เครื่องบันทึกเวลาโดยใช้ RFID

TIME RECORDER USING RFID

โดย

นายมารุตต์ พรหมจันทร์แดง

นายสันติพงษ์ บุญผลิตกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

เครื่องบันทึกเวลาโดยใช้ RFID

TIME RECORDER USING RFID

โดย

นายมารุตต์ พรหมจันทร์แดง 45010613

นายสันติพงษ์ บุญผลิตกุล 45010816

อาจารย์ที่ปรึกษา

รศ. จิรวัฒน์ ปานกลาง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ ปีการศึกษา 2548

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องบันทึกเวลาโดยใช้ RFID (TIME RECORDER USING RFID)

ผู้จัดทำ

1.นายมารุตต์ พรหมจันทร์แดง รหัส 45010613

2.นายสันติพงษ์ บุญผลิตกุล รหัส 45010816

..... อาจารย์ที่ปรึกษา

(รศ.จิรวัฒน์ ปานกลาง)

เครื่องบันทึกเวลาโดยใช้ RFID

นายมารุตต์ พรหมจันทร์แดง 45010613
นายสันติพงษ์ บุญผลิตกุล 45010816
รศ.จิรวัฒน์ ปานกลาง (อาจารย์ที่ปรึกษา)
ปีการศึกษา 2548

บทคัดย่อ

โครงงานนี้เป็นการประยุกต์ใช้งาน RFID ซึ่งในปัจจุบันเริ่มเป็นที่รู้จักกันอย่างแพร่หลายและนิยมนำมาใช้งานทางด้านการรักษาความปลอดภัย หรือการใช้งานเป็นฐานข้อมูล สำหรับโครงงานนี้จะใช้งาน RFID แบบอ่านได้เพียงอย่างเดียว ความถี่ 125 kHz ซึ่งใช้กับบัตรแบบอ่านเพียงอย่างเดียว ไม่สามารถเปลี่ยนแปลงค่าของข้อมูลหรือรหัสผู้ใช้งานลงในบัตรได้ ในบัตรแต่ละใบจะมีข้อมูลรหัสไม่ซ้ำกัน ดังนั้นจึงสามารถนำมาใช้งานได้โดยการเก็บค่าข้อมูลรหัสของบัตรนั้นไว้ในหน่วยความจำก่อน แล้วจึงนำมาใช้งานโดยการเปรียบเทียบข้อมูลจากรหัสของบัตรด้วยไมโครคอนโทรลเลอร์ ตระกูล MCS-51 ใช้งานในการควบคุม การปิด-เปิด อุปกรณ์ที่ต่อใช้งานภายนอกอื่นๆ ในการใช้งานด้านฐานข้อมูล จะใช้บันทึกข้อมูลเวลาจาก real time clock ลงในหน่วยความจำ เพื่อบันทึกเวลาที่ใช้งานขณะนั้น หรือใช้บันทึกเวลา เข้า-ออก สำหรับการลงเวลาทำงาน เพื่อความสะดวก รวดเร็ว แม่นยำ และมีความน่าเชื่อถือ โดยการใช้งาน EEPROM เป็นฐานข้อมูลรหัสบัตรและหน่วยความจำบันทึกเวลาซึ่งสามารถบันทึกรหัสบัตรได้ 1,635 รหัส และบันทึกหน่วยความจำเวลาได้ 5,460 ชุดข้อมูล การใช้งานผู้ใช้งานสามารถรับรู้สถานะได้จากการแสดงผลบนจอ LCD และเสียงสัญญาณเตือน อีกทั้งยังสามารถนำมาใช้งานร่วมกับคอมพิวเตอร์เพื่อความสะดวกในการใช้งานฐานข้อมูล ด้วยการใช้งานทางด้านซอฟท์แวร์ ทำให้ผู้ใช้งานสามารถใช้งานได้โดยไม่ยุ่งยากนัก

TIME RECORDER USING RFID

Mr.Maroot Promchandang 45010613
Mr.Santipong Boonphalitkul 45010816
Assoc.Prof. Jirawat Panklang (Advisor)
Education Year 2005

Abstract

This project present an application of RFID. In present, it is known in wide-spread and mostly use in security or database. For this project, RFID module is a reader, frequency at 125 kHz, is used with card that data or ID can not be changed. ID in each card is unique thus we can compare ID data from each card with database to let MCS-51 microcontroller control other devices on-off. For database use, it is used to record time data from real time clock to the memory in order to collect time and in-out status thus it is facility, fast, accurate and reliable. We use EEPROM to record card ID database and time memory which database can be recorded 1,635 IDs and memory can be recorded 5,460 data. User can get the status from LCD display and sound alert from buzzer. This project can be used with personal computer for facility in database by using software computer thus it can be used easier.

กิตติกรรมประกาศ

โครงงานเครื่องบันทึกเวลาโดยใช้ RFID นี้ เกิดขึ้นและสำเร็จได้ แม้ว่าจะมีอุปสรรคหลาย ๆ ด้านเข้ามาขัดขวาง รบกวนการทำงานก็ตาม ก็เพราะด้วยความกรุณาและการชี้แนะจากอาจารย์ที่ปรึกษา รศ.จิรวัฒน์ ปานกลาง ที่ชี้นำแนวทาง ชี้แนะการแก้ไขปัญหาต่าง ๆ ให้คำแนะนำและให้คำปรึกษาที่ดีเสมอมา จึงทำให้โครงงานนี้สำเร็จออกมาเป็นที่น่าพอใจอย่างมาก รวมทั้งบรรดาอาจารย์ท่านอื่นที่ได้เคยสอนสั่งวิชาการด้านต่าง ๆ ให้ได้มีความรู้ติดตัวมาจนสามารถนำมาประยุกต์ใช้ในการคิดวิเคราะห์ แก้ปัญหาต่างๆ ได้ ด้วยความกรุณานี้ลูกศิษย์จะระลึกพระคุณไว้เสมอ ขอขอบพระคุณอาจารย์อย่างสูง และขอบคุณบรรดาเพื่อน ๆ ทั้งหลายที่คอยช่วยเหลือ ให้คำปรึกษา เอื้อเฟื้อเผื่อแผ่และให้กำลังใจตลอดมา อันเป็นแรงผลักดันให้สามารถทำงานได้อย่างไม่ย่อท้อ โดยเฉพาะเพื่อนร่วมอาจารย์โปรเจ็คเดียวกันที่ช่วยเหลือซึ่งกันและกันเสมอมา ให้ความสนุกสนานและกำลังใจทำให้มีกำลังกายต่อสู้กับปัญหาต่าง ๆ นานาได้อย่างไม่ลดละ ขอบใจมากเพื่อน อีกทั้งคุณพ่อ คุณแม่และญาติพี่น้องทั้งหลาย ที่คอยเป็นกำลังใจและแรงกดดันให้พยายามตั้งใจศึกษาเล่าเรียน พากเพียรจนสำเร็จได้ในวันนี้ ขอขอบคุณพระคุณพ่อและคุณแม่อย่างสูงที่ได้อบรมสั่งสอนลูกคนนี้ให้เป็นคนดี คนขยัน มีมานะอดทน และคอยว่ากล่าวตักเตือนเรื่อยมาอันถือเป็นก้าวแรกของความสำเร็จในวันนี้ ขอขอบพระคุณทุก ๆ ท่านด้วยใจจริง ขอบคุณครับ

มารุตต์ พรหมจันทร์แดง

(นายมารุตต์ พรหมจันทร์แดง)

สันติพงษ์ บุญผลิตกุล

(นายสันติพงษ์ บุญผลิตกุล)

ผู้จัดทำ

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันดูเหมือนว่าเวลาทุกนาทีชั่งมีค่ามาก ในการทำงานเวลาก็เป็นเรื่องที่สำคัญมาก เพราะมันสามารถบอกถึงความรับผิดชอบของบุคคลใดบุคคลหนึ่งได้ โดยเฉพาะในเวลาเข้างานตอนเช้าจะต้องเข้างานให้ทันเวลา ซึ่งคนส่วนใหญ่ต้องเสียเวลาในการเดินทางมากเนื่องจากที่พักอยู่ไกลกับที่ทำงาน และยังต้องเสียเวลาในการเข้างานอีกเพราะในการเข้างานต้องมีการบันทึกเวลาในการเข้าทำงาน บันทึกชื่อ ดังนั้นถ้าบริษัทหรือโรงงานใดมีพนักงานเป็นจำนวนมากจะทำให้ต้องเสียเวลาในการบันทึกข้อมูลของพนักงานมาก และอาจจะทำให้เกิดปัญหาที่พนักงานมาทันเวลาเข้าทำงานแต่เสียเวลาในการเข้าแถวรอการบันทึกเวลา บันทึกชื่อในการเข้าทำงาน ทำให้พนักงานต้องเข้าทำงานสาย

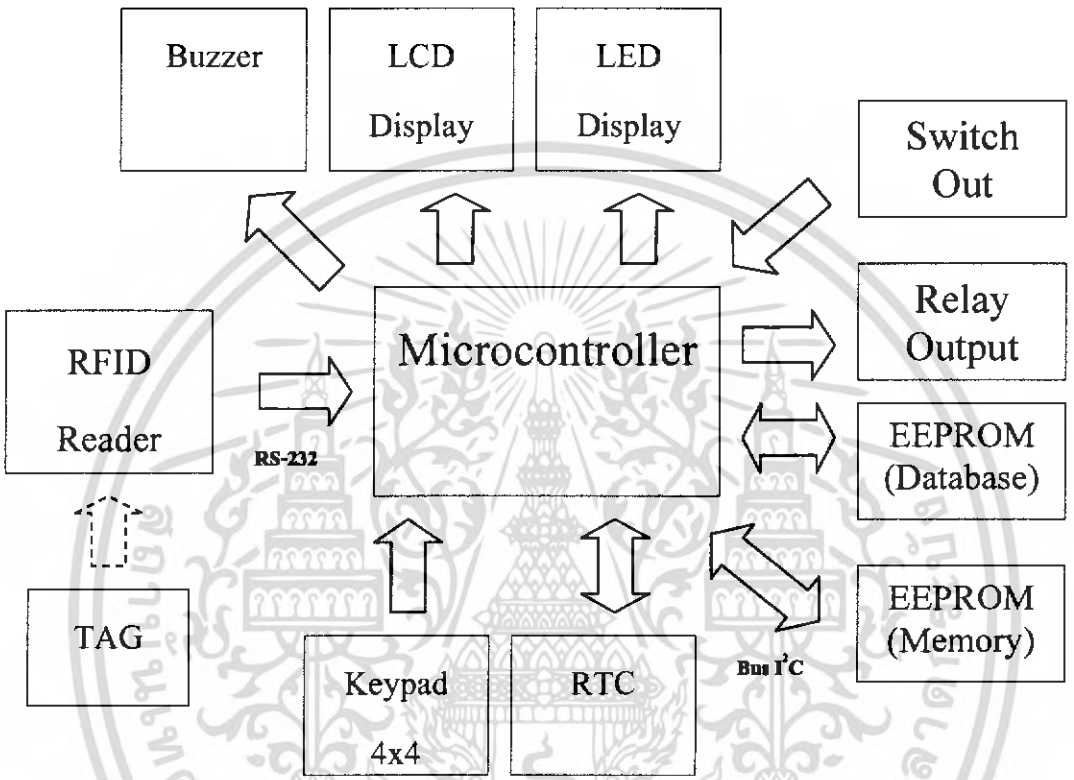
โครงงานเครื่องบันทึกเวลาโดยใช้ RFID นี้มีจุดประสงค์หลักเพื่อใช้ในการบันทึกเวลาโดยที่จะใช้เครื่องอ่านสัญญาณ (RFID Reader) รับข้อมูลจากแผ่นป้ายระบุข้อมูล (RFID Tags) จากนั้นส่งข้อมูลที่ได้ไปยังไมโครคอนโทรเลอร์ เพื่อทำการประมวลผลซึ่งจะมีการแสดงผลผ่านจอ LCD และ LED จากนั้นนำข้อมูลที่ได้เก็บไว้ในหน่วยความจำภายนอก และนำข้อมูลไปใช้งานร่วมกับคอมพิวเตอร์ต่อไป

โครงงานนี้เป็นการประยุกต์ใช้งาน RFID ซึ่งสะดวกกว่าการใช้งานแบบการใช้บาร์โค้ด RFID มีจุดเด่นอยู่ที่การอ่านข้อมูลจากบัตรหรือแท็ก (TAG) ได้หลายๆ แท็กแบบไร้สัมผัสและสามารถอ่านค่าได้แม้ในสภาพที่ทัศนวิสัยไม่ดี ทนต่อความเปียกชื้น แรงสั่นสะเทือน การกระทบกระแทก สามารถอ่านข้อมูลได้ด้วยความเร็วสูง

RFID ประกอบด้วยแผ่นป้ายระบุข้อมูล (RFID Tags) และเครื่องอ่านสัญญาณ(RFID Reader) แผ่นป้ายระบุข้อมูล (RFID Tags) ประกอบด้วยแผงวงจรไมโครชิปกับเสาอากาศขนาดจิ๋วที่ฝังเป็นส่วนหนึ่งของแผ่นป้ายระบุข้อมูล ระบบการลงเวลาจะใช้แผ่นป้ายระบุข้อมูลเป็นแบบ Passive โดยใช้ความถี่ที่ 125 MHz

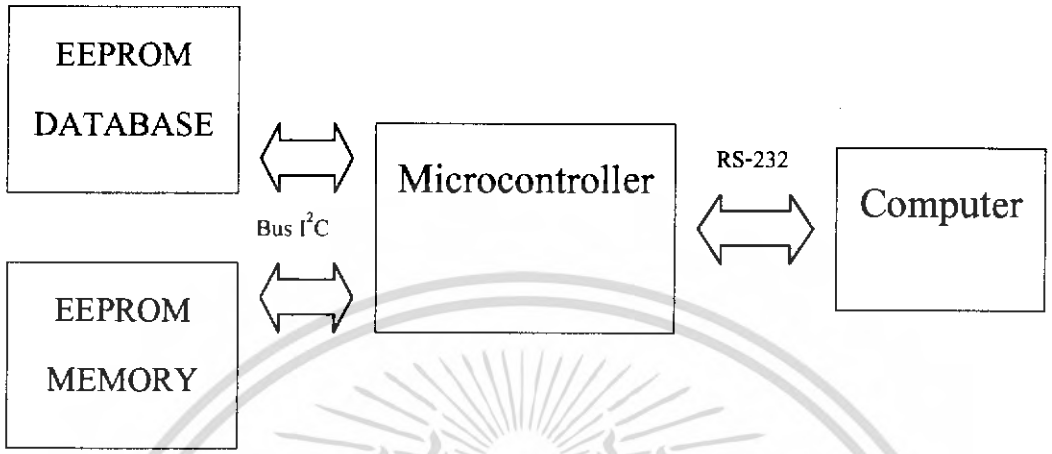
เครื่องอ่านสัญญาณ (RFID Reader) ทำหน้าที่สร้างความถี่สัญญาณวิทยุ ซึ่งความถี่ที่สร้างขึ้นจะมีขนาดเท่ากับที่แผ่นป้ายระบุข้อมูล (RFID Tags) สามารถตอบสนองได้ (125 MHz) โดยอาศัยทฤษฎีการเหนี่ยวนำสัญญาณไฟฟ้า เมื่อคลื่นสัญญาณกระทบกับแผ่นป้ายระบุข้อมูล (RFID Tags) เพื่อให้แผ่นป้ายระบุข้อมูล (RFID Tags) ส่งข้อมูลของตัวเองกลับมายังเครื่องอ่านสัญญาณ (RFID Reader) จากนั้นจะแปลงสัญญาณที่ได้รับให้อยู่ในรูปดิจิตอลเพื่อใช้ประมวลผลต่อไป

โครงงานนี้แบ่งออกเป็น 2 ส่วน คือ ส่วนที่ติดต่อกลับผู้ใช้งานโดยตรงและส่วนที่ใช้ในการแลกเปลี่ยนข้อมูลกลับคอมพิวเตอร์



รูปที่ 1.1 ไดอะแกรมโครงสร้างของเครื่องบันทึกเวลาโดยใช้ RFID

ในส่วนที่ติดต่อกลับผู้ใช้งานโดยตรงนั้นจะใช้เครื่องบันทึกเวลาโดยใช้ RFID ซึ่งจะทำการติดตั้งไว้หน้าประตูหรือสิ่งกีดขวาง การใช้งานเมื่อผู้ใช้งานแตะบัตร (TAG) ไมโครคอนโทรเลอร์จะทำการตรวจสอบข้อมูลจากบัตรกับข้อมูลที่ทำการบันทึกไว้ในฐานข้อมูล EEPROM DATABASE ว่าตรงกันหรือไม่ ถ้าไม่ตรงกันประตูก็จะไม่เปิด แต่ถ้าข้อมูลตรงกันประตูก็จะเปิดออก สำหรับการบันทึกเวลาจะทำการบันทึกเมื่อทำการกดปุ่มบันทึกการลงเวลาเข้าหรือออก ซึ่งจะบันทึกอยู่ในหน่วยความจำ EEPROM MEMORY



รูปที่ 1.2 ไดอะแกรมการเชื่อมต่อกับคอมพิวเตอร์

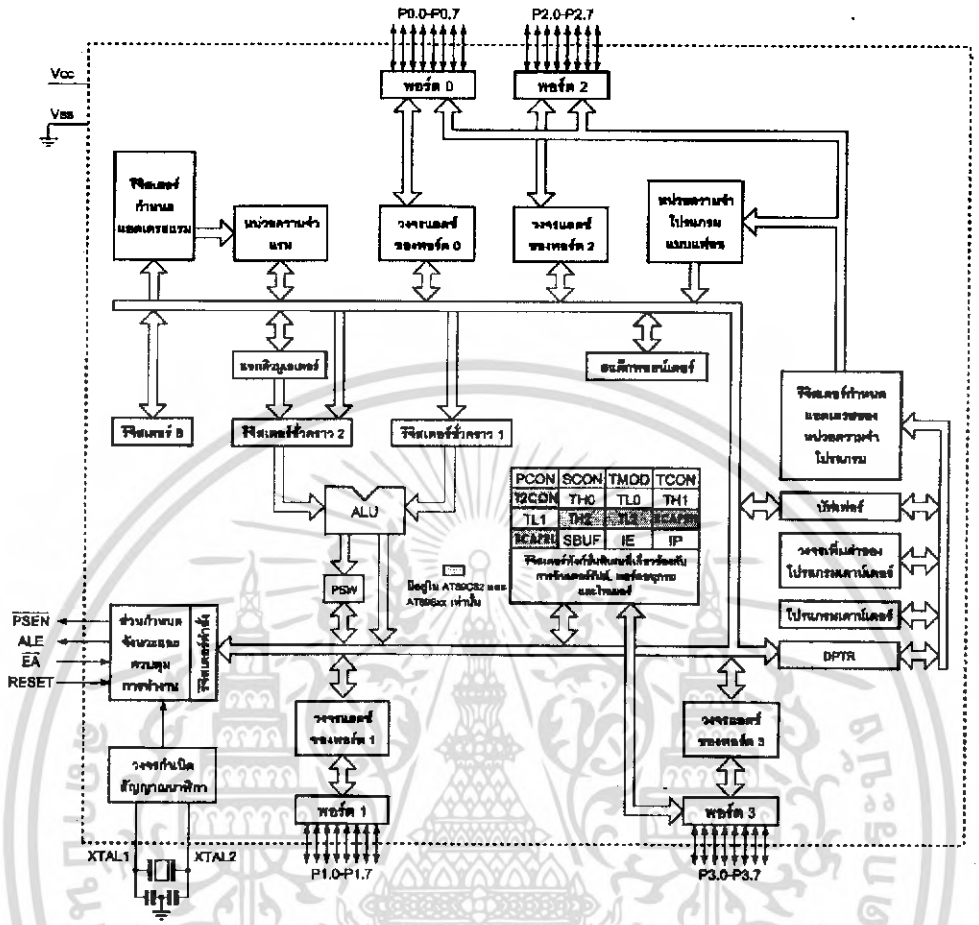
ส่วนที่ใช้ในการแลกเปลี่ยนข้อมูลกลับคอมพิวเตอร์ การใช้งานคือนำหน่วยความจำ (EEPROM) จากเครื่องบันทึกเวลาโดยใช้ RFID เชื่อมต่อกับ EEPROM DATABASE/MEMORY READER ทำการเชื่อมต่อกับคอมพิวเตอร์เพื่อใช้แลกเปลี่ยนข้อมูลและการเก็บข้อมูล

บทที่ 2
ไมโครคอนโทรลเลอร์ MCS-51

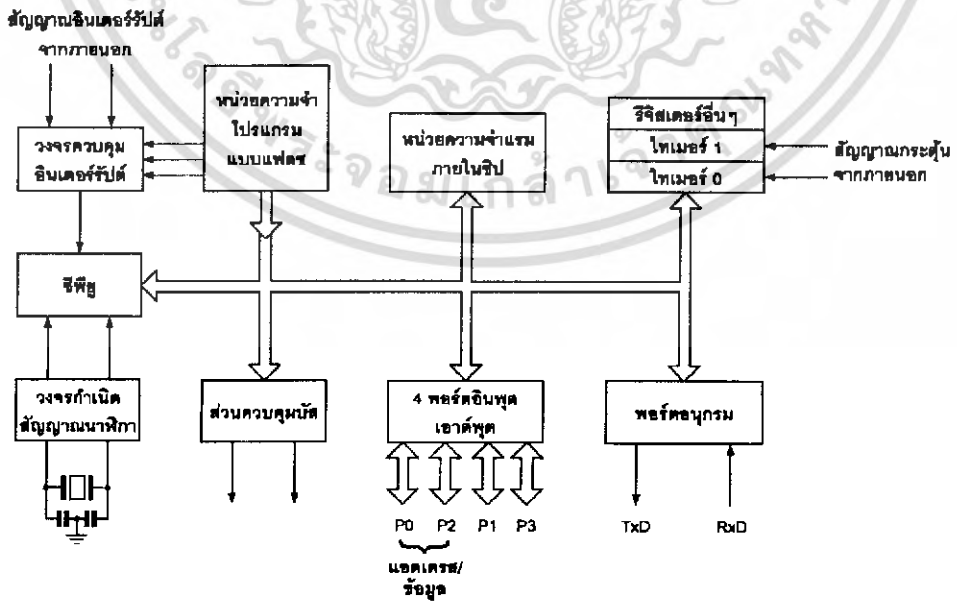
2.1 โครงสร้างและสถาปัตยกรรมของไมโครคอนโทรลเลอร์ MCS-51

2.1.1 คุณสมบัติของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 อนุกรม AT89xx

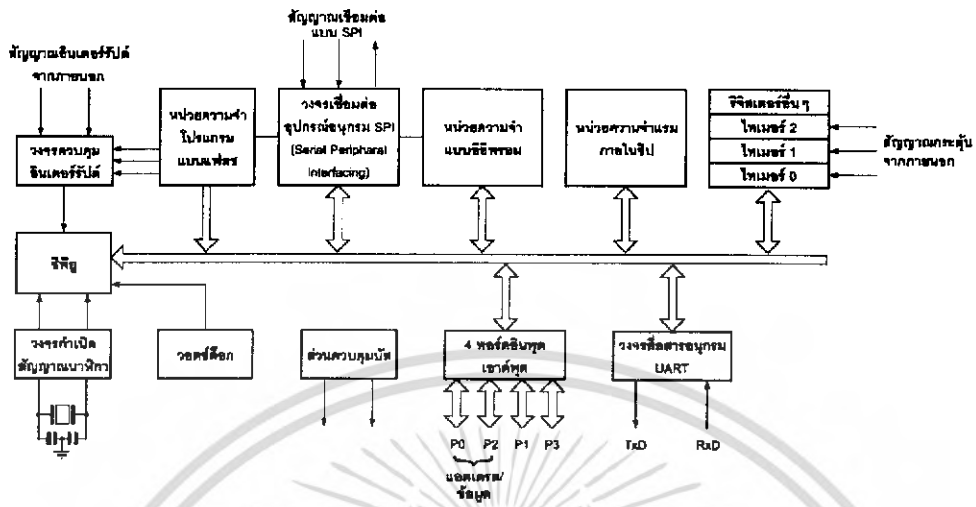
- เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียูขนาด 8 บิต
- ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนใหม่ได้พันครั้ง
- หน่วยความจำข้อมูลพื้นฐานเป็นหน่วยความจำแบบแรม
- ขาพอร์ตเป็นแบบสองทิศทาง สามารถใช้งานเป็นได้ทั้งอินพุตและเอาท์พุต
- มีวงจรสื่อสารอนุกรมแบบฟูลดูเพล็กซ์
- ไทเมอร์/เคาน์เตอร์ขนาด 16 บิตอย่างน้อย 2 ตัว
- สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ได้ 6 ประเภท
- สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้สูงสุด 64 กิโลไบต์
- มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป
- มีวงจรสื่อสารอนุกรมแบบ SPI สำหรับในอนุกรม AT89Sxx
- มีวอตช์ด็อกไทเมอร์ในตัว สำหรับในอนุกรม AT89Sxx



รูปที่ 2.1 รายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์ MCS-51



รูปที่ 2.2 โครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชในอนุกรม AT89Cxx



รูปที่ 2.3 โครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชในอนุกรม AT89Sxx

2.1.2 การจัดขาไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 ทุกเบอร์มีสถาปัตยกรรมและขาใช้งานพื้นฐานเหมือนกัน

รูปที่ 2.4 การจัดขามาตรฐานของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89C5x

ขา VCC ใช้สำหรับต่อไฟเลี้ยง +5 V

ขา GND เป็นขากราวด์ ต่อลงกราวด์ของระบบ

ขาพอร์ต 0 (P0.0-P0.7) มี 8 ขาแต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ต 0 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้ โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสภาพปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้น ขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้กระบวนการมัลติเพล็กซ์เข้าช่วย เพื่อสลับการทำงานเป็นได้ทั้งขาติดต่อแอดเดรสและขาข้อมูล

ขาพอร์ต 1 (P1.0-P1.7) มี 8 ขาแต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล "1"ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย นอกจากนั้นในอนุกรม AT89Sxx จะใช้ขา P1.0 เป็นขาอินพุตสำหรับนับค่าของไทเมอร์ 2 และ P1.1 เป็นขาอินพุตทริกเกอร์ของไทเมอร์ 2 ในขณะที่ขา P1.4 –P1.7 เป็นขาสำหรับเชื่อมต่อแบบ SPI เพื่อทำการโปรแกรมข้อมูล

ขาพอร์ต 2 (P2.0-P2.7) มี 8 ขาแต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ต 2 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้ โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสภาพปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้น ขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก

ขาพอร์ต 3 (P3.0-P3.7) มี 8 ขาแต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ต 2 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้ โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสภาพปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้น ขาพอร์ต 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษ ดังมีรายละเอียดขั้นตอนต่อไปนี้

P3.0 ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

P3.1 ใช้เป็นขาอินพุตสำหรับส่งข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

P3.2 ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 0 หรือขา INT0

P3.3 ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขา INT1

P3.4 ใช้เป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0 หรือ T0

P3.5 ใช้เป็นขาอินพุตสำหรับรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขา T1

P3.6 ใช้เป็นขาสัญญาณ WR ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

P3.7 ใช้เป็นขาสัญญาณ RDในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

ขารีเซต (Reset) ใช้ในการรีเซตการทำงานของไมโครคอนโทรลเลอร์ โดยการป้อนสัญญาณเพื่อรีเซตสถานะที่ขานี้ต้องอยู่ในระดับรีเซตอย่างน้อย 2 แมชชีนไซเกิล โดยที่จงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างปกติ

ขา ALE/PROG (Address Latch Enable/Program Pulse input) เป็นขาที่ใช้ในการควบคุมการแลตช์ของขาพอร์ต 0 เมื่อมีการใช้งานหน่วยความจำภายนอก นอกจากนั้นขานี้ยังใช้เป็นขาสำหรับรับพัลส์ของการโปรแกรมสำหรับโปรแกรมข้อมูลลงในไมโครคอนโทรลเลอร์ MCS-51 ในรุ่นที่มีหน่วยความจำโปรแกรมเป็นแบบอีพรอม

ขา PSEN (Program Store Enable) ขานี้ใช้ในการส่งสัญญาณเพื่อร้องขอติดต่อกับหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอก ตัวไมโครคอนโทรลเลอร์ออกมาที่ขานี้ 2 ครั้งในแต่ละแมชีนไซเกิล แต่ถ้าหากติดต่อกับหน่วยความจำข้อมูลภายนอก ขานี้จะไม่มีการส่งสัญญาณใดๆออกมา

ขา EA/Vpp (External Access Enable/Programming voltage input) ใช้สำหรับเลือกติดต่อหน่วยความจำโปรแกรมภายนอกหรือภายในตัวไมโครคอนโทรลเลอร์ ถ้าหากขานี้เป็น "0" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมภายนอก แต่ถ้าหากขานี้เป็น "1" เลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายในไมโครคอนโทรลเลอร์ นอกจากนี้ที่ขานี้ยังเป็นขาอินพุตสำหรับรับแรงดันไฟสูงสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์ สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบเฟลชต้องการแรงดันสำหรับการโปรแกรม +12 V

ขา XTAL1 และ **XTAL2** เป็นขาสำหรับติดต่อคริสตอลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์

เบอร์ของ ไมโครคอนโทรลเลอร์	หน่วยความจำโปรแกรม	หน่วยความจำข้อมูล	จำนวนไทเมอร์/ เคาน์เตอร์ 16 บิต
AT89C1051	แบบแฟลชขนาด 1 กิโลไบต์	แรม 64 ไบต์	1
AT89C2051	แบบแฟลชขนาด 2 กิโลไบต์	แรม 128 ไบต์	2
AT89C51	แบบแฟลชขนาด 4 กิโลไบต์	แรม 128 ไบต์	2
AT89C52	แบบแฟลชขนาด 8 กิโลไบต์	แรม 256 ไบต์	3
AT89C55	แบบแฟลชขนาด 20 กิโลไบต์	แรม 256 ไบต์	3
AT89S8252	แบบแฟลชขนาด 8 กิโลไบต์	แรม 256 ไบต์ EEPROM 2 กิโลไบต์	3
AT89S53	แบบแฟลชขนาด 12 กิโลไบต์	แรม 256 ไบต์	3

ตารางที่ 2.1 รายละเอียดโดยสรุปบางส่วนของไมโครคอนโทรลเลอร์

2.2 ไทเมอร์/เคาน์เตอร์ของไมโครคอนโทรลเลอร์ MCS-51

2.2.1 รีจิสเตอร์ควบคุมการทำงานของไทเมอร์/เคาน์เตอร์หรือ TCON (Timer/Counter Control Register)

เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 88H ในพื้นที่ของรีจิสเตอร์ SFR สามารถเข้าถึงได้ในระดับบิตมีรายละเอียดการทำงานดังนี้

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

TF1 (Timer 1 overflow flag) : เซตด้วยกระบวนการทางฮาร์ดแวร์ เมื่อค่าของรีจิสเตอร์ Timer 1 เกิดการนับเกินหรือเกิดโอเวอร์โฟลว การเคลียร์บิตนี้ทำได้ด้วยกระบวนการทางฮาร์ดแวร์เช่นกัน โดยบิตนี้ จะเคลียร์เมื่อมีการอินเตอร์รัปต์เกิดขึ้น

TR1 (Timer 1 run control bit) : ใช้ในการเปิดปิดการทำงานของไทเมอร์ 1 (เอ็นเอเบิลหรือดิสเอเบิล) ทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์ ถ้าต้องการให้ไทเมอร์ 1 ทำงานต้องเซตบิตนี้ให้เป็น "1"

TF0 (Timer 0 overflow flag) : เซตด้วยกระบวนการทางฮาร์ดแวร์เมื่อค่าของรีจิสเตอร์ Timer 0 เกิดการนับเกินหรือเกิดโอเวอร์โฟลว การเคลียร์บิตนี้ทำได้ด้วยกระบวนการทางฮาร์ดแวร์เช่นกัน โดยบิตนี้จะเคลียร์เมื่อมีการอินเตอร์รัปต์เกิดขึ้น

TR0 (Timer 0 run control bit) : ใช้ในการเปิดปิดการทำงานของไทเมอร์ 0 (เอ็นเอเบิลหรือดิสเอเบิล) ทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์ ถ้าต้องการให้ไทเมอร์ 0 ทำงานต้องเซตบิตนี้ให้เป็น "1"

IE1 (External Interrupt 1 edge flag) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์ สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อสามารถตรวจจับขอบขาของสัญญาณอินเตอร์รัปต์จากภายนอกที่ขาอินพุตอินเตอร์รัปต์ 1 (INT1) ได้และจะทำการเคลียร์เมื่อมีการบริการอินเตอร์รัปต์เกิดขึ้น

IT1 (Interrupt 1 type control bit) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์โดยใช้ในการเลือกลักษณะของสัญญาณอินเตอร์รัปต์จากภายนอกที่ต้องการให้ทำการตอบสนอง สำหรับขาอินพุตอินเตอร์รัปต์ 1 (INT1) การเซตและเคลียร์ทำได้ด้วยกระบวนการซอฟต์แวร์

"0" เลือกขอบขาลงของสัญญาณ (falling edge)

"1" เลือกระดับลอจิกต่ำ (low level triggered)

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

IT0 (Interrupt 0 type control bit) : บิตนี้จะใช้ในกระบวนการอินเตอร์รัปต์โดยใช้ในการเลือกลักษณะของสัญญาณอินเตอร์รัปต์จากภายนอกที่ต้องการให้ทำการตอบสนอง สำหรับขาอินพุตอินเตอร์รัปต์ 0 (INT0) การเซตและเคลียร์ทำได้ด้วยกระบวนการซอฟต์แวร์

"0" เลือกขอบขาลงของสัญญาณ (falling edge)

"1" เลือกระดับลอจิกต่ำ (low level triggered)

2.2.2 รีจิสเตอร์เลือกโหมดการทำงานของไทเมอร์/เคาน์เตอร์หรือ TMOD (Timer/CounterMode Control Register)

เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 89H ในพื้นที่ของรีจิสเตอร์ SFR ไม่สามารถเข้าถึงได้ในระดับบิต แบ่งการทำงานออกเป็น 2 ส่วน คือ 4 บิต ล่างใช้ในการเลือกโหมดการทำงาน

ของไทเมอร์ 0 และ 4 บิต บนใช้ในการเลือกโหมดการทำงานของไทเมอร์ 1 ดังนั้นในการอธิบายการทำงานจะขออธิบายเพียงส่วนเดียวดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
GATE	**C/T**	**M1**	**M0**	**GATE**	**C/T**	**M1**	**M0**
ไทเมอร์ 1				**ไทเมอร์ 0**			

Gate : ใช้เลือกลักษณะการควบคุมการทำงานของไทเมอร์/เคาน์เตอร์

"0" ไทเมอร์/เคาน์เตอร์ จะทำงานเมื่อบิต TRxในรีจิสเตอร์TCONเป็น "1" เรียกการควบคุมแบบนี้ว่าการควบคุมทางซอฟต์แวร์

"1" ไทเมอร์/เคาน์เตอร์ จะทำงานเมื่อบิต TRxในรีจิสเตอร์TCONเป็น "1" และสถานะลอจิกที่ขาอินพุตอินเตอร์รัปต์ INT0 และ INT1 เป็น "1" เรียกการควบคุมแบบนี้ว่าการควบคุมทางฮาร์ดแวร์

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

"0" เลือกให้ทำงานเป็นไทเมอร์ โดยที่จะใช้สัญญาณอินพุตจากสัญญาณนาฬิกาภายในของไมโครคอนโทรลเลอร์

"1" เลือกให้ทำงานเป็นเคาน์เตอร์ โดยรับสัญญาณอินพุตทางขา T0 หรือ T1

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

"00" เลือกให้ทำงานในโหมดไทเมอร์/เคาน์เตอร์ 13 บิต

"01" เลือกให้ทำงานในโหมดไทเมอร์/เคาน์เตอร์ 16 บิต

"10" เลือกให้ทำงานในโหมดไทเมอร์/เคาน์เตอร์ ขนาด 8 บิตแบบตั้งค่าอัตโนมัติ

"11" สำหรับไทเมอร์ 0 เลือกให้ทำงานในโหมดไทเมอร์/เคาน์เตอร์แยกส่วน โดยแยกออกเป็นไทเมอร์/เคาน์เตอร์ 8 บิต 2 ตัว รีจิสเตอร์ TL0 จะได้รับการควบคุมการเปิดปิดจากบิต TR0 ในรีจิสเตอร์ TCON และรีจิสเตอร์ TH0 ซึ่งเป็นไทเมอร์/เคาน์เตอร์ 8 บิต อีกตัวหนึ่ง จะได้รับการควบคุมจากบิต TR1 ในรีจิสเตอร์ TCON ในกรณีของไทเมอร์ 1 เป็นการสั่งให้ไทเมอร์/เคาน์เตอร์ 1 หยุดทำงาน (ดิสเอเบิล)

2.3 กระบวนการอินเตอร์รัปต์ของไมโครคอนโทรลเลอร์ MCS-51

2.3.1 รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัปต์หรือ IE (Interrupt Enable register)

มีแอดเดรสอยู่ที่ A8H ในพื้นที่ของรีจิสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิต สามารถเข้าถึงได้ในระดับบิต ใช้ในการเอ็นเอเบิลการตอบสนองการอินเตอร์รัปต์ในแบบต่าง ๆ มีรายละเอียดการทำงานดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
EA	**-**	**ET2**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**

EA(Global enable/disable interrupt) : ใช้เอ็นเอเบิลและดิสเอเบิลการตอบสนองการอินเตอร์รัปต์ทั้งหมด

"0" ดิสเอเบิลการอินเตอร์รัปต์ นั่นคือ กำหนดให้ไมโครคอนโทรลเลอร์ไม่ตอบสนองการอินเตอร์รัปต์

"1" เอ็นเอเบิลการอินเตอร์รัปต์ นั่นคือ กำหนดให้ไมโครคอนโทรลเลอร์สามารถตอบสนองการอินเตอร์รัปต์จากแหล่งกำเนิดต่างๆ

ถ้าต้องการให้ไมโครคอนโทรลเลอร์ตอบสนองการอินเตอร์รัปต์ ไม่ว่าจะแหล่งกำเนิดใดจะต้องเซตบิตนี้ก่อนเสมอ สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

ET2 (Timer 2 interrupt enable) : ใช้ในการเอ็นเอเบิลการอินเตอร์รัปต์อันเนื่องมาจากการโอเวอร์โฟลวหรือการแคปเจอร์ในไทเมอร์/เคาน์เตอร์ 2 จะมีเฉพาะในเบอร์ AT89C52 และในอนุกรม AT89Sxx เท่านั้น บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

SM2 : ใช้ในการเอ็นเอเบิลการสื่อสารในแบบมัลติโปรเซสเซอร์

ES (Serial port interrupt enable bit) : ใช้ในการเอ็นเอเบิลการอินเตอร์รัปต์ อันเนื่องมากจากการรับหรือส่งข้อมูลทางพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51 สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

ET1 (Timer 1 interrupt enable) : ใช้ในการเอ็นเอเบิลการอินเตอร์รัปต์ อันเนื่องมาจากการโอเวอร์โฟลวในไทเมอร์/เคาน์เตอร์ 1 บิตนี้ สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

EX1 (External interrupt 1 enable bit) : ใช้ในการเอ็นเอเบิลการอินเตอร์รัปต์ อันเนื่องมาจากสัญญาณภายนอกที่ป้อนเข้ามายังขา INT1 บิตนี้สามารถเซตและเคลียร์ ด้วยกระบวนการทางซอฟต์แวร์

ET0 (Timer 0 interrupt enable) : ใช้ในการเอ็นเอเบิลการอินเตอร์รัปต์อันเนื่องมาจากการโอเวอร์โฟลวในไทเมอร์/เคาน์เตอร์ 0 บิตนี้ สามารถเซต และเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

EX0 (External interrupt 1 enable bit) : ใช้ในการเอ็นเอเบิลการอินเตอร์รัปต์อันเนื่องมาจากสัญญาณภายนอกที่ป้อนเข้ามายังขา INT0 บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

สำหรับบิต 6 ของรีจิสเตอร์ IE ไม่มีการใช้งาน ต้องกำหนดให้เป็น "0" เสมอ

2.3.2 รีจิสเตอร์จัดลำดับความสำคัญการตอบสนองการอินเตอร์รัปต์หรือ IP (Interrupt Priority register)

มีแอดเดรสอยู่ที่ 0B8H ในพื้นที่ของรีจิสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิต สามารถเข้าถึงได้ในระดับบิต ใช้ในการเลือกลำดับความสำคัญของการตอบสนองการอินเตอร์รัปต์ว่า ต้องการให้ตอบสนองสัญญาณอินเตอร์รัปต์จากแหล่งกำเนิดใดเป็นลำดับก่อนหลัง ถ้าต้องการให้การอินเตอร์รัปต์จากแหล่งกำเนิดใดมีความสำคัญสูงสุด ให้กำหนดที่บิดนั่นเป็น "1" ดังมีรายละเอียดของรีจิสเตอร์ IP ดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
-	-	PT2	PS	PT1	PX1	PT0	PX0

PT2 (Timer 2 interrupt priority bit) : ใช้ในการกำหนดความสำคัญของการอินเตอร์รัปต์อันเนื่องมาจากการโอเวอร์โฟลวหรือการแคปเจอร์ในไทเมอร์/เคาน์เตอร์2 จะมีเฉพาะใน MCSเบอร์ AT89C52 และในกนุกรม AT89Sxx เท่านั้น บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

PS (Serial port interrupt priority bit) : ใช้ในการกำหนดความสำคัญของการอินเตอร์รัปต์อันเนื่องมากจากการรับหรือส่งข้อมูล ทางพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51 สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

PT1 (Timer 1 interrupt priority bit) : ใช้ในการกำหนดความสำคัญของการอินเตอร์รัปต์อันเนื่องมาจากการโอเวอร์โฟลวในไทเมอร์/เคาน์เตอร์ 1 บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

PX1 (External interrupt 1 priority bit) : ใช้ในการกำหนดความสำคัญของการอินเตอร์รัปต์อันเนื่องมาจากสัญญาณภายนอกที่ป้อนเข้ามายังขา INT1 บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

PT0 (Timer 0 interrupt priority bit) : ใช้ในการกำหนดความสำคัญของการอินเตอร์รัปต์อันเนื่องมาจากการโอเวอร์โฟลวในไทเมอร์/เคาน์เตอร์ 0 บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

PX0 (External interrupt 0 priority bit) : ใช้ในการกำหนดความสำคัญของการอินเตอร์รัปต์อันเนื่องมาจากสัญญาณภายนอกที่ป้อนเข้ามายังขา INT0 บิตนี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

สำหรับบิต 6 และ 7 ของรีจิสเตอร์ IP ไม่มีการใช้งาน ต้องกำหนดให้เป็น "0" เสมอ

2.3.3 การอินเตอร์รัปต์จากไทเมอร์/เคาน์เตอร์ 0 และ 1

แหล่งกำเนิดอินเตอร์รัปต์นี้จัดเป็นแหล่งกำเนิดอินเตอร์รัปต์ภายในแบบหนึ่ง โดยใช้การเกิดโอเวอร์โฟลวจากการนับค่าในไทเมอร์/เคาน์เตอร์ ภายในไมโครคอนโทรลเลอร์ MCS-51 เมื่อไทเมอร์ 0 เกิดการโอเวอร์โฟลวก็จะทำการเซตบิต TF0ในรีจิสเตอร์ TCON และถ้าไทเมอร์ 1 เกิดโอเวอร์โฟลว บิต TF1 ในรีจิสเตอร์ TCON จะได้รับการเซตเช่นเดียวกัน

ค่าแอดเดรสอินเตอร์รัปต์เวกเตอร์ของการอินเตอร์รัปต์แบบนี้อยู่ที่ 000BH สำหรับไทเมอร์ 0 และ 001BH สำหรับไทเมอร์ 1

อย่างไรก็ตามการอินเตอร์รัปต์แบบนี้จะเกิดขึ้นหรือมีการตอบสนองก็ต่อเมื่อมีการเอ็นเอเบิลการอินเตอร์รัปต์ โดยการเซนบิต EA ,ET0 และ ET1 ในรีจิสเตอร์ IE

2.3.4 การอินเตอร์รัปต์จากพอร์ตอนุกรม

แหล่งกำเนิดอินเตอร์รัปต์นี้จัดเป็นแหล่งกำเนิดอินเตอร์รัปต์ภายในแบบหนึ่ง เมื่อวงจรพอร์ตอนุกรมส่งหรือรับข้อมูลเสร็จสมบูรณ์ก็จะกำเนิดสัญญาณอินเตอร์รัปต์ขึ้น โดยการเซตบิต R1 ในกรณีรับข้อมูลและบิต TI ในกรณีส่งข้อมูลบิต RI และ TI อยู่ในรีจิสเตอร์ SCON ซึ่งจะได้กล่าวถึงต่อไปในบทที่ว่าด้วยเรื่องพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51

ค่าแอดเดรสอินเตอร์รัปต์เวกเตอร์ของการอินเตอร์รัปต์แบบนี้อยู่ที่ 0023H การอินเตอร์รัปต์ในแบบนี้สามารถแทนได้ด้วยการออร์กันของบิต RI และ TI

2.4 พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51

2.4.1 รีจิสเตอร์บัฟเฟอร์ของพอร์ตอนุกรมหรือ SBUF (Serial data buffer register)

แอดเดรสอยู่ที่ 99H ในพื้นที่ของรีจิสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิตแบ่งออกเป็น 2 ส่วนคือ รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล (transmit buffer register) และรับข้อมูล (receive buffer register) เมื่อมีการเขียนข้อมูลมายังรีจิสเตอร์ SBUF ข้อมูลนั้นจะถูกส่งต่อออกไปยังบัฟเฟอร์สำหรับส่งข้อมูล เพื่อส่งออกจากไมโครคอนโทรลเลอร์ผ่านทางขา TxD หรือ ขา P3.1 ในกรณีที่มีการอ่านข้อมูลจากรีจิสเตอร์ SBUF ข้อมูลจะถูกส่งผ่านไปยังรีจิสเตอร์บัฟเฟอร์สำหรับรับข้อมูลเพื่อส่งต่อไปยังไมโครคอนโทรลเลอร์ต่อไป สำหรับการรับข้อมูลอนุกรมจากภายนอกนั้นจะผ่านมาทางขา RxD หรือ P3.0 ของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

2.4.2 รีจิสเตอร์ควบคุมการทำงานของพอร์ตอนุกรมหรือ SCON (Serial port Control Register)

เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 98H ในพื้นที่ของรีจิสเตอร์ SFR สามารถเข้าถึงได้ในระดับบิต มีรายละเอียดการทำงานดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

SM0-SM1 (Serial port mode bit 0-1) : ใช้ในการเลือกโหมดการทำงานของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51 ดังมีรายละเอียดต่อไปนี้

(multiprocessor) ในการทำงานของโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 ถ้าบิตนี้เป็น "1" บิต RI จะไม่แอกทีฟถ้าบิตที่ 9 ที่รับเข้ามาเป็น "0" (ข้อมูลบิตที่ 9 เก็บไว้ที่บิต RB8) ในการทำงานโหมด 1 ถ้าบิตนี้เซต บิต RI จะไม่แอกทีฟถ้ายังไม่ได้รับบิตหยุด ส่วนในโหมด 0 บิตนี้ไม่มีการใช้งาน

REN (Enable serial reception) : ใช้ในการเอ็นเอเบิลการรับข้อมูลของพอร์ตอนุกรม ทำการเซตและเคลียร์ด้วย กระบวนการทางซอฟแวร์ ถ้าต้องการให้มีการรับข้อมูลต้องเซตบิตนี้ให้เป็น "1"

TB8 : ใช้สำหรับเก็บข้อมูลบิตที่ 9 ที่ต้องการส่งออกไปในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 เซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

RB8 : ใช้สำหรับรับข้อมูลบิตที่ 9 ที่เข้ามาในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 แต่ถ้าหากพอร์ตอนุกรมทำงานอยู่ในโหมด 1 และบิต SM2 เป็น

"0" ข้อมูลที่บิต RB8 คือข้อมูลของบิตหยุด (STOP bit) สำหรับในการทำงานโหมด 0 บิตนี้จะไม่ใช้งาน บิต RB8 นี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

TI (Transmit Interrupt flag) : ใช้ในการแสดงการเกิดอินเตอร์รัปต์เมื่อมีการส่งข้อมูลออกจากพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์เมื่อทำการส่งข้อมูลบิตที่ 8 ไปเรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่น บิตนี้จะเซตเมื่อมีการเริ่มต้นส่งบิตหยุดออกไป การที่จะเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

RI (Receive Interrupt flag) : ใช้แสดงการเกิดอินเตอร์รัปต์เมื่อมีการรับข้อมูลเข้าสู่พอร์ตอนุกรม สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อทำการรับข้อมูลบิตที่ 8 เรียบร้อยแล้วในการทำงานของโหมด 0 ส่วนในการทำงานของโหมดอื่น บิตนี้จะเซตเมื่อมีสามารถรับบิตหยุดของข้อมูลอนุกรมไปได้ครึ่งทางแล้ว ยกเว้นในกรณีที่บิต SM2 มีการเซตบิตนี้จะเซตได้ก็ต่อเมื่อการรับบิตหยุดหรือบิตที่ 9 เกิดขึ้นอย่างสมบูรณ์แล้ว การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์

2.4.3 โหมดการทำงานของพอร์ตอนุกรมใน MCS-51

พอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 สามารถเลือกการทำงานได้ 4 โหมด คือ

1. โหมด 0 เป็นการกำหนดให้พอร์ตอนุกรมทำงานในลักษณะชิฟต์รีจิสเตอร์
2. โหมด 1 เป็นการกำหนดให้เป็น UART ขนาด 8 บิต สามารถเลือกอัตราบอดได้
3. โหมด 2 เป็นการกำหนดให้เป็น UART ขนาด 9 บิต โดยมีอัตราบอดคงที่
4. โหมด 3 เป็นการกำหนดให้เป็น UART ขนาด 9 บิต สามารถเลือกอัตราบอดได้

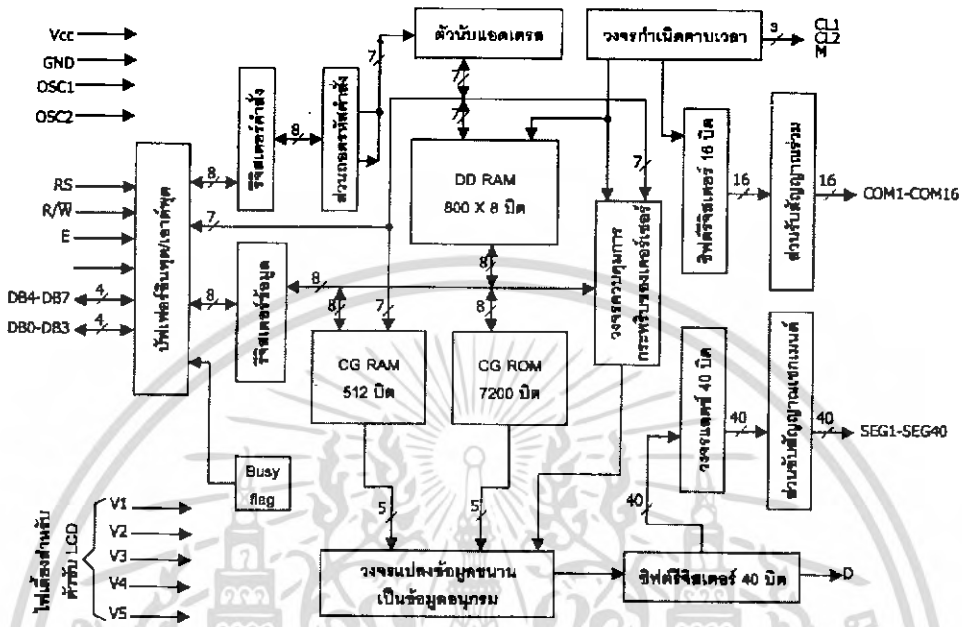
การเลือกโหมดทำได้ด้วยการกำหนดข้อมูลให้แก่บิต SM0 และ SM1 ในรีจิสเตอร์SCON

2.5 LCD MODULE

2.5.1 ส่วนประกอบของ LCD MODULE ประกอบด้วย 3 ส่วนหลัก

1) ตัวแสดงผล(display) ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็นโดยอาศัยแสงจากภายนอก ดังนั้นจึงต้องมีมุมในการมองข้อมูลที่แสดงผลบนจอ LCD
2) ตัวควบคุม(controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาควบคุมการทำงานของโมดูล LCD เช่น ลบจอภาพแสดงตัวอักษร หรือเลื่อนเคอร์เซอร์ เป็นต้น ตัวควบคุมนี้ใช้ชิปควบคุมโดยเฉพาะชิปที่นิยมใช้คือ เบอร์HD44780และHD61830 โดยHD44780จะใช้ควบคุม LCD แบบอักขระส่วนHD61830 ใช้ควบคุม LCD แบบกราฟิก
3) ตัวขับ(driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงข้อมูลตามที่กำหนด ชิปที่ใช้ทำหน้าที่เป็นตัวขับนี้ได้แก่ เบอร์HD44100H และ เบอร์ MSM5259

โครงสร้างภายในของตัวควบคุม LCD MODULE



รูปที่ 2.5 รูปไดอะแกรมการทำงานของ LCD MODULE แบบอักขระ

2.5.2 LCD MODULE ขนาด 16 ตัวอักษร 2 บรรทัด(LCD 16x2)

โมดูล LCD ขนาด 16x2 มีการใช้งานทั้งสิ้น 16 ขา การทำงานของแต่ละขามีดังนี้

Vss (ขา 1) : ต่อกราวด์

Vdd (ขา 2) : ต่อไฟเลี้ยง 5 โวลต์

Vo (ขา 3) : เป็นขาอินพุตรับแรงดันเพื่อปรับความเข้มของการแสดงผล

RS (ขา 4): เป็นขาอินพุตใช้ในการแยกชนิดของข้อมูล ที่ทำการประมวลผลในขณะนั้นว่าเป็นคำสั่งสำหรับรีจิสเตอร์ IR หรือเป็นข้อมูลสำหรับรีจิสเตอร์ DR โดยถ้าขานี้เป็น "0" ข้อมูลที่ส่งมาจะเป็นคำสั่ง แต่ถ้าขานี้เป็น "1" ข้อมูลที่ส่งมาจะเป็นข้อมูลสำหรับการแสดงผล

R/W (ขา 5): เป็นขาที่ใช้เลือกการอ่านหรือเขียนข้อมูลกับ โมดูล LCD ถ้าเป็น "0" เป็นการกำหนดให้เขียนข้อมูล แต่ถ้าเป็น "1" จะเป็นการอ่านข้อมูล

E (ขา 6) : เป็นขาสำหรับรับสัญญาณ พัลส์เอ็นนาเบิลโมดูล LCD ให้ทำงาน

D0-D7(ขา7-14) : เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่าง LCD กับอุปกรณ์ภายนอกขนาด 8 บิต

2.5.3 การเขียนคำสั่งและข้อมูลให้แก่ LCD MODULE

กำหนดโหมดการทำงานให้แก่โมดูล LCD ก่อนจากนั้นจึงค่อยส่งข้อมูล(data)ที่ต้องการแสดงผล เนื่องจากบัสข้อมูลของโมดูล LCD มี 8 เส้นคือ D0-D7 และใช้เป็นทางผ่านของทั้งคำสั่งและข้อมูล ดังนั้นในการส่งคำสั่งและข้อมูลจึงต้องอาศัยการกำหนดสัญญาณลอจิกที่ขา RS ถ้าหากที่ขา RS ได้ลอจิก "0" หมายความว่า ข้อมูลที่ป้อนให้แก่โมดูล LCD ขณะนั้นเป็นคำสั่ง ในทางตรงข้าม หากขา RS ได้รับลอจิก "1" ข้อมูลที่ป้อนให้ขณะนั้นเป็นข้อมูลที่ใช้ในการแสดงผล

เมื่อต้องการเขียนหรืออ่านข้อมูลใน CGRAM และ DDRAM เริ่มต้นต้องกำหนดแอดเดรสที่ต้องการอ่านหรือเขียนก่อน โดยใช้คำสั่งเลือกแอดเดรส จากนั้นกำหนดให้ขา RS เป็น "1" เพื่อแจ้งให้ตัวควบคุมภายในโมดูล LCD ทราบว่า ข้อมูลที่ปรากฏต่อไปนี้เป็นข้อมูลปกติไม่ใช่คำสั่ง ในกรณีที่ต้องการอ่านข้อมูลต้องกำหนดให้ขา R/W เป็น "1" ข้อมูลขนาด 8 บิต (หรือ 4 บิต) ก็จะปรากฏบนบัสข้อมูล โดยข้อมูลที่อ่านออกมาได้จะเป็นข้อมูลจากแอดเดรสของ CGRAM หรือ DDRAM ตามที่ต้องการ

2.5.4 จังหวะการทำงานของ LCD MODULE

ในการติดต่อกับโมดูล LCD จะต้องมีการหน่วงเวลาหลังจากที่ทำการส่งรหัสคำสั่งหรือข้อมูล เนื่องจากต้องรอให้คอนโทรลเลอร์ภายใน LCDโมดูล แปลความหมายของรหัสคำสั่งให้เรียบร้อยก่อนจากนั้นจึงจะรับข้อมูลหรือดำเนินการต่อไป

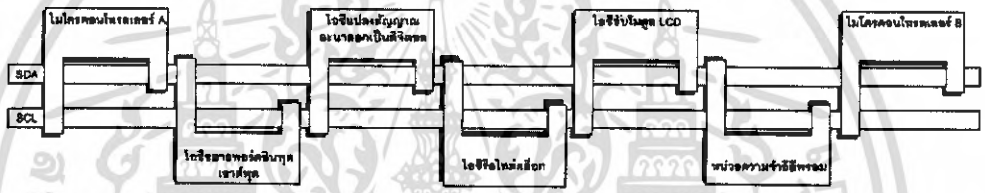
ดังนั้นในการใช้งานโมดูล LCD ผู้เขียนโปรแกรมต้องมีโปรแกรมเพื่อหน่วงเวลารอให้โมดูล LCD พร้อมทำงานด้วย โดยเมื่อเริ่มจ่ายไฟให้แก่โมดูล LCD ต้องรอประมาณ 10 มิลลิวินาที เพื่อให้โมดูล LCD ทำการเตรียมความพร้อมหรืออินิเชียล(initial)หลังจากนั้นก็จะกำหนดลอจิกให้แก่ขา RS ของโมดูล LCD แล้วต้องหน่วงเวลาอีกประมาณ 2 มิลลิวินาทีเพื่อให้คอนโทรลเลอร์ในโมดูล LCD แปลความหมายของลอจิกที่ขา RS ว่า ข้อมูลต่อไปที่จะได้รับนั้นเป็นรหัสคำสั่งหรือเป็นข้อมูลที่ต้องการแสดงผล จากนั้นจะเป็นการส่งข้อมูลมารอที่บัสข้อมูล D0-D7 (กรณีทำงานในโหมด 8 บิต) ขั้นตอนต่อไปจะเป็นการส่งสัญญาณพัลส์ไปที่ขา E เพื่อเอ็นนาเบิลโมดูล LCD ให้รับข้อมูลจากบัสข้อมูลเข้าไป โดยพัลส์ที่ป้อนเข้าที่ขา E ของโมดูล LCD ต้องเป็นพัลส์ขอบขาขึ้น จากนั้นทำการหน่วงเวลา 2 มิลลิวินาที

บทที่ 3
ระบบบัส I^2C

3.1 ระบบบัส I^2C

3.1.1 ความรู้เบื้องต้นเกี่ยวกับ I^2C

I^2C ย่อมาจาก Inter-IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซี ได้รับการพัฒนาขึ้นโดยฟิลิปส์ (Philips) ด้วยจุดหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อสั่งงาน และควบคุมภายใต้สัญญาณเพียง 2 เส้น เส้นหนึ่งคือ สายข้อมูล อีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน



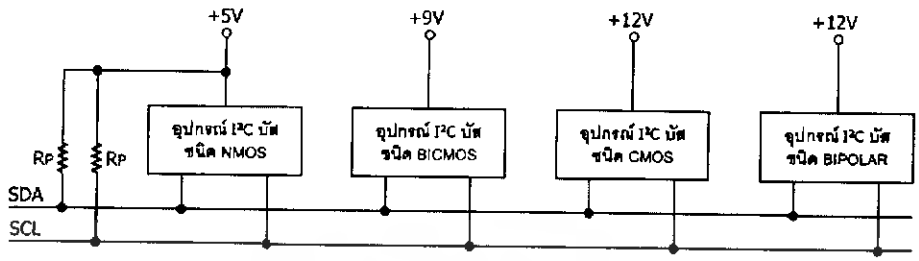
รูปที่ 3.1 ผังการแสดงการเชื่อมต่อของอุปกรณ์ต่างๆบนบัส I^2C

แสดงผังของการเชื่อมต่ออุปกรณ์ต่างๆบนบัส I^2C อุปกรณ์ที่ทำการเชื่อมต่อบนบัส I^2C มีหลากหลาย ไม่ว่าจะเป็นไอซีขยายพอร์ตอินพุตเอาต์พุต (I/O Expander) ,ไอซีแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC) และแปลงสัญญาณดิจิตอลเป็นอะนาลอก (DAC) ,ไอซีรีลไทม์คล็อก (RTC) ,ไอซีขับโมดูล LCD หน่วยความจำEEPROM และไมโครคอนโทรเลอร์

3.1.2 คุณสมบัติทั่วไปของบัส I^2C

สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data line) และสายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock line) ซึ่งเป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพูลอัปกับแรงดัน 5 โวลต์ ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติ(standard mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (fast mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF



รูปที่ 3.2 การต่อตัวต้านทานพูลอัปบนสายสัญญาณในระบบบัส

3.1.3 สภาวะที่เกิดขึ้นบนบัส I²C

มี 5 สภาวะดังนี้

1) **บัสว่าง (Bus not busy)** เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้งคู่ หมายความว่า การถ่ายทอดข้อมูลเกิดขึ้นได้

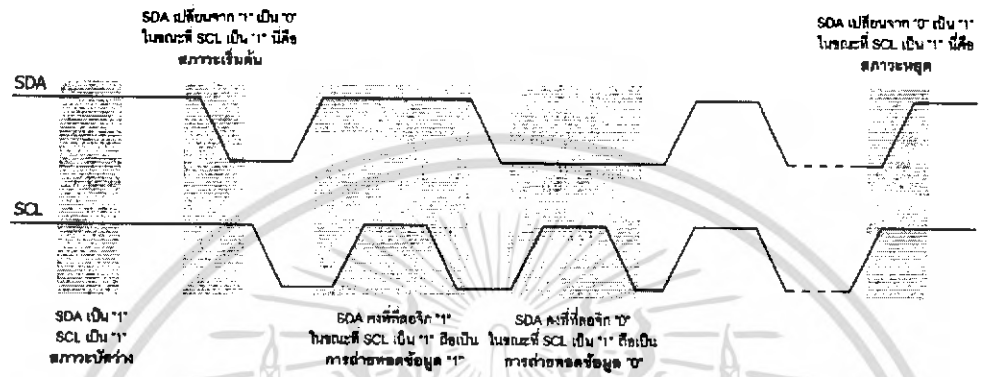
2) **เริ่มต้นการถ่ายทอดข้อมูล (Start data transfer)** เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า **สภาวะเริ่มต้น (Start)**

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

4) **ข้อมูลดำรงอยู่บนบัส (Data valid)** สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่าเป็น "0" หรือ "1" ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอด ช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น

5) **รับรู้ข้อมูล (acknowledge)** เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่า **บิตรับรู้ (acknowledge bit)** มีสถานะเป็นลอจิกสูง หลังจากส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่ง

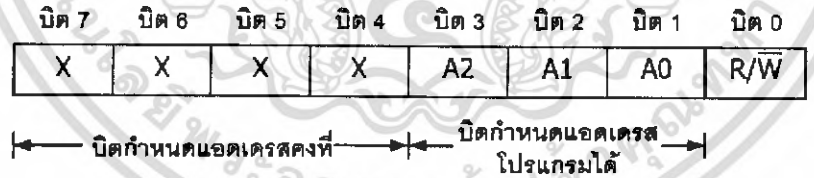
สัมพันธ์กับสัญญาณนาฬิกาเพื่อตอบสนองบิตรับรู้ที่ส่งมาจากตัวส่ง ทางด้านตัวรับจะส่งบิตรับรู้ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้เพื่อตอบสนองให้ทราบว่าได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว



รูปที่ 3.3 ไดอะแกรมเวลาแสดงสถานะต่างๆในบัส I^2C

3.1.4 การทำงานบนบัส I^2C

อันดับแรกในการทำงานบนบัส I^2C คือการอ้างถึงอุปกรณ์แต่ละตัว ซึ่งในที่นี้จะอธิบายถึงการอ้างถึงทั้งสองแบบ



รูปที่ 3.4 รูปแบบของข้อมูลกำหนดแอดเดรสที่ใช้ในการอ้างถึงแบบ 7 บิต

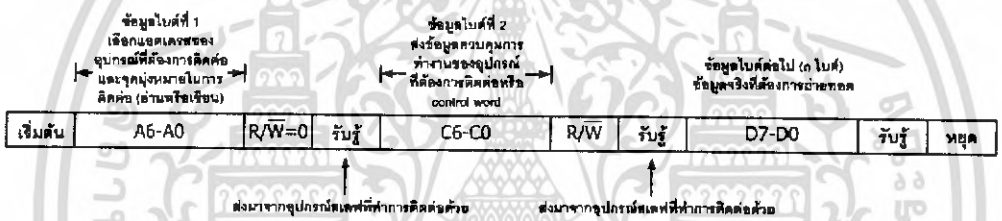
การอ้างแบบ 7 บิต (7-bit addressing)

ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้นคือ ข้อมูลที่ใช้ในการอ้างถึงอุปกรณ์ที่ต้องการติดต่อ หรือข้อมูลกำหนดแอดเดรส ใน 7 บิตบนรวมทั้งบิต MSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็นบิตกำหนดแอดเดรสคงที่ (fixed address bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิต ไม่สามารถเปลี่ยนแปลง

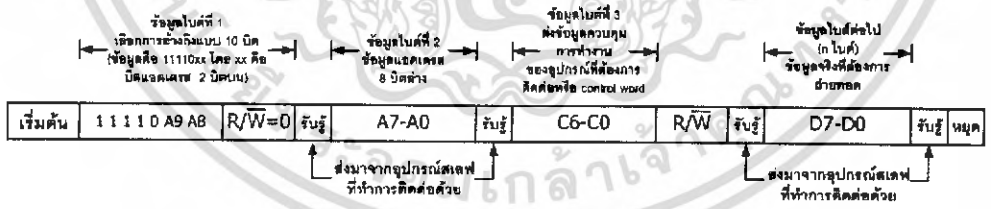
แก้ไขได้ ถัดมาอีก 3 บิตเป็นบิตกำหนดแอดเดรสที่สามารถโปรแกรมได้ (programmable address bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0-A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I^2C ส่วนในบิต LSB เป็นบิตที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์กับอุปกรณ์สเลฟตัวนั้นๆ หากบิต LSB เป็น "0" หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ

ข้อมูลในไบต์ต่อมาคือ **ข้อมูลควบคุม (control byte)** ในอุปกรณ์แต่ละตัวมีการกำหนดข้อมูลควบคุมที่แตกต่างกันออกไป

ข้อมูลในไบต์ต่อมาคือ **ข้อมูลที่ทำการถ่ายทอดจริง (data)** หลังจากที่มีการถ่ายทอดข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกับมาด้วยทุกครั้ง เพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้



รูปที่ 3.5 รูปแบบของข้อมูลอนุกรมที่ใช้ติดต่อกับอุปกรณ์บัส I^2C เมื่อใช้การอ้างถึงแบบ 7 บิต



รูปที่ 3.6 รูปแบบของข้อมูลอนุกรมที่ใช้ติดต่อกับอุปกรณ์บัส I^2C เมื่อใช้การอ้างถึงแบบ 10 บิต

การอ้างถึงแบบ 10 บิต (10-bit addressing)

ในการอ้างถึงแบบนี้ ยังคงใช้รูปแบบข้อมูลอนุกรมที่เหมือนกับแบบ 7 บิต หากแต่จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อย โดยในข้อมูลไบต์แรกหลังจากเกิดสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของอุปกรณ์ที่ต้องการติดต่อ ในบิต

LSB ของข้อมูลไบต์แรกยังคงเป็นการกำหนดว่าต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบต์ต่อมาเป็นข้อมูลแอดเดรสในไบต์ที่ 2 ของอุปกรณ์ที่ต้องการติดต่อด้วย ข้อมูลถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนั้นก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อ

3.2 DS1307 ไอซีสร้างฐานเวลาจริงหรือรีลไทม์คล็อก(RTC)

3.2.1 รายละเอียดของรีลไทม์คล็อก(RTC)

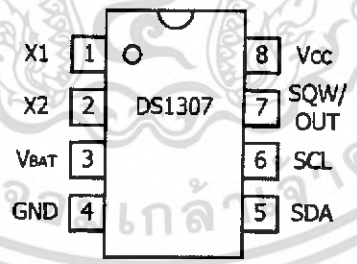
ผู้ผลิตคือ ดัลลัสเซมิคอนดักเตอร์(Dallas semiconductor)มีหน้าที่สร้างฐานเวลาจริงให้แก่ระบบไมโครคอนโทรลเลอร์ โดย DS1307 จะให้ข้อมูลเกี่ยวกับเวลาทั้งหมด ไม่ว่าจะเป็นค่าของเวลาที่ละเอียดถึงหลักวินาที,นาที,ชั่วโมง,วันที่(date),วันในสัปดาห์(day),เดือนและปี โดยสามารถปรับวันเดือนปีให้ตรงตามปฏิทินได้อย่างถูกต้อง รวมถึงการกำหนดวันในปีอธิกสุรทินคุณสมบัติทางเทคนิคที่สำคัญมีดังนี้

-เป็นไอซีรีลไทม์คล็อกให้ข้อมูลตั้งแต่วินาทีจนถึงปี รวมถึงการปรับวันในปีอธิกสุรทินด้วย สามารถให้ข้อมูลเวลาได้อย่างเที่ยงตรงถึงปีคริสตศักราช 2100

-มีหน่วยความจำนอนโวลาไทล์แรม 56 ไบต์อยู่ภายใน สามารถใช้เก็บข้อมูลทั่วไปได้

-ใช้การเชื่อมต่อแบบระบบ I^2C

-มีวงจรตรวจจับไฟเลี้ยงต่ำหรือหายไปอย่างอัตโนมัติ และสามารถรักษาข้อมูลเวลาไว้ได้แม้ไม่มีไฟเลี้ยงไอซี



รูปที่ 3.7 การจัดขาของไอซี DS 1307 ไอซีสร้างฐานเวลาจริง (RTC)

การใช้งานของ DS1307

- **Vcc (ขา 8)**
- **GND (ขา 4)**

-VBAT(ขา 3) ใช้ต่อกับแบตเตอรี่ 3โวลต์เพื่อรักษาการทำงานของจงจรสร้างฐานเวลาของ DS1307 ให้คงอยู่ต่อไปแม้ไม่มีไฟเลี้ยง

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

-SQW/OUT(ขา7) ที่ขานี้จะมีสัญญาณรูปสี่เหลี่ยมส่งออกมา โดยสามารถเลือกความถี่ได้ 1Hz,4.069kHz,80792kHz และ 32kHz ในการใช้งานต้องต่อความต้านทาน 1k พูลอัปที่ขานี้ด้วย

-X1,X2(ขา1 และ 2) ใช้ต่อกับคริสตอลความถี่มาตรฐาน 32.768 kHz เพื่อใช้เป็นฐานเวลาในการสร้างค่าความจริง ในการใช้งานต้องต่อคริสตอลเข้ากับขาทั้งสองนี้และที่แต่ละขาต้องต่อตัวเก็บประจุค่าต่ำๆ ประมาณ 15 pF คร่อมกับขากราวด์

3.2.2 การทำงานของ DS1307

ไอซี DS1307 จัดการเชื่อมต่อในแบบบัส I2C โดยจะทำงานเป็นอุปกรณ์สเลฟเสมอ ดั้งนั้น การติดต่อเพื่อใช้งาน จึงต้องกำหนดรูปแบบตามที่กำหนดไว้ในการติดต่อแบบI2C วงจรออสซิลเลเตอร์ถือเป็นหัวใจหลักของไอซี เนื่องจากเป็นจุดเริ่มต้นของการสร้างข้อมูลเวลาจริง ในขณะที่ DS13078 ทำงานที่ขา SQW/OUT จะมีสัญญาณพัลส์สี่เหลียมส่งออกมาตลอดเวลาในกรณีที่มีการอีนาเบิลวงจรกำเนิดสัญญาณพัลส์ที่รีจิสเตอร์ควบคุม ค่าความถี่ของสัญญาณนี้สามารถเลือกได้ 4 ค่าคือ 1Hz,4.096kHz,8.192kHz และ 32kHz พร้อมกันนั้นก็จะมีการเก็บค่าของเวลาไว้ในหน่วยความจำนอนโวลาไทล์แรม ซึ่งมีขนาดรวม 64 ไบต์ แต่จัดสรรให้ใช้เก็บข้อมูลเวลา 8 ไบต์ และเป็นหน่วยความจำสำหรับเก็บข้อมูลทั่วไปสำหรับผู้ใช้งานอีก 56 ไบต์

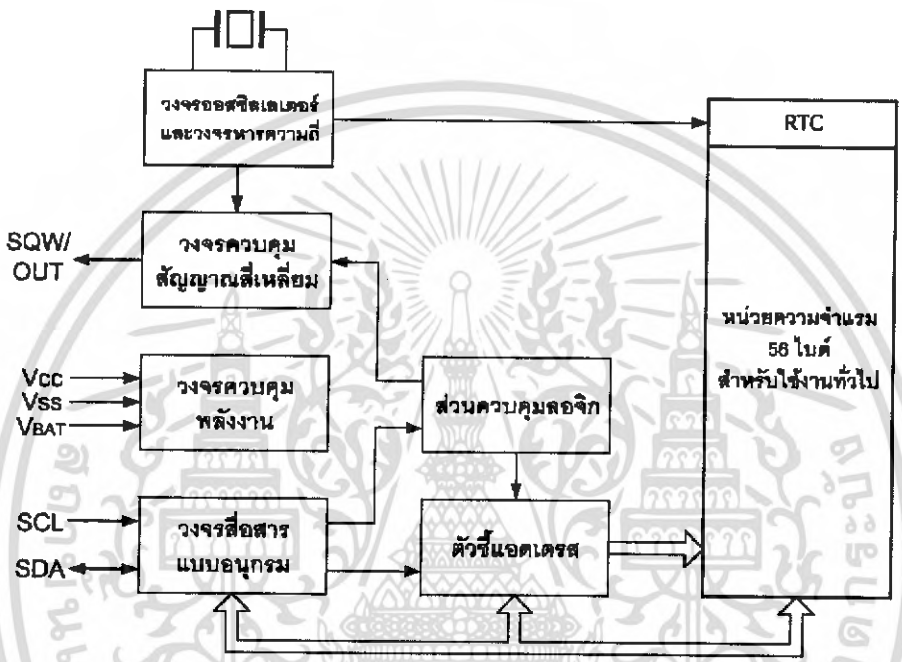
วงจรควบคุมพลังงานไฟฟ้าจะคอยตรวจสอบสภาวะของไฟเลี้ยงไอซี หากไฟเลี้ยงต่ำกว่า $1.25xV_{BAT}$ ก็จะควบคุมให้ DS1307 หยุดการทำงานรีเซตค่าตัวนับแอดเดรสภายใน ทำให้ไม่สามารถติดต่อกับ DS1307 ได้ ดังนั้นในการใช้งาน DS1307 ต้องระมัดระวังอย่าให้ไฟเลี้ยงตกต่ำกว่า $1.25xV_{BAT}$ หรือประมาณ 3.75 V ในกรณีที่ใช้ V_{BAT}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

3.2.3 การจัดสรรหน่วยความจำใน DS1307

หน่วยความจำภายใน DS1307 พื้นที่ 7 ไบต์แรกตั้งแต่แอดเดรส 00H-06H เป็นพื้นที่ของรีจิสเตอร์ควบคุมการทำงานของ DS1307 แสดงรายละเอียดของรีจิสเตอร์ค่าเวลาและรีจิสเตอร์ควบคุมของ DS1307

ด้วยการจัดสรรพื้นที่แบบนี้ ทำให้ผู้ใช้งานสามารถเรียกข้อมูลเวลาออกมาได้ตามที่ต้องการ โดยไม่จำเป็นต้องอ่านออกมาทั้งหมดก็ได้ ค่าของเวลาทั้งหมดจะอยู่ในรูปของเลขฐานสิบ สำหรับ

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



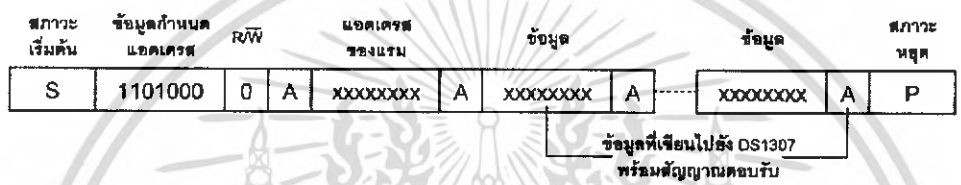
รูปที่ 3.8 โครงสร้างภายในของไอซีรีลไทม์คล็อกเบอร์ DS 1307

	บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0	ค่าของข้อมูล
00H	CH	ข้อมูลวินาที (หลักสิบ)			ข้อมูลวินาที (หลักหน่วย)				00-59
01H	X	ข้อมูลนาที (หลักสิบ)			ข้อมูลนาที (หลักหน่วย)				00-59
02H	X	12 ชั่วโมง / 24 ชั่วโมง	ชั่วโมง (หลักสิบ) / แอมพม	ข้อมูลชั่วโมง (หลักสิบ)	ข้อมูลชั่วโมง (หลักหน่วย)				01-12 / 00-23
03H	X	X	X	X	X	ข้อมูลวันในสัปดาห์			1-7
04H	X	X	ข้อมูลวันที่ (หลักสิบ)		ข้อมูลวันที่ (หลักหน่วย)				01-28/29 / 01-30 / 01-31
05H	X	X	X	ข้อมูลเดือน (หลักสิบ)	ข้อมูลเดือน (หลักหน่วย)				01-12
06H	ข้อมูลปี (หลักสิบ)				ข้อมูลปี (หลักหน่วย)				00-99
07H	OUT	X	X	SQWE	X	X	RS1	RS0	
08H-3FH	แรม 56 ไบต์								

รูปที่ 3.9 การจัดสรรหน่วยความจำแรมภายใน DS 1307 และรายละเอียดของรีจิสเตอร์เก็บค่าเวลาและรีจิสเตอร์ควบคุมของ DS 1307

3.2.4 โหมดการทำงานของ DS1307

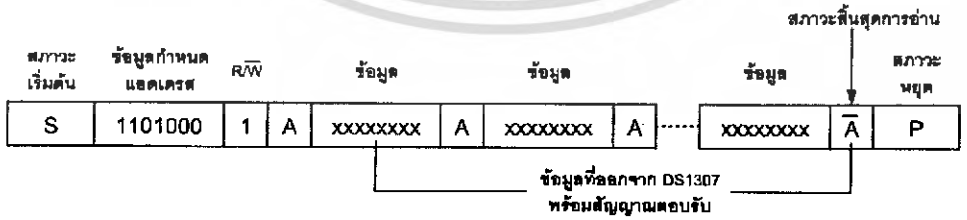
มีด้วยกัน 2 โหมดคือ โหมดเขียนข้อมูลและโหมดอ่านข้อมูล ในการใช้งาน DS1307 ตามปกติการใช้งานเฉพาะโหมดอ่านข้อมูลเท่านั้น เนื่องจากไมโครคอนโทรลเลอร์จะติดต่อกับ DS1307 เพื่ออ่านข้อมูลของเวลาไปใช้งาน โหมดการเขียนข้อมูลจะถูกใช้งานก็ต่อเมื่อต้องการตั้งค่าเวลาใหม่และต้องการเขียนข้อมูลลงในหน่วยความจำใช้งานทั่วไป อย่างไรก็ตามเมื่อเริ่มต้นติดต่อกับ DS1307 จำเป็นอย่างยิ่งที่จะต้องเข้าสู่โหมดการเขียนข้อมูลก่อนเพื่อกำหนดแอดเดรสที่ต้องการอ่านข้อมูล จากนั้นจึงเปลี่ยนโหมดการทำงานมาเป็นโหมดการอ่านข้อมูล



รูปที่ 3.10 รูปแบบของข้อมูลสำหรับติดต่อกับ DS 1307 ในโหมดการเขียนข้อมูล

โหมดการเขียนข้อมูล

เริ่มต้นเมื่อไมโครคอนโทรลเลอร์ทำการกำหนดสภาวะเริ่มต้น (START : S) จากนั้นส่งข้อมูลกำหนดแอดเดรส 1101000 ตามด้วยข้อมูลเลือกการเขียน นั่นคือค่า 0 จากนั้นจะรอการตอบรับจาก DS1307 เมื่อมีการตอบรับมาเรียบร้อย ก็เริ่มทยอยเขียนข้อมูลลงไปครั้งละแอดเดรส หลังจากเขียนข้อมูลในแต่ละแอดเดรส จะต้องหยุดรอการตอบรับจาก DS1307 ทุกครั้งจึงสามารถเขียนข้อมูลต่อไปได้เมื่อเขียนเรียบร้อยแล้วให้ส่งสภาวะหยุด (STOP : P) เป็นอันสิ้นสุดกระบวนการเขียนข้อมูล



รูปที่ 3.11 รูปแบบของข้อมูลสำหรับติดต่อกับ DS 1307 ในโหมดการอ่านข้อมูล

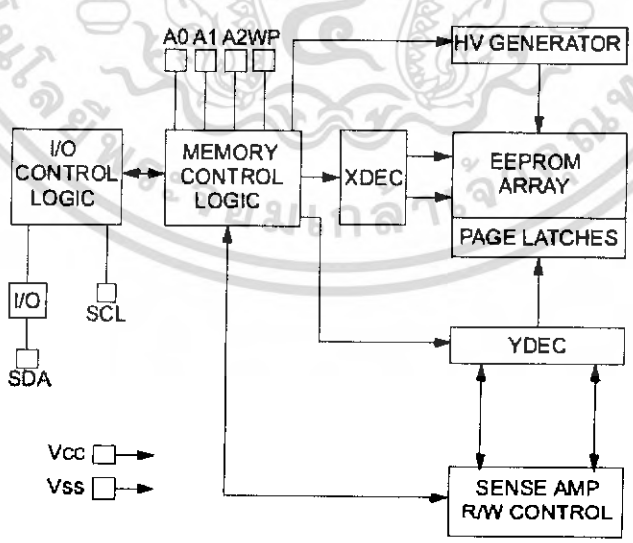
โหมดการอ่านข้อมูล

เริ่มต้นการทำงานเหมือนการโหมดการเขียนข้อมูลคือ ไมโครคอนโทรลเลอร์กำหนดสภาวะเริ่มต้นแล้วส่งข้อมูลกำหนดแอดเดรสตามด้วยข้อมูลเลือกการอ่าน ซึ่งเท่ากับ 1 จากนั้นรอการตอบรับจาก DS1307 เมื่อตอบรับเรียบร้อย DS1307 จะทยอยส่งข้อมูลออกมาให้ไมโครคอนโทรลเลอร์คราวละ 1 แอดเดรสหรือ 1 ไบต์ โดยแอดเดรสที่เลือกอ่านข้อมูลจะต้องมีการกำหนดมาก่อนล่วงหน้าด้วยโหมดการเขียนข้อมูลวิธีการง่ายๆคือ เข้าสู่โหมดการเขียนข้อมูลก่อน เมื่อถึงจังหวะที่ต้องเขียนข้อมูลให้ทำการสร้างสภาวะเริ่มต้นและส่งข้อมูลกำหนดแอดเดรสใหม่อีกครั้ง ตามด้วยเลือกโหมดการอ่านข้อมูล ข้อมูลที่ออกมาจาก DS1307 ก็จะเป็นข้อมูลจากแอดเดรสที่กำหนดไว้ก่อนหน้านี้

3.3 EEPROM

3.3.1 รายละเอียด EEPROM

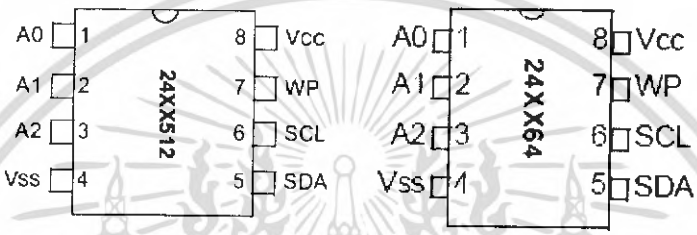
- เป็นหน่วยความจำภายนอกที่ใช้ระบบบัส I^2C ในการติดต่อสื่อสาร
- เป็นหน่วยความจำภายนอกชนิดที่สามารถเขียนและอ่านข้อมูลได้ โดยข้อมูลยังคงอยู่เมื่อแม้ ขณะที่ไม่มีไฟเลี้ยงICจึงทำให้หน่วยความจำชนิดนี้สามารถเก็บข้อมูลไว้ได้ ซึ่งเป็นลักษณะเด่นของหน่วยความจำ EEPROM
- ผู้ใช้งานสามารถใช้งานเขียนหรืออ่านข้อมูลผ่านระบบบัส I^2Cได้โดยง่าย ในปัจจุบันผู้ผลิตได้ผลิตให้มีหน่วยความจำหลายขนาดเพื่อให้สามารถเลือกใช้ได้ตามความเหมาะสม



รูปที่ 3.12 รายละเอียดโครงสร้างหลักของ EEPROM

3.3.2 คุณสมบัติของ I²C Serial EEPROM

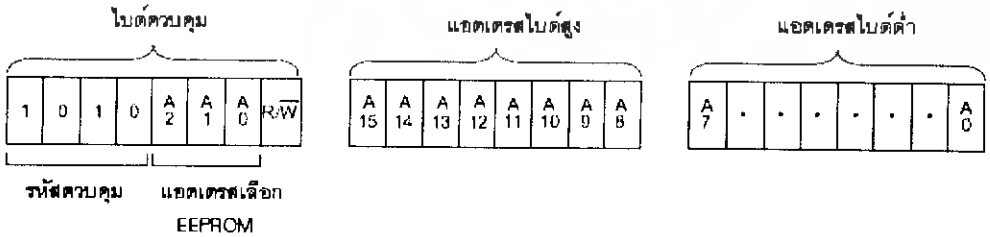
- ใช้สาย 2 เส้นในการเชื่อมต่อกับระบบ I^2C
- ใช้งานได้ที่ความถี่สูงสุดได้ถึง 400 KHz
- สามารถป้องกันการเขียนข้อมูลได้
- สามารถลบและเขียนได้ประมาณ 1,000,000 รอบ
- สามารถเก็บข้อมูลได้นานมากกว่า 200 ปี



รูปที่ 3.13 การจัดขามาตรฐานของ EEPROM ทั้ง 2 เบอร์

3.3.3 การใช้งาน

- ขา A0,A1,A2 เป็นขาสำหรับตั้งค่าแอดเดรสเพื่อเลือกใช้งานสามารถติดต่อใช้งาน EEPROMได้ทั้งหมดพร้อมกัน 8ตัว
- SDA และ SCL เป็นขาสำหรับเชื่อมต่อกับไมโครคอนโทรลเลอร์บนระบบบัส I2C
- ขา Write Protect (WP) ใช้สำหรับป้องกันการเขียนข้อมูลเมื่อต่อขานี้เข้ากับ Vss จะสามารถเขียนและอ่านข้อมูลได้ตามปกติ แต่ถ้าต่อขานี้เข้ากับ Vcc จะทำให้ไม่สามารถเก็บข้อมูลได้ แต่ยังสามารถอ่านข้อมูลได้เป็นปกติ



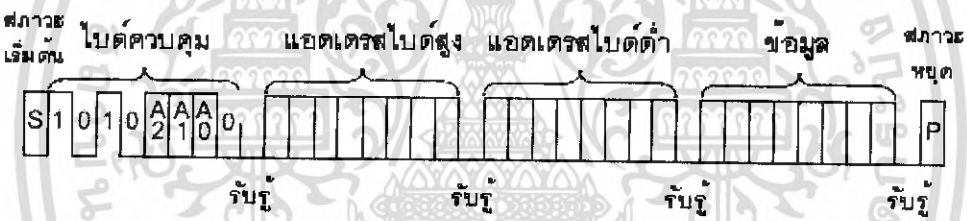
รูปที่ 3.14 แสดงแอดเดรสต่างๆของ EEPROM

การอ้างถึงแอดเดรสของ EEPROM

มีข้อมูลกำหนดแอดเดรส (Control Byte) เป็นข้อมูลไบต์แรกที่ได้รับหลังจากสภาวะเริ่มต้น จะมีรหัสควบคุมของ EEPROM 4 บิต คือ 1010 และตามด้วยข้อมูลแอดเดรสของ EEPROM ที่ต้องการอ้างถึงอีก 3 บิต คือ A2, A1 และA0 ในบิตสุดท้ายเป็นบิตสำหรับเลือกการเขียนและอ่านข้อมูล เมื่อบิตนี้เป็น "0" จะเป็นการเขียนข้อมูลและเมื่อเป็น "1" จะเป็นการอ่านข้อมูล จากนั้นตามด้วยแอดเดรสไบต์สูงและไบต์ต่ำของ EEPROM ที่ต้องการติดต่อเรียกใช้งานในแอดเดรสนั้นๆ

โหมดการเขียนข้อมูล

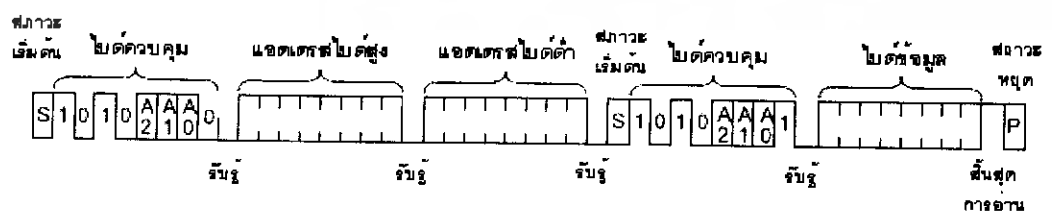
หลังจากสภาวะเริ่มต้นที่รับจากอุปกรณ์หลัก ซึ่งจะส่งรหัสควบคุม (4 บิต) แอดเดรสเลือก EEPROM (3 บิต) และตามด้วยลอจิก "0" สำหรับการเขียนข้อมูล จากนั้นจะรอรับสัญญาณรับรู้ หลังจากนั้นจึงจะส่งข้อมูลแอดเดรสไบต์สูง และจะรอรับสัญญาณรับรู้แล้วจึงส่งแอดเดรสไบต์ต่ำ หลังจากนั้นจะรอรับสัญญาณรับรู้อีกครั้งและส่งข้อมูลเป็นไบต์ที่ต้องการเขียนเก็บไว้ใน EEPROM แอดเดรสดังกล่าว หลังจากการเขียนเสร็จสิ้นจะหยุดการเขียนข้อมูลด้วยการส่งสภาวะหยุด



รูปที่ 3.15 รูปแบบของข้อมูลสำหรับติดต่อกับ EEPROM ในโหมดการเขียนข้อมูล

โหมดการอ่านข้อมูล

สำหรับโหมดการอ่านข้อมูลการทำงานจะเริ่มต้นเหมือนกับโหมดการเขียนข้อมูล จากนั้นจะเริ่มสภาวะเริ่มต้นอีกครั้งตามด้วยข้อมูลกำหนดแอดเดรสโดยบิตสุดท้ายเป็นลอจิก "1" สำหรับการอ่านข้อมูลเมื่อทำการรับข้อมูลจาก EEPROM จนเสร็จสิ้นแล้วจะส่งสัญญาณสภาวะสิ้นสุดการอ่านและตามด้วยสภาวะหยุด



รูปที่ 3.16 รูปแบบของข้อมูลสำหรับติดต่อกับ EEPROM ในโหมดการอ่านข้อมูล

บทที่ 4
พอร์ตอนุกรม (Serial Port)

4.1 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมแบ่งออกได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ดังนั้นการสื่อสารแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา,ข้อมูลและกราวด์

การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัสคือการรับและส่งข้อมูลไปในสาย โดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (baudrate) มีหน่วยเป็น บิตต่อวินาที (bit per second : bps)

รูปแบบการถ่ายทอดข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนคือ

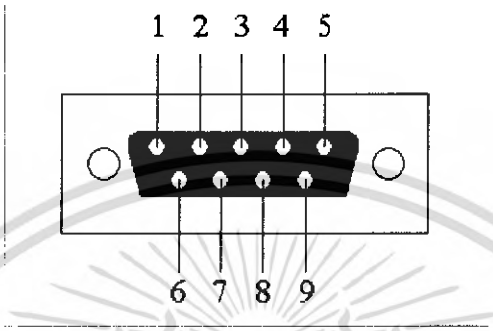
1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรมจะมีขนาด 5-8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1-2 บิต

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver/Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 100, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์

4.2 มาตรฐานพอร์ตอนุกรมแบบ RS-232

มาตรฐาน RS-232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment :DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่า อุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรเลอร์หรือไมโครคอนพิวเตอร์ ซึ่ง

มีความสามารถในการสร้างข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232



รูปที่ 4.1 คอนเน็กเตอร์อนุกรม 9 ขาหรือแบบ DB-9

4.3 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

จะใช้คอนเน็กเตอร์แบบ DB-9 ตัวผู้ รายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS-232

-Data Carrier Detect : DCD หรืออาจเรียกว่า **Carrier Detect : CD** ขานี้จะแอกทีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็ม สำหรับการใชงานปกติขานี้จะไม่ถูกใช้งานมากนัก

-Receive Data : RD หรือ **RxD** ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์บัฟเฟอร์

-Transmitted Data : TD หรือ **TxD** ใช้ส่งข้อมูลออกจากคอมพิวเตอร์ โดบนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

-Data Terminal Ready : DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าต้องการติดต่อด้วย โดยขา DTR นี้ต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทาง และขา DTR ของอุปกรณ์ปลายทางต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์

-Signal Ground : GND กราวด์ของระบบ

-Data Set Ready : DSR ขานี้จะใช้คู่กับขาDTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR รี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR

-Request To Send : RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS

-Clear To Send : CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ข้อมูลที่ขา TxD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

-Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ จะใช้งานก็ต่อเมื่อกับคโปรแกรมที่มีการตรวจสอบสัญญาณนี้เท่านั้น

คอนเน็กเตอร์ DB-9	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	Data Carrier Detect : DCD	อินพุต
2	Received Data : RxD	อินพุต
3	Transmitted Data : TxD	เอาต์พุต
4	Data Terminal Ready : DTR	เอาต์พุต
5	Signal Ground : GND	-
6	Data Set Ready : DSR	อินพุต
7	Request To Send : RTS	เอาต์พุต
8	Clear To Send : CTS	อินพุต
9	Ring Indicator : RI	อินพุต

ตารางที่ 4.1 แสดงการจัดขาคอนเน็กเตอร์อนุกรมตามมาตรฐาน RS-232 แบบ DB-9

4.4 UART

UART มาจากคำว่า Universal Asynchronous Receiver Transmitter หมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัส

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัส แล้วส่งออกไปและทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์

ภายใน UART จะมีส่วนวงจรสร้างบอดเรดแบบโปรแกรมได้ (programmable buadrate generator) โดยกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิต UART สามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็กซ์ (half duplex) และฟูลดูเพล็กซ์ (full duplex) โดยการส่งแบบฮาล์ฟดูเพล็กซ์เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กซ์ นั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

ชนิดของ UART

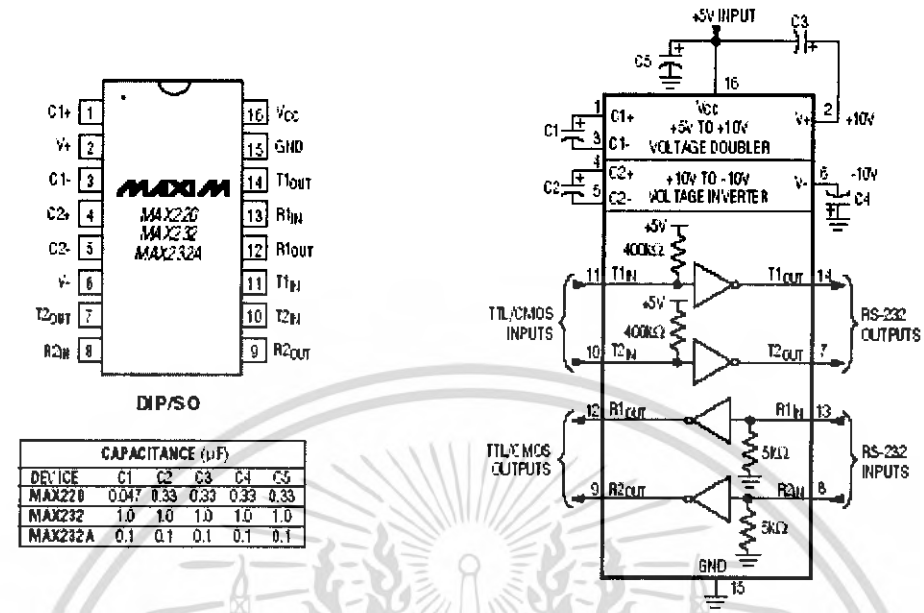
ในเครื่องคอมพิวเตอร์ทั่วไปมี UART ที่ใช้งานกันอยู่ 2 เบอร์คือ 8250 ซึ่งเป็น UART มาตรฐานที่ใช้มายาวนาน UART เบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเดียวกัน ทำให้การรับส่งข้อมูลถูกจำกัดอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้น

UART อีกเบอร์หนึ่งคือ 16450 มีความสามารถรับส่งข้อมูลได้ที่ความเร็ว 115,200 บิตต่อวินาที และเพิ่มรีจิสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิพต์รีจิสเตอร์แบบ FIFO (First In First Out) ขนาด 16 บิตเข้าไป ทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ 256 กิโลบิตต่อวินาที

4.5 การเชื่อมต่อกับพอร์ตอนุกรมของคอมพิวเตอร์

การใช้งานพอร์ตอนุกรมของไมโครคอนโทรเลอร์ MCS-51 นิยมใช้การติดต่อเพื่อแลกเปลี่ยนข้อมูลกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมในมาตรฐาน RS-232 เป็นส่วนใหญ่ แต่เนื่องจากระดับสัญญาณของพอร์ตอนุกรม RS-232 มีระดับตั้งแต่ 3 ถึง 12 โวลต์ และ -3 ถึง -12 โวลต์ ในขณะที่ระดับสัญญาณของไมโครคอนโทรเลอร์ MCS-51 อยู่ในระดับทีทีแอล ดังนั้นจึงไม่สามารถเชื่อมต่อพอร์ตอนุกรมของไมโครคอนโทรเลอร์ MCS-51 เข้ากับพอร์ตอนุกรมของคอมพิวเตอร์ได้โดยตรง จึงต้องอาศัยการเชื่อมต่อผ่านไอซีพิเศษที่ทำหน้าที่ในการแปลงระดับสัญญาณ

ไอซีที่ทำหน้าที่ในการแปลงระดับสัญญาณนี้ ต้องทำการแปลงข้อมูลส่งของไมโครคอนโทรเลอร์ MCS-51 จากระดับทีทีแอลไปเป็นระดับของ RS-232 และทำการแปลงข้อมูลรับจากคอมพิวเตอร์จากระดับของ RS-232 เป็นระดับทีทีแอลเพื่อให้สามารถถ่ายทอดไปยังไมโครคอนโทรเลอร์ได้อย่างสมบูรณ์ ไอซีที่ใช้คือ MAX 232



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

รูปที่ 4.2 รายละเอียดเบื้องต้นของไอซีแปลงสัญญาณเพื่อเชื่อมต่อกับพอร์ตอนุกรมของคอมพิวเตอร์

บทที่ 5

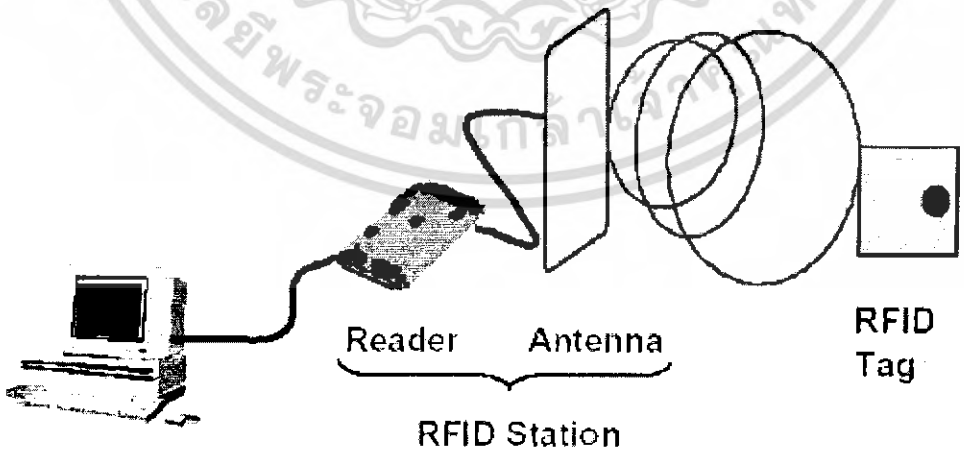
RFID

5.1 RFID คืออะไร

RFID ย่อมาจาก Radio Frequency Identification เป็นระบบระบุเอกลักษณ์ของวัตถุด้วยคลื่นความถี่วิทยุ ที่ได้ถูกพัฒนามาตั้งแต่ปี ค.ศ. 1980 เพื่อวัตถุประสงค์หลักเพื่อนำไปใช้งานแทนระบบบาร์โค้ด (Barcode)โดยจุดเด่นของ RFID ก็อยู่ตรงการอ่านข้อมูลจากแท็ก (Tag) ได้หลายๆ แท็กแบบไร้สัมผัสและสามารถอ่านค่าได้แม้ในสภาพที่ทัศนวิสัยไม่ดี ทนต่อความเปียกชื้น ทนต่อแรงสั่นสะเทือน การกระทบกระแทก และสามารถจะอ่านข้อมูลได้ด้วยความเร็วสูงโดยข้อมูลจะถูกเก็บไว้ในไมโครชิปที่อยู่ในแท็ก ในปัจจุบันได้มีการนำ RFID ไปประยุกต์ใช้งานในด้านอื่นๆ นอกเหนือจากนำมาใช้แทนระบบบาร์โค้ดแบบเดิม ใช้ในบัตรชนิดต่างๆ เช่น บัตรสำหรับผ่านเข้าออกห้องพัก บัตรที่จอดรถตามศูนย์การค้าต่างๆ อาจพบเห็นอยู่ในรูปของแท็กสินค้าซึ่งมีขนาดเล็กจนสามารถแทรกลงระหว่างชั้นของเนื้อกระดาษได้ หรือเป็นแคปซูลขนาดเล็กฝังเอาไว้ในตัวสัตว์เพื่อบันทึกประวัติต่างๆ เป็นต้น

5.2 ส่วนประกอบของระบบ RFID

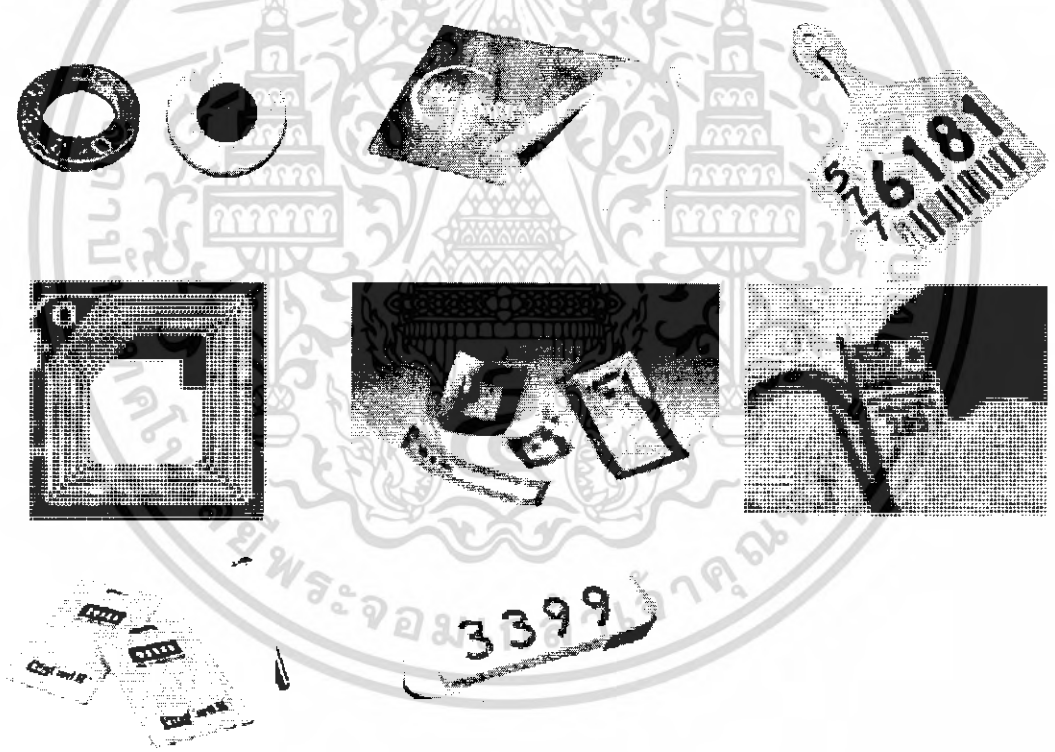
ในระบบ RFID จะมีองค์ประกอบหลักๆอยู่ 2 ส่วนด้วยกัน ส่วนแรกคือทรานสปอนเดอร์หรือแท็ก(Transponder/Tag) ที่ใช้ติดกับวัตถุต่างๆที่เราต้องการโดยแท็กที่ว่าจะบันทึกข้อมูลเกี่ยวกับวัตถุชิ้นนั้นๆเอาไว้ ส่วนที่สองก็คือเครื่องสำหรับอ่าน/เขียนข้อมูลภายในแท็ก (Interrogator/Reader) ด้วยคลื่นความถี่วิทยุ



รูปที่ 5.1 แสดงภาพรวมของระบบ RFID

5.2.1 แท็ก (Tag)

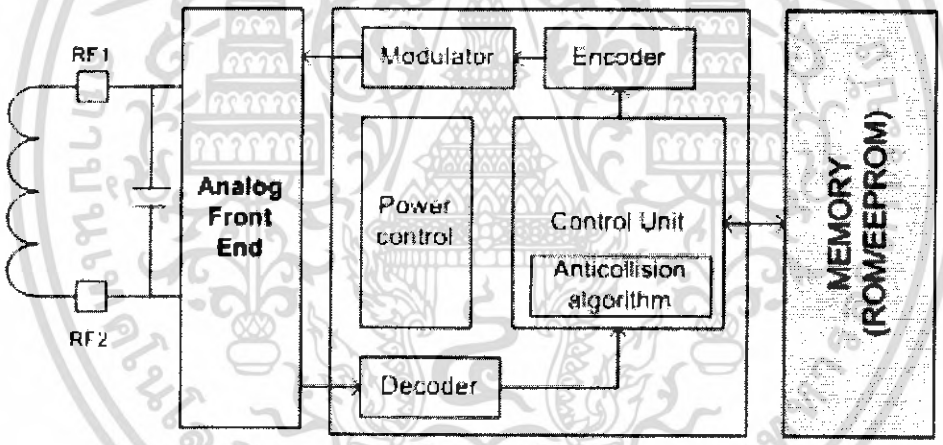
โครงสร้างภายในของแท็กจะประกอบด้วย 2 ส่วนใหญ่ๆ ได้แก่ ขดลวดขนาดเล็กซึ่งทำหน้าที่เป็นสายอากาศ (Antenna) สำหรับรับ-ส่งสัญญาณคลื่นความถี่วิทยุ และสร้างพลังงานป้อนให้ส่วนของไมโครชิพ (Microchip) ที่ทำหน้าที่เก็บข้อมูลของวัตถุ เช่น รหัสสินค้า โดยทั่วไปตัวแท็กอาจอยู่ในชนิดทั้งเป็น กระดาษ แผ่นฟิล์ม พลาสติก มีขนาดและรูปร่างต่างๆกันไป ทั้งนี้ขึ้นอยู่กับวัสดุที่จะนำเอาไปติด และมีหลายรูปแบบ เช่น ขนาดเท่าบัตรเครดิต เหรียญ กระดุม ฉลากสินค้า แคปซูล เป็นต้น (พิจารณารูปที่ 2) แต่โดยหลักการอาจแบ่งแท็กที่มีใช้งานกันอยู่นั้นจะมีอยู่ 2 ชนิดใหญ่ๆ โดยแต่ละชนิดก็จะมีความแตกต่างกันในแง่ของการใช้งาน ราคา โครงสร้างและหลักการทำงานอยู่ อธิบายแยกเป็นหัวข้อดังนี้



รูปที่5.2 แสดง RFID แท็กในรูปแบบต่างๆ

Passive RFID Tags

แท็กชนิดนี้ไม่ต้องอาศัยแหล่งจ่ายไฟภายนอกใดๆ เพราะภายในแท็กจะมีวงจรกำเนิดไฟฟ้าเหนี่ยวนำขนาดเล็กเป็นแหล่งจ่ายไฟในตัวอยู่ทำให้การอ่านข้อมูลทำได้ไม่ไกลมากนัก ระยะอ่านสูงสุดประมาณ 1 เมตรขึ้นอยู่กับความแรงของเครื่องส่งและคลื่นความถี่วิทยุที่ใช้โดยปกติแท็กชนิดนี้มักมีหน่วยความจำขนาดเล็ก โดยทั่วไปประมาณ 16 ถึง 1,024 ไบต์ มีขนาดเล็กและน้ำหนักเบา ราคาต่อหน่วยต่อไอซีของแท็กชนิดพาสซีฟที่มีการผลิตออกมา จะมีทั้งขนาดและรูปร่างเป็นได้ตั้งแต่แท่งหรือแผ่นขนาดเล็กจนแทบไม่สามารถมองเห็นได้ไปจนถึงขนาดใหญ่สะดุดตา ซึ่งต่างก็มีความเหมาะสมกับชนิดงานที่แตกต่างกัน โดยทั่วไปโครงสร้างภายในส่วนที่เป็นไอซีของแท็กนั้นก็จะประกอบด้วย 3 ส่วนหลักๆ ได้แก่ ส่วนของควบคุมการทำงานของภาครับส่งสัญญาณวิทยุ (Analog Front-End) ส่วนควบคุมภาคลอจิก (Digital Control Unit) ส่วนของหน่วยความจำ (Memory) ซึ่งอาจจะเป็นแบบ ROM หรือ EEPROM

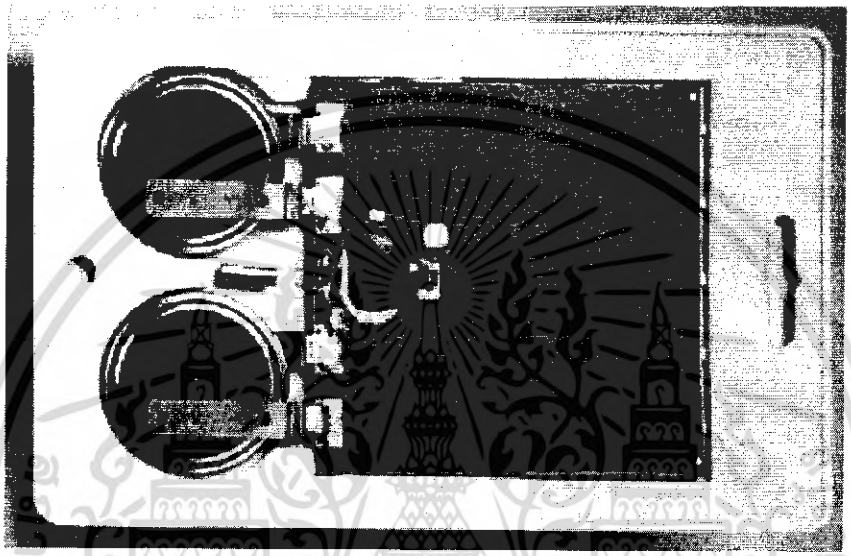


รูปที่ 5.3 แสดงบล็อกไดอะแกรมของ Passive Tag

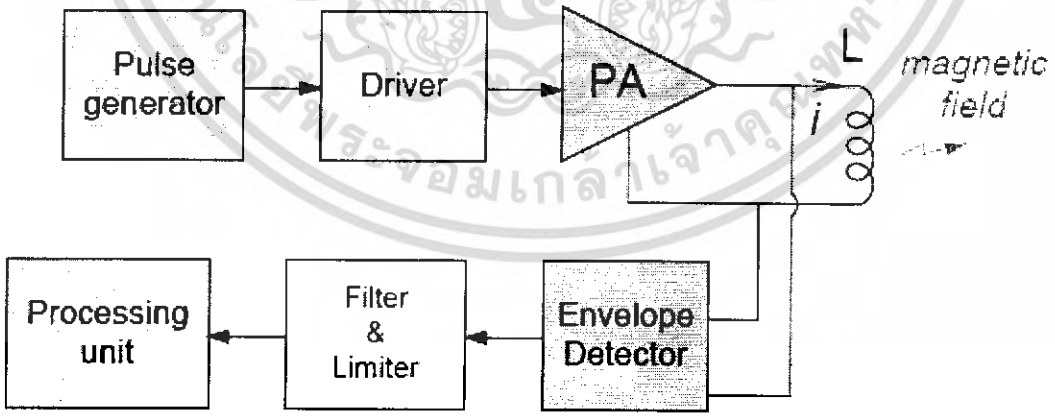
Active RFID Tags

แท็กชนิดนี้จะต้องอาศัยแหล่งจ่ายไฟจากแบตเตอรี่ภายนอก เพื่อจ่ายพลังงานให้กับวงจรภายในทำงานโดยแท็กแบบนี้สามารถมีหน่วยความจำภายในขนาดใหญ่ได้ถึง 1 เมกะไบต์ และสามารถอ่านได้ในระยะไกลสูงสุดประมาณ 10 เมตรแม้ว่าแท็กชนิดนี้จะมีข้อดีอยู่หลายข้อแต่ก็มีข้อเสียอยู่ด้วยเหมือนกัน เช่น มีราคาต่อหน่วยแพงมีขนาดค่อนข้างใหญ่ และมีระยะเวลาในการทำงานที่จำกัดนอกจากการแบ่งจากชนิดที่ว่ามาแล้ว แท็กก็ยังถูกแบ่งประเภทจากรูปแบบในการใช้

งานได้เป็น 3 แบบคือ แบบที่สามารถถูกอ่านและเขียนข้อมูลได้อย่างอิสระ (Read–Write), แบบเขียนได้เพียงครั้งเดียวเท่านั้นแต่อ่านได้อย่างอิสระ (Write-Once Read-Many หรือ WORM) และแบบอ่านได้เพียงอย่างเดียว (Read-Only) อย่างไรก็ตามเนื่องจากในการใช้งานจะนิยมใช้แท็กชนิดพาสซีฟมากกว่าในที่นี้จึงจะขอกล่าวถึงเฉพาะแท็กชนิดนี้เป็นหลัก



รูปที่ 5.4 แสดงรูปตัวอย่าง Active Tag ที่มีแบตเตอรี่ Lithium 2 ก้อนอยู่ภายนอก



รูปที่ 5.5 แสดงโครงสร้างภายในเครื่องอ่าน

5.2.2 เครื่องอ่าน (Reader)

โดยหน้าที่ของเครื่องอ่านก็คือ การเชื่อมต่อเพื่อเขียนหรืออ่านข้อมูลลงในแท็กด้วยสัญญาณความถี่วิทยุภายในเครื่องอ่านจะประกอบด้วย เสาอากาศที่ทำจากขดลวดทองแดง เพื่อใช้รับ-ส่งสัญญาณ ภาครับและภาคส่งสัญญาณวิทยุ และวงจรควบคุมการอ่าน-เขียนข้อมูล จำพวกไมโครคอนโทรลเลอร์ และส่วนของการติดต่อกับคอมพิวเตอร์ ดังรูปที่ 5.5

โดยทั่วไปเครื่องอ่านจะประกอบด้วยส่วนประกอบหลักดังนี้

- ภาครับและส่งสัญญาณวิทยุ
- ภาคสร้างสัญญาณพาหะ
- ขดลวดที่ทำหน้าที่เป็นสายอากาศ
- วงจรจูนสัญญาณ
- หน่วยประมวลผลข้อมูล และภาคติดต่อกับคอมพิวเตอร์

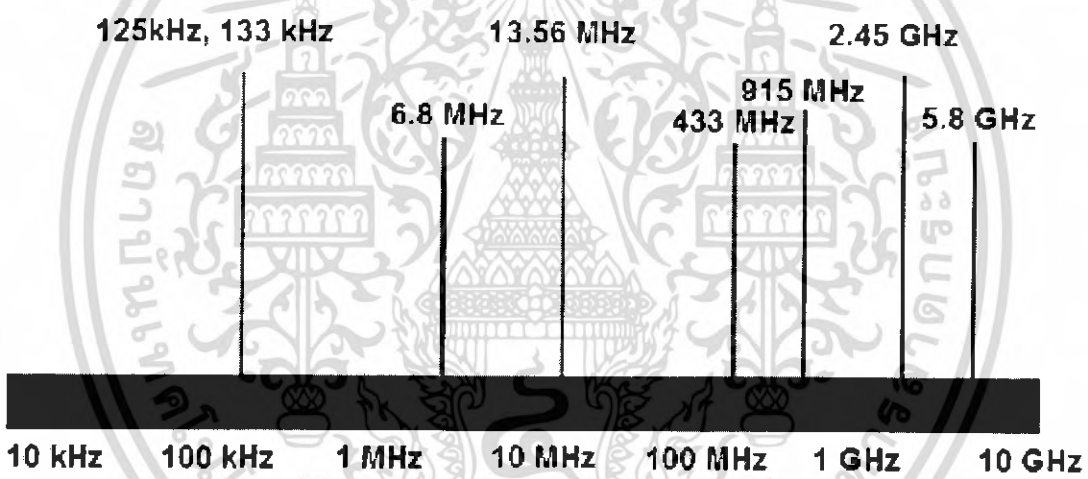
โดยทั่วไปหน่วยประมวลข้อมูลที่อยู่ภายในเครื่องอ่านมักใช้เป็นไมโครคอนโทรลเลอร์ ซึ่งอัลกอริทึมที่อยู่ภายในโปรแกรม จะทำหน้าที่ถอดรหัสข้อมูล (Decoding) ที่ได้รับ และทำหน้าที่ติดต่อกับคอมพิวเตอร์ โดยลักษณะ ขนาด และรูปร่างของเครื่องอ่านจะแตกต่างกันไปตามประเภทของการใช้งาน เช่น แบบมือถือขนาดเล็กหรือ ติดผนัง จนไปถึงขนาดใหญ่เท่าประตู (Gate size) เป็นต้น ดังแสดงในรูปที่ 5.6

รูปที่ 5.6 แสดงรูปตัวอย่างเครื่องอ่านแบบต่างๆ

5.3 คลื่นพาหะในระบบ RFID

ในปัจจุบันคลื่นพาหะที่ใช้งานกันในระบบ RFID จะอยู่ในย่านความถี่ ISM (Industrial-Scientific-Medical) ซึ่งเป็นย่านความถี่ที่กำหนดในการใช้งานในเชิงอุตสาหกรรม วิทยาศาสตร์ และการแพทย์ สามารถใช้งานได้โดยไม่ตรงกับย่านความถี่ที่ใช้งานในการสื่อสารโดยทั่วไป โดยมี 3 ย่านความถี่ใช้งาน คือ สำหรับคลื่นพาหะที่ใช้กันในระบบ RFID อาจแบ่งออกได้เป็น 3 ย่านหลัก ได้แก่

- ย่านความถี่ต่ำ (Low Frequency : LH) ต่ำกว่า 150 kHz
- ย่านความถี่สูง (High Frequency : HF) 13.56 MHz
- ย่านความถี่สูงยิ่ง (Ultra High Frequency : UHF) 433/868/915 MHz



รูปที่ 5.7 แสดงความถี่ย่านที่ระบบ RFID ถูกใช้งาน

ในแง่การใช้งาน 2 ย่าน ความถี่แรกจะเหมาะสำหรับใช้กับงานที่มีระยะการสื่อสารข้อมูลในระยะใกล้ (LHระยะอ่านประมาณ 10-20 เซนติเมตร และ HF ระยะอ่านประมาณ 1 เมตร) เช่น การตรวจสอบการผ่านเข้าออกพื้นที่ การตรวจหาและเก็บประวัติในสัตว์ ส่วนย่านความถี่สูงยิ่งจะถูกใช้กับงานที่มีระยะการสื่อสารข้อมูลในระยะไกล(UHF ระยะอ่านประมาณ 1-10 เมตร) เช่น ระบบเก็บค่าบริการทางด่วน และในปัจจุบันระบบ RFID กำลังถูกวิจัยและพัฒนาในย่านความถี่ไมโครเวฟที่ความถี่ 2.4 GHz และความถี่ 5.8 GHz เพื่อใช้งานที่ต้องการระยะอ่านที่ไกลกว่า 10

เมตร เป็นต้นในแง่ของราคาและความเร็วในการสื่อสารข้อมูล เมื่อเทียบกันแล้ว RFID ซึ่งใช้คลื่นพาหะย่านความถี่สูง เป็นระบบที่มีความเร็วในการส่งข้อมูลสูงสุดและมีราคาแพงที่สุดด้วยเช่นกัน ส่วน RFID ที่ใช้คลื่นพาหะในอีก 2 ย่านความถี่จะมีระดับราคาและความเร็วลดหลั่นกันไป

5.4 ตัวอย่างการใช้งาน RFID

ในปัจจุบันการนำระบบ RFID มาประยุกต์ใช้งานหลากหลายประเภท เช่น

• ทดแทนระบบบาร์โค้ด (Barcode) รุ่นเก่า

• Access Control / Personal Identification หรือการเข้า-ออกอาคาร แทนการใช้บัตรแม่เหล็กรูดเป็นเดิมเนื่องบัตรแม่เหล็กรูดมากๆก็จะเสื่อม แต่บัตรแบบ RFID (Proximity Card) ใช้เพียงแตะหรือแสดงผ่านหน้าเครื่องอ่านเท่านั้น รวมทั้งยังสามารถใช้กับการเช็คเวลาเข้า-ออกงานของพนักงาน

• ห่วงโซ่อุปทาน และระบบลอจิสติกส์ ภาพที่จะเห็นในโรงงานอนาคตคือ สามารถติด Tag ไว้กับชิ้นงานเมื่อชิ้นงานผ่านสายพานขนสินค้าในโรงงาน แต่ละแผนกจะรู้ว่าต้องทำอย่างไร ติดอะไรบ้าง และต้องส่งไปที่ไหนต่อ รวมถึงการจัดการสินค้าในคลังสินค้า ว่ารับสินค้ามาเมื่อใด จะต้องเก็บไว้ที่ไหน จะส่งไปที่ไหนยังไง ใครจะมารับ ส่วนภาพที่ผู้บริโภคจะเห็นคือ การซื้อสินค้าในซูเปอร์มาร์เก็ต เวลาซื้อก็หยิบใส่ตะกร้าคิดเงินผ่านเครื่องอ่าน RFID ครั้งเดียวคิดเงินได้ทันที ไม่ต้องหยิบมายิงบาร์โค้ดทีละชิ้นให้เสียเวลา และเตือนผู้ซื้อได้หากสินค้าที่ซื้อหมดอายุ

• ระบบ Animal Tracking มาใช้เหมาะกับเกษตรกรไทย ในการพัฒนาด้านปศุสัตว์ให้เป็นระบบฟาร์ม ออโตเมชัน ด้วยชิป RFID ติดตัวสัตว์เลี้ยง ทำให้สามารถทราบเจ้าของ ตรวจสอบสายพันธ์ การให้อาหารและการควบคุมโรคติดต่อในสัตว์ รวมถึงการสร้าง Food Traceability สำหรับใช้สู้กับข้อกีดกันทางการค้าของสหรัฐอเมริกา และกลุ่มสหภาพยุโรป ที่อยู่ระหว่างตัดสินใจว่าผู้ส่งออกสินค้าเนื้อสัตว์ชำแหละ

• ระบบตั๋วอิเล็กทรอนิกส์ (e-ticket) เช่น บัตรทางด่วน บัตรรถไฟฟ้าใต้ดิน

• ระบบหนังสือเดินทางอิเล็กทรอนิกส์ (e-passport) ที่ทางประเทศสหรัฐกำลังกำหนดมาตรฐานการเข้าออกของประเทศของเค้า เพื่อป้องกันผู้ก่อการร้ายรวมไปถึง E-citizen ด้วย

• ระบบกุญแจอิเล็กทรอนิกส์ (Immobilizer) ในรถยนต์ ป้องกันกุญแจผิดในการขโมยรถยนต์ (Smart Key entry) พวก Keyless ในรถยนต์ราคาแพงบางรุ่นก็เริ่มนำมาใช้งานแล้ว

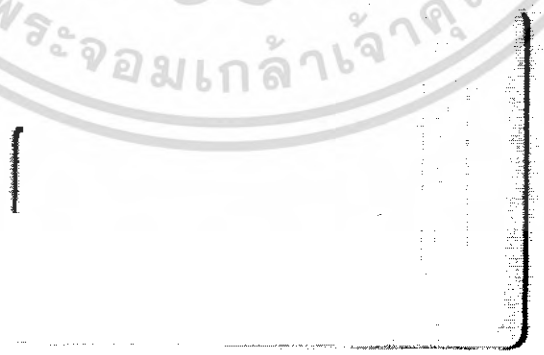
• ระบบห้องสมุดดิจิตอล ในการยืมคืนอัตโนมัติ ทำให้ผู้ใช้บริการได้รวดเร็วและสะดวกสบายยิ่งขึ้น

รูปที่ 5.8 แสดงภาพการประยุกต์ใช้งาน RFID ในงานต่างๆ

5.5 เครื่องอ่านบัตรป้ายระบุข้อมูล GP-8

GP-8 เป็นวงจรสำเร็จรูปที่มีราคาต่ำ คุณภาพสูง สำหรับการใช้กับสายอากาศภายนอกแบบง่ายๆ ลักษณะเด่นคือ มีระยะการอ่านปานกลาง และมีขนาดเล็ก GP-8 จะมีระยะอ่านบัตรได้ดีที่แรงดัน 5 โวลต์ ทำให้เหมาะสำหรับการประยุกต์ใช้งานที่มีความหลากหลาย หรือระบบควบคุมที่จำเพาะเจาะจง สามารถจะกำหนดรูปแบบของเอาท์พุทได้หลายแบบ ทั้งแบบ Wiegand, Magstripe และ Serial ASCII(RS232) ทำให้ง่ายต่อการใช้งาน

รูปที่ 5.9 เครื่องอ่าน GP-8



รูปที่ 5.10 บัตรป้ายระบุข้อมูล RFID แบบอ่านอย่างเดียว

ทฤษฎี

เครื่องอ่านทำงานด้วยสนามแม่เหล็กเหนี่ยวนำ ความถี่ 125 KHz เมื่อเลื่อนบัตรหรือแท็ก (Tag/Transponder)เข้าใกล้เครื่องอ่าน บัตรจะดึงกำลังงานจากสนามแม่เหล็กให้มีค่าเพียงพอที่จะทำให้วงจรขนาดเล็กที่อยู่ภายในบัตรสามารถทำงานได้ ข้อมูลจะถูกส่งผ่านจากบัตรโดยพาหะของการเปลี่ยนขนาดของคลื่น(Amplitude modulation) ข้อมูลรหัสชี้เฉพาะของบัตรจะถูกโปรแกรมลงในหน่วยความจำภายในเครื่องอ่านบัตร การเปลี่ยนของสนามแม่เหล็กจะทำให้เครื่องอ่านสามารถตรวจพบบัตร และทำการแปลงกลับข้อมูลจากบัตรให้กลับมาเป็นข้อมูลเดิมได้

รายละเอียด

กำลังงานที่ต้องการ	: 5-13.5 โวลต์ แรงดันคงที่ DC กระแส 55 มิลลิแอมป์ ที่แหล่งจ่ายไฟ 12 โวลต์
การเชื่อมต่อ	: Wiegand, Magstrape, 9.6 K Baud Serial ASCII(RS232)
ระยะการอ่าน	: ระยะ 5.0 เซนติเมตร ที่แรงดัน 12 โวลต์ กับบัตรมาตรฐาน ISO
ย่านความถี่	: 125 KHz มาตรฐาน หรือ 134.2 KHz แบบสั่งทำพิเศษ
บัตรป้ายระบุข้อมูล	: แบบอ่านได้อย่างเดียว
ตัวแสดงแสง / เสียง	: LED และ Buzzer
ขนาด	: 41 x 24 x 10 มิลลิเมตร
น้ำหนัก	: น้อยกว่า 50 กรัม

การกำหนดขา

ขา 1	แรงดัน 0 โวลต์ (กราวนด์)
ขา 2	แรงดัน 5 – 13.5 โวลต์
ขา 3	โปรแกรมอินพุท
ขา 4	แสดงบัตรเอาท์พุท
ขา 5	ข้อมูลเอาท์พุท RS232, Magstripe และ Wiegand0
ขา 6	สัญญาณนาฬิกา Magstripe และ Wiegand1
ขา 7	ควบคุมสัญญาณเสียงภายนอก
ขา 8	ขับ LED
ขา 9	ต่อทรานซิสเตอร์ขับ Buzzer
ขา 10	กราวนด์เสาอากาศภายนอก
ขา 11	ขับเสาอากาศภายนอก

การโปรแกรม

รูปแบบของเอาท์พุตสามารถเลือกได้ 3 แบบ คือ Wiegand, Magnetic Emulation และ Serial ASCII(RS232)

Wiegand		**Magstripe**	
Pin 1	Ground 0V	Pin 1	Ground 0V
Pin 2	Power +V	Pin 2	Power +V
Pin 5	Data0	Pin 6	Clock (Strobe)
Pin 6	Data1	Pin 5	Data
Pin 3	Connect to Pin 6	Pin 4	Card Present
Pin 4	No Connection	Pin 3	Connect to Pin 4

Serial ASCII (RS232)

Pin 1	Ground 0V
Pin 2	Power +V
Pin 5	Data
Pin 3	No Connection
Pin 4	No Connection
Pin 6	No Connection

โครงสร้างข้อมูล (ASCII)

อัตราบอด : 9600,N,8,1

STX (02 HEX)	DATA (10 HEX)	CR	LF	ETX (03 HEX)

เริ่มด้วยรหัสควบคุมที่ระบุโดยโรงงาน กำหนดเป็น 'STX'(Start of text)(02 HEX) ตามด้วยข้อมูลตัวอักษรฐาน16 อีก 10 ตัว CR(Carriage return)(0D HEX) \ LF(Line feed)(0A HEX) เป็น

รหัสควบคุมทำให้จอภาพรับตัวอักษรย้ายไปอยู่ทางซ้าย และขึ้นบรรทัดใหม่หลังจากข้อมูลถูกส่งแล้ว 'ETX'(End of text)(03 HEX) เป็นตัวรหัสควบคุมระบุการสิ้นสุดการส่งข้อมูลตัวอักษร

โครงสร้างข้อมูล(Magstripe)

ความเร็ว : Simulated to 40 IPS (Inch per Second)

10 LEADING ZERO	SS	DATA (14 DIGITS)	ES	LRC	10 TRAILING ZEROS

ค่าศูนย์ 10 ตัวนำหน้าเป็นการเตรียมสำหรับการรับข้อมูล 14 ตัวเลข SS (Start Sentinel ประกอบด้วย 11010 ES (End Sentinel) ประกอบด้วย 11111 LRC (Longitude Redundancy Check character) เป็นตัวตรวจสอบตัวอักษร ท้ายสุดเป็น ค่าศูนย์ปิดท้าย 10 ตัว ข้อมูลเลขฐานสิบหกจากบัตรจะถูกแปลงก่อนการส่งออกไป เช่น

Hexadecimal = 0411115EA6

$= (6*16^0+10*16^1+14*16^2+5*16^3+1*16^4+1*16^5+1*16^6+1*16^7+4*16^8)$

= 00017466220198 => Denary string

โครงสร้างข้อมูล (Wiegand Format-26 Bit)

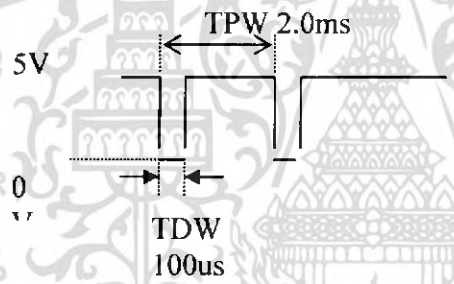
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P	S	S	S	S	S	S	S	S	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	P
P	E	E	E	E	E	E	E	E	E	E	E	E													
													O	O	O	O	O	O	O	O	O	O	O	O	P
SUMMED FOR EVEN PARITY (E)													SUMMED FOR ODD PARITY (O)												

รูปที่ 5.11 โครงสร้างข้อมูล(Wiegand Format-26 Bit)

P	Parity (Even or ODD) Start Bit and Stop Bit
S	Site Bits from Card or Reader
C	Card Number Bits from Card
SYSDSSW1-W26	Side bits from Card (24 bits Card Data)
MSB	Normal 01
LSB	Normal 24

Data Timing Specification

Pulse Interval(TPW)	2.0mS +/- 3%
Pulse Width (TDW)	100uS +/- 3%

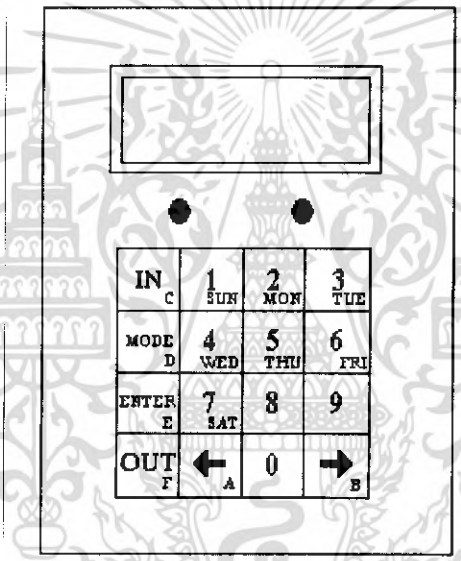


รูปที่ 5.12 แสดงลักษณะเวลาในการส่งข้อมูลแบบ Wiegand

บทที่ 6
การออกแบบเครื่องบันทึกเวลาโดยใช้ RFID

6.1 การออกแบบและเลือกอุปกรณ์

การออกแบบส่วนของตัวเครื่องที่ดี จะทำให้ผู้ใช้งานสามารถใช้ได้อย่างสะดวกไม่ยุ่งยากนัก ดังนั้นในการออกแบบเครื่องบันทึกเวลาโดยใช้ RFID จึงได้เลือกใช้อุปกรณ์ต่างๆ ซึ่งสามารถทำให้เกิดความสะดวกกับผู้ใช้งาน และแสดงผลการใช้งานให้ได้รับทราบ โดยเลือกใช้อุปกรณ์ตามลักษณะการใช้งานดังนี้



รูปที่ 6.1 การออกแบบตัวเครื่อง

6.1.1 ตัวประมวลผล

เลือกใช้ไมโครคอนโทรเลอร์ตระกูล MCS-51 เบอร์ AT89C52 เนื่องจากมีฟังก์ชันการทำงานเพียงพอกับความต้องการทั้งจำนวนพอร์ต 3 พอร์ต การติดต่อแบบอนุกรม, มีไทเมอร์ในตัว รวมไปถึงการมีขนาดของหน่วยความจำโปรแกรมแบบเฟลชขนาด 8 กิโลไบต์ และหน่วยความจำข้อมูลแรม 256 ไบต์ การใช้งานขึ้นอยู่กับการเขียนโปรแกรมเพื่อประมวลผลและสั่งการทำงานโดยขาของพอร์ตต่างๆ และเลือกใช้ไมโครคอนโทรเลอร์เบอร์ AT89C2051 สำหรับใช้ถ่ายโอนข้อมูลระหว่าง EEPROM กับ คอมพิวเตอร์ เนื่องจากมีความต้องการใช้งานพอร์ตเพียงเชื่อมต่อกับ EEPROM,LED แสดงผลและเชื่อมต่อผ่านคอมพิวเตอร์เท่านั้นจึงไม่จำเป็นต้องใช้งานพอร์ตมากนัก

6.1.2 การรับค่ารหัสบัตรจากผู้ใช้งาน

เลือกใช้ชุดเครื่องอ่านค่าจากบัตร RFID เนื่องจากความสะดวกในการใช้งาน ใช้แค่การสัมผัสบัตรหรือเลื่อนบัตรเข้าใกล้ตัวเครื่องเพื่อให้เครื่องรับรหัสบัตรไปตรวจสอบและลงเวลา ทำให้ไม่เสียเวลาเหมือนกับการตอกบัตรลงเวลาทำงาน

6.1.3 การแสดงผล

เลือกใช้ LCD ในการแสดงผล เนื่องจากสามารถแสดงข้อความเพื่อแสดงให้ผู้ใช้งานทราบถึงการทำงานในขณะนั้นๆ ได้เป็นอย่างดีและมีการแสดงผลด้วย LED เพื่อแสดงสถานะการทำงานและการอนุญาตให้สามารถใช้งานได้ อีกทั้งยังมีการใช้ Buzzer เพื่อแสดงผลในรูปแบบเสียงเพื่อให้ผู้ใช้ทราบการทำงานอีกด้วย

6.1.4 การรับค่าจากปุ่มกด

เลือกใช้ Keypad ขนาด 4x4 เนื่องจากนอกจากจะสามารถกำหนดจัดตัวเลข 0-9 ได้แล้วยังสามารถกำหนดปุ่มอื่นๆเพิ่ม เพื่อใช้งานตามต้องการได้ โดยขึ้นอยู่กับการเขียนโปรแกรมเพื่อเลือกใช้งานปุ่มต่างๆ

6.1.5 การบันทึกข้อมูล

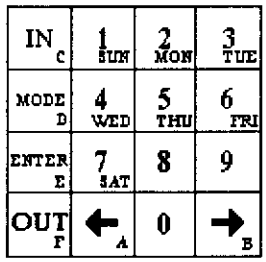
เลือกใช้ EEPROM ชนิดที่ใช้ระบบบัส I^2C เนื่องจาก EEPROM สามารถจะเขียนและอ่านข้อมูลได้ สามารถเก็บข้อมูลไว้ได้แม้ไม่มีไฟเลี้ยง อีกทั้งแบบที่เป็นระบบบัส I^2C จะใช้ร่วมกับขาเพียง 2 ขา คือ SDA และ SCL ทำให้ประหยัดขาไมโครคอนโทรเลอร์ และสามารถต่ออุปกรณ์บัส I^2C พ่วงขนานได้

6.1.6 การใช้งานด้านเวลา

เลือกใช้ RTC เบอร์ DS1307 เนื่องจากใช้ระบบบัส I^2C เช่นเดียวกับ EEPROM นำมาต่อพ่วงขนานบนบัส I^2C ได้เลย สามารถใช้งานด้านเวลาได้อย่างแม่นยำ โดยจะใช้ถ่าน BACK UP สำรองไฟสำหรับให้ตัว IC นับเวลาอยู่ตลอด ทำให้เวลาไม่ผิดเพี้ยน

6.2 การออกแบบปุ่มกด

เนื่องจากการบันทึกการลงเวลาทำงานมีความจำเป็นที่จะต้องเลือกได้ว่า ต้องการบันทึกเวลาเข้า หรือ บันทึกเวลาออก รวมทั้งสามารถปรับตั้งค่าต่าง ๆ ของการทำงานเช่น วันที่ และเวลาได้ด้วย ดังนั้นจึงได้ออกแบบปุ่มกดให้ได้ตรงตามความต้องการดังนี้



รูปที่ 6.2 แสดงปุ่มของ Keypad ที่นำมาใช้งาน

ปุ่ม IN

ใช้สำหรับกดเลือกเพื่อการบันทึกเวลาเข้า และรอการสัมผัสบัตรประจำตัวของผู้ใช้ หากมีการสัมผัสบัตรจะตรวจสอบรหัสบัตรกับข้อมูลรหัสที่เก็บไว้ หากถูกต้องจะทำการบันทึกเวลาเข้า ถ้าไม่มีการสัมผัสบัตรภายในเวลาที่กำหนด จะกลับสู่หน้าหลัก

ปุ่ม MODE

ใช้สำหรับกดเลือกโหมดการทำงานของเครื่อง ซึ่งจะตรวจสอบการสัมผัสบัตรจากผู้ดูแลเครื่อง(Master) เท่านั้น ผู้ใช้งานทั่วไปจะไม่สามารถเข้าไปใช้งานโหมดนี้ได้ เมื่อเข้าโหมดนี้แล้วจะมีโหมดย่อยให้เลือก คือ SET DATE, SET TIME, ADD USER,DELETE USER, PASSWORD IN และ CAPACITY

ปุ่ม ENTER

ใช้สำหรับกดเลือก การรับค่ารหัสผ่าน(Password IN) โดยไม่ต้องใช้บัตร จะต้องใส่รหัสผ่านให้ถูกต้องจึงจะสามารถผ่านได้ รหัสผ่านจะเป็นตัวเลข 0 - 9 และเครื่องหมาย ←, → ซึ่งจะกำหนดไว้ก่อนแล้ว โดยผู้ดูแลเครื่อง

ปุ่ม OUT

ใช้สำหรับกดเลือก เพื่อการบันทึกเวลาออก และรอการสัมผัสบัตรประจำตัวของผู้ใช้ หากมีการสัมผัสบัตรจะตรวจสอบรหัสบัตรกับข้อมูลรหัสที่เก็บไว้ หากถูกต้องจะทำการบันทึกเวลาออก ถ้าไม่มีการสัมผัสบัตรภายในเวลาที่กำหนด จะกลับสู่หน้าหลัก

ปุ่ม ตัวเลข 0 – 9 และ เครื่องหมาย ← , →

ใช้สำหรับกดเพื่อใส่ค่ารหัสผ่าน(Password IN) เพื่อใช้ตรวจสอบรหัสที่กด กับรหัสผ่านที่ผู้ดูแลเครื่องกำหนดไว้ และใช้เลื่อนตำแหน่งไปทางด้านซ้ายหรือขวา

ปุ่มตัวอักษรพิเศษ A-F , SUN-SAT

ใช้สำหรับการปรับตั้งค่าวันและใส่รหัสในโหมดการลบรหัส

6.3 โหมดการทำงาน

โหมดการทำงานของเครื่องบันทึกเวลาโดยใช้ RFID แบ่งเป็นส่วนใหญ่ๆได้ 4 ส่วน ได้แก่

1. การบันทึกเวลาเข้า (IN)
2. การบันทึกเวลาออก (OUT)
3. การกดรหัสเพื่อเปิดประตู (ENTER)
4. การปรับตั้งค่าการใช้งานของเครื่อง (MODE)

โดยในส่วนของการบันทึกเวลาเข้าและออก รวมทั้งการกดรหัสเพื่อเปิดประตู จะเป็นส่วนที่ผู้ใช้งานทั่วไปสามารถใช้งานได้ แต่ในส่วนของการปรับแต่งค่าการใช้งานของเครื่องบันทึกเวลาโดยใช้ RFID จะสงวนไว้สำหรับผู้ดูแลเครื่องเท่านั้น ผู้ใช้งานทั่วไปจะไม่สามารถเข้าไปใช้งานในส่วนนี้ได้โดยในส่วนของการปรับตั้งค่าหรือการกดปุ่ม MODE จะมีโหมดย่อยดังนี้

1. Set Date : ใช้ปรับการตั้งค่า วัน, วันที่, เดือน และ ปี ของเครื่อง
2. Set Time : ใช้ปรับการตั้งค่าเวลา ทั้งหลักชั่วโมง, นาที และ วินาที
3. Add User : ใช้เพิ่มรหัสผู้ใช้งานลงในหน่วยความจำ Database
4. Delete User : ใช้ลบรหัสผู้ใช้งานออกจากหน่วยความจำ Database
5. Change Password : ใช้เปลี่ยนการตั้งค่ารหัสเพื่อเปิดประตู
6. Capacity : ใช้ตรวจสอบพื้นที่หน่วยความจำของ Database และ Memory

6.4 การออกแบบการบันทึกข้อมูลใน EEPROM

เนื่องจากมีการใช้งานเครื่องบันทึกเวลาด้วย RFID ในด้านของการใช้งานเป็นเครื่องมือรักษาความปลอดภัยผู้ที่มีบัตรผ่านหรือทราบรหัสผ่านเท่านั้นที่สามารถปลดล็อกประตูได้ จึงจำเป็นต้องมีการบันทึกรหัสบัตรเก็บไว้ในฐานข้อมูลก่อนล่วงหน้า และนำมาเรียกใช้งานเปรียบเทียบกับรหัสจากบัตร อีกทั้งยังมีการใช้งานการบันทึก วันและเวลา เข้า-ออกของผู้ใช้งานเพื่อจะนำไปใช้งานในการเก็บข้อมูลโดยใช้งานร่วมกับคอมพิวเตอร์ ซึ่งขนาดของ EEPROM นั้นมีขนาดจำกัด จึงต้องมีการออกแบบการบันทึกข้อมูลใน EEPROM เพื่อให้สามารถบันทึกข้อมูลได้จำนวนมากที่สุดในขนาดความจุที่มีอยู่

6.4.1 การบันทึกข้อมูลใน EEPROM Database

ในส่วนของ Database จะมีการเก็บข้อมูลสำคัญๆ หลักๆ 3 อย่างคือ ข้อมูลรหัสผ่าน (Password IN) , ข้อมูลรหัสผู้ดูแลเครื่อง (Master ID), และข้อมูลรหัสผู้ใช้งาน (User ID) แต่ในการอ้างถึงข้อมูลใน EEPROM จำเป็นต้องอ้างโดยใช้แอดเดรส ดั้งนั้นจึงเพิ่มส่วนสำหรับเก็บข้อมูลแอดเดรสสุดท้าย (Last Address) ของการบันทึกข้อมูลไว้สำหรับใช้อ้างในการบันทึกข้อมูลครั้งต่อไป

Last Address : มีขนาด 2 ไบต์ ใช้กับแอดเดรสไบต์สูง (Address High Byte) และ แอดเดรสไบต์ต่ำ (Address Low Byte)

Password ID : มีขนาด 4 ไบต์ ใช้กับรหัสผ่านปลดล็อกประตู (8 ตัว)

Master ID : มีขนาด 5 ไบต์ ใช้เก็บรหัสของผู้ดูแลเครื่อง (10 ตัว)

User ID : มีขนาด 5 ไบต์ ใช้เก็บรหัสของผู้ใช้งาน (10 ตัว)

0000	0002	0006	000B	0010	0015		1FFF
Last Address	Password IN	Master ID	ID 1	ID 2	-------	------	

6.4.2 การบันทึกข้อมูลใน EEPROM Memory

ในส่วนของ Memory จะมีส่วนที่บันทึกเหมือนกับ Database คือการเก็บข้อมูลแอดเดรสสุดท้าย (Last Address) สำหรับข้อมูลที่จะบันทึกใน Memory นั้นได้แก่ วันที่/เดือน/ปี (DATE), รหัสของผู้ใช้งาน (User ID), เวลาเข้า (Time IN) และเวลาออก (Time OUT) โดยการบันทึกแต่ละครั้งจะทำการบันทึกข้อมูลเป็นชุดๆ

Last Address : มีขนาด 2 ไบต์ ใช้กับแอดเดรสไบต์สูง (Address High Byte) และ แอดเดรสไบต์ต่ำ (Address Low Byte)

DATE : มีขนาด 3 ไบต์ ใช้เก็บข้อมูล วันที่/เดือน/ปี

User ID : มีขนาด 5 ไบต์ ใช้เก็บรหัสของผู้ใช้งาน

Time IN : มีขนาด 2 ไบต์ ใช้เก็บข้อมูลเวลาเข้า เป็น ชั่วโมง:นาที

Time OUT : มีขนาด 2 ไบต์ ใช้เก็บข้อมูลเวลาออก เป็น ชั่วโมง:นาที

0000	0002	0005	000A	000C	000E	FFFF
Last Address	DATE	User ID	Time IN	Time OUT	----------	

6.5 การเลือกขนาดของ EEPROM

เนื่องจาก EEPROM มีหลายขนาดให้เลือกใช้ จึงจำเป็นต้องมีหลักเกณฑ์ในการเลือกมาใช้งานเพื่อให้เกิดความเหมาะสมกับขนาดของข้อมูลที่ต้องการบันทึกโดย EEPROM ที่มีในปัจจุบันมีขนาดดังนี้

512 Kbits มีแอดเดรส 0000-FFFF = 65,536 bytes

256 Kbits มีแอดเดรส 0000-7FFF = 32,768 bytes

64 Kbits มีแอดเดรส 0000-1FFF = 8,192 bytes

32 Kbits มีแอดเดรส 0000-0FFF = 4,096 bytes

16 Kbits มีแอดเดรส 0000-07FF = 2,048 bytes

จากขนาดของ EEPROM หลายๆ ขนาด พิจารณาจากจำนวนไบต์ที่สามารถใช้งานได้ ทำให้เลือกใช้ EEPROM Database ขนาด 512 Kbits และ EEPROM Memory ขนาด 64 Kbits โดยมีการคำนวณดังนี้

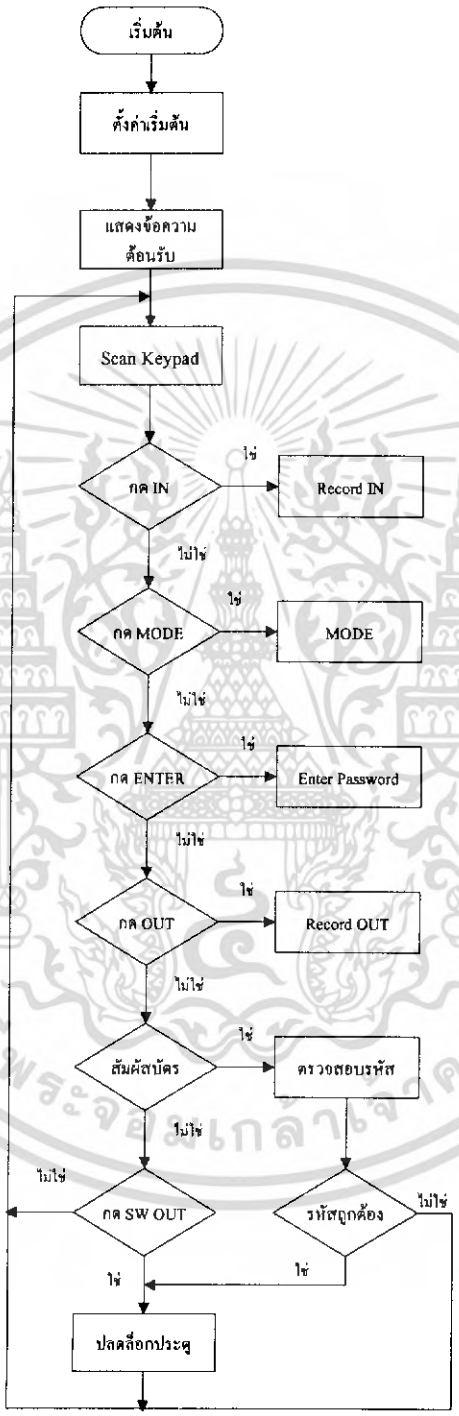
EEPROM Database

- เลือกใช้ 64 Kbits = 8,192 bytes
- ข้อมูลที่บันทึก ได้แก่ 1. Last Address (2 bytes)
  2. Password ID (4 bytes)
  3. Master ID (5 bytes)
  4. User ID (5 bytes)
- ดังนั้นมีพื้นที่เหลือสำหรับการบันทึก User ID = 8,192 - (2+4+5) = 8181 bytes
- สามารถบันทึก User ID ได้ = 8181/5 = 1636.2 หรือ 1636 ID

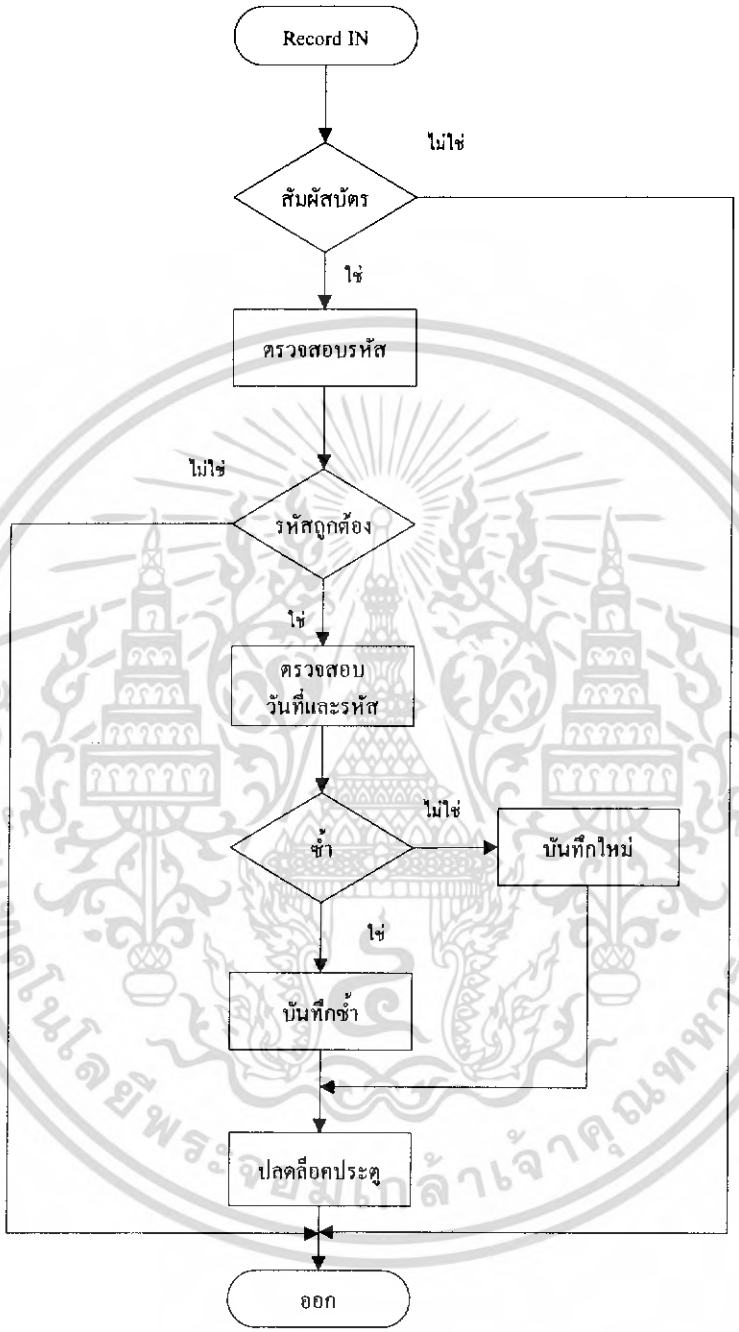
EEPROM Memory

- เลือกใช้ 512 Kbits = 65,536 bytes
- ข้อมูลที่บันทึก ได้แก่ 1. Last Address (2 bytes)
  2. DATE (3 bytes)
  3. User ID (5 bytes)
  4. Time IN (2 bytes)
  5. Time OUT (2 bytes)
- ข้อมูลแต่ละชุดมีขนาด = (3+5+2+2) = 12 bytes
  ดังนั้นมีพื้นที่เหลือสำหรับการบันทึกข้อมูลแต่ละชุด = 65,536 – 2 = 65,534 bytes
- สามารถบันทึกบันทึกชุดข้อมูลได้ = 65,534/12 = 5,461 ชุดข้อมูล

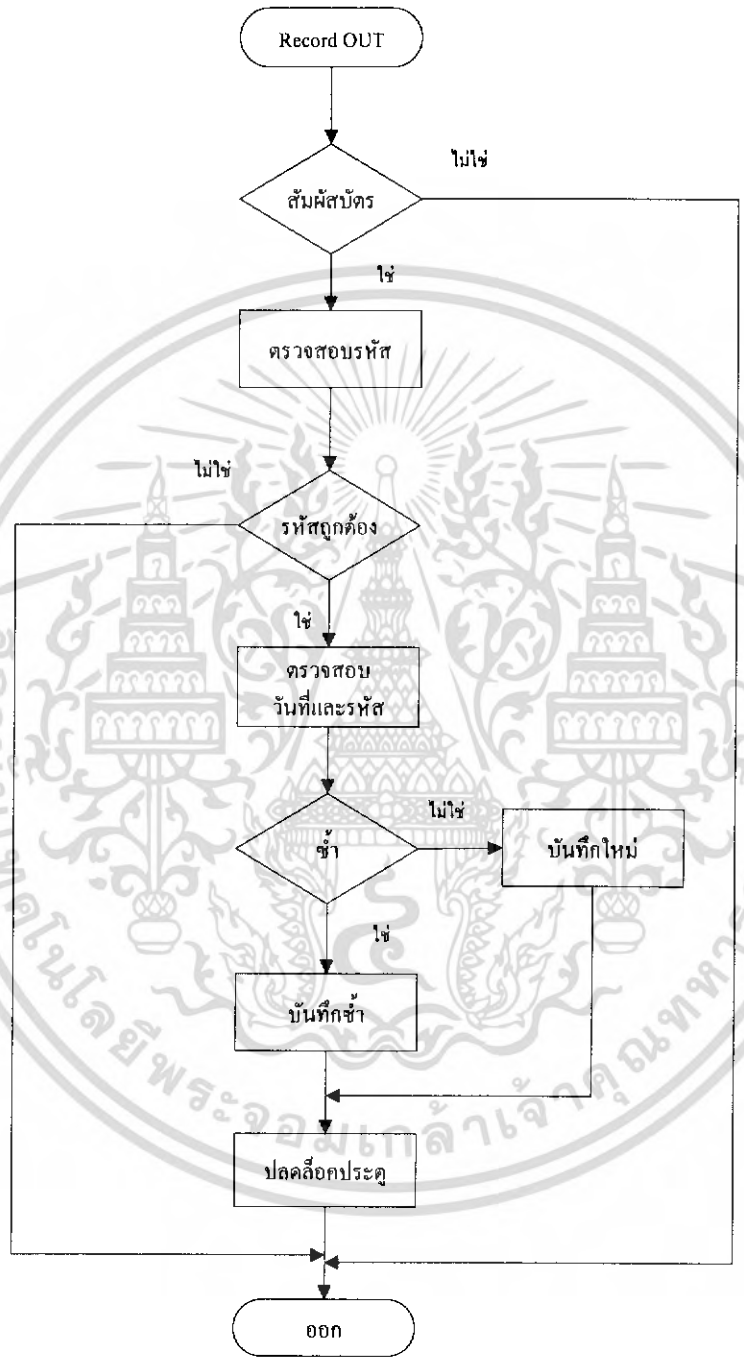
6.6 การออกแบบลำดับการทำงาน



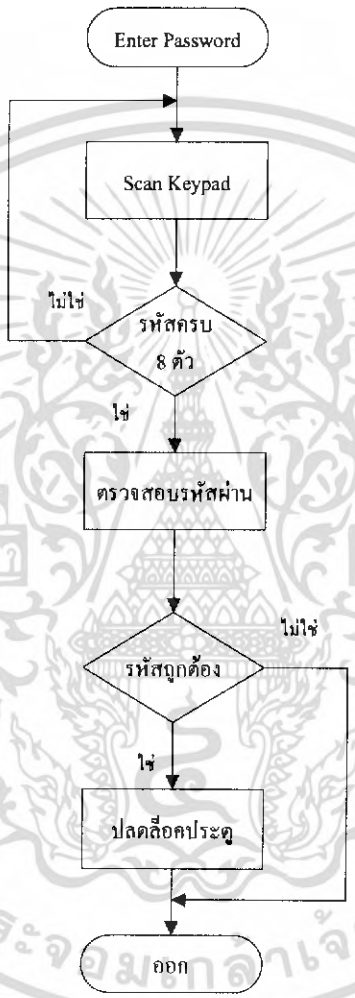
รูปที่ 6.3 ลำดับการทำงานของโปรแกรม



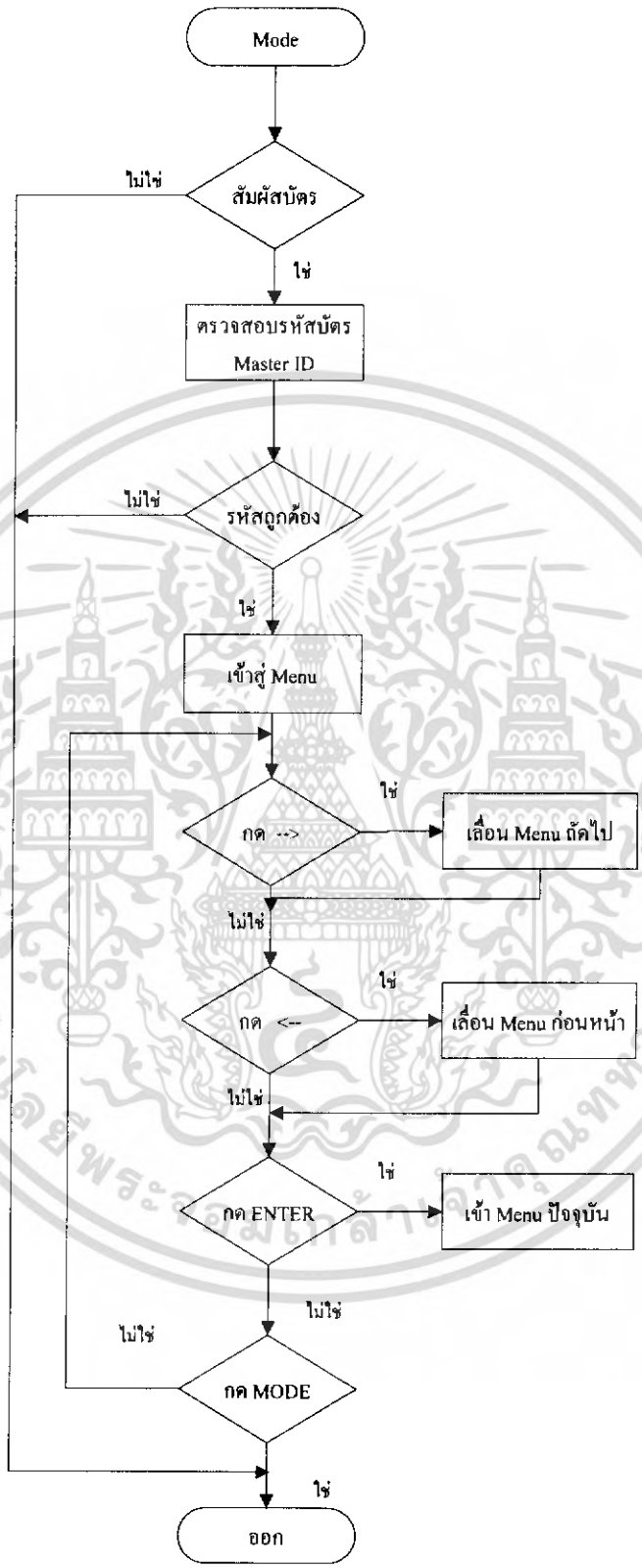
รูปที่ 6.4 ลำดับการทำงานบันทึกเวลาเข้า



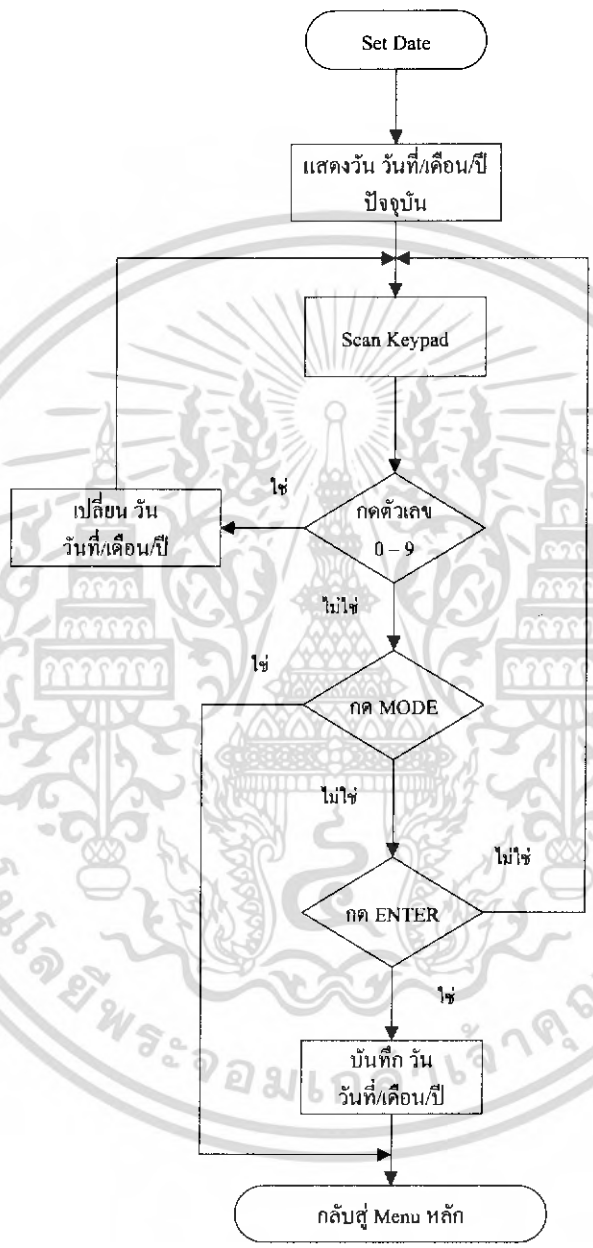
รูปที่ 6.5 ลำดับการทำงานบันทึกเวลาออก



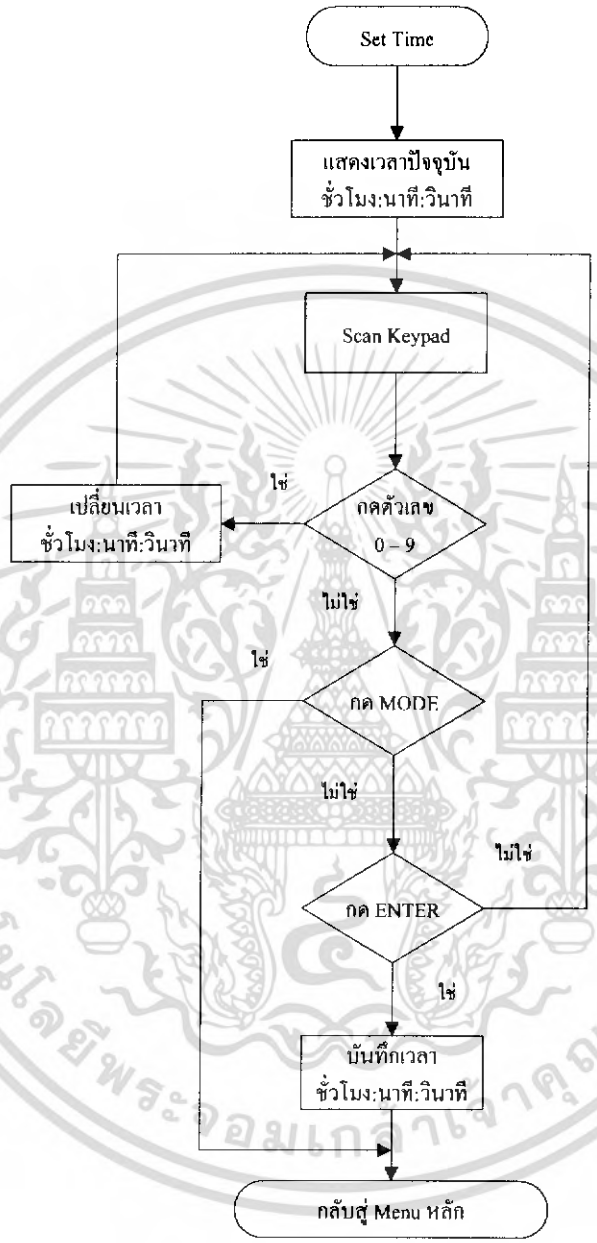
รูปที่ 6.6 ลำดับการทำงานป้อนรหัสผ่าน



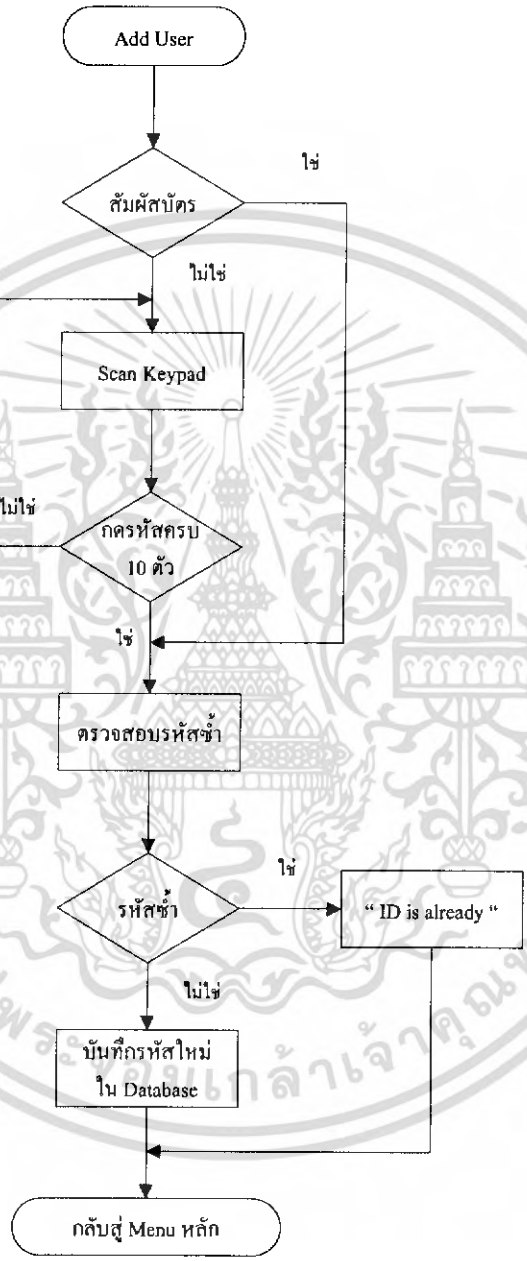
รูปที่ 6.7 ลำดับการทำงาน MODE



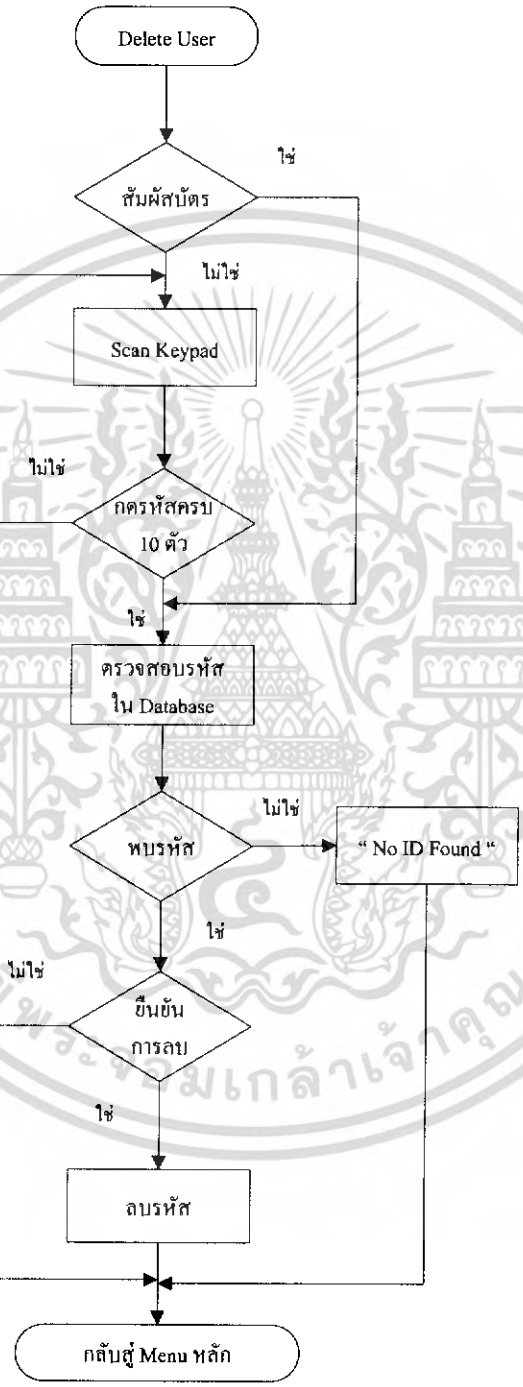
รูปที่ 6.8 ลำดับการทำงานโหมด Set Date



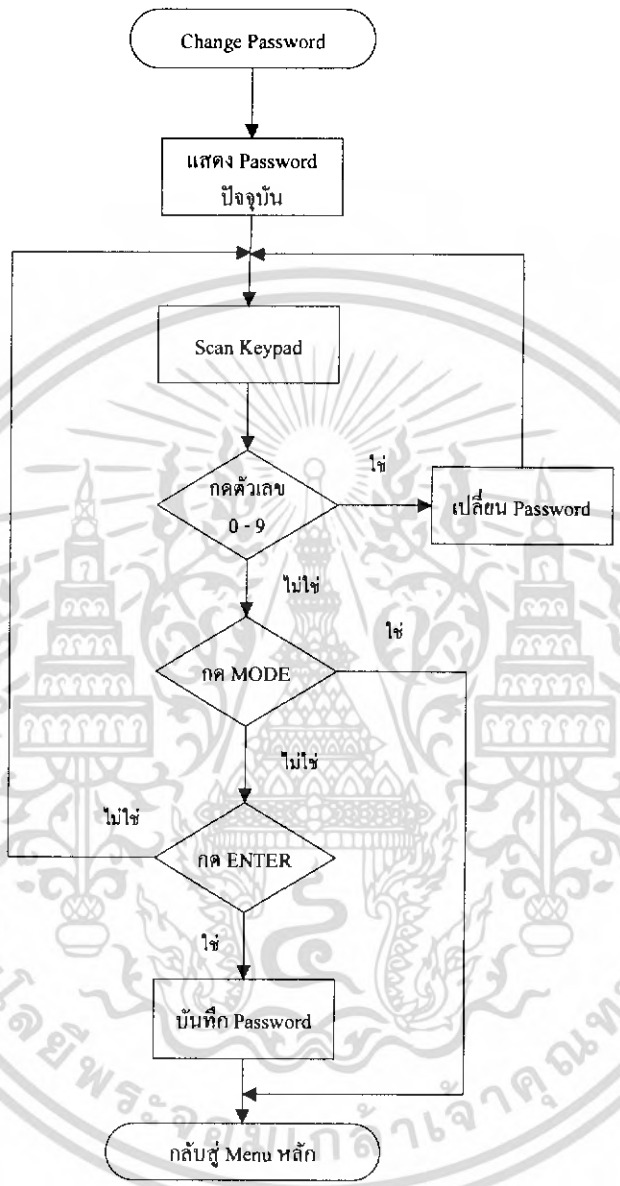
รูปที่ 6.9 ลำดับการทำงานโหมด Set Time



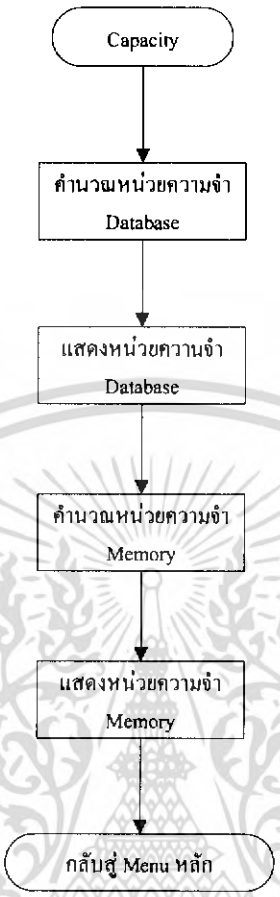
รูปที่ 6.10 ลำดับการทำงานโหมด Add User



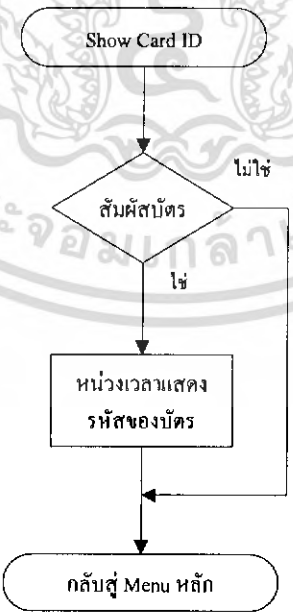
รูปที่ 6.11 ลำดับการทำงานโหมด Delete User



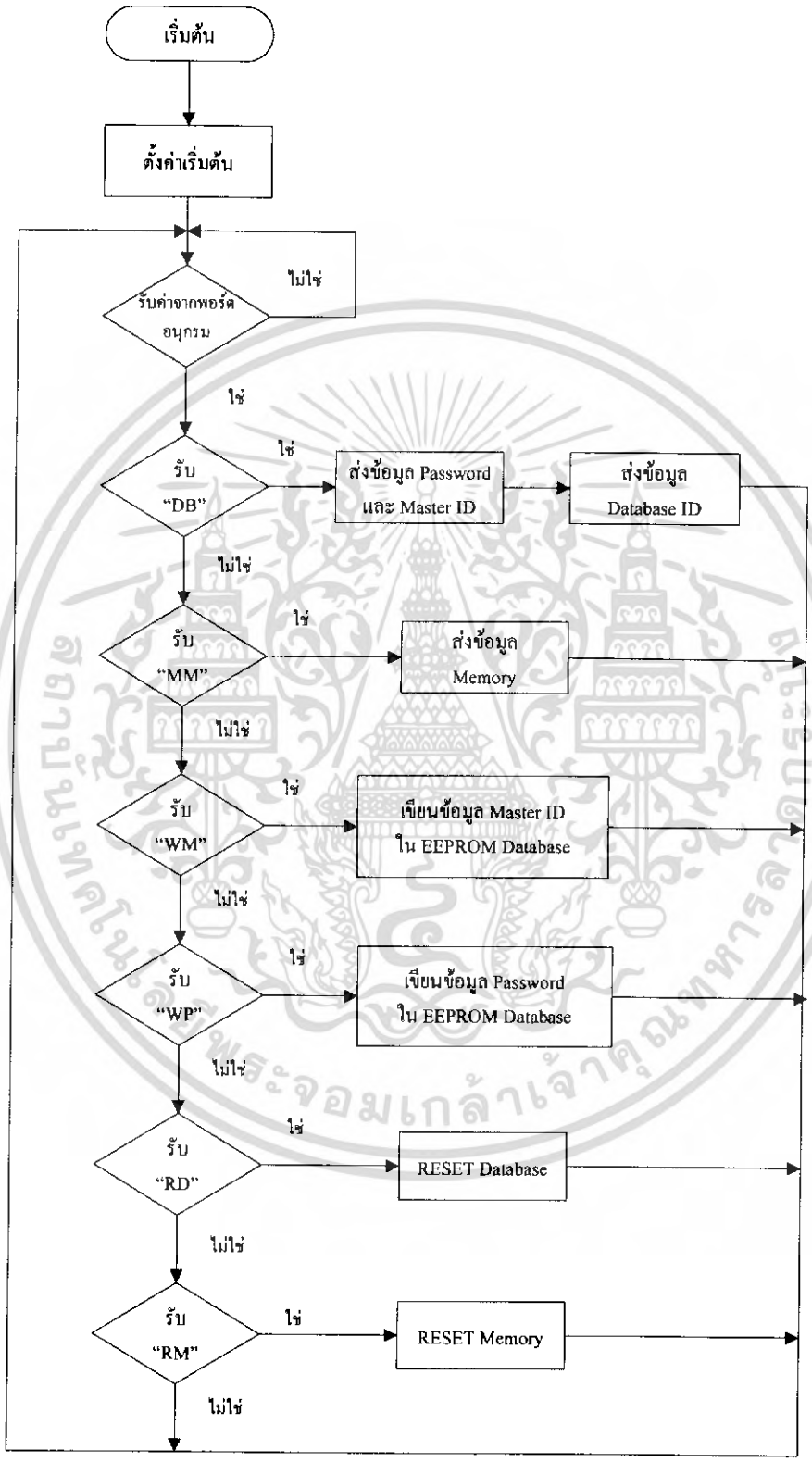
รูปที่ 6.12 ลำดับการทำงานโหมด Change Password



รูปที่ 6.13 ลำดับการทำงานโหมด Capacity



รูปที่ 6.14 ลำดับการทำงานโหมด Show Card ID

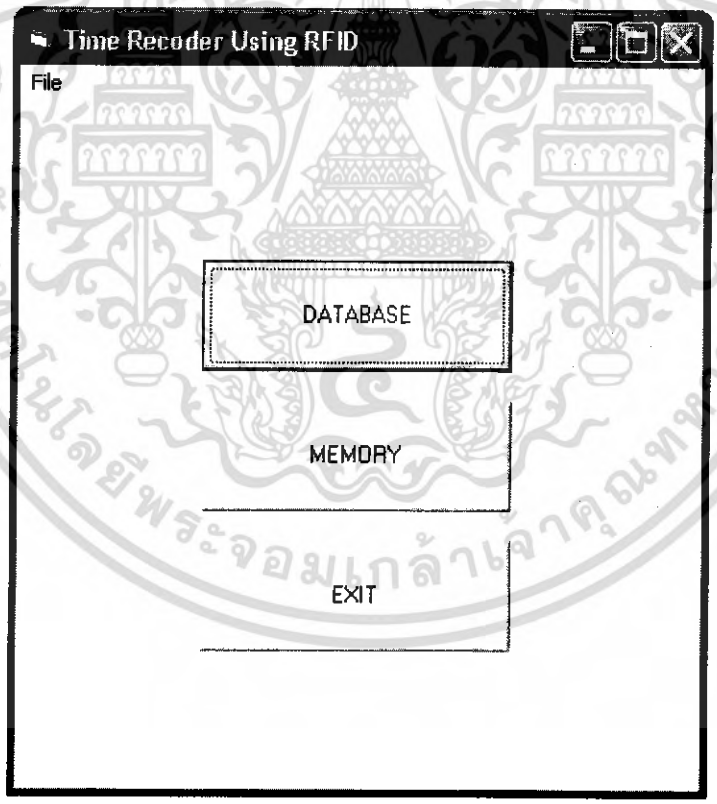


รูปที่ 6.15 ลำดับการทำงานของเครื่องอ่าน EEPROM

บทที่ 7
การเชื่อมต่อ EEPROM กับคอมพิวเตอร์

เครื่องบันทึกเวลาโดยใช้ RFID จะมีการบันทึกข้อมูลรหัสบัตรใน EEPROM Database และบันทึกข้อมูลเวลาใน EEPROM Memory ดังนั้นหากต้องการนำข้อมูลที่บันทึกอยู่ภายใน EEPROM มาใช้งานจึงจำเป็นต้องมีการดึงข้อมูลออกมาจาก EEPROM การเชื่อมต่อ EEPROM กับคอมพิวเตอร์ เพื่อแลกเปลี่ยนข้อมูลและนำข้อมูลมาเก็บบันทึกไว้ในฐานข้อมูลของคอมพิวเตอร์จะทำให้สามารถจัดการกับข้อมูลได้สะดวกมากยิ่งขึ้น การเชื่อมต่อกับคอมพิวเตอร์จะเชื่อมต่อผ่านพอร์ตอนุกรม Serial Port โดยใช้โปรแกรม Visual Basic 6.0 ในการเชื่อมต่อและจัดการกับข้อมูล

7.1 หน้าต่างหลักของโปรแกรม



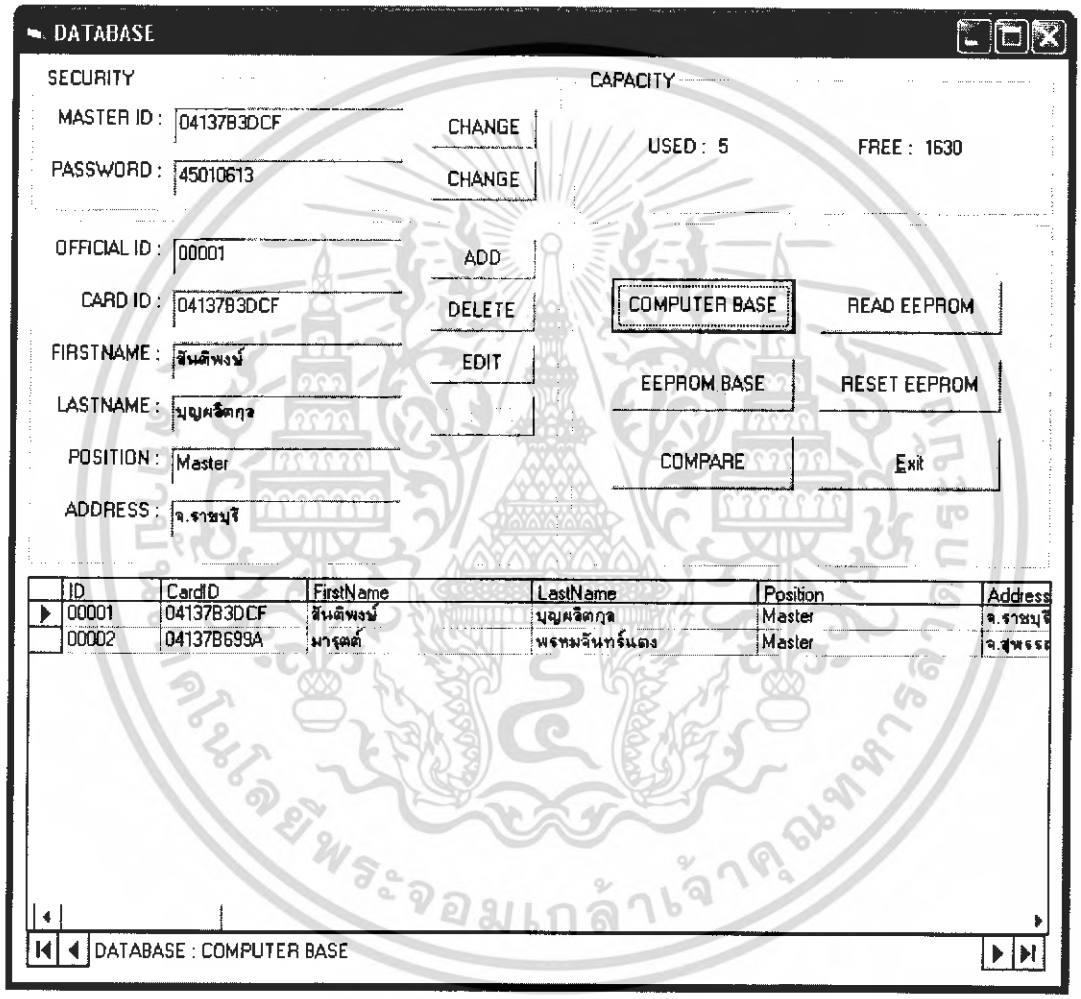
รูปที่ 7.1 หน้าต่างหลักของโปรแกรมเชื่อมต่อกับ EEPROM

ปุ่ม DATABASE เพื่อเลือกใช้งานฐานข้อมูล Database และใช้งาน EEPROM Database

ปุ่ม MEMORY เพื่อเลือกใช้งานฐานข้อมูล Memory และใช้งาน EEPROM Memory

ปุ่ม EXIT เพื่อออกจากโปรแกรม Time Recorder Using RFID

7.2 หน้าต่างฐานข้อมูล DATABASE



รูปที่ 7.2 หน้าต่างฐานข้อมูล DATABASE

เมื่อเปิดหน้าต่างฐานข้อมูล DATABASE จะแสดงตารางฐานข้อมูลรหัสที่บันทึกอยู่ภายในคอมพิวเตอร์ สามารถแก้ไข เพิ่มเติมรหัสบัตรและข้อมูลต่างๆ ในฐานข้อมูลคอมพิวเตอร์ได้ แต่หากไม่ทำการอ่านข้อมูลจาก EEPROM Database ก่อน จะไม่สามารถใช้งานฟังก์ชันที่เกี่ยวกับ EEPROM Database ได้เลย หลังจากการอ่านข้อมูลจาก EEPROM Databaseแล้ว จะ

แสดงรหัสบัตรของผู้ดูแลเครื่อง, รหัสผ่านประตู, ความจุของหน่วยความจำที่ใช้ไปและหน่วยความจำที่ยังเหลืออยู่ รวมทั้งแสดงตารางฐานข้อมูลรหัสบัตรที่อ่านได้จาก EEPROM Database ทั้งหมด และสามารถเปรียบเทียบรหัสบัตรที่อยู่ใน EEPROM Database กับรหัสบัตรที่มีอยู่ในคอมพิวเตอร์ได้ หากทำการ RESET EEPROM จะตั้งค่าเริ่มต้นใหม่ให้EEPROM Database

7.2.1 การทำงานของปุ่มกดต่างๆ

-ปุ่ม COMPUTER BASE เพื่อเลือกใช้งานฐานข้อมูล Database ที่มีอยู่ในคอมพิวเตอร์

-ปุ่ม EEPROM BASE เพื่อเลือกใช้งานฐานข้อมูล Database ที่อ่านได้จาก EEPROM Database

-ปุ่ม COMPARE เพื่อเปรียบเทียบรหัสบัตรที่อ่านได้จาก EEPROM Database กับฐานข้อมูลใน คอมพิวเตอร์

-ปุ่ม READ EEPROM เพื่ออ่านค่ารหัสบัตรที่บันทึกอยู่ใน EEPROM Database

-ปุ่ม RESET EEPROM เพื่อรีเซตค่ารหัสบัตรที่บันทึกอยู่ใน EEPROM Database

-ปุ่ม EXIT เพื่อออกจากหน้าต่างฐานข้อมูล DATABASE

-ปุ่ม CHANGE เพื่อเปลี่ยนรหัสบัตรของผู้ดูแลเครื่อง และรหัสผ่านประตู

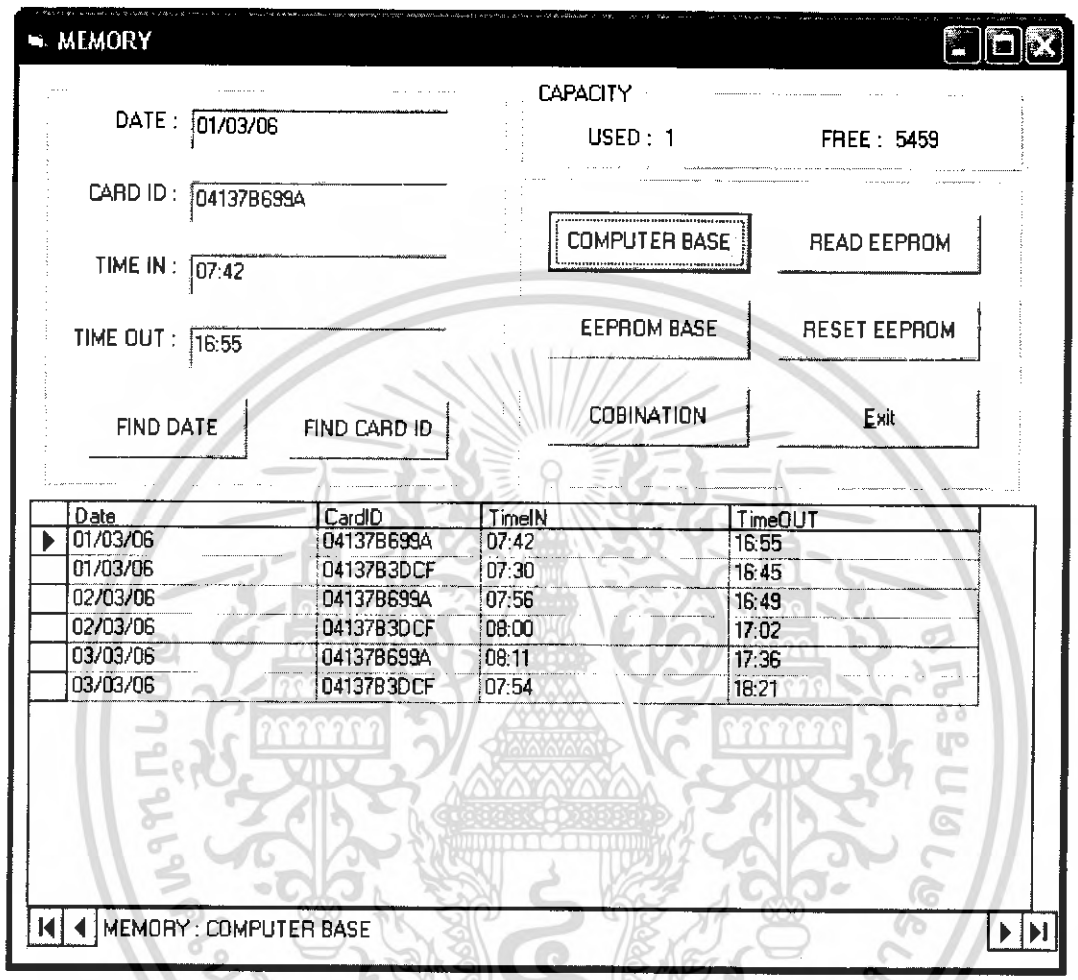
-ปุ่ม ADD เพื่อเพิ่มรหัสบัตรและข้อมูลใหม่ในฐานข้อมูลคอมพิวเตอร์

-ปุ่ม DELETE เพื่อลบรหัสบัตรและข้อมูลที่แสดงอยู่ในขณะนั้นออกจากฐานข้อมูลคอมพิวเตอร์

-ปุ่ม EDIT เพื่อแก้ไขรหัสบัตรและข้อมูลในฐานข้อมูลคอมพิวเตอร์ที่แสดงอยู่ขณะนั้น

-ปุ่ม UPDATE เพื่อยืนยันการบันทึกรหัสบัตรและข้อมูลลงในฐานข้อมูลคอมพิวเตอร์

7.3 หน้าต่างฐานข้อมูล MEMORY



รูปที่ 7.3 หน้าต่างฐานข้อมูล MEMORY

เมื่อเปิดหน้าต่างฐานข้อมูล MEMORY จะแสดงตารางฐานข้อมูลรหัสบัตรและเวลาที่บันทึกอยู่ภายในคอมพิวเตอร์ สามารถค้นหาข้อมูลวันที่หรือรหัสบัตรที่ต้องการได้ หลังจากการอ่านข้อมูลจาก EEPROM Memory แล้วจะแสดงความจุของหน่วยความจำที่ใช้ไปและหน่วยความจำที่ยังเหลืออยู่ รวมทั้งแสดงตารางฐานข้อมูลรหัสบัตรและเวลาที่อ่านได้จาก EEPROM Memory และสามารถบันทึกข้อมูลรหัสบัตรและเวลาที่อ่านได้จาก EEPROM Memory ไปไว้รวมกับฐานข้อมูลคอมพิวเตอร์ได้ หากทำการ RESET EEPROM จะตั้งค่าเริ่มต้นใหม่ให้ EEPROM Memory

7.3.1 การทำงานของปุ่มกดต่างๆ

-ปุ่ม COMPUTER BASE เพื่อเลือกใช้งานฐานข้อมูล Memory ที่มีอยู่ในคอมพิวเตอร์

-ปุ่ม EEPROM BASE เพื่อเลือกใช้งานฐานข้อมูล Memory ที่อ่านได้จาก EEPROM Memory

-ปุ่ม COBINATION เพื่อบันทึกข้อมูลจาก EEPROM Memory ไว้ในฐานข้อมูลคอมพิวเตอร์

-ปุ่ม READ EEPROM เพื่ออ่านข้อมูลเวลาที่บันทึกอยู่ใน EEPROM Memory

-ปุ่ม RESET EEPROM เพื่อรีเซตข้อมูลที่บันทึกอยู่ใน EEPROM Memory

-ปุ่ม EXIT เพื่อออกจากหน้าต่างฐานข้อมูล MEMORY

-ปุ่ม FIND DATE เพื่อค้นหาข้อมูลเฉพาะวันที่ ที่ต้องการเท่านั้น

-ปุ่ม FIMD ID เพื่อค้นหาข้อมูลเฉพาะรหัสบัตรที่ต้องการเท่านั้น

บทที่8
สรุปและวิจารณ์

8.1 สรุปและวิจารณ์

เครื่องบันทึกเวลาโดยใช้ RFID ออกแบบมาสำหรับติดตั้งบริเวณทางเข้าประตู เพื่อใช้สำหรับการบันทึกลงเวลาเข้า-ออกงานและรักษาความปลอดภัยของผู้ใช้งาน โดยการรักษาความปลอดภัยนั้นจะเป็นการอนุญาตให้เฉพาะผู้ที่มีบัตร ซึ่งมีรหัสของบัตรบันทึกไว้ที่หน่วยความจำของเครื่องหรือผู้ที่ทราบรหัสผ่านปลดล็อกประตูเท่านั้น ผู้ที่ไม่มีบัตรไม่ทราบรหัสผ่านและรหัสบัตรไม่ตรงกับรหัสในหน่วยความจำจะไม่สามารถปลดล็อกประตูได้ การใช้งานบันทึกเวลาจะต้อง กดเลือกการบันทึกเวลาก่อนคือ IN หรือ OUT แล้วจึงสัมผัสบัตรกับตัวเครื่อง หากไม่มีการกดเลือกการบันทึกเวลาก่อนจะเป็นเพียงการปลดล็อกประตูเท่านั้น และหากรหัสบัตรไม่ตรงกับหน่ายความจำก็จะไม่ทำการบันทึกเวลาทั้งเข้าและออก สำหรับการใช้งานของผู้ดูแลเครื่องสามารถปรับตั้งค่าต่างๆได้คือ Set Date, Set Time, Change Password, Add User, Delete User,ดูหน่วยความจำ Capacityและดู ID ของบัตร โดยการเข้าไปปรับตั้งค่าจะสามารถกระทำได้เพียงผู้ดูแลเครื่องที่มีบัตร Master ID เท่านั้น ผู้ใช้งานทั่วไปไม่สามารถเข้าไปใช้งานในโหมดนี้ได้ สำหรับการนำข้อมูลที่บันทึกในหน่วยความจำ EEPROM มาใช้งาน จะนำมาใช้ร่วมกับคอมพิวเตอร์ผ่านพอร์ตอนุกรมโดยใช้อีกบอร์ดหนึ่งเป็นตัวจัดการกับข้อมูลใน EEPROM ผ่านโปรแกรม Visual Basic เพื่ออ่านค่าข้อมูลรหัสบัตร Databaseและ Memory อีกทั้งยังใช้ในการเปลี่ยนแปลงข้อมูลรหัสผ่านปลดล็อกประตูและรหัสบัตรของผู้ดูแลเครื่อง และใช้ในการตั้งค่าเริ่มต้นใหม่ให้กับ Memory หลังจากอ่านข้อมูลมาเก็บไว้ในเครื่องคอมพิวเตอร์แล้ว

เครื่องบันทึกเวลาโดยใช้ RFID สร้างขึ้นจากไมโครคอนโทรเลอร์ตระกูล MCS-51 เบอร์ AT89C52 มีหน่วยความจำโปรแกรมขนาด 8 กิโลไบต์ ซึ่งเพียงพอกับการเขียนโปรแกรมเพื่อการประมวลผลและการสั่งงาน ในส่วนของการแสดงผลเพื่อติดต่อกับผู้ใช้งานให้ทราบสถานการณ์ทำงาน โดยแสดงผลทั้งข้อความจาก LCD, หลอดไฟ LED และเสียงจาก Buzzer สามารถใช้งานได้อย่างสะดวกและมีประสิทธิภาพ รหัสที่บันทึกในหน่วยความจำและการตรวจสอบรหัสบัตร ทำได้อย่างถูกต้องแม่นยำและรวดเร็ว (การตรวจสอบรหัสผ่านปลดล็อกประตู) รวมไปถึงการปรับตั้งค่าของผู้ดูแลเครื่องก็สามารถทำได้อย่างถูกต้อง ทั้งการเปลี่ยนวัน, วันที่, เดือน, ปี, เวลา, การบันทึกเพิ่ม ID , การลบ ID, การเปลี่ยนรหัสผ่านปลดล็อกประตูและการดูหน่วยความจำที่เหลือ การถ่ายโอนข้อมูลจาก EEPROM ไปเก็บไว้ในคอมพิวเตอร์ทำได้อย่างถูกต้อง โดยการสั่งงานจาก

คอมพิวเตอร์ให้ไมโครคอนโทรเลอร์ เบอร์ AT89C2051 ทำการถ่ายโอนข้อมูล ผ่านโปรแกรม Visual Basic ยังทำได้ไม่ดีนัก ทำได้เพียงอ่านข้อมูลจาก EEPROM มาเก็บไว้ในไฟล์และสั่งเขียนข้อมูลลงใน EEPROM ไม่ได้ทำให้สามารถจัดการกับข้อมูลได้หลากหลายกว่านี้ เช่นการแยกเก็บข้อมูลเป็นรายเดือน, การตรวจสอบการเข้างานสาย, การทำการเก็บข้อมูลแบบแฟ้มประวัติรายบุคคล ,การทำเป็นรูปแบบรายงานฯลฯ ซึ่งการจะทำให้มีรายละเอียดมากๆและซับซ้อนต้องอาศัยระยะเวลาในการศึกษาและความชำนาญด้านการเขียนโปรแกรม Visual Basic

8.2 ปัญหาในการทำงานและแนวทางแก้ไข

1. เครื่องบันทึกเวลาโดยใช้ RFID จำเป็นต้องใช้เครื่องอ่านบัตร RFID เครื่องอ่านบัตรที่คาดการณ์ไว้ว่าจะใช้ในโครงงานนี้ ซึ่งสามารถใช้งานได้ในระยะไกลไม่สามารถหาซื้อได้ หลังจากการใช้เวลานานในการหาแล้วไม่ได้ จึงตัดสินใจเปลี่ยนเป็นตัว GP8-10 แทน ซึ่งสามารถใช้ได้เพียงระยะใกล้ๆเท่านั้น
2. เนื่องจากการเขียนโปรแกรมเพื่อควบคุมการทำงานของไมโครคอนโทรลเลอร์ในโครงงานนี้ ได้ใช้ภาษาC ในการเขียนซึ่งไม่เคยใช้มาก่อน จึงต้องใช้เวลานานในการศึกษาทำความเข้าใจหลักการเขียนโปรแกรม
3. เมื่อเขียนโปรแกรมเพื่อควบคุมไมโครคอนโทรลเลอร์แล้ว เกิดปัญหาเนื่องจากการลำดับการทำงานไม่ดีพอ ทำให้ไม่สามารถทำงานได้ถูกต้องและต้องแก้ไขโดยการลำดับการทำงานใหม่
4. การถ่ายโอนข้อมูลระหว่าง EEPROM กับคอมพิวเตอร์ ไม่สามารถทำได้โดยตรง จึงจำเป็นต้องใช้ไมโครคอนโทรเลอร์อีกตัวมาเป็นตัวประมวลผลและแลกเปลี่ยนระบบบัส I^2C กับ EEPROM และพอร์ตอนุกรมกับคอมพิวเตอร์
5. การเขียนโปรแกรม Visual Basic เพื่อควบคุมการทำงานและแลกเปลี่ยนข้อมูลกับ EEPROM กระทำได้เพียงอ่านและเขียนข้อมูลจาก EEPROM เท่านั้น ไม่สามารถทำให้ใช้งานได้หลากหลายในด้านของการจัดการกับฐานข้อมูล อาจสามารถทำได้ดีขึ้นหากมีความชำนาญในการเขียนโปรแกรม Visual Basic ในด้านของฐานข้อมูลมากกว่านี้

หนังสืออ้างอิง

1. ธีรบูลย์ หล่อวิเชียรรุ่ง,นคร ภักดีชาติ,ชัยวัฒน์ ลิ้มพรจิตรวิไล, " ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษา C ",หน้า 163-171.

2. วรพจน์ กรแก้ววัฒนกุล, ชัยวัฒน์ ลิ้มพรจิตรวิไล, " เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช ",หน้า 236-243.

3. วัชรากร หนูทอง, อนุกูล น้อยไม้, ปรินันท์ วรรณสว่าง, " เอกสารเผยแพร่เทคโนโลยี RFID ",2547,หน้า 1-9.

4. อรรถพล บุญยะโภคา,วรพจน์ กรแก้ววัฒนกุล,ชัยวัฒน์ ลิ้มพรจิตรวิไล, "เรียนรู้และปฏิบัติการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่านพอร์ตอนุกรม",หน้า 93-104.

5. ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ, " สาร NECTEC ",เดือนกันยายน-ตุลาคม,2547,หน้า 11-60 และ 15-22.

6. ฉัททวุฒิ พืชผล,พิชิต สันติกุลานนท์,พร้อมเลิศ หล่อวีจิตร, "คู่มือเรียน Visual Basic 6 ", เดือนกรกฎาคม,2547,หน้า1-205.

ภาคผนวก

ProxID Micro Reader GP8-10
Instruction Manual

Contents

Rev 1.46A May 15, 2000

Information in this manual is subject to change without notice.

1. Introduction

The GP8-10 is a low cost high performance OEM proximity reader module for use with a simple external antenna. The module features medium read range and small dimensions. The GP8-10 also has good read range at 5 Volts, making it ideally suited to a wide variety of applications, particularly access control. The same basic unit can be configured to most of the common output formats, including Wiegand and Magstripe, making it easy to upgrade existing installations.

2. Features

* **Low cost**
* **Medium read range**
* **Small outline**
* **Wide Voltage range**
* **Potted for environmental protection**
* **Externally programmable interface**
* **Plug-in fitting**

3. Theory

The reader generates a 125KHz inductive field that extends some way beyond the reader module. When a transponder is placed within the vicinity of the reader module, it draws power from this field and providing the field is of sufficient strength the internal microcircuits contained in the transponder begin to function. Data is transferred from the transponder by means of amplitude modulation in such a manner that the transponder varies the rate at which it draws power from the field in a way that corresponds to the internal identity code programmed in this internal memory. These changes in field power can be detected by the reader and converted back into a copy of the original data.

4. Specifications

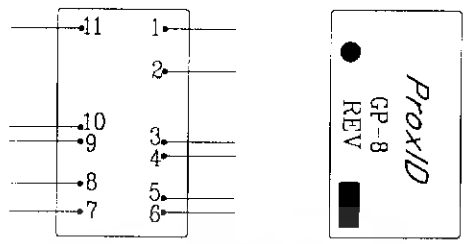
Power Requirements	5-13.5 Volts regulated DC at 55 mA typical with a 12V supply. A linear regulator is recommended.
Interface	Wiegand, Magstripe, 9.6 K Baud Serial ASCII (RS232) or special to customer specifications.
Read Range	Production Pass range is 5.0cm @ 12V with ISO card
Typical Maximum Read - in ideal conditions	Range 7.0 cm at 12.0V with ISO card.
Frequency	125KHz standard or 134.2KHz to special order.
Transponder	Read Only.
Audio/visual Indication	LED output and Buzzer
Dimensions	41 x 24 x 10mm
Weight	<50gm

5. Pin Assignment

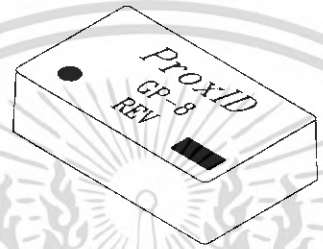
Pin1)	Power 0 Volt
Pin2)	Power 5.0-13.5 Volts
Pin3)	Program Input
Pin4)	Card Present Output with internal 4K7 pull-up
Pin5)	Data Output RS232, Magstripe data & Wiegand0, with internal 4K7 pull-up (pull up only for Wiegand and Magstripe)
Pin6)	Magstripe clock & Wiegand1, with internal 4K7 pull-up
Pin7)	External Beep control input *
Pin8)	LED Drive (use 470-IK series resistor)
Pin9)	Buzzer pre-driver (requires driver transistor)
Pin10)	External Antenna Ground
Pin11)	External Antenna Drive

Note * Connect Pin1 to Pin7 if external beeper control is not required

GP-8 module



Bottom view Top view



6. Programming

The output format can be customer programmed. The available formats are Wiegand, Magnetic Emulation and Serial ASCII (RS232)

Wiegand		**Magstripe**	
Pin 1	Ground 0V	Pin 1	Ground 0V
Pin 2	Power +V	Pin 2	Power +V
Pin 5	Data0	Pin 6	Clock (Strobe)
Pin 6	Data1	Pin 5	Data
Pin 3	Connect to Pin 6	Pin 4	Card Present
Pin 4	No Connection	Pin 3	Connect to Pin 4

Serial ASCII (RS232)

Pin 1	Ground 0V
Pin 2	Power +V
Pin 5	Data
Pin 3	No Connection
Pin 4	No Connection
Pin 6	No Connection

7. Data Structure

7-1. Data Structure (ASCII)

BAUD RATE: 9600,N,8,1

STX(02 HEX)	DATA (10 HEX)	CR	LF	ETX (03 HEX)

The start character is factory defined as an 'STX' (02 HEX). This is followed by 10 Hex characters of data. The CR\LF characters serve to bring the received screen text back to the left hand side and on the line below after the data bytes have been sent. The 'ETX ' (03 HEX) character denotes the end of the current transmission.

7-2. Data Structure (Magstripe)

Speed : Simulated to 40 IPS (Inch per Second)

10 LEADING ZEROS	SS	DATA (14 DIGITS)	ES	LRC	10 TRAILING ZEROS

The 10 leading zeros prepare the receiving unit to accept the data. The data is 14 digits long. SS is the Start Sentinel consisting of 11010. ES is the End Sentinel consisting of 11111. LRC is the Longitudinal Redundancy Check character. Lastly there are 10 trailing zeros. Magstripe 8 digits and 6 digits are available for special request.

The hexadecimal data from the card is first converted to a denary string before transmission. For example, a card containing the hexadecimal data (0411115EA6) , will be converted to denary and sent as denary **00017466220198** (14 digits)

$(6* 16^0 + 10 * 16^1 + 14 *16^2 + 5 * 16^3 + 1 * 16^4 + 1*16^5 + 1 * 16^6 + 1 * 16^7 + 4 * 16^8) =$ **00017466220198**

7-3 Data Structure (Wiegand Format-26 Bit)

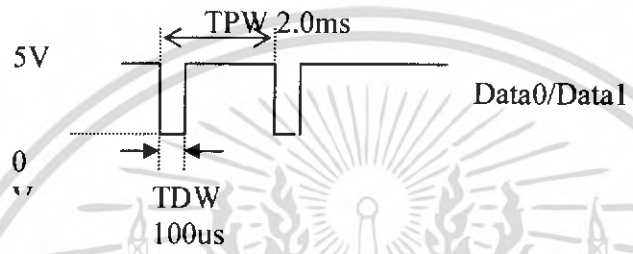
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P	S	S	S	S	S	S	S	S	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	P
P	E	E	E	E	E	E	E	E	E	E	E	E													
													O	O	O	O	O	O	O	O	O	O	O	O	P
SUMMED FOR EVEN PARITY (E)													SUMMED FOR ODD PARITY (O)												

Note:

P	Parity (Even or Odd) Start Bit and Stop Bit
S	Site Bits from Card or Reader
C	Card Number Bits from Card
SYRDSSW1-W26	Site bits from Card (24 bits Card Data)
MSB	Normal 01
LSB	Normal 24

Data Timing Specification

Pulse Interval(TPW)	2.0mS +/- 3%
Pulse Width (TDW)	100uS +/- 3%



8. Trouble Shooting

In case of problems the following procedure should be followed.

Failure to read

1) Turn off the power to the GP8.
2) Check the power input connections making sure that they are not reversed.
3) Check the programming pin is correctly connected.
4) Measure the supply voltage and confirm it is in the range 5-13.5V.
5) If the supply has a current limit, set this to 100mA.
6) Turn on the power.
7) Ensure the current is below 100mA, if there is no current being drawn or the current is in excess of 100mA, then the unit has possibly sustained damage.
8) Check that the LED drive goes high immediately upon switch on. This will indicate that the processor is functioning.
9) Ensure the external buzzer (pin7) is at logic '0'(off) or logic '1'(on)
10) Ensure the power supply output is free from ripple and noise.

9. External Antenna

A simple experimental antennae consisting of 8 turns of 0.8mm diameter copper wire loosely wound on a 10x10 cm former and measuring 16uH will give a range of approximately 30cm when tuned with a 0.1uF capacitor. Ranges in excess of 40cm can be obtained with a suitable antennae and capacitor. The required inductance will differ if the GP8 is placed within the coil and required values can vary from 16-17.1uH depending on the exact position. For example a small antenna wound around the GP8 will need to be about 17uH however a 10cm radius coil will require a value of about 16uH.

Note that the best read ranges are found when the inductance is made slightly more than that required for resonance. Polypropylene, Polycarbonate or Polyphenylene-Sulphide capacitors are recommended. Ensure the tuning capacitors are capable of working continuously at 125KHz at the measured AC voltage present on the coil (6-18vRMS) depending on coil Q. Use only low loss capacitors.

Suitable tuning capacitors include:

Wima MKP 2 0.1/100 VDC	*15 VRMS max @ 80 ℃*	*(PP)*
Wima MKI 2 0.1/100 VDC	*12 VRMS max @80 ℃*	*(PPS)*
Wima SMR 100nF100VDC		*(PPS)*
Panasonic ECHU1H104	*7 VRMS*	*(PPS)*
Evox Rifa PMR15 104K250L4	*32VRMS max @ 80 ℃*	*(PP) (0.2m/s air flow)*
Evox Rifa PHE427FB6100J	*43VRMS max @ 80 ℃*	*(PP) (0.2m/s air flow)*

CODE ของเครื่อง Time Recorder Using RFID

```
#include <AT89X52.H>
#include <lcd16x2.h>
#include <scankey4x4.h>
#include <i2c.h>
#define DS1307_ID 0xD0
#define EEPROM_MEM 0xA0              // EEPROM 512 Kbits.
#define EEPROM_DB 0xA2               // EEPROM 64 Kbits. New Board use 0xA2.
#define DELAY_TIME 0x03E8    // Define value for Delay Time.(0x01F4=5s,0x03E8=10s)

unsigned char rd_buf[14],rd_id[5],pwd_in[8],eeprom_buf[5];
unsigned char key,m_page,addr_h,addr_l,count_10ms;          // Global Parameter.
unsigned char n,m;                                          // Variable Parameter.
unsigned int time_10ms;

sbit sw_out = P2^2;
sbit buzzer = P2^3;
sbit led_g = P2^4;
sbit led_r = P2^5;
sbit relay = P2^6;
sbit lcd_bl = P2^7;                  //Turn On-Off LCD's Black Light.

bit chk_in = 0;
bit chk_out = 0;
bit chk_int = 0;
bit chk_swap = 0;
bit chk_id = 0;
bit chk_on = 0;
bit chk_sday = 0;
bit err = 0;
bit chk_loop = 0;
bit chk_time = 0;

code unsigned char *scr1 = "TIME RECORDER";
code unsigned char *scr2 = "USING RFID";
code unsigned char *recin = "Record IN";
code unsigned char *recout = "Record OUT";
code unsigned char *touch = "<< TOUCH CARD >>";
code unsigned char *complete = "COMPLETE!";
code unsigned char *valid = "Card Valid";
code unsigned char *invalid = "Card Invalid";
code unsigned char *en_pass = "Enter Password";
code unsigned char *master = "Card Master";
code unsigned char *s_date = "SET DATE";
code unsigned char *s_time = "SET TIME";
code unsigned char *s_add = "ADD USER";
code unsigned char *s_del = "DELETE USER";
code unsigned char *s_pwd = "CHANGE PASSWORD";
code unsigned char *s_cap = "CAPACITY";
code unsigned char *s_sho = "SHOW CARD ID";
code unsigned char *u_f = "USED  FREE";
code unsigned char *nfound = "No ID Found";
code unsigned char *con_del = "Comfirm Delete";
code unsigned char *n_del = "Not Delete";
code unsigned char *already = "ID is Already!";
code unsigned char *error = "ERROR!";
```

```
/*-----------------------------------------------*/
/*        Declaration For RTC                    */
/*-----------------------------------------------*/
unsigned char sec,min,hour,day,date,month,year,control;
unsigned char day_buf[3];
unsigned char date_buf[6];
unsigned char time_buf[6];

void send_to_lcd(unsigned char value)
{
        unsigned char buf = 0;
        buf = value & 0xF0;    // Filter for high byte
        buf = (buf>>4)|(0x30);  // convert to ascii code
        lcd_text(buf);
        buf = value & 0x0F;                    // Filter for low byte
        buf = buf | 0x30;      // convert to ascii code
        lcd_text(buf);
}

unsigned char DS1307_rd(unsigned char addr_rtc)
{
        unsigned char ret;
        i2c_start();
        i2c_wrdata(DS1307_ID);  // Write DS1307 ID for connect
        i2c_wrdata(addr_rtc);       // Write RAM address on DS1307 for connect
        i2c_start();
        i2c_wrdata(DS1307_ID+1);// Write DS1307 ID for Read Mode connect
        ret = i2c_rddata();
        i2c_stop();
        return(ret);
}

void DS1307_wrtime(void)
{
        hour = (time_buf[0]<<4)|(time_buf[1]);      // Convet input hour to Hex
        min = (time_buf[2]<<4)|(time_buf[3]);       // Convet input min to Hex
        sec = (time_buf[4]<<4)|(time_buf[5]);       // Convet input sec to Hex
        if(hour>0x23)
        hour = 0x23;                                // IF hour over range reload maximum
        if(min>0x59)
        min = 0x59;                                 // IF min over range reload maximum
        if(sec>0x59)
        sec = 0x59;                                 // IF sec over range reload maximum
        i2c_start();
        i2c_wrdata(DS1307_ID);                      // Write DS1307 ID for connect
        i2c_wrdata(0x00);                           // Write control byte to access RAM addr 00H
        i2c_wrdata(sec);                            // Write sec on RAM address 00H
        i2c_wrdata(min);                            // Write min on RAM address 01H
        i2c_wrdata(hour);                           // Write hour on RAM address 02H
        i2c_stop();
}
```

```
void DS1307_wrdate(void)
{
        date = (date_buf[0]<<4)|(date_buf[1]);      // Convet input date to Hex
        month = (date_buf[2]<<4)|(date_buf[3]);     // Convet input month to Hex
        year = (date_buf[4]<<4)|(date_buf[5]);      // Convet input year value Hex
        if(year>0x99)
        year = 0x99;                                // IF year over range reload maximum
        if(month>0x12)
        month = 0x12;                               // IF month over range reload maximum
        if(date>0x31)
        date = 0x31;                                // IF date over range reload maximum
        i2c_start();                                // i2c start condition
        i2c_wrdata(DS1307_ID);                      // Write DS1307 ID for connect
        i2c_wrdata(0x03);                           // Write control byte to access RAM addr 04H
        i2c_wrdata(day);                            // Write Day.
        i2c_wrdata(date);                           // Write date on RAM address 04H
        i2c_wrdata(month);                          // Write month on RAM address 05H
        i2c_wrdata(year);                           // Write year on RAM address 06H
        i2c_stop();                                 // i2c stop condition
}

void sw_day (void)
{
switch (day)
        {
                case 1:  day_buf[0] = 'S';
                         day_buf[1] = 'U';
                         day_buf[2] = 'N';
                         break;
                case 2:  day_buf[0] = 'M';
                         day_buf[1] = 'O';
                         day_buf[2] = 'N';
                         break;
                case 3:  day_buf[0] = 'T';
                         day_buf[1] = 'U';
                         day_buf[2] = 'E';
                         break;
                case 4:  day_buf[0] = 'W';
                         day_buf[1] = 'E';
                         day_buf[2] = 'D';
                         break;
                case 5:  day_buf[0] = 'T';
                         day_buf[1] = 'H';
                         day_buf[2] = 'U';
                         break;
                case 6:  day_buf[0] = 'F';
                         day_buf[1] = 'R';
                         day_buf[2] = 'I';
                         break;
                case 7:  day_buf[0] = 'S';
                         day_buf[1] = 'A';
                         day_buf[2] = 'T';
                         break;
                default: day_buf[0] = '1';
                         day_buf[1] = '-';
                         day_buf[2] = '7';
                         day = 0x01;
```

```
                        chk_sday = 0;
                        break;
        }
}

void show_date (void)
{
        day = DS1307_rd(0x03);          // Read day before show on LCD
        date = DS1307_rd(0x04);         // Read date before show on LCD
        month = DS1307_rd(0x05);        // Read month before show on LCD
        year = DS1307_rd(0x06);         // Read year before show on LCD
        sw_day();
        lcd_command(0x01);
        lcd_command(0x81);
        for (n=0;n<3;n++)
                lcd_text(day_buf[n]);
        lcd_command(0x86);                      // Set LCD address 06H
        send_to_lcd(date);                      // Write date show on LCD
        lcd_text('/');                          // Write "/" show on LCD
        send_to_lcd(month);                     // Write month show on LCD
        lcd_text('/');                          // Write "/" show on LCD
        send_to_lcd(year);                      // Write year show on LCD
}

void show_time (void)
{
        sec = DS1307_rd(0x00);          // Read sec.
        min = DS1307_rd(0x01);          // Read min.
        hour = DS1307_rd(0x02);         // Read hour.
        lcd_command(0xC6);
        send_to_lcd(hour);
        lcd_text(':');
        send_to_lcd(min);
        lcd_text(':');
        send_to_lcd(sec);
}

void show_setdate (void)
{
        unsigned char buf = 0;
        lcd_command(0x01);
        lcd_command(0x80);
        for(n=0;n<8;n++)
                lcd_text(*(s_date+n));
        day = DS1307_rd(0x03);
        date = DS1307_rd(0x04);
        month = DS1307_rd(0x05);
        year = DS1307_rd(0x06);
        sw_day();
        lcd_command(0xC1);
        for (n=0;n<3;n++)
                lcd_text(day_buf[n]);
        lcd_command(0xC6);
        buf = date & 0xF0;
        date_buf[0] = (buf>>4);
        lcd_text(date_buf[0] | 0x30);
        buf = date & 0x0F;
```

```
        date_buf[1] = buf ;
        lcd_text(buf | 0x30);
        lcd_text('/');
        buf = month & 0xF0;
        date_buf[2] = (buf>>4);
        lcd_text(date_buf[2] | 0x30);
        buf = month & 0x0F;
        date_buf[3] = buf ;
        lcd_text(buf | 0x30);
        lcd_text('/');
        buf = year & 0xF0;
        date_buf[4] = (buf>>4);
        lcd_text(date_buf[4] | 0x30);
        buf = year & 0x0F;
        date_buf[5] = buf ;
        lcd_text(buf | 0x30);
        lcd_command(0xC1);
        lcd_command(0x0F);
}

void show_settime (void)
{
        unsigned char buf = 0;
        lcd_command(0x01);
        lcd_command(0x80);
        for(n=0;n<8;n++)
                lcd_text(*(s_time+n));
        sec = DS1307_rd(0x00);
        min = DS1307_rd(0x01);
        hour = DS1307_rd(0x02);
        lcd_command(0xC6);
        buf = hour & 0xF0;
        time_buf[0] = (buf>>4);
        lcd_text(time_buf[0] | 0x30);
        buf = hour & 0x0F;
        time_buf[1] = buf ;
        lcd_text(buf | 0x30);
        lcd_text(':');
        buf = min & 0xF0;
        time_buf[2] = (buf>>4);
        lcd_text(time_buf[2] | 0x30);
        buf = min & 0x0F;
        time_buf[3] = buf ;
        lcd_text(buf | 0x30);
        lcd_text(':');
        buf = sec & 0xF0;
        time_buf[4] = (buf>>4);
        lcd_text(time_buf[4] | 0x30);
        buf = sec & 0x0F;
        time_buf[5] = buf ;
        lcd_text(buf | 0x30);
        lcd_command(0xC6);
        lcd_command(0x0F);
}
```

```
/*****************************************************/
void delay_10ms (unsigned char ms10)
{
        unsigned char ms10_buf = 0;
        TF0 = 0;
        TR0 = 1;
        while(1)
        {
                while(~TF0);
                TF0 = 0;
                TH0 = 0xDC;
                TL0 = 0x00;
                ms10_buf++;
                if(ms10_buf == ms10)
                {
                        TR0 = 0;
                        break;
                }
        }
}

void welcome (void)
{
        lcd_command(0x01);              //Clear Screen.
        lcd_command(0x81);
        for(n=0;n<13;n++)
                lcd_text(*(scr1+n));
        lcd_command(0xC3);
        for(n=0;n<10;n++)
                lcd_text(*(scr2+n));
}

void show_in (void)
{
        lcd_command(0x01);
        lcd_command(0x83);
        for(n=0;n<9;n++)
                lcd_text(*(recin+n));
}

void show_out (void)
{
        lcd_command(0x01);
        lcd_command(0x83);
        for(n=0;n<10;n++)
                lcd_text(*(recout+n));
}

void show_complete (void)
{
        lcd_command(0xC3);
        for(n=0;n<9;n++)
                lcd_text(*(complete+n));
        delay_10ms(50);                 //Delay show "Complete".
}
```

```
void show_error (void)
{
        lcd_command(0x01);
        lcd_command(0x85);
        for(n=0;n<6;n++)
                lcd_text(*(error+n));
        delay_10ms(100);
}

void beep (unsigned char bt)
{
        for(n=0;n<bt;n++)
        {
                buzzer = 0;
                delay_10ms(30);
                buzzer = 1;
                delay_10ms(20);
        }
}

void read_addr (unsigned char eep)
{
        err = 0;
        i2c_start();
        if(i2c_wrdata(eep))
        {
                i2c_stop();
                err = 1;
        }
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_start();
        i2c_wrdata(eep+1);
        addr_h = i2c_rddata();
        i2c_start();
        i2c_wrdata(eep+1);
        addr_l = i2c_rddata();
        i2c_NACK();
        i2c_stop();
}

void write_addr (unsigned char eepr)
{
        i2c_start();
        i2c_wrdata(eepr);
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_wrdata(addr_h);
        i2c_wrdata(addr_l);
        i2c_stop();
}

void db_id (unsigned char ad_h,ad_l)
{
        err = 0;
        i2c_start();
        if(i2c_wrdata(EEPROM_DB))
```

```
        {
                i2c_stop();
                err = 1;
        }
        i2c_wrdata(ad_h);
        i2c_wrdata(ad_l);
        for(n=0;n<5;n++)
        {
                i2c_start();
                i2c_wrdata(EEPROM_DB+1);
                eeprom_buf[n] = i2c_rddata();
        }
        i2c_NACK();
        i2c_stop();
}

void update_addr (unsigned char plus)
{
        unsigned int addr;
        addr = addr_h;
        addr = (addr<<8) | addr_l;
        addr = addr + plus;
        addr_h = (addr & 0xFF00)>>8;// Filter High bytes & Shift 8 bits.
        addr_l = addr & 0x00FF;             // Filter Low bytes.
}

void new_rec (void)
{
        if(chk_in)
        {
                show_in();
                read_addr(EEPROM_MEM);
                if(~err)
                {
                        i2c_start();
                        i2c_wrdata(EEPROM_MEM);
                        i2c_wrdata(addr_h);
                        i2c_wrdata(addr_l);
                        i2c_wrdata(date);
                        i2c_wrdata(month);
                        i2c_wrdata(year);
                        for(n=0;n<5;n++)
                                i2c_wrdata(rd_id[n]);
                        i2c_wrdata(hour);
                        i2c_wrdata(min);
                        i2c_wrdata(0xFF);
                        i2c_wrdata(0xFF);
                        i2c_stop();
                        delay_10ms(1);
                        update_addr(12);
                        write_addr(EEPROM_MEM);
                        beep(1);
                        show_complete();
                }
                else
```

```
                {
                        err = 0;
                        show_error();
                        beep(2);
                }
        }
        if(chk_out)
        {
                show_out();
                read_addr(EEPROM_MEM);
                if(~err)
                {
                        i2c_start();
                        i2c_wrdata(EEPROM_MEM);
                        i2c_wrdata(addr_h);
                        i2c_wrdata(addr_l);
                        i2c_wrdata(date);
                        i2c_wrdata(month);
                        i2c_wrdata(year);
                        for(n=0;n<5;n++)
                                i2c_wrdata(rd_id[n]);
                        i2c_wrdata(0xFF);
                        i2c_wrdata(0xFF);
                        i2c_wrdata(hour);
                        i2c_wrdata(min);
                        i2c_stop();
                        delay_10ms(1);
                        update_addr(12);
                        write_addr(EEPROM_MEM);
                        beep(1);
                        show_complete();
                }
                else
                {
                        err = 0;
                        show_error();
                        beep(2);
                }
        }
}

void wr_eeprom (void)
{
        bit chk_sameid;
        bit chk_samedate;
        unsigned int addrl,addr_id,loopid,l;
        read_addr(EEPROM_MEM);
        if(~err)
        {
                addrl = addr_h;
                addrl = (addrl<<8) | addr_l;
                addr_id = addrl - 9;
                loopid = (addrl-2)/12;
                addr_h = (addr_id & 0xFF00)>>8;
                addr_l = addr_id & 0x00FF;
        }
        else
```

```
{
        err = 0;
        loopid = 0;
        chk_sameid = 0;
        show_error();
}

for(l=0;l<loopid;l++)
{
        i2c_start();
        i2c_wrdata(EEPROM_MEM);
        i2c_wrdata(addr_h);
        i2c_wrdata(addr_l);
        for(n=0;n<5;n++)
        {
                i2c_start();
                i2c_wrdata(EEPROM_MEM+1);
                eeprom_buf[n] = i2c_rddata();
        }
        i2c_NACK();
        i2c_stop();

        for(n=0;n<5;n++)
        {
                if(rd_id[n] == eeprom_buf[n])
                        chk_sameid = 1;
                else
                {
                        chk_sameid = 0;
                        addr_id = addr_h;
                        addr_id = (addr_id<<8) | addr_l;
                        addr_id = addr_id - 12;
                        addr_h = (addr_id & 0xFF00)>>8;
                        addr_l = addr_id & 0x00FF;
                        n = 5;
                }
        }
        if(chk_sameid)
                l = loopid;
}
if(chk_sameid)
{
        date = DS1307_rd(0x04);  // Read date for memory.
        month = DS1307_rd(0x05);// Read month for memory.
        year = DS1307_rd(0x06);  // Read year for memory.
        hour = DS1307_rd(0x02); // Read hour for memory.
        min = DS1307_rd(0x01);  // Read min for memory.

        addr_id = addr_id - 3;                          // Check Same DATE.
        addr_h = (addr_id & 0xFF00)>>8;
        addr_l = addr_id & 0x00FF;

        i2c_start();                                    // Read DATE in EEPROM.
        i2c_wrdata(EEPROM_MEM);
        i2c_wrdata(addr_h);
        i2c_wrdata(addr_l);
        for(n=0;n<3;n++)
```

```
                {
                        i2c_start();
                        i2c_wrdata(EEPROM_MEM+1);
                        eeprom_buf[n] = i2c_rddata();
                }
                i2c_NACK();
                i2c_stop();
                if((eeprom_buf[0]==date)&(eeprom_buf[1]==month)&(eeprom_buf[2]==year))
                        chk_samedate = 1;
                else
                        chk_samedate = 0;
                if(chk_samedate)
                {
                        if(chk_in)
                        {
                                show_in();
                                addr_id = addr_id+8;
                                addr_h = (addr_id & 0xFF00)>>8;
                                addr_l = addr_id & 0x00FF;
                                i2c_start();
                                i2c_wrdata(EEPROM_MEM);
                                i2c_wrdata(addr_h);
                                i2c_wrdata(addr_l);
                                i2c_wrdata(hour);
                                i2c_wrdata(min);
                                i2c_stop();
                        }
                        if(chk_out)
                        {
                                show_out();
                                addr_id = addr_id+10;
                                addr_h = (addr_id & 0xFF00)>>8;
                                addr_l = addr_id & 0x00FF;
                                i2c_start();
                                i2c_wrdata(EEPROM_MEM);
                                i2c_wrdata(addr_h);
                                i2c_wrdata(addr_l);
                                i2c_wrdata(hour);
                                i2c_wrdata(min);
                                i2c_stop();
                        }
                        beep(1);
                        show_complete();
                }
                else
                        new_rec();
        }
        else
                new_rec();

        RI = 0;
        chk_swap = 0;                   //For show Time.
}
```

```
void check_id (void)
{
        unsigned int loop,l,addrs;
        read_addr(EEPROM_DB);
        if(~err)
        {
                addrs = addr_h;
                addrs = (addrs<<8) | addr_l;
                loop = (addrs - 0x000B) / 5;
        }
        else
        {
                err = 0;
                loop = 0;
                chk_id = 0;
                show_error();
        }
        addr_h = 0x00;
        addr_l = 0x0B;
        for(l=0;l<loop;l++)                     //Loop Check Valid ID. m = time to loop.
        {
                db_id(addr_h,addr_l);
                for(n=0;n<5;n++)
                {
                        if(rd_id[n] == eeprom_buf[n])
                                chk_id = 1;
                        else
                        {
                                chk_id = 0;
                                update_addr(5);
                                n = 5;
                        }
                }
                if (chk_id)
                        l = loop;
        }
        if(chk_id)
        {
                lcd_command(0x01);              //Clear Screen.
                lcd_command(0x82);
                for(n=0;n<10;n++)
                        lcd_text(*(valid+n));
                chk_on = 1;
                if(chk_in | chk_out)
                        wr_eeprom();
                beep(1);
        }
        else
        {
                lcd_command(0x01);              //Clear Screen.
                lcd_command(0x82);
                for(n=0;n<12;n++)
                        lcd_text(*(invalid+n));
                chk_on = 0;
                beep(3);
                chk_swap = 1;                   //For show Welcome.
        }
```

```
}

void check_master (void)
{
        db_id(0x00,0x06);
        if(~err)
        {
                for(n=0;n<5;n++)
                {
                        if(rd_id[n] == eeprom_buf[n])
                                chk_id = 1;
                        else
                        {
                                chk_id = 0;
                                break;
                        }
                }
        }
        else
        {
                err = 0;
                chk_id = 0;
                show_error();
        }
        if(chk_id)
        {
                lcd_command(0x01);              //Clear Screen.
                lcd_command(0x82);
                for(n=0;n<10;n++)
                        lcd_text(*(valid+n));
                delay_10ms(50);                 // Show Correct card.
        }
        else
        {
                lcd_command(0x01);              //Clear Screen.
                lcd_command(0x82);
                for(n=0;n<12;n++)
                        lcd_text(*(invalid+n));
                delay_10ms(100);                // Show Incorrect card.
        }
}

void show_touch (void)
{
        lcd_command(0xC0);
        for(n=0;n<16;n++)
                lcd_text(*(touch+n));
}

void show_back (void)
{
        lcd_command(0xC0);
        lcd_text('<');
        lcd_text('<');
}
```

```
void show_next (void)
{
        lcd_command(0xCE);
        lcd_text('>');
        lcd_text('>');
}

void rd_serial (void)
{
        unsigned char buffer[14];
        if(RI)
        {
                TR0 = 0;            //Disable Interrupt Timer0.
                ET0 = 0;            //Disable Interrupt Address.

                for(n=0;n<14;n++)
                {
                        while(~RI);
                        RI = 0;
                        rd_buf[n] = SBUF;

                        if(rd_buf[n] > 64)
                                buffer[n]=rd_buf[n] - 0x37;     //Convert ASCII to Text A-Z.
                        if(rd_buf[n] < 58)
                                buffer[n]=rd_buf[n] - 0x30;     //Convert ASCII to Number 0-9.
                }
                lcd_command(0x01);
                lcd_command(0x80);
                for(n=1;n<11;n++)
                        lcd_text(rd_buf[n]);        //Show ID for a while.

                rd_id[0] = (buffer[1]<<4)|(buffer[2]);
                rd_id[1] = (buffer[3]<<4)|(buffer[4]);
                rd_id[2] = (buffer[5]<<4)|(buffer[6]);
                rd_id[3] = (buffer[7]<<4)|(buffer[8]);
                rd_id[4] = (buffer[9]<<4)|(buffer[10]);
                RI = 0;
                chk_int = 1;
                chk_loop = 1;
        }
}
// ------------------------------------------------------ //
//              Function MODE                             //
// ------------------------------------------------------ //
void mode_page (void)
{
        switch(m_page)
        {
                case 0 :  break;
                case 1 :  lcd_command(0x01);
                                        lcd_command(0x80);
                                        for(n=0;n<8;n++)
                                                lcd_text(*(s_date+n));
                                        show_next();
                                        break;
```

```
                case 2 :  lcd_command(0x01);
                                lcd_command(0x80);
                                for(n=0;n<8;n++)
                                        lcd_text(*(s_time+n));
                                show_back();
                                show_next();
                                break;
                case 3 :  lcd_command(0x01);
                                lcd_command(0x80);
                                for(n=0;n<8;n++)
                                        lcd_text(*(s_add+n));
                                show_back();
                                show_next();
                                break;
                case 4 :  lcd_command(0x01);
                                lcd_command(0x80);
                                for(n=0;n<11;n++)
                                        lcd_text(*(s_del+n));
                                show_back();
                                show_next();
                                break;
                case 5 :  lcd_command(0x01);
                                lcd_command(0x80);
                                for(n=0;n<15;n++)
                                        lcd_text(*(s_pwd+n));
                                show_back();
                                show_next();
                                break;
                case 6 :  lcd_command(0x01);
                                lcd_command(0x80);
                                for(n=0;n<8;n++)
                                        lcd_text(*(s_cap+n));
                                show_back();
                                show_next();
                                break;
                case 7 :  lcd_command(0x01);
                                lcd_command(0x80);
                                for(n=0;n<12;n++)
                                        lcd_text(*(s_sho+n));
                                show_back();
                                break;
        }
}

void mode_date (void)
{
        bit chk_enter = 0;

                show_setdate();
                chk_sday  = 0;
                while(~chk_enter)      // Check for press ENTER
                {                                                       // setting Date value 6
times(dd/mm/yy)
                        key = scankey();  // Scankey and Keep to push_key
                        if(key != 0xFF)    // Check value key for push?
```

```
        {
        if(key < 0x0A)
        {
                if(~chk_sday)
                {
                        day = key;
                        sw_day();
                        lcd_command(0xC1);
                        for (n=0;n<3;n++)
                                lcd_text(day_buf[n]);
                        lcd_command(0xC6);
                        chk_sday = 1;
                        n = 0;
                }
                else
                {
                        date_buf[n] = key;
                        n++;
                        key = key | 0x30;       // convert to ascii code
                        lcd_text(key);          // send value input key on LCD
                        if(n==2)                    // IF push key 2 time shift 1 time
                        lcd_command(0xC9);
                        if(n==4)                    // IF push key 4 time shift 1 time
                        lcd_command(0xCC);
                        if(n==6)
                        {
                                chk_sday = 0;
                                n = 0;
                                lcd_command(0xC1);
                        }
                }
        }
                        if(key == 0x0D)                 // Press MODE.
                                chk_enter = 1;
                        if(key == 0x0E)                 // Press ENTER.
                                {
                                DS1307_wrdate();        // Write Date data to DS1307
                                chk_enter = 1;
                                }
                        }
                }
                lcd_command(0x0C);
                chk_sday = 0;
                chk_swap = 0;
}

void mode_time (void)
{
        bit chk_enter = 0;

                show_settime();
                chk_enter = 0;
                n = 0;
                while(~chk_enter)       // Check for end insert setting
                {                               // Date value 6 time(dd/mm/yy)
                        key = scankey();  // Scankey and Keep to push_key
                        if(key != 0xFF)     // Check value key for push?
```

```
                {
                        if(key < 0x0A)
                        {
                                time_buf[n] = key;
                                n++;
                                key = key | 0x30; // convert to ascii code
                                lcd_text(key);      // send value input key on LCD
                                if(n==2)            // IF push key 2 time shift 1 time
                                        lcd_command(0xC9);
                                if(n==4)            // IF push key 4 time shift 1 time
                                        lcd_command(0xCC);
                                if(n==6)
                                {
                                        n = 0;
                                        lcd_command(0xC6);
                                }
                        }
                        if(key == 0x0D)                 // Press MODE.
                                chk_enter = 1;
                        if(key == 0x0E)                 // Press ENTER.
                        {
                                DS1307_wrtime();        // Write Date data to DS1307
                                chk_enter = 1;
                        }
                }
        }
        lcd_command(0x0C);
        chk_swap = 0;
}

void mode_add (void)
{
        unsigned int addrad,loop_ad,ad;

        show_touch();
        lcd_command(0x80);
        lcd_command(0x0F);
        RI = 0;             // Protect reading Card before Read Serial.
        time_10ms = 0;
        chk_loop = 0;
        n = 0;
        TR0 = 1;            //Enable Interrupt Timer0.***
        ET0 = 1;            //Enable Interrupt Addres.***

        do
        {
                rd_serial();
                key = scankey();
                if(key != 0xFF)
                {
                        if(n==0)
                                lcd_command(0x01);
                        rd_buf[n] = key;
                        if(rd_buf[n]<0x0A)
                                lcd_text(rd_buf[n]+0x30);
                        if(rd_buf[n]>9)
                                lcd_text(rd_buf[n]+0x37);
```

```
                n++;
                time_10ms = 0;
                if(n==10)
                {
                        rd_id[0] = (rd_buf[0]<<4)|(rd_buf[1]);
                        rd_id[1] = (rd_buf[2]<<4)|(rd_buf[3]);
                        rd_id[2] = (rd_buf[4]<<4)|(rd_buf[5]);
                        rd_id[3] = (rd_buf[6]<<4)|(rd_buf[7]);
                        rd_id[4] = (rd_buf[8]<<4)|(rd_buf[9]);
                        chk_int = 1;
                        chk_loop = 1;
                }
        }
}
while(~chk_loop);
lcd_bl = 0;                     // ON backlight LCD.
TR0 = 0;                        //Disable Interrupt Timer0.***
ET0 = 0;                        //Disable Interrupt Addres.***
lcd_command(0x0C);

if(chk_int)
{
        delay_10ms(80); // ** For seeing ID code from card.
        read_addr(EEPROM_DB);
        if(~err)
        {
                addrad = addr_h;
                addrad = (addrad<<8) | addr_l;
                loop_ad = (addrad - 0x000B) / 5;
        }
        else
        {
                err = 0;
                loop_ad = 0;
                chk_id = 1;
                show_error();
        }
        addr_h = 0x00;
        addr_l = 0x0B;
        for(ad=0;ad<loop_ad;ad++)
        {

                db_id(addr_h,addr_l);
                for(m=0;m<5;m++)
                {
                        if(rd_id[m] == eeprom_buf[m])
                                chk_id = 1;
                        else
                        {
                                chk_id = 0;
                                update_addr(5);
                                m = 5;
                        }
                }
                if(chk_id)
                        ad = loop_ad;
        }
```

```
                if(chk_id)          // New ID is same to Database ID.
                {
                        lcd_command(0x01);
                        lcd_command(0x81);
                        for(m=0;m<14;m++)
                                lcd_text(*(already+m));
                        delay_10ms(100);
                        chk_id =0;
                }
                else                        // This ID isn't in Database ID.--> ADD NEW.
                {
                        read_addr(EEPROM_DB);
                        i2c_start();
                        i2c_wrdata(EEPROM_DB);
                        i2c_wrdata(addr_h);             // Write to addr_h,addr_l
                        i2c_wrdata(addr_l);
                        for(n=0;n<5;n++)
                                i2c_wrdata(rd_id[n]);
                        i2c_stop();
                        delay_10ms(1);          // Use when the next process use its again.
                        update_addr(5);
                        write_addr(EEPROM_DB);
                        show_complete();
                        RI = 0;                 // Clear for Reder send more than 1 time.
                }
                chk_int = 0;
        }
}

void mode_del (void)
{
        bit key_p;
        unsigned char addrdh,addrdl;
        unsigned int addrd,loop_d,d;

        show_touch();
        lcd_command(0x80);
        lcd_command(0x0F);
        RI = 0;                 // Protect reading Card before Read Serial.
        time_10ms = 0;
        chk_loop = 0;
        n = 0;
        TR0 = 1;                //Enable Interrupt Timer0.***
        ET0 = 1;                //Enable Interrupt Addres.***
        do
        {
                rd_serial();
                key = scankey();
                if(key != 0xFF)
                {
                        if(n==0)
                                lcd_command(0x01);
                        rd_buf[n] = key;
                        if(rd_buf[n]<0x0A)
                                lcd_text(rd_buf[n]+0x30);
                        if(rd_buf[n]>9)
                                lcd_text(rd_buf[n]+0x37);
```

```
                n++;
                time_10ms = 0;
                if(n==10)
                {
                        rd_id[0] = (rd_buf[0]<<4)|(rd_buf[1]);
                        rd_id[1] = (rd_buf[2]<<4)|(rd_buf[3]);
                        rd_id[2] = (rd_buf[4]<<4)|(rd_buf[5]);
                        rd_id[3] = (rd_buf[6]<<4)|(rd_buf[7]);
                        rd_id[4] = (rd_buf[8]<<4)|(rd_buf[9]);
                        chk_int = 1;
                        chk_loop = 1;
                }
        }
}
while(~chk_loop);
lcd_bl = 0;                     // ON backlight LCD.
TR0 = 0;                        //Disable Interrupt Timer0.***
ET0 = 0;                        //Disable Interrupt Addres.***
lcd_command(0x0C);
if(chk_int)
{
        chk_int = 0;
        delay_10ms(100);                        // Delay Show ID Card.
        read_addr(EEPROM_DB);
        if(~err)
        {
                addrd = addr_h;
                addrd = (addrd<<8) | addr_l;
                loop_d = (addrd - 0x000B) / 5;
        }
        else
        {
                err = 0;
                loop_d = 0;
                show_error();
                chk_id = 0;
        }
        addr_h = 0x00;
        addr_l = 0x0B;
        for(d=0;d<loop_d;d++)
        {

                db_id(addr_h,addr_l);
                for(m=0;m<5;m++)
                {
                        if(rd_id[m] == eeprom_buf[m])
                                chk_id = 1;
                        else
                        {
                                chk_id = 0;
                                update_addr(5);
                                m = 5;
                        }
                }
                if(chk_id)
                        d = loop_d;
        }
```

```
if(chk_id)
{
        chk_id = 0;
        addrdh = addr_h;
        addrdl = addr_l;
        lcd_command(0x01);
        lcd_command(0x81);
        for(m=0;m<14;m++)
                lcd_text(*(con_del+m));
        lcd_command(0xC3);

        rd_buf[0] = (rd_id[0] & 0xF0)>>4;
        rd_buf[1] = rd_id[0] & 0x0F;
        rd_buf[2] = (rd_id[1] & 0xF0)>>4;
        rd_buf[3] = rd_id[1] & 0x0F;
        rd_buf[4] = (rd_id[2] & 0xF0)>>4;
        rd_buf[5] = rd_id[2] & 0x0F;
        rd_buf[6] = (rd_id[3] & 0xF0)>>4;
        rd_buf[7] = rd_id[3] & 0x0F;
        rd_buf[8] = (rd_id[4] & 0xF0)>>4;
        rd_buf[9] = rd_id[4] & 0x0F;

        for(m=0;m<10;m++)               // Show "Confirm Delete (ID)".
        {
            if(rd_buf[m]<0x0A)
                lcd_text(rd_buf[m]+0x30);
            if(rd_buf[m]>0x09)
                lcd_text(rd_buf[m]+0x37);
        }
        key_p = 1;
        while(key_p)
        {
            key = scankey();
            if(key == 0x0E)
            {
                read_addr(EEPROM_DB);
                addrd = addr_h;
                addrd = (addrd<<8) | addr_l;
                addrd = addrd-5;
                addr_h = (addrd & 0xFF00)>>8;
                addr_l = addrd & 0x00FF;
                db_id(addr_h,addr_l);       // Read ID fromm
addr_h,addr_l.
                i2c_start();
                i2c_wrdata(EEPROM_DB);
                i2c_wrdata(addrdh);         // Write to address DELETE.
                i2c_wrdata(addrdl);
                for(m=0;m<5;m++)
                        i2c_wrdata(eeprom_buf[m]);
                i2c_stop();
                delay_10ms(1);// Use when the next process use its again.

                write_addr(EEPROM_DB);
                lcd_command(0x01);      // Show "Delete User"
                lcd_command(0x82);      // "Complete!".
                for(m=0;m<11;m++)
                        lcd_text(*(s_del+m));
```

```
                                        show_complete();
                                        key_p = 0;
                                }
                                if(key == 0x0D)
                                {
                                        lcd_command(0x01);
                                        lcd_command(0x83);
                                        for(m=0;m<10;m++)
                                                lcd_text(*(n_del+m));
                                        delay_10ms(150);
                                        key_p = 0;
                                }
                        }
                }
                else
                {
                        lcd_command(0x01);
                        lcd_command(0x82);
                        for(m=0;m<11;m++)
                                lcd_text(*(nfound+m));
                        delay_10ms(150);
                }
        }
}

void mode_pwd (void)
{
        bit i = 0;

        lcd_command(0x01);
        lcd_command(0x80);
        for(n=0;n<14;n++)
                lcd_text(*(en_pass+n));

        db_id(0x00,0x02);                               //Read Password From EEPROM
        if(~err)
        {
                pwd_in[0] = (eeprom_buf[0] & 0xF0)>>4;
                pwd_in[1] = (eeprom_buf[0] & 0x0F);
                pwd_in[2] = (eeprom_buf[1] & 0xF0)>>4;
                pwd_in[3] = (eeprom_buf[1] & 0x0F);
                pwd_in[4] = (eeprom_buf[2] & 0xF0)>>4;
                pwd_in[5] = (eeprom_buf[2] & 0x0F);
                pwd_in[6] = (eeprom_buf[3] & 0xF0)>>4;
                pwd_in[7] = (eeprom_buf[3] & 0x0F);
//-------------------------------------------------------//
                lcd_command(0xC0);
                for(n=0;n<8;n++)
                        lcd_text(pwd_in[n] + 0x30);
        }
        else
        {
                err = 0;
                show_error();
                i = 1;
        }
```

```
lcd_command(0xC0);
lcd_command(0x0F);
n = 0;
while(~i)
{
        key = scankey();
        if(key != 0xFF)
        {
                if(key < 0x0A)
                {
                        if(n<8)
                        {
                                pwd_in[n] = key;
                                lcd_text(pwd_in[n] | 0x30);//Convert Num to ASCII.
                                n++;
                        }
                }
                if(key == 0x0D)             //Press MODE.
                {
                        lcd_command(0x0C);
                        i = 1;
                }
                if(key == 0x0E)             //Press ENTER
                {
//------------ Write Password To EEPROM !.----------//
                        eeprom_buf[0] = (pwd_in[0]<<4)|(pwd_in[1]);
                        eeprom_buf[1] = (pwd_in[2]<<4)|(pwd_in[3]);
                        eeprom_buf[2] = (pwd_in[4]<<4)|(pwd_in[5]);
                        eeprom_buf[3] = (pwd_in[6]<<4)|(pwd_in[7]);
                        i2c_start();
                        i2c_wrdata(EEPROM_DB);
                        i2c_wrdata(0x00);
                        i2c_wrdata(0x02);
                        for(m=0;m<4;m++)
                                i2c_wrdata(eeprom_buf[m]);
                        i2c_stop();
                        lcd_command(0x01);
                        lcd_command(0x0C);              //For non - cirsor.
                                i = 1;
                }
                if(key == 0x0A)             // <-.
                {
                        if(n>0)
                        {
                                n--;
                                lcd_command(0xC0+n);
                        }
                }
                if(key == 0x0B)             //->.
                {
                        if(n<8)
                        {
                                n++;
                                lcd_command(0xC0+n);
                        }
                }
```

```
                }
        }
}

void mode_cap (void)
{
                unsigned int addrc;

                lcd_command(0x01);
                lcd_command(0x80);
                lcd_text('I');
                lcd_text('D');
                lcd_command(0x85);
                for(n=0;n<10;n++)
                        lcd_text(*(u_f+n));
                read_addr(EEPROM_DB);
                if(~err)
                {
                        addrc = addr_h;
                        addrc = (addrc<<8) | addr_l;
                        addrc = (addrc - 0x000B)/5;
                        rd_buf[0] = addrc/1000;
                        rd_buf[1] = (addrc/100)%10;
                        rd_buf[2] = (addrc/10)%10;
                        rd_buf[3] = addrc%10;
                }
                else
                {
                        rd_buf[0] = 0;
                        rd_buf[1] = 0;
                        rd_buf[2] = 0;
                        rd_buf[3] = 0;
                }
                lcd_command(0xC5);
                for(n=0;n<4;n++)
                        lcd_text(rd_buf[n]+0x30);
                if(~err)
                {
                        addrc = 0x0663 - addrc;          // Total 0x0663 = 1635 times.
                        rd_buf[0] = addrc/1000;
                        rd_buf[1] = (addrc/100)%10;
                        rd_buf[2] = (addrc/10)%10;
                        rd_buf[3] = addrc%10;
                }
                else
                {
                        rd_buf[0] = 0;
                        rd_buf[1] = 0;
                        rd_buf[2] = 0;
                        rd_buf[3] = 0;
                        err = 0;
                }
                lcd_command(0xCB);
                for(n=0;n<4;n++)
                        lcd_text(rd_buf[n]+0x30);
                delay_10ms(250);                         // Delay Show Database Capacity.
                delay_10ms(250);
```

```
// ------------------------------------ //

                lcd_command(0x01);
                lcd_command(0x80);
                lcd_text('M');
                lcd_text('E');
                lcd_text('M');
                lcd_command(0x85);
                for(n=0;n<10;n++)
                        lcd_text(*(u_f+n));
                read_addr(EEPROM_MEM);
                if(~err)
                {
                        addrc = addr_h;
                        addrc = (addrc<<8) | addr_l;
                        addrc = (addrc - 0x0002)/12;
                        rd_buf[0] = addrc/1000;
                        rd_buf[1] = (addrc/100)%10;
                        rd_buf[2] = (addrc/10)%10;
                        rd_buf[3] = addrc%10;
                }
                else
                {
                        rd_buf[0] = 0;
                        rd_buf[1] = 0;
                        rd_buf[2] = 0;
                        rd_buf[3] = 0;
                }
                lcd_command(0xC5);
                for(n=0;n<4;n++)
                        lcd_text(rd_buf[n]+0x30);
                if(~err)
                {
                        addrc = 0x1554 - addrc;          // Total 0x1554 = 5460 times.
                        rd_buf[0] = addrc/1000;
                        rd_buf[1] = (addrc/100)%10;
                        rd_buf[2] = (addrc/10)%10;
                        rd_buf[3] = addrc%10;
                }
                else
                {
                        rd_buf[0] = 0;
                        rd_buf[1] = 0;
                        rd_buf[2] = 0;
                        rd_buf[3] = 0;
                        err = 0;
                }
                lcd_command(0xCB);
                for(n=0;n<4;n++)
                        lcd_text(rd_buf[n]+0x30);
                delay_10ms(250);                         // Delay Show Database Capacity.
                delay_10ms(250);
}
```

```
void mode_sho (void)
{
        show_touch();
        RI = 0;
        time_10ms = 0;
        chk_loop = 0;
        TR0 = 1;            //Enable Interrupt Timer0.***
        ET0 = 1;            //Enable Interrupt Addres.***
        do
        {
                rd_serial();
                key = scankey();
                if (key == 0x0D)
                        chk_loop = 1;
        }
        while(~chk_loop);
        lcd_bl = 0;                 // ON backlight LCD.
        TR0 = 0;                    //Disable Interrupt Timer0.
        ET0 = 0;                    //Disable Interrupt Addres.
        if(chk_int)
        {
                delay_10ms(250);
                delay_10ms(250);
                delay_10ms(250);
                delay_10ms(250);
                chk_int = 0;
        }
}

void enter_mode (void)
{
        if(m_page == 1)             // MODE SET DATE.
                mode_date();

        if(m_page == 2)             // MODE SET TIME.
                mode_time();

        if(m_page == 3)             // MODE ADD USER.
                mode_add();

        if(m_page == 4)             // MODE DELETE USER.
                mode_del();

        if(m_page == 5)             // MODE CHANGE PASSWORD.
                mode_pwd();

        if(m_page == 6)             // MODE CAPACITY.
                mode_cap();
        if(m_page == 7)             // MODE SHOW CARD ID.
                mode_sho();
}

void timer_out (void) interrupt 1
{
        TF0 = 0;
        time_10ms++;
        count_10ms++;
```

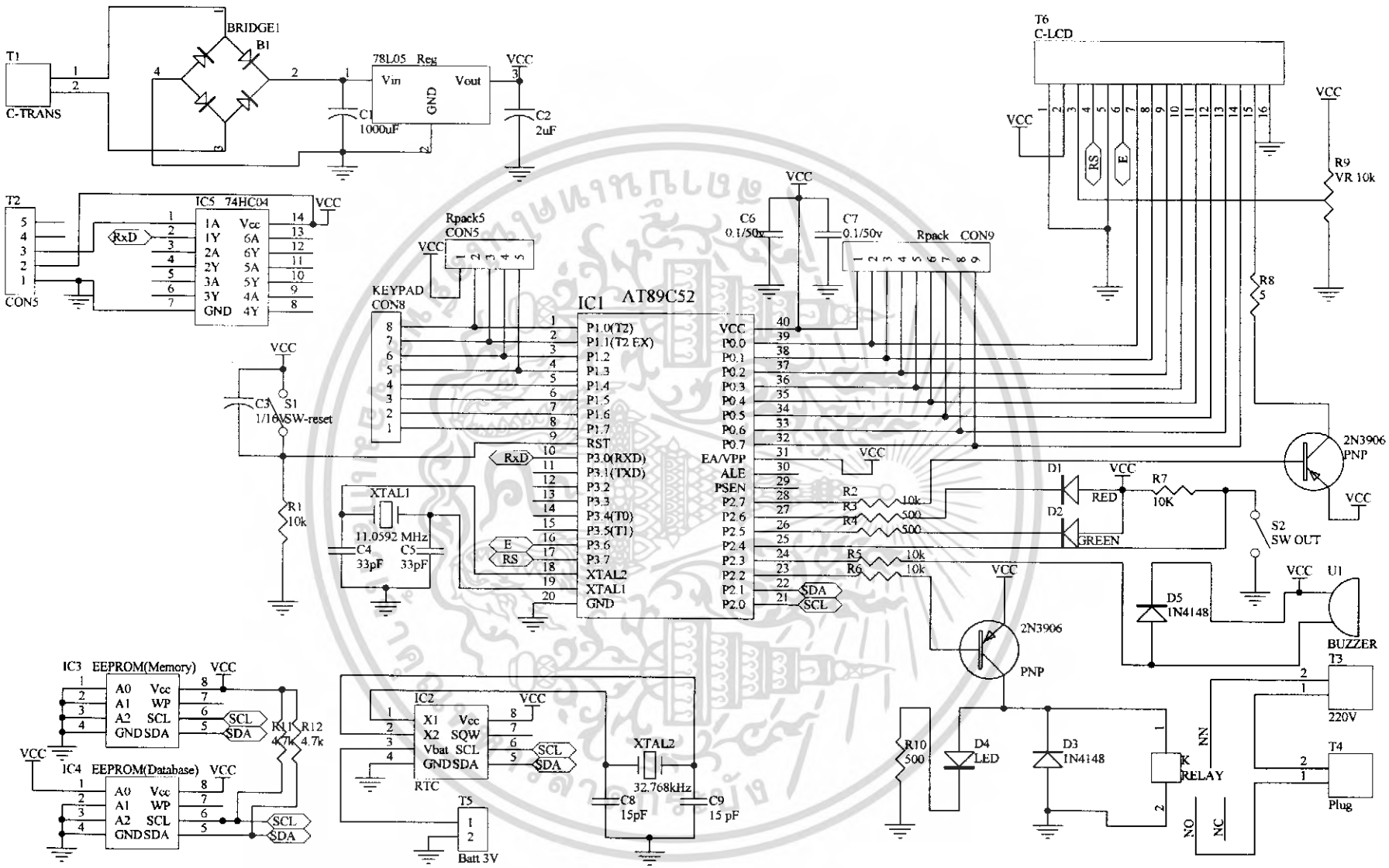
```
        TH0 = 0xDC;
        TL0 = 0x00;
        if(time_10ms == DELAY_TIME)
        {
                time_10ms = 0;
                chk_loop = 1;
                if(~lcd_bl)
                        lcd_bl = 1;
        }
        if(count_10ms == 100)
        {
                count_10ms = 0;
                chk_time = 1;
        }
}

//*****************************************************//
//                      Main Program                   //
//*****************************************************//
void main(void)
{
        TMOD = 0x21;
        SCON = 0x50;
        TH1 = 0xFD;
        TL1 = 0xFD;
        TH0 = 0xDC;
        TL0 = 0x00;
        EA = 1;
        ET0 = 1;        //      Set to Interrupt1 Address Timer0 All time.
        TR0 = 1;        //
        TR1 = 1;        //When use Timer1 (Serial Input) must set to 1.

        lcd_bl = 1;
        led_r = 0;
        led_g = 1;
        sw_out = 1;
        buzzer = 1;
        relay = 1;
//---------------------
        lcd_init();
        welcome();
        while(1)
        {
                if(chk_loop)
                {
                        chk_loop = 0;
                        if(chk_on)
                        {
                                led_g = 0;
                                led_r = 1;
                                relay = 0;
                                chk_on = 0;
                        }
                        else
                        {
                                led_g = 1;
                                led_r = 0;
```

```
                relay = 1;
        }
        chk_swap = ~chk_swap;
        if(chk_swap)
        {
                show_date();
                show_time();
                count_10ms = 0;
        }
        else
                welcome();
        time_10ms = 0;
        ET0 = 1;                    // Enable after use serial;
        TR0 = 1;                    //
}
if(chk_swap)                        // For Show Update Time.
{
        if(chk_time)
        {
                chk_time = 0;
                show_time();
        }
}

key = scankey();
if(key != 0xFF)
{
        if(key == 0x0C)             //Record IN.
        {
                lcd_bl = 0;                 // ON backlight LCD.
                RI = 0;                     // Protect reading Card before.
                show_in();
                show_touch();
                time_10ms = 0;
                chk_loop = 0;
                do
                {
                        rd_serial();
                        key = scankey();
                        if(key == 0x0D)             // Press MODE = Out of loop.
                        {
                                chk_loop = 1;
                                key = 0xFF;
                        }
                }
                while(~chk_loop);

                if(chk_int)
                        chk_in = 1;
        }
        if(key == 0x0D)                             //MODE.
        {
                lcd_bl = 0;                         // ON backlight LCD.
                RI = 0;                             // Protect reading Card before.

                lcd_command(0x01);
                lcd_command(0x80);
```

```
for(n=0;n<11;n++)
        lcd_text(*(master+n));
show_touch();
time_10ms = 0;
chk_loop = 0;
do
{
        rd_serial();
        key = scankey();
        if(key == 0x0D)             // Press MODE = Out of loop.
                chk_loop = 1;
}
while(~chk_loop);

if(chk_int)
{
        check_master();
        chk_int = 0;
}
if(chk_id)
{
        chk_id = 0;
        m_page = 1;
        mode_page();

        while(m_page != 0)
        {
                key = scankey();
                if(key != 0xFF)
                {
                        if(key == 0x0D)  // Press key MODE.
                        {
                                RI = 0;// repair when read card.
                                m_page = 0;
                        }
                        if(key == 0x0E)  // Press key ENTER.
                                enter_mode();
                        if(key == 0x0A)  // Press key <-.
                        {
                                m_page--;
                                if(m_page == 0)
                                m_page = 7;
                        }
                        if(key == 0x0B)  // Press key ->.
                        {
                                m_page++;
                                if(m_page == 8)
                                m_page = 1;
                        }
                        mode_page();
                }
        }
}
if(key == 0x0E)             //ENTER.
{
        lcd_bl = 0;                     // ON backlight LCD.
```

```
lcd_command(0x01);
lcd_command(0x80);
for(n=0;n<14;n++)
        lcd_text(*(en_pass+n));
lcd_command(0xC0);
lcd_command(0x0F);      // set Cursor.

db_id(0x00,0x02);       //Read Password From EEPROM.
if(~err)
{
        time_10ms = 0;
        chk_loop = 0;
}
else
{
        err = 0;
        show_error();
        chk_loop = 1;
}
        n = 0;
do
{
        key = scankey();

        if(key < 0x0A)
        {
        rd_buf[n] = key;
        lcd_text(rd_buf[n] + 0x30);//Convert Num to ASCII.
                n++;
                time_10ms = 0;
        }
        if(key == 0x0A)         // <-.
        {
                time_10ms = 0;
                if(n>0)
                {
                        n--;
                        lcd_command(0xC0+n);
                        lcd_text(' ');
                        lcd_command(0xC0+n);
                }
        }
        if(key == 0x0D)         // Press MODE = Out of loop.

                chk_loop = 1;
        if(n == 8)
        {
                pwd_in[0] = (rd_buf[0]<<4) | rd_buf[1];
                pwd_in[1] = (rd_buf[2]<<4) | rd_buf[3];
                pwd_in[2] = (rd_buf[4]<<4) | rd_buf[5];
                pwd_in[3] = (rd_buf[6]<<4) | rd_buf[7];

                for(m=0;m<4;m++)
                {
                        if(pwd_in[m] == eeprom_buf[m])
                                chk_on = 1;
                        else
```

```
                                        {
                                                chk_on = 0;
                                                m = 4;
                                        }
                                }
                                chk_loop = 1;

                        }
                }
                while(~chk_loop);
                RI = 0;                         // Protect before read Card.
                lcd_command(0x0C);              // No Cursor.
        }
        if(key == 0x0F)         //Record OUT.
        {
                lcd_bl = 0;                     // ON backlight LCD.
                RI = 0;                         // Protect before read Card .
                show_out();
                show_touch();

                time_10ms = 0;
                chk_loop = 0;
                do
                {
                        rd_serial();
                        key = scankey();
                        if(key == 0x0D)  // Press MODE = Out of loop.
                                chk_loop = 1;
                }
                while(~chk_loop);

                if(chk_int)
                        chk_out = 1;
        }
    }
    rd_serial();                //Touch Card For Passing Only!

    if(chk_int)                 //Check Card for Open the Door.
    {
            lcd_bl = 0;
            delay_10ms(40);  // For seeing ID code from card.
            check_id();
            chk_int = 0;
            chk_id = 0;
            chk_in = 0;
            chk_out = 0;
            RI = 0;             // Clear for Reder send more than 1 time.
    }
    while(~sw_out)   // Check Press key "Switch Out".
    {
            lcd_bl = 0;                             // ON backlight LCD.
            chk_loop = 1;
            chk_on = 1;
    }
}

}
```



Title		
Size A4	Number	Revision
Date: 19-Mar-2006		Sheet of
File: D:\Programmer\Keypad.ddb		Drawn By

CODE ของเครื่อง EEPROM DATABASE/MEMORY READER

```
#include <AT892051.H>
#include <i2cs.h>

#define EEPROM_MEM 0xA0          // EEPROM 512 Kbits.
#define EEPROM_DB 0xA2           // EEPROM 64 Kbits. Old Board use 0xA8.

sbit LED = P1^2;
bit err;
unsigned char addr_l,addr_h,rd_buf[2],n;

void read_addr (unsigned char eep)
{
        err = 0;
        i2c_start();
        if(i2c_wrdata(eep))
        {
                i2c_stop();
                err = 1;
        }
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_start();
        i2c_wrdata(eep+1);
        addr_h = i2c_rddata();
        i2c_start();
        i2c_wrdata(eep+1);
        addr_l = i2c_rddata();
        i2c_NACK();
        i2c_stop();
}

void send_OK (void)
{
        SBUF = 'O';
        while(~TI);
        TI = 0;
        SBUF = 'K';
        while(~TI);
        TI = 0;
}

void send_ER (void)
{
        SBUF = 'E';
        while(~TI);
        TI = 0;
        SBUF = 'R';
        while(~TI);
        TI = 0;
}

void send_DBcap (void)
{
        unsigned char send_buf[4];
        unsigned int saddr;
        saddr = addr_h;
```

```
        saddr = (saddr<<8) | addr_l;
        saddr = (saddr - 0x000B)/5;                    // Send Value Used Address.
        send_buf[0] = saddr/1000;
        send_buf[1] = (saddr/100)%10;
        send_buf[2] = (saddr/10)%10;
        send_buf[3] = saddr%10;
        for(n=0;n<4;n++)
                send_buf[n] = send_buf[n]+0x30;
        for(n=0;n<4;n++)
        {
                SBUF = send_buf[n];
                while(~TI);
                TI = 0;
        }
}

void send_MEMcap (void)
{
        unsigned char send_buf[4];
        unsigned int saddr;
        saddr = addr_h;
        saddr = (saddr<<8) | addr_l;
        saddr = (saddr - 0x0002)/12;                   // Send Value Used Address.
        send_buf[0] = saddr/1000;
        send_buf[1] = (saddr/100)%10;
        send_buf[2] = (saddr/10)%10;
        send_buf[3] = saddr%10;
        for(n=0;n<4;n++)
                send_buf[n] = send_buf[n]+0x30;
        for(n=0;n<4;n++)
        {
                SBUF = send_buf[n];
                while(~TI);
                TI = 0;
        }
}

void send_PM (void)
{
        unsigned char pm_buf,p;
        i2c_start();
        i2c_wrdata(EEPROM_DB);
        i2c_wrdata(0x00);
        i2c_wrdata(0x02);
        for(p=0;p<9;p++)
        {
                i2c_start();
                i2c_wrdata(EEPROM_DB+1);
                pm_buf = i2c_rddata();
                rd_buf[0] = (pm_buf & 0xF0)>>4;        // Seperate to 2 byte.
                rd_buf[1] = pm_buf & 0x0F;
                for(n=0;n<2;n++)
                {
                        if(rd_buf[n] > 0x09)           // Convert to ASCII.
                                rd_buf[n] = rd_buf[n]+0x37;
                        if(rd_buf[n] < 0x0A)
                                rd_buf[n] = rd_buf[n]+0x30;
```

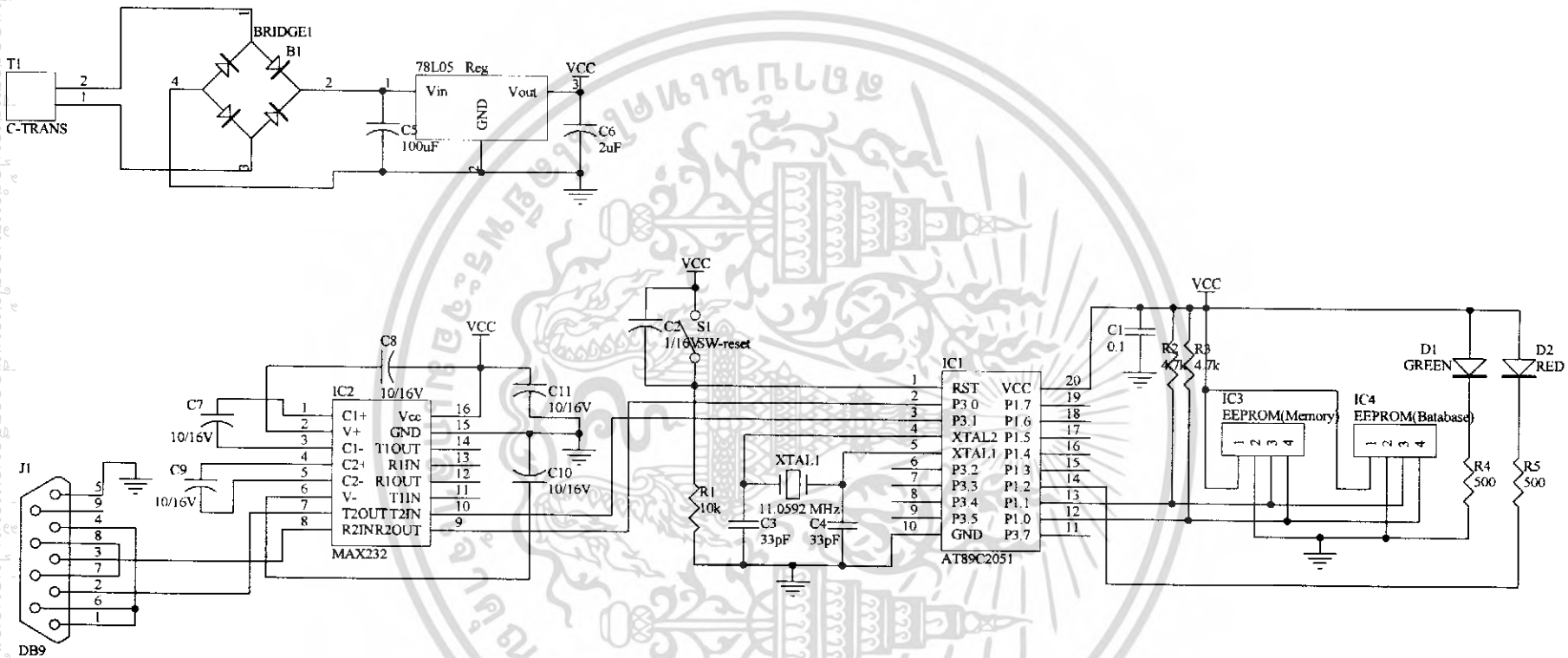
```
                }
                SBUF = rd_buf[0];                                   // Send First byte.
                while(~TI);
                TI = 0;
                SBUF = rd_buf[1];                                   // Send Second byte.
                while(~TI);
                TI = 0;
        }
        i2c_NACK();
        i2c_stop();
}

void send_DB (void)
{
        unsigned char db_buf;
        unsigned int addrd,d;
        LED = 0;
        read_addr(EEPROM_DB);                                       // Calculate Loop for
send DB.
        if(~err)
        {
                send_OK();
                send_DBcap();                                       // Send Capacity.
                send_PM();
                addrd = addr_h;
                addrd = (addrd<<8) | addr_l;
                addrd = addrd - 0x000B;

                i2c_start();
                i2c_wrdata(EEPROM_DB);
                i2c_wrdata(0x00);
                i2c_wrdata(0x0B);
                for(d=0;d<addrd;d++)
                {
                        i2c_start();
                        i2c_wrdata(EEPROM_DB+1);
                        db_buf = i2c_rddata();
                        rd_buf[0] = (db_buf & 0xF0)>>4;              // Seperate to 2 byte.
                        rd_buf[1] = db_buf & 0x0F;

                        for(n=0;n<2;n++)
                        {
                                if(rd_buf[n] > 0x09)                // Convert to ASCII.
                                        rd_buf[n] = rd_buf[n]+0x37;
                                if(rd_buf[n] < 0x0A)
                                        rd_buf[n] = rd_buf[n]+0x30;
                        }
                        SBUF = rd_buf[0];                                   // Send First
byte.
                        while(~TI);
                        TI = 0;
                        SBUF = rd_buf[1];                                   // Send
Second byte.
                        while(~TI);
                        TI = 0;
                }
```

```
                i2c_NACK();
                i2c_stop();
        }
        else
        {
                err = 0;
                send_ER();
        }
        LED = 1;
}

void send_MEM (void)
{
        unsigned char mem_buf;
        unsigned int addrm,m;
        LED = 0;
        read_addr(EEPROM_MEM);                  // Calculate Loop for send MEM.
        if(~err)
        {
                send_OK();
                send_MEMcap();                  // Send Capacity.
                addrm = addr_h;
                addrm = (addrm<<8) | addr_l;
                addrm = addrm - 0x0002;

                i2c_start();
                i2c_wrdata(EEPROM_MEM);
                i2c_wrdata(0x00);
                i2c_wrdata(0x02);
                for(m=0;m<addrm;m++)
                {
                        i2c_start();
                        i2c_wrdata(EEPROM_MEM+1);
                        mem_buf = i2c_rddata();

                        rd_buf[0] = (mem_buf & 0xF0)>>4;        // Seperate to 2 byte.
                        rd_buf[1] = mem_buf & 0x0F;
                        for(n=0;n<2;n++)
                        {
                                if(rd_buf[n] > 0x09)
                                        rd_buf[n] = rd_buf[n]+0x37;      // Convert to ASCII.
                                if(rd_buf[n] < 0x0A)
                                        rd_buf[n] = rd_buf[n]+0x30;
                        }
                        SBUF = rd_buf[0];                               // Send First byte.
                        while(~TI);
                        TI = 0;
                        SBUF = rd_buf[1];                               // Send Second byte.
                        while(~TI);
                        TI = 0;
                }
                i2c_NACK();
                i2c_stop();
        }
        else
```

```
        {
                err = 0;
                send_ER();
        }
        LED = 1;
}

void wr_WP (void)
{
        unsigned char wr_buf[8],wp[4];
        LED = 0;

        for(n=0;n<8;n++)
        {
                while(~RI);
                RI = 0;
                wr_buf[n] = SBUF;
                wr_buf[n] = wr_buf[n]-0x30;        // Convert ASCII to Hex.(Number)
        }

        wp[0] = (wr_buf[0]<<4) | (wr_buf[1]);      // Combine 2 to 1 byte.
        wp[1] = (wr_buf[2]<<4) | (wr_buf[3]);
        wp[2] = (wr_buf[4]<<4) | (wr_buf[5]);
        wp[3] = (wr_buf[6]<<4) | (wr_buf[7]);

        i2c_start();
        i2c_wrdata(EEPROM_DB);
        i2c_wrdata(0x00);
        i2c_wrdata(0x02);
        for(n=0;n<4;n++)
                i2c_wrdata(wp[n]);
        i2c_stop();
        LED = 1;
}

void wr_WM (void)
{
        unsigned char wr_buf[10],wm[5];
        LED = 0;

        for(n=0;n<10;n++)
        {
                while(~RI);
                RI = 0;
                wr_buf[n] = SBUF;
                if(wr_buf[n] < 0x3A)
                        wr_buf[n] = wr_buf[n]-0x30;
                if(wr_buf[n] > 0x40)                    // Convert ASCII to Hex.
                        wr_buf[n] = wr_buf[n]-0x37;
        }

        wm[0] = (wr_buf[0]<<4) | (wr_buf[1]);      // Combine 2 to 1 byte.
        wm[1] = (wr_buf[2]<<4) | (wr_buf[3]);
        wm[2] = (wr_buf[4]<<4) | (wr_buf[5]);
        wm[3] = (wr_buf[6]<<4) | (wr_buf[7]);
        wm[4] = (wr_buf[8]<<4) | (wr_buf[9]);
```

```
        i2c_start();
        i2c_wrdata(EEPROM_DB);
        i2c_wrdata(0x00);
        i2c_wrdata(0x06);
        for(n=0;n<5;n++)
                i2c_wrdata(wm[n]);
        i2c_stop();
        LED = 1;
}

void wr_RD (void)
{
        LED = 0;
        i2c_start();
        i2c_wrdata(EEPROM_DB);
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_wrdata(0x0B);
        i2c_stop();
        LED = 1;
}

void wr_RM (void)
{
        LED = 0;
        i2c_start();
        i2c_wrdata(EEPROM_MEM);
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_wrdata(0x00);
        i2c_wrdata(0x02);
        i2c_stop();
        LED = 1;
}

void main (void)
{
        TMOD = 0x21;
        SCON = 0x50;
        TH1 = 0xFD;
        TL1 = 0xFD;
        TI = 0;
        RI = 0;
        TR1 = 1;
        LED = 1;

        while(1)
        {
                if(RI)
                {
                        for(n=0;n<2;n++)
                        {
                                while(~RI);
                                RI = 0;
                                rd_buf[n] = SBUF;
```

```
                }
            if((rd_buf[0]=='D') & (rd_buf[1]=='B'))       // Read ID from Database send to COM.
                send_DB();

            if((rd_buf[0]=='M') & (rd_buf[1]=='M'))// Read DATA from Memory send to COM
                send_MEM();

            if((rd_buf[0]=='W') & (rd_buf[1]=='M'))      // Write MASTER to EEPROM_DB.
                wr_WM();

            if((rd_buf[0]=='W') & (rd_buf[1]=='P'))      // Write PASSWORD to EEPROM_DB.
                wr_WP();

            if((rd_buf[0]=='R') & (rd_buf[1]=='D'))// Write DEFAULT addr to EEPROM_DB.
                wr_RD();

            if((rd_buf[0]=='R') & (rd_buf[1]=='M'))// Write DEFAULTaddr to EEPROM_MEM.
                wr_RM();
            }
        }
}
```



Title		
Size: A4	Number	Revision
Date: 13-Mar-2006		Sheet of
File: D:\Programmer\Keypad.ddb		Drawn By

Library of Time Recorder Using RFID

I²C

```
/*Filename        : i2c.h*/
#include<intrins.h>
sbit SDA = P2^1;
sbit SCL = P2^0;

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

```
        i2c_delay();
        data_bit = SDA;
        SCL = 0;
        i2c_delay();
        return(data_bit);                    // if send_bit = 0 i2c OK!
}
unsigned char i2c_rddata(void)
{
        bit rd_bit;
        unsigned char i,dat;
        dat = 0x00;
        for(i=0;i<8;i++)
        {
                i2c_delay();
                SCL = 1;
                i2c_delay();
                rd_bit = SDA;
                dat = dat<<1;
                dat = dat | rd_bit;
                SCL = 0;
        }
        SDA = 1;
        i2c_delay();
        i2c_clk();
        SCL = 1;
        return(dat);
}
void i2c_NACK()
{
        SDA = 1;
        i2c_delay();
        i2c_clk();
        SCL = 1;
}
```

Keypad 4x4

```
/*Filename        : scankey4x4.h*/
sbit c1=P1^5;     //P1^7
sbit c2=P1^6;     //P1^6
sbit c3=P1^7;     //P1^5
sbit c4=P1^4;     //P1^4
sbit r1=P1^3;     //P1^0
sbit r2=P1^2;     //P1^1
sbit r3=P1^1;     //P1^2
sbit r4=P1^0;     //P1^3

void delay_db(int time)
{
        do
        {
        time--;
        }while(time>0);
}
unsigned char scankey(void)
{
        unsigned char ret=0xFF;
```

```
c1=0;
if(r1==0)                       // 1
{
        delay_db(3000);
        ret=0x01;
        while(~r1);
}
if(r2==0)                       // 4
{
        delay_db(3000);
        ret=0x04;
        while(~r2);
}
if(r3==0)                       // 7
{
        delay_db(3000);
        ret=0x07;
        while(~r3);
}
if(r4==0)                       // * = 0x0A
{
        delay_db(3000);
        ret=0x0A;
        while(~r4);
}
c1=1;

c2=0;
if(r1==0)                       // 2
{
        delay_db(3000);
        ret=0x02;
        while(~r1);
}
if(r2==0)                       // 5
{
        delay_db(3000);
        ret=0x05;
        while(~r2);
}
if(r3==0)                       // 8
{
        delay_db(3000);
        ret=0x08;
        while(~r3);
}
if(r4==0)                       // 0
{
        delay_db(3000);
        ret=0x00;
        while(~r4);
}
c2=1;

c3=0;
if(r1==0)                       // 3
{
```

```
            delay_db(3000);
            ret=0x03;
            while(~r1);
        }
        if(r2==0)                       // 6
        {
            delay_db(3000);
            ret=0x06;
            while(~r2);
        }
        if(r3==0)                       // 9
        {
            delay_db(3000);
            ret=0x09;
            while(~r3);
        }
        if(r4==0)                       // # = 0x0B
        {
            delay_db(3000);
            ret=0x0B;
            while(~r4);
        }
        c3=1;

        c4=0;
        if(r1==0)                       // IN = 0x0C
        {
            delay_db(3000);
            ret=0x0C;
            while(~r1);
        }
        if(r2==0)                       // MODE = 0x0D
        {
            delay_db(3000);
            ret=0x0D;
            while(~r2);
        }
        if(r3==0)                       // SET = 0x0E
        {
            delay_db(3000);
            ret=0x0E;
            while(~r3);
        }
        if(r4==0)                       // OUT = 0x0F
        {
            delay_db(3000);
            ret=0x0F;
            while(~r4);
        }
        c4=1;
        return(ret);
}
```

LCD 16x2

```
/*Filename        : lcd16x2.h*/
sbit e =P3^7;
sbit rs=P3^6;
void delay_lcd(int tick)
{
        int i,j;
        for (i=0;i<tick;i++)
        for (j=0;j<50;j++);          // Oldtime = 250
}
void lcd_command(unsigned char com)
{
        rs = 0;
        e = 1;
        P0 = com;
        delay_lcd(5);
        e = 0;
        delay_lcd(5);
}
void lcd_text(unsigned char text)
{
        rs = 1;
        e =1;
        P0 = text;
        delay_lcd(5);
        e = 0;
        delay_lcd(5);
}
void lcd_init()
{
        delay_lcd(250);
        delay_lcd(250);
        lcd_command(0x38);
        lcd_command(0x0C);
        lcd_command(0x01);
}
```

Library of EEPROM Database/Memory Reader

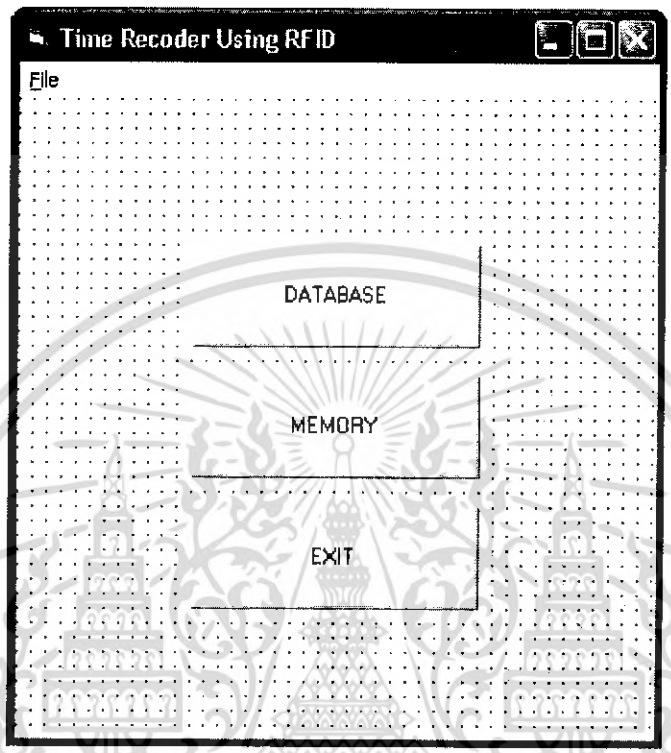
I²C

```
/*Filename        : i2cs.h*/
#include<intrins.h>
sbit SDA = P1^0;
sbit SCL = P1^1;
void i2c_delay(void)
{
        unsigned char i;
        for(i=0;i<20;i++)
        _nop_();
}
void i2c_clk(void)
{
        i2c_delay();
        SCL = 1;
        i2c_delay();
        SCL = 0;
}
void i2c_start(void)
{
        if(SCL)
        SCL = 0;
        SDA = 1;
        SCL = 1;
        i2c_delay();
        SDA = 0;
        i2c_delay();
        SCL = 0;
}
void i2c_stop(void)
{
        if(SCL)
        SCL = 0;
        SDA = 0;
        i2c_delay();
        SCL = 1;
        i2c_delay();
        SDA = 1;
}
bit i2c_wrdata(unsigned char dat)
{
        bit data_bit;
        unsigned char i;
        for(i=0;i<8;i++)
        {
                data_bit = dat & 0x80;
                SDA = data_bit;
                i2c_clk();
                dat = dat<<1;
        }
        SDA = 1;
        i2c_delay();
        SCL = 1;
```

```
        i2c_delay();
        data_bit = SDA;
        SCL = 0;
        i2c_delay();
        return(data_bit);                    // if send_bit = 0 i2c OK!
}
unsigned char i2c_rddata(void)
{
        bit rd_bit;
        unsigned char i,dat;
        dat = 0x00;
        for(i=0;i<8;i++)
        {
                i2c_delay();
                SCL = 1;
                i2c_delay();
                rd_bit = SDA;
                dat = dat<<1;
                dat = dat | rd_bit;
                SCL = 0;
        }
        SDA = 1;
        i2c_delay();
        i2c_clk();
        SCL = 1;
        return(dat);
}
void i2c_NACK()
{
        SDA = 1;
        i2c_delay();
        i2c_clk();
        SCL = 1;
}
```

CODE Visual Basic

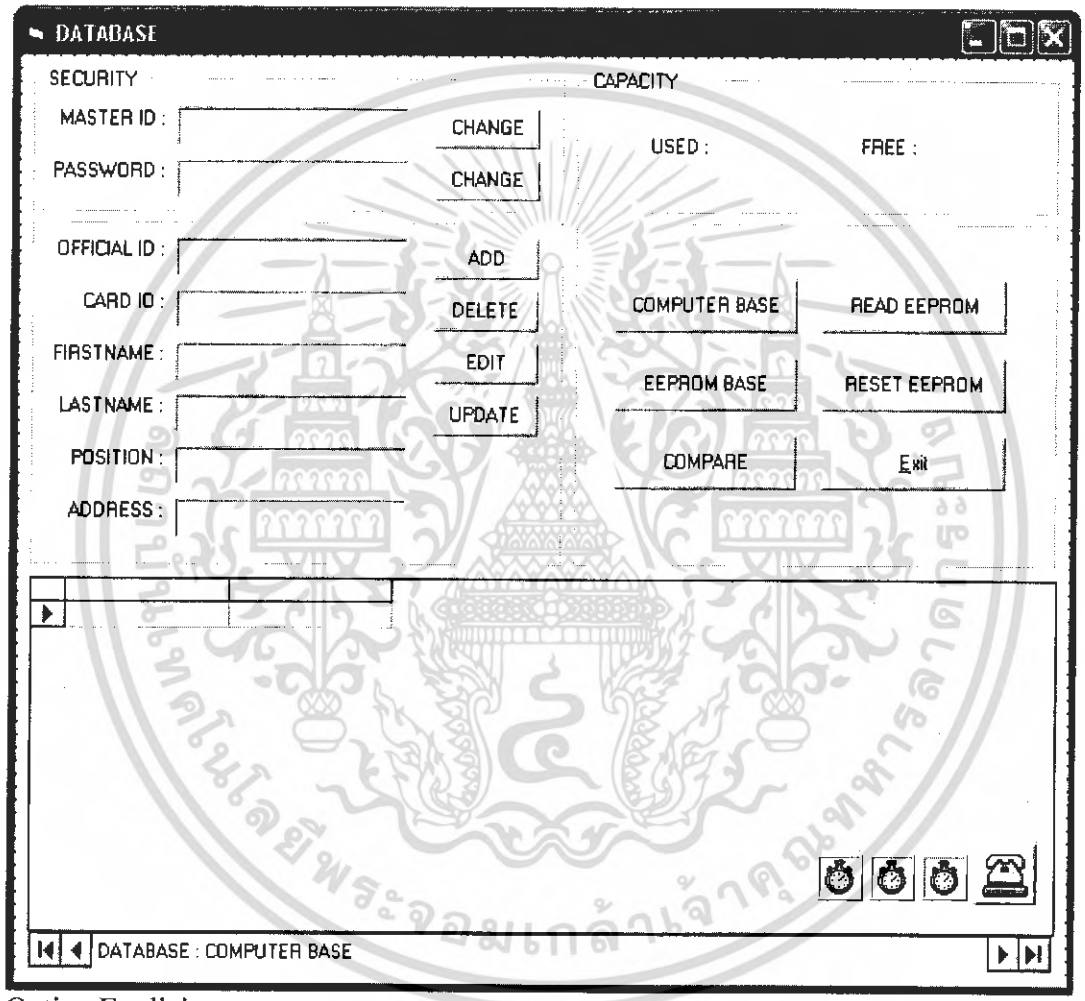
โปรแกรมส่วนหน้าต่างหลัก




```
Option Explicit
Private Sub CmdExit_Click()
   End
End Sub

Private Sub mdb_Click()
   CmdDatabase.Value = True
End Sub

Private Sub mmem_Click()
   CmdMemory.Value = True
End Sub

Private Sub mexit_Click()
   End
End Sub

Private Sub CmdDatabase_Click()
   Form2.Show
   Form1.Visible = False
End Sub
```

```
Private Sub CmdMemory_Click()
   Form3.Show
   Form1.Visible = False
End Sub
```

โปรแกรมส่วนหน้าต่าง **DATABASE**




```
Option Explicit
Dim data As String

Private Sub CmdChgMas_Click()
   Dim strMas As String
   Dim Mas_buf As String
   Dim Response As Variant
   Mas_buf = TxtMaster.Text
Change_Master:
   strMas = InputBox("Please,enter new Master ID.", "Change Master ID", Mas_buf,
300, 200)
   If Len(strMas) = 10 Then
```

```
      Response = MsgBox("Do you want to write MASTER ID : " + strMas + " to
EEPROM DATABASE?", vbQuestion + vbYesNo, "Write MASTER ID")
      If Response = vbYes Then
         MSComm1.PortOpen = True
         MSComm1.Output = "WM"
         MSComm1.Output = strMas
         TxtMaster.Text = strMas
         MSComm1.PortOpen = False
      Else
         TxtMaster.Text = Mas_buf
      End If
   Else
      Response = MsgBox("MASTER ID must be 10 Digits. Do you want to change
MASTER ID again?", vbQuestion + vbYesNo, "WRONG MASTER ID")
      If Response = vbYes Then
         GoTo Change_Master
      Else
      TxtMaster.Text = Mas_buf
      End If
   End If
End Sub

Private Sub CmdChgPwd_Click()
   Dim strPwd As String
   Dim Pwd_buf As String
   Dim Response As Variant
   Pwd_buf = TxtPwd.Text
Change_Pwd:
   strPwd = InputBox("Please,enter new password.", "Change Password", Pwd_buf,
300, 200)
   If Len(strPwd) = 8 Then
      Response = MsgBox("Do you want to write PASSWORD : " + strPwd + " to
EEPROM DATABASE?", vbQuestion + vbYesNo, "Write PASSWORD")
      If Response = vbYes Then
         MSComm1.PortOpen = True
         MSComm1.Output = "WP"
         MSComm1.Output = strPwd
         TxtPwd.Text = strPwd
         MSComm1.PortOpen = False
      Else
         TxtMaster.Text = Pwd_buf
      End If
   Else
      Response = MsgBox("PASSWORD must be 8 Digits. Do you want to change
PASSWORD again?", vbQuestion + vbYesNo, "WRONG MASTER ID")
      If Response = vbYes Then
         GoTo Change_Pwd
```

```
        Else
        TxtMaster.Text = Pwd_buf
        End If
    End If
End Sub


Private Sub CmdComB_Click()
    Adodc1.RecordSource = "SELECT * FROM DAT ORDER BY ID ASC;"
'Select for DATABASE ON Com.
    Adodc1.Refresh
    Adodc1.Caption = "DATABASE : COMPUTER BASE"
    CmdAdd.Enabled = True
    CmdDel.Enabled = True
    CmdEdit.Enabled = True
    LbOffice.Enabled = True
    TxtOffice.Enabled = True
    LbFname.Enabled = True
    TxtFname.Enabled = True
    LbLname.Enabled = True
    TxtLname.Enabled = True
    LbPos.Enabled = True
    TxtPos.Enabled = True
    LbAddr.Enabled = True
    TxtAddr.Enabled = True
End Sub

Private Sub CmdCompare_Click()
    Adodc1.RecordSource = "SELECT EEPROM_DAT.CardID,FirstName,LastName
FROM EEPROM_DAT LEFT OUTER JOIN DAT ON EEPROM_DAT.CardID =
DAT.CardID;"
    Adodc1.Refresh
    Adodc1.Caption = "DATABASE : COMPARE"
    CmdAdd.Enabled = False
    CmdDel.Enabled = False
    CmdEdit.Enabled = False
    LbOffice.Enabled = False
    TxtOffice.Enabled = False
    LbFname.Enabled = True
    TxtFname.Enabled = True
    LbLname.Enabled = True
    TxtLname.Enabled = True
    LbPos.Enabled = False
    TxtPos.Enabled = False
    LbAddr.Enabled = False
    TxtAddr.Enabled = False
End Sub
```

```
Private Sub CmdEEB_Click()
   Adodc1.RecordSource = "SELECT * FROM EEPROM_DAT;"        'Select for
DATABASE ON EEPROM.
   Adodc1.Refresh
   Adodc1.Caption = "DATABASE : EEPROM BASE"
   DataGrid1.AllowDelete = False
   CmdAdd.Enabled = False
   CmdDel.Enabled = False
   CmdEdit.Enabled = False
   LbOffice.Enabled = False
   TxtOffice.Enabled = False
   LbFname.Enabled = False
   TxtFname.Enabled = False
   LbLname.Enabled = False
   TxtLname.Enabled = False
   LbPos.Enabled = False
   TxtPos.Enabled = False
   LbAddr.Enabled = False
   TxtAddr.Enabled = False
End Sub

Private Sub CmdAdd_Click()
   Adodc1.Recordset.AddNew
   TxtOffice.SetFocus
   TxtOffice.Locked = False
   TxtID.Locked = False
   TxtFname.Locked = False
   TxtLname.Locked = False
   TxtPos.Locked = False
   TxtAddr.Locked = False
   CmdUpdate.Enabled = True
End Sub

Private Sub CmdDel_Click()
   If MsgBox("Are you sure to delete this record?", vbExclamation + vbYesNo,
"Delete Record!") = vbYes Then
      Adodc1.Recordset.Delete
      Adodc1.Recordset.MoveNext
      If Adodc1.Recordset.EOF = True Then
         Adodc1.Recordset.MoveLast
      End If
   End If
End Sub
```

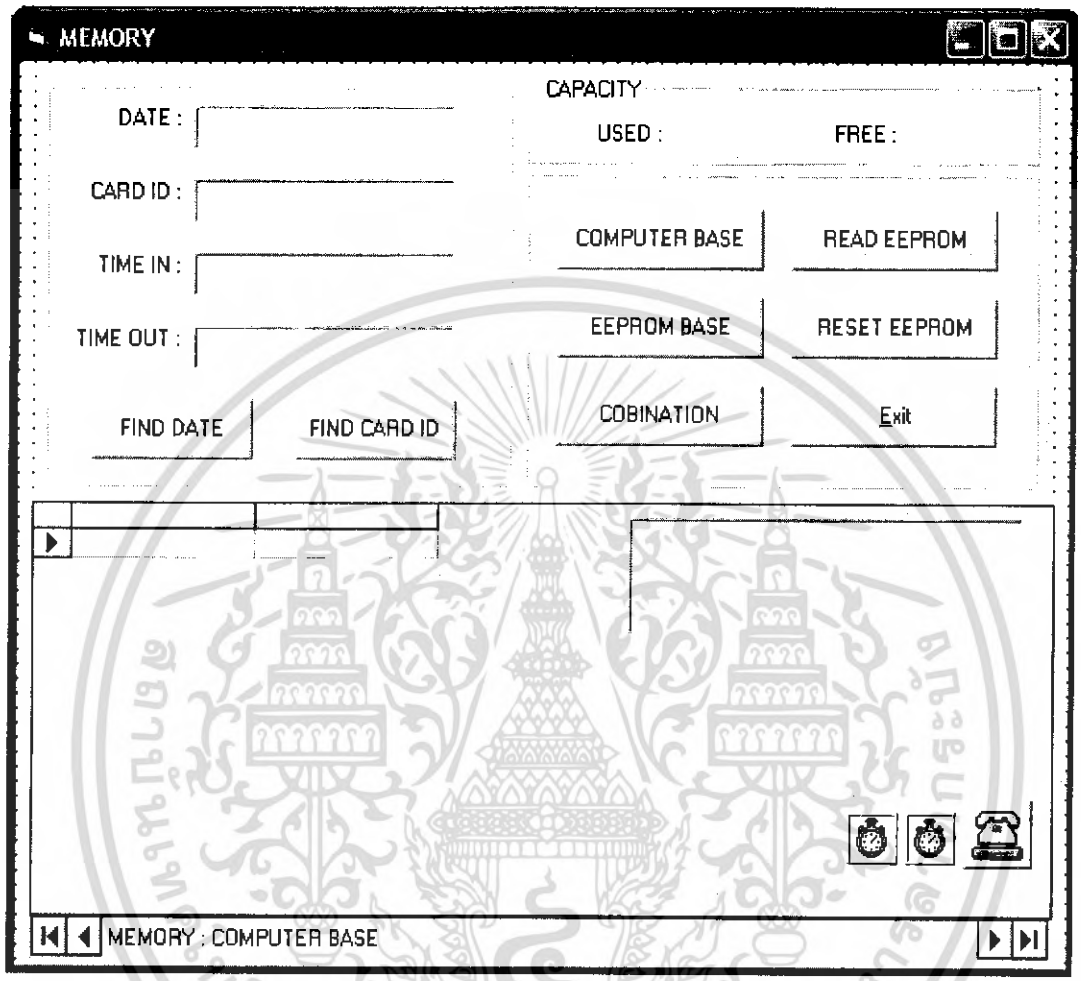
```
Private Sub CmdEdit_Click()
   TxtOffice.SetFocus
   TxtOffice.Locked = False
   TxtID.Locked = False
   TxtFname.Locked = False
   TxtLname.Locked = False
   TxtPos.Locked = False
   TxtAddr.Locked = False
   CmdUpdate.Enabled = True
End Sub

Private Sub CmdReset_Click()
 Dim Response As Variant
   Response = MsgBox("Reset EEPROM will delete all DATABASE. Do you want to
continue?", vbExclamation + vbYesNo, "RESET EEPROM DATABASE")
   If Response = vbYes Then
      If (MSComm1.PortOpen = False) Then
         MSComm1.CommPort = 1
         MSComm1.Settings = "9600,n,8,1"
         MSComm1.PortOpen = True
      End If
      MSComm1.Output = "RD"
      MSComm1.PortOpen = False
      CmdEEB.Enabled = False
      CmdCompare.Enabled = False
      CmdReset.Enabled = False

      Adodc1.RecordSource = "SELECT * FROM EEPROM_DAT;"
      Adodc1.Refresh
      If (Adodc1.Recordset.BOF = False) Or (Adodc1.Recordset.EOF = False) Then
DEL:
      Adodc1.Recordset.MoveFirst
      If (Adodc1.Recordset.EOF = False) Then
         Adodc1.Recordset.Delete
         GoTo DEL
      End If
   End If
   Else
      MsgBox "RESET DATABASE is canceled, DATABASE is not deleted.",
vbExclamation + vbOKOnly, "CANCEL RESET"
   End If
End Sub

Private Sub CmdUpdate_Click()
   Adodc1.Recordset.Update
   TxtOffice.Locked = True
   TxtID.Locked = True
```

```
    TxtFname.Locked = True
    TxtLname.Locked = True
    TxtPos.Locked = True
    TxtAddr.Locked = True
    CmdUpdate.Enabled = False
End Sub
Private Sub CmdExit_Click()
    If (MSComm1.PortOpen = True) Then
        MSComm1.PortOpen = False
    End If
        Adodc1.RecordSource = "SELECT * FROM EEPROM_DAT;"
        Adodc1.Refresh
        If (Adodc1.Recordset.BOF = False) Or (Adodc1.Recordset.EOF = False) Then
DEL:
        Adodc1.Recordset.MoveFirst
        If (Adodc1.Recordset.EOF = False) Then
            Adodc1.Recordset.Delete
            GoTo DEL
        End If
    End If
    Unload Form2
    Form1.Visible = True
End Sub

Private Sub CmdRead_Click()
    If (MSComm1.PortOpen = False) Then
        MSComm1.CommPort = 1
        MSComm1.Settings = "9600,n,8,1"
        MSComm1.PortOpen = True
    End If
    MSComm1.Output = "DB"
    Timer1.Interval = 100
    Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
    Dim Num As Integer
    MSComm1.InputLen = 2
    data = MSComm1.Input
    If (data = "OK") Then
        LbMaster.Enabled = True
        TxtMaster.Enabled = True
        CmdChgMas.Enabled = True
        LbPwd.Enabled = True
        TxtPwd.Enabled = True
        CmdChgPwd.Enabled = True
        CmdEEB.Enabled = True
```

```
        CmdReset.Enabled = True
        CmdCompare.Enabled = True
        CmdAdd.Enabled = False
        CmdDel.Enabled = False
        CmdEdit.Enabled = False
        LbOffice.Enabled = False
        LbFname.Enabled = False
        LbLname.Enabled = False
        LbPos.Enabled = False
        LbAddr.Enabled = False
        TxtOffice.Enabled = False
        TxtFname.Enabled = False
        TxtLname.Enabled = False
        TxtPos.Enabled = False
        TxtAddr.Enabled = False

        MSComm1.InputLen = 4
        data = MSComm1.Input
        Num = Val(data)
        LbUse.Caption = Num
        LbFree.Caption = 1635 - Num
        MSComm1.InputLen = 8
        data = MSComm1.Input
        TxtPwd.Text = data
        MSComm1.InputLen = 10
        data = MSComm1.Input
        TxtMaster.Text = data

        DataGrid1.AllowAddNew = True
        DataGrid1.AllowUpdate = True
        Timer2.Interval = 100                ' Set Timer2
        Timer2.Enabled = True
    ElseIf (data = "ER") Then
        MsgBox "EEPROM DATABASE is not connected! Please, Check connection.",
vbExclamation + vbOKOnly, "DATABASE UNPLUG!"
        MSComm1.PortOpen = False
    Else
        MsgBox "Please, check serial connection and power supply.", vbExclamation +
vbOKOnly, "Connection error!"
        MSComm1.PortOpen = False
    End If
        Timer1.Enabled = False
End Sub

Private Sub Timer2_Timer()
On Error GoTo Got_error
        MSComm1.InputLen = 10
```

```
    data = MSComm1.Input
    If (data <> "") Then
      Adodc1.RecordSource = "SELECT * FROM EEPROM_DAT;"        'Select
for DATABASE ON EEPROM.
      Adodc1.Refresh
      Adodc1.Caption = "DATABASE : EEPROM BASE"
      Adodc1.Recordset.AddNew
      TxtID.Text = data                        ' Input CARD ID
      Adodc1.Recordset.Update
    Else
      MSComm1.PortOpen = False
      DataGrid1.AllowAddNew = False
      DataGrid1.AllowUpdate = False
      Timer2.Enabled = False
      MsgBox "READ EEPROM MEMORY : COMPLETE", vbInformation +
vbOKOnly, "Read Complete!"
      Adodc1.Refresh
    End If
    Exit Sub
Got_error:
      MSComm1.PortOpen = False
      Adodc1.Refresh
      DataGrid1.AllowAddNew = False
      DataGrid1.AllowUpdate = False
      Timer2.Enabled = False
      MsgBox "Data in EEPROM DATABASE is already in DATABASE.",
vbInformation + vbOKOnly, "DATABASE repeated!"
End Sub
```

โปรแกรมส่วนหน้าต่าง **MEMORY**




```
Option Explicit
Dim data As String
Dim Bookmark As Variant

Private Sub CmdComBase_Click()
   Adodc1.RecordSource = "SELECT * FROM MEM ORDER BY Date ASC;"
'Select for DATABASE ON EEPROM.
   Adodc1.Refresh
   Adodc1.Caption = "MEMORY : COMPUTER BASE"
End Sub

Private Sub CmdCombination_Click()
   Dim D As String
   Dim C As String
   Dim I As String
   Dim O As String
```

```
    On Error Resume Next
    Adodc1.RecordSource = "SELECT * FROM EEPROM_MEM ORDER BY Date
ASC;"        'Select for DATABASE ON EEPROM.
    Adodc1.Refresh
    Adodc1.Caption = "MEMORY : COMBINATION"
    If (Adodc1.Recordset.BOF = False) Or (Adodc1.Recordset.EOF = False) Then
DEL:
        Adodc1.Recordset.MoveFirst
        D = TxtDate.Text
        C = TxtID.Text
        I = TxtTimeIN.Text
        O = TxtTimeOut.Text
        If (Adodc1.Recordset.EOF = False) Then
            Adodc1.Recordset.Delete
            Adodc1.RecordSource = "SELECT * FROM MEM ORDER BY Date ASC;"
            Adodc1.Refresh
            Adodc1.Recordset.AddNew
            TxtDate.Text = D
            TxtID.Text = C
            TxtTimeIN.Text = I
            TxtTimeOut.Text = O
            Adodc1.Recordset.Update
            Adodc1.RecordSource = "SELECT * FROM EEPROM_MEM ORDER BY
Date ASC;"
            Adodc1.Refresh
            GoTo DEL
        End If
    End If
    Adodc1.RecordSource = "SELECT * FROM MEM;"
    Adodc1.Refresh
End Sub

Private Sub CmdEEB_Click()
    Adodc1.RecordSource = "SELECT * FROM EEPROM_MEM ORDER BY Date
ASC;"        'Select for DATABASE ON EEPROM.
    Adodc1.Refresh
    Adodc1.Caption = "MEMORY : EEPROM BASE"
End Sub

Private Sub CmdExit_Click()
    If (MSComm1.PortOpen = True) Then
        MSComm1.PortOpen = False
    End If
        Adodc1.RecordSource = "SELECT * FROM EEPROM_MEM;"
        Adodc1.Refresh
        If (Adodc1.Recordset.BOF = False) Or (Adodc1.Recordset.EOF = False) Then
DEL:
```

```
        Adodc1.Recordset.MoveFirst
        If (Adodc1.Recordset.EOF = False) Then
          Adodc1.Recordset.Delete
          GoTo DEL
        End If
      End If
      Unload Form3
      Form1.Visible = True
End Sub

Private Sub CmdFdate_Click()
    Dim Response As Variant
    Dim Day As String
Find_Date:
    Day = InputBox("Please enter DATE which you want to find.", "Find DATE",
"DD/MM/YY", 100, 200)
    If Len(Day) = 8 Then
       TxtCode.Text = "SELECT * FROM MEM WHERE Date ='" + Day + "'; "
       Adodc1.RecordSource = TxtCode.Text
       Adodc1.Refresh
    Else
       Response = MsgBox("Data in DATE is DD/MM/YY. Do you want to FIND
DATE again?", vbQuestion + vbYesNo, "INVALID DATE")
       If Response = vbYes Then
         GoTo Find_Date
       End If
    End If
End Sub

Private Sub CmdFID_Click()
    Dim Response As Variant
    Dim ID As String
Find_ID:
    ID = InputBox("Please enter CARD ID which you want to find.", "Find CARD
ID", "1234567890", 100, 200)
    If Len(ID) = 10 Then
       TxtCode.Text = "SELECT * FROM MEM WHERE CardID ='" + ID + "'; "
       Adodc1.RecordSource = TxtCode.Text
       Adodc1.Refresh
    Else
       Response = MsgBox("CARD ID must be 10 Digits. Do you want to FIND
CARD ID again?", vbQuestion + vbYesNo, "INVALID CARD ID")
       If Response = vbYes Then
         GoTo Find_ID
       End If
    End If
End Sub
```

```
Private Sub CmdRead_Click()
   If (MSComm1.PortOpen = False) Then
      MSComm1.CommPort = 1
      MSComm1.Settings = "9600,n,8,1"
      MSComm1.PortOpen = True
   End If
   Adodc1.RecordSource = "SELECT * FROM EEPROM_MEM;"
   Adodc1.Refresh
   Adodc1.Caption = "MEMORY : EEPROM BASE"
   MSComm1.Output = "MM"
   Timer1.Interval = 100
   Timer1.Enabled = True
End Sub

Private Sub CmdReset_Click()
   Dim Response As Variant
   Response = MsgBox("Reset EEPROM will delete all MEMORY. Do you want to
continue?", vbExclamation + vbYesNo, "RESET EEPROM MEMORY")
   If Response = vbYes Then
      If (MSComm1.PortOpen = False) Then
         MSComm1.CommPort = 1
         MSComm1.Settings = "9600,n,8,1"
         MSComm1.PortOpen = True
      End If
      MSComm1.Output = "RM"
      MSComm1.PortOpen = False
      CmdEEB.Enabled = False
      CmdCombination.Enabled = False
      CmdReset.Enabled = False

      Adodc1.RecordSource = "SELECT * FROM EEPROM_MEM;"
      Adodc1.Refresh
      If (Adodc1.Recordset.BOF = False) Or (Adodc1.Recordset.EOF = False) Then
DEL:
         Adodc1.Recordset.MoveFirst
         If (Adodc1.Recordset.EOF = False) Then
            Adodc1.Recordset.Delete
            GoTo DEL
         End If
      End If

   Else
      MsgBox "RESET MEMORY is canceled, MEMORY is not deleted.",
vbInformation + vbOKOnly, "CANCEL RESET"
   End If
End Sub
```

```
Private Sub Timer1_Timer()
   Dim Num As Integer
   MSComm1.InputLen = 2
   data = MSComm1.Input
   If (data = "OK") Then
      MSComm1.InputLen = 4
      data = MSComm1.Input
      Num = Val(data)
      LbUse.Caption = Num
      LbFree.Caption = 5460 - Num
      DataGrid1.AllowAddNew = True
      DataGrid1.AllowUpdate = True
      CmdEEB.Enabled = True
      CmdReset.Enabled = True
      CmdCombination.Enabled = True

      Timer2.Interval = 100                ' Set Timer2
      Timer2.Enabled = True
  ElseIf (data = "ER") Then
      MsgBox "EEPROM MEMORY is not connected! Please, Check connection.",
vbExclamation + vbOKOnly, "MEMORY UNPLUG!"
      MSComm1.PortOpen = False
   Else
      MsgBox "Please, check serial connection and power supply.", vbExclamation +
vbOKOnly, "Connection error!"
      MSComm1.PortOpen = False
   End If
      Timer1.Enabled = False
End Sub

Private Sub Timer2_Timer()
      Dim p1 As String
      Dim p2 As String
      Dim p3 As String
On Error GoTo Got_error
      MSComm1.InputLen = 6
      data = MSComm1.Input
      If (data <> "") Then
         Adodc1.Recordset.AddNew
         p1 = Mid(data, 1, 2)
         p2 = Mid(data, 3, 2)
         p3 = Mid(data, 5, 2)
         TxtDate.Text = p1 + "/" + p2 + "/" + p3    'Input DATE
         MSComm1.InputLen = 10
         data = MSComm1.Input
         TxtID.Text = data                    ' Input CARD ID
         MSComm1.InputLen = 4
```

```
        data = MSComm1.Input
        If (data <> "FFFF") Then
           p1 = Mid(data, 1, 2)
           p2 = Mid(data, 3, 2)
           TxtTimeIN.Text = p1 + ":" + p2            ' Input Time IN
        Else
           TxtTimeIN.Text = ""
        End If
        MSComm1.InputLen = 4
        data = MSComm1.Input
        If (data <> "FFFF") Then
           p1 = Mid(data, 1, 2)
           p2 = Mid(data, 3, 2)
           TxtTimeOut.Text = p1 + ":" + p2        ' Input Time OUT
        Else
           TxtTimeOut.Text = ""
        End If
        Adodc1.Recordset.Update
     Else
        MSComm1.PortOpen = False
        DataGrid1.AllowAddNew = False
        DataGrid1.AllowUpdate = False
        Timer2.Enabled = False
        MsgBox "READ EEPROM MEMORY : COMPLETE", vbInformation +
vbOKOnly, "Read Complete!"
        Adodc1.Refresh
     End If
     Exit Sub
Got_error:
        MSComm1.PortOpen = False
        Adodc1.Refresh
        DataGrid1.AllowAddNew = False
        DataGrid1.AllowUpdate = False
        Timer2.Enabled = False
        MsgBox "Data in EEPROM MEMORY is already in MEMORY.",
vbInformation + vbOKOnly, "MEMORY repeated!"
End Sub
```