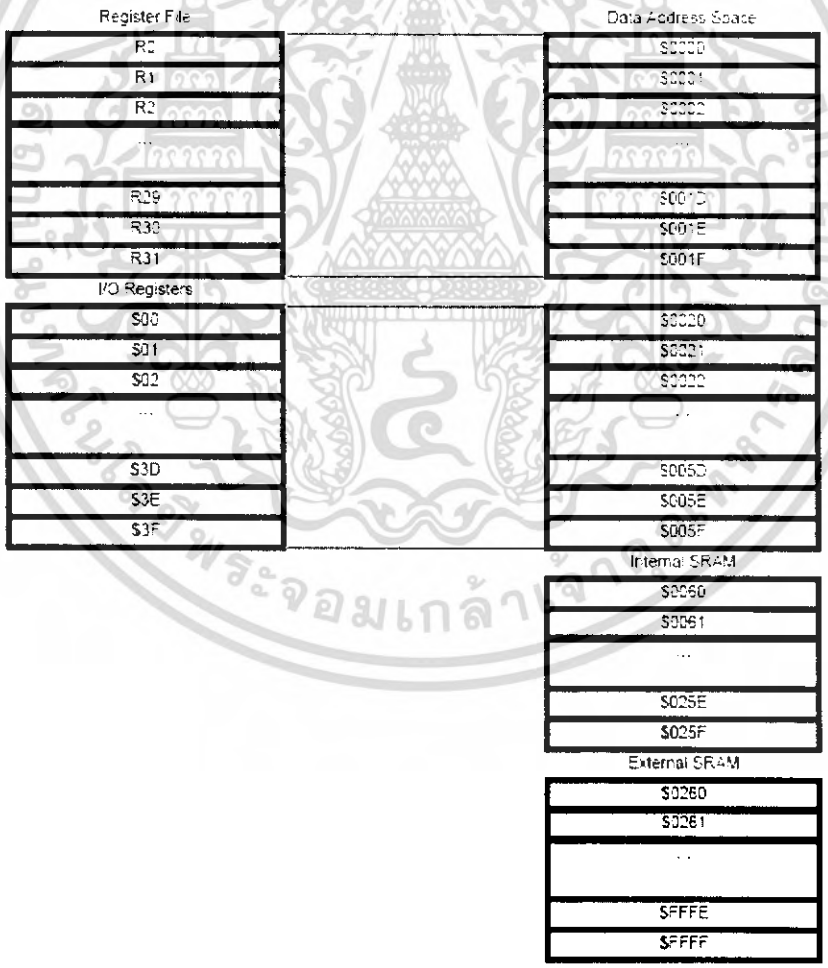
2.3.5.2.2 หน่วยประมวลผลทางคณิตศาสตร์และลอจิก

ระบบการประมวลผลที่มีประสิทธิภาพของ AVR คือ ALU สามารถเชื่อสารข้อมูลโดยตรงกับรีจีสเตอร์ใช้งานทั่วไปได้ทั้ง 32 ตัว โดย ALU ได้จัดแบ่งระบบการจัดการข้อมูลไว้ 3 ส่วน คือ ส่วนของการจัดการทางคณิตศาสตร์ ส่วนของการกระทำทางลอจิก และในส่วนของการกระทำกับบิต

2.3.5.2.3 หน่วยความจำ SRAM

หน่วยความจำภายใน MCU จัดไว้ 352 ตำแหน่ง โดยหน่วยความจำทั้งหมดถูกแบ่งออกเป็นพื้นที่ของรีจีสเตอร์ใช้งาน I/O และหน่วยความจำภายใน SRAM

การเข้าถึงข้อมูลถูกแบ่งออกเป็น 5 ส่วน คือ Direct, indirect with Displacement, Indirect with Pre – Decrement และ Indirect with Post – Increment โดย 96 ตำแหน่งแจกแจงเป็นส่วนของรีจีสเตอร์ และอีก 512 ตำแหน่งสำหรับหน่วยความจำภายใน SRAM

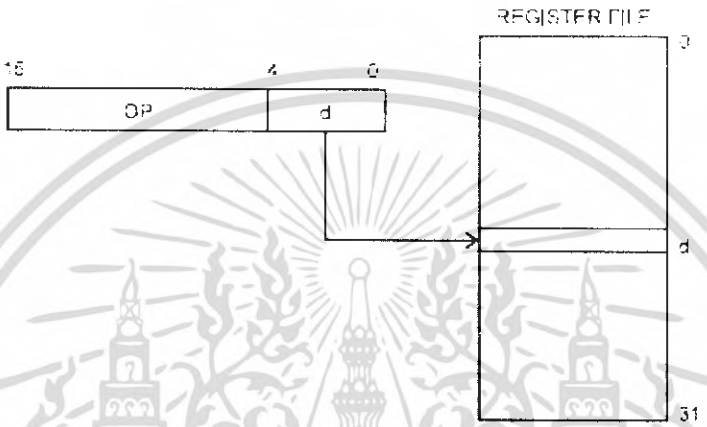


รูปที่ 2.23 แสดงการจัดการหน่วยความจำ SRAM

2.3.5.3 การเข้าถึงข้อมูลของ AT90S8515

ไมโครคอนโทรลเลอร์AT90S8515เป็นไมโครคอนโทรลเลอร์ที่ถูกสร้างขึ้นมาจากสถาปัตยกรรมแบบ RISC โดย Program Memory เป็นแบบ FLASH ซึ่งโครงสร้างภายในถูกออกแบบให้มีการเข้าถึงข้อมูลได้หลายลักษณะดังต่อไปนี้

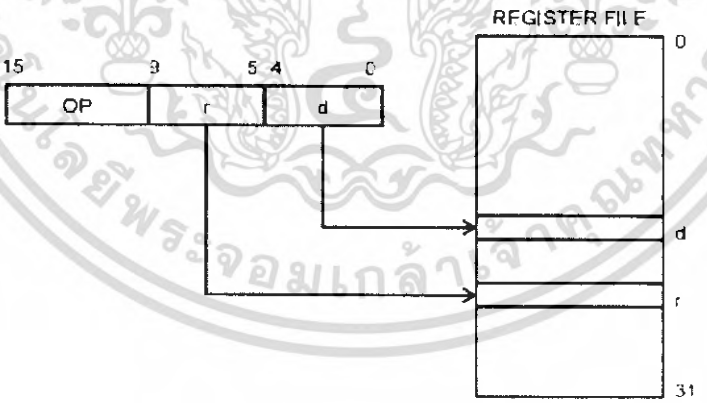
2.3.5.3.1 Register Direct, Single Register Rd



รูปที่ 2.24 แสดงการเข้าถึงข้อมูลแบบ Direct Single Register Rd

ค่าของตัวกระทำจะเก็บไว้ในรีจีสเตอร์ d(RD)

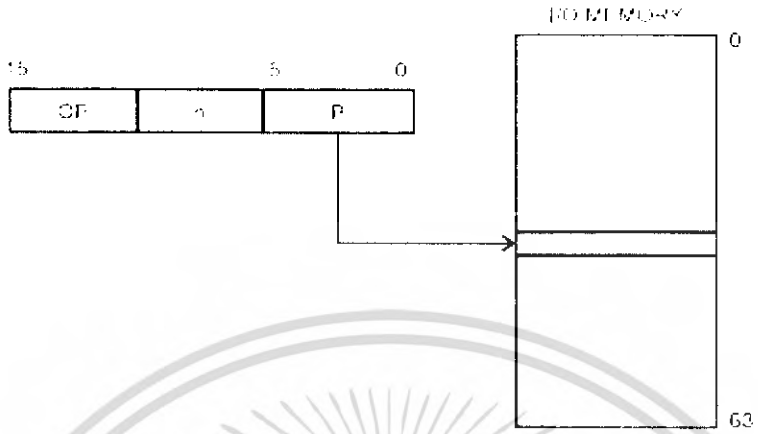
2.3.5.3.2 Register Direct ,Two Register Rd And Rr



รูปที่ 2.25 แสดงการเข้าถึงข้อมูลแบบ Direct Register Addressing Rd – and Rr

ค่าของตัวกระทำจะเก็บไว้ในรีจีสเตอร์ r(Rd) และ d(Rd) และผลของการกระทำจะเก็บไว้ในรีจีสเตอร์ d(Rd)

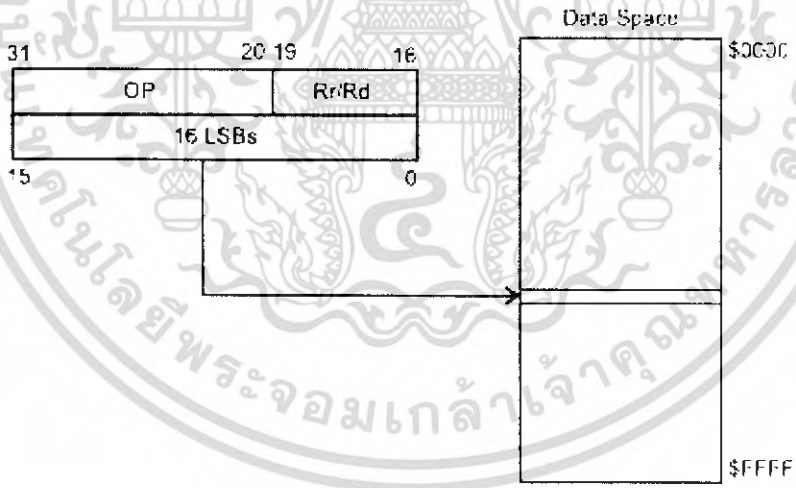
2.3.5.3.3 I/O Direct



รูปที่ 2.26 แสดงการเข้าถึงข้อมูลแบบ I/O Direct Addressing

ตำแหน่งของตัวกระทำจะถูกเก็บไว้ใน 6 บิต ของรหัสคำสั่ง n คือ ตำแหน่งของ รีจีสเตอร์ที่เป็น Source หรือ Destination

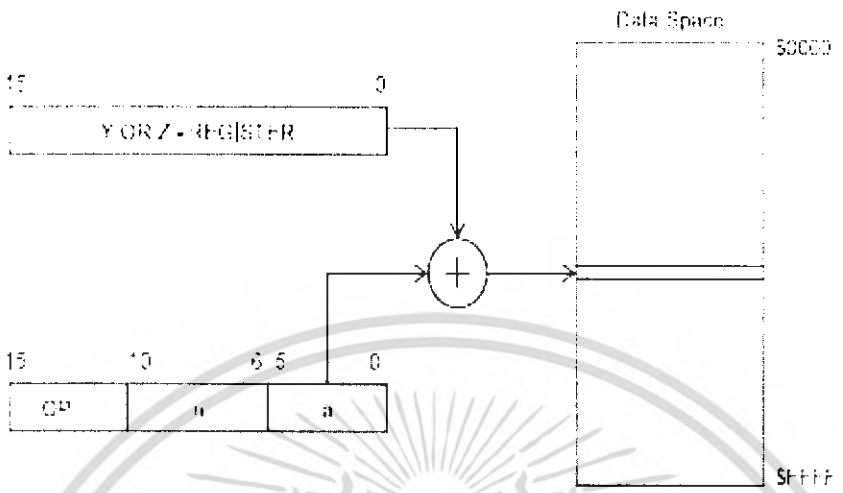
2.3.5.3.4 Data Direct



รูปที่ 2.27 แสดงการเข้าถึงข้อมูลแบบ Direct Data Addressing

ตำแหน่งหน่วยความจำขนาด 16 บิต จะเก็บไว้ใน 16 บิตล่างของรหัสคำสั่งที่มีขนาด 2 Word โดย Rd / Rr จะใช้ในการกำหนดเป็นรีจีสเตอร์ Source หรือDestination

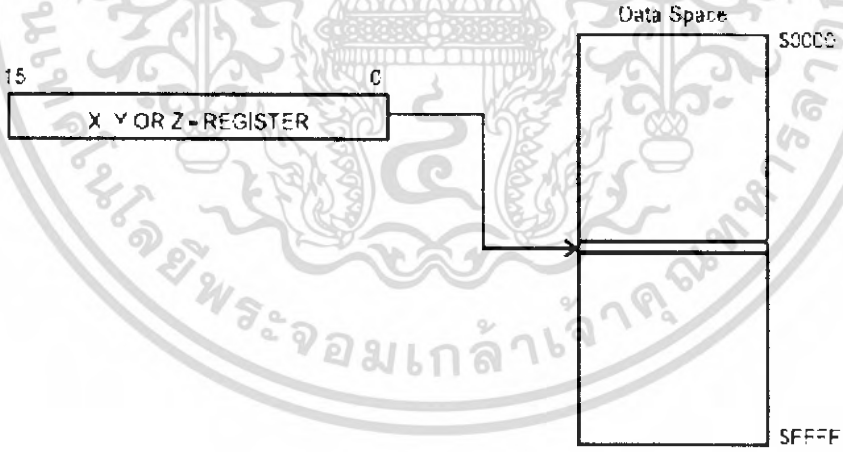
2.3.5.3.5 Data Indirect with Displacement



รูปที่ 2.28 แสดงการเข้าถึงข้อมูลแบบ Data Indirect With Displacement

ตำแหน่งของตัวกระทำคือผลบวกของรีจีสเตอร์YหรือZ กับตำแหน่งหน่วยความจำขนาด 6 บิต ที่บรรจุในรหัสคำสั่ง

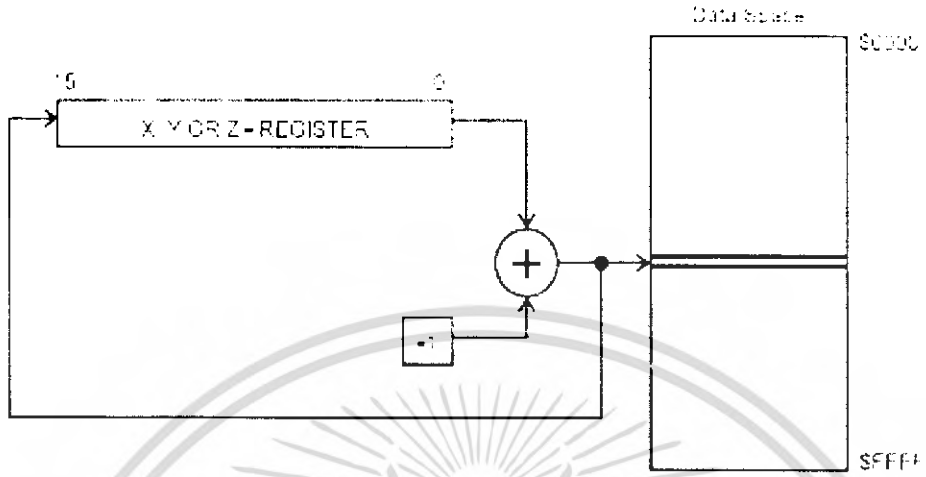
2.3.5.3.6 Data Indirect



รูปที่ 2.29 แสดงการเข้าถึงข้อมูลแบบ Data Indirect Addressing

ตำแหน่งของตัวกระทำคือค่าของรีจีสเตอร์ X, Y และ Z

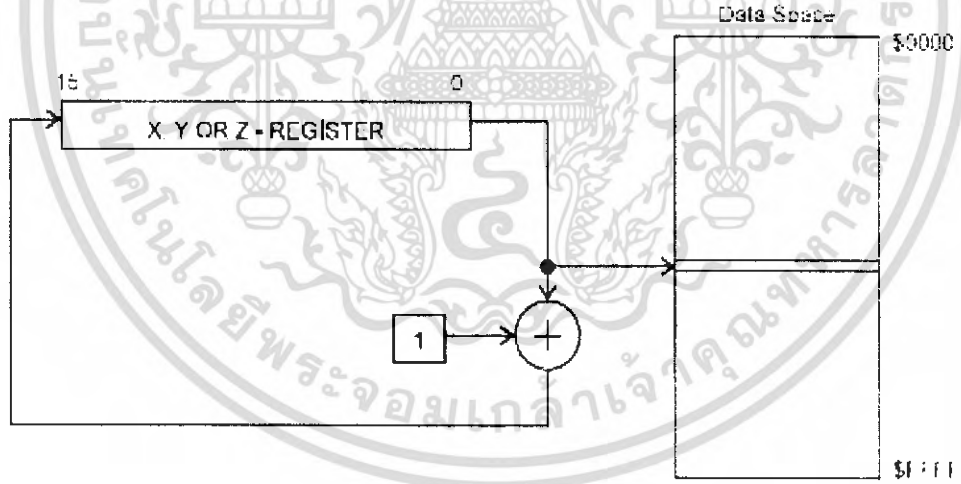
2.3.5.3.7 Data Indirect with Pre-decrement



รูปที่ 2.30 แสดงการเข้าถึงข้อมูลแบบ Data Indirect With Pre-decrement

รีจีสเตอร์ X, Y และ Z จะถูกลดค่าก่อนการทำคำสั่ง 1 ค่า ซึ่งตำแหน่งของตัวกระทำคือค่าที่อยู่ในรีจีสเตอร์ X, Y และ Z ภายหลังจากที่ลดแล้ว 1 ค่า

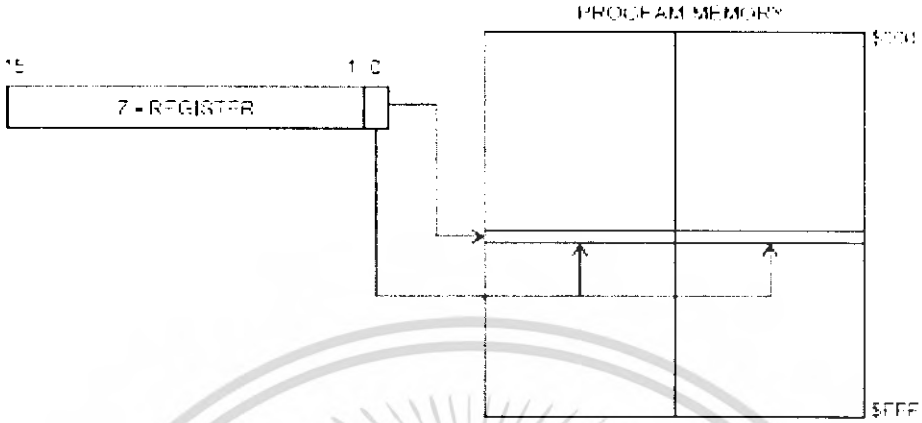
2.3.5.3.8 Data Indirect with Post-increment



รูปที่ 2.31 แสดงการเข้าถึงข้อมูลแบบ Data Indirect With Pre – Decrement

รีจีสเตอร์ X, Y และ Z จะถูกเพิ่มก่อนการทำคำสั่ง 1 ค่า ซึ่งตำแหน่งของตัวกระทำคือค่าที่อยู่ในรีจีสเตอร์ X, Y และ Z ภายหลังจากที่เพิ่มแล้ว 1 ค่า

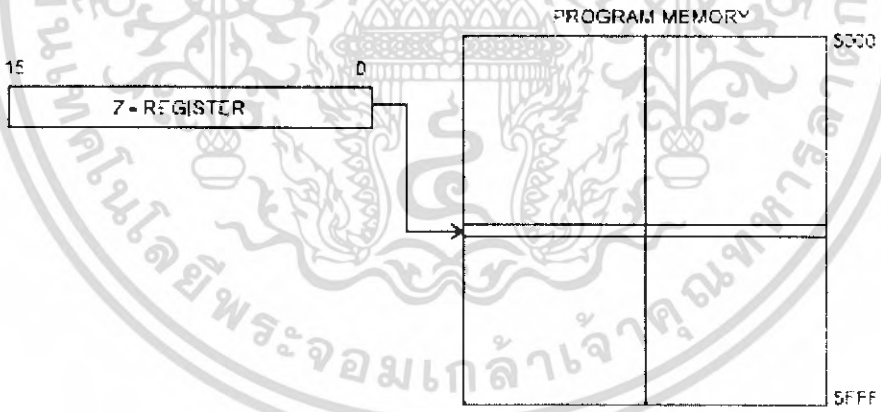
2.3.5.3.9 Constant Addressing Using the LPM Instruction



รูปที่ 2.32 แสดงการเข้าถึงข้อมูลแบบ Constant Addressing Using The LPM Instruction

ตำแหน่งหน่วยความจำที่ CPU จะกระทำคำสั่งจะถูกเก็บไว้ในรีจีสเตอร์ Z โดย 15 บิตบนของ รีจีสเตอร์ Z จะถูกใช้การกำหนดตำแหน่งของหน่วยความจำในช่วง 0 – 2K/4K ส่วน LSB บิตในรีจีสเตอร์ Z จะเป็นบิตที่ใช้เลือก Low Byte หรือ High Byte (LSB = 0 จะเป็นการเลือก Low Byte , LSB = 1 จะเป็นการเลือก High Byte)

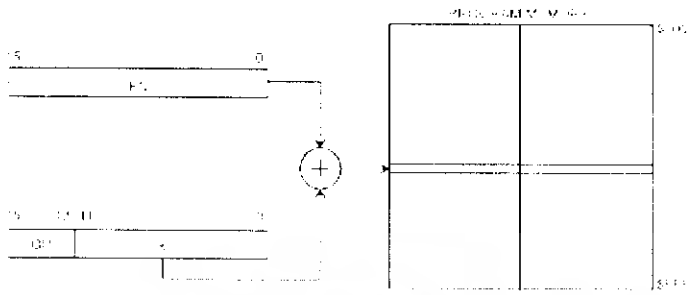
2.3.5.3.10 Indirect Program Addressing IJMP and ICALL



รูปที่ 2.33 แสดงการเข้าถึงข้อมูลแบบ Code Memory Constant Addressing

CPU จะกระทำคำสั่งที่ตำแหน่งหน่วยความจำที่บรรจุในรีจีสเตอร์ Z (นำค่าในรีจีสเตอร์ Z ใส่ลงในรีจีสเตอร์ PC)

2.3.5.3.11 Relative Program Addressing RJMP and RCALL



รูปที่ 2.34 แสดงการเข้าถึงข้อมูลแบบ Relation Program Memory Addressing

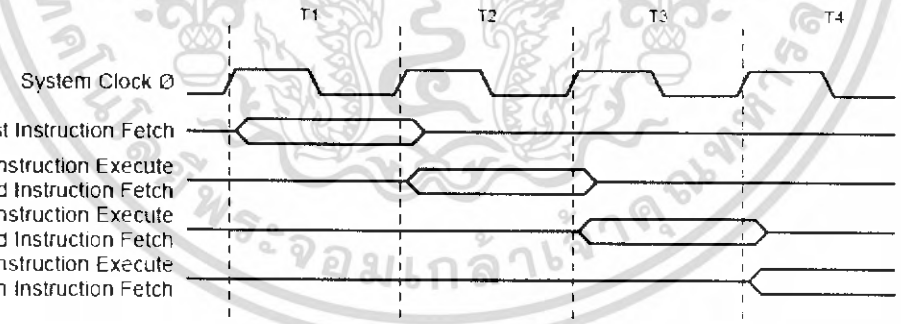
CPU จะกระโดดไปทำงานที่ตำแหน่ง PC + K + 1 โดยค่าของ K จะอยู่ระหว่าง – 2048 ถึง 2047

2.3.5.4 หน่วยความจำข้อมูลแบบ EEPROM

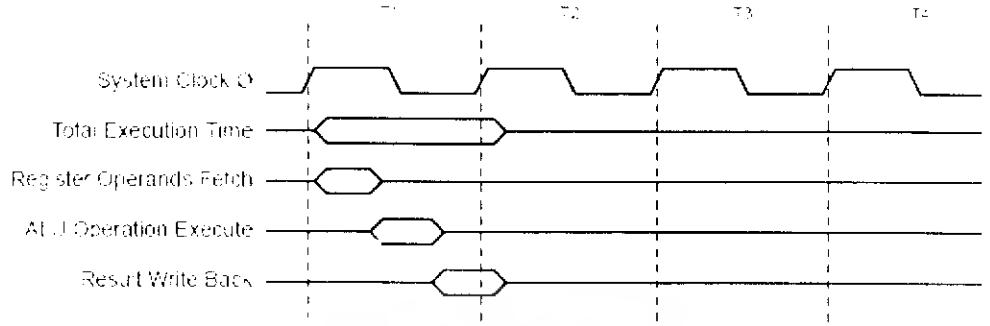
ไมโครคอนโทรลเลอร์ AT90S8515 มีหน่วยความจำแบบ EEPROM ขนาด 256/512 Byte ซึ่งสามารถอ่านและเขียนครั้งละ 1 Byte โดยสามารถทำการลบและเขียนข้อมูลใหม่ได้ 100,000 ครั้ง

คาบเวลาของการเข้าถึงข้อมูลใหม่หน่วยความจำและการปฏิบัติคำสั่ง

ในหัวข้อนี้จะอธิบายถึงคาบเวลาของการเข้าถึงข้อมูลในหน่วยความจำและคาบเวลาของการปฏิบัติคำสั่งรูปที่ 2.35 แสดงการ Fetch และ Executions แบบขนาดของ CPU ซึ่งทำให้การปฏิบัติคำสั่งได้ 1 MIPS ต่อ MHz

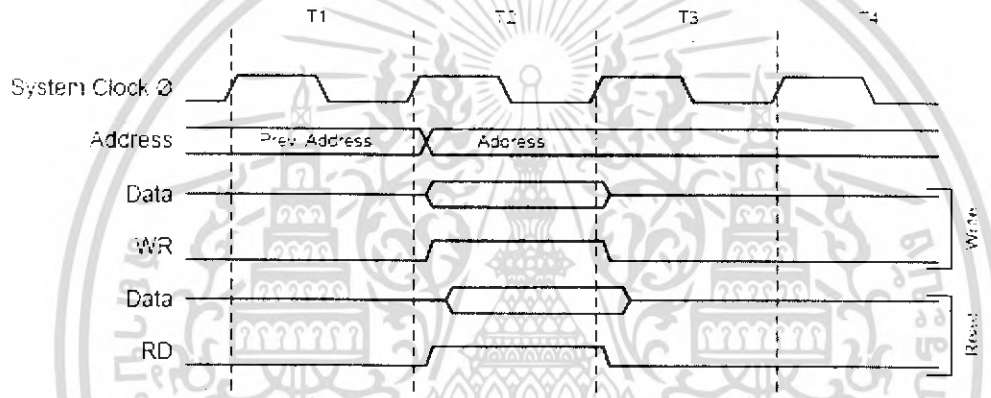


รูปที่ 2.35 แสดงคาบเวลาการทำงานของ ALU โดยใช้คาบเวลาในการปฏิบัติคำสั่งใน 1 Clock



รูปที่ 2.36 แสดงการทำงานของ ALU ใน 1 Cycle

การเข้าถึงข้อมูลใน SRAM จะใช้คาบเวลา 2 Clock ซึ่งแสดงในรูปที่ 2.37



รูปที่ 2.37 แสดงการเข้าถึงข้อมูล SRAM ภายใน CPU

2.3.6 ตำแหน่ง I/O รีจีสเตอร์สถานะและการใช้งาน EEPROM

2.3.6.1 I/O Memory

Address	Name	Function
$3F($5F)	SREG	Status Register
$3E($5E)	SPH	Stack Pointer High
$3D($5D)	SPL	Stack Pointer Low
$3B($5B)	GIMSK	General Interrupt Mask Register
$3A($5A)	GIFR	General interrupt Flag Register
$39($59)	TIMSK	Timer/Counter Interrupt Mask Register
$38($58)	TIFR	Timer/Counter Interrupt Flag Register
$35($55)	MCUCR	MCU general Control Register
$33($53)	TCCR0	Timer/Counter0 Control Register

$32($52)	TCNT0	Timer/Counter0 (8-bit)
$2F($4F)	TCCR1A	Timer/Counter1 Control Register A
$2E($4E)	TCCR1B	Timer/Counter1 Control Register B
$2D($4D)	TCNT1H	Timer/Counter0 High Byte
$2C($4C)	TCNT1L	Timer/Counter0 Low Byte
$2B($4B)	OCR1AH	Timer/Counter0 Output Compare Register A High Byte
$2A($4A)	OCR1AL	Timer/Counter0 Output Compare Register A Low Byte
$29($49)	OCR1BH	Timer/Counter0 Output Compare Register B High Byte
$28($48)	OCR1BL	Timer/Counter0 Output Compare Register B Low Byte
$25($45)	ICR1H	T/C 1 Input Capture Register High Byte
$24($44)	ICR1L	T/C 1 Input Capture Register Low Byte
$21($41)	WDTCR	Watchdog Timer Control Register
$1F($3E)	EEARH	EEPROM Address Register High Byte (AT90S8515)
$1E($3E)	EEARL	EEPROM Address Register Low Byte
$1D($3D)	EEDR	EEPROM Data Register
$1C($3C)	EECR	EEPROM Control Register
$1B($3B)	PORTA	Data Register, Port A
$1A($3A)	DDRA	Data Direction Register, Port A
$19($39)	PINA	Input Pins, Port A
$18($38)	PORTB	Data Register, Port B
$17($37)	DDRB	Data Direction Register, Port B
$16($36)	PINB	Input Pins, Port B
$15($35)	PORTC	Data Register, Port C
$14($34)	DDRC	Data Direction Register, Port C
$13($33)	PINC	Input Pins, Port C
$12($32)	PORTD	Data Register, Port D
$11($31)	DDRD	Data Direction Register, Port D
$10($30)	PIND	Input Pins, Port D
$0F($2F)	SPDR	SPI I/O Data Register

$0E($2E)	SPSR	SPI Status Register
$0D($2D)	SPCR	SPI Control Register
$0C($2C)	UDR	UART I/O Data Register
$0B($2B)	USR	UART Status Register
$0A($2A)	UCR	UART Control Register
$09($29)	UBRR	UART Baud Rate Register
$08($28)	ACSR	Analog Comparator Control and Status Register

ตารางที่ 2.1 แสดงการวางตำแหน่ง I/O ของ AT90S8515

2.3.6.2 The Status Register – SREG

รีจีสเตอร์ SREG ถูกจัดวางไว้ที่ตำแหน่ง $3F($5F)

รูปที่ 2.38 แสดงตำแหน่งรีจีสเตอร์ SREG ซึ่งถูกจัดวางไว้ที่ตำแหน่ง $3F

- **Bit 7 – I : Global Interrupt Enable** บิต I ต้องถูก SET เป็น 1 เมื่อต้องการกำหนดให้ Interrupt ทั้งหมดทำงานโดยอินเตอร์รัพท์แต่ละแหล่งจะสามารถแบ่งแยกกันกำหนดได้ในรีจีสเตอร์ GIMSK และ TIMSK แต่ถ้าบิต I ถูก CLAR เป็น 0

อินเตอร์รัพท์ทุกตัวจะถูก DISABLE โดยบิต I จะถูก Clear โดยอัตโนมัติเมื่อเกิดการอินเตอร์รับและถูก SET โดยการใช้คำสั่ง RET

- **Bit 6 – T : Bit Copy Storage**การใช้คำสั่ง BLD และคำสั่ง BST จะเป็นคำสั่งที่ใช้บิต T ทำหน้าที่เป็น Source หรือ Destination โดยบิตต่างๆ ในรีจีสเตอร์สามารถ Copy บิต T ลงในบิตของรีจีสเตอร์ได้โดยใช้คำสั่ง BST
- **Bit 5 – H : Half Carry Flag** บิต H แสดงการเกิด Half Carry Flag
- **Bit 4 – S : Sign Bit, S = N ⊕ V** เป็นบิตที่ใช้แสดงเครื่องหมาย
- **Bit 3 – V : Two's Complement** แสดงการทำ Two's Complement
- **Bit 2 – N : Negative Flag** แสดงการกระทำคำสั่งที่เป็นลบของการกระทำทางคณิตศาสตร์และลอจิก
- **Bit 1 – Z : Zero Flag** แสดงการทำคำสั่งทางคณิตศาสตร์และลอจิกที่ให้ผลเป็น 0

Bit 0 – C : Carry Flag แสดงการเกิด Carry Flag

2.3.6.3 The Stack Pointer – SP

ใน AT90S8515 จะใช้รีจีสเตอร์ขนาด 8 บิต bit 2 ตัว ในตำแหน่ง S3E และ S3D ทำหน้าที่เป็น Stack Pointer ก่อนที่จะนำไปใช้งาน

Bit	15	14	13	12	11	10	9	8	
$3E ($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

รูปที่ 2.39 แสดงตำแหน่งรีจีสเตอร์ Stack Pointer ซึ่งถูกจัดวางไว้ที่ตำแหน่ง$3E และ $3D

2.3.7 รีจีสเตอร์และการอินเตอร์รัพท์

2.3.7.1 การรีจีสเตอร์และการอินเตอร์รัพท์

ไมโครคอนโทรลเลอร์ AT90S8515 มีระบบการตอบสนองสัญญาณอินเตอร์รัพท์จาก 16 แหล่งสัญญาณ โดยได้แยกอินเตอร์รัพท์เวกเตอร์ของแต่ละอินเตอร์รัพท์ออกจากกัน ในการควบคุมการตอบสนองของอินเตอร์รัพท์แต่ละแหล่ง สามารถแยกการควบคุมได้จากบิต Enable ของอินเตอร์รัพท์นั้นๆ และบิต I ซึ่งใช้ควบคุมการอินเตอร์รัพท์ทั้งหมด โดยตำแหน่งแรกๆใน Program Memory จะเป็นตำแหน่งที่ถูกใช้เป็นพื้นที่ของอินเตอร์รัพท์เวกเตอร์และรีเซ็ท ซึ่งตารางที่ 2 แสดงอินเตอร์รัพท์เวกเตอร์ต่างๆ โดยเริ่มจากอินเตอร์รัพท์ที่มีระดับความสำคัญสูงสุด คือ RESET จนถึงอินเตอร์รัพท์ที่มีระดับความสำคัญที่สุด

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	Hardware Pin and Watchdog Reset
2	$001	INT0	External Interrupt Request 0
3	$002	INT1	External Interrupt Request 1
4	$003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	$004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	$005	TIMER1 COMPB	Timer/Counter1 Compare Match B
7	$006	TIMER1 OVF	Timer/Counter1 Overflow
8	$007	TIMER0, OVF	Timer/Counter0 Overflow
9	$008	SPI, STC	Serial Transfer Complete
10	$009	UART RX	UART, Rx Complete
11	$00A	UART UDRE	UART Data Register Empty
12	$00B	UART TX	UART, Tx Complete
13	$00C	ANA_COMP	Analog Comparator

ตารางที่ 2.2 ตารางอินเตอร์รัพท์เวกเตอร์

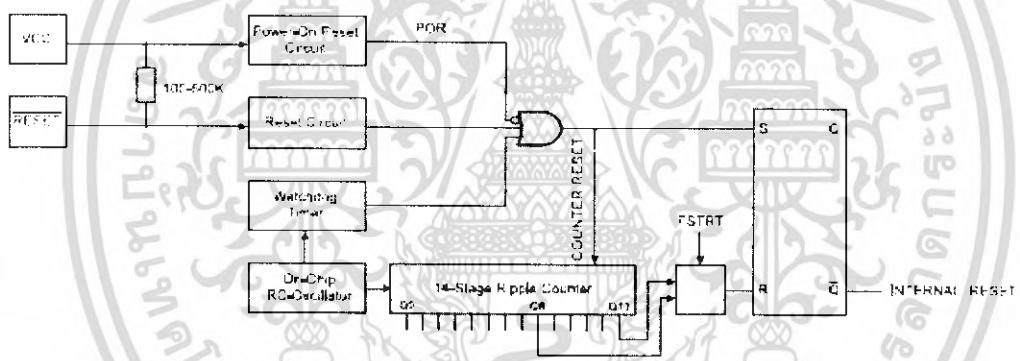
2.3.7.2 สัญญาณรีเซ็ท

ไมโครคอนโทรลเลอร์ AT90S8515 มีสัญญาณรีเซ็ท 3 แหล่ง

- Power on Reset ไมโครคอนโทรลเลอร์ จะรีเซ็ทเมื่อมีการจ่ายไฟให้กับขา Vcc และ GND

  External Reset ไมโครคอนโทรลเลอร์จะรีเซ็ท เมื่อมีสัญญาณลอจิก Low เข้ามาที่ขารีเซ็ทเป็นระยะเวลามากกว่า 2 คาบเวลาของสัญญาณ XTAL
- Watchdog Reset ไมโครคอนโทรลเลอร์จะรีเซ็ทเมื่อถึงคาบเวลาของ Watchdog และ Watchdog จะต้องถูกกำหนดให้ทำงาน

ระหว่างที่เกิดการรีเซ็ท รีจีสเตอร์ทั้งหมดจะถูกกำหนดให้ค่าเริ่มต้น และโปรแกรมจะเริ่มทำงานที่ตำแหน่ง $0000 โดยคำสั่งที่ตำแหน่ง $0000 จะต้องเป็นคำสั่ง RJMP แต่ถ้าโปรแกรมไม่มีการกำหนดให้มีการใช้อินเตอร์รัพท์พื้นที่ส่วนที่เป็นอินเตอร์รัพท์เวกเตอร์จะถูกใช้เป็นพื้นที่ของโปรแกรม



รูปที่ 2.40 แสดงโครงสร้างของวงจรรีเซ็ท

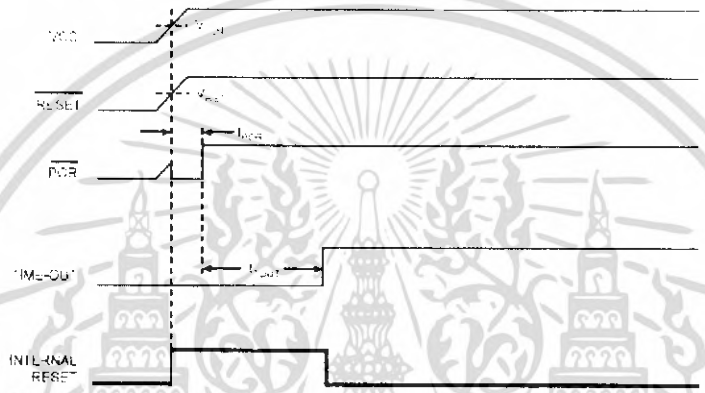
Symbol	Parameter	Min	Typ	Max	Units
V_{POT}	Power-On Reset Threshold Voltage	1.8	2	2.2	V
V_{RST}	RESET Pin Threshold Voltage		$V_{CC}/2$		V
t_{POR}	Power-On Reset Period	2	3	4	ms
t_{TOUT}	Reset Delay Time-Out Period FSTRT Unprogrammed	11	16	21	ms
t_{TOUT}	Reset Delay Time-Out Period FSTRT Programmed	1.0	1.1	1.2	ms

ตารางที่ 2.3 คุณลักษณะของสัญญาณรีเซ็ท

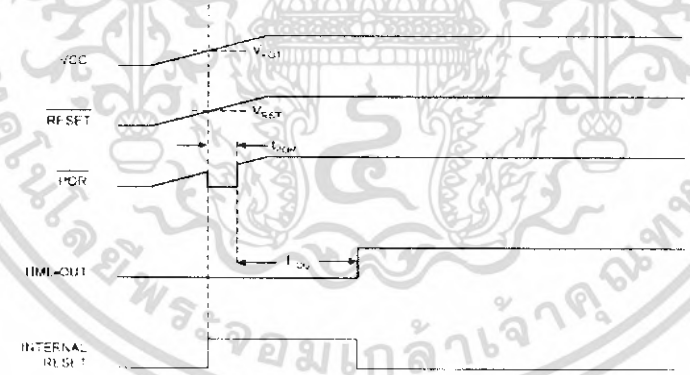
2.3.7.3 Power On Reset

วงจรของ Power On Reset (POR) ถูกสร้างขึ้นเพื่อให้แน่ใจว่า MCU จะไม่ทำงานถ้าระดับของแรงดันไฟเลี้ยงวงจรยังไม่ถึงระดับที่จะทำให้ระบบการทำงานภายในของ MCU ทำงานได้อย่างถูกต้อง

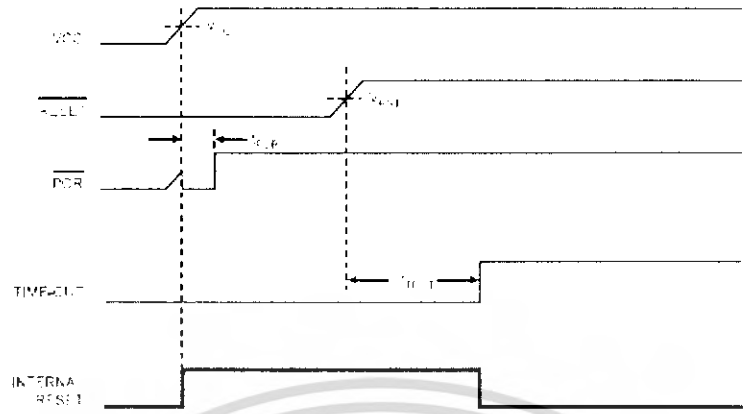
โดยบิต FSTRT ใช้ในการเลือกคาบเวลา TIME – OUT ซึ่งเมื่อบิต FSTRT ถูกโปรแกรมจะทำให้ช่วงเวลา TIME – OUT มีคาบเวลาน้อยลง ซึ่งมีประโยชน์อย่างมากเมื่อใช้ OSC ที่เป็น Ceramic Resonator



รูปที่ 2.41 แสดง MCU Start – Up, RESET Tied to Vcc Rapidly Rising Vcc



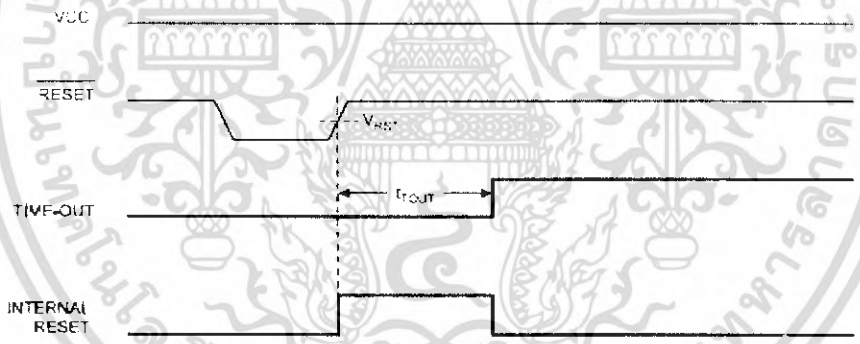
รูปที่ 2.42 แสดง MCU Start – Up, RESET Tied to Vcc Slowly Rising Vcc



รูปที่ 2.43 แสดง MCU Start – Up, RESET Controlled Externally

2.3.7.4 External Reset

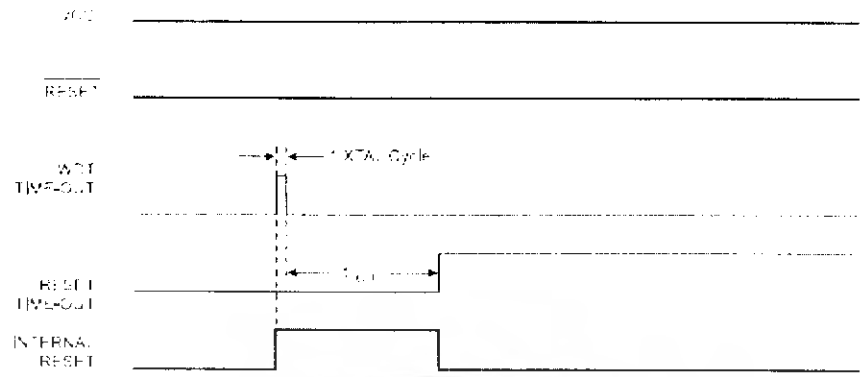
การรีเซ็ทจากภายนอกจะเกิดขึ้นโดยการให้ลอจิก LOW ที่ขา Reset อย่างน้อย 2 Clock ของสัญญาณนาฬิกา ซึ่งเมื่อสัญญาณที่ขอบขาขึ้นถึงระดับ Threshold Voltage จะทำให้วงจร Time Delay เริ่มทำงาน ซึ่ง CPU จะถูกรีเซ็ทเมื่อคาบเวลาของ Time Delay มากกว่าคาบเวลา Time – Out



รูปที่ 2.44 แสดงคาบเวลาของการรีเซ็ทจากภายนอก

2.3.7.5 Watchdog Reset

เมื่อ WTD TIME – OUT เริ่มกำเนิดสัญญาณ PLUSE ที่มีคาบเวลาเท่ากับ Clock ของสัญญาณนาฬิกา โดยช่วงขอบขาลงของ Pulse จะทำให้ระบบการหน่วงเวลาเริ่มทำงานจนหมดช่วงเวลาของ TIME – OUT จึงทำให้ CPU รีเซ็ท



รูปที่ 2.45 แสดง Watchdog Reset During Operation

2.3.7.6 Interrupt Handling

ไมโครคอนโทรลเลอร์ AT90S8515 มีรีจีสเตอร์ที่ใช้งานอินเตอร์รัพท์ 2 ตัว คือ GIMSK และ TIMSK ที่ใช้บรรจุบิตควบคุมและแสดงบิตสถานะของอินเตอร์รัพท์ต่างๆ

เมื่อมีอินเตอร์รัพท์เกิดขึ้นจะทำให้บิต I ซึ่งเป็นบิตที่ใช้ควบคุมอินเตอร์รัพท์ถูกเคลียร์เป็น 0 และจะทำให้อินเตอร์รัพท์ทั้งหมดจะถูก Disable ผู้ใช้สามารถเซ็ทบิต I ให้เป็น 1 เมื่อต้องการ Enable Interrupt

เมื่อเกิดอินเตอร์รัพท์จะทำให้ CPU กระโดดไปทำโปรแกรมบริการอินเตอร์รัพท์ และเคลียร์ Interrupt Flag (ของอินเตอร์รัพท์ที่เกิดขึ้น) ให้เป็น 0 หรือผู้ใช้สามารถเคลียร์ Interrupt Flag ได้โดยการเขียนลอจิก 1 ลงใน Flag ที่ต้องการเคลียร์

2.3.7.7 The General Interrupt Mask Register – GIMSK

Bit	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1	INT0	-	-	-	-	-	-	GIMSK
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial value	0	0	0	0	0	0	0	0	

รูปที่ 2.46 แสดงตำแหน่ง The General Interrupt Mask Register – GIMSK ซึ่งถูกจัดวางไว้ที่ตำแหน่ง $3B

- **Bit 7 – INT1 : External Interrupt I Enable** เป็นบิตที่ใช้ Enable การอินเตอร์รัพท์จากภายนอกที่ตำแหน่งขา INT1 โดยเมื่อบิตนี้ถูกเซ็ทเป็น 1 จะเป็นการยอมให้เกิดการอินเตอร์รัพท์จากขา INT1 (บิต I จะต้องถูกเซ็ทเป็น 1) โดยผู้ใช้สามารถเลือกลักษณะของสัญญาณที่เข้ามาที่ขา INT1 ที่มีผลทำให้เกิดอินเตอร์รัพท์ได้จากบิต ISC00 และ ISC01 ซึ่งอินเตอร์รัพท์เวกเตอร์จะอยู่ที่ตำแหน่ง $002

Bit 6 – INT0 : Exterrupt Request 0 Enable เป็นบิตที่ใช้ Enable การอินเตอร์รัพท์จากภายนอกที่ตำแหน่งขา INT0 โดยเมื่อบิตนี้ถูกเซ็ทเป็น 1 จะเป็นการยินยอมให้เกิดการอินเตอร์รัพท์จากขา INT0 ที่มีผลทำให้เกิดการอินเตอร์รัพท์ได้จากบิต ISC00 และ ISC01 ซึ่งอินเตอร์รัพท์เวกเตอร์จะอยู่ที่ตำแหน่ง S001

Bit 5 – Res : Reserved bits ในไมโครคอนโทรลเลอร์ AT90S8515 สงวนบิตไว้เป็น 0

2.3.7.8 The General Interrupt Flag Register – GIFR

Bit	7	6	5	4	3	2	1	0	
$3A ($5A)	INTF1	INTF0	-	-	-	-	-	-	GIFR
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial value	0	0	0	0	0	0	0	0	

รูปที่ 2.47 แสดงตำแหน่ง The General Interrupt Flag Register – GIFR ซึ่งถูกจัดวางไว้ที่ตำแหน่ง S3A

- **Bit 7 – INTF : External Interrupt Flag1**เป็นบิตแสดงสถานะการเกิดอินเตอร์รัพท์จาก :External Interrupt1 เมื่อมีสัญญาณตามที่กำหนดในบิต ISC11 และ ISC10 เข้ามาที่ขา INTF1 ถูกเซ็ทเป็น 1 ซึ่งถ้าบิต I ในรีจีสเตอร์ SREG และบิต INT1 จะส่งผลให้เกิดอินเตอร์รัพท์และ CPU กระโดดไปทำงานในตำแหน่ง $002 และบิต INTF1 ถูกเคลียร์เป็น 0 โดยอัตโนมัติ หรือสามารถเคลียร์ได้โดยการเขียนลอจิก 1 ลงในบิต INTF1
- **Bit 6 – INTF : External Interrupt Flag0**เป็นบิตที่ใช้แสดงสถานะการเกิดอินเตอร์รัพท์จาก INT0 เมื่อมีสัญญาณตามที่กำหนดในบิต ISC00 และ ISC01 เข้ามาที่ขา INTO0 ซึ่งทำให้ INTF0 ถูกเซ็ทเป็น 1 ซึ่งถ้า I ในรีจีสเตอร์ SREG และบิต INT0 ถูกเคลียร์เป็น 0 โดยอัตโนมัติ หรือสามารถเคลียร์ได้โดยการเขียนลอจิก 1 ลงในบิต INTF0
- **Bit 5 – Res : Reserved bits** ใน AT90S8515 ถูกสงวนไว้โดยปกติบิตนี้จะเป็น 0

2.3.8 พอร์ทอินพุต/เอาท์พุต

2.3.8.1 พอร์ท A

เป็นพอร์ท 2 ทิศทางขนาด 8 บิต โดยมีหน่วยควบคุมต่างๆของพอร์ท คือ รีจีสเตอร์ PORTA REGISTER อยู่ที่ตำแหน่ง $1B($3B), รีจีสเตอร์ DDRA(DATA DIRECTION REGISTER) อยู่ที่ตำแหน่ง $19($39) โดย PINA จะสามารถอ่านได้อย่างเดียว ไม่สามารถเขียนข้อมูลลงได้ ในขณะที่ PORTA และ DDRA สามารถอ่านและเขียนได้ โดยแต่ละขาสัญญาณของ

พอร์ท A จะสามารถกำหนดให้ค่าความต้านทาน Pull Low ในขณะที่ความต้านทาน Pull Up ภายใน ACTIVE จะทำให้ MCU จ่ายกระแสออกมาภายนอก

พอร์ท A จะถูกนำไปใช้งานอีกอย่างหนึ่งคือ เป็นขาสัญญาณอินพุตของวงจร ANALOG TO DIGITAL โดยถ้าบางขาของพอร์ท A ถูกใช้งานเป็นขาเอาท์พุตของ ANALOG TO DIGITAL ผู้ใช้จะต้องไม่เปลี่ยนแปลงสภาวะการทำงานของพอร์ท A ในขณะที่วงจร ANALOG TO DIGITAL กำลัง Conversion สัญญาณ ซึ่งอาจจะทำให้การทำการแปลงสัญญาณถูกขัดจังหวะ จะมีผลทำให้การเปลี่ยนแปลงสัญญาณผิดพลาด

การทำงานในโหมดประหยัดพลังงานจะทำให้ SCHMITT TRIGGER ของส่วนเอาท์พุตถูกตัดออก

รูปที่ 2.48 แสดงตำแหน่งรีจีสเตอร์ที่ควบคุมพอร์ท A

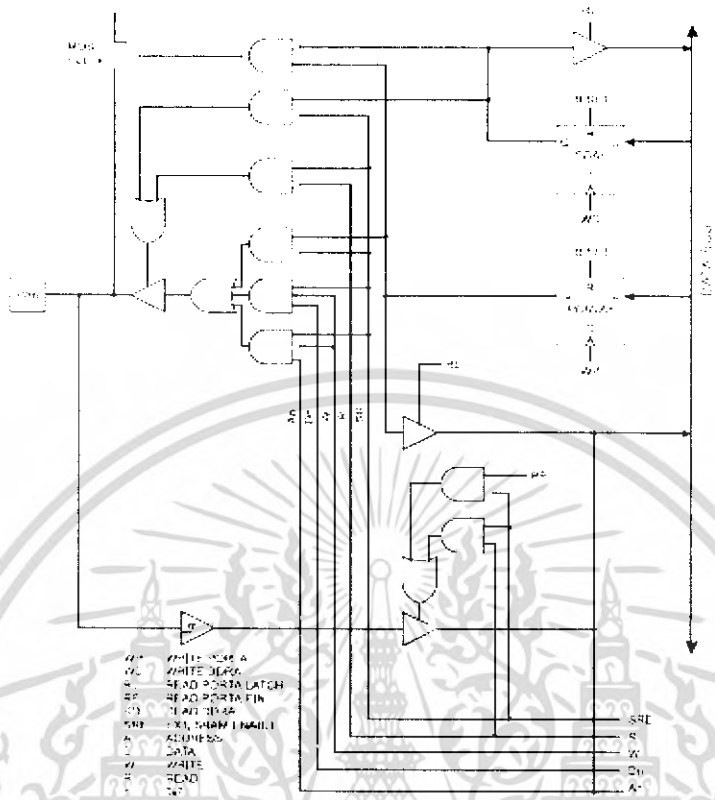
PINA ไม่ใช้รีจีสเตอร์ ซึ่งตำแหน่งที่กำหนดจะเป็นตำแหน่งของแต่ละขาของพอร์ท โดยเมื่ออ่าน PORT A จะเป็นการอ่านค่าที่ LATCH ส่วนการอ่านค่า PINA จะเป็นการอ่านค่าจริงๆของขาพอร์ท

การใช้งาน พอร์ท A เป็นขาอินพุตและเอาท์พุต

บิต DDAn ในรีจีสเตอร์ DDRA จะเป็นบิตที่ใช้ในการกำหนดทิศทางของแต่ละพอร์ท ถ้าบิต DDAn ถูกเซ็ทเป็น 1 จะทำให้ขาของพอร์ทนั้นเป็นเอาท์พุต แต่เมื่อ DDAn ถูกเคลียเป็น 0 จะทำให้ขานั้นถูกกำหนดเป็นอินพุต ซึ่งสภาวะของขาต่างๆ แสดงดังตารางที่ 4

DDAn	PORTAn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)
0	1	Input	Yes	PAn will source current if ext. pulled low.
1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	Push-Pull One Output

ตารางที่ 2.4 แสดง **DDRn Effect on PORTA Pins**



รูปที่ 2.49 แสดงโครงสร้างของพอร์ท A (PIN PA0 – PA7)

2.3.8.2 พอร์ท B

พอร์ท B เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยมีรีจีสเตอร์ที่ใช้ควบคุมพอร์ท B อยู่ 3 ตัว คือ รีจีสเตอร์ PORTB อยู่ที่ตำแหน่ง $18($36), รีจีสเตอร์ DDRB อยู่ที่ตำแหน่ง $17($37) และ PINB อยู่ที่ตำแหน่ง $16($36) โดยพอร์ท A แต่ละขาสามารถแยกกำหนดให้มีความต้านทาน Pull Up ได้ตามต้องการ ซึ่งในแต่ละขาสามารถรับกระแส (SINK CURRENT) ได้ 20 mA โดยขาของพอร์ท B สามารถใช้งานเป็นฟังก์ชั่นอื่นๆ ได้ดังตารางที่ 2.5

Port Pin	Alternate Functions
PB0	T0 (Timer/Counter 0 external counter input)
PB1	T1 (Timer/Counter 1 external counter input)
PB2	AIN0 (Analog comparator positive input)
PB3	AIN1 (Analog comparator negative input)
PB4	$\overline{SS}$ (SPI Slave Select input)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB7	SCK (SPI Bus Serial Clock)

ตารางที่ 2.5 ตารางการใช้งานฟังก์ชั่นอื่นๆของพอร์ท

The Port B Data Register - PORTB

Bit	7	6	5	4	3	2	1	0	
$18 ($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

The Port B Data Direction Register - DDRB

Bit	7	6	5	4	3	2	1	0	
$17 ($37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

The Port B Input Pins Address - PINB

Bit	7	6	5	4	3	2	1	0	
$16 ($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial value	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

รูปที่ 2.50 แสดงตำแหน่งรีจีสเตอร์ที่ควบคุมพอร์ท B

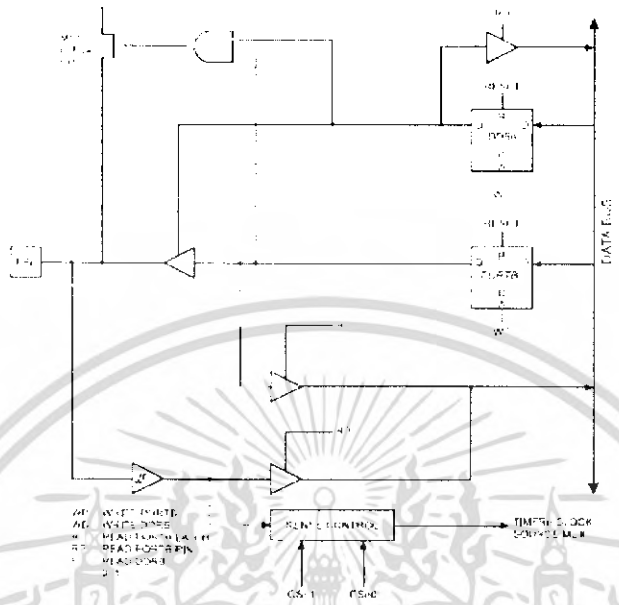
DDBn	PORTBn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)
0	1	Input	Yes	PBn will source current if ext. pulled low
1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	Push-Pull One Output

ตารางที่ 2.6 ตารางแสดงการใช้งานพอร์ท B เป็นพอร์ทเอาท์พุตและอินพุต

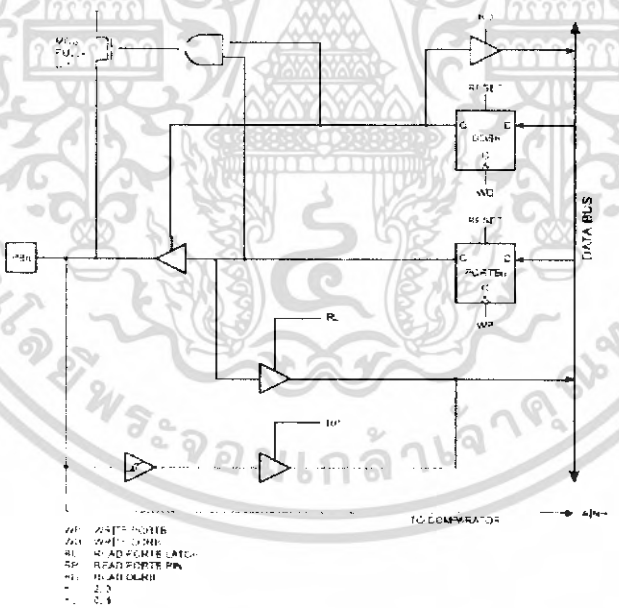
ฟังก์ชั่นอื่นๆที่ใช้ขาสัญญาณของ PORT B

- SCK – PORTB,Bit7 เป็นขา CLOCK ที่ใช้งานในส่วนของการสื่อสารแบบ SPI
- MISO – PORTB,Bit6 เป็นขารับข้อมูลการสื่อสารแบบ SPI
- MOSI – PORTB,Bit5 เป็นขาส่งข้อมูลในการสื่อสารแบบ SPI
- SS – PORTB,Bit4 ใช้เป็นขาควบคุมการทำงานในโหมด SPI
- ANI1 – PORTB,Bit3เป็นขาอินพุต 1 ของการทำงานในส่วนของ Analog Compare
- ANI0 – PORTB,Bit2เป็นขาอินพุต 0 ของการทำงานในส่วนของ Analog Compare
- T1 – PORTB,Bit1 เป็นขาอินพุตที่ใช้ในส่วนของ Timer 1
- T0 – PORTB,Bit0 เป็นขาอินพุตที่ใช้ในส่วนของ Timer 0

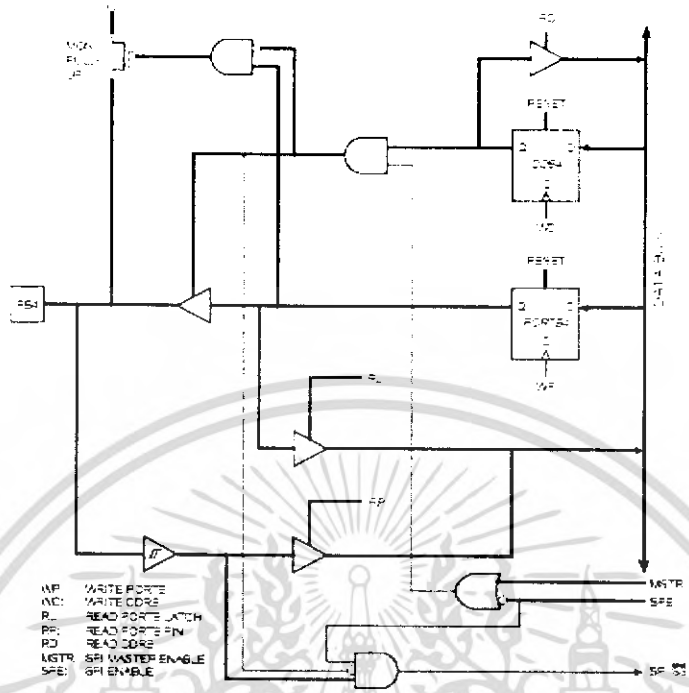
โครงสร้างของ PORT B



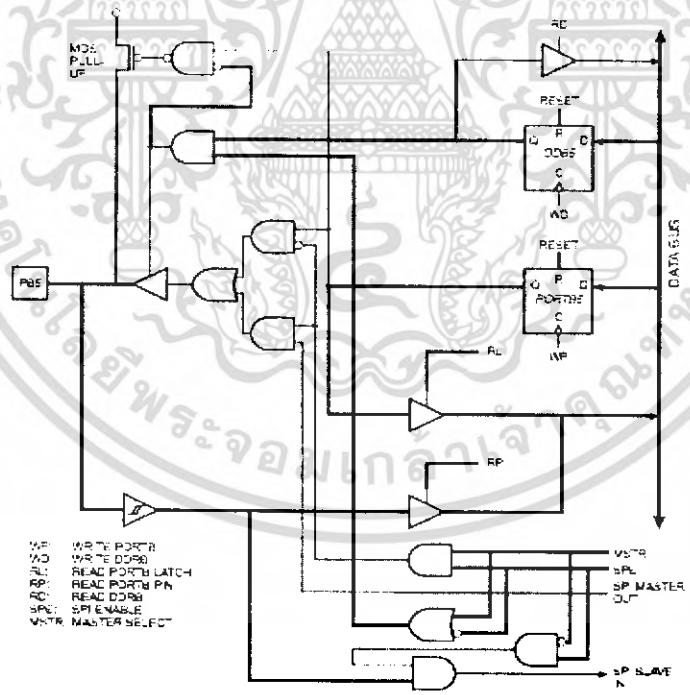
รูปที่ 2.51 แสดงโครงสร้างของพอร์ท B (Pins PB0 และ PB1)



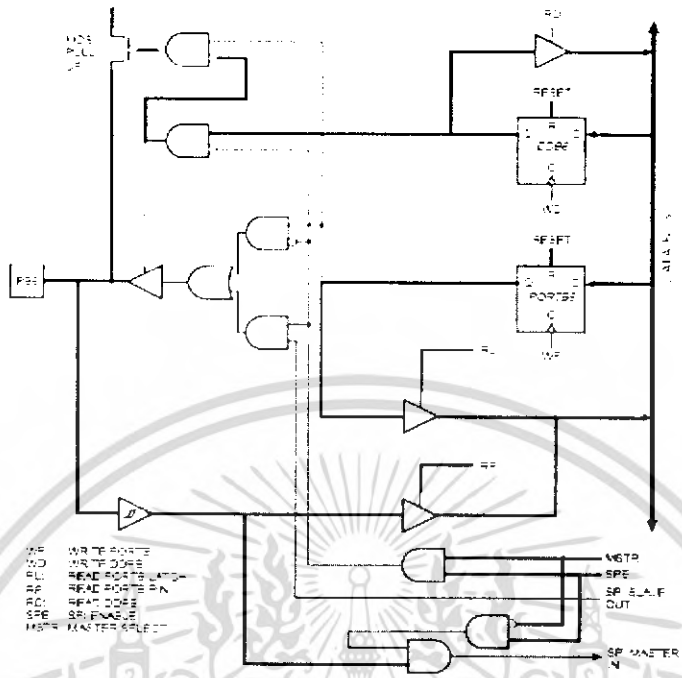
รูปที่ 2.52 แสดงโครงสร้างของพอร์ท B (PB2 และ PB3)



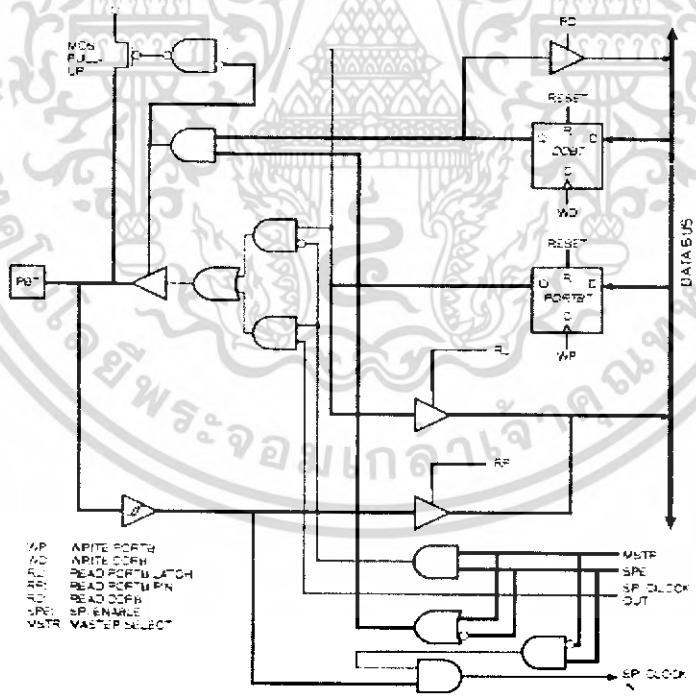
รูปที่ 2.53 แสดงโครงสร้างของพอร์ท B (PB4)



รูปที่ 2.54 แสดงโครงสร้างของพอร์ท B (PB5)



รูปที่ 2.55 แสดงโครงสร้างของพอร์ท B (PB6)



รูปที่ 2.56 แสดงโครงสร้างของพอร์ท B (PB7)

2.3.8.3 พอร์ท C

พอร์ท C เป็นพอร์ท 2 ทิศทางขนาด 8 บิต โดยมีหน่วยควบคุมการทำงานของพอร์ทจำนวน 3 หน่วย คือ PORTC (Data – Register – PORTC) อยู่ที่ตำแหน่งของหน่วยความจำ$15($35), DDRC(Data Direction Register PORTC) อยู่ที่ตำแหน่งหน่วย $14($34) และ PINC (Port C Input Pin – PINC) อยู่ที่ตำแหน่ง $13($33) โดน PINC จะสามารถอ่านได้อย่างเดียว ในขณะที่ PORTC และ DDRC จะสามารถทั้งอ่านและเขียนได้โดยแต่ละขาของพอร์ท C สามารถแยกการกำหนดความต้านทาน Pull Up ได้ ในขณะที่พอร์ท C แต่ละขาสามารถรับกระแส (SINK CURRENT) ได้ 20mA โดยถ้าภายในกำหนดให้มีความต้านทาน Pull Upและภายนอกมีความต้านทาน Pull Low จะทำให้ MCU จ่ายกระแสออกภายนอก

The Port C Data Register - PORTC

Bit	7	6	5	4	3	2	1	0	
$15 ($35)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

The Port C Data Direction Register - DDRC

Bit	7	6	5	4	3	2	1	0	
$14 ($34)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

The Port C Input Pins Address - PINC

Bit	7	6	5	4	3	2	1	0	
$13 ($33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial value	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

รูปที่ 2.57 แสดงตำแหน่งรีจีสเตอร์ที่ควบคุมพอร์ท C

PINC ไม่ใช่ REGISTER เพราะฉะนั้นตำแหน่งที่กำหนดจะเป็นตำแหน่งของแต่ละค่าของพอร์ท C โดยถ้าเป็นการอ่านของพอร์ท C จะเป็นการอ่านที่ LATCH ในขณะที่การอ่านค่าจาก PINC จะเป็นการอ่านค่าจริงของพอร์ท C

Port C as General Digital I/O

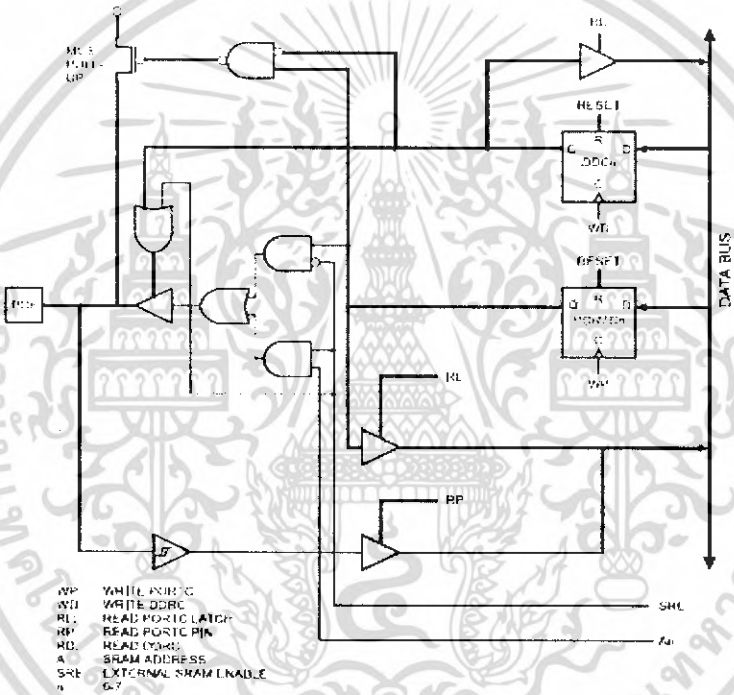
ทั้ง 8 บิตของพอร์ท C จะมีลักษณะเหมือนกัน โดย PCn, แทนพอร์ทต่างๆของพอร์ท C และ DDCn เป็นบิตที่ใช้ควบคุมทิศทางของขานั้นๆ โดยถ้า DDCn มีการเซ็ทเป็น 1 แสดงว่าเป็นขาเอาท์พุต แต่ถ้า DDCn ถูกเคลียร์ให้เป็น 0 จะเป็นการกำหนดให้ขานั้นๆเป็นอินพุต ถ้าต้องการให้ MOS Pull Up เกิดการ ACTIVE จะต้องเซ็ทให้ PORTCn มีค่าเป็น 1 และถ้าไม่ต้องการให้ MOS Pull Up เกิดการ ACTIVE จะเคลียร์ให้ PORTCn เป็น 0 โดยลักษณะการกำหนดการใช้งานของแต่ละขาดังตารางที่ 2.7

DDCn	PORTCn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)
0	1	Input	Yes	PCn will source current if ext. pulled low
1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	Push-Pull One Output

ตารางที่ 2.7 ตารางแสดงการใช้งานพอร์ท C

ฟังก์ชั่นอื่นๆ ของ PORT C

โดยเมื่อบิต AS2 ในรีจีสเตอร์ ASSR ถูกเซ็ทให้เป็น 1 จะเป็นการกำหนดให้มีการรับ CLOCK จากภายนอกของ TIMER/COUNTER2 ที่ตำแหน่งขา PC6, PC7



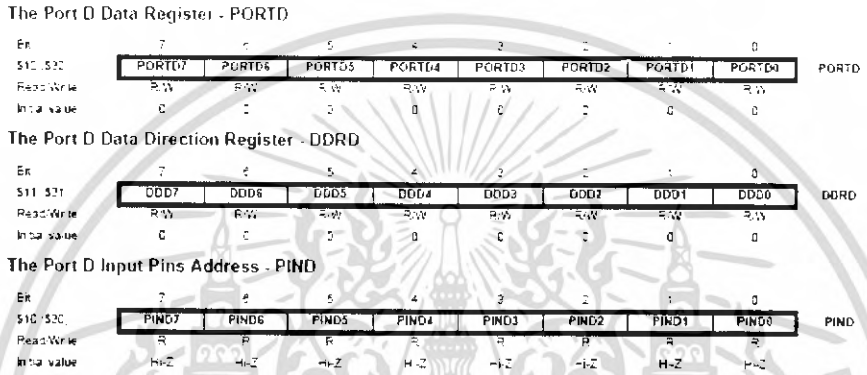
รูปที่ 2.58 แสดงโครงสร้างของ PORT C

2.3.8.4 พอร์ท D

พอร์ท D เป็นพอร์ท 2 ทิศทางขนาด 8 บิต ที่มีหน่วยควบคุมการทำงาของพอร์ท คือ PORTD (DATA Register – PORTD) อยู่ที่ตำแหน่ง $12($30) โดย PIND สามารถอ่านได้อย่างเดียว ในขณะที่ PORTD และ DDRD สามารถทั้งอ่านและเขียน โดยพอร์ท D สามารถรับกระแสได้ 20mA ซึ่งแต่ละขาของพอร์ท D สามารถเลือกฟังก์ชั่นการทำงานอื่นๆ ได้อีกดังตารางที่ 2.8

Port Pin	Alternate Function
PD0	RDX (UART Input line)
PD1	TDX (UART Output line
PD2	INT0 (External interrupt 0 input)
PD3	INT1 (External interrupt 1 input)
PD5	OC1A (Timer/Counter1 Output compare A match output)
PD6	$\overline{WR}$ (Write strobe to external memory)
PD7	$\overline{RD}$ (Read strobe to external memory)

ตารางที่ 2.8 ตารางฟังก์ชั่นอื่นๆ ของพอร์ท **D**



รูปที่ 2.59 แสดงตำแหน่งรีจีสเตอร์ที่ควบคุมพอร์ท D

PIND ไม่ใช่รีจีสเตอร์ ฉะนั้นตำแหน่งที่กำหนดจะเป็นตำแหน่งจริงของขาพอร์ท โดยเมื่ออ่านค่าของ PORTD และ DDRD จะเป็นการอ่านค่าที่ LATCH ส่วนการอ่านค่าของ PIND จะเป็นการอ่านค่าจริงๆ ที่เกิดขึ้นที ขาของพอร์ท D

การใช้งานของพอร์ท D เป็นพอร์ทอินพุตและเอาท์พุต

โดย PDn แทนขาใดๆ ของพอร์ท D และ DDRn เป็นตำแหน่งบิตในรีจีสเตอร์ DDRD ซึ่งรีจีสเตอร์ DDRD ทำหน้าที่ควบคุมทิศทางของพอร์ท D โดยถ้าเซ็ทให้เป็น 1 จะเป็นการกำหนดให้พอร์ททำงานเป็นเอาท์พุตแต่ถ้าเคลียร์ให้เป็น 0 จะกำหนดให้พอร์ททำงานเป็น อินพุต

DDDn	PORTDn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)
0	1	Input	Yes	PDn will source current if ext. pulled low
1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	Push-Pull One Output

ตารางที่ 2.9 ตารางการใช้งานพอร์ท **D**

การทำงานฟังก์ชั่นอื่นของพอร์ท D

- **RD – Port D Bit7** เป็นพอร์ทสำหรับแจ้งการทำงานกับ External SRAM ให้อ่านข้อมูล

- **WR – Port D Bit6** เป็นพอร์ทสำหรับแจ้งการทำงานกับ External SRAM ให้เขียนข้อมูล

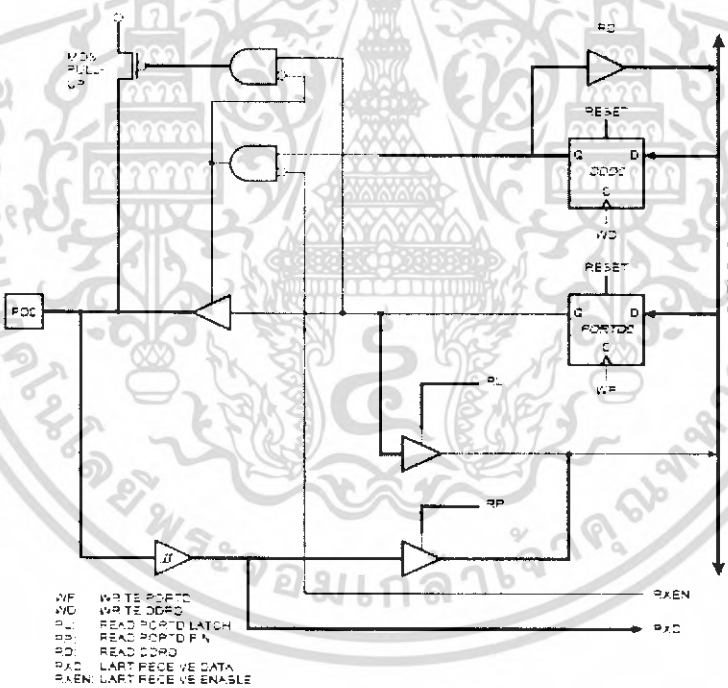
  OC1A – PORTD,BIT5 ขา OC1A หรือบิต 5 ของพอร์ท D สามารถถูกนำไปใช้งานเป็นขาเอาท์พุตของ Compare Output Match ของ Timer/Counter1

- **INT1 – PORTD Bit3**เป็นขาที่รับเอาสัญญาณอินเตอร์รัพท์จากภายนอก ผ่านทางพอร์ท INT1

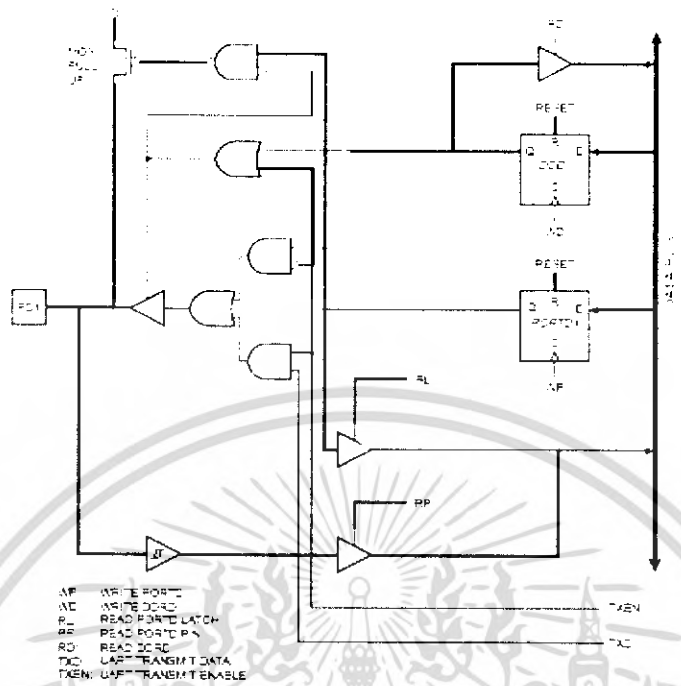
  INT0 – PORTD Bit2 เป็นขาที่รับเอาสัญญาณอินเตอร์รัพท์จากภายนอก ผ่านทางพอร์ท INT0

- **TXD – PORTD Bit1** เป็นขาที่ใช้ส่งข้อมูล สำหรับ UART

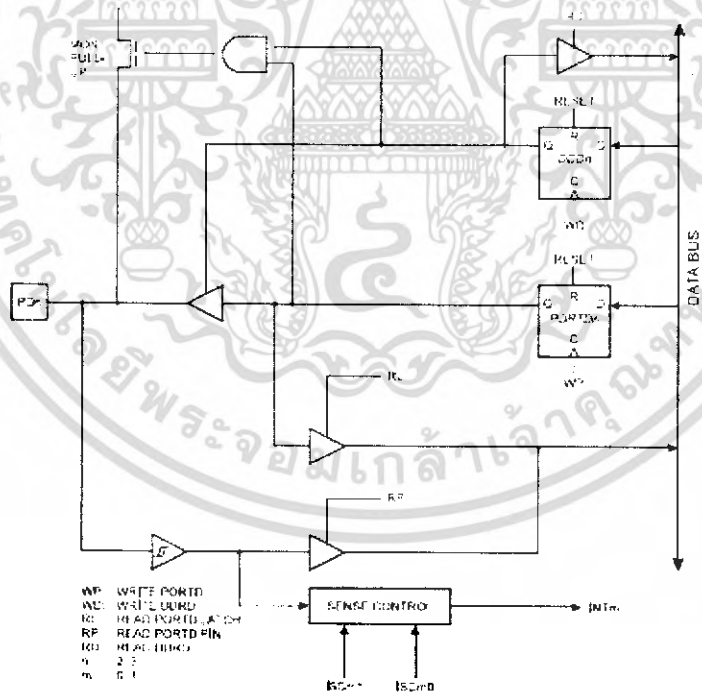
- **RXD – PORTD Bit0** เป็นขาที่ใช้รับข้อมูล สำหรับ UART



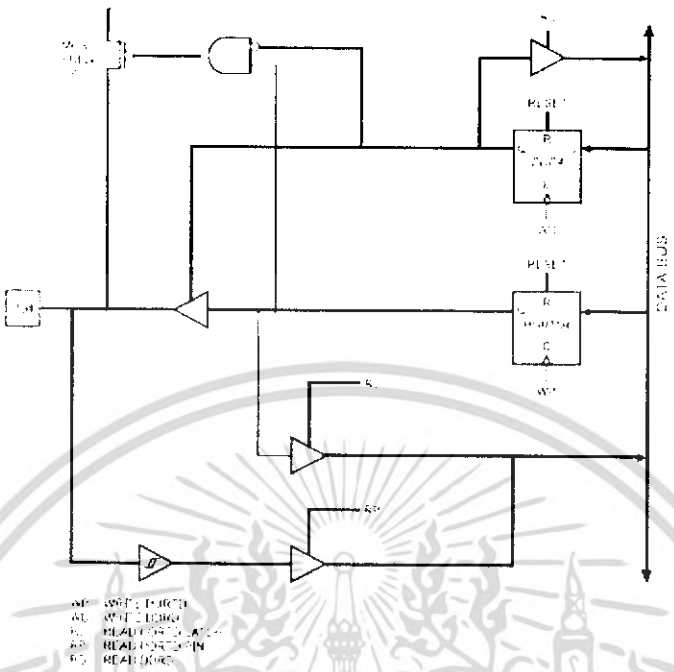
รูปที่ 2.60 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PDO)



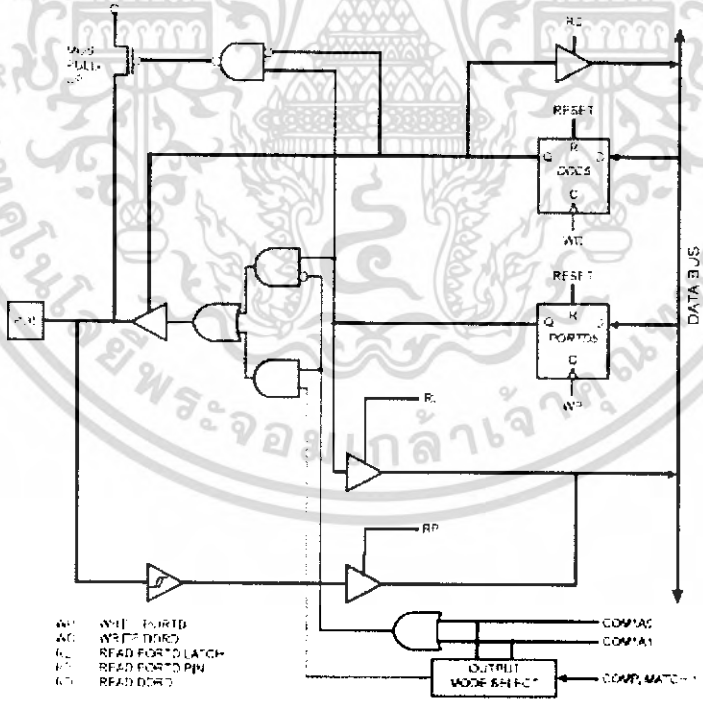
รูปที่ 2.61 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PD1)



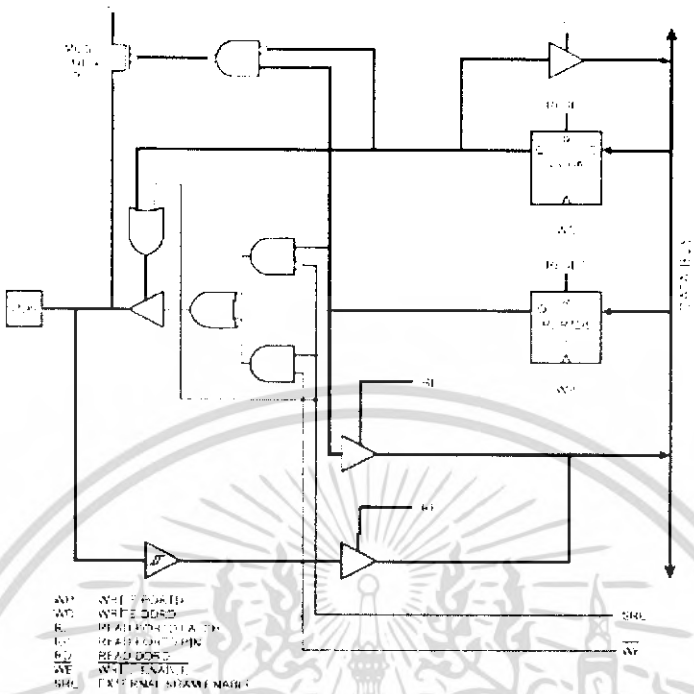
รูปที่ 2.62 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PD2และPD3)



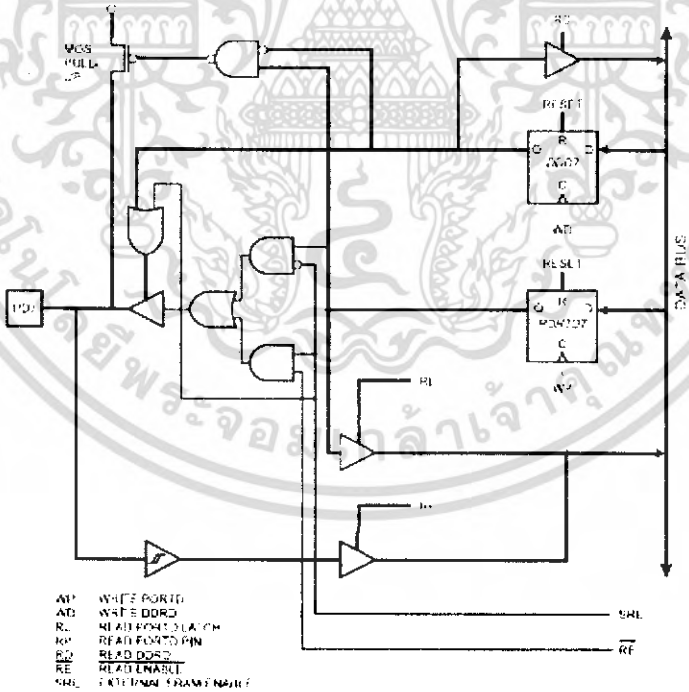
รูปที่ 2.63 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PD4)



รูปที่ 2.64 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PD5)



รูปที่ 2.65 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PD6)



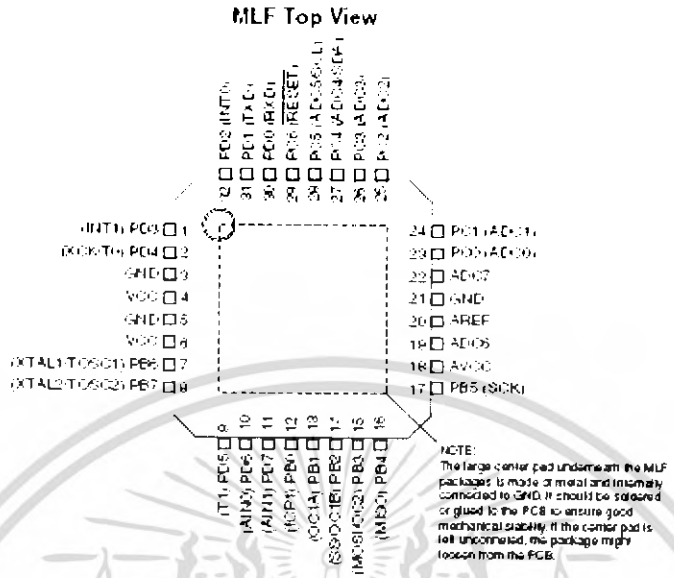
รูปที่ 2.66 แสดงโครงสร้างของ Port D Schematic Diagram (Pin PD7)

2.4 ทฤษฎีพื้นฐานเกี่ยวกับไมโครคอนโทรลเลอร์ ATmega8

2.4.1 สถาปัตยกรรม และคุณสมบัติโดยทั่วไปของ ATmega8

- สถาปัตยกรรมภายในถูกออกแบบให้ใช้สถาปัตยกรรมแบบ RISC (Reduce Instruction Set Computer)
  มีคำสั่งควบคุมการทำงานของไมโครคอนโทรลเลอร์จำนวน 130 คำสั่ง
- หน่วยความจำแบบ Flash สำหรับบันทึก Program Memory ขนาด 8 Kbytes
- หน่วยความจำแบบ EEPROM สำหรับบันทึก Data Memory ขนาด 512 byte
- หน่วยความจำแบบ RAM ขนาด 1 Kbytes
- กลุ่มรีจีสเตอร์ใช้งานทั่วไป 8 บิต จำนวน 32 ตัว
- ความถี่สัญญาณนาฬิกา 0 - 16 MHz
- ระบบการตรวจจับสัญญาณอนาลอก(Analog Comparator)
- ระบบตรวจจับการทำงานผิดพลาดของ CPU (Watchdog Timer With On – Chip Oscillator)
- ระบบอินเตอร์รัพท์จากภายนอก (External Interrupt)
- Timer/Counter ขนาด 16 บิต 1 Channel
- Timer/Counter ขนาด 8 บิต 2 Channel
- ระบบการสื่อสารผ่านพอร์ทอนุกรม
- ปรับค่า RC ภายในได้

ATmega8 เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ที่สถาปัตยกรรมแบบ RISC (Reduce Instruction Set Computer) ซึ่งทำให้การประมวลผลมีความเร็ว 1 คำสั่ง/1สัญญาณนาฬิกา หรือ CPU สามารถประมวลผลคำสั่งได้ 1 MIPS/MHz

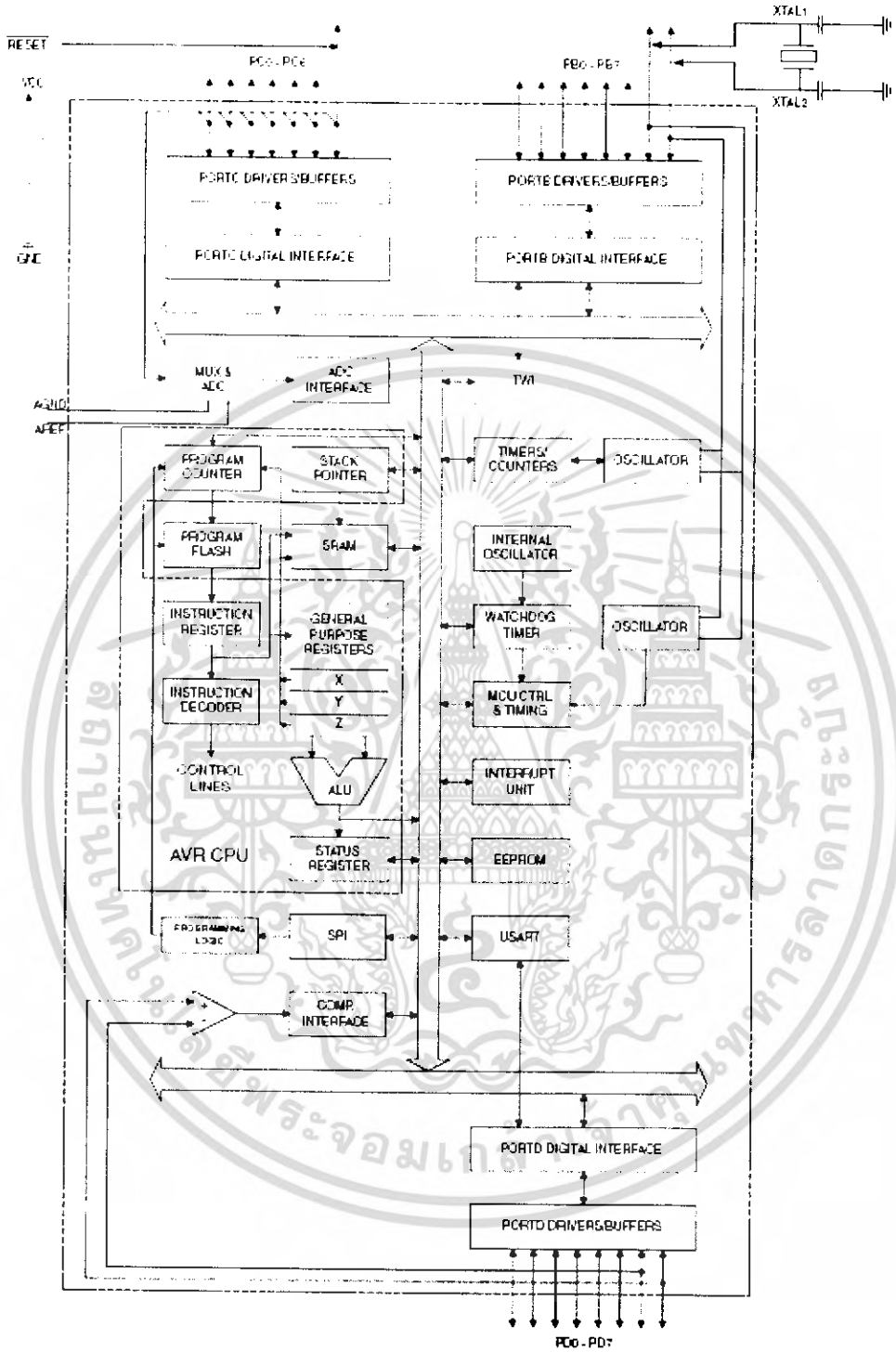


รูปที่ 2.67 โครงสร้างภายนอกและตำแหน่งขาของ ATmega8

ภายในประกอบด้วยรีจีสเตอร์ใช้งานทั่วไปขนาด 8 บิต จำนวน 32 ตัว ซึ่งแต่ตัวจะต่อเข้ากับ ALU โดยตรงทำให้การประมวลต่อ 1 คำสั่งมีความเร็วกว่า CPU ที่สถาปัตยกรรมแบบ CISC

2.4.2 โครงสร้างภายในของ ATmega8

AT90S8515 มีหน่วยความจำสำหรับ Program Memory แบบ Flash ขนาด 8 Kbytes หน่วยความจำสำหรับ Data Memory แบบ EEPROM ขนาด 512 Byte และหน่วยความจำแบบ RAM ขนาด 512 Byte มีพอร์ทที่สามารถทำงานได้ 2 ทิศทาง จำนวน 32 เส้นสัญญาณ และ ระบบ Timer/Counter จำนวน 2 ชุดที่มีโหมดการทำงาน



รูปที่ 2.68 บล็อกไดอะแกรมโครงสร้างของ ATmega8

2.4.3 รายละเอียดโครงสร้างภายนอกของ ATmega8

Vcc ขาจ่ายไฟให้กับ CPU

GND ขาต่อกราวด์

Port B (PB7..PB0)XTAL1/XTAL2/OSC1/OSC2เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA โดยแต่ละขาสัญญาณจะถูกใช้งานในฟังก์ชั่นอื่น ๆ ได้อีก

Port C (PC7..PC0) เป็นพอร์ท 2 ทิศทาง ขนาด 7 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA โดยแต่ละขาสัญญาณจะถูกใช้งานในฟังก์ชั่นอื่น ๆ ได้อีก

PC6/RESET ใช้เป็นขารีเซ็ทโดยการป้อน0

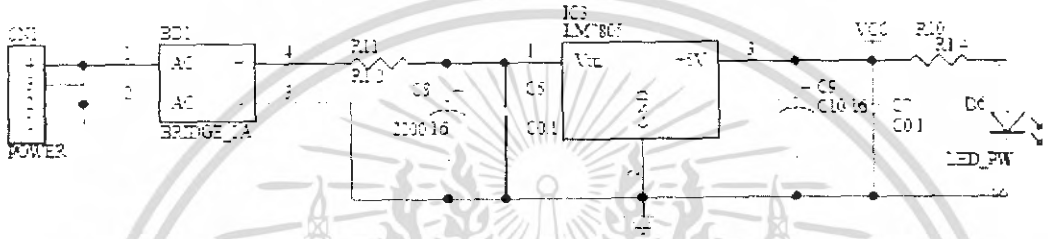
Port D (PD7..PD0) เป็นพอร์ท 2 ทิศทาง ขนาด 8 บิต โดยสามารถกำหนดให้แต่ละขาของพอร์ทสามารถ Pull Up ภายในแยกจากกัน ซึ่งสามารถรับกระแส Sink 20mA โดยแต่ละขาสัญญาณจะถูกใช้งานในฟังก์ชั่นอื่น ๆ ได้อีก

บทที่ 3
การออกแบบและการสร้าง

ในบทนี้จะเป็นเรื่องของการคำนวณและการสร้างวงจรขึ้นมา โดยอาศัยบล็อคไดอะแกรมที่กล่าวมาแล้วในบทที่ 1 มาใช้ในการสร้างวงจร โดยในการสร้างวงจรจะทำให้ได้ใกล้เคียงกับทฤษฎีและผลการทำงานที่ต้องการมากที่สุด

3.1 วงจรส่วนรถประจำทาง

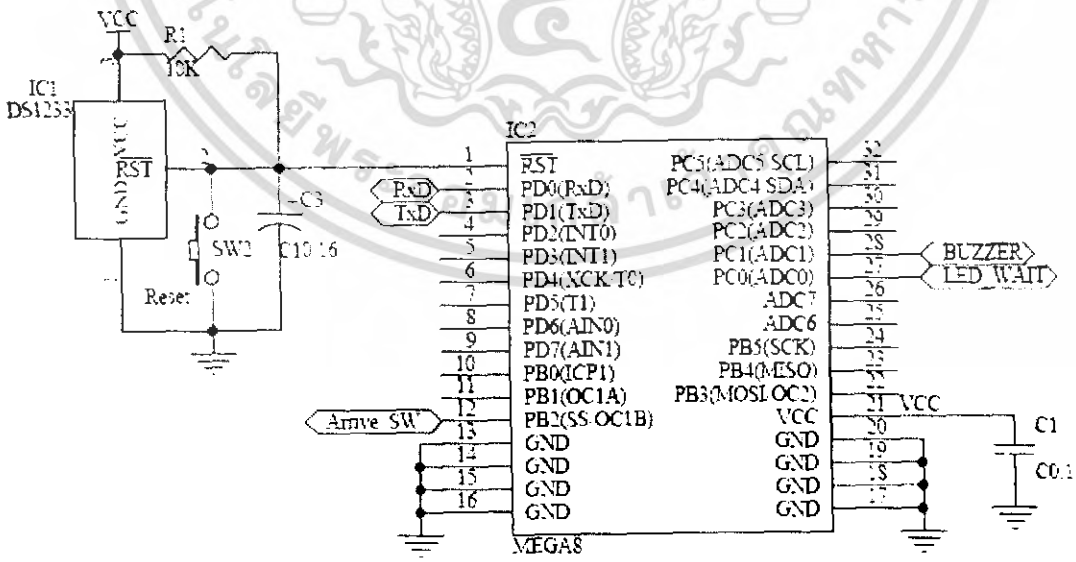
3.1.1 วงจรเพาเวอร์ซัพพลาย (Power Supply)



รูปที่ 3.1 วงจรเพาเวอร์ซัพพลายของส่วนรถประจำทาง

จากรูปที่ 3.1 เป็นวงจรในส่วนของเพาเวอร์ซัพพลาย ที่จ่ายกระแสไฟฟ้าให้กับส่วนต่างๆ ของวงจรส่วนรถประจำทาง จะได้รับแรงดัน +12 โวลต์ จากแบตเตอร์รี่ของรถประจำทาง โดยจะผ่านไดโอดบริดจ์เพื่อจัดขั้วของแรงดันให้ถูกต้อง จากนั้นใช้ไอซีเรกกูเลต 7805 ได้เอาต์พุทเป็นแรงดัน +5 โวลต์

3.1.2 วงจรส่วนไมโครคอนโทรลเลอร์ของส่วนรถประจำทาง

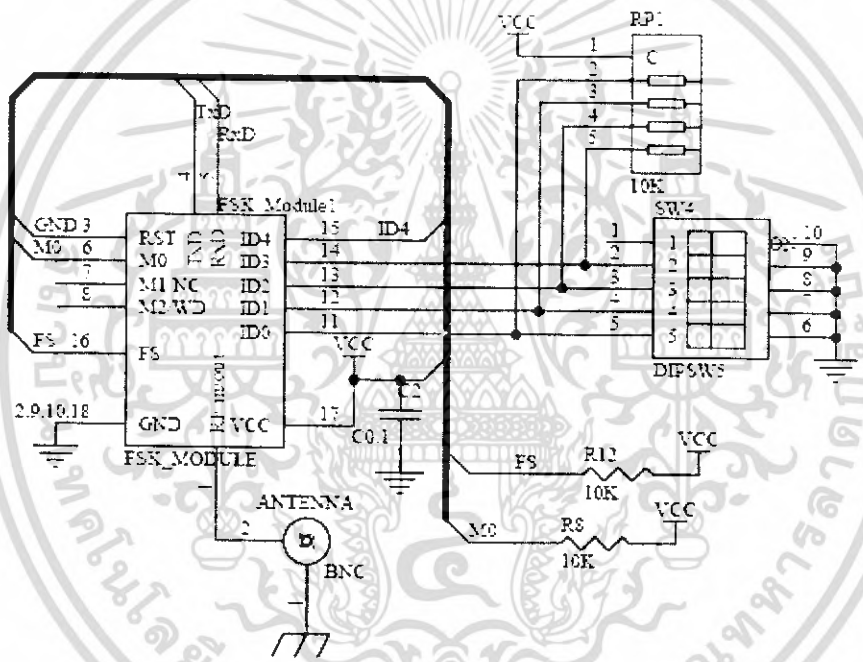


รูปที่ 3.2 วงจรไมโครคอนโทรลเลอร์ของส่วนรถประจำทาง

จากรูปที่ 3.2 แสดงวงจรในส่วนไมโครคอนโทรลเลอร์ของส่วนรถประจำทาง โดยใช้

- พอร์ท PD0 และพอร์ท PD1 ทำหน้าที่ติดต่อสื่อสารแบบ UART กับโมดูลเอฟเอสเค เพื่อนำข้อมูลมาประมวลผลต่อไป
- พอร์ท PB2 ต่อกับสวิตช์ชนิดกดติดปล่อยดับเพื่อให้คนขับรถประจำทางกดเมื่อรถประจำทางได้ถึงป้ายแล้ว
- พอร์ท PC1 ต่อกับลำโพงชนิดบัซเซอร์
- พอร์ท PC 0 ต่อกับหลอดแอลอีดี (LED) เพื่อแสดงเมื่อมีผู้โดยสารรออยู่ที่ป้ายรถประจำทาง

3.1.3 วงจรส่วนเอฟเอสเคโมดูลของส่วนรถประจำทาง



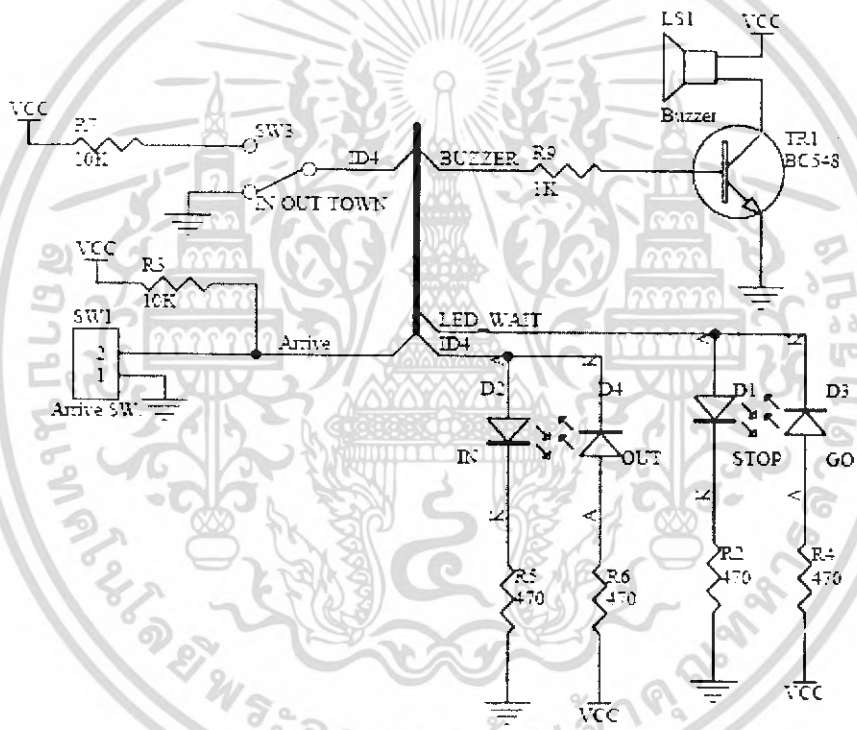
รูปที่ 3.3 วงจรเอฟเอสเคโมดูลของส่วนรถประจำทาง

วงจรส่วนเอฟเอสเคโมดูล ที่หน้าที่รับข้อมูลจากไมโครคอนโทรลเลอร์แล้วมามอดูเลตเป็นสัญญาณที่อยู่ในรูปคลื่นความถี่วิทยุแล้วส่งออกอากาศไปยังเครื่องรับ และรับสัญญาณที่อยู่ในรูปคลื่นความถี่วิทยุมาดีมอดูเลตเป็นข้อมูล เพื่อส่งให้ไมโครคอนโทรลเลอร์ประมวลผลต่อไป จากรูปที่ 3.3

- ขา 11 ถึงขา 14 จะต่อกับสวิตช์เพื่อเลือกช่องสัญญาณที่ใช้ในการติดต่อสื่อสารข้อมูล
- ขา 15 เป็นขาที่ใช้เลือกช่องสัญญาณในการติดต่อสื่อสารเช่นกัน แต่ในที่นี้ใช้ในการกำหนดว่ารถประจำทางวิ่งเป็นขาเข้าเมือง หรือออกจากเมือง

- ขา 16 จะเป็นขากำหนดความถี่ใช้งาน 433.93 เมกะเฮิร์ต หรือ 434.33 เมกะเฮิร์ต ในที่นี้เลือก 434.33 เมกะเฮิร์ต โดยการให้ลอจิก "1"
- ขา 4 เป็นขา TxD ต่อกับขา TxD ของไมโครคอนโทรลเลอร์เพื่อรับข้อมูลที่ต้องการส่งจากไมโครคอนโทรลเลอร์
- ขา 5 เป็นขา RxD ต่อกับขา RxD ของไมโครคอนโทรลเลอร์เพื่อส่งข้อมูลที่ได้รับมาส่งให้ไมโครคอนโทรลเลอร์
- ขา 6 เป็น M0 จะเป็นขากำหนดอัตราบิทในการรับส่งข้อมูล 9600 บิทต่อวินาที หรือ 4800 บิทต่อวินาที ในที่นี้เลือก 9600 บิทต่อวินาที โดยการให้ลอจิก "1"

3.1.4 วงจรรับค่าและแสดงผลของส่วนรถประจำทาง



รูปที่ 3.4 วงจรรับค่าและแสดงผลของส่วนรถประจำทาง

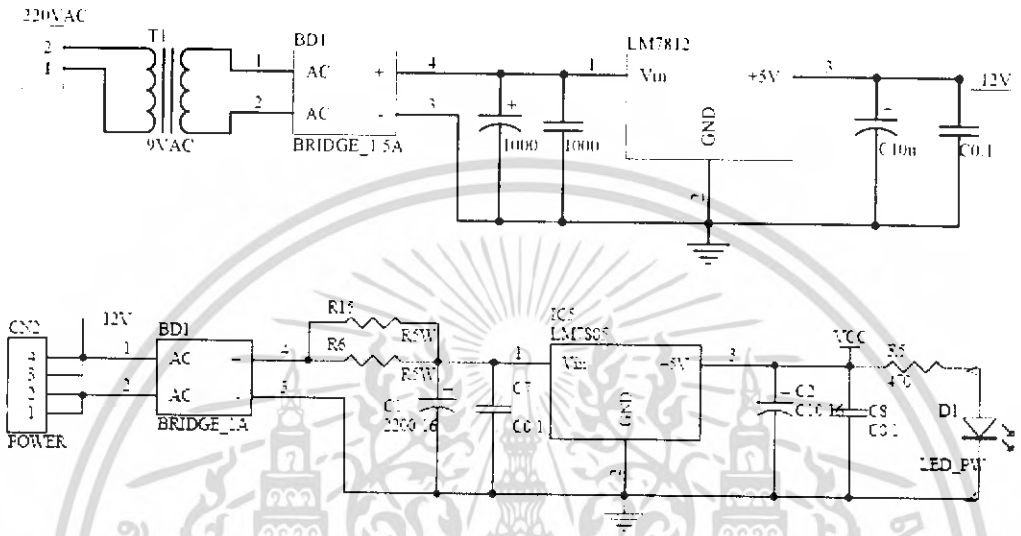
จากรูปที่ 3.4 แสดงวงจรรับค่าและแสดงผลของส่วนรถประจำทาง โดย

- SW1 จะต่อกับสวิตช์เพื่อกำหนดค่าส่งให้ไมโครคอนโทรลเลอร์ โดยคนขับรถโดยสารจะทำการกดสวิตช์เมื่อรถวิ่งถึงป้าย เพื่อกำหนดข้อมูลที่จะส่งไปยังป้ายรถประจำทาง
- SW3 จะเป็นสวิตช์เลือกช่องสัญญาณในการสื่อสารของเอฟเอสเคโมดูล และกำหนดว่ารถเป็นรถขาเข้าเมือง หรือออกจากเมือง
- D2 และ D4 เป็นหลอดแอลอีดีแสดงตำแหน่งสวิตช์ SW3 ว่าได้เลือกไปที่ตำแหน่งใด

- **D1 และ D3** เป็นหลอดแอลอีดีแสดงผลบอกพนักงานขับรถทราบว่ามีผู้โดยสารรออยู่ที่ป้ายรถประจำทางหรือไม่ โดย D1 จะสว่างเมื่อมีผู้โดยสารรออยู่ที่ป้ายรถประจำทาง และD3 จะสว่างในสภาวะปกติ

3.2 วงจรส่วนป้ายรถประจำทาง

3.2.1 วงจรเพาเวอร์ซัพพลาย (Power Supply)



รูปที่ 3.5 วงจรเพาเวอร์ซัพพลายของส่วนป้ายรถประจำทาง

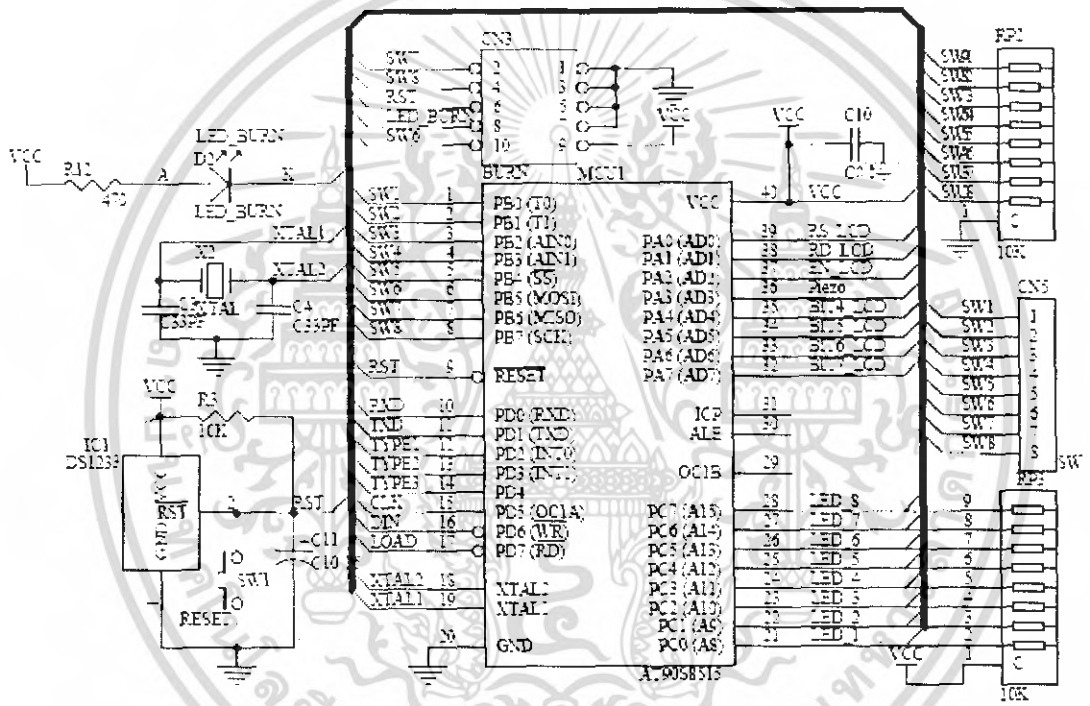
จากรูปที่ 3.5 เป็นวงจรในส่วนของเพาเวอร์ซัพพลาย หม้อแปลงจะแปลงแรงดันจาก 220 โวลต์เป็นแรงดัน 9 โวลต์ผ่านไดโอดบริดจ์ได้แรงดันไฟฟ้ากระแสตรงประมาณ +12.7 โวลต์ จากนั้นใช้ไอซีเรกกูเลต LM7812 ได้เอาต์พุตเป็นแรงดัน +12 โวลต์ เพื่อจ่ายกระแสไฟฟ้าให้กับวงจรในส่วนต่อไป จากนั้น CN2 จะรับแรงดัน +12 โวลต์ ผ่านไดโอดบริดจ์เพื่อจัดขั้วแรงดันให้ถูกต้อง แล้วทำการลดระดับและรักษาระดับแรงดัน +5โวลต์ ด้วยไอซีเรกกูเลต LM7805 เพื่อจ่ายให้วงจรต่างๆต่อไป

3.2.2 วงจรส่วนไมโครคอนโทรลเลอร์ของส่วนป้ายรถประจำทาง

จากรูปที่ 3.6 แสดงวงจรไมโครคอนโทรลเลอร์ของส่วนป้ายรถประจำทาง โดยที่ประกอบด้วย

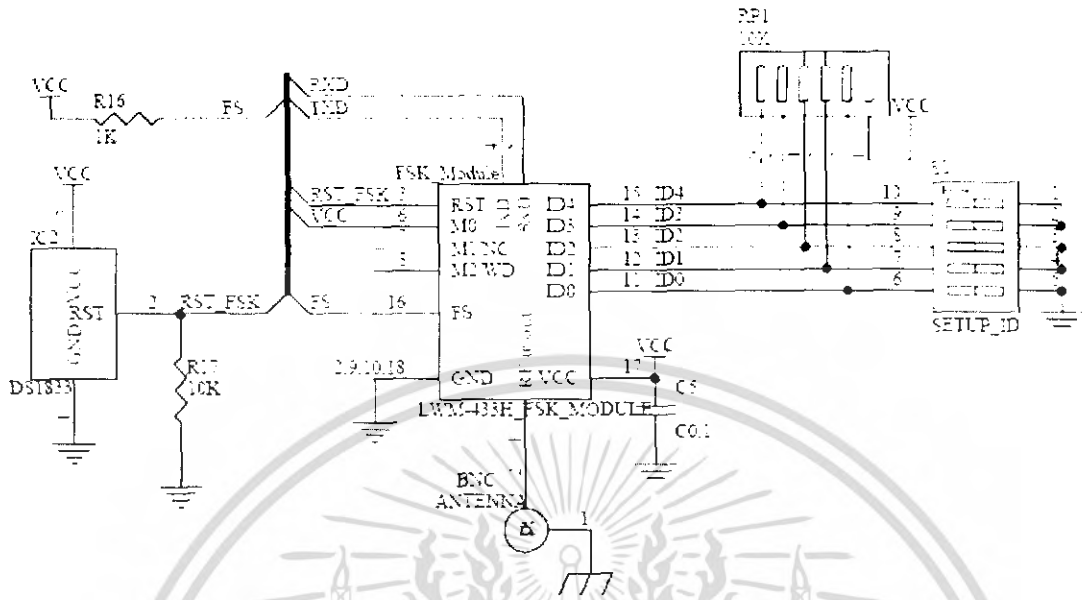
- **พอร์ท A** เป็นพอร์ทที่ใช้ควบคุมการแสดงผลของจอแอลซีดี (LCD) ที่ใช้แสดงผลข้อมูลในการทดลอง
- **พอร์ท B** ต่อกับสวิตช์เพื่อกำหนดค่าของสายรถประจำทางที่ผู้โดยสารกำลังรออยู่
- **พอร์ท C** ต่อกับหลอดแอลอีดีและจอแสดงผลของสายรถประจำทาง แอลอีดีจะสว่างเมื่อรถประจำทางสายที่กำหนดด้วยแอลอีดีหลอดนั้นเข้ามาในรัศมีการรับส่งข้อมูล

- CN3 เป็นคอนเนคเตอร์ที่ใช้ในการเขียนโปรแกรมให้กับไมโครคอนโทรลเลอร์ โดยจะมีหลอดแอลอีดี D2 สว่างเมื่อมีการเขียนโปรแกรม
- ขา 10 เป็นขา RxD ใช้ในการรับข้อมูลจากเอฟเอสเคโมดูล
- ขา 11 เป็นขา TxD ใช้ในการส่งข้อมูลไปยังเอฟเอสเคโมดูล
- ขา12,13 และขา 14 ต่อกับหลอดแอลอีดี เพื่อแสดงชนิดของรถประจำทางที่กำลังวิ่งเข้าป้ายรถประจำทางในการการทดลอง



รูปที่ 3.6 วงจรไมโครคอนโทรลเลอร์ของส่วนป้ายรถประจำทาง

3.2.3 วงจรส่วนเอฟเอสเคโมดูลของส่วนป้ายรถประจำทาง

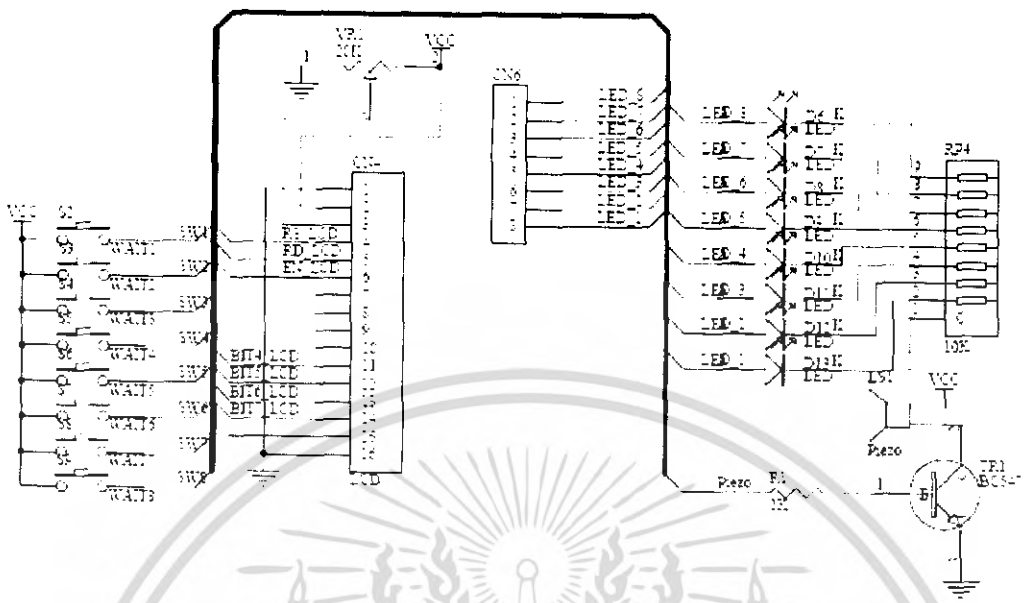


รูปที่ 3.7 วงจรเอฟเอสเคโมดูลของส่วนป้ายรถประจำทาง

วงจรส่วนเอฟเอสเคโมดูลที่หน้าที่เช่นเดียวกับวงจรเอฟเอสเคโมดูลในส่วนรถประจำทาง จากรูปที่ 3.7

- ขา 11 ถึงขา 15 จะต่อกับสวิตซ์เพื่อเลือกช่องสัญญาณที่ใช้ในการติดต่อสื่อสารข้อมูล
- ขา 16 จะเป็นขากำหนดความถี่ใช้งาน 433.93 เมกะเฮิร์ต หรือ 434.33 เมกะเฮิร์ต ในที่นี้เลือก 434.33 เมกะเฮิร์ต โดยการให้ลอจิก "1"
- ขา 4 เป็นขา TxD ต่อกับขา TxD ของไมโครคอนโทรลเลอร์เพื่อรับข้อมูลที่ต้องการส่งจากไมโครคอนโทรลเลอร์

3.2.4 วงจรรับค่าและแสดงผลของส่วนรถประจำทาง



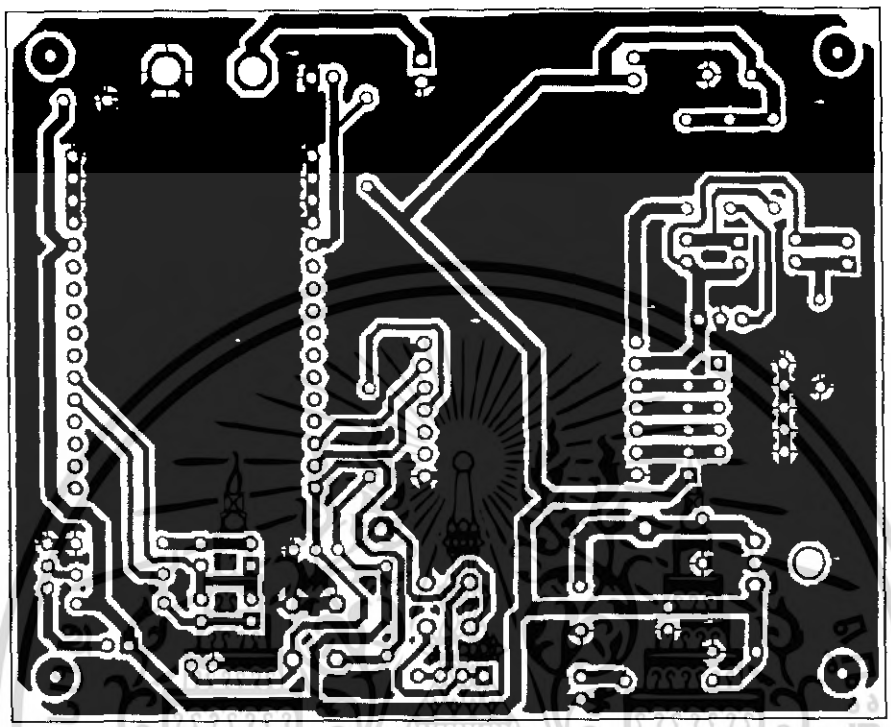
รูปที่ 3.8 วงจรรับค่าและแสดงผลของส่วนป้ายรถประจำทาง

จากรูปที่ 3.8 แสดงวงจรรับค่าและแสดงผลของส่วนป้ายรถประจำทางประกอบด้วย

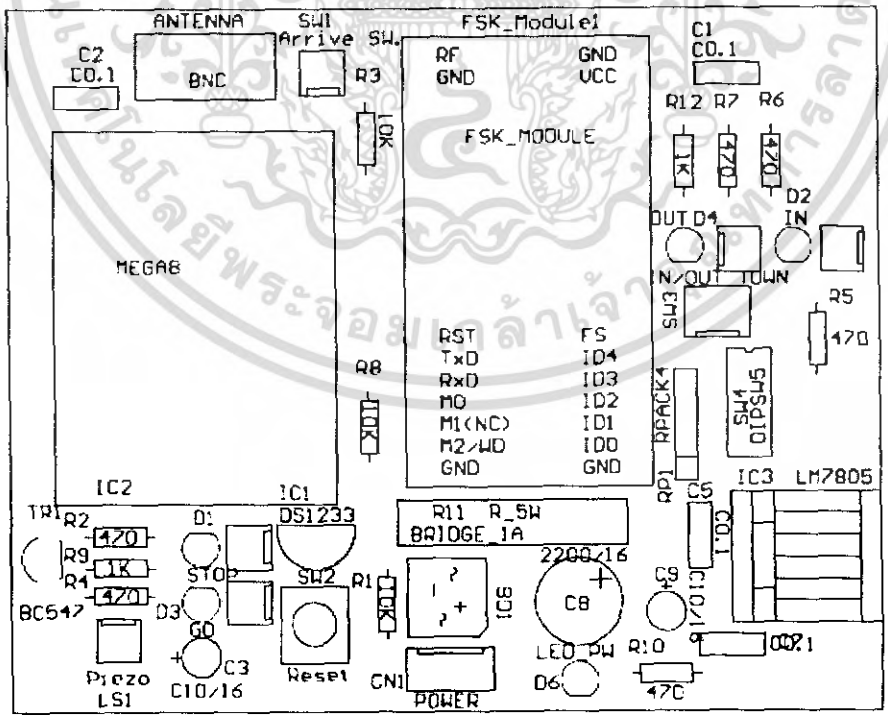
- **S2 ถึง S9** เป็นสวิตช์สำหรับให้ผู้โดยสารกด เพื่อระบุว่าต้องการรอรถประจำทางหมายเลขที่เข้ามาในรัศมีการรับส่งข้อมูล
- **LED1 ถึง LED8** เป็นหลอดแอลอีดีที่ใช้แสดงผล เมื่อรถประจำทางกำลังเข้ามาในรัศมีการรับส่งข้อมูลในที่นี้สามารถแสดงได้ 8 หมายเลข
- **CN4** เป็นคอนเนคเตอร์ต่อกับจอแอลซีดีเพื่อใช้แสดงผลในการทดลอง
- **CN6** เป็นคอนเนคเตอร์ต่อกับจอแสดงผลหมายเลขรถประจำทาง

3.3 การออกแบบแผ่นวงจร

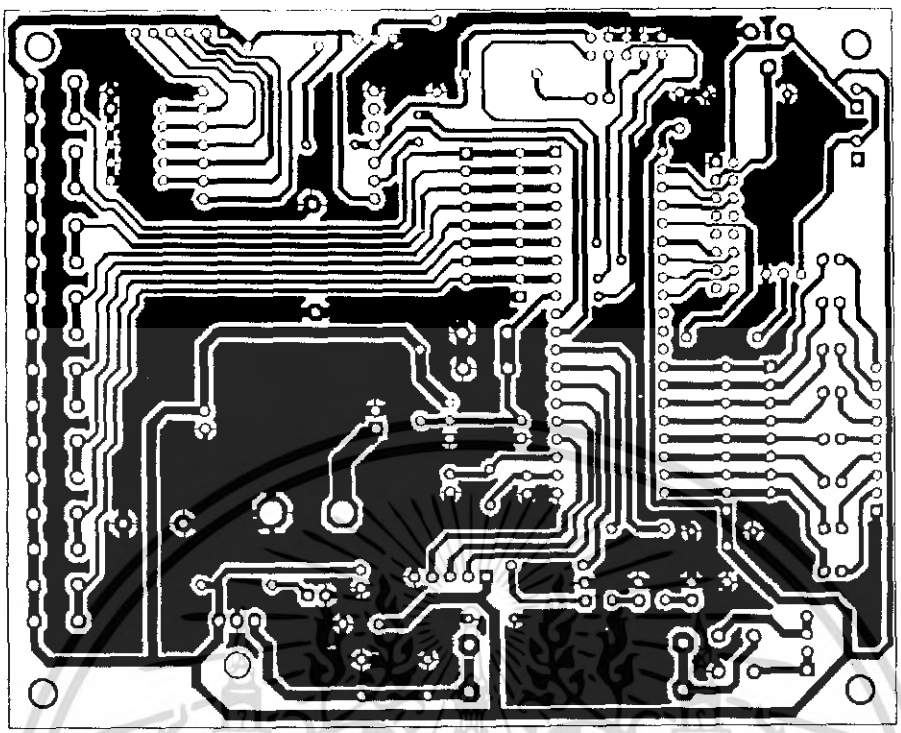
จากวงจรในส่วนต่างๆ สามารถออกแบบแผ่นวงจรได้ดังนี้



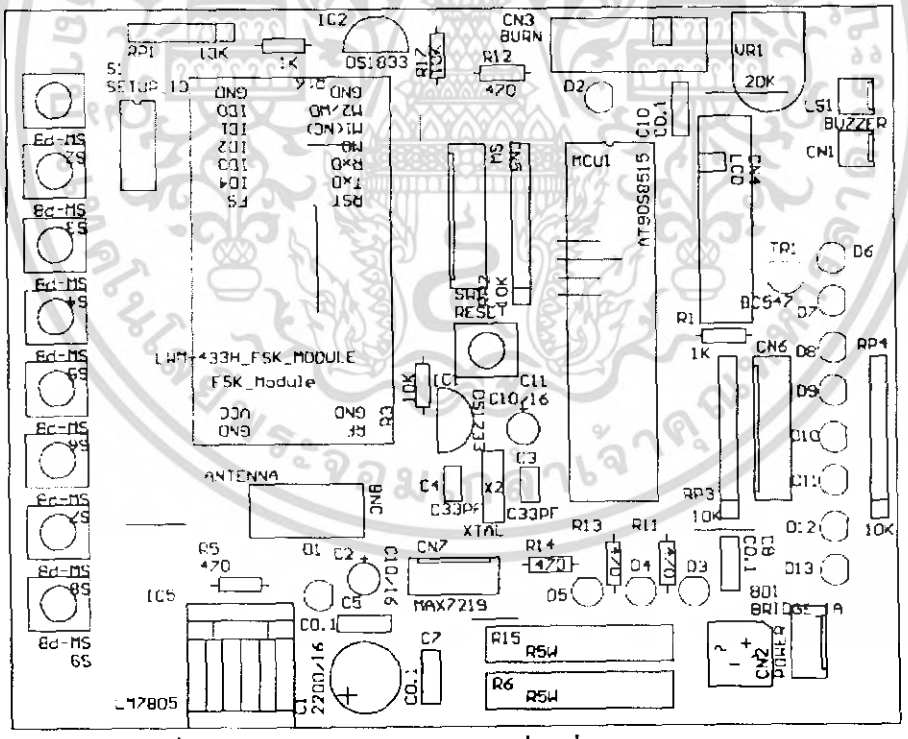
รูปที่ 3.9 ลายทองแดงด้านบัดกรีของเครื่องที่รถประจำทาง



รูปที่ 3.10 แสดงการวางอุปกรณ์ของเครื่องที่รถประจำทาง



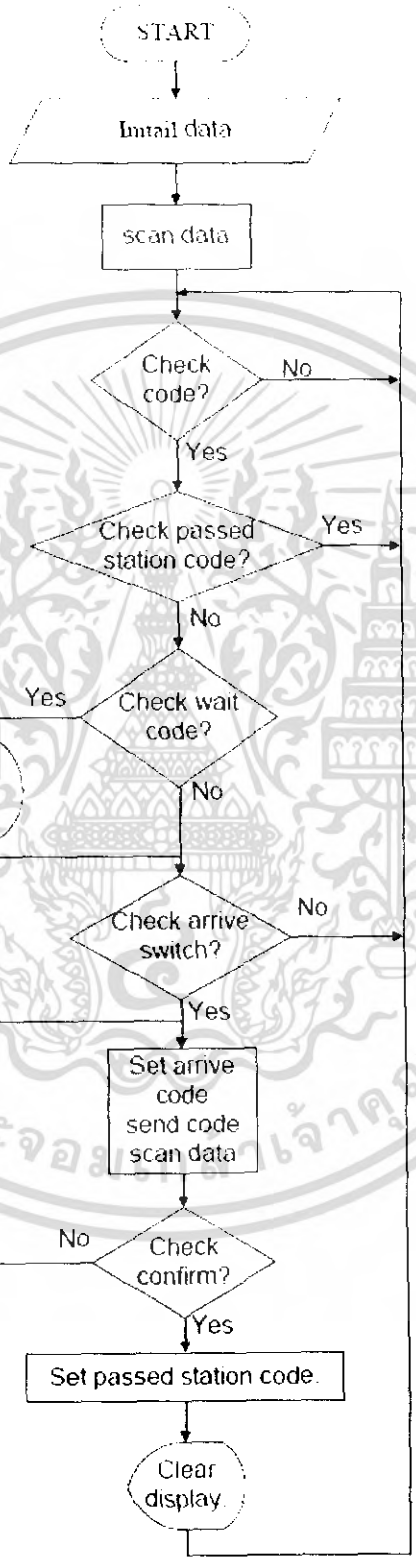
รูปที่ 3.11 ลายทองแดงด้านบัดกรีของเครื่องที่ป้ายรถประจำทาง



รูปที่ 3.12 แสดงวางอุปกรณ์ของเครื่องที่ป้ายรถประจำทาง

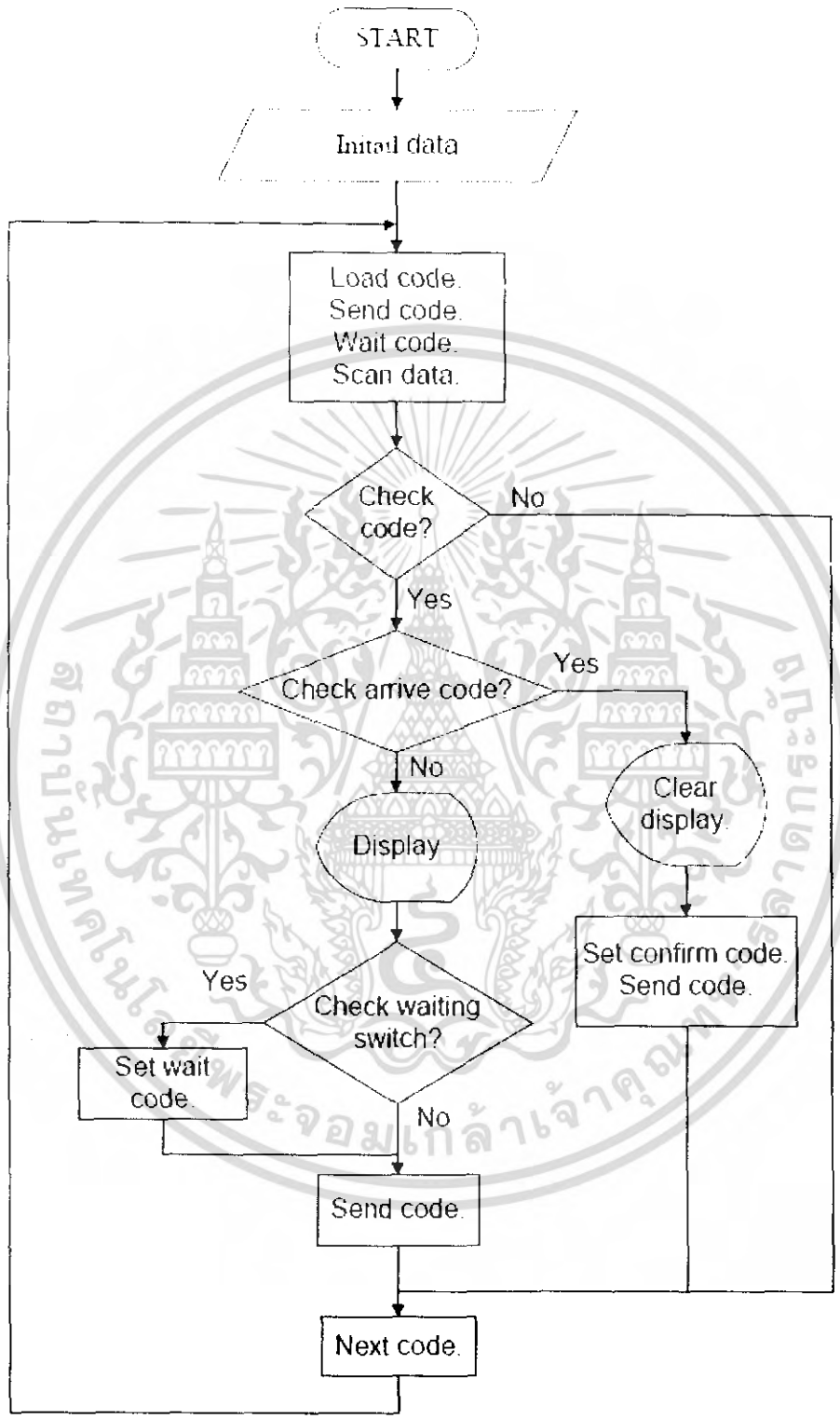
3.4 การออกแบบโปรแกรม

3.4.1 โปรแกรมส่วนรถประจำทาง



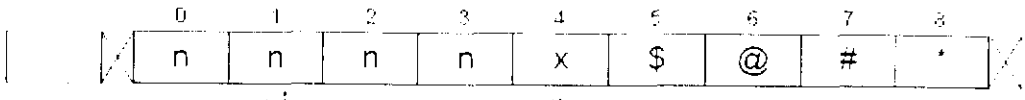
รูปที่ 3.13 แสดง Flow Chart การทำงานของโปรแกรมของส่วนรถประจำทาง

3.4.2 โปรแกรมส่วนป้ายรถประจำทาง



รูปที่ 3.14 แสดง Flowchart การทำงานของโปรแกรมของส่วนป้ายรถประจำทาง

3.4.3 การจัดสรรข้อมูลในการสื่อสารระหว่างรถประจำทางกับป้ายรถประจำทาง



รูปที่ 3.15 แสดงการจัดสรรข้อมูลของส่วนรถประจำทาง

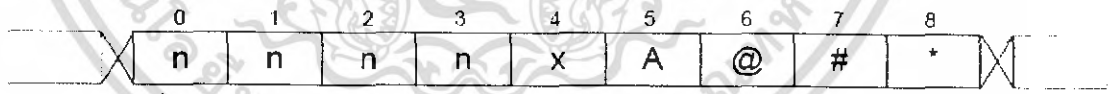
จากรูปที่ 3.15 แสดงการจัดสรรข้อมูลในการเขียนโปรแกรมสำหรับส่วนรถประจำทาง สามารถอธิบายได้ดังนี้คือ

- ข้อมูลตำแหน่งที่ 0 ถึง 3 แทนหมายเลขของรถประจำทาง
- ข้อมูลตำแหน่งที่ 4 จะแทนชนิดของรถประจำทาง โดยกำหนดให้

  x คือ รถธรรมดา

  y คือ รถปรับอากาศ (ปอ.)

  z คือ รถปรับอากาศพิเศษ (ปอ.พ.)
- ข้อมูลตำแหน่งที่ 5 จะเป็นตำแหน่งสำหรับตรวจสอบค่าของสถานีป้ายรถประจำทาง
- ข้อมูลตำแหน่งที่ 6 จะเป็นตำแหน่งสำหรับตั้งค่าแสดงเมื่อรถประจำทางถึงป้ายแล้ว
- ข้อมูลตำแหน่งที่ 7 จะเป็นตำแหน่งสำหรับตรวจสอบว่ามีการกดสวิตช์ รอรถประจำทางสายนี้อยู่หรือไม่
- ข้อมูลตำแหน่งที่ 8 จะเป็นตำแหน่งสำหรับตรวจสอบว่าที่รถประจำทางได้รับสัญญาณแจ้งว่ารถประจำทางถึงป้ายแล้ว เพื่อทำการเตรียมข้อมูลสำหรับการติดต่อกับป้ายรถประจำทางถัดไป



รูปที่ 3.16 แสดงการจัดสรรข้อมูลของส่วนป้ายรถประจำทาง

จากรูปที่ 3.16 แสดงการจัดสรรข้อมูลในการเขียนโปรแกรมสำหรับส่วนป้ายรถประจำทาง สามารถอธิบายได้ดังนี้คือ

- ข้อมูลตำแหน่งที่ 0 ถึง 3 แทนหมายเลขของรถประจำทาง
- ข้อมูลตำแหน่งที่ 4 จะแทนชนิดของรถประจำทาง โดยป้ายจะตรวจสอบชนิดของรถประจำทางจากข้อมูลตำแหน่งนี้
- ข้อมูลตำแหน่งที่ 5 จะเป็นตำแหน่งแสดงค่าของสถานีป้ายรถประจำทาง
- ข้อมูลตำแหน่งที่ 6 จะเป็นตำแหน่งสำหรับตรวจสอบเมื่อรถประจำทางถึงป้ายแล้ว เพื่อทำการเคลียร์ค่าการแสดงผล

ข้อมูลตำแหน่งที่ 7 จะเป็นตำแหน่งตั้งค่าสำหรับแสดงเมื่อมีการกดสวิตช์ รอรถประจำทางสายที่กำลังจะเข้าป้าย

- ข้อมูลตำแหน่งที่ 8 จะเป็นตำแหน่งสำหรับส่งค่าเมื่อรถประจำทางถึงป้ายแล้ว เพื่อให้รถประจำทางเตรียมข้อมูลสำหรับการติดต่อกับป้ายรถประจำทางถัดไป

บทที่ 4

การทดลอง และผลการทดลอง

การทำงานของเครื่องรับส่ง

แบ่งการทำงานเป็น 2 ส่วน สำคัญ คือ

1. เครื่องรับส่งที่ป้ายรถประจำทาง
2. เครื่องรับส่งที่รถประจำทาง

โดยที่เครื่องรับส่งทั้ง 2 ส่วนจะทำงานสัมพันธ์กัน เริ่มต้นด้วย เครื่องรับส่งที่ป้ายรถประจำทาง จะส่งสัญญาณ เพื่อตรวจสอบสายรถประจำทางที่เข้ามาในเขตรัศมีการสื่อสาร

เมื่อรถประจำทางวิ่งเข้ามาในเขตรัศมีการสื่อสารของเครื่องรับส่งที่อยู่บริเวณป้าย รถประจำทางจะได้รับสัญญาณจากเครื่องรับส่งที่อยู่บริเวณป้าย ส่งออกมาตรวจสอบสายรถประจำทางที่เข้ามาในรัศมีการสื่อสาร โดยรถประจำทางจะตรวจสอบหมายเลขประจำตัวรถ กับข้อมูลที่ได้รับมา ถ้าหากหมายเลขประจำตัวรถ ตรงกับข้อมูลที่ป้ายส่งออกมา ก็จะส่งสัญญาณแจ้งให้เครื่องรับส่งที่อยู่บริเวณป้ายรถประจำทางทราบว่า รถประจำทางสายใดกำลังจะวิ่งเข้ามาถึงป้าย โดยที่ป้ายรถประจำทางจะมีเสียงสัญญาณดังเตือนแล้วแสดงเป็นสายรถประจำทาง ให้ผู้โดยสารทราบ และมีสวิตซ์ให้ผู้โดยสารกด ถ้าหากว่าต้องการที่จะขึ้นรถประจำทางสายนั้น

เมื่อป้ายแสดงสายรถประจำทางแล้ว จะพิจารณาข้อมูลเป็น 2 กรณี คือ

1. กรณีไม่มีผู้โดยสารกดสวิตซ์ กรณีนี้ที่ป้ายแสดงหมายเลขจะแสดงสายรถประจำทางอยู่ตลอด จนกระทั่งรถประจำทางได้ผ่านป้ายรถประจำทาง พนักงานขับรถจะกดสวิตซ์ที่อยู่บนรถประจำทาง ทำให้ป้ายที่แสดงสายรถประจำทางดับลง
2. กรณีที่มีผู้โดยสารกดสวิตซ์ กรณีนี้เครื่องรับส่งที่อยู่บริเวณป้าย จะส่งสัญญาณ แจ้งให้พนักงานขับรถทราบว่า มีผู้โดยสารต้องการจะขึ้นรถสายที่แสดงบนป้าย ให้รถเข้าจอดเพื่อรับผู้โดยสาร หลังจากรับผู้โดยสารเรียบร้อย เมื่อรถออกจากป้าย พนักงานขับรถจะกดสวิตซ์ที่อยู่บนรถประจำทาง ทำให้ป้ายที่แสดงสายรถประจำทางดับลง

ข้อมูลที่สื่อสารในแต่ละช่วง

- **ข้อมูลข้อมูลที่ป้ายส่งออก**

  1111yA(@#*/2222yA(@#*/3333zA(@#*/4444zA(@#*/5555xA(@#*/6666xA(@#*/7777xA(@#*/8888xA(@#*

  ข้อมูลหมายเลขประจำตัวรถประจำทาง

  หมายเลขประจำตัวรถ เริ่มต้น จะเป็น 3333zS(@#* (เป็นหมายเลขเฉพาะของแต่ละสาย)

- **ข้อมูลเมื่อรถประจำทางเข้าเขตรัศมีการสื่อสารกับป้ายรถประจำทาง**

  3333zA@#* (กรณีที่ข้อมูลจากป้ายส่งออกมาเป็น 3333zA(@#* ตัวอักษร S จะถูกแทนที่ด้วย A หมายถึง รถประจำทางได้เข้ามาในเขตป้ายรถประจำทาง A)

- **ข้อมูลเมื่อมีผู้โดยสารกดสวิตซ์บริเวณป้ายรถประจำทาง**

  3333zA@w* (จากปกติข้อมูลที่ส่งออกจากป้ายจะเป็น 3333zA@#* แต่เมื่อมีผู้โดยสารกดสวิตซ์ ตัวอักษร # จะถูกแทนที่ด้วย w หมายถึง มีผู้โดยสารต้องการจะขึ้นรถสายที่แสดงบนป้าย)

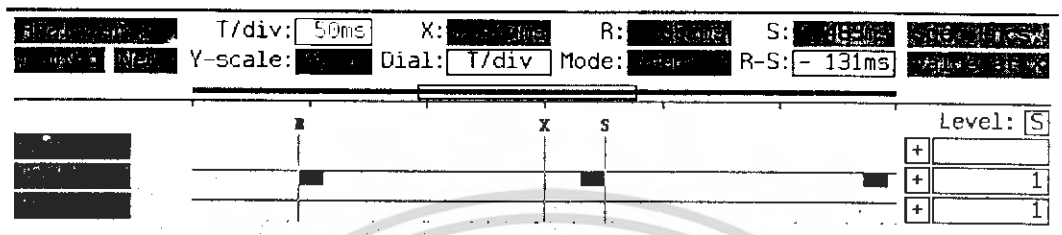
- **ข้อมูลเมื่อพนักงานขับรถกดสวิตซ์หลังจากผ่านป้าย**

  กรณีไม่มีผู้โดยสารกดสวิตซ์ ข้อมูลจะเป็น 3333zAa#* (จากข้อมูลหลังจากเข้าเขตรัศมีการสื่อสารกับป้ายรถประจำทาง A 3333zA@#* เมื่อพนักงานขับรถกดสวิตซ์ ตัวอักขร @ จะถูกแทนที่ด้วย a หมายถึง รถประจำทาง ผ่านป้ายแล้ว)

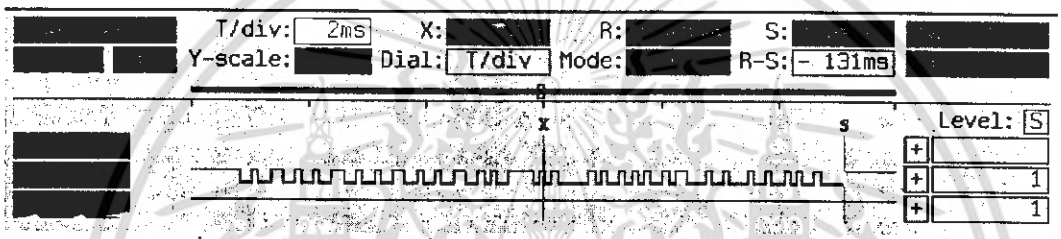
  กรณีไม่มีผู้โดยสารกดสวิตซ์ ข้อมูลจะเป็น 3333zAaw* (จากข้อมูลหลังจากเข้าเขตรัศมีการสื่อสารกับป้ายรถประจำทาง A 3333zA@w* เมื่อพนักงานขับรถกดสวิตซ์ ตัวอักขร @ จะถูกแทนที่ด้วย a หมายถึง รถประจำทาง ผ่านป้ายแล้ว)

การทดลองการรับส่งข้อมูลที่ป้ายรถประจำทาง

4.2.1 การรับส่งข้อมูลของเครื่องที่ป้ายรถประจำทางขณะที่ไม่มีรถเข้ามาในรัศมีการรับส่งข้อมูล



รูปที่ 4.1 แสดงสัญญาณเมื่อไม่มีรถประจำทางเข้ามาในรัศมีการรับส่งข้อมูล

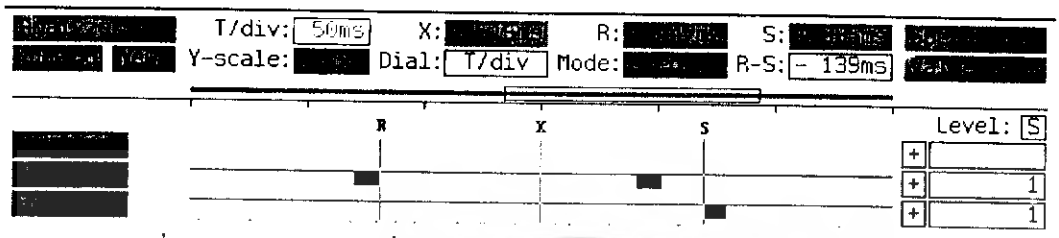


รูปที่ 4.2 แสดงสัญญาณเมื่อไม่มีรถประจำทางเข้ามาในรัศมีการรับส่งข้อมูล

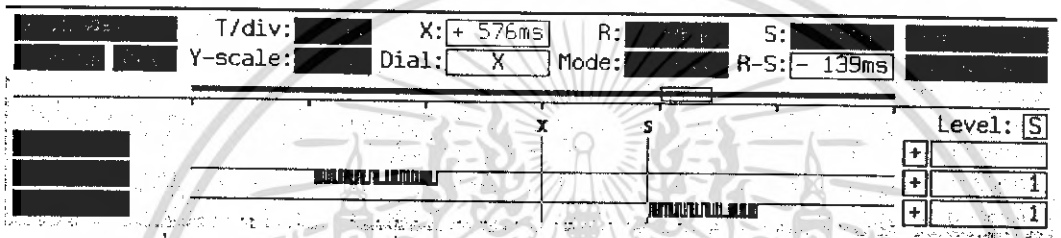
สัญญาณที่ส่งออก (Tx) เป็นข้อมูลของสายประจำทางที่เข้าจอดในป้าย ป้ายนี้ จากตัวอย่างที่ทดลองมีทั้งหมด 8 สาย ข้อมูลที่ส่งออกไปจึงมี8 ชุด 1111yA@#*/2222yA@#*/3333zA@#*/4444zA@#*/5555xA@#*/6666xA@#*/7777xA@#* /8888xA@#*

4.2.2 การรับส่งข้อมูลของเครื่องที่ป้ายรถประจำทางขณะที่มีรถประจำทางเข้ามาหนึ่งคัน

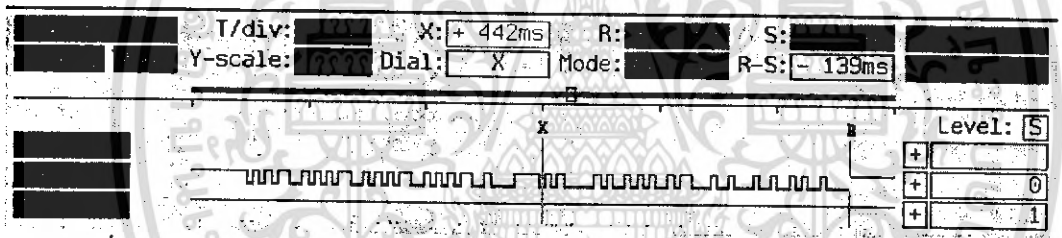
4.2.2.1 เมื่อไม่มีการกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง



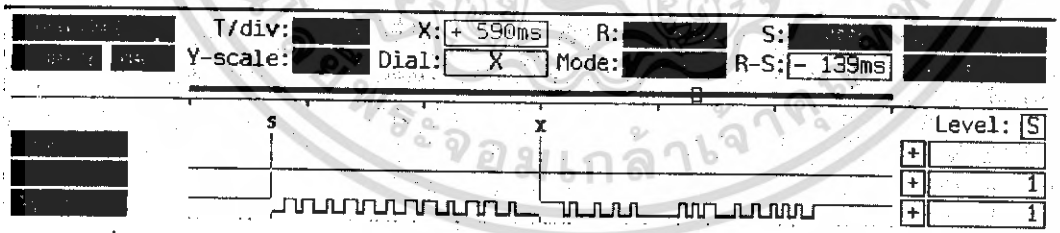
รูปที่ 4.3 แสดงสัญญาณเมื่อไม่มีการกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง



รูปที่ 4.4 แสดงสัญญาณเมื่อไม่มีการกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง



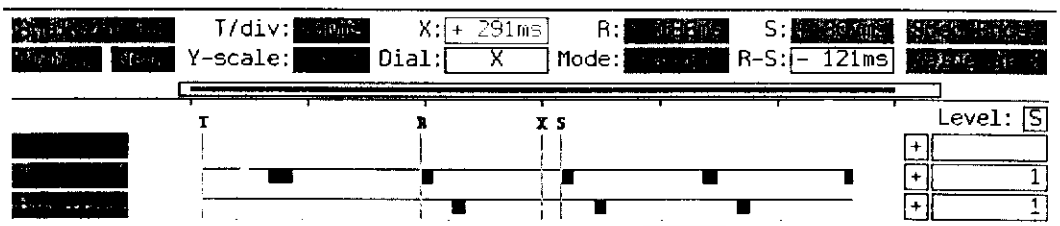
รูปที่ 4.5 แสดงสัญญาณที่ส่งออก(Tx) เมื่อไม่มีการกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง



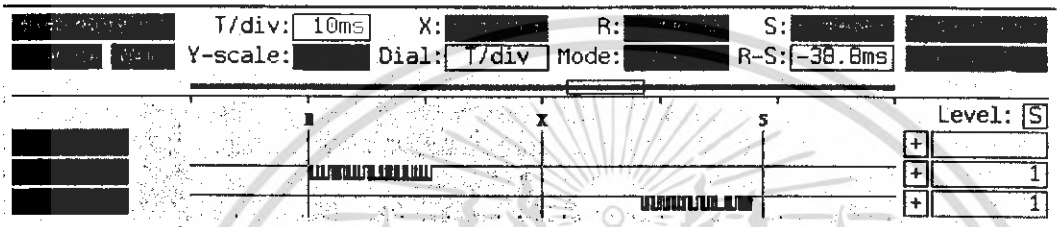
รูปที่ 4.6 แสดงสัญญาณที่รับ(Rx)เมื่อไม่มีการกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง

สัญญาณที่ส่งออก (Tx) ยังเป็นข้อมูลชุดเดิม ส่วนสัญญาณที่ได้รับ (Rx) เป็นสัญญาณที่ได้รับการตอบกลับจากรถ มีข้อมูลเป็น 3333zA@#*

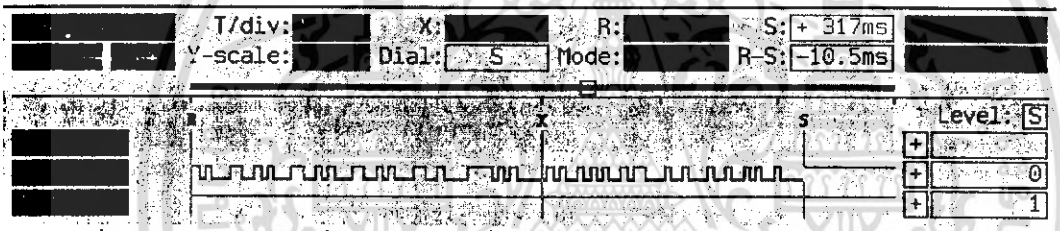
4.2.2.2 เมื่อมีการกดสวิตซ์รอรถประจำทางที่ป้ายรถประจำทาง



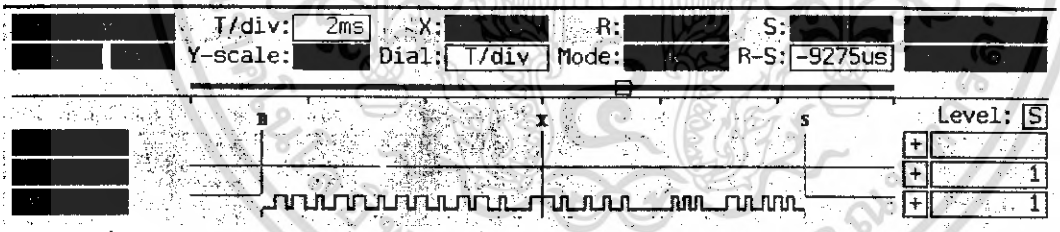
รูปที่ 4.7 แสดงสัญญาณเมื่อมีการกดสวิตซ์รอรถประจำทางที่ป้ายรถประจำทาง



รูปที่ 4.8 แสดงสัญญาณเมื่อมีการกดสวิตซ์รอรถประจำทางที่ป้ายรถประจำทาง



รูปที่ 4.9 แสดงสัญญาณที่ส่งออก (Tx) เมื่อมีการกดสวิตซ์รอรถประจำทางที่ป้ายรถประจำทาง

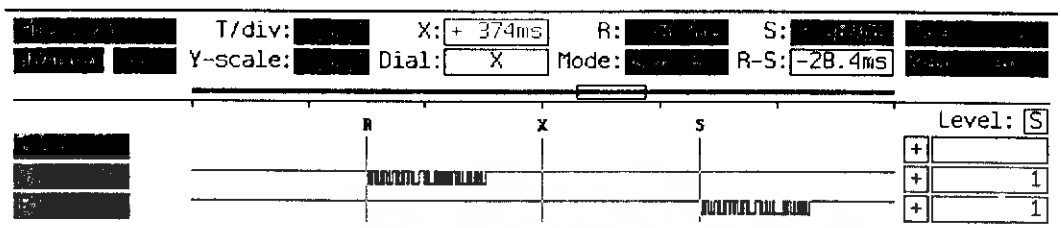


รูปที่ 4.10 แสดงสัญญาณที่รับ (Rx) เมื่อมีการกดสวิตซ์รอรถประจำทางที่ป้ายรถประจำทาง

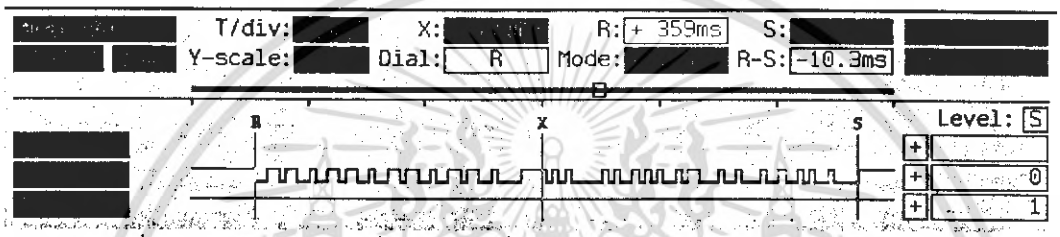
จากสัญญาณส่งออก (Tx) เริ่มแรกที่เครื่องรับส่งที่อยู่บริเวณป้ายส่งออก เมื่อมีผู้โดยสารต้องการจะขึ้นรถสายที่แสดงบนป้าย แล้วกดสวิตช์ที่ป้าย ทำให้ข้อมูลส่วนของสายรถที่ต้องการขึ้นมีการเปลี่ยนแปลงข้อมูล จาก 3333zA@#* เป็น 3333zA@w* (จาก # เปลี่ยนเป็น w แสดงว่ามีผู้โดยสารรอขึ้นรถประจำทาง)

4.2.3 การรับส่งข้อมูลของเครื่องที่ป้ายรถประจำทางขณะที่มีรถประจำทางเข้ามาสองคัน

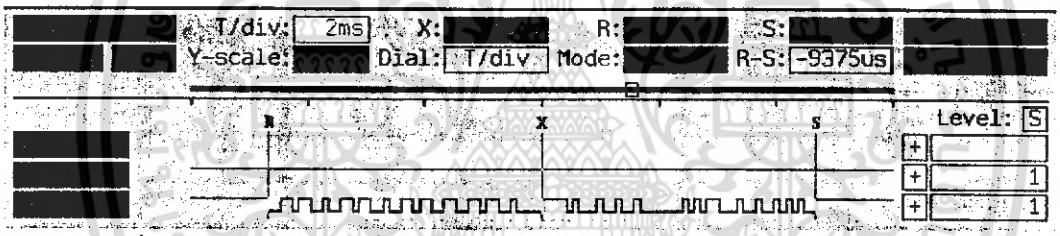
4.2.3.1 เมื่อผู้โดยสารกดสวิตซ์รอรถประจำทางคันที่หนึ่ง



รูปที่ 4.11 แสดงสัญญาณเมื่อผู้โดยสารกดสวิตซ์รอรถประจำทางคันที่หนึ่ง



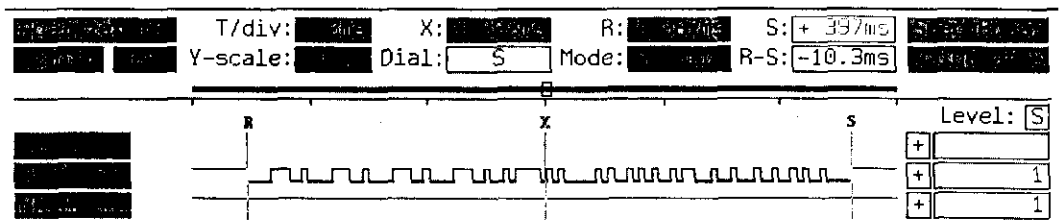
รูปที่ 4.12 แสดงสัญญาณที่ส่ง (Tx) เมื่อผู้โดยสารกดสวิตซ์รอรถประจำทางคันที่หนึ่ง



รูปที่ 4.13 แสดงสัญญาณที่รับ (Rx) เมื่อผู้โดยสารกดสวิตซ์รอรถประจำทางคันที่หนึ่ง

สัญญาณที่ส่งออก (Tx) จากการกดสวิตซ์ที่เครื่องรับส่งบริเวณป้าย เพื่อรอรถประจำทางคันแรก จะเป็นหมายเลข 3333zA@w*

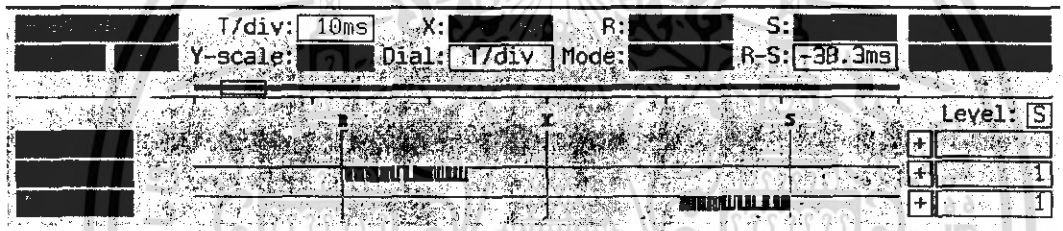
4.2.3.2 เมื่อผู้โดยสารกดสวิตซ์รอรถประจำทางคันที่สอง



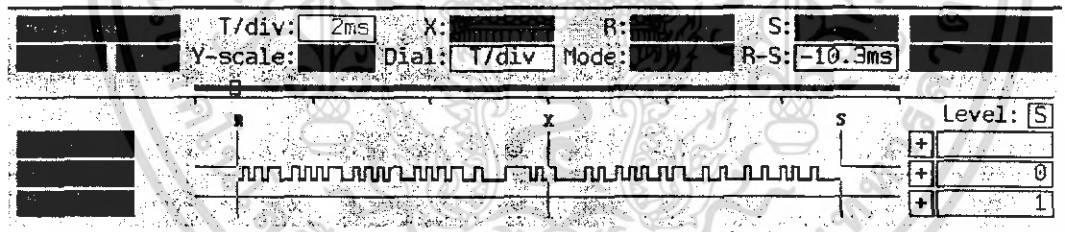
รูปที่ 4.14 แสดงสัญญาณเมื่อผู้โดยสารกดสวิตซ์รอรถประจำทางคันที่สอง

สัญญาณที่ส่งออก (Tx) จากการกดสวิตช์ที่เครื่องรับส่งบริเวณป้าย เพื่อรอรถประจำทางคันที่สอง จะเป็นหมายเลข 6666xAw#* (เป็นข้อมูลจากเครื่องรับส่ง ที่อยู่บนรถประจำทางคันที่สอง หมายเลขประจำตัวรถ จะไม่เหมือนกัน)

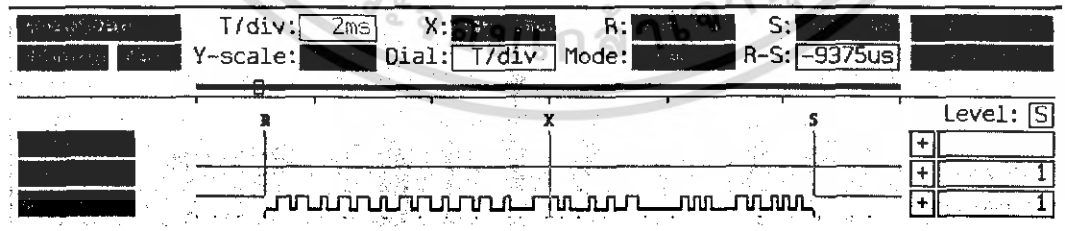
4.2.3.3 เมื่อรถประจำทางคันที่หนึ่งมาถึงป้ายรถประจำทาง



รูปที่ 4.15 แสดงสัญญาณเมื่อรถประจำทางคันที่หนึ่งมาถึงป้ายรถประจำทาง



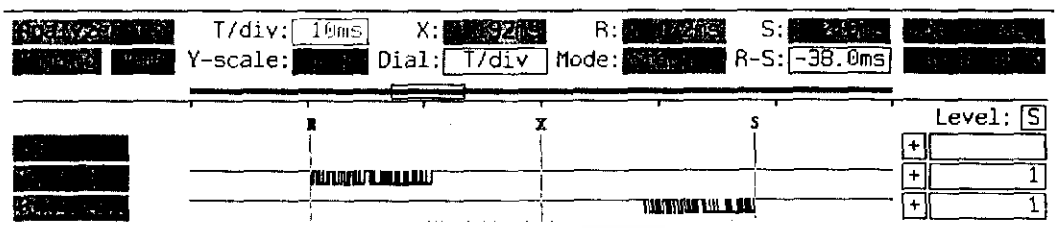
รูปที่ 4.16 แสดงสัญญาณที่ส่ง (Tx) ขณะที่รถประจำทางคันที่หนึ่งมาถึงป้ายรถประจำทาง



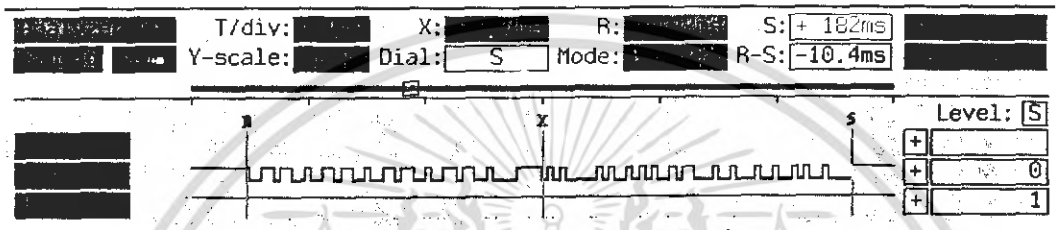
รูปที่ 4.17 แสดงสัญญาณที่รับ (Rx) ขณะที่รถประจำทางคันที่หนึ่งมาถึงป้ายรถประจำทาง

สัญญาณที่ส่งออก (Tx) ยังเป็นข้อมูลชุดเดิม ส่วนสัญญาณที่ได้รับ (Rx) เป็นสัญญาณที่ได้รับการตอบกลับจากรถ มีข้อมูลเป็น 3333zA@#*

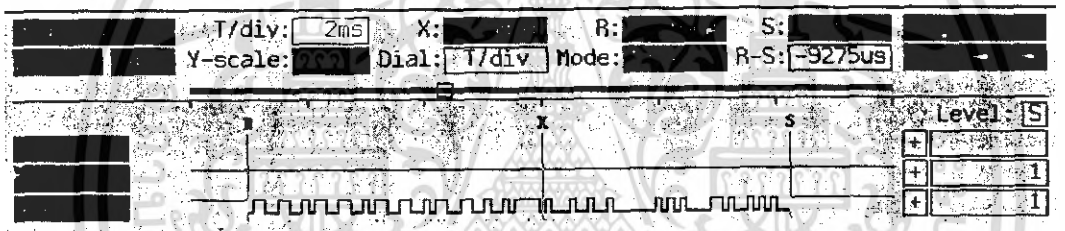
4.2.3.4 เมื่อรถประจำทางคันที่สองมาถึงป้ายรถประจำทาง



รูปที่ 4.18 แสดงสัญญาณเมื่อรถประจำทางคันที่สองมาถึงป้ายรถประจำทาง



รูปที่ 4.19 แสดงสัญญาณที่ส่ง (Tx) ขณะที่รถประจำทางคันที่สองมาถึงป้ายรถประจำทาง

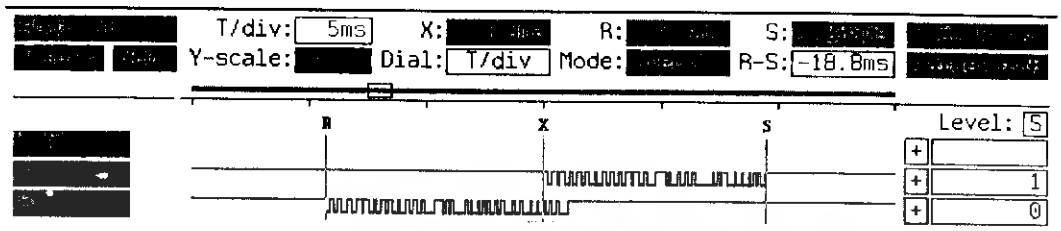


รูปที่ 4.20 แสดงสัญญาณที่รับ (Rx) ขณะที่รถประจำทางคันที่สองมาถึงป้ายรถประจำทาง

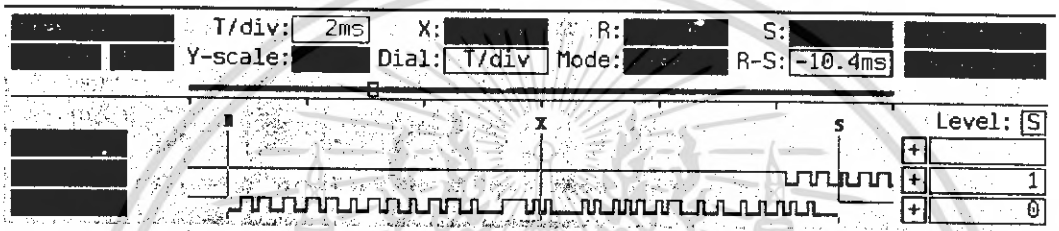
สัญญาณที่ส่งออก (Tx) ยังเป็นข้อมูลชุดเดิม ส่วนสัญญาณที่ได้รับ (Rx) เป็นสัญญาณที่ได้รับการตอบกลับจากรถ มีข้อมูลเป็น 6666xA@#*

การทดลองการรับส่งข้อมูลของเครื่องที่รถประจำทาง

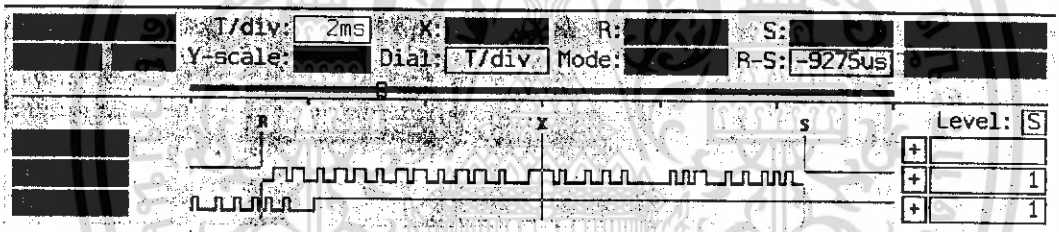
4.3.1 ขณะที่รถประจำทางเข้าไปในรัศมีการสื่อสารข้อมูล



รูปที่ 4.21 แสดงสัญญาณเมื่อรถประจำทางเข้าไปในรัศมีการสื่อสาร



รูปที่ 4.22 แสดงสัญญาณที่รับ(Rx)เมื่อรถประจำทางเข้าไปในรัศมีการสื่อสาร

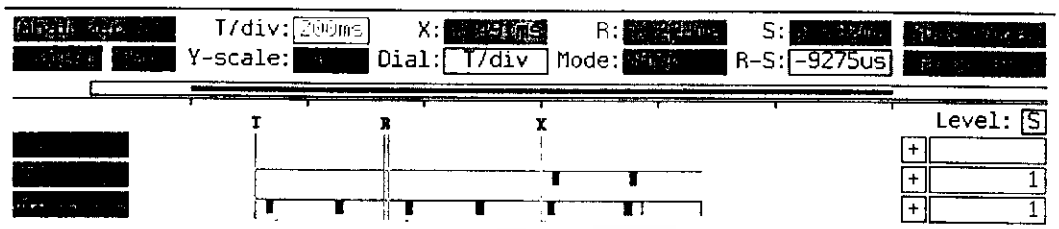


รูปที่ 4.23 แสดงสัญญาณที่ส่ง(Tx)เมื่อรถประจำทางเข้าไปในรัศมีการสื่อสาร

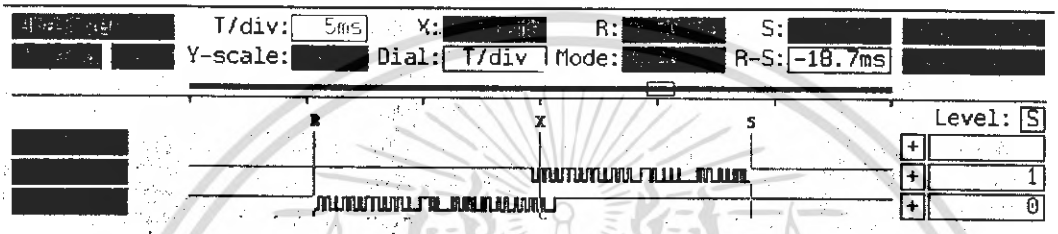
สัญญาณที่ได้รับ (Rx) เป็นข้อมูลของสายประจำทางที่เข้าจอดในป้ายที่จะถึง จากตัวอย่างที่ทดลองมีทั้งหมด 8 สาย ข้อมูลที่ส่งออกไปจึงมี8 ชุด 1111yA@#*/2222yA@#*/3333zA@#*/4444zA@#*/5555xA@#*/6666xA@#*/7777xA@#*/8888xA@#*

ส่วนสัญญาณส่งออก (Tx) เป็นข้อมูลของรถที่แสดงว่าเข้ามาในรัศมีป้ายรถประจำทาง คือ 3333zA@#*

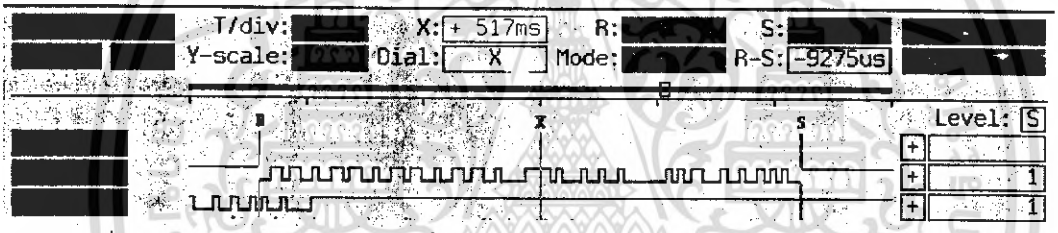
4.3.2 เมื่อมีผู้โดยสารกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง



รูปที่ 4.24 แสดงสัญญาณเมื่อผู้โดยสารกดสวิตช์รถรถประจำทางที่ป้ายรถประจำทาง



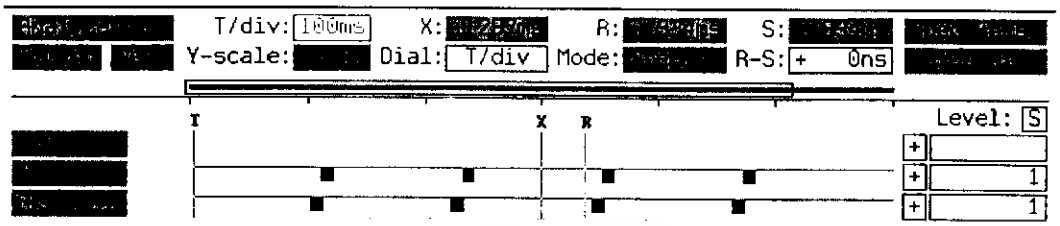
รูปที่ 4.25 แสดงสัญญาณเมื่อผู้โดยสารกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง



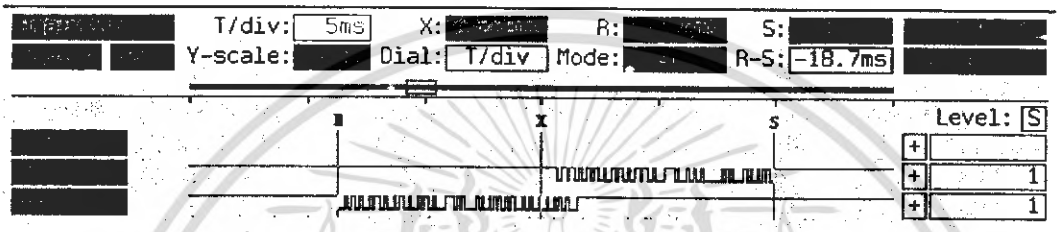
รูปที่ 4.26 แสดงสัญญาณส่ง(Tx)เมื่อผู้โดยสารกดสวิตช์รอรถประจำทางที่ป้ายรถประจำทาง

จากสัญญาณที่รับ(Rx) เริ่มแรกที่เครื่องรับส่งที่อยู่บริเวณป้ายส่งออก เมื่อมีผู้โดยสารต้องการจะขึ้นรถสายที่แสดงบนป้าย แล้วกดสวิตช์ที่ป้าย ทำให้ข้อมูลส่วนของสายรถที่ต้องการขึ้นมีการเปลี่ยนแปลงข้อมูล จาก 3333zA@#* เป็น 3333zA@w* (จาก # เปลี่ยนเป็น w แสดงว่ามีผู้โดยสารรอขึ้นรถประจำทาง)

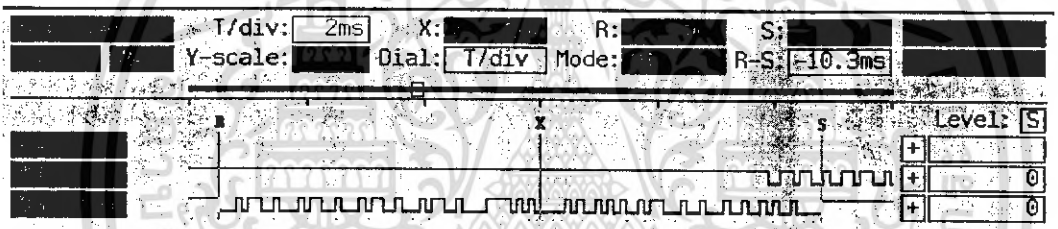
4.3.3 เมื่อรถประจำทางมาถึงป้ายรถประจำทาง



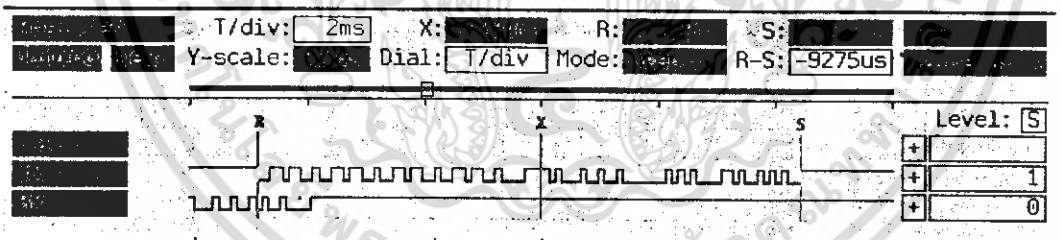
รูปที่ 4.27 แสดงสัญญาณเมื่อรถประจำทางมาถึงป้ายรถประจำทาง



รูปที่ 4.28 แสดงสัญญาณเมื่อรถประจำทางมาถึงป้ายรถประจำทาง



รูปที่ 4.29 แสดงสัญญาณที่รับ(Rx) เมื่อรถประจำทางมาถึงป้ายรถประจำทาง



รูปที่ 4.30 แสดงสัญญาณที่ส่ง(Tx) เมื่อรถประจำทางมาถึงป้ายรถประจำทาง

สัญญาณที่รับ (Rx) เป็นข้อมูลของสายประจำทางที่เข้าจอดในป้าย เป็นข้อมูลเดิมที่ออกมาจากเครื่องรับส่งบริเวณป้ายรถประจำทาง

3333zAa#* (จากข้อมูลหลังจากเข้าเขตรัศมีการสื่อสารกับป้ายรถประจำทาง A 3333zA@#* เมื่อพนักงานขับรถกดสวิตช์ ตัวอักขร @ จะถูกแทนที่ด้วย a หมายถึง รถประจำทางผ่านป้ายแล้ว) โดยข้อมูลที่ส่งออก (Tx) จะส่งออกไปอย่างต่อเนื่อง เพื่อจะเช็คความให้แน่ใจว่าข้อมูลได้ถูกส่งให้เครื่องรับส่งที่ป้ายรถประจำทางแล้ว

บทที่ 5

สรุปและวิจารณ์

จากโครงงานการสร้างระบบจัดการป้ายรถประจำทาง ที่ได้ออกแบบและสร้างเครื่องที่ป้ายรถประจำทางและเครื่องที่รถประจำทาง สามารถทำงานตามที่ออกแบบไว้ได้คือ

เมื่อรถประจำทางเข้ามายังป้ายรถประจำทาง ป้ายรถประจำทางนั้นก็จะแสดงหมายเลขของรถประจำทางที่กำลังจะเข้ามาได้ ถ้าหากมีผู้โดยสารกดสวิตช์ส่งสัญญาณว่ารอรถประจำทางคันที่กำลังเข้ามายังป้ายรถประจำทาง เครื่องที่รถประจำทางก็สามารถแสดงให้พนักงานขับรถทราบว่ามีผู้โดยสารรออยู่ที่ป้ายรถประจำทางข้างหน้า

กรณีรถประจำทางวิ่งสวนทางกันจะใช้คุณสมบัติของเอฟเอสเคโมดูลที่มีหลายช่องสัญญาณในการสื่อสาร โดยจะให้พนักงานขับรถโดยสารกดสวิตช์เลือกว่าจะเป็นการขับขาเข้าเมืองหรือออกจากเมือง

การทดลองและผลการทดลองได้แสดงไว้ในบทที่ 4 สามารถสรุปและวิจารณ์ผลการทดลองได้ดังนี้

5.1 สรุปผลการทดลองของเครื่องที่ป้ายรถประจำทาง

ขณะที่ยังไม่มีรถประจำทางเข้ามาในรัศมีการรับส่งข้อมูล เครื่องที่ป้ายรถประจำทางจะทำหน้าที่ส่งสัญญาณของรหัสรถประจำทาง ที่จะผ่านป้ายรถประจำทางป้ายนี้ออกไปอย่างเดียว ที่ภาครับจะไม่มีสัญญาณเข้ามา โดยสัญญาณของรหัสแต่ละหมายเลขจะมีความห่างกันประมาณ 120 ms เมื่อมีรถประจำทางเข้ามาในรัศมีการรับส่งข้อมูล เครื่องที่ป้ายรถประจำทางจะสามารถรับสัญญาณตอบกลับจากรถประจำทางได้ แล้วนำเอาสัญญาณที่รับได้ไปประมวลผลแสดงหมายเลขของรถประจำทางที่เข้ามาถึงป้ายรถประจำทาง แสดงให้ผู้โดยสารทราบว่ารถประจำทางหมายเลขใดกำลังจะเข้ามาถึงป้ายรถประจำทาง แล้วผู้โดยสารก็สามารถกดสวิตช์ส่งสัญญาณไปยังรถประจำทาง เพื่อแจ้งให้พนักงานขับรถทราบว่ามีผู้โดยสารรออยู่ที่ป้ายรถประจำทางที่กำลังจะไปถึง โดยสัญญาณที่ผู้โดยสารกดสวิตช์จะถูกนำไปรวมกับสัญญาณของหมายเลขที่ส่งไปยังรถประจำทาง เมื่อรถประจำทางมาถึงที่ป้ายรถประจำทางก็จะส่งสัญญาณมายังป้ายรถประจำทาง เพื่อทำการเคลียร์หมายเลขที่ป้ายรถประจำทางแสดงอยู่ และเครื่องที่ป้ายรถประจำทางจะส่งสัญญาณตอบกลับ เพื่อแสดงว่าป้ายรถประจำทางได้รับรู้แล้วว่ารถประจำทางมาถึงป้ายรถประจำทางนี้ เพื่อที่รถประจำทางรู้ว่าได้ผ่านป้ายรถประจำทางนี้ไปแล้ว

5.2 สรุปผลการทดลองของเครื่องที่รถประจำทาง

ขณะที่รถประจำทางยังไม่เข้าไปในรัศมีการรับส่งข้อมูล เครื่องที่รถประจำทางจะไม่มีสัญญาณทั้งในส่วนของการรับและการส่ง เมื่อรถประจำทางเข้าไปในรัศมีการรับส่งข้อมูล กับป้ายรถประจำทางจะสามารถรับสัญญาณจากป้ายรถประจำทางได้ แล้วนำสัญญาณที่รับได้มาประมวลผล ถ้าหากข้อมูลที่รับมาตรงกับรหัสของรถประจำทาง รถประจำทางก็จะส่งสัญญาณตอบกลับไปยังป้ายรถประจำทาง เพื่อให้ป้ายรถประจำทางแสดงผลว่ารถประจำทางหมายเลขนี้กำลังจะเข้าไปที่ป้ายรถประจำทางนั้น หลังจากที่ป้ายรถประจำทางแสดงหมายเลขรถประจำทางที่กำลังจะเข้าป้ายรถประจำทางแล้ว ถ้าหากมีผู้โดยสารกดสวิตช์เพื่อรอรถประจำทางคันนี้อยู่ รถประจำทางก็จะได้รับสัญญาณเพื่อแสดงให้พนักงานขับรถทราบว่ามีผู้โดยสารรอขึ้นอยู่ที่ป้ายรถประจำทางที่จะถึง และเมื่อรถประจำทางถึงป้ายรถประจำทาง รถประจำทางก็จะส่งสัญญาณไปยังป้ายรถประจำทาง และรอป้ายรถประจำทางส่งสัญญาณตอบกลับมา เพื่อทำการบันทึกค่าว่าได้ผ่านป้ายรถประจำทางนี้มาแล้ว เพื่อจะตัดการรับส่งข้อมูลกับป้ายรถประจำทางที่ผ่านมา จนกว่าจะเข้ารัศมีการรับส่งข้อมูลกับป้ายรถประจำทางถัดไป

5.3 ปัญหาและการแก้ไข

- ข้อมูลผลิตภัณฑ์จากผู้ผลิตเอฟเอสเคโมดูลมีบางส่วนที่ไม่ถูกต้อง ทำให้ได้ผลการทำงานที่ผิดพลาด แก้ไขโดยการติดต่อสอบถามข้อมูลจากผู้ผลิตโดยตรง
- ในการรับส่งข้อมูลมีบางจังหวะที่ข้อมูลในรีจิสเตอร์รับข้อมูล มีการเลื่อนของข้อมูลทำให้การประมวลผลผิดพลาด
- แก้ไขโดยการเขียนโปรแกรมตรวจสอบข้อมูลและนำข้อมูลมาจัดเรียงใหม่ตามรูปแบบที่ใช้ในการรับส่ง

5.4 ข้อเสนอแนะและแนวทางการพัฒนา

- สร้างส่วนแสดงผลที่มีขนาดใหญ่ขึ้นสามารถมองเห็นได้ชัดเจน เนื่องจากเครื่องที่สร้างเป็นเครื่องต้นแบบจอแสดงผลจะมีขนาดเล็กจึงอาจมีผลต่อการมองเห็นเมื่อนำไปใช้งานจริง
- เพิ่มส่วนแสดงผลที่สามารถแสดงเป็นเสียง เพื่อประโยชน์ต่อผู้ที่พิการทางสายตา
- เพิ่มวงจรส่งสัญญาณเมื่อรถประจำทางมาถึงป้ายรถประจำทาง เพื่อรีเซ็ตข้อมูลของรถประจำทางจากป้ายรถประจำทาง และให้รถบันทึกค่าว่าได้ผ่านป้ายรถประจำทางไปแล้ว เพราะเครื่องต้นแบบที่สร้างกำหนดให้พนักงานขับรถเป็นผู้กดสวิตช์ส่งสัญญาณ

เอกสารอ้างอิง

1. ทีมงาน ETT, "คู่มือการทดลอง AVR AT90S8535 และ ATMEGA163 , บริษัท อีทีที จำกัด ฉบับที่ 1,หน้า 35 – 65, 2546

2. Richard Barnett, Larry O' Cull and Sarah Cox, "Embedded C Programming and the Atmel AVR", Thomson Delmar Learning, 2003

3. www.atmel.com

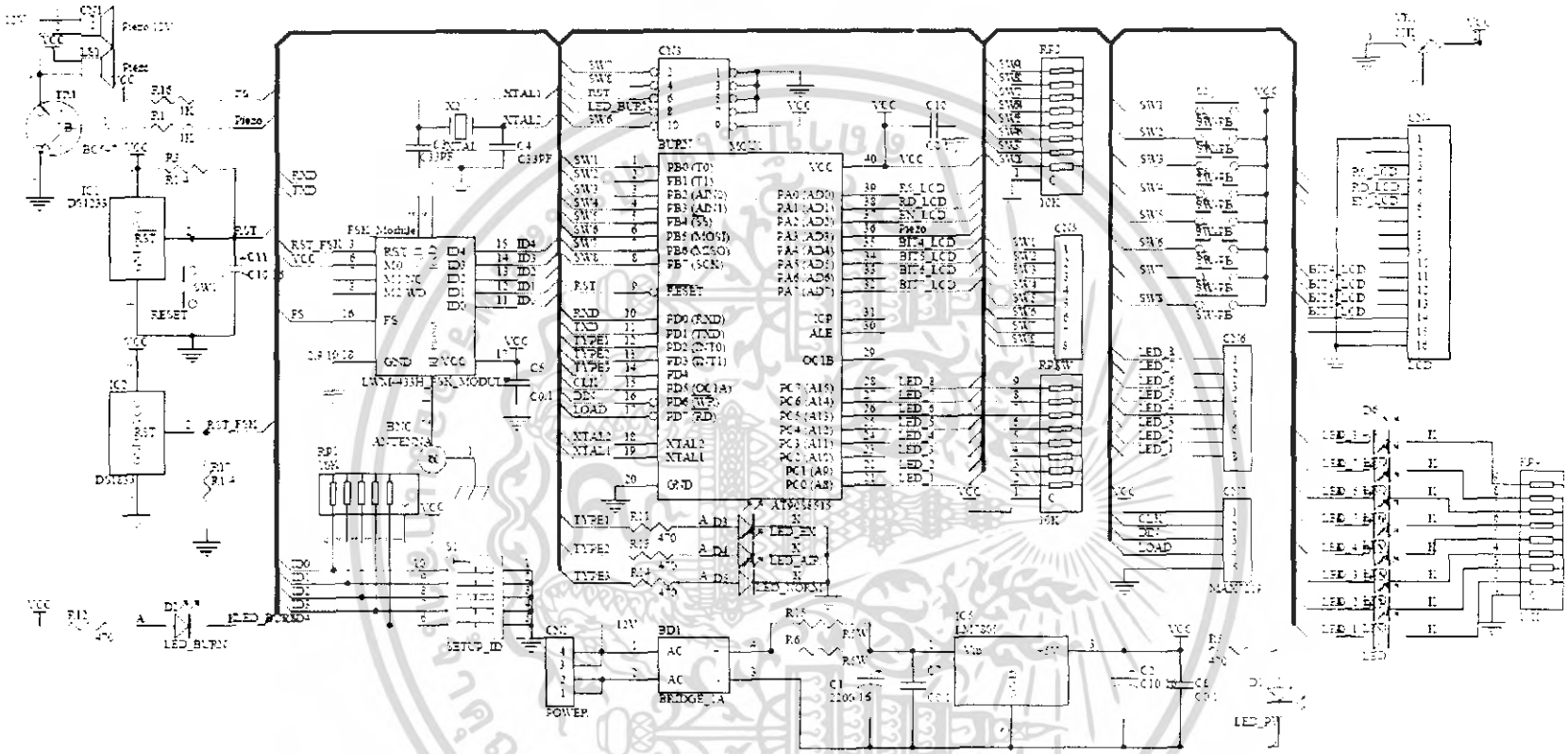
4. www.p-hz.com

ภาคผนวก ก.

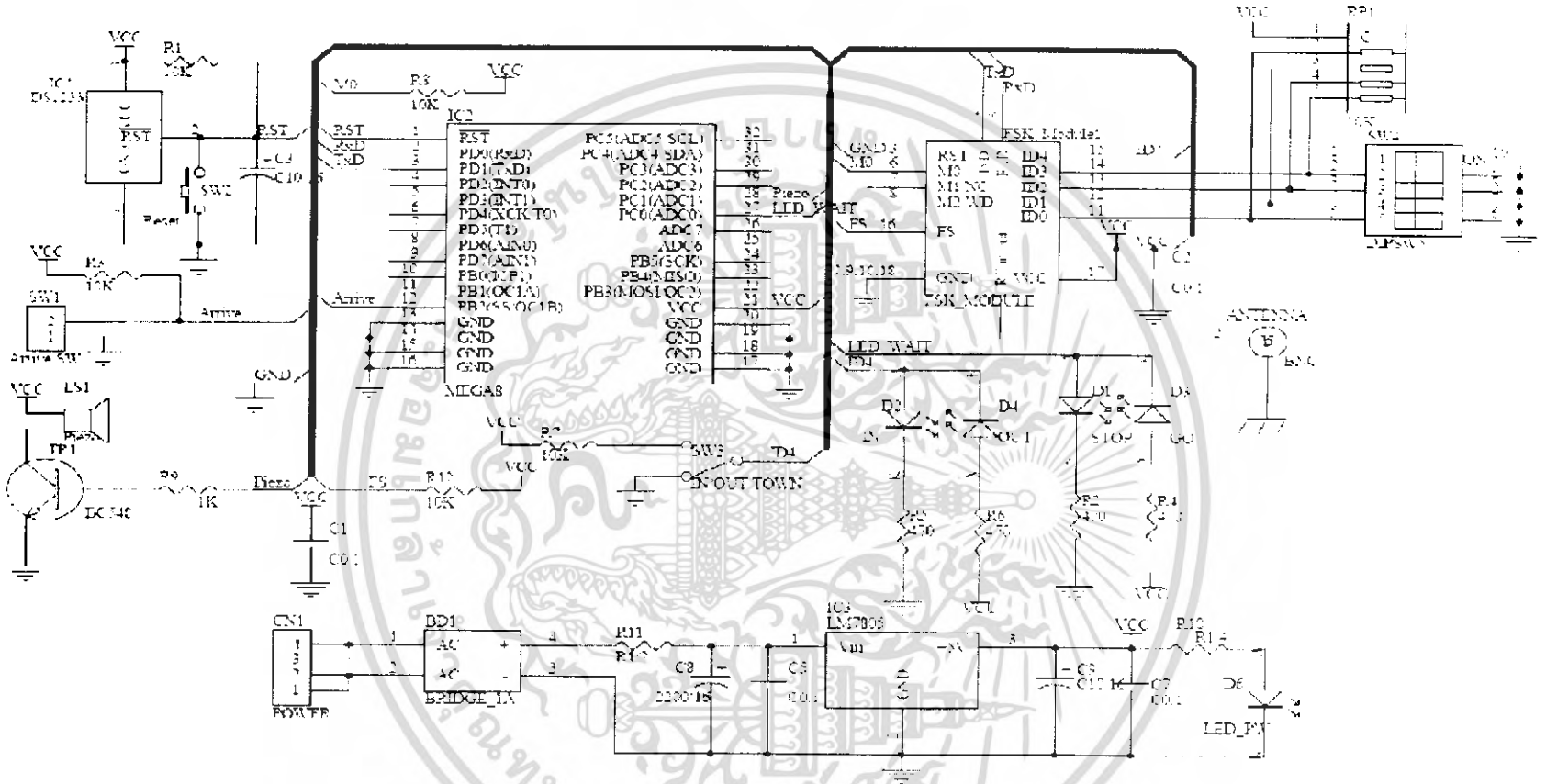
วงจรการใช้งาน

และ

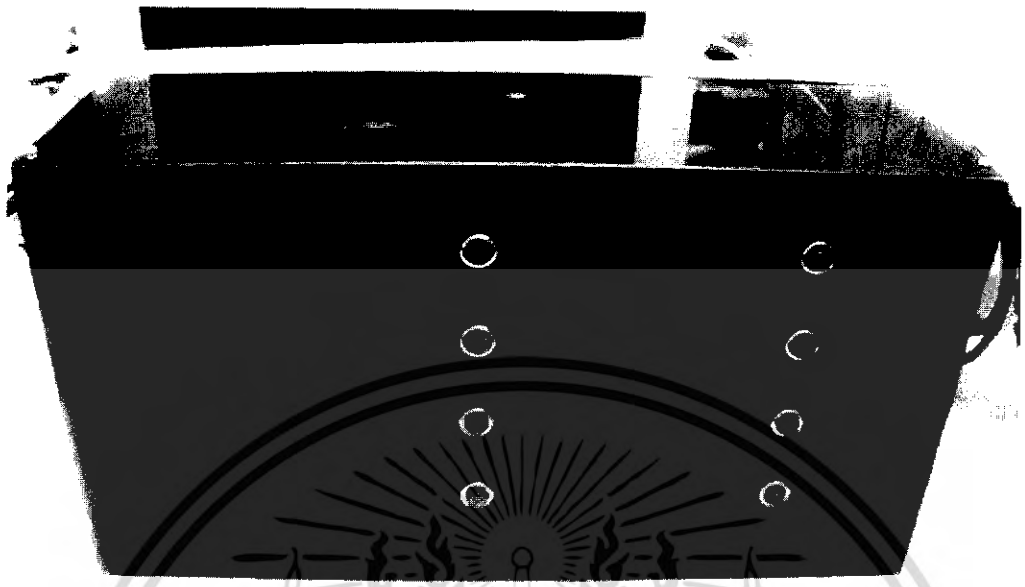
รูปเครื่องรับส่ง



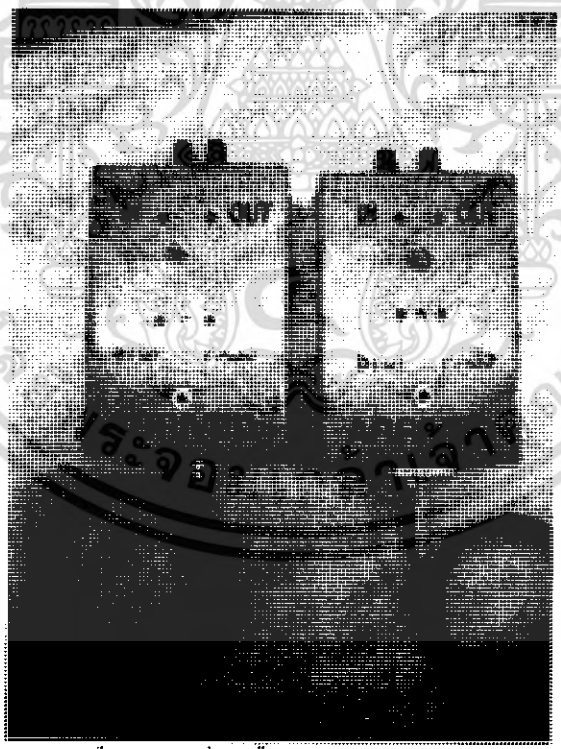
วงจรรวมของเครื่องรับส่งที่ป้ายรถประจำทาง



วงจรรวมของเครื่องรับส่งบนรถประจำทาง



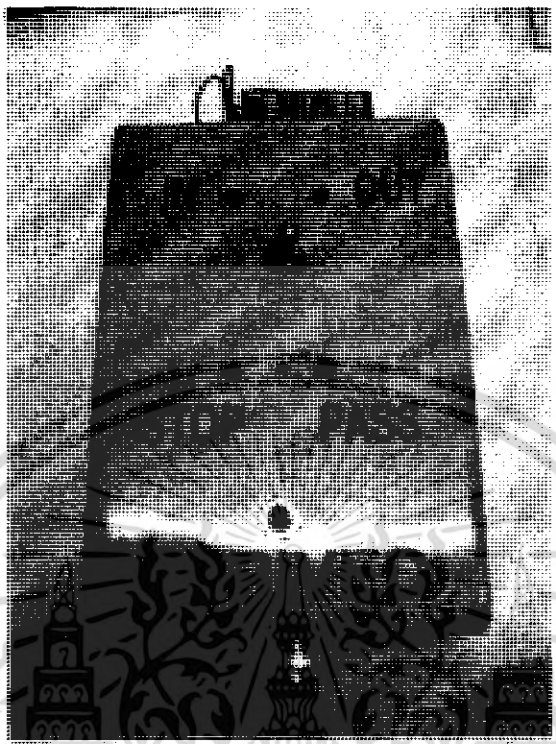
ป้ายแสดงสายรถประจำทาง



เครื่องรับส่งสัญญาณติดตั้งไว้บนรถประจำทาง

ป้ายแสดงสายรถประจำทาง แสดงหมายเลขรถที่ผ่านป้ายรถประจำทางทั้งหมด

สถานะไฟสีจากเครื่องเงินที่ผู้ขับรถประจำทาง



ขณะไฟ STOP แดงจะมีผู้โดยสารกดวิทย์ที่ป้าย

เมื่อรถประจำทางผ่านป้าย พนักงานเก็บค่าโดยสารกดวิทย์

ภาคผนวก ข.

โปรแกรม

โปรแกรมของเครื่องที่รถประจำทาง

```
/*****************************************************
Project: Bus Stop Management
Version: 1.0
Date    : 2/12/2006
Author  : MIXS
Company : KMITL
Comments:  Bus
Chip type           : ATmega8
Program type        : Application
Clock frequency     : 11.059200 MHz
Memory model        : Small
External SRAM size: 0
Data Stack size     : 256
*****************************************************/
#include <mega8.h>
// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Declare your global variables here
#define Arrive_SW PINB.2
#define LED_wait  PORTC.0
#define PIEZO     PORTC.1
char rx_A;
//code bus=6666,type:x= normal,$ for check station
//@ for check arrive,# for check wait,* for check confrim
unsigned char code[10]      = {"6666x$@#*"};
```

```
unsigned char rx[10]          {"000000000"};
unsigned char rx_clr[10]      {"000000000"};
unsigned char tx[10]          {"000000000"};
unsigned char tx2[10]         {"3333z$@a*"};        //code of Bus
unsigned char passed_st[7]    {"000000"};
signed char comp,comp_w,comp_st,comp_c;
//**********************Beeper Function*****************
void beeper()
{
   int b;

      for(b=0;b < 2;b++)
      {
         PIEZO =1;
         delay_ms(80);
         PIEZO =0;
         delay_ms(80);
      }

}
void beeper2()
{
         PIEZO =1;
         delay_ms(100);
         PIEZO =0;
}
void main(void)
{
// Declare your local variables here
// Input/Output Ports initialization
```

```
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;
// Port C initialization
// Func6=In Func5=In Func4=In Func3=In Func2=In Func1=Out Func0=Out
// State6=T State5=T State4=T State3=T State2=T State1=0 State0=0
PORTC=0x00;
DDRC=0x03;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
```

```
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
```

```
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x47;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

   LED_wait=1;          //Initial status
While (1)
   {
scan_data:                  //scan data
      strcpy(tx,code);
      scanf("%9s",rx);
      comp=strncmp(rx,code,5);    //check data
      if(comp != 0)
        { goto scan_data;
        }
//************Check passed station*****************
      comp_st = strncmp(rx,passed_st,6);
      if(comp_st == 0)
        {goto scan_data;}
      else{printf("%9s",tx);}
```

```
//*********************check_wait*****************
    comp_w = strpos(rx,'w');
    if(comp_w != -1)
      { LED_wait=0;          //LED STOP
       beeper();             //beeper
      }
//*********************chexk Arrive*****************
    if(Arrive_SW != 0)
     {goto scan_data;}
    else{ beeper2();
          LED_wait=0;         //LED GO
          rx_A=strpos(tx,'#'); //set arrive
          tx[rx_A]='a';
        }
send_data:  printf("%9s",tx);   //send data
//*****************wait & scan data***************
        delay_ms(10);
        scanf("%9s",rx);
//*******************check confirm*****************
        comp_c = strpos(rx,'c');
        if(comp_c != -1)
           {strncpy(passed_st,rx,6);
            strcpy(rx,rx_clr);   //clear rx
            LED_wait=1;          //clear LED wait(LED STOP)
            goto  scan_data;     //scan new data & new Bus Stop
           }
        else{goto send_data;}     //send data
    };
}
```

โปรแกรมของเครื่องป้ายรถประจำทาง

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.6 Professional
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com
Project : Bus Stop Program
Version : 4.0
Date    : 2/14/2006
Author  : MIXS
Company : KMITL
Comments: Bus Stop Management.


Chip type           : AT90S8515
Clock frequency     : 7.372800 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 128
*****************************************************/
#include <90s8515.h>
// Alphanumeric LCD Module functions
#asm
  .equ __lcd_port=0x1B ;PORTA
#endasm
#include <lcd.h>
#define RXB8 1
#define TXB8 0
```

```
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7
#define FRAMING_ERROR (1<<FE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
// UART Receiver buffer
#define RX_BUFFER_SIZE 9
char rx_buffer[RX_BUFFER_SIZE];
#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif
// This flag is set on UART Receiver buffer overflow
bit rx_buffer_overflow;
// UART Receiver interrupt service routine
interrupt [UART_RXC] void uart_rx_isr(void)
{  char status,data;
  status=USR;
  data=UDR;
  if ((status & (FRAMING_ERROR | DATA_OVERRUN))==0)
    {   rx_buffer[rx_wr_index]=data;
        if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
        if (++rx_counter == RX_BUFFER_SIZE)
            { rx_counter=0;
              rx_buffer_overflow=1;
            };
```

```
  };
}
#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the UART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{   char data;
    while (rx_counter==0);
    data=rx_buffer[rx_rd_index];
    if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
    #asm("cli")
    --rx_counter;
    #asm("sei")
    return data;
}
#pragma used-
#endif
// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Standard Input/Output functions
#define TYPE1 PORTD.2
#define TYPE2 PORTD.3
#define TYPE3 PORTD.4
int LED_wait,BUZZER,num_disp,ID_fix,ID_pass=0x0f,ID,i,num_key,t,b,k;
char rx_c;
```

```
unsigned char rx[10]          = {"000000000"};
unsigned char rx_buff[10]     = {"000000000"};
unsigned char tx[10]          = {"000000000"};
unsigned char data[10]        = {"000000000"};
unsigned char clr_code[10]    = {"000000000"};
unsigned char rx_in[6]  = {"00000"};
signed char comp,comp_a,comp_c,comp_rx_tx,comp_TYPE1,comp_TYPE2,comp_TYPE3;
//code bus=1111 to 8888,type:x = normal,type:y = Air Bus,type:z = Air Bus Express.
//A,B,C,...:Bus Stop station's name,@: for check arrive,# for set wait.* for set confrim
unsigned char data_base_code[8][10]  = {
"1111yA@#*",
"2222yA@#*",
"3333zA@#*",
"4444xA@#*",
"5555xA@#*",
"6666xA@#*",
"7777xA@#*",
"8888xA*#$"
};
//***********Function*****************************************
//**********************arrange data*****************************
void arrange_data()
{ unsigned char data_buff,data_new[10];
   strcpy(data_new,data);
   for(k=0;k<=9;k++)
      { if(data_new[8]!='*')
         { data_buff =data_new[8];
           data_new[8]=data_new[7];
           data_new[7]=data_new[6];
           data_new[6]=data_new[5];
```

```
            data_new[5]=data_new[4];
            data_new[4]=data_new[3];
            data_new[3]=data_new[2];
            data_new[2]=data_new[1];
            data_new[1]=data_new[0];
            data_new[0]=data_buff;
           // goto ckeck;
           }
         else break;
      }
  strncpy(rx,data_new,9);  //copy code from rx buffer
 }
//*************************************************************
void scan_data()
{    delay_ms(10);
     UCR=0x98;
     for(b=0;b<=9;b++)       //recieve data code from bus
       {data[b]=rx_buffer[b]; //to 'data' (buffer)
       }
       if(data[9]!='*')
         { arrange_data(); }
       else
       strncpy(rx,data,9);  //copy code from rx buffer
}
//**********************Beeper Function************************
void beeper3()
{           PORTA.3=1;
           delay_ms(50);
           PORTA.3=0;
           delay_ms(50);
```

```
}
void beeper2()
{       PORTA.3=1;
        delay_ms(100);
        PORTA.3=0;
}
void beeper()
{        PORTA.3=1;
        delay_ms(50);
        PORTA.3=0;
        delay_ms(50);
        PORTA.3=1;
        delay_ms(50);
        PORTA.3=0;
}
//**********set confirm*****************************************
void set_confirm()
{  rx_c=strpos(tx,'*');
   tx[rx_c]='c';
}
//**********clear display***************************************
void clear_display()
{   switch (t)
    {       case 0: {PORTC.0=0; beeper3();
            num_disp= 9;
            set_confirm();
            break;}
      case 1: {PORTC.1=0; beeper3();
            num_disp= 9;
            set_confirm();
```

```
            break; }
        case 2: {PORTC.2=0; beeper3();
            num_disp= 9;
            set_confirm();
            break; }
        case 3: {PORTC.3=0; beeper3();
            num_disp= 9;
            set_confirm();
            break; }
        case 4: {PORTC.4=0; beeper3();
            num_disp= 9;
            set_confirm();
            break; }
        case 5: {PORTC.5=0; beeper3();
            num_disp= 9;
            set_confirm();
            break; }
        case 6: {PORTC.6=0; beeper3();
            num_disp= 9;
            set_confirm();
            break; }
        case 7: {PORTC.7=0; beeper3();
            num_disp= 9;
            set_confirm();
            break; }
        }
}
//**********show display**************************************
void show_display()
{   switch (t)
```

```
}     case 0: {if (PINC.0 ==0){beeper();PORTC.0 = 1; }
      PORTC.0 = 1;
            break;}
      case 1: {if (PINC.1 ==0){beeper();PORTC.1 = 1; }
      PORTC.1 = 1;   break;}
      case 2: {if (PINC.2 ==0){beeper();PORTC.2 = 1; }
      PORTC.2 = 1;   break;}
    case 3: {if (PINC.3 ==0){beeper();PORTC.3 = 1; }
      PORTC.3 = 1;    break;}
    case 4: {if (PINC.4 ==0){beeper();PORTC.4 = 1; }
      PORTC.4 = 1;   break;}
    case 5: {if (PINC.5 ==0){beeper();PORTC.5 = 1; }
      PORTC.5 = 1;   break;}
    case 6: {if (PINC.6 ==0){beeper();PORTC.6 = 1; }
      PORTC.6 = 1;    break;}
    case 7: {if (PINC.7 ==0){beeper();PORTC.7 = 1; }
      PORTC.7 = 1;   break;}
    case 9: break;
    }
}
//********set wait******** **********************************
void set_wait()
{  char rx_key;
   rx_key=strpos(rx,'#');
   rx[rx_key]='w';
}
//*************scan key wait************************************
void scan_key_wait(int n)
{   switch(n)
```

```
{ case 0: { if(PINB.0 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper2();
                }
                break;
            }
 case 1: { if(PINB.1 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper2();
                }
                break;
            }
case 2: { if(PINB.2 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper2();
                }
                break;
            }
case 3: { if(PINB.3 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper2();
                }
                break;
            }
case 4: { if(PINB.4 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper2();
                }
                break;
            }
case 5: { if(PINB.5 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper2();
                }
                break;}
case 6: { if(PINB.6 == 1)
```

```
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper20;
                }
                break;
            }
    case 7: { if(PINB.7 == 1)
                {strcpy(tx,data_base_code[num_key]); set_wait();  beeper20;
                }
                break;}
    case 9:  break;
    }
}
//*****************Main Program*****************************************
void main(void)
{
// Declare your local variables here
// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=0 State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x08;
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;
// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTC=0x00;
```

```
DDRC=0xFF;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=Out Func3=Out Func2=Out Func1=In Func0=In
// State7=T State6=T State5=T State4=0 State3=0 State2=0 State1=T State0=T
PORTD=0x00;
DDRD=0x1C;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
```

```
OCR1BL=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
GIMSK=0x00;
MCUCR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
// UART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// UART Receiver: On
// UART Transmitter: On
// UART Baud rate: 9600
UCR=0x98;
UBRR=0x2F;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
// LCD module initialization
lcd_init(16);
// Global enable interrupts
#asm("sei")
//**********************TEST LED**********************************
        PORTC.0=1;
       delay_ms(50);
        PORTC.1=1;
       delay_ms(50);
        PORTC.2=1;
       delay_ms(50);
```

```
        PORTC.3=1;
        delay_ms(50);
        PORTC.4=1;
        delay_ms(50);
        PORTC.5=1;
        delay_ms(50);
        PORTC.6=1;
        delay_ms(50);
        PORTC.7=1;
        delay_ms(50);
        PORTC.0=0;
        delay_ms(50);
        PORTC.1=0;
        delay_ms(50);
        PORTC.2=0;
        delay_ms(50);
        PORTC.3=0;
        delay_ms(50);
        PORTC.4=0;
        delay_ms(50);
        PORTC.5=0;
        delay_ms(50);
        PORTC.6=0;
        delay_ms(50);
        PORTC.7=0;
        delay_ms(50);
//*****************************************************************************
while(1)
  {
//**********Initail*************************************************************
```

```
    clear_display();
    //lcd_clear();
    strncpy(rx_buffer,clr_code,9);
    TYPE1=0;
    TYPE2=0;
    TYPE3=0;
     for (t=0;t<8;t++)                  //t=address in data base
        {
//************send data (Tx)*********************************************
        strncpy(data,clr_code,9);
        printf("%9s\t",data_base_code[t]); //send code to bus
        strcpy(tx,data_base_code[t]);     //tx = data sent
        delay_ms(100);                  //wait data
     //lcd_gotoxy(12,0);                //LCD View Code
     //lcd_puts(tx);
//************Scan Data >>>Rx<<<*********************************
       scan_data();                     //scan data
     //lcd_gotoxy(0,0);                 //LCD View Code
     //lcd_puts(rx);
//++++++++++++++++++++TEST TYPE+++++++++++++++++++++-+++-++-++-++
     comp_TYPE1 = strrpos(rx,'x');    //Type normal bus
        if (comp_TYPE1 != -1)
           { TYPE1=1;}
     comp_TYPE2 = strrpos(rx,'y');    //Type Air bus
        if (comp_TYPE2 != -1)
           { TYPE2=1; }
     comp_TYPE3 = strrpos(rx,'z');    //Type Express bus
        if (comp_TYPE3 != -1)
           { TYPE3=1; }
//************************Check code send = scan ? ************
```

```
        comp_rx_tx = strncmp(rx,tx,5); //check rx = tx ?
        if (comp_rx_tx != 0)
          { goto next; }            //new code to send
//*****************************************************************
        delay_ms(1);
        scan_data();
//*********************Check arrive bus code*********************
        comp_a = strrpos(rx,'a');  //check arrive bus?
        if(comp_a != -1)
          { clear_display();    //clear Display & set confirm
            rx_c=strpos(tx,'*'); //Set confirm
            tx[rx_c]='c';
            printf("%9s",tx);   //send confirm
            beeper();           //Beeper
            goto next;          //next code
           }
        else{ num_disp=t;
            show_display();     //show coming bus
            scan_key_wait(t);   //check wait key
            strcpy(tx,rx);
           }
//*******************send_code:********************************
        printf("%9s",tx);           //send code
        //lcd_gotoxy(4,1);          //LCD View Code
next:
        }
    };
}
```

ภาคผนวก ค.

Data Sheet

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 130 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 8K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 1K Byte Internal SRAM
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Three PWM Channels
  - 8-channel ADC in TQFP and QFN/MLF package
    Eight Channels 10-bit Accuracy
  - 6-channel ADC in PDIP package
    Eight Channels 10-bit Accuracy
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-lead PDIP, 32-lead TQFP, and 32-pad QFN/MLF
- Operating Voltages
  - 2.7 - 5.5V (ATmega8L)
  - 4.5 - 5.5V (ATmega8)
- Speed Grades
  - 0 - 8 MHz (ATmega8L)
  - 0 - 16 MHz (ATmega8)
- Power Consumption at 4 Mhz, 3V, 25°C
  - Active: 3.6 mA
  - Idle Mode: 1.0 mA
  - Power-down Mode: 0.5 µA

ATMEL®

8-bit AVR® with 8K Bytes In-System Programmable Flash

ATmega8
ATmega8L

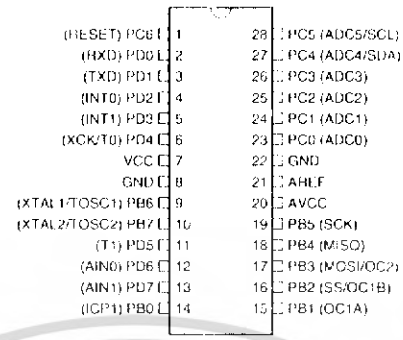
Summary

2486PS–AVR–02/0€

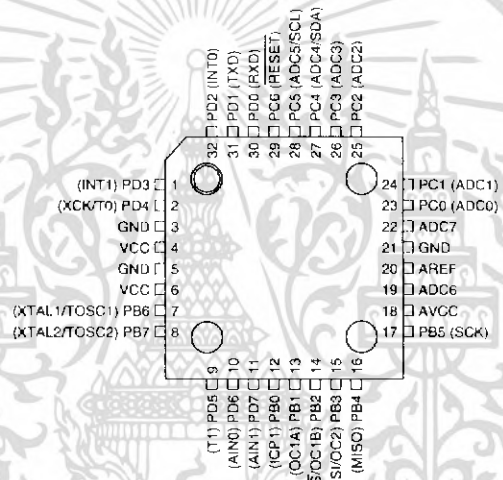
ATMEL

Pin Configurations

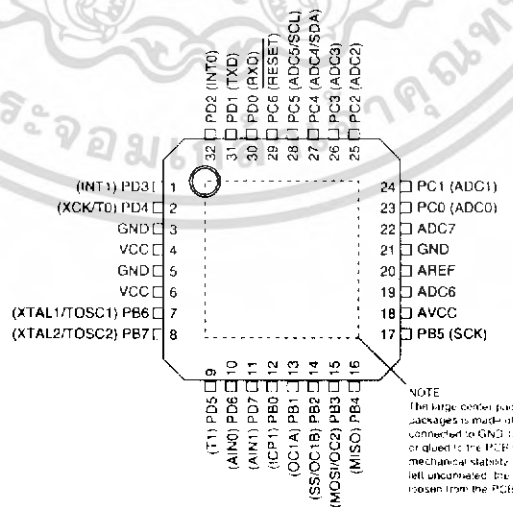
PDIP



TQFP Top View



MLF Top View



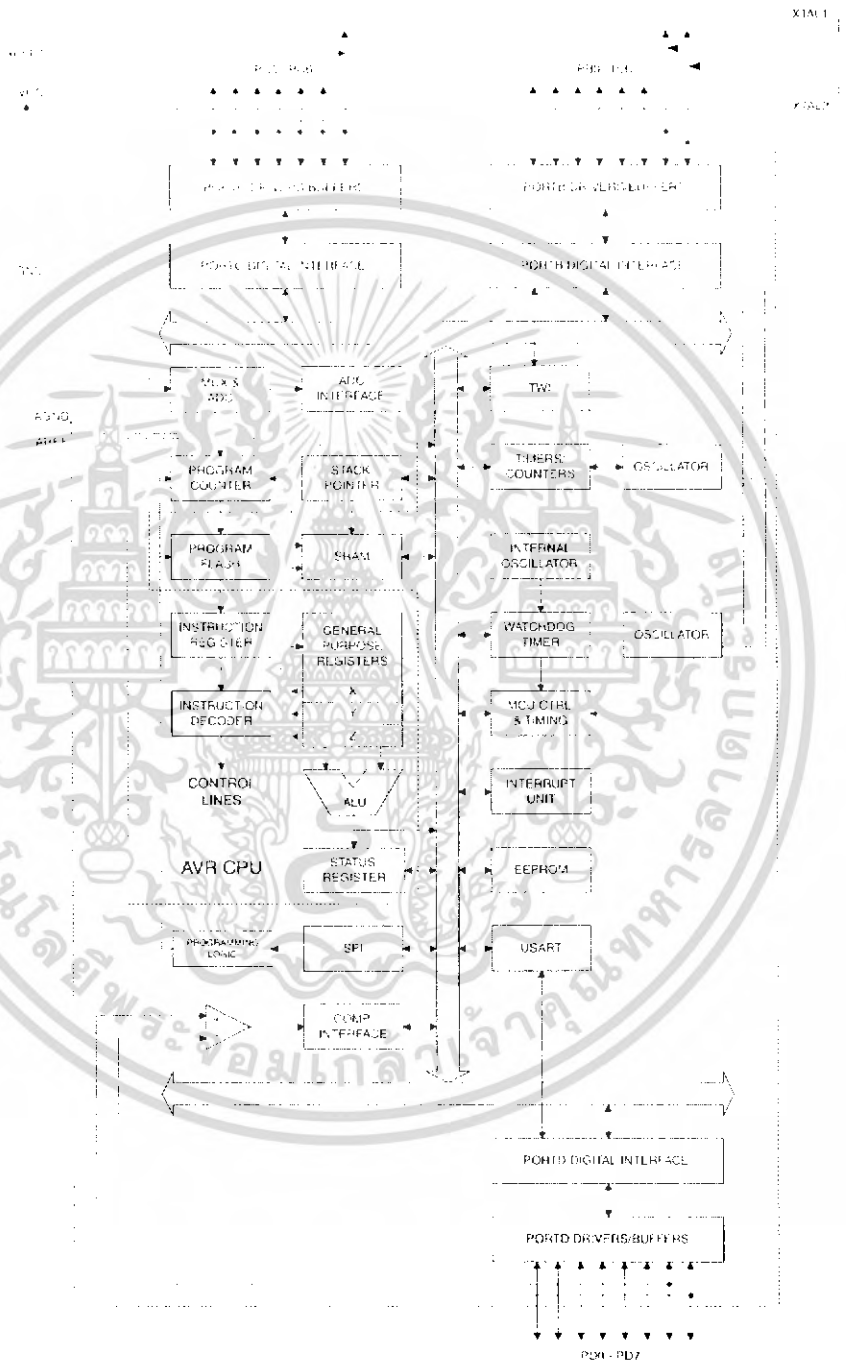
NOTE: The large center pad underneath the MLF package is made of metal and internally connected to GND. It should be soldered or glued to the PCB to ensure good mechanical stability. If the center pad is left unsoldered, the package might loosen from the PCB.

Overview

The ATmega8 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8 achieves throughputs approaching 1 MIPS per MHz, allowing the system designer to optimize power consumption versus processing speed.

Block Diagram

Figure 1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega8 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1K byte of SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, a 6-channel ADC (eight channels in TQFP and QFN/MLF packages) with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next Interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption.

The device is manufactured using Atmel's high density non-volatile memory technology. The Flash Program memory can be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash Section will continue to run while the Application Flash Section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega8 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega8 AVR is supported with a full suite of program and system development tools, including C compilers, macro assemblers, program debugger/simulators, In-Circuit Emulators, and evaluation kits.

:claimer

Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

Pin Descriptions

VCC

Digital supply voltage.

GND

Ground.

Port B (PB7..PB0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in "Alternate Functions of Port B" on page 56 and "System Clock and Clock Options" on page 23.

Port C (PC5..PC0)

Port C is an 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 36. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated on page 59.

Port D (PD7..PD0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega8 as listed on page 61.

$\overline{\text{RESET}}$

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 36. Shorter pulses are not guaranteed to generate a reset.

AV_{CC}	AV_{CC} is the supply voltage pin for the A/D Converter, Port C (3..0), and ADC (7..6). It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that Port C (5..4) use digital supply voltage, V_{CC}.
AREF	AREF is the analog reference pin for the A/D Converter.
ADC7..6 (TQFP and QFN/MLF Package Only)	In the TQFP and QFN/MLF package, ADC7..6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

Resources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

gister Summary

ddress	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	9
3E (0x5E)	SPH	–	–	–	–	–	SP10	SP9	SP8	11
3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
3C (0x5C)	Reserved									
3B (0x5B)	GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE	47, 65
3A (0x5A)	GIFR	INTF1	INTF0	–	–	–	–	–	–	66
39 (0x59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	79, 100, 120
38 (0x58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	71, 101, 120
37 (0x57)	SPMCR	SPMIE	RWWSB	–	PGWRT	BLBSET	PGERS	SPMEN		211
36 (0x56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	169
35 (0x55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	31, 64
34 (0x54)	MCUCSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	39
33 (0x53)	TCCR0	–	–	–	–	–	CS02	CS01	CS00	70
32 (0x52)	TCNT0	Timer/Counter0 (8 Bits)								70
31 (0x51)	OSCCAL	Oscillator Calibration Register								29
30 (0x50)	SFIOR	–	–	–	–	ACME	PUD	PSR2	PSR10	56, 73, 121, 191
2F (0x4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	95
2E (0x4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	98
2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High byte								99
2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low byte								99
2B (0x4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High byte								99
2A (0x4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low byte								99
29 (0x49)	OCR1BH	Timer/Counter1 – Output Compare Register B High byte								99
28 (0x48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low byte								99
27 (0x47)	ICR1H	Timer/Counter1 – Input Capture Register High byte								100
26 (0x46)	ICR1L	Timer/Counter1 – Input Capture Register Low byte								100
25 (0x45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	115
24 (0x44)	TCNT2	Timer/Counter2 (8 Bits)								117
23 (0x43)	OCR2	Timer/Counter2 Output Compare Register								117
22 (0x42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	117
21 (0x41)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	41
20[1] (0x40)[1]	UBRRH	URSEL	–	–	–	UBRR[11:8]				156
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	154
1F (0x3F)	EEARH	–	–	–	–	–	–	–	EEAR8	18
1E (0x3E)	EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	18
1D (0x3D)	EEDR	EEPROM Data Register								18
1C (0x3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	18
1B (0x3B)	Reserved									
1A (0x3A)	Reserved									
19 (0x39)	Reserved									
18 (0x38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	63
17 (0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	63
16 (0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	63
15 (0x35)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	63
14 (0x34)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	63
13 (0x33)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	63
12 (0x32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	63
11 (0x31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	63
10 (0x30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
0F (0x2F)	SPDR	SPI Data Register								129
0E (0x2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	129
0D (0x2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	127
0C (0x2C)	UDR	USART I O Data Register								151
0B (0x2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	152
0A (0x2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	153
09 (0x29)	UBRRL	USART Baud Rate Register Low byte								156
08 (0x28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	192
07 (0x27)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	203
06 (0x26)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	205
05 (0x25)	ADCH	ADC Data Register High byte								206
04 (0x24)	ADCL	ADC Data Register Low byte								206
03 (0x23)	TWDR	Two-wire Serial Interface Data Register								171
02 (0x22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	171

Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x01 (0x21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS1	TWPS0	171
0x00 (0x20)	TWBR	Two-wire Serial Interface Bit Rate Register								169

Notes:
1. Refer to the USART description for details on how to access UBRRH and UCSRC.
2. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
3. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

nstruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd − Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd − Rr − C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd - K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**

truction Set Summary (Continued)

	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
TRANSFER INSTRUCTIONS					
	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
W	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
	Rd, K	Load Immediate	Rd ← K	None	1
	Rd, X	Load Indirect	Rd ← (X)	None	2
	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
	Rd, Y	Load Indirect	Rd ← (Y)	None	2
	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
	Rd, Z	Load Indirect	Rd ← (Z)	None	2
	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
	X, Rr	Store Indirect	(X) ← Rr	None	2
	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
	Y, Rr	Store Indirect	(Y) ← Rr	None	2
	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
	Z, Rr	Store Indirect	(Z) ← Rr	None	2
	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
		Load Program Memory	R0 ← (Z)	None	3
	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
		Store Program Memory	(Z) ← R1:R0	None	-
	Rd, P	In Port	Rd ← P	None	1
	P, Rr	Out Port	P ← Rr	None	1
H	Rr	Push Register on Stack	STACK ← Rr	None	2
	Rd	Pop Register from Stack	Rd ← STACK	None	2
ND BIT-TEST INSTRUCTIONS					
	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
	Rd	Rotate Left Through Carry	Rd(0)← C,Rd(n+1)← Rd(n),C← Rd(7)	Z,C,N,V	1
	Rd	Rotate Right Through Carry	Rd(7)← C,Rd(n)← Rd(n+1),C← Rd(0)	Z,C,N,V	1
	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
P	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
I	s	Flag Set	SREG(s) ← 1	SREG(s)	1
R	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
		Set Carry	C ← 1	C	1
		Clear Carry	C ← 0	C	1
		Set Negative Flag	N ← 1	N	1
		Clear Negative Flag	N ← 0	N	1
		Set Zero Flag	Z ← 1	Z	1
		Clear Zero Flag	Z ← 0	Z	1
		Global Interrupt Enable	I ← 1	I	1
		Global Interrupt Disable	I ← 0	I	1
		Set Signed Test Flag	S ← 1	S	1
		Clear Signed Test Flag	S ← 0	S	1
		Set Twos Complement Overflow	V ← 1	V	1
		Clear Twos Complement Overflow	V ← 0	V	1
		Set T in SREG	T ← 1	T	1
nemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**

Instruction Set Summary (Continued)

CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1

Features

- AVR - High Performance and Low Power RISC Architecture
- 118 Powerful Instructions - Most Single Clock Cycle Execution
- 8K bytes of In-System Reprogrammable Flash
  - SPI Serial Interface for Program Downloading
  - Endurance: 1,000 Write/Erase Cycles
- 512 bytes EEPROM
  - Endurance: 100,000 Write/Erase Cycles
- 512 bytes Internal SRAM
- 32 x 8 General Purpose Working Registers
- 32 Programmable I/O Lines
- Programmable Serial UART
- SPI Serial Interface
- V_{CC}: 2.7 - 6.0V
- Fully Static Operation
  - 0 - 8 MHz 4.0 - 6.0V,
  - 0 - 4 MHz 2.7 - 4.0V
- Up to 8 MIPS Throughput at 8 MHz
- One 8-Bit Timer/Counter with Separate Prescaler
- One 16-Bit Timer/Counter with Separate Prescaler and Compare and Capture Modes
- Dual PWM
- External and Internal Interrupt Sources
- Programmable Watchdog Timer with On-Chip Oscillator
- On-Chip Analog Comparator
- Low Power Idle and Power Down Modes
- Programming Lock for Software Security

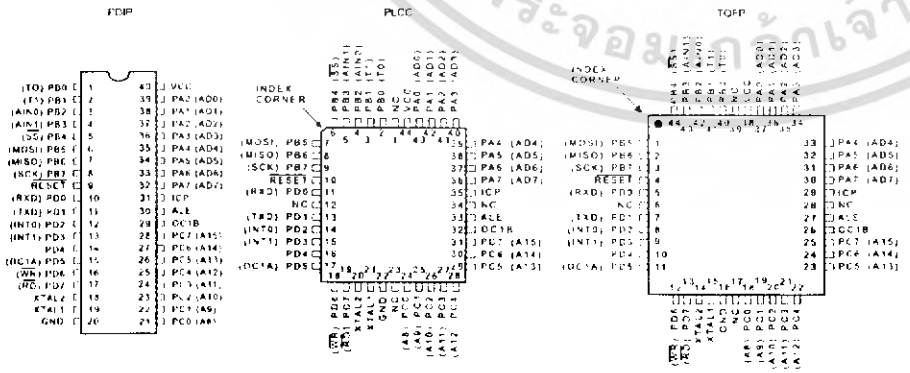
Description

The AT90S8515 is a low-power CMOS 8-bit microcontroller based on the AVR ® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the AT90S8515 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

(continued)

Pin Configurations



8-Bit AVR® Microcontroller with 8K bytes In-System Programmable Flash

AT90S8515

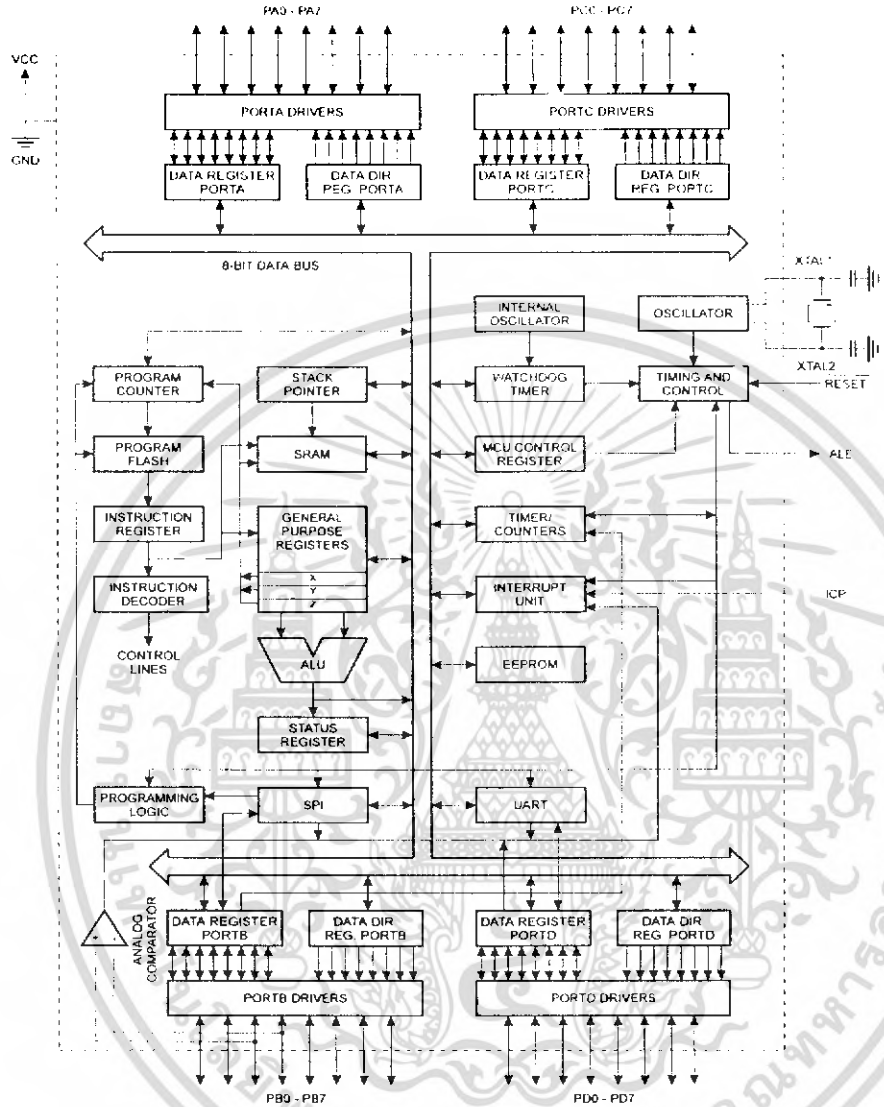
Preliminary

Rev. 0841DS–06/98

Note: This is a sumary document. For the complete 76 page document, please visit our Web site at *www.atmel.com* or e-mail at *literature@atmel.com* and request literature number 0841D.

Block Diagram

Figure 1. The AT90S8515 Block Diagram



The AT90S8515 provides the following features: 8K bytes of In-System Programmable Flash, 512 bytes EEPROM, 512 bytes SRAM, 32 general purpose I/O lines, 32 general purpose working registers, flexible timer/counters with compare modes, internal and external interrupts, a programmable serial UART, programmable Watchdog Timer with internal oscillator, an SPI serial port and two software selectable power saving modes. The Idle Mode stops the CPU while allowing the SRAM, timer/counters, SPI port and interrupt system to continue functioning. The power down mode saves the register contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset.

The device is manufactured using Atmel's high density non-volatile memory technology. The on-chip in-system programmable Flash allows the program memory to be reprogrammed in-system through an SPI serial interface or by a conventional nonvolatile memory programmer. By combining an enhanced RISC 8-bit CPU with In-System Programmable Flash on a monolithic chip, the Atmel AT90S8515 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The AT90S8515 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Pin Descriptions

V_{CC}
Supply voltage

GND
Ground

Port A (PA7..PA0)
Port A is an 8-bit bidirectional I/O port. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers can sink 20mA and can drive LED displays directly. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated.

Port A serves as Multiplexed Address/Data input/output when using external SRAM.

Port B (PB7..PB0)
Port B is an 8-bit bidirectional I/O pins with internal pull-up resistors. The Port B output buffers can sink 20 mA. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated.

Port B also serves the functions of various special features of the AT90S8515 as listed on page 46.

Port C (PC7..PC0)
Port C is an 8-bit bidirectional I/O port with internal pull-up resistors. The Port C output buffers can sink 20 mA. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated.

Port C also serves as Address output when using external SRAM.

Port D (PD7..PD0)
Port D is an 8-bit bidirectional I/O port with internal pull-up resistors. The Port D output buffers can sink 20 mA. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated.

Port D also serves the functions of various special features of the AT90S8515 as listed on page 52.

$\overline{\text{RESET}}$
Reset input. A low on this pin for two machine cycles while the oscillator is running resets the device.

XTAL1
Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2
Output from the inverting oscillator amplifier

ICP
ICP is the input pin for the Timer/Counter1 Input Capture function.

OC1B
OC1B is the output pin for the Timer/Counter1 Output CompareB function

ALE
ALE is the Address Latch Enable used when the External Memory is enabled. The ALE strobe is used to latch the low-order address (8 bits) into an address latch during the first access cycle, and the AD0-7 pins are used for data during the second access cycle.

Crystal Oscillator

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 2. Either a quartz crystal or a ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 3.

Figure 2. Oscillator Connections

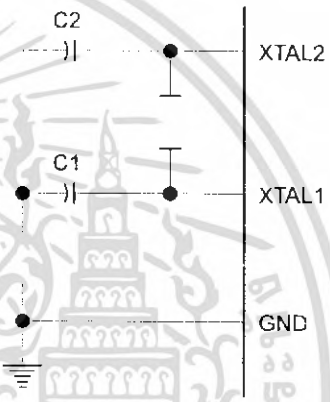
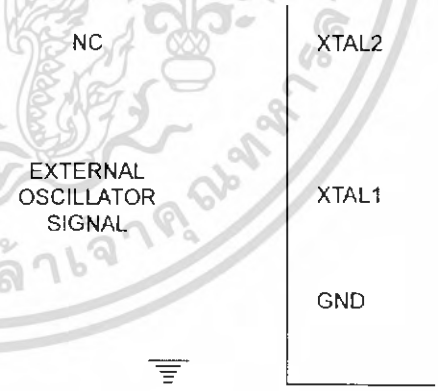


Figure 3. External Clock Drive Configuration

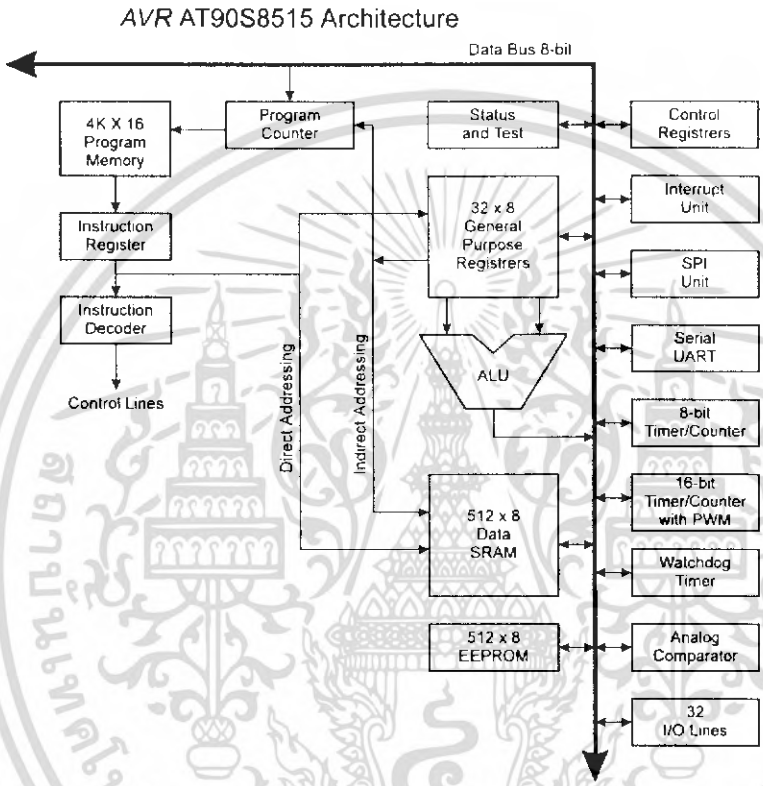


AT90S8515 Architectural Overview

The fast-access register file concept contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This means that during one single clock cycle, one ALU (Arithmetic Logic Unit) operation is executed. Two operands are output from the register file, the operation is executed, and the result is stored back in the register file - in one clock cycle.

Six of the 32 registers can be used as three 16-bits indirect address register pointers for Data Space addressing - enabling efficient address calculations. One of the three address pointers is also used as the address pointer for the constant table look up function. These added function registers are the 16-bits X-register, Y-register and Z-register.

Figure 4. The AT90S8515 *AVR* Enhanced RISC Architecture



The ALU supports arithmetic and logic functions between registers or between a constant and a register. Single register operations are also executed in the ALU. Figure 4 shows the AT90S8515 AVR Enhanced RISC microcontroller architecture.

In addition to the register operation, the conventional memory addressing modes can be used on the register file as well. This is enabled by the fact that the register file is assigned the 32 lowermost Data Space addresses ($00 - $1F), allowing them to be accessed as though they were ordinary memory locations.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, Timer/Counters, A/D-converters, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the register file, $20 - $5F.

The *AVR* uses a Harvard architecture concept - with separate memories and buses for program and data. The program memory is executed with a two stage pipeline. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is in-system programmable Flash memory.

With the relative jump and call instructions, the whole 4K address space is directly accessed. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

During interrupts and subroutine calls, the return address program counter (PC) is stored on the stack. The stack is effectively allocated in the general data SRAM, and consequently the stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initial-

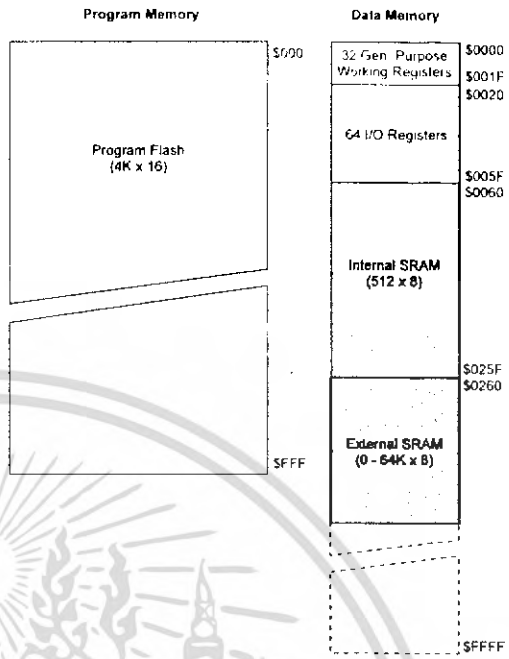
ize the SP in the reset routine (before subroutines or interrupts are executed). The 16-bit stack pointer SP is read/write accessible in the I/O space.

The 512 bytes data SRAM can be easily accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the status register. All the different interrupts have a separate interrupt vector in the interrupt vector table at the beginning of the program memory. The different interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address the higher the priority.

Figure 5. Memory Maps



T90S8515 Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	18
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	19
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	19
$3C ($5C)	Reserved									
$3B ($5B)	GIMSK	INT1	INT0	-	-	-	-	-	-	24
$3A ($5A)	GIFR	INTF1	INTF0							24
$39 ($59)	TIMSK	TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	-	24
$38 ($58)	TIFR	TOV1	OCF1A	OCF1B	-	ICF1	-	TOV0	-	25
$37 ($57)	Reserved									
$36 ($56)	Reserved									
$35 ($55)	MCUCR	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00	26
$34 ($54)	Reserved									
$33 ($53)	TCCR0	-	-	-	-	-	CS02	CS01	CS00	29
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)								30
$31 ($51)	Reserved									
$30 ($50)	Reserved									
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	PWM11	PWM10	32
$2E ($4E)	TCCR1B	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10	33
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte								34
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								34
$2B ($4B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								35
$2A ($4A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								35
$29 ($49)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								35
$28 ($48)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								35
$27 ($47)	Reserved									
$26 ($46)	Reserved									
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								36
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								36
$23 ($43)	Reserved									
$22 ($42)	Reserved									
$21 ($41)	WDTCR	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0	38
$20 ($40)	Reserved									
$1F ($3F)	Reserved	-	-	-	-	-	-	-	EEAR8	39
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								39
$1D ($3D)	EEDR	EEPROM Data Register								39
$1C ($3C)	EECR	-	-	-	-	-	EEMWE	EEWE	EERE	40
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	54
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	54
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	54
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	56
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	56
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	56
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	61
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	61
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	61
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	63
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	63
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
$0F ($2F)	SPDR	SPI Data Register								45
$0E ($2E)	SPSR	SPIF	WCOL	-	-	-	-	-	-	44
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	44
$0C ($2C)	UDR	UART I/O Data Register								48
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR	-	-	-	48
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	49
$09 ($29)	UBRR	UART Baud Rate Register								51
$08 ($28)	ACSR	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	52
...	Reserved									
$00 ($20)	Reserved									

AT90S8515 Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2

AT90S8515 Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	3
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1

LWM-433H

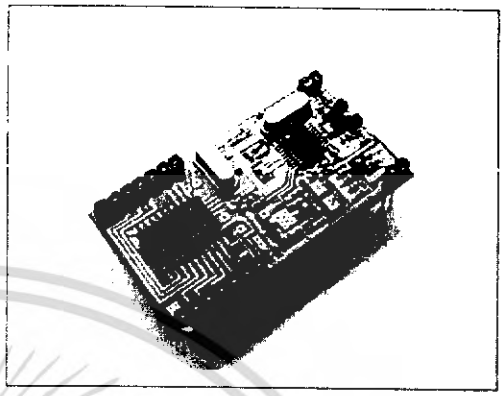
433.93/434.33 Dual Channel FSK Data Transceiver Module

Specifications and information are subject to change without notice. **February 2005 rev 1.1**

pHz Communications Co., Ltd.
9/204 Moo 1 Buatongkcha Bangbuatong Nonthaburi 11110 THAILAND
Tel: +66-0-2925-8563, +66-0-2925-8565 Fax: 66-0-2925-8564 E-mail: phz@p-Hz.com

LWM-433H
433.93/434.33 MHz Dual Channel FSK Data Transceiver Module

The LWM-433H is a miniature PCB mounting FSK data transceiver module. The LWM-433H is a dual channel data transceiver module designed to operate in 433 MHz ISM frequency band. The module enables the simple implementation of a data link at 5 mW transmitted power and data rate of 4800 or 9600 bit/s. LWM-434H module enables half-duplex directional wireless connectivity using 5V battery powered or handheld applications.

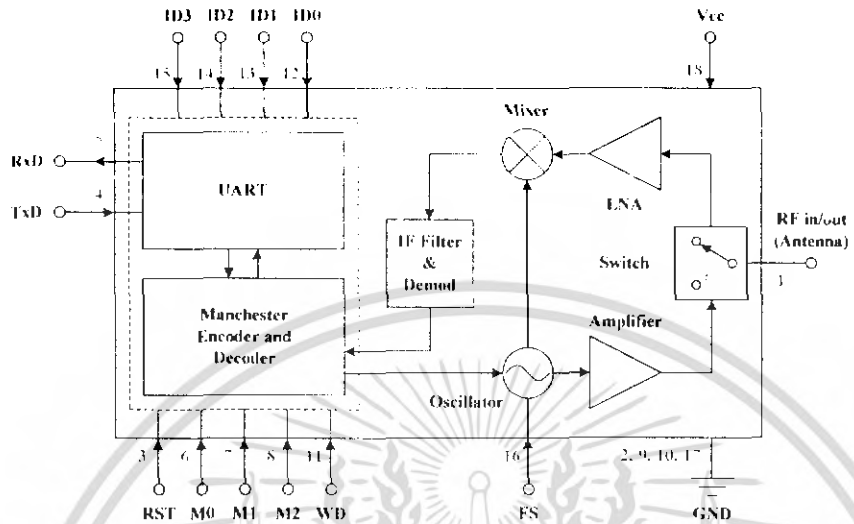


Features

- 433.93 or 434.33 MHz version
- Frequency Shift Keying (FSK) transceiver for digital data transmission
- Operate with 5 VDC supply (*Vcc*)
- 5 mW (+7 dBm) output power @ 5 VDC supply
- Low current consumption < 25 mA @ 5 VDC supply
- -105 dBm receiver's sensitivity @ supply = 5 VDC, BR = 9600 bit/s and BER < 10^{-3}
- Standard UART data connectivity
- Data rate 4800 or 9600 bit/s in packet mode
- Continues data up to 4000 bytes per packet in package mode
- Data rate 1200 to 9600 bit/s in direct mode (not available now)
- Usable range up to 500 meters in open ground (used with AMB-434UM)
- Crystal based oscillator
- High frequency stability with frequency error < 100 ppm
- Small size and very low cost
- Enable facility

The LWM-433H module is a high performance-data transceiver module. With high efficiency and high data rate, the modules will suit one-to-one and multi-node wireless links. Because of their small size, high performance and low voltage requirement a module is ideal for use in portable, battery-powered applications such as alarm and security system, automatic meter reading (AMR), home automation, remote control, surveillance, wireless communication, EPOS and inventory tracking, remote industrial process monitoring and computer networking.

pHz Communications Co., Ltd.
9/204 Moo 1 Buatongkeha Bangbuatong Nonthaburi 11110 THAILAND
Tel: +66-0-2925-8563, +66-0-2925-8565 Fax: 66-0-2925-8564 E-mail: phz@p-Hz.com



Block diagram

Pin description:

Vcc *(pin 17)* — DC power supply at 5.0 VDC

ID4 – ID0 *(pin 11-15)* — 4 bits identify code for selection of receiver in communications

TxD *(pin 4)* — Transmitted data input (UART-RxD)

RxD *(pin 5)* — Received data output (UART-TxD)

RST *(pin 3)* — Master reset of micro-controller (build-in)

M0 *(pin 6)* — Bit rate selector, logic "1" for 9600 bit/s and "0" for 4800 bit/s. This pin used for package transfer mode only.

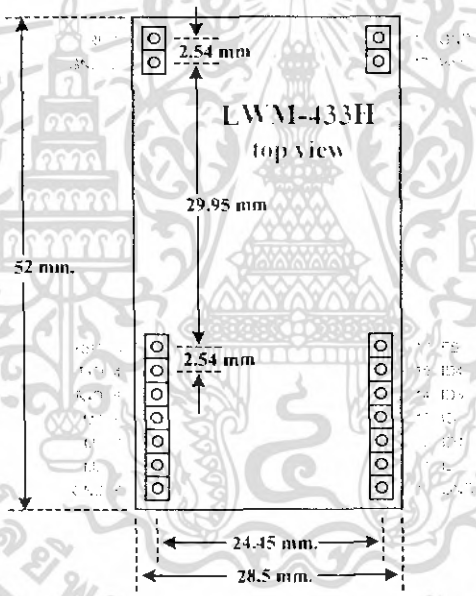
M1 *(pin 7)* — Mode selector, logic "1" for packet transfer mode and logic "0" for real-time transfer mode *(Real-time mode not available now)*

M2 *(pin 8)* — Request to send (RTS) used to transmit data in real-time mode, active high at this pin to turn on transmitter and active low to turn on receiver *(Not available now)*

pHz Communications Co., Ltd.
9/204 Moo 1 Buatongkeha Bangbuatong Nonthaburi 11110 THAILAND
Tel: +66-0-2925-8563, +66-0-2925-8565 Fax: 66-0-2925-8564 E-mail: phz@p-Hz.com

Pin description: (continue)

FS *(pin 16)*	Frequency selector, logic "0" for 433.93 MHz and logic "1" for 434.33 MHz frequency operation
RF in/out *(pin 1)*	50Ω RF input from antenna connected to this pin and DC isolated internally. (50Ω antenna are recommended for high efficiency data transmissions)
GND *(pin 2,9,10,18)*	Ground pin (these pin should be connected to RF return path)



Physical dimension

pHz Communications Co., Ltd.
9/204 Moo 1 Buatongkeha Bangbuatong Nonthaburi 11110 THAILAND
Tel: +66-0-2925-8563, +66-0-2925-8565 Fax: 66-0-2925-8564 E-mail: phz@p-Hz.com

Absolute Maximum Rating:

Operating temperature: 0 °C to +55°C
Storage temperature: -30 °C to +85°C
Operating supply voltage: 6 VDC
Antenna (pin 5): ± 50 VDC

Electrical Specifications:

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Vcc	Main supply voltage	Vcc at pin 17	4.5	5.0	5.5	V
Icc	Main supply current	Icc at pin 17 @ Vcc = 5V		25		mA
ID4-ID0	Identify code	Input logic "1" at pin 15-11	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 15-11	-0.1	0	0.8	V
RST	Master reset (Active high)	Normal operation @ pin 3	-0.1	0	1.2	V
		Reset operation @ pin 3	Vcc-0.1	Vcc	Vcc+0.6	V
RxD	UART received Data (Normal High Level)	Input logic "1" at pin 4	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 4	-0.1	0	0.8	V
TxD	UART transmitted Data (Normal High Level)	Input logic "1" at pin 5	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 5	-0.1	0	0.8	V
M0	Bit rate selector ("0" for 4800 bit/s) ("1" for 9600 bit/s)	Input logic "1" at pin 6	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 6	-0.1	0	0.8	V
M1	Mode selector ("0" for real-time mode) ("0" for package mode)	Input logic "1" at pin 7	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 7	-0.1	0	0.8	V
Dmax	Maximum Data	in package mode		4000		byte
M2	Request to send (in real-time mode) High = Tx, Low = Rx	Input logic "1" at pin 8	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 8	-0.1	0	0.8	V

pHz Communications Co., Ltd.
9/204 Moo 1 Buatongkeha Bangbuatong Nonthaburi 11110 THAILAND
Tel: +66-0-2925-8563, +66-0-2925-8565 Fax: 66-0-2925-8564 E-mail: phz@p-Hz.com

Electrical Specifications: (continue)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
FS	Frequency selector ("0" for 433.93 MHz) ("1" for 434.33 MHz)	Input logic "1" at pin 16	Vcc-0.1	Vcc	Vcc+0.6	V
		Input logic "0" at pin 16	-0.1	0	0.8	V
Fo	Output frequency ("0" for 19200 bit/s) ("1" for 9600 bit/s)	433.93 MHz operation	433.88	433.93	433.98	MHz
		434.33 MHz operation	434.28	434.33	434.38	MHz
Fer	Frequency Error	433.93 and 434.33 MHz	50	70	100	ppm
Po	Output power	Output power @ pin 1	+4	+5	+6	mW
2nd	Second harmonic	25° C		-30	-20	dBc

Note: Case temperature = 25°C, Test source impedance = 50 ohms.

pHz Communications Co., Ltd.
9/204 Moo 1 Buatongkeha Bangbuatong Nonthaburi 11110 THAILAND
Tel: +66-0-2925-8563, +66-0-2925-5865 Fax: 66-0-2925-8564 E-mail: phz@p-Hz.com