ระบบควบคุมและรักษาความปลอดภัยการจอดรถอัตโนมัติ

AUTOMATIC CONTROL & SECURITY CARPARKING

โดย

นางสาวกนกวลี สมัยมาก 49015043

นายปิยะณัฐ สามคำ 49015062

นายวุฒิพงษ์ ดวงฤทธิ์ 49015071

อาจารย์ที่ปรึกษา

อาจารย์สุรพล บุญจันทร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ระบบควบคุมและรักษาความปลอดภัยการจอดรถอัตโนมัติ

AUTOMATIC CONTROL & SECURITY CARPARKING

โดย

นางสาวกนกวลี สมัยมาก

นายปิยะณัฐ สามคำ

นายวุฒิพงษ์ ดวงฤทธิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญญานิพนธ์ปีการศึกษา 2551

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบควบคุมและรักษาความปลอดภัยการจอดรถอัตโนมัติ

AUTOMATIC CONTROL & SECURITY CARPARKING

ผู้จัดทำ

1. น.ส.กนกวลี สมัยมาก 49015043
2. นาย ปิยะณัฐ สามคำ 49015062
3. นาย นายวุฒิพงษ์ ดวงฤทธิ์ 49015071

_____ อาจารย์ที่ปรึกษา

(อาจารย์สุรพล บุญจันทร์)

โครงงานเลขที่ 512525

ระบบควบคุมและรักษาความปลอดภัยการจอดรถอัตโนมัติ

AUTOMATIC CONTROL AND SECURITY CARPARKING

โดย นางสาว กนกวลี สมัยมาก 49015043

นาย ปิยะณัฐ สามคำ 49015062

นาย วุฒิพงษ์ ดวงฤทธิ์ 49015071

อาจารย์ที่ปรึกษา ผศ.สุรพล บุญจันทร์

บทคัดย่อ

โครงงานปริญญานิพนธ์นี้ เป็นโครงงานที่พัฒนาขึ้นเพื่อความสะดวกและความปลอดภัยของผู้ใช้บริการลานจอดรถ โดยนำเอาไมโครคอนโทรเลอร์ มาทำงานร่วมกับโปรแกรมทางคอมพิวเตอร์เพื่อช่วยควบคุมระบบดังกล่าวโดยใช้เซนเซอร์ในการตรวจสอบที่ว่างและจำนวนรถในลานจอดรถแล้วนำข้อมูลทั้งหมดที่ได้ส่งไปยังคอมพิวเตอร์โดยผ่านไมโครคอนโทรเลอร์ ซึ่งค่าที่ได้จะถูกนำไปคำนวณเพื่อแสดงจำนวนที่จอดรถทั้งหมดรวมไปถึงค่าบริการสำหรับการจอด และ ยังถ่ายรูปป้ายทะเบียนกับคนขับไว้ด้วยเพื่อเปรียบเทียบระหว่างทางเข้ากับทางออกและยังสามารถเรียกภาพมาดูได้ในภายหลังเมื่อต้องการ

ABSTRACT

This project develop for conveniently and security of customer service. Use microcontroller work to share a program computer for control this project . Detect system using sensors scan each car park position and send obtained signals to computer by microcontroller . After that ,Bring it to calaulate all position car parking and parking fee then take a photo of licence plate and driver to compare between entrance and exit. The photo can request to audit for security.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้จะสำเร็จลุล่วงด้วยดี เนื่องมาจากคณะผู้จัดทำได้รับการอนุเคราะห์อย่างสูงยิ่งจาก ผศ.สุรพล บุญจันทร์ ที่ได้ช่วยเหลือให้คำแนะนำชี้แนวทางในการแก้ปัญหา ตลอดจนความรู้ความเข้าใจในด้านต่างๆ

ขอขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมโทรคมนาคมทุกท่าน ที่ประสิทธ์ประสาทวิชาความรู้ทางด้านโทรคมนาคม ให้ทางคณะผู้จัดทำได้มีแนวทางในการทำโครงงานนี้

ขอขอบคุณเพื่อนภาควิชาวิศวกรรมโทรคมนาคมทุกคนที่ได้ช่วยเหลือให้การทำโครงงานนี้สำเร็จไปได้ด้วยดี

สุดท้ายนี้ขอขอบพระคุณพ่อแม่ ที่ให้ความรัก ความห่วงใย กำลังใจ กำลังทรัพย์ และทุกสิ่งทุกอย่างกับทางคณะผู้จัดทำ

คณะผู้จัดทำ

สารบัญ

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ปัจจุบันนี้สังคมเมืองเติบโตขึ้นมากมีรถยนต์ตามท้องถนนมากขึ้นทำให้เกิดปัญหาการหาที่จอดรถได้แสนลำบากเรื่องนี้เป็นปัญหามาก ยิ่งตามสถานที่ราชการหรือสถานที่คนไปกันเยอะแทบไม่มีที่จอดรถเลยทำให้บางครั้งต้องมีการจอดรถไว้อีกสถานที่แล้วก็เดินมา (บางครั้งก็ไกลมาก) หรือจอดไว้ตามข้างทางจนทำให้เกิดปัญหารถติดตามมาอีก เนื่องจากจำนวนรถที่เข้ามาจอดมีจำนวนมากหรืออาจจะมีการแย่งที่จอดรถกันโดยเฉพาะในเวลาที่เร่งด่วน ซึ่งทำให้เสียเวลาหรือค่าใช้จ่าย(น้ำมัน) โดยเปล่าประโยชน์

โครงงานนี้สามารถนำมาใช้เพื่อลดปัญหาต่างๆเหล่านี้ลงได้ โดยระบบจะมีการนับรถทุกๆคันที่เข้ามาจากเซนเซอร์ที่ถูกติดตั้งไว้ตรงทางเข้า และมีการแสดงจำนวนที่ว่าง ตำแหน่งที่ว่างโดยมีการถ่ายภาพรถ เพื่อทำการเปรียบเทียบเมื่อมีการเข้า-ออก เพื่อความปลอดภัยในทรัพย์สินของผู้ใช้บริการอีกด้วย

โครงงานชิ้นนี้จะเป็นการนำเอาไมโครคอนโทรลเลอร์มาประยุกต์ใช้งานร่วมกับโปรแกรมคอมพิวเตอร์ของระบบที่จอดรถอัตโนมัติ

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1.2.1 เพื่อศึกษาการออกแบบและสร้างแบบจำลองอาคารจอดรถอัตโนมัติ

1.2.2 เพื่อศึกษาการทำงานของ Microcontroller (MCS-51) สำหรับการควบคุมระบบ Displayแสดงผล,การเชื่อมต่อกับคอมพิวเตอร์,ระบบการขับเคลื่อนมอเตอร์

1.2.3 เพื่อศึกษาและทำความเข้าใจเกี่ยวกับการเขียนโปรแกรมเดลไฟ

1.2.4 เพื่อศึกษาการใช้งานไมโครคอลโทลเลอร์

1.3 ขอบเขตของปริญญานิพนธ์

1.3.1 ขอบเขตของอุปกรณ์ทางด้านฮาร์ดแวร์

1.3.1.1 ระบบเซนเซอร์ทั้งภาครับและภาคส่งสามารถใช้ตรวจเช็คสภาวะต่างๆ

1.3.1.2 สามารถใช้ Microcontroller ควบคุมระบบ Display แสดงผล

1.3.1.3 สามารถใช้ Microcontroller เชื่อมต่อกับคอมพิวเตอร์

1.3.1.4 สามารถใช้ Microcontroller ควบคุมระบบขับเคลื่อนมอเตอร์

1.3.2 ขอบเขตทางด้านซอฟต์แวร์

1.3.2.1 เขียนโปนแกรมภาษาซีให้กับ Microcontroller เพื่อใช้เชื่อมต่อกับโปรแกรมคอมพิวเตอร์

1.3.2.2 เขียนโปรแกรมเดลไฟ เพื่อใช้เชื่อมต่อกับกล้อง

1.3.2.3 เขียนโปรแกรมเดลไฟ เพื่อใช้ควบคุมระบบการทำงานอาคารจอดรถอัตโนมัติ

1.3.2.4 เขียนโปรแกรมเดลไฟ เพื่อใช้เชื่อมต่อกับเครื่องปรินส์เตอร์

1.4 ประโยชน์ที่คาดว่าจะได้รับจากปริญญานิพนธ์

1.4.1 สามารถนำสิ่งประดิษฐ์นี้ไปใช้เป็นต้นแบบในการสร้างได้

1.4.2 สามารถทำให้การจอดรถสะดวกยิ่งขึ้น

1.5 แผนการดำเนินงาน

หลังจากที่ผู้จัดทำโครงการได้ทำการกำหนดจุดประสงค์ของโครงการและขอบเขตของโครงการเป็นที่เรียบร้อยแล้ว ต่อมาจึงได้วางแผนการดำเนินงานซึ่งสรุปเป็นตารางแผนภูมิการทำงานได้ดังนี้

ขั้นตอนการดำเนินงาน	ระยะเวลาการดำเนินงาน									
	2551								2552	
	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.
1. ออกแบบแผนผังการทำงานของโครงงาน	⟷									
2. ศึกษาพื้นฐานการทำงานของไมโครคอนโทรลเลอร์ และโปรแกรม Delphi 7		⟷	⟷	⟷						
3. ศึกษาการทำงาน sensor และ IC 555			⟷	⟷						

4. ออกแบบและสร้างวงจรควบคุมไมโครคอนโทรลเลอร์				↔	↔	↔				
5. ออกแบบและสร้างวงจรควบคุมเซนเซอร์, Display แสดงผล ,ชุดขับเคลื่อนมอเตอร์				↔	↔	↔				
6. ออกแบบแบบจำลองอาคารจอดรถและสร้างอาคารจอดรถ				↔	↔	↔				
7. ติดตั้งวงจรต่างๆลงในแบบจำลองอาคารจอดรถ						↔				
8. เขียนโปรแกรมเดลไฟ ในส่วนประมวลผลหลัก , เชื่อมต่อกับกล้อง , เชื่อมต่อกับปริ้นเตอร์		↔	↔	↔	↔	↔	↔			
9. แก้ไขส่วนต่างๆในตัวแบบจำลองอาคารจอดรถ และแก้ไขในส่วนของโปรแกรม							↔	↔		
10. ทำการทดลองและบันทึกผลการทดลอง									↔	↔
11. รายงาน และ สรุปผลโครงการ										↔

ตารางที่ 1.1 ตารางแผนภูมิการดำเนินงาน

บทที่ 2

ทฤษฎีและหลักการ

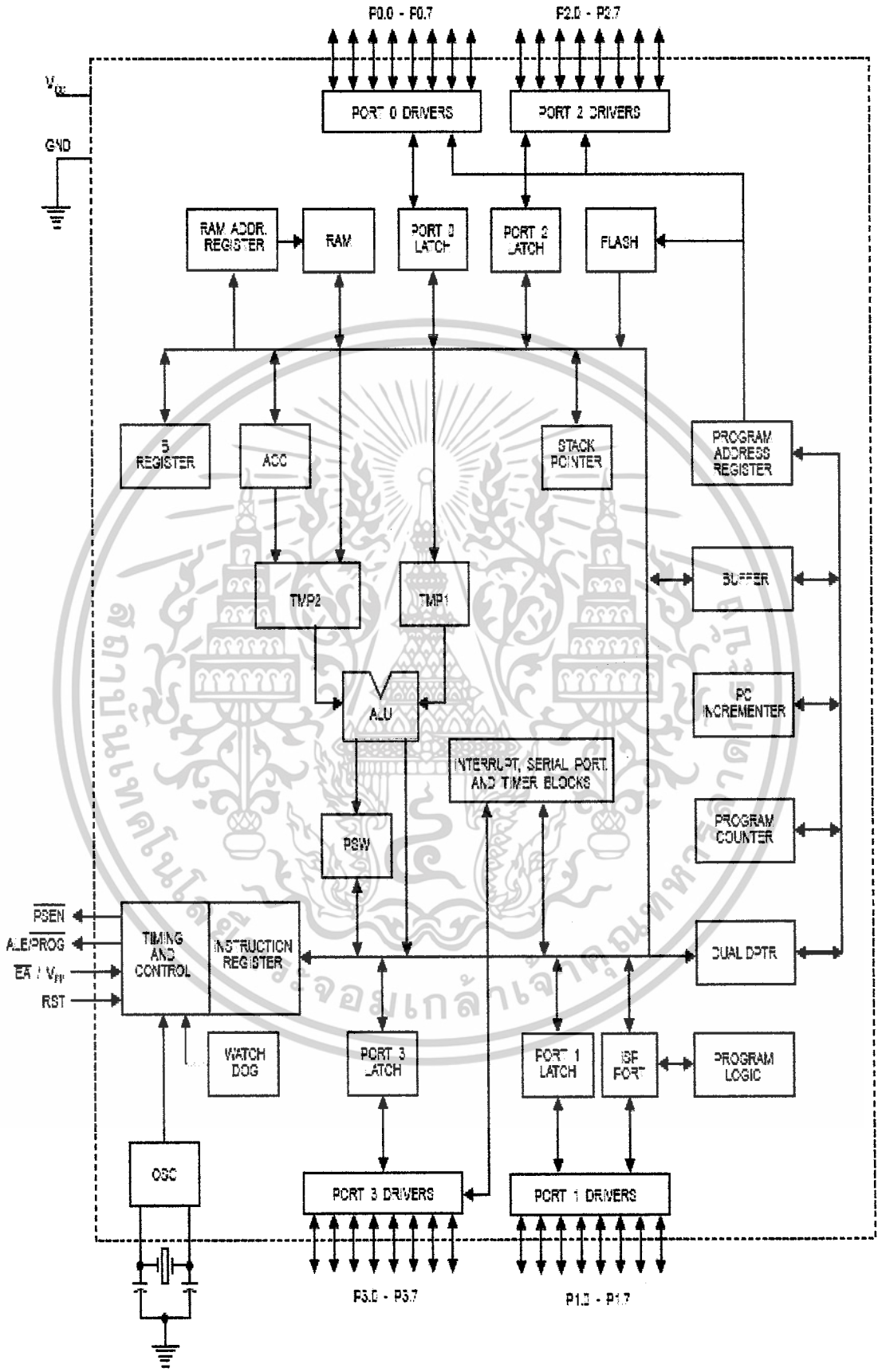
2.1 ไมโครคอนโทรลเลอร์ MCS-51

โครงสร้างภายในพื้นฐานของไมโครคอนโทรลเลอร์ตระกูล MCS-51 เบอร์ 8051 ประกอบด้วยอุปกรณ์ต่างๆดังนี้ ส่วนของหน่วยความจำภายในสำหรับเก็บข้อมูลขนาด 128 ไบท์ (Internal Data Memory) ส่วนของหน่วยความจำภายในสำหรับเก็บโปรแกรมที่มีขนาด 4 กิโลไบท์ (Internal Program Memory) อุปกรณ์ควบคุมการอินเตอร์รัพท์ (Interrupt Control Unit) ตัวตั้งเวลาและตัวนับเวลาขนาด 16 บิท 2 ชุด (Timer/Counter 0 and Timer/Counter 1) พอร์ตควบคุมการสื่อสารอนุกรมแบบ Full Duplex ซึ่งสามารถรับส่งข้อมูลพร้อมกันได้ พอร์ตขนานสำหรับติดต่อกับอุปกรณ์ภายนอกจำนวน 4 พอร์ต พอร์ตละ 8 บิท และวงจรผลิตสัญญาณนาฬิกาภายใน

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีจำนวนมาก ขึ้นกับโครงสร้างภายใน บางเบอร์มีหน่วยความจำภายในเป็นแบบรอม บางเบอร์เป็นแบบอีพรอม บางเบอร์มีแรมภายใน 128 ไบท์ บางเบอร์มี 256 ไบท์ เป็นต้น ซึ่งคุณสมบัติทั่วไปของ MCS-51 มีดังนี้

1. ใช้ HMCS และ CHMOS เทคโนโลยีในการสร้างและทำงานด้วยแหล่งจ่ายไฟขนาด 5Vเพียงแหล่งเดียว
2. ซีพียูมีขนาด 8บิต
3. มีวงจรออสซิลเลเตอร์ และวงจรนาฬิกาบนชิป
4. รีจิสเตอร์แบงก์ 4 ชุด
5. รีจีสเตอร์แบงก์ 6 ชุด
6. มีTimer,Counter ขนาด 16 บิต 2 ชุดและสำหรับเบอร์ 8032/8052 มี 3ชุด
7. มีพอร์ตไอโอแบบขนานสองทิศทางจำนวน 4 พอร์ต พอร์ตละ 8บิตรวมทั้งหมดเป็น 32 เส้นแต่จะเหลือเพียง 16 เส้นสำหรับเบอร์ 8031 อีก 16 เส้นจะใช้ในการเข้าถึงทางแอดเดรสและข้อมูล
8. พอร์ตแบบอนุกรมสามารถที่จะโปรแกรมการรับส่งแบบ Full Duplex ที่มีความเร็วสูง
9. หนึ่งวัฏจักรคำสั่งจะกินเวลา 1 ไมโครวินาที ด้วยการใช้ crystal 12 MHz
10. แอดเดรสข้อมูลภายนอกได้ 64 กิโลไบต์
11. แอดเดรสโปรแกรมภายนอกได้ 64 กิโลไบต์
12. สามารถกำหนดเลขที่อยู่ของข้อมูลได้ในระดับไบต์และระดับบิต
13. มีซอฟต์แวร์บิตแฟลกสำหรับผู้ที่จะกำหนดเองได้ 128 ตำแหน่งบิต
14. โครงสร้างอินเตอร์รัพต์จะติดตั้งได้ 5 แหล่ง และ 6 แหล่งสำหรับ 8032/8052 พร้อมด้วยการจัดไพรโอริตี้ได้ 2 ระดับ
15. ตัวโพรเซสเซอร์สามารถใช้งานแบบบูลีนได้ สำหรับการใช้กับกระบวนการงานควบคุม
16. มีคำสั่งคูณ และหารทางฮาร์ดแวร์ที่ทำงานได้ภายใน 4 ไมโครวินาที
17. ตัวเลขทางคณิตศาสตร์ใช้ได้ทั้งในระดับไบนารีและเดซิมอล
18. การใช้พื้นที่สแตกสำหรับโปรแกรมย่อยต่างทำได้กว้างกว่า MCS-48

บล็อกไดอะแกรม

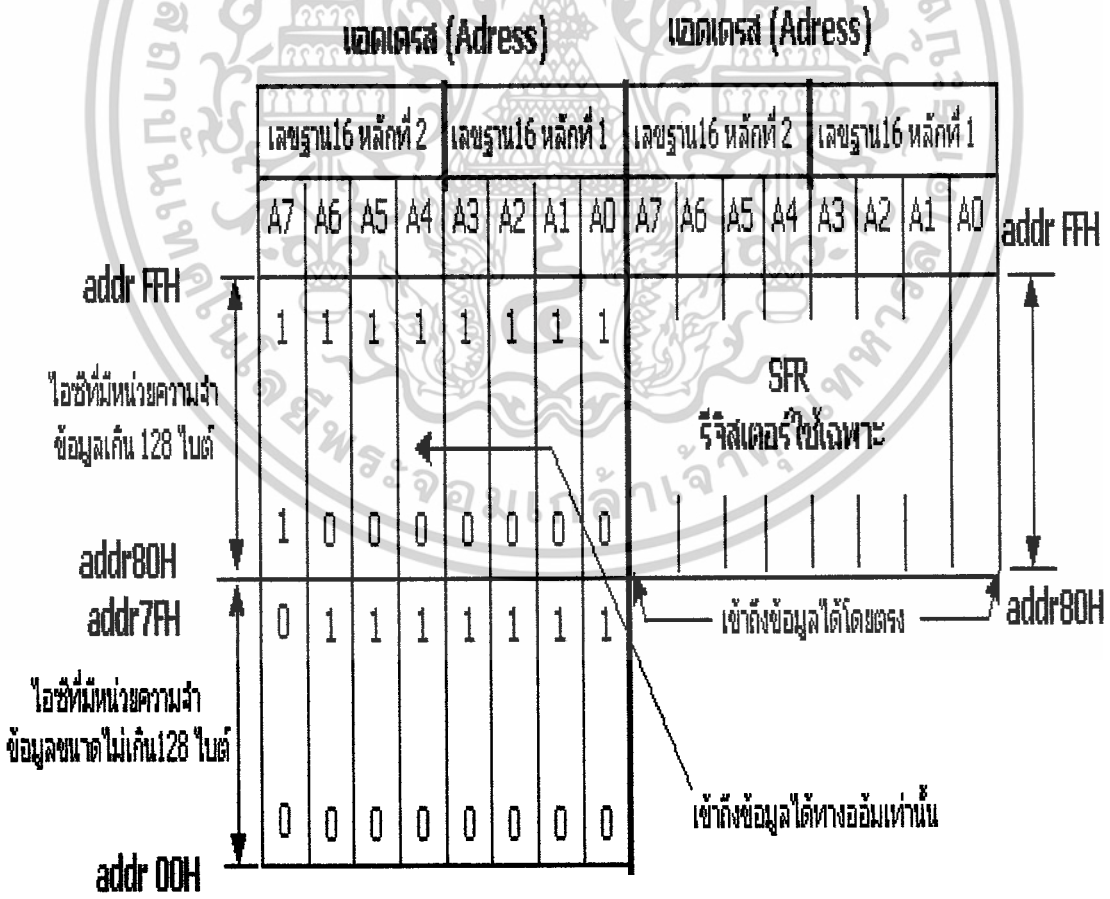


รูปที่ 2.1 แสดงโครงสร้างภายในของ MCS-51

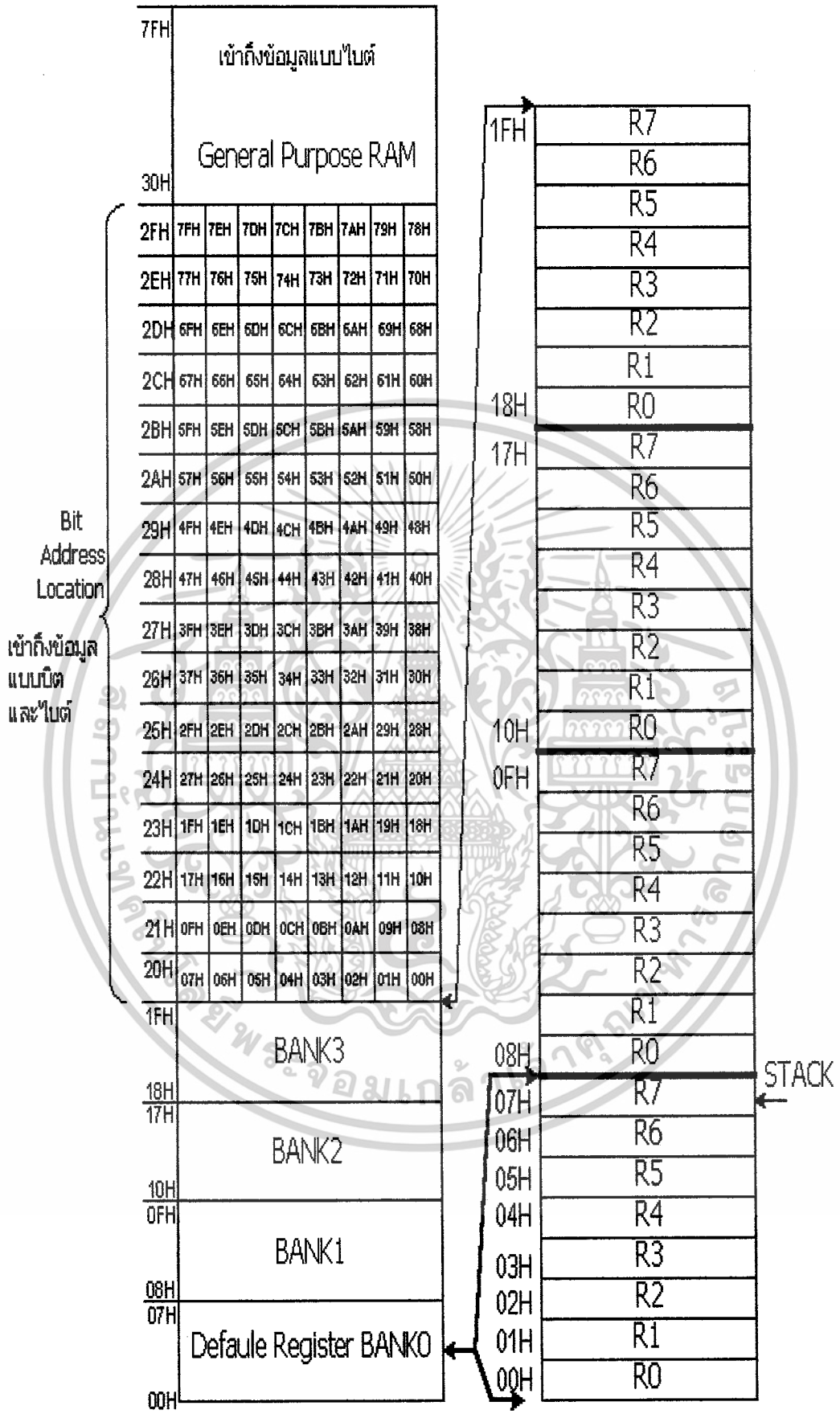
2.1.1 หน่วยความจำภายใน

หน่วยความจำสำหรับเก็บโปรแกรมทำหน้าที่เก็บโปรแกรมที่ผู้ใช้เขียนขึ้น เพื่อควบคุมการทำงานของ ไมโครคอนโทรลเลอร์ โดยหน่วยความจำจะเป็นแบบ ROM มีความจุ 4 KBytes (ตำแหน่ง 0000H - 0FFFH) ในการใช้งาน เราสามารถกำหนดให้ไมโครคอนโทรลเลอร์ เลือกใช้โปรแกรมที่เก็บอยู่ภายในตัวไมโครคอนโทรลเลอร์ หรือโปรแกรมที่เก็บอยู่ในหน่วยความจำ (EPROM) ที่อยู่ภายนอกก็ได้ การเลือกการติดต่อทำได้โดย การป้อนสัญญาณควบคุมที่ขา EA (External Access) ถ้าต้องการให้ไมโครคอนโทรลเลอร์ติดต่อกับ โปรแกรมที่อยู่ในหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ จะต่อขานี้ กับลอจิก 1 หากต้องการให้ ไมโครคอนโทรลเลอร์ ติดต่อกับโปรแกรมที่เก็บอยู่ในหน่วยความจำภายนอก จะต่อขานี้กับลอจิก 0 การติดต่อกับหน่วยความโปรแกรมจำภายนอก จะติดต่อได้ทั้งหมด 64 Kbytes (ตำแหน่ง 0000H – FFFFH)

ในกรณีที่กำหนด ให้ไมโครคอนโทรลเลอร์ ติดต่อกับโปรแกรมที่อยู่ในหน่วยความจำ ภายในตัวไมโครคอนโทรลเลอร์จะติดต่อได้ 4 Kbytes หากตำแหน่งของโปรแกรม มีค่าเกินกว่าตำแหน่งของหน่วยความจำภายใน (โปรแกรมยาวเกินกว่า 4 Kbytes) ตัวไมโครคอนโทรลเลอร์ จะทำการติดต่อกับโปรแกรมที่อยู่ในหน่วยความจำภายนอกอัตโนมัติ



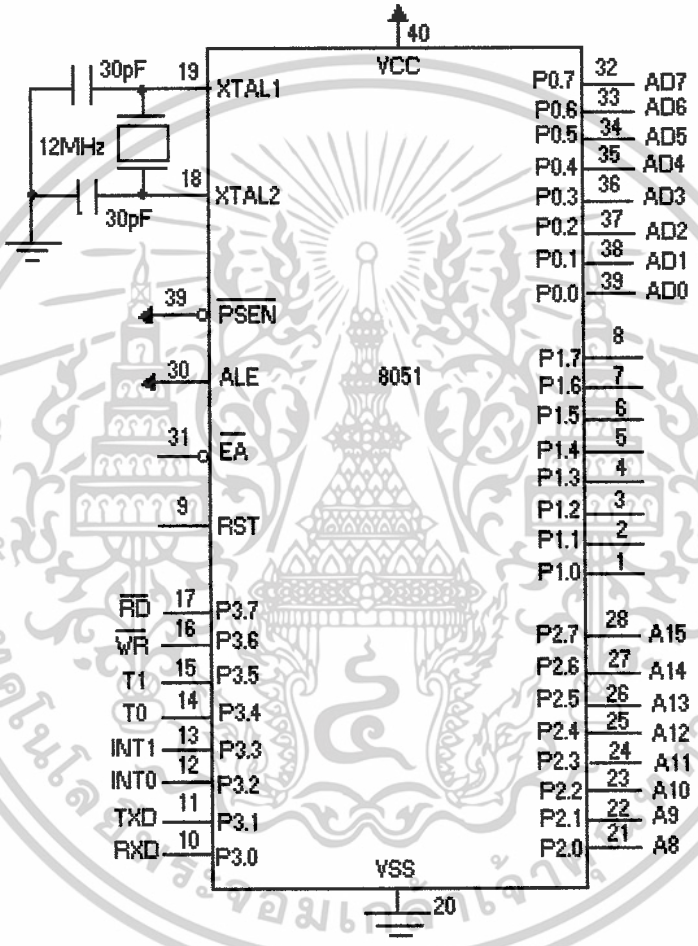
ตารางที่ 2.1 แสดงตำแหน่งแอดเดรสของหน่วยความจำข้อมูลภายใน(Internal data memory)



ตารางที่ 2.2 แสดงการเข้าถึงข้อมูลแบบบิตและไบต์

2.1.2 รีจีสเตอร์ภายในไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 จะมีรีจิสเตอร์ซึ่งอำนวยความสะดวกในการใช้งานตามคำสั่งต่างๆ และมีแอคคิวมูเลเตอร์ รีจีสเตอร์ B ซึ่งใช้ในการคูณและหาร รีจิสเตอร์สถานะ เป็นตัวชี้ตำแหน่งข้อมูล 2x8 บิตและ 1x16 บิต พอร์ตหมายเลขศูนย์ถึงพอร์ตหมายเลขสามและมีรีจิสเตอร์เป็นตัวชี้ข้อมูล ซึ่งใช้ส่งและรับข้อมูลแบบอนุกรม มีรีจิสเตอร์ 16 บิต ที่เป็นวงจรตั้งเวลาและวงจรนับรีจิสเตอร์ซึ่งจะจองไว้สำหรับใช้สำหรับตัวที่ 3 เป็นรีจิสเตอร์คำสั่งสำหรับหน้าที่พิเศษ เช่น การขัดจังหวะ เรียลไทม์คล็อก (Realtime clock)



รูปที่ 2.2 การจัดวางขาต่างๆ ของไมโครคอนโทรลเลอร์ MCS-51

2.1.3 ความเร็วของไมโครคอนโทรลเลอร์ MCS-51

เมื่อเทียบกับไมโครคอนโทรลเลอร์ 6502 และไมโครโปรเซสเซอร์ Z80 คำสั่งทั้งหมดของไมโครคอนโทรลเลอร์ MCS-51 จะทำงานด้วยรอบคำสั่งเดียวกับภาษาเครื่อง คือภายในระยะเวลาของสัญญาณนาฬิกา 12 ลูกและเมื่อใช้ความถี่ 12 เมกะเฮิรตซ์ ดังนั้นแล้วหนึ่งรอบคำสั่งของภาษาเครื่องจะเท่ากับ 1 ไมโครวินาที ซึ่งความเร็วในการประมวลผลของไมโครคอนโทรลเลอร์ MCS-51 เท่ากับไอซี 6502 หน่วยประมวลผลกลางซึ่งใช้ความถี่ 2 เมกะเฮิรตซ์ หรือจะเท่ากับไมโครโปรเซสเซอร์ Z80 มีหน่วย

ประมวลผลกลางที่จะใช้ความถี่ 8 บิต ต่อหนึ่งครั้ง เพื่อให้ความสะดวกในการเขียนโปรแกรมโดยไม่ต้องลำบากในการปิดกั้น (Mask) บิตที่ไม่ต้องการ

2.1.4 สายต่างๆของบัสและพอร์ต

โครงสร้างภายในของไมโครคอนโทรลเลอร์เบอร์ MCS-51 โดยให้มีบัสสองทิศทาง 4 เส้น และพอร์ทขนาด 8 บิตตามทฤษฎี แต่ที่จริงแล้วนั้นจะได้ก็ต่อเมื่อมีการใช้หน่วยความจำ และภายในตัวรอมหรือแรม เมื่อไม่ใช้หน่วยความจำภายใน พอร์ต 0 และ 2 ใช้เป็นบัสของข้อมูลและตำแหน่ง ดังนั้น พอร์ต 2 พอร์ต ใช้งานเป็นอินพุทและเอาท์พุท ซึ่งพอร์ต 2 เป็นสายสัญญาณ ตำแหน่ง A15...A8 ส่วนพอร์ต 0 ทำหน้าที่เป็นสายสัญญาณตำแหน่ง A7...A0 ออกจากบิตข้อมูล D7...D0 เอาท์พุทของขา RD และ WR มาจากสายสัญญาณเอาท์พุทของพอร์ต P3 โดยโปรแกรมภายในใช้สายสัญญาณ RD และ WR เพื่อการเขียนข้อมูลและอ่านข้อมูลกับหน่วยความจำภายนอก และจะไม่ทำงานเมื่อมีภาษาเครื่องเก็บอยู่ในหน่วยความจำภายในและถ้าหากขา PSEN เป็นขารับสัญญาณสำหรับเปิดให้มีการอ่านหน่วยความจำภายนอก ถ้าสังเกตทุกๆรอบคำสั่งระหว่างการทำงานด้วยโปรแกรมในรอมหรือหน่วยความจำแบบ อีอีพรอม สัญญาณ PSEN ทำงานถึงสองครั้งเหมือนสัญญาณ ALE เพราะว่ามีการอ่านข้อมูลจำนวน 8 ไบท์ หน่วยความจำภายนอกไม่มีข้อมูลบรรจุอยู่ขา PSEN ก็จะไม่ทำงานเช่นกันและส่วนไมโครคอนโทรลเลอร์เบอร์ 8052AH จะไม่ใช้สัญญาณขา PSEN เลย ขา EA เป็นขาอินพุทที่จะใช้ร่วมกับตำแหน่งสัญญาณภายนอก โดยจะมีค่าลอจิก "0" และเมื่อไมโครโปรเซสเซอร์ทำการอ่านคำสั่งจากหน่วยความจำภายนอกซึ่งโดยปกติแล้วไมโครคอนโทรลเลอร์จะอ่านหน่วยความจำภายในน้อยกว่าอ่านจากหน่วยความจำภายนอก ขา EA ยังเป็นขาอินพุทสำหรับป้อนแรงดันไฟฟ้า 21 โวลต์ เพื่อที่จะเขียนโปรแกรมให้กับหน่วยความจำแบบอีอีพรอม และสำหรับกรณีที่ใช้ไมโครคอนโทรลเลอร์ 8051 หรือไมโครคอนโทรลเลอร์ 8052

2.1.5 วงจรนับ/วงจรตั้งเวลา

ไมโครคอนโทรลเลอร์เบอร์ 8052 มีวงจรนับและวงจรตั้งเวลาชนิด 16 บิต มากกว่าไมโครคอนโทรลเลอร์เบอร์ 8051 อยู่หนึ่งตัว ในการทำงานของวงจรนับและวงจรตั้งเวลาเป็นดังนี้ เมื่อทำงานเป็นวงจรตั้งเวลา รีจิสเตอร์วงจรนับจะเพิ่มขึ้นเมื่อมีสัญญาณป้อนให้ทางอินพุท T0,T1 หรือ T2 มีเฉพาะไมโครคอนโทรลเลอร์ 8052 เป็นของสัญญาณขาลง อัตราการนับสัญญาณสูงสุดคือ 1/24 ของความเร็วสัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ วงจรนับและวงจรตั้งเวลา 0 และ 1 มีวิธีโปรแกรมให้ทำงานได้ต่างกันถึง 4 แบบ ซึ่งรวมทั้งการทำงานเป็น 8 บิต หรือ 16 บิต และการบรรจุค่าพรีเซตหนึ่งค่าได้เองอย่างอัตโนมัติ วงจรนับและวงจรตั้งเวลาที่ 1 ซึ่งเลือกโปรแกรมให้ทำหน้าที่เป็นตัวกำเนิดสัญญาณของอัตราการส่งบิตออกไปยังพอร์ตอนุกรม สำหรับใช้เชื่อมต่อ วงจรนับและวงจรตั้งเวลาที่ 2 เฉพาะไมโครคอนโทรลเลอร์ 8052 เท่านั้นและมีการทำงานย่อยๆ อีก 3 ชนิด ดังนี้

1. วงจรนับ 16 บิต ที่สามารถโหลดค่ากลับคืนเองอย่างอัตโนมัติ
2. วงจรนับที่จองไว้ชนิด 16 บิต
3. วงจรกำเนิดสัญญาณของการส่งบิต เพื่อใช้ในการเชื่อมโยงข้อมูล

2.1.6 การรับส่งข้อมูลอนุกรม

คุณสมบัติที่สำคัญข้อหนึ่งของของ MCS-51คือ พอร์ตแบบอนุกรมที่มีอยู่ในตัวชิป 8051 ในไมโครโปรเซสเซอร์ทั่วไป การเพิ่มพอร์ตอนุกรมจะต้องเพิ่ม UART และวงจรควบคุมไอซีที่สนับสนุนการเชื่อมต่อแบบอนุกรม ซึ่งอุปกรณ์เหล่านี้มีความซับซ้อนและมีราคาค่อนข้างสูงการเชื่อต่อแบอนุกรมของไมโครคอนโทรลเลอร์ 8051 นั้นเป็นแบบ full duplex ซึ่งหมายความว่ามันสามารถทำการรับและส่งข้อมูลได้ในเวลาเดียวกัน การเชื่อต่อแบบอนุกรมนี้จะมีการพักข้อมูลที่รับเข้ามา (receive-buffered) ซึ่งหมายความว่าก่อนที่ข้อมูลชิ้นแรกจะถูกส่งไปยังไมโครโปรเซสเซอร์ พอร์ตอนุกรมจะสามารถรอรับข้อมูลอนุกรมชิ้นที่สองได้ อย่างไรก็ตามข้อมูลชิ้นแรกจะต้องถูกส่งไปยังไมโครโปรเซสเซอร์ก่อนที่ข้อมูลชิ้นที่สองที่รับเข้ามาจะถูกนำไปเก็บในแลตช์ข้อมูล มิฉะนั้นข้อมูลชิ้นแรกจะถูกเขียนทับ การเชื่อมต่อแบบอนุกรมนี้มีการใช้รีจีสเตอร์ 2 ตัว ที่ทำหน้าที่รับและส่งข้อมูล เราติดต่อกับรีจีสเตอร์ทั้งสองตัวนี้ได้โดยการอ้างอิงรีจีสเตอร์พิเศษตัวหนึ่งที่มีชื่อว่า SBUF ถ้าเราทำการเขียนข้อมูลลง SBUF แสดงว่าเราอ่านข้อมูลที่อยู่ในรีจีสเตอร์รับข้อมูล จะเห็นได้ว่ารีจีสเตอร์ที่ทำหน้าที่รับและส่งข้อมูลทั้งสองนี้มีค่าแอดเดรสค่าเดียวกัน

2.1.7 การขัดจังหวะและชุดคำสั่ง

ไมโครคอนโทรลเลอร์ 8051 รับการขัดจังหวะได้ 5 แหล่ง ส่วนไมโครคอนโทรลเลอร์ 8052 รับการขัดจังหวะจากอุปกรณ์อื่นๆ ได้ถึง 6 แหล่ง คือ ขา INT0 และขา INT1 ซึ่งกำหนดให้ใช้ระดับพัลส์หรือขอบขาของพัลส์ก็ได้ วงจรนับและวงจรตั้งเวลาที่ 0 และ 1 ซึ่งสำหรับการใช้งานไมโครคอนโทรลเลอร์ 8051 จะเพิ่มวงจรตั้งเวลาหรือวงจรนับตัวที่ 2 และตัวสุดท้ายจากพอร์ตอนุกรมสามารถกำหนดลำดับความสำคัญของการขัดจังหวะได้ 2 ระดับไม่ต้องอาศัยวงจรจากภายนอกเข้ามาช่วยแต่ละแหล่งการขัดจังหวะ 5 หรือ 6 แหล่งนั้น สามารถกำหนดให้เป็นเวคเตอร์เฉพาะ ตัวชี้ตำแหน่ง ดังนั้นเมื่อมีการขัดจังหวะเข้ามา ตัวโปรเซสเซอร์จะกระโดดไปที่ส่วนของโปรแกรมที่ทำงานตามวัตถุประสงค์ของการขัดจังหวะนั้นและหลังจากเก็บข้อมูลต่างๆของโปรแกรมการนับลงในสแตก

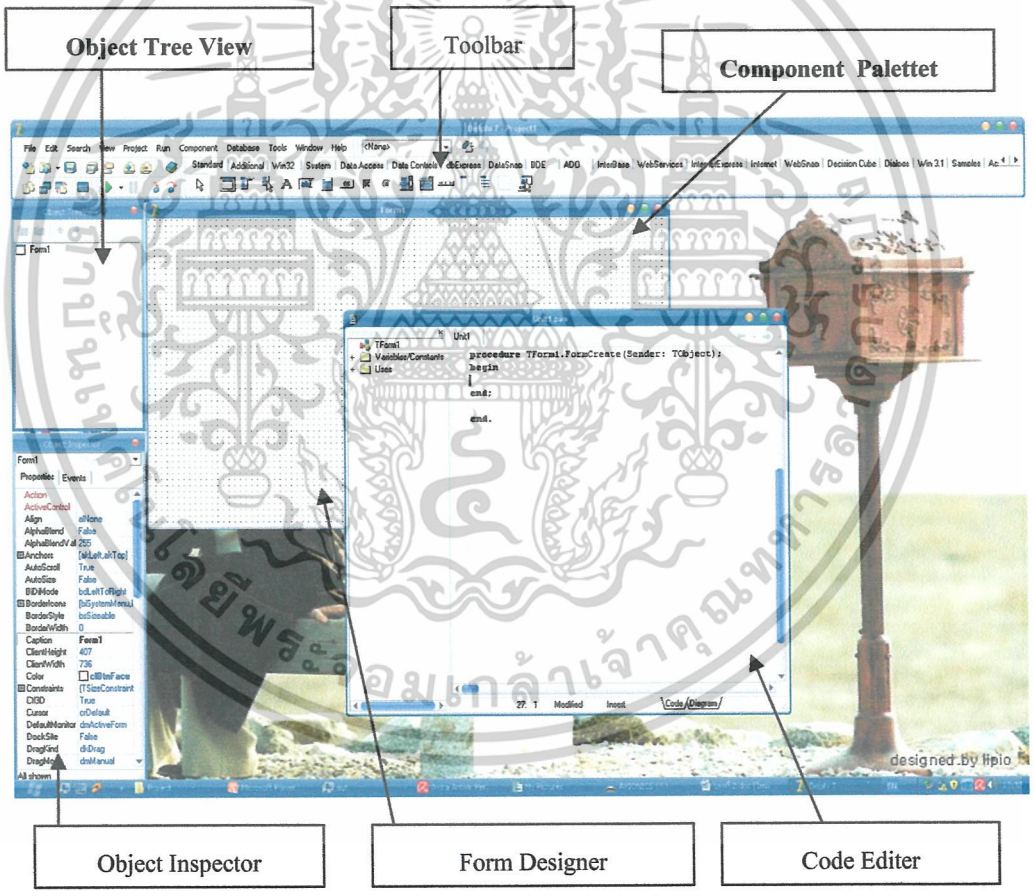
2.2 Delphi 7

Delphi 7 คือซอฟต์แวร์ที่เรานำมาใช้ในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชั่น หรือซอฟต์แวร์อีกที่ โดยมันจะประกอบไปด้วยเครื่องมือชนิดต่างๆ ที่ใช้ทำให้การเขียนโปรแกรมทำได้ง่ายและสะดวก

Delphi 7 จัดเป็นเครื่องมือเขียนโปรแกรมชนิด Visual Programming เช่นเดียวกับ Visual Basic หรือ Visual C++ โดยมีข้อดีคือ สามารถเขียนโปรแกรมได้ง่าย และให้ผลงานออกมาอย่างรวดเร็ว ซึ่งจะแตกต่างจากเครื่องมือเขียนโปรแกรมรุ่นเดิมๆ เช่น Turbo Pascal หรือ Borland C ที่มีความยุ่งยากในการใช้งานและการเรียนรู้ในการเขียนโปรแกรม สามารถเขียนโปรแกรมได้ง่าย และให้ผลงานออกมาอย่างรวดเร็ว ดังนั้นจึงจัดให้ Delphi 7 เป็นซอฟต์แวร์ประเภท RAD หรือ Rapid Application Development ซึ่งแปลว่าสามารถสร้างแอพพลิเคชันได้อย่างรวดเร็ว

2.2.1 จุดเด่นของ Delphi 7

Delphi 7 นั้นผ่านการพัฒนามาเกือบ 10 ปี ตั้งแต่เวอร์ชัน 1.0 ที่ทำงานบน Window 3.1X โดยมีจุดเด่นมากตั้งแต่สมัยนั้นคือ โปรแกรมที่ได้จากการเขียนโปรแกรมมีขนาดเล็ก ทำงานได้อย่างรวดเร็ว ซึ่งมักจะถูกนำไปเปรียบเทียบกับ Visual Basic 3.0 ในสมัยนั้น อีกประการหนึ่ง Delphi ใช้ภาษาออบเจ็กต์ปาสคาล จึงเคยถูกเปรียบว่าเป็น Visual Pascal มาแล้ว เวอร์ชันปัจจุบันของ Delphi นั้นได้รับการพัฒนาให้สามารถสร้างแอพพลิเคชันที่ทำงานบน Windows ได้ดีเหมือนเดิม โดยมีการปรับปรุงให้สามารถพัฒนาแอพพลิเคชันตามแนวความคิดของ .NET ซึ่งจะช่วยให้สามารถเขียนโปรแกรมครั้งเดียว แล้วนำไปใช้งานบนอุปกรณ์ต่างๆ ไม่ว่าจะเป็น PDA,โทรศัพท์มือถือ และ สามารถใช้งานบนเว็บได้ ขณะเดียวกันนั้น Delphi 7 ก็ได้รับการพัฒนาให้สามารพัฒนาแอพพลิเคชันแบบข้ามแพล็ตฟอร์มได้ นั่นคือสามารถพัฒนาแอพพลิเคชันที่ทำงานได้ทั้งบน Windows และ Linux นั่นเอง



รูปที่ 2.3 แสดงเมนูการใช้งานต่างๆของ Delphi 7

2.2.2 หลักการเขียนโปรแกรมด้วย Delphi 7

หลักการเขียนโปรแกรมกับ Delphi7 ซึ่งจะใช้ภาษาออบเจ็กต์ปาสคาล(คล้ายกับปาสคาล แต่มีความสามารถด้าน OOP เพิ่มเข้ามา) โดยจะมีหลักการเบื้องต้น และแสดงตัวอย่างประกอบในทุกหัวข้อ เพื่อให้เกิดความเข้าใจ ซึ่งผู้อ่านสามารถทดลองเขียนโปรแกรมตามตัวอย่างได้ทันที

2.2.3 ตัวแปรและชนิดข้อมูล

ก่อนจะเขียนโปรแกรมได้นั้นสิ่งแรกที่ต้องรู้จักก็คือ ข้อมูล (Data) เพราะข้อมูลคือ สิ่งที่เราใช้ในการควบคุม และแลกเปลี่ยนกันระหว่างผู้ใช้งานกับตัวโปรแกรม

เมื่อมีข้อมูลแล้วเราจะต้องสร้างสิ่งที่เรียกว่า ตัวแปร (Variable) ขึ้นมาเก็บข้อมูลนั้นไว้ ซึ่ง Delphi 7 นั้น สามารถทำงานกับข้อมูลได้หลายชนิด โดยแบ่งข้อมูลชนิดต่างๆนั้นได้เป็นกลุ่มดังต่อไปนี้

- ข้อมูลชนิดเบื้องต้น (Simple Type) บางครั้งเรียกว่า Pre-define Data type เป็นชนิดข้อมูลเบื้องต้นที่ Delphi ใช้งานพื้นฐาน เช่น ตัวอักษร, เลขจำนวนเต็ม,เลขทศนิยม,ค่าตรรกยะ เป็นต้น

- ข้อมูลชนิดข้อความ(String) เป็นการนำเอาข้อมูลเบื้องต้นชนิดตัวอักษรมาเรียงต่อกัน

- ข้อมูลชนิดโครงสร้าง (Structure Type) เป็นชนิดข้อมูลเกิดจากการนำเอาชนิดข้อมูลเบื้องต้นมาประกอบกันเป็นโครงสร้างเพื่อเก็บข้อมูลเป็นชุดๆ เช่น อาร์เรย์,ไฟล์,

- ข้อมูลชนิดอ้างอิง(Pointer) เป็นข้อมูลที่เราใช้ในการอ้างอิงกับหน่วยความจำในคอมพิวเตอร์ซึ่งจะใช้บอกตำแหน่งเก็บข้อมูล

จะเห็นได้ว่า Delpi 7 นั้นสามารถทำงานได้กับข้อมูลหลายรูปแบบ มีการแบ่งตัวแปรออกเป็นหมวดหมู่ใหม่เลือกใช้งานได้ตามลักษณะของข้อมูลที่จะใช้งานโดยเราจะใช้สิ่งที่เรียกว่าตัวแปร (Variable) มาใช้ในการเก็บข้อมูล ซึ่งถ้าจะเปรียบเทียบไปแล้ว ตัวแปรก็เหมือนรถบรรทุก ซึ่งจะบรรทุกข้อมูลซึ่งมีหลากหลายชนิดกันนั่นหมายถึงก็ต้องมีรถบรรทุกหลายแบบให้เหมาะกับข้อมูลแต่ละชนิด

2.2.3.1 การประกาศตัวแปร

ในการใช้งานข้อมูลนั้นต้องมีตัวแปรมารองรับ ตัวแปรจะถูกใช้งานได้ก็ต้องมีการประกาศ (Variable Declaration) เสียก่อน ซึ่งก็เหมือนกับต้องมาลงทะเบียนให้ Delphi รู้จักก่อนถึงจะใช้งานได้

สำหรับการประกาศตัวแปร มักจะประกาศไว้ตอนต้นโปรแกรม หรือตอนต้นของโปรแกรมย่อย ซึ่งจะประกาศตัวแปรชนิดเดียวพร้อมกันหลายๆตัวก็ได้ เราอาจจะตั้งชื่อตัวแปรให้สอดคล้องกับความหมายของข้อมูลที่เก็บไว้ภายใน ซึ่งจะทำให้โปรแกรมที่เราเขียนขึ้นมานั้นอ่านได้ง่าย เมื่อนำมาแก้ไขภายหลัง ก็จะทำได้สะดวกไม่ยุ่งยากเหมือนกับการตั้งชื่อตัวแปรที่ไม่สื่อความหมาย

ในการตั้งชื่อตัวแปรนั้นเรามีกฎเกณฑ์ที่ต้องเข้าใจดังนี้

1.ตัวแปรต้องต้นด้วยอักษร หรือเส้นขีดล่าง (_) เท่านั้น แล้วตามด้วยตัวอักษร,ตัวเลข,เครื่องหมาย

2.ห้ามเว้นวรรคในชื่อตัวแปร และในตัวแปรห้ามมีเครื่องหมายต่างๆเช่น $, %, *, @, +, - เป็นต้น

3.ตัวแปรหนึ่งตัวความยาวไม่เกิน 63 ตัวอักษร (ถ้าเกินกว่านั้น Delphi จะตัดส่วนที่เกินออกไปโดยอัตโนมัติ)

4. ตัวอักษรพิมพ์ใหญ่ พิมพ์เล็กมีความแตกต่างกัน เพราะฉะนั้นตัวแปร Name ,name ,NAME จึงถือเป็นตัวแปรคนละตัวกัน

5. ตัวแปรที่ตั้งขึ้นห้ามตรงกับคำที่สงวนไว้ (Reserve Word)เช่น begin, end, if, function, procedure เป็นต้น

2.2.3.2 ตัวอย่างการตั้งชื่อตัวแปรที่ดี

การตั้งชื่อตัวแปรที่ดีนั้นเราจะตั้งชื่ออย่างไรก็ได้ให้ถูกต้องตามกฎการตั้งชื่อที่ได้อธิบายจากหัวข้อก่อนหน้านี้ แต่แม้จะตั้งชื่อได้อย่างถูกต้องก็ใช่ว่าเป็นการตั้งชื่อตัวแปรที่ดี

การจั้งชื่อตัวแปรที่ดีนั้นต้องตั้งชื่อให้สื่อความหมายคือ เลือกชื่อที่มีความหมายสอดคล้องกับสิ่งที่จะจัดเก็บ ซึ่งชื่อตัวแปรที่ดีต้องสามารถคาดเดาได้ว่ามีหน้าที่อะไร เช่น Name ก็น่าจะหมายความว่าเก็บชื่อคนซึ่งเป็นตัวแปรชนิดข้อความ,ส่วนตัวแปร Age ก็น่าจะเก็บอายุ ซึ่งน่าจะเป็นตัวแปรชนิดตัวเลข เป็นต้น.

การตั้งชื่อตัวแปรมีความสำคัญมาก เพราะเมื่อใดที่เราเริ่มเขียนโปรแกรมยาวขึ้น ซับซ้อนขึ้น ถ้าเราตั้งชื่อไม่สอดคล้องแล้ว ย่อมทำให้เกิดความสับสนขึ้นในภายหลัง ทำให้การปรับปรุงแก้ไขโปรแกรมที่เขียนขึ้นทำได้ยาก เพราะสับสนกับตัวแปรที่ตั้งชื่อไว้ เมื่อประกาศตัวแปรได้แล้วเราสามารถกำหนดค่าให้กับตัวแปร ซึ่งก็คือข้อมูลที่เป็นชนิดเดียวกันกับตัวแปรนั่นเอง ทำให้การปรับปรุงแก้ไขโปรแกรมที่เขียนขึ้นทำได้ยาก เพราะสับสนกับตัวแปรที่ตั้งชื่อไว้ซึ่งมีข้อยกเว้นบางกรณีที่เราสามารถกำหนดข้อมูลคนละชนิดกับชนิดของตัวแปรได้ เช่น กำหนดค่า 1.00 ให้กับตัวแปรจำนวนเต็ม(แต่ในการเก็บข้อมูลนั้นจะเก็บเป็น 1) เป็นต้น ซึ่งจะเห็นได้ว่าตัวแปรนั้นจะเป็นชนิดตัวเลขแต่ถ้าตัวแปรเก็บตัวอักษร เรากำหนดค่าที่เป็นตัวเลขไปให้ก็จะเกิดข้อผิดพลาดขึ้น

2.2.4 ชนิดข้อมูลที่สร้างจากข้อมูลพื้นฐาน

เราสามารถผสมผสาน หรือเพิ่มเติมความสามารถให้กับข้อมูลชนิดพื้นฐานให้กลายเป็นข้อมูลชนิดใหม่ขึ้นมาได้ ซึ่งมีให้ใช้งานกันหลายรูปแบบ

2.2.4.1 ข้อมูลชนิด Enumerated

Enumerated คือชนิดข้อมูลที่เก็บเป็นชุด ซึ่งมีจำนวนข้อมูลที่เก็บในชุดนั้นแน่นอน ทำให้สะดวกต่อการอ้างอิงถึงข้อมูลที่เก็บอยู่ภายใน โดยมีรูปแบบของการประกาศตัวแปรชนิด Enumerated ดังนี้

2.2.4.2 ข้อมูลชนิด SubRange

SubRange คือชนิดข้อมูลที่มีการกำหนดค่าของข้อมูลเป็นช่วง โดยต้องกำหนดค่าต่ำสุด และค่าสูงสุดของข้อมูลช่วงนั้น ซึ่งชนิดข้อมูลที่จะนำมาใช้กับ SubRange ได้นั้นต้องเป็นชนิด Boolean, Char, Integer หรือ Enumerated ซึ่งเป็นชนิดข้อมูลที่มีจำนวนข้อมูลอยู่แน่นอน

2.2.4.3 ข้อมูลชนิด Array

อาร์เรย์ (Array) เป็นรูปแบบการจัดเก็บข้อมูลแบบเป็นชุด ซึ่งข้อมูลแต่ละตัวในอาร์เรย์ต้องเป็นข้อมูลชนิดเดียวกัน โดยจะใช้อินเด็กซ์ (Index) ในการอ้างอิงถึงข้อมูลแต่ละตัวในอาร์เรย์หรืออาจจะกำหนดชนิดข้อมูลก่อน แล้วค่อยประกาศตัวแปรอาร์เรย์ก็ได้

2.2.4.4 Dynamic Array

ในบางครั้งการทำงานกับอาร์เรย์เราอาจจะพบปัญหาต่อไปนี้

1.ยากที่จะกำหนดขนาดของอาร์เรย์ : เช่นการสร้างอาร์เรย์ที่เก็บรายชื่อพนักงานบริษัทแต่ไม่ทราบว่ามีพนักงานกี่คน ซึ่งมีโอกาสจะเพิ่ม หรือลดได้ตลอด

2.การประกาศหาอาร์เรย์ขนาดใหญ่ : จากปัญหาข้อที่ 1 อาจทำให้เราแก้ปัญหาโดยประกาศอาร์เรย์ที่ใหญ่ๆไปเลย ซึ่งทำให้มีการใช้พื้นที่ไม่คุ้มค่า เพราะมีหน่วยความจำที่จองไว้มากแต่ไม่ได้ใช้

3.ประกาศอาร์เรย์ขนาดใหญ่เอาไว้ แต่ไม่ได้ถูกใช้งานทันที : แม้บางครั้งต้องใช้อาร์เรย์ขนาดใหญ่ก็จริง แต่กว่าจะได้ใช้งานก็อาจจะเป็นการทำงานในช่วงท้ายๆของโปรแกรม ทำให้หน่วยความจำที่จองไว้ตั้งแต่ต้นไม่ได้นำมาใช้งานทันที เป็นการสิ้นเปลืองทรัพยากรของระบบโดยที่ไม่จำเป็น

จากปัญหาข้างต้นทำให้มีการคิดไดนามิกอาย์เรย์ (Dynamic Array) ขึ้นมา โดยจะเป็นอาร์เรย์ที่มีคุณสมบัติพิเศษคือ ไม่จำเป็นต้องระบุขนาดในตอนประกาศ แต่สามารถเปลี่ยนขนาดอาร์เรย์ได้ในขณะที่แอพพลิเคชั่นทำงาน ซึ่งเราต้องเขียนโค้ดจัดการเอง

ในการใช้งานเราจะประกาศตัวแปรแบบอาร์เรย์เอาไว้โดยไม่ระบุขนาด และจะใช้การระบุขนาดของอาร์เรย์ด้วยฟังก์ชัน SetLength ส่วนอินเด็กซ์ของไดนามิกซ์อาร์เรย์นั้นจะนับเริ่มตั้งแต่ 0 ขึ้นไป

2.2.4.5 ข้อมูลชนิด Set

Set คือกลุ่มของข้อมูลที่มีสมาชิกเท่าไรก็ได้ โดยไม่สนใจลำดับของสมาชิกที่อยู่ภายใน(สมาชิกที่อยู่ภายในอาจจะเป็นข้องมูลแบบ Enumerated หรือ SubRange ก็ได้)อีกทั้งถ้าสมาชิกนั้นซ้ำกันก็ให้ถือว่าเป็นตัวเดียวกันหรือจะกำหนดชนิดข้อมูลก่อน แล้วค่อยประกาศ และใช้งานตัวแปรชนิด Set ก็ได้

2.2.4.6 ข้อมูลชนิด Record

Record เป็นชนิดของข้อมูลที่สามารถเก็บข้อมูลชนิดอื่นๆ ไว้ภายในได้ โดยเราเรียกข้อมูลแต่ละตัวที่อยู่ใน Record ว่าฟิลด์ (Field) ก่อนจะใช้งาน Record ได้นั้นเราจะต้องประกาศตัวแปรชนิด Record เสียก่อน หรือกำหนดชนิดข้อมูลก่อนแล้วค่อยประกาศตัวแปรก็ได้

2.2.4.7 ข้อมูลชนิด Object

Object คือข้อมูลชนิดที่ใช้สร้างออบเจ็กต์ หรือคอมโพเนนท์ของ Delphi ซึ่งในการประกาศตัวแปรชนิด Object เราจะต้องมีส่วนของพร็อพเพอร์ตี้ เมธอดของออบเจ็กต์ด้วย ซึ่งจะคล้ายกับการประกาศตัวแปรแบบ Record

2.2.5 การใช้งานคงที่

ในบางครั้งเราอาจจะต้องใช้ข้อมูลค่าหนึ่งซึ่งมีค่าเหมือนเดิมไปตลอดโดยไม่เปลี่ยนแปลง เราเรียกข้อมูลตัวนั้นๆ ว่าค่าคงที่ (Constant) ค่าคงที่นั้นสามารถกำหนดให้เป็นข้อมูลชนิดใดก็ได้ ไม่จำเป็นต้องเป็นตัวเลข หรือตัวอักษรเท่านั้น

สำหรับชนิดของค่าคงที่นั้น Delphiจะกำหนดชนิดให้เองโดยอัตโนมัติ โดยพิจารณาว่าค่าที่กำหนดให้นั้นอยู่ในช่วงใด ดังตาราง

ขอบเขตค่าคงที่(เลขฐานสิบหก)	ขอบเขตของค่าคงที่(เลขฐานสิบ)	ชนิดที่กำหนด
-$8000000000000000..-$80000001	-2^63..-2147483649	Int64
-$80000000..$8001	-2147483648..-32769	Integer
-$8000..-$81	-32768..-129	Smallint
-$80..-1	-128..-1	Shortint
0..$7F	0..127	0..127
$80..$FF	128..255	Byte
$0100..$7FFF	256..32767	0..32767
$8000..$FFFF	32798..65535	Word
$10000..$7FFFFFFF	65536..2147483647	0..2147483647
$80000000..$FFFFFFFF	2147483648..4294967295	Cardinal
$100000000..$7FFFFFFFFFFFFFFF	4294967296..2^63-1	Int64

ตารางที่ 2.3 แสดงค่าคงที่ต่างๆ ในโปรแกรม Delphi 7

2.2.6 Expression และ Statement

หากจะสังเกตการณ์เขียนโปรแกรมก็จะเห็นว่า เราต้องประกาศตัวแปร กำหนดค่าให้กับตัวแปร และมีการทำงานโดยการสั่งให้ตัวแปรเก็บค่า หรืออ่านค่าในตัวแปรนั้น ซึ่งเราจะเขียนเป็นประโยคคำสั่ง จะเรียกประโยคคำสั่งแต่ละประโยคนี้ว่า Statement ซึ่งเราจะใช้เราจะใช้เครื่องหมาย ; แทนการจบประโยคคำสั่งต่างๆในประโยคคำสั่งบางคำสั่งอาจจะประกอบด้วยประโยคย่อยๆโดยเฉพาะประโยคที่มีการคำนวณหรือมีการกำหนดค่าให้กับตัวแปร ซึ่งเราเรียกประโยคย่อยนี้ว่า Expression หรือจะเป็นประโยคการทำงานที่เกี่ยวกับการตัดสินใจ

2.2.7 ตัวดำเนินการใน Delphi 7

ในการทำงานของโปรแกรมเรามักจะมีการคำนวณค่าตัวแปร การเปรียบเทียบตัวแปร หรือการกระทำอื่นๆที่ใช้งานค่าที่เก็บอยู่ในตัวแปรนั้น ซึ่งเราจะใช้สิ่งที่เรียกว่า ตัวดำเนินการ (Operator)ในการกระทำนั้น เราแบ่งตัวดำเนินการได้เป็น 4 ประเภท ได้แก่

2.2.7.1 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ หรือ Comparison Operator นี้ทำหน้าที่เปรียบเทียบค่าของตัวแปร 2 ค่า ในลักษณะมากกว่า หรือน้อยกว่า โดยจะให้ผลลัพธ์เป็นค่าจริง (True) หรือเท็จ (False)

2.2.7.2 ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการทางคณิตศาสตร์ หรือ Arithmetic Operator ก็คือ เครื่องหมายที่เราใช้คำนวณทางคณิตศาสตร์ต่าง ๆ

2.2.7.3 ตัวดำเนินการด้านตรรกศาสตร์

ตัวดำเนินการทางตรรกศาสตร์ หรือ Logical Operator เป็นตัวดำเนินการที่ใช้เปรียบเทียบค่าของตัวแปร 2 ค่าในลักษณะทางตรรกศาสตร์

2.2.7.4 ตัวดำเนินการเกี่ยวกับเซต

ตัวดำเนินการชนิดนี้จะกระทำกับข้อมูลชนิด Set หรือ SubRange

2.2.7.5 ลำดับความสำคัญของตัวดำเนินการ

ใน Expression ที่มีตัวดำเนินการหลายตัวปนกันอยู่นั้น เราอาจจะสับสนว่าจะเริ่มต้นทำงานกับตัวดำเนินการตัวใดก่อนตัวใดหลังนั้น ให้ใช้หลักพิจารณาดังต่อไปนี้

- ให้ดูว่ามีวงเล็บครอบอยู่หรือไม่ ถ้ามีให้ทำงานกับตัวดำเนินการที่อยู่ในวงเล็บในสุดก่อนแล้วค่อยไล่ออกมาหาวงเล็บภายนอก

- ถ้ามีวงเล็บครอบอยู่ในระดับเดียวกัน ให้เรียงความสำคัญโดยเรียงจากซ้ายไปขวา

- ถ้ามีตัวดำเนินการมากกว่า 2 ตัวอยู่ในวงเล็บเดียวกันให้เรียงลำดับความสำคัญ

2.2.8 การเขียนคำอธิบายโปรแกรม

ในการเขียนโปรแกรมไม่ว่าจะใช้ภาษาใดๆก็ตาม จะมีสิ่งที่เรียกว่า Comment หรือคำอธิบายโปรแกรมมาด้วย เพราะมันเป็นเครื่องมือชิ้นสำคัญที่ผู้เขียนโปรแกรมจะเขียนเพิ่มเติมลงในโปรแกรมเพื่อใส่คำอธิบายของสิ่งที่เขียนขึ้น โดยคำอธิบายนั้นจะไม่ถือเป็นคำสั่งในการทำงาน

สำหรับการเขียนคำอธิบายโปรแกรมใน Delphi นั้นทำได้ 3 รูปแบบ ได้แก่

1.ใช้เครื่องหมาย //

2.ใช้เครื่องหมาย /* กับ */ ครอบข้อความ

3.ใช้เครื่องหมาย { กับ } ครอบข้อความ

คำอธิบายโปรแกรมนั้นช่วยให้โปรแกรมอ่านง่านขึ้น ทำให้การปรับปรุงแก้ไขในภายหลังทำได้ง่ายกว่าไม่ได้เขียนไว้ เพราะเป็นไปได้ที่เราเองจะไม่ได้แก้ไขในสิ่งที่ได้เขียนขึ้น

2.2.9 สิ่งที่ควรคำนึงเมื่อลงมือเขียนโปรแกรม

หลังจากที่เราได้รู้จักกับหลักการเบื้องต้นในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชันไปหลายรูปแบบหลายตัวอย่างแล้ว ตอนนี้เราจะมาวางรากฐานที่มีความสำคัญสำหรับการเขียนปีแกรมในอนาคตดังนี้

2.2.9.1 ลักษณะการเขียนโปแกรมที่ดี

การเขียนโปรแกรมที่ดีคงเป็นเพียงแค่การเขียนโปรแกรมให้ทำงานได้ถูกต้องสมบูรณ์เท่านั้น หากแต่ยังต้องประกอบด้วยลักษณะบางประการที่จะช่วยเสริมสร้างให้โปรแกรมที่เขียนมีคุณค่าขึ้นอีก

- เขียนโฟลว์ชาร์ตให้ได้ก่อนลงมือเขียนโปรแกรม : เพราะจะเป็นการทบทวนความคิดของเราให้ถูกต้องซึ่งจะช่วยลดเวลา และความสับสนไปได้มาก

- เขียนคำอธิบายประกอบเสมอ : ไม่จำเป็นต้องเขียนทุกคำสั่ง แต่ควรเขียนกำกับ ณ จุดที่สำคัญ โดยแสดงแนวคิดที่มา หรืออธิบายการทำงานในจุดนั้น

- การตั้งชื่อ Identifier ต้องสื่อความหมาย : เพราะจะช่วยลดความสับสนไปได้มาก ควรเปลี่ยนชื่อจากชื่อดีฟอลต์ที่ Delphi ตั้งมาให้ด้วย

- การใช้ Indent หรือ Tab แสดงความเป็นบล็อกของโค้ด : จะช่วยให้โค้ดอ่านง่าย ตรวจสอบแก้ไข ปรับปรุงได้ง่ายกว่า

จะเห็นได้ว่าลักษณะโปรแกรมที่ดีนั้นจำเป็นต้องอ่านแล้วทำความเข้าใจง่ายด้วย เพราะจะช่วยให้การแก้ไขในภายหลังทั้งจากผู้ใช้งานเอง หรือบุคคลอื่นที่เข้ามามีส่วนร่วมในการพัฒนาจะสามารถต่อติดได้อย่างรวดเร็ว ทำให้ผลงานออกมาเป็นที่น่าพอใจ

2.2.9.2 ลักษณะการเขียนโปรแกรมที่ไม่ดี

ตัวอย่างที่ไม่น่าจะเกิดขึ้นในการเขียนโปรแกรมดูบ้าง ทั้งนี้เพื่อที่จะเป็นการดักเอาไว้แต่ต้นก่อนที่มันจะกลายเป็นอุปนิสัยที่แก้ไขได้ยาก

- เขียนไปคิดไป : แบบนี้เป็นบ่อยสำหรับมือใหม่ แต่พอเจอโจทย์ยากๆ การทำเช่นนี้จะทำให้ไม่สามารถแก้ไขปัญหาที่ซับซ้อน

- ไม่มีคำอธิบายประกอบในจุดที่สำคัญ : ทำให้อ่านโปรแกรมในภายหลังยาก แม้ว่าจะเป็นคนเขียนขึ้นมาเองก็ตาม

- ใช้แต่ชื่อดีฟอลต์ที่ Delphi สร้างมาให้ : แม้จะไม่ใช่เรื่องผิด แต่ก็ทำให้โปรแกรมเราอ่านยากและมีปัญหาในภายหลัง

2.2.10 การควบคุมทิศทางการทำงานของโปรแกรม

ทดลองเขียนโปรแกรมจากบทที่ผ่านๆมาเราจะเห็นว่า โปรแกรมนั้นเป็นโปรแกรมที่ทำงานง่ายๆ แต่ในความเป็นจริงการทำงานของแอพพลิเคชันนั้นซับซ้อนกว่าในตัวอย่างมาก ดังนั้น จึงจำเป็นต้องอาศัยความสามารถของตัวภาษาในการเขียนโปรแกรมเพื่อทำให้โปรแกรมสามารถทำงานได้ตามรูปแบบหรือเงื่อนไขที่ต้องการให้ได้

ในการเขียนโปรแกรมก็คือ การจำลองตัวเราลงไปเพื่อจัดการกับสถานการณ์ต่างๆซึ่งเราต้องมีข้อมูลประกอบในการพิจารณา ถ้าข้อมูลบอกเราอย่างหนึ่งเราก็ต้องทำงานแบบหนึ่ง แต่ถ้าบอกข้อมูลอีกอย่างเราก็คงต้องทำงานอีกแบบที่แตกต่างกัน ซึ่งการที่ต้องจัดการกับสถานการณ์ต่างๆนี้เราเรียกมันว่า การควบคุมทิศทางของโปรแกรม (Control Flow)

การควบคุมทิศทางของโปรแกรม มีอยู่ด้วยกัน 2 รูปแบบ ได้แก่

- การตัดสินใจ : การตัดสินใจก็คือ การเลือกตัวเลือกจากตัวเลือกที่มีอยู่
- การทำงานแบบวนซ้ำ : คือการทำงานแบบเดิมวนซ้ำอีกครั้ง จนกว่าจะครบจำนวน หรือถูกกำหนด หยุดออกมาจากการวนซ้ำ

ในการควบคุมทิศทางไม่ว่าจะเป็นการตัดสินใจ หรือการทำงานแบบวนซ้ำ เราจะต้องอาศัยการพิจารณาข้อมูลที่มีอยู่ประกอบในการควบคุม

2.2.10.1 การตัดสินใจ

การตัดสินใจ (Decision) เป็นการเลือกทางใดทางหนึ่งจากตัวเลือกที่มีให้ โดยการเลือกนั้นจะพิจารณาจากตัวแปรที่เรากำหนดให้เป็นเงื่อนไขในการเลือก ซึ่งเราจะใช้ตัวดำเนินการแบบต่างๆ มาใช้ในการตัดสินใจเลือก

- การตัดสินใจเลือกหนึ่งตัวเลือก จาก 2 ตัวเลือกที่มีมาให้ การตัดสินใจแบบนี้จะมีตัวเลือกมาให้ 2 ตัวเลือกให้เราเลือกเพียงหนึ่งเดียว ซึ่งก็จะเหมือนกับเราต้องตอบคำถามประเภท Yes/No นั่นเอง
- การตัดสินใจเลือกหนึ่งตัวเลือก จากตัวเลือกที่มีมาให้มากกว่า 2 ตัวเลือก การตัดสินใจแบบนี้ จะมีตัวเลือกให้มามากกว่า 2 ตัวเลือกขึ้นไป ซึ่งเราต้องเลือกเพียงตัวเลือกเดียว ซึ่งก็จะเหมือนกับนักเรียนกำลังทำข้อสอบที่ต้องกาตัวเลือกเพียงตัวเลือกเดียวเท่านั้น

2.2.10.2 การทำงานแบบวนซ้ำ

การทำงานแบบวนซ้ำ (Iteration) เป็นการทำงานแบบซ้ำไปซ้ำมาตามเงื่อนไขที่ได้กำหนดไว้ ซึ่งเราแบ่งการทำงานแบบวนซ้ำได้2ประเภทได้แก่

- การวนซ้ำแบบมีจำนวนรอบที่แน่นอน เป็นการที่เรากำหนดรอบการทำงานวนซ้ำไว้แน่นอนแล้วว่าต้องทำงานซ้ำไปซ้ำมากี่รอบซึ่งเราต้องมีตัวแปรหนึ่งตัวมาทำหน้าที่นับจำนวนรอบของการทำงานว่าครบรอบแล้วหรือไม่ถ้าครบแล้วก็ถือว่าสิ้นสุดการทำงาน

2.2.11 การใช้งานif...then...else

if...then...else จะเป็นการทำงานที่มีการเลือกหนึ่งตัวเลือกจากตัวเลือก 2 ตัวที่มีอยู่ ซึ่งเราจะต้องมีตัวแปรหนึ่งตัวที่จะถูกตรวจสอบว่ามีค่าเป็นจริงหรือเท็จ ถ้าเป็นจริง (TRUE) ก็จะต้องเลือกตัวเลือกแรกซึ่งอยู่หลังคำว่า then แต่ถ้าตรวจสอบว่าเป็นเท็จ ก็จะต้องเลือกตัวเลือกอีกตัวซึ่งอยู่หลังคำว่า else

2.2.11 .1 การใช้งาน if...then...else เบื้องต้น

เราจะเขียนโปรแกรมโดยใช้งาย if...then...elseอย่างง่ายโดยจำลองสถานการณ์ในชีวิตประจำวันในลักษณะของคำถามสำรวจความคิดเห็นเกี่ยวกับการสำรวจความชอบของคุณว่าคุณชอบฟังดนตรีหรือไม่ ถ้าชอบไปรับประทานอาหารนอกบ้านหรือไม่

2.2.11.2 การใช้งาน if...then...else ทำงานซ้อนกัน

ในบางครั้งเงื่อนไขการเลือกทีละตัวอาจจะมีลักษณะที่ต่อเนื่องกันไป เช่น ถ้าเราถามว่าคุณเป็นผู้ชายหรือผู้หญิง ถ้าหากตอบว่าเป็นชาย เราอาจจะถามต่อว่าอายุมากกว่า 18 ปีหรือไม่ (แต่ถ้าตอบว่าผู้หญิงเราก็อาจจะไม่ถามต่อ หรือถามคำถามอื่นๆก็ได้)

2.2.11.3 การใช้งาน if...then...else แบบต่อเนื่อง

ในบางครั้งเราต้องการตรวจสอบเงื่อนไขไปเรื่อยๆ ทีละเงื่อนไขว่าจริงหรือไม่ ซึ่งก็จะมีลักษณะการใช้งาน if...then...else ต่อเนื่องกันไป ซึ่งจะซ้อนเข้าไปกี่ชั้นก็ได้

2.2.12 การใช้งาน case...of

เราจะใช้ case...of ในการเลือกหนึ่งตัวเลือกจากตัวเลือกที่มีหลายตัวเลือก (คือมากกว่า 2 ตัวขึ้นไป) โดยเราจะใช้ตัวแปรหนึ่งตัวตรวจสอบว่า ตรงกับตัวเลือกใด โดยที่ตัวเลือกแต่ละตัวเราก็จะกำหนดให้มีการทำงานที่แตกต่างกันด้วย

2.2.12.1 การใช้งาน while...do

While...do เป็นการทำงานที่เราต้องการให้มีการวนซ้ำไปเรื่อยๆ โดยทุกๆครั้งที่จะวนซ้ำใหม่ให้มีการตรวจสอบเงื่อนไขการวนซ้ำก่อนทุกครั้ง ถ้าเงื่อนไขเป็นจริง (True) ก็วนซ้ำต่อไป แต่ถ้าเป็นเท็จ (False)ก็ให้หยุดการ วนซ้ำ

2.2.12.2 การใช้ repeat...until

การทำงานของ repeat...until นั้นจะแตกต่างจาก while...do ตรงที่การหยุดการวนซ้ำจะหยุด

2.2.12.3 การใช้งาน for...to...do

เราใช้งาน for...to...do เพื่อให้แอพพลิเคชันทำงานซ้ำด้วยจำนวนรอบที่แน่นอน ซึ่งจะใช้ตัวแปรหนึ่งตัวทำหน้าที่ในการนับว่าทำซ้ำครบรอบหรือไม่ โดยที่แต่ละครั้งของการวนรอบตัวแปรที่นับรอบนั้นจะเพิ่มค่าขึ้นทีละหนึ่งๆ และเพิ่มไปเรื่อยๆจนกว่าจะมากกว่าค่าสุดท้าย

2.2.12.4 การใช้งาน for…downto…do

ปกติเราจะใช้การเพิ่มค่าให้กับตัวแปรในการนับรอบเข้าไปทีละ 1 ในทุกๆครั้งของการวนซ้ำ แต่เรายังสามารถใช้การลดค่าตัวแปรในการนับรอบลงทีละ 1 ได้เช่นเดียวกัน โดยการใช้คำสั่ง for…downto…do

2.2.12.5 การใช้งาน for…to…do แบบซ้อนกัน

ในการตรวจสอบเงื่อนไขที่ซับซ้อนนั้นเราจะมีการตรวจสอบเงื่อนไขที่ซ้อนๆกันเข้ามาช่วย เช่นเดียวกับในการทำงานของการวนซ้ำเราก็สามารถวนซ้ำแบบซ้อนกันได้ ตัวอย่างเช่น การวนซ้ำเพื่อเติมข้อมูลลงในตาราง(คล้ายกับตัวแปร 2 มิติ) เราจะต้องวนซ้ำโดยเข้าข้อมูลจากตารางทีละแถว ซึ่งแต่ละแถวก็จะต้องเข้าถึงทีละคอลัมน์ ซึ่งหมายความว่าเราต้องเริ่มวนซ้ำจากคอลัมน์ก่อน เมื่อหมดหนึ่งแถวแล้วจึงวนซ้ำในแต่ละคอลัมน์ของแถวถัดไปจนครบทุกแถว

2.2.13 การกระโดดออกจากการวนซ้ำ

การทำงานแบบวนซ้ำนั้นปกติเราจะใช้การตรวจสอบเงื่อนไขการวนซ้ำ หรือตรวจสอบตัวแปรที่ใช้นับรอบการวนซ้ำเพื่อพิจารณาว่าจะทำงานวนซ้ำต่อไปหรือไม่ แต่บ่อยครั้งที่เราคงต้องการจบการทำงานแบบวนซ้ำด้วยเงื่อนไขพิเศษ ซึ่งทำได้โดยการใช้คำสั่ง break

เพื่อความเข้าใจจะสร้างแอพพลิเคชันง่ายๆให้มีการสุ่มตัวเลข 20 ครั้ง ทุกๆครั้งที่สุ่มตัวเลขระหว่าง 0 ถึง 9 ขึ้นมาแล้วนำมาบวกกับตัวเลขที่ได้จากการสุ่มครั้งที่ผ่านๆมา ถ้าผลบวกเกิน 70 ให้กระโดดออกจากการวนซ้ำ แม้ว่าจะยังสุ่มไม่ถึง 20 ครั้งก็ตาม ซึ่งจะเขียนโค้ดสำหรับการวนซ้ำได้ทุกรูปแบบ

2.2.14 การสร้างและใช้งานโปรแกรมย่อย

ในโลกของความเป็นจริงนั้น เราจะเห็นได้ว่าในบริษัทหนึ่งจะมีการแบ่งหน้าที่ออกเป็นฝ่ายต่างๆ เช่น ฝ่ายผลิต,ฝ่ายบัญชี,ฝ่ายการตลาด และบุคคล ซึ่งเหตุผลที่ต้องแบ่งเพราะงานในบริษัทนั้นมีหลายส่วน แต่ละส่วนก็มีความซับซ้อน อีกทั้งงานของแต่ละฝ่ายก็เป็นอิสระจากกัน แต่ก็ยังมีการแลกเปลี่ยนข้อมูลระหว่างกันในการทำงานด้วย เช่น ฝ่ายผลิตส่งข้อมูลวัตถุดิบให้ฝ่ายจัดซื้อ, ฝ่ายขายส่งรายงานการขายแจ้งให้ผู้บริหารทราบ เป็นต้น

ในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชันก็มีลักษณะคล้ายกัน จำเป็นอย่างยิ่งที่เราต้องมีการแบ่งงานออกเป็นส่วนๆ ซึ่งแต่ละส่วนก็จะมีหน้าที่เฉพาะของตนเองเป็นอิสระจากกัน มีการสื่อสารแลกเปลี่ยนข้อมูลระหว่างกัน ซึ่งเราเรียกหน่วยย่อยๆของโปรแกรมที่แยกจากกันนี้ว่า โปรแกรมย่อย

2.2.15 รู้จักกับโปรแกรมย่อย

โปรแกรมย่อย(Sub Program) เป็นส่วนของโปรแกรมที่ถูกสร้างขึ้นมาเพื่อทำงานเล็กๆชิ้นหนึ่งให้สำเร็จ ซึ่งโปรแกรมย่อยนี้สามารถถูกเรียกใช้งานซ้ำกันได้จากหลายๆครั้งตัวอย่างเช่นในการสร้างบ้านซึ่ง

ถือว่าเป็นโปรแกรมหลัก อาจจะมีการทาสีเป็นโปรแกรมย่อย ซึ่งเราสามารถเรียกใช้การทาสีได้กับห้องต่างๆได้ทั้งห้องต่างๆในบ้าน นอกบ้าน รวมถึงทาสีทับสีเดิมได้ด้วย โดยที่เราไม่ต้องสนใจว่าโปรแกรมย่อยของการทาสีจะต้องประกอบด้วยขั้นตอนย่อยๆภายในอย่างไรบ้าง

ข้อดีของโปรแกรมย่อยคือ ลดความซ้ำซ้อนในการเขียนโค้ด โดยจะเก็บชุดคำสั่งที่ต้องถูกเรียกใช้ซ้ำๆเข้าไว้ด้วยกัน ทำให้ไม่ต้องเขียนโค้ดชุดเดิมซ้ำอยู่เรื่อยๆ และยังช่วยให้การแก้ไขในภายหลังทำได้ง่าย โดยที่เราสามารถแก้ไขเฉพาะโปรแกรมย่อยไม่ต้องเข้าไปแก้ไขที่โปรแกรมหลัก ทำให้แก้ไขได้ง่ายขึ้น

2.2.16 สร้างและใช้งาน Procedure

โพรซีเจอร์(Procedure) คือโปรแกรมย่อยที่ทำงานเสร็จแล้วจะไม่มีการคืนค่าใดๆ กลับมายังผู้ที่เรียกใช้งานโพรซีเจอร์นั้น เราจึงใช้โพรซีเจอร์ในงานที่ไม่สนใจ หรือไม่ต้องการผลลัพธ์ที่ได้จากการทำงานของโพรซีเจอร์ เพียงแค่สั่งให้โพรซีเจอร์ทำงานเท่านั้นก็เพียงพอ

2.2.17 สร้างและใช้งาน Function

ฟังก์ชัน(Function) คือโปรแกรมย่อยที่ถูกเรียกใช้งาน และภายหลังที่ทำงานเสร็จจะคืนค่ากลับมา (Return) ให้กับผู้ที่เรียกฟังก์ชันนั้นได้นำไปใช้ต่อ ดังนั้นการใช้งานฟังก์ชันจึงมักจะสนใจที่ผลการทำงานที่ได้รับกลับมา ซึ่งจะหมายถึงการนำค่านั้นไปใช้ในคำสั่งถัดไปหลังจากเรียกใช้ฟังก์ชันนั้นหรือเราอาจจะให้กับผู้ที่เรียกฟังก์ชันนั้นได้นำไปใช้ต่อ ดังนั้นการใช้งานฟังก์ชันจึงมักจะสนใจที่ผลการทำงานใช้ตัวแปรที่เรียกว่า Result เก็บค่าที่จะคืนจากฟังก์ชันก็ได้

2.2.18 ขอบเขตการใช้งาน (Scoping)

ในการเขียนโปรแกรมด้วย Delphi นั้นเราจะมีการตั้งชื่ออ็อบเจ็กซ์, ชื่อตัวแปร, ชื่อฟังก์ชัน, ชื่อโพรซีเจอร์, ชื่อเมธอดต่างๆกัน เราเหมารวมเรียกชื่อที่ตั้งนี้ว่า Identifier

สำหรับ Identifier แต่ละตัวที่ตั้งขึ้นมาก็จะมีขอบเขตการทำงานคือ สามารถใช้งานได้ในขอบเขตหนึ่งซึ่งก็ขึ้นอยู่กับการประกาศชื่อ Identifier นั้น ซึ่งใน Delphi 7 นั้น Identifier มีขอบเขตได้ดังนี้

- Local Scope เป็นขอบเขตที่อยู่ในโปรแกรมย่อยเดียวกัน (อยู่ในโพรซีเจอร์ หรือฟังก์ชันเดียวกัน)หรืออยู่ในอีเวนท์เดียวกัน

- Unit Scope เป็นขอบเขตที่อ้างอิงได้เมื่ออยู่ในยูนิตเดียวกัน อ้างอิงข้ามโปรแกรมย่อยในยูนิตเดียวกันได้ ซึ่งจะประกาศรายชื่อ Identifier นั้นไว้ในส่วน Implementation

- Global Scope เป็นขอบเขตที่อ้างอิงได้จากยูนิตอื่นๆซึ่งจะประกาศรายชื่อ Identifier นั้นไว้ในส่วนInterface

เพราะฉะนั้นเราสามารถตั้งชื่อ Identifier เช่น ชื่อของตัวแปรเหมือนกันได้ แต่นั่นหมายถึงเราต้องให้มันใช้งานได้ในขอบเขตการทำงานกันจึงจะไม่ทำให้การทำงานผิดพลาด

2.2.19 การส่งผ่านตัวแปรเมื่อเรียกใช้งานโปรแกรมย่อย

ที่ผ่านมาไม่ว่าจะเป็นการเรียกใช้งานโพรซีเจอร์หรือฟังก์ชันนั้นเราจะต้องส่งชุดข้อมูลไปทำงานในเรียกค่าข้อมูลที่ถูกส่งไปพารามิเตอร์ว่าอาร์กิวเมนต์(Argument)

สำหรับการเขียนโปรแกรมกับ Delphi 7 นั้นเราสามารถส่งผ่านค่าตัวแปรไปทำงานในโปรแกรมย่อยได้3แบบได้แก่

2.2.19.1 การส่งพารามิเตอร์แบบ Value การส่งพารามิเตอร์ มันจะก๊อปปี้ค่าที่ส่งไปไว้ในตัวแปรอีกตัวที่อยู่ในโปรแกรมย่อยเพราะฉะนั้นถ้าเรามีการเปลี่ยนแปลงค่าที่เก็บในตัวแปรใหม่ขณะที่มันทำงานในโปรแกรมย่อยก็จะไม่มีผลถึงค่าที่เก็บในโปรแกรมหลัก

2.2.19.2 การส่งพารามิเตอร์แบบ Reference การส่งพารามิเตอร์แบบนี้ จะตรงข้ามกันแบบ Value คือแทนที่จะส่งก๊อปปี้ค่าไปให้ แต่จะส่งแอดเดรส หรือตำแหน่งของหน่วยความจำที่เก็บค่าไปให้เพราะฉะนั้นถ้ามีการเปลี่ยนแปลงค่าในโปรแกรมย่อยย่อมมีผลถึงค่าที่เก็บในโปรแกรมหลัก

2.2.19.3 การส่งพารามิเตอร์แบบ Constant การส่งพารามิเตอร์แบบนี้ เมื่อส่งไปถึงโปรแกรมย่อยแล้ว จะไม่อนุญาตให้มีการแก้ไขค่าในพารามิเตอร์

จะเห็นว่าการส่งพารามิเตอร์แบบ Value นี้เหมือนกับเราถ่ายสำเนาเอกสารไว้ก่อนแล้วส่งไป (แต่ตัวจริงยังเก็บอยู่ที่เรา) เพราะฉะนั้นม่ว่าภายในโปรแกรมย่อยจะเปลี่ยนแปลงค่าในสำเนาอย่างไรก็ไม่มีผลกับตัวจริงนั่นเอง ซึ่งจะแตกต่างกันกับการส่งพารามิเตอร์แบบ Reference ที่เสมือนกับส่งเอกสารต้นฉบับจริงไปให้

2.2.20 เทคนิคสำหรับการเขียนโปรแกรมเพิ่มเติม

ในการเขียนโปรแกรมกับ Delphi 7 นั้นแม้ว่าเราจะเข้าใจเกี่ยวกับภาษาออบเจ็กต์ปาสคาลดีแล้ว แต่ก็ยังมีเทคนิคที่น่าศึกษาเอาไว้เพื่อให้การเขียนโปรแกรมทำได้อย่างสะดวก ผลงานออกมาก็น่าใช้งาน ดังที่จะกล่าวต่อไปนี้

2.2.20.1 เทคนิคการเรียงลำดับ Tab Order

ในการใช้งานแอพพลิเคชันร่วมกับคีย์บอร์ดนั้น ทุกครั้งที่เรากดปุ่มแท็บจะเห็นว่าแอพพลิเคชันจะเปลี่ยนการโฟกัสไปทำงานยังคอมโพเนนท์ตัวอื่นๆซึ่งมันจะกลายเป็นเรื่องชวนปวดหัวมากถ้าหากการเปลี่ยนแปลงไปโฟกัสมีลำดับที่ไม่เหมาะสม

ตัวอย่างเช่น ฟอร์มที่กรอกข้อมูลของผู้ใช้งาน เราต้องการให้กรอกชื่อก่อน ตามด้วยนามสกุล และอายุ จากนั้นจึงเป็นการเลือกว่าจะคลิกปุ่ม ตกลง และ ยกเลิก ซึ่งเราจะเรียงลำดับของการเปลี่ยนแปลงไปโฟกัสไปยังคอมโพเนนท์ต่างๆด้วยการกดปุ่ม Tab ไปเรื่อยๆ ว่า Tab Order ลำดับของ Tab order นั้นจะถูกกำหนดให้โดยอัตโนมัติตามลำดับของการคลิกเลือกคอมโพเนนต์เข้ามาในฟอร์มในตอนสร้างครั้งแรก คอมโพเนนต์ใดถูกนำมาวางก่อนก็จะถูกกำหนดลำดับให้ก่อน ถ้ามีการลบคอมโพเนนท์ออกไป Delphi ก็จะเรียงลำดับ Tab Order ให้ใหม่โดยอัตโนมัติ

2.2.20.2 เทคนิคการใช้งานดีฟอลต์พารามิเตอร์

บ่อยครั้งที่พบว่าการพารามิเตอร์ไปให้โปรแกรมย่อยนั้นจะมีเฉพาะค่าที่ใช้เป็นส่วนใหญ่กันเราก็สามารถกำหนดค่าที่ใช้บ่อยๆนั้นเป็นค่าดีฟอลต์ไปได้เลย ซึ่งถ้าผู้ใช้ไม่ได้กำหนดค่า ก็สามารถใช้ค่าดีฟอลต์นี้ทดแทนได้

2.2.20.3 เทคนิคการตั้งชื่อคอมโพเนนต์

จากตัวอย่างหลายๆบทที่ผ่านมาจะเห็นว่าผู้เขียนได้กำหนดชื่อให้คอมโพเนนต์ใหม่ โดยไม่ใช้ชื่อดีฟอลต์ Delphi กำหนดให้ ทั้งนี้เพราะต้องการกำหนดชื่อให้สื่อความหมาย และไม่ซ้ำกัน

หลักการที่จะขอแนะนำในการตั้งชื่อคอมโพเนนต์ก็คือ ตั้งชื่อให้สื่อความหมายใกล้เคียงกับสิ่งที่ต้องการจะสื่อสาร เช่น ตั้งชื่อปุ่มให้ผู้ใช้คลิกตกลงว่า btnOK หรือตั้งชื่อตัวเลียนแบบ Radio Button ที่เป็นตัวเลือกของผู้ชายว่า rbtMale เป็นต้น

จะเห็นว่าหลักการตั้งชื่อของคอมโพเนนต์ที่ได้แนะนำไปแล้วนั้นเราแบ่งส่วนของชื่อออกเป็น 2 ส่วน คือ ส่วนที่เป็นภาษาอังกฤษ 3 ตัวแรกที่บอกชนิดของคอมโพเนนท์ (เรียกสั้นๆว่า Prefix) กับส่วนที่เป็นชื่อที่สื่อความหมายของคอมโพเนนท์ตัวนั้น

2.2.21 การใช้งาน Built-in Function

ในการเขียนโปรแกรมจริงๆนั้นเราไม่จำเป็นที่จะต้องสร้างฟังก์ชัน หรือโพรซีเจอร์ขึ้นมาเองทั้งหมดทั้งนี้เพราะ Delphi 7 ได้จัดเตรียมฟังก์ชัน และโพรซีเจอร์ต่างๆที่จำเป็นแทบทุกรูปแบบเอาไว้ให้เลือกใช้งานกันอยู่แล้ว โดยฟังก์ชันสำเร็จรูปต่างๆ จะถูกแบ่งออกเป็นหมวดหมู่ และเก็บอยู่ในไฟล์ยูนิตมาตรฐานที่เรียกว่า RTL(Run-Time Library)ซึ่งมีดังนี้

ยูนิต System	เป็นยูนิตหลักของRTLเก็บรายละเอียดเกี่ยวกับระบบเช่น คลาสต่างๆ,ชนิดของข้อมูล,การจัดการหน่วยความจำเบื้องต้น และฟังก์ชันที่เกี่ยวกับการเขียนโปรแกรมข้ามแพล็ตฟอร์ม(ข้ามระหว่าง Windows กับลีนุกซ์)
ยูนิต SysInit	เป็นยูนิตที่เก็บรายละเอียดเกี่ยวกับฟังก์ชันที่จำเป็นต่อการทำงานของแอพพลิเคชันขณะที่แอพพลิเคชันเริ่มทำงาน (เรามักจะไม่ได้ใช้งานโดยตรง)
ยูนิต SysUtils	เป็นยูนิตที่เก็บฟังก์ชันอเนกประสงค์ต่างๆเช่น IntToStr,Format เป็นต้น ซึ่งครอบคลุมทั้งการสร้างแอพพลิเคชันบน Windows และลีนุกซ์
ยูนิต SysConst	เก็บค่าคงที่ตัวอักษรที่ถูกเรียกใช้งานโดยยูนิตต่างๆ
ยูนิต Math	เก็บฟังก์ชันที่เกี่ยวกับการคำนวณด้านคณิตศาสตร์
ยูนิต DateUtils	เก็บฟังก์ชันที่ช่วยจัดการเกี่ยวกับวันเดือนปี และเวลา
ยูนิต StrUtils	เป็นยูนิตที่เก็บฟังก์ชันที่ใช้จัดการข้อความ

ยูนิต StrUtils เป็นยูนิตที่เก็บฟังก์ชันที่ใช้จัดการข้อความ

ยูนิต ConvUtils เป็นยูนิตที่เก็บฟังก์ชันเกี่ยวกับการแปลงระหว่างหน่วยต่างๆ

ยูนิต StdConvs เป็นยูนิตที่เก็บฟังก์ชันเกี่ยวกับการแปลงระหว่างหน่วยต่างๆ

ยูนิต Types เป็นยูนิตที่เก็บฟังก์ชันเกี่ยวกับการจัดการข้อมูลชนิดต่างๆระหว่างกัน ซึ่งครอบคลุมทั้งชนิดข้อมูลบน Windows และลีนุกซ์

ยูนิต DelphiMM เป็นยูนิตที่เก็บฟังก์ชันที่ใช้จัดการเกี่ยวกับหน่วยความจำและ ShareMem

เราสามารถตรวจสอบได้ว่า Delphi 7 ได้มีการใช้งานยูนิตอะไรบ้าง เมื่อเราเริ่มสร้างโปรเจ็กต์โดยการคลิกที่ Code Explorer แล้วเลือกดูยูนิตที่ต้องการ ซึ่งยูนิตที่ใช้งานจะปรากฏอยู่หลังคำว่า uses ในส่วนของ Code Editor ถ้าเราต้องการใช้งานฟังก์ชันสำเร็จรูปจากยูนิตใดเพิ่มเติม ก็ให้เพิ่มต่อท้ายเข้าไปในส่วน uses ได้เลย จะเห็นว่ายูนิตสำคัญบางตัว เช่น ยูนิต System ไม่ได้บรรจุไว้ ซึ่งก็ไม่ต้องกังวลเพราะตอนที่คอมไพล์ Delphi จะเพิ่มเข้าไปให้โดยอัตโนมัติ(ถ้าเราไปเพิ่มเข้าไปเองอีก Delphi จะแจ้ง Error ให้ทราบ) ส่วนการตรวจสอบว่าฟังก์ชันที่เราต้องการนั้นเก็บอยู่ในยูนิตใดเราสามารถเรียกดูผ่านระบบช่วยเหลือ

2.2.21.1 กลุ่มฟังก์ชันด้านการแปลงชนิดข้อมูล

กลุ่มฟังก์ชันเหล่านี้จะทำการแปลงชนิดข้อมูลจากชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง ทั้งนี้เพื่อให้สามารถใช้งานข้ามประเภทได้ โดยที่ไม่เกิดข้อผิดพลาด

- StrToint แปลงข้อมูลจากสตริงให้เป็นเลขจำนวนเต็ม -
- IntToStr แปลงข้อมูลจากเลขจำนวนเต็มให้สตริง
- FloatToStr แปลงข้อมูลจากเลขทศนิยมให้เป็นสตริง
- DateToStr แปลงข้อมูลจากวันเดือนปีให้เป็นสตริง
- TimeToStr แปลงข้อมูลจากเวลาให้เป็นสตริง
- DateTimeToStr แปลงข้อมูลจากวันเวลาให้เป็นสตริง
- BoolToStr แปลงข้อมูลจากค่าตรรกศาสตร์ให้เป็นข้อความคือ True หรือ False
- BoolToInt แปลงข้อมูลจากค่าตรรกศาสตร์ให้เป็นเลขจำนวนเต็ม โดย True = 0 ส่วน False = -1

2.2.21.2 กลุ่มฟังก์ชันด้านคณิตศาสตร์

Delphi 7 เตรียมฟังก์ชันที่จำเป็นต่อการคำนวณด้านคณิตศาสตร์ไว้อย่างมากมาย โดยจะขอยกตัวอย่างเฉพาะฟังก์ชันที่ใช้งานบ่อยๆดังนี้

- RandomFrom เป็นการสุ่มค่าตัวเลข โดยกำหนดค่าเริ่มต้นที่จะสุ่มค่า
- RandomRange เป็นการสุ่มค่าตัวเลข โดยกำหนดขอบเขตของค่าที่เป็นไปได้

- Sin เป็นฟังก์ชันที่ใช้คำนวณค่า Sine โดยเรากำหนดค่ามุมเข้าไป
- Cos เป็นฟังก์ชันที่ใช้คำนวณค่า Cosine โดยเรากำหนดค่ามุมเข้าไป
- Floor รีเทิร์นค่าจำนวนเต็มที่มากที่สุดที่ใกล้เคียงเลขทศนิยมที่เรากำหนด
- Cell รีเทิร์นค่าจำนวนน้อยที่มากที่สุดที่ใกล้เคียงเลขทศนิยมที่เรากำหนด
- Int รีเทิร์นค่าเป็นส่วนที่เป็นเลขจำนวนเต็มจากเลขทศนิยมที่เรากำหนด
- Frac รีเทิร์นค่าเป็นส่วนที่เป็นเลขหลังจุดทศนิยมจากเลขทศนิยมที่กำหนด
- Exp รีเทิร์นค่า Exponential ของเลขที่กำหนด
- Min รีเทิร์นค่าที่น้อยที่สุดของเลข 2 จำนวนที่เรากำหนด
- Max รีเทิร์นค่าที่มากที่สุดของเลข 2 จำนวนที่เรากำหนด
- Mean รีเทิร์นค่าเฉลี่ย (ทางสถิติ) ของจำนวนที่เก็บอยู่ในอาร์เรย์
- StdDev รีเทิร์นค่าเบี่ยงเบนมาตรฐาน (ทางสถิติ) ของจำนวนที่เก็บอยู่ในอาร์เรย์
- Sum หาผลรวมของจำนวนที่เก็บอยู่ในอาร์เรย์
- Pi รีเทิร์นค่า Pi() ซึ่งเราจะใช้ในการคำนวณทางเรขาคณิต
- Sqr เป็นการหาค่าเลขยกกำลังสองเลขที่กำหนด
- Sqrt เป็นการหาค่ารากที่สองของเลขที่กำหนด
- Sign รีเทิร์นค่า –1ถ้าต่ำกว่า0,รีเทิร์นค่า1 ถ้ามากกว่า0,รีเทิร์นค่า0 ถ้าเท่ากับ 0
- CompareValue เป็นการเปรียบเทียบเลขทศนิยม 2 ค่า
- DivMod รีเทิร์นเลขจำนวนเต็มที่เป็นเศษจากการหาร

2.2.21.3 กลุ่มฟังก์ชันด้านสตริง

สำหรับการจัดการข้อความนั้น Delphi 7 ก็มีฟังก์ชันให้เลือกใช้มากมายซึ่งจะช่วยลดเวลาในการเขียนฟังก์ชันจัดการไปได้มาก

- CompareText เป็นการเปรียบเทียบข้อความ 2 ข้อความ
- CompareStr เป็นการเปรียบเทียบข้อความ 2 ข้อความ
- Concat เป็นการนำข้อความ 2 ข้อความมาต่อกัน
- Length เป็นความยาวของข้อความ
- PosEx เป็นการค้นหา Substring ใน String คือหาข้อความย่อยในข้อความ
- Trim เป็นการตัดข้อความ
- TrimLeft เป็นการตัดข้อความออก โดยเริ่มนับตัวอักษรที่ตัดออก

- TrimRight	เป็นการตัดข้อความออก โดยเริ่มนับตัวอักษรที่ตัดออกจากทางขวามือของตัวอักษร
UpperCase	เป็นการแปลงข้อความทั้งหมดให้เป็นตัวพิมพ์ใหญ่(ภาษาไทยไม่มีผล)
LowerCase	เป็นการแปลงข้อความทั้งหมดให้เป็นตัวพิมพ์เล็ก(ภาษาไทยไม่มีผล)
Val	ปืนการแปลงค่าจากเลขทั้งจำนวนเต็มและทศนิยมให้อยู่ในรูปของข้อความแล้วเก็บลงไปในตัวแปรสตริงที่กำหนด

2.2.21.4 กลุ่มฟังก์ชันเกี่ยวกับวันเวลา

วันเวลาเป็นข้อมูลที่มีการจัดการค่อนข้างลำบาก ดังนั้นเราจึงน่าจะรู้จักฟังก์ชันสำคัญๆในการจัดการเกี่ยวกับวันเดือนปี และเวลาดังต่อไปนี้

Date	เป็นการหาวันเดือนปีปัจจุบันตามนาฬิกาของเครื่อง
Time	เป็นการหาเวลาปัจจุบันตามนาฬิกาของเครื่อง
Now	รีเทิร์นทั้งวันเดือนปีและเวลาตามนาฬิกาของเครื่อง
FormatDateTime	เป็นการจัดรูปแบบการแสดงผลเกี่ยวกับวันเดือนปีและเวลา
IsLeapYear	เป็นการตรวจสอบว่าปีที่มีวันที่ 29 กุมภาพันธ์หรือไม่ ซึ่งทุก4ปี จะมี leap year คือปีที่มีวันที่ 29 กุมภาพันธ์หนึ่งครั้ง
DayOf	เป็นการดึงตัวเลขส่วนที่เป็นวันออกมาจากข้อมูลที่เป็นวันเดือนปี หรือวันเวลา
MonthOf	เป็นการดึงตัวเลขส่วนที่เป็นเดือนออกมาจากข้อมูลที่เป็นวัน เดือนปี หรือวันเวลา
YearOf	เป็นการดึงตัวเลขส่วนที่เป็นปีออกมาจากข้อมูลที่เป็นวันเดือนปี หรือวันเวลา
HourOf	เป็นการดึงตัวเลขส่วนที่เป็นชั่วโมงออกมาจากข้อมูลที่เป็นวันเดือนปี หรือวันเวลา
MinuteOf	เป็นการดึงตัวเลขส่วนที่เป็นนาทีออกมาจากข้อมูลที่เป็น
SecondOf	เป็นการดึงตัวเลขส่วนที่เป็นวินาทีออกมาจากข้อมูลที่เป็นวันเดือนปี หรือวันเวลา
Daysbetween	เป็นการคำนวณผลต่างของวันเดือนปี 2 ค่า รีเทิร์นค่าเป็นจำนวนเต็ม (ถ้าเป็นคนละวันกันแต่ต่างกันไม่ถึง 24ชั่วโมง ผลต่างจะเป็น 0)

DaySpan	เป็นการคำนวณผลต่างของวันเดือนปี 2 ค่า รีเทิร์นค่าเป็นจำนวนทศนิยม (มีเศษของวันด้วย)
IsToday	เป็นการตรวจสอบว่าเป็นวันเดือนปีปัจจุบันหรือไม่

2.2.22 ภาษาออบเจ็กต์ปาสคาล

ใน Delphi 7นั้นเราได้เรียนรู้เกี่ยวกับการเขียนโปรแกรมในลักษณะต่างๆซึ่งใช้งานภาษาออบเจ็กต์ปาสคาล ซึ่งมีความคล้ายคลึงกับภาษาปาสคาลมาก ทำให้ผู้ที่มีพื้นฐานภาษาปาสคาลมาก่อนใช้งาน Delphi ได้อย่างรวดเร็ว ซึ่งภาษาออบเจ็กต์ปาสคาลนั้นมีความสามารถในตัวภาษาอีกหลายอย่างที่น่าเรียนรู้เพื่อที่เราจะสามารถใช้งานภาษาออบเจ็กต์ปาสคาลได้อย่างเต็มความสามารถที่มีอยู่

เราใช้ภาษาออบเจ็กต์ปาสคาลในการเขียนโปรแกรมกับ Delphi 7 ซึ่งเราจะเก็บซอร์สโค้ดที่ได้เขียนขึ้นไว้ในไฟล์ที่เรียกว่า ยูนิต(Unit)ซึ่งจะถูกบันทึกไว้ในไฟล์ pas และเมื่อไฟล์ยูนิตถูกคอมไพล์ก็จะสร้างไฟล์ที่เป็นผลจากการคอมไพล์มีนามสกุลเป็น .dcu

ปกติเมื่อเริ่มสร้างแอพพลิเคชันใน Delphi 7 (ก็คือการสร้างโปรเจ็กต์ใหม่ขึ้นมา) จะมีการสร้างไฟล์ ชื่อ Unit1.pas ขึ้นมาให้เราเขียนโค้ดเข้าไป เมื่อเราตรวจสอบจาก Code Editer จะพบว่า Delphi ได้เขียนโค้ดบางส่วนไว้ให้แล้ว

ไฟล์ยูนิตหนึ่งๆจะใช้กับการเขียนโค้ดสำหรับฟอร์ม 1 ฟอร์ม ดังนั้นถ้าโปรเจ็กต์ของเราประกอบด้วยฟอร์มมากกว่าหนึ่งฟอร์มสามารถมีไฟล์ยูนิตที่มีนามสกุล .pas ได้มากกว่า 1 ไฟล์ได้ เมื่อเรามีการนำคอมโพเนนท์เข้าไปยังฟอร์มก็จะพบว่า Delphi ได้เขียนโค้ดให้เราในตอนต้นของไฟล์ยูนิตแล้ว

2.2.22.1 โครงสร้างของยูนิต

เมื่อกลับมามองไฟล์ Unit1.pas ที่ Delphi สร้างไว้ให้เราสามารถแบ่งโค้ดออกเป็นส่วนต่างๆดังนี้

- ส่วน Interface ส่วนนี้เป็นส่วนที่ใช้ประกาศชนิดข้อมูล,ตัวแปร,ค่าคงที่ ,ออบเจ็กต์ โพรซีเจอร์และฟังก์ชันซึ่งทุกสิ่งที่บรรจุไว้ในส่วนนี้สามารถเข้าถึงและใช้งานได้จากยูนิตอื่นๆต้องไม่ลืมว่าโปรเจ็กต์หนึ่งๆสามารถมีไฟล์ยูนิต ได้หลายตัว ซึ่งพื้นที่ส่วนinterfaceนี้จะเริ่มจากคำว่า interfaceไปจนถึง mplementation

- ส่วน implementation ส่วนนี้จะทำหน้าที่เหมือนกับส่วน Interface ต่างกันตรงที่ขอบเขตการเข้าถึงข้อมูลคือ จะเข้าถึงชนิดข้อมูล,ตัวแปร ,ค่าคงที่ ,ออบเจ็กต์ โพรซีเจอร์และฟังก์ชัน ได้จากสิ่งที่อยู่ขอบเขตเฉพาะภายในยูนิตนี้เท่านั้นยูนิตอื่นๆไม่มีสิทธิ์เข้าถึงซึ่งพื้นที่ส่วนimplementation นี้จะเริ่มต้นจากคำว่า implementation ไปจนถึง initialization

- ส่วน initialization ส่วนนี้จะใช้เก็บคำสั่งที่ถูกเรียกใช้งานก่อนการทำงานของแอพพลิเคชัน โดยปกติจะทำงานก่อนที่จะมีการสร้างออบเจ็กต์ หรือฟอร์มขึ้นมา ดังนั้นเราจึงมักใช้พื้นที่ส่วนนี้กำหนดค่า

ให้กับตัวแปรบางตัว ซึ่งพื้นที่ของส่วน initialization นี้เริ่มต้นจากคำว่า initialization ไปจนถึง finalization (ถ้าไม่มีคำว่า finalization ก็จะถึงคำว่า end) สำหรับส่วนนี้จะมีก็ได้หรือไม่มีก็ได้แล้วแต่โปรแกรม

- ส่วน finalization ส่วนนี้จะทำหน้าที่ตรงกันข้ามกันกับ initialization ซึ่งพื้นที่ส่วน finalization นี้จะเริ่มตั้งแต่คำว่า finalization ไปจนถึง end.

2.2.23 ไฟล์โปรเจ็กต์

ปกติเราจะเขียนโปรแกรมที่ไฟล์ยูนิตเท่านั้น แต่ในการทำงานจริงๆของแอพพลิเคชันนั้นจะต้องเริ่มจากการทำงานของไฟล์โปรเจ็กต์ก่อน ซึ่งรายละเอียดการทำงานของไฟล์โปรเจ็กต์นี้เราไม่ต้องจัดการเอง แต่สามารถดูการทำงานภายในได้โดยเลือกเมนู Project > View Source ซึ่งใน Code Editor จะเพิ่มแท็บที่เป็นรายละเอียดของไฟล์โปรเจ็กต์

สำหรับโค้ดของไฟล์โปรเจ็กต์นั้นประกอบไปด้วยส่วนสำคัญ 3 ส่วน คือ

- program heading
- uses clause
- การทำงานของโปรแกรม

. สำหรับส่วนที่เป็น program heading นั้นจะบอกถึงชื่อของโปรแกรม ในที่นี้ก็คือชื่อที่เราตั้งให้ ในตอนบันทึกชื่อไฟล์โปรเจ็กต์นั่นเอง และชื่อนี้จะถูกสร้างเป็นไฟล์ .EXE ด้วยส่วน use clause จะเป็นส่วนที่เราใช้ประกาศยูนิตที่ใช้งาน ถ้ามีมากกว่าหนึ่งยูนิตในนี้ก็จะเก็บรายการของยูนิตทั้งหมดที่ใช้เอาไว้ส่วนสุดท้ายจะเก็บรายละเอียดของคำสั่งในการทำงานของ แอพพลิเคชันซึ่งทุกๆแอพพลิเคชันที่เป็นชนิดเดียวกันจะมีส่วนเหมือนกันหมดคือมีส่วนที่กำหนดค่าเริ่มต้นสร้างฟอร์มขึ้นมาแล้วเริ่มการทำงานให้กับแอพพลิเคชัน (ซึ่งถ้าเป็นแอพพลิเคชันทั่วไปฟอร์มที่ถูกสร้างขึ้นมาก็จะรอการโต้ตอบจากผู้ใช้งาน)

2.2.24 แนวคิดการเขียนโปรแกรมแบบOOP

ที่ผ่านมาเราได้เรียนรู้การเขียนโปรแกรมในลักษณะการเขียนโปรแกรมในลักษณะโครงสร้าง (Structure Programming) ซึ่งเลียนแบบแนวคิดของคนทั่วไป แต่ในระบบงานที่ซับซ้อนขึ้นการเขียนโปรแกรมแบบโครงสร้างอาจทำให้เกิดความยุ่งยากมากทำให้เกิดมีแนวคิดการเขียนโปรแกรมแบบใหม่ที่เรียกว่า Object Oriented programming หรือ OOP ขึ้นมา ซึ่งช่วยแก้ไขปัญหาการเขียนโปรแกรมที่ซับซ้อนเพิ่มมากขึ้นแม้การเขียนโปรแกรมแบบ OOP เหมือนยุ่งยาก แต่ก็เป็นการเลียนแบบแนวคิดของสิ่งที่เราพบเห็นในสิ่งต่างๆที่อยู่รอบตัวทั้งนั้น

2.2.24.1 คลาสกับออบเจ็กต์

แนวคิดของ OOP นั้นจะกำหนดให้สิ่งต่างๆ เป็นวัตถุ หรือออบเจ็กต์ (Object) ซึ่งออบเจ็กต์แต่ละตัวจะถูกสร้างขึ้นมาจากพิมพ์ที่เรียกว่า คลาส (Class) ดังนั้นออบเจ็กต์ที่สร้างจากคลาสจะถือว่าเป็นคนละตัวกันแต่มีชนิดเดียวกัน เพราะสร้างจากคลาสเดียวกัน ตัวอย่างเช่น คลาสของรถยนต์ ที่สร้างออกมาคนละคัน เมื่อมามองถึง

ออบเจ็กต์ตัวหนึ่งๆ จะเห็นว่ามันจะต้องมีข้อมูลที่ใช้กำหนดลักษณะเฉพาะของออบเจ็กต์ซึ่งเราเรียกว่า พร็อพเพอร์ตี้ (Property) ตัวอย่างเช่น ออบเจ็กต์รถยนต์จะมีพร็อพเพอร์ตี้ที่แสดงลักษณะเฉพาะ เช่น ยี่ห้อรถยนต์. สีรถยนต์, ทะเบียนรถยนต์,ราคา เป็นต้น

เมื่อมีพร็อพเพอร์ตี้แล้วสิ่งหนึ่งที่มักจะขาดไม่ได้ในออบเจ็กต์ก็คือ ความสามารถในการทำงานของออบเจ็กต์ซึ่งเราเรียกว่า เมธอด (Method) ตัวอย่างเช่น ออบเจ็กต์รถยนต์ก็จะมีเมธอดต่างๆเช่น สตาร์ทรถยนต์,เลี้ยวรถ,หยุดรถ เป็นต้น

เมื่อเราสร้างออบเจ็กต์หนึ่งๆขึ้นมาจากคลาส ในภาษาของการเขียนโปรแกรมจะกล่าวได้ว่าเป็น " ออบเจ็กต์ ก็คือ อินสแตนซ์ (Instance) ของคลาส"

2.2.24.2 การเข้าถึงข้อมูลที่เก็บภายในคลาส

ในคลาสจะประกอบด้วยข้อมูลต่างๆนั่นคือ พร็อพเพอร์ตี้ และเมธอด ซึ่งศัพท์ของนักเขียนโปรแกรมจะเรียกทั้งสองอย่างรวมๆว่า เมมเบอร์ (Member) ของคลาส

จุดดีอย่างหนึ่งของคลาสคือการที่ไม่อนุญาตให้เข้าถึงข้อมูลภายในของคลาสได้โดยตรงจะต้องกระทำผ่านเมมเบอร์ของคลาสซึ่งก็สอดคล้องกับชีวิตจริง ที่เราใช้งานรถยนต์ได้จากสิ่งที่เตรียมไว้ให้เช่น พวงมาลัย, กระปุกเกียร์,คันเร่ง ฯลฯ โดยไม่ต้องสนใจการสตาร์ทรถว่าต้องเกิดจาการทำงานของกลไกภายในอะไรบ้าง เพราะถ้าขืนสนใจจะทำให้การขับขี่ยุ่งยากมาก แถมมีโอกาสเกิดความผิดพลาดได้มาก

สำหรับการเข้าถึงข้อมูลภายในคลาส Delphi นั้นแบ่งได้เป็น 3 ระดับได้แก่

- Private เป็นระดับการเข้าถึงข้อมูลที่เข้าถึงได้เฉพาะข้อมูลที่อยู่ภายในตัวคลาส หรือ อินสแตนซ์ของคลาสเท่านั้น ไม่อนุญาตให้ผู้ใช้ หรือคลาสอื่นใดเข้าถึงได้
- Public เป็นระดับการเข้าถึงข้อมูลที่เข้าถึงได้จากผู้ใช้คลาสทุกคน หรือคลาสอื่นๆ โดยไม่จำกัด
- Static เป็นระดับการเข้าถึงข้อมูลที่เข้าถึงได้จากผู้ใช้คลาสแบบกลุ่ม หรือคลาสอื่นโดยไม่จำกัด
- Protested เป็นระดับการเข้าถึงข้อมูลเฉพาะคลาสที่ถูกสืบทอดมาจากคลาสนี้เท่านั้น (เรื่องของการสืบทอด ของคลาสจะได้กล่าวในหัวข้อถัดไป)

เพื่อความเข้าใจจะลองสร้างคลาส สุนัข ขึ้นมา โดยมีรายละเอียดของเมมเบอร์ และระดับการเข้าถึงแต่ละเมมเบอร์ ดังตาราง

เมมเบอร์	รายละเอียด	ระดับการเข้าถึง
พร็อพเพอร์ตี้	สายพันธุ์	Private
	จำนวนขา	Public
	สีขน	Public
เมธอด	วิ่งเร็ว	Public
	เห่า	Public
	ว่ายน้ำ	Public

ตารางที่ 2.4 แสดงรายละเอียดของเมมเบอร์ และระดับการเข้าถึงแต่ละเมมเบอร์

ทดลองสร้างอินสแตนซ์ขึ้นมา มีรายละเอียดดังตาราง

อินสแตนซ์	เจ้าตูบ	เจ้าโทน	เจ้าดิ๊ก
สายพันธุ์	4	4	4
สีขน	น้ำตาล	ขาว	ดำ
ลักษณะหาง	หางยาวโค้ง	หางสั้นกุด	หางสั้น
วิ่งเร็ว	-	-	-
เห่า	-	-	-
ว่ายน้ำ	-	-	-

ตารางที่ 2.5 ทดลองสร้างอินสแตนซ์

2.2.24.3 Constructor และ Destructor

นอกเหนือจากเมธอดที่เรากำหนดให้คลาสแล้ว ในการเขียนโปรแกรมแบบ OOP นั้นยังต้องมีเมธอดที่จำเป็นอยู่อีก 2 ตัว ซึ่งจะต้องทำหน้าที่ทุกครั้งเมื่อออบเจ็กต์นั้นเริ่มสร้างขึ้นมา หรือกำลังจะเลิกใช้งานไปแล้ว

Constructor เป็นเมธอดที่ทำงานตอนที่เราเริ่มสร้างออบเจ็กต์ขึ้นมาจากคลาส โดยจะทำหน้าที่กำหนดค่าเริ่มต้นที่จำเป็นให้กับพร็อพเพอร์ตี้ต่างๆจับจองทรัพยากรของระบบที่จำเป็นต่อการทำงานของออบเจ็กต์

Destructor เป็นเมธอดที่ทำงานในลักษณะตรงกันข้ามคือ ทำหน้าที่ตอนที่ออบเจ็กต์นั้นเลิกใช้งาน โดยจะคืนทรัพยากรให้แก่ระบบปฏิบัติการ

2.2.25 คุณสมบัติสำคัญของการเป็น OOP

ที่ผ่านมาเป็นการอธิบายแนวคิดของการเขียนโปรแกรมแบบ OOP คร่าวๆ ซึ่งการเขียนโปรแกรมแบบ OOP นั้นมีรายละเอียดเกินกว่าจะกล่าวไว้ในปริญญานิพนธ์เล่มนี้ได้ จึงขอแนะนำพื้นฐานอีกเล็กน้อยก่อนจะศึกษาและนำไปใช้งาน

สำหรับคุณสมบัติสำคัญที่ต้องรู้เมื่อคิดโปรแกรมแบบ OOP ก็คือ คุณสมบัติสำคัญ 3 ประการดังนี้

- Encapsulation เป็นคุณสมบัติที่ว่า เราไม่สนใจรายละเอียดที่ไม่สมควรจะสนใจ เช่น เราเขียนโปรแกรมซึ่งมีการใช้เมนูใน Delphi เราก็ไม่ต้องไปสนใจว่า Delphi นั้นมีวิธีการจัดการเมนู เช่น การวาดเมนู, การแสดงเมนู, การซ่อนเมนู, การขีดเขียนเครื่องหมาย หรือใส่รูปภาพบนเมนูอย่างไร เราก็เพียงแต่กำหนดคุณสมบัติและใช้งานตามความสามารถที่มีให้เท่านั้น
- Inheritance เป็นคุณสมบัติที่ว่าคลาสต้องสามารถสืบทอดได้เช่นเดียวกับภาษาโปรแกรมที่เรากำหนดเป็นคอมโพเนนต์ (ทั้งที่มองเห็น และมองไม่เห็น) ก็ต้องสืบ

- ทอดได้ โดยดีไรฟ์คลาส (คลาสที่ถูกสืบทอดมา) นั้น สามารถเพิ่มเติมพร็อพเพอร์ตี้และเมธอดได้อีก รวมทั้งสามารถปรับเปลี่ยนพร็อพเพอร์ตี้ หรือเมธอดเดิมได้ตามความเหมาะสม
- Polymorphism เป็นคุณสมบัติที่ว่า สามารถเปลี่ยนแปลงความสามารถของคลาสให้เข้ากับสภาพแวดล้อมได้ ตัวอย่างง่ายๆ เช่น เราสร้างคลาสที่ชื่อว่า Shape ซึ่งจะใช้สร้างออบเจ็กต์เป็นรูปทรงต่างๆ เช่น วงกลม,สามเหลี่ยม,สี่เหลี่ยม เป็นต้น แล้วเราก็มีเมธอด Area ซึ่งใช้คำนวณหาพื้นที่ของรูปทรง แน่นอนว่าเมธอด Area ของการเรียกใช้แต่ละครั้งต้องคำนึงด้วยว่าระบุพร็อพเพอร์ตี้ชนิดรูปทรงว่าเป็นอะไร ซึ่งก็จะทำให้เรามีวิธีการคำนวณหาที่แตกต่างกัน

2.2.26 หลักการสร้างแอพพลิเคชันด้วยคอมโพเนนต์

เคล็ดลับความสำเร็จของการพัฒนาแอพพลิเคชันซอฟต์แวร์ที่ผ่านมาก็คือ การที่เรามีเครื่องมือช่วยให้เขียนโปรแกรม และสร้างแอพพลิเคชันที่ดีอย่าง Delphi ที่ช่วยให้เราสามารถออกแบบ,เขียนโปรแกรม ,ทดสอบ และแก้ไขสิ่งที่ได้สร้างขึ้นมานั้นได้อย่างรวดเร็ว ยืดหยุ่น ซึ่งเมื่อผ่านไปหลายเวอร์ชันก์มีคอมโพเนนต์ชนิดต่างๆเพิ่มขึ้นมาอย่างมากมายให้ใช้งานแทบไม่หมด

2.2.27 การเรียกใช้งานคอมโพเนนต์

2.2.27.1 การกำหนดพร็อพเพอร์ตี้ให้คอมโพเนนต์

2.2.27.2 การเรียกใช้งานเมธอด

2.2.27.3 การเขียนโปแรกมเพื่อจัดการกับอีเวนต์ต่างๆ

2.2.28 กลุ่มของคอมโพเนนต์ใน Delphi 7

Delphi 7 ได้จัดแบ่งคอมโพเนนต์ต่างๆออกมาเป็นกลุ่มๆให้เลือกใช้งานได้ตามลักษณะของงาน ซึ่งมีมากมายถึง 33 กลุ่ม โดยเก็บแยกกันไว้ใน Component Palette ซึ่งประกอบไปด้วย

1. Standard : เป็นกลุ่มคอมโพเนนต์มาตรฐานซึ่งมีการใช้งานทั่วไปใน Windows

2. Additional : เป็นกลุ่มคอมโพเนนท์เพิ่มเติมพิเศษช่วยเสริมการทำงานของคอมโพเนนท์มาตรฐาน ช่วยให้แอพพลิเคชันใช้งานสะดวก และสวยงามน่าใช้

3. Win32 : เป็นกลุ่มคอมโพเนนต์ซึ่งเกิดมาพร้อมกับระบบปฏิบัติการ 32 บิตของไมโครซอฟท์ (Windows 95,98,NT,2000,Me และ XP)

4. System : เป็นกลุ่มคอมโพเนนต์ที่สร้างเพื่อติดต่อกับการทำงานภายในของระบบคอมพิวเตอร์ที่เราใช้งาน

5. Data Access : เป็นกลุ่มคอมโพเนนท์ที่ใช้ในการติดต่อกับฐานข้อมูล

6. Data Controls : เป็นกลุ่มคอมโพเนนท์ที่ใช้สร้างส่วนติดต่อผู้ใช้ในแอพพลิเคชันด้านฐานข้อมูล

7. dbExpress : เป็นกลุ่มของคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันด้สนฐานข้อมูลด้วย dbExpress

8. DataSnap : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันฐานข้อมูลแบบ multi-tiered

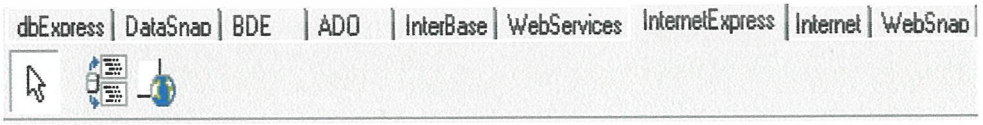
9. BDE : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างการเชื่อมต่อกับฐานข้อมูลที่ใช้ Borland Database Engine (BDE)

10. ADO : เป็นกลุ่มคอมโพเนนต์ที่สร้างเพื่อรองรับเทคโนโลยีการเชื่อต่อฐานข้อมูลแบบ ADO ของไมโครซอฟท์

11. InterBase : เป็นกลุ่มคอมโพเนนต์ที่สร้างขึ้นมาเพื่อรองรับเทคโนโลยีการเชื่อมต่อฐานข้อมูลแบบ InterBase ของบอร์แลนด์

12. WebServices : เป็นกลุ่มคอมโพเนนต์ที่ใช้เขียนแอพพลิเคชันฝั่งไคลเอนท์ที่จะเรียกใช้ WebServices ผ่าน SOAP

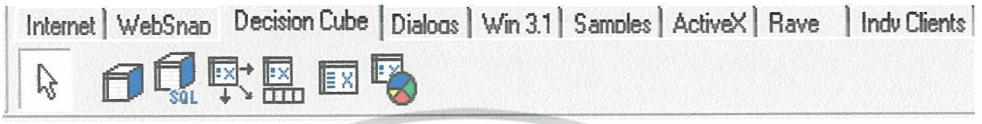
13. InternetExpress : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันที่ใช้งานผ่านอินเตอร์เน็ตในรูปแบบใหม่ซึ่งรองรับ XML



14. Internet : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันอินเตอร์เน็ตในฝั่งเซิร์ฟเวอร์

15. WebSnap : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันอินเตอร์เน็ตในฝั่งเซิร์ฟเวอร์

16. Decision Cube : เป็นกลุ่มคอมโพเนนต์

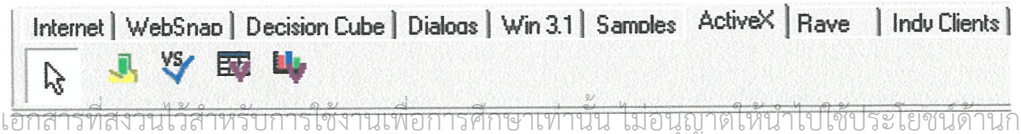


17. Dialogs : เป็นกลุ่มคอมโพเนนต์ที่นำมาเรียกใช้ไดอะล็อกมาตรฐานของ Windows เช่น ไดอะล็อก เปิดไฟล์ บันทึกไฟล์ หรือเลือกสี เป็นต้น

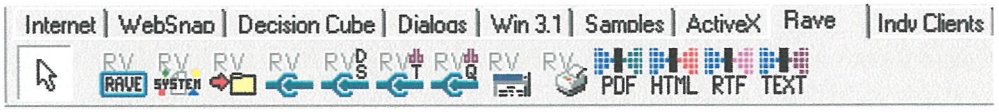
18. Win3.1 : เป็นกลุ่มคอมโพเนนต์ที่นำมาสร้างแอพพลิเคชันที่ต้องการให้ใช้งานร่วมกับคอมโพเนนต์ในสไตล์ของ Windows 3.1ได้ ซึ่งจะใช้งานเข้ากันได้กับคอมโพเนนต์ของ Delphi เวอร์ชันแรกๆ

19. Samples : เป็นกลุ่มคอมโพเนนต์ที่ Delphi ได้เตรียมเอาไว้ให้ศึกษาถึงการสร้างคอมโพเนนต์ขึ้นมาใช้เอง ซึ่งจะมีซอร์สโค้ดของคอมโพเนนต์ทั้งหมดด้วย โดยจะอยู่ที่

20. ActiveX : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างตามแนวคิด ActiveX Controlของไมโครซอฟท์



21. Rave : เป็นกลุ่มคอมโพเนนต์ที่นำมาใช้สร้างส่วนที่เกี่ยวกับรายงานของแอพพลิเคชันฐานข้อมูล



22. Indy Client : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันในฝั่งไคลเอนท์ตามแนวคิดของ Internet Direct หรือ Indy ซึ่งก็คือคอมโพเนนต์โอเพ่นเซอร์ส ซึ่งจะเรียกใช้งานโปรโตคอลอินเตอร์เน็ตมาตรฐานๆ

23. Indy Server : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างแอพพลิเคชันในฝั่งเซิร์ฟเวอร์ตามแนวคิดของ Internet Direct หรือ Indy ซึ่งก็คือคอมโพเนนต์โอเพ่นเซอร์ส ซึ่งจะเรียกใช้งานโปรโตคอลอินเตอร์เน็ตมาตรฐานๆ

24. Indy Intercepts : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างส่วนที่เกี่ยวกับการเข้ารหัส การบีบอัดข้อมูลสำหรับแอพพลิเคชันตามแนวคิดของ Internet Direct หรือ Indy

25. Indy I/O Handlers : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างส่วนที่เกี่ยวกับ I/O และ Socket สำหรับแอพพลิเคชันตามแนวคิดของ Internet Direct หรือ Indy

26. Indy Misc : เป็นกลุ่มคอมโพเนนต์ที่ช่วยให้แอพพลิเคชันตามแนวคิดของ Internet Direct หรือ Indy มีลูกเล่นและความน่าใช้งานเพิ่มมากขึ้น

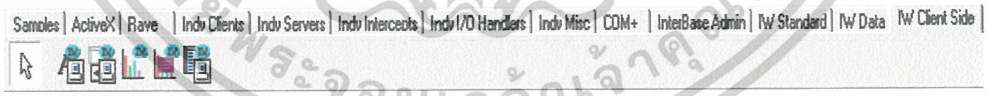
27. COM+ : เป็นกลุ่มคอมโพเนนต์ที่ทำหน้าที่สร้างแอพพลิเคชันตามแนวความคิดของ COM+ (COM Plus)

28. InterBase Admin : เป็นกลุ่มคอมโพเนนต์ที่ช่วยเพิ่มความสะดวกในการพัฒนาแอพพลิเคชันด้วย InterBase 6

29. IW Standard : เป็นกลุ่มคอมโพเนนต์มาตรฐานที่ใช้สร้างเว็บแอพพลิเคชันตามแนวคิดของ IntraWeb

30. IW data : เป็นกลุ่มคอมโพเนนต์มาตรฐานข้อมูลที่ใช้สร้างเว็บแอพพลิเคชันตามแนวคิดของ IntraWeb

31. IW Client Side : เป็นกลุ่มคอมโพเนนต์ที่ใช้สร้างเว็บแอพพลิเคชันในฝั่งไคลเอนท์ตามแนวคิดของ IntraWeb



32. IW Control : : เป็นกลุ่มคอมโพเนนต์ทั่วไปที่ใช้สร้างเว็บแอพพลิเคชันตามแนวคิดของ IntraWeb

33. Servers : เป็นกลุ่มคอมโพเนนต์ที่ใช้เป็นออบเจ็กต์ COM Server

2.2.29 คอมโพเนนต์ Label : ป้ายข้อความ

คอมโพเนนต์ Label นั้นจะทำหน้าที่แสดงข้อความตามที่เราต้องการ ซึ่งจะเป็นข้อความที่ให้ผู้ใช้งานอ่านอย่างเดียว (ไม่ต้องการให้แก้ไขข้อความ) ซึ่งมักจะแสดงข้อความแนะนำ หรือเป็นผลลัพธ์จากการทำงาน ดังนั้นหลายๆคนจึงเรียกคอมโพเนนต์ Label ว่า ป้ายข้อความ

พร็อพเพอร์ตี้สำคัญของ Label

- Caption — เป็นข้อความที่จะแสดงผลใน Label
- AutoSize — ถ้ากำหนดเป็น True จะทำให้มีการปรับขนาดของ Label ให้พอดีกับข้อความที่อยู่ภายในโดยอัตโนมัติ (ปกติจะกำหนดเป็น True)
- Alignment — เป็นรูปแบบการจัดการวางแนวของข้อความว่าจะชิดซ้าย ชิดขวาหรืออยู่ตรงกลาง
- Layout — เป็นตำแหน่งของข้อความที่จะวางอยู่บน Label ว่าจะติดขอบด้านบน อยู่ตรงกลาง หรือติดขอบด้านล่าง (ปกติจะกำหนดให้ชิดขอบด้านบน)
- Transparent — ถ้ากำหนดเป็น True จะทำให้พื้นหลังเป็นแบบโปร่งใส แต่ถ้ากำหนดเป็น False จะทำให้พื้นหลังของ Label เป็นแบบทึบ (Opaque)
- Wordwrap — ถ้ากำหนดเป็น True จะทำให้ข้อความที่ยาวเกินขนาดของ Label ถูกบังคับให้ขึ้นบรรทัดใหม่ (ปกติกำหนดเป็น False)

2.2.30 คอมโพเนนต์ Button : ปุ่มคำสั่ง

คอมโพเนนต์ Button เป็นคอมโพเนนต์ที่ถูกใช้งานมากที่สุด ทำหน้าที่รับคำสั่งจากผู้ใช้งานด้วยการคลิกหรือกดปุ่ม <Enter> ซึ่งเราจะใช้คอมโพเนนต์ Button แทนคำสั่งหนึ่งคำสั่ง

พร็อพเพอร์ตี้สำคัญของ Button

- Caption — เป็นข้อความที่ปรากฏบนปุ่ม
- Default — ถ้ากำหนดเป็น True จะทำให้ทุกครั้งที่กดปุ่ม <Enter> มีผลเหมือนกับการคลิกปุ่มนั่นคือจะทำให้เกิดอีเวนต์OnClick (เมื่อกดปุ่ม <Enter>) ปกติกำหนดเป็น False และจะต้องมีปุ่มเดียวในฟอร์มเท่านั้นที่กำหนดเป็น True ได้
- Cancel — ถ้ากำหนดเป็น True จะทำให้ทุกครั้งที่กดปุ่ม <Esc> มีผลเหมือนกับการคลิกปุ่มนั่นคือจะทำให้เกิดอีเวนต์ OnClick (เมื่อกดปุ่ม<Esc>)

- Cursor เป็นลักษณะของเคอร์เซอร์เมาส์ที่จะแสดงเมื่อวางทาบเหนือ
- Hint เป็นข้อความที่เราใช้อธิบายเมื่อเมาส์มาทาบอยู่เหนือปุ่มหรือเรียกอีกอย่างว่า Tooltip
- ShowHint ถ้ากำหนดเป็น True จะมีการแสดงข้อความที่กำหนดในพร็อพเพอร์ตี้ Hint เมื่อวางเมาส์ทาบอยู่เหนือปุ่ม
- Anchors เป็นการกำหนดว่าปุ่มจะมีการปรับตำแหน่งฟอร์มตามขนาดที่เปลี่ยนไปหรือไม่ปกติจะกำหนดให้ยึดแนวบน และขอบฟอร์ม ตามที่เปลี่ยนไปหรือไม่ ปกติจะกำหนดให้มีการปรับตำแหน่งตามแนวอื่นๆได้

เมธอดสำคัญของ Button

- SetFocus เป็นการทำให้ปุ่มนั้นโดน Focus

อีเวนต์สำคัญของ Button

- อีเวนต์ OnClick เป็นอีเวนต์หลักที่จะถูกเรียกเมื่อคลิกที่ปุ่มนี้

2.2.31 คอมโพเนนต์ Edit : ช่องรับข้อความ

เราจะใช้คอมโพเนนต์ Edit ในการรับข้อความที่ป้อนจากผู้ใช้งานเข้ามา นอกจากนี้ยังสามารถใช้แสดงข้อความในลักษณะต่างๆได้คล้ายๆกับ Label อีกด้วย

พร็อพเพอร์ตี้สำคัญของ Edit

- Text เป็นข้อความที่จะแสดงใน Edit
- MaxLength เป็นจำนวนตัวอักษรสูงที่สุดที่ยอมให้ผู้ใช้งานป้อนเข้ามาใช้จำกัดความยาวของข้อความที่รับเข้า
- ReadOnly ถ้ากำหนดเป็น True จะทำหีคอมโพเนนต์ Edit ทำหน้าที่แสดงข้อความเพียงอย่างเดียว จะไม่สามารถป้อนข้อมูลเข้าไปได้
- PasswordChar เป็นการกำหนดตัวอักษรที่จะแสดงผลเมื่อเราใช้คอมโพเนนต์ Edit ตัวนั้นในการรับรหัสผ่านจากผู้ใช้งาน (เรามักจะกำหนดด้วย)
- CharCase เป็นการแปลงข้อความป้อนเข้ามาให้แสดงผลเป็นตัวพิมพ์ใหญ่ทั้งหมดหรือเป็นตัวพิมพ์เล็กทั้งหมดหรือไม่ต้องแปลง
- SelText เป็นข้อความที่ได้จากการทำแถบสีดำเลือก (ทำไฮ

ไลท์) ที่สวยงามในคอมโพเนนต์ Edit

- SelStart — เป็นตำแหน่งตัวอักษรตัวแรกของ SelText โดยจะเทียบกับตัวอักษรที่มีทั้งหมดในพร็อพเพอร์ตี้ Text
- SelLength — เป็นความยาวของ SelText
- AutoSelect — ปกติกำหนดเป็น True หมายความว่าจะเลือกข้อความทั้งหมดใน Edit เมื่อเรากดปุ่ม <Tab>เข้าไปยังคอมโพเนนต์ Edit ตัวนั้น

เมธอดสำคัญของ Edit

- Clear — เป็นการลบข้อความทั้งหมดออกจากคอมโพเนนต์ Edit
- ClearSelection — เป็นการลบข้อความเฉพาะส่วนที่ทำไฮไลท์ไว้ (ก็คือ SelTex นั่นเอง)
- CutToClipboard — เป็นการตัดข้อความที่ทำไฮไลท์ไว้นั้นเก็บในคลิปบอร์ด
- CopyToClipboard — เป็นการก๊อปปี้ข้อความที่ทำไฮไลท์ไว้นั้นเก็บในคลิปบอร์ด
- PasteFromClipboard — เป็นการนำข้อความจากคลิปบอร์ดมาแทรก ณ ตำแหน่ง ของเคอร์เซอร์ในคอมโพเนนต์ Edit
- SelectAll — เป็นการเลือกข้อความทั้งหมดในคอมโพเนนต์ Edit
- SetFocus — เป็นการทำให้ปุ่มนั้นโดน Focus

อีเวนต์สำคัญของ Edit

- OnChange — เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการเปลี่ยนแปลงข้อความภายในคอมโพเนนต์ Edit
- OnEnter — เป็นเหตุการณ์ที่เกิดขึ้นเมื่อย้ายการทำงานเข้าไปสู่คอมโพเนนต์ Edit (คอมโพเนนต์ Edit ถูก Focus)
- OnExit — เป็นเหตุการณ์ที่เกิดขึ้นเมื่อย้ายการทำงานออกจากคอมโพเนนต์

2.2.32 คอมโพเนนต์ Memo : รับข้อความได้มากกว่า 1 บรรทัด

คอมโพเนนต์ Memo นั้นมีความสามารถเช่นเดียวกับคอมโพเนนต์ Edit แต่มีความพิเศษกว่าตรงที่สามารถแสดงข้อความได้มากกว่า 1 บรรทัด และสามารถอ้างอิงข้อความในบรรทัดที่ต้องการได้ด้วย

พร็อพเพอร์ตี้ที่สำคัญของ Memo

- Test เป็นข้อความที่แสดงภายในคอมโพเนนต์ Memo
- ScrollBars ถ้ากำหนดเป็น True จะมีการแสดง Scrollbar อัตโนมัติ เมื่อพื้นที่แสดงผลไม่เพียงพอ
- WantTabs ถ้ากำหนดเป็น True จะทำให้เมื่อกดปุ่ม <Tab> จะมีการเว้นย่อหน้าให้ แต่ถ้ากำหนดเป็น False จะทำให้มีการย้ายการทำงานออกจากคอมโพเนนต์ไปยังคอมโพเนนต์อื่นๆ ตามลำดับในพร็อพเพอร์ตี้ Tab Order
- WantReturns ถ้าเรากำหนดเป็น True จะทำให้เมื่อกดปุ่ม <Enter> จะมีการขึ้นบรรทัดใหม่
- Alignment เป็นการกำหนดตำแหน่งของข้อความในคอมโพเนนต์ Memo
- Lines เป็นพร็อพเพอร์ตี้ที่ใช้เก็บและจัดการข้อความที่อยู่ในคอมโพเนนต์ Memoโดยที่ข้อมูลในคอมโพเนนต์ Memo จะเก็บอยู่ใน บรรทัดต่างๆ (เริ่มจากบรรทัดที่ 0) ซึ่งเราสามารถเพิ่มข้อความ หรือลบข้อความในบรรทัดที่ต้องการได้โดยใช้พร็อพเพอร์ตี้ Lines ซึ่งพร็อพเพอร์ตี้นี้จะช่วยให้การทำงานของโปรแกรมง่ายขึ้น

2.2.33 คอมโพเนนต์ RadioButton : ตัวเลือกที่เลือกที่เลือกได้เพียงตัวเดียว

ใช้งานคอมโพเนนต์ RadioButton ในการสร้างรายการตัวเลือกซึ่งจะยอมให้ผู้ใช้งานเลือกได้ เพียงตัวเลือกเดียวเท่านั้น ซึ่งก็จะเหมือนการฟังวิทยุที่เราสามารถฟังได้เพียงทีละสถานี

พร็อพเพอร์ตี้สำคัญของ RadioButton

- Caption เป็นข้อความที่แสดงบนตัว RadioButton
- Checked ถ้ากำหนดเป็น True แสดงว่ามีการเลือกที่คอมโพเนนต์ตัวนี้ ซึ่งจะทำให้มีการเปลี่ยนค่าที่คอมโพเนนต์ RadioButton ตัวอื่นๆ เป็น False ทั้งหมด
- Alignment เป็นการกำหนดแนวการวางของข้อความบน RadioButton

2.2.34 คอมโพเนนต์ Checkbox : เลือกได้มากกว่าหนึ่งตัวเลือก

คอมโพเนนต์ Checkbox จะใช้ในการสร้างรายการตัวเลือกได้เหมือนกับคอมโพเนนต์ RadioButton แต่จะแตกต่างกันตรงที่สามารถให้ผู้ใช้งานเลือกตัวเลือกได้มากกว่า 1 ตัว

พร็อพเพอร์ตี้สำคัญของ Checkbox

- Caption เป็นข้อความที่แสดงบนตัว
- Checked ถ้ากำหนดเป็น True แสดงว่ามีการเลือกที่คอมโพเนนต์ตัวนี้
- AllowGrayed ถ้ากำหนดเป็น True จะทำให้เราไม่สามรถเลือกตัวเลือกนี้ได้ โดยตัวเลือกจะเปลี่ยนเป็นสีเทา
- Alignment เป็นการกำหนดแนวการวางของข้อความบน Checkbox
- State เป็นการบอกสถานะของ Checkbox ว่าเป็นอย่างไร ถูกเลือก ไม่ถูกเลือกหรือไม่สามารถเลือกได้

2.2.35 คอมโพเนนต์ ListBox : รายการข้อมูล

คอมโพเนนต์ ListBox เป็นวิธีการหนึ่งในการจำกัดจำนวนของตัวเลือกที่เป็นไปได้ทั้งหมดโดยเก็บไว้ในรายการของ ListBox ซึ่งเมื่อเราสามารถคลิกเลือกรายการที่ต้องการได้ทันที

พร็อพเพอร์ตี้สำคัญของ ListBox

- Columns เป็นการแสดงจำนวนคอลัมน์ที่แสดงรายการข้อมูลภายใน ListBox
- ItemIndex เป็นลำดับที่ของตัวเลือกภายในคอมโพเนนต์ ListBox โดยเริ่มจาก ItemIndex ที่ 0
- MultiSelect ถ้ากำหนดเป็น true จะทำให้สามารถเลือกได้พร้อมกันครั้งละหลายๆ ตัวเลือก ซึ่งพร็อพเพอร์ตี้ตัวนี้จะไม่แสดงใน Object Inspector
- Selected เป็นพร็อพเพอร์ตี้ที่ใช้ตรวจสอบว่ารายการใดบ้างใน ListBox ที่ถูกเลือก

2.2.36 คอมโพเนนต์ ComboBox : ตัวเลือกแบบป้อนข้อมูลได้

คอมโพเนนต์ ComboBox นั้นเป็นรายการตัวเลือกเหมือนกับ ListBox แต่พิเศษกว่าตรงที่นอกจากจะเลือกรายการตามข้อมูลที่มีได้แล้ว ยังสามารถคีย์ชื่อเข้าไปเหมือนกับคอมโพเนนต์ Edit ได้ด้วย พูดง่ายๆ ก็คือเป็นการผสมกันระหว่างคอมโพเนนต์ ListBox กับ Edit นั่นเอง

2.2.37 งานพิมพ์ และรายงาน

โดยจะกล่าวถึงการแสดงผลข้อมูลออกเครื่องพิมพ์ ทั้งรูปแบบกราฟิก และตัวอักษรด้วยคอมโพเนนต์มาตรฐานของเดลไฟ รวมทั้งการพิมพ์รายงานด้วยคอมโพเนนต์ที่สนับสนุนงานการพิมพ์ด้านฐานข้อมูล เนื้อหาโดยรวมจะประกอบไปด้วย

2.2.37.1 ระบบการแสดงผลออกเครื่องพิมพ์ แนะนำสวนระบบการควบคุมการแสดงผลออกเครื่องพิมพ์ ที่เดลไฟสามารถใช้ได้ โดยจำแนกตามชนิดการติดต่อ

2.2.37.2 การใช้งานคอมโพเนนต์ Tprinter กล่าวถึงความเข้าใจพื้นฐานของการใช้งานคอมโพเนนต์ Tprinter และการนำไปประยุกต์ใช้ร่วมกับคอมโพเนนต์มาตรฐานที่ใช้ในระบบแสดงผลเพื่อที่จะสามารถแสดงผลตัวอักษรในรูปแบบต่างๆและภาพกราฟฟิกออกเครื่องพิมพ์ตามความต้องการ

2.2.37.3 การใช้งานชุดคอมโพเนนต์ TQuickReport กล่าวถึงโครงสร้าง และรายละเอียดของชุดคอมโพเนนต์ TQuickReport และการใช้คอมโพเนนท์ส่วนต่างๆสร้างเป็นรายงาน โดยเชื่อมโยงกับฐานข้อมูลที่มีอยู่เพื่อที่สามารถแสดงผลเป็นรายงานตามต้องการ

2.2.38 ระบบการแสดงผลออกของเครื่องพิมพ์

เดลไฟมีส่วนระบบควบคุมการแสดงผลออกเครื่องพิมพ์ ซึ่งสามารถจำแนกแบ่งออกตามชนิดของการติดต่อได้เป็น 2 ส่วนคือ ส่วนควบคุมการแสดงผลออกเครื่องพิมพ์ที่ติดต่อกับเอาท์พุตดีไวซ์ (Output Device)โดยตรง และส่วนควบคุมการแสดงผลออกเครื่องพิมพ์โดยอาศัยความสามารถของ DIB (Device-independent bitmaps)

2.2.38.1 การแสดงผลออกเครื่องพิมพ์ด้วยการติดต่อกับเอาต์พุตดีไวซ์

โดยตรงการติดต่อเอาต์พุตดีไวซ์โดยตรงเป็นการเปลี่ยนเส้นทางการออกเอาต์พุตข้อมูลจากจอภาพแสดงผล หรือแหล่งบันทึกข้อมูลหรือฮาร์ดดิสก์ (Hard Disk) ไปเป็นเครื่องพิมพ์ ด้วยคำสั่งภายในของเดลไฟคือ AssignPm ซึ่งเป็นการเปลี่ยนชนิดดีไวซ์สำหรับการสำหรับการบันทึกข้อมูล

2.2.38.2 การแสดงผลออกเครื่องพิมพ์ด้วยระบบ DIB

เป็นการอาศัยความสามารถในการแสดงผลของระบบวินโดว์ โดยผ่านระบบควบคุมการแสดงผล DIB (Device-independent bitmaps) ด้วยการเปลี่ยนชนิดดีไวซ์ไดรเวอร์จากจอภาพเป็นเครื่องพิมพ์ ทำให้สามารถแสดงผลออกดีไวซ์ชนิดต่างๆโดยไม่ต้องเขียนโปรแกรมส่วนการแสดงผลใหม่ และสามารถแสดงผลทั้งภาพกราฟฟิก รูปเรขาคณิต และตัวอักษรได้พร้อมๆกัน

Tprinter เป็นออบเจ็กต์ที่เดลไฟเตรียมไว้ให้ใช้แสดงผลออกเครื่องพิมพ์ผ่านระบบการแสดงผล DIB ของวินโดวส์ มีรายละเอียดดังต่อไปนี้

1.การใช้งานออบเจ็กต์ Tprinter

ออบเจ็กต์ Tprinter ถือเป็นคอมโพเนนต์พิเศษสำหรับการควบคุมการใช้งานเครื่องพิมพ์ ที่ไม่ได้บรรจุไว้ในแถบชุดคอมโพเนนต์ (Components Palette) การใช้งานจำเป็นต้องระบุเรียกยูนิต Printers ไว้ในส่วน Uses ของฟอร์มที่เรียกใช้

Tprinter มีพร็อพเพอร์ตี้ และกิจกรรมที่สำคัญดังต่อไปนี้ :

Aborted	ชนิดบูลลีน (Boolean) กำหนดเป็น TRUE ถ้าต้องการยกเลิกการพิมพ์ในปัจจุบัน
Canvas	ชนิดออบเจ็กต์ (Objects) เป็นออบเจ็กต์ที่รวบรวมเครื่องมือสำหรับแสดงผลตัวอักษร หรือวาดภาพ
Copies	ชนิดจำนวนเต็ม(Integer) ระบุจำนวนชุดที่ต้องการพิมพ์
Fonts	ชนิดออบเจ็กต์ ระบุรายการฟอนต์ที่ใช้แสดงตัวอักษร สำหรับงานพิมพ์ในปัจจุบัน
PageHeight	ชนิดจำนวนเต็ม ระบุความสูงของหน้ากระดาษที่สามารถพิมพ์ได้
PageNumber	ชนิดจำนวนเต็ม ระบุหน้าที่พิมพ์อยู่ในปัจจุบัน
Pagewidth	ชนิดจำนวนเต็ม ระบุความกว้างของหน้ากระดาษที่สามารถพิมพ์ได้
Printers	ระบุรายชื่อเครื่องพรินต์เตอร์ (Printer) ที่ติดตั้งไว้ในระบบ
Printing	ชนิดของบูลลีน แฟลกส์สำหรับสำหรับตรวจสอบสถานะการพิมพ์ ให้ค่าเป็น True หมายถึงกำลังพิมพ์
BeginDoc	กิจกรรมสำหรับการเตรียมการเรนเดอร์(Render) ข้อมูลสำหรับพิมพ์
EndDoc	กิจกรรมสำหรับระบุการสิ้นสุดการเตรียมข้อมูล และสั่งเริ่มพิมพ์ออกเครื่องพิมพ์
NewPage	กิจกรรมสำหรับขึ้นหน้าใหม่

2. ขั้นตอนการสั่งพิมพ์ด้วยออบเจ็กต์ Tprinter

การสั่งพิมพ์ด้วยออบเจ็กต์ Tprinter สามารถแบ่งเป็น 3 ขั้นตอนดังนี้

- เตรียมการพิมพ์ เรียกกิจกรรม BeginDoc ของออบเจ็กต์ Tprinter เพื่อกำหนดค่าเบื้องต้น และเตรียมการเรนเดอร์ภาพ

- เรนเดอร์ภาพ เรนเดอร์ภาพโดยการใช้ออบเจ็กต์ Tcanvas แสดงตัวอักษร หรือวาดภาพ

- สั่งพิมพ์ออกเครื่องพิมพ์ ระบุการสิ้นสุดของการเตรียมข้อมูลพร้อมกับสั่งพิมพ์ออกเครื่องพิมพ์ด้วยกิจกรรม EndDoc

2.2.39 โปรแกรม Printer

โปรแกรม Printer เป็นโปรแกรมตัวอย่างแสดงการใช้คำสั่งควบคุมให้เครื่องพิมพ์แสดงผลตามต้องการ การใช้โปรแกรมตัวอย่างนี้จึงจำเป็นต้องติดตั้งเครื่องพิมพ์ต่อพ่วงอยู่ด้วยเพื่อแสดงตัวอย่างการสั่งพิมพ์โปรแกรมมีฟังก์ชันแสดงการทำงานอยู่ 3 หมวดหมู่ด้วยกันคือ

2.2.39.1 พิมพ์ข้อความด้วยวิธีการติดต่อกับเอาต์พุตดีไวซ์โดยตรง แสดงการสั่งพิมพ์โดยการติดต่อกับเอาต์พุตดีไวซ์โดยตรง ด้วยคำสั่งมาตรฐานของเดลไฟ คือ Write และ Writeln

2.2.39.2 พิมพ์ด้วย TPrinter Object เป็นการแสดงการพิมพ์ทั้งข้อมูลชนิด Text และรูปภาพกราฟิก รวมทั้งใช้คำสั่งวาดรูปบน Tprinter Object

2.2.39.3 แสดงรายละเอียดของเครื่องพิมพ์ ในโปรแกรมส่วนนี้จะแสดงการอ่านรายละเอียดต่างของเครื่องพิมพ์ เช่น ชื่อเครื่องพิมพ์ พอร์ตการสื่อสาร ความละเอียดในการพิมพ์

2.2.40 โครงสร้างรายงาน

การสร้างรายงานด้วยชุดคอมโพเนนต์ TQuickRepot จะแตกต่างกับการใช้งานคอมโพเนนต์ TPrinter ที่ ออกแบบให้สร้างภาพที่ต้องการพิมพ์ด้วยคอมโพเนนต์ Tcanvas แบบอิสระ กล่าวคือ จะไม่สามารถใช้กิจกรรมต่าง ๆ ของคอมโพเนนต์ Tcanvas ในการวาดรูปสี่เหลี่ยม วงกลม หรือแสดงตำแหน่งแสดงผลข้อมูลต่าง ๆ แต่จะต้องใช้ชุดคอมโพเนนต์พิเศษที่เตรียมไว้ให้สำหรับสร้างเอกสาร โดยมีจุดมุ่งหมายให้สร้างอยู่ในรูปแบบรายงานมากกว่าแบบอิสระ ที่จะกำหนดรูปแบบการแสดงผลอย่างไรก็ได้เหมือนกับการใช้งาน Tprinter

TqulckReport เป็นคล้ายที่อยู่ในรูปแบบฟอร์มชนิดหนึ่งของเดลไฟ สามารถออกแบบ และจัดรูปแบบการแสดงผลในลักษณะเดียวกันกับฟอร์มมาตรฐาน (Tfom) ด้วย คอมโพเนนต์พิเศษสำหรับสร้างรายงาน โดยการกำหนดขอบเขต หรือส่วนของข้อมขล ซึ่งมีความหมายเฉพาะในแต่ละส่วน รวมเรียกว่า แบนด์ (Band) มีส่วนประกอบหลักที่สำคัญดังต่อไปนี้

1.pageHeader Band ชุดคอมโพเนนต์ที่บรรจุในส่วนนี้ จะถูกจัดพิมพ์ในทุกแผ่นทีส่วนหัวของกระดาษ ข้อมูลที่พิมพ์ในส่วนนี้ส่วนใหญ่เป็นข้อมูลที่ต้องการให้พิมพ์ออกในทุก ๆ หน้าของเอกสาร เช่น ชื่อเอกสาร,หมายเลขหน้า และวันเดือนปี เป็นต้น

2.TItle Band เป็นส่วนที่ใช้แสดงชุดข้อมูลประเภทหัวเรื่อง จะพิมพ์เฉพาะหน้าแรกที่ตำแหน่งบนส่วนหัวของกระดาษ อยู่รองจาก PageHeader เช่น ชื่อเรื่อง หรือชื่อรายงาน

3.Column Header Band ใช้แสดงคำอธิบายของข้อมูลที่จะแสดงผลตามมา จะพิมพ์ใหม่ทุกครั้งที่เปลี่ยนกลุ่มข้อมูล หรือขึ้นหน้ากระดาษใหม่ เช่น ส่วนอธิบายรายละเอียดของสินค้า อันประกอบไปด้วย รหัสสินค้า ชื่อสินค้า จำนวน และราคา เป็นต้น

4.DetaiI Band ส่วนที่ใช้แสดงผลเนื้อข้อมูล เช่น รูปภาพ ข้อความ หรือข้อมูลจากฐานข้อมูลสามารถมีกลุ่มข้อมูลย่อยต่าง ๆ เช่น Child Band หรือ SubDetail Band สำหรับแสดงผลข้อมูลคัญย่อย รายละเอียดเพิ่มเติมของข้อมูลที่แสดงอยู่ใน Detail Band เป็นต้น

5.Summary Band ใช้แสดงผลข้อมูลประเภทสรุปผลรวมต่าง ๆ ที่เกิดจากการสร้างนิพจน์ (Expresslon) การหาจำนวนข้อมูลทั้งหมด (Data count) เช่น จำนวนสินค้าทั้งหมด หรือยอดเงินรวมทั้งสิ้น

6.PageFooter Band ข้อมูลที่บรรจุในส่วนนี้จะถูกพิมพ์ส่วนล่างสุดของหน้ากระดาษ ของทุก ๆ หน้า เช่น หมายเลขหน้าที่พิมพ์

2.2.40.1 การสร้างรายงาน

การสร้างรายงานสามารถทำได้โดยเลือกเมนู Flle-->New และเลือก Report ในหน้า New โปรแกรมจะแสดงฟอร์ม QuickReportซึ่งสามารถกำหนดแบนด์ที่ต้องการใช้งานได้ในหน้าต่างObject lnspector ในส่วนของพร็อพเพอร์ตี้ Bands โดยกำหนดชนิดของแบนด์ที่ต้องการใช้ให้เป็นTRUE ซึ่งสามารถโส่ชุดคอมโพเนนต์พิเศษลงในแบนด์ส่วนต่าง ๆ เพื่อกำหนดรูปแบบการแสดงผลดังต่อไปนี้

QRLabel	คอมโพเนนต์หรับพิมพ์ข้อความชนิดตัวอักษร
QRDBText	สำหรับพิมพ์ข้อความชนิดตัวอักษรโดยติดต่อกับฐานข้อมูล
QRExpr	พิมพ์ผลของนิพจน์ ที่คำนวณจากสูตร ฟังก์ชัน หรือรับค่าจากฐานข้อมูลที่กำหนด
QRSysData	พิมพ์ผลเป็นวันที่ เวลา หมายเลขหน้า หรือไตเติลตามชนิดที่เลือก
QRMemo	พิมพ์ข้อความทีบรรจุในพร็อพเพอร์ตี้ Llnes เหมือนกับ TMemo
QRExprMemo	คอมโพเนนต์ที่รวมเอาความสามารถของ QRExpr และ Tmemo เข้าด้วยกัน โดย สามารถกำหนดรูปแบบการแสดงผลแบบฟอร์มสำหรับกรอกรายการโดยกำหนดนิพจน์ให้กับฟิลด์ข้อมูลที่แสดง ภายใต้แครื่องหมายปีกกา ({})
QRRichText	สำหรับพิมพ์ข้อมูลของเวิร์ดโปรเซสเซอร์ชนิด Rlch Text File Formatted(RTF) ที่สร้างด้วยโปรแกรม Word Pad หรือ Mlcrosoft Word
QRDBRich	ให้ผลลัพธ์เช่นเดียวกันกับ QRRichText โดยติดต่อผ่านฐานข้อมูล และอ่านข้อมูลที่บรรจุอยู่ในฟิลด์ข้อมูลชนิด BLOB Filed จากฐานข้อมูล
QRShape	สำหรับพิมพ์รูปกรอกสี่เหลี่ยม เส้นตรงชนิดต่าง ๆ เพื่อใช้ประกอบการสร้าง

QRImage สำหรับพิมพ์รูปภาพจากคอมโพเนนต์ TPicture

QRDBImage สำหรับพิมพ์รูปภาพ โดยติดต่อผ่านฐานข้อมูล และอ่านข้อมูลที่บรรจุอยู่ในฟิลด์ข้อมูลชนิต BLOB Filed จากฐานข้อมูล

QRChat สำหรับพิมพ์รูปกราฟชนิดต่าง ๆ จากชุดคอมโพเนนต์ TChart

การสร้างรายงาน จะแตกต่างจากการพิมพ์งานแบบทั่วไป เนื่องจากต้องคำนึงถึงผล หรือ ข้อมูลที่ได้รับจากงาน กล่าวคือผู้จัดทำต้องตอบคำถาม "ต้องการรายงานอะไร" เพื่อให้ได้ผลลัพธ์ตามจุดประสงค์ที่ต้องการ เพราะคุณค่าของรายงานจะขึ้นอยู่กับรูปแบบการนำเสนอข้อมูลที่พิมพ์อยู่ในเนื้อหาของรายงาน

2.2.40.2 โปรแกรม Quick Report Demonstration

โปรแกรม QRRept เป็นโปรแกรมสาธิตการใช้งาน TquickReport ในการพิมพ์ข้อมูลออกเครื่องพิมพ์ โดยโปรแกรมแบ่งรูปแบบการทำงานออกเป็น 3 ส่วนดังต่อไปนี้

1.แสดงรายละเอียดของคอมโพเนนต์ TquickReport

2.แสดงการพิมพ์ข้อมูตัวอักษร รูปสี่เหลี่ยมเรขาคณิต รูปภาพออกเครื่องพิมพ์ โดยใช้ชุดคอมโพเนนต์พิเศษ ที่กำหนดให้ใช้สำหรับการพิมพ์รายงานของชุดคอมโพเนนต์ QuickReport

3.แสดงการพิมพ์รายงานโดยอ่านข้อมูลจากฐานข้อมูล แสดงเป็นรายงาน 2 แบบ คือ

- แบบมาตรฐาน โดยใช้แบนด์มาตรฐาน
- แบบใช้ Chlld Band ช่วยแสดงผล ในการจัดรายละเอียดของชุดข้อมูลเป็นกลุ่ม

2.2.40.3 เมนูหลัก (UQRMAIN.PAS)

เมนูหลัก UQRMAIN. PAS เป็นโปรแกรมส่วนควบคุมการแสดงผลในรูปแบบต่าง ๆ มีส่วนของโปรแกรมแสดงอยู่ และมีกิจกรรมที่ คัญดังต่อไปนี้

Procedure TFqrMain.DispReportSettlng : เป็นกิจกรรมสำหรับแสดงรายละเอียดของ QuickReport1ที่ใช้ในระบบ โดยแสดงชื่อ ฟอนต์ ขนาดของตัวอักษรขนาดของกระดาษที่ใช้ และรูปแบบการวางกระดาษ ที่กำหนดไว้ในระบบ

procedure TFqrMain.Botton2Click(sender: TObject) : กิจกรรมสำหรับสั่งพิมพ์ภาพ Preview จากรายการที่เลือก โดยเรียก กิจกรรม QuickReport1. Dopreview

procedure TFqrMain.Button3Click(sender: TObject) : กิจกรรมภหรับพิมพ์ออก เครื่องพิมพ์ โดยเรียกกิจกรรม QulckReport1 DoPrint

procedure TFqrMain.Button6Click(sender: TObject) : กิจกรรมสำหรับแสดงการพิมพ์ ข้อมูลชนิดตัวอักษรด้วย TQRMemo โดยให้อ่านข้อมูลจากแฟ้มข้อมูลชนิดเท็กซ์ไฟล์ (.TXT) ให้กับกิจกรรม DopaintMemo

procedure TFqrMain.Button7click(Sender: TObject) : กิจกรรมสำหรับแสดงการพิมพ์แฟ้มข้อมูลชนิด Rich Text File Format (.RTF)โดยอ่านจากแฟ้มข้อมูลที่กำหนด

procedure TFqrMain.Button8click(Sender: TObject) : กิจกรรมสำหรับพิมพ์ภาพ Bitmap ออกเครื่องพิมพ์ ด้วยกิจกรรม DoPaintPlcture

procedure TFqrMain.BitBtn1click(sender: TObject) : กิจกรรมสำหรับเลือกแฟ้มข้อมูล ชนิดต่างๆที่ใช้แสดงตัวอย่าง คือข้อมูลชนิดเท็กซ์ไฟล์ (.TXT) ข้อมูล Rlch Text Formatted (.RTF) และแฟ้มรูปภาพ (.BMP) โดยให้ค่าชนิดของแฟ้มข้อมูลกับตัวแปร FileType

2.2.41 การสื่อสาร

เนื้อหาหลักของบทนี้จะกล่าวถึงการนำเดลไฟมาพัฒนาโปรแกรมด้านการสื่อสารด้วยคำสั่งของ Win 32 เอง และการนำคอมโพเนนต์ด้านการสื่อสารบางส่วนมาประยุกต์ใช้งาน รวมทั้งการสร้างข้อมูลที่ใช้เป็นโปรโตคอลในระดับแอปพลิเคชันเลเยอร์ ประกอบไปด้วย

1 การสื่อสารด้วยพอร์ตสื่อmsแบบอนุกรม RS-232 (serial port)

2 การสื่อสารรผ่านระบบเน็ตเวิร์กด้วยโปรโตคอล TCP/IP

3 การเคลื่อนย้ายแฟ้มข้อมูลด้วยระบบวินโดวส์เวิร์กกรุ๊ป

การสื่อสารด้วยพอร์ตสื่อสารแบบอนุกรม RS-232 (serial port) เป็นเรื่องของการใช้ประโยชน์จากอุปกรณ์มาตรฐานของระบบคอมพิวเตอร์ในการติดต่อลื่อสารกับเครื่องคอมพิวเตอร์เครื่องอื่น โดยเนื้อหาจะกล่าวถึงการควบคุมพอร์ตสื่อสารให้สามารถรับส่งข้อมูลได้

1.การสื่อสารผ่านระบบเน็ตเวิร์กด้วยโปรโตคอล TCP/IP เป็นการสื่อสารอีกระดับชั้นที่ต้องพึ่งพาอุปกรณ์การสื่อสารพิเศษอื่น เช่น แลนคาร์ด (LAN Card) เพื่อเพิ่มศักยภาพในการรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ ด้วยโปรโตคอล TCP/IP ที่มีความสำคัญเพิ่มมากขึ้นทุกวันในระบบการสื่อสารปัจจุบัน โดยเนื้อหาจะกล่าวถึงการนำคอมโพเนนต์มาตรฐานของเดลไฟชุด Client/Server มาใช้งาน

2.การเคลื่อนย้ายข้อมูลด้วยระบบวินโดวสัเวิร์กกรุ๊ป การอาศัยศักยภาพของวินโดวส์เวิร์กกรุ๊ป ที่มีมาตั้งแต่วินโดวส์เวอร์ชั่น 3.11 ในการเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์เครื่องต้นทาง ผ่านระบบเน็ตเวิร์กไปยังเครื่องปลายทาง โดยไม่จำกัดชนิดของโปรโตคอลที่นำมาใช้

การเคลื่อนย้ายแฟ้มข้อมูลผ่านระบบเน็ตเวิร์กด้วยบริการ FTP ความสามรถที่แตกแขนงออกไปของโปรโตคอล TCP/IP ที่สร้างขึ้นเพื่อช่วยในการเคลื่อนย้ายแฟ้มข้อมูลเน็ตเวิร์กที่ต่างกัน (lndependent Network) เนื้อหาในส่วนนี้จะกล่าวถึงการนำชุดคอมโพเนนต์ของเดลไฟมาใช้งานเช่นกัน

3.การสื่อสารด้วยพอร์ตสื่อสารแบบอนุกรม RS-232 (serial port)ในอดีตกาล การสื่อสารด้วยพอร์ตอนุกรม RS-232 เป็นสิ่งที่มีมาดั้งแต่ตัน เริ่มตั้งแต่สมัยการโปรแกรมบนระบบปฏิบัติการ DOS ด้วยเครื่องที่ใช้ซีพียูระดับ 8 บิต เป็นต้นมา ในครั้งนั้นการเขียนโปรแกรมควบคุมพอร์ตสื่อสารชนิดนี้ต้องเขียนด้วยภาษาระดับต่ำ เช่นภาษาเครื่อง และผู้ที่เขียนจะต้องมีความรู้ในระดับฮาร์ดแวร์เกี่ยวกับอุปกรณ์ที่ใช้ค่อนข้างดี ต่อมาเมื่อไมโครซอฟต์ออกระบบปฏิบัติการวินโดวส์รุ่น 3.x ได้มีการเพิ่มเติมความสามารถในการเข้าถึงอุปกรณ์ชนิดนี้ โดยเตรียมชุดคำสั่งระดับสูงไว้สื่อสารกับพอร์ตโดยตรง แต่ยังคงต้องการความรู้ระดับฮาร์ดแวร์ในการกำหนด และติดตั้งพอร์ตสื่อสารให้สามารถใช้งานกับระบบ

ในปัจจุบัน ระบบปฏิบัติการวินโดวส์รุ่น 9x เป็นต้นไปได้ยกเลิกการติดต่อกับพอร์ตสื่อสารแบบเดิมที่ใช้ในวินโดวส์รุ่น 3.x และเพิ่มเติมดีไวซ์ไดรเวอร์ตัวใหม่เพื่อใช้ควบคุมพอร์ตสื่อสารนี้โดยให้ผู้ใช้มองเห็นเป็นอุปกรณ์สื่อสารชนิดหนึ่ง และสามารถใช้โปรโตคอลมาตรฐานที่เตรียมไว้ติดต่อกับอุปกรณ์ชนิดนั้น โดยผู้ใช้ไม่จำเป็นต้องมีความรู้ระดับฮาร์ดแวร์ดังเดิม

ถึงแม้ระบบปฏิบัติการวินโดวส์รุ่นใหม่จะเพิ่มเติมความสามารถในการควบคุมพอร์ตอนุกรม RS-232 ให้สามารถควบคุมและใช้งานได้ง่ายขึ้นก็ตาม แต่ในบางกรณี ผู้ใช้ต้องการใช้พอร์ตสื่อสารชนิดนี้ติดต่อกับอุปกรณ์ชนิดพิเศษอื่นๆ ที่วินโดวส์ไม่ได้กำหนดให้เป็นมาตรฐาน เช่น ติดต่อกับฮสต์คอมพิวเตอร์ในแบบอิสระ, การติดต่อกับตู้ชุมสายโทรศัพท์ PABX (prlvate AutomatlC Branch Exchange) การต่อวงจรควบคุมอุปกรณ์ไฟฟ้า หรืออิเล็กทรอนิกส์อื่นๆ ทำให้ผู้ใช้มีความจำเป็นต้องเขียนโปรแกรมระดับต่ำกว่าในการสื่อสารกับอุปกรณ์ชนิดต่างๆดังนี้

2.2.41.1 เตรียมการ

ก่อนจะเริ่มการใดๆ ต้องรู้จักกับตัวฮาร์ดแวร์จริงชนิดนี้เป็นอันดับแรก ลักษณะของพอร์ตสื่อสารทางด้านกายภาพชนิดนี้เป็นรูปสี่เหลี่ยมคางหมูมุมมน ติดตั้งอยู่ด้านหลังเครื่องคอมพิวเตอร์ มีเข็มเล็กๆ ยื่นออกมาจากฐานจำนวน 9 เข็ม สำหรับพอร์ตสื่อสารชนิด 9 พิน เราเรียกพอร์ตชนิดนี้ว่า “ หัวคอนเน็กเตอร์ดีบีเก้าตัวผู้ “ (DB 9 connector Male-Type) และอีกแบบมีเข็มจำนวน 25 เข็ม สำหรับพอร์ตสื่อสารชนิด 25 พินที่เราเรียกว่า “หัวคอนเน็กเตอร์ ดีบียี่สิบห้า” (DB 25 connector Male-Type) โดยปกติบนเครื่องคอมพิวเตอร์ทั่วไป จะกำหนดพอร์ตสื่อสารชนิด 9 พินเป็นพอร์ตสื่อสารที่ 1 (COM1) พอร์ตสื่อสารชนิด 25 พินเป็นพอร์ตสื่อสารที่ 2 (COM2) แต่ก็มีความเป็นไปได้ทีทั้งสองพอร์ตจะเป็นชนิด 9 พินแบบเดียวกัน โดยปกติเราจะใช้พอร์ตสื่อสารที่1 ชนิด 9 พินนี้ ต่อกับเมาส์ หรืออุปกรณ์ชี้นำอื่นๆ

การตรวจสอบทางด้านลอจิก บนระบบปฏิบิตการวินโดวส์ 9x ว่ามีการติดตั้งพอร์ตเหล่านี้ไว้หรือไม่สามารถทำได้โดยตรวจสอบจากไอคอนซิสเต็มในหน้าต่างคอนโทรลพาเนล โดยเลือกแสดงผลที่หน้า Devlce Manager และดับเบิลคลิกที่ “Port (COM & LPT)” ระบบ จะแสดงรายการ Communlcatlon port ที่ติดตั้งไว้ในระบบ และหากดับเบิลคลิกที่รายการพอร์ตสื่อสารอีกครั้ง ระบบจะแสดงรายละเอียดทรัพยากรต่าง ๆ ของพอร์ตสื่อสาร ในกรณีที่ไม่พบพอร์ตสื่อสารใดๆ ติดตั้งอยู่ ขอให้ตรวจสอบจากหนังสือคู่มือเครื่องคอมพิวเตอร์ที่ใช้ เพื่อทำการติดตั้งพอร์ตก่อนเริ่มดำเนินการต่อไป

2.2.41.2.สายสัญญาณ

สายสัญญาณเป็นปัจจัยที่สำคัญอยางหนึ่งในการสื่อสารผ่านพอร์ตอนุกรม RS 232 เนื่องจากเป็นสิ่งที่เชื่อมโยงให้เครื่องต้นทาง และปลายทางสามารถติดต่อถึงกันได้การส่งข้อมูลระหวางเครื่องคอมพิวเตอร์ให้ได้ผลลัพธ์ที่สมบูรณ์ ควรใช้ศาย Null Modem ที่มีฉนวน (shleld) หุ้มสายสัญญาณ(wire) ที่อยู่ด้านใน เพื่อป้องกันสัญญาณรบกวน แต่ในกรณีที่ไม่สามารถหาสายสัญญาณดังกล่าวได้ เราสามารถที่จะสร้างสายสัญญาณขึ้นเอง โดยต้องทำความเข้าใจก่อนว่าสายสัญญาณชนิดนี้มีอัตราความเสี่ยงต่อความ

ผิดพลาดของข้อมูลค่อนข้างสูง เนื่องจากอาจเกิดจากสัญญาณรบกวนจากภายนอก และความไม่ได้มาตรฐานของสายที่ใช้ ดังนั้นขอแนะนำให้ใช้เพื่อทดสอบโปรแกรมเท่านั้นในการสร้างสัญญาณชั่วคราวนี้ สามารถใช้สายโทรศัพท์แบบบ 4 เส้นสัญญาณ มีลักษณะเป็นสายกลม มีเส้นสัญญาณหุ้มปลอกสีแดง เหลือง เขียวและดำอยู่ภายใน การใช้สายสัญญาณประเภทนี้ต้องพึงระวังเรื่องสัญญาณรบกวนเป็นพิเศษ เนื่องจากไม่มีฉนวนป้องกันสัญญาณรบกวน สำหรับการทดสอบครั้งนี้ เนื่องจากเครื่องโฮสต์คอมพิวเตอร์มีพอร์ตสื่อสารแบบ DB-25 และเครื่องคอมพิวเตอรที่ใช้ติดต่อกับโฮสต์มีพอร์ต สื่อสาร DB-9 วางอยู่ การสร้างสายสัญญาณจึงต้องใช้หัวสัญญาณชนิด DB-25 ด้านหนึ่งเพื่อต่อเข้ากับพอร์ตของโฮสต์ และ DB-9 อีกด้านหนึ่งเพื่อต่อเข้ากับพอร์ตสื่อสารของเครื่องคอมพิวเตอร์ที่โช้ ฟิลด์ข้อมูลที่ใช้มีความหมายดังต่อไปนี้

hCommFile	ใช้บันทึกค่าแฮนเดิล (Handle) ที่ใช้ติดต่อกับพอร์ตสื่อสาร
commPort	ระบุหมายเลขพอร์ตอนุกรมที่ต้องการเปิดใช้ สามารถมีค่าเป็น 1 2,3,4 ขึ้นอยู่กับพอร์ตที่สามารถเปิดใช้
rxBuffer	เก็บค่าขนาดของหน่วยความจำภายในที่ใช้เป็นที่พักในการรับข้อมูล จะกำหนด ให้มีขนาดใหญ่กว่าเฟรมข้อมูลที่ใช้
txBuffer	เก็บค่าขนาดของหนวยความจำภายในสำหรับพักข้อมูลที่ใช้ส่งไปปลายทาง
baudRate	เก็บค่าความเร็วที่ใช้ในการติดต่อกับพอร์ตสื่อสารปลายทาง ค่าที่กำหนดได้ คือ
	CBR_110 ความเร็ว 11o bps — CBR_19200 ความเร็ว 19200 bps
	CBR_300 ความเร็ว 300 bps — CBR_38400 ความเร็ว 38400 bps
	CBR_600 ความเร็ว 600 bps — CBR_56000 ความเร็ว 56000 bps
	CBR_1200 ความเร็ว 120o bps — CBR_57600 ความเร็ว 57600 bps
baudRate	CBR-2400 ความเร็ว 240o bps — CBR_115200 ความเร็ว 115200 bps
	CBR_4800 ความเร็ว 4800 bps — CBR_128000 ความเร็ว 128000 bps
	CBR_9600 ความเร็ว 9600 bps — CBR_256000 ความเร็ว 256000 bps
	CBR_14400 ความเร็ว 14400 bps
byteSize	ระบุจำนานบิตที่ใช้ในข้อมูลแต่ละไบต์ มีค่าระหว่าง 4-8 บิต
parity	ระบุชนิดพาริตี้ที่ใช้เช็คข้อมูล ค่าที่กำหนดได้ คือ
	EVENPARITY ชนิด Even parl — MARKPARITY ชนิด Mark parlty
	NOPARITY ไม่เช็ค Parity — ODDPARITY ชนิด Odd parity
stopBits	ระบุจำนวนสต็อปบิตที่ใช้
	ONESTOPBIT 1 stop bit
	ONE5STOPBITS 1.5 stop bits
	TWOSTOPBIT 2 stop bits

2.2.41.3 สายสัญญาณ RS-485

RS-232, RS-422, RS-423 และ RS-485 เป็นการสื่อสารแบบพอร์ตอนุกรม สำหรับคอมพิวเตอร์และอุปกรณ์ต่างๆ ถ้าพูดถึงRS-232คงจะรู้จักกัน เพราะเป็นฮาร์ดแวร์มาตราฐานที่ติดมากับเครื่อง Desktop (ความจริงในสมัยก่อนเครื่องNotebookก็มีRS-232

RS-232 จะมีข้อจำกัดอยู่หลายอย่าง เช่น ความยาวของสายต้องไม่เกิน 50 ฟุต และความเร็วสูงสุดอยู่ที่ 20 kbs ซึ่งไม่เพียงพอสำหรับการสื่อสารที่ต้องเดินสายไกล/ความเร็วสูง ต่อมาได้มี RS-485 มาแทนที่ RS-232

ปัญหาหลักของ RS-232 คือไม่ทนต่อ Noise เนื่องจากข้อมูลในสายรับและสายส่ง ต้องเปรียบเทียบระดับสัญญาณกับ GND เมื่อ GND ถูกรบกวนทำให้ GND เปลี่ยนไปจากเดิม แต่ RS485 ไม่ได้ใช้การอ้างอิงสัญญาณกับ GND RS-485 ใช้ความแตกต่างระหว่างสาย 2 สาย (A และ B) เป็นตัวบอกว่า Logic "1" หรือ Logic "0" วิธีนี้จะป้องกัน GND loop ที่เกิดขึ้น

1.การใช้ไมโครคอนโทรลเลอร์สื่อสารแบบ RS485

RS-485 เป็นการรับส่งแบบ Half-Duplex การเขียนโปรแกรมจะกำหนดให้มี Master 1 ตัวเพื่อคอยจัดคิวการสื่อสารใน Network และให้อุปกรณ์ที่เหลือเป็น Slave โดย Slave แต่ละตัวจะมี Address ของตัวเอง เวลาที่ Master ต้องการจะสื่อสารกับ Slave ทำได้โดย ส่ง Address ที่ต้องการจะสื่อสารออกไป แล้วตามด้วยฟังก์ชัน Slave ทุกตัวจะรับข้อมูลได้เหมือนกัน Slave จะเช็คดูว่า Address นั้นใน Address ของตัวเองหรือไม่ ถ้าเป็น Address ของตัวเองก็จะทำการตอบข้อมูลกลับตามที่ Master ต้องการ

2. โปรโตคอลที่ใช้ในการสื่อสารใน RS-485

เราสามารถกำหนดโปรโตคอลเองได้ว่าจะให้มีลักษณะยังไง หรือจะใช้ Open โปรโตคอลก็ได้ เช่นโปรโตคอล MODBUS ที่นิยมใช้ใน PLC งานอุตสาหกรรม IC ที่นิยมใช้แปลงสัญญาณ UART <—> RS-485 จะเป็นเบอร์ SN75176 มีราคาถูก สามารถต่อได้มากสุด 32 Node

3.คำแนะนำในการเขียนโปรแกรม

โดยปกติแล้วเราจะ Jump RE และ DE ของ SN75176 เข้าด้วยกัน เวลาจะส่งข้อมูลออกไปต้องให้ MCU ส่ง "1" มาที่ขา RE และ DE เพื่อ Enable การส่งและเมื่อส่งข้อมูลเสร็จแล้วต้องส่ง "0" มาที่ขา RE และ DE เพื่อรอรับข้อมูล ใน Bus RS-485 ถ้ามีตัวใดตัวหนึ่ง Enable DE ไว้ตัวที่เหลือจะไม่สามารถส่งข้อมูลได้เลยและเมื่อส่งข้อมูลเสร็จแล้วต้องส่ง "0" มาที่ขา RE และ DE เพื่อรอรับข้อมูล ใน Bus RS-485 ถ้ามีตัวใดตัวหนึ่ง Enable DE ไว้ตัวที่เหลือจะไม่สามารถส่งข้อมูลได้เลย

2.3 ทฤษฎีที่เกี่ยวกับแสงอินฟราเรด

รังสีอินฟราเรดมีลักษณะเช่นเดียวกับแสงธรรมชาติ รวมถึงคลื่นวิทยุ และรังสีเอกซ์ ซึ่งมีการแผ่รังสีออกมาในรูปคลื่นความถี่

คุณสมบัติต่างๆ ของแสงอินฟราเรด มีดังนี้ คือ

1. เป็นแสงที่ไม่สามารถมองเห็นได้ด้วยตาเปล่า
2. เดินทางในอากาศได้ด้วยความเร็ว 3×10^{8} เมตรต่อวินาที
3. เกิดการสะท้อน และหักเหของแสงได้

4. มีความยาวคลื่นที่ใช้ประมาณ 780-1200 นาโนเมตร

ในการส่งแสงอินฟราเรดจากตัวส่งแสงออกมายังตัวรับแสงจะต้องให้ตัวรับแสงวางอยู่ตรงกับตัวส่งแสงเพราะถ้าหากวางไม่ตรงกันตัวรับแสงจะไม่สามารถรับแสงได้เนื่องจากแสงเดินทางเป็นเส้นตรง

2.3.1 ตัวส่งแสงอินฟราเรด

2.3.1.1 แอลอีดี

แอลอีดีที่ให้แสงสีแดงนั้นเป็นไดโอดที่ทำมาจากสารกึ่งตัวนำที่เรียกว่าแกลเลียมอะเซไนด์ (GaAs) ซึ่งมีประสิทธิภาพการใช้งานและความเชื่อถือสูงจะปล่อยแสงออกมาอยู่ในย่านความยาวคลื่นประมาณ 940 นาโนเมตร คุณสมบัติทั่วไปของแอลอีดีก็เหมือนกับของไดโอดโดยในขณะไบแอสตรง มันจะให้กระแสไหลผ่านได้และจะเกิดแรงดันตกคร่อมตัวมัน มีค่าประมาณ 0.6-1 โวลท์ และในขณะไบแอสกลับแอลอีดีจะมีค่าแรงดันพังที่ต่ำกว่าไดโอดธรรมดา

นอกจากแอลอีดีที่เปล่งแสงสีแดงแล้ว เรายังพบแอลอีดีที่ให้แสงสีอื่นอีกมากมาย เช่น แสงสีเขียว สีเหลือง สีแสด เป็นต้น การที่แอลอีดีให้แสงสีแตกต่างกันนี้ขึ้นอยู่กับชนิดของเนื้อสารกึ่งตัวนำที่ใช้หรือขึ้นอยู่กับแถบพลังงานช่องว่างนั่นเองเพราะความยาวคลื่นของแอลอีดีจะขึ้นอยู่กับแถบพลังงานช่องว่างของสารโดยตรง ประโยชน์ของแอลอีดีในปัจจุบันมีมากมาย โดยสิ่งที่เห็นได้ชัดก็คือการใช้แอลอีดีเป็นอุปกรณ์จำพวกภาคแสดงผลของเครื่องคิดเลข เครื่องมือวัด ฯลฯ

แสงที่เปล่งออกมาจากตัวไดโอดเปล่งแสง (LED) แบ่งออกได้เป็น 2 ชนิด คือ แสงที่ตาคนมองเห็น และแสงที่ตามองไม่เห็น

ก) แสงที่ตาคนมองเห็น ไดโอดเปล่งแสง (LED) ที่เปล่งแสงชนิดที่ตาคนมองเห็น 3 สี คือ สีแดง สีเขียว และสีเหลือง บางครั้งอาจออกสีส้ม ซึ่งก็คือสีเหลืองเข้ม ไดโอดเปล่งแสงชนิดนี้เรียกว่า ไลท์อีมิตติ้งไดโอด (Light Emitting Diode) หรือ LED เหมือนชื่อที่กล่าวมา

ข) แสงที่คนมองไม่เห็น ไดโอดเปล่งแสง (LED) ที่เปล่งแสงที่ตาคนมองไม่เห็นเพราะเป็นแสงที่อยู่ในย่านอินฟราเรด (Infrared Light) ซึ่งขณะที่เปล่งแสงออกมาตาคนจะมองไม่เห็น ไดโอดเปล่งแสงชนิดนี้เรียกว่า อินฟราเรดอิมิตติ้งไดโอด (Infrared Emitting Diode) หรือ IRED แต่สามารถเรียกชื่อแบบรวมๆกันว่า ไดโอดเปล่งแสงหรือ LED

แสงที่ถูกเปล่งออกมาจากไดโอดเปล่งแสง (LED) ที่มีสีต่างกันนั้นขึ้นอยู่กับเนื้อสารกึ่งตัวนำที่นำมาใช้ผลิต เมื่อใช้เนื้อสารก่งตัวนำต่างกัน เนื้อที่ใช้และสีที่ได้มีความแตกต่างกันดังนี้

1. ใช้สารแกลเลียมอาร์ซีไนด์ (Gallium Arsenide) ใช้ตัวย่อ GaAs ไดโอดเปล่งแสงอินฟราเรดออกมา
2. ใช้สารแกลเลียมอาร์ซีไนด์ ฟอสไฟต์ (Gallium Arsenide Phosphide) ใช้ตัวย่อ GaAsP ไดโอดเปล่งแสงสีแดง หรือสีเหลืองออกมา
3. ใช้สารแกลเลียมฟอสไฟต์ (Gallium Phosphide) ใช้ตัวย่อ GaP ไดโอดเปล่งแสง (LED) จะเปล่งแสงสีแดงหรือสีเขียวออกมา

รูปที่ 2.4 ตัวอย่างเซนเซอร์ด้านรับและส่ง

2.3.2 ลักษณะไดโอดเปล่งแสงแบบอินฟราเรด

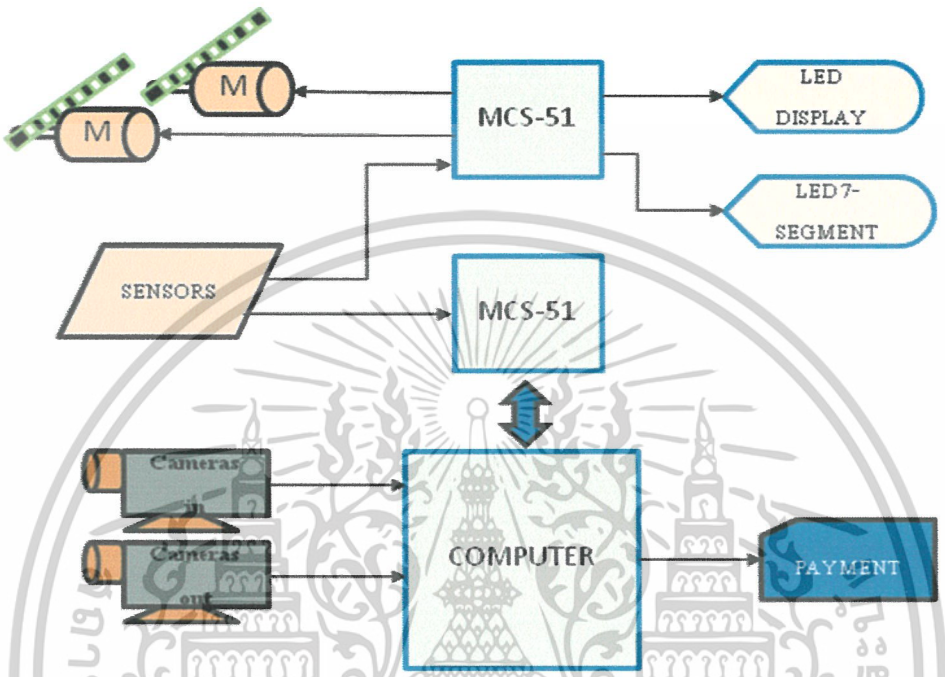
แสงที่เปล่งออกมาจากตัวไดโอดเปล่งแสง (LED) เกิดขึ้นตรงบริเวณรอยต่อของสารชนิด P และสารชนิด N เกิดการกระจัดกระจายออกรอบตัว ทำให้ความเข้มของแสงลดลงการใช้ไดโอดเปล่งแสง (LED) จำเป็นต้องมีความเข้มของแสงรวมไปในทิศทางใดทิศทางหนึ่งเฉพาะ ยิ่งถ้าเป็นไดโอดเปล่งแสง (LED) ที่เปล่งแสงอินฟราเรดออกมา ยิ่งมีความจำเป็นเพราะนิยมนำไปใช้กับระบบควบคุมระยะไกล (Remote Control) ต้องให้แสงเคลื่อนที่ไปในทิศทางเดียว ดังนั้นในส่วนของตัวถังไดโอดเปล่งแสง (LED) จึงต้องมีส่วนสะท้อนแสง และเลนส์รวมแสงเพิ่มเข้าไป

2.3.3 ตัวรับแสงอินฟราเรด

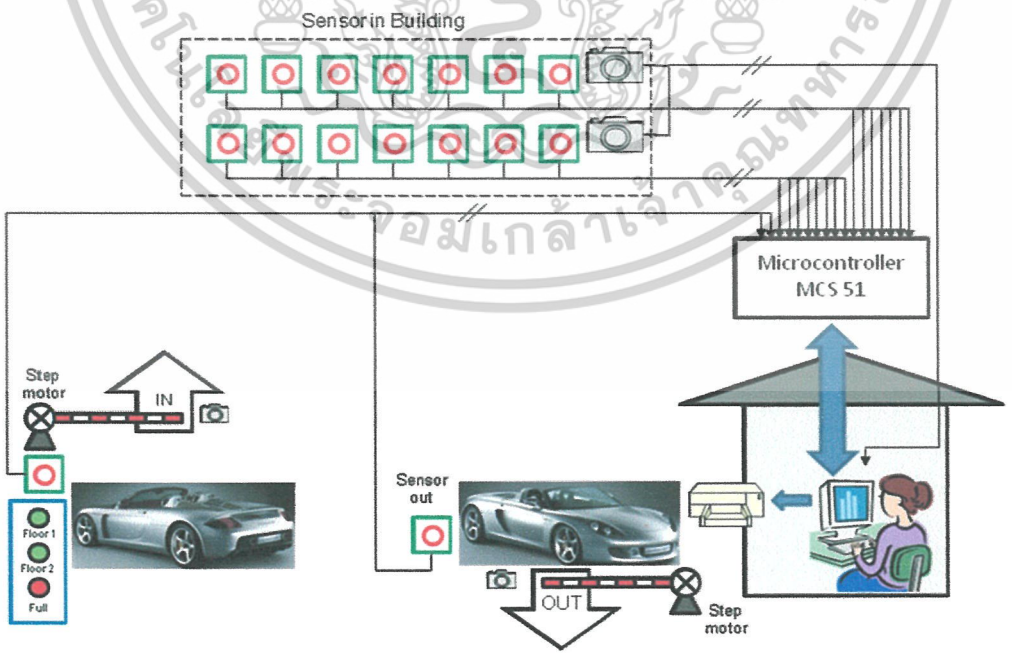
อุปกรณ์ตัวรับแสง หมายถึง อุปกรณ์ที่สามารถเปลี่ยนพลังงานแสงให้แปรค่ากับค่าขอพลังงานทางไฟฟ้าได้ โดยตัวอุปกรณ์จะต้องประกอบด้วยแสงกึ่งตัวนำซึ่งจะนำมาต่อเชื่อมให้เกิดเป็นรอบต่อหรือเป็นเนื้อสารกึ่งตัวนำอย่างเดียวก็ได้

บทที่ 3
การคำนวณและการสร้าง

โครงงานนี้เป็นการนำเอาไมโครคอนโทรลเลอร์มาประยุกต์ใช้ในการทำงานของระบบที่จอดรถอัตโนมัติ โดยจะมีขั้นตอนกานทำงานดังบล็อกไดอะแกรมด้านล่างนี้



รูปที่ 3.1 แสดงบล็อกไดอะแกรมของโครงงาน



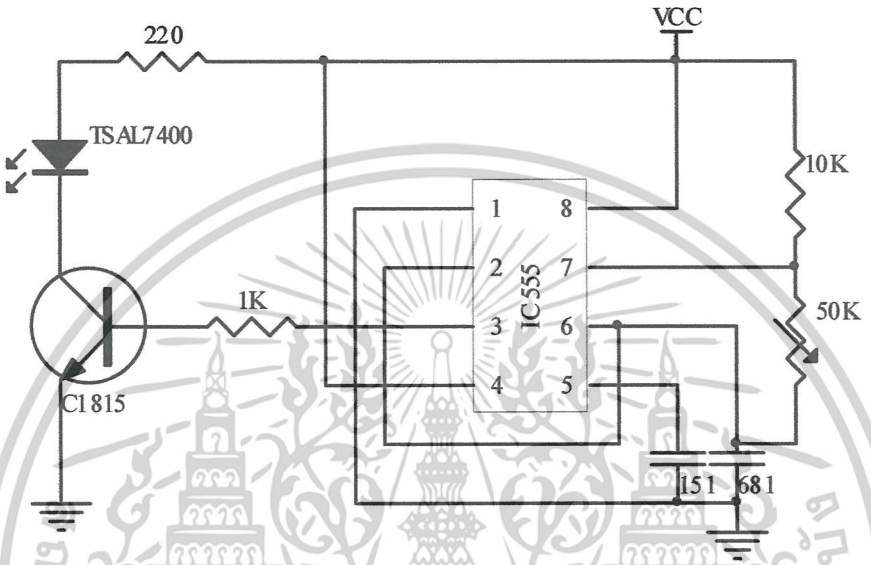
รูปที่ 3.2 แสดงแผนผังของโครงงาน

3.1โครงสร้างทางฮาร์ดแวร์ (Hardware)

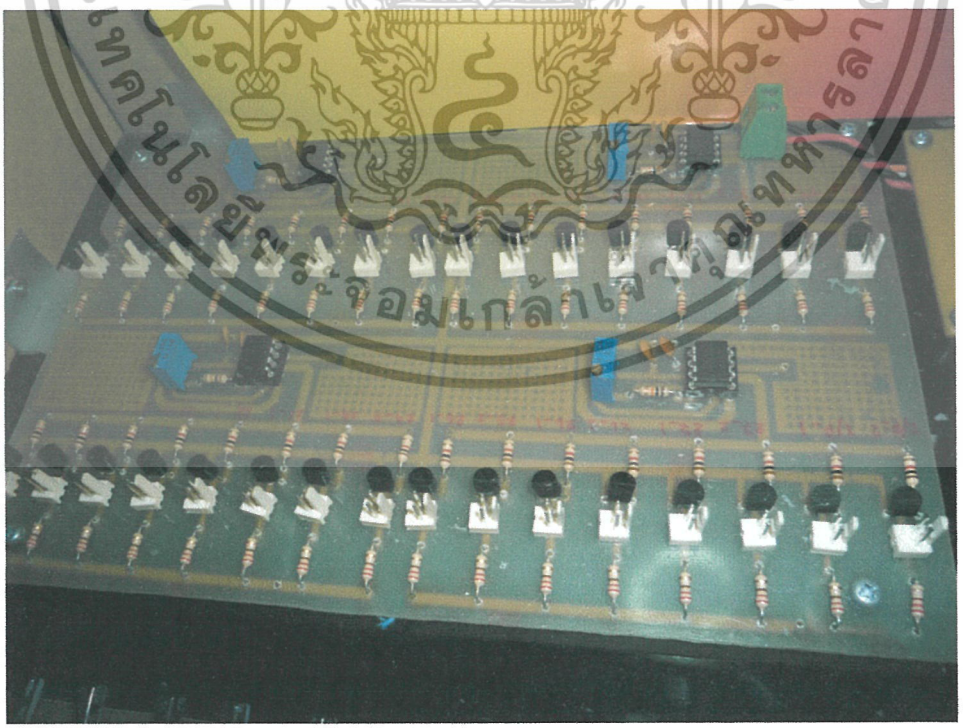
3.1.1 วงจรเซนเซอร์

3.1.1.1 วงจรเซนเซอร์ด้านส่ง

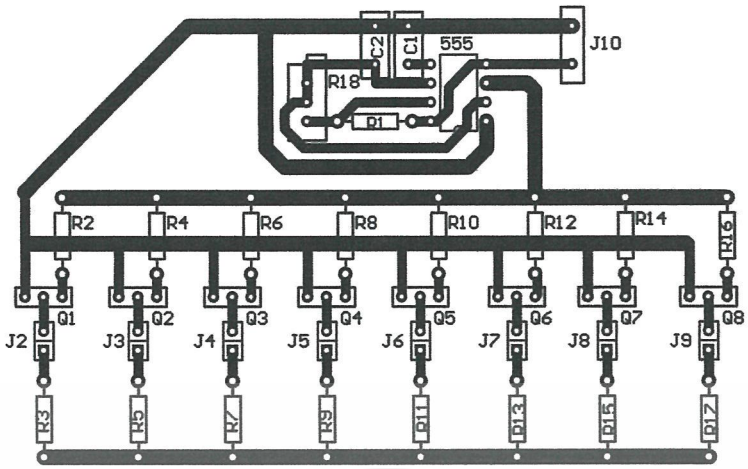
โดยในวงจรที่ใช้ต้องการสร้างพัลส์ความถี่ประมาณ 38 kHz เพื่อให้ตรงกับความถี่ของวงจรทางด้านรับสัญญาณอินฟราเรดโดยสัญญาณพัลส์ที่สร้างได้จะถูกส่งไปยังวงจรไดรว์ โดยใช้เพาเวอร์ทรานซิสเตอร์เบอร์ 2SC1815



รูปที่ 3.3 แสดงวงจรอินฟราเรดทางด้านส่ง



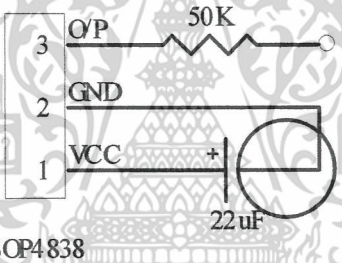
รูปที่ 3.4 แสดงแผงวงจรของอินฟราเรดทางด้านส่ง



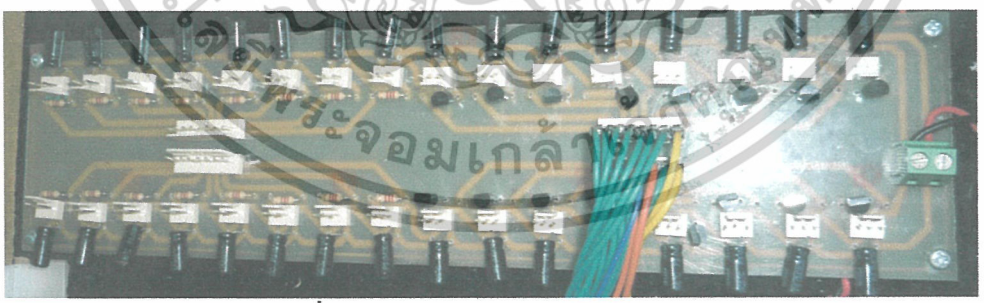
รูปที่ 3.5 แสดงลายวงจรของอินฟราเรดทางด้านส่ง

3.1.1.2 วงจรเซนเซอร์ด้านรับ

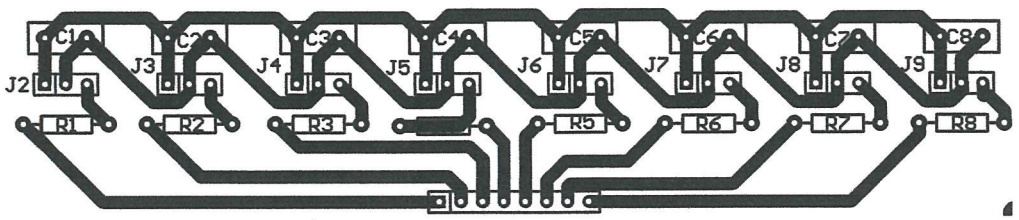
ส่วนวงจรทางด้านรับสัญญาณอินฟราเรดจะใช้โฟโต้ไดโอด (Photo Diode)เบอร์ 4838 ซึ่งสามารถรับสัญญาณอินฟราเรดที่ความถี่ 38 kHz ซึ่งภายในโฟโต้ไดโอดเบอร์ 4838 จะประกอบด้วยส่วนรับสัญญาณอินฟราเรด และภาคขยายสัญญาณที่เอาท์พุท



รูปที่ 3.6 แสดงวงจรเซนเซอร์ด้านรับ



รูปที่ 3.7 แสดงแผงวงจรเซนเซอร์ด้านรับ



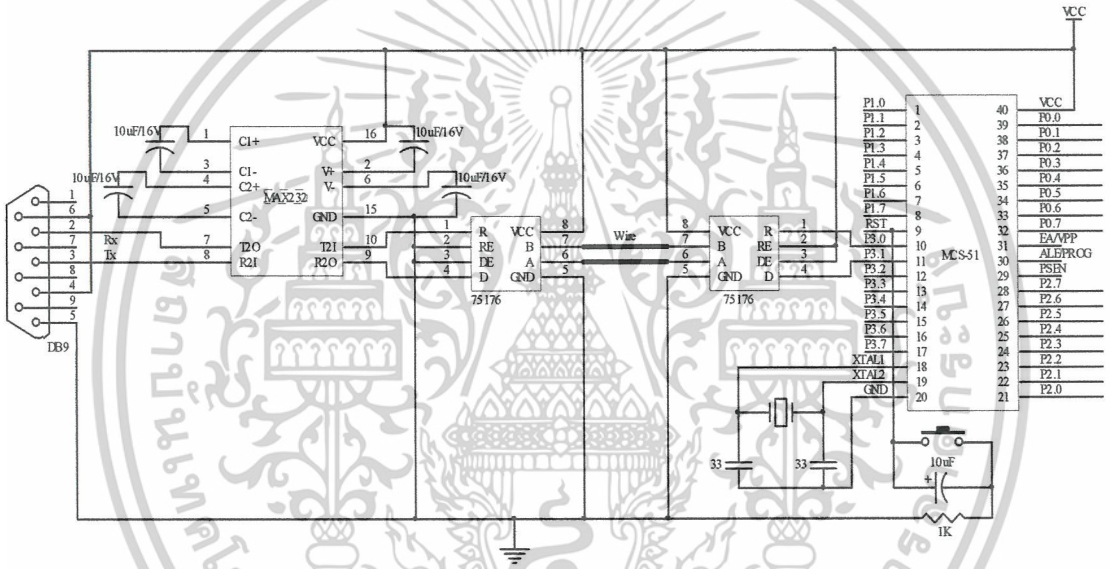
รูปที่ 3.8 แสดงลายวงจรของเซนเซอร์ด้านรับ

3.1.2 ส่วนของไมโครคอนโทรลเลอร์

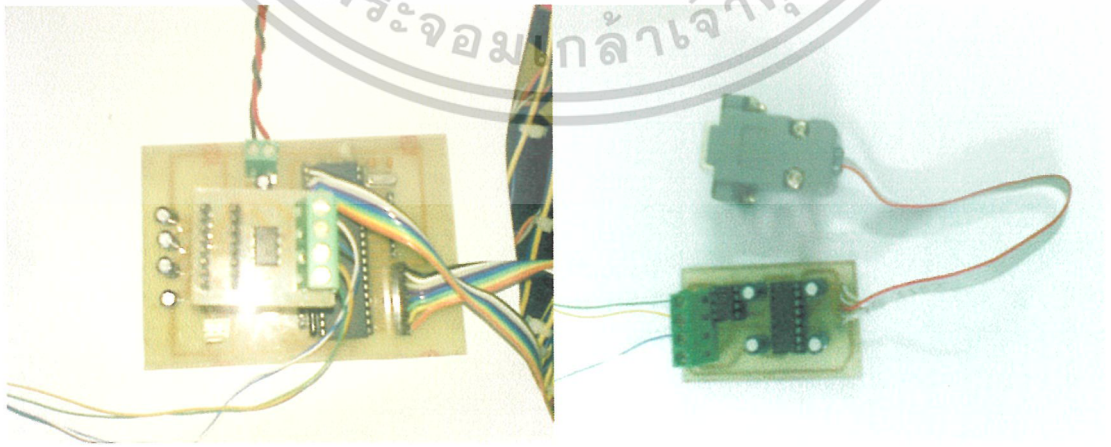
ในส่วนของไมโครคอนโทรลเลอร์จะแบ่งการทำงานออกเป็น 2 ส่วน ส่วนแรกเป็นไมโครคอนโทรลเลอร์ที่ใช้ในการส่งข้อมูลจากเซนเซอร์ไปยังคอมพิวเตอร์ และ ส่วนที่สองเป็นไมโครคอนโทรลเลอร์ที่ใช้ในการแสดง Display ของการทำงานในอาคารที่จอดรถเบื้องต้นรวมไปถึงควบคุมมอเตอร์บริเวณทางเข้าและทางออก ซึ่งทั้งสองส่วนมีรายละเอียดดังนี้

3.1.2.1 ไมโครคอนโทรลเลอร์ส่วนแรก

ไมโครคอนโทรลเลอร์ส่วนแรก เป็นส่วนที่ใช้ควบคุมการส่งข้อมูลที่รับค่ามาจาก Sensor ให้แก่คอมพิวเตอร์เพื่อนำไปคำนวณต่อไป โดยการส่งข้อมูลจะใช้ มาตรฐาน RS-485 ในการส่งข้อมูล การส่งจะส่งข้อมูลออกมาจากไมโครคอนโทรลเลอร์ผ่าน ไอซี 75176 ออกมายังสายส่งและรับด้วย ไอซี75176 อีกตัวหนึ่งทางด้านรับ ไอซี 75176 จะส่งข้อมูลไปยังคอมพิวเตอร์ผ่านทางพอร์ทอนุกรมอีกทีผ่านทาง ไอซี MAX232 เพื่อเปลี่ยนกลับมาเป็นมาตรฐาน RS-232



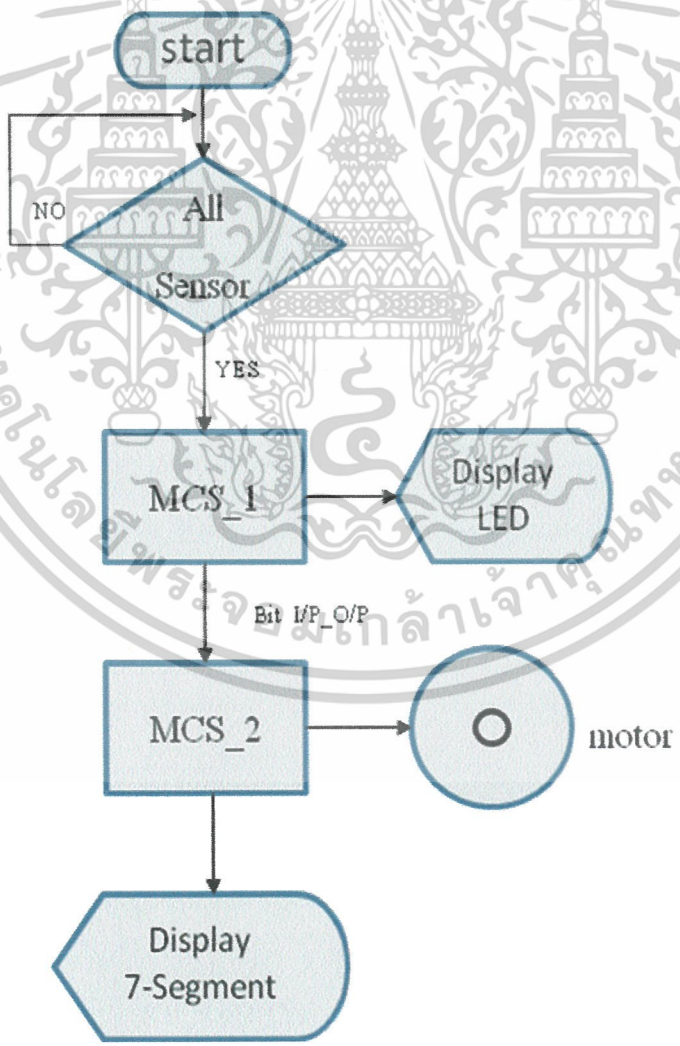
รูปที่ 3.9 แสดงวงจรการทำงานของไมโครคอนโทรลเลอร์ ส่วนแรก



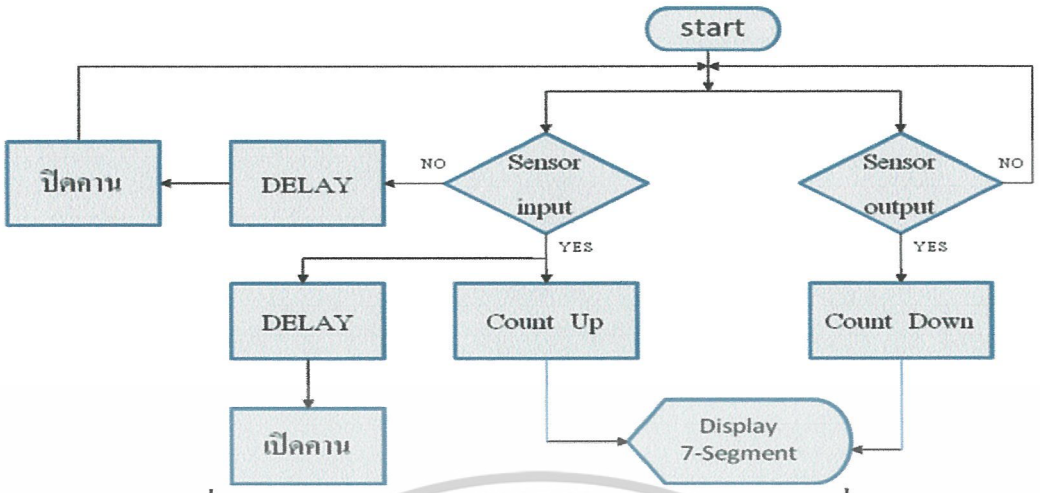
รูปที่ 3.10 แสดงแผงวงจรของไมโครคอนโทรลเลอร์ ส่วนแรก

3.1.2.2 ไมโครคอนโทรลเลอร์ส่วนที่สอง

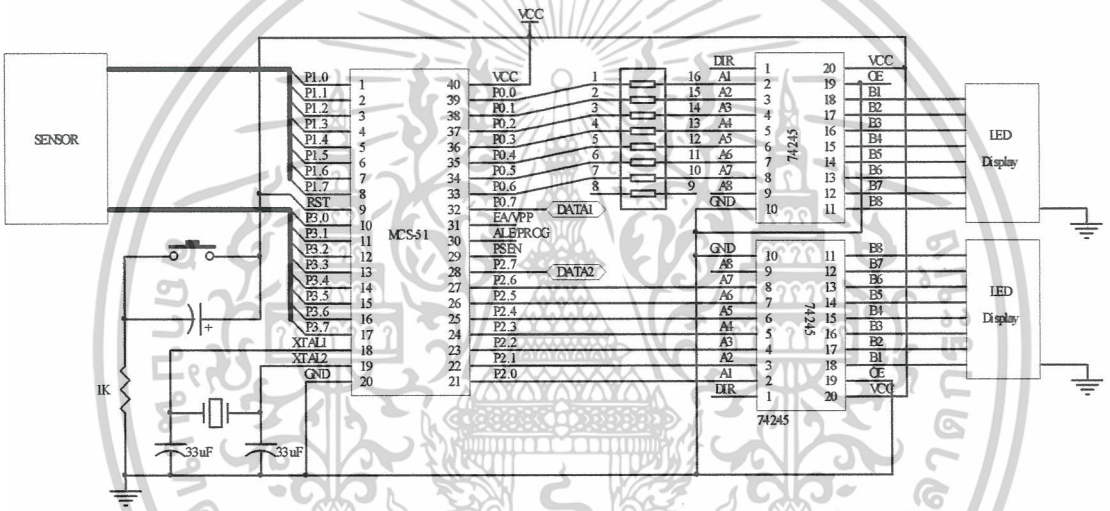
ในไมโครคอนโทรลเลอร์ส่วนที่สอง จะมีไมโครคอนโทรลเลอร์ 2 ตัวมาทำงานร่วมกัน โดยที่จะรับค่ามาจาก Sensor เช่นกันกับส่วนแรก โดยไมโครคอนโทรเลอร์ตัวแรกจะเป็นตัวรับค่าจาก Sensor เพื่อมาแสดงยัง LED แสดงตำแหน่งของรถที่จอดอยู่ในอาคารทั้ง 14 คัน ส่วนทางเข้าและทางออกเมื่อรับค่าเข้ามาจะส่งไปยัง ไมโครคอนโทรเลอร์ตัวที่สอง ซึ่งเมื่อได้รับค่าจากไมโครคอนโทรเลอร์ตัวแรก จะนำค่าที่ได้ไปเช็คสถานะเพื่อนำไปประมวลผล 2 ขั้นตอน ขั้นแรกจะนับจำนวนรถที่เข้ามาทั้งหมดภายในอาคารและแสดงออกทาง 7-Segment โดยเมื่อมีรถเข้าจะนับขึ้นและมีรถออกจะนับลง ขั้นตอนที่สองจะควบคุมการทำงานของมอเตอร์บริเวณทางเข้า โดยที่มอเตอร์จะทำงานทุกครั้งที่มีรถเข้ามายังทางเข้าโดยจะหน่วงเวลาไว้ชั่วขณะก่อนที่จะทำงานหรือเปิดคานกั้นขึ้นเพื่อให้เวลากล้องได้ถ่ายภาพ เมื่อใดที่รถเข้ามาครบ 14 คัน ลานจอดรถเต็มมอเตอร์จะหยุดทำงาน จนกว่าจะมีรถออกจากลานจอดรถคือมีรถน้อยกว่า14 คันในลานจอดรถ มอเตอร์จำจะทำงานอีกครั้งและการทำงานของไมโครคอนโทรเลอร์ทั้งสองส่วนจะเป็นแบบนี้ไปเรื่อยๆ



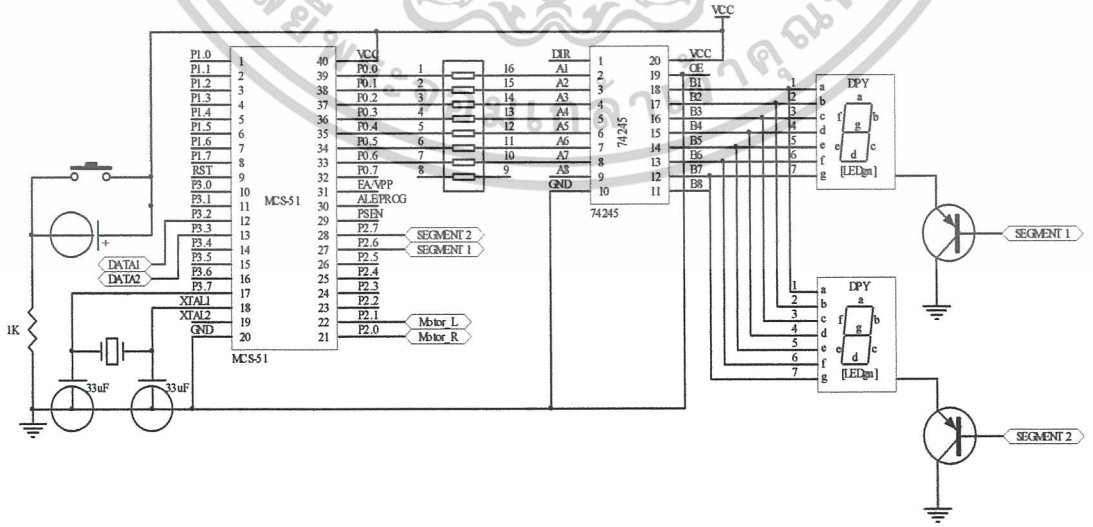
รูปที่ 3.11 แสดงแผงผังการทำงานของไมโครคอนโทรลเลอร์ส่วนที่สอง



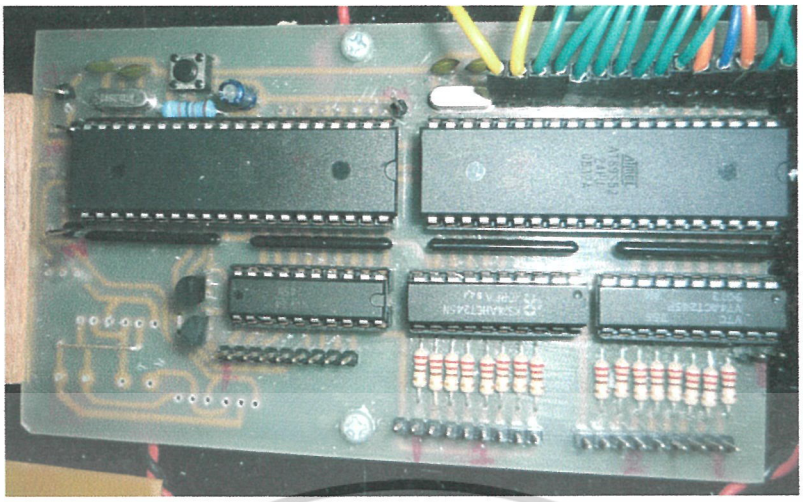
รูปที่ 3.12 แสดงแผนผังการทำงานของไมโครคอนโทรลเลอร์ตัวที่สอง



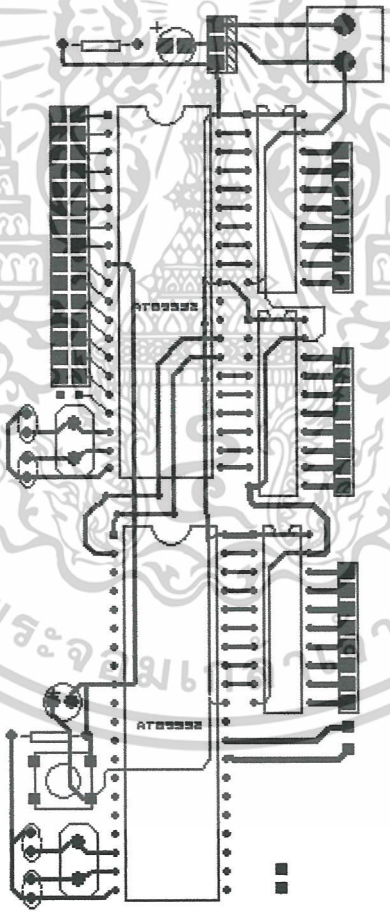
รูปที่ 3.13 แสดงวงจรการทำงานของไมโครคอนโทรลเลอร์ ส่วนที่สอง ตัวที่ 1



รูปที่ 3.14 แสดงวงจรการทำงานของไมโครคอนโทรลเลอร์ ส่วนที่สอง ตัวที่ 2



รูปที่ 3.15 แสดงแผงวงจรของไมโครคอนโทรลเลอร์ ส่วนที่สอง



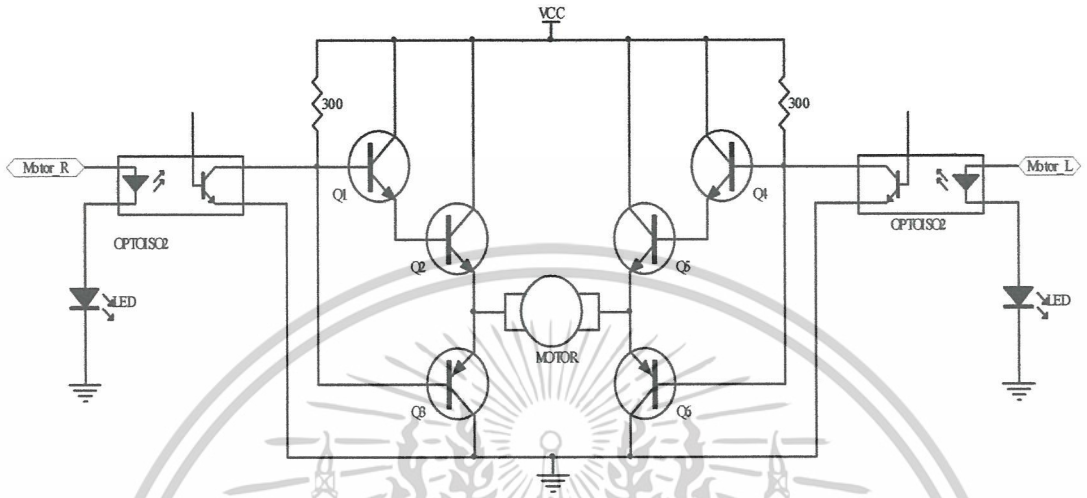
รูปที่ 3.16 แสดงลายวงจรของไมโครคอนโทรลเลอร์ ส่วนที่สอง

3.1.3 วงจรควบคุมมอเตอร์

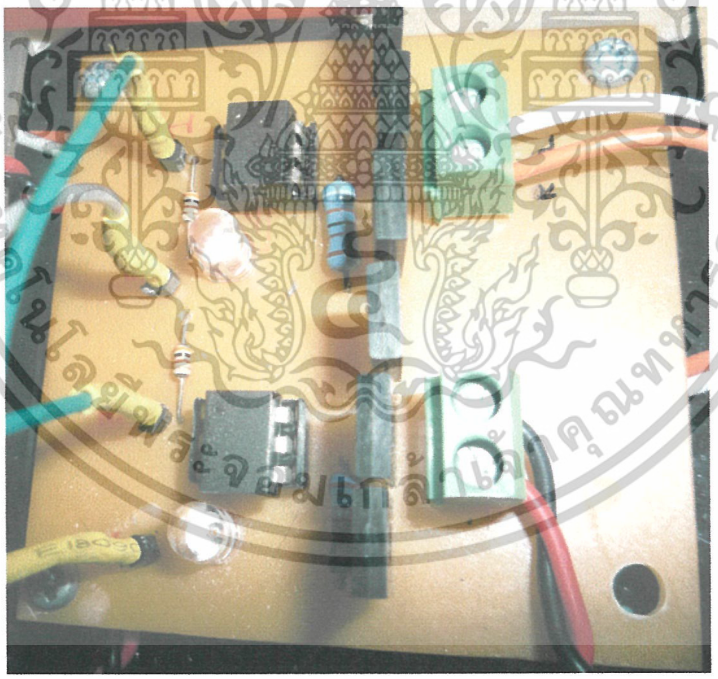
วงจรควบคุมมอเตอร์จะมี 2 วงจรคือ บริเวณทางเข้า ควบคุมการหมุนด้วยไมโครคอนโทรลเลอร์ตัวที่สองของส่วนที่สอง และ บริเวณทางออกจะควบคุมโดยการกดของผู้เก็บค่าบริการเองเมื่อมีการชำระค่าบริการเรียบร้อยแล้ว

3.1.3.1 วงจรควบคุมมอเตอร์ทางเข้า

วงจรควบคุมมอเตอร์บริเวณทางเข้าจะควบคุมด้วยการรับค่าอินพุตมาจากไมโครคอนโทรลเลอร์ ซึ่งออกมาจาก Moter_R และ Motor_L พอร์ท 2.0 และ พอร์ท 2.1 ตามลำดับ โดยเมื่อส่งค่ามาทาง Moter_R คานที่กลั้นจะเปิด ส่งค่ามาทาง Motor_L คานที่กลั้นจะปิดโดยวงจรจะเป็นดังรูป



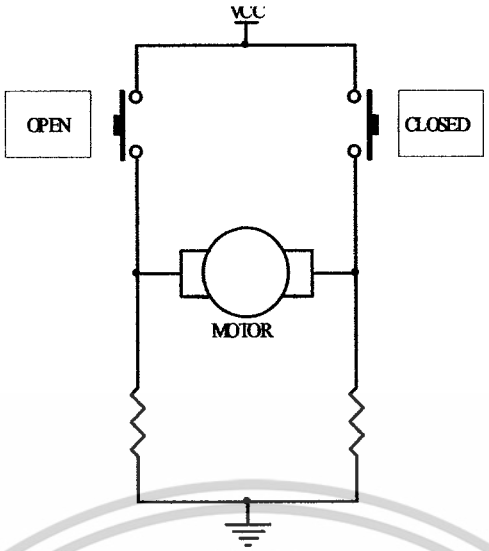
รูปที่ 3.17 แสดงวงจรควบคุมมอเตอร์ทางเข้า



รูปที่ 3.18 แสดงแผงวงจรควบคุมมอเตอร์ทางเข้า

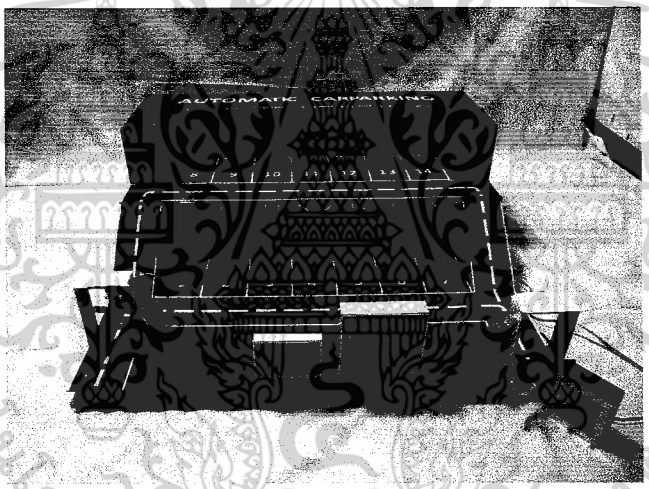
3.1.3.2 วงจรควบคุมมอเตอร์ทางออก

วงจรควบคุมมอเตอร์บริเวณทางออกจะเป็นวงจรควบคุมการหมุนของมอเตอร์ง่ายๆ เมื่อกดสวิตซ์ OPEN คานจะเปิด กดสวิตซ์ CLOSED คานจะปิด โดยอาศัยหลักการการป้อนแหล่งจ่ายไฟสลับขั้วให้แก่มอเตอร์ มอเตอร์จะหมุนกลับทาง

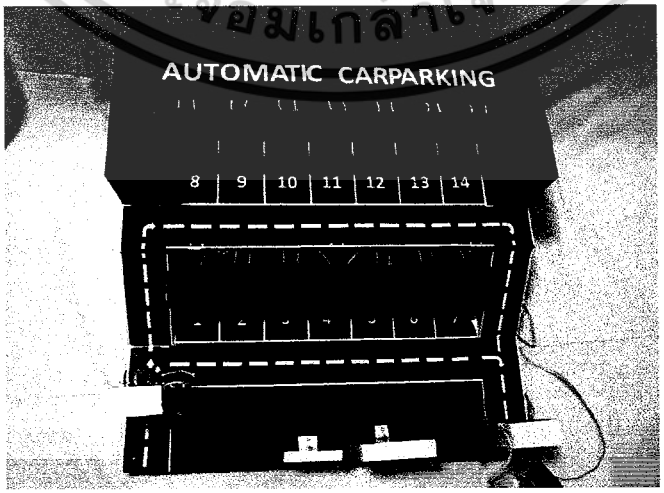


รูปที่ 3.19 แสดงวงจรควบคุมมอเตอร์ทางออก

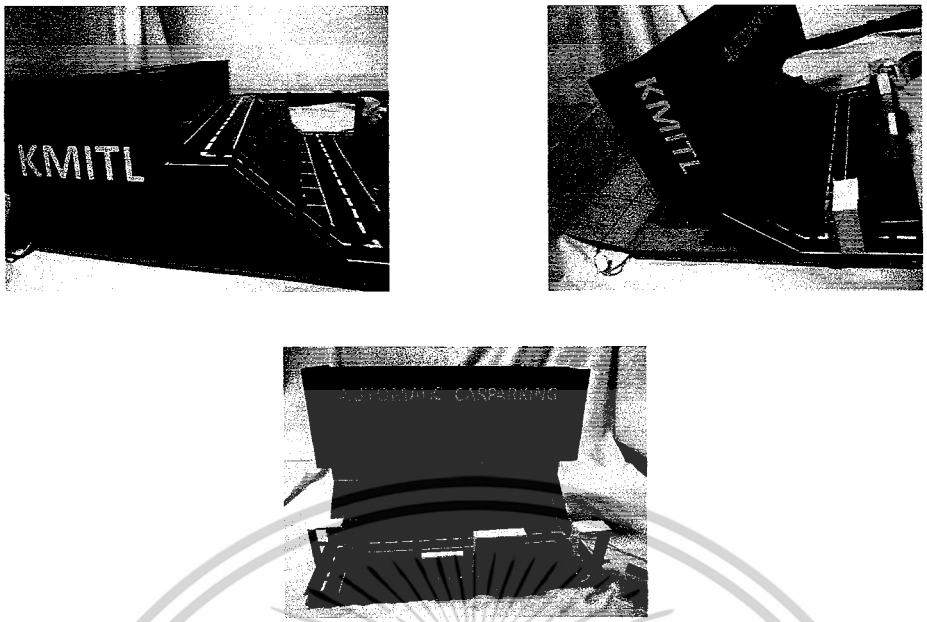
3.1.4 การออกแบบอาคารจอดรถ



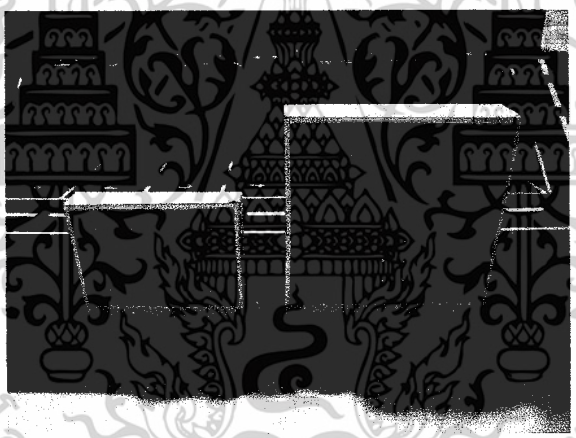
รูปที่ 3.20 แสดงอาคารจอดรถด้านหน้า



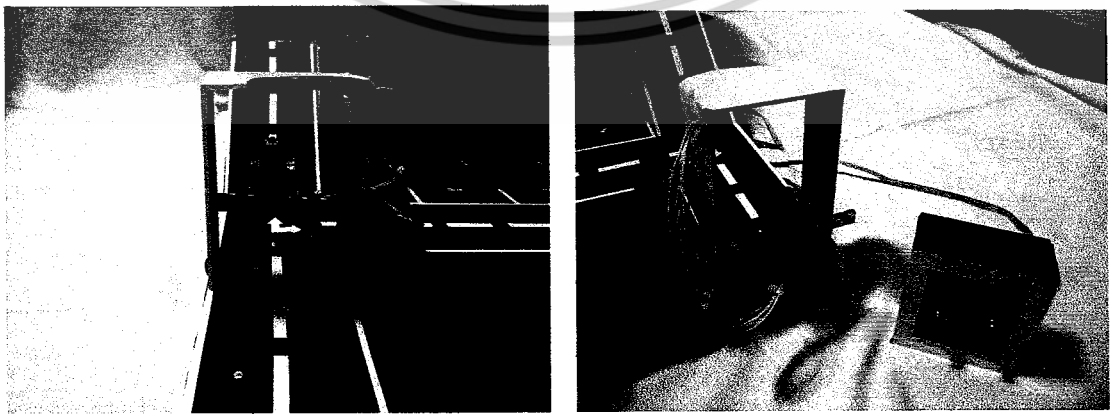
รูปที่ 3.21 แสดงอาคารจอดรถด้านบน



รูปที่ 3.22 แสดงอาคารจอดรถด้านต่างๆ



รูปที่ 3.23 แสดง LED และ 7-Segment สำหรับแสดงผลหน้าอาคารจอดรถ



รูปที่ 3.24 แสดงทางเข้าและทางออกของอาคารจอดรถ

3.2 โครงสร้างทางซอฟท์แวร์

ในการเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ MCS-51 จะใช้ภาษาซีในการเขียนเพื่อควบคุมข้อมูลที่รับเข้ามาและส่งข้อมูลต่างๆต่อไปให้กับโปรแกรม Delphi 7

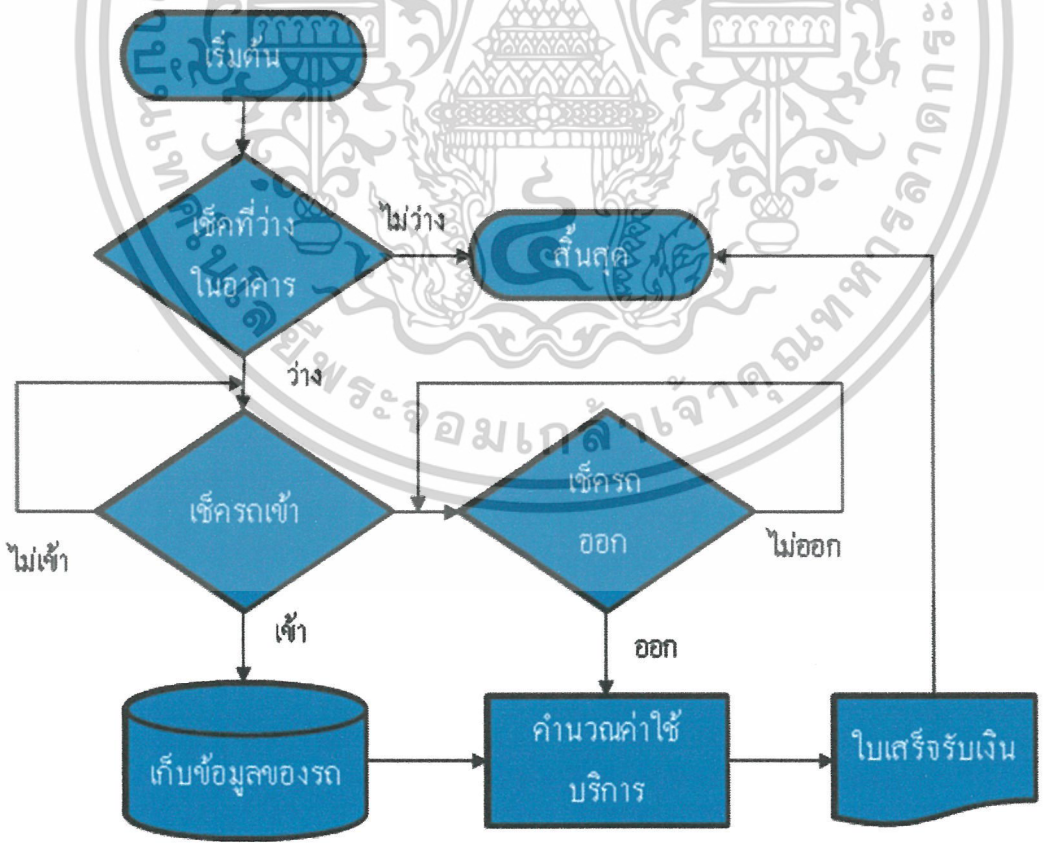
โครงสร้างของระบบในส่วนของโปรแกรมจะแยกได้ 3 ส่วนคือ

1. เมื่อโปรแกรม Delphi 7 ได้รับข้อมูลจากเซนเซอร์จะนำข้อมูลที่ได้มาแสดงตำแหน่งและจำนวนรถในอาคาร โดยจะแสดงผลเป็นภาพกราฟฟิกบนหน้าจอ เมื่อตำแหน่งใดมีรถมาจอดจะมีรูปรถขึ้นมา ตำแหน่งใดว่างจะไม่มีรูปรถ

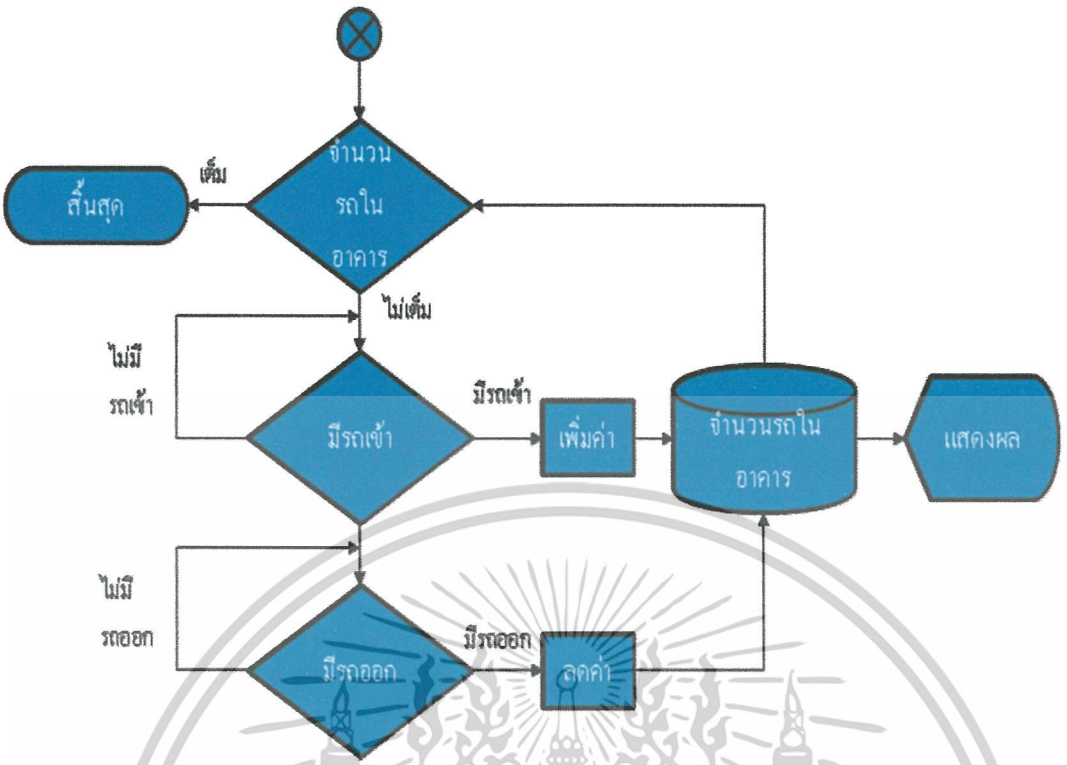
2. โปรแกรม Delphi 7 จำทำการถ่ายรูปของรถที่เข้ามาจอดทั้งทางเข้าและออก โดยเมื่อมีรถเข้ามา Delphi 7 จะถ่ายรูปแล้วเก็บไว้ แล้วเมื่อมีรถมาทางออก Delphi 7 จะถ่ายรูปไว้อีกครั้งเพื่อให้ผู้ดูแลระบบได้นำภาพดังกล่าวมาเปรียบเทียบกัน

3. โปรแกรม Delphi 7 จะทำการคิดค่าบริการโดยเริ่มคิดเวลาจากการเริ่มเก็บภาพและหยุดเวลาเมื่อนำภาพดังกล่าวมาเปรียบเทียบกัน โดย Delphi 7 จะทำการพิมพ์ใบเสร็จค่าบริการออกมาโดยบอกถึงเวลาเข้า-เวลาออกและค่าบริการ

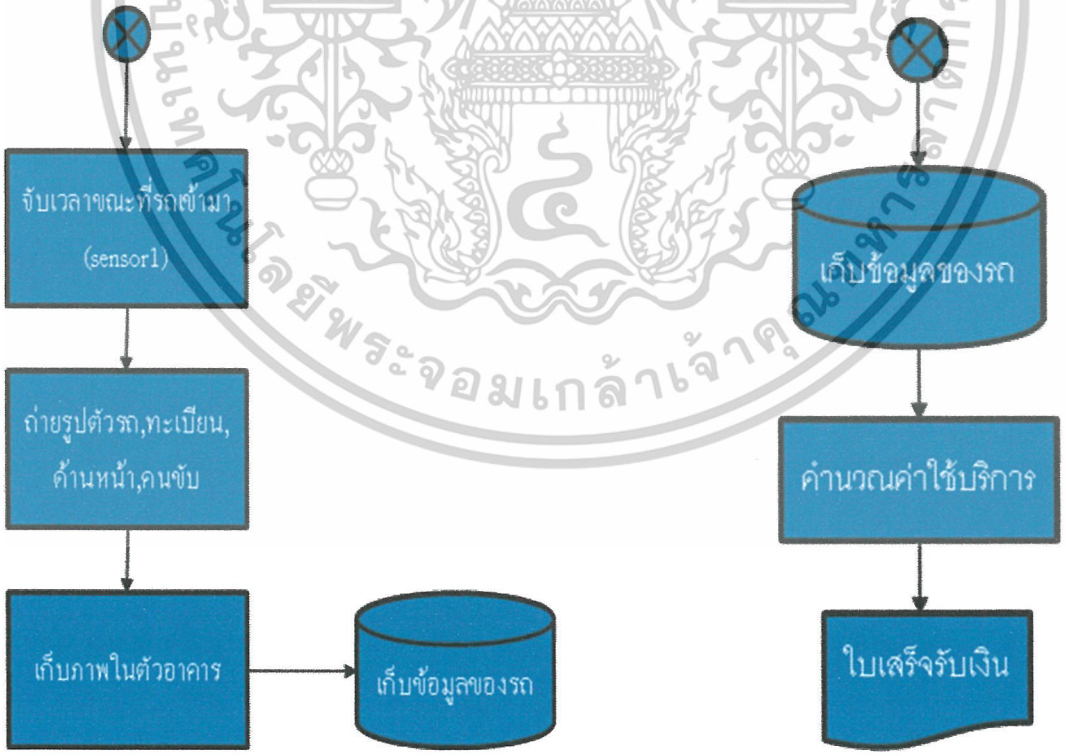
3.2.1 โฟว์ชาร์ทแสดงการทำงานของโปรแกรม



รูปที่ 3.25 แสดงโฟว์ชาร์ทของระบบควบคุมและรักษาความปลอดภัยการจอดรถอัตโนมัติ



รูปที่ 3.26 โฟว์ชาร์ทการทำงานการเช็คที่ว่างในอาคาร



รูปที่ 3.27 โฟว์ชาร์ทการทำงานเมื่อมีรถเข้า

รูปที่ 3.28 ผังการทำงานเมื่อมีรถออก

บทที่ 4

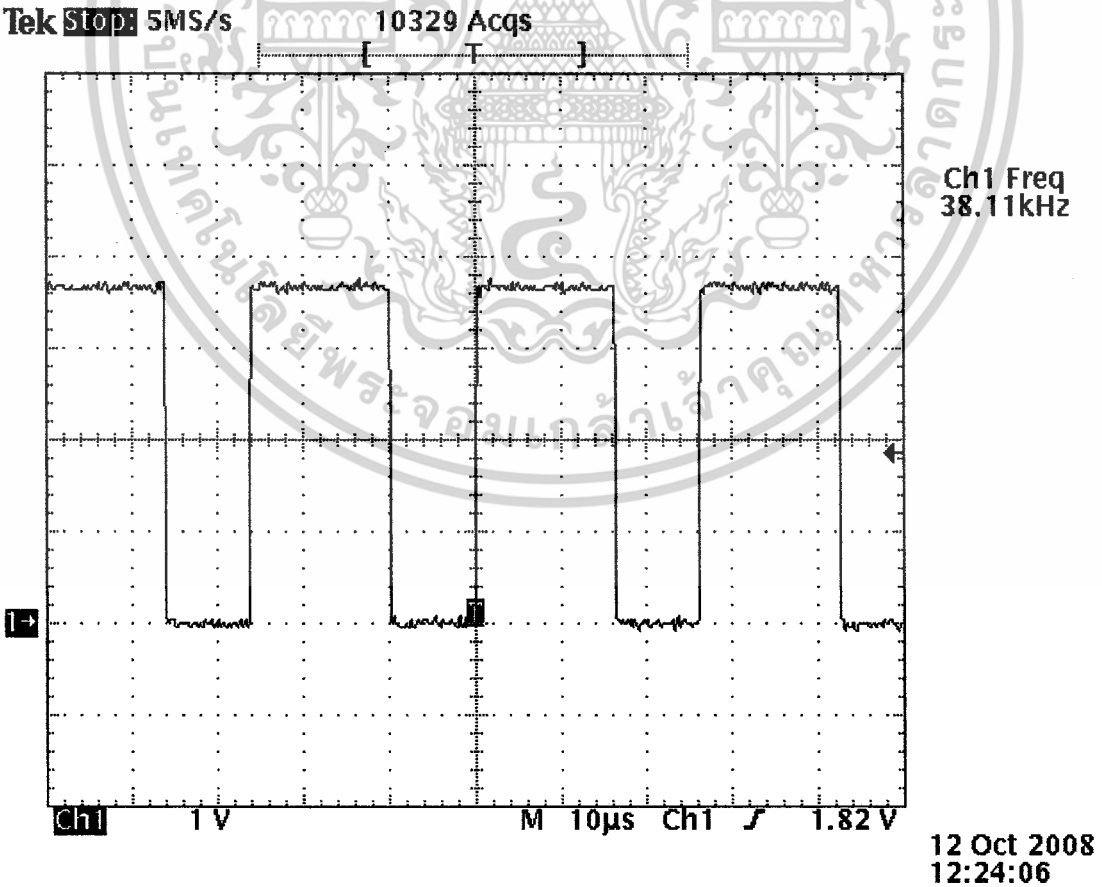
การทดลองและผลการทดลอง

ในโครงงานนี้แบ่งเป็น 2 ส่วนการทำงาน คือส่วนของ ฮาร์ดแวร์ และ ซอฟท์แวร์ โดยในช่วงแรกจะทดลองในส่วนของฮาร์ดแวร์ก่อน ซึ่งการทดลองจะเป็นการทดลองการทำงานของแผงวงจรการทำงาน 2 แผงวงจร ได้แก่ แผงวงจรที่ใช้ในการโปรแกรมให้กับ MCS-51 และ แผงวงจรที่ใช้ในการเชื่อมต่อ ชุดเซนเซอร์ แต่ละตัวทั้งด้านส่งและรับ เพื่อส่งข้อมูลไปยังส่วนการทำงานต่างๆ

4.1 การทดลองและผลการทดลอง

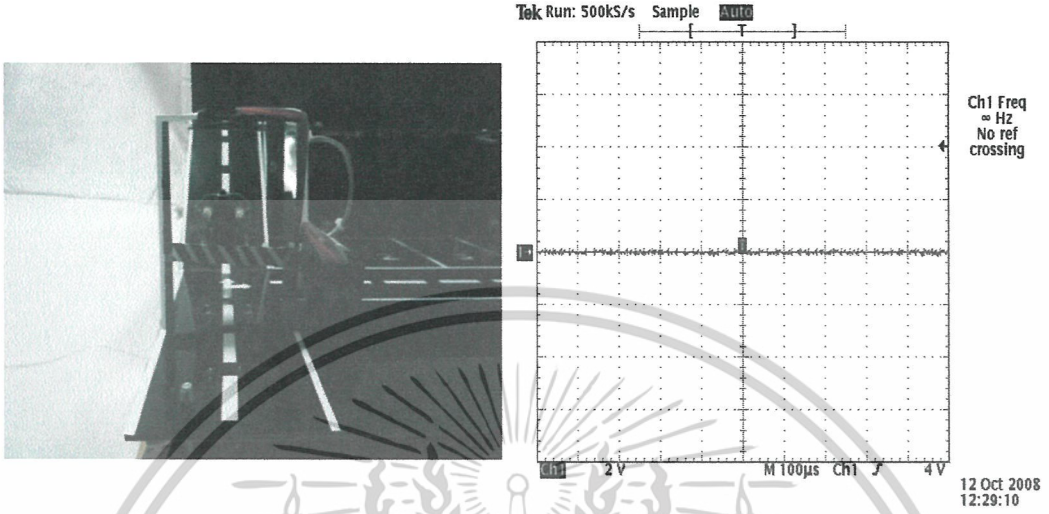
4.1.1 ผลการวัดความถี่ 38KHz จากวงจรเซนเซอร์ด้านส่งและค่าของแรงดันเอาท์พุตของเซนเซอร์ด้านรับ

โดยวัดสัญญาณของทางด้านส่งที่ขา 3 ของไอซี 555 และ ทางด้านรับที่ขา 1 ของโฟโต้ไดโอด เบอร์ 4838 ได้ 38KHz



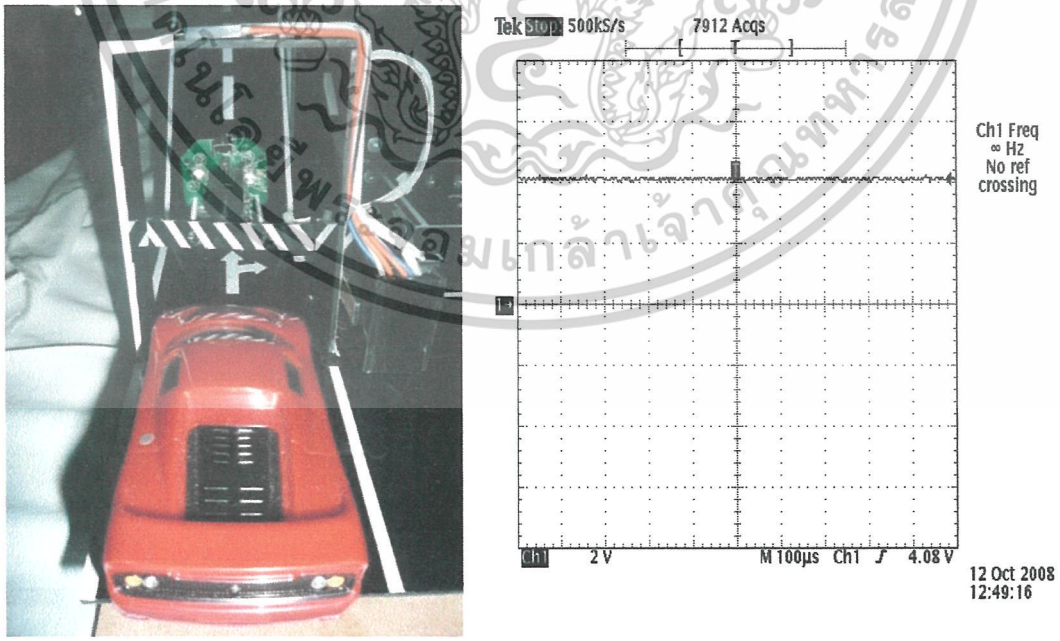
รูปที่ 4.1 แสดงรูปสัญญาณ 38KHz จากออสซิลโลสโคป

การทดลองการทำงานของเซนเซอร์ประตูทางเข้าของอาคารจอดรถเมื่อยังไม่มีรถจอดหรือยังไม่มีรถเข้ามากีดขวางลำคลื่นของเซนเซอร์ตัวส่ง ทำให้สัญญาณที่ได้เป็นแอกทีฟโลว์(Active Low)



รูปที่ 4.2 แสดงรูปสัญญาณเมื่อยังไม่มีสิ่งกีดขวางเข้ามากั้นระหว่างด้านรับและด้านส่ง

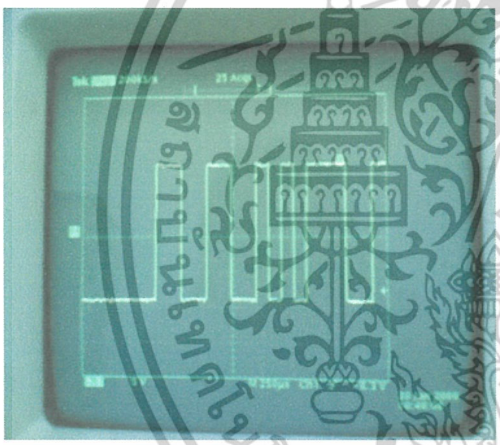
การทดลองการทำงานของเซนเซอร์ประตูทางเข้าของอาคารจอดรถเมื่อมีรถเข้ามาจอดหรือมีรถเข้ามากีดขวางลำคลื่นของเซนเซอร์ตัวส่ง ทำให้สัญญาณที่ได้เป็นแอกทีฟไฮ (Active High) มีค่าแรงดันที่วัดได้ 4.08 โวลท์



รูปที่ 4.3 แสดงรูปสัญญาณเมื่อมีสิ่งกีดขวางเข้ามากั้นระหว่างด้านรับและด้านส่ง

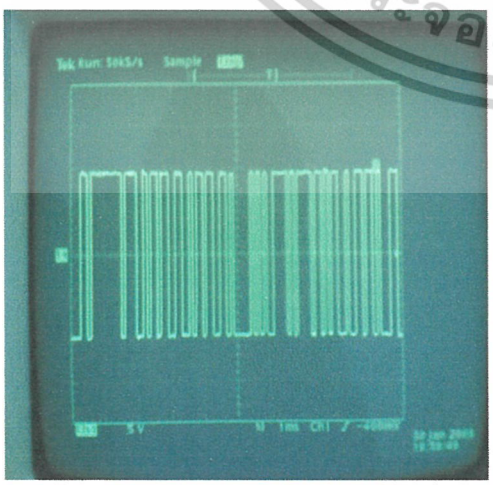
4.1.2 ผลการวัดสัญญาณอะซิงโครนัสจากไมโครคอนโทรเลอร์ไปสู่คอมพิวเตอร์โดยผ่านพอร์ทอนุกรม

จากการที่ไมโครคอนโทรลเลอร์ใช้โปรแกรมภาษาซีในการเขียนโปรแกรมสั่งงานเพื่อใช้ควบคุมการส่งข้อมูลออกไปยังคอมพิวเตอร์ โดยการส่งนั้นจะส่งข้อมูลออกไปเป็นเฟรม ข้อมูลในหนึ่งเฟรมนี้จะประกอบไปด้วยค่าของสถานะต่างๆ ในส่วนของประตูทางเข้า-ออก และ ตัวอาคารจอดรถครั้งเดียวเป็นสัญญาณอะซิงโครนัสที่ได้จึงออกมาเป็นเฟรมสัญญาณ แต่ละชุดสัญญาณจะบอกสถานะทุกค่าของเซนเซอร์ที่ใช้งานอยู่ถ้าเซนเซอร์ที่ถูกใช้งานอยู่นั้นจะทำให้มีพัลส์สัญญาณขึ้นมาเป็นสัญญาณรูปสี่เหลี่ยม สัญญาณจากการทดลองที่ได้จึงสังเกตได้อยากเพราะว่าแต่ละเฟรมสัญญาณนั้นมีความยาวมาก จึงพยายามวัดสัญญาณที่ช่วงต้นชุด หรือ ต้นขบวนของสัญญาณ โดยวัดสัญญาณให้เห็นในความต่างในแต่ละ Time Division ใน Time Division ที่มีค่าน้อยสุดคือ 250 uS นั้นจะสามารถสังเกตได้ดีที่สุดว่ามีการส่งสัญญานออกมาเป็นพัลส์

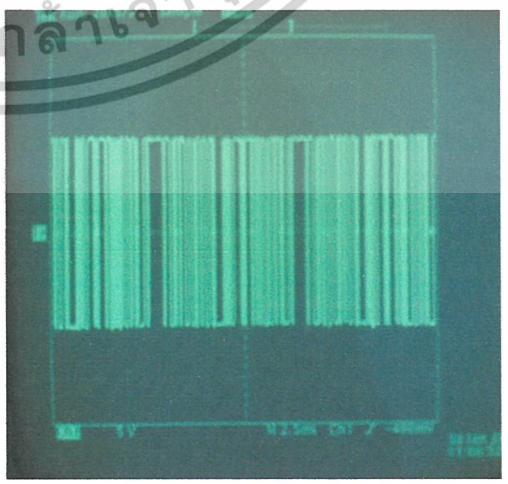


แสดงสัญญาณที่ 250 uS

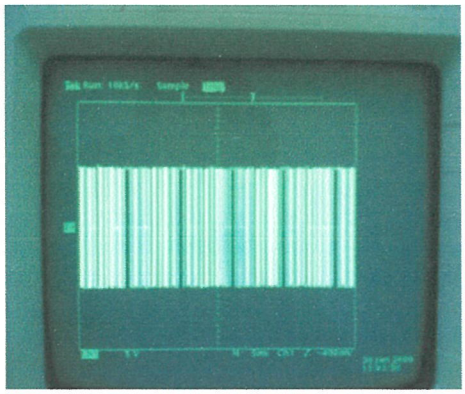
แสดงสัญญาณที่ 500 uS



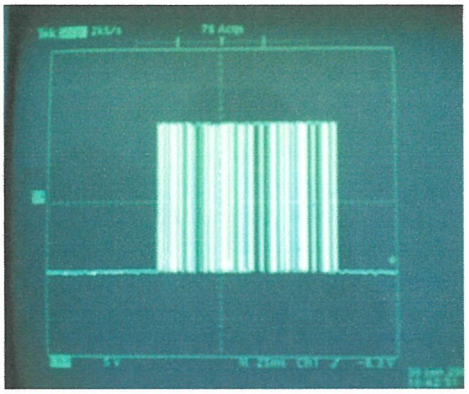
แสดงสัญญาณที่ 1 mS



แสดงสัญญาณที่ 2.5 mS

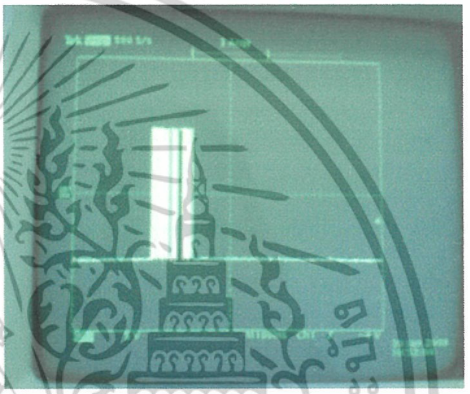


แสดงสัญญาณที่ 5 mS

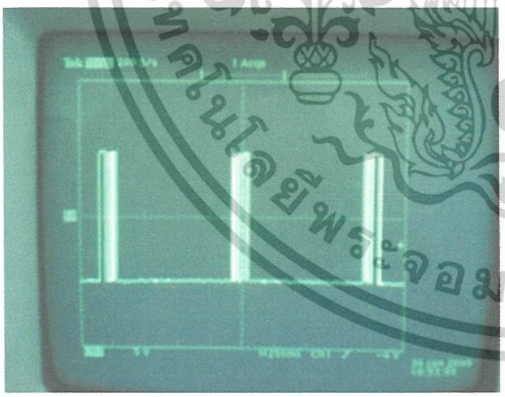


แสดงสัญญาณที่ 25 mS

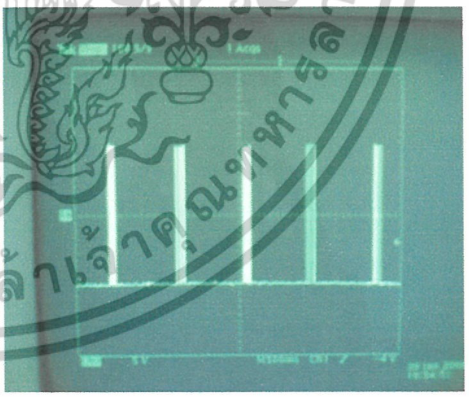
แสดงสัญญาณที่ 50 mS



แสดงสัญญาณที่ 100 mS



แสดงสัญญาณที่ 250 mS



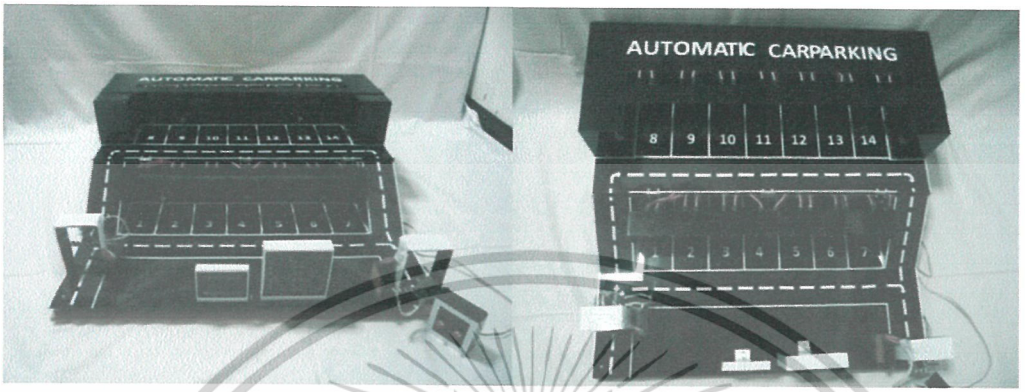
แสดงสัญญาณที่ 500 mS

รูปที่ 4.4 แสดงภาพสัญญาณอะซิงโครนัส

จากรูปสัญญาณทั้งหมดจะเห็นว่าสัญญาณที่ส่งออกทางพอร์ทอนุกรมจะส่งค่าที่รับเข้ามาจากเซนเซอร์แล้วส่งออกไปเป็นระยะๆ สังเกตจากรูปที่เวลา 500 mS จะเห็นชัดว่าสัญญาณที่ส่งออกมาจะมีลักษณะเป็นพัลส์ขึ้นมาส่วนที่ว่างไว้คือการเว้นช่วงการส่งเพื่อให้เวลาโปรแกรม Delphi ได้คำนวณ

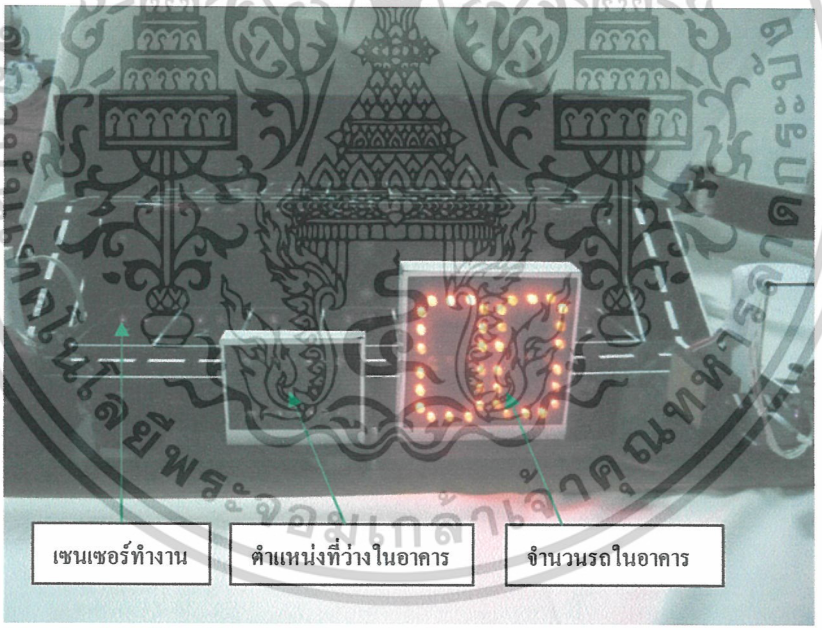
4.2 ผลการทดลองการทำงานของระบบเปิดปิดทางเข้าออกและระบบแสดงผล

4.2.1อาคารจอดรถอัตโนมัติตอนที่ยังไม่ได้เริ่มทำงาน



รูปที่ 4.5 แสดงอาคารจอดรถในขณะที่ยังไม่ได้เริ่มการทำงาน

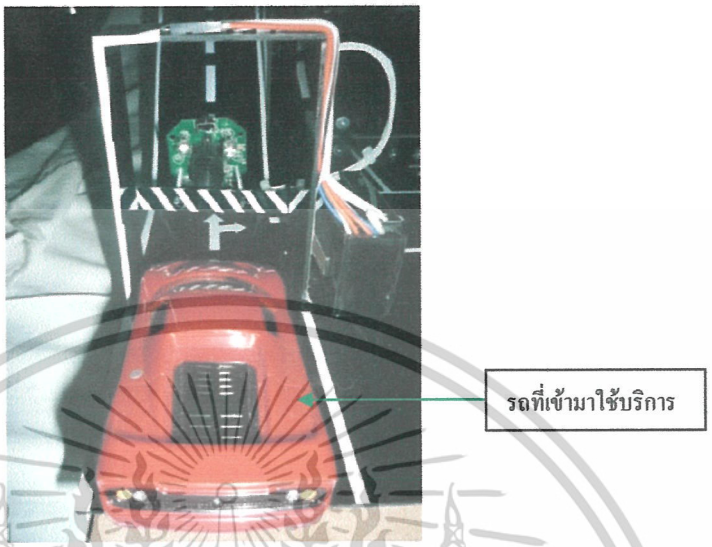
4.2.2 อาคารจอดรถอัตโนมัติตอนที่เริ่มทำงานพร้อมใช้งาน



รูปที่ 4.6 แสดงภาพอาคารจอดรถอัตโนมัติเมื่อทำการพร้อมใช้งาน

เมื่อเปิดสวิตซ์ให้กับอาคารจอดรถอัตโนมัติ จะเป็นการเริ่มทำงานของวงจรต่างๆ เริ่มจากเซนเซอร์ตัวรับส่งทำการเช็คสถานะในตัวอาคารว่ามีที่ว่าง และมีจำนวนรถที่จอดในอาคารจอดรถหรือไม่ เนื่องจากในตัวอาคารจอดรถยังไม่มีรถเข้ามาจอด จึงส่งค่าสภาวะที่ได้มาให้จอแสดงผลที่ว่างในอาคารและจำนวนรถที่จอดอยู่ในอาคารจอดรถ ข้างหน้าอาคารจอดรถอัตโนมัติ

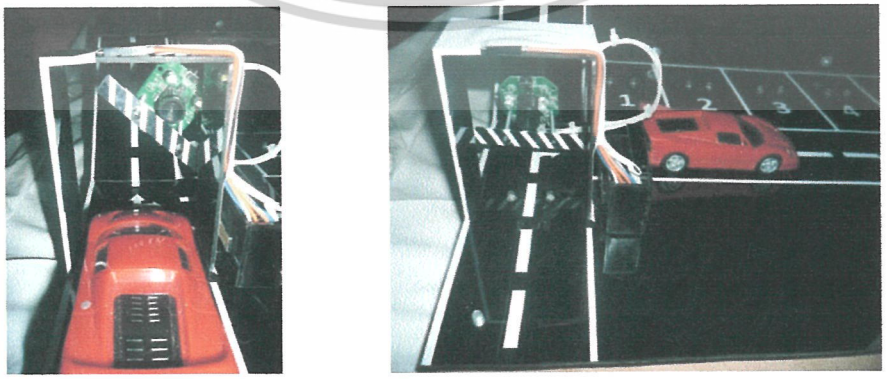
4.2.3 เมื่อมีรถเข้ามาจอดตรงประตูทางเข้าอาคาร ตัวรถที่เข้ามาทับเซนเซอร์ตัวส่งทำให้ไม่สามารถส่งไปยังตัวรับได้



รูปที่ 4.7 เมื่อมีรถเข้ามากีดขวางระหว่างเซนเซอร์ตัวส่งกับตัวรับ

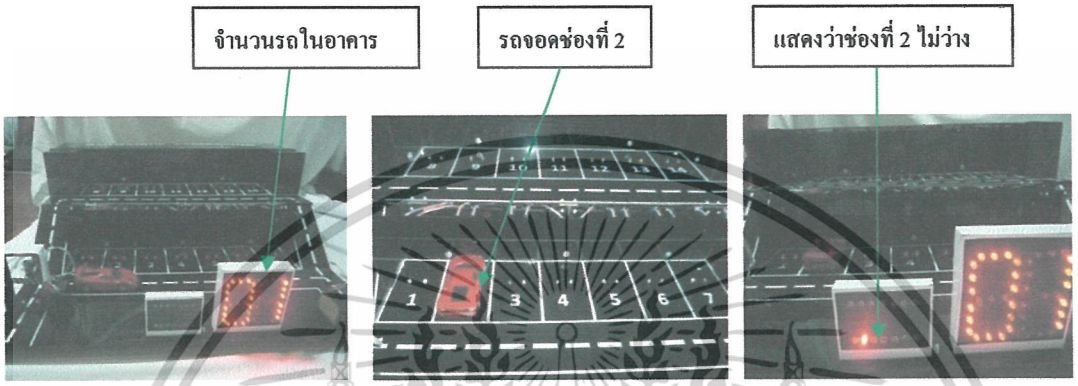
เมื่อมีรถเข้าใช้บริการอาคารจอดรถอัตโนมัติ รถที่เข้ามาจะเข้ามาทับตรงเซนเซอร์ตัวส่งทำให้ Microcontroller ตรวจจับได้ว่ามีรถเข้ามาตรงประตูทางเข้า Microcontroller จะทำการหน่วงเวลาไว้สักพักหนึ่งเพื่อให้กล้องได้ถ่ายรูปเก็บไว้ โดยรูปที่ถ่ายเก็บไว้นี้จะใช้ในส่วนของโปรแกรมคอมพิวเตอร์เป็นตัวดำเนินการ

4.2.4 รถที่เข้ามาจะจอดอยู่ตรงหน้าประตูทางเข้าขณะหนึ่งเพื่อให้กล้องได้ถ่ายภาพเก็บไว้ เมื่อกล้องถ่ายภาพได้ทำการถ่ายภาพเสร็จแล้ว คานก็จะเปิดออก ถ้ารถยังไม่เคลื่อนออกไปจากเซนเซอร์ตัวส่งคานก็จะยังคงค้างอยู่ยังไม่ปิด แต่ถ้ารถเคลื่อนที่ออกไปเซนเซอร์สามารถส่งไปยังตัวรับได้ก็จะทำการหน่วงเวลาอีกเพื่อปิดคานลงมาเหมือนเดิม



รูปที่ 4.8 แสดงภาพคานที่เปิดออกและคานที่ปิดเมื่อรถได้ผ่านออกไปแล้ว

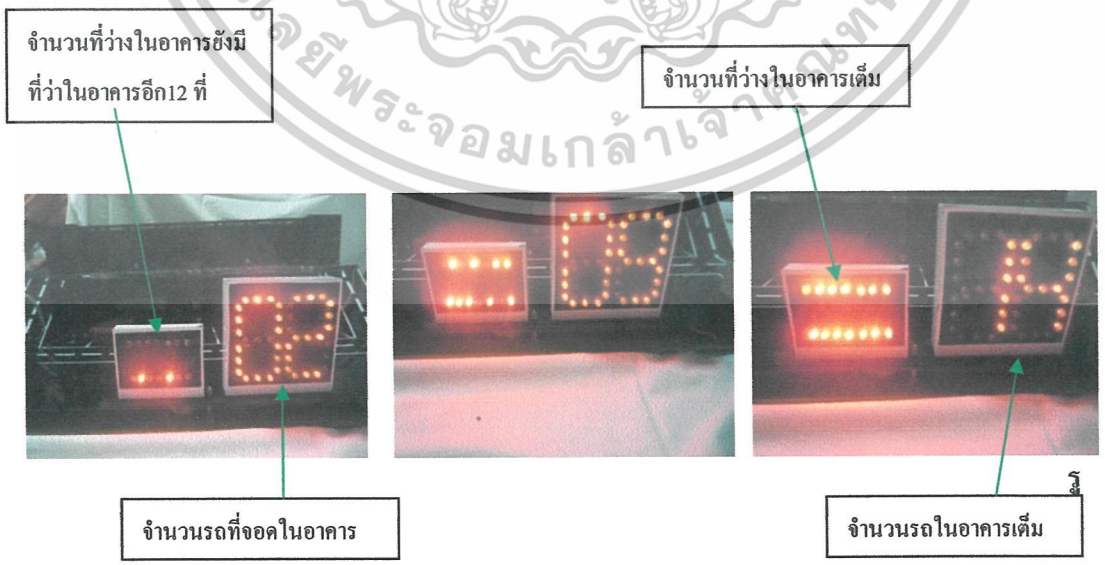
4.2.5 รถที่เคลื่อนที่ผ่านประตูทางเข้ามา Microcontroller จะทำการนับรถคันที่ผ่านไปนั้นโดยจะแสดงผลตรงจอแสดงผลที่ข้างหน้าอาคารจอดรถอัตโนมัติ เมื่อรถคันดังกล่าวหาที่จอดรถได้แล้ว และนำรถไปจอดตรงที่ว่างนั้น Microcontroller ก็จะนำค่าที่ได้จากการตรวจจับสถานะของรถที่เข้าไปจอดนั้นแล้วส่งไปแสดงผลที่จอแสดงผลของที่ว่างในอาคารว่ารถคันดังกล่าวนั้นไปจอดที่ไหน และในอาคารจอดรถอัตโนมัติได้เหลือที่ว่างในอาคารตรงไหนบ้าง



รูปที่ 4.9 แสดงรถที่เข้าไปจอดในอาคาร แสดงจำนวนรถและที่ว่างในอาคาร

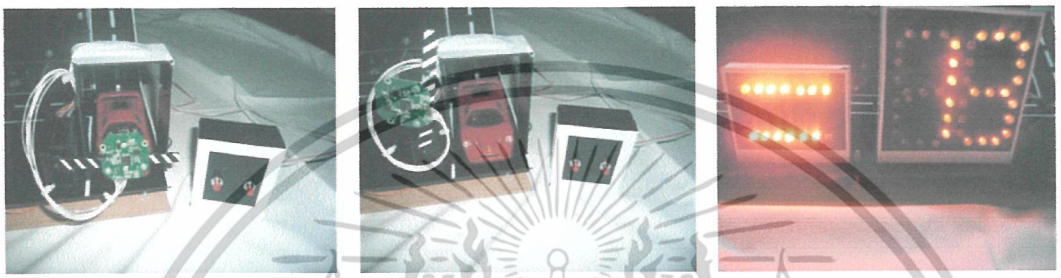
4.2.6 เมื่อมีรถเข้าไปใช้บริการในอาคารจอดรถอัตโนมัติมากขึ้นเรื่อยๆ Microcontroller ก็จะทำงานซ้ำๆกันจนมีรถที่เข้ามาใช้บริการในอาคารจอดรถจนเต็ม

ถ้าจำนวนรถในอาคารจอดรถอัตโนมัติเต็มเมื่อมีรถคันที่ 15 เข้ามาอีก มอเตอร์จะไม่เปิดประตูให้รถคันที่ 15 เข้าไปใช้บริการ



4.10 แสดงที่ว่างในอาคารจอดรถ และจำนวนรถในอาคารจอดรถ

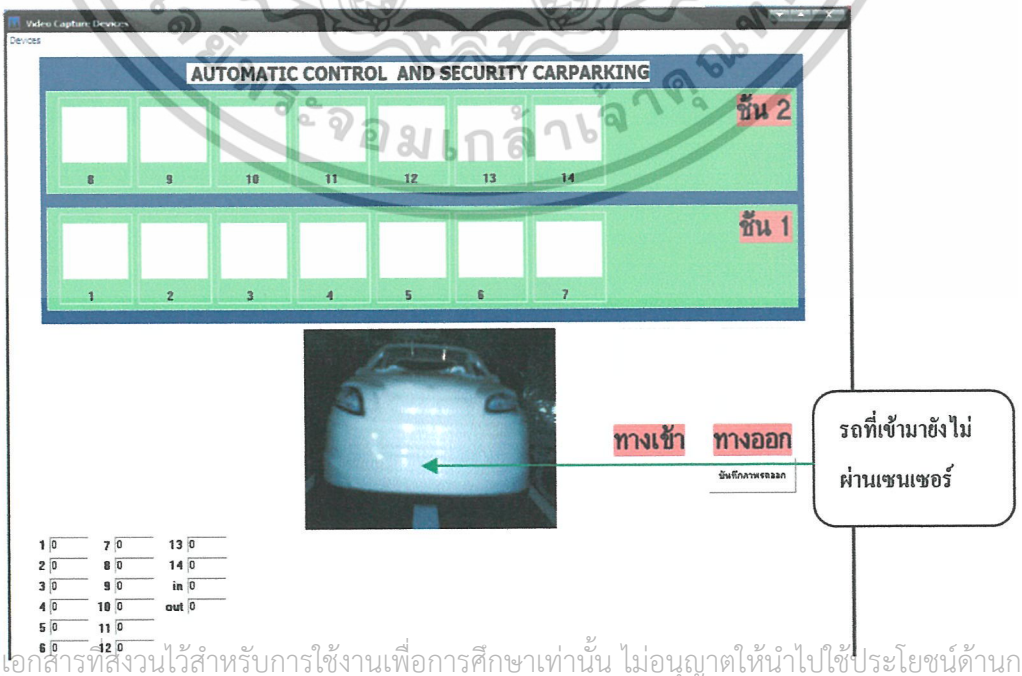
4.2.7 จากการที่มีรถเข้ามาใช้บริการอาคารจอดรถ เมื่อผู้รับบริการทำธุระเสร็จต้องการนำรถออกจากอาคารจอดรถอัตโนมัติ Microcontroller ทำงานโดยเมื่อรถเคลื่อนออกจากเซนเซอร์ที่จอดรถ Microcontroller จะแสดงว่าที่ว่างที่รถคันดังกล่าวออกไปนั้นบนจอแสดงผลเมื่อรถคันดังกล่าวเคลื่อนที่มาถึงประตูทางออกมากีดขวางเซนเซอร์ตัวส่งในส่วนของประตูทางออกใช้สวิตซ์ในการเปิดปิดคานเมื่อรถเคลื่อนที่ผ่านเซนเซอร์ตัวรับออกจากตัวอาคารจอดรถแล้ว Microcontroller ตรวจจับสถานะทำให้ลดจำนวนรถที่แสดงบนจอแสดงผลลง

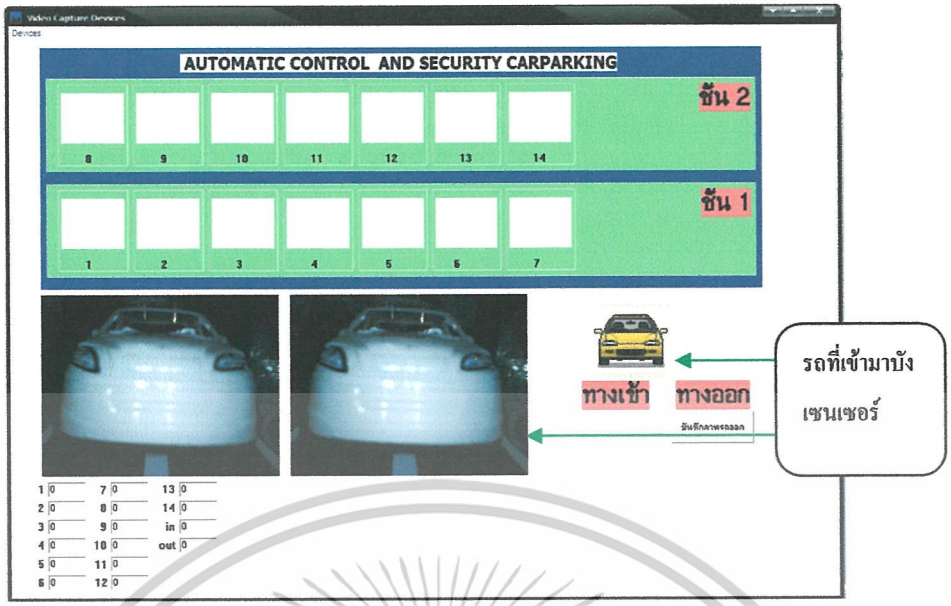


รูปที่ 4.11 แสดงภาพการใช้บริการของอาคารจอดรถ

4.3 ผลการทดลองการทำงานของโปรแกรมเดลไฟ

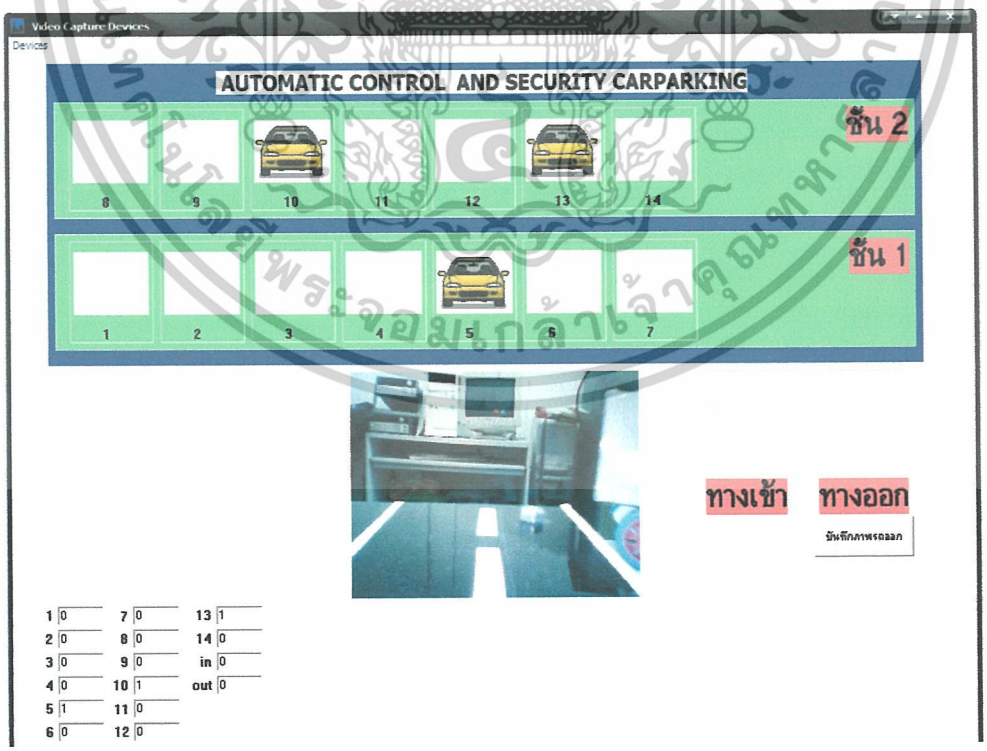
อาคารจอดรถอัตโนมัติมีการเริ่มต้นการทำงานโดย เมื่อมีรถเข้ามาใช้บริการอาคารจอดรถเมื่อมีรถเข้ามาจอดกีดขวางทางของเซนเซอร์ตัวส่งตรวจเช็คสถานะ เมื่อทราบสถานะแล้วว่ามีสิ่งของมาบังไม่ให้สามารถส่งไปยังตัวรับได้ ทำให้โปรแกรม Microcontroller ทำงานทำการเชื่อมต่อคอมพิวเตอร์ผ่านพอร์ทอนุกรม





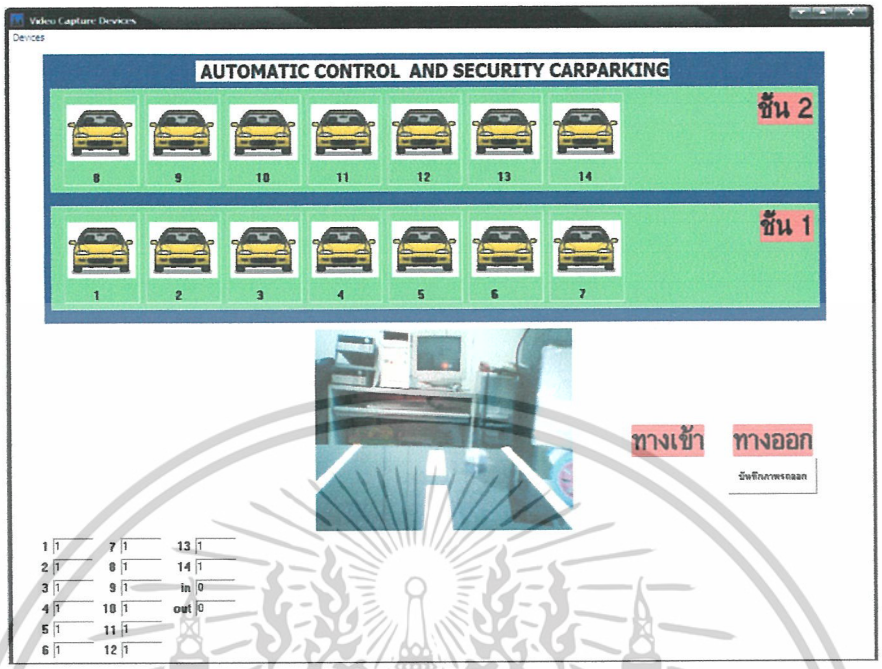
รูปที่ 4.12 แสดงเมื่อมีรถเข้ามาใช้บริการ

เมื่อรถผ่านเข้ามาตรงประตูทางเข้าจะทำการถ่ายรูปภาพเก็บไว้ในหน่วยความจำ เพื่อสามารถนำมาเปรียบเทียบกับรถตอนขาออก รถที่เข้ามาในอาคารได้แล้วจะทำการหาที่จอดรถ เมื่อหาที่จอดรถได้แล้วจึงนำรถเข้าไปจอดในช่องต่างๆ ในส่วนของโปรแกรมเคลไฟได้มีการแสดงผลบนหน้าจอคอมพิวเตอร์



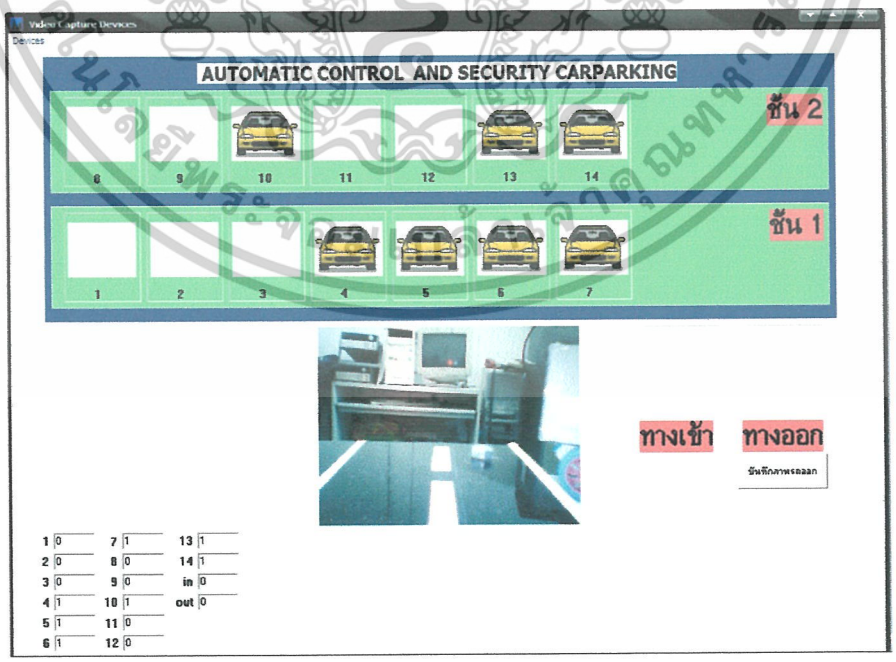
รูปที่ 4.13 แสดงตำแหน่งที่รถเข้าไปจอดอยู่

เมื่อมีรถเข้ามาจอดในอาคารจนเต็มในส่วนของโปรแกรมเดลไฟ จะแสดงผล

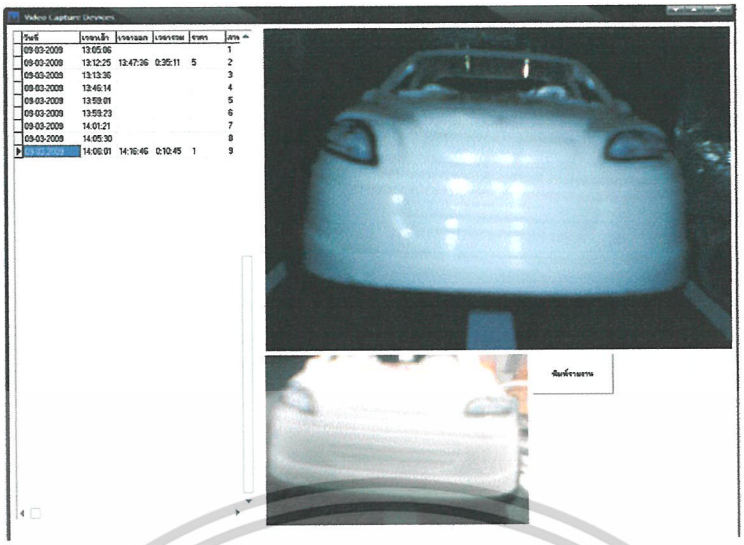


รูปที่ 4.14 แสดงการจอดรถในอาคารจนเต็ม

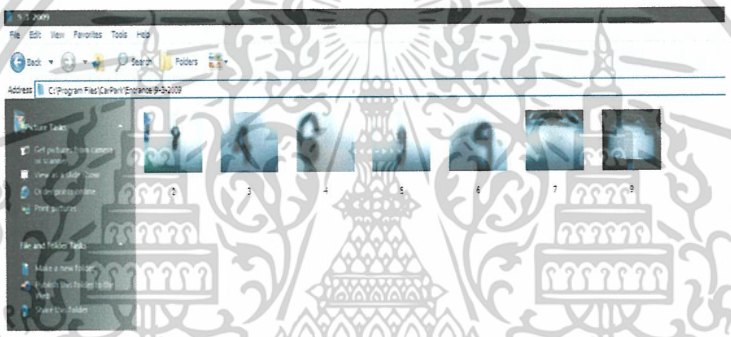
เมื่อมีรถออกจากอาคาร โปรแกรมเดลไฟจะทำการลดค่าตำแหน่งที่ว่างในตัวอาคารจอดรถลงเมื่อรถออกไปถึงประตูทางออก รถคันดังกล่าวไปกีดขวางเซนเซอร์ตัวส่ง Microcontroller จะทำการลดค่าในการแสดงผล



รูปที่ 4.15 แสดงรูปการจอดรถในอาคารจอดรถ

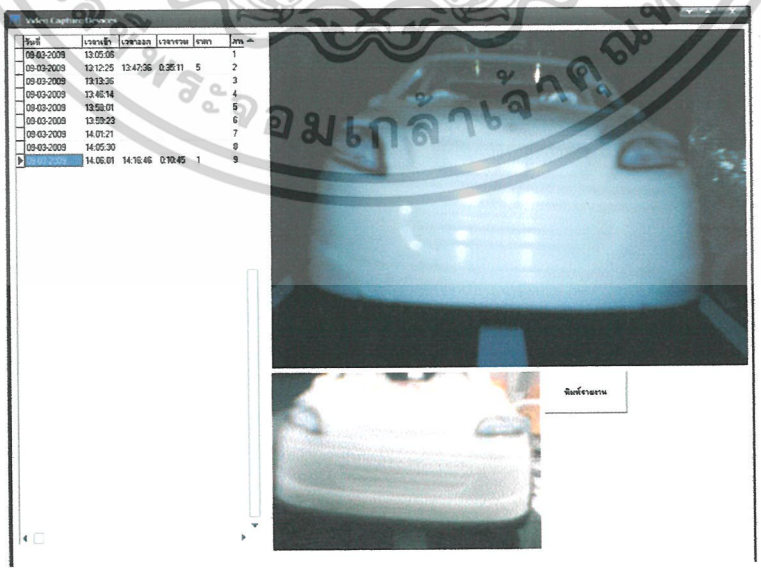


รูปที่ 4.16 แสดงภาพเมื่อรถเข้ามาจอดตรงประตูออก



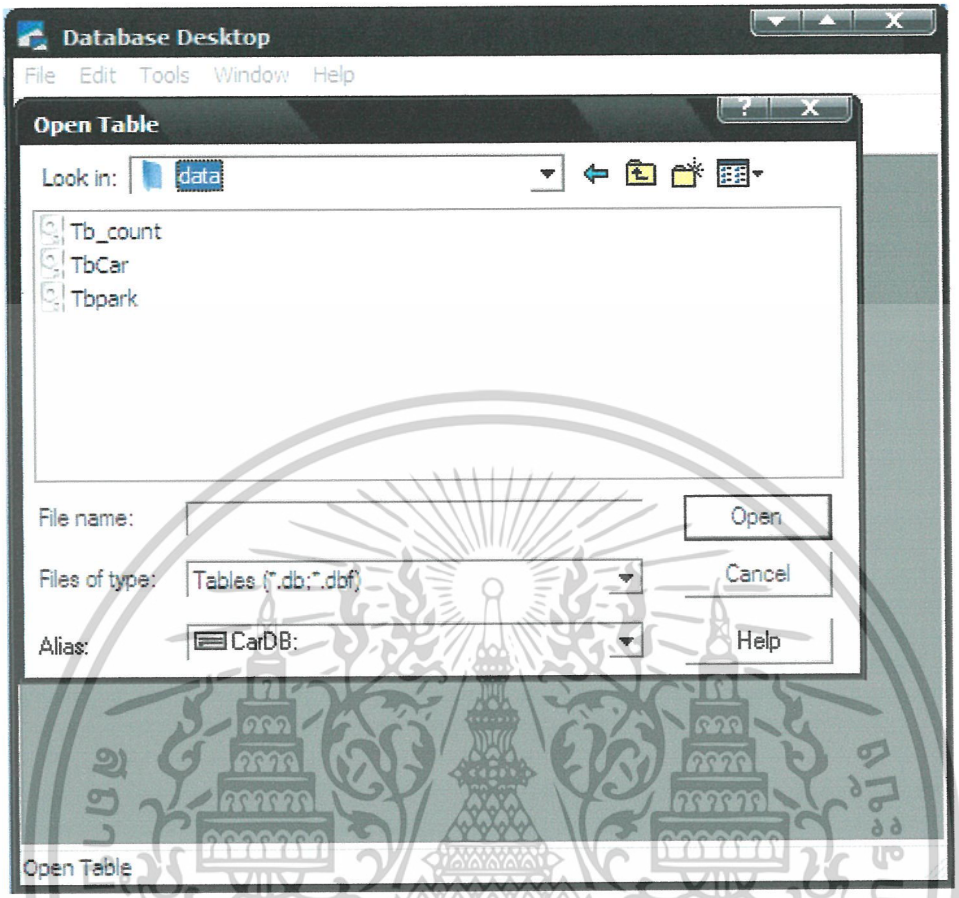
รูปที่ 4.17 แสดงภาพเมื่อรถเข้ามาจอดตรงประตูออกจะทำการถ่ายภาพเก็บไว้

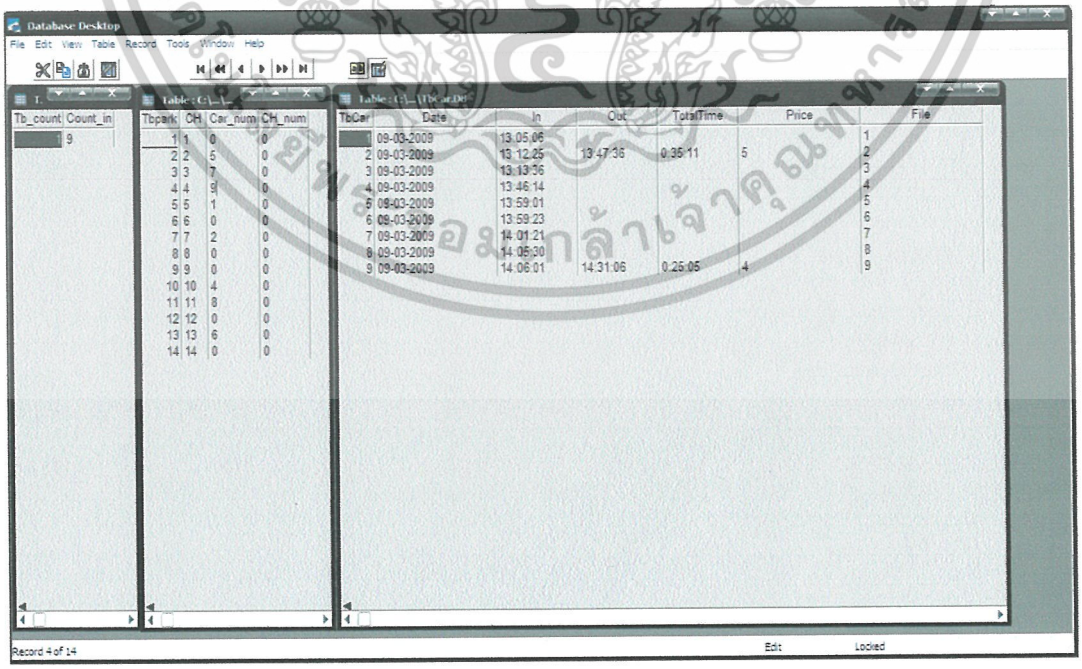
เมื่อรถเข้ามาถึงประตูทางออกจะทำให้โปรแกรมเคลไฟมีป๊อบอัพขึ้นมาทางประตูทางออก



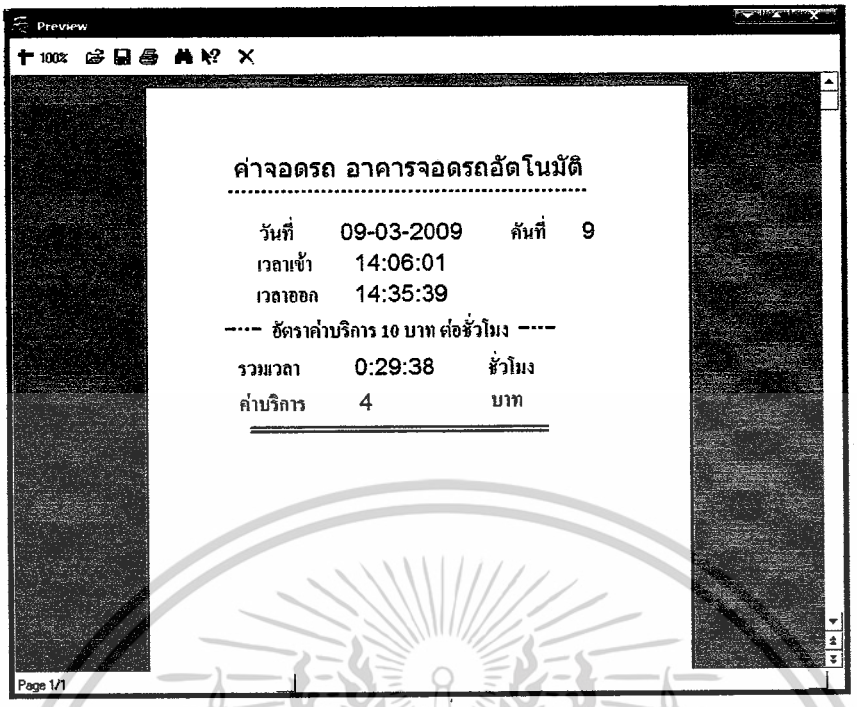
รูปที่ 4.18 แสดงป๊อบอัพที่แสดงขึ้นมา

เมื่อรถออกมาถึงเซนเซอร์และได้ถ่ายรูปเก็บไว้แล้วจะมีคาต้าเบสเก็บค่าไว้





รูปที่ 4.19 แสดงคาต้าเบสของการเข้าออก เวลา ค่าบริการต่างๆ



รูปที่ 4.20 แสดงใบเสร็จรับเงินของอัตราค่าจอดรถ

บทที่ 5
บทสรุปและวิจารณ์

ในการทำโครงงานปริญญานิพนธ์ แบบจำลองระบบบอกตำแหน่งพื้นที่ว่างในอาคารจอดรถอัตโนมัติ เพื่อศึกษาในด้านต่างๆที่เกี่ยวข้องกับโครงงาน เช่น การทำงานของอุปกรณ์ตรวจจับสัญญาณทางแสง การเขียนโปรแกรมด้วยDelphi กับการเชื่อมต่อกับอุปกรณ์ภายนอก และการทำงานของไมโครคอนโทรลเลอร์ เป็นต้น ซึ่งขอบเขตของโครงงานมีดังนี้ โดยในส่วนของฮาร์ดแวร์ ประกอบไปด้วย 2 ส่วนคือ ควบคุมการส่งข้อมูลให้แก่ Delphi และ ควบคุมการแสดงผลเบื้องต้นบริเวณหน้าอาคารจอดรถ ส่วนของซอร์ฟแวร์ที่เขียนจะมี 3 คือ แสดงผลบนหน้าจอ เก็บข้อมูลการจอดของแต่ละวัน และ คำนวณค่าบริการ

5.1 สรุปผลการทดลอง

1.จากการทดลองส่วนของฮาร์ดแวร์ทั้งหมดการทดสอบวงจรสามารถใช้งานได้แต่มีปัญหาบ้างเกี่ยวกับการความร้อนของภาคจ่ายไฟและลำคลื่นของเซนเซอร์ที่มากเกินไป

2.จากการทดลองส่วนของซอร์ฟแวร์ส่วนใหญ่ทำงานได้ผลแต่มีปัญหาตรงกล้องบริเวณทางออกเพราะ Delphi ไม่ค่อยเข้ากับการทำงานพร้อมกัน 2 กล้อง

5.2 ปัญหาและแนวทางการแก้ไข

ปัญหาและแนวทางแก้ไขที่เกิดขึ้นของการทำโครงงานปริญญานิพนธ์แบบจำลองระบบบอกตำแหน่งพื้นที่ว่างในอาคารจอดรถอัตโนมัติ นี้คือ

1. ปัญหา

การส่งแสงอินฟราเรดจากตัวส่งไปยังตัวรับ ระยะห่างของอุปกรณ์ทั้งสองทำให้ลำแสงที่ส่งไปมีลำคลื่นกว้างมากเกินไปจนไปโดนเซนเซอร์ตัวรับด้านข้าง

แนวทางการแก้ไข

หารถจำลองที่มีขนาดสูงพอที่จะบังลำคลื่นที่เลยมาจากด้านข้าง

2. ปัญหา

แหล่งจ่ายไปบริเวณภาครักษาแรงดันมีอาการร้อนมากเนื่องจากวงจรมีโหลดมากเกินไปจึงจ่ายกระแสไม่พอ เลยเกิดความร้อน

แนวทางการแก้ไข

เปลี่ยนไอซี ภาครักษาแรงดันให้จ่ายกระแสได้สูงขึ้น

3. ปัญหา

มอเตอร์ที่ใช้บริเวณทางเข้าออกมีแรงขับไม่พอที่จะยกคานปิดเปิดได้

แนวทางการแก้ไข

เปลี่ยนมอเตอร์ตัวใหม่ที่มีแรงมากขึ้นมาใส่แทน

4. ปัญหา

การทดลองการเขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ ให้ส่งเข้าพอร์ทอนุกรมเนื่องจากภาษาแอสเซมบีย่งยากเกินและซับซ้อนเกินไปเสี่ยงต่อการใช้เวลาในการเขียน

แนวทางแก้ไข

ทำการเรียนรู้ภาษาซี เพื่อนำมาเขียนแทน

5. ปัญหา

เนื่องจากการส่งตามมาตรฐานแบบ RS-232 มีระยะที่น้อยทำให้มีการลดทอนของศัญญาณ

แนวทางแก้ไข

เปลี่ยนมาตรฐานการส่งเป็นแบบ RS-485 เพื่อได้ระยะทางที่ไกลขึ้นและลดการลดทอนของสัญญาณ

บรรณานุกรม

1. ประพนธ์ อัศวภาณุวัฒน์. Delphi Episode II เทคนิคและการพัฒนาโปรแกรมด้วยเดลไฟ.
กรุงเทพมหานคร : ซีเอ็ดยูเคชั่น, 2543

2. สัจจะ จรัสรุ่งรวีวร, จักรพงษสุขประเสริฐ. เริ่มต้นอย่างมืออาชีพด้วย Delphi 7 ฉบับสมบูรณ์. นนทบุรี :
อินโฟเพรส, 2546

3. สันติ นุราช,อุกฤษฏ์ ตันทสุทธานนท์. เรียนรู้ไมโครคอนโทรลเลอร์ MCS-51 ฉบับภาษา C
Micro Research Technology Ltd.,Part

4. อุดม รานอก.ภาษา C สำหรับงานควบคุมไมโครคอนโทรลเลอร์ MCS-51
นนทบุรี: ไอดีซีฯ,2548

5. รศ.สมยศ จุณณปิยะ.การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS-51
ภาควิชาวิศวกรรมโทรคมนาคม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พิมพ์ครั้งที่ 6 พ.ศ.2550

ภาคผนวก ก.
โปรแกรมภาษาซี

```
#include <reg52.h>
#include <intrins.h>
#include <stdio.h>

sbit f1_CH1 = P1^0;
sbit f1_CH2 = P1^1;
sbit f1_CH3 = P1^2;
sbit f1_CH4 = P1^3;
sbit f1_CH5 = P1^4;
sbit f1_CH6 = P1^5;
sbit f1_CH7 = P1^6;

sbit f2_CH1 = P3^0;
sbit f2_CH2 = P3^1;
sbit f2_CH3 = P3^2;
sbit f2_CH4 = P3^3;
sbit f2_CH5 = P3^4;
sbit f2_CH6 = P3^5;
sbit f2_CH7 = P3^6;

unsigned char C1[7];
unsigned char C2[7];

void delay(int i)
{
        unsigned int j;
        do
        {
                for(j = 0;j < 100;j++){}
                i--;
```

```
        }while(i > 0);
}
void initial()
{
        P0 = 0xff;
        P1 = 0xff;
        P2 = 0xff;
        P3 = 0xff;

   SCON  = 0x40;
   TMOD |= 0x21
   TH1   = 0xfd
   TR1   = 1;
   TI    = 1;

   TH0 = 0xb7;
        TL0 = 0xaa;
        IE = 0x83;
        ES  = 0;
        IT0 = 1;
        IE0 = 0;
        EX0 = 1;
        EA  = 1;
}
void read_1st()
{
        if(f1_CH1 == 1)
         C1[1] = '1';
        else
         C1[1] = '0';
```

```
if(f1_CH2 == 1)
 C1[2] = '1';
else
 C1[2] = '0';

if(f1_CH3 == 1)
 C1[3] = '1';
else
 C1[3] = '0';

if(f1_CH4 == 1)
 C1[4] = '1';
else
 C1[4] = '0';

if(f1_CH5 == 1)
 C1[5] = '1';
else
 C1[5] = '0';

if(f1_CH6 == 1)
 C1[6] = '1';
else
 C1[6] = '0';

if(f1_CH7 == 1)
 C1[7] = '1';
else
 C1[7] = '0';

}
```

```
void read_2st()
{
        if(f2_CH1 == 1)
         C1[1] = '1';
        else
         C1[1] = '0';

        if(f2_CH2 == 1)
         C1[2] = '1';
        else
         C1[2] = '0';

        if(f2_CH3 == 1)
         C1[3] = '1';
        else
         C1[3] = '0';

        if(f2_CH4 == 1)
         C1[4] = '1';
        else
         C1[4] = '0';

        if(f2_CH5 == 1)
         C1[5] = '1';
        else
         C1[5] = '0';

        if(f2_CH6 == 1)
         C1[6] = '1';
        else
```

```
        C1[6] = '0';

        if(f2_CH7 == 1)
         C1[7] = '1';
        else
         C1[7] = '0';
}
void main()
{  unsigned char i;
   delay(100);
   initial();
        while(1)
   {
                read_1st();
                read_2st();

        printf("f1_CH1:%c,f1_CH2:%c,f1_CH3:%c,f1_CH4:%c,f1_CH5:%c,f1_CH6:%c,f1_CH7:%c",C1[1],C1[2]
,C1[3],C1[4],C1[5],C1[6],C1[7]);

        printf("f2_CH1:%c,f2_CH2:%c,f2_CH3:%c,f2_CH4:%c,f2_CH5:%c,f2_CH6:%c,f2_CH7:%c",C2[1],C2[2]
,C2[3],C2[4],C2[5],C2[6],C2[7]);
      delay(1000);
        }
}
```

ภาษาซีที่เขียนลง MCS-51 ส่วนที่สองตัวที่ 1 ควบคุมการแสดงผลให้ LED และส่งค่าให้แก่ MCS-51 ส่วนที่สองตัวที่ 2

```
#include <REG52.h>

sbit LED0   = P0^0;
sbit LED1   = P0^1;
sbit LED2   = P0^2;
sbit LED3   = P0^3;
sbit LED4   = P0^4;
sbit LED5   = P0^5;
sbit LED6   = P0^6;
sbit LED7   = P0^7;

sbit LED8   = P2^0;
sbit LED9   = P2^1;
sbit LED10  = P2^2;
sbit LED11  = P2^3;
sbit LED12  = P2^4;
sbit LED13  = P2^5;
sbit LED14  = P2^6;
sbit LED15  = P2^7;

sbit SW_0   = P1^0;
sbit SW_1   = P1^1;
sbit SW_2   = P1^2;
sbit SW_3   = P1^3;
sbit SW_4   = P1^4;
sbit SW_5   = P1^5;
sbit SW_6   = P1^6;
sbit SW_7   = P1^7;
```

```
sbit SW_8    = P3^0;
sbit SW_9    = P3^1;
sbit SW_10   = P3^2;
sbit SW_11   = P3^3;
sbit SW_12   = P3^4;
sbit SW_13   = P3^5;
sbit SW_14   = P3^6;
sbit SW_15   = P3^7;

unsigned char m = 0;

void delay(unsigned int tick)
{
        unsigned char i,j;              // For keep counter loop
        for(i=0;i<tick;i++)             // Loop delay
        for(j=0;j<200;j++);
}

void main(void)
{
  P0 = 0x00;
        P2 = 0x00;
        P1 = 0xFF;
        P3 = 0xFF;
        LED7=1;
        LED15=1;

        while(1)
        {
         if(SW_0==0) LED0=1;
         else     LED0=0;
```

```
if(SW_1==0) LED1=1;
else    LED1=0;

if(SW_2==0) LED2=1;
else    LED2=0;

if(SW_3==0) LED3=1;
else    LED3=0;

if(SW_4==0) LED4=1;
else    LED4=0;

if(SW_5==0) LED5=1;
else    LED5=0;

if(SW_6==0) LED6=1;
else    LED6=0;

if(SW_7==0) LED8=1;
else    LED8=0;

if(SW_8==0) LED9=1;
else    LED9=0;

if(SW_9==0) LED10=1;
else    LED10=0;

if(SW_10==0) LED11=1;
else    LED11=0;
```

```
        if(SW_11==0) LED12=1;
        else     LED12=0;

        if(SW_12==0) LED13=1;
        else     LED13=0;

        if(SW_13==0) LED14=1;
        else     LED14=0;


        while (SW_14==0)
        {
          LED7=0;
    delay(100);
                }

        while (SW_15==0)
        {
          LED15=0;
                delay(100);

                }

to_out1:LED7=1;
                LED15=1;
                delay(100);
                }
  }
```

ภาษาซีที่เขียนลง MCS-51 ส่วนที่สองตัวที่ 2 ควบคุมการทำงานให้แก่มอเตอร์บริเวณทางเข้า

```
#include <REG52.h>

sbit digit1  = P2^6;
sbit digit2  = P2^7;
sbit sw_up   = P3^2;
sbit sw_down = P3^3;
sbit MOTOR_R = P2^0;
sbit MOTOR_L = P2^1;


code unsigned char display[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char value = 0;
unsigned char seg1 = 0;
unsigned char seg2 = 0;

void delay(int tick)
{
        int i,j;
        for(i=0;i<tick;i++)
        for(j=0;j<250;j++);
}

void multiplex()
{
        seg1 = value%10;
        seg2 = value/10;
        digit1 = 0;
        digit2 = 1;
        P0 = display[seg2];
```

```
        delay(5);

        digit1 = 1;
        digit2 = 0;
        P0 = display[seg1];
        delay(5);
}

void scan_sw()
{
         if(sw_up==0)
        {
         if(value<=13)
         {
          delay(2000);
                MOTOR_R=1;
                delay(200);
                MOTOR_R=0;

dex1:   if(sw_up==0)goto        dex1;
                delay(700);
                value++;
                MOTOR_L=1;
                delay(200);
                MOTOR_L=0;
                multiplex();

                }
        }
```

```
        MOTOR_L=0;
         if(sw_down==0)
         {
                multiplex();

dex:    if(sw_down==0)
    {
                multiplex();
    goto        dex;
                }
                if(value!=0)
                {
                delay(120);
                value--;
                multiplex();
                }
                }
}
void main(void)
{
  MOTOR_R = 0;
        MOTOR_L = 0;

        P0 = 0x00;
        digit2 = 1;
        digit1 = 1;
        while(1)
        {
                scan_sw();
                multiplex();
        }
```

ภาคผนวก ข.

โปรแกรม DELPHI 7

```
unit uMain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DSUtil, StdCtrls, DSPack, DirectShow9, ExtCtrls, jpeg, Menus,
  VaClasses, VaComm, Grids, DBGrids, FR_Class, DB, DBTables, FR_DSet,
  FR_DBSet,BDE;

type
  Tfrmmain = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Bevel1: TBevel;
    Image1: TImage;
    Bevel2: TBevel;
    Image2: TImage;
    Bevel3: TBevel;
    Image3: TImage;
    Bevel4: TBevel;
    Image4: TImage;
    Bevel5: TBevel;
    Image5: TImage;
    Bevel6: TBevel;
    Image6: TImage;
    Bevel7: TBevel;
    Image7: TImage;
```

Panel3: TPanel;

Bevel8: TBevel;

Image8: TImage;

Bevel9: TBevel;

Image9: TImage;

Bevel10: TBevel;

Image10: TImage;

Bevel11: TBevel;

Image11: TImage;

Bevel12: TBevel;

Image12: TImage;

Bevel13: TBevel;

Image13: TImage;

Bevel14: TBevel;

Image14: TImage;

Timer1: TTimer;

Label14: TLabel;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

Label11: TLabel;

Label12: TLabel;

Label13: TLabel;

Bevel16: TBevel;

Label56: TLabel;

MainMenu1: TMainMenu;

Devices: TMenuItem;

SampleGrabber: TSampleGrabber;

Filter: TFilter;

FilterGraph: TFilterGraph;

VideoWindow: TVideoWindow;

SnapShot: TButton;

VaComm1: TVaComm;

Image16: TImage;

Label16: TLabel;

Label22: TLabel;

Label21: TLabel;

Timer2: TTimer;

Report: TfrReport;

Button1: TButton;

Label17: TLabel;

Button2: TButton;

frDBDataSet1: TfrDBDataSet;

Image: TImage;

Label19: TLabel;

Label20: TLabel;

Table1: TTable;

Timer3: TTimer;

DataSource1: TDataSource;

Button3: TButton;

Label23: TLabel;

Label24: TLabel;

Label25: TLabel;

Label26: TLabel;

Label27: TLabel;

```
    Label28: TLabel;
    Label29: TLabel;
    Label30: TLabel;
    Label31: TLabel;
    Label32: TLabel;
    Label33: TLabel;
    Label34: TLabel;
    Label35: TLabel;
    Label36: TLabel;
    Label37: TLabel;
    Button4: TButton;
    Edit1: TEdit;
    Label18: TLabel;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SnapShotClick(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure Timer2Timer(Sender: TObject);
    procedure DBGrid1CellClick(Column: TColumn);
    procedure Button1Click(Sender: TObject);
    procedure Timer3Timer(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure OnSelectDevice(sender: TObject);
  end;
```

```
var
 frmmain: Tfrmmain;
 SysDev: TSysDevEnum;
 files:integer;

implementation

uses uview;

{$R *.dfm}

procedure Tfrmmain.OnSelectDevice(sender: TObject);
begin
 FilterGraph.ClearGraph;
 FilterGraph.Active := false;
 Filter.BaseFilter.Moniker := SysDev.GetMoniker(TMenuItem(Sender).tag);
 FilterGraph.Active := true;
 with FilterGraph as ICaptureGraphBuilder2 do
  RenderStream(@PIN_CATEGORY_PREVIEW, nil, Filter as IBaseFilter, SampleGrabber as IBaseFilter,
VideoWindow as IbaseFilter);
 FilterGraph.Play;
end;

procedure Tfrmmain.Timer1Timer(Sender: TObject);
var Status:String;
  Index:Integer;
begin
 timer1.Enabled := false;
 Index := Pos('CH1:',Edit1.Text);
 if Index <> 0 then
 begin
```

```
Status := Copy(Edit1.Text,Index+4,1);
if Status = '0' then
  begin
  Image1.Visible := false;
  Label1.Transparent := false;
  Label1.Height:= 65;
  Label1.Width := 80;
  end
 else
  begin
  Image1.Visible := true;
  Label1.Transparent := true;
  end;
end;

Index := Pos('CH2:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
   begin
   Image2.Visible := false;
   Label2.Transparent := false;
   Label2.Height:= 65;
   Label2.Width := 80;
   end
  else
   begin
   Image2.Visible := true;
   Label2.Transparent := true;
   end;
```

```
end;

Index := Pos('CH3:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
  begin
  Image3.Visible := false;
  Label3.Transparent := false;
  Label3.Height:= 65;
  Label3.Width := 80;
  end
 else
  begin
  Image3.Visible := true;
  Label3.Transparent := true;
  end;
end;

Index := Pos('CH4:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
  begin
  Image4.Visible := false;
  Label4.Transparent := false;
  Label4.Height:= 65;
  Label4.Width := 80;
  end
```

```
  else
    begin
    Image4.Visible := true;
    Label4.Transparent := true;
    end;
end;

Index := Pos('CH5:',Edit1.Text);
if Index <> 0 then
begin
  Status := Copy(Edit1.Text,Index+4,1);
  if Status = '0' then
    begin
    Image5.Visible := false;
    Label5.Transparent := false;
    Label5.Height:= 65;
    Label5.Width := 80;
    end
  else
    begin
    Image5.Visible := true;
    Label5.Transparent := true;
    end;
end;

Index := Pos('CH6:',Edit1.Text);
if Index <> 0 then
begin
  Status := Copy(Edit1.Text,Index+4,1);
  if Status = '0' then
    begin
```

```
   Image6.Visible := false;
   Label6.Transparent := false;
   Label6.Height:= 65;
   Label6.Width := 80;
   end
  else
   begin
   Image6.Visible := true;
   Label6.Transparent := true;
   end;
end;

Index := Pos('CH7:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
   begin
   Image7.Visible := false;
   Label7.Transparent := false;
   Label7.Height:= 65;
   Label7.Width := 80;
   end
  else
   begin
   Image7.Visible := true;
   Label7.Transparent := true;
   end;
end;

Index := Pos('CH8:',Edit1.Text);
```

```
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
  begin
  Image8.Visible := false;
  Label8.Transparent := false;
  Label8.Height:= 65;
  Label8.Width := 80;
  end
 else
  begin
  Image8.Visible := true;
  Label8.Transparent := true;
  end;
end;

Index := Pos('CH9:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
  begin
  Image9.Visible := false;
  Label9.Transparent := false;
  Label9.Height:= 65;
  Label9.Width := 80;
  end
 else
  begin
  Image9.Visible := true;
```

```
   Label9.Transparent := true;
   end;
end;

Index := Pos('CH10:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+5,1);
 if Status = '0' then
   begin
   Image10.Visible := false;
   Label10.Transparent := false;
   Label10.Height:= 65;
   Label10.Width := 80;
   end
 else
   begin
   Image10.Visible := true;
   Label10.Transparent := true;
   end;
end;

Index := Pos('CH11:',Edit1.Text);
if Index <> 0 then
begin
  Status := Copy(Edit1.Text,Index+5,1);
  if Status = '0' then
    begin
    Image11.Visible := false;
    Label11.Transparent := false;
    Label11.Height:= 65;
```

```
  Label11.Width := 80;
  end
 else
  begin
  Image11.Visible := true;
  Label11.Transparent := true;
  end;
end;

Index := Pos('CH12:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+5,1);
 if Status = '0' then
  begin
  Image12.Visible := false;
  Label12.Transparent := false;
  Label12.Height:= 65;
  Label12.Width := 80;
  end
 else
  begin
  Image12.Visible := true;
  Label12.Transparent := true;
  end;
end;

Index := Pos('CH13:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+5,1);
```

```
if Status = '0' then
  begin
  Image13.Visible := false;
  Label13.Transparent := false;
  Label13.Height:= 65;
  Label13.Width := 80;
  end
 else
  begin
  Image13.Visible := true;
  Label13.Transparent := true;
  end;
end;

Index := Pos('CH14:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+5,1);
 if Status = '0' then
  begin
  Image14.Visible := false;
  Label14.Transparent := false;
  Label14.Height:= 65;
  Label14.Width := 80;
  end
 else
  begin
  Image14.Visible := true;
  Label14.Transparent := true;
  end;
end;
```

```
Index := Pos('IN:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+3,1);
 if Status = '0' then
  begin
  Image15.Visible := false;
  Label15.Transparent := false;
  Label15.Height:= 65;
  Label15.Width := 80;
  end
 else
  begin
  Image15.Visible := true;
  Label15.Transparent := true;
  timer2.Enabled := True;
  end;
end;

Index := Pos('OUT:',Edit1.Text);
if Index <> 0 then
begin
 Status := Copy(Edit1.Text,Index+4,1);
 if Status = '0' then
  begin
  //Image16.Visible := false;
  //Label16.Transparent := false;
  //Label16.Height:= 65;
  //Label16.Width := 80;
```

```
    end
   else
    begin
    //Image16.Visible := true;
    //Label16.Transparent := true;
    //timer2.Enabled := True;
     frmview := Tfrmview.Create(nil);
     frmview.ShowModal;
     frmview.Free;
    end;
 end;

end;

procedure Tfrmmain.FormCreate(Sender: TObject);
var
 i: integer;
 Device: TMenuItem;
begin
 SysDev:= TSysDevEnum.Create(CLSID_VideoInputDeviceCategory);
 if SysDev.CountFilters > 0 then
  for i := 0 to SysDev.CountFilters - 1 do
  begin
   Device := TMenuItem.Create(Devices);
   Device.Caption := SysDev.Filters[i].FriendlyName;
   Device.Tag := i;
   Device.OnClick := OnSelectDevice;
   Devices.Add(Device);
  end;
 end;
```

```
procedure Tfrmmain.SnapShotClick(Sender: TObject);
begin
 Image.Visible := true;
 SampleGrabber.GetBitmap(Image.Picture.Bitmap);

end;

procedure Tfrmmain.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
 SysDev.Free;
 FilterGraph.ClearGraph;
 FilterGraph.Active := false;
end;

procedure Tfrmmain.Timer2Timer(Sender: TObject);
begin
 Timer2.Enabled := False;
 Image.Visible := true;
 SampleGrabber.GetBitmap(Image.Picture.Bitmap);
 timer3.Enabled := true;

end;

procedure Tfrmmain.DBGrid1CellClick(Column: TColumn);
begin
 //Image.Picture.LoadFromFile('C:\cardata\Pic\'+Table1.FieldValues['File']+'.bmp');
end;

procedure Tfrmmain.Button1Click(Sender: TObject);
begin
{   Table1.Edit;
```

```
   Table1.FieldValues['Out']   := TimeToStr(now);
   Table1.FieldValues['TotalTime'] := TimeToStr(StrToTime(Table1.FieldValues['Out'])-
StrToTime(Table1.FieldValues['In']));
   Table1.Post;  DbiSaveChanges(Table1.Handle);
   Table1.Refresh;

   Report.ShowReport;}
end;

procedure Tfrmmain.Timer3Timer(Sender: TObject);
begin
    timer3.Enabled := False;

   if Table1.RecordCount > 0 then
   begin
    Table1.Last;
   files := strToint(Table1.FieldValues['File'])+1;
   end
   else
   begin
    files := 1;
   end;

   Table1.Append;
   Table1.FieldValues['Date'] := DateToStr(now);
   Table1.FieldValues['In']   := TimeToStr(now);
   Table1.FieldValues['File'] := intTostr(files);
   Table1.Post;  DbiSaveChanges(Table1.Handle);
   Table1.Refresh;

   Table1.Last;
```

```
  Image.Picture.SaveToFile('C:\cardata\Pic\'+Table1.FieldValues['File']+'.bmp');

end;

procedure Tfrmmain.Button2Click(Sender: TObject);
begin
   timer3.Enabled := False;

  if Table1.RecordCount > 0 then
  begin
   Table1.Last;
   files := strToint(Table1.FieldValues['File'])+1;
  end
  else
  begin
   files := 1;
  end;

  Table1.Append;
  Table1.FieldValues['Date'] := DateToStr(now);
  Table1.FieldValues['In']   := TimeToStr(now);
  Table1.FieldValues['File'] := intTostr(files);
  Table1.Post;  DbiSaveChanges(Table1.Handle);
  Table1.Refresh;

  Table1.Last;
  Image.Picture.SaveToFile('C:\cardata\Pic\'+Table1.FieldValues['File']+'.bmp');
end;

procedure Tfrmmain.Button3Click(Sender: TObject);
begin
```

```
  Image.Visible := false;
  timer1.Enabled := false;
end;

procedure Tfrmmain.Edit1Change(Sender: TObject);
begin
   timer1.Enabled := true;
end;

end.
```

Features

- Compatible with MCS-51® Products
- 8K Bytes of In-System Programmable (ISP) Flash Memory
  - Endurance: 1000 Write/Erase Cycles
- 4.0V to 5.5V Operating Range
- Fully Static Operation: 0 Hz to 33 MHz
- Three-level Program Memory Lock
- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Eight Interrupt Sources
- Full Duplex UART Serial Channel
- Low-power Idle and Power-down Modes
- Interrupt Recovery from Power-down Mode
- Watchdog Timer
- Dual Data Pointer
- Power-off Flag

Description

The AT89S52 is a low-power, high-performance CMOS 8-bit microcontroller with 8K bytes of in-system programmable Flash memory. The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 80C51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with in-system programmable Flash on a monolithic chip, the Atmel AT89S52 is a powerful microcontroller which provides a highly-flexible and cost-effective solution to many embedded control applications.

The AT89S52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89S52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset.

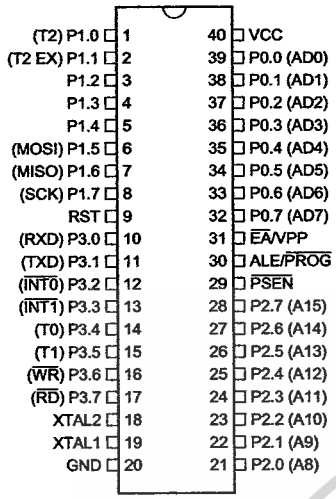
8-bit Microcontroller with 8K Bytes In-System Programmable Flash

AT89S52

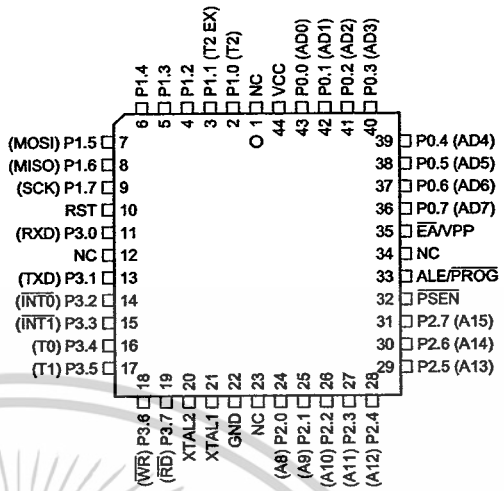
Rev. 1919A-07/01

Pin Configurations

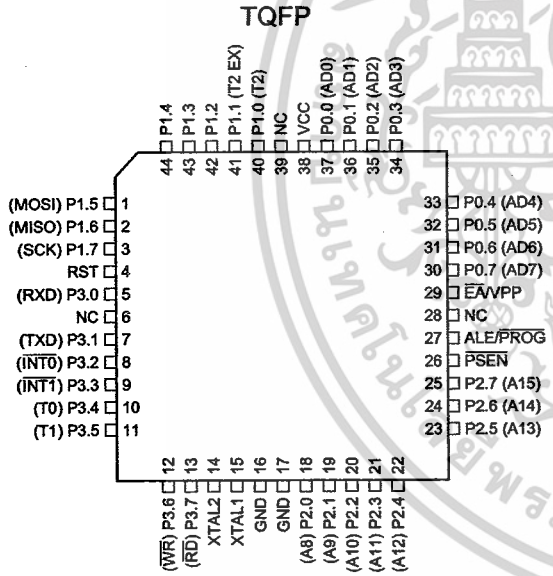
PDIP



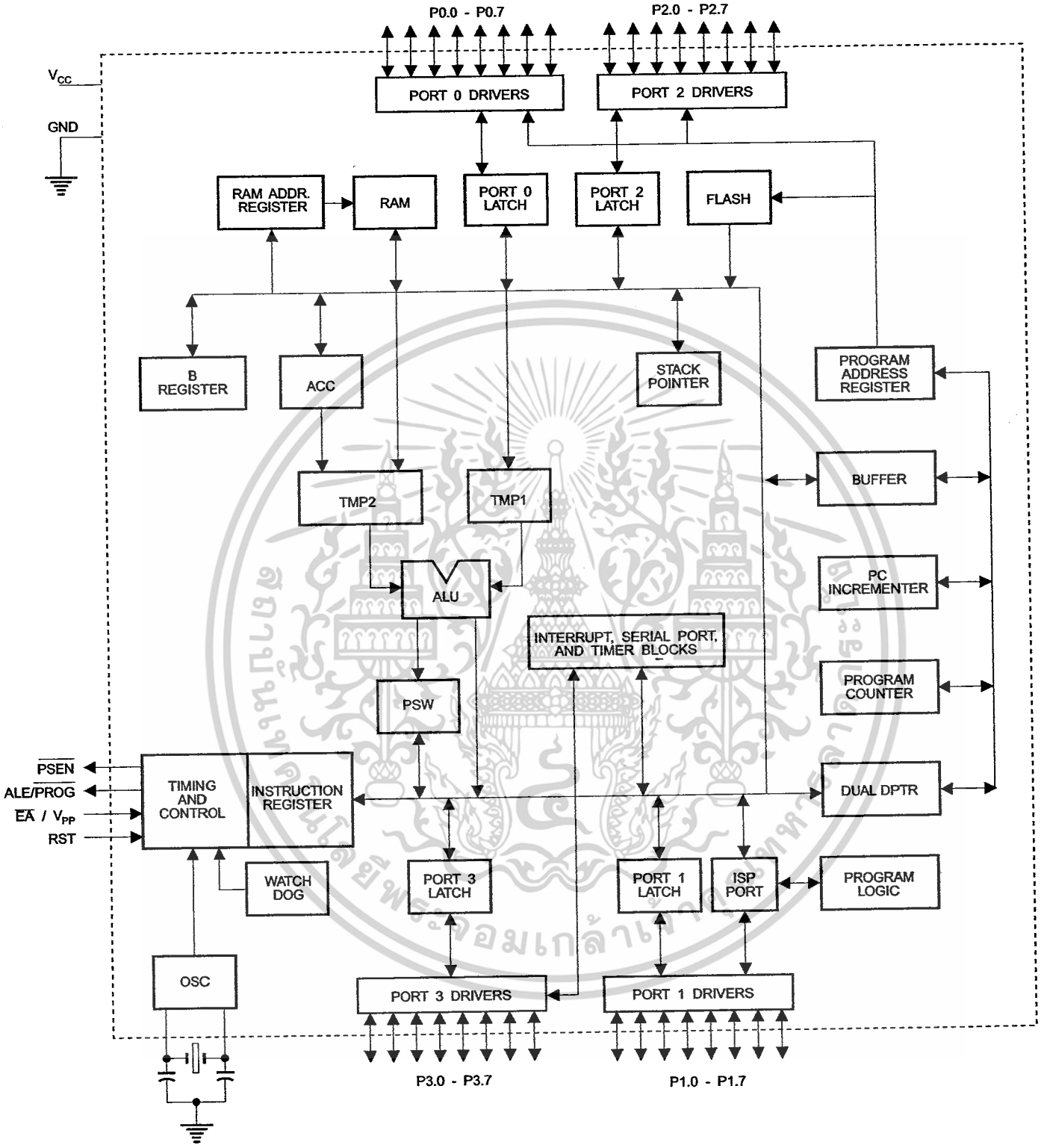
PLCC



TQFP



Block Diagram



Pin Description

VCC
Supply voltage.

GND
Ground.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1
Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

In addition, P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively, as shown in the following table.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port Pin	Alternate Functions
P1.0	T2 (external count input to Timer/Counter 2), clock-out
P1.1	T2EX (Timer/Counter 2 capture/reload trigger and direction control)
P1.5	MOSI (used for In-System Programming)
P1.6	MISO (used for In-System Programming)
P1.7	SCK (used for In-System Programming)

Port 2
Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89S52, as shown in the following table.

Port 3 also receives some control signals for Flash programming and verification.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST
Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. This pin drives High for 96 oscillator periods after the Watchdog times out. The DISRTO bit in SFR AUXR (address 8EH) can be used to disable this feature. In the default state of bit DISRTO, the RESET HIGH out feature is enabled.

ALE/$\overline{PROG}$
Address Latch Enable (ALE) is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external data memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is

veakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

PSEN

Program Store Enable ($\overline{PSEN}$) is the read strobe to external program memory.

When the AT89S52 is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory.

EA/VPP

External Access Enable. $\overline{EA}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming.

XTAL1

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

XTAL2

Output from the inverting oscillator amplifier.

Table 1. AT89S52 SFR Map and Reset Values

0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP XX000000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 0X000000								0AFH
0A0H	P2 11111111		AUXR1 XXXXXXX0				WDTRST XXXXXXXX		0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX00XX0		8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

Special Function Registers

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in Table 1.

Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect.

User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

Timer 2 Registers: Control and status bits are contained in registers T2CON (shown in Table 2) and T2MOD (shown in Table 3) for Timer 2. The register pair (RCAP2H, RCAP2L) are the Capture/Reload registers for Timer 2 in 16-bit capture mode or 16-bit auto-reload mode.

Interrupt Registers: The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the six interrupt sources in the IP register.

Table 2. T2CON – Timer/Counter 2 Control Register

T2CON Address = 0C8H — Reset Value = 0000 0000B

Bit Addressable

Bit	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$
	7	6	5	4	3	2	1	0

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

Table 3a. AUXR: Auxiliary Register

AUXR Address = 8EH Reset Value = XXX00XX0B

Not Bit Addressable

	–	–	–	WDIDLE	DISRTO	–	–	DISALE
Bit	7	6	5	4	3	2	1	0

–	Reserved for future expansion	
DISALE	Disable/Enable ALE	
	DISALE	Operating Mode
	0	ALE is emitted at a constant rate of 1/6 the oscillator frequency
	1	ALE is active only during a MOVX or MOVC instruction
DISRTO	Disable/Enable Reset out	
	DISRTO	
	0	Reset pin is driven High after WDT times out
	1	Reset pin is input only
WDIDLE	Disable/Enable WDT in IDLE mode	
	WDIDLE	
	0	WDT continues to count in IDLE mode
	1	WDT halts counting in IDLE mode

Dual Data Pointer Registers: To facilitate accessing both internal and external data memory, two banks of 16-bit Data Pointer Registers are provided: DP0 at SFR address locations 82H-83H and DP1 at 84H-85H. Bit DPS = 0 in SFR AUXR1 selects DP0 and DPS = 1 selects DP1. The user should always initialize the DPS bit to the appropriate value before accessing the respective Data Pointer Register.

Power Off Flag: The Power Off Flag (POF) is located at bit 4 (PCON.4) in the PCON SFR. POF is set to "1" during power up. It can be set and rest under software control and is not affected by reset.

Table 3b. AUXR1: Auxiliary Register 1

AUXR1 Address = A2H Reset Value = XXXXXXX0B

Not Bit Addressable

	–	–	–	–	–	–	–	DPS
Bit	7	6	5	4	3	2	1	0

–	Reserved for future expansion	
DPS	Data Pointer Register Select	
	DPS	
	0	Selects DPTR Registers DP0L, DP0H
	1	Selects DPTR Registers DP1L, DP1H

Memory Organization

MCS-51 devices have a separate address space for Program and Data Memory. Up to 64K bytes each of external Program and Data Memory can be addressed.

Program Memory

If the $\overline{EA}$ pin is connected to GND, all program fetches are directed to external memory.

On the AT89S52, if $\overline{EA}$ is connected to V_{CC}, program fetches to addresses 0000H through 1FFFH are directed to internal memory and fetches to addresses 2000H through FFFFH are to external memory.

Data Memory

The AT89S52 implements 256 bytes of on-chip RAM. The upper 128 bytes occupy a parallel address space to the Special Function Registers. This means that the upper 128 bytes have the same addresses as the SFR space but are physically separate from SFR space.

When an instruction accesses an internal location above address 7FH, the address mode used in the instruction specifies whether the CPU accesses the upper 128 bytes of RAM or the SFR space. Instructions which use direct addressing access of the SFR space.

For example, the following direct addressing instruction accesses the SFR at location 0A0H (which is P2).

```
MOV 0A0H, #data
```

Instructions that use indirect addressing access the upper 128 bytes of RAM. For example, the following indirect addressing instruction, where R0 contains 0A0H, accesses the data byte at address 0A0H, rather than P2 (whose address is 0A0H).

```
MOV @R0, #data
```

Note that stack operations are examples of indirect addressing, so the upper 128 bytes of data RAM are available as stack space.

Watchdog Timer (One-time Enabled with Reset-out)

The WDT is intended as a recovery method in situations where the CPU may be subjected to software upsets. The WDT consists of a 13-bit counter and the Watchdog Timer Reset (WDTRST) SFR. The WDT is defaulted to disable from exiting reset. To enable the WDT, a user must write 01EH and 0E1H in sequence to the WDTRST register (SFR location 0A6H). When the WDT is enabled, it will increment every machine cycle while the oscillator is running. The WDT timeout period is dependent on the external clock frequency. There is no way to disable the WDT except through reset (either hardware reset or WDT overflow reset). When WDT overflows, it will drive an output RESET HIGH pulse at the RST pin.

Using the WDT

To enable the WDT, a user must write 01EH and 0E1H in sequence to the WDTRST register (SFR location 0A6H). When the WDT is enabled, the user needs to service it by writing 01EH and 0E1H to WDTRST to avoid a WDT overflow. The 13-bit counter overflows when it reaches 8191 (1FFFH), and this will reset the device. When the WDT is enabled, it will increment every machine cycle while the oscillator is running. This means the user must reset the WDT at least every 8191 machine cycles. To reset the WDT the user must write 01EH and 0E1H to WDTRST. WDTRST is a write-only register. The WDT counter cannot be read or written. When WDT overflows, it will generate an output RESET pulse at the RST pin. The RESET pulse duration is 96xTOSC, where TOSC=1/FOSC. To make the best use of the WDT, it should be serviced in those sections of code that will periodically be executed within the time required to prevent a WDT reset.

WDT During Power-down and Idle

In Power-down mode the oscillator stops, which means the WDT also stops. While in Power-down mode, the user does not need to service the WDT. There are two methods of exiting Power-down mode: by a hardware reset or via a level-activated external interrupt which is enabled prior to entering Power-down mode. When Power-down is exited with hardware reset, servicing the WDT should occur as it normally does whenever the AT89S52 is reset. Exiting Power-down with an interrupt is significantly different. The interrupt is held low long enough for the oscillator to stabilize. When the interrupt is brought high, the interrupt is serviced. To prevent the WDT from resetting the device while the interrupt pin is held low, the WDT is not started until the interrupt is pulled high. It is suggested that the WDT be reset during the interrupt service for the interrupt used to exit Power-down mode.

To ensure that the WDT does not overflow within a few states of exiting Power-down, it is best to reset the WDT just before entering Power-down mode.

Before going into the IDLE mode, the WDIDLE bit in SFR AUXR is used to determine whether the WDT continues to count if enabled. The WDT keeps counting during IDLE (WDIDLE bit = 0) as the default state. To prevent the WDT from resetting the AT89S52 while in IDLE mode, the user should always set up a timer that will periodically exit IDLE, service the WDT, and reenter IDLE mode.

With WDIDLE bit enabled, the WDT will stop to count in IDLE mode and resumes the count upon exit from IDLE.

UART

The UART in the AT89S52 operates the same way as the UART in the AT89C51 and AT89C52. For further information on the UART operation, refer to the ATMEL Web site (http://www.atmel.com). From the home page, select 'Products', then '8051-Architecture Flash Microcontroller', then 'Product Overview'.

Timer 0 and 1

Timer 0 and Timer 1 in the AT89S52 operate the same way as Timer 0 and Timer 1 in the AT89C51 and AT89C52. For further information on the timers' operation, refer to the ATMEL Web site (http://www.atmel.com). From the home page, select 'Products', then '8051-Architecture Flash Microcontroller', then 'Product Overview'.

Timer 2

Timer 2 is a 16-bit Timer/Counter that can operate as either a timer or an event counter. The type of operation is selected by bit C/$\overline{T2}$ in the SFR T2CON (shown in Table 2). Timer 2 has three operating modes: capture, auto-reload (up or down counting), and baud rate generator. The modes are selected by bits in T2CON, as shown in Table 3. Timer 2 consists of two 8-bit registers, TH2 and TL2. In the Timer function, the TL2 register is incremented every machine cycle. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

Table 3. Timer 2 Operating Modes

RCLK +TCLK	CP/$\overline{RL2}$	TR2	MODE
0	0	1	16-bit Auto-reload
0	1	1	16-bit Capture
1	X	1	Baud Rate Generator
X	X	0	(Off)

In the Counter function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since two machine cycles (24 oscillator periods) are required to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. To ensure that a given level is sampled at least once before it changes, the level should be held for at least one full machine cycle.

Capture Mode

In the capture mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 is a 16-bit timer or counter which upon overflow sets bit TF2 in T2CON. This bit can then be used to generate an interrupt. If EXEN2 = 1, Timer 2 performs the same operation, but a 1-to-0 transition at external input T2EX also causes the current value in TH2 and TL2 to be captured into RCAP2H and RCAP2L, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set. The EXF2 bit, like TF2, can generate an interrupt. The capture mode is illustrated in Figure 5.

Auto-reload (Up or Down Counter)

Timer 2 can be programmed to count up or down when configured in its 16-bit auto-reload mode. This feature is invoked by the DCEN (Down Counter Enable) bit located in the SFR T2MOD (see Table 4). Upon reset, the DCEN bit is set to 0 so that timer 2 will default to count up. When DCEN is set, Timer 2 can count up or down, depending on the value of the T2EX pin.

Figure 5. Timer in Capture Mode

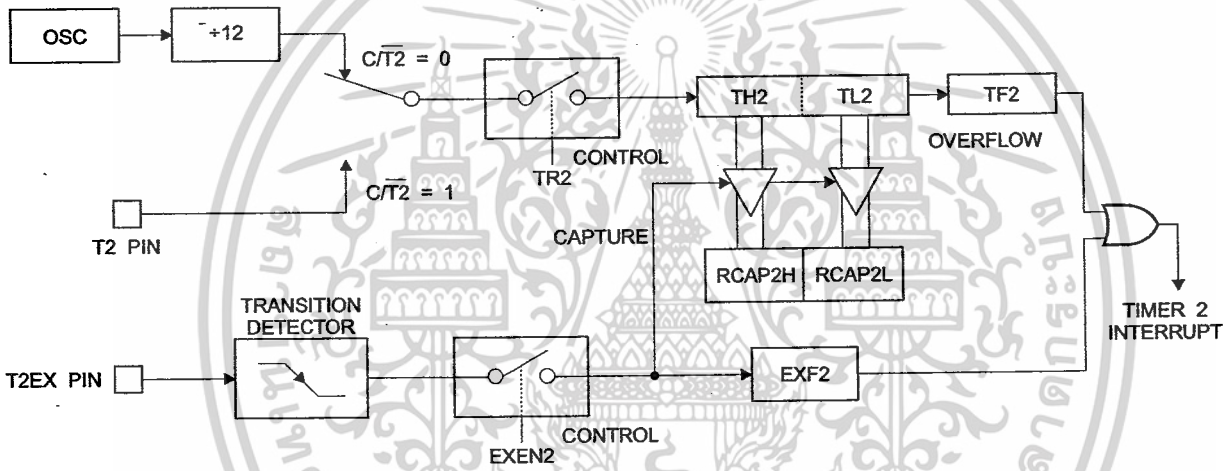


Figure 6 shows Timer 2 automatically counting up when DCEN=0. In this mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 counts up to 0FFFFH and then sets the TF2 bit upon overflow. The overflow also causes the timer registers to be reloaded with the 16-bit value in RCAP2H and RCAP2L. The values in Timer in Capture ModeRCAP2H and RCAP2L are preset by software. If EXEN2 = 1, a 16-bit reload can be triggered either by an overflow or by a 1-to-0 transition at external input T2EX. This transition also sets the EXF2 bit. Both the TF2 and EXF2 bits can generate an interrupt if enabled.

Setting the DCEN bit enables Timer 2 to count up or down, as shown in Figure 6. In this mode, the T2EX pin controls the direction of the count. A logic 1 at T2EX makes Timer 2 count up. The timer will overflow at 0FFFFH and set the TF2 bit. This overflow also causes the 16-bit value in RCAP2H and RCAP2L to be reloaded into the timer registers, TH2 and TL2, respectively.

A logic 0 at T2EX makes Timer 2 count down. The timer underflows when TH2 and TL2 equal the values stored in RCAP2H and RCAP2L. The underflow sets the TF2 bit and causes 0FFFFH to be reloaded into the timer registers.

The EXF2 bit toggles whenever Timer 2 overflows or underflows and can be used as a 17th bit of resolution. In this operating mode, EXF2 does not flag an interrupt.

Figure 6. Timer 2 Auto Reload Mode (DCEN = 0)

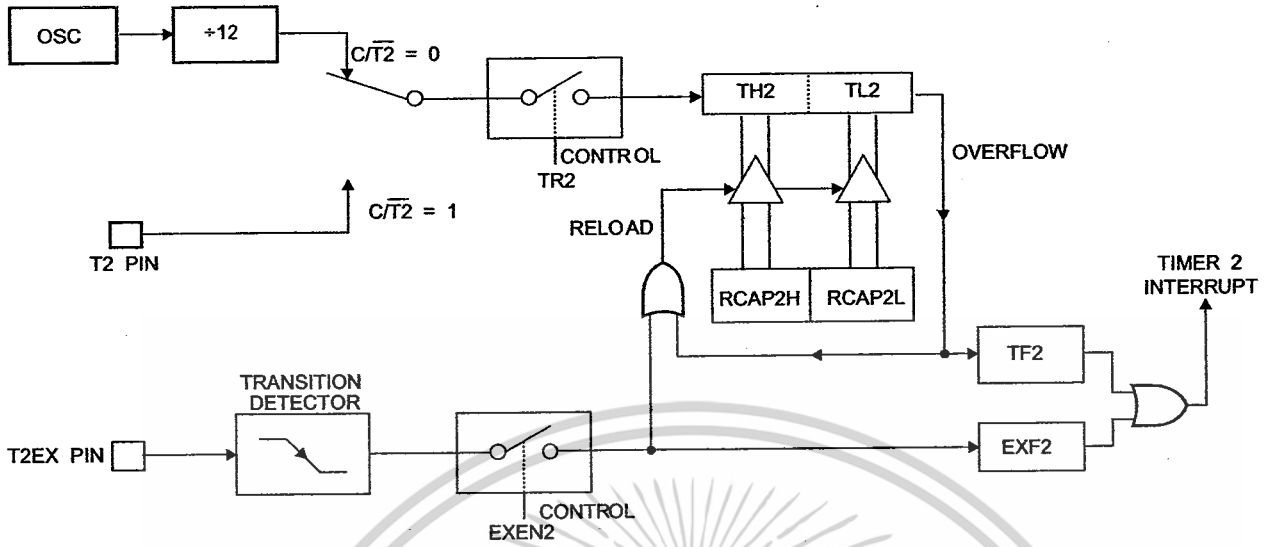


Table 4. T2MOD – Timer 2 Mode Control Register

T2MOD Address = 0C9H — Reset Value = XXXX XX00B

Not Bit Addressable

Bit	7	6	5	4	3	2	1	0
	–	–	–	–	–	–	T2OE	DCEN

Symbol	Function
–	Not implemented, reserved for future
T2OE	Timer 2 Output Enable bit
DCEN	When set, this bit allows Timer 2 to be configured as an up/down counter

Figure 7. Timer 2 Auto Reload Mode (DCEN = 1)

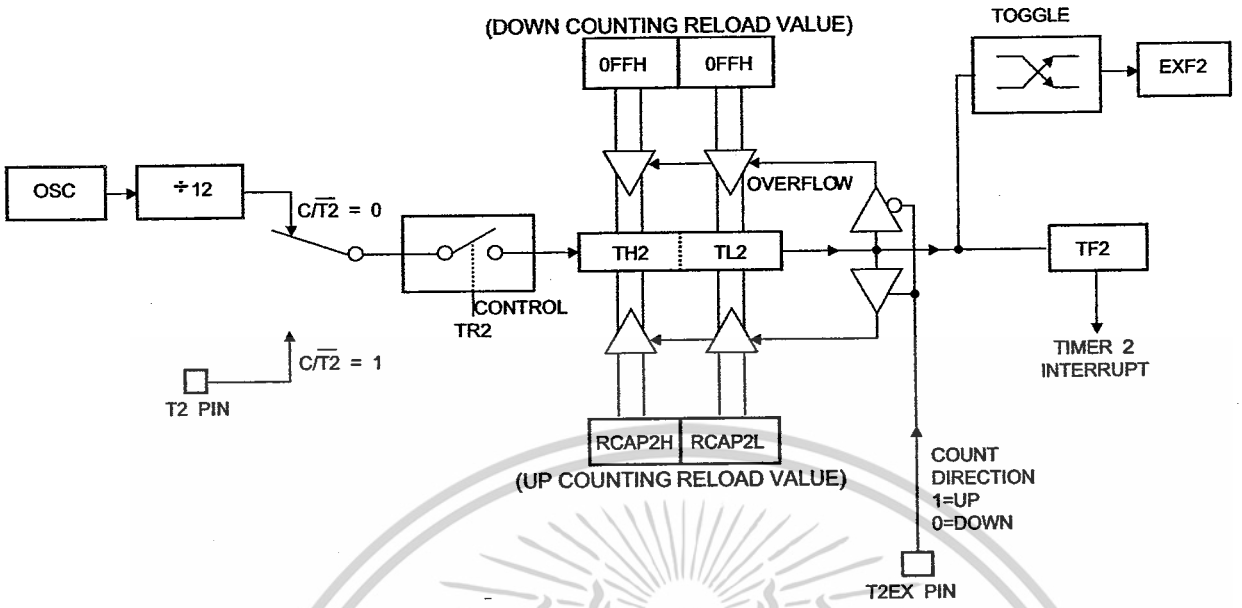
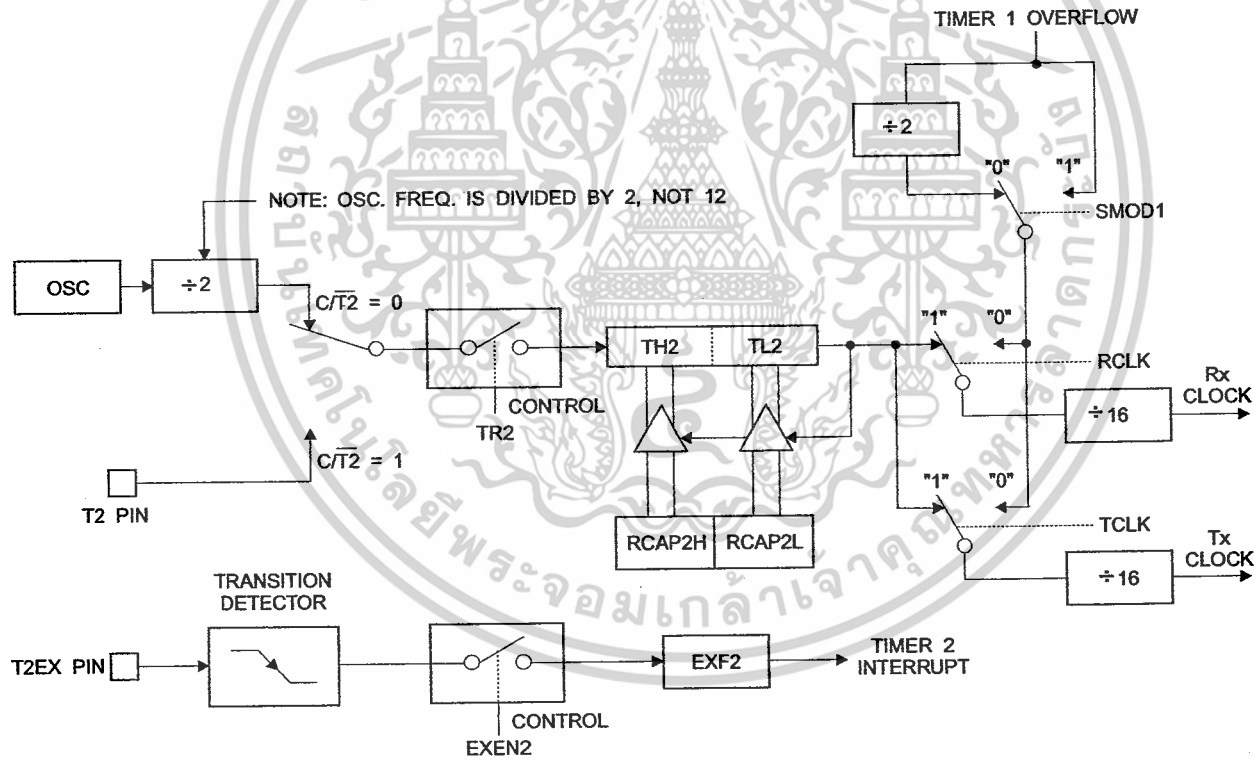


Figure 8. Timer 2 in Baud Rate Generator Mode



Baud Rate Generator

Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Table 2). Note that the baud rates for transmit and receive can be different if Timer 2 is used for the receiver or transmitter and Timer 1 is used for the other function. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 8.

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

The baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate according to the following equation.

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

The Timer can be configured for either timer or counter operation. In most applications, it is configured for timer operation ($C/\overline{T2} = 0$). The timer operation is different for Timer 2 when it is used as a baud rate generator. Normally, as a timer, it increments every machine cycle (at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (at 1/2 the oscillator frequency). The baud rate formula is given below.

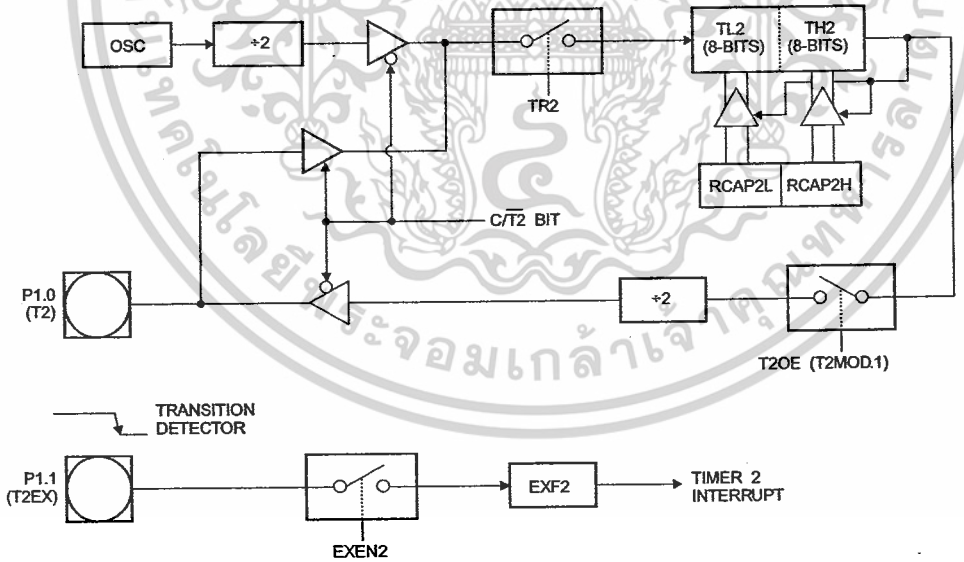
$$\frac{\text{Modes 1 and 3}}{\text{Baud Rate}} = \frac{\text{Oscillator Frequency}}{32 \times [65536\text{-RCAP2H,RCAP2L})]}$$

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

Timer 2 as a baud rate generator is shown in Figure 8. This figure is valid only if RCLK or TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2 and will not generate an interrupt. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus, when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt.

Note that when Timer 2 is running (TR2 = 1) as a timer in the baud rate generator mode, TH2 or TL2 should not be read from or written to. Under these conditions, the Timer is incremented every state time, and the results of a read or write may not be accurate. The RCAP2 registers may be read but should not be written to, because a write might overlap a reload and cause write and/or reload errors. The timer should be turned off (clear TR2) before accessing the Timer 2 or RCAP2 registers.

Figure 9. Timer 2 in Clock-Out Mode



Programmable Clock Out

A 50% duty cycle clock can be programmed to come out on P1.0, as shown in Figure 9. This pin, besides being a regular I/O pin, has two alternate functions. It can be programmed to input the external clock for Timer/Counter 2 or to output a 50% duty cycle clock ranging from 61 Hz to 4 MHz at a 16 MHz operating frequency.

To configure the Timer/Counter 2 as a clock generator, bit $C/\overline{T2}$ (T2CON.1) must be cleared and bit T2OE (T2MOD.1) must be set. Bit TR2 (T2CON.2) starts and stops the timer.

The clock-out frequency depends on the oscillator frequency and the reload value of Timer 2 capture registers (RCAP2H, RCAP2L), as shown in the following equation.

$$\text{Clock-Out Frequency} = \frac{\text{Oscillator Frequency}}{4 \times [65536-(\text{RCAP2H,RCAP2L})]}$$

In the clock-out mode, Timer 2 roll-overs will not generate an interrupt. This behavior is similar to when Timer 2 is used as a baud-rate generator. It is possible to use Timer 2 as a baud-rate generator and a clock generator simultaneously. Note, however, that the baud-rate and clock-out frequencies cannot be determined independently from one another since they both use RCAP2H and RCAP2L.

Interrupts

The AT89S52 has a total of six interrupt vectors: two external interrupts ($\overline{INT0}$ and $\overline{INT1}$), three timer interrupts (Timers 0, 1, and 2), and the serial port interrupt. These interrupts are all shown in Figure 10.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once.

Note that Table 5 shows that bit position IE.6 is unimplemented. In the AT89S52, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future AT89 products.

Timer 2 interrupt is generated by the logical OR of bits TF2 and EXF2 in register T2CON. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and that bit will have to be cleared in software.

The Timer 0 and Timer 1 flags, TF0 and TF1, are set at S5P2 of the cycle in which the timers overflow. The values are then polled by the circuitry in the next cycle. However, the Timer 2 flag, TF2, is set at S2P2 and is polled in the same cycle in which the timer overflows.

Table 5. Interrupt Enable (IE) Register

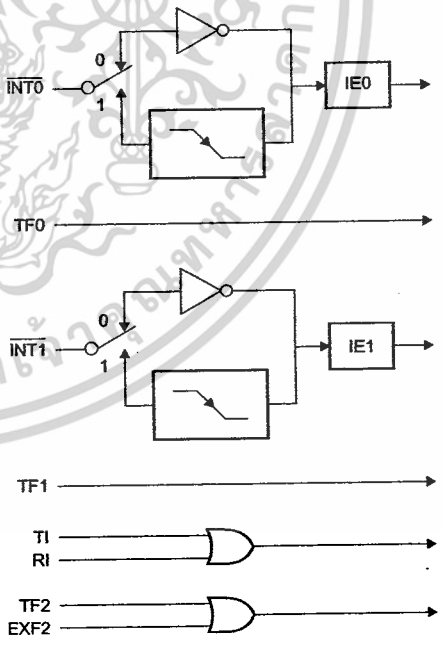
(MSB)							(LSB)
EA	–	ET2	ES	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.

Symbol	Position	Function
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	Reserved.
ET2	IE.5	Timer 2 interrupt enable bit.
ES	IE.4	Serial Port interrupt enable bit.
ET1	IE.3	Timer 1 interrupt enable bit.
EX1	IE.2	External interrupt 1 enable bit.
ET0	IE.1	Timer 0 interrupt enable bit.
EX0	IE.0	External interrupt 0 enable bit.
User software should never write 1s to unimplemented bits, because they may be used in future AT89 products.		

Figure 10. Interrupt Sources



Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier that can be configured for use as an on-chip oscillator, as shown in Figure 11. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven, as shown in Figure 12. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

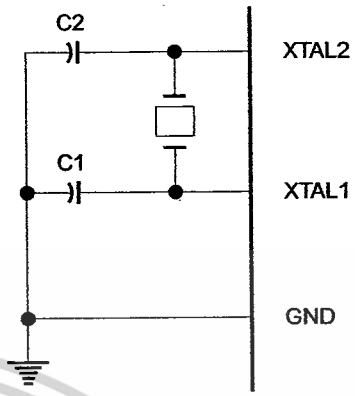
In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

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

Power-down Mode

In the Power-down mode, the oscillator is stopped, and the instruction that invokes Power-down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the Power-down mode is terminated. Exit from Power-down mode can be initiated either by a hardware reset or by an enabled external interrupt. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Figure 11. Oscillator Connections



Note: C1, C2 = 30 pF ±10 pF for Crystals
= 40 pF ±10 pF for Ceramic Resonators

Figure 12. External Clock Drive Configuration

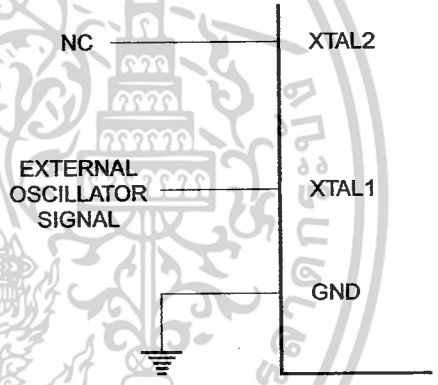


Table 6. Status of External Pins During Idle and Power-down Modes

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power-down	Internal	0	0	Data	Data	Data	Data
Power-down	External	0	0	Float	Data	Data	Data

Program Memory Lock Bits

The AT89S52 has three lock bits that can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the following table.

Table 7. Lock Bit Protection Modes

Program Lock Bits				
	LB1	LB2	LB3	Protection Type
1	U	U	U	No program lock features
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the Flash memory is disabled
3	P	P	U	Same as mode 2, but verify is also disabled
4	P	P	P	Same as mode 3, but external execution is also disabled

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value and holds that value until reset is activated. The latched value of $\overline{EA}$ must agree with the current logic level at that pin in order for the device to function properly.

Programming the Flash – Parallel Mode

The AT89S52 is shipped with the on-chip Flash memory array ready to be programmed. The programming interface needs a high-voltage (12-volt) program enable signal and is compatible with conventional third-party Flash or EPROM programmers.

The AT89S52 code memory array is programmed byte-by-byte.

Programming Algorithm: Before programming the AT89S52, the address, data, and control signals should be set up according to the Flash programming mode table and Figures 13 and 14. To program the AT89S52, take the following steps:

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise $\overline{EA}/V_{PP}$ to 12V.
5. Pulse ALE/$\overline{PROG}$ once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 50 µs.

Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

$\overline{Data}$ Polling: The AT89S52 features $\overline{Data}$ Polling to indicate the end of a byte write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on P0.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. $\overline{Data}$ Polling may begin any time after a write cycle has been initiated.

Ready/$\overline{Busy}$: The progress of byte programming can also be monitored by the RDY/$\overline{BSY}$ output signal. P3.0 is pulled low after ALE goes high during programming to indicate $\overline{BUSY}$. P3.0 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The status of the individual lock bits can be verified directly by reading them back.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 000H, 100H, and 200H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(000H) = 1EH indicates manufactured by Atmel
(100H) = 52H indicates 89S52
(200H) = 06H

Chip Erase: In the parallel programming mode, a chip erase operation is initiated by using the proper combination of control signals and by pulsing ALE/$\overline{PROG}$ low for a duration of 200 ns - 500 ns.

In the serial programming mode, a chip erase operation is initiated by issuing the Chip Erase instruction. In this mode, chip erase is self-timed and takes about 500 ms.

During chip erase, a serial read from any address location will return 00H at the data output.

Programming the Flash – Serial Mode

The Code memory array can be programmed using the serial ISP interface while RST is pulled to V_{CC}. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After RST is set high, the Programming Enable instruction needs to be executed first before other operations can be executed. Before a reprogramming sequence can occur, a Chip Erase operation is required.

The Chip Erase operation turns the content of every memory location in the Code array into FFH.

Either an external system clock can be supplied at pin XTAL1 or a crystal needs to be connected across pins XTAL1 and XTAL2. The maximum serial clock (SCK)

requency should be less than 1/16 of the crystal frequency. With a 33 MHz oscillator clock, the maximum SCK requency is 2 MHz.

Serial Programming Algorithm

To program and verify the AT89S52 in the serial programming mode, the following sequence is recommended:

1. Power-up sequence:

   Apply power between VCC and GND pins.

   Set RST pin to "H".

   If a crystal is not connected across pins XTAL1 and XTAL2, apply a 3 MHz to 33 MHz clock to XTAL1 pin and wait for at least 10 milliseconds.
2. Enable serial programming by sending the Programming Enable serial instruction to pin MOSI/P1.5. The frequency of the shift clock supplied at pin SCK/P1.7 needs to be less than the CPU clock at XTAL1 divided by 16.
3. The Code array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. The write cycle is self-timed and typically takes less than 1 ms at 5V.
4. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output MISO/P1.6.
5. At the end of a programming session, RST can be set low to commence normal device operation.

Power-off sequence (if needed):

Set XTAL1 to "L" (if a crystal is not used).

Set RST to "L".

Turn V_{CC} power off.

Data Polling: The Data Polling feature is also available in the serial mode. In this mode, during a write cycle an attempted read of the last byte written will result in the complement of the MSB of the serial output byte on MISO.

Serial Programming Instruction Set

The Instruction Set for Serial Programming follows a 4-byte protocol and is shown in Table 10.

Programming Interface – Parallel Mode

Every code byte in the Flash array can be programmed by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Table 8. Flash Programming Modes

Mode	V_{CC}	RST	$\overline{PSEN}$	ALE/ $\overline{PROG}$	$\overline{EA}$/ V_{PP}	P2.6	P2.7	P3.3	P3.6	P3.7	P0.7-0 Data	P2.4-0 Address	P1.7-0 Address
Write Code Data	5V	H	L	(2)	12V	L	H	H	H	H	D_{IN}	A12-8	A7-0
Read Code Data	5V	H	L	H	H	L	L	L	H	H	D_{OUT}	A12-8	A7-0
Write Lock Bit 1	5V	H	L	(3)	12V	H	H	H	H	H	X	X	X
Write Lock Bit 2	5V	H	L	(3)	12V	H	H	H	L	L	X	X	X
Write Lock Bit 3	5V	H	L	(3)	12V	H	L	H	H	L	X	X	X
Read Lock Bits 1, 2, 3	5V	H	L	H	H	H	H	L	H	L	P0.2, P0.3, P0.4	X	X
Chip Erase	5V	H	L	(1)	12V	H	L	H	L	L	X	X	X
Read Atmel ID	5V	H	L	H	H	L	L	L	L	L	1EH	X 0000	00H
Read Device ID	5V	H	L	H	H	L	L	L	L	L	52H	X 0001	00H
Read Device ID	5V	H	L	H	H	L	L	L	L	L	06H	X 0010	00H

Notes:
1. Each $\overline{PROG}$ pulse is 200 ns - 500 ns for Chip Erase.
2. Each $\overline{PROG}$ pulse is 200 ns - 500 ns for Write Code Data.
3. Each $\overline{PROG}$ pulse is 200 ns - 500 ns for Write Lock Bits.
4. RDY/$\overline{BSY}$ signal is output on P3.0 during programming.
5. X = don't care.

Figure 13. Programming the Flash Memory (Parallel Mode)

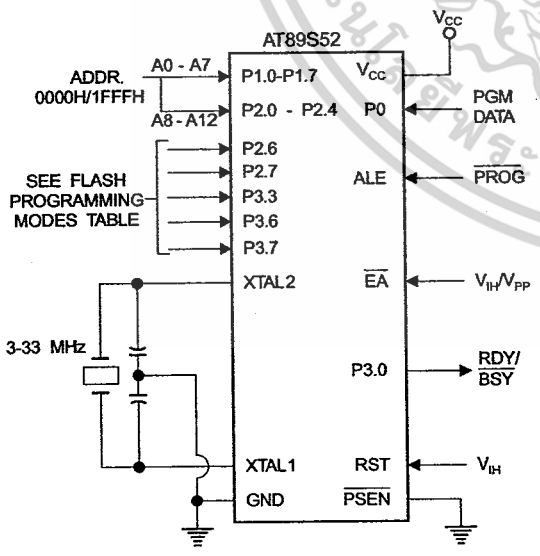
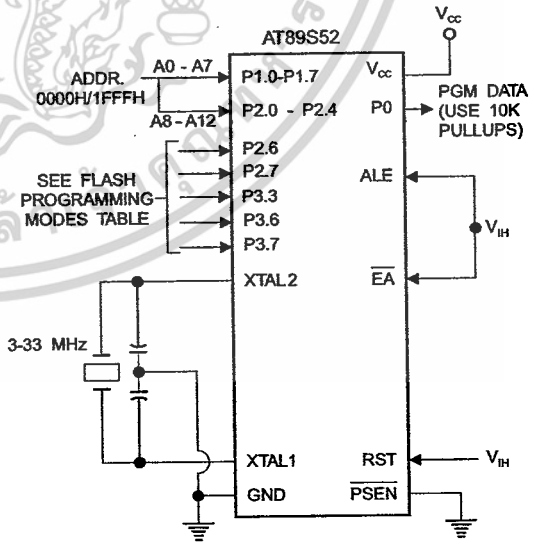


Figure 14. Verifying the Flash Memory (Parallel Mode)



Flash Programming and Verification Characteristics (Parallel Mode)

T_A = 20°C to 30°C, V_{CC} = 4.5 to 5.5V

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Supply Voltage	11.5	12.5	V
I_{PP}	Programming Supply Current		10	mA
I_{CC}	V_{CC} Supply Current		30	mA
$1/t_{CLCL}$	Oscillator Frequency	3	33	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{EHSH}	P2.7 ($\overline{ENABLE}$) High to V_{PP}	$48t_{CLCL}$		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}	V_{PP} Hold After $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	0.2	1	µs
t_{AVQV}	Address to Data Valid		$48t_{CLCL}$	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQZ}	Data Float After $\overline{ENABLE}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		1.0	µs
t_{WC}	Byte Write Cycle Time		50	µs

Figure 15. Flash Programming and Verification Waveforms – Parallel Mode

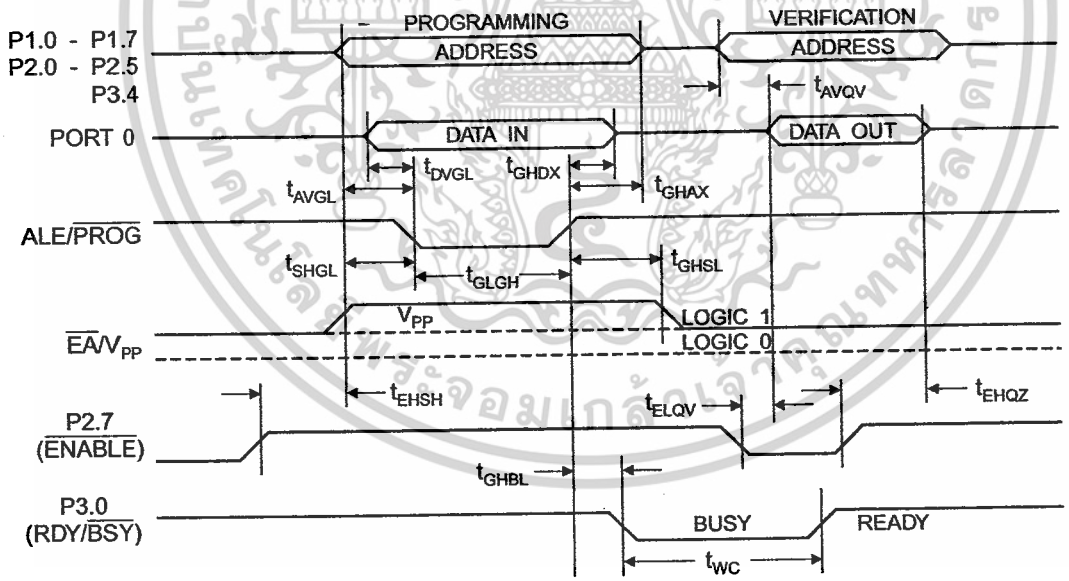
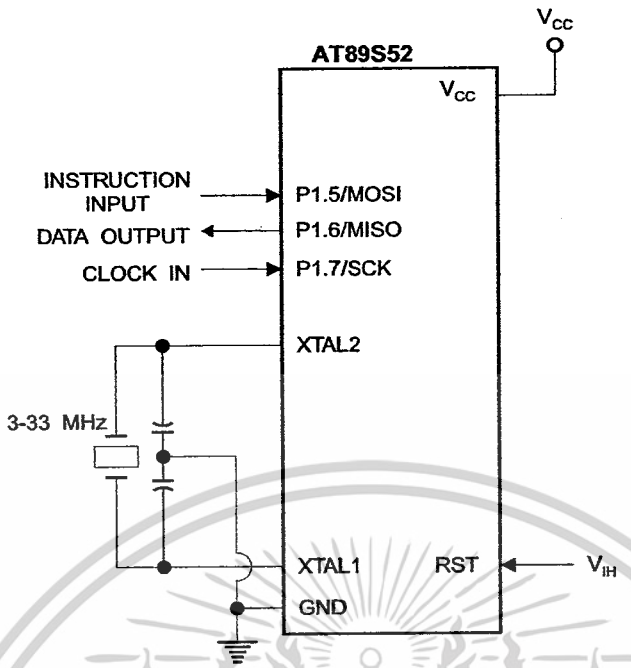


Figure 16. Flash Memory Serial Downloading



Flash Programming and Verification Waveforms – Serial Mode

Figure 17. Serial Programming Waveforms

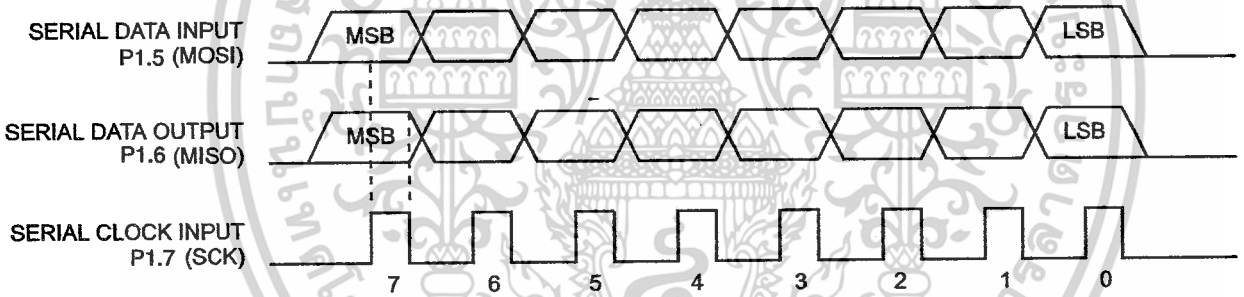


Table 9. Serial Programming Instruction Set

Instruction	Instruction Format Byte 1	Byte 2	Byte 3	Byte 4	Operation
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx 0110 1001 (Output)	Enable Serial Programming while RST is high
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase Flash memory array
Read Program Memory (Byte Mode)	0010 0000	xxx A12 A11 A10 A9 A8	A7 A6 A5 A4 A3 A2 A1 A0	D7 D6 D5 D4 D3 D2 D1 D0	Read data from Program memory in the byte mode
Write Program Memory (Byte Mode)	0100 0000	xxx A12 A11 A10 A9 A8	A7 A6 A5 A4 A3 A2 A1 A0	D7 D6 D5 D4 D3 D2 D1 D0	Write data to Program memory in the byte mode
Write Lock Bits[2]	1010 1100	1110 00 B1 B2	xxxx xxxx	xxxx xxxx	Write Lock bits. See Note (2).
Read Lock Bits	0010 0100	xxxx xxxx	xxxx xxxx	xx LB3 LB2 LB1 xx	Read back current status of the lock bits (a programmed lock bit reads back as a '1')
Read Signature Bytes[1]	0010 1000	xxx A5 A4 A3 A2 A1 A0	xxx xxxx	Signature Byte	Read Signature Byte
Read Program Memory (Page Mode)	0011 0000	xxx A12 A11 A10 A9 A8	Byte 0	Byte 1... Byte 255	Read data from Program memory in the Page Mode (256 bytes)
Write Program Memory (Page Mode)	0101 0000	xxx A12 A11 A10 A9 A8	Byte 0	Byte 1... Byte 255	Write data to Program memory in the Page Mode (256 bytes)

Notes: 1. The signature bytes are not readable in Lock Bit Modes 3 and 4.

2. B1 = 0, B2 = 0 —> Mode 1, no lock protection
   B1 = 0, B2 = 1 —> Mode 2, lock bit 1 activated
   B1 = 1, B2 = 0 —> Mode 3, lock bit 2 activated
   B1 = 1, B1 = 1 —> Mode 4, lock bit 3 activated

   } Each of the lock bits needs to be activated sequentially before Mode 4 can be executed.

After Reset signal is high, SCK should be low for at least 64 system clocks before it goes high to clock in the enable data bytes. No pulsing of Reset signal is necessary. SCK should be no faster than 1/16 of the system clock at XTAL1.

For Page Read/Write, the data always starts from byte 0 to 255. After the command byte and upper address byte are latched, each byte thereafter is treated as data until all 256 bytes are shifted in/out. Then the next instruction will be ready to be decoded.

Serial Programming Characteristics

Figure 18. Serial Programming Timing

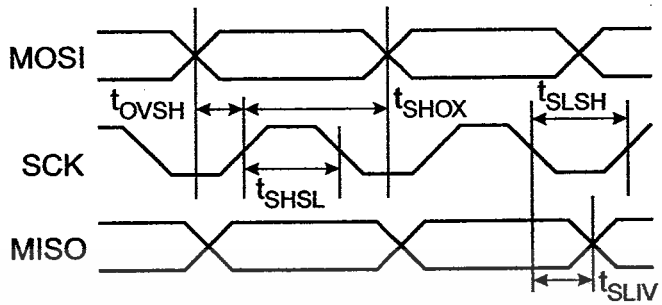


Table 10. Serial Programming Characteristics, T_A = -40° C to 85° C, V_{CC} = 4.0 - 5.5V (Unless otherwise noted)

Symbol	Parameter	Min	Typ	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0		33	MHz
t_{CLCL}	Oscillator Period	30			ns
t_{SHSL}	SCK Pulse Width High	$2\ t_{CLCL}$			ns
t_{SLSH}	SCK Pulse Width Low	$2\ t_{CLCL}$			ns
t_{OVSH}	MOSI Setup to SCK High	t_{CLCL}			ns
t_{SHOX}	MOSI Hold after SCK High	$2\ t_{CLCL}$			ns
t_{SLIV}	SCK Low to MISO Valid	10	16	32	ns
t_{ERASE}	Chip Erase Instruction Cycle Time			500	ms
t_{SWC}	Serial Byte Write Cycle Time		-	$64\ t_{CLCL} + 400$	µs

Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0V to +7.0V
Maximum Operating Voltage	6.6V
DC Output Current	15.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

The values shown in this table are valid for T_A = -40°C to 85°C and V_{CC} = 4.0V to 5.5V, unless otherwise noted.

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low Voltage	(Except $\overline{EA}$)	-0.5	0.2 V_{CC}-0.1	V
V_{IL1}	Input Low Voltage ($\overline{EA}$)		-0.5	0.2 V_{CC}-0.3	V
V_{IH}	Input High Voltage	(Except XTAL1, RST)	0.2 V_{CC}+0.9	V_{CC}+0.5	V
V_{IH1}	Input High Voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC}+0.5	V
V_{OL}	Output Low Voltage[1] (Ports 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Output Low Voltage[1] (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High Voltage (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA, V_{CC} = 5V ±10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)	I_{OH} = -800 µA, V_{CC} = 5V ±10%	2.4		V
		I_{OH} = -300 µA	0.75 V_{CC}		V
		I_{OH} = -80 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1,2,3)	V_{IN} = 0.45V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	V_{IN} = 2V, V_{CC} = 5V ±10%		-650	µA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V_{IN} < V_{CC}		±10	µA
RRST	Reset Pulldown Resistor		10	30	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		25	mA
		Idle Mode, 12 MHz		6.5	mA
	Power-down Mode[1]	V_{CC} = 5.5V		50	µA

Notes:
1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 10 mA
   Maximum I_{OL} per 8-bit port:
   Port 0: 26 mA Ports 1, 2, 3: 15 mA
   Maximum total I_{OL} for all output pins: 71 mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum V_{CC} for Power-down is 2V.

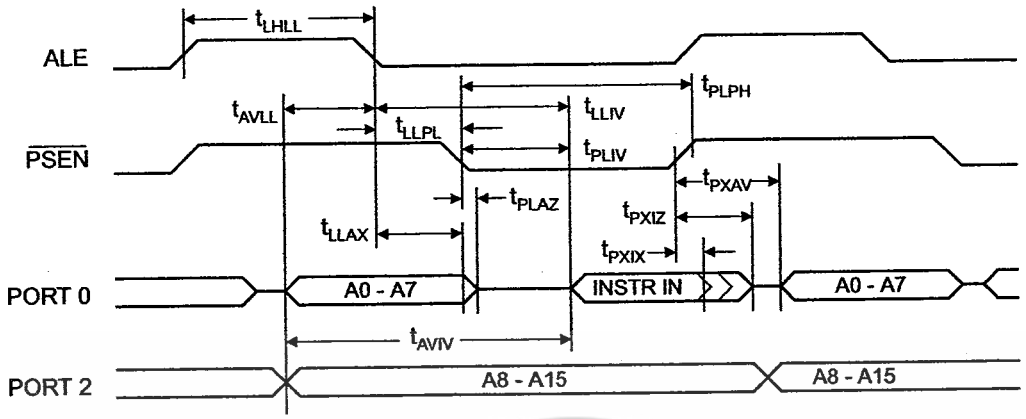
AC Characteristics

Under operating conditions, load capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; load capacitance for all other outputs = 80 pF.

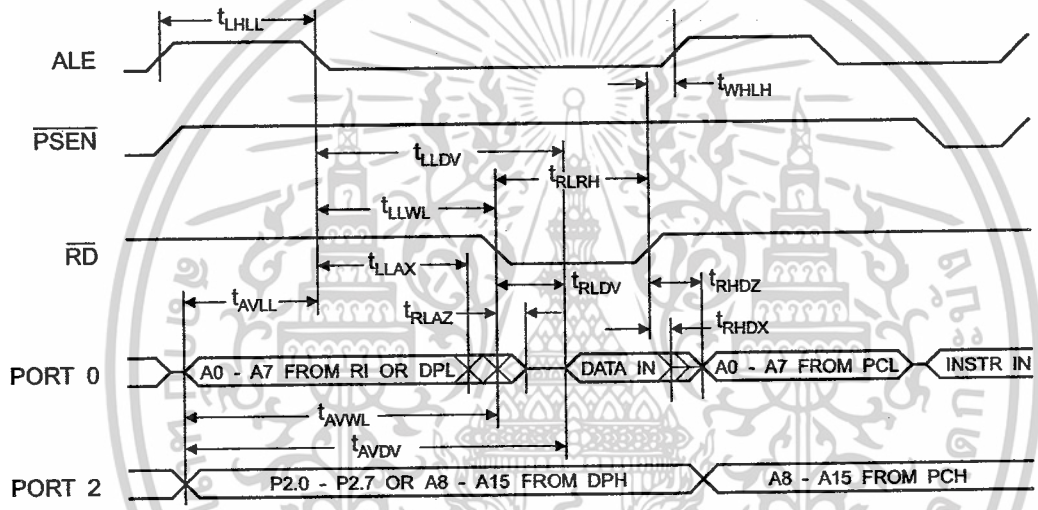
External Program and Data Memory Characteristics

Symbol	Parameter	12 MHz Oscillator		Variable Oscillator		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency			0	33	MHz
t_{LHLL}	ALE Pulse Width	127		$2t_{CLCL}$-40		ns
t_{AVLL}	Address Valid to ALE Low	43		t_{CLCL}-25		ns
t_{LLAX}	Address Hold After ALE Low	48		t_{CLCL}-25		ns
t_{LLIV}	ALE Low to Valid Instruction In		233		$4t_{CLCL}$-65	ns
t_{LLPL}	ALE Low to $\overline{PSEN}$ Low	43		t_{CLCL}-25		ns
t_{PLPH}	$\overline{PSEN}$ Pulse Width	205		$3t_{CLCL}$-45		ns
t_{PLIV}	$\overline{PSEN}$ Low to Valid Instruction In		145		$3t_{CLCL}$-60	ns
t_{PXIX}	Input Instruction Hold After $\overline{PSEN}$	0		0		ns
t_{PXIZ}	Input Instruction Float After $\overline{PSEN}$		59		t_{CLCL}-25	ns
t_{PXAV}	$\overline{PSEN}$ to Address Valid	75		t_{CLCL}-8		ns
t_{AVIV}	Address to Valid Instruction In		312		$5t_{CLCL}$-80	ns
t_{PLAZ}	$\overline{PSEN}$ Low to Address Float		10		10	ns
t_{RLRH}	$\overline{RD}$ Pulse Width	400		$6t_{CLCL}$-100		ns
t_{WLWH}	$\overline{WR}$ Pulse Width	400		$6t_{CLCL}$-100		ns
t_{RLDV}	$\overline{RD}$ Low to Valid Data In		252		$5t_{CLCL}$-90	ns
t_{RHDX}	Data Hold After $\overline{RD}$	0		0		ns
t_{RHDZ}	Data Float After $\overline{RD}$		97		$2t_{CLCL}$-28	ns
t_{LLDV}	ALE Low to Valid Data In		517		$8t_{CLCL}$-150	ns
t_{AVDV}	Address to Valid Data In		585		$9t_{CLCL}$-165	ns
t_{LLWL}	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	$3t_{CLCL}$-50	$3t_{CLCL}$+50	ns
t_{AVWL}	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		$4t_{CLCL}$-75		ns
t_{QVWX}	Data Valid to $\overline{WR}$ Transition	23		t_{CLCL}-30		ns
t_{QVWH}	Data Valid to $\overline{WR}$ High	433		$7t_{CLCL}$-130		ns
t_{WHQX}	Data Hold After $\overline{WR}$	33		t_{CLCL}-25		ns
t_{RLAZ}	$\overline{RD}$ Low to Address Float		0		0	ns
t_{WHLH}	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	t_{CLCL}-25	t_{CLCL}+25	ns

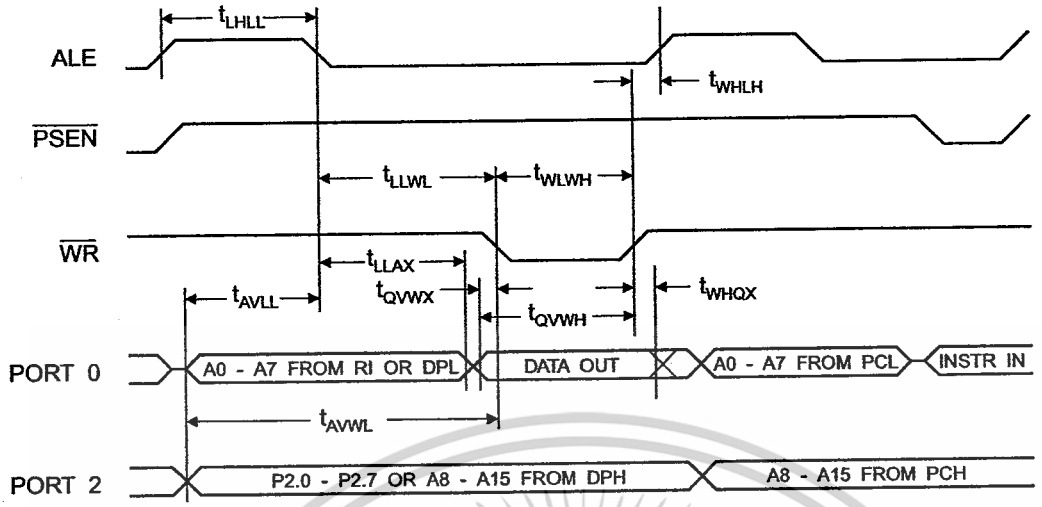
External Program Memory Read Cycle



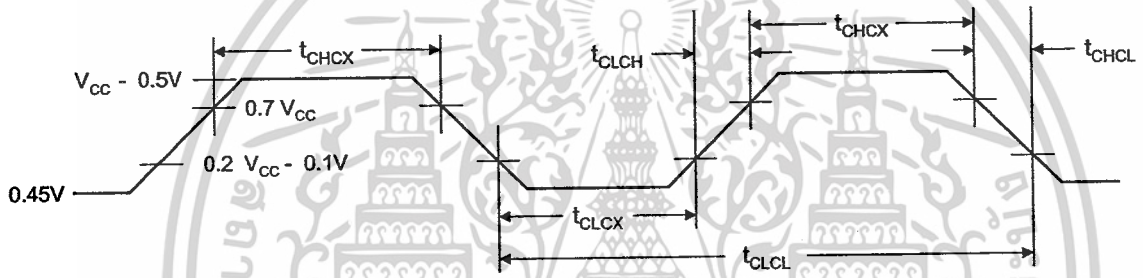
External Data Memory Read Cycle



External Data Memory Write Cycle



External Clock Drive Waveforms



External Clock Drive

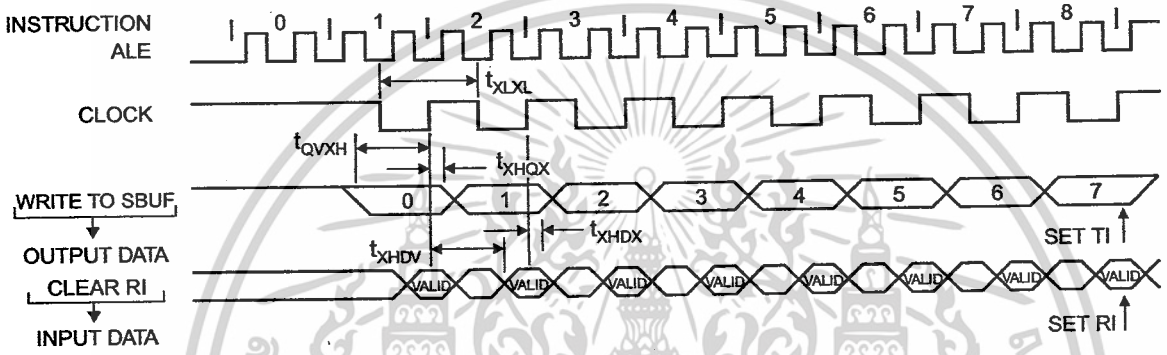
Symbol	Parameter	Min	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0	33	MHz
t_{CLCL}	Clock Period	30		ns
t_{CHCX}	High Time	12		ns
t_{CLCX}	Low Time	12		ns
t_{CLCH}	Rise Time		5	ns
t_{CHCL}	Fall Time		5	ns

Serial Port Timing: Shift Register Mode Test Conditions

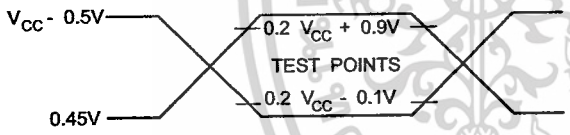
The values in this table are valid for V_{CC} = 4.0V to 5.5V and Load Capacitance = 80 pF.

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold After Clock Rising Edge	50		$2t_{CLCL}$-80		ns
t_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}$-133	ns

Shift Register Mode Timing Waveforms

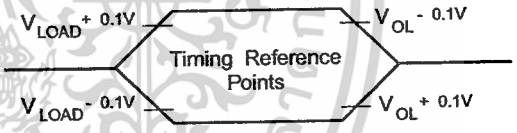


AC Testing Input/Output Waveforms(1)



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5V for a logic 1 and 0.45V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms(1)



Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs.

Ordering Information

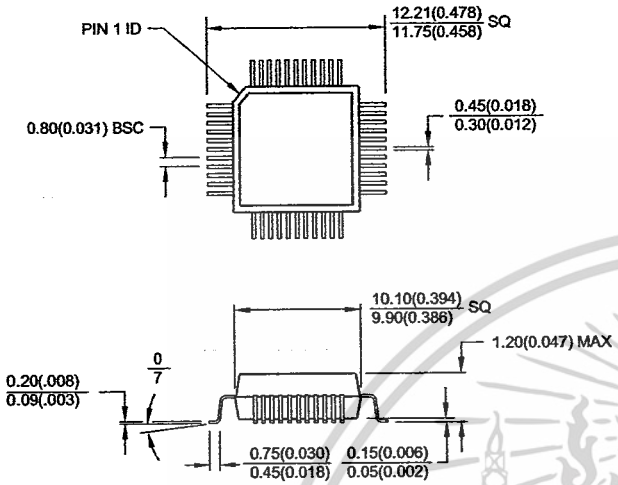
Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
24	4.0V to 5.5V	AT89S52-24AC AT89S52-24JC AT89S52-24PC	44A 44J 40P6	Commercial (0° C to 70° C)
		AT89S52-24AI AT89S52-24JI AT89S52-24PI	44A 44J 40P6	Industrial (-40° C to 85° C)
33	4.5V to 5.5V	AT89S52-33AC AT89S52-33JC AT89S52-33PC	44A 44J 40P6	Commercial (0° C to 70° C)

= Preliminary Availability

Package Type	
44A	44-lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
44J	44-lead, Plastic J-leaded Chip Carrier (PLCC)
40P6	40-pin, 0.600" Wide, Plastic Dual Inline Package (PDIP)

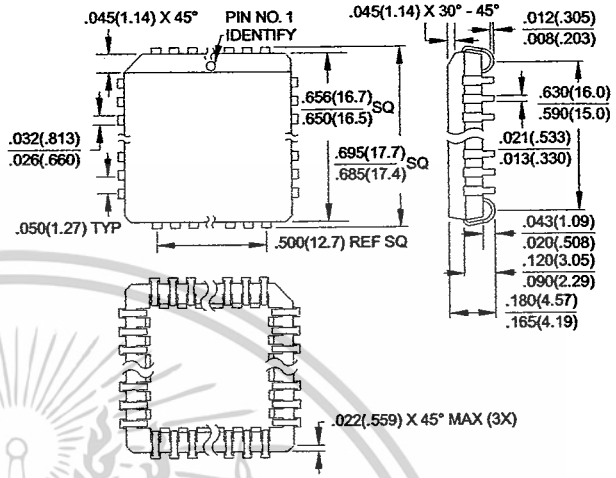
Packaging Information

44A, 44-lead, Thin (1.0 mm) Plastic Gull Wing Quad Flat Package (TQFP)
Dimensions in Millimeters and (Inches)*

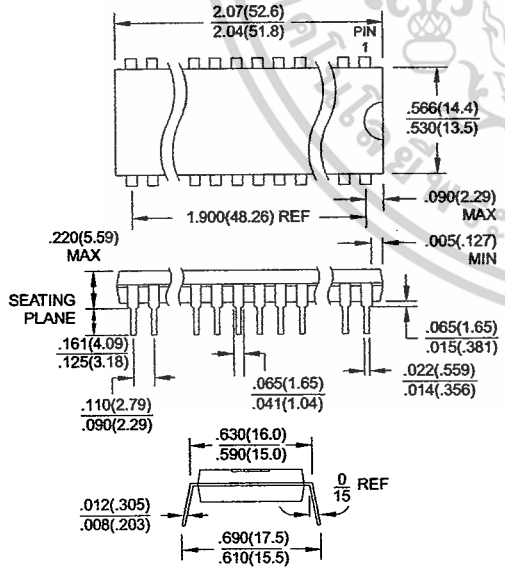


*Controlling dimension: millimeters

44J, 44-lead, Plastic J-leaded Chip Carrier (PLCC)
Dimensions in Inches and (Millimeters)



40P6, 40-pin, 0.600" Wide, Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-011 AC



Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Product Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Atmel Heilbronn
Theresienstrasse 2
POB 3535
D-74025 Heilbronn, Germany
TEL (49) 71 31 67 25 94
FAX (49) 71 31 67 24 23

Atmel Nantes
La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 0 2 40 18 18 18
FAX (33) 0 2 40 18 19 60

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-357-000
FAX (44) 1355-242-743

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

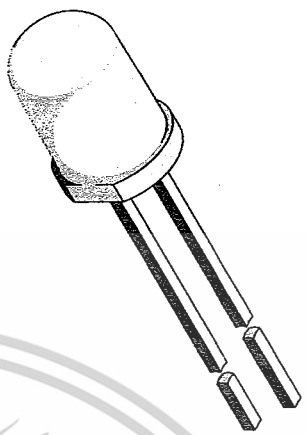
BBS
1-(408) 436-4309

Rev.1919A-07/01/xM

GaAs/GaAlAs IR Emitting Diode in ø 5 mm (T–1¾) Package

Description

TSAL7400 is a high efficiency infrared emitting diode in GaAlAs on GaAs technology, molded in clear plastic packages.
In comparison with the standard GaAs on GaAs technology these emitters achieve more than 100 % radiant power improvement at a similar wavelength.
The forward voltages at low current and at high pulse current roughly correspond to the low values of the standard technology. Therefore these emitters are ideally suitable as high performance replacements of standard emitters.



Features

- Extra high radiant power and radiant intensity
- High reliability
- Low forward voltage
- Suitable for high pulse current operation
- Standard T–1¾ (ø 5 mm) package
- Angle of half intensity $\varphi = \pm 25°$
- Peak wavelength $\lambda_p = 940$ nm
- Good spectral matching to Si photodetectors

Applications

Infrared remote control units with high power requirements
Free air transmission systems
Infrared source for optical counters and card readers
IR source for smoke detectors

Absolute Maximum Ratings

$T_{amb} = 25°C$

Parameter	Test Conditions	Symbol	Value	Unit
Reverse Voltage		V_R	5	V
Forward Current		I_F	100	mA
Peak Forward Current	$t_p/T = 0.5, t_p = 100\ \mu s$	I_{FM}	200	mA
Surge Forward Current	$t_p = 100\ \mu s$	I_{FSM}	1.5	A
Power Dissipation		P_V	210	mW
Junction Temperature		T_j	100	°C
Operating Temperature Range		T_{amb}	–55...+100	°C
Storage Temperature Range		T_{stg}	–55...+100	°C
Soldering Temperature	t ≦ 5sec, 2 mm from case	T_{sd}	260	°C
Thermal Resistance Junction/Ambient		R_{thJA}	350	K/W

Basic Characteristics

T_{amb} = 25°C

Parameter	Test Conditions	Symbol	Min	Typ	Max	Unit
Forward Voltage	I_F = 100 mA, t_p = 20 ms	V_F		1.35	1.6	V
	I_F = 1 A, t_p = 100 µs	V_F		2.6	3	V
Temp. Coefficient of V_F	I_F = 100mA	TK_{VF}		–1.3		mV/K
Reverse Current	V_R = 5 V	I_R			10	µA
Junction Capacitance	V_R = 0 V, f = 1 MHz, E = 0	C_j		25		pF
Radiant Intensity	I_F = 100 mA, t_p = 20 ms	I_e	25	40		mW/sr
	I_F = 1.0 A, t_p = 100 µs	I_e	220	310		mW/sr
Radiant Power	I_F = 100 mA, t_p = 20 ms	ϕ_e		35		mW
Temp. Coefficient of ϕ_e	I_F = 20 mA	$TK_{\phi e}$		–0.6		%/K
Angle of Half Intensity		φ		±25		deg
Peak Wavelength	I_F = 100 mA	λ_p		940		nm
Spectral Bandwidth	I_F = 100 mA	$\Delta\lambda$		50		nm
Temp. Coefficient of λ_p	I_F = 100 mA	$TK_{\lambda p}$		0.2		nm/K
Rise Time	I_F = 100 mA	t_r		800		ns
Fall Time	I_F = 100 mA	t_f		800		ns
Virtual Source Diameter	method: 63% encircled energy	ø		2.8		mm

Typical Characteristics (T_{amb} = 25°C unless otherwise specified)

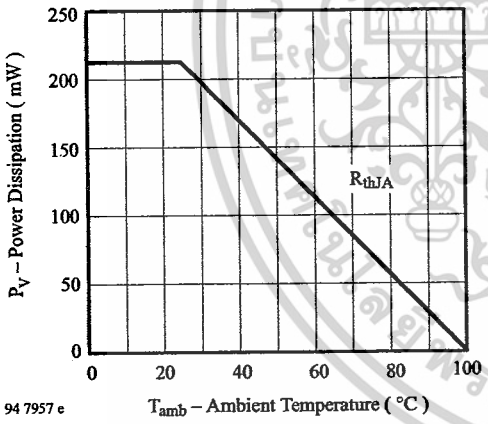


Figure 1. Power Dissipation vs. Ambient Temperature

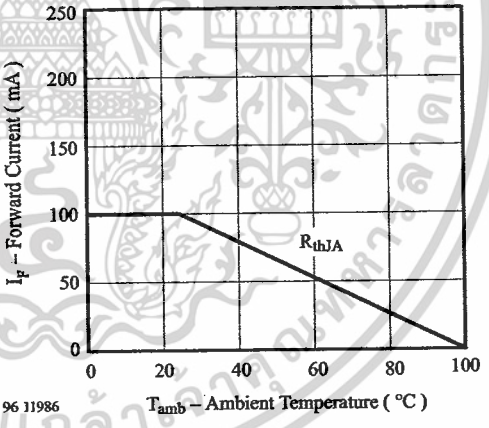


Figure 2. Forward Current vs. Ambient Temperature

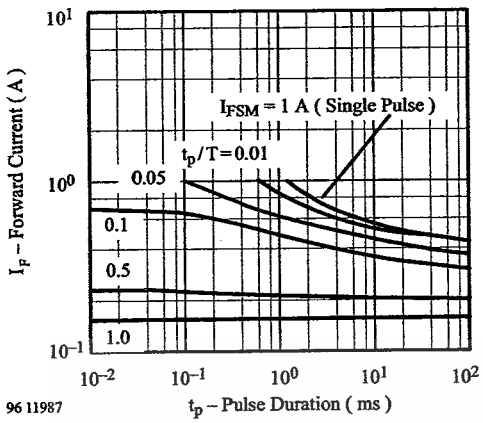


Figure 3. Pulse Forward Current vs. Pulse Duration

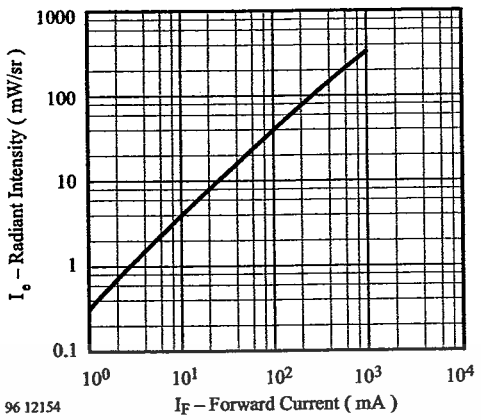


Figure 6. Radiant Intensity vs. Forward Current

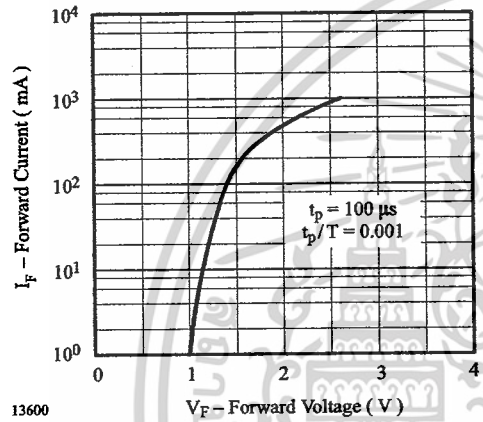


Figure 4. Forward Current vs. Forward Voltage

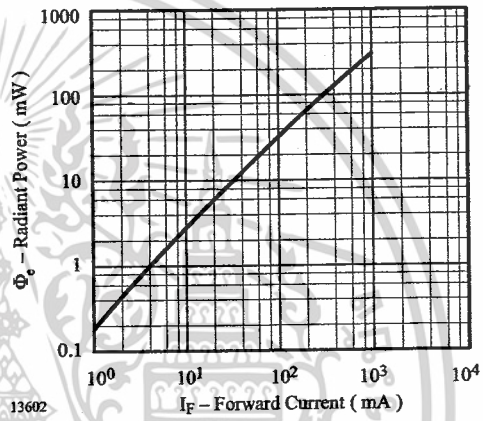


Figure 7. Radiant Power vs. Forward Current

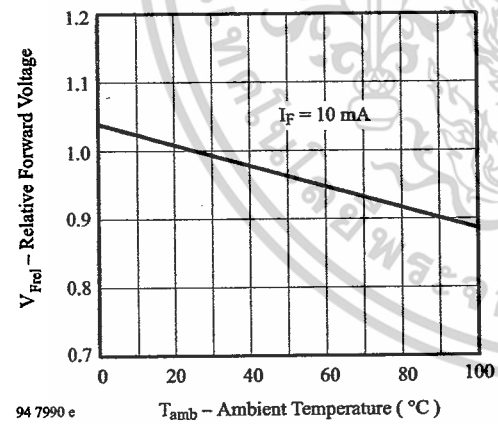


Figure 5. Relative Forward Voltage vs. Ambient Temperature

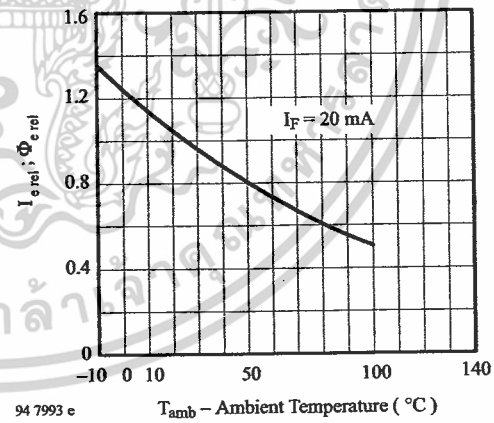


Figure 8. Rel. Radiant Intensity\Power vs. Ambient Temperature

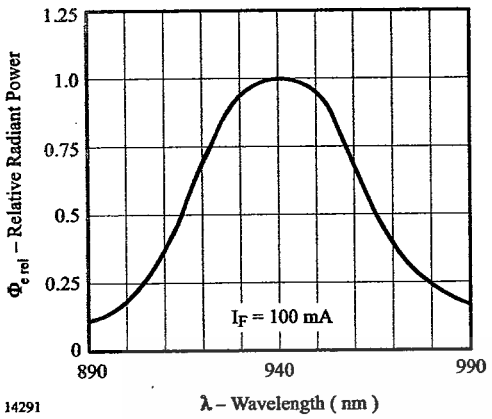


Figure 9. Relative Radiant Power vs. Wavelength

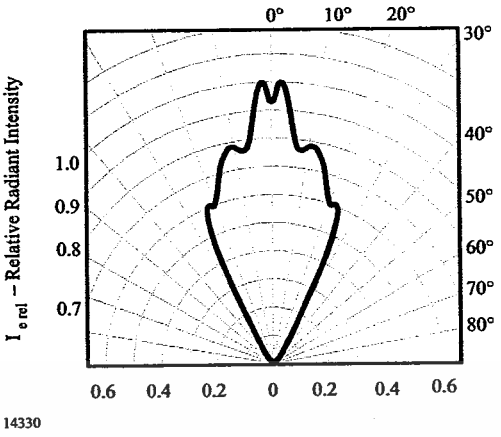
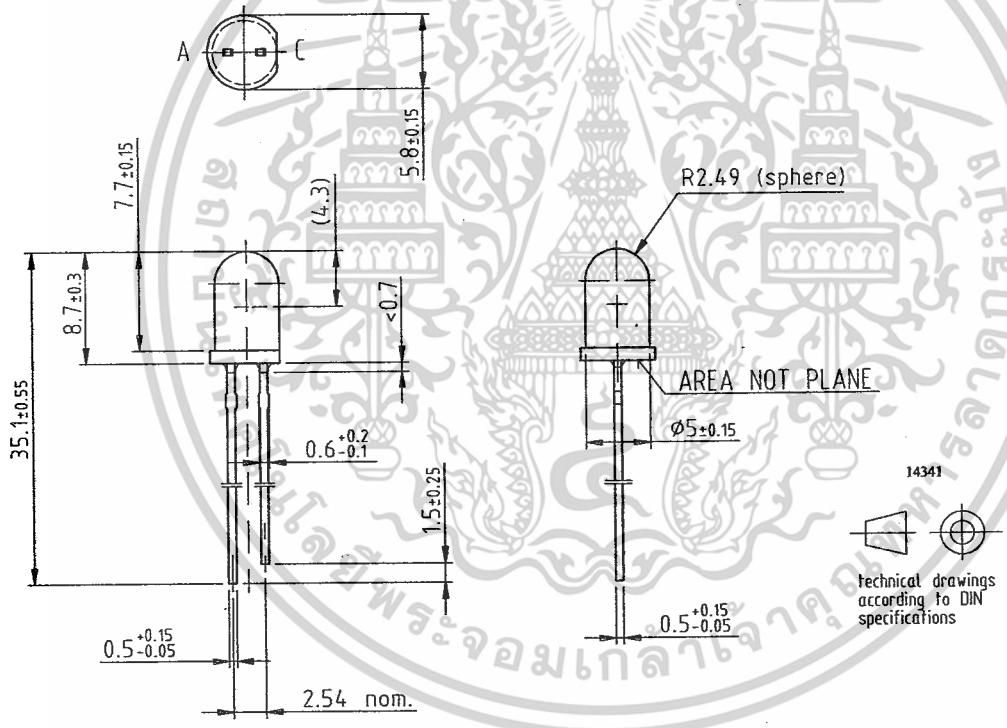


Figure 10. Relative Radiant Intensity vs. Angular Displacement

Dimensions in mm



Ozone Depleting Substances Policy Statement

It is the policy of **Vishay Semiconductor GmbH** to

1. Meet all present and future national and international statutory requirements.

2. Regularly and continuously improve the performance of our products, processes, distribution and operating systems with respect to their impact on the health and safety of our employees and the public, as well as their impact on the environment.

It is particular concern to control or eliminate releases of those substances into the atmosphere which are known as ozone depleting substances (ODSs).

The Montreal Protocol (1987) and its London Amendments (1990) intend to severely restrict the use of ODSs and forbid their use within the next ten years. Various national and international initiatives are pressing for an earlier ban on these substances.

Vishay Semiconductor GmbH has been able to use its policy of continuous improvements to eliminate the use of ODSs listed in the following documents.

1. Annex A, B and list of transitional substances of the Montreal Protocol and the London Amendments respectively

2. Class I and II ozone depleting substances in the Clean Air Act Amendments of 1990 by the Environmental Protection Agency (EPA) in the USA

3. Council Decision 88/540/EEC and 91/690/EEC Annex A, B and C (transitional substances) respectively.

Vishay Semiconductor GmbH can certify that our semiconductors are not manufactured with ozone depleting substances and do not contain such substances.

Vishay Semiconductor GmbH, P.O.B. 3535, D-74025 Heilbronn, Germany
Telephone: 49 (0)7131 67 2831, Fax number: 49 (0)7131 67 2423

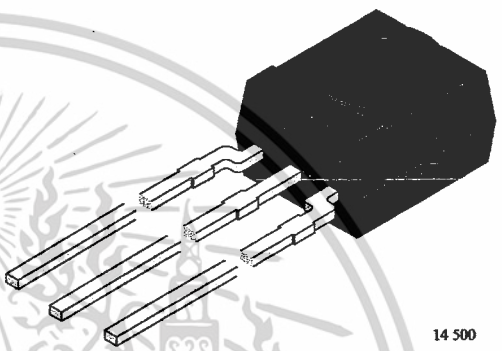
Photo Modules for PCM Remote Control Systems

Available types for different carrier frequencies

Type	fo	Type	fo
TSOP4830	30 kHz	TSOP4833	33 kHz
TSOP4836	36 kHz	TSOP4837	36.7 kHz
TSOP4838	38 kHz	TSOP4840	40 kHz
TSOP4856	56 kHz		

Description

The TSOP48.. – series are miniaturized receivers for infrared remote control systems. PIN diode and preamplifier are assembled on lead frame, the epoxy package is designed as IR filter.
The demodulated output signal can directly be decoded by a microprocessor. TSOP48.. is the standard IR remote control receiver series, supporting all major transmission codes.

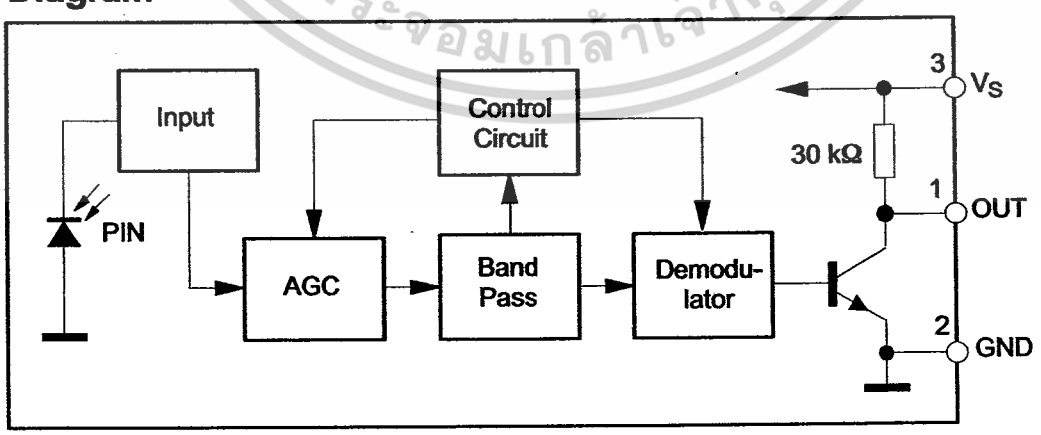


14 500

Features

- Photo detector and preamplifier in one package
- Internal filter for PCM frequency
- Improved shielding against electrical field disturbance
- TTL and CMOS compatibility
- Output active low
- Low power consumption
- High immunity against ambient light
- Continuous data transmission possible (800 bit/s)
- Suitable burst length ≥10 cycles/burst

Block Diagram



9612226

Absolute Maximum Ratings

T_{amb} = 25°C

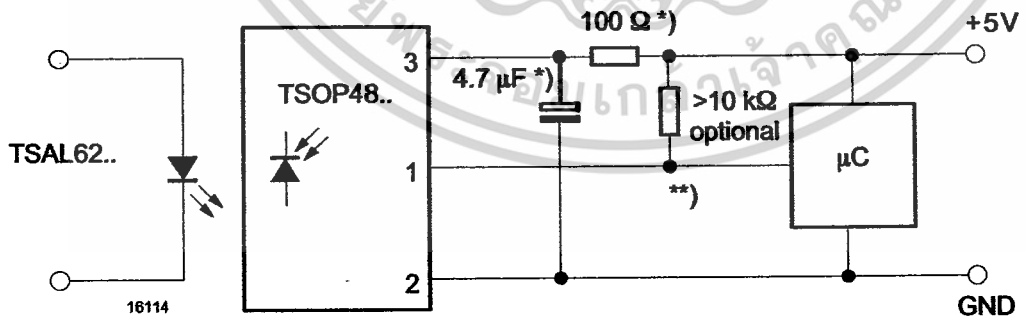
Parameter	Test Conditions	Symbol	Value	Unit
Supply Voltage	(Pin 3)	V_S	–0.3...6.0	V
Supply Current	(Pin 3)	I_S	5	mA
Output Voltage	(Pin 1)	V_O	–0.3...6.0	V
Output Current	(Pin 1)	I_O	5	mA
Junction Temperature		T_j	100	°C
Storage Temperature Range		T_{stg}	–25...+85	°C
Operating Temperature Range		T_{amb}	–25...+85	°C
Power Consumption	(T_{amb} ≦ 85 °C)	P_{tot}	50	mW
Soldering Temperature	t ≦ 10 s, 1 mm from case	T_{sd}	260	°C

Basic Characteristics

T_{amb} = 25°C

Parameter	Test Conditions	Symbol	Min	Typ	Max	Unit
Supply Current (Pin 3)	V_S = 5 V, E_v = 0	I_{SD}	0.8	1.1	1.5	mA
	V_S = 5 V, E_v = 40 klx, sunlight	I_{SH}		1.4		mA
Supply Voltage (Pin 3)		V_S	4.5		5.5	V
Transmission Distance	E_v = 0, test signal see fig.7, IR diode TSAL6200, I_F = 250 mA	d		35		m
Output Voltage Low (Pin 1)	I_{OSL} = 0.5 mA, E_e = 0.7 mW/m²	V_{OSL}			250	mV
Irradiance (30 – 40 kHz)	Pulse width tolerance: $t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$, test signal see fig.7	$E_{e\,min}$		0.2	0.4	mW/m²
Irradiance (56 kHz)	Pulse width tolerance: $t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$, test signal see fig.7	$E_{e\,min}$		0.3	0.6	mW/m²
Irradiance	$t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$	$E_{e\,max}$	30			W/m²
Directivity	Angle of half transmission distance	$\varphi_{1/2}$		±45		deg

Application Circuit



*) recommended to suppress power supply disturbances
**) The output voltage should not be hold continuously at a voltage below 3.3V by the external circuit.

Suitable Data Format

The circuit of the TSOP48.. is designed in that way that unexpected output pulses due to noise or disturbance signals are avoided. A bandpassfilter, an integrator stage and an automatic gain control are used to suppress such disturbances.
The distinguishing mark between data signal and disturbance signal are carrier frequency, burst length and duty cycle.

The data signal should fullfill the following condition:

- Carrier frequency should be close to center frequency of the bandpass (e.g. 38kHz).
- Burst length should be 10 cycles/burst or longer.
- After each burst which is between 10 cycles and 70 cycles a gap time of at least 14 cycles is necessary.
- For each burst which is longer than 1.8ms a corresponding gap time is necessary at some time in the data stream. This gap time should be at least 4 times longer than the burst.
- Up to 800 short bursts per second can be received continuously.

Some examples for suitable data format are:
NEC Code, Toshiba Micom Format, Sharp Code, RC5 Code, RC6 Code, R–2000 Code.

When a disturbance signal is applied to the TSOP48.. it can still receive the data signal. However the sensitivity is reduced to that level that no unexpected pulses will occure.

Some examples for such disturbance signals which are suppressed by the TSOP48.. are:

- DC light (e.g. from tungsten bulb or sunlight)
- Continuous signal at 38kHz or at any other frequency
- Signals from fluorescent lamps with electronic ballast with high or low modulation (see Figure A or Figure B).

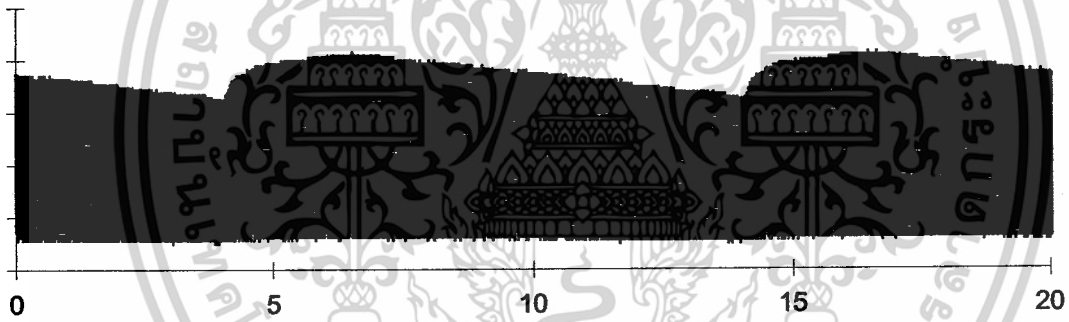


Figure A: IR Signal from Fluorescent Lamp with low Modulation

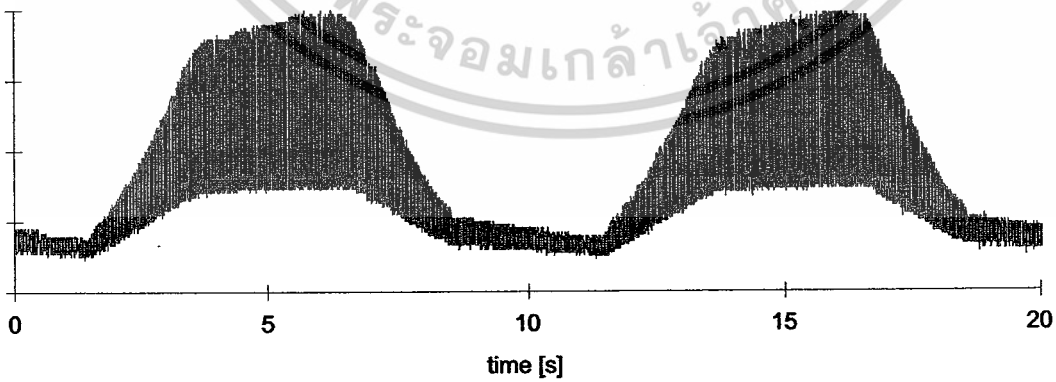


Figure B: IR Signal from Fluorescent Lamp with high Modulation

Typical Characteristics (T_{amb} = 25°C unless otherwise specified)

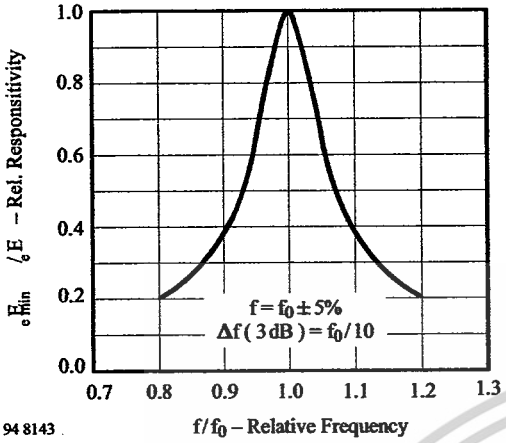


Figure 1. Frequency Dependence of Responsivity

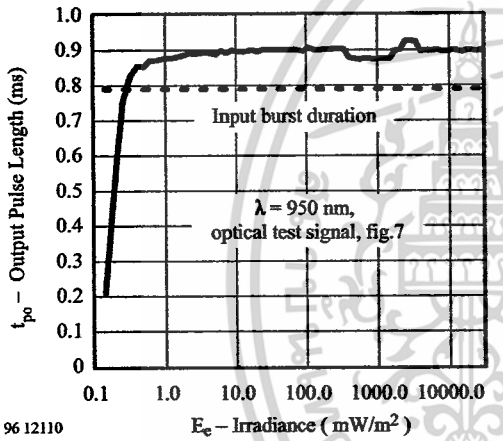


Figure 2. Sensitivity in Dark Ambient

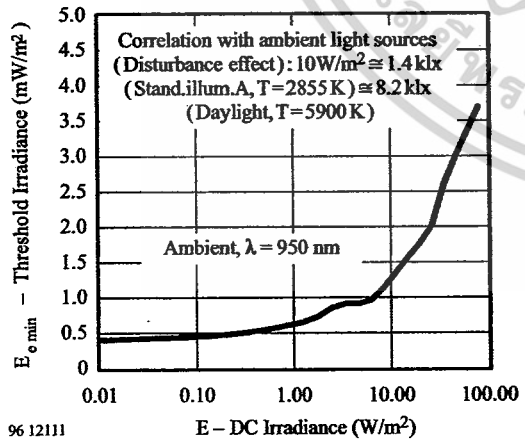


Figure 3. Sensitivity in Bright Ambient

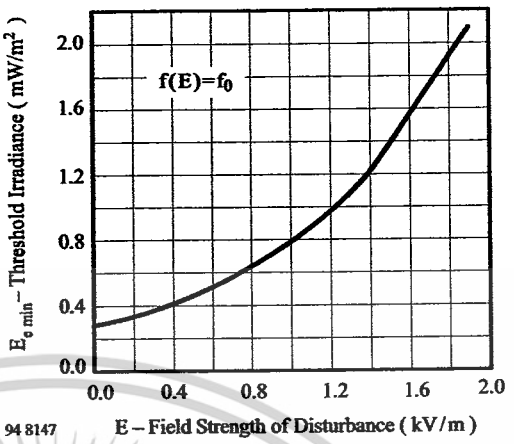


Figure 4. Sensitivity vs. Electric Field Disturbances

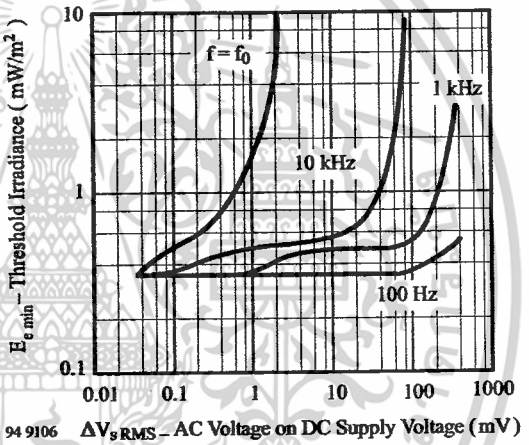


Figure 5. Sensitivity vs. Supply Voltage Disturbances

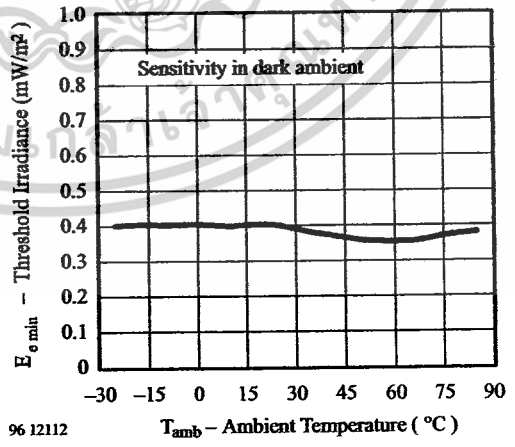


Figure 6. Sensitivity vs. Ambient Temperature

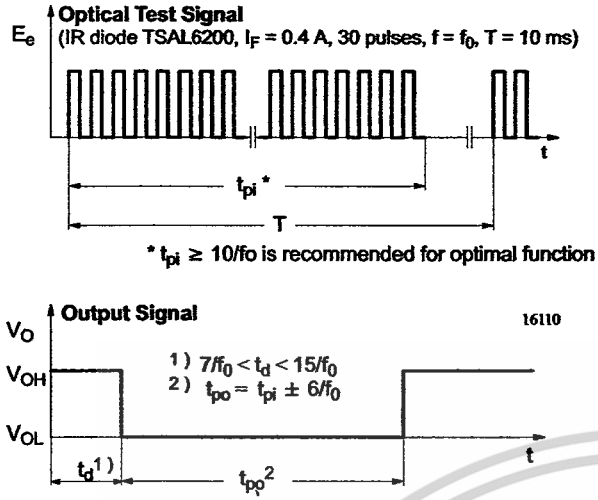


Figure 7.

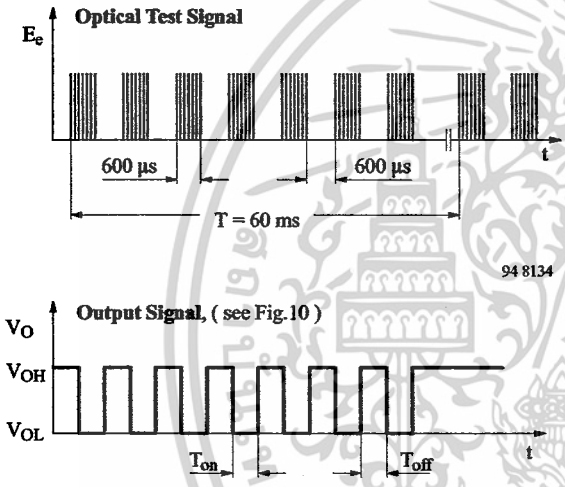


Figure 8. Output Function

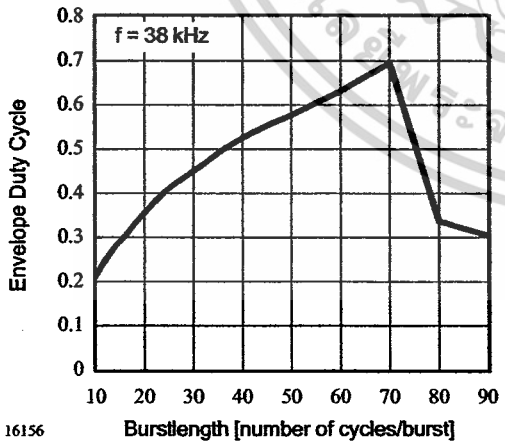


Figure 9. Max. Envelope Duty Cycle vs. Burstlength

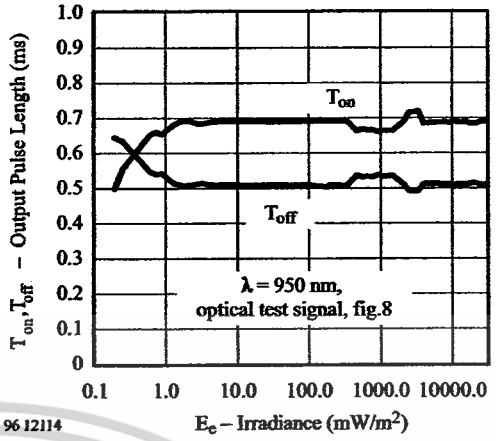


Figure 10. Output Pulse Diagram

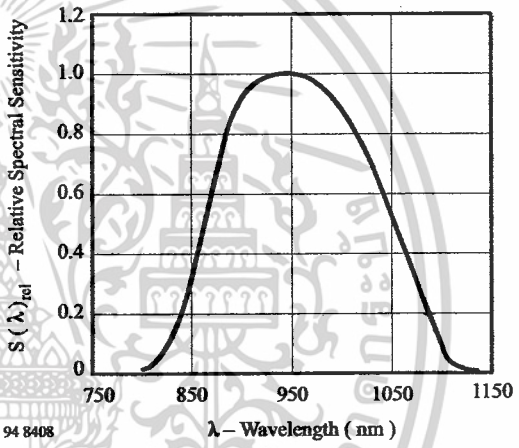


Figure 11. Relative Spectral Sensitivity vs. Wavelength

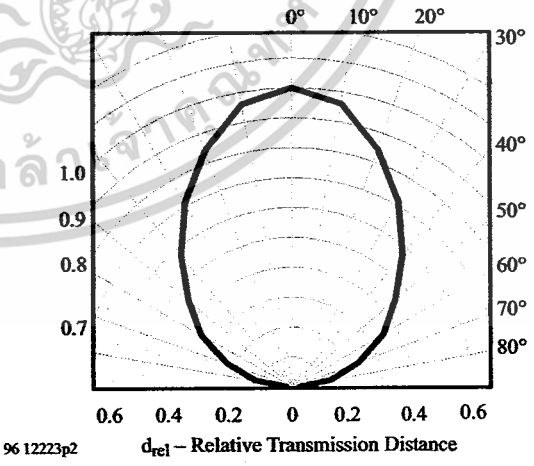
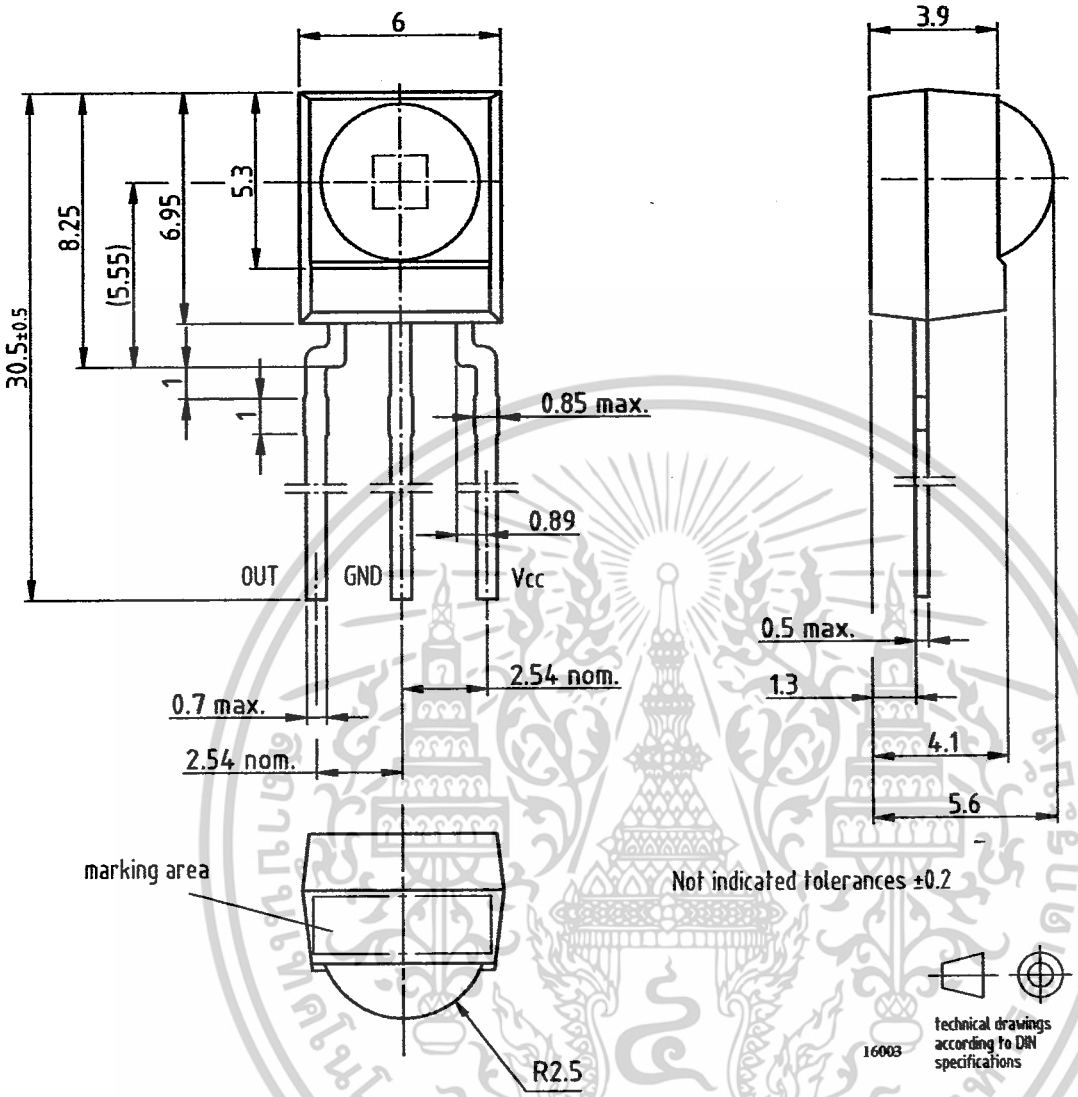


Figure 12. Directivity

Dimensions in mm



Ozone Depleting Substances Policy Statement

It is the policy of **Vishay Semiconductor GmbH** to

1. Meet all present and future national and international statutory requirements.

2. Regularly and continuously improve the performance of our products, processes, distribution and operating systems with respect to their impact on the health and safety of our employees and the public, as well as their impact on the environment.

It is particular concern to control or eliminate releases of those substances into the atmosphere which are known as ozone depleting substances (ODSs).

The Montreal Protocol (1987) and its London Amendments (1990) intend to severely restrict the use of ODSs and forbid their use within the next ten years. Various national and international initiatives are pressing for an earlier ban on these substances.

Vishay Semiconductor GmbH has been able to use its policy of continuous improvements to eliminate the use of ODSs listed in the following documents.

1. Annex A, B and list of transitional substances of the Montreal Protocol and the London Amendments respectively

2. Class I and II ozone depleting substances in the Clean Air Act Amendments of 1990 by the Environmental Protection Agency (EPA) in the USA

3. Council Decision 88/540/EEC and 91/690/EEC Annex A, B and C (transitional substances) respectively.

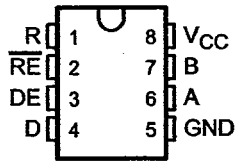
Vishay Semiconductor GmbH can certify that our semiconductors are not manufactured with ozone depleting substances and do not contain such substances.

Vishay Semiconductor GmbH, P.O.B. 3535, D-74025 Heilbronn, Germany
Telephone: 49 (0)7131 67 2831, Fax number: 49 (0)7131 67 2423

SLLS101B – JULY 1985 – REVISED JUNE 1999

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

D OR P PACKAGE
(TOP VIEW)



description

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

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

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

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

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

Function Tables

DRIVER

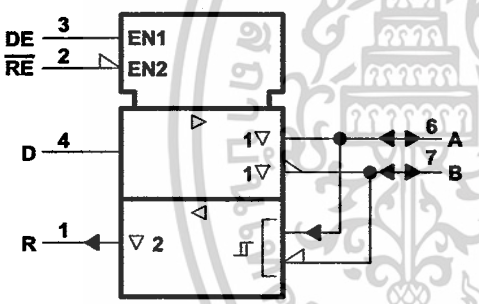
INPUT D	ENABLE DE	OUTPUTS A	OUTPUTS B
H	H	H	L
L	H	L	H
X	L	Z	Z

RECEIVER

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

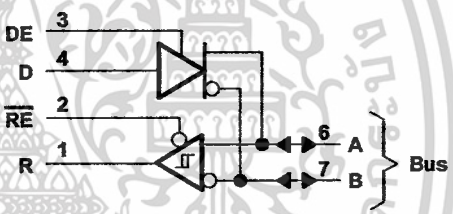
H = high level, L = low level, ? = indeterminate,
X = irrelevant, Z = high impedance (off)

logic symbol†

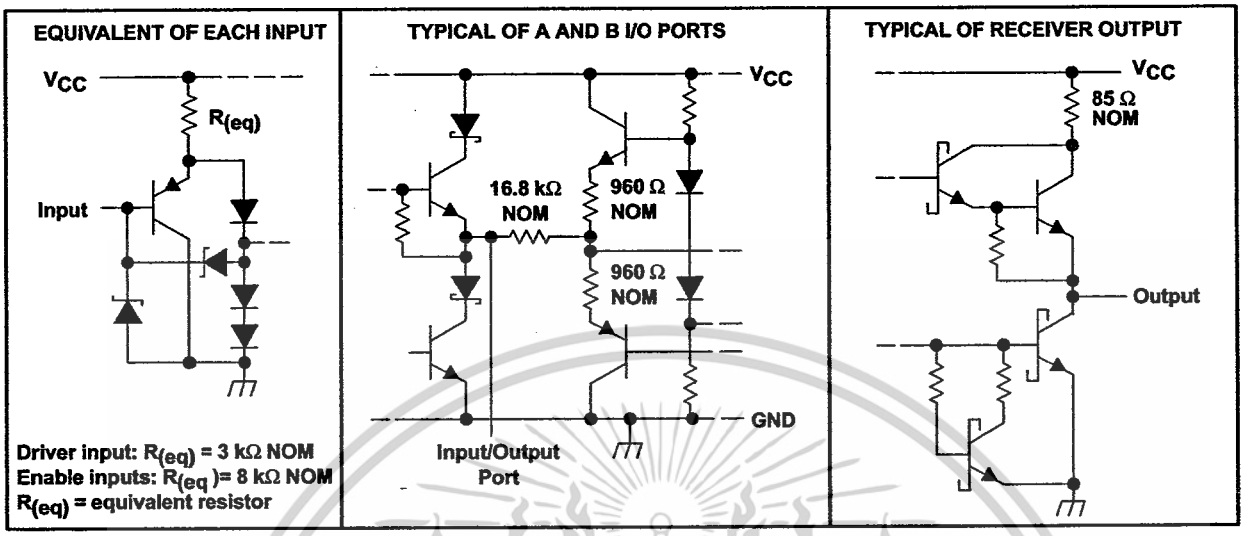


† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

logic diagram (positive logic)



schematics of inputs and outputs



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

Supply voltage, V_{CC} (see Note 1) ..... 7 V
Voltage range at any bus terminal ..... −10 V to 15 V
Enable input voltage, V_I ..... 5.5 V
Package thermal impedance, θ_{JA} (see Note 2): D package ..... 197°C/W
P package ..... 104°C/W
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds ..... 260°C
Storage temperature range, T_{stg} ..... −65°C to 150°C

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

NOTES: 1. All voltage values, except differential input/output bus voltage, are with respect to network ground terminal.
2. The package thermal impedance is calculated in accordance with JESD 51, except for through-hole packages, which use a trace length of zero.

recommended operating conditions

		MIN	TYP	MAX	UNIT
Supply voltage, V_{CC}		4.75	5	5.25	V
Voltage at any bus terminal (separately or common mode), V_I or V_{IC}				12	V
				−7	
High-level input voltage, V_{IH}	D, DE, and $\overline{RE}$	2			V
Low-level input voltage, V_{IL}	D, DE, and $\overline{RE}$			0.8	V
Differential input voltage, V_{ID} (see Note 3)				±12	V
High-level output current, I_{OH}	Driver			−60	mA
	Receiver			−400	µA
Low-level output current, I_{OL}	Driver			60	mA
	Receiver			8	
Operating free-air temperature, T_A	SN65176B	−40		105	°C
	SN75176B	0		70	

NOTE 3: Differential-input/output bus voltage is measured at the noninverting terminal A with respect to the inverting terminal B.

DRIVER SECTION

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

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

† The power-off measurement in ANSI Standard TIA/EIA-422-B applies to disabled outputs only and is not applied to combined inputs and outputs.
‡ All typical values are at V_{CC} = 5 V and T_A = 25°C.
§ $\Delta|V_{OD}|$ and $\Delta|V_{OC}|$ are the changes in magnitude of V_{OD} and V_{OC}, respectively, that occur when the input is changed from a high level to a low level.
¶ The minimum V_{OD2} with a 100-Ω load is either 1/2 V_{OD1} or 2 V, whichever is greater.
NOTES: 4. See ANSI Standard TIA/EIA-485-A, Figure 3.5, Test Termination Measurement 2.
5. This applies for both power on and off; refer to ANSI Standard TIA/EIA-485-A for exact conditions. The TIA/EIA-422-B limit does not apply for a combined driver and receiver terminal.

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

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{d(OD)}$	Differential-output delay time	R_L = 54 Ω, See Figure 3		15	22	ns
$t_{t(OD)}$	Differential-output transition time			20	30	ns
t_{PZH}	Output enable time to high level	See Figure 4		85	120	ns
t_{PZL}	Output enable time to low level	See Figure 5		40	60	ns
t_{PHZ}	Output disable time from high level	See Figure 4		150	250	ns
t_{PLZ}	Output disable time from low level	See Figure 5		20	30	ns

SYMBOL EQUIVALENTS

DATA-SHEET PARAMETER	TIA/EIA-422-B	TIA/EIA-485-A
V_O	V_{oa}, V_{ob}	V_{oa}, V_{ob}
$\lvert V_{OD1}\rvert$	V_o	V_o
$\lvert V_{OD2}\rvert$	V_t (R_L = 100 Ω)	V_t (R_L = 54 Ω)
$\lvert V_{OD3}\rvert$		V_t (Test Termination Measurement 2)
$\Delta\lvert V_{OD}\rvert$	$\lvert\, \lvert V_t\rvert - \lvert\overline{V}_t\rvert \,\rvert$	$\lvert\, \lvert V_t - \overline{V}_t\rvert \,\rvert$
V_{OC}	$\lvert V_{os}\rvert$	$\lvert V_{os}\rvert$
$\Delta\lvert V_{OC}\rvert$	$\lvert V_{os} - \overline{V}_{os}\rvert$	$\lvert V_{os} - \overline{V}_{os}\rvert$
I_{OS}	$\lvert I_{sa}\rvert$, $\lvert I_{sb}\rvert$	
I_O	$\lvert I_{xa}\rvert$, $\lvert I_{xb}\rvert$	I_{ia}, I_{ib}

RECEIVER SECTION

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

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{IT+}	Positive-going input threshold voltage	V_O = 2.7 V,	I_O = −0.4 mA			0.2	V
V_{IT-}	Negative-going input threshold voltage	V_O = 0.5 V,	I_O = 8 mA	−0.2‡			V
V_{hys}	Input hysteresis voltage ($V_{IT+} - V_{IT-}$)				50		mV
V_{IK}	Enable Input clamp voltage	I_I = −18 mA				−1.5	V
V_{OH}	High-level output voltage	V_{ID} = 200 mV, See Figure 2	I_{OH} = −400 µA,	2.7			V
V_{OL}	Low-level output voltage	V_{ID} = −200 mV, See Figure 2	I_{OL} = 8 mA,			0.45	V
I_{OZ}	High-impedance-state output current	V_O = 0.4 V to 2.4 V				±20	µA
I_I	Line input current	Other input = 0 V, See Note 6	V_I = 12 V			1	mA
			V_I = −7 V			−0.8	
I_{IH}	High-level enable input current	V_{IH} = 2.7 V				20	µA
I_{IL}	Low-level enable input current	V_{IL} = 0.4 V				−100	µA
r_I	Input resistance	V_I = 12 V		12			kΩ
I_{OS}	Short-circuit output current			−15		−85	mA
I_{CC}	Supply current (total package)	No load	Outputs enabled		42	55	mA
			Outputs disabled		26	35	

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the less positive (more negative) limit is designated minimum, is used in this data sheet for common-mode input voltage and threshold voltage levels only.
NOTE 6: This applies for both power on and power off. Refer to EIA Standard TIA/EIA-485-A for exact conditions.

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

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

PARAMETER MEASUREMENT INFORMATION

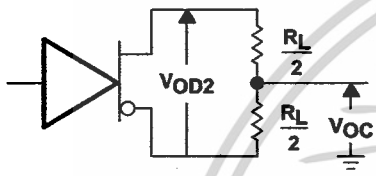


Figure 1. Driver V_{OD} and V_{OC}

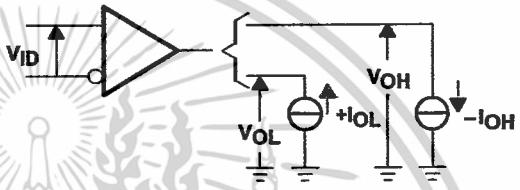
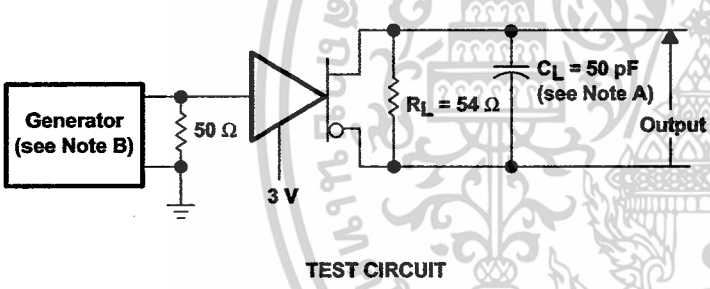
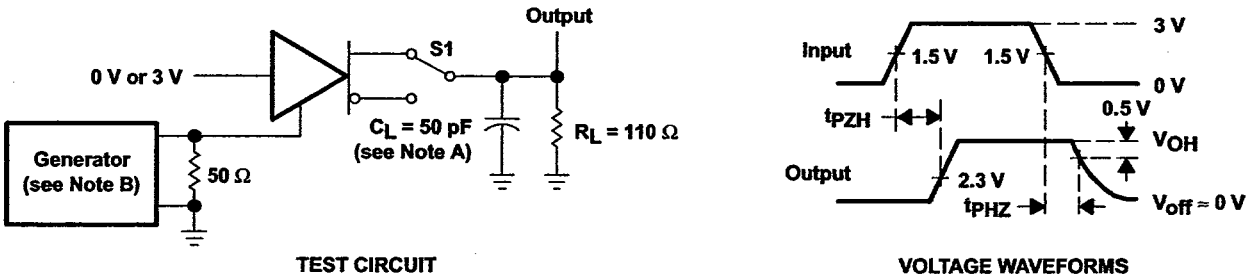


Figure 2. Receiver V_{OH} and V_{OL}



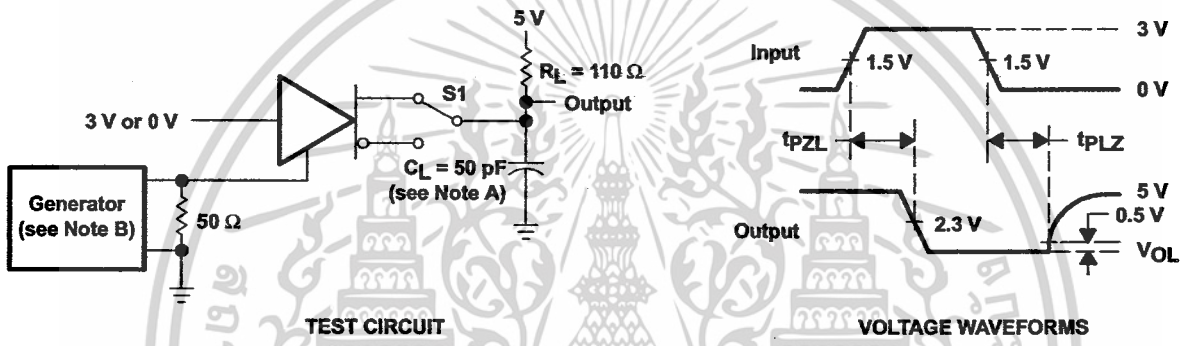
NOTES: A. C_L includes probe and jig capacitance.
B. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 1 MHz, 50% duty cycle, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_O = 50 Ω.

Figure 3. Driver Test Circuit and Voltage Waveforms



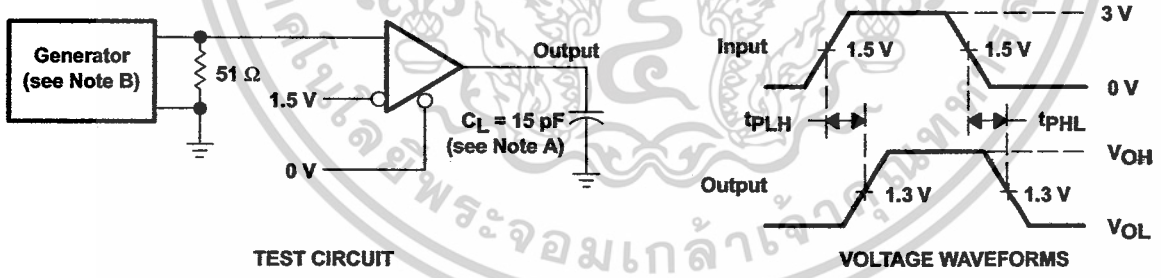
NOTES: A. C_L includes probe and jig capacitance.
B. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 1 MHz, 50% duty cycle, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_O = 50 Ω.

Figure 4. Driver Test Circuit and Voltage Waveforms



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

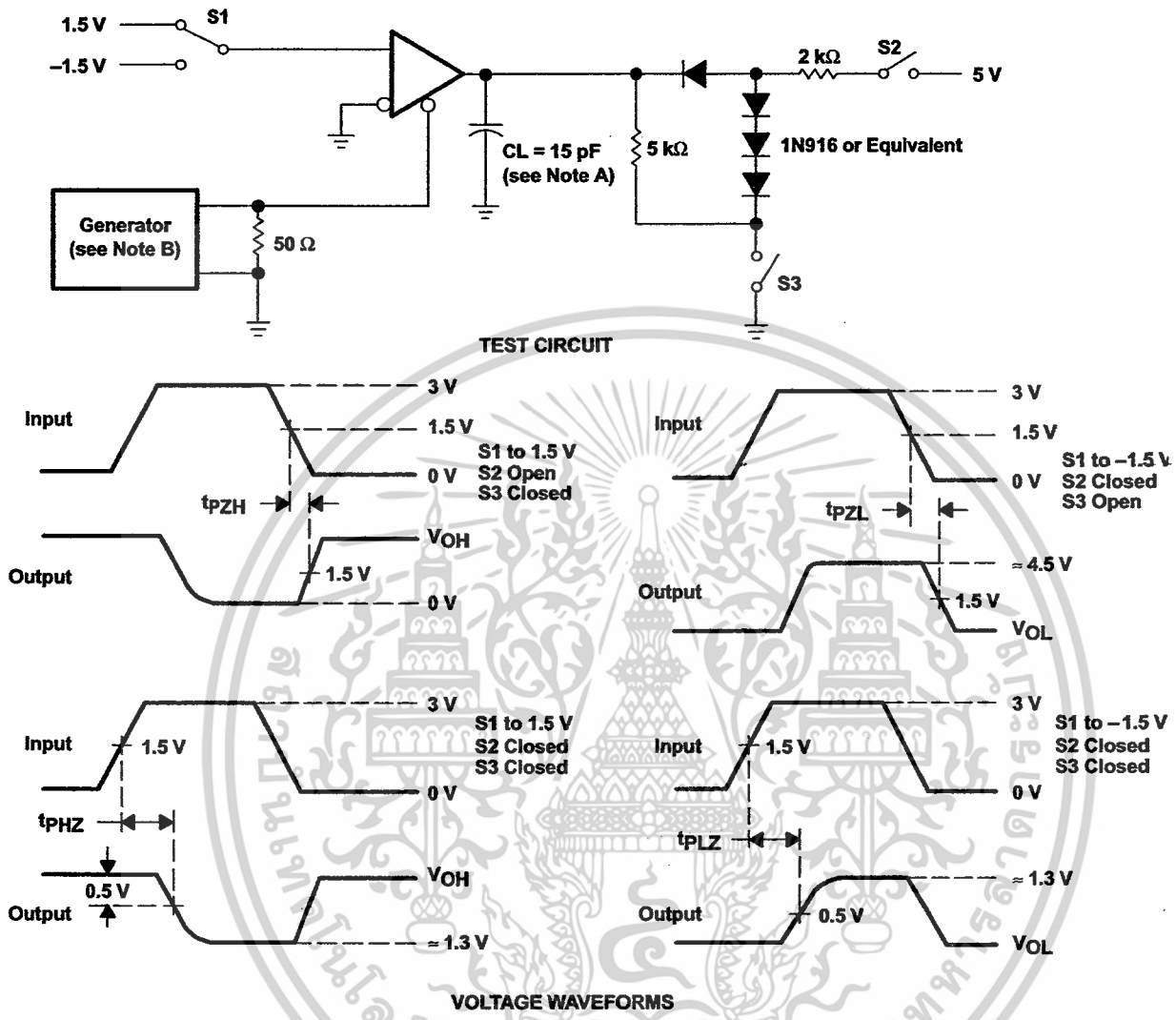
Figure 5. Driver Test Circuit and Voltage Waveforms



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

Figure 6. Receiver Test Circuit and Voltage Waveforms

PARAMETER MEASUREMENT INFORMATION



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

Figure 7. Receiver Test Circuit and Voltage Waveforms

TYPICAL CHARACTERISTICS

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

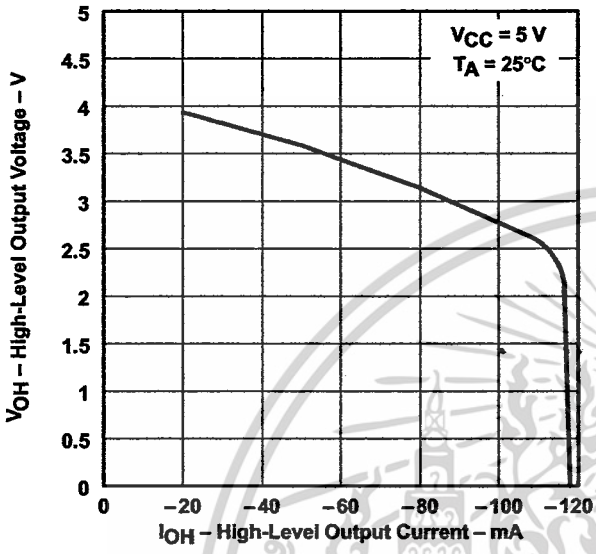


Figure 8

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

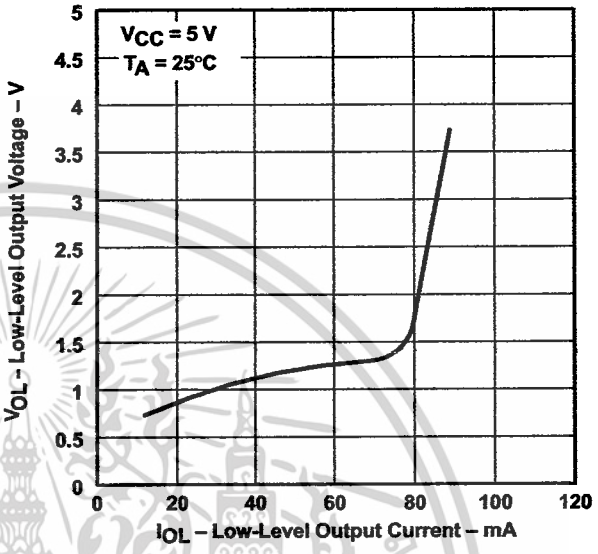


Figure 9

DRIVER
DIFFERENTIAL OUTPUT VOLTAGE
vs
OUTPUT CURRENT

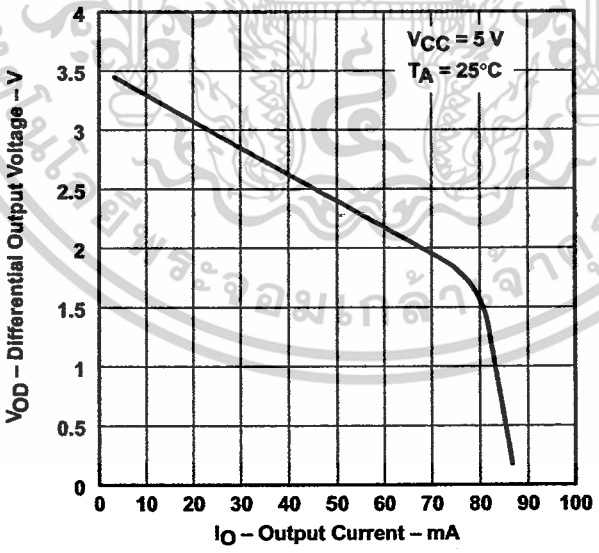


Figure 10

TYPICAL CHARACTERISTICS

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

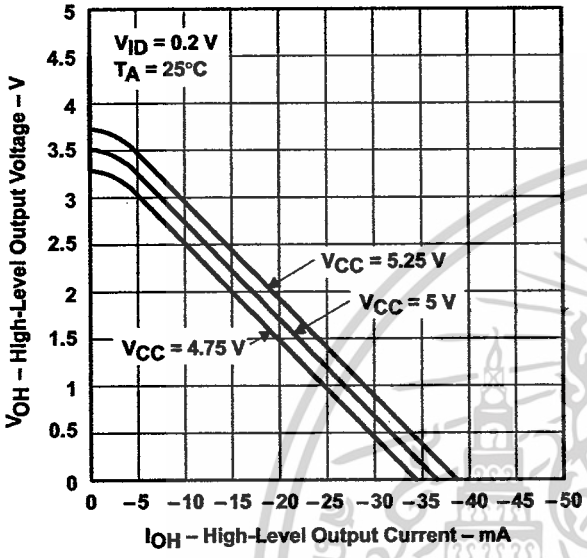
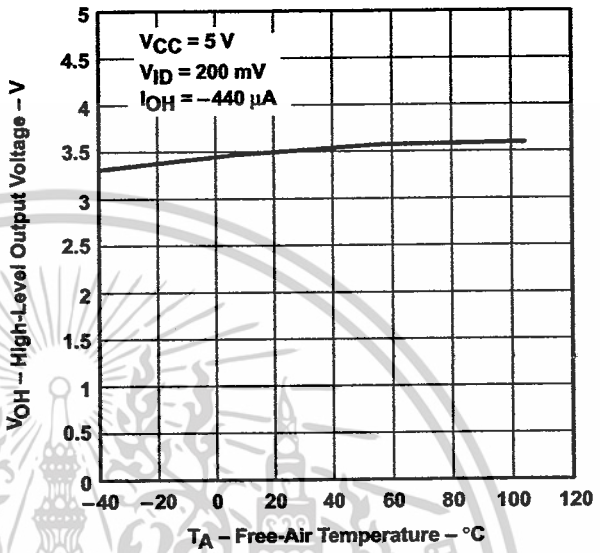


Figure 11

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



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

Figure 12

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

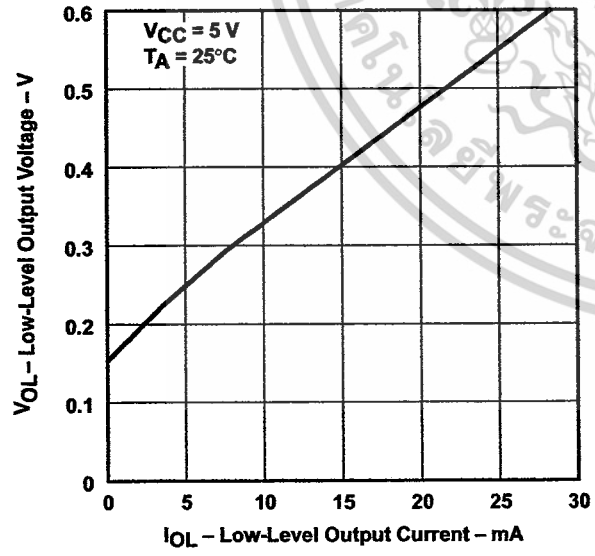


Figure 13

RECEIVER
LOW-LEVEL OUTPUT VOLTAGE
vs
FREE-AIR TEMPERATURE

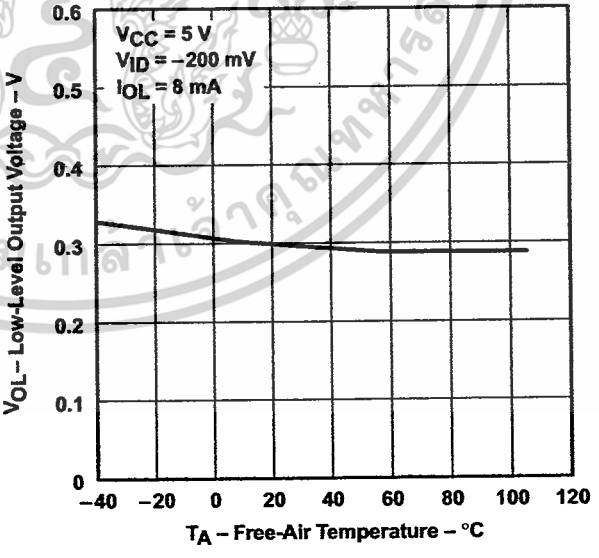


Figure 14

TYPICAL CHARACTERISTICS

RECEIVER
OUTPUT VOLTAGE
vs
ENABLE VOLTAGE

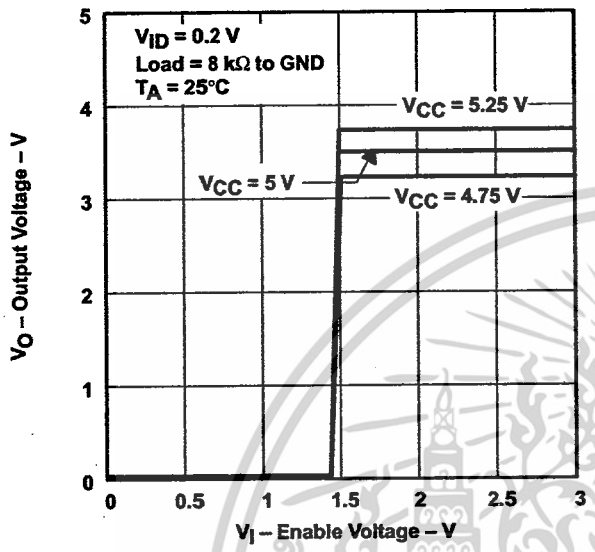


Figure 15

RECEIVER
OUTPUT VOLTAGE
vs
ENABLE VOLTAGE

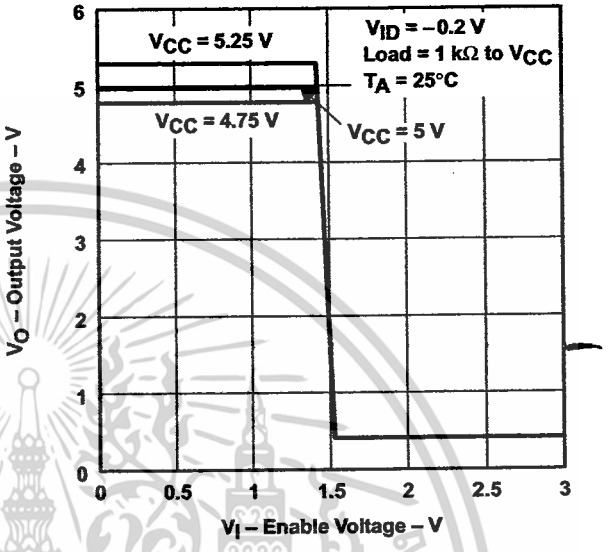
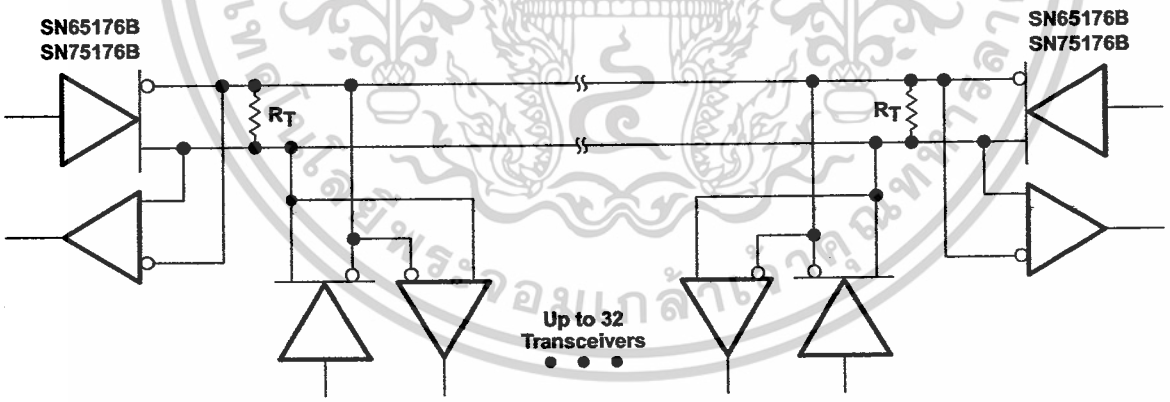


Figure 16

APPLICATION INFORMATION



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

Figure 17. Typical Application Circuit

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.