ปริญญานิพนธ์

ระบบเช็คอิน - เช็คเอาต์โรงแรม

HOTEL CHECK IN - CHECK OUT SYSTEM

นายณัฐพล ธรรมบุตร
นางสาวพิรดา ตำคำ
นายสันติ สว่างเมฆ
นายโสภณ แซ่ลิ้ม

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2546

ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

ชื่อหัวข้อ ระบบเช็คอิน-เช็คเอาต์โรงแรม

Hotel Check in-Check out System

ชื่อนักศึกษา

1. นายณัฐพล ธรรมบุตร	รหัสประจำตัว	45035339
2. นางสาวพิรดา ต๋ำคำ	รหัสประจำตัว	45035350
3. นายสันติ สว่างเมฆ	รหัสประจำตัว	45035362
4. นายโสภณ แซ่ลิ้ม	รหัสประจำตัว	45035367

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ที่ปรึกษา อาจารย์สุชิน อาจหาญ

อาจารย์ที่ปรึกษาร่วม อาจารย์วรวิทย์ สมหา

คณะกรรมการสอบปริญญานิพนธ์	ลายมือชื่อ
1. อาจารย์ประเสริฐ เคนพันค้อ	
2. อาจารย์พิชญ์สินี มงคลขจิต	
3. อาจารย์สุชิน อาจหาญ	
4. อาจารย์วรวิทย์ สมหา	
5. อาจารย์อำพล ทองระอา	

วัน/เดือน/ปีที่สอบ วันอังคารที่ 30 มีนาคม พ.ศ. 2547 เวลา 17.00 น.

สถานที่สอบ ห้อง ค.311 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.สุรสิทธิ์ ราตรี)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 31 เดือน มี.ค. พ.ศ. 47

<BT4620252>

ระบบเช็คอิน-เช็คเอาต์โรงแรม

ปริญญานิพนธ์

เรื่อง ระบบเช็คอิน - เช็คเอาต์โรงแรม

Hotel Check in - Check out System

วัตถุประสงค์

1. เพื่อศึกษาระบบเช็คอิน - เช็คเอาต์ ในโรงแรมและการเขียนโปรแกรมอ่านเขียนบัตรแถบแม่เหล็ก
2. เพื่อออกแบบวงจรของระบบเช็คอิน - เช็คเอาต์โรงแรม
3. เพื่อสร้างระบบเช็คอิน - เช็คเอาต์โรงแรม
4. เพื่อนำระบบเช็คอิน - เช็คเอาต์โรงแรมมาทำการทดลองตรวจสอบความสมบูรณ์ได้
5. เพื่อนำระบบเช็คอิน - เช็คเอาต์โรงแรมไปใช้งานได้จริง

ประโยชน์ที่คาดว่าจะได้รับ

1. มีความรู้ความเข้าใจในระบบเช็คอิน - เช็คเอาต์โรงแรมและการเขียนโปรแกรมอ่านเขียนบัตรแถบแม่เหล็กได้
2. ได้แบบวงจรของระบบเช็คอิน - เช็คเอาต์โรงแรม
3. ได้ระบบเช็คอิน - เช็คเอาต์โรงแรม
4. ได้ผลการทดลองที่สมบูรณ์ของระบบเช็คอิน - เช็คเอาต์โรงแรม
5. นำระบบเช็คอิน - เช็คเอาต์โรงแรมไปใช้งานได้จริง

ชื่อหัวข้อ	ระบบเช็คอิน - เช็คเอาต์โรงแรม
นักศึกษา	นายณัฐพล ธรรมบุตร
	นางสาวพิรดา ต๋ำคำ
	นายสันติ สว่างเมฆ
	นายโสภณ แซ่ลิ้ม
อาจารย์ที่ปรึกษา	อาจารย์สุชิน อาจหาญ
อาจารย์ที่ปรึกษาร่วม	อาจารย์วรวิทย์ สมหา
หลักสูตร	ครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชา	อิเล็กทรอนิกส์และคอมพิวเตอร์
ปีการศึกษา	2546

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอ ระบบเช็คอิน – เช็คเอาท์ในโรงแรม ควบคุมการทำงานระบบด้วย ไมโครคอนโทรเลอร์เบอร์ 89C51RD2 รับส่งข้อมูลของแต่ละห้องด้วยมาตรฐาน RS-485 รับข้อมูลจากบัตรแถบแม่เหล็กมาแสดงผลที่คอมพิวเตอร์ พร้อมทั้งแสดงระบบการเข้าพักเวลาเข้าออกและค่าใช้จ่าย ต่างๆ ด้วยโปรแกรม Visual C++ ให้เปิดประตูด้วยโซลินอยล์ และตรวจจับควันด้วยตัวตรวจจับควัน

Thesis Title	Hotel Check in - Check out System	
Students	Mr.Natapon	Thummabutara
	Miss.Pirada	Takum
	Mr.Santi	Sawangmek
	Mr.Sophon	Saelim
Advisor	Mr.Suchin	Adhan
Co-Advisor	Mr.Worawit	Somha
Education Level	Bachelor of Science in Industrial Education	
Program in	Industrial Instrument Technology	
Academic Year	2003	

ABSTRACT

This thesis present the Hotel Check in - Check out System. It has been implemented using the microcontroller 89C51RD2 sent and receive data of classroom using the standard RS485 resave source code. This show check in & check out on computer with Visual C++ Program. Cause open / close the door with solinoil and using smoke detector in time this fire.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สามารถลุล่วงไปด้วยดี เนื่องมาจากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน ขอขอบคุณอาจารย์สุชิน อาจหาญ อาจารย์วรวิทย์ สมหา ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม และอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่ให้ความอนุเคราะห์ เครื่องมือและอุปกรณ์ รวมทั้งยังให้คำแนะนำ แนวความคิด แนวความรู้ต่างๆ และแนวทางการแก้ปัญหาในการจัดทำปริญญานิพนธ์ ขอขอบคุณห้องสมุดคณะครุศาสตร์ อุตสาหกรรม ห้องสมุดคณะวิศวกรรมศาสตร์ และหอสมุดกลาง ที่ช่วยอำนวยความสะดวกและเอื้อเฟื้อสถานที่ในการค้นคว้าหาข้อมูล สุดท้ายที่ควรระลึกถึงอย่างยิ่งคือ บิดาและมารดาที่เป็นผู้ให้การสนับสนุนด้านการศึกษาและเป็นผู้ให้กำลังใจด้วยดีตลอดมาตั้งแต่อดีตจนถึงปัจจุบัน

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ปัจจุบันมีการแข่งขันทางด้านการโรงแรมมากขึ้น ไม่ว่าจะเป็นในส่วนของการให้บริการหรือด้านสถานที่พักเพื่อดึงดูดลูกค้าให้เข้ามาใช้บริการทางด้านโรงแรมมากขึ้น โดยโรงแรมแต่ละแห่งนั้นต่างก็พยายามดึงเอาสิ่งแปลกใหม่มานำเสนอ เพื่อการตอบสนองความต้องการความสะดวกสบายและความพึงพอใจต่อลูกค้าผู้มาใช้บริการ

ระบบเช็คอิน - เช็คเอาต์โรงแรมในปัจจุบัน ต้องอาศัยพนักงานในการดำเนินการตรวจสอบห้องพักและป้อนข้อมูลเข้าสู่คอมพิวเตอร์และต้องใช้กุญแจในการเปิดปิดห้องพัก ซึ่งอาจมีปัญหาทางด้านความปลอดภัยและความไม่สะดวกของลูกค้า จึงต้องการใช้ระบบคอมพิวเตอร์เข้ามาช่วยในการเช็คอิน - เช็คเอาต์โรงแรมเพื่อเพิ่มความสะดวกและความปลอดภัยยิ่งขึ้น

1.2 ขีดความสามารถของโครงงาน

1. สามารถตั้งค่าและคิดคำนวณค่าห้องพักตอนเช็คเอาต์ของลูกค้าได้
2. มีข้อมูลประวัติการเข้า - ออกห้องพัก
3. สามารถตรวจสอบการเข้า - ออกห้องพักได้
4. ใช้บัตรแถบแม่เหล็กในการเปิด - ปิดห้องพัก
5. ระบบเช็คอิน- เช็คเอาต์โรงแรมมีจำนวน 3 ห้องและสามารถขยายระบบได้อีก 250 ห้อง

1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่างๆ เพื่อสะดวกต่อการศึกษาและทำความเข้าใจ ในแต่ละบทจะประกอบด้วยเนื้อหาดังต่อไปนี้

บทที่ 1 บทนำ ซึ่งมีเนื้อหาเกี่ยวกับความเป็นมาและความสำคัญที่ทำให้เกิดโครงงานนี้ขึ้นมา รวมทั้งยังกล่าวถึงวัตถุประสงค์ขอบเขตและประโยชน์ของการทำปริญญานิพนธ์ในครั้งนี้

บทที่ 2 ทฤษฎีและหลักการ กล่าวถึงเนื้อหาที่นำมาอ้างอิง ใช้เป็นแนวทางในการออกแบบและสร้างระบบเช็คอิน - เช็คเอาต์โรงแรม

บทที่ 3 การออกแบบและการสร้าง กล่าวถึงเนื้อหาโดยละเอียดตั้งแต่ขั้นตอนในการออกแบบวงจรในส่วนของการอ่านแถบแม่เหล็ก การนำส่วนต่างๆ มาประกอบและทำงานร่วมกัน เพื่อให้ทำงานร่วมกันอย่างประสิทธิภาพ

บทที่ 4 การทดลองและผลการทดลอง เป็นการนำเสนอในส่วนของการทดลองและผลการทดลองโดยแบ่งการทดลองออกเป็นส่วนๆ ตามการออกแบบพร้อมบันทึกการทดลองในแต่ละส่วน

บทที่ 5 บทสรุปปัญหาแนวทางแก้ไขและพัฒนา เป็นการสรุปเกี่ยวกับความสามารถ ประสิทธิภาพการทำงานของระบบเช็คอิน - เช็คเอาต์โรงแรมและกล่าวถึงปัญหาที่เกิดนับตั้งแต่การเริ่มสร้างโครงงานจนกระทั่งโครงงานเสร็จสมบูรณ์ ตลอดจนแนวทางแก้ไขปัญหา พร้อมทั้งเสนอแนวทางการพัฒนาระบบเช็คอิน - เช็คเอาต์โรงแรมให้สามารถนำไปใช้งานได้และปรับปรุงให้ประสิทธิภาพมากยิ่งขึ้น

ภาคผนวก ก แสดงภาพเครื่องต้นแบบ การติดตั้ง การเชื่อมต่อกับอุปกรณ์อื่นๆ ขณะใช้งานจริง

ภาคผนวก ข ประกอบด้วยแผนผังรายละเอียดของวงจรและแผ่นวงจรพิมพ์

ภาคผนวก ค แสดงรายการอุปกรณ์ที่ใช้งานในแต่ละวงจร

ภาคผนวก ง แสดงแผนผังการทำงานและรหัสต้นฉบับของโปรแกรมทั้งหมดที่สร้างขึ้นเพื่อประกอบการทำงานของโครงงาน

ภาคผนวก จ เป็นคู่มือการใช้งานระบบเช็คอิน - เช็คเอาต์โรงแรม

ภาคผนวก ฉ แสดงรายละเอียดและคุณสมบัติของอุปกรณ์สำคัญที่ใช้ในโครงงาน

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล่าวนำ

เนื้อหาของปริญญานิพนธ์ในบทนี้เป็นทฤษฎีและหลักการที่นำมาใช้ประกอบในโครงงาน โดยจะประกอบด้วยทฤษฎี หลักการที่เกี่ยวข้องกับคุณสมบัติ, การทำงานของไมโครคอนโทรลเลอร์ MCS - 51, พอร์ตอนุกรม, ชุดอ่านบัตรแถบแม่เหล็ก และโซลินอยด์

2.2 ไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ในตระกูล MCS-51ที่มีขนาด 8 บิต ทุกๆ เบอร์จะมีสถาปัตยกรรมพื้นฐานที่เหมือนกัน เพื่อความเหมาะสมในการนำเอาไปใช้งานตามความต้องการต่างๆ แต่เดิม 8051 ถูกสร้างด้วยวิธี HMOS I แต่ในปัจจุบันได้สร้างด้วยวิธี HMOS II จึงมีชื่อเป็น 8051AH ไมโครคอนโทรลเลอร์ตระกูล 51 นั้น ถึงแม้ว่าจะมีหลายเบอร์แต่เราก็จะเรียกว่าเป็น "8051" ซึ่งหมายถึงไมโครคอนโทรลเลอร์ตระกูล 51 ส่วนเบอร์ 8032 และ 8052 มีหน่วยความจำภายในเพิ่มขึ้นและมีวงจรนับ/จับเวลา ขนาด 16 บิต เพิ่มขึ้นมาดังตารางที่ 2.1

ตารางที่ 2.1 ไมโครคอนโทรลเลอร์ตระกูล 8051

เบอร์	หน่วยความจำภายใน		จำนวนไทเมอร์/เคาน์เตอร์	อินเตอร์รัพต์หมายเลข
	เก็บโปรแกรม	เก็บข้อมูล		
8052H	8K x 8 ROM	256 x 8 ROM	3 x 16 Bit	6
8051H	4K x 8 ROM	128 x 8 ROM	2 x 16 Bit	5
8051	4K x 8 ROM	256 x 8 ROM	2 x 16 Bit	5
8032AH	ไม่มี	128 x 8 ROM	3 x 16 Bit	6
8031AH	ไม่มี	128 x 8 ROM	2 x 16 Bit	5
8031	ไม่มี	128 x 8 ROM	2 x 16 Bit	5
8751H	4K x 8 EPROM	128 x 8 ROM	2 x 16 Bit	5
80751H-12	4K x 8 EPROM	128 x 8 ROM	2 x 16 Bit	5

2.2.1 คุณสมบัติของไมโครคอนโทรลเลอร์ MCS-51

1) ต้องการแหล่งจ่ายไฟ +5 โวลต์ ชุดเดียว

2) มีหน่วยความจำโปรแกรม (Program Memory) ขนาด 4 กิโลไบต์ สำหรับเบอร์ 8051 และ 8031 สำหรับเบอร์ 8052 มีหน่วยความจำถึง 8 กิโลไบต์

3) มีหน่วยความจำสำหรับเก็บข้อมูล (Data Memory) ขนาด 128 ไบต์ สำหรับเบอร์ 8052 ขึ้นไปมีถึง 256 ไบต์

4) มีหน่วยความจำสำหรับเก็บโปรแกรมและข้อมูลแยกจากกันอย่างละ 64 กิโลไบต์

5) มีไทเมอร์เคาน์เตอร์ ขนาด 16 บิต 2 ชุด (สำหรับเบอร์ 8052 มี 3 ชุด) ทำงานได้ 4 โหมด

6) รับอินเตอร์รัพท์ได้ 6 แหล่ง 5 เวกเตอร์ สำหรับเบอร์ 8052 ขึ้นไปมี 8 แหล่ง 6 เวกเตอร์

7) มีพอร์ตรับส่งข้อมูลอนุกรม (UART) 2 พอร์ตแบบ Full Duplex เลือกรูปได้ 4 โหมด

8) มีคำสั่งในการทำ AND, OR หรือ Complement ได้ทั้งแบบ 8 บิตและ 1 บิต

9) มีวงจรออสซิลเลเตอร์ภายใน

2.2.2 โครงสร้างของไมโครคอนโทรลเลอร์ MCS-51

ภายใน 8051 จะประกอบด้วยเกตชนิดต่างๆ เช่น วงจรถอดรหัสคำสั่ง, วงจรสัญญาณนาฬิกา เป็นต้น โครงสร้างภายในของ 8051 จะประกอบด้วยส่วนย่อยๆ ดังรูปที่ 2.1 โครงสร้างของ 8051 จะประกอบด้วย 3 ส่วน หลักๆ ดังนี้

1) ซีพียู (Central Processing Unit) ส่วนนี้ทำหน้าที่สร้างสัญญาณควบคุมการติดต่อกับส่วนอื่นๆ เรียกว่า วงจรควบคุม (Control Unit) สัญญาณที่สร้างจากวงจรควบคุม ได้แก่ สัญญาณสำหรับการติดต่อกับหน่วยความจำ, อุปกรณ์รับข้อมูลเข้าหรือส่งข้อมูลออก ซึ่งส่วนควบคุมการขัดจังหวะ และส่วนควบคุมบัสก็เป็นส่วนหนึ่งของวงจรควบคุมด้วย การสร้างสัญญาณวงจรควบคุมจากซีพียูนี้ จะสร้างสัญญาณโดยการถอดรหัสจากคำสั่งที่มีการกำหนดไว้และสัญญาณที่สร้างขึ้นมาจะอ้างอิงกับสัญญาณนาฬิกา ที่สร้างขึ้นจากวงจรออสซิลเลเตอร์ เพื่อให้ทุกๆ ส่วนทำงานประสานกันอย่างถูกต้อง ในซีพียูยังประกอบด้วยส่วนประมวลผล (Arithmetic Logic Unit) ทำหน้าที่ประมวลผลข้อมูล เช่น การบวก, การลบ, การคูณหรือการหาร ข้อมูลแล้วนำผลลัพธ์ไปเก็บไว้ในหน่วยความจำที่ต้องการ

2) หน่วยความจำ (Memory) มีไว้สำหรับจดจำข้อมูล ซึ่งในการนำข้อมูลเข้าและออกจากหน่วยความจำ จำเป็นต้องรู้ตำแหน่งของหน่วยความจำ (Address) ในการนำข้อมูลเข้าไปเก็บในหน่วยความจำเรียกว่า การเขียนข้อมูลและการนำเข้าข้อมูลออกจากหน่วยความจำ เรียกว่า การอ่านข้อมูล ในไมโครคอนโทรลเลอร์ 8051 ข้อมูลในแต่ละตำแหน่งจะมีขนาด 8 บิต ดังนั้นแต่ละ

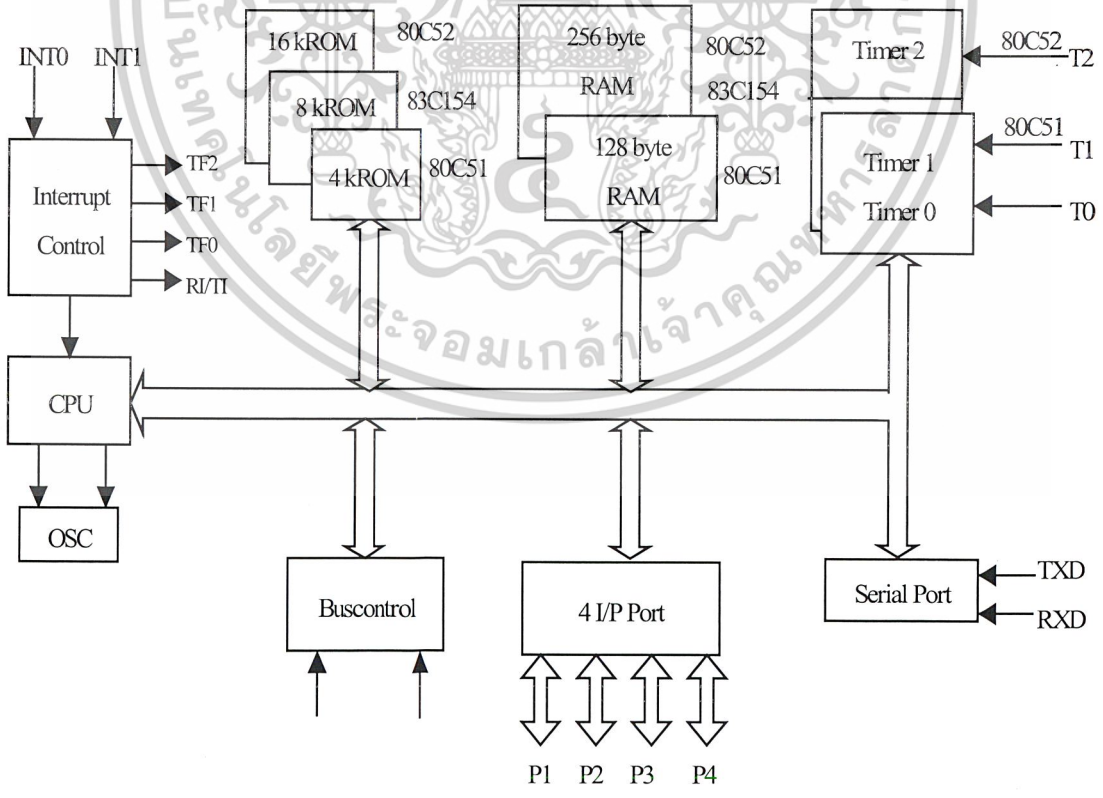
ตำแหน่งของหน่วยความจำสามารถเก็บข้อมูลมีค่าได้ระหว่าง 00000000_2 ถึง 11111111_2 หรือ 00H ถึง 0FF ในการติดต่อหน่วยความจำจะต้องมีสัญญาณ 3 กลุ่ม คือ

2.1) ตำแหน่งที่ต้องการติดต่อกับหน่วยความจำ ซึ่งในไมโครคอนโทรลเลอร์ 8051จะมีหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลที่มีขนาดสูงสุดชนิดละ 65536 ตำแหน่ง (64 กิโลไบต์) ดังนั้นการอ้างตำแหน่งของหน่วยความจำจะต้องใช้เส้นแสดงตำแหน่งของหน่วยความจำ จะต้องใช้เส้นแสดงตำแหน่งในเลขฐานสอง ทั้งหมด 16 เส้น (2^{16} เท่ากับ 65536)

2.2) ข้อมูลที่อ่านหรือเขียนกับหน่วยความจำในตำแหน่งที่เราต้องการ

2.3) สัญญาณควบคุมจะส่งไปยังหน่วยความจำ เพื่อจะบอกกับหน่วยความจำว่าต้องอ่านหรือเขียนข้อมูล โดยวงจรถอดรหัสคำสั่ง จะทำการสร้างสัญญาณควบคุมจากคำสั่งที่อ่านเข้ามาจากหน่วยความจำโปรแกรม

3) อุปกรณ์อินพุต/เอาต์พุต (Input/Output Device) เป็นส่วนใช้ส่งข้อมูลเข้าหรือนำข้อมูลออกจากไมโครคอนโทรลเลอร์ 8051 ทำให้สามารถติดต่อกับอุปกรณ์ภายใน อุปกรณ์อินพุตเอาต์พุต ได้แก่ อินพุต/เอาต์พุตพอร์ตแบบขนาน วงจรนับเวลา/จับเวลา 0 วงจรนับเวลา/จับเวลา 1 พอร์ตสื่อสารอนุกรม



รูปที่ 2.1 โครงสร้างของ 8051

3.1) พอร์ตขนานเป็นที่สำหรับใช้รับส่งข้อมูลซึ่งเป็นสัญญาณดิจิตอลเข้าหรือออกจากตัวไมโครคอนโทรลเลอร์ 8051 มีทั้งหมด 4 พอร์ต โดยแต่พอร์ตจะรับส่งข้อมูลได้ 8 บิต มีพอร์ต P0, P1, P2 และ P3 บางพอร์ตใช้งานได้มากกว่า 1 อย่าง

3.2) วงจรนับเวลา/จับเวลา 0 และวงจรนับเวลา/จับเวลา 1 เป็นวงจรที่สามารถทำการนับจำนวนไซเคิลของสัญญาณที่ต่อจากภายนอกของไมโครคอนโทรลเลอร์ 8051 หรือจำนวนของสัญญาณนาฬิกาภายในไมโครคอนโทรลเลอร์ 8051 ก็ได้ สามารถตั้งค่าเริ่มต้นของการนับและการค่าการนับได้โดยซีพียู

3.3) พอร์ตอนุกรม ซีพียูจะอ่านข้อมูล พอร์ตอนุกรมเป็นแบบ 8 บิต แต่ละข้อมูลจะถูกส่งออกมาจากไมโครคอนโทรลเลอร์ 8051 เรียงไปทีละบิตออกจากขา TXD ในการรับส่งข้อมูลก็จะรับเข้ามาทีละบิตทางขา RXD และจัดเรียงใหม่เป็น 8 บิต เพื่อให้ซีพียูอ่านไปใช้งานต่อไปในไมโครคอนโทรลเลอร์ 8051 มีพอร์ตใช้งานได้หลายแบบทำให้สะดวกแก่การนำไปใช้งานต่างๆ ได้มากมาย การนำพอร์ตไปใช้งานจะต้องเขียนโปรแกรมขึ้นมาควบคุม

2.3 ความรู้เบื้องต้นเกี่ยวกับพอร์ตอนุกรม

การเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพ่วงภายนอกหรือคอมพิวเตอร์ด้วยกันมีกัน 2 รูปแบบ คือ รับส่งข้อมูลแบบขนานและรับส่งข้อมูลแบบอนุกรม

การรับส่งข้อมูลแบบขนานเป็นการรับและส่งข้อมูลครั้งละ 4 ถึง 8 บิตในเวลาเดียวกัน ทำให้การรับและส่งข้อมูลมีความเร็วสูง แต่จำนวนสายที่ใช้ในการถ่ายทอดข้อมูลมีมากเท่ากับจำนวนบิตของข้อมูลที่ทำการถ่ายทอด นอกจากนั้นยังมีสายที่ใช้สำหรับควบคุมและตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตข้อมูล

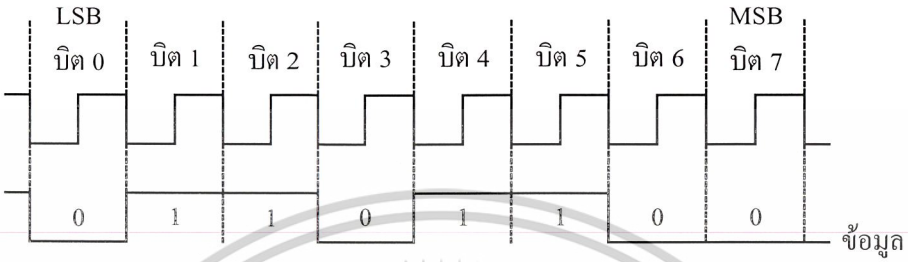
ในขณะที่การรับส่งข้อมูลแบบอนุกรมจะเป็นการรับส่งข้อมูลครั้งละ 1 บิต โดยมีรูปแบบการรับส่งที่เป็นมาตรฐาน ต้องมีการตรวจสอบความพร้อมในการรับและส่งข้อมูลของตัวส่งและตัวรับ การรับส่งข้อมูลแบบอนุกรมมีข้อดีในเรื่องจำนวนสายสัญญาณที่น้อยมากและไม่แปรผันตามจำนวนบิตของข้อมูล ระยะทางในการรับส่งข้อมูลสูงกว่าแบบขนานมาก

การสื่อสารแบบอนุกรมแบ่งออกเป็น 2 แบบ คือ การสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส

2.3.1 การสื่อสารข้อมูลแบบซิงโครนัส

การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วยตัวอย่างการส่งข้อมูลแบบซิงโครนัส คือ คีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของ

สัญญาณนาฬิกา ส่วนสายอีกเส้นจะเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้น คือ สัญญาณนาฬิกา, ข้อมูลและกราวด์ รูปที่ 2.1 แสดงให้เห็นถึงไดอะแกรมเวลาของการสื่อสารข้อมูลแบบซิงโครนัส



รูปที่ 2.2 ไดอะแกรมเวลาของการสื่อสารข้อมูลแบบซิงโครนัส

2.3.2 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัส คือ การรับและส่งข้อมูลโดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วย แต่จะใช้การกำหนดค่าอัตราความเร็วในการรับและส่งข้อมูลให้มีค่าเท่ากัน ซึ่งเรียกอัตราเร็วนี้ว่า อัตราบอดหรือบอดเรต (Baud Rate) มีหน่วยเป็น บิตต่อวินาที

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกัน คือ

1) บิตเริ่มต้น

2) บิตข้อมูลแบบอนุกรม มีขนาด 5, 6, 7 หรือ 8 บิต

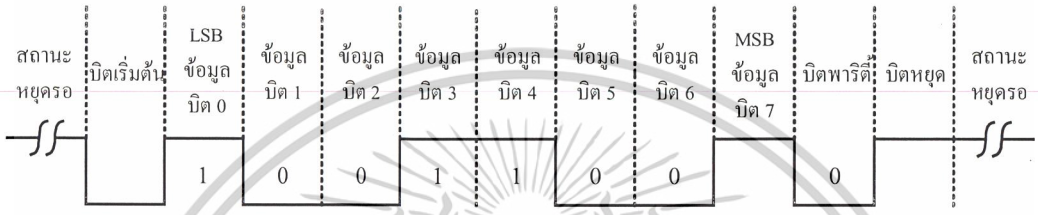
3) บิตตรวจสอบพาริตี้ (Parity Bit) มีขนาด 1 บิต หรือไม่มีบิต

4) บิตปิดท้ายหรือบิตหยุด (Stop Bit) มีขนาด 1, 1.5, หรือ 2 บิต

รูปที่ 2.3 แสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส เมื่อไม่มีการส่งข้อมูลขา DATA จะมีสถานะลอจิก “1” เรียกสถานะนี้ว่า สถานะหยุดรอ (Waiting Stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา DATA มีลอจิก “0” ด้วยช่วงระยะเวลา 1 บิต เรียกบิตนี้ว่าบิตเริ่มต้น (Start Bit) จากนั้นบิตข้อมูลจะถูกส่งออกไปโดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุดหรือบิต LSB ก่อน ซึ่งข้อมูลที่ต้องการส่งอาจมีจำนวน 5, 6, 7 หรือ 8 บิตก็ได้ จากนั้นตามด้วยบิตพาริตี้ (Parity Bit) ซึ่งใช้ในการตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่ง คือ บิตปิดท้ายหรือบิตหยุด (Stop Bit) โดยจะเป็นการทำให้ขา DATA มีสถานะลอจิก “1” อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสหรืออัตราบอดหรือบอดเรตที่ใช้สำหรับพอร์ตอนุกรม RS-232 มีด้วยกันหลายค่า ได้แก่ 110, 150, 300, 600,

1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที โดยมีค่ามากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ เนื่องจากบอดเรต คือ ค่าของจำนวนบิตที่สามารถส่งได้ใน 1 วินาที สมมติว่า ข้อมูลอนุกรมมีขนาด 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูล 1 ไบต์ จะมีความยาวเท่ากับ 10 บิต ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9,600 บิตต่อวินาที ก็จะสามารถส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที



รูปที่ 2.3 รูปแบบของข้อมูลแบบอะซิงโครนัส

การตรวจสอบพาริตี้สามารถกำหนดเป็นแบบคี่ (Odd), แบบคู่ (Even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ พาริตี้คี่หรือพาริตี้คู่แสดงถึงจำนวนลอจิก “1” ทั้งหมดภายในข้อมูลที่ส่งไป 1 ไบต์ รวมพาริตี้ว่ามีจำนวนเป็นเลขคู่หรือเลขคี่ ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิต มีค่าเท่ากับ 99H หรือ 10011001B จะเห็นว่าข้อมูลในไบต์มีจำนวนลอจิก “1” จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้น ถ้ากำหนดค่าพาริตี้เป็นคู่ ค่าของพาริตี้บิตจะต้องมีลอจิกเป็น “0” แต่ถ้ากำหนดพาริตี้เป็นคี่ ค่าของบิตพาริตี้จะต้องเป็น “1” เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตี้เป็นคี่

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART (Universal Asynchronous Receiver Transmitter) ซึ่งทางภาครับจะต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ที่ตรงกันเอาไว้ว่าจะตรวจสอบพาริตี้คี่หรือพาริตี้คู่ โดยการนับจำนวนลอจิก “1” ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแสดงข้อผิดพลาดออกมาให้ผู้ใช้ทราบ กระบวนการดังกล่าวเป็นวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้นในการรับส่งข้อมูลที่ง่ายที่สุด แต่มันสามารถตรวจสอบได้เมื่อมีบิตข้อมูลที่ทำการรับส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ไอซี UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ไอซี UART เบอร์ 8250 ไอซี UART เหล่านี้มีระดับแรงดันของลอจิกเป็นแบบทีทีแอล (+5 โวลต์) แต่เพื่อให้แรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูล

สามารถทำได้ที่ระยะไกลมากยิ่งขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิก “0” จะมีระดับแรงดัน –3 โวลต์ ถึง –12 โวลต์ และลอจิก “1” มีระดับแรงดัน +3 โวลต์ ถึง +12 โวลต์

2.4 พอร์ตอนุกรม RS-232

มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้ส่งผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดหนึ่งซึ่งอยู่ห่างไกลโดยสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ได้วางมาตรฐานที่มีชื่อเรียกกันว่า EIA RS232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ –3 โวลต์ ถึง –12 โวลต์ แสดงว่ามีข้อมูล (mark) และ +3 โวลต์ ถึง +12 โวลต์ แสดงว่าเป็นช่องว่าง (space)

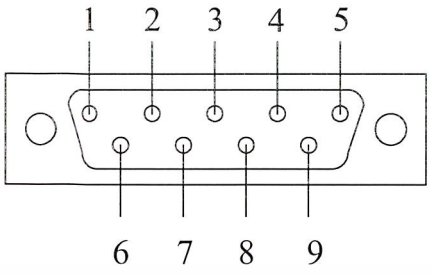
มาตรฐาน RS-232 ถูกใช้ในการกำหนดรูปแบบการสื่อสารข้อมูลกันระหว่างอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating DCE) อุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรลเลอร์หรือไมโครคอมพิวเตอร์ซึ่งมีความสามารถในการสร้างบิตข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE ทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัด คือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ที่โมเด็มจะเป็นแบบ DCE

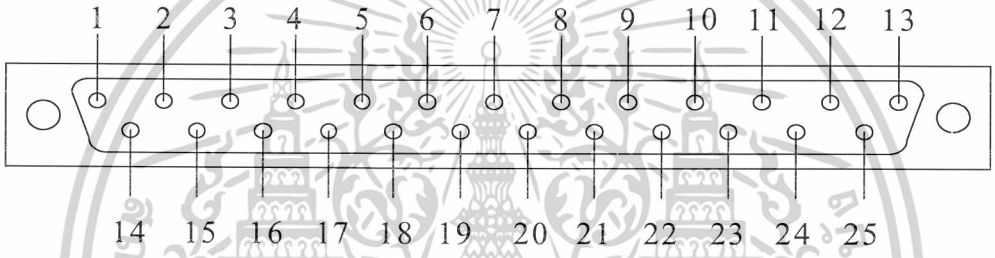
สำหรับการใช้งานคอมพิวเตอร์ พอร์ตอนุกรม RS-232 ถูกใช้เพื่อเชื่อมต่อกับโมเด็ม, เมาส์ และเครื่องพิมพ์ที่สามารถติดต่อทางพอร์ตอนุกรมได้

2.4.1 คอนเน็กเตอร์สำหรับพอร์ตอนุกรม RS-232

มาตรฐานการเชื่อมต่อแบบ RS232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้ หรือ DB-9 ตัวผู้ ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้น เช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่นๆ ที่เคยมีใช้งานมาในอดีตไม่ค่อยมีความสำคัญมากนักจึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาในรูปที่ 2.4



(ก) คอนเน็กเตอร์อนุกรม 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)



(ข) คอนเน็กเตอร์อนุกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

รูปที่ 2.4 คอนเน็กเตอร์อนุกรม

2.4.2 การเชื่อมต่อคอนเน็กเตอร์สำหรับพอร์ตอนุกรม RS-232

สำหรับการเชื่อมต่อสายระหว่างคอมพิวเตอร์กับอุปกรณ์ภายนอกแสดงดังในรูปที่ 2.5 ลูกศรในรูป แสดงถึงทิศทางของข้อมูล การเชื่อมต่อในรูปที่ 2.5 (ก) เป็นการเชื่อมต่อแบบ NULL MODEM หรือการเชื่อมต่อโดยตรงโดยไม่ต้องผ่านโมเด็ม ส่วนการเชื่อมต่อในรูปที่ 2.5 (ข) เป็นการเชื่อมต่อโดยใช้สัญญาณน้อยที่สุดเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูลและเส้นสุดท้ายเป็นกราวด์

ขา Data Carrier Detect : DCD หรืออาจเรียกว่า Carrier Detect : CD ขานี้จะแอกตีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูล เช่น โมเด็ม สำหรับใช้งานปกติ ขานี้จะไม่ถูกนำมาใช้งานมากนัก

ขา Recceive Data : RD หรือ RXD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยจะนำข้อมูลที่อ่านได้ไปเก็บไว้ในรีจิสเตอร์บัฟเฟอร์

ตารางที่ 2.2 การจัดขาสัญญาณของพอร์ตอนุกรมในแบบต่างๆ และหน้าที่การทำงาน

คอนเน็คเตอร์ DB-9	คอนเน็คเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	Data Carrier Detect : DCD	อินพุต
2	3	Received Data : RXD	อินพุต
3	2	Transmitted Data : TXD	เอาต์พุต
4	20	Data Terminal Ready : DTR	เอาต์พุต
5	7	Single Ground : GND	-
6	6	Data Set Ready : DSR	อินพุต
7	4	Request To Send : RTS	เอาต์พุต
8	5	Clear To Send : CTS	อินพุต
9	22	Ring Indicator : RI	อินพุต

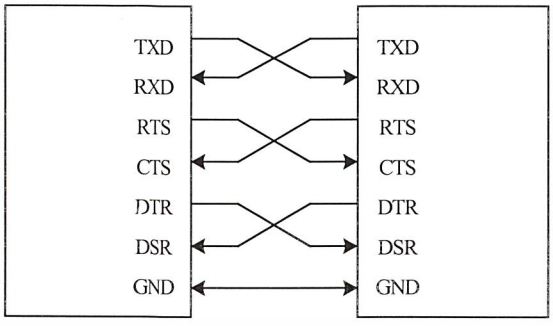
ขา Transmitted Data : TD หรือ TXD ขานี้ใช้เพื่อส่งข้อมูลอนุกรมออกจากคอมพิวเตอร์ โดยการนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

ขา Data Terminal Ready : DTR เป็นขาเอาต์พุตที่ใช้สำหรับส่งสัญญาณออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าต้องการติดต่อกับอุปกรณ์ปลายทาง โดยขา DTR นี้จะต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทาง และขา DTR ของอุปกรณ์ปลายทางจะต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์และถ้าใช้การเชื่อมต่อแบบ 3 สาย ต้องเชื่อมต่อขา DTR และ DSR ของพอร์ตอนุกรมเข้าด้วยกัน และจะต้องต่อเชื่อมเข้ากับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

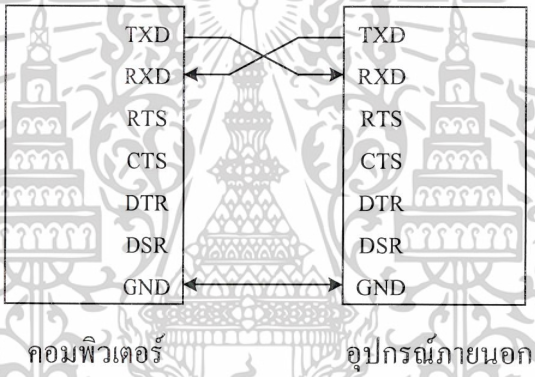
ขา Signal Ground : GND เป็นขากราวด์ของสัญญาณ

ขา Data Set Ready : DSR ขานี้จะใช้ควบคู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอก

ขา Request To Send : RTS เป็นขาเอาต์พุตสำหรับส่งสัญญาณร้องขอให้อุปกรณ์ปลายทางส่งข้อมูลมาให้คอมพิวเตอร์โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ซึ่งในกรณีที่มีการเชื่อมต่อแบบ 3 สาย จะต้องเชื่อมต่อขา RTS และ CTS เข้าด้วยกัน เพื่อให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา



(ก) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ Null Modem



(ข) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ RS-232
ในลักษณะที่ใช้สายสัญญาณน้อยที่สุดเพียง 3 เส้น

รูปที่ 2.5 การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมในรูปแบบต่างๆ

ขา Clear To Send : CTS เป็นขาอินพุตทำหน้าที่รอรับสัญญาณที่ส่งเข้ามา เมื่อมีการส่งสัญญาณเข้ามาที่ขานี้ ข้อมูลที่ขา TXD จะถูกส่งออกไป ขานี้จะใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลแล้วหรือยัง

ขา Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มแล้วยังมีความต้องการตรวจสอบสัญญาณเรียกจากสายโทรศัพท์

2.4.3 รีจีสเตอร์ของพอร์ตอนุกรม RS-232

UART (Universal Asynchronous Receiver Transmitter) เป็นอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัส สำหรับการสื่อสารแบบอนุกรมบนคอมพิวเตอร์ถือว่า UART เป็นหัวใจสำคัญของการสื่อสารแบบอนุกรม

หน้าที่หลักของ UART คือ ทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัส แล้วส่งออกไปและทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามา UART ให้เป็นขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจกแจงข้อมูลอื่นๆ ให้คอมพิวเตอร์รับทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต), รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพารีตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ (programmable buadrate generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารอยู่ในช่วง 1 – 65,535 ซึ่ง UART สามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็กซ์ (half duplex) และฟูลดูเพล็กซ์ (full duplex) โดยการส่งแบบฮาล์ฟดูเพล็กซ์เป็นการส่งข้อมูลแบบทิศทางเดียว ส่วนการส่งข้อมูลแบบฟูลดูเพล็กซ์นั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

ในเครื่องคอมพิวเตอร์โดยทั่วไปมี UART ที่ใช้งานกันอยู่ 2 เบอร์ คือ 8250 ซึ่งเป็น UART มาตรฐานที่มีใช้กันมายาวนาน UART เบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเดียวกัน ทำให้การรับและส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้น แต่ UART เบอร์นี้ถือว่าเป็นต้นแบบของ UART ที่ใช้กันในคอมพิวเตอร์ คอมพิวเตอร์ทุกๆ รุ่นจะต้องสนับสนุนการทำงานตามรูปแบบของ UART เบอร์นี้

UART อีกเบอร์หนึ่ง คือ 16450 มีความสามารถรับส่งข้อมูลได้ที่ความเร็ว 115200 บิตต่อวินาที และเพิ่มรีจีสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิฟต์รีจีสเตอร์แบบ FIFO (First In First Out) ขนาด 16 ไบต์เข้าไป ทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ 256 กิโลบิตต่อวินาทีได้ โดยคอมพิวเตอร์ในปัจจุบันใช้ UART เบอร์นี้หรือใหม่กว่า เช่น เบอร์ TL164750 ซึ่งมีรีจีสเตอร์แบบ FIFO ขนาด 64 ไบต์ ทำงานได้ที่ระดับแรงดัน +5 โวลต์ และ +3 โวลต์ มีโหมดประหยัดพลังงาน สามารถรับส่งข้อมูลได้ที่ความเร็ว 1 เมกะบิตต่อวินาที เมื่อใช้สัญญาณนาฬิกา 16 เมกะเฮิร์ต

ความเร็ว ในการส่งข้อมูลที่มากมายของ UART เบอร์ใหม่ๆ ก็ไม่ได้ช่วยการรับส่งข้อมูลของคอมพิวเตอร์เร็วขึ้น เนื่องจากว่าคอมพิวเตอร์ยังใช้ความถี่ของสัญญาณนาฬิกาในการแปลงข้อมูลเพียง 1.8432 เมกะเฮิร์ต

เครื่องคอมพิวเตอร์โดยทั่วไปสามารถต่อพอร์ตอนุกรมสูงสุดได้ 4 พอร์ต มีชื่อเรียกว่าเป็น COM1, COM2, COM3 และ COM4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างก็ใช้งาน UART ภายในคอมพิวเตอร์ในการติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน ในแผนผังการทำงานภายในพอร์ตอนุกรมประกอบด้วยรีจีสเตอร์ 8 บิต 8 ตัว ที่ใช้งานร่วมกับ UART แอดเดรสของรีจีสเตอร์ภายในพอร์ตอนุกรมสามารถคำนวณได้จากค่ารีจีสเตอร์พื้นฐานของพอร์ตอนุกรม ตัวอย่าง พอร์ตอนุกรม COM1 มีแอดเดรสอยู่ที่ 3F8H ตำแหน่งของรีจีสเตอร์ต่างๆ จะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยมีรีจีสเตอร์ที่ใช้งานพอร์ตอนุกรมมีดังนี้

00H เป็นรีจีสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามาหรือเตรียมข้อมูลก่อนที่จะส่งออก

01H รีจีสเตอร์เอ็นเอเบิลการอินเตอร์รัพต์ ใช้เซตโหมดการอินเตอร์รัพต์ของพอร์ตอนุกรม

02H รีจีสเตอร์แสดงโหมดการอินเตอร์รัพต์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัพต์

03H รีจีสเตอร์สำหรับกำหนดรูปแบบของข้อมูล

04H รีจีสเตอร์ควบคุมโมเด็ม ใช้ตรวจสอบบิตสำหรับติดต่อกับโมเด็ม เช่น RTS หรือ DTR

05H รีจีสเตอร์แสดงสถานะการรับและการส่งข้อมูลแบบอนุกรม

06H รีจีสเตอร์แสดงสถานะของโมเด็ม ซึ่งจะแสดงสถานะของขา DCD, RI, DSR และ CTS

06H รีจีสเตอร์สำหรับการเก็บข้อมูลชั่วคราว

2.5 ความรู้เบื้องต้นของ RS-485

RS-485 เป็นรูปแบบที่นำมาจากรูปแบบของ RS-422 ทุกประการ เพียงแต่มีการดัดแปลงให้ใช้สายสัญญาณเพียง 1 คู่สายบิดเกลียว โดยการใช้เป็นทั้งสายส่งและสายรับในสายคู่เดียวกัน ซึ่งการที่จะทำงานในลักษณะนี้ได้นั้น ที่ขั้วของอุปกรณ์ทุกตัวจะต้องมีการต่อบัฟเฟอร์ที่สามารถปรับให้เป็นความต้านทานสูงได้ คือ สามารถตัดออกจากวงจรชั่วคราวได้ เพื่อใช้ในการบริหารสัญญาณและมักจะเป็นการสื่อสารในลักษณะของเครื่องแม่ข่ายและเครื่องลูกข่ายคือ จะมีเครื่องแม่ข่ายเป็นตัว

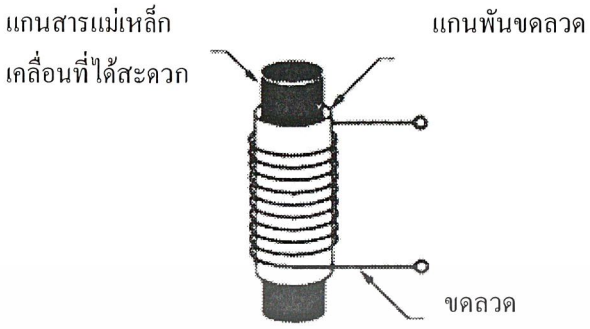
หลักอยู่1 ตัว ซึ่งจะคอยควบคุมดูแลและสั่งการทั้งหมด ส่วนเครื่องลูกข่ายที่ต่ออยู่ทุกตัวก็จะรอรับการสั่งงานจากเครื่องแม่ข่ายเครื่องลูกข่ายแต่ละตัวจะต้องมีชื่อเรียก เพื่อที่เครื่องแม่ข่ายจะเรียกใช้ในการสั่งงาน มีอีกวิธีการหนึ่ง ซึ่งได้ถูกพัฒนาขึ้นใหม่ คือ การสื่อสารกันโดยไม่มีเครื่องแม่ข่ายหลัก คือทุกตัวในโครงข่ายเป็นเครื่องแม่ข่ายทั้งหมด แต่สามารถส่งข้อมูลด่วนออกมาควบคุมตัวอื่นได้โดยฉับพลัน จากการที่ได้ทำการทดลองได้ผลสถิติว่ารูปแบบหลังนี้สามารถลดการสื่อสารโดยไม่จำเป็นลงได้ถึง 30% ซึ่งขึ้นอยู่กับการพัฒนาซอฟต์แวร์ที่ซับซ้อน

ตารางที่ 2.3 คุณสมบัติของ RS-232 และ RS-485

ข้อมูลของแต่ละรูปแบบ	RS-232	RS-485
หมวดการทำงาน	Single Ended	Differential
จำนวนตัวลูกสูงสุด	1 Drive 1 Recp	1 Drive 32 Recp
ความยาวสูงสุดของสายเคเบิ้ล	50 ฟุต	4,000 ฟุต
อัตราความเร็วการส่งข้อมูล	20 Kbit/sec	10 Mbit/sec
แรงดันส่งออกสูงสุดเอาต์พุต	+/-25 V	-7V ถึง +12V
ค่าความต้านทานโหลด (Load)	3 – 7 KΩ	54 Ω
ความไวในการรับ	+/-3 V	+/-200 mV
ความต้านทานอินพุต	3 – 7 KΩ	มากกว่า 12 KΩ

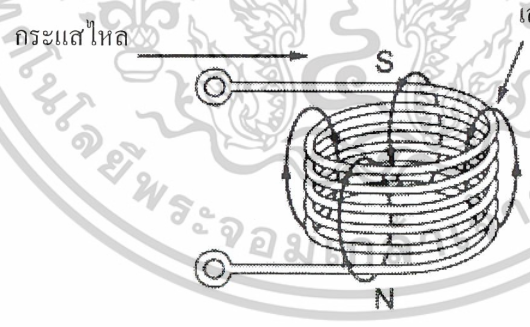
2.6 โซลินอยด์

โซลินอยด์สามารถที่จะประยุกต์ใช้กับงานด้านต่างๆ ที่ต้องการเชื่อมโยงพลังงานทางไฟฟ้ามาเป็นพลังงานกลโดยตรง โดยที่สัญญาณไฟฟ้าที่ป้อนเข้ามาทางขดลวดจะทำให้แกนสารแม่เหล็กของโซลินอยด์เกิดการเคลื่อนที่ขึ้น และการเคลื่อนที่นี้สามารถนำไปใช้ประโยชน์ได้หลายอย่าง เช่น ขัดกลอนประตู เอาไว้ไปถีบกระเดื่องทำให้กลไกทำงานหรือหยุดทำงาน



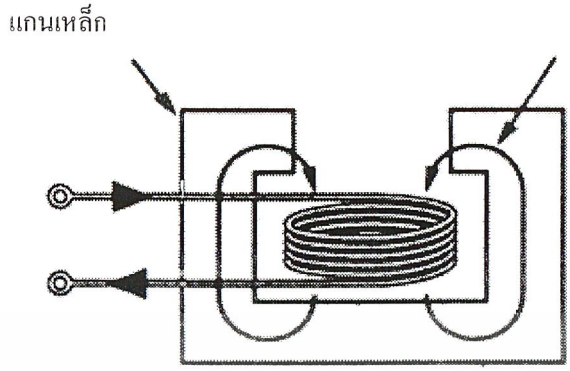
รูปที่ 2.6 โครงสร้างของโซลินอยด์

หลักการทำงานของโซลินอยด์ คือ เมื่อมีกระแสไฟฟ้าไหลในขดลวดตัวนำใดๆ ก็จะเกิดสนามแม่เหล็กไฟฟ้าขึ้นรอบๆ ตัวนำนั้นและถ้านำขดลวดที่ยาวกว่ากันมาขดเป็นวงหลายวงก็จะทำให้เกิดลักษณะของขดลวดขึ้น และสนามแม่เหล็กที่เกิดจากขดลวดแต่ละขดก็จะอยู่ในทิศทางที่เสริมกันทำให้ก็จะเกิดเส้นแรงของสนามแม่เหล็กถาวรขึ้น ซึ่งพร้อมที่จะดูดสารแม่เหล็กทันทีเนื่องจากสภาพรอบๆ ขดลวดเป็นอากาศเส้นแรงแม่เหล็กจึงไม่เข้มข้นนัก

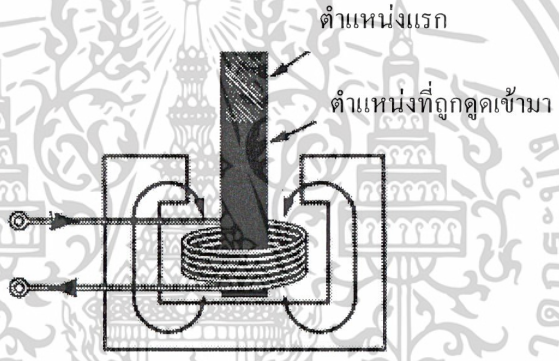


รูปที่ 2.7 ทิศทางของสนามแม่เหล็กที่เกิดขึ้นในขดลวดที่มีกระแสไหล

เพื่อไม่ให้สนามแม่เหล็กเกิดการกระจายจึงใส่แกนเหล็กอ่อนรูปตัว C เข้ามารอบๆ ขดลวดเพื่อให้สนามแม่เหล็กมากขึ้นดังรูปที่ 2.9 และถ้าเอาแกนกระทุ้ง (plunger) มาใส่เข้าไปตรงกลางขดลวดในตำแหน่งที่ 1 แกนกระทุ้งจะถูกดูดให้ลึกในตำแหน่งที่ 2 ยิ่งระยะใกล้มากเท่าไรแรงดูดก็จะยิ่งมากขึ้นตามระยะของแท่งแกนเหล็ก



รูปที่ 2.8 การเพิ่มเหล็กอ่อนเข้ามาเพื่อเพิ่มความเข้มของสนามแม่เหล็ก



รูปที่ 2.9 การเคลื่อนที่ของแกนกระทุ้ง

2.7 บัตรแถบแม่เหล็ก

บัตรแถบแม่เหล็กที่ใช้อยู่ทั่วไปก็จะเป็นประเภทบัตรของธนาคารต่างๆ เช่น บัตรเอทีเอ็ม, บัตรเครดิต, บัตรเดบิตและบัตรประเภทอื่นๆ ซึ่งบัตรนี้สามารถบันทึกได้แค่ตัวเลขหรือหมายเลขประจำบัตรเท่านั้นไม่สามารถที่จะบรรจุข้อมูลมากๆได้ ส่วนหลักการทำงาน คือ เมื่อเครื่องอ่านแถบแม่เหล็กบนบัตรเครื่องก็จะทำการส่งข้อมูลไปที่ศูนย์เพื่อทำการค้นหา ประมวลผลและส่งข้อมูลที่เป็นผลลัพธ์กลับออกมาทางหน้าจอเครื่องคอมพิวเตอร์หรือเครื่องลูกข่ายอื่นๆ ข้อดีของบัตรแถบ แม่เหล็กก็คือ ช่วยให้ผู้ใช้บัตรไม่ต้องจำรหัสประจำตัว หรือรหัสบัตรที่มีหลายตัวเพื่อป้องกันความผิดพลาดในการใส่รหัสผิด ส่วนข้อเสียก็คือ บัตรแถบแม่เหล็กไม่ทนต่อสื่อที่เป็นสนามแม่เหล็กเมื่อเข้าใกล้บริเวณที่มีสนามแม่เหล็กแรงมาก ก็อาจจะทำให้ข้อมูลบนบัตรสูญหายหรือผิดพลาดได้และใน

การติดต่อกับศูนย์ข้อมูลต้องมีสายสัญญาณในการติดต่อสื่อสารระหว่างศูนย์กับคอมพิวเตอร์หรือเครื่องลูกข่ายซึ่งจะต้องเสียค่าใช้จ่ายเพิ่ม

บัตรแถบแม่เหล็กที่ใช้ในโครงงานนี้ จะใช้มาตรฐานเดียวกับบัตรแม่เหล็กเอทีเอ็มที่ใช้ในธนาคารทั่วไป โดยรูปแบบของข้อมูลที่บันทึกบนแถบแม่เหล็กและคุณสมบัติต่างๆ ของบัตรแถบแม่เหล็กจะเป็นตามมาตรฐานสากล ISO 7810 - 7813 เพื่อที่จะสามารถพัฒนาระบบต่อไปได้

รูปที่ 2.10 ขนาดมาตรฐานของบัตรแถบแม่เหล็ก

จากรูปที่ 2.10 ตามมาตรฐานของ ISO เราสามารถพิจารณาแทร็กต่างๆที่ใช้งานได้เป็น 3 แทร็กโดยมีรายละเอียดในการใช้งานแต่ละแทร็กดังนี้

แทร็กที่ 1 เป็นแทร็กที่สามารถทำการอ่านข้อมูลได้อย่างเดียว เรียกว่า Read only โดยข้อมูลประกอบด้วยตัวอักษรและตัวเลข อยู่ในช่วงบริเวณเส้นขนาน a และเส้นขนาน b

แทร็กที่ 2 เป็นแทร็กที่สามารถอ่านข้อมูลได้เพียงอย่างเดียวเช่นกันแต่ข้อมูลจะมีลักษณะเป็นตัวเลขเพียงอย่างเดียว อยู่ในช่วงเส้นขนาน b และเส้นขนาน c ซึ่งเป็นแทร็กที่ใช้งานทั่วไป

แทร็กที่ 3 เป็นแทร็กที่สามารถทำการอ่านและเขียนข้อมูลลงไปได้ ซึ่งเก็บข้อมูลที่เป็นตัวเลข ซึ่งอยู่ในช่วงเส้นขนาน c และเส้นขนาน d

2.5.1 ชุดรหัสข้อมูลของบัตรแถบแม่เหล็ก

ในการใช้งานโดยทั่วไป ของบัตรแถบแม่เหล็กจะใช้งานในส่วนของแทร็กที่ 2 ถ้าเป็นบัตรเอทีเอ็มของธนาคาร ในส่วนของแทร็กนี้จะเป็นการเก็บข้อมูลเกี่ยวกับข้อมูลของธนาคารและเลขที่บัญชี ดังนั้นแต่ละธนาคารจึงสามารถใช้บริการร่วมกันได้

ข้อมูลที่บันทึกในแทร็กที่ 2 ของบัตรแม่เหล็กจะเป็นตัวเลขเพียงอย่างเดียว โดยที่ตัวเลขหนึ่งตัวจะประกอบด้วยบิตข้อมูลแบบ BCD 4 บิตและบิตพาริตี้ 1 บิตซึ่งใช้ในการตรวจสอบข้อมูลแต่ละตัวเลขโดยตรวจสอบแบบพาริตี้คี่ตามมาตรฐาน ISO ที่ระบุไว้ว่าจำนวนข้อมูลสูงสุดในแทร็กที่ 2 จะมีได้ไม่เกิน 40 ตัว (รวมสัญลักษณ์การเริ่มต้นและสิ้นสุด) ในส่วนของชุดข้อมูลตัวเลขแต่ละตัวสำหรับแทร็กที่ 2 แสดงไว้ในตารางที่ 2.3

ตารางที่ 2.4 รหัสข้อมูลแต่ละตัวในแทร็กที่ 2

P	b4	b3	b2	b1	รหัส
1	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
1	0	0	1	1	3
0	0	1	0	0	4
1	0	1	0	1	5
1	0	1	1	0	6
0	0	1	1	1	7
0	0	1	0	0	8
1	1	0	0	1	9
1	1	0	1	0	A
0	1	0	1	1	B1
1	1	1	0	0	A
0	1	1	0	1	B2
0	1	1	1	0	A
1	1	1	1	1	B3

2.5.2 รูปแบบข้อมูลบัตรแถบแม่เหล็ก

รูปแบบของการบันทึกข้อมูลในบัตรแม่เหล็กที่ใช้กันโดยทั่วไป ซึ่งแต่ละสัญลักษณ์มีความหมายดังรูปที่ 2.11

SYN	START	DATA	SEP	DATA	-----	STOP	LRC	SYNC

รูปที่ 2.11 รูปแบบของข้อมูลที่บันทึกในแทร็กที่ 2 บนบัตรแม่เหล็ก

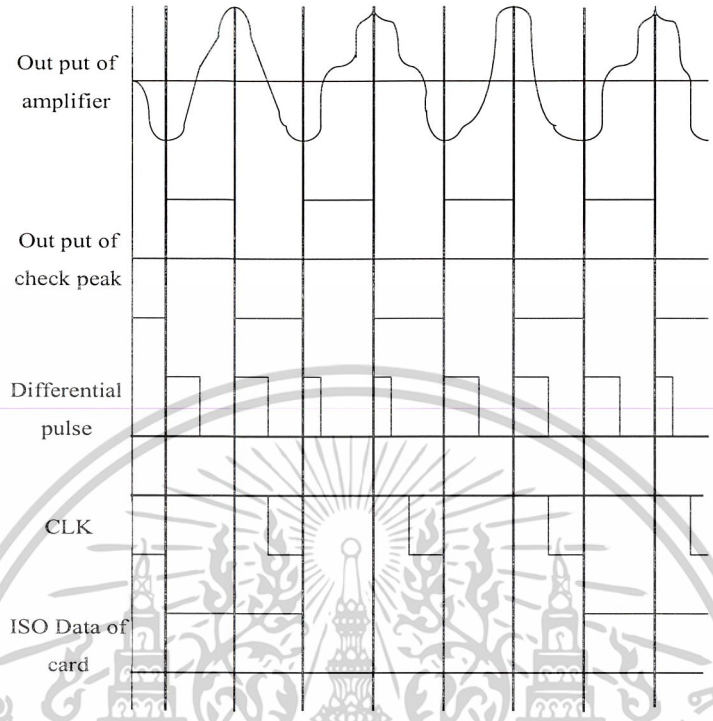
จากสัญลักษณ์ดังรูปที่ 2.11 มีความหมายดังนี้

- SYN (Synchronization character) เป็นการแสดงการเริ่มต้นและปิดท้ายของข้อมูล ใช้รหัส 00H
- Start (Start bit) เป็นการแสดงการเริ่มต้นในการอ่านหรือบันทึกข้อมูล ใช้รหัส 0BH
- DATA เป็นส่วนของข้อมูลที่เป็นตัวเลข 0 - 9
- SEP (Separate) เป็นตัวเชื่อมหรือกั้นระหว่างข้อมูล ใช้รหัส 0DH
- STOP (Stop bit) เป็นการแสดงการสิ้นสุดของข้อมูล ใช้รหัส 0FH
- LRC (Longitude Redundancy Check) เป็นการตรวจสอบชุดข้อมูลในแนวนอน

2.7.3 การอ่านรหัสข้อมูลบัตรแถบแม่เหล็ก

การอ่านข้อมูลจากบัตรแถบแม่เหล็ก ทำได้โดยให้หัวอ่านสัมผัสกับแถบแม่เหล็กซึ่งเคลื่อนที่ด้วยความเร็วคงที่ดังรูปที่ 2.12 ฟลักซ์ที่เกิดจากแม่เหล็กถาวรขนาดเล็กบนแถบแม่เหล็กจะผ่านจากแถบของหัวอ่านไปยังแกนการเปลี่ยนแปลงของฟลักซ์ตามข้อมูลนั้น

จุดสูงสุดของแรงดันที่อ่านได้นั้นจะตรงกับจุดที่สนามแม่เหล็กบนแถบแม่เหล็กกลับทิศทางกันพอดี ดังนั้นถ้าขยายแรงดันนี้ขึ้นและตรวจหาจุดสูงสุดด้วยวิธีตรวจจับความแตกต่าง แล้วเปลี่ยนเป็นสัญญาณพัลส์ก็จะได้ข้อมูลที่บันทึกอยู่ในบัตรแม่เหล็กดังรูปที่ 2.13



รูปที่ 2.12 สัญญาณการอ่านข้อมูล

หลักการเบื้องต้นส่วนของการอ่านและบันทึกข้อมูลในบัตรแถบแม่เหล็ก

- Amplifier of tape head ทำหน้าที่ขยายสัญญาณของหัวเทปที่อ่านข้อมูลบัตรแม่เหล็ก

- Check peak ทำหน้าที่ในการตรวจจับระดับสัญญาณสูงสุดและต่ำสุดของสัญญาณที่ผ่านการขยายสัญญาณแล้ว เพื่อที่จะได้สัญญาณพัลส์ที่ช่วงเวลาต่างกัน

- Wave shaping ทำหน้าที่ปรับเปลี่ยนรูปสัญญาณให้มีขนานเพียงพอ ที่จะนำไปถอดรหัสข้อมูล

- FM demodulator ทำหน้าที่ถอดรหัสข้อมูลจากสัญญาณพัลส์ที่เป็นแบบความแตกต่าง ซึ่งผ่านการปรับระดับสัญญาณให้เพียงพอที่จะให้ไมโครคอนโทลเลอร์ถอดรหัสได้ โดยส่วนนี้จะได้ส่วนที่เป็น Data และ Clock กลับคืนมา

2.7.4 การบันทึกข้อมูลบัตรแถบแม่เหล็ก

การบันทึกข้อมูลลงบนแถบแม่เหล็กจะใช้วิธีการบันทึกแบบดิจิตอลในลักษณะเช่นเดียวกับการใช้ในแผ่น ฟล็อปปี้ดิสก์ การบันทึกข้อมูลลงบนบัตรแถบแม่เหล็กนั้นจะต้องป้อนกระแสพัลส์ที่มีทั้งด้านบวกและด้านลบพร้อมทั้งมีขนาดเพียงพอป้อนเข้าที่ขดลวดของหัวบันทึกซึ่งกดอยู่บนแถบแม่เหล็กที่เคลื่อนที่ด้วยความเร็วคงที่ แถบแม่เหล็กจะถูกเปลี่ยนให้มีรูปแบบของขั้วแม่เหล็กตาม

ฟลักซ์แม่เหล็กที่รั่วไหลจากแถบของหัวบันทึก แถบแม่เหล็กจะเกิดเป็นแม่เหล็กถาวรขนาดเล็กเรียงตัวกันตามขั้วบวกหรือขั้วลบของพัลส์และความกว้างของพัลส์ สัญญาณที่บันทึกเนื่องจากกระแสพัลส์ที่ใช้ในการบันทึกมีขนาดเพียงพอที่หัวบันทึก จะทำให้แถบแม่เหล็กมีสนามแม่เหล็กอิ่มตัวได้ ดังนั้น เมื่อทำการบันทึกข้อมูลตัวข้อมูลที่มีอยู่เดิมจะถูกเขียนทับและหายไปเหลือเพียงข้อมูลใหม่เท่านั้น รูปแบบการบันทึกข้อมูลลงในบัตรแถบแม่เหล็กส่วนใหญ่จะมีรูปแบบ F2F หมายถึง การเข้ารหัสแบบสองความถี่และการมอดูเลตเชิงความถี่ โดยการบันทึกข้อมูลในลักษณะเช่นนี้จะบันทึกข้อมูลและสัญญาณนาฬิกาเข้าไว้ในแทร็กเดียวกัน นอกจากนี้ยังสามารถบันทึกข้อมูลแยกแทร็กทีละแทร็ก เช่น แบบ NRZI (Non Return To Zero Invented Recording)

2.8 ฐานข้อมูล

ฐานข้อมูล (Database) คือ วิธีการจัดเก็บข้อมูลที่สัมพันธ์กันอย่างมีระเบียบ ซึ่งจะทำให้ง่ายต่อการใช้งานและค้นหาข้อมูล ซึ่งฐานข้อมูลที่คนส่วนใหญ่คุ้นเคย คือ ฐานข้อมูลเชิงสัมพันธ์ เป็นรูปแบบการจัดการเก็บข้อมูลในฐานข้อมูลที่สัมพันธ์กัน โดยมองข้อมูลในลักษณะของตารางต่างๆ ที่มีความสัมพันธ์กัน

2.8.1 องค์ประกอบของฐานข้อมูล

1) แอพพลิเคชั่นฐานข้อมูล

แอพพลิเคชั่นฐานข้อมูล เป็นแอพพลิเคชั่นที่สร้างไว้ให้ผู้ใช้งานสามารถติดต่อกับฐานข้อมูลได้อย่างสะดวก ซึ่งมีรูปแบบการติดต่อกับฐานข้อมูลแบบเมนูหรือกราฟิก โดยไม่จำเป็นต้องมีความรู้เกี่ยวกับฐานข้อมูลเลยก็สามารถเรียกใช้งานฐานข้อมูลได้

2) ระบบการจัดการฐานข้อมูล

ระบบการจัดการฐานข้อมูล เป็นซอฟต์แวร์ที่ทำหน้าที่จัดการฐานข้อมูลในฐานข้อมูล ทั้งการจัดเก็บ การแสดงผล การค้นหา การสำรองข้อมูล เป็นต้น โดยเป็นเครื่องมือในการทำงานของผู้บริหารฐานข้อมูล และเป็นตัวกลางที่เชื่อมผ่านระหว่างแอพพลิเคชั่นฐานข้อมูลที่สร้างขึ้นกับตัวข้อมูลในฐานข้อมูล ตัวอย่าง DBMS เช่น Microsoft Access, FoxPro, SQLServer, Oracle, Informix, DB2 เป็นต้น

3) ดาต้าเบสเซิร์ฟเวอร์

ดาต้าเบสเซิร์ฟ เป็นคอมพิวเตอร์ที่คอยให้บริการในการจัดการฐานข้อมูล ซึ่งก็คือคอมพิวเตอร์ที่ระบบจัดการฐานข้อมูลทำงานอยู่นั้นเอง เพราะฉะนั้นจึงมักจะเป็นคอมพิวเตอร์ที่มีประสิทธิภาพสูงกว่าคอมพิวเตอร์ที่ใช้งานทั่วไป

4) ข้อมูล

ข้อมูล คือ ตัวเนื้อของข้อมูลที่ใช้งาน ซึ่งจะถูกเก็บในหน่วยความจำของดาต้าเบสเซิร์ฟเวอร์ โดยจะถูกเรียกมาใช้งานจากระบบจัดการฐานข้อมูล

5) ผู้บริหารฐานข้อมูล

ผู้บริหารฐานข้อมูล เป็นคนที่ทำหน้าที่ดูแลข้อมูลในฐานข้อมูลผ่านระบบจัดการฐานข้อมูล ซึ่งจะควบคุมให้การทำงานเป็นไปอย่างราบรื่น นอกจากนี้ยังทำหน้าที่กำหนดผู้ที่จะมีสิทธิ์ใช้งานฐานข้อมูล กำหนดในเรื่องความปลอดภัยของการใช้งาน พร้อมทั้งดูแลดาต้าเบสเซิร์ฟเวอร์ให้ทำงานอย่างเป็นปกติด้วย

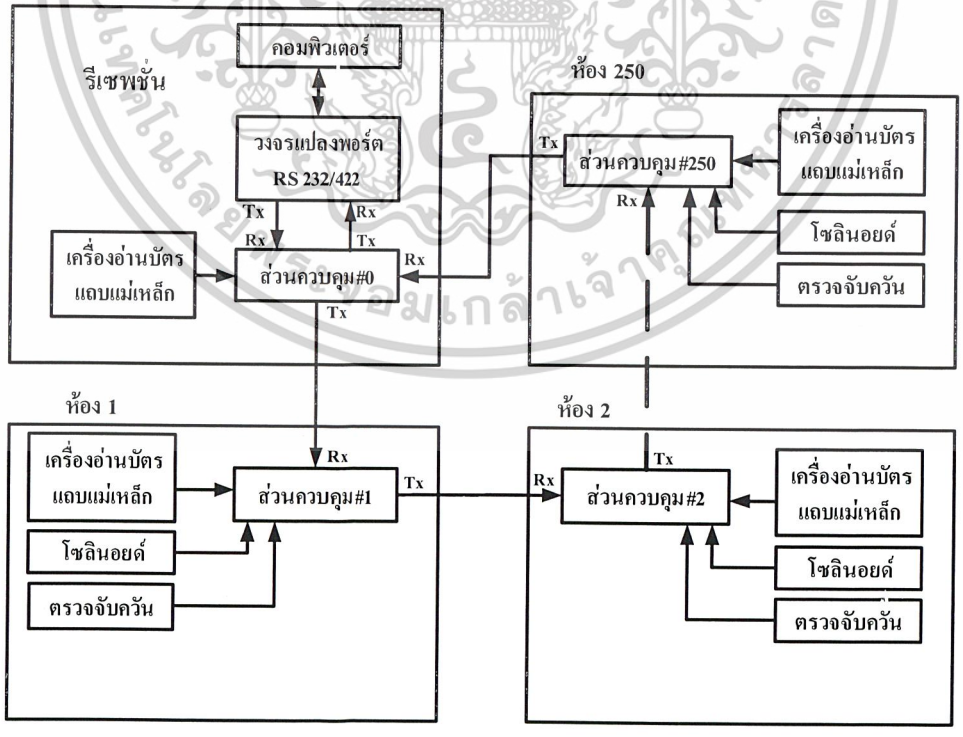
2.8.2 ประโยชน์จากการประมวลผลด้วยฐานข้อมูล

1) ลดความซ้ำซ้อนของข้อมูล
2) สามารถหลีกเลี่ยงความขัดแย้งของข้อมูลได้ในระดับหนึ่ง
3) สามารถใช้ข้อมูลร่วมกันได้
4) สามารถควบคุมความเป็นมาตรฐานได้
5) สามารถจัดหาระบบความปลอดภัยที่รัดกุมได้
6) สามารถควบคุมความคงสภาพของข้อมูลได้
7) สามารถสร้างสมดุลในความขัดแย้งของความต้องการได้
8) เกิดความเป็นอิสระของข้อมูล

บทที่ 3
การออกแบบ การสร้าง และการทำงาน

3.1 กล่าวนำ

ในการออกแบบและการสร้างระบบเช็คอิน - เช็คเอาต์โรงแรมจะประกอบด้วยส่วนที่สำคัญอยู่ 3 ส่วนคือส่วนของรีเซพชั่น (Reception), ส่วนของห้องพักและฐานข้อมูล สำหรับการทำงานจะเริ่มจากการจ่ายไฟเลี้ยงให้กับวงจรและมีตัวแปลงสัญญาณ RS-232/RS-422 เพื่อใช้ในการรับส่งข้อมูลผ่าน RS-485 ไปยังบอร์ดไมโครคอนโทรลเลอร์แต่ละห้องเพื่อใช้ควบคุมการทำงานของอุปกรณ์ในส่วนของห้องพักซึ่งมีเครื่องอ่านบัตรแถบแม่เหล็กที่คอยควบคุมโซลินอยด์ในการเปิดปิดห้องพักและตัวตรวจจับควันจะช่วยในส่วนของด้านความปลอดภัยในกรณีที่เกิดไฟไหม้ นอกจากนี้ยังมีส่วนที่เป็นโปรแกรมเพื่อใช้ในการแสดงผลบนจอคอมพิวเตอร์ที่รีเซ็พชั่น ซึ่งจะแสดงข้อมูลการเข้าพักโรงแรมของห้องพักแต่ละห้องและยังแสดงผลในส่วนของการเกิดไฟไหม้ว่าเป็นห้องไหนจาก รูปที่ 3.1 เป็นโครงสร้างและผังการทำงานของระบบเช็คอิน - เช็คเอาต์ในโรงแรม



รูปที่ 3.1 ผังงานของระบบเช็คอิน - เช็คเอาต์โรงแรม

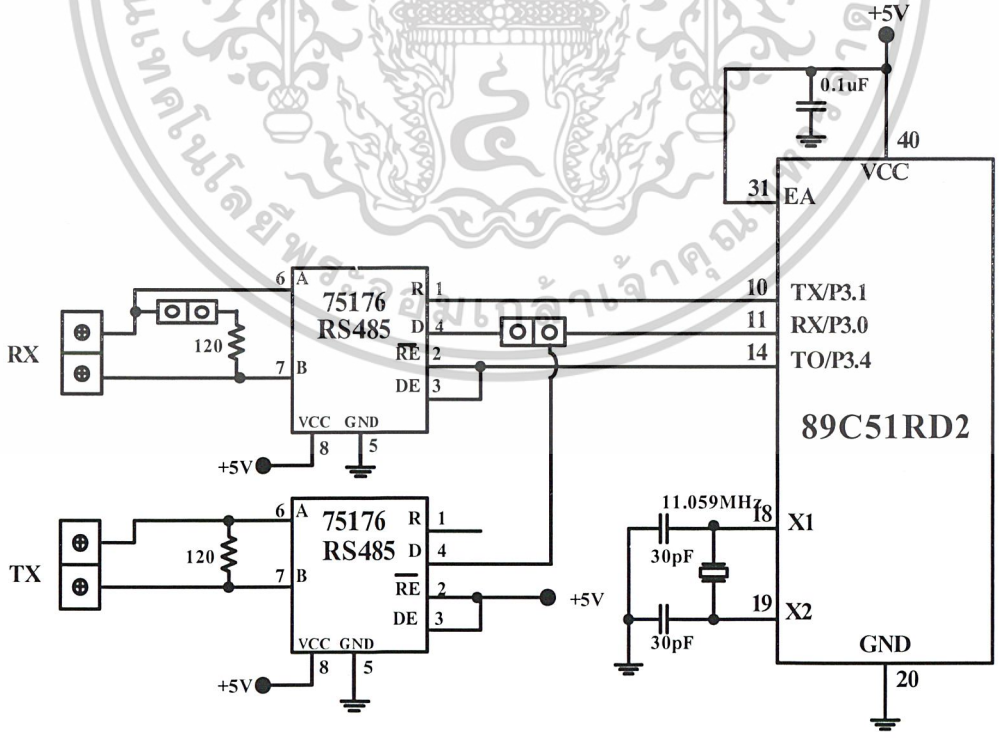
3.2 ส่วนของรีเซพชั่น

ส่วนของรีเซพชั่น (Reception) จะทำหน้าที่รับส่งข้อมูลการเข้าพักของลูกค้าแล้วนำข้อมูลที่ได้จากลูกค้าไปเก็บไว้ในฐานข้อมูลแล้วทำการกำหนดรหัสบัตรแถบแม่เหล็กที่ลูกค้าเข้าพักในแต่ละห้องให้แก่ลูกค้าเพื่อนำบัตรแถบแม่เหล็กไปเปิดห้องพักแทนการเปิดด้วยลูกกุญแจและยังรับข้อมูลในส่วนของห้องพักในส่วนของระบบรักษาความปลอดภัยเช่น มีไฟไหม้ในห้องหรือเปล่า เป็นต้น

3.2.1 การออกแบบวงจรรับและส่งข้อมูล

การออกแบบการรับและส่งข้อมูลจะใช้ไมโครคอนโทรลเลอร์เบอร์ 89C51RD2 ทำหน้าที่รับส่งข้อมูลจากบอร์ดไมโครคอนโทรลเลอร์ตัวส่งไปยังบอร์ดไมโครคอนโทรลเลอร์ตัวรับ โดยขา RX ของไมโครคอนโทรลเลอร์ตัวรับจะต่อกับขา TX ของไมโครคอนโทรลเลอร์ตัวส่ง และขา TX ของไมโครคอนโทรลเลอร์ตัวรับจะต่อกับขา RX ของไมโครคอนโทรลเลอร์ตัวส่งอีกตัวหนึ่ง

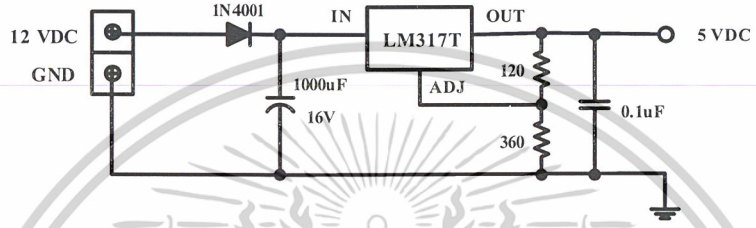
ในรูปที่ 3.2 เป็นออกแบบวงจรการรับส่งข้อมูลของไมโครคอนโทรลเลอร์ จะเป็นลักษณะคล้ายกับการต่อระบบเครือข่าย เพื่อให้การรับส่งข้อมูลมีประสิทธิภาพในการรับส่งข้อมูลได้ระยะที่ไกลมากขึ้นจะใช้ RS-485 (เบอร์ 15176) ช่วยในการรับส่งข้อมูลซึ่งมีอัตราความเร็วในการรับส่งข้อมูลที่สูงโดยตัวหนึ่งจะเป็นตัวรับ (RX) และอีกตัวหนึ่งจะเป็นตัวส่ง (TX)



รูปที่ 3.2 วงจรการรับและส่งข้อมูล

3.2.2 การออกแบบวงจรจ่ายไฟเลี้ยง

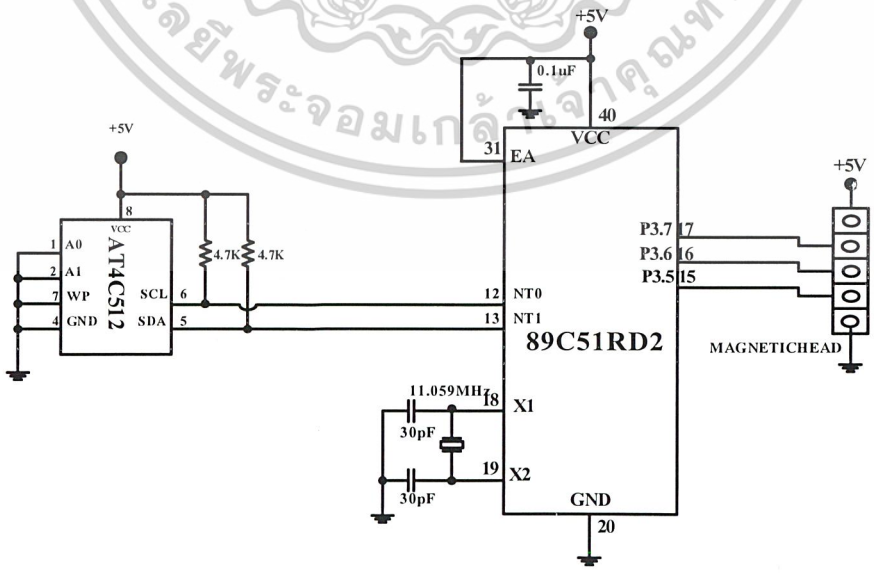
จากรูปที่ 3.3 เป็นการออกแบบการสร้างวงจรไฟฟ้าจ่ายให้กับบอร์ดไมโคลคอนโทรลเลอร์ เป็นไฟฟ้ากระแสตรง 12 โวลต์ และ 5 โวลต์ โดยเริ่มป้อนแรงดันไฟฟ้าให้วงจร 12 โวลต์ และจะลดระดับแรงดันไฟที่จ่ายให้วงจรเหลือ 5 โวลต์ โดยใช้ LM317 ช่วยในการลดระดับแรงดันซึ่งมีไดโอด 1N4001 และคาปาซิเตอร์ 1000 uF ทำหน้าที่ในการเรียงกระแสก่อนที่จะเข้า LM317



รูปที่ 3.3 วงจรจ่ายไฟเลี้ยง

3.2.3 การออกแบบวงจรหน่วยความจำ

จากรูปที่ 3.4 เป็นการออกแบบในส่วนของหน่วยความจำ (MEMORY) โดยให้ AT24C512 เป็นเสมือนหน่วยความจำ ทำหน้าที่เก็บสถานะการทำงานครั้งล่าสุดของบอร์ดไมโครคอนโทลเลอร์ เมื่อไม่มีไฟฟ้ามาจ่ายให้วงจร แต่ถ้ามีการจ่ายไฟให้แก่วงจรอีกครั้งสถานะการทำงานก็จะยังคงเดิม

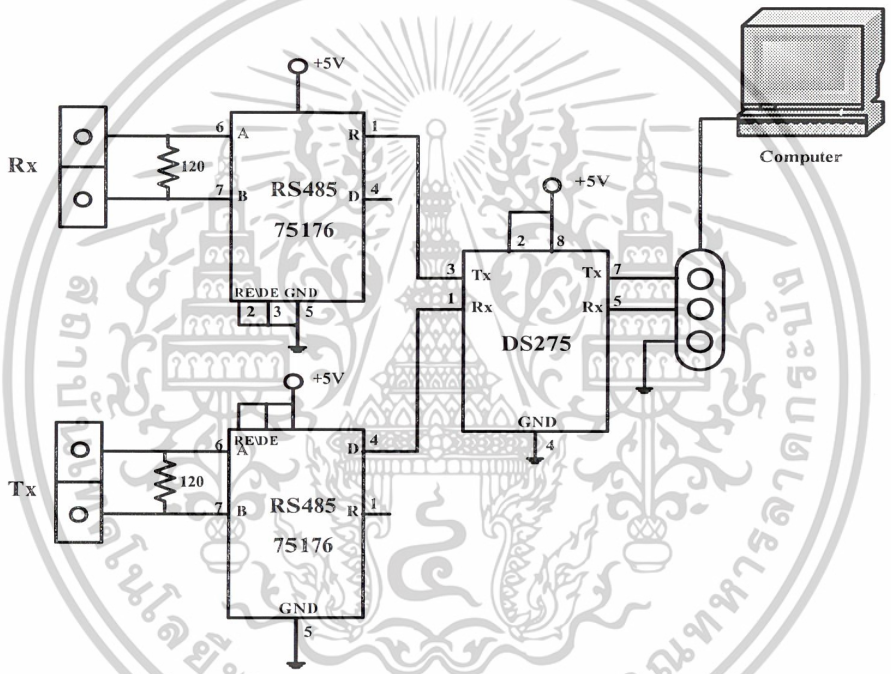


รูปที่ 3.4 วงจรหน่วยความจำ

สำหรับพอร์ตการทำงานP3.7, P3.6และ P3.5 เป็นการต่อเทอร์มินอร์เพื่อรอการต่อใช้งานในส่วนของเครื่องอ่านบัตรแถบแม่เหล็ก (MEGNATIC GARD) เพื่อจะนำไปเก็บเป็นข้อมูลต่อไป

3.2.4 การออกแบบวงจรการรับส่งข้อมูลในส่วนของการแสดงผล

จากรูปที่ 3.5 เป็นวงจรรับส่งข้อมูลในส่วนของการแสดงผล ซึ่งจะใช้คอมพิวเตอร์เป็นตัวแสดงผลโดยจะต่อใช้งานทางด้านพอร์ตอนุกรมจะทำการรับส่งข้อมูลจากไมโครคอนโทรลเลอร์เข้ามาเก็บไว้ในฐานข้อมูลโดยจะทำการต่อ RS-485 (เบอร์ 75176) เพื่อการรับส่งข้อมูลทาง RX และTX ก่อนที่จะต่อกับ RS-232 (เบอร์ DS275) เพื่อต่อเข้ากับพอร์ตอนุกรมของส่วนการแสดงผล



รูปที่ 3.5 วงจรรับส่งข้อมูลในส่วนของการแสดงผล

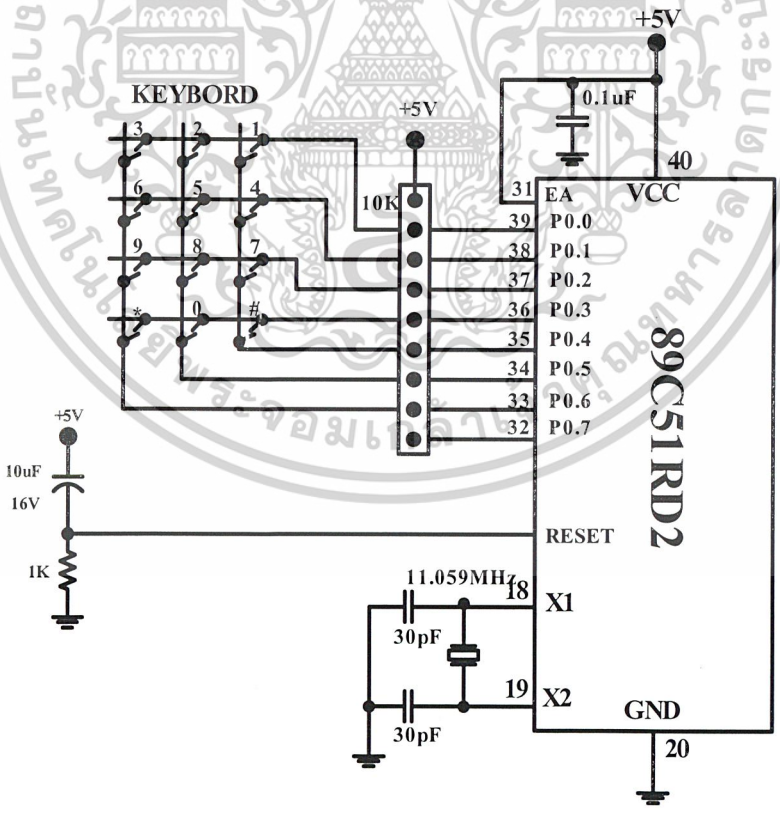
3.3 ส่วนของห้องพัก

ในส่วนของห้องพักการเข้าห้องพักจะต้องใช้บัตรแถบแม่เหล็กในการเปิดห้องพักโดยจะใช้บัตรแถบแม่เหล็กเป็นเสมือนรหัสผ่านเพื่อเปิดห้องพัก โดยในส่วนของห้องพักจะมีการรับส่งข้อมูลถึงกันตลอดเวลากับส่วนของรีเซพชั่นเพื่อจะเป็นข้อมูลในการเก็บรหัสผ่าน โดยทางด้านรีเซพชั่นจะเป็นตัวกำหนดรหัสของบัตรแถบแม่เหล็ก แล้วนำมาเก็บไว้ในส่วนของฐานข้อมูลและคอยส่งข้อมูลไปยังห้องพักเพื่อรอคอยการตอบรับจากทางด้านการเข้าห้องพักคือรหัสที่บัตรแถบแม่เหล็กตรงกับในส่วนของข้อมูลที่เก็บไว้ก็จะสามารถเปิดห้องพักได้

ส่วนของห้องพักไม่แต่จะควบคุมการเปิดปิดโซลินอยด์เข้าห้องพักเพียงอย่างเดียว ยังมีหน้าที่ควบคุมความปลอดภัยจากการเกิดอัคคีภัยภายในห้องด้วย โดยที่คอยควบคุมการทำงานของอุปกรณ์ตรวจจับควันจากการเกิดอัคคีภัยภายในห้องพัก ซึ่งจะเป็นหน้าที่ของไมโครคอนโทลเลอร์ที่ควบคุมการทำงานของอุปกรณ์ตรวจจับควัน โดยไมโครคอนโทลเลอร์จะเป็นตัวรับสัญญาณจากตัวอุปกรณ์ตัวตรวจจับควัน แล้วจะส่งสัญญาณแจ้งไปยังส่วนของรีเซพชั่น เพื่อเป็นการเตือนความปลอดภัยของผู้ที่เข้าพัก และยังเป็นส่วนคอยควบคุมการจ่ายไฟภายในห้องโดยอัตโนมัติเมื่อมีผู้เข้ามาภายในห้อง

3.3.1 การออกแบบวงจรคีย์บอร์ด

จากรูปที่ 3.6 เป็นการออกแบบวงจรคีย์บอร์ด ซึ่งทำหน้าที่เป็นตัวป้อนรหัสผ่านในการเปิดเข้าห้องพักจากกรณีที่ไม่ต้องการใช้บัตรแถบแม่เหล็กในการเปิดห้องพัก โดยจะออกแบบคีย์บอร์ดในลักษณะของ 3x4 แถวมี 12 ปุ่มมีเลข 0-9, # และ * การต่อใช้งานจะต่อเข้ากับพอร์ต P1.0 - P1.7 โดยมีการต่อค่าความต้านทานแบบแถวไว้ 10 กิโลโอห์ม

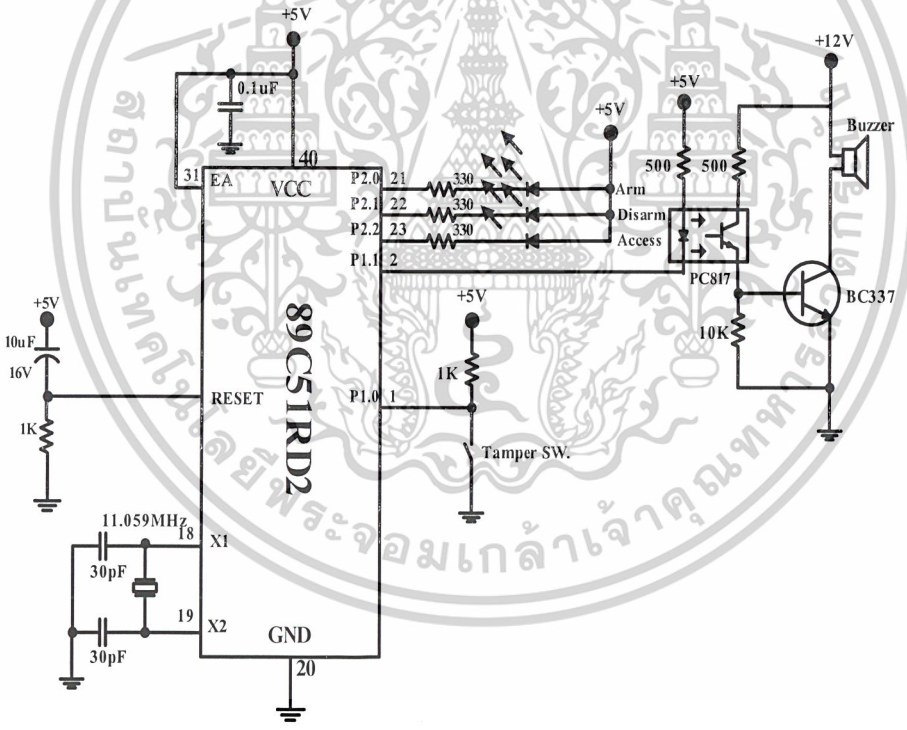


รูปที่ 3.6 วงจรคีย์บอร์ด

3.3.2 การออกแบบวงจรบัสเซอร์

จากรูปที่ 3.7 เป็นการออกแบบวงจรขับบัสเซอร์ (Buzzer) ซึ่งจะเป็นส่วนของสัญญาณเสียง วงจรบัสเซอร์จะทำหน้าที่ในการส่งสัญญาณเสียงออกมาเพื่อเป็นการบอกสถานะการทำงาน เมื่อมีการกดคีย์บอร์ดและเมื่อมีการรับและส่งข้อมูล การใช้งานในส่วนของวงจรบัสเซอร์จะต่อใช้งานกับพอร์ต P1.2 ของไมโครคอนโทลเลอร์ โดยสัญญาณเตือนนี้จะมีเสียงลักษณะอย่างไรนั้นจะขึ้นอยู่กับการเขียนโปรแกรมควบคุม

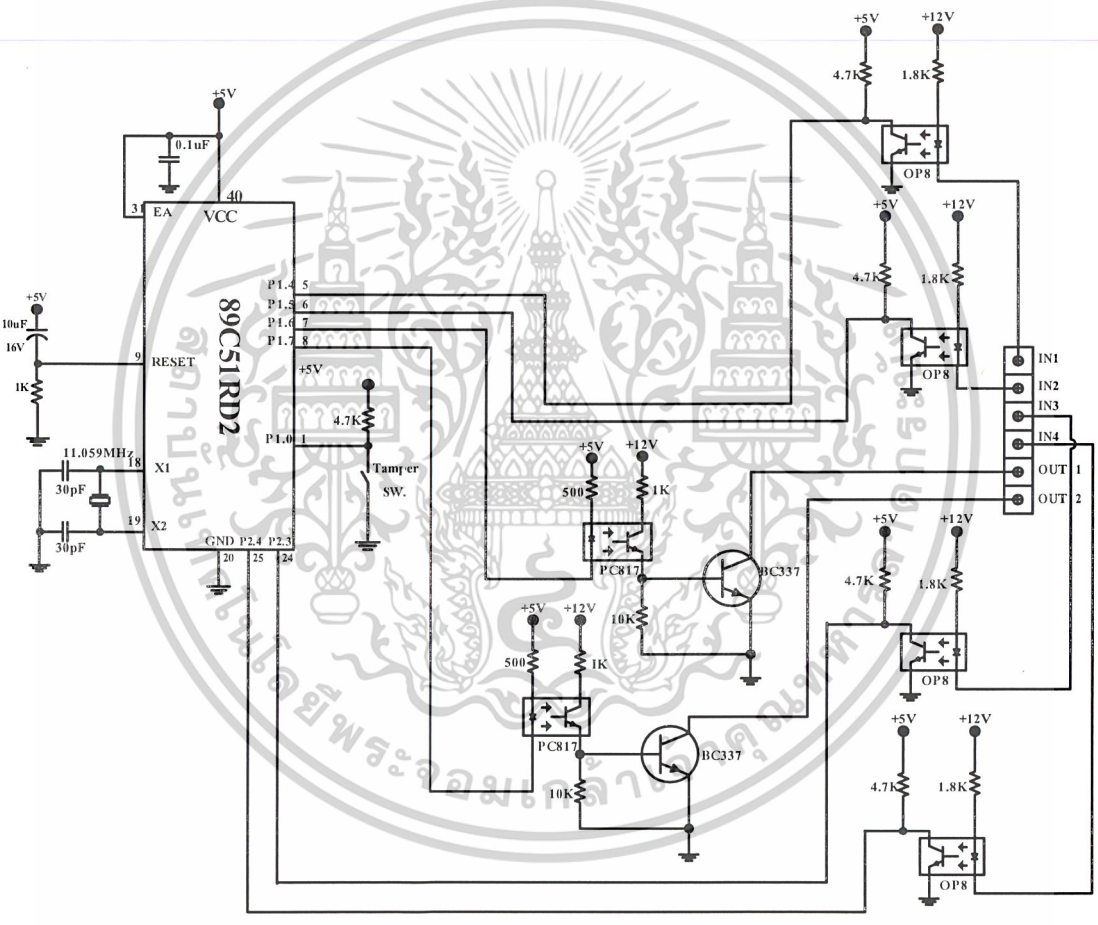
สำหรับพอร์ต P2.0, P2.1 และP2.3 ของไมโครคอนโทลเลอร์ จะเป็นการต่อใช้งานในส่วนของการแสดงผลเพื่อเป็นการบอกสถานะการทำงานออกมาในรูปของหลอด LED แสดงในส่วนของสัญญาณ (Arm), ไม่มีสัญญาณ (Disarm) และการรับส่งข้อมูล (Access) โดยทั้งสามส่วนทำการจะแสดงออกมาพร้อมกับสัญญาณเสียงบัสเซอร์เพื่อแจ้งสถานะการทำงานของระบบ



รูปที่ 3.7 วงจรบัสเซอร์

3.3.3 การออกแบบวงจรควบคุมอุปกรณ์

จากรูปที่ 3.8 เป็นการออกแบบในส่วนของวงจรควบคุมอุปกรณ์ภายในห้องพักโดยจะใช้พอร์ต P1.4, P1.5, P2.3 และ P2.4 ทำหน้าที่ในการรับสัญญาณอินพุตที่ได้จากการส่งข้อมูลมาจากอุปกรณ์ภายนอก เช่น ตัวตรวจจับควัน จะทำหน้าที่ส่งสัญญาณมายังบอร์ดไมโครคอนโทลเลอร์เพื่อทำการประมวลผลส่งต่อไปยังส่วนของรีเซพชั่น และในส่วนของพอร์ต OUT1, OUT2 จะทำหน้าที่ในการส่งสัญญาณควบคุมโซลินอยด์เพื่อสั่งให้โซลินอยด์เปิดประตูห้องพัก



รูปที่ 3.8 วงจรควบคุมอุปกรณ์

3.4 ฐานข้อมูล

ส่วนในฐานข้อมูลจะทำหน้าที่เก็บข้อมูลของผู้ที่เข้าพักในโรงแรมและรหัสบัตรแถบแม่เหล็กเพื่อใช้ในการเปิด-ปิดประตูอัตโนมัติ ส่วนของข้อมูลของผู้ที่เข้าพักนั้น เช่น ชื่อ (Name), ที่อยู่ (Address), เบอร์โทรศัพท์ (Telephone), อายุ (Age), สัญชาติ (Nationality) และหมายเลขบัตรเครดิต (Credit card) เป็นต้น นอกจากนี้ระบบฐานข้อมูลยังเก็บข้อมูลในส่วนของห้องพักและจะแสดงสถานะในการเข้าพัก เช่น สถานะของห้องพัก (Room status) จะแสดงในส่วนของการเข้าพักว่าห้องไหนมีการเข้าพักแล้วหรือไม่ และในส่วนของฐานข้อมูลจะทำการติดต่อโดยใช้โปรแกรม Visual C++

บทที่ 4

การทดลองและผลการทดลอง

4.1 บทนำ

เพื่อให้การทดลองและการตรวจสอบการทำงานของระบบสะดวกขึ้น จึงแบ่งภาคการทำงานของระบบออกเป็นส่วนต่างๆ คือ การทำงานของชุดเปิด - ปิดประตูอัตโนมัติ, การทำงานของชุดตรวจสอบไฟไหม้, ส่วนของเครื่องอ่านบัตรแถบแม่เหล็ก, การรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุม, ส่วนของการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คอิน, ส่วนของการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คเอาต์, ระบบโดยรวมของระบบเช็คอิน - เช็คเอาต์โรงแรมทำการทดลองได้ผลการทดลองดังนี้

4.2 การทดลองการทำงานของชุดเปิด - ปิดประตูอัตโนมัติ

การทดลองการทำงานของชุดเปิด - ปิดระตูอัตโนมัติกล่าวคือ การใช้ไฟฟ้าป้อนให้กับตัวอุปกรณ์เพื่อศึกษาหาจุดการทำงานของอุปกรณ์เปิด - ปิดประตูอัตโนมัติ

4.2.1 การทดลอง

1) ต่ออุปกรณ์เปิด-ปิดประตูอัตโนมัติเข้ากับแหล่งจ่ายไฟฟ้าที่เพิ่มและลดได้

2) ทำการตั้งค่าแรงดันไฟฟ้าของแหล่งจ่ายให้มีค่าเท่ากับ 5 โวลต์ พร้อมสังเกตการทำงานของอุปกรณ์เปิด - ปิด ประตูอัตโนมัติว่าทำงานหรือไม่ทำงาน แล้วทำการบันทึกผล

3) ทำการเพิ่มขนาดของแรงดันไฟฟ้าขึ้นทีละ 1 โวลต์ จนกว่าจะถึง 12 โวลต์ พร้อมสังเกตการทำงานของอุปกรณ์เปิด - ปิดประตูอัตโนมัติว่าทำงานหรือไม่ทำงาน แล้วทำการบันทึกผล

4.2.2 ผลการทดลอง

หลังจากที่ได้ทำการทดลองโดยการป้อนแรงดันไฟฟ้าเข้าไปที่ตัวเปิด - ปิดประตูอัตโนมัติแล้วได้ผลการทดลองดัง ตารางที่ 4.1

ตารางที่ 4.1 การทดลองโดยการป้อนแรงดันไฟฟ้าเข้าไปที่ตัวเปิด - ปิดประตูอัตโนมัติ

ครั้งที่	ขนาดแรงดันไฟฟ้า	ผลที่ได้จากตัวเปิด-ปิดประตู
1	5 โวลต์	ไม่ทำงาน
2	6 โวลต์	ไม่ทำงาน
3	7 โวลต์	ไม่ทำงาน
4	8 โวลต์	ไม่ทำงาน
5	9 โวลต์	ทำงาน
6	10 โวลต์	ทำงาน
7	11 โวลต์	ทำงาน
8	12 โวลต์	ทำงาน

จากผลการทดลองสังเกตได้ว่าตัวเปิด - ปิดประตูอัตโนมัติจะทำงานอยู่ในช่วงของขนาดแรงดันไฟฟ้าที่ระดับ 8 โวลต์ขึ้นไป แต่ถ้าหากแรงดันไฟฟ้าต่ำกว่าระดับ 8 โวลต์ ตัวเปิด - ปิดประตูอัตโนมัติจะไม่ทำงาน

4.3 การทำงานของชุดตรวจสอบไฟไหม้

การทดลองการทำงานของชุดตรวจสอบไฟไหม้ คือ การป้อนควันเข้าไปที่ตัวอุปกรณ์ตรวจสอบไฟไหม้ เพื่อที่จะหาระยะเวลาในการทำงานของตัวอุปกรณ์

4.3.1 การทดลอง

1) ต่ออุปกรณ์ตรวจสอบไฟไหม้เข้ากับแหล่งจ่ายไฟฟ้าขนาด 12 โวลต์

2) ทำการป้อนควันอุปกรณ์ตรวจจับไฟไหม้ โดยใช้ควันจากธูปในการทดลอง เพื่อจับเวลาในการทำงานของอุปกรณ์ตรวจจับไฟไหม้

3) ปริมาณของควันไฟสามารถเพิ่มได้โดยเพิ่มจำนวนธูปขึ้นทีละ 1 ดอก เริ่มตั้งแต่ 1 ดอก และสิ้นสุดที่ 5 ดอก

4.3.2 ผลการทดลอง

ตารางที่ 4.2 การทดลองโดยการป้อนควันเข้าไปที่ตัวตรวจสอบไฟไหม้

ครั้งที่	จำนวนธูป	เวลาที่ใช้	ผลที่ได้ตัวตรวจสอบควัน
1	1 ดอก	10.87 วินาที	ทำงาน
2	2 ดอก	9.80 วินาที	ทำงาน
3	3 ดอก	6.60 วินาที	ทำงาน
4	4 ดอก	5.50 วินาที	ทำงาน
5	5 ดอก	5.08 วินาที	ทำงาน

ผลการทดลองสังเกตได้ว่าตัวตรวจสอบไฟไหม้จะเริ่มทำงานก็ต่อเมื่อได้รับปริมาณควันที่เหมาะสม ซึ่งถ้าตัวตรวจสอบไฟไหม้ได้รับปริมาณควันมากจะทำงานได้เร็ว

4.4 การทดลองเครื่องอ่านบัตรแถบแม่เหล็ก

การทดลองในส่วนของเครื่องอ่านบัตรแถบแม่เหล็ก คือ การนำบัตรแถบแม่เหล็กมาทำการรูดผ่านเครื่องอ่านบัตรแถบแม่เหล็ก เพื่อที่จะดูว่าเครื่องอ่านบัตรแถบแม่เหล็กที่สร้างขึ้นนั้นใช้งานได้จริงหรือไม่

4.4.1 การทดลอง

1) เขียนโปรแกรมการรับข้อมูลจากเครื่องอ่านบัตรแถบแม่เหล็ก

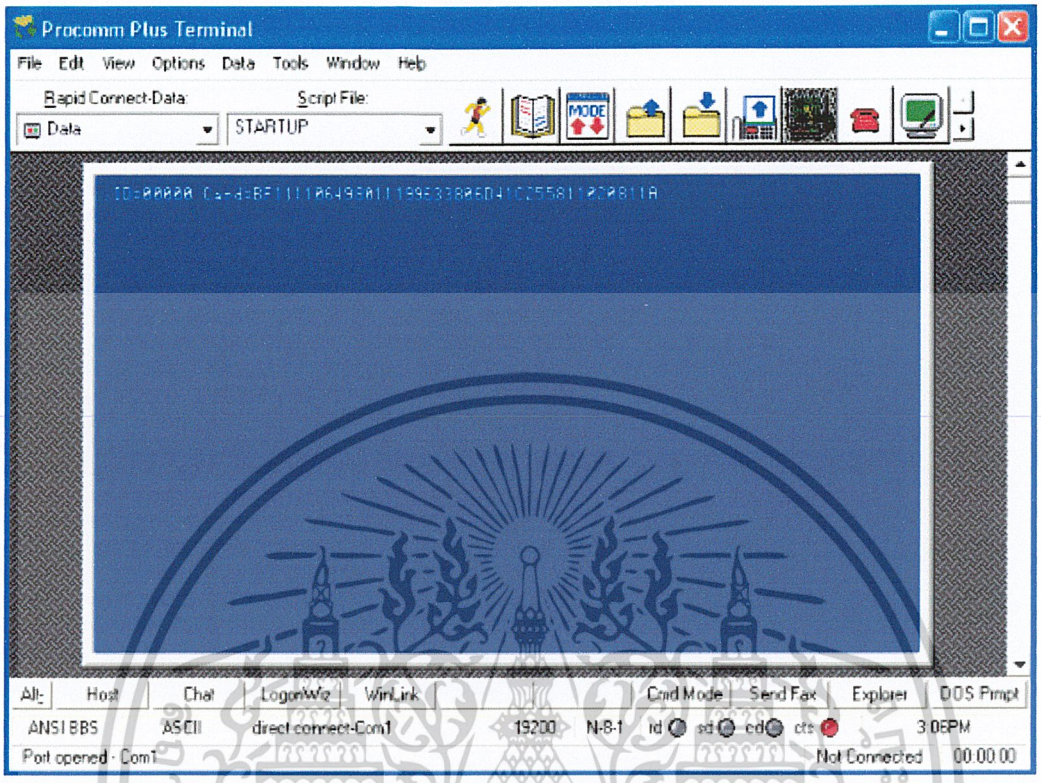
2) ทำการเปิดโปรแกรม Procomm เพื่อที่จะใช้แสดงผลการอ่านบัตรแถบแม่เหล็ก

3) นำบัตรแถบแม่เหล็กมาทำการรูดที่เครื่องอ่านบัตรแถบแม่เหล็ก

4) สังเกตการแสดงผลที่โปรแกรม Procomm และทำการบันทึก

4.4.2 ผลการทดลอง

จากการทดลองในส่วนของเครื่องอ่านบัตรแถบแม่เหล็กจะได้ผลการทดลอง คือ เครื่องอ่านบัตรแถบแม่เหล็กสามารถอ่านบัตรแถบแม่เหล็ก และนำรหัสจากบัตรแถบแม่เหล็กไปแสดงผลที่โปรแกรม Procomm ได้ ดังรูปที่ 4.8



รูปที่ 4.1 รหัสของบัตรแถบแม่เหล็กที่อ่านเครื่องอ่านบัตรแถบแม่เหล็ก

4.5 การรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุม

การทดลองการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุมของห้องพัก ทำการทดลองเพื่อจะตรวจสอบว่าสามารถทำการส่งข้อมูลกันได้จริงหรือไม่

4.5.1 การทดลอง

1) ทำการต่อพอร์ตคอม 1 ของคอมพิวเตอร์เข้ากับชุดแปลง RS 232/RS 422 และบอร์ดควบคุมห้องพัก

2) ทำการเรียกโปรแกรม Procomm ขึ้นมาเพื่อใช้แสดงผลในการทดลอง

3) ตั้งอัตราบอร์ดเรทไว้ที่ 19200 และเลือกพอร์ตคอม 1 ที่ตัวโปรแกรม

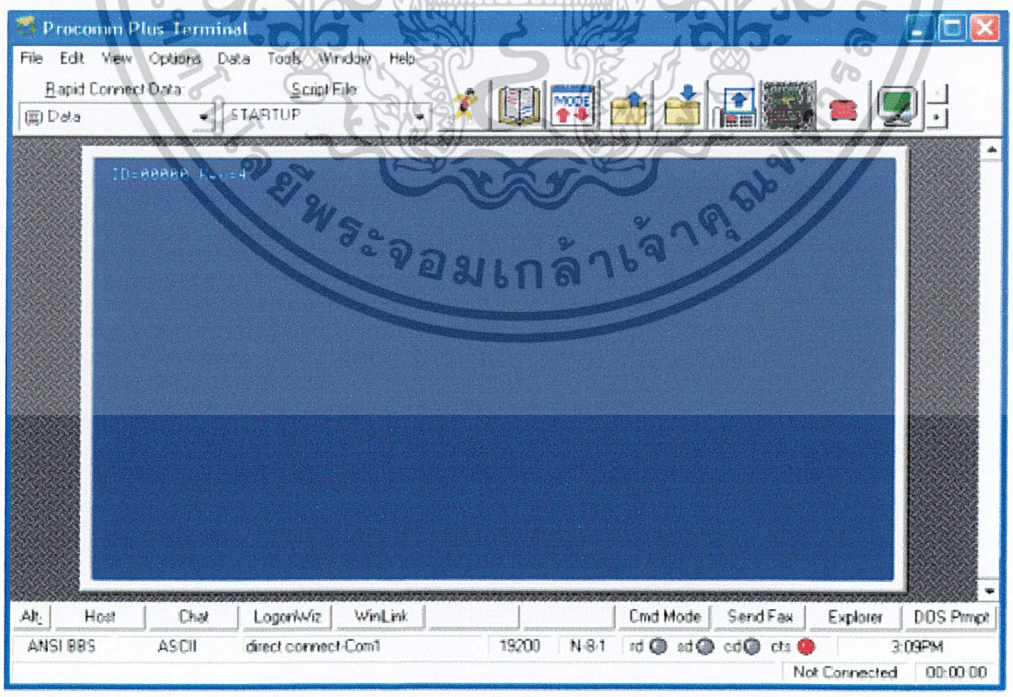
4) ทำการกดคีย์ต่างๆ จากบอร์ดควบคุมและบันทึกผลการทดลอง

4.5.2 ผลการทดลอง

หลังจากที่ได้ทำการทดลองส่วนของการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุม โดยการกดคีย์ที่บอร์ดควบคุมส่งไปยังคอมพิวเตอร์ แล้วได้ผลการทดลองดังตารางที่ 4.3

ตารางที่ 4.3 ผลการทดลองการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุม

ครั้งที่	การกดคีย์จากบอร์ดควบคุม	การแสดงผลบนคอมพิวเตอร์
1	แถวที่ 1 ตัวที่ 1	ID 0000 Key = 7
2	แถวที่ 1 ตัวที่ 2	ID 0000 Key = 8
3	แถวที่ 1 ตัวที่ 3	ID 0000 Key = 9
4	แถวที่ 2 ตัวที่ 1	ID 0000 Key = 4
5	แถวที่ 2 ตัวที่ 2	ID 0000 Key = 5
6	แถวที่ 2 ตัวที่ 3	ID 0000 Key = 6
7	แถวที่ 3 ตัวที่ 1	ID 0000 Key = 1
8	แถวที่ 3 ตัวที่ 2	ID 0000 Key = 2
9	แถวที่ 3 ตัวที่ 3	ID 0000 Key = 3
10	แถวที่ 4 ตัวที่ 1	ID 0000 Key = 10
11	แถวที่ 4 ตัวที่ 2	ID 0000 Key = 0
12	แถวที่ 4 ตัวที่ 3	ID 0000 Key = 12



รูปที่ 4.2 การทดลองส่วนของการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุม

การทดลองที่ได้พบว่าในส่วนของการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุมห้องพักมีการรับส่งข้อมูลถึงกันได้

4.6 การทดลองการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คอิน

การทดลองในส่วนของการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คอิน คือ การทดลองดูว่าโปรแกรมการเช็คอิน - เช็คเอาต์ ที่สร้างขึ้นสามารถทำงานได้อย่างถูกต้องหรือไม่

4.6.1 การทดลอง

1) ทำการเปิดโปรแกรมการเช็คอิน - เช็คเอาต์
2) ทำการเลือกห้องพักที่ว่าง
3) ทำการกรอกข้อมูลต่างๆ ของลูกค้า เมื่อเสร็จแล้วกดปุ่มเช็คอิน
4) ดูผลการเปลี่ยนแปลงที่โปรแกรมการเช็คอิน - เช็คเอาต์

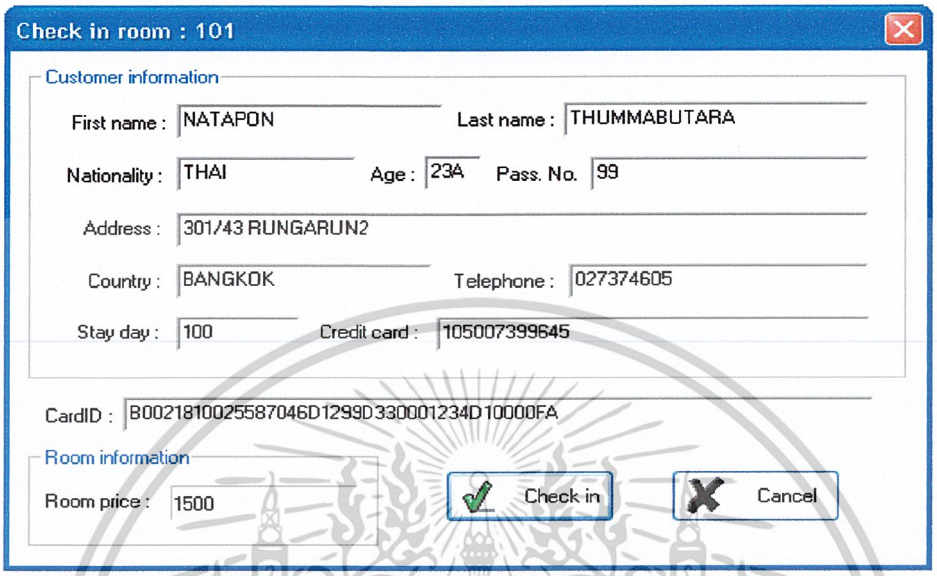
4.6.2 ผลการทดลอง

จากการการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คอิน จะได้ผลการทดลองดังนี้

1) เมื่อทำการเปิดโปรแกรมการเช็คอิน - เช็คเอาต์ ดังรูปที่ 4.2

รูปที่ 4.3 หน้าต่างโปรแกรมระบบเช็คอิน – เช็คเอาต์โรงแรม

2) เมื่อทำการเลือกห้องพักที่ว่างจะหน้าต่างให้กรอกข้อมูลต่างๆ ของลูกค้า ดังรูปที่ 4.3



รูปที่ 4.4 การกรอกข้อมูลของลูกค้าเพื่อทำการเช็คอิน

3) เมื่อมีการกรอกข้อมูลของผู้ที่ทำการเช็คอินเพื่อใช้บริการเสร็จ จะเกิดการเปลี่ยนแปลงที่หน้าต่างของระบบโปรแกรมการเช็คอิน - เช็คเอาต์โรงแรม เพื่อแสดงว่ามีผู้พักแล้ว ดังรูปที่ 4.5

รูปที่ 4.5 การเปลี่ยนแปลงหน้าต่างโปรแกรมระบบการเช็คอิน - เช็คเอาต์โรงแรม

4.7 การทดลองในส่วนของการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คเอาต์

การทดลองในส่วนของการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คเอาต์ คือ การทดลองดูว่าโปรแกรมการเช็คอิน - เช็คเอาต์ ที่สร้างขึ้นสามารถทำงานได้อย่างถูกต้องหรือไม่

4.7.1 การทดลอง

1) ทำการเปิดโปรแกรมการเช็คอิน – เช็คเอาต์

2) ทำการเลือกห้องพักที่มีคนพักอาศัยอยู่

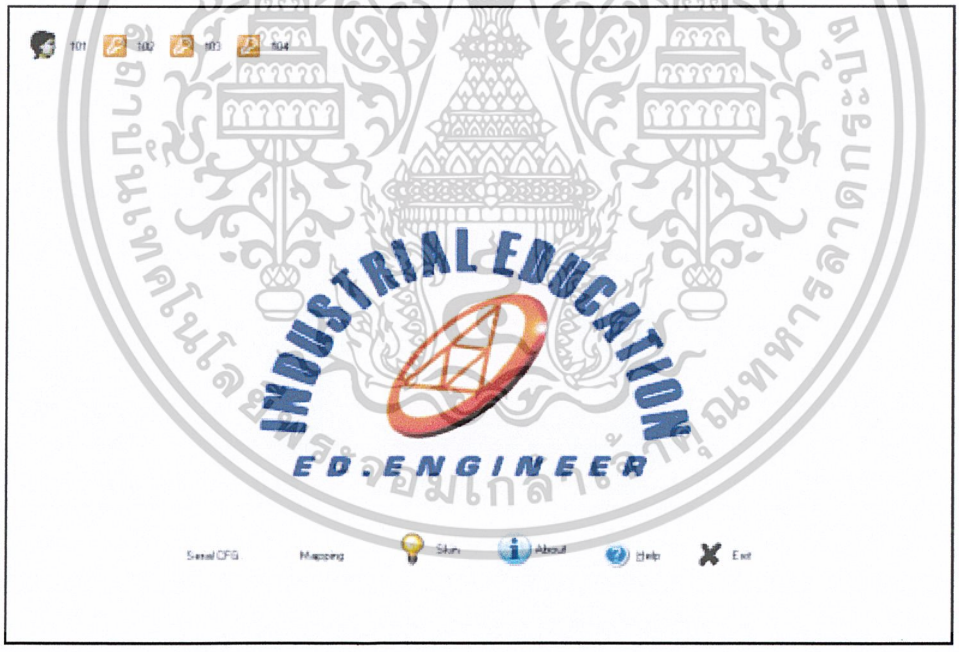
3) เมื่อเสร็จแล้วกดปุ่มเช็คเอาต์

4) ดูผลการเปลี่ยนแปลงที่โปรแกรมการเช็คอิน - เช็คเอาต์

4.7.2 ผลการทดลอง

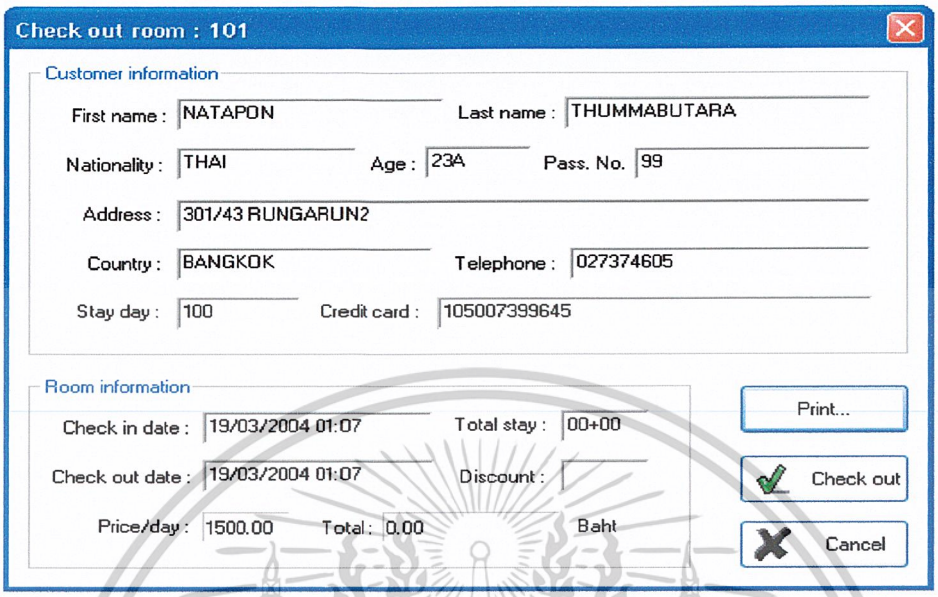
จากการทดลองการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คเอาต์ จะได้ผลการทดลองดังนี้

1) เมื่อทำการเปิดโปรแกรมการเช็คอิน - เช็คเอาต์ดังรูปที่ 4.5



รูปที่ 4.6 หน้าต่างโปรแกรมระบบเช็คอิน - เช็คเอาต์โรงแรมขณะมีผู้เช็คอิน

2) ทำการเลือกห้องพักที่มีคนพักอาศัยอยู่จะปรากฏหน้าต่างแสดงข้อมูลต่างๆ ของลูกค้า ดังรูปที่ 4.6



รูปที่ 4.7 ข้อมูลของลูกค้าก่อนที่จะเช็คเอาต์

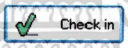
3) เมื่อกดปุ่มเช็คเอาต์ จะทำให้เกิดการเปลี่ยนแปลงที่โปรแกรมการเช็คอิน - เช็คเอาต์ ดังรูปที่ 4.7

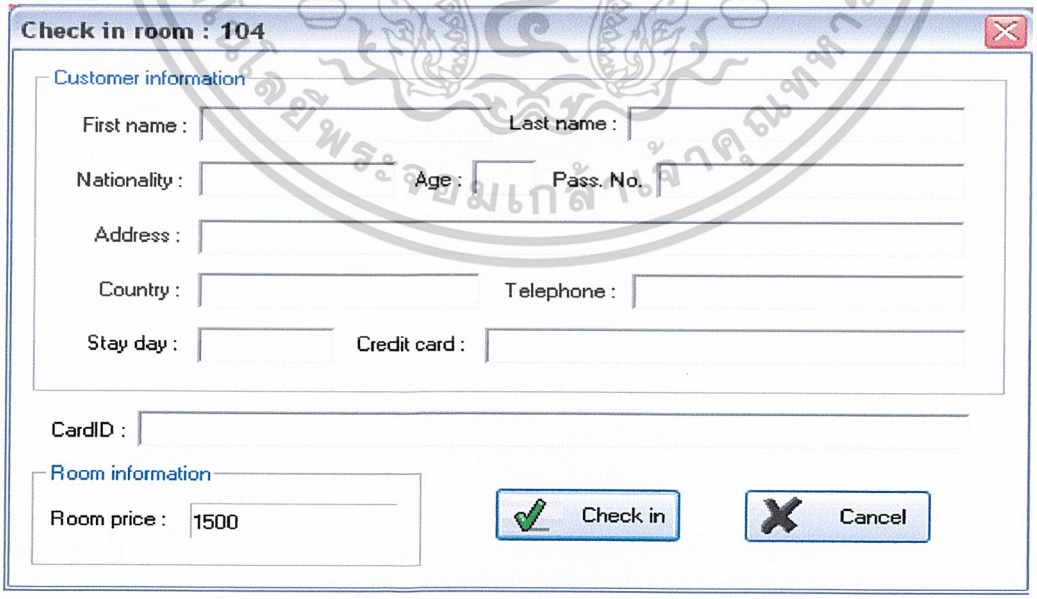
รูปที่ 4.8 การเปลี่ยนแปลงของหน้าต่างโปรแกรมเมื่อมีผู้เช็คเอาต์แล้ว

4.8 ระบบเช็คอิน - เช็คเอาต์โรงแรม

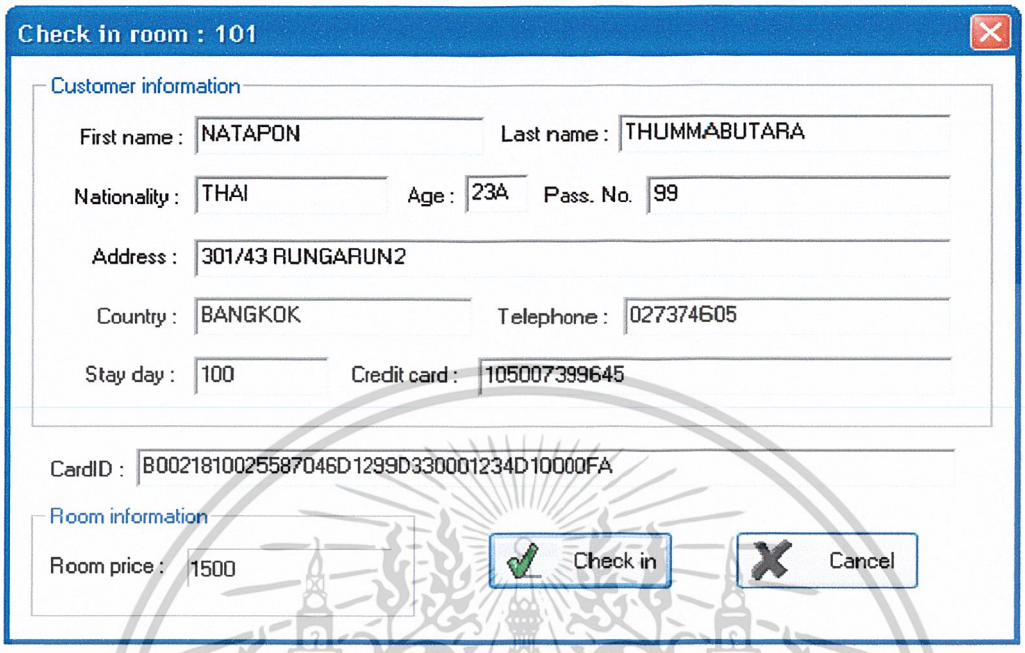
การทดลองระบบเช็คอิน - เช็คเอาต์โรงแรม เป็นการทดลองระบบโดยรวมทั้งหมดว่าทำงานได้หรือไม่

4.8.1 การทดลอง

เมื่อทำการทดลองส่วนต่างๆ เสร็จ จากนั้นทำการเชื่อมต่อส่วนต่างๆ ในระบบเข้าด้วยกันให้เรียบร้อย ระบบโดยรวมคือการต่อเครื่องอ่านบัตรแถบแม่เหล็ก ตัวตรวจสอบไฟไหม้และตัวเปิด-ปิดประตูอัตโนมัติเข้ากับบอร์ดควบคุมหน้าห้องพักแล้วทำการเชื่อมต่อห้องพักทั้งหมดเข้ากับส่วนของการเก็บข้อมูลเข้าด้วยกันจากนั้นทำการป้อนข้อมูลคนที่ต้องการเข้าพักเมื่อทำการกรอกข้อมูลคนที่เข้าพักโดยคลิกที่ไอคอน  จะปรากฏดังรูปที่ 4.9 เริ่มทำการการใส่ข้อมูลที่ First Name คือชื่อจริง, Last Name คือนามสกุล, Nationality คือสัญชาติ, Age คืออายุ, Pass No คือ รหัสบัตรประชาชน, Address คือที่อยู่, Country คือเมืองที่อยู่, Telephone คือเบอร์โทรศัพท์ที่สามารถติดต่อได้, Stay Day คือวันและเวลาที่ต้องการเข้าพัก, Credit card คือรหัสของบัตรแถบแม่เหล็กเรียบร้อยแล้วทำการรูดบัตรแถบแม่เหล็ก ข้อมูลจะถูกบันทึกลงในช่อง Card ID เมื่อทำการใส่ข้อมูลครบแล้วทำการกดไอคอน  ข้อมูลจะถูกบันทึกในฐานข้อมูล การรูดบัตรแถบแม่เหล็กที่เปรียบเสมือนกุญแจเปิด - ปิดประตูอัตโนมัติ สามารถนำบัตรที่ทำการกรอกข้อมูลแล้วไปเปิด-ปิดประตูและตรวจสอบไฟไหม้ด้วยการปล่อยควันไปที่ห้องพัก

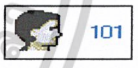


รูปที่4.9 การกรอกข้อมูลเพื่อทำการเช็คอิน

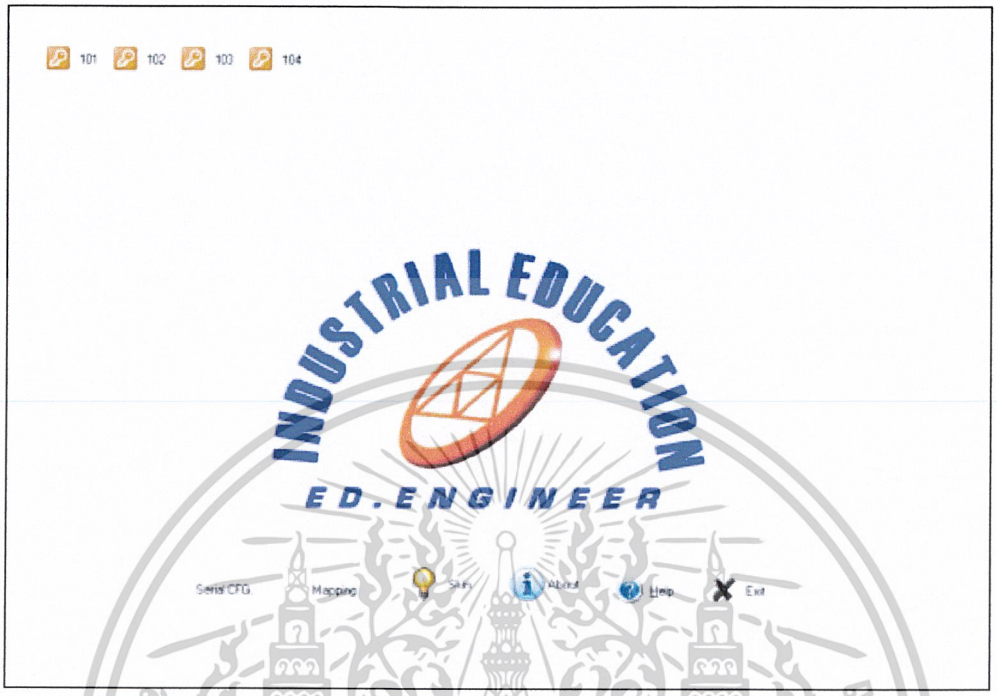


รูปที่ 4.10 การกรอกข้อมูลในการเข้าใช้บริการ

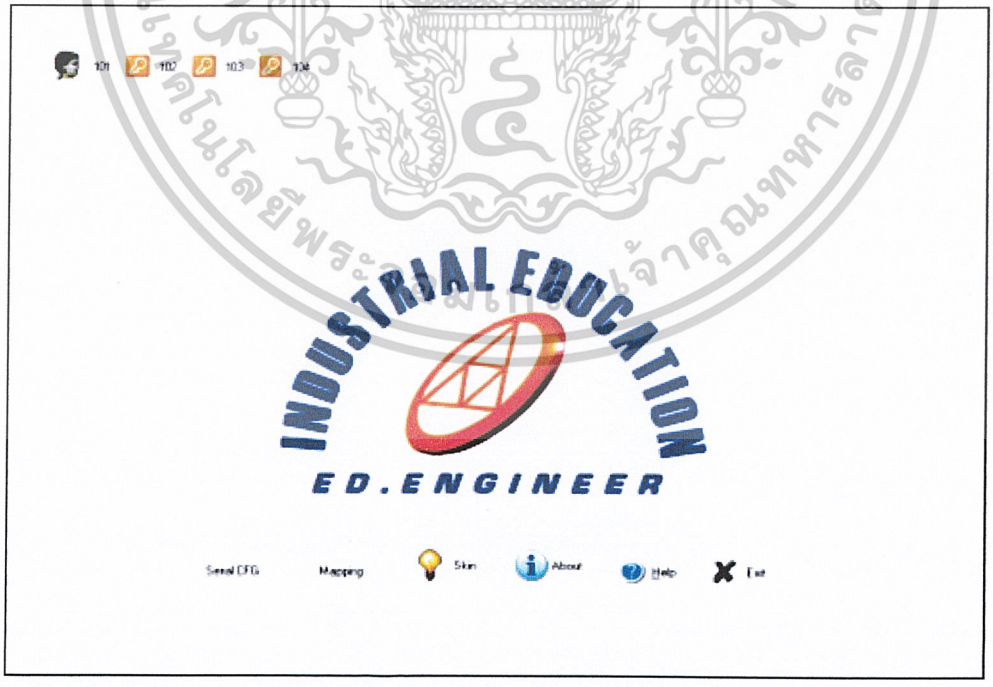
4.8.2 ผลการทดลอง

เมื่อทำการกรอกข้อมูลคนที่เข้าพักเสร็จเรียบร้อยไอคอนจะเปลี่ยนเป็น 

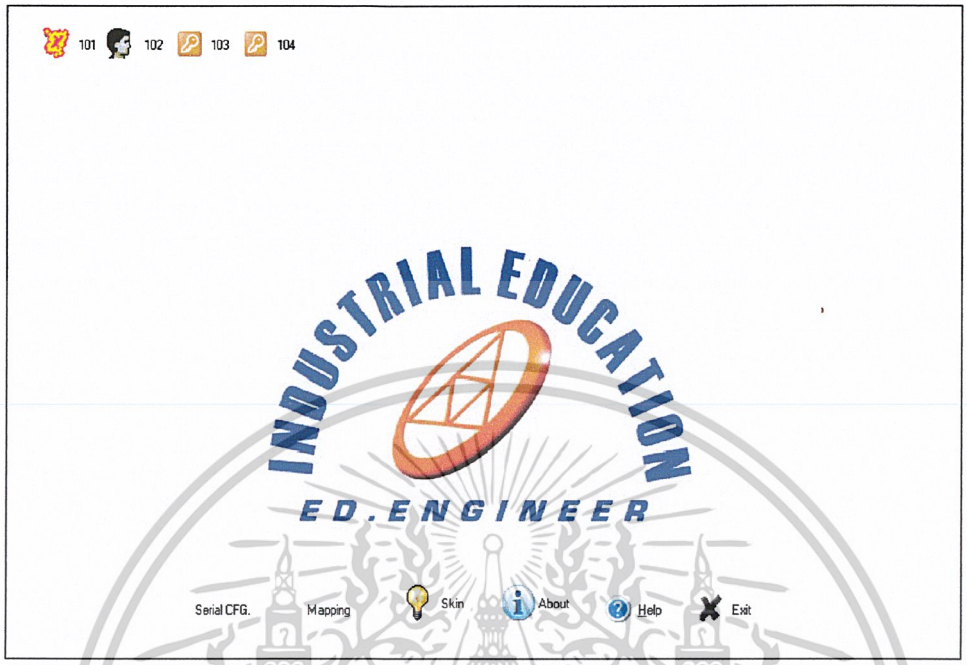
แสดงให้ทราบถึงการมีคนเข้าพักห้องพักนี้แล้ว และสามารถคลิกที่ไอคอนนี้เพื่อดูข้อมูลคนเข้าพักได้ เริ่มนำบัตรแม่เหล็กที่ได้ใส่ใน Card ID แล้วไปรูดเข้าห้องที่ 1 ก็จะสามารถเปิด-ปิดประตูอัตโนมัติได้ และเมื่อป้อนควันเข้าไปก็จะมีไอคอนปรากฏดังรูปที่ 4.12 และยังมีสัญญาณเตือนยาวที่ปล่อยจากบัสเซอร์



รูปที่ 4.11 หน้าต่างโปรแกรมการเช็คอิน - เช็คเอาต์เมื่อเริ่มทำงาน



รูปที่ 4.12 หน้าต่างโปรแกรมการเช็คอิน - เช็คเอาต์เมื่อมีผู้เข้าพัก



รูปที่ 4.13 หน้าต่างโปรแกรมระบบเช็คอิน - เช็คเอาต์เมื่อเกิดไฟไหม้

เมื่อมีการรูดบัตรแถบแม่เหล็กที่บอร์ดไมโครคอนโทลเลอร์และรหัสตรงกับฐานข้อมูลที่ใส่เก็บไว้ ก็จะสามารถทำการเปิด - ปิดประตูอัตโนมัติได้และเมื่อทำการปล่อยควันเข้าไปในปริมาณที่เพียงพอก็จะทำให้ตัวตรวจจับควัน ไฟทำงาน ก็จะเกิดเสียงเตือนภัยขึ้น

6) การออกแบบวงจรขับบัสเซอร์ (Buzzer) ซึ่งจะเป็นส่วนของสัญญาณเสียงวงจรบัสเซอร์จะทำหน้าที่ในการส่งสัญญาณเสียงออกมาเพื่อเป็นการบอกสถานะการทำงาน เมื่อมีการกดคีย์บอร์ดและเมื่อมีการรับและส่งข้อมูล การใช้งานในส่วนของวงจรบัสเซอร์จะต่อใช้งานกับพอร์ต P1.2 ของไมโครคอนโทรลเลอร์ โดยสัญญาณเตือนนี้จะมีเสียงลักษณะอย่างไรนั้นจะขึ้นอยู่กับการเขียนโปรแกรมควบคุม

7) วงจรควบคุมอุปกรณ์ จะทำหน้าที่ในการรับและส่งสัญญาณควบคุมการทำงานของอุปกรณ์จากภายนอก เช่น ส่งสัญญาณควบคุมไปยังอุปกรณ์เปิดปิดประตูอัตโนมัติให้ทำการเปิดปิดประตูได้ หรือรับสัญญาณจากอุปกรณ์ตรวจจับไฟไหม้

8) ฐานข้อมูล จะทำหน้าที่ในเก็บรหัสของบัตรแถบแม่เหล็กแล้วจะทำการส่งรหัสบัตรแถบแม่เหล็กไปเก็บไว้ในบอร์ดไมโครคอนโทรลเลอร์เพื่อใช้เป็นรหัสผ่านในการเปิดเข้าห้องพัก และยังใช้เก็บข้อมูลของผู้ที่เข้าพักในส่วนของชื่อ (Name), ที่อยู่ (Address), เบอร์โทรศัพท์ (Telephone), อายุ (Age), สัญชาติ (Nationality)และหมายเลขบัตรเครดิต (Credit card) เป็นต้น นอกจากนี้ยังใช้เก็บข้อมูลในส่วนของห้องพัก เช่น สถานะของห้องพัก (Room status) ราคาของห้องพัก (Room price) ในส่วนของการติดต่อกับฐานข้อมูลจะทำการติดต่อโดยใช้โปรแกรม Visual C++

จากที่ได้ทำการทดลองระบบเช็คอิน - เช็คเอาต์โรงแรมในบทที่ 4 ทำให้สามารถนำเอาผลการทดลองและปัญหาในส่วนต่างๆ มาประเมินผลและสรุปวิเคราะห์ถึงสาเหตุที่ได้ดังนี้

1) ส่วนการทดลองการทำงานของชุดเปิด - ปิดประตูอัตโนมัติ

การทำงานของชุดเปิด-ปิดประตูอัตโนมัติจะทำการจ่ายแรงดันไฟฟ้าเข้าไปที่ตัวเปิดปิดประตูอัตโนมัติ โดยทำการจ่ายแรงดันไฟฟ้าที่ 5 โวลต์และทำการเพิ่มขนาดของแรงดันไฟฟ้าขึ้นทีละ 1 โวลต์ จนกว่าจะถึง 12 โวลต์ พร้อมสังเกตการทำงานของอุปกรณ์เปิดประตูอัตโนมัติว่าอยู่ในสถานะทำงานหรือไม่ทำงาน ซึ่งปรากฏว่าตัวเปิดปิดประตูอัตโนมัติจะทำงานอยู่ในช่วงของขนาดแรงดันไฟฟ้าที่ระดับ 8 โวลต์ขึ้นไป แต่ถ้าหากแรงดันไฟฟ้าต่ำกว่าระดับ 8 โวลต์ ตัวเปิดปิดประตูอัตโนมัติจะไม่สามารถทำงานได้

2) ส่วนการทดลองการทำงานของชุดตรวจสอบไฟไหม้

การทำงานของชุดตรวจสอบไฟไหม้จะทำการต่ออุปกรณ์ตรวจสอบไฟไหม้เข้ากับแหล่งจ่ายไฟฟ้าขนาด 12 โวลต์ แล้วทำการป้อนควันให้กับอุปกรณ์ตรวจจับไฟไหม้ โดยใช้ควันจากธูปในการทดลอง เพื่อจับเวลาในการทำงานของอุปกรณ์ตรวจจับไฟไหม้ และเพิ่มปริมาณของควันไฟสามารถโดยเพิ่มจำนวนธูปขึ้นทีละ 1 ดอก เริ่มตั้งแต่ 1 ดอกและสิ้นสุดที่ 5 ดอก ซึ่งตัวตรวจสอบไฟไหม้จะเริ่มทำงานก็ต่อเมื่อได้รับปริมาณควันที่เหมาะสม ซึ่งถ้าตัวตรวจสอบไฟไหม้ได้รับปริมาณควันมากจะทำงานได้เร็ว

3) ส่วนการทดลองการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดไมโครคอนโทรลเลอร์

การรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดไมโครคอนโทรลเลอร์โดยใช้โปรแกรม Procomm เพื่อแสดงผลการทดลอง โดยการกดคีย์ที่บอร์ดควบคุมที่ประจำตำแหน่งที่ห้องพักส่งไปยังคอมพิวเตอร์ ซึ่งส่วนการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ดควบคุมสามารถรับส่งข้อมูลได้จริง ดังนั้นจึงสามารถทำการผลิตระบบจริงได้

4) ส่วนการทดลองการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คอิน

การแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คอิน โดยทำการเปิดโปรแกรมของระบบเพื่อทำการเช็คอินโดยจะเลือกห้องพักที่ว่าง แล้วทำการกรอกข้อมูลของลูกค้าและเก็บรหัสข้อมูลของบัตรแถบแม่เหล็กเพื่อใช้ในการเปิด - ปิดห้องพัก เมื่อเสร็จแล้วกดปุ่มเช็คอินจะเกิดการเปลี่ยนแปลงที่หน้าต่างโปรแกรมการเช็คอิน - เช็คเอาต์ ที่บอกถึงมีการเข้าพักห้องนั้นแล้ว

5) ส่วนการทดลองการแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คเอาต์

การแสดงผลที่หน้าจอคอมพิวเตอร์เวลาเช็คเอาต์ทำการเปิดโปรแกรมการเช็คอิน - เช็คเอาต์ โดยทำการเลือกห้องพักที่มีคนพักอาศัยอยู่ จะปรากฏหน้าต่างแสดงข้อมูลต่างๆ ของลูกค้าที่มีการระบุวันที่และเวลาเข้าพักจนถึงวันที่และเวลาที่อยู่ในปัจจุบันพร้อมทั้งค่าใช้จ่าย เมื่อเสร็จแล้วกดปุ่มเช็คเอาต์ จะทำให้เกิดการเปลี่ยนแปลงหน้าตาของโปรแกรมการเช็คอิน - เช็คเอาต์ ที่บ่งบอกถึงห้องพักมีการเช็คเอาต์แล้ว

6) ส่วนการทดลองการทำงานของเครื่องอ่านบัตรแถบแม่เหล็ก

การทำงานของเครื่องอ่านบัตรแถบแม่เหล็ก โดยทำการเขียนโปรแกรมรับข้อมูลบัตรแถบแม่เหล็กและเปิดโปรแกรม Procomm เพื่อใช้แสดงผลการอ่านบัตรแถบแม่เหล็ก นำบัตรแถบแม่เหล็กมาทำการรูดที่เครื่องอ่านบัตรแถบแม่เหล็ก เครื่องอ่านบัตรแถบแม่เหล็กสามารถอ่านบัตรแถบแม่เหล็ก และนำไปแสดงรหัสของบัตรแถบแม่เหล็กที่โปรแกรม Procomm

7) ระบบเช็คอิน - เช็คเอาท์โรงแรม

ระบบเช็คอิน - เช็คเอาต์ของโรงแรมจะประกอบด้วยส่วนที่สำคัญ 2 ส่วน คือ ส่วนแรกจะเป็นส่วนของฐานข้อมูลที่เก็บข้อมูลของผู้มาใช้บริการและการแสดงสถานะของห้องพัก ในการเก็บข้อมูลนั้นทางโรงแรมจะจัดเก็บประวัติของผู้เข้ามาใช้บริการ เช่น ชื่อ - สกุล, ที่อยู่, เบอร์โทรศัพท์, สัญชาติ ฯลฯ และในส่วนของการแสดงผลสถานะของห้องพักเป็นการแสดงให้ทราบว่าห้องใดมีผู้เข้าพักแล้วบ้าง ส่วนที่สองเป็นส่วนของห้องพักที่จะทำการเปิดปิดประตูและการตรวจสอบควันไฟจากการเกิดไฟไหม้ ส่วนที่สำคัญของระบบเช็คอิน - เช็คเอาต์ของโรงแรมนี้คือ ส่วนของรีเซ็พชั่น ที่เก็บรหัสของบัตรแถบแม่เหล็กที่ได้จากเครื่องอ่านบัตรแถบแม่เหล็กในส่วนของรีเซ็พชั่นรหัสของบัตรแถบแม่เหล็กและข้อมูลของผู้ใช้บริการจะถูกเก็บไว้ในฐานข้อมูล เพื่อใช้เป็นรหัสผ่านและเก็บ

ไว้เป็นประวัติของผู้ใช้บริการ ส่วนของรหัสที่จัดเก็บไว้ในฐานข้อมูลจะเก็บไว้เพื่อรอการยืนยันรหัสผ่านของบัตรแถบแม่เหล็กที่ได้จากเครื่องอ่านบัตรแถบแม่เหล็กจากส่วนของห้องพัก และอีกส่วนคือส่วนของห้องพักจะเป็นส่วนที่ยืนยันรหัสผ่านของบัตรแถบแม่เหล็กเพื่อเปิดห้องพัก ถ้าการยืนยันรหัสผ่านของบัตรแถบแม่เหล็กที่ได้จากเครื่องอ่านบัตรแถบแม่เหล็กในส่วนของห้องพักไม่ตรงกับรหัสของบัตรแถบแม่เหล็กในส่วนของฐานข้อมูล ไมโครคอนโทรลเลอร์ในส่วนของห้องพักก็ไม่สามารถสั่งให้ประตูเปิดออกได้ แต่ถ้ามีการยืนยันรหัสผ่านของบัตรแถบแม่เหล็กในส่วนของห้องพักตรงกับรหัสของบัตรแถบแม่เหล็กที่ฐานข้อมูลเก็บไว้ไมโครคอนโทรลเลอร์ในส่วนของห้องพักก็จะสามารถสั่งให้ประตูเปิดออกได้ นอกจากนี้ส่วนของห้องพักยังมีการตรวจสอบการเกิดไฟไหม้ภายในห้องพัก โดยจะมีอุปกรณ์ตรวจจับควันไฟเพื่อคอยส่งสัญญาณเตือนแจ้งสถานะไปยังส่วนของรีเซ็พชั่น และระบบเช็คอิน - เช็คเอาต์ของโรงแรมจะมีการรับส่งข้อมูลติดต่อกันอยู่ตลอดเวลาในส่วนของรีเซ็พชั่นกับส่วนของห้องพักทำให้สามารถรับรู้การทำงานของส่วนต่างๆ ได้

5.2 ปัญหาและแนวทางการแก้ไข

จากการดำเนินการสร้างและทดสอบการทำงานของระบบเช็คอิน - เช็คเอาต์โรงแรมพบว่าเกิดปัญหาเกิดขึ้นหลายประการ ซึ่งสามารถสรุปได้ดังนี้

1) ปัญหา จากการวางระบบเช็คอิน - เช็คเอาต์โรงแรมต้องใช้งบประมาณในการจัดทำสูง เพราะอุปกรณ์ที่ใช้ในการทำงานของระบบมีราคาสูง เช่น ตัวอ่านบัตรแถบแม่เหล็ก และอุปกรณ์ตรวจสอบไฟไหม้ เป็นต้น จึงทำให้เกิดการสิ้นเปลืองในส่วนของการซื้ออุปกรณ์มาก

แนวทางแก้ไข ในส่วนของอุปกรณ์ตรวจสอบไฟไหม้ สามารถที่จะหมุนเวียนในการทดลองในแต่ละห้องพักได้ แต่อาจจะต้องใช้เวลานานมากขึ้นในส่วนของการทดลอง และในส่วนของบัตรแถบแม่เหล็กก็เช่นเดียวกัน จึงทำให้เกิดการล่าช้าในการสับเปลี่ยนอุปกรณ์การทดลอง

2) ปัญหา การทำงานของระบบเช็คอิน - เช็คเอาต์ต้องมีการจ่ายแรงดันไฟฟ้าให้กับระบบอยู่ตลอดเวลาเพื่อให้ระบบมีความสมบูรณ์ ถ้าเกิดเหตุการณ์ไฟฟ้าดับจะทำให้ระบบเกิดความผิดพลาดขึ้นได้

แนวทางแก้ไข ต้องจัดทำระบบไฟฟ้าสำรอง เพื่อจ่ายให้กับแก่ระบบเพื่อที่การทำงานของระบบจะได้สมบูรณ์อยู่ตลอดเวลาไม่เกิดปัญหาในเรื่องของกรณีการเกิดไฟฟ้าดับ และในส่วนของห้องพักก็เช่นกันต้องทำการเก็บสถานะเดิมในส่วนของรหัสบัตรแถบแม่เหล็กไว้ด้วย เมื่อระบบเกิดการผิดพลาดหรือเกิดการเปลี่ยนแปลงขึ้นก็จะจำสถานะดังเดิมได้

3) ปัญหา การใช้งานระบบเป็นระยะเวลานาน อาจทำให้อุปกรณ์บางอย่างเกิดการชำรุดเสียหายได้ เช่น อุปกรณ์ในบอร์ดไมโครคอนโทรลเลอร์ ของส่วนห้องพักชำรุดอาจทำให้ระบบไม่สามารถทำงานได้

แนวทางแก้ไข ควรจะมีการตรวจสอบและซ่อมบำรุงรักษาระบบเป็นประจำทุกๆ เดือน เพื่อตรวจสอบอุปกรณ์ทุกส่วนให้มีสภาพสมบูรณ์พร้อมใช้งานอยู่ตลอดเวลา

4) ปัญหา ซอฟแวร์ของระบบในส่วนของการแสดงผลของคอมพิวเตอร์ ยังมีส่วนที่เป็นการคิดค่าใช้จ่ายในการเข้าพักยังไม่สมบูรณ์ ต้องทำการเพิ่มในส่วนของซอฟแวร์ไปด้วย

แนวทางแก้ไข ต้องทำการเก็บรวบรวมข้อมูลในส่วนของการคิดค่าใช้จ่าย ในการเข้าพักเพื่อที่จะนำมาทำการปรับปรุงในส่วนของซอฟแวร์ เพื่อสามารถแสดงผลในส่วนของคอมพิวเตอร์ให้มีความสบบูรณ์มากขึ้น

5.3 แนวทางการพัฒนา

ปัจจุบันมีการแข่งขันทางด้านการโรงแรมมากขึ้น ไม่ว่าจะเป็นในส่วนของการให้บริการหรือด้านสถานที่พักเพื่อดึงดูดลูกค้าให้เข้ามาใช้บริการทางด้านโรงแรมมากขึ้น โดยโรงแรมแต่ละแห่งนั้นต่างก็พยายามดึงเอาสิ่งแปลกใหม่มานำเสนอ เพื่อการตอบสนองความต้องการความสะดวกสบายและความพึงพอใจต่อผู้เข้ามาใช้บริการ จากเหตุผลดังกล่าวสามารถเพิ่มประสิทธิภาพให้กับการเข้าพักในโรงแรมได้มากขึ้นดังต่อไปนี้

1. สามารถตรวจสอบสถานะของห้องพักได้ว่าขณะนี้มีผู้ใช้บริการพักอยู่ภายในห้องพักหรือไม่ ซึ่งจะใช้ในส่วนของตัวตรวจจับความเคลื่อนไหวติดไว้ภายในห้องพัก โดยที่จะไม่ใช้กล้องโทรทัศน์วงจรปิดภายในห้อง

2. พัฒนาระบบโดยการเพิ่มส่วนของการเปิด - ปิด ไฟฟ้าภายในห้องพักแบบอัตโนมัติ ซึ่งจะเป็นการควบคุมในส่วนของหลอดไฟ, เครื่องปรับอากาศและเครื่องใช้ไฟฟ้าต่างๆ ภายในห้องเพื่อเพิ่มความสะดวกสบายยิ่งขึ้น

3. มีการเพิ่มระบบรักษาความปลอดภัยในด้านการเปิด-ปิดห้องพัก โดยการเพิ่มส่วนของการอ่านลายนิ้วมือแทนในส่วนของการอ่านบัตรแถบแม่เหล็กหรือเพิ่มเทคโนโลยีอื่นๆ ได้อีก

4. พัฒนาระบบเช็คอิน - เช็คเอาต์ ให้มีการเข้าห้องพักได้มากกว่า 250 ห้อง

บรรณานุกรม

ธีรวัฒน์ ประกอบผล. **ภาษาแอสเซมบลี สำหรับ MCS -51**. กรุงเทพฯ : สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น). 2546

บัณฑิต จามรภูติ. **คู่มือการใช้งาน Protel 99**. เชียงใหม่ : บัณฑิตเพรส. 2544

ปรเมษฐ์ ประนยานันทน์ และ ปิยพงศ์ เผ่าวณิช. **ไมโครคอนโทรลเลอร์ MCS -51**. กรุงเทพฯ : ซีเอ็ดยูเคชั่น. ม.ป.ป.

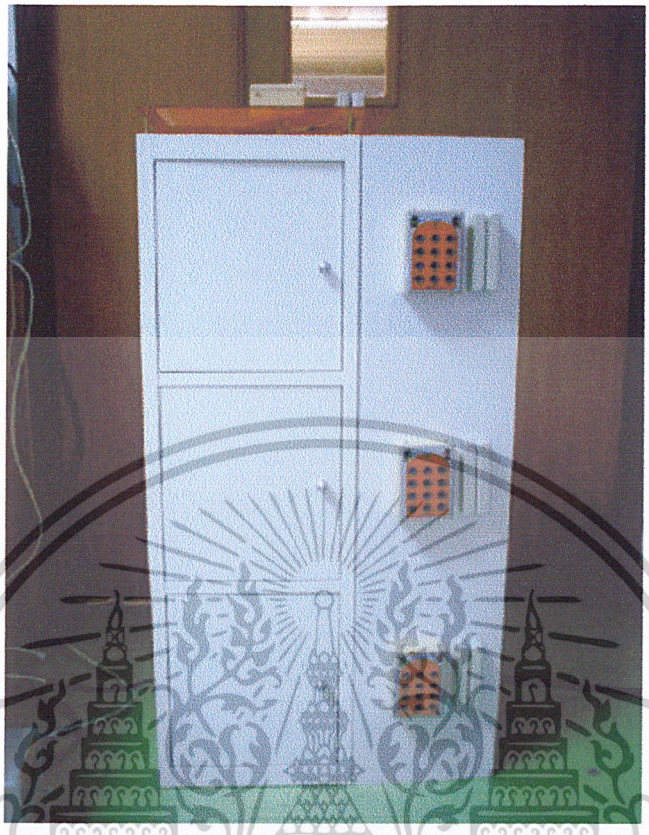
ยุทธนา ลีลาศวัฒนกุล. **คู่มือการเขียนโปรแกรมและใช้งาน Visual C++ 6.0 ฉบับโปรแกรมเมอร์**. นนทบุรี : อินโฟเพรส. 2544

ยุทธนา ลีลาศวัฒนกุล. **คู่มือการเขียนโปรแกรม Advanced Visual C++ Version 6.0 ฉบับ Database Programming**. กรุงเทพฯ : ซัคเซส มีเดีย. ม.ป.ป.

Davis Chapman and Jeff Heaton. **Sams Teach Yourself Visual C++ 6 in 21 Day , Professional Referance Edition**. America : Sams A Division of Macmillan Computer. 1998

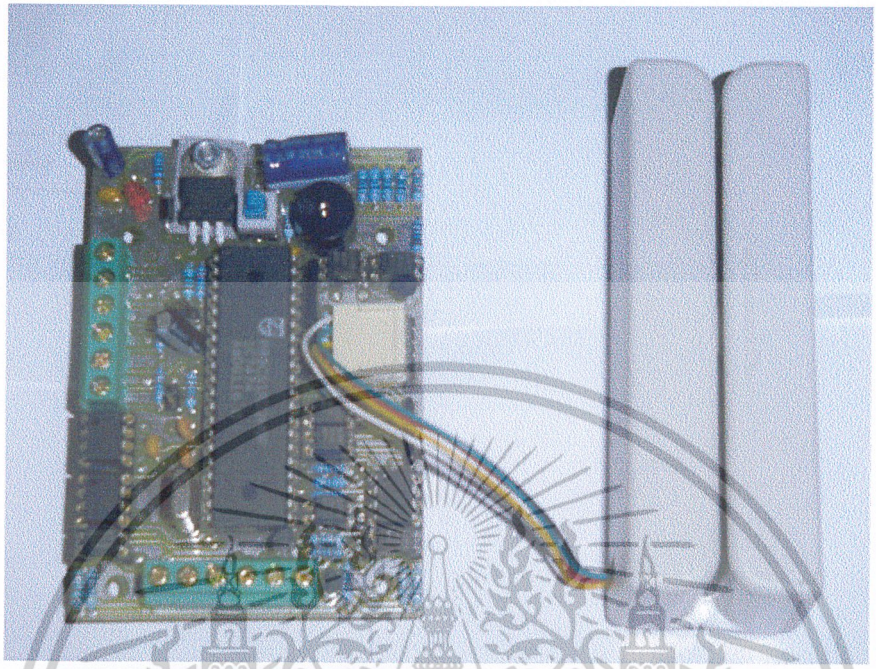
Richard Johnsonbaugh and Martin Kalin. **Applications Programming in C++**. America : Prentice Hall. 1999

ภาคผนวก ก
เครื่องต้นแบบ



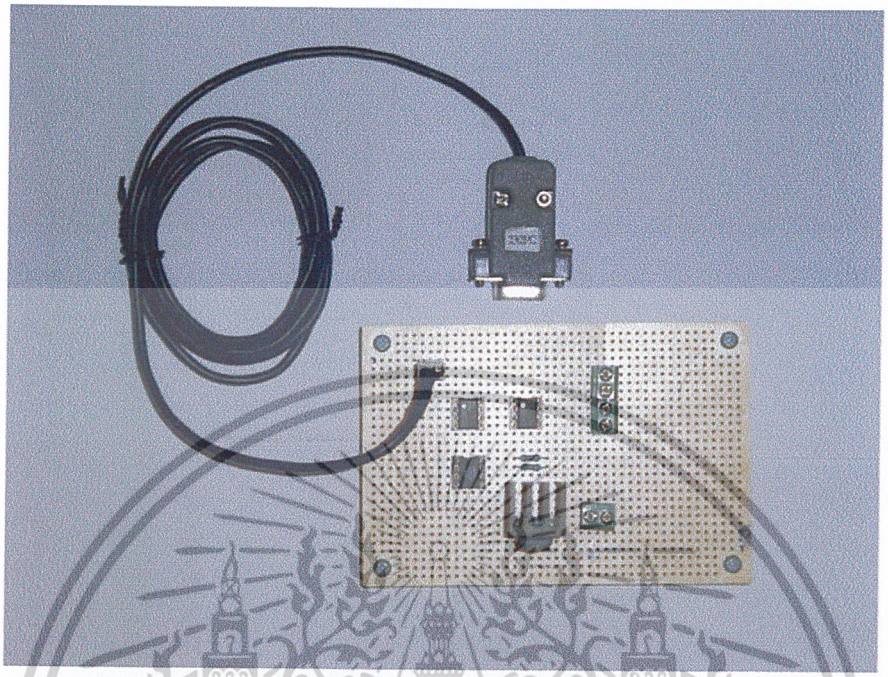
รูปที่ ก.1 ภาพด้านหน้าของระบบเช็คอิน - เช็คเอาต์โรงแรม

รูปที่ ก.2 ภาพชุดเครื่องอ่านบัตรแถบแม่เหล็ก

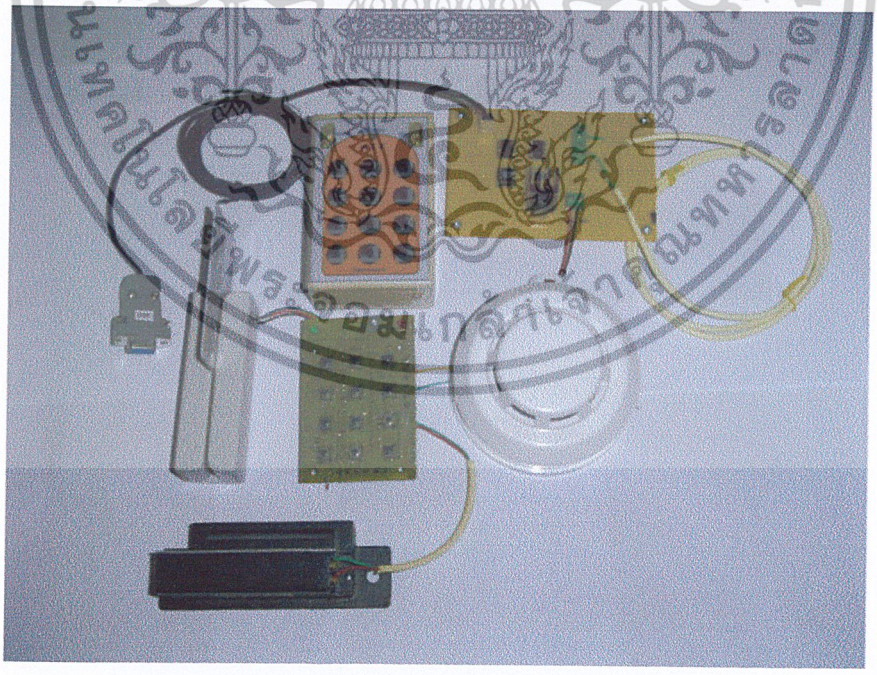


รูปที่ ก.3 ภาพด้านในชุดเครื่องอ่านบัตรแถบแม่เหล็ก

รูปที่ ก.4 ภาพด้านในของระบบเช็คอิน - เช็คเอาต์โรงแรม



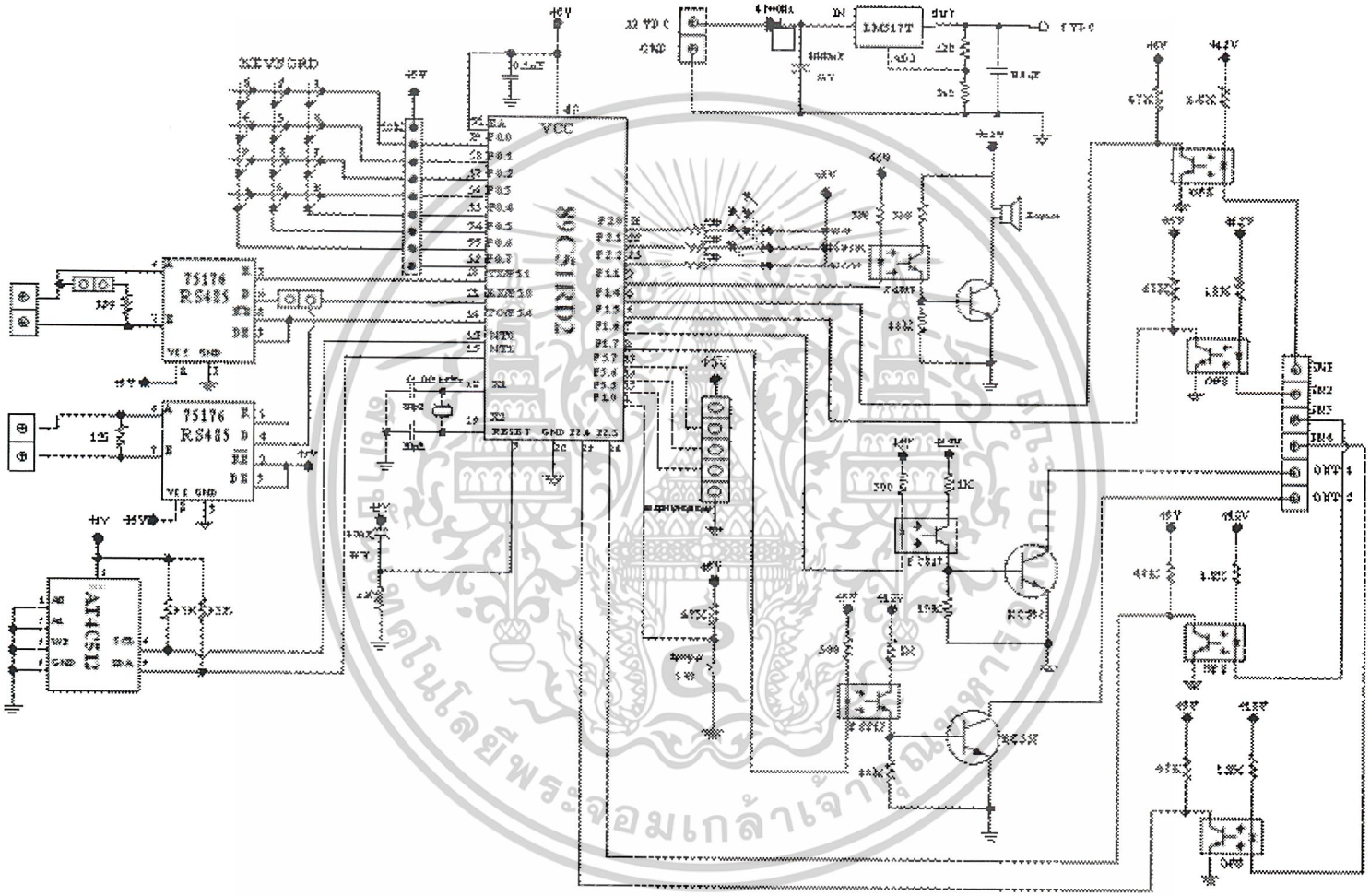
รูปที่ ก.5 ภาพชุดแปลงพอร์ต RS-232 เป็น RS-485



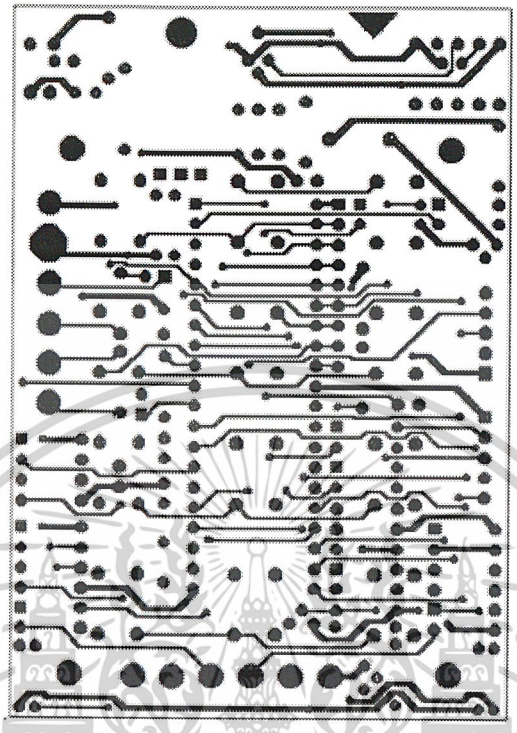
รูปที่ ก.6 ภาพชุดทดลองระบบเช็คอิน - เช็คเอาต์โรงแรม

ภาคผนวก ข

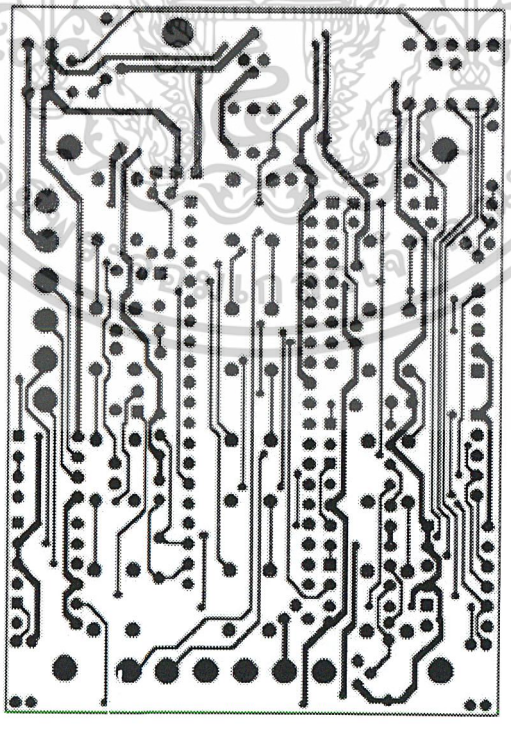
วงจรและแผ่นวงจรพิมพ์



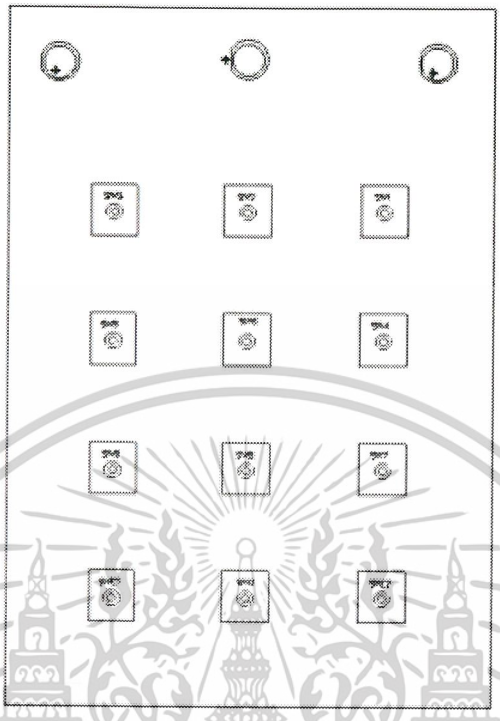
รูปที่ ข.1 วงจรระบบเช็คอิน - เช็คเอาต์โรงแรม



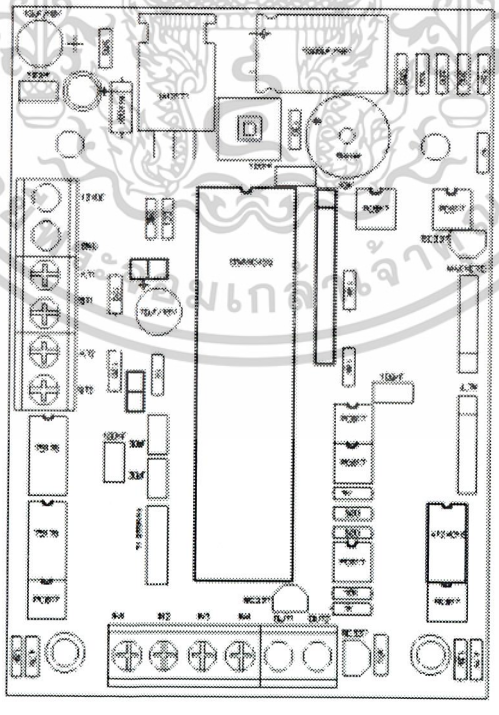
รูปที่ ข.2 แผ่นวงจรพิมพ์วงจรด้านบน



รูปที่ ข.3 แผ่นวงจรพิมพ์วงจรด้านล่าง



รูปที่ ข.4 ตำแหน่งการวางอุปกรณ์แผ่นวงจรพิมพ์ด้านบน



รูปที่ ข.5 ตำแหน่งการวางอุปกรณ์แผ่นวงจรพิมพ์ด้านล่าง

ภาคผนวก ค
รายการอุปกรณ์

ตารางที่ ค.1 รายการอุปกรณ์ของวงจรรับส่งข้อมูล

ชื่ออุปกรณ์	รายละเอียด	จำนวน
1.วงจรรวม	89C51RD2	5 ตัว
	75176	2 ตัว
	AT24C512	1 ตัว
	LM317T	1 ตัว
2. ออปโตคัปเปิ้ล	817	6 ตัว
3. สวิตช์กดติดปล่อยดับ	2 ขา	13 ตัว
4. คอนเน็คเตอร์	9 ขา	1 ตัว
	6 ขา	1 ตัว
	5 ขา	1 ตัว
5. ตัวต้านทาน	120Ω ¼ W	3 ตัว
	330Ω ¼ W	3 ตัว
	360Ω ¼ W	1 ตัว
	500Ω ¼ W	4 ตัว
	1KΩ ¼ W	3 ตัว
	1.8KΩ ¼ W	4 ตัว
	4.7KΩ ¼ W	7 ตัว
	10KΩ ¼ W	3 ตัว
6. ตัวเก็บประจุ	0.1 uF	2 ตัว
	10 uF / 16V	1 ตัว
	1000 uF / 16 v	1 ตัว
	33pF	1 ตัว
7.ไดโอด	LED	3 ตัว
	1N4001	1 ตัว
8. ทรานซิสเตอร์	BC337	3 ตัว
9. บัสเซอร์		1 ตัว
10. ฮีสซิ้งค์		1 ตัว
11. คริสตอล	11.059MHz	1 ตัว

ตารางที่ ค.2 รายการอุปกรณ์ของวงจรแปลงพอร์ต RS-232 เป็น RS-485

ชื่ออุปกรณ์	รายละเอียด	จำนวน
1.วงจรรวม	DS275	1 ตัว
	SN7517	2 ตัว
2. ตัวต้านทาน	120Ω ¼ W	2 ตัว
3. ฮีสซิ้งค์		1 ตัว
4. คอนเน็คเตอร์	2 ขา	1 ตัว
	4 ขา	1 ตัว

ภาคผนวก ง
แผนผังการทำงานและรหัสต้นฉบับของโปรแกรม

โปรแกรมควบคุมการทำงานหลัก

```
                    /*******************************/
                    /*   Access System Version 1.0 */
                    /*        CPU-89C51RD2         */
                    /*  Last update : 00/00/2548   */
                    /*******************************/

#include <at89x52.h>
#include <stdio.h>
#include <absacc.h>
#include <ctype.h>
#include <stdlib.h>
#include <intrins.h>
#include <string.h>

#define     CR      13      /* Constance define */
#define     LF      10
#define     TRUE    1
#define     FALSE   0
#define     BUFSIZE 64

sbit    SoundBit =  P1^1;       /* Buzzer bit */
sbit    ArmLED   =  P2^0;       /* Led bit */
sbit    DarmLED =   P2^1;
sbit    AccsLED =   P2^2;
sbit    CARDDA   =  P3^5;       /* Magnetic card reader */
sbit    CARDCK   =  P3^6;
sbit    CARDCP   =  P3^7;
sbit    RS       =  P3^4;
sbit    OUT1     =  P1^6;   /* out port */
sbit    OUT2     =  P1^7;
sbit    IN1      =  P1^4;   /* in port */
sbit    IN2      =  P1^5;
sbit    IN3      =  P1^6;
sbit        SCL     =   P3^2;       /* I2c SCL bit */
sbit        SDA     =   P3^3;       /* I2c SDA bit */

sfr         AUXR    =   0x8e;    /* Expand internal memory (89C52RD2
only) */
sfr         CCAPM4  =   0xde;   /* Module 4 use for watch dog */
sfr         CCAP4L  =   0xee;
sfr         CCAP4H  =   0xfe;
sfr         CMOD    =   0xd9;
sfr         CH      =   0xf9;
sfr         CL      =   0xe9;
sfr         CCON    =   0xd8;

unsigned     int     data    ID, res;
unsigned    char    xdata   MAGBUF[41];         /* magnetic buffer */
unsigned    char    xdata   MAGOUT[41];         /* magnetic output */
                        char    xdata   ASCBUF[BUFSIZE];     /*
Serial buffer */
                        char    xdata   ASCBUF1[BUFSIZE];   /* Serial
buffer1 */
```

```
char    data    a, b, k_a, k_b;
char    code    scankey[]={0xbf, 0xdf, 0xef};
char    code    keycode[]={3, 6, 9, 12, 2, 5, 8, 0, 1, 4, 7, 11};
unsigned        char    PageAdd, Offset;   /* Use for Write string
to EEPROM */
unsigned        char    code    EEPADD[]={ 0xa0, 0xa2, 0xa4, 0xa6,
0xa8,
                                        0xaa, 0xac, 0xae};
bit     error, COMSEND, COMRDY, empflag, fireflag, RoomFlag, TestBit;

unsigned char data Second, SecDiv60, Minute, TimeOut, ChrIdx,
TimeRoom, TimeLED;
/*
    Watchdog initial
*/
void WDInit ()
{
    CCAPM4=0x4c;    /* Select comaere mode */
    CCAP4L=0x00;    /* 3 Sec of trig */
    CCAP4H=0x01;
    CMOD|=0x44;     /* Enable WD & Use input of PCA from Timer 0 */
    CCON|=0x40;     /* Enable PCA */
}
/*
    Watchdog timer trig
*/
void WDReset (void)
{
    EA=0;
    CCAP4L=CL;    /* Trig by change time of Module4 sub by 1 */
    CCAP4H=CH+1;
    EA=1;
}
/*
    Bit delay
*/
void Bitdl (void)
{
    _nop_();    /* Delay time for read EEPROM */
    _nop_();
}
/*
    Clear(L) SCL line
*/
void Clrscl (void)
{
    SCL = 0;      /* clear line  */
    Bitdl();
}
/*
    Set(H) SCL line
*/
void Setscl (void)
{
    SCL = 1;      /* set line   */
    while(!SCL);
    Bitdl();
}
```

```
/*
    Pulse gen for SCL line
*/
void Plscl (void)
{
    Setscl();
    Clrscl();
}
/*
    Set stop condition
*/

void Stop (void)
{
    SDA = 0;Setscl();
    SDA = 1;Bitdl();
}
/*
    Set start condition
*/
void Start (void)
{
    SDA = 1;
    Setscl();
    SDA = 0;
    Bitdl();
    Clrscl();
}
/*
    Receive data(byte) from I2C bus
*/
char Rxbyte (void)
{
    unsigned char tmp, dat;
    dat = 0;
    for(tmp=1; tmp<=8; tmp++)   /* Send 1 byte to EEPROM */
    {
        dat <<= 1;Setscl();
        dat |= SDA;Clrscl();
    }
    return(dat);
}
/*
    Transmit data(byte) to I2C bus
*/
bit Txbyte (unsigned com)
{
    bit erflag;
    unsigned char tmp;
    for(tmp=1; tmp<=8; tmp++)
    {  SDA   = com & 0x80;
        com <<= 1;
        Plscl();
    }
    SDA = 1;
    Setscl();
    if(SDA) erflag = 1;
    else    erflag = 0;
```

```
    Clrscl();
    return(erflag);
}
/*  Get data from any register  */
/*  in  -->>  address(addr)     */
/*  in  -->>  register(reg)     */
/*  out <<--  data(*dat)        */
/*  err <<--  0 = no error,1 = error */
bit Getreg (char addr,char reg,char *dat)
{
    if((SCL==0) || (SDA==0))
    {                           /* check line first      */
        return(1);
    }
    Start();
    if(Txbyte(addr))
    {                           /* Xmit register address  */
        Stop();
        return(1);
    }
    if(Txbyte(reg))
    {                           /* Xmit register command  */
        Stop();
        return(1);
    }
    Start();                    /* Set start condition    */
    if(Txbyte(addr|0x01))
    {                          /* Xmit register address + receive
command */
        Stop();
        return(1);
    }
    *dat = Rxbyte();           /* Recv data byte         */
    SDA = 1;
    Plscl();
    Stop();
    return(0);                 /* return data to caller  */
}
/*  Put data to any register   */
/*  in  -->>  address(addr)    */
/*  in  -->>  register(reg)    */
/*  in  -->>  data(dat)        */
/*  err <<--  0 = no error,1 = error */
bit Putreg(char addr,char reg,char dat)
{
    if((SCL==0) || (SDA==0))
    {                           /* Ckeck line first      */
        return(1);              /* line is busy return 1 */
    }
    Start();                    /* Set Start condition   */
    if(Txbyte(addr))
    {                           /* Xmit register address */
        Stop();
        return(1);
    }
    if(Txbyte(reg))
    {                           /* Xmit register command */
        Stop();
```

```
        return(1);
    }
    Txbyte(dat);                /* Xmit data              */
    Stop();                     /* Set stop condition     */
    return(0);
}
/*
    Read single byte from EEPROM
*/
char ReadByte (int add)
{
unsigned    char    dat;
    EA=0;
    Offset=add&0x00ff;
    add>>=8;
    PageAdd=EEPADD[add];
    Getreg (PageAdd, Offset, &dat);
    return (dat);
    EA=1;
}
/*
    Write single byte to EEPROM
*/
void WriteByte (int add, char byte)
{
    EA=0;
    Offset=add&0x00ff;
    add>>=8;
    PageAdd=EEPADD[add];
    Putreg (PageAdd, Offset, byte);
    for (res=0; res<512; res++)
        WDReset ();
    EA=1;
}
/*
    Read String from EEPROM
*/
void ReadStr (char *p, int add, int count)
{
    while (count>0)
    {
        *p=ReadByte (add);
        p++;
        add++;
        count--;
    }
}
/*
    Write String to EEPROM
*/
void WriteStr (char *p, int add, int count)
{
    while (count>0)
    {
        WriteByte (add, *p);
        p++;
        add++;
        count--;
```

```
    }
}
/*
    Recieve data from serial port 0
*/
char RcvChr (void)
{
    COMRDY=0;   /* reset flag */
    WDReset ();
    if  (strncmp (ASCBUF+1, "MAP", 3)==0) /* Protocal detect */
    {
        ID=atoi (ASCBUF+5);
        WriteStr (ASCBUF, 0, 20);
        COMSEND=TI=1;
        printf (":MAP=%d\r", ID+1);
        COMSEND=0;
        return (0);
    }
    if  (strncmp (ASCBUF+1, "GETID", 5)==0) /* Protocal detect */
    {
        COMSEND=TI=1;
        printf (":ID=%05d\r", ID);
        printf (":GETID\r");
        COMSEND=0;
        return (0);
    }
    if  (strncmp (ASCBUF+1, "ONID", 4)==0) /* Protocal detect */
    {
        res=atoi (ASCBUF+6);
        if  (ID==res)
        {
            COMSEND=TI=1;
            printf (":IDON=%05d\r", res);
            COMSEND=0;
            return (-2);
        }
    }
    if  (strncmp (ASCBUF+1, "TEST", 4)==0) /* Protocal detect */
        TestBit=1;
    COMSEND=TI=1;
    printf ("%s", ASCBUF);  /* Send to next board */
    COMSEND=0;
    return (-1);    /* No command match */
}
/*
    Delay function
    input : int loop counter
*/
void delay (unsigned int count)
{
    WDReset ();
    for (; count; count--)      /* Loop Delay */
    {
        if  (COMRDY)    /* Serial 0 ready */
            RcvChr ();
    }
}
/*
```

```
    Generate sound
    input : count
*/
void Beep (char count)
{
    if  (count<0)   /* Long beep */
    {
        count=0-count;
        SoundBit=0;         /* turn on buzzer */
        Second=SecDiv60=0;
        while (Second<=count)       /* Delay time */
        {
            WDReset ();
            if  (COMRDY)     /* Serial 0 ready */
                RcvChr ();
        }
        SoundBit=1;      /* Turn off it */
        return;
    }

    for (; count; count--)  /* short beep */
    {
        SoundBit=0;         /* turn on buzzer */
        SecDiv60=0;
        while (SecDiv60<4)      /* Delay time */
        {
            WDReset ();
            if  (COMRDY)     /* Serial 0 ready */
                RcvChr ();
        }
        SoundBit=1;      /* Turn off it */
        SecDiv60=0;
        while (SecDiv60<4)      /* delay time */
        {
            WDReset ();
            if  (COMRDY)     /* Serial 0 ready */
                RcvChr ();
        }
    }
    SoundBit=1;
}

/*
    Check Keyboard hit
*/
bit kbhit (void)
{
    P0=0x8f;
    if  ((P0&0x0f)!=0x0f)   /* Key hit return true */
        return (TRUE);
    else
        return (FALSE);     /* Non return false */
}
/*
    Read keyboard
*/
char keyboard (void)
{
```

```
    while (1)
    {
        Minute=0;   /* Reset minute time out */
        for (k_a=0; k_a<3; k_a++)   /* Loop scan 4 bit */
        {
            WDReset ();
            P0=scankey[k_a];    /* Check key press */
            k_b=P0&0x0f;
            if  (k_b==0x0f)
                continue;
            else
            {
//                delay (1000);   /* Key debouce */
                k_b=P0&0x0f;
                if  (k_b==0x0f)
                    continue;
                if  (k_b==0x0e)      /* bit 1 */
                    k_b=0;
                if  (k_b==0x0d)      /* bit 2 */
                    k_b=1;
                if  (k_b==0x0b)      /* bit 3 */
                    k_b=2;
                if  (k_b==0x07)      /* bit 4 */
                    k_b=3;
                k_a*=4;
                k_b+=k_a;
                k_b=keycode[k_b];   /* Key encode */
                Beep (1);
                TimeOut=0;
                while (kbhit ())     /* Loop while depress */
                {
                    WDReset ();
                    if  (COMRDY)
                        RcvChr ();
                }
                TestBit=0;
                return (k_b);   /* return key */
            }
        }
    }
}


/*
    Timer interrupt
*/
static void timer0_isr (void) interrupt 1 using 1
{
    SecDiv60++;
    TimeLED++;
    if  (SecDiv60==60)       /* 1/60 sec */
    {
        SecDiv60=0;
        TimeOut++;
        TimeRoom++;
        Second++;
        if  (Second==60)     /* 1 sec */
        {
```

```
            Second=0;
            Minute++;       /* minute */
        }

        if  (TimeOut>=3)     /* timer for serial */
            ChrIdx=0;

        if  (RoomFlag)  /* timer for room flag */
        {
            if  (TimeRoom>30)
            {
                RoomFlag=0; /* turn of all */
                OUT2=1;
            }
        }
    }
    TH0 = 0x88;                             /* high byte */
    TL0 = 0;                                /* reload timer 0 value low
byte  */
    TR0 = 1;                                /* start timer 0
*/
}
/*
    Magnetic card read
*/
void magread (void)
{
    unsigned char a,i,j;     /* private data */
    bit dat,first;
//    while (CARDCP);                       /* wait for CP=0 */
    a = 0;
    i = 0;
    j = 0;
    first = 1;
    while (!CARDCP)
    {
        while (CARDCK)
        {                                   /* wait for CK=0 */
            if (CARDCP) return;
        }
        _nop_ ();
        _nop_ ();
        dat = CARDDA;
        if (first)
        {
            if (!dat)
                first = 0;        /* don't care first bit = 1 */
        }
        if (~first)
        {
            a = a >> 1;
            if (dat)
                a = a | 0x80;
            i++;
            if (i==5)
            {
                a = a >> 3;
                a = ~a & 0x1f;
```

```
                MAGBUF[j] = a;
                j++;
                i = 0;
            }
        }
        while (!CARDCK);                // wait for CK=1
    }
}
/*
    Magnetic card check
*/
bit magchk (void)
{                       /* magnetic check parity */
    unsigned char a,b,i,j;              /* and change to ascii (to
MAGOUT) */
    bit flag;
    for (i=0;i<=39;i++)
    {
        a = MAGBUF[i];
        if (a==0)
            break;
        b = 0;
        for (j=0;j<=4;j++)
        {
            flag = a & 1;
            a = a >> 1;
            if (flag) b++;
        }
        if ((b % 2)==0)
            return (1);
        a = MAGBUF[i] & 0xf;
        if (a>=0xa)
            a = (a - 9) | 0x40;
        else
            a = a | 0x30;
        MAGOUT[i] = a;
    }
    if (MAGOUT[0]!='B')
        return (1);
    else
        return (0);
}


/*
    Magnetic reverse direction
*/
void magreverse (void)
{                                       /* magnetic reverse data */
    unsigned char a,b,i,j,p,x,y;
    bit dat,skip;

    for (i=0;i<=39;i++)
        MAGOUT[i] = 0;                          /* clear MAGOUT */
    i = 40; p = 0;
    x = 0;
    y = 0;
    skip = 1;
```

```
    while (i>0)
    {
        i--;
        a = MAGBUF[i];
        if (skip && a==0);                                    /* nothing
*/
        else
        {
            a = a << 3;
            for (j=1;j<=5;j++)
            {
                b = a & 0x80;
                a = a << 1;
                if (b==0x80)
                    dat = 1;
                else dat = 0;
                if (skip && dat==1)
                    skip = 0;                    /* find first bit = 1 */
                if (!skip)
                {                                        /* reverse data */
                    if (dat)
                        p = p | 0x80;
                    else
                        p = p & 0x7f;
                    p = p >> 1; y++;
                    if (y==5)
                    {
                        MAGOUT[x++] = p >> 2;
                        p = 0; y = 0;
                    }
                }
            }
        }
    }

    if (y>0)
    {                                /* check end data */
        while (y!=5) { p = p >> 1; y++; }
        MAGOUT[x] = p >> 2;
    }

    for (i=0;i<=39;i++)
        MAGBUF[i] = MAGOUT[i];                /* move to MAGBUF */
}

/*
    Clear buffer
*/
void magclear (void)
{                    /* magnetic clear buffer */
    unsigned char i;
    for (i=0;i<=40;i++)
    {
        MAGBUF[i] = 0;
        MAGOUT[i] = 0;
    }
}
/*
```

```
    Interrupt from serial port
*/
static void serial (void) interrupt 4 using 1
{
    if  (RI)    /* Recieve bit detected */
    {
        TimeOut=0;
        if  (COMRDY)    /* Command process not finish */
        {
            ChrIdx=0;
            getkey ();
        }
        else
        {
            ASCBUF[ChrIdx]=getkey ();   /* Put to buffer */
            ASCBUF[ChrIdx+1]=0;
            RI=0;   /* Clear recieve bit */

            if  ((ASCBUF[ChrIdx]==CR)||(ASCBUF[ChrIdx]==LF))    /*
Cr/Lf detected */
            {
                ChrIdx=0;   /* Zero pointer */
                COMRDY=1;
            }
            else
                ChrIdx++;
        }

        if  (ChrIdx>=BUFSIZE-1)   /* Buffer full */
            ChrIdx=0;           /* Zero index */
    }

    if  ((TI==1)&&(COMSEND==0))     /* Clear send flag */
        TI=0;
}
/*
    Main program
*/
void main (void)
{
    AUXR=0;                     /* Used Expand memory (89c52RD2 only) */
    SCL      = 0;               /* Excite I2C line */
    SDA      = 0;
    delay (256);
    SCL      = 1;
    SDA      = 1;

    SCON=0x52;      /* Init serial port */
    TMOD=0x21;      /* Set timer mode */
    PCON=0;
    TH1=0xfd;       /* 19200 for RD2 */
    TR1=1;      /* Enable interrupt */
    TH0=0x88;       /* Timer interrupt 1/60 sec for RD2 */
    TL0=0;
    ET0=1;          /* Timer/Counter 0 interrupt enable */
    TR0=1;
    ES=1;       /* enable serial interrupt */
    EA=1;           /* Enable all interrupt */
```

```
    WDInit ();  /* enable watchdog */
    ReadStr(ASCBUF, 0, 20);
    ID=atoi (ASCBUF+5);
    COMRDY=RS=0;
    empflag=0;
    fireflag=0;
    RoomFlag=0;
    TestBit=0;
//    COMSEND=TI=1;
//    printf ("Program start\n");
//    COMSEND=0;
    while (1)
    {
        if  (TestBit)
        {
            ArmLED^=1;
            DarmLED=1;
        }
        else
        {
            DarmLED^=1;
            ArmLED=1;
        }
        TimeLED=0;
        while (TimeLED<=30)
        {
            WDReset ();
            if  (!IN2)  /* check fire alarm room */
            {
                if  (!fireflag)
                {
                    fireflag=1;
                    COMSEND=TI=1;
                    printf (":ID=%05d FIRE\r", ID);
                    COMSEND=0;
                    SecDiv60=0;
                    while (SecDiv60<30)       /* Delay time */
                    {
                        WDReset ();
                        if  (COMRDY)    /* Serial 0 ready */
                            RcvChr ();
                    }
                }
            }
            else
            {
                if  (fireflag)
                {
                    fireflag=0;
                    SecDiv60=0;
                    while (SecDiv60<30)       /* Delay time */
                    {
                        WDReset ();
                        if  (COMRDY)    /* Serial 0 ready */
                            RcvChr ();
                    }
                }
            }
```

```
if  (!IN1)  /* check in room */
{
    if  (!empflag)
    {
        empflag=1;
        OUT2=0;
        COMSEND=TI=1;
        printf (":ID=%05d NOEMP\r", ID);
        COMSEND=0;
        SecDiv60=0;
        while (SecDiv60<30)       /* Delay time */
        {
            WDReset ();
            if  (COMRDY)     /* Serial 0 ready */
                RcvChr ();
        }
    }
}
else    /* no peple in room */
{
    if  (empflag)
    {
        empflag=0;
        COMSEND=TI=1;
        printf (":ID=%05d EMP\r", ID);
        COMSEND=0;
        SecDiv60=0;
        while (SecDiv60<30)       /* Delay time */
        {
            WDReset ();
            if  (COMRDY)     /* Serial 0 ready */
                RcvChr ();
        }
        TimeRoom=0;
        RoomFlag=1;
    }
}
if  (COMRDY)
{
    if  (RcvChr ()==-2)  /* Turn on solinoi */
    {
        OUT1=0;
        DarmLED=1;
        ArmLED=1;
        AccsLED=0;
        Beep (-3);  /* delay 3 second */
        AccsLED=1;
        OUT1=1;
    }
}
if  (kbhit ())  /* key check */
{
    a=keyboard ();
    COMSEND=TI=1;
    printf (":ID=%05d Key=%d\r", ID, (int)a);
    COMSEND=0;
}
```

```
            if  (!CARDCP)
            {
                magclear ();                        /* clear buffer */
                magread ();                        /* read raw data */
                error = magchk ();                 /* check data */
                if (error)
                {                       /* if error type reverse */
                    magreverse ();
                    error = magchk ();
                }
                if (error)  /* read error */
                {
                    COMSEND=TI=1;
                    printf (":ID=%05d Card error.\r", ID);
                    COMSEND=0;
                }
                else    /* card ready */
                {
                    COMSEND=TI=1;
                    printf (":ID=%05d Card=%s\r", ID, MAGOUT);
                    COMSEND=0;
                }
            }
        }
    }
}
```

โปรแกรมแสดงผลบนคอมพิวเตอร์

```
//
//    Class:            CButtonST
//
//    Compiler:   Visual C++
//    Tested on:  Visual C++ 5.0
//                      Visual C++ 6.0
//
//    Version:    See GetVersionC() or GetVersionI()
//
//    Created:    xx/xxxx/1998
//    Updated:    17/October/2001
//
//    Author:           Davide Calabro'
      davide_calabro@yahoo.com
//
      http://www.softechsoftware.it
//
//    Note:       Code for the PreSubclassWindow and OnSetStyle
functions
//                has been taken from the COddButton class
//                published by Paolo Messina and Jerzy
Kaczorowski
//
//    Disclaimer
//    ----------
//    THIS SOFTWARE AND THE ACCOMPANYING FILES ARE DISTRIBUTED "AS
IS" AND WITHOUT
//    ANY WARRANTIES WHETHER EXPRESSED OR IMPLIED. NO REPONSIBILITIES
FOR POSSIBLE
//    DAMAGES OR EVEN FUNCTIONALITY CAN BE TAKEN. THE USER MUST
ASSUME THE ENTIRE
//    RISK OF USING THIS SOFTWARE.
//
//    Terms of use
//    ------------
//    THIS SOFTWARE IS FREE FOR PERSONAL USE OR FREEWARE
APPLICATIONS.
//    IF YOU USE THIS SOFTWARE IN COMMERCIAL OR SHAREWARE
APPLICATIONS YOU
//    ARE GENTLY ASKED TO DONATE 1$ (ONE U.S. DOLLAR) TO THE AUTHOR:
//
//          Davide Calabro'
//          P.O. Box 65
//          21019 Somma Lombardo (VA)
//          Italy
//
#ifndef _BTNST_H
#define _BTNST_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
// Return values
#ifndef     BTNST_OK
#define     BTNST_OK                                    0
#endif
#ifndef     BTNST_INVALIDRESOURCE
#define     BTNST_INVALIDRESOURCE                       1
#endif
#ifndef     BTNST_FAILEDMASK
#define     BTNST_FAILEDMASK                            2
#endif
#ifndef     BTNST_INVALIDINDEX
#define     BTNST_INVALIDINDEX                          3
#endif
#ifndef     BTNST_INVALIDALIGN
#define     BTNST_INVALIDALIGN                          4
#endif

class CButtonST : public CButton
{
public:
    CButtonST();
      ~CButtonST();

    enum    {     ST_ALIGN_HORIZ     = 0,              // Icon/bitmap
on the left, text on the right
                  ST_ALIGN_VERT,                       //
Icon/bitmap on the top, text on the bottom
                  ST_ALIGN_HORIZ_RIGHT                 //
Icon/bitmap on the right, text on the left
                };

      enum  {     BTNST_COLOR_BK_IN = 0,         // Background color
when mouse is INside
                  BTNST_COLOR_FG_IN,                   //
Text color when mouse is INside
                  BTNST_COLOR_BK_OUT,                  //
Background color when mouse is OUTside
                  BTNST_COLOR_FG_OUT,                  //
Text color when mouse is OUTside
                  BTNST_COLOR_BK_FOCUS,                //
Background color when the button is focused
                  BTNST_COLOR_FG_FOCUS,                // Text
color when the button is focused

                  BTNST_MAX_COLORS
                };

      // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CButtonST)
      public:
      virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
      virtual BOOL PreTranslateMessage(MSG* pMsg);
      protected:
      virtual void PreSubclassWindow();
      virtual LRESULT DefWindowProc(UINT message, WPARAM wParam,
LPARAM lParam);
      //}}AFX_VIRTUAL
```

```
public:
	DWORD SetSound(LPCTSTR lpszSound, HMODULE hMod = NULL, BOOL
bPlayOnClick = FALSE, BOOL bPlayAsync = TRUE);
	DWORD SetDefaultColors(BOOL bRepaint = TRUE);
	DWORD SetColor(BYTE byColorIndex, COLORREF crColor, BOOL
bRepaint = TRUE);
	DWORD GetColor(BYTE byColorIndex, COLORREF* crpColor);

	DWORD SetCheck(int nCheck, BOOL bRepaint = TRUE);
	int GetCheck();

	DWORD SetURL(LPCTSTR lpszURL = NULL);
	void DrawTransparent(BOOL bRepaint = FALSE);

	BOOL GetDefault();
	DWORD SetAlwaysTrack(BOOL bAlwaysTrack = TRUE);

	void SetTooltipText(int nText, BOOL bActivate = TRUE);
	void SetTooltipText(LPCTSTR lpszText, BOOL bActivate = TRUE);
	void ActivateTooltip(BOOL bEnable = TRUE);

	DWORD SetBtnCursor(int nCursorId = NULL, BOOL bRepaint = TRUE);

	DWORD SetFlat(BOOL bFlat = TRUE, BOOL bRepaint = TRUE);
	DWORD SetAlign(BYTE byAlign, BOOL bRepaint = TRUE);

	DWORD DrawBorder(BOOL bDrawBorder = TRUE, BOOL bRepaint =
TRUE);
	DWORD DrawFlatFocus(BOOL bDrawFlatFocus, BOOL bRepaint = TRUE);

	DWORD SetIcon(int nIconIn, int nIconOut = NULL);
	DWORD SetIcon(HICON hIconIn, HICON hIconOut = NULL);

	DWORD SetBitmaps(int nBitmapIn, COLORREF crTransColorIn, int
nBitmapOut = NULL, COLORREF crTransColorOut = 0);
	DWORD SetBitmaps(HBITMAP hBitmapIn, COLORREF crTransColorIn,
HBITMAP hBitmapOut = NULL, COLORREF crTransColorOut = 0);

	DWORD SetMenu(UINT nMenu, HWND hParentWnd, BOOL bRepaint =
TRUE);

	static short GetVersionI()		{return 34;}
	static LPCTSTR GetVersionC()  {return (LPCTSTR)_T("3.4");}

protected:
    //{{AFX_MSG(CButtonST)
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message);
	afx_msg void OnKillFocus(CWnd* pNewWnd);
	afx_msg void OnMouseMove(UINT nFlags, CPoint point);
	afx_msg void OnSysColorChange();
	afx_msg BOOL OnClicked();
	afx_msg void OnActivate(UINT nState, CWnd* pWndOther, BOOL
bMinimized);
	afx_msg void OnEnable(BOOL bEnable);
	afx_msg void OnCancelMode();
	afx_msg UINT OnGetDlgCode();
	//}}AFX_MSG
```

```
	afx_msg HBRUSH CtlColor(CDC* pDC, UINT nCtlColor);
	virtual DWORD OnDrawBackground(CDC* pDC, LPCRECT pRect);
	virtual DWORD OnDrawBorder(CDC* pDC, LPCRECT pRect);

	BOOL		m_bIsFlat;				// Is a flat button?
	BOOL		m_bMouseOnButton; // Is mouse over the button?
	BOOL		m_bDrawTransparent;		// Draw transparent?
	BOOL		m_bIsPressed;			// Is button pressed?
	BOOL		m_bIsFocused;			// Is button focused?
	BOOL		m_bIsDisabled;			// Is button disabled?
	BOOL		m_bIsDefault;			// Is default button?
	BOOL		m_bIsCheckBox;			// Is the button a
checkbox?
	BYTE		m_byAlign;				// Align mode
	BOOL		m_bDrawBorder;			// Draw border?
	BOOL		m_bDrawFlatFocus; // Draw focus rectangle for flat
button?
	COLORREF	m_crColors[BTNST_MAX_COLORS]; // Colors to be used
	HMENU		m_hMenu;				// Handle to associated
menu
	HWND		m_hParentWndMenu; // Handle to window for menu
selection
	BOOL		m_bMenuDisplayed; // Is menu displayed ?

private:
	LRESULT OnSetStyle(WPARAM wParam, LPARAM lParam);
	LRESULT OnMouseLeave(WPARAM wParam, LPARAM lParam);

	void CancelHover();
	void FreeResources(BOOL bCheckForNULL = TRUE);
	void PrepareImageRect(BOOL bHasTitle, RECT* rpItem, CRect*
rpTitle, BOOL bIsPressed, DWORD dwWidth, DWORD dwHeight, CRect*
rpImage);
	HBITMAP CreateBitmapMask(HBITMAP hSourceBitmap, DWORD dwWidth,
DWORD dwHeight, COLORREF crTransColor);
	void DrawTheIcon(CDC* pDC, BOOL bHasTitle, RECT* rpItem, CRect*
rpTitle, BOOL bIsPressed, BOOL bIsDisabled);
	void DrawTheBitmap(CDC* pDC, BOOL bHasTitle, RECT *rItem, CRect
*rCaption, BOOL bIsPressed, BOOL bIsDisabled);
	void PaintBk(CDC* pDC);

	void InitToolTip();

	HCURSOR		m_hCursor;				// Handle to cursor
	CToolTipCtrl m_ToolTip;			// Tooltip

	CDC		m_dcBk;
	CBitmap		m_bmpBk;
	CBitmap*	m_pbmpOldBk;

	BOOL		m_bAlwaysTrack;			// Always hilight button?
	int		m_nCheck;				// Current value for
checkbox
	UINT		m_nTypeStyle;			// Button style

	TCHAR		m_szURL[_MAX_PATH];		// URL to open when clicked
```

```
#pragma pack(1)
      typedef struct _STRUCT_ICONS
      {
            HICON       hIcon;                  // Handle to icon
            DWORD       dwWidth;          // Width of icon
            DWORD       dwHeight;         // Height of icon
      } STRUCT_ICONS;
#pragma pack()

#pragma pack(1)
      typedef struct _STRUCT_BITMAPS
      {
            HBITMAP           hBitmap;          // Handle to bitmap
            DWORD       dwWidth;          // Width of bitmap
            DWORD       dwHeight;         // Height of bitmap
            HBITMAP           hMask;                  // Handle to
mask bitmap
            COLORREF    crTransparent;    // Transparent color
      } STRUCT_BITMAPS;
#pragma pack()

      STRUCT_ICONS      m_csIcons[2];
      STRUCT_BITMAPS    m_csBitmaps[2];

#pragma pack(1)
      typedef struct _STRUCT_SOUND
      {
            TCHAR       szSound[_MAX_PATH];
            LPCTSTR           lpszSound;
            HMODULE           hMod;
            DWORD       dwFlags;
      } STRUCT_SOUND;
#pragma pack()

      STRUCT_SOUND      m_csSounds[2];    // Index 0 = Over 1 =
Clicked

      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
immediately before the previous line.

#endif
#if
!defined(AFX_MAPPING_H__61D9C1CB_9355_4227_909C_625C2DC150E0__INCLUDE
D_)
#define
AFX_MAPPING_H__61D9C1CB_9355_4227_909C_625C2DC150E0__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Mapping.h : header file
//
```

```
////////////////////////////////////////////////////////////////////////
////////
// CMapping dialog

//#include "OLETB2Set.h"

class CMapping : public CDialog
{
// Construction
public:
	CByteArray m_rClass;
	CByteArray m_sT;
	CWordArray m_rPrice;
	CWordArray m_rID;
	CWordArray m_mID;

	CMapping(CWnd* pParent = NULL);   // standard constructor
	~CMapping ();

// Dialog Data
	//{{AFX_DATA(CMapping)
	enum { IDD = IDD_MAPPING };
	CListCtrl   m_clist;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMapping)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CMapping)
	virtual BOOL OnInitDialog();
	virtual void OnCancel();
	virtual void OnOK();
	afx_msg void OnDblclkListMap(NMHDR* pNMHDR, LRESULT* pResult);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_MAPPING_H__61D9C1CB_9355_4227_909C_625C2DC150E0__INCLUDE
D_)
#if
!defined(AFX_MYLISTCTRL_H__1F7F8718_2E10_4B12_ADF3_7AE55105CDD0__INCL
UDED_)
```

```
#define
AFX_MYLISTCTRL_H__1F7F8718_2E10_4B12_ADF3_7AE55105CDD0__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MyListCtrl.h : header file
//

/////////////////////////////////////////////////////////////////////////
////////
// CMyListCtrl window

class CMyListCtrl : public CListCtrl
{
// Construction
public:
      CMyListCtrl();

// Attributes
public:

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CMyListCtrl)
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CMyListCtrl();

      // Generated message map functions
protected:
      //{{AFX_MSG(CMyListCtrl)
            // NOTE - the ClassWizard will add and remove member
functions here.
      //}}AFX_MSG

      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////
////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_MYLISTCTRL_H__1F7F8718_2E10_4B12_ADF3_7AE55105CDD0__INCL
UDED_)
// OLETB1Set.h : interface of the COLETB1Set class
//
/////////////////////////////////////////////////////////////////////////
////////
```

```
#if
!defined(AFX_OLETB1SET_H__05CF00B8_97E9_4A4D_9845_C16C1293E61C__INCLU
DED_)
#define
AFX_OLETB1SET_H__05CF00B8_97E9_4A4D_9845_C16C1293E61C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTable1
{
public:
	CTable1()
	{
		memset( (void*)this, 0, sizeof(*this) );
	};

	wchar_t m_Address[128];
	DATE m_CheckInDate;
	DATE m_CheckOutDate;
	wchar_t m_City[65];
	wchar_t m_Country[65];
	wchar_t m_CreditCard[65];
	float m_Discount;
	short m_MachineID;
	wchar_t m_MasterCardID[65];
	wchar_t m_MateCardID[65];
	wchar_t m_Name[65];
	wchar_t m_Nationality[65];
	wchar_t m_PersonalID[65];
	wchar_t m_Postal[65];
	wchar_t m_Res1[65];
	wchar_t m_Res2[65];
	unsigned char m_RoomClass;
	float m_RoomCost;
	short m_RoomNumber;
	unsigned char m_RoomStatus;
	short m_StayDay;
	wchar_t m_Surname[65];
	wchar_t m_TelNumber[65];
	float m_Total;
	wchar_t m_UserCardID[65];


BEGIN_COLUMN_MAP(CTable1)
		COLUMN_ENTRY_TYPE(17, DBTYPE_WSTR, m_Address)
		COLUMN_ENTRY_TYPE(9, DBTYPE_DATE, m_CheckInDate)
		COLUMN_ENTRY_TYPE(12, DBTYPE_DATE, m_CheckOutDate)
		COLUMN_ENTRY_TYPE(18, DBTYPE_WSTR, m_City)
		COLUMN_ENTRY_TYPE(19, DBTYPE_WSTR, m_Country)
		COLUMN_ENTRY_TYPE(21, DBTYPE_WSTR, m_CreditCard)
		COLUMN_ENTRY_TYPE(23, DBTYPE_R4, m_Discount)
		COLUMN_ENTRY_TYPE(1, DBTYPE_I2, m_MachineID)
		COLUMN_ENTRY_TYPE(5, DBTYPE_WSTR, m_MasterCardID)
		COLUMN_ENTRY_TYPE(6, DBTYPE_WSTR, m_MateCardID)
		COLUMN_ENTRY_TYPE(13, DBTYPE_WSTR, m_Name)
```

```
            COLUMN_ENTRY_TYPE(16, DBTYPE_WSTR, m_Nationality)
            COLUMN_ENTRY_TYPE(15, DBTYPE_WSTR, m_PersonalID)
            COLUMN_ENTRY_TYPE(20, DBTYPE_WSTR, m_Postal)
            COLUMN_ENTRY_TYPE(8, DBTYPE_WSTR, m_Res1)
            COLUMN_ENTRY_TYPE(11, DBTYPE_WSTR, m_Res2)
            COLUMN_ENTRY_TYPE(4, DBTYPE_UI1, m_RoomClass)
            COLUMN_ENTRY_TYPE(3, DBTYPE_R4, m_RoomCost)
            COLUMN_ENTRY_TYPE(2, DBTYPE_I2, m_RoomNumber)
            COLUMN_ENTRY_TYPE(25, DBTYPE_UI1, m_RoomStatus)
            COLUMN_ENTRY_TYPE(10, DBTYPE_I2, m_StayDay)
            COLUMN_ENTRY_TYPE(14, DBTYPE_WSTR, m_Surname)
            COLUMN_ENTRY_TYPE(22, DBTYPE_WSTR, m_TelNumber)
            COLUMN_ENTRY_TYPE(24, DBTYPE_R4, m_Total)
            COLUMN_ENTRY_TYPE(7, DBTYPE_WSTR, m_UserCardID)
END_COLUMN_MAP()

};

class COLETB1Set : public CCommand<CAccessor<CTable1> >
{
public:

      CString          m_strFilter;

      HRESULT Open()
      {
            CDataSource db;
            CSession    session;
            HRESULT          hr;

            CDBPropSet  dbinit(DBPROPSET_DBINIT);
            dbinit.AddProperty(DBPROP_AUTH_CACHE_AUTHINFO, true);
            dbinit.AddProperty(DBPROP_AUTH_ENCRYPT_PASSWORD, false);
            dbinit.AddProperty(DBPROP_AUTH_MASK_PASSWORD, false);
            dbinit.AddProperty(DBPROP_AUTH_PASSWORD, "");
            dbinit.AddProperty(DBPROP_AUTH_USERID, "Admin");
            dbinit.AddProperty(DBPROP_INIT_DATASOURCE,
"D:\\PAC\\Debug\\RemoteDB.mdb");
            dbinit.AddProperty(DBPROP_INIT_MODE, (long)16);
            dbinit.AddProperty(DBPROP_INIT_PROMPT, (short)4);
            dbinit.AddProperty(DBPROP_INIT_PROVIDERSTRING, "");
            dbinit.AddProperty(DBPROP_INIT_LCID, (long)1033);

      dbinit.AddProperty(DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO,
false);

            hr =
db.OpenWithServiceComponents("Microsoft.Jet.OLEDB.4.0", &dbinit);
            if (FAILED(hr))
                  return hr;

            hr = session.Open(db);
            if (FAILED(hr))
                  return hr;

            CDBPropSet  propset(DBPROPSET_ROWSET);
            propset.AddProperty(DBPROP_CANFETCHBACKWARDS, true);
            propset.AddProperty(DBPROP_IRowsetScroll, true);
```

```
            propset.AddProperty(DBPROP_IRowsetChange, true);
            propset.AddProperty(DBPROP_UPDATABILITY,
DBPROPVAL_UP_CHANGE | DBPROPVAL_UP_INSERT | DBPROPVAL_UP_DELETE );

            hr = CCommand<CAccessor<CTable1> >::Open(session, "SELECT
* FROM Table1"+ m_strFilter, &propset);
            if (FAILED(hr))
                  return hr;

            return MoveNext();
      }

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_OLETB1SET_H__05CF00B8_97E9_4A4D_9845_C16C1293E61C__INCLU
DED_)
// OLETB2Set.h : interface of the COLETB2Set class
//
/////////////////////////////////////////////////////////////////////
////////

#if
!defined(AFX_OLETB2SET_H__971F569D_4F65_42B0_93FF_F7DF19D16746__INCLU
DED_)
#define
AFX_OLETB2SET_H__971F569D_4F65_42B0_93FF_F7DF19D16746__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTable2
{
public:
      CTable2()
      {
            memset( (void*)this, 0, sizeof(*this) );
      };

      wchar_t m_Address[128];
      wchar_t m_Age[65];
      DATE m_CheckInDate;
      DATE m_CheckOutDate;
      wchar_t m_Country[65];
      wchar_t m_CreditCard[65];
      float m_Discount;
      short m_MachineID;
      wchar_t m_MasterCardID[65];
      wchar_t m_MateCardID[65];
      wchar_t m_Name[65];
      wchar_t m_Nationality[65];
      wchar_t m_PersonalID[65];
      wchar_t m_Postal[65];
```

```
	wchar_t m_Res1[65];
	wchar_t m_Res2[65];
	unsigned char m_RoomClass;
	float m_RoomCost;
	short m_RoomNumber;
	unsigned char m_RoomStatus;
	short m_StayDay;
	wchar_t m_Surname[65];
	wchar_t m_TelNumber[65];
	float m_Total;
	wchar_t m_UserCardID[65];


BEGIN_COLUMN_MAP(CTable2)
		COLUMN_ENTRY_TYPE(17, DBTYPE_WSTR, m_Address)
		COLUMN_ENTRY_TYPE(18, DBTYPE_WSTR, m_Age)
		COLUMN_ENTRY_TYPE(9, DBTYPE_DATE, m_CheckInDate)
		COLUMN_ENTRY_TYPE(12, DBTYPE_DATE, m_CheckOutDate)
		COLUMN_ENTRY_TYPE(19, DBTYPE_WSTR, m_Country)
		COLUMN_ENTRY_TYPE(21, DBTYPE_WSTR, m_CreditCard)
		COLUMN_ENTRY_TYPE(23, DBTYPE_R4, m_Discount)
		COLUMN_ENTRY_TYPE(1, DBTYPE_I2, m_MachineID)
		COLUMN_ENTRY_TYPE(5, DBTYPE_WSTR, m_MasterCardID)
		COLUMN_ENTRY_TYPE(6, DBTYPE_WSTR, m_MateCardID)
		COLUMN_ENTRY_TYPE(13, DBTYPE_WSTR, m_Name)
		COLUMN_ENTRY_TYPE(16, DBTYPE_WSTR, m_Nationality)
		COLUMN_ENTRY_TYPE(15, DBTYPE_WSTR, m_PersonalID)
		COLUMN_ENTRY_TYPE(20, DBTYPE_WSTR, m_Postal)
		COLUMN_ENTRY_TYPE(8, DBTYPE_WSTR, m_Res1)
		COLUMN_ENTRY_TYPE(11, DBTYPE_WSTR, m_Res2)
		COLUMN_ENTRY_TYPE(4, DBTYPE_UI1, m_RoomClass)
		COLUMN_ENTRY_TYPE(3, DBTYPE_R4, m_RoomCost)
		COLUMN_ENTRY_TYPE(2, DBTYPE_I2, m_RoomNumber)
		COLUMN_ENTRY_TYPE(25, DBTYPE_UI1, m_RoomStatus)
		COLUMN_ENTRY_TYPE(10, DBTYPE_I2, m_StayDay)
		COLUMN_ENTRY_TYPE(14, DBTYPE_WSTR, m_Surname)
		COLUMN_ENTRY_TYPE(22, DBTYPE_WSTR, m_TelNumber)
		COLUMN_ENTRY_TYPE(24, DBTYPE_R4, m_Total)
		COLUMN_ENTRY_TYPE(7, DBTYPE_WSTR, m_UserCardID)
END_COLUMN_MAP()

};

class COLETB2Set : public CCommand<CAccessor<CTable2> >
{
public:

	CString		m_strFilter;

	HRESULT Open()
	{
		CDataSource db;
		CSession	session;
		HRESULT		hr;

		CDBPropSet	dbinit(DBPROPSET_DBINIT);
		dbinit.AddProperty(DBPROP_AUTH_CACHE_AUTHINFO, true);
		dbinit.AddProperty(DBPROP_AUTH_ENCRYPT_PASSWORD, false);
```

```
            dbinit.AddProperty(DBPROP_AUTH_MASK_PASSWORD, false);
            dbinit.AddProperty(DBPROP_AUTH_PASSWORD, "");
            dbinit.AddProperty(DBPROP_AUTH_USERID, "Admin");
            dbinit.AddProperty(DBPROP_INIT_DATASOURCE,
"D:\\PAC\\Debug\\RemoteDB.mdb");
            dbinit.AddProperty(DBPROP_INIT_MODE, (long)16);
            dbinit.AddProperty(DBPROP_INIT_PROMPT, (short)4);
            dbinit.AddProperty(DBPROP_INIT_PROVIDERSTRING, "");
            dbinit.AddProperty(DBPROP_INIT_LCID, (long)1033);

      dbinit.AddProperty(DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO,
false);

            hr =
db.OpenWithServiceComponents("Microsoft.Jet.OLEDB.4.0", &dbinit);
            if (FAILED(hr))
                  return hr;

            hr = session.Open(db);
            if (FAILED(hr))
                  return hr;

            CDBPropSet  propset(DBPROPSET_ROWSET);
            propset.AddProperty(DBPROP_CANFETCHBACKWARDS, true);
            propset.AddProperty(DBPROP_IRowsetScroll, true);
            propset.AddProperty(DBPROP_IRowsetChange, true);
            propset.AddProperty(DBPROP_UPDATABILITY,
DBPROPVAL_UP_CHANGE | DBPROPVAL_UP_INSERT | DBPROPVAL_UP_DELETE );

            hr = CCommand<CAccessor<CTable2> >::Open(session, "SELECT
* FROM Table2"+ m_strFilter, &propset);
            if (FAILED(hr))
                  return hr;

            return MoveNext();
      }

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_OLETB2SET_H__971F569D_4F65_42B0_93FF_F7DF19D16746__INCLU
DED_)
// OLETB3Set.h : interface of the COLBTB3Set class
//
////////////////////////////////////////////////////////////////////
////////

#if
!defined(AFX_OLBTB3SET_H__8A7B800C_B5BC_445A_934F_465D18D5DB06__INCLU
DED_)
#define
AFX_OLBTB3SET_H__8A7B800C_B5BC_445A_934F_465D18D5DB06__INCLUDED_

#if _MSC_VER > 1000
```

```
#pragma once
#endif // _MSC_VER > 1000

class CTable3
{
public:
	CTable3()
	{
		memset( (void*)this, 0, sizeof(*this) );
	};

	wchar_t m_CardID[26];
	DATE m_EntryDate;
	DATE m_EntryTime;
	unsigned char m_Event;
	int m_ID;
	DATE m_OutDate;
	DATE m_OutTime;
	int m_RoomNumber;


BEGIN_COLUMN_MAP(CTable3)
		COLUMN_ENTRY_TYPE(4, DBTYPE_WSTR, m_CardID)
		COLUMN_ENTRY_TYPE(6, DBTYPE_DATE, m_EntryDate)
		COLUMN_ENTRY_TYPE(5, DBTYPE_DATE, m_EntryTime)
		COLUMN_ENTRY_TYPE(3, DBTYPE_UI1, m_Event)
		COLUMN_ENTRY_TYPE(1, DBTYPE_I4, m_ID)
		COLUMN_ENTRY_TYPE(8, DBTYPE_DATE, m_OutDate)
		COLUMN_ENTRY_TYPE(7, DBTYPE_DATE, m_OutTime)
		COLUMN_ENTRY_TYPE(2, DBTYPE_I4, m_RoomNumber)
END_COLUMN_MAP()

};

class COLETB3Set : public CCommand<CAccessor<CTable3> >
{
public:

	CString m_strFilter; // for search
	HRESULT Open()
	{
		CDataSource db;
		CSession    session;
		HRESULT           hr;

		CDBPropSet  dbinit(DBPROPSET_DBINIT);
		dbinit.AddProperty(DBPROP_AUTH_CACHE_AUTHINFO, true);
		dbinit.AddProperty(DBPROP_AUTH_ENCRYPT_PASSWORD, false);
		dbinit.AddProperty(DBPROP_AUTH_MASK_PASSWORD, false);
		dbinit.AddProperty(DBPROP_AUTH_PASSWORD, "");
		dbinit.AddProperty(DBPROP_AUTH_USERID, "Admin");
		dbinit.AddProperty(DBPROP_INIT_DATASOURCE,
"D:\\PAC\\Debug\\RemoteDB.mdb");
		dbinit.AddProperty(DBPROP_INIT_MODE, (long)16);
		dbinit.AddProperty(DBPROP_INIT_PROMPT, (short)4);
		dbinit.AddProperty(DBPROP_INIT_PROVIDERSTRING, "");
		dbinit.AddProperty(DBPROP_INIT_LCID, (long)1033);
```

```
      dbinit.AddProperty(DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO,
false);

            hr =
db.OpenWithServiceComponents("Microsoft.Jet.OLEDB.4.0", &dbinit);
            if (FAILED(hr))
                  return hr;

            hr = session.Open(db);
            if (FAILED(hr))
                  return hr;

            CDBPropSet  propset(DBPROPSET_ROWSET);
            propset.AddProperty(DBPROP_CANFETCHBACKWARDS, true);
            propset.AddProperty(DBPROP_IRowsetScroll, true);
            propset.AddProperty(DBPROP_IRowsetChange, true);
            propset.AddProperty(DBPROP_UPDATABILITY,
DBPROPVAL_UP_CHANGE | DBPROPVAL_UP_INSERT | DBPROPVAL_UP_DELETE );

            hr = CCommand<CAccessor<CTable3> >::Open(session, "SELECT
* FROM Table3", &propset);
            if (FAILED(hr))
                  return hr;

            return MoveNext();
      }

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_OLBTB3SET_H__8A7B800C_B5BC_445A_934F_465D18D5DB06__INCLU
DED_)
// PACDlg.h : header file
//

#if
!defined(AFX_PACDLG_H__37FEC92E_1494_4960_AA4C_0D4F8292BEC8__INCLUDED
_)
#define AFX_PACDLG_H__37FEC92E_1494_4960_AA4C_0D4F8292BEC8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////
////////
// CPACDlg dialog

#include "afxdlgs.h"
#include "ButtonST.h"
#include "RoomChkIn.h"
#include "RoomChkOut.h"
#include "WinXPButtonST.h"
```

```
#include "XPStyleButtonST.h"
#include "Mapping.h"

#include "OLETB1Set.h"
#include "OLETB2Set.h"
#include "OLETB3Set.h"
#include "SerialPort.h" // Added by ClassView
#include "SerialCfg.h"


#define TRANSPARENTCOLOR RGB(255, 0, 255);

class CPACDlg : public CDialog
{
// Construction
public:
      void ClearBuffer();
      void CheckInOut (int rNo);
      CRoomChkIn* pRoomChkIn;
      CRoomChkOut* pRoomChkOut;
      CSerialPort PortComx;
      int PortNumber;
      int CharIndex;
      char ASCBUF[1024];

//    CWordArray      m_mID;
//    CWordArray      m_rID;
//    CWordArray      m_rPrice;
//    CByteArray      m_rClass;
//    CStringArray    m_uCardID;
//    CStringArray    m_cInTime;
//    CStringArray    m_cInDate;
//    CwordArray      m_sTay;
//    CStringArray    m_cOutTime;
//    CStringArray    m_cOutDate;


//    CByteArray m_sT;

      COLETB1Set* m_pSet1;
      COLETB2Set* m_pSet2;
      COLETB3Set* m_pSet3;
      CPACDlg(CWnd* pParent = NULL);      // standard constructor

// Dialog Data
      //{{AFX_DATA(CPACDlg)
      enum { IDD = IDD_PAC_DIALOG };
      //}}AFX_DATA

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CPACDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);      //
DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:
```

```
	HICON m_hIcon;
	void OnRXChar (WPARAM wParam, LPARAM lParam);

	// Generated message map functions
	//{{AFX_MSG(CPACDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnDestroy();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnClose();
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	virtual void OnOK();
	afx_msg void On101();
	afx_msg void On102();
	afx_msg void OnAbout();
	afx_msg void OnSetup();
	afx_msg void OnSkin();
	afx_msg void OnMap();
	afx_msg void On103();
	afx_msg void On104();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:

	CButtonST	m_btnExit;
	CButtonST	m_btnHelp;
	CButtonST	m_btnSet;
	CButtonST	m_btnAbout;
	CButtonST	m_btnSkin;

	CButtonST	m_btn[10];

	CButtonST	m_btnMap;
//	CThemeHelperST		m_ThemeHelper;

	BITMAP			m_Bitmap;
	HBITMAP			m_hBitmap;

	CString			m_bkImg;

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_PACDLG_H__37FEC92E_1494_4960_AA4C_0D4F8292BEC8__INCLUDED
_)
#if
!defined(AFX_PROPERTYLIST_H__74205380_1B56_11D4_BC48_00105AA2186F__IN
CLUDED_)
#define
AFX_PROPERTYLIST_H__74205380_1B56_11D4_BC48_00105AA2186F__INCLUDED_

#if _MSC_VER > 1000
```

```
#pragma once
#endif // _MSC_VER > 1000
// PropertyList.h : header file
//

#define PIT_COMBO 0  //PIT = property item type
#define PIT_EDIT  1
#define PIT_COLOR 2
#define PIT_FONT  3
#define PIT_FILE  4

#define IDC_PROPCMBBOX   712
#define IDC_PROPEDITBOX  713
#define IDC_PROPBTNCTRL  714


/////////////////////////////////////////////////////////////////////////
////////
//CPropertyList Items
class CPropertyItem
{
// Attributes
public:
      CString m_propName;
      CString m_curValue;
      int m_nItemType;
      CString m_cmbItems;

public:
      CPropertyItem(CString propName, CString curValue,
                    int nItemType, CString cmbItems)
      {
            m_propName = propName;
            m_curValue = curValue;
            m_nItemType = nItemType;
            m_cmbItems = cmbItems;
      }
};

/////////////////////////////////////////////////////////////////////////
////////
// CPropertyList window

class CPropertyList : public CListBox
{
// Construction
public:
      CPropertyList();

// Attributes
public:

// Operations
public:
      int AddItem(CString txt);
      int AddPropItem(CPropertyItem* pItem);

// Overrides
```

```
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CPropertyList)
	public:
	virtual void MeasureItem(LPMEASUREITEMSTRUCT
lpMeasureItemStruct);
	virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
	protected:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	virtual void PreSubclassWindow();
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CPropertyList();

	// Generated message map functions
protected:
	//{{AFX_MSG(CPropertyList)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnSelchange();
	afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
	afx_msg void OnKillFocus(CWnd* pNewWnd);
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnMouseMove(UINT nFlags, CPoint point);
	//}}AFX_MSG
	afx_msg void OnKillfocusCmbBox();
	afx_msg void OnSelchangeCmbBox();
	afx_msg void OnKillfocusEditBox();
	afx_msg void OnChangeEditBox();
	afx_msg void OnButton();

	DECLARE_MESSAGE_MAP()

	void InvertLine(CDC* pDC,CPoint ptFrom,CPoint ptTo);
	void DisplayButton(CRect region);

	CComboBox m_cmbBox;
	CEdit m_editBox;
	CButton m_btnCtrl;
	CFont m_SSerif8Font;

	int m_curSel,m_prevSel;
	int m_nDivider;
	int m_nDivTop;
	int m_nDivBtm;
	int m_nOldDivX;
	int m_nLastBox;
	BOOL m_bTracking;
	BOOL m_bDivIsSet;
	HCURSOR m_hCursorArrow;
	HCURSOR m_hCursorSize;
};

/////////////////////////////////////////////////////////////////////////
////////

//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_PROPERTYLIST_H__74205380_1B56_11D4_BC48_00105AA2186F__IN
CLUDED_)
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by PAC.rc
//
#define IDC_BUTTON_PRINT                3
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_PAC_DIALOG                  102
#define IDR_MAINFRAME                   128
#define IDI_CANCEL1                     129
#define IDI_CANCEL2                     130
#define IDI_EMP                         132
#define IDI_EMP1                        133
#define IDB_SKY                         134
#define IDI_SETUP                       135
#define IDC_HAND                        136
#define IDI_HELP                        137
#define IDD_CHECKIN                     141
#define IDI_OK                          142
#define IDD_CHECKOUT                    143
#define IDB_BITMAP1                     144
#define IDR_WAVEOHO                     145
#define IDR_WAVECLICK                   146
#define IDI_ABOUT                       147
#define IDI_SKIN                        148
#define IDD_MAPPING                     149
#define IDD_SERIAL                      150
#define IDI_FIRE                        152
#define IDI_NOEMP                       153
#define IDI_NOEMP1                      154
#define IDI_NOT                         160
#define IDI_NOT1                        161
#define IDI_EMP2                        164
#define IDB_BITMAP2                     165
#define IDC_SETUP                       1000
#define IDC_101                         1001
#define IDC_102                         1002
#define IDC_103                         1003
#define IDC_EDIT_FIRSTNAME              1003
#define IDC_104                         1004
#define IDC_EDIT_LASTNAME               1004
#define IDC_EDIT_AGE                    1006
#define IDC_EDIT_ADDRESS                1008
#define IDC_EDIT_CARDID                 1009
#define IDC_EDIT_PHONE                  1010
#define IDC_EDIT_STAY                   1011
#define IDC_EDIT_CREDIT                 1012
#define IDC_EDIT_INDATE                 1015
#define IDC_EDIT_OUTDATE                1016
#define IDC_EDIT_TOTAL_DAY              1017
#define IDC_EDIT_DISCOUNT               1018
```

```
#define IDC_STATIC_FNAME                1019
#define IDC_STATIC_ROOMINF              1020
#define IDC_STATIC_ABOUT                1021
#define IDC_ABOUT                       1022
#define IDC_SKIN                        1023
#define IDC_MAP                         1024
#define IDC_LIST_MAP                    1025
#define IDC_PARITYCOMBO                 1027
#define IDC_DATABITSCOMBO               1028
#define IDC_STOPBITSCOMBO               1029
#define IDC_BAUDRATECOMBO               1030
#define IDC_PORTNO                      1031
#define IDC_CHECK1                      1034
#define IDC_EDIT_COUNTRY                1034
#define IDC_CHECK2                      1035
#define IDC_EDIT_PASSNO                 1035
#define IDC_CHECK3                      1036
#define IDC_EDIT_NATION                 1036
#define IDC_CHECK4                      1037
#define IDC_STATIC_TOTAL                1037
#define IDC_CHECK5                      1038
#define IDC_STATIC_PRICE                1038
#define IDC_CHECK6                      1039
#define IDC_CHECK7                      1040
#define IDC_CHECK8                      1041
#define IDC_CHECK9                      1042
#define IDC_SENDBUFFERCOMBO             1043

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        166
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1043
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
#if
!defined(AFX_ROOMCHKIN_H__81A4C30B_DC24_4674_9CF1_F74454899B06__INCLU
DED_)
#define
AFX_ROOMCHKIN_H__81A4C30B_DC24_4674_9CF1_F74454899B06__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// RoomChkIn.h : header file
//

/////////////////////////////////////////////////////////////////////////
////////
// CRoomChkIn dialog

#include "XPStyleButtonST.h"

class CRoomChkIn : public CDialog
{
```

```
// Construction
public:
	CString m_strWindowText;
	CRoomChkIn(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CRoomChkIn)
	enum { IDD = IDD_CHECKIN };
	CString	m_strAddress;
	CString	m_strAge;
	CString	m_strCredit;
	CString	m_strFirstname;
	CString	m_strLastname;
	CString	m_strPhone;
	CString	m_strStay;
	CString	m_strCountry;
	CString	m_strNation;
	CString	m_strPassno;
	CString	m_strCardID;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CRoomChkIn)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:
	void OnCardID (WPARAM wParam, LPARAM lParam);

	// Generated message map functions
	//{{AFX_MSG(CRoomChkIn)
	virtual BOOL OnInitDialog();
	afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT
nCtlColor);
	virtual void OnOK();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	CXPStyleButtonST  m_btnChk;
	CXPStyleButtonST  m_btnCancel;

	CThemeHelperST          m_ThemeHelper;

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_ROOMCHKIN_H__81A4C30B_DC24_4674_9CF1_F74454899B06__INCLU
DED_)
```

```
#if
!defined(AFX_ROOMCHKOUT_H__3E2D78AB_24AD_4E79_A62B_4DDDDE1B40A4__INCL
UDED_)
#define
AFX_ROOMCHKOUT_H__3E2D78AB_24AD_4E79_A62B_4DDDDE1B40A4__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// RoomChkOut.h : header file
//

/////////////////////////////////////////////////////////////////////////
////////
// CRoomChkOut dialog

#include "XPStyleButtonST.h"

class CRoomChkOut : public CDialog
{
// Construction
public:
      CString m_strWindowText;
      float Total;
      CRoomChkOut(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CRoomChkOut)
      enum { IDD = IDD_CHECKOUT };
      CString     m_strAddress;
      CString     m_strAge;
      CString     m_strCredit;
      CString     m_strDiscount;
      CString     m_strFirstname;
      CString     m_strLastname;
      CString     m_strPhone;
      CString     m_strStay;
      CString     m_strCountry;
      CString     m_strNation;
      CString     m_strPassno;
      CString     m_strTotalday;
      CString     m_strPrice;
      CString     m_strTotal;
      CString     m_strChkinDate;
      CString     m_strChkoutDate;
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CRoomChkOut)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:
```

```
	// Generated message map functions
	//{{AFX_MSG(CRoomChkOut)
	virtual BOOL OnInitDialog();
	afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT
nCtlColor);
	afx_msg void OnUpdateEditDiscount();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	CXPStyleButtonST  m_btnChk;
	CXPStyleButtonST  m_btnCancel;
	CXPStyleButtonST  m_btnPrint;

	CThemeHelperST          m_ThemeHelper;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_ROOMCHKOUT_H__3E2D78AB_24AD_4E79_A62B_4DDDDE1B40A4__INCL
UDED_)
#if
!defined(AFX_SERIALCFG_H__3E397E3D_3CA2_4451_A03A_87C374C21B0F__INCLU
DED_)
#define
AFX_SERIALCFG_H__3E397E3D_3CA2_4451_A03A_87C374C21B0F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SerialCfg.h : header file
//

/////////////////////////////////////////////////////////////////////
////////
// CSerialCfg dialog

class CSerialCfg : public CDialog
{
// Construction
public:
	CSerialCfg (CWnd* pParent, DCB dcb);
	CSerialCfg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CSerialCfg)
	enum { IDD = IDD_SERIAL };
	CString     m_strBaudRate;
	BOOL  m_CommBreakDetected;
	BOOL  m_CommCTSDetected;
	BOOL  m_CommDSRDetected;
	BOOL  m_CommERRDetected;
	BOOL  m_CommRingDetected;
	BOOL  m_CommRLSDDetected;
	BOOL  m_CommRXChar;
```

```
	BOOL	m_CommRXCharFlag;
	BOOL	m_CommTXEmpty;
	CString	m_strDataBits;
	CString	m_strParity;
	CString	m_strSendBuff;
	CString	m_strStopBits;
	int	m_PortNo;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSerialCfg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:
	DCB m_dcb;

	// Generated message map functions
	//{{AFX_MSG(CSerialCfg)
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_SERIALCFG_H__3E397E3D_3CA2_4451_A03A_87C374C21B0F__INCLU
DED_)
// SerialPort.h: interface for the CSerialPort class.
//
////////////////////////////////////////////////////////////////////
/

#if
!defined(AFX_SERIALPORT_H__7331CCC9_73C3_42EF_9643_8CF05851EFC1__INCL
UDED_)
#define
AFX_SERIALPORT_H__7331CCC9_73C3_42EF_9643_8CF05851EFC1__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CSerialPort : public CObject
{
public:
	BOOL StopMonitor ();
	void WriteToPort (char* string);
	DWORD GetWriteBufferSize ();
	DWORD GetCommEvents();
```

```
      DCB GetDCB ();
      BOOL StartMonitor ();
      BOOL InitSerialPort (CWnd* PortOwner, UINT PortNo=1, UINT
Baud=19200,
            char Parity='N', UINT DataBits=8, UINT StopBits=1,
            DWORD dwCommEvent=EV_RXCHAR|EV_CTS, UINT BuffSize=512);
      CSerialPort();
      virtual ~CSerialPort();

protected:
      void ProcessErrorMessage (char* ErrorText);
      static void WriteChar (CSerialPort* port);
      static void ReceiveChar (CSerialPort* port, COMSTAT comstat);
      static UINT CommThread (LPVOID pParam);
      UINT m_nPortNr;
      CRITICAL_SECTION m_csCommunicationSync;

      CWinThread* m_Thread;
      BOOL m_bThreadAlive;

      HANDLE m_hWriteEvent;
      HANDLE m_hShutdownEvent;
      HANDLE m_hComm;

      HANDLE m_hEventArray[3];

      OVERLAPPED m_ov;
      COMMTIMEOUTS m_CommTimeouts;
      DCB m_dcb;

      CWnd* m_pOwner;
      char* m_szWriteBuffer;
      DWORD m_dwCommEvents;
      DWORD m_nWriteBufferSize;
};

#endif //
!defined(AFX_SERIALPORT_H__7331CCC9_73C3_42EF_9643_8CF05851EFC1__INCL
UDED_)
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if
!defined(AFX_STDAFX_H__B198B6EB_B419_4874_B872_AFE338D64468__INCLUDED
_)
#define AFX_STDAFX_H__B198B6EB_B419_4874_B872_AFE338D64468__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN          // Exclude rarely-used stuff from
Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
```

```
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>         // MFC support for Internet Explorer 4
Common Controls

#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>                    // MFC ODBC database classes
#endif // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>                   // MFC DAO database classes
#endif // _AFX_NO_DAO_SUPPORT

#include <afxdtctl.h>         // MFC support for Internet Explorer 4
Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                   // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <atlbase.h>
extern CComModule _Module;
#include <atlcom.h>
#include <atldbcli.h>
#include <afxoledb.h>


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_STDAFX_H__B198B6EB_B419_4874_B872_AFE338D64468__INCLUDED
_)
//
//    Class:            CThemeHelperST
//
//    Compiler:   Visual C++
//    Tested on:  Visual C++ 6.0
//
//    Version:    See GetVersionC() or GetVersionI()
//
//    Created:    09/January/2002
//    Updated:    13/January/2002
//
//    Author:           Davide Calabro'
      davide_calabro@yahoo.com
//
      http://www.softechsoftware.it
//
//    Note:       Based on the CVisualStylesXP code
//                      published by David Yuheng Zhao
(yuheng_zhao@yahoo.com)
//
//    Disclaimer
//    ----------
//    THIS SOFTWARE AND THE ACCOMPANYING FILES ARE DISTRIBUTED "AS
IS" AND WITHOUT
```

```
//    ANY WARRANTIES WHETHER EXPRESSED OR IMPLIED. NO REPONSIBILITIES
FOR POSSIBLE
//    DAMAGES OR EVEN FUNCTIONALITY CAN BE TAKEN. THE USER MUST
ASSUME THE ENTIRE
//    RISK OF USING THIS SOFTWARE.
//
//    Terms of use
//    ------------
//    THIS SOFTWARE IS FREE FOR PERSONAL USE OR FREEWARE
APPLICATIONS.
//    IF YOU USE THIS SOFTWARE IN COMMERCIAL OR SHAREWARE
APPLICATIONS YOU
//    ARE GENTLY ASKED TO DONATE 1$ (ONE U.S. DOLLAR) TO THE AUTHOR:
//
//          Davide Calabro'
//          P.O. Box 65
//          21019 Somma Lombardo (VA)
//          Italy
//
#ifndef _THEMEHELPERST_H_
#define _THEMEHELPERST_H_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef     HTHEME
#define     HTHEME      HANDLE
#endif

class CThemeHelperST
{
public:
      CThemeHelperST();
      virtual ~CThemeHelperST();

      HTHEME OpenThemeData(HWND hwnd, LPCWSTR pszClassList);
      HRESULT CloseThemeData(HTHEME hTheme);
      HRESULT DrawThemeBackground(HTHEME hTheme, HDC hdc, int
iPartId, int iStateId, const RECT* pRect, const RECT* pClipRect);
      HRESULT DrawThemeText(HTHEME hTheme, HDC hdc, int iPartId, int
iStateId, LPCWSTR pszText, int iCharCount, DWORD dwTextFlags, DWORD
dwTextFlags2, const RECT* pRect);
      BOOL IsThemeActive();
      BOOL IsAppThemed();

      static short GetVersionI()          {return 10;}
      static LPCTSTR GetVersionC()  {return (LPCTSTR)_T("1.0");}

private:
      LPVOID GetProc(LPCSTR szProc, LPVOID pfnFail);

      typedef HTHEME(__stdcall *PFNOPENTHEMEDATA)(HWND hwnd, LPCWSTR
pszClassList);
      static HTHEME __stdcall OpenThemeDataFail(HWND, LPCWSTR)
      {return NULL;}

      typedef HRESULT(__stdcall *PFNCLOSETHEMEDATA)(HTHEME hTheme);
```

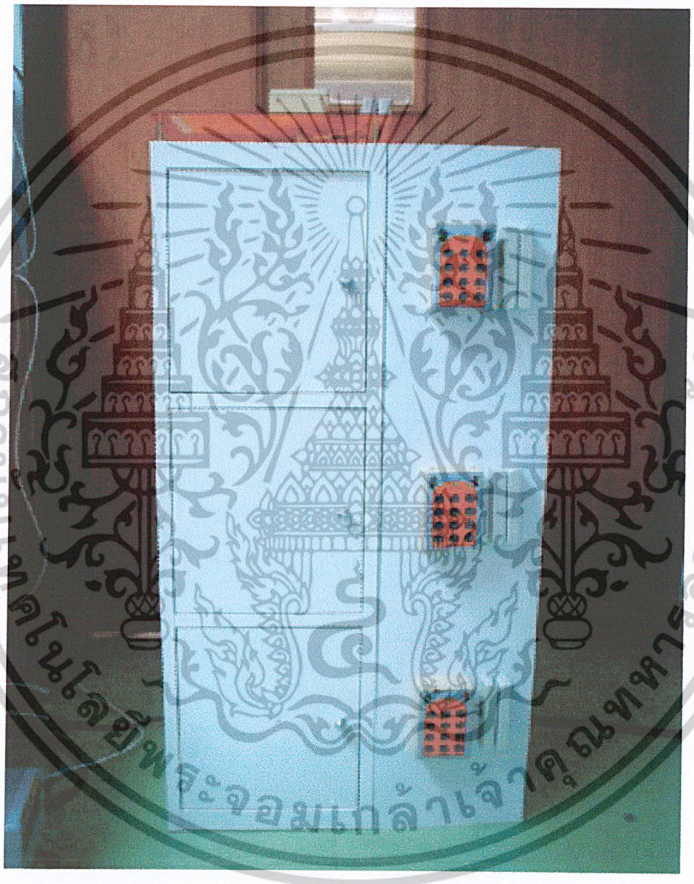
```
      static HRESULT __stdcall CloseThemeDataFail(HTHEME)    {return
E_FAIL;}

      typedef HRESULT(__stdcall *PFNDRAWTHEMEBACKGROUND)(HTHEME
hTheme, HDC hdc, int iPartId, int iStateId, const RECT* pRect,  const
RECT* pClipRect);
      static HRESULT __stdcall DrawThemeBackgroundFail(HTHEME, HDC,
int, int, const RECT*, const RECT*) {return E_FAIL;}

      typedef HRESULT (__stdcall *PFNDRAWTHEMETEXT)(HTHEME hTheme,
HDC hdc, int iPartId, int iStateId, LPCWSTR pszText, int iCharCount,
DWORD dwTextFlags, DWORD dwTextFlags2, const RECT* pRect);
      static HRESULT __stdcall DrawThemeTextFail(HTHEME, HDC, int,
int, LPCWSTR, int, DWORD, DWORD, const RECT*)    {return E_FAIL;}

      typedef BOOL(__stdcall *PFNISAPPTHEMED)();
      static BOOL __stdcall IsAppThemedFail()    {return FALSE;}

      typedef BOOL (__stdcall *PFNISTHEMEACTIVE)();
      static BOOL __stdcall IsThemeActiveFail() {return FALSE;}

      HMODULE           m_hDll;
};

#endif
//
//    Class:            CWinXPButtonST
//
//    Compiler:   Visual C++
//                      eMbedded Visual C++
//    Tested on:  Visual C++ 6.0
//                      Windows CE 3.0
//
//    Created:    03/September/2001
//    Updated:    11/September/2001
//
//    Author:           Davide Calabro'
      davide_calabro@yahoo.com
//
//    Disclaimer
//    ----------
//    THIS SOFTWARE AND THE ACCOMPANYING FILES ARE DISTRIBUTED "AS
IS" AND WITHOUT
//    ANY WARRANTIES WHETHER EXPRESSED OR IMPLIED. NO REPONSIBILITIES
FOR POSSIBLE
//    DAMAGES OR EVEN FUNCTIONALITY CAN BE TAKEN. THE USER MUST
ASSUME THE ENTIRE
//    RISK OF USING THIS SOFTWARE.
//
//    Terms of use
//    ------------
//    THIS SOFTWARE IS FREE FOR PERSONAL USE OR FREEWARE
APPLICATIONS.
//    IF YOU USE THIS SOFTWARE IN COMMERCIAL OR SHAREWARE
APPLICATIONS YOU
//    ARE GENTLY ASKED TO DONATE 1$ (ONE U.S. DOLLAR) TO THE AUTHOR:
//
//          Davide Calabro'
```

```
//          P.O. Box 65
//          21019 Somma Lombardo (VA)
//          Italy
//
#ifndef _WINXPBUTTONST_H_
#define _WINXPBUTTONST_H_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#include "ButtonST.h"

class CWinXPButtonST : public CButtonST
{
public:
      CWinXPButtonST();
      virtual ~CWinXPButtonST();

      DWORD SetRounded(BOOL bRounded, BOOL bRepaint = TRUE);

protected:
      virtual DWORD OnDrawBackground(CDC* pDC, LPCRECT pRect);
      virtual DWORD OnDrawBorder(CDC* pDC, LPCRECT pRect);

private:
      BOOL  m_bIsRounded;           // Borders must be rounded?
};

#endif
//
//    Class:          CXPStyleButtonST
//
//    Compiler:   Visual C++
//    Tested on:  Visual C++ 6.0
//
//    Version:    See GetVersionC() or GetVersionI()
//
//    Created:    21/January/2002
//    Updated:    21/January/2002
//
//    Author:           Davide Calabro'
      davide_calabro@yahoo.com
//
      http://www.softechsoftware.it
//
//    Disclaimer
//    ----------
//    THIS SOFTWARE AND THE ACCOMPANYING FILES ARE DISTRIBUTED "AS
IS" AND WITHOUT
//    ANY WARRANTIES WHETHER EXPRESSED OR IMPLIED. NO REPONSIBILITIES
FOR POSSIBLE
//    DAMAGES OR EVEN FUNCTIONALITY CAN BE TAKEN. THE USER MUST
ASSUME THE ENTIRE
//    RISK OF USING THIS SOFTWARE.
//
//    Terms of use
```

```
//    ------------
//    THIS SOFTWARE IS FREE FOR PERSONAL USE OR FREEWARE
APPLICATIONS.
//    IF YOU USE THIS SOFTWARE IN COMMERCIAL OR SHAREWARE
APPLICATIONS YOU
//    ARE GENTLY ASKED TO DONATE 1$ (ONE U.S. DOLLAR) TO THE AUTHOR:
//
//          Davide Calabro'
//          P.O. Box 65
//          21019 Somma Lombardo (VA)
//          Italy
//
#ifndef _XPSTYLEBTNST_H
#define _XPSTYLEBTNST_H

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "ButtonST.h"
#include "ThemeHelperST.h"

class CXPStyleButtonST  : public CButtonST
{
public:
      CXPStyleButtonST();
      virtual ~CXPStyleButtonST();

      void SetThemeHelper(CThemeHelperST* pTheme);

      static short GetVersionI()      {return 10;}
      static LPCTSTR GetVersionC()  {return (LPCTSTR)_T("1.0");}

protected:
      virtual DWORD OnDrawBackground(CDC* pDC, LPCRECT pRect);
      virtual DWORD OnDrawBorder(CDC* pDC, LPCRECT pRect);

private:
      CThemeHelperST*         m_pTheme;
};

#endif
```

ภาคผนวก จ
คู่มือการใช้งาน

คู่มือการใช้งาน
ระบบเช็คอิน - เช็คเอาต์โรงแรม



ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

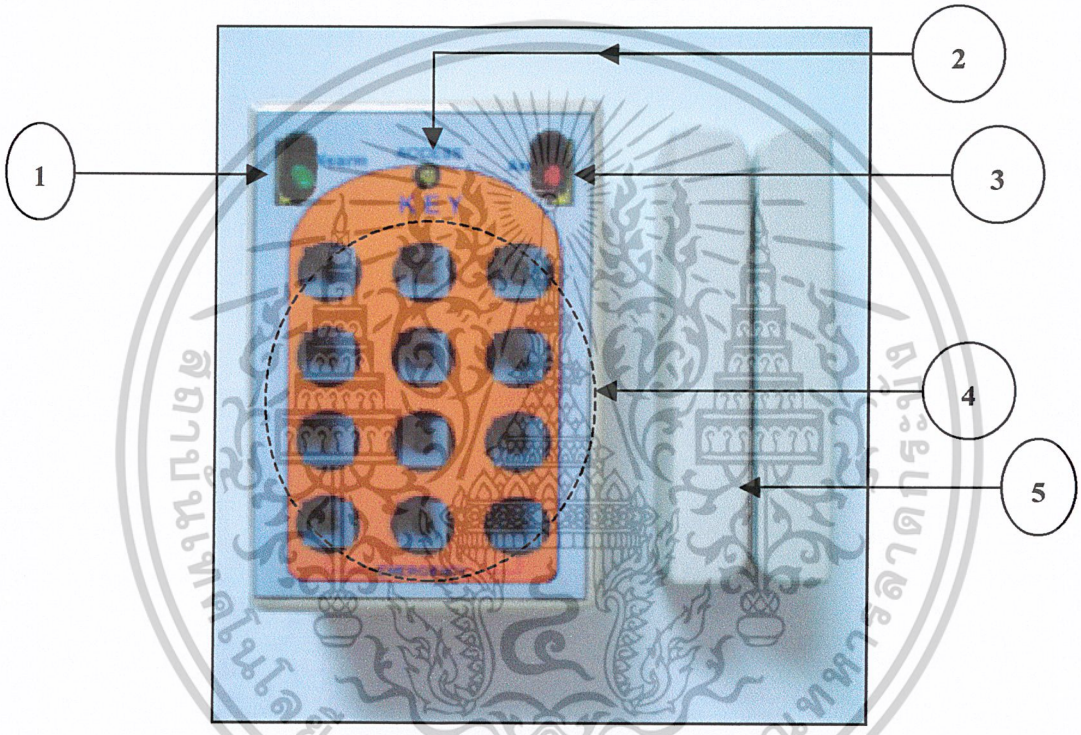
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

1. คำแนะนำเบื้องต้น

ก่อนที่จะลงมือใช้งานระบบเช็คอิน - เช็คเอาต์โรงแรม ควรทำการศึกษาการใช้งานจากคู่มือให้เข้าใจเพื่อการใช้งานที่ถูกต้อง เพื่อเป็นการป้องกันการเสียหายที่อาจเกิดขึ้นกับระบบเช็คอิน - เช็คเอาต์โรงแรม

2. ส่วนประกอบและปุ่มควบคุม



รูปที่ จ.1 ส่วนประกอบและปุ่มควบคุมของระบบเช็คอิน - เช็คเอาต์โรงแรม

จากรูปที่ จ.1 มีรายละเอียดดังต่อไปนี้

หมายเลขที่ 1 ไฟแสดงผลเมื่อมีคนอยู่

หมายเลขที่ 2 ไฟแสดงผลการทำงานของเครื่อง

หมายเลขที่ 3 ไฟแสดงผลเมื่อเกิดเหตุการณ์ไฟไหม้

หมายเลขที่ 4 ปุ่มกดรหัส

หมายเลขที่ 5 เครื่องอ่านบัตรแถบเหล็ก

3. การติดตั้งและใช้งาน

3.1 ต่อบอร์ดคอนโทรลเข้ากับเข้ากับเครื่องคอมพิวเตอร์ก่อนใช้งาน

3.2 จ่ายไฟฟ้ากระแสตรงให้กับบอร์ดไมโครคอนโทรลเลอร์

3.3 เปิดเครื่องคอมพิวเตอร์พร้อมเรียกโปรแกรมการเช็คอิน - เช็คเอาต์โรงแรม

3.4 กรณีที่ต้องการเช็คอินให้เลือกห้องพักที่ไม่มีรูปคนพักอาศัย

3.5 จะปรากฏหน้าต่างโปรแกรมให้ทำการกรอกข้อมูลของผู้เข้าพัก

3.6 เมื่อกรอกข้อมูลต่างๆ ของผู้เข้าพักเสร็จเรียบร้อยแล้ว ทำการกดปุ่ม Check in หากกรอกข้อมูลต่างๆ ไม่ครบจะปรากฏข้อความเตือนว่ายังกรอกข้อมูลไม่ครบถ้วน

3.7 เมื่อกรอกข้อมูลของผู้เข้าพักเสร็จแล้ว ให้รูดบัตรแถบแม่เหล็กเพื่อเก็บไว้เป็นฐานข้อมูล

3.8 เมื่อต้องการที่จะเปิดห้องให้นำบัตรแถบแม่เหล็กไปรูดที่บอร์ดไมโครคอนโทรลเลอร์ในส่วนของห้องพัก ก็จะสามารถเปิดประตูได้

3.9 กรณีที่ต้องการเช็คเอาต์ให้เลือกห้องพักของคนที่ต้องการเช็คเอาต์

3.10 จะปรากฏหน้าต่างแสดงรายละเอียดของบุคคลที่ต้องการจะทำการเช็คเอาต์

3.11 ทำการกดปุ่ม Check out

3.13 กรณีต้องการทำการเปลี่ยนพื้นหลังของโปรแกรมให้กดปุ่ม Skin

3.14 กรณีต้องการออกจากโปรแกรมให้ทำการกดปุ่ม Exit

4. การแก้ปัญหาเบื้องต้น

เมื่อประสบปัญหาในการใช้งานระบบเช็คอิน - เช็คเอาต์โรงแรม สามารถตรวจสอบแนวทางแก้ไขปัญหาเบื้องต้นได้ จากตารางข้างล่างนี้

อาการ	สาเหตุ/หรือวิธีแก้ไข
ระบบไม่ทำงาน, ไฟไม่เข้าบอร์ดไมโครคอลโทรลเลอร์	ตรวจสอบว่าเปิดโปรแกรมอยู่หรือไม่,ตรวจสอบว่าเชื่อมต่อชุดจ่ายไฟหรือไม่
รูดบัตรแถบแม่เหล็กที่เครื่องอ่านแล้วประตูห้องไม่สามารถเปิดได้	ตรวจสอบชุดกลไกระบบเปิดปิดประตูอัตโนมัติ, ดูว่าโปรแกรมเปิดอยู่หรือไม่

5. การดูแลและข้อควรระวัง

5.1 การดูแลรักษา

- เก็บระบบเช็คอิน - เช็คเอาต์โรงแรม และสายในการเชื่อมต่อให้เรียบร้อยหลังการใช้งาน

5.2 ข้อควรระวัง

- อย่าต่อแหล่งจ่ายไฟฟ้าสลับขั้ว
- อย่าให้วงจรหรือสายเกิดการช๊อต

6. ข้อมูลจำเพาะ

คุณสมบัติ	รายละเอียด
ความเร็วสูงสุดในการส่งถ่ายข้อมูลกับพอร์ตอนุกรม	19200 bit/sec
จำนวนพอร์ตเชื่อมต่อ	พอร์ต RS – 232/422 และ พอร์ต RS-485
ส่วนแสดงผล	คอมพิวเตอร์
แหล่งจ่ายพลังงาน	ไฟฟ้ากระแสตรง 5 โวลต์ และ 12 โวลต์

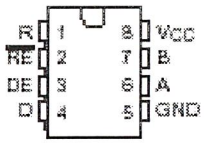
ภาคผนวก ฉ

รายละเอียดและคุณสมบัติของอุปกรณ์

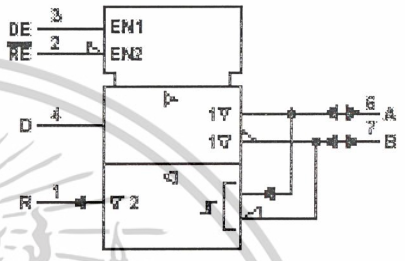
SLLS101A – JULY 1995 – REVISED MAY 1995

- Bidirectional Transceivers
- Meet or Exceed the Requirements of ANSI Standards EIA/TIA-422-B and RS-485 and ITU Recommendations V.11 and X.27
- Designed for Multipoint Transmission on Long Bus Lines in Noisy Environments
- 3-State Driver and Receiver Outputs
- Individual Driver and Receiver Enables
- Wide Positive and Negative Input/Output Bus Voltage Ranges
- Driver Output Capability . . . ±60 mA Max
- Thermal Shutdown Protection
- Driver Positive and Negative Current Limiting
- Receiver Input Impedance . . . 12 kΩ Min
- Receiver Input Sensitivity . . . ±200 mV
- Receiver Input Hysteresis . . . 50 mV Typ
- Operate From Single 5-V Supply

D OR P PACKAGE
(TOP VIEW)

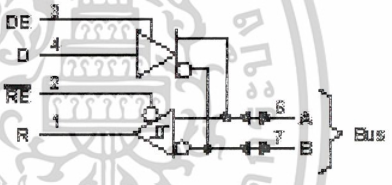


logic symbol†



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

logic diagram (positive logic)



description

The SN65176B and SN75176B differential bus transceivers are monolithic integrated circuits designed for bidirectional data communication on multipoint bus transmission lines. They are designed for balanced transmission lines and meet ANSI Standards EIA/TIA-422-B and RS-485 and ITU Recommendations V.11 and X.27.

The SN65176B and SN75176B combine a 3-state differential line driver and a differential input line receiver, both of which operate from a single 5-V power supply. The driver and receiver have active-high and active-low enables, respectively, that can be externally connected together to function as a direction control. The driver differential outputs and the receiver differential inputs are connected internally to form differential input/output (I/O) bus ports that are designed to offer minimum loading to the bus whenever the driver is disabled or V_{CC} = 0. These ports feature wide positive and negative common-mode voltage ranges making the device suitable for party-line applications.

Function Tables

DRIVER

INPUT D	ENABLE DE	OUTPUTS A	OUTPUTS B
H	H	H	L
L	H	L	H
X	L	Z	Z

RECEIVER

DIFFERENTIAL INPUTS A – B	ENABLE $\overline{RE}$	OUTPUT R
$V_{ID} \geq 0.2$ V	L	H
–0.2 V < V_{ID} < 0.2 V	L	?
$V_{ID} \leq$ –0.2 V	L	L
X	H	Z
Open	L	H

H = high level, L = low level, ? = indeterminate, X = irrelevant, Z = high impedance (off)

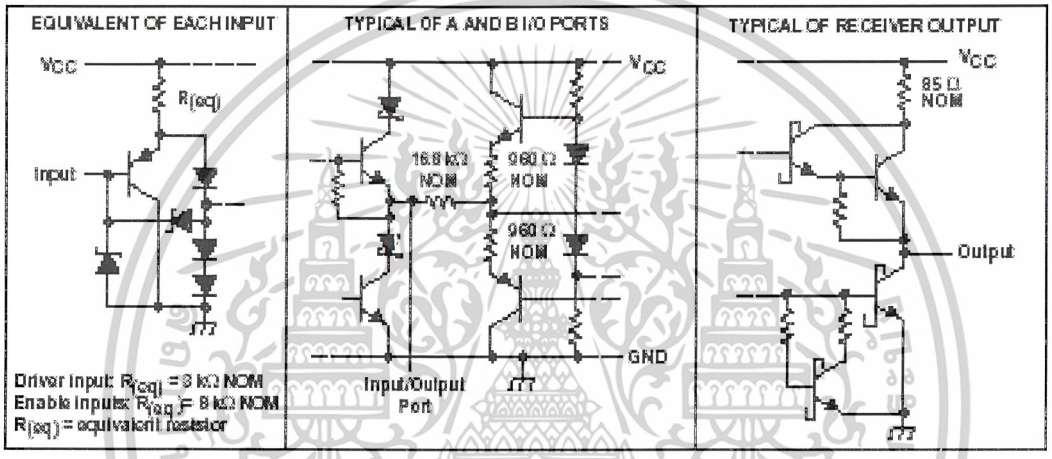
description (continued)

The driver is designed for up to 60 mA of sink or source current. The driver features positive- and negative-current limiting and thermal shutdown for protection from line-fault conditions. Thermal shutdown is designed to occur at a junction temperature of approximately 150°C. The receiver features a minimum input impedance of 12 kΩ, an input sensitivity of ±200 mV, and a typical input hysteresis of 50 mV.

The SN65176B and SN75176B can be used in transmission line applications employing the SN75172 and SN75174 quadruple differential line drivers and SN75173 and SN75175 quadruple differential line receivers.

The SN65176B is characterized for operation from −40°C to 105°C and the SN75176B is characterized for operation from 0°C to 70°C.

schematics of inputs and outputs



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{CC} (see Note 1) ..... 7 V
Voltage range at any bus terminal ..... −10 V to 15 V
Enable input voltage, V_I ..... 5.5 V
Continuous total power dissipation ..... See Dissipation Rating Table
Operating free-air temperature range, T_A: SN65176B ..... −40°C to 105°C
SN75176B ..... 0°C to 70°C
Storage temperature range, T_{stg} ..... −65°C to 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds ..... 260°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage values, except differential input/output bus voltage, are with respect to network ground terminal.

DISSIPATION RATING TABLE

PACKAGE	$T_A \le 25°C$ POWER RATING	DERATING FACTOR ABOVE $T_A = 25°C$	$T_A = 70°C$ POWER RATING	$T_A = 105°C$ POWER RATING
D	725 mW	5.8 mW/°C	464 mW	261 mW
P	1100 mW	8.8 mW/°C	704 mW	396 mW

recommended operating conditions

		MIN	TYP	MAX	UNIT
Supply voltage, V_{CC}		4.75	5	5.25	V
Voltage at any bus terminal (separately or common mode), V_I or V_{IC}				12	V
				−7	
High-level input voltage, V_{IH}	D, DE, and $\overline{RE}$	2			V
Low-level input voltage, V_{IL}	D, DE, and $\overline{RE}$			0.8	V
Differential input voltage, V_{ID} (see Note 2)				±12	V
High-level output current, I_{OH}	Driver			−60	mA
	Receiver			−400	µA
Low-level output current, I_{OL}	Driver			60	mA
	Receiver			8	
Operating free-air temperature, T_A	SN65176B	−40		105	°C
	SN75176B	0		70	

NOTE 2: Differential input/output bus voltage is measured at the noninverting terminal A with respect to the inverting terminal B.

DRIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{IK}	Input clamp voltage	I_I = –18 mA				–1.5	V
V_O	Output voltage	I_O = 0		0		6	V
$\lvert V_{OD1}\rvert$	Differential output voltage	I_O = 0		1.5	3.6	6	V
$\lvert V_{OD2}\rvert$	Differential output voltage	R_L = 100 Ω, See Figure 1		1.02 V_{OD1} or 2¶			V
		R_L = 54 Ω, See Figure 1		1.5	2.5	5	V
V_{OD3}	Differential output voltage	See Note 4		1.5		5	V
$\Delta\lvert V_{OD}\rvert$	Change in magnitude of differential output voltage§	R_L = 54 Ω or 100 Ω, See Figure 1				±0.2	V
V_{OC}	Common-mode output voltage					+3 –1	V
$\Delta\lvert V_{OC}\rvert$	Change in magnitude of common-mode output voltage§					±0.2	V
I_O	Output current	Output disabled, See Note 3	V_O = 12 V			1	mA
			V_O = –7 V			–0.8	
I_{IH}	High-level input current	V_I = 2.4 V				20	µA
I_{IL}	Low-level input current	V_I = 0.4 V				–400	µA
I_{OS}	Short-circuit output current	V_O = –7 V				–250	mA
		V_O = 0				150	
		V_O = V_{CC}				250	
		V_O = 12 V				250	
I_{CC}	Supply current (total package)	No load	Outputs enabled		42	70	mA
			Outputs disabled		26	35	

† The power-off measurement in ANSI Standard EIA/TIA-422-B applies to disabled outputs only and is not applied to combined inputs and outputs.
‡ All typical values are at V_{CC} = 5 V and T_A = 25°C.
§ $\Delta\lvert V_{OD}\rvert$ and $\Delta\lvert V_{OC}\rvert$ are the changes in magnitude of V_{OD} and V_{OC}, respectively, that occur when the input is changed from a high level to a low level.
¶ The minimum V_{OD2} with a 100-Ω load is either 1.02 V_{OD1} or 2 V, whichever is greater.
NOTES: 3. See ANSI Standard RS-485 Figure 3.5, Test Termination Measurement 2.
4. This applies for both power on and off; refer to ANSI Standard RS-485 for exact conditions. The EIA/TIA-422-B limit does not apply for a combined driver and receiver terminal.

switching characteristics, V_{CC} = 5 V, R_L = 110 Ω, T_A = 25°C (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{d(OD)}$	Differential-output delay time	R_L = 54 Ω, See Figure 3		15	23	ns
$t_{t(OD)}$	Differential-output transition time			20	30	ns
t_{PZH}	Output enable time to high level	See Figure 4		85	120	ns
t_{PZL}	Output enable time to low level	See Figure 5		40	60	ns
t_{PHZ}	Output disable time from high level	See Figure 4		150	250	ns
t_{PLZ}	Output disable time from low level	See Figure 5		20	30	ns

SYMBOL EQUIVALENTS

DATA SHEET PARAMETER	EIA/TIA-422-B	RS-485
V_O	V_{oa}, V_{ob}	V_{oa}, V_{ob}
$\|V_{OD1}\|$	V_O	V_O
$\|V_{OD2}\|$	V_t (R_L = 100 Ω)	V_t (R_L = 54 Ω)
$\|V_{OD3}\|$		V_t (Test Termination Measurement 2)
$\Delta\|V_{OD}\|$	$\|\|V_t\| - \|\overline{V_t}\|\|$	$\|\|V_t\| - \|\overline{V_t}\|\|$
V_{OC}	$\|V_{os}\|$	$\|V_{os}\|$
$\Delta\|V_{OC}\|$	$\|V_{os} - \overline{V_{os}}\|$	$\|V_{os} - \overline{V_{os}}\|$
I_{OS}	$\|I_{sa}\|$, $\|I_{sb}\|$	
I_O	$\|I_{oa}\|$, $\|I_{ob}\|$	I_{oa}, I_{ob}

RECEIVER SECTION

electrical characteristics over recommended ranges of common-mode input voltage, supply voltage, and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{IT+}	Positive-going input threshold voltage	V_O = 2.7 V, I_O = −0.4 mA				0.2	V
V_{IT-}	Negative-going input threshold voltage	V_O = 0.5 V, I_O = 8 mA		−0.2‡			V
V_{hys}	Input hysteresis voltage ($V_{IT+} - V_{IT-}$)				50		mV
V_{IK}	Enable input clamp voltage	I_I = −18 mA				−1.5	V
V_{OH}	High-level output voltage	V_{ID} = 200 mV, See Figure 2	I_{OH} = −400 µA,	2.7			V
V_{OL}	Low-level output voltage	V_{ID} = −200 mV, See Figure 2	I_{OL} = 8 mA,			0.45	V
I_{OZ}	High-impedance-state output current	V_O = 0.4 V to 2.4 V				±20	µA
I_I	Line input current	Other input = 0 V, See Note 5	V_I = 12 V			1	mA
			V_I = −7 V			−0.8	
I_{IH}	High-level enable input current	V_{IH} = 2.7 V				20	µA
I_{IL}	Low-level enable input current	V_{IL} = 0.4 V				−100	µA
r_I	Input resistance	V_I = 12 V		12			kΩ
I_{OS}	Short-circuit output current			−15		−85	mA
I_{CC}	Supply current (total package)	No load	Outputs enabled		42	55	mA
			Outputs disabled		28	39	

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the less-positive (more-negative) limit is designated minimum, is used in this data sheet for common-mode input voltage and threshold voltage levels only.
NOTE 5: This applies for both power on and power off. Refer to EIA Standard RS-485 for exact conditions.

switching characteristics, V_{CC} = 5 V, C_L = 15 pF, T_A = 25°C

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low- to high-level output	V_{ID} = 0 to 3 V, See Figure 6		21	35	ns
t_{PHL}	Propagation delay time, high- to low-level output			23	35	ns
t_{PZH}	Output enable time to high level	See Figure 7		10	20	ns
t_{PZL}	Output enable time to low level			12	20	ns
t_{PHZ}	Output disable time from high level	See Figure 7		20	35	ns
t_{PLZ}	Output disable time from low level			17	25	ns

PARAMETER MEASUREMENT INFORMATION

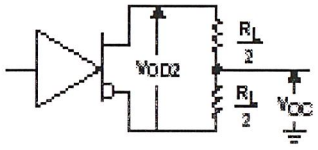


Figure 1. Driver V_{OD} and V_{CC}

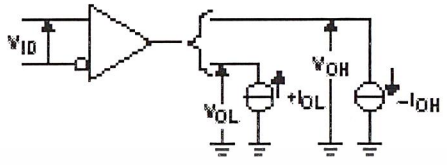


Figure 2. Receiver V_{OH} and V_{OL}

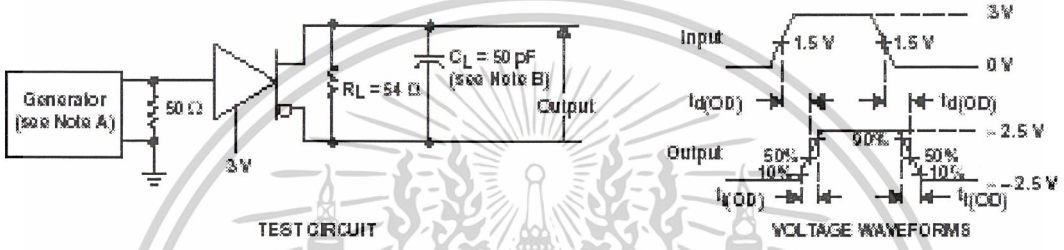


Figure 3. Driver Test Circuit and Voltage Waveforms

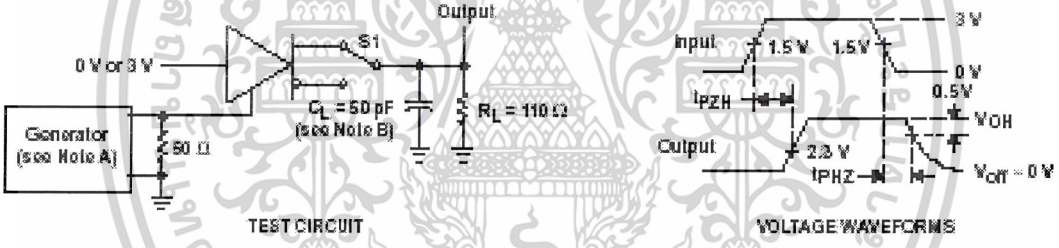


Figure 4. Driver Test Circuit and Voltage Waveforms

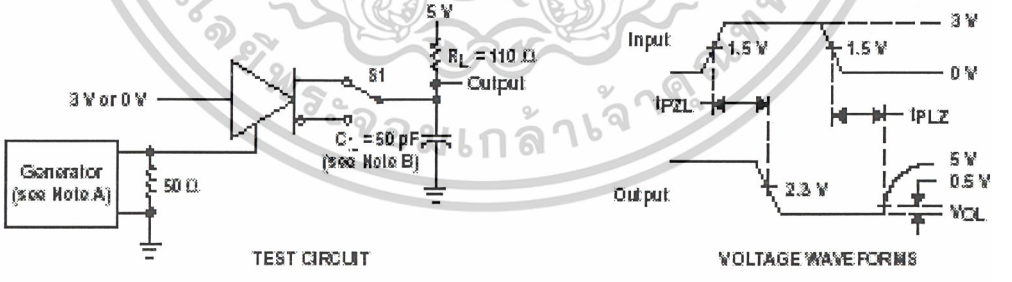


Figure 5. Driver Test Circuit and Voltage Waveforms

NOTES: A. The input pulse is supplied by a generator having the following characteristics: PRR = 1 MHz, 50% duty cycle, t_r = 6 ns, t_f = 6 ns, Z_O = 50 Ω.
B. C_L includes probe and jig capacitance.

PARAMETER MEASUREMENT INFORMATION

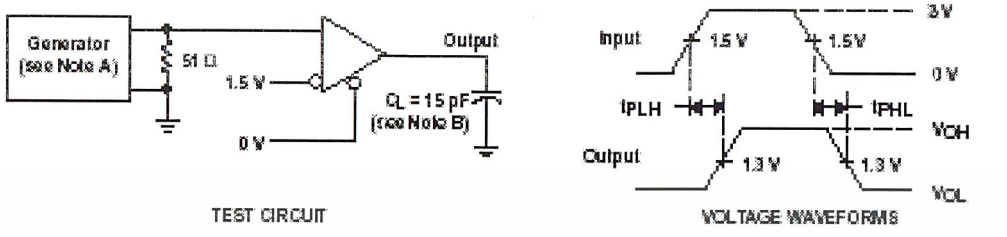


Figure 6. Receiver Test Circuit and Voltage Waveforms

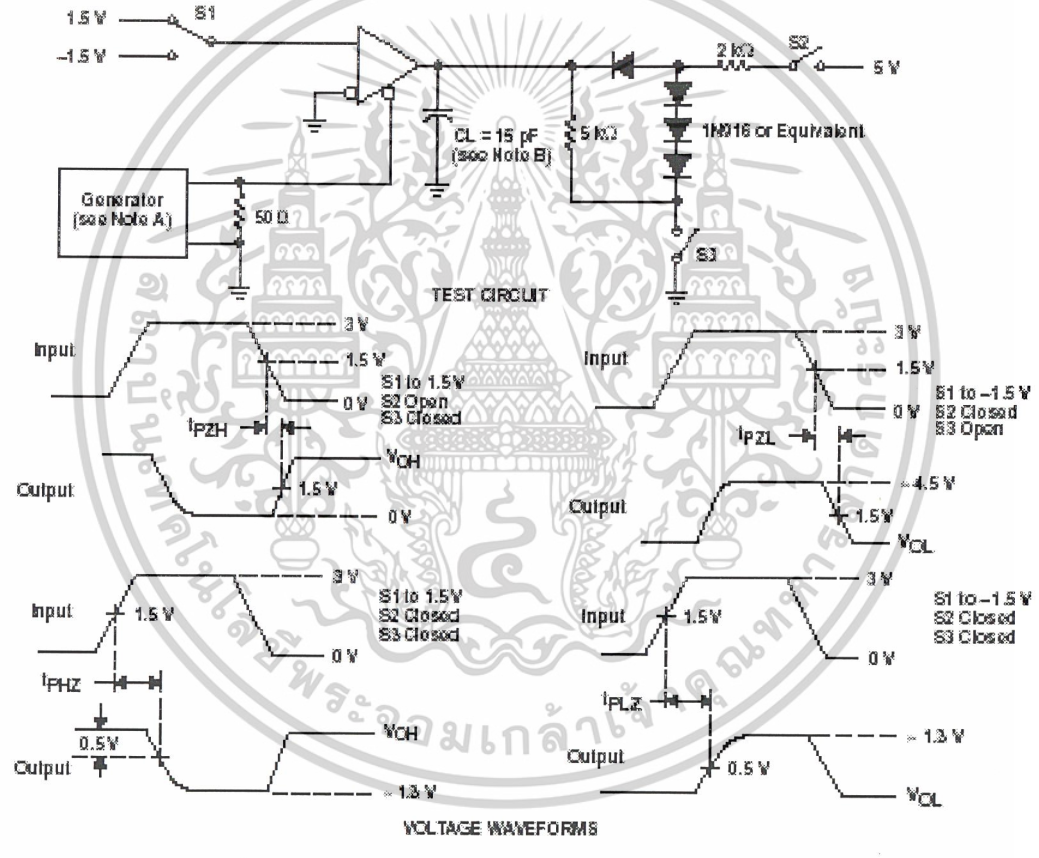


Figure 7. Receiver Test Circuit and Voltage Waveforms

NOTES: A. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 1 MHz, 50% duty cycle, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_O ≈ 50 Ω.
B. C_L includes probe and jig capacitance.

TYPICAL CHARACTERISTICS

DRIVER
HIGH-LEVEL OUTPUT VOLTAGE
vs
HIGH-LEVEL OUTPUT CURRENT

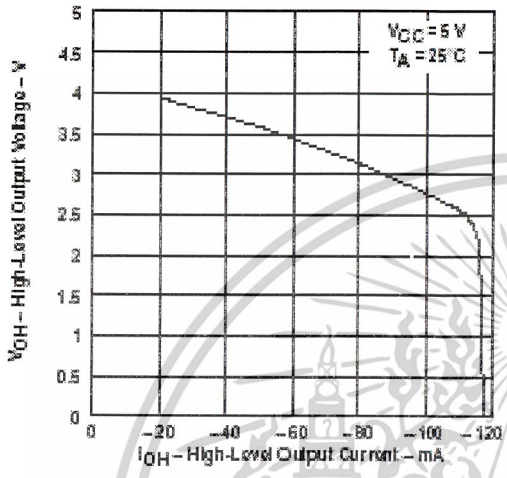


Figure 8

DRIVER
LOW-LEVEL OUTPUT VOLTAGE
vs
LOW-LEVEL OUTPUT CURRENT

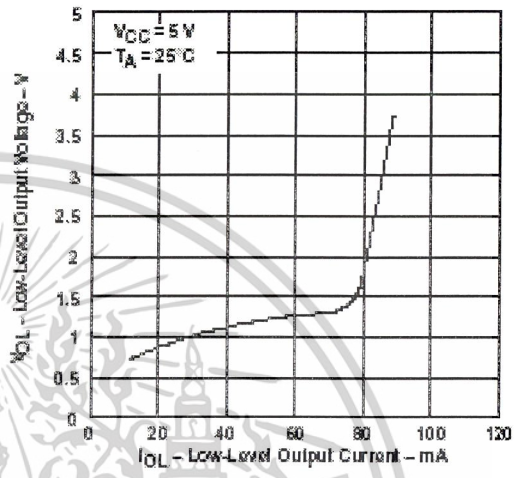


Figure 9

DRIVER
DIFFERENTIAL OUTPUT VOLTAGE
vs
OUTPUT CURRENT

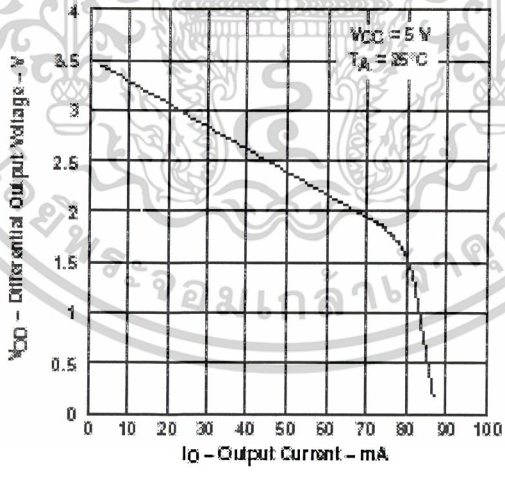


Figure 10

TYPICAL CHARACTERISTICS

RECEIVER
HIGH-LEVEL OUTPUT VOLTAGE
vs
HIGH-LEVEL OUTPUT CURRENT

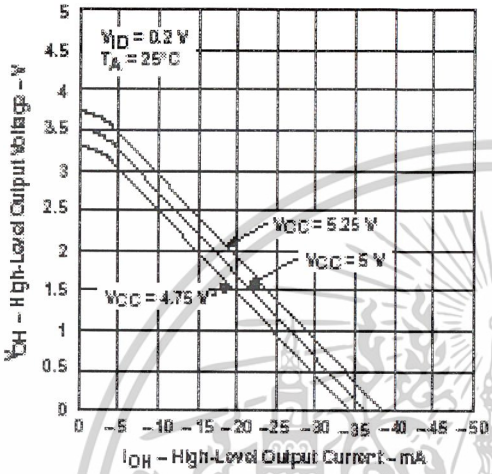
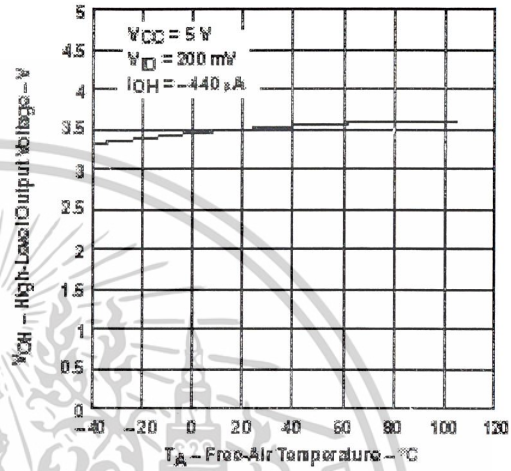


Figure 11

RECEIVER
HIGH-LEVEL OUTPUT VOLTAGE
vs
FREE-AIR TEMPERATURE†



† Only the 0°C to 70°C portion of the curve applies to the SN75176B.

Figure 12

RECEIVER
LOW-LEVEL OUTPUT VOLTAGE
vs
LOW-LEVEL OUTPUT CURRENT

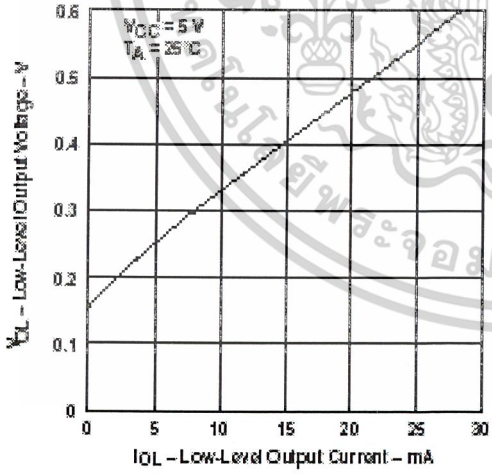


Figure 13

RECEIVER
LOW-LEVEL OUTPUT VOLTAGE
vs
FREE-AIR TEMPERATURE

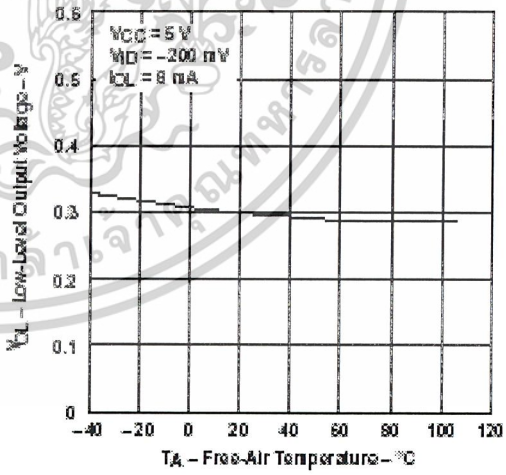


Figure 14

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

TYPICAL CHARACTERISTICS

RECEIVER
OUTPUT VOLTAGE
vs
ENABLE VOLTAGE

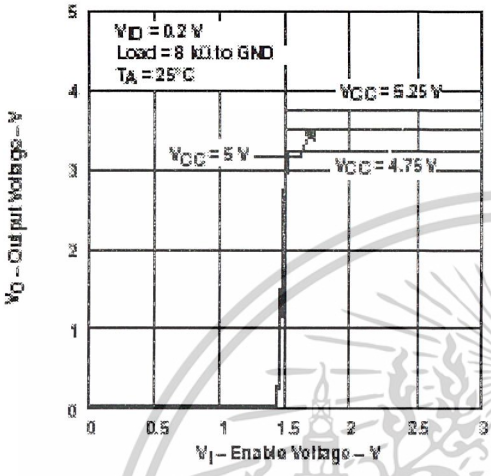


Figure 15

RECEIVER
OUTPUT VOLTAGE
vs
ENABLE VOLTAGE

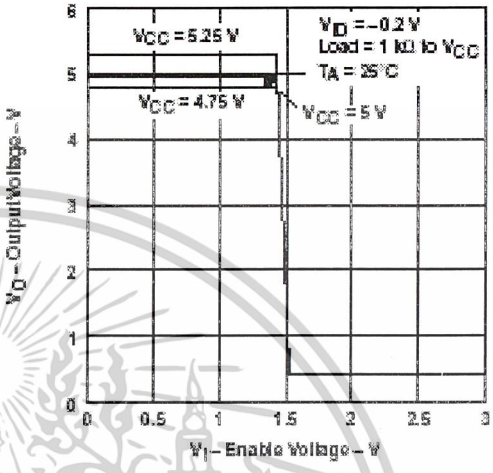


Figure 16

APPLICATION INFORMATION

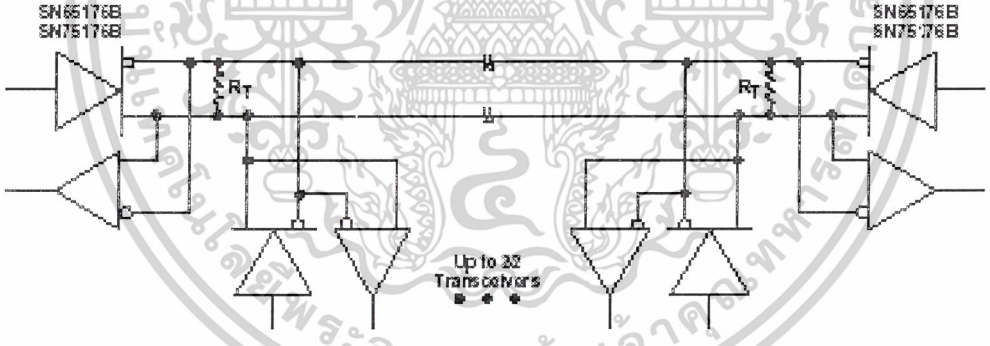


Figure 17. Typical Application Circuit

NOTE: The line should be terminated at both ends with its characteristic impedance ($R_T = Z_O$). Stub lengths off the main line should be kept as short as possible.

INTEGRATED CIRCUITS

DATA SHEET

P89C51RB2/P89C51RC2/P89C51RD2

80C51 8-bit Flash microcontroller family

16KB/32KB/64KB ISP/IAP Flash with 512B/512B/1KB RAM

Preliminary specification

1999 Nov 22

IC28 Data Handbook

Philips Semiconductors

PHILIPS

DESCRIPTION

The P89C51RB2/RC2/RD2 device contains a non-volatile 16kB/32kB/64kB Flash program memory that is both parallel programmable and serial In-System and In-Application Programmable. In-System Programming (ISP) allows the user to download new code while the microcontroller sits in the application. In-Application Programming (IAP) means that the microcontroller fetches new program code and reprograms itself while in the system. This allows for remote programming over a modem link. A default serial loader (boot loader) program in ROM allows serial In-System programming of the Flash memory via the UART without the need for a loader in the Flash code. For In-Application Programming, the user program erases and reprograms the Flash memory by use of standard routines contained in ROM.

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

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

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

The added features of the P89C51RB2/RC2/RD2 makes it a powerful microcontroller for applications that require pulse width modulation, high-speed I/O and up/down counting capabilities such as motor control.

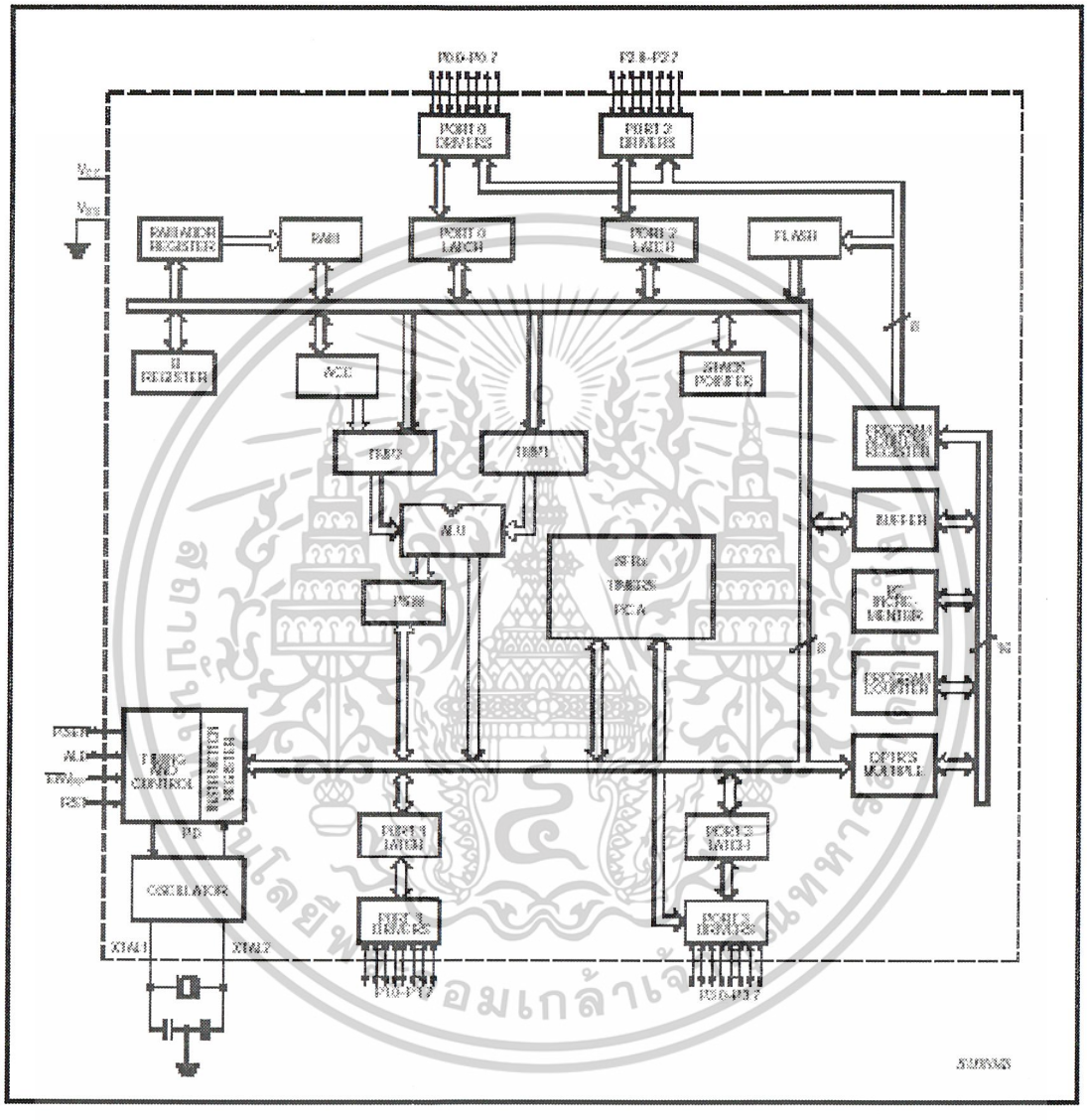
FEATURES

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

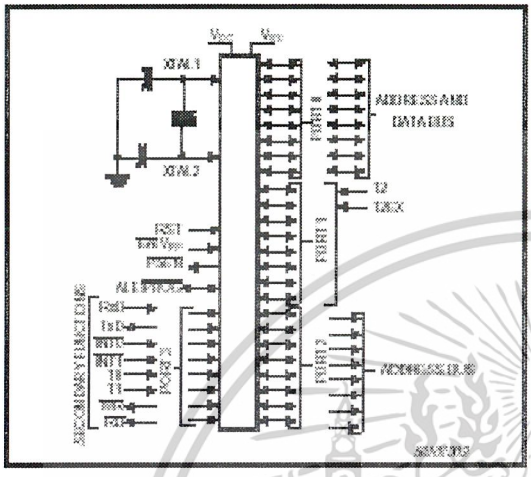
ORDERING INFORMATION

	PHILIPS (EXCEPT NORTH AMERICA) PART ORDER NUMBER PART MARKING	PHILIPS NORTH AMERICA PART ORDER NUMBER	MEMORY		TEMPERATURE RANGE (°C) AND PACKAGE	VOLTAGE RANGE	FREQUENCY (MHz)		DWG #
			FLASH	RAM			6 CLOCK MODE	12 CLOCK MODE	
1	P89C51RB2HBP	P89C51RB2BP	16 kB	512 B	0 to +70, PDIP	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT129-1
2	P89C51RB2HFP	P89C51RB2FP	16 kB	512 B	–40 to +85, PDIP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT129-1
3	P89C51RB2HBA	P89C51RB2BA	16 kB	512 B	0 to +70, PLCC	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT187-2
4	P89C51RB2HFA	P89C51RB2FA	16 kB	512 B	–40 to +85, PLCC	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
5	P89C51RB2HBB	P89C51RB2BB	16 kB	512 B	0 to +70, PQFP	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT307-2
6	P89C51RB2HFB	P89C51RB2FB	16 kB	512 B	–40 to +85, PQFP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT307-2
7	P89C51RC2HBP	P89C51RC2BP	32 kB	512 B	0 to +70, PDIP	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT129-1
8	P89C51RC2HFP	P89C51RC2FP	32 kB	512 B	–40 to +85, PDIP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT129-1
9	P89C51RC2HBA	P89C51RC2BA	32 kB	512 B	0 to +70, PLCC	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT187-2
10	P89C51RC2HFA	P89C51RC2FA	32 kB	512 B	–40 to +85, PLCC	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
11	P89C51RC2HBB	P89C51RC2BB	32 kB	512 B	0 to +70, PQFP	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT307-2
12	P89C51RC2HFB	P89C51RC2FB	32 kB	512 B	–40 to +85, PQFP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT307-2
13	P89C51RD2HBP	P89C51RD2BP	64 kB	1 kB	0 to +70, PDIP	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT129-1
14	P89C51RD2HFP	P89C51RD2FP	64 kB	1 kB	–40 to +85, PDIP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT129-1
15	P89C51RD2HBA	P89C51RD2BA	64 kB	1 kB	0 to +70, PLCC	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT187-2
16	P89C51RD2HFA	P89C51RD2FA	64 kB	1 kB	–40 to +85, PLCC	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT187-2
17	P89C51RD2HBB	P89C51RD2BB	64 kB	1 kB	0 to +70, PQFP	4.5–5.5 V	0 to 30 MHz	0 to 33 MHz	SOT307-2
18	P89C51RD2HFB	P89C51RD2FB	64 kB	1 kB	–40 to +85, PQFP	4.5–5.5 V	0 to 20 MHz	0 to 33 MHz	SOT307-2

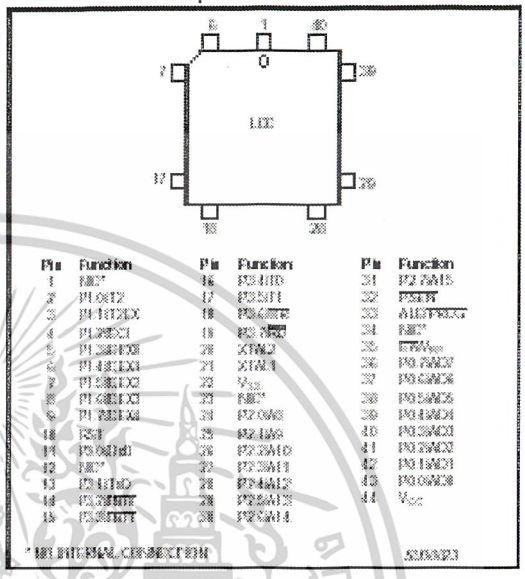
BLOCK DIAGRAM



LOGIC SYMBOL



Plastic Leaded Chip Carrier



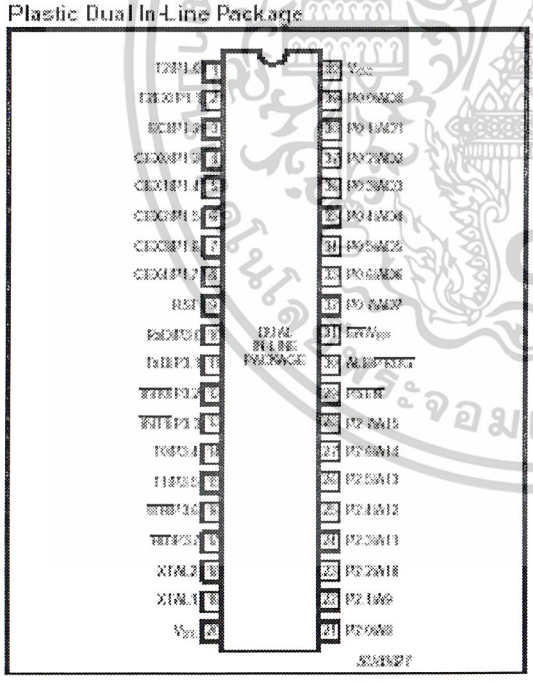
Pin	Function	Pin	Function	Pin	Function
1	NIC*	16	P3.4/T0	31	P2.2/A10
2	P1.0/T2	17	P3.5/T1	32	P2.3/A11
3	P1.1/T2EX	18	P3.6/WR	33	A10/P2.4/A12
4	P1.2/ECI	19	P3.7/RD	34	NIC*
5	P1.3/CEX0	20	XTAL2	35	EW/V_{PP}
6	P1.4/CEX1	21	XTAL1	36	P0.7/AD7
7	P1.5/CEX2	22	V_{SS}	37	P0.6/AD6
8	P1.6/CEX3	23	NIC*	38	P0.5/AD5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

* NO INTERNAL CONNECTION

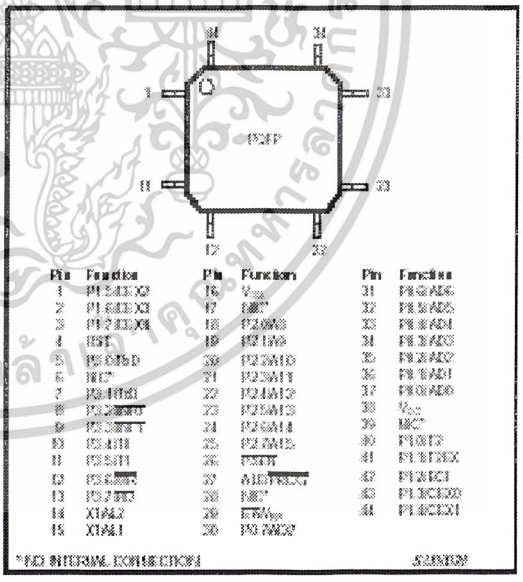
SU00423

PINNING

Plastic Dual In-Line Package



Plastic Quad Flat Pack



Pin	Function	Pin	Function	Pin	Function
1	P1.5/CEX3/CI	16	V_{SS}	31	P0.0/AD0
2	P1.6/CEX4/CI	17	NIC*	32	P0.1/AD1
3	P1.7/CEX5/CI	18	P2.0/A8	33	P0.2/AD2
4	RST	19	P2.1/A9	34	P0.3/AD3
5	P3.0/RxD	20	P2.2/A10	35	P0.4/AD4
6	NIC*	21	P2.3/A11	36	P0.5/AD5
7	P3.1/TxD	22	P2.4/A12	37	P0.6/AD6
8	P3.2/INT0	23	P2.5/A13	38	P0.7/AD7
9	P3.3/INT1	24	P2.6/A14	39	NIC*
10	P3.4/T0	25	P2.7/A15	40	P1.0/T2
11	P3.5/T1	26	PSEN	41	P1.1/T2EX
12	P3.6/WR	27	A16/P2.8/A16	42	P1.2/ECI
13	P3.7/RD	28	NIC*	43	P1.3/CEX0
14	XTAL2	29	EW/V_{PP}	44	P1.4/CEX1
15	XTAL1	30	P0.7/AD7		

* NO INTERNAL CONNECTION

SU00424

PIN DESCRIPTIONS

MNEMONIC	PIN NUMBER			TYPE	NAME AND FUNCTION
	PDIP	PLCC	PQFP		
V_{SS}	20	22	16	I	**Ground:** 0 V reference.
V_{CC}	40	44	38	I	**Power Supply:** This is the power supply voltage for normal, idle, and power-down operation.
P0.0–0.7	39–32	43–36	37–30	I/O	**Port 0:** Port 0 is an open-drain, bidirectional I/O port. Port 0 pins that have 1s written to them float and can be used as high-impedance inputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external program and data memory. In this application, it uses strong internal pull-ups when emitting 1s.
P1.0–P1.7	1–8	2–9	40–44, 1–3	I/O	**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pull-ups on all pins except P1.6 and P1.7 which are open-drain. Port 1 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 1 pins that are externally pulled low will source current because of the internal pull-ups. (See DC Electrical Characteristics: I_{IL}). Alternate functions for 80C51RB2/RC2/RD2 Port 1 include:
	1	2	40	I/O	**T2 (P1.0):** Timer/Counter 2 external count input/Clockout (see Programmable Clock-Out)
	2	3	41	I	**T2EX (P1.1):** Timer/Counter 2 Reload/Capture/Direction Control
	3	4	42	I	**ECI (P1.2):** External Clock Input to the PCA
	4	5	43	I/O	**CEX0 (P1.3):** Capture/Compare External I/O for PCA module 0
	5	6	44	I/O	**CEX1 (P1.4):** Capture/Compare External I/O for PCA module 1
	6	7	1	I/O	**CEX2 (P1.5):** Capture/Compare External I/O for PCA module 2
	7	8	2	I/O	**CEX3 (P1.6):** Capture/Compare External I/O for PCA module 3
	8	9	3	I/O	**CEX4 (P1.7):** Capture/Compare External I/O for PCA module 4
P2.0–P2.7	21–28	24–31	18–25	I/O	**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. Port 2 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 2 pins that are externally being pulled low will source current because of the internal pull-ups. (See DC Electrical Characteristics: I_{IL}). Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @DPTR). In this application, it uses strong internal pull-ups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @Ri), port 2 emits the contents of the P2 special function register.
P3.0–P3.7	10–17	11, 13–19	5, 7–13	I/O	**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pull-ups. Port 3 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 3 pins that are externally being pulled low will source current because of the pull-ups. (See DC Electrical Characteristics: I_{IL}). Port 3 also serves the special features of the 80C51RB2/RC2/RD2, as listed below:
	10	11	5	I	**RxD (P3.0):** Serial input port
	11	13	7	O	**TxD (P3.1):** Serial output port
	12	14	8	I	**INT0 (P3.2):** External interrupt
	13	15	9	I	**INT1 (P3.3):** External interrupt
	14	16	10	I	**T0 (P3.4):** Timer 0 external input
	15	17	11	I	**T1 (P3.5):** Timer 1 external input
	16	18	12	O	**WR (P3.6):** External data memory write strobe
	17	19	13	O	**RD (P3.7):** External data memory read strobe
RST	9	10	4	I	**Reset:** A high on this pin for two machine cycles while the oscillator is running resets the device. An internal resistor to V_{SS} permits a power-on reset using only an external capacitor to V_{CC}.
ALE	30	33	27	O	**Address Latch Enable:** Output pulse for latching the low byte of the address during an access to external memory. In normal operation, ALE is emitted twice every machine cycle, and can be used for external timing or clocking. Note that one ALE pulse is skipped during each access to external data memory. ALE can be disabled by setting SFR auxiliary.0. With this bit set, ALE will be active only during a MOVX instruction.

MNEMONIC	PIN NUMBER			TYPE	NAME AND FUNCTION
	PDIP	PLCC	PQFP		
$\overline{PSEN}$	29	32	26	O	**Program Store Enable:** The read strobe to external program memory. When executing code from the external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory. $\overline{PSEN}$ is not activated during fetches from internal program memory.
$\overline{EA}$/V_{PP}	31	35	29	I	**External Access Enable/Programming Supply Voltage:** $\overline{EA}$ must be externally held low to enable the device to fetch code from external program memory locations. If $\overline{EA}$ is held high, the device executes from internal program memory. The value on the $\overline{EA}$ pin is latched when RST is released and any subsequent changes have no effect. This pin also receives the programming supply voltage (V_{PP}) during Flash programming.
XTAL1	19	21	16	I	**Crystal 1:** Input to the inverting oscillator amplifier and input to the internal clock generator circuits.
XTAL2	18	20	14	O	**Crystal 2:** Output from the inverting oscillator amplifier.

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

Table 1. Special Function Registers

SYMBOL	DESCRIPTION	DIRECT ADDRESS	BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION MSB							LSB	RESET VALUE
ACC*	Accumulator	E0H	E7	E6	E5	E4	E3	E2	E1	E0	00H
AUXR#	Auxiliary	8EH	–	–	–	–	–	–	EXTRAM	AO	xxxx xx10B
AUXR1#	Auxiliary 1	A2H	–	–	DPHOV	–	GF2	0	–	DPS	xxxx xx0B
B*	B register	F0H	F7	F6	F5	F4	F3	F2	F1	F0	00H
CCAP0H#	Module 0 Capture High	FAH									xxxx xxxxB
CCAP1H#	Module 1 Capture High	FBH									xxxx xxxxB
CCAP2H#	Module 2 Capture High	FCH									xxxx xxxxB
CCAP3H#	Module 3 Capture High	FDH									xxxx xxxxB
CCAP4H#	Module 4 Capture High	FEH									xxxx xxxxB
CCAP0L#	Module 0 Capture Low	EAH									xxxx xxxxB
CCAP1L#	Module 1 Capture Low	EBH									xxxx xxxxB
CCAP2L#	Module 2 Capture Low	ECH									xxxx xxxxB
CCAP3L#	Module 3 Capture Low	EDH									xxxx xxxxB
CCAP4L#	Module 4 Capture Low	EEH									xxxx xxxxB
CCAPM0#	Module 0 Mode	DAH	–	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	x0x0 0000B
CCAPM1#	Module 1 Mode	DBH	–	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	x0x0 0000B
CCAPM2#	Module 2 Mode	DCH	–	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	x0x0 0000B
CCAPM3#	Module 3 Mode	DDH	–	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	x0x0 0000B
CCAPM4#	Module 4 Mode	DEH	–	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	x0x0 0000B
			DF	DE	DD	DC	DB	DA	D9	D8	
CCON*	PCA Counter Control	D8H	CF	CR	–	CCF4	CCF3	CCF2	CCF1	CCF0	00x0 0000B
CH#	PCA Counter High	F9H									00H
CL#	PCA Counter Low	E9H									00H
CMOD#	PCA Counter Mode	D9H	CIDL	WDTE	–	–	–	CPS1	CPS0	ECF	00xx x000B
DPTR:	Data Pointer (2 bytes)										
DPH	Data Pointer High	83H									00H
DPL	Data Pointer Low	82H									00H
			AF	AE	AD	AC	AB	AA	A9	A8	
IE*	Interrupt Enable 0	A8H	EA	EC	ET2	ES	ET1	EX1	ET0	EX0	00H
			BF	BE	BD	BC	BB	BA	B9	B8	
IP*	Interrupt Priority	B8H	–	PPC	PT2	PS	PT1	PX1	PT0	PX0	x0x0 0000B
			B7	B6	B5	B4	B3	B2	B1	B0	
IPH#	Interrupt Priority High	B7H	–	PPCH	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	x0x0 0000B
			87	86	85	84	83	82	81	80	
P0*	Port 0	80H	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	FFH
			97	96	95	94	93	92	91	90	
P1*	Port 1	90H	CEX4	CEX3	CEX2	CEX1	CEX0	ECI	T2EX	T2	FFH
			A7	A6	A5	A4	A3	A2	A1	A0	
P2*	Port 2	A0H	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	FFH
			B7	B6	B5	B4	B3	B2	B1	B0	
P3*	Port 3	B0H	RD	WR	T1	T0	INT1	INT0	TXD	RXD	FFH
PCON#1	Power Control	87H	SMOD1	SMOD0	–	–	GF1	GF0	PD	IDL	00xx 0000B

* SFRs are bit addressable.
\# SFRs are modified from or added to the 80C51 SFRs.
– Reserved bits.
1. Reset value depends on reset source.

Table 1. Special Function Registers (Continued)

SYMBOL	DESCRIPTION	DIRECT ADDRESS	BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION MSB LSB								RESET VALUE
			D7	D6	D5	D4	D3	D2	D1	D0	
PSW*	Program Status Word	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	00000000B
RCAP2H#	Timer 2 Capture High	CBH									00H
RCAP2L#	Timer 2 Capture Low	CAH									00H
SADDR#	Slave Address	A9H									00H
SADEN#	Slave Address Mask	B9H									00H
SBUF	Serial Data Buffer	99H									xxxxxxxxB
			9F	9E	9D	9C	9B	9A	99	98	
SCON*	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	00H
SP	Stack Pointer	81H									07H
			8F	8E	8D	8C	8B	8A	89	88	
TCON*	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00H
			CF	CE	CD	CC	CB	CA	C9	C8	
T2CON*	Timer 2 Control	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2	00H
T2MOD#	Timer 2 Mode Control	C9H	–	–	–	–	–	–	T2OE	DCEN	xxxxxx00B
TH0	Timer High 0	8CH									00H
TH1	Timer High 1	8DH									00H
TH2#	Timer High 2	CDH									00H
TL0	Timer Low 0	8AH									00H
TL1	Timer Low 1	8BH									00H
TL2#	Timer Low 2	CCH									00H
TMOD	Timer Mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00H
WDTRST	Watchdog Timer Reset	A6H									

* SFRs are bit addressable.
\# SFRs are modified from or added to the 80C51 SFRs.
– Reserved bits.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier. The pins can be configured for use as an on-chip oscillator.

To drive the device from an external clock source, XTAL1 should be driven while XTAL2 is left unconnected. Minimum and maximum high and low times specified in the data sheet must be observed.

This device is configured at the factory to operate using 6 clock periods per machine cycle, referred to in this datasheet as "6 clock mode". (This yields performance equivalent to twice that of standard 80C51 family devices). It may be optionally configured on commercially-available EPROM programming equipment to operate at 12 clocks per machine cycle, referred to in this datasheet as "12 clock mode". Once 12 clock mode has been configured, it cannot be changed back to 6 clock mode.

RESET

A reset is accomplished by holding the RST pin high for at least two machine cycles (12 oscillator periods in 6 clock mode, or 24 oscillator periods in 12 clock mode), while the oscillator is running. To ensure a good power-on reset, the RST pin must be high long enough to allow the oscillator time to start up (normally a few milliseconds) plus two machine cycles. At power-on, the voltage on V_{CC} and RST must come up at the same time for a proper start-up. Ports 1, 2, and 3 will asynchronously be driven to their reset condition when a voltage above V_{IH1} (min) is applied to RESET.

The value on the $\overline{EA}$ pin is latched when RST is deasserted and has no further effect.

LOW POWER MODES

Stop Clock Mode

The static design enables the clock speed to be reduced down to 0 MHz (stopped). When the oscillator is stopped, the RAM and Special Function Registers retain their values. This mode allows step-by-step utilization and permits reduced system power consumption by lowering the clock frequency down to any value. For lowest power consumption the Power Down mode is suggested.

Idle Mode

In the Idle mode (see Table 2), the CPU puts itself to sleep while all of the on-chip peripherals stay active. The instruction to invoke the Idle mode is the last instruction executed in the normal operating mode before the Idle mode is activated. The CPU contents, the on-chip RAM, and all of the special function registers remain intact during this mode. The Idle mode can be terminated either by any enabled interrupt (at which time the process is picked up at the interrupt service routine and continued), or by a hardware reset which starts the processor in the same manner as a power-on reset.

Power-Down Mode

To save even more power, a Power Down mode (see Table 2) can be invoked by software. In this mode, the oscillator is stopped and the instruction that invoked Power Down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values down to 2.0 V and care must be taken to return V_{CC} to the minimum specified operating voltage before the Power Down Mode is terminated.

Either a hardware reset or external interrupt can be used to exit from Power Down. Reset redefines all the SFRs but does not change the on-chip RAM. An external interrupt allows both the SFRs and the on-chip RAM to retain their values.

To properly terminate Power Down, the reset or external interrupt should not be executed before V_{CC} is restored to its normal operating level and must be held active long enough for the oscillator to restart and stabilize (normally less than 10 ms).

With an external interrupt, INT0 and INT1 must be enabled and configured as level-sensitive. Holding the pin low restarts the oscillator but bringing the pin back high completes the exit. Once the interrupt is serviced, the next instruction to be executed after RETI will be the one following the instruction that put the device into Power Down.

Design Consideration

- When the Idle mode is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

POWER OFF FLAG

The Power Off Flag (POF) is set by on-chip circuitry when the V_{CC} level on the P89C51RB2/RC2/RD2 rises from 0 to 5 V. The POF bit can be set or cleared by software allowing a user to determine if the reset is the result of a power-on or a warm start after power-down. The V_{CC} level must remain above 3 V for the POF to remain unaffected by the V_{CC} level.

ONCE™ Mode

The ONCE ("On-Circuit Emulation") Mode facilitates testing and debugging of systems without the device having to be removed from the circuit. The ONCE Mode is invoked by:

1. Pull ALE low while the device is in reset and PSEN is high;
2. Hold ALE low as RST is deactivated.

While the device is in ONCE Mode, the Port 0 pins go into a float state, and the other port pins and ALE and PSEN are weakly pulled high. The oscillator circuit remains active. While the device is in this mode, an emulator or test CPU can be used to drive the circuit. Normal operation is restored when a normal reset is applied.

Programmable Clock-Out

A 50% duty cycle clock can be programmed to come out on P1.0. This pin, besides being a regular I/O pin, has two alternate functions. It can be programmed:

1. to input the external clock for Timer/Counter 2, or
2. to output a 50% duty cycle clock ranging from 122 Hz to 8 MHz at a 16 MHz operating frequency (61 Hz to 4 MHz in 12 clock mode).

To configure the Timer/Counter 2 as a clock generator, bit C/T2 (in T2CON) must be cleared and bit T20E in T2MOD must be set. Bit TR2 (T2CON.2) also must be set to start the timer.

The Clock-Out frequency depends on the oscillator frequency and the reload value of Timer 2 capture registers (RCAP2H, RCAP2L) as shown in this equation:

$$\text{Oscillator Frequency} \over n \times (65536 - \text{RCAP2H, RCAP2L})$$

n = 2 in 6 clock mode
4 in 12 clock mode

Where (RCAP2H,RCAP2L) = the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

In the Clock-Out mode, Timer 2 roll-overs will not generate an interrupt. This is similar to when it is used as a baud-rate generator. It is possible to use Timer 2 as a baud-rate generator and a clock generator simultaneously. Note, however, that the baud-rate and the Clock-Out frequency will be the same.

Table 2. External Pin Status During Idle and Power-Down Mode

MODE	PROGRAM MEMORY	ALE	PSEN	PORT 0	PORT 1	PORT 2	PORT 3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power-down	Internal	0	0	Data	Data	Data	Data
Power-down	External	0	0	Float	Data	Data	Data

TIMER 2 OPERATION

Timer 2

Timer 2 is a 16-bit Timer/Counter which can operate as either an event timer or an event counter, as selected by C/$\overline{T2}$" in the special function register T2CON (see Figure 1). Timer 2 has three operating modes: Capture, Auto-reload (up or down counting), and Baud Rate Generator, which are selected by bits in the T2CON as shown in Table 3.

Capture Mode

In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2=0, then timer 2 is a 16-bit timer or counter (as selected by C/$\overline{T2}$" in T2CON) which, upon overflowing sets bit TF2, the timer 2 overflow bit. This bit can be used to generate an interrupt (by enabling the Timer 2 interrupt bit in the IE register). If EXEN2= 1, Timer 2 operates as described above, but with the added feature that a 1- to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2 like TF2 can generate an interrupt (which vectors to the same location as Timer 2 overflow interrupt. The Timer 2 interrupt service routine can interrogate TF2 and EXF2 to determine which event caused the interrupt). The capture mode is illustrated in Figure 2. There is no reload value for TL2 and TH2 in this mode. Even when a capture event occurs from T2EX, the counter keeps on counting T2EX pin transitions or osc/6 pulses (osc/12 in 12 clock mode).

Auto-Reload Mode (Up or Down Counter)

In the 16-bit auto-reload mode, Timer 2 can be configured (as either a timer or counter (C/$\overline{T2}$" in T2CON)) then programmed to count up or down. The counting direction is determined by bit DCEN (Down Counter Enable) which is located in the T3MOD register (see Figure 3). When reset is applied the DCEN=0 which means Timer 2 will default to counting up. If DCEN bit is set, Timer 2 can count up or down depending on the value of the T2EX pin.

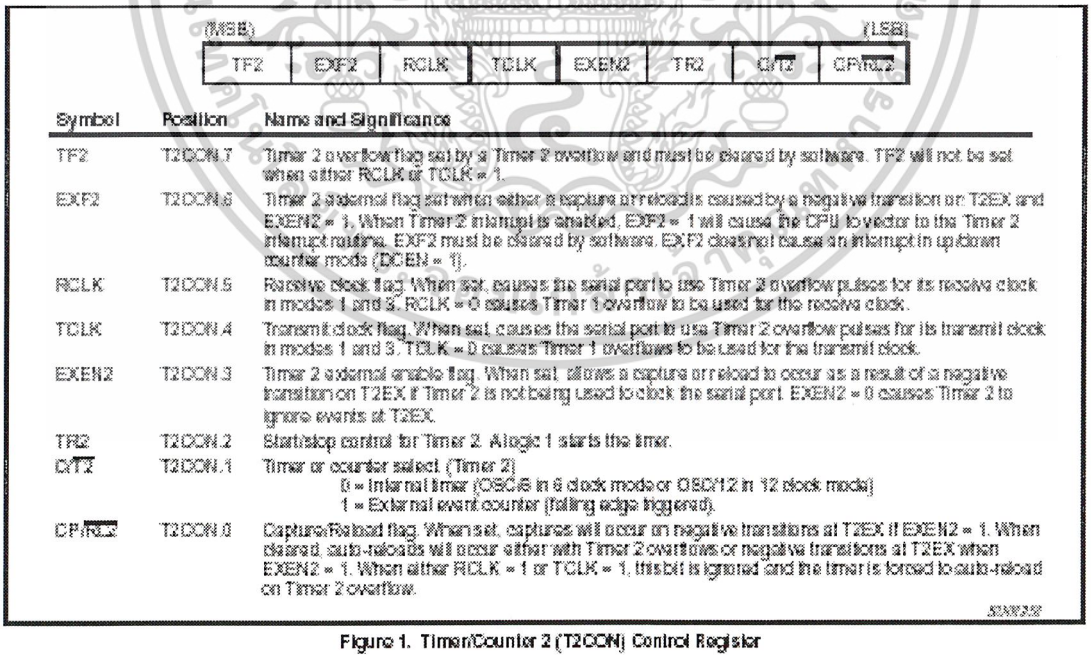
Figure 4 shows Timer 2 which will count up automatically since DCEN=0. In this mode there are two options selected by bit EXEN2 in T2CON register. If EXEN2=0, then Timer 2 counts up to 0FFFFH and sets the TF2 (Overflow Flag) bit upon overflow. This causes the Timer 2 registers to be reloaded with the 16-bit value in RCAP2L and RCAP2H. The values in RCAP2L and RCAP2H are preset by software means.

If EXEN2=1, then a 16-bit reload can be triggered either by an overflow or by a 1-to-0 transition at input T2EX. This transition also sets the EXF2 bit. The Timer 2 interrupt, if enabled, can be generated when either TF2 or EXF2 are 1.

In Figure 5 DCEN=1 which enables Timer 2 to count up or down. This mode allows pin T2EX to control the direction of count. When a logic 1 is applied at pin T2EX Timer 2 will count up. Timer 2 will overflow at 0FFFFH and set the TF2 flag, which can then generate an interrupt, if the interrupt is enabled. This timer overflow also causes the 16-bit value in RCAP2L and RCAP2H to be reloaded into the timer registers TL2 and TH2.

When a logic 0 is applied at pin T2EX this causes Timer 2 to count down. The timer will underflow when TL2 and TH2 become equal to the value stored in RCAP2L and RCAP2H. Timer 2 underflow sets the TF2 flag and causes 0FFFFH to be reloaded into the timer registers TL2 and TH2.

The external flag EXF2 toggles when Timer 2 underflows or overflows. This EXF2 bit can be used as a 17th bit of resolution if needed. The EXF2 flag does not generate an interrupt in this mode of operation.



Symbol	Position	Name and Significance
TF2	T2CON.7	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK or TCLK = 1.
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software. EXF2 does not cause an interrupt in up/down counter mode (DCEN = 1).
RCLK	T2CON.5	Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	T2CON.4	Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	T2CON.2	Start/stop control for Timer 2. A logic 1 starts the timer.
C/$\overline{T2}$	T2CON.1	Timer or counter select. (Timer 2) 0 = Internal timer (OSC/6 in 6 clock mode or OSC/12 in 12 clock mode) 1 = External event counter (falling edge triggered).
CP/$\overline{RL2}$	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto-reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

SU00232

Figure 1. Timer/Counter 2 (T2CON) Control Register

Table 3. Timer 2 Operating Modes

RCLK + TCLK	CP/RL2	TR2	MODE
0	0	1	16-bit Auto-reload
0	1	1	16-bit Capture
1	X	1	Baud rate generator
X	X	0	(off)

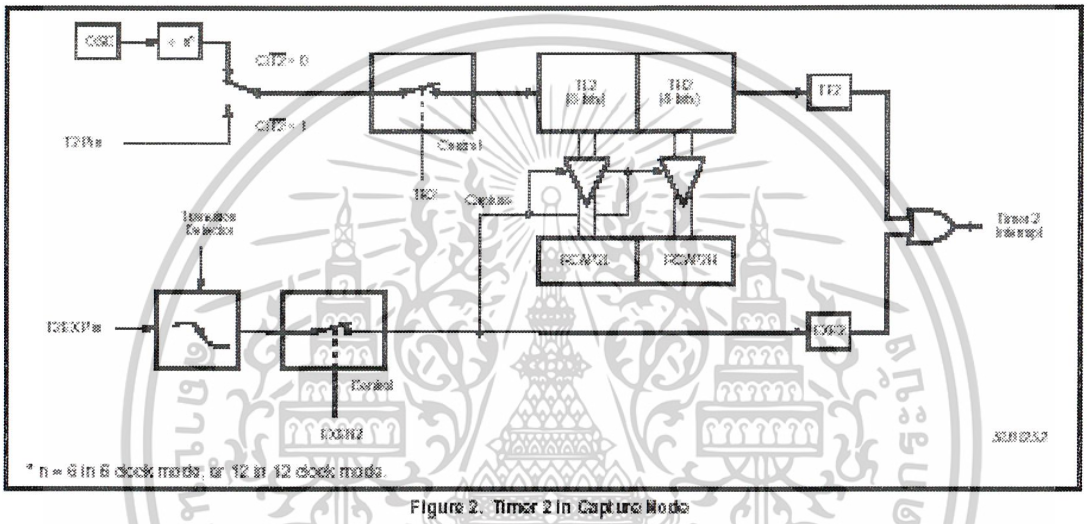


Figure 2. Timer 2 in Capture Mode

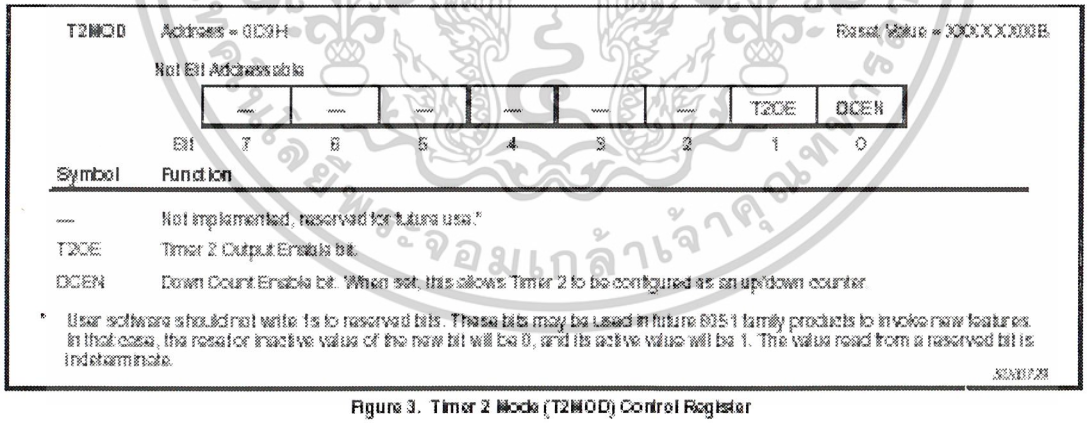


Figure 3. Timer 2 Mode (T2MOD) Control Register

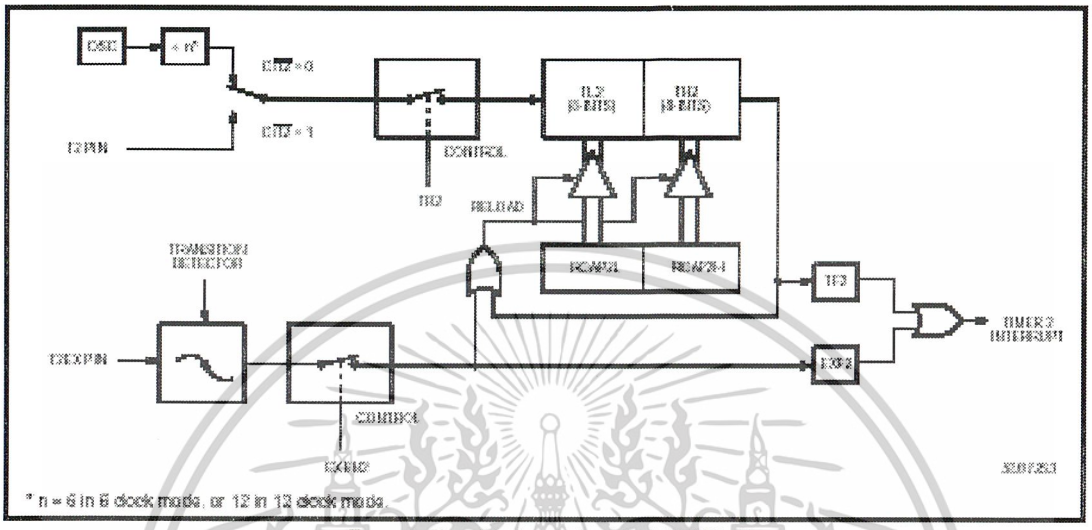


Figure 4. Timer 2 in Auto-Reload Mode (DCEN = 0)

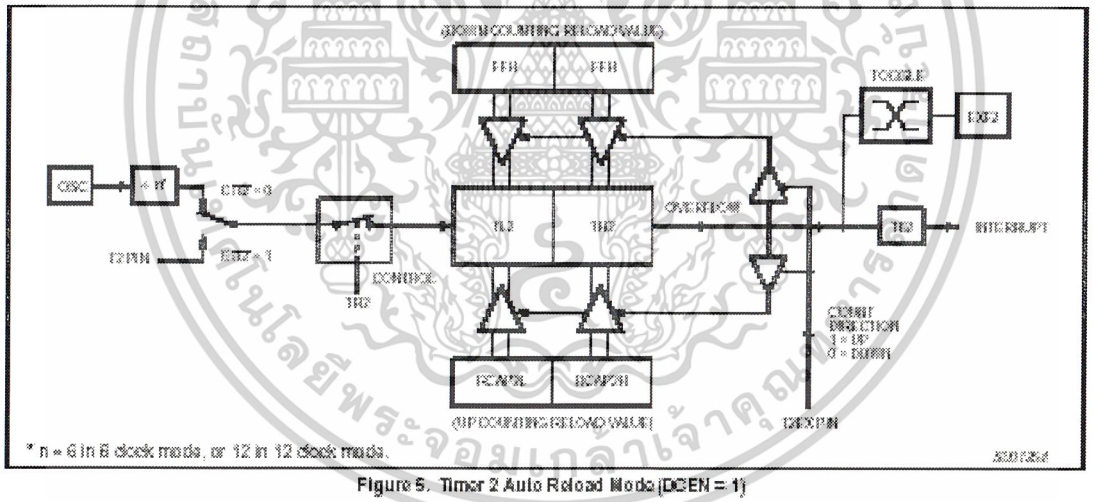


Figure 5. Timer 2 Auto Reload Mode (DCEN = 1)

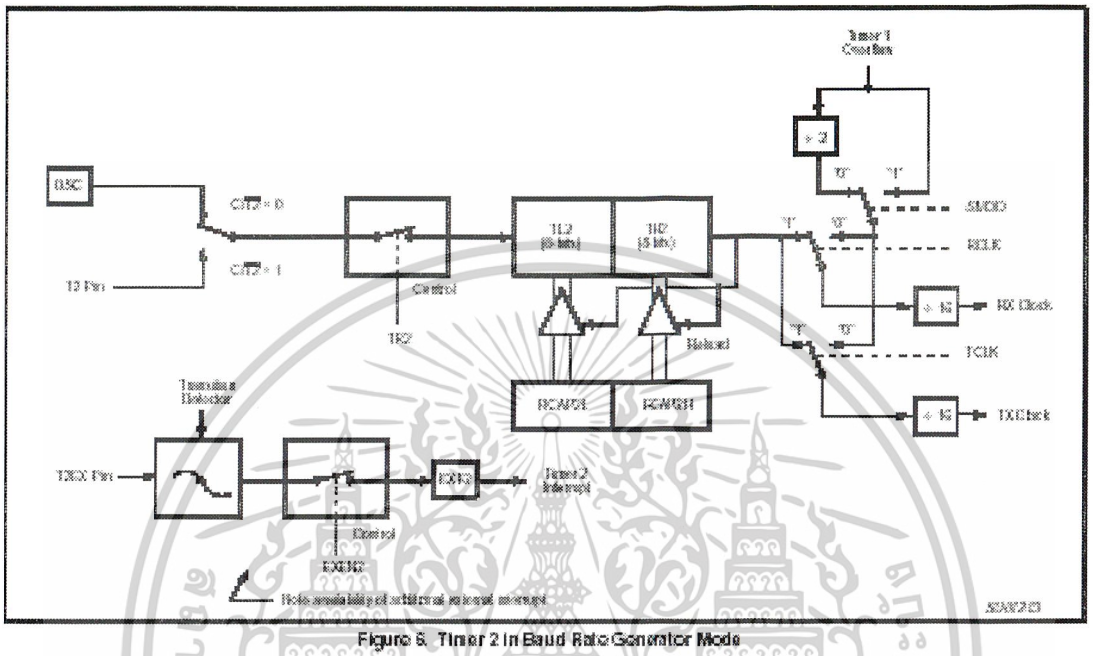


Figure 6. Timer 2 in Baud Rate Generator Mode

Table 4. Timer 2 Generated Commonly Used Baud Rates

Baud Rate		Osc Freq	Timer2	
12 clock mode	6 clock mode		RCAP2H	RCAP2L
375 k	750 k	12 MHz	FF	FF
9.6 k	19.2 k	12 MHz	FF	D9
2.8 k	5.6 k	12 MHz	FF	B2
2.4 k	4.8 k	12 MHz	FF	64
1.2 k	2.4 k	12 MHz	FE	C8
300	600	12 MHz	FB	1E
110	220	12 MHz	F2	AF
300	600	6 MHz	FD	8F
110	220	6 MHz	F9	57

Baud Rate Generator Mode

Bits TCLK and/or RCLK in T2CON (Table 4) allow the serial port transmit and receive baud rates to be derived from either Timer 1 or Timer 2. When TCLK= 0, Timer 1 is used as the serial port transmit baud rate generator. When TCLK= 1, Timer 2 is used as the serial port transmit baud rate generator. RCLK has the same effect for the serial port receive baud rate. With these two bits, the serial port can have different receive and transmit baud rates – one generated by Timer 1, the other by Timer 2.

Figure 6 shows the Timer 2 in baud rate generation mode. The baud rate generation mode is like the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

The baud rates in modes 1 and 3 are determined by Timer 2's overflow rate given below:

$$\text{Modes 1 and 3 Baud Rates} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The timer can be configured for either "timer" or "counter" operation. In many applications, it is configured for "timer" operation ($C/\overline{T2}$=0). Timer operation is different for Timer 2 when it is being used as a baud rate generator.

Usually, as a timer it would increment every machine cycle (i.e., $^1/_6$ the oscillator frequency in 6 clock mode, $^1/_{12}$ the oscillator frequency in 12 clock mode). As a baud rate generator, it increments at the oscillator frequency in the clock mode ($^{OSC}/_2$ in 12 clock mode). Thus the baud rate formula is as follows:

$$\text{Modes 1 and 3 Baud Rates} = \frac{\text{Oscillator Frequency}}{[n^* \times (65536 - (RCAP2H, RCAP2L))]}$$

*n = 16 in 6 clock mode
32 in 12 clock mode

Where: (RCAP2H, RCAP2L)= The content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

The Timer 2 as a baud rate generator mode shown in Figure 6, is valid only if RCLK and/or TCLK = 1 in T2CON register. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Thus, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Also if the EXEN2 (T2 external enable flag) is set, a 1-to-0 transition in T2EX (Timer/counter 2 trigger input) will set EXF2 (T2 external flag) but will not cause a reload from (RCAP2H, RCAP2L) to (TH2,TL2). Therefore when Timer 2 is in use as a baud rate generator, T2EX can be used as an additional external interrupt, if needed.

When Timer 2 is in the baud rate generator mode, one should not try to read or write TH2 and TL2. As a baudrate generator, Timer 2 is incremented every state time (osc/2) or asynchronously from pin T2; under these conditions, a read or write of TH2 or TL2 may not be accurate. The RCAP2 registers may be read, but should not be written to, because a write might overlap a reload and cause write and/or reload errors. The timer should be turned off (clear TR2) before accessing the Timer 2 or RCAP2 registers.

Table 4 shows commonly used baud rates and how they can be obtained from Timer 2.

Summary of Baud Rate Equations

Timer 2 is in baud rate generating mode. If Timer 2 is being clocked through pin T2(P1.0) the baud rate is:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

If Timer 2 is being clocked internally, the baud rate is:

$$\text{Baud Rate} = \frac{f_{osc}}{n^* \times [65536 - (RCAP2H, RCAP2L)]}$$

* n = 16 in 6 clock mode
32 in 12 clock mode

Where f_{osc} = Oscillator Frequency

To obtain the reload value for RCAP2H and RCAP2L, the above equation can be rewritten as:

$$RCAP2H, RCAP2L = 65536 - \left(\frac{f_{osc}}{n^* \times \text{Baud Rate}}\right)$$

Timer/Counter 2 Set-up

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set, separately, to turn the timer on. See Table 5 for set-up of Timer 2 as a timer. Also see Table 6 for set-up of Timer 2 as a counter.

Table 5. Timer 2 as a Timer

MODE	T2CON	
	INTERNAL CONTROL (Note 1)	EXTERNAL CONTROL (Note 2)
16-bit Auto-Reload	00H	08H
16-bit Capture	01H	09H
Baud rate generator receive and transmit same baud rate	34H	36H
Receive only	24H	26H
Transmit only	14H	16H

Table 6. Timer 2 as a Counter

MODE	T2MOD	
	INTERNAL CONTROL (Note 1)	EXTERNAL CONTROL (Note 2)
16-bit	02H	0AH
Auto-Reload	03H	0BH

NOTES:
1. Capture/reload occurs only on timer/counter overflow.
2. Capture/reload occurs on timer/counter overflow and a 1-to-0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baudrate generator mode.

Enhanced UART

The UART operates in all of the usual modes that are described in the first section of *Data Handbook IC20, 80C51-Based 8-Bit Microcontrollers*. In addition the UART can perform framing error detect by looking for missing stop bits, and automatic address recognition. The UART also fully supports multiprocessor communication as does the standard 80C51 UART.

When used for framing error detect the UART looks for missing stop bits in the communication. A missing bit will set the FE bit in the SCON register. The FE bit shares the SCON.7 bit with SM0 and the function of SCON.7 is determined by PCON.6 (SMOD0) (see Figure 7). If SMOD0 is set then SCON.7 functions as FE. SCON.7 functions as SM0 when SMOD0 is cleared. When used as FE, SCON.7 can only be cleared by software. Refer to Figure 8.

Automatic Address Recognition

Automatic Address Recognition is a feature which allows the UART to recognize certain addresses in the serial bit stream by using hardware to make the comparisons. This feature saves a great deal of software overhead by eliminating the need for the software to examine every serial address which passes by the serial port. This feature is enabled by setting the SM2 bit in SCON. In the 9 bit UART modes, mode 2 and mode 3, the Receive Interrupt flag (RI) will be automatically set when the received byte contains either the "Given" address or the "Broadcast" address. The 9-bit mode requires that the 9th information bit is a 1 to indicate that the received information is an address and not data. Automatic address recognition is shown in Figure 9.

The 8 bit mode is called Mode 1. In this mode the RI flag will be set if SM2 is enabled and the information received has a valid stop bit following the 8 address bits and the information is either a Given or Broadcast address.

Mode 0 is the Shift Register mode and SM2 is ignored.

Using the Automatic Address Recognition feature allows a master to selectively communicate with one or more slaves by invoking the Given slave address or addresses. All of the slaves may be contacted by using the Broadcast address. Two special Function Registers are used to define the slave's address, SADDR, and the address mask, SADEN. SADEN is used to define which bits in the SADDR are to be used and which bits are "don't care". The SADEN mask can be logically ANDed with the SADDR to create the "Given" address which the master will use for addressing each of the slaves. Use of the Given address allows multiple slaves to be recognized while excluding others. The following examples will help to show the versatility of this scheme:

Slave 0	SADDR	=	1100 0000
	SADEN	=	1111 1101
	Given	=	1100 00X0

Slave 1	SADDR	=	1100 0000
	SADEN	=	1111 1110
	Given	=	1100 000X

In the above example SADDR is the same and the SADEN data is used to differentiate between the two slaves. Slave 0 requires a 0 in bit 0 and it ignores bit 1. Slave 1 requires a 0 in bit 1 and bit 0 is ignored. A unique address for Slave 0 would be 1100 0010 since slave 1 requires a 0 in bit 1. A unique address for slave 1 would be 1100 0001 since a 1 in bit 0 will exclude slave 0. Both slaves can be selected at the same time by an address which has bit 0 = 0 (for slave 0) and bit 1 = 0 (for slave 1). Thus, both could be addressed with 1100 0000.

In a more complex system the following could be used to select slaves 1 and 2 while excluding slave 0:

Slave 0	SADDR	=	1100 0000
	SADEN	=	1111 1001
	Given	=	1100 0000
Slave 1	SADDR	=	1110 0000
	SADEN	=	1111 1010
	Given	=	1110 0X0X
Slave 2	SADDR	=	1110 0000
	SADEN	=	1111 1100
	Given	=	1110 00XX

In the above example the differentiation among the 3 slaves is in the lower 3 address bits. Slave 0 requires that bit 0 = 0 and it can be uniquely addressed by 1110 0110. Slave 1 requires that bit 1 = 0 and it can be uniquely addressed by 1110 and 0101. Slave 2 requires that bit 2 = 0 and its unique address is 1110 0011. To select Slaves 0 and 1 and exclude Slave 2 use address 1110 0100, since it is necessary to make bit 2 = 1 to exclude slave 2.

The Broadcast Address for each slave is created by taking the logical OR of SADDR and SADEN. Zeros in this result are treated as don't-cares. In most cases, interpreting the don't-cares as ones, the broadcast address will be FF hexadecimal.

Upon reset SADDR (SFR address 0A9H) and SADEN (SFR address 0B9H) are loaded with 0s. This produces a given address of all "don't cares" as well as a Broadcast address of all "don't cares". This effectively disables the Automatic Addressing mode and allows the microcontroller to use standard 80C51 type UART drivers which do not make use of this feature.

SCON Address = 98H — Reset Value = 0000 0000B

Bit Addressable

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

(SMOD0 = 0/1)*

Symbol	Function
FE	Framing Error bit. This bit is set by the receiver when an invalid stop bit is detected. The FE bit is not cleared by valid frames but should be cleared by software. The SMOD0 bit must be set to enable access to the FE bit.
SM0	Serial Port Mode Bit 0. (SMOD0 must = 0 to access bit SM0)
SM1	Serial Port Mode Bit 1 SM0 SM1 Mode — Description — Baud Rate** 0 0 0 — shift register — $f_{osc}/6$ (6 clock mode) or $f_{osc}/12$ (12 clock mode) 0 1 1 — 8-bit UART — variable 1 0 2 — 9-bit UART — $f_{osc}/32$ or $f_{osc}/16$ (6 clock mode) or $f_{osc}/64$ or $f_{osc}/32$ (12 clock mode) 1 1 3 — 9-bit UART — variable
SM2	Enables the Automatic Address Recognition feature in Modes 2 or 3. If SM2 = 1 then RI will not be set unless the received 9th data bit (RB8) is 1, indicating an address, and the received byte is a Given or Broadcast Address. In Mode 1, if SM2 = 1 then RI will not be activated unless a valid stop bit was received, and the received byte is a Given or Broadcast Address. In Mode 0, SM2 should be 0.
REN	Enables serial reception. Set by software to enable reception. Clear by software to disable reception.
TB8	The 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired.
RB8	In modes 2 and 3, the 9th data bit that was received. In Mode 1, if SM2 = 0, RB8 is the stop bit that was received. In Mode 0, RB8 is not used.
TI	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.
RI	Receive interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

NOTE:
*SMOD0 is located at PCON.6.
**f_{osc} = oscillator frequency

SU01263

Figure 7. SCON: Serial Port Control Register

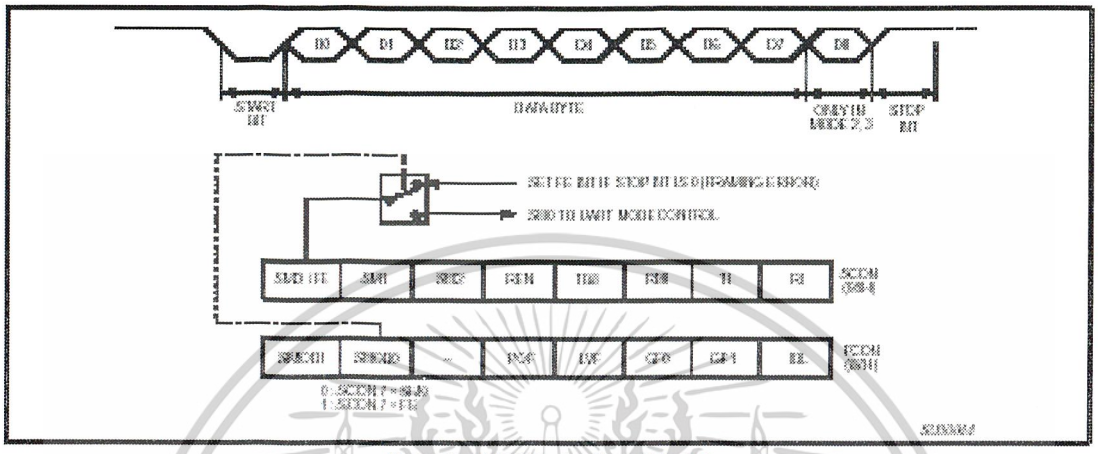


Figure 8. UART Framing Error Detection

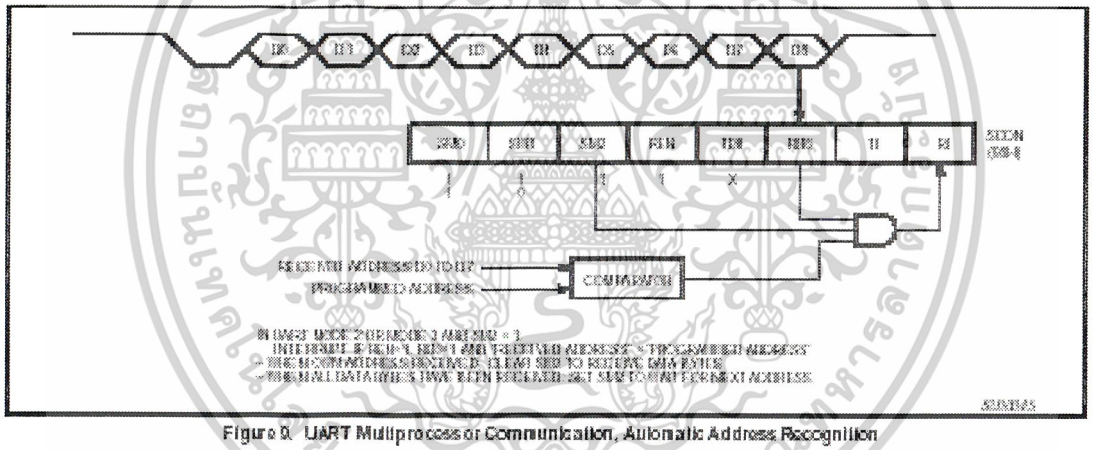


Figure 9. UART Multiprocessor Communication, Automatic Address Recognition

Interrupt Priority Structure

The P89C51RB2/RC2/RD2 has a 7 source four-level interrupt structure (see Table 7).

There are 3 SFRs associated with the four-level interrupt. They are the IE, IP, and IPH. (See Figures 10, 11, and 12.) The IPH (Interrupt Priority High) register makes the four-level interrupt structure possible. The IPH is located at SFR address B7H. The structure of the IPH register and a description of its bits is shown in Figure 12.

The function of the IPH SFR, when combined with the IP SFR, determines the priority of each interrupt. The priority of each interrupt is determined as shown in the following table:

PRIORITY BITS		INTERRUPT PRIORITY LEVEL
IPH.x	IPx	
0	0	Level 0 (lowest priority)
0	1	Level 1
1	0	Level 2
1	1	Level 3 (highest priority)

The priority scheme for servicing the interrupts is the same as that for the 80C51, except there are four interrupt levels rather than two as on the 80C51. An interrupt will be serviced as long as an interrupt of equal or higher priority is not already being serviced. If an interrupt of equal or higher level priority is being serviced, the new interrupt will wait until it is finished before being serviced. If a lower priority level interrupt is being serviced, it will be stopped and the new interrupt serviced. When the new interrupt is finished, the lower priority level interrupt that was stopped will be completed.

Table 7. Interrupt Table

SOURCE	POLLING PRIORITY	REQUEST BITS	HARDWARE CLEAR?	VECTOR ADDRESS
X0	1	IE0	N (L)[1] Y (T)[2]	03H
T0	2	TF0	Y	0BH
X1	3	IE1	N (L) Y (T)	13H
T1	4	TF1	Y	1BH
PCA	5	CF, CCFn n = 0–4	N	33H
SP	6	RI, TI	N	23H
T2	7	TF2, EXF2	N	2BH

NOTES:
1. L = Level activated
2. T = Transition activated

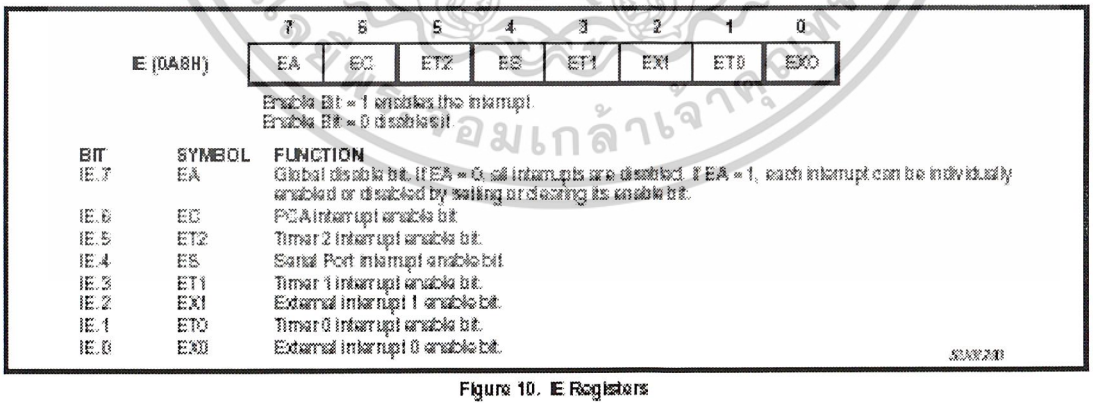
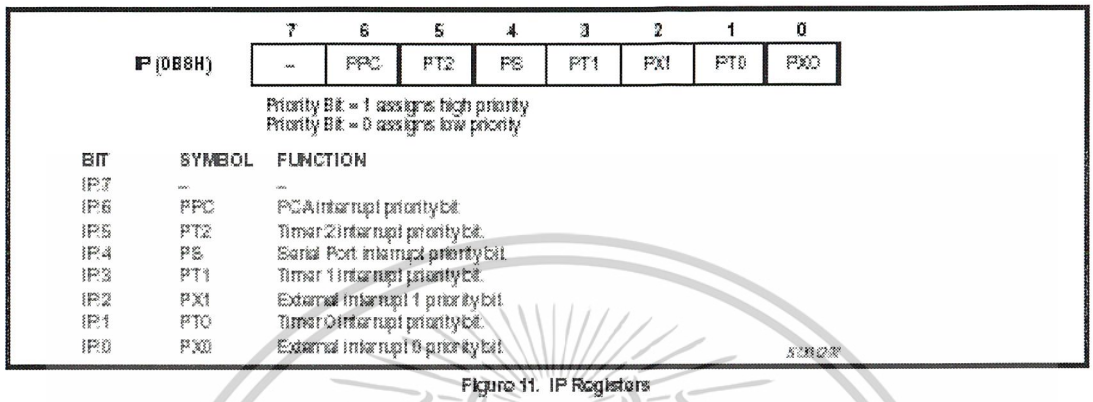


Figure 10. IE Registers

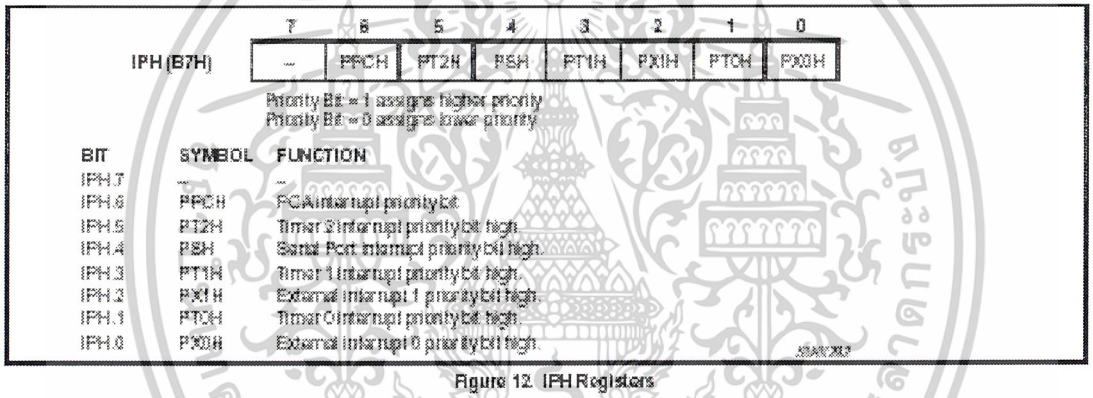


	7	6	5	4	3	2	1	0
IP (0B8H)	–	PPC	PT2	PS	PT1	PX1	PT0	PX0

Priority Bit = 1 assigns high priority
Priority Bit = 0 assigns low priority

BIT	SYMBOL	FUNCTION
IP.7	–	–
IP.6	PPC	PCA interrupt priority bit
IP.5	PT2	Timer 2 interrupt priority bit.
IP.4	PS	Serial Port interrupt priority bit.
IP.3	PT1	Timer 1 interrupt priority bit.
IP.2	PX1	External interrupt 1 priority bit.
IP.1	PT0	Timer 0 interrupt priority bit.
IP.0	PX0	External interrupt 0 priority bit.

Figure 11. IP Registers



	7	6	5	4	3	2	1	0
IPH (B7H)	–	PPCH	PT2H	PSH	PT1H	PX1H	PT0H	PX0H

Priority Bit = 1 assigns higher priority
Priority Bit = 0 assigns lower priority

BIT	SYMBOL	FUNCTION
IPH.7	–	–
IPH.6	PPCH	PCA interrupt priority bit.
IPH.5	PT2H	Timer 2 interrupt priority bit high.
IPH.4	PSH	Serial Port interrupt priority bit high.
IPH.3	PT1H	Timer 1 interrupt priority bit high.
IPH.2	PX1H	External interrupt 1 priority bit high.
IPH.1	PT0H	Timer 0 interrupt priority bit high.
IPH.0	PX0H	External interrupt 0 priority bit high.

Figure 12. IPH Registers

Reduced EMI Mode

The AO bit (AUXR.0) in the AUXR register when set disables the ALE output.

Reduced EMI Mode

AUXR (8EH)

7	6	5	4	3	2	1	0
–	–	–	–	–	–	EXTRAM	AO

AUXR.1 EXTRAM
AUXR.0 AO Turns off ALE output

Dual DPTR

The dual DPTR structure (see Figure 13) is a way by which the chip will specify the address of an external data memory location. There are two 16-bit DPTR registers that address the external memory, and a single bit called DPS = AUXR1/bit0 that allows the program code to switch between them.

- New Register Name: AUXR1#
- SFR Address: A2H
- Reset Value: xxxxxxx0B

AUXR1 (A2H)

7	6	5	4	3	2	1	0
–	–	EHBOOT	–	GF2	0	–	DPS

Where:
DPS = AUXR1/bit0 = Switches between DPTR0 and DPTR1.

Select Reg	DPS
DPTR0	0
DPTR1	1

The DPS bit status should be saved by software when switching between DPTR0 and DPTR1.

The GF2 bit is a general purpose user-defined flag. Note that bit 2 is not writable and is always read as a zero. This allows the DPS bit to be quickly toggled simply by executing an INC AUXR1 instruction without affecting the GF2 bit.

The EHBOOT bit determines whether the BOOTROM is enabled or disabled. This bit will automatically be set if the status byte is non-zero during reset or PSEN is pulled low, ALE floats high, and EA = V_{IH} on the falling edge of reset. Otherwise, this bit will be cleared during reset.

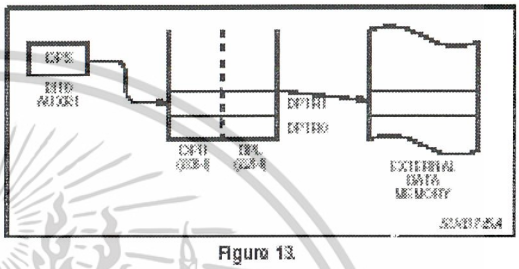


Figure 13.

DPTR Instructions

The instructions that refer to DPTR refer to the data pointer that is currently selected using the AUXR1/bit 0 register. The six instructions that use the DPTR are as follows:

INC DPTR	Increments the data pointer by 1
MOV DPTR, #data16	Loads the DPTR with a 16-bit constant
MOVC A, @ A+DPTR	Move code byte relative to DPTR to ACC
MOVX A, @ DPTR	Move external RAM (16-bit address) to ACC
MOVX @ DPTR, A	Move ACC to external RAM (16-bit address)
JMP @ A + DPTR	Jump indirect relative to DPTR

The data pointer can be accessed on a byte-by-byte basis by specifying the low or high byte in an instruction which accesses the SFRs. See Application Note *AN458* for more details.

Programmable Counter Array (PCA)

The Programmable Counter Array available on the 80C51 RB2/RC2/RD2 is a special 16-bit Timer that has five 16-bit capture/compare modules associated with it. Each of the modules can be programmed to operate in one of four modes: rising and/or falling edge capture, software timer, high-speed output, or pulse width modulator. Each module has a pin associated with it in port 1. Module 0 is connected to P1.3(CEX0), module 1 to P1.4(CEX1), etc. The basic PCA configuration is shown in Figure 14.

The PCA timer is a common time base for all five modules and can be programmed to run at: 1/6 the oscillator frequency, 1/2 the oscillator frequency, the Timer 0 overflow, or the input on the ECI pin (P1.2). The timer count source is determined from the CPS1 and CPS0 bits in the CMOD SFR as follows (see Figure 17):

CPS1	CPS0	PCA Timer Count Source
0	0	1/6 oscillator frequency (6 clock mode) 1/12 oscillator frequency (12 clock mode)
0	1	1/2 oscillator frequency (6 clock mode) 1/4 oscillator frequency (12 clock mode)
1	0	Timer 0 overflow
1	1	External input at ECI pin

In the CMOD SFR are three additional bits associated with the PCA. They are CIDL which allows the PCA to stop during idle mode, WDTE which enables or disables the watchdog function on module 4, and ECF which when set causes an interrupt and the PCA overflow flag CF (in the CCON SFR) to be set when the PCA timer overflows. These functions are shown in Figure 18.

The watchdog timer function is implemented in module 4 (see Figure 24).

The CCON SFR contains the run control bit for the PCA and the flags for the PCA timer (CF) and each module (refer to Figure 16). To run the PCA the CR bit (CCON.6) must be set by software. The PCA is shut off by clearing this bit. The CF bit (CCON.7) is set when the PCA counter overflows and an interrupt will be generated if the ECF bit in the CMOD register is set. The CF bit can only be cleared by software. Bits 0 through 4 of the CCON register are the flags for the modules (bit 0 for module 0, bit 1 for module 1, etc.) and are set by hardware when either a match or a capture occurs. These flags also can only be cleared by software. The PCA interrupt system shown in Figure 16.

Each module in the PCA has a special function register associated with it. These registers are: CCAPM0 for module 0, CCAPM1 for module 1, etc. (see Figure 19). The registers contain the bits that control the mode that each module will operate in. The ECCF bit (CCAPMn.0 where n=0, 1, 2, 3, or 4 depending on the module) enables the CCF flag in the CCON SFR to generate an interrupt when a match or compare occurs in the associated module. PWM (CCAPMn.1) enables the pulse width modulation mode. The TOG bit (CCAPMn.2) when set causes the CEX output associated with the module to toggle when there is a match between the PCA counter and the module's capture/compare register. The match bit MAT (CCAPMn.3) when set will cause the CCFn bit in the CCON register to be set when there is a match between the PCA counter and the module's capture/compare register.

The next two bits CAPN (CCAPMn.4) and CAPP (CCAPMn.5) determine the edge that a capture input will be active on. The CAPN bit enables the negative edge, and the CAPP bit enables the positive edge. If both bits are set both edges will be enabled and a capture will occur for either transition. The last bit in the register ECOM (CCAPMn.6) when set enables the comparator function. Figure 20 shows the CCAPMn settings for the various PCA functions.

There are two additional registers associated with each of the PCA modules. They are CCAPnH and CCAPnL and these are the registers that store the 16-bit count when a capture occurs or a compare should occur. When a module is used in the PWM mode these registers are used to control the duty cycle of the output.

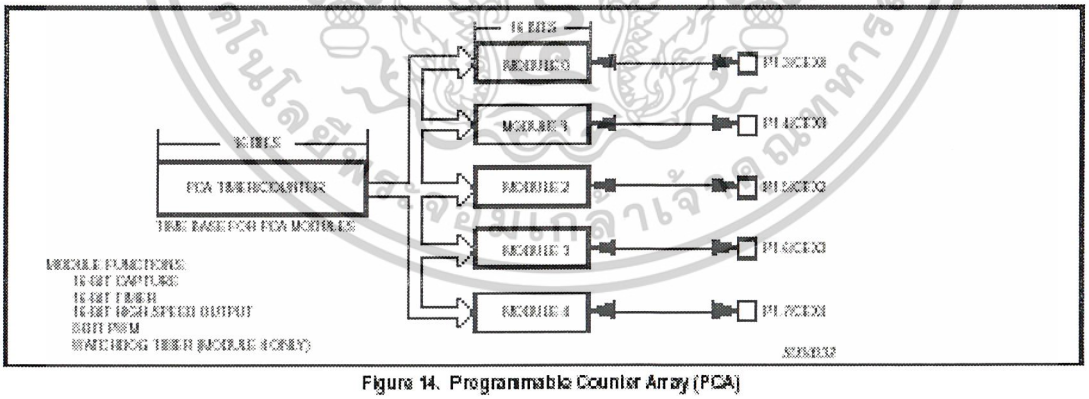


Figure 14. Programmable Counter Array (PCA)

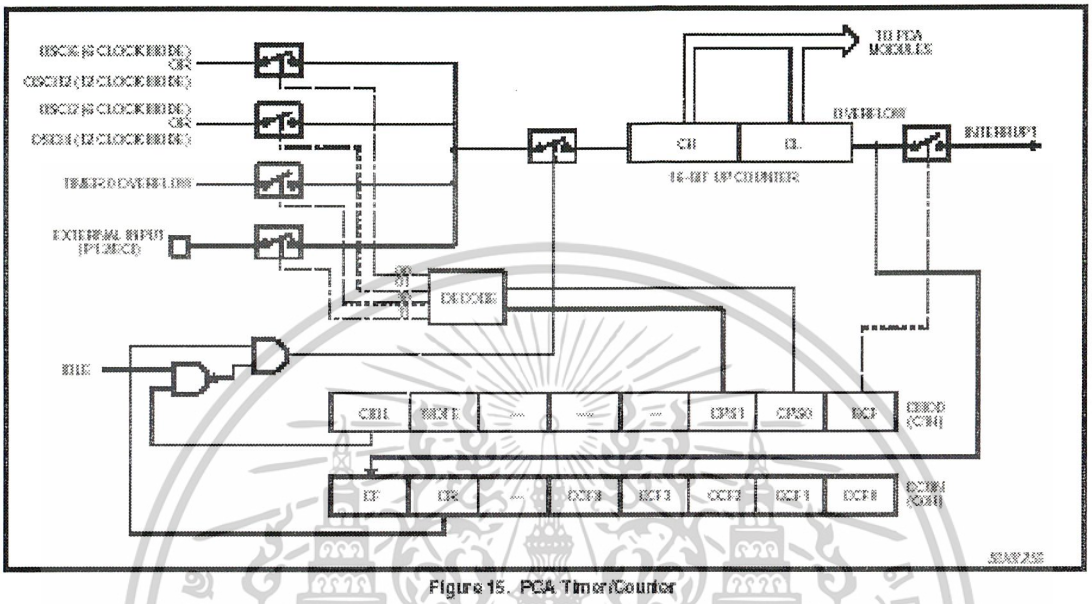


Figure 15. PCA Timer/Counter

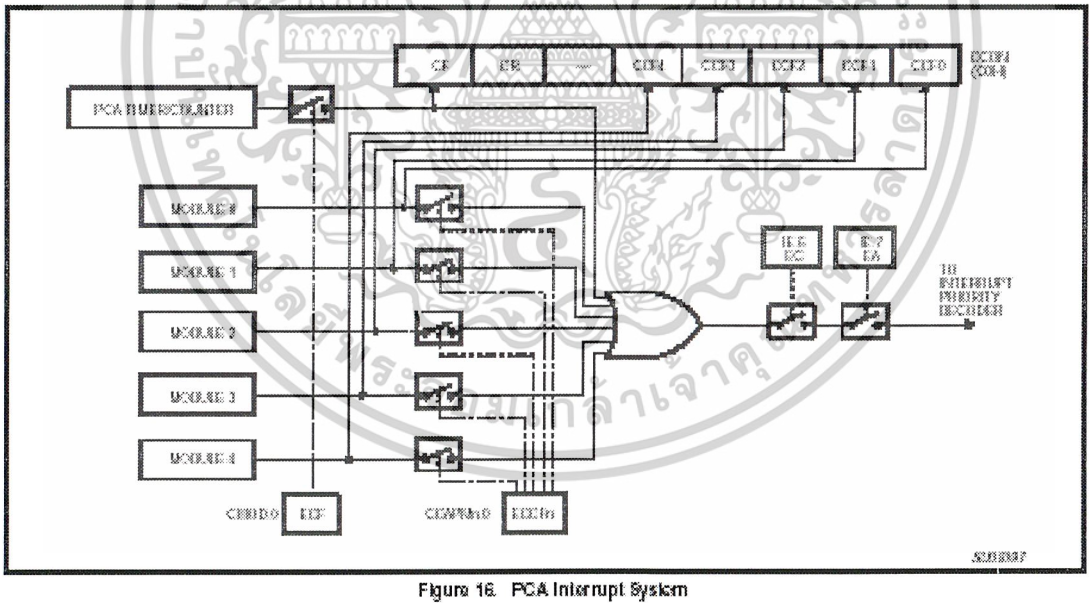


Figure 16. PCA Interrupt System

CMOD Address = D9H — Reset Value = 00XX X000B

CIDL	WDTE	–	–	–	CPS1	CPS0	ECF
7	6	5	4	3	2	1	0

Bit:

Symbol	Function
CIDL	Counter Idle control: CIDL = 0 programs the PCA Counter to continue functioning during Idle Mode. CIDL = 1 programs it to be gated off during Idle.
WDTE	Watchdog Timer Enable: WDTE = 0 disables Watchdog Timer function on PCA Module 4. WDTE = 1 enables it.
–	Not implemented, reserved for future use.*
CPS1	PCA Count Pulse Select bit 1.
CPS0	PCA Count Pulse Select bit 0.
ECF	PCA Enable Counter Overflow interrupt: ECF = 1 enables CF bit in CCON to generate an interrupt. ECF = 0 disables that function of CF.

CPS1 CPS0 Selected PCA input*

CPS1	CPS0	Selected PCA input*
0	0	0 — Internal clock, f_{OSC}/6 in 6 clock mode (f_{OSC}/12 in 12 clock mode)
0	1	1 — Internal clock, f_{OSC}/2 in 6 clock mode (f_{OSC}/4 in 12 clock mode)
1	0	2 — Timer 0 overflow
1	1	3 — External clock at ECI/P1.2 pin (max. rate = f_{OSC}/4 in 6 clock mode, f_{OSC}/8 in 12 clock mode)

NOTE:
* User software should not write 1s to reserved bits. These bits may be used in future 80C51 family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1. The value read from a reserved bit is indeterminate.
** f_{OSC} = oscillator frequency

SU01308

Figure 17. CMOD: PCA Counter Mode Register

CCON Address = D8H — Reset Value = 00X0 0000B

Bit Addressable

CF	CR	–	CCF4	CCF3	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0

Bit:

Symbol	Function
CF	PCA Counter Overflow flag. Set by hardware when the counter rolls over. CF flags an interrupt if bit ECF in CMOD is set. CF may be set by either hardware or software but can only be cleared by software.
CR	PCA Counter Run control bit. Set by software to turn the PCA counter on. Must be cleared by software to turn the PCA counter off.
–	Not implemented, reserved for future use.*
CCF4	PCA Module 4 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.
CCF3	PCA Module 3 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.
CCF2	PCA Module 2 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.
CCF1	PCA Module 1 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.
CCF0	PCA Module 0 interrupt flag. Set by hardware when a match or capture occurs. Must be cleared by software.

NOTE:
* User software should not write 1s to reserved bits. These bits may be used in future 80C51 family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1. The value read from a reserved bit is indeterminate.

SU01309

Figure 18. CCON: PCA Counter Control Register

CCAPMn Address: CCAPM0 0DAH; CCAPM1 0DBH; CCAPM2 0DCH; CCAPM3 0DDH; CCAPM4 0DEH — Reset Value = XX00 0000B

Not Bit Addressable

–	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
7	6	5	4	3	2	1	0

Symbol	Function
–	Not implemented, reserved for future use*.
ECOMn	Enable Comparator. ECOMn = 1 enables the comparator function.
CAPPn	Capture Positive, CAPPn = 1 enables positive edge capture.
CAPNn	Capture Negative, CAPNn = 1 enables negative edge capture.
MATn	Match. When MATn = 1, a match of the PCA counter with this module's compare/capture register causes the CCFn bit in CCON to be set, flagging an interrupt.
TOGn	Toggle. When TOGn = 1, a match of the PCA counter with this module's compare/capture register causes the CEXn pin to toggle.
PWMn	Pulse Width Modulation Mode. PWMn = 1 enables the CEXn pin to be used as a pulse width modulated output.
ECCFn	Enable CCF interrupt. Enables compare/capture flag CCFn in the CCON register to generate an interrupt.

NOTE:
*User software should not write 1s to reserved bits. These bits may be used in future 80C51 family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1. The value read from a reserved bit is indeterminate.

SU00130

Figure 19. CCAPMn: PCA Modules Compare/Capture Registers

–	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	MODULE FUNCTION
X	0	0	0	0	0	0	0	No operation
X	X	1	0	0	0	0	X	16-bit capture by a positive-edge trigger on CEXn
X	X	0	1	0	0	0	X	16-bit capture by a negative trigger on CEXn
X	X	1	1	0	0	0	X	16-bit capture by a transition on CEXn
X	1	0	0	1	0	0	X	16-bit Software Timer
X	1	0	0	1	1	0	X	16-bit High Speed Output
X	1	0	0	0	0	1	0	8-bit PWM
X	1	0	0	1	X	0	X	Watchdog Timer

Figure 20. PCA Module Modes (CCAPMn Register)

PCA Capture Mode
To use one of the PCA modules in the capture mode either one or both of the CCAPM bits CAPN and CAPP for that module must be set. The external CEX input for the module (on port 1) is sampled for a transition. When a valid transition occurs the PCA hardware loads the value of the PCA counter registers (CH and CL) into the module's capture registers (CCAPnL and CCAPnH). If the CCFn bit for the module in the CCON SFR and the ECCFn bit in the CCAPMn SFR are set then an interrupt will be generated. Refer to Figure 21.

16-bit Software Timer Mode
The PCA modules can be used as software timers by setting both the ECOM and MAT bits in the modules CCAPMn register. The PCA timer will be compared to the module's capture registers and when a match occurs an interrupt will occur if the CCFn (CCON SFR) and the ECCFn (CCAPMn SFR) bits for the module are both set (see Figure 22).

High Speed Output Mode
In this mode the CEX output (on port 1) associated with the PCA module will toggle each time a match occurs between the PCA counter and the module's capture registers. To activate this mode the TOG, MAT, and ECOM bits in the module's CCAPMn SFR must be set (see Figure 23).

Pulse Width Modulator Mode
All of the PCA modules can be used as PWM outputs. Figure 24 shows the PWM function. The frequency of the output depends on the source for the PCA timer. All of the modules will have the same frequency of output because they all share the PCA timer. The duty cycle of each module is independently variable using the module's capture register CCAPLn. When the value of the PCA CL SFR is less than the value in the module's CCAPLn SFR the output will be low, when it is equal to or greater than the output will be high. When CL overflows from FF to 00, CCAPLn is reloaded with the value in CCAPHn. this allows updating the PWM without glitches. The PWM and ECOM bits in the module's CCAPMn register must be set to enable the PWM mode.

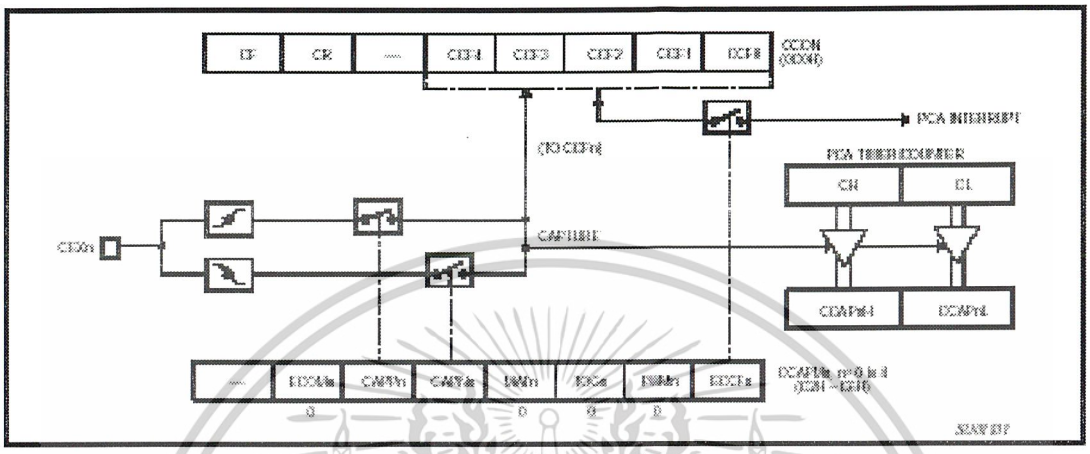


Figure 21. PCA Capture Mode

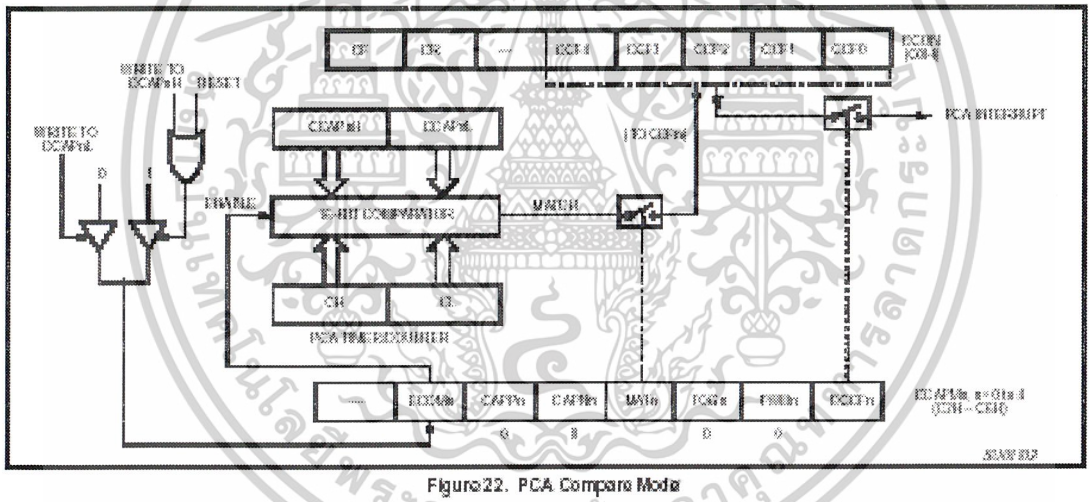


Figure 22. PCA Compare Mode

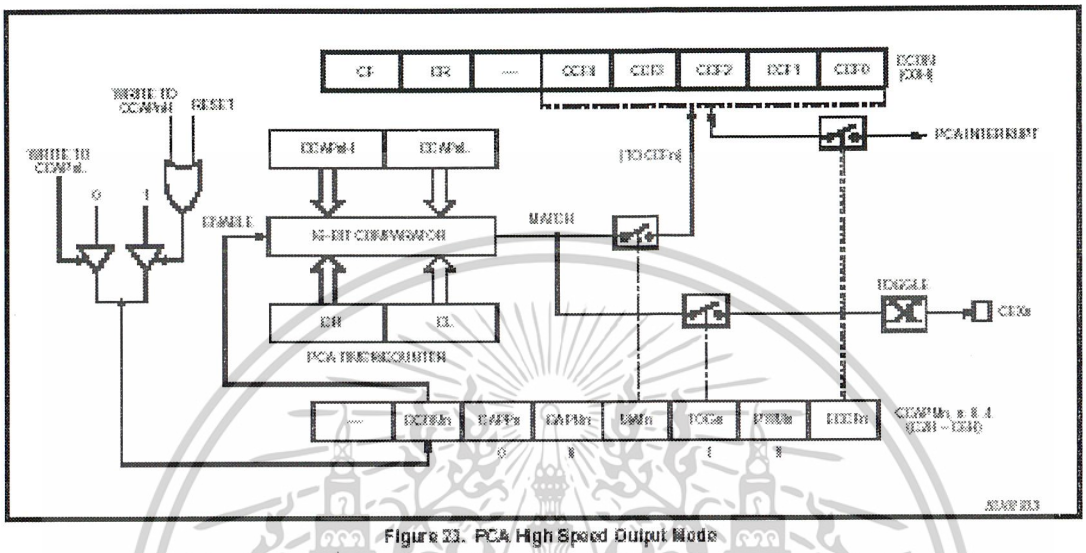


Figure 23. PCA High Speed Output Mode

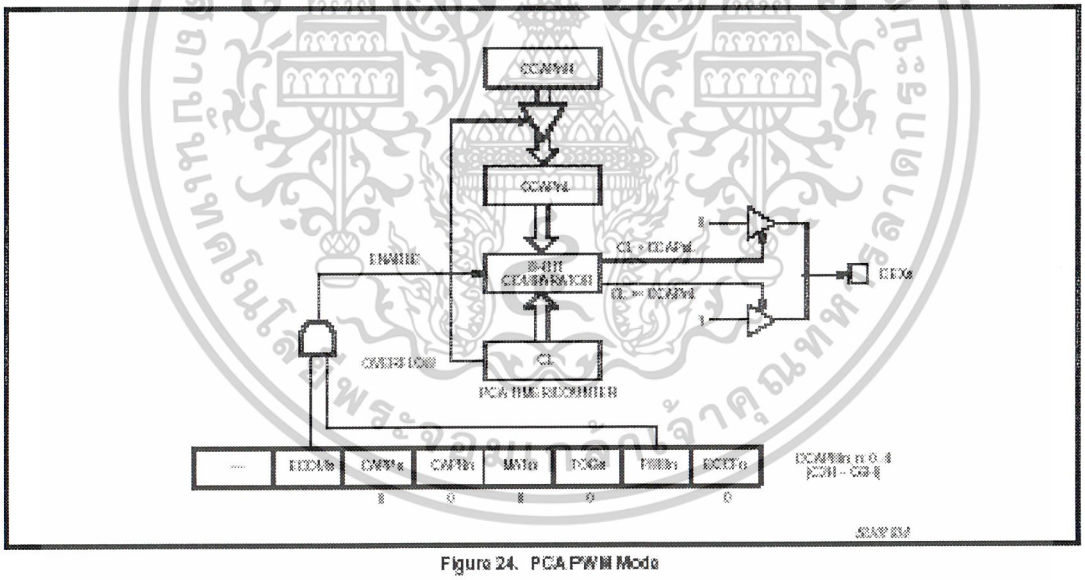


Figure 24. PCA PWM Mode

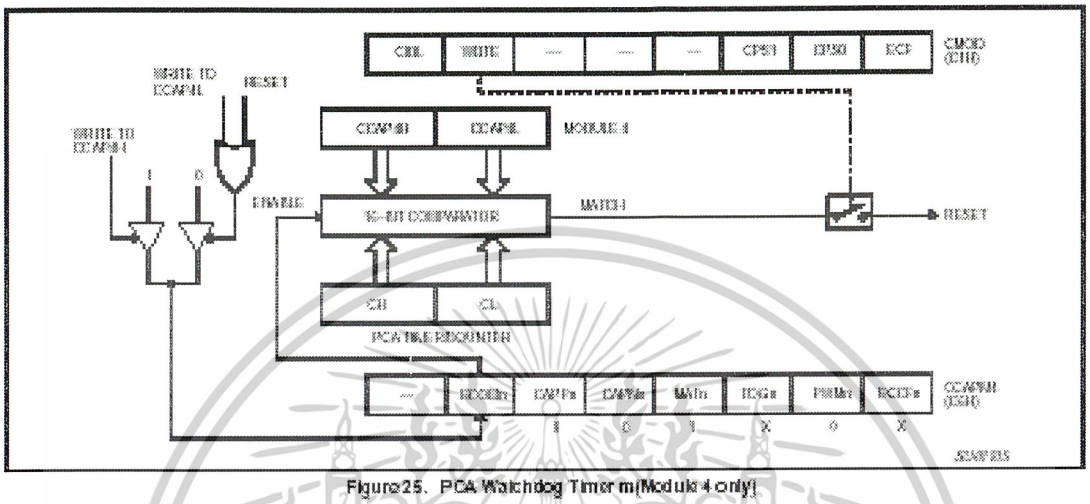


Figure 25. PCA Watchdog Timer in Module 4 only

PCA Watchdog Timer

An on-board watchdog timer is available with the PCA to improve the reliability of the system without increasing chip count. Watchdog timers are useful for systems that are susceptible to noise, power glitches, or electrostatic discharge. Module 4 is the only PCA module that can be programmed as a watchdog. However, this module can still be used for other modes if the watchdog is not needed.

Figure 25 shows a diagram of how the watchdog works. The user pre-loads a 16-bit value in the compare registers. Just like the other compare modes, this 16-bit value is compared to the PCA timer value. If a match is allowed to occur, an internal reset will be generated. This will not cause the RST pin to be driven high.

In order to hold off the reset, the user has three options:

1. periodically change the compare value so it will never match the PCA timer,
2. periodically change the PCA timer value so it will never match the compare values, or
3. disable the watchdog by clearing the WDTE bit before a match occurs and then re-enable it.

The first two options are more suitable because the watchdog timer is never disabled as in option #3. If the program counter ever goes astray, a match will eventually occur and cause an internal reset. The second option is also not recommended if other PCA modules are being used. Remember, the PCA timer is the time base for all modules; changing the time base for other modules would not be a good idea. Thus, in most applications the first solution is the best option.

Figure 26 shows the code for initializing the watchdog timer. Module 4 can be configured in either compare mode, and the WDTE bit in CMOD must also be set. The user's software then must periodically change (CCAP4H,CCAP4L) to keep a match from occurring with the PCA timer (CH,CL). This code is given in the WATCHDOG routine in Figure 26.

This routine should not be part of an interrupt service routine, because if the program counter goes astray and gets stuck in an infinite loop, interrupts will still be serviced and the watchdog will keep getting reset. Thus, the purpose of the watchdog would be defeated. Instead, call this subroutine from the main program within 2^{16} count of the PCA timer.

```
INIT_WATCHDOG:
    MOV CCAPM4, #4CH          ; Module 4 in compare mode
    MOV CCAP4L, #0FFH         ; Write to low byte first
    MOV CCAP4H, #0FFH         ; Before PCA timer counts up to
                              ; FFFF Hex, these compare values
                              ; must be changed
    ORL CMOD, #40H            ; Set the WDTE bit to enable the
                              ; watchdog timer without changing
                              ; the other bits in CMOD
;
;*************************************************************
;
; Main program goes here, but CLR MATCHDOG periodically.
;
;*************************************************************
;
MATCHDOG:
    CLR EA                    ; Hold off interrupts
    MOV CCAP4L, #00           ; Next compare value is within
    MOV CCAP4H, CH            ; 255 counts of the current PCA
    SETB EA                   ; timer value
    RET
```

Figure 25. PCA Watchdog Timer Initialization Code

Expanded Data RAM Addressing

The P89C51RB2/RC2/RD2 has internal data memory that is mapped into four separate segments: the lower 128 bytes of RAM, upper 128 bytes of RAM, 128 bytes Special Function Register (SFR), and 256 bytes expanded RAM (ERAM) (768 bytes for the RD2).

The four segments are:

1. The Lower 128 bytes of RAM (addresses 00H to 7FH) are directly and indirectly addressable.
2. The Upper 128 bytes of RAM (addresses 80H to FFH) are indirectly addressable only.
3. The Special Function Registers, SFRs, (addresses 80H to FFH) are directly addressable only.
4. The 256/768-bytes expanded RAM (ERAM, 00H – 1FFH/2FFH) are indirectly accessed by move external instruction, MOVX, and with the EXTRAM bit cleared. See Figure 27.

The Lower 128 bytes can be accessed by either direct or indirect addressing. The Upper 128 bytes can be accessed by indirect addressing only. The Upper 128 bytes occupy the same address space as the SFR. That means they have the same address, but are physically separate from SFR space.

When an instruction accesses an internal location above address 7FH, the CPU knows whether the access is to the upper 128 bytes of data RAM or to SFR space by the addressing mode used in the instruction. Instructions that use direct addressing access SFR space. For example:

MOV 0A0H,#data

accesses the SFR at location 0A0H (which is P2). Instructions that use indirect addressing access the Upper 128 bytes of data RAM.

For example:

MOV @R0,#data

where R0 contains 0A0H, accesses the data byte at address 0A0H, rather than P2 (whose address is 0A0H).

The ERAM can be accessed by indirect addressing, with EXTRAM bit cleared and MOVX instructions. This part of memory is physically located on-chip, logically occupies the first 7936-bytes of external data memory.

With EXTRAM = 0, the ERAM is indirectly addressed, using the MOVX instruction in combination with any of the registers R0, R1 of the selected bank or DPTR. An access to ERAM will not affect ports P0, P3.6 (WR#) and P3.7 (RD#). P2 SFR is output during external addressing. For example, with EXTRAM = 0,

MOVX @R0,#data

where R0 contains 0A0H, access the ERAM at address 0A0H rather than external memory. An access to external data memory locations higher than the ERAM will be performed with the MOVX DPTR instructions in the same way as in the standard 80C51, so with P0 and P2 as data/address bus, and P3.6 and P3.7 as write and read timing signals. Refer to Figure 28.

With EXTRAM = 1, MOVX @Ri and MOVX @DPTR will be similar to the standard 80C51. MOVX @Ri will provide an 8-bit address multiplexed with data on Port 0 and any output port pins can be used to output higher order address bits. This is to provide the external paging capability. MOVX @DPTR will generate a 16-bit address. Port 2 outputs the high-order eight address bits (the contents of DPH), while Port 0 multiplexes the low-order eight address bits (DPL) with data. MOVX @Ri and MOVX @DPTR will generate either read or write signals on P3.6 (WR) and P3.7 (RD).

The stack pointer (SP) may be located anywhere in the 256 bytes RAM (lower and upper RAM) internal data memory. The stack may not be located in the ERAM.

AUXR Address = 8EH Reset Value = xxxx xx0Bb

Not Bit Addressable

—	—	—	—	—	—	EXTRAM	AO
7	6	5	4	3	2	1	0

Bit

Symbol	Function
AO	Disable/Enable ALE AO — Operating Mode 0 — ALE is emitted at a constant rate of 1/6 the oscillator frequency (6 clock mode, 1/6 f_{OSC} in 12 clock mode). 1 — ALE is active only during a MOVX or MOVC instruction.
EXTRAM	Internal/External RAM access using MOVX @Ri/@DPTR EXTRAM — Operating Mode 0 — Internal ERAM access using MOVX @Ri/@DPTR 1 — External data memory access.
—	Not implemented, reserved for future use*.

NOTE:
*User software should not write 1s to reserved bits. These bits may be used in future 80C51 family products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1. The value read from a reserved bit is indeterminate.

SU00120

Figure 27. AUXR: Auxiliary Register

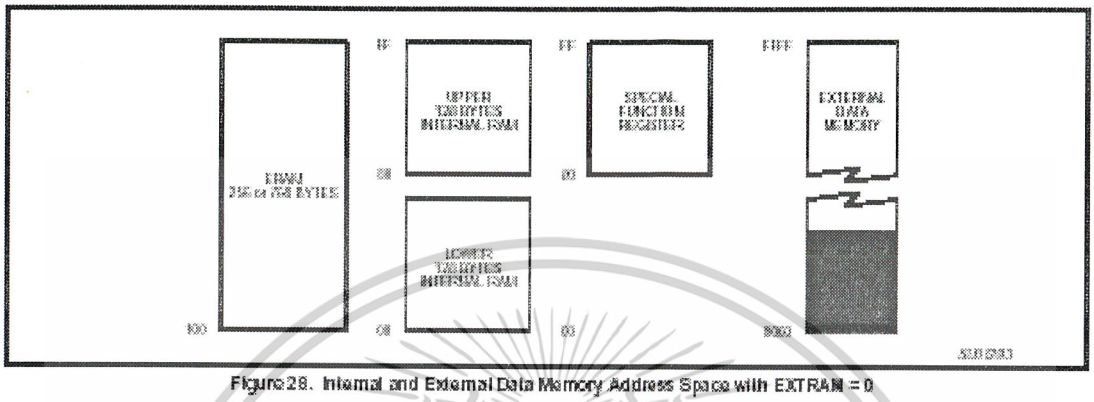


Figure 28. Internal and External Data Memory Address Space with EXTRAM = 0

HARDWARE WATCHDOG TIMER (ONE-TIME ENABLED WITH RESET-OUT FOR P89C51RB2/RC2/RD2)

The WDT is intended as a recovery method in situations where the CPU may be subjected to software upset. The WDT consists of a 14-bit counter and the WatchDog Timer reset (WDTRST) SFR. The WDT is disabled at reset. To enable the WDT, user must write 01EH and 0E1H in sequence to the WDTRST, SFR location 0A6H. When WDT is enabled, it will increment every machine cycle while the oscillator is running and there is no way to disable the WDT except through reset (either hardware reset or WDT overflow reset). When WDT overflows, it will drive an output reset HIGH pulse at the RST-pin (see the note below).

Using the WDT

To enable the WDT, user must write 01EH and 0E1H in sequence to the WDTRST, SFR location 0A6H. When WDT is enabled, the user needs to service it by writing to 01EH and 0E1H to WDTRST to avoid WDT overflow. The 14-bit counter overflows when it reaches 16383 (3FFFH), and this will reset the device. When WDT is enabled, it will increment every machine cycle while the oscillator is running. This means the user must reset the WDT at least every 16383 machine cycles. To reset the WDT, the user must write 01EH and 0E1H to WDTRST. WDTRST is a write only register. The WDT counter cannot be read or written. When WDT overflows, it will generate an output RESET pulse at the reset pin (see note below). The RESET pulse duration is $98 \times T_{OSC}$ (6 clock mode, 196 in 12 clock mode), where $T_{OSC} = 1/f_{OSC}$. To make the best use of the WDT, it should be serviced in those sections of code that will periodically be executed within the time required to prevent a WDT reset.

ABSOLUTE MAXIMUM RATINGS[1, 2, 3]

PARAMETER	RATING	UNIT
Operating temperature under bias	0 to +70 or −40 to +85	°C
Storage temperature range	−65 to +150	°C
Voltage on $\overline{EA}$/V_{PP} pin to V_{SS}	0 to +13.0	V
Voltage on any other pin to V_{SS}	−0.5 to +6.5	V
Maximum I_{OL} per I/O pin	15	mA
Power dissipation (based on package heat transfer limitations, not device power consumption)	1.5	W

NOTES:
1. Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any conditions other than those described in the AC and DC Electrical Characteristics section of this specification is not implied.
2. This product includes circuitry specifically designed for the protection of its internal devices from the damaging effects of excessive static charge. Nonetheless, it is suggested that conventional precautions be taken to avoid applying greater than the rated maximum.
3. Parameters are valid over operating temperature range unless otherwise specified. All voltages are with respect to V_{SS} unless otherwise noted.

DC ELECTRICAL CHARACTERISTICS

T_{amb} = 0°C to +70°C or −40°C to +85°C; 5 V ± 10%; V_{SS} = 0 V

SYMBOL	PARAMETER	TEST CONDITIONS	LIMITS			UNIT
			MIN	TYP[1]	MAX	
V_{IL}	Input low voltage	4.5 V < V_{CC} < 5.5 V	−0.5		$0.2V_{CC}$−0.1	V
V_{IH}	Input high voltage (ports 0, 1, 2, 3, $\overline{EA}$)		$0.2V_{CC}$+0.9		V_{CC}+0.5	V
V_{IH1}	Input high voltage, XTAL1, RST		$0.7V_{CC}$		V_{CC}+0.5	V
V_{OL}	Output low voltage, ports 1, 2, 3[2]	V_{CC} = 4.5 V; I_{OL} = 1.6 mA[2]			0.4	V
V_{OL1}	Output low voltage, port 0, ALE, $\overline{PSEN}$[7, 8]	V_{CC} = 4.5 V; I_{OL} = 3.2 mA[2]			0.4	V
V_{OH}	Output high voltage, ports 1, 2, 3[3]	V_{CC} = 4.5 V; I_{OH} = −30 µA	V_{CC} − 0.7			V
V_{OH1}	Output high voltage (port 0 in external bus mode), ALE[9], $\overline{PSEN}$[9]	V_{CC} = 4.5 V; I_{OH} = −3.2 mA	V_{CC} − 0.7			V
I_{IL}	Logical 0 input current, ports 1, 2, 3	V_{IN} = 0.4 V	−1		−75	µA
I_{TL}	Logical 1-to-0 transition current, ports 1, 2, 3[4]	V_{IN} = 2.0 V See Note 4			−650	µA
I_{LI}	Input leakage current, port 0	0.45 < V_{IN} < V_{CC} − 0.3			±10	µA
I_{CC}	Power supply current (see Figure 36): Active mode (see Note 5) Idle mode (see Note 5) Power-down mode or clock stopped (see Figure 42 for conditions)	See Note 5 T_{amb} = 0°C to 70°C T_{amb} = −40°C to +85°C		× 1	40 50	µA µA
R_{RST}	Internal reset pull-down resistor		40		225	kΩ
C_{IO}	Pin capacitance[10] (except $\overline{EA}$)				15	pF

NOTES:

1. Typical ratings are not guaranteed. The values listed are at room temperature, 5 V.
2. Capacitive loading on ports 0 and 2 may cause spurious noise to be superimposed on the V_{OL} s of ALE and ports 1 and 3. The noise is due to external bus capacitance discharging into the port 0 and port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE pin may exceed 0.8 V. In such cases, it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input. I_{OL} can exceed these conditions provided that no single output sinks more than 5 mA and no more than two outputs exceed the test conditions.
3. Capacitive loading on ports 0 and 2 may cause the V_{OH} on ALE and $\overline{PSEN}$ to momentarily fall below the V_{CC}−0.7 specification when the address bits are stabilizing.
4. Pins of ports 1, 2 and 3 source a transition current when they are being externally driven from 1 to 0. The transition current reaches its maximum value when V_{IN} is approximately 2 V.
5. See Figures 39 through 42 for I_{CC} test conditions and Figure 38 for I_{CC} vs. Freq.
   Active mode: $I_{CC(MAX)}$ = (1.8 × FREQ. + 20)mA for all devices in 6 clock mode; (0.9 × FREQ. + 20)mA in 12 clock mode.
   Idle mode: $I_{CC(MAX)}$ = (0.7 × FREQ. +1.0)mA in 6 clock mode; (0.35 × FREQ. +1.0)mA in 12 clock mode.
6. This value applies to T_{amb} = 0°C to +70°C.
7. Load capacitance for port 0, ALE, and $\overline{PSEN}$ = 100 pF, load capacitance for all other outputs = 80 pF.
8. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 15 mA (NOTE: This is 25°C specification)
   Maximum I_{OL} per 8-bit port: 26 mA
   Maximum total I_{OL} for all outputs: 71 mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
9. ALE is tested to V_{OH1}, except when ALE is off then V_{OH} is the voltage specification.
10. Pin capacitance is characterized but not tested. Pin capacitance is less than 25 pF. Pin capacitance of ceramic package is less than 15 pF (except $\overline{EA}$ is 25 pF).

AC ELECTRICAL CHARACTERISTICS (6 CLOCK MODE)

T_{amb} = 0°C to +70°C or −40°C to +85°C, V_{CC} = 5 V ±10%, V_{SS} = 0V[1, 2, 3]

SYMBOL	FIGURE	PARAMETER	VARIABLE CLOCK[4]		20 MHz CLOCK[4]		UNIT
			MIN	MAX	MIN	MAX	
$1/t_{CLCL}$	29	Oscillator frequency	0	20	0	20	MHz
t_{LHLL}	29	ALE pulse width	$t_{CLCL}-40$		10		ns
t_{AVLL}	29	Address valid to ALE low	$0.5t_{CLCL}-20$		5		ns
t_{LLAX}	29	Address hold after ALE low	$0.5t_{CLCL}-20$		5		ns
t_{LLIV}	29	ALE low to valid instruction in		$2t_{CLCL}-65$		35	ns
t_{LLPL}	29	ALE low to PSEN low	$0.5t_{CLCL}-20$		5		ns
t_{PLPH}	29	PSEN pulse width	$1.5t_{CLCL}-45$		30		ns
t_{PLIV}	29	PSEN low to valid instruction in		$1.5t_{CLCL}-60$		15	ns
t_{PXIX}	29	Input instruction hold after PSEN	0		0		ns
t_{PXIZ}	29	Input instruction float after PSEN		$0.5t_{CLCL}-20$		5	ns
t_{AVIV}	29	Address to valid instruction in		$2.5t_{CLCL}-80$		45	ns
t_{PLAZ}	29	PSEN low to address float		10		10	ns
Data Memory							
t_{RLRH}	30, 31	RD pulse width	$3t_{CLCL}-100$		50		ns
t_{WLWH}	30, 31	WR pulse width	$3t_{CLCL}-100$		50		ns
t_{RLDV}	30, 31	RD low to valid data in		$2.5t_{CLCL}-90$		35	ns
t_{RHDX}	30, 31	Data hold after RD	0		0		ns
t_{RHDZ}	30, 31	Data float after RD		$t_{CLCL}-20$		5	ns
t_{LLDV}	30, 31	ALE low to valid data in		$4t_{CLCL}-150$		50	ns
t_{AVDV}	30, 31	Address to valid data in		$4.5t_{CLCL}-185$		60	ns
t_{LLWL}	30, 31	ALE low to RD or WR low	$1.5t_{CLCL}-50$	$1.5t_{CLCL}+50$	25	125	ns
t_{AVWL}	30, 31	Address valid to WR low or RD low	$2t_{CLCL}-75$		25		ns
t_{QVWX}	30, 31	Data valid to WR transition	$0.5t_{CLCL}-25$		0		ns
t_{WHQX}	30, 31	Data hold after WR	$0.5t_{CLCL}-20$		5		ns
t_{QVWH}	31	Data valid to WR high	$3.5t_{CLCL}-150$		45		ns
t_{RLAZ}	30, 31	RD low to address float		0		0	ns
t_{WHLH}	30, 31	RD or WR high to ALE high	$0.5t_{CLCL}-20$	$0.5t_{CLCL}+20$	5	45	ns
External Clock							
t_{CHCX}	33	High time	20	$t_{CLCL}-t_{CLCX}$			ns
t_{CLCX}	33	Low time	20	$t_{CLCL}-t_{CHCX}$			ns
t_{CLCH}	33	Rise time		5			ns
t_{CHCL}	33	Fall time		5			ns
Shift Register							
t_{XLXL}	32	Serial port clock cycle time	$8t_{CLCL}$		300		ns
t_{QVXH}	32	Output data setup to clock rising edge	$9t_{CLCL}-133$		117		ns
t_{XHQX}	32	Output data hold after clock rising edge	$t_{CLCL}-30$		20		ns
t_{XHDX}	32	Input data hold after clock rising edge	0		0		ns
t_{XHDV}	32	Clock rising edge to input data valid		$8t_{CLCL}-133$		117	ns

NOTES:
1. Parameters are valid over operating temperature range unless otherwise specified.
2. Load capacitance for port 0, ALE, and PSEN = 100 pF; load capacitance for all other outputs = 80 pF.
3. Interfacing the microcontroller to devices with float times up to 45 ns is permitted. This limited bus contention will not cause damage to Port 0 drivers.
4. Parts are tested to 2 MHz but are guaranteed to operate down to 0 Hz.

AC ELECTRICAL CHARACTERISTICS (12 CLOCK MODE)

T_{amb} = 0°C to +70°C or −40°C to +85°C, V_{CC} = 5 V ±10%, V_{SS} = 0 V[1, 2, 3]

SYMBOL	FIGURE	PARAMETER	VARIABLE CLOCK[4]		33 MHz CLOCK[4]		UNIT
			MIN	MAX	MIN	MAX	
$1/t_{CLCL}$	29	Oscillator frequency	0	33	0	33	MHz
t_{LHLL}	29	ALE pulse width	$2t_{CLCL}-40$		21		ns
t_{AVLL}	29	Address valid to ALE low	$t_{CLCL}-25$		5		ns
t_{LLAX}	29	Address hold after ALE low	$t_{CLCL}-25$		5		ns
t_{LLIV}	29	ALE low to valid instruction in		$4t_{CLCL}-65$		55	ns
t_{LLPL}	29	ALE low to PSEN low	$t_{CLCL}-25$		5		ns
t_{PLPH}	29	PSEN pulse width	$3t_{CLCL}-45$		45		ns
t_{PLIV}	29	PSEN low to valid instruction in		$3t_{CLCL}-60$		30	ns
t_{PXIX}	29	Input instruction hold after PSEN	0		0		ns
t_{PXIZ}	29	Input instruction float after PSEN		$t_{CLCL}-25$		5	ns
t_{AVIV}	29	Address to valid instruction in		$5t_{CLCL}-80$		70	ns
t_{PLAZ}	29	PSEN low to address float		10		10	ns
Data Memory							
t_{RLRH}	30, 31	RD pulse width	$6t_{CLCL}-100$		82		ns
t_{WLWH}	30, 31	WR pulse width	$6t_{CLCL}-100$		82		ns
t_{RLDV}	30, 31	RD low to valid data in		$5t_{CLCL}-90$		60	ns
t_{RHDX}	30, 31	Data hold after RD	0		0		ns
t_{RHDZ}	30, 31	Data float after RD		$2t_{CLCL}-28$		32	ns
t_{LLDV}	30, 31	ALE low to valid data in		$8t_{CLCL}-150$		90	ns
t_{AVDV}	30, 31	Address to valid data in		$9t_{CLCL}-165$		105	ns
t_{LLWL}	30, 31	ALE low to RD or WR low	$3t_{CLCL}-50$	$3t_{CLCL}+50$	40	140	ns
t_{AVWL}	30, 31	Address valid to WR low or RD low	$4t_{CLCL}-75$		45		ns
t_{QVWX}	30, 31	Data valid to WR transition	$t_{CLCL}-30$		0		ns
t_{WHQX}	30, 31	Data hold after WR	$t_{CLCL}-25$		5		ns
t_{QVWH}	31	Data valid to WR high	$7t_{CLCL}-130$		80		ns
t_{RLAZ}	30, 31	RD low to address float		0		0	ns
t_{WHLH}	30, 31	RD or WR high to ALE high	$t_{CLCL}-25$	$t_{CLCL}+25$	5	55	ns
External Clock							
t_{CHCX}	33	High time	17	$t_{CLCL}-t_{CLCX}$			ns
t_{CLCX}	33	Low time	17	$t_{CLCL}-t_{CHCX}$			ns
t_{CLCH}	33	Rise time		5			ns
t_{CHCL}	33	Fall time		5			ns
Shift Register							
t_{XLXL}	32	Serial port clock cycle time	$12t_{CLCL}$		360		ns
t_{QVXH}	32	Output data setup to clock rising edge	$10t_{CLCL}-133$		167		ns
t_{XHQX}	32	Output data hold after clock rising edge	$2t_{CLCL}-80$		50		ns
t_{XHDX}	32	Input data hold after clock rising edge	0		0		ns
t_{XHDV}	32	Clock rising edge to input data valid		$10t_{CLCL}-133$		167	ns

NOTES:
1. Parameters are valid over operating temperature range unless otherwise specified.
2. Load capacitance for port 0, ALE, and PSEN = 100 pF, load capacitance for all other outputs = 80 pF.
3. Interfacing the microcontroller to devices with float times up to 45 ns is permitted. This limited bus contention will not cause damage to Port 0 drivers.
4. Parts are tested to 33 MHz, but guaranteed to operate down to 0 Hz.

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has five characters. The first character is always 't' (= time). The other characters, depending on their positions, indicate the name of a signal or the logical status of that signal. The designations are:

A – Address
C – Clock
D – Input data
H – Logic level high
I – Instruction (program memory contents)
L – Logic level low, or ALE
P – PSEN
Q – Output data
R – RD signal
t – Time
V – Valid
W – WR signal
X – No longer a valid logic level
Z – Float

Examples: t_{AVLL} = Time for address valid to ALE low.
t_{LLPL} = Time for ALE low to PSEN low.

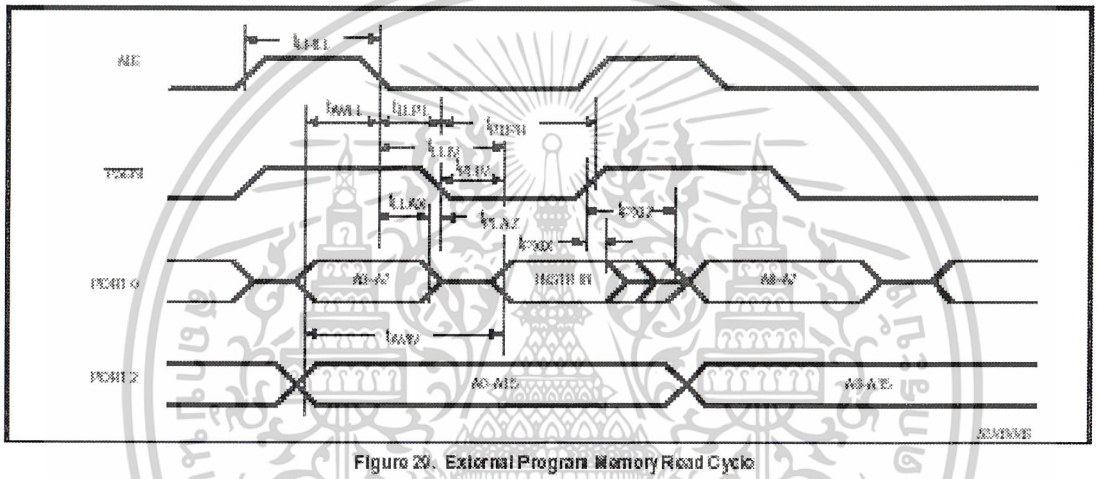


Figure 29. External Program Memory Read Cycle

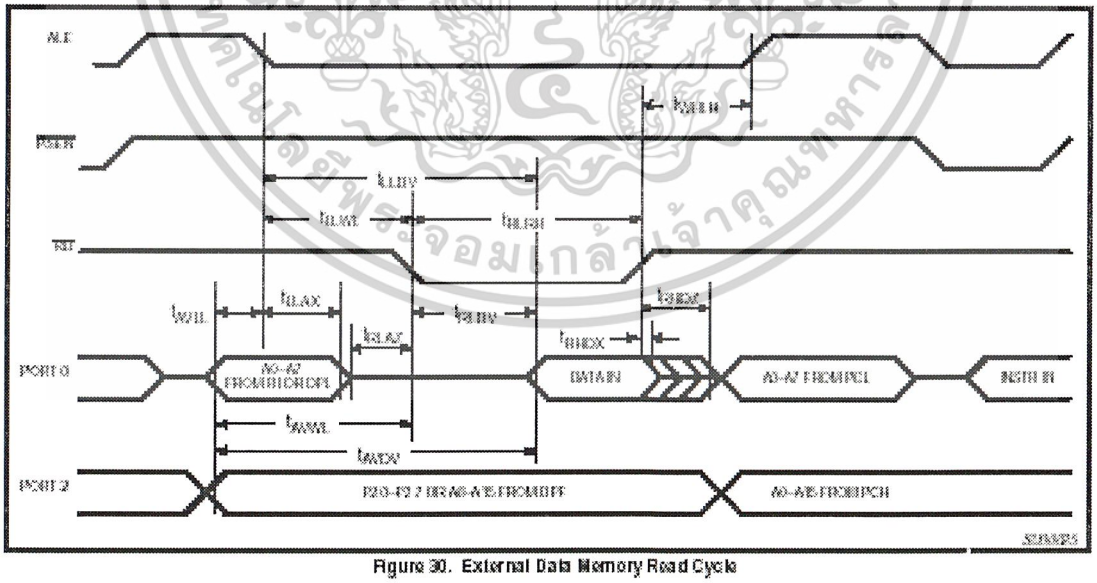


Figure 30. External Data Memory Read Cycle

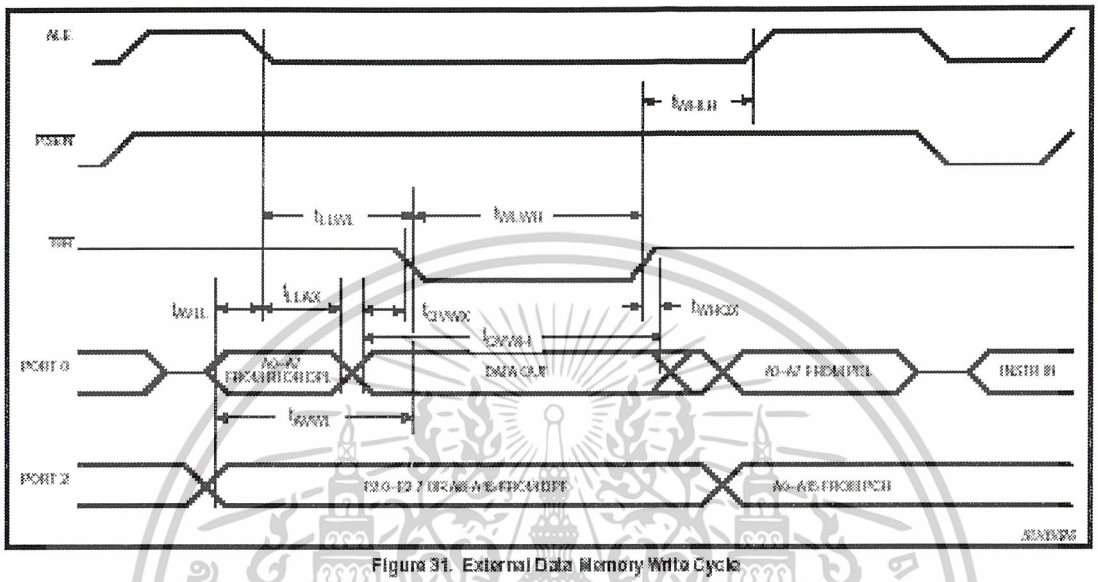


Figure 31. External Data Memory Write Cycle

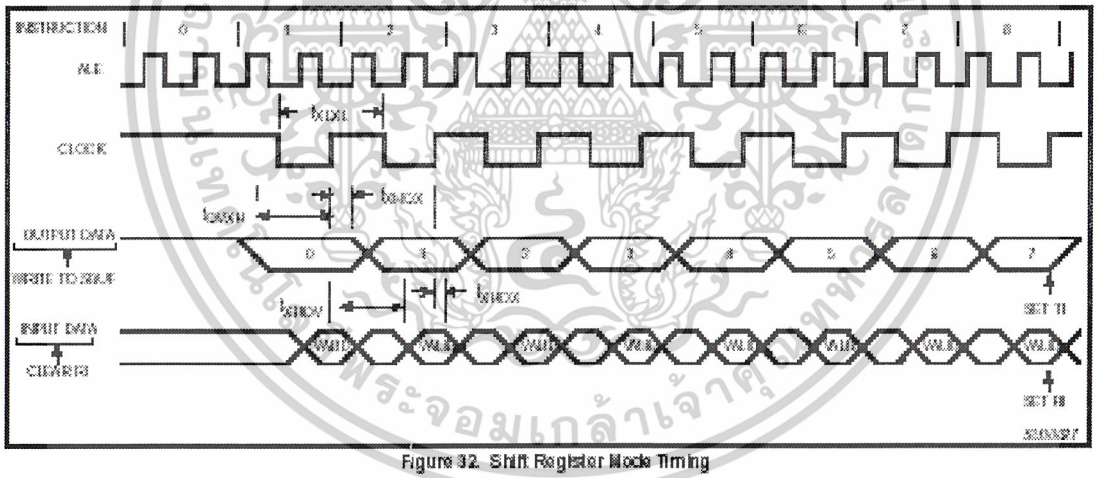


Figure 32. Shift Register Mode Timing

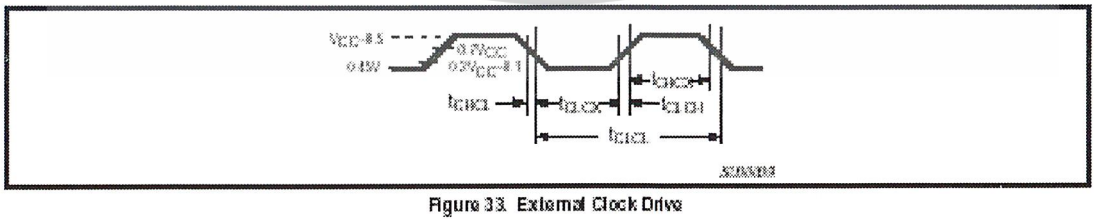


Figure 33. External Clock Drive

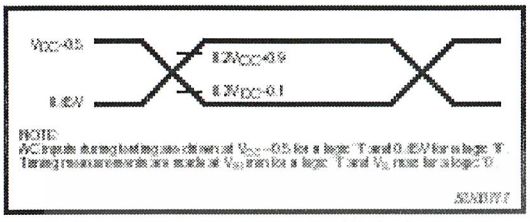


Figure 34. AC Testing Input/Output

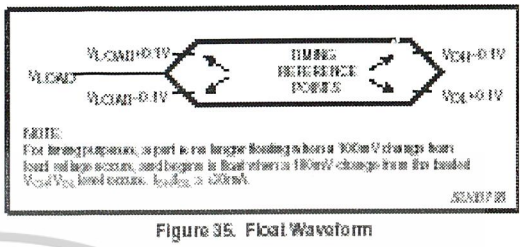


Figure 35. Float Waveform

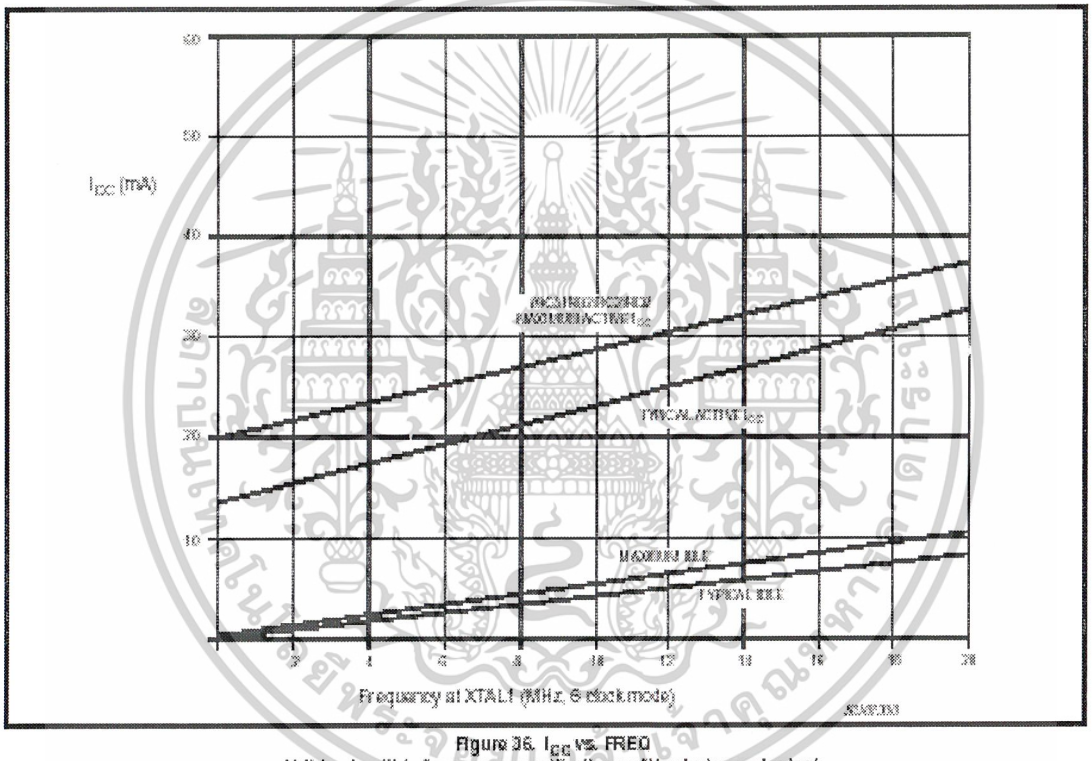


Figure 36. I_{CC} vs. FREQ
Valid only within frequency specifications of the device under test

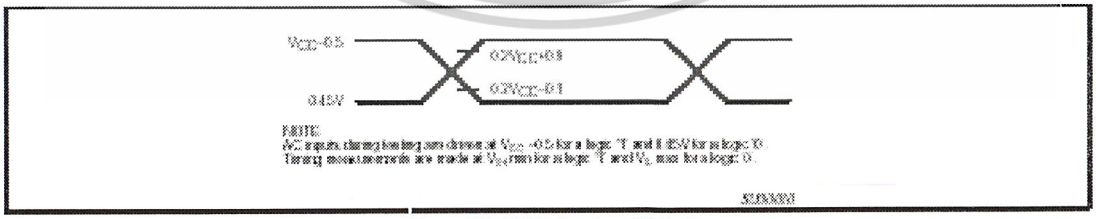


Figure 37. AC Testing Input/Output

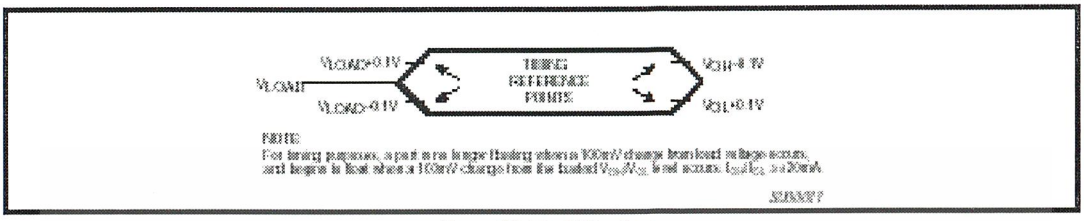


Figure 38. Float Waveform

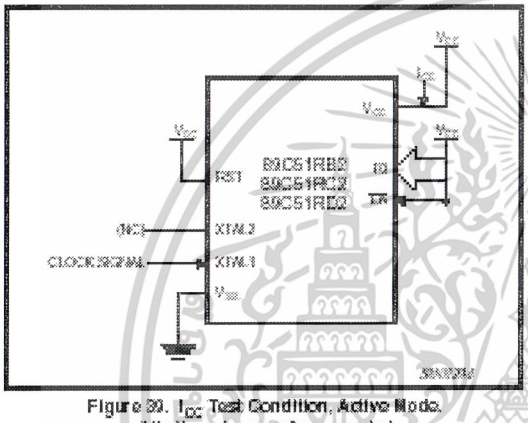


Figure 39. I_{CC} Test Condition, Active Mode.
All other pins are disconnected

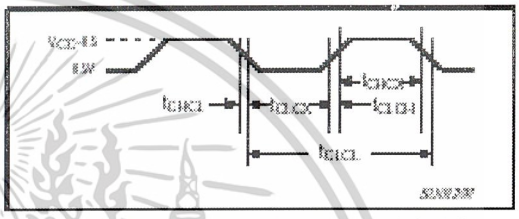


Figure 41. Clock Signal Waveform for I_{CC} Tests in Active and Idle Modes.
$t_{CLCL} = t_{CHCL} = 10$ ns

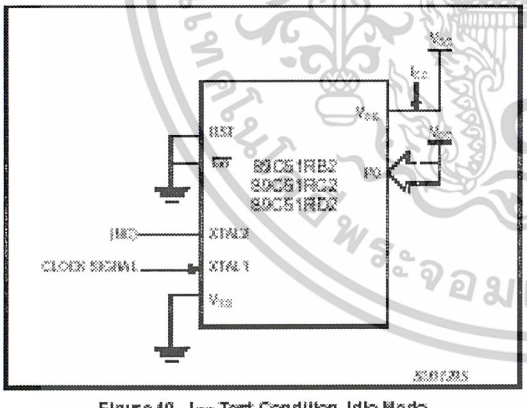


Figure 40. I_{CC} Test Condition, Idle Mode.
All other pins are disconnected

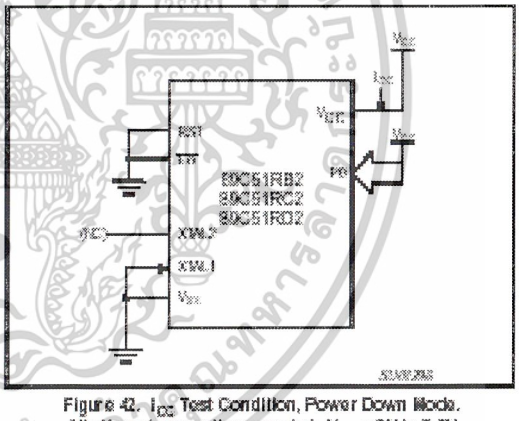


Figure 42. I_{CC} Test Condition, Power Down Mode.
All other pins are disconnected; V_{CC} = 2V to 5.5V

FLASH EPROM MEMORY

GENERAL DESCRIPTION

The P89C51RB2/RC2/RD2 Flash memory augments EPROM functionality with in-circuit electrical erasure and programming. The Flash can be read and written as bytes. The Chip Erase operation will erase the entire program memory. The Block Erase function can erase any Flash block. In-system programming and standard parallel programming are both available. On-chip erase and write timing generation contribute to a user friendly programming interface.

The P89C51RB2/RC2/RD2 Flash reliably stores memory contents even after 10,000 erase and program cycles. The cell is designed to optimize the erase and programming mechanisms. In addition, the combination of advanced tunnel oxide processing and low internal electric fields for erase and programming operations produces reliable cycling. The P89C51RB2/RC2/RD2 uses a +5 V V_{PP} supply to perform the Program/Erase algorithms.

FEATURES

- Flash EPROM internal program memory with Block Erase.
- Internal 1 kB boot ROM, containing low-level in-system programming routines and a default serial loader. User program can call these routines to perform In-Application Programming (IAP). The Boot ROM can be turned off to provide access to the full 64 kB Flash memory.
- Boot vector allows user provided Flash loader code to reside anywhere in the Flash memory space. This configuration provides flexibility to the user.
- Default loader in Boot ROM allows programming via the serial port without the need for a user provided loader.
- Up to 64 kB external program memory if the internal program memory is disabled ($\overline{EA}$ = 0).
- Programming and erase voltage +5 V (+12 V tolerant).
- Read/Programming/Erase:
  - Byte-wise read (100 ns access time).
  - Byte Programming (20 µs).
  - Typical erase times:
    Block Erase (8 kB or 16 kB) in 3 seconds.
    Full Erase (64 kB) in 3 seconds.
- Parallel programming with 87C51 compatible hardware interface to programmer.
- In-system programming.
- Programmable security for the code in the Flash.
- 10,000 minimum erase/program cycles for each byte.
- 10-year minimum data retention.

CAPABILITIES OF THE PHILIPS 89C51 FLASH-BASED MICROCONTROLLERS

Flash organization

The P89C51RB2/RC2/RD2 contains 16KB/32KB/64K bytes of Flash program memory. This memory is organized as 5 separate blocks. The first two blocks are 8 kB in size, filling the program memory space from address 0 through 3FFF hex. The final three blocks are 16 kB in size and occupy addresses from 4000 through FFFF hex.

Figure 43 depicts the Flash memory configurations.

Flash Programming and Erasure

There are three methods of erasing or programming of the Flash memory that may be used. First, the Flash may be programmed or erased in the end-user application by calling low-level routines through a common entry point in the Boot ROM. The end-user application, though, must be executing code from a different block than the block that is being erased or programmed. Second, the on-chip ISP boot loader may be invoked. This ISP boot loader will, in turn, call low-level routines through the same common entry point in the Boot ROM that can be used by the end-user application. Third, the Flash may be programmed or erased using the parallel method by using a commercially available EPROM programmer. The parallel programming method used by these devices is similar to that used by EPROM 87C51, but it is not identical, and the commercially available programmer will need to have support for these devices.

Boot ROM

When the microcontroller programs its own Flash memory, all of the low level details are handled by code that is permanently contained in a 1 kB Boot ROM that is separate from the Flash memory. A user program simply calls the common entry point with appropriate parameters in the Boot ROM to accomplish the desired operation. Boot ROM operations include things like: erase block, program byte, verify byte, program security lock bit, etc. The Boot ROM overlays the program memory space at the top of the address space from F000 to FFFF hex, when it is enabled. The Boot ROM may be turned off so that the upper 1 kB of Flash program memory are accessible for execution.

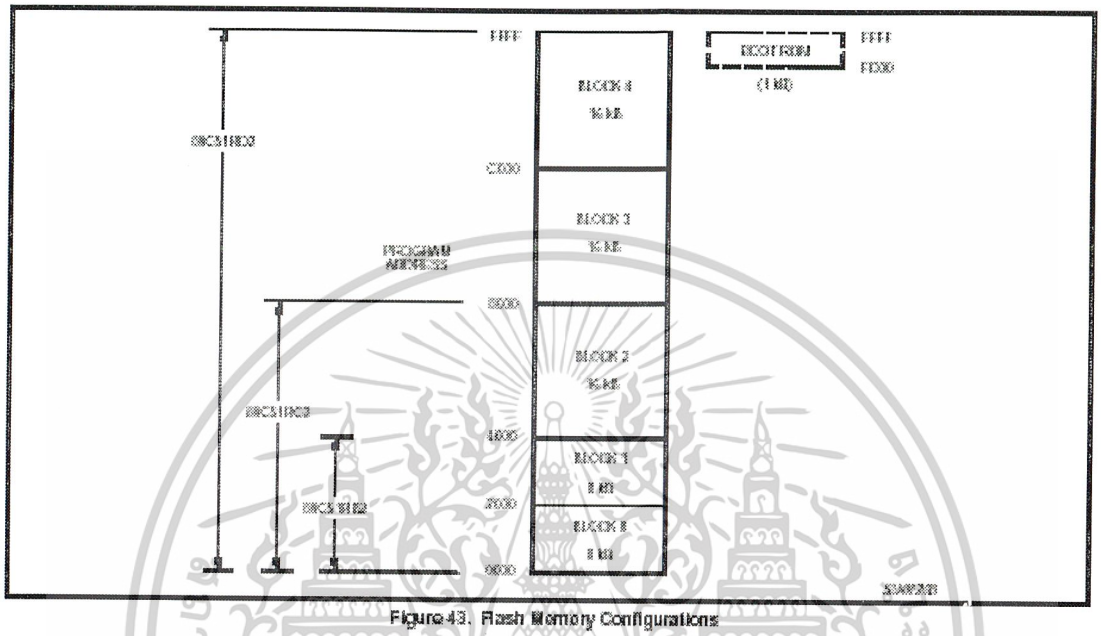


Figure 43. Flash Memory Configurations

Power-On Reset Code Execution

The P89C51RB2/RC2/RD2 contains two special Flash registers: the BOOT VECTOR and the STATUS BYTE. At the falling edge of reset, the P89C51RB2/RC2/RD2 examines the contents of the Status Byte. If the Status Byte is set to zero, power-up execution starts at location 0000H, which is the normal start address of the user's application code. When the Status Byte is set to a value other than zero, the contents of the Boot Vector is used as the high byte of the execution address and the low byte is set to 00H. The factory default setting is 0FCH, corresponds to the address 0FC00H for the factory masked-ROM ISP boot loader. A custom boot loader can be written with the Boot Vector set to the custom boot loader.

NOTE: When erasing the Status Byte or Boot Vector, both bytes are erased at the same time. It is necessary to reprogram the Boot Vector after erasing and updating the Status Byte.

Hardware Activation of the Boot Loader

The boot loader can also be executed by holding PSEN LOW, EA greater than V_{HH} (such as +5 V), and ALE HIGH (or not connected) at the falling edge of RESET. This is the same effect as having a non-zero status byte. This allows an application to be built that will normally execute the end user's code but can be manually forced into ISP operation.

If the factory default setting for the Boot Vector (0FCH) is changed, it will no longer point to the ISP masked-ROM boot loader code. If this happens, the only way it is possible to change the contents of the Boot Vector is through the parallel programming method, provided that the end user application does not contain a customized loader that provides for erasing and reprogramming of the Boot Vector and Status Byte.

After programming the Flash, the status byte should be programmed to zero in order to allow execution of the user's application code beginning at address 0000H.

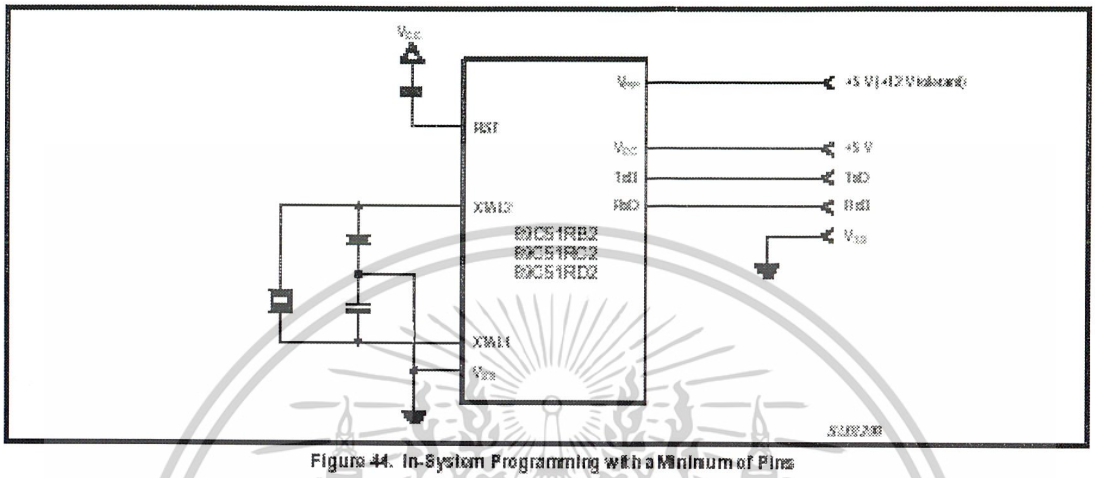


Figure 44. In-System Programming with a Minimum of Pins

In-System Programming (ISP)

The In-System Programming (ISP) is performed without removing the microcontroller from the system. The In-System Programming (ISP) facility consists of a series of internal hardware resources coupled with internal firmware to facilitate remote programming of the P89C51RB2/RC2/RD2 through the serial port. This firmware is provided by Philips and embedded within each P89C51RB2/RC2/RD2 device.

The Philips In-System Programming (ISP) facility has made in-circuit programming in an embedded application possible with a minimum of additional expense in components and circuit board area.

The ISP function uses five pins: TxD, RxD, V_{SS}, V_{CC}, and V_{PP} (See Figure 44). Only a serial connector needs to be available to interface your application to an external circuit in order to use this feature. The V_{PP} supply should be adequately decoupled and V_{PP} not allowed to exceed datasheet limits.

Using the In-System Programming (ISP)

The ISP feature allows for a wide range of baud rates to be used in your application, independent of the oscillator frequency. It is also adaptable to a wide range of oscillator frequencies. This is accomplished by measuring the bit-time of a single bit in a received character. This information is then used to program the baud rate in terms of timer counts based on the oscillator frequency. The ISP feature requires that an initial character (an uppercase U) be sent to the P89C51RB2/RC2/RD2 to establish the baud rate. The ISP firmware provides auto-echo of received characters.

Once baud rate initialization has been performed, the ISP firmware will only accept Intel Hex-type records. Intel Hex records consist of ASCII characters used to represent hexadecimal values and are summarized below:

:NNAAAARRDD..DDCC<crlf>

In the Intel Hex record, the "NN" represents the number of data bytes in the record. The P89C51RB2/RC2/RD2 will accept up to 16 (10H) data bytes. The "AAAA" string represents the address of the first byte in the record. If there are zero bytes in the record, this field is often set to 0000. The "RR" string indicates the record type. A record type of "00" is a data record. A record type of "01" indicates the end-of-file mark. In this application, additional record types will be added to indicate either commands or data for the ISP facility. The maximum number of data bytes in a record is limited to 16 (decimal). ISP commands are summarized in Table 8.

As a record is received by the P89C51RB2/RC2/RD2, the information in the record is stored internally and a checksum calculation is performed. The operation indicated by the record type is not performed until the entire record has been received. Should an error occur in the checksum, the P89C51RB2/RC2/RD2 will send an "X" out the serial port indicating a checksum error. If the checksum calculation is found to match the checksum in the record, then the command will be executed. In most cases, successful reception of the record will be indicated by transmitting a "." character out the serial port (displaying the contents of the internal program memory is an exception).

In the case of a Data Record (record type 00), an additional check is made. A "." character will NOT be sent unless the record checksum matched the calculated checksum and all of the bytes in the record were successfully programmed. For a data record, an "X" indicates that the checksum failed to match, and an "R" character indicates that one of the bytes did not properly program. It is necessary to send a type 02 record (specify oscillator frequency) to the P89C51RB2/RC2/RD2 before programming data.

The ISP facility was designed so that specific crystal frequencies were not required in order to generate baud rates or time the programming pulses. The user thus needs to provide the P89C51RB2/RC2/RD2 with information required to generate the proper timing. Record type 02 is provided for this purpose.

WinISP, a software utility to implement ISP programming with a PC, is available from Philips. Commercial serial ISP programmers are available from third parties. Please check the Philips web site (www.semiconductors.philips.com) for additional information.

Table 8. Intel-Hex Records Used by In-System Programming

RECORD TYPE	COMMAND/DATA FUNCTION
00	Program Data :nnaaaa00dd....ddcc Where: nn = number of bytes (hex) in record aaaa = memory address of first byte in record dd....dd = data bytes cc = checksum Example: :1000[illegible]
01	End of File (EOF), no operation :xxxxxx01cc Where: xxxxxx = required fields, but value is a "don't care" cc = checksum Example: :[illegible]
02	Sparsly Oscillator Frequency :xxxxxx02yyyycc Where: xxxxxx = required fields, but value is a "don't care" yyyy = integer oscillator frequency command used to nearest MHz cc = checksum Example: :[illegible]

RECORD TYPE	COMMAND/DATA FUNCTION
03	Miscellaneous Write Functions :nnxxxx03ffaabbcc Where: nn = number of bytes (hex) in record xxxx = required field, but value is a "don't care" 03 = Write Function ff = subfunction code aa = selection code dd = data input (as needed) cc = checksum Subfunction Code = 01 (Erase Blocks) ff = 01 aa = block code as shown below: block 5, 0k to 4k, 00H block 1, 4k to 8k, 20H block 2, 16k to 32k, 40H block 3, 32k to 64k, 80H block 4, 48k to 64k, C0H Example: :020000030104F7 erase block 4 Subfunction Code = 04 (Erase Boot Vector and Status Byte) ff = 04 aa = don't care Example: :020000030407F7 erase boot vector and status byte Subfunction Code = 05 (Program Security Bits) ff = 05 aa = 00 program security bit 1 (disable writing to Flash) 01 program security bit 2 (disable Flash verify) 02 program security bit 3 (disable external memory) Example: :020000030501F5 program security bit 2 Subfunction Code = 06 (Program Status Byte or Boot Vector) ff = 06 aa = 00 program status byte 01 program boot vector Example: :020000030601F4 program boot vector with 01H Subfunction Code = 07 (Full Chip Erase) Erases all blocks, security bits, and sets status and boot vector to default values ff = 07 aa = don't care dd = don't care Example: :020000030700F2 full chip erase
04	Display Device Data or Blank Check – Record type 04 causes the contents of the entire Flash array to be sent out the serial port in a formatted display. This display consists of an address and the contents of 16 bytes starting with that address. No display of the device contents will occur if security bit 2 has been programmed. Data to the serial port is initiated by the reception of any character and terminated by the reception of any character. General Format of Function 04 :nnxxxx04ssssmmffcc Where: nn = number of bytes (hex) in record xxxx = required field, but value is a "don't care" 04 = "Display Device Data or Blank Check" function code ssss = starting address mmmm = ending address ff = subfunction 00 = display data 01 = blank check cc = checksum Example: :05400004400043FFFC003 display 4000-43FF

RECORD TYPE	COMMAND/DATA FUNCTION
05	Miscellaneous Read Functions General Format of Function 05 :02xxxx05ffsscc Where: 02 = number of bytes (hex) in record xxxx = required field, but value is a "don't care" 05 = "Miscellaneous Read" function code ffss = subfunction and selection code 0000 = read signature byte - manufacturer id (D8h) 0001 = read signature byte - device id # 1 (C2h) 0002 = read signature byte - device id # 2 0100 = read security bits 0101 = read status byte 0102 = read boot vector cc = checksum Example: :0200000501FBh read signature byte - device id # 1
06	Direct Load of Baud Rate General Format of Function 06 :02xxxx06hhllcc Where: 02 = number of bytes (hex) in record xxxx = required field, but value is a "don't care" 06 = "Direct Load of Baud Rate" function code hh = high byte of Timer 2 ll = low byte of Timer 2 cc = checksum Example: :0200000604F0h

In Application Programming Method

Several In Application Programming (IAP) calls are available for use by an application program to permit selective erasing and programming of Flash sectors. All calls are made through a common interface, PGM_MTP. The programming functions are selected by setting up the microcontroller's registers before making a call to PGM_MTP at FFF0H. The oscillator frequency is an integer number rounded down to the nearest megahertz. For example, set R0 to 11 for 11.0592 MHz. Results are returned in the registers. The IAP calls are shown in Table 9.

Table 9. IAP calls

IAP CALL	PARAMETER
PROGRAM DATA BYTE	Input Parameters: R0 = osc freq (integer) R1 = 02h DPTR = address of byte to program ACC = byte to program Return Parameter ACC = 00 if pass, !00 if fail
ERASE BLOCK	Input Parameters: R0 = osc freq (integer) R1 = 01h DPH = block code as shown below: block 0 - 8k to 16k, 00H block 1 - 4k to 8k, 01H block 2 - 16k to 32k, 40H block 3 - 32k to 64k, 80H block 4 - 48k to 64k, C0H DPL = 00h Return Parameter none
ERASE BOOT VECTOR	Input Parameters: R0 = osc freq (integer) R1 = 04h DPH = 00h DPL = don't care Return Parameter none
PROGRAM SECURITY BIT	Input Parameters: R0 = osc freq (integer) R1 = 05h DPH = 00h DPL = 00h - security bit # 1 (Inhibit writing to Flash) 01h - security bit # 2 (Inhibit Flash verify) 02h - security bit # 3 (Inhibit external moving) Return Parameter none
PROGRAM STATUS BYTE	Input Parameters: R0 = osc freq (integer) R1 = 06h DPH = 00h DPL = 00h - program status byte ACC = status byte Return Parameter ACC = status byte
PROGRAM BOOT VECTOR	Input Parameters: R0 = osc freq (integer) R1 = 07h DPH = 00h DPL = 00h - program boot vector ACC = boot vector Return Parameter ACC = boot vector
READ DEVICE DATA	Input Parameters: R1 = 03h DPTR = address of byte to read Return Parameter ACC = value of byte read

IAP CALL	PARAMETER
READ MANUFACTURER ID	Input Parameters: R0 = exe freq (integer) R1 = 00h R2 = 00h R3 = 00h (manufacturer ID) Return Parameter ACC = value of byte read
READ DEVICE ID # 1	Input Parameters: R0 = exe freq (integer) R1 = 00h R2 = 00h R3 = 01h (device ID # 1) Return Parameter ACC = value of byte read
READ DEVICE ID # 2	Input Parameters: R0 = exe freq (integer) R1 = 00h R2 = 00h R3 = 02h (device ID # 2) Return Parameter ACC = value of byte read
READ SECURITY BITS	Input Parameters: R0 = exe freq (integer) R1 = 00h R2 = 00h R3 = 04h (security byte) Return Parameter ACC = value of byte read
READ STATUS BYTE	Input Parameters: R0 = exe freq (integer) R1 = 07h R2 = 00h R3 = 01h (status byte) Return Parameter ACC = value of byte read
READ BOOT VECTOR	Input Parameters: R0 = exe freq (integer) R1 = 07h R2 = 00h R3 = 02h (boot vector) Return Parameter ACC = value of byte read
FULL CHIP ERASE	Input Parameters: R0 = exe frequency R1 = 04h R2 = don't care R3 = don't care Return Parameter none

Security

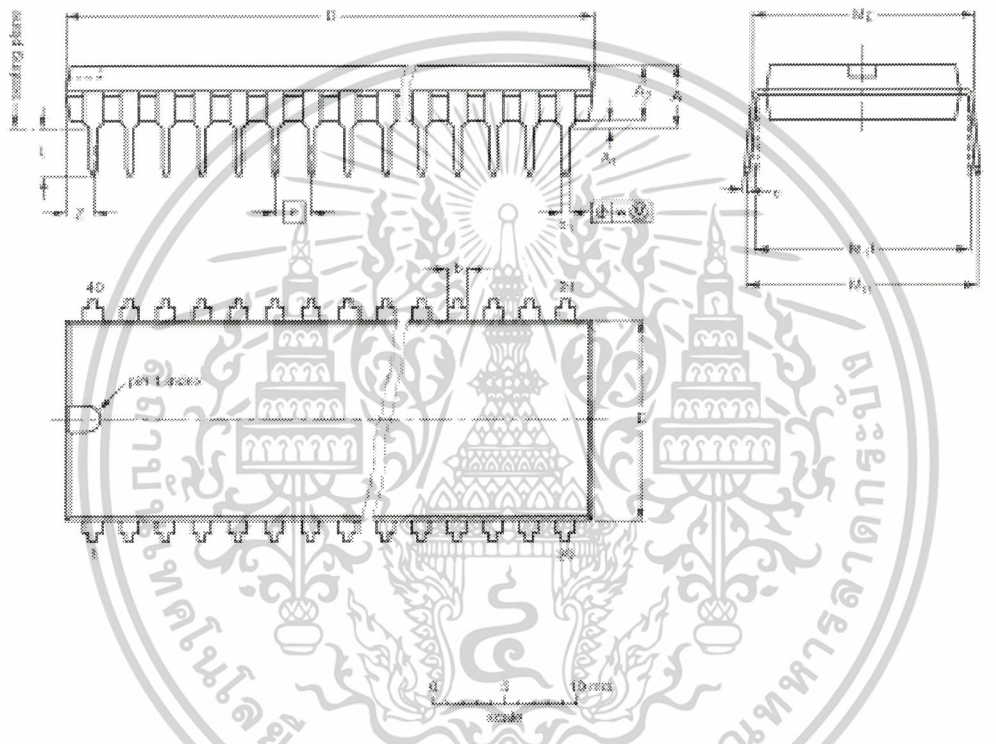
The security feature protects against software piracy and prevents the contents of the Flash from being read. The Security Lock bits are located in Flash. The P89C51RB2/RC2/RD2 has three programmable security lock bits that will provide different levels of protection for the on-chip code and data (see Table 10).

Table 10.

SECURITY LOCK BITS[1]			PROTECTION DESCRIPTION
LB1	LB2	LB3	
X	X	X	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory.
1	X	X	Block erase is disabled. Erase or programming of the status byte or boot vector is disabled.
X	1	X	Verify of code memory is disabled.
X	X	1	External execution is disabled.

NOTE:
1. Security bits are independent of each other. Full-chip erase may be performed regardless of the state of the security bits.

DIP40: plastic dual in-line package; 40 leads (600 mil) — SOT129-1



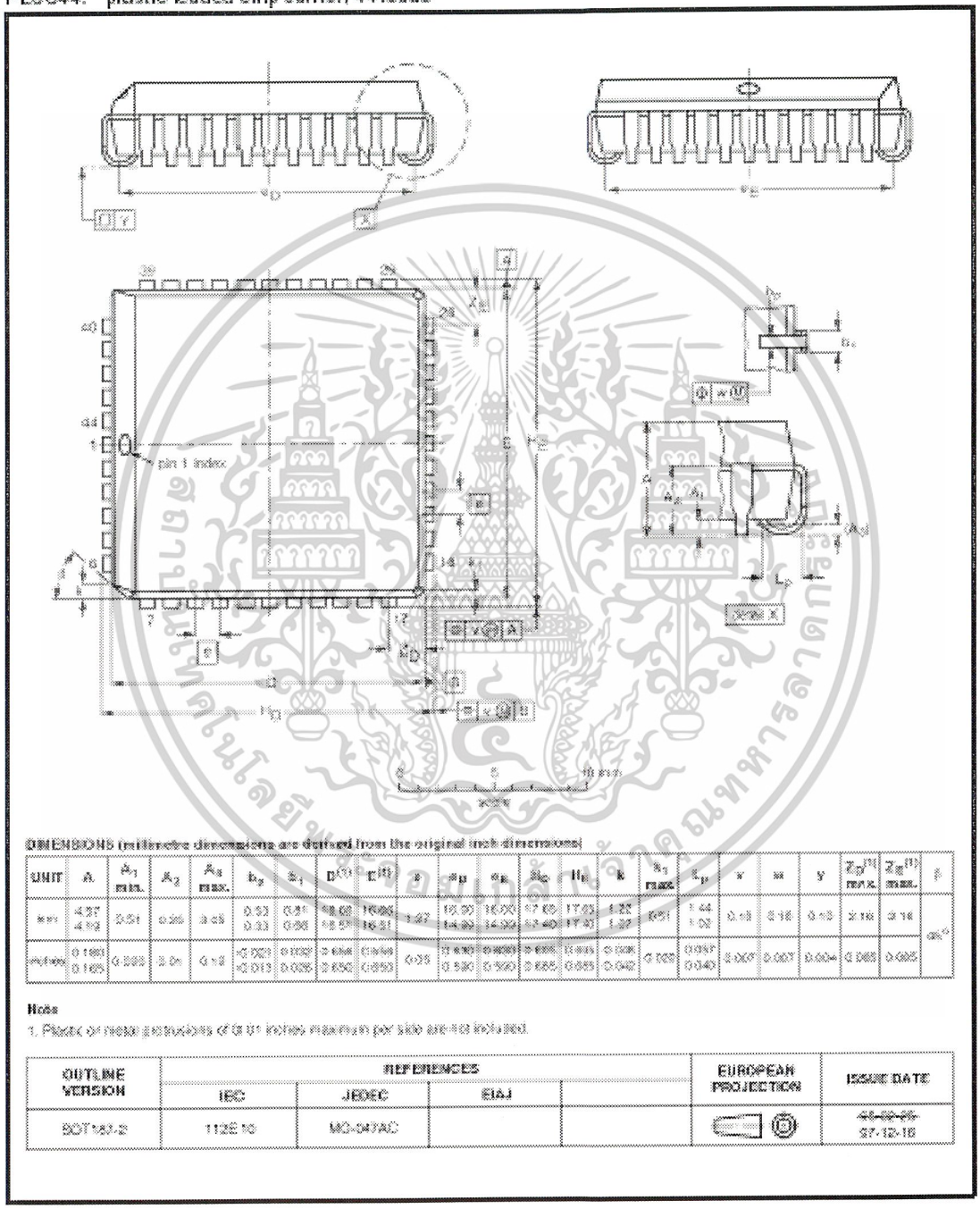
DIMENSIONS (inch dimensions are derived from the original mm dimensions)

UNIT	A max.	A_1 min.	A_2 max.	b	b_1	c	D[1]	E[1]	e	e_1	L	M_E	M_H	w	Z[1] max.
mm	4.7	0.51	4.0	1.70 1.14	0.53 0.38	0.36 0.23	52.50 51.50	14.1 13.7	2.54	15.24	3.60 3.05	16.80 15.24	17.42 15.90	0.254	2.29
inches	0.19	0.020	0.16	0.067 0.045	0.021 0.015	0.014 0.009	2.067 2.028	0.56 0.54	0.10	0.60	0.14 0.12	0.62 0.60	0.69 0.63	0.01	0.090

Note
1. Plastic or metal protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT129-1	051G08	MO-015AJ				92-11-17 95-01-14

PLCC44: plastic leaded chip carrier; 44 leads — SOT187-2



DIMENSIONS (millimetre dimensions are derived from the original inch dimensions)

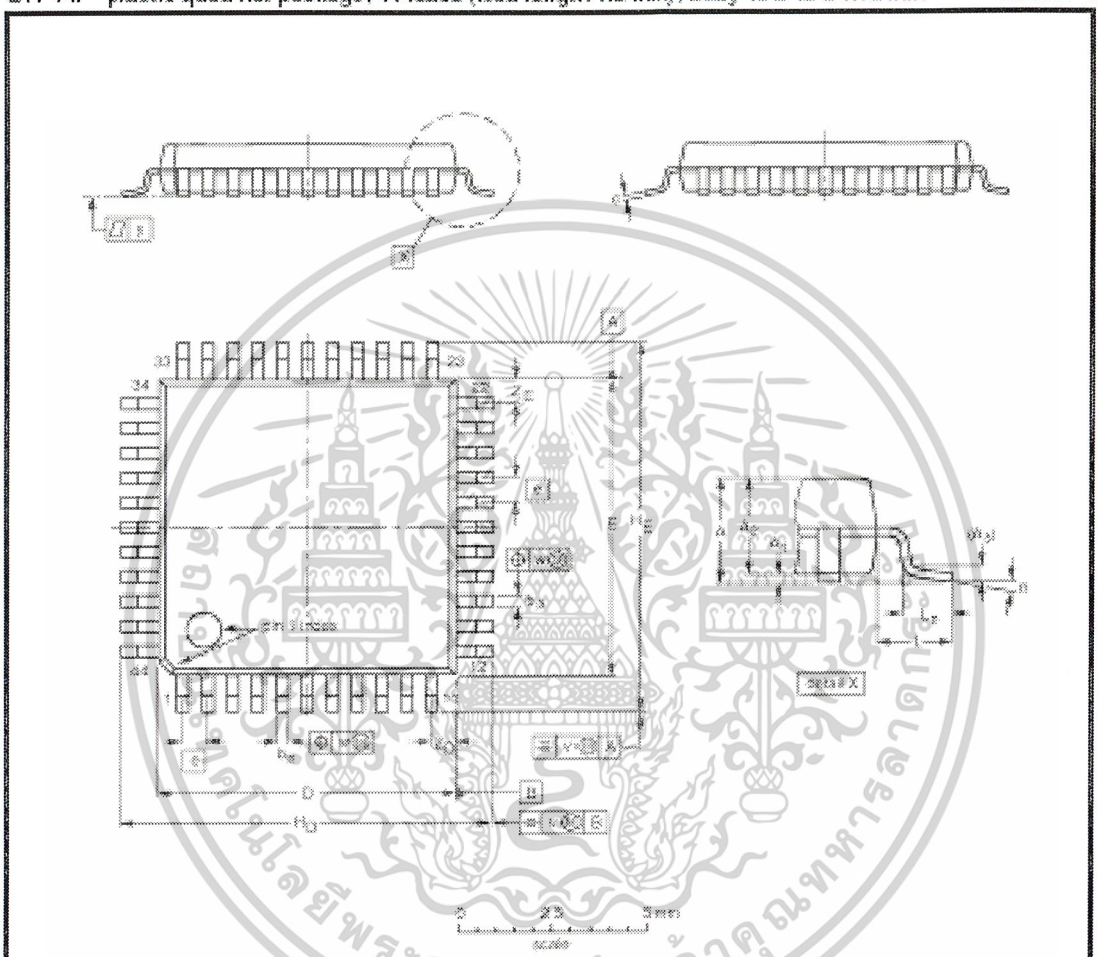
UNIT	A	A_1 min.	A_2	A_3 max.	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_D	H_E	L_p	Q	v	w	y	$Z_D^{(1)}$ max.	$Z_E^{(1)}$ max.	β
mm	4.57 4.19	0.51	3.05	0.25	0.53 0.33	0.81 0.66	16.66 16.51	16.66 16.51	1.27	17.65 17.40	17.65 17.40	1.22 1.07	0.51	0.13	0.18	0.13	2.16	2.16	45°
inches	0.180 0.165	0.020	0.120	0.01	0.021 0.013	0.032 0.026	0.656 0.650	0.656 0.650	0.05	0.695 0.685	0.695 0.685	0.048 0.042	0.020	0.005	0.007	0.005	0.085	0.085	

Note

1. Plastic or metal protrusions of 0.01 inches maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT187-2	112E10	MO-047AC				~~88-02-04~~ 97-12-16

QFP44: plastic quad flat package; 44 leads (lead length 1.3 mm); body 10 x 10 x 1.75 mm — SOT307-2



DIMENSIONS (mm are the original dimensions)

UNIT	$A_{max.}$	A_1	A_2	A_3	b_p	c	D[1]	E[1]	e	H_D	H_E	L	L_p	v	w	y	Z_D[1]	Z_E[1]	θ
mm	2.10	0.25 0.05	1.85 1.65	0.25	0.40 0.25	0.25 0.14	10.1 9.9	10.1 9.9	0.8	12.9 12.3	12.9 12.3	1.3	0.95 0.55	0.15	0.15	0.1	1.2 0.8	1.2 0.8	10° 0°

Note

1. Plastic or metal protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT307-2						~~95-02-04~~ 97-08-01

Data sheet status

Data sheet status	Product status	Definition [1]
Objective specification	Development	This data sheet contains the design target or goal specifications for product development. Specification may change in any manner without notice.
Preliminary specification	Qualification	This data sheet contains preliminary data, and supplementary data will be published at a later date. Philips Semiconductors reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
Product specification	Production	This data sheet contains final specifications. Philips Semiconductors reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

[1] Please consult the most recently issued data sheet before initiating or completing a design.

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone: 800-234-7381

Date of release: 11-99

Document order number: 9397-750-06614

Let's make things better.

Philips Semiconductors

PHILIPS

ประวัติผู้แต่ง

ชื่อ-สกุล นายณัฐพล ธรรมบุตร

วัน เดือน ปีเกิด 15 มิถุนายน 2524

ภูมิลำเนาเดิม 36 หมู่ 13 ต.ส้มป่อย อ.ราษีไศล จ.ศรีสะเกษ 33160

ประวัติการศึกษา

ประถมศึกษา โรงเรียนบ้านโก จ.ศรีสะเกษ

มัธยมศึกษาตอนต้น โรงเรียนศรีสะเกษวิทยาลัย จ.ศรีสะเกษ

ประกาศนียบัตรวิชาชีพ วิทยาลัยเทคนิคมีนบุรี

ประกาศนียบัตรวิชาชีพชั้นสูง วิทยาลัยเทคนิคมีนบุรี

ปริญญาตรี สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม
คณะครุศาสตร์อุตสาหกรรม สจล.

คติพจน์ จะดีจะชั่วก็อยู่ที่ตัวเรา

ประวัติผู้แต่ง

ชื่อ-สกุล นางสาวพิรดา ต๋ำคำ

วัน เดือน ปีเกิด 6 พฤษภาคม 2524

ภูมิลำเนาเดิม 90 หมู่ 10 ต.ป่าแดด อ.แม่สรวย จ.เชียงราย 57180

ประวัติการศึกษา

ประถมศึกษา โรงเรียนชุมชนบ้านโป่ง จ.เชียงราย

มัธยมศึกษาตอนต้น โรงเรียนดำรงราษฎร์สงเคราะห์ จ.เชียงราย

ประกาศนียบัตรวิชาชีพ วิทยาลัยเทคนิคเชียงราย จ.เชียงราย

ประกาศนียบัตรวิชาชีพชั้นสูง วิทยาลัยเทคนิคเชียงราย จ.เชียงราย

ปริญญาตรี สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม
คณะครุศาสตร์อุตสาหกรรม สจล.

คติพจน์ ความกตัญญูเป็นเครื่องหมายของความดี

ประวัติผู้แต่ง

ชื่อ-สกุล	นายสันติ สว่างเมฆ
วัน เดือน ปีเกิด	18 กันยายน 2524
ภูมิลำเนาเดิม	60 หมู่ 8 ต.สามง่าม อ.สามง่าม จ.พิจิตร 66140

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนวังลูกช้าง จ.พิจิตร
มัธยมศึกษาตอนต้น	โรงเรียนสามง่ามชนูปถัมภ์ จ.พิจิตร
ประกาศนียบัตรวิชาชีพ	วิทยาลัยเทคนิคพิจิตร จ.พิจิตร
ประกาศนียบัตรวิชาชีพชั้นสูง	วิทยาลัยเทคนิคพิจิตร จ.พิจิตร
ปริญญาตรี	สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม สจล.
คติพจน์	อย่าเชื่อในสิ่งที่เห็น

ประวัติผู้แต่ง

ชื่อ-สกุล	นายโสภณ แซ่ลิ้ม
วัน เดือน ปีเกิด	17 ตุลาคม 2524
ภูมิลำเนาเดิม	34 หมู่ 2 ต.บางใหญ่ อ.บางใหญ่ จ.นนทบุรี 11140

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนวัดเอนกศิษฐาราม จ.นนทบุรี
มัธยมศึกษาตอนต้น	โรงเรียนเตรียมอุดมศึกษาพัฒนาการ จ.นนทบุรี
ประกาศนียบัตรวิชาชีพ	โรงเรียนเทคโนโลยีสยาม กรุงเทพมหานคร
ประกาศนียบัตรวิชาชีพชั้นสูง	สถาบันเทคโนโลยีราชมงคลวิทยาเขต พระนครเหนือ
ปริญญาตรี	สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม สจล.
คติพจน์	สุดๆ กับชีวิต