ระบบรักษาความปลอดภัย

Security System

โดย

นายวรวุฒิ เขตพิบูลชัย

นายวสันต์ ทองสุข

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ระบบรักษาความปลอดภัย

Security System

โดย

นายวรวุฒิ เขตพิบูลชัย รหัส 46010663

นายวสันต์ ทองสุข รหัส 46010677

อาจารย์ที่ปรึกษา

อาจารย์ ชินภัทร นันทจิวากรชัย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง ระบบรักษาความปลอดภัย

Security System

ผู้จัดทำ

นายวรวุฒิ เขตพิบูลชัย รหัส 46010663

นายวสันต์ ทองสุข รหัส 46010677

_____อาจารย์ที่ปรึกษา

(อ. ชินภัทร นันทจิวากรชัย)

ระบบรักษาความปลอดภัย

นาย วรวุฒิ เขตพิบูลชัย รหัส 46010663
นาย วสันต์ ทองสุข รหัส 46010677
อ. ชินภัทร นันทจิวากรชัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

ในปัจจุบันเทคโนโลยีทางอิเล็กทรอนิกส์ได้ก้าวไปไกลซึ่งระบบรักษาความปลอดภัยและอำนวยความสะดวกสามารถนำเอาสมาร์ตการ์ดมาประยุกต์ใช้ได้ ในโครงงานนี้ใช้บัตรโทรศัพท์สาธารณะที่ใช้หมดแล้วมาทำเป็นบัตรของระบบรักษาความปลอดภัย โดยบัตรโทรศัพท์สาธารณะเป็นบัตรสมาร์ตการ์ดหน่วยความจำชนิด SLE 4436 เครื่องอ่านบัตรใช้ไมโครคอนโทรลเลอร์ MCS-51 โดยจะได้ข้อมูลในส่วนหมายเลขบัตรและควบคุมการส่งข้อมูลจากเครื่องอ่านสมาร์ตการ์ดมายังคอมพิวเตอร์ด้วยการสื่อสารแบบอนุกรมตามมาตรฐาน RS-232 ข้อมูลที่ได้จะนำมาวิเคราะห์ในฐานข้อมูลที่มีอยู่โดยใช้โปรแกรม Microsoft Visual Basic 6.0 และโปรแกรม Microsoft Access ในการออกแบบและสร้างโปรแกรมที่ทำการวิเคราะห์บานข้อมูลดังกล่าวรวมทั้งการแสดงรายละเอียดผลการวิเคราะห์ผ่านจอคอมพิวเตอร์

Security System

Mr. Worrawut Ketphiboonchai ID.46010663

Mr. Wasan Thongsuk ID.46010677

Chinapat Nantagiwagornchai Advisor

Educational Year 2006

Abstract

Electronics technologies have been made progresses every day now we can apply smartcards for security and facilitate system. This project utilizes discarded TOT cards to make security and facilitate system cards. TOT cards is SLE 4436 memory smart card reader was used microcontroller MCS-51 to get identity card number and control data transfer from smart card reader to computer by using RS-232 serial port standard .The data analysis is using Microsoft Visual Basic Version 6.0 and Microsoft Access data base and show analysis result detail through computer monitor.

กิตติกรรมประกาศ

ทางผู้จัดทำขอกราบขอบพระคุณอาจารย์ ชินภัทร นันทจิวากรชัย อาจารย์ที่ปรึกษาเป็นอย่างสูง ที่ให้คำแนะนำและให้คำปรึกษาในการทำโครงงานชิ้นนี้จนสำเร็จตามขอบเขตที่ได้วางไว้ ขอกราบขอบพระคุณอาจารย์ พิชัย คูศิริวานิชกร ซึ่งเป็นอาจารย์อีกท่านหนึ่งที่ให้คำปรึกษาและสนับสนุนในหลายๆด้าน ขอขอบคุณ พ่อ แม่ เพื่อนๆและน้องๆ ที่คอยให้กำลังใจ ให้ความช่วยเหลือต่างๆในการทำงาน และขอขอบคุณปริญญานิพนธ์ทุกเล่มที่เป็นแนวทางให้ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี

นาย วรวุฒิ เขตพิบูลชัย

นายวสันต์ ทองสุข

สารบัญ

สารบัญรูป

หน้า

บทที่ 2

บทที่ 3

บทที่ 4

บทที่ 5

บทที่ 6

บทที่ 7

บทที่ 8

สารบัญตาราง

บทที่ 1 บทนำ

1.1 ความเป็นมาของโครงงาน

เนื่องจากปัจจุบัน การพัฒนาในด้านเทคโนโลยีการเก็บข้อมูลและการเชื่อมต่อข้อมูลมีความก้าวหน้าไปมาก และเป็นที่น่าสนใจในการนำมาประยุกต์ใช้ในชีวิตประจำวันให้มากขึ้น โดยเฉพาะอย่างยิ่งการเก็บข้อมูลแบบหนึ่งที่กำลังเป็นที่น่าสนใจเป็นอย่างมากในปัจจุบันคือ สมาร์ตการ์ด ซึ่งเริ่มจะมีการใช้มากขึ้นในปัจจุบัน จากเหตุผลนี้จึงเป็นที่น่าสนใจที่จะศึกษาและนำมาประยุกต์ใช้สมาร์ตการ์ดให้มีบทบาทในชีวิตประจำวันมากขึ้น โดยในโครงงานนี้มุ่งเน้นการประยุกต์ไปใช้ในด้านระบบรักษาความปลอดภัยและอำนวยความสะดวกแก่ผู้ใช้ สำหรับ สมาร์ตการ์ด ที่ใช้ในโครงงานนี้ได้ประยุกต์มาจากบัตรโทรศัพท์ TOT Card ที่ไม่สามารถชำระค่าโทรศัพท์ได้แล้ว ซึ่งบัตร TOT Card นี้ก็เป็นบัตร สมาร์ตการ์ต ชนิดหนึ่งที่เราหาได้ง่ายและถือว่าเป็นการนำของที่หมดค่าแล้วมาประยุกต์ให้เกิดประโยชน์อีกทางหนึ่ง

1.2 วัตถุประสงค์ของโครงงาน

1. ศึกษาโครงสร้างและการอินเตอร์เฟสกับบัตรสมาร์ตการ์ด โดยเฉพาะอย่างยิ่งบัตร TOT Card
2. ศึกษาการสื่อสารแบบอนุกรมมาตรฐาน RS-232 เพื่อส่งข้อมูลที่ได้จากบัตรสมาร์ตการ์ดสู่คอมพิวเตอร์เพื่อวิเคราะห์ผลต่อไป
3. ศึกษาการเขียนโปรแกรมประยุกต์สำหรับงานด้านการจัดการฐานข้อมูลและการแสดงผล

1.3 ขอบเขตโครงงาน

ทำการสร้างเครื่องอ่านข้อมูลจากบัตร TOT Card และวงจรที่ช่วยจัดการข้อมูลที่ได้จากเครื่องอ่านข้อมูลเข้าสู่เครื่องคอมพิวเตอร์ ในส่วนของโปรแกรมประยุกต์ที่จัดการด้านฐานข้อมูลและแสดงผลจะสามารถแสดงข้อมูลต่างๆ ของเจ้าของบัตรแต่ละใบที่ทำการลงทะเบียน ไว้กับระบบและทำการฟ้องเมื่อมีบัตรที่ไม่ได้ลงทะเบียนกับระบบมาใช้งาน พร้อมทั้งเพิ่มส่วนของการเตือนภัยตามจุดต่างๆโดยจะติดตั้ง เซนเซอร์ไว้สำหรับตรวจจับการงัดเข้ามาโดยไม่ได้รับอนุญาตอีกด้วย

1.4 วิธีการดำเนินการ

1. ศึกษาการอินเตอร์เฟสเพื่ออ่านข้อมูลภายในบัตร TOT Card พร้อมออกแบบและสร้างเป็นเครื่องอ่านข้อมูลจากบัตร TOT Card

2. ศึกษาการเชื่อมโยงข้อมูลระหว่างเครื่องอ่านข้อมูลบัตร TOT Card กับคอมพิวเตอร์

3. เขียนโปรแกรมประยุกต์ที่มีความสามารถในการรับข้อมูลจากเครื่องอ่านบัตร TOT Card พร้อมทั้งนำข้อมูลที่ได้มาใช้ในการค้นหาข้อมูลที่สอดคล้องกันในฐานข้อมูลที่มีอยู่ และแสดงผลข้อมูลที่สอดคล้องในฐานข้อมูลบนคอมพิวเตอร์

4. สร้างวงจรเซนเซอร์เพื่อตรวจจับบริเวณหน้าต่างพร้อมทั้งส่งสัญญาณกลับมายังคอมพิวเตอร์เพื่อเตือนให้ผู้ใช้รับทราบ

1.5 ประโยชน์ที่ได้รับ

1. เพิ่มประสิทธิภาพความปลอดภัย
2. สร้างความสะดวกสบายแก่ผู้ใช้ในการทำกิจกรรมต่างๆในชีวิตประจำวัน
3. นำของที่หมดค่ามาประยุกต์ใช้ให้เกิดประโยชน์

บทที่ 2 สมาร์ตการ์ด(Smart Card)

2.1 ความหมายของสมาร์ตการ์ด

Smart Card หมายถึง การ์ดที่มีหน่วยความจำ หรือ มี Microprocessor ฝังอยู่ในการ์ด อาจจะเป็น Chip หน่วยความจำข้อมูล หรือลบข้อมูล หรือไม่เช่นนั้นก็สามารถปรับปรุงเปลี่ยนแปลง ข้อมูลบนตัวการ์ดนี้ อีกประเภทคือ การ์ดที่มีหน่วยความจำคงที่ หรือที่เรียกว่า memory chip card เช่น การ์ดโทรศัพท์ เป็นต้น ลักษณะตัวการ์ด สมาร์ตการ์ด นั้นจะเป็นแผ่นพลาสติกขนาดเท่ากับบัตรเครดิต หรือ ขนาดใกล้เคียงกับนามบัตร ซึ่งเป็นขนาดมาตรฐานทั่วโลก ภายในการ์ดนี้ก็จะมีเนื้อที่ส่วนที่เป็นหน่วยความจำอยู่บนการ์ต ซึ่งในส่วนนี้เองที่จะเป็นส่วนบันทึกข้อมูลอยู่ภายในหากจะเพิ่มเติมข้อมูล หรืออ่านข้อมูลจากบัตรก็จะต้องมีเครื่องอ่านบัตรที่เรียกว่า Smart Card Reader

Smart Card นั้นจะไม่เหมือนกับการ์ดชนิดแถบแม่เหล็ก เพราะ สมาร์ตการ์ด นั้นจะมีข้อมูลที่จำเป็น และข้อมูลข่าวสารอื่นๆอยู่บนการ์ด ดังนั้นการอ่านข้อมูลจึงไม่ต้องย้อนกลับไปค้นข้อมูลจากศูนย์ข้อมูลอันเป็นการเสียเวลาเช่นเดียวกับการ์ดแถบแม่เหล็ก (เช่นบัตร เอทีเอ็ม) นี่ก็จะเป็นข้อดีของสมาร์ตการ์ด

2.2 ประวัติความเป็นมาของสมาร์ตการ์ด

Smart Card นั้นประดิษฐ์ขึ้นมาในปี ค.ศ. 1974 จวบกระทั่งวันนี้มี สมาร์ตการ์ด ใช้อยู่ทั่วโลกราว 100 ล้านใบ จากหลายๆ ผู้ผลิต 95 เปอร์เซ็นต์ ใช้ในประเทศแถบยุโรป , อเมริกาใต้ และประเทศในแถบเอเชีย คาดว่าสิ้นปีนี้อาจใช้งานถึง 3000 ล้านใบ กว่า 15 เปอร์เซ็นต์ ก็จะใช้งานในสหรัฐอเมริกาและแคนาดา ในจำนวนนี้ประมาณว่า จำนวน 900 ล้านใบ จะใช้ในกิจการบริการทางด้านการเงิน การ์ดธนาคาร หรือ การ์ดเอทีเอ็ม การ์ดด้านรักษาความปลอดภัย การ์ดที่เกี่ยวกับโทรศัพท์ไร้สาย การ์ดที่เกี่ยวกับการสื่อสารดาวเทียม และการ์ดที่เกี่ยวกับการให้บริการด้านเคเบิลทีวี เป็นต้น

ปัจจุบันบางประเทศใกล้บ้านเรานั้น ปัจจุบันเค้าก็เอามาใช้เต็มรูปแบบแล้ว บ้านเราก็คงต้องอีกระยะหนึ่ง หรืออีกไม่นานท่านก็จะคุ้นเคยกับคำว่า E – Money ที่อาจจะต้องมีการนำเอาสมาร์ตการ์ต มาใช้งานด้วยอย่างแน่นอน ปัจจุบันราคาเครื่องผลิต สมาร์ตการ์ด ราคายังสูงมาก นับว่าเป็น 10 ล้าน ต่อเครื่อง ถ้าได้ใช้งานอย่างคุ้มค่าก็ต้องให้บริการแก่หน่วยงานใหญ่ๆ ที่ส่วนตัว หน้าตาอย่างไร มีตำหนิรูปพรรณที่ไหน บ้านอยู่ไหน เป็นลูกใคร กรุ๊ปเลือดใด เคยไปบริจาคเลือดที่ไหน กี่ครั้งรวมไปถึงประวัติการเรียน การทำงาน ความสามารถพิเศษ ได้กระทำความดี หรือมีประวัติเสียส่วนใด เช่น อุบัติเหตุ ก่อคดีวิวาท เป็นโจทย์ เป็นจำเลย คดีแพ่ง คดี อาญา จ่ายภาษี จ่ายประกันสังคม มี

รถยนต์กี่คัน เมื่อจำเป็นต้องใช้งานจำนวนมากก็อาจจะคุ้ม นั่นก็ต้องขึ้นอยู่กับจำนวนการผลิตเท่านั้น ว่าไปแล้วอีกไม่นานนักหากการจักระบบรักษาความปลอดภัย ระบบการจัดเก็บ ระบบการเช็คชื่อ ไม่ว่าจะเป็นบริษัท ห้างร้าน โรงงาน หรือตามโรงเรียนต่างๆ ในอนาคตย่อมไม่อาจจะหลีกเลี่ยงการนำเอา สมาร์ตการ์ด มาใช้อย่างแน่นอน รวมไปถึงระบบการรักษาความปลอดภัยของประชาชน เช่น บัตรประจำตัวประชาชนก็อาจจะเปลี่ยนเป็นแบบ สมาร์ตการ์ด เมื่อเข้าโรงพยาบาลก็จะรู้ทันทีว่าแพ้ยาอะไร ใครจะเป็นคนจ่ายเงินค่ารักษาพยาบาล แพทย์ก็ทำการรักษาอย่างถูกต้องทันต่อการช่วยชีวิตทันทีทันควัน หรือในบางธุรกิจ เช่น บริษัทจำหน่ายรถยนต์ เมื่อรถยนต์เข้าอู่ก็จะอ่านข้อมูลทราบรายละเอียดของรถทันที ใครเป็นเจ้าของ ซื้อเมื่อไร หมดประกันเมื่อใด ซ่อมอะไรไปบ้าง ชิ้นส่วนใดจะต้องเปลี่ยน รสนิยมของเจ้าของเป็นอย่างไร จ่ายเงิน ช้า เร็วชอบของแท้หรือของเทียม เป็นต้น

เทคโนโลยี	Magnetic Card	Optical Card	Smart Card
อัตราการบกพร่อง	2 %	2 %	0.03 %
จำนวนครั้งที่เข้าไปยุ่งเกี่ยว	400	400	100
จำนวนครั้งที่บกพร่องต่อปี	1000	800	100
จำนวนเครื่องที่สามารบำรุงรักษาโดยใช้ช่างเทคนิค 1 คน	20	15	100
ค่าใช้จ่ายอะไหล่	400	500	100

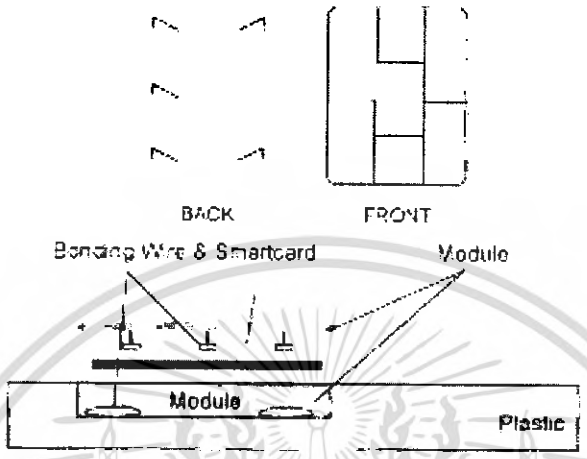
ตารางที่ 2.1 การเปรียบเทียบการใช้งานการ์ดชนิดต่างๆของ France Telecom

2.3 ข้อดีของ สมาร์ตการ์ด

1. มีความน่าไว้วางใจกว่าบัตรที่ใช้แถบแม่เหล็ก
2. สามารถเก็บข้อมูลได้มากกว่าบัตรที่เป็นแถบแม่เหล็กเป็นร้อยๆเท่า
3. ลดโอกาสที่จะเข้าไปยุ่งเกี่ยวและป้องกันการปลอมแปลงด้วยระบบป้องกันที่ซับซ้อน
4. สามารถเปลี่ยนมือและสามารถนำกลับมาใช้ใหม่ได้
5. ทำหน้าที่ต่างๆได้มากมาย
6. สามารถนำไปใช้งานต่างๆได้อย่างกว้างขวาง เช่น การขนส่ง การธนาคาร และการรักษาสุขภาพ เป็นต้น
7. สามารถประยุกต์ใช้กับอุปกรณ์อิเล็กทรอนิกส์ แบบพกพาต่างๆได้ เช่น เครื่องโทรศัพท์ และ เครื่องคอมพิวเตอร์กระเป๋าหิ้ว
8. ทำงานด้วยเทคโนโลยีเซมิคอนดัคเตอร์ที่มีการพัฒนาอย่างรวดเร็ว

2.4 ส่วนประกอบและโครงสร้างของ สมาร์ตการ์ด

สมาร์ตการ์ดประกอบไปด้วยบัตรพลาสติก กาวหรือวัสดุที่ใช้เชื่อมต่อ และหน้าสัมผัสที่บรรจุ ชิป สมาร์ตการ์ตเรียบร้อยแล้ว ซึ่งส่วนประกอบต่างๆแสดงดังรูป



รูปที่ 2.1 ส่วนประกอบของสมาร์ตการ์ด

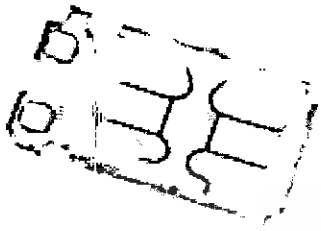
2.4.1 ตัวบัตรพลาสติก(Plastic Card)

ขนาดของบัตรพลาสติกที่นำมาทำสมาร์ตการ์ดกำหนดโดยมาตรฐานระหว่างประเทศ คือ ISO 7810 โดยมาตรฐานนี้ยังได้กำหนดถึงคุณลักษณะทางกายภาพของพลาสติกที่นำมาใช้ทำบัตรด้วย เช่น ความคลาดเคลื่อนของอุณหภูมิ และความยืดหยุ่นตัวในการใช้งาน ตำแหน่งของหน้าสัมผัสทางไฟฟ้า และการทำงานของมัน ตลอดจนกำหนดว่าการติดต่อกับวงจรรวม (Integrated Circuit) หรือ IC กับโลกภายนอกเป็นอย่างไรอีกด้วย มีพลาสติกอยู่ 4 ชนิดที่นำมาใช้ผลิตสมาร์ตการ์ด ได้แก่ PVC (Poly Vinyl Chloride), ABS (Acrylonitrile Butadiene Styrene), PC (Polycarbonate), PET (Polyethylene Terephthalate) แต่ที่นิยมกันมากในประเทศไทยคือ PVC (Poly Vinyl Chloride) และ ABS (Acrylonitrile Butadiene Styrene) อย่างไรก็ดี การใช้ พีวีซีมีข้อดีคือสามารถพิมพ์ลายนูนได้ แต่ไม่สามารถนำกลับมาใช้ใหม่ ไม่สามารถย่อยสลายในธรรมชาติได้ ส่วน เอบีเอส ไม่สามารถพิมพ์นูนได้แต่นำกลับมาใช้งานใหม่ได้

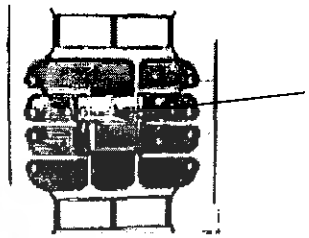
2.4.2 หน้าสัมผัสและชิปสมาร์ตการ์ด (Smart Card Module)

สมาร์ดการ์ดโมดูลหรือหน้าสัมผัสและชิปสมาร์ตการ์ดคือ ส่วนที่แสดงความเป็นตัวตนของสมาร์ดการ์ดที่สุด สมาร์ดการ์ดบางชนิดเมื่อหยิบขึ้นมาเราไม่สามารถทราบได้เลยว่ามันคือ

สมาร์ตการ์ดที่มีการฝังชิป ไว้ในบัตร โดยส่วนที่จะแสดงภาพลักษณ์ที่ชัดเจนของสมาร์ตการ์ดคือ สมาร์ตการ์ดโมดูล

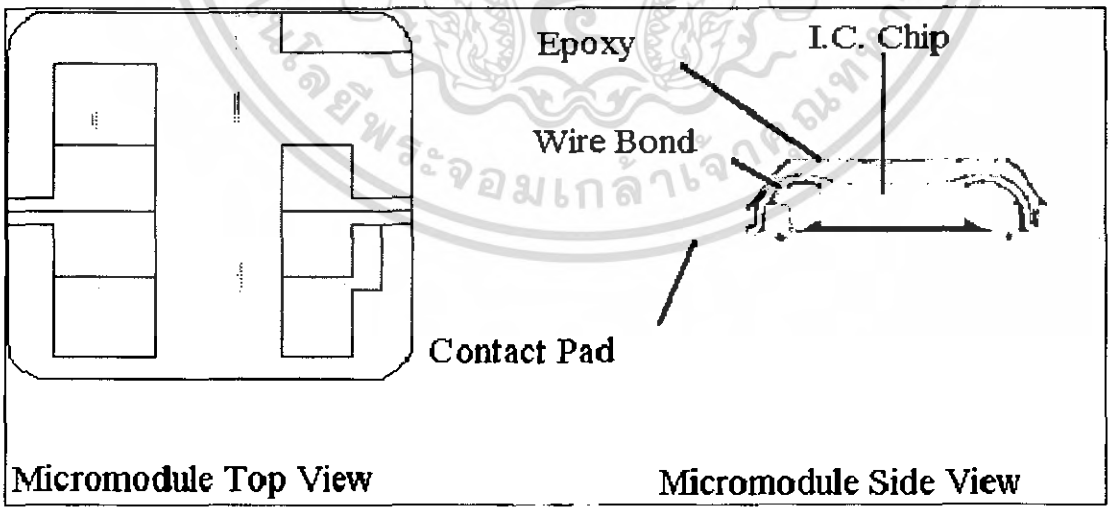


รูปที่ 2.2 หน้าสัมผัส



รูปที่ 2.3 ชิปสมาร์ตการ์ด

ในการผลิตสมาร์ตการ์ดโมดูล ส่วนที่เป็นหน้าสัมผัสสมาร์ตการ์ดประกอบด้วยโลหะหลายชนิดประกอบกัน แต่ละส่วนจะถูกยึดด้วยฟิล์มบางๆ ทางด้านหลังของหน้าสัมผัสเพื่อให้คงรูปอยู่ได้ แถบฟิล์มตัวนี้จะมีการเจาะช่องเล็กๆ สำหรับการเชื่อมต่อสายนำสัญญาณ กับสมาร์ตชิปกับหน้าสัมผัส หลังจากที่วางชิปสมาร์ตการ์ดลงตำแหน่งที่ต้องการ และทำการเชื่อมต่อสายนำสัญญาณจากชิป สมาร์ตการ์ด เข้ากับหน้าสัมผัสเรียบร้อยแล้ว ขั้นสุดท้ายจะเป็นการผนึก ชิปเพื่อต้องการผนึกชิปและสายนำสัญญาณต่างๆ จากสิ่งแวดล้อมภายนอก ขั้นสุดท้ายจะเป็นการนำหน้าสัมผัสและชิปใส่ลงในบัตรพลาสติกและทดสอบการทำงานของชิปขั้นสุดท้าย



รูปที่ 2.4 ตัวอย่าง สมาร์ตการ์ดโมดูล

2.5 องค์ประกอบในการใช้งานสมาร์ตการ์ด

2.5.1 ตัวบัตรและชิป

บัตรและชิป สมาร์ตการ์ด เป็นส่วนแรกที่จะกล่าวถึงเพราะสมาร์ตการ์ดมีหลากหลายรูปแบบ หลากหลายการใช้งานโดยหลักการแล้วสมาร์ตการ์ดเป็นเพียงบัตรฝัง ชิป IC ที่สามารถเก็บข้อมูลได้เท่านั้น ผู้ออกแบบระบบมีหน้าที่นำสมาร์ตการ์ดมาใช้งานอย่างชาญฉลาดเหมาะสมตามประเภทงาน และบริหารข้อมูลภายในสมาร์ตการ์ดให้เกิดความปลอดภัยสูงสุด

สมาร์ตการ์ดที่นำมาใช้งานมีตั้งแต่ราคาใบละไม่กี่ร้อยบาท ไปถึงใบละหลายพันบาท โดยในปัจจุบันเราสามารถเห็นการใช้งานสมาร์ตการ์ด ในหลายรูปแบบเช่น บัตรโทรศัพท์ ซิมการ์ดในโทรศัพท์มือถือ บัตรเข้าออกที่อยู่อาศัย บัตรนักศึกษา บัตรพนักงาน บัตรเติมน้ำมันแบบเครดิต (Fleet Card) บัตรแทนเงินสด ซิมการ์ดในโทรศัพท์มือถือซึ่งมีการกำหนดเป็นมาตรฐาน GSM โดยผู้ผลิตสมาร์ตการ์ดต้องผลิตสมาร์ตการ์ดที่มีโครงสร้างข้อมูลภายในตามที่มาตรฐาน GSM กำหนด

2.5.2 สมาร์ตการ์ดรีดเดอร์

สมาร์ตการ์ดรีดเดอร์จะประกอบไปด้วยขาสำหรับเชื่อมสัญญาณกับหน้าสัมผัสบน ชิปสมาร์ตการ์ด (Card Contact) หรือ เป็นเสาอากาศรับส่งวิทยุสำหรับสมาร์ตการ์ดแบบไม่มีหน้าสัมผัส (Contact less) และหน่วยประมวลผลพร้อมหน่อยความจำสำหรับติดต่อสื่อสารกับ ชิป สมาร์ตการ์ดโดยตรง การสร้าง สมาร์ตการ์ดรีดเตอร์ขึ้นใช้เองสามารถทำได้โดยการนำไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ มาประยุกต์ใช้งานในการเชื่อมต่อกับสมาร์ตการ์ด

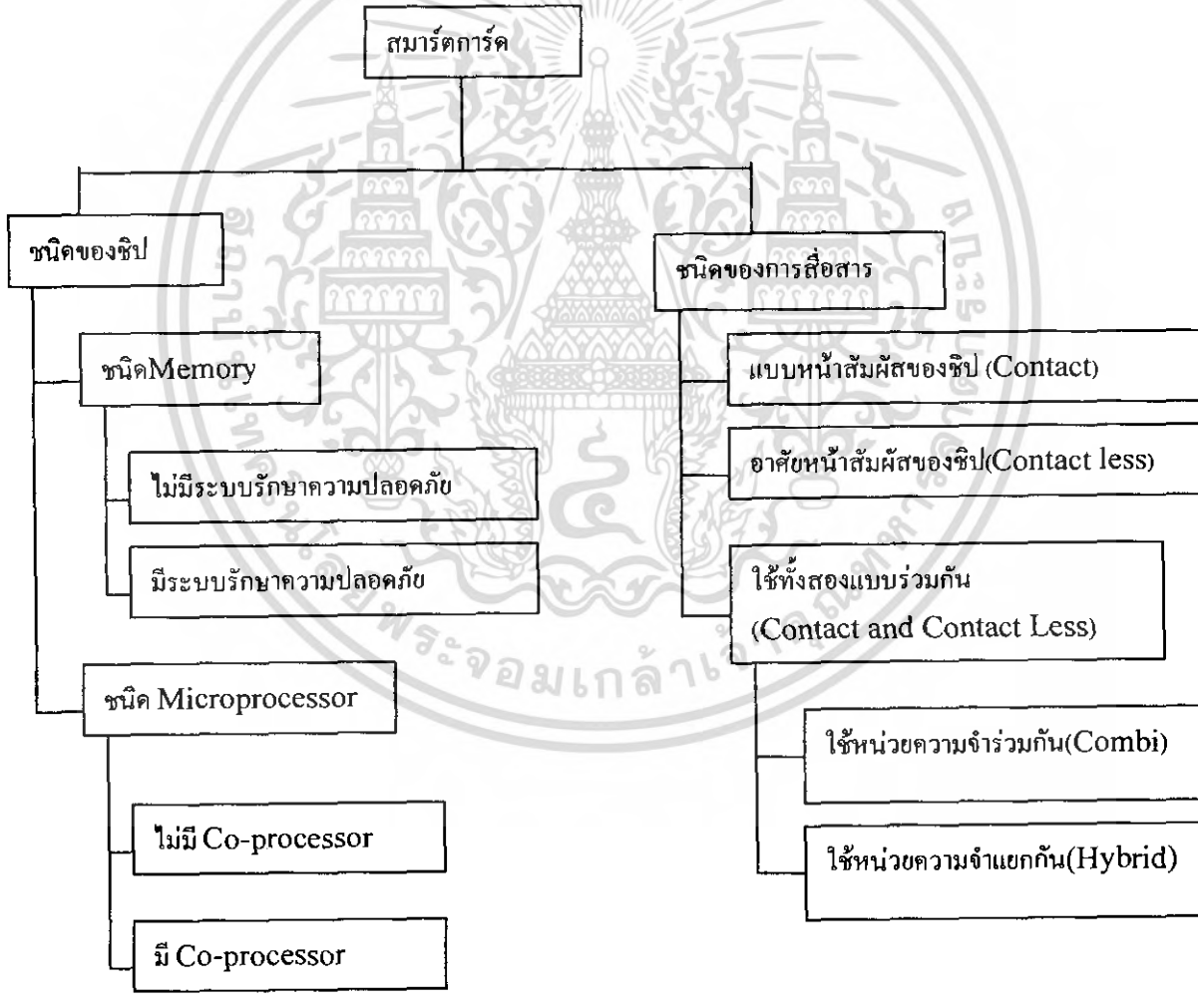
2.5.3 ซอฟต์แวร์

ในการใช้งานสมาร์ตการ์ดนอกจากตัวบัตรสมาร์ตการ์ด สมาร์ตการ์ดรีดเดอร์แล้วยังมีส่วนประกอบอีกส่วนที่สำคัญคือ ซอฟต์แวร์สำหรับจัดการข้อมูลภายในสมาร์ตการ์ดและซอฟต์แวร์ด้านการบริหารงานด้านบัตรหรืออาจเรียกได้ว่าระบบ Front – end ซึ่งระบบ Front – end ของสมาร์ตการ์ดจะแตกต่างจากระบบบัตรแถบแม่เหล็ก เนื่องจากสมาร์ตการ์ตไม่จำเป็นต้องมีการติดต่อสื่อสารกับ Front – end ทุกครั้งที่ทำรายการเหมือนในระบบบัตรเครดิต ทำให้ระบบ Front – end ของสมาร์ตการ์ดมีเวลามากพอในการบริหารงานด้านอื่นๆ หากต้องการติดต่อสื่อสารกับระบบ Front – end ของสมาร์ตการ์ดจำเป็นต้องใช้สมาร์ตการ์ดรีดเดอร์ที่มีส่วนสำหรับติดต่อสื่อสารไม่ว่าจะเป็น MODEM , Ethernet , Local Area Network, ระบบสื่อสารด้วยเครื่องวิทยุ ,ระบบสื่อสารอนุกรม RS – 485/422 สำหรับการสื่อสารในบริเวณพื้นที่ให้บริการที่ไม่กว้างใหญ่นักเพื่อใช้รับส่งข้อมูลระหว่าง Front – end เมื่อจำเป็น

2.5.4 Back Office

เป็นซอฟต์แวร์สำหรับดูแลระบบทั้งหมด ประกอบด้วยซอฟต์แวร์สำหรับป้อนข้อมูลเกี่ยวกับบัตร และผู้ถือบัตรเพื่อออกบัตรใหม่ (Card Issuer) ซอฟต์แวร์สำหรับการออกรายงานต่างๆ (Report) และซอฟต์แวร์ส่วนสุดท้ายคือสำหรับให้บริการผู้ถือบัตร เช่น ซอฟต์แวร์สำหรับเติมเงินลงในชิป (สมาร์ตการ์ดที่ใช้แทนเงินสด) ซอฟต์แวร์สำหรับเติม- แลกแต้มในบัตรสะสมแต้ม (Royalty Card) ปกติแล้วซอฟต์แวร์ในส่วนของ Back-end และ Back Office ที่กล่าวมาต้องทำงานร่วมกับสมาร์ตการ์ดรีดเดอร์เสมอ เพราะเพียงสมาร์ตการ์ดไม่สามารถทำรายการใดๆได้เ

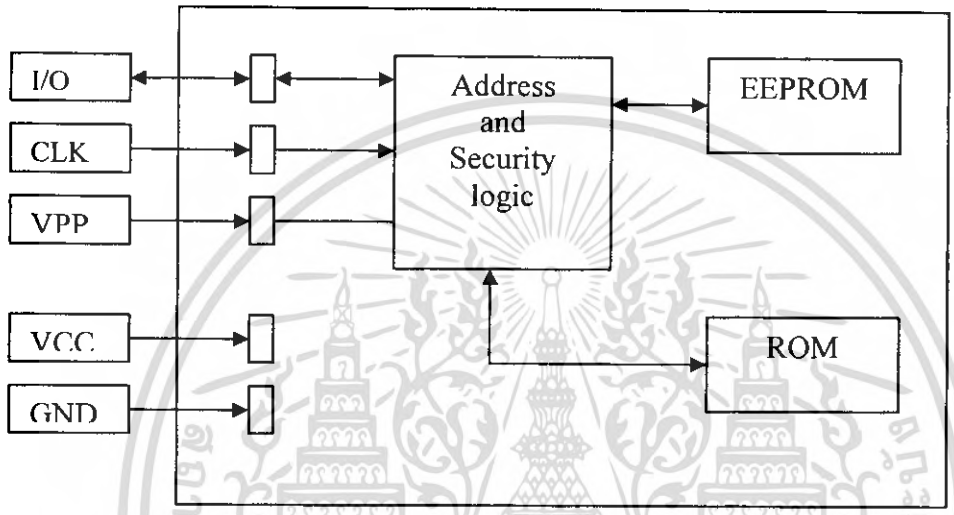
2.6 ประเภทของสมาร์ตการ์ด



รูปที่ 2.5 การแบ่งสมาร์ตการ์ดตามชนิดของหน่วยความจำและประเภทของหน้าสัมผัส

2.6.1 Memory Card (Synchronous Card)

สมาร์ตการ์ดแบบ Memory หรืออีกชื่อหนึ่งคือ Synchronous Card เนื่องจากสมาร์ดการ์ดชนิดนี้มีการรับ ส่งข้อมูลตามสัญญาณนาฬิกาที่ป้อนให้แก่ชิป (ข้อมูลแต่ละบิตที่ส่งให้แก่ ชิป ต้องสัมพันธ์กับสัญญาณนาฬิกา) สมาร์ตการ์ดชนิดนี้มีโครงสร้างที่ประกอบไปด้วย วงจรสำหรับติดต่อสื่อสารภายนอก หน่วยความจำข้อมูล และหน่วยความจำสำหรับเก็บชุดคำสั่งของสมาร์ตการ์ด



รูปที่ 2.6 บล็อกไดอะแกรมโครงสร้างในชิปสมาร์ตการ์ดชนิด Memory

สมาร์ตการ์ดที่เป็นพื้นฐานในปัจจุบัน ก็คือสมาร์ตการ์ดชนิด Free Access Memory สมาร์ตการ์ดชนิดนี้เปิดโอกาสให้อ่านข้อมูลหรือเขียนข้อมูลในแอดเดรสใดๆ ก็ได้ตามชื่อของสมาร์ตการ์ดชนิดนี้ ไม่มีการป้องกันข้อมูลใดๆภายในการ์ดชนิดนี้เป็นการ์ดที่มีความปลอดภัยต่ำสุด แต่การอ่านข้อมูลก็ไม่สามารถอ่านได้ง่ายนักเนื่องจากเมื่อมีการออกแบบหน่วยความจำให้มีการสลับตำแหน่งของบิตข้อมูลโดยมีวงจรควบคุมการสลับตำแหน่งของบิตเป็นส่วนป้องกันข้อมูลอีกต่อหนึ่ง ดังนั้นการอ่านข้อมูลแบบธรรมดา จะไม่ได้ข้อมูลที่ถูกต้องหากไม่ได้ติดต่อกับวงจรควบคุมการสลับตำแหน่งของบิตโดยตรง

นอกจากสมาร์ตการ์ดแบบชนิด Memory ธรรมดายังมีการใส่วงจรกำหนดเงื่อนไขการเข้าถึงข้อมูลลงไปด้วย ทำให้สามารถกำหนดเงื่อนไขการอ่าน เขียนข้อมูลได้ทุก ไบต์ โดยสมาร์ตการ์ดที่มีการป้องกันการอ่าน เขียนข้อมูลชนิดนี้ถูกเรียกว่า PIN Protect Memory เนื่องจากการเข้าถึงข้อมูลจะต้องแสดงรหัสผ่านให้บัตรรับทราบก่อนจึงจะสามารถเข้าถึงข้อมูลได้ วงจรกำหนดเงื่อนไขการเข้าถึงข้อมูลจะมีบิตพิเศษที่เรียกว่า Bit Protect ซึ่งเป็นบิตข้อมูลที่ฝากไว้กับข้อมูลให้เป็นบิตที่ 9 แต่ไม่สามารถแก้ไขด้วยคำสั่งเขียนข้อมูลธรรมดา เพราะ Bit Protect ไม่ได้เป็นส่วนหนึ่งของข้อมูลจริงๆ

ในการแก้ไข Bit Protect นี้ จะสามารถ ทำการเปลี่ยนแปลงได้เพียงครั้งเดียวด้วยคำสั่งเฉพาะเท่านั้น เช่น หากต้องการบังคับไม่ให้ข้อมูลไบต์ใดไม่สามารถแก้ไขได้ก็ให้ทำการเคลียร์บิตที่ 9 ของข้อมูลไบต์นั้นๆ แต่สำหรับรหัสผ่านในการเข้าถึงข้อมูลสามารถเปลี่ยนแปลงได้แต่ต้องแสดงรหัสผ่านชุดเก่าให้บัตรได้รับทราบก่อนจึงจะสามารถเปลี่ยนแปลงรหัสผ่านได้

สมาร์ดการ์ดชนิด Memory เป็นสมาร์ตการ์ดที่เป็นพื้นฐานของสมาร์ตการ์ดรุ่นใหม่ๆในปัจจุบัน ด้วยโครงสร้างและการทำงานที่ง่ายต่อการทำความเข้าใจ ราคาถูก สามารถเก็บข้อมูลได้เป็นจำนวนมาก และความเร็วในการประมวลผลไม่มากเกินไปจึงทำให้สมาร์ตการ์ดชนิดนี้เหมาะสมที่จะนำไปประยุกต์ใช้งานกับข้อมูลที่ไม่มีความสำคัญมากนัก เช่น บัตรลงเวลา บัตรผ่านประตู บัตรโทรศัพท์ ปัจจุบันบัตรสมาร์ดการ์ดชนิด Memory มีหน่วยความจำสูงสุด 64 กิโลไบต์ และอนาคดความจุของสมาร์ดการ์ดจะเพิ่มขึ้นไปอีก

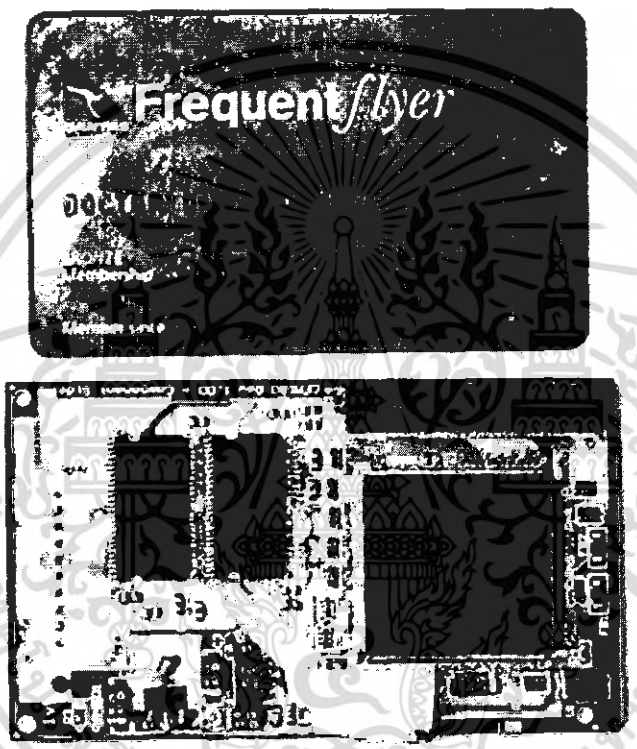
2.6.2 การ์ดแบบออพทิคอลเมมโมรี (Optical Memory Cards)

รูปที่ 2.7 ตัวอย่างบัตรแบบ Optical Memory Cards

เป็นสมาร์ตการ์ดแบบที่ใช้ส่งข้อมูลทางแสง (Optical) ในส่วนที่มีข้อมูลนั้นก็จะมีลักษณะเหมือนแผ่นพลาสติก หรือเศษของแผ่นซีดี อัดอยู่บนการ์ดนี้ ซึ่งในขณะนี้มีการนำมาใช้บ้างแล้ว Optical Memory Card นี้ สามารถเก็บข้อมูลได้ถึง 4 MB แต่ลงหรือบรรจุข้อมูลได้ครั้งเดียว เปลี่ยนแปลงหรือโยกย้ายข้อมูลออกจากการ์ดนี้ไม่ได้ดังนั้นการ์ด Optical Memory Cards นี้ก็มักจะถูกนำไปใช้ในวงการ การเก็บรักษาข้อมูล เช่น ข้อมูลทางการแพทย์ ข้อมูลเกี่ยวกับบุคคล ข้อมูลการท่องเที่ยว เป็นต้น ปัจจุบัน ตัวการ์ดเองก็ไม่มี Microprocessor อยู่ในตัวการ์ดแต่ในอนาคตก็อาจมีการปรับปรุงให้ทันสมัยมากกว่านี้ อีกประการหนึ่งคือ เครื่องอ่านข้อมูลของบัตร Optical Memory Cards นี้ราคายังค่อนข้างแพงและที่สำคัญยังไม่มีมาตรฐานมาบังคับใช้

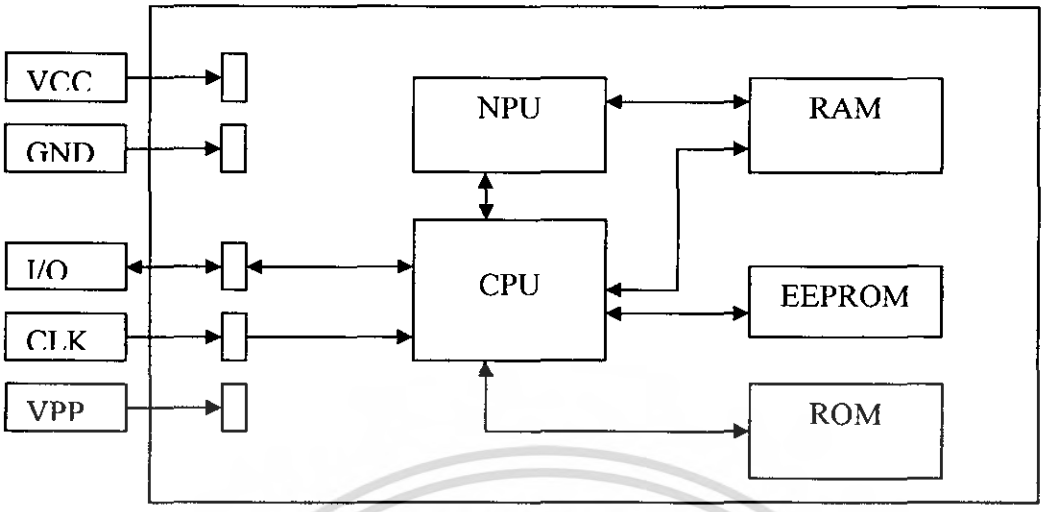
2.6.3 Processor Card (Asynchronous Card)

สมาร์ตการ์ดชนิดนี้เป็นสมาร์ตการ์ดที่ได้รับการปรับปรุงจากสมาร์ตการ์ดชนิด Memory ด้วยการใส่เทคโนโลยีไมโครโปรเซสเซอร์เข้าไปในชิป เพื่อให้ ชิป สามารถประมวลผลข้อมูลและเพิ่มความปลอดภัยให้แก่ข้อมูลให้สูงขึ้น การที่ใส่ไมโครโปรเซสเซอร์ลงไปใน ชิป ทำให้จำเป็นต้องมีการเพิ่มส่วนของหน่วยความจำสำหรับจัดเก็บระบบปฏิบัติการไมโครโปรเซสเซอร์และหน่วยความจำชั่วคราวสำหรับการประมวลผล นอกจากนี้การใส่ชิปประมวลผลทางคณิตศาสตร์ลง



รูปที่ 2.8 ตัวอย่างบัตร Processor Card (Asynchronous Card)

ในชิปสมาร์ตการ์ดเพื่อช่วยในการประมวลผลข้อมูลด้วยอัลกอริทึมสำหรับเข้า- ถอดรหัส ทำให้สมาร์ตการ์ดชนิด Processor มีความเร็วสูงกว่าสมาร์ตการ์ดชนิด Memory หลายเท่า

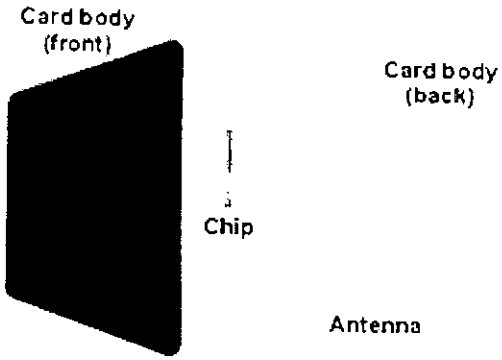


รูปที่ 2.9 บล็อกไดอะแกรมโครงสร้างภายในชิปสมาร์ตการ์ดชนิด Processor Card

ในการรับส่งข้อมูลของสมาร์ตการ์ดชนิดนี้จะใช้หน้าสัมผัสชนิดเดียวกันกับ สมาร์ตการ์ดชนิด Memory โดยสัญญาณนาฬิกาที่ป้อนจะถูกใช้เป็นสัญญาณนาฬิกาให้กับไมโครโปรเซสเซอร์ภายในสมาร์ตการ์ด ข้อมูลที่รับ ส่ง ข้อมูลเป็น 9600 บิต/วินาที ก็จะสามารถติดต่อกับไมโครโปรเซสเซอร์ ของชิป ได้แล้วแต่การเข้าถึงข้อมูลจะไม่สามารถทำได้เหมือนกับการ์ดชนิด Memory การเข้าถึงข้อมูลต้องกระทำผ่าน โปรเซสเซอร์ของสมาร์ตการ์ดเท่านั้น ไม่ว่าจะเป็นการอ่านหรือเขียนข้อมูลก็ตามเพราะหน่วยความจำจะอยู่ภายในควบคุมของโปรเซสเซอร์เพียงอย่างเดียว ข้อดีอีกอย่างที่ไม่สามารถติดต่อกับหน่วยความจำของชิปโดยตรงก็คือ การลอบเข้าถึงข้อมูลโดยไม่ได้รับอนุญาตแทบเป็นไปไม่ได้ยกเว้นมีความบกพร่องในการกำหนดเงื่อนไขในการเข้าถึงข้อมูลที่เป็นความลับ

2.6.4 Contact less Smart Card

สมาร์ตการ์ดชนิด contact less ใช้เทคโนโลยีสื่อสารผ่านทางคลื่นวิทยุ โดยการส่งคลื่นวิทยุความถี่ 13.56 MHz ที่ได้รับการมอดูเลตข้อมูลแล้วส่งให้กับชิปสมาร์ตการ์ต ทางด้านชิปสมาร์ตการ์ดจะใช้ขดลวด เป็นเสารับ ส่งสัญญาณ โดยเสารับส่งสัญญาณนี้จะเป็นขดลวดขนาดเล็กที่ถูกฝังอยู่ไหนตัวบัตร ภายนอกบัตรชนิดนี้แทบดูไม่ออกว่าเป็นบัตรสมาร์ตการ์ด

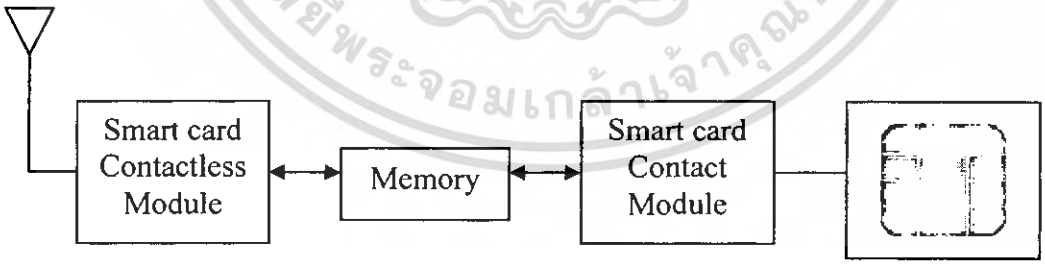


รูปที่ 2.10 Smart card Module ของ Contact less Smart Card

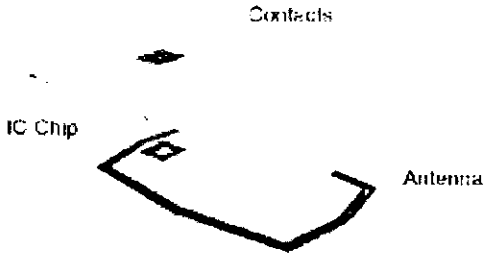
สมาร์ตการ์ดชนิดนี้ถูกออกแบบให้ใช้กระแสไฟฟ้าต่ำเพราะกระแสไฟฟ้าที่ได้จากคลื่นวิทยุเพียงเล็กน้อยทำให้สมาร์ตการ์ดแบบธรรมดาทั่วไปไม่สามารถทำงานได้ สมาร์ตการ์ดชนิดนี้รุ่นแรกๆ มาสามารถทำคำสั่งที่ซับซ้อนได้เช่นการเข้ารหัสข้อมูล หรือคำสั่งที่ใช้เวลาในการประมวลผลนานมากๆ แต่ปัจจุบันสามารถเข้ารหัสที่ยุ่งยากได้แล้วด้วยการเพิ่มวงจรสำหรับเข้ารหัสภายในชิป

2.6.5 Com-Bi Card

สมาร์ดการ์ดชนิดนี้เป็นการรวมเอาสมาร์ดการ์ดแบบ contact และสมาร์ตการ์ดชนิด contact less เข้าด้วยกัน โดยใช้หน่วยความจำข้อมูลเดียวกันเพื่อให้การทำรายการที่จำเป็นต้องอยู่ภายใต้การควบคุมและสามารถใช้งานทั่วไปได้อย่างสะดวกสบายผ่านทางคลื่นวิทยุ



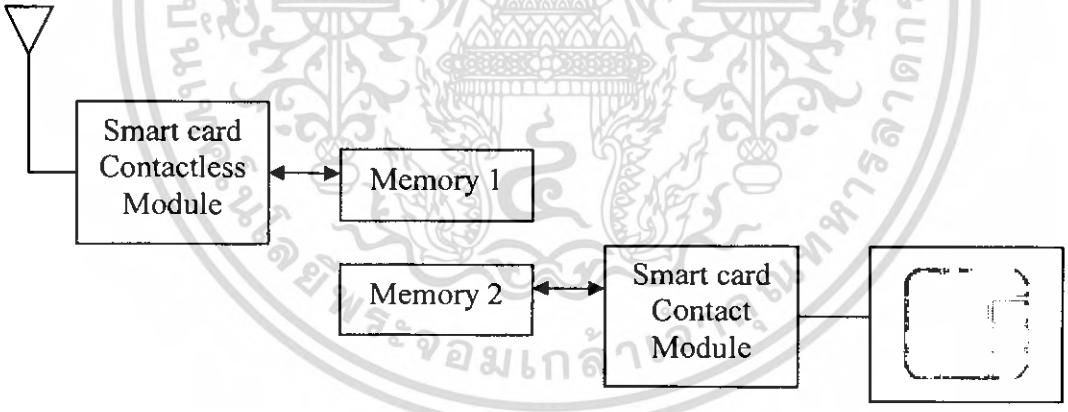
รูปที่ 2.11 โครงสร้างภายในของสมาร์ตการ์ดชนิด Com-Bi Card



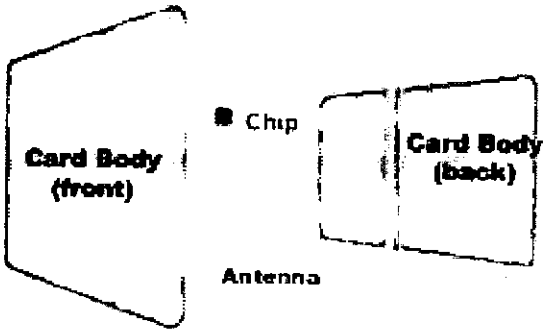
รูปที่ 2.12 Smart card Module ของ Com-Bi Card

2.6.6 Hybrid Smart Card

สมาร์ตการ์ดชนิดนี้มีลักษณะโครงสร้างภายในคล้ายกับประเภท Com-bi Card แต่จะแตกต่างกันในเรื่องของหน่วยความจำข้อมูล โดยหน่วยความจำข้อมูลจะถูกแยกจากกันอย่างสิ้นเชิงระหว่าง Contact และ Contact less เพื่อความสะดวกในการใช้งาน ซึ่งปัจจุบัน Hybrid Card จะมีความหมายรวมถึงบัตรที่มีคุณสมบัติในการใช้งานตั้งแต่สองอย่างขึ้นไป บัตรสมาร์ตการ์ดที่มีทั้งชิปสมาร์ตการ์ดและแถบแม่เหล็ก บัตรสมาร์ตการ์ดทั้งที่เป็น Contact และ Contact less



รูปที่ 2.13 โครงสร้างภายในของสมาร์ตการ์ดชนิด Hybrid Card



รูปที่ 2.14 Smart card Module ของ Hybrid Card

2.7 รูปแบบของสมาร์ตการ์ดที่นำมาใช้ในโครงงาน

ในโครงงานได้นำบัตรโทรศัพท์สาธารณะ TOT Card ที่ใช้งานหมดแล้วมาใช้ เป็นบัตรโทรศัพท์รูปแบบใหม่ที่ใช้เทคโนโลยีใหม่ล่าสุดของระบบโทรศัพท์สาธารณะ ที่ใช้อยู่ในปัจจุบัน โดยบัตรโทรศัพท์สาธารณะ TOT Card เป็นรูปแบบหนึ่งของ Memory Card เครื่องโทรศัพท์จะมีอุปกรณ์รักษาความปลอดภัย หรือ SAM (Security Access Module) ติดตั้งในตัวเครื่องเพื่อทำการตรวจสอบบัตร และมูลค่าบัตร จึงเป็นการป้องกันการนำบัตรปลอมมาใช้

2.7.1 บัตรโทรศัพท์สาธารณะ TOT Card

รูปที่ 2.15 ตัวอย่างบัตรโทรศัพท์ TOT

เป็นบัตรพลาสติกที่มีขนาดเท่ากับขนาดมาตรฐาน ของบัตรเครดิต ที่ใช้ในปัจจุบันซึ่งบรรจุชิป บันทึกข้อมูลดิดไว้บนหน้าบัตร สำหรับเป็นสื่อสัญญาณระหว่างบัตรกับเครื่องโทรศัพท์ที่ มีหัวอ่าน พร้อมระบบรักษาความปลอดภัย ซึ่งใช้ในการตรวจสอบความถูกต้องของข้อมูลในบัตร

องค์การโทรศัพท์ ใช้มาตรฐาน EURO Chip นำมาให้บริการ และเพื่อป้องกัน การลอกเลียนหรือปลอมแปลงบัตร องค์การโทรศัพท์จึงนำระบบ การสร้างรหัสหลัก Master Key อันเป็นลิขสิทธิ์ขององค์การโทรศัพท์ เป็นองค์ประกอบหลักในการเข้ารหัสข้อมูล เพื่อผลิตบัตรซึ่งสามารถป้องกันการปลอมแปลงบัตรโทรศัพท์ได้เป็นอย่างดี คุณสมบัติ ทนทานในทุกสภาพ อายุการใช้งาน 3 ปี โดยระบุที่ด้านหลังบัตรทุกใบ ชิปเป็นแบบ Memory Chip ซึ่งเมื่อใช้จนมูลค่าในบัตรหมดแล้ว ไม่สามารถเติมมูลค่าเงินลงไปในบัตรได้อีกทั้งนี้ในอนาคตอันใกล้ องค์การโทรศัพท์จะพัฒนาบริการไปสู่บริการกระเป๋าเงินสดอิเล็กทรอนิกส์ คือใบเดียวสามารถจับจ่ายใช้สอยหลายบริการทั่วประเทศ

TOT Card เป็น TOKEN Memory Card ภายในการ์ดชนิดนี้จะมีการเก็บข้อมูลในลักษณะการนับจำนวน ซึ่งจำนวนนับนี้จะเป็นตัวเลขแทนมูลค่าของเงินที่ระบุบนบัตร การนับเลขเป็นการนับถอยหลังเพื่อเป็นการนับมูลค่าคงเหลือภายในบัตร หมายความว่าหากใช้บัตรในการโทรศัพท์เรื่อยๆ มูลค่าในบอกก็จะถูกลดลงตามไปด้วยเช่นกัน ในการเข้าถึงข้อมูลสมาร์ตการ์ดชนิดนี้ต้องมีการแสดงรหัสผ่านให้บัตรรับทราบเหมือนกับ Memory Card แต่ไม่มี Bit Protect เท่านั้น การเข้าถึงข้อมูลในหน่วยความจำของสมาร์ตการ์ดชนิดนี้ต้องอ้างอิงกับแอดเดรสเสมอ ทั้งนี้ต้องขึ้นอยู่กับข้อกำหนดของสมาร์ตการ์ดแต่ละรุ่น แต่ใน TOKEN Memory Card นี้จะสามารถเข้าถึงและสามารถเปลี่ยนแปลงข้อมูลได้เพียงบางส่วนเท่านั้น โดยส่วนเราสามารถทำการเข้าถึงได้จะเป็นส่วนหมายเลขประจำของแต่ละบัตร (Serial Number) ซึ่งบัตร TOT Card แต่ละใบจะมีเลขประจำตัวประจำบัตรที่ไม่ซ้ำกัน

2.7.2 การจัดการหน่วยความจำภายใน

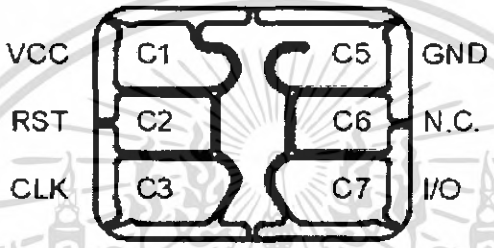
ในสมาร์ตการ์ดชนิด Memory มีการแบ่งวิธีการจัดการหน่วยความจำเป็นสองแบบคือ bitwise และ bytewise การจัดการหน่วยความจำแบบ bitwise เป็นการจัดการหน่วยความจำที่ใช้สมาร์ตการ์ดรุ่นแรกๆ การจัดการหน่วยความจำแบบนี้มักใช้บอกขนาดของหน่วยความจำของสมาร์ตการ์ดเป็นหน่วย บิต เช่น 1 กิโลบิต (128 ไบต์) , 8 กิโลบิต (1 กิโลไบต์), สาเหตุที่ bitwise จัดการหน่วยความจำข้อมูลเป็น บิต เนื่องจากข้อมูลที่ใช้รับ ส่ง ในสมาร์ตการ์ดชนิดนี้ทำกันในระดับบิต เท่านั้น หมายความว่าการรับส่งข้อมูลไม่จำเป็นต้องทำให้ครบทั้ง 8 บิต หรือ 1 ไบต์ การจัดการหน่วยความจำแบบนี้สามารถอ่านข้อมูลที่บิตใดก็ได้ ซึ่งมีใช้ในสมาร์ตการ์ดที่มีหน่วยความจำข้อมูลไม่มากกนัก

สำหรับการจัดการหน่วยความจำข้อมูลแบบ bytewise เป็นการจัดการหน่วยความจำที่อ้างถึงข้อมูลขนาด 8 บิต์ หรือ 1 ไบต์เต็ม การรับ ส่งข้อมูลกับชิปต้องทำการรับส่งให้ครบทั้ง 8 บิตจึงจะทำให้การรับส่งเสร็จสมบูรณ์(หากทำไม่สมบูรณ์ ชิปสมาร์ตการ์ดจะยกเลิกการรับ ส่งข้อมูลครั้งนั้นๆ) นอกจากนี้การอ้างอิงหน่วยความจำยังแตกต่างกันเช่น บางผู้ผลิตกำหนดให้แอดเดรสที่ต่อเนื่องกัน

ตั้งแต่แอดเดรสที่ 0 ถึงแอดเดรสสุดท้าย บางผู้ผลิตแบ่งหน่วยความจำออกเป็น เพจ (Page) แต่ละเพจมีขนาดแตกต่างกันขึ้นกับรุ่นที่ผลิต ทำให้การอ้างอิงถึงข้อมูลใดๆ ในหน่วยความจำของสมาร์ตการ์ดแต่ละแบบไม่เหมือนกัน ตามแต่ผู้ผลิตจะออกแบบ

2.7.3รูปแบบของสมาร์ตชิปในบัตร TOT

ในบัตร TOT Card จะมีหน้าสัมผัสอยู่ 6 หน้า โดยจะไม่มีขา RFU ทั้งสองข้าง



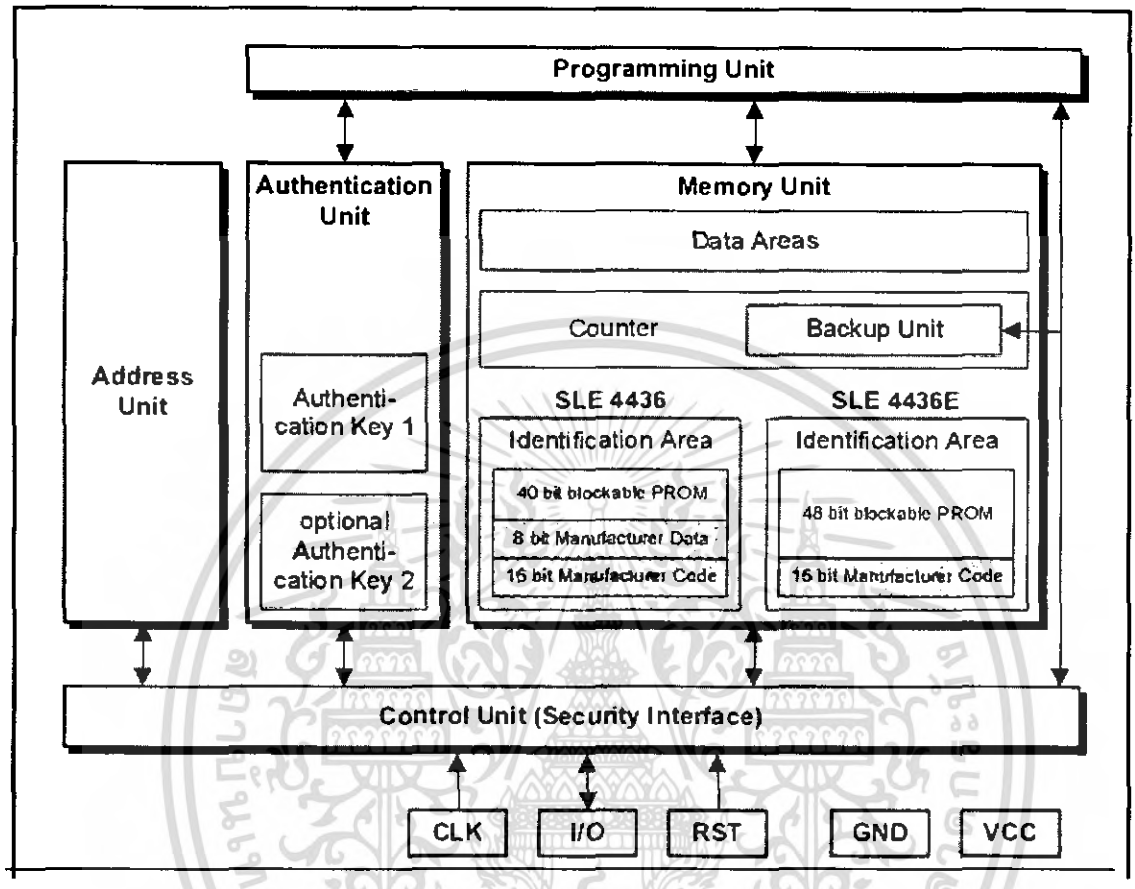
รูปที่ 2.17 หน้าสัมผัสของของบัตร TOT

บัตรโทรศัพท์สาธารณะ TOT Card เป็นสมาร์ตการ์ดแบบ Synchronous Card เบอร์ SLE 4436 โดยจะเป็นรูปแบบของเดบิตการ์ดหรือบัตรชนิดพื้นฐานที่นำไปใช้เพื่อเป็นมูลค่าแทนเงินสดในการใช้งานโทรศัพท์สาธารณะ ภายในจะมีการบันทึกข้อมูลในรูปของ Unit Counter อยู่ภายในส่วนของ หน่วยบันทึกข้อมูลภายในบัตร (Memory Unit) โดยหลังจากการบันทึกข้อมูลลงในบัตรโดยผู้ผลิตแล้วข้อมูลส่วนหนึ่งเช่น Customer Code , หมายเลขบัตร , วันหมดอายุของบัตร จะถูกเขียนลงไปอย่างถาวรไม่สามารถแก้ไขได้ ในขณะที่ข้อมูลอีกส่วนหนึ่งคือ Unit Counter จะสามารถลดค่าลงเพียงอย่างเดียวมาสามารถเพิ่มค่าได้ โครงสร้างของส่วนที่เก็บข้อมูล Unit Counter เป็นฟิวส์ขนาด 40 บิต (Logical Fuse) จากการใช้งานข้อมูลของ Unit Counter จะถูกลดค่าลงจนเป็น 0 แต่ข้อมูลอื่นๆ ในบัตรยังคงอยู่ ในโครงงานนี้จะใช้ข้อมูลที่ยังคงอยู่นี้มาประยุกต์ใช้ในงานระบบรักษาความปลอดภัย

2.7.4โครงสร้างภายในบัตร SLE 4436

บัตร SLE 4436 บัตรชนิดนี้ถูกออกแบบมาเพื่อการใช้บริการพรีเพด (Pre-Paid) หรือบริการจ่ายเงินก่อนค่อยใช้บริการโครงสร้างภายในชิป ไอซีของบัตร SLE 4436 โดยทั่วไปจะประกอบไปด้วยหน่วยความจำแบบ EEPROM ขนาด 221 บิต หน่วยความจำ ROM ขนาด 16 บิต ,ส่วนควบคุม

และรักษาความปลอดภัยให้ข้อมูล (Control Security Unit) ,ส่วนประมวลผลเฉพาะสำหรับการรับรอง (Authentication)



รูปที่ 2.18 ไดอะแกรมแสดงส่วนการทำงานภายในสมาร์ตการ์ดตระกูล SLE 4436

สำหรับบัตร SLE 4436 ที่ถูกนำมาผ่านกระบวนการเพื่อใช้เป็นบัตร TOT Card ภายในหน่วยบันทึกข้อมูล(Memory Unit) จะถูกบรรจุไว้เป็นข้อมูลขนาด 48 ไบต์ ข้อมูลจะถูกบันทึกตั้งแต่ในกระบวนการผลิตของโรงงานไม่สามารถเข้าไปเปลี่ยนแปลงหรือแก้ไขข้อมูลได้ ที่สามารถแบ่งออกเป็น 5 ส่วนได้แก่

- ข้อมูลชุดที่ 1 มีขนาด 3 ไบต์ เป็นข้อมูลที่ระบุถึง Factory Code มีไว้สำหรับระบุข้อมูลของบัตรไปสร้างแอพพลิเคชั่นเพื่อให้บริการ เช่น ถ้าต้องการใช้เกี่ยวกับการใช้บัตรนี้ทำบริการเครื่องขายสินค้าอัตโนมัติก็จะมีการระบุรหัสประจำตัวของผู้ให้บริการ เพื่อให้สามารถแยกแยะตัวผู้ให้บริการได้

- ข้อมูลชุดที่ 2 มีขนาด 5 ไบต์ เป็นข้อมูลหมายเลขบัตรซึ่งถูกเก็บอยู่ในรูปรหัส BCD โดยเป็นตัวเลขขนาด 10 หลัก

- ข้อมูลชุดที่ 3 มีขนาด 5 ไบต์ เป็น Balance Unit หรือมูลค่าตัวเงินของบัตร การนับมูลค่าเงินหรือ Balance Counter จะมีวิธีการคำนวณตามวิธีการใช้งาน

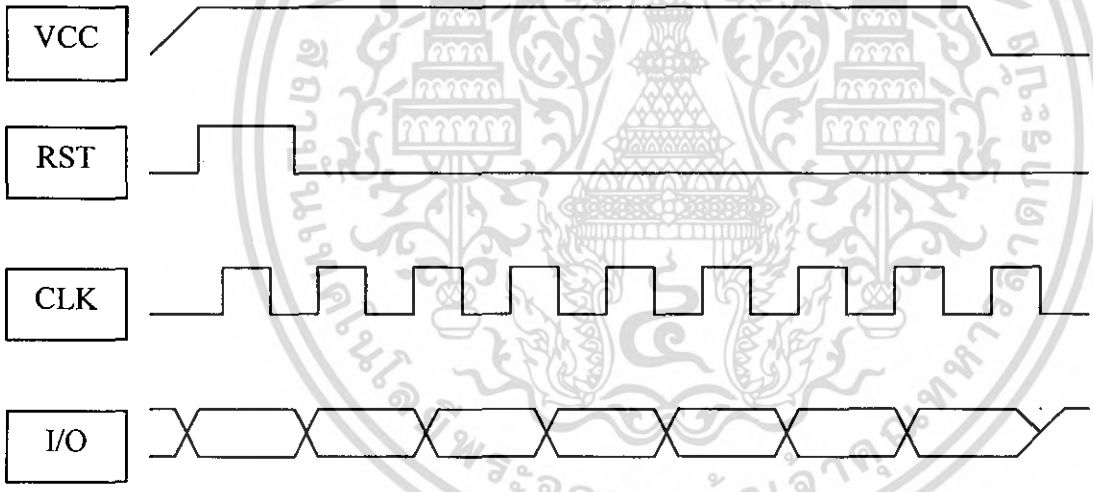
- ข้อมูลชุดที่ 4 มีขนาด 32 ไบต์ เป็นข้อมูลลับของทางผู้ผลิตสมาร์ดการ์ด

- ข้อมูลชุดที่ 5 มีขนาด 1 ไบต์ เป็นข้อมูลของ ปี และ เดือน ที่บัตรจะหมดอายุ

ข้อมูลไบต์ที่ 0-2	ข้อมูลไบต์ที่ 3-7	ข้อมูลไบต์ที่ 8-12	ข้อมูลไบต์ที่ 13-46	ข้อมูลไบต์ที่ 47
Customer Code	Card Number	Balance Counter	ข้อมูลที่เป็นความลับของผู้ผลิต	วันหมดอายุของบัตร

ตารางที่ 2.2 โครงสร้างและรายละเอียดที่เกี่ยวกับข้อมูลทั้ง 48 ไบต์ ในบัตร SLE 4436

2.7.5การติดต่อกับการ์ด SLE 4436



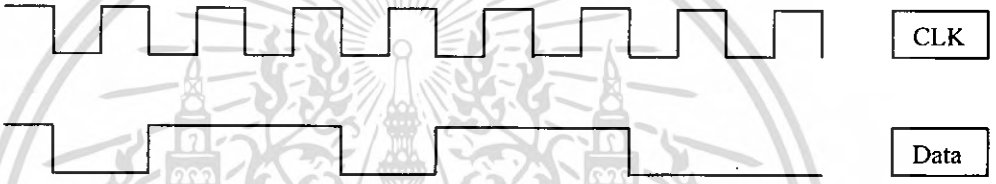
รูปที่ 2.19 แผนผังทางเวลาของสัญญาณที่เกี่ยวข้องสำหรับการอ่านข้อมูลจากบัตร SLE 4436

จ่ายไฟดรงให้แก่บัตร กำหนดให้สัญญาณที่ขา Clock เป็นลอจิกสูง กำหนดให้สัญญาณที่ขา RST เป็นลอจิกสูงประมาณ 10 uS แล้วให้ลอจิกต่ำที่ขาเป็นเวลา 10 uS จากนั้นกำหนดให้สัญญาณที่ขาClock เป็นลอจิกสูงค้างไว้นานประมาณ 5 uS ให้อ่านข้อมูลจากขา I/O โดยเลื่อนบิตข้อมูลไปทีละ 1 บิต ซึ่งหมายความว่าเมื่อป้อนสัญญาณ Clock ครบ 8 ลูก อ่านข้อมูลจากขา I/O เลื่อนข้อมูลและเก็บค่าจนครบ 8 บิต ข้อมูลที่ได้มาสุดท้ายก็คือข้อมูลขนาด 1 ไบต์ (ข้อมูลบิตแรกที่ได้ เป็น MSB) ใช้รูปแบบของสัญญาณตามที่ได้กล่าวมาจนได้สัญญาณที่ขา Clock ครบ 384 ลูก

บทที่ 3 การสื่อสารแบบอนุกรม

3.1 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นจะแบ่งออกได้เป็น 2 แบบ คือการสื่อสารอนุกรมแบบซิงโครนัส และการสื่อสารแบบอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งแบบซิงโคนัสก็คือ คีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นหนึ่งจะเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้ต้องใช้สายเชื่อมต่ออย่างน้อยที่สุด 3 เส้น คือ สัญญาณนาฬิกา , ข้อมูล ,และกราวด์



รูปที่ 3.1 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

3.2 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโคนัส คือ การรับและส่งข้อมูลไปในสายโดยไม่จำเป็นต้องมีสัญญาณนาฬิกา ร่วมด้วยเหมือนแบบการรับส่งข้อมูลแบบซิงโคนัส แต่จะใช้กำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให่ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรด (Baudrate) มีหน่วยเป็น บิตต่อวินาที (bit per second : bps)

รูปแบบข้อมูลที่ใช้ในการรับส่งแบบอะซิงโคนัสประกอบด้วย 4 ส่วนด้วยกัน คือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลอนุกรมจะมีขนาด 5 , 6, 7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิต หรือไม่มี
4. บิตปิดท้าย(Stop Bit) จะมีขนาต 1 , 1.5 หรือ 2 บิต

เมื่อไม่มีข้อมูลที่จะส่ง ขาดาต้า จะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (waiting stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา ดาต้า มีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต เรียกบิตนี้ว่า บิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไปโดยเริ่มจากบิตที่มีนัยสำคัญต่ำที่สุด(LSB)

ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต อาจจะมีจำนวนบิต 5 , 6, 7 หรือ 8 บิตก็ได้ จากนั้นตามด้วย บิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่ส่งคือ บิตปิดท้าย ซึ่งจะให้ขาดค้ามี ลอจิก "1" อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต , 1.5 บิด หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับรับและส่งข้อมูลแบบอะซิงโคนัสเรียกว่า Universal Asynchronous Receiver/Transmitter หรือ UART อัตราเร็วในการส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโคนัส คือ บอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรด มาตรฐานที่ใช้สำหรับ พอร์ตอนุกรม RS – 232 ได้แก่ 110 , 150 , 300 , 600 , 1200 , 2400 , 4800 , 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มขึ้นตามเทคโนโลยีของคอมพิวเตอร์ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่าน โมเด็ม อาจจะสามารถกำหนดค่า บอดเรด ได้สูงถึง 115200 บิตต่อวินาที เนื่องจาก บอดเรด คือจำนวนบิตของข้อมูลที่สามารถส่งได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบ พาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูลที่รับส่งเท่ากับ 10 บิด ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็สามารถรับส่งข้อมูลได้ด้วยอัตราเร็ว 960 ไบต์ต่อนาที และถ้ามีการใช้บิต พาริตี้ ความเร็วในการรับส่งจะเหลือ 872 ไบต์ต่อวินาที

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (Odd) , แบบคู่ (Even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิกสูง ภายในข้อมูลที่ส่งไป 1 ไบต์ ว่ามีจำนวนเป็นเลขคู่หรือเลขคี่ โดยต้องรวม บิตพาริตี้เข้าไปด้วย ยกตัวอย่าง ข้อมูลที่จะทำการส่งมีขนาด 8 บิด และมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสองจะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิกเป็น "0" แต่ถ้าพาริตี้เป็นคี่ ค่าที่บิตพาริตี้จะต้องเป็น "1" เพื่อให้ข้อมูล 1 ไบต์รวมทั้ง บิดพาริตี้มีจำนวนบิตที่ลอจิก "1" รวมกันเป็นเลขคี่

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART โดยภาครับจะต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ ให้ตรงกันว่าจะตรวจสอบพาริตี้คี่หรือคู่ จากนั้นภาครับของ UART จะตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแจ้งข้อผิดพลาดให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดของการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิดเดียวเท่านั้น ถ้าข้อมูลที่ส่งมีข้อผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้เป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

ข้อมูล	บิตพาริตี้คู่	บิตพาริตี้คี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	1	0
11111111	0	1

ตารางที่ 3.1 แสดงบิตพาริตี้ของข้อมูล

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART เบอร์ 8250 UART ชิปเหล่านี้มีระดับแรงดันแบบทีทีแอล (0 และ +5V) แต่เพื่อให้แรงดันเป็นมาตรฐาน RS- 232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอล จะถูกแปลงเป็นแรงดันที่สูงขึ้นโดยลอจิก "0" มีระดับแรงดัน + 3 V ถึง + 12 V ในขณะที่ ลอจิก "1" มีระดับแรงดัน -3 V ถึง -12 V

3.3 มาตรฐานพอร์ตอนุกรมแบบ RS- 232

มาตรฐานการเชื่อมต่อแบบอนุกรม RS- 232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อส่งข้อมูลแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS- 232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้เพื่อสื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดซึ่งอยู่ไกลกัน โดยคณะกรรมการที่เรียกว่า สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ได้วางมาตรฐานที่มีชื่อเรียกกันว่า EIA RS- 232 มาตรฐานนี้ใช้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB – 25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3 ถึง -12 V แสดงว่ามีข้อมูล (Mask) และ +3 ถึง +12 V แสดงว่าเป็นช่องว่าง (Spacc)

มาตรฐาน RS- 232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่า อุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัว เช่น ไมโครคอนโทรลเลอร์ หรือ ไมโครคอมพิวเตอร์

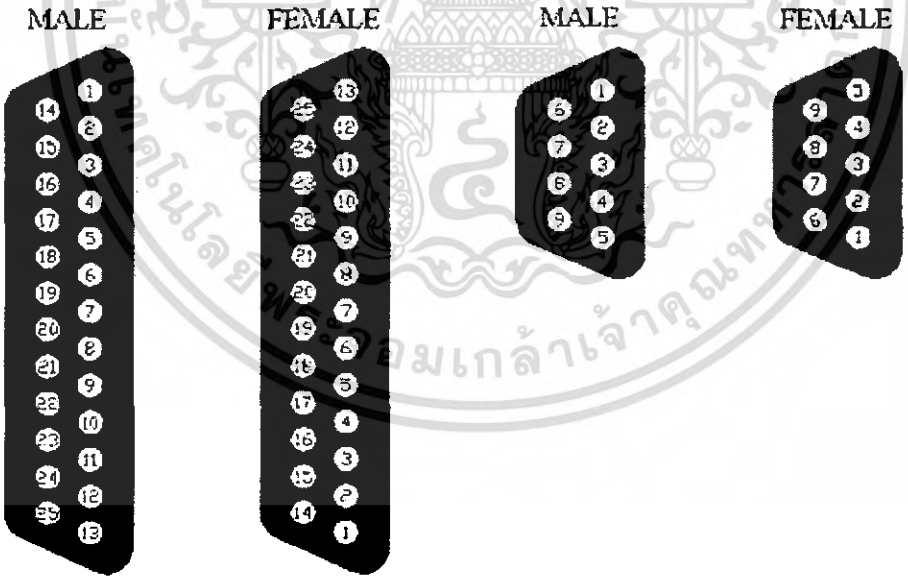
ซึ่งมีความสามารถในการสร้างข้อมูลอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่ เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS- 232

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัดคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ในโมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานบนคอมพิวเตอร์พอร์ตอนุกรม RS- 232 มักถูกเชื่อมต่อกับโมเด็มหรือเมาส์ โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุดถึง 20 เมตร

3.3.1 คอนเน็กเตอร์สำหรับพอร์ต RS- 232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS- 232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ DB – 9 ตัวผู้ ซึ่งคอนเน็กเตอร์แบบ DB – 25 จะมีขาต่อใช้งานเพียง 9 เส้น เช่นเดียวกับคอนเน็กเตอร์แบบ DB 9 เนื่องจากขาอื่นๆที่เคยใช้งานในอดีต ปัจจุบันมีใช้ไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาในรูปที่ 3.2 และตารางที่ 3.2



รูปที่ 3.2 คอนเน็กเตอร์อนุกรม

คอนเน็กเตอร์ DB-9	คอนเน็กเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	Data Carrier Detect : DCD	อินพุท
2	3	Received Data : RxD	อินพุท
3	2	Transmitted Data : TxD	เอาท์พุท
4	20	Data Terminal Ready : DTR	เอาท์พุท
5	7	Signal Ground : GND	-
6	6	Data Set Ready : DSR	อินพุท
7	4	Request To Send : RTS	เอาท์พุท
8	5	Clear To Send : CTS	อินพุท
9	22	Ring Indicator : RI	อินพุท

ตารางที่ 3.2 หน้าที่การทำงานของขาอนุกรม

สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกแสดงดังในรูปที่ 3.3 ลูกศรในรูปแสดงทิศทางการไหลของข้อมูลซึ่งจากรูปเป็นการต่อแบบ Null Model หรือการเชื่อมต่อโดยไม่ต้องผ่านโมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็คเด็มรูปแบบ และถ้าต้องการต่อลักษณะที่ใช้สายสัญญาณ 3 เส้น จะต่อโดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้น สำหรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ สำหรับรายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS – 232 มีดังนี้

- Data Carrier Detect : DCD หรืออาจเรียกว่า Carrier Detect : CD ขานี้จะแอกทีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูล เช่น โมเด็ม สำหรับการใช้งานแบบปรกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก

- Receive Data : RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์โดยนำข้อมูลที่อ่านได้ เก็บไว้ใน รีจิสเตอร์บัฟเฟอร์

- Transmitted Data : TD หรือ TxD ใช้ส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

- Data Terminal Ready : DTR เป็นขาสัญญาณที่ส่งออกมาจากคอมพิวเตอร์เพื่อให้อุกรณ์ปลายทางรับรู้ว่า ต้องการดิดต่อด้วย โดยขา DTR นี้ต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ ปลายทาง และขา DTR ของอุปกรณ์ปลายทางต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ถ้าใช้การเชื่อมต่อเป็นแบบ Null Model ซึ่งใช้สายในการเชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DSR ของตัวเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

- Signal Ground : GND กราวด์ระบบ

- Data Set Ready : DSR ขานี้จะใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR

- Request To Send : RTS เป็นขอสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทาง ส่งข้อมูลมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้ในการเชื่อมต่อแบบ Null Model 3 สาย จะต้องเชื่อมต่อขา RTS และ CTS ของตัวมันเองเข้าด้วยกันเพื่อจะให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

- Clear To Send : CTS สายนี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ข้อมูลที่ขา TxD จะถูกส่งออกไป ดังนั้นขานี้จึงใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

- Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ปกติในการสื่อสาร โดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการใช้งานกับโมเด็มและโปรแกรมมีการตรวจสอบสายสัญญาณนี้เท่านั้น

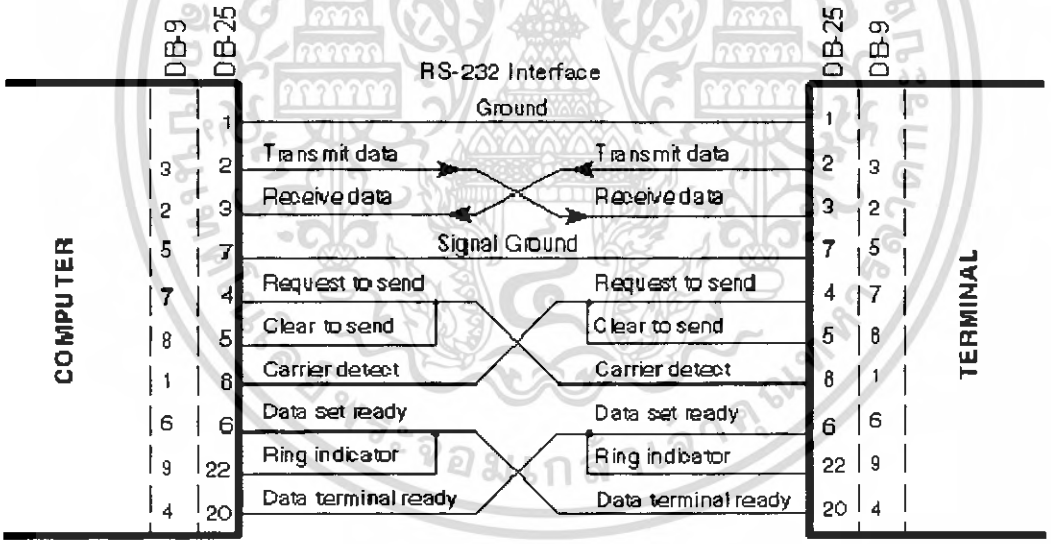


Figure 1. Direct-to-computer RS-232 Interface

รูปที่ 3.3 การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์

3.3.2 UART (Universal Asynchronous Receiver Transmitter)

เป็นอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโคนัสนั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโคนัส แล้วส่งออกไป และทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโคนัสที่ป้อยเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจ้งข้อมูลอื่นๆให้คอมพิวเตอร์รับทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต) , รูปแบบการส่งข้อมูล , ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพาริตี้ , เฟรมข้อมูล , โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ (Programmable buadrate generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกา UART โดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารให้อยู่ในช่วง 1 – 65,535 UART สามารถรับส่งข้อมูลได้ทั้งแบบ Half Duplex และ Full Duplex โดยการส่งแบบ Half Duplex เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบ Full Duplex เป็นการส่งที่สามารถรับและส่งได้ในคราวเดียวกัน

3.3.3 ชนิดของ UART

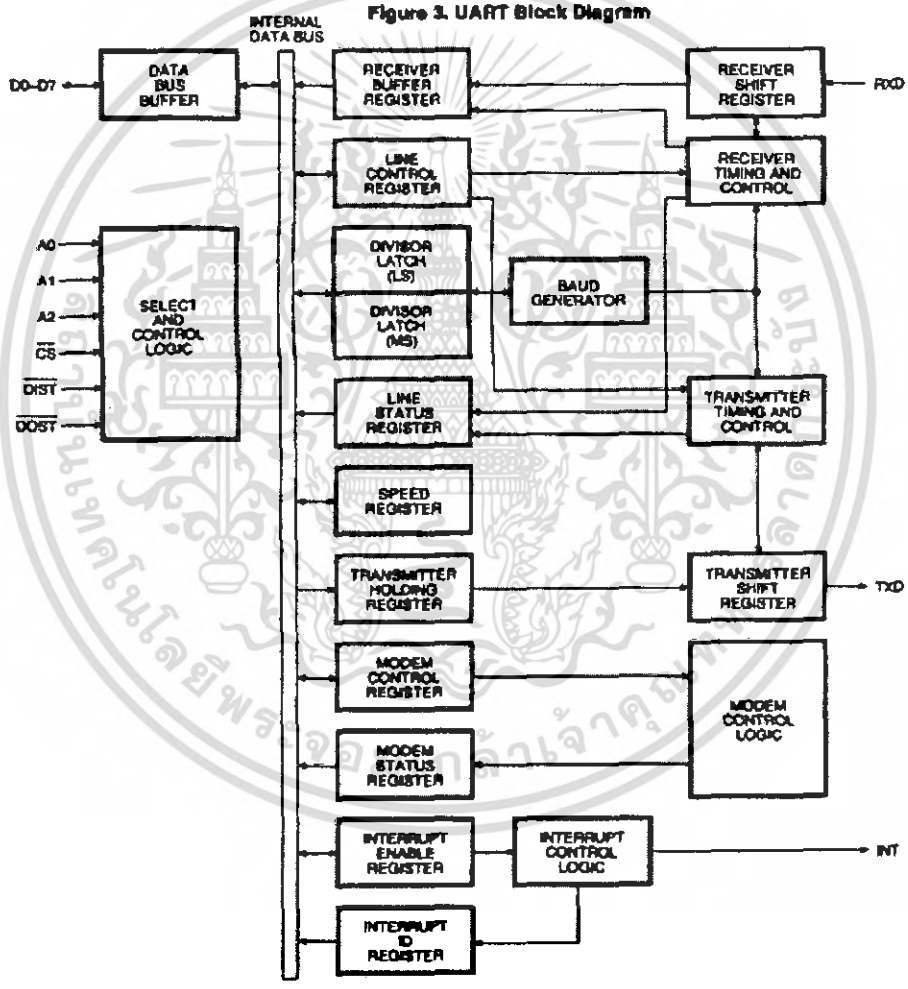
ในเครื่องคอมพิวเตอร์ทั่วไปมี UART ที่ใช้งานกันอยู่ 2 เบอร์คือ 8250 ซึ่งเป็น UART มาตรฐานที่ใช้กันมายาวนาน UART เบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเดียวกันทำให้การรับและส่งข้อมูลถูกจำกัดอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้นแต่ UART เบอร์นี้ก็ถือว่าเป็นต้นแบบของ UART ที่ใช้ในคอมพิวเตอร์ โดยคอมพิวเตอร์ทุกๆรุ่นจะต้องสนับสนุนการทำงานตามแบบของ UART เบอร์นี้

UART อีกเบอร์หนึ่งคือ 16450 มีความสามารถในการรับ ส่งข้อมูลที่ความเร็ว 115,200 บิตต่อวินาที และเพิ่มรีจิสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิพรีจิสเตอร์แบบ FIFO (First IN First Out) ขนาด 16 ไบต์เข้าไปทำให้สนับสนุนในการส่งข้อมูลความเร็วที่ 256 กิโลบิตต่อวินาทีได้ โดยคอมพิวเตอร์ในปัจจุบันใช้ UART เบอร์นี้หรือใหม่กว่า เช่นเบอร์ TL16C 750 ซึ่งมีรีจิสเตอร์แบบ FIFO ขนาด 64 ไบต์ ทำงานได้ที่ระดับแรงดัน +5V และ +3V มีโหมดประหยัดพลังงาน สามารถรับส่งข้อมูลที่ความเร็ว 1 เมกะบิตต่อวินาทีเมื่อใช้สัญญาณนาฬิกา 16 MHz

อย่างไรก็ตามความเร็วในการส่งข้อมูลที่มากมายของ UART เบอร์ใหม่ๆ ก็ไม่ได้ช่วยให้การรับ ส่งข้อมูลของคอมพิวเตอร์เร็วขึ้น เนื่องจากคอมพิวเตอร์ยังใช้ความถี่ของสัญญาณนาฬิกาในการแปลงข้อมูลเพียง 1.8432 MHz เท่านั้น

3.4 วงจรภายในและรีจิสเตอร์ของพอร์ตอนุกรม

เครื่องคอมพิวเตอร์ทั่วไปสามารถอ่านพอร์ตอนุกรมสูงสุดได้ 4 พอร์ต มีชื่อเรียกเป็น COM1 , COM2 , COM3 และ COM4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างก็ใช้งาน UART ภายในคอมพิวเตอร์ติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน



รูปที่ 3.4 ไดอะแกรมการทำงานภายในของพอร์ตอนุกรมของคอมพิวเตอร์

ภายในจะประกอบด้วย รีจิสเตอร์ 8 บิต 8 ตัว ที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอร์ภายในพอร์ตอนุกรมสามารถคำนวณได้จากค่ารีจิสเตอร์พื้นฐานของพอร์ตอนุกรม ยกตัวอย่าง พอร์ตอนุกรม COM1 มีแอดเดรสอยู่ที่ 3F8H ตำแหน่งของรีจิสเตอร์ต่างๆ จะเป็นตำแหน่งที่บวกไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ตอนุกรมมีดังนี้

00H รีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้าหรือเตรียมข้อมูลก่อนที่จะส่งออกไป
01H รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัป ใช้เซ็ตโหมดการอินเตอร์รัป ของพอร์ตอนุกรม
02H รีจิสเตอร์แสดงโหมดการอินเตอร์รัปใช้เพื่อตรวจสอบโหมดการอินเตอร์รัป
03H รีจิสเตอร์กำหนดรูปแบบข้อมูล
04H รีจิสเตอร์ควบคุมโมเด็ม ใช้ตรวจสอบบิตสำหรับติดต่อโมเด็ม เช่น RTS หรือ DTR
05H รีจิสเตอร์แสดงสถานการณ์รับ และการส่งข้อมูลแบบอนุกรม
06H รีจิสเตอร์แสดงสถานะโมเด็ม ซึ่งแสดงสถานะของขา DCD , RI , DSR และ CTS
07H รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว

3.4.1 รีจิสเตอร์ตำแหน่ง 00H (รีจิสเตอร์บัฟเฟอร์)

เป็นรีจิสเตอร์เก็บข้อมูลที่รับเข้ามาและส่งออก โดยการติดต่อกับรีจิสเตอร์นี้เพื่อเก็บข้อมูลจะต้องกำหนดให้บิด DLAB ในรีจิสเตอร์กำหนดรูปแบบข้อมูล (03H) มีสถานะเป็น “0” ซึ่งการเขียนข้อมูลมายังแอดเดรสนี้เป็นการส่งข้อมูลไปยังรีจิสเตอร์ส่งข้อมูลและข้อมูลจะส่งออกไปแบบอนุกรม สำหรับการรับข้อมูลเมื่อรับมาแล้ว จะส่งไปยังรีจิสเตอร์เก็บข้อมูลหลังจากอ่านค่าจากรีจิสเตอร์นี้ออกไปรีจิสเตอร์นี้จะถูกเคลียร์และเตรียมพร้อมสำหรับข้อมูลที่จะรับไบต์ต่อไป

3.4.2 รีจิสเตอร์ตำแหน่ง 01H (รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัป)

เป็นรีจิสเตอร์สำหรับการเอ็นเอเบิลการอินเตอร์รัป ซึ่งเป็นการกำหนดให้ UART สร้างสัญญาณอินเตอร์รัปขึ้นมา ฟังก์ชันการทำงานในแต่ละบิตของรีจิสเตอร์มีดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 1
0	0	0	0	SINP	ERBK	TBE	RxRD

ตารางที่ 3.3 ฟังก์ชันการทำงานของรีจิสเตอร์ตำแหน่ง 01H

บิต 4 – 7 บิตเหล่านี้ไม่ถูกใช้งาน กำหนดให้เท่ากับ “0”

SINP เอ็นเอเบิลการอินเตอร์รัปเนื่องจากเกิดจากการเปลี่ยนสถานะที่ขาอินพุท CTS , DSR , DCD หรือ RI

“1” เอ็นเอเบิลการอินเตอร์รัป

“0” ดิสเอเบิล

ERBK เอ็นเอเบิลการอินเตอร์รัปเนื่องจากเกิดความผิดพลาดขึ้นด้วยสาเหตุจาก พาริตี้ , โอเวอร์รัน , เฟรมข้อมูล หรือการเบรกข้อมูล

“1” เอ็นเอเบิลการอินเตอร์รัป

“0” ดิสเอเบิล

TBE เอ็นเอเบิลการอินเตอร์รัปเนื่องจากรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

“1” เอ็นเอเบิลการอินเตอร์รัป

“0” ดิสเอเบิล

RxRD เอ็นเอเบิลการอินเตอร์รัปเนื่องจากรีจิสเตอร์บัฟเฟอร์ได้รับข้อมูลแล้ว

“1” เอ็นเอเบิลการอินเตอร์รัป

“0” ดิสเอเบิล

3.4.3 รีจิสเตอร์ตำแหน่ง 02H (รีจิสเตอร์แสดงโหมดการอินเตอร์รัป)

มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 1
0	0	0	0	0	ID1	ID0	PND

ตารางที่ 3.4 ฟังก์ชันการทำงานของรีจิสเตอร์ตำแหน่ง 02H

บิต 3-7 ไม่ได้ใช้งานอ่านค่าได้เท่ากับ “0”

ID1 , ID0 ใช้งานร่วมกันเพื่อแจ้งสาเหตุของการอินเตอร์รัป

“00” เกิดการอินเตอร์รัปเนื่องจากการเปลี่ยนแปลงขาอินพุตขึ้น การอินเตอร์รัปแบบนี้มีนัยสำคัญเป็นอันดับ 4

“01” เกิดการอินเตอร์รัปเนื่องจากรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง การอินเตอร์รัปแบบนี้มีนัยสำคัญเป็นอันดับ 3

“10” เกิดการอินเตอร์รัปเนื่องจากข้อมูลถูกเก็บลงรีจิสเตอร์บัฟเฟอร์สำหรับข้อมูลเรียบร้อยแล้ว การอินเตอร์รัปแบบนี้มีนัยสำคัญเป็นอันดับ 3

"11" เกิดการอินเตอร์รัปเนื่องจากความผิดพลาดในการถ่ายทอดข้อมูล หรือเกิดการเบรก (Break : เกิดการหยุดถ่ายเทข้อมูลกะทันหัน) การอินเตอร์รัปแบบนี้มีนัยสำคัญเป็นอันดับ 1 หรือมีนัยสำคัญสูงสุด

PND ใช้แสดงสถานะของการเกิดอินเตอร์รัป

"1" แสดงว่าไม่มีการอินเตอร์รัป

"0" แสดงว่ามีการอินเตอร์รัป

เมื่อมีการสร้างสัญญาณอิรเตอร์รัปขึ้น จะต้องมีการเคลียร์ค่าก่อนที่จะให้เกิดการอินเตอร์รัปครั้งต่อไป โดยสามารถทำได้ดังนี้

- ถ้าเกิดการอินเตอร์รัปเนื่องจากการเปลี่ยนแปลงขาอินพุตจะต้องอ่านค่ารีจิสเตอร์แสดงสถานะของโมเด็ม (รีจิสเตอร์ตำแหน่ง 06H) เพื่อเคลียร์ค่าการอินเตอร์รัป
- ถ้าเกิดการอินเตอร์รัปเนื่องจากรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง จะต้องเขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ข้อมูล (รีจิสเตอร์ตำแหน่ง 00H) หรืออ่านค่ารีจิสเตอร์แสดงสถานะอินเตอร์รัป (รีจิสเตอร์ตำแหน่ง 02H) เพื่อเคลียร์ค่าการอินเตอร์รัป
- ถ้าเกิดการอินเตอร์รัปเนื่องจากข้อมูลถูกเก็บลงรีจิสเตอร์บัฟเฟอร์สำหรับข้อมูลเรียบร้อยแล้ว จะต้องเคลียร์ค่าการอินเตอร์รัปโดยการอ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์
- ถ้าเกิดการอินเตอร์รัปเนื่องจากความผิดพลาดในการถ่ายทอดข้อมูลหรือเกิดการเบรก จะต้องเคลียร์ค่าอินเตอร์รัปโดยการอ่านค่ารีจิสเตอร์แสดงสถานการณ์รับหรือการส่งข้อมูลอนุกรม

3.4.4 รีจิสเตอร์ตำแหน่ง 03H (รีจิสเตอร์กำหนดรูปแบบข้อมูล)

มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 1
DLAB	BRK	PAR2	PAR1	PAR0	STOP	DAB1	DAB0

ตารางที่ 3.5 ฟังก์ชันการทำงานของรีจิสเตอร์ตำแหน่ง 03H

DLAB ใช้ในการกำหนดหน้าที่การทำงานของรีจิสเตอร์บัฟเฟอร์ (00H)

"1" เป็นการเข้าสู่โหมดการหารค่าบอดเรต

"0" เป็นการเข้าถึงรีจิสเตอร์บัฟเฟอร์ (รีจิสเตอร์ตำแหน่ง 00H) และรีจิสเตอร์สำหรับการเอ็นเอเบิลการอินเตอร์รัป (รีจิสเตอร์ตำแหน่ง 01H)

เมื่อบิต DLAB เป็น "1" รีจิสเตอร์บัฟเฟอร์ (00H) และรีจิสเตอร์สำหรับการเอ็นเอเบิลการอินเตอร์รัป (01H) จะใช้สำหรับโหลดค่าการหารความถี่สำหรับกำหนดค่าบอดเรต โดยรีจิสเตอร์ 00H เก็บค่าตัวหารไบต์ต่ำ ส่วนรีจิสเตอร์ 01H ใช้เก็บตัวหารไบต์สูง การหาค่าบอดเรตสามารถเขียนเป็นสมการได้ดังนี้

บอดเรต = 115200/ค่าตัวหาร 16 บิต

ค่าตัวเลข 115200 มาจากความถี่คริสตัลในวงจร UART ภายในเครื่องคอมพิวเตอร์ โดยคริสตัลที่ใช้มีความถี่ 1.8432 MHz วงจรภายใน UART จะหารค่าความถี่นี้ด้วย 16 ทำให้ได้ค่าความถี่ 115200 Hz ออกมา ค่าตัวหาร 16 บิต = ข้อมูลในรีจิสเตอร์ 00H + (256 * ข้อมูลในรีจิสเตอร์ 01H) ถ้าต้องการบอดเรตเท่ากับ 9600 ค่าตัวหารที่ใช้จะต้องมีค่าเท่ากับ 12 ซึ่งค่านี้จะถูกเขียนลงในรีจิสเตอร์ 00H และเขียนค่า 0 ลงไปใน รีจิสเตอร์ 01H ค่าตัวหารที่ทำให้เกิดค่าบอดเรตสูงสุดที่ 115200 บิตต่อวินาที คือ ค่า 0001 นั่นคือค่า รีจิสเตอร์ 00H มีค่าเท่ากับ 1 และรีจิสเตอร์ 00H มีค่าเท่ากับ 0

BRK ใช้ควบคุมการหยุดถ่ายทอดข้อมูล

"1" สามารถหยุดหรือเบรกได้

"0" ไม่มีการหยุดหรือเบรกได้

PAR2,PAR1,PAR0 ใช้เพื่อควบคุมบิตพาริตี้ได้

"000" ไม่ใช้บิตพาริตี้

"001" กำหนดบิตพาริตี้คี่

"011" กำหนดพาริตี้คู่

"101" มาร์ก (Mark)

"111" ช่องว่าง(Space)

STOP ใช้กำหนดจำนวนบิตปิดท้าย

"1" มีบิตปิดท้าย 2 บิต

"0" มีบิตปิดท้าย 1 บิต

DAB1,DAB0 ใช้ร่วมกันในการกำหนดจำนวนบิตข้อมูลที่ต้องการถ่ายทอด

"00" จำนวนบิตข้อมูลเท่ากับ 5 บิต

"01" จำนวนบิตข้อมูลเท่ากับ 6 บิต

"10" จำนวนบิตข้อมูลเท่ากับ 7 บิต

"11" จำนวนบิตข้อมูลเท่ากับ 8 บิต

3.4.5 รีจิสเตอร์ตำแหน่ง 04H (รีจิสเตอร์ควบคุมโมเด็ม)

มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 1
0	0	0	LOOP	OUT2	OUT1	RTS	DTR

ตารางที่ 3.6 ฟังก์ชันการทำงานของรีจิสเตอร์ตำแหน่ง 04H

บิต 5-7 ไม่มีการใช้งาน อ่านค่าได้เท่ากับ 0

LOOP "1" เอ็นเอเบิลการส่งค่ากลับ

"0" ดิสเอเบิล

OUT1,

OUT2 "1" เอ็นเอเบิลการใช้งานภายใน

"0" ดิสเอเบิล

RTS ใช้ควบคุมการทำงานของขา RTS (Ready To Set)

"1" เอ็นเอเบิล

"0" ดิสเอเบิล

DTR ใช้ควบคุมการทำงานของขา DTR(Data Terminal Ready)

"1" เอ็นเอเบิล

"0" ดิสเอเบิล

3.4.6 รีจิสเตอร์ตำแหน่ง 05H(รีจิสเตอร์แสดงสถานการณ์รับและการส่งข้อมูลแบบอนุกรม)

ใช้งานร่วมกับรีจิสเตอร์แสดงโหมดและสถานะของการอินเตอร์รัป (รีจิสเตอร์ตำแหน่ง 02H) เพื่อแสดงสาเหตุของการเกิดอินเตอร์รัป มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 1
0	TXE	TBE	BREK	FRME	PARE	OVFE	RxRD

ตารางที่ 3.7 ฟังก์ชันการทำงานของรีจิสเตอร์ตำแหน่ง 05H

TXE (Transmitter Empty)

"1" รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

"0" ยังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

TBE (Transmitter Buffer Empty)

"1" รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

"0" ยังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

BREK(Break)

"1" UART ตรวจพบการเบรก

"0" ไม่มีการเบรก

FRME(Frame)

"1" UART ตรวจพบความผิดพลาดด้านเฟรมข้อมูล

"0" ไม่มีความผิดพลาดทางพาริตี้

OVRE(Overrun Error)

"1" UART ตรวจพบความผิดพลาดแบบโอเวอร์รัน

"0" ไม่มีความผิดพลาดแบบโอเวอร์รัน

RxRD(Received Data Ready)

"1" มีการรับข้อมูลมาเก็บในบัฟเฟอร์

"0" ไม่มีข้อมูล

3.4.7 รีจิสเตอร์ตำแหน่ง 06H (รีจิสเตอร์แสดงสถานะโมเด็ม)

ใช้เพื่อกำหนดสถานะสัญญาณอินพุต ของพอร์ตอนุกรม RS -232 ซึ่งได้แก่สัญญาณ DCD , DSR , CTS และ RI สำหรับการเชื่อมต่อใช้งานแบบเอนกประสงค์ดังมีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 1
DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

ตารางที่ 3.8 ฟังก์ชันการทำงานของรีจิสเตอร์ตำแหน่ง 06H

DCD ใช้แสดงสภาวะของขา DCD

"1" แสดงว่าที่ขา DCD เป็นลอจิก "1"

"0" แสดงว่าที่ขา DCD เป็นลอจิก "0"

RI ใช้แสดงสภาวะของขา RI

"1" แสดงว่าที่ขา RI เป็นลอจิก "1"

"0" แสดงว่าที่ขา RI เป็นลอจิก "0"

DSR ใช้แสดงสภาวะของขา DSR

"1" แสดงว่าที่ขา DSR เป็นลอจิก "1"

"0" แสดงว่าที่ขา DSR เป็นลอจิก "0"

DCTS (Delta Clear To Send) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นที่บิด CTS

"1" แสดงว่าบิตCTSเกิดการเปลี่ยนแปลงเมื่อเทียบกับจากการอ่านครั้งที่แล้ว

"0" แสดงว่าบิตCTSไม่มีเกิดการเปลี่ยนแปลงเมื่อเทียบกับจากการอ่านครั้งที่แล้ว

DRI (Delta Ring Indicator) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นที่บิด RI

"1" แสดงว่าบิตRIเกิดการเปลี่ยนแปลงเมื่อเทียบกับจากการอ่านครั้งที่แล้ว

"0" แสดงว่าบิตRIไม่มีเกิดการเปลี่ยนแปลงเมื่อเทียบกับจากการอ่านครั้งที่แล้ว

DDCD (Delta Data Carrier Detect) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นที่บิต DDCD

"1" แสดงว่าบิต เกิดการเปลี่ยนแปลงเมื่อเทียบกับจากการอ่านครั้งที่แล้ว

"0" แสดงว่าบิต ไม่มีเกิดการเปลี่ยนแปลงเมื่อเทียบกับจากการอ่านครั้งที่แล้ว

DCTS (Delta Clear To Send) ใช้แสดงสภาวะของ CTS

"1" แสดงว่าที่ขา CTS เป็นลอจิก "I"

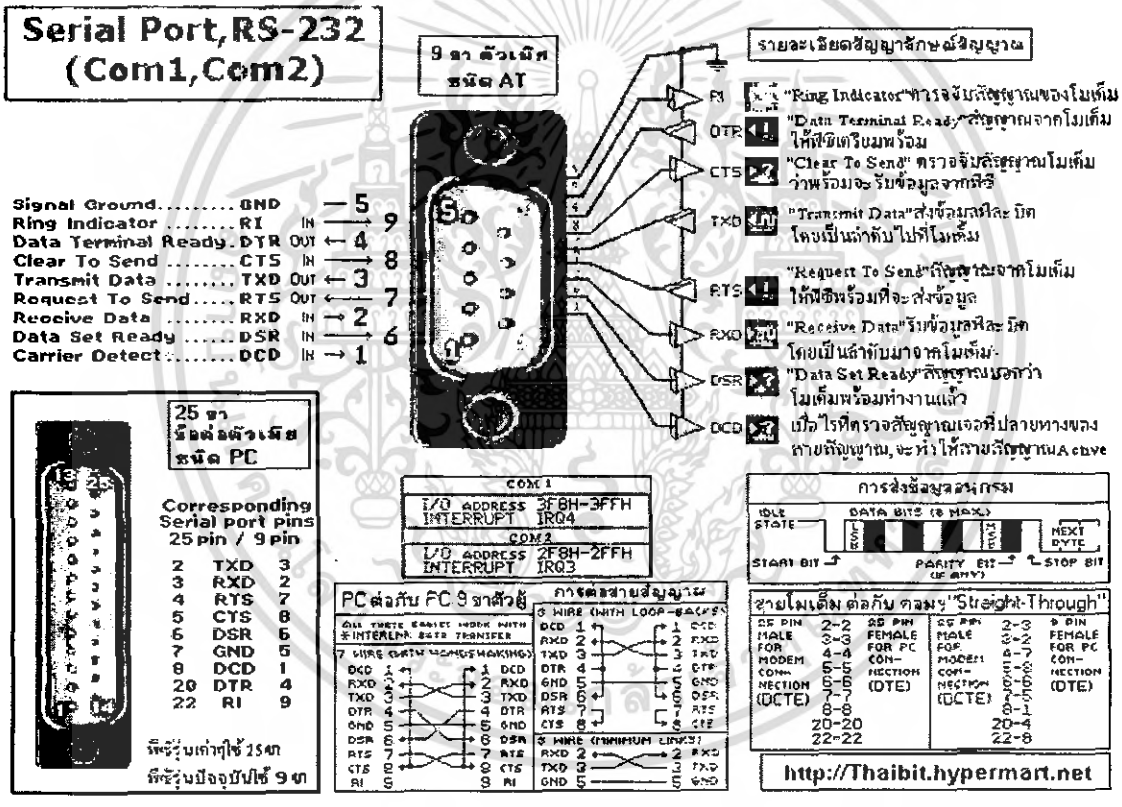
"0" แสดงว่าที่ขา CTS เป็นลอจิก "0"

3.4.8 รีจิสเตอร์ตำแหน่ง 07H(รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว)

ทำหน้าที่เป็นหน่วยความจำแรมขนาด 1 ไบต์ การอ่านและเขียนข้อมูลที่รีจิสเตอร์ตัวนี้ไม่ส่งผลใดๆต่อการใช้งาน UART

3.5 ลักษณะสัญญาณอินพุตและเอาท์พุตของพอร์ต RS-232

สัญญาณเอาท์พุทที่ใช้ควบคุม (RTS และ DCD) และสัญญาณแสดงสถานะอินพุต (CTS CSR และ DCD) ของพอร์ตอนุกรม RS-232 จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้งภาครับและภาคส่ง จะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาท์พุตออกมาเป็นทีทีแอลเท่านั้น ดังนั้นเมื่อสัญญาณถูกส่งออกมาจาก UART จะต้องส่งเข้าวงจรขับเพื่อปรับแรงดันเพื่อให้ระดับแรงดันเป็นไปตามมาตนฐาน RS-232 ก่อนส่งออกไปจากคอมพิวเตอร์สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็ต้องมีวงจรขับลักษณะนี้เช่นเดียวกัน เพื่อให้ได้สัญญาณในระดับเดียวกัน แต่วงจรขับที่ใช้ทั้งภายในคอมพิวเตอร์และอุปกรณ์เชื่อมต่อปลายทางนั้นจะถูกกลับสถานะ



รูปที่ 3.5 ภาพประกอบการต่อขาต่างๆของพอร์ตอนุกรม

3.6 แอดเดรสของพอร์ตอนุกรม

แอดเดรสพื้นฐานของพอร์ตอนุกรมมี 4 ตำแหน่งดังนี้คือ

COM1 : 3F8H

COM2 : 2F8H

COM3 : 3E8H

COM4 : 2E8H

บิต 3	บิต 2	บิต 1	จำนวนพอร์ต
0	0	0	ไม่มีพอร์ตอนุกรม
0	0	1	มีพอร์ตอนุกรม 1 พอร์ต
0	1	0	มีพอร์ตอนุกรม 1 พอร์ต
0	1	1	มีพอร์ตอนุกรม 2 พอร์ต
1	0	0	มีพอร์ตอนุกรม

ตารางที่ 3.9 ข้อมูลในแอดเดรส 0000 : 0411H ที่ใช้แจ้งจำนวนพอร์ตอนุกรม

เมื่อเริ่มต้นเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสภายในคอมพิวเตอร์จะทำการตรวจสอบแอดเดรสของพอร์ตอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ตอนุกรม ไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบต์ สำหรับพอร์ตอนุกรมต่างๆ จะเก็บแอดเดรสไว้ดังนี้

COM1 = 0000 : 0400H และ 0000 : 0401H

COM2 = 0000 : 0402H และ 0000 : 0403H

COM3 = 0000 : 0404H และ 0000 : 0405H

COM4 = 0000 : 0406H และ 0000 : 0407H

นอกจากนี้ที่หน่วยความจำแอดเดรส 0000 : 0411H ยังใช้สำหรับแสดงจำนวนของพอร์ดอนุกรมที่มีใช้อยู่ในเครื่องคอมพิวเตอร์อีกด้วย

บทที่ 4 ไมโครคอนโทรลเลอร์ MCS-51

4.1 คุณลักษณะพื้นฐานของ MCS-51

คุณสมบัติของไมโครคอนโทรลเลอร์ AT89CX051

-มีหน่วยความจำโปรแกรมชนิด Flash Memory ขนาด 2 Kbytes

-มีหน่วยความจำแรม 8 Bit ขนาด 128 Byte

-ทำงานที่ความเร็วนาฬิกาสูงสุด 24 MHz

-มีอินพุทเอาท์พุทพอร์ตขนาด 15 บิต

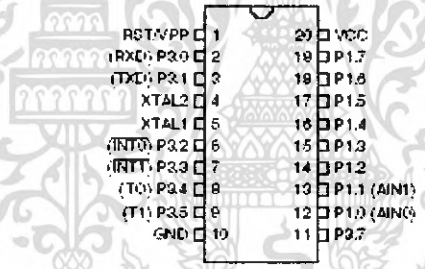
-มีสัญญาณอินเตอร์รัพท์ได้ 3 แหล่ง

-มีพอร์ตสื่อสารแบบอนุกรม 1 ช่อง(UART)

-มีวงจรเปรียบเทียบสัญญาณอนาลอก(Analog Comparator Input) 1 ช่อง

-มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป

4.2 ลักษณะการจัดขาของ MCS-51



รูปที่ 4.1 การจัดขามาตรฐานของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89C51

ขา Vcc ใช้สำหรับต่อไฟเลี้ยง +5V

ขา GND เป็นขากราวด์ สำหรับต่อกับกราวด์ของระบบ

ขาพอร์ต 1 (P1.0-P1.7) มีขา 8ขา แต่ละขาเรียกได้เป็น 1 บิตสามารถกำหนดให้เป็นทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุท สามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย

ขาพอร์ต 3(P3.0-P3.7) มีขา7 ขา แต่ละขาเรียกได้เป็น 1 บิต แต่ในส่วนของวงจรภายในไอซีจะมีขาของพอร์ต 3 อยู่ทั้งหมด 8 ขา เพียงแต่ขา P 3.6จะไม่ได้ต่อออกมาใช้งานภายนอก แต่ใช้เป็นขารับสถานะของผลการเปรียบเทียบสัญญาณ Analog Comparator Input ระหว่างพอร์ต P1.1 และ P1.1 จากภายนอกของไอซี ดังนั้นขาทั้ง 7ขาที่ต่อใช้งานสามารถกำหนดให้เป็นได้ทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุทสามารถทำได้โดยการ

เขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุทอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ตอินพุทได้ นอกจากนั้นขาพอร์ต 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษ ดังมีรายละเอียดขั้นตอนต่อไปนี้

P3.0 ใช้เป็นขาอินพุทสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

P3.1 ใช้เป็นขาอินพุทสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

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

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

P3.4 ใช้เป็นขาอินพุทสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0หรือขา T0

P3.5 ใช้เป็นขาอินพุทสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 1หรือขา T1

P3.7 ใช้เป็นขาอินพุทและเอาท์พุทสำหรับใช้งานทั่วไป

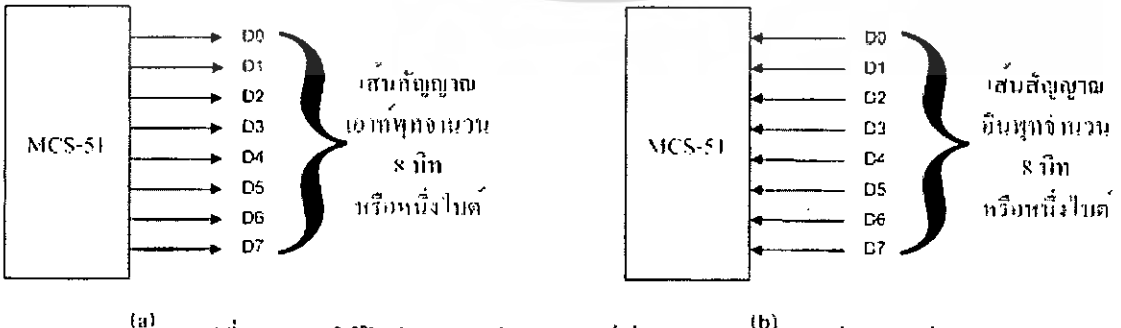
***P3.6** อยู่ภายในไอซีไม่ได้ต่อออกมาภายนอก แต่ใช้เป็นขารับสถานะของการเปรียบเทียบสัญญาณ Analog Comparator Input ระหว่างพอร์ต P1.0 และ P1.1 จากภายนอก

ขารีเซต (Reset) ใช้ในการรีเซตการทำงานของไมโครคอนโทรลเลอร์ โดยใช้การป้อนสัญญาณเพื่อรีเซตสถานะที่ขานี้ต้องอยู่ในระดับรีเซตอย่างน้อย 2แมชชีนไซเคิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างเป็นปกติ

ขา XTAL1 และ XTAL2 เป็นขาสำหรับต่อคริสตัลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์

4.3 พอร์ตอินพุทและพอร์ตเอาท์พุท

พอร์ต คือ แอดเดรสหนึ่งที่ได้รับการกำหนดไว้เพื่อการโอนย้ายข้อมูลระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ภายนอก การกำหนดประเภทของการติดต่อขึ้นอยู่กับทิศทางการไหลของข้อมูล เมื่อพิจารณาจากไมโครคอนโทรลเลอร์เป็นหลัก จากรูปที่ 2.3(a) เป็นการใช้ไมโครคอนโทรลเลอร์เป็นเอาท์พุทพอร์ต และจากรูป 2.3(b) เป็นการใช้ไมโครคอนโทรลเลอร์เป็นอินพุทพอร์ต



รูปที่ 4.2 การใช้ไมโครคอนโทรลเลอร์เป็นอินพุทและเอาท์พุทพอร์ต

4.3.1 การใช้งานพอร์ตเป็นอินพุท

การใช้งานพอร์ตเป็นการอินพุทข้อมูลจะต้องเริ่มต้นด้วยการส่งข้อมูลที่มีค่าเป็น 1ออกมาทางบิตของพอร์ตสั้นก่อนเป็นอันดับแรก เพื่อหยุดการทำงานของทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทของบิตนั้น ทำให้ขาสัญญาณของบิตถูกต่อเข้ากับตัวต้านทานซึ่งทำหน้าที่ Pull-up ภายในซึ่งมีผลทำให้บิตนั้นของพอร์ต 2 , 1และ 3เป็นสภาวะลอจิกสูง ตัวต้านทานนี้มีค่าประมาณ 50 kohm ซึ่งเป็นค่าที่สูงมาก และทำให้อุปกรณ์ภายนอกสามารถขับสัญญาณของพอร์ตเหล่านี้เป็นลอจิกต่ำได้ง่าย สำหรับบิตของพอร์ต 0นั้นแม้ว่าจะมีหลักการทำงานที่คล้ายคลึงกันกับบิตของพอร์ตอื่นๆ แต่เนื่องจากไม่มีตัวต้านทานซึ่งทำหน้าที่ Pull-up ภายในไว้ ทำให้เมื่อทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นหยุดการทำงาน ก็จะเป็นผลให้สัญญาณนี้อยู่ในสภาวะอิมพีแดนซ์สูงแทน

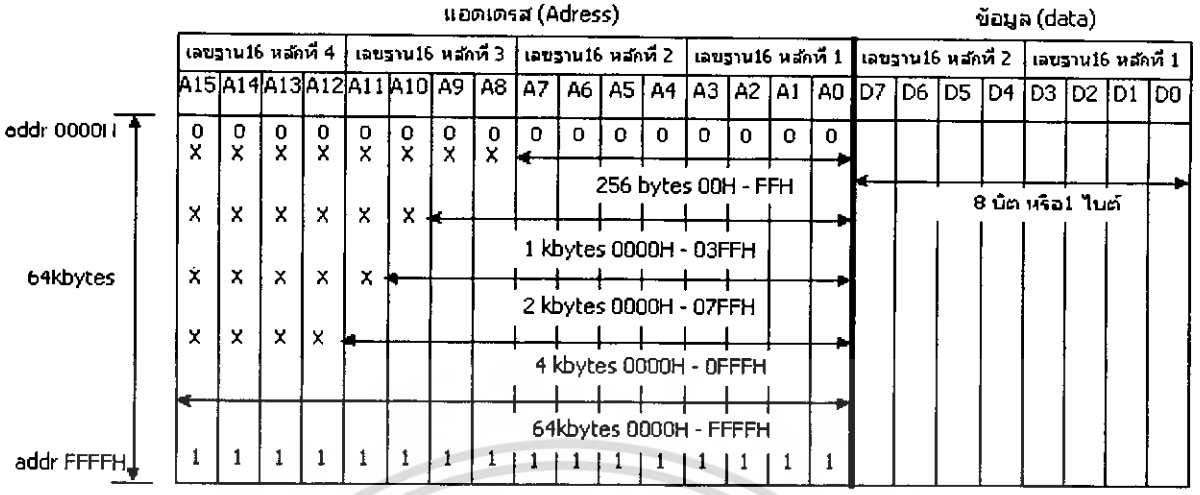
4.3.2 การใช้งานพอร์ตเป็นเอาท์พุท

เมื่อมีการส่งข้อมูลที่มีค่าเป็น 0ให้กับแต่ละบิตของพอร์ตทุกพอร์ต ข้อมูลนี้จะถูกส่งให้กับฟลิปฟลอป ซึ่งจะค้างค่านี้ไว้ และมีผลทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นทำงาน ดังนั้นขาสัญญาณก็จะมีสภาวะลอจิกเป็นลอจิกต่ำด้วย

ส่วนการส่งข้อมูลที่มีค่าเป็น 1ออกมานั้น ในกรณีที่เป็นการทำงานในแต่ละบิตของพอร์ต 2 , 1หรือ 3จะทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาท์พุทนั้นหยุดทำงาน มีผลทำให้ขาของสัญญาณเป็นลอจิกสูงด้วยตัวต้านทานที่ Pull-up อยู่ภายในนั้น แต่สำหรับการใช้งานในแต่ละบิตทางพอร์ต 0นั้นจะมีผลแตกต่างออกไป โดยขาสัญญาณจะมีสภาวะอิมพีแดนซ์สูงแทน เนื่องจากไม่มีตัวต้านทานภายในเชื่อมต่ออยู่นั้นเอง ดังนั้นการใช้งานพอร์ต 0เป็นการนำข้อมูลออกทางเอาท์พุท จึงจำเป็นต้องใช้ตัวต้านทานภายนอก Pull-up สัญญาณไว้กับลอจิกสูงแทน

4.4 หน่วยความจำโปรแกรมของ MCS-51

หน่วยความจำสำหรับเก็บโปรแกรม (Program Memory) หรือหน่วยความจำรหัสคำสั่ง (Code Memory) จะทำหน้าที่เก็บชุดคำสั่งเพื่อให้ไมโครคอนโทรลเลอร์ปฏิบัติตามคำสั่งนั้นๆ AT89C2051 จะมีหน่วยความจำที่เก็บโปรแกรมได้ 2Kbytes หน่วยความจำจะเป็นลักษณะแบบแฟลช ที่มีคุณสมบัติในการใช้งานโดยสามารถจะทำการลบข้อมูลด้วยไฟฟ้า และเก็บข้อมูลเข้าเก็บไว้ในตัวไมโครคอนโทรลเลอร์ได้กว่า 1000ครั้ง โดยใช้เครื่อง

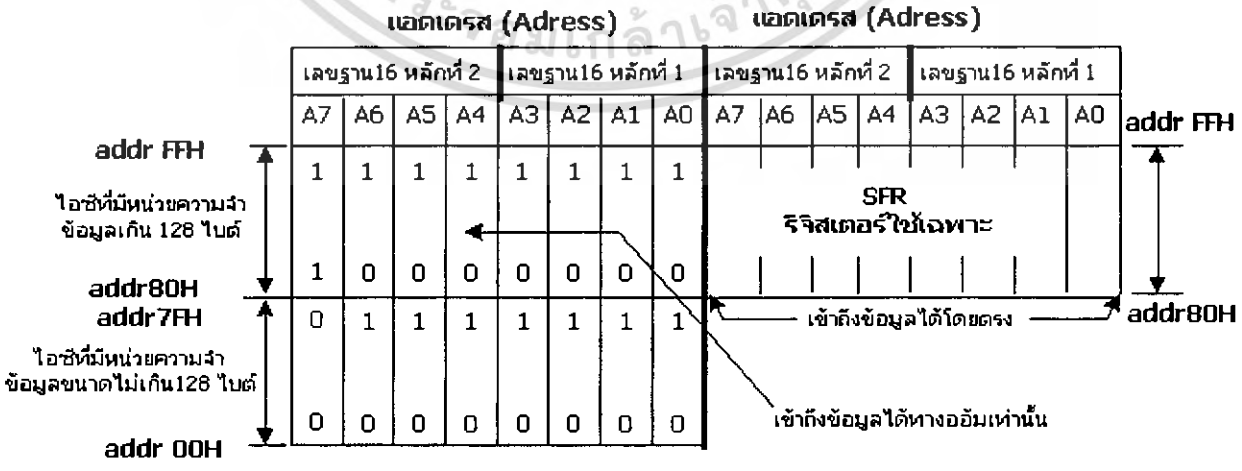


รูปที่ 4.3 แสดงการจัดพื้นที่ของหน่วยความจำโปรแกรมของไมโครคอนโทรลเลอร์ MCS-51

ส่วนของแอดเดรส(ADDRESS) ไม่สามารถที่จะใช้ตำแหน่งเดียวกันได้ แต่ข้อมูล (DATA) สามารถที่จะมีข้อมูลเหมือนกันได้

4.5 หน่วยความจำข้อมูลของ MCS-51

หน่วยความจำข้อมูล(RAM) จะทำหน้าที่เก็บรักษาข้อมูล โดยข้อมูลอาจจะเก็บค่าหลังจากไมโครคอนโทรลเลอร์ทำการประมวลผล หรือเก็บค่าข้อมูลที่จะให้กับไมโครคอนโทรลเลอร์ประมวลผลในขณะนั้นและจะทำหน้าที่เป็น สแตก(Stack) บางส่วน ไมโครคอนโทรลเลอร์ AT89C2051 จะมีหน่วยความจำที่เก็บข้อมูลได้ 128 byte



รูป 4.4 ตารางตำแหน่งแอดเดรสของหน่วยความจำข้อมูลภายใน

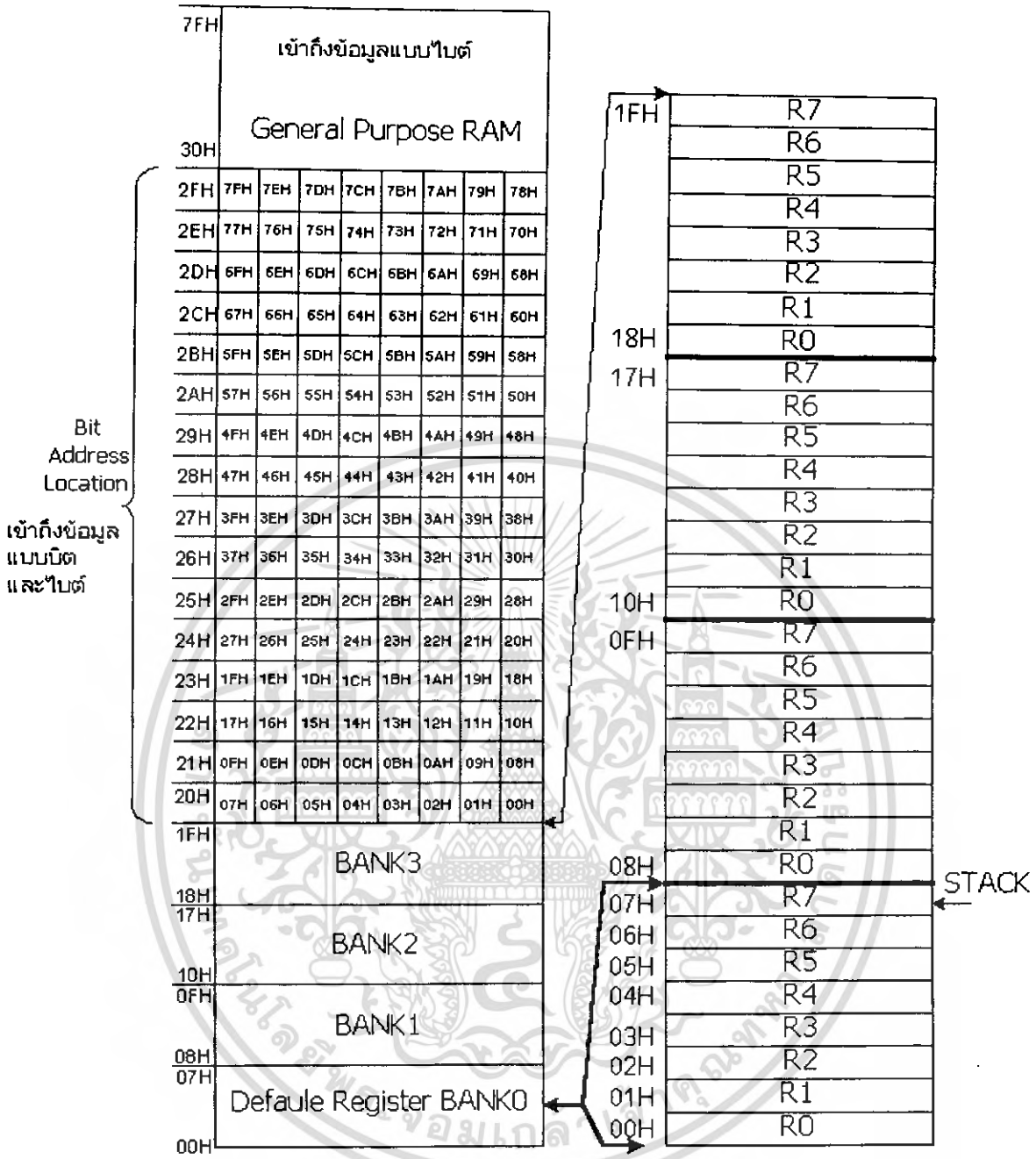
หน่วยความจำข้อมูลภายในยังแบ่งส่วนของการใช้งานได้อีกเป็นสองส่วนคือ หน่วยความจำข้อมูลภายใน 128ไบต์จะเป็นหน่วยความจำที่ใช้งานทั่วไปอยู่ที่ตำแหน่งแอดเดรส 00H-7FH และหน่วยความจำในตำแหน่งแอดเดรสที่ 80H-FFH ซึ่งจะเป็นส่วนของรีจิสเตอร์เฉพาะ (Special Function Registor)

โดยหน่วยความจำข้อมูลภายในที่ตำแหน่งแอดเดรส 00H-7FH ก็ยังสามารถแบ่งออกเป็นส่วนย่อยได้ดังนี้

-พื้นที่ในหน่วยความจำข้อมูลดำแหน่งที่ 00H-1FH จำนวน 32ไบต์ จะถูกแบ่งออกเป็น 4 กลุ่ม เรียกว่า แบงค์(Bank) และในแต่ละแบงค์จะมี 8ไบต์ พื้นที่ในแต่ละแบงค์จะถูกใช้งานเป็นรีจิสเตอร์ที่ใช้งานทั่วไป(R0-R7) โดยที่รีจิสเตอร์ R0 จะอยู่ในตำแหน่งแรกของแต่ละแบงค์ และรีจิสเตอร์ R7 จะอยู่ในดำแหน่งสุดท้ายของแต่ละแบงค์ ในการนำไปใช้งานจะเลือกใช้รีจิสเตอร์ได้เพียงแบงค์เดียว และเลือกใช้พื้นที่ของรีจิสเตอร์ R0-R7 ในแบงค์ใดๆก็ได้ โดยการกำหนดค่าของข้อมูลที่รีจิสเตอร์ PSW ในส่วนของรีจิสเตอร์เฉพาะ (Special Function Register) หากไม่มีการกำหนดใดๆเลย เมื่อทำการรีเซ็ตให้กับไมโครคอนโทรลเลอร์จะถูกกำหนด ให้เริ่มต้นใช้งานที่รีจิสเตอร์ R0-R7 ในหน่วยความจำตำแหน่งแบงค์ 0ให้เอง

-พื้นที่ในหน่วยความจำข้อมูลภายในตำแหน่งที่แอดเดรสที่ 20H-2FH จำนวน 16ไบค์ เป็นส่วนที่ใช้งานในลักษณะการเข้าข้อมูลแบบไบต์หรือแบบบิตได้ และสามารถอ้างตำแหน่งแบบบิตได้โดยตรง เพียงแต่ระบุตำแหน่งหรือชื่อของบิตนั้นๆได้ ซึ่งจะมีอยู่จำนวน 128บิต แต่ละบิตจะมีหมายเลขตำแหน่งของบิดคือ 00H-7FH โดยตำแหน่งบิตที่ 00H ก็คือข้อมูลบิดที่ต่ำสุดในตำแหน่งที่แอดเดรสที่ 20H หรืออาจเรียกว่า (20H.1) และตำแหน่งของบิตที่ 7FH คือข้อมูลบิตที่สูงสุดในตำแหน่งแอดเดรสที่ 2FH หรืออาจเรียกว่า (20H.7) การอ้างตำแหน่งบิดจะทำให้โปรแกรมทำงานได้เร็วขึ้น

-พื้นที่ในหน่วยความจำข้อมูลภาในตำแหน่งที่แอดเดรสที่ จะเป็นพื้นที่ของหน่วยความจำใช้งานทั่วไป และการติดต่อกับข้อมูลในตำแหน่งด่างๆของหน่วยความจำส่วนนี้จะอ้างตำแหน่งข้อมูลในลักษณะแบบไบต์เท่านั้นและพื้นที่ส่วนนี้อาจใช้เป็นสแตกได้



รูปที่ 4.5 แสดงตำแหน่งของหน่วยความจำแบบไบต์และแบบบิต

4.6 รีจิสเตอร์หน้าที่พิเศษ

เป็นรีจิสเตอร์ที่ทำหน้าที่ควบคุมการทำงานของอุปกรณ์หรือพอร์ตของ 89C 2051ทั้งหมด โดยมีตำแหน่งอยู่ในบริเวณแอดเดรส 80H-FFH การใช้งานรีจิสเตอร์หน้าที่พิเศษเหล่านี้สามารถทำได้ทั้งการระบุถึงชื่อรีจิสเตอร์หรือตำแหน่งแอดเดรสที่เป็นของรีจิสเตอร์นั้นก็ได้

4.6.1 แอกคิวมูเลเตอร์ (Accumulator)

เป็นรีจิสเตอร์ขนาด 8บิต ทำหน้าที่เก็บข้อมูลที่ส่งให้หน่วยทำงานในซีพียูและเก็บผลลัพธ์ที่ได้จากการทำงานนั้น การใช้งานในโปรแกรมจะเรียกว่า รีจิสเตอร์ A

4.6.2 รีจิสเตอร์ B

เป็นรีจิสเตอร์ที่ใช้สำหรับการทำคำสั่งการคูณหารตัวเลข ในกรณีที่ไม่ใช้การคำนวณทางด้านคณิตศาสตร์ ก็สามารถนำไปใช้งานเช่นเดียวกับรีจิสเตอร์ทั่วไปได้

4.6.3 โปรแกรมเคาน์เตอร์(Program Counter)

เป็นรีจิสเตอร์ที่ใช้สำหรับการชี้ตำแหน่งแอดเดรสของหน่วยความจำโปรแกรม ซึ่งจะต้องไปทำงานในลำดับถัดไป การใช้งานในโปรแกรมจะเรียกว่า รีจิสเตอร์ PC

4.6.4 สแต็กพอยน์เตอร์(Stack Pointer)

เป็นรีจิสเตอร์ขนาด 8บิต ทำหน้าที่เก็บตำแหน่งของตัวชี้หรือพอยน์เตอร์ของบริเวณสแต็กสำหรับเก็บข้อมูลแอกคิวมูเลเตอร์ รีจิสเตอร์ต่างๆ รวมทั้งข้อมูลจากโปรแกรม ค่าเริ่มต้นของสแต็กจะอยู่ที่ตำแหน่ง 07H การใช้งานในโปรแกรมจะเรียกว่ารีจิสเตอร์ SP

4.6.5 ตัวชี้ข้อมูลหรือดาต้าพอยน์เตอร์(Data Pointer)

เป็นรีจิสเตอร์ขนาด 16บิต ซึ่งเรียกว่า รีจิสเตอร์ DPTR และสามารถใช้งานแยกออกเป็นรีจิสเตอร์ขนาด 8บิต สองตัว คือ รีจิสเตอร์ DPH และ DPL เพื่อเก็บค่าแอดเดรสของหน่วยความจำที่จะต้องใช้งานภายในโปรแกรม หรืออาจเป็นแอดเดรสของอุปกรณ์ภายนอก

4.6.6 โปรแกรมสเตตัสเวิร์ด(PSW)

รีจิสเตอร์นี้ทำหน้าที่บอกถึงแฟลกสภาวการณ์ทำงานต่างๆ รวมทั้งบิตสำหรับการกำหนดเลือกแบงก์(Bank) ของรีจิสเตอร์ที่ใช้งานด้วย

4.6.7 รีจิสเตอร์ที่เกี่ยวข้องกับพอร์ต(Port Register)

รีจิสเตอร์เหล่านี้จะมีความเกี่ยวข้องกับการทำงานของพอร์ตอินพุทเอาท์พุทโดยตรงซึ่งจะเป็นรีจิสเตอร์ขนาด 8บิต สามารถใช้งานได้ทั้งในลักษณะการอินพุทหรือการเอาท์พุทข้อมูลได้

4.6.8 รีจิสเตอร์ SBUF

เป็นบัฟเฟอร์ขนาด 8บิต สำหรับการสื่อสารข้อมูลแบบอนุกรมทั้งการรับและการส่งข้อมูล

4.6.9 รีจิสเตอร์ PCON

เป็นรีจิสเตอร์ควบคุมการทำงานในสามลักษณะ ซึ่งได้แก่ การควบคุมการทำงานของโปรแกรมเซ็นเซอร์ การกำหนดอัตราทวีคูณของอัตราเร็วในการสื่อสารข้อมูลอนุกรมและแฟลกสภาวะการทำงานทั่วไป

4.6.10 รีจิสเตอร์ IP,IE,TMOD,SCON

เป็นกลุ่มรีจิสเตอร์ที่ทำหน้าที่ควบคุมการทำงานของอินเตอร์รัปต์ต่างๆ

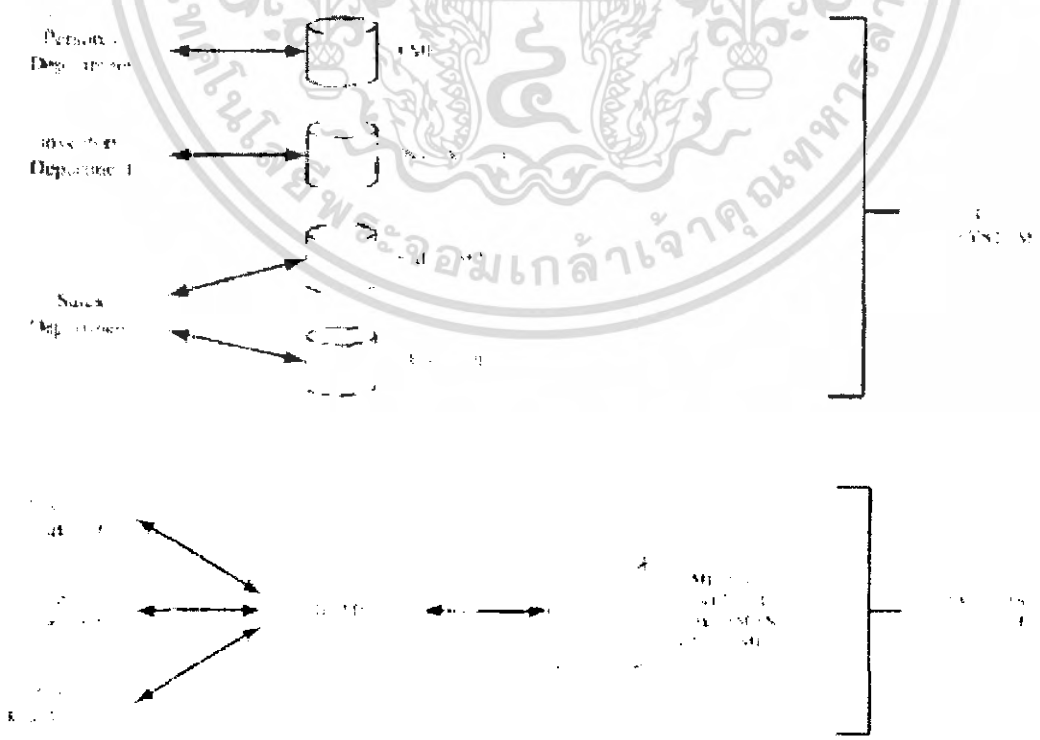
บทที่ 5 Data Base & Visual Basic

5.1 Data Base

5.1.1 ระบบฐานข้อมูล (Data Base System)

ในชีวิตประจำวันปัจจุบัน ไม่ว่าจะดำเนินงานใดๆมนุษย์จะต้องเกี่ยวข้องกับข้อมูลอย่างใดอย่างหนึ่งเสมอ เช่น การติดต่อราชการที่ต้องใช้ข้อมูลจากบัตรประชาชน การติดต่อกับธนาคารที่ต้องใช้ข้อมูลจากสมุดเงินฝาก เป็นต้น โดยเฉพาะอย่างยิ่ง เมื่อเทคโนโลยีของโลกพัฒนาขึ้น จนกระทั่งปัจจุบันที่มีคอมพิวเตอร์แทนที่การจัดเก็บข้อมูลในกระดาษ เนื่องจากปริมาณข้อมูลมีมากขึ้น ประกอบกับความต้องการใช้ข้อมูลมีมากขึ้น รวมทั้งข้อมูลได้เปลี่ยนไปเป็นปัจจัยที่มีผลต่อการแข่งขันทางธุรกิจ การจัดเก็บข้อมูลจึงได้เปลี่ยนไป และเกิดคำว่า "ฐานข้อมูล"

Data Base System คืออะไร จากปัญหาต่างๆที่เกิดขึ้นใน ได้ก่อให้เกิดการจัดเก็บข้อมูลรูปแบบใหม่ขึ้น ที่เรียกว่า "ฐานข้อมูล" การจัดเก็บข้อมูลในฐานข้อมูลนี้จะแตกต่างจากการเก็บข้อมูลในระบบแฟ้มข้อมูล เนื่องจากฐานข้อมูลเป็นการเอาข้อมูลต่างๆที่มีความสัมพันธ์กัน ซึ่งแต่เดิมจัดเก็บอยู่ในแต่ละแฟ้มข้อมูลมาจัดเก็บไว้ในที่เดียวกัน เช่น ข้อมูลพนักงาน สินค้าคงคลัง พนักงานขาย และลูกค้า ซึ่งแต่เดิมจัดเก็บอยู่ในรูปของแฟ้มของฝ่ายต่างๆ ได้ถูกนำมาจัดเก็บรวมกันไว้ในฐานข้อมูลเดียวกัน ซึ่งเป็นฐานข้อมูลรวมของบริษัท ส่งผลให้แต่ละฝ่ายใช้ข้อมูลร่วมกันและสามารถแก้ไขปัญหาต่างๆที่เกิดขึ้นในระบบแฟ้มข้อมูลได้ดังรูป



รูปที่ 5.1 ระบบแฟ้มข้อมูล

ข้อมูลต่างๆที่ถูกจัดเก็บเป็นบานข้อมูล นอกจากจะเป็นข้อมูลที่สัมพันธ์กันแล้ว ยังต้องเป็นข้อมูลที่ได้รับการสนับสนุนการดำเนินงานอย่างน้อยอย่างใดอย่างหนึ่งขององค์กร ดังนั้นอาจกล่าวได้ว่าแต่ละฐานข้อมูลเทียบได้กับระบบแฟ้มข้อมูล 1 ระบบ และจะเรียกฐานข้อมูลที่จัดทำขึ้นเพื่อสนับสนุนการดำเนินงานอย่างใดอย่างหนึ่งนั้นว่า "ระบบฐานข้อมูล" (Data Base System) เช่นระบบฐานข้อมูลเงินเดือน ซึ่งเป็นฐานข้อมูลที่จัดเก็บข้อมูลต่างๆที่สนับสนุนการจัดทำสำมะโนประชากร เป็นต้น

5.1.2 ระบบแฟ้มข้อมูล (File System)

ในอดีตองค์กรต่างๆ มักจัดเก็บเอกสารต่างๆไว้ในแฟ้มเอกสารต่างๆ ซึ่งมีความเกี่ยวข้องทางข้อมูลน้อยหรืออาจไม่มีเลย ขึ้นอยู่กับความต้องการในการใช้ข้อมูลนั้นๆ เช่น ประวัติการรักษาพยาบาล ที่โดยทั่วไปมักจะแยกเก็บเอกสารเฉพาะคนไข้แต่ละคน หรือประวัติพนักงาน ที่อาจแยกเก็บเอกสารตามฝ่ายที่สังกัด ซึ่งข้อมูลพนักงานของแต่ละคนในแต่ละแฟ้มเอกสารจะมีความเกี่ยวข้องกันตามฝ่ายที่สังกัด เช่นแฟ้มเอกสารประวัติพนักงานฝ่ายธุรการ แฟ้มเอกสารพนักงานฝ่ายการเงิน เป็นต้น แต่ต่อมาเมื่อองค์กรมีขนาดใหญ่ขึ้น จากเดิมที่สามารถค้นหาเอกสารจากแฟ้มข้อมูลเพียงแฟ้มเดียว ก็เริ่มต้องค้นหาเอกสารจากแฟ้มหลายแฟ้มจำนวนมากขึ้น ส่งผลให้การค้นเอกสารเป็นงานที่ต้องใช้เวลามาก และมีความยากลำบากมากขึ้น การจัดเก็บเอกสารในคอมพิวเตอร์ จึงถูกริเริ่มมาใช้ในองค์กรแทนการจัดเก็บในรูปแบบเดิม แต่การจัดเก็บเอกสารในคอมพิวเตอร์ยุคแรกๆ ยังคงไม่ค่อยมีประสิทธิภาพมากนัก เนื่องจากยังมีรูปแบบการจัดเก็บเอกสารคล้ายกับการจัดเก็บแบบเดิมอยู่ โดยเป็นเพียงการนำเอกสารต่างๆในแต่ละแฟ้มเอกสารมาจัดเก็บในรูปของแฟ้มข้อมูลแทน และด้วยเหตุนี่จึงต้องอาศัยผู้ชำนาญ เช่น โปรแกรมเมอร์ หรือนักวิเคราะห์เข้ามาช่วยกำหนดโครงสร้างของแฟ้มข้อมูล เพื่อที่จะสารถนำแฟ้มข้อมูลนั้นไปจัดเก็บข้อมูลและสามารถนำไปประมวลผลได้ตามความต้องการ

จากบทบาทของคอมพิวเตอร์ที่เข้ามามีอิทธิพลในการดำเนินการขององค์กร ส่งผลให้การจัดเก็บข้อมูลในแฟ้มข้อมูล มีการใช้แพร่หลายมากยิ่งขึ้น จากเดิมที่เพียง 2 หรือ 3 แฟ้มข้อมูล ได้เพิ่มขึ้นเป็นจำนวน 10 หรือ 20 แฟ้มข้อมูล ดังนั้นจึงต้องมีการเข้ามาควบคุมด้านโครงสร้าง และการใช้งานแฟ้มข้อมูลต่างๆ ให้มีความเหมาะสมต่อการใช้งานมากยิ่งขึ้น และรวบแฟ้มข้อมูลเหล่านี้เข้าเป็นระบบที่เรียกว่า "ระบบแฟ้มข้อมูล" (File System)

การใช้งานระบบแฟ้มข้อมูล จะต้องอาศัยโปรแกรมเมอร์พัฒนาโปรแกรม เพื่ออ่านข้อมูลต่างๆขึ้นมาประมวลผล เพื่อให้ได้ผลลัพธ์ตามที่ผู้ใช้ต้องการ ภาษาคอมพิวเตอร์ที่ใช้พัฒนาโปรแกรมโดยทั่วไปจะได้แก่ ภาษาคอมพิวเตอร์ในยุคที่ 3 (Third Generation Language : 3GL) เช่น ภาษา COBAL, FORTRAN, BASIC เป็นต้น แต่ภาษาคอมพิวเตอร์เหล่านี้มีข้อจำกัดในการเรียกใช้ข้อมูลจากแฟ้มข้อมูล เนื่องจากภาษาคอมพิวเตอร์เหล่านี้ จะต้องอ้างถึงแฟ้มข้อมูลตามโครงสร้างทาง

กายภาพของข้อมูลภายในแฟ้มข้อมูลนั้น เช่น เมื่อต้องการ อ้างถึงแฟ้มข้อมูลลูกค้า ซึ่งมีโครงสร้างดังนี้

CUSTOMER

CUST_NO	CUST_NAME	CUST_ADDRESS	CUST_PHONE
C0001	ABC Industry	111 Moo 4 Samutprakarn	552-000
C0002	Thai Steel CO., Ltd	100 Petchkasem Ratchaburi	321-180
C0003	ITT Computer CO., Ltd	90 III Bldg.Bangkok	433-0015
C0004	KKK (Thailand) CO., Ltd	18 Sathorn Bldg. Bangkok	236-8897

ตาราง 5.1 ตัวอย่างโครงสร้างข้อมูลในแฟ้มลูกค้า

ในภาษา COBAL จะต้องมีการนิยาม (Declare) โครงสร้างของ Record เช่น ประเภทของข้อมูล ขนาดของแต่ละ Field ไว้ในส่วนของ Data Division ตามโครงสร้างทางกายภาพของ Record ในแฟ้มลูกค้าก่อน จึงจะสมารถอ้างถึง Field ต่างๆ ในโปรแกรมได้ดังตัวอย่าง

```
01 CUSTOMER.
    02 CUST_NO        PIC X(5).
    02 CUST _NAME     PIC X(50).
    02 CUST_ADDRESS   PIC X(50).
    02 CUST_PHONE     PIC X(8).
```

ด้วยเหตุนี้จึงส่งผลให้การพัฒนาในแต่ละโปรแกรมขึ้นใช้งานกับแฟ้มข้อมูลต่างๆมีความซับซ้อน และต้องใช้เวลาค่อนข้างมาก รวมทั้งการทำให้แต่ละโปรแกรมถูกผูกติดอยู่กับแฟ้มข้อมูลต่างๆ ดังนั้นเมื่อต้องมีการเปลี่ยนแปลงโครงสร้างของแฟ้มข้อมูลใดข้อมูลหนึ่ง จึงต้องแก้ไขโปรแกรมต่างๆ ที่มีการเรียกใช้แฟ้มข้อมูลนั้นตามไปด้วย ส่งผลให้ค่าใช้จ่ายในการบำรุงรักษาค่อนข้างสูง โดยเฉพาะอย่างยิ่งในกรณีที่ต้องมีการแก้ไขโปรแกรมจำนวนมาก

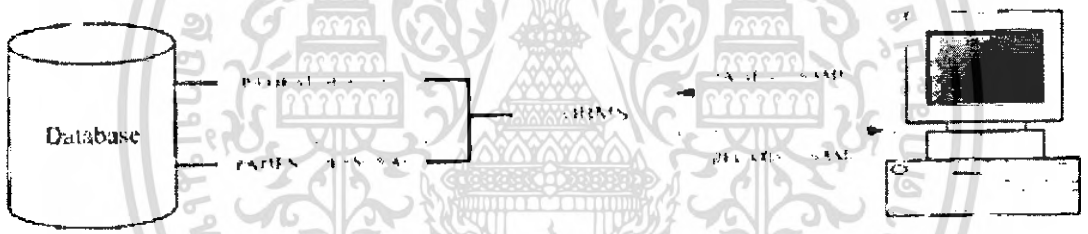
ในยุคเริ่มต้นของการใช้ระบบแฟ้มข้อมูล แต่ละหน่วยงานในองค์กร จะมีการสร้างระบบแฟ้มข้อมูลขึ้นใช้งานภายในหน่วยงานของตนเอง เช่น ระบบแฟ้มข้อมูลการขายของฝ่ายการตลาด ระบบแฟ้มข้อมูลพนักงานของฝ่ายการพนักงาน ระบบแฟ้มข้อมูลสินค้าคงคลังของฝ่ายคลังสินค้า เป็นต้น ซึ่งแต่ละหน่วยงานนั้น จะมีการพัฒนาโปรแกรมที่ใช้ข้อมูลจากระบบแฟ้มข้อมูลการขายในการออกใบสั่งซื้อ ใบกำกับสินค้า รายงานแสดงยอดการขายในแต่ละเดือน หรือฝ่ายคลังสินค้าที่มีการ

พัฒนาโปรแกรมที่ใช้ข้อมูลจากระบบแฟ้มข้อมูลสินค้าคงคลังในการพิมพ์รายงานแสดงยอดคงคลังของสินค้าต่างๆ

5.1.3 องค์ประกอบของระบบฐานข้อมูล

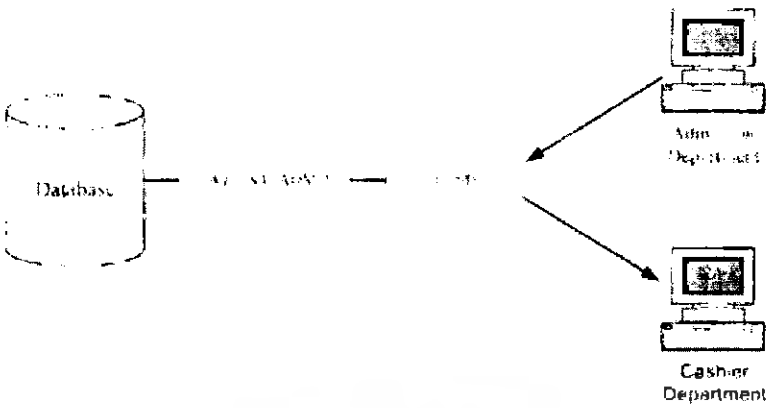
1.ข้อมูล (Data)

ข้อมูลที่จัดเก็บอยู่ในฐานข้อมูล ไม่ว่าจะเป็นเครื่องคอมพิวเตอร์ส่วนบุคคล ไปจนถึงเครื่องคอมพิวเตอร์ขนาดใหญ่อย่างเช่นเครื่อง Mainframe ข้อมูลในแต่ละส่วนจะต้องสามารถนำมาใช้ร่วมกันได้ เช่นเมื่อแพทย์รักษาผู้ป่วย แพทย์จะอาศัยประวัติจากการรักษาพยาบาลของผู้ป่วย (PATIENT_HISTORY) มาประกอบการรักษา แต่กรณีที่ต้องการติดต่อญาติผู้ป่วย ซึ่งข้อมูลในส่วนนี้ไม่ปรากฏอยู่ในประวัติการรักษาพยาบาล ทางพยาบาลสามารถนำชื่อผู้ป่วย (Field "PATIENT_NAME") ไปค้นหาชื่อญาติ (Field "RELATIVE_NAME") ในทะเบียนผู้ป่วย (PATIENT_PERSONAL) ได้โดยไม่ต้องเก็บชื่อญาติของผู้ป่วยในประวัติการรักษาพยาบาลแต่อย่างใดดังรูป



รูปที่ 5.2 ตัวอย่างการจัดเก็บข้อมูลของผู้ป่วย

นอกจากคุณลักษณะนี้แล้ว ในเครื่องคอมพิวเตอร์ขนาดใหญ่ที่มีผู้ใช้จำนวนมาก ข้อมูลในฐานข้อมูลจะต้องสามารถถูกใช้ร่วมกัน จากผู้ใช้หลายๆคนได้ เช่น ข้อมูลในการจองห้องพักของผู้ป่วย(PATIENT_ADMIN) จะต้องสามารถนำไปใช้ในการออกใบเสร็จรับเงินเพื่อเก็บค่ารักษาพยาบาลโดยฝ่ายการเงินได้ในขณะเดียวกัน ดังรูป



รูปที่ 5.3 ตัวอย่างการใช้ข้อมูลร่วมกันของแต่ละฝ่าย

2.ฮาร์ดแวร์ (Hard Ware)

อุปกรณ์ทางคอมพิวเตอร์ที่มีความเกี่ยวข้องกับระบบฐานข้อมูลจะประกอบด้วย 2 ส่วน หลักๆ ดังนี้

* หน่วยความจำสำรอง (Secondary Storage) เนื่องจากเป็นอุปกรณ์ทางคอมพิวเตอร์ ที่ใช้จัดเก็บข้อมูลของฐานข้อมูล ดังนั้นสิ่งที่ต้องคำนึงถึงสำหรับอุปกรณ์ในส่วนนี้คือ ความจุของหน่วยความจำสำรองที่นำมาใช้ จัดเก็บฐานข้อมูลนั้น

* หน่วยประมวลผลและหน่วยความจำหลัก เนื่องจากเป็นอุปกรณ์ที่จะต้องทำงานร่วมกัน เพื่อนำข้อมูลจากฐานข้อมูลขึ้นมาประมวลผลตามคำสั่งที่กำหนด ดังนั้นสิ่งที่ต้องคำนึงถึงอุปกรณ์ส่วนนี้คือ ความเร็วของหน่วยประมวลผล และขนาดของหน่วยความจำหลักของเครื่องคอมพิวเตอร์ที่นำมาใช้ประมวลผลร่วมกับฐานข้อมูลนั้น

3.ซอฟท์แวร์ (Software)

ในการติดต่อข้อมูลกับฐานข้อมูลของผู้ใช้ จะต้องกระทำผ่านทางโปรแกรมที่มีชื่อว่า Data Base Management System DBMS

หน้าที่หลักของโปรแกรม DBMS คือทำให้การเรียกใช้ข้อมูลจากฐานข้อมูล เป็นอิสระจากส่วนของ Hardware หรือกล่าวอีกนัยหนึ่ง โปรแกรม DBMS จะมีหน้าที่จัดการและควบคุมความถูกต้อง ความซ้ำซ้อนและความสัมพันธ์ต่างๆภายในฐานข้อมูลแทนโปรแกรมเมอร์ ส่งผลให้ผู้ใช้สามารถเรียกใช้ข้อมูลจากฐานข้อมูลได้โดยที่ไม่จำเป็นต้องทราบโครงสร้างทางกายภาพของข้อมูลในระดับที่ลึกเช่นเดียวกับโปรแกรมเมอร์ เนื่องจากโปรแกรม DBMS นี้ จะมีส่วนของ Query Language ซึ่งเป็นภาษาที่ประกอบด้วยคำสั่งต่างๆที่ใช้ในการจัดการและเรียกใช้ข้อมูลจากฐานข้อมูล ซึ่งสามารถ

นำไปใช้ร่วมกับภาษาคอมพิวเตอร์อื่นๆได้เพื่อพัฒนาเป็นโปรแกรมที่สามารถเรียกใช้ข้อมูลมาเพื่อประมวลผล สำหรับรายละเอียดของโปรแกรม DBMS จะกล่าวถึงต่อไป

4.ผู้ใช้ระบบฐานข้อมูล (User)

ผู้ที่เรียกใช้ข้อมูลจากฐานข้อมูลมาใช้งานสามารถแบ่งออกเป็น 3 กลุ่มได้ดังนี้

*Application Programmer ได้แก่ผู้ที่ทำหน้าที่พัฒนาโปรแกรม เพื่อเรียกใช้ข้อมูลจากระบบฐานข้อมูลมาประมวลผล โดยที่โปรแกรมที่พัฒนาขึ้นส่วนใหญ่ มักจะใช้ร่วมกับคำสั่งในกลุ่ม Data Manipulation Language : DML ของ Query Language เพื่อเรียกใช้ข้อมูลจากฐานข้อมูล สำหรับรายละเอียดของคำสั่งจะกล่าวต่อไป

*End User ได้แก่ ผู้ที่นำข้อมูลจากฐานข้อมูลไปใช้งาน ซึ่งแบ่งออกเป็น 2 กลุ่มดังนี้

Naive User ได้แก่ ผู้ใช้ที่เรียกใช้ข้อมูลจากฐานข้อมูลโดยอาศัยโปรแกรมที่พัฒนาขึ้น

Sophisticated User ได้แก่ ผู้ที่เรียกใช้ข้อมูลจากฐานข้อมูลโดยคำสั่งของ Query Language ซึ่ง โดยทั่วไปผลิตภัณฑ์ทางด้านฐานข้อมูลที่จำหน่ายอยู่ในตลาดจะมีส่วนที่ยอมให้ผู้ใช้ ได้ใช้ประโยคคำสั่งของ Query Language เพื่อเรียกใช้ข้อมูลจากฐานข้อมูลได้โดยตรง สำหรับประโยคคำสั่งเหล่านี้จะถูก Query Processor ของโปรแกรม DBMS แปลงให้อยู่ในรูปของคำสั่งในกลุ่ม Data Manipulation Language

* Data Base Administrator :DBA ได้แก่ผู้บริหารที่ควบคุมและตัดสินใจในการกำหนดโครงสร้างฐานข้อมูล ชนิดของข้อมูล วิธีจัดเก็บข้อมูล รูปแบบในการเรียกใช้ข้อมูล ความปลอดภัยของข้อมูล และกฎระเบียบที่ใช้กำหนดความถูกต้องของข้อมูลในฐานข้อมูล โดยอาศัยคำสั่งในกลุ่ม Data Definition Language : DDL ซึ่งเป็นอีกส่วนหนึ่งของ Query Language เป็นตัวกำหนด

หน้าที่ของ DBMS

1. ทำหน้าที่แปลงคำสั่งที่ใช้จัดการกับข้อมูลภายในฐานข้อมูล ให้อยู่ในรูปแบบที่ฐานข้อมูลเข้าใจ
2. ทำหน้าที่ในการนำคำสั่งต่างๆ ซึ่งได้รับการแปลแล้ว มาสั่งให้ฐานข้อมูลทำงาน เช่น การเรียกใช้ข้อมูล (Retrive) การจัดเก็บข้อมูล (Update) การลบข้อมูล (Delete) การเพิ่มข้อมูล (Add) เป็นต้น
3. ทำหน้าที่ป้องกันความเสียหายที่จะเกิดขึ้นกับข้อมูลภายในฐานข้อมูล โดยจะคอยตรวจสอบว่าคำสั่งใดสามารถทำงานได้ และคำสั่งใดที่ไม่สามารถทำงานได้
4. ทำหน้าที่รักษาความสัมพันธ์ของข้อมูลภายในฐานข้อมูลให้มีความถูกต้องอยู่เสมอ

5. ทำหน้าที่เก็บรายละเอียดต่างๆที่เกี่ยวข้องกับข้อมูลภายในฐานข้อมูลไว้ใน Data Dictionary ซึ่งรายละเอียดเหล่านี้มักถูกเรียกว่า ข้อมูลของข้อมูล (Metadata)
6. ทำหน้าที่ควบคุมให้ฐานข้อมูลทำงานอย่างถูกต้องและมีประสิทธิภาพ

5.1.4 ประโยชน์ของฐานข้อมูล

1.สามารถลดความซ้ำซ้อนของข้อมูล โดยไม่จำเป็นต้องจัดเก็บข้อมูลที่ซ้ำๆกันไว้ในระบบแฟ้มข้อมูลของแต่ละหน่วยงานเหมือนเช่นเดิม แต่สามารถนำข้อมูลมาใช้ร่วมกันในคุณลักษณะ Integrated แทน

2.สามารถหลีเลี่ยงความขัดแย้งของข้อมูล เนื่องจากไม่จำเป็นต้องเก็บข้อมูลที่ซ้ำซ้อนกันในหลายแฟ้มข้อมูลดังนั้นในการแก้ไขข้อมูลในแต่ละชุดจะไม่ก่อให้เกิดค่าที่แตกต่างกันได้

3. แต่ละหน่วยงานในองค์กรสามารถใช้ข้อมูลร่วมกันได้

4. สามารถกำหนดให้ข้อ มูลมีรูปแบบที่เป็นมาตรฐานเดียวกันได้ เพื่อให้ผู้ใช้ข้อมูลในฐานข้อมูลชุดเดียวกันสามารถเข้าใจและสื่อสารถึงความหมายเดียวกัน

5. สามารถกำหนดความปลอดภัยให้กับข้อมูลได้ โดยกำหนดความสามารถในการเรียกใช้ข้อมูลของผู้ใช้แต่ละคนให้แตกต่างตามความรับผิดชอบ

6. สามารถรักษาความถูกต้องของข้อมูลได้ โดยระบุเกณฑ์ในการกำหนดความผิดพลาดที่อาจเกิดขึ้นจากการป้อนข้อมูลผิด

7. สามารถตอบสนองความต้องการใช้ข้อมูลได้หลายรูปแบบ

8. ทำให้ข้อมูลเป็นอิสระจากโปรแกรมที่เราใช้งานข้อมูลนั้น ซึ่งส่งผลให้ผู้พัฒนาโปรแกรมสามารถแก้ไขโครงสร้างข้อมูล โดยไม่กระทบต่อโปรแกรมที่เรียกใช้ข้อมูลนั้น

5.1.5 ภาษาทางด้านฐานข้อมูล

ผลิตภัณฑ์ทางด้านฐานข้อมูลที่มีโครงสร้างในแบบRelational ซึ่งมีจำหน่ายอยู่ในท้องตลาดสิ่งที่จำเป็นสำหรับผลิตภัณฑ์เหล่านี้ ได้แก่ การมีภาษาทางด้านฐานข้อมูลหรือที่เรียกว่า Query Language เช่นภาษา Structured Query Language (SQL) ภาษา Query – by – Example (QBE) และภาษา Quel เป็นต้นภาษาเหล่านี้ได้ถูกพัฒนาขึ้นจากแนวคิดที่ต่างกัน เช่น ภาษา QBE ซึ่งถูกพัฒนาขึ้นจากแนวคิด Relational Calculus ส่วนภาษา Quel ถูกพัฒนาขึ้นจากแนวคิดของ Tuple Relational Calculus ในขณะที่ภาษา SQL ถูกพัฒนาขึ้นจากแนวคิดของ Relational Calculus และ Relational Algebra เป็นหลัก แต่อย่างไรก็ตามภาษาที่ได้รับความนิยมมากที่สุดได้แก่ ภาษา Structured Query Language

Structured Query Language เป็นภาษาทางด้านฐานข้อมูล ที่นิยมมากที่สุดภาษาหนึ่ง โดยมักถูกเรียกย่อๆว่า SQL เริ่มต้นพัฒนาครั้งแรกโดย San Jose Research Laboratory ของบริษัท IBM

โดยแรกเริ่มเรียกว่า Sequel และได้ถูกนำมาใช้เป็นต้นแบบของภาษา SQL ของผลิตภัณฑ์ทางด้านฐานข้อมูลจำนวนมาก แต่อย่างไรก็ตามภาษา SQL ยังมีความแตกต่างกันในรายละเอียดทางด้านการใช้งาน คำสั่งต่างๆของภาษา SQL สามารถแบ่งตามลักษณะการใช้งานออกได้เป็น 3 กลุ่ม ดังนี้

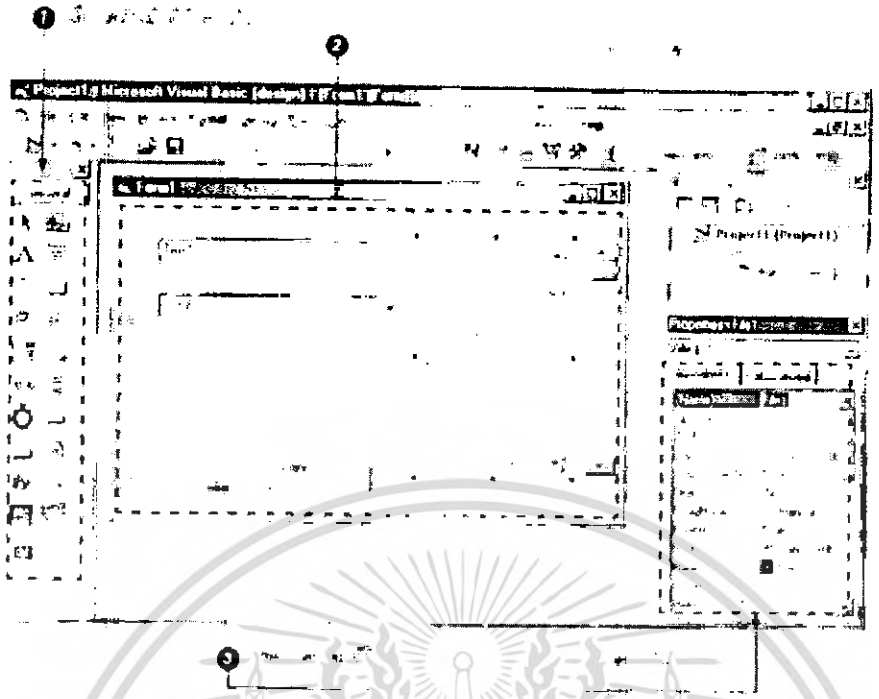
1. Data Definition Language : DDL เป็นกลุ่มคำสั่งที่ใช้สร้างฐานข้อมูล หรือใช้สำหรับกำหนดโครงสร้างให้กับ Relation ภายในบานข้อมูล เช่น การเพิ่ม เปลี่ยนแปลง ลบ เป็นต้น
2. Data Manipulation : DML เป็นกลุ่มคำสั่งที่พัฒนาขึ้นตามแนวคิดของ Relational Algebra และ Record Relation Calculus โดยประกอบด้วยคำสั่งสำหรับเพิ่ม ลบ หรือเปลี่ยนแปลงข้อมูลในฐานข้อมูล
3. Data Query Language เป็นกลุ่มคำสั่ง DML ประเภทหนึ่งที่ใช้ในการเลือกข้อมูลจาก Relation ขึ้นมาแสดงผลตามรูปแบบที่ต้องการ

5.2 Visual Basic

ก่อนทำความรู้จักกับ VB6 เราจะต้องทราบความหมายของการเขียนโปรแกรมก่อน ซึ่งการเขียนโปรแกรมก็คือการสั่งให้คอมพิวเตอร์ทำงานตามที่เราต้องการ เช่น โปรแกรมฝึกพิมพ์ดีด เป็นโปรแกรม ที่สั่งให้โปรแกรมตอบโต้กับการกดแป้นคีย์บอร์ด เพื่อสอนให้ผู้ใช้พิมพ์ดีด เป็นต้น

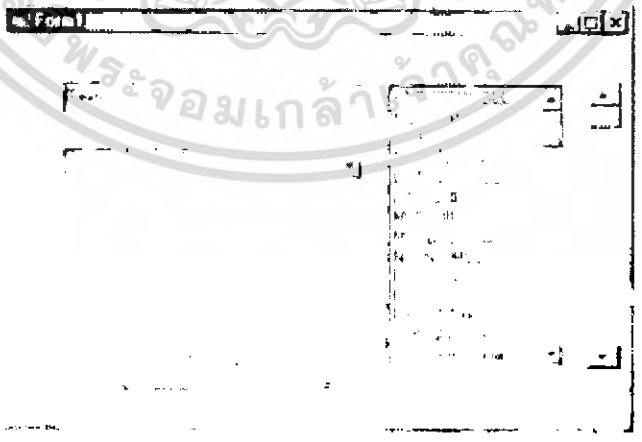
สำหรับ VB6 เป็นเครื่องมือที่ใช้สร้างโปรแกรมบนระบบปฏิบัติการ Windows ที่ใช้งานง่ายโดยในการสร้างโปรแกรมใน VB6 นั้นจะเป็นการเลือกเครื่องมือต่างๆมาออกแบบหน้าจอของโปรแกรมที่เราจะสร้างขึ้น ซึ่งเราเรียกการเขียนโปรแกรมลักษณะนี้ว่า Visual Programming การเขียนโปรแกรมแบบนี้ เราไม่จำเป็นต้องเขียนคำสั่งต่างๆมากนักก็สามารถสร้างโปรแกรมได้อย่างรวดเร็ว

การเขียนโปรแกรมแบบ Visual Programming นั้นมีลักษณะดังรูป



รูปที่ 5.4 การเขียนโปรแกรมแบบ Visual Programming

1. เลือกเครื่องมือที่ต้องการใช้งาน
2. นำเครื่องมือนั้นมาวางบนหน้าต่างโปรแกรมของเรา
3. กำหนดคุณสมบัติต่างๆเช่น สี และรูปแบบตัวอักษร เป็นต้น
4. เมื่อสั่งรันโปรแกรม หน้าต่างของโปรแกรมจะมีลักษณะคล้ายกับที่ออกแบบไว้



รูปที่ 5.5 หน้าต่างของโปรแกรมเมื่อทำการสั่งรัน

หลังจากที่เราได้ออกแบบหน้าจอโปรแกรมตามวิธีดังกล่าวมาแล้วเราจำเป็นต้องเขียนโปรแกรมควบคุมการทำงานด้วยโดยใช้ภาษา Basic (ย่อมาจาก Beginners All-Purpose Symbolic Instruction Code) ซึ่งเป็นภาษาที่ใช้งานง่ายเหมาะสำหรับผู้เริ่มต้นศึกษาการเขียนโปรแกรมบน Windows

นอกจากนี้โปรแกรมอื่นๆของไมโครซอฟท์เช่น Microsoft Excel , Microsoft Access เป็นต้นก็ใช้ภาษา Visual Basic เป็นส่วนหนึ่งของโปรแกรมด้วยจึงสามารถใช้ความรู้ที่มีอยู่เกี่ยวกับ VB6 ในการเขียนโปรแกรมต่างๆด้วยเครื่องมือเหล่านั้นได้ซึ่งจะช่วยให้การลงทุนศึกษา VB6 นั้นคุ้มค่า

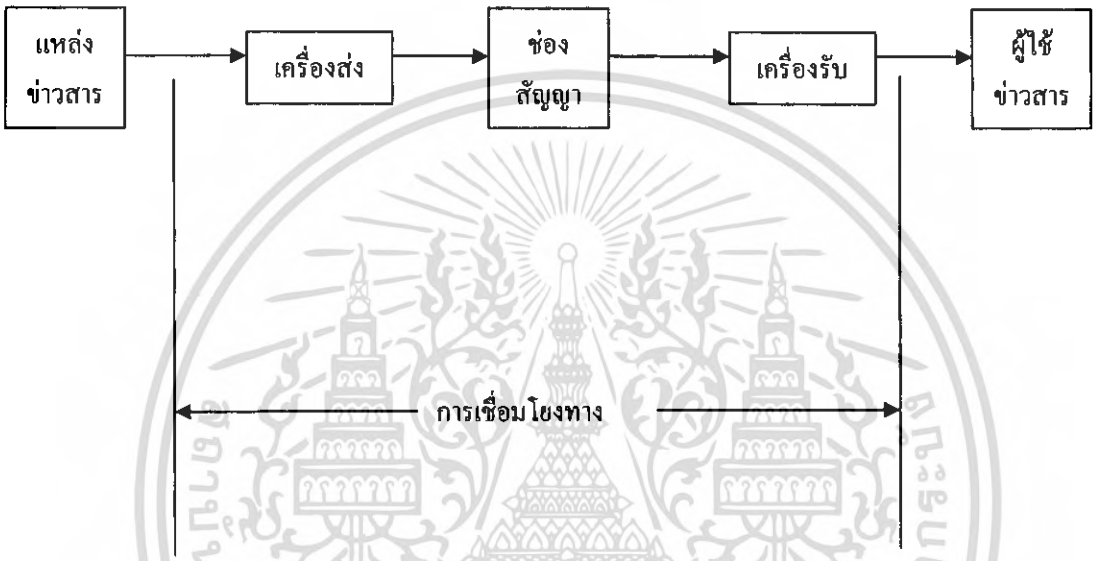
สำหรับ VB6 นั้นเป็นเครื่องมือที่สร้างโปรแกรมต่างๆได้หลากหลายดังต่อไปนี้

1. โปรแกรมทั่วไปที่รันบนระบบปฏิบัติการ Windows โดยเราสามารถสร้างโปรแกรมทางด้านกราฟฟิก โปรแกรมจัดการไฟล์ โปรแกรมคำนวณพื้นฐาน ให้ตรงกับความต้องการของเราได้
2. โปรแกรมทางด้านบานข้อมูล VB6 นั้นช่วยให้การสร้างโปรแกรมฐานข้อมูลเป็นเรื่องง่ายเนื่องจากมีเครื่องมือต่างๆเกี่ยวกับฐานข้อมูลอย่างครบถ้วน เช่นเครื่องมือในการติดต่อกับฐานข้อมูลทั้ง Microsoft Access หรือฐานข้อมูลบนระบบ Client-Server เช่น Microsoft SQL Server โดยการติดต่อกับฐานข้อมูลนั้นเพียงแต่เรากำหนดตำแหน่งของฐานข้อมูลพร้อมทั้งข้อมูลที่จำเป็นในการติดต่อกับฐานข้อมูลเท่านั้น เราก็สามารถติดต่อกับฐานข้อมูลนั้นได้ทันที
3. คอมโพเนนท์ทางด้าน Active X ซึ่งได้แก่ Active X Component , Active X Control และ Active X Document ซึ่งเป็นเครื่องมือที่ช่วยให้เราสามารถนำส่วนของโปรแกรมที่เราได้สร้างแล้วไปใช้ในโปรแกรมอื่นๆได้เช่น Microsoft Excel เป็นต้น
4. โปรแกรมที่รันบนอินเตอร์เน็ตหรืออินทราเน็ตผ่านทาง Web Browser ด้วยความสามารถของ VB6 ช่วยให้เราสามารถสร้างโปรแกรมที่รันบนอินเตอร์เน็ตได้อย่างง่ายดาย โดยที่ไม่ต้องเรียนรู้การเขียนคำสั่งด้วยภาษา HTML หรือภาษา Script ที่ใช้งานบนอินเตอร์เน็ต

บทที่ 6 ทฤษฎีการสื่อสาร

6.1 ระบบสื่อสารทางอิเล็กทรอนิกส์

การสื่อสารอิเล็กทรอนิกส์ หมายถึง การส่ง-การรับ และการประมวลผลของข้อมูล หรือข่าวสารระหว่างจุด 2 จุด หรือมากกว่า ด้วยการใช้อิเล็กทรอนิกส์



รูปที่ 6.1 บล็อกแสดงการสื่อสาร

ตัวกลางการสื่อสาร เป็นช่องทางหรือตัวกลางซึ่งสัญญาณของระบบสื่อสารใช้เป็นทางผ่านจากจุดส่งไปยังจุดรับสัญญาณ โดยชนิดของตัวกลางสื่อสารมี 2 แบบคือ

1)แบบมีสาย ซึ่งอาจจะเป็นสายตัวนำไฟฟ้า 1 คู่ หรือ เส้นใยนำแสงก็ได้

2)แบบไร้สาย สัญญาณของระบบสื่อสารแบบนี้จะอยู่ในรูปของคลื่นแม่เหล็กไฟฟ้า

เครื่องส่ง เป็นอุปกรณ์หรือวงจรทางอิเล็กทรอนิกส์ ที่ถูกออกแบบมาสำหรับการแปลงสัญญาณจากแหล่งกำเนิดสัญญาณที่จะสื่อสาร ให้กลายเป็นสัญญาณที่มีรูปแบบ และระดับพลังงานที่เหมาะสมกับตัวกลางการสื่อสารของแต่ละแบบ

เครื่องรับ เป็นอุปกรณ์ และวงจรอิเล็กทรอนิกส์ ที่ทำหน้าที่แปลงสัญญาณที่รับมาได้จากตัวกลาง ให้กลายเป็นสัญญาณที่มีรูปแบบ และระดับพลังงานที่เหมาะสมกับอุปกรณ์ปลายทางด้านรับ

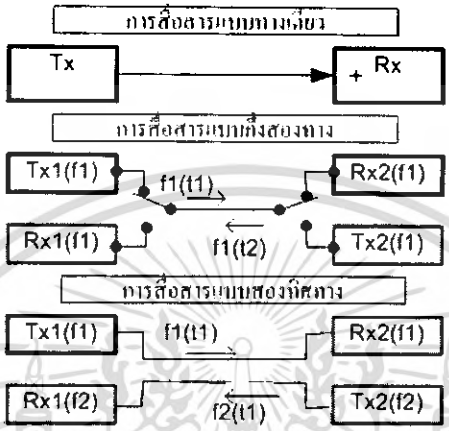
ถ้ามีสัญญาณมากกว่าหนึ่งสัญญาณในอาณาบริเวณเดียวกัน และสัญญาณเหล่านั้นมีค่าแถบความถี่ที่ซ้อนทับกันจะทำให้เกิดการรบกวนซึ่งกันและกัน

รูปแบบของการสื่อสาร ซึ่งแบ่งตามทิศทางของการสื่อสารหรือช่องสื่อสารได้ 3 แบบ คือ

1)แบบทิศทางเดียว มีฝ่ายหนึ่งจะส่งเพียงอย่างเดียว และอีกฝ่ายหนึ่งจะรับเพียงอย่างเดียว ซึ่งการสื่อสารแบบนี้จะใช้ช่องสื่อสารเพียงช่องเดียว

2)แบบกึ่งสองทาง เป็นการสื่อสาร 2 ทาง ในเวลาที่ต่างกันอีกฝ่ายหนึ่งจะส่ง และอีกฝ่ายหนึ่งจะรับ ซึ่งการสื่อสารแบบนี้จะใช้ช่องสื่อสารเพียงช่องเดียวเช่นกัน

3)แบบสองทาง เป็นการสื่อสารแบบ 2 ทิศทาง โดยทั้งสองฝ่ายสามารถรับและส่งได้พร้อมกัน ในเวลาเดียวกัน ซึ่งการสื่อสารแบบนี้จะใช้ช่องสื่อสาร 2 ช่อง



รูปที่ 6.2 การสื่อสารแบบต่างๆ

6.2 การมอดดูเลต

การมอดดูเลต คือกระบวนการเลื่อนแถบความถี่ของสัญญาณ ซึ่งในขบวนการมอดดูเลตนี้เราใช้คลื่นพาห์ที่มีความถี่สูงเป็นพาหะ แล้วเปลี่ยนแปลงคุณสมบัติบางอย่างของพาหะด้วยสัญญาณที่จะทำการส่ง

การมอดดูเลตให้กับคลื่นพาหะแบ่งออกได้เป็น 2 แบบ

1.มอดดูเลตทางแอมพลิจูด (amplitude modulation หรือ AM)

ขนาดของคลื่นพาห์จะเปลี่ยนแปลงตามขนาดของสัญญาณ

2.มอดดูเลตทางมุม แบ่งย่อยได้ 2 รูปแบบ

2.1มอดดูเลตทางความถี่ (Frequency modulation หรือ FM) ความถี่ของคลื่นพาห์จะเปลี่ยนแปลงตามขนาดของสัญญาณ

2.2มอดดูเลตทางเฟส (phase modulation หรือ PM) เฟสของคลื่นพาห์จะเปลี่ยนแปลงตามขนาดของสัญญาณ

6.3การดีมอดดูเลตหรือการดีเทค เป็นกระบวนการย้อนกลับของการมอดดูเลต ซึ่งก็คือการแยกสัญญาณออกจากคลื่นพาห์

จุดประสงค์ของการมอดูเลชั่น

1. ทำให้สัญญาณมีกำลังสูง

2. ทำให้สัญญาณมีความถี่สูงขึ้น ซึ่งเหมาะสำหรับการรับส่งสัญญาณมากขึ้นเพราะใช้เสาอากาศที่สั้นลงได้

3. สามารถแบ่งความถี่ให้หลายๆสัญญาณส่งพร้อมกันภายใต้ตัวพาห์ตัวเดียวกันได้ เรียกว่า การมัลติเพล็กซิง

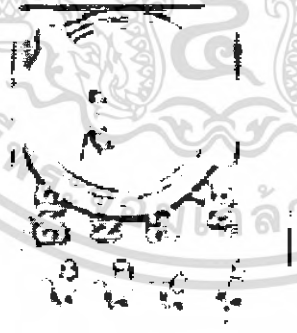
4. ทำให้สัญญาณมีความต้านทานการรบกวนของสัญญาณรบกวนได้ดีขึ้น

ในการติดต่อกันจะใช้ RF เบอร์ TLP434 และRLP434 ซึ่งเป็นโมดุลในการรับ-ส่งข้อมูลด้วยคลื่นความถี่วิทยุที่ 315 MHz เป็นการมอดดูเลทสัญญาณแบบ ASK (Amplitude Shift Keying) โดย TLP434 เป็นโมดุลตัวส่งที่มี 4 ขา ประกอบไปด้วย

RF Transmitter

1. GND
2. Data In
3. Vcc
4. Antenna (RF Output)

ดังรูปที่ 3 แสดงขา 1 , 2 , 3 และขา 4 เรียงจากซ้ายไปขวาตามลำดับ



รูปที่ 6.3 แสดง TLP434 ภาคส่ง RF

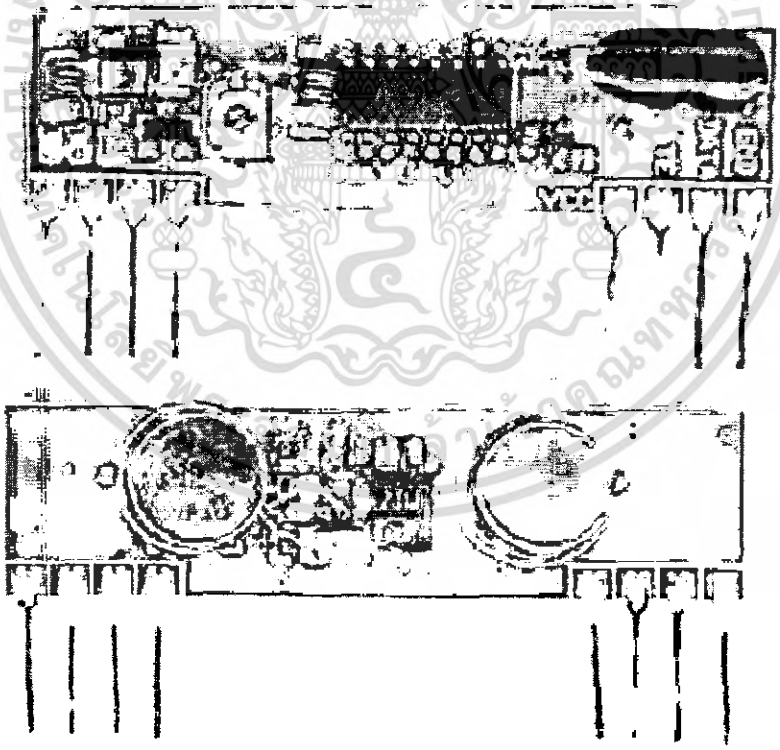
เมื่อทำการ Set ค่า A_0 ถึง A_7 แล้วป้อนข้อมูลเข้าทาง D_8–D_{11} และทำการ Set ขา TE ของ HT12E ให้เป็น 0 แล้วข้อมูลจะถูก Encoder มาที่ขา D_{OUT} ของ HT12E และเมื่อเรานำขานี้ไปต่อกับขา Data In ของ TLP434 ก็จะกลายเป็นการส่งข้อมูลออกไปทาง Antenna ของ TLP434

RF Receiver

RLP434 ซึ่งเป็นโมดุลที่มี 8ขา ประกอบด้วย

1. GND
2. Digital Data Output
3. Linear Output/Test
4. Vcc
5. Vcc
6. GND
7. GND
8. Antenna

ดังรูปที่ 5 เรียงตามขาจากซ้ายไปขวา 1, 2, 3, 4, 5, 6, 7 และ 8 ตามลำดับ

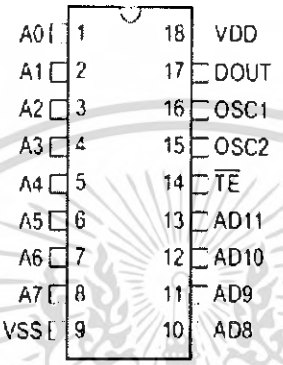


รูปที่ 6.4 แสดง RLP434 ภาครับ RF

การประยุกต์ใช้งานเราจะใช้งานร่วมกับ IC เบอร์ HT12E และ HT12D ซึ่งเป็นตัว Encoder และDecoder เพื่อที่จะทำให้สามารถเพิ่มช่องสัญญาณในการรับ-ส่งได้

เมื่อข้อมูลไปถึงภาครับ โดยรับจาก Antenna ของ RLP434 ซึ่งข้อมูลนี้ก็คือข้อมูลที่ถูก Encoder มาแล้วเราก็นำข้อมูลนี้ผ่านเข้าไปที่ HT12D (ที่ถูก Set ขา A_0 และ A_7 ให้ตรงกับตัวส่งแล้ว) ทางขา Data In ของ HT12D ข้อมูลที่ถูก Decode แล้วก็จะปรากฏอยู่ที่ D_8–D_{11} ของ HT12D ในขณะที่ขา VT ของ HT12D ก็จะเป็น Logic 1 เพื่อแสดงว่า HT12D สามารถ Decode สัญญาณได้

HT12E IC Encoder ของภาคส่ง



รูปที่ 6.5 แสดงภาพ IC HT12E

—Address/data programming (preset)

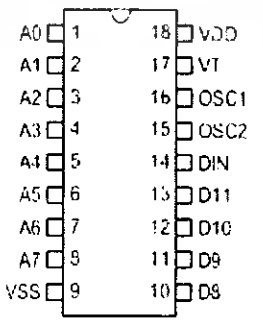
สถานะของแต่ละขาของ Address/data สามารถกำหนดค่าจำเพาะได้โดยให้ preset เป็นลอจิก Low หรือ High ถ้าสัญญาณ transmission-enable นำมาใช้ encoder สแกนและส่งผ่านสถานะของ 12 บิต ของ address/data แบบอนุกรมผ่าน A0 ถึง AD11 สำหรับ HT12E encoder

เนื่องจากการส่งผ่านข้อมูล บิตเหล่านี้จะถูกส่งด้วยกระบวนการซิงโครนัส ถ้า Trigger signal ไม่ทำงาน IC จะเข้าสู่โหมด standby คือไม่ทำงานและกระแสจะลดลงน้อยกว่า 1uA สำหรับแหล่งจ่ายแรงดัน 5 โวลต์

Transmission enable

สำหรับ Encoder HT12E การส่งผ่านจะถูกกำหนดโดย ใช้สัญญาณ Low ให้กับขา TE

HT12D IC Decoder ของภาครับ



รูปที่ 6.6 HT12D IC Decoder ภาครับ

การทำงานของ HT12D

12 บิต อนุกรมของ Decoder สามารถแบ่งได้หลายกรณีโดยการรวม ขา Address และขา Data โดยการจับคู่กับ 12 บิต อนุกรมของ encoder

Decoder รับข้อมูลที่ส่งมาจาก encodcr แปล N บิตแรกของรหัสเป็น Address และ 12-N บิต สุดท้ายเป็น Data เมื่อ N เป็นรหัส Address สัญญาณที่ขา DIN active จึงจะทำการ decode ข้อมูล

ข้อมูล Decoder จะ check ข้อมูลจาก Address ที่ได้รับมาอย่างค่อเนื่อง 3 ครั้ง ถ้า Address Code ตรงกับ Address ของ decoder 12-N บิตของข้อมูลจะถูก decode ไปที่ขา Output และขา VT จะเป็น High เพื่อบอกว่าข้อมูลถูกต้อง

ดังนั้นขา VT จะเป็น High เมื่อข้อมูลถูกต้องเท่านั้น โดยปรกติจะเป็น Low

บทที่ 7 การออกแบบและการทำงานของวงจร

7.1 องค์ประกอบโดยรวมของระบบ

การประยุกต์ใช้งานสมาร์ตการ์ดในโครงงานนี้ได้ถูกออกแบบมาเพื่อใช้งานด้านการรักษาความปลอดภัยและการอำนวยความสะดวก นั่นคือ ในการรักษาความปลอดภัยจะเป็นการนำเอาสมาร์ตการ์ดนี้มาใช้ในการเข้า ออกสถานที่ต่างๆเช่น หอพักส่วนตัว หรือสถานบริการที่เจ้าของบัตรเป็นสมาชิกอยู่ เป็นต้น ซึ่งนอกจากจะเป็นการป้องกันการเข้าออกสถานที่นั้นๆแล้ว ยังสามารถตรวจสอบการใช้สถานที่นั้นๆของเจ้าของบัตร ไม่ว่าจะเป็นในเรื่องของจำนวนครั้งที่ใช้ เวลาที่ใช้ และข้อมูลอื่นๆที่เป็นส่วนในการเพิ่มความปลอดภัยให้มากขึ้น นอกจากนี้เรายังสามารถเอาไปประยุกต์ใช้ในอีกหลายด้าน เช่นการนำไปประยุกต์ใช้เป็น บัตรที่สามารถชำระค่าบริการต่างๆในบริเวณที่เปิดให้บริการ เพื่ออำนวยความสะดวกให้กับผู้ใช้บริการมากขึ้น ซึ่งก็ขึ้นอยู่กับผู้ออกแบบจะนำไปประยุกต์ใช้งาน

สิ่งที่สำคัญที่สุดสำหรับการสร้างระบบนี้มี 2 ส่วนคือ ส่วนของวงจรที่ทำหน้าที่ในการอ่านข้อมูลที่เก็บอยู่ในสมาร์ตการ์ด ซึ่งในที่นี้หมายถึงบัตร TOT Card ที่ไม่สามารถชำระค่าโทรศัพท์ได้แล้วนั่นเอง และส่วนของโปรแกรมประยุกต์บนคอมพิวเตอร์ที่ทำการวิเคราะห์ข้อมูลที่เครื่องอ่านสามารถอ่านได้ เพื่อใช้ในการรักษาความปลอดภัยและอำนวยความสะดวกต่อผู้ใช้นั่นเอง

7.2 หลักการทำงานเครื่องอ่านบัตร TOT Card

ในรูปที่ 5.1 คือวงจรที่สมบูรณ์ของเครื่องอ่านรหัสบัตร TOT Card สำหรับดูแลความเรียบร้อยในการเข้าออกสถานที่ต่างๆ ซึ่งการทำงานของวงจรนี้ถูกควบคุมด้วย ไมโครคอนโทรลเลอร์ตระกูล MCS – 51 เบอร์ AT 89C51 หน้าที่หลักทั่วไปของไมโครคอนโทรลเลอร์ตัวนี้คือ ควบคุมการรับส่งข้อมูล กับเครื่องคอมพิวเตอร์ และสามารถจัดการด้านข้อมูลในกรณีที่มีการจัดเก็บข้อมูลหรือมีการบันทึกข้อมูลบางอย่างไว้เป็นฐานข้อมูล กลไกในการที่ AT 89C51 จะสื่อสารกับสมาร์ตการ์ดซึ่งในที่นี้คือบัตร TOT Card นั้น เกิดจากรูปแบบสัญญาณที่พอร์ต P2.0 –P2.4 (ขาที่ 21-25 ของ IC ตามลำดับ) ซึ่งเป็นไปตามที่กล่าวมาแล้วในบทที่แล้ว

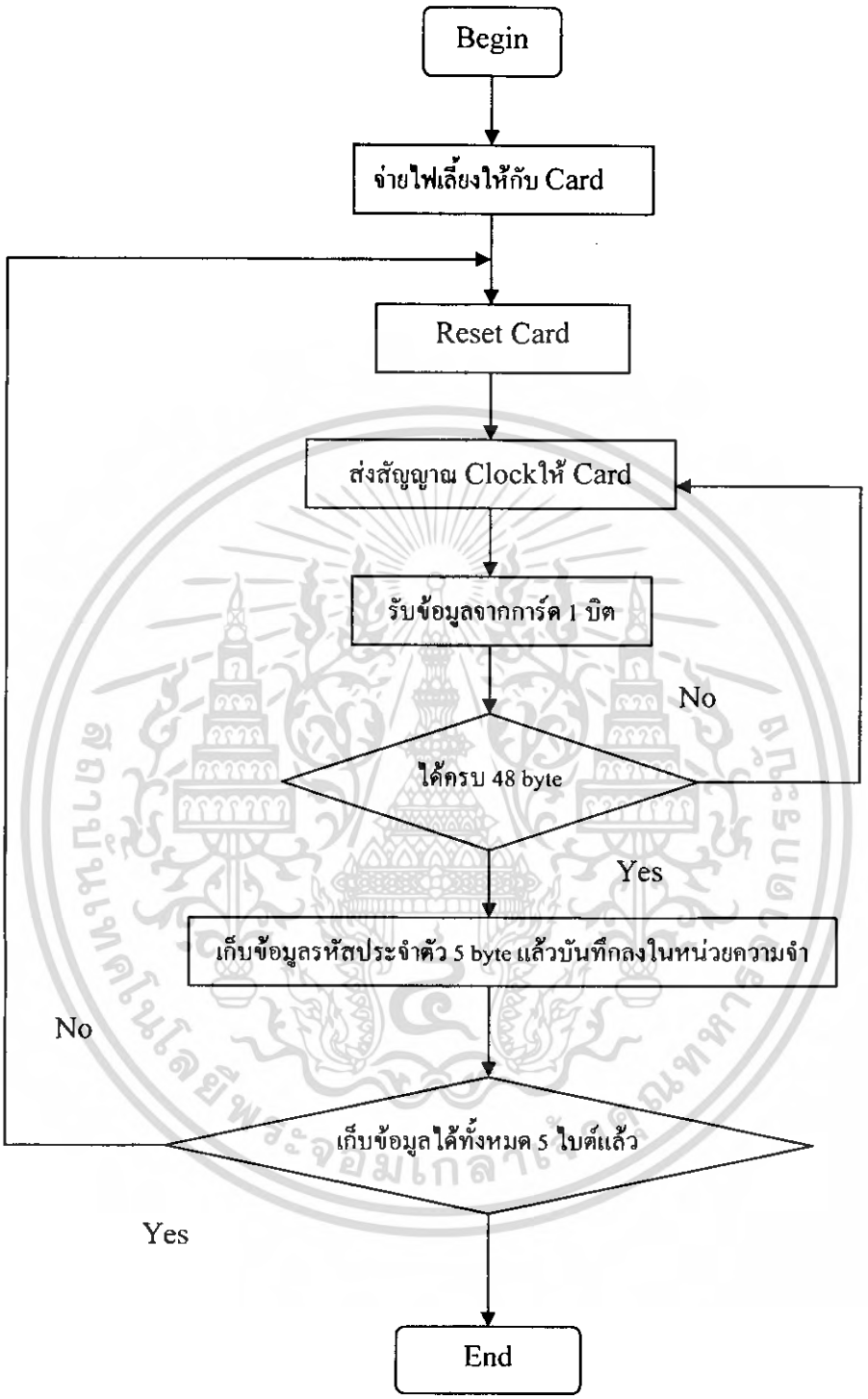
สำหรับการสื่อสารข้อมูลกับสมาร์ตการ์ดนั้นจะมี AT 89C51 ควบคุมการจ่ายไฟให้กับบัตรทางบิต P2.1 สัญญาณจากบิตนี้จะถูกส่งไปทริกให้ทรานซิสเตอร์ 2N4403 ซึ่งทำงานเป็นสวิตช์ตัดต่อแหล่งจ่ายไฟให้บัตรทำงานหรือหยุดทำงาน บิต P2.2 และบิต P2.3 จะทำหน้าที่เป็นขาสัญญาณ

รีเซตและสัญญาณนาฬิกา ตามลำดับ ส่วนบิต P2.0 ถูกใช้งานในการตรวจสอบว่าบัตรถูกเสียบเข้ามายังช่องรับแล้วหรือไม่ จากสถานะสวิทซ์ ที่ถูกซ่อนอยู่ภายในซ็อกเกตรับบัตรสมาร์ตการ์ด

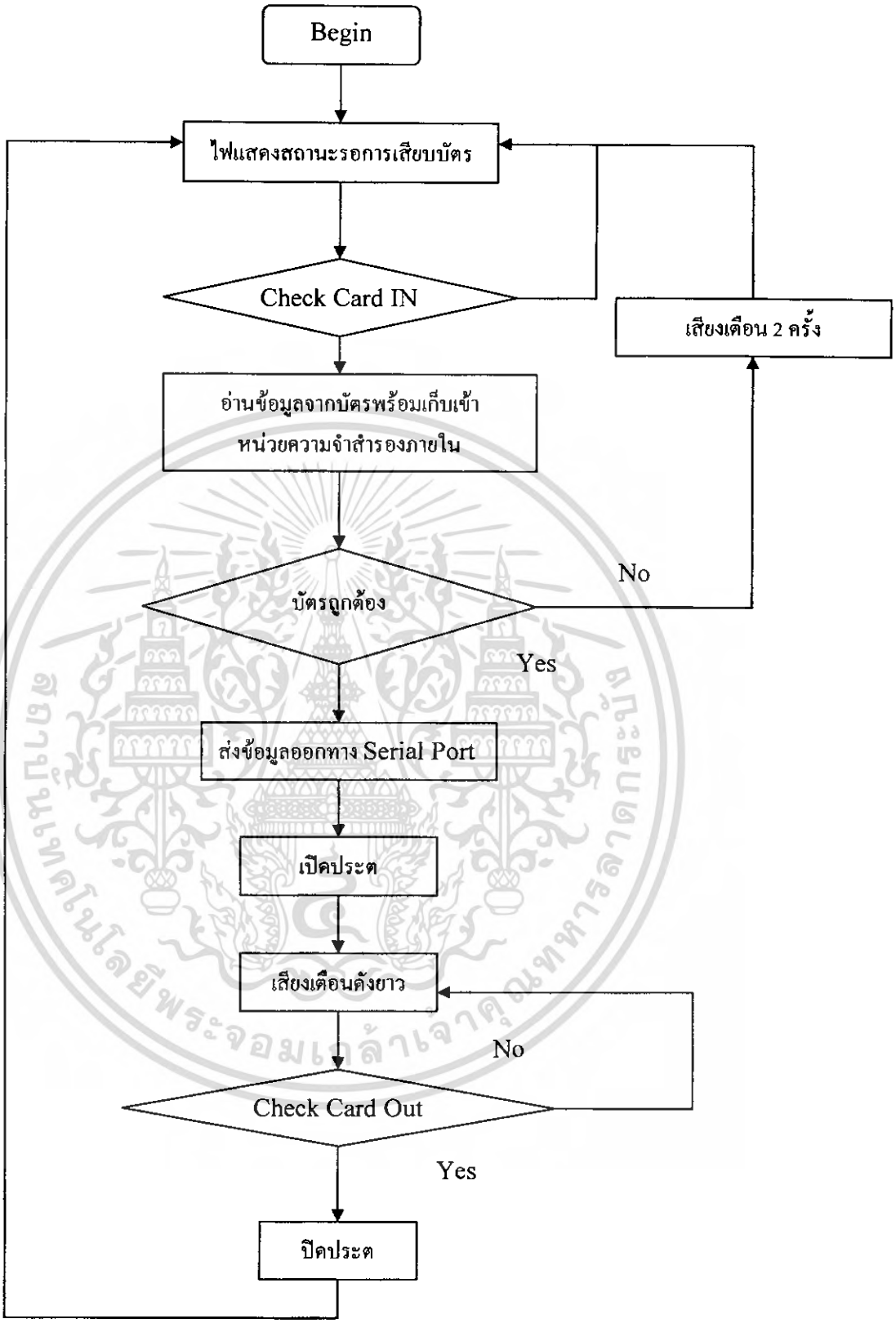
การสื่อสารระหว่าง AT 89C51 กับพอร์ตอนุกรม RS – 232 ของเครื่องคอมพิวเตอร์ในแง่ของกระบวนการใช้จะถูกจัดการโดยโปรแกรมที่เขียนและบันทึกไว้ใน AT 89C51 ส่วนในแง่ของระดับสัญญาณที่ใช้ในการสื่อสารจะถูกจัดการด้วยไอซี MAX 232 ซึ่งจะทำการแปลงระดับแรงดันจากที่ระดับ 0 และ 5 โวลต์ ไปเป็น -12 และ +12 โวลต์ เพื่อช่วยให้คอมพิวเตอร์สามารถเข้าใจข้อมูลที่ AT 89C51 ส่งไปได้

หน้าที่อีกอย่างของ AT 89C51 คือการควบคุมอุปกรณ์ภายนอก อย่างเช่น บัซเซอร์ , หลอดแอลอีดี , ออปโตไอโซเลเตอร์ , จอแอลซีดี และ คีย์แพด (ในกรณีที่มีการติดตั้งเพิ่มเติม) นั่นคือ บัซเซอร์จะถูกควบคุมจากบิต P1.7 บิตนี้จะจ่ายสัญญาณเป็นชุดของพัลส์ความถี่ ออกมายังทรานซิสเตอร์ 2N4403 เพื่อขับให้บัซเซอร์กำเนิดเสียง ส่วนหลอดแอลอีดี (LED1-LED4) จะถูกขับโดยตรงจาก AT 89C51 โดยจะจำกัดกระแสที่จ่ายให้แอลอีดีแต่ละตัวด้วยความต้านทาน 220 โอห์ม นอกจากนั้นยังมีส่วนอื่นๆอีกคือ ออปโตไอโซเลเตอร์แบบไดแอค เบอร์ MOC 3020M ควบคุมโดย P1.6 เพื่อจะนำไปทริกขาเกทของไดรแอค เบอร์ Q4010L4 แล้วนำไปควบคุมอุปกรณ์ไฟฟ้าที่จุดที่ได้ตามต้องการ

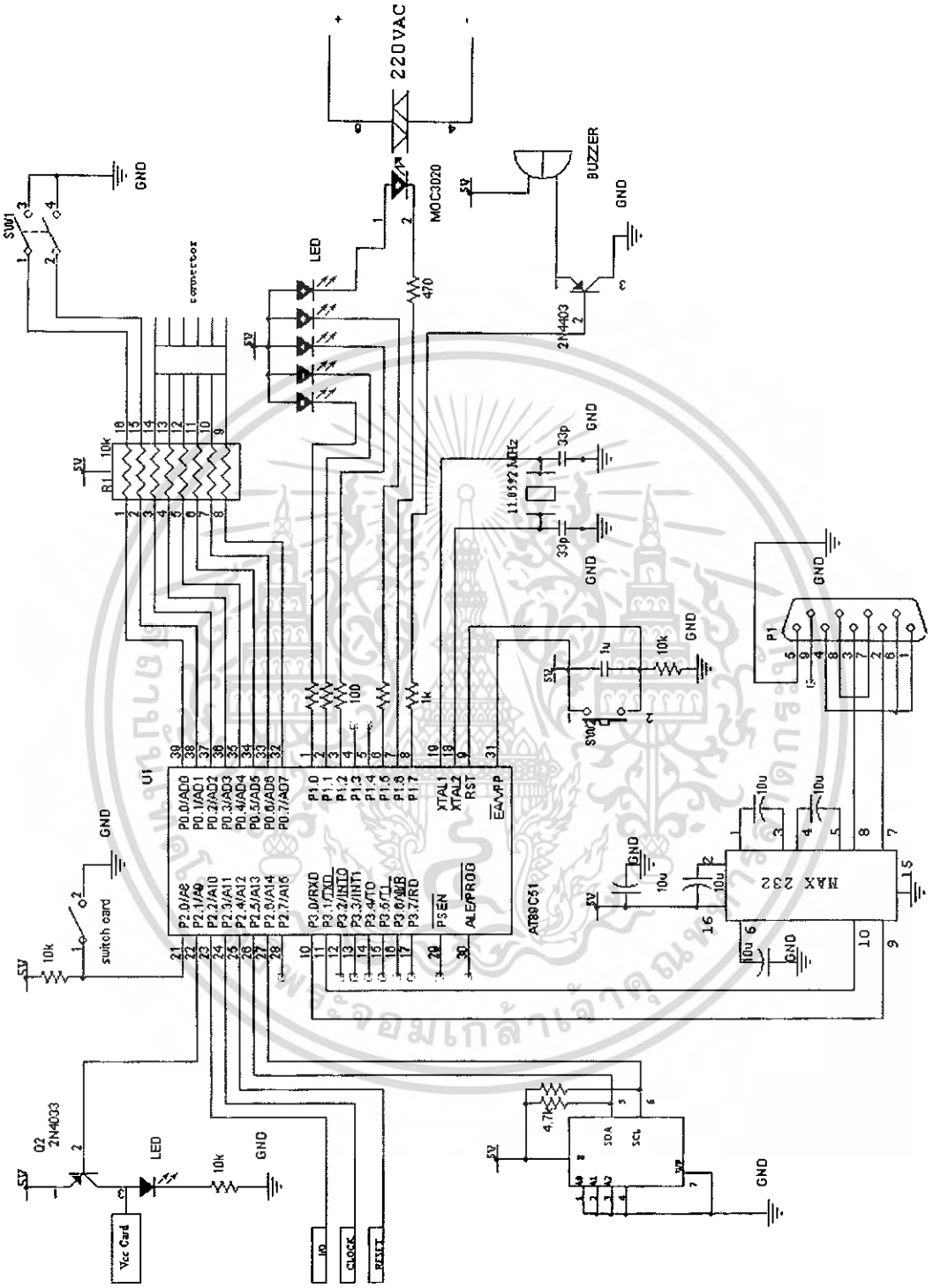
สำหรับในส่วนของแหล่งจ่ายไฟของทั้งวงจรนั้นจะใช้ Adaptor ที่มีขายทั่วไปที่มีแรงดันเอาท์พุทเท่ากับ 5 โวลต์ ทำให้ไม่จำเป็นต้องต่อวงจรเพิ่มเติมก็สามารถนำมาใช้ได้ทันที



รูปที่ 7.1 โฟลวชาร์ตการทำงานของโปรแกรมในส่วนของการอินเตอร์เฟสกับบัตร TOT No



รูปที่ 7.2 โฟลวชาร์ตการทำงานของวงจรเครื่องอ่านหมายเลขบัตร TOT

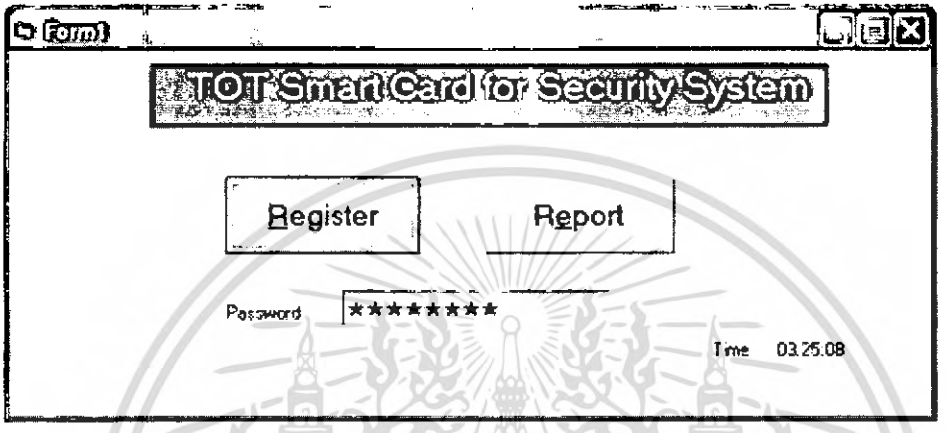


รูปที่ 7.3 วงจรเครื่องอ่านบัตรสมาร์ตการ์ด

7.2 หลักการทำงานของโปรแกรมประยุกต์

7.2.1 ส่วนประกอบภายในหน้าต่างโปรแกรม

การทำงานของโปรแกรมประยุกต์ที่ได้สร้างขึ้นนั้นมีลักษณะการทำงานที่สอดคล้องกับเครื่องอ่านบัตรสมาร์ทการ์ด คือ โหมดการลงทะเบียนของผู้ใช้บัตร และโหมดการผ่านเข้าออกประตู โดยเมื่อทำการรันโปรแกรมจะปรากฏหน้าต่างแรกเพื่อให้ทำการป้อนรหัสผ่านเพื่อเป็นการจำกัดการเข้ามาใช้โปรแกรมในการแก้ไขข้อมูลต่างๆ



รูปที่ 7.1 หน้าแรกของโปรแกรม

จากนั้นก็จะปรากฏหน้าต่างที่สองซึ่งจะใช้เป็นหน้าหลักที่ใช้ทั้งในการลงทะเบียนผู้ใช้และใช้แสดงข้อมูลเมื่อมีผู้ผ่านเข้าออก ซึ่งจะประกอบไปด้วยส่วนต่างๆดังนี้ คือ

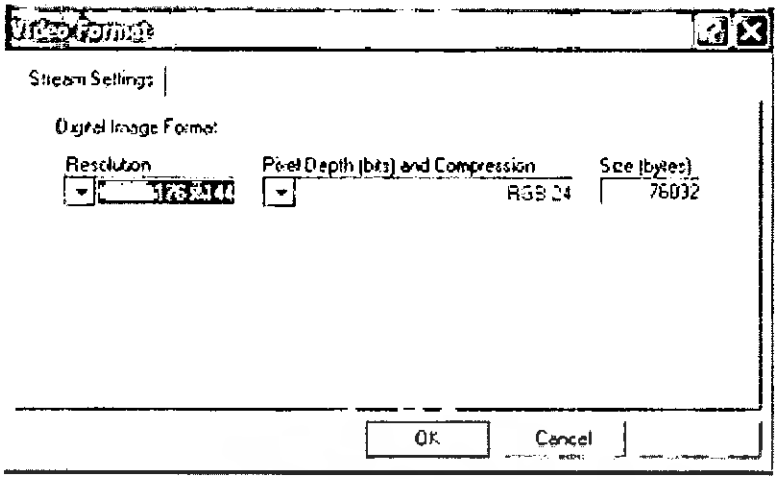
-กลุ่มของ Text Box ที่ใช้แสดงข้อมูลประวัติของผู้ใช้ที่ทำการเสียบบัตร

-กลุ่มของ ปุ่มคำสั่งต่างๆที่ใช้ในการจัดการด้านข้อมูลประกอบด้วย ปุ่มเพิ่มข้อมูล (Add) จัดเก็บข้อมูล (Update) , ลบข้อมูล (Delete) , Refresh , ออกจากโปรแกรม (Close)

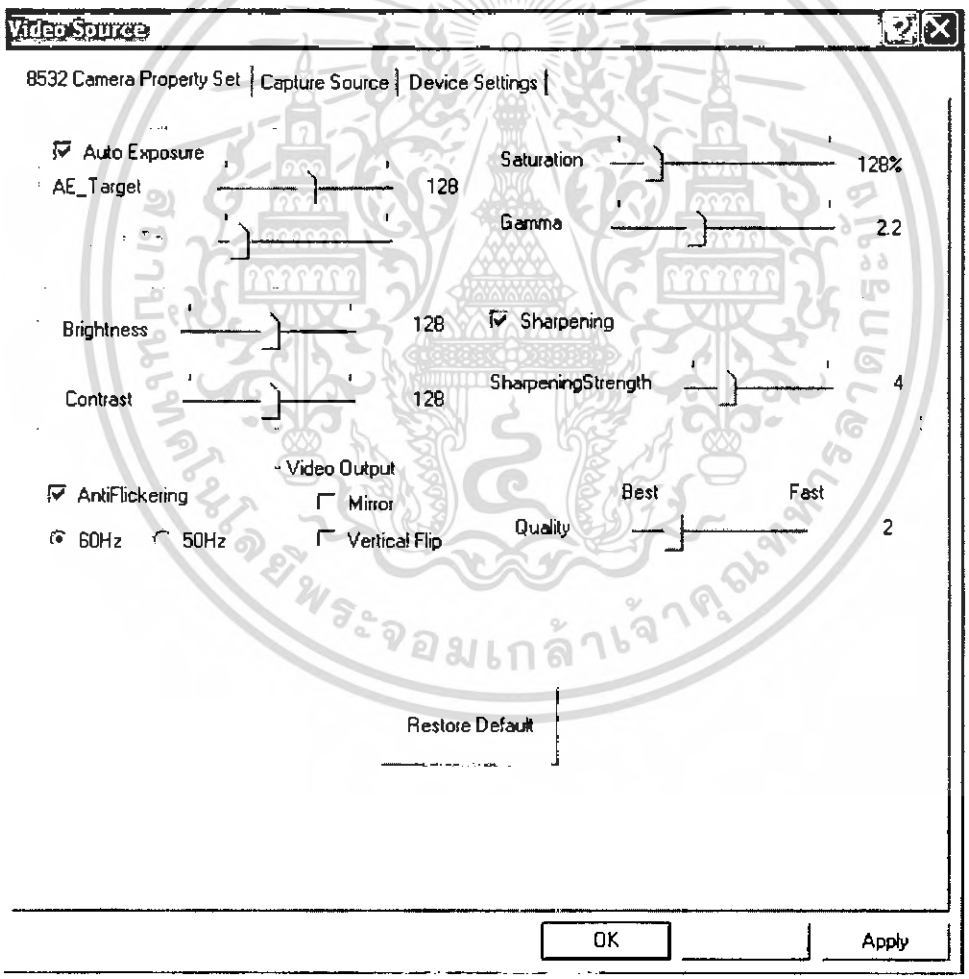
-ช่องสำหรับป้อนข้อมูลเพื่อค้นหาโดยจะทำการค้นหาตามชื่อ ของผู้ใช้

-ส่วนของการแสดงรูป โดยจะมีการแสดงรูปของผู้ใช้ ณ เวลาปัจจุบันเปรียบเทียบกับรูปที่เก็บไว้ในฐานข้อมูลเพื่อตรวจสอบและยืนยันการใช้บัตร

-กลุ่มของ ปุ่มคำสั่งที่จะทำงานเกี่ยวกับรูปภาพโดยประกอบด้วย ปุ่มบันทึกรูปภาพ (Capture) , ปุ่มกำหนดรูปแบบของภาพ (Format) โดยสามารถกำหนดค่าต่างๆได้ดังรูปที่ 4.2 , ปุ่มกำหนดสีและความคมชัดของภาพ (Color) โดยสามารถกำหนดค่าต่างๆได้ดังรูปที่ 4.3



รูปที่ 7.2 หน้าต่างสำหรับปรับรูปแบบของภาพ



รูปที่ 7.3 หน้าต่างสำหรับปรับสีและความคมชัดของภาพ

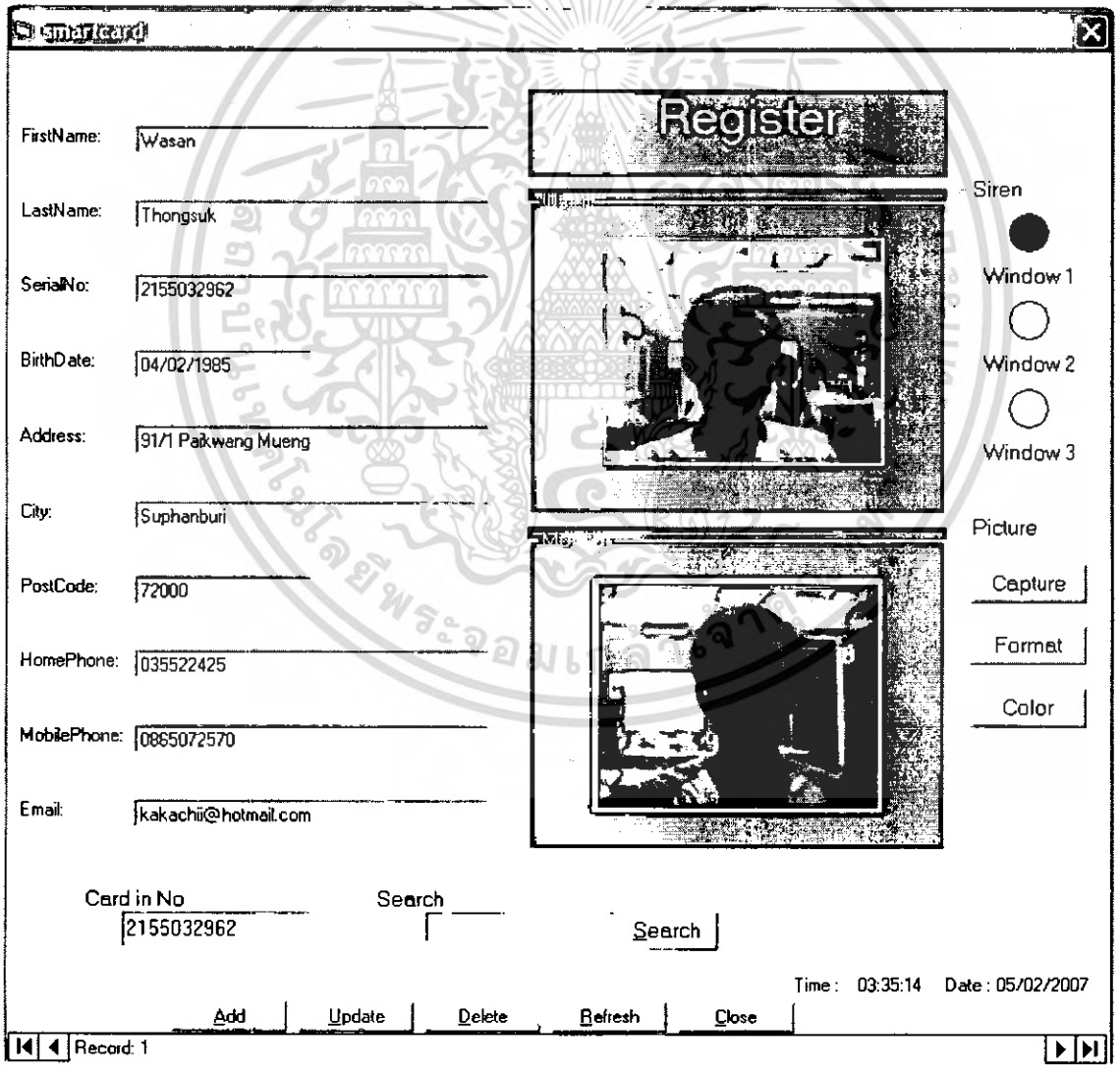
-กลุ่มของสัญญาณแสดงการเตือนเมื่อเซนเซอร์ตามจุดต่างๆทำงานโดยกะพริบเป็นสีแดง

-ส่วนแสดงเวลา และวันที่ปัจจุบัน

7.2.2 โหมดการทำงาน

โหมดลงทะเบียน สามารถแบ่งออกเป็นขั้นตอนดังนี้

1. เสียบบัตรเข้าไปยังเครื่องอ่านบัตร หมายเลขบัตรจะถูกอ่านแล้วส่งไปยังคอมพิวเตอร์
2. กดที่ปุ่ม Add ในช่องที่แสดงข้อมูลหมายเลขบัตรจะมีหมายเลขบัตรที่เสียบเพื่อลงทะเบียนแสดงขึ้นมา
3. ป้อนข้อมูลต่างๆลงในช่องที่ใช้แสดงข้อมูลของผู้ขอใช้บัตรให้ครบทุกช่อง
4. กดปุ่ม Capture เพื่อทำการบันทึกภาพโดยให้ตั้งชื่อภาพให้ตรงกับหมายเลขบัตรที่แสดงอยู่ในตอนแรก
5. กดปุ่ม Update ข้อมูลต่างๆที่กรอกไว้จะถูกบันทึกลงฐานข้อมูลเรียบร้อย



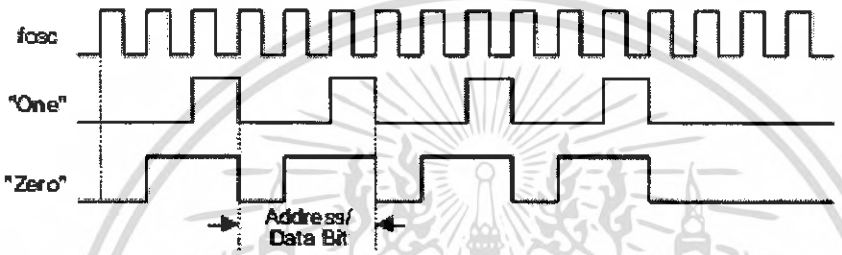
รูปที่ 7.4 หน้าโปรแกรมหลักที่ใช้ลงทะเบียนและแสดงผล

โหมดการผ่านเข้าออก ซึ่งเป็นหน้าที่ของเจ้าหน้าที่รักษาความปลอดภัยที่จะต้องตรวจสอบข้อมูลการผ่านเข้าออกโดยการตรวจสอบหมายเลขบัตรที่ถูกเสียบเข้ามาผ่านทางช่องแสดงหมายเลขบัตร Card in No กับหมายเลขบัตรที่ฐานข้อมูลแสดงไว้ และตรวจสอบใบหน้าบุคคลโดยเปรียบเทียบกับภาพจากกล้องวีดีโอ กับภาพที่เก็บไว้ในฐานข้อมูล

7.3 หลักการออกแบบวงจร RF Module

7.3.1 วงจรเข้ารหัสสัญญาณ (Encoder)

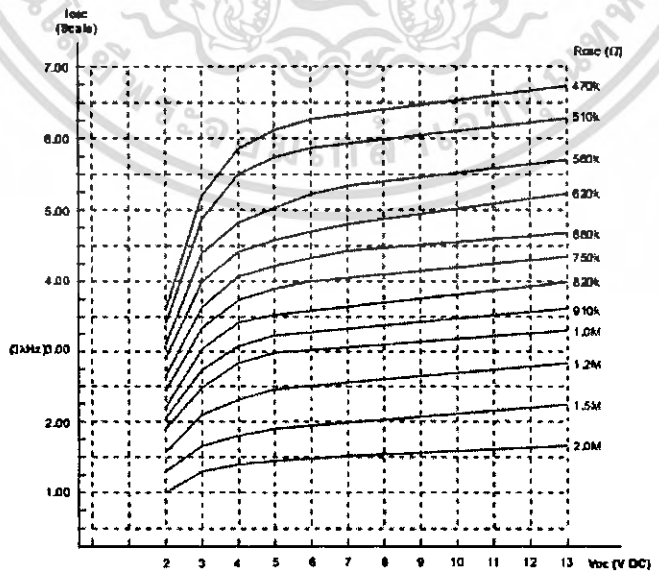
เราใช้ IC เบอร์ HT12E ในการเข้ารหัสข้อมูล 12 bit โดยในการส่งสัญญาณที่ "0" จะส่งสัญญาณ 011 และ สัญญาณที่ "1" จะส่งสัญญาณ 001 ออกไป



Address/Data bit waveform for the HT12E

รูปที่ 7.5 Waveform ของ Address และ Data

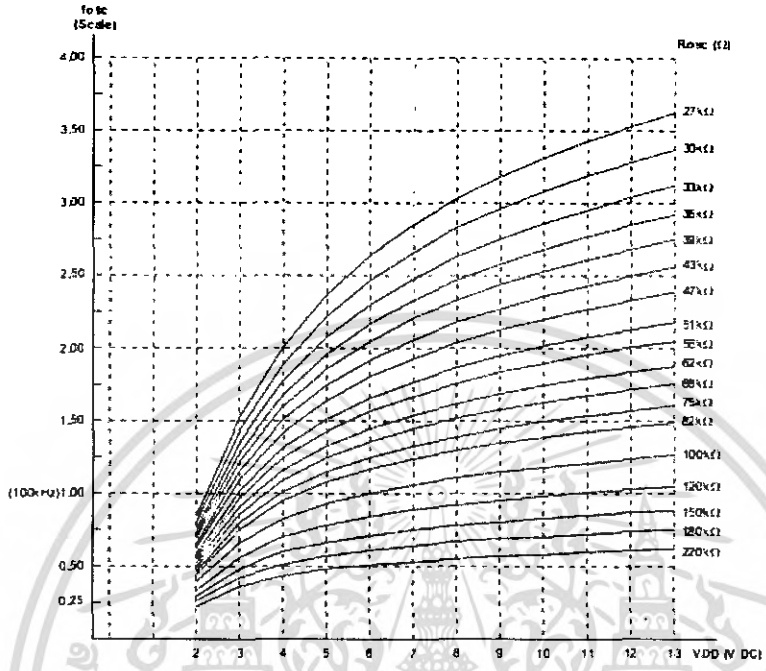
ซึ่งภายใน HT12E มีตัวสร้างสัญญาณนาฬิกาภายใน ความถี่ที่ใช้จะขึ้นอยู่กับ Rosc โดยเราเลือกใช้ความถี่ 2.5 KHz ที่ Vcc = 3 V ซึ่งได้ Rosc = 1M Ω



รูปที่ 7.6 กราฟที่ใช้ในการพิจารณาค่าความต้านทาน

7.3.2 ถอดรหัสสัญญาณ (Decoder)

ส่วน HT12D ความถี่ foscd (decoder) เท่ากับ 50 ของ ความถี่ fosce (encoder) จะได้ foscd = 125 kHz ที่ Vcc = 5 V ได้ Rosc = 68k Ω

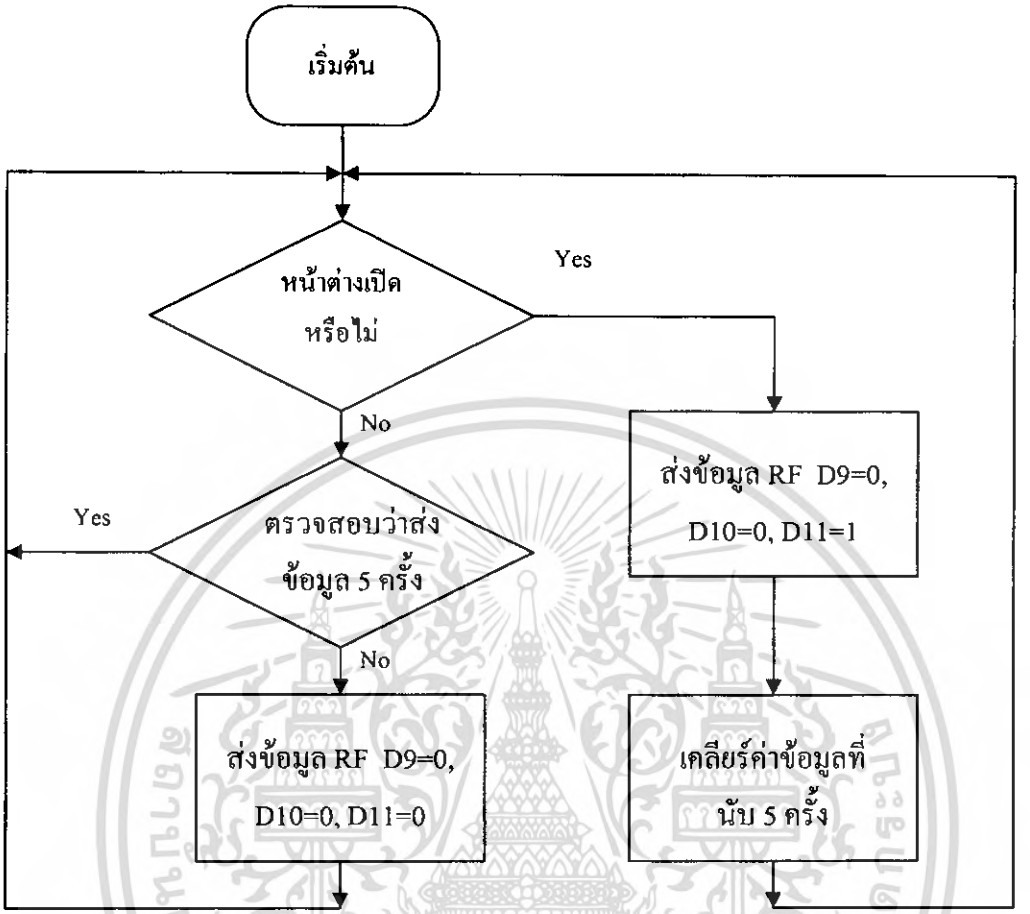


รูปที่ 7.7 กราฟที่ใช้ในการพิจารณาค่าความต้านทาน

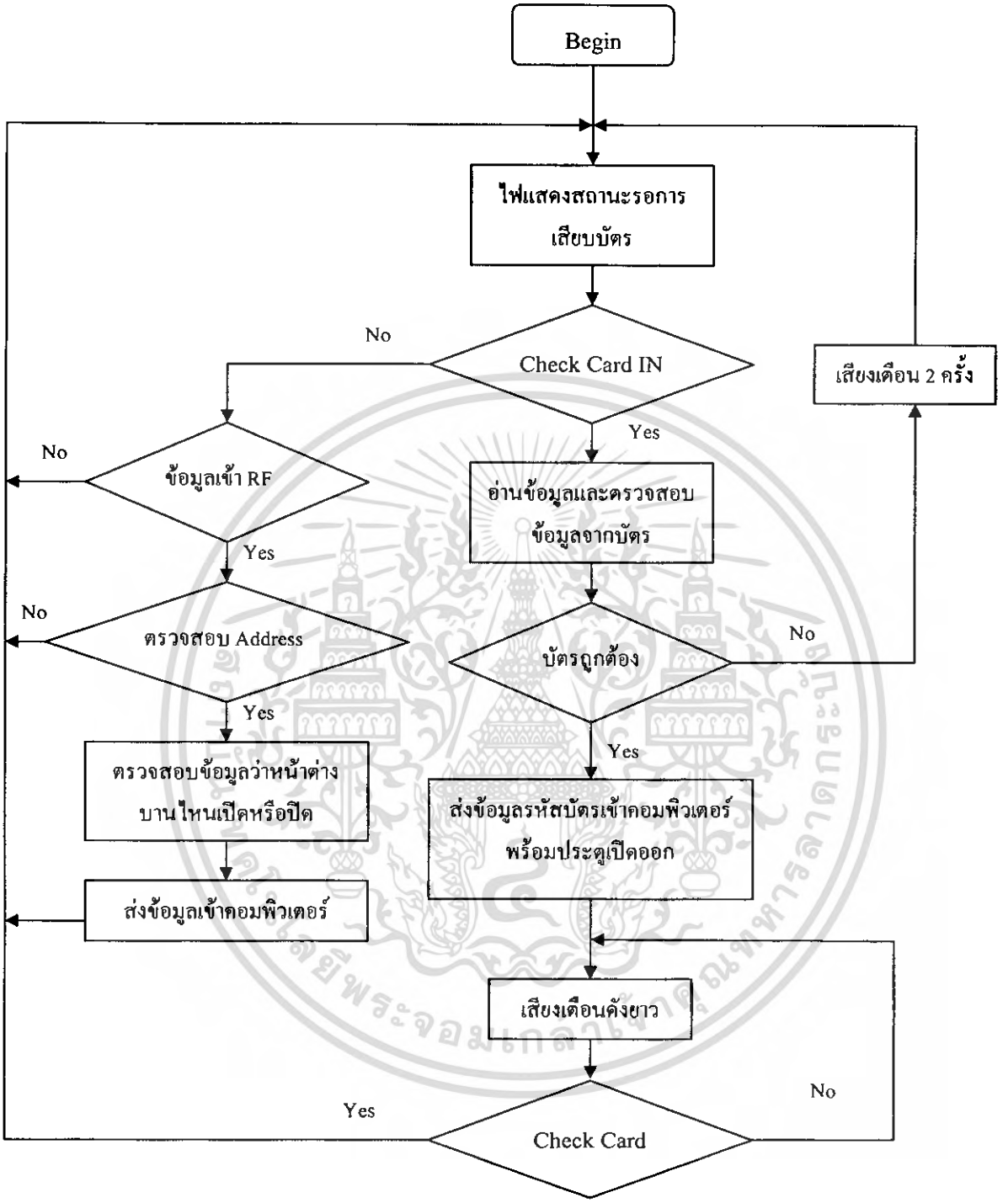
7.4 หลักการทำงานของวงจร RF Module

โปรเจคนี้นำ RF Module มาใช้ในส่วนของเซนเซอร์ที่หน้าต่าง ซึ่งใช้ไมโครคอนโทรลเลอร์ AT89C2051 เป็นตัวควบคุมการส่งข้อมูลและตรวจสอบการเปิดปิดของหน้าต่าง เมื่อหน้าต่างบานที่ 1 ปิดอยู่ เซนเซอร์ที่หน้าต่างจะส่งสัญญาณเข้ามาที่พอร์ต P1.0 ไมโครคอนโทรลเลอร์จะส่งสัญญาณลอจิก 0 ไปที่ Encoder ที่ขา TE จำนวน 5 ครั้ง ครั้งละประมาณ 16 ms แล้วจึงทำการส่งข้อมูล D9=0, D10=0, D11=0 ที่ขา Encoder พร้อมแอดเดสออกไปที่ RF Module ตัวส่ง ซึ่งจะทำการมอดดูเลทสัญญาณ แล้วจึงส่งออกไป เมื่อหน้าต่างบานที่ 1 เปิดจะส่งข้อมูล D9=0, D10=0, D11=1 ส่วนเมื่อหน้าต่างบานที่ 2 ปิดจะส่งข้อมูล D9=1, D10=0, D11=0 และเมื่อหน้าต่างบานที่ 2 เปิดจะส่งข้อมูล D9=1, D10=0, D11=1 ออกไป

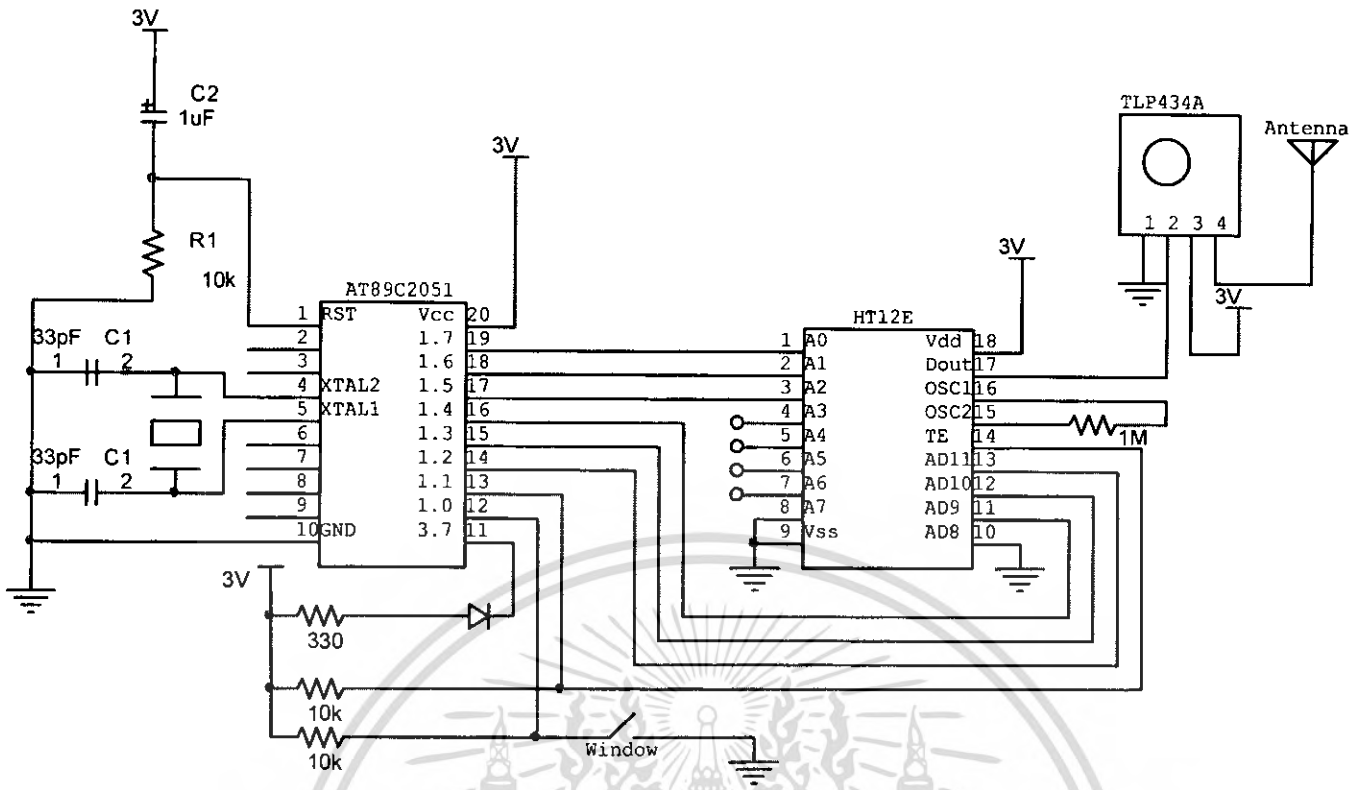
ส่วนที่ภาครับเมื่อ RF Module รับสัญญาณเข้ามาจะทำการส่งข้อมูลไป ที่ขา DIN ของ Decoder ซึ่ง Dccoder จะทำการถอดรหัสสัญญาณว่าแอดเดสตรงกับที่กำหนดไว้หรือไม่ก่อน แล้วจึงส่งข้อมูลเข้าเครื่องอ่านว่าหน้าต่างบานไหนเปิดหรือปิดอยู่ เครื่องอ่านจะส่งข้อมูลเข้าคอมพิวเตอร์ผ่านทาง Serial Port ต่อไป



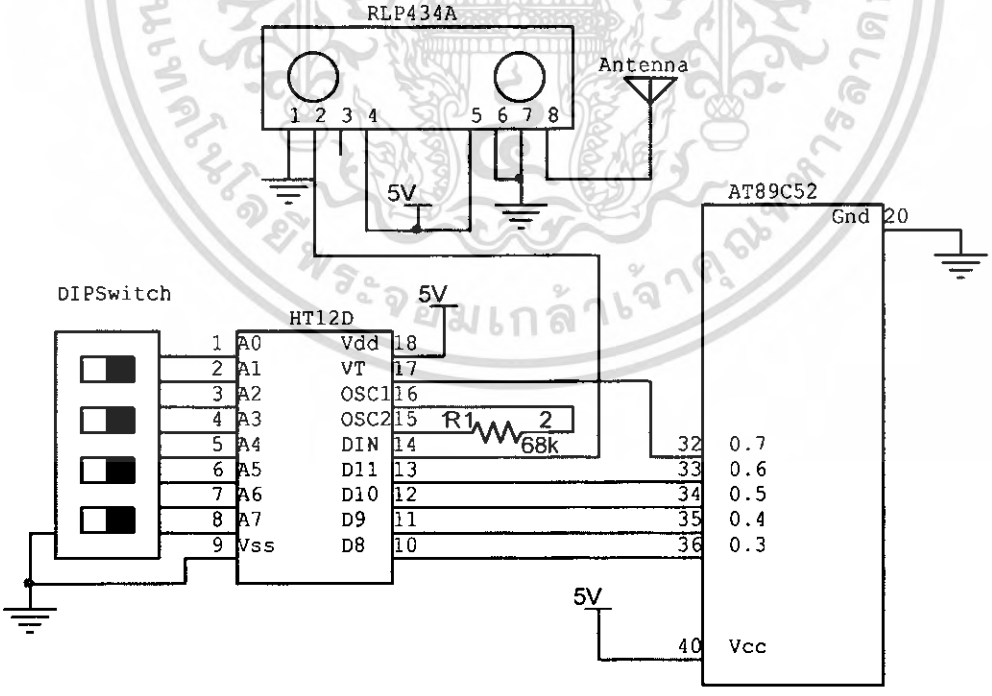
รูปที่ 7.8 โฟลวชาร์ตการทำงานของ RF ตัวส่ง



รูปที่ 7.9 โฟลวชาร์ตการทำงานของเครื่องอ่านบัตรสมาร์ทการ์ด TOT



รูปที่ 7.0 รูปวงจรภาคส่งของ RF Module



รูปที่ 7.1 รูปวงจรภาครับของ RF Module

บทที่ 8 การทดลองและผลการทดลอง

8.1 การทดลองวัดความผิดพลาดการรับส่งข้อมูล

8.1.1 ใช้สายรับสัญญาณยาว 5 ฟุต

8.1.2 ความเร็วในการรับข้อมูล 9600 บิตต่อวินาที ผ่านพอร์ต COM1

8.1.3 ทำการทดลองโดยการรับข้อมูลจากบัตร สมาร์ตการ์ด ตัวอย่าง 7 บัตรจำนวนบัตร 100 ครั้ง

ผลการทดลองโดยการรับข้อมูลจากบัตร

หมายเลขบัตร	หมายเลขประจำตัวบัตร	จำนวนครั้งที่ถูกต้อง	จำนวนครั้งที่ผิดพลาด	เปอร์เซ็นต์ความผิดพลาด
1*	2044003256	99	1	1 %
2	2059041604	100	0	0 %
3	2163048240	100	0	0%
4	2138030228	100	0	0%
5	2155032962	100	0	0%
6	2018008356	100	0	0%
7	2134098322	100	0	0%

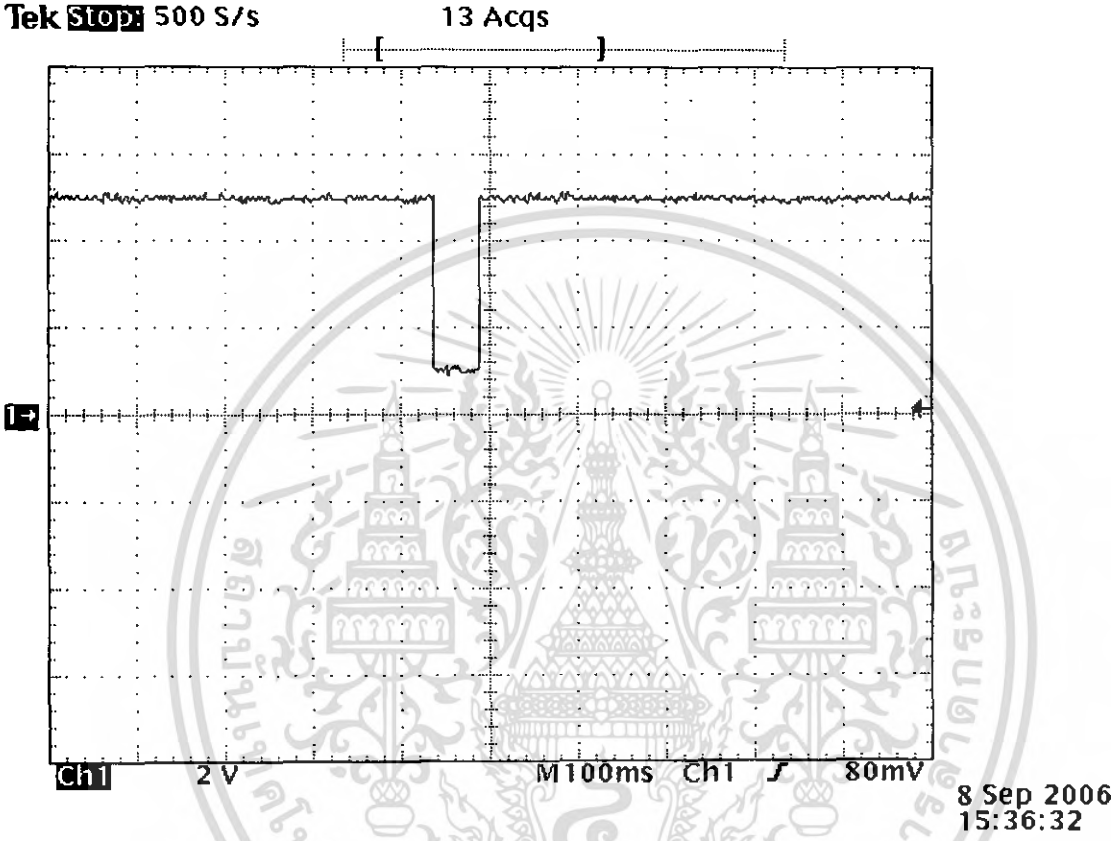
ดารางที่ 8.1 ผลการทดลองความผิดพลาดของเครื่องอ่านบัตร

หมายเหตุ : ความผิดพลาดที่วัดได้เป็น 1* = Not Found Data

8.2 การทดลองวัดรูปสัญญาณ ณ ตำแหน่งขาต่างๆในสมาร์ตการ์ด

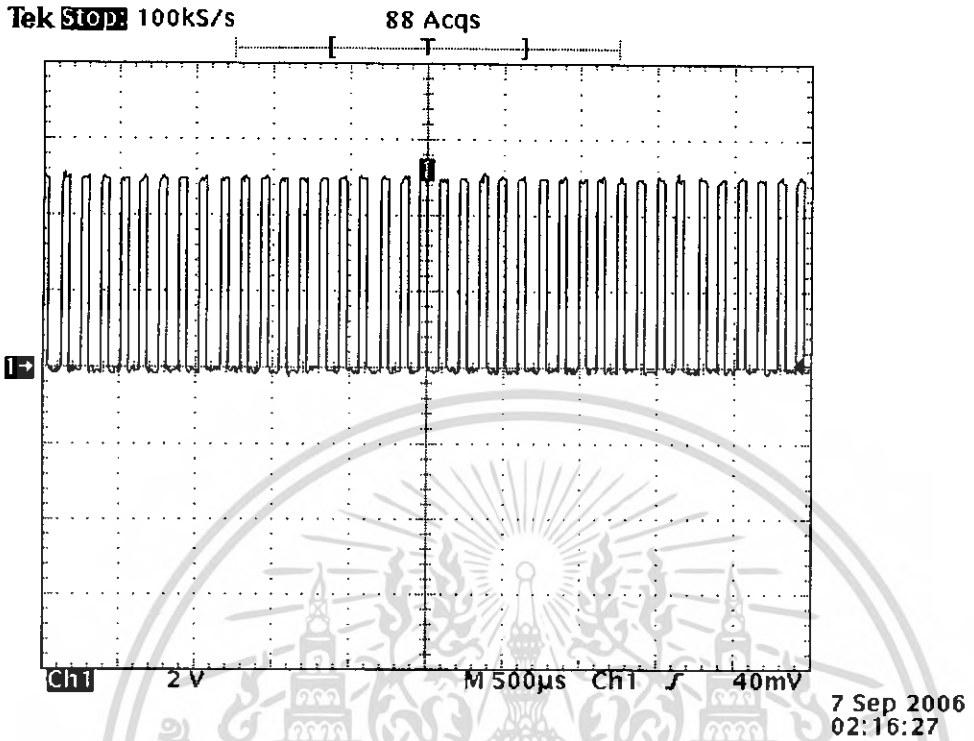
- ทำการวัดสัญญาณที่ขาต่างๆของสมาร์ตการ์ด ซึ่งได้แก่ Vcc , RST , CLK , I/O

8.2.1 สัญญาณที่ขา Vcc



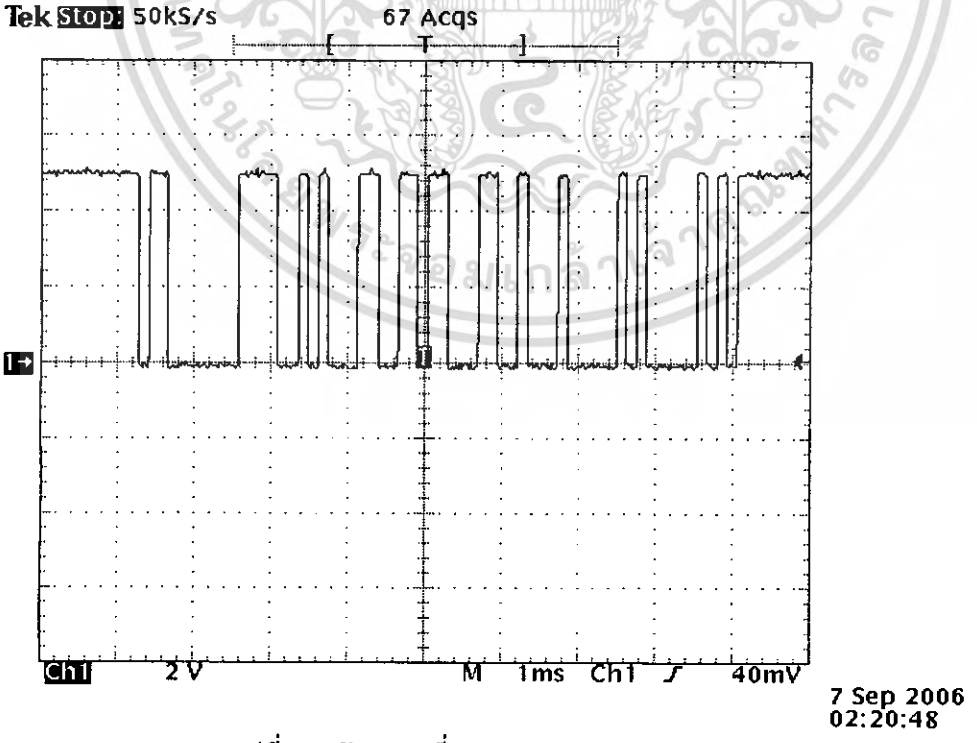
รูปที่ 8.1 สัญญาณที่ขา Vcc

8.2.2 สัญญาณที่ขา CLK

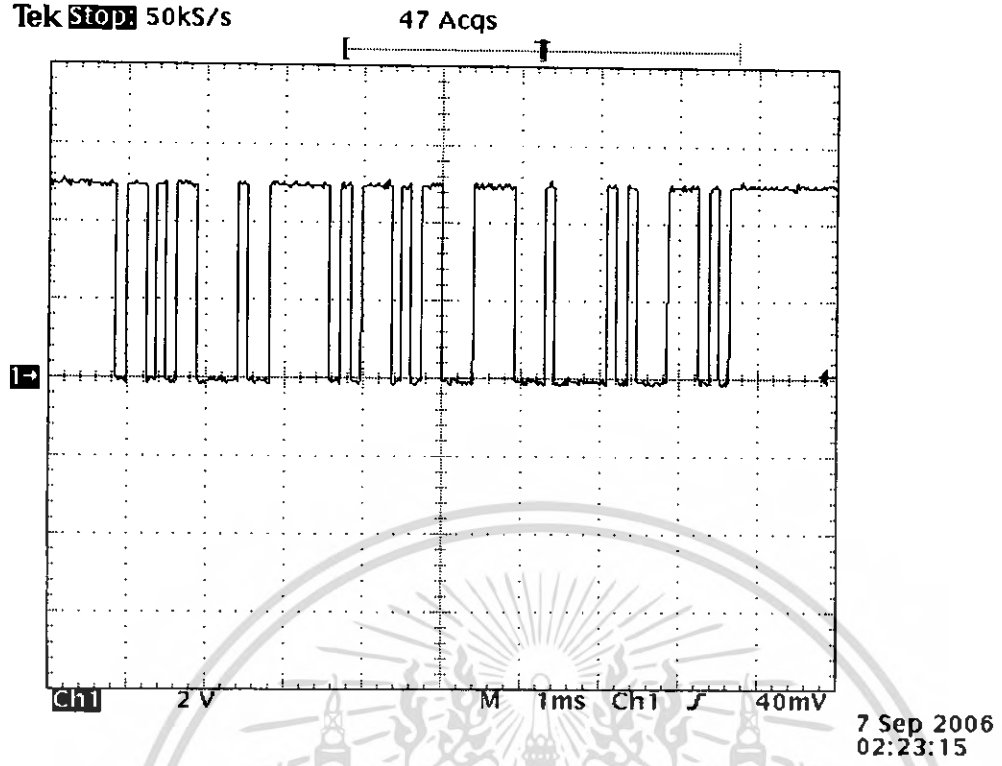


รูปที่ 8.2 สัญญาณที่ขา CLK

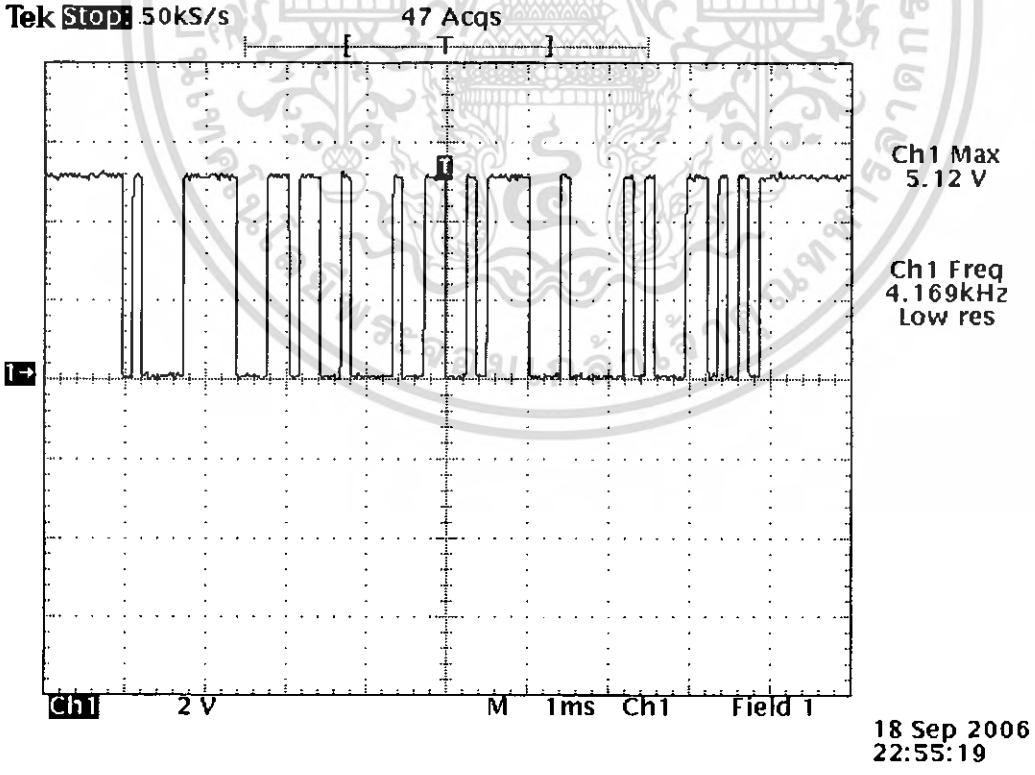
8.2.3 สัญญาณที่ขา I/O



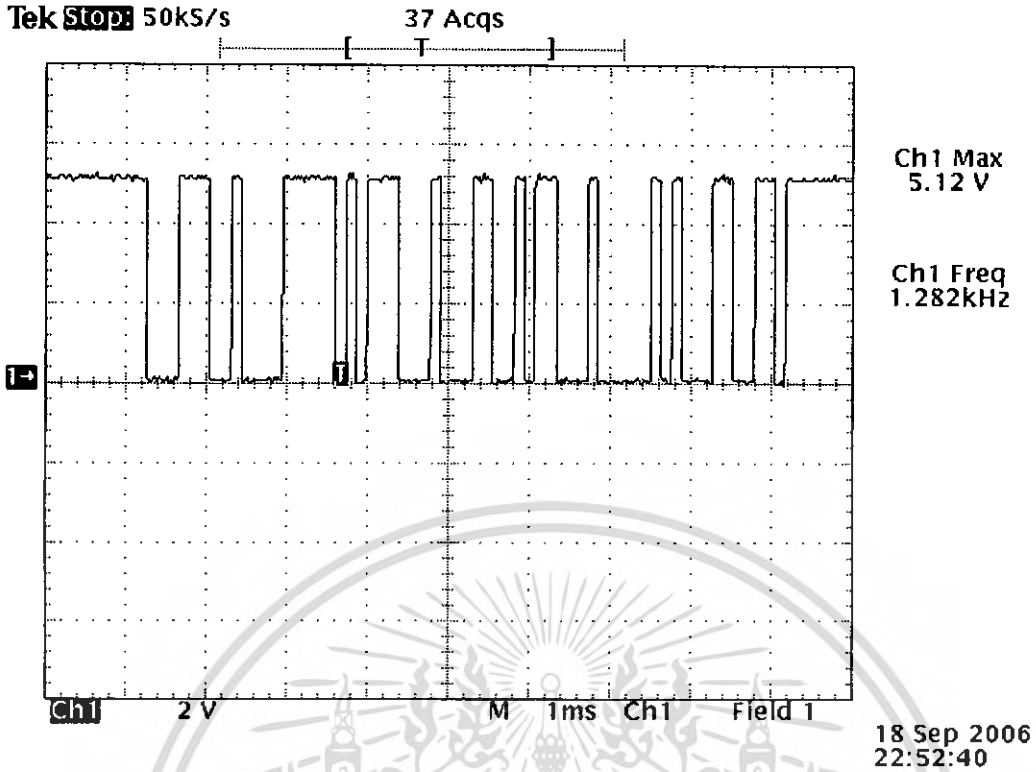
รูปที่ 8.3 สัญญาณที่ขา I/O ของบัตรหมายเลข 1



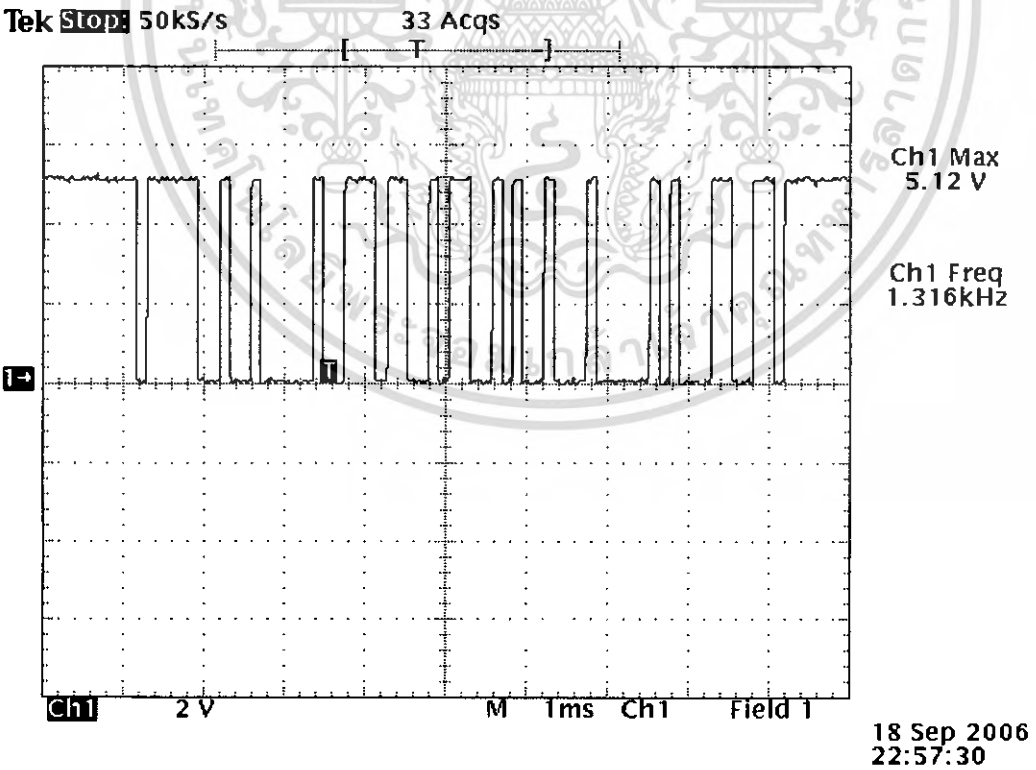
รูปที่ 8.4 สัญญาณที่ขา I/O ของบัตรหมายเลข 2



รูปที่ 8.5 สัญญาณที่ขา I/O ของบัตรหมายเลข 3



รูปที่ 8.6 สัญญาณที่ขา I/O ของบัตรหมายเลข 4



รูปที่ 8.7 สัญญาณที่ขา I/O ของบัตรหมายเลข 5

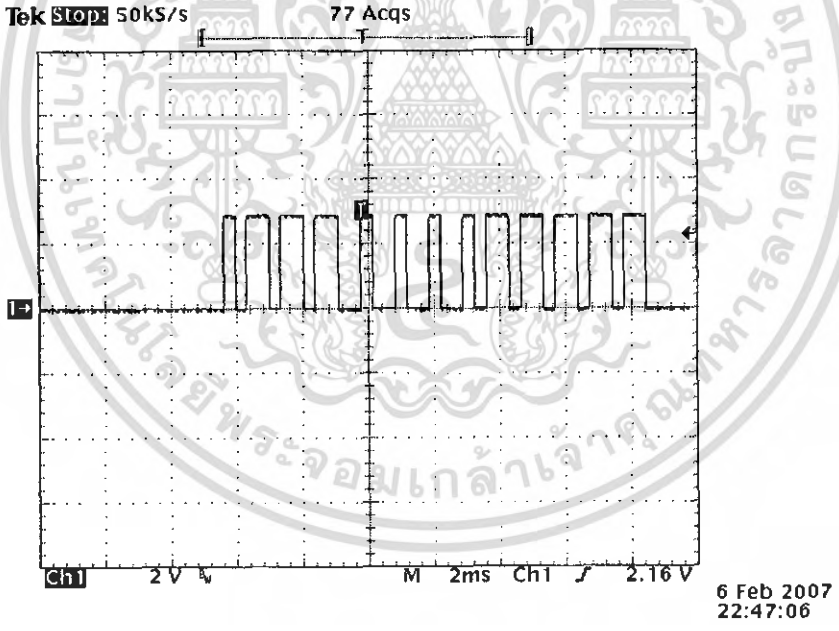
8.3 ผลการทดลองการรับส่งข้อมูลของวงจร RF

8.3.1 ทำการส่งข้อมูลจากตำแหน่งต่างๆ ที่ระยะ 1 , 3 และ 5 เมตรจุดละ 10 ครั้ง

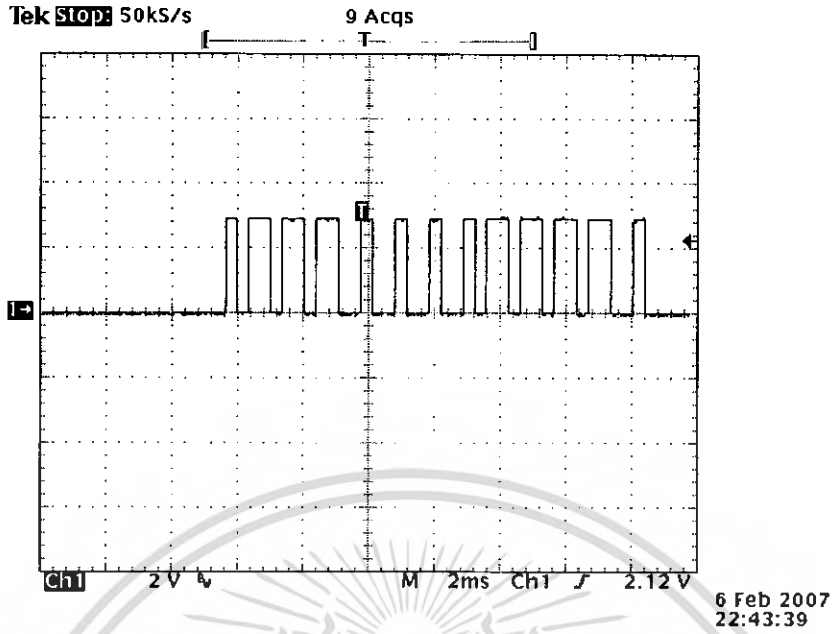
8.3.2 แสดงผลโดยใช้สัญญาณไฟที่สร้างขึ้นจากโปรแกรมประยุกต์ และสัญญาณเสียงจากBuzzer

ตำแหน่ง	สัญญาณไฟ	สัญญาณเสียง
หน้าต่าง 1	30	30
หน้าต่าง 2	30	30

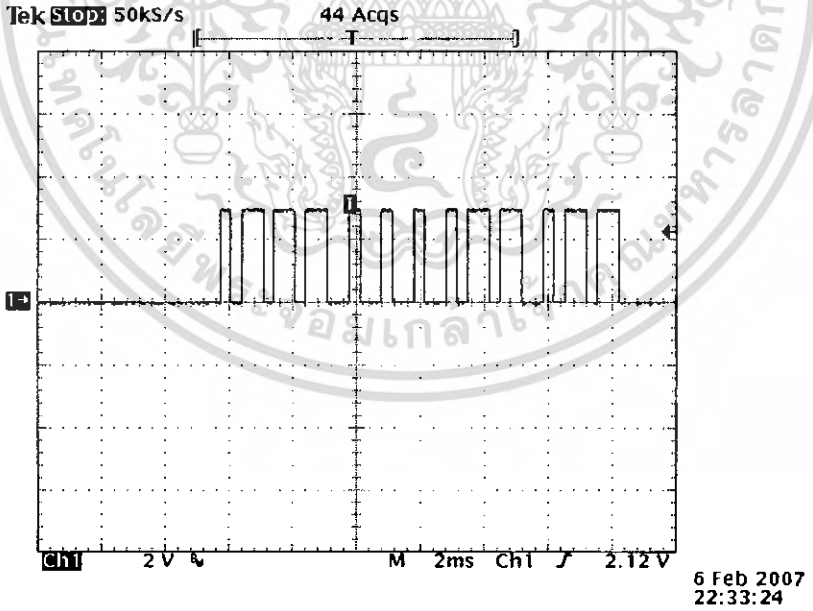
ตารางที่ 8.2 แสดงความถูกต้องในการรับส่งข้อมูลของวงจร RF



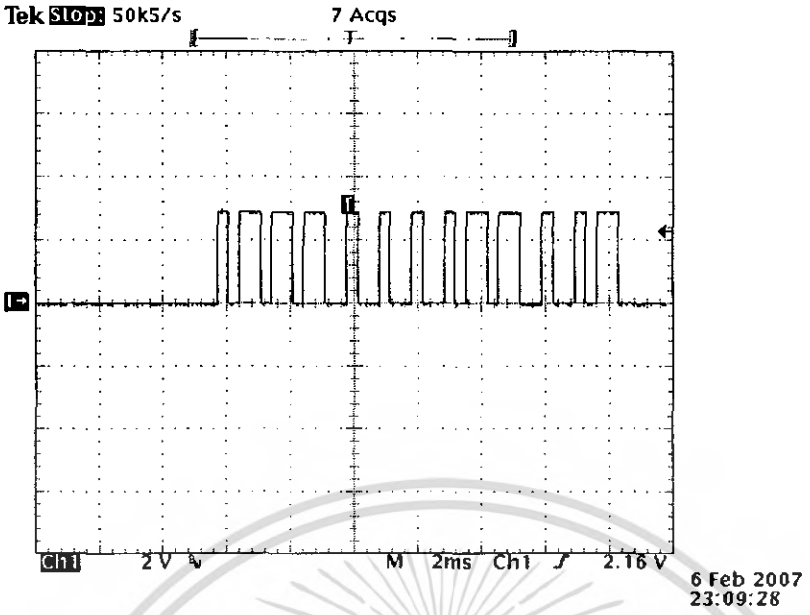
รูปที่ 8.8 รูปสัญญาณที่ Encoder ในขณะที่หน้าต่างบานที่ 1 ถูกปิด



รูปที่ 8.9 รูปสัญญาณที่ Encoder ในขณะที่หน้าต่างบานที่ 1 ถูกเปิด



รูปที่ 8.10 รูปสัญญาณที่ Encoderในขณะที่หน้าต่างบานที่ 2 ถูกปิด



รูปที่ 8.11 รูปสัญญาณที่ Encoder ในขณะที่หน้าด่างบานที่ 2 ถูกเปิด

บทที่ 9 สรุปผลการทดลอง

จากผลการทดลองในบทที่ 5 จะเห็นว่า เปอร์เซ็นต์ผิดพลาดของข้อมูลที่เครื่องอ่านบัตรสมาร์ตการ์ดอ่านได้และส่งให้คอมพิวเตอร์นั้น มีความผิดพลาดน้อยมากและสำหรับส่วนที่ผิดพลาดนี้เกิดขึ้นเมื่อทำการเสียบบัตรแล้วเครื่องอ่านสามารถสั่งให้ประตูเปิดได้แต่ข้อมูลไม่แสดงที่หน้าของโปรแกรมบนคอมพิวเตอร์ โดยโปรแกรมได้แสดงข้อความเตือนว่าไม่พบข้อมูล แสดงว่าบัตรที่เสียบเข้ามามีความถูกต้องแล้วแต่เกิดความผิดพลาดระหว่างการส่งข้อมูลผ่าน Serial Port ไปยังคอมพิวเตอร์ การแก้ปัญหาในเบื้องต้นคือให้ทำการเสียบบัตรใหม่อีกครั้งเพื่อตรวจสอบความถูกต้อง

อย่างไรก็ตามโครงงานนี้จะเกิดประโยชน์สูงสุดและมีประสิทธิภาพมากน้อยเพียงใดขึ้นอยู่กับการนำไปใช้งานโดยจะต้องมีการปรับปรุง วงจรของเครื่องอ่านบัตรสมาร์ทการ์ด เซนเซอร์ที่ใช้ตรวจสอบตามจุดต่างๆ และที่สำคัญคือโปรแกรมประยุกต์ที่สร้างขึ้นให้มีความเหมาะสมกับลักษณะการใช้งานให้มากที่สุด

บรรณานุกรม

1. ธาริน สิทธิธรรมชารี," Visual Basic Version 6.0", Success Media Co.,Ltd , ครั้งที่ 1 , กันยายน 2548.

2. กิตติ ภักดีวัฒนะกุล , จำลอง ครูอุตสาหะ , " Visual Basic 6 ฉบับฐานข้อมูล " , KTP Com & Consult , ครั้งที่ 2 , มีนาคม 2543.

3. ศุภชัย สมพานิช , "Advance Data Base Programming ด้วย VB6 + VB.NET" ,DEV Book , ครั้งที่ 1 , เมษายน 2548.

4. อภิชาติ ภู่พลับ , "เริ่มต้นเขียนโปรแกรมติดต่อและควบคุมฮาร์ดแวร์ด้วย Visual Basic " DEV Book , ครั้งที่ 1, กันยายน 2546.

ภาคผนวก

```
#include <AT89X55.H>
#include <stdio.h>
sbit chkset=P0^0;
sbit chkclr=P0^1;
sbit ledset=P1^0;
sbit ledcd=P1^1;
sbit lednocd=P1^2;
sbit ledclr=P1^3;
sbit open=P1^6;
sbit sound=P1^7;
sbit chkcard=P2^0;
sbit vcc=P2^1;
sbit rst=P2^2;
sbit clk=P2^3;
sbit IO=P2^4;
bit error=0,tr=0;
unsigned char KPSMC[7][5],SMC[48],door,num=0,same=0;
// ****************************** Delay ************************************
void dmsec(unsigned int tick)
{       unsigned char j;
        unsigned int i;
        for(i=0;i<tick;i++)
            for(j=0;j<100;j++);
}
void dusec()
{
 unsigned char i=20;
 while(i>0)
        {
            i--;
```

```
        }
}
//*************************** Reset *********************************
void reset()
{
 rst=1;
 dusec();
 clk=1;
 dusec();
 clk=0;
 dusec();
 rst=0;
 dusec();
}
// ************************** Read I/O ***********************************
void read()
{
        unsigned char c,x,a;
        unsigned int i;
        c=0;
        x=0;
        a=0;
        for(i=0;i<384;i++)
        {
            if(chkcard==0)
            {
                IO=1;
                a>>=1;
            if(IO)
             {a|=0x80;}
            else
             {a&=0x7f;}
```

```
clk=1;
dusec();
        clk=0;
dusec();
if(++c==8)
{
                SMC[x++]=a;
                c=0;
                a=0;
        }
}
else
{
i=386;
        door=2;
}
}
if(door!=2)
{
c=0;
x=0;
for(i=0;i<5;i++)
{
        if(SMC[i+3]==0xff)
        {
                c++;
        }
        if(SMC[i+3]==0x00)
        {
                x++;
        }
}
```

```
                if((c==5)||(x==5))
                {
                        door=2;
                        c=0;
                        x=0;
                }
        }
}
// ********************************* Keep data ***********************************
void kpdata()
{
        unsigned char i,j,right=0;
        for(i=0;i<num;i++)
        {
                for(j=0;j<5;j++)
                {
                        if(KPSMC[i][j]==SMC[j+3])
                        {
                                right++;
                        }
                }
                if(right==5)
                {
                        i=8;
                }
                else
                {
                        right=0;
                }
        }
        if(right!=5)
        {
```

```
                tr=1;
                for(i=0;i<5;i++)
                {
                        KPSMC[num][i]=SMC[i+3];
                }
                num++;
        }
}
// ********************************* Check data ***********************************
void chkSMC()
{
        unsigned char i,j,true=0;
        if(door!=2)
        {
                if(num>0)
                {
                        for(i=0;i<num;i++)
                        {
                                for(j=0;j<5;j++)
                                {
                                        if(KPSMC[i][j]==SMC[j+3])
                                        {
                                                true++;
                                        }
                                }
                                if(true==5)
                                {
                                        same=i;
                                        door=1;
                                        i=8;
                                }
                                else
```

```
                        {
                                true=0;
                                door=2;
                        }
                }
        }
        else
        {
                door=2;
        }
    }
}
// ********************************* Clear Data ************************************
void clrSMC()
{
 unsigned char i;
 for(i=0;i<48;i++)
  {
   SMC[i]=0x00;
  }
}
void clrKPSMC()
{
        unsigned char i,j;
        for(i=0;i<7;i++)
        {
            for(j=0;j<5;j++)
            {
                KPSMC[i][j]=0x00;
            }
        }
        num=0;
```

```
}
// ***************************** Set Password ************************************
void setpsd()
{
        unsigned char j=0,k=0;
        while(j<30)
        {
            j++;
            dmsec(500);
            if(chkclr==0)
            {
                dmsec(4000);
                if(chkclr==0)
                {
                    clrKPSMC();
                    j=31;
                    ledclr=0;
                    dmsec(2000);
                    ledclr=1;
                }
                else
                {
                    j=30;
                }
            }
            if(chkcard==0)
            {
                vcc=0;
                dusec();
                rcset();
                read();
                dusec();
```

```
                vcc=1;
                if(door==2)
                {
                        j=30;
                }
                else
                {
                        j=31;
                        kpdata();
                }
        }
}
if(j==31)
{
        sound=0;
        dmsec(1000);
        sound=1;
}
if(j==30)
{
        error=1;
        dmsec(500);
        sound=0;
        ledset=0;
        dmsec(500);
        sound=1;
        ledset=1;
        dmsec(500);
        sound=0;
        ledset=0;
        dmsec(500);
        sound=1;
```

```
                ledset=1;
            }
    }
// ****************************** Serial ******************************************
void init_serial()
{
 TR1=0;
 TMOD=0x20;
 SCON=0x50;
 TH1=0xfd;
 RI=0;
 TI=0;
 EA=1;
 ES=1;
 TR1=1;
}
/*void send(unsigned char datsend)
{
 TI=0;
 SBUF=datsend;
 while(~TI);
 TI=0;
}*/
// ********************************** Main ****************************************
void main()
{
  unsigned char q,j;
  num=0;
  init_serial();
  P0=0xff;
  P1=0xff;
  chkset=1;
```

```
chkclr=1;
chkcard=1;
vcc=1;
clk=0;
rst=0;
clrSMC();
while(1)
        {
            if(chkset==0)
            {
                dmsec(4000);
                if(chkset==0)
                {
                    ledset=0;
                    setpsd();
                    ledset=1;
                    TI=1;
                    if(tr==1)
                    {
                        tr=0;
                        printf(" >>> Data Keep to Memory and Send to
Computer");
                        printf("\n");
                        for(q=0;q<5;q++)
                        {
                            printf("%b02X",KPSMC[num-1][q]);
                        }
                        printf("\n");
                    }
                    if(error==1)
                    {
                        error=0;
```

```
                printf(">>> Error");
                printf("\n");
        }
        printf(">>> Data in Memory");
        printf("\n");
        for(q=0;q<7;q++)
        {
                for(j=0;j<5;j++)
                {
                        printf("%b02X",KPSMC[q][j]);
        }
        printf("\n");
        }
        TI=0;
        while(chkcard==0)
        {
                dmsec(500);
        }
    }
}
else
/************************** Read Card ************************/
{
    door=0;
    if(chkcard==0)
    {
        ledcd=0;
        dmsec(500);
        if(chkcard==0)
        {
                vcc=0;
                dusec();
```

```
                reset();                                    //      reset
                read();                                     //      read
data
                    dusec();
                vcc=1;
                    chkSMC();                                       //
        check data
                    if(door==1)
                    {
                        TI=1;
                        printf(" >>> Right Password and Data Sent to
Computer");
                        printf("\n");
                        for(j=0;j<5;j++)
                        {
                            printf("%b02X",KPSMC[same][j]);
                        }
                        printf("\n");
                        printf(" >>> Data in Memory");
                        printf("\n");
                        for(q=0;q<7;q++)
                        {
                            for(j=0;j<5;j++)
                            {
                                printf("%b02X",KPSMC[q][j]);

                            }
                            printf("\n");
                        }
                        TI=0;
```

```
                dusec();
                open=0;
                sound=0;
                dmsec(5000);
                open=1;
                sound=1;
        }
        if(door==2)
        {
                TI=1;
                printf(" >>> Wrong Password");
                printf("\n");
                for(j=0;j<48;j++)
                {
                        printf("%b02X",SMC[j]);
                }
                printf("\n");
                printf(" >>> Data in Memory");
                printf("\n");
                for(q=0;q<7;q++)

                {
                        for(j=0;j<5;j++)
                        {
                                printf("%b02X",KPSMC[q][j]);

                        }
                        printf("\n");
                }
                TI=0;
                sound=0;
                dmsec(500);
```

```
                    sound=1;
                    dmsec(500);
                    sound=0;
                    dmsec(500);
                    sound=1;
                }
                while(chkcard==0)
                {
                    dmsec(500);
                }
            }
        }
        else
        {
            clrSMC();
            same=0;
            ledcd=1;
        lednocd=0;
            dmsec(10);
        lednocd=1;
        }
    }
}
}
```

Source code

```
Option Explicit
Private data As Variant
Private Sub Command2_Click() 'SAVE IMAGE
 Dim filename As String
  If mCmnDlg.VBGetSaveFileName(filename, _
              filter:="Bitmap files (*.bmp)|*.bmp", _
              InitDir:=App.Path, _
              DlgTitle:="Save Frame As Bitmap File", _
              DefaultExt:="bmp", _
              Owner:=Me.hWnd) _
                    Then
    On Error Resume Next
    Call ezVidCap2.SaveDIB(filename)
    If Err Then
      MsgBox Err.Description, vbInformation, App.Title
    End If
  End If
End Sub

Private Sub Command3_Click()
ezVidCap2.ShowDlgVideoFormat
End Sub

Private Sub Command4_Click() 'SEARCH
Dim filename As String
On Error Resume Next
If Text1.Text = "" Then
  MsgBox "Please insert First Name", vbOKOnly + vbCritical, "EROR"
  Text1.SetFocus
Else
  With datPrimaryRS.Recordset
```

```
        .MovcFirst
        .Find "FirstName Like '" & Text1.Text & "'"
      filename = "D:\xxx\" + txtFields(2).Text + ".bmp"
      Picture1.Picture = LoadPicture(filename)
     If .EOF Then
        MsgBox "Not Found Data ", vbOKOnly + vbInformation, " Search Result"
           .MoveFirst
        End If

        End With
        End If
        Text1.Text = ""
End Sub

Private Sub Command5_Click()
ezVidCap2.ShowDlgVideoSource
End Sub

Private Sub datPrimaryRS_WillMove(ByVal adReason As ADODB.EventReasonEnum, adStatus
As ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
Dim fso As New FileSystemObject
Dim f As File
Dim ts As TextStream
 Set f = fso.GetFile("d:\Report.txt")
 Set ts = f.OpenAsTextStream(ForAppending)
   If Text2.Text <> "" Then
 ts.WriteLine "First Name :  " & txtFields(0).Text
 ts.WriteLine "Last Name :  " & txtFields(1).Text
 ts.WriteLine "Card Number :  " & txtFields(2).Text
 ts.WriteLine "Time :  " & Label3.Caption
 ts.WriteLine "Date :  " & Label4.Caption
 ts.WriteLine ""
```

```
 ts.Close
   End If
End Sub

Private Sub Form_Load()
MSComm1.CommPort = 1
MSComm1.PortOpen = True
MSComm1.Settings = "9600,n,8,1"
   Timer2.Interval = 300
   Timer2.Enabled = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
  Screen.MousePointer = vbDefault
End Sub
Private Sub datPrimaryRS_Error(ByVal ErrorNumber As Long, Description As String, ByVal
Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long,
fCancelDisplay As Boolean)
  MsgBox "Data error event hit err:" & Description
End Sub

Private Sub datPrimaryRS_MoveComplete(ByVal adReason As ADODB.EventReasonEnum,
ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As
ADODB.Rccordset)
  datPrimaryRS.Caption = "Record: " & CStr(datPrimaryRS.Recordset.AbsolutePosition)
End Sub

Private Sub datPrimaryRS_WillChangeRecord(ByVal adRcason As ADODB.EventReasonEnum,
ByVal cRecords As Long, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As
ADODB.Recordset)
  Dim bCancel As Boolean
  Select Case adReason
```

```
  Case adRsnAddNew
  Case adRsnClose
  Case adRsnDelete
  Case adRsnFirstChange
  Case adRsnMove
  Case adRsnRequery
  Case adRsnResynch
  Case adRsnUndoAddNew
  Case adRsnUndoDelete
  Case adRsnUndoUpdate
  Case adRsnUpdate
  End Select
  If bCancel Then adStatus = adStatusCancel
End Sub

Private Sub cmdAdd_Click()
  On Error GoTo AddErr
  datPrimaryRS.Recordset.AddNew
    txtFields(2).Text = Text2.Text
  Exit Sub
AddErr:
    MsgBox Err.Description
End Sub

Private Sub cmdDelete_Click()
  On Error GoTo DeleteErr
  With datPrimaryRS.Recordset
    .Delete
    .MoveNext
    If .EOF Then .MoveLast
  End With
  Exit Sub
```

```
DeleteErr:
  MsgBox Err.Description
End Sub

Private Sub cmdRefresh_Click()
  On Error GoTo RefreshErr
  datPrimaryRS.Refresh
  Exit Sub
RefreshErr:
  MsgBox Err.Description
End Sub

Private Sub cmdUpdate_Click()
  On Error GoTo UpdateErr
  datPrimaryRS.Recordset.UpdateBatch adAffectAll
  Exit Sub
UpdateErr:
  MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
    Form1.Show
  Unload Me
End Sub

Private Sub Text2_Change() 'ACCESS THE DOOR
Dim filename As String
On Error Resume Next
'If Text2.MaxLength = 10 Then
    With datPrimaryRS.Recordset
        .MoveFirst
        .Find "SerialNo = '" & Text2.Text & "'"
```

```
    filename = "D:\xxx\" + txtFields(2).Text + ".bmp"
    Picture1.Picture = LoadPicture(filename)
   If .EOF Then
   Text2.Text = ""
      MsgBox "Not Found Data ", vbOKOnly + vbInformation, " Search Result"
         .MoveFirst
      End If
      End With
End Sub

Private Sub Timer1_Timer()
If Text3.Text = "1" Then
Shape1.BackColor = vbRed
   ElseIf Text3.Text = "2" Then
   Shape2.BackColor = vbRed

      ElseIf Text3.Text = "3" Then
      Shape3.BackColor = vbRed
      End If
End Sub

Private Sub Timer2_Timer()
On Error Resume Next
Label3.Caption = Format(Now, "hh:mm:ss")
Label4.Caption = Format(Date, "dd/mm/yyyy")
   MSComm1.DTREnable = True
   MSComm1.EOFEnable = False
   MSComm1.InputLen = 0
   data = MSComm1.Input
   Text2.Text = StrConv((data), vbUnicode)
  Shape1.BackColor = vbWhite
  Shape2.BackColor = vbWhite
```

```
  Shape3.BackColor = vbWhite
End Sub
```

```
Option Private Module
Option Explicit
Private Const MAX_PATH = 1024
Private Const MAX_FILE = 512
Private Type SHITEMID
   cb As Long
   abID As Byte
End Type
Private Type ITEMIDLIST
   mkid As SHITEMID
End Type

Public Enum SPECIAL_FOLDERS
   vbCSIDL_DESKTOP = &H0&
   vbCSIDL_PROGRAMS = &H2&
   vbCSIDL_CONTROLS = &H3&
   vbCSIDL_PRINTERS = &H4&
   vbCSIDL_PERSONAL = &H5&
   vbCSIDL_FAVORITES = &H6&
   vbCSIDL_STARTUP = &H7&
   vbCSIDL_RECENT = &H8&
   vbCSIDL_SENDTO = &H9&
   vbCSIDL_BITBUCKET = &HA&
   vbCSIDL_STARTMENU = &HB&
   vbCSIDL_DESKTOPDIRECTORY = &H10&
   vbCSIDL_DRIVES = &H11&
   vbCSIDL_NETWORK = &H12&
   vbCSIDL_NETHOOD = &H13&
   vbCSIDL_FONTS = &H14&
   vbCSIDL_TEMPLATES = &H15&
```

```
End Enum

Private Declare Function SHGetSpecialFolderLocation Lib "shell32.dll" _
                        (ByVal hwndOwner As Long, _
                        ByVal nFolder As SPECIAL_FOLDERS, _
                        pidl As ITEMIDLIST) As Long
Public Const NOERROR As Long = 0
Private Type OPENFILENAME
   lStructSize As Long
   hwndOwner As Long
   hInstance As Long
   lpstrFilter As String
   lpstrCustomFilter As String
   nMaxCustFilter As Long
   nFilterIndex As Long
   lpstrFile As String
   nMaxFile As Long
   lpstrFileTitle As String
   nMaxFileTitle As Long
   lpstrInitialDir As String
   lpstrTitle As String
   Flags As Long
   nFileOffset As Integer
   nFileExtension As Integer
   lpstrDefExt As String
   lCustData As Long
   lpfnHook As Long
   lpTemplateName As Long
End Type

Private Declare Function GetOpenFileName Lib "COMDLG32" _
   Alias "GetOpenFileNameA" (filestruct As OPENFILENAME) As Long
```

```
Private Declare Function GetSaveFileName Lib "COMDLG32" _
   Alias "GetSaveFileNameA" (filestruct As OPENFILENAME) As Long
Private Declare Function GetFileTitle Lib "COMDLG32" _
   Alias "GetFileTitleA" (ByVal szFile As String, _
   ByVal szTitle As String, ByVal cbBuf As Integer) As Integer

Private Declare Function GetOpenFileNamePreview Lib "MSVFW32" _
   Alias "GetOpenFileNamePreviewA" (filestruct As OPENFILENAME) As Long
Private Declare Function GetSaveFileNamePreview Lib "MSVFW32" _
   Alias "GetSaveFileNamePreviewA" (filestruct As OPENFILENAME) As Long

Public Enum EOpenFile
   OFN_READONLY = &H1
   OFN_OVERWRITEPROMPT = &H2
   OFN_HIDEREADONLY = &H4
   OFN_NOCHANGEDIR = &H8
   OFN_SHOWHELP = &H10
   OFN_ENABLEHOOK = &H20
   OFN_ENABLETEMPLATE = &H40
   OFN_ENABLETEMPLATEHANDLE = &H80
   OFN_NOVALIDATE = &H100
   OFN_ALLOWMULTISELECT = &H200
   OFN_EXTENSIONDIFFERENT = &H400
   OFN_PATHMUSTEXIST = &H800
   OFN_FILEMUSTEXIST = &H1000
   OFN_CREATEPROMPT = &H2000
   OFN_SHAREAWARE = &H4000
   OFN_NOREADONLYRETURN = &H8000
   OFN_NOTESTFILECREATE = &H10000
   OFN_NONETWORKBUTTON = &H20000
   OFN_NOLONGNAMES = &H40000
   OFN_EXPLORER = &H80000
```

```
    OFN_NODEREFERENCELINKS = &H100000
    OFN_LONGNAMES = &H200000
End Enum

Private Declare Function CommDlgExtendedError Lib "COMDLG32" () As Long

Public Enum EDialogError
    CDERR_DIALOGFAILURE = &HFFFF
    CDERR_GENERALCODES = &H0
    CDERR_STRUCTSIZE = &H1
    CDERR_INITIALIZATION = &H2
    CDERR_NOTEMPLATE = &H3
    CDERR_NOHINSTANCE = &H4
    CDERR_LOADSTRFAILURE = &H5
    CDERR_FINDRESFAILURE = &H6
    CDERR_LOADRESFAILURE = &H7
    CDERR_LOCKRESFAILURE = &H8
    CDERR_MEMALLOCFAILURE = &H9
    CDERR_MEMLOCKFAILURE = &HA
    CDERR_NOHOOK = &HB
    CDERR_REGISTERMSGFAIL = &HC
    PDERR_PRINTERCODES = &H1000
    PDERR_SETUPFAILURE = &H1001
    PDERR_PARSEFAILURE = &H1002
    PDERR_RETDEFFAILURE = &H1003
    PDERR_LOADDRVFAILURE = &H1004
    PDERR_GETDEVMODEFAIL = &H1005
    PDERR_INITFAILURE = &H1006
    PDERR_NODEVICES = &H1007
    PDERR_NODEFAULTPRN = &H1008
    PDERR_DNDMMISMATCH = &H1009
    PDERR_CREATEICFAILURE = &H100A
```

```
    PDERR_PRINTERNOTFOUND = &H100B
    PDERR_DEFAULTDIFFERENT = &H100C
    CFERR_CHOOSEFONTCODES = &H2000
    CFERR_NOFONTS = &H2001
    CFERR_MAXLESSTHANMIN = &H2002
    FNERR_FILENAMECODES = &H3000
    FNERR_SUBCLASSFAILURE = &H3001
    FNERR_INVALIDFILENAME = &H3002
    FNERR_BUFFERTOOSMALL = &H3003
    CCERR_CHOOSECOLORCODES = &H5000
End Enum

Private Declare Function SHBrowseForFolder Lib "Shell32" (lpbi As TBrowseInfo) As Long
Private Declare Function SHGetPathFromIDList Lib "Shell32" Alias "SHGetPathFromIDListA" _
                        (ByVal pidl As Long, _
                        ByVal pszPath As String) As Long
Public Declare Sub CoTaskMemFree Lib "ole32.dll" (ByVal pv As Long
Private Type TBrowseInfo
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Public Enum BROWSE_FLAGS
   vbBIF_NONE = &H0&
    vbBIF_RETURNONLYFSDIRS = &H1&
    vbBIF_DONTGOBELOWDOMAIN = &H2&
```

```
    vbBIF_STATUSTEXT = &H4&
    vbBIF_RETURNFSANCESTORS = &H8&
    vbBIF_BROWSEFORCOMPUTER = &H1000&
    vbBIF_BROWSEFORPRINTER = &H2000&
    vbBIF_BROWSEINCLUDEFILES = &H4000&
End Enum

Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
                        (ByVal hWnd As Long, _
                        ByVal wMsg As Long, _
                        ByVal wParam As Long, _
                        lParam As Any) As Long
Private Const WM_USER As Long = &H400&
Private Const BFFM_INITIALIZED As Long = 1&
Private Const BFFM_SETSTATUSTEXTA As Long = (WM_USER + 100)
Private Const BFFM_SETSTATUSTEXTW As Long = (WM_USER + 104)
Private Const BFFM_ENABLEOK As Long = (WM_USER + 101)
Private Const BFFM_SETSELECTIONA As Long = (WM_USER + 102)
Private Const BFFM_SETSELECTIONW As Long = (WM_USER + 103
Private Const LMEM_FIXED As Long = &H0&
Private Const LMEM_ZEROINIT As Long = &H40&
Private Declare Function LocalAlloc Lib "kernel32" _
                        (ByVal uFlags As Long, _
                        ByVal uBytes As Long) As Long
Private Declare Function LocalFree Lib "kernel32" _
                        (ByVal hMem As Long) As Long
Private Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" _
                        (pDest As Any, _
                        pSource As Any, _
                        ByVal dwLength As Long)
```

```
Private Const sEmpty As String = ""
Public Function VBGetOpenFileName(filename As String, _
                Optional FileTitle As String, _
                Optional FileMustExist As Boolean = True, _
                Optional MultiSelect As Boolean = False, _
                Optional ReadOnly As Boolean = False, _
                Optional HideReadOnly As Boolean = False, _
                Optional filter As String = "All (*.*)| *.*", _
                Optional FilterIndex As Long = 1, _
                Optional InitDir As String = "", _
                Optional DlgTitle As String = "", _
                Optional DefaultExt As String = "", _
                Optional Owner As Long = -1, _
                Optional Flags As Long = 0) As Boolean
  Dim opfile As OPENFILENAME, s As String, afFlags As Long
With opfile
  .lStructSize = Len(opfile)
  .Flags = (-FileMustExist * OFN_FILEMUSTEXIST) Or _
       (-MultiSelect * OFN_ALLOWMULTISELECT) Or _
       (-ReadOnly * OFN_READONLY) Or _
       (-HideReadOnly * OFN_HIDEREADONLY) Or _
       (Flags And CLng(Not (OFN_ENABLEHOOK Or _
                OFN_ENABLETEMPLATE)))
  If Owner <> -1 Then .hwndOwner = Owner
  .lpstrInitialDir = lnitDir
  .lpstrDefExt = DefaultExt
  .lpstrTitle = DlgTitle
  Dim ch As String, i As Integer
  For i = 1 To Len(filter)
    ch = Mid$(filter, i, 1)
    If ch = "|" Or ch = ":" Then
      s = s & vbNullChar
```

```
        Else
            s = s & ch
        End If
    Next
    s = s & vbNullChar & vbNullChar
    .lpstrFilter = s
    .nFilterIndex = FilterIndex
    s = filename & String$(MAX_PATH - Len(filename), 0)
    .lpstrFile = s
    .nMaxFile = MAX_PATH
    s = FileTitle & String$(MAX_FILE - Len(FileTitle), 0)
    .lpstrFileTitle = s
    .nMaxFileTitle = MAX_FILE
    If GetOpenFileName(opfile) Then
        VBGetOpenFileName = True
        filename = Left$(.lpstrFile, InStr(.lpstrFile, vbNullChar) - 1)
        FileTitle = Left$(.lpstrFileTitle, InStr(.lpstrFileTitle, vbNullChar) - 1)
        Flags = .Flags
        FilterIndex = .nFilterIndex
        filter = FilterLookup(.lpstrFilter, FilterIndex)
        If (.Flags And OFN_READONLY) Then ReadOnly = True
    Else
        VBGetOpenFileName = False
        filename = vbNullChar
        FileTitle = vbNullChar
        Flags = 0
        FilterIndex = -1
        filter = vbNullChar
    End If
End With
End Function
```

```
Public Function VBGetSaveFileName(filename As String, _
                Optional FileTitle As String, _
                Optional OverWritePrompt As Boolean = True, _
                Optional filter As String = "All (*.*)| *.*", _
                Optional FilterIndex As Long = 1, _
                Optional InitDir As String = "", _
                Optional DlgTitle As String = "", _
                Optional DefaultExt As String = "", _
                Optional Owner As Long = -1, _
                Optional Flags As Long) As Boolean
  Dim opfile As OPENFILENAME, s As String
With opfile
  .lStructSize = Len(opfile)
  .Flags = (-OverWritePrompt * OFN_OVERWRITEPROMPT) Or _
       OFN_HIDEREADONLY Or _
       (Flags And CLng(Not (OFN_ENABLEHOOK Or _
                OFN_ENABLETEMPLATE)))
  If Owner <> -1 Then .hwndOwner = Owner
  .lpstrInitialDir = InitDir
  .lpstrDefExt = DefaultExt
  .lpstrTitle = DlgTitle

  Dim ch As String, i As Integer
  For i = 1 To Len(filter)
    ch = Mid$(filter, i, 1)
    If ch = "|" Or ch = ":" Then
      s = s & vbNullChar
    Else
      s = s & ch
    End If
  Next
  s = s & vbNullChar & vbNullChar
```

```
    .lpstrFilter = s
    .nFilterIndex = FilterIndex
    s = filename & String$(MAX_PATH - Len(filename), 0)
    .lpstrFile = s
    .nMaxFile = MAX_PATH
    s = FileTitle & String$(MAX_FILE - Len(FileTitle), 0)
    .lpstrFileTitle = s
    .nMaxFileTitle = MAX_FILE
    If GetSaveFileName(opfile) Then
       VBGetSaveFileName = True
       filename = Left$(.lpstrFile, InStr(.lpstrFile, vbNullChar) - 1)
       FileTitle = Left$(.lpstrFileTitle, InStr(.lpstrFileTitle, vbNullChar) - 1)
       Flags = .Flags
       FilterIndex = .nFilterIndex
       filter = FilterLookup(.lpstrFilter, FilterIndex)
    Else
       VBGetSaveFileName = False
       filename = vbNullChar
       FileTitle = vbNullChar
       Flags = 0
       FilterIndex = 0
       filter = vbNullChar
    End If
End With
End Function

Private Function FilterLookup(ByVal sFilters As String, ByVal iCur As Long) As String
    Dim iStart As Long, iEnd As Long, s As String
    iStart = 1
    If sFilters = vbNullChar Then Exit Function
    Do
       iEnd = InStr(iStart, sFilters, vbNullChar)
```

```
        If iEnd = 0 Then Exit Function
        iEnd = InStr(iEnd + 1, sFilters, vbNullChar)
        If iEnd Then
            s = Mid$(sFilters, iStart, iEnd - iStart)
        Else
            s = Mid$(sFilters, iStart)
        End If
        iStart = iEnd + 1
        If iCur = 1 Then
            FilterLookup = s
            Exit Function
        End If
        iCur = iCur - 1
    Loop While iCur
End Function

Public Function VBGetFileTitle(sFile As String) As String
    Dim sFileTitle As String, cFileTitle As Integer
    cFileTitle = MAX_PATH
    sFileTitle = String$(MAX_PATH, 0)
    cFileTitle = GetFileTitle(sFile, sFileTitle, MAX_PATH)
    If cFileTitle Then
        VBGetFileTitle = ""
    Else
        VBGetFileTitle = Left$(sFileTitle, InStr(sFileTitle, vbNullChar) - 1)
    End If
End Function

Public Function VBGetOpenFileNamePreview(filename As String, _
                Optional FileTitle As String, _
                Optional FileMustExist As Boolean = True, _
                Optional MultiSelect As Boolean = False, _
```

```
                Optional ReadOnly As Boolean = False, _
                Optional HideReadOnly As Boolean = False, _
                Optional filter As String = "All (*.*)| *.*", _
                Optional FilterIndex As Long = 1, _
                Optional InitDir As String, _
                Optional DlgTitle As String, _
                Optional DefaultExt As String, _
                Optional Owner As Long = -1, _
                Optional Flags As Long = 0) As Boolean
    Dim opfile As OPENFILENAME, s As String, afFlags As Long
With opfile
    .lStructSize = Len(opfile)
    .Flags = (-FileMustExist * OFN_FILEMUSTEXIST) Or _
          (-MultiSelect * OFN_ALLOWMULTISELECT) Or _
          (-ReadOnly * OFN_READONLY) Or _
          (-HideReadOnly * OFN_HIDEREADONLY) Or _
          (Flags And CLng(Not (OFN_ENABLEHOOK Or _
                OFN_ENABLETEMPLATE)))
    If Owner <> -1 Then .hwndOwner = Owner
    .lpstrInitialDir = InitDir
    .lpstrDefExt = DefaultExt
    .lpstrTitle = DlgTitle
    Dim ch As String, i As Integer
    For i = 1 To Len(filter)
      ch = Mid$(filter, i, 1)
      If ch = "|" Or ch = ":" Then
        s = s & vbNullChar
      Else
        s = s & ch
      End If
    Next
    s = s & vbNullChar & vbNullChar
```

```
    .lpstrFilter = s
    .nFilterIndex = FilterIndex
    s = filename & String$(MAX_PATH - Len(filename), 0)
    .lpstrFile = s
    .nMaxFile = MAX_PATH
    s = FileTitle & String$(MAX_FILE - Len(FileTitle), 0)
    .lpstrFileTitle = s
    .nMaxFileTitle = MAX_FILE
    If GetOpenFileNamePreview(opfile) Then
       VBGetOpenFileNamePreview = True
       filename = Left$(.lpstrFile, InStr(.lpstrFile, vbNullChar) - 1)
       FileTitle = Left$(.lpstrFileTitle, InStr(.lpstrFileTitle, vbNullChar) - 1)
       Flags = .Flags
       FilterIndex = .nFilterIndex
       filter = FilterLookup(.lpstrFilter, FilterIndex)
       If (.Flags And OFN_READONLY) Then ReadOnly = True
    Else
       VBGetOpenFileNamePreview = False
       filename = vbNullChar
       FileTitle = vbNullChar
       Flags = 0
       FilterIndex = -1
       filter = vbNullChar
    End If
End With
End Function

Public Function VBGetSaveFileNamePreview(filename As String, _
                Optional FileTitle As String = "", _
                Optional FileMustExist As Boolean = True, _
                Optional MultiSelect As Boolean = False, _
                Optional ReadOnly As Boolean = False, _
```

```
                Optional HideReadOnly As Boolean = False, _
                Optional filter As String = "All (*.*)| *.*", _
                Optional FilterIndex As Long = 1, _
                Optional InitDir As String = "", _
                Optional DlgTitle As String = "", _
                Optional DefaultExt As String = "", _
                Optional Owner As Long = -1, _
                Optional Flags As Long = 0) As Boolean
    Dim opfile As OPENFILENAME, s As String, afFlags As Long
With opfile
    .lStructSize = Len(opfile)
    .Flags = (-FileMustExist * OFN_FILEMUSTEXIST) Or _
          (-MultiSelect * OFN_ALLOWMULTISELECT) Or _
          (-ReadOnly * OFN_READONLY) Or _
          (-HideReadOnly * OFN_HIDEREADONLY) Or _
          (Flags And CLng(Not (OFN_ENABLEHOOK Or _
                OFN_ENABLETEMPLATE)))
    If Owner <> -1 Then .hwndOwner = Owner
    .lpstrInitialDir = InitDir
    .lpstrDefExt = DefaultExt
    .lpstrTitle = DlgTitle
    Dim ch As String, i As Integer
    For i = 1 To Len(filter)
      ch = Mid$(filter, i, 1)
      If ch = "|" Or ch = ":" Then
        s = s & vbNullChar
      Else
        s = s & ch
      End If
    Next
    s = s & vbNullChar & vbNullChar
    .lpstrFilter = s
```

```
    .nFilterIndex = FilterIndex

    s = filename & String$(MAX_PATH - Len(filename), 0)
    .lpstrFile = s
    .nMaxFile = MAX_PATH
    s = FileTitle & String$(MAX_FILE - Len(FileTitle), 0)
    .lpstrFileTitle = s
    .nMaxFileTitle = MAX_FILE
    If GetSaveFileNamePreview(opfile) Then
       VBGetSaveFileNamePreview = True
       filename = Left$(.lpstrFile, InStr(.lpstrFile, vbNullChar) - 1)
       FileTitle = Left$(.lpstrFileTitle, InStr(.lpstrFileTitle, vbNullChar) - 1)
       Flags = .Flags

       FilterIndex = .nFilterIndex

       filter = FilterLookup(.lpstrFilter, FilterIndex)
       If (.Flags And OFN_READONLY) Then ReadOnly = True
    Else
       VBGetSaveFileNamePreview = False
       filename = vbNullChar
       FileTitle = vbNullChar
       Flags = 0
       FilterIndex = -1
       filter = vbNullChar
    End If
End With
End Function

Public Function BrowseForFolder(ByVal hwndOwner As Long, _
                                ByVal sTitle As String, _
                                Optional ByVal initFolder As String = "", _
```

```
                Optional ByVal vRoot As SPECIAL_FOLDERS = vbCSIDL_DESKTOP, _
                Optional ByVal Flags As BROWSE_FLAGS = vbBIF_NONE) As String
    Dim BI As TBrowseInfo
    Dim lpsz As Long
    Dim pidl As Long
    Dim sPath As String
    BI.hwndOwner = hwndOwner
    BI.pszDisplayName = String$(MAX_PATH, 0)
    BI.lpszTitle = sTitle
    BI.pidlRoot = vRoot
    BI.ulFlags = Flags
    BI.lpfn = vbGetProcAddress(AddressOf BrowseCallbackProc)
    If initFolder <> "" Then
       lpsz = LocalAlloc(LMEM_FIXED Or LMEM_ZEROINIT, Len(initFolder))
       Call MoveMemory(ByVal lpsz, ByVal initFolder, Len(initFolder))
       BI.lParam = lpsz
    End If
    'show dialog here
    pidl = SHBrowseForFolder(BI)
    sPath = String$(MAX_PATH, 0)
    Call SHGetPathFromIDList(pidl, sPath)
    If pidl <> 0 Then
       Call CoTaskMemFree(pidl)
       BrowseForFolder = StrZToStr(sPath)
    End If
    If lpsz <> 0 Then
       Call LocalFree(lpsz)
    End If
End Function

Private Function StrZToStr(s As String) As String
Dim TempString As String
```

```
    TempString = Left$(s, InStr(s, vbNullChar) - 1)
    If TempString = "" Then
       StrZToStr = s
    Else
       StrZToStr = TempString
    End If
End Function


Private Function ExistFile(ByVal sSpec As String) As Boolean
    On Error Resume Next
    Call FileLen(sSpec)
    ExistFile = (Err = 0)
End Function
Private Function vbGetProcAddress(ByVal lpfunc As Long) As Long
    vbGetProcAddress = lpfunc
End Function
Private Function BrowseCallbackProc(ByVal hWnd As Long, _
                        ByVal uMsg As Long, _
                        ByVal lParam As Long, _
                        ByVal lpData As Long) As Long
  Sclect Case uMsg
     Case BFFM_INITIALIZED
       Call SendMessage(hWnd, BFFM_SETSELECTIONA, _
                1&, ByVal lpData)
       Case Else
       End Select
End Function
```

Infineon
technologies

Security & Chip Card ICs

SLE 4436/36E

Intelligent 221–Bit EEPROM Counter
for > 20000 Units with Security Logic
and High Security Authentication

Short Product Information 07.99

Pin Description

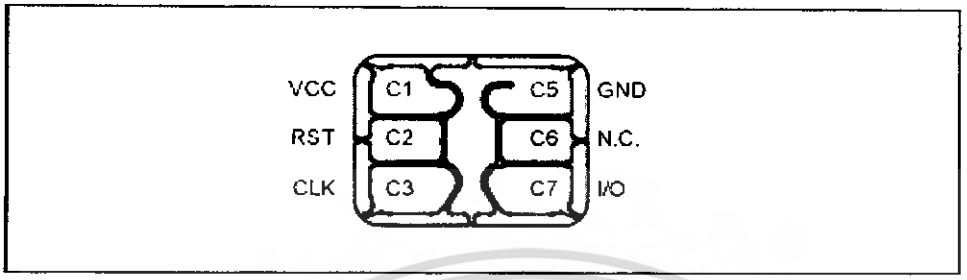


Figure 1 Pin Configuration Wire-bonded Module (top view)

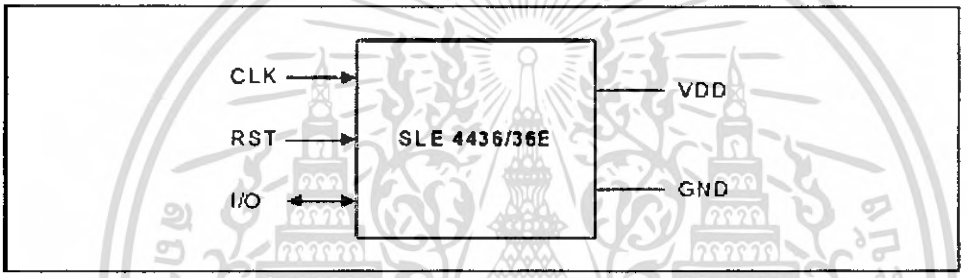


Figure 2 Pad Configuration Die

Table 2 Pin Definitions and Functions

Card Contact	Symbol	Function
C1	VCC	Supply voltage
C2	RST	Control input (Reset Signal)
C3	CLK	Clock input
C5	GND	Ground
C6	N.C.	Not connected
C7	I/O	Bi-directional data line (open drain)

General Description

SLE 4436/36E is designed for applications in prepaid telephone cards. The chip consists of an EEPROM memory of 221 bit, a ROM of 16 bits, a control/security unit and a special computing unit for chip authentication. The shaded blocks in the block diagram (Figure 3) contain the enhanced features of SLE 4436/36E compared to SLE 4406/06E.

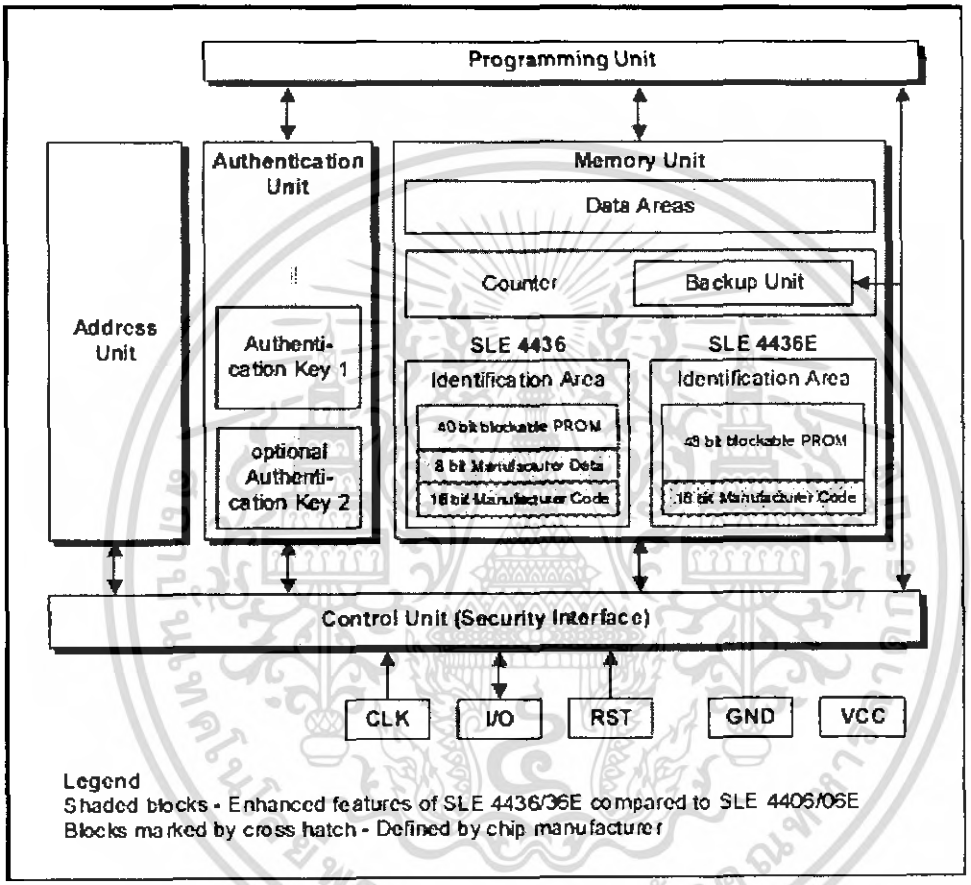
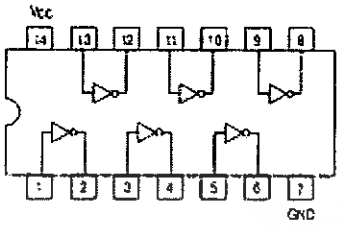


Figure 3 Block Diagram

- **Memory Unit**
  Counter, Identification Data (e.g. serial number, expiry date) and Data Areas.
- **Address Unit**
  Setting of the address counter is synchronously with the CLK.
- **Programming Unit**
  The programming voltage for the EEPROM/PROM is generated internally.

SN74LS04

Hex Inverter



ON Semiconductor

http://onsemi.com

LOW POWER SCHOTTKY

GUARANTEED OPERATING RANGES

Symbol	Parameter	Min	Typ	Max	Unit
V_{CC}	Supply Voltage	4.75	5.0	5.25	V
T_A	Operating Ambient Temperature Range	0	25	70	°C
I_{OH}	Output Current – High			-0.4	mA
I_{OL}	Output Current – Low			8.0	mA

PLASTIC
N SUFFIX
CASE 646

SOIC
D SUFFIX
CASE 751A

SOEIAJ
M SUFFIX
CASE 965

ORDERING INFORMATION

Device	Package	Shipping
SN74LS04N	14 Pin DIP	2000 Units/Box
SN74LS04D	SOIC-14	55 Units/Rail
SN74LS04DR2	SOIC-14	2500/Tape & Reel
SN74LS04M	SOEIAJ-14	See Note 1
SN74LS04MEL	SOEIAJ-14	See Note 1

1. For ordering information on the EIAJ version of the SOIC package, please contact your local ON Semiconductor representative.

Bookmark

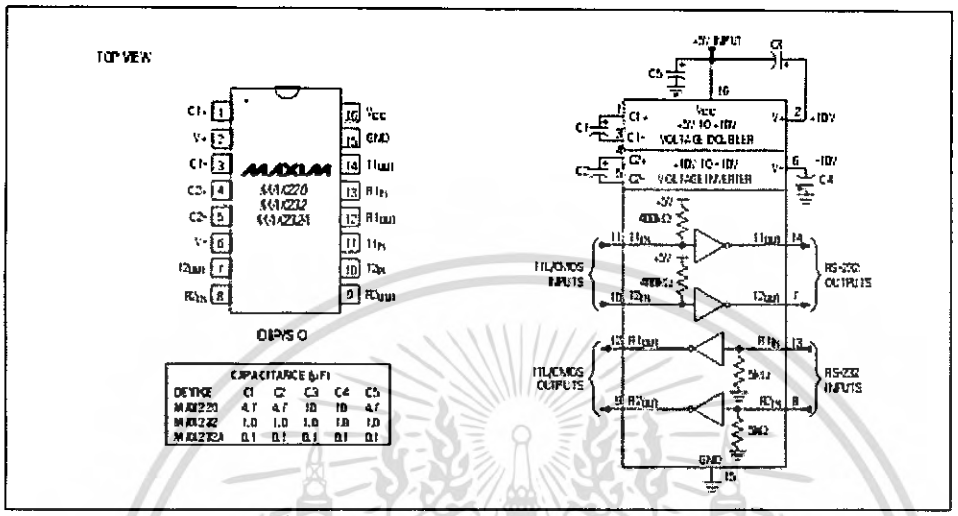


Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

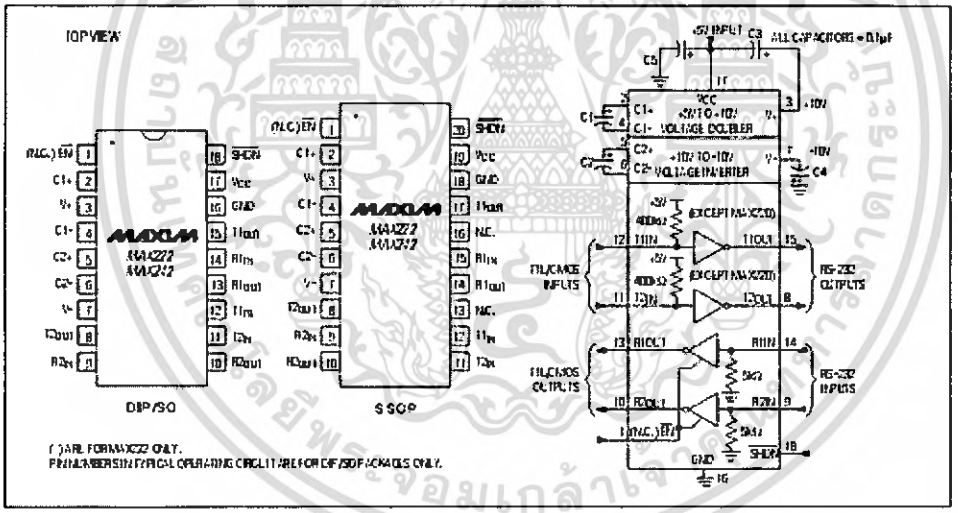
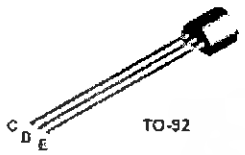


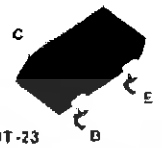
Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

FAIRCHILD
SEMICONDUCTOR

2N4403 MMBT4403



TO-92



SOT-23
Mark: 2T

PNP General Purpose Amplifier

This device is designed for use as a general purpose amplifier and switch requiring collector currents to 500 mA.

Absolute Maximum Ratings*

TA = 25°C unless otherwise noted

Symbol	Parameter	Value	Units
V_{CEO}	Collector-Emitter Voltage	40	V
V_{CBO}	Collector-Base Voltage	40	V
V_{EBO}	Emitter-Base Voltage	5.0	V
I_C	Collector Current - Continuous	600	mA
T_J, T_{stg}	Operating and Storage Junction Temperature Range	-55 to +150	°C

* These ratings are limiting values above which the serviceability of any semiconductor device may be impaired.

NOTES:
1) These ratings are based on a maximum junction temperature of 150 degrees C.
2) These are steady state limits. The factory should be consulted on applications involving pulsed or low duty cycle operations.

Thermal Characteristics

TA = 25°C unless otherwise noted

Symbol	Characteristic	Max		Units
		2N4403	*MMBT4403	
P_D	Total Device Dissipation Derate above 25°C	625 5.0	350 2.8	mW mW/°C
$R_{\theta JC}$	Thermal Resistance, Junction to Case	83.3		°C/W
$R_{\theta JA}$	Thermal Resistance, Junction to Ambient	200	357	°C/W

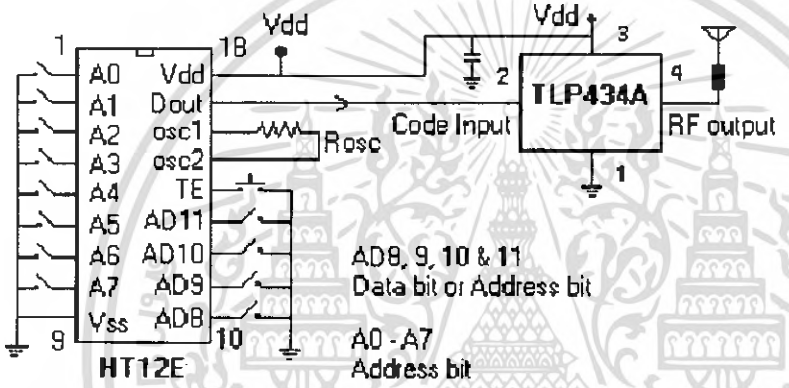
* Device mounted on FR-4 PCB 1.6" X 1.6" X 0.06".

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Vcc	Operating supply voltage		2.0	-	12.0	V
Icc 1	Peak Current (2V)		-	-	1.64	mA
Icc 2	Peak Current (12V)		-	-	19.4	mA
Vh	Input High Voltage	Idata= 100uA (High)	Vcc-0.5	Vcc	Vcc+0.5	V
Vl	Input Low Voltage	Idata= 0 uA (Low)	-	-	0.3	V
FO	Absolute Frequency	315Mhz module	314.8	315	315.2	MHz
PO	RF Output Power- 50ohm	Vcc = 9V-12V	-	16	-	dBm
		Vcc = 5V-6V	-	14	-	dBm
DR	Data Rate	External Encoding	512	4.8K	200K	bps

Notes : (Case Temperature = 25°C +- 2°C , Test Load Impedance = 50 ohm)

Application Circuit :

Typical Key-chain Transmitter using HT12E-18DIP, a Binary 12 bit Encoder from Holtek Semiconductor Inc.



Symbol	Parameter	Conditions	Min	Typ	Max	
Vcc	Operating supply voltage		3.3	5.0V	6.0	V
Iod	Operating Current		-	4.5		mA
Vdata	Data Out	Idata = +200 uA (High)	Vcc-0.5	-	Vcc	V
		Idata = -10 uA (Low)	-	-	0.3	V

Electrical Characteristics

Characteristics	SYM	Min	Typ	Max	Unit
Operation Radio Frequency	FC	315, 418 and 433.92			MHz
Sensitivity	Pref		-110		dBm
Channel Width			+-500		Khz
Noise Equivalent BW			1		Khz
Receiver Turn On Time			5		ms
Operation Temperature	Top	-20	-	80	C
Baseband Data Rate			1.8		KHz

Application Circuit :

Typical RF Receiver using HT12D-18DIP, a Binary 12 bit Decoder with 8 bit uC HT-18RXX from Holtek Semiconductor Inc.