ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

ปริญญานิพนธ์ ชุดตอบคำถามสำหรับการสอนทางไกล

ANSWERING MACHINE FOR LONG DISTANCE LEARNING

นักศึกษา

1. นายกิตติพงษ์ ปานาพุฒ รหัสประจำตัว 40031303
2. นางสาวพรวิไล สุขมาก รหัสประจำตัว 40031320
3. นางสาวศิริพร มิขำ รหัสประจำตัว 40031331
4. นายอุกฤษณ์ ธัญญานนท์ รหัสประจำตัว 40031342

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

1. ดร. สุรสิทธิ์ ราตรี
2. อาจารย์โกศล ตราชู

คณะกรรมการสอบปริญญานิพนธ์	ลายมือชื่อ
1. ดร. สุรสิทธิ์ ราตรี	.....
2. อาจารย์กิติพงศ์ มะโน	.....
3. อาจารย์อำพล ทองระอา	.....
4. อาจารย์สุรพงษ์ สิริพงศ์ดี	.....
5. อาจารย์สุระชัย พิมพ์สาลี	.....
6. อาจารย์อมรชัย ชัยชนะ	.....

วัน/เดือน/ปีที่สอบ 22 พฤศจิกายน พ.ศ. 2541 เวลา 14.00-15.00 น.

สถานที่สอบ ห้อง ค.310 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม .....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 6 เดือน ..... พ.ศ. 42

ปริญญานิพนธ์

ชุดตอบคำถามสำหรับการสอนทางไกล

ANSWERING MACHINE FOR LONG DISTANCE LEARNING

นายกิตติพงษ์ ปานาพุฒ

นางสาวพรวิไล สุขมาก

นางสาวศิริพร มิขำ

นายอุกฤษณ์ ธัญญานนท์

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์

เรื่อง ชุดตอบคำถามสำหรับการสอนทางไกล

ANSWERING MACHINE FOR LONG DISTANCE LEARNING

ผู้จัดทำ

1. นายกิตติพงษ์ ปานาพุฒ
2. นางสาวพรวิไล สุขมาก
3. นางสาวศิริพร มิขำ
4. นายอุกฤษณ์ ธัญญานนท์

อาจารย์ที่ปรึกษา

ลงนาม .....

(ดร. สุรสิทธิ์ ราตรี)

ลงนาม .....

(อาจารย์โกศล ตราชู)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม .....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง ชุดตอบคำถามสำหรับการสอนทางไกล

ANSWERING MACHINE FOR LONG DISTANCE LEARNING

วัตถุประสงค์

1. เพื่อศึกษาการรับรหัสจากชุดตอบคำถาม จำนวน 4 ชุด นำมาเก็บไว้เป็นชุดไฟล์ข้อมูลบนเครื่องคอมพิวเตอร์
2. เพื่อออกแบบชุดตอบคำถาม จำนวนชุด 4 ชุด
3. เพื่อทำการสร้างชุดตอบคำถาม จำนวนชุด 4 ชุด และโปรแกรมสำหรับรับข้อมูลมา เก็บไว้บนคอมพิวเตอร์
4. เพื่อทดลองหรือทดสอบและสามารถนำชุดตอบคำถาม จำนวนชุด 4 ชุด และโปรแกรมสำหรับรับข้อมูลเก็บไว้บนคอมพิวเตอร์
5. เพื่อนำชุดตอบคำถามไปใช้งานได้

ประโยชน์ที่คาดว่าจะได้รับ

1. มีความรู้เรื่อง วิธีการรับรหัสจากชุดชุดตอบคำถามจำนวน 4 ชุด นำมาเก็บไว้เป็นไฟล์ข้อมูลบนคอมพิวเตอร์
2. ได้วงจรต้นแบบ ชุดรหัสแป้นพิมพ์
3. ได้เครื่องต้นแบบ ชุดรหัสแป้นพิมพ์ 4 ชุด (ประกอบ ด้วย แป้นพิมพ์ 4 ชุดและ ตัวรับ)
4. ได้ผลการทดลองและผลการทดลอง ผลการใช้งานชุดรหัสแป้นพิมพ์ 4 ชุด
5. นำชุดรหัสแป้นพิมพ์ไปใช้งานด้านการสอน ในสถาบันการศึกษาต่าง ๆ

ชุดตอบคำถามสำหรับการสอนทางไกล

นายกิตติพงษ์ ปานาพุฒ
นางสาวพรวิไล สุขมาก
นางสาวศิริพร มิขำ
นายอุกฤษณ์ ธัญญานนท์

อาจารย์ที่ปรึกษา
ดร. สุรสิทธิ์ ราตรี
อาจารย์โกศล ตราชู

ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ เสนอ ชุดตอบคำถามสำหรับการสอนทางไกล เพื่อนำไปเป็นเครื่องต้นแบบที่สามารถนำไปใช้งาน ในการส่งคำตอบสำหรับการเรียนการสอนทางไกล แบบทางเดียว ซึ่งจะทำให้ครูผู้สอนสามารถประเมินผลการเรียนการสอนได้ ซึ่งการสร้างชุดดังกล่าวนี้ได้ทำการออกแบบโดยใช้ไมโครคอนโทรลเลอร์ซึ่งเป็นตัวควบคุมการทำงาน ลักษณะการทำงานจะเป็นการสื่อสารแบบอนุกรมกับเครื่องคอมพิวเตอร์กับชุดตอบคำถาม ซึ่งเครื่องคอมพิวเตอร์จะเป็นอุปกรณ์เก็บข้อมูล โดยจะมีโปรแกรมเป็นตัวควบคุมการใช้งาน

ANSWERING MACHINE FOR LONG DISTANCE LEARNING

MR.KITTIPONG PANAPUT
MISS. PORNWILAI SUKMAK
MISS. SIRIPORN MI-CKUM
MR.UGRIT TANYANON

ADVISORS

DR.SURASIT RATREE
MR.KOSON TRACHU

1998

ABSTRACT

This Thesis presents the project of answering machine for long distance learning. The target is the prototype of its that can use for long distance learning in one direction case. That will help the teacher can evaluation the students. This suit create and design used by microcontroller to control . It use serial communication between computer and the answering machine. The Computer use to stored data by control program.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จล่วงไปได้ด้วยดี เนื่องจากได้รับคำปรึกษาอย่างดียิ่ง จากอาจารย์ที่ปรึกษาปริญญานิพนธ์ทุกท่าน รวมทั้งอาจารย์ภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่มีความกรุณาให้คำปรึกษา และข้อเสนอแนะต่าง ๆ

ขอขอบคุณบิดา-มารดา ที่ได้ให้ความช่วยเหลือทางด้านทุนทรัพย์ และได้ให้กำลังใจในการทำปริญญานิพนธ์ฉบับนี้ และขอขอบคุณเพื่อน ๆ สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ในความมีน้ำใจเอื้อเฟื้อ อันดีงามที่มีให้กันเสมอมา

สารบัญ

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ(ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ในการเรียนการสอนทางไกลในปัจจุบันที่เป็นแบบทางเดียว ซึ่งในการสอนจำเป็นจะต้องมีการวัดประเมินผลการเรียนเป็นสิ่งหนึ่งที่ครูผู้สอนที่จะต้องทราบคำตอบจากผู้เรียนว่ามีความเข้าใจในบทเรียนเพียงใด ซึ่งจะต้องมีการส่งคำตอบจากผู้เรียนที่อยู่ห่างไกล ว่ามีการตอบส่วนมากข้อใดและส่วนน้อยข้อใด เพื่อที่จะสามารถวิเคราะห์ตัวผู้เรียนหรือตัวผู้สอนได้ ซึ่งเครื่องมือที่นำมาใช้ต้องอาศัยเทคโนโลยีและอุปกรณ์ที่มีคุณภาพ แต่อุปกรณ์ต่าง ๆ เหล่านั้นที่ใช้ประกอบการเรียนการสอนนั้นมีราคาสูงเกือบทั้งหมด จะต้องนำเข้ามาจากต่างประเทศ ซึ่งจะทำให้ทางรัฐบาลมีงบประมาณไม่เพียงพอที่จะจัดซื้อให้กับสถานศึกษา จึงได้เกิดความคิดในการสร้าง ชุดตอบคำถามสำหรับการสอนทางไกล(Answering machine for long distance learning) นี้ขึ้น

1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังต่อไปนี้

1 มีช่องสัญญาณอินพุต จากแป้นพิมพ์ จำนวน 4 ชุด
2. สามารถติดต่อสื่อสารทางพอร์ตอนุกรมของคอมพิวเตอร์
3. ข้อมูลที่ได้จากการกดจะเก็บเป็นไฟล์ พร้อมที่จะนำไปใช้งาน
4. สามารถที่จะใช้กับระบบปฏิบัติการ ดอส

1.3 เนื้อหาโดยสังเขป

เนื้อหาในปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่าง ๆ เพื่อความสะดวกต่อการศึกษาและในการทำความเข้าใจ ในแต่ละบทประกอบด้วยเนื้อหาที่สำคัญดังนี้

บทที่ 2 ทฤษฎีและหลักการ โดยประกอบไปด้วยเนื้อหาในเรื่องของมาตรฐานการสื่อสาร RS-232 มาตราฐานการสื่อสาร RS-485 หลักการการส่งสัญญาณข้อมูล BUS ต่าง ๆ ทฤษฎีการรับ-ส่งข้อมูล แบบ OSI (กล่าวถึงรายละเอียด 2 ระดับ คือ Physical Layer และ Data link layer) การจัดการข้อมูลอนุกรมของไมโครคอนโทรลเลอร์ 8051

บทที่ 3 การออกแบบ การสร้างและการทำงาน โดยกล่าวถึงการสร้างและการออกแบบฮาร์ดแวร์ และซอฟต์แวร์ รวมทั้งหลักการทำงานในส่วนต่าง ๆ

บทที่ 4 การทดลอง ผลการทดลอง โดยกล่าวถึงขั้นตอนการทดลองทางด้านฮาร์ดแวร์และซอฟต์แวร์ของโครงงานนี้เพื่อตรวจสอบว่าโครงงานนี้สามารถทำงานได้ตรงตามวัตถุประสงค์ที่ตั้งไว้หรือไม่

บทที่ 5 บทสรุป ปัญหา แนวทางการแก้ปัญหาและการพัฒนา เป็นการสรุปผลการทำงานและได้เสนอแนะแนวทางในการแก้ปัญหา ให้มีประสิทธิภาพและการนำมาใช้งานอย่างได้อย่างกว้างขวางมากขึ้น

ในภาคผนวกแสดงรายละเอียดของข้อมูล โปรแกรมและรายการอุปกรณ์ต่าง ๆ ที่ได้ใช้จัดทำโครงงานดังนี้

ภาคผนวก ก รูปเครื่องต้นแบบ

ภาคผนวก ข ผังการทำงานและโปรแกรมการทำงาน

ภาคผนวก ค คู่มือการใช้งาน

ภาคผนวก ง ลายวงจรพิมพ์และการวางอุปกรณ์บนแผ่นพิมพ์

ภาคผนวก จ รายการอุปกรณ์

ภาคผนวก ฉ รายละเอียดข้อมูลและคุณสมบัติของอุปกรณ์

บทที่ 2
ทฤษฎีและหลักการ

2.1 เทคนิคการสื่อสารข้อมูลดิจิตอล (Digital Data Communication - Technique)

สำหรับอุปกรณ์คอมพิวเตอร์ 2 เครื่องที่เชื่อมโยงกันด้วยสายสื่อสารเพื่อแลกเปลี่ยนข้อมูลกัน โดยปกติแล้วข้อมูลจะถูกส่งผ่านที่ละ 1 บิตต่อครั้งผ่านสายสื่อสาร แต่ละบิตของข้อมูลที่ถูกส่งผ่านไปอย่างต่อเนื่องกัน อาจจะส่งไปในลักษณะแบบ อนุกรม (Serial) หรือแบบ ขนาน (Parallel) เพื่อให้อัตราการส่งข้อมูลเพิ่มมากขึ้น เวลา (หมายถึง อัตราช่วงเวลา หรือช่องว่าง) ของบิตเหล่านี้จะต้องเท่ากันทั้งทางด้านเครื่องส่ง และเครื่องรับ เทคนิคที่ทำให้เวลาที่ปลายทางทั้งสองด้านพร้อมกันมี 2 วิธีคือ วิธีแบบอะซิงโครนัส (Asynchronization) และวิธีแบบซิงโครนัส (Synchronization)

2.1.1 ความแตกต่างของการสื่อสารข้อมูลแบบขนานกับแบบอนุกรม

ในการสื่อสารข้อมูลโดยผ่านสายสื่อสารทำได้ 2 วิธีคือ การสื่อสารข้อมูลแบบอนุกรมหรือเรียงลำดับ (Serial) และการสื่อสารข้อมูลแบบขนาน (Parallel) การสื่อสารข้อมูลดิจิตอลทั้ง 2 แบบมีความแตกต่างกันหลายประการดังนี้

1. การส่งบิตต่างกัน

คำว่า " อนุกรม " หมายถึงหนึ่งต่อหนึ่งเรียงลำดับกันไป ดังนั้นการส่งข้อมูล(หรือบิต)แบบอนุกรมจึงเป็นการส่งข้อมูลทีละ 1 บิตต่อครั้งผ่านทางสายการสื่อสาร

แต่การส่งข้อมูลแบบขนานจะเป็นการส่งข้อมูลเป็นชุดของบิตเรียกว่า ไบต์ (Byte) จำนวนบิตในแต่ละไบต์ขึ้นอยู่กับจำนวนสายข้อมูล (Data Line) เช่น ถ้าสายสื่อสารมีสายข้อมูล 8 สาย ดังนั้นในการส่งข้อมูลทีละ 1 บิตต่อครั้งต่อสายสื่อสาร จะได้จำนวนข้อมูลทั้งหมดเท่ากับ 8 บิต หรือ 1ไบต์ โดยมีการแปลงรหัส (Code) ของบิตแทนตัวอักขระ (Character) ก่อนทำการส่งออกไป

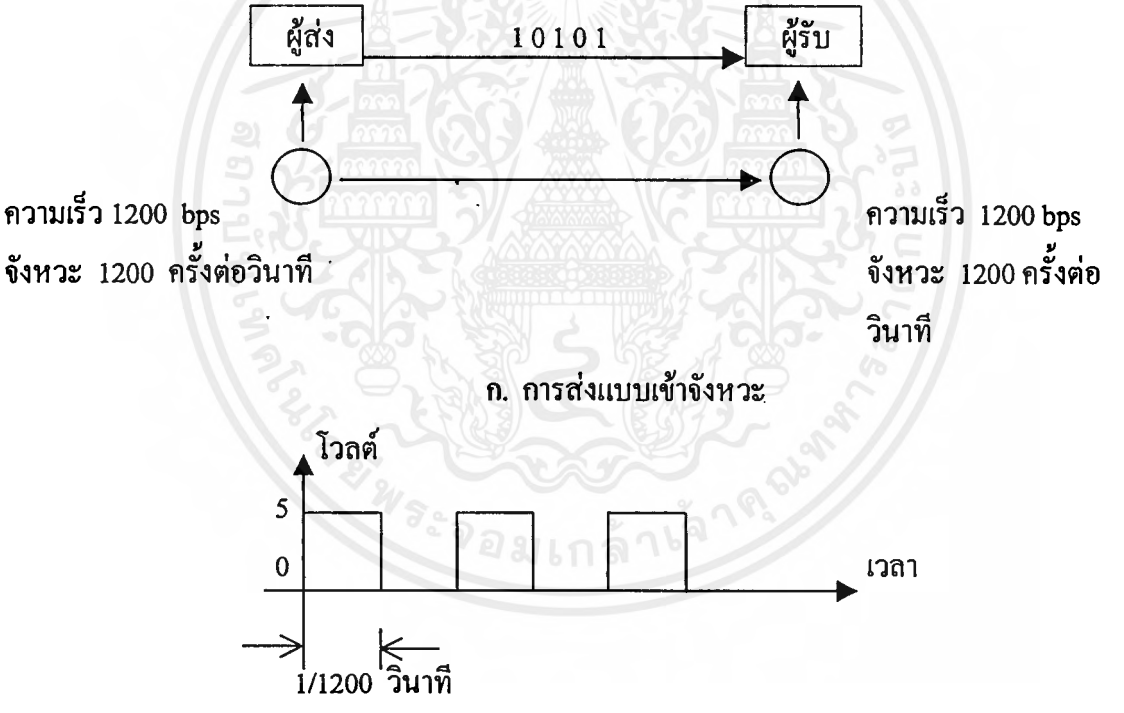
2. ระยะทางไกลต่างกัน

การส่งข้อมูลแบบอนุกรมเป็นการส่งข้อมูลทีละ 1 บิต ความผิดพลาดจึงเป็นไปได้น้อยมาก จึงเหมาะสำหรับการส่งข้อมูลในระยะทางไกล ๆ เช่น จากไมโครคอมพิวเตอร์ไปยังลูกข่ายที่อยู่คนละชั้น หรือคนละอาคารหรือไกลกว่านั้น

ส่วนการส่งข้อมูลแบบขนานนั้น แม้จะสามารถส่งข้อมูลได้เป็นจำนวนมากแต่โอกาสผิดพลาดก็สามารถเกิดขึ้นได้มากด้วยเช่นกัน โดยเฉพาะในการส่งข้อมูลระยะทางไกล ๆสัญญาณข้อมูลอาจจะจางหายหรือผิดเพี้ยนไปกับความต้านทานของสายส่งได้ ดังนั้นจึงเหมาะกับการส่งข้อมูลในระยะใกล้คือน้อยกว่า 100 ฟุต เช่น ระหว่างเครื่องคอมพิวเตอร์กับเครื่องพิมพ์ เป็นต้น

3. การเข้าจังหวะ (Synchronous) ต่างกัน

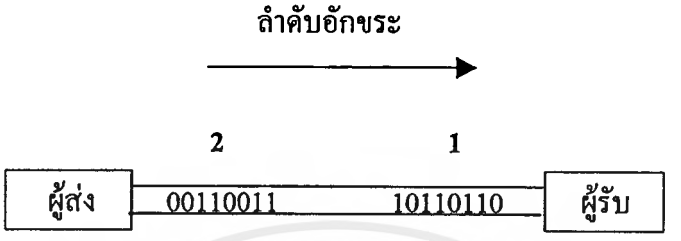
3.1 การเข้าจังหวะบิต (Bit Synchronization) ในการส่งข้อมูลแบบอนุกรมข้อมูลจะถูกส่งทีละ 1 บิต เรียงลำดับของการรับส่ง และการรับข้อมูลจะต้องตรงกัน นั่นคือผู้ส่ง และผู้รับจะต้องส่ง และรับข้อมูลด้วยความถี่เดียวกัน และด้วยอัตราความเร็วเท่ากัน เรียกว่า การเข้าจังหวะบิต เทคนิคในการทำให้ลำดับของบิตทั้ง 2 ด้านตรงกันคือการใช้สัญญาณนาฬิกา (Clock) กำหนดจังหวะของเวลาบิตเริ่มต้น และบิตสิ้นสุด หรือทั้งอักขระให้พร้อมกันทั้งทางผู้ส่ง และผู้รับ



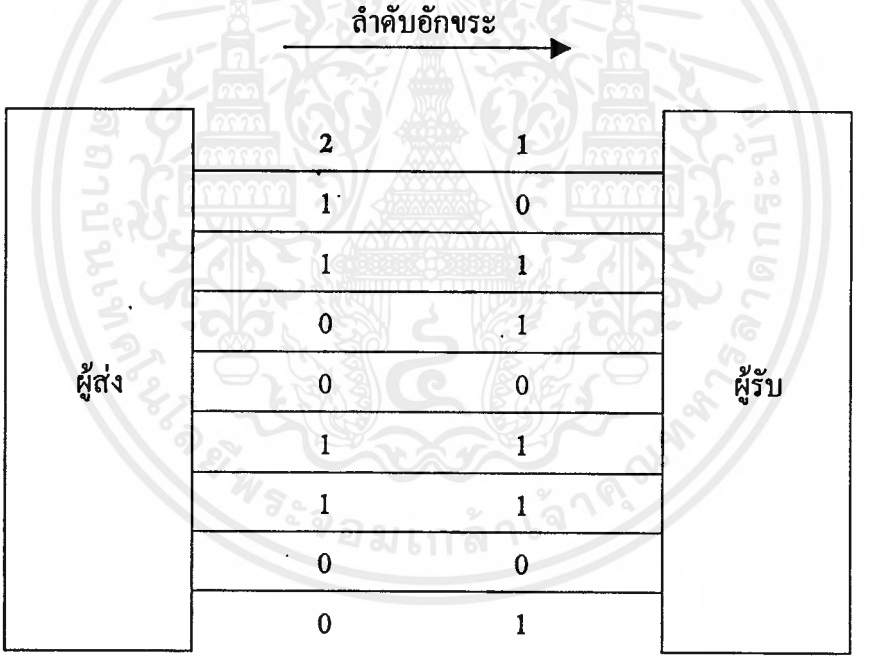
รูปที่ 2.1 เทคนิคการเข้าจังหวะบิตโดยใช้สัญญาณนาฬิกา

3.2 การเข้าจังหวะอักขระ (Character Synchronization) ในการส่งข้อมูลแบบอนุกรมนั้นผู้รับจะต้องจัดลำดับของบิตที่รับมารวมเป็นตัวอักขระ ตำแหน่งของแต่ละบิตในตัวอักขระจะต้องถูกต้อง แต่สำหรับในการส่งข้อมูลแบบขนาน เนื่องจากข้อมูลจะถูกส่งมาทีละอักขระอยู่แล้วผู้รับ

เพียงแต่ตรวจสอบว่าบิตใดเป็นบิตเริ่มต้น และบิตใดเป็นบิตสุดท้ายของแต่ละอักขระ วิธีการที่จะทำให้รู้ว่าบิตใดอยู่ตำแหน่งใดของตัวอักขระ ก็ต้องอาศัยหลักการส่งข้อมูลแบบซิงโครนัส และแบบ อะซิงโครนัส (Synchronous and Asynchronous Transmission)



ก. การส่งสัญญาณข้อมูลแบบอนุกรม (ทีละบิต)



ข. การส่งสัญญาณข้อมูลแบบขนาน (ทีละอักขระ)

รูปที่ 2.2 การส่งสัญญาณข้อมูลแบบอนุกรมและแบบขนาน

3.3 ค่าใช้จ่ายต่างกัน ในการส่งข้อมูลแบบอนุกรมต้องการเพียง 1 ช่องทางสื่อสาร(Channel) ในสายส่งสัญญาณ และความเร็วในการส่งข้อมูลจัดอยู่ในขั้นความเร็วต่ำคือ 300-1,200 บิตต่อวินาที สายส่งสัญญาณที่ใช้จึงสามารถเลือกใช้แบบราคาถูกได้ เช่น สายเกลียวคู่ เป็นต้น

ส่วนการส่งข้อมูลแบบขนานนั้นต้องการช่องทางสื่อสารอย่างน้อย 8ช่องทาง (1ไบต์= 8บิต) ดังนั้นถ้าจะส่งข้อมูลด้วยความเร็วสูง (มากกว่า 9,600 บิตต่อวินาที) จะต้องใช้สายส่งคุณภาพสูง เช่น สายโคแอกเชียล ดังนั้นราคาของสายส่งสัญญาณจึงต้องสูงกว่าการส่งข้อมูลแบบอนุกรม

2.1.2 การส่งข้อมูลแบบอะซิงโครนัสและแบบซิงโครนัส

ในการส่ง และรับข่าวสาร สิ่งที่เครื่องส่ง และเครื่องรับจะต้องมีเหมือนกัน ได้แก่ อัตราเร็วของการส่งช่วงเวลาของสัญญาณ และช่องว่างระหว่างบิตก่อนจะทำการส่งสัญญาณข้อมูลดิจิตอล เราจะต้องทำการแปลงรหัสของตัวอักขระของข่าวสารที่จะส่งเป็นรหัสของบิตเสียก่อน แล้วจึงส่งแต่ละบิตออกไปยังผู้รับ จากนั้นผู้รับจะต้องทำความเข้าใจ (อ่าน) กับกลุ่มของบิตที่แทนตัวอักขระเหล่านั้นว่าบิตใดเป็นบิตแรก และบิตใดเป็นบิตสุดท้ายของอักขระ แต่ถ้าหากว่าจังหวะของเวลาในการอ่าน และการส่ง-รับข้อมูลของเครื่องส่ง และเครื่องรับต่างกัน ก็จะทำให้เกิดความผิดพลาดในการส่ง-รับข้อมูลได้ การเข้าจังหวะหรือการซิงโครนัส (Synchronization) ของเวลาระหว่างผู้ส่ง-ผู้รับจำเป็นอย่างยิ่งที่จะต้องเหมือนกัน และเท่ากัน

ในการเข้าจังหวะสามารถแบ่งได้เป็น 3 ระดับ คือ

1. การเข้าจังหวะบิต หรือ การซิงโครนัสบิต (Bit Synchronization) เพื่อกำหนดจุดเริ่มต้น และจุดสิ้นสุดของการส่งข้อมูลของแต่ละบิต

2. การเข้าจังหวะอักขระ หรือ การซิงโครนัสอักขระ (Character Synchronization) เพื่อกำหนดจุดเริ่มต้น และจุดสิ้นสุดการส่งข้อมูลของแต่ละตัวอักขระ

3. การเข้าจังหวะบล็อก หรือ การซิงโครนัส (Block Synchronization) เพื่อกำหนดตำแหน่งจุดเริ่มต้น และจุดสิ้นสุดของจำนวนข้อมูลขนาดใหญ่ หรือบล็อกข้อมูล

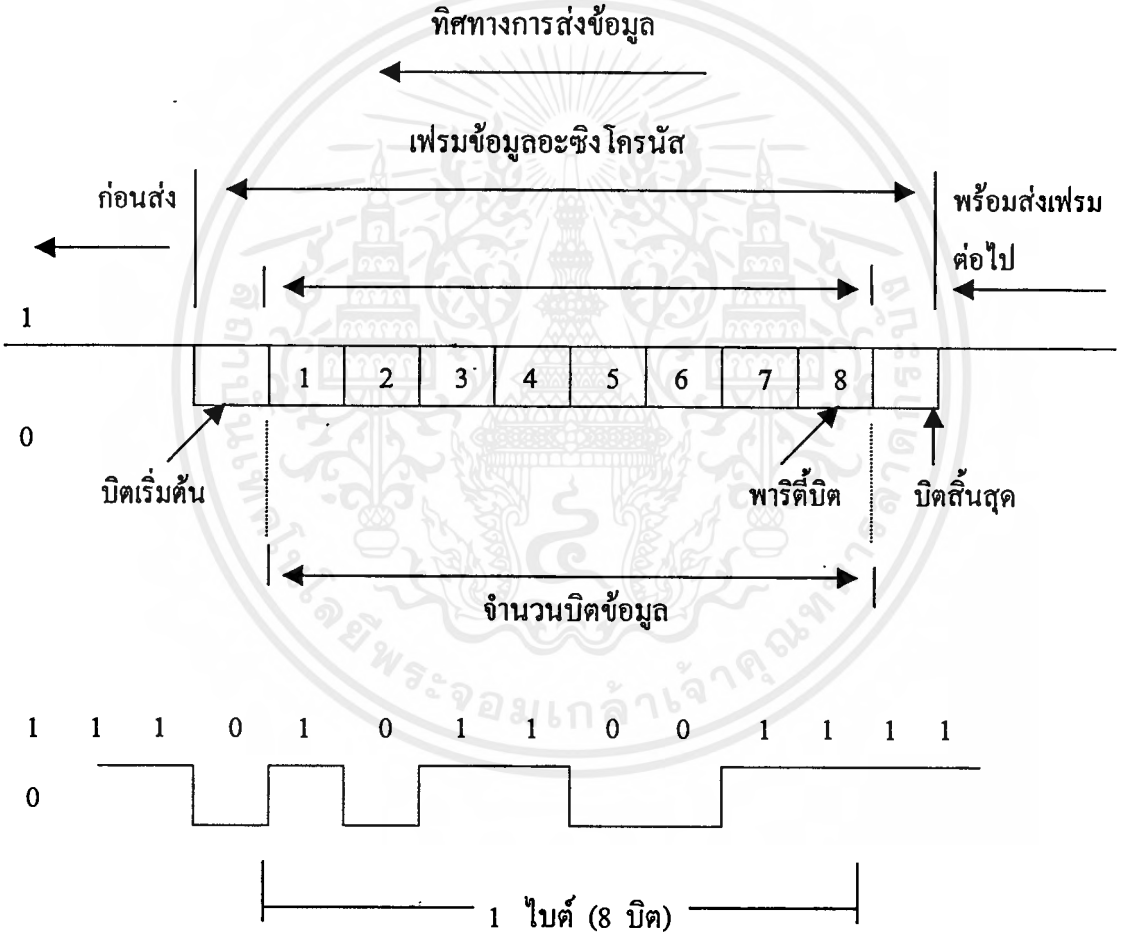
เทคนิคการสื่อสารข้อมูลระหว่างผู้ส่งกับผู้รับข้อมูลคือ วิธีการอะซิงโครนัส(Asychronouse) บางครั้งเรียกว่า วิธีเริ่ม และหยุด (Start/Stop) และอีกวิธีหนึ่งก็คือ วิธีซิงโครนัส (Synchronous)

เทคนิคในการส่งข้อมูลแบบอะซิงโครนัส และแบบซิงโครนัสนี้จะเกี่ยวข้องกับเรื่องของโปรโตคอล (Protocal) ที่ใช้ในการควบคุมการสื่อสารข้อมูลระหว่างอุปกรณ์คอมพิวเตอร์ 2 เครื่อง หรือทั้งเครือข่าย โปรโตคอลที่ใช้ในการส่งข้อมูลแบบอะซิงโครนัสที่รู้จักกันทั่วไปคือ โปรโตคอลเทเลไทป์ หรือ TTY (Teletype Protocal) ส่วนโปรโตคอลที่ใช้ในการส่งข้อมูลแบบซิงโครนัสที่รู้จักกันดีได้แก่ BSC หรือไบซิงก์ (Binary Synchronous Communicattions : BISYNC) SDLC (Synchronous Data Link Control) และ HDLC (High-level Data Link Control) เป็นต้น

1. การส่งข้อมูลแบบอะซิงโครนัส

ในการส่งข้อมูลดิจิตอลแบบอะซิงโครนัส กลุ่มของบิตจำนวน 5 บิต (รหัสโบดอต) หรือ 8 บิต (รหัสแอสกี้) จะแทนตัวอักขระที่ถูกส่งออกไปเป็นเฟรม (Frame) บางครั้งเราเรียกว่าเป็นแบบ "Start/Stop"

การส่งข้อมูลจะส่งทีละอักขระโดยที่ช่วงเวลาระหว่างแต่ละอักขระจะเป็นเท่าไรก็ได้ ดังนั้นตัวเครื่องรับจะต้องตรวจสอบว่า บิตใดเป็นบิตเริ่มต้นของอักขระ และบิตใดเป็นบิตสุดท้ายของอักขระ



รูปที่ 2.3 การส่งสัญญาณข้อมูลดิจิตอลเป็นเฟรมโดยวิธีอะซิงโครนัส

ในการส่งอักขระแต่ละตัวอักขระจะประกอบด้วยบิตเริ่มต้น (1 บิต) + ข้อมูล (8 หรือ 7 บิต) + 1 พาริตี้บิต(แต่อาจจะไม่ใช้ก็ได้) + บิตสิ้นสุด (1 บิต) รวมเป็น 10 บิต คิดเป็น 1 เฟรม

1.1 ขั้นตอนการส่งข้อมูล

ขั้นตอนของการส่งข้อมูลดิจิตอลโดยวิธีอะซิงโครนัสมีดังนี้

1. ก่อนจะเริ่มทำการส่งข้อมูล สัญญาณจะมีค่าเป็น "1" ตลอดเวลา

2. เมื่อเริ่มส่งข้อมูลสัญญาณของบิตแรกจะเปลี่ยนเป็น "0" นั่นคือบิตเริ่มต้น เครื่องรับจะเริ่มสัญญาณนาฬิกาของตัวเอง เมื่อเวลาผ่านไป ½ บิต ถ้าสัญญาณยังคงเป็น "0" ต่อไป อีก ½ บิต ต่อมาก็จะเป็นการเริ่มของสัญญาณข้อมูล แต่ถ้าสัญญาณกลับไปเป็น "1" อีก ก็แสดงว่าเกิดความผิดพลาดอันเกิดจากสัญญาณรบกวนในสายส่ง และยังไม่มีสัญญาณข้อมูลใดๆ ส่งมายังปลายทาง

หลังจากได้เริ่มบิตเริ่มต้นแล้ว ผู้ส่งจะเริ่มส่งรหัสบิตของอักขระ อาจจะเป็น 5 บิต หรือ 8 บิต หรือ 7 บิต แล้วตามด้วยพาริตี้บิต ตามรูป 2.3 เป็นการส่งสัญญาณข้อมูลขนาด 8 บิต สำหรับ 1 อักขระ โดยเป็นสัญญาณข้อมูล 7 บิต บิตที่ 8 เป็นพาริตี้บิตคี่ (Odd) จากนั้นสัญญาณจะเป็น "1" ไปอีก 1 บิต ซึ่งถือว่าเป็นบิตสิ้นสุด สัญญาณจะเป็น "1" ต่อไปเรื่อยๆ จนกว่าจะเริ่มมีการส่งสัญญาณข้อมูลในเฟรมต่อไป

การส่งสัญญาณข้อมูลแบบอะซิงโครนัสนั้นมีใช้กันอย่างกว้างขวาง เพราะเทคนิคในการส่งสัญญาณข้อมูลไม่ยาก รวมสายส่งสัญญาณมีราคาถูก ส่วนใหญ่ใช้ในการส่ง-รับข้อมูลกันระหว่างเครื่องคอมพิวเตอร์บุคคล (PC) กับศูนย์บริการข้อมูลที่อยู่ไกลออกไป เช่น โฮสต์คอมพิวเตอร์ของตลาดหลักทรัพย์หรือระบบธนาคาร

เนื่องจากการสื่อสารข้อมูลดิจิตอลแบบอะซิงโครนัสมีความเร็วในการส่งข้อมูลต่ำ จึงมักใช้กับลูกข่ายที่ไม่มีบัฟเฟอร์ นอกจากนั้นเวลาประมาณ 20 เปอร์เซ็นต์ของการส่งอักขระแต่ละตัวจะสูญเสียไปกับบิตเริ่มต้น บิตสิ้นสุดและบิตตรวจสอบ

ในการส่งข้อมูลที่เป็นบล็อกมีอักขระมากกว่า 1 อักขระ มักจะส่งโดยวิธีซิงโครนัสซึ่งจะกล่าวถึงต่อไป

1.2 การตรวจสอบความผิดพลาด

การตรวจสอบความผิดพลาดของข้อมูลในการสื่อสารข้อมูลดิจิตอลแบบอะซิงโครนัส เราสามารถตรวจสอบได้จากพาริตี้ ซึ่งแบ่งเป็นพาริตี้คี่ (Odd) และพาริตี้คู่ (Even) เช่น ถ้าเราส่งข้อมูลเป็นพาริตี้คู่ นั่นคือเมื่อรวมบิตของอักขระทั้งหมดที่เป็น "1" กับพาริตี้บิต (ซึ่งอาจจะเป็น "0" หรือ "1") แล้วจะได้เป็นจำนวนคู่ ซึ่งเมื่อเครื่องรับได้รับเฟรมข้อมูลไปแล้ว ถ้ารวมบิตของ "1" ทั้งหมดได้เป็นจำนวนคี่ แล้วแสดงว่าข้อมูลที่รับมามีความผิดพลาดเกิดขึ้น

ตัวอย่างจากรูปที่ 2.3 บิตข้อมูลมี 7 บิตคือ 1011001 และกำหนดพาริตี้บิตเป็นพาริตี้คู่ ดังนั้นค่าของพาริตี้บิตจึงเป็น "1" เพื่อให้จำนวนบิต "1" ทั้งหมดใน 8 บิตรวมกันเป็นจำนวนคี่ วิธีตรวจสอบความผิดพลาดในการส่งข้อมูลแบบอะซิงโครนัสหรือเรียกว่า การตรวจสอบพาริตี้

2. การส่งข้อมูลแบบซิงโครนัส

สำหรับเทคนิคในการสื่อสารข้อมูลคอมพิวเตอร์ที่ให้ประสิทธิภาพที่ดีกว่าการสื่อสารข้อมูลแบบอะซิงโครนัส คือ การสื่อสารข้อมูลแบบซิงโครนัสลักษณะของข้อมูลที่ถูกส่งผ่านสายสื่อสารจะถูกส่งไปเป็นบล็อกของอักขระหรือกลุ่มบิต (Block of Characters or Bits) โดยไม่จำเป็นต้องมีบิตเริ่มต้น และบิตสิ้นสุดเช่น เดียวกับการสื่อสารข้อมูลแบบอะซิงโครนัส

การพิจารณาเวลาเริ่มต้น และเวลาที่สิ้นสุดของบล็อกข้อมูลแต่ละบล็อก เราพิจารณาจากกลุ่มบิตส่วนหัวคุมการส่งข้อมูลมากกว่าที่จะเป็นบิตข้อมูล ข้อมูลที่รวมกับข่าวสารควบคุมการส่งข้อมูลนี้เราเรียกว่า เฟรม (Frame)

รูปแบบของเฟรมนั้นจะต้องขึ้นอยู่กับว่าในการส่งข้อมูลนั้นจะเป็นการส่งแบบซิงโครนัสอักขระ (Character Synchronization) หรือแบบซิงโครนัสบิต (Bit Synchronization)

2.2 ทฤษฎีของการรับ – ส่งข้อมูล แบบ OSI

แบบจำลองของการรับและการส่งข้อมูล OSI แบบจำลองนี้ถูกพัฒนาขึ้นโดย International Standard Organization (OSI) (องค์การที่สร้างมาตรฐานระหว่างประเทศ) และจำลองนี้มีชื่อเรียกว่า OSI (Open System Interconnection) ความหมายของชื่อนี้ ก็คือ การเปิดกว้างสำหรับระบบใดๆ ก็ตามต้องการสื่อสารด้วยกัน ต่อไปจะเรียกว่า OSI เท่านั้น

OSI ประกอบด้วย 7 เลเยอร์ หลักการสำหรับทั้ง 7 เลเยอร์ก็คือ

1. เลเยอร์จะถูกกำหนดขึ้นมาเมื่อมีข้อแตกต่างด้านความนัย (abstraction)
2. แต่ละเลเยอร์จะมีการกำหนดการทำงานอย่างละเอียด และสมบูรณ์
3. แต่ละฟังก์ชั่นภายในเลเยอร์จะพยายามมุ่งไปสู่ในระดับมาตรฐานของโปรโตคอล
4. ขอบเขตของเลเยอร์จะถูกเลือก และจำกัดให้มีปริมาณการเชื่อมต่อ (interface) ระหว่างเลเยอร์น้อยที่สุด
5. จำนวนของเลเยอร์จะต้องมากพอที่จะทำให้ฟังก์ชั่นที่จำเป็น และแตกต่างกัน ไม่อยู่ในเลเยอร์เดียวกัน และเลเยอร์จะต้องมีไม่มากจนเกินไปจนทำให้ โครงสร้างใหญ่เกินไป

ในหัวข้อต่อไปนี้จะขอกล่าวถึงเลเยอร์ 2 ระดับแรก คือ Physical Layer และ DataLink Layer

2.2.1 ฟิสิกส์คัลเลเยอร์ (Physical Layer)

ภายใน Physical Layer จะเป็นการเกี่ยวพันกับการส่งข้อมูลดิบเป็นบิตผ่านช่องสื่อสารข้อมูล ในวิธีการออกแบบจะต้องทำให้แน่ใจว่า ข้อมูล "1" ที่ส่งออกไปและที่ปลายทางสามารถรับ "1" ได้ถูกต้อง จึงเกิดคำถามที่เกี่ยวข้องก็คือต้องใช้แรงดันเท่าใดสำหรับแทนเลข "1" และต้องใช้เท่าใดสำหรับแทนเลข "0" และแต่ละบิตจะต้องห่างกันเท่าใดการส่งข้อมูลสามารถทำการส่งแบบสองทิศทางได้หรือไม่ ก่อนเริ่มต้นส่งข้อมูล จะต้องทำให้เกิดการเชื่อมต่อได้อย่างไร และเมื่อทำการส่งเสร็จแล้วจะยกเลิกการต่อได้อย่างไร ในส่วนที่ลึกลงไปอีกก็คือเรื่องของ คอนเนคเตอร์ (Connecter) หรือเป็นอุปกรณ์ที่ทำหน้าที่เป็นขั้วต่อของสายเข้ากับเครื่องคอมพิวเตอร์หรืออุปกรณ์สื่อสาร ว่าต้องการจะใช้กี่ขา และในแต่ละขาทำหน้าที่อะไร จะเห็นได้ว่าประเด็นที่น่าสนใจจะเกี่ยวข้องกับทางไฟฟ้า ทางรูปร่างภายนอก และยังรวมต่อถึงในวิธีการเชื่อมต่ออีกด้วย รวมทั้งตัวการที่อยู่ภายใต้ Physical Layer ซึ่งจะทำหน้าที่ส่งข้อมูลจริงๆ ส่วนใหญ่แล้วขอบเขตของ Physical Layer จะเกี่ยวพันโดยตรงกับทางไฟฟ้า

ขั้นตอนการทำงานของ Physical Layer จะประกอบไปด้วย คือ

1. Machanical คือ Layout ของ Physical Layer และเป็นการอินเตอร์เฟสสัญญาณ
2. Electrical คือ ระดับแรงดันที่อยู่ในสาย
3. Functional คือ คุณสมบัติของแต่ละบิตที่ในสาย
4. Procedural คือ การตรวจความพร้อมของสัญญาณไฟฟ้า
5. Activate คือ การทำให้ข้อมูลส่งไปในสายส่ง เป็นการเริ่มต้นการส่ง
6. Maintain คือ การส่งบิตของข้อมูล (ต่อ บิตของข้อมูล)
7. Deactivate คือ เป็นฟังก์ชั่นในการยกเลิกการส่ง (Terminate ข้อมูล)

2.2.2 ดาต้าลิงค์เลเยอร์ (Data Link Layer)

จุดประสงค์หลักของ Data Link Layer ก็พยายามทำให้การส่งข้อมูลดิบเหมือนกับไม่มีการผิดพลาดเกิดขึ้นให้เลเยอร์ที่สูงขึ้นไป (ซึ่งก็คือ network Layer) นำไปใช้งานได้อย่างถูกต้อง วิธีการก็คือทางฝ่ายส่งจะทำการแตกข้อมูลออกเป็นก้อนๆ เรียกว่า เฟรมข้อมูล (Data – frame) ทำการส่งเฟรมข้อมูลออกไปทีละชุด และรอการตอบรับ (Acknowledge frame) ซึ่งตอบกลับมาโดยผู้รับ โดยปกติแล้ว Physical Layer จะไม่คำนึงถึงข้อมูลว่ามีความหมายใด ๆ จึงเป็นหน้าที่ของ Data Link Layer ที่จะต้องทำการสร้าง และตอบรับขอบเขตของเฟรม (Frame boundary) ซึ่งสามารถทำได้โดยการเติมบิตเข้าไปในจุดเริ่มต้น และจุดสุดท้ายของเฟรม แต่จุดที่ต้องระวังก็คือถ้าหากข้อมูลที่เพิ่มเข้าไปซ้ำกับข้อมูลที่ต้องการส่งจริงๆ

สัญญาณรบกวนจากภายนอกก็เป็นปัญหาหนึ่งที่อาจจะทำให้เฟรมขาดหายไปได้ ในกรณีนี้โปรแกรมที่ควบคุม Data Link Layer ที่เครื่องต้นทางจะส่งข้อมูลซ้ำมาใหม่ อย่างไรก็ตามการส่งเฟรมเดียวกันออกมาหลายๆ ครั้งก็อาจจะทำให้เกิดเฟรมซ้ำกันได้ วิธีการป้องกันก็คือ เฟรมที่ซ้ำกันจะส่งออกไปก็ต่อเมื่อมีการตอบรับ (Acknowledgment frame) ส่งมาบอกว่าข้อมูลหายหรือถูกทำลายไปก่อน ภายใน Data Link Layer จะมีการเชื่อมต่อ (Interface) หลายๆ แบบเพื่อให้ Network Layer สามารถเลือกใช้ได้

ข้อที่น่าสนใจอีกอย่างหนึ่งก็คือ ทำอย่างไรจึงจะให้ทางฝ่ายที่เก็บข้อมูลซึ่งส่งเร็วกว่าทางฝ่ายรับ สามารถทำงานได้ วิธีการบางอย่างที่น่าสนใจก็คือการอธิบายวิธีการสื่อสาร ให้สม่ำเสมอโดยการพักข้อมูลในบัฟเฟอร์ (Buffer) ข้อมูลไว้ชั่วคราวแล้วค่อยส่งต่อออกไป

มีจุดน่าสนใจอีกอย่างก็คือถ้าหากสายส่งสามารถส่งข้อมูลได้ 2 ทางแล้วจะก่อให้เกิดปัญหาทางด้านซอฟท์แวร์แก่ Data Link Layer ปัญหาก็คือเฟรมตอบรับจาก A ไปยัง B จะต้องแย่งชิงการใช้งานสายส่งกับเฟรมข้อมูล B ไปยัง A

2.3 มาตรฐานการสื่อสาร RS-232

ในยุคแรกๆ คอมพิวเตอร์ทุกขนาด (ส่วนใหญ่จะเป็นขนาดใหญ่ถึงใหญ่มาก) จะสื่อสารถึงกันผ่านทางสายสัญญาณ ซึ่งจะนำพาสัญญาณดิจิตอลล้วนๆ ในทางปฏิบัติแล้วระยะทางไกลที่สุดที่สามารถส่งข้อมูลได้จะไกลประมาณ 1 ไมล์ โดยมีความเร็วในการส่งข้อมูลเพียง 300 บิตต่อวินาที (bps) รูปแบบการสื่อสารแบบนี้ผู้ใช้คอมพิวเตอร์หรือผู้ใช้เครื่องลูกข่าย (Terminal) จำเป็นจะต้องติดตั้งสายนำสัญญาณชนิดพิเศษต้องมีตัวทวนสัญญาณทุกๆ 2000-3000 ฟุต เพื่อให้การสื่อสารข้อมูลนั้นสามารถเชื่อถือได้ ระบบเครือข่ายยุคเริ่มแรกนี้ยากที่จะสื่อสารข้อมูลกับบริเวณที่อยู่นอกเหนือไปจากบริเวณที่ติดตั้งเครื่องไว้

ด้วยมาตรฐานในขณะนั้น นั่นเป็นวิธีการที่ง่าย แต่ค่าใช้จ่ายสูงในการขนส่งกลุ่มข้อมูลระหว่างคอมพิวเตอร์กับผู้ใช้โดยเฉพาะเมื่อผู้ใช้บางส่วนมองว่าการสื่อสารแบบดิจิตอล ในระบบนี้เป็นเหมือนตัวขัดขวางอนาคตของการประมวลผลขนาดใหญ่ และการสื่อสารทางไกลในระบบดิจิตอลทั้งหมด

ต่อมาเกิดมีแนวคิดในการใช้งานระบบนำสัญญาณสาธารณะทั่วไป (ระบบสายโทรศัพท์ธรรมดา) ในการส่ง และรับข้อมูลดิจิตอลขึ้นมา เครื่องโมเด็มหรือชุดข้อมูลก็ถูกผลักดันเข้าสู่การให้บริการ

ชุดของข้อมูลจะถูกจัดเตรียมไว้ที่ศูนย์คอมพิวเตอร์แต่ละแห่งเพื่อใช้ในการผสมสัญญาณและถอดรหัสสัญญาณดิจิตอลที่จะส่งไประหว่างคอมพิวเตอร์เครื่องอื่นๆ ที่อยู่ห่างไกลออกไปผ่าน

ทางระบบโทรศัพท์แบบแอนะล็อก อุปกรณ์ตัวใหม่ที่เรียกว่าตัวผสมสัญญาณ และถอดรหัสสัญญาณ (Modulation and Demodulation) ก็ได้กลายเป็นสิ่งที่เรารู้จักกันในปัจจุบันในนามของ "โมเด็ม" (Modem)

และแล้วก็เป็น RS-232 ด้วยการกำเนิดขึ้นมาของโมเด็มวิถีทางในการประมวลผลจากระยะไกลก็สดใสขึ้นแม้ว่าจะมีค่าใช้จ่ายสูง และใครๆ ก็สามารถทำโมเด็มขึ้นมาได้ ในเดือนพฤษภาคม ปี ค.ศ.1960 ได้มีเหตุการณ์ที่จำเป็นต้องมีการกำหนดมาตรฐานการอินเตอร์เฟสทางไฟฟ้าระหว่างคอมพิวเตอร์กับโมเด็มหรือชุดข้อมูล มีการวางแผนกำหนดแรงดันไฟฟ้าด้านมาตรฐาน พร้อมทั้งตัวแปรทางสัญญาณมาตรฐาน และชื่อเรียกมาตรฐานขึ้นมา เพื่อแยกแยะตัวนำสัญญาณในสายเคเบิลที่ต่อกับคอมพิวเตอร์ และโมเด็ม

เทคโนโลยีการประมวลผลจากระยะไกล (Teleprocessing technology) ได้รับความนิยมอย่างรวดเร็ว และเครื่องคอมพิวเตอร์ และเครื่องลูกข่ายจำนวนมากที่ถูกสร้างขึ้นโดยผู้ผลิตต่าง ๆ กันก็จำเป็นจะต้องมีการติดต่อสื่อสารข้อมูลระหว่างกันมากขึ้น ถ้าปราศจากมาตรฐานในแบบเดียวกันแล้วอุตสาหกรรมประมวลผลจากระยะไกลทั้งหมดก็จะหยุดชะงักจากความไม่มีมาตรฐานนั้น

ข้อเสนอของ RS-232 ได้กำหนดการอินเตอร์เฟสทั้งทางลอจิก และทางกายภาพระหว่างอุปกรณ์คอมพิวเตอร์กับโมเด็ม โดยพื้นฐานแล้วจำเป็นต้องมีการอินเตอร์เฟส 2 ชุด เพื่อให้ได้การสื่อสารแบบดิจิตอลที่สมบูรณ์ ระหว่างคอมพิวเตอร์ 2 เครื่อง เครื่องลูกข่ายจะมีการอินเตอร์เฟสทั้งทางลอจิก ทางกายภาพกับโมเด็ม จะประกอบด้วยสายนำสัญญาณหลาย ๆ เส้น สำหรับเพื่อควบคุมส่งและรับข้อมูลและยังเป็นตัวกำหนดจังหวะและเวลากล่าว โดยสรุปก็คือ การอินเตอร์เฟสระหว่างคอมพิวเตอร์กับโมเด็มจะมีมาตรฐานอยู่ชุดหนึ่งและเรียกว่า การอินเตอร์เฟสแบบ EIA RS-232

การอินเตอร์เฟสชุดที่ 2 จะเชื่อมต่อโมเด็มเข้ากับช่องทางการสื่อสาร(สายโทรศัพท์, เส้นแก้วใยนำแสง, ดาวเทียม, และอื่นๆ) สำหรับผู้ใช้คอมพิวเตอร์ส่วนบุคคลส่วนใหญ่แล้ว ช่องทางการสื่อสารนี้ก็คือ สาย 2 เส้นของโทรศัพท์แบบแอนะล็อก

2.3.1 ลักษณะวงจรของ RS-232

จากตารางที่ 2.1 จะแสดงสายสัญญาณเพียง 11 เส้นจาก 25 เส้น ที่เป็นไปได้ของระบบ RS-232 ที่ต้องการใช้ในการทำการสื่อสารระหว่าง DTE ไปยัง DCE ให้สมบูรณ์ส่วนมากแล้วสามารถละทิ้งสายวงจรตัวตรวจจับอัตราสัญญาณข้อมูลได้ (Data Signal Rate Detector) และสายวงจรกราวด์ (Protective Ground) ออกไปได้ ทำให้เหลือสายสัญญาณที่ต้องต่อเพียง 9 เส้น

RS-232 เป็นข้อกำหนดการอินเตอร์เฟสมาตรฐาน และสามารถใช้เพื่อจุดประสงค์อื่นๆต่างๆกันไป เช่น การสื่อสารแบบซิงโครนัส (Synchronous communication) และรูปแบบการสื่อสาร

ที่ต้องการสัญญาณนาฬิกา และสัญญาณกำหนดจังหวะเวลาเพิ่มเติมขึ้นมาในความเป็นจริงแล้วคุณสามารถทำให้มีการสนทนากันจาก DTE ไปยัง DCE โดยใช้สายเพียง 3 เส้น จากจำนวน 11 เส้น ที่แสดงในตารางที่ 3 ถ้าอุปกรณ์ DTE และ DCE ใช้ซอฟต์แวร์ที่เขียนขึ้นตามความต้องการของลูกค้า (Custom written software) ก็จะใช้เพียง TD , RD และสายกราวด์สัญญาณเท่านั้นในการย้ายข้อมูลไปตามสายตัวนำ 3 เส้นนี้

ต่อไปนี้คือ ข้อกำหนดของขาสัญญาณ 11 ขา ที่นำไปใช้งาน

1. ขา 1 (Protective Ground Circuit,AA) ขานี้จะต่อเข้ากับตัวถังของอุปกรณ์ และสามารถต่อเข้ากราวด์ภายนอกถ้าอุปกรณ์อื่นๆ ต้องใช้ขานี้

ตารางที่ 2.1 รายละเอียดสำหรับการอินเตอร์เฟส RS-232 โดยใช้คอนเน็กเตอร์แบบ DB-25

หมายเลขขาสัญญาณ	ชื่อของสายสัญญาณ
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Common
8	Received Line Signal Detector
9	Reserved For Testing
10	Reserved For Testing
11	Unassigned
12	Secondary Receive Line Signal Detector
13	Secondary Clear To Send
14	Secondary Transmitted Data
15	Transmission Signal Element Timing
16	Secondary Received Data

ตารางที่ 2.1 (ต่อ)

หมายเลขขาสัญญาณ	ชื่อของสายสัญญาณ
17	Receiver Signal Element Timing
18	Unsigned
19	Secondary Request To Send
20	Data Terminal Ready
21	Signal Quality detector
22	Ring Indicator
23	Data Signal Rate Detector
24	Transmitter Signal Element Timing
25	Usassigned

2. ขา 2 (Transmitted Data Circuit BA,TD) เป็นขาสัญญาณข้อมูลที่ออกมาจากอุปกรณ์ DTE กระแสบิตข้อมูลอนุกรมจากขานี้ คือข้อมูลที่จะถูกถ่ายทอดไปโดยโมเด็ม หรือถูกถอดรหัสโดยอุปกรณ์ DCE ที่มี

3. ขา 3 (Received Data Circuit BB,RD) สัญญาณที่ขานี้จะถูกสร้างจากอุปกรณ์ DCE กระแสบิตข้อมูลอนุกรมนี้จะกำเนิดขึ้นที่อุปกรณ์ DTE ปลายทาง และเป็นผลผลิตของวงจรรับข้อมูลของอุปกรณ์ DCE สัญญาณนี้มักจะเป็นข้อมูลดิจิตอลที่ถูกสร้างขึ้นโดยอุปกรณ์ DCE ที่มีความฉลาดหรือจากวงจรถอดสัญญาณ (demodulation) ของโมเด็ม

4. ขา 4 (Request to Send Circuit CA,RTS) สัญญาณนี้จะเตรียมพร้อมอุปกรณ์ DCE สำหรับการทำการส่งข้อมูล เมื่อสัญญาณ RTS นี้อยู่ในสภาวะ "สภาวะการทำงาน" จะทำให้อุปกรณ์ DCE อยู่ในโหมดส่งข้อมูล (transmit mode) ในขณะที่สัญญาณนี้อยู่ในสภาวะ "OFF" จะทำให้อุปกรณ์ DCE อยู่ในโหมดรับข้อมูล (receive mode) อุปกรณ์ DCE ควรจะตอบสนองต่อสัญญาณ RTS "สภาวะการทำงาน" โดยการทำให้สัญญาณ Clear to Send (CTS) อยู่ในสภาวะ "การทำงาน" ด้วยเมื่อสัญญาณ RTS อยู่ในสภาวะ "OFF" สัญญาณนี้จะอยู่ในสภาวะไม่ทำงาน ขึ้นอีก จนกว่าสัญญาณ CTS จะอยู่สภาวะ "OFF" เสียก่อน สัญญาณนี้จะถูกใช้ร่วมกับสัญญาณ DTR , DSR และ DCD ขาสัญญาณ RTS จะถูกใช้อย่างมากในการควบคุมการไหลของข้อมูล

5. ขา 5 (Clear To Send Circuit CB ,CTR) สัญญาณนี้จะตอบรับกลับไปยังอุปกรณ์ DTE เมื่อได้รับสัญญาณ RTS และข้อมูลสามารถส่งออกไปได้ข้อมูลจะถูกส่งไปมาตรฐานที่ใช้ใน

การสื่อสารได้ก็ต่อเมื่อสัญญาณ CTS นี้อยู่ในสภาวะ "สภาวะการทำงาน"เท่านั้น สัญญาณนี้จะใช้ร่วมกับขา DTR , DSR และ DCD ขาสัญญาณ CTS นี้จะใช้ร่วมกับขา RTS สำหรับควบคุมการไหลของข้อมูล

6. ขา 6 (Data Set Ready Circuit CC,DSR) สัญญาณ DSR จะบอกต่ออุปกรณ์ DTE ว่าอุปกรณ์ DCE ได้ต่อกับตัวกลางการสื่อสารที่ถูกต้องแล้ว และในบางกรณีจะบ่งชี้ว่าสายโทรศัพท์อยู่ในสภาวะ "OFF HOOK" และในสภาวะนี้จะเป็นตัวบ่งชี้ว่าอุปกรณ์ DCE กำลังอยู่ในโหมด dialing หรือกำลังติดต่อกับอุปกรณ์ DCE อีกตัวหนึ่งอยู่ เมื่อสัญญาณ DSR นี้อยู่ในสภาวะ "OFF" อุปกรณ์ DTE ก็ควรจะถูกกำหนดให้ไม่สนใจสัญญาณอื่นๆ ทั้งหมดจากอุปกรณ์ DCE ถ้าสัญญาณนี้ถูกทำให้อยู่ในสภาวะ "OFF" ก่อนอุปกรณ์ DTR แล้วอุปกรณ์ DTE ก็จะสรุปว่าการสื่อสารกันนั้นสิ้นสุดลง

ตารางที่ 2.2 รายละเอียดการต่อคอนเน็กเตอร์แบบ DB 9 ตามมาตรฐาน RS-232

หมายเลขขาสัญญาณ	ชื่อของสายสัญญาณ
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Common
8	Data Carrier Detect
9	Ring Indicator

7. ขา 7 (Signal common Circuit , AB) สายตัวนี้จะให้สัญญาณอ้างอิงของกราวด์ร่วมกันสำหรับวงจรการแลกเปลี่ยนข้อมูลทั้งหมด ยกเว้นวงจร AA หรือ วงจร protective ground ข้อกำหนด RS-232B จะอนุญาตให้วงจรนี้ถูกต่อเพิ่มเติมเข้ากับ Protective ground ภายในอุปกรณ์ DCE ได้ ถ้าจำเป็น

8. ขา 8 (Data Carrier Dectect Circuit CF. DCD) ขานี้ยังรู้จักกันในนามของ Received Line Signal Detect (RLSD) หรือขา Carrier Detect (CD) สัญญาณนี้จะแอกทีฟเมื่อ

เกิดสัญญาณพาหะที่เหมาะสมระหว่างอุปกรณ์ DCE ที่สถานีกับที่อยู่ในระยะไกลเมื่อสัญญาณนี้อยู่ในสภาวะ "OFF" สัญญาณที่ขา RD ควรจะถูกทำให้ค้างอยู่ในสภาวะ "Mark" (สถานะ "1" ในเลขฐานสอง)

9. ขา 20 (Data Terminal Ready Circuit CD, DTR) สัญญาณ DTR ถูกใช้ในการควบคุมการสวิตช์อุปกรณ์ DCE เข้ากับตัวกลางในการสื่อสารสัญญาณ DTR "สภาวะการทำงาน" บ่งชี้ว่าอุปกรณ์ DCE ที่กำลังต่อเชื่อมกันอยู่ ก็ยังคงต่อเชื่อมกัน และถ้าไม่มีการต่อเชื่อมกันก็สามารถทำการต่อเชื่อมกันครั้งใหม่ได้ปกติแล้วสัญญาณ DTR จะอยู่ในสภาวะหยุดการทำงาน เพื่อกระตุ้นให้เกิดภาวะการทำงาน HOOK (วางสาย) (Ring Indicator Circuit CE, RI) สภาวะการทำงาน" ของสัญญาณนี้จะบ่งชี้ว่าได้รับสัญญาณเรียกสายโทรศัพท์จากตัวกลางในการสื่อสาร (สายโทรศัพท์) ปกติแล้วจะขึ้นอยู่กับโปรแกรมควบคุม ในการที่จะทำให้เกิดสัญญาณนี้ขึ้นหรือไม่

10. ขา 23 (Data signal Rate Detector Circuit CH/CI, DSRD) วงจร CH เป็นส่วนประกอบของ CTE และวงจร CI เป็นส่วนประกอบของ DCE สัญญาณที่ขานี้ถูกใช้ในการเลือกค่าอัตราการส่งสัญญาณข้อมูลค่าใดค่าหนึ่งในสองค่าในกรณีที่ใช้โมเด็มที่มีอัตราการส่งข้อมูลได้ 2 ค่า (Dual-rate modems) ถ้าสัญญาณที่ขานี้เป็นการทำงานก็จะเป็นการเลือกอัตราการส่งข้อมูลที่มีค่าสูงที่สุดใน 2 ค่านั้น

อุปกรณ์ที่ใช้มาตรฐาน RS-232 จะมีตัวแปลที่สำคัญ 4 ตัวจะทำให้สามารถทำการสื่อสารแบบอะซิงโครนัสได้สำเร็จซึ่งก็คือ อัตราเร็วในการส่งข้อมูล (Baud rate), จำนวนของบิตข้อมูล จำนวนของบิตสิ้นสุด และบิตตรวจสอบความถูกต้องของข้อมูล (Parity bit) แต่อย่างไรก็ตามการเชื่อมต่อแบบอะซิงโครนัสส่วนใหญ่มักจะไม่ใช้งานพาริตี้บิต เพื่อให้ง่ายต่อการใช้งาน ในตอนเริ่มแรกแล้ว พาริตี้ถูกใช้เพื่อช่วยในการตรวจสอบความถูกต้องของข้อมูลในการเชื่อมโยงข้อมูลที่สำคัญ ๆ ข้อเสียของพาริตี้คือ ถ้าชุดข้อมูลเสียหายใช้การไม่ได้ ก็จะไม่สามารถเรียกชื่อข้อมูลกลับคืนมาได้ไม่ว่าจะใช้เทคนิคพาริตี้แบบไหนก็ตาม

ชุดข้อมูลที่ใช้ส่งในแบบอะซิงโครนัสโดยทั่วไป แล้วจะประกอบด้วยบิตเริ่มต้น , บิตข้อมูลจำนวน 8 บิต และบิตสิ้นสุด ชุดข้อมูลแบบอื่น ๆ พบได้ทั่วไปจะมีบิตข้อมูล 7 บิต พาริตี้คู่และบิตสิ้นสุด

ความกว้างของแต่ละบิตถูกกำหนดด้วยอัตราในการส่งข้อมูล ยกตัวอย่างเช่น ในการเชื่อมโยงที่อัตราเร็วในการส่งข้อมูลที่ 9,600 bps ความกว้างของแต่ละข้อมูลจะเท่ากับ 104 ไมโครวินาที เป็นคาบเวลา ตามสูตร P = 1/f เมื่อ P คือคาบเวลามีหน่วยเป็นวินาที และ f คือความถี่มีหน่วยเป็นเฮิรตซ์ ในกระบวนการที่ก้าวหน้ามากขึ้นค่า Baud rate ยังสามารถวัดได้จากจำนวนครั้งของการเปลี่ยนเฟสของสัญญาณในช่วงเวลากำหนดไว้ได้อีกด้วย

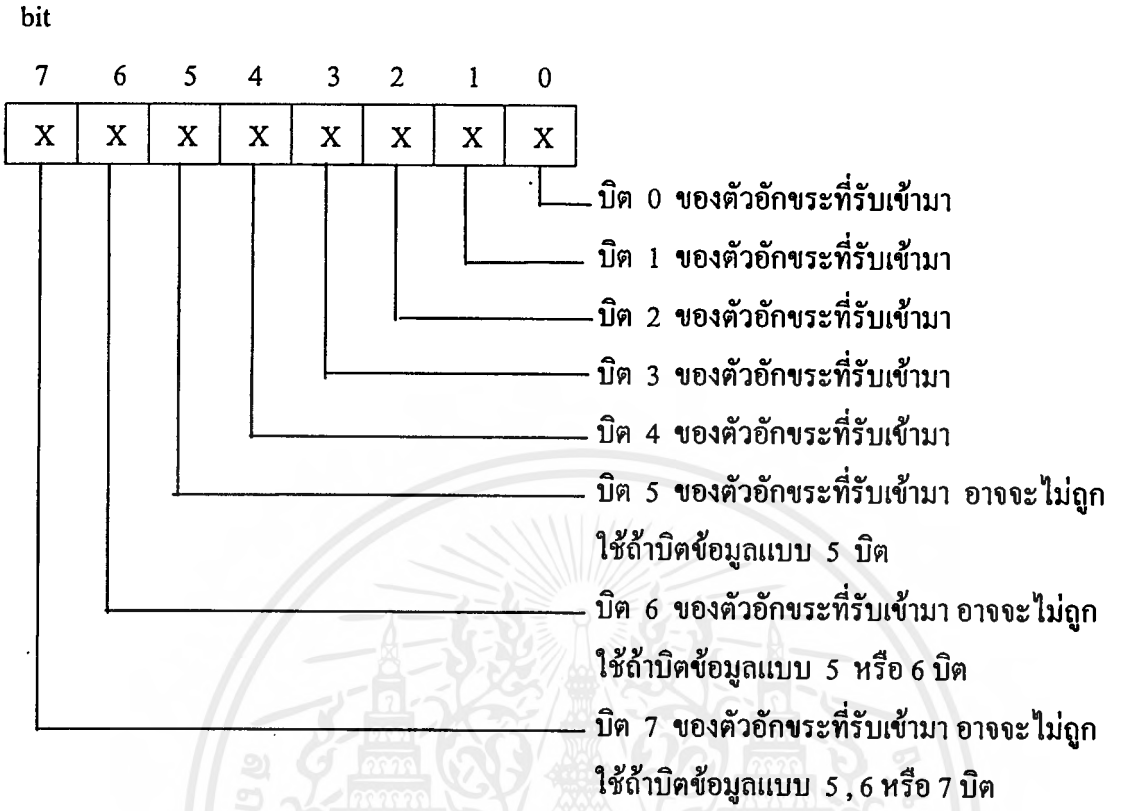
ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส บิตแรกที่ถูกส่งออกไป คือบิตเริ่มต้น บิตนี้จะทำให้อุปกรณ์ DTE ภาครับ สามารถกำหนดช่วงกำหนดการรับบิตข้อมูลให้สัมพันธ์กับอุปกรณ์ DTE ภาคที่ส่งข้อมูลออกมา บิตเริ่มต้นก็คือลอจิก "0" (space) ที่จะคงค่านี้อยู่เป็นเวลา 1 คาบเวลาที่มีความกว้างเท่ากับ 1 บิตปกติแล้วสายสัญญาณเชื่อมต่อนี้ ควรจะอยู่ในสภาวะลอจิก "1" (mark) ก่อนที่จะมีการส่งบิตเริ่มต้นออกมา

เมื่ออุปกรณ์ DTE ด้านรับ ตรวจพบบิตเริ่มต้นมันก็จะรอคอยเป็นเวลานาน 1.5 เท่าของความกว้างของบิตข้อมูล หรือนาน 156 ไมโครวินาที สำหรับอัตราเร็วในข้อมูลที่ 9,600 bps ก่อนที่จะอ่านค่าสถานะของสายสัญญาณเชื่อมต่อนั้น แนวความคิดในการทำเช่นนี้ก็เพื่อให้อุปกรณ์ DTE ด้านรับข้อมูลทำการอ่านค่าสถานะของบิตข้อมูลในจังหวะที่ใกล้กับตรงกลางของบิตข้อมูลให้มากที่สุดเท่าที่จะทำได้ หลังจากรอคอยเป็นเวลานานเท่ากับ 1.5 เท่าของความกว้างของบิตข้อมูลในครั้งแรกแล้ว อุปกรณ์ DTE ภาครับก็จะเปลี่ยนกลับไปอ่านค่าสถานะของบิตข้อมูลที่ส่งเข้ามาในทุก ๆ 104 ไมโครวินาที ถ้าทุกอย่างเป็นไปตามที่วางแผนไว้แล้ว ข้อมูลส่วนที่เหลือจะถูกอ่านเข้ามาตรงกลางของแต่ละบิตข้อมูล

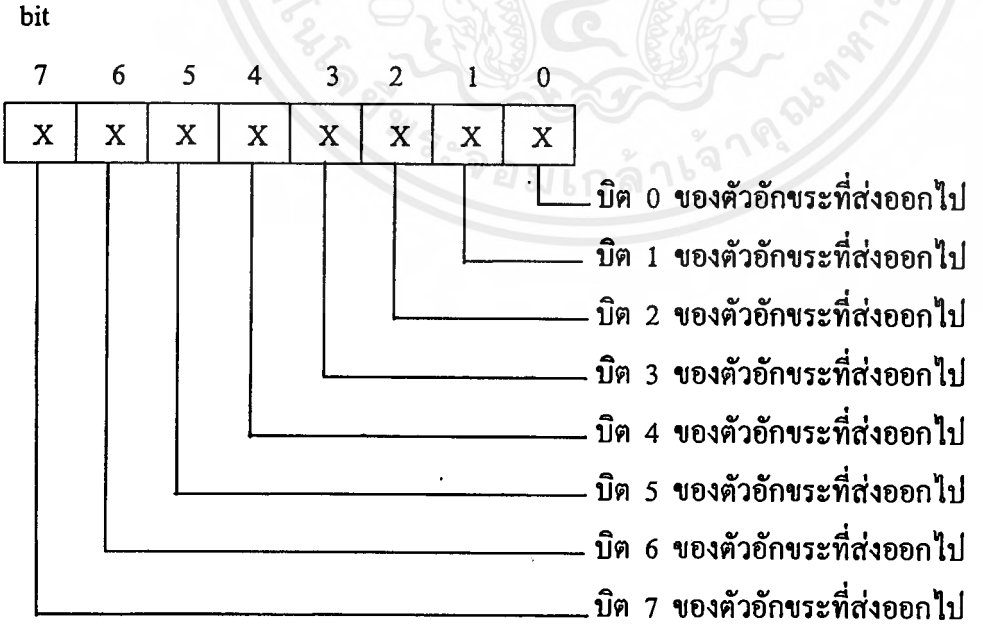
บิตสิ้นสุดข้อมูลจะบอกถึงจุดสิ้นสุดของการส่งตัวอักขระและปกติแล้วจะใช้บิตสิ้นสุดข้อมูล 1 หรือ 2 บิต ในการบอกถึงการสิ้นสุดการส่งตัวอักขระ กระบวนการทั้งหมดนี้ ซึ่งประกอบด้วยบิตเริ่มต้น, บิตข้อมูล, บิตสิ้นสุด จะกระทำซ้ำแล้วซ้ำอีกสำหรับการส่งตัวอักขระแต่ละตัวเพื่อขจัดความผิดเพี้ยนของจังหวะเวลาที่อาจค่อย ๆ สะสมเพิ่มขึ้นได้

สาเหตุหนึ่งที่ทำให้เกิดความผิดพลาดขึ้นกับข้อมูลคือปัญหาเรื่องสายสื่อสาร ในการแก้ไขเราสามารถทำได้โดยการปรับแต่งสายสื่อสาร ในกรณีที่สายสื่อสารที่ใช้ส่ง-รับข้อมูลมีคุณภาพไม่ค่อยดีเราสามารถติดตั้งอุปกรณ์อิเล็กทรอนิกส์เข้ากับสายสื่อสาร เพื่อลดการรบกวนของสัญญาณ หรือลดการสะท้อนของสัญญาณ การปรับแต่งสายสื่อสารทำได้เฉพาะกรณีของการเชื่อมโยงแบบจุดต่อจุดเท่านั้นอีกกรณีหนึ่งคือ การใช้เครื่องปรับเท่า หรือ Equalizer เป็นการปรับแต่งคุณสมบัติของช่องทางการสื่อสาร หรือสายสื่อสารเช่นกัน โดยมากจะมีอยู่ในโมเด็ม โดยโมเด็มจะเป็นตัวรับสัญญาณข้อมูลเข้ามา จากนั้นเครื่องปรับเท่าจะทำงานโดยอัตโนมัติในการปรับแต่งสัญญาณให้ได้รูปร่างดีที่สุด

ปัจจุบันนี้ อุปกรณ์ส่วนใหญ่ที่มีวางจำหน่ายอยู่จะเป็นไปตามมาตรฐาน RS-232C หรือ RS-232D (มาตรฐาน CCITT V.24 และ V.28 ก็ยังคงมีใช้อยู่ทั่วไปอย่างกว้างขวาง) วงจรของ RS-232 ส่วนมากไม่ได้ใช้ในการติดต่อสื่อสารระหว่างเครื่องเทอร์มินัล 2 เครื่องหรือ เครื่องคอมพิวเตอร์ 2 เครื่องโดยตรง



รูปที่ 2.4 รีจีสเตอร์พักข้อมูลที่รับเข้ามา (Receiver Buffer Register)

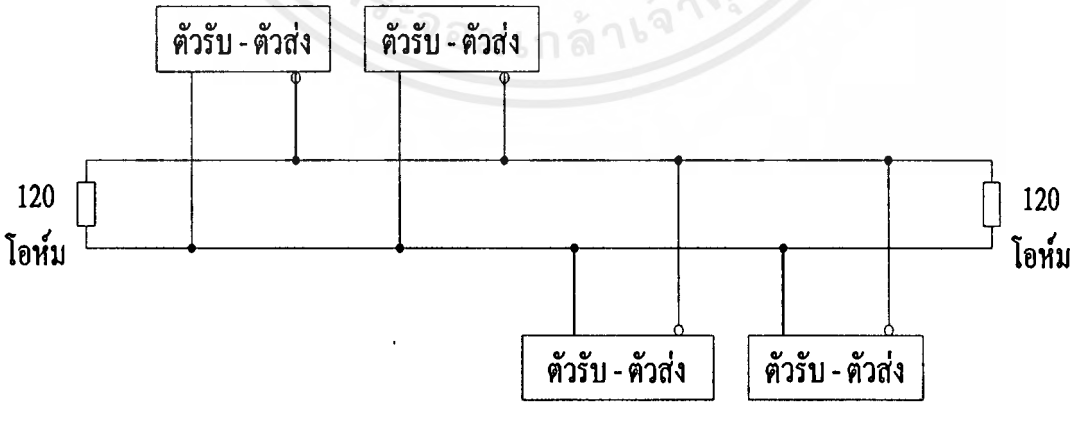


รูปที่ 2.5 รีจีสเตอร์ที่เก็บข้อมูลที่จะส่ง (Transmitter Holding Register)

2.4 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-485

การสื่อสารข้อมูลตามมาตรฐานที่ได้กล่าวมาข้างต้นคือ RS-232C นั้น เป็นมาตรฐานการสื่อสารข้อมูลในแบบที่ใช้การสื่อสารข้อมูลระหว่างอุปกรณ์ หรือแบบจุดต่อจุด (Point to Point) ส่วน RS-422A นั้นเป็นมาตรฐานที่พัฒนามาจาก RS-232C เพื่อทำให้สามารถทำการสื่อสารข้อมูลได้ในระยะทางที่ไกลขึ้น และอัตราการสื่อสารข้อมูลเพิ่มขึ้น แต่ก็ยังเป็นการสื่อสารข้อมูลจากอุปกรณ์หนึ่งไปยังอุปกรณ์อื่นๆ ได้สูงสุดเพียงไม่เกิน 10 ตัวเท่านั้น และไม่สามารถส่งข้อมูลย้อนกลับจากอุปกรณ์ 10 ตัวนั้นกลับมายังอุปกรณ์หลักได้ หรืออาจกล่าวได้ว่าการสื่อสารข้อมูลตามมาตรฐาน RS-422A นั้นเป็นการสื่อสารข้อมูลแบบซิมเพล็กซ์ คือทิศทางข้อมูลเป็นแบบทางเดียวตลอดเวลา ดังนั้นจึงไม่สามารถทำการออกแบบระบบให้เป็นโครงข่ายข้อมูลได้

ด้วยเหตุนี้จึงได้มีการพัฒนามาตรฐานการสื่อสารข้อมูลแบบใหม่ขึ้นเพื่อที่รองรับความต้องการดังกล่าวนี้ ซึ่งก็คือ มาตรฐาน RS-485 สำหรับมาตรฐาน RS-485 นั้น จะเป็นมาตรฐานที่อาศัยหลักการทำงานของสัญญาณแบบดิฟเฟอเรนเชียลเช่นเดียวกับมาตรฐาน RS-422A แต่จะสามารถสื่อข้อมูลได้ทั้ง 2 ทิศทางในสายสัญญาณเพียงคู่เดียว ซึ่งเป็นการสื่อสารข้อมูลแบบฮาล์ฟดูเพล็กซ์ จากผลของการใช้สัญญาณในลักษณะของดิฟเฟอเรนเชียลนี้ จะทำให้ได้ระยะทาง และความเร็วในการสื่อสารข้อมูลสูง เช่นเดียวกับมาตรฐานการสื่อสารข้อมูล RS-422A แต่มาตรฐาน RS-485 นั้นสามารถที่จะสื่อสารระหว่างอุปกรณ์ทั้งการรับ และการส่งได้สูงสุด ถึง 32 ตัว หรืออาจกล่าวได้ว่าการสื่อสารข้อมูลตามมาตรฐาน RS-485 นั้นเป็นการสื่อสารข้อมูลแบบหลายจุด (Multi-Point Communication) โดยโครงสร้างในการสื่อสารข้อมูลแบบ RS-485 แสดงดังรูปที่ 2.18



รูปที่ 2.6 โครงสร้างของการสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-485

ตารางที่ 2.3 การเปรียบเทียบมาตรฐานการสื่อสารข้อมูลของ EIA

พารามิเตอร์	RS-232C	RS-423A	RS-422A	RS-485
โหมดการทำงาน	Single-ended	Single-ended	Differrential	Differrential
จำนวนของตัวรับ และตัวส่งที่ยอมรับได้	1 ตัวส่ง 1 ตัวรับ	1 ตัวส่ง 10 ตัวรับ	1 ตัวส่ง 10 ตัวรับ	32 ตัวส่ง 32 ตัวรับ
ความยาวของคู่สาย สัญญาณรับส่งข้อมูล	50 ฟุต	4000 ฟุต	4000 ฟุต	4000 ฟุต
อัตราการส่งข้อมูลสูงสุด (บิตต่อวินาที)	20 kb	100 kb	10 Mb	10 Mb
ค่าความต้านทาน เอาท์พุทของตัวส่ง(Ω)	NA - Power On 300 - Power Off	NA – Power On 60 kΩ - Power Off	NA – Power On 60 kΩ - Power Off	120 kΩ Power On,Off
ค่าความต้านทานอินพุตของ ตัวรับ(Ω)	3 kΩ ถึง 7kΩ	4 kΩ	4 kΩ	12 kΩ
ความไวตัวรับ	±3 V	±200 mV	200 mV	±200 mV

ตารางที่ 2.4 ข้อเปรียบเทียบระหว่างการส่งข้อมูลแบบขนานและแบบอนุกรม

รูปแบบการสื่อสาร ข้อมูล	แบบขนาน	แบบอนุกรม
1. ระยะทาง	ปกติจะน้อยกว่า 100 ฟุต	ส่งได้ตั้งแต่ระยะทางสั้นๆ ไปจน ถึงระยะทางเป็นไมล์
2. ความเร็ว	อัตราความเร็วสูงมากในระยะที่ไม่ ไกลมากนัก กำหนดได้เป็นจำนวน บิตต่อวินาที	อัตราความเร็วของข้อมูลที่ใช้กัน อยู่ทั่วไปจะอยู่ในช่วง 0 ถึง 2 ล้าน บิตต่อวินาที
4. ความผิดพลาดของ สัญญาณ	ถ้าส่งในระยะทางไกลๆ ความผิด พลาดของข้อมูลจะเกิดขึ้นง่าย	การผิดพลาดของสัญญาณจะมี น้อยลง

ตารางที่ 2.4 (ต่อ)

รูปแบบการสื่อสารข้อมูล	แบบขนาน	แบบอนุกรม
5. ค่าใช้จ่าย	ถ้าส่งในระยะทางไกลๆ จะสิ้นเปลืองค่าใช้จ่ายมาก เพราะต้องใช้สายส่งสัญญาณหลายเส้น	สิ้นเปลืองน้อยกว่าหลายเท่า ถึงแม้ว่าจะใช้อุปกรณ์เปลี่ยนสัญญาณของข้อมูลจากแบบขนานไปเป็นแบบอนุกรม แล้วส่งผ่านสายส่งใช้อุปกรณ์ในการแปลงสัญญาณกลับมาเป็นแบบขนานอีก ก็ยังลงทุนน้อยกว่า

2.4.1 รูปแบบของการสื่อสารข้อมูลแบบ RS-485

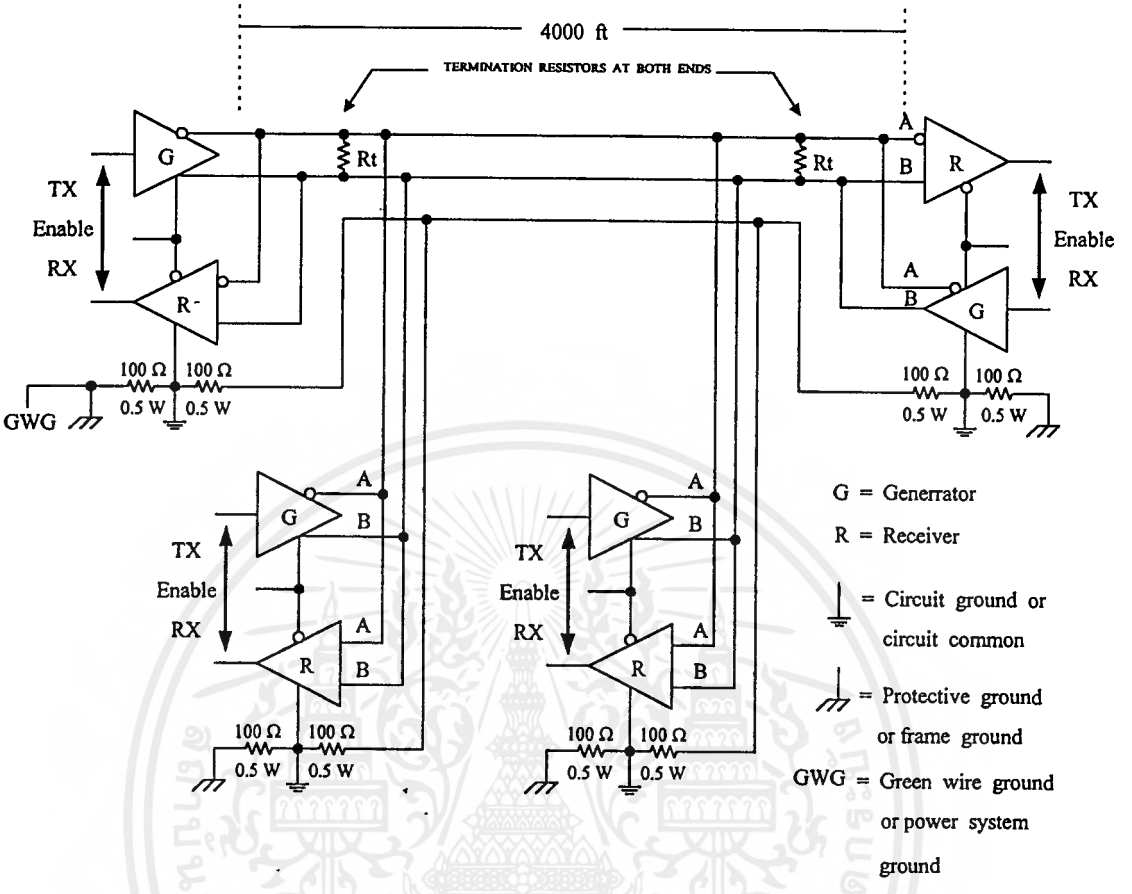
รูปแบบหรือลักษณะการต่อใช้งานของ RS-485 ในลักษณะของเครือข่าย สามารถแบ่งออกได้เป็น 2 รูปแบบ คือ

1. เครือข่ายแบบที่ใช้สายนำสัญญาณสองเส้น (Two Wire)

การสื่อสารข้อมูล RS-485 แบบสองสาย นี้ ถือเป็นการสื่อสารแบบซิมเพล็กซ์ คือ ลักษณะของการรับ-ส่ง ข้อมูล จะเป็นไปในลักษณะผลัดกันรับ-ส่ง โดยการกำหนดว่าจะเป็นการให้เป็นการรับหรือส่ง จะถูกกำหนดด้วยตัวแม่ข่าย (MASTER) ข้อดีของเครือข่ายแบบนี้ คือ จะช่วยประหยัดสายสัญญาณที่ใช้ในการวางเครือข่าย ส่วนข้อเสีย คือ ความเร็วในการรับ-ส่งข้อมูลจะช้าลง

2. เครือข่ายแบบที่ใช้สายนำสัญญาณสี่เส้น (Four Wire)

การสื่อสารข้อมูลมาตรฐาน RS-485 แบบที่ใช้สายนำสัญญาณสี่เส้นนี้ ถือเป็นการสื่อสารแบบฟูลดูเพล็กซ์ คือ ลักษณะของการรับ-ส่งข้อมูลสามารถทำการรับ และส่งข้อมูลได้ในเวลาเดียวกัน ทั้งนี้เพราะว่ามี บัสข้อมูล (DATA BUS) อยู่ จำนวน 4 เส้น ข้อดีของเครือข่ายแบบนี้ คือ ความเร็วของการรับ-ส่งข้อมูลจะเร็วกว่า แบบที่ใช้สายนำสัญญาณสองเส้น แต่ก็มีข้อเสีย คือ จะสิ้นเปลืองค่าใช้จ่ายในส่วนของสายนำสัญญาณมากกว่าแบบที่ใช้สายนำสัญญาณสองเส้นประมาณ 2 ชุด โดยที่โหลด 1 ชุด จะประกอบด้วยตัวส่ง 1 ตัว และตัวรับ 1 ตัว และค่าของความต้านทานที่ต่อคร่อมระหว่างสายสัญญาณมีค่า 60 Ω



รูปที่ 2.7 เครือข่ายของ RS-485 แบบที่ใช้สายนำสัญญาณสองเส้น (Two Wire)

- เอาต์พุตของตัวส่งในสภาวะหยุดการทำงาน มีกระแสรั่วไหลไม่เกิน 100μAในช่วงแรงดันไฟฟ้าระหว่าง -7 ถึง 12 V

- เอาต์พุตของตัวส่งให้แรงดันไฟฟ้าเอาต์พุต 1.5 ถึง 5 V ในช่วงแรงดันไฟฟ้าระหว่าง -7 ถึง 12 V

- ตัวส่งมีวงจรป้องกันตัวเองที่ส่วนเอาต์พุต ในกรณีที่ตัวส่งหลายๆ ตัว ส่งข้อมูลออกมาพร้อมๆ กัน

2.1 คุณสมบัติเฉพาะของตัวรับ RS-485

- ค่าความต้านทานอินพุตมีค่าสูง โดยมีค่าไม่น้อยกว่า 12 kΩ

- ตัวรับ มีค่าแรงดันไฟฟ้าอินพุต ระหว่าง -7 ถึง 12 V

- ตัวรับ สามารถตอบสนองต่อสัญญาณที่แตกต่างจากสัญญาณโหมดร่วมได้ ±200 mV

2.2 ความหมายของยูนิตโหลด (Unit Load)

ยูนิตโหลด เป็นจำนวนที่มากที่สุดของตัวรับ และตัวส่ง ที่สามารถใช้งานบนคู่สายสัญญาณรับ-ส่ง คู่หนึ่ง โดยจะขึ้นกับค่า Unit Load (U.L.) ซึ่ง RS-485 ยอมรับได้ที่ 32 Unit Load ต่อคู่สายสัญญาณ 1 คู่

นิยามของยูนิตโหลด เป็นโหลดที่ใช้กระแส 1 mA ที่แรงดันไฟฟ้าโหมดร่วม 12 V ซึ่งโหลดนี้ประกอบด้วย ตัวส่ง และ/หรือ ตัวรับ แต่ไม่รวมค่าความต้านทานที่เกิดจากความต้านทานที่ต่อคร่อมคู่สายสัญญาณรับ-ส่ง

2.3 คุณสมบัติเฉพาะของคู่ตัวรับ-ส่ง

1. เป็นไปตามมาตรฐาน RS-485, RS-422A, CCITT V.11 และ X.27
2. เอาต์พุตของตัวส่งเป็นแบบ 3 ส่วน (State) ยกเว้น SN 75179B
3. เอาต์พุตตัวส่งสามารถขับกระแสได้สูงสุด 60 mA
4. ค่าความต้านทานอินพุตของตัวรับมีค่าประมาณ 20 kΩ(น้อยที่สุด)
5. ตัวรับมีค่าความไวอินพุต (input sensitivity) ประมาณ ±200 mV
6. ใช้ไฟเลี้ยง 5 VDC

2.4.2 การคำนวนหาจำนวนคู่ตัวรับ-ส่ง RS-485

ตัวส่ง : ค่าของกระแสรั่ว เมื่ออยู่ในสภาวะออฟที่ 12 V มีค่าไม่เกิน 0.1 mA

ตัวส่งมีค่า (U.L.) = 0.1 mA/1.0 mA = U.L.

ตัวรับ : ค่าของกระแสอินพุตที่แรงดันไฟฟ้าอินพุต 12 V มีค่าไม่เกิน 0.6 mA

ตัวรับมีค่า (U.L.) = 0.6 mA/1.0 mA = 0.6 mA

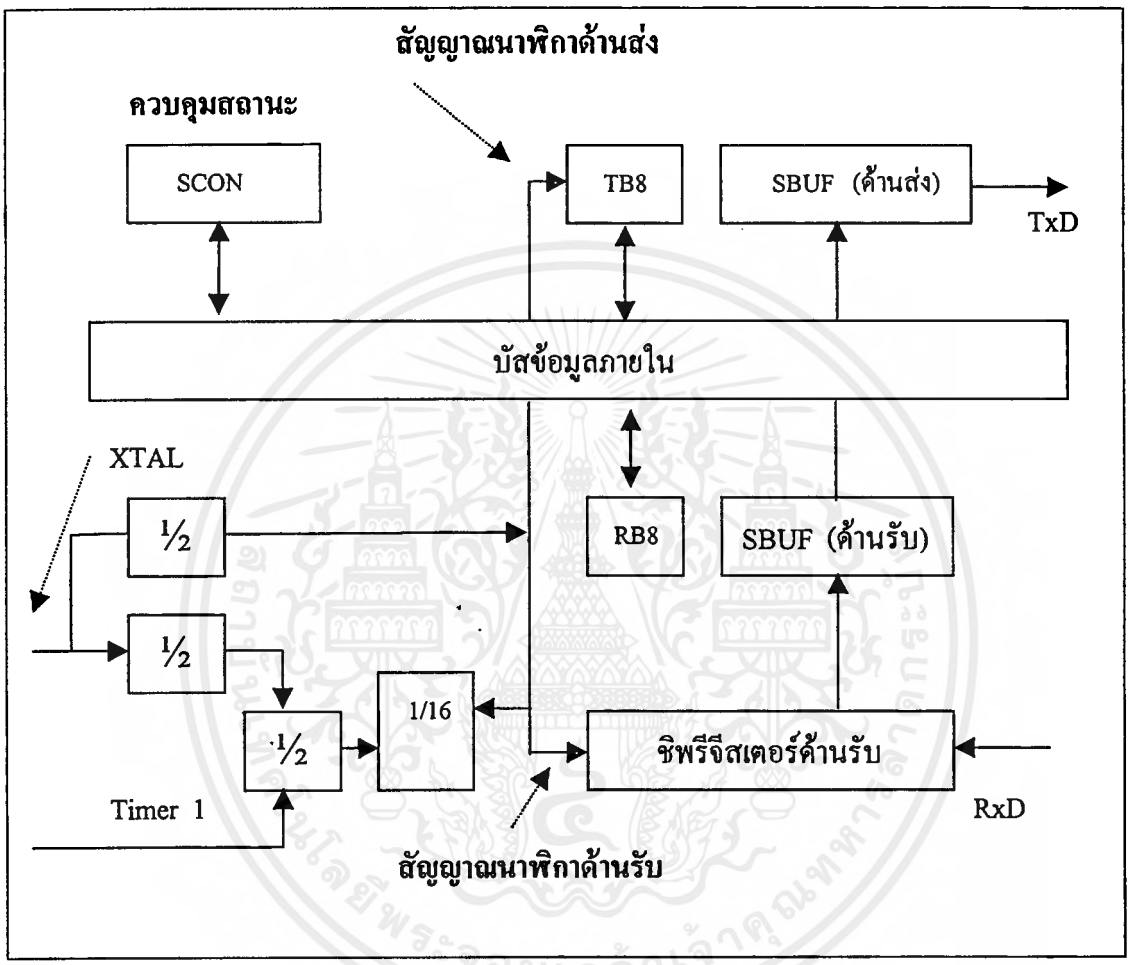
เมื่อพิจารณาโหลด 1 (ตัวรับและตัวส่ง) มีค่า = (0.1+0.6)mA/1.0 mA) = 0.7 U.L.

สายสัญญาณรับ-ส่ง ข้อมูลคู่หนึ่งสามารถรองรับตัวรับ-ส่งได้ = 32/0.7 = 45 คู่

2.5 การจัดการข้อมูลอนุกรมของไมโครคอนโทรลเลอร์ 8051

พอร์ตอนุกรมของ 8051 มีโครงสร้างการทำงานในแบบที่ เรียกว่า ฟูลดูเพล็กซ์ (Full Duplex) ซึ่งหมายถึง ความสามารถในการรับ และส่งข้อมูลอนุกรมได้ในเวลาเดียวกันจากรูปที่ 2.9 แสดงให้เห็นถึงแผนภาพการทำงานอย่างง่ายของวงจรส่วนจัดการข้อมูลอนุกรมของ 8051 โดยทางด้านวงจรของตัวส่ง (Transmitter) ประกอบด้วยรีจีสเตอร์ SBUF ทำหน้าที่เก็บข้อมูลที่จะส่งออก การใช้คำสั่งเขียนหรือโอนย้ายข้อมูลมายังรีจีสเตอร์นี้จะเป็นการส่งข้อมูลนั้นออกไปยังพอร์ตอนุกรมทางขาสัญญาณ TxD (พอร์ต 3.11)โดยอัตโนมัติ ส่วนวงจรด้านตัวรับ (Reciever)

ประกอบด้วยรีจีสเตอร์ SBUF เช่นเดียวกัน แต่ทำหน้าที่เก็บข้อมูลที่นำมาจากส่วนของวงจรเลื่อนบิตหรือชิพรีจีสเตอร์ (Shift register) ของวงจรจัดการข้อมูลอนุกรมภายใน สัญญาณข้อมูลที่รับเข้าจะผ่านมาทางขาสัญญาณ RxD (พอร์ต 3.10)



รูปที่ 2.8 แผนภาพแสดงการทำงานของวงจรส่วนการรับและส่งข้อมูลอนุกรมของ 8051

พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ 8051 สามารถที่จะโปรแกรมให้ทำหน้าที่ในรูปแบบต่าง ๆ กันได้สี่แบบ (หรือเรียกว่าโหมดการทำงาน) โดยการกำหนดค่าบิต SM0 และ SM1 ซึ่งอยู่ภายในรีจีสเตอร์ควบคุมและบอกสถานะ SCON ดังแสดงในรูปที่ 2.9 และภายในตารางที่ 2.5 จะแสดงโหมดการทำงานทั้ง 4 แบบพอร์ตอนุกรม มีดังนี้

นอกจากนี้โหมด 2 และ 3 ยังมีการดำเนินการแบบพิเศษออกไป โดยสามารถนำมาใช้ประโยชน์ในการสื่อสารข้อมูลที่มีไมโครโปรเซสเซอร์หลายตัวทำงานร่วมกันได้

ตารางที่ 2.5 โหมดการทำงานทั้ง 4 แบบของพอร์ตอนุกรม

โหมดทำงาน	คำอธิบาย
โหมด 0	เป็นการขยายพอร์ตอินพุต โดยทำงานร่วมกับไอซีชิพรีจีสเตอร์ภายนอกประเภททีทีแอลหรือซีมอส
โหมด 1	ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART (Universal asychronous receiver / transmitter) โดยการใช้กลุ่มข้อมูลแบบ 10 บิต และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้
โหมด 2	ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และการกำหนดอัตราความเร็วในการส่งข้อมูลคงที่
โหมด 3	ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้

จากแผนภาพรูปที่ 2.8 ชิพรีจีสเตอร์ภายในตัวส่งจะทำหน้าที่ในการเลื่อนบิตข้อมูลออกไปภายนอกโดยไม่มีการบัฟเฟอร์ และเมื่อใดได้มีการเขียนข้อมูลให้กับรีจีสเตอร์ SBUF แสดงว่ามีความต้องการที่จะส่งข้อมูลนี้ออกไปแบบอนุกรม สำหรับชิพรีจีสเตอร์ทางด้านรับจะทำการเลื่อนบิตข้อมูลที่รับเข้ามาเก็บไว้ เมื่อบิตของข้อมูลที่รับมาครบถ้วนตามจำนวนที่กำหนดไว้ตามลักษณะโหมดการทำงานต่าง ๆ แล้ว จะถูกย้ายไปเก็บยังรีจีสเตอร์ SBUF ต่อไป อย่างไรก็ตามการที่จะย้ายข้อมูลนี้จะเกิดขึ้น ก็ต่อเมื่อรีจีสเตอร์ SBUF นั้นไม่มีข้อมูลที่จะทำการส่งหรือได้ส่งข้อมูลออกไปเสร็จสิ้นแล้ว จึงทำการย้ายข้อมูลได้ตามต้องการ ซึ่งอัตราการส่งข้อมูลเทียบกับช่วงเวลาหนึ่งๆ เรียกว่า อัตราบอด (Baud rate) อัตรานี้จะเป็นตัวบ่งบอกความเร็วการรับส่งข้อมูลอุปกรณ์ ส่วนจำนวนข้อมูลการส่งในหนึ่งครั้ง จะเรียกว่า เฟรม (Frame) โดยมีขอบเขตเริ่มต้นตั้งแต่บิตเริ่มต้นจนถึงบิตสิ้นสุด จำนวนของข้อมูลที่อยู่ในระหว่างบิตดังกล่าวทั้งสอง คือ จำนวนบิตในหนึ่ง เฟรม ซึ่งนิยมใช้ 8 บิต การรับส่งข้อมูลแบบนี้มีข้อเสียคือ หากอัตราบอดไม่เท่ากันระหว่างตัวรับและตัวส่ง จะทำให้การตรวจจับบิตเริ่มต้นและบิตสิ้นสุด ตลอดจนสถานะของข้อมูลไม่แน่นอนส่งผลทำให้เกิดารรับส่งข้อมูลผิดพลาดขึ้นได้

ชื่อบิต : SCON ตำแหน่ง : 98h ค่าบิตเริ่มต้น : 0000 0000

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

ชื่อบิต	ตำแหน่ง	ความหมาย
SM0	SCON.7	แฟล็กกำหนดการทำงานแบบมัลติโปรเซสเซอร์
SM1	SCON.6	บิตเลือกโหมดการทำงาน
SM2	SCON.5	บิตเลือกโหมดการทำงาน
REN	SCON.4	แฟล็กยอมให้มีการรับข้อมูล
TB8	SCON.3	ค่าของบิตที่ 9 สำหรับการส่งข้อมูลออก
RB8	SCON.2	ค่าของบิตที่ 9 ของข้อมูลที่รับเข้า
TI	SCON.1	แฟล็กแสดงการอินเตอร์รัพต์ภายหลังการส่งข้อมูล
RI	SCON.0	แฟล็กแสดงการอินเตอร์รัพต์เมื่อมีข้อมูลรับเข้า

รูปที่ 2.9 รีจีสเตอร์ควบคุมการทำงานและบอกสถานะการสื่อสารข้อมูลอนุกรม SCON

2.5.1 การอินเตอร์รัพต์ของการสื่อสารอนุกรม

เนื่องจากการส่งข้อมูลหรือการรับข้อมูลอนุกรมในการส่งข้อมูลไบต์หนึ่ง ๆ ค่อนข้างจะที่ใช้เวลานานหลายมิลลิวินาที ดังนั้นเพื่อให้การจัดการเกี่ยวกับการสื่อสารข้อมูลแบบนี้เป็นไปอย่างมีประสิทธิภาพ 8051 จึงได้กำหนดบิตหรือแฟล็กสถานะที่เกี่ยวข้องทั้งหมด จัดรวมอยู่ภายในรีจีสเตอร์ SCON เท่านั้น แฟล็ก TI ซึ่งจะมีค่าเป็น 1 เมื่อข้อมูลได้ทำการส่งออกไปภายนอกเสร็จสิ้นแล้ว และแฟล็ก RI ซึ่งจะมีค่าเป็น 1 จะมีผลทำให้ทราบว่าได้รับข้อมูลผ่านเข้ามาทางพอร์ตอนุกรม เมื่อแฟล็กตัวใดตัวหนึ่งนี้มีค่าเป็น 1 จะมีผลทำให้เกิดการอินเตอร์รัพต์ขึ้น ดังนั้นภายในโปรแกรมจะต้องทำการตรวจสอบจากสภาวะของแฟล็กเหล่านี้เองว่ามีการอินเตอร์รัพต์ขึ้นด้วยสาเหตุใด จากนั้นจึงคอยกำหนดค่า 0 ให้กับแฟล็กนั้น ลักษณะนี้จะมีความแตกต่างไปจากการอินเตอร์รัพต์จากสัญญาณอื่น ๆ เช่น วงจรนับ/จับเวลา เป็นต้น ซึ่งจะมีการกำหนดค่า 0 ให้กับแฟล็กสถานะที่เกี่ยวข้องโดยอัตโนมัติ ภายหลังจากที่ได้เข้าไปทำงานยังส่วนของโปรแกรมย่อยบริการอินเตอร์รัพต์ ดังนั้นจึงสังเกตความแตกต่างในส่วนนี้ไว้ด้วยและเมื่อเกิดการอินเตอร์รัพต์ในระบบไมโครคอนโทรลเลอร์ไม่ว่าจะเป็นการอินเตอร์รัพต์จากฮาร์ดแวร์ หรือ ซอฟต์แวร์ จะยังไม่บริการจนกว่าซีพียูจะเอ็กซีคิวคำสั่งปัจจุบันในขณะนั้นเสร็จสมบูรณ์

2.5.2 กระบวนการรับและส่งข้อมูลอนุกรมของไมโครคอนโทรลเลอร์ 8051

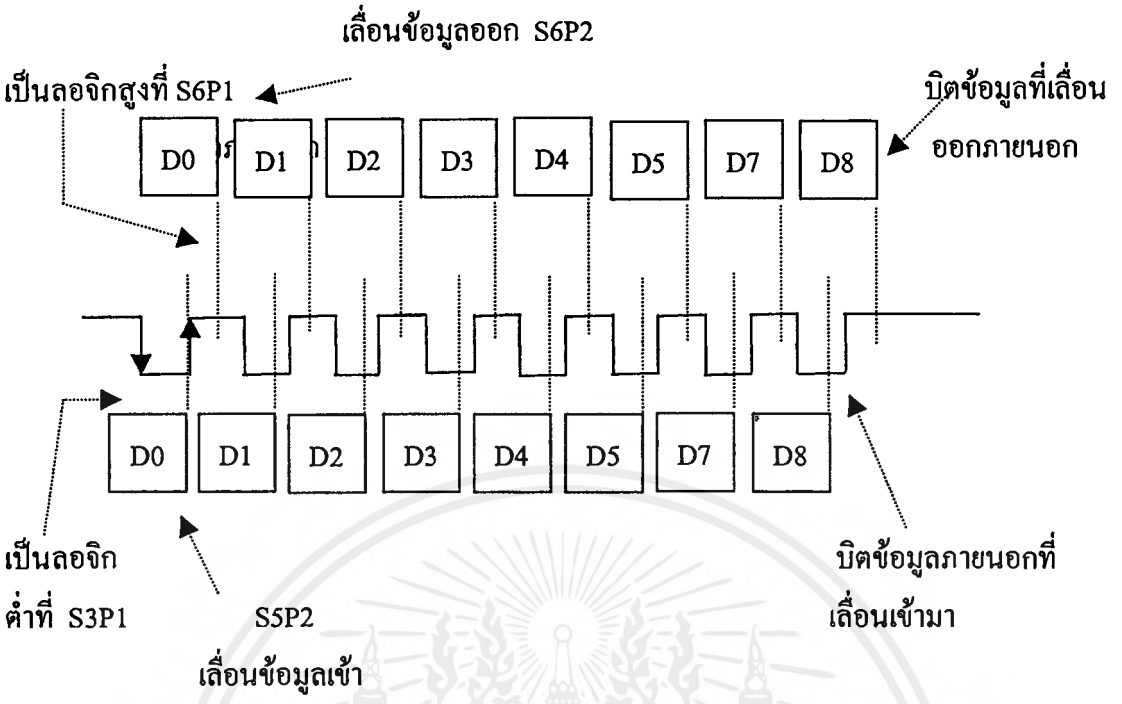
การส่งข้อมูลออกทางพอร์ตอนุกรมของ 8051 จะเริ่มต้นขึ้น ภายหลังเมื่อมีการเขียนข้อมูลลงในรีจีสเตอร์ SBUF ข้อมูลนี้จะถูกจัดการด้วยวิธีการทางด้านฮาร์ดแวร์ ในการเลื่อนบิต และส่งสัญญาณออกไปภายนอกโดยอัตโนมัติ เมื่อข้อมูลเหล่านี้ได้ส่งออกครบถ้วนแล้ว จึงจะทำการกำหนดค่าของแฟล็ก TI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าขณะนี้รีจีสเตอร์ SBUF ว่าง และพร้อมที่จะส่งข้อมูลไบต์ต่อไปแล้ว ในกรณีที่ผู้ใช้เขียนข้อมูลใหม่ลงในรีจีสเตอร์ SBUF โดยไม่รอให้แฟล็ก TI มีค่าเป็น 1 ก่อนจะมีผลทำให้ข้อมูลที่ส่งออกไปผิดพลาดได้

สำหรับการรับข้อมูลจากพอร์ตอนุกรมจะต้องเริ่มต้นโดยการกำหนดค่าบิต REN (Reciever Enable) ให้มีค่าเป็น 1 ก่อน หลังจากนั้นเมื่อมีบิตของข้อมูลถูกส่งเข้ามาจากภายนอกระบบฮาร์ดแวร์ของ 8051 จึงจะทำการเลื่อนบิตเหล่านี้เข้ามาโดยอัตโนมัติ และเมื่อบิตสุดท้ายถูกเลื่อนเข้ามาเรียบร้อยแล้ว ข้อมูลนั้นจะถูกย้ายมาเก็บไว้ยังรีจีสเตอร์ SBUF และทำการกำหนดให้แฟล็ก RI ให้มีค่าเป็น 1 ซึ่งมีผลทำให้เกิดการอินเตอร์รัพต์โปรแกรมขึ้น

2.5.3 พอร์ตอนุกรมโหมด 0

การทำงานของพอร์ตอนุกรมในโหมด 0 เป็นการขยายพอร์ตอินพุตหรือพอร์ตเอาต์พุตให้มีจำนวนมาก โดยจะทำการสร้างสัญญาณนาฬิกาขึ้นเพื่อให้เป็นจังหวะในการทำงานที่พร้อมกัน (Synchronizing) สำหรับการเลื่อนบิตเข้าหรือออกจากไอซีรีจีสเตอร์ภายนอก เมื่อมีการโอนย้ายข้อมูลมายังรีจีสเตอร์ในแต่ละครั้งก็จะมีผลให้เกิดการส่งบิตข้อมูลทั้ง 8 บิต ออกมา แม้ว่าแฟล็กสถานะ TI จะยังคงมีค่าเป็น 1 อยู่ก็ตาม นอกจากนี้แล้วเมื่อใดก็ตามที่ค่าของแฟล็กสถานะ RI เป็นค่า 1 ก็ควรที่ย้ายข้อมูลที่รับเข้ามานั้นออกไปจากรีจีสเตอร์ SBUF เสียก่อนที่จะได้มีการกำหนดค่าแฟล็ก RI ให้เป็น 0 เพื่อรับข้อมูลใหม่ต่อไป

การทำงานของพอร์ตอนุกรมในโหมด 0 เป็นการรับ และส่งอนุกรมจำนวน 8 บิต โดยใช้เพียงขาสัญญาณ RxD เท่านั้น ส่วนขาสัญญาณ TxD จะนำไปใช้เพื่อเป็นขาสัญญาณนาฬิกาในการให้จังหวะการเลื่อนข้อมูลกับวงจรเลื่อนบิตภายนอก สำหรับอัตราการเลื่อนบิต (หรืออัตราบอด) จะถูกกำหนดไว้คงที่ที่ค่า 1/12 ของความถี่ออสซิลเลเตอร์ จากรูปที่ 2.10 แสดงให้เห็นถึงแผนภาพเวลาสัญญาณต่าง ๆ ในโหมด 0 เมื่อมีการรับ และส่งข้อมูล 1 ไบต์ โดยสัญญาณนาฬิกาในการเลื่อนบิตนี้จะเกิดขึ้นภายในตัวของ 8051 เอง และมีจุดประสงค์เพื่อนำไปใช้งานสำหรับวงจรชิพรีจีสเตอร์ภายนอกเท่านั้น



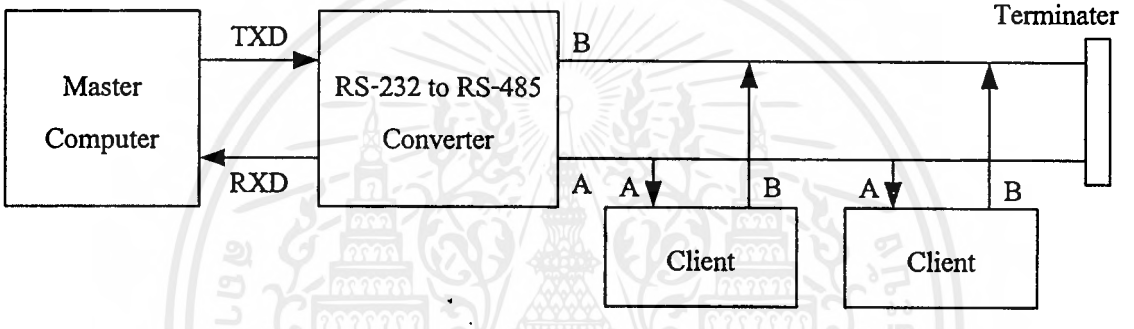
รูปที่ 2.10 แผนภาพเวลาของสัญญาณอนุกรมโหมด 0

สัญญาณนาฬิกาที่สร้างขึ้นทางขาสัญญาณ TxD นี้จะสลับค่าไปมาจากระดับลอจิกสูงไปหาต่ำในราวช่วงใกล้เคียงกับเวลาขอบขาลงของสัญญาณ ALE ซึ่งอยู่ในคาบเวลาออสซิลเลเตอร์ที่ 15 ภายหลังจากที่ได้ทำคำสั่งการโอนย้ายข้อมูลมายังรีจีสเตอร์ SBUF หรือคำสั่งที่ทำให้ค่าแฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้นสัญญาณนาฬิกานี้ก็จะเปลี่ยนแปลงอีกครั้งราวช่วงใกล้เคียงกับเวลาขอบขาลงสัญญาณ ALE ในคาบเวลาออสซิลเลเตอร์หลังจากนั้นอีก 6 คาบ และจะดำเนินไปในลักษณะเช่นนี้จนกระทั่งข้อมูล 8 บิต ได้ถูกส่งออกไปเรียบร้อยแล้ว เมื่อสัญญาณของขาขึ้นของสัญญาณนาฬิกานี้เกิดขึ้นครบจำนวน 8 ครั้งแล้ว จึงจะมีผลทำให้แฟล็กสถานะ TI หรือ RI มีค่าเป็น 1 ขึ้น และสถานะของขาสัญญาณ TxD ก็จะเป็นระดับลอจิกสูงไปโดยตลอด

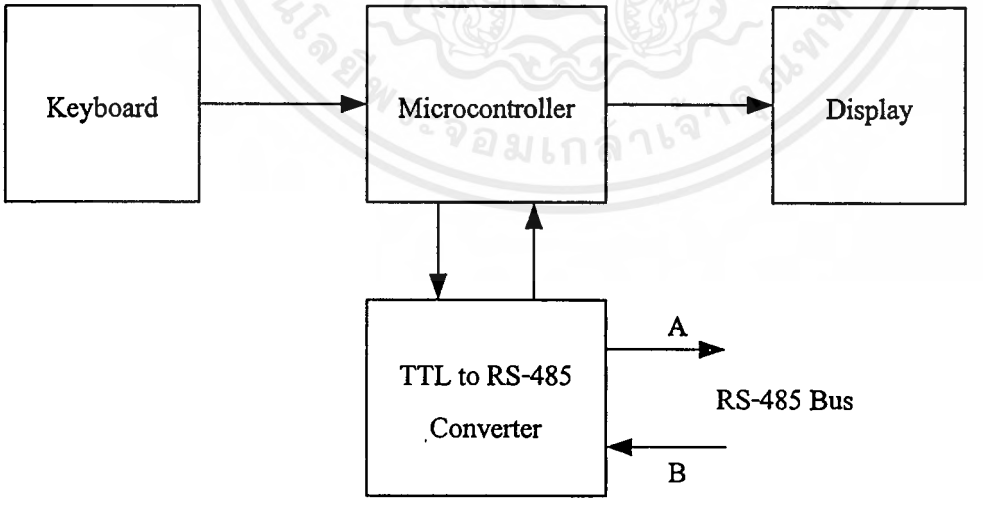
ข้อมูลที่จะส่งไปภายนอกจะถูกเลื่อนบิตนัยสำคัญต่ำออกไปก่อน และเป็นลำดับแรก โดยจะเริ่มต้นขึ้นเวลาเริ่มต้นของคาบเวลาออสซิลเลเตอร์ ภายหลังจากที่ได้รับคำสั่งโอนย้ายข้อมูลมายังรีจีสเตอร์ SBUF สำหรับบิตแรกของข้อมูลที่รับเข้ามานั้น จะถูกค้างข้อมูลไว้ด้วยขอบขาขึ้นของสัญญาณนาฬิกาในคาบออสซิลเลเตอร์ ที่ 24 ภายหลังจากที่มีการกำหนดให้แฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้น ในเวลาคาบเวลาออสซิลเลเตอร์ อีก 12 คาบถัดมาก็จะได้รับบิตถัดไป ซึ่งจะดำเนินการในลักษณะเช่นนี้จนกระทั่งได้รับบิตข้อมูลครบทั้ง 8 บิต

บทที่ 3
การออกแบบ การสร้าง และการทำงาน

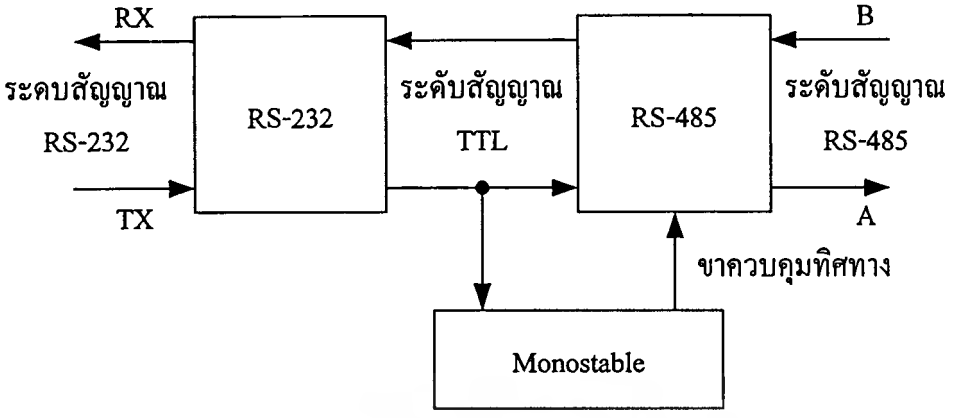
ในบทนี้จะเป็นการนำเอาวิธีการออกแบบ การสร้าง และการทำงานของวงจรส่วนต่าง ๆ ของโครงงานนี้ ซึ่งองค์ประกอบของตัวโครงงานสามารถแยกได้เป็น 2 ส่วนใหญ่ ๆ คือ ชุดรับส่งคำตอบ และชุดแปลงข้อมูล RS-485 เป็น RS-232 เพื่อต่อเข้ากับพอร์ตอนุกรมของเครื่องคอมพิวเตอร์



รูปที่ 3.1 การต่อในระบบบัส



รูปที่ 3.2 บล็อกไดอะแกรมชุดตอบคำถาม



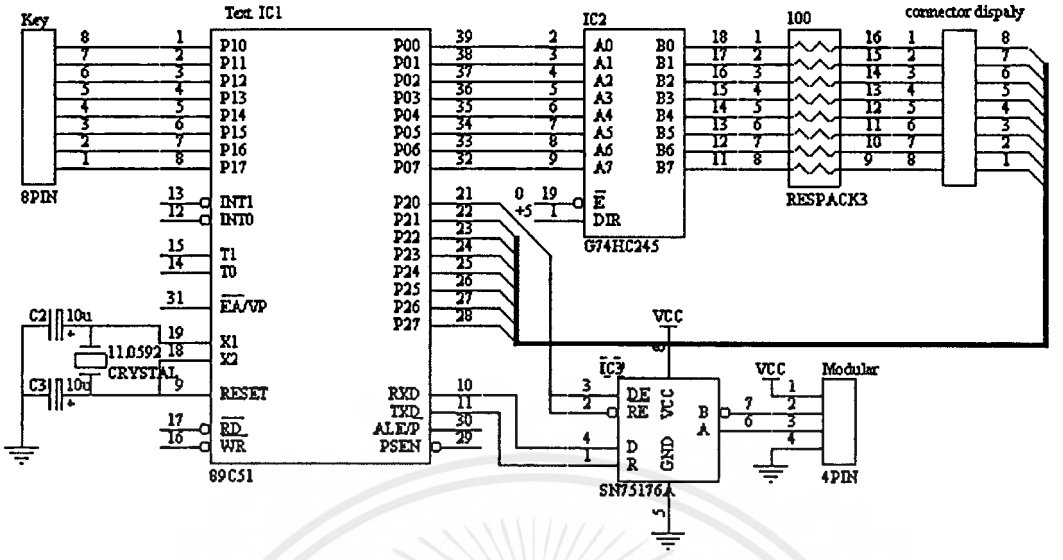
รูปที่ 3.3 บล็อกไดอะแกรมชุดเปลี่ยนระดับสัญญาณ จาก RS-232/RS-485

3.1 การออกแบบชุดรับส่ง

ข้อมูลที่จะส่งไปภายนอกนั้นจะถูกเลื่อนบิตนัยสำคัญต่ำออกไปก่อนเป็นลำดับแรก โดยจะเริ่มขึ้นในเวลาเริ่มต้นของคาบเวลาออสซิลเลเตอร์ ภายหลังจากที่ได้ทำคำสั่งการโอนย้ายข้อมูลมายังรีจิสเตอร์ SBUF สำหรับบิตแรกของข้อมูลที่รับเข้ามานั้นจะถูกค้างข้อมูลไว้ด้วยขอบขาขึ้นของสัญญาณนาฬิกาในคาบสัญญาณออสซิลเลเตอร์ที่ 24 ภายหลังจากที่มีการกำหนดให้มีแฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้นในเวลาคาบเวลาออสซิลเลเตอร์อีก 12 คาบถัดมาก็จะได้รับบิตต่อไป ซึ่งจะดำเนินการในลักษณะเช่นนี้จนกระทั่งได้จำนวนบิตข้อมูลครบทั้ง 8 บิต คำตอบ

3.1.1 คุณสมบัติของของตัวควบคุม

ไมโครคอนโทรลเลอร์	: 89C51
หน่วยความจำ	: ROM ภายใน ขนาด 4 กิโลไบต์
	RAM ภายใน ขนาด 128 ไบต์
พอร์ตอินพุต	: ขนาด 8 บิต คือ P1 ของ 89C51,ขา RX
พอร์ตเอาต์พุต	: ขนาด 16 บิต คือ P0, P2 ของ 89C51,ขา TX
คีย์บอร์ด	: แบบ เมตริกซ์ 3 x 4 ขนาด 12 คีย์
จอแสดงผล	: ใช้ 7 SEGMENT 5 หลัก
สัญญาณนาฬิกา	: ใช้คริสตอลความถี่ 11.0592 MHz
การส่งข้อมูลอนุกรม	: ใช้มาตรฐาน RS 485 โดย IC 75176

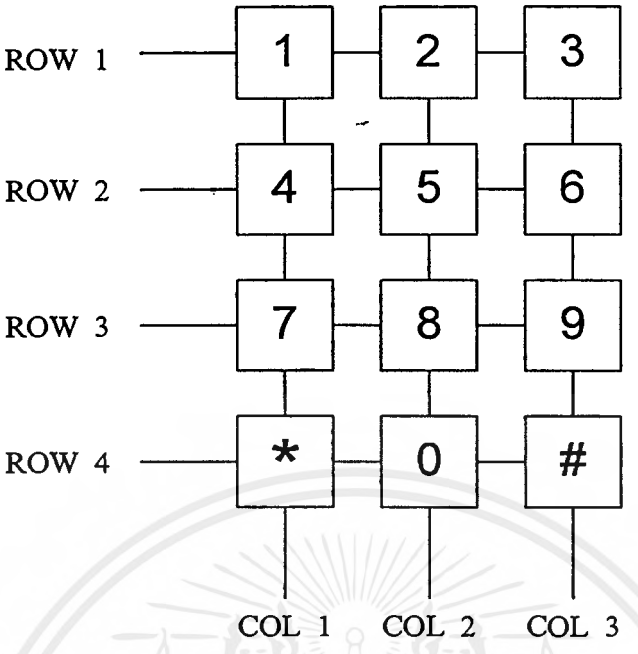


รูปที่ 3.4 วงจรชุดส่งข้อมูล

3.1.2 การเชื่อมต่อคีย์บอร์ด

จากรูปเป็นวงจรของคีย์บอร์ดขนาด 3 x 4 คือมี ROW ที่ใช้เป็นด้าน INPUT (อ่านข้อมูลของคีย์บอร์ด) 3 เส้น ซึ่งจำเป็นต้องมี R - PULL UP ไว้ เมื่อยังไม่ได้กดคีย์จะได้มีสถานะคงที่อยู่ค่า ๆ หนึ่ง เพราะเราจะ scan logic 0 ทางด้านคอลัมน์ N หลักการก็คือ ส่งข้อมูลลอจิก 0 ให้กับคอลัมน์แรก และอ่านข้อมูลทางด้านแถว ดูว่าเส้นใดเป็น 0 บ้าง ถ้ามีแสดงว่ามีการกดคีย์ ก็ไปทำการตรวจเช็คว่าเป็นแถวใด และคอลัมน์ใด เพื่อเข้ารหัสคีย์แล้วแปลงเป็นค่าที่ต้องการไปใช้งาน แต่ถ้าอ่านแล้วไม่มีเส้นใดในแถวเป็น 0 ก็จะทำงานคอลัมน์ถัดไปให้เป็น 0 แล้วก็อ่านแถวใหม่ทำอย่างนี้จนกว่าจะหมดคอลัมน์

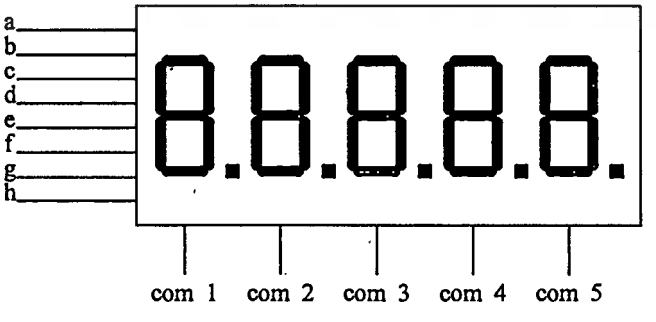
การให้ค่ารหัสของคีย์ลงในหน่วยความจำสำหรับเก็บค่ารหัสคีย์นั้น สามารถทำได้โดยตรวจดูแถวที่ให้ค่าเป็น “0” และให้ค่าประจำแถวเอาไว้ และเมื่อคอลัมน์ใดมีการกดคีย์จะทำให้ระดับเป็น “0” ให้ค่าประจำหลักของคอลัมน์นั้นจากนั้นจึงนำมารวมกันกับค่าประจำแถวโดยให้ค่าประจำแถวเป็นสี่บิตบนและค่าประจำหลักเป็นสี่บิตล่างก็จะได้ค่าที่เป็นค่ารหัสแสกนของคีย์ โดยที่ค่าที่เป็นค่ารหัสแสกนของคีย์นี้เป็นค่าที่ยังไม่สามารถนำไปใช้ได้ทันทีเนื่องจากเป็นค่าที่เกิดจากการให้ค่าตัวเลขทางแถวและหลักมารวมกันเฉยๆ โดยให้ทางแถวเป็นส่วนของสี่บิตสูง และทางคอลัมน์เป็นส่วนของสี่บิตต่ำ ต้องทำการแปลงโดยการเปรียบเทียบค่ารหัสแสกนว่าเป็นของคีย์ใดแล้วจึงให้ค่าที่เป็นค่าของการกดของคีย์นั้นๆ จริงๆ ได้



รูปที่ 3.5 คีย์บอร์ดแบบเมตริกซ์ 3 x 4 จุด

3.1.3 การเชื่อมต่อเข้ากับจอแสดงผล

จากรูปเป็นวงจรในส่วนของ 7 segment จำนวน 5 หลัก เป็นชนิดคอมมอนคาโธดทางด้านอาโนดของแต่ละ segment จะต่อกับพอร์ต 0 ซึ่งต้องมี buffer เพื่อขับ segment เพราะพอร์ต 0 ของ 89C51 เป็นลักษณะของ Drain เปิด และไม่มีค่าความต้านทานสำหรับดึงระดับสัญญาณจึงไม่สามารถทำให้ segment ติดได้ ดังนั้นจำเป็นต้องต่อความต้านทานสำหรับดึงระดับสัญญาณให้แก่ segment ด้วย การต่อ คอมมอนเลือกหลักเราจะต่อเข้ากับพอร์ต P2 การทำงานของ segment จะต้องส่งลอจิก 0 ให้ทางด้าน P0 และต้องทำการ decode ตัวเลขด้วย



รูปที่ 3.6 การต่อจอแสดงผลแบบ 7 segment จำนวน 5 หลัก

3.1.4 การส่งข้อมูลอนุกรมตามมาตรฐาน RS-485

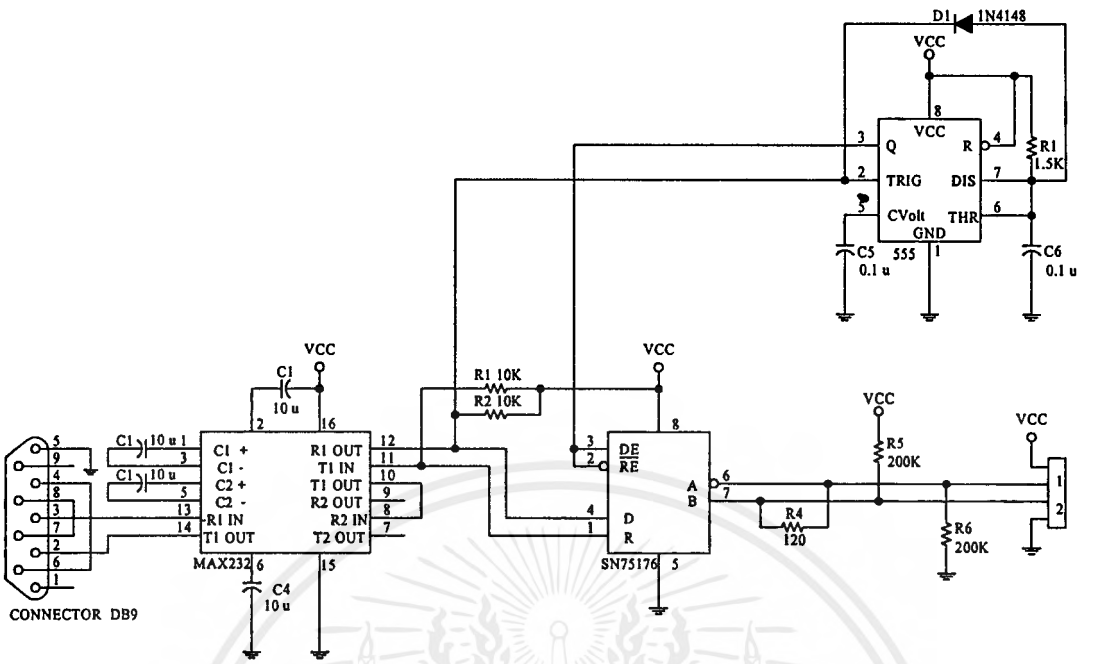
จากวงจรเราต้องใช้ ไอซี 75176 ซึ่งสามารถต่อเข้ากับขา TX, RX ของ 89C51 เพื่อต้องการยกระดับสัญญาณของการสื่อสารให้มากขึ้น และสามารถส่งได้ไกล และอัตราการส่งข้อมูลมากขึ้น สำหรับไอซี 75176 จะมีขาสัญญาณบอกว่าอยู่ในสถานะส่งหรือรับ

3.1.5 วงจรแปลงสัญญาณจากมาตรฐาน RS-232 เป็น RS-485

วงจรเปลี่ยนสัญญาณการส่งข้อมูลจากระบบ RS-232 ให้เป็นการส่งข้อมูลแบบ RS-485 (2 เส้น) ซึ่งจะทำการส่งข้องมูลแบบ RS 485 นี้ จะได้ไกลขึ้น (4,000 ฟุต) และเป็นการต่อแบบ Multipoint คือ สามารถต่อพร้อม ๆ กัน ในการติดต่อได้หลายจุด (32 จุด) โดยจะต้องทำการกำหนดให้รับส่งไม่พร้อมกันโดยขณะที่ตัวใดตัวหนึ่งส่งข้อมูลใน Line อยู่ตัวอื่น ๆ จะต้องกำหนดตัวเองให้รับข้อมูลหรือ Enable 3 state ทั้งหมด ซึ่งอาจจะกำหนดมาตรฐานในการรับการส่งขึ้นเอง เช่น ให้ตัวเครื่อง PC เป็นตัวแม่ และตัวบอร์ด control ต่าง ๆ เป็นตัวลูกก็อาจจะกำหนดข้อตกลงไว้ว่า ตัวแม่จะต้องส่งข้อมูลเป็นเบอร์ของตัวลูกไปก่อน และถ้าตัวลูกตัวไหนรับข้อมูลแล้วเป็นเบอร์ของตัวเองก็ให้ส่งข้อมูลกลับมายังตัวแม่ได้ ส่วนตัวอื่น ๆ ก็หยุดส่ง

ในวงจรจะประกอบไปด้วยส่วนแปลงระดับสัญญาณจาก RS-232 ซึ่งเป็น ±12V ให้เป็น 0, 5 V ซึ่งเป็นระดับ TTL เพื่อส่งผ่านให้กับไอซีเบอร์ 75176 โดยขา TX จะต้องเข้ากับขา 2 ของไอซีเบอร์ 555 ทำหน้าที่เป็น สัญญาณกระตุ้นวงจร Mono statble โดยอัตราบิตต่อวินาทีจะขึ้นอยู่กับค่า R ที่ต่อจากขา 8 กับขา 6 ของ IC 555 ซึ่งเอาต์พุตขา 3 ของ ไอซีเบอร์ 555 จะนำไปควบคุมขา 2, 3 ของไอซีเบอร์ 75176 ซึ่งเป็นขาควบคุมการที่จะให้อยู่ในสภาวะรับหรือส่งข้อมูล

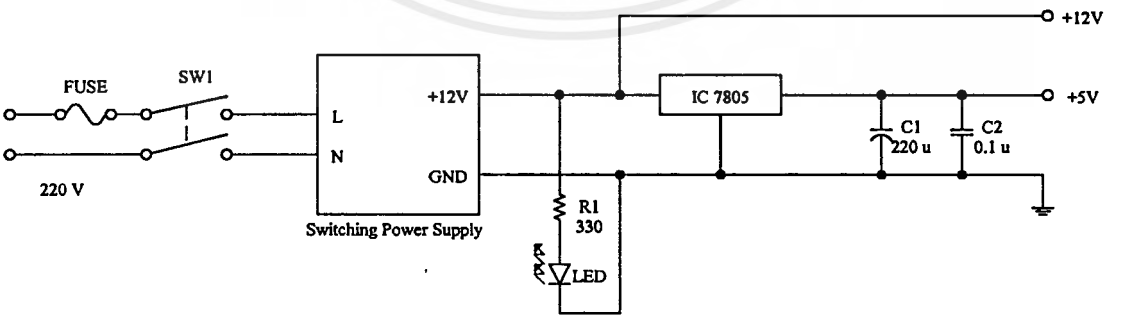
รายละเอียดการทำงานของวงจรมีดังนี้ เมื่อข้อมูลจากขาTX ของพอร์ตอนุกรมของเครื่องคอมพิวเตอร์มีการส่งข้อมูลออกมาเข้ามาที่ไอซี MAX232 ไอซี MAX232 ก็จะทำการเปลี่ยนระดับสัญญาณจาก RS-232 ให้เป็นระดับสัญญาณ TTL และข้อมูลจะถูกส่งต่อไปยังไอซี 75176 ส่วนหนึ่งจะถูกส่งต่อไปยังไอซี 555 เพื่อใช้เป็นตัวกระตุ้นอัตราบิตที่ต้องการ จากนั้นข้อมูลก็จะถูกแปลงจากระดับ TTL ไปเป็นระดับ RS-485 ดังรูปที่ 3.7 สำหรับ R_4 ค่า 120 โอห์ม นั้นต่อไว้เพื่อเป็นความต้านทานสำหรับปรับระดับความต้านทานทางด้านไฟกระแสสลับ และยังต้องมีความต้านทานค่าเดียวกันนี้ต่ออยู่ที่ปลายของสายสัญญาณเช่นกัน ส่วนค่า R_5 และ R_6 ค่า 200 กิโลโอห์ม ต่อไว้เพื่อปรับระดับของสัญญาณให้มีค่าใกล้เคียงกับระดับสูงสุดของระดับสัญญาณที่ควรจะเป็น เพื่อเป็นการป้องกันสัญญาณรบกวนที่อาจจะเกิดขึ้นในสายนำสัญญาณ



รูปที่ 3.7 วงจรเปลี่ยนระดับสัญญาณ RS-232 เป็น RS-485

3.1.6 การออกแบบแหล่งจ่ายไฟ

จากรูป 3.8 เป็นการนำชุดจ่ายไฟแบบสวิตชิ่ง แปลงจาก 220 V เป็น 12V จากนั้นเพื่อให้เหมาะสมสำหรับการทำงานของชุดตอบคำถามจำเป็นต้องมีการรักษาระดับแรงดันให้คงที่ด้วย ไอซีรักษาระดับแรงดัน(LM 7805) จากนั้นกรองด้วยตัวเก็บประจุค่า 220 µF และ 0.1 µF จึงสามารถจ่ายไฟไปเลี้ยงส่วนต่างๆวงจรต่อไป



รูปที่ 3.8 วงจรภาคจ่ายไฟ

3.2 การออกแบบในส่วนของโปรแกรม

3.2.1 โปรแกรมของชุดตอบคำถาม

จากผังการทำงานในรูปที่ ง.1 โปรแกรมจะคอยรอรับข้อมูลจากเครื่องคอมพิวเตอร์มาทำการวิเคราะห์ ถ้ามีการอ้างอิงถึงตัวใดตัวหนึ่งซึ่งสามารถตรวจได้จากโปรโตคอลการรับซึ่งมีรายละเอียดดังนี้

;	X	Y	Z	:

; ใช้แทนรหัสเริ่มต้นชุดข้อมูล

X ใช้แทนหมายเลขของชุดตอบคำถาม มีหมายเลข 1,2,3 และ 4 (ซึ่งเครื่องต้นแบบทั้งหมด มีจำนวน 4 เครื่อง โดยที่ X แทนด้วยหมายเลขดังกล่าว จำนวนสูงสุด 256 ชุด)

Y ใช้แทนรหัสคำสั่งซึ่งมีรายละเอียดดังนี้

รหัส 0 เป็นคำสั่งลบข้อมูลเก่า

รหัส 1 เป็นคำสั่งรับข้อมูลจากปุ่มกดของชุดตอบคำถาม

รหัส 2 เป็นคำสั่งส่งข้อมูลกลับไปยังเครื่องคอมพิวเตอร์

Z ใช้แทนข้อมูลของคำสั่งในการทำงานของชุดตอบคำถาม แต่ขณะนี้ยังไม่ใช้งาน

: ใช้แทนรหัสจบชุดข้อมูล

อักขระที่ใช้ในการสื่อสารทั้งหมดอยู่ในรูปแบบของรหัสแอสกี้

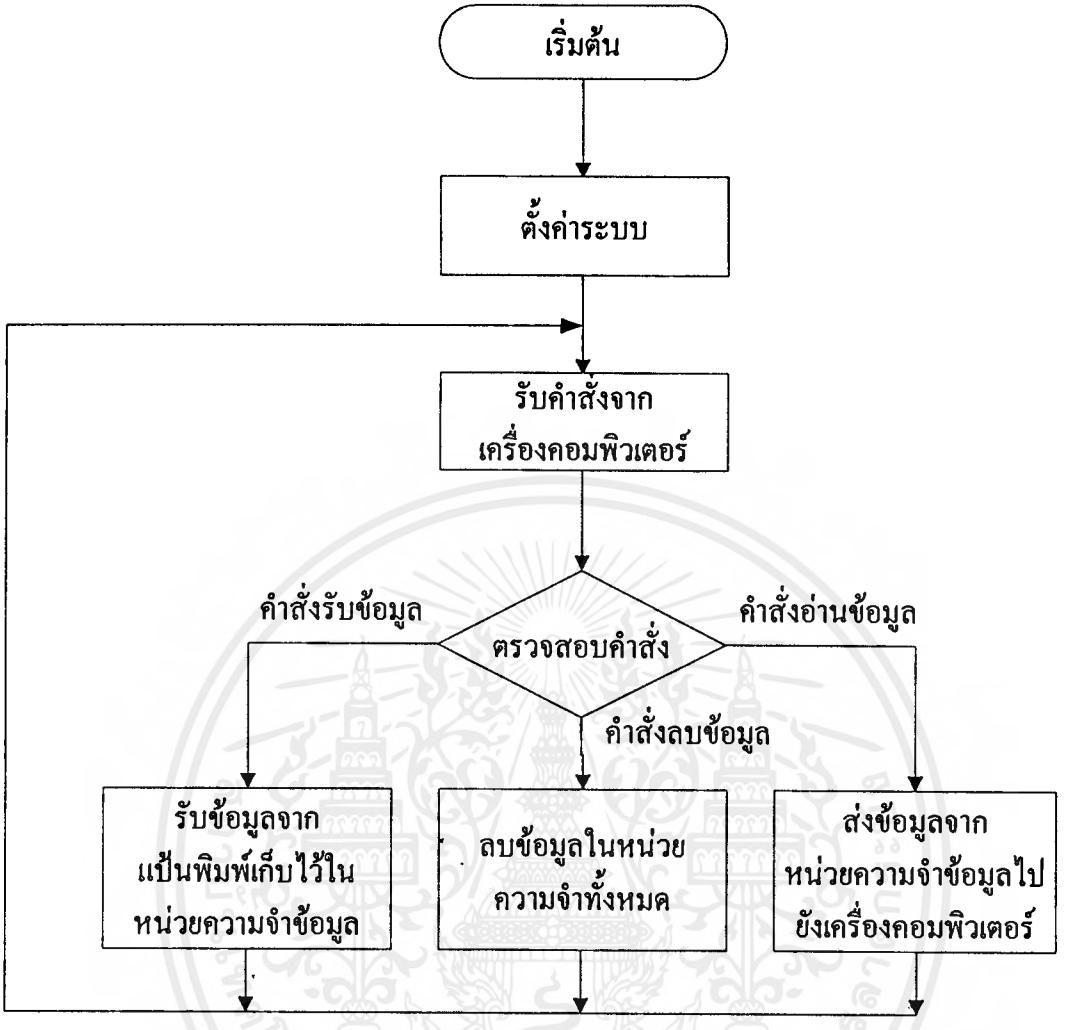
การตรวจสอบคำสั่งสามารถตรวจสอบได้ทันที ตัวอย่างเช่น ชุดของโปรโตคอลดังต่อไปนี้

;100: และชุดตอบคำถามนี้มีรหัสประจำชุดเป็น 1 ก็จะทำการลบข้อมูลเก่า

;110: และชุดตอบคำถามนี้มีรหัสประจำชุดเป็น 1 ก็จะทำการรับข้อมูลจากปุ่มกด

;220: และชุดตอบคำถามนี้มีรหัสประจำชุดเป็น 2 ก็จะทำการส่งข้อมูลไปยังเครื่องคอมพิวเตอร์

ชุดของข้อมูลที่ส่งออกไปนั้นจะทำการส่งออกไปเป็นลำดับเรียงกันโดยเริ่มจาก ไบต์เริ่มต้นชุดข้อมูลจะถูกส่งไปก่อน จากนั้นจึงตามด้วยไบต์หมายเลขของชุดตอบคำถาม ต่อมาด้วยไบต์รหัสคำสั่ง และต่อมาอีกด้วยไบต์ข้อมูลของคำสั่ง ท้ายที่สุดก็จะเป็นไบต์จบชุดข้อมูลจะเป็นการส่งข้อมูลคำสั่งไปยังชุดตอบคำถามเพื่อให้ทำงานตามต้องการ



รูปที่ 3.9 ผังการทำงานหลักของโปรแกรมบนชุดตอบคำถาม

3.2.2 การทำงานของโปรแกรมบนเครื่องคอมพิวเตอร์

เมื่อเปิดโปรแกรม Control Center ขึ้นมาโปรแกรมจะรอรับการเลือกใช้งานพอร์ตอนุกรม ซึ่งมีให้เลือกอยู่ 2 พอร์ตด้วยกัน คือ COM1 และ COM2 เมื่อเลือกพอร์ตที่ทำการสื่อสารได้แล้ว โปรแกรมจะเข้าสู่สภาวะรอรับคำสั่งที่จะใช้สั่งงานชุดตอบคำถามมีหัวข้อคำสั่งงานต่างๆ ดังต่อไปนี้

1. Clear all child buffers ทำการลบข้อมูลของชุดตอบคำถามทุกชุด
2. Input all child ทำการรับข้อมูลจากปุ่มกดของชุดรับข้อมูลทุกชุด
3. Read all child ทำการอ่านข้อมูลจากชุดตอบคำถามขึ้นมาแสดงผลทุกชุด
4. Clear one child buffers ทำการลบข้อมูลของชุดตอบคำถามหนึ่งชุด
5. Input one child ทำการรับข้อมูลจากปุ่มกดของชุดรับข้อมูลหนึ่งชุด
6. Read one child ทำการอ่านข้อมูลจากชุดตอบคำถามขึ้นมาแสดงผลหนึ่งชุด
7. Write data to file ทำการบันทึกข้อมูลที่อ่านได้ลงในแฟ้มข้อมูล

ข้อมูลที่ชุดตอบคำถามส่งกลับเข้ามาจะใช้โปรโตคอลต่างกับการส่งข้อมูลคำสั่งไปยังชุดตอบคำถาม มีรายละเอียด ดังต่อไปนี้

;	X	6	d_1	d_2	d_3	d_4	d_5	d_6	S	:

; ใช้แทนรหัสเริ่มต้นชุดข้อมูล

X ใช้แทนหมายเลขของชุดตอบคำถาม มีหมายเลข 1,2,3 และ 4 (ซึ่งเครื่องต้นแบบทั้งหมด มีจำนวน 4 เครื่อง โดยที่ X แทนด้วยหมายเลขดังกล่าว)

6 เป็นจำนวนข้อมูลที่ชุดตอบคำถามส่งกลับมา

d_1 ผลลัพธ์ที่เกิดจากการกดแป้นพิมพ์หลักที่ 1 มีค่าตั้งแต่ “0” ถึง “9”

d_2 ผลลัพธ์ที่เกิดจากการกดแป้นพิมพ์หลักที่ 2 มีค่าตั้งแต่ “0” ถึง “9”

d_3 ผลลัพธ์ที่เกิดจากการกดแป้นพิมพ์หลักที่ 3 มีค่าตั้งแต่ “0” ถึง “9”

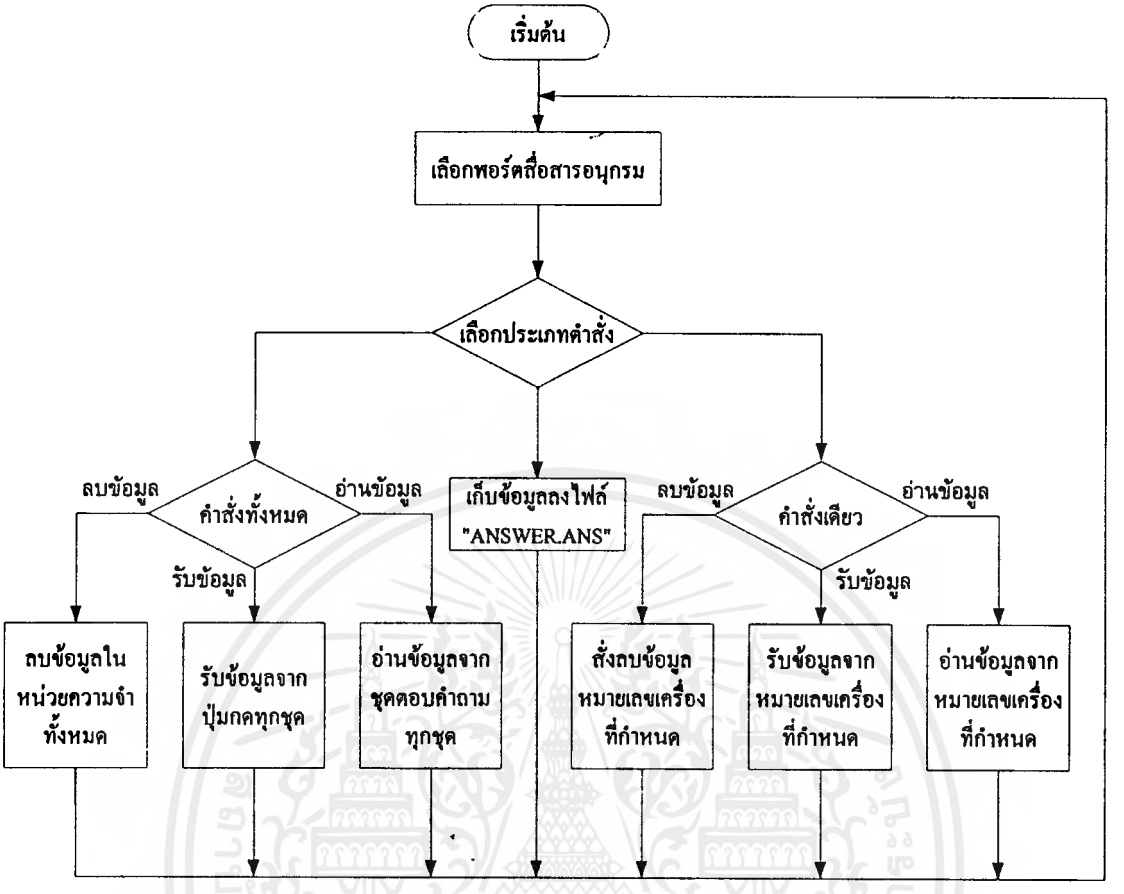
d_4 ผลลัพธ์ที่เกิดจากการกดแป้นพิมพ์หลักที่ 4 มีค่าตั้งแต่ “0” ถึง “9”

d_5 ผลลัพธ์ที่เกิดจากการกดแป้นพิมพ์หลักที่ 5 มีค่าตั้งแต่ “0” ถึง “9”

d_6 ผลลัพธ์ที่เกิดจากการกดแป้นพิมพ์หลักที่ 6 มีค่าตั้งแต่ “0” ถึง “9”

S เป็นผลรวมของข้อมูลของหลักที่ 1 ถึง 6 ใช้สำหรับตรวจสอบความผิดพลาด

: ใช้แทนรหัสจบชุดข้อมูล



รูปที่ 3.10 ผังการทำงานของโปรแกรมบนเครื่องคอมพิวเตอร์

บทที่ 4

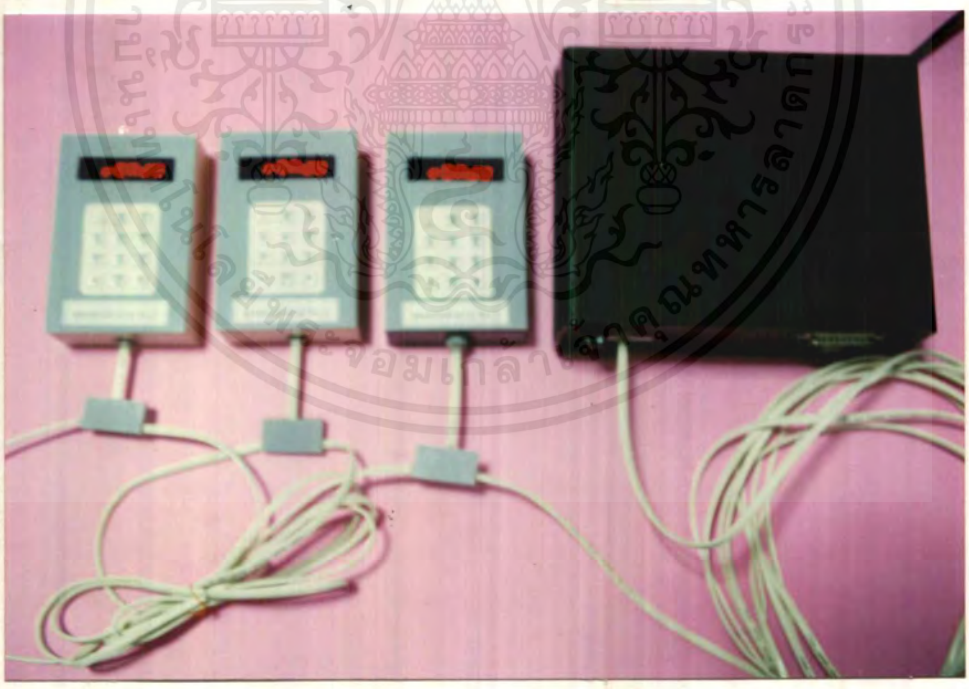
การทดลอง และผลการทดลอง

การทดลองการทำงานของชุดตอบคำถามโดยใช้ไมโครคอนโทรลเลอร์ และโปรแกรมเก็บข้อมูลคำตอบลงในเครื่องคอมพิวเตอร์ โดยเครื่องคอมพิวเตอร์จะควบคุมการทำงานของชุดรับส่งอีกทีหนึ่ง

4.1 การทดสอบตัวเครื่องเบื้องต้น

การทดสอบตัวเครื่องเบื้องต้นมีลำดับขั้นตอนดังต่อไปนี้

1. ต่อสายไฟเข้ากับชุดตอบคำถามและเครื่องแปลงสัญญาณ RS-232 เป็น RS-485 จากนั้นต่อสายสัญญาณ จากพอร์ตอนุกรมของเครื่องคอมพิวเตอร์ต่อเข้ากับเครื่องแปลงสัญญาณ RS-232 เป็น RS-485

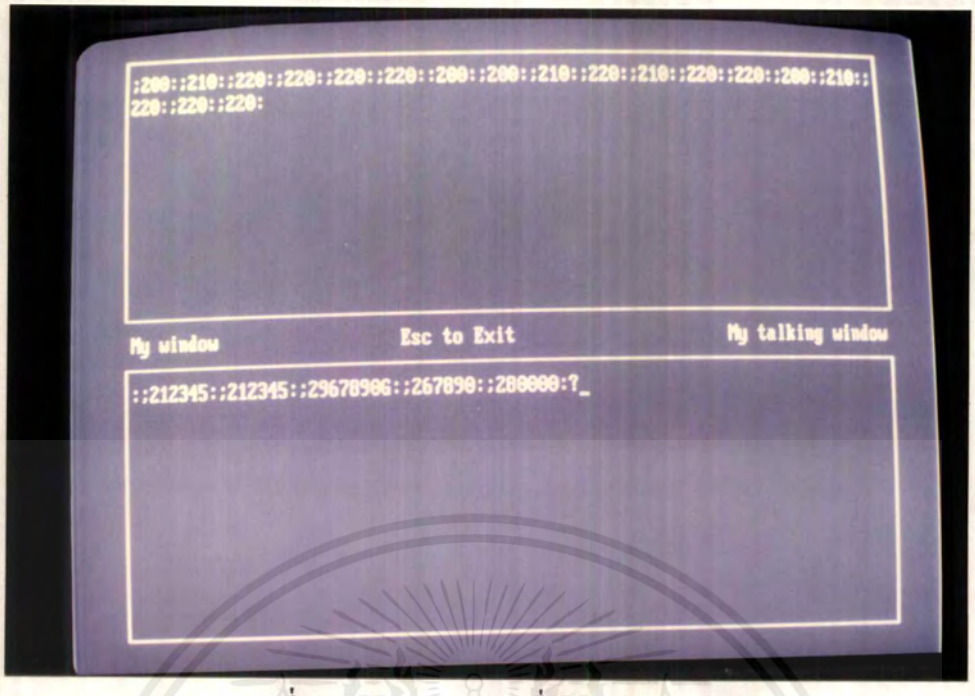


รูปที่ 4.1 การต่อชุดตอบคำถาม

2. เปิดสวิตซ์ ปิด/เปิด (ON/OFF) ป้อนไฟให้กับเครื่องแปลงสัญญาณ RS-232 เป็น RS-485 ซึ่งจะสามารถจ่ายไฟไปให้กับชุดตอบคำถามได้ด้วย และเปิดเครื่องคอมพิวเตอร์เพื่อเข้าสู่โปรแกรมรับข้อมูลคำตอบ
3. หลังจากนั้นหน้าจอของชุดตอบคำถามจะปรากฏข้อความ ready แสดงว่าเครื่องพร้อมที่จะทำงาน

รูปที่ 4.2 สภาวะที่ชุดตอบคำถามพร้อมที่จะทำงาน

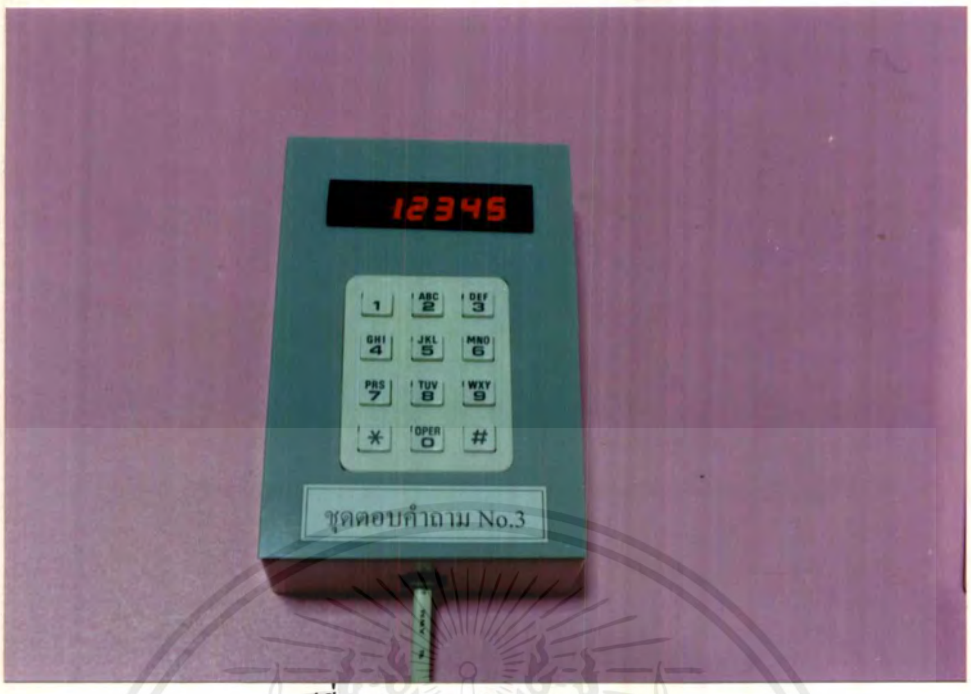
4. ทำการรันโปรแกรมติดต่อสื่อสารอนุกรม SE.EXE เพื่อ ทำการทดลองรับ-ส่งข้อมูล โดยกำหนด อัตราความเร็วจำนวนบิตต่อวินาที 9600,N,8,p จากนั้นพิมพ์ ;100: สังเกตว่าจะปรากฏข้อความ CLEAR ที่ ตัวลูกที่ 1 คำสั่งนี้เป็นการสั่งลบข้อมูลที่ได้จากการกด KEY ทำคำสั่งนี้ไปทุกตัวโดยการเปลี่ยน จาก ;100:_;200:_;300:…;n00:ไปจนครบจำนวนตัวลูก



รูปที่ 4.3 โปรแกรมติดต่อสื่อสารอนุกรม

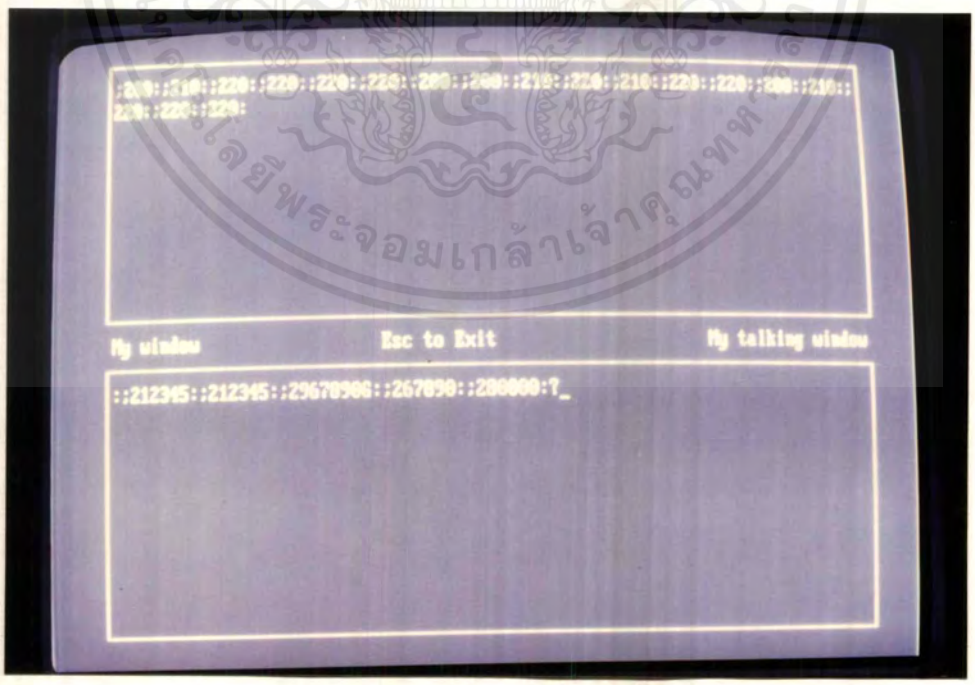
5. พิมพ์ ;110: สังเกตว่าจะปรากฏข้อความ INPUT ที่ตัวลูกที่ 1 คำสั่งนี้เป็นการสั่งอนุญาตการกด KEY ทำคำสั่งนี้ไปทุกตัวโดยการเปลี่ยน จาก ;110:_;210:_;310:...;n10:ไปจนครบ จะเห็นว่าทุกตัวจะพร้อมที่จะรับข้อมูล

รูปที่ 4.4 สภาวะที่เครื่องพร้อมรับคำตอบ



รูปที่ 4.5 ชุดตอบคำถามถูกกด

6. ทำการป้อนข้อมูลโดยการกด KEY ที่ตัวชุดตอบคำถามแล้วตามด้วยกด #
7. พิมพ์ ;120: สังเกตว่าจะปรากฏข้อความ SEND ที่ตัวลูกที่ 1 คำสั่งนี้เป็นการสั่งส่งข้อมูลจากการกด KEY ทำคำสั่งนี้ไปทุกตัวโดยการเปลี่ยน จาก ;120:_;220:_;320:...;n20: ไปจนครบจะเห็นว่าทุกตัวจะส่งข้อมูล



รูปที่ 4.6 สภาวะที่ชุดตอบคำถามส่งข้อมูลไปคอมพิวเตอร์

ข้อมูลที่ได้จากการส่งของชุดตอบคำถามจะมีรูปแบบดังนี้ ; ตามด้วยหมายเลขเครื่อง และจำนวนบิตที่ส่งมา จากนั้นอีก 5 บิตที่ส่งมาคือ ค่าที่กด และตามด้วยบิตที่ได้จากการบวกของบิตข้อมูล และตามด้วย : เช่น ;16XXXXXS:

4.2 สรุปผลการทดลอง

การทดลองและทดสอบชุดตอบคำถามที่สร้างขึ้นสำหรับการเรียนการสอนทางไกลชุดนี้ได้ทำการทดสอบโดยการทดลองการรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์และตัวชุดตอบคำถาม โดยการทดลองได้แบ่งออกเป็น

1. การสั่งให้ลบข้อมูลเดิมในหน่วยความจำสำหรับเก็บข้อมูล
   การสั่งให้ลบข้อมูลในหน่วยความจำสำหรับเก็บข้อมูลของชุดตอบคำถาม สั่งได้โดยการส่งโปรโตคอลที่มีรหัสและหมายเลขของชุดตอบคำถามไปยังชุดตอบคำถามในระบบ เมื่อชุดตอบคำถามชุดใดตรวจสอบได้ว่าเป็นการสั่งงานของตัวชุดนั้นๆ ก็จะทำการลบข้อมูลเดิมในหน่วยความจำทันทีและปรากฏข้อความ Clear ขึ้นที่ส่วนแสดงผล

2. การสั่งให้รับข้อมูลจากแป้นพิมพ์ของชุดตอบคำถาม
   การสั่งให้รับข้อมูลจากแป้นพิมพ์ของชุดตอบคำถาม สั่งโดยการส่งโปรโตคอลที่มีรหัสคำสั่งรับข้อมูลจากแป้นพิมพ์และหมายเลขชุดตอบคำถามที่ต้องการให้รับข้อมูล เมื่อชุดตอบคำถามตรวจสอบได้ว่าเป็นคำสั่งให้ชุดนั้นๆ รับข้อมูลก็จะทำการรับข้อมูลจากแป้นพิมพ์ที่กด โดยสามารถที่จะใส่ข้อมูลอะไรก็ได้สามารถที่จะแก้ไขข้อมูลได้ตามต้องการจนกว่าจะกดแป้นพิมพ์ “#” เมื่อกดแป้นพิมพ์นี้แล้วข้อมูลจะเก็บไว้ในหน่วยความจำและไม่สามารถจะแก้ไขใดๆ ได้อีก

3. การสั่งให้ชุดตอบคำถามส่งข้อมูลคำตอบกลับมายังเครื่องคอมพิวเตอร์
   การสั่งให้ชุดตอบคำถามส่งข้อมูลคำตอบกลับมายังเครื่องคอมพิวเตอร์สั่งโดย การส่งโปรโตคอลและรหัสคำสั่งการส่งข้อมูลกลับเมื่อชุดตอบคำถามชุดใดรับและตรวจสอบได้ว่าเป็นของชุดนั้น ก็จะทำการส่งข้อมูลกลับมายังเครื่องคอมพิวเตอร์ โดยจะส่งกลับมาในรูปของโปรโตคอลการส่งข้อมูลกลับ และในชุดของโปรโตคอลการส่งข้อมูลกลับก็จะมีข้อมูลอยู่ทั้งหมด 6 ไบต์ด้วยกัน

บทที่ 5
สรุป ปัญหา แนวทางแก้ไข และการพัฒนา

5.1 บทสรุป

ชุดตอบคำถามสำหรับการสอนทางไกลนี้ โดยใช้ไมโครคอนโทรลเลอร์เบอร์ 89C51 ที่ทำหน้าที่ในการนำค่าคำตอบจาก Key ที่ทำการกดส่งไปยังเครื่องคอมพิวเตอร์ ได้ถูกออกแบบให้มีจำนวนหลายชุดในการตอบคำถามในแต่ละเครื่อง และการติดต่อเป็นแบบการสื่อสารอนุกรม ซึ่งการติดต่อสื่อสารเป็นแบบบัส ได้ทำการแบ่งเป็น 2 ส่วน ส่วนแรก คือ ชุดตอบคำถามโดยใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมการทำงาน ส่วนที่ 2 คือ ชุดรับและบันทึกข้อมูลซึ่งส่วนนี้เป็นส่วนที่สำคัญในการควบคุมชุดตอบคำถามทั้งหมด

ชุดตอบคำถามนี้เมื่อมาทดลองใช้งาน ปรากฏว่าสามารถทำการรับ-ส่ง และบันทึกข้อมูลได้ แล้วสามารถนำมาแสดงผลได้อย่างถูกต้อง และมีประสิทธิภาพ

5.2 ปัญหาและแนวทางแก้ไข

1. ในการออกแบบช่วงแรกได้ใช้ไมโครคอนโทรลเลอร์ เบอร์ 89C2051 ในการสร้างได้ประสบปัญหา คือเนื่องจาก พอร์ตที่ทำการควบคุมการแสดงผลมีจำนวนไม่เพียงพอ
แนวทางการแก้ไข คือ ได้ทำการนำเอาไมโครคอนโทรลเลอร์ เบอร์ 89C51 ที่มีจำนวนพอร์ตมากพอในการควบคุมการแสดงผล

2. ในการทำการต่อบัสของพอร์ตอนุกรมนั้น ไม่สามารถใช้งานได้
แนวทางการแก้ไข คือ ได้ทำการต่อรีจิสเตอร์ค่า 120 โอห์ม ปิดที่ปลายของสายสัญญาณนั้น ๆ

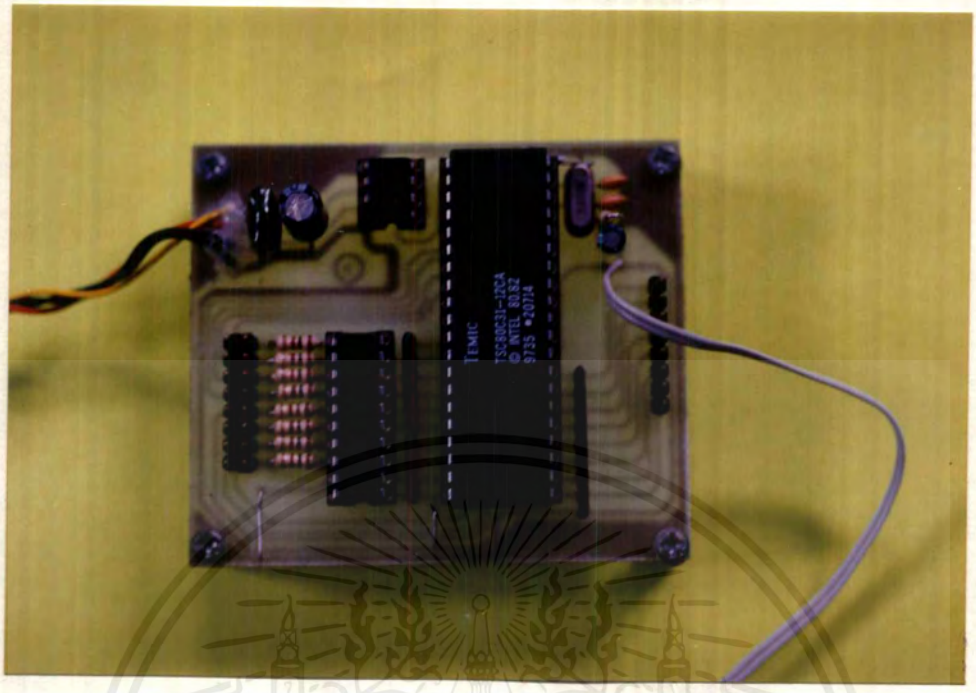
3. ในการทำงานของชุดตอบคำถามจะมีปัญหาในเรื่องของแหล่งจ่ายไฟที่เป็นแบบสวิตชิ่งที่ต่อกับชุดตอบคำถาม
แนวทางการแก้ไข คือ ได้ทำการนำคาปาซิเตอร์ค่า 0.01 μF/50 V มาต่อคร่อมแหล่งจ่ายไฟ และได้ทำการต่อตัวถังของ คริสตอลลงกราวด์

4. การทำงานของชุดแปลงสัญญาณจาก RS-232 เป็น RS-485ไม่สามารถทำงานได้
แนวทางการแก้ไข คือ ได้ทำการนำคาปาซิเตอร์ค่า 10 μF / 50 V มาต่อคร่อมกับแหล่งจ่ายไฟ

5.3 แนวทางการพัฒนาโครงงาน

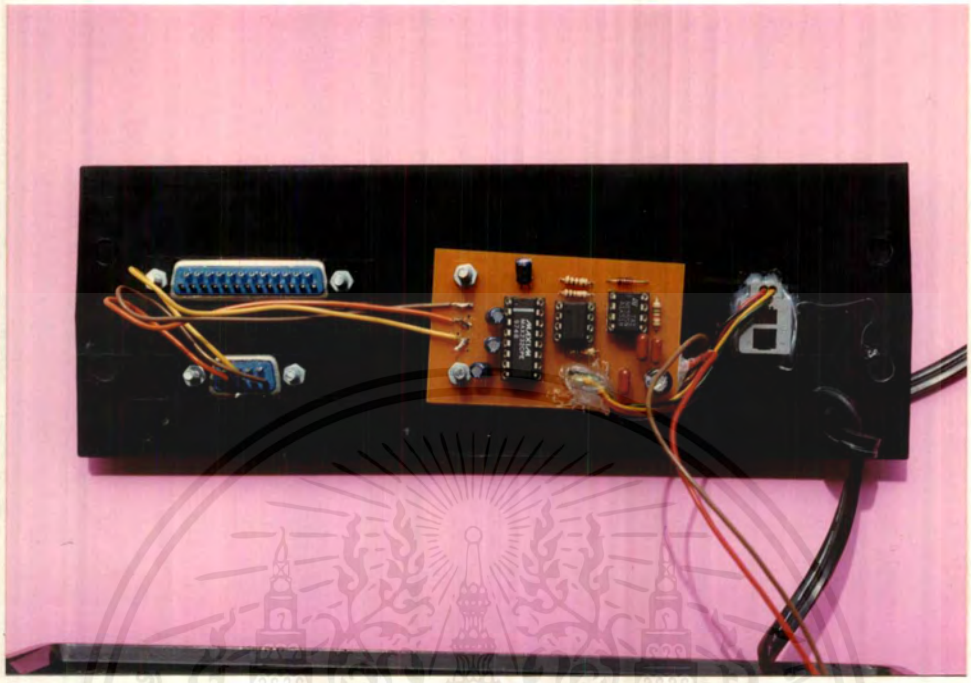
1. ในการพัฒนาทางด้านฮาร์ดแวร์ คือ ชุดตอบคำถาม จำเป็นที่จะต้องทำการปรับปรุงการแสดงผลให้มีการเพิ่มความสว่างให้มากกว่านี้ โดยทำการเปลี่ยนแปลงการต่อร่วมของ 7 Segment เข้า 89C51 มาเป็นการต่อผ่านทรานซิสเตอร์ลงกราวด์แทน
2. ในการพัฒนาทางด้าน Keyboard ให้มีจำนวนเพียงพอกับความต้องการ
3. การพัฒนาทางด้านโปรแกรมการรับ และการส่งข้อมูล ตัวโปรโตคอลการรับส่งควรจะมีมาตรฐานที่ดีกว่านี้ และโครงสร้างโปรแกรมควรปรับปรุงให้มีการตอบสนองการเรียกรับและส่งข้อมูลด้วย
4. โปรแกรมยังขาดส่วนของการแสดงผลออกทางจอภาพแสดงผลของผู้สั่ง ซึ่งควรจะต้องให้มีการแสดงผลด้วย
5. ต้องมีการพัฒนาในการส่งข้อมูลคำตอบที่ได้จากผู้เรียนผ่านเครือข่ายระบบสื่อสาร

ภาคผนวก ก
รูปเครื่องต้นแบบ

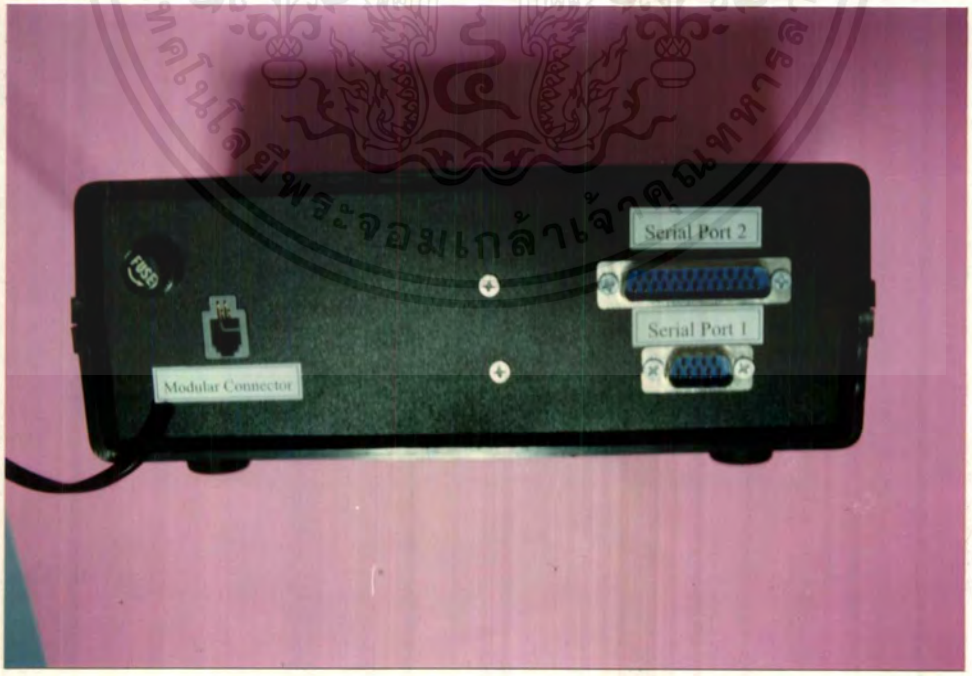


รูปที่ ก.1 แผงวงจรชุดตอบคำถาม

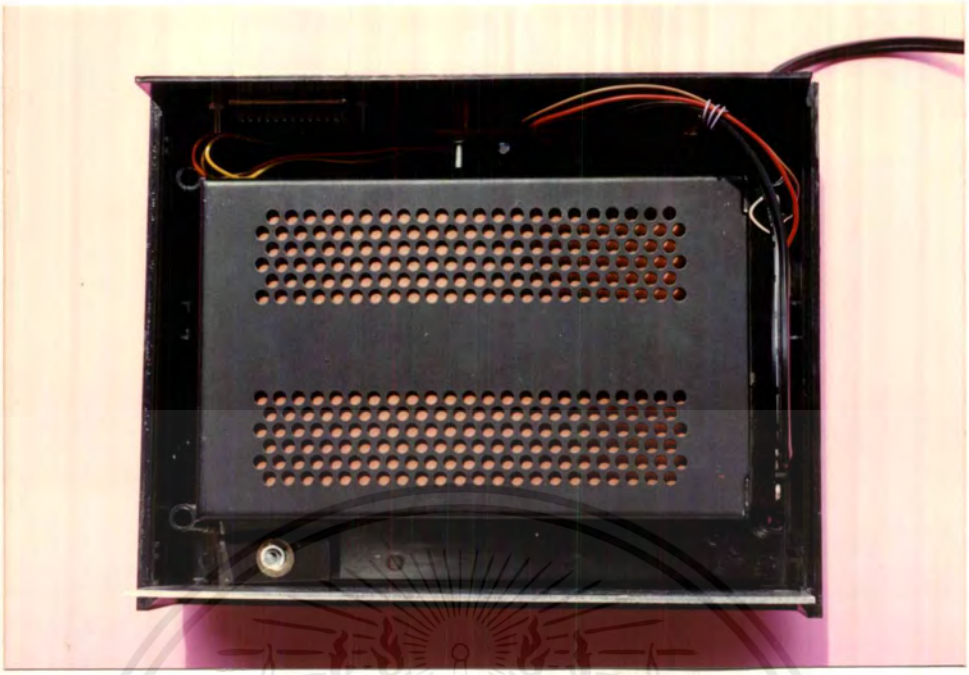
รูปที่ ก.2 ด้านหน้าชุดตอบคำถาม



รูปที่ ก.3 การลงอุปกรณ์ภายในเครื่องแปลงสัญญาณ RS-232 เป็น RS-485

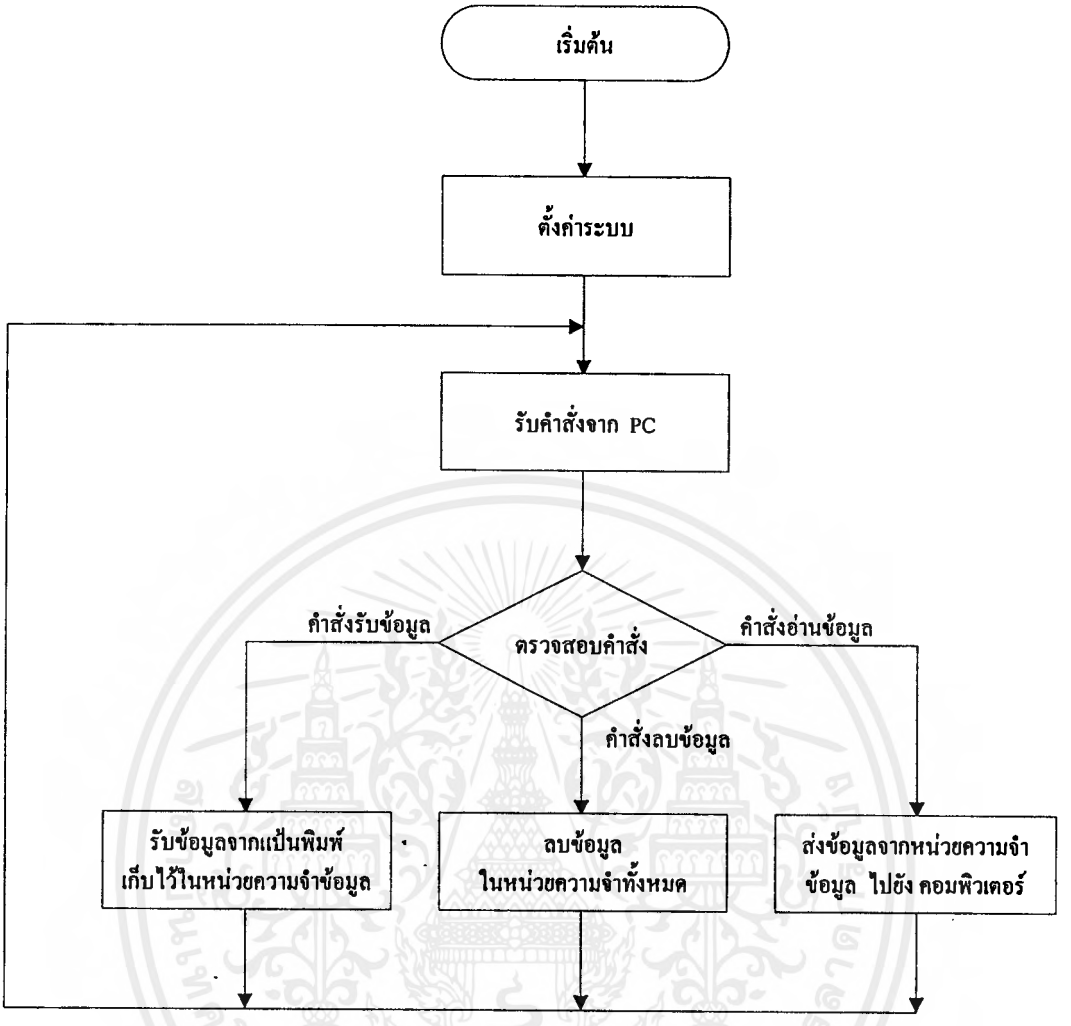


รูปที่ ก.4 ด้านหลังของเครื่องแปลงสัญญาณ RS-232 เป็น RS-485

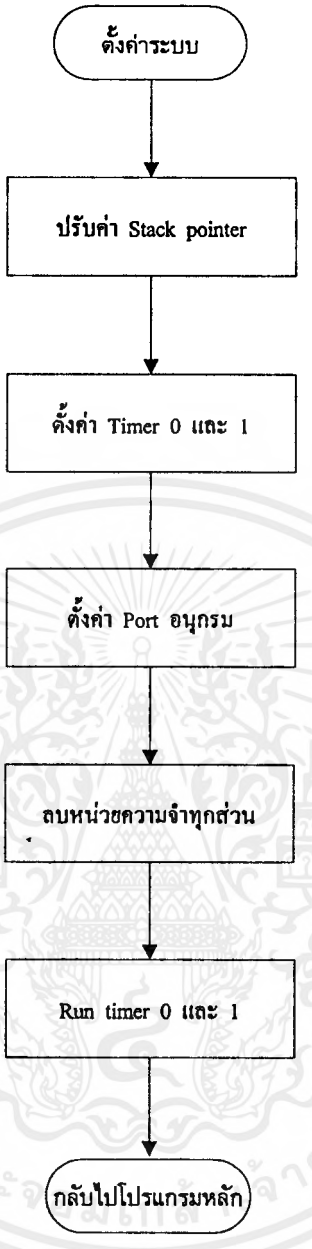


รูปที่ ก.5 ด้านบนของเครื่องแปลงสัญญาณ RS-232 เป็น RS-485

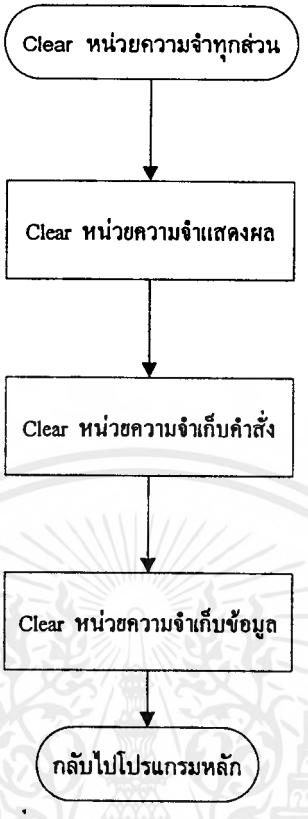
ภาคผนวก ข
ผังการทำงานและโปรแกรมการทำงาน



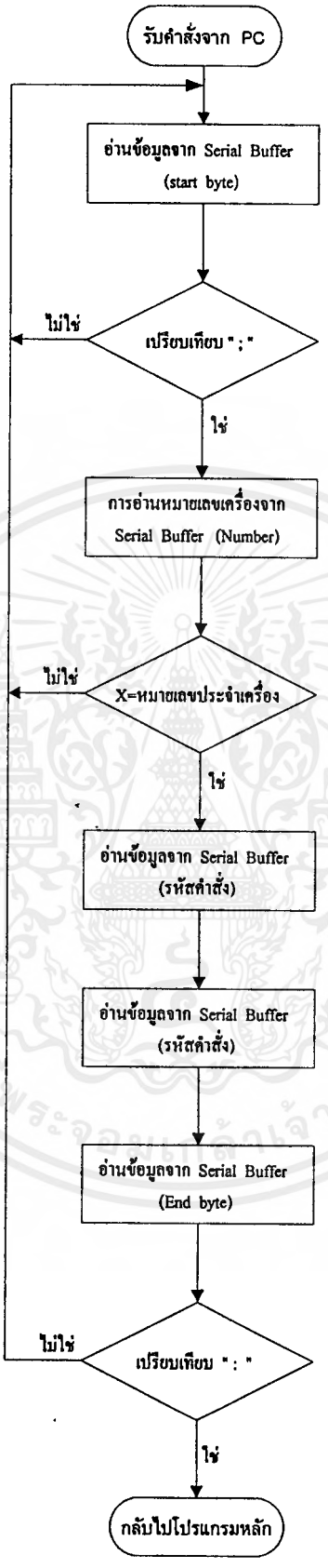
รูปที่ ข.1 ผังการทำงานหลักของโปรแกรมบนชุดตอบคำถาม



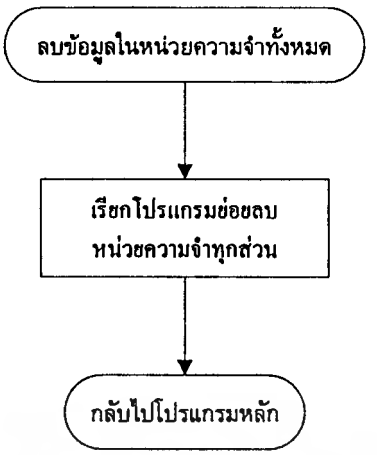
รูปที่ ข.2 ผังการทำงานของโปรแกรมย่อย ตั้งค่าระบบ



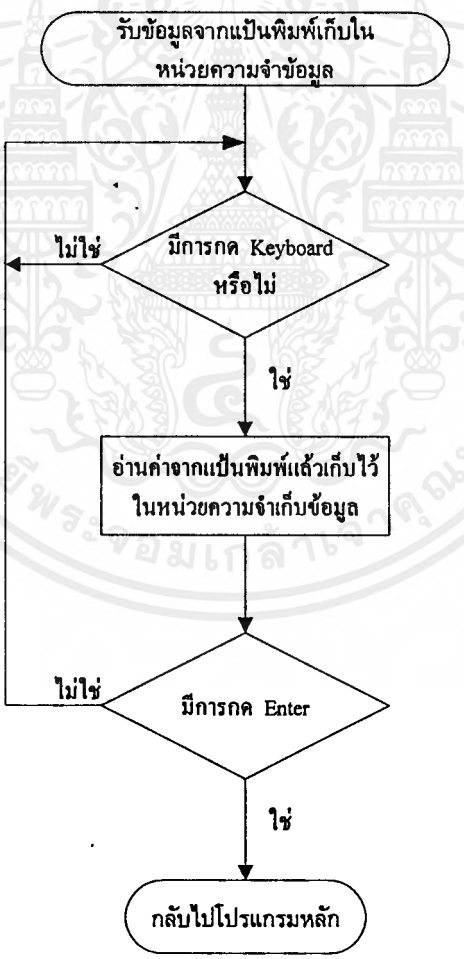
รูปที่ ข.3 ผังการทำงานของโปรแกรมย่อย ลบข้อมูลในหน่วยความจำ



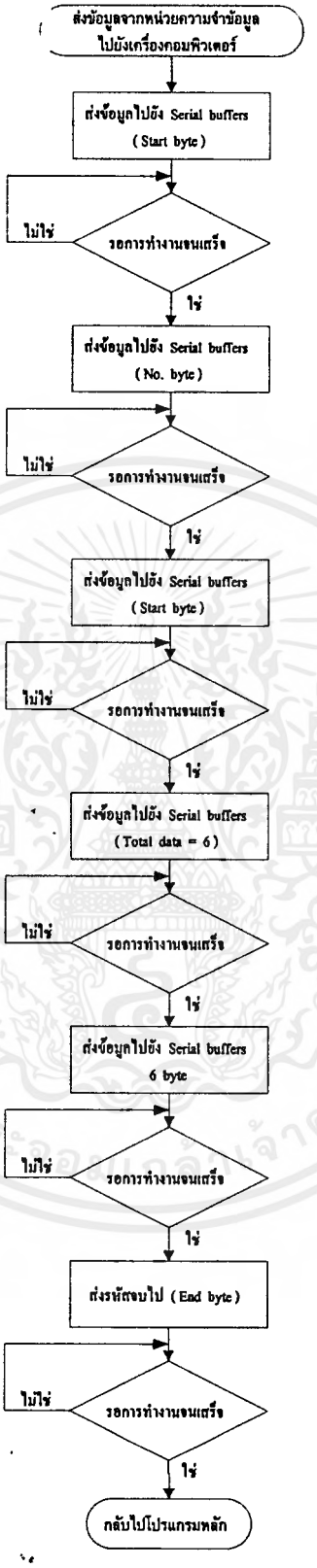
รูปที่ ข.4 ผังการทำงานของโปรแกรมย่อย รับคำสั่งจากเครื่องคอมพิวเตอร์



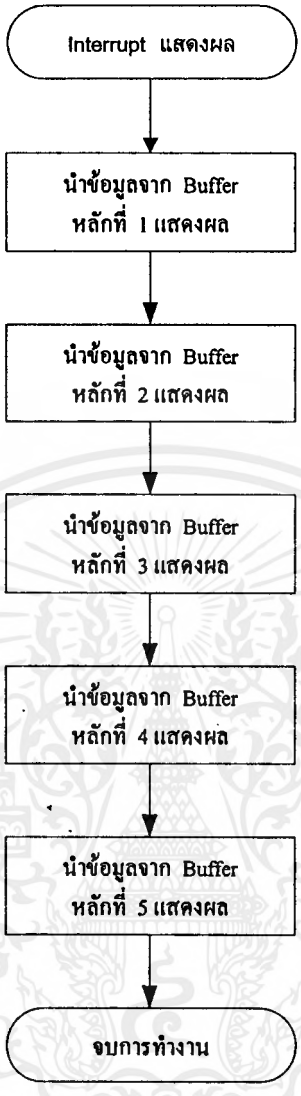
รูปที่ ข.5 ผังการทำงานของโปรแกรมย่อย Clear ข้อมูลในหน่วยความจำทุกส่วน



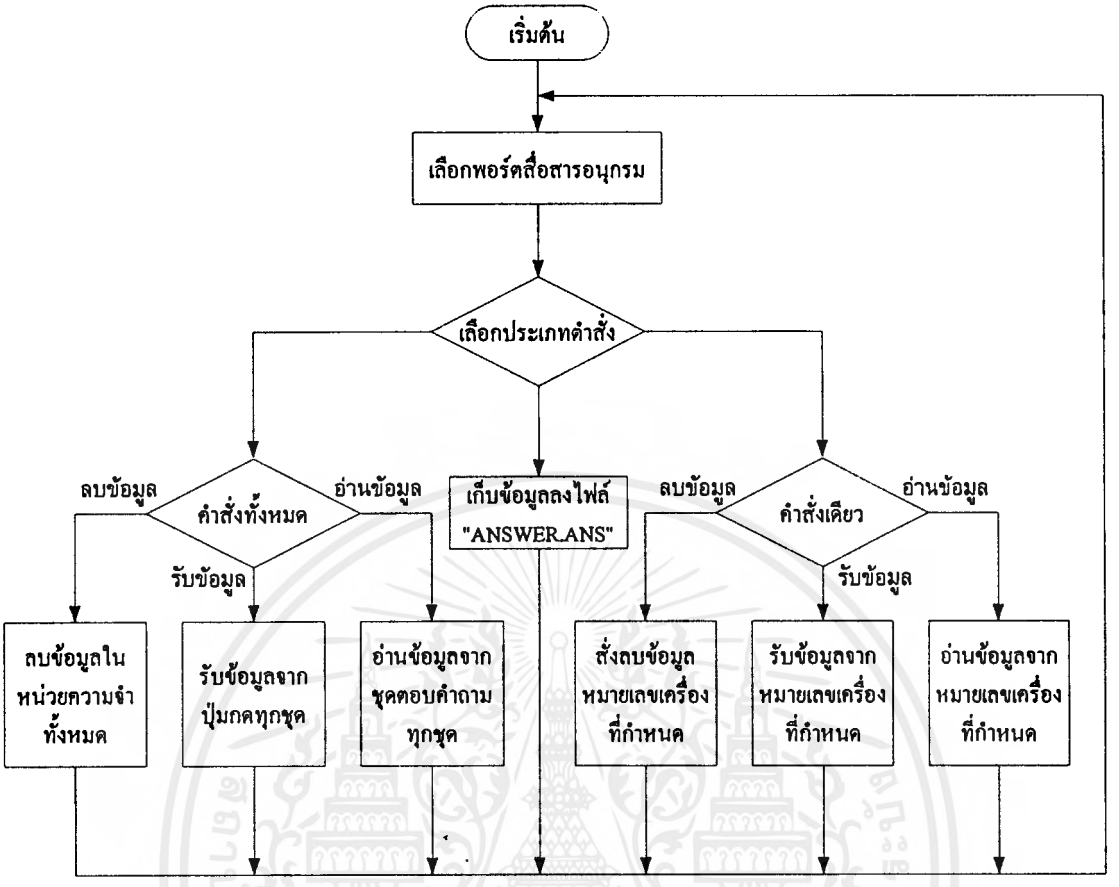
รูปที่ ข.6 ผังการทำงานของ การรับข้อมูลจากคีย์บอร์ด



รูปที่ ข.7 ผังการทำงานของการ ส่งข้อมูลจากหน่วยความจำเก็บข้อมูลไปยังเครื่องคอมพิวเตอร์



รูปที่ ข.8 ผังการทำงานของ โปรแกรมตั้งเวลาขัดจังหวะ



รูปที่ ข.9 ผังการทำงานของคอมพิวเตอร์

โปรแกรมควบคุมการทำงานบนชุดตอบคำถาม

```
; start internal ram use at 30h
; 30-3fh use to buffers of key press
; 40h-44h use to buffers of segment
; 45h buffers data of keyboard
; start bit address use at 20h
; 20h use keyboard flag "have press" if press = "1" if not = "0"
; 21h use to check dot if "1" has dot if "0" not have dot
; 22h flag key use for input message "0"=first display "1" display complete
; 50h-5fh use for buffers to communication with computer
; detail :
; receive  50h = start byte to recieve
;        51h = selecter number of machine
;        52h = command
;        53h = command data
;        54h = end byte of recieve
;
; transmit 55h = start byte to transmit
;        56h = number of machine that transmit
;        57h = total data
;        58h = data buffer
;        59h = data buffer
;        5ah = data buffer
;        5bh = data buffer
;        5ch = data buffer
;        5dh = data buffer
;        5eh = sum of data
;        5fh = end byte of transmit
;
```

```
; Command to each machine
; 0 : clear all data in buffers
; 1 : input data
; 2 : transmit data
;
; p2.0 '0' turn rs-485 recieve data
;      '1' turn rs-485 transmit data
;
; This part is define machine number to use

machine equ    'X'                    ; change this to specify number machine
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      org     0000h
entry:  ajmp    start
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      org     000bh
      ajmp    tim0isr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:
      mov     sp,#60h              ; stack pointer start
      mov     ie,#82h              ; enable timer0 & serial interrupt
      mov     scon,#52h            ; serial mode 1 recieve & transmit data
      mov     tmod,#21h            ; timer 0 mode 1 & timer 1 mode 2
      mov     th1,#0fdh            ; boud rate 9600 kbps
      mov     tl0,#00h
      mov     th0,#00h
      clr     tf0                  ; clear interrupt flag timer 0
      setb    tr0                  ; run timer 0
```

```
        setb   tr1                  ; run timer 1 (serial)
        setb   ti                   ; off serial transmit flag
        setb   ri                   ; off serial recieve flag
                                    ; p2.0 '0' turn rs-485 recieve data
                                    ;      '1' turn rs-485 transmit data
        acall  clear_all            ; clear all buffers
        mov    55h,#';'             ; start byte
        mov    56h,#machine         ; number machine
        mov    5fh,#':'             ; end byte
;;;;;;;;;;;;;;;; Main program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
main:
        acall  clear_disbuf
        mov    dptr,#ready_msg
        acall  display_msg

        acall  recieve_command      ; recieve command & command data

        mov    a,52h                ; command buffers
        cjne   a,#'0',next_cmmd1    ; if command "0"
        acall  clear_all            ; then clear all buffers
        mov    55h,#';'             ; start byte
        mov    56h,#machine         ; number machine
        mov    5fh,#':'             ; end byte

        mov    dptr,#clear_msg

        acall  display_msg          ; display clear message on segment

        acall  delay
```

```
        acall   delay
        acall   delay
        acall   delay
        ajmp    main                ; and goto main loop
next_cmmd1:                         ; else check next command
        cjne    a,#'1',next_cmmd2   ; if command "1"
input_again:
        acall   clear_data_buffers
        mov     dptr,#input_msg
        acall   display_msg         ; display input message
        clr     21h
        acall   inkey               ; input data to buffers
        jnb     21h,take_protocal
        sjmp    input_again
take_protocal:
        acall   protocal            ; convert data to protocal
                                    ; for ready to transmit
        ajmp    main                ; goto main loop
next_cmmd2:
        cjne    a,#'2',main         ; if command "2"
        mov     dptr,#send_msg
        acall   display_msg         ; display send message
        acall   send_data           ; send data to pc
        mov     r0,#0
        acall   delay
        acall   delay
        acall   delay
        acall   delay
        acall   delay
        ajmp    main            ; goto main loop
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
send_data:
      setb   p2.0
      mov    r0,#01
      acall  delay
      clr    ti
      mov    sbuf,#0ffh     ; synchronize charactor
      jnb    ti,$

      mov    r0,#01
      acall  delay
      clr    ti
      mov    sbuf,55h       ; start byte
      jnb    ti,$

      mov    r0,#01
      acall  delay
      clr    ti
      mov    sbuf,#0ffh     ; synchronize charactor
      jnb    ti,$

      mov    r0,#01
      acall  delay
      clr    ti
      mov    sbuf,56h       ; number machine
      jnb    ti,$

      mov    r0,#01
      acall  delay
      clr    ti
```

```
mov    sbuf,#0ffh     ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,57h       ; total data
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,#0ffh     ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,58h       ; data byte 1
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,#0ffh     ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
```

```
mov    sbuf,59h        ; data byte 2
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,#0ffh      ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,5ah        ; data byte 3
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,#0ffh      ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,5bh        ; data byte 4
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
```

```
mov    sbuf,#0ffh     ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,5ch       ; data byte 5
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,#0ffh     ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,5dh       ; data byte 6
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
mov    sbuf,#0ffh     ; synchronize charactor
jnb    ti,$

mov    r0,#01
acall  delay
clr    ti
```

```
	mov	sbuf,5eh	; sum of data
	jnb	ti,$

	mov	r0,#01
	acall	delay
	clr	ti
	mov	sbuf,#0ffh	; synchronize charactor
	jnb	ti,$

	mov	r0,#01
	acall	delay
	clr	ti
	mov	sbuf,5fh	; end byte
	jnb	ti,$
	ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
protocal:
	mov	a,58h
	add	a,59h
	add	a,5ah
	add	a,5bh
	add	a,5ch
	add	a,5dh
	cjne	a,#0ffh,normal	; if sum != 0ffh it normal sum
	mov	a,#00h		; if sum  = 0ffh it special sum
normal:
	mov	5eh,a		; sum of data 6 bytes
	mov	55h,#';'	; start byte
	mov	56h,#machine	; number machine
```

```
        mov     57h,#'6'        ; total data
        mov     5fh,#':'        ; end byte
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
clear_data_buffers:
        mov     58h,#00h
        mov     59h,#00h
        mov     5ah,#00h
        mov     5bh,#00h
        mov     5ch,#00h
        mov     5dh,#00h
        mov     5eh,#00h
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
clear_all:
        mov     r0,#30h         ;
        mov     r1,#30h         ; buffers 30h -> 5fh is clear
clr1:   mov     @r1,#00h
        inc     r1
        djnz    r0,clr1
        ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
recieve_command:
        clr     p2.0
recv_start:
        clr     ri
        jnb     ri,$
```

```
mov   50h,sbuf
mov   a,50h
cjne  a,#';',recv_start      ; check start byte if true
                             ; then recieve next byte
                             ; else recieve again
clr   ri
jnb   ri,$
mov   51h,sbuf
mov   a,51h
cjne  a,#machine,recv_start ; check number byte if true
                            ; then recieve next byte
                            ; else recieve all again
clr   ri
jnb   ri,$
mov   52h,sbuf              ; recieve command byte

clr   ri
jnb   ri,$
mov   53h,sbuf              ; recieve command data byte

clr   ri
jnb   ri,$
mov   54h,sbuf
mov   a,54h
cjne  a,#':',recv_start     ; check end byte if true
                            ; then exit subrutine
                            ; else recieve all again
ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
inkey:
    clr    22h
again:
    acall  scankey
    jb     20h,op              ; if press key then o/p
    sjmp   again               ; else try again
op:
    jb     22h,op1
    acall  clear_disbuf1
    setb   22h
op1:
    mov    a,45h
    cjne   a,#0bh,display      ; if press enter
                               ; then exit
    ret
display:                       ; else write key press
    acall  write_seg           ; on to screen
    jnb    21h,next_op
    ret
next_op:
    clr    20h
    sjmp   again
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
display_msg:
    push   dph
    push   dpl
    push   02h
    push   00h
    push   01h
```

```
    mov    r1,#5
    mov    r0,#40h
    mov    r2,#0
next_char:
    mov    a,r2
    movc   a,@a+dptr
    mov    @r0,a
    inc    r0
    inc    r2
    djnz   r1,next_char

    pop    01h
    pop    00h
    pop    02h
    pop    dpl
    pop    dph
    ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
clear_disbuf:
    mov    40h,#00h
    mov    41h,#00h
    mov    42h,#00h
    mov    43h,#00h
    mov    44h,#00h
    mov    45h,#00h
    clr    20h
    clr    21h
    ret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
clear_disbufl:
     mov    40h,#00h
     mov    41h,#00h
     mov    42h,#00h
     mov    43h,#00h
     mov    44h,#00h
     clr    20h
     clr    21h
     ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
scankey:
     jnb    20h,not_press
     ajmp   press

not_press:
     mov    p1,#0ffh

     clr    p1.0

chk_1: jb     p1.3,chk_4
     jnb    p1.3,$
     mov    45h,#1
     mov    5eh,#'1'
     setb   20h                 ; press now
chk_4: jb     p1.4,chk_7
     jnb    p1.4,$
     mov    45h,#4
```

```
        mov    5eh,#'4'
        setb   20h                    ; press now
chk_7:  jb     p1.5,chk_star
        jnb    p1.5,$
        mov    45h,#7
        mov    5eh,#'7'
        setb   20h                    ; press now
chk_star:
        jb     p1.6,chk_2
        jnb    p1.6,$
        mov    45h,#10                ; 10 is dot & cancel code
        mov    5eh,#'*'
        setb   20h                    ; press now

chk_2:  setb   p1.0
        clr    p1.1

        jb     p1.3,chk_5
        jnb    p1.3,$
        mov    45h,#2
        mov    5eh,#'2'
        setb   20h                    ; press now
chk_5:  jb     p1.4,chk_8
        jnb    p1.4,$
        mov    45h,#5
        mov    5eh,#'5'
        setb   20h                    ; press now
chk_8:  jb     p1.5,chk_0
        jnb    p1.5,$
        mov    45h,#8
```

```
        mov    5eh,#'8'
        setb   20h                    ; press now
chk_0:  jb     p1.6,chk_3
        jnb    p1.6,$
        mov    45h,#0
        mov    5eh,#'0'
        setb   20h                    ; press now

chk_3:  setb   p1.1
        clr    p1.2

        jb     p1.3,chk_6
        jnb    p1.3,$
        mov    45h,#3
        mov    5eh,#'3'
        setb   20h                    ; press now
chk_6:  jb     p1.4,chk_9
        jnb    p1.4,$
        mov    45h,#6
        mov    5eh,#'6'
        setb   20h                    ; press now
chk_9:  jb     p1.5,chk_shrp
        jnb    p1.5,$
        mov    45h,#9
        mov    5eh,#'9'
        setb   20h                    ; press now
chk_shrp:
        jb     p1.6,no_press
        jnb    p1.6,$
        mov    45h,#11                ; enter code
```

```
    mov    5eh,#'#'
    setb   20h                  ; press now

no_press:
press:
    mov    p1,#0ffh
    ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
write_seg:
    push   dph
    push   dpl
    mov    a,45h
    cjne   a,#0ah,shift         ; if press star then o/p dot
    setb   21h                  ; has dot
    sjmp   exit_seg
shift:
    mov    dptr,#digit
    mov    a,45h
    movc   a,@a+dptr
    mov    45h,a
    cjne   a,#0dbh,shift1
    mov    a,44h
    cjne   a,#00h,shift1
    sjmp   exit_seg
shift1:
    mov    40h,41h
    mov    41h,42h
    mov    42h,43h
    mov    43h,44h
```

```
    mov   44h,45h

    mov   58h,59h
    mov   59h,5ah
    mov   5ah,5bh
    mov   5bh,5ch
    mov   5ch,5dh
    mov   5dh,5eh
exit_seg:
    pop   dpl
    pop   dph
    ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
tim0isr:
    push  00h
    push  01h

    clr   tr0          ; stop timer 0
    mov   p0,40h
    clr   p2.3
    mov   r0,#04h
    acall delay
    setb  p2.3

    mov   p0,41h
    clr   p2.2
    mov   r0,#04h
    acall delay
    setb  p2.2
```

```
        mov    p0,42h
        clr    p2.1
        mov    r0,#04h
        acall  delay
        setb   p2.1

        mov    p0,43h
        clr    p2.7
        mov    r0,#04h
        acall  delay
        setb   p2.7

        mov    p0,44h
        clr    p2.5
        mov    r0,#04h
        acall  delay
        setb   p2.5


        mov    tl0,#00h
        mov    th0,#0d0h
        clr    tf0             ; clear interrupt flag timer 0
        setb   tr0             ; run timer 0
        pop    01h
        pop    00h
        reti
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
delay: mov     r1,#00h
```

```
del:   djnz   r1,del
       djnz   r0,del
       ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
digit: db     0dbh,042h,09eh,05eh,047h,05dh
       db     0ddh,04ah,0dfh,05fh,00ah,00bh
send_msg:
       db     05dh,09dh,0c4h,0d6h,000h
clear_msg:
       db     094h,091h,09dh,0cfh,084h
ready_msg:
       db     084h,09dh,0cfh,0d6h,057h
input_msg:
       db     042h,0c4h,08fh,0d0h,085h
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       end
```

โปรแกรมควบคุมและรับส่งข้อมูลชุดตอบคำถาม

```
// Offset              register
//  0                  transmit buffer
//  0                  receive buffer
//  0                  latch divisor LSB
//  1                  latch divisor MSB
//  1                  interrupt enable
//  2                  interrupt priority
//  3                  line control
//  4                  modem control
//  5                  line status
//  6                  modem status
// Program Communication with all Child
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>
#define  COM1       0x3f8          // Communication Port 1
#define  COM2       0x2f8          // Communication Port 2

int  SelectCom(void);              // Select communication port
void Connect(int);                 // Connect with MCS-51 all child
int  TxReady(int);                 // Check Com Port ready to transmit
int  RxReady(int);                 // Check Com Port ready to recieve
void Send(int,char);               // Send data to com port
unsigned char Read(int);           // Read data from com port
unsigned char GetCommand(void); // Get command from file and keyboard
void ClearBuffers(int,char);       // Clear buffers in controller
void InputData(int,char);          // Input key data to controller
void ReadData(int,char);           // Read data from controller
```

```
int  Initial(void);                  // Initial status of system
void ClearAns(void);                 // Clear answer box
void ClearAll(int);                  // Clear all child buffers
void InputAll(int);                  // Input all child
void ReadAll(int);                   // Read all data all child
void ClearOne(int);                  // Clear one machine
void InputOne(int);                  // Input one machine
void ReadOne(int);                   // Read onr machine
void WriteFile(void);                // Write data to file

unsigned int  del_time=15;           // delay time of communication
unsigned long Waiting=300000;        // long time to recieve
unsigned char dat[12];               // buffers data read from child
unsigned char data[6][6];            // data save to files

int  press=0;                        // flag to specifies data read from file
                                     // one time if it not 0 it not take
                                     // old command from file

//////////////////////////////////////////////////////////////////////////
int main()
{
  int  Com;                          // Store number of communication port
  int  OK;
  char ch;

  Com = Initial();
  if(Com==0) return(0);
  do {
    gotoxy(10,22);
```

```
    printf("Select Command >> "); ch=GetCommand();
    switch(ch) {
                        case 0x08:putch(' ');break;
                        case  13:break;
                        case  27:return(0);
                        case  '1':putch(ch);
                            ClearAll(Com);
                            break;
                        case  '2':putch(ch);
                            InputAll(Com);
                            break;
                        case  '3':putch(ch);
                            ReadAll(Com);
                            break;
                        case  '4':putch(ch);
                            ClearOne(Com);
                            break;
                        case  '5':putch(ch);
                            InputOne(Com);
                            break;
                        case  '6':putch(ch);
                            ReadOne(Com);
                            break;
                        case  '7':putch(ch);
                            WriteFile();
                            break;
    }
  } while(1);
}
```

```
////////////////////////////////////////////////////////////////////////
int Initial(void)
{
  int Com,i,j;

  for(i=0;i<=6;i++) for(j=0;j<=6;j++) data[i][j] = 0;
  textbackground(1);
  textcolor(14);
  clrscr();
  textcolor(14);
  printf("\n ยอออออออออออออออออออออออออออออออออออออออออออ\
\หอออออออออออออออออออออออออออออออออออออออออออออออออป");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
  printf("\n บ                    \
\บ                    บ");
```

```
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n บ                    \
  \บ                  บ");
    printf("\n ศออออออออออออออออออออออออออออออออออออออออออออออออ\
  \ศออออออออออออออออออออออออออออออออออออออออออออออออผ");

    gotoxy(33,1);printf("Control Center.");
    gotoxy(3,6); printf("   1.Clear all child buffers");
    gotoxy(3,8); printf("   2.Input key all child");
```

```
  gotoxy(3,10);printf("  3.Read data from all child");
  gotoxy(3,12);printf("  4.Clear child buffers");
  gotoxy(3,14);printf("  5.Input key one child");
  gotoxy(3,16);printf("  6.Read data from one child");
  gotoxy(3,18);printf("  7.Write data to files");

  ClearAns();
  Com = SelectCom();
  if(Com == -1) return(-1);
  Connect(Com);
  return(Com);
}

////////////////////////////////////////////////////////////////////////
void ClearAns(void)
{
  textbackground(1);  gotoxy(42,6);    cprintf("  1. ");
  textbackground(0);                   cprintf("     0 ");

  textbackground(1);  gotoxy(42,8);    cprintf("  2. ");
  textbackground(0);                   cprintf("     0 ");

  textbackground(1);  gotoxy(42,10);   cprintf("  3. ");
  textbackground(0);                   cprintf("     0 ");

  textbackground(1);  gotoxy(42,12);   cprintf("  4. ");
  textbackground(0);                   cprintf("     0 ");

  textbackground(1);  gotoxy(42,14);   cprintf("  5. ");
  textbackground(0);                   cprintf("     0 ");
```

```
  textbackground(1);  gotoxy(42,16);   cprintf("  6. ");
  textbackground(0);                   cprintf("          0 ");
  textbackground(1);
}

////////////////////////////////////////////////////////////////////////
void ClearOne(int Com)
{
  char Num,ch;

  do {
    gotoxy(10,22);
    printf("Select clear machine >> "); ch = getch();
    switch(ch){
                        case  '1':putch('1');Num=ch;break;
                        case  '2':putch('2');Num=ch;break;
                        case  '3':putch('3');Num=ch;break;
                        case  '4':putch('4');Num=ch;break;
                        case  '5':putch('5');Num=ch;break;
                        case  '6':putch('6');Num=ch;break;
                        case 0x08:putch(' ');break;
                        case   27:gotoxy(10,22);printf("                    ");return;
                        case   13:switch(Num){
                              case '1':textbackground(0);
                                       gotoxy(48,6); cprintf("       Clear ");
                                       ClearBuffers(Com,Num);
                                       delay(1000);
                                       gotoxy(48,6); cprintf("          0 ");
                                       break;
                              case '2':textbackground(0);
```

```
                gotoxy(48,8); cprintf("      Clear ");
                ClearBuffers(Com,Num);
                delay(1000);
                gotoxy(48,8); cprintf("         0 ");
                break;
        case '3':textbackground(0);
                gotoxy(48,10);cprintf("      Clear ");
                ClearBuffers(Com,Num);
                delay(1000);
                gotoxy(48,10);cprintf("         0 ");
                break;
        case '4':textbackground(0);
                gotoxy(48,12);cprintf("      Clear ");
                ClearBuffers(Com,Num);
                delay(1000);
                gotoxy(48,12);cprintf("         0 ");
                break;
        case '5':textbackground(0);
                gotoxy(48,14);cprintf("      Clear ");
                ClearBuffers(Com,Num);
                delay(1000);
                gotoxy(48,14);cprintf("         0 ");
                break;
        case '6':textbackground(0);
                gotoxy(48,16);cprintf("      Clear ");
                ClearBuffers(Com,Num);
                delay(1000);
                gotoxy(48,16);cprintf("         0 ");
                break;
}
```

```
                    break;
    }
    textbackground(1);
  }while(1);
}

////////////////////////////////////////////////////////////////////////
void InputOne(int Com)
{
  char Num,ch;

  gotoxy(28,22);putch(' ');
  do {
    gotoxy(10,22);
    printf("Select input machine >> "); ch = getch();
    switch(ch){
                    case '1':putch('1');Num=ch;break;
                    case '2':putch('2');Num=ch;break;
                    case '3':putch('3');Num=ch;break;
                    case '4':putch('4');Num=ch;break;
                    case '5':putch('5');Num=ch;break;
                    case '6':putch('6');Num=ch;break;
                    case 0x08:putch(' ');break;
                    case  27:gotoxy(10,22);printf("                    ");return;
                    case  13:switch(Num){
                        case '1':textbackground(0);
                                 gotoxy(48,6); cprintf("      Input ");
                                 InputData(Com,Num);
                                 delay(1000);
                                 break;
```

```
                case '2':textbackground(0);
                        gotoxy(48,8); cprintf("      Input ");
                        InputData(Com,Num);
                        delay(1000);
                        break;
                case '3':textbackground(0);
                        gotoxy(48,10);cprintf("      Input ");
                        InputData(Com,Num);
                        delay(1000);
                        break;
                case '4':textbackground(0);
                        gotoxy(48,12);cprintf("      Input ");
                        InputData(Com,Num);
                        delay(1000);
                        break;
                case '5':textbackground(0);
                        gotoxy(48,14);cprintf("      Input ");
                        InputData(Com,Num);
                        delay(1000);
                        break;
                case '6':textbackground(0);
                        gotoxy(48,16);cprintf("      Input ");
                        InputData(Com,Num);
                        delay(1000);
                        break;
            }
            break;
    }
    textbackground(1);
}while(1);}
```

```
////////////////////////////////////////////////////////////////////////
void ReadOne(int Com)
{
  char Num,ch;
  int  err = 0;

  gotoxy(28,22);putch(' ');
  do {
    gotoxy(10,22);
    printf("Select read machine >> "); ch = getch();
    switch(ch){
                    case  '1':putch('1');Num=ch;break;
                    case  '2':putch('2');Num=ch;break;
                    case  '3':putch('3');Num=ch;break;
                    case  '4':putch('4');Num=ch;break;
                    case  '5':putch('5');Num=ch;break;
                    case  '6':putch('6');Num=ch;break;
                    case 0x08:putch(' ');break;
                    case   27:gotoxy(10,22);printf("                    ");return;
                    case   13:switch(Num){
                          case '1':textbackground(0);
                                   ReadData(Com,Num);
                                   data[0][0] = dat[3];
                                   data[0][1] = dat[4];
                                   data[0][2] = dat[5];
                                   data[0][3] = dat[6];
                                   data[0][4] = dat[7];
                                   data[0][5] = dat[8];
                                   textbackground(0);
                                   err = 0;
```

```
                for(Num=0;Num<6;Num++)
                if(data[0][Num]>'9') err = 1;
                if(err) { gotoxy(49,6); cprintf("Not Ready!..."); }
                else {
                      gotoxy(49,6); cprintf("          ");
                      for(Num=0;Num<6;Num++) {
                        gotoxy(55+Num,6);
                        putch(data[0][Num]);
                      }
                }
                break;
case '2':textbackground(0);
                ReadData(Com,Num);
                data[1][0] = dat[3];
                data[1][1] = dat[4];
                data[1][2] = dat[5];
                data[1][3] = dat[6];
                data[1][4] = dat[7];
                data[1][5] = dat[8];
                textbackground(0);
                err = 0;
                for(Num=0;Num<6;Num++)
                if(data[1][Num]>'9') err = 1;
                if(err) { gotoxy(49,8); cprintf("Not Ready!..."); }
                else {
                      gotoxy(49,8); cprintf("          ");
                      for(Num=0;Num<6;Num++) {
                        gotoxy(55+Num,8);
                        putch(data[1][Num]);
                      }
```

```
                }
                break;
        case '3':textbackground(0);
                ReadData(Com,Num);
                data[2][0] = dat[3];
                data[2][1] = dat[4];
                data[2][2] = dat[5];
                data[2][3] = dat[6];
                data[2][4] = dat[7];
                data[2][5] = dat[8];
                textbackground(0);
                err = 0;
                for(Num=0;Num<6;Num++)
                if(data[2][Num]>'9') err = 1;
                if(err) { gotoxy(49,10); cprintf("Not Ready!..."); }
                else {
                     gotoxy(49,10); cprintf("          ");
                     for(Num=0;Num<6;Num++) {
                      gotoxy(55+Num,10);
                       putch(data[2][Num]);
                     }
                }
                break;
        case '4':textbackground(0);
                ReadData(Com,Num);
                data[3][0] = dat[3];
                data[3][1] = dat[4];
                data[3][2] = dat[5];
                data[3][3] = dat[6];
                data[3][4] = dat[7];
```

```
data[3][5] = dat[8];
textbackground(0);
err = 0;
for(Num=0;Num<6;Num++)
if(data[3][Num]>'9') err = 1;
if(err) { gotoxy(49,12); cprintf("Not Ready!..."); }
else {
      gotoxy(49,12); cprintf("          ");
      for(Num=0;Num<6;Num++) {
        gotoxy(55+Num,12);
        putch(data[3][Num]);
      }
}
break;
case '5':textbackground(0);
ReadData(Com,Num);
data[4][0] = dat[3];
data[4][1] = dat[4];
data[4][2] = dat[5];
data[4][3] = dat[6];
data[4][4] = dat[7];
data[4][5] = dat[8];
textbackground(0);
err = 0;
for(Num=0;Num<6;Num++)
if(data[4][Num]>'9') err = 1;
if(err) { gotoxy(49,14); cprintf("Not Ready!..."); }
else {
      gotoxy(49,14); cprintf("          ");
      for(Num=0;Num<6;Num++) {
```

```
                        gotoxy(55+Num,14);
                        putch(data[4][Num]);
                    }
                }
                break;
            case '6':textbackground(0);
                ReadData(Com,Num);
                data[5][0] = dat[3];
                data[5][1] = dat[4];
                data[5][2] = dat[5];
                data[5][3] = dat[6];
                data[5][4] = dat[7];
                data[5][5] = dat[8];
                textbackground(0);
                err = 0;
                for(Num=0;Num<6;Num++)
                if(data[5][Num]>'9') err = 1;
                if(err) { gotoxy(49,16); cprintf("Not Ready!..."); }
                else {
                    gotoxy(49,16); cprintf("          ");
                    for(Num=0;Num<6;Num++) {
                        gotoxy(55+Num,16);
                        putch(data[5][Num]);
                    }
                }
                break;
        }
        break;
    }   textbackground(1);
}while(1);}
```

```
////////////////////////////////////////////////////////////////////////
unsigned char GetCommand(void)
{
  FILE *cmd;
  char filename[30] = "/project/command.cmd";
  char ch;
  int  flag=1;

  do {
    if(kbhit()) return(getch());
    cmd = fopen(filename,"r");
    ch = fgetc(cmd);
    switch(ch) {
                    case '0':press=0;
                      break;
                    case '1':if(!press) {
                      press=1;
                      fclose(cmd);
                      return(ch);
                      }
                     break;
                   case '2':if(!press) {
                      press=1;
                      fclose(cmd);
                      return(ch);
                     }
                    break;
                  case '3':if(!press) {
                      press=1;
                      fclose(cmd);
```

```
                    return(ch);
                  }
                  break;
      }
      fclose(cmd);
    } while(flag);
    fclose(cmd);
    return(0);
}

////////////////////////////////////////////////////////////////////
void WriteFile(void)
{
  FILE *file;
  char i,j,ch,filename[30]="/project/answer.ans";

  textbackground(0); gotoxy(10,22); cprintf("Writing to ""ANSWER.ANS""");

  file = fopen(filename,"w");
  for(i=0;i<6;i++) for(j=0;j<6;j++) {
    putc(data[i][j],file);
    putc(13,file);
  }

 for(i=0;i<14;i++) for(j=0;j<6;j++) {
    putc(0,file);
    putc(13,file);
  }

 fclose(file);
```

```
  getch(); textbackground(1);
  gotoxy(10,22); cprintf("                    ");
}

////////////////////////////////////////////////////////////////////////
void ReadAll(int Com)
{
  char Num;
  int i,j,err=0;

  for(i=0;i<=6;i++) for(j=0;j<=6;j++) data[i][j] = 0;

  for(i=0;i<=6;i++) data[0][i] = 0;
  ReadData(Com,'1');
  data[0][0] = dat[3];
  data[0][1] = dat[4];
  data[0][2] = dat[5];
  data[0][3] = dat[6];
  data[0][4] = dat[7];
  data[0][5] = dat[8];
  textbackground(0);
  err = 0;
  for(Num=0;Num<6;Num++)
  if(data[0][Num]>'9') err = 1;
  if(err) {
    gotoxy(49,6); cprintf("Not Ready!...");
    for(Num=0;Num<6;Num++) data[0][Num] = 0;
  } else for(Num=0;Num<6;Num++) {
    gotoxy(55+Num,6);
```

```
    putch(data[0][Num]);
  }

 ,delay(2000);
  for(i=0;i<=6;i++) data[1][i] = 0;
  ReadData(Com,'2');
  data[1][0] = dat[3];
  data[1][1] = dat[4];
  data[1][2] = dat[5];
  data[1][3] = dat[6];
  data[1][4] = dat[7];
  data[1][5] = dat[8];
  textbackground(0);
  err = 0;
  for(Num=0;Num<6;Num++)
  if(data[1][Num]>'9') err = 1;
  if(err) {
    gotoxy(49,8); cprintf("Not Ready!...");
    for(Num=0;Num<6;Num++) data[1][Num] = 0;
  } else for(Num=0;Num<6;Num++) {
    gotoxy(55+Num,8);
    putch(data[1][Num]);
  }

  delay(2000);
  for(i=0;i<=6;i++) data[2][i] = 0;
  ReadData(Com,'3');
  data[2][0] = dat[3];
  data[2][1] = dat[4];
  data[2][2] = dat[5];
```

```
data[2][3] = dat[6];
data[2][4] = dat[7];
data[2][5] = dat[8];
textbackground(0);
err = 0;
for(Num=0;Num<6;Num++)
if(data[2][Num]>'9') err = 1;
if(err) {
  gotoxy(49,10); cprintf("Not Ready!...");
  for(Num=0;Num<6;Num++) data[2][Num] = 0;
} else for(Num=0;Num<6;Num++) {
  gotoxy(55+Num,10);
  putch(data[2][Num]);
}

delay(2000);
for(i=0;i<=6;i++) data[3][i] = 0;
ReadData(Com,'4');
data[3][0] = dat[3];
data[3][1] = dat[4];
data[3][2] = dat[5];
data[3][3] = dat[6];
data[3][4] = dat[7];
data[3][5] = dat[8];
textbackground(0);
err = 0;
for(Num=0;Num<6;Num++)
if(data[3][Num]>'9') err = 1;
if(err) {
  gotoxy(49,12); cprintf("Not Ready!...");
```

```
  for(Num=0;Num<6;Num++) data[3][Num] = 0;
} else for(Num=0;Num<=6;Num++) {
  gotoxy(55+Num,12);
  putch(data[3][Num]);
}

delay(2000);
for(i=0;i<=6;i++) data[4][i] = 0;
ReadData(Com,'5');
data[4][0] = dat[3];
data[4][1] = dat[4];
data[4][2] = dat[5];
data[4][3] = dat[6];
data[4][4] = dat[7];
data[4][5] = dat[8];
textbackground(0);
err = 0;
for(Num=0;Num<6;Num++)
if(data[4][Num]>'9') err = 1;
if(err) {
  gotoxy(49,14); cprintf("Not Ready!...");
  for(Num=0;Num<6;Num++) data[4][Num] = 0;
} else for(Num=0;Num<6;Num++) {
  gotoxy(55+Num,14);
  putch(data[4][Num]);
}

delay(2000);
for(i=0;i<=6;i++) data[5][i] = 0;
ReadData(Com,'6');
```

```
    data[5][0] = dat[3];
    data[5][1] = dat[4];
    data[5][2] = dat[5];
    data[5][3] = dat[6];
    data[5][4] = dat[7];
    data[5][5] = dat[8];
    textbackground(0);
    err = 0;
    for(Num=0;Num<6;Num++)
    if(data[5][Num]>'9') err = 1;
    if(err) {
      gotoxy(49,16); cprintf("Not Ready!...");
      for(Num=0;Num<6;Num++) data[5][Num] = 0;
    } else for(Num=0;Num<6;Num++) {
      gotoxy(55+Num,16);
      putch(data[5][Num]);
    }
   textbackground(1);

}

////////////////////////////////////////////////////////////////////////
void InputAll(int Com)
{
  char Num;

  textbackground(0);
  gotoxy(48,6); cprintf("      Input ");
  gotoxy(48,8); cprintf("      Input ");
  gotoxy(48,10);cprintf("      Input ");
```

```
  gotoxy(48,12);cprintf("      Input ");
  gotoxy(48,14);cprintf("      Input ");
  gotoxy(48,16);cprintf("      Input ");

  textbackground(1);
  for(Num='1';Num<='6';Num++) InputData(Com,Num);
}

////////////////////////////////////////////////////////////////////////
void ClearAll(int Com)
{
  char Num;

  textbackground(0);
  gotoxy(48,6); cprintf("      Clear ");
  gotoxy(48,8); cprintf("      Clear ");
  gotoxy(48,10);cprintf("      Clear ");
  gotoxy(48,12);cprintf("      Clear ");
  gotoxy(48,14);cprintf("      Clear ");
  gotoxy(48,16);cprintf("      Clear ");

  for(Num='1';Num<='6';Num++) ClearBuffers(Com,Num);

  delay(1000);
  gotoxy(48,6); cprintf("          0 ");
  gotoxy(48,8); cprintf("          0 ");
  gotoxy(48,10);cprintf("          0 ");
  gotoxy(48,12);cprintf("          0 ");
  gotoxy(48,14);cprintf("          0 ");
  gotoxy(48,16);cprintf("          0 ");
```

```
  textbackground(1);
}

////////////////////////////////////////////////////////////////////////
void ClearBuffers(int Com,char Num)
{
  while(!TxReady(Com));  delay(del_time); Send(Com,';');
  while(!TxReady(Com));  delay(del_time); Send(Com,Num);
  while(!TxReady(Com));  delay(del_time); Send(Com,'0');
  while(!TxReady(Com));  delay(del_time); Send(Com,'0');
  while(!TxReady(Com));  delay(del_time); Send(Com,':');
}

////////////////////////////////////////////////////////////////////////
void InputData(int Com,char Num)
{
  while(!TxReady(Com));  delay(del_time); Send(Com,';');
  while(!TxReady(Com));  delay(del_time); Send(Com,Num);
  while(!TxReady(Com));  delay(del_time); Send(Com,'1');
  while(!TxReady(Com));  delay(del_time); Send(Com,'0');
  while(!TxReady(Com));  delay(del_time); Send(Com,':');
}

////////////////////////////////////////////////////////////////////////
void ReadData(int Com,char Num)
{
  int  Error;                    // 0 = no error  1 = error
  long wait=100000;
  char ch;
```

```
while(!TxReady(Com));  delay(del_time); Send(Com,';');
while(!TxReady(Com));  delay(del_time); Send(Com,Num);
while(!TxReady(Com));  delay(del_time); Send(Com,'2');
while(!TxReady(Com));  delay(del_time); Send(Com,'0');
while(!TxReady(Com));  delay(del_time); Send(Com,':');

// start byte
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[0]=0xff; }
wait=Waiting;
while(((dat[0]=Read(Com))==0xff)&&(wait--));

// number machine
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[1]=0xff; }
wait=Waiting;
while(((dat[1]=Read(Com))==0xff)&&(wait--));

// total data
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[2]=0xff; }
wait=Waiting;
while(((dat[2]=Read(Com))==0xff)&&(wait--));

// data 1
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
```

```
if(wait<=0) { dat[3]=0xff; return; }
wait=Waiting;
while(((dat[3]=Read(Com))==0xff)&&(wait--));

// data 2
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[4]=0xff; return; }
wait=Waiting;
while(((dat[4]=Read(Com))==0xff)&&(wait--));

// data 3
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[5]=0xff; return; }
wait=Waiting;
while(((dat[5]=Read(Com))==0xff)&&(wait--));

// data 4
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[6]=0xff; return; }
wait=Waiting;
while(((dat[6]=Read(Com))==0xff)&&(wait--));

// data 5
wait=Waiting;
while((Read(Com)!=0xff)&&(wait--));
if(wait<=0) { dat[7]=0xff; return; }
wait=Waiting;
```

```
  while(((dat[7]=Read(Com))==0xff)&&(wait--));

  // data 6
  wait=Waiting;
  while((Read(Com)!=0xff)&&(wait--));
  if(wait<=0) { dat[8]=0xff; return; }
  wait=Waiting;
  while(((dat[8]=Read(Com))==0xff)&&(wait--));

  // sum of data
  wait=Waiting;
  while((Read(Com)!=0xff)&&(wait--));
  if(wait<=0) { dat[9]=0xff; }
  wait=Waiting;
  while(((dat[9]=Read(Com))==0xff)&&(wait--));

  // end byte
  wait=Waiting;
  while((Read(Com)!=0xff)&&(wait--));
  if(wait<=0) { dat[10]=0xff; }
  wait=Waiting;
  while(((dat[10]=Read(Com))==0xff)&&(wait--));
}

//////////////////////////////////////////////////////////////////////////
int RxReady(int Com)
{
  char ch;
  ch = inportb(Com+5);
  if(ch&&0x01) return(1);          // recieve ready
```

```
  else                    return(0);        // data not ready
}

////////////////////////////////////////////////////////////////////////
int TxReady(int Com)
{
  char ch;

  ch = inportb(Com+5);
  if(ch&&0x20)        return(1); // transmit ready
  else                return(0);        // transmit not ready
}

////////////////////////////////////////////////////////////////////////
unsigned char Read(int Com)
{
  return(inportb(Com));
}

////////////////////////////////////////////////////////////////////////
void Send(int Com,char ch)
{
  outport(Com,ch);
}

////////////////////////////////////////////////////////////////////////
int SelectCom(void)
{
  char ch,ch1=0;
  int  Com;
```

```
  do {
    gotoxy(2,25); printf("Select Port Com > "); ch = getch();
    switch(ch) {
                        case '1':gotoxy(20,25); putch(ch); Com = 0x3f8; ch1=ch; break;
                        case '2':gotoxy(20,25); putch(ch); Com = 0x2f8; ch1=ch; break;
                        case  13:if(ch1==0) break;
                           gotoxy(2,25);
                           printf("Connect port COM:%c at 9,600 boud.",ch1);
                           printf("                                    'Esc' to exit");
                           return(Com);
                        case 0x08:gotoxy(20,25); putch(' '); gotoxy(20,25); ch1=0; break;
                        case  27:return(0);
    }
  } while(1);
}

//////////////////////////////////////////////////////////////////////
void Connect(int Com)
{
  if(Com == 0x3f8) Com = 0;
  if(Com == 0x2f8) Com = 1;
  asm {
    mov         dx,Com
    mov         ah,0x00
    mov         al,0xe3
    int         0x14
  }
}
```

ภาคผนวก ค
คู่มือการใช้งาน

คู่มือการใช้งาน

ชุดตอบคำถามสำหรับการสอนทางไกล
ANSWERING MACHINE FOR LONG DISTANCE LEARNING

รูปที่ ค.1 ชุดตอบคำถามสำหรับการสอนทางไกล

ฟังก์ชั่นคีย์และหน้าที่การทำงานประกอบไปด้วยส่วนต่าง ๆ ดังนี้

คีย์บอร์ดเมตริกซ์ลำดับ 0-9 , * , #

1. คีย์ 0-9 การใช้งาน คีย์ 0-9 จะใช้เป็นชุดป้อนข้อมูล
2. คีย์ * การใช้งาน คีย์ * จะใช้เป็นการแก้ไขคำตอบที่เราต้องการใหม่โดยการกดคีย์ * แล้วทำการป้อนคำตอบใหม่
3. คีย์ # การใช้งาน คีย์ # จะใช้แทนคีย์ Enter เพื่อที่จะบันทึกข้อมูลจากคีย์ที่ตอบไปใน Buffer เพื่อเตรียมการส่งข้อมูลต่อไป

ขั้นตอนการตรวจสอบความพร้อมก่อนการใช้งาน

1. ต่อสายสัญญาณพอร์ตอนุกรมจากเครื่องเปลี่ยนสัญญาณ (Concentator) เข้าพอร์ตอนุกรมที่เครื่องคอมพิวเตอร์
2. ต่อสายสัญญาณจากเครื่องเปลี่ยนสัญญาณ (Concentator) เข้าตามชุดตอบคำถามแต่ละเครื่อง
3. จากนั้นทำการเปิดเครื่องคอมพิวเตอร์และเปิดเครื่องเปลี่ยนสัญญาณ แล้วก็จะปรากฏคำว่า Ready ที่ชุดตอบคำถาม

ฟังก์ชั่นคีย์ที่ใช้ในงานโปรแกรม

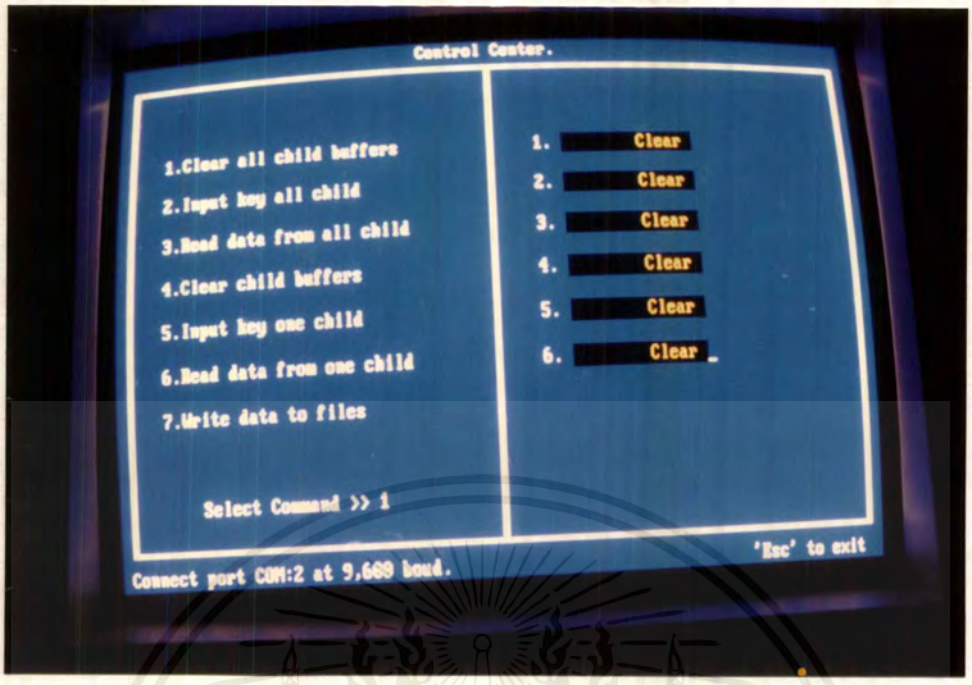
ขั้นแรกเมื่อเข้าสู่โปรแกรมให้ทำการเลือกพอร์ตสื่อสารอนุกรม จากนั้นก็จะปรากฏเมนูเพื่อทำการเลือกใช้งาน ดังนี้

1. การสั่งลบข้อมูลในชุดตอบคำถามทุกชุด
2. การอนุญาตให้ชุดตอบคำถามสามารถกดตอบคำถามได้
3. การอ่านข้อมูลจากชุดทุกชุดมาเก็บไว้
4. การสั่งลบข้อมูลทีละชุดโดยสามารถกำหนดชุดที่ต้องการลบได้
5. การอนุญาตให้ชุดตอบคำถามตอบได้โดยสามารถกำหนดชุดที่ต้องการได้
6. การอ่านข้อมูลโดยสามารถกำหนดชุดที่ต้องการได้
7. การนำข้อมูลที่ได้มาเก็บเป็นไฟล์

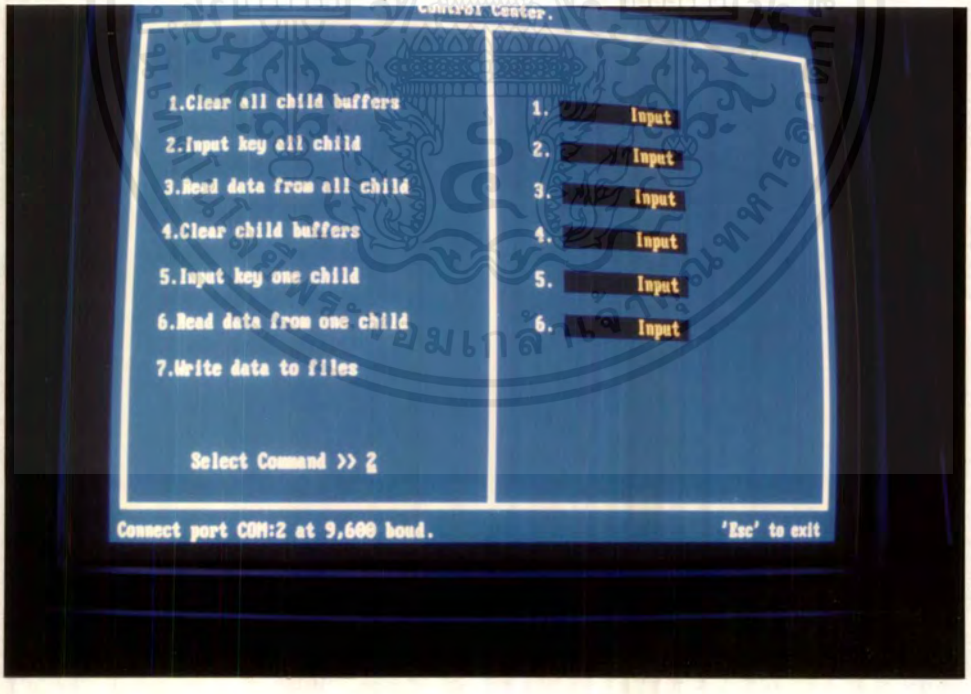
ESC ทำการยกเลิกคำสั่งและทำการออกจากโปรแกรม

ขั้นตอนการใช้งานมีอยู่ 2 รูปแบบ คือ

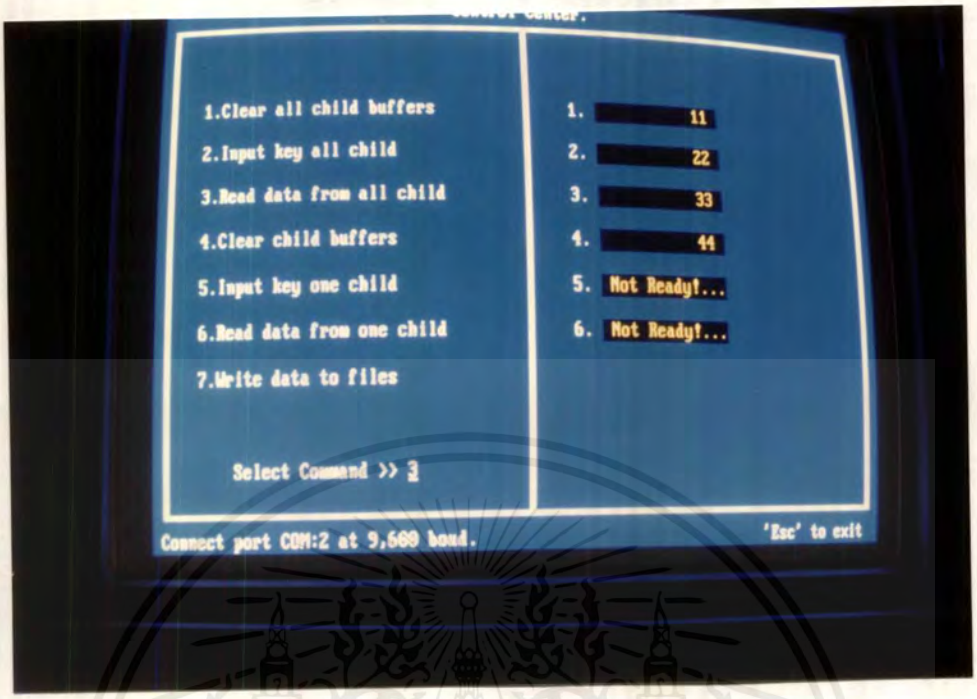
1. แบบอัตโนมัติ เมื่อต้องการลบข้อมูลที่ต้องการส่งจากชุดตอบคำถามทุกชุดให้เลือกหมายเลข 1 จากนั้นจะปรากฏคำว่า Clear ที่ชุดตอบคำถามทุกชุด และเมื่อต้องการป้อนข้อมูลจากทุกชุด ต่อจากนั้นให้เลือกหมายเลข 2 จะปรากฏคำว่า Input ที่ชุดตอบคำถามทั้งหมด และเมื่อต้องการข้อมูลจากชุดตอบคำถามทั้งหมด ให้เลือกที่กดหมายเลข 3 จะปรากฏคำว่า Send ชั่วขณะที่ทุกชุด แสดงว่าได้มีการส่งข้อมูลที่กดไปยังเครื่องคอมพิวเตอร์แล้ว และค่าที่ได้ก็จะปรากฏที่หน้าจอคอมพิวเตอร์



รูปที่ ค.2 หน้าจอการสั่งลบข้อมูลให้กับชุดตอบคำถามสำหรับการสอนทางไกล



รูปที่ ค.3 หน้าจอการสั่งป้อนข้อมูลให้กับชุดตอบคำถามสำหรับการสอนทางไกล



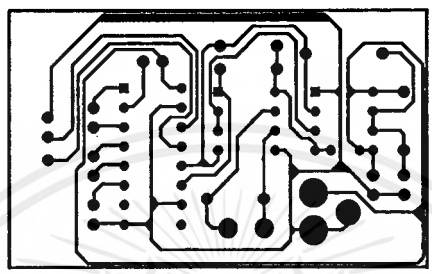
รูปที่ ค.4 หน้าจอการสั่งรับข้อมูลจากชุดตอบคำถามสำหรับการสอนทางไกล

2. แบบกำหนดเอง เมื่อต้องการลบข้อมูลที่ต้องการส่งจากชุดตอบคำถามเป็นชุด ให้เลือกหมายเลข 4 จากนั้นให้กดหมายเลขชุดที่ต้องการลบจะปรากฏคำว่า Clear ที่ชุดนั้น และเมื่อต้องการป้อนข้อมูลเป็นรายชุด ให้เลือกหมายเลข 5 จากนั้นใส่หมายเลขชุดที่ต้องการป้อนข้อมูล ก็จะปรากฏคำว่า Input ที่ชุดนั้น และเมื่อต้องการข้อมูลจากชุดตอบคำถามเป็นรายชุด ให้เลือกที่กดหมายเลข 6 และทำการเลือกหมายเลขชุดที่ต้องการ จากนั้นจะปรากฏคำว่า Send ชั่วขณะที่ชุดนั้น แสดงว่าได้มีการส่งข้อมูลที่กดไปยังเครื่องคอมพิวเตอร์แล้ว และค่าที่ได้ก็จะปรากฏที่หน้าจอคอมพิวเตอร์

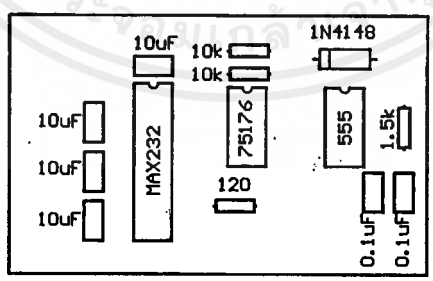
ขั้นตอนการบันทึกข้อมูลที่ป้อนมาเก็บไว้เป็นไฟล์

เมื่อได้ข้อมูลจากชุดตอบคำถามแล้ว จากนั้นต้องการเก็บเป็นไฟล์ก็ให้เลือกหมายเลข 7 ตามด้วยการกดคีย์บอร์ดอะไรก็ได้ ก็จะได้ไฟล์ ชื่อ ANSWER . ANS

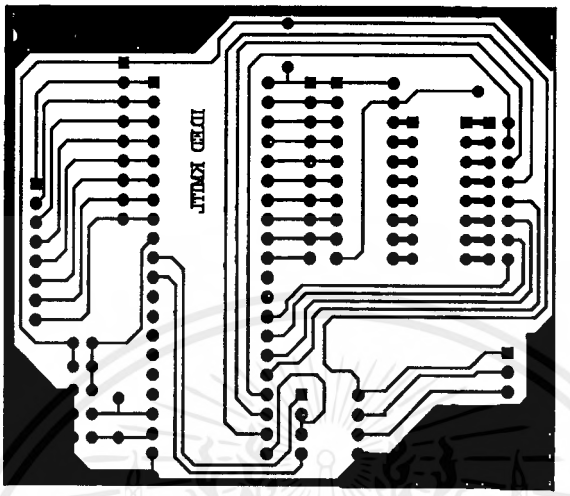
ภาคผนวก ง
ลายวงจรพิมพ์ และการวางอุปกรณ์บนแผ่นพิมพ์



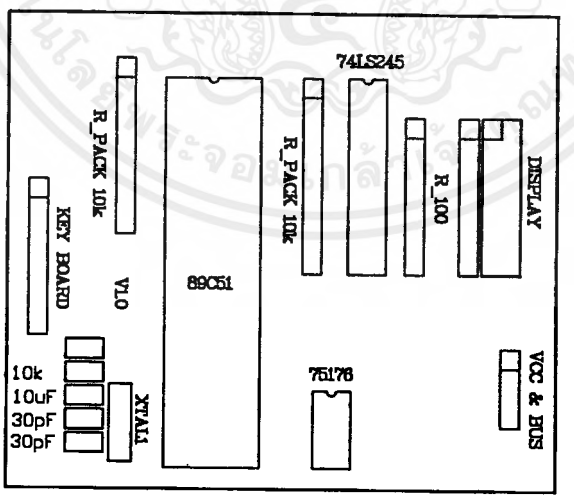
รูปที่ ง.1 ลายวงจรพิมพ์วงจรเปลี่ยนระดับสัญญาณ RS-485 เป็น RS-232



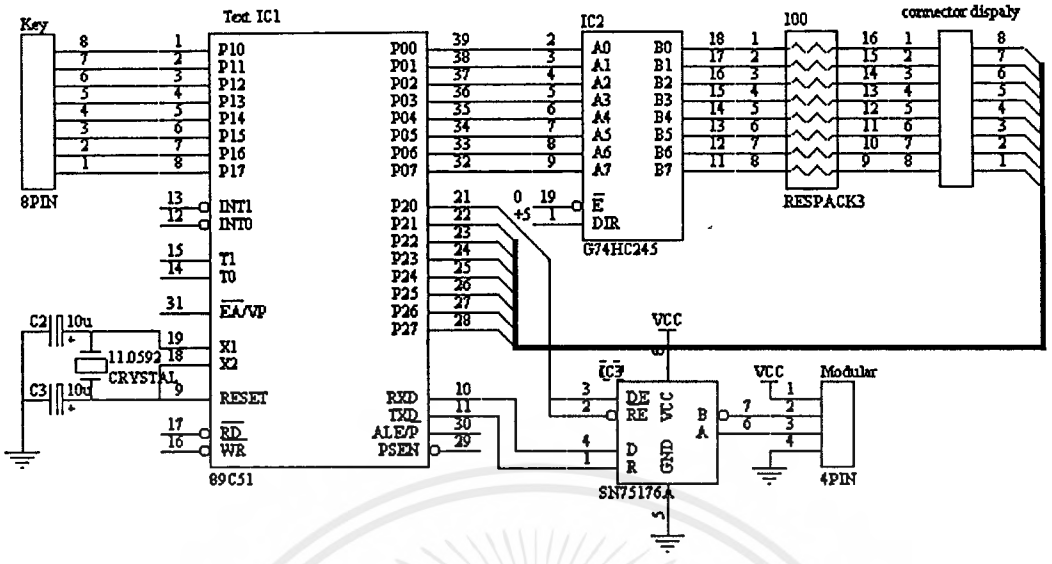
รูปที่ ง.2 การวางอุปกรณ์บนแผ่นวงจรพิมพ์วงจรเปลี่ยนระดับสัญญาณ RS-485 เป็น RS-232



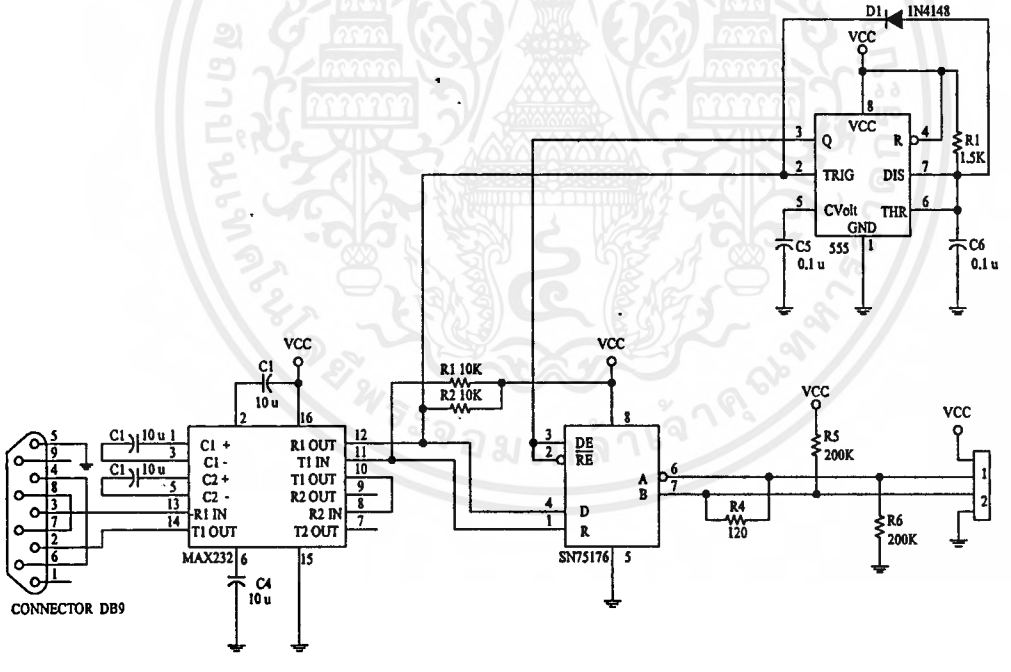
รูปที่ ง.3 ลายวงจรพิมพ์วงจรชุดตอบคำถาม



รูปที่ ง.4 การวางอุปกรณ์บนแผ่นวงจรพิมพ์วงจรชุดตอบคำถาม



รูปที่ ง.5 วงจรชุดตอบคำถาม



รูปที่ ง.6 วงจรเปลี่ยนระดับสัญญาณ RS-485 เป็น RS-232

ภาคผนวก จ
รายการอุปกรณ์

รายการอุปกรณ์

วงจรชุดส่งข้อมูล (แต่ละชุด) จากวงจรรูปที่ 3.4

อุปกรณ์	จำนวน
ไอซี AT89C51	1 ตัว
ไอซี 74HC245	1 ตัว
ไอซี 75176	1 ตัว
ตัวต้านทาน 10 kΩ	1 ตัว
ตัวต้านทาน 100 Ω	8 ตัว
ต้านทานแบบ แพ็ค 9 ขา 10 kΩ	1 ตัว
ตัวต้านทานแบบ แพ็ค 10 ขา 10 kΩ	1 ตัว
ตัวเก็บประจุ 10 μF	4 ตัว
ตัวเก็บประจุ 0.1 μF	1 ตัว
ตัวเก็บประจุ 220 μF	1 ตัว
คริสตอล 11.0592 MHz	1 ตัว
คีย์บอร์ดเมตริกซ์ ขนาด 3 X 4	1 ตัว
จอแสดงผล (7 Segment) 5 หลัก	5 ตัว
ขั้วต่อ	1 ตัว
สายนำสัญญาณ (ภายใน 4 เส้น)	10 เมตร
แผ่นลายวงจรพิมพ์	1 แผ่น

วงจรเปลี่ยนระดับสัญญาณ RS-232 เป็น RS-485 จากวงจรรูปที่ 3.7

อุปกรณ์	จำนวน
ตัวต้านทาน 10 kΩ	2 ตัว
ตัวต้านทาน 15 kΩ	1 ตัว
ตัวต้านทาน 120 kΩ	1 ตัว
ตัวเก็บประจุ 10 μF	4 ตัว
ตัวเก็บประจุ 0.1 μF	2 ตัว
ไดโอด IN4148	1 ตัว
ไอซี 75176	1 ตัว
ไอซี MAX232	1 ตัว
ไอซี NE 555	1 ตัว
แหล่งจ่ายไฟแบบสวิตชิ่ง	1 ชุด

ภาคผนวก ฉ
รายละเอียดข้อมูลและคุณสมบัติของอุปกรณ์

Features

- Compatible with MCS-51™ Products
- 4 Kbytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-Level Program Memory Lock
- 128 x 8-Bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low Power Idle and Power Down Modes

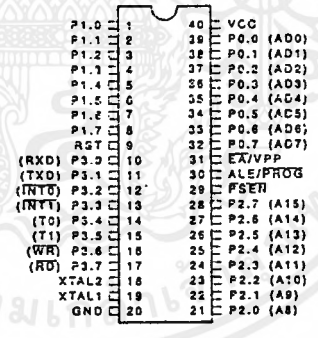
Description

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4 Kbytes of Flash Programmable and Erasable Read Only Memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ 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 Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

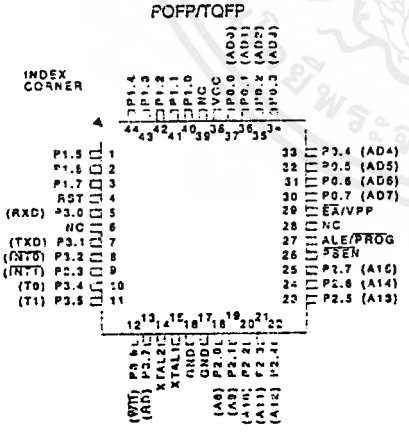
(continued)

Pin Configurations

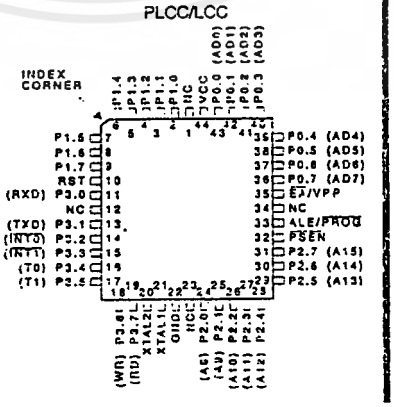
PDIP/Cerdip



PQFP/TQFP



PLCC/LCC

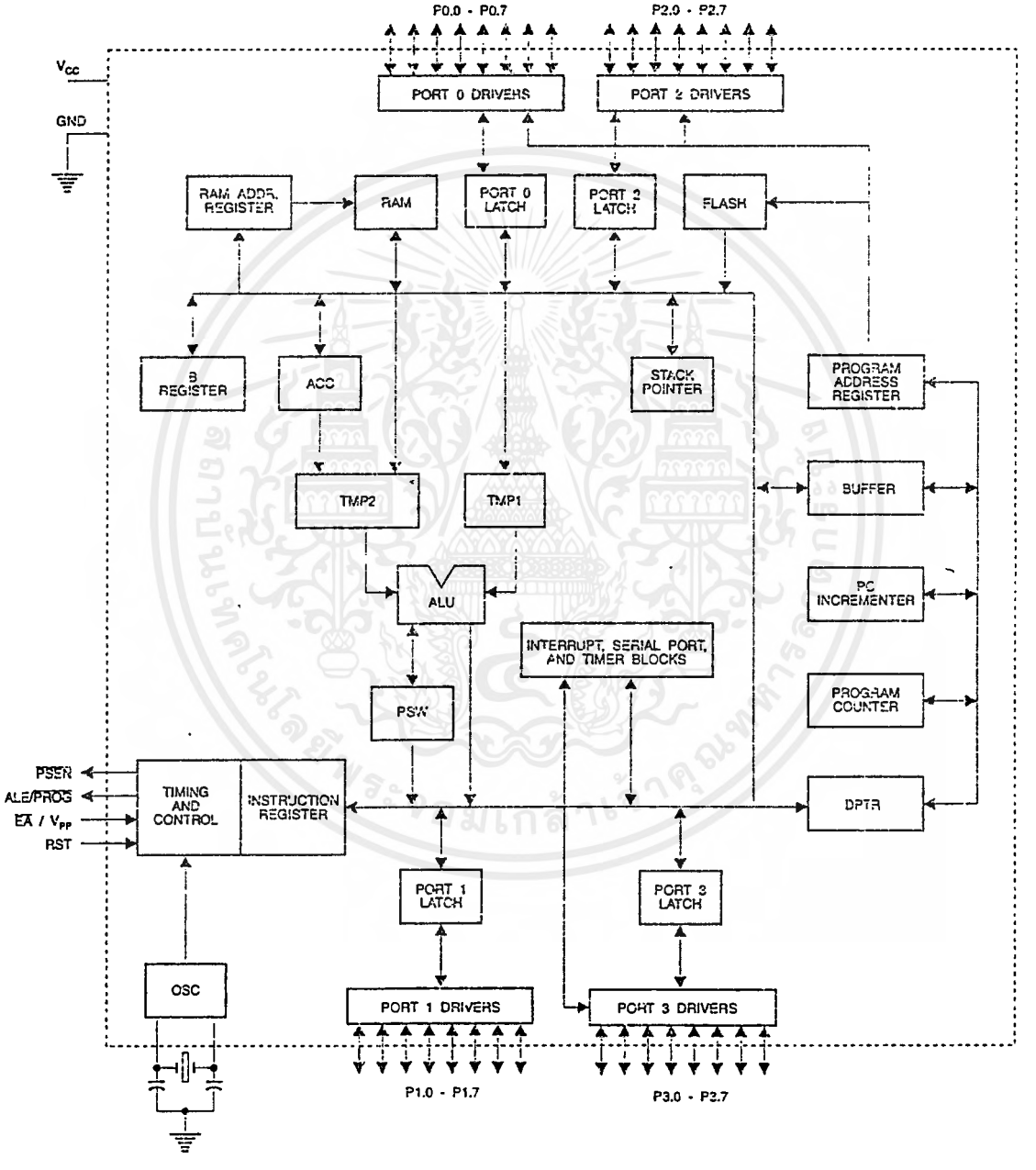


8-Bit Microcontroller with 4 Kbytes Flash

AT89C51

0265E

Block Diagram



Description (Continued)

The AT89C51 provides the following standard features: 4 Kbytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 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 hardware reset.

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 may 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.

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

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 it 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 AT89C51 as listed below:

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)

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

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable 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 weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

$\overline{PSEN}$

Program Store Enable is the read strobe to external program memory.

(continued)

Pin Description (Continued)

When the AT89C51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

EA/VPP

External Access Enable. 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, EA will be internally latched on reset.

EA should be strapped to VCC for internal program executions.

This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming, for parts that require 12-volt VPP.

XTAL1

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

XTAL2

Output from the inverting oscillator amplifier.

Oscillator Characteristics

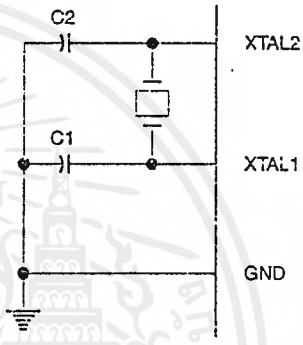
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. 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.

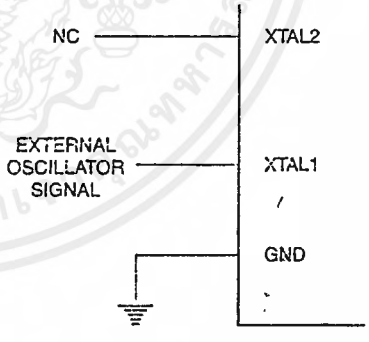
It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hard-

Figure 1. Oscillator Connections



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

Figure 2. External Clock Drive Configuration



Status of External Pins During Idle and Power Down

Mode	Program Memory	ALE	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

ware 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 is terminated by reset, the instruction following the one that invokes idle should not be one that writes to a port pin or to external memory.

Power 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. The only exit from power down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before Vcc is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Program Memory Lock Bits

On the chip are three lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below:

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. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

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 is disabled.
3	P	P	U	Same as mode 2, also verify is disabled.
4	P	P	P	Same as mode 3, also external execution is disabled.

Programming the Flash

The AT89C51 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (Vcc) program enable signal. The low voltage programming mode provides a convenient way to program the AT89C51 inside the user's system, while the high-voltage programming mode is compatible with conventional third party Flash or EPROM programmers.

The AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	V_{PP} = 12 V	V_{PP} = 5 V
Top-Side Mark	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Signature	(030H)=1EH (031H)=51H (032H)=FFH	(030H)=1EH (031H)=51H (032H)=05H

The AT89C51 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C51, the address, data and control signals should be set up according to the Flash programming mode table and Figures 3 and 4. To program the AT89C51, 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 12 V for the high-voltage programming mode.
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 1.5 ms. 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 AT89C51 features $\overline{Data}$ Polling to indicate the end of a write cycle. During a write cycle, an at-

(continued)

Programming the Flash (Continued)

tempted read of the last byte written will result in the complement of the written datum on PO.7: Once the write cycle has been completed, true data are valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

Ready/Busy: The progress of byte programming can also be monitored by the RDY/BSY output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 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 lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/PROG low for 10 ms. The code array is written with all "1"s. The chip erase operation must be executed before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H,

031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel
(031H) = 51H indicates 89C51
(032H) = FFH indicates 12 V programming
(032H) = 05H indicates 5 V programming

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased 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.

Flash Programming Modes

Mode		RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	EA/V_{PP}	P2.6	P2.7	P3.6	P3.7
Write Code Data		H	L	- _ -	H/12V(1)	L	H	H	H
Read Code Data		H	L	H	H	L	L	H	H
Write Lock	Bit - 1	H	L	- _ -	H/12V	H	H	H	H
	Bit - 2	H	L	- _ - (2)	H/12V	H	H	L	L
	Bit - 3	H	L	- _ -	H/12V	H	L	H	L
Chip Erase		H	L	- _ -	H/12V	H	L	L	L
Read Signature Byte		H	L	H	H	L	L	L	L

Notes: 1. The signature byte at location 032H designates whether V_{PP} = 12 V or V_{PP} = 5 V should be used to enable programming.

2. Chip Erase requires a 10 ms $\overline{PROG}$ pulse.

Figure 3. Programming the Flash

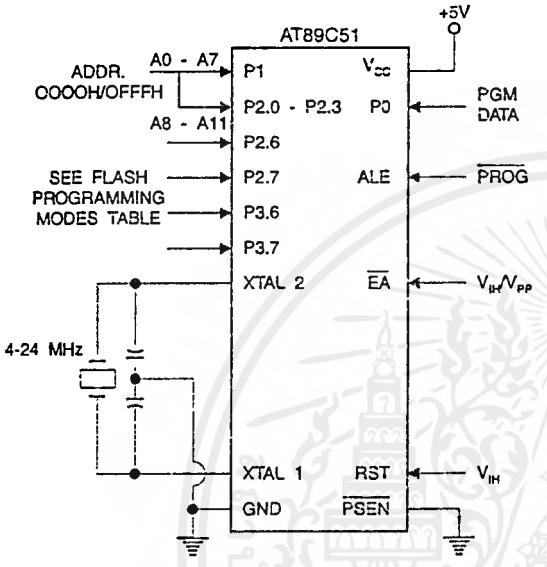
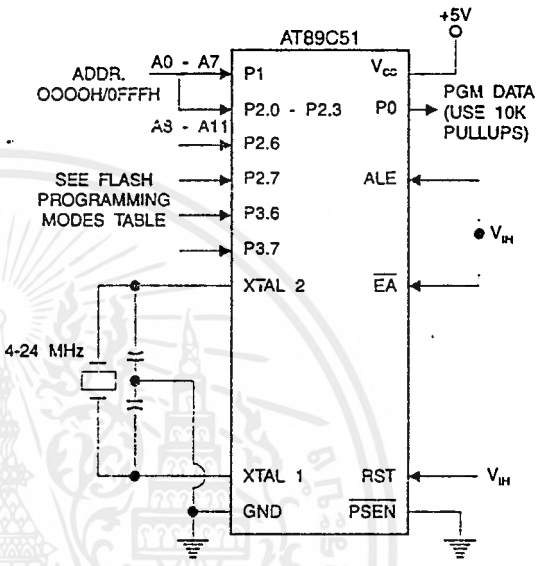


Figure 4. Verifying the Flash



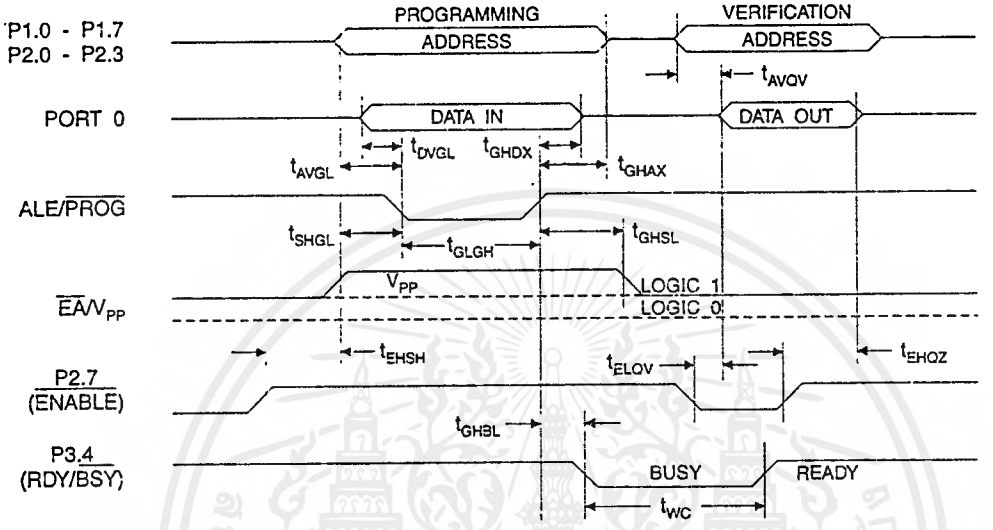
Flash Programming and Verification Characteristics

T_A = 21°C to 27°C, V_{CC} = 5.0 ± 10%

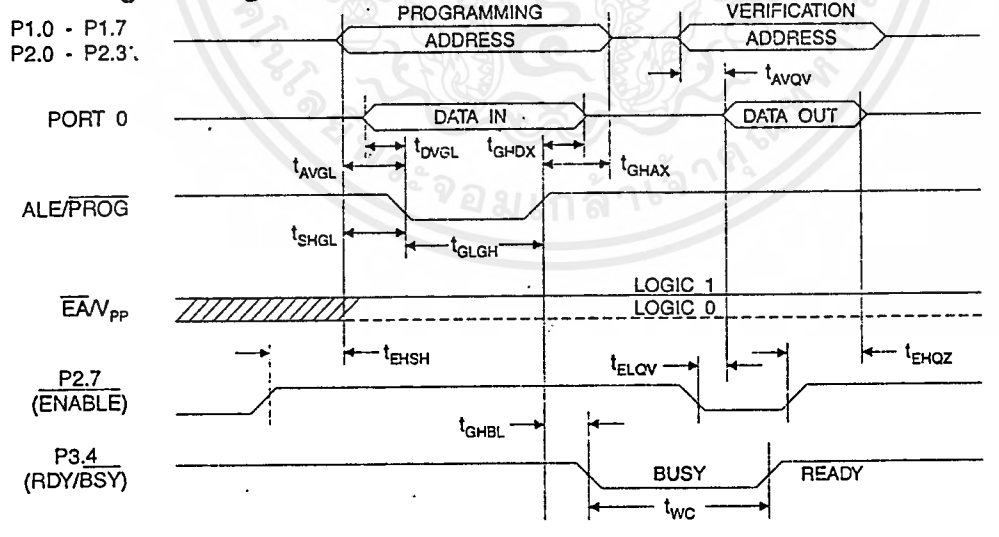
Symbol	Parameter	Min	Max	Units
V_{PP}(1)	Programming Enable Voltage	11.5	12.5	V
I_{PP}(1)	Programming Enable Current		1.0	mA
1/t_{CLCL}	Oscillator Frequency	4	24	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	48t_{CLCL}		
t_{GHAX}	Address Hold After $\overline{PROG}$	48t_{CLCL}		
t_{DVGL}	Data Setup 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}	48tCLCL		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}(1)	V_{PP} Hold After $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{AVQV}	Address to Data Valid		48tCLCL	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		48t_{CLCL}	
t_{EHQV}	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		2.0	ms

Note: 1. Only used in 12-volt programming mode.

Flash Programming and Verification Waveforms - High Voltage Mode



Flash Programming and Verification Waveforms - Low Voltage Mode



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	-1.0 V to +7.0 V
Maximum Operating Voltage	6.6 V
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.

D.C. Characteristics

T_A = -40°C to 85°C, V_{CC} = 5.0 V ± 20% (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, PSEN)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High Voltage (Ports 1,2,3, ALE, PSEN)	I_{OH} = -60 µA, V_{CC} = 5 V ± 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} = 5 V ± 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} = 0.45 V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	V_{IN} = 2 V		-650	µA
I_{LI}	Input Leakage Current (Port 0, EA)	0.45 < V_{IN} < V_{CC}		±10	µA
RRST	Reset Pulldown Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		20	mA
		Idle Mode, 12 MHz		5	mA
	Power Down Mode[2]	V_{CC} = 6 V		100	µA
		V_{CC} = 3 V		40	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 IOL for all output pins:71 mA
If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum VCC for Power Down is 2 V.

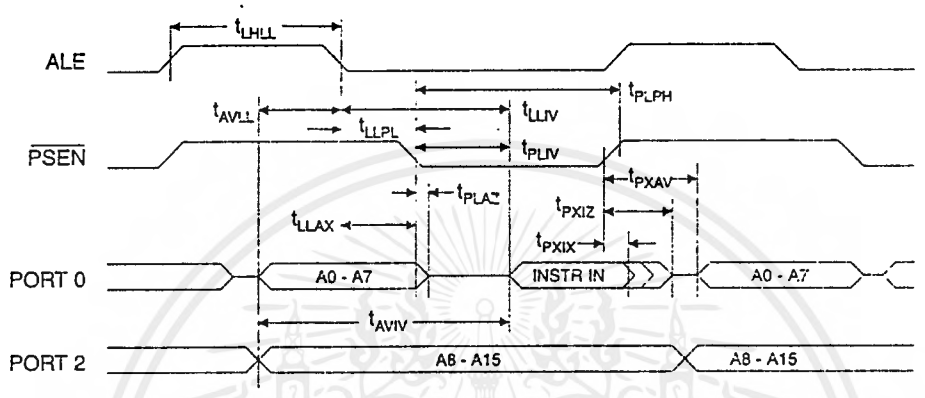
A.C. Characteristics

(Under Operating Conditions; Load Capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; Load Capacitance for all other outputs = 80 pF)

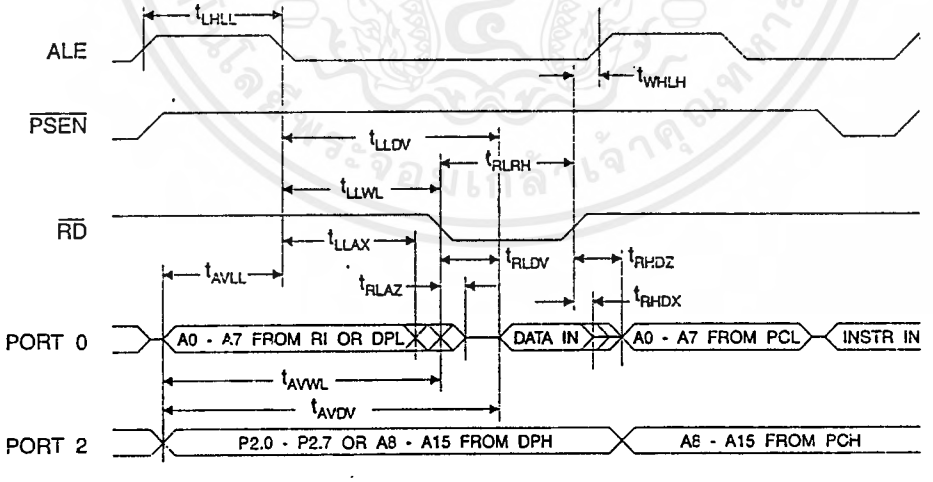
External Program and Data Memory Characteristics

Symbol	Parameter	12 MHz Oscillator Min	12 MHz Oscillator Max	16 to 24 MHz Oscillator Min	16 to 24 MHz Oscillator Max	Units
1/t_{CLCL}	Oscillator Frequency			0	24	MHz
t_{LHLL}	ALE Pulse Width	127		2t_{CLCL}-40		ns
t_{AVLL}	Address Valid to ALE Low	28		tCLCL-13		ns
t_{LLAX}	Address Hold After ALE Low	48		tCLCL-20		ns
t_{LLIV}	ALE Low to Valid Instruction In		233		4tCLCL-65	ns
t_{LLPL}	ALE Low to $\overline{PSEN}$ Low	43		t_{CLCL}-13		ns
t_{PLPH}	$\overline{PSEN}$ Pulse Width	205		3t_{CLCL}-20		ns
t_{PLIV}	$\overline{PSEN}$ Low to Valid Instruction In		145		3t_{CLCL}-45	ns
t_{PXIX}	Input Instruction Hold After $\overline{PSEN}$	0		0		ns
t_{PXIZ}	Input Instruction Float After $\overline{PSEN}$		59		t_{CLCL}-10	ns
t_{PXAV}	$\overline{PSEN}$ to Address Valid	75		t_{CLCL}-8		ns
t_{AVIV}	Address to Valid Instruction In		312		5t_{CLCL}-55	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		8tCLCL-150	ns
t_{AVDV}	Address to Valid Data In		585		9tCLCL-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}-20		ns
t_{QVWH}	Data Valid to $\overline{WR}$ High	433		7t_{CLCL}-120		ns
t_{WHQX}	Data Hold After $\overline{WR}$	33		t_{CLCL}-20		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}-20	t_{CLCL}+25	ns

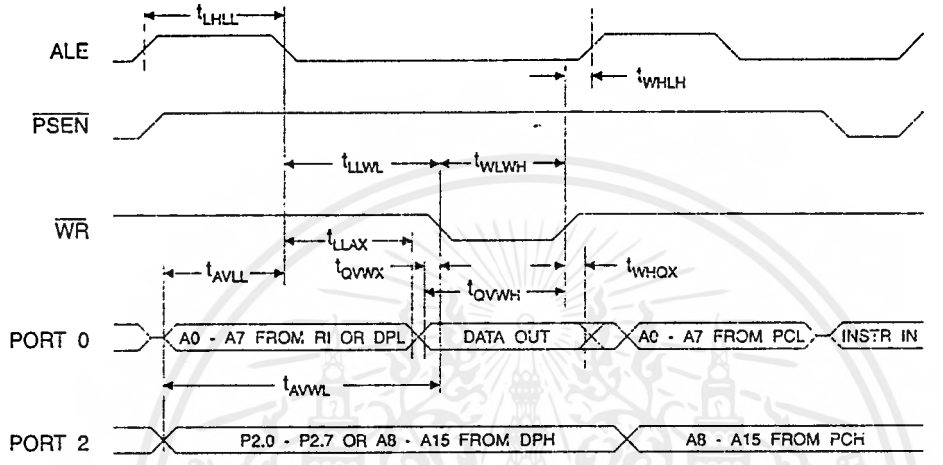
External Program Memory Read Cycle



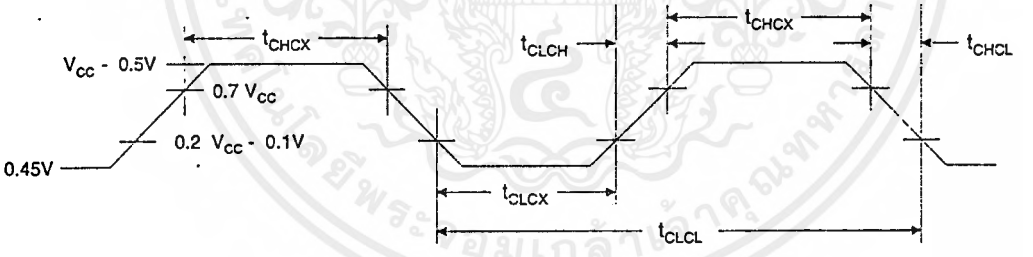
External Data Memory Read Cycle



External Data Memory Cycle



External Clock Drive Waveforms



External Clock Drive

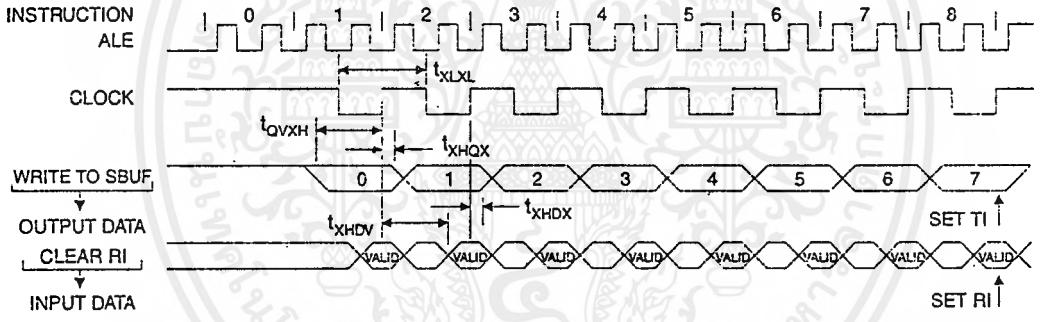
Symbol	Parameter	Min	Max	Units
1/t_{CLCL}	Oscillator Frequency	0	24	MHz
t_{CLCL}	Clock Period	41.6		ns
t_{CHCX}	High Time	15		ns
t_{CLCX}	Low Time	15		ns
t_{CLCH}	Rise Time		20	ns
t_{CHCL}	Fall Time		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

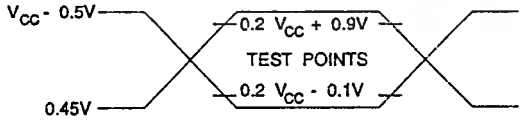
(V_{CC} = 5.0 V ± 20%; Load Capacitance = 80 pF)

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		$2t_{CLCL}$-33		ns
t_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		10tCLCL-133	ns

Shift Register Mode Timing Waveforms

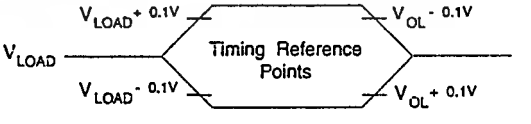


AC Testing Input/Output Waveforms[1]



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5 V for a logic 1 and 0.45 V 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
12	5 V ± 20%	AT89C51-12AC AT89C51-12JC AT89C51-12PC AT89C51-12QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C51-12AI AT89C51-12JI AT89C51-12PI AT89C51-12QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)
		AT89C51-12AA AT89C51-12JA AT89C51-12PA AT89C51-12QA	44A 44J 40P6 44Q	Automotive (-40°C to 125°C)
	5 V ± 10%	AT89C51-12DM AT89C51-12LM	40D6 44L	Military (-55°C to 125°C)
		AT89C51-12DM/883 AT89C51-12LM/883	40D6 44L	Military/883C Class B, Fully Compliant (-55°C to 125°C)
16	5 V ± 20%	AT89C51-16AC AT89C51-16JC AT89C51-16PC AT89C51-16QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C51-16AI AT89C51-16JI AT89C51-16PI AT89C51-16QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)
		AT89C51-16AA AT89C51-16JA AT89C51-16PA AT89C51-16QA	44A 44J 40P6 44Q	Automotive (-40°C to 125°C)
20	5 V ± 20%	AT89C51-20AC AT89C51-20JC AT89C51-20PC AT89C51-20QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C51-20AI AT89C51-20JI AT89C51-20PI AT89C51-20QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)

Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
24	5 V ± 20%	AT89C51-24AC AT89C51-24JC AT89C51-24PC AT89C51-24QC	44A 44J 44P6 44Q	Commercial (0°C to 70°C)
		AT89C51-24AI AT89C51-24JI AT89C51-24PI AT89C51-24QI	44A 44J 44P6 44Q	Industrial (-40°C to 85°C)

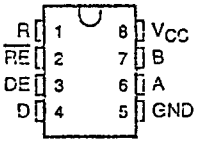
Package Type	
44A	44 Lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
40D6	40 Lead, 0.600" Wide, Non-Windowed, Ceramic Dual Inline Package (Cerdip)
44J	44 Lead, Plastic J-Leaded Chip Carrier (PLCC)
44L	44 Pad, Non-Windowed, Ceramic Leadless Chip Carrier (LCC)
40P6	40 Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44Q	44 Lead, Plastic Gull Wing Quad Flatpack (PQFP)

SN75176A
DIFFERENTIAL BUS TRANSCEIVER

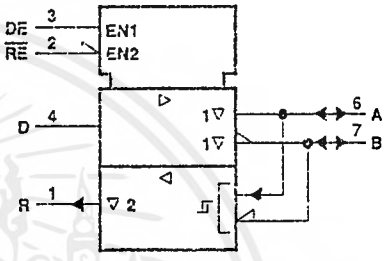
SLLS100A – JUNE 1984 – REVISED MAY 1995

- Bidirectional Transceiver
- Meets or Exceeds the Requirements of ANSI Standards EIA/TIA-422-B and ITU Recommendation V.11
- 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
- Operates From Single 5-V Supply
- Low Power Requirements

D OR P PACKAGE
(TOP VIEW)

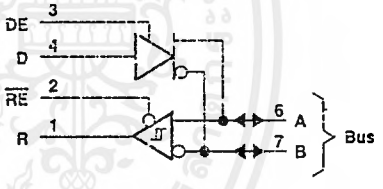


logic symbol†



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

logic diagram (positive logic)



description

The SN75176A differential bus transceiver is a monolithic integrated circuit designed for bidirectional data communication on multipoint bus-transmission lines. It is designed for balanced transmission lines and meets ANSI Standard EIA/TIA-422-B and ITU Recommendation V.11.

The SN75176A combines a 3-state differential line driver and a differential input line receiver, both of which operate from a single 5-V power supply. The driver and receiver have active-high and active-low enables, respectively, that can be externally connected together to function as a direction control. The driver differential outputs and the receiver differential inputs are connected internally to form differential input/output (I/O) bus ports that are designed to offer minimum loading to the bus whenever the driver is disabled or V_{CC} = 0. These ports feature wide positive and negative common-mode voltage ranges making the device suitable for party-line applications.

Function Tables

DRIVER

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

RECEIVER

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

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

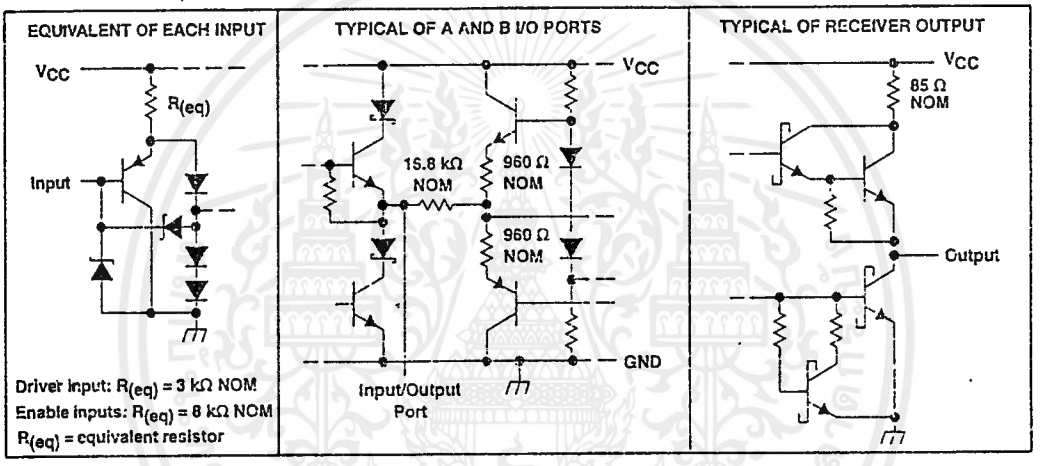
description (continued)

The driver is designed to handle loads 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 SN75176A 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 SN75176A is characterized for operation from 0°C to 70°C.

schematics of inputs and outputs



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

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

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

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

DISSIPATION RATING TABLE

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

recommended operating conditions

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

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

DRIVER SECTION

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

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{IK}	Input clamp voltage	$I_I = -18$ mA				−1.5	V
V_{OH}	High-level output voltage	$V_{IH} = 2$ V, $V_{IL} = 0.8$ V, $I_{OH} = -33$ mA			3.7		V
V_{OL}	Low-level output voltage	$V_{IH} = 2$ V, $V_{IL} = 0.8$ V, $I_{OH} = 33$ mA			1.1		V
$\lvert V_{OD1}\rvert$	Differential output voltage	$I_O = 0$				$2V_{OD2}$	V
$\lvert V_{OD2}\rvert$	Differential output voltage	$R_L = 100\ \Omega$,	See Figure 1	2	2.7		V
		$R_L = 54\ \Omega$,	See Figure 1	1.5	2.4		
$\Delta\lvert V_{OD}\rvert$	Change in magnitude of differential output voltage‡					±0.2	V
V_{OC}	Common-mode output voltage§	$R_L = 54\ \Omega$ or $100\ \Omega$, See Figure 1				3	V
$\Delta\lvert V_{OC}\rvert$	Change in magnitude of common-mode output voltage‡					±0.2	V
I_O	Output current	Output disabled, See Note 3	$V_O = 12$ V			1	mA
			$V_O = -7$ V			−0.8	
I_{IH}	High-level input current	$V_I = 2.4$ V				20	µA
I_{IL}	Low-level input current	$V_I = 0.4$ V				−400	µA
I_{OS}	Short-circuit output current	$V_O = -7$ V				−250	mA
		$V_O = V_{CC}$				250	
		$V_O = 12$ V				500	
I_{CC}	Supply current (total package)	No load	Outputs enabled		35	50	mA
			Outputs disabled		26	40	

† All typical values are at $V_{CC} = 5$ V and $T_A = 25°C$.
‡ $\Delta\lvert V_{OD}\rvert$ and $\Delta\lvert V_{OC}\rvert$ are the changes in magnitude of V_{OD} and V_{OC} respectively, that occur when the input is changed from a high level to a low level.
§ In ANSI Standard EIA/TIA-422-B, V_{OC}, which is the average of the two output voltages with respect to GND, is called output offset voltage, V_{OS}.
NOTE 3: This applies for both power on and off; refer to ANSI Standard EIA/TIA-422-B for exact conditions.

switching characteristics, $V_{CC} = 5$ V, $T_A = 25°C$

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{d(OD)}$	Differential-output delay time	$R_L = 60\ \Omega$, See Figure 3		40	60	ns
$t_{t(OD)}$	Differential-output transition time			65	95	ns
t_{PZH}	Output enable time to high level	$R_L = 110\ \Omega$, See Figure 4		55	90	ns
t_{PZL}	Output enable time to low level	$R_L = 110\ \Omega$, See Figure 5		30	50	ns
t_{PHZ}	Output disable time from high level	$R_L = 110\ \Omega$, See Figure 4		85	130	ns
t_{PLZ}	Output disable time from low level	$R_L = 110\ \Omega$, See Figure 5		20	40	ns

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 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 3	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			12			kΩ
I_{OS}	Short-circuit output current			−15		−85	mA
I_{CC}	Supply current (total package)	No load	Outputs enabled		35	50	mA
			Outputs disabled		26	40	

† 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 3: This applies for both power on and power off. Refer to ANSI Standard EIA/TIA-422-B 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} = −1.5 V to 1.5 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	30	ns
t_{PZL}	Output enable time to low level			12	30	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

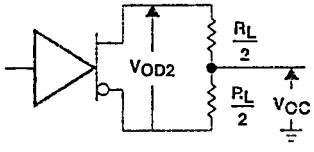


Figure 1. Driver V_{OD} and V_{OC}

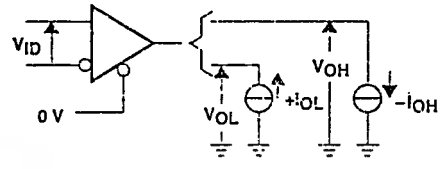
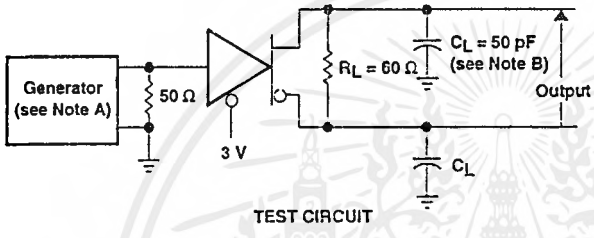


Figure 2. Receiver V_{OH} and V_{OL}



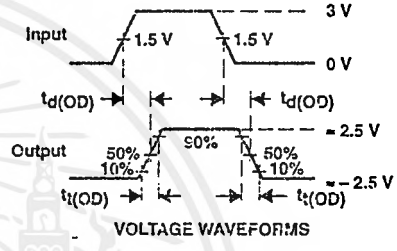
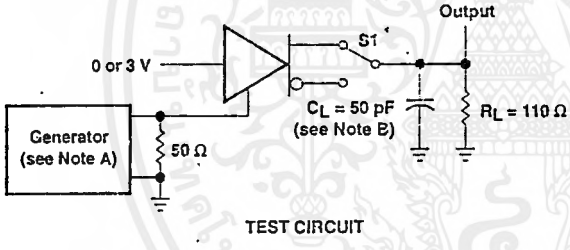


Figure 3. Driver Test Circuit and Voltage Waveforms



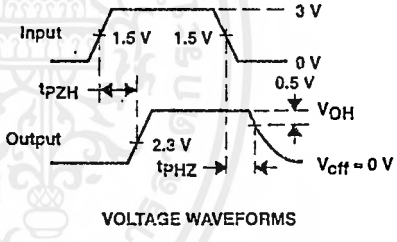
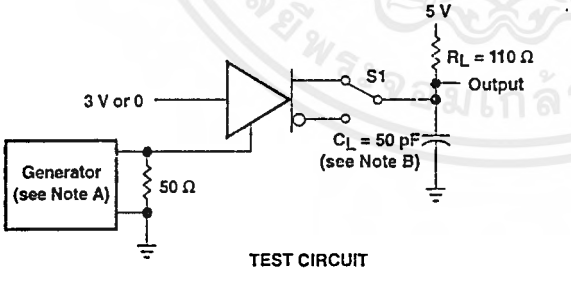
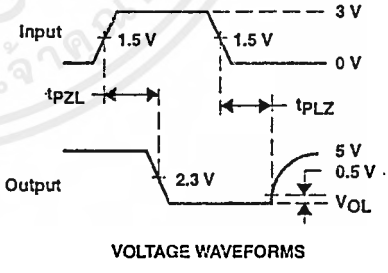


Figure 4. Driver Test Circuit and Voltage Waveforms





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

Figure 5. Driver Test Circuit and Voltage Waveforms

PARAMETER MEASUREMENT INFORMATION

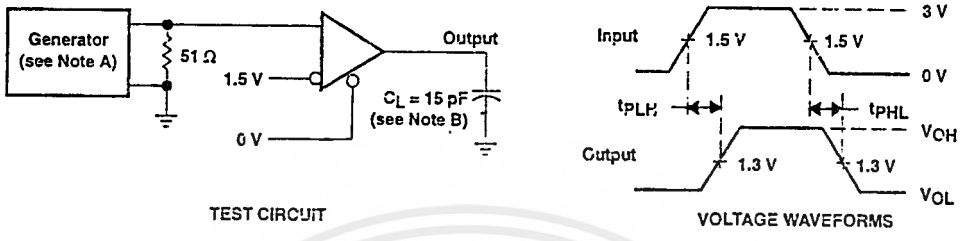
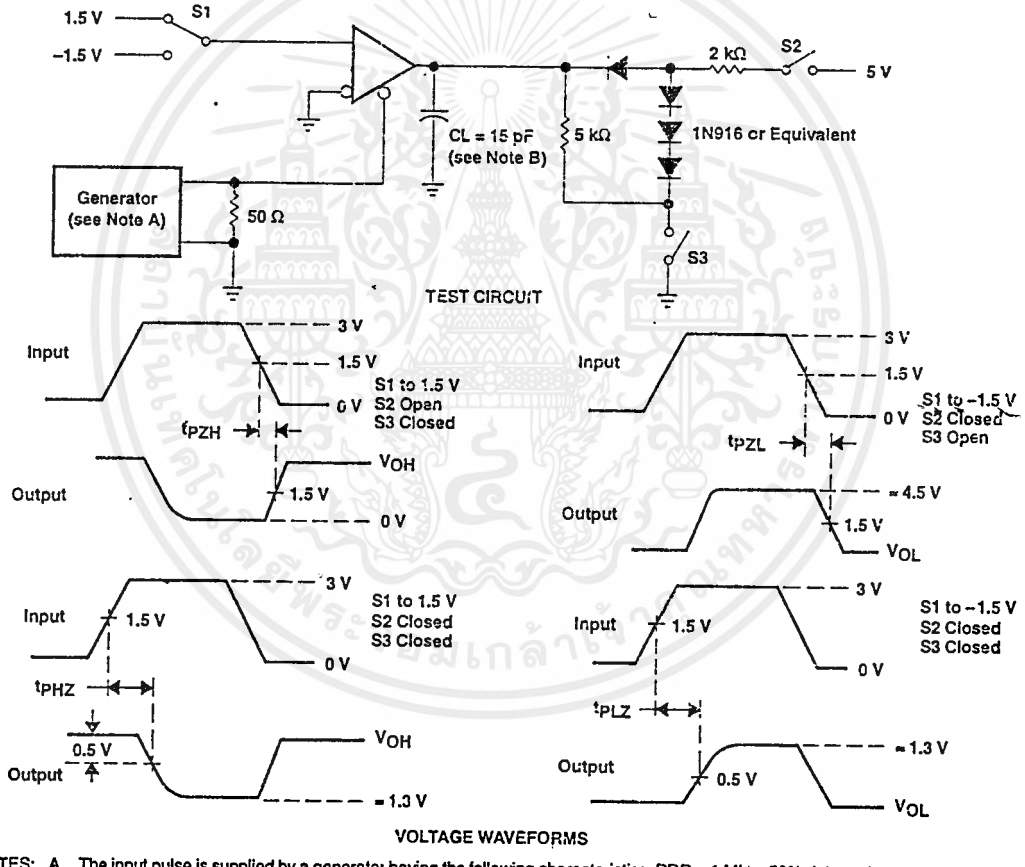


Figure 6. Receiver Test Circuit and Voltage Waveforms



NOTES: A. The input pulse is supplied by a generator having the following characteristics: PRR = 1 MHz, 50% duty cycle, $t_r \leq 6$ ns, $t_f \leq 5$ ns, $Z_O = 50\ \Omega$.
B. C_L includes probe and jig capacitance.

Figure 7. Receiver Test Circuit and Voltage Waveforms

TYPICAL CHARACTERISTICS

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

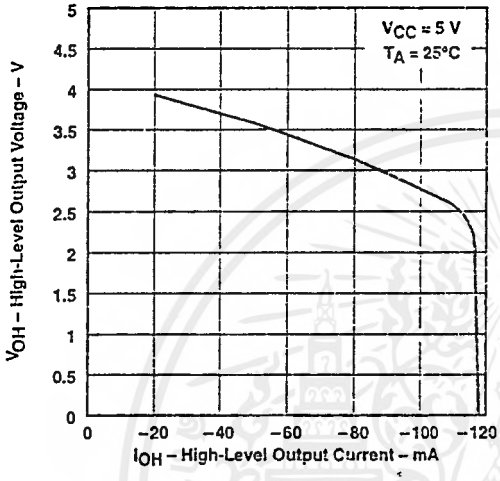


Figure 8

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

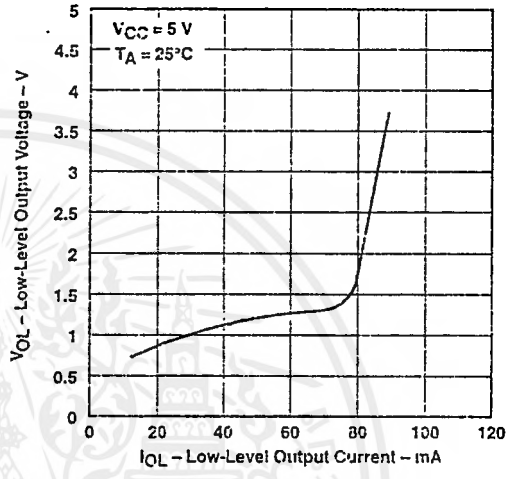


Figure 9

DRIVER
DIFFERENTIAL OUTPUT VOLTAGE
vs
OUTPUT CURRENT

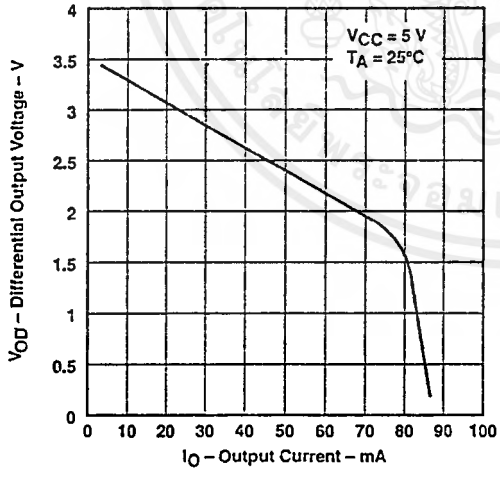


Figure 10

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

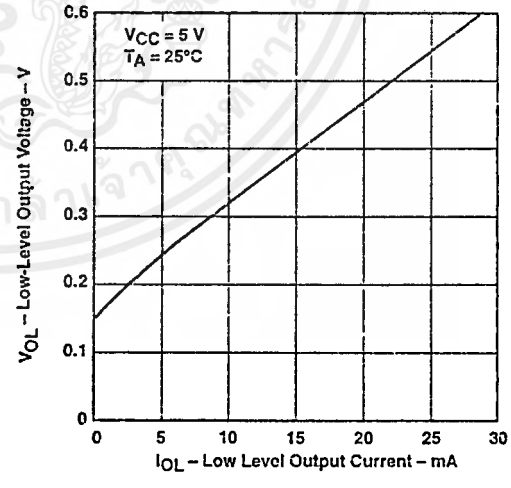


Figure 11

TYPICAL CHARACTERISTICS

RECEIVER
LOW-LEVEL OUTPUT VOLTAGE
vs
FREE-AIR TEMPERATURE

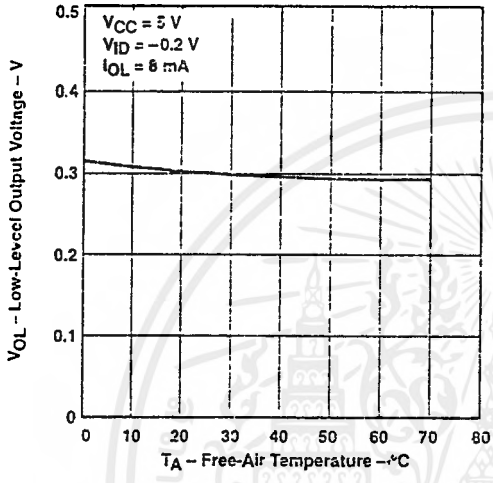


Figure 12

RECEIVER
OUTPUT VOLTAGE
vs
ENABLE VOLTAGE

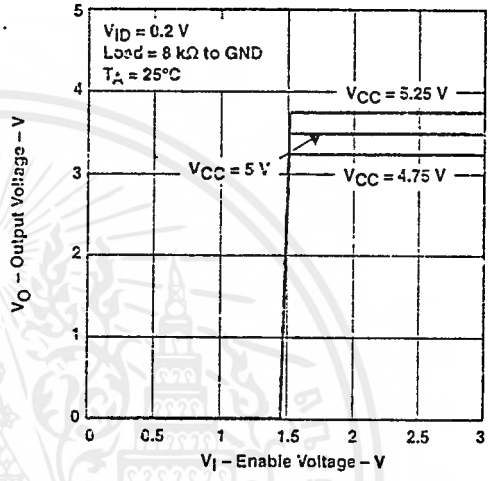


Figure 13

RECEIVER
OUTPUT VOLTAGE
vs
ENABLE VOLTAGE

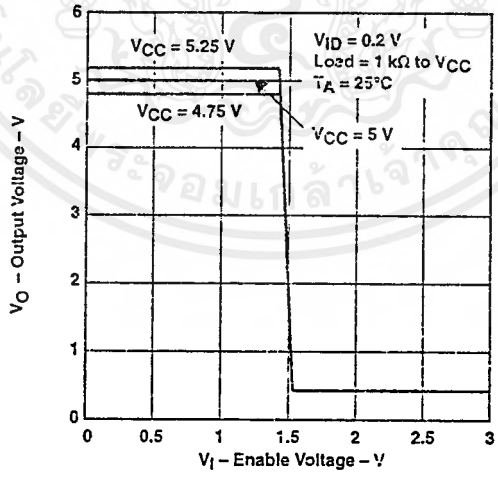
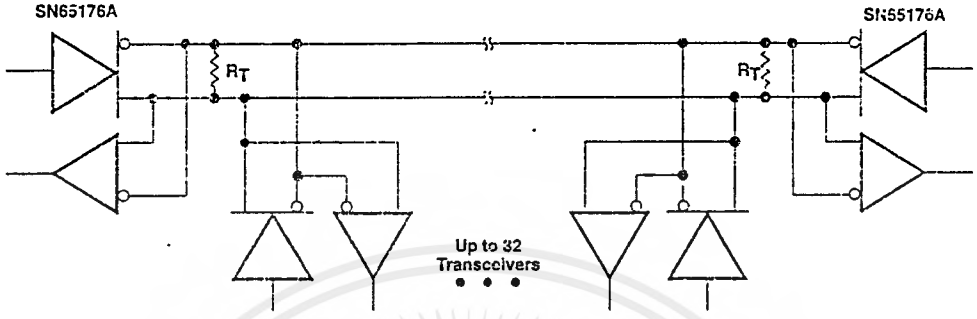


Figure 14

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 15. Typical Application Circuit

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software 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, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

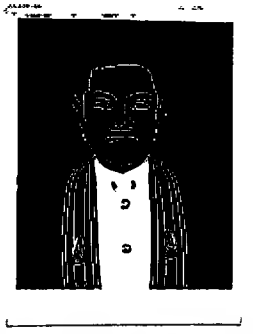
In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI 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.

บรรณานุกรม

1. เซมิคอนดัคเตอร์อิเล็กทรอนิกส์. คิจิตอล/ไมโครคอมพิวเตอร์, กรุงเทพมหานคร: ซีเอ็ดยูเคชั่น, 2540
2. จิรศักดิ์ เหลืองอุไร. การสื่อสารอนุกรมบน PC, กรุงเทพมหานคร: ซีเอ็ดยูเคชั่น, 2538
3. น.ต. ฉัตรชัย สุมามาลย์. การสื่อสารข้อมูลคอมพิวเตอร์และระบบเครือข่าย, กรุงเทพมหานคร: ด่านสุทธาการพิมพ์, 2521
4. สุนทร วิทูสุรพจน์. การใช้งานไมโครคอนโทรลเลอร์ตระกูล 8051, กรุงเทพมหานคร: : ซีเอ็ดยูเคชั่น, 2537
5. GIS-GROUP. เครือข่ายคอมพิวเตอร์, กรุงเทพมหานคร: สำนักพิมพ์ฟิสิกส์เซ็นเตอร์, 2536

ประวัติผู้แต่ง

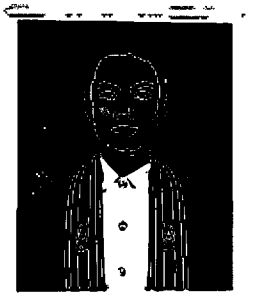


ชื่อผู้ทำปริญญานิพนธ์	นายกิตติพงษ์ ปานาพุฒ
วันเดือนปีเกิด	27 มิถุนายน 2520
สถานที่เกิด	จังหวัดหนองบัวลำภู
ภูมิลำเนาเดิม	จังหวัดหนองบัวลำภู
ที่อยู่ปัจจุบัน	45 หมู่ 10 ต.นาด่าน อ.สุวรรณคูหา จ.หนองบัวลำภู 39270
โทรศัพท์	042 - 314250

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนบ้านกุดฮู – ชุมภูทอง
มัธยมตอนต้น	โรงเรียนบ้านกุดดินจี่พิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคอุดรธานี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคอุดรธานี
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุสาหกรรม
ผลงานที่ได้รับรางวัล	อันดับ 1 การแข่งขันทักษะวิชาชีพระดับ ปวส. ภาคตะวันออกเฉียงเหนือ ปีการศึกษา2539
ทุนการศึกษา	-
คติพจน์	ความสำเร็จไม่มาด้วยความบังเอิญ แต่เกิดจากความพยายาม

ประวัติผู้แต่ง

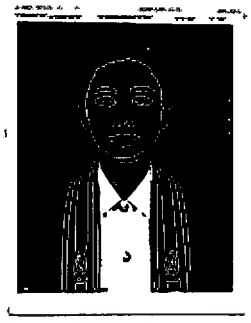


ชื่อผู้ทำปริญญานิพนธ์	นางสาวพรวิไล สุขมาก
วันเดือนปีเกิด	11 สิงหาคม 2519
สถานที่เกิด	จังหวัดสิงห์บุรี
ภูมิลำเนาเดิม	จังหวัดสิงห์บุรี
ที่อยู่ปัจจุบัน	94/35 ต.บางพุทรา อ.เมือง จ.สิงห์บุรี 16000
โทรศัพท์	036 - 512294

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนอนุบาลสิงห์บุรี
มัธยมศึกษาตอนต้น	โรงเรียนสิงห์บุรี
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคสิงห์บุรี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคสิงห์บุรี
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	ทุนกู้ยืมเพื่อการศึกษา
คติพจน์	คิดที่จะทำอะไรแล้วต้องทำให้ดีที่สุด

ประวัติผู้แต่ง



ชื่อผู้ทำปริญญานิพนธ์	นางสาวศิริพร มิขำ
วันเดือนปีเกิด	8 มกราคม 2520
สถานที่เกิด	จังหวัดนนทบุรี
ภูมิลำเนาเดิม	จังหวัดนนทบุรี
ที่อยู่ปัจจุบัน	97/2 หมู่ 11 ต.ตลาดขวัญ อ.เมือง จ.นนทบุรี 11000
โทรศัพท์	02 - 9689627

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนการัญศึกษา
มัธยมศึกษาตอนต้น	โรงเรียนนนทบุรีพิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	-
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตนนทบุรี
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	ทุนกู้ยืมเพื่อการศึกษา
คติพจน์	ทำวันนี้ให้ดีที่สุด

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายอุกฤษณ์ ธัญญานนท์
วันเดือนปีเกิด	16 มีนาคม 2519
สถานที่เกิด	จังหวัดสงขลา
ภูมิลำเนาเดิม	จังหวัดสงขลา
ที่อยู่ปัจจุบัน	3 หมู่ 6 ต.รัตภูมิ อ.ควนเนียง จ.สงขลา 90110
โทรศัพท์	076 - 272361

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนเทศบาล 4 (วัดคลองเรียน)
มัธยมศึกษาตอนต้น	โรงเรียนเชิงทะเลวิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคสุราษฎร์ธานี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคหาดใหญ่
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	คุณไม่ใช่ผม ผมไม่ใช่คุณ คุณอย่ามาตามผม และผมไม่ตามคุณ