ระบบเก็บข้อมูลระดับความดังเสียง

Sound Level Indication System

โดย

นายทีฆนนท์ ภู่สุโข

นายธนู หาญถนอม

นายธิติ ส่งทอง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

ระบบเก็บข้อมูลระดับความดังเสียง

Sound Level Indication System

โดย

นายทีฆนนท์	ภู่สุโข	45010307
นายธนู	หาญถนอม	45010334
นายธิติ	ส่งทอง	45010348

อาจารย์ที่ปรึกษา

ดร.พรชัย ทรัพย์นิธิ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **ระบบเก็บข้อมูลระดับความดังเสียง**

(Sound Level Indication System)

ผู้จัดทำ

1. นายทีฆนนท์	**ภู่สุโข**	**45010307**
2. นายธนู	**หาญถนอม**	**45010334**
3. นายธิติ	**ส่งทอง**	**45010348**

.....อาจารย์ที่ปรึกษา

(ดร.พรชัย ทรัพย์นิธิ)

ระบบเก็บข้อมูลระดับความดังเสียง

(Sound Level Indication System)

โดย	นายทีฆนนท์	ภู่สุโข	45010307
	นายธนู	หาญถนอม	45010334
	นายธิติ	ส่งทอง	45010348

อาจารย์ที่ปรึกษา ดร.พรชัย ทรัพย์นิธิ

บทคัดย่อ

โครงงานนี้เป็นการสร้างระบบการเก็บข้อมูลระดับความดังของเสียง โดยข้อมูลที่รับได้จากไมโครโฟนจะถูกแปลงเป็นข้อมูลทางไฟฟ้าส่งไปเก็บในระบบฐานข้อมูลในคอมพิวเตอร์ผ่านการสื่อสารแบบไร้สายโดยใช้คลื่นวิทยุ

ABSTRACT

This project is intended to construct the sound level indication system. The system provides measured data and stores it in a database system via wireless communication.

สารบัญ

สารบัญภาพ

สารบัญตาราง

บทที่ 1
บทนำ

ปัจจุบันมลภาวะของเสียงมีผลกระทบต่อมนุษย์อย่างมาก ดังนั้นจึงได้มีการกำหนดระดับของเสียงที่ทำอันตรายต่อมนุษย์ไว้หลายระดับขึ้นอยู่กับ ระยะเวลาในการรับฟังที่จะทำให้เกิดอันตรายได้ จากการใช้เครื่องวัดระดับสัญญาณเสียงแบบเดิมผู้ใช้ต้องเฝ้ารอวัดตามช่วงเวลาที่กำหนดวัด ซึ่งในการวัดที่ต้องการวัดผลแบบสุ่มวัดต่อเนื่อง ที่ต้องการผลการวัดสุ่มต่อเนื่องเป็นเวลานานๆจะเป็นการยุ่งยากและเสียเวลา ดังนั้นจึงได้มีการพัฒนาระบบการวัดระดับเสียงตามแบบที่มีอยู่เดิม เป็นเครื่องวัดระดับสัญญาณเสียงที่สามารถวัดระดับสัญญาณเสียงได้ ตามการตั้งช่วงเวลาในการวัดและบันทึก ผู้ใช้สามารถกำหนดช่วงเวลาการวัดและบันทึกได้ เป็นผลให้เกิดความสะดวกในการเก็บข้อมูลและสะดวกในการวัดที่ไม่ต้องเฝ้าวัด ซึ่งโครงงานเครื่องวัดระดับเสียงนี้ใช้ไมโครโปรเซสเซอร์เป็นตัวควบคุม คุณสมบัติการทำงานของเครื่อง สามารถแสดงผลการวัดระดับเสียงรบกวนเป็นหน่วย dBA โดยแสดงออกบนจอแสดงผล LCD สามารถแสดงผลการวัดที่บันทึกเก็บจากข้อมูลที่วัดและบันทึกไว้โดยสามารถเรียกข้อมูลที่เก็บไว้แล้วออกมาดูได้ และสามารถถ่ายโอนข้อมูลให้กับคอมพิวเตอร์ผ่านทาง PORT RS-232 เพื่อใช้ในการเก็บข้อมูลและนำผลข้อมูลการวัดระดับสัญญาณไปใช้งานตามความต้องการต่อไป

บทที่2
ทฤษฎีและหลักการ

2.1 เครื่องมือวัดระดับเสียง (Sound Level Meter)

เครื่องมือที่ใช้วัดระดับเสียง คือ Sound Level Meter ประกอบด้วยไมโครโฟนที่รับสัญญาณหลายทิศทาง (Nondirectional Microphone) , ชุดปรับค่าความดังของเสียง , วงจรขยาย , ชุดแสดงผล และชุดปรับเปลี่ยนความถี่ ค่าของระดับเสียงที่อ่านได้จากมิเตอร์จะอยู่ในรูปของค่า Root mean square(RMS) ที่ระดับเปรียบเทียบ ดังนั้น เครื่องมือชนิดนี้จะไม่วัดค่าออกมาเป็นค่าสูงสุด (Peak Level)

มิเตอร์โดยทั่วๆไปจะตอบสนองกับ 2 ช่วงได้แก่ ความเร็วสูง และความเร็วต่ำ ถ้าเสียงที่เราต้องการวัดมีระดับสูงๆต่ำๆควรปรับไปที่ความเร็วสูง ส่วนเสียงที่มีความถี่คงที่ควรปรับไปที่ความเร็วต่ำ โดยทั่วๆไปชุดช่วยปรับ (Weighting Network) จะมีอยู่ 3 ระดับให้เลือกใช้งาน คือ A , B และ C ซึ่งคุณสมบัติของความถี่ที่ตอบสนองจะได้ดังรูป 2.1 ชุดทั้ง 3 จะเป็นตัวเลือกค่าประมาณกับหูมนุษย์ที่ระดับเสียงแตกต่างกัน หูของมนุษย์จะไม่ไวต่อเสียงที่มีความสูงมากๆและความถี่ต่ำมากๆ ดังนั้นจึงจำเป็นต้องมีชุดช่วยปรับ

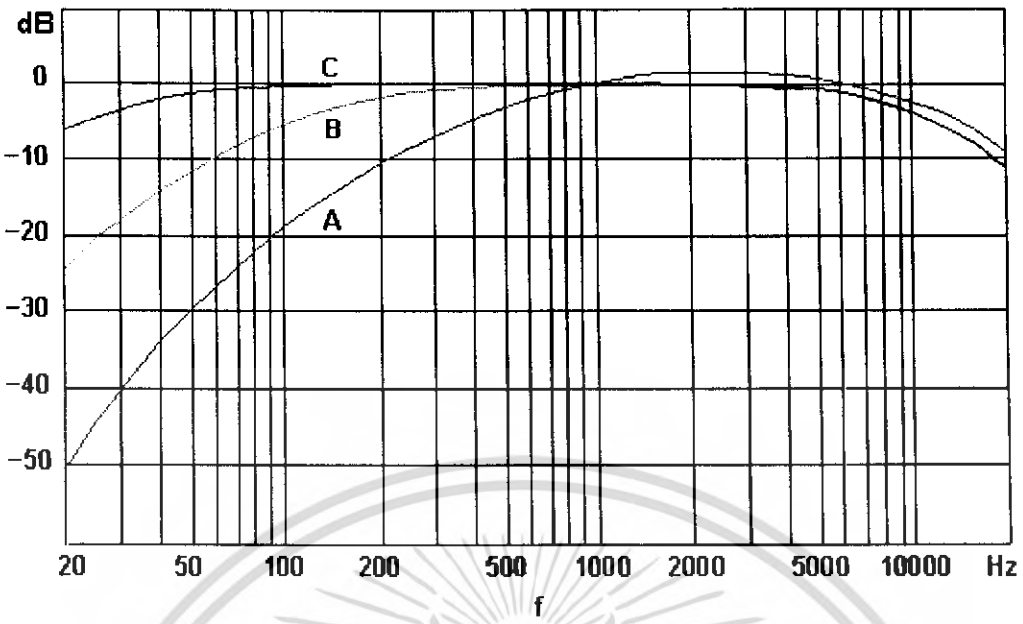
Weighting Network A จะใช้เมื่อเสียงระดับต่ำกว่า 55 dB

Weighting Network B สำหรับเสียงระดับระหว่าง 55 dB และ 85 dB

Weighting Network C สำหรับระดับเสียง 85 dB

ข้อแตกต่างที่สำคัญของค่าที่อ่านได้จาก สเกล A และ C ถ้าเป็น C Network ค่าที่ได้จะเรียบ (Flat) ค่าเสียงที่อ่านได้จะเรียกว่า Sound Pressure Level ส่วน A Network เคอร์ฟจะตกมากที่ความถี่ต่ำ ซึ่งจะคล้ายกับเสียงของหูมนุษย์ ค่าที่อ่านได้จะเรียกว่า Sound Level หรือเทียบกับ dBA การวัดระดับเสียง สามารถวัดได้ทั้งสเกล A หรือสเกล C การวัดที่สำคัญที่สุดคือ การวัดจากสเกล A

ถ้าเครื่องมือวัดระดับเสียงมีวงจรขยายที่มีความถี่ตอบสนองเรียบตลอดย่าน จาก 20 – 20,000 Hz เมื่อรวมกับการใช้ตำแหน่ง A , B และ C ค่าที่อ่านได้จะขึ้นอยู่กับไมโครโฟนที่ใช้



รูปที่ 2.1 สเกลถ่วงสัญญาณแบบ A , B และ C [1]

Frequency	Curve A	Curve B	Curve C
Hz	dB	dB	dB
16	-56.7	-28.5	- 8.5
31.5	-39.4	-17.1	- 3.0
63	-26.2	- 9.3	- 0.8
125	-16.1	- 4.2	- 0.2
250	- 8.6	-1.3	0
500	- 3.2	- 0.3	0
1000	0	0	0
2000	1.2	- 0.1	- 0.2
4000	1.0	- 0.7	- 0.8
8000	- 1.1	- 2.9	- 3.0
16000	- 6.6	- 8.4	- 8.5

ตารางที่ 2.1 แสดงสเกลถ่วงสัญญาณแบบ A , B และ C [1]

ปัจจุบันนี้มีมิเตอร์วัดระดับเสียงอยู่ 2 มาตรฐาน คือ American Standard Specification For General – Purpose Sound Level Meter , S 1 – 4 – 1971 และ International Electro technical Commission Publication 179 , Precision Sound Level Meter (IEC 179 – 1965)

2.2 ไมโครโฟน (Microphone)

อุปกรณ์ไมโครโฟน ทำหน้าที่เปลี่ยนคลื่นเสียง (Sound wave) หรืออากาศจากแหล่งกำเนิดเสียง เช่น เสียงพูด เสียงเพลง เสียงเครื่องดนตรี เป็นต้น ให้เป็นสัญญาณไฟฟ้า ไหลไปตามสายไมโครโฟนสู่ภาคขยายเสียง ไมโครโฟนเป็นส่วนที่สำคัญที่สุดของเครื่องมือวัดระดับเสียงในโครงงานชิ้นนี้ เนื่องจากความแม่นยำและความแน่นอนขึ้นอยู่กับประเภทของไมโครโฟน ปัจจัยที่สำคัญในการเลือกใช้ชนิดของไมโครโฟน ได้แก่ ผลการตอบสนองทางความถี่ (Frequency Response) ความไว (Sensitivity) และทิศทางการรับเสียง (Directionality) เป็นต้น

ประเภทของไมโครโฟน เมื่อแบ่งตามวัสดุที่ใช้ในไมโครโฟน มี 6 ชนิด คือ

2.2.1 ไมโครโฟนชนิดคาร์บอน (Carbon Microphone) ไมโครโฟนชนิดนี้ให้เสียงที่มีคุณภาพไม่ค่อยดี ปัจจุบันใช้ในเครื่องโทรศัพท์เท่านั้น

2.2.2 ไมโครโฟนชนิดคริสตัล (Crystal Microphone) ไมโครโฟนที่มีราคาถูก น้ำหนักเบาแต่ไม่ทนต่อสภาพความร้อน หรือความชื้นสูง เพราะอาจทำให้คริสตัลเสื่อมได้ ไมโครโฟนแบบนี้ให้กำลังไฟฟ้าออกมาสูง และสามารถรักษาสภาพสัญญาณได้ดีจึงไม่ต้องอาศัยหม้อแปลง (Transformer) ในตัวของไมโครโฟนช่วยแต่อย่างใด สามารถส่งสัญญาณไปยังเครื่องขยายเสียงได้โดยตรง สามารถใช้สายไมโครโฟนต่อยาวออกไปได้ไม่เกิน 25 ฟุต เพราะถ้าพ่วงสายยาวกว่านี้จะทำให้มีสัญญาณอื่นมารบกวนได้ และทำให้สัญญาณจากไมโครโฟนอ่อนลงมาก

2.2.3 ไมโครโฟนชนิดเซรามิค (Ceramic Microphone) มีลักษณะการออกแบบหรือหลักการทำงานคล้ายกับไมโครโฟนชนิดคริสตัล ต่างกันที่วัสดุเซรามิคมีคุณภาพดีกว่าคริสตัล เพราะทนทานต่อการเปลี่ยนแปลงของอุณหภูมิและความชื้นมากกว่า

2.2.4 ไมโครโฟนชนิดคอนเดนเซอร์ (Condenser Microphone) เป็นไมโครโฟนที่กำลังนิยมใช้อยู่ในปัจจุบัน สามารถรับเสียงได้ไวมาก มีราคาแพงและมักติดอยู่กับเครื่องบันทึกเสียงทั่ว ๆ ไป

2.2.5 ไมโครโฟนชนิดริบบอน (Ribbon or Velocity Microphone) เป็นไมโครโฟนที่บอบบางเสียง่าย ไม่มีไดอะแฟรม การทำงานอาศัยการสั่นสะเทือนของแผ่นริบบอน มีลักษณะบางเบา และขึงตึงอยู่ระหว่างแม่เหล็กถาวรกำลังสูงและจะทำงานทันทีเมื่อได้รับการสั่นสะเทือนเป็นไมโครโฟนที่มีคุณภาพสูงและควบคุมสัญญาณได้ดีที่สุด (Highest Fidelity) แต่ไม่ค่อยนิยมใช้กันมาก เพราะมีข้อเสียคือ ไม่เหมาะต่อ

งานสถานที่ แม้แต่เสียงลมพัดก็จะรับเสียงเอาไว้หมด อาจแก้ได้โดยใช้วัสดุกันลม เป็นกระบอกฟองน้ำสวมครอบแต่ก็ไม่ได้ผลนัก นอกจากนี้ยังมีปัญหาอื่น ๆ อีก เช่น สัญญาณไฟฟ้าที่ได้ออกมาค่อนข้างต่ำ (Low Output) ต้องใช้เครื่องขยายเสียงที่มีกำลังแรง และ คุณภาพสูง ถ้าพูดใกล้มาก เสียงลมหายใจจะกลบเสียงที่พูด ไมโครโฟนชนิดนี้ไม่นิยมใช้นอกสถานที่ มักพบในสถานีส่งวิทยุ โทรทัศน์และบันทึกเสียง

2.2.6 ไมโครโฟนชนิดไดนามิค (Dynamic Microphone) เป็นชนิดที่ได้รับความนิยมมาก เนื่องจากให้คุณภาพเสียงดีเหมือนธรรมชาติ มีความทนทาน เหมาะสมกับการกระจายเสียงหรือระบบเสียงหลายประเภท แต่ราคาค่อนข้างสูง

2.3 อิมพีแดนซ์ (Impedance) ค่าอิมพีแดนซ์ คือตัวเลขที่แสดงค่าความต้านทานของไมโครโฟนที่เกิดขึ้นขณะที่มีสัญญาณไฟฟ้าความถี่เสียง หรือกระแสสลับไหลผ่านโดยมีหน่วยเป็นโอห์ม แบ่งเป็น 2 ประเภท คือ

2.3.1 อิมพีแดนซ์สูง หรือมีค่าความต้านทานสูง (High Impedance) จะมีค่าอยู่ในช่วง 5,10,50 หรืออาจถึง 100 กิโลโอห์ม (KQ) จะให้กำลังของสัญญาณออกมาต่ำ (Low Power Output) มีเสียงรบกวนได้ง่าย เช่นเสียงฮัม ยิ่งถ้าต่อสายยาว ๆ หรือเกินกว่า 25 ฟุต ก็ยิ่งทำให้สูญเสียกำลังของสัญญาณมากขึ้น คุณภาพของเสียงจะลดลงด้วย ใช้ต่อร่วมกับเครื่องขยายเสียงโดยต่อช่องที่ช่อง High Impedance

2.3.2 อิมพีแดนซ์ต่ำหรือมีค่าความต้านทานต่ำ(Low Impedance) มีค่าอิมพีแดนซ์อยู่ในช่วง 200 ถึง 600 โอห์มซึ่งมีคุณภาพดีให้กำลังของสัญญาณออกสูง (High Power Output) ไม่มีเสียงรบกวนสามารถใช้กับสายยาว ๆ ได้แต่จะมีความไวในการรับเสียงต่ำใช้ต่อร่วมกับเครื่องขยายเสียงที่ช่อง Low Impedance

2.4 ผลการตอบสนองความถี่ (Frequency Response) ของเสียง คือความสามารถของไมโครโฟนในการรับความถี่ของคลื่นเสียงได้กว้างและมีความเรียบมากน้อย ซึ่งไมโครโฟนแต่ละชนิดก็จะออกแบบมาเพื่อใช้ในลักษณะงานต่าง ๆ กัน ฉะนั้น จึงมีความสามารถในการตอบสนองความถี่ต่าง ๆกัน มีหน่วยเป็นเฮิรตซ์ (Hertz: Hz) เช่น ไมโครโฟน สำหรับพูดในที่ชุมชน ประกาศ สั่งงาน การเรียนการสอนในห้องเรียน จะใช้ช่วงการตอบสนองความถี่ต่ำ และแคบ เช่น 300-5,000 เฮิรตซ์ แต่ถ้าต้องการคุณภาพของเสียงเรียบและแยกความถี่ได้กว้างขึ้น ควรอยู่ในช่วง 70-10,000 เฮิรตซ์ ถ้าต้องการคุณภาพของเสียงที่ดีเยี่ยมนอกจากเสียงพูดแล้ว ยังมีเสียงดนตรีด้วย ควรต้องใช้ไมโครโฟนที่ให้ผลตอบสนองความถี่ที่กว้างและเก็บความถี่ได้ละเอียดยิ่งขึ้น ควรอยู่ในช่วง 50-15,000 เฮิรตซ์ แต่ราคาก็จะค่อนข้างแพง

2.5 ความไวของไมโครโฟน (Microphone Sensitivity)

คุณสมบัติที่สำคัญที่สุดของไมโครโฟน คือ ความไวในการรับสัญญาณ ความไวของไมโครโฟนจะแสดงหน่วยเป็น dB / μbar เมื่อ dBV คือ ค่าระดับแรงดันขาออกเป็น dB เทียบกับ

1V ที่กำลังอัดของคลื่นเสียง 1 μbar (หรือ 0.1 N/m^2) ความไวของไมโครโฟนโดยทั่วๆไปจะอยู่ในย่าน -50 ถึง -125 dBV / μbar ลักษณะโดยทั่วไปไมโครโฟนตัวใหญ่ๆจะมีความไวสูง แต่ความถี่ตอบสนองต่ำ และไมโครโฟนแบบเล็กๆจะมีความไวต่ำ แต่ความถี่ตอบสนองสูง

ถ้าไมโครโฟนที่ใช้มีความไว (Sensitivity) 0.1 V / μbar
แรงดันที่แท้จริงที่ได้จากไมโครโฟนสามารถคำนวณได้จากความไวเป็น V / μbar

$$E(V) = \frac{E}{0.1}\mu bar$$

$$0.1\text{V} = 1\mu\text{bar}$$

$$E(V) = \frac{E}{0.1}\times 1\mu bar = P$$

$$dB(SPL) = 20\log\frac{P}{2\times 10^{-4}}$$

เมื่อ $E(V)$ คือ แรงดันเป็นโวลท์คิดจากกำลังอัด 1 μbar

P คือ ความดันของเสียงสัมพันธ์กับค่า SPL

$dB(SPL)$ คือ เดซิเบลที่เปรียบเทียบกับ $2\times 10^{-4}\ \mu bar$

Examples	Sound Pressure Level dBSPL	Sound Pressure $N/m^2 = Pa$	Sound Intensity $watts/m^2$
30 m from jet aircraft	140	200	100
Threshold of pain	130	63.2	10
Threshold of discomfort	120	20	1
Chainsaw 1m distance	110	6.3	0.1
Disco 1 m from speaker	100	2	0.01
Diesel truck (10 m away)	90	0.63	0.001
Kerbside of busy road	80	0.2	0.0001
Vacuum cleaner, distance 1 m	70	0.063	0.00001
Conversational speec, 1m	60	0.02	0.000001
Average home	50	0.0063	0.0000001
Quiet library	40	0.002	0.00000001
Quiet bedroom at night	30	0.00063	0.000000001
Background in TV studio	20	0.0002	0.0000000001
Rustling leaf	10	0.000063	0.00000000001
Threshold of hearing	0	0.00002	0.000000000001

ตารางที่ 2.2 แสดงระดับความดังเสียงในสถานที่ต่างๆ [2]

2.6 มาตรฐานเสียง

2.6.1 สถานประกอบการ

มาตรฐานความปลอดภัย ตามประกาศกระทรวงมหาดไทย เรื่องความปลอดภัยในการทำงานเกี่ยวกับสภาวะแวดล้อมกำหนดไว้ในช่วงเวลาทำงานต่อเนื่อง7-8 ชั่วโมง ในสถานประกอบการต้องมีระดับเสียงเฉลี่ยไม่เกิน 90 เดซิเบล เอ

2.6.2 สิ่งแวดล้อมของชุมชน

เกณฑ์มาตรฐานตามประกาศคณะกรรมการสิ่งแวดล้อมแห่งชาติ ฉบับที่ 15 (พ.ศ.2540) ซึ่งกำหนดมาตรฐานระดับเสียง เฉลี่ย 24 ชั่วโมง ไม่เกิน 70 เดซิเบล เอ ส่วนระดับเสียงกลางวัน-กลางคืน โดยองค์กรปกป้องและคุ้มครองสิ่งแวดล้อม ประเทศสหรัฐอเมริกา ไม่เกิน 55 และ 45 เดซิเบล เอ ระดับความดังของเสียงสูงสุด ไม่เกิน 115 เดซิเบล เอ

2.7 มาตรฐานเหตุรำคาญด้านเสียงจากสถานประกอบการ

- ชุมชนและบ้านพักอาศัย

ระดับความดังของเสียง ต้องมีค่ามากกว่าระดับเสียงพื้นฐาน ไม่เกิน 10 เดซิเบล เอ

ค่าสูงสุดของระดับเสียงรบกวน ที่เกิดจากยานพาหนะ ชนิดต่างๆ ของมาตรฐานสากลที่หลายๆ ประเทศในโลกยอมรับมีตัวอย่างดังนี้

- จักรยานติดเครื่องยนต์ (MOPEDS) 70-80dB(A)
- รถจักรยานยนต์ (MOTORCYCLES) 80-90 dB(A)
- รถยนต์ส่วนบุคคล (PRIVATE CAR) 80-90 dB(A)
- รถบรรทุกของหนัก (TRUCKS AND BUSSES) 82-92 dB(A)
- รถไถนา (TRACTORS) 85-95 dB(A)
- รถจักร (LOCOMOTIVES) 89-95 dB(A)
- ยานพาหนะอื่นๆหรือล้อเลื่อน (OTHER VEHICLES) 82-92 dB(A)

2.8 ไมโครคอนโทรลเลอร์ ตระกูล MCS-51

2.8.1 คุณสมบัติของไมโครคอนโทรลเลอร์ ตระกูล MCS-51

คุณสมบัติของไมโครคอนโทรลเลอร์ MCS-51 ที่สำคัญๆมีดังนี้

- ต้องการแหล่งจ่ายไฟ 5โวลต์ เพียงชุดเดียว

- มีหน่วยความจำ สำหรับเก็บโปรแกรมควบคุมการทำงานอยู่ภายในชิปจำนวน 4 กิโลไบต์ (เบอร์ 8031, 8032 ไม่มีหน่วยความจำส่วนนี้ ส่วนเบอร์ 8052 มีหน่วยความจำส่วนนี้ 8 กิโลไบต์ และเบอร์ 83C51FB จะมีหน่วยความจำส่วนนี้รวมทั้งสิ้น 16 กิโลไบต์)

- มีหน่วยความจำสำหรับโปรแกรมและข้อมูล (RAM) อยู่ภายในชิปจำนวน 128 ไบต์ (ในเบอร์ 8031, 8051) หรือ 256 ไบต์ (ในเบอร์ 8031, 8052)

- สามารถใช้หน่วยความจำสำหรับโปรแกรมและข้อมูลที่อยู่ภายนอกชิปได้ อย่างละ 64 กิโลไบต์ แยกจากกัน

- คำสั่งส่วนใหญ่ใช้เวลาเพียง 1 ไมโครวินาที เมื่อใช้คริสตอลความถี่ 12 เมกกะเฮิรตซ์

- มีพอร์ตที่สามารถรับหรือส่งข้อมูลได้ทั้ง 2 ทิศทาง จำนวน 4 พอร์ต พอร์ตละ 8 บิต หรือสามารถใช้งานเป็นพอร์ตขนาด 1บิต แยกจากกัน ทำให้เหมือนมีพอร์ตขนาด 1บิต ใช้งานรวมทั้งสิ้น 32 พอร์ต

- รับและส่งข้อมูลแบบอนุกรมได้ในตัวโดยสามารถกำหนดความเร็วในการรับและส่งข้อมูล (Baud rate) ได้ตั้งแต่ 300 ถึง 375 กิโลบิตต่อวินาที

- จัดลำดับความสำคัญของสัญญาณอินเตอร์รัปต์ได้ 2 ระดับ

- มีรีจิสเตอร์สำหรับใช้งานเป็นไทเมอร์ หรือ เคาน์เตอร์ เพื่อนับจำนวนสัญญาณนาฬิกาภายในชิป หรือนับเปลี่ยนสถานะของสัญญาณภายนอก 16 บิต จำนวน 2 ตัว เพื่อใช้สำหรับนับจำนวน pulse วัดความกว้างของ pulse หรือใช้วัดช่วงเวลา (ในเบอร์ 8052 จะมี 3ตัว)

- หน่วยความจำสำหรับเก็บข้อมูลภายในบางส่วน สามารถเข้าถึงข้อมูลได้ทั้งระดับบิต เพื่อให้การออกแบบโปรแกรมและการควบคุมระบบทำได้ง่ายขึ้น

- มีคำสั่งคูณและหารเลขในตัวเอง

- สามารถประมวลผลแบบบูลีน เพื่อใช้งานควบคุมโดยเฉพาะ

2.8.2 คุณสมบัติทั่วไปของ AT89C51

ไมโครคอนโทรลเลอร์ AT89C51 จะมีขาใช้งานพื้นฐานดังแสดงในรูปที่ 2.1 โดยมีรายละเอียดขั้นต้น ดังนี้

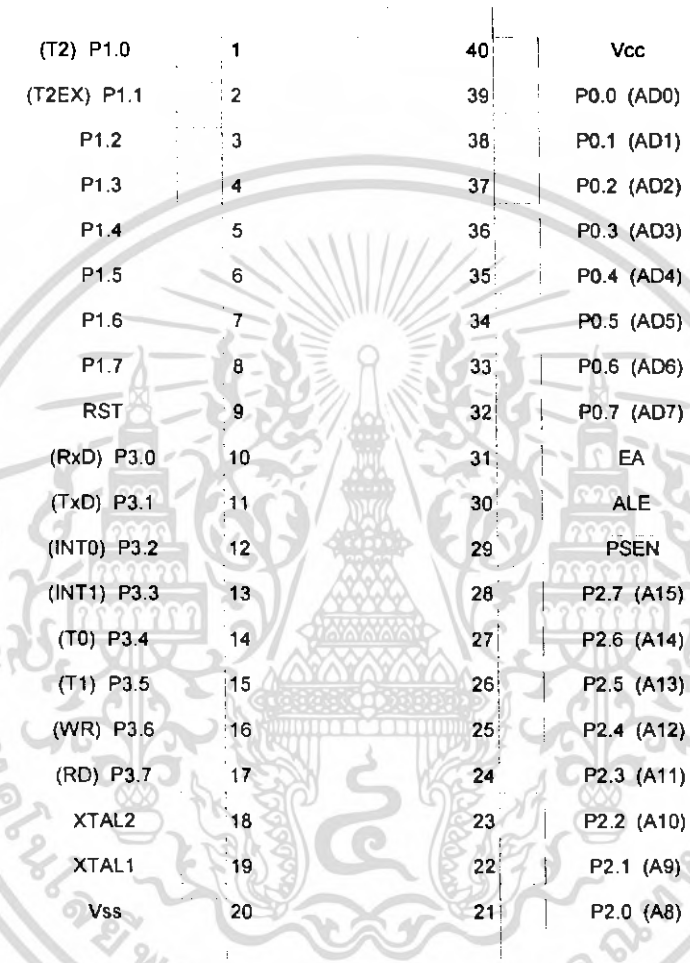
ขา Vcc (ขา 20) ใช้สำหรับต่อไฟเลี้ยง +5

ขา GND (ขา 40) เป็นขากราวด์ สำหรับต่อกับกราวด์ของระบบ

ขาพอร์ต 0 (ขา 32-39 หรือ P0.7-P0.0) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการให้ขาพอร์ต 0 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล 1 ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย เพื่อให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) ซึ่งมีสถานะ high impedance จึงจะสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ตนี้ยังถูกใช้

ในการติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้กระบวนการมัลติเพล็กซ์เข้าช่วย เพื่อสลับการทำงานให้เป็นได้ทั้งขาติดต่อแอดเดรสและขาข้อมูล

ขาพอร์ต 1 (ขา 1-8 หรือ P1.0-P1.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการให้ขาพอร์ตใดเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล 1 ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย เพื่อให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float)



รูปที่ 2.2 การจัดขามาตรฐานของไมโครคอนโทรลเลอร์ MCS-51

ขาพอร์ต 2 (ขา 21-28 หรือ P2.0-P2.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล 1 ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) ซึ่งมีสถานะ high impedance สามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก (A8-A15)

ขาพอร์ต 3 (ขา 10-17 หรือ P3.0-P3.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียน

ข้อมูล 1 ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) ซึ่งมีสถานะ high impedance สามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ต 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษ ดังมีรายละเอียดขั้นต้นต่อไปนี้

ขา P3.0 ใช้สำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

ขา P3.1 ใช้สำหรับส่งข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

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

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

ขา P3.4 ใช้รับสัญญาณไทเมอร์จากภายนอกช่อง 0 หรือขา T0

ขา P3.5 ใช้รับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขา T1

ขา P3.6 ใช้เป็นขาสัญญาณ $\overline{\text{WR}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

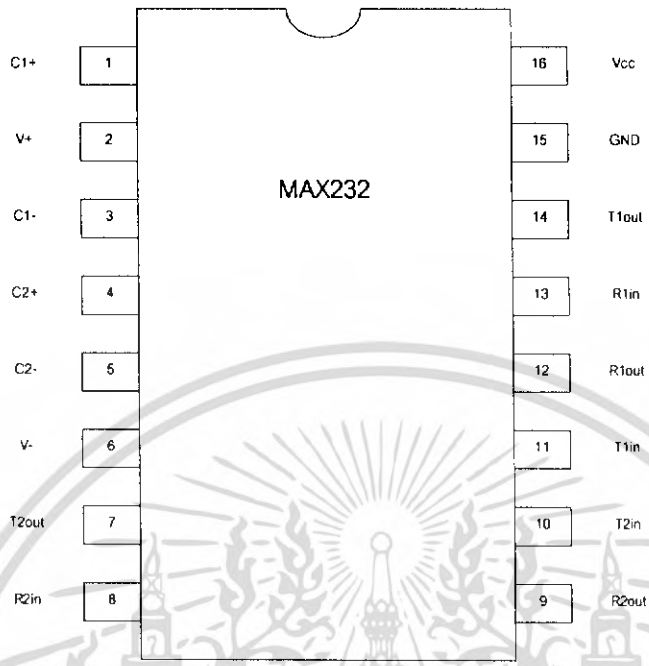
ขา P3.7 ใช้เป็นขาสัญญาณ $\overline{\text{RD}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

ซึ่งการใช้งานขาพอร์ต 3 ในหน้าที่พิเศษดังกล่าวนี้ จะต้องโหลดค่า 1 ไปยังแต่ละบิตที่ต้องการใช้ก่อนทุกครั้ง

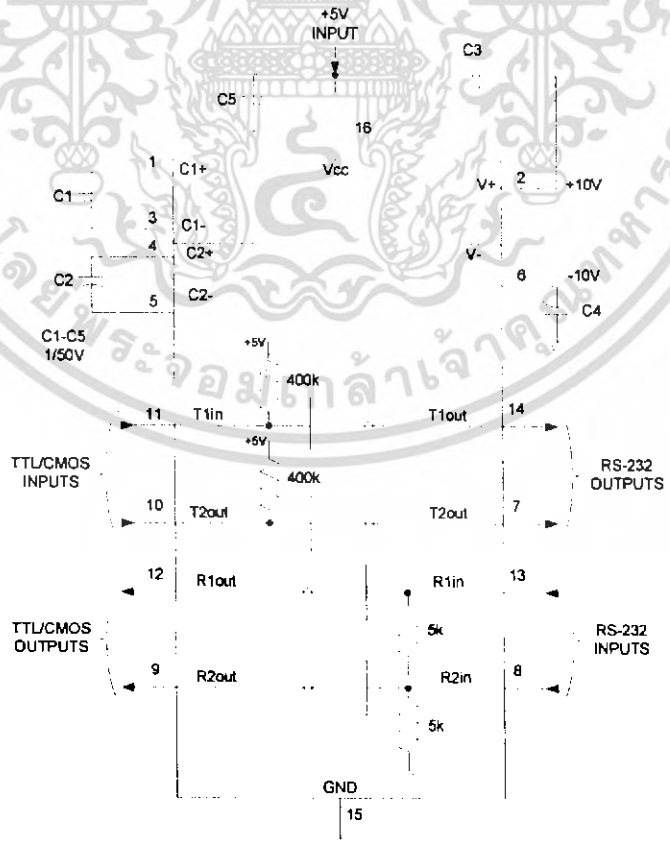
ขารีเซต(ขา 9 หรือ RST) ใช้ในการรีเซตการทำงานของไมโครคอนโทรลเลอร์เพื่อเริ่มต้นการทำงานใหม่ ซึ่งจะใช้เมื่อเริ่มจ่ายพลังงานหรือเมื่อโปรแกรมทำงานผิดพลาด โดยในการป้อนสัญญาณเพื่อรีเซต สถานะที่ขานี้ต้องอยู่ในระดับรีเซตอย่างน้อย 2 แมชชีนไซเคิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างปกติ

2.8.3 การ เชื่อมต่อกับพอร์ตอนุกรมของคอมพิวเตอร์

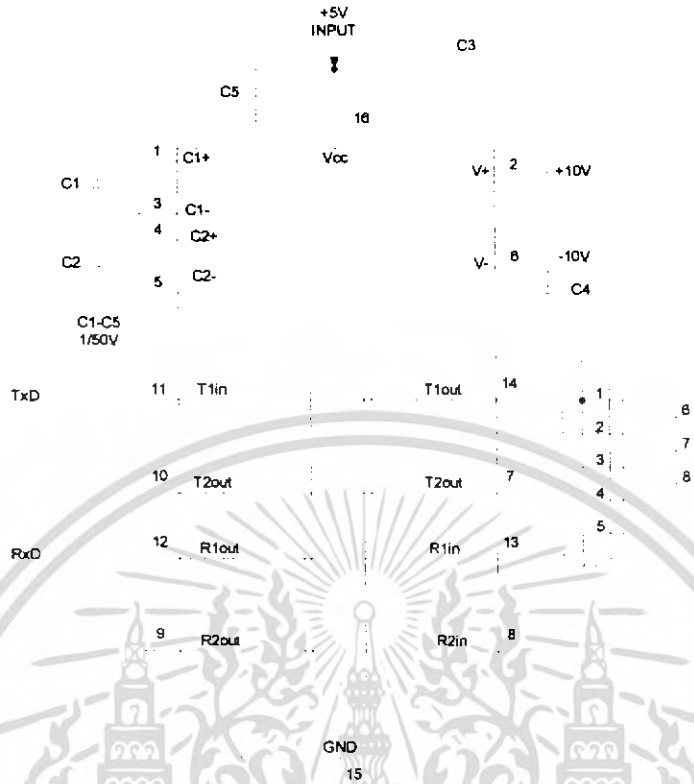
การใช้งานวงจรพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 มักนิยมใช้ในการติดต่อเพื่อแลกเปลี่ยนข้อมูลกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมในมาตรฐาน RS-232 เป็นส่วนใหญ่ แต่เนื่องจากระดับสัญญาณของพอร์ตอนุกรม RS-232 มีระดับตั้งแต่ ±3 โวลต์ถึง ±12 โวลต์ในขณะที่ระดับสัญญาณของไมโครคอนโทรลเลอร์ MCS-51 อยู่ในระดับทีทีแอล ดังนั้นจึงไม่สามารถเชื่อมต่อพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 เข้ากับพอร์ตอนุกรมของคอมพิวเตอร์ได้โดยตรง จึงต้องอาศัยการเชื่อมต่อผ่าน ไอซีที่ทำหน้าที่ในการแปลงระดับแรงดันของสัญญาณจากระดับทีทีแอลไปเป็นระดับแรงดันตามมาตรฐาน RS-232 ไอซีดังกล่าวมีด้วยกันหลายเบอร์จากหลายผู้ผลิต อาทิ MAX232 จาก MAXIM หรือ ICL232 จาก HARRIS เป็นต้น ในรูปที่ 2.3 แสดงการจัดขาของไอซี ICL232 ซึ่งใช้ในการแปลงสัญญาณ RS-232 และในรูปที่ 2.4 แสดงโครงสร้างภายในของไอซี ส่วนวงจรของการต่อกับไมโครคอนโทรลเลอร์ MCS-51 แสดงในรูปที่ 2.5



รูปที่ 2.3 การจัดขาของ MAX232 หรือ ICL232



รูปที่ 2.4 โครงสร้างภายในของ MAX232 หรือ ICL232



รูปที่ 2.5 แสดงวงจรเชื่อมต่อ MAX232 หรือ ICL232 เข้ากับพอร์ตอนุกรมของคอมพิวเตอร์และไมโครคอนโทรลเลอร์ MCS-51

2.9 ความรู้เบื้องต้นเกี่ยวกับพอร์ตอนุกรม

การเคลื่อนที่ย้ายข้อมูลจากเครื่องคอมพิวเตอร์ไปยังอุปกรณ์ต่อพ่วงภายนอกหรือคอมพิวเตอร์ด้วยกันมี 2 รูปแบบคือ รับส่งข้อมูลแบบขนานและรับส่งข้อมูลแบบอนุกรม

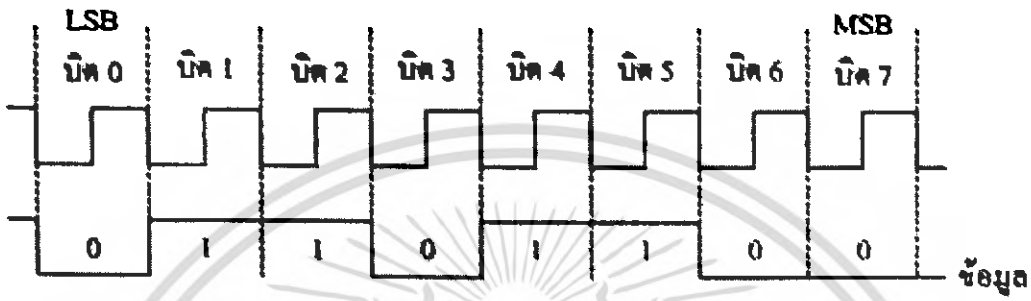
การรับส่งข้อมูลแบบขนานเป็นการรับและส่งข้อมูลคราวละ 4 ถึง 8 บิตในเวลาเดียวกัน ทำให้การรับและส่งข้อมูลมีความเร็วสูง แต่จำนวนสายที่ใช้ในการถ่ายทอดข้อมูลมีมากเท่ากับจำนวนบิตของข้อมูลที่ทำการถ่ายทอด นอกจากนั้นยังมีสายที่ใช้สำหรับควบคุมและตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตข้อมูลก็ได้

ในขณะที่การรับส่งข้อมูลแบบอนุกรมจะเป็นการรับส่งข้อมูลครั้งละ 1 บิต โดยมีรูปแบบการรับส่งข้อมูลที่เป็นมาตรฐาน ต้องมีการตรวจสอบความพร้อมในการรับและส่งข้อมูลของตัวส่งและตัวรับ การรับส่งข้อมูลแบบอนุกรมมีข้อดีในเรื่องจำนวนสายสัญญาณที่น้อยมาก และไม่แปรผันตามจำนวนบิตของข้อมูล ระยะทางในการรับส่งข้อมูลสูงกว่าแบบขนานมาก

การสื่อสารแบบอนุกรมแบ่งออกเป็น 2 แบบคือ การสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส

2.9.1 การสื่อสารข้อมูลแบบซิงโครนัส

การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมกันอยู่กับการรับและการส่งสัญญาณ ด้วยตัวอย่างการส่งแบบซิงโครนัส ก็คือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของนาฬิกา ส่วนสายอีกเส้นจะเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูลและกราวด์

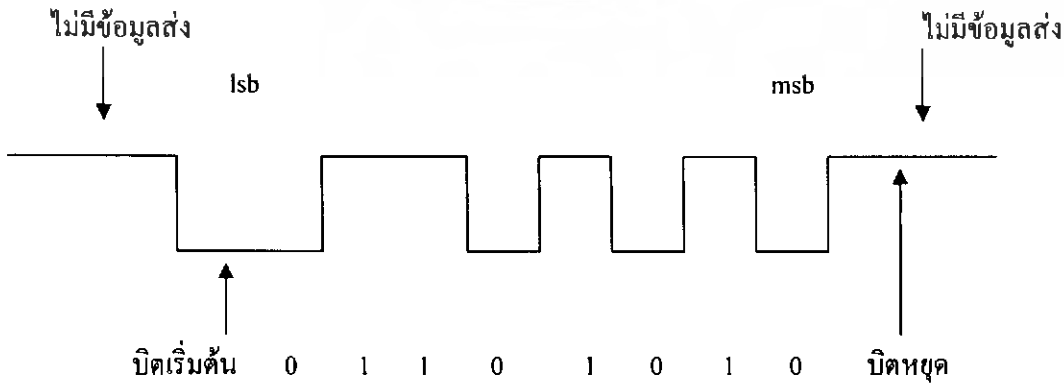


รูปที่ 2.6 แผนผังการทำงานเวลาของการสื่อสารข้อมูลแบบซิงโครนัส

2.9.2 การสื่อสารข้อมูลแบบอะซิงโครนัส

เป็นการแก้ปัญหาการซิงโครไนเซชั่นวิธีหนึ่ง การส่งข้อมูลแบบอะซิงโครนัสเกี่ยวข้องกับการส่งอักขระ (Character) แต่ละตัวที่เวลาใดก็ได้ รูปแบบของการส่งข้อมูลแบบอะซิงโครนัสเริ่มต้นด้วยบิตเริ่มต้น (start bit) เป็นการบอกจุดเริ่มต้นของการส่งข้อมูล คือ เป็นการเปลี่ยนสถานะของตัวกลางในการส่งข้อมูลจาก idle state (สถานะที่ไม่มีข้อมูลส่ง) ซึ่งระดับแรงดันในสายส่งข้อมูลเป็นบิต 1 หรือ Marking มาเป็นสถานะที่มีการส่งข้อมูล

การส่งข้อมูลที่เป็นข้อความ (Text Data) ระหว่างอุปกรณ์ปลายทางกับคอมพิวเตอร์มักเป็นการส่งข้อมูลแบบอะซิงโครนัส ภายหลังจากส่งบิตเริ่มต้นออกไปแล้ว จะเป็นบิตข้อมูลจำนวน 7 บิต หรือ 8 บิต แล้วแต่ว่าจะเป็นรหัสอักขระชนิดใด ตามด้วยพาริตี้บิต (parity bit) จำนวน 1 บิต สำหรับการตรวจวัดความผิดพลาดของข้อมูลและปิดท้ายด้วยบิตหยุด (stop bit) จำนวน 1 บิต หรือมากกว่า



รูปที่ 2.7 ตัวอย่างการส่งข้อมูลแบบอะซิงโครนัส

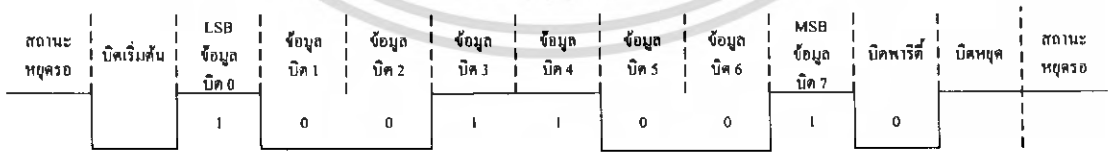
การสื่อสารข้อมูลแบบอะซิงโครนัส คือ การรับและส่งข้อมูลโดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วย แต่จะใช้การกำหนดอัตราความเร็วในการรับและส่งข้อมูลให้มีค่าเท่ากัน ซึ่งเรียกอัตราเร็วนี้ว่า อัตราบอดเรต (Baud Rate) มีหน่วยเป็น บิตต่อวินาที

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัส ประกอบด้วย 4 ส่วนด้วยกัน คือ

1. บิตเริ่มต้น (Start bit) มีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรม มีขนาด 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity bit) มีขนาด 1 บิต หรือไม่มี
4. บิตปิดท้ายหรือบิตหยุด (Stop bit) มีขนาด 1 บิต

รูปที่ 2.8 แสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส เมื่อไม่มีการส่งข้อมูลขา DATA จะมีสถานะลอจิก "1" เรียกสถานะนี้ว่า สถานะหยุดรอ (Waiting Stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากให้ขา DATA มีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต เรียกบิตนี้ว่าบิตเริ่มต้น (Start Bit) จากนั้นบิตข้อมูลจะถูกส่งออกไปโดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุดหรือบิต LSB ก่อน ซึ่งข้อมูลที่ต้องการส่งอาจมีจำนวน 5, 6, 7 หรือ 8 บิตก็ได้ จากนั้นตามด้วยพาริตี้บิต (Parity Bit) ซึ่งใช้ในการตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งก็คือ บิตปิดท้ายหรือบิตหยุด (Stop Bit) โดยจะเป็นการทำให้ขา DATA มีสถานะลอจิก "1" อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสหรือ อัตราบอด หรือ บอดเรตที่ใช้สำหรับพอร์ตอนุกรม RS-232 มีด้วยกันหลายค่า ได้แก่ 110, 150, 300, 600, 1200, 2,400, 4,800, 9,600 และ 19,200 บิตต่อวินาที โดยมีค่ามากขึ้นตามเทคโนโลยีของคอมพิวเตอร์เนื่องจากบอดเรตคือค่าของจำนวนบิตที่สามารถส่งได้ใน 1 วินาที สมมติว่า ข้อมูลอนุกรมมีขนาด 8 บิตไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูล 1 ไบต์ จะมีความยาวเท่ากับ 10 บิต ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9,600 ต่อวินาที ก็จะสามารถส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที



รูปที่ 2.8 รูปแบบของข้อมูลแบบอะซิงโครนัส

การตรวจสอบพาริตี้สามารถกำหนดเป็นแบบคี่ (Odd), แบบคู่ (Even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ พาริตี้คี่ หรือพาริตี้คู่แสดงถึงจำนวนลอจิก "1" ทั้งหมดภายในข้อมูลที่ส่งไป 1 ไบต์รวมพาริตี้ว่ามีจำนวนเป็นเลขคู่หรือเลขคี่ ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิต มีค่าเท่ากับ 99H หรือ 10011

001B จะเห็นว่าข้อมูลในไบต์มีจำนวนลอจิก "1" จำนวน 4 ตัว ซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตี้เป็นคู่ ค่าของพาริตี้บิตจะต้องมีลอจิกเป็น "0" แต่ถ้ากำหนดพาริตี้เป็นคี่ ค่าของพาริตี้บิตจะต้องมีลอจิกเป็น "1" เพื่อให้ข้อมูลไบต์รวมทั้งบิตพาริตี้เป็นคี่

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART (Universal Asynchronous Receiver Transmitter) ซึ่งทางภาครับจะต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ที่ตรงกันเอาไว้ว่า จะตรวจสอบพาริตี้คี่ หรือพาริตี้คู่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่ แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแสดงข้อผิดพลาดออกมาให้ผู้ใช้รับทราบ กระบวนการดังกล่าวเป็นวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้นในการรับส่งข้อมูลที่ง่ายที่สุด แต่มันสามารถตรวจสอบได้เมื่อมีบิตข้อมูลที่ทำการรับส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

2.9.3 การเชื่อมต่อไมโครโปรเซสเซอร์เพื่อรับส่งข้อมูลอนุกรม (UART)

มีอยู่ 2 โหมดด้วยกันคือ

- Single Processor Mode
- Multiprocessor Mode

Single Processor Mode : ในโหมดนี้เราจะใช้ไมโครคอนโทรลเลอร์ 2 ตัวเชื่อมเข้าหากัน

Multiprocessor Mode : ในโหมดนี้เราจะใช้ไมโครคอนโทรลเลอร์ 1 ตัวเป็นตัวแม่ (Master) และอีก 0-256 เป็นตัวลูก(Slave)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

รูปที่ 2.9 รีจิสเตอร์ที่ใช้ควบคุมการรับส่งข้อมูลอนุกรม SCON (อยู่ใน SFR ตำแหน่ง 98H)

SM2 บิตเลือกการทำงานแบบ Single Processor Environment หรือ Multiprocessor Environment

1 : เลือก Multiprocessor Environment ใช้ได้กับโหมด 2,3

0 : เลือก Single Processor Environment ใช้ได้กับทุกโหมด

เมื่อเลือกการทำงานรับข้อมูลแบบ Multiprocessors Mode แล้ว

ถ้าข้อมูลบิตที่ 9 ที่รับได้มีค่าเป็น 1 RI จะเซ็ท

ถ้าข้อมูลบิตที่ 9 ที่รับได้มีค่าเป็น 0 RI จะไม่เซ็ท

REN (Receive Enable) บิตควบคุมให้รับหรือไม่รับข้อมูล

1 : ให้รับข้อมูลได้

0 : ห้ามรับข้อมูล

TB8 (Transmit bit D8) ข้อมูลบิตที่ 9 ที่จะส่งออกไปในโหมด 2,3 ให้ใส่ในบิตนี้ได้เลย

RB8 (Receive bit D8) ข้อมูลบิตที่ 9 ที่รับเข้ามาจะมาเก็บในบิตนี้ (ข้อมูลบิตที่ 9 ก็คือค่าใน TB8 ทางด้านส่ง นั่นเอง)

TI แฟลกซ์ TI จะเป็น 1 เมื่อสิ้นสุดการส่งข้อมูล 1 ไบต์

RI แฟลกซ์ RI จะเป็น 1 เมื่อรับข้อมูลเสร็จ 1 ไบต์ (บิต RI , TI ผู้เขียนโปรแกรมจะต้องเคลียร์เอง)

2.9.4 มาตรฐานพอร์ตอนุกรมแบบ RS-232

เป็นมาตรฐานที่ใช่กันกว้างขวางที่สุดถูกประกาศในปี1969 โดย Electronic Industries Association (EIA) เพื่อกำหนดการเชื่อมต่ออุปกรณ์รับส่งข้อมูลปลายทาง (Data Terminal Equipment: DCE) โดยที่ RS ย่อมาจาก Recommended Standard และ 232 เป็นหมายเลขบ่งบอกของมาตรฐานตัวนี้ C เป็นหมายเลขของฉบับสุดท้ายของมาตรฐานตัวนี้

มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้ส่งผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดซึ่งอยู่ห่างไกล โดยสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association: EIA) ได้วางมาตรฐานที่มีชื่อเรียกว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3V จนถึง -12V แสดงว่ามีข้อมูล (mark) และ +3V จนถึง +12V แสดงว่าเป็นช่องว่าง (Space)

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

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัดคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ในโมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานคอมพิวเตอร์ พอร์ตอนุกรม RS-232 ถูกใช้เพื่อเชื่อมต่อกับโมเด็ม, เมาส์ และเครื่องพิมพ์ที่สามารถติดต่อทางพอร์ตอนุกรมได้

2.9.5 สัญญาณที่ใช้ทั้งหมดใน RS-232-C

-Protective Ground (PG –ขาที่ 1)

หมายถึงตัวถังของเครื่องหรือสายดิน

-Transmit Data (TD ขาที่ 2)

เป็นสัญญาณที่ส่งออกมาจาก DTE (ตัวไมโครคอมพิวเตอร์) ไปยังยังโมเด็มหรือต่อเข้าโดยตรงกับไมโครคอมพิวเตอร์ตัวอื่นหรือเครื่องพิมพ์ เมื่อไม่มีสัญญาณส่งออกสถานภาพของลอจิกจะมีค่าเท่ากับ

"1"สภาวะ "OFF" หรือเทียบเท่ากับ stop bit ไม่ว่าจะเป็นระบบอะไร DTE ต้องไม่ส่งข้อมูลออกไปจนกว่าสัญญาณ

1. Request To Send (RTS)

2. Clear To Send (CTS)

3. Data Set Ready (DSR)

4. Data Terminal Ready (DTR)

ทั้งหมดนี้อยู่ในสภาวะ "ON"

-Receive Data (RD ขาที่ 3)

เป็นทางของสัญญาณเข้าไปยัง DTE เมื่อไม่มีสัญญาณรับเข้ามา ขานี้จะมีสถานภาพเป็น "1" หรือสภาวะ "OFF"

-Request To Send (RST ที่ขา4)

จาก DTE ไปยัง DCE

สภาวะ ON คือ บังคับให้ DCE อยู่ใน Transmitting Mode ต่อไป

สภาวะ OFF คือ บังคับให้ DCE อยู่ใน Receiving Mode ต่อไป

การเปลี่ยนจาก OFF ไป ON เป็นการบอกให้ DCE จัดการกับระบบสื่อสาร เพื่อให้ช่องทางต่อเชื่อมและให้สัญญาณ Clear To Send (CTS) กลับมาเป็นการบอกว่าส่งได้

การเปลี่ยนจาก ON ไป OFF เป็นการบอกให้ DCE ส่งข้อมูลผ่านช่องสื่อสารให้หมดแล้วกลับไปอยู่ใน Receiving mode พร้อมกับให้ CTS เป็น 0

- Clear To Send (CTS ขาที่ 5)

จาก DCE ไป DTE สภาวะ NO หมายความว่าข้อมูลจาก DTE ขา 2 จะถูกส่งต่อไปในช่องทางสื่อสาร(โมเด็มส่งข้อมูลออกสายโทรศัพท์) ทันที CTS จะ ON หลังจาก DSR และ RTS อยู่ในสภาวะON และการเชื่อมของวงจรสื่อสาร(ชุมสายโทรศัพท์) เสร็จแล้ว

-Data Set Ready (DTS ขาที่ 6)

จาก DCEไป DTE คือความพร้อมของโมเด็มนั้นเอง จะเป็น NO (พร้อม) ต่อเมื่อ

1. DCE (โมเด็ม) เปิดเครื่องอยู่ และอยู่ในสภาวะoff – hook (เหมือนยกหูโทรศัพท์)

2. DCE ไม่อยู่ใน Test Mode

3. DCE ทำการส่งสัญญาณไปยังชุมสายเสร็จแล้ว

DSR อยู่ในสภาวะ ON เป็นการบอก DTE ว่า โมเด็มต่อเข้ากับสายโทรศัพท์เรียบร้อยแล้ว และพร้อมที่จะส่ง

DSR อยู่ในสภาวะ OFF หมายถึงให้ DTE ตรวจ Ring Indicator

-Signal Group (SG ขาที่ 7)

คือสายร่วมของสัญญาณทุกตัว

-Carrier Detect (CD ขาที่ 8)

จาก DCE ไป DTE

สภาวะON หมายถึง DCE จับสัญญาณพาหะในช่องทางสื่อสารที่จะทำการดีมอดูเลตได้

สภาวะOFF คือไม่ได้รับสัญญาณอะไรเลย หรือได้รับสัญญาณแต่ไม่สามารถดีมอดูเลตเอาข้อมูลออกมาได้

-Data Terminal Ready (DTR ขาที่ 20)

จาก DTE ไป DCE

สภาวะON หมายถึงว่า DCE เตรียมเพื่อเชื่อมต่อกับตัวอื่น และรักษาช่องทางติดต่อไว้ต่อไป (การเชื่อม Channel ทำได้หลายทาง คือ หมุนเรียกด้วยมือหรืออัตโนมัติ) ถ้า DCE สามารถตอบรับสัญญาณเรียก (Call)ได้ ก็ให้ตอบรับ(Answering) เมื่อมีสัญญาณเรียก Ring Indicator และ DTR ON อยู่

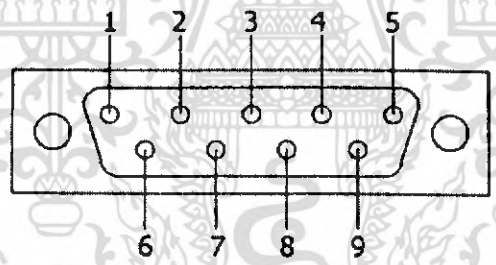
สภาวะOFF คือวางหู และเมื่อ OFF แล้วไม่ต้อง ON อีกจนกว่า DSR จะOFF

-Ring Indicator (RI ขาที่ 22)

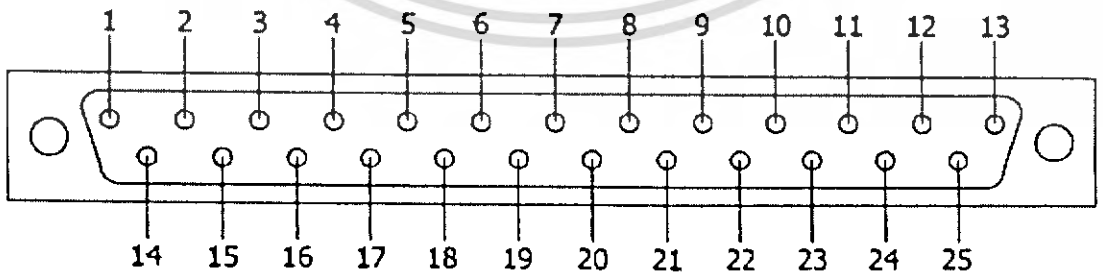
จาก DCE ไป DTE เหมือนสัญญาณเรียกของโทรศัพท์ แต่เป็นดิจิตอล ใช้ในระบบตอบได้อัตโนมัติ (Auto-answer) สัญญาณนี้จะ ON เมื่อมีสัญญาณกระดิ่งเข้ามา และจะ OFF ระหว่างเสียงดังของกระดิ่ง

2.9.6 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้ หรือ DB-9 ตัวผู้ ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาใช้งานเพียง 9 เส้น เช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่นๆ ที่เคยมีใช้งานม ลงรูปร่างตำแหน่งขาในรูปที่ 2.9



(ก) คอนเน็กเตอร์อนุกรม 9 ขา หรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)



(ข) คอนเน็กเตอร์อนุกรม 25 ขา หรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

รูปที่ 2.10 คอนเน็กเตอร์อนุกรม

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

ตารางที่ 3 การจัดขาสัญญาณของพอร์ตอนุกรมในแบบต่างๆ และหน้าที่การทำงาน

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

ขา Received Data: RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์โดยจะนำข้อมูลที่อ่านได้ไปเก็บไว้ในรีจิสเตอร์บัฟเฟอร์

ขา Transmitted Data: TD หรือ TxD ขานี้ใช้เพื่อส่งข้อมูลอนุกรมออกจากคอมพิวเตอร์โดยการนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

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

ขา Single Ground: GND เป็นขากราวด์ของสัญญาณ

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

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

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

ขา Ring Indicator: RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มแล้วยังมีความต้องการตรวจสอบสัญญาณเรียกสายโทรศัพท์

2.9.7 รีจิสเตอร์ที่เกี่ยวข้องกับการทำงานของพอร์ตอนุกรมใน MCS – 51

ในการทำงานของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS – 51 มีรีจิสเตอร์ที่เกี่ยวข้องอยู่ 2 ตัว ดังนี้

2.9.7.1 รีจิสเตอร์บัฟเฟอร์ของพอร์ตอนุกรมหรือ SBUF (Serial Data Buffer Register)

มีแอดเดรสอยู่ที่ 99H ในพื้นที่ของรีจิสเตอร์ฟังก์ชันพิเศษหรือ SFR มีขนาด 8 บิต แบ่งเป็น 2 ส่วนคือ รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลและรับข้อมูล เมื่อมีการเขียนข้อมูลมายังรีจิสเตอร์ SBUF ข้อมูลนั้นจะถูกส่งต่อไปยังบัฟเฟอร์สำหรับส่งข้อมูล เพื่อส่งออกจากไมโครคอนโทรลเลอร์ผ่านทางขา TxD หรือขา P3.1 ในกรณีที่มีการอ่านข้อมูลจากรีจิสเตอร์ SBUF ข้อมูลจะถูกส่งผ่านไปยังรีจิสเตอร์บัฟเฟอร์สำหรับข้อมูล เพื่อส่งต่อไปยังไมโครคอนโทรลเลอร์ต่อไป สำหรับการรับข้อมูลอนุกรมจากภายนอกนั้นจะผ่านมาทางขา RxD หรือ P3.0 ของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลช

2.9.7.2 รีจิสเตอร์ควบคุมการทำงานของพอร์ตอนุกรมหรือ SCON (Serial Port Control Register)

เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 98H ในพื้นที่ของรีจิสเตอร์ SFR สามารถเข้าถึงได้ในระดับบิต มีรายละเอียดการทำงานดังนี้

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

รูปที่ 2.11 แสดงรีจิสเตอร์ควบคุมการทำงานในแต่ละบิตของพอร์ทอนุกรม

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

SM2 : ใช้ในการเอ็นเอเบิลการสื่อสารในแบบมัลติโปรเซสเซอร์ ในการทำงานของโหมด 2 และ 3 ของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS – 51 ถ้าบิตนี้เป็น "1" บิต RI จะไม่แอคตีฟ ถ้าบิตที่ 9 ที่รับเข้ามาเป็น "0" ในการทำงานโหมด 1 ถ้าบิตนี้เซต บิต RI จะไม่แอคตีฟถ้ายังไม่ได้รับบิตหยุดส่วนในโหมด 0 บิตนี้ไม่มีการใช้งาน

REN (Enable serial reception) : ใช้ในการเอ็นเอเบิลการรับข้อมูลของพอร์ตอนุกรม ทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

TB8 : ใช้สำหรับเก็บข้อมูลบิตที่ 9 ที่ต้องการส่งออกไปในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมใน MCS – 51 เซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

RB8 : ใช้สำหรับรับข้อมูลบิตที่ 9 ที่เข้ามาในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS – 51 แต่ถ้าหากพอร์ตอนุกรมทำงานในโหมด 1 และ บิต SM2 เป็น "0" ข้อมูลที่บิตRB8 คือ ข้อมูลของบิตหยุด สำหรับในการทำงานในโหมด 0 บิตนี้ไม่ใช้งาน บิต RB8 นี้สามารถเซตด้วยกระบวนการทางซอฟต์แวร์

TI (Transmit Interrupt flag) : ใช้ในการแสดงการเกิดอินเตอร์รัปต์เมื่อมีการส่งข้อมูลออกจากพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS – 51 สามารถเซตได้ด้วยกระบวนการฮาร์ดแวร์ เมื่อมีการส่งข้อมูลบิตที่ 8 ไปเรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่น บิตนี้จะเซตเมื่อมีการเริ่มต้นส่งบิตหยุดออกไป การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

RI (Receive Interrupt flag) : ใช้ในการแสดงการเกิดอินเตอร์รัปต์เมื่อมีการรับข้อมูลเข้าสู่พอร์ตอนุกรม สามารถเซตได้ด้วยกระบวนการฮาร์ดแวร์ เมื่อมีการรับข้อมูลบิตที่ 8 ไปเรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่น บิตนี้จะเซตเมื่อรับบิตหยุดของข้อมูลอนุกรมไปได้ครึ่งทางแล้ว ยกเว้นกรณีที่บิต SM2 มีการเซต บิตนี้จะเซต ได้ก็ต่อเมื่อการรับบิตหยุดหรือบิตที่ 9 เกิดขึ้นอย่างสมบูรณ์แล้ว การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

2.9.8 โหมดการทำงานของพอร์ตอนุกรมใน MCS – 51

2.9.8.1 การใช้งานพอร์ทสื่อสารอนุกรมแบบ Single Processor

พอร์ทสื่อสารอนุกรมมีโครงสร้างการทำงานในแบบที่เรียกว่าฟูลดูเพล็กซ์ (Full Duplex) สามารถรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน

- ทางด้านส่งใช้ขา TxD (พอร์ท 3.1)
- ทางด้านรับใช้ขา RxD (พอร์ท 3.0)

Serial Port Buffer (SBUF) ใช้เป็นบัฟเฟอร์สำหรับรับส่งและส่งข้อมูลอนุกรมโดยมีอยู่ 2 ตัว

- การส่งข้อมูล ข้อมูลที่จะส่งให้ใส่ใน SBUF โดยใช้คำสั่ง MOV SBUF,A โดยเตรียมข้อมูลที่จะส่งเข้า A ก่อน
- การรับข้อมูล ข้อมูลที่ได้รับจะอยู่ใน SBUF การถ่ายข้อมูลออกมาใช้คำสั่ง MOV A,SBUF แล้วจึงนำข้อมูลใน A ไปใช้

พอร์ทสื่อสารอนุกรมสามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM1 และ SM0 ซึ่งอยู่ในรีจิสเตอร์ควบคุม พอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS – 51 สามารถเลือกการทำงานได้ถึง 4 โหมด มีดังนี้

SM0	SM1	โหมด	การทำงาน
0	0	0	Shift register ความเร็วในการรับหรือส่งข้อมูลเท่ากับ (1/12) ของ CPU Osc
0	1	1	8 Bit UART ความเร็วในการรับหรือส่งข้อมูลกำหนดได้จาก Timer 1,2
1	0	2	9 Bit UART ความเร็วในการรับหรือส่งข้อมูล = (1/32) หรือ (1/64) เท่าของ CPU OSC โดยขึ้นกับบิต SMOD ใน PCON
1	1	3	9 Bit UART ความเร็วในการรับหรือส่งข้อมูลกำหนดที่ Timer 1,2

ตารางที่ 2.4 แสดง SM0,SM1 บิตเลือกโหมดการทำงาน

1. โหมด 0 เป็นการกำหนดให้พอร์ตอนุกรมทำงานในลักษณะชิฟต์รีจิสเตอร์ การทำงานในโหมดนี้ของไมโครคอนโทรลเลอร์ MCS – 51 จะใช้ในการเชื่อมต่อกับไอซีรีจิสเตอร์ภายนอกเพื่อทำการขยายพอร์ตอินพุตหรือเอาต์พุต แต่ไม่นิยมใช้งานมากนัก เนื่องจากในไมโครคอนโทรลเลอร์ MCS – 51 เองมีพอร์ตอยู่ค่อนข้างมาก และติดต่อกับพอร์ตเหล่านั้นได้ง่ายและเร็วกว่ามาก

2. กำหนดจากอัตราการเกิดโอเวอร์โฟลว (Overflow) ของไทเมอร์ 1 ใน AT89C51 ส่วนในไมโครคอนโทรลเลอร์ เบอร์ AT98C52 และในอนุกรม AT89Sxx สามารถเลือกใช้อัตราการเกิดโอเวอร์โฟลวของไทเมอร์ 1 หรือไทเมอร์ 2 ในการกำหนดอัตราบอดได้ การทำงานในโหมดนี้ได้รับความนิยมสูงสุด เนื่องจากมีกระบวนการที่ไม่ซับซ้อนและสามารถทำการรับส่งข้อมูลกับคอมพิวเตอร์ได้อย่างมีประสิทธิภาพ

3. โหมด 2 เป็นการกำหนดให้เป็น UART ขนาด 9 บิต โดยมีอัตราบอดคงที่

4. โหมด 3 เป็นการกำหนดให้เป็น UART ขนาด 9 บิต สามารถเลือกอัตราบอดได้

การเลือกโหมดทำได้ด้วยการกำหนดข้อมูลให้แก่บิต SM0 และ SM1 ในรีจิสเตอร์ SCON

2.9.8.2 อัตราบอดของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS – 51

โหมด 0

อัตราบอดของโหมดนี้มีค่าคงที่ โดยสามารถคำนวณได้จากสูตร

อัตราบอดในโหมด = ความถี่ของสัญญาณนาฬิกา/2 หน่วยเป็น บิตต่อวินาที

โหมด 1 และ 3

เนื่องจากทั้งสองโหมดนี้สามารถเลือกแหล่งกำเนิดอัตราบอดได้ 2 แหล่ง คือ จากอัตราการเกิดโอเวอร์โฟลวของไทเมอร์ 1 และ 2 สำหรับอัตราบอดเมื่อใช้การโอเวอร์โฟลวของไทเมอร์ 1 จะต้องใช้ค่าบิต SMOD ในรีจิสเตอร์ PCON มาพิจารณาประกอบด้วย สามารถคำนวณค่าอัตราบอดได้จาก

อัตราบอด = $(2^{\text{ค่าของบิตSMOD}}/32)$ * อัตราโอเวอร์โฟลวของไทเมอร์ 1

ถ้าหากในไทเมอร์ 1 ไม่ได้เอ็นเอเบิลการอินเตอร์รัปต์ไว้ สามารถคำนวณค่าอัตราบอดได้จาก

อัตราบอด = $(2^{\text{ค่าของบิตSMOD}}/32)$ * (ความถี่สัญญาณนาฬิกา/{12x[256 – (TH1)}) การกำหนดอัตราบอดโดยใช้ไทเมอร์ 1 แสดงในตารางที่5

โหมด 2

ในโหมดนี้อัตราบอดจะขึ้นอยู่กับค่าของบิต SMOD ในรีจิสเตอร์ PCON ถ้า SMOD เป็น "0" อัตราบอดจะเท่ากับ 1/64 ของสัญญาณนาฬิกา ในกรณีที่ SMOD เป็น "1" อัตราบอดจะเท่ากับ 1/32 ของความถี่สัญญาณนาฬิกา สามารถแสดงเป็นสูตรคำนวณทางคณิตศาสตร์ได้ดังนี้

อัตราบอด = $(2^{\text{ค่าของบิตSMOD}}/64)$ * ความถี่สัญญาณนาฬิกา

อัตราบอด (บิตต่อวินาที : bps)	ความถี่ สัญญาณนาฬิกา	SMOD	ไทเมอร์ 1		
			C/T	โหมด	ค่ารีโหลด
โหมด 0 : สูงสุด 1 MHz	12 MHz	×	×	×	×
โหมด 2 : สูงสุด 375 kHz	12 MHz	1	×	×	×
โหมด 1, 3 : 62.5 kHz	11.0592 MHz	1	0	2	FFH
19.2 k (19,200)	11.0592 MHz	1	0	2	FDH
9.6 k (9,600)	11.0592 MHz	0	0	2	FDH
4.8 k (4,800)	11.0592 MHz	0	0	2	FAH
2.4 k (2,400)	11.0592 MHz	0	0	2	F4H
1.2 k (1,200)	11.0592 MHz	0	0	2	E8H
137.5	11.0592 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

ตารางที่ 2.5 การเลือกอัตราบอดของวงจรพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS – 51

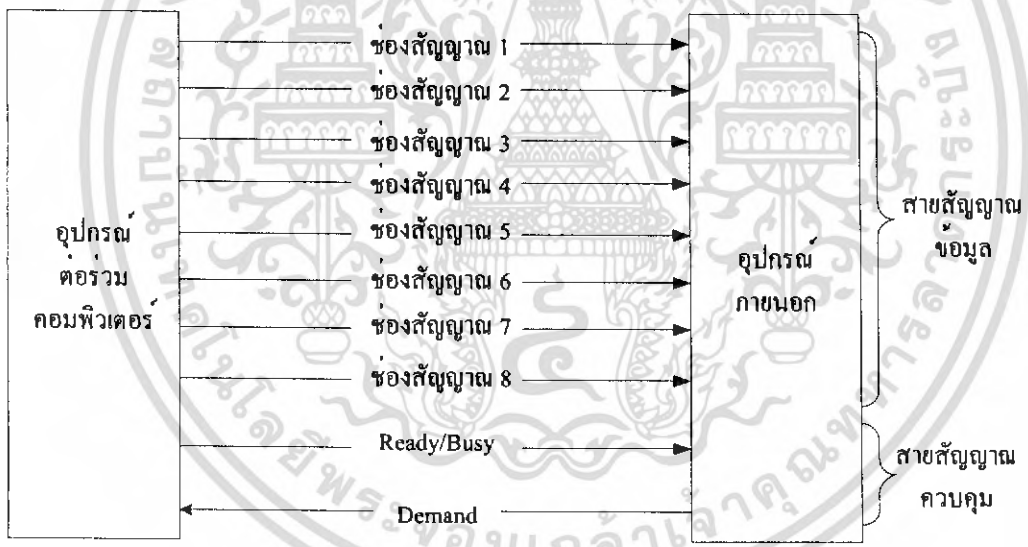
2.10 การส่งข้อมูลแบบขนาน (Parallel Transmission)

รูปที่ 2.9 แสดงการส่งข้อมูลแบบขนาน แต่ละบิตในบล็อกของบิตข้อมูลที่จะส่งมีสายสัญญาณสำหรับส่งเป็นของตนเอง ดังนั้นบิตทุกบิตจะถูกส่งออกไปพร้อมๆ กัน ในการส่งข้อมูลแบบขนานจะต้องมีวงจรควบคุม(Control Circuit) แยกออกมาจากวงจรส่งข้อมูล (Data Circuit) สำหรับการซิงโครไนซ์ (Synchronization) สัญญาณที่ส่งมา

ผู้ส่งจะใช้วงจรควบคุมวงจรหนึ่งสำหรับแจ้งให้ผู้รับทราบว่า พร้อมที่ส่งข้อมูลแล้ว (Ready/Busy line) ผู้รับจะใช้วงจรควบคุมอีกวงจรหนึ่งสำหรับแจ้งให้ผู้ส่งทราบว่าได้รับข้อมูลแล้ว และพร้อมที่จะรับข้อมูลชุดใหม่ (Demand line)

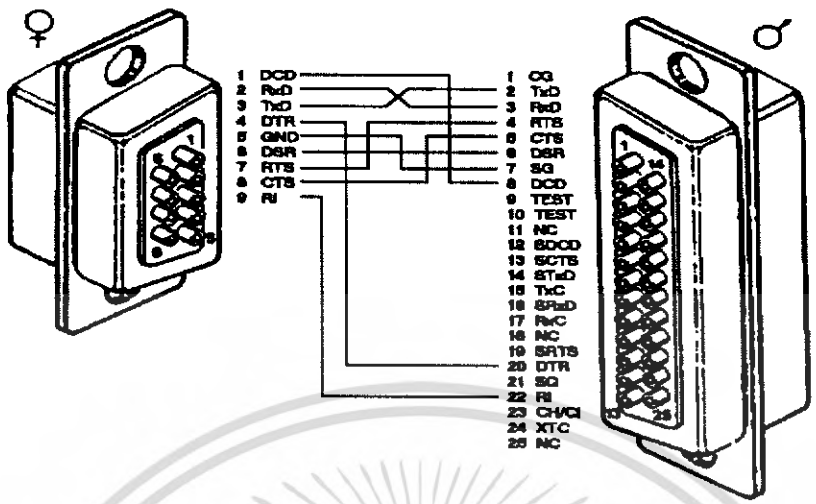
เรามักใช้การส่งข้อมูลแบบขนานสำหรับการส่งข้อมูลระยะสั้น เช่นการต่อร่วมระหว่างคอมพิวเตอร์กับอุปกรณ์ต่อพ่วง (Peripheral) หรือเครื่องมือวัดทางวิทยาศาสตร์ เนื่องจากมีการลดทอนของสัญญาณเนื่องจากความต้านทานของสาย และปัญหาที่เกิดขึ้นหากระยะทางของสายยาว คือ ระดับของกราวนด์ในทางไฟฟ้าที่จุดรับผิดไปจากจุดส่งทำให้เกิดการผิดพลาดในการรับสัญญาณทางฝ่ายรับ

และอีกกรณีหนึ่งคือเมื่อระยะทางในการส่งข้อมูลไกลขึ้น ต้นทุนของสายส่งสัญญาณหลายๆเส้นมีค่าเพิ่มมากขึ้นตามไปด้วย



รูปที่ 2.12 การส่งข้อมูลแบบขนานครั้งละ 8 บิต

2.11 คอนเน็กเตอร์แบบ 9 พิน (DB-9) และแบบ 25 พิน (DB-25)



รูปที่ 2.13 แสดงคอนเน็กเตอร์แบบ DB-9 และ แบบ DB-25

หมายเลขขาสัญญาณ	ชื่อของสายสัญญาณ
1	Data Carrier Detect
2	Receive Data
3	Transmitted Data
4	Data Terminal Ready
5	Signal Common
6	Data Set Ready
7	Request to Send
8	Clear to Send
9	Ring Indicator

ตารางที่ 2.5 รายละเอียดขาต่างๆของคอนเน็กเตอร์ DB-9

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 Ground
8	Received Line Signal Detector
9	Received for Data Set Testing
10	Received for Data Set Testing
11	Unsigned
12	Secondary Received Line Signal Detector
13	Secondary Transmitted Data
14	Secondary Clear to Send
15	Transmitted Signal Element Timing (DTE source)
16	Secondary Received Data
17	Received 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 Select (DTE/DCE Source)
24	Transmit Signal Element Timing (DTE Source)
25	Unsigned

ตารางที่ 2.6 รายละเอียดขาต่างๆของคอนเน็กเตอร์ DB-25

บทที่ 3
การคำนวณและการสร้าง

แนวความคิดการนำไมโครโปรเซสเซอร์มาช่วยในการเก็บข้อมูลระดับสัญญาณเสียง เกิดจากปัญหาในการวัดระดับสัญญาณเสียงรบกวนที่ต้องการหาค่าแบบเฉลี่ย ซึ่งต้องสุ่มวัดระดับสัญญาณเสียงกันอย่างต่อเนื่อง ทำให้ผู้วัดต้องเสียเวลาและเป็นการยุ่งยากเมื่อต้องวัดแบบต่อเนื่องเป็นเวลานานๆ ในการสร้างเครื่องวัดระดับสัญญาณเสียงในโครงงานนี้ได้แบ่งออกเป็นสองส่วน คือส่วนภาครับสัญญาณเสียง และส่วนภาคควบคุมการทำงาน

แนวทางในการออกแบบ

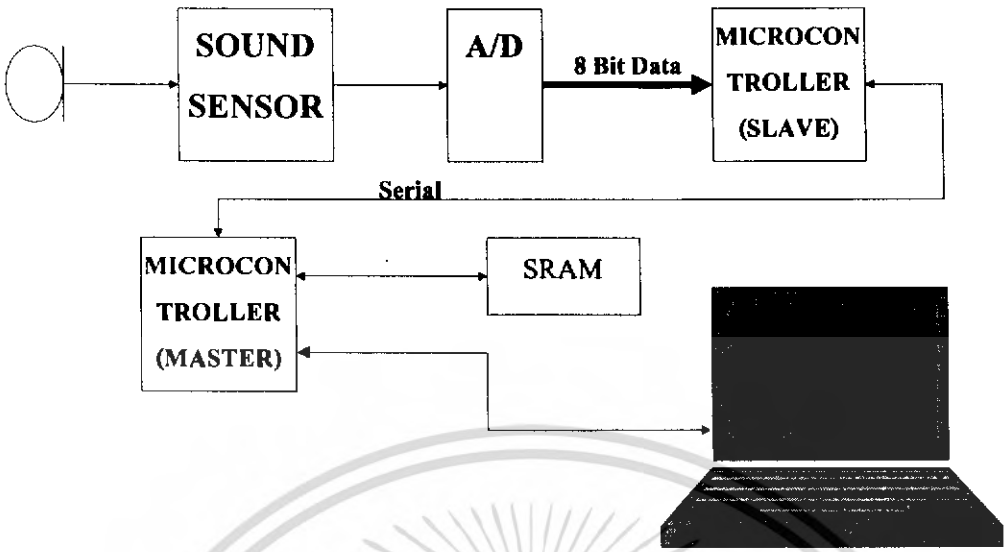
ในการออกแบบต้องกำหนดคุณสมบัติของเครื่องตามความต้องการเพื่อกำหนดขอบเขตของงานให้เหมาะสมกับระยะเวลาที่ใช้งาน โดยจัดทำตัวอุปกรณ์สลาฟขึ้นจำนวน 4 ตัวเพื่อใช้ในการวัดตามจุดต่างๆจำนวน 4 จุด และอุปกรณ์มาสเตอร์จำนวน 1 ตัวใช้ในการรับข้อมูล

คุณสมบัติ

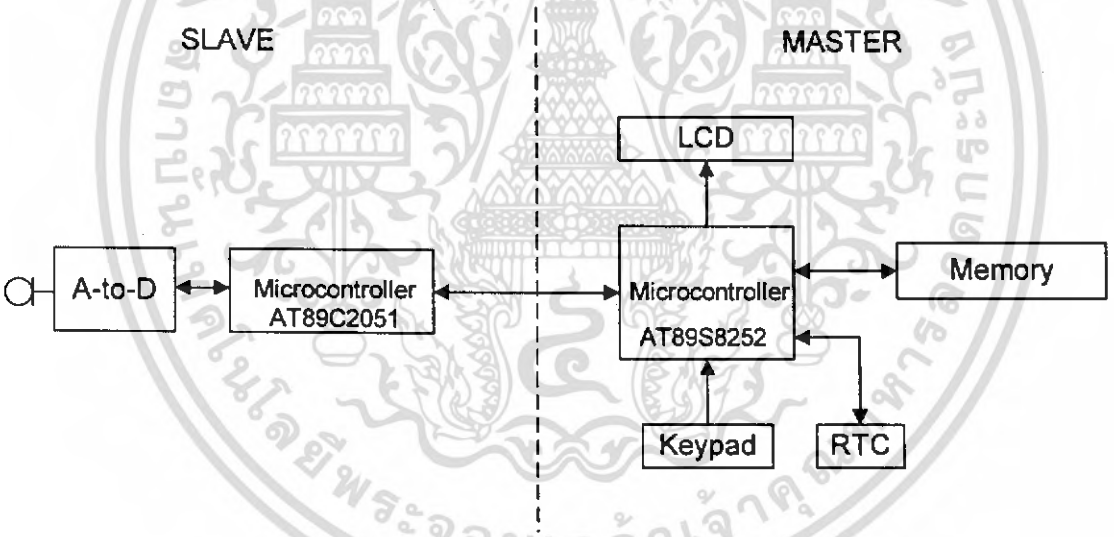
1. แสดงผลการวัดมีหน่วยเป็นเดซิเบล เอ โดยอ่านผลการวัดได้โดยตรง ไม่ต้องบวกกับค่าการลดทอนสัญญาณเสียง
2. สามารถวัดระดับสัญญาณเสียงตามเวลาขณะปัจจุบันได้
3. สามารถตั้งเวลาการวัดได้ โดยกำหนดช่วงเวลาที่ต้องการวัดตามความต้องการของผู้ใช้
4. ขณะทำการวัดระดับสัญญาณเสียงจะทำการแสดงเวลาขณะที่วัด
5. สามารถส่งข้อมูลผลการวัดระดับความดังเสียง ไปเก็บยังเครื่องคอมพิวเตอร์ได้โดยที่ข้อมูลของการวัดจะประกอบด้วย เวลาที่เริ่มต้นวัด เวลาที่สิ้นสุดในการวัด และช่วงระยะห่างในการทำการวัด โดยจะแสดงเวลาขณะที่ทำการวัดและย่านที่ใช้ในการวัดเป็นเดซิเบล เอ
6. สามารถแสดงผลการวัดอย่างคร่าวๆบนตัวเครื่องเอง โดยจะแสดงเพียงผลการวัดระดับสัญญาณเสียง และย่านในการวัดเดซิเบล เอ
7. สามารถตั้งเวลาให้กับ Real Time Clock ซึ่งเป็นฐานเวลาให้กับเครื่องได้

ขั้นตอนการออกแบบ

โดยสัญญาณเสียงที่รับได้ทางไมโครโฟนจะถูกแปลงเป็นสัญญาณไฟฟ้า และนำสัญญาณไปผ่านวงจรขยายสัญญาณขนาดเล็ก (Small Signal Amplifier) เพื่อให้มีกำลำขยายเพิ่มขึ้นจากนั้นนำสัญญาณที่ได้ไปผ่านวงจร Full – Wave Rectifier เพื่อให้แรงดันอยู่ในช่วง 0 ถึง 5 โวลท์ เพื่อให้สัญญาณสามารถผ่านเข้าวงจรแปลงสัญญาณจากอนาล็อกเป็นสัญญาณดิจิตอล (Analog To Digital) ได้ และสัญญาณดิจิตอลที่ได้จะถูกนำไปเก็บไว้ในหน่วยความจำโดยใช้ไมโครคอนโทรลเลอร์ในการควบคุม



รูปที่ 3.1 บล็อคไดอะแกรมการทำงานของโครงงาน



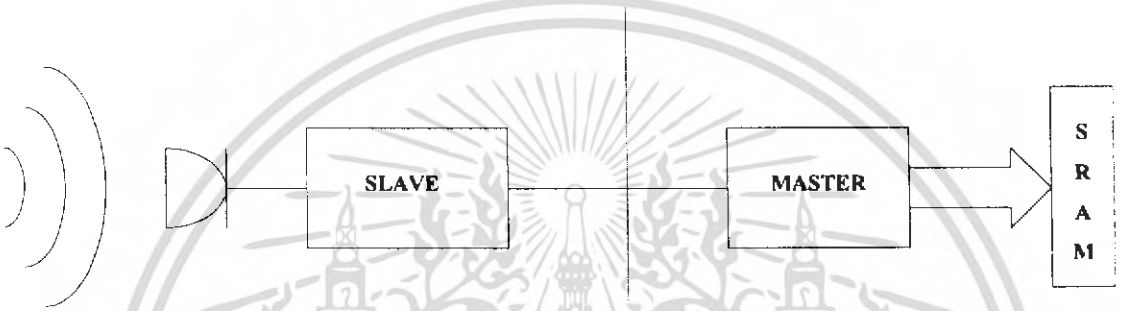
รูปที่ 3.2 บล็อกไดอะแกรมของวงจรรวม

3.1 การออกแบบวงจรโดยรวม

ในลำดับแรกขออธิบายการทำงานของระบบโดยรวมก่อน โดยระบบที่ออกแบบนั้นใช้ในการเก็บข้อมูลของระดับความดังเสียงโดยที่เสียงที่เป็นสัญญาณอนาล็อกนั้นจะถูกแปลงเป็นสัญญาณไฟฟ้าโดยวงจรไมโครโฟนก่อนจากนั้นแรงดันทางไฟฟ้าจะถูกแปลงให้เป็นข้อมูลทางดิจิตอลโดยใช้วงจรแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล ซึ่งเราเลือกใช้ ไอซี ADC0820 นำไปเชื่อมต่อกับ

ไมโครคอนโทรลเลอร์เบอร์ AT89C2051 ซึ่งใช้ในการควบคุมการทำงานของวงจรแปลงสัญญาณจากอนาล็อกเป็นสัญญาณดิจิตอลและยังเป็นตัวที่ทำหน้าที่ส่งข้อมูลที่ได้จากวงจรแปลงสัญญาณจากอนาล็อกเป็นสัญญาณดิจิตอลไปให้กับไมโครคอนโทรลเลอร์อีกตัวหนึ่งซึ่งเป็นเบอร์ AT89S8250 โดยที่ AT89S8250 จะนำข้อมูลที่ได้รับทางพอร์ทอนุกรมนำไปเก็บในหน่วยความจำแรมต่อไป

จากที่ได้กล่าวมาแล้วข้างต้นนั้นจะเห็นได้ว่าในการออกแบบวงจรไมโครคอนโทรลเลอร์นั้นเราจะเห็นว่ามีการใช้ไมโครคอนโทรลเลอร์ทำงานร่วมกัน 2 ตัว โดยในที่นี้จะขอเรียกอุปกรณ์ที่ทำหน้าที่เกี่ยวกับการแปลงสัญญาณเสียงเป็นข้อมูลดิจิตอลว่าอุปกรณ์สลาฟ (Slave) และอุปกรณ์ที่ทำหน้าที่รับข้อมูลมาและจัดเก็บในหน่วยความจำว่าอุปกรณ์มาสเตอร์ (Master) โดยแสดงเป็นแผนผังวงจรได้ดังนี้



รูปที่ 3.3 การทำงานระหว่างอุปกรณ์สลาฟและอุปกรณ์มาสเตอร์

ดังนั้นแนวทางในการออกแบบจึงสามารถแบ่งส่วนของงานออกเป็น 2 ส่วนใหญ่ๆดังนี้

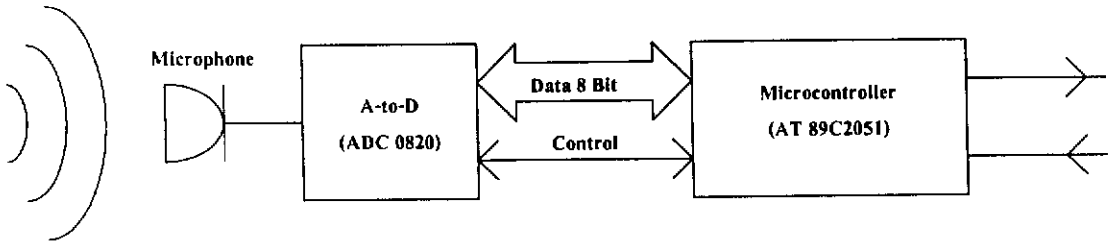
1. การออกแบบในส่วนของอุปกรณ์สลาฟ (Slave)
2. การออกแบบในส่วนของอุปกรณ์มาสเตอร์ (Master)

ซึ่งรายละเอียดจะกล่าวในหัวข้อต่อไปนี้

3.2 การออกแบบอุปกรณ์สลาฟ (Slave)

3.2.1 หน้าที่การทำงาน

เมื่อเริ่มต้นการทำงานอุปกรณ์สลาฟจะอยู่ในโหมดสแตนด์บาย คือมันจะรอคอยคำสั่งเริ่มการทำงานที่จะส่งมาจากอุปกรณ์มาสเตอร์ เมื่อมีคำสั่งเข้ามาจะมีการสั่งให้วงจรแปลงสัญญาณจากอนาล็อกเป็นสัญญาณดิจิตอลในอุปกรณ์สลาฟทำการแปลงแรงดันไฟฟ้าที่รับได้จากวงจรไมโครโฟนไปเป็นข้อมูลดิจิตอลขนาด 8 บิท จากนั้นอุปกรณ์สลาฟจะจัดส่งข้อมูลชุดนี้กลับไปให้อุปกรณ์มาสเตอร์เพื่อจัดเก็บ จากนั้นอุปกรณ์สลาฟก็กลับมาในสถานะสแตนด์บายเหมือนเดิมโดยแผนผังของวงจรในอุปกรณ์สลาฟเป็นดังนี้

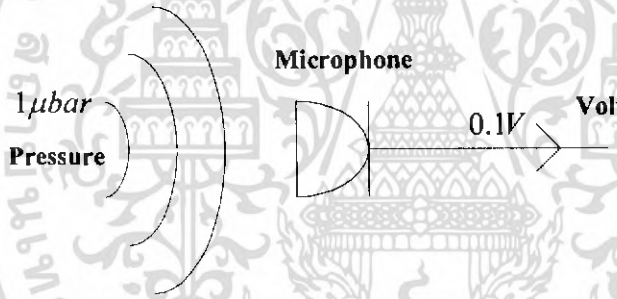


รูปที่ 3.4 การทำงานงานของอุปกรณ์สลาฟ

3.2.2 อุปกรณ์ที่ใช้ในตัวสลาฟ

3.2.2.1 ไมโครโฟน (Microphone)

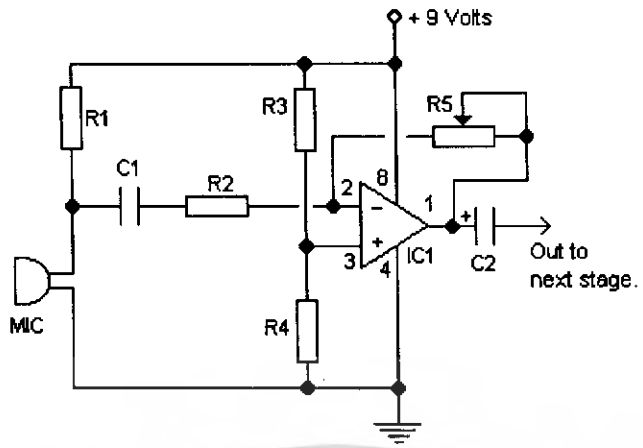
ในวงจรจะใช้คอนแคนซ์เซอร์ไมโครโฟน ซึ่งมีหน้าที่แปลงแรงดันอากาศที่มาตกกระทบไปเป็นแรงดันทางไฟฟ้า ซึ่งเราได้ใช้ไมโครโฟนที่มีค่าความไว (Sensitivity) 0.1 โวลท์ต่อไมโครบาร์ ($V / \mu Bar$)



รูปที่ 3.5 การแปลงแรงดันอากาศที่มาตกกระทบไปเป็นแรงดันทางไฟฟ้าของไมโครโฟน

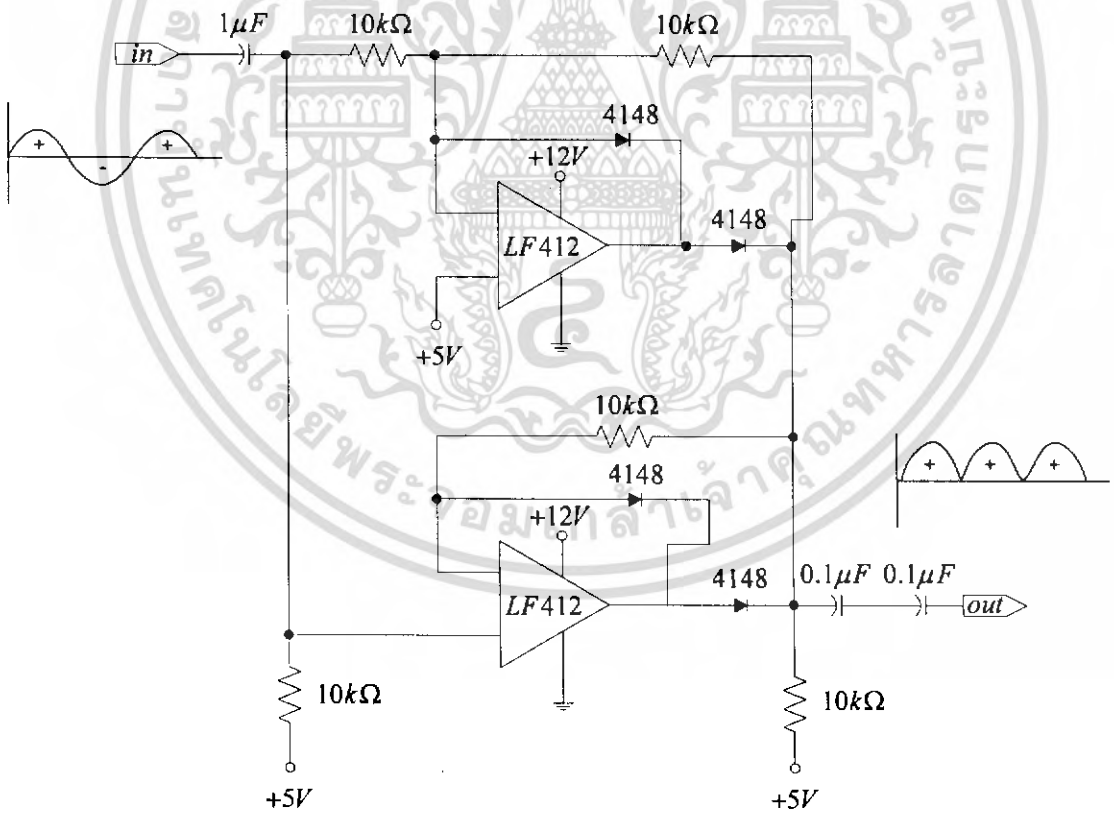
3.2.2.2 วงจรขยายสัญญาณขนาดเล็ก (Small Signal Amplifier)

ใช้ในการขยายสัญญาณที่ได้จากไมโครโฟนซึ่งมีขนาดเล็กมากให้มีขนาดที่ใหญ่ขึ้นก่อนที่จะป้อนเข้าสู่วงจรแปลงสัญญาณจากอนาล็อกเป็นสัญญาณดิจิตอลโดยในส่วนของวงจรขยายสัญญาณขนาดเล็กได้เลือกใช้ไอซี LM 358



รูปที่ 3.6 วงจรขยายสัญญาณขนาดเล็ก (Small Signal Amplifier)

เอาท์พุทที่ออกจากวงจรขยายสัญญาณขนาดเล็กในรูปที่ 3.6 จะได้เอาท์พุทเป็นสัญญาณไซน์เวฟ (Sine-Wave) นำสัญญาณที่ออกจากวงจรขยายสัญญาณขนาดเล็กไปผ่านวงจร Full – Wave Rectifier ดังรูปที่ 3.7

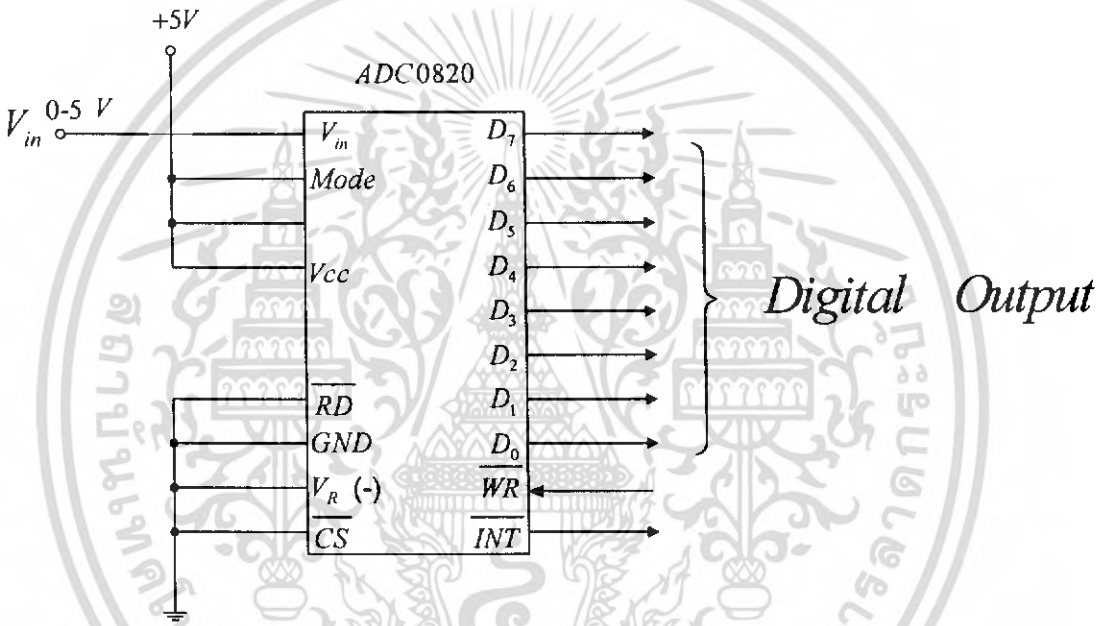


รูปที่ 3.7 วงจร Full – Wave Rectifier

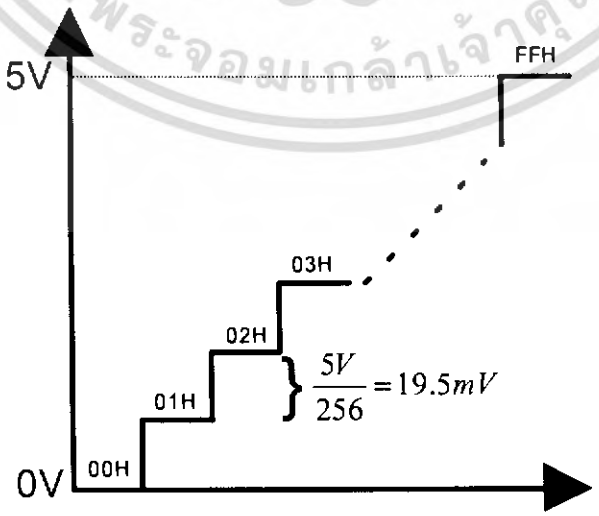
นำสัญญาณที่ได้ออกมาจากวงจร Full – Wave Rectifier ในรูปที่ 3.7 (0 ถึง 5โวลท์) ไปทำการแปลงเป็นสัญญาณดิจิตอลเพื่อนำสัญญาณนี้ไปเก็บในหน่วยความจำในขั้นต่อไป

3.2.2.3 วงจรแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล (Analog- to-Digital Converter)

ในวงจรนี้ได้เลือกใช้ไอซี ADC0820 รับอินพุทที่เป็นสัญญาณอนาล็อก จากวงจรวงจรขยายสัญญาณขนาดเล็กเข้ามาทางขา Vin (ขา1) และแปลงเป็นสัญญาณดิจิตอลเอาท์พุทมีขนาด 8 บิทออกทางขา D0-D7 โดยในการทำงานจะมีสัญญาณควบคุมเข้ามาทางขา $\overline{WR}$ (ขา16) โดยถ้ามีพัลส์ "0" เข้ามาที่ขา $\overline{WR}$ แล้วไอซี ADC0820 ก็จะเริ่มการทำงานและเมื่อการทำงานเสร็จสิ้น (ใช้เวลาประมาณ $1.5\mu s$) ก็จะมีการแจ้งไปยังไมโครคอนโทรลเลอร์



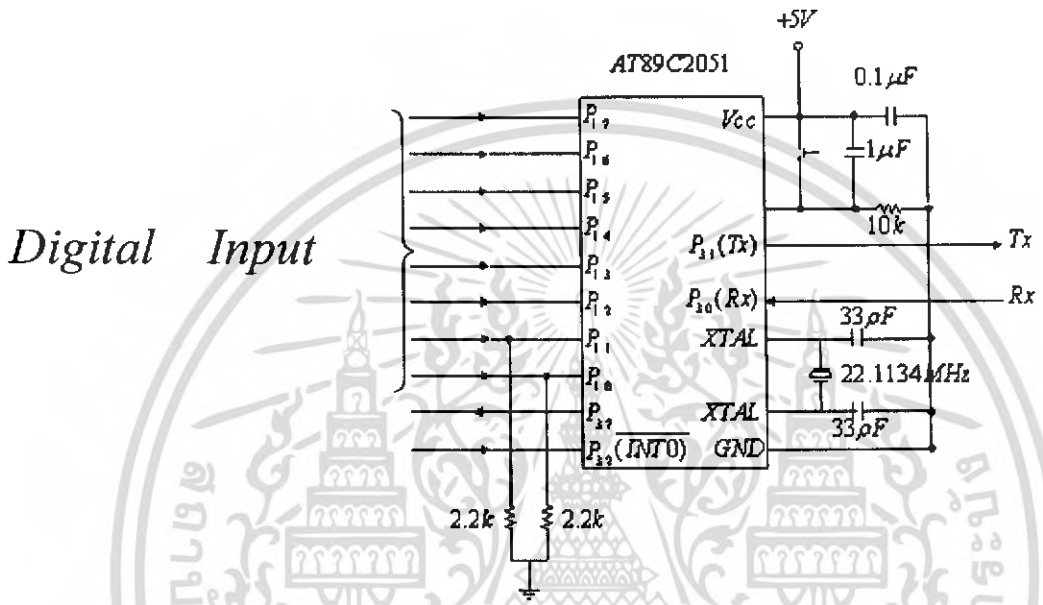
รูปที่ 3.8 ไอซี ADC0820



รูปที่ 3.9 การเข้ารหัสสัญญาณในวงจร Analog to Digital ขนาด 8 บิท (256 ค่า)

3.2.2.4 ไมโครคอนโทรลเลอร์ (Microcontroller)

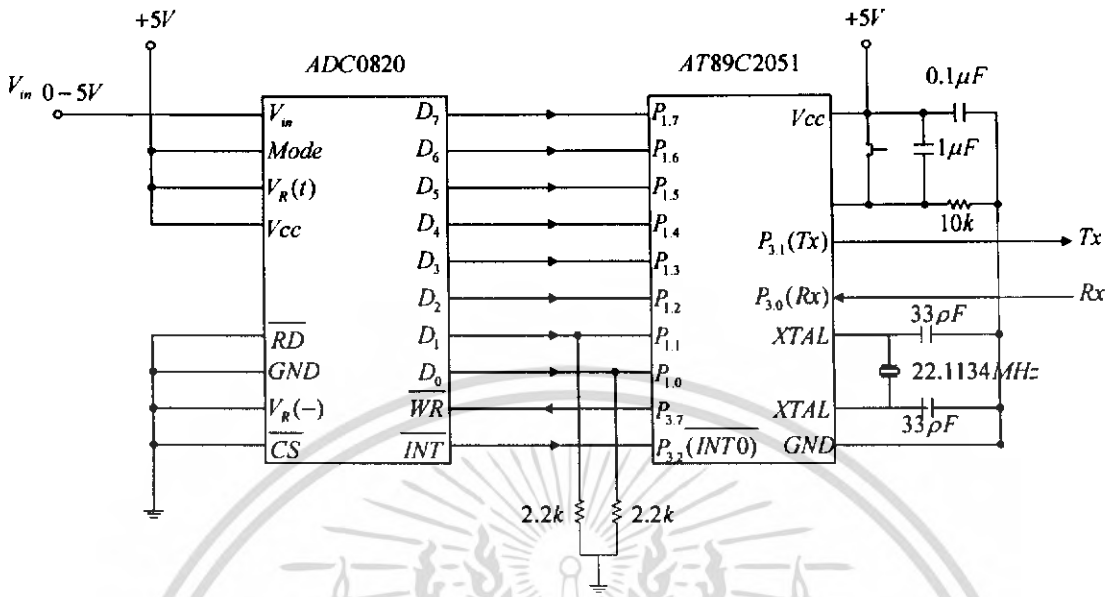
ไมโครคอนโทรลเลอร์ถือเป็นส่วนที่สำคัญที่สุดของอุปกรณ์สลาฟ เพราะเป็นตัวที่ควบคุมการทำงานให้กับไอซี ADC0820 และยังมีหน้าที่ติดต่อสื่อสารกับอุปกรณ์มาสเตอร์ผ่านทางพอร์ทอนุกรมอีกด้วย โดยในส่วนของอุปกรณ์สลาฟนี้เลือกใช้ไอซี AT89C2051 ซึ่งมีพอร์ทอินพุทจำนวน 2 พอร์ทและมีหน่วยความจำโปรแกรมภายในขนาด 2 กิโลไบท์ ซึ่งเพียงพอกับขนาดของโปรแกรมที่ใช้สำหรับควบคุมการทำงานของอุปกรณ์สลาฟ



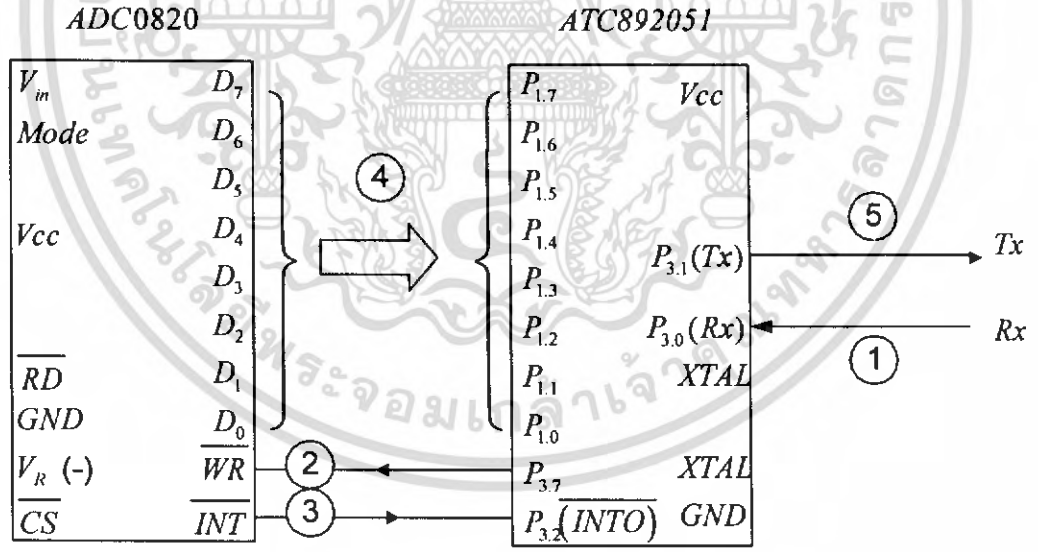
รูปที่ 3.10 วงจรไอซี AT89C2051

จากรูปเมื่อมีการสั่งการมาจากอุปกรณ์มาสเตอร์ผ่านทาง Rx (ขา2) ของไอซี AT89C2051 จะมีการสร้างพัลส์ "0" ออกมาทางพอร์ท 3.7 (ขา11) เพื่อสั่งการให้ไอซี ADC0820 เริ่มทำการแปลงสัญญาณจากอนาล็อกเป็นดิจิตอล และเมื่อมีพัลส์ "0" เข้ามาที่ขา $\overline{INT0}$ (ขา6) ไอซี AT89C2051 ก็จะทำการอ่านข้อมูลดิจิตอลขนาด 8 บิทที่เข้ามาทางพอร์ท 1 แล้วทำการส่งข้อมูลชุดนี้ออกทาง Tx (ขา3) ไปให้กับอุปกรณ์มาสเตอร์ต่อไป

เมื่อนำวงจรในส่วนต่างๆมาต่อรวมกันเป็นอุปกรณ์ในส่วนของตัวสลาฟ



รูปที่ 3.11 วงจรของอุปกรณ์สลาฟ



รูปที่ 3.12 การทำงานระหว่างไอซี ADC0820 กับไอซี ATC89C2051

โดยเราสามารถอธิบายรูปได้ดังนี้

1. คือสัญญาณควบคุมจากอุปกรณ์มาสเตอร์เข้ามาทางขา Rx
2. คืออุปกรณ์สลาฟสั่งการให้ไอซี ADC0820 ทำงานทางขา P3.7

3. คือสัญญาณอินเทอร์รัพท์จากไอซี ADC0820 เป็นการบอกว่าทำการแปลงสัญญาณจากสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลเรียบร้อยแล้ว
4. คือไอซี AT89C2051 ทำการอ่านค่าข้อมูลดิจิตอลจากพอร์ท 1
5. คือข้อมูลจากพอร์ท 1 จะถูกส่งไปให้กับอุปกรณ์มาสเตอร์ทางขา Tx

3.3 การออกแบบอุปกรณ์มาสเตอร์ (Master)

3.3.1 หน้าที่และการทำงานของอุปกรณ์มาสเตอร์

อุปกรณ์มาสเตอร์นั้นมีหน้าที่และการทำงานที่มากและซับซ้อนกว่าในส่วนของอุปกรณ์สลาฟ เพราะนอกจากจะเป็นผู้ควบคุมให้อุปกรณ์สลาฟทำงานแล้วนั้น อุปกรณ์มาสเตอร์ยังต้องมีการติดต่อกับผู้ใช้งานและยังต้องสามารถเก็บข้อมูลในหน่วยความจำได้อีกด้วย โดยเราสามารถสรุปเป็นฟังก์ชั่นการทำงานของอุปกรณ์มาสเตอร์ได้ดังนี้

1) ติดต่อสื่อสารกับอุปกรณ์สลาฟผ่านทางพอร์ทอนุกรม
2) แสดงสถานะการทำงานให้ผู้ใช้ทราบผ่านทางหน้าจอแอลซีดี
3) รับค่าอินพุทจากผู้ใช้ผ่านทางคีย์แพด
4) ติดต่อกับฐานข้อมูลทางเวลา (Real Time Clock)
5) เข้ากับข้อมูลภายในหน่วยความจำภายนอกขนาด 64 Kbytes

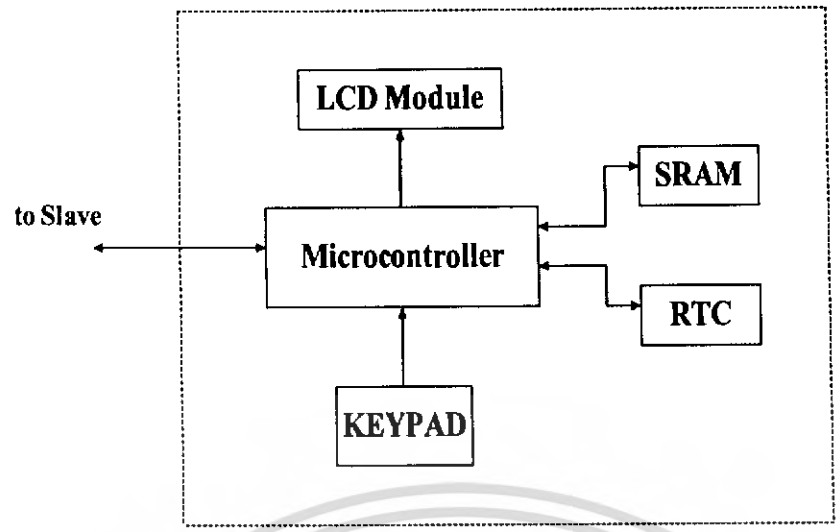
3.3.2 การออกแบบวงจรส่วนต่างๆในอุปกรณ์มาสเตอร์

ในการออกแบบอุปกรณ์มาสเตอร์นั้นมีความซับซ้อนพอสมควรเพราะมีการนำอุปกรณ์หลายๆ อย่างเข้ามาต่อรวมกันจึงเกิดปัญหาคือ จำนวนของขาสัญญาณของไมโครคอนโทรลเลอร์นั้นมีไม่เพียงพอที่จะใช้ควบคุมอุปกรณ์ได้ทุกตัว ดังนั้นจึงได้มีการใช้ระบบบัสแบบ I^2C ซึ่งมีการใช้สายสัญญาณเพียง 2 เส้น ในการเชื่อมต่อไมโครคอนโทรลเลอร์กับอุปกรณ์ต่างๆ จึงเป็นการลดจำนวนของสายสัญญาณลงไปได้มาก

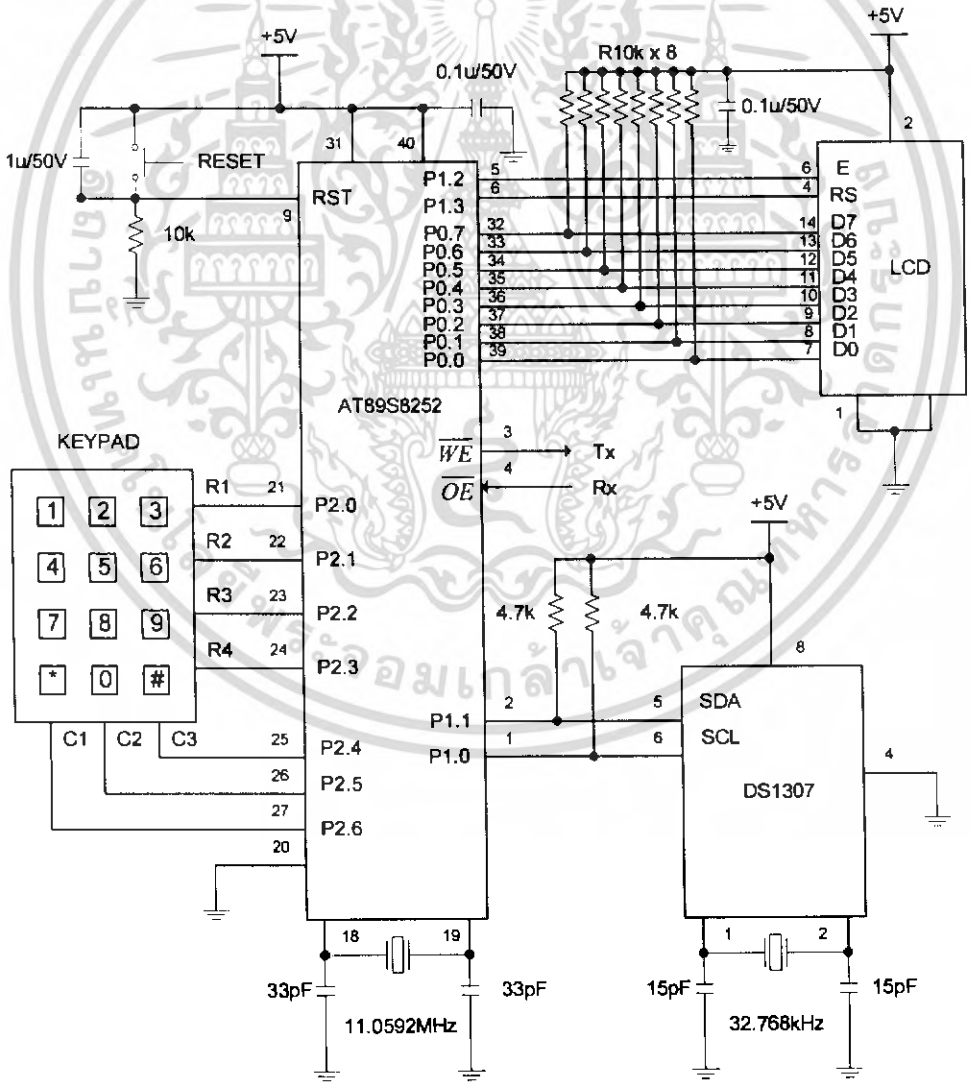
จากฟังก์ชั่นการทำงานของอุปกรณ์มาสเตอร์ดังที่กล่าวไปเมื่อหัวข้อที่แล้ว ทำให้เราพอที่จะทราบได้ว่าอุปกรณ์มาสเตอร์นั้นจะต้องประกอบไปด้วยอุปกรณ์ต่างๆดังต่อไปนี้

1) แอลซีดี โมดูล (LCD Module) : ใช้แสดงสถานการณ์ทำงานกับผู้ใช้
2) คีย์แพด (KEYPAD) : ใช้รับค่าอินพุทจากผู้ใช้
3) หน่วยความจำภายใน (SRAM) : ใช้เก็บข้อมูลต่างๆ
4) Real Time Clock (RTC) : ใช้เพื่อเป็นฐานข้อมูลทางเวลาให้กับอุปกรณ์
5) ไมโครคอนโทรลเลอร์ (Microcontroller) : ถือเป็นสมองของอุปกรณ์มาสเตอร์ใช้ควบคุมการทำงานของอุปกรณ์

โดยเราสามารถเขียนเป็น บล็อกไดอะแกรมในส่วนของอุปกรณ์มาสเตอร์ได้ดังนี้



รูปที่ 3.13 บล็อคไดอะแกรมในส่วนของอุปกรณ์มาสเตอร์

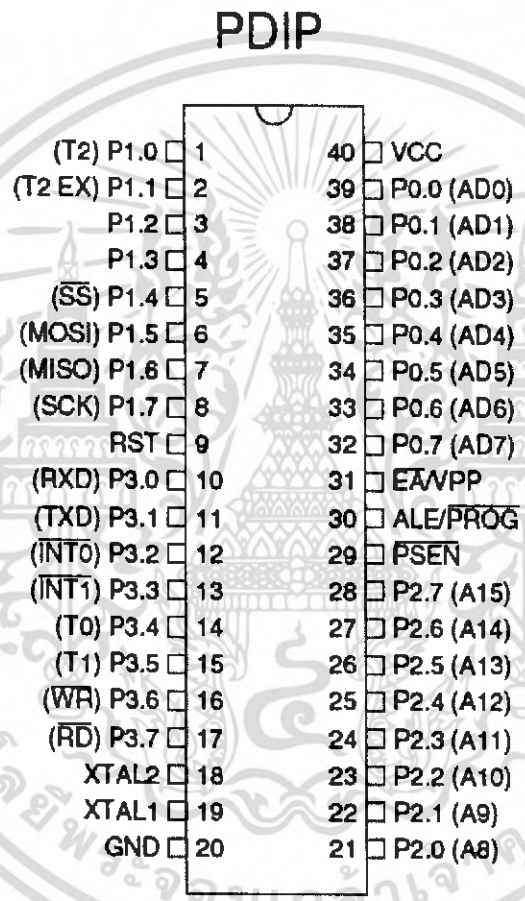


รูปที่ 3.14 วงจรรวมของอุปกรณ์มาสเตอร์

โดยในโครงงานนี้สามารถแบ่งการออกแบบอุปกรณ์มาสเตอร์เป็นส่วนต่างๆได้ดังนี้

3.3.2.1 ไมโครคอนโทรลเลอร์ เบอร์ AT89S8252

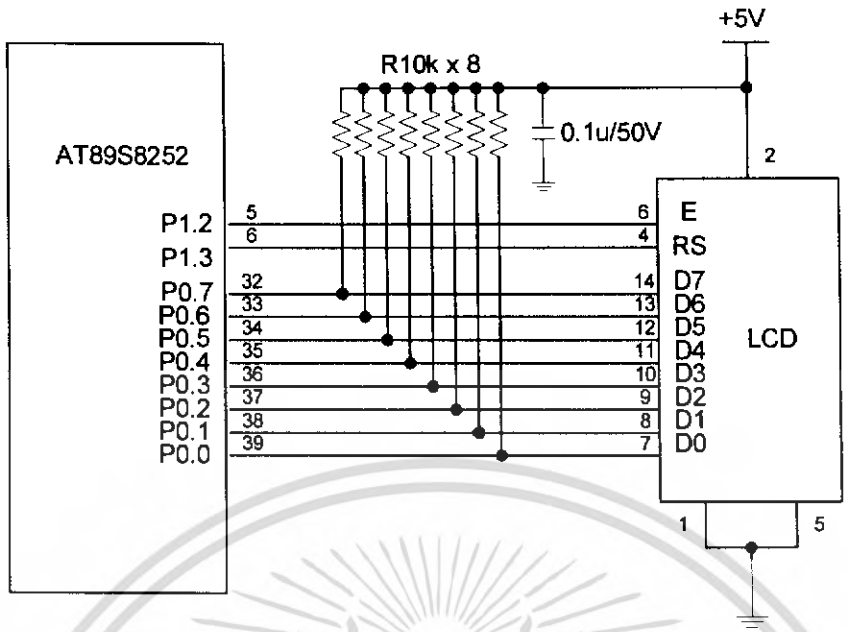
ไมโครคอนโทรลเลอร์ถือเป็นหัวใจหลักของอุปกรณ์มาสเตอร์เพราะมีหน้าที่ควบคุมการทำงานทั้งหมด และเนื่องจากอุปกรณ์ต่อพ่วงที่มีมากทำให้โปรแกรมที่ใช้ในการควบคุมจึงมีมากและซับซ้อนตามไปด้วยซึ่งต่างจากอุปกรณ์สลาฟที่มีอุปกรณ์ต่อพ่วงแค่ตัวเดียว ดังนั้นในโครงงานนี้จึงเลือกใช้ไมโครคอนโทรลเลอร์เบอร์ AT89S8252 ซึ่งมีหน่วยความจำโปรแกรมภายใน 8 k ซึ่งสามารถรองรับขนาดของโปรแกรมได้เพียงพอและยังมีพื้นที่เหลือสำหรับการเขียนโปรแกรมเพิ่มเติมไปในอนาคตอีกด้วย



รูปที่ 3.15 ไอซี AT89S8252

3.3.2.2 โมดูล แอลซีดี (LCD Module)

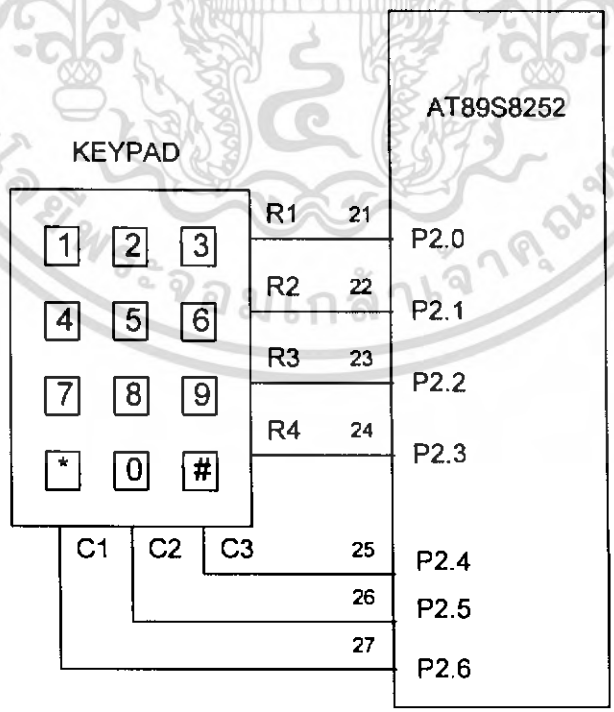
โมดูล แอลซีดี ใช้ในการแสดงสถานะการทำงานให้ผู้ใช้ทราบโดย โมดูล แอลซีดี ที่ใช้ในโครงงานนี้เป็นโมดูลแอลซีดีเบอร์ CCM-1620CSL-V2 ซึ่งสามารถแสดงผลได้ 2 บรรทัด บรรทัดละ 16 ตัวอักษร โดยมีการเชื่อมต่อกับ AT89S8252 ดังรูป



รูปที่ 3.16 แสดงการเชื่อมต่อวงจรแอลซีดีเข้ากับไมโครคอนโทรลเลอร์

3.3.2.3 คีย์แพด (Keypad)

คีย์แพดใช้เพื่อรับอินพุทจากผู้ใช้ โดยคีย์แพดที่ใช้จะเป็นแบบ 4 แถว 3 คอลัมน์ โดยเชื่อมต่ออยู่กับ พอร์ท 2 ของ AT89S8252 ดังรูป

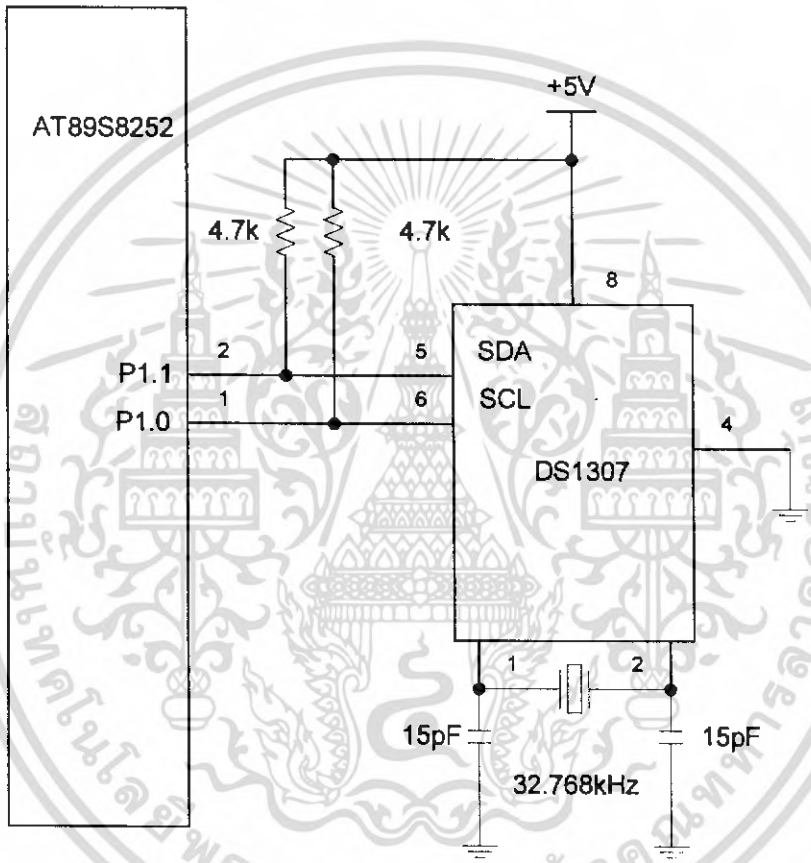


รูปที่ 3.17 การเชื่อมต่อส่วนของคีย์แพดเข้ากับไมโครคอนโทรลเลอร์

3.3.2.4 Real Time Clock (RTC)

RTC มีไว้เพื่อสร้างฐานข้อมูลทางเวลาให้กับอุปกรณ์มาสเตอร์ โดยค่าเวลาเริ่มต้นของการทำงานที่ผู้ใช้ป้อนเข้ามาผ่านทางคีย์แพดจะถูกบันทึกใน RTC และไมโครคอนโทรลเลอร์จะใช้วิธีอ่านข้อมูลเวลาจาก RTC เพื่อเริ่มต้นการส่งสัญญาณควบคุมไปยังอุปกรณ์สลาฟตามเวลาที่กำหนดในโปรแกรม

ในโครงงานนี้เลือกใช้ไอซี DS1307 ซึ่งเป็น Real Time Clock ที่สามารถเชื่อมต่อบนระบบบัส I^2C ได้ โดยวงจรการเชื่อมต่อกับ AT89S8252 สามารถแสดงได้ดังนี้



รูปที่ 3.18 การเชื่อมต่อวงจร Real Time Clock เข้ากับไมโครคอนโทรลเลอร์

3.3.2.5 หน่วยความจำข้อมูลภายนอก (SRAM)

ในโครงงานนี้จะมีการนำข้อมูลที่ได้จากการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลแต่ละครั้งเก็บในหน่วยความจำภายนอก ซึ่งปริมาณข้อมูลจะมากหรือน้อยขึ้นอยู่กับปัจจัยต่อไปนี้

1) ความถี่ของการเก็บข้อมูล

2) จำนวนอุปกรณ์สลาฟที่ต่ออยู่กับอุปกรณ์มาสเตอร์

โดยยิ่งถ้าปริมาณข้อมูลมีมากขึ้นเท่าไหร่ ระยะเวลารวมที่อุปกรณ์มาสเตอร์จะสามารถเก็บข้อมูลได้ก่อนที่จะเต็มความจุของหน่วยความจำภายใน (SRAM) ก็จะยิ่งน้อยลง ซึ่งเราสามารถเขียนเป็นความสัมพันธ์ได้ดังนี้

$$T_t = \frac{Capacity \times T_s}{n}$$

เมื่อ T_t = ระยะเวลารวมที่อุปกรณ์มาสเตอร์เก็บข้อมูลได้ก่อนความจุของหน่วยความจำจะเต็ม

$Capacity$ = ความจุของหน่วยความจำแรม (RAM)

T_s = คาบเวลาในการเก็บข้อมูลแต่ละครั้ง

n = จำนวนของอุปกรณ์สลาฟที่ต่อพ่วงอยู่

โดยในโครงงานนี้เลือกใช้หน่วยความจำภายใน (SRAM) เบอร์ CY62256LL – 70DC ซึ่งมีความจุ 32 Kbytes 2 ตัวทำงานร่วมกันทำให้มีความจุรวมทั้งหมด 64 Kbytes ดังนั้นถ้ากำหนดให้คาบเวลาในการเก็บข้อมูลแต่ละครั้งห่างกัน 15 วินาที และมีอุปกรณ์สลาฟต่อพ่วงอยู่เพียงตัวเดียว เราจะได้ค่าเวลารวมดังนี้

$$T_t = \frac{64k \times 15}{1}$$

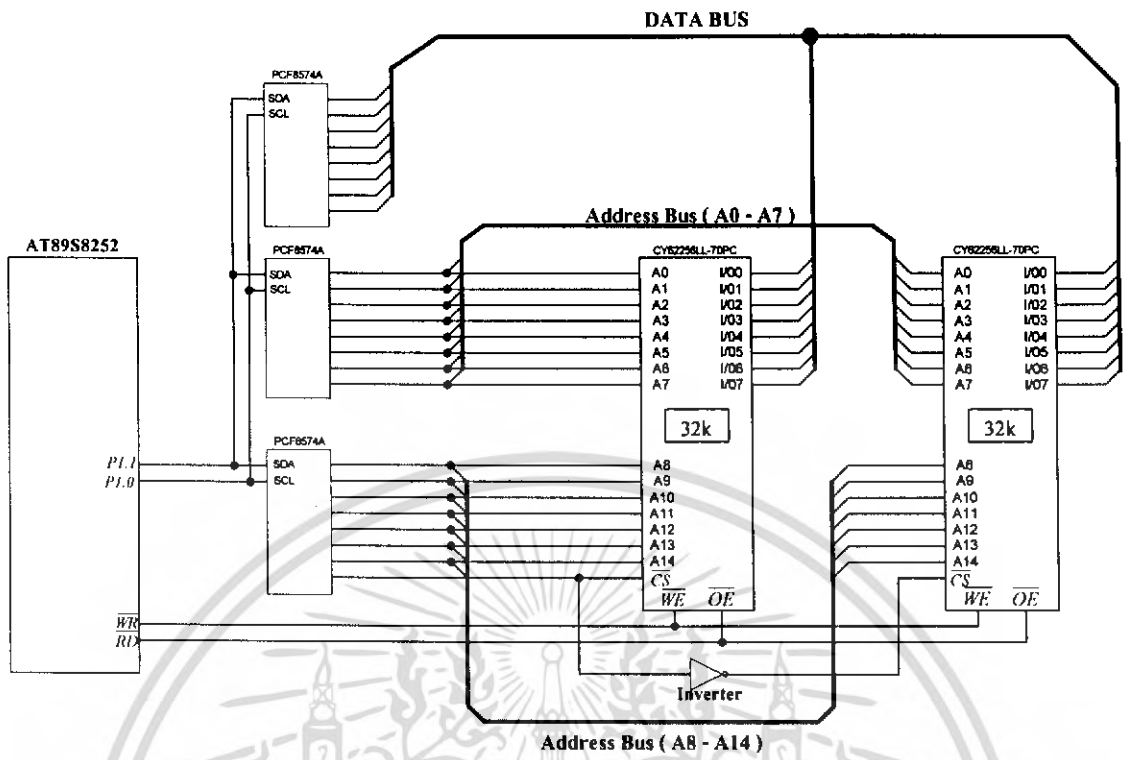
$$= 64 \times 1024 \times 15$$

$$= 983,040 \quad \text{วินาที}$$

หรือ $T_t \approx 11$ วัน

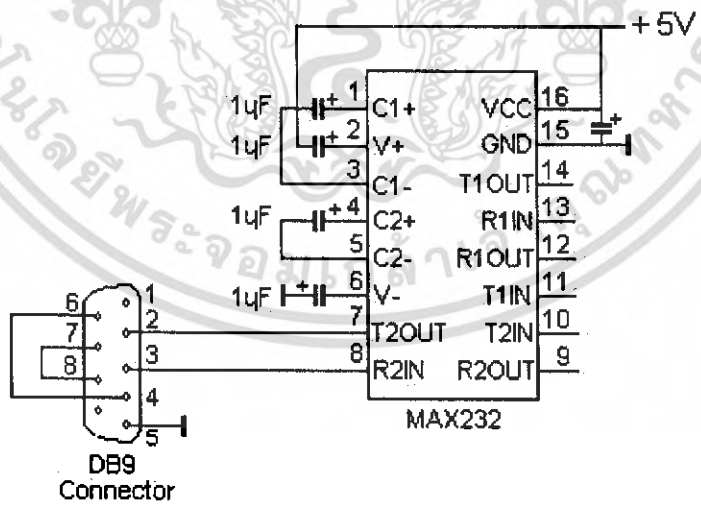
ดังนั้นหมายความว่า ข้อมูลที่เก็บอยู่ในหน่วยความจำจะเป็นข้อมูลของ 11 วันล่าสุดเท่านั้น

ในการเชื่อมต่อกับไมโครคอนโทรลเลอร์นั้น เนื่องจากการเชื่อมต่อหน่วยความจำภายใน (SRAM) ขนาด 64 Kbytes จะต้องใช้ขาสัญญาณในการกำหนด Address ของหน่วยความจำเป็นจำนวน 16 ขา หรือต้องใช้ Port ของไมโครคอนโทรเลอร์ถึง 2 พอร์ท ซึ่งไม่เพียงพอ จึงได้มีการใช้ IC PEF8574A ซึ่งเป็น IC ที่ใช้ในการขยาย I/O พอร์ท ซึ่งสามารถเชื่อมต่อบนบัสแบบ I^2C ได้ จึงทำให้สามารถเชื่อมต่อกับไมโครคอนโทรลเลอร์โดยใช้สายสัญญาณเพียง 2 เส้นเท่านั้น และรูปวงจรการเชื่อมต่อหน่วยความจำเข้ากับ AT89S8252 โดยผ่านอุปกรณ์ขยายพอร์ท PCF8574A สามารถแสดงได้เป็นดังนี้

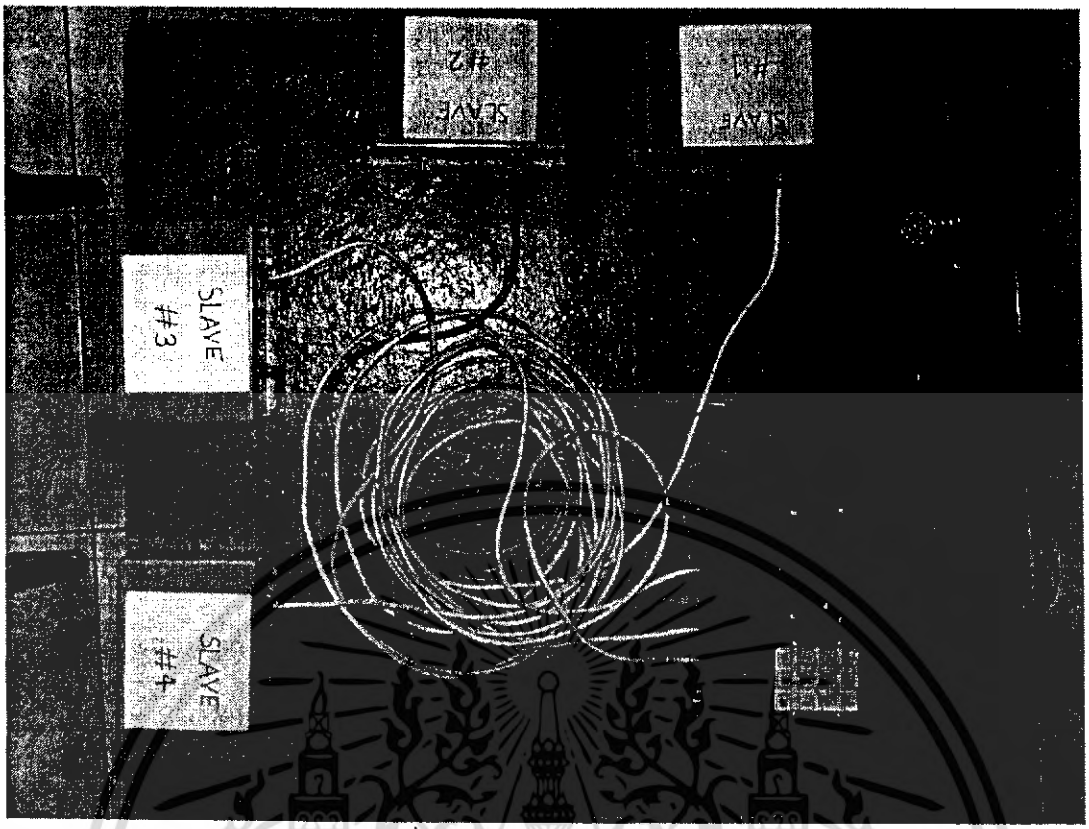


รูปที่ 3.19 วงจรแสดงการเชื่อมต่อกับหน่วยความจำภายใน (SRAM)

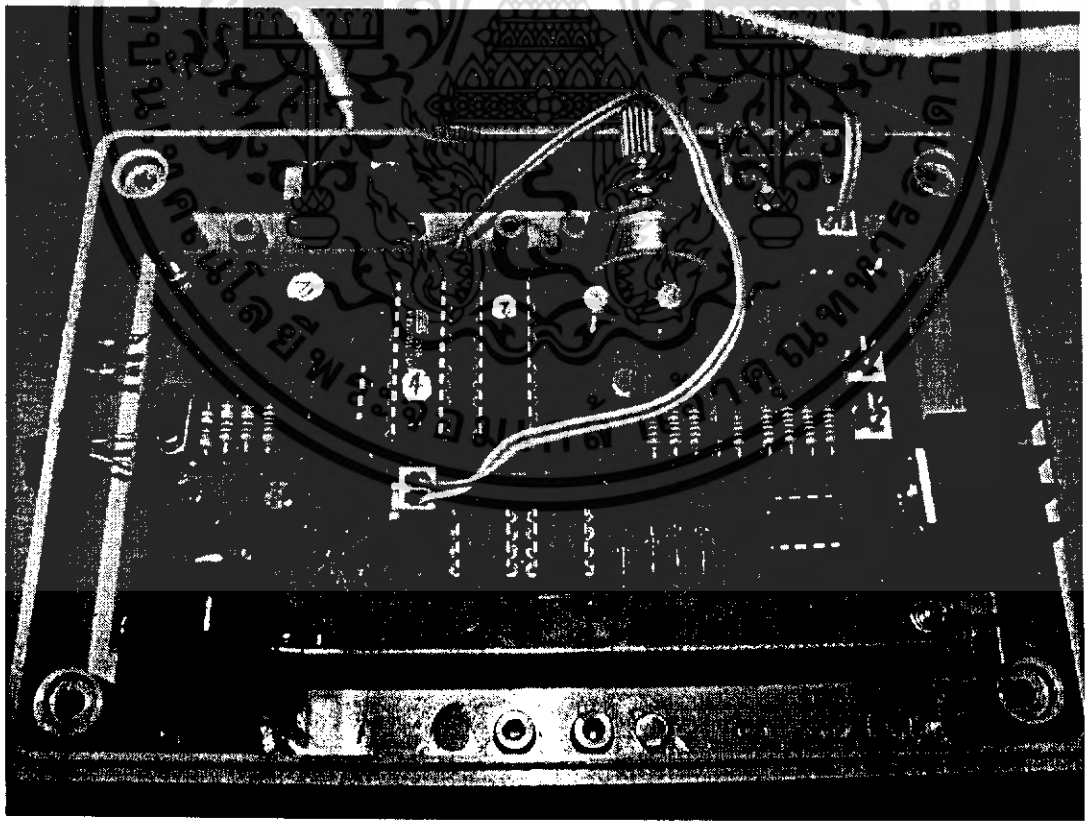
3.3.2.6 การเชื่อมต่อทาง serial port กับคอมพิวเตอร์



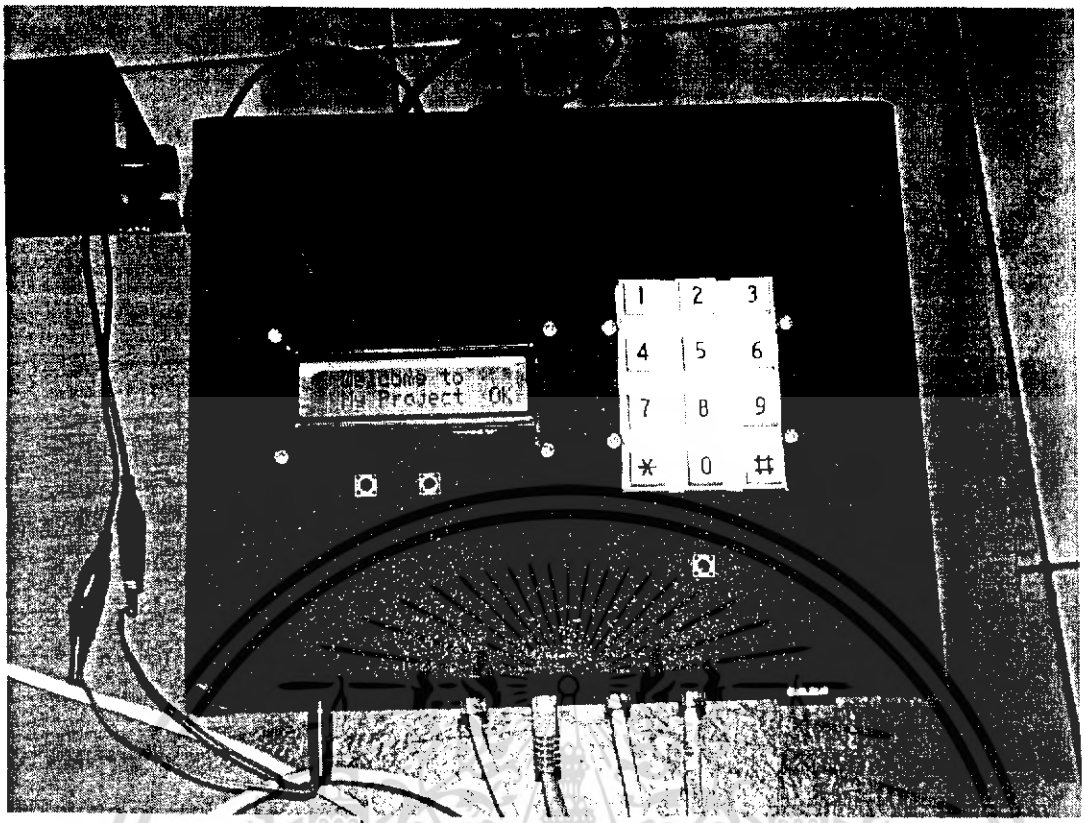
รูปที่ 3.20 วงจรส่งสัญญาณข้อมูลจากเครื่องคอมพิวเตอร์



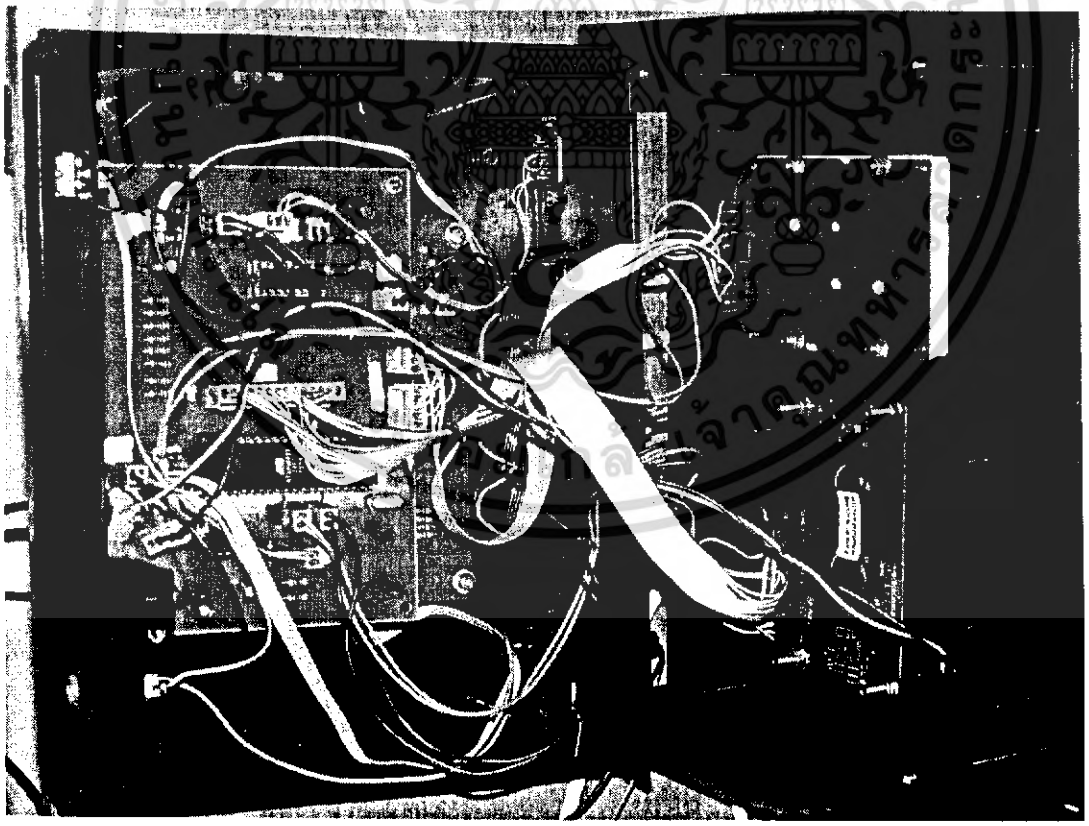
รูปที่ 3.21 แสดงอุปกรณ์โดยรวม



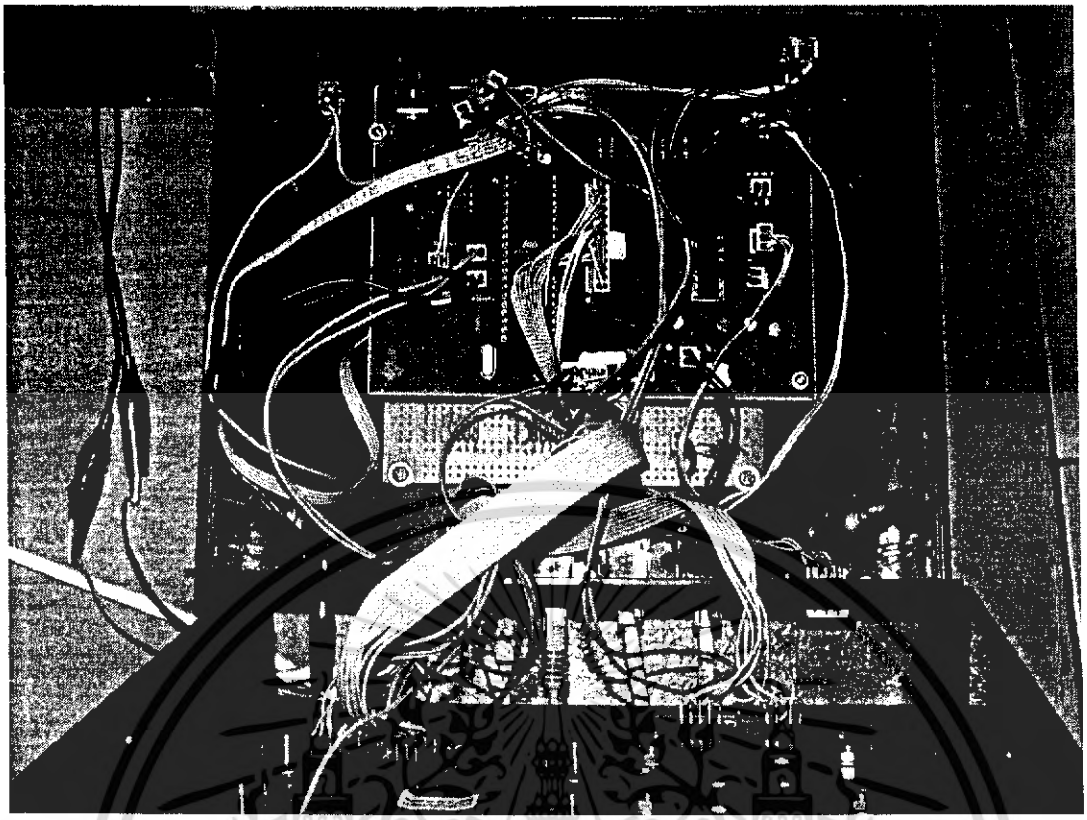
รูปที่ 3.22 แสดงอุปกรณ์ภายในของวงจรสลาฟ



รูปที่ 3.23 แสดงอุปกรณ์ของวงจรมาสเตอร์ (1)



รูปที่ 3.24 แสดงอุปกรณ์ภายในของวงจรมาสเตอร์ (2)



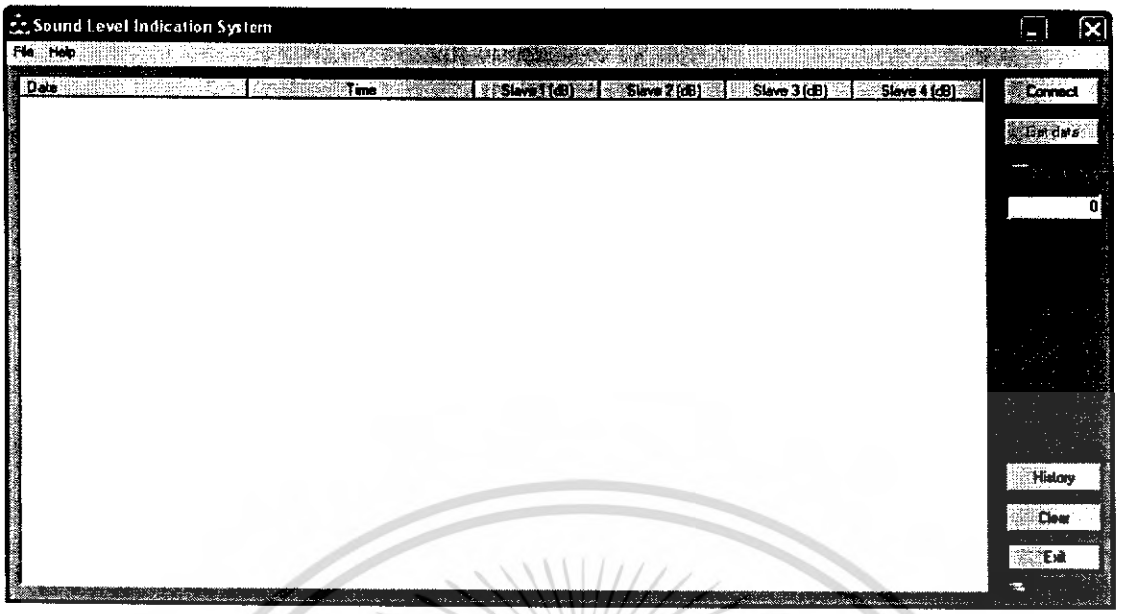
รูปที่ 3.25 แสดงอุปกรณ์ภายในของวงจรมาสเตอร์ (3)

3.4 ส่วนของโปรแกรมการแสดงผลบนหน้าจอคอมพิวเตอร์

เมื่อส่วนของวงจรสลาฟได้วัดระดับความดังเสียงและทำการส่งข้อมูลเสียงไปยังส่วนของมาสเตอร์เพื่อทำการเก็บและแสดงระดับความดังเสียงที่วัดได้ไว้ที่แรมและแสดงผลที่หน้าจอแอลซีดี จากนั้นจึงทำการเชื่อมต่อกับคอมพิวเตอร์ทาง serial port เพื่อทำการแสดงข้อมูลที่วัดได้บนหน้าจอคอมพิวเตอร์เพื่อให้สะดวกต่อการแสดงผลและการถ่ายข้อมูลที่ได้เก็บไว้ในแรมที่อุปกรณ์มาสเตอร์ออกไป

โดยเมื่ออุปกรณ์มาสเตอร์เชื่อมต่อกับ Serial Port กับคอมพิวเตอร์เพื่อจะทำการถ่ายเทข้อมูลที่เก็บได้ออกไปแสดงผลยังหน้าจอคอมพิวเตอร์นั้น โดยโปรแกรมที่เขียนขึ้นมา ซึ่งใช้ภาษา VISUAL C++

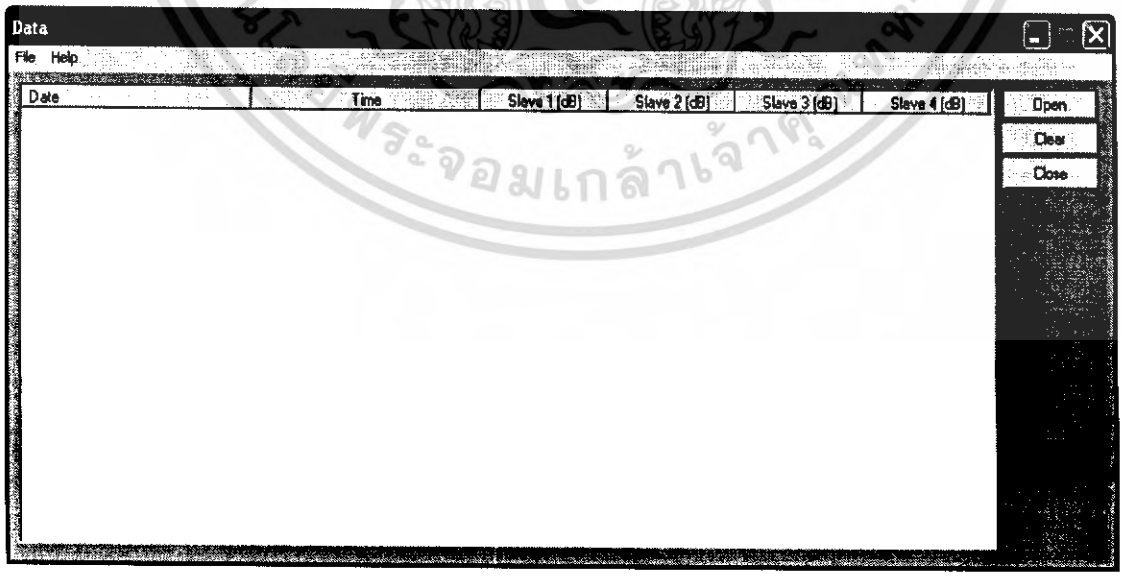
เมื่อเปิดโปรแกรมจะแสดงหน้าต่างดังนี้



รูปที่ 3.26 หน้าต่างโปรแกรมสำหรับการแสดงค่าบนคอมพิวเตอร์

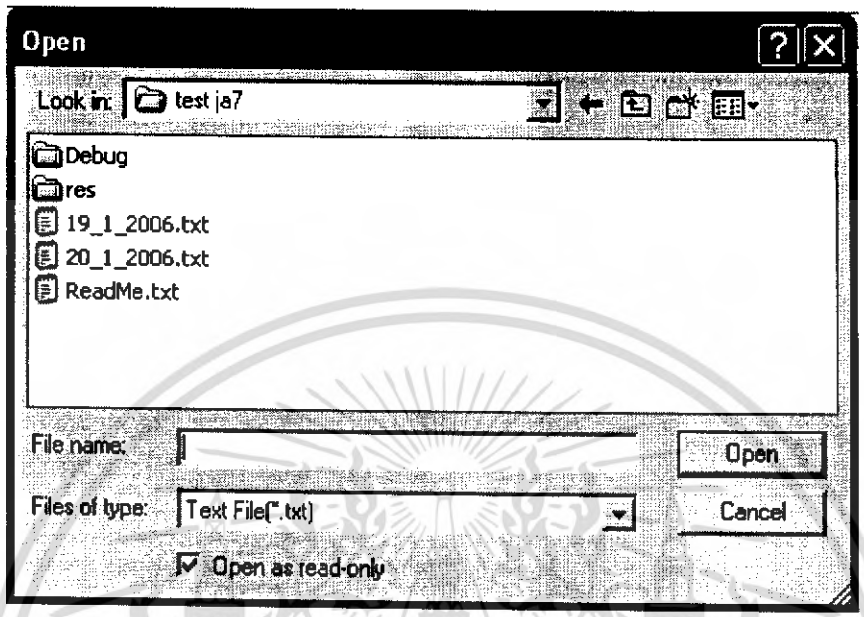
ในรูปเป็นหน้าตาของโปรแกรมเครื่องวัดระดับความดังเสียง ปุ่ม Connect ใช้ในการตรวจสอบสถานะของ Serial port ว่ามีอุปกรณ์เชื่อมต่ออยู่หรือไม่ ปุ่ม Get data ใช้สำหรับการส่งอินเทอร์รัฟไปยังอุปกรณ์ที่เชื่อมต่ออยู่ให้ทำการส่งข้อมูลที่เก็บอยู่ออกมา ปุ่ม History ใช้สำหรับเปิดดูข้อมูลที่ได้ทำการส่งมาแล้วทั้งหมด ปุ่ม Clear ใช้สำหรับเคลียร์ข้อมูลที่แสดงอยู่ที่หน้าต่าง ปุ่ม Exit ใช้สำหรับปิดโปรแกรม

ถ้าต้องการจะดูข้อมูลที่ได้ทำการวัดค่าไว้แล้วให้ทำการกดปุ่ม History จะแสดงหน้าต่างดังรูปที่ 3.27



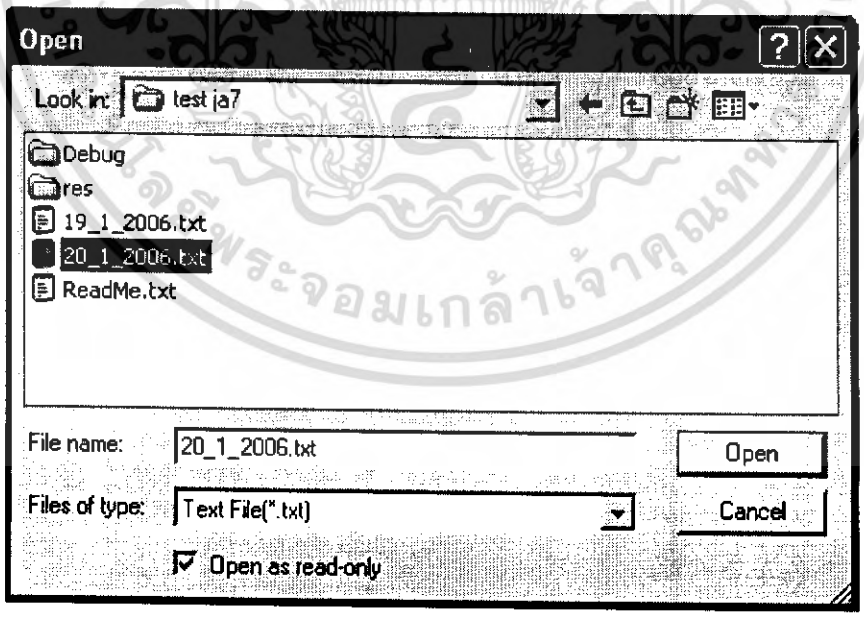
รูปที่ 3.27 หน้าต่างโปรแกรมเมื่อได้ทำการกดปุ่ม History

ถ้าต้องการเปิดดูข้อมูลที่ได้ทำการวัดเก็บไว้แล้ว ก็ให้ทำการกดปุ่ม Open เพื่อหาไฟล์ที่ได้ทำการบันทึกไว้จะแสดงหน้าต่างดังรูปที่ 3.28



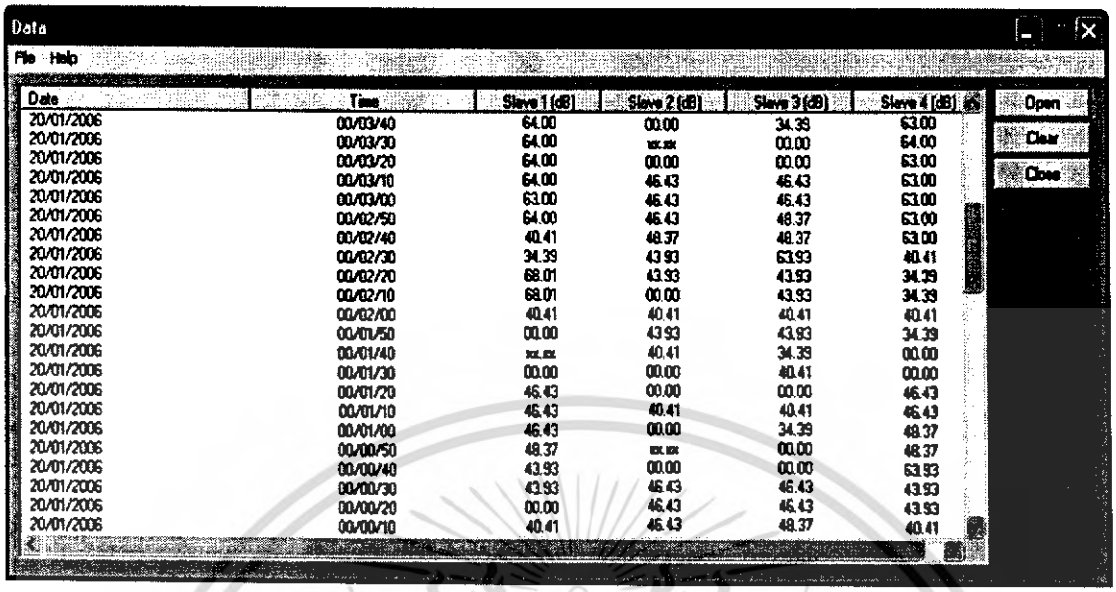
รูปที่ 3.28 แสดงหน้าต่างไฟล์ข้อมูลต่างๆที่ได้ทำการบันทึกเก็บไว้

จากนั้นให้ทำการเลือกไฟล์ที่ต้องการ แล้วกด Open จะได้หน้าต่างขึ้นมาดังรูปที่ 3.29



รูปที่ 3.29 แสดงการเลือกไฟล์ต่างๆที่ต้องการ

หน้าต่างโปรแกรมเมื่อได้ทำการเปิดไฟล์ที่ต้องการจะเป็นดังรูปที่ 3.30

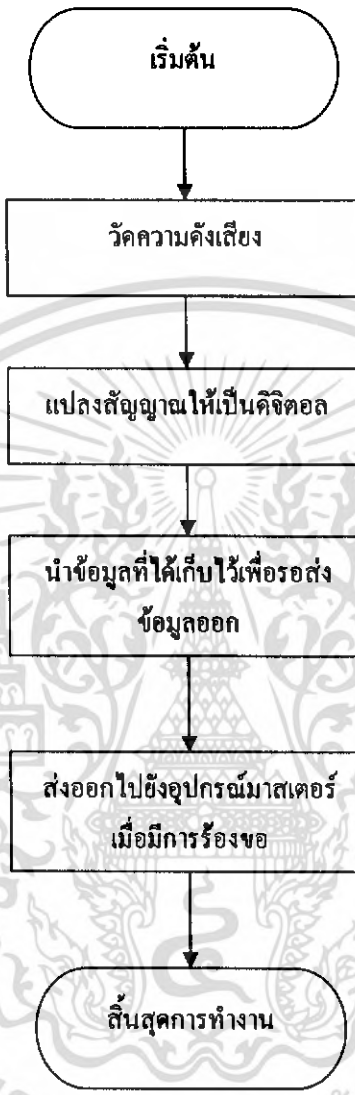


รูปที่ 3.30 แสดงข้อมูลในไฟล์ที่ได้ทำการเก็บบันทึกไว้

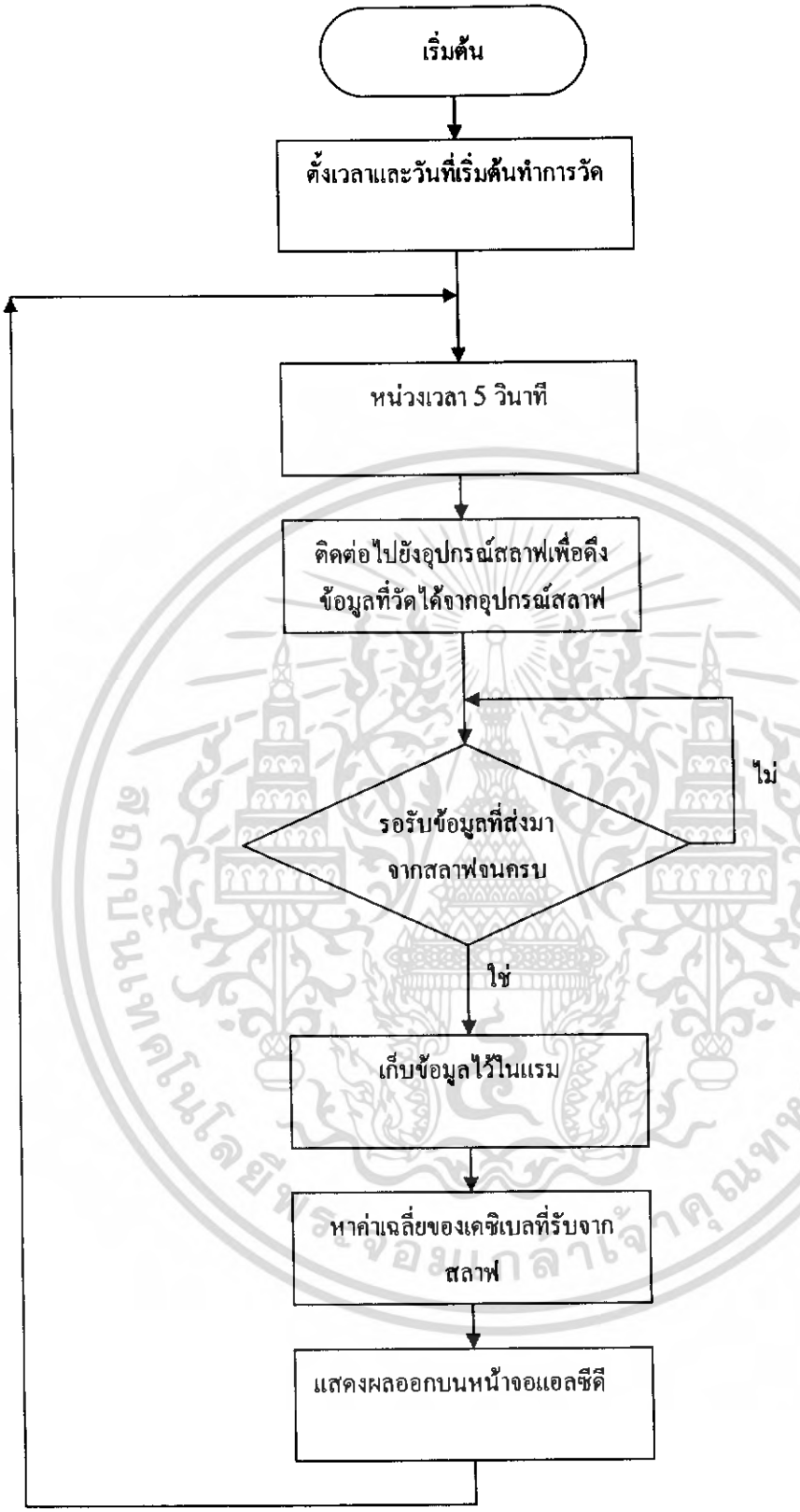
เมื่อทำการกดปุ่ม Exit จะแสดงหน้าจอดังรูปที่ 3.31

รูปที่ 3.31 แสดงหน้าต่างต้องการปิดโปรแกรมเมื่อกดปุ่ม Exit

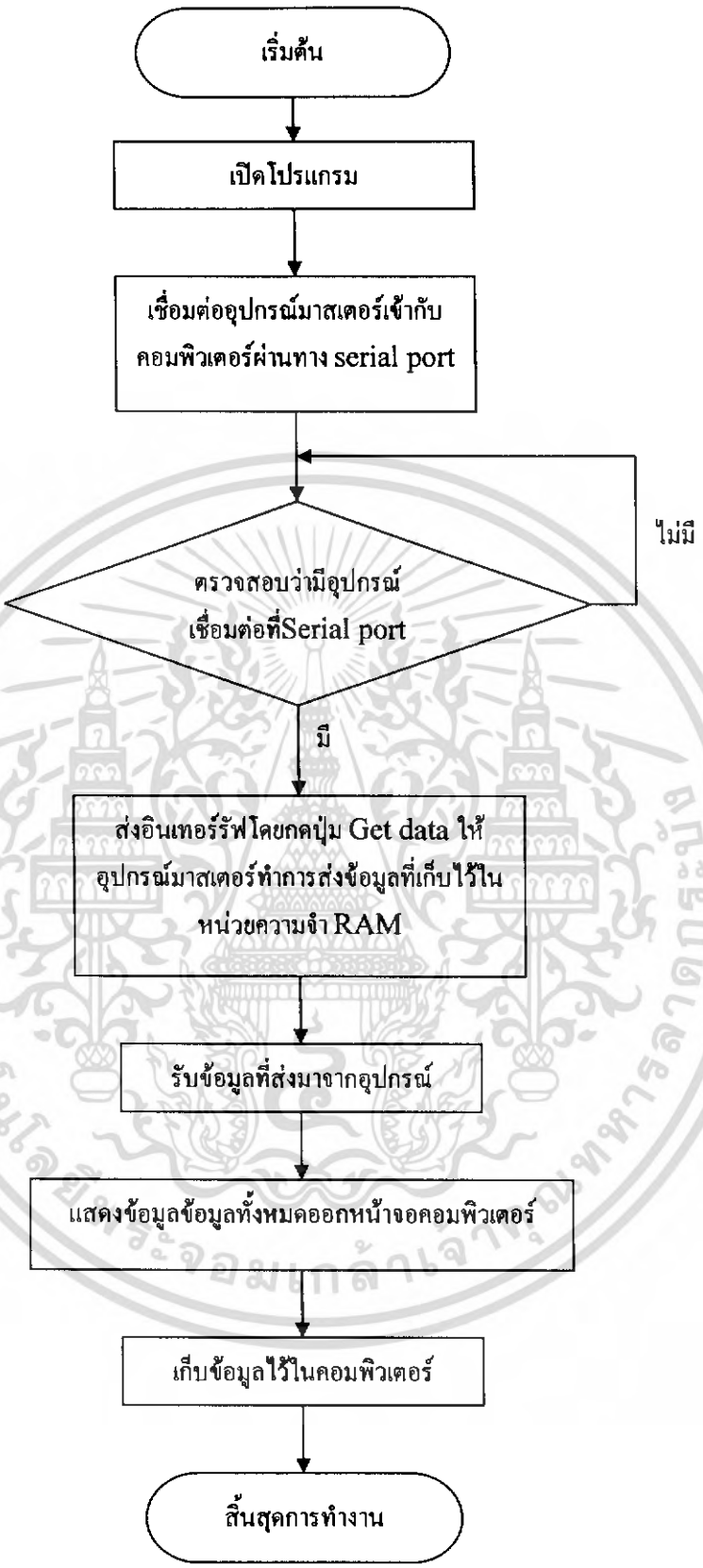
3.5 ส่วนของโฟล์ชาร์ต แสดงการทำงานของระบบ



รูปที่ 3.32 โฟล์ชาร์ตแสดงการทำงานของอุปกรณ์สลาฟ



รูปที่ 3.33 โฟล์ชาร์ตแสดงการทำงานของอุปกรณ์อุปกรณ์มาสเตอร์



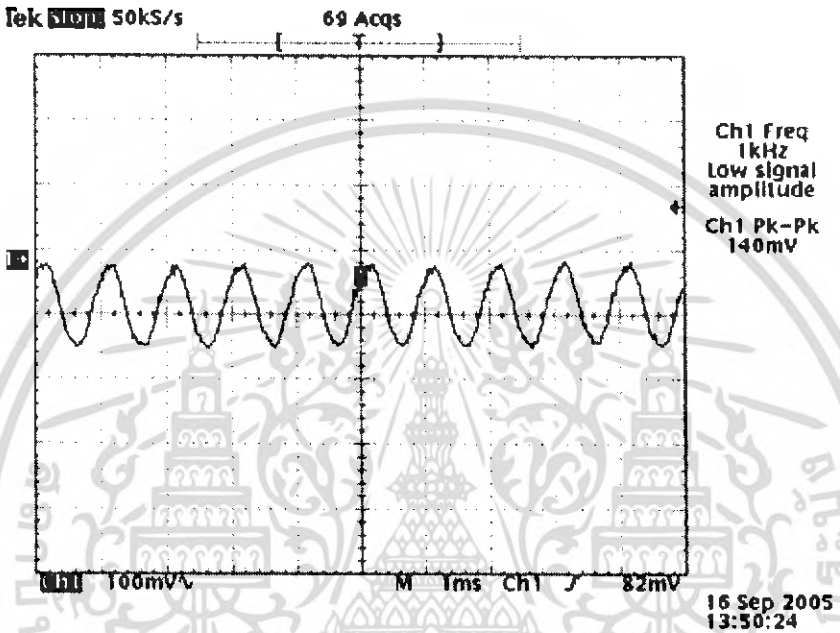
รูปที่ 3.34 โฟล์ชาร์ตการทำงานของโปรแกรมรับค่าจากอุปกรณ์มาสเตอร์

บทที่ 4

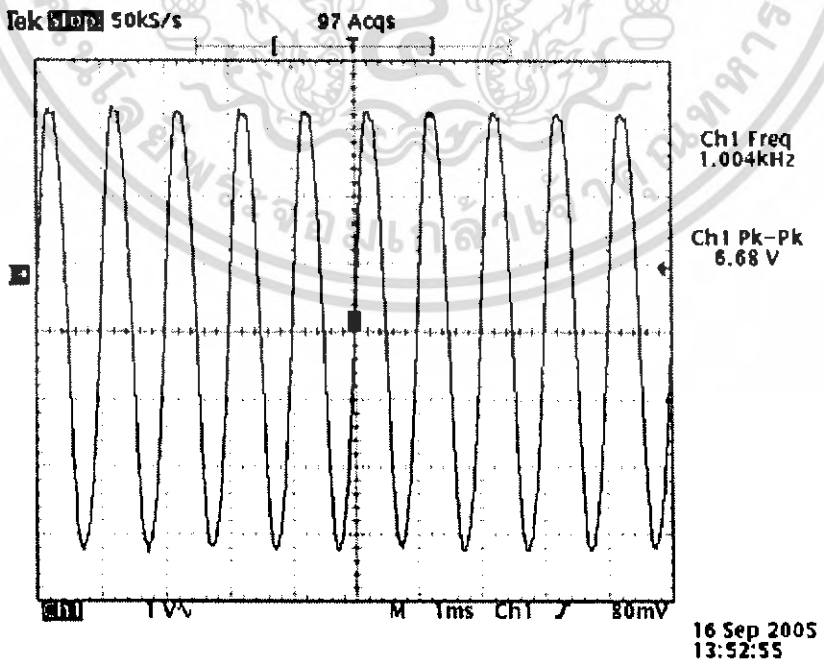
การทดลองและผลการทดลอง

4.1 การทดลองส่วนของวงจรสลาฟ

จากการทดลองวัดอัตราการขยายของวงจรขยายสัญญาณขนาดเล็ก (Small Signal Amplifier) โดยการป้อนสัญญาณไซน์เวฟ (Sinc Wave) ที่ความถี่ 1 kHz แรงดันอินพุท โดยที่เอาท์พุทไม่ถูกตัดยอด (Clip)

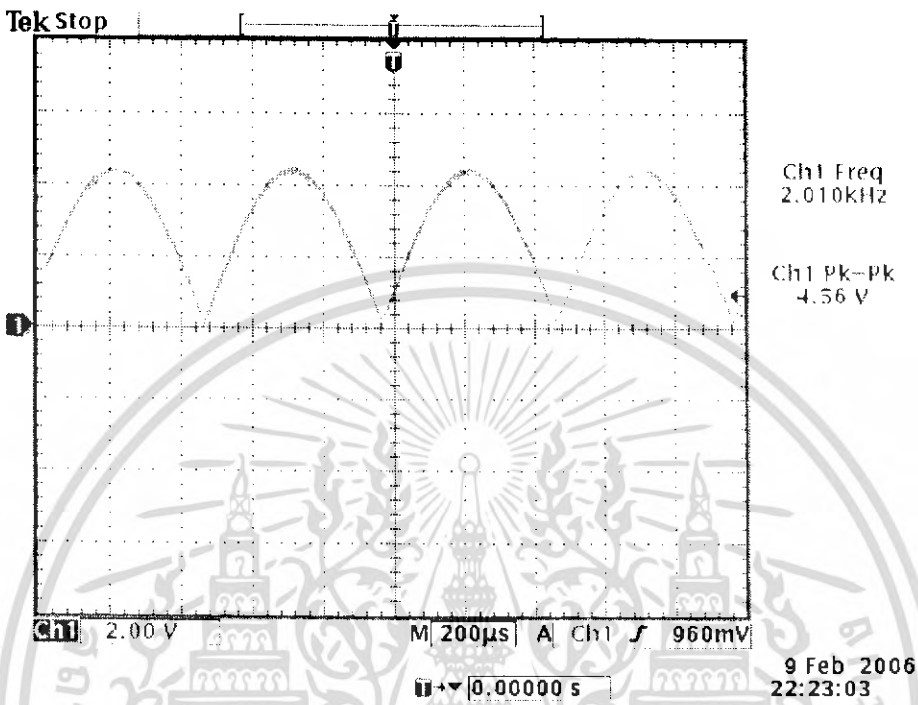


รูปที่ 4.1 สัญญาณที่วัดได้หลังไมโครโฟน



รูปที่ 4.2 เอาท์พุทวงจรขยายสัญญาณขนาดเล็ก (Small Signal Amplifier)

เนื่องจาก IC ADC0820 (A-to-D) จะรับสัญญาณอินพุทขนาดแรงดัน 0 ถึง 5 โวล์ท เพราะฉะนั้น ต้องนำสัญญาณเอาท์พุทที่ได้จากวงจรขยายสัญญาณขนาดเล็ก มาเข้าวงจร Full-wave Rectifier เพื่อให้มีขนาดแรงดันของสัญญาณทางซีกบวกเท่านั้น



รูปที่ 4.3 เอาท์พุทวงจร Full-wave Rectifier

นำสัญญาณเอาท์พุทที่ได้จากวงจร Full-wave Rectifier ซึ่งเป็นสัญญาณอนาล็อกไปทำการแปลงสัญญาณไปเป็นสัญญาณดิจิตอลโดยใช้วงจร Analog to Digital จากนั้นนำสัญญาณดิจิตอลที่ได้ไปเข้าวงจรไมโครคอนโทรลเลอร์และทำการแสดงผลออกทางหน้าจอแอลซีดีต่อไป

4.2 ส่วนของการแคริเบต (Calibrate)

แบ่งเป็นขั้นตอนได้ดังนี้

1. ใช้เครื่อง Sound Level Meter รุ่น YEW - TYPE 3604 SOUND LEVEL METER ในการแคริเบต

รูปที่ 4.4 แสดงเครื่อง Sound Level Meter รุ่น YEW - TYPE 3604 SOUND LEVEL METER

2. เครื่อง Sound Level Meter ที่ใช้เป็นดังรูปที่ 4.3.1 ประกอบด้วยสเกลที่ใช้อ่านค่าเดซิเบลและปุ่มที่ใช้ในการปรับค่าต่างๆ 3 ปุ่ม คือ

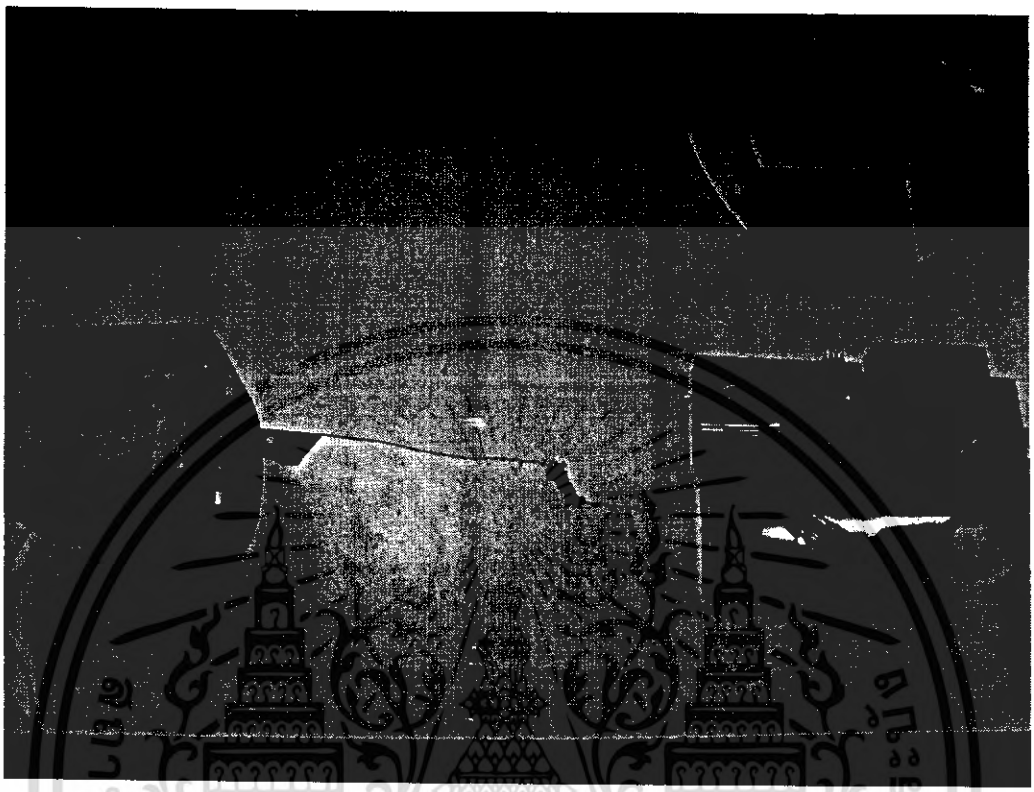
2.1 ปุ่มปรับค่าเดซิเบล ที่ใช้ในการวัดซึ่งมีตั้งแต่ 30 ถึง 120 เดซิเบล

2.2 ปุ่มปรับสเกล ที่ใช้ในการวัดว่าต้องการวัดในหน่วยเดซิเบลใด A , B , C

2.3 ปุ่มที่ใช้ในการอ่านค่าจากเข็ม จะมีให้เลือกอยู่สองโหมดคือ SLOW และ FAST

3. ในการใช้งานเริ่มต้นเราจะหมุนปุ่มไปอยู่ที่ SLOW จากนั้นปรับค่าที่ใช้ในการวัดให้อ่านค่าเป็นเดซิเบล A เพราะเป็นค่าที่ใกล้เคียงกับที่หูได้ยินจริง ต่อจากนั้นปรับค่าเดซิเบลสูงสุดที่ต้องการวัดให้ไปอยู่ที่ 90 เดซิเบล

4. วางลำโพงห่างจากอุปกรณ์สลาฟและเครื่อง Sound Level Meter มาตราฐานเป็นระยะทาง 50 เซนติเมตร



รูปที่ 4.5 ขั้นตอนในการแคริเบต (1)

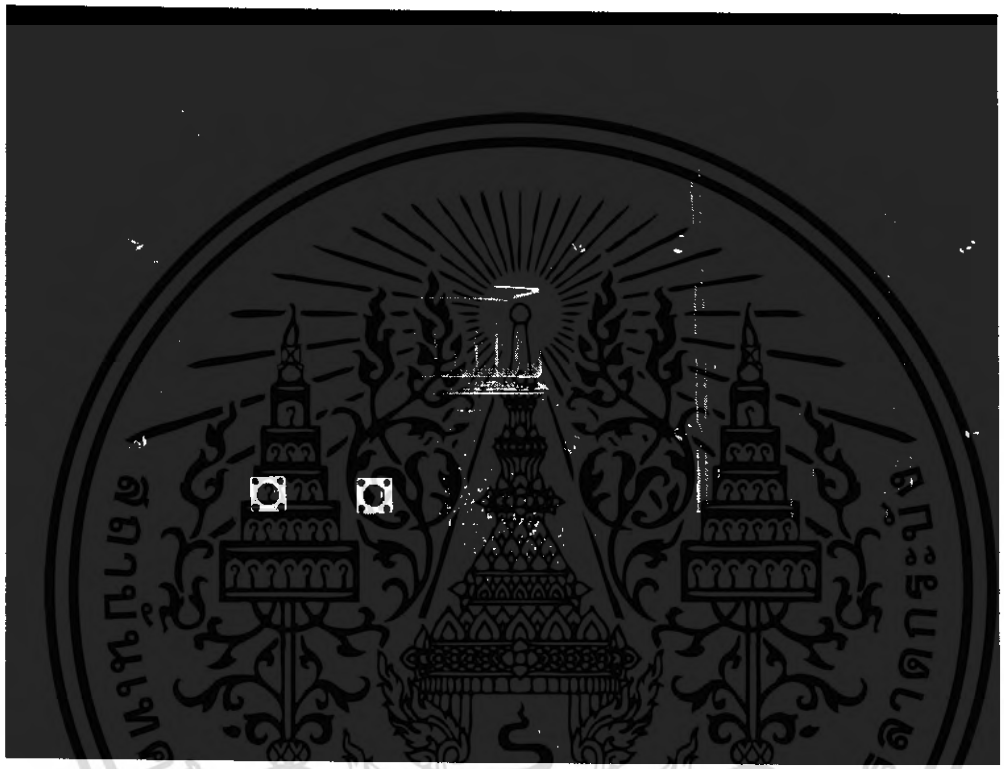
5. ป้อนสัญญาณไซน์เวฟ ความถี่ 1 KHz ให้กับลำโพง
6. ปรับค่าความต้านทานในวงจรสลาฟให้ได้เอาท์พุทไซน์เวฟโดยไม่เกิดความผิดเพี้ยนของสัญญาณ
7. ปรับแอมปลิจูดที่ฟังก์ชั่นเจเนอเรเตอร์ แล้วอ่านค่าที่ได้จากเครื่อง Sound Level Meter โดยกำหนดค่าสูงสุดที่ 90 dB
8. ทำการปรับแอมปลิจูดที่ฟังก์ชั่นเจเนอเรเตอร์ไปเป็นค่าต่างๆแล้วอ่านค่าที่ได้จากเครื่อง Sound Level Meter เทียบกับอ่านค่าโวลล์ที่วัดได้จากเครื่อง สโคบ scope แล้วทำการแปลงค่าโวลล์ที่อ่านได้ให้อยู่ในรูปเดซิเบล

รูปที่ 4.6 ขั้นตอนในการแคริเบต (2)

4.3 การทดลองส่วนของอุปกรณ์มาสเตอร์

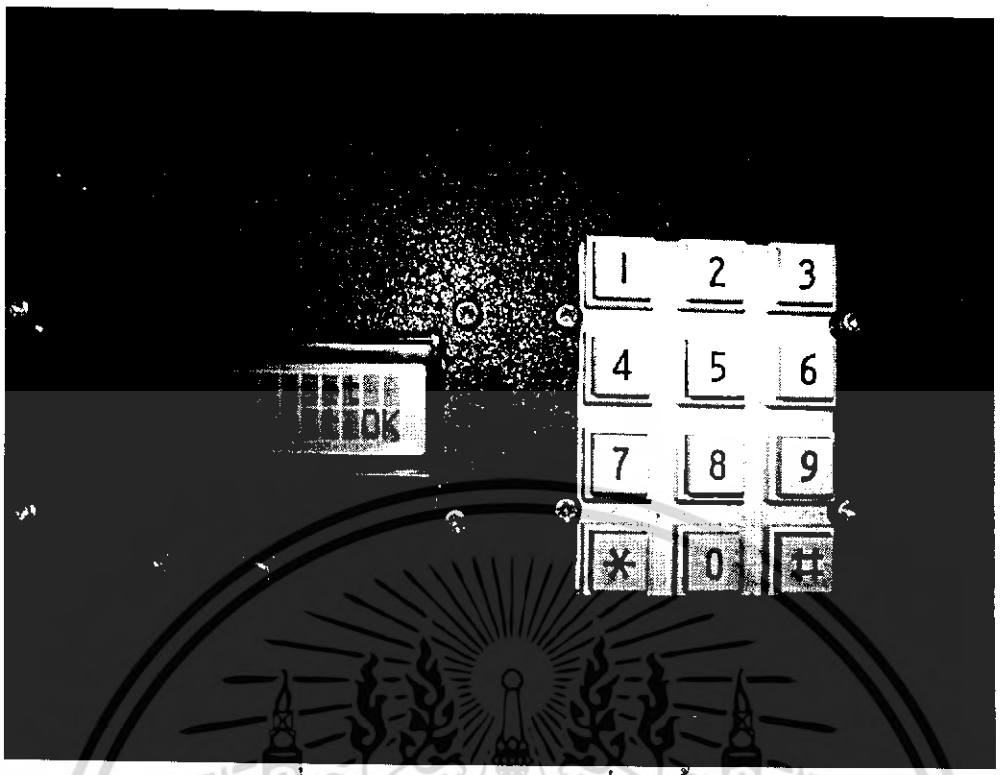
เมื่อส่วนของมาสเตอร์ได้รับสัญญาณที่ส่งมาจากส่วนของสลาฟ ก็จะทำการเก็บระดับความดังเสียงที่วัดได้ไว้ใน RAM และจะแสดงผลออกทางหน้าจอแอลซีดี

เริ่มต้นการทำงานระบบจะทำการแสดงผลที่หน้าจอแอลซีดีเป็น "Welcome To My Project" เมื่อต้องการทำการเก็บข้อมูลก็ให้ทำการกดปุ่ม OK ซึ่งก็คือปุ่ม # ในคีย์แพท ดังแสดงในรูปที่ 4.7



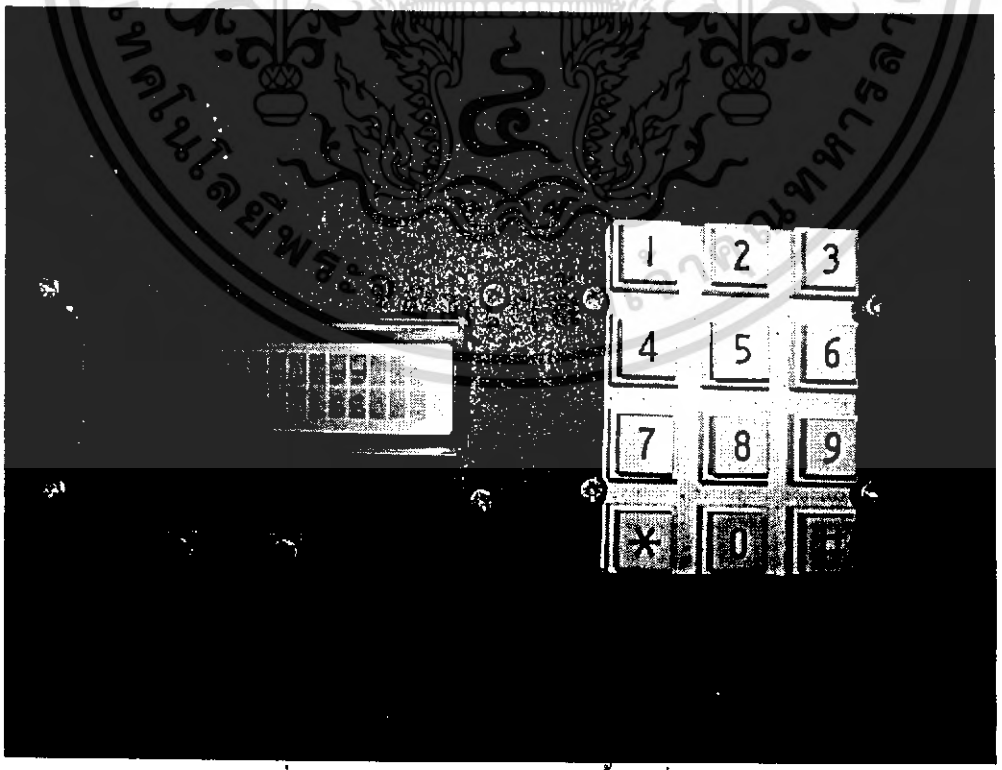
รูปที่ 4.7 การทำงานในโหมดเริ่มการใช้งาน

เมื่อกดปุ่ม OK แล้ว จากนั้นจะต้องทำการตั้งค่าเวลาที่เริ่มต้นการทำงาน โดยที่หน้าจอแอลซีดีจะทำการแสดงผลเป็น "You must set Clock First" ให้กดปุ่ม OK ดังแสดงในรูปที่ 4.8



รูปที่ 4.8 การทำงานในโหมดให้เริ่มการตั้งเวลา

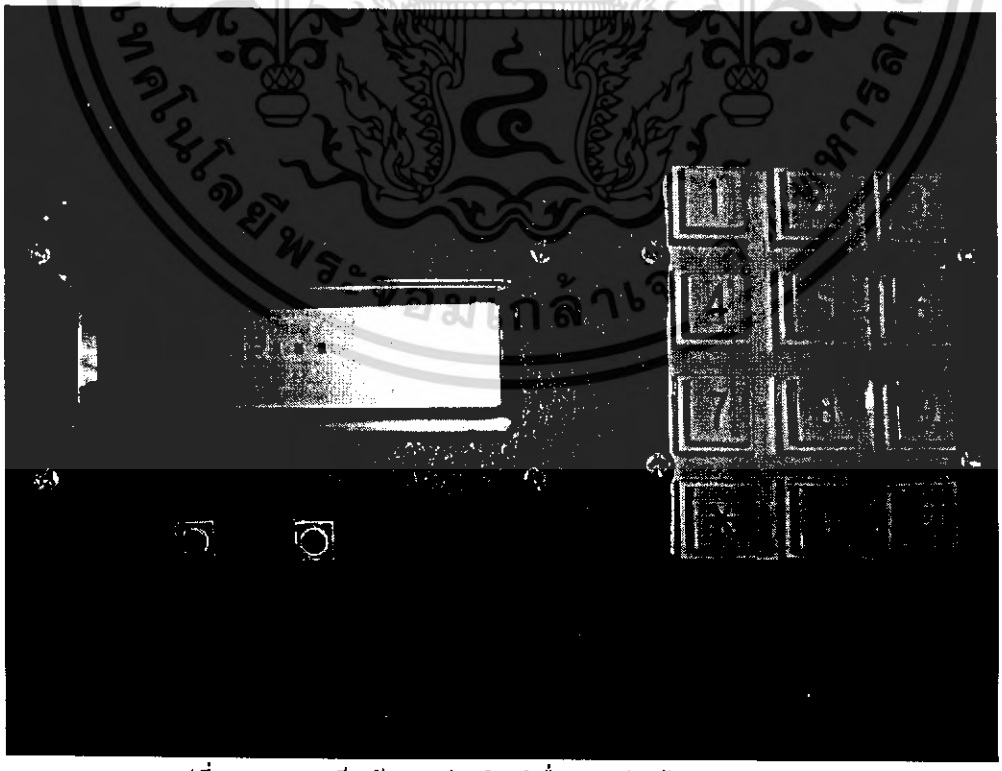
เมื่อกดปุ่ม OK แล้ว จากนั้นจะต้องทำการตั้งค่าเวลาและวันที่ลงไป โดยที่แอลซีดีจะแสดงเป็น "Date dd / mm / yy" และ "Time hh : mm : ss" ดังในรูปที่ 4.9 เมื่อตั้งค่าวันที่และเวลาเสร็จแล้วก็กดปุ่ม OK ดังรูปที่ 4.10



รูปที่ 4.9 การทำงานในโหมดการตั้งวันที่และเวลา

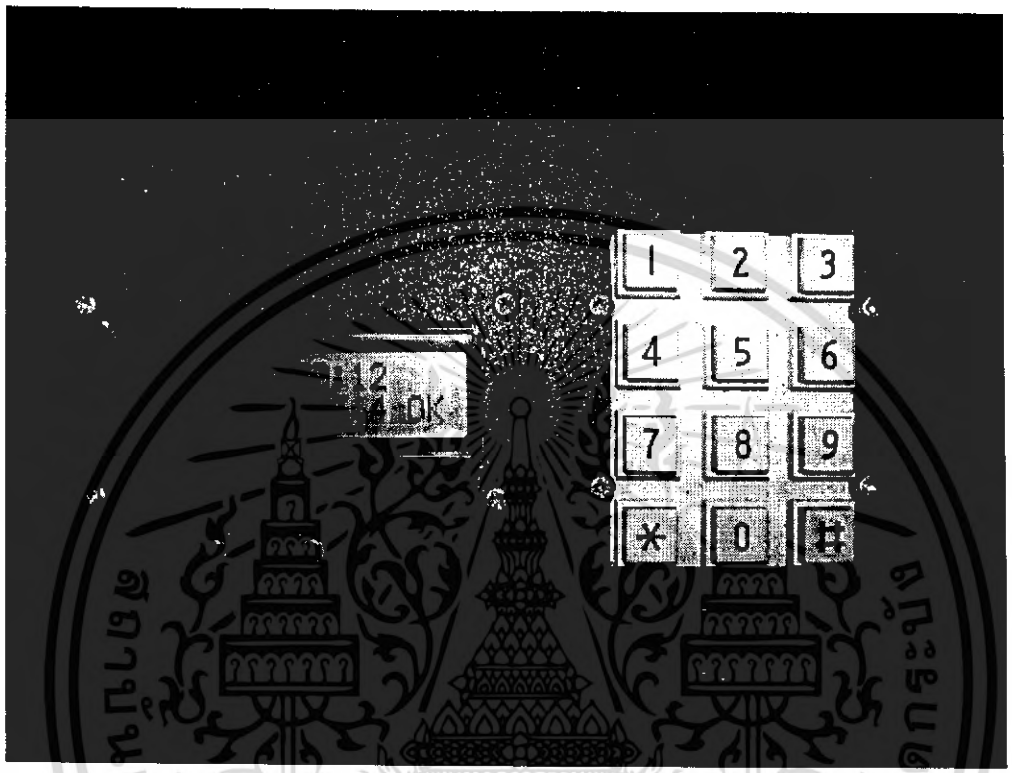
รูปที่ 4.10 การทำงานในโหมดการป้อนค่าการตั้งวันที่และเวลา

จากนั้นระบบก็จะทำการเริ่มต้นเก็บข้อมูลความดังเสียงเมื่อกดปุ่ม OK และเริ่มการเก็บข้อมูลเสียงใหม่เมื่อเวลาผ่านไปทุกๆ 5 วินาที ดังในรูปที่ 4.11 ซึ่งจะแสดงข้อความว่า Working ทุกๆครั้งที่มีการเก็บข้อมูลใหม่



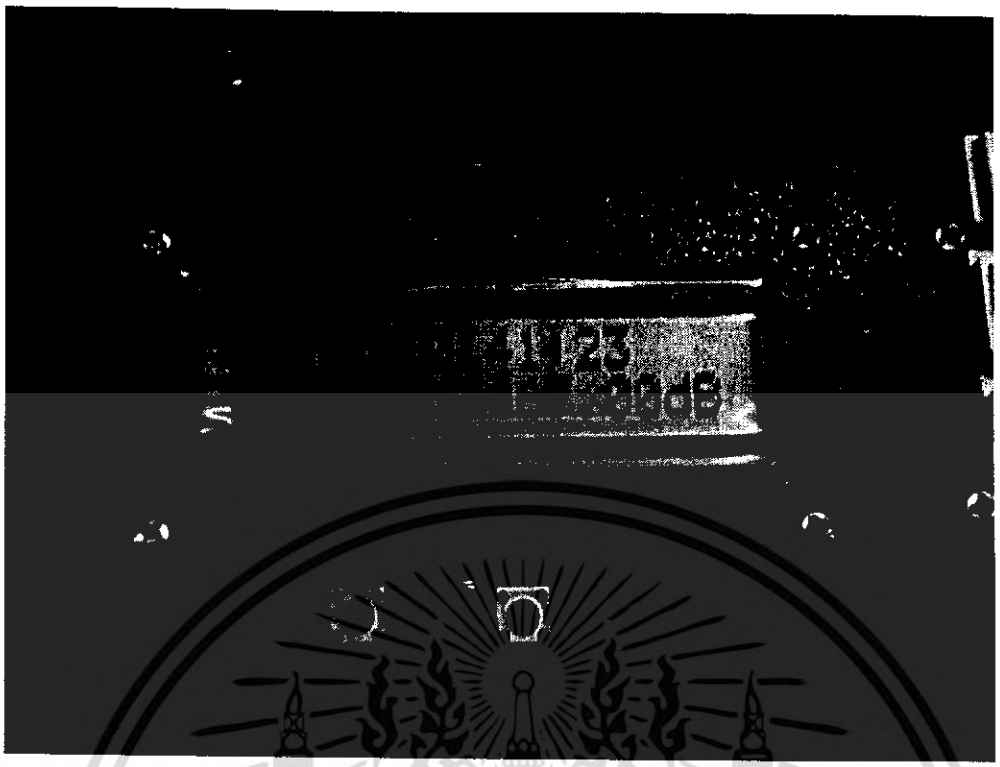
รูปที่ 4.11 การเก็บข้อมูลเสียงใหม่เมื่อเวลาผ่านไปทุกๆ 5 วินาที

เมื่อมีการเริ่มต้นเก็บข้อมูลที่หน้าจอแอลซีดีจะแสดงเวลาและตัวสลาฟทั้งหมดดังในรูปที่ 4.12 ซึ่งถ้าต้องการดูว่าสลาฟตัวใดอ่านค่าได้เท่าไหร่ก็ทำได้โดยการกดปุ่มซ้ายและขวาที่อยู่ข้างล่างจอแอลซีดีดังรูป จะมีลูกศร > ปรากฏหน้าสลาฟตัวนั้นๆ



รูปที่ 4.12 หน้าจอแสดงจำนวนสลาฟทั้งหมด

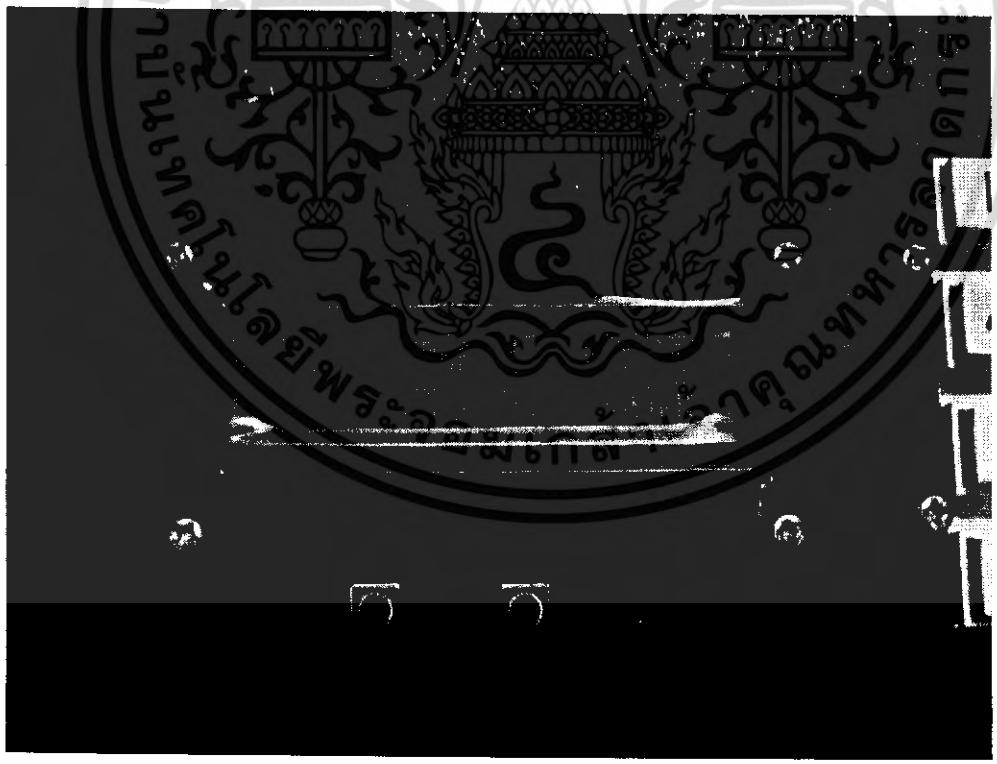
ถ้าต้องการดูข้อมูลที่สลาฟตัวใดก็ให้ทำการกดปุ่มเลื่อนลูกศรซ้ายหรือขวาไปหน้าตัวสลาฟนั้นๆ แล้วทำการกดปุ่ม # ที่คีย์แพด หากต้องการกลับมาหน้าจอเดิมให้ทำการกดปุ่ม *



รูปที่ 4.13 การแสดงผลการเก็บข้อมูลเสียงทางหน้าจอแอลซีดีของตัวสลาฟ 1

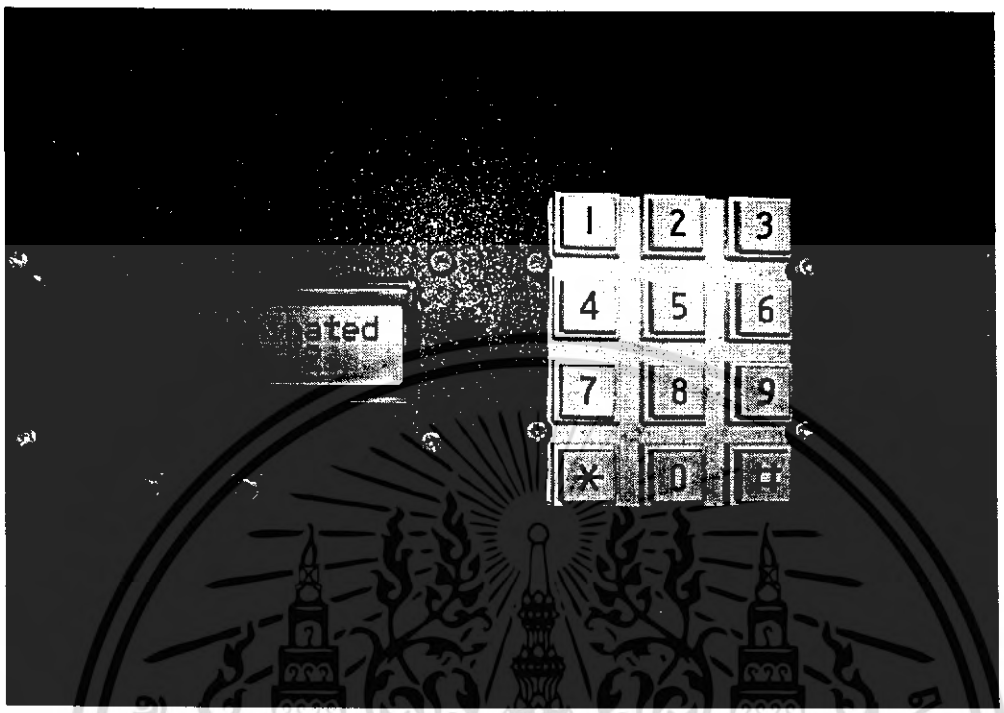
รูปที่ 4.14 การแสดงผลการเก็บข้อมูลเสียงทางหน้าจอแอลซีดีของตัวสลาฟ 2

รูปที่ 4.15 การแสดงผลการเก็บข้อมูลเสียงทางหน้าจอแอลซีดีของตัวสลาฟ 3



รูปที่ 4.16 การแสดงผลการเก็บข้อมูลเสียงทางหน้าจอแอลซีดีของตัวสลาฟ 4

เมื่อกด stop ระบบจะหยุดเก็บข้อมูล และจะแสดงระยะเวลาล่าสุดที่ได้ทำการหยุดการเก็บเก็บข้อมูลดังในรูปที่ 4.17

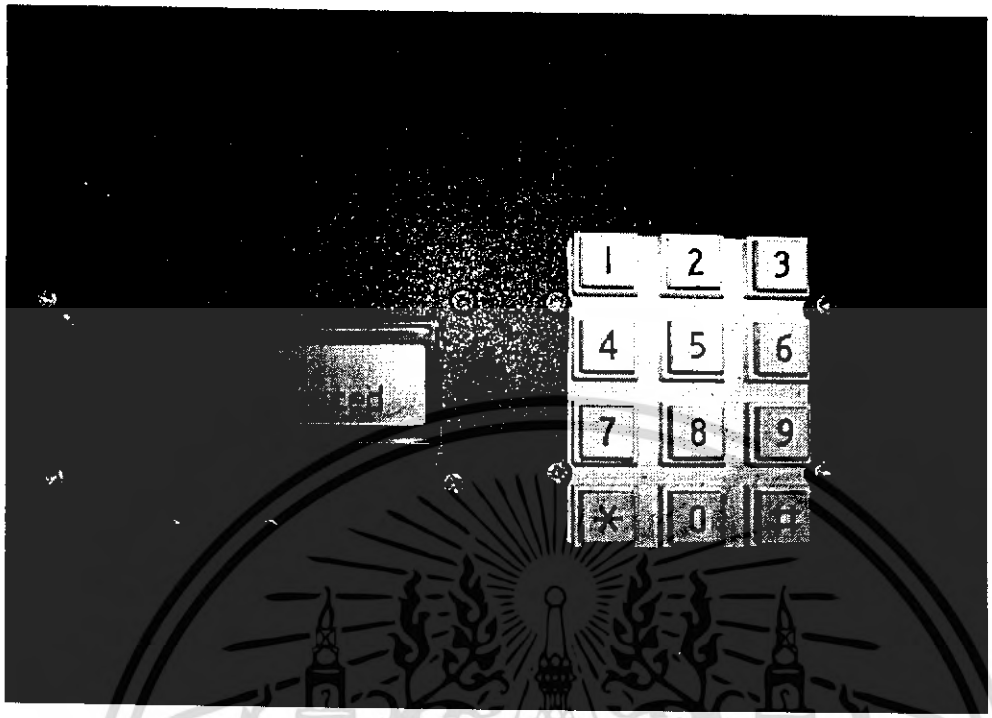


รูปที่ 4.17 แสดงระยะเวลาสิ้นสุดที่ได้ทำการเก็บข้อมูลเมื่อได้ทำการกดปุ่ม stop

และจะแสดงระยะเวลาเริ่มต้นและสิ้นสุดการเก็บข้อมูลนั้นๆดังในรูปที่ 4.18

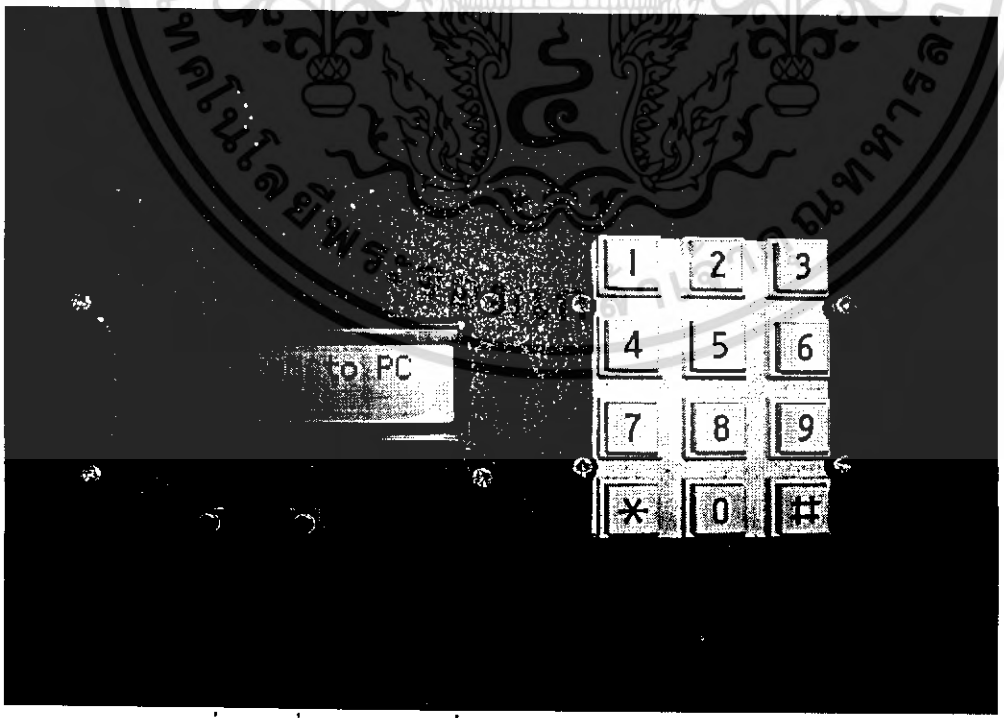
รูปที่ 4.18 แสดงระยะเวลาเริ่มต้นและสิ้นสุดการเก็บข้อมูล

จากนั้นระบบจะทำการแจ้งให้ทราบว่าได้ทำการเก็บข้อมูลไปทั้งหมดกี่ตัว



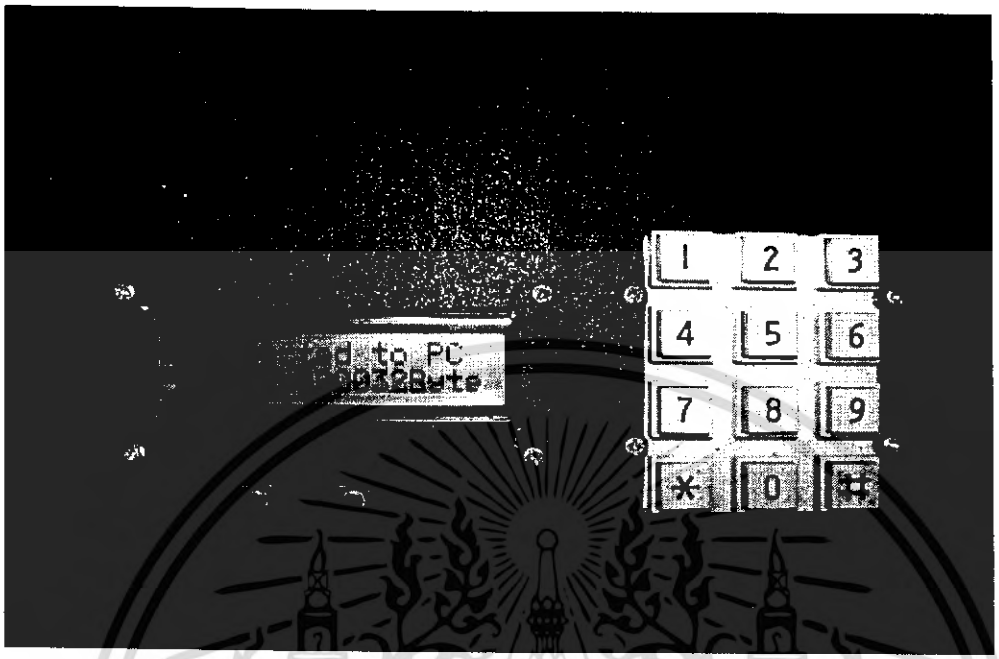
รูปที่ 4.19 แสดงจำนวนข้อมูลทั้งหมดที่ได้ทำการเก็บไว้

เมื่อมีการเชื่อมต่อกับ serial port กับคอมพิวเตอร์ ระบบจะแสดงทางหน้าจอแอลซีดี Connected tp PC ดังในรูปที่ 4.20



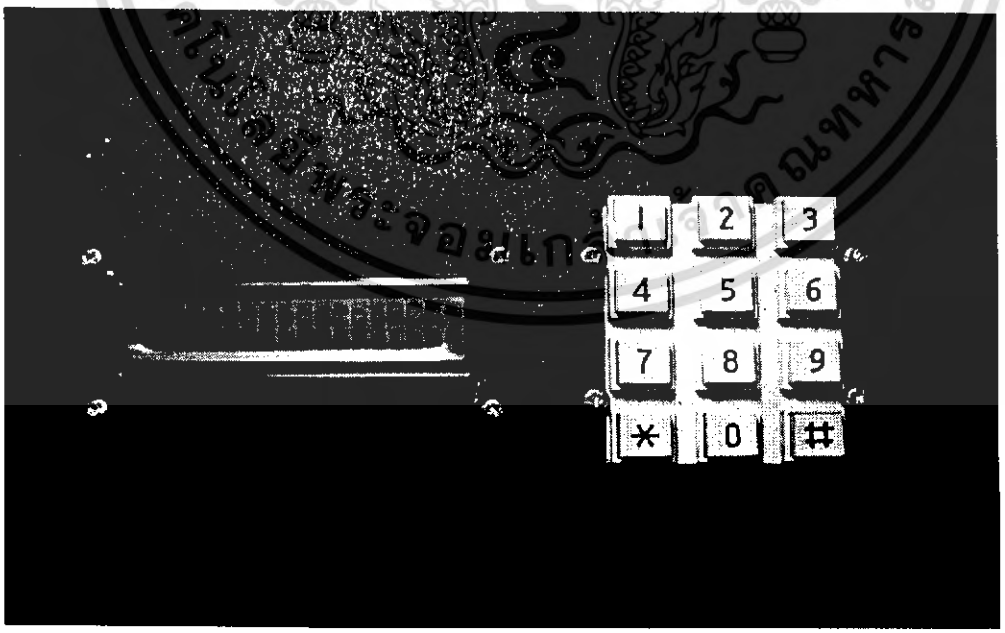
รูปที่ 4.20 เมื่อระบบมีการเชื่อมต่อกับพอร์ตอนุกรมคอมพิวเตอร์

เมื่อมีการส่งข้อมูลจากอุปกรณ์มาสเตอร์ไปยังโปรแกรมในคอมพิวเตอร์ โดยจะแสดงจำนวนไบท์ข้อมูลทั้งหมดที่ทำการส่งออก จะมีการแสดงข้อความดังในรูปที่ 4.21



รูปที่ 4.21 เมื่อมีการส่งข้อมูลจากอุปกรณ์มาสเตอร์ไปยังโปรแกรมในคอมพิวเตอร์

เมื่ออุปกรณ์มาสเตอร์ส่งข้อมูลไปยังคอมพิวเตอร์ผ่านทาง serial port เสร็จสิ้น จะแสดงที่หน้าจอแอลซีดีดังรูป 4.22

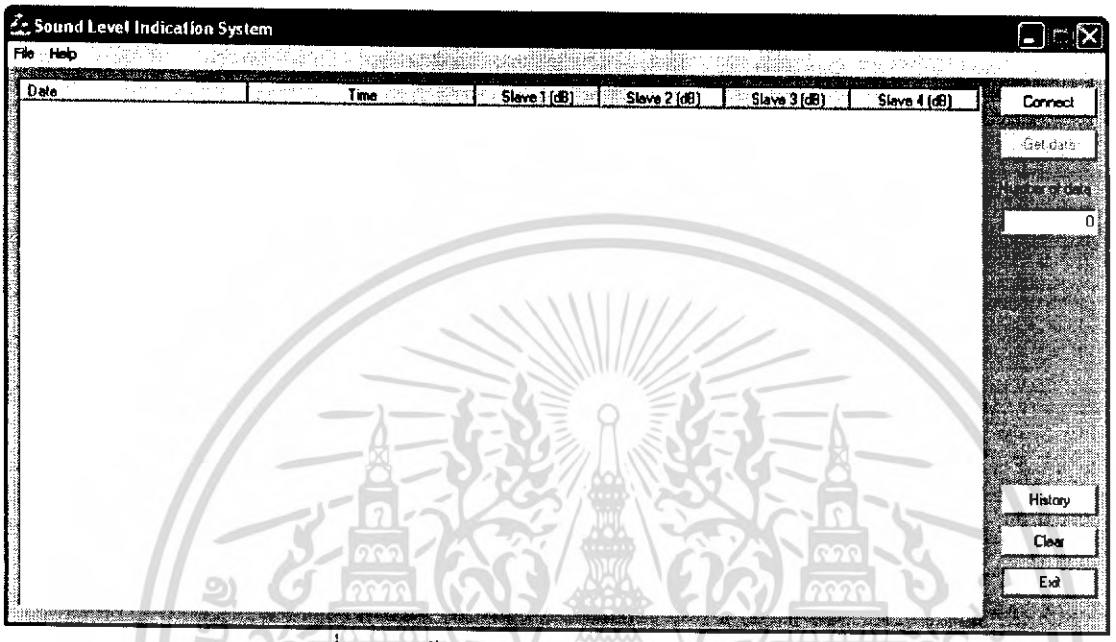


รูปที่ 4.22 เมื่ออุปกรณ์มาสเตอร์ส่งข้อมูลไปยังคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมเสร็จสิ้น

4.3 การใช้โปรแกรม Visual C ++ รับค่าทางพอร์ตอนุกรม

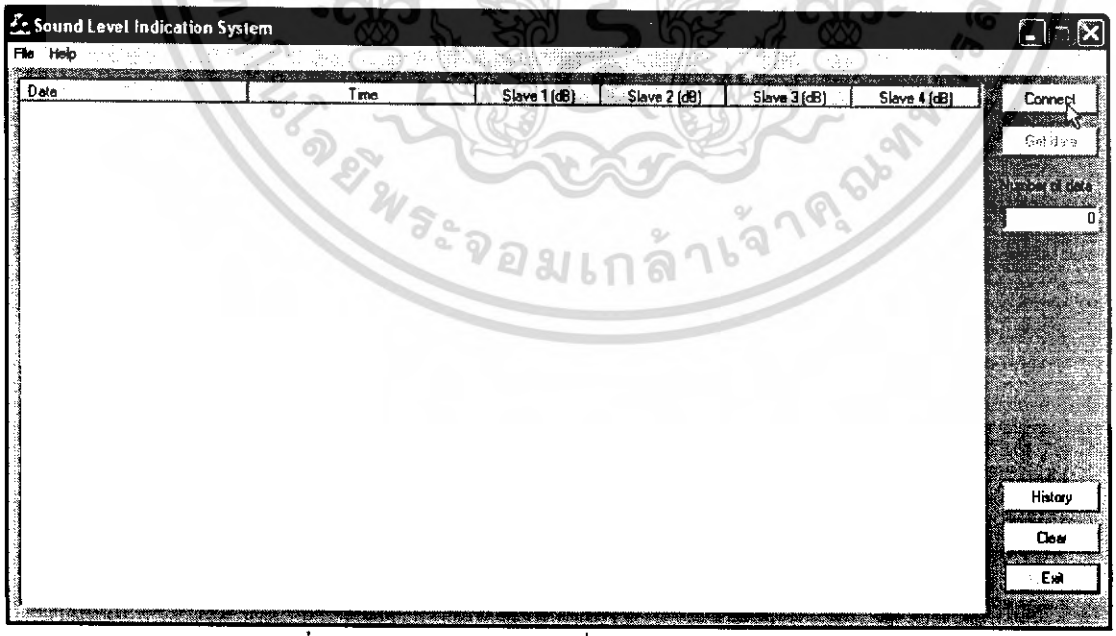
เมื่ออุปกรณ์มาสเตอร์ทำการเชื่อมต่อกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมจะมีขั้นตอนการทำงานดังนี้

1. เมื่อเปิดโปรแกรมจะแสดงหน้าตาดังรูปที่ 4.23



รูปที่ 4.23 หน้าตาโปรแกรมรับค่าทางพอร์ตอนุกรม

2. ทำการเชื่อมต่อกับอุปกรณ์มาสเตอร์ผ่านทางพอร์ตอนุกรมโดยการคลิกปุ่ม Connect

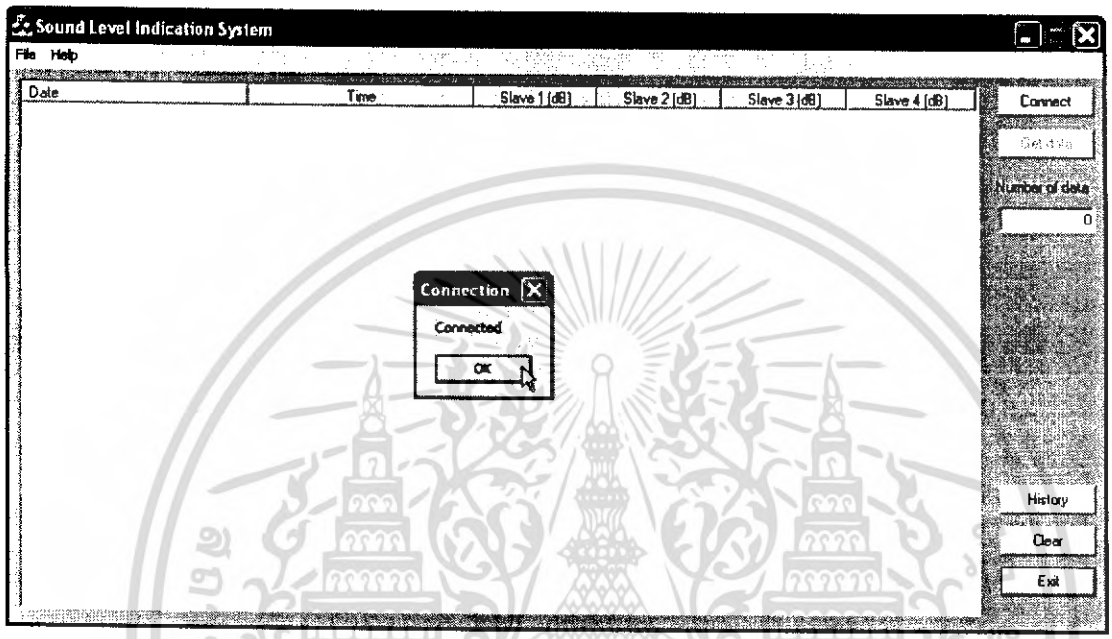


รูปที่ 4.24 หน้าต่างโปรแกรมเมื่อทำการคลิกปุ่ม Connect

3. เมื่อคลิกปุ่ม Connect โปรแกรมจะทำการตรวจสอบที่ port com1 ว่ามีการอุปกรณ์เชื่อมต่อกับคอมพิวเตอร์ทางพอร์ตอนุกรมหรือไม่

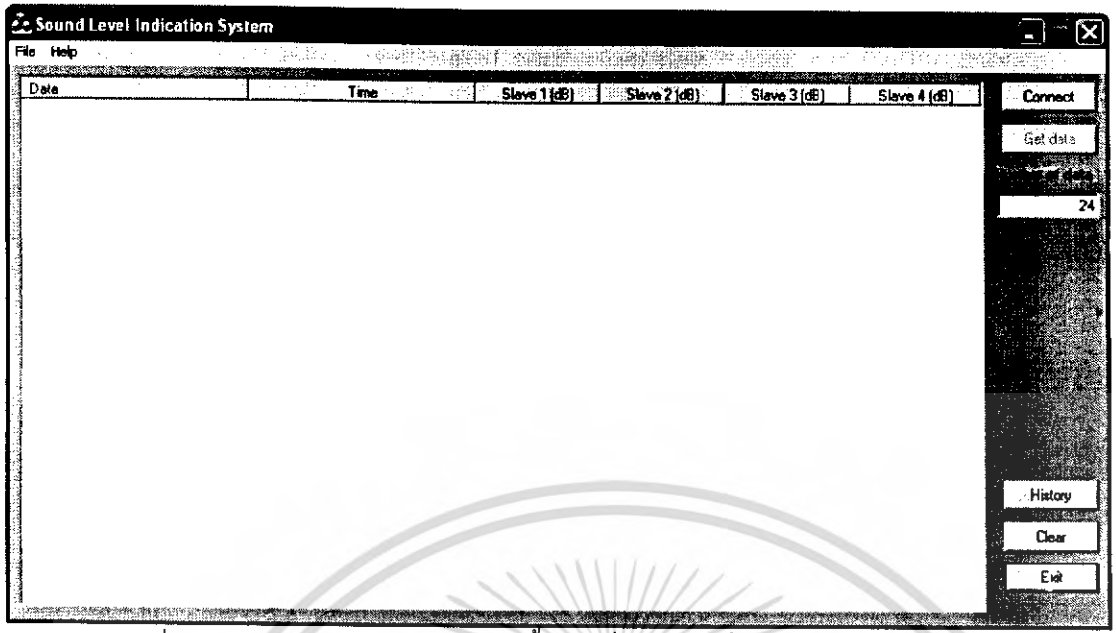
- ถ้าไม่มี โปรแกรมจะแสดงข้อความ error ให้ทำการเชื่อมต่ออุปกรณ์กับพอร์ตอนุกรม

- ถ้ามี โปรแกรมจะมีการสอบถามว่าต้องการเชื่อมต่อกับอุปกรณ์นั้นรึเปล่าดังในรูปที่ 4.25 ถ้าต้องการเชื่อมต่อกับอุปกรณ์นั้นให้ทำการคลิกปุ่ม OK



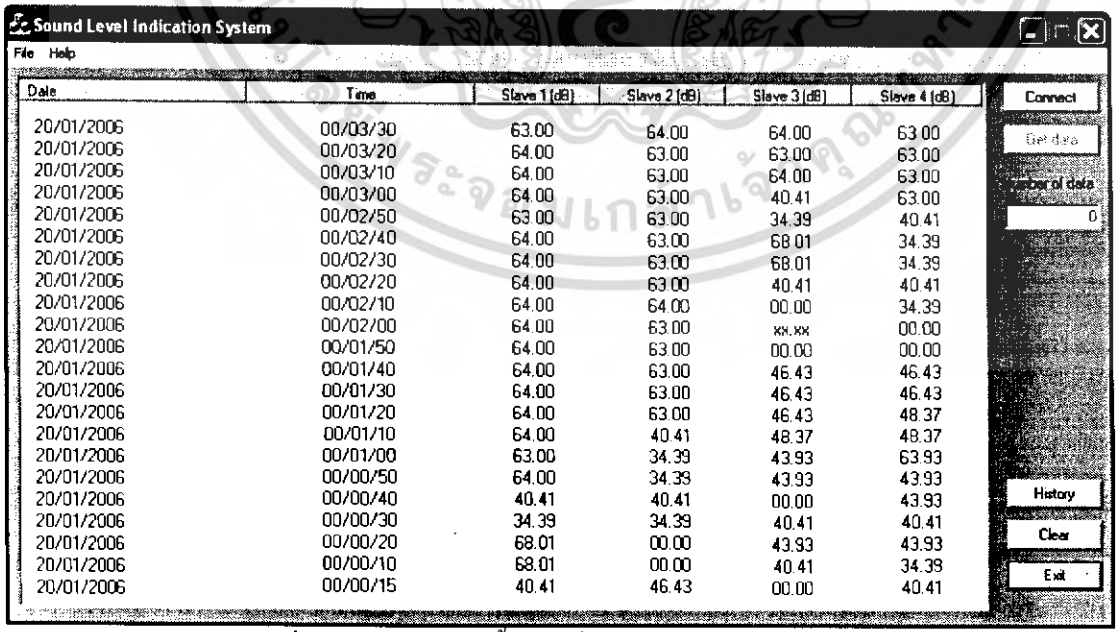
รูปที่ 4.25 แสดงการยืนยันการเชื่อมต่อกับอุปกรณ์เมื่อมีอุปกรณ์เชื่อมต่อที่พอร์ตอนุกรม

4. เมื่อคลิกปุ่ม Ok โปรแกรมจะทำการเชื่อมต่อกับอุปกรณ์ ถ้าต้องการให้อุปกรณ์ที่เชื่อมต่ออยู่ทำการส่งข้อมูลที่เก็บอยู่ออกมาแสดงผลทางหน้าจอคอมพิวเตอร์ให้คลิกที่ปุ่ม Get data ระบบจะทำการแสดงข้อมูลทั้งหมดที่ส่งเข้ามาในช่อง Number of data



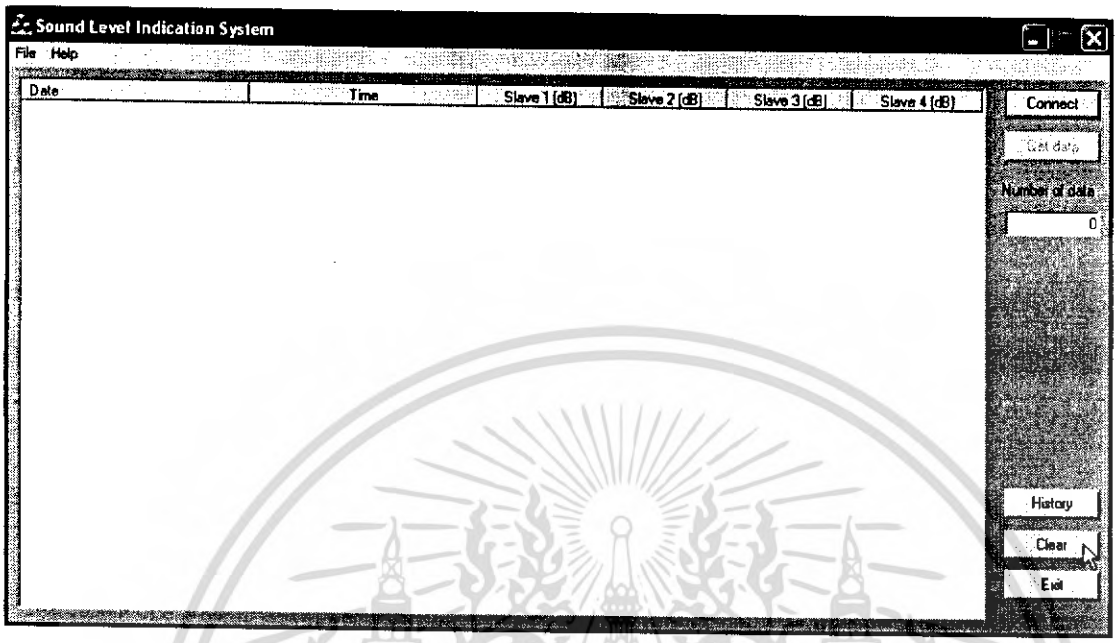
รูปที่ 4.26 หน้าต่างแสดงจำนวนข้อมูลทั้งหมดที่รับเข้ามาเมื่อทำการคลิกปุ่ม Get data

5. จากนั้นก็รอเพื่อให้ระบบทำการโหลดข้อมูลจากอุปกรณ์ที่เชื่อมต่อมาแสดงบนหน้าจอคอมพิวเตอร์ โดยข้อมูลที่แสดงนั้นจะเป็นข้อมูลที่ได้ทำการวัดค่าและเก็บไว้ในตัวอุปกรณ์ที่เชื่อมต่อ ข้อมูลที่แสดงออกมาทางหน้าต่างของโปรแกรมจะเป็น วันที่ที่ได้ทำการเก็บข้อมูลจะแสดงในช่อง Date , ระยะเวลาที่เริ่มทำการเก็บข้อมูลโดยจะทำการเก็บทุกๆ 5 วินาทีจะแสดงในช่อง Time และระดับความดังเสียงที่ได้ทำการวัดเป็นหน่วยเดซิเบลจะแสดงในช่อง dB ดังในรูปที่ 4.27 เมื่อรับข้อมูลเข้ามาโปรแกรมจะทำการเก็บข้อมูลไว้ในคอมพิวเตอร์โดยอัตโนมัติและสามารถเปิดดูข้อมูลเก่าๆได้



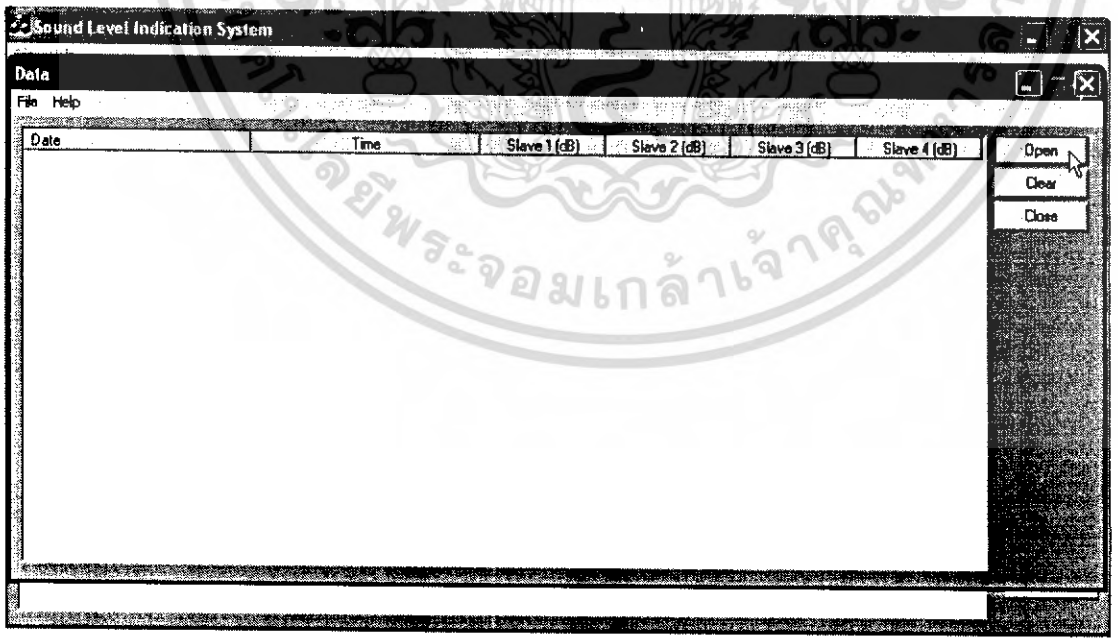
รูปที่ 4.27 แสดงข้อมูลทั้งหมดที่ได้รับจากอุปกรณ์มาสเตอร์

6. จากรูปที่ 4.27 เมื่อทำการคลิกที่ปุ่ม Clear โปรแกรมจะทำการเคลียร์หน้าจอให้ว่างเพื่อจะรับข้อมูลเข้ามาใหม่ จะเป็นดังรูปที่ 4.28



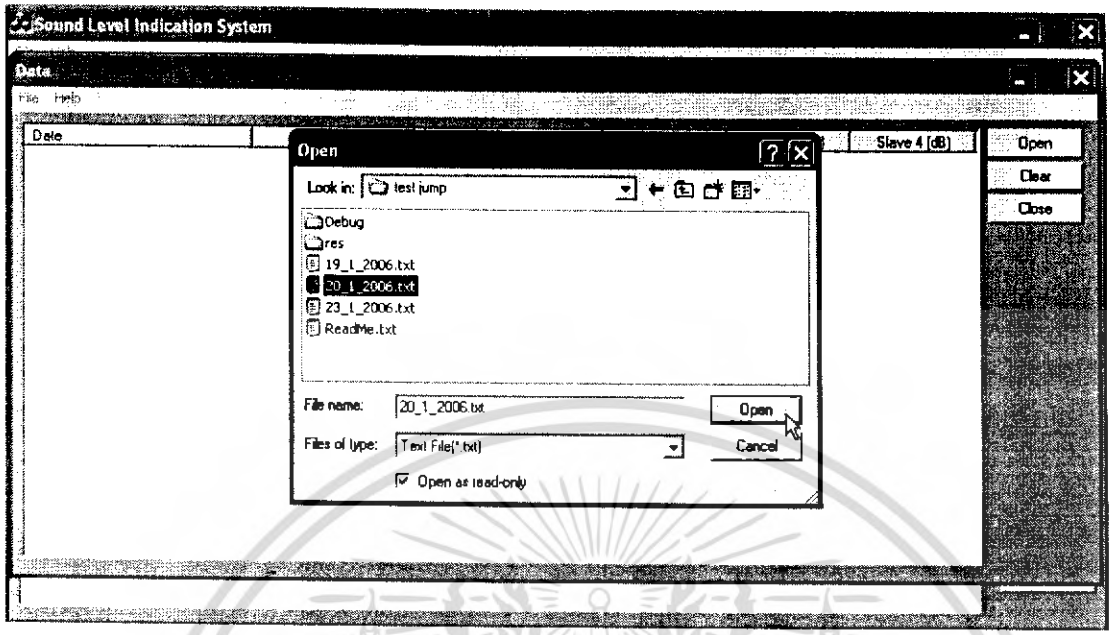
รูปที่ 4.28 แสดงหน้าต่างเมื่อทำการคลิกปุ่ม Clear

7. จากรูปที่ 4.28 ถ้าต้องการดูข้อมูลที่ได้ทำการเก็บไว้ทั้งหมดสามารถเข้าไปดูได้โดยการคลิกที่ปุ่ม History โดยเมื่อคลิกที่ปุ่ม History โปรแกรมจะแสดงหน้าต่างเป็นดังรูปที่ 4.29



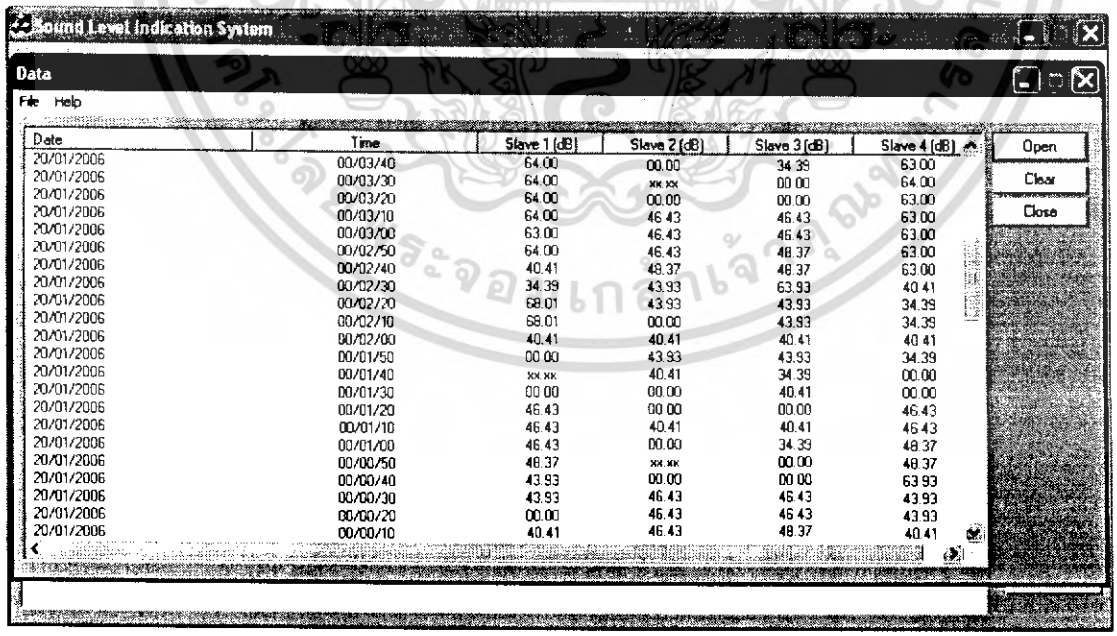
รูปที่ 4.29 หน้าต่างโปรแกรมเมื่อทำการคลิกปุ่ม History

8. จากนั้นให้ทำการคลิกที่ปุ่ม Open เพื่อเลือกหาข้อมูลที่ต้องการดูดังในรูปที่ 4.30



รูปที่ 4.30 หน้าต่างแสดงไฟล์ข้อมูลเก่าที่โปรแกรมได้เก็บไว้

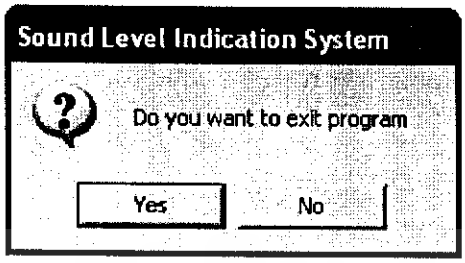
9. เมื่อได้ข้อมูลที่ต้องการให้คลิกที่ปุ่ม Open เพื่อทำการแสดงข้อมูลระดับความดังเสียงที่ได้เก็บไว้ ดังในรูปที่ 4.31



Date	Time	Slave 1 (dB)	Slave 2 (dB)	Slave 3 (dB)	Slave 4 (dB)
20/01/2006	00/03/40	64.00	00.00	34.39	63.00
20/01/2006	00/03/30	64.00	xx.xx	00.00	64.00
20/01/2006	00/03/20	64.00	00.00	00.00	63.00
20/01/2006	00/03/10	64.00	46.43	46.43	63.00
20/01/2006	00/03/00	63.00	46.43	46.43	63.00
20/01/2006	00/02/50	64.00	46.43	48.37	63.00
20/01/2006	00/02/40	40.41	48.37	48.37	63.00
20/01/2006	00/02/30	34.39	43.93	63.93	40.41
20/01/2006	00/02/20	68.01	43.93	43.93	34.39
20/01/2006	00/02/10	68.01	00.00	43.93	34.39
20/01/2006	00/02/00	40.41	40.41	40.41	40.41
20/01/2006	00/01/50	00.00	43.93	43.93	34.39
20/01/2006	00/01/40	xx.xx	40.41	34.39	00.00
20/01/2006	00/01/30	00.00	00.00	40.41	00.00
20/01/2006	00/01/20	46.43	00.00	00.00	46.43
20/01/2006	00/01/10	46.43	40.41	40.41	46.43
20/01/2006	00/01/00	46.43	00.00	34.39	48.37
20/01/2006	00/00/50	48.37	xx.xx	00.00	48.37
20/01/2006	00/00/40	43.93	00.00	00.00	63.93
20/01/2006	00/00/30	43.93	46.43	46.43	43.93
20/01/2006	00/00/20	00.00	46.43	46.43	43.93
20/01/2006	00/00/10	40.41	46.43	48.37	40.41

รูปที่ 4.31 หน้าต่างแสดงการเปิดไฟล์ข้อมูลเก่าที่ได้บันทึกไว้ออกมาดู

10. เมื่อต้องการออกจากโปรแกรมให้ทำการคลิกที่ปุ่ม Exit โปรแกรมก็จะถามว่าต้องการออกจากโปรแกรมรึเปล่า ถ้าต้องการให้ทำการคลิกปุ่ม Yes ถ้าไม่ต้องการให้คลิกปุ่ม No



รูปที่ 4.32 หน้าต่างแสดงการยืนยันการออกจากโปรแกรม

บทที่ 5
วิจารณ์และสรุปการทดลอง

เนื่องจากในโครงงานเครื่องวัดระดับความดังเสียงนี้ได้มีการแบ่งส่วนการทำงานเป็นหลายๆส่วน ดังนั้นจึงได้วิจารณ์และสรุปผลแยกออกเป็นส่วนๆได้ดังนี้

1. อุปกรณ์สลาฟ

1.1 ส่วนของวงจรไมโครโฟน

วงจรส่วนนี้เป็นวงจรส่วนหน้าที่อยู่ในอุปกรณ์สลาฟมีหน้าที่แปลงความดังเสียง(แรงดันอากาศ) ไปเป็นสัญญาณทางไฟฟ้าเพื่อป้อนให้กับวงจร A/D

จากการทำงานนั้นพบว่าประสิทธิภาพในการทำงานยังไม่ดีเท่าที่ควรซึ่งพอจะสรุปปัญหาได้ดังนี้

- ไมโครโฟนวัดสัญญาณเสียงที่ดังมากๆ ไม่ได้โครงงานนี้จึงจำกัดการวัดสูงสุดไว้ที่ 90dB
- ไมโครโฟนมีค่า sensitivity ต่ำนั่นคือค่าความดังที่เครื่องวัดได้จะมีการเปลี่ยนแปลงเพียงเล็กน้อยเมื่อสัญญาณอินพุทเข้ามา

จากปัญหาข้างต้นจะเห็นว่าในการที่จะเพิ่มประสิทธิภาพของวงจรส่วนนี้ประเด็นหลักอยู่ที่ควรไมโครโฟนที่มีค่า sensitivity สูงและสามารถวัดสัญญาณเสียงที่มีความดังมากๆได้

1.2 วงจรA/D

วงจรนี้มีหน้าที่แปลงโวลเตจจากวงจรไมโครโฟนไปเป็นข้อมูลดิจิตอลเพื่อส่งไปให้กับอุปกรณ์มาสเตอร์

จากการทำงานพบว่าวงจรส่วนนี้ไม่มีปัญหาอะไร สามารถทำงานได้ตามที่ออกแบบไว้ คือแปลงสัญญาณอนาลอกเป็นดิจิตอลเมื่อได้รับคำสั่งจากส่วนควบคุม

1.3 ไมโครคอนโทรลเลอร์(AT89C2051)

เป็นส่วนควบคุมการทำงานของอุปกรณ์สลาฟ หน้าที่หลักคือรอรับการติดต่อที่มาจากอุปกรณ์มาสเตอร์จากนั้นจึงไปควบคุมให้ A/D ทำงาน

จากการทดลองการทำงานไม่พบปัญหาอะไรทั้งการรับการติดต่อที่มาจากอุปกรณ์มาสเตอร์และการควบคุม A/D แต่ปัญหาก็มีเล็กน้อยคือ ไมโครคอนโทรลเลอร์เบอร์AT89C2051ที่เลือกใช้ในโครงงานเป็นไมโครคอนโทรลเลอร์ขนาดเล็ก(20ขา) มี I/O พอร์ทมาให้เพียง 2 พอร์ท จึงลำบากในการเพิ่มเติมอุปกรณ์ต่อพ่วงอื่นๆเข้าไปในอนาคต

2. อุปกรณ์มาสเตอร์

2.1 LCD โมดูล

สามารถทำงานได้เป็นปกติ แต่มีข้อจำกัดในเรื่องของจำนวนข้อมูลที่แสดงได้ในแต่ละหน้าจอ เนื่องจากเป็นจอ LCD ขนาด 2 บรรทัด

2.2 Real Time Clock

เป็นฐานข้อมูลทางเวลาให้กับระบบในโครงงานใช้ IC DS1307 การทำงานส่วนใหญ่ไม่พบปัญหาอะไร แต่ปัญหาที่จะพบเป็นบางครั้งก็คือ ค่าที่วัดได้จากการอ่านข้อมูลใน DS1307 ผิดเพี้ยนไปซึ่งน่าจะเกิดจากกราวนด์ของวงจรมีปัญหา

2.3 หน่วยความจำ

เป็นที่เก็บข้อมูลของอุปกรณ์มาสเตอร์ ก่อนที่จะทำการย้ายข้อมูลไปไว้ในคอมพิวเตอร์ จากการทดลองพบว่าสามารถอ่านและเขียนข้อมูลไปยังหน่วยความจำได้โดยไม่มีปัญหา(ซอร์สโค้ดดูได้ในภาคผนวก ส่วนรูปวงจรและ การคำนวณระยะเวลาที่หน่วยความจำสามารถเก็บค่าได้ ดูในบทที่ 3)

2.4 ไมโครคอนโทรลเลอร์(AT89S8252)

สามารถควบคุมอุปกรณ์ต่อพ่วงทุกตัวได้เป็นอย่างดี ปัญหาคือ ในการนำข้อมูล 8 บิท(ที่ได้มาจากอุปกรณ์สลาฟ) มาหาค่าเดซิเบลจะใช้วิธีการเปิดค่าจากในตาราง(Look-up Table) ดังนั้นจึงต้องการพื้นที่ที่ใช้สำหรับเก็บโปรแกรมจำนวนมาก

3. ส่วนของระบบสื่อสัญญาณระหว่างอุปกรณ์สลาฟและมาสเตอร์

ในส่วนนี้ใช้การสื่อสารแบบอนุกรมระหว่างอุปกรณ์ทั้งสองโดยสามารถแยกอุปกรณ์สลาฟออกไปไกลจากอุปกรณ์มาสเตอร์ได้หลายร้อยเมตรโดยการใช้ IC MAX487 เป็นตัวไดรว์กระแส ส่วนจำนวนของอุปกรณ์สลาฟที่นำมาใช้เชื่อมต่อได้คือ 256 ตัว แต่ทั้งนี้จำนวนของอุปกรณ์สลาฟที่นำมาต่อพ่วงจะมีผลกับระยะเวลาที่หน่วยความจำสามารถเก็บค่าได้(ดูในบทที่ 3)

4. ส่วนของการเชื่อมต่อกับคอมพิวเตอร์

ส่วนนี้เป็นการพัฒนาโปรแกรมติดต่อกับผู้ใช้(Graphic User Interface,GUI)เพื่อให้ผู้ใช้สามารถดึงข้อมูลจากหน่วยความจำของอุปกรณ์มาสเตอร์มาเก็บและแสดงผลบนจอคอมพิวเตอร์ได้ แนวทางในการพัฒนาในส่วนนี้คือ การเขียนโปรแกรมให้สามรถนำข้อมูลมาแสดงผลเป็นรูปกราฟได้

บรรณานุกรม

[1] นิรุธ อำนวยศิลป์, "คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0" ,ซัคเซส มีเดีย กรุงเทพฯ พ.ศ. 2544

[2] รศ.นิภา ลีลารุจิ , "วิศวกรรมกระจายเสียง" , สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, พิมพ์ครั้งที่ 2 พ.ศ.2541

[3] ประจิน พลังสันติกุล, "เรียนรู้และใช้งาน CCS C คอมไพเลอร์ เขียนโปรแกรมภาษา C ควบคุมไมโครคอนโทรลเลอร์ ", บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด

ภาคผนวก

โปรแกรมของส่วนควบคุมหลัก (Master)

```
// master.c

#include <REG8252.H>
#include<i2c.h>
#include<ds1307.h>
#include<sram.h>
#include<scankey.h>
#include<lcd.h>
#include<page.h>
#include<setupclock.h>
#include<showdb.h>
#include<serial.h>
//-------------------------------------------------------------------------
-------
#define ok_button 0x11
#define no_press 0xFF
#define back 0x10
//-------------------------------------------------------------------------
-------
unsigned char hour,min,sec1,sec2,1sec,d=0x00,i;
unsigned char key,r,q,dat=0x00,slv=0x01,num_slv = 0x04;
unsigned int addr_ram,n,j;
bit status1=0,status2=0,m=0,ex=0;
sbit re = P1^6;                 //pin 7
sbit de = P1^7;                 //pin 8
//--------------------Delay3-----------------------------------------
-------------
void delay_3(unsigned char tick)
{
      unsigned char y,z;
      for(y=0;y<tick;y++)
      for(z=0;z<200;z++);
}
//-----------------switch_right-----------------------------------
-------------
void switch_right(void) interrupt 0
{
      if((m==0)&&(slv<num_slv))
            slv++;
      delay_3(150);
}
//-----------------switch_left-----------------------------------
-------------
void switch_left(void) interrupt 2
{
      if((m==0)&&(slv>1))
            slv--;
      delay_3(150);
}
//-----------------delay_1-----------------------------------
-------------
void delay_1(unsigned d_sec)
{
      i = 0;
      sec1  = ds1307_rd(0x00);
      while((i<d_sec)&&(~ex))
```

```
{
            key = scankey();
            if((key == ok_button)&&(m == 0))

            {
                        lcd_clear();
                        m = 1;
            }
            if(key == back)
            {
                  if(m == 0)
                  {
                        lcd_clear();
                        ex = 1;
                  }
                  else
                  {
                        lcd_clear();
                        m = 0;
                  }
            }
            if(m == 1)
            {
                  IE = 0x00;
                  if(slv == 1)
                  {
                        showdb(sram_rd(addr_ram-1));
                        page_6();
                  }
                  if(slv == 2)
                  {
                        showdb(sram_rd((addr_ram-1)+16382));
                        page_7();
                  }
                  if(slv == 3)
                  {
                        showdb(sram_rd((addr_ram-1)+2*16382));
                        page_8();
                  }
                  if(slv == 4)
                  {
                        showdb(sram_rd((addr_ram-1)+3*16382));
                        page_9();
                  }
                  hour = ds1307_rd(0x02);
                  min = ds1307_rd(0x01);
                  sec2 = ds1307_rd(0x00);
                  lcd_showtime(hour,0x85);
                  lcd_showchar(0x3A);
                  lcd_showtime(min,0x88);
                  lcd_showchar(0x3A);
            lcd_showtime(sec2,0x8B);

                  if(sec2 != sec1)
                        {
                              i++;
                              sec1 = sec2;
                        }
            }
            if((m==0)&&(~ex))
            {
```

```
                        IE = 0x85;
                        page_5();
                        lcd_command(0xC5+(slv-1)*2);
                        lcd_showchar(0x3E);
                        hour = ds1307_rd(0x02);
                        min = ds1307_rd(0x01);
                        sec2 = ds1307_rd(0x00);
                        lcd_showtime(hour,0x85);
                        lcd_showchar(0x3A);
                        lcd_showtime(min,0x88);
                        lcd_showchar(0x3A);
                        lcd_showtime(sec2,0x8B);

                        if(sec2 != sec1)
                              {
                                    i++;
                                    sec1 = sec2;
                              }
                  }
      }
}
//---------------------Serial_intrp---------------------------------
-----------------
void serial_intrp(void) interrupt 4
{
      if(RI)
      {
            RI = 0;
            if(SBUF == 0x30)
                  status1 = 1;
            if((SBUF == 0x31)&&(status1 == 1))
                  status2 = 1;
      }
      if(TI)
            TI = 0;
}
//-----------------------Delay2---------------------------------------
-----------------
void delay_2(unsigned d2_sec)
{
      i=0;
      sec1  = ds1307_rd(0x00);
      while(i<d2_sec)

      {
                  sec2  = ds1307_rd(0x00);
                  if(sec2 != sec1)
                        {
                              i++;
                              sec1 = sec2;
                        }
      }
}
//-------------------------****MAIN****-------------------------------
-----------------
void main(void)
{
      lcd_init();
      serial_init();
      SM2 = 0;
      TB8 = 1;
```

```
	IT0 = 1;
	IT1 = 1;
	de = 0;
	re = 0;
	page_1();
                                //Welcome To My Project
	while(scankey() != ok_button);
	lcd_clear();
	page_2();
                                //You Must Set Time First
	while(scankey() != ok_button);
	lcd_clear();
	page_3();
                                //Setup Clock Screen
	setclock();
	sec1 = sram_rd(0x0005);
                        //sec before getdata = sec1
	addr_ram = 0x0008;

	while(~ex)
		{
                        IE = 0x00;
                        lcd_clear();
                        page_4();
                                //Working...
                        for(i=1;i<=num_slv;i++)
                                //get data
                        {
                                de = 1;
                                serial_snd(i);
                                de = 0;
                                dat=serial_rcv();
                                sram_wr(addr_ram+(i-1)*16382,dat);
                        }
                        addr_ram++;
                        if(addr_ram>16389)
                                addr_ram = 0x0008;
                        sec2 = ds1307_rd(0x00);
                                //sec after getdata = sec2
                        if(sec1<sec2)
                                d = sec2-sec1;
                        if(sec1>sec2)
                                d = (60%sec1)+sec2;
                        if(sec1 == sec2)
                                d = 0;
                        delay_1(5-d);
		}
		lsec = sec2;
		n = addr_ram;
//---------------------------***COM***-----------------------------------
-----------
		IE = 0x90;
		while(1)
		{
			if(status1 == 0)
			{
                                lcd_clear();
                                page_10();
                                lcd_showtime(hour,0xC5);
		//show hour
                                lcd_showchar(0x3A);
```

```
                lcd_showtime(min,0xC8);
            //show min
                lcd_showchar(0x3A);
                lcd_showtime(lsec,0xCB);
            //show sec
                delay_2(2);
            }
            if(status1 == 0)
            {
                lcd_clear();
                page_11();
                lcd_showtime(sram_rd(0x0003),0x85);
            //show start hour
                lcd_showchar(0x3A);
                lcd_showtime(sram_rd(0x0004),0x88);
//show start min
                lcd_showchar(0x3A);
                lcd_showtime(sram_rd(0x0005),0x8B);
      //show start sec
                lcd_showtime(hour,0xC5);
      //show last hour
                lcd_showchar(0x3A);
                lcd_showtime(min,0xC8);
            //show last min
                lcd_showchar(0x3A);
                lcd_showtime(lsec,0xCB);
            //show last sec
                delay_2(2);
            }
            if(status1 == 0)
            {
                lcd_clear();
                page_12();
                lcd_origin();
                r=(num_slv*(n-8))/10000;
                q=(num_slv*(n-8))%10000;
                lcd_showchar(0x30|r);
                r=q/1000;
                q=q%1000;
                lcd_showchar(0x30|r);
                r=q/100;
                q=q%100;
                lcd_showchar(0x30|r);
                r=q/10;
                q=q%10;
                lcd_showchar(0x30|r);
                lcd_showchar(0x30|q);
                delay_2(2);
            }
            if(status1 == 1)
            //connected to pc
            {
                SBUF = 0x39;
                //ACK to PC
                lcd_clear();
                page_13();

                while(status2 == 0);
                status1 = 0;
                status2 = 0;
```

```
                        page_14();
//Send Time
      for(addr_ram=0x0000;addr_ram<0x0008;addr_ram++)
                        {
                              SBUF = sram_rd(addr_ram);
                        }
//Send Data
                        j=0;
                        for(addr_ram=0x0008;addr_ram<n;addr_ram++)
                        {
                              for(i=1;i<=num_slv;i++)
                              {
                                    SBUF = sram_rd(addr_ram+(i-
1)*16382);
                              }
                              j++;
                              r=(num_slv*j)/10000;
                              q=(num_slv*j)%10000;
                              lcd_showchar(0x30|r);
                              r=q/1000;
                              q=q%1000;
                              lcd_showchar(0x30|r);
                              r=q/100;
                              q=q%100;
                              lcd_showchar(0x30|r);
                              r=q/10;
                              q=q%10;
                              lcd_showchar(0x30|r);
                              lcd_showchar(0x30|q);
                              lcd_command(0xC7);
                        }
                        delay_2(2);
                        page_15();
                        //Finished Sending
                        delay_2(2);
                  }
            }
}
```

โปรแกรมของส่วนควบคุมย่อย (Slave)

//slave1

```
#include<REG2051.H>
#include<serial.h>
//-----------------------------------------------------------------------
unsigned char dat = 0x00;
sbit wr = P3^7;          //pin 11
sbit de = P3^3;      //pin 7
sbit re = P3^4;          //pin 8
//-----------------------------------------------------------------------
void A2D_intrp(void) interrupt 0
{
      de = 1;
      SBUF = P1;
}
//-----------------------------------------------------------------------
void master_intrp(void) interrupt 4
{
      unsigned char id_slv=0x01;          //****address of slave****
      if(RI)
      {
            RI = 0;
            RB8 = 0;
            if(SBUF==id_slv)
            {
                  wr = 0;
                  wr = 1;
            }
      }
      if(TI)
      {
            TI = 0;
            de = 0;
      }
}
//-----------------------------------------------------------------------
void main(void)
{
      serial_init();
      SM2 = 1;
      TB8 = 0;
      P1 = 0xFF;
      IE = 0x91;
      IT0 = 1;
      de = 0;
      re = 0;
      wr = 1;
      while(1);
}
```

//slave2

```
#include<REG2051.H>
#include<serial.h>
//-----------------------------------------------------------------------
unsigned char dat = 0x00;
sbit wr = P3^7;          //pin 11
sbit de = P3^3;      //pin 7
sbit re = P3^4;          //pin 8
//-----------------------------------------------------------------------
void A2D_intrp(void) interrupt 0
{
      de = 1;
      SBUF = P1;
}
//-----------------------------------------------------------------------
void master_intrp(void) interrupt 4
{
      unsigned char id_slv=0x02;          //****address of slave****
      if(RI)
      {
            RI = 0;
            RB8 = 0;
            if(SBUF==id_slv)
            {
                  wr = 0;
                  wr = 1;
            }
      }
      if(TI)
      {
            TI = 0;
            de = 0;
      }
}
//-----------------------------------------------------------------------
void main(void)
{
      serial_init();
      SM2 = 1;
      TB8 = 0;
      P1 = 0xFF;
      IE = 0x91;
      IT0 = 1;
      de = 0;
      re = 0;
      wr = 1;
      while(1);
}
```

//slave3

```
#include<REG2051.H>
#include<serial.h>
//---------------------------------------------------------------------
unsigned char dat = 0x00;
sbit wr = P3^7;          //pin 11
sbit de = P3^3;      //pin 7
sbit re = P3^4;          //pin 8
//---------------------------------------------------------------------
void A2D_intrp(void) interrupt 0
{
      de = 1;
      SBUF = P1;
}
//---------------------------------------------------------------------
void master_intrp(void) interrupt 4
{
      unsigned char id_slv=0x03;          //****address of slave****
      if(RI)
      {
            RI = 0;
            RB8 = 0;
            if(SBUF==id_slv)
            {
                  wr = 0;
                  wr = 1;
            }
      }
      if(TI)
      {
            TI = 0;
            de = 0;
      }
}
//---------------------------------------------------------------------
void main(void)
{
      serial_init();
      SM2 = 1;
      TB8 = 0;
      P1 = 0xFF;
      IE = 0x91;
      IT0 = 1;
      de = 0;
      re = 0;
      wr = 1;
      while(1);
}
```

//slave4

```
#include<REG2051.H>
#include<serial.h>
//---------------------------------------------------------------------
unsigned char dat = 0x00;
sbit wr = P3^7;          //pin 11
sbit de = P3^3;      //pin 7
sbit re = P3^4;          //pin 8
//---------------------------------------------------------------------
void A2D_intrp(void) interrupt 0
{
      de = 1;
      SBUF = P1;
}
//---------------------------------------------------------------------
void master_intrp(void) interrupt 4
{
      unsigned char id_slv=0x04;            //****address of slave****
      if(RI)
      {
            RI = 0;
            RB8 = 0;
            if(SBUF==id_slv)
            {
                  wr = 0;
                  wr = 1;
            }
      }
      if(TI)
      {
            TI = 0;
            de = 0;
      }
}
//---------------------------------------------------------------------
void main(void)
{
      serial_init();
      SM2 = 1;
      TB8 = 0;
      P1 = 0xFF;
      IE = 0x91;
      IT0 = 1;
      de = 0;
      re = 0;
      wr = 1;
      while(1);
}
```

โปรแกรมส่วนของ Library File

```
// ds1307.h

#define DS1307_ID 0xD0
//---------------------------------------------------------------------
--------------
void ds1307_wrdate(unsigned char day,unsigned char month,unsigned
char year)
{
      i2c_start();                                        //start
condition
      i2c_wrdata(DS1307_ID);                        //select ds1307
      i2c_wrdata(0x04);
      i2c_wrdata(day);
      i2c_wrdata(month);
      i2c_wrdata(year);
      i2c_stop();
}

void ds1307_wrtime(unsigned char hr,unsigned char min,unsigned char
sec)
{
      i2c_start();
      i2c_wrdata(DS1307_ID);
      i2c_wrdata(0x00);
      i2c_wrdata(sec);
      i2c_wrdata(min);
      i2c_wrdata(hr);
      i2c_stop();
}

unsigned char ds1307_rd(unsigned char addr)
{
      unsigned char ret=0x00;
      i2c_start();
      i2c_wrdata(DS1307_ID);
      i2c_wrdata(addr);
      i2c_start();
      i2c_wrdata(DS1307_ID+1);
      ret = i2c_rddata();
      i2c_stop();
      return(ret);
}
```

```
// i2c.h

#include<intrins.h>
//-----------------------------
sbit SCL = P1^0;          //pin 1
sbit SDA = P1^1;          //pin 2
//-----------------------------
void i2c_delay(void)
{
      unsigned char i;
      for(i=0;i<20;i++)
      _nop_();
}

void i2c_clk(void)
{
      i2c_delay();
      SCL = 1;
      i2c_delay();
      SCL = 0;
}

void i2c_start(void)
{
      if(SCL)
      SCL = 0;
      SDA = 1;
      SCL = 1;
      i2c_delay();
      SDA = 0;
      i2c_delay();
      SCL = 0;
}

void i2c_stop(void)
{
      if(SCL)
      SCL = 0;
      SDA = 0;
      i2c_delay();
      SCL = 1;
      i2c_delay();
      SDA = 1;
}

/*void i2c_nack(void)
{
      SDA = 1;
      i2c_delay();
      i2c_clk();
      SCL = 1;
}*/

bit i2c_wrdata(unsigned char dat)
{
      bit data_bit;
      unsigned char i;
      for(i=0;i<8;i++)
      {
            data_bit = dat & 0x80;
            SDA = data_bit;
```

```
            i2c_clk();
            dat = dat<<1;
        }
        SDA = 1;
        i2c_delay();
        SCL = 1;
        i2c_delay();
        data_bit = SDA;
        SCL = 0;
        i2c_delay();
        return(data_bit);
}

unsigned char i2c_rddata(void)
{
        bit rd_bit;
        unsigned char i,dat;
        dat = 0x00;
        for(i=0;i<8;i++)
        {
            i2c_delay();
            SCL = 1;
            i2c_delay();
            rd_bit = SDA;
            dat = dat<<1;
            dat = dat | rd_bit;
            SCL = 0;
        }
        SDA = 1;
        i2c_delay();
        i2c_clk();
        SCL = 1;
        return(dat);
}
```

```
// lcd.h

#include<string.h>
//-------------------------------------------------------------------------
--
sbit e = P1^4;                  //pin 5
sbit rs = P1^5;                 //pin 6
//-------------------------------------------------------------------------
---
void delay(int tick)
{
      int i,j;                          // For keep counter loop
      for(i=0;i<tick;i++)               // Loop delay
      for(j=0;j<250;j++);
}

void lcd_command(unsigned char com) //write command
{
      rs = 0;                           // Write Command to LCD
      e = 1;                            // Start generate pulse clock LCD
      P0 = com;                   // Send data to LCD port
      delay(10);
      e = 0;
      delay(10);
}

void lcd_oncursor()
{
      lcd_command(0x0F);
}

void lcd_offcursor()
{
      lcd_command(0x0C);
}

void lcd_clear()
{
      lcd_command(0x01);
}

void lcd_right()
{
      lcd_command(0x14);
}

void lcd_left()
{
      lcd_command(0x10);
}

void lcd_origin()
{
      lcd_command(0x02);
}

void lcd_showchar(unsigned char character)
{
      rs = 1;                           // Write Data to LCD
      e = 1;                            //Start generate pulse clock LCD
      P0 = character;             // Send data to LCD port
```

```
        delay(10);
        e = 0;
        delay(10);
}

void lcd_showtext(unsigned char *text,unsigned char position)
{
        unsigned char i;
        lcd_origin();
        lcd_command(position);
        for(i=0;i<strlen(text);i++)
              lcd_showchar(*(text+i));
}

void lcd_showtime(unsigned char dat,unsigned char position)
{
        unsigned char buf1,buf2 = 0;
        buf1 = dat & 0xF0;
        buf1 = (buf1>>4)|(0x30);
        buf2 = dat & 0x0F;
        buf2 = (buf2)|(0x30);
        lcd_origin();
        lcd_command(position);
        lcd_showchar(buf1);
        lcd_showchar(buf2);
}

void lcd_init()
{
        delay(500);                    // Delay for initial LCD
        lcd_command(0x38);       // 8 bit display ,5*7 dot,2 lines
        lcd_offcursor();               // Display ON , none cursor
        lcd_clear();                   // Clear screen
}
```

```
// page.h

unsigned char *ok_text = "OK";
//-------------------------------------------------
void page_1(void)
{
      unsigned char *line1 = "Welcome to";
      unsigned char *line2 = "My Project";
      lcd_showtext(line1,0x82);
      lcd_showtext(line2,0xC2);
      lcd_showtext(ok_text,0xCE);
}

void page_2(void)
{
      unsigned char *line1 = "You must set";
      unsigned char *line2 = "Clock First";
      lcd_showtext(line1,0x82);
      lcd_showtext(line2,0xC2);
      lcd_showtext(ok_text,0xCE);
}

void page_3(void)
{
      unsigned char *line1 = "Date dd/mm/yy";
      unsigned char *line2 = "Time hh:mm:ss";
      lcd_showtext(line1,0x80);
      lcd_showtext(line2,0xC0);
}

void page_4(void)
{
      unsigned char *line1 = "Working...";
      lcd_showtext(line1,0x80);
}

void page_5(void)
{
      unsigned char *line1 = "Time ";
      unsigned char *line2 = "Slave 1 2 3 4";
      lcd_showtext(line1,0x80);
      lcd_showtext(line2,0xC0);
      lcd_showtext(ok_text,0xCE);
}
void page_6(void)
{
      unsigned char *line1 = "Time ";
      unsigned char *line2 = "Slv1";
      lcd_showtext(line1,0x80);
      lcd_showtext(line2,0xC0);
}
void page_7(void)
{
      unsigned char *line1 = "Time ";
      unsigned char *line2 = "Slv2";
      lcd_showtext(line1,0x80);
      lcd_showtext(line2,0xC0);
}
void page_8(void)
{
      unsigned char *line1 = "Time ";
```

```
        unsigned char *line2 = "Slv3";
        lcd_showtext(line1,0x80);
        lcd_showtext(line2,0xC0);
}
void page_9(void)
{
        unsigned char *line1 = "Time ";
        unsigned char *line2 = "Slv4";
        lcd_showtext(line1,0x80);
        lcd_showtext(line2,0xC0);
}

void page_10(void)
{
        unsigned char *line1 = "SystemTerminated";
        unsigned char *line2 = "at";
        lcd_showtext(line1,0x80);
        lcd_showtext(line2,0xC0);
}

void page_11(void)
{
        unsigned char *line1 = "from";
        unsigned char *line2 = "to";
        lcd_showtext(line1,0x80);
        lcd_showtext(line2,0xC0);
}

void page_12(void)
{
        unsigned char *line2 = "Data Collected";
        lcd_showtext(line2,0xC1);
;

void page_13(void)
{
        unsigned char *line1 = "Connected to PC";
        lcd_showtext(line1,0x80);
;

void page_14(void)
{
        unsigned char *line2 = "Sending     Bytes";
        lcd_showtext(line2,0xC0);
}

void page_15(void)
{
        unsigned char *line2 = "Finished Sending";
        lcd_showtext(line2,0xC0);
}
```

// scankey.h

```
sbit c1 = P2^6;          // Bit Column
sbit c2 = P2^5;          // Bit Column2
sbit c3 = P2^4;          // Bit Column3
sbit r1 = P2^0;          // Bit Row1
sbit r2 = P2^1;          // Bit Row2
sbit r3 = P2^2;          // Bit Row3
sbit r4 = P2^3;          // Bit Row4
//-------------------------------------------------------------------
void delay_db(int time)
{
      do                 // do-while loop for delay
      {
       time--;                  // Decrease counter
      }while(time>0);           // If time>0 work in block
}

unsigned char scankey(void)      // Return value key
{
      unsigned char ret = 0xFF;      // Initial value ret = 0xFF
//-------------------------------------------------------------------
----------
      c1 = 0;                     // Start Check column1
      if(r1==0)                  // Check push key 1
      {
                  delay_db(30000);          // Delay debounce
            ret = 0x01;                 // Return value = 0x01
      }
      if(r2==0)                  // Check push key 4
      {
            delay_db(30000);    // Delay debounce
            ret = 0x04;         // Return value = 0x04
      }
      if(r3==0)                  // Check push key 7
      {
            delay_db(30000);    // Delay debounce
            ret = 0x07;         // Return value = 0x07
      }
      if(r4==0)                  // Check push key *
      {
            delay_db(30000);    // Delay debounce
            ret = 0x10;         // Return value = 0x10
      }
      c1 = 1;                     // Stop  check Column1
//-------------------------------------------------------------------
--
      c2 = 0;                       // Start Check column2
      if(r1==0)                  // Check push key 2
      {
            delay_db(30000);          // Delay debounce

            ret = 0x02;         // Return value = 0x02
      }
      if(r2==0)                  // Check push key 5
      {
            delay_db(30000);          // Delay debounce
            ret = 0x05;         // Return value = 0x05
      }
      if(r3==0)                  // Check push key 8
      {
```

```
            delay_db(30000);          // Delay debounce
            ret = 0x08;                     // Return value = 0x08
      }
      if(r4==0)                  // Check push key 0
      {
            delay_db(30000);          // Delay debounce
            ret = 0x00;          // Return value = 0x00
      }
      c2 = 1;                      // Stop  check Column2
//-------------------------------------------------------------------
----------
      c3 = 0;                     // Start Check column3
      if(r1==0)                  // Check push key 3
      {
            delay_db(30000);  // Delay debounce
            ret = 0x03;                     // Return value = 0x03
      }
      if(r2==0)                  // Check push key 6
      {
            delay_db(30000);          // Delay debounce
            ret = 0x06;          // Return value = 0x06
      }
      if(r3==0)                  // Check push key 9
      {
            delay_db(30000);          // Delay debounce
            ret = 0x09;          // Return value = 0x09
      }
      if(r4==0)                  // Check push key #
      {
            delay_db(30000);          // Delay debounce
            ret = 0x11;          // Return value = 0x11
      }
      c3 = 1;                     // Stop check Column3
//-------------------------------------------------------------------
-------------
      return(ret);                     // Return key value
}
```

```
// serial.h

void serial_init(void)
{
      PCON = 0x00;
      TMOD = 0x20;
      SCON = 0xF0;       //Mode3 MultiProcessor
      TH1 = 0xFA;        //Baud Rate = 9600
      TR1 = 1;
}

void serial_snd(unsigned char a)
{
      SBUF = a;
      while(~TI);
      TI = 0;
}

unsigned char serial_rcv(void)
{
      unsigned char b;
      while(~RI);
      RI = 0;
      b = SBUF;
      return(b);
}
```

```
// setupclock.h

#define ok_button 0x11
#define no_press 0xFF
#define back 0x10
unsigned char day_,month_,year_,hour_,min_,sec_;
//-------------------------------------------------------------------
----
void setclock(void)
{
      unsigned char key=0xFF,position=0x85,c=0,buf[12],*ok_text =
"OK";
      bit status=0;
      lcd_origin();
      lcd_command(0x85);
      lcd_oncursor();
      while(!((status==1)&&(key==ok_button)))
      {
            key=scankey();
            if(key!=no_press)
                  {
                        if(key!=ok_button)
                              {
                                    if(key!=back)
                                          {
                                                if(c<12)
                                                      {

      lcd_showchar(0x30|key);

      position++;

      if((position==0x87)||(position==0x8A)||(position==0xC7)||(posit
ion==0xCA))
                                                                  {

      lcd_right();

      position++;
                                                                  }

      if(position==0x8D)
                                                                  {

      lcd_offcursor();

      lcd_origin();

      lcd_command(0xC5);

      lcd_oncursor();

      position=0xC5;
                                                                  }

      if(position==0xCD)
                                                                  {

      status = 1;

      lcd_offcursor();
```

```
        lcd_showtext(ok_text,0xCE);
        lcd_origin();
        lcd_command(0xCD);
        lcd_oncursor();
                                                                    }
        buf[c]=key;
                                                                c++;
                                                            }
                                                    }
                                            else
                                                    {
                                                        if(c>0)
                                                            {

        lcd_left();
                                                                position-
-;
        if((position==0x87)||(position==0x8A)||(position==0xC7)||(posit
ion==0xCA))
                                                                {
        lcd_left();
        position--;
                                                                }
        if(position==0xC4)
                                                                {
        lcd_offcursor();
        lcd_origin();
        lcd_command(0x8C);
        lcd_oncursor();
        position=0x8C;
                                                                }
                                                            c--;
                                                        }
                                                    }
                                                }
                                            }
        }

        lcd_offcursor();
        day_=(buf[0]<<4)|(buf[1]);
        month_=(buf[2]<<4)|(buf[3]);
        year_= (buf[4]<<4)|(buf[5]);
        hour_= (buf[6]<<4)|(buf[7]);
        min_= (buf[8]<<4)|(buf[9]);
        sec_= (buf[10]<<4)|(buf[11]);
        ds1307_wrdate(day_,month_,year_);
```

```
        ds1307_wrtime(hour_,min_,sec_);
        sram_wr(0x0000,day_);
        sram_wr(0x0001,month_);
        sram_wr(0x0002,year_);
        sram_wr(0x0003,hour_);
        sram_wr(0x0004,min_);
        sram_wr(0x0005,sec_);
}
```

```
//  sram.h

#define data_id 0x70
#define addrlow_id 0x72
#define addrhigh_id 0x74
//-------------------------------------------------------------
sbit wr=P1^2;      //pin 3
sbit rd=P1^3;      //pin 4
//-------------------------------------------------------------
void sram_wr(unsigned int addr,unsigned char dat)
{
      unsigned char addrhigh,addrlow;
      addrhigh = addr & 0xFF00;
      addrhigh = addrhigh>>8;
      addrlow = addr & 0x00FF;
      wr=1;
      rd=1;
      i2c_start();
      i2c_wrdata(addrlow_id);
      i2c_wrdata(addrlow);
      i2c_stop();
      i2c_start();
      i2c_wrdata(addrhigh_id);
      i2c_wrdata(addrhigh);
      i2c_stop();
      i2c_start();
      i2c_wrdata(data_id);
      i2c_wrdata(dat);
      i2c_stop();
      wr=0;
}

unsigned char sram_rd(unsigned int addr)
{
      unsigned char ret;
      unsigned char addrhigh,addrlow;
      addrhigh = addr & 0xFF00;
      addrhigh = addrhigh>>8;
      addrlow = addr & 0x00FF;
      wr=1;
      rd=1;
      i2c_start();
      i2c_wrdata(addrlow_id);
      i2c_wrdata(addrlow);
      i2c_stop();
      i2c_start();
      i2c_wrdata(addrhigh_id);
      i2c_wrdata(addrhigh);
      i2c_stop();
      i2c_start();
      i2c_wrdata(data_id);
      i2c_wrdata(0xFF);
      i2c_stop();
      rd=0;
      i2c_start();
      i2c_wrdata(data_id+1);
      ret=i2c_rddata();
      i2c_stop();
      return(ret);
}
```

```
// showdb.h

void showdb(unsigned char dat)
{
      unsigned char *r;
      switch(dat)
      {
            case 0x00   :     r = "Avr=00.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x01   :     r = "Avr=43.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x02   :     r = "Avr=44.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x03   :     r = "Avr=45.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x04   :     r = "Avr=46.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x05   :     r = "Avr=47.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x06   :     r = "Avr=48.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x07   :     r = "Avr=49.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x08   :     r = "Avr=50.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x09   :     r = "Avr=51.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x0A   :     r = "Avr=52.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x0B   :     r = "Avr=53.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x0C   :     r="Avr=54.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x0D   :     r="Avr=54.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x0E   :     r="Avr=55.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x0F   :     r="Avr=56.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x10   :     r="Avr=56.00dB";
                              lcd_showtext(r,0xC5);
                              break;
            case 0x11   :     r="Avr=56.00dB";
                              lcd_showtext(r,0xC5);
                              break;
```

```
case 0x12   :   r="Avr=56.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x13   :   r="Avr=57.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x14   :   r="Avr=57.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x15   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x16   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x17   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x18   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x19   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x1A   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x1B   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x1C   :   r="Avr=58.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x1D   :   r="Avr=59.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x1E   :   r="Avr=60.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x1F   :   r="Avr=60.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x20   :   r="Avr=60.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x21   :   r="Avr=60.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x22   :   r="Avr=60.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x23   :   r="Avr=61.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x24   :   r="Avr=61.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x25   :   r="Avr=61.00dB";
                lcd_showtext(r,0xC5);
                break;
case 0x26   :   r="Avr=61.00dB";
```

```
                        lcd_showtext(r,0xC5);
                        break;
case 0x27   :       r="Avr=61.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x28   :       r="Avr=61.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x29   :       r="Avr=61.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x2A   :       r="Avr=61.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x2B   :       r="Avr=61.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x2C   :       r="Avr=61.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x2D   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x2E   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x2F   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x30   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x31   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x32   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x33   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x34   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x35   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x36   :       r="Avr=62.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x37   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x38   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x39   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x3A   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
```

```
                        break;
case 0x3B   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x3C   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x3D   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x3E   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x3F   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x40   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x41   :       r="Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x42   :       r = "Avr=63.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x43   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x44   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x45   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x46   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x47   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x48   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x49   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x4A   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x4B   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x4C   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x4D   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x4E   :       r = "Avr=64.00dB";
                        lcd_showtext(r,0xC5);
                        break;
```

```
case 0x4F   :      r = "Avr=64.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x50   :      r = "Avr=64.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x51   :      r = "Avr=64.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x52   :      r = "Avr=64.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x53   :      r = "Avr=64.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x54   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x55   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x56   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x57   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x58   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x59   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x5A   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x5B   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x5C   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x5D   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x5E   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x5F   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x60   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x61   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x62   :      r = "Avr=65.00dB";
                     lcd_showtext(r,0xC5);
                     break;
case 0x63   :      r = "Avr=66.00dB";
```

```
                        lcd_showtext(r,0xC5);
                        break;
case 0x64   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x65   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x66   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x67   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x68   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x69   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x6A   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x6B   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x6C   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x6D   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x6E   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x6F   :       r = "Avr=66.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x70   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x71   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x72   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x73   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x74   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x75   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x76   :       r = "Avr=67.00dB";
                        lcd_showtext(r,0xC5);
                        break;
case 0x77   :       r = "Avr=68.00dB";
                        lcd_showtext(r,0xC5);
```

```
                                    break;
            case 0x78   :       r = "Avr=68.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x79   :       r = "Avr=69.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x7A   :       r = "Avr=69.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x7B   :       r = "Avr=69.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x7C   :       r = "Avr=72.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x7D   :       r = "Avr=72.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x7E   :       r = "Avr=73.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x7F   :       r = "Avr=73.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x80   :       r = "Avr=73.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x81   :       r = "Avr=74.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x82   :       r = "Avr=74.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x83   :       r = "Avr=80.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x84   :       r = "Avr=82.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x85   :       r = "Avr=87.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            case 0x86   :       r = "Avr=90.00dB";
                                    lcd_showtext(r,0xC5);
                                    break;
            default             :       r = "Avr=xx.xxdB";
                                    lcd_showtext(r,0xC5);
                                    break;
        }
}
```

โปรแกรม Visual C ++ รับค่าทางพอร์ตอนุกรม

```
// TestPortsDlg.cpp : implementation file
#include "stdafx.h"
#include "TestPorts.h"
#include "TestPortsDlg.h"
#include "string.h"
#include "LookdataDlg.h"
#include <stdio.h>
#include <stdlib.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

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

// CAboutDlg dialog used for App About
DCB dcb;
char * chCommPort = "COM1";
int valReturn;
HANDLE hComp;
char chBuff[50];
unsigned long nReadPort;
unsigned long nWrittenPort;
CString r;
CString s[4];
int slave_count;
int sec_count;
int power_ten(int number){
      int i,ans;
      ans = 1 ;
      for(i=0;i<number;i++){
            ans = ans * 10;
      }
      return ans;
}
char* int_to_str(int number){
      char str[11] = { '0','0','0','0','0','0','0','0','0','0','\0'
};
      int i,j,p,s,tmp,max;
      max = 9; // str[11] => 11 - 2 = 9;
      j = 0;
      tmp = number;
      for(i=max;i>=0;i--){
            p = power_ten(i);
            s = (tmp / p);
            str[j] = s + '0';
            j++;
            tmp = tmp % p;
      }
      str[j] = '\0';
      for(i=0;i<=max;i++){
            if(str[i] != '0'){
                  return strdup(&str[i]);
                  break;
            }
      }
      return strdup(&str[max]);
```

```
}
void show(unsigned char dat)
{
      switch(dat)
      {
            case 0x00   :     r = "Avr=00.00dB";
                                    break;
            case 0x01   :     r = "Avr=43.00dB";
                                    break;
            case 0x02   :     r = "Avr=44.00dB";
                                    break;
            case 0x03   :     r = "Avr=45.00dB";
                                    break;
            case 0x04   :     r = "Avr=46.00dB";
                                    break;
            case 0x05   :     r = "Avr=47.00dB";
                                    break;
            case 0x06   :     r = "Avr=48.00dB";
                                    break;
            case 0x07   :     r = "Avr=49.00dB";
                                    break;
            case 0x08   :     r = "Avr=50.00dB";
                                    break;
            case 0x09   :     r = "Avr=51.00dB";
                                    break;
            case 0x0A   :     r = "Avr=52.00dB";
                                    break;
            case 0x0B   :     r = "Avr=53.00dB";
                                    break;
            case 0x0C   :     r="Avr=54.00dB";
                                    break;
            case 0x0D   :     r="Avr=54.00dB";
                                    break;
            case 0x0E   :     r="Avr=55.00dB";
                                    break;
            case 0x0F   :     r="Avr=56.00dB";
                                    break;
            case 0x10   :     r="Avr=56.00dB";
                                    break;
            case 0x11   :     r="Avr=56.00dB";
                                    break;
            case 0x12   :     r="Avr=56.00dB";
                                    break;
            case 0x13   :     r="Avr=57.00dB";
                                    break;
            case 0x14   :     r="Avr=57.00dB";
                                    break;
            case 0x15   :     r="Avr=58.00dB";
                                    break;
            case 0x16   :     r="Avr=58.00dB";
                                    break;
            case 0x17   :     r="Avr=58.00dB";
                                    break;
            case 0x18   :     r="Avr=58.00dB";
                                    break;
            case 0x19   :     r="Avr=58.00dB";
                                    break;
            case 0x1A   :     r="Avr=58.00dB";
                                    break;
            case 0x1B   :     r="Avr=58.00dB";
                                    break;
```

```
case 0x1C   :     r="Avr=58.00dB";
                        break;
case 0x1D   :     r="Avr=59.00dB";
                        break;
case 0x1E   :     r="Avr=60.00dB";
                        break;
case 0x1F   :     r="Avr=60.00dB";
                        break;
case 0x20   :     r="Avr=60.00dB";
                        break;
case 0x21   :     r="Avr=60.00dB";
                        break;
case 0x22   :     r="Avr=60.00dB";
                        break;
case 0x23   :     r="Avr=61.00dB";
                        break;
case 0x24   :     r="Avr=61.00dB";
                        break;
case 0x25   :     r="Avr=61.00dB";
                        break;
case 0x26   :     r="Avr=61.00dB";
                        break;
case 0x27   :     r="Avr=61.00dB";
                        break;
case 0x28   :     r="Avr=61.00dB";
                        break;
case 0x29   :     r="Avr=61.00dB";
                        break;
case 0x2A   :     r="Avr=61.00dB";
                        break;
case 0x2B   :     r="Avr=61.00dB";
                        break;
case 0x2C   :     r="Avr=61.00dB";
                        break;
case 0x2D   :     r="Avr=62.00dB";
                        break;
case 0x2E   :     r="Avr=62.00dB";
                        break;
case 0x2F   :     r="Avr=62.00dB";
                        break;
case 0x30   :     r="Avr=62.00dB";
                        break;
case 0x31   :     r="Avr=62.00dB";
                        break;
case 0x32   :     r="Avr=62.00dB";
                        break;
case 0x33   :     r="Avr=62.00dB";
                        break;
case 0x34   :     r="Avr=62.00dB";
                        break;
case 0x35   :     r="Avr=62.00dB";
                        break;
case 0x36   :     r="Avr=62.00dB";
                        break;
case 0x37   :     r="Avr=63.00dB";
                        break;
case 0x38   :     r="Avr=63.00dB";
                        break;
case 0x39   :     r="Avr=63.00dB";
                        break;
case 0x3A   :     r="Avr=63.00dB";
```

```
                        break;
case 0x3B   :       r="Avr=63.00dB";
                        break;
case 0x3C   :       r="Avr=63.00dB";
                        break;
case 0x3D   :       r="Avr=63.00dB";
                        break;
case 0x3E   :       r="Avr=63.00dB";
                        break;
case 0x3F   :       r="Avr=63.00dB";
                        break;
case 0x40   :       r="Avr=63.00dB";
                        break;
case 0x41   :       r="Avr=63.00dB";
                        break;
case 0x42   :       r = "Avr=63.00dB";
                        break;
case 0x43   :       r = "Avr=64.00dB";
                        break;
case 0x44   :       r = "Avr=64.00dB";
                        break;
case 0x45   :       r = "Avr=64.00dB";
                        break;
case 0x46   :       r = "Avr=64.00dB";
                        break;
case 0x47   :       r = "Avr=64.00dB";
                        break;
case 0x48   :       r = "Avr=64.00dB";
                        break;
case 0x49   :       r = "Avr=64.00dB";
                        break;
case 0x4A   :       r = "Avr=64.00dB";
                        break;
case 0x4B   :       r = "Avr=64.00dB";
                        break;
case 0x4C   :       r = "Avr=64.00dB";
                        break;
case 0x4D   :       r = "Avr=64.00dB";
                        break;
case 0x4E   :       r = "Avr=64.00dB";
                        break;
case 0x4F   :       r = "Avr=64.00dB";
                        break;
case 0x50   :       r = "Avr=64.00dB";
                        break;
case 0x51   :       r = "Avr=64.00dB";
                        break;
case 0x52   :       r = "Avr=64.00dB";
                        break;
case 0x53   :       r = "Avr=64.00dB";
                        break;
case 0x54   :       r = "Avr=65.00dB";
                        break;
case 0x55   :       r = "Avr=65.00dB";
                        break;
case 0x56   :       r = "Avr=65.00dB";
                        break;
case 0x57   :       r = "Avr=65.00dB";
                        break;
case 0x58   :       r = "Avr=65.00dB";
                        break;
```

```
case 0x59   :     r = "Avr=65.00dB";
                    break;
case 0x5A   :     r = "Avr=65.00dB";
                    break;
case 0x5B   :     r = "Avr=65.00dB";
                    break;
case 0x5C   :     r = "Avr=65.00dB";
                    break;
case 0x5D   :     r = "Avr=65.00dB";
                    break;
case 0x5E   :     r = "Avr=65.00dB";
                    break;
case 0x5F   :     r = "Avr=65.00dB";
                    break;
case 0x60   :     r = "Avr=65.00dB";
                    break;
case 0x61   :     r = "Avr=65.00dB";
                    break;
case 0x62   :     r = "Avr=65.00dB";
                    break;
case 0x63   :     r = "Avr=66.00dB";
                    break;
case 0x64   :     r = "Avr=66.00dB";
                    break;
case 0x65   :     r = "Avr=66.00dB";
                    break;
case 0x66   :     r = "Avr=66.00dB";
                    break;
case 0x67   :     r = "Avr=66.00dB";
                    break;
case 0x68   :     r = "Avr=66.00dB";
                    break;
case 0x69   :     r = "Avr=66.00dB";
                    break;
case 0x6A   :     r = "Avr=66.00dB";
                    break;
case 0x6B   :     r = "Avr=66.00dB";
                    break;
case 0x6C   :     r = "Avr=66.00dB";
                    break;
case 0x6D   :     r = "Avr=66.00dB";
                    break;
case 0x6E   :     r = "Avr=66.00dB";
                    break;
case 0x6F   :     r = "Avr=66.00dB";
                    break;
case 0x70   :     r = "Avr=67.00dB";
                    break;
case 0x71   :     r = "Avr=67.00dB";
                    break;
case 0x72   :     r = "Avr=67.00dB";
                    break;
case 0x73   :     r = "Avr=67.00dB";
                    break;
case 0x74   :     r = "Avr=67.00dB";
                    break;
case 0x75   :     r = "Avr=67.00dB";
                    break;
case 0x76   :     r = "Avr=67.00dB";
                    break;
case 0x77   :     r = "Avr=68.00dB";
```

```
                                        break;
            case 0x78   :       r = "Avr=68.00dB";
                                        break;
            case 0x79   :       r = "Avr=69.00dB";
                                        break;
            case 0x7A   :       r = "Avr=69.00dB";
                                        break;
            case 0x7B   :       r = "Avr=69.00dB";
                                        break;
            case 0x7C   :       r = "Avr=72.00dB";
                                        break;
            case 0x7D   :       r = "Avr=72.00dB";
                                        break;
            case 0x7E   :       r = "Avr=73.00dB";
                                        break;
            case 0x7F   :       r = "Avr=73.00dB";
                                        break;
            case 0x80   :       r = "Avr=73.00dB";
                                        break;
            case 0x81   :       r = "Avr=74.00dB";
                                        break;
            case 0x82   :       r = "Avr=74.00dB";
                                        break;
            case 0x83   :       r = "Avr=80.00dB";
                                        break;
            case 0x84   :       r = "Avr=82.00dB";
                                        break;
            case 0x85   :       r = "Avr=87.00dB";
                                        break;
            case 0x86   :       r = "Avr=90.00dB";
                                        break;
            default             :       r = "Avr=xx.xxdB";
                                        break;
      }

}

class CAboutDlg : public CDialog
{
public:
      CAboutDlg();
      static UINT Thread1(LPVOID Param);
// Dialog Data
      //{{AFX_DATA(CAboutDlg)
      enum { IDD = IDD_ABOUTBOX };
      //}}AFX_DATA

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

// Implementation
protected:
      //{{AFX_MSG(CAboutDlg)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
    CWinThread* pThread1;
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

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

// CTestPortsDlg dialog

CTestPortsDlg::CTestPortsDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CTestPortsDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CTestPortsDlg)
	m_text1 = _T("");
	m_text2 = _T("");
	m_sTime = _T("");
	m_time = _T("");
	numbyte = 0;
	control = "0";
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon
in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CTestPortsDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTestPortsDlg)
	DDX_Control(pDX, IDC_LIST, m_clist);
	DDX_Text(pDX, IDC_NUM, numbyte);

	DDX_Control(pDX, IDC_GetData, m_getdata);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTestPortsDlg, CDialog)
	//{{AFX_MSG_MAP(CTestPortsDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON3, OnButton3)
	ON_BN_CLICKED(IDC_GetData, OnGetData)
	ON_WM_TIMER()
//*******
	ON_MESSAGE(WM_COMM_RXCHAR, OnCommunication)
```

```
	ON_MESSAGE(WM_COMM_CTS_DETECTED, OnCTSDetected)
//********
	ON_BN_CLICKED(IDC_DATA, OnData)
	ON_COMMAND(ID_FILE_RESET, OnFileReset)
	ON_COMMAND(ID_FILE_EXIT, OnFileExit)
	ON_COMMAND(ID_HELP_ABOUTPROGRAM, OnHelpAboutprogram)
	ON_COMMAND(ID_FILE_HISTORY, OnFileHistory)
	ON_BN_CLICKED(IDC_CLEAR, OnClear)
	ON_BN_CLICKED(IDC_EXIT, OnExit)
	ON_BN_CLICKED(IDC_CONNECT, OnConnect)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

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

// CTestPortsDlg message handlers

BOOL CTestPortsDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

/*****************
	hComp = CreateFile( chCommPort,
	GENERIC_READ | GENERIC_WRITE,
	0,
	NULL,
	OPEN_EXISTING,
	0,
	NULL
	);

	dcb.BaudRate = CBR_9600;
	dcb.ByteSize = 8;
	dcb.Parity = NOPARITY;
	dcb.StopBits = ONESTOPBIT;
//*****************/



	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.

	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
```

```
	// Set the icon for this dialog.  The framework does this
automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	SetTimer(ID_CLOCK_TIMER,1000,NULL);   // set timer


	LV_COLUMN column; //Add column report

	column.mask=LVCF_FMT|LVCF_SUBITEM|LVCF_TEXT|LVCF_WIDTH;
	column.fmt=LVCFMT_CENTER;

	column.pszText="Date";
	column.iSubItem=0;
	column.cx=180;
	m_clist.InsertColumn(0,&column);

	column.pszText="Time";
	column.iSubItem=1;
	column.cx=180;
	m_clist.InsertColumn(1,&column);

	column.pszText="Slave 1 (dB)";
	column.iSubItem=2;
	column.cx=100;
	m_clist.InsertColumn(2,&column);

	column.pszText="Slave 2 (dB)";
	column.iSubItem=3;
	column.cx=100;
	m_clist.InsertColumn(3,&column);

	column.pszText="Slave 3 (dB)";
	column.iSubItem=4;
	column.cx=100;
	m_clist.InsertColumn(4,&column);

	column.pszText="Slave 4 (dB)";
	column.iSubItem=5;
	column.cx=100;
	m_clist.InsertColumn(5,&column);

	// TODO: Add extra initialization here

	m_getdata.EnableWindow(FALSE);

		UpdateData(TRUE);
	if (m_Port.InitPort(this,1,9600,'N',8,1))
	{
		m_text1="FOUND";
	    m_Port.StartMonitoring();
	}
	else
		{
			// port not found
			m_text1="NOT FOUND";
		}
		UpdateData(FALSE);
```

```
	return TRUE;  // return TRUE  unless you set the focus to a
control
}

void CTestPortsDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the
code below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.

void CTestPortsDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(),
0);
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

HCURSOR CTestPortsDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}


void CTestPortsDlg::OnButton3()
{
/*/	ReadFile(hComp,&chBuff,8,&nReadPort,NULL);
	m_text1=chBuff;
	UpdateData(FALSE);*/
}

void CTestPortsDlg::OnConnect(){
	UpdateData();
	m_Port.WriteToPort("0");
```

```
	control = '0';
	numbyte = 0;
	UpdateData(FALSE);
}


void CTestPortsDlg::OnGetData()
{
	  UpdateData();

	m_Port.WriteToPort("1");
	control = '1';
	numbyte = 0;
	UpdateData(FALSE);


}


void CTestPortsDlg::OnTimer(UINT nIDEvent)
{
	UpdateData(TRUE);

	CTime curTime=CTime::GetCurrentTime();

	m_sDate2.Format("%d_%d_%d",curTime.GetDay(),
					curTime.GetMonth(),
					curTime.GetYear());


	UpdateData(FALSE);

	CDialog::OnTimer(nIDEvent);
}

void CTestPortsDlg::OnData()
{
	CLookdataDlg lookdatadlg;
	lookdatadlg.DoModal();
}


void CTestPortsDlg::OnFileReset()
{
	UpdateData(TRUE);
	m_text1.Empty();

	strncpy(chBuff,"",50);

	m_time="";
	UpdateData(FALSE);

}

void CTestPortsDlg::OnFileHistory()
{
	CLookdataDlg lookdatadlg;
	lookdatadlg.DoModal();
}

void CTestPortsDlg::OnFileExit()
{
```

```
	if (MessageBox("Do you want to exit program","Sound Level
Indication System",
	MB_ICONQUESTION|MB_YESNO)==IDYES)
		DestroyWindow();
}

void CTestPortsDlg::OnHelpAboutprogram()
{
 MessageBox("Sound Level Indication System :  This project is
intended to construct the sound level indication system. The system
provides  measured data and stores it in a database system via
wireless communication.","About program",MB_OK);
}

void CTestPortsDlg::OnClear()
{
	m_clist.DeleteAllItems();
}

void CTestPortsDlg::OnCancel()
{
	if (MessageBox("Do you want to exit program","Sound Level
Indication System",
	MB_ICONQUESTION|MB_YESNO)==IDYES)
		DestroyWindow();
}

void CTestPortsDlg::OnExit()
{
	if (MessageBox("Do you want to exit program","Sound Level
Indication System",
	MB_ICONQUESTION|MB_YESNO)==IDYES)
		DestroyWindow();
}

void CTestPortsDlg::ReadTime(){

	BYTE hb,lb;
	CString str[6];

	UpdateData();

	hb = 0;
	lb = 0;

	hb = data[5] >> 4;
	lb = data[5] & 15; // 15 = 0000 1111
	sec = (hb * 10) + lb;

	hb = data[4] >> 4;
	lb = data[4] & 15; // 15 = 0000 1111
	min = (hb * 10) + lb;

	if(((data[3]>>6) & 1) == 1){ // 12 hour mode

		ampm = (data[3] >> 5) & 1;
		hb = (data[3] >> 4) & 1; // 1 = 0000 0001;
		lb = data[3] & 15; // 15 = 0000 11111
		hr = (hb * 10) + lb;
		if (ampm == 1) {// pm
			hr = hr + 12;
```

```
            }
            else{ // am
                  // do nothing
            }
      }
      else{ // 24 hour mode

            hb = data[3] >> 4;
            lb = data[3] & 15; // 15 = 0000 1111
            hr = (hb * 10) + lb;
      }
      hb = data[2] >> 4;
      lb = data[2] & 15; // 15 = 0000 11111
      date = (hb * 10) + lb;

      hb = data[1] >> 4;
      lb = data[1] & 15; // 15 = 0000 11111
      mon = (hb * 10) + lb;

      hb = data[0] >> 4;
      lb = data[0] & 15; // 15 = 0000 11111
      year = (hb * 10) + lb;
      year = year + 2000;



      UpdateData(false);

}


LONG CTestPortsDlg::OnCommunication(WPARAM ch, LPARAM port)
{

  int i;
  LV_ITEM item;
  CString tmp;
  unsigned int character;
  char str[80];
  time_t rawtime;
  time_t tmp_time;


  UpdateData(TRUE);

      if (control  == '0'){

            if (ch == '9'){
                  m_getdata.EnableWindow(TRUE);
                  MessageBox("Connected","Connection",MB_OK);
                  numbyte = 0;
            }
            return 0;
      }

      if (control  == '1'){

        UpdateData(TRUE);

            m_getdata.EnableWindow(FALSE);
            data[numbyte] = ch;
```

```
        numbyte++;


    UpdateData(false);
        if (numbyte == 6){
              ReadTime();
              slave_count = 0;
              sec_count = 0;
        }
        if (numbyte == 7){

        }

        if (numbyte < 8){
              return 0;
        }
}

character = ch;

tmp = int_to_str(character);
show(character);

if (slave_count < 4){

        s[slave_count] = r;
}

slave_count++;

if (slave_count == 4){

        slave_count = 0;

        time ( &rawtime );
        timeinfo = localtime ( &rawtime );

        timeinfo->tm_year = year - 1900;
        timeinfo->tm_mon = mon - 1;
        timeinfo->tm_mday = date;
        timeinfo->tm_hour = hr;
        timeinfo->tm_min = min;
        //timeinfo->tm_sec = sec + ((numbyte- 7)* 5);
        timeinfo->tm_sec = sec + (sec_count * 15);
        sec_count++;

        tmp_time = mktime(timeinfo);

        strftime(str,80," %d/%m/%Y ",timeinfo);
        m_sDate = str;

        item.mask=LVIF_TEXT;
        item.iItem=0;
        item.iSubItem=0;
        LPTSTR p = m_sDate.GetBuffer(0);
        item.pszText=p;
        i = m_clist.InsertItem(&item);


        strftime(str,80," %H:%M:%S ",timeinfo);
        m_sTime = str;
```

```
            item.mask=LVIF_TEXT;
            item.iItem=i;
            item.iSubItem=1;
            LPTSTR x = m_sTime.GetBuffer(0);
            item.pszText=x;
            m_clist.SetItem(&item);

            for(i=0;i<4;i++){

                  item.iSubItem=2+i;
                  LPTSTR y = r.GetBuffer(0);
                  item.pszText=y;
                  m_clist.SetItem(&item);

            }

            FILE *stream;

            stream = fopen(m_sDate2+".txt","a");

            if( stream != NULL ){
                  fprintf( stream,"%s %s %s %s %s
%s\n",m_sDate,m_sTime,s[0],s[1],s[2],s[3]);
                  fclose( stream );
            }
      }

  UpdateData(FALSE);
  return 0;
}

LONG CTestPortsDlg::OnCTSDetected(WPARAM, LPARAM port)
{


  return 0;
}
```

National Semiconductor

June 1999

ADC0820
8-Bit High Speed µP Compatible A/D Converter with Track/Hold Function

General Description

By using a half-flash conversion technique, the 8-bit ADC0820 CMOS A/D offers a 1.5 µs conversion time and dissipates only 75 mW of power. The half-flash technique consists of 32 comparators, a most significant 4-bit ADC and a least significant 4-bit ADC.

The input to the ADC0820 is tracked and held by the input sampling circuitry eliminating the need for an external sample-and-hold for signals moving at less than 100 mV/µs.

For ease of interface to microprocessors, the ADC0820 has been designed to appear as a memory location or I/O port without the need for external interfacing logic.

Key Specifications

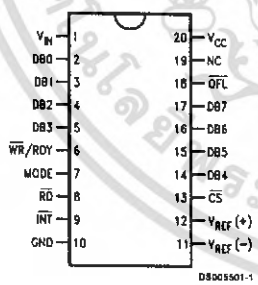
- Resolution — 8 Bits
- Conversion Time — 2.5 µs Max (RD Mode); 1.5 µs Max (WR-RD Mode)
- Low Power — 75 mW Max
- Total Unadjusted Error — ±½ LSB and ± 1 LSB

Features

- Built-in track-and-hold function
- No missing codes
- No external clocking
- Single supply — 5 V_{CC}
- Easy interface to all microprocessors, or operates stand-alone
- Latched TRI-STATE® output
- Logic inputs and outputs meet both MOS and T^2L voltage level specifications
- Operates ratiometrically or with any reference value equal to or less than V_{CC}
- 0V to 5V analog input voltage range with single 5V supply
- No zero or full-scale adjust required
- Overflow output available for cascading
- 0.3" standard width 20-pin DIP
- 20-pin molded chip carrier package
- 20-pin small outline package
- 20-pin shrink small outline package (SSOP)

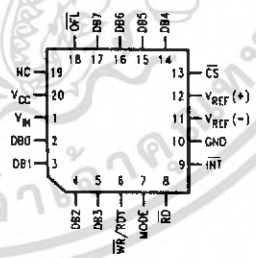
Connection and Functional Diagrams

Dual-In-Line, Small Outline and SSOP Packages



Top View

Molded Chip Carrier Package



TRI-STATE® is a registered trademark of National Semiconductor Corporation.

Timing Diagrams (Continued)

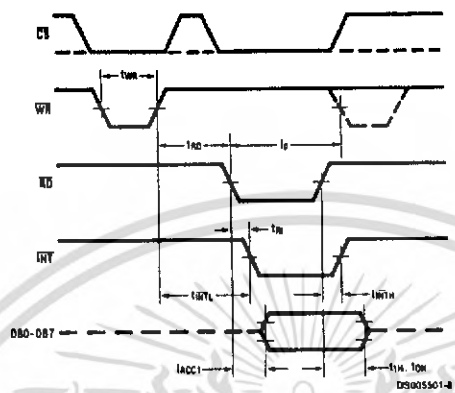


FIGURE 3. WR-RD Mode (Pin 7 is High and $t_{RD} < t_I$)

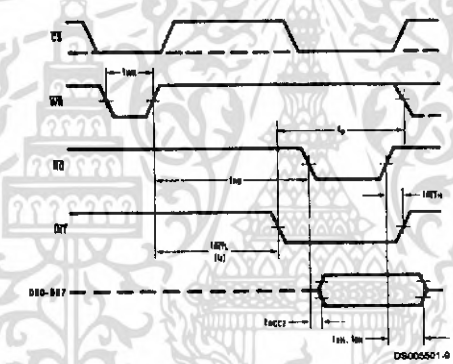


FIGURE 4. WR-RD Mode (Pin 7 is High and $t_{RD} > t_I$)

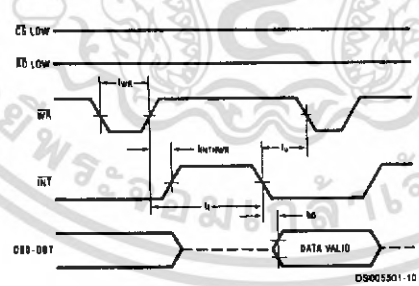


FIGURE 5. WR-RD Mode (Pin 7 is High)
Stand-Alone Operation

Description of Pin Functions

Pin	Name	Function
1	V_{IN}	Analog input; range $=GND \le V_{IN} \le V_{CC}$
2	DB0	TRI-STATE data output—bit 0 (LSB)
3	DB1	TRI-STATE data output—bit 1
4	DB2	TRI-STATE data output—bit 2
5	DB3	TRI-STATE data output—bit 3
6	$\overline{WR}$ /RDY	**WR-RD Mode** $\overline{WR}$: With $\overline{CS}$ low, the conversion is started on the falling edge of $\overline{WR}$. Approximately 800 ns (the preset internal time out, t_i) after the $\overline{WR}$ rising edge, the result of the conversion will be strobed into the output latch, provided that $\overline{RD}$ does not occur prior to this time out (see *Figures 3, 4*). **RD Mode** RDY: This is an open drain output (no internal pull-up device). RDY will go low after the falling edge of $\overline{CS}$; RDY will go TRI-STATE when the result of the conversion is strobed into the output latch. It is used to simplify the interface to a microprocessor system (see *Figure 2*).
7	Mode	Mode: Mode selection input—it is internally tied to GND through a 50 µA current source. RD Mode: When mode is low WR-RD Mode: When mode is high
8	$\overline{RD}$	**WR-RD Mode** With $\overline{CS}$ low, the TRI-STATE data outputs (DB0-DB7) will be activated when $\overline{RD}$ goes low (see *Figure 5*). $\overline{RD}$ can also be used to increase the speed of the converter by reading data prior to the preset internal time out (t_i, ~800 ns). If this is done, the data result transferred to output latch is latched after the falling edge of the $\overline{RD}$ (see *Figures 3, 4*). **RD Mode** With $\overline{CS}$ low, the conversion will start with $\overline{RD}$ going low, also $\overline{RD}$ will enable the TRI-STATE data outputs at the completion of the conversion. RDY going TRI-STATE and $\overline{INT}$ going low indicates the completion of the conversion (see *Figure 2*).
9	$\overline{INT}$	**WR-RD Mode** $\overline{INT}$ going low indicates that the conversion is completed and the data result is in the output latch. $\overline{INT}$ will go low, ~800 ns (the preset internal time out, t_i) after the rising edge of $\overline{WR}$ (see *Figure 4*); or $\overline{INT}$ will go low after the falling edge of $\overline{RD}$, if $\overline{RD}$ goes low prior to the 800 ns time out (see *Figure 3*). $\overline{INT}$ is reset by the rising edge of $\overline{RD}$ or $\overline{CS}$ (see *Figures 3, 4*). **RD Mode** $\overline{INT}$ going low indicates that the conversion is completed and the data result is in the output latch. $\overline{INT}$ is reset by the rising edge of $\overline{RD}$ or $\overline{CS}$ (see *Figure 2*).
10	GND	Ground
11	$V_{REF}(-)$	The bottom of resistor ladder, voltage range: $GND \le V_{REF}(-) \le V_{REF}(+)$ (Note 5)
12	$V_{REF}(+)$	The top of resistor ladder, voltage range: $V_{REF}(-) \le V_{REF}(+) \le V_{CC}$ (Note 5)
13	$\overline{CS}$	$\overline{CS}$ must be low in order for the $\overline{RD}$ or $\overline{WR}$ to be recognized by the converter.
14	DB4	TRI-STATE data output—bit 4
15	DB5	TRI-STATE data output—bit 5
16	DB6	TRI-STATE data output—bit 6
17	DB7	TRI-STATE data output—bit 7 (MSB)
18	$\overline{OFL}$	Overflow output—If the analog input is higher than the $V_{REF}(+)$, $\overline{OFL}$ will be low at the end of conversion. It can be used to cascade 2 or more devices to have more resolution (9, 10-bit). This output is always active and does not go into TRI-STATE as DB0–DB7 do.
19	NC	No connection
20	V_{CC}	Power supply voltage

1.0 Functional Description

1.1 GENERAL OPERATION

The ADC0820 uses two 4-bit flash A/D converters to make an 8-bit measurement (*Figure 1*). Each flash ADC is made up of 15 comparators which compare the unknown input to a reference ladder to get a 4-bit result. To take a full 8-bit reading, one flash conversion is done to provide the 4 most significant data bits (via the MS flash ADC). Driven by the 4 MSBs, an internal DAC recreates an analog approximation of the input voltage. This analog signal is then subtracted from the input, and the difference voltage is converted by a second 4-bit flash ADC (the LS ADC), providing the 4 least significant bits of the output data word.

The internal DAC is actually a subsection of the MS flash converter. This is accomplished by using the same resistor

DATA SHEET

PCF8574
Remote 8-bit I/O expander for I²C-bus

Product specification
Supersedes data of September 1994
File under Integrated Circuits, IC12

1997 Apr 02

Philips Semiconductors

PHILIPS

Remote 8-bit I/O expander for I²C-bus PCF8574

1 FEATURES

- Operating supply voltage 2.5 to 6 V
- Low standby current consumption of 10 μA maximum
- I²C to parallel port expander
- Open-drain interrupt output
- 8-bit remote I/O port for the I²C-bus
- Compatible with most microcontrollers
- Latched outputs with high current drive capability for directly driving LEDs
- Address by 3 hardware address pins for use of up to 8 devices (up to 16 with PCF8574A)
- DIP16, or space-saving SO16 or SSOP20 packages.

2 GENERAL DESCRIPTION

The PCF8574 is a silicon CMOS circuit. It provides general purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I²C).

The device consists of an 8-bit quasi-bidirectional port and an I²C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line ($\overline{INT}$) which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I²C-bus. This means that the PCF8574 can remain a simple slave device.

The PCF8574 and PCF8574A versions differ only in their slave address as shown in Fig.9.

3 ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
PCF8574P; PCF8574AP	DIP16	plastic dual in-line package; 16 leads (300 mil)	SOT38-1
PCF8574T; PCF8574AT	SO16	plastic small outline package; 16 leads; body width 7.5 mm	SOT162-1
PCF8574TS	SSOP20	plastic shrink small outline package; 20 leads; body width 4.4 mm	SOT266-1

4 BLOCK DIAGRAM

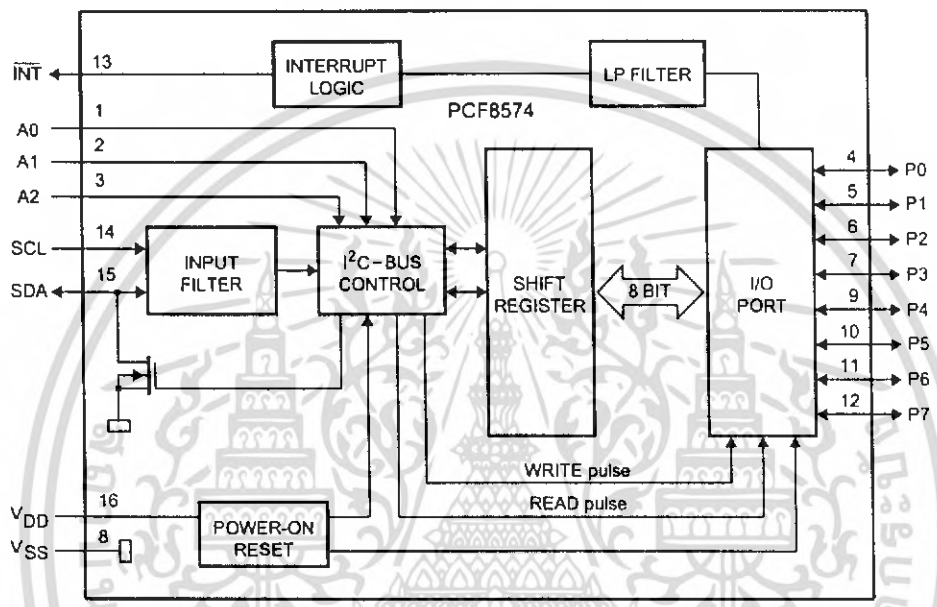


Fig.1 Block diagram (SOT38-1 and SOT162-1).

5 PINNING

SYMBOL	PIN		DESCRIPTION
	DIP16; SO16	SSOP20	
A0	1	6	address input 0
A1	2	7	address input 1
A2	3	9	address input 2
P0	4	10	quasi-bidirectional I/O 0
P1	5	11	quasi-bidirectional I/O 1
P2	6	12	quasi-bidirectional I/O 2
P3	7	14	quasi-bidirectional I/O 3
V_{SS}	8	15	supply ground
P4	9	16	quasi-bidirectional I/O 4
P5	10	17	quasi-bidirectional I/O 5
P6	11	19	quasi-bidirectional I/O 6
P7	12	20	quasi-bidirectional I/O 7
$\overline{INT}$	13	1	interrupt output (active LOW)
SCL	14	2	serial clock line
SDA	15	4	serial data line
V_{DD}	16	5	supply voltage
n.c.	–	3	not connected
n.c.	–	8	not connected
n.c.	–	13	not connected
n.c.	–	18	not connected

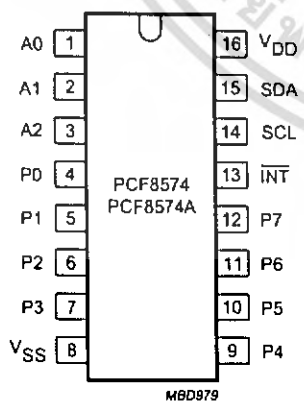


Fig.2 Pin configuration (DIP16; SO16).

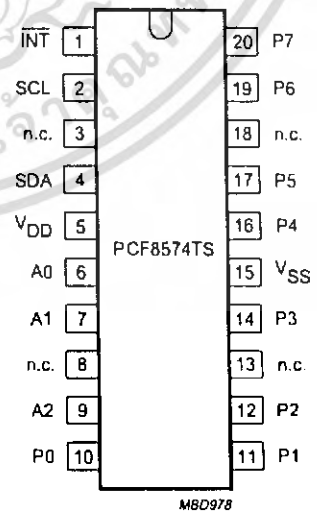


Fig.3 Pin configuration (SSOP20).

DALLAS SEMICONDUCTOR

DS1307
64 x 8 Serial Real-Time Clock

www.maxim-ic.com

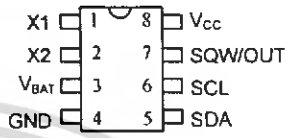
FEATURES

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap-year compensation valid up to 2100
- 56-byte, battery-backed, nonvolatile (NV) RAM for data storage
- Two-wire serial interface
- Programmable squarewave output signal
- Automatic power-fail detect and switch circuitry
- Consumes less than 500nA in battery backup mode with oscillator running
- Optional industrial temperature range: -40°C to +85°C
- Available in 8-pin DIP or SOIC
- Underwriters Laboratory (UL) recognized

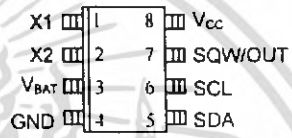
ORDERING INFORMATION

DS1307	8-Pin DIP (300-mil)
DS1307Z	8-Pin SOIC (150-mil)
DS1307N	8-Pin DIP (Industrial)
DS1307ZN	8-Pin SOIC (Industrial)

PIN ASSIGNMENT



DS1307 8-Pin DIP (300-mil)



DS1307 8-Pin SOIC (150-mil)

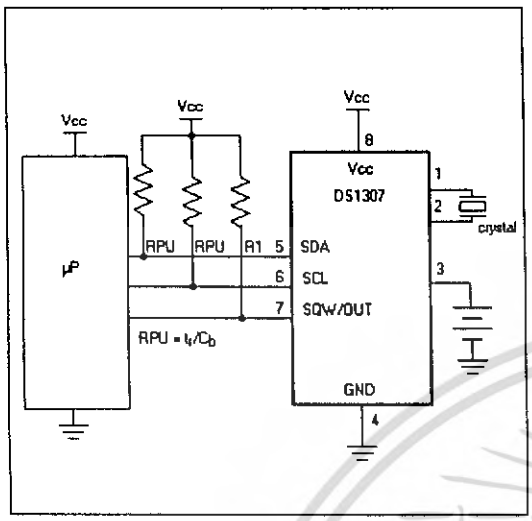
PIN DESCRIPTION

V_{CC}	- Primary Power Supply
X1, X2	- 32.768kHz Crystal Connection
V_{BAT}	- +3V Battery Input
GND	- Ground
SDA	- Serial Data
SCL	- Serial Clock
SQW/OUT	- Square Wave/Output Driver

DESCRIPTION

The DS1307 Serial Real-Time Clock is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit that detects power failures and automatically switches to the battery supply.

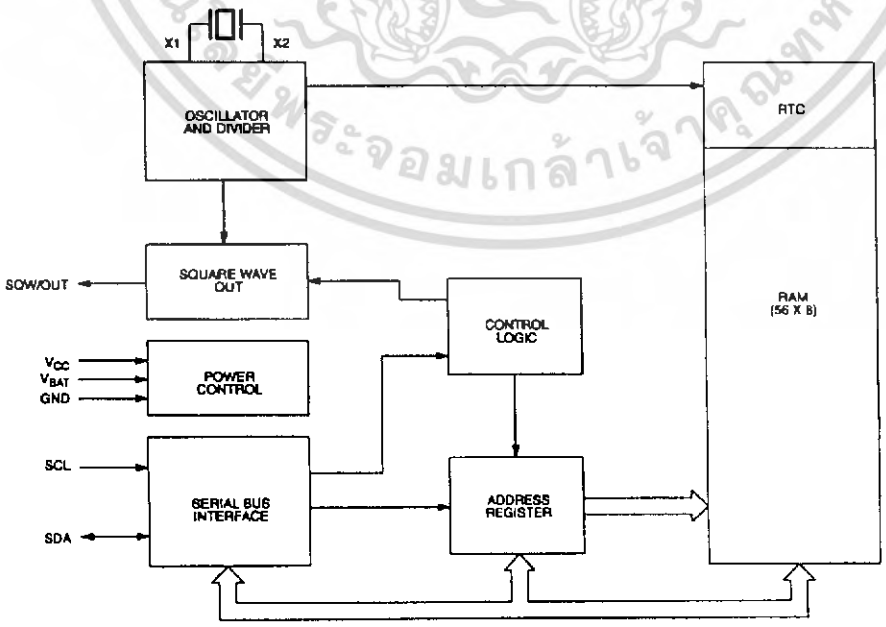
TYPICAL OPERATING CIRCUIT



OPERATION

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT} the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When V_{CC} falls below V_{BAT} the device switches into a low-current battery backup mode. Upon power-up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT} + 0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the serial RTC.

DS1307 BLOCK DIAGRAM Figure 1



CLOCK ACCURACY

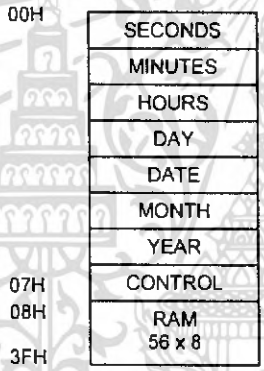
The accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacitive load of the oscillator circuit and the capacitive load for which the crystal was trimmed. Additional error will be added by crystal frequency drift caused by temperature shifts. External circuit noise coupled into the oscillator circuit may result in the clock running fast. See Application Note 58, "Crystal Considerations with Dallas Real-Time Clocks" for detailed information.

Please review Application Note 95, "Interfacing the DS1307 with a 8051-Compatible Microcontroller" for additional information.

RTC AND RAM ADDRESS MAP

The address map for the RTC and RAM registers of the DS1307 is shown in Figure 2. The RTC registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multi-byte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

DS1307 ADDRESS MAP Figure 2



CLOCK AND CALENDAR

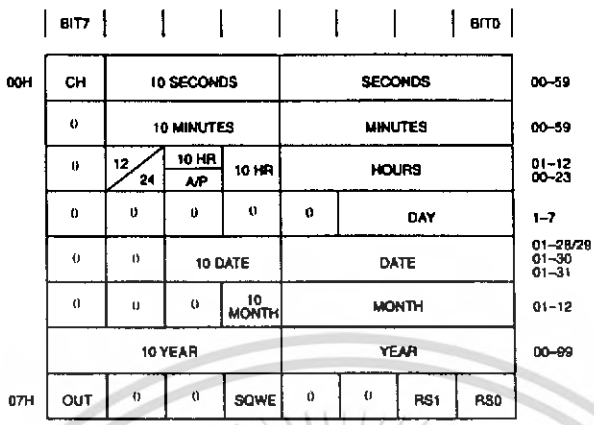
The time and calendar information is obtained by reading the appropriate register bytes. The RTC registers are illustrated in Figure 3. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format. Bit 7 of register 0 is the clock halt (CH) bit. When this bit is set to a 1, the oscillator is disabled. When cleared to a 0, the oscillator is enabled.

Please note that the initial power-on state of all registers is not defined. Therefore, it is important to enable the oscillator (CH bit = 0) during initial configuration.

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10 hour bit (20-23 hours).

On a 2-wire START, the current time is transferred to a second set of registers. The time information is read from these secondary registers, while the clock may continue to run. This eliminates the need to re-read the registers in case of an update of the main registers during a read.

DS1307 TIMEKEEPER REGISTERS Figure 3



CONTROL REGISTER

The DS1307 control register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

OUT (Output control): This bit controls the output level of the SQW/OUT pin when the square wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0.

SQWE (Square Wave Enable): This bit, when set to a logic 1, will enable the oscillator output. The frequency of the square wave output depends upon the value of the RS0 and RS1 bits. With the square wave output set to 1Hz, the clock registers update on the falling edge of the square wave.

RS (Rate Select): These bits control the frequency of the square wave output when the square wave output has been enabled. Table 1 lists the square wave frequencies that can be selected with the RS bits.

SQUAREWAVE OUTPUT FREQUENCY Table 1

RS1	RS0	SQW OUTPUT FREQUENCY
0	0	1Hz
0	1	4.096kHz
1	0	8.192kHz
1	1	32.768kHz

CYPRESS

CY62256

256K (32K x 8) Static RAM

Features

- High speed: 55 ns and 70 ns
- Voltage range: 4.5V–5.5V operation
- Low active power (70 ns, LL version)
  — 275 mW (max.)
- Low standby power (70 ns, LL version)
  — 28 µW (max.)
- Easy memory expansion with $\overline{CE}$ and $\overline{OE}$ features
- TTL-compatible inputs and outputs
- Automatic power-down when deselected
- CMOS for optimum speed/power
- Package available in a standard 450-mil-wide (300-mil body width) 28-lead narrow SOIC, 28-lead TSOP-1, 28-lead reverse TSOP-1, and 600-mil 28-lead PDIP packages

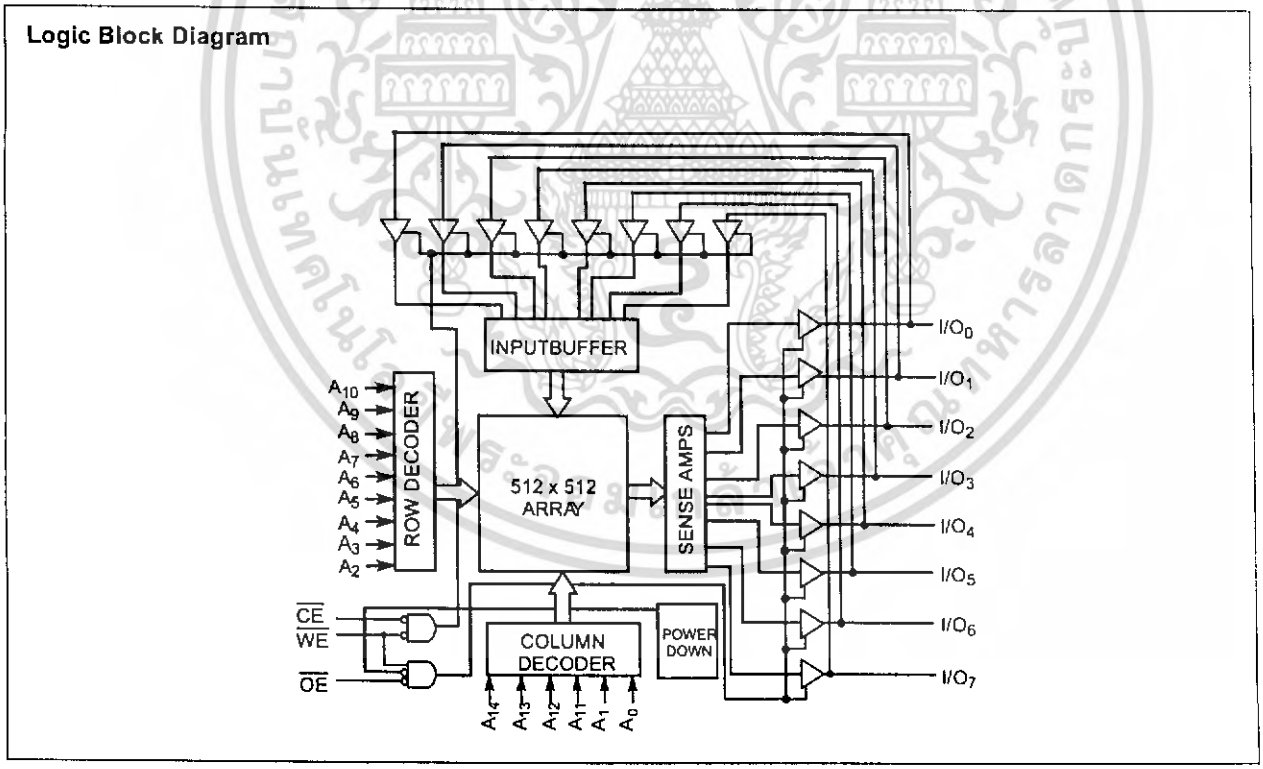
Functional Description[1]

The CY62256 is a high-performance CMOS static RAM organized as 32K words by 8 bits. Easy memory expansion is provided by an active LOW chip enable ($\overline{CE}$) and active LOW output enable ($\overline{OE}$) and three-state drivers. This device has an automatic power-down feature, reducing the power consumption by 99.9% when deselected.

An active LOW write enable signal ($\overline{WE}$) controls the writing/reading operation of the memory. When $\overline{CE}$ and $\overline{WE}$ inputs are both LOW, data on the eight data input/output pins (I/O_0 through I/O_7) is written into the memory location addressed by the address present on the address pins (A_0 through A_{14}). Reading the device is accomplished by selecting the device and enabling the outputs, $\overline{CE}$ and $\overline{OE}$ active LOW, while $\overline{WE}$ remains inactive or HIGH. Under these conditions, the contents of the location addressed by the information on address pins are present on the eight data input/output pins.

The input/output pins remain in a high-impedance state unless the chip is selected, outputs are enabled, and write enable ($\overline{WE}$) is HIGH.

Logic Block Diagram

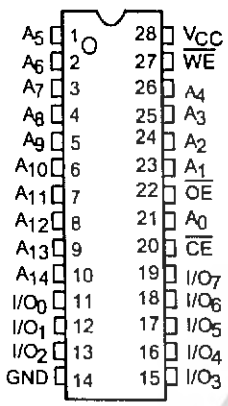


Note:

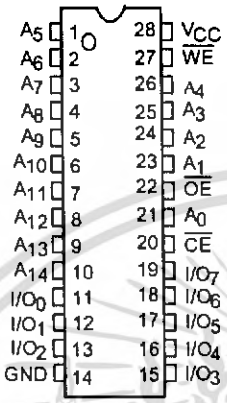
1. For best practice recommendations, please refer to the Cypress application note "System Design Guidelines" on http://www.cypress.com.

Pin Configurations

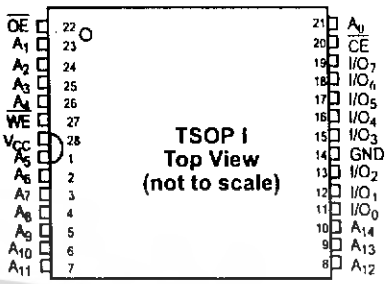
Narrow SOIC
Top View



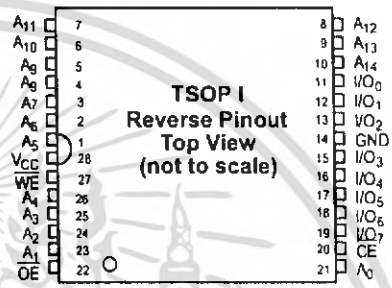
DIP
Top View



TSOP I
Top View
(not to scale)



TSOP I
Reverse Pinout
Top View
(not to scale)



Maximum Ratings

(Above which the useful life may be impaired. For user guidelines, not tested.)

Storage Temperature ..... –65°C to +150°C

Ambient Temperature with Power Applied ..... 0°C to +70°C

Supply Voltage to Ground Potential (Pin 28 to Pin 14) ..... –0.5V to +7.0V

DC Voltage Applied to Outputs in High-Z State[2] ..... –0.5V to V_{CC} + 0.5V

DC Input Voltage[2] ..... –0.5V to V_{CC} + 0.5V

Output Current into Outputs (LOW) ..... 20 mA

Static Discharge Voltage ..... > 2001V (per MIL-STD-883, Method 3015)

Latch-up Current ..... > 200 mA

Operating Range

Range	Ambient Temperature	V_{CC}
Commercial	0°C to +70°C	5V ± 10%
Industrial	–40°C to +85°C	5V ± 10%

Electrical Characteristics Over the Operating Range

Parameter	Description	Test Conditions		CY62256–55 Min.	CY62256–55 Typ.[3]	CY62256–55 Max.	CY62256–70 Min.	CY62256–70 Typ.[3]	CY62256–70 Max.	Unit
V_{OH}	Output HIGH Voltage	V_{CC} = Min., I_{OH} = –1.0 mA		2.4			2.4			V
V_{OL}	Output LOW Voltage	V_{CC} = Min., I_{OL} = 2.1 mA				0.4			0.4	V
V_{IH}	Input HIGH Voltage			2.2		V_{CC} +0.5V	2.2		V_{CC} +0.5V	V
V_{IL}	Input LOW Voltage			–0.5		0.8	–0.5		0.8	V
I_{IX}	Input Leakage Current	GND ≤ V_I ≤ V_{CC}		–0.5		+0.5	–0.5		+0.5	µA
I_{OZ}	Output Leakage Current	GND ≤ V_O ≤ V_{CC}, Output Disabled		–0.5		+0.5	–0.5		+0.5	µA
I_{CC}	V_{CC} Operating Supply Current	V_{CC} = Max., I_{OUT} = 0 mA, f = f_{MAX} = $1/t_{RC}$			28	55		28	55	mA
			L		25	50		25	50	mA
			LL		25	50		25	50	mA
I_{SB1}	Automatic CE Power-down Current—TTL Inputs	Max. V_{CC}, $\overline{CE}$ ≥ V_{IH}, V_{IN} ≥ V_{IH} or V_{IN} ≤ V_{IL}, f = f_{MAX}			0.5	2		0.5	2	mA
			L		0.4	0.6		0.4	0.6	mA
			LL		0.3	0.5		0.3	0.5	mA

Notes:

2. V_{IL} (min.) = –2.0V for pulse durations of less than 20 ns.
3. Typical specifications are the mean values measured over a large sample size across normal production process variations and are taken at nominal conditions (T_A = 25°C, V_{CC}). Parameters are guaranteed by design and characterization, and not 100% tested.

Features

- Compatible with MCS-51™ Products
- 2K Bytes of Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- 2.7V to 6V Operating Range
- Fully Static Operation: 0 Hz to 24 MHz
- Two-Level Program Memory Lock
- 128 x 8-Bit Internal RAM
- 15 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial UART Channel
- Direct LED Drive Outputs
- On-Chip Analog Comparator
- Low Power Idle and Power Down Modes

8-Bit Microcontroller with 2K Bytes Flash

AT89C2051

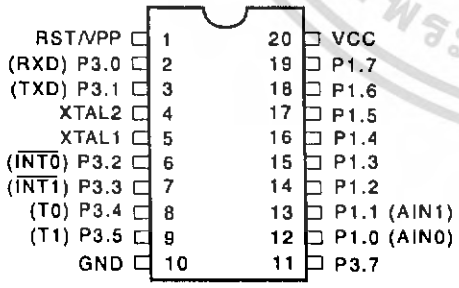
Description

The AT89C2051 is a low-voltage, high-performance CMOS 8-bit microcomputer with 2K Bytes 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. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C2051 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

The AT89C2051 provides the following standard features: 2K Bytes of Flash, 128 bytes of RAM, 15 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, a precision analog comparator, on-chip oscillator and clock circuitry. In addition, the AT89C2051 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 Configuration

PDIP/SOIC



0368D-B–12/97

Features

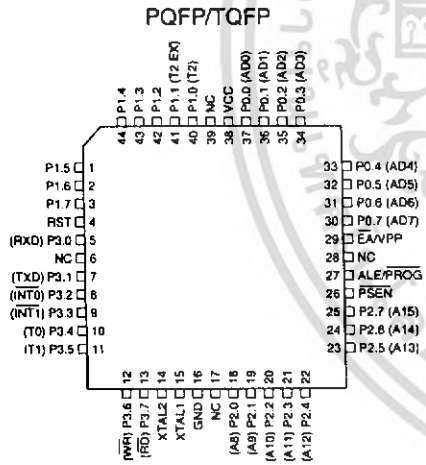
- Compatible with MCS-51™ Products
- 8K Bytes of In-System Reprogrammable Flash Memory
- Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Eight Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

Description

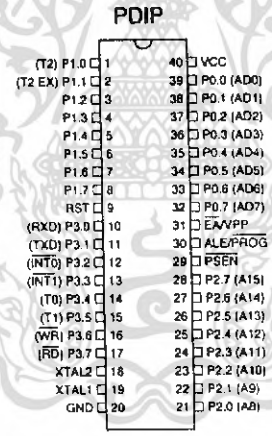
The AT89C52 is a low-power, high-performance CMOS 8-bit microcomputer with 8K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 80C51 and 80C52 instruction 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 AT89C52 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

Pin Configurations

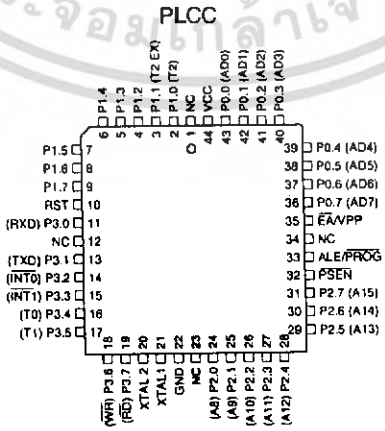
PQFP/TQFP



PDIP



PLCC



8-bit Microcontroller with 8K Bytes Flash

AT89C52

Rev. 0313H–02/00

19-4323; Rev 14; 8/04

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multidrop RS-232 Networks

AutoShutdown and UCSP are trademarks of Maxim Integrated Products, Inc.

Next-Generation Device Features

- **For Low-Voltage, Integrated ESD Applications MAX3222E/MAX3232E/MAX3237E/MAX3241E/ MAX3246E: +3.0V to +5.5V, Low-Power, Up to 1Mbps, True RS-232 Transceivers Using Four 0.1µF External Capacitors (MAX3246E Available in a UCSP™ Package)**
- **For Low-Cost Applications MAX221E: ±15kV ESD-Protected, +5V, 1µA, Single RS-232 Transceiver with AutoShutdown™**

Ordering Information

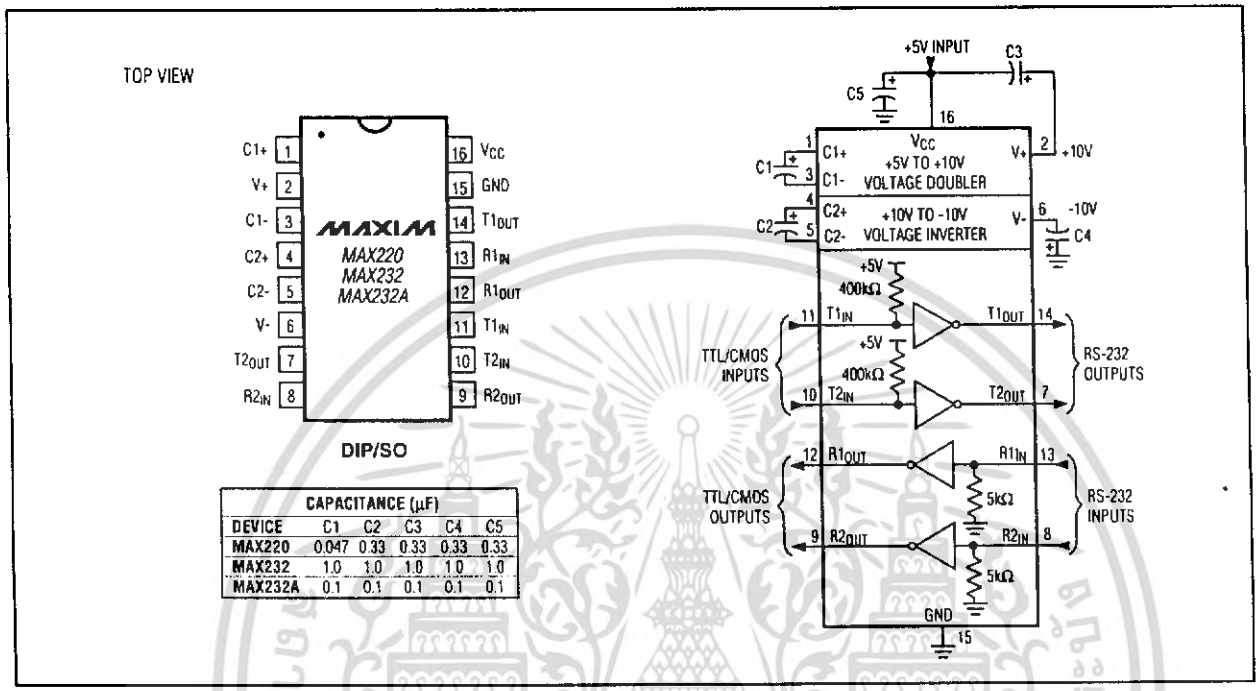
PART	TEMP RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.047/0.33	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies, same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies, single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package



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

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

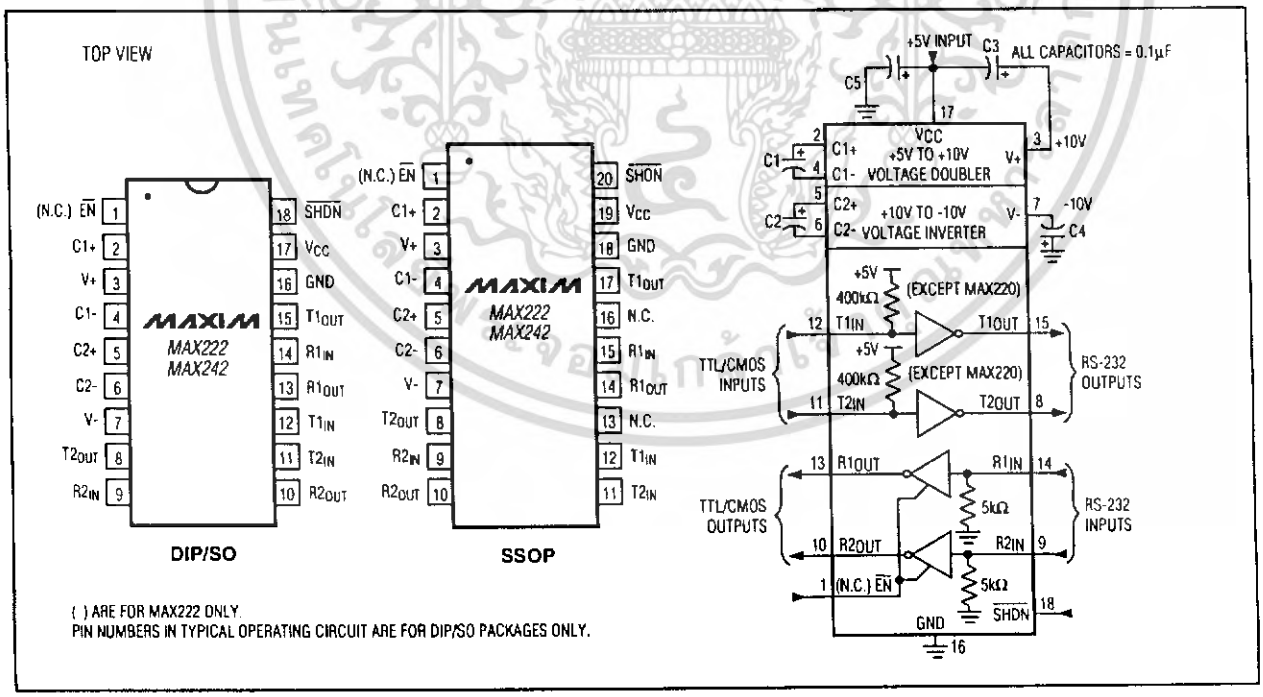


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit