เครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมย่าน C และ KU

Amplitude Scintillation Recorder of C and KU band Satellite Signal

โดย

นาย ปิยะ ถิรพันธุ์เมธี

นาย สามารถ ตันติถาวรวัฒน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

สาขาเทคโนโลยีอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

หัวข้อปริญญานิพนธ์ AMPLITUDE SCINTILLATION RECORDER OF C AND KU BAND SATELLITE SIGNAL

จัดทำโดย	นาย ปิยะ	ถิรพันธุ์เมธี
	นายสามารถ	ตันติถาวรวัฒน์
ภาควิชา	เทคนิคอุตสาหกรรม	
อาจารย์ที่ปรึกษา	อาจารย์ชวลิต	เบญจางคประเสริฐ
	อาจารย์อรลาภ	แสงอรุณ

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

_____ อาจารย์ที่ปรึกษา
(_____)

_____ กรรมการ
(_____)

_____ กรรมการ
(_____)

_____ กรรมการ
(_____)

เครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมย่าน C และ KU

Amplitude Scintillation Recorder of C and KU band Satellite Signal

โดย นาย ปิยะ ถิรพันธุ์เมธี
นาย สามารถ ตันติถาวรวัฒน์
อาจารย์ที่ปรึกษา อ.ชวลิต เบญจางคประเสริฐ
อ.อรลาภ แสงอรุณ

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการออกแบบและพัฒนานำไมโครคอมพิวเตอร์มาประยุกต์ใช้ในการเก็บบันทึกการเปลี่ยนแปลงของระดับสัญญาณจากดาวเทียมลงบนหน่วยความจำแทนเครื่องบันทึกแบบปากกา (Pen Recorder) ซึ่งบันทึกบนกระดาษ (Strip Chart) การติดต่อระหว่างสัญญาณภายนอก กับคอมพิวเตอร์เป็นแบบอนุกรม โดยโครงสร้างของระบบประกอบด้วย ส่วนของฮาร์ทแวร์และซอฟแวร์ในส่วนของฮาร์ทแวร์คือ ส่วนอุปกรณ์ที่ทำหน้าที่แปลงสัญญาณอนาล็อกเป็นดิจิตอล และส่วนประมวลผลคือไมโครคอนโทลเลอร์ และส่วนที่ทำหน้าที่ขยายสัญญาณจาก เอ จี ซี ส่วนของซอฟแวร์คือส่วนของภาษาแอสเซมบลี้ของ MCS-51 และโปรแกรมภาษาปาสคาล ที่ใช้ในการควบคุม การบันทึกและแสดงผล ตลอดจนการหาค่าส่วนเบี่ยงเบนมาตรฐานและค่าต่ำสุดสูงสุดของสัญญาณ

ผลการทดลองศึกษาและพัฒนาสามารถนำไมโครคอมพิวเตอร์มาประยุกต์ใช้เก็บบันทึกสัญญาณได้ทั้งรูปร่างของสัญญาณที่เปลี่ยนแปลงตามเวลาจริง และบันทึกเป็นตัวเลขบอกระดับสัญญาณที่เปลี่ยนแปลงด้วยอัตราเร็ว 20 จุดต่อวินาที ทั้งสามารถนำข้อมูลที่บันทึกได้มาคำนวณทางสถิติหาค่าความเบี่ยงเบนมาตรฐานและค่าต่ำสุดสูงสุดของระดับสัญญาณเพื่อประโยชน์ในการวิเคราะห์ปรากฏการณ์การเปลี่ยนแปลงอย่างกระทันหันของสัญญาณดาวเทียมเนื่องจากชั้นบรรยากาศโทรโพสเฟียร์ได้ผลถูกต้อง

PROJECT REPORT TITLE : AMPLITUDE SCINTILLATION RECORD OF C AND KU BAND SATELLITE SIGNAL

NAME : MR. PIYA THIRAPANMETHEE
: MR. SAMART TONTITHAVORNWAT

PROJECT REPORT ADVISOR : MR.CHAOVALIT BENJARKPRASAT
: MS.ORNLARP SIANGAROON

DEPARTMENT OF : INDUSTRIAL TECHNOLOGY

ACADEMIC YEAR : 1997

ABSTRACT

This project proposed the designation and development of recording techniques in satellite signal. Use the microcomputer in stead of per recorder. The communication between computer and external signal is series interface. The measurement apparatus consist of the analog to digital converts , microcontroller and amplifier. CPU for controlling this unit using assembly language. The graphic mode and processing using Pascal language. The real time signal can be display and evaluated. This experiment shows the result from computer in reliable.

กิตติกรรมประกาศ

ในการทำปริญญานิพนธ์ เรื่องเครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมย่าน C และ KU ต้องขอขอบคุณท่านอาจารย์ชวลิต เบญจางคประเสริฐ และ ท่านอาจารย์อรลาภ แสงอรุณ ที่ได้ให้คำแนะนำที่เป็นประโยชน์และจัดหาเครื่องมือต่างๆเพื่อใช้ในการจัดทำปริญญานิพนธ์นี้ รวมทั้งคณาจารย์ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คณะผู้จัดทำ

สารบัญ

ภาคผนวก

คู่มือการใช้งานเครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียม

การใช้งานโปรแกรมบนเครื่องคอมพิวเตอร์

โปรแกรมภาษาแอสเซมบลี้ไมโครคอนโทรเลอร์

โปรแกรมภาษาปาสคาล

ไอซี ADC 0804 A/D Converter

ไอซี ds 1202 Real Time Clock

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

เริ่มจากการที่ในปัจจุบันนี้การสื่อสารผ่านดาวเทียมเป็นเทคโนโลยีที่มีความนิยมใช้กันอย่างแพร่หลายไม่ว่าจะเป็นเพื่อประโยชน์ในทางธุรกิจหรือเพื่อการบันเทิง แต่ปัญหาหนึ่งของการสื่อสารผ่านดาวเทียมก็คือการที่สภาวะแวดล้อมต่างๆจะมีผลกระทบต่อระดับของสัญญาณทำให้สัญญาณที่ภาครับอาจเกิดการผิดพลาดขึ้นได้ง่าย ในปัจจุบันเครื่องมือที่ใช้ในการบันทึกการเปลี่ยนแปลงของระดับสัญญาณดาวเทียมนี้เป็นการบันทึกลงในกระดาษทำให้มีความสิ้นเปลืองกระดาษที่ใช้ในการบันทึกเป็นอันมากอีกทั้งการอ่านค่าจากกระดาษที่บันทึกไว้อาจมีความคลาดเคลื่อนได้ง่าย

1.2 วัตถุประสงค์

ในการทำโครงงานนี้จึงเป็นการสร้างเครื่องมือขึ้นมาเพื่อทำการบันทึกการเปลี่ยนแปลงของระดับสัญญาณที่เปลี่ยนไปเมื่อสภาวะแวดล้อมต่างๆเปลี่ยนไปโดยเป็นการสร้างเครื่องมือขึ้นมาทดแทนเครื่องแบบเก่าที่เป็นการบันทึกลงกระดาษ ซึ่งจะทำให้เป็นการประหยัดและสามารถเก็บข้อมูลได้ละเอียดกว่าการบันทึกแบบเก่ารวมทั้งยังสามารถประหยัดเวลาที่ใช้ในการวิเคราะห์ข้อมูลได้อีกด้วย

1.3 ขอบเขตของปริญญานิพนธ์

เครื่องมือที่สร้างขึ้นนี้เป็นการสร้างเครื่องบันทึกข้อมูลขึ้นมาแทนเครื่องบันทึกด้วยกระดาษโดยจะมีคุณสมบัติคือจะสามารถทำการบันทึกลงในตัวเครื่องเองโดยไม่ต้องต่อกับเครื่องคอมพิวเตอร์ก็ได้โดยค่าที่ทำการบันทึกการบันทึกนี้จะเป็นค่าที่เก็บอยู่ในรูปของตัวเลขของสัญญาณแรงดันจากภาค AGC ของเครื่องรับสัญญาณดาวเทียม ซึ่งจากการใช้แรงดันจากวงจร AGC นี้ทำให้สามารถทำการบันทึกข้อมูลได้ไม่ว่าสัญญาณที่ต้องการบันทึกนี้จะเป็นสัญญาณ C BAND หรือสัญญาณ KU BAND โดยทีในการบันทึกนี้จะมีการสุ่มสัญญาณขึ้นมาทำการบันทึกอยู่ที่ 20 ครั้งต่อ 1 วินาที ซึ่งจะทำให้สามารถบันทึกสัญญาณโดยไม่ต้องต่อร่วมกับเครื่องคอมพิวเตอร์ได้เป็นเวลานานประมาณ 30 นาที (ในที่นี้มี RAM อยู่ 32 กิโลไบท์ สามารถขยายได้สูงสุดเป็น 64 กิโลไบท์) แล้วทำการส่งข้อมูลที่บันทึกได้นี้ไปแสดงผลในรูปของกราฟแสดงการเปลี่ยนแปลงของสัญญาณต่อเวลาที่เปลี่ยนไปบนเครื่องคอมพิวเตอร์การที่นำไปทำการแสดงผลบนเครื่องคอมพิวเตอร์นี้ จะทำให้สามารถอ่านข้อมูลออกมาเป็นค่าที่แน่นอนหรือสามารถทำการบันทึกข้อมูลและแสดงผลการเปลี่ยนแปลงของสัญญาณเป็นรูปกราฟในลักษณะเป็น Real Time บนเครื่องคอมพิวเตอร์ได้ นอกจากนี้แล้วข้อมูลที่

บันทึกไว้จะถูกเก็บอยู่ในรูปของแฟ้มข้อมูลบนคอมพิวเตอร์สามารถทำการเปิดดูได้ตลอดเวลาและนำมาทำการวิเคราะห์หาค่าต่างๆได้โดยง่าย ซึ่งในโปรแกรมที่เขียนขึ้นเพื่อควบคุมการทำงานนี้จะมีการวิเคราะห์การกระจายแบบ STD ของสัญญาณที่ทำการบันทึกเอาไว้ และในส่วนของข้อมูลที่เราไม่ต้องการใช้ก็สามารถทำการลบทิ้งได้ทำให้ไม่เป็นการสิ้นเปลืองเหมือนกับการบันทึกด้วยกระดาษ

จากที่กล่าวมานี้ จึงได้ทำการรวบรวมข้อมูลรายละเอียดทั้งทฤษฎีวงจรและเทคนิคต่างๆในการสร้างเครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมโดยมีรายละเอียดดังที่กล่าวในบทต่อไป

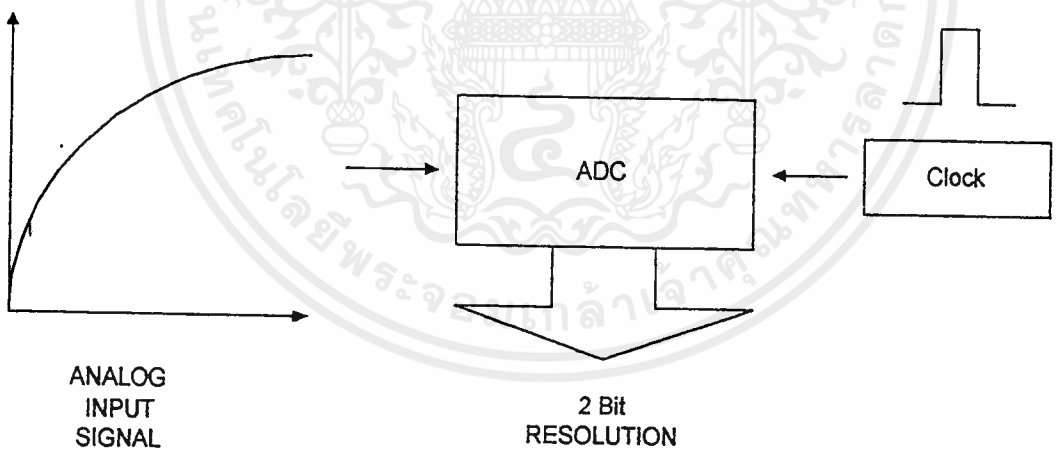
บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

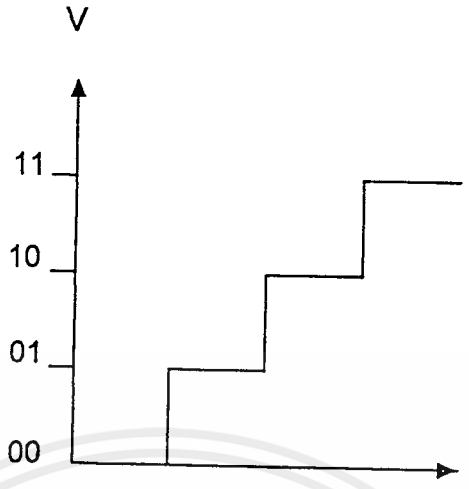
2.1 การแปลงสัญญาณอะนาล็อกเป็นสัญญาณดิจิตอลแบบขนาน

A/D แบบแฟลช (Flash Converstor) เป็น ADC ที่เร็วที่สุดในในบรรดา ADC ที่ใช้เทคนิคแบบอื่น ๆ ลักษณะของวงจรจะใช้ชุดของตัวเปรียบเทียบ (Comparator) ที่ต่อในลักษณะขนานกัน เพื่อจะทำการแปลงสัญญาณอะนาล็อกทางอินพุตให้เป็นรหัสดิจิตอล

A/D คอนเวอร์ทเตอร์ (Analog to Digital Converters) หรือ ADC ใช้สำหรับการแปลงสัญญาณอินพุตที่เป็นอะนาล็อกให้เป็นจำนวนจำกัดของดิจิตอลบิท ผลลัพธ์ที่ได้จะอยู่ในรูปของ "WORD" ทางดิจิตอลซึ่งจะกลายเป็นรหัสฐานสองที่แทนระดับแต่ละระดับของสัญญาณอะนาล็อก

ค่าเวลาการแปลง (Conversion Time) เป็นเกณฑ์ที่สำคัญอีกตัวหนึ่งของ ADC โดยในการแปลงสัญญาณอะนาล็อกให้เป็นสัญญาณทางดิจิตอลไม่ได้เกิดขึ้นโดยทันทีทันใดแต่ต้องมีการผ่านขบวนการต่าง ๆ ด้วย





รูปที่ 2.1 การแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล

2.2 ทฤษฎีการแซมปลิ้ง (Sampling Theory)

เนื่องจาก ADC ต้องการค่าเวลาขณะหนึ่งที่ใช้ในขบวนการแปลงสัญญาณอะนาล็อกให้เป็นสัญญาณทางดิจิตอล ช่วงเวลาช่วงหนึ่งจะใช้สำหรับการสุ่มตัวอย่าง (Sampling) ของสัญญาณตัวอย่าง อัตราการเปลี่ยนสัญญาณสูงสุดมีค่าเท่ากับ ส่วนกลับของค่าเวลาการเปลี่ยน (Conversion rate เท่ากับ 1/ Conversion time)

ตัวคอนเวอร์ทเตอร์จะสุ่มตัวอย่างของสัญญาณด้วยอัตราต่ำสุด เป็น 2 เท่าของความถี่สูงสุดของสัญญาณอินพุตที่เข้ามา

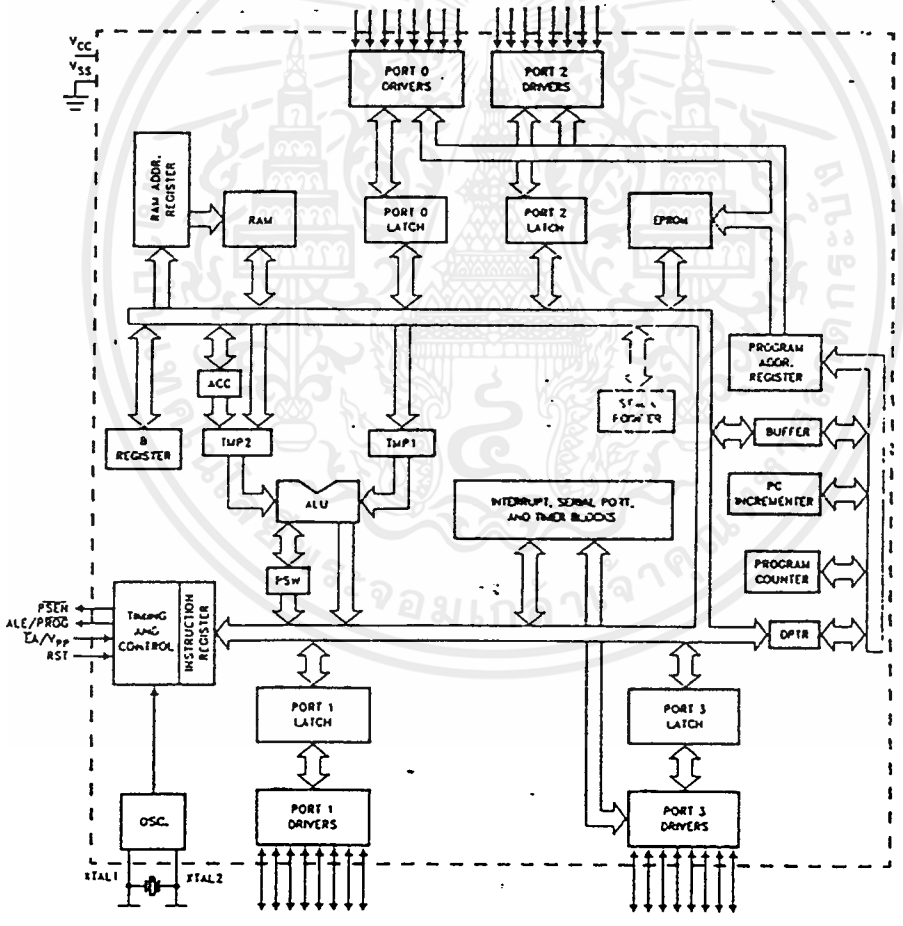
2.3 คุณสมบัติของ ADC0804

ไอซีเบอร์ ADC0804 ซึ่งเป็นไอซีแปลงสัญญาณจากอนาลอกเป็นสัญญาณดิจิตอล โดยที่ ADC0804 นี้เป็นไอซีประเภท CMOS ขนาด 8 บิตซึ่งทำงานแบบ Successive Approximation สามารถทำงานได้ทั้งหมดภายในตัวมันเอง ภายในตัวไอซีประกอบไปด้วยวงจรสร้างสัญญาณนาฬิกา และกำหนดค่าความถี่ที่ได้จาก R1 และ C1

สัญญาณนาฬิกาสูงสุดที่ใช้ได้กับไอซีหนึ่งก็คือ 640 กิโลเฮิรตซ์ซึ่งจะทำให้ใช้เวลาในการแปลง 100 ไมโครวินาที โดยมีอัตราการแซมปลิ้งที่ 10 กิโลเฮิรตซ์แต่ในการใช้งานในที่นี้จะไม่ใช้ค่าสูงสุดเพราะจะทำให้การส่งข้อมูลมีค่าเกิน 960 แซมปลิ้งต่อวินาที ในโหมดการทำงานด้วยตนเองไม่ต้องมีสัญญาณควบคุมจากภายนอก ขา READ (ขา 2) กับขา ChipSelect (ขา 1) จะถูกต่อลงกราวด์ ส่วนขา Interrupt Output (ขา 5) จะถูกต่อไปยังขา Write Data Input (ขา 3) เพื่อให้การแปลงสัญญาณออกไปภายนอกเป็นไปอย่างอัตโนมัติ

2.4 ไมโครคอนโทรลเลอร์ MCS-51

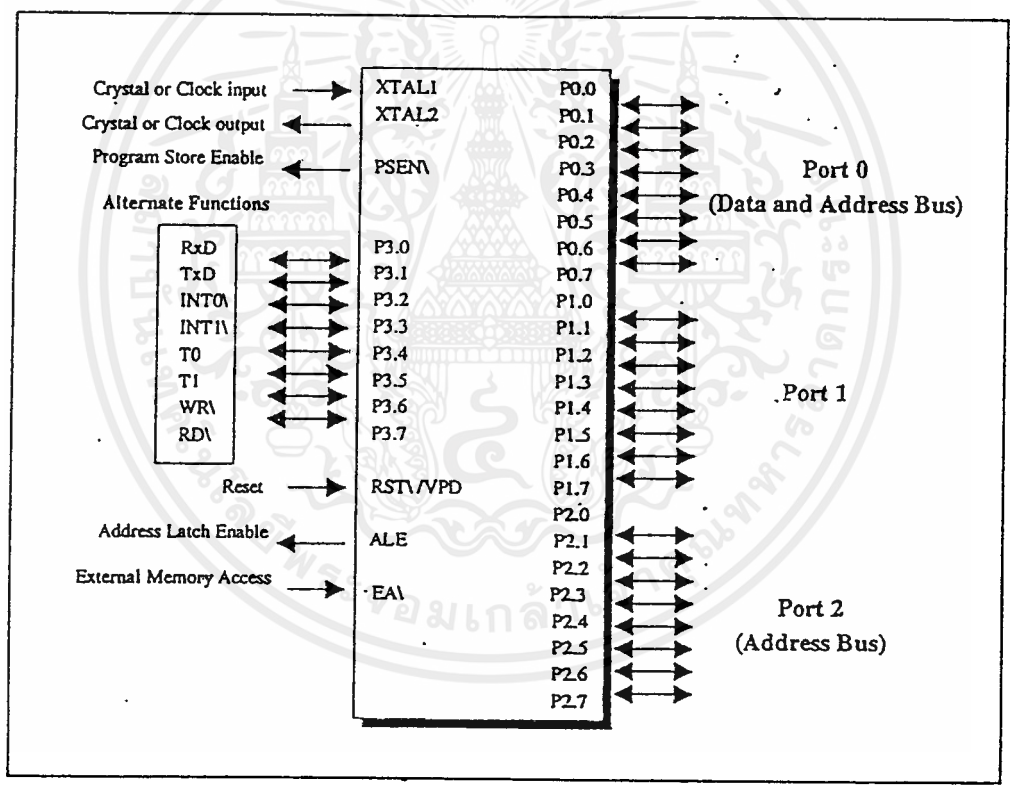
ไมโครคอนโทรลเลอร์ที่ใช้ในโครงงานนี้ คือ MCS-51 เป็นไมโครคอนโทรลเลอร์แบบชิพเดี่ยว ซึ่งผลิตโดยบริษัท Intel โดยมีการประมวลผลแบบ 8 บิต



รูปที่ 2.2 สถาปัตยกรรมภายใน 8051

ข้อมูลเกี่ยวกับ MCS-51

- วงจรรวมแบบ Dual Inline Package (DPI) ขาทั้งหมด 40 ขา
- ใช้ไฟ VCC +5 โวลท์ สามารถต่อเข้ากับอุปกรณ์เลอจิกแบบ TTL ได้โดยตรง
- มีหน่วยความจำสำหรับโปรแกรมภายในขนาด 4 กิโลไบต์
- ความถี่สัญญาณนาฬิกา 11.0529 เมกะเฮิรตซ์
- มีพอร์ทแบบขนาน (Porrallet Port) สำหรับข้อมูลเข้าออกจำนวน 32 บิต
- มีพอร์ทอนุกรมมาตรฐาน RS232 แบบ Full Duplex
- สามารถอ้างหน่วยความจำภายนอกสูงสุด 64 กิโลไบท์



รูปที่ 2.3 ไดอะแกรมขาของ 8051 แบบ DIP

หน้าที่การทำงานของขา MCS-51

VCC ขาป้อนไฟเลี้ยง + 5

VSS ขาที่ต่อกับกราวด์ของแหล่งจ่ายไฟ

Port 0 เป็นพอร์ทขนาด 8 บิต โดยใช้ได้ทั้งการรับ-ส่ง ตำแหน่งและข้อมูลกับหน่วยความจำโดยพอร์ท 0 นี้ จะส่งข้อมูลเพียงชั่วขณะหนึ่ง แล้วจะกลับมาทำหน้าที่รับข้อมูลต่อ

Port1 นี้จะทำหน้าที่รับและส่งข้อมูลเท่านั้น

Port2 ใช้ส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อและใช้เป็นพอร์ทรับและส่งข้อมูลกับภายนอก

Port 3

P3.0/RXD	(Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม
P3.1/TXD	(Serial Output Port) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม
P3.2/INTO	(External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก
P3.3/INT1	(External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก
P3.4/TO	(Time/Counter 0 External Input) ขารับสัญญาณเข้าไปยังวงจร Timer/Counter 0 ทำหน้าที่ที่นับจำนวนไซเคิลของสัญญาณ TO นี้ หรือสัญญาณนาฬิกาก็ได้
P3.5/T1	(Time/Counter 0 External Input) ขารับสัญญาณเข้าไปยัง Timer/Counter 1 ซึ่งมีการทำงานเหมือนกับ TO
P3.6/WR	(External Data Memory Write Strobe) ขาสัญญาณควบคุมการเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก

P3.7/RD (External Data Memory Read Strobe) ขาสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอก

RST ขารีเซ็ทจะใช้ทำการรีเซ็ทการทำงานของ 8051 ที่ขาRST ภายในจะมีตัวต้านทานต่อระหว่างขานี้กับกราวด์ถ้าป้อนสภาวะลอจิก 1 ก็จะทำการรีเซ็ทการทำงาน จึงสามารถต่อตัวเก็บบรรจุภายนอกระหว่างขา RST กับไฟเลี้ยง + 5โวลท์เพื่อ Power on reset

ALE (ADDRESS LATCH ENABLE)

สัญญาณนี้จะใช้บอกกับอุปกรณ์ภายนอก 8051 ว่าขณะนี้สัญญาณนี้ Active (L00IC 1) จะมีการส่งข้อมูลที่เป็น8 บิตล่างของตำแหน่งหน่วยความจำภายนอก 8051 ที่ต้องการติดต่อออกไปทางพอร์ท 0 อุปกรณ์ภายนอกจะใช้สัญญาณ ALE Latch ข้อมูลไว้เฉพาะพอร์ท 0

PSEN (PROGRAM STORE ENABLE)

ขานี้ปกติจะเป็นลอจิก 1 แต่เมื่อเป็นลอจิก 0 ก็คือต้องการอ่านคำสั่ง (Fetch Instruction) ที่จะนำไปทำงานมาจากหน่วยความจำ สำหรับโปรแกรมภายนอก 8051 แต่กรณีอ่านคำสั่ง ซึ่งเก็บอยู่ภายใน 8051 สัญญาณนี้จะไม่เปลี่ยนสภาวะ

EA (External Access)

ถ้าเป็นลอจิก 0 ที่ขา EA นี้ แสดงว่าโปรแกรมที่ต้องการให้ทำงานถูกเก็บไว้ภายนอก 8051 ถ้าเป็นลอจิก 1 แสดงว่าโปรแกรมที่ต้องการทำงานถูกเก็บไว้ภายใน ROM 8051

XTAL 1

ถ้าต้องการใช้สัญญาณนาฬิกาภายนอกควบคุม 8051 ก็ให้ป้อนเข้ามาขานี้แต่ต้องการใช้วงจรออสซิลเลเตอร์ภายในให้ต่อ Crystal ร่วมกับคาปาซิเตอร์ ซึ่งในที่นี้มีค่าประมาณ 20 pF

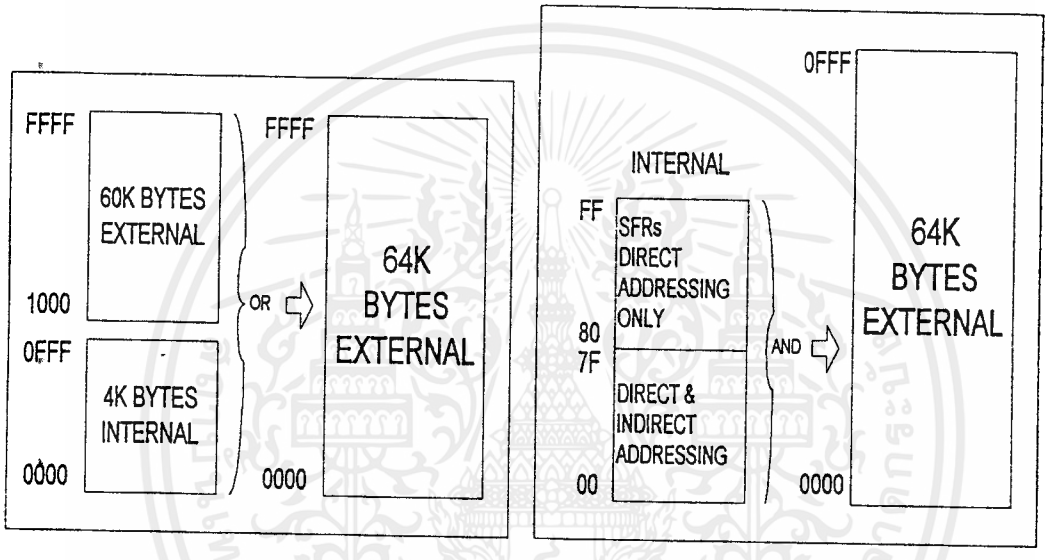
หน่วยความจำภายใน 8051

หน่วยความจำภายใน 8051 แบ่งออกเป็น 2 แบบ คือ

-หน่วยความจำสำหรับโปรแกรม (Program Area)

-หน่วยความจำสำหรับเก็บข้อมูล (Data Area)

ดังแสดงในรูปที่ 2.4



รูปที่ 2.4 แสดงค่าของหน่วยความจำ 8051

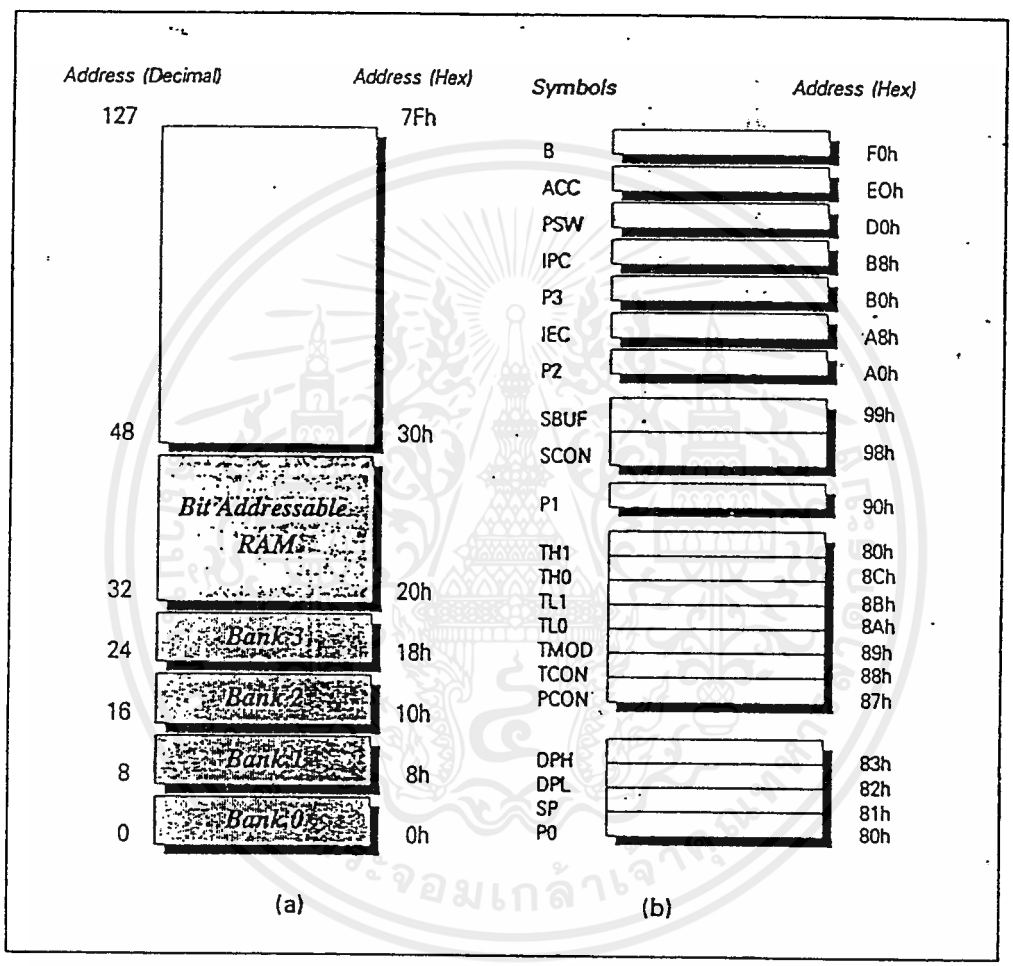
หน่วยความจำสำหรับโปรแกรม เป็นหน่วยความจำที่ 8051 ใช้สำหรับเก็บโปรแกรมภาษาเครื่องที่ 8051 จะทำงานเมื่อเริ่มป้อนไปเลี้ยงให้กับ 8051 ซึ่งหน่วยความจำสำหรับโปรแกรมนี้ สามารถเลือกได้ว่า จะเป็นหน่วยความจำที่อยู่ภายใน 8051 หรือภายนอก 8051

หน่วยความจำสำหรับข้อมูลที่ 8051 ใช้สำหรับเก็บหรือพักข้อมูลระหว่างการทำงานซึ่งหน่วยความจำนี้มี 2 แบบ

-แบบที่หนึ่งมีขนาด 128 ไบท์ อยู่ภายใน 8051

-แบบที่สองมีขนาด 64 K ไบท์ ต้องต่อเพิ่มเติมเข้าไปภายนอก 8051

รูปที่ 2.5 เป็นหน่วยความจำภายใน 8051 ซึ่งมีขนาด 128 ไบท์ โดยมีค่าตำแหน่งของหน่วยความจำ ตั้งแต่ 00H ถึง 7FH โดยแบ่งออกเป็น 3 กลุ่ม ดังนี้



รูปที่ 2.5 หน่วยความจำภายในของ 8051 ตำแหน่งที่ 00H ถึง 7FH

1. Register bank 0, bank 1, bank 2 และ bank 3, ช่วงตำแหน่งของหน่วยความจำที่ 00H ถึง 1FH โดยหน่วยความจำแบบนี้แบ่งออกเป็น 4 ชุด ชุดละ 8 ไบท์ แต่ละชุดเรียกว่า BANK แต่ละ byte ใน 1 BANK จะมีชื่อของ Register ว่า RO, R1, R2, R3, R4, R5, R6, และ R7 Register เหล่านี้จะเรียกใช้งานระหว่างการทำงานของโปรแกรมได้อย่างสะดวก และ Register เหล่านี้จะมีชื่อซ้ำกันทุก BANK โดยต่างกันที่ ตำแหน่งของหน่วยความจำ ในการใช้งานจะใช้ได้ครั้งละ 1 BANK เท่านั้น โดยการกำหนดค่าใน Register PSW รูปที่ 2.6เป็น Register แต่ละ BANK และค่าตำแหน่งของหน่วยความจำ

รีจิสเตอร์	ตำแหน่งหน่วยความจำ			
	BANK 0	BANK 1	BANK 2	BANK 3
R0	0	8	10	18
R1	1	9	11	19
R2	2	A	12	1A
R3	3	B	13	1B
R4	4	C	14	1C
R5	5	D	15	1D
R6	6	E	16	1E
R7	7	F	17	1F

รูปที่ 2.6 ตำแหน่งของรีจิสเตอร์ R0 ถึง R7

2. Bit Address Area เป็นหน่วยความจำในช่วงตำแหน่ง 20 H ถึง 4FH หน่วยความจำแต่ละบิทในพื้นที่นี้ จะสามารถตรวจสอบหรือตั้งค่า 1 หรือ 0 ได้โดยการโปรแกรม และในแต่ละบิทของข้อมูลในหน่วยความจะมีค่าตำแหน่งดังใน Memory Map รูปที่ 2.6

3. Scratched Pod Area เป็นช่วงหน่วยความจำตำแหน่ง 30H ถึง 7FH หน่วยความจำในช่วงนี้สามารถใช้งานในการเก็บข้อมูลทั่วไป เช่น Stact Pointer

Flag Register เป็น Register ที่ใช้เก็บสภาวะที่เกิดขึ้นระหว่างการคำนวน หรือจะใช้เลือก Bank ของ 8051 Register นี้คือ PSW Program Status Word มีขนาด 8 bit

ชื่อบิต: PSW ตำแหน่ง: D0h ค่าบิตเริ่มต้น: 0000 0111

CY	AC	F0	RS1	RS0	OV	-	P

ชื่อบิต	ตำแหน่ง	ความหมาย
CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
F0	PSW.5	Flag 0
RS1	PSW.4	บิตสำหรับเลือกรีจิสเตอร์แบงก์ บิต 1
RS0	PSW.3	บิตสำหรับเลือกรีจิสเตอร์แบงก์ บิต 0
OV	PSW.2	Overflow Flag
-	PSW.1	
P	PSW.0	Parity Flag

รูปที่ 2.7 PSW ของ 8051

PSW.0 บิท 0 คือ Parity bit โดยจะใช้บอกว่าใน Register Accumultor มี 1 เป็นจำนวนคู่หรือจำนวนคี่

PSW.1 บิท 1 บิทที่ไม่ได้ใช้งาน

PSW.2 บิท 2 คือ Overflow Flag เป็นบิทที่บอกการคำนวณนั้น ทำให้เกิดตัวทดขึ้นในระหว่างการคำนวณตัวทดนี้เกิดจาก บิทที่ 6 ไปยังบิทที่ 7

PSW.3,PSW.4 สำหรับ 2 บิทนี้ จะใช้งานร่วมกันเพื่อเป็นตัวบอกว่าขณะนี้ใช้ Register R0 ถึง R7 ใน BANK ใด ดังตาราง

บิท 4 (RB1)	บิทที่ 3 (RB0)	Register bank	address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

รูปที่ 2.8 บิทของ PSW.3 และ PSW.4

PSW.5 เป็นเอนกประสงค์

PSW.6 คือ Auxiliary Flag

PSE.7 คือ Carry Flag

Spectial Funtion Register, SFR

ใน 8051 มี Register ที่สำหรับใช้งานเฉพาะ คือข้อมูลที่ถูกนำไปเก็บไว้ใน Register เหล่านี้จะมีความหมายเฉพาะตัวของ Register โดยแต่ละ Register จะมีตำแหน่งของตัว

มันเองดังแสดงในตารางข้างล่าง ซึ่งจากตารางจะแสดง Symbol และชื่อของ Register และช่องสุดท้ายคือ ตำแหน่งของ Register

รีจิสเตอร์	ความหมาย	ความสามารถการอ้างถึงแบบบิต
ACC	Accumulator	ได้
B	B register	ได้
PSW	Program Status Word	ได้
SP	Stack Pointer	ได้
DPTR	Data pointer (DPH & DPL)	ได้
P0	Port 0	ได้
P1	Port 1	ได้
P2	Port 2	ได้
P3	Port 3	ได้
IP	Interrupt Priority	ได้
IE	Interrupt Enable	ได้
TMOD	Timer / counter mode	ไม่ได้
TCON	Timer / counter control	ได้
TH0	Timer / counter 0 (high)	ไม่ได้
TL0	Timer / counter 0 (low)	ไม่ได้
TH1	Timer / counter 1 (high)	ไม่ได้
TL1	Timer / counter 1 (low)	ไม่ได้
SCON	Serial control	ไม่ได้
SBUF	Serial data buffer	ไม่ได้
PCON	Power control	ไม่ได้

*ACC	00000000	TMOD	00000000
*B	00000000	*TCON	00000000
*PSW	00000000	T2CON	00000000
SP	00000111	TH0	00000000
DPTR		TL0	00000000
DPH	00000000	TH1	00000000
DPL	00000000	TL1	00000000
*P0	11111111	TH2	00000000
*P1	11111111	TL2	00000000
*P2	11111111	RCAP2H	00000000
*P3	11111111	RCAP2L	00000000
*IP	8051 XXX00000,	*SCON	00000000
	8052 XX000000	SBUF	Indeterminate
*IE	8051 0XX00000,	PCON	HMOS 0XXXXXXX
	8052 0X000000		CHMOS 0XXX0000

รูปที่ 2.9 แสดงตำแหน่งของหน่วยความจำแต่ละบิท

และรูปที่ 2.10 แสดงตำแหน่งหน่วยความจำแต่ละบิท

จากรูปข้างล่างนี้จะเห็นว่า Register SFR จะใช้งานเฉพาะอย่างโดยง่าย SFR สามารถเข้าถึงแบบ BIT ADDRESS เพื่อใช้ในการตรวจสอบสภาวะการทำลายได้รวดเร็วขึ้น

<u>การทำงานของ Special Function Register</u>

1. Accumulater Register มีขนาด 8 bit ที่ใช้สำหรับการคำนวณและเก็บค่าอ้างอิงและให้อ่านค่าของข้อมูลจากภายนอก ซึ่งต้องผ่าน Register นี้

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* = Bit addressable
+ = 8052 only

รูปที่ 2.10 แสดงตำแหน่งหน่วยความจำแต่ละบิทของ 8051

2. Register B ซึ่งเป็น Register ที่ใช้สำหรับการคูณ และการหาร โดย Register B จะเก็บตัวคูณและผลลัพธ์ บิท 8 ถึง 15 ในคำสั่งการคูณ สำหรับการหาร Register B จะเก็บตัวหารและผลการหาร

3. Programs status word ดังที่กล่าวมาแล้ว

4. Stack Pointer Register นี้จะใช้เก็บตำแหน่งของหน่วยความจำภายใน 8051 ที่ ใช้เก็บตำแหน่งเดิมของโปรแกรมก่อนทำงานคำสั่ง CALL และ PUSH

5. Data Pointer Register โดย Register นี้มีขนาด 12 Bit หน้าที่ของ Register นี้ใช้สำหรับชี้ตำแหน่งในหน่วยความจำ

6. PORT 0 ถึง 3 เป็นขาที่ใช้ติดต่อรับข้อมูลจากภายนอกกับ 8051

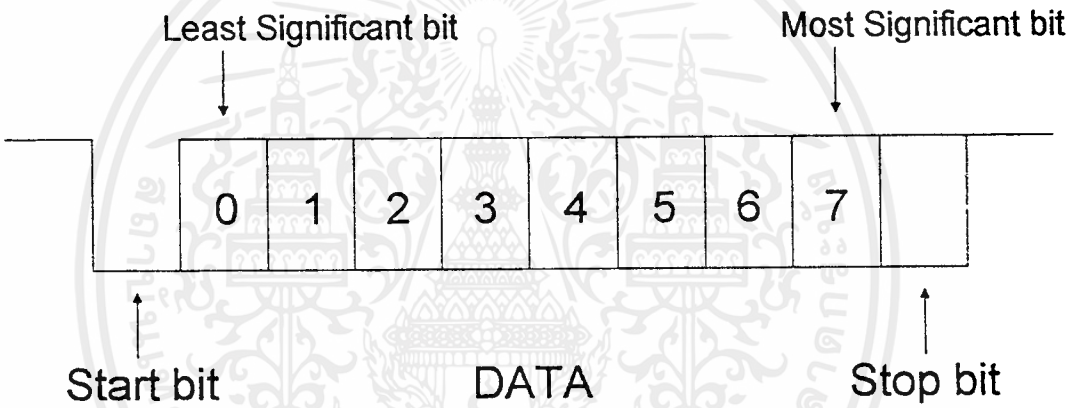
7. Serial Data Buffer

Serial Data Buffer

รีจิสเตอร์นี้มีขนาด 8 บิทและโครงสร้างภายในแล้วรีจิสเตอร์นี้มี 2 ตัวที่มีชื่อเดียวกันตัวหนึ่งสำหรับเก็บข้อมูลที่จะส่งแบบอนุกรมออกจาก 8051 และอีกตัวหนึ่งสำหรับรับข้อมูลแบบอนุกรมเข้ามา ดังนั้น Serial Port ของ 8051 จึงเรียกว่ามีการทำงานแบบ Full Duplex เพราะสามารถส่งและรับข้อมูลได้ในเวลาเดียวกันเนื่องจากมีรีจิสเตอร์สำหรับส่งและรับแยกออกจากกัน ข้อมูลที่ต้องการจะส่งออกก็ให้เขียนไปยังรีจิสเตอร์ SBUF แล้วส่งงานให้ส่งข้อมูลออกมา ข้อมูลในรีจิสเตอร์ จะเริ่มส่งออกโดยเริ่มจากบิท 0 ถึง 7 ตามลำดับถ้าข้อมูลมีข้อมูลเข้ามาทางขา RED ก็จะถูกเก็บไปไว้ในรีจิสเตอร์นี้โดยถือว่าข้อมูลบิทแรกที่เข้ามาคือบิท 0

Serial Port จะสามารถกำหนดให้การทำงาน รับ-ส่ง ข้อมูลแบบอนุกรมได้ 4 โหมด (MODE) โดยการกำหนดในรีจิสเตอร์ SCON (Serial Port Control Register) แต่ละโหมด การทำงานของ Serial Port มีดังนี้

MODE 0: ในโหมดนี้จะมีการรับหรือส่งข้อมูลแบบอนุกรมทางขา RXD และขา TXD จะส่งสัญญาณ Clock ที่ใช้สำหรับเลื่อน (Shift) ข้อมูล 1 ชุดของข้อมูลจะประกอบด้วยข้อมูล 8 บิทเท่านั้นและจะเริ่มการรับส่งข้อมูล-ส่งข้อมูลจากบิท 0 จนถึงบิท 7 ตามลำดับ อัตราการส่งข้อมูลแบบอนุกรมจะเท่ากับ 1/12 เท่าของความถี่สัญญาณนาฬิกาที่ใช้กับ 8051



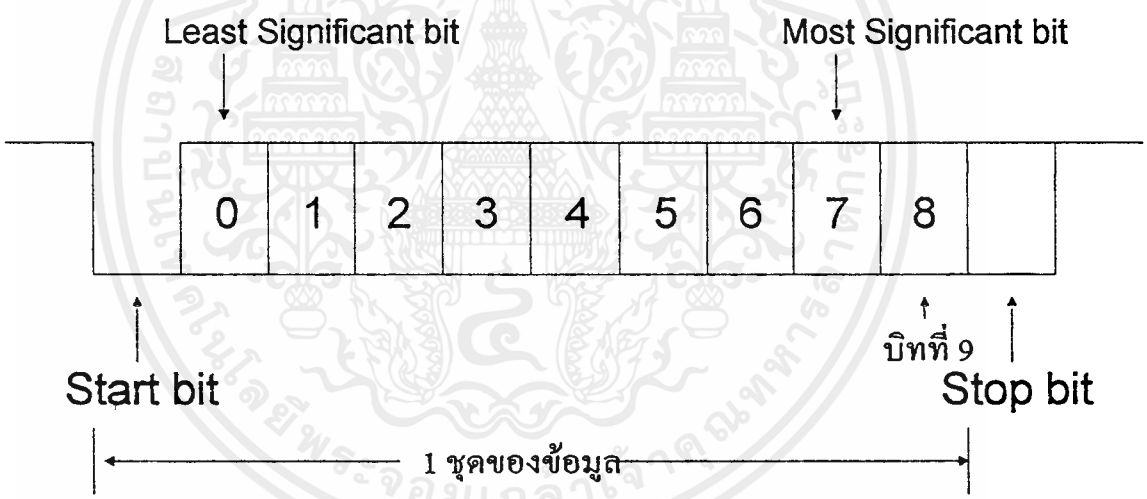
รูปที่ 2.11 ชุดข้อมูลอนุกรมในโหมด 1

MODE 1: -ข้อมูลที่รับ-ส่ง 1 ชุดในโหมดนี้จะมี 10 บิท ผ่านทางขา RXD และ TXD ตามลำดับเริ่มต้นการรับส่งข้อมูลด้วย Start bit 1 บิท (ลอจิกเป็น 0), ข้อมูล 8 บิท (เริ่มจากบิท 0), Stop bit 1 บิท (ลอจิก 1) การส่งข้อมูลโหมดนี้มีดังรูป 2.11

เมื่อรับข้อมูลอนุกรมเข้ามาข้อมูล 8 บิทจะถูกเก็บในรีจิสเตอร์ SBUF และ Stop Bit จะถูกเก็บไปที่บิท RB8 ในรีสเตอร์ SCON ในการส่งข้อมูลออกก็ จะเขียนข้อมูลที่ต้อง

การส่งไปยังรีจิสเตอร์ SBUF อัตราการส่งข้อมูลในโหมดนี้สามารถกำหนดได้ตามต้องการ โดยจะขึ้นกับการเกิด Overflow ใน Timer 1

MODE 2: การรับ-ส่งข้อมูลของโหมด 2 1 ชุดจะมี 11 บิท ข้อมูลจะส่งออกผ่านทางขา TXD และรับเข้ามาทางขา RXD ข้อมูลแต่ละชุดจะเริ่มต้นด้วย Start bit 1 บิท, ข้อมูล 8 บิท (เริ่มจากบิท 0), ข้อมูลบิทที่ 9 จำนวน 1 บิทและ Stop Bit อีก 1 บิทข้อมูลบิทที่ 9 ที่จะส่งออกนี้สามารถกำหนดได้ว่าจะให้เป็น 1 หรือ 0



รูปที่ 2.12 ชุดข้อมูลอนุกรมในโหมด 2

อัตราการส่งข้อมูลจะกำหนดให้เป็น 1/32 หรือ 1/64 เท่าของความถี่สัญญาณนาฬิกาที่ใช้กับ 8051

MODE 3: การส่งข้อมูลโหมดนี้ 1 ชุดมี 11 บิท เหมือนกับโหมด 2 ทุกประการ แตกต่างกันตรงอัตราการส่งข้อมูลเท่านั้น คืออัตราการส่งข้อมูลในโหมด 3 นี้สามารถกำหนดได้ตามต้องการ โดยจะขึ้นกับการเกิด Overflow ใน Timer 1 เหมือนกับ โหมด 1

SCON (Serial Port Control Register)

รีจิสเตอร์ SCON มีขนาด 8 บิท ใช้สำหรับควบคุมการส่งและรับข้อมูลผ่านทาง Serial Port แต่ละบิทของข้อมูลในรีจิสเตอร์นี้ มีความหมายเฉพาะดังรูปที่ 27

ในรูปที่ 2.13 บิท RI จะเป็นชื่อของบิท 0 และ SMO จะเป็นบิทที่ 7 ของรีจิสเตอร์ SCON ซึ่งความหมายหรือการทำงานของแต่ละบิทมีดังนี้

RI (Receive Interrupt Flag)

บิทนี้จะถูกกำหนดโดย ฮาร์ดแวร์ให้มีค่าเป็น 0 หรือ 1 โดยที่ในการรับข้อมูลโหมด 0 นั้นบิท RB8 จะมีค่าเป็น 1 เมื่อมีข้อมูลเข้ามาครบทั้ง 8 บิท ส่วนในโหมดอื่นบิท RB8 จะเป็น 1 ก็ต่อเมื่อข้อมูลเข้ามาถึงเวลาครึ่งหนึ่งของ Stop Bit

SCON: SERIAL PORT CONTROL REGISTER. -BIT ADDRESSABLE.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 SCON. 7 Serial Port mode specifier. (NOTE 1)

SM1 SCON. 6 Serial Port mode specifier. (NOTE 1)

SM2 SCON. 5 Enables the multiprocessor communication feature in modes 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0 (See Table 9).

REN SCON. 4 Set/Cleared by software to Enable/Disable reception.

TB8 SCON. 3 The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.

RB8 SCON. 2 In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.

TI SCON. 1 Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.

RI SCON. 0 Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8-Bit UART	Variable
1	0	2	9-Bit UART	Fosc./64 OR Fosc./32
1	1	3	9-Bit UART	Variable

SERIAL PORT SET-UP:

Table 9

MODE	SCON	SM2 VARIATION
0	10H	Single Processor Environment (SM2 = 0)
1	50H	
2	90H	
3	D0H	
0	NA	Multiprocessor Environment (SM2 = 1)
1	70H	
2	B0H	
3	F0H	

รูปที่ 2.13 Serial Port Control Register (SCON)

TI (Transmit Interrupt Flag)

ค่าในบิท TI จะถูกกำหนดให้เป็น 1 หรือ 0 ด้วยฮาร์ดแวร์ โดยในการส่งข้อมูลแบบอนุกรมโหมด 0 บิทนี้ จะเป็น 1 เพื่อจะบอกว่าการส่งข้อมูลในรีจิสเตอร์ SBUF ออกไปทางพอร์ทอนุกรมครบทั้ง 8 บิท แต่ถ้าเป็นการส่งข้อมูลแบบอนุกรมในโหมดอื่น จะทำให้ข้อมูลในบิท TI เป็น 1 เมื่อเริ่มการส่ง Stop Bit

RB8

เมื่อมีการกำหนดให้รับข้อมูลในโหมด 2 และ 3 จะใช้บิทนี้สำหรับเก็บข้อมูลบิทที่ 9 ที่เข้ามาทางพอร์ทอนุกรม ส่วนในโหมด1 นั้นบิทนี้จะเก็บ Stop Bit ซึ่งมีค่าเป็น 1 นั่นเองในโหมด 0 บิทนี้จะไม่ถูกใช้งาน

TB8

ในการส่งข้อมูลแบบอนุกรมโหมด 2 และ 3 จะใช้บิทนี้เก็บข้อมูลบิทที่ 9 ส่วนโหมดอื่นจะไม่ใช้งาน บิทนี้

REN (Receive Enable)

เป็นบิทที่จะใช้กำหนด ให้ทำการรับข้อมูลเข้ามาจากทางพอร์ทอนุกรม หรือไม่ถ้าบิทนี้เป็น 1 ก็จะรับข้อมูลเข้ามา แต่ถ้าเป็น 0 ก็จะไม่รับข้อมูลที่ ขา RXD เข้ามา

SM2

เป็นบิทสำหรับควบคุมการทำงานของฮาร์ดแวร์ที่จะทำให้บิท RI เป็น 1หรือไม่ ในกรณีที่บิท SM2 เป็น 0 ค่าในบิท RI ก็จะเป็นไปตามที่ได้อธิบายมาแล้ว ในเรื่องบิท RI แต่ถ้าบิท SM2 = 1

โหมด 2 และ 3 ปกติแล้วบิท RI จะเป็น 1 เมื่อ SM2 เป็น 1แล้ว RI จะเป็น 1 ก็ต่อเมื่อ ข้อมูลบิทที่ 9 ที่ เข้ามามีค่าเป็น 1 ถ้าข้อมูลบิทที่ 9 เข้ามาเป็น 0 จะไม่ทำให้บิท RI มีค่าเป็น 1

ในโหมด 1 บิท RI มีค่าเป็น 1 เมื่อข้อมูล Stop Bit เข้ามายังพอร์ทอนุกรมถูกต้องแต่ถ้า Stop Bit ไม่เข้ามายังพอร์ทอนุกรม อันอาจเกิดจากปัญหาในการส่งข้อมูลแล้วบิท RI จะมีค่าเป็น 0 ในโหมด 0 บิทนี้จะมีค่าเป็น 0 เสมอ

SM0, SM1

เป็น 2 บิทที่ใช้งานร่วมกันเพื่อกำหนดโหมดของการรับ-ส่งข้อมูลของพอร์ทอนุกรม ค่าใน 2 บิทนี้จะกำหนดโหมดได้ดังนี้

SM0	SM1	MODE	Description
0	0	0	Shift register
0	1	1	8 - bit UART
1	0	2	9 - bit UART
1	1	3	9 - bit UART

Timer Register

ใน 8051 จะมีวงจร Timer อยู่ 2 ชุดคือ Timer 0 และ Timer 1 ใน Timer แต่ละชุด จะมี Register ขนาด บิท อยู่ 2 ตัวเพื่อเก็บค่าการนับของ Timer ได้สูงสุดถึง 16 บิท ใน Timer 0 รีจิสเตอร์ TH0, TL0 และใน Timer 1 คือรีจิสเตอร์ TH1 TL1 Tlx จะเก็บค่าของการนับ 8 บิทล่างและ THx จะเก็ยค่าของการนับ 8 บิทบน

TMOD Timer/Counter mode register

Gate เป็นบิทที่ใช้ควบคุมให้ Timer ทำงานหรือไม่ถ้าบิทนี้ของ Timer x ถูกตั้งเป็น 1 จะทำให้ Timer ทำงานก็ต่อเมื่อที่ขา INTx มีสภาวะลอจิกเป็น 1 และบิท Trx ในรีจิสเตอร์ TCON เป็น 1 ด้วย

C/T บิทนี้ใช้สำหรับเลือกการทำงานของ Timer ว่าจะใช้เป็น Timer หรือ Counter

M1, Mo เป็น 2 บิทที่ใช้ร่วมกันเพื่อเลือกโหมดการทำงานของ Timer การทำงานโหมด 0, 1 และ 2 ของ Timer 0 จะเหมือนกับ Timer 1 แต่ในโหมด 3

M1	M0	การทำงาน
0	0	โหมด 0 รีจิสเตอร์ Thx และ Tlx ทำตัวเป็นตัวนับ 13 บิท ค่าจากการนับ 8 บิทบนมาจาก 8 บิทของ Thx และอีก 5 บิทล่างมาจากค่า 5 บิทของรีจิสเตอร์ Tlx โดยที่ 3 บิทบน ของ Tlx จะไม่ต้องสนใจเลย
0	1	โหมด 1 รีจิสเตอร์ Thx และ Tlx ทำตัวเป็นตัวนับ 16 บิท ค่าจากการนับ 8 บิทบนอยู่ในรีจิสเตอร์ Thx และค่าจากการนับ 8 บิทล่างอยู่ในรีจิสเตอร์ Tlx
1	0	โหมด 2 ในการนับของรีจิสเตอร์ Tlx ขนาด 8 บิท เมื่อนับถึงค่าสูงสุดคือ FFH เมื่อทำการนับต่อไปจะเกิดการ Overflow แล้วก็จะ "Reload" เอาข้อมูลจาก Thx เข้าไปยัง Tlx เข้าไปยัง Tlx เพื่อเป็นค่าเริ่มต้นในการนับครั้งต่อไป
1	1	โหมด 3 การทำงานของ Timer 0 และ Timer 1 จะต่างกันดังที่จะกล่าวต่อไป

รูปที่ 2.14 การเลือกการทำงานของ Timmer1

TCON Timer Control Register

รีจิสเตอร์ขนาด 8 บิท นี้ใช้ควบคุมการทำงานและ บอกสภาวะของ Timer 0 และ Timer 1 แต่ บิทของรีจิสเตอร์นี้จะทำงานต่างกันดังรูปที่ 2.15

TCON: TIMER/COUNTER CONTROL REGISTER. BIT ADDRESSABLE.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 TCON. 7 Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.

TR1 TCON. 6 Timer 1 run control bit. Set/cleared by software to turn Timer/Counter 1 ON/OFF.

TF0 TCON. 5 Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.

TR0 TCON. 4 Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.

IE1 TCON. 3 External Interrupt 1 edge flag. Set by hardware when External Interrupt edge is detected. Cleared by hardware when interrupt is processed.

IT1 TCON. 2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

IE0 TCON. 1 External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.

IT0 TCON. 0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TMOD: TIMER/COUNTER MODE CONTROL REGISTER. NOT BIT ADDRESSABLE.

GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

GATE When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/T Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

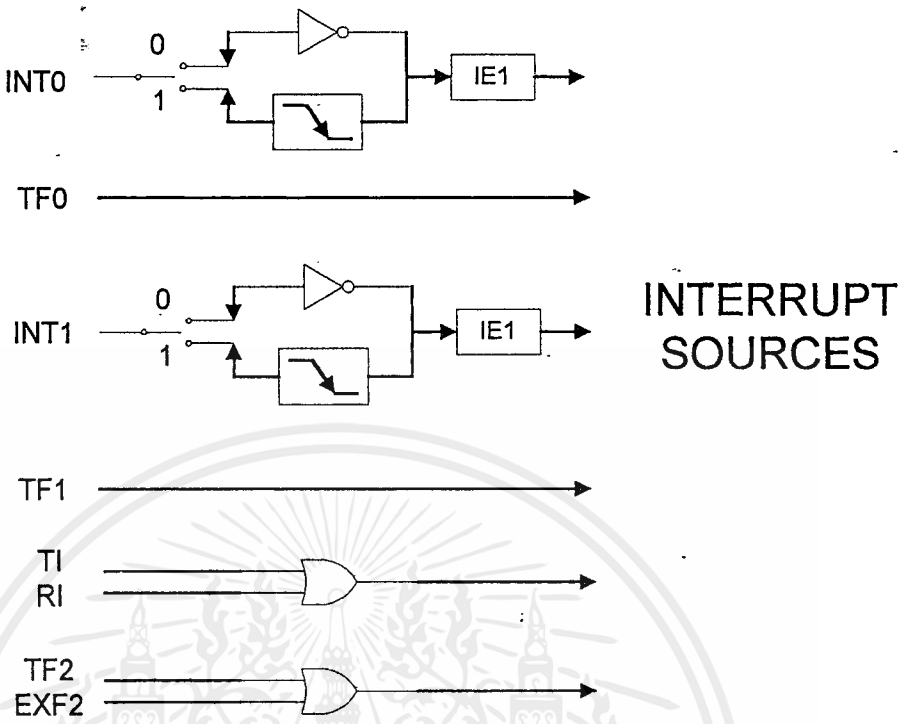
M1 Mode selector bit. (NOTE 1)

M0 Mode selector bit. (NOTE 1)

รูปที่ 2.15 TCON Timer Control register

IE (Interrupt Enable Register)

การขัดจังหวะการทำงานเป็นการที่มีสัญญาณหนึ่งหรือคำสั่งหนึ่งที่ จะทำให้การทำงานการปกติของ โปรแกรมถูกขัดจังหวะใน 8051 จะสามารถขัดจังหวะด้วยสัญญาณจาก 6 แหล่งดังรูปที่ 2.16



รูปที่ 2.16 แหล่งกำเนิดสัญญาณขัดจังหวะ

สัญญาณขัดจังหวะที่ 5 ในรูปที่ 2.17 จะสามารถทำให้เกิดการขัดจังหวะได้ 2 วิธี คือมีข้อมูลเข้ามาทางพอร์ทอนุกรมเก็บอยู่ที่รีจิสเตอร์ SBUF และกรณีที่ข้อมูลใน SBUF ส่งออกไปทางพอร์ทอนุกรมหมดแล้ว ไม่ว่าเกิดกรณีใด ๆ ก็ทำให้เกิดการขัดจังหวะขึ้น

PCON (Power Control Register)

8051 เป็นไมโครคอนโทรลที่สร้างขึ้นด้วยเทคโนโลยี่ทั้งแบบ CHMOS และ HMOS ซึ่งแบบ CHMOS มีข้อดีตรงที่ใช้กำลังไฟต่ำกว่าแบบ HMOS ดังนั้นต่อไปในอนาคตจึงจะมีแต่เฉพาะรุ่น CHMOS เท่านั้น นอกจากนี้แล้ว 8051 ยังมีข้อดีอีกตรงที่สามารถลดการใช้กำลังไฟลงได้โดยการทำงานใน Idle mode และ Power Down Mode ใน Idle Mode นั้น สัญญาณนาฬิกาออกจากออสซิลเลเตอร์จะป้อนให้เฉพาะส่วน Interrupt, Serial Port และ Timer ในส่วนอื่นจะไม่มีสัญญาณนาฬิกาไปเลี้ยง แต่มีไปเลี้ยง ให้กับทุกส่วนในวงจรการใช้

กำลังไฟจึงลดลงมาก ส่วนใน Power Down Mode นั้น ออสซิลเลเตอร์จะหยุดทำงานทำให้ไม่มีสัญญาณนาฬิกาไปเลี้ยงส่วนใด ๆ ในวงจรเลยแต่ข้อมูลภายในรีจิสเตอร์จะยังคง อยู่ไม่สูญหายไป รายละเอียดของแต่ละโหมดจะได้กล่าวต่อไป

ชื่อบิต: PCON **ตำแหน่ง:** 97h **ค่าบิตเริ่มต้น:** 0xxx 0000

SMOD	-	-	-	GF1	GF0	PD	IDL

ชื่อบิต	ตำแหน่ง	ความหมาย
SMOD	PCON.7	บิตกำหนดการทวีคูณของอัตราบอดปกติ
	PCON6	
	PCON.5	
	PCON.4	
GF1	PCON.3	แฟล็กสำหรับให้ผู้ใช้ใช้งานทั่วไป Flag 1
GF0	PCON.2	แฟล็กสำหรับให้ผู้ใช้ใช้งานทั่วไป Flag 0
PD	PCON.1	บิตสำหรับการกำหนด Power down
IDL	PCON.0	บิตสำหรับการกำหนด Idle mode

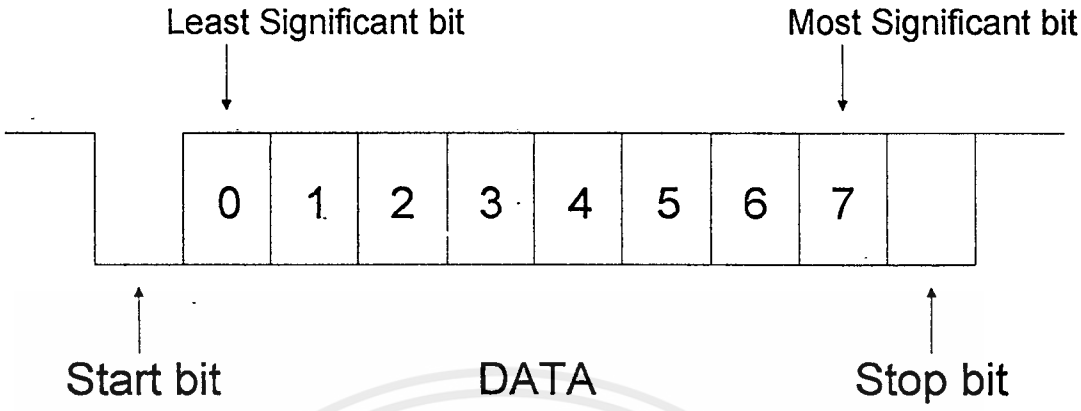
รูปที่ 2.17 PCON : Power Control Register

SMOD บิท 7 เป็นบิทที่ใช้ร่วมในการกำหนดอัตราการส่งข้อมูล (Baud Rate) ผ่านทางอนุกรม ซึ่งในการรับ-ส่ง ข้อมูลผ่านทางพอร์ทอนุกรม โหมด 1 และ 3 จะสามารถกำหนดอัตราการส่งข้อมูลได้ ตามอัตราการเกิด Overflow ใน Timer 1 ถ้าบิทนี้เป็น 1 จะทำให้อัตราการส่งข้อมูลเพิ่มขึ้น 2 เท่า รายละเอียดการส่งข้อมูลผ่านพอร์ทอนุกรม

2.5 การส่งและรับข้อมูลแบบอนุกรมผ่าน 8051

สำหรับ MCS 8051 สามารถส่งและรับข้อมูลอนุกรมแบบ Universal Asynchrnous Receive Transmitter (UART) แบบ Full duplex ที่สามารถเลือกรูปแบบในการรับส่งได้ถึง 4 แบบ

สำหรับในโครงงานได้เลือกใช้การส่งและรับข้อมูลใน MODE 1



รูปที่ 2.18 การส่งข้อมูลแบบอนุกรมใน MODE1

โดย ข้อมูลที่รับส่ง 1 ชุด มีขนาด 10 Bit

Start bit 1 บิท (Logic 0)

DATA bit 8 บิท

Stop bit 1 บิท (Logic 1)

และอัตราความเร็วในการส่งข้อมูลมีค่าเป็น 9600 B/s.

ในการทำงานในการส่งและรับข้อมูล และอัตราความเร็วในการส่งข้อมูลจะต้องทำการใช้งาน SFR ต่าง ๆ ดังนี้

- SCON (Serial Port Control Register)

Scon เป็น register เพื่อควบคุมการส่งและรับข้อมูลแบบ Serial port ในที่ นี้ได้กำหนดให้การรับส่งข้อมูลทำงานใน MODE 1 โดยข้อมูล 1 ชุด มี ขนาด 10 Bit ซึ่งประกอบด้วย Start bit 1 บิท (Logic 0), DATA bit 8 bit และ Stop bit 1 บิท (Logic 1) และเป็น Register เพื่อให้มีการ รับข้อมูลเข้ามาทาง RXD.

- TMOD (Timer/Counter Mode Register)

TMOD เป็น register ที่ทำหน้าที่ควบคุมการทำงานของ Timer หรือ Counter ในการใช้งานนี้ได้กำหนดใช้งานให้ใช้ Timer 1 และเลือกการทำงานของ Timer 1 ใน Mode 2 โดยทำงานแบบ "Reload" และเป็น Register ที่เป็นตัวกำหนดให้ วงจรในการรับส่งทำงาน

- TCON (Timer Control Register)

TCON เป็น register เพื่อใช้ตรวจสอบสภาวะการทำงานของ Timer ในที่นี้ได้ใช้ Register นี้ในการตรวจสอบว่ามีข้อมูลเข้ามาทาง RXD หรือ ไม่ และใช้ตรวจสอบในการส่งข้อมูล

- IE (Interrupt Enable Register)

IE เป็น register ที่ใช้ควบคุมในการ Inerrupt

- PCON (Power Control Register)

PCON เป็น register ที่ควบคุมการใช้งาน ในรูปของการลดกำลังงานและอัตราเร็วในการส่งข้อมูล

Register ทั้งหมดที่กล่าวมานี้จะต้องทำงานที่สอดคล้องกันหมด จึงจะทำให้การรับส่งข้อมูลแบบ Universal Asynchonous Receive Transmitter (UART) ทำงานในอัตราความเร็ว (Baud Rate) ที่กำหนด

การกำหนด Baud Rate นั้น ได้กำหนดค่าที่ TH1 และวงจรทำงานใน Mode Timer 1 โดยกำหนดที่ ขา C/T โดยความถี่ OSC จะถูกวงจรหาร 12 มารออยู่ที่ Control ดังนั้น การ START ให้วงจรทำงาน ก็ขึ้นอยู่กับ output ของ AND gate ที่มี Logic "1" โดยถูกควบคุมด้วย TR1 และ gate หรือ INT

การทำงานโหมด 1

การส่งข้อมูล

บิท SMOD จะเป็นตัวเลือกว่า สัญญาณ Timer 1 Overflow ที่ ส่งไปยังวงจร 16 จะถูกหาร 2 ก่อนหรือไม่ ถ้า SMOD เป็น 1 สัญญาณ Timer 1 Overflow จะไม่ถูกหารแต่ถ้า SMOD เป็น 1 สัญญาณ Timer Overflow จะถูกหาร 2 ก่อน ที่จะเข้าวงจรหาร 16 การส่งข้อมูลจะเริ่มจากการที่มีคำสั่งเขียนข้อมูลไปยังรีจีสเตอร์ SBUF จะมีสัญญาณ Write to SBUF เกิดขึ้นเพื่อรับข้อมูลจาก Internal Bus ด้านบนไปเก็บยังรีจีสเตอร์ SBUF แล้วทำให้เอาท์พุทของ D FLIP FLOP ทางซ้ายของ SBUF มีค่าเป็น 1 และ เป็นบิทที่ 9 ของการส่งข้อมูลสัญญาณ Writc to SBUFยังส่งไปยังTX control ด้วย ขณะนี้ ข้อมูลในวงจรหาร 16 มีค่าเป็นอะไรไม่ทราบจึงจะรอจนกว่าข้อมูลในวงจรหาร 16 นับเพิ่มขึ้นจนถึงค่าสูงสุดแล้ววนกลับเป็น 0 คือเกิดการวนกลับทำให้เริ่มการส่งข้อมูลที่เวลา S1P1 ของไซเคิลเครื่องถัดไป (การส่งข้อมูลออกจะสัมพันธ์กับการเกิด Overflow ในวงจรหาร 16) สัญญาณ SEND จาก TX Control เปลี่ยนสภาวะลอจิกเป็น 0 แล้วเริ่มส่งข้อมูลที่เป็น Start บิท (0) ออกไป เมื่อส่ง Start Bit ออกไปแล้ววงจร TX Control ก็จะทำให้สัญญาณ DATA เป็น 1 เพื่อเลื่อนข้อมูลใน SBUF ออกไป เริ่มจากบิท 0 จนถึงบิทที่ 7 การส่งข้อมูลนี้จะเกิดขึ้นเมื่อสัญญาณ TX Clock เปลี่ยนสถานะจาก 0 เป็น 1 ดังในรูปที่ 2.20 ขณะที่ข้อมูลถูกเลื่อนออกไปนั้น จะมี 0 ถูกเลื่อนเข้ามาทางซ้ายของรีจีสเตอร์ SBUF เมื่อข้อมมูลเลื่อนออกไปทั้ง 8 บิท แล้วบิทที่ 9 ซึ่งเป็น 1 และตอนต้นอยู่ทางซ้ายสุดจะถูกเลื่อนมาอยู่ในตำแหน่งสุดท้ายทางขวาของรีจีสเตอร์ SBUF ทำให้ Zero Detector รู้ว่าเป็นข้อมูลบิทสุดท้ายแล้วที่ส่งออกโดยจะมีสัญญาณมาบอกกับวงจร TX Control ด้วย เมื่อ TX Control ส่งสัญญาณ Shift ออกไปเป็นการส่งข้อมูลบิทสุดท้าย (บิท 7) ออกไป ก็จะรออีก 1 TX Clock (Bit Clock) ก็จะทำให้ขา TXD ส่งข้อมูล Stop Bit (1) ออกมา สัญญาณ DATA ซึ่งมีสภาวะลอจิกเป็น 1 มาตั้งแต่เริ่มส่งข้อมูลบิท 0 ก็จะกลับเป็น 0 และบิท TI จะเป็น 1 เพื่อรอการสิ้นสุดการส่งข้อมูลทั้งหมดจะสิ้นสุดลงเมื่อสัญญาณ TX Clock ไซเคิลที่ 10 นับตั้งแต่สัญญาณ SEND เปลี่ยนสภาวะลอจิกเป็น 0

การรับข้อมูล

การรับข้อมูลจะขึ้นกับอัตราการเกิด Overflow ใน Timer 1 แล้วหาร 2 หรือ ไม่ขึ้นกับค่าของบิท SMOD สัญญาณนี้จะไปเข้าวงจรหาร 16 และเป็นตัวกำหนดอัตราการรับข้อมูลการรับข้อมูลจะเริ่มจากวงจร 1-T0-0 Transition Detector พบว่าสัญญาณที่ขา RXD เปลี่ยนจาก 1 เป็น 0 ซึ่งหมาย

ถึงมีข้อมูล Start bit เข้ามา การตรวจสอบนี้จะกระทำได้ด้วยอัตราเดียวกับสัญญาณที่เข้าวงจรหาร 16 เมื่อพบการเปลี่ยนสถานะลอจิกที่ขา RXD ก็จะเริ่มการรับข้อมูล ขณะนี้จะเซ็ทวงจรหาร 16 ให้มีค่าเป็น 0 เพื่อสร้างสัญญาณ RX Clock ให้เข้าจังหวะ (Synchronous) กับข้อมูลที่เข้ามาโดยสัญญาณ RX Clock จะเป็น 1 เมื่อการนับของ วงจรหาร 16 มีค่าเป็น 15 ขณะที่วงจรหาร 16 นับถึง 7,8 และ 9 จะมีการตรวจสอบข้อมูลที่ เข้ามาเป็นค่านั้นถ้าในการตรวจสอบ Start Bit แล้วพบว่าเกิดผิดพลาด คือไม่เป็น 0 ก็จะรีเซ็ทการทำงานเพื่อไปตรวจสอบการเปลี่ยนสถานะจาก 1 เป็น 0 ของข้อมูลที่ขา RXD ใหม่ แต่ถ้าพบ Start bit ก็จะเก็บข้อมูลทั้งหมดที่เข้ามาโดยเลื่อนข้อมูลเข้าไปยัง Input Shift Register ที่มีสัญญาณควบคุมการเลื่อนข้อมูล (shift) ส่งมาจาก RX control ในตอนเริ่มต้นการรับข้อมูลจะมีการเขียนข้อมูล 1FFH ไปเก็บใน Input Shift Register ขณะที่ข้อมูลถูกเลื่อนเข้าไปทางขวาของ Input Shift Register ก็จะมี 1 ถูกเลื่อนออกไปทางซ้ายทุกครั้งที่มีข้อมูลเข้ามา เมื่อ Start bit ที่รับเข้ามาถูกเลื่อนไปถึงซ้ายสุดของ Input Shift Register ก็จะมีสัญญาณไปบอก RX Control หลังจากข้อมูลบิทสุดท้ายเข้ามาแล้วก็จะโหลด (Load) เอาข้อมูล 8 บิท ไปเก็นในรีจิสเตอร์ SBUF พร้อมทั้ง Set ค่าในบิท RI และ RB8 ของรีจีสเตอร์ SCON แต่การโหลดข้อมูลไปเก็บนี้จะเกิดขึ้นได้ก็ต่อเมื่อ

1. RI = 0 และ
2. SM2 = 0 หรือถ้า SM2 = 1 จะต้องได้รับ stop bit เป็น 1

ถ้าไม่มีสภาวะใดสภาวะหนึ่งดังกล่างแล้ว ข้อมูลที่รับเข้ามาก็จะถูกทิ้งไปคือไม่โหลดไปเก็บในรีจีสเตอร์ SBUF ถ้ามีสภาวะดังกล่าวถูกต้อง stop bit จะถูกนำไปเก็บในรีจีสเตอร์ RB8 และ บิท RI จะเป็น 1 แต่ไม่ว่าทั้ง 2 กรณีจะเกิดหรือไม่ก็จะกลับไปสู่การตรวจสภาวะเปลี่ยนจาก 1 เป็น 0 ที่ขา RXD เพื่อรับข้อมูลต่อไป

ในการรับข้อมูลแบบอนุกรมโหมด 1 นี้ อัตราการส่งข้อมูลแต่ละบิท (Baud Rate) จะขึ้นกับอัตราการเกิด overfllow ใน Timer 1 .ในขณะที่ใช้ Timer 1 เป็นตัวกำหนด Baud Rate นี้จะต้อง Disable ไม่ให้เกิดการขัดจังหวะเนื่องมาจากการ Overflow Timer 1 อาจใช้ในโหมดของ Timer หรือ Counter ก็ได้ ซึ่งเมื่อการนับในรีจิสเตอร์ตัวนับมีค่าสูงสุดแล้วกลับมาเป็น 0 ก็จะเกิด Overflow เช่นเดียวกัน แต่โดยปกติแล้วจะใช้ Timer 1 นี้ในโหมดของ Timer ที่มีการทำงานแบบ Auto Reload โหมด 2 เพื่อว่าเมื่อค่าในการนับโดยรีจิสเตอร์ TL1 ถึงค่าสูงสุดก็จะโหลดค่าในรีจิสเตอร์ TH1 มาไว้ใน TL1 สำหรับเป็นค่าเริ่มต้นการนับต่อไปโดยที่ SMOD เป็นบิทหนึ่งในรีจิสเตอร์ PCON

2.6 พอร์ท 8255

8255 เป็นไอซี LSI ขนาด 40 ขา ดังรูปที่ 2.21 ซึ่งแสดงตำแหน่งของขาต่าง ๆ ทั้ง 40 ขาสำหรับรูปที่ 36 แสดงแผนผังภายในของ 8255

ซึ่ง 8255 นี้มีพอร์ทสำหรับส่งข้อมูลอยู่ด้วยกัน 3 พอร์ท มีชื่อดังนี้ พอร์ A,B และ C โดยพอร์ C นี้จะแบ่งออกเป็น 2 ส่วน คือ พอร์ท C บน (CLO) กับพอร์ท C ล่าง (CHI) และยังมีอีกพอร์ทหนึ่ง ซึ่งทำหน้าที่ควบคุมการทำงานของพอร์ท A,B และ C โดยรับคำสั่งมาจาก CPU พอร์ทนี้เรียกว่าพอร์ทควบคุม (Control port) การทำงานของพอร์ท จะถูกกำหนดโดย CPU โดย CPU จะส่งข้อมูลทางดาต้าบัส (Data Bus) ให้แก่พอร์ทควบคุมหน้าที่ของขาต่าง ๆ

CS (Chip Select)

ขาใช้ในการเลือกว่าจะให้ 8255 ตัวนี้ทำงานหรือไม่ โดยถ้าได้รับลอจิก 0 จะทำให้ 8255เชื่อมต่อเข้ากับระบบบัสต่าง ๆ ของ CPU แต่ถ้าเป็นลอจิก 1 จะอยู่ในสภาวะ High Impednce

RD (Read Enable)

ถ้าได้รับลอจิก 0 และ CS ได้รับลอจิก 0 เช่นกัน แสดงว่า 8255 ทำการส่งข้อมูลจากพอร์ทที่ CPU ต้องการติดต่อด้วยนั้นให้แก่ CPU ทางดาต้าบัส

WR (Write Enaqble)

ถ้าได้รับลอจิก 0 พร้อมกับ CS แล้ว 8255 จะส่งข้อมูลจากดาต้าบัสของ CPU ออกไปยังพอร์ทที่ CPU กำหนดไว้

RESET

ทำการ Reset 8255 เข้าสู่โหมดอินพุท ทุก ๆ พอร์ท และเคลียร์สถานะต่าง ๆ ของ 8255

D0-D7

เป็น Data Bus ที่ใช้รับส่งข้อมูลกับ CPU

A0-A1

คือขาแอดเดรสเลือกพอร์ทที่ CPU ต้องการติดต่อ

00 = พอร์ท A

01 = พอร์ท B

10 = พอร์ท C

11 = พอร์ทควบคุม

PA0-PA7

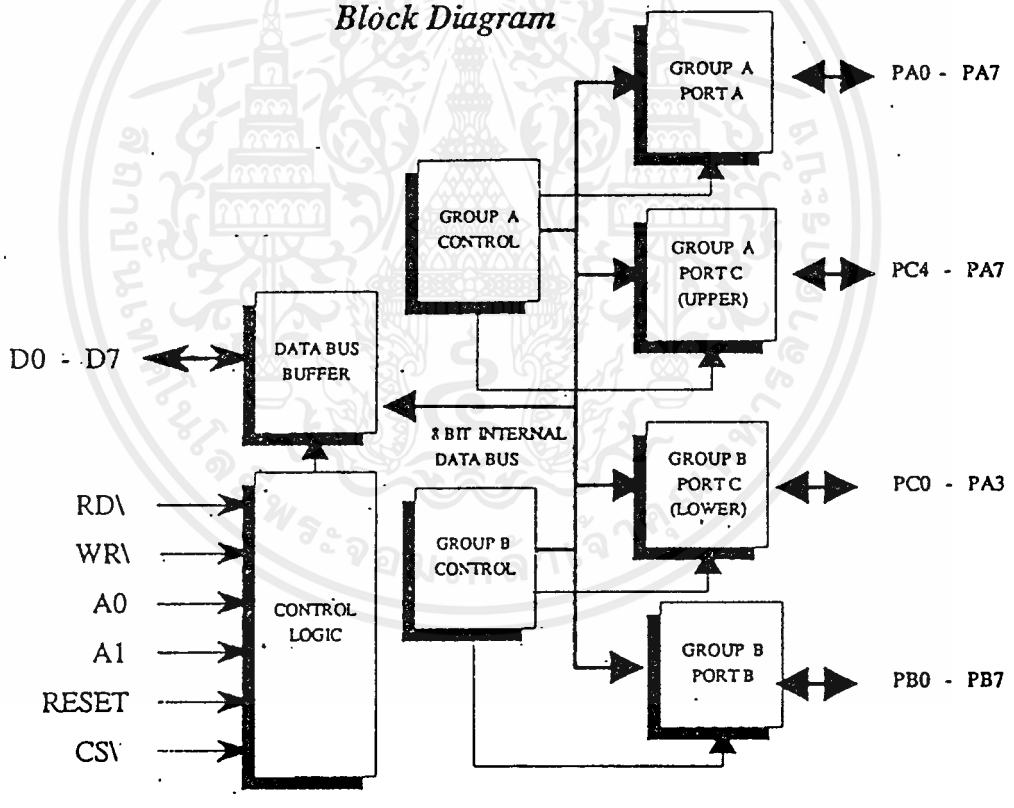
เป็นขาสัญญาณของพอร์ท A

PB0-PB7

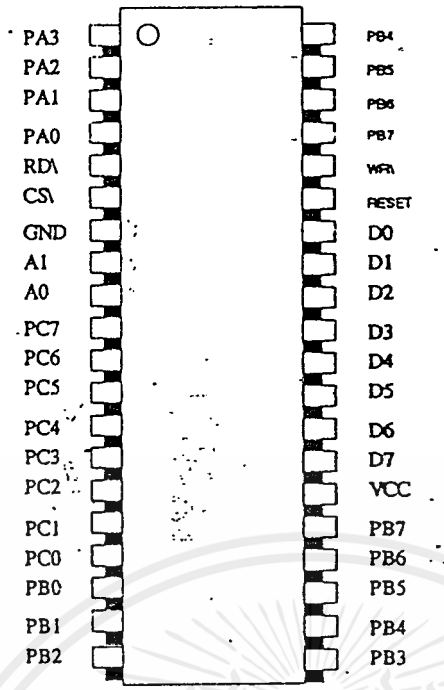
เป็นขาสัญญาณของพอร์ท B

PC0-PC7

เป็นขาสัญญาณของพอร์ท C โดยแยกเป็น PC0-PC3 และ PC4-PC7 โดยสามารถแยกการทำงานได้โดยอิสระ



รูปที่ 2.19 โครงสร้างของ 8255



Pin Names

D0 - D7	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
CS\	CHIP SELECT
RD\	READ INPUT
WR\	WRITE INPUT
A0 - A1	PORT ADDRESS
PA0 - PA7	PORT A
PB0 - PB7	PORT B
PC0 - PC7	PORT C

รูปที่ 2.20 การวางขาของ 8255

พอร์ทควบคุม

เป็นพอร์ทการกำหนดการทำงานของ 8255 โดยควบคุมจาก CPU ทำการส่งรหัสควบคุม ผ่านทาง ดาต้าบัสมายังพอร์ทควบคุมของ 8255 รหัสควบคุมนี้มีขนาด 1 ไบท์เรียกว่า Control byte

การใช้งาน 8255

8255 แบ่งการทำงานออกเป็น 3 โหมด ดังนี้

โหมด 0 เป็นโหมดอินพุทหรือเอาท์พุทพอร์ทอย่างใดอย่างหนึ่ง ซึ่งเรียกว่า Simple I/0 Port และสามารถทำงานได้ทั้ง 3 พอร์ท คือ A, B และ

โหมด 1 เป็นโหมดอินพุทหรือเอาท์พุทอย่างใดอย่างหนึ่ง ซึ่งทำงานแบบ handshaking ซึ่งสามารถทำงานได้เฉพาะพอร์ท A และ B การทำงานแบบ Handshaking ก็คือระหว่าง CPU กับพอร์ทและอุปกรณ์ภายนอกนั้น ขณะที่รับส่งข้อมูลกัน นอกจากจะรับข้อมูลกันแล้ว ยังต้องมีสัญญาณในการตอบรับในแต่ละครั้งของการรับส่งข้อมูลกันแล้ว โดยผู้รับกับผู้ส่งจะต้องทำงาน

สัมพันธ์กันตลอดเวลา โดยพอร์ A และ B ใช้ในการรับส่งข้อมูล ส่วนพอร์ท C จะเป็นรับและส่งสัญญาณควบคุม

โหมด 2 ในโหมด 2 นี้ เป็นการรับส่งข้อมูลแบบ Handsbking เช่นกัน โดยในโหมดนี้ จะใช้พอร์ท A เท่านั้นในการใช้งาน และพอร์ท A ในโหมดนี้อินพุทและเอาท์พุทพอร์ท (Bidirectional) และใช้พอร์ท C เป็นพอร์ทในการส่งสัญญาณควบคุม ดังตารางข้างล่างการใช้งานในโหมดนี้ ซึ่งใช้เพียงพอร์ท A ขณะเดียวกันนี้ สามารถใช้พอร์ท B ในการทำงานโหมด 0 และ 1 ได้

2.7 สัญญาณนาฬิกา ds 1202

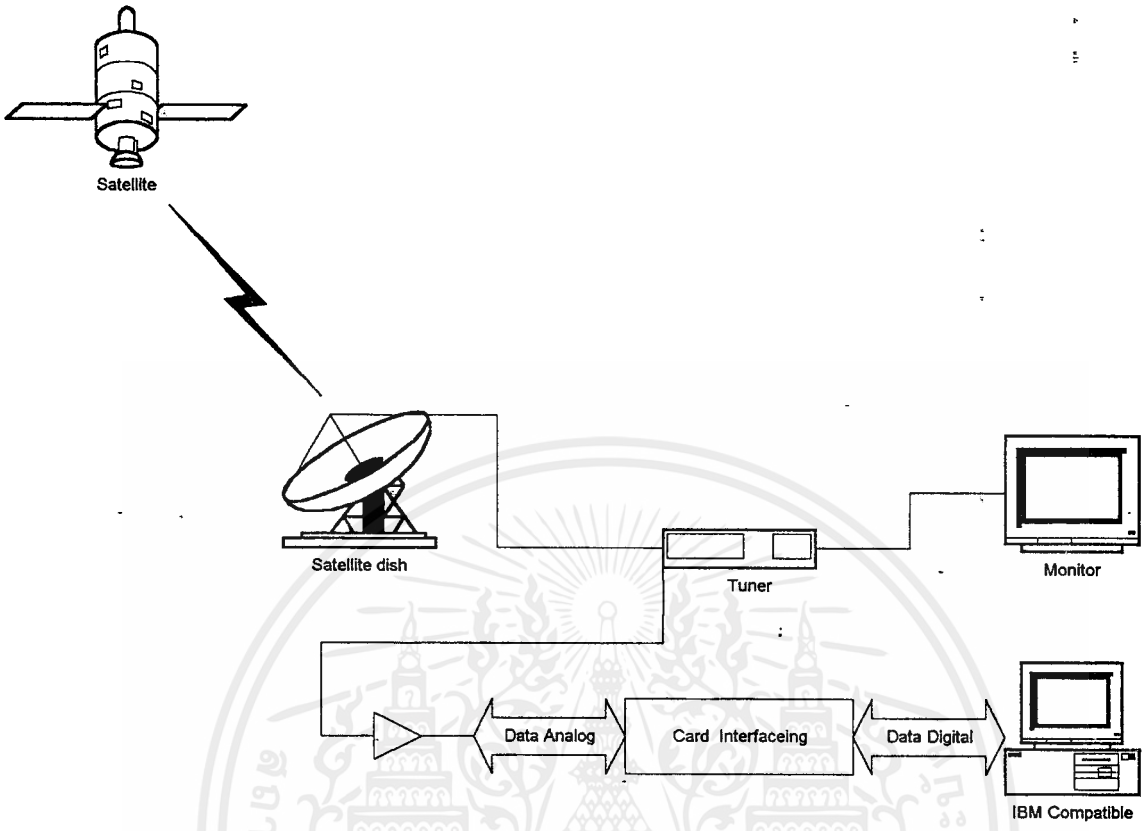
ds 1202 เป็นไอซี RTC (Real Time Clock) ที่สามารถนำมาใช้งานทำเป็นนาฬิกาที่แท้จริงคือไม่ต้องทำการปรับแต่งเมื่อทำการโปรแกรมตั้งเวลาไปแล้ว โดยในที่ ds 1202 จะทำการต่อร่วมกับไอซีเบอร์ MAX691 และคริสตอล 32.768 KHz และแบตเตอรี่แบบลิธี่ยม ds 1202 ทำการต่อร่วมกับ CPU แบบอนุกรมโดยสามารถทำได้ด้วยการใช้สายเพียง 3 เส้นคือ ขา RST (Reset) ขา I/O (Data Link) และขา SCLK (Serial Clock) ขาสัญญาณทั้ง 3 นี้จะทำการต่อเข้ากับขา P1.6 , P1.4 , P1.5 ของ CPU เมื่อต้องการทราบค่าของเวลา CPU จะทำการอ่านจาก RTC

บทที่ 3
วิธีการดำเนินงาน

เนื่องจากสัญญาณความถี่ขาลง (Downlink) ของสัญญาณดาวเทียมนั้นมีความถี่สูงมากดังนั้นในการรับสัญญาณจึงมีความผิดเพี้ยนอันเนื่องมาจากสภาวะแวดล้อมต่างๆได้ง่ายซึ่งอาจมีผลทำให้ข้อมูลที่ได้รับจากสัญญาณดังกล่าวมีความผิดพลาดเกิดขึ้นได้ซึ่งจะทำให้เกิดปัญหาอื่นๆตามมาเมื่อนำเอาข้อมูลที่ผิดพลาดนี้ไปใช้งานต่างๆต่อไป ในการทำปริญญานิพนธ์นี้มีจุดมุ่งหมายเพื่อสร้างเครื่องมือที่จะใช้ในการบันทึกถึงการเปลี่ยแปลงของสัญญาณที่ได้รับโดยการออกแบบสร้างวงจรเพื่อใช้งานร่วมกับเครื่องคอมพิวเตอร์ให้เป็นเครื่องมือในการบันทึกข้อมูลดังกล่าว

3.1 หลักการทำงาน

การทำงานของโครงงานโดยรวมจะประกอบด้วยส่วนที่เป็นฮาร์ดแวร์และซอฟแวร์ ส่วนของทางด้านฮาร์ทแวร์จะมีหลักการทำงานคือทำการขยายสัญญาณอินพุทที่มีความแรงของสัญญาณต่ำๆให้มีความแรงสูงขึ้นและทำการเปลี่ยนสัญญาณที่ทำการขยายแล้วจากสัญญาณอนาลอกไปเป็นสัญญาณดิจิตอลจากนั้นจะทำการนำเอาสัญญาณดิจิตอลที่ได้นี้ป้อนเข้าสู่เครื่องคอมพิวเตอร์ทาง พอร์ท RS232c เพื่อเป็นข้อมูลที่จะใช้ในการจำลองรูปสัญญาณที่จะทำการแสดงออกทางจอภาพ

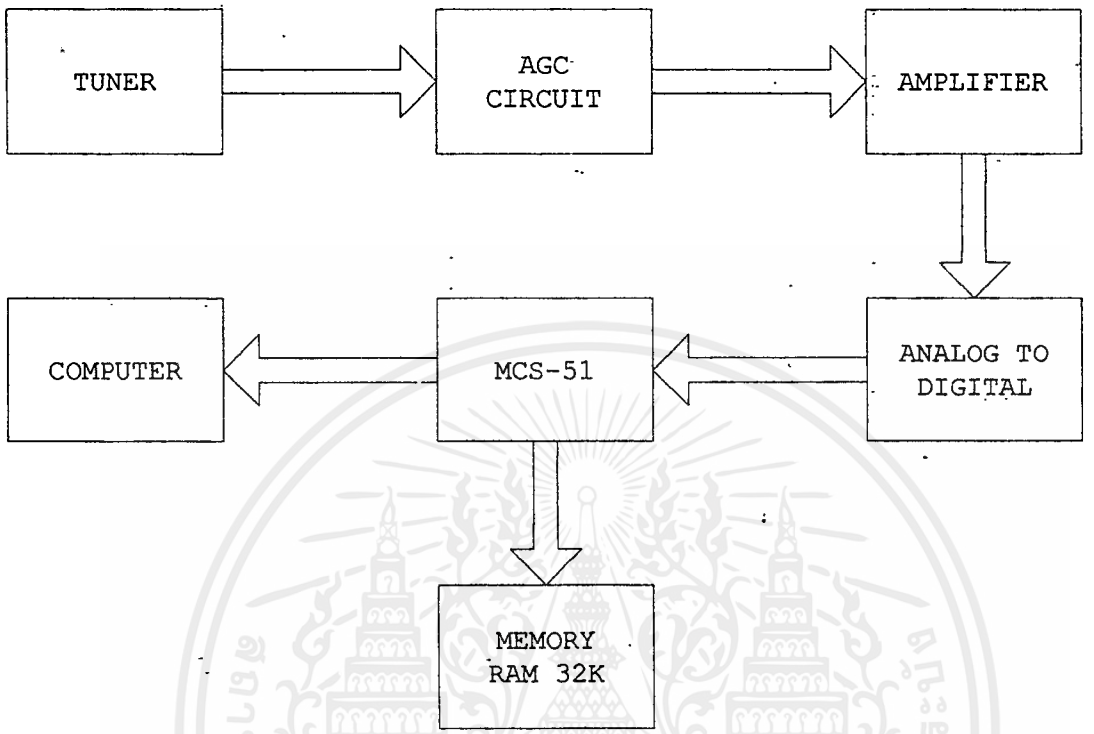


รูปที่ 3.1 แสดงการนำเครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมไปใช้งาน

เนื่องจากสัญญาณที่เครื่องคอมพิวเตอร์สามารถรับรู้และนำไปทำการบันทึกและประมวลผลได้นั้นจะต้องเป็นที่อยู่ในรูปของสัญญาณดิจิตอลเท่านั้นแต่สัญญาณที่ได้จากจานรับสัญญาณดาวเทียมเป็นสัญญาณที่อยู่ในรูปของสัญญาณอนาลอกจึงมีความจำเป็นการการที่เราจะต้องทำการเปลี่ยนสัญญาณจากสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอลเสียก่อนจึงจะสามารถนำไปใช้งานกับเครื่องคอมพิวเตอร์ได้แต่ปัญหาอันเนื่องมาจากว่าสัญญาณของความถี่ขาลงของสัญญาณดาวเทียมจะมีความถี่สูงมากคืออยู่ในช่วงความถี่ประมาณ 3.7 - 4.2 GHz สำหรับสัญญาณของย่าน C Band และความถี่สูงถึง 11.7 - 12.2 GHz สำหรับความถี่ย่าน KU Band และมี Bandwidth ประมาณ 500 MHz ซึ่งจะพบว่าเรายังไม่มีอุปกรณ์ที่สามารถที่จะทำการแปลงสัญญาณอนาลอกที่มีความถี่สูงขนาดนั้นให้เป็นสัญญาณดิจิตอลได้

ดังนั้นในการออกแบบวงจรที่ทำการสร้างขึ้นสำหรับทำโครงงานนี้จึงไม่สามารถที่จะนำเอาสัญญาณจาก LNB ของภาครับสัญญาณมาใช้เป็นสัญญาณอินพุทให้วงจรได้โดยตรงและเนื่องจากในการทำโครงงานนี้มีวัตถุประสงค์ที่จะสร้างเครื่องบันทึกการเปลี่ยนแปลงของสัญญาณ ดังนั้นในการทำโครงงานนี้จึงจะใช้สัญญาณจากวงจร Automatic Cain Control (AGC.) จากทางจูนเนอร์ของเครื่องรับสัญญาณดาวเทียมให้เป็นสัญญาณอินพุทให้แก่วงจรโดยยึดหลักที่ว่าการทำงานของวงจร AGC. นี้จะเป็นตัวควบคุมอัตราการขยายของภาคขยายสัญญาณต่างๆในเครื่องรับสัญญาณเพื่อที่จะทำให้ไม่เกิดปัญหาในกรณีที่สัญญาณมีความแรงมากหรือน้อยเกินไป โดยที่ถ้าสัญญาณที่เข้ามามีความแรงมากก็จะไปทำการลดอัตราการขยายลงและถ้าสัญญาณที่เข้ามามีความแรงต่ำก็จะส่งสัญญาณไปบอกให้เพิ่มอัตราการขยายขึ้นดังนั้นจะเห็นได้ว่าสัญญาณจากวงจร AGC. นี้จะทำให้เราทราบถึงการเปลี่ยนแปลงของความแรงของสัญญาณที่รับเข้ามาได้ดังนั้นเราจึงใช้สัญญาณจากวงจร AGC. นี้เป็นสัญญาณอ้างอิงถึงการเปลี่ยนแปลงของสัญญาณที่ได้รับมาสัญญาณที่ได้จากวงจร AGC. นี้ก็มีความถี่ไม่สูงมากจึงสามารถที่จะทำการแปลงสัญญาณจากสัญญาณอนาลอกให้เป็นสัญญาณดิจิตอลเพื่อที่จะนำไปใช้งานกับเครื่องคอมพิวเตอร์ได้ แต่สัญญาณที่ได้จากวงจร AGC. นี้มีระดับความแรงของสัญญาณต่ำมากจึงต้องมีการเข้าวงจรขยายสัญญาณก่อนจึงจะสามารถนำสัญญาณไปเข้าวงจรแปลงสัญญาณจากอนาลอกเป็นสัญญาณดิจิตอลได้ หลังจากที่สัญญาณมีความแรงเพียงพอที่จะใช้งานกับวงจรเปลี่ยนสัญญาณอนาลอกเป็นสัญญาณดิจิตอลแล้วก็จะทำการป้อนสัญญาณให้วงจรจากจะทำให้เราได้สัญญาณเป็นดิจิตอลขนาด 8 บิทแบบขนานแต่ในการทำโครงงานนี้จะทำการติดต่อกับ Port RS232C ซึ่งเป็น Port แบบอนุกรมดังนั้นจึงต้องนำสัญญาณที่ได้ไปทำการเปลี่ยนจากสัญญาณดิจิตอลแบบขนาน 8 บิตให้เป็นแบบอนุกรม 8 บิต ซึ่งในการเปลี่ยนให้ข้อมูลเป็นแบบอนุกรมนี้จะต้องทำการจัดให้มีอัตราความเร็ว 9600 บิตต่อวินาที 1 เฟรมของข้อมูลจะประกอบไปด้วยข้อมูลขนาด 8 บิตและรวมกับบิตเริ่มต้นกับบิตลงท้ายอีก 1 บิต ดังนั้นในการส่งข้อมูล 1 เฟรมจะเป็นการส่งชุดของข้อมูลขนาด 10 บิตดังนั้นในการติดต่อกับพอร์ตอนุกรมนี้จะสามารถทำการส่งข้อมูลได้ 960 ไบต์ต่อวินาที

BLOCK DIAGRAM ของวงจร



รูปที่ 3.2 Block Diagram ของวงจร

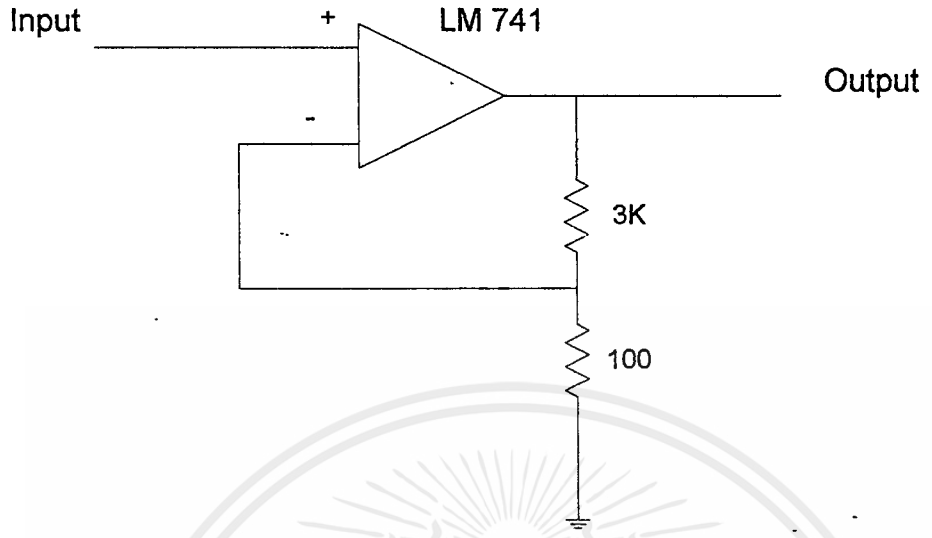
3.2 หลักการออกแบบวงจรขยายสัญญาณ AGC

ในที่นี้เนื่องจากสัญญาณอินพุทมีระดับต่ำมากจึงต้องมีการขยายสัญญาณให้มีระดับความแรงเพียงพอในการใช้งานกับไอซีที่เปลี่ยนสัญญาณจากอนาลอกเป็นดิจิตอล จากวงจรเราใช้ไอซีเบอร์ LM741 ซึ่งเป็นไอซีประเภทออปแอมป์โดยทำการจัดวงจรให้มีการขยายเป็นแบบไม่กลับเฟสซึ่งสามารถทำการคำนวณได้จากสมการ

$$A_V = 1 + R_F / R_I$$

โดยที่ในวงจรนี้จะทำการกำหนดค่าให้ R_F มีค่า 3K และ R_i มีค่า 100 โอห์ม และมีอีกส่วนเป็นวงจรกันชนเพื่อทำการแมทซ์ชิ่งโดยที่วงจรกันชนนี้จะมีอัตราการขยายเท่ากับหนึ่งดังรูปที่ 3.3

3.3 หลักการออกแบบวงจรการเปลี่ยนสัญญาณจากแบบอนาลอกไปเป็นแบบดิจิตอล



รูปที่ 3.3 แสดงวงจรขยายสัญญาณ เอจีซี

ในวงจรนี้อาศัยไอซีเบอร์ ADC0804 ซึ่งเป็นไอซีแปลงสัญญาณจากอนาลอกเป็นสัญญาณดิจิตอล โดยที่ ADC0804 นี้เป็นไอซีประเภท CMOS ขนาด 8 บิตซึ่งทำงานแบบ Successive Approximation สามารถทำงานได้ทั้งหมดภายในตัวมันเอง ภายในตัวไอซีประกอบไปด้วยวงจรสร้างสัญญาณนาฬิกาและกำหนดค่าความถี่ที่ได้จาก R1 และ C1

สัญญาณนาฬิกาสูงสุดที่ใช้ได้กับไอซี 0804 ก็คือ 640 กิโลเฮิรตซ์ซึ่งจะทำให้ใช้เวลาในการแปลง 100 ไมโครวินาที โดยมีอัตราการแซมปลิ้งที่ 10 กิโลเฮิรตซ์แต่ในการใช้งานในที่นี้จะไม่ใช้ค่าสูงสุดเพราะจะทำให้การส่งข้อมูลมีค่าเกิน 960 แซมปลิ้งต่อวินาที ในโหมดการทำงานด้วยตนเองไม่ต้องมีสัญญาณควบคุมจากภายนอก ขา READ (ขา 2) กับขา ChipSelect (ขา 1) จะถูกต่อลงกราวด์ ส่วนขา Interrupt Output (ขา 5) จะถูกต่อไปยังขา Write Data Input (ขา 3) เพื่อให้การแปลงสัญญาณออกไปภายนอกเป็นไปอย่างอัตโนมัติ

3.4 การออกแบบ SOFTWARE

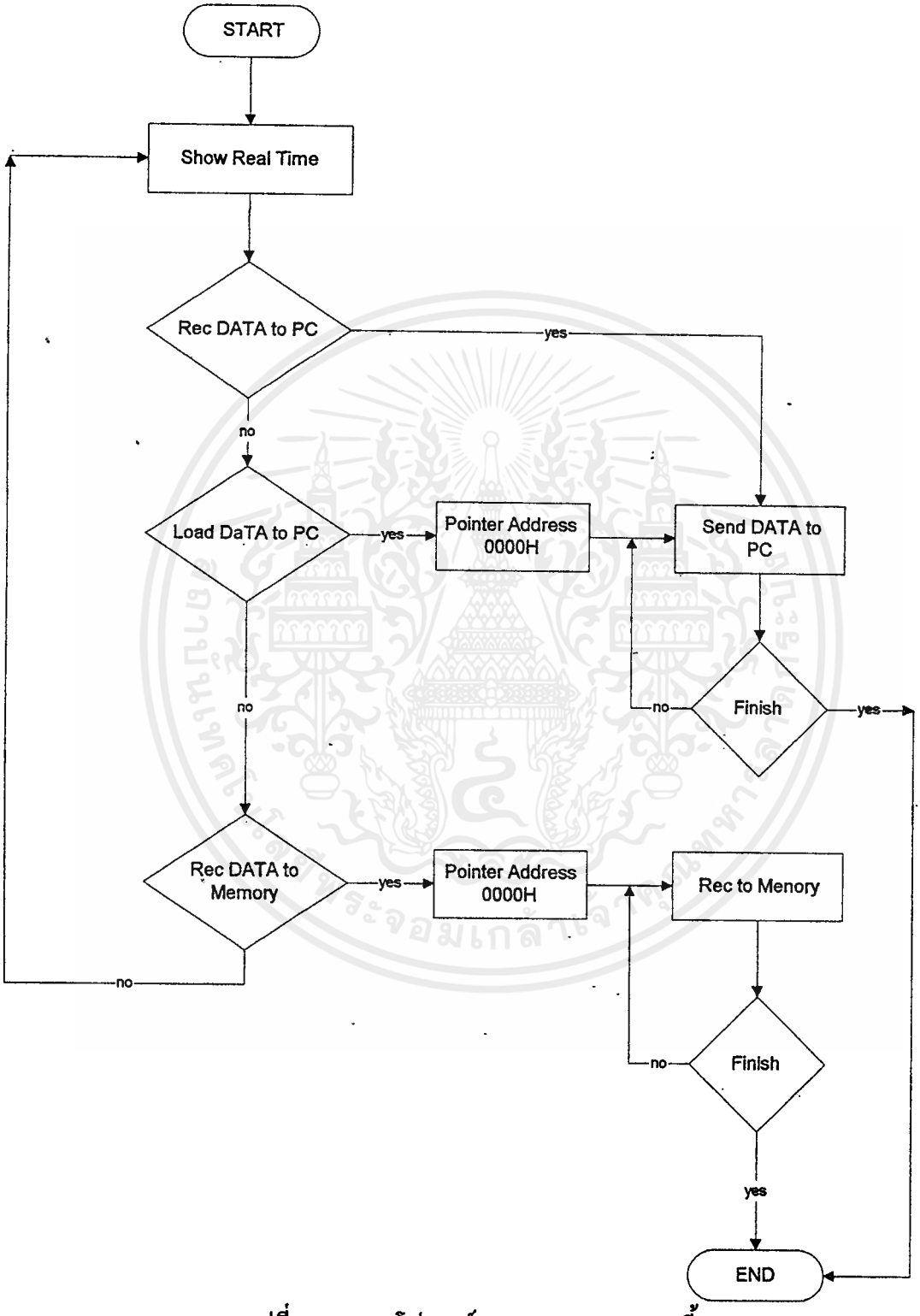
SOFTWARE ที่ใช้จะแบ่งออกเป็น 2 ส่วนใหญ่คือส่วนของภาษา Assembly ที่ใช้อ่านข้อมูลและควบคุมอัตราการ Sampling บน MCS-51 และส่วนของภาษา PASCAL ที่ใช้ในการแสดงผลและวิเคราะห์หาค่าการกระจายแบบ STD บนเครื่องคอมพิวเตอร์

ส่วนของภาษา Assembly ที่ใช้ในการควบคุมการทำงานของ MCS-51 สามารถแบ่งออกเป็นส่วนต่างๆได้ดังต่อไปนี้คือ

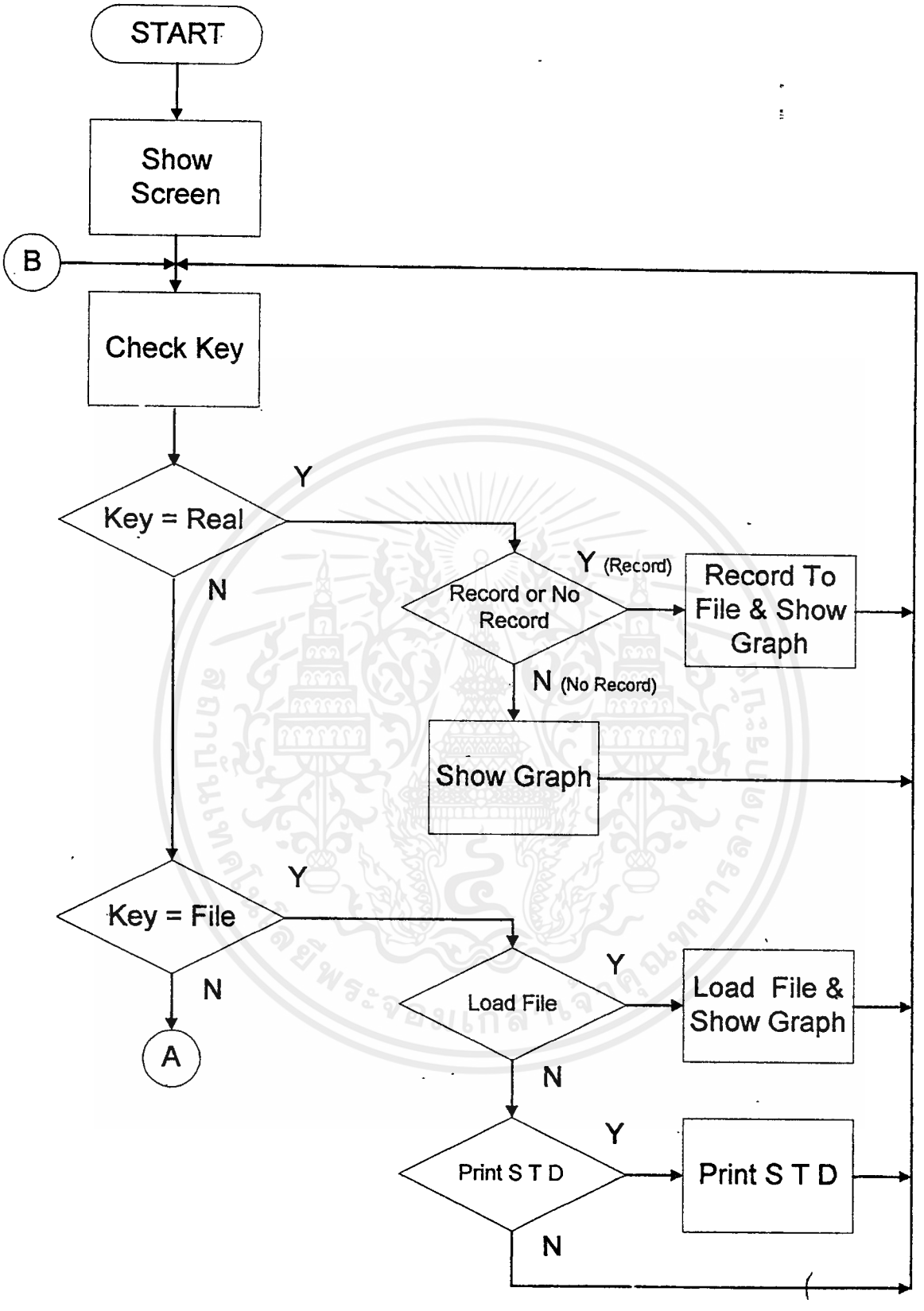
- ส่วนทำการอ่านข้อมูลจากวงจร A/D
- ส่วนการติดต่อ RS-232C
- ส่วนการบันทึกข้อมูล
- ส่วนการส่งข้อมูลแบบ REAL TIME
- ส่วนการแสดงผลการทำงานผ่านทาง LCD
- ส่วนการควบคุม Chip RTC DS-1202
- ส่วนการรับคำสั่งจาก KEYBOARD

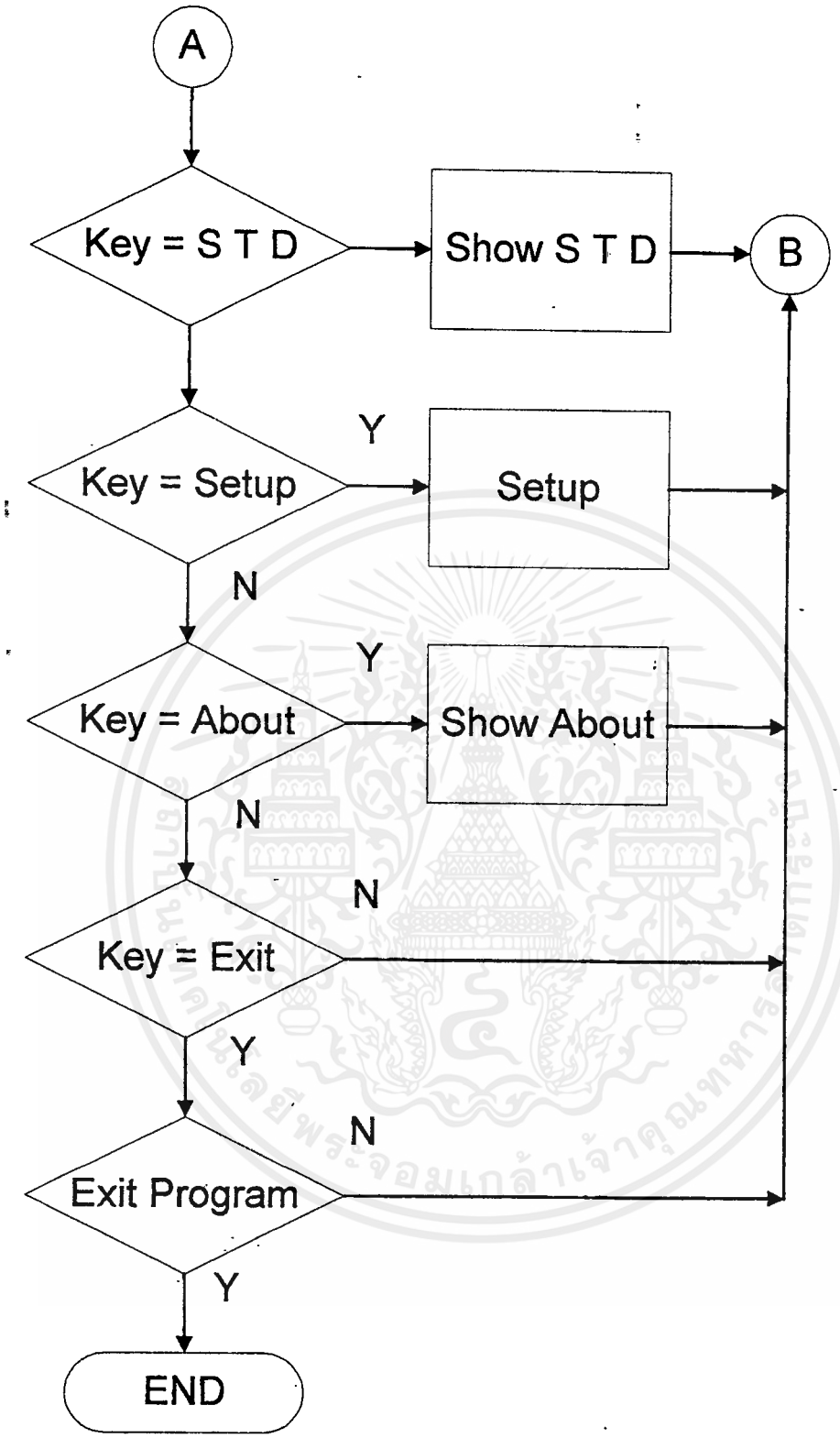
ส่วนของภาษาปาสคาล เป็นส่วนควบคุมการแสดงผลและวิเคราะห์ข้อมูลบนเครื่องคอมพิวเตอร์สามารถทำการแบ่งออกเป็นส่วนต่างๆดังต่อไปนี้

- การรับข้อมูลจากพอร์ท RS232c
- การบันทึกข้อมูลลงบนเครื่องคอมพิวเตอร์
- การแสดงผลรูปกราฟออกทางจอภาพ
- การหาค่า V_{p-p} ในหน่วยเดซิเบล
- การหาค่า STD



รูปที่ 3.4 แสดงโฟลชาร์ตของภาษาแอสเซมบลี้





รูปที่ 3.5 โฟลต์ชาร์ตโปรแกรมภาษาปาสคาล

บทที่ 4

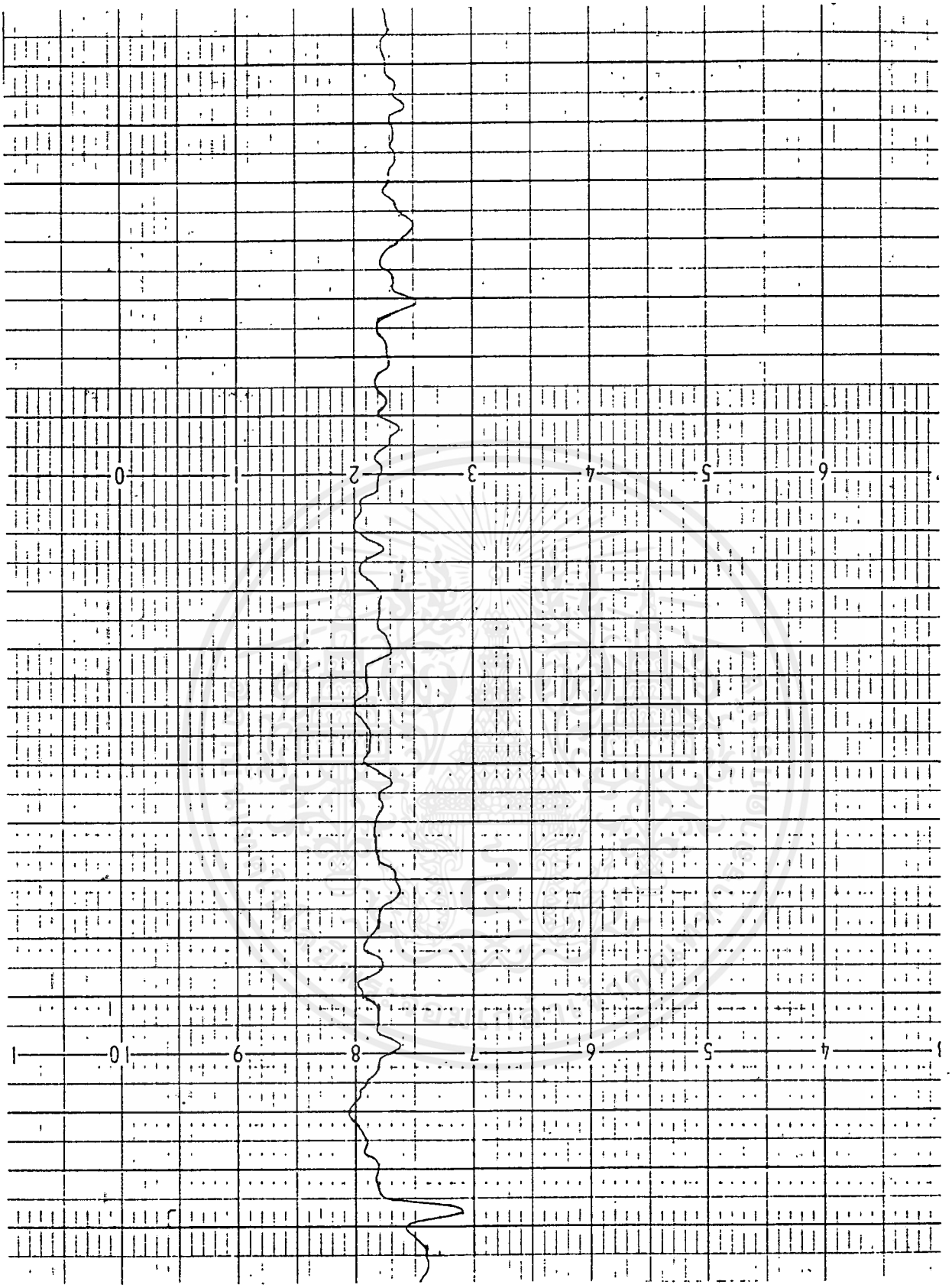
ผลการทดลอง

4.1 การบันทึกลงในหน่วยความจำแล้วนำมาแสดงผลบนเครื่องคอมพิวเตอร์

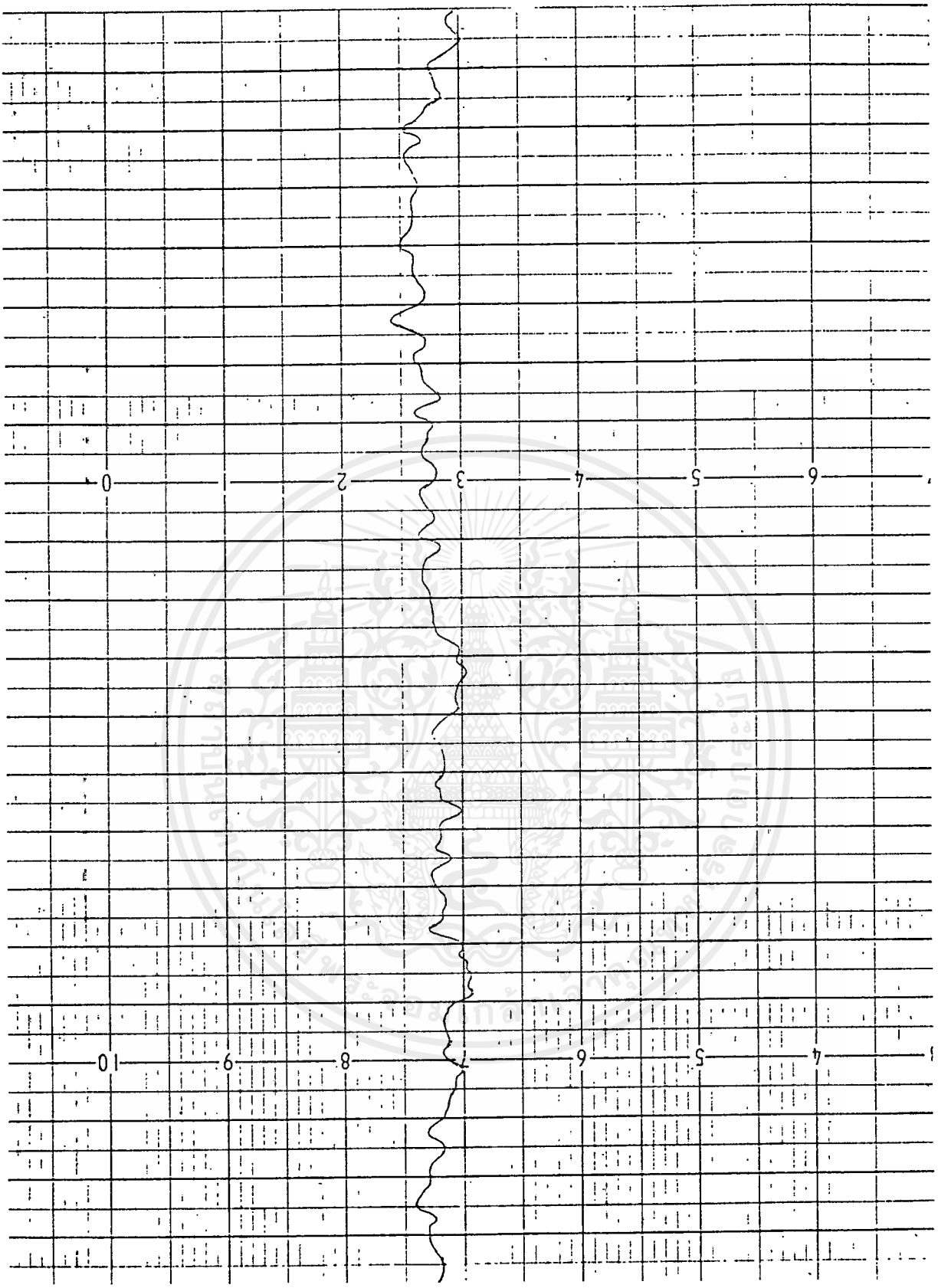
การทดลองดังกล่าวพบว่าสามารถทำการบันทึกข้อมูลได้เป็นระยะเวลาประมาณ 27 นาที ทั้งนี้ขึ้นอยู่กับจำนวนหน่วยความจำของตัวไมโครคอนโทรลเลอร์ที่นำมาใช้ซึ่งในการทดลองนี้มีหน่วยความจำ 32 กิโลไบต์ ถ้าต้องการให้บันทึกได้นานกว่านี้สามารถกระทำได้โดยการเพิ่มหน่วยความจำของเครื่องเข้าไปอีก

4.2 การบันทึกลงในเครื่องคอมพิวเตอร์โดยตรง

การบันทึกข้อมูลลงในเครื่องคอมพิวเตอร์โดยตรงสามาถทำการบันทึกและจำลองรูปแบบสัญญาณให้เห็นถึงการเปลี่ยนแปลงของระดับสัญญาณเมื่อเวลาเปลี่ยนไปได้ซึ่งข้อจำกัดในการเก็บข้อมูลขึ้นอยู่กับขนาดของความจุของฮาร์ดดิสส์ของเครื่องคอมพิวเตอร์ที่ทำการติดตั้งโปรแกรมซึ่งนอกจากนั้นแล้วยังสามารถทำการคำนวณหาค่าทางสถิติที่ต้องการได้



รูปที่ 4.1 กราฟที่ได้จากเครื่องบันทึกแบบปากกา



รูปที่ 4.2 กราฟที่ได้จากเครื่องบันทึกแบบปากกา

ผลการเปรียบเทียบระหว่างค่าที่อ่านได้จากกราฟกับค่าที่อ่านได้จากโปรแกรม (หน่วยเป็นมิลลิโวลท์)

ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม	ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม	ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม
26	26	24	24	24	24
26	26	24	24	23	23
27	27	25	25	23	23
27	27	25	25	24	24
27	27	25	25	24	24
27	27	27	27	24	24
27	27	26	26	23	24
27	27	26	26	23	23
27	27	26	26	24	24
27	26	25	25	25	25
27	27	24	24	25	25
27	28	24	24	24	24
28	28	23	23	23	23
28	27	23	23	23	23
27	26	24	24	22	22
26	26	24	24	22	22
25	25	27	27	23	23
25	25	27	27	23	23
24	24	27	27	22	22
24	24	26	26	22	22
25	25	25	25	23	23
25	25	25	24	23	23
25	25	23	23	23	23
24	24	24	24	23	23
24	24	24	24	24	24
25	25	25	25	23	23
25	25	24	24	23	23
25	25	25	25	23	23
24	24	23	23	24	24
24	24	24	24	24	24
24	24	25	24	25	25
24	24	25	25	25	25
24	24	24	24	24	24

ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม	ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม	ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม
24	24	22	22	27	27
24	24	22	22	27	27
23	23	22	22	27	26
23	23	22	22	26	26
23	23	23	22	26	26
24	24	23	23	26	26
24	24	23	23	25	25
24	24	23	23	25	25
23	24	23	23	25	24
23	23	24	24	25	25
24	24	24	24	26	26
24	24	23	23	26	26
24	23	23	23	27	27
23	23	24	24	27	27
24	24	24	24	26	26
23	23	27	27	27	27
25	25	27	27	28	28
25	25	29	29	28	28
25	25	29	29	28	28
24	24	29	29	29	29
24	24	29	28	29	29
24	25	28	28	28	28
24	24	26	26	28	28
23	23	25	25	28	28
24	24	25	25	29	29
23	23	26	26	29	29
23	23	27	27	27	27
23	23	27	27	28	28
23	23	27	27	28	28
22	22	27	27	29	29
22	22	27	27	29	29

ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม	ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม	ค่าที่อ่านได้จากกราฟ	ค่าที่อ่านได้จากโปรแกรม
28	28	29	29	26	27
28	28	28	28	26	26
27	27	29	29	27	27
28	28	28	28	28	28
29	29	30	30	28	28
29	29	30	30	29	29
29	29	31	31	29	29
28	28	32	32	28	28
29	29	31	31	28	28
29	29	31	31	28	28
29	29	32	32	27	27
29	29	32	32	27	27
29	28	31	32	29	29
30	30	31	31	29	29
29	29	29	29	30	30
29	29	30	30	29	29
28	28	30	30	28	28
28	28	30	30	28	28
29	29	29	29	28	28
29	29	31	31	30	30
30	30	31	31	30	30
30	30	30	30	31	31
30	30	30	29	31	31
30	30	28	28	31	31
31	31	29	29	31	31
31	30	28	28	30	30
30	30	28	28	30	30
29	29	27	27	29	29
29	29	27	27	28	28
30	30	27	27	28	28
		27	27	29	29
		26	26	29	29

จากผลการทดลองที่ได้เป็นการอ่านค่าที่ทำการบันทึกด้วยเครื่องบั่นทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมซึ่งทำการบันทึกเมื่อวันที่ 19 เมษายน 1997 เวลา 19 นาฬิกา 54 นาที ถึงเวลา 19 นาฬิกา 56 นาที รวมทำการบันทึกเป็นเวลา 2 นาที โดยที่ในวันดังกล่าวสภาพอากาศเป็นปกติ

บทที่ 5
บทสรุปและข้อเสนอแนะ

5.1 สรุปผลการทดลอง

จากผลการทดลองพบว่าสัญญาณที่ทำการบันทึกได้โดยใช้เครื่องคอมพิวเตอร์กับที่ได้จากเครื่องบันทึกกระดาษมีรูปสัญญาณใกล้เคียงกันแต่ไม่เหมือนกันทั้งนี้เนื่องจากหน่วยที่ทำการบันทึกในแกน X แตกต่างกันคือในเครื่องบันทึกด้วยกระดาษมีการวัดหน่วยเป็น CM / Sec แต่ในเครื่องบันทึกด้วยคอมพิวเตอร์นี้ในแกน X มีหน่วย เป็น 40Point / Sec และค่าโวลท์เตจที่วัดได้มีค่าใกล้เคียงกันเนื่องมาจากการบันทึกนี้เป็นการบันทึกในลักษณะของดิจิตอลคือการสุ่มเอนเวลลอบของสัญญาณทำให้มีความแตกต่างกันเล็กน้อยเนื่องจากเครื่องบันทึกด้วยกระดาษทำงานในลักษณะ Real Time และเป็นการบันทึกสัญญาณในแบบอนาลอก

5.2 ปัญหาที่พบในการทดลอง

1.มีปัญหาเกี่ยวกับสัญญาณที่จะนำมาทำการวัดทั้งนี้เนื่องจากว่าสัญญาณ Down Link ของดาวเทียมมีความถี่สูงมากจึงเสียเวลาในการศึกษาไปเป็นอันมากว่าจะทำการวัดสัญญาณอย่างไรจึงจะสามารถรู้ว่าสัญญาณมีการเปลี่ยนแปลงเกิดขึ้น

2.สัญญาณจากวงจร AGC มีปัญหาค่อนข้างมากทั้งนี้เนื่องจากว่าสัญญาณมีระดับแรงดันที่ต่ำมากทำให้วงจรที่ทำหน้าที่เป็น A/D ไม่สามารถจัดระดับการเปลี่ยนแปลงของสัญญาณได้

3.ในการออกแบบวงจรขยายสัญญาณ AGC . พบว่าเมื่อทำการทดลองขยายสัญญาณโดยการใช้ OP AMP พบว่าวงจรสามารถทำการขยายสัญญาณให้ได้ตามที่ต้องการ แต่เมื่อนำเอาส่วนของวงจรขยายนี้ไปทำการต่อรวมกับภาคอื่นๆ มีผลทำให้เกิดสัญญาณรบกวนเกิดขึ้นทำให้การทำงานทั้งหมดผิดพลาด

3.ในการทดลองกำหนดให้มีการ Sampling ที่ 20 จุดต่อวินาที ดังนั้นถ้าสัญญาณที่เข้ามามีการเปลี่ยนแปลงอย่างรวดเร็วจะทำให้ข้อมูลที่ได้ออกมามีความผิดพลาด

4.ใช้เวลาในการศึกษา Chip DS-1202 ที่นำมาใช้เป็น RTC

5.ในการบันทึกลงในหน่วยความจำโดยไม่มีเครื่องคอมพิวเตอร์ต่อร่วมนั้นทำให้หน่วยความจำเต็มเร็วมากจนไม่สามารถนำไปใช้งานได้

5.3 การแก้ปัญหา

1.ทำการใช้สัญญาณจากวงจร AGC ในการตรวจสอบการเปลี่ยนแปลงของระดับสัญญาณที่ได้รับเข้ามาจากจูนเนอร์

2.ทำการขยายสัญญาณ AGC ให้มีระดับแรงดันสูงมากพอที่จะทำให้วงจร A/D ทำงานได้

3.ศึกษาการเขียนโปรแกรมภาษา Assambly เพื่อใช้ร่วมกับ Chip ต่างๆที่นำมาต่อร่วมด้วย

4.ทำการกำหนดให้ทำการ Sampling ที่ 20 จุดต่อวินาทีเพื่อที่จะให้การเก็บข้อมูลโดยไม่ทำการต่อร่วมเครื่องคอมพิวเตอร์สามารถทำได้เป็นเวลาประมาณ 30 นาที

5.4 แนวทางในการพัฒนา

1.ปรับปรุงหา A/D ที่ทำงานที่ความถี่สูงมากๆมาใช้แทน ADC0804 ที่ใช้อยู่ซึ่งจะสามารถที่จะทำการบันทึกสัญญาณที่เป็นสัญญาณ Down Link จริงๆของสัญญาณดาวเทียมได้

2.พัฒนาหาวงจรขยายที่มีการผิดเพี้ยนน้อยกว่านี้มาใช้

3.พัฒนาโปรแกรมโดยการปรับปรุงให้สามารถใช้ประสิทธิภาพของ MCS-51 ให้ได้มากกว่านี้

4.พัฒนาให้โปรแกรมโดยให้สามารถทำการพิมพ์ค่าที่ได้ในรูปแบบของกราฟออกทางเครื่องพิมพ์ได้ทุกชนิดไม่ว่าจะเป็น Dot Matrix หรือ Laser Printer

5.ปรับปรุง Software บนเครื่องคอมพิวเตอร์ให้สามารถทำการวิเคราะห์ข้อมูลได้มากกว่านี้

6.ปรับปรุง Software บนเครื่องคอมพิวเตอร์ให้มีการจัดเก็บข้อมูลที่ประหยัดเนื้อที่มากกว่านี้

เอกสารอ้างอิง

1. Abdulrahman Ali Aboudabra , Yoshiaki Moriya and Masamori Iida , The Signal Level Idicatio Method Tropospheric Scintillation of KU Band , Tokai University , 1996
2. สันติ สถิตวรรธนะ , กลยุทธ ฉิมแย้ม , พรสุข เทศเจริญ , นิภา ลีลารุจิ , ณรงค์ เหมกรณ์ , ระบบตรวจจับสัญญาณไฟฟ้าในระบบสื่อสาร , การประชุมทางวิชาการวิศวกรรมไฟฟ้า ครั้งที่19ณ มหาวิทยาลัยขอนแก่นหน้าที่ GM-24
3. อรลาภ แสงอรุณ , อุทัย ศรีธีระวิโรจน์ , Mana plamkulvaich , Pof. Yoshiaki Moriya , วิธีการแสดงปรากฏการณ์การเปลี่ยนแปลงอย่างกระทันหันของสัญญณดาวเทียมเนื่องจากชั้นบรรยากาศโทรโพสเฟียร์ , การประชุมทางวิชาการวิศวกรรมไฟฟ้า ครั้งที่19 ณ มหาวิทยาลัยขอนแก่น หน้าที่ GM-33
4. พงระพี เตชพาหงษ์ , แอดวานซ์เอ็มเอสดอส , ซีเอ็ดยูเคชั่น , 2533
5. สุรศักดิ์ สงวนพงษ์ , การเขียนโปรแกรมขั้นสูงแอดวานซ์เทอร์โบปาสคาล , ซีเอ็ดยูเคชั่น , 2532
6. Haward M. Berlin , Desingn of Op-Amp Circuit With Experiments , E&L Instrument Inc , 1980
7. ผ.ศ.พิพัฒน์ เลาหสงคราม , ภาษาแอสเซมบลี้สำหรับไมโครคอนโทรลเลอร์ MCS-48&MCS-51 , ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
8. ผ.ศ.พิพัฒน์ เลาหสงคราม , ไมโครคอนโทรลเลอร์ MCS-48&MCS-51 , ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

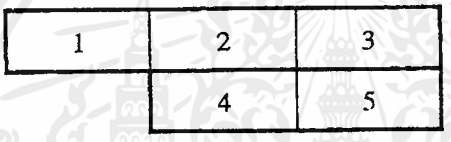
ภาคผนวก

การใช้งานเครื่องบันทึกการเปลี่ยนแปลงสัญญาณดาวเทียม

เครื่องบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียมนี้มีการทำงานแบ่งออกเป็น 2 โหมดด้วยกันคือการบันทึกลงในหน่วยความจำของเครื่องซึ่งจะทำการบันทึกค่าเป็นระดับแรงดันของสัญญาณโดยภายในเครื่องนี้จะประกอบไปด้วยหน่วยความจำ 32 กิโลไบต์ซึ่งจะสามารถทำการบันทึกสัญญาณได้เป็นเวลา 27 นาที และอีกโหมดเป็นการทำการบันทึกลงในเครื่องคอมพิวเตอร์ซึ่งสามารถทำการบันทึกเก็บลงเป็นแฟ้มข้อมูลและแสดงผลเป็นกราฟในลักษณะของการเปลี่ยนแปลงค่าแรงดันตามเวลาที่เปลี่ยนไป

การทำงานจะแบ่งออกเป็น 2 ส่วนคือส่วนของตัวเครื่องเองและส่วนของ Software ที่ทำการแสดงผลบนเครื่องคอมพิวเตอร์

ส่วนของ Hard Ware ประกอบไปด้วย Keyboard ควบคุมการทำงานดังรูป



การทำงานของแต่ละปุ่มมีหน้าที่ดังนี้

1.ปุ่ม LOAD ทำหน้าที่ส่งข้อมูลที่บันทึกได้ไปยังเครื่องคอมพิวเตอร์

2.ปุ่ม SEND ทำหน้าที่ส่งข้อมูลที่อ่านได้ไปยังเครื่องคอมพิวเตอร์โดยตรง

3.ปุ่ม ENTER ทำน้าที่ตรวจสอบว่าต้องการทำงานตามที่แสดงใน LCD

4.ปุ่ม RECORD ทำหน้าที่บันทึกสัญญาณลงในหน่วยความจำของเครื่องเอง

5.ปุ่ม Cancel ทำหน้าที่ยกเลิกการทำงานที่กำลังดำเนินการอยู่กลับสู่โหมดเตรียมพร้อม

การใช้งานโปรแกรมบันทึกการเปลี่ยนแปลงระดับสัญญาณดาวเทียม

การใช้งานโปรแกรมสามารถใช้ได้ทั้งจากคีย์บอร์ดหรือจากเมาส์โดยการใช้เมาส์ก็เพียงแต่เลื่อนตำแหน่งของเมาส์ไปที่ปุ่มเมนูแล้วกดเมาส์ก็จะเป็นการเรียกใช้เมนูนั้นหรือสามารถกดคีย์ตามตัวอักษรที่แสดงไว้ ก็จะเป็นการเรียกใช้งานได้เหมือนกับดังรูปที่ 1

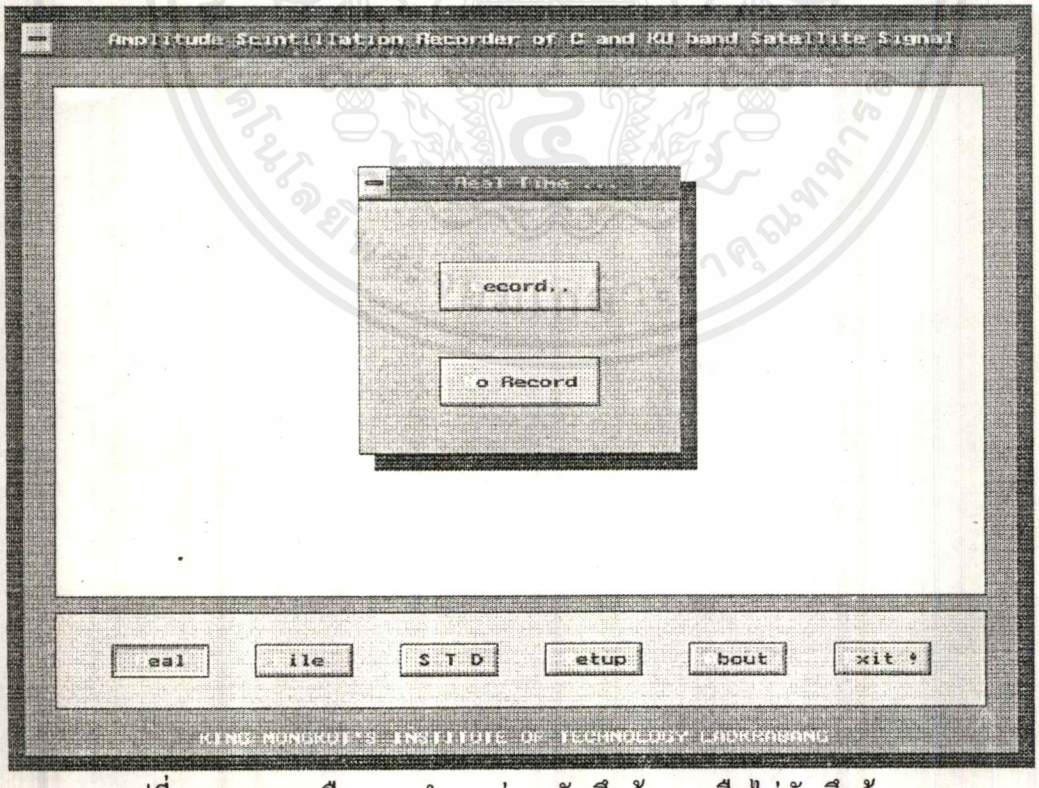
โดยถ้าจะทำการนำข้อมูลจากเครื่องบันทึกสัญญาณมาได้บนคอมพิวเตอร์ก็สามารถทำได้โดยเลื่อนตำแหน่งของเมาส์ไปที่ปุ่มเมนู Real และจะทำการกดเมาส์หรือกดตัวอักษร "R" ก็จะเป็นการเรียกใช้ เมนู Real โดยจะมีการแสดงเมนูย่อย ซึ่งจะเป็นการให้เรียกว่าจะบันทึกข้อมูลใส่ในคอมพิวเตอร์หรือจะใช้บันทึกข้อมูลก็ได้ ดังรูปที่2 ถ้าจะยกเลิกก็สามารถกดปุ่ม ESC ก็จะกลับไปที่เมนูหลักทันที ถ้าเลือก Record โปรแกรมก็จะให้ตั้งชื่อไฟล์ได้ 8 ตัวอักษร โดยไม่ต้องใส่ นามสกุล ดังรูปที่ 3 แล้วโปรแกรมก็จะนำข้อมูลที่อ่านได้ใส่ลงในไฟล์ (และแสดงผลออกทางจอภาพดังรูปที่4หรือถ้าเลือกNoRecordโปรแกรมก็จะนำข้อมูลมาแสดงที่หน้าจอคอมพิวเตอร์เท่านั้นโดยใช้บันทึกลงไฟล์เมื่อจะให้หยุดการบันทึกก็เป็นแค่กดคีย์ใดๆหรือกดเมาส์ก็จะหยุดทำการบันทึก

ถ้าจะนำข้อมูลที่บันทึกไว้แล้วมาแสดงอีกก็สามารถเรียกได้โดยการเลือกเมนูfileก็จะมีเมนูย่อยออกมาดังรูปที่5โดยจะแสดงชื่อไฟล์ปัจจุบันที่จะทำการloadข้อมูลออกมาซึ่งเราสามารถเปลี่ยนใหม่ได้โดยเลือกเมนู Change file menu ก็จะให้เราตั้งชื่อใหม่และถ้าต้องการดูข้อมูล ก็เลือกไปที่ Load file ก็เลือกไปที่ Load file ก็จะแสดงข้อมูลออกมาในรูปของกราฟ หรือถ้าจะนำข้อมูลแสดงออกทางเครื่องพิมพ์ก็เพียงแต่ทำการเลือกเมนูPrintก็จะได้ผลออกทางเครื่องพิมพ์เป็นตัวเลขดังรูปที่ 6

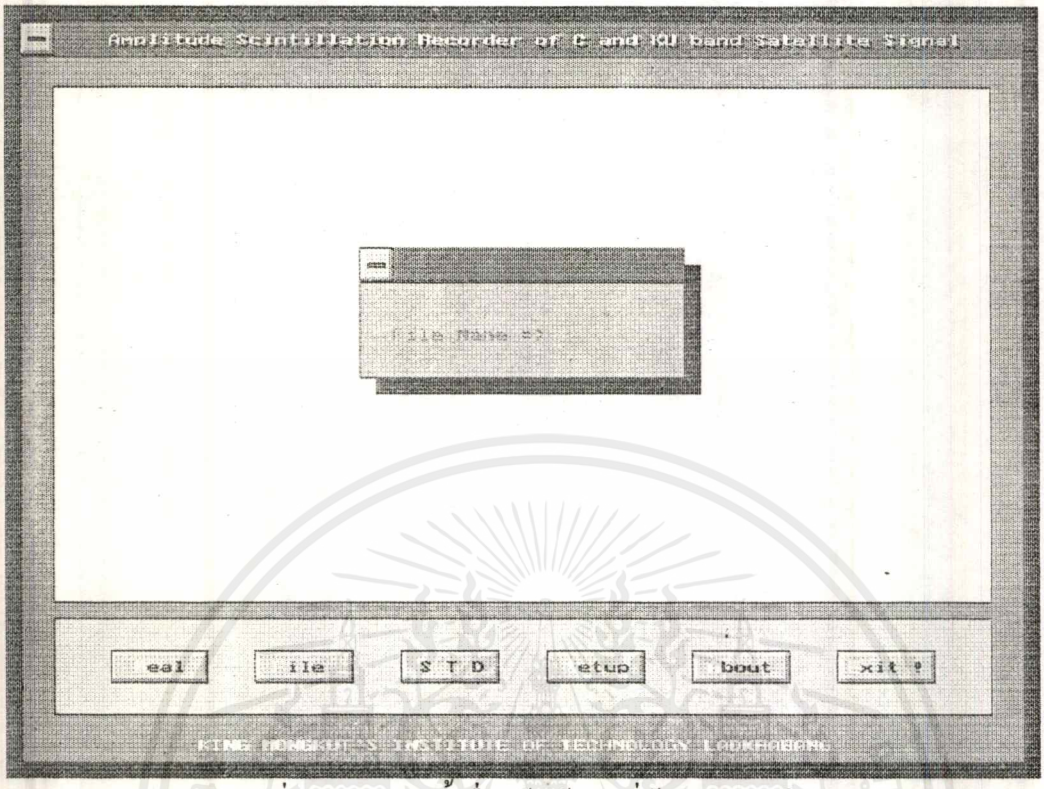
ถ้าต้องการตั้งค่า การแสดงผลข้อมูลทางจอภาพก็สามารถเลือกได้จากเมนู Setup ดังรูปที่ 7 ซึ่งเราสามารถกำหนดให้แสดงผลเป็น 1 จุดต่อวินาที, 5 จุดต่อวินาที, 10 จุดต่อวินาที หรือ 20 จุดต่อวินาทีก็ได้ และจะให้แสดงผลแบบต่อเนื่องไปตลอดหรือจะให้หยุดทีละ 1 หน้าจอก็ได้ หลังจากตั้งค่าเสร็จแล้วก็เลือกไปที่ ON ก็จะเป็นการเก็บค่าต่าง ๆ

สำหรับเมนู About เป็นการแสดงเกี่ยวกับการจัดทำโปรแกรมนี้ดังรูปที่ 8ส่วนเมนู Exit ก็จะให้เลือกว่าจะออกจากโปรแกรมหรือไม่ดังรูปที่ 9

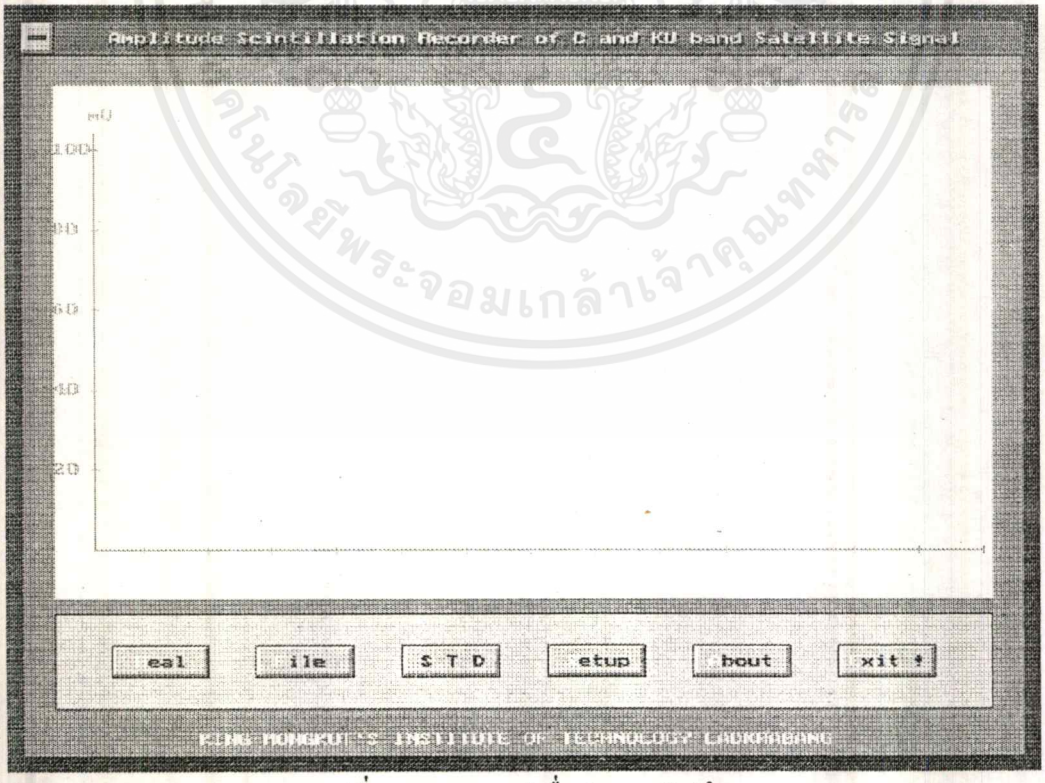
รูปที่ 1 เมื่อเริ่มต้นเข้าสู่โปรแกรม



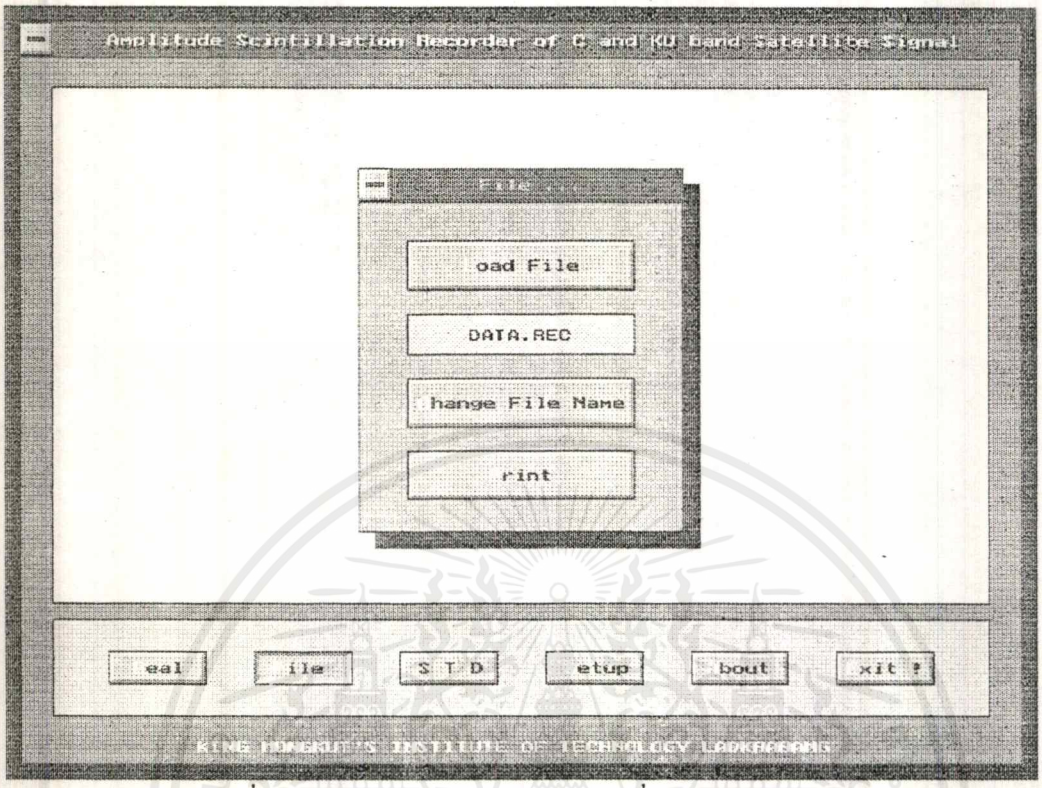
รูปที่ 2 แสดงเมนูเลือกการทำงานว่าจะบันทึกข้อมูลหรือไม่บันทึกข้อมูล



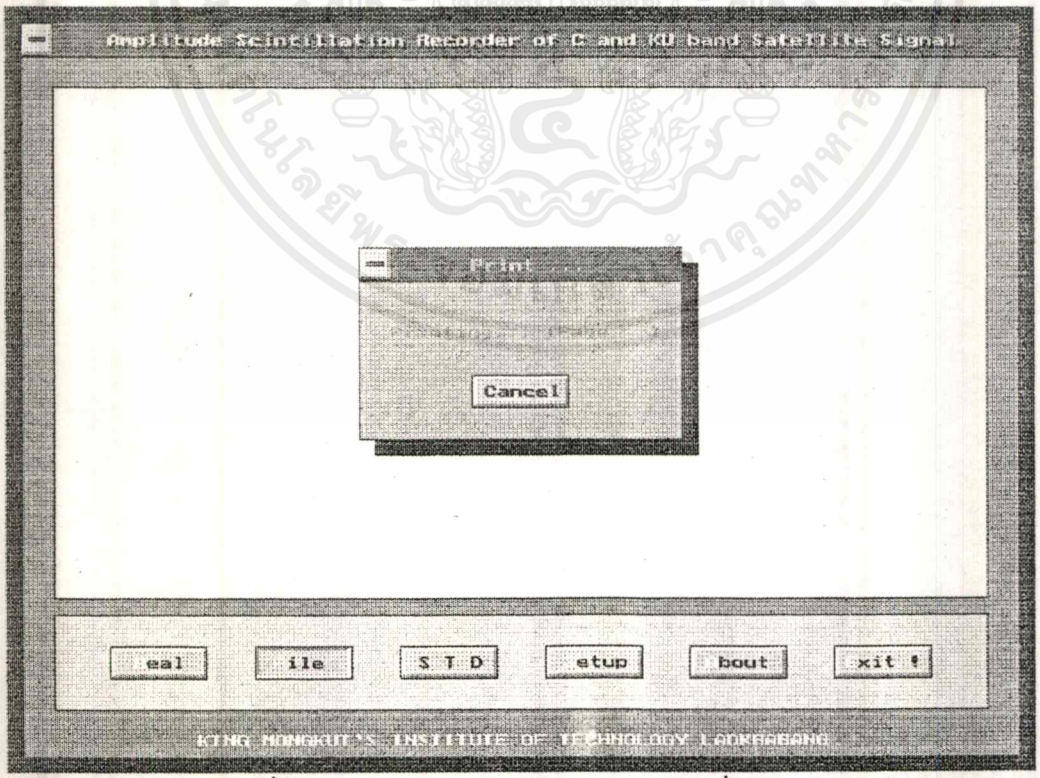
รูปที่ 3 แสดงการตั้งชื่อแฟ้มข้อมูลที่ต้องการบันทึก



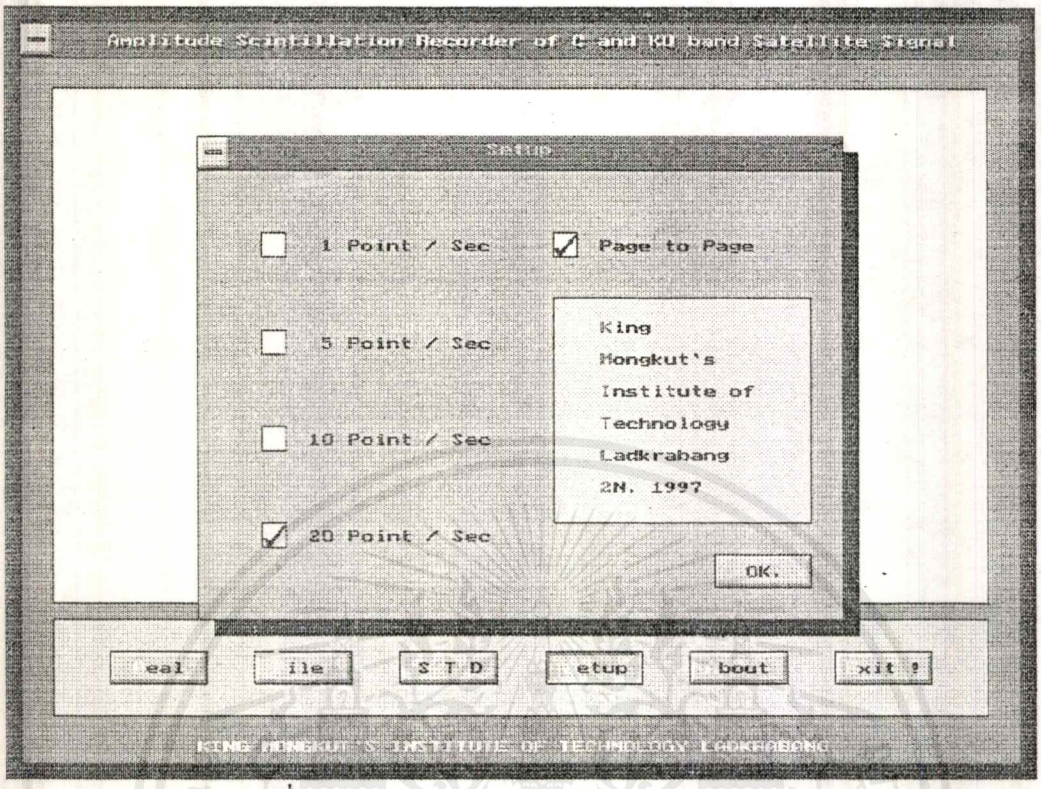
รูปที่ 4 แสดงกราฟเพื่อทำการบันทึก



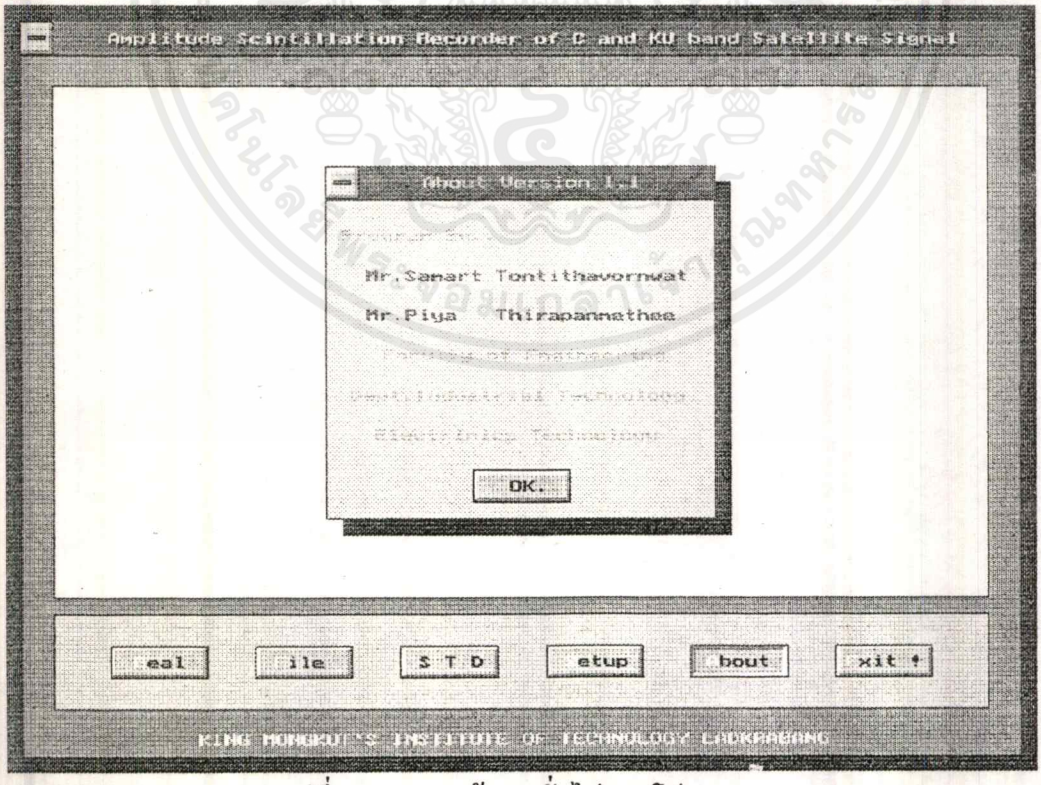
รูปที่ 5 แสดงเมนูการเลือกแฟ้มข้อมูลที่ต้องการแสดงผล



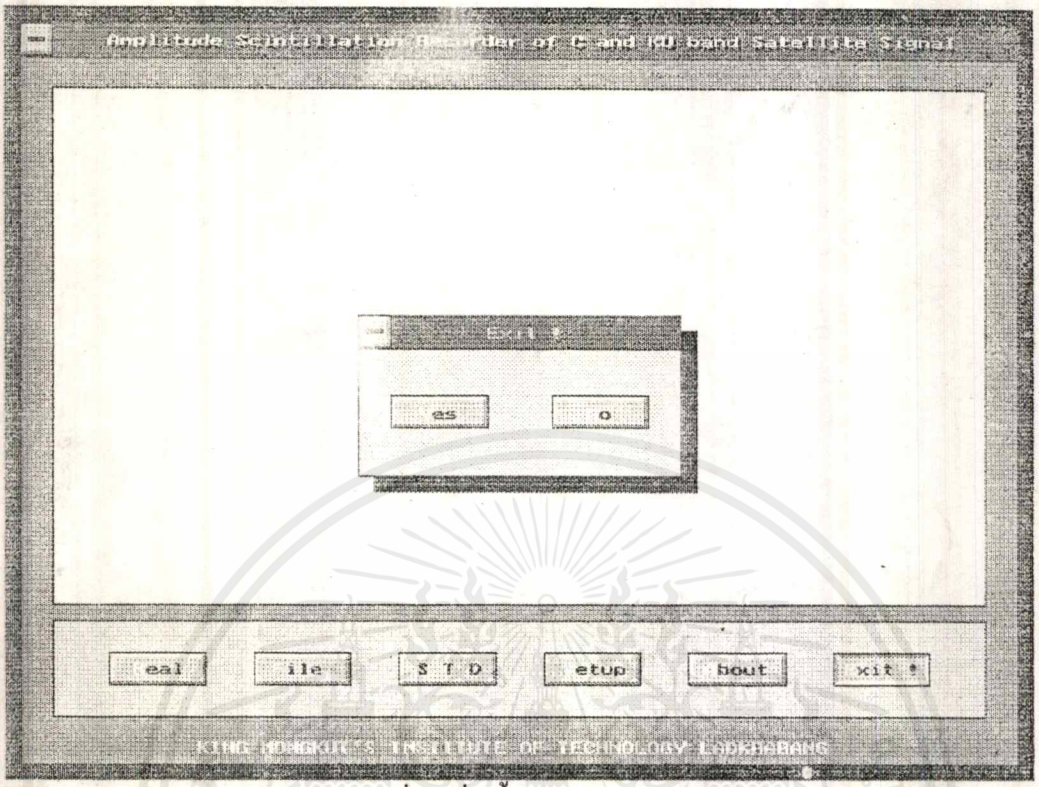
รูปที่ 6 แสดงผลขณะพิมพ์ข้อมูลออกทางเครื่องพิมพ์



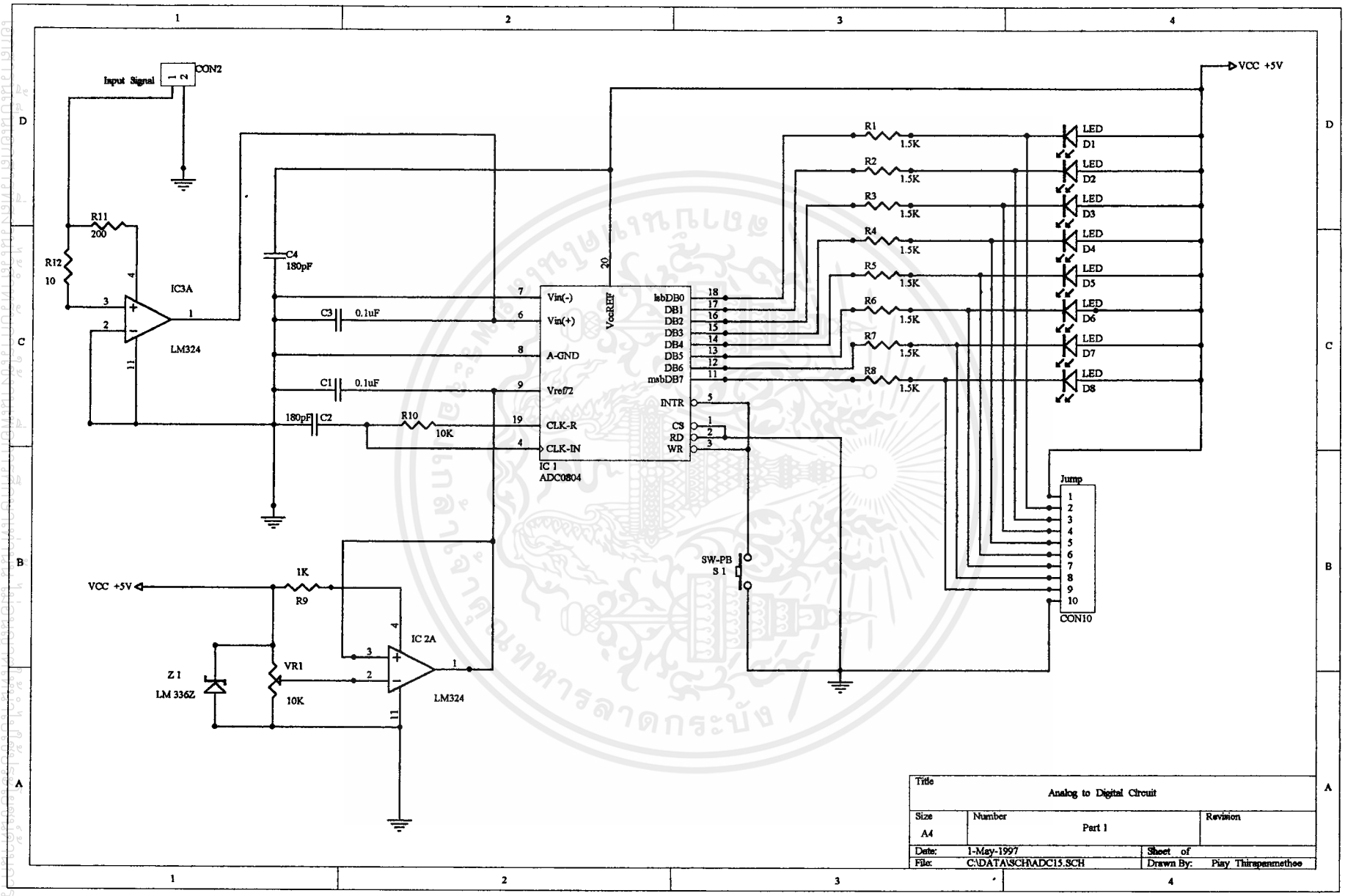
รูปที่ 7 แสดงเมนูการกำหนดรูปแบบการแสดงผล

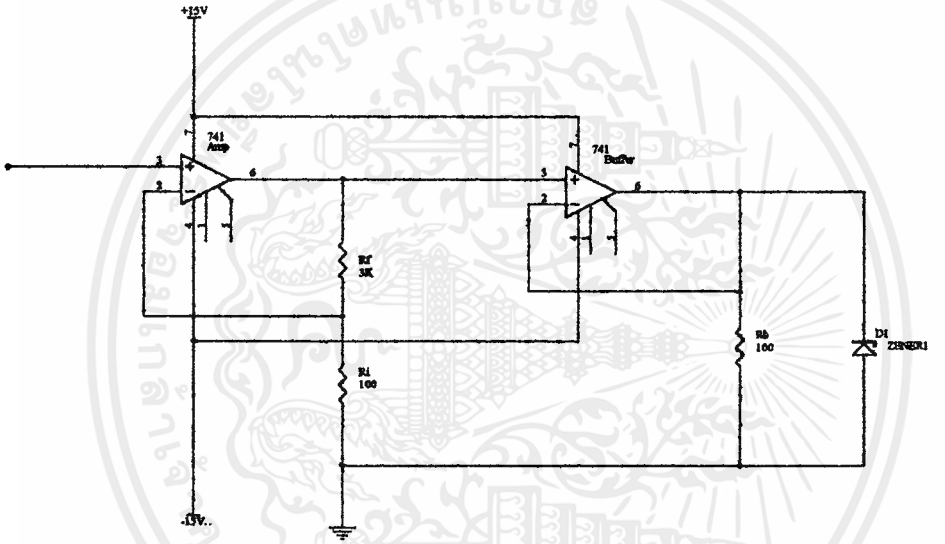


รูปที่ 8 แสดงผลข้อมูลทั่วไปของโปรแกรม

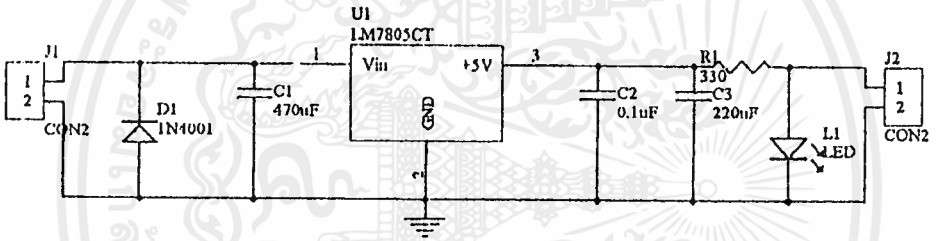


รูปที่ 9 เมื่อสิ้นสุดการทำงาน

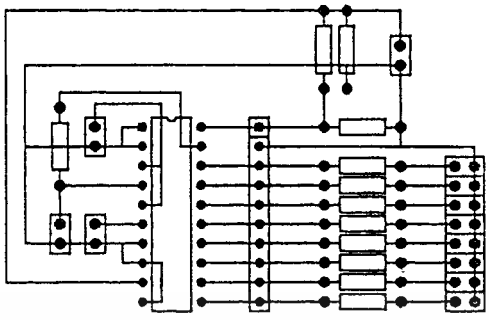

Input Signal
CON2
VCC +5V
R11
200
R12
10
IC3A
LM324
C4
180pF
C3
0.1uF
C1
0.1uF
180pF
C2
R10
10K
IC 1
ADC0804
Vin(-)
Vin(+)
A-GND
Vref/2
CLK-R
CLK-IN
VccREF
lsbDB0
DB1
DB2
DB3
DB4
DB5
DB6
msbDB7
INTR
CS
RD
WR
R1
R2
R3
R4
R5
R6
R7
R8
1.5K
LED
D1
D2
D3
D4
D5
D6
D7
D8
Jump
CON10
SW-PB
S 1
1K
R9
VR1
10K
Z 1
LM 336Z
IC 2A
LM324
Title
Analog to Digital Circuit
Size
A4
Number
Part 1
Revision
Date:
1-May-1997
Sheet of
File:
C:\DATA\SCH\ADC15.SCH
Drawn By:
Piay Thirapanmethee



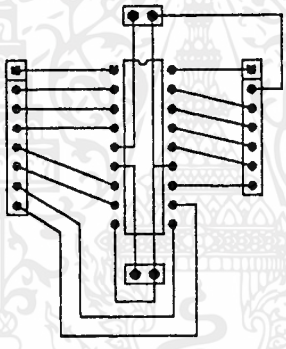
Title	AGC. Amplifier		
Size B	Number		Revision
Date:	1-May-1997	Sheet of	
File:	C:\DATA\WORK\AGC_AMP.SCH	Drawn By:	Pirs Thipparuethai



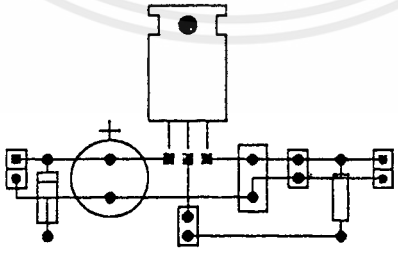
Title		
Size: B	Number	Revision
Date: 1-May-1997		Sheet of
File: C:\DATA\SCH\POWER.SCH		Drawn By:



ลายวงจรการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล



ลายวงจรควบคุมการกดสวิตช์



ลายวงจรแหล่งจ่าย 5 โวลท์

```
; ********** Program for MCS-51 **********
; PROJECT
; AMPLITUDE SCINTILLATION RECOARDER OF C AND KU BAND
; SATELLITE SIGNAL
; PROGRAM BY ELECTRONICS TECHNOLOGY 2N. 1997

                ORG     0000H

CONTLCD         EQU     0FA00H          ;CONTROL PORT LCD
READBUSY        EQU     0FA01H          ;READ BUSY
WRITEDAT        EQU     0FA02H          ;WRITE DATA
CONTPORT        EQU     F803H           ;CONTROL PORT  8255 - 1
PORTA           EQU     0F800H          ;PORT A
PORTB           EQU     0F801H          ;PORT B
PORTC           EQU     0F802H          ;PORT C

; ********** MAIN **********

MAIN:           LCALL   INITLCD
                LCALL   INITPORT
                LCALL   INITSERIAL

MAIN1:          LCALL   TDISP
                LJMP    MAIN1

; ********** BYTE WRITE **********
; DS 1202
; IN  R6 = COMMAND
; OUT R7 = DATA

BWR:            CLR     P1.4            ;COMMAND BYTE WRITE
                LCALL   DELAY
```

```
                SETB    P1.6            ;RST = 1
                LCALL   DELAY

                MOV     B,#08H          ;SEND COMMAND
                CLR     C

BWR1:           MOV     A,R6
                RRC     A
                MOV     R6,A
                MOV     P1.4,C
                LCALL   SCLKRW
                DJNZ    B,BWR1

                MOV     B,#08H          ;SEND DATA BYTE
                CLR     C

BWR2:           MOV     A,R7
                RRC     A
                MOV     R7,A
                MOV     P1.4,C
                LCALL   SCLKRW
                DJNZ    B,BWR2

                CLR     P1.6            ;SET RST = 0
                LCALL   DELAY
                RET

; ****************************** BYTE READ ************************
; DS, 1202
; IN  R6 = COMMAND
; OUT R7 = DATA
```

```
BRD:        SETB    P1.4            ;COMMAND BYTE READ
            LCALL   DELAY
            SETB    P1.6            ;RST = 1
            LCALL   DELAY

            MOV     B,#08H          ;SEND COMMAND
            CLR     C

BRD1:       MOV     A,R6
            RRC     A
            MOV     R6,A
            MOV     P1.4,C
            LCALL   SCLKCOM
            DJNZ    B,BRD1

            MOV     B,#08H          ;REC DATA BYTE
            MOV     R7,#00H         ;CLR DATA BUFFER

BRD2;       LCALL   SCLKRW
            MOV     A,R7
            MOV     C,P1.4          ;READ SERIAL DATA
            RRC     A
            MOV     R7,A
            DJNZ    B,BRD2
            CLR     P1.6            ;RST = 0
            LCALL   DELAY

            RET

; ********** SERIAL CLOCK COMMAND **********

SCLKCOM:    CLR     P1.5
```

```
                LCALL   DELAY
                SETB    P1.5
                LCALL   DELAY

                RET

; *************** SERIAL CLOCK READ / WRITE DATA ********************
; CONTROL DS1202

SCLKRW:         SETB    P1.5
                LCALL   DELAY
                CLR     P1.5
                LCALL   DELAY
                RET

; ***************************** DELAY ********************************

DELAY:          MOV     R1,#05H
                DJNZ    R1,$
                RET

; ***************************** TIME DISPLAY *************************
; DISPLAY DATA FROM DS1202 TO LCD

TDISP:
                MOV     DPTR,#CONTLCD
                MOV     A,#080H
                MOVX    @DPTR,A
                LCALL   WAITBF

                MOV     R6,#85H         ;READ HOUR
                LCALL   BRD
```

```
                MOV     A,R7
                LCALL   OUTLCD
                MOV     A,#03AH
                LCALL   WRITECH
                MOV     R6,#83H         ;READ MIN
                LCALL   BRD
                MOV     A,R7
                LCALL   OUTLCD
                MOV     A,#03AH
                LCALL   WRITECH
                MOV     R6,#081H        ;READ SEC
                LCALL   BRD
                MOV     A,R7
                LCALL   OUTLCD

                MOV     B,A
LOOP:           LCALL   CHECKKEY
                MOV     R6,#081H
                LCALL   BRD
                MOV     A,R7
                CJNE    A,B,NOTEQUAL    ; CHECK 1 SEC
                LJMP    LOOP

NOTEQUAL:       RET

; ************************* CHECK  KEY *******************************
; CHECK FOR KEY USE

CHECKKEY:       PUSH    DPH
                PUSH    DPL
                MOV     DPTR,#PORTA
                MOVX    A,@DPTR
```

```
                CJNE    A,#0F1H,KEYREAL     ; KEY REAL TIME
                MOV     DPTR,#REALTIME
                LCALL   WRITEST

LOCKKEY1:       MOV     DPTR,#PORTA
                MOVX    A,@DPTR
                CJNE    A,#0F0H,LOCKKEY2
                MOV     DPTR,#REAL_ING
                LCALL   WRITEST
                LCALL   READADC
                LCALL   CLEARLCD
                SJMP    KEYSTOP

LOCKKEY2:       CJNE    A,#0F4H,LOCKKEY1
                LCALL   CLEARLCD
                SJMP    KEYSTOP

KEYREAL:        CJNE    A,#0F5H,KEYSET      ; KEY RECORD ON MCS-51
                MOV     DPTR,#RECCORD
                LCALL   WRITEST

LOCKKEY3:       MOV     DPTR,#PORTA
                MOVX    A,@DPTR
                CJNE    A,#0F0H,LOCKKEY4
                MOV     DPTR,#RECCORD_ING
                LCALL   WRITEST
                LCALL   RECCORDRAM
                LCALL   CLEARLCD
                SJMP    KEYSTOP

LOCKKEY4:       CJNE    A,#0F4H,LOCKKEY3
```

```
                LCALL   CLEARLCD
                SJMP    KEYSTOP

KEYSET:         CJNE    A,#0F2H,KEYSTOP    ; KEY SET TIME
                MOV     DPTR,#KEYTIME
                LCALL   WRITEST

LOCKKEY5:       MOV     DPTR,#PORTA
                MOVX    A,@DPTR
                CJNE    A,#0F0H,LOCKKEY6
                MOV     DPTR,#LOAD_ING
                LCALL   WRITEST
                LCALL   RAMTOCOM
                LCALL   CLEARLCD
                SJMP    KEYSTOP

LOCKKEY6:       CJNE    A,#0F4H,LOCKKEY5
                LCALL   CLEARLCD

KEYSTOP:        POP     DPL
                POP     DPH
                RET

; ********************** CONVERT CODE TO LCD ***********************
; CONVERT CODE FOR DISPLAY ON LCD

OUTLCD:         MOV     B,A
                ANL     A,#0F0H
                SWAP    A
                ADD     A,#030H
                LCALL   WRITECH
                MOV     A,B
```

```
                ANL     A,#00FH
                ADD     A,#030H
                LCALL   WRITECH
                RET

; ************************** INTTIAL LCD ****************************
; FOR START UP LCD DISPLAY

INITLCD:        MOV     DPTR,#CONTLCD
                MOV     A,#038H         ; FUNCTION SET
                MOVX    @DPTR,A
                LCALL   WAITBF
                MOV     A,#00CH         ; DISPLAY
                MOVX    @DPTR,A
                LCALL   WAITBF
                MOV     A,#001H         ; CLEAR DISPLAY
                MOVX    @DPTR,A
                LCALL   WAITBF
                MOV     A,#080H         ; SET  DDRAM ADDRESS
                MOVX    @DPTR,A
                LCALL   WAITBF

                MOV     DPTR,#TITLE1
                LCALL   WRITEST
                LCALL   DELAY5SEC
                LCALL   CLEARLCD

                RET

; ********************** DELAY 5 SEC FOR SHOW ************************
; WAIT FOR SHOW SCREE LCD
```

```
DELAY5SEC:      MOV     R7,#0FFH
L1:             MOV     R6,#0FFH
L2:             MOV     R5,#022H

                DJNZ    R5,$
                DJNZ    R6,L2
                DJNZ    R7,L1

                RET

; ********************** WRITE ASCII TO LCD ************************

WRITECH:        PUSH    DPL
                PUSH    DPH
                MOV     DPTR,#WRITEDAT
                MOVX    @DPTR,A
                CALL    WAITBF
                POP     DPH
                POP     DPL

                RET

; ********************** WRITE STRING TO LCD ***********************

WRITEST:        PUSH    DPL
                PUSH    DPH

                PUSH    DPL
                PUSH    DPH

                MOV     DPTR,#CONTLCD
                MOV     A,#080H             ; LINE 1
```

```
        MOVX    @DPTR,A
        CALL    WAITBF

        POP     DPH
        POP     DPL

        MOV     R0,#08H

LST1:   MOV     A,#00H
        MOVC    A,@A+DPTR
        CJNE    A,#00H,LST2
        SJMP    LST4

LST2:   LCALL   WRITECH
        INC     DPTR
        DJNZ    R0,LST1

LST3:   PUSH    DPL
        PUSH    DPH

        MOV     DPTR,#CONTLCD
        MOV     A,#0C0H         ; LINE 2
        MOVX    @DPTR,A
        CALL    WAITBF

        POP     DPH
        POP     DPL

LST5:   MOV     A,#00H
        MOVC    A,@A+DPTR
        CJNE    A,#00H,LST6
        SJMP    LST4
```

```
LST6:           LCALL   WRITECH
                INC     DPTR
                SJMP    LST5

LST4:           POP     DPH
                POP     DPL

                RET

; ************************** WAIT FOR READY ************************

WAITBF:         PUSH    DPH
                PUSH    DPL
                MOV     DPTR,#READBUSY

RDY:            MOVX    A,@DPTR
                JB      ACC.7,RDY

                POP     DPL
                POP     DPH

                RET

; ************************** CLEAR SCREEN LCD  ***********************

CLEARLCD:       MOV     DPTR,#CONTLCD
                MOV     A,#01H          ; CLEAR DISPLAY
                MOVX    @DPTR,A

                RET
```

```
; ***************************** INITIAL PORT ************************

INITPORT:       MOV     DPTR,#CONTPORT
                MOV     A,#09BH
                MOVX    @DPTR,A

                RET



; ******************** SERIAL COMMUNICATION CONFIG ****************
; SERIAL COMMUNICATION (MODE1) 9600 BAUD
; READ REAL TIME FROM PORT1

INITSERIAL:     MOV     PCON,#00H       ;smod = 0
                MOV     SCON,#40H       ;serial mode1
                MOV     TMOD,#20H       ;timer1 mode2
                MOV     IE,#00H         ;set int
                MOV     TH1,#0FDH       ;9600 baud
                SETB    TR1             ;start timer1

                RET

; ********************** READ A/D FROM PORT B ***********************

READADC:        PUSH    DPH
                PUSH    DPL
                PUSH    ACC

                MOV     DPTR,#PORTB
WRITE:          MOVX    A,@DPTR
```

```
; ********************** SEND DATA TO PC **********************
; SEND SINGNAL RECOARD TO PC

                MOV     SBUF,A          ;send DATA

WAIT:           JNB     TI,WAIT         ;wait
                CLR     TI
                LCALL   DELAYAD

; ************************* CHECK KEY STOP ************************

                PUSH    DPH
                PUSH    DPL

                MOV     DPTR,#PORTA
                MOVX    A,@DPTR
                CJNE    A,#0F4H,READY1
                MOV     DPTR,#STOPREAL
                LCALL   WRITEST
                LCALL   DELAY5SEC

                POP     DPL
                POP     DPH

                SJMP    READY
READY1:         POP     DPL
                POP     DPH

; ***************************************************************

                SJMP    WRITE
READY:          POP     ACC
```

```
                POP     DPL
                POP     DPH
                RET

DELAYAD:        MOV     R0,#0FDH        ; 1 CY
DELAY1:         MOV     R1,#059H        ; 1 CY
DELAY2:         DJNZ    R1,DELAY2       ; 2 CY
                DJNZ    R0,DELAY1       ; 2 CY
                RET                     ; 2 CY

; ********************** REC TO DATA MEMORY ************************

RECCORDRAM:     PUSH    DPH
                PUSH    DPL
                MOV     DPTR,#0005H     ; START ADDRESS RAM
                PUSH    DPH
                PUSH    DPL

LL1:            MOV     DPTR,#PORTB
                MOVX    A,@DPTR
                POP     DPL
                POP     DPH
                MOVX    @DPTR,A
                INC     DPTR
                PUSH    DPH
                PUSH    DPL
                LCALL   DELAYAD
                MOV     R7,DPH
                MOV     R6,DPL

; ************************** CHECK KEY ************************
```

```
STOPER:     LCALL   DELAY5SEC
            POP     DPL
            POP     DPH

            RET

; ********************** SEND DATA TO COM ************************
; FORM MEMORY ON MCS-51

RAMTOCOM:   PUSH    DPH
            PUSH    DPL
            MOV     DPTR,#0005H

L_MEM:      MOVX    A,@DPTR
            INC     DPTR
            CJNE    A,#0FFH,S_TO_C
            SJMP    E_MEM

;************************ SEND DATA TO PC **********************

S_TO_C:     MOV     SBUF,A          ;send DATA

WAITRS:     JNB     TI,WAITRS       ;wait
            CLR     TI
            SJMP    L_MEM

E_MEM:      MOV     DPTR,#MEM_END
            LCALL   WRITEST
            LCALL   DELAY5SEC
            POP     DPL
            POP     DPH
```

```
                RET

; ******************************** data ****************************

TITLE1:         DB      " KMIT'L 1997 ",00H
REALTIME:       DB      "SEND REALTIME ?",00H
REAL_ING:       DB      " SENDING ...  ",00H
STOPREAL:       DB      " SEND STOP ... ",00H
RECCORD:        DB      " RECORD DATA ? ",00H
RECCORD_ING:    DB      " RECORDING ... ",00H
STOPRAM:        DB      " RECORD STOP... ",00H
KEYSTART:       DB      "   START    ",00H
KEYSTOP1:       DB      "   STOP     ",00H
KEYUP:          DB      " ELECTRONICS ",00H
KEYDOWN:        DB      " TECHNOLOGY  ",00H
KEYTIME:        DB      " LOAD MEMMORY ? ",00H
LOAD_ING:       DB      " LOADING ...  ",00H
MEM_END:        DB      " MEMORY EMPTY . ",00H
RAMFULL:        DB      " MEMMORY FULL ! ",00H

                END
```

```
Program Project_KMIT_L;
Uses    Crt,Graph,Dos;
Var     Grdriver,Grmode  : Integer;
        Regs   : Registers;
        st     : String;
        Fi     : Text;
        Return : Integer;

        { ************************************************************** }
        { ************************** Unit ************************** }
        { ************************************************************** }

Procedure GReadStr(x,y : Integer;Var str : String ;max,color : Byte);
Var     ch             : Char;
        ln             : Byte;
        OldColor       : Integer;
        OldFillStyle   : FillSettingsType;
Begin
  Str:='';
  OldColor:=GetColor;
  GetFillSettings(OldFillStyle);
  setfillstyle(1,0);
  SetColor(Color);
  Repeat
    ln:=Length(str);
    SetFillStyle(OldFillStyle.pattern,OldFillStyle.color);

    If (length(str) <= max-1) Then
     Begin
      ch:=Readkey;
      If ch in [#32..#126] Then Str:=Str+ch;
```

```
    End
   Else
    Begin
      ch:=readkey; If ch <> #13 Then str[max]:=ch;
    End;
   If ch=#8 Then Str:=Copy(Str,1,length(str)-1);
   Bar(x,y,x+(max*8)+2,y+7);
   OutTextXy(x,y,str);
  Until ch = #13;
  SetColor(OldColor);
  SetFillStyle(OldFillStyle.pattern,OldFillStyle.color);
  Bar(x+(ln*8),y,x+(ln*8)+2,y+7);
End;

Procedure Box(x1,y1,x2,y2,BkCol : Integer);
Begin
    SetFillStyle(SolidFill,BkCol);
    Bar(x1,y1,x2,y2);
    Setcolor(0);
    Rectangle(x1,y1,x2,y2);
End;

Procedure Click_Box(x,y,Push : Integer);
Begin
    Box(x,y,x+15,y+15,15);
    If Push = 0 Then
      Begin
        Box(x,y,x+15,y+15,15);
      End
      Else
        Begin
```

```
            Setcolor(0);
            SetLineStyle(0,0,3);
            Line(x+2,y+10,x+4,y+14);
            Line(x+4,y+14,x+14,y+2);
            SetLineStyle(0,0,1);
        End;
End;

Procedure Check_box(x,y : Integer; Var z : Integer);
Begin
    z := GetPixel(x+2,y+10)
End;

Procedure Box3d(x1,y1,x2,y2,Col,On : integer);
Begin
    SetFillstyle(SolidFill,Col);
    Bar(x1,y1,x2,y2);
    If (On=1) Then Setcolor(15)
      Else Setcolor(8);
          Rectangle(x1+1,y1+1,x2-1,y2-1);
          Rectangle(x1+2,y1+2,x2-2,y2-2);
    If (On=1) Then Setcolor(8)
      Else Setcolor(15);
          Line(x2-1,y1+1,x2-1,y2-1);
          Line(x2-2,y1+2,x2-2,y2-2);
          Line(x1+2,y2-2,x2-2,y2-2);
          Line(x1+1,y2-1,x2-1,y2-1);
    Setcolor(0);
    Rectangle(x1,y1,x2,y2);
End;
```

```
Procedure TabBar(x1,y1,x2,y2,C : Integer);
Begin
    SetFillStyle(SolidFill,0);          { SharDow  }
    Bar(x1+10,y1+10,x2+10,y2+10);
    Box3d(x1,y1,x1+20,y1+20,7,1);
    SetFillStyle(SolidFill,1);
    Bar(x1+21,y1,x2,y1+20);
    Box(x1,y1+21,x2,y2,C);
    Setcolor(0);
    Rectangle(x1+5,y1+9,x1+15,y1+11);
    Setcolor(8);
    Line(x1+6,y1+10,x1+14,y1+10);
End;

Procedure CheckKey(Var Xmouse,Ymouse,Key : Integer);
Var  M1  : Integer;
Begin
    Key := 0;
    M1  := 0;
    Repeat
    Regs.Ax := 3;
    Intr($33,Regs);
    M1 := Regs.Bx;
    Xmouse := Regs.Cx;
    Ymouse := Regs.Dx;
    If Keypressed Then
    Begin
        Regs.Ax := 1;
        Intr($16,Regs);
        Key := Regs.Al;
    End;
```

```
    Until (Key <> 0) or (M1 = 1);
End;

Procedure SavePic(x1,y1,x2,y2 : Integer; Var r : Pointer);
Var Sz : Word;
Begin
    Sz := ImageSize(x1,y1,x2,y2);
    GetMem(r,sz);
    GetImage(x1,y1,x2,y2,r^);
End;

Procedure ReturnPic(x1,y1,x2,y2: Integer ; Var p : Pointer);
Var Sz : Word;
Begin
    PutImage(x1,y1,p^,0);
    Sz := ImageSize(x1,y1,x2,y2);
    FreeMem(p,ImageSize(x1,y1,x2,y2));
End;

Procedure SetArea (mX,mY,Xm,Ym : Integer);
Begin
     Regs.Ax := 7;
     Regs.Cx := mX;
     Regs.Dx := Xm;
     Intr($33,Regs);
     Regs.Ax := 8;
     Regs.Cx := mY;
     Regs.Dx := Ym;
     Intr($33,Regs);
End;
```

```
Procedure  ResetArea;
Begin
    Regs.Ax := 7;
    Regs.Cx := 0;
    Regs.Dx := GetMaxX;
    Intr($33,Regs);
    Regs.Ax := 8;
    Regs.Cx := 0;
    Regs.Dx := GetMaxY;
    Intr($33,Regs);
End;

Procedure CloseMouse;
Begin
    Regs.Ax := 2;
    Intr($33,Regs);              { Hidden Mouse }
End;

Procedure OpenMouse;
Begin
    Regs.Ax := 1;
    Intr($33,Regs);              { Show Mouse }
End;

Procedure Timer(Var Timing : String);
Var  Hour,Min,Sec,mSec : Word;
    st,st1          : String;
Begin
    GetTime(Hour,Min,Sec,mSec);
    Str(Hour,st);
    If Length(st) = 1 Then st := '0' + st;
```

```
    st := st + ':';
    Str(Min,st1);
    If Length(st1) = 1 Then st1 := '0' + st1;
    st1 := st1 + ':';
    st := st + st1;
    Str(Sec,st1);
    If Length(st1) = 1 Then st1 := '0' + st1;
    st := st + st1;
    Timing := st;
End;

Procedure CheckError(IoErr : Integer);
Var StErr   : String;
   Pic     : Pointer;
   x,y,key : Integer;
Begin
    Case IoErr of
        1 : StErr := 'Invalid function number';
        2 : StErr := 'File not found';
        3 : StErr := 'Path not found';
        4 : StErr := 'Too many open files';
        5 : StErr := 'File access denied';
        6 : StErr := 'Invalid file handle';
       12 : StErr := 'Invalid file access code';
       15 : StErr := 'Invalid drive number';
       16 : StErr := 'Cannot remove current directory';
       17 : StErr := 'Cannot rename across drives';
       18 : StErr := 'No more files';
      100 : StErr := 'Disk read error';
      101 : StErr := 'Disk write error';
      102 : StErr := 'File not assigned';
```

```
103 : StErr := 'File not open';
104 : StErr := 'File not open For input';
105 : StErr := 'File not open For output';
106 : StErr := 'Invalid numeric Format';
150 : StErr := 'Disk is write-protected';
151 : StErr := 'Bad drive request struct length';
152 : StErr := 'Drive not ready';
154 : StErr := 'CRC error in data';
156 : StErr := 'Disk seek error';
157 : StErr := 'Unknown media type';
158 : StErr := 'Sector Not Found';
159 : StErr := 'Printer out of paper';
160 : StErr := 'Device write fault';
161 : StErr := 'Device read fault';
162 : StErr := 'Hardware failure';
200 : StErr := 'Division by zero';
201 : StErr := 'Range check error';
202 : StErr := 'Stack overflow error';
203 : StErr := 'Heap overflow error';
204 : StErr := 'Invalid pointer operation';
205 : StErr := 'Floating point overflow';
206 : StErr := 'Floating point underflow';
207 : StErr := 'Invalid floating point operation';
208 : StErr := 'Overlay manager not installed';
209 : StErr := 'Overlay file read error';
210 : StErr := 'Object not initialized';
211 : StErr := 'Call to abstract method';
212 : StErr := 'Stream registration error';
213 : StErr := 'Collection index out of range';
214 : StErr := 'Collection overflow error';
215 : StErr := 'Arithmetic overflow error';
```

```
        216 : StErr := 'General Protection fault';
    End;
    SavePic(200,185,445,305,Pic);
    TabBar(210,190,430,290,11);
    SetColor(12);
    OuttextXY(300,198,'Error !');
    SetColor(0);
    y := (Length(StErr) Div 2) * 8;
    x := 320-y;
    OutTextXy(x,250,StErr);
    Writeln(#07);
    CheckKey(x,y,key);
    ReturnPic(200,185,445,305,Pic);
End;

Procedure ReadFile_INI(head : string;Var val : string);
Begin
    Assign(fi,'KMITL.INI');
    {$I-}
        Reset(Fi);
    {$I+}
    Return := IOResult;
    If Return <> 0 Then
      Begin
          CheckError(Return);
      End;
    Repeat
        Readln(fi,st);
        Val := copy(st,length(head)+4,length(st));
        St := copy(st,1,length(head));
    Until (st = head) OR (EOF(FI));
```

```
    Close(fi);
End;

Procedure WriteOption(Data_1,Data_2,Data_3 : string);
Begin
    Assign(fi,'KMITL.INI');
    {$I-}
        Rewrite(Fi);
    {$I+}
    Return := IOResult;
    If Return <> 0 Then
      Begin
          CheckError(Return);
          CloseGraph;
          Writeln('Good Bye ...');
          Halt;
      End
      Else
         Begin
             Writeln(fi,'[Amplitude Scintillation Recorder of C and KU band Satellite
Signal]');
             Writeln(fi,'');
             Writeln(fi,'POINT = '+data_1);
             Writeln(fi,'PAGE TO PAGE = '+data_2);
             Writeln(fi,'FILE = '+data_3);
             Writeln(fi,'');
             Write(fi,'[End Of KMITL.INI]');
             Close(fi);
         End;
End;
```

```
Procedure ClearScreen;
Begin
    SetFillStyle(1,15);
    Bar(30,50,610,370);
End;

Procedure OutTextExtra(x,y,color : Integer;Text : String);
Begin
    Setcolor(0);
    OutTextXY(x,y,Text);
    SetColor(color);
    OutTextXY(x,y,copy(Text,1,1));
    SetColor(0);
End;

{ ************************************************************ }
{ ************************* End Unit ************************** }
{ ************************************************************ }

{ ************************************************************ }
{ ********************** Start Program ********************** }
{ ************************************************************ }

Procedure Screen;
Begin
    TabBar(10,10,630,470,9);
    Setcolor(0);
    OuttextXY(67,19,'Amplitude Scintillation Recorder of C and KU band Satellite Signal');
    Setcolor(15);
    OuttextXY(65,17,'Amplitude Scintillation Recorder of C and KU band Satellite Signal');
```

```
    OuttextXY(120,456,'KING MONGKUT`S INSTITUTE OF TECHNOLOGY
LADKRABANG');
    Box(30,380,610,440,10);
    Box(30,50,610,370,15);
    Box3d(65,400,125,420,7,1);
    Box3d(155,400,215,420,7,1);
    Box3d(245,400,305,420,7,1);
    Box3d(335,400,395,420,7,1);
    Box3d(425,400,485,420,7,1);
    Box3d(515,400,575,420,7,1);
    OutTextExtra(80,407,15,'Real');
    OutTextExtra(167,407,15,'File');
    OutTextXY(257,407,'S T D');
    OutTextExtra(347,407,15,'Setup');
    OutTextExtra(435,407,15,'About');
    OutTextExtra(522,407,15,'Exit !');
End;

  { ************************************************************ }
  { ************** Read FileName To File INI ***************** }
  { ************************************************************ }

Procedure Input_Name;
Var  Pic1          : Pointer;
     FileName      : String;
     INI1,INI2     : String;
     I             : Integer;
Begin
     SavePic(220,150,430,240,Pic1);
     TabBar(220,150,420,230,3);
     OutTextXY(240,200,'File Name =>');
```

```
    GreadStr(340,200,FileName,8,1);
    ReadFile_INI('POINT',INI1);
    ReadFile_INI('PAGE TO PAGE',INI2);
    For I := 1 to Length(FileName) Do
      Begin
        FileName[I] := UpCase(FileName[I]);
      End;
    WriteOption(INI1,INI2,FileName);
    ReturnPic(220,150,430,240,Pic1);
End;

  { ************************************************************ }
  { ****************** Scale Range For Load ******************** }
  { ************************************************************ }

Procedure Scale;
Var  x,y,z     : Integer;
Begin
    SetFillStyle(1,15);
    Bar(30,50,610,370);
    Setcolor(3);
    Line(55,80,55,340);
    Line(55,340,605,340);
    For y := 1 to 14 Do
      Begin
        x := 45+(y*40);
        Line(x,338,x,342);
      End;
   For x := 1 to 5 Do
      Begin
        y := 340-(x*50);
```

```
            Line(53,y,57,y);
            z := x*20;
            str(z,st);
            OutTextXY(30,y-4,st);
        End;
    OutTextXY(52,65,'mV');
    Setcolor(0);
End;

  { ************************************************************ }
  { ****************** Scale Range For S T D ******************* }
  { ************************************************************ }

Procedure ScaleSTD;
Var  x,y,z     : Integer;
Begin
    SetFillStyle(1,15);
    Bar(30,50,610,370);
    Setcolor(3);
    Line(55,80,55,340);
    Line(55,340,580,340);
    For y := 1 to 12 Do
       Begin
           x := 55+(y*40);
           Line(x,338,x,342);
           Str(y/10:2:1,st);
           OutTextXy(x-10,347,st);
       End;
    For x := 1 to 5 Do
       Begin
           y := 340-(x*50);
```

```
            Line(53,y,57,y);
            Str(x,st);
            OutTextXy(42,y-4,st);
        End;
    OutTextXY(35,65,'Vp-p(dB)');
    OutTextXY(300,360,'S.D.(dB)');
    Setcolor(0);
End;

  { ************************************************************ }
  { ************* Converse Data From RS-232 to File ************ }
  { ************************************************************ }

Procedure Detect(VData : Integer;Var RData : Integer);
Var  xCal          : Real;
Begin
    xCal := (Vdata * 5) / (255 * 40);  { 255 * 40 }
    Rdata := Round(xCal * 1000);       { Up mVolt To Volt }
End;

  { ************************************************************ }
  { ***************** Record Data From RS-232 ****************** }
  { ************************************************************ }

Procedure SaveRealTime;
Var  xOld,yOld,xNew,yNew   : Integer;
    DataTof                : Integer;
    FileName,TimeData      : String;
    x,y,k                  : Integer;
Begin
    Input_Name;
```

```
ReadFile_INI('FILE',FileName);
Timer(TimeData);
Assign(Fi,FileName+'.REC');
{$I-}
Rewrite(Fi);
{$I+}
Return := IOResult;
If Return <> 0 Then
  Begin
     CheckError(Return);
     Exit;
  End;
Scale;
xNew := 55;   xOld := 55;   yOld := 340;
Writeln(Fi,TimeData);
Regs.Ah := 0;                    { Initial Com Port }
Regs.Al := 227;
Regs.Dx := 1;                    { 0 = Com1   1 = Com2 }
Intr($14,Regs);
Repeat
    Regs.Ah := 2;
    Regs.Dx := 1;                { 0 = Com1   1 = Com2 }
    Intr($14,Regs);
    yNew := Regs.Al;
    Detect(yNew,DataTof);
    Writeln(Fi,DataTof);
    yNew := 340 - yNew;          { No up signal *2 }
    Line(xOld,yOld,xNew,yNew);
    xOld := xNew;     yOld := yNew;
    xNew := xNew + 2;
    If xNew > 600 Then
```

```
        Begin
            Scale;
            xNew := 55;    xOld := 55;
        End;
    Until Keypressed;
    Timer(TimeData);
    Writeln(Fi,TimeData);
    Close(Fi);
    CheckKey(x,y,k);
    ClearScreen;
End;

  { ********************************************************** }
  { ************* Show Data From RS-232 No Record ************** }
  { ********************************************************** }

Procedure RealTime;
Var  xOld,yOld,xNew,yNew   : Integer;
     x,y,k                 : Integer;
Begin
    Scale;
    Regs.Ah := 0;
    Regs.Al := 227;
    Regs.Dx := 1;                    { 0 = Com1   1 = Com2 }
    Intr($14,Regs);
    xNew := 55;   xOld := 55;   yOld := 340;
    Repeat
        Regs.Ah := 2;
        Regs.Dx := 1;                { 0 = Com1   1 = Com2 }
        Intr($14,Regs);
        yNew := Regs.Al;
```

```
        yNew := 340 - yNew;          { no up signal * 2 }
        Line(xOld,yOld,xNew,yNew);
        xOld := xNew;     yOld := yNew;
        xNew := xNew + 2;
        If xNew > 600 Then
          Begin
            Scale;
            xNew := 55;    xOld := 55;
          End;
    Until Keypressed;
    CheckKey(x,y,k);
    ClearScreen;
End;

  { ************************************************************ }
  { *************** Menu Record Data From RS-232 *************** }
  { ************************************************************ }

Procedure Click_Read;
Var Pic2              : Pointer;
   PointX,PointY,KeyD : Integer;
Begin
    CloseMouse;
    SavePic(220,100,430,290,Pic2);
    Box3d(65,400,125,420,7,0);
    OutTextExtra(80,407,15,'Real');
    TabBar(220,100,420,280,3);
    Setcolor(12);
    Outtextxy(280,107,'Real Time ...');
    Box3d(270,160,370,190,7,1);
    Box3d(270,220,370,250,7,1);
```

```
Setcolor(0);
OuttextExtra(290,172,15,'Record..');
OuttextExtra(285,232,15,'No Record');
SetArea(220,120,420,280);
OpenMouse;
Repeat
CheckKey(PointX,PointY,KeyD);
If KeyD = 0 Then
  Begin
    If (PointX > 270) And (PointX < 370) And (PointY > 160) And (PointY < 190)
Then
      Begin
        CloseMouse;
        Box3d(270,160,370,190,7,0);
        OuttextExtra(290,172,15,'Record..');
        Delay(100);
        ReturnPic(220,100,430,290,Pic2);
        ResetArea;
        Box3d(65,400,125,420,7,1);
        OutTextExtra(80,407,15,'Real');
        SaveRealTime;
        KeyD := 28;
      End;
    If (PointX > 270) And (PointX < 370) And (PointY > 220) And (PointY < 250)
Then
      Begin
        CloseMouse;
        Box3d(270,220,370,250,7,0);
        OuttextExtra(285,232,15,'No Record');
        Delay(100);
        ReturnPic(220,100,430,290,Pic2);
```

```
            ResetArea;
            Box3d(65,400,125,420,7,1);
            OutTextExtra(80,407,15,'Real');
            RealTime;
            KeyD := 28;
        End;
End;
If (KeyD = 114) or (KeyD = 82) Then     { R or r }
  Begin
      CloseMouse;
      Box3d(270,160,370,190,7,0);
      OuttextExtra(290,172,15,'Record..');
      Delay(100);
      ReturnPic(220,100,430,290,Pic2);
      ResetArea;
      Box3d(65,400,125,420,7,1);
      OutTextExtra(80,407,15,'Real');
      SaveRealTime;
      KeyD := 28;
  End;
If (KeyD = 110) or (KeyD = 78) Then     { N or n }
  Begin
      CloseMouse;
      Box3d(270,220,370,250,7,0);
      OuttextExtra(285,232,15,'No Record');
      Delay(100);
      ReturnPic(220,100,430,290,Pic2);
      ResetArea;
      Box3d(65,400,125,420,7,1);
      OutTextExtra(80,407,15,'Real');
      RealTime;
```

```
            KeyD := 28;
        End;
      If KeyD = 27 Then
        Begin
            CloseMouse;
            ReturnPic(220,100,430,290,Pic2);
            ResetArea;
            Box3d(65,400,125,420,7,1);
            OutTextExtra(80,407,15,'Real');
            KeyD := 28;
        End;
    Until KeyD = 28;
    OpenMouse;
End;

  { ************************************************************ }
  { ****************** BarStatus Show Time ******************** }
  { ************************************************************ }

Procedure BarStatus(Var ShStr : String);
Var lh : Integer;
Begin
    lh := (Length(ShStr) Div 2) * 8;
    Box(540,445,620,465,15);
    OutTextXY(580-lh,452,ShStr);
End;

  { ************************************************************ }
  { ************* Converse Data From File to Screen ************ }
  { ************************************************************ }
```

```
Procedure  UnDetect(VData : Integer;Var RData : Integer);
Var   xCal              : Real;
Begin
    xCal := VData / 1000;
    Rdata := Round((xcal * 255 * 40) / 5);          { Up mVolt To Volt }
End;

  { ************************************************************ }
  { ************** Load File Data to Show Scree *************** }
  { ************************************************************ }

Procedure LoadRealTime;
Var xO,yO,x,y   : Integer;
   Int,Code     : Integer;
   bx,by,bk     : Integer;
   Name,Points,Pages : String;
   Pic          : Pointer;
   I,Coder,J,Pd     : Integer;

Begin
    ReadFile_INI('POINT',Points);
    Val(Points,I,Coder);
    Case I Of
        1 : J := 20;
        5 : J := 4;
       10 : J := 2;
       20 : J := 1;
        End;
    ReadFile_INI('PAGE TO PAGE',Pages);
    Pd := 0;
    If Pages = 'ON' Then
```

```
  Begin
     Pd := 1;
  End;
ReadFile_INI('FILE',Name);
Assign(Fi,Name+'.REC');
{$I-}
Reset(Fi);
{$I+}
Return := IOResult;
If Return <> 0 Then
  Begin
     CheckError(Return);
  End
  Else
    Begin
       Scale;
       ReadLn(Fi,st);
       BarStatus(st);
       x := 55;
       xO := 55;  yO := 340;
       Readln(Fi,st);
       While NOT EOF(Fi) Do
          Begin
            For I := 1 to J DO
              Begin
                 Readln(Fi,st);
              End;
            If Length(st) < 4 Then
              Begin
                 Val(st,Int,Code);
                 y := Int;
```

```
UnDetect(y,y);
y := 340 - y;
Line(xO,yO,x,y);
xO := x;     yO := y;
x := x + 2;
If x > 600 Then
  Begin
     SetArea(55,80,605,340);
     OpenMouse;
     If Pd = 1 Then
       Begin
          CheckKey(bx,by,bk);
       End;
     ResetArea;
     If bk = 27 Then
       Begin
          CloseMouse;
          ClearScreen;
          Exit;
       End;
     CloseMouse;
     Scale;
     x := 55;    xO := 55;
  End;
Delay(1);
End;

  End;
Close(Fi);
SetArea(55,80,605,340);
OpenMouse;
```

```
            CheckKey(bx,by,bk);
            ResetArea;
            If bk = 27 Then
              Begin
                CloseMouse;
                ClearScreen;
                Exit;
              End;
            CloseMouse;
          End;
      ClearScreen;
End;

  { ************************************************************ }
  { ***************** Option Out Data to Printer ************** }
  { ************************************************************ }

Procedure CopyToPrn(content : String);
Var b : Byte;
Begin
   For b := 1 to Length(content) Do
      Begin
         Regs.Ah := 0;
         Regs.Al := ord(content[b]);
         Regs.Dx := Pred(1);
         Intr($17,Regs);
      End;
End;

Procedure PageStatus(Var ChStr : String);
Var lh : Integer;
```

```
Begin
    lh := (Length(ChStr) Div 2) * 8;
    SetFillStyle(1,3);
    Bar(376,190,400,210);
    OutTextXY(386-lh,200,ChStr);
End;

Procedure Init_Prn;
Begin
   CopyToPrn(#27+'E');
   CopyToPrn(#27+'&l1X');
   CopyToPrn(#27+'&l0S');
   CopyToPrn(#27+'&l0O');
End;

Procedure Now_Print;
Var  Pic1              : Pointer;
   i,ColX,ColY,Line,Page : Integer;
   PageSt,FName        : String;
   Mk1,Mk2,Mk3         : Integer;
Begin
   CloseMouse;
   SavePic(220,150,430,280,Pic1);
   TabBar(220,150,420,270,3);
   SetColor(12);
   OutTextXY(290,158,'Print ...');
   Setcolor(9);
   OutTextXY(240,200,'Printing... [Page   ]');
   SetColor(0);
   Box3d(290,230,350,250,7,1);
   OutTextXY(298,237,'Cancel');
```

```
SetArea(220,170,420,270);
OpenMouse;

Init_Prn;
Line := 1;
Page := 1;
ReadFile_INI('FILE',FName);
CopyToPrn('   [File Name '+Fname+'.REC]');
CopyToPrn(#13#10);
Assign(Fi,FName+'.REC');
Reset(Fi);
Readln(Fi,st);
CopyToPrn('   [Time '+st+']');
CopyToPrn(#13#10);
CopyToPrn(#13#10);
While NOT EOF(Fi) Do
    Begin
        For i := 1 to 10 Do
            Begin
                Readln(Fi,st);
                ColX := Length(st);
                For ColY := 1 to 7-ColX Do
                    Begin
                        CopyToPrn(' ');
                    End;
                If Length(st) > 4 Then
                  Begin
                      CopyToPrn(#13#10#13#10);
                      CopyToPrn('   [Time '+st+']');
                  End
                  Else
```

```
                CopyToPrn(st);
            End;
        CopyToPrn(#13#10);
        Inc(Line);

        Mk1 := 0;
        Regs.Ax := 3;
        Intr($33,Regs);
        Mk1 := Regs.Bx;
        If Mk1 = 1 Then
          Begin
              Mk2 := Regs.Cx;
              Mk3 := Regs.Dx;
              If (Mk2 > 290) And (MK2 < 350) And (Mk3 > 230) And (Mk3 < 250)
Then
                Begin
                  CopyToPrn(#13#10);
                  CopyToPrn(#12);
                  CloseMouse;
                  ResetArea;
                  ReturnPic(220,150,430,280,Pic1);
                  OpenMouse;
                  Exit;
                End;
          End;

        If Line > 55 Then
          Begin
            Line := 1;
            CopyToPrn(#13#10);
            Str(page,PageSt);
```

```
                PageStatus(PageSt);
                For ColY := 1 to 35 Do
                  Begin
                     CopyToPrn(' ');
                  End;
                CopyToPrn('[Page '+PageSt+']');
                CopyToPrn(#13#10);
                CopyToPrn(#12);
                Inc(Page);
              End;
          End;
      CopyToPrn(#13#10);
      Str(page,PageSt);
      PageStatus(PageSt);
      For ColY := 1 to 35 Do
        Begin
          CopyToPrn(' ');
        End;
        CopyToPrn('[Page '+PageSt+']');
        CopyToPrn(#13#10);
      CopyToPrn(#12);

      CloseMouse;
      ResetArea;
      ReturnPic(220,150,430,280,Pic1);
      OpenMouse;
  End;

    { ************************************************************ }
    { ***************** Menu Load Data From File ***************** }
    { ************************************************************ }
```

```
Procedure Click_Load;
Var Pic3  : Pointer;
    x,y,k : Integer;
    Name  : String;
Begin
     CloseMouse;
     Box3d(155,400,215,420,7,0);
     OutTextExtra(167,407,15,'File');
     SavePic(220,100,430,335,Pic3);
     TabBar(220,100,420,325,3);
     Setcolor(12);
     Outtextxy(295,107,'File ...');
     Box3d(250,145,390,175,7,1);
     Box3d(250,230,390,260,7,1);
     Box3d(250,275,390,305,7,1);
     Setcolor(0);
     OuttextExtra(285,157,15,'Load File');
     OuttextExtra(257,242,15,'Change File Name');
     OutTextExtra(300,287,15,'Print');
     SetArea(220,120,420,325);
     Box(250,190,390,215,11);
     ReadFile_INI('FILE',Name);
     Name := Name + '.REC';
     y := Length(Name) Div 2;
     x := 320 - y*8;
     OutTextXY(x,200,Name);
     OpenMouse;
     Repeat
          CheckKey(x,y,k);
          If k = 27 Then                    { [ESC] }
```

```
  Begin
     CloseMouse;
     ReturnPic(220,100,430,335,Pic3);
     ResetArea;
     OpenMouse;
     k := 28;
  End;
If (k = 108) or (k = 76) Then        { l OR L }
  Begin
     CloseMouse;
     Box3d(250,145,390,175,7,0);
     OuttextExtra(285,157,15,'Load File');
     Delay(100);
     ReturnPic(220,100,430,335,Pic3);
     ResetArea;
     LoadRealTime;
     OpenMouse;
     K := 28;
  End;
If (k = 99) or (k = 67) Then         { C OR c }
  Begin
     CloseMouse;
     Box3d(250,230,390,260,7,0);
     OuttextExtra(257,242,15,'Change File Name');
     Delay(100);
     ReturnPic(220,100,430,335,Pic3);
     ResetArea;
     Input_Name;
     OpenMouse;
     K := 28;
  End;
```

```
If (k = 112) or (k = 80) Then        { P or p }
  Begin
      CloseMouse;
      Box3d(250,275,390,305,7,0);
      OutTextExtra(300,287,15,'Print');
      Delay(100);
      ReturnPic(220,100,430,335,Pic3);
      ResetArea;
      OpenMouse;
      Now_Print;
      K := 28;
  End;
If k = 0 Then
  Begin
      If (X > 250) And (X < 390) And (Y > 145) And (Y < 175) Then
        Begin
          CloseMouse;
          Box3d(250,145,390,175,7,0);
          OuttextExtra(285,157,15,'Load File');
          Delay(100);
          ReturnPic(220,100,430,335,Pic3);
          ResetArea;
          LoadRealTime;
          OpenMouse;
          K := 28;
        End;
      If (X > 250) And (X < 390) And (Y > 230) And (Y < 260) Then
        Begin
          CloseMouse;
          Box3d(250,230,390,260,7,0);
          OuttextExtra(257,242,15,'Change File Name');
```

```
                Delay(100);
                ReturnPic(220,100,430,335,Pic3);
                ResetArea;
                Input_Name;
                OpenMouse;
                K := 28;
              End;
            If (X > 250) And (X < 390) And (Y > 275) And (Y < 305) Then
              Begin
                CloseMouse;
                Box3d(250,275,390,305,7,0);
                OutTextExtra(300,287,15,'Print');
                Delay(100);
                ReturnPic(220,100,430,335,Pic3);
                ResetArea;
                Now_Print;
                OpenMouse;
                K := 28;
              End;

        End;
    Until k = 28;
    CloseMouse;
    Box3d(155,400,215,420,7,1);
    OutTextExtra(167,407,15,'File');
    OpenMouse;
End;

{ ************************************************************ }
{ ************** ConVert File Rec To File Ref **************** }
{ ************************************************************ }
```

```
Procedure FileRecToRef;
Var.    Fr    : Text;
        Name  : String;
        ret   : Integer;
        Iret  : Real;
Begin
    ReadFile_INI('FILE',Name);
    Assign(Fi,Name+'.REC');
    {$I-}
    Reset(Fi);
    {$I+}
    Return := IOResult;
    If Return <> 0 Then
      Begin
          CheckError(Return);
          CloseGraph;
          Writeln('Good Bye ...');
          Halt;
      End;
    Assign(Fr,Name+'.REF');
    {$I-}
    Rewrite(Fr);
    {$I+}
    Return := IOResult;
    If Return <> 0 Then
      Begin
          CheckError(Return);
          CloseGraph;
          Writeln('Good Bye ...');
          Halt;
```

```
      End;
    Readln(Fi,st);
    Writeln(Fr,st);
    Repeat
        Readln(Fi,st);
        If Length(st) < 4 Then
          Begin
            Val(st,Iret,ret);
            Iret := Iret /  5;
            Str(Iret:4:2,st);
            Writeln(Fr,st);
          End
          Else
            Writeln(Fr,st);
    Until EOF(Fi);
    Close(Fi);
    Close(Fr);
End;

  { ************************************************************ }
  { ********* Calculate Data And Converse to Vpp,Std ********** }
  { ************************************************************ }

Procedure Calculate;
Var  Ft,Fv              : Text;
    res,x,y             : Integer;
    std                 : array[1..10] OF Real;
    std1,std2,std4,z,i  : Real;
    std3,std5,lnx,lny   : Real;
    Name,Valve          : String;
Begin
```

```
ReadFile_INI('FILE',Name);
Assign(Fv,Name+'.VPP');
{$I-}
Rewrite(Fv);
{$I+}
Return := IOResult;
If Return <> 0 Then
  Begin
      CheckError(Return);
      CloseGraph;
      Writeln('Good Bye ...');
      Halt;
  End;
Assign(Ft,Name+'.STD');
{$I-}
Rewrite(Ft);
{$I+}
Return := IOResult;
If Return <> 0 Then
  Begin
      CheckError(Return);
      CloseGraph;
      Writeln('Good Bye ...');
      Halt;
  End;
Assign(Fi,Name+'.REC');
{$I-}
Reset(Fi);
{$I+}
Return := IOResult;
If Return <> 0 Then
```

```
  Begin
      CheckError(Return);
      CloseGraph;
      WriteIn('Good Bye ...');
      Halt;
  End;
ReadIn(Fi,st);
y := 0;
Repeat
    std1 := 0;
    For x := 1 to 10 Do
      Begin
        ReadIn(Fi,st);
        If Length(st) > 4 Then
          Begin
            x := 10;
            y := 2;
          End;
        Val(st,i,res);
        i := i / 5;
        std[x] := i;
        std1 := std[x] + std1;
      End;

  For x := 1 to 10 Do
    Begin
      For y := x+1 to 10 Do
        Begin
          If (std[x] > std[y]) Then
            Begin
              z := std[x];
```

```
                        std[x] := std[y];
                        std[y] := z;
                    End;
                End;
            End;
        If std[1] = 0 Then std[1] := 1;
        lny := std[10] / std[1];
        If lny = 0 Then lny := 1;
        lnx := 20 * (ln(lny) / ln(10));
{        lnx := lnx * 10;
        x := Round(lnx);      }
        Str(lnx:4:2,Valve);
        Writeln(Fv,Valve);

         std1 := sqr(std1);
         std3 := 0;
         For x := 1 to  10 Do
            Begin
                If y = 2 Then x := 10;
                std2 := sqr(std[x]);
                std3 := std3 + std2;
            End;

         std4 := (10 * std3) - std1;
         std5 := std4 / 90;
         std5 := sqrt(std5);
         Str(std5:4:2,Valve);
         Writeln(Ft,Valve);
    Until (length(st) > 4) or (EOF(Fi));
    Close(Ft);
    Close(Fi);
```

```
    Close(Fv);
End;

  { ********************************************************** }
  { ********************* Cursor Point X  ******************** }
  { ********************************************************** }

Procedure PointX(CursorX : Integer;Var RcursorX : Integer);
Begin
    RcursorX := CursorX * 5;
End;

  { ********************************************************** }
  { ********************* Cursor Point Y  ******************** }
  { ********************************************************** }

Procedure PointY(CursorY : Integer;Var RcursorY : Integer);
Begin
    RcursorY := CursorY * 4;
End;

  { ********************************************************** }
  { ***************** Menu Plot STD From File ***************** }
  { ********************************************************** }

Procedure Click_Plot;
Var Fs         : Text;
   code,ic,jc : Integer;
   Name        : String;
   Bx,By,Bk    : Integer;
   i,j         : Real;
```

```
Begin
    CloseMouse;
    Box3d(245,400,305,420,7,0);
    Outtextxy(257,407,'S T D');
    FileRecToRef;
    Calculate;
    ScaleSTD;
    ReadFile_INI('FILE',Name);
    Assign(Fi,Name+'.STD');
    Reset(Fi);
    Assign(Fs,Name+'.VPP');
    Reset(Fs);
    Repeat
        Readln(Fi,st);              { STD }
        val(st,i,code);
        i := i * 100;
        ic := Round(i);
        PointY(ic,ic);
        Readln(Fs,st);              { VPP }
        Val(st,j,code);
        j := j * 100;
        jc := Round(j);
        PointX(jc,jc);
        jc := jc Div 10;
        SetFillStyle(1,0);
        FillEllipse(55+ic,340-jc,2,2);
    Until EOF(Fi) or EOF(Fs);
    CheckKey(Bx,By,Bk);
    Box3d(245,400,305,420,7,1);
    Outtextxy(257,407,'S T D');
    Close(Fi);
    Close(Fs);
```

```
    ClearScreen;
    OpenMouse;
End;

  { ************************************************************ }
  { ******************* Menu Setup Output ********************** }
  { ************************************************************ }

Procedure Click_Setup;
Var Pic4                        : Pointer;
   ButtomA,ButtomB,KeyC,PushBox : Integer;
   str,st_ini,stON              : String;
Begin
    CloseMouse;
    SavePic(120,80,530,390,Pic4);
    Box3d(335,400,395,420,7,0);
    OutTextExtra(347,407,15,'Setup');
    TabBar(120,80,520,380,3);
    SetColor(12);
    OuttextXY(300,85,'Setup');
    Setcolor(0);
    SetArea(120,100,520,380);
    Box3d(440,340,500,360,7,1);
    Outtextxy(460,347,'OK.');
    ReadFile_INI('POINT',str);
    Click_Box(160,140,0);
    Click_BOx(160,200,0);
    Click_Box(160,260,0);
    Click_BOx(160,320,0);
    If str = '1'  Then Click_Box(160,140,1);
    If str = '5'  Then Click_BOx(160,200,1);
```

```
If str = '10' Then Click_Box(160,260,1);
If str = '20' Then Click_BOx(160,320,1);
OutTextXY(190,145,' 1 Point / Sec');
OutTextXY(190,205,' 5 Point / Sec');
OutTextXY(190,265,'10 Point / Sec');
OutTextXY(190,325,'20 Point / Sec');
ReadFile_INI('PAGE TO PAGE',stON);
If stON = 'ON' Then
  Click_Box(340,140,1)
Else
  Click_Box(340,140,0);
OutTextXY(370,145,'Page to Page');
Box(340,180,500,320,11);
OutTextExtra(370,195,1,'King');
OutTextExtra(370,215,1,'Mongkut`s');
OutTextExtra(370,235,1,'Institute of');
OutTextExtra(370,255,1,'Technology');
OutTextExtra(370,275,1,'Ladkrabang');
OutTextExtra(370,295,1,'2N. 1997');
OpenMouse;
Repeat
CheckKey(ButtomA,ButtomB,KeyC);
CloseMouse;
If KeyC = 0 Then
  Begin
    If (ButtomA > 440) And (ButtomA < 500) And (ButtomB > 340) And (ButtomB <
360) Then
      Begin
        KeyC := 13;
      End;
```

```
If (ButtomA > 160) And (ButtomA < 175) And (ButtomB > 140) And (ButtomB <
155) Then
        Begin
            st_ini := '1';
            Click_Box(160,140,1);
            Click_Box(160,200,0);
            Click_Box(160,260,0);
            Click_Box(160,320,0);
        End;
    If (ButtomA > 160) And (ButtomA < 175) And (ButtomB > 200) And (ButtomB <
215) Then
        Begin
            st_ini := '5';
            Click_Box(160,140,0);
            Click_Box(160,200,1);
            Click_Box(160,260,0);
            Click_Box(160,320,0);
        End;
    If (ButtomA > 160) And (ButtomA < 175) And (ButtomB > 260) And (ButtomB <
275) Then
        Begin
            st_ini := '10';
            Click_Box(160,140,0);
            Click_Box(160,200,0);
            Click_Box(160,260,1);
            Click_Box(160,320,0);
        End;
    If (ButtomA > 160) And (ButtomA < 175) And (ButtomB > 320) And (ButtomB <
335) Then
        Begin
            st_ini := '20';
```

```
            Click_Box(160,140,0);
            Click_Box(160,200,0);
            Click_Box(160,260,0);
            Click_Box(160,320,1);
          End;
        If (ButtomA > 340) And (ButtomA < 355) And (ButtomB > 140) And (ButtomB <
155) Then
          Begin
            Check_Box(340,140,PushBox);
            If PushBox = 0 Then
              Begin
                stON := 'OFF';
                Click_Box(340,140,0);
              End
              Else
                Begin
                  stON := 'ON';
                  Click_Box(340,140,1);
                End;
          End;
        Delay(200);
      End;
      If KeyC = 13 Then
        Begin
          KeyC := 28;
        End;
      OpenMouse;
    Until KeyC = 28;
    Check_Box(340,140,PushBox);
    If PushBox = 0 Then
      Begin
```

```
            stON := 'ON';
          End
        Else
          Begin
            stON := 'OFF';
          End;
        Check_Box(160,140,PushBox);
        If PushBox = 0 Then st_ini := '1';
        Check_Box(160,200,PushBox);
        If PushBox = 0 Then st_ini := '5';
        Check_Box(160,260,PushBox);
        If PushBox = 0 Then st_ini := '10';
        Check_Box(160,320,PushBox);
        If PushBox = 0 Then st_ini := '20';
        ReadFile_INI('FILE',str);
        WriteOption(st_ini,stON,str);
        CloseMouse;
        Box3d(440,340,500,360,7,0);
        Outtextxy(460,347,'OK.');
        Delay(100);
        Box3d(335,400,395,420,7,1);
        OutTextExtra(347,407,15,'Setup');
        ResetArea;
        ReturnPic(120,80,530,390,Pic4);          { **** }
        OpenMouse;
    End;

    { ********************************************************** }
    { ******************* Show About Program ******************* }
    { ********************************************************** }
```

```
Procedure Click_About;
Var Bot_A,Bot_B,Key_C : Integer;
   Pic5              : Pointer;
Begin
    CloseMouse;
    SavePic(200,100,450,330,Pic5);
    Box3d(425,400,485,420,7,0);
    OutTextExtra(435,407,15,'About');
    TabBar(200,100,440,320,11);
    Setcolor(12);
    Outtextxy(260,107,'About Version 1.1');
    Setcolor(1);
    Outtextxy(225,165,'Mr.Samart Tontithavornwat');
    Outtextxy(225,190,'Mr.Piya   Thirapanmethee');
    Setcolor(13);
    Outtextxy(210,140,'Program By..');
    Outtextxy(235,215,'Faculty of Engineering');
    Outtextxy(215,240,'Dept.Industrial Technology');
    Outtextxy(230,265,'Electrinics Technology');
    Setcolor(0);
    SetArea(200,120,440,320);
    Box3d(290,290,350,310,7,1);
    Outtextxy(313,297,'OK.');
    OpenMouse;
    Repeat
    CheckKey(Bot_A,Bot_B,Key_C);
    If Key_C = 0 Then
      Begin
        If (Bot_A > 290) And (Bot_A < 350) And (Bot_B > 290) And (Bot_B < 310) Then
          Begin
          Key_C := 13;
```

```
        End;
      End;
    If Key_C = 13 Then
      Begin
        Key_C := 28;
      End;
    Until Key_C = 28;
    CloseMouse;
    Box3d(290,290,350,310,7,0);
    Outtextxy(313,297,'OK.');
    Delay(100);
    Box3d(425,400,485,420,7,1);
    OutTextExtra(435,407,15,'About');
    ResetArea;
    ReturnPic(200,100,450,330,Pic5);
    OpenMouse;
End;

  { ********************************************************** }
  { ***************** Menu Exit Program ****************** }
  { ********************************************************** }

Procedure Click_Exit;
Var Mx,My,Ck : Integer;
   Pic6     : Pointer;
Begin
    CloseMouse;
    Box3d(515,400,575,420,7,0);
    OutTextExtra(522,407,15,'Exit !');
    SavePic(210,185,435,305,Pic6);
    TabBar(220,190,420,290,11);
```

```
Box3d(240,240,300,260,7,1);
Box3d(340,240,400,260,7,1);
SetColor(12);
OuttextXY(300,198,'Exit !');
Setcolor(0);
OuttextExtra(258,248,15,'Yes');
OuttextExtra(362,248,15,'No');
SetArea(220,210,420,290);
OpenMouse;
Repeat
    CheckKey(Mx,My,Ck);
  If Ck = 0 Then
    Begin
    If (Mx > 240) And (Mx < 300) And (My > 240) And (My < 260) Then
      Begin
        CloseMouse;
        Box3d(240,240,300,260,7,0);
        OuttextExtra(258,248,15,'Yes');
        Delay(100);
        Closegraph;
        Writeln('Good Bye...');
        Halt;
      End;
    If (Mx > 340) And (Mx < 400) And (My > 240) And (My < 260) Then
      Begin
        CloseMouse;
        Box3d(340,240,400,260,7,0);
        OuttextExtra(362,248,15,'No');
        Delay(100);
        Box3d(340,240,400,260,7,1);
        OuttextExtra(362,248,15,'No');
```

```
            OpenMouse;
            Ck := 27;
        End;
    End;
  If Ck = 13 Then                    { Enter }
    Begin
            CloseMouse;
            Box3d(240,240,300,260,7,0);
            OuttextExtra(258,248,15,'Yes');
            Delay(100);
            Closegraph;
            Writeln('Good Bye...');
            Halt;
    End;
  If (Ck = 121) or (Ck = 89) Then      { Y or y }
    Begin
          CloseMouse;
          Box3d(240,240,300,260,7,0);
          OuttextExtra(258,248,15,'Yes');
          Delay(100);
          Closegraph;
          Writeln('Good Bye...');
          Halt;
    End;
  IF (Ck = 110) or (Ck = 78) Then      { N or n }
    Begin
          CloseMouse;
          Box3d(340,240,400,260,7,0);
          OuttextExtra(362,248,15,'No');
          Delay(100);
          Box3d(340,240,400,260,7,1);
```

```
            OuttextExtra(362,248,15,'No');
            OpenMouse;
            Ck := 27;
        End;
    Until Ck = 27;
    ResetArea;
    CloseMouse;
    ReturnPic(210,185,435,305,Pic6);
    Box3d(515,400,575,420,7,1);
    OutTextExtra(522,407,15,'Exit !');
    OpenMouse;
End;

  { ************************************************************ }
  { ******************* Initial File INI ********************** }
  { ************************************************************ }

Procedure OpenFile_INI;
Var  x,y,key                : Integer;
Begin
    Assign(Fi,'KMITL.INI');
    {$I-}
        Rewrite(Fi);
    {$I+}
    Return := IOResult;
    If Return <> 0 Then
      Begin
          CheckError(Return);
          Closegraph;
          Writeln('Good Bye ...');
          Halt;
```

```
        SetColor(0);
      End;
    Writeln(Fi,'[Amplitude Scintillation Recorder of C and KU band Satellite Signal]');
    Writeln(Fi,'');
    Writeln(Fi,'POINT = 20');
    Writeln(Fi,'PAGE TO PAGE = ON');
    Writeln(Fi,'FILE = DATA');
    Writeln(Fi,'');
    Write(Fi,'[End Of KMITL.INI]');
    Close(Fi);
End;

  { ************************************************************ }
  { ***************** Check Data in File INI ****************** }
  { ************************************************************ }

Procedure Check_File_INI;
Var  x,y,key       : Integer;
   Pic8          : Pointer;
Begin
   Assign(fi,'KMITL.INI');
   {$I-}
      Reset(Fi);
   {$I+}
   Return := IOResult;
   If Return <> 0 Then
     Begin
        CheckError(Return);
        OpenFile_INI;
     End
     Else
```

```
    Begin
      Readln(Fi,st);
      Close(Fi);
      If st <> '[Amplitude Scintillation Recorder of C and KU band Satellite Signal]'
Then
        Begin
          SavePic(210,185,435,305,Pic8);
          TabBar(220,190,420,290,11);
          SetColor(12);
          OuttextXY(300,198,'Error !');
          SetColor(12);
          OutTextXy(260,250,'File INI Error !');
          Writeln(#07);
          Repeat
            CheckKey(x,y,key);
          Until key <> 0;
          ReturnPic(210,185,435,305,Pic8);
          SetColor(0);
          OpenFile_INI;
        End;
    End;
End;

  { ********************************************************** }
  { ********** Open Graphics VGA 640 * 480 : 16 Color ********** }
  { ********************************************************** }

Procedure OpenGraph_Vga;
Var CrtType : Byte absolute $0040:$0049;
Begin
  If CrtType <> 3 Then
```

```
    Begin                                   { If crt not match }
     Writeln(#7,'This Program Need [EGA..VGA] Card Only !');
     Halt(0);
    End;
   grdriver:=vga;
   grmode:=vgahi;
   initgraph(grdriver,grmode,'');
   If graphresult < 0 Then
    Begin
     Writeln(#7,'Error : ',GraphErrorMsg(GraphResult));
     Halt(1);
    End;
End;

Procedure Init_Mouse;
Begin
    Regs.Ax := 0;
    Intr($33,Regs);                 { Check Initial Mouse }
    If Regs.Ax = 0 Then
      Begin
         Writeln(#7,'Mouse Driver Not Install !');
         Halt(0);
      End;


End;

  { ************************************************************ }
  { ************************  Main  ************************** }
  { ************************************************************ }
```

```
Var  MouseX,MouseY,ChKey : Integer;

Begin

    Init_Mouse;                    { Check Mouse Driver }

    OpenGraph_Vga;                 { Initial Graphics }

    Screen;                        { Show Screen }

    Check_File_INI;                { Check File INI }

    OpenMouse;

    Repeat
        CheckKey(MouseX,MouseY,ChKey);

        If ChKey = 0 Then
          Begin

          If (MouseX > 65) And (MouseX < 125) And (MouseY > 400)
            And (MouseY < 420) Then
            Begin
                Click_Read;
            End;

          If (MouseX > 155) And (MouseX < 215) And (MouseY > 400)
            And (MouseY < 420) Then
            Begin
                Click_Load;
            End;
```

```
If (MouseX > 245) And (MouseX < 305) And (MouseY > 400)
  And (MouseY < 420) Then
  Begin
     Click_Plot;
  End;

If (MouseX > 335) And (MouseX < 395) And (MouseY > 400)
  And (MouseY < 420) Then
  Begin
     Click_Setup;
  End;

If (MouseX > 425) And (MouseX < 485) And (MouseY > 400)
  And (MouseY < 420) Then
  Begin
     Click_About;
  End;

If (MouseX > 515) And (MouseX < 575) And (MouseY > 400)
  And (MouseY < 420) Then
  Begin
     Click_Exit;
  End;

If (MouseX > 11) And (MouseX < 31) And (MouseY > 11)
  And (MouseY < 31) Then
  Begin
     Click_Exit;
  End;
```

```
        End;
      If Chkey  = 27 Then                    { [ESC]  }
        Begin
            Click_exit;
        End;
      If (Chkey = 114) or (chkey = 82) Then      { R or r }
        Begin
            Click_Read;
        End;
      If (Chkey = 102) or (chkey = 70) Then      { F or f }
        Begin
            Click_Load;
        End;
      If (Chkey = 97) or (chkey = 65) Then       { A or a }
        Begin
            Click_About;
        End;
      If (Chkey = 101) or (chkey = 69) Then      { E or e }
        Begin
            Click_Exit;
        End;
      If (Chkey = 115) or (chkey = 83) Then      { S or s }
        Begin
            Click_Setup;
        End;
    Until (ChKey = 28);

End.
```

National Semiconductor

A to D, D to A

ADC0801, ADC0802, ADC0803, ADC0804 8-Bit μP Compatible A/D Converters

General Description

The ADC0801, ADC0802, ADC0803, ADC0804 are CMOS 8-bit, successive approximation A/D converters, which use a modified potentiometric ladder—similar to the 256R products. They are designed to meet the NSC MICROBUS™ standard to allow operation with the 8080A control bus, and TRI-STATE® output latches directly drive the data bus. These A/Ds appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed.

A new differential analog voltage input allows increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

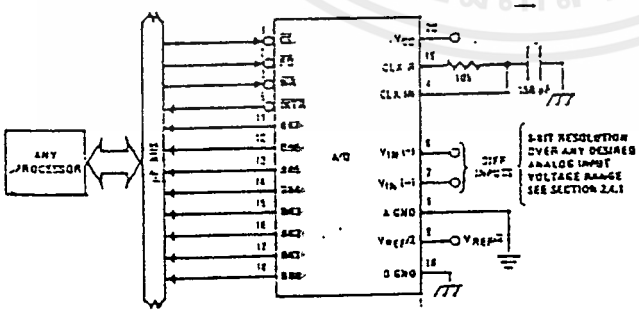
Features

- MICROBUS (8080A) compatible—no interfacing logic needed
- Easy interface to all microprocessors, or operates "stand alone"
- Differential analog voltage inputs
- Logic inputs and outputs meet T^2L voltage level specifications
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3" standard width 20-pin DIP package

Key Specifications

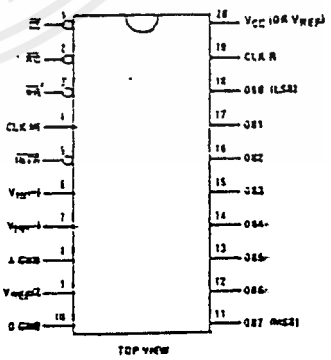
- Resolution 8 bits
- Total error ±1/4 LSB, ±1/2 LSB and ±1 LSB
- Conversion time 100 μs
- Access time 135 ns
- Single supply 5 V_{DC}
- Operates ratiometrically or with 5 V_{DC}, 2.5 V_{DC}, or analog span adjusted voltage reference

Typical Application



Connection Diagram

ADC 080X
Dual-In-Line Package



Absolute Maximum Ratings (Notes 1 and 2)

Supply Voltage (V_{CC}) (Note 3)	6.5V
Voltage at Any Input	−0.3V to (V_{CC} +0.3V)
Storage Temperature Range	−65°C to +150°C
Package Dissipation at T_A = 25°C	875 mW
Lead Temperature (Soldering, 10 seconds)	300°C

Operating Ratings (Notes 1 and 2)

Temperature Range (Note 1)	$T_{MIN} \leq T_A \leq T_{MAX}$
ADC0801/02/03 LD	−55°C ≤ T_A ≤ +125°C
ADC0801/02/03/04 LCD	−40°C ≤ T_A ≤ +85°C
ADC0801/02/03/04 LCN	0°C ≤ T_A ≤ 70°C
Range of V_{CC} (Note 1)	4.5 V_{DC} to 6.3 V_{DC}

Electrical Characteristics

Converter Specifications:

V_{CC} = 5 V_{DC}, $V_{REF}/2$ = 2.500 V_{DC}, $T_{MIN} \leq T_A \leq T_{MAX}$ and f_{CLK} = 640 kHz unless otherwise stated.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
ADC0801: Total Adjusted Error (Note 8)	With Full-Scale Adj.			±1/4	LSB
ADC0802: Total Unadjusted Error (Note 8)	Completely Unadjusted			±1/2	LSB
ADC0803: Total Adjusted Error (Note 8)	With Full-Scale Adj.			±1/2	LSB
ADC0804: Total Unadjusted Error (Note 8)	Completely Unadjusted			±1	LSB
$V_{REF}/2$ Input Resistance	Input Resistance at Pin 9	1.0	1.3		kΩ
Analog Input Voltage Range	(Note 4) V(+) or V(−)	Gnd−0.05		V_{CC}+0.05	V
DC Common-Mode Rejection	Over Analog Input Voltage Range		±1/16	±1/8	LSB
Power Supply Sensitivity	V_{CC} = 5 V_{DC} ±10% Over Allowed $V_{IN}(+)$ and $V_{IN}(-)$ Voltage Range (Note 4)		±1/16	±1/8	LSB

Electrical Characteristics

Timing Specifications: V_{CC} = 5 V_{DC} and T_A = 25°C unless otherwise noted.

	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNIT
f_{CLK}	Clock Frequency	V_{CC} = 5V, (Note 5)	100	640	1280	kHz
		V_{CC} = 5V	100	640	800	kHz
T_c	Conversion Time	(Note 6)	66		73	1/f_{CLK}
CR	Conversion Rate in Free-Running Mode	$\overline{INTR}$ tied to $\overline{WR}$ with $\overline{CS}$ = 0 V_{DC}, f_{CLK} = 640 kHz			8770	Hz
t_{WI}/t_{WR}L	Width of $\overline{WR}$ Input (Start Pulse Width)	$\overline{CS}$ = 0 V_{DC} (Note 7)	100			ns
ACC	Access Time (Delay from Falling Edge of $\overline{RD}$ to Output Data Valid)	C_L = 100 pF (Use Bus Driver IC for Larger C_L)		135	200	ns
t_{1H}, t_{0H}	TRI-STATE Control (Delay from Rising Edge of $\overline{RD}$ to Hi-Z-State)	C_L = 10 pF, R_L = 10k (See TRI-STATE Test Circuits)		125	250	ns
t_{WI}	Delay from Falling Edge of $\overline{WR}$ to Reset of $\overline{INTR}$			300	450	ns
C_{IN}	Input Capacitance of Logic Control Inputs			5	7.5	pF
C_{OUT}	TRI-STATE Output Capacitance (Data Buffers)			5	7.5	pF

Electrical Characteristics

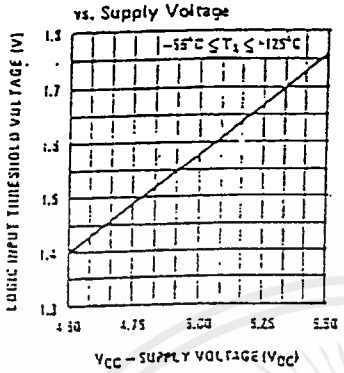
Digital Levels and DC Specifications:

V_{CC} = 5 V_{DC} and $T_{MIN} \leq T_A \leq T_{MAX}$, unless otherwise noted.

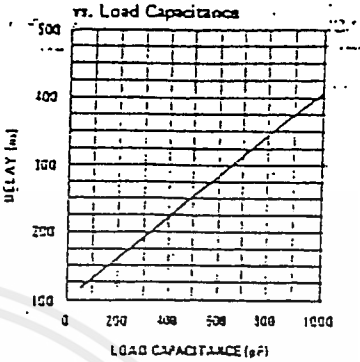
PARAMETER		CONDITIONS	MIN	TYP	MAX	UNITS
CONTROL INPUTS (Note: CLK IN (pin 4) is the input of a Schmitt trigger circuit and is therefore specified separately)						
V_{IN} (1)	Logical "1" Input Voltage (Except Pin 4 CLK IN)	V_{CC} = 5.25 V_{DC}	2.0		15	V_{DC}
V_{IN} (0)	Logical "0" Input Voltage (Except Pin 4 CLK IN)	V_{CC} = 4.75 V_{DC}			0.8	V_{DC}
V_{T^+}	CLK IN (Pin 4) Positive Going Threshold Voltage		2.7	3.1	3.5	V_{DC}
V_{T^-}	CLK IN (Pin 4) Negative Going Threshold Voltage		1.5	1.8	2.1	V_{DC}
V_H	CLK IN (Pin 4) Hysteresis $(V_{T^+}) - (V_{T^-})$		0.6	1.3	2.0	V_{DC}
I_{IN} (1)	Logical "1" Input Current (All Inputs)	V_{IN} = 5 V_{DC}		0.005	1	μA_{DC}
I_{IN} (0)	Logical "0" Input Current (All Inputs)	V_{IN} = 0 V_{DC}	−1	−0.005		μA_{DC}
I_{CC}	Supply Current (Includes Ladder Current)	f_{CLK} = 640 kHz, T_A = 25°C and $\overline{CS}$ = "1"		1.3	2.5	mA
DATA OUTPUTS AND $\overline{INTR}$						
V_{OUT} (0)	Logical "0" Output Voltage	I_O = 1.6 mA, V_{CC} = 4.75 V_{DC}			0.4	V_{DC}
V_{OUT} (1)	Logical "1" Output Voltage	I_O = −360 µA, V_{CC} = 4.75 V_{DC}	2.4			V_{DC}
I_{OUT}	TRI-STATE Disabled Output Leakage (All Data Buffers)	V_{OUT} = 0 V_{DC}	−3			μA_{DC}
		V_{OUT} = 5 V_{DC}			3	μA_{DC}
	Output Short Circuit Current	T_A = 25°C				
I_{SOURCE}		V_{OUT} Short to Gnd	4.5	6		mA_{DC}
I_{SINK}		V_{OUT} Short to V_{CC}	9.0	16		mA_{DC}

Typical Performance Characteristics

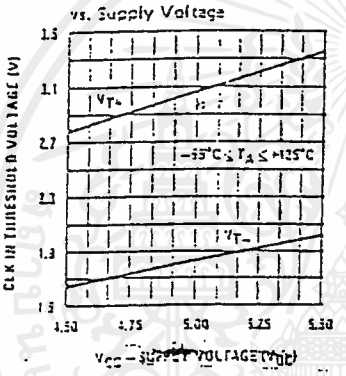
Logic Input Threshold Voltage vs. Supply Voltage



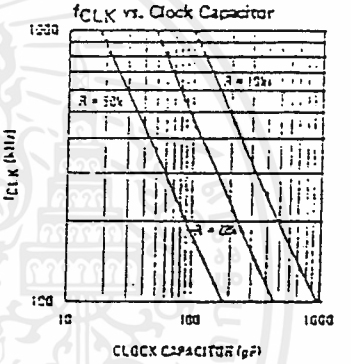
Delay From Falling Edge of $\overline{RD}$ to Output Data Valid vs. Load Capacitance



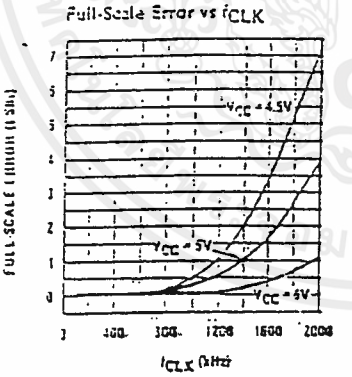
CLK IN Schmitt Trip Levels vs. Supply Voltage



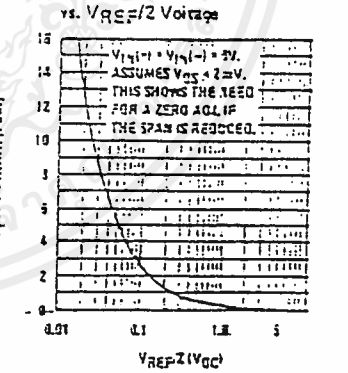
f_{CLK} vs. Clock Capacitor



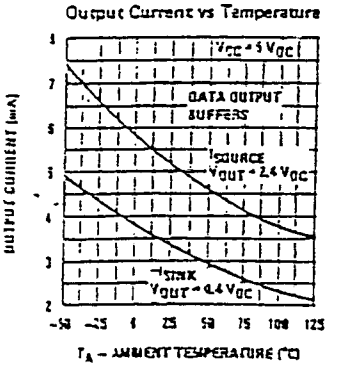
Full-Scale Error vs f_{CLK}



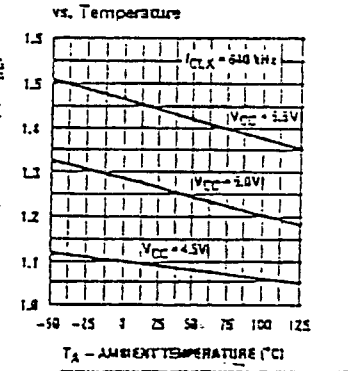
Effect of Unadjusted Offset Error vs. $V_{REF}/2$ Voltage



Output Current vs Temperature

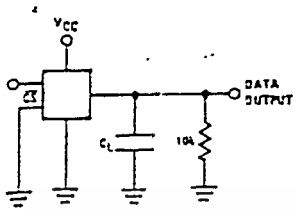


Power Supply Current vs. Temperature

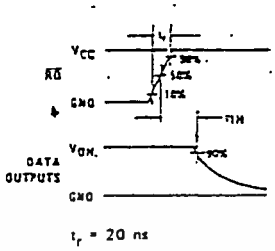


TRI-STATE® Test Circuits and Waveforms

t_{1H}

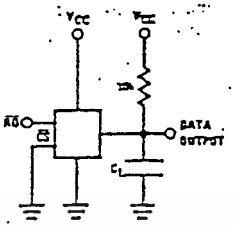


t_{1H}, C_L = 10 pF

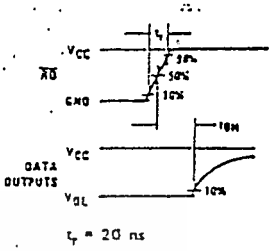


t_r = 20 ns

t_{0H}

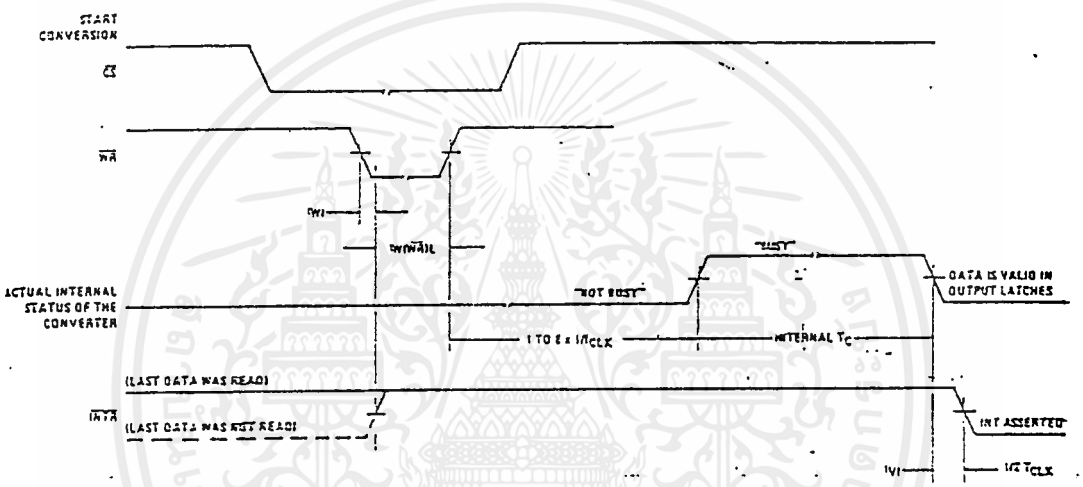


t_{0H}, C_L = 10 pF

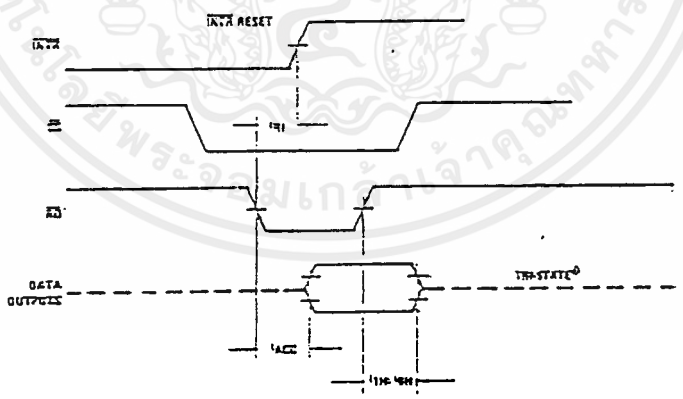


t_r = 20 ns

Timing Diagrams



Output Enable and Reset $\overline{INTR}$



Note: All timing is measured from the 50% voltage points.

1.0 UNDERSTANDING A/D ERROR SPECS

A perfect A/D transfer characteristic (staircase waveform) is shown in *Figure 1a*. The horizontal scale is analog input voltage and the particular points labeled are in steps of 1 LSB (19.53 mV with 2.5V tied to the $V_{REF}/2$ pin). The digital output codes which correspond to these inputs are shown as D−1, D, and D+1. For the perfect A/D, not only will center-value (A−1, A, A+1, . . .) analog inputs produce the correct output digital codes, but also each riser (the transitions between adjacent output codes) will be located ±1/2 LSB away from each center-value. As shown, the risers are ideal and have no width. Correct digital output codes will be provided for a range of analog input voltages which extend ±1/2 LSB from the ideal center-values. Each tread (the range of analog input voltage which provides the same digital output code) is therefore 1 LSB wide.

Figure 1b shows worst case error plot for ADC0801. All center-valued inputs are guaranteed to produce the correct output codes and the adjacent risers are guaranteed to be no closer to the center-value points than ±1/4 LSB. In other words, if we apply an analog input equal to the center-value ±1/4 LSB, *we guarantee* that the A/D will produce the correct digital code. The maximum range of the position of the code transition is indicated by the horizontal arrow and it is guaranteed to be no more than 1/2 LSB.

The error curve of *Figure 1c* shows worst case error plot for ADC0802. Here we guarantee that if we apply an analog input equal to the LSB analog voltage center-value the A/D will produce the correct digital code.

Next to each transfer function is shown the corresponding error plot. Many people may be more familiar with error plots than transfer functions. The analog input voltage to the A/D is provided by either a linear ramp or by the discrete output steps of a high resolution DAC. Notice that the error is continuously displayed and includes the quantization uncertainty of the A/D. For example the error at point 1 of *Figure 1a* is +1/2 LSB because the digital code appeared 1/2 LSB in advance of the center-value of the tread. The error plots always have a constant negative slope and the abrupt upside steps are always 1 LSB in magnitude.

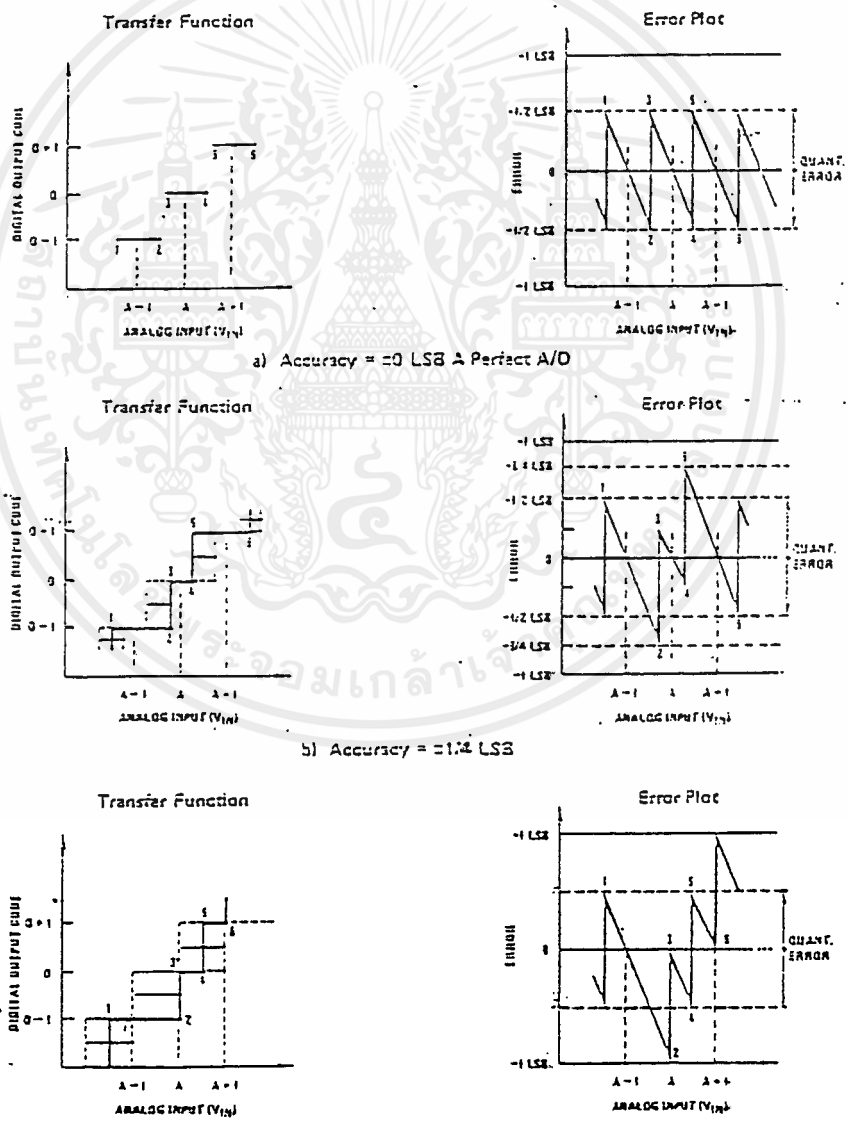


FIGURE 1. Clarifying the Error Specs of an A/D Converter

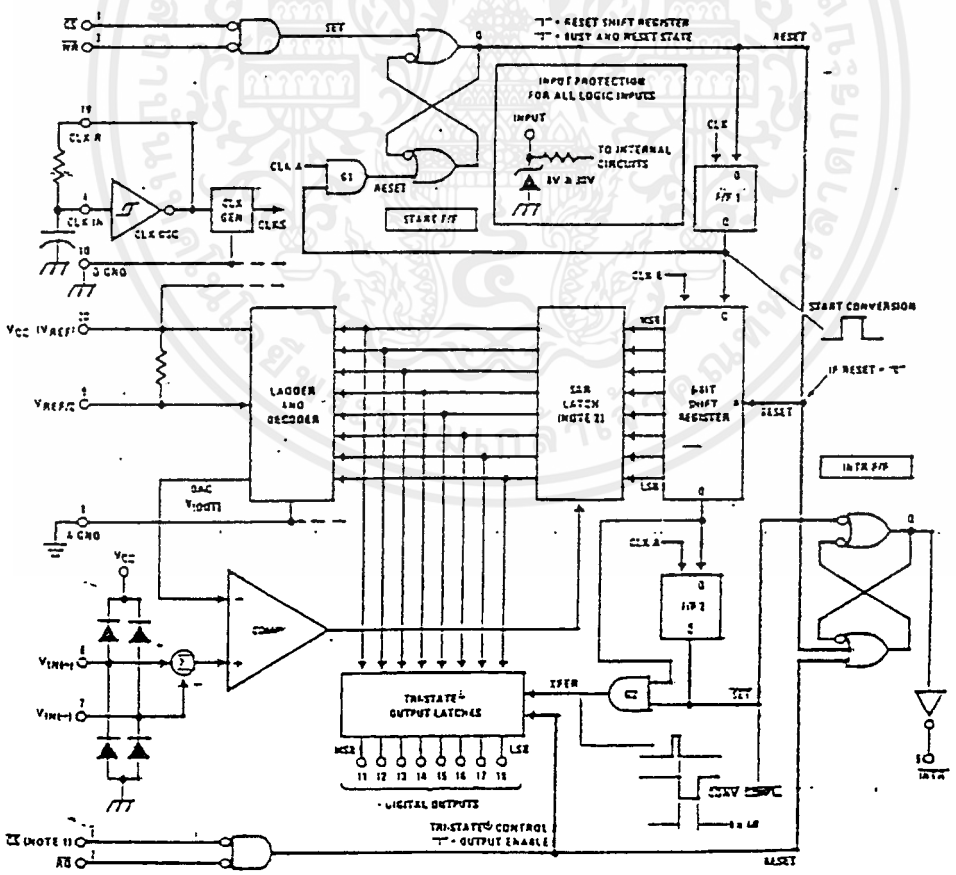
2.0 FUNCTIONAL DESCRIPTION

The ADC0801 series contains a circuit equivalent of the 256R network. Analog switches are sequenced by successive approximation logic to match the analog difference input voltage $[V_{IN}(+) - V_{IN}(-)]$ to a corresponding tap on the R network. The most significant bit is tested first and after 8 comparisons (64 clock cycles) a digital 8-bit binary code (1111 1111 = full-scale) is transferred to an output latch and then an interrupt is asserted ($\overline{INTR}$ makes a high-to-low transition). The device may be operated in the free-running mode by connecting $\overline{INTR}$ to the $\overline{WR}$ input with $\overline{CS}$ = 0. To insure start-up under all possible conditions, an external $\overline{WR}$ pulse is required during the first power-up cycle. A conversion in process can be interrupted by issuing a second start command.

On the high-to-low transition of the $\overline{WR}$ input the internal SAR latches and the shift register stages are reset. As long as the $\overline{CS}$ input and $\overline{WR}$ input remain low, the A/D will remain in a reset state. *Conversion will start from 1 to 8 clock periods after at least one of these inputs makes a low-to-high transition.*

A functional diagram of the A/D converter is shown in *Figure 2*. All of the package pinouts are shown and the major logic control paths are drawn in heavier weight lines.

The converter is started by having $\overline{CS}$ and $\overline{WR}$ simultaneously low. This sets the start flip-flop (F/F) and the resulting "1" level resets the 8-bit shift register, resets the Interrupt ($\overline{INTR}$) F/F and inputs a "1" to the D flop, F/F1, which is at the input end of the 8-bit shift register. Internal clock signals then transfer this "1" to the Q output of F/F1. The AND gate, G1, combines this "1" output with a clock signal to provide a reset signal to the start F/F. If the set signal is no longer present (either $\overline{WR}$ or $\overline{CS}$ is a "1") the start F/F is reset and the 8-bit shift register then can have the "1" clocked in, which starts the conversion process. If the set signal were to still be present, this reset pulse would have no effect (both outputs of the start F/F would momentarily be at a "1" level) and the 8-bit shift register would continue to be held in the reset mode. This logic therefore allows for wide $\overline{CS}$ and $\overline{WR}$ signals and the converter will start after at least one of these signals returns high and the internal clocks again provide a reset signal for the start F/F.



Note 1: $\overline{CS}$ shown twice for clarity.

Note 2: SAR = Successive Approximation Register.

FIGURE 2. Block Diagram

After the "1" is clocked through the 8-bit shift register (which completes the SAR search) it appears as the input to the D flop, F/F 2. As soon as this "1" is output from the shift register, the AND gate, G2, causes the new digital word to transfer to the TRI-STATE output latches. When F/F 2 is subsequently clocked, the $\overline{Q}$ output makes a high-to-low transition which causes the INTR F/F to set. An inverting buffer then supplies the $\overline{INTR}$ output signal.

When data is to be read, the combination of both $\overline{CS}$ and $\overline{RD}$ being low will cause the INTR F/F to be reset and the TRI-STATE output latches will be enabled to provide the 8-bit digital outputs.

2.1 Digital Control Inputs

The digital control inputs ($\overline{CS}$, $\overline{RD}$, and $\overline{WR}$) meet standard T^2L logic voltage levels. These signals have been renamed when compared to the standard A/D Start and Output Enable labels. In addition, these inputs are active low to allow an easy interface to microprocessor control busses. For non-microprocessor based applications, the $\overline{CS}$ input (pin 1) can be grounded and the standard A/D Start function is obtained by an active low pulse applied at the $\overline{WR}$ input (pin 3) and the Output Enable function is caused by an active low pulse at the $\overline{RD}$ input (pin 2).

2.2 Analog Differential Voltage Inputs and Common-Mode Rejection

This A/D has additional applications flexibility due to the analog differential voltage input. The $V_{IN}(-)$ input (pin 7) can be used to automatically subtract a fixed voltage value from the input reading (tare correction). This is also useful in 4 mA—20 mA current loop conversion. In addition, common-mode noise can be reduced by use of the differential input.

The time interval between sampling $V_{IN}(+)$ and $V_{IN}(-)$ is 4-1/2 clock periods. The maximum error voltage due to this slight time difference between the input voltage samples is given by:

$$\Delta V_{\epsilon}(MAX) = (Vp)\,(2\pi f_{cm})\left(\frac{4.5}{f_{CLK}}\right)$$

where:

ΔV_{ϵ} is the error voltage due to sampling delay

Vp is the peak value of the common-mode voltage

f_{cm} is the common-mode frequency

As an example, to keep this error to 1/4 LSB (~5 mV) when operating with a 60 Hz common-mode frequency, f_{cm}, and using a 640 kHz A/D clock, f_{CLK}, would allow a peak value of the common-mode voltage, Vp, which is given by:

$$Vp = \frac{(\Delta V_{\epsilon(MAX)})(f_{CLK})}{(2\pi f_{cm})\,(4.5)}$$

or

$$Vp = \frac{(5 \times 10^{-3})\,(640 \times 10^{3})}{(6.28)\,(60)\,(4.5)}$$

which gives

$$Vp \cong 1.9V.$$

The allowed range of analog input voltages usually places more severe restrictions on input common-mode noise levels.

An analog input voltage with a reduced span and a relatively large zero offset can be easily handled by making use of the differential input (see section 2.4 Reference Voltage Flexibility).

2.3 Analog Inputs

2.3.1 Input Current

Due to the internal switching action, displacement currents will flow at the analog inputs. This is due to on-chip stray capacitance to ground. The voltage on this capacitance is switched and will result in currents entering the $V_{IN}(+)$ input and leaving the $V_{IN}(-)$ input which will depend on the analog differential input voltage levels. These current transients occur at the leading edge of the internal clocks. They rapidly decay and *do not cause errors* as the on-chip comparator is strobed at the end of the clock period.

2.3.2 Input Bypass Capacitors

Bypass capacitors at the inputs will average these charges and, cause a DC current to flow through the output resistances of the analog signal sources. This charge pumping action is worse for continuous conversions with the $V_{IN}(+)$ input voltage at full-scale. For continuous conversions with a 640 kHz clock frequency with the $V_{IN}(+)$ input at 5V, this DC current is at a maximum of approximately 5 μA. Therefore, *bypass capacitors should not be used at the analog inputs or the $V_{REF}/2$ pin* for high resistance sources (> 1 kΩ). If input bypass capacitors are necessary for noise filtering and high source resistance is desirable to minimize capacitor size, the detrimental effects of the voltage drop across this input resistance, which is due to the average value of the input current, can be eliminated with a full-scale adjustment while the given source resistor and input bypass capacitor are both in place. This is possible because the average value of the input current is a precise linear function of the differential input voltage.

2.3.3 Input Source Resistance

Large values of source resistance where an input bypass capacitor is not used, *will not cause errors* as the input currents settle out prior to the comparison time. If a low pass filter is required in the system, use a low-valued series resistor (≤ 1 kΩ) for a passive RC section or add an op amp RC active low pass filter. For low source resistance applications, (≤ 1 kΩ), a 0.1 μF bypass capacitor at the inputs will prevent pickup due to series lead inductance of a long wire. A 100Ω series resistor can be used to isolate this capacitor—both the R and C are placed outside the feedback loop—from the output of an op amp, if used.

input. Zero error is the difference between the actual DC input voltage which is necessary to just cause an output digital code transition from 0000 0000 to 0000 0001 and the ideal 1/2 LSB value (1/2 LSB = 9.8 mV for $V_{REF}/2$ = 2.500 V_{DC}).

2.5.2 Full-Scale

The full-scale adjustment can be made by applying a differential input voltage which is 1-1/2 LSB down from the desired analog full-scale voltage range and then adjusting the magnitude of the $V_{REF}/2$ input (pin 9) for a digital output code which is just changing from 1111 1110 to 1111 1111. When offsetting the zero and using a span adjusted $V_{REF}/2$ voltage, the full-scale adjustment is made by inputting V_{MIN} to the V_{IN} (−) input of the A/D and applying a voltage to the V_{IN} (+) input which is given by:

$$V_{IN}(+)\text{ fs adj} = V_{MAX} - 1.5\left[\frac{(V_{MAX}-V_{MIN})}{256}\right]$$

where:

V_{MAX} = The high end of the analog input range

and

V_{MIN} = the low end (the offset zero) of the analog range. (Both are ground referenced.)

2.6 Clocking Option

The clock for the A/D can be derived from the CPU clock or an external RC can be added to provide self-clocking. The CLK IN (pin 4) makes use of a Schmitt trigger as shown in *Figure 4*.

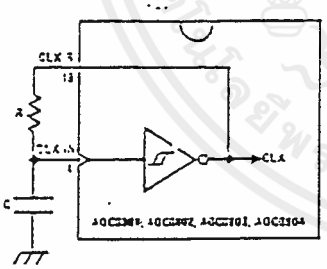


FIGURE 4. Self-Clocking the A/D

Heavy capacitive or DC loading of the clock R pin should be avoided as this will disturb normal converter operation. Loads less than 50 pF, such as driving up to 7 A/D converter clock inputs from a single clock R pin of 1 converter, are allowed. For larger clock line loading, a CMOS or low power T^2L buffer or PNP input logic should be used to minimize the loading on the clock R pin (do not use a standard T^2L buffer).

2.7 Restart During a Conversion

If the A/D is restarted ($\overline{CS}$ and $\overline{WR}$ go low and return high) during a conversion, the converter is reset and a new conversion is started. The output data latch is not updated if the conversion in process is not allowed to be completed, therefore the data of the previous conversion remains in this latch.

2.8 Continuous Conversions

For operation in the free-running mode an initializing pulse should be used, following power-up, to insure circuit operation. In this application, the $\overline{CS}$ input is grounded and the $\overline{WR}$ input is tied to the $\overline{INTR}$ output. This $\overline{WR}$ and $\overline{INTR}$ node should be momentarily forced to logic low following a power-up cycle to guarantee operation.

2.9 Driving the Data Bus

This MOS A/D, like MOS microprocessors and memories, will require a bus driver when the total capacitance of the data bus gets large. Other circuitry, which is tied to the data bus, will add to the total capacitive loading, even in TRI-STATE (high impedance mode). Backplane bussing also greatly adds to the stray capacitance of the data bus.

There are some alternatives available to the designer to handle this problem. Basically, the capacitive loading of the data bus slows down the response time, even though DC specifications are still met. For systems operating with a relatively slow CPU clock frequency, more time is available in which to establish proper logic levels on the bus and therefore higher capacitive loads can be driven (see typical characteristics curves).

At higher CPU clock frequencies time can be extended for I/O reads (and/or writes) by inserting wait states (8030) or using clock extending circuitry (8300).

Finally, if time is short and capacitive loading is high, external bus drivers must be used. These can be TRI-STATE buffers (low power Schottky is recommended such as the DM74LS240 series) or special higher drive current products which are designed as bus drivers. High current bipolar bus drivers with PNP inputs are recommended.

2.10 Power Supplies

Noise spikes on the V_{CC} supply line can cause conversion errors as the comparator will respond to this noise. A low inductance tantalum filter capacitor should be used close to the converter V_{CC} pin and values of 1 μF or greater are recommended. If an unregulated voltage is available in the system, a separate LM340LAZ-5.0 in TO-92, 5V voltage regulator for the converter (and other analog circuitry) will greatly reduce digital noise on the V_{CC} supply.

2.11 Wiring and Hook-Up Precautions

Standard digital wire wrap sockets are not satisfactory for breadboarding this A/D converter. Sockets on PC boards can be used and all logic signal wires and leads should be grouped and kept as far away as possible from the analog signal leads. Exposed leads to the analog inputs can cause undesired digital noise and hum pickup, therefore shielded leads may be necessary in many applications.

A single point analog ground should be used which is separate from the logic ground points. The power supply bypass capacitor and the self-clocking capacitor (if used) should both be returned to digital ground. Any $V_{REF}/2$ bypass capacitors, analog input filter capacitors, or input signal shielding should be returned to the analog ground point. A test for proper grounding is to measure the zero error of the A/D converter. Zero errors in excess of 1/4 LSB can usually be traced to improper board layout and wiring (see section 2.5.1 for measuring the zero error).

3.0 TESTING THE A/D CONVERTER

There are many degrees of complexity associated with testing an A/D converter. One of the simplest tests is to apply a known analog input voltage to the converter and use LEDs to display the resulting digital output code as shown in *Figure 5*.

For ease of testing, the $V_{REF}/2$ (pin 9) should be supplied with 2.560 V_{DC} and a V_{CC} supply voltage of 5.12 V_{DC} should be used. This provides an LSB value of 20 mV.

If a full-scale adjustment is to be made, an analog input voltage of 5.090 V_{DC} (5.120 − 1 1/2 LSB) should be applied to the $V_{IN}(+)$ pin with the $V_{IN}(-)$ pin grounded. The value of the $V_{REF}/2$ input voltage should then be adjusted until the digital output code is just changing from 1111 1110 to 1111 1111. This value of $V_{REF}/2$ should then be used for all the tests.

The digital output LED display can be decoded by dividing the 8 bits into 2 hex characters, the 4 most significant (MS) and the 4 least significant (LS). Table I shows the fractional binary equivalent of these two 4-bit groups. By adding the decoded voltages which are obtained from the column: Input voltage value for a 2.560 $V_{REF}/2$ of both the MS and the LS groups, the value of the digital display can be determined. For example, for an output LED display of 1011 0110 or B6 (in hex), the voltage values from the table are 3.520 + 0.120 or 3.640 V_{DC}. These voltage values represent the center-values of a perfect A/D converter. The effects of quantization error have to be accounted for in the interpretation of the test results.

For a higher speed test system, or to obtain plotted data, a digital-to-analog converter is needed for the test set-up. An accurate 10-bit DAC can serve as the precision voltage source for the A/D. Errors of the A/D under test can be provided as either analog voltages or differences in 2 digital words.

A basic A/D tester which uses a DAC and provides the error as an analog output voltage is shown in *Figure 6*. The 2 op amps can be eliminated if a lab DVM with a numerical subtraction feature is available to directly readout the difference voltage, "A−C". The analog input voltage can be supplied by a low frequency ramp generator and an X-Y plotter can be used to provide analog error (Y axis) versus analog input (X axis). The construction details of a tester of this type are provided in the NSC application note AN-179, "Analog-to-Digital Converter Testing".

For operation with a microprocessor or a computer-based test system, it is more convenient to present the errors digitally. This can be done with the circuit of *Figure 7*, where the output code transitions can be detected as the 10-bit DAC is incremented. This provides 1/4 LSB steps for the 8-bit A/D under test. If the results of this test are automatically plotted with the analog input on the X axis and the error (in LSB's) as the Y axis, a useful transfer function of the A/D under test results. For acceptance testing, the plot is not necessary and the testing speed can be increased by establishing internal limits on the allowed error for each code.

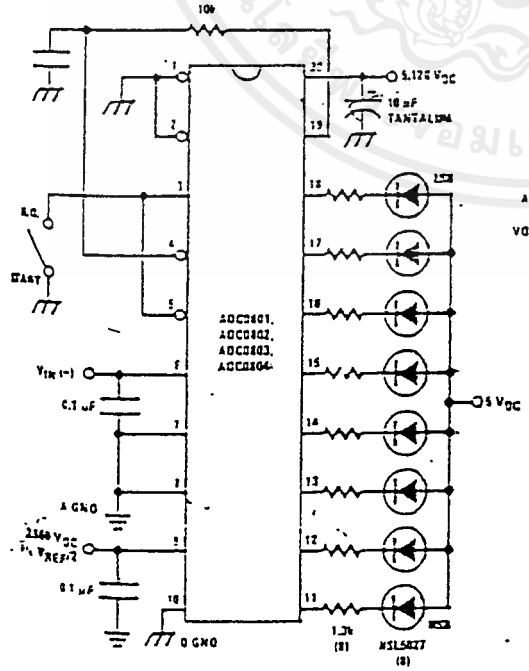


FIGURE 5. Basic A/D Tester

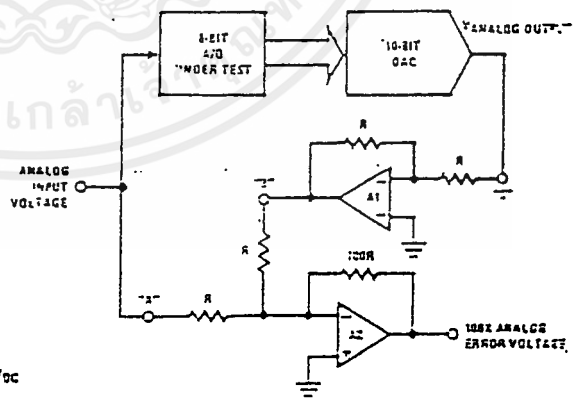


FIGURE 6. A/D Tester with Analog Error Output

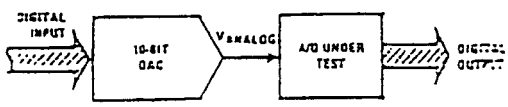


FIGURE 7. Basic "Digital" A/D Tester

2.3.4 Noise

The leads to the analog inputs (pins 6 and 7) should be kept as short as possible to minimize input noise coupling. Both noise and undesired digital clock coupling to these inputs can cause system errors. The source resistance for these inputs should, in general, be kept below 5 kΩ. Larger values of source resistance can cause undesired system noise pickup. Input bypass capacitors, placed from the analog inputs to ground, will eliminate system noise pickup but can create analog scale errors as these capacitors will average the transient input switching currents of the A/D (see section 2.3.3). This scale error depends on both a large source resistance and the use of an input bypass capacitor. This error can be eliminated by doing a full-scale adjustment of the A/D (adjust $V_{REF}/2$ for a proper full-scale reading—see section 2.5.2 on Full-Scale Adjustment) with the source resistance and input bypass capacitor in place.

2.4 Reference Voltage

2.4.1 Span Adjust

For maximum applications flexibility, these A/Ds have been designed to accommodate a 5 V_{DC}, 2.5 V_{DC} or an adjusted voltage reference. This has been achieved in the design of the IC as shown in *Figure 3*.

Notice that the reference voltage for the IC is either 1/2 of the voltage which is applied to the V_{CC} supply pin, or is equal to the voltage which is externally forced at the $V_{REF}/2$ pin. This allows for a ratiometric voltage reference using the V_{CC} supply, a 5 V_{DC} reference voltage can be used for the V_{CC} supply or a voltage less than 2.5 V_{DC} can be applied to the $V_{REF}/2$ input for increased application flexibility. The internal gain to the $V_{REF}/2$ input is 2 to allow this factor of 2 reduction in the $V_{REF}/2$ voltage.

An example of the use of an adjusted reference voltage is to accommodate a reduced span—or dynamic voltage range of the analog input voltage. If the analog input voltage were to range from 0.5 V_{DC} to 3.5 V_{DC} instead of 0V to 5 V_{DC}, the span would be 3V. With 0.5 V_{DC} applied to the V_{IN} (−) pin to absorb the offset, the reference voltage can be made equal to 1/2 of the 3V span or 1.5 V_{DC}. The A/D now will encode the V_{IN} (+) signal from 0.5V to 3.5V with the 0.5V input corresponding to zero and the 3.5 V_{DC} input corresponding to full-scale. The full 8 bits of resolution are therefore applied over this reduced analog input voltage range.

2.4.2 Reference Accuracy Requirements

The converter can be operated in a ratiometric mode or an absolute mode. In ratiometric converter applications, the magnitude of the reference voltage is a factor in both the output of the source transducer and the output of the A/D converter and therefore cancels out in the final digital output code. In absolute conversion applications, both the initial value and the temperature stability of the reference voltage are important accuracy factors in the operation of the A/D converter. For $V_{REF}/2$ voltages of 2.5 V_{DC} nominal value, initial errors of ±10 mV_{DC} will cause conversion errors of ±1 LSB due to the gain of 2 of the $V_{REF}/2$ input. In reduced span applications, the initial value and the stability of the $V_{REF}/2$ input voltage become even more important. For example, if the span is reduced to 2.5V, the analog input LSB voltage value is correspondingly reduced from 20 mV (5V span) to 10 mV and 1 LSB at the $V_{REF}/2$ input becomes 5 mV. As can be seen, this reduces the allowed initial tolerance of the reference voltage and requires correspondingly less absolute change with temperature variations. Note that spans smaller than 2.5V place even tighter requirements on the initial accuracy and stability of the reference source.

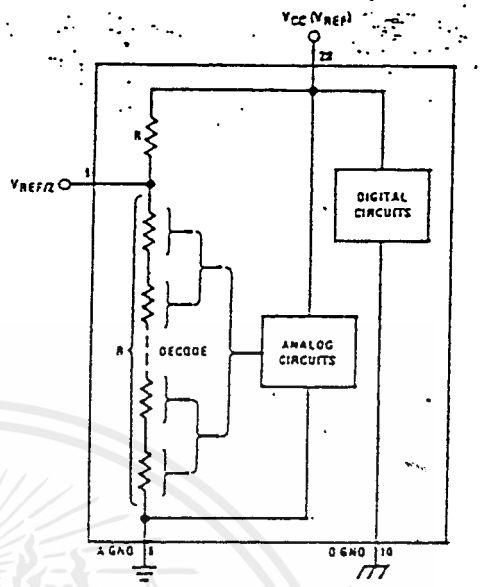


FIGURE 3. The $V_{REFERENCE}$ Design on the IC

In general, the magnitude of the reference voltage will require an initial adjustment. Errors due to an improper value of reference voltage appear as full-scale errors in the A/D transfer function. IC voltage regulators may be used for references if the ambient temperature changes are not excessive. The LM336B 2.5V IC reference diode (from National Semiconductor) is available which operates with a 5V input voltage and has a temperature stability of 1.8 mV typ (6 mV max) over $0^\circ C \leq T_A \leq +70^\circ C$. Other temperature range parts are also available.

2.5 Errors

2.5.1 Zero Error

The zero of the A/D does not require adjustment. If the minimum analog input voltage value, $V_{IN(MIN)}$, is not ground, a zero offset can be done. The converter can be made to output 0000 0000 digital code for this minimum input voltage by biasing the A/D V_{IN} (−) input at this $V_{IN(MIN)}$ value (see Applications section). This utilizes the differential mode operation of the A/D.

The zero error of the A/D converter relates to the location of the first riser of the transfer function and can be measured by grounding the V (−) input and applying a small magnitude positive voltage to the V (+)

TABLE I. DECODING THE DIGITAL OUTPUT LEDs

HEX	BINARY	FRACTIONAL BINARY VALUE FOR MS GROUP	FRACTIONAL BINARY VALUE FOR LS GROUP	OUTPUT VOLTAGE CENTER VALUES WITH $V_{REF}/2 = 2.560\ V_{DC}$ VMS GROUP*	OUTPUT VOLTAGE CENTER VALUES WITH $V_{REF}/2 = 2.560\ V_{DC}$ VLS GROUP
F	1 1 1 1	15/16	15/256	4.800	0.300
E	1 1 1 0	7/8	7/128	4.480	0.280
D	1 1 0 1	13/16	13/256	4.160	0.260
C	1 1 0 0	3/4	3/64	3.840	0.240
B	1 0 1 1	11/16	11/256	3.520	0.220
A	1 0 1 0	5/8	5/128	3.200	0.200
9	1 0 0 1	9/16	9/256	2.880	0.180
8	1 0 0 0	1/2	1/32	2.560	0.160
7	0 1 1 1	7/16	7/256	2.240	0.140
6	0 1 1 0	3/8	3/128	1.920	0.120
5	0 1 0 1	5/16	5/256	1.600	0.100
4	0 1 0 0	1/4	1/64	1.280	0.080
3	0 0 1 1	3/16	3/256	0.960	0.060
2	0 0 1 0	1/8	1/128	0.640	0.040
1	0 0 0 1	1/16	1/256	0.320	0.020
0	0 0 0 0			0	0

*V Display Output = VMS Group + VLS Group

4.0 MICROPROCESSOR INTERFACING

To discuss the interface with 8080A, 6800 and SC/MP-II microprocessors, a common sample subroutine structure is used. The microprocessor starts the A/D, reads and stores the results of 16 successive conversions, then returns to the user's program. The 16 data bytes are stored at location 0200 to 020F. All Data and Addresses will be given in hexadecimal form. Software and hardware details are provided separately for each type of microprocessor.

4.1 Interfacing 8080 Microprocessor Derivatives (8048, 8085)

This converter has been designed to directly interface with an 8080A-2 microprocessor (MICROBUS class 2). The A/D can be mapped into memory space (using standard memory address decoding for $\overline{CS}$ and the $\overline{MEMR}$ and $\overline{MEMW}$ strobes) or it can be controlled as an I/O device by using the $\overline{I/O\ R}$ and $\overline{I/O\ W}$ strobes and decoding the address bits A0 — A7 (or address bits A8 — A15 as they will contain the same 8-bit address information) to obtain the $\overline{CS}$ input. Using the I/O space provides 256 additional addresses and may allow a simpler 8-bit address decoder but the data can only be input to the accumulator. To make use of the additional memory reference instructions, the A/D should be mapped into memory space. An example of an A/D in I/O space is shown in *Figure 8*.

The standard control bus signals of the 8080 ($\overline{CS}$, $\overline{RD}$ and $\overline{WR}$) can be directly wired to the digital control inputs of the A/D and the bus timing requirements are met to allow both starting the converter and outputting the data onto the data bus. A bus driver should be used for larger microprocessor systems where the data bus leaves the PC board and/or must drive capacitive loads larger than 100 pF.

4.1.1 Sample 8080A CPU Interfacing Circuitry and Program

The following sample program and associated hardware may be used to input data from the converter to the INS8080A CPU chip set (comprised of the INS8080A microprocessor, the INS8228 system controller and the INS8224 clock generator). For simplicity, the A/D is controlled as an I/O device, specifically an 8-bit bi-directional port located at an arbitrarily chosen port address, E0. The TRI-STATE output capability of the A/D eliminates the need for a peripheral interface device, however address decoding is still required to generate the appropriate $\overline{CS}$ for the converter.

It is important to note that in systems where the A/D converter is 1-of-8 or less I/O mapped devices, no address decoding circuitry is necessary. Each of the 8 address bits (A0 to A7) can be directly used as $\overline{CS}$ inputs for each I/O device.

4.2 Interfacing the Z-80

The Z-80 control bus is slightly different from that of the 8080. General $\overline{RD}$ and $\overline{WR}$ strobes are provided and separate memory request, $\overline{MREQ}$, and I/O request, $\overline{IORQ}$, signals are used which have to be combined with the generalized strobes to provide the equivalent 8080 signals. An advantage of operating the A/D in I/O space with the Z-80 is that the CPU will automatically insert one wait state (the $\overline{RD}$ and $\overline{WR}$ strobes are extended one clock period) to allow more time for the I/O devices to respond. Logic to map the A/D in I/O space is shown in *Figure 9*.

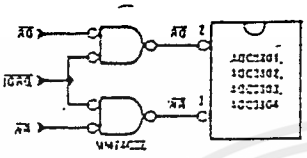


FIGURE 9. Mapping the A/D as an I/O device for use with the Z-80 CPU

Additional I/O advantages exist as software DMA routines are available and use can be made of the output data transfer which exists on the upper 8 address lines (A8 to A15) during I/O input instructions. For example, MUX channel selection for the A/D can be accomplished with this operating mode.

4.3 Interfacing 6800 Microprocessor Derivatives (6502, etc.)

The control bus for the 6800 microprocessor derivatives does not use the $\overline{RD}$ and $\overline{WR}$ strobe signals. Instead it employs a single R/$\overline{W}$ line and additional timing, if needed, can be derived from the $\phi 2$ clock. All I/O devices are memory mapped in the 6800 system, and a special signal, VMA, indicates that the current address is valid. *Figure 10* shows an interface schematic where the A/D is memory mapped in the 6800 system. For simplicity, the $\overline{CS}$ decoding is shown using 1/2 DM8092. Note that in many 6800 systems, an already decoded $\overline{A15}$ line is brought out to the common bus at pin 21. This can be tied directly to the $\overline{CS}$ pin of the A/D, provided that no other devices are addressed at HEX ADDR: 4XXX or 5XXX.

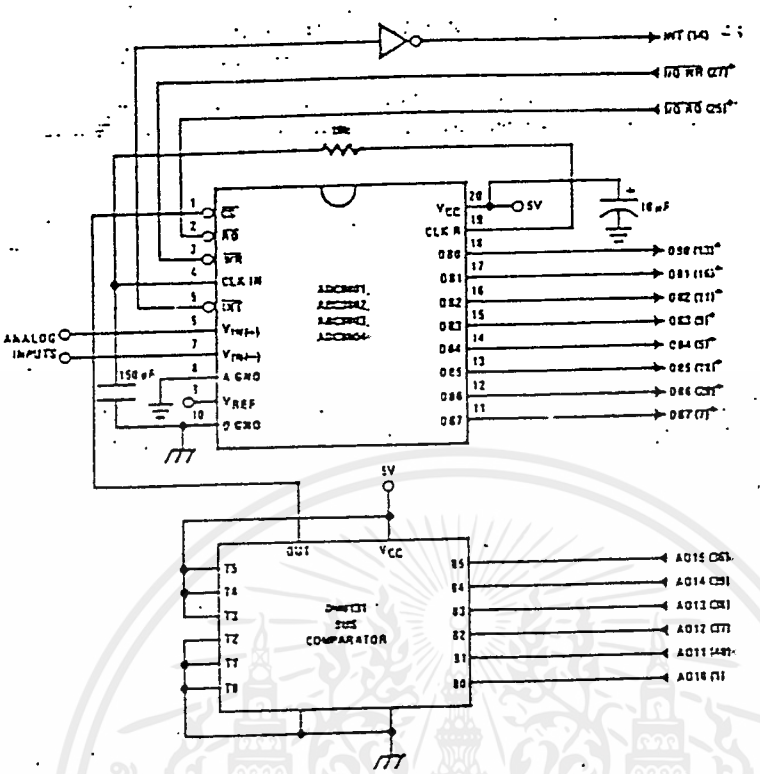
The following subroutine essentially performs the same function as in the case of the 8080A interface and it can be called from anywhere in the user's program.

In *Figure 11* the ADC0801 series is interfaced to the M6800 microprocessor through (the arbitrarily chosen) Port B of the MC6820 or MC6821 Peripheral Interface Adapter, (PIA). Here the $\overline{CS}$ pin of the A/D is grounded since the PIA is already memory mapped in the M6800 system and no $\overline{CS}$ decoding is necessary. Also notice that the A/D output data lines are connected to the microprocessor bus under program control through the PIA and therefore the A/D $\overline{RD}$ pin can be grounded.

SAMPLE PROGRAM FOR *FIGURE 10* ADC0801—MC6800 CPU INTERFACE

```
0010  0F 36     DATAIN  STX   TEMP2    ; Save contents of X
0012  CE 00 2C           LDX   #$002C   ; Upon IRQ low CPU
0015  FF FF F8           STX   $FFF8    ; jumps to 002C
0018  B7 50 00           STAA  $5000    ; Starts ADC0801
0018  0E                 CLI
001C  3E        CONVRT   WAI            ; Wait for interrupt
001D  DE 34              LDX   TEMP1
001F  8C 02 0F           CPX   #$020F   ; Is final data stored?
0022  27 14              BEQ   ENDP
0024  B7 50 00           STAA  $5000    ; Restarts ADC0801
0027  08                 INX
0028  DF 34              STX   TEMP1
002A  20 F0              BRA   CONVRT
002C  DE 34     INTRPT   LDX   TEMP1
002E  B6 50 00           LDAA  $5000    ; Read data
0031  A7 00              STAA  X        ; Store it at X
0033  3B                 RTI
0034  02 00     TEMP1   FDB   $0200     ; Starting address for
                                        ; data storage
0036  00 00     TEMP2   FDB   $0000
0038  CE 02 00  ENDP    LDX   #$0200   ; Reinitialize TEMP1
0038  DF 34              STX   TEMP1
003D  DE 36              LDX   TEMP2
003F  39                 RTS            ; Return from subroutine
                                        ; To user's program
```

Note 1: In order for the microprocessor to service subroutines and interrupts, the stack pointer must be dimensioned in the user's program.



Note 1: *Pin numbers for the INS8228 system controller, others are INS8080A.*

Note 2: *Pin 23 of the INS8228 must be tied to +12V through a 1 kΩ resistor to generate the RST 7 instruction when an interrupt is acknowledged as required by the accompanying sample program.*

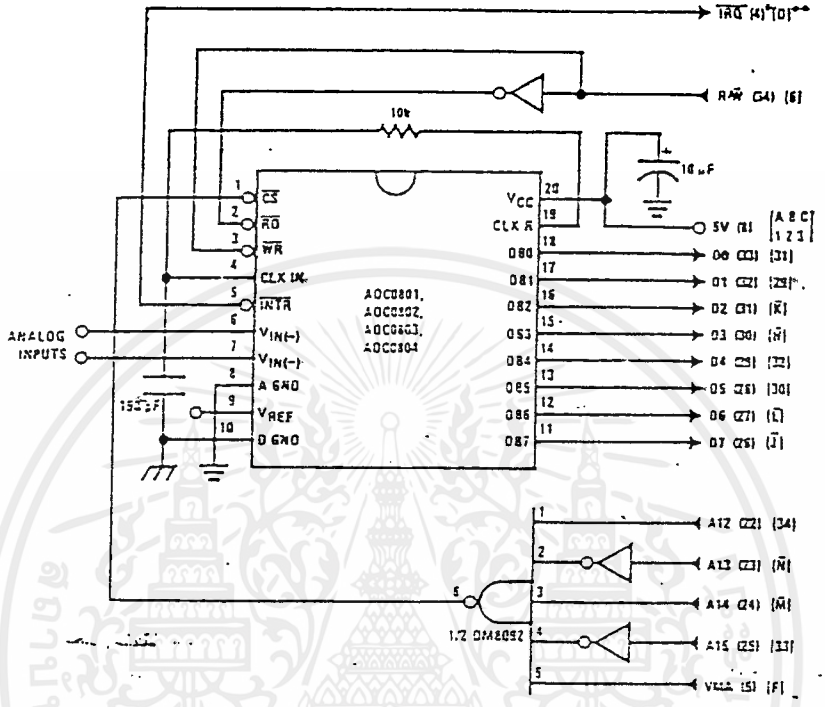
FIGURE 8. ADC0801—INS8080A CPU Interface

SAMPLE PROGRAM FOR *FIGURE 3* ADC0801—INS8080A CPU INTERFACE

```
0038    C3 00 03    RST 7:      JMP     LD DATA
  .         .           .
  .         .           .
0100    21 00 02    START:      LXI H 0200H     ; HL pair will point to
                                                ; data storage locations
0103    31 00 04    RETURN:     LXI SP 0400H    ; Initialize stack pointer (Note 1)
0106    7D                      MOV A, L        ; Test = of bytes entered
0107    FE 0F                   CPI 0F H        ; If = = 15, JMP to
0109    CA 13 01                JZ CONT         ; user program
010C    D3 E0                   OUT E0 H        ; Start A/D
010E    FB                      EI              ; Enable interrupt
010F    00          LOOP:       NOP             ; Loop until end of
0110    C3 0F 01                JMP LOOP        ; conversion
0113      .         CONT:         .
  .       .                       .
  .       .         (User program to .
  .       .         process data)   .
  .       .           .             .
  .       .           .             .
0300    DB E0       LD DATA:    IN E0 H         ; Load data into accumulator
0302    77                      MOV M, A        ; Store data
0303    23                      INX H           ; Increment storage pointer
0304    C3 03 01                JMP  RETURN
```

Note 1: The stack pointer must be dimensioned because a RST 7 instruction pushes the PC onto the stack.

Note 2: All addresses used were arbitrarily chosen.



Note 1: Numbers in parentheses refer to MC6800 CPU pin out.

Note 2: Numbers or letters in brackets refer to standard M6800 system common bus code.

FIGURE 10. ADC0801 – MC6800 CPU Interface

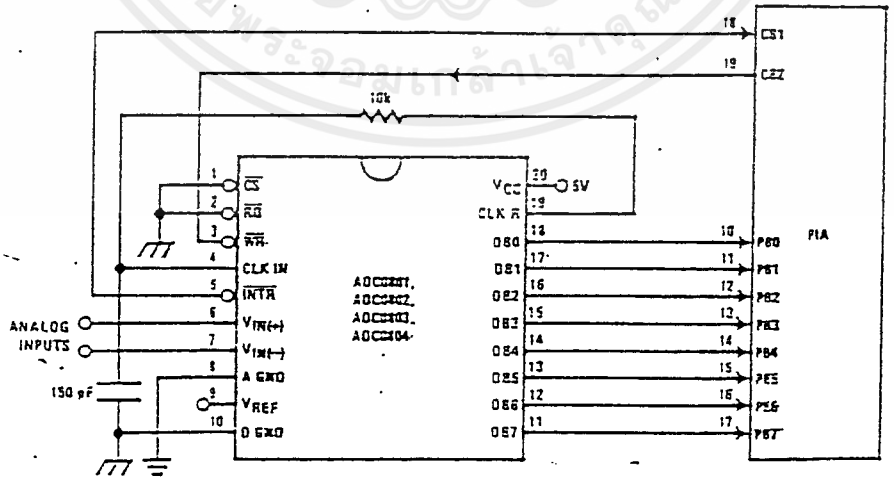
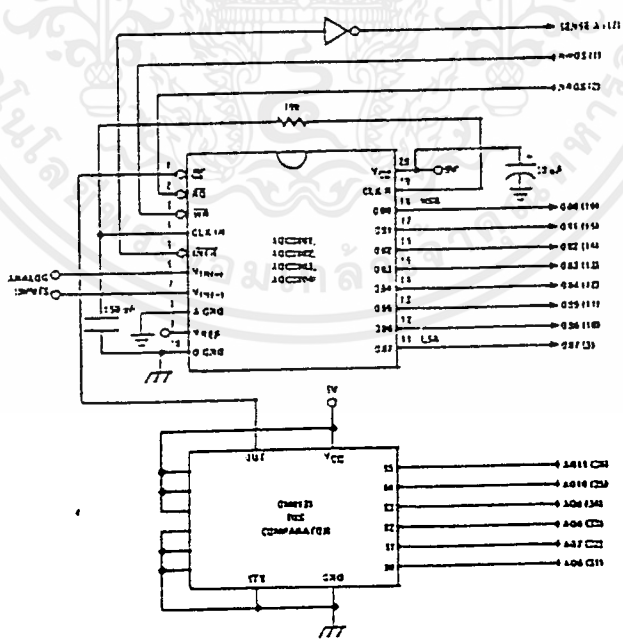


FIGURE 11. ADC0801—MC6820 PIA Interface

A sample interface program equivalent to the previous one, is shown below. The PIA Data and Control Registers of Port B are located at HEX addresses 8006 and 8007, respectively.

SAMPLE PROGRAM FOR *FIGURE 11* ADC0801—MC6820 PIA INTERFACE

```
0010  CE 00 38   DATAIN   LDX   =0038    ; Upon IRQ low CPU
0013  FF FF F8           STX   SFFF8     ; jumps to 0038
0016  B6 80 06           LDAA  PIACRB    ; Clear possible IRQ flags
0019  4F                 CLRA
001A  B7 80 07           STAA  PIACRB
001D  B7 80 06           STAA  PIAORB    ; Set Port B as input
0020  0E                 CLI
0021  C6 34              LDAB  #$34
0023  86 30              LDAA  #$30
0025  F7 80 07   CONVRT  STAB  PIACRB    ; Starts ADC0801
0028  B7 80 07           STAA  PIACRB
002B  3E                 WAI             ; Wait for interrupt
002C  DE 40              LDX   TEMP1
002E  8C 02 0F           CPX   #$020F    ; Is final data stored?
0031  27 0F              BEQ   ENDP
0033  08                 INX
0034  DF 40              STX   TEMP1
0036  20 E9              BRA   CONVRT
0038  DE 40      INTRPT  LDX   TEMP1
003A  B6 80 06           LDAA  PIAORB    ; Read data in
003D  A7 00              STAA  X         ; Store it at X
003F  3B                 RTI
0040  02 00      TEMP1   FDB   $0200     ; Starting address for
                                         ; data storage
0042  CE 02 00   ENDP    LDX   #$0200    ; Reinitialize TEMP1
0045  DF 40              STX   TEMP1
0047  39                 RTS             ; Return from worduame
                 PIAORB  EQU   $8006     ; To user's program
                 PIACRB  EQU   $8007
```



*Pin numbers in parentheses are for the SC/MP CPU.

FIGURE 12. ADC0801 — SC/MP-II Microprocessor Interface

4.4 Interfacing the INS8060—SC/MP-II

The SC/MP-II interface technique with the ADC0801 series *Figure 12*, is similar to the 8080A CPU interface:

The A/D is treated as a peripheral and it is mapped into the memory space of the SC/MP-II system. An address, 0D00, is assigned to the A/D and the $\overline{CS}$ signal is shown to be decoded by a bus comparator, DM8131. The $\overline{RD}$ and $\overline{WR}$ pins of the A/D are tied directly to the Write Data Strobe, NWDS, and Read Data Strobe, NRDS, pins of the SC/MP-II CPU. Notice that the $\overline{INTR}$ signal should be inverted before being tied to the SENSE A pin of the SC/MP-II. A sample interface program is shown below.

5.0 GENERAL APPLICATIONS

The following applications show some interesting uses for the A/D. The fact that one particular microprocessor is used is not meant to be restrictive. Each of these application circuits would have its counterpart using any microprocessor which is desired.

5.1 Multiple ADC0801 Series to MC6800 CPU Interface

To transfer analog data from several channels to a single microprocessor system, a multiple converter scheme presents several advantages over the conventional multiplexer single-converter approach. With the ADC0801 series, the differential inputs allow individual span adjustment for each channel. Furthermore, all analog input channels are sensed simultaneously, which essentially divides the microprocessor's total system servicing time by the number of channels, since all conversions occur simultaneously. This scheme is shown in *Figure 12*.

SAMPLE PROGRAM FOR *FIGURE 12* ADC0801—SC/MP-II MICROPROCESSOR INTERFACE

```
0100        08          NOP
0101     C4 02          LDI02
0103        35          XPAH(P1)
0104     C4 0D          LDI0D
0106        36          XPAH(P2)
0107     C4 03          LDI03
0109        37          XPAH(P3)
010A     C4 00          LDI00
010C        31          XPAL(P1)      ; P1=0200, P1 points to 1st byte address
010D     C4 00          LDI00
010F     C9 11          ST(P1+11)     ; Zero the byte count in address 0211
0112        32          XPAL(P2)      ; P2=0D00, P2 points to A/D
0113     CA 00  START:  ST(P2)        ; START the A/D
0115     C4 00          LDI00
0117        33          XPAL(P3)      ; P3=0300, P3 points to DATA in sub.
0118        05          IEN           ; starting address
0119        08  LOOP:   NOP
011A     90 FE          JMP(LOOP)
                User's Program
011C           USER    NOP
011D                   NOP

0300     C2 00  DATA IN: LD(P2)       ; Load A/D data into accumulator
0302     CD 01          ST@1(P1)      ; Store A/D data and increment byte
                                      ; address
0304     A9 11          ILD(P1+11)    ; Increment byte count
0306     C4 0F          LDI0F
0308        03          SCL
0309     F9 11          CAD(P1+11)    ; 0F—(P1+11): Is byte count = 16?
030B     98 03          JZ(USER)      ; If byte count = 16 jump to user's
                                      ; program
030D     C4 13          LDI13
030F        33          XPAL(P3)      ; P3=0113
0310        3F          XPPC(P3)      ; Go to START and do another conversion
```

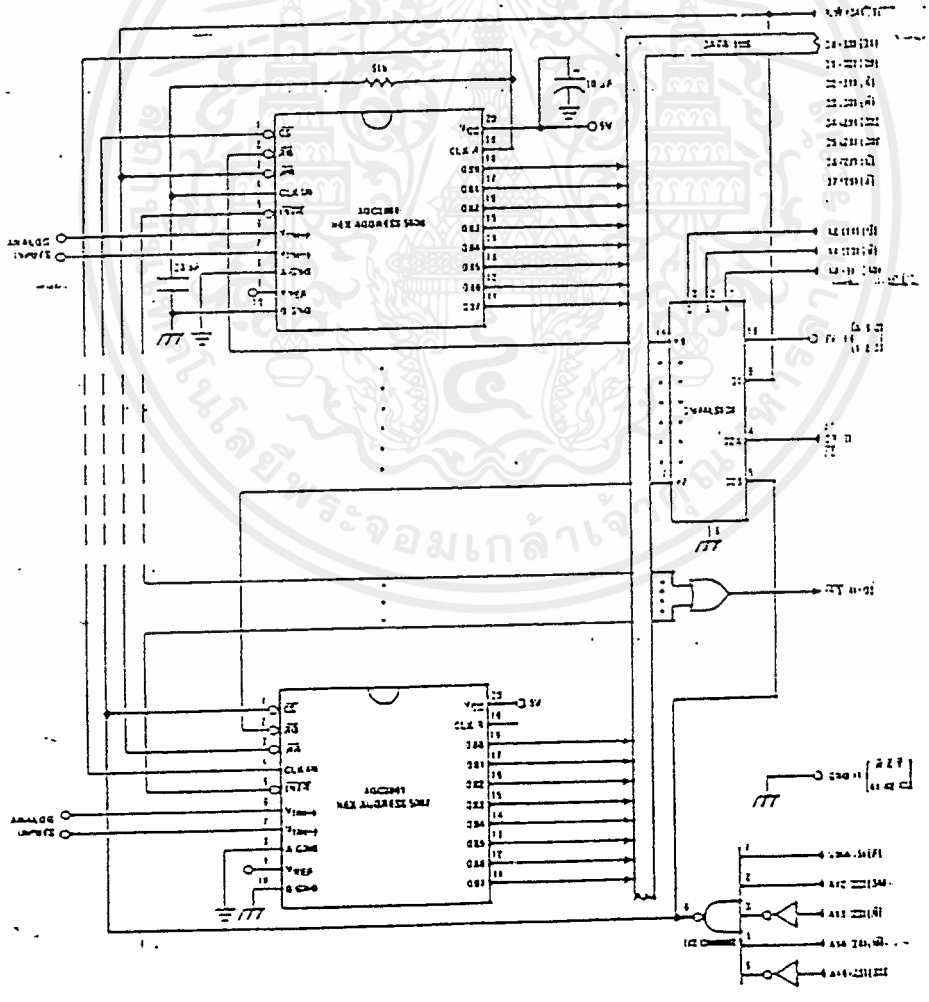
The following schematic and sample subroutine (DATA IN) may be used to interface (up to) 8 ADC0801's directly to the MC6800 CPU. This scheme can easily be extended to allow the interface of more converters. In this configuration the converters are (arbitrarily) located at HEX address 5000 in the MC6800 memory space. To save components, the clock signal is derived from just one RC pair on the first converter. This output drives the other A/Ds.

All the converters are started simultaneously with a STORE instruction at HEX address 5000. Note that any other HEX address of the form 5XXX will be decoded by the circuit, pulling all the $\overline{CS}$ inputs low. This can easily be avoided by using a more definitive address decoding scheme. All the interrupts are ORed together to insure that all A/Ds have completed their conversion before the microprocessor is interrupted.

The subroutine, DATA IN, may be called from anywhere in the user's program. Once called, this routine initializes the CPU, starts all the converters simultaneously and waits for the interrupt signal. Upon receiving the interrupt, it reads the converters (from HEX addresses 5000 through 5007) and stores the data successively at (arbitrarily chosen) HEX addresses 0200 to 0207, before returning to the user's program. All CPU registers then recover the original data they had before servicing DATA IN.

5.2 Auto-Zeroed Differential Transducer Amplifier and A/D Converter

The differential inputs of the ADC0801 series eliminate the need to perform a differential to single ended conversion for a differential transducer. Thus, one op. amp can be eliminated since the differential to single ended conversion is provided by the differential input of the ADC0801 series. In general, a transducer preamp is required to take advantage of the full A/D converter input dynamic range.



Note 1: Numbers in parentheses refer to MC6800 CPU pin out.
Note 2: Numbers or letters in brackets refer to standard M6800 system common bus code.

FIGURE 13. Interfacing Multiple A/Ds in a MC6800 System

PROGRAM FOR *FIGURE 13* INTERFACING MULTIPLE A/Ds IN AN MC6800 SYSTEM

ADDRESS	HEX CODE		MNEMONICS		COMMENTS
0010	DF 44	DATAIN	STX	TEMP	;Save Contents of X
0012	CE 00 2A		LDX	#$002A	;Upon $\overline{IRQ}$ LOW CPU
0015	FF FF F8		STX	$FFF8	;Jumps to 002A
0018	B7 50 00		STAA	$5000	;Starts all A/D's
001B	0E		CLI		
001C	3E		WAI		;Wait for interrupt
001D	CE 50 00		LDX	#$5000	
0020	DF 40		STX	INDEX1	;Reset both INDEX
0022	CE 02 00		LDX	#$0200	;1 and 2 to starting
0025	DF 42		STX	INDEX2	;addresses
0027	DE 44		LDX	TEMP	
0029	39		RTS		;Return from subroutine
002A	DE 40	INTRPT	LDX	INDEX1	;INDEX1 → X
002C	A6 00		LDAA	X	;Read data in from A-D at X
002E	08		INX		;Increment X by one
002F	DF 40		STX	INDEX1	;X → INDEX1
0031	DE 42		LDX	INDEX2	;INDEX2 → X
0033	A7 00		STAA	X	;Store data at X
0035	8C 02 07		CPX	#$0207	;Have all A/D's been read?
0038	27 05		BEQ	RETURN	;Yes: branch to RETURN
003A	08		INX		;No: increment X by one
003B	DF 42		STX	INDEX2	;X → INDEX2
003D	20 E8		BRA	INTRPT	;Branch to 002A
003F	3B	RETURN	RTI		
0040	50 00	INDEX1	FDB	$5000	;Starting address for A-D
0042	02 00	INDEX2	FDB	$0200	;Starting address for data storage
0044	00 00	TEMP	FDB	$0000	

Note 1: In order for the microprocessor to service subroutines and interrupts, the stack pointer must be dimensioned in the user's program.

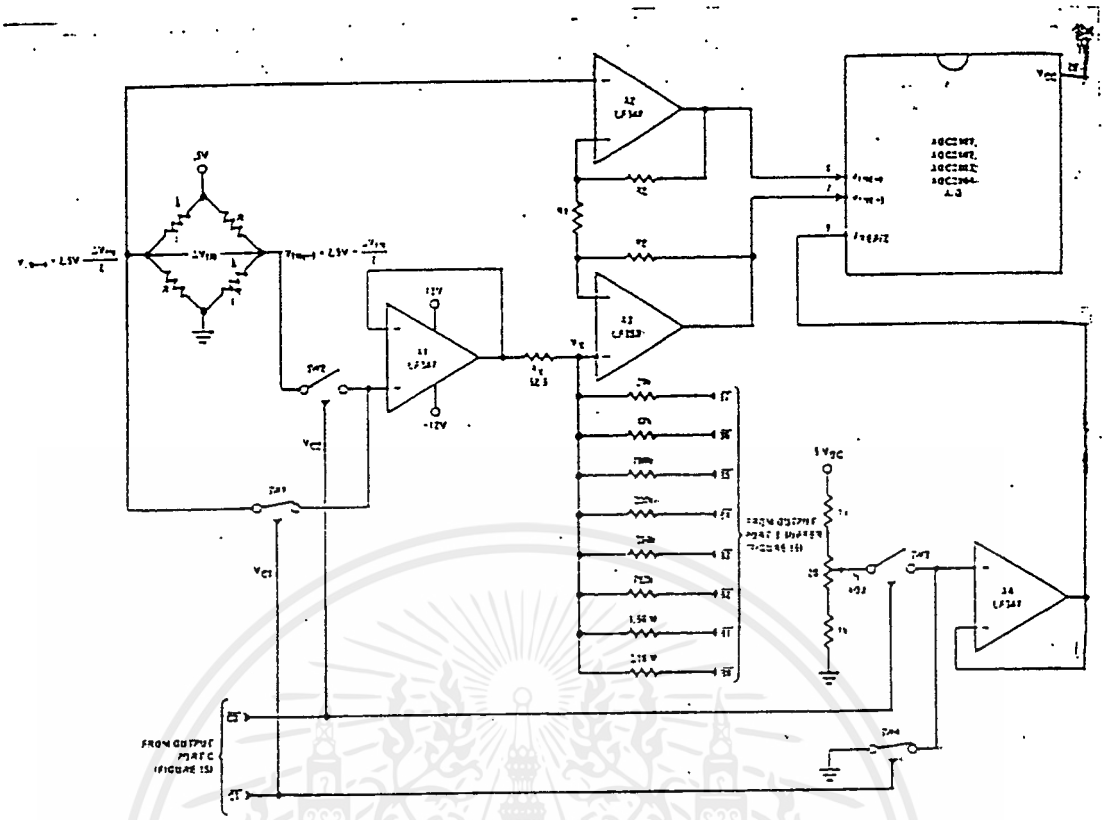
For amplification of DC input signals, a major system error is the input offset voltage of the amplifiers used for the preamp. *Figure 14* is a gain of 100 differential preamp whose offset voltage errors will be cancelled by a zeroing subroutine which is performed by the INS8080A microprocessor system. The total allowable input offset voltage error for this preamp is only 50 μV for 1/4 LSB error. This would obviously require very precise amplifiers. The expression for the differential output voltage of the preamp is:

$$V_o = \underbrace{[V_{IN}(+) - V_{IN}(-)]}_{\text{SIGNAL}} \underbrace{\left[1 + \frac{2R2}{R1}\right]}_{\text{GAIN}} + \underbrace{(V_{os2} - V_{os1} - V_{os3} = I_x R_x)}_{\text{DC ERROR TERM}} \underbrace{\left(1 + \frac{2R2}{R1}\right)}_{\text{GAIN}}$$

where I_x is the current through resistor R_x. All of the offset error terms can be cancelled by making $\pm I_x R_x = V_{os1} + V_{os3} - V_{os2}$. This is the principle of this auto-zeroing scheme.

The INS8080A uses the 3 I/O ports of an INS8255 Programmable Peripheral Interface (PPI) to control the auto zeroing and input data from the ADC0801 as shown in *Figure 15*. The PPI is programmed for basic I/O operation (mode 0) with Port A being an input port and Ports B and C being output ports. Two bits of Port C are used to alternately open or close the 2 switches at the input of the preamp. Switch SW1 is closed to force the preamp's differential input to be zero during the zeroing subroutine and then opened and SW2 is then closed for conversion of the actual differential input signal. Using 2 switches in this manner eliminates concern for the ON resistance of the switches as they must conduct only the input bias current of the input amplifiers.

Output Port B is used as a successive approximation register by the 8080 and the binary scaled resistors in series with each output bit create a D/A converter. During the zeroing subroutine, the voltage at V_x increases or decreases as required to make the differential output voltage equal to zero. This is accomplished by insuring that the voltage at the output of A1 is approximately 2.5V so that a logic "1" (5V) on any output of Port B will source current into node V_x thus raising the voltage at V_x and making the output differential more negative. Conversely, a logic "0" (0V) will pull current out of node V_x and decrease the voltage, causing the differential output to become more positive. For the resistor values shown, V_x can move ±12 mV with a resolution of 50 μV which will null the offset error term to 1/4 LSB of full-scale for the ADC0801. It is important that the voltage levels which drive the auto-zero resistors be constant. Also, for symmetry, a logic swing of 0V to 5V is convenient. To achieve this, a CMOS buffer is used for the logic output signals of Port B and this CMOS package is powered with a stable 5V source. Buffer amplifier A1 is necessary so that it can source or sink the D/A output current.



Note 1: R2 = 49.5 R1

Note 2: Switches are CD4066BC CMOS analog switches.

Note 3: The 9 resistors used in the auto-zero section can be ±5% tolerance.

FIGURE 14. Gain of 100 Differential Transducer Preamp

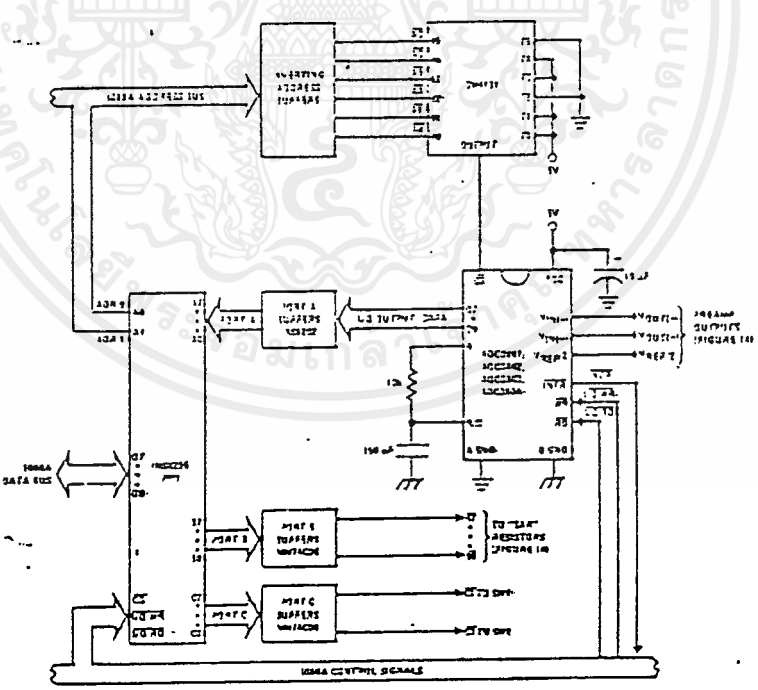


FIGURE 15. Microprocessor Interface Circuitry for Differential Preamp

A flow chart for the zeroing subroutine is shown in *Figure 16*. It must be noted that the ADC0801 series will output an all zero code when it converts a negative input ($V_{IN}(-) \geq V_{IN}(+)$). Also, a logic inversion exists as all of the I/O ports are buffered with inverting gates.

Basically, if the data read is zero, the differential output voltage is negative, so a bit in Port B is cleared to pull V_X more negative which will make the output more positive for the next conversion. If the data read is not zero, the output voltage is positive so a bit in Port B is set to make V_X more positive and the output more negative. This continues for 8 approximations and the differential output eventually converges to within 5 mV of zero.

The actual program is given in *Figure 17*. All addresses used are compatible with the SLC 80/10 microcomputer system. In particular:

Port A and the ADC0801 are at port address E4
Port B is at port address E5
Port C is at port address E6
PPI control word port is at port address E7
Program Counter automatically goes to ADDR:3C30 upon acknowledgement of an interrupt from the ADC0801

5-3 Multiple A/D Converters in a Z-80 Interrupt Driven Mode

In data acquisition systems where more than one A/D converter (or other peripheral device) will be interrupting program execution of a microprocessor, there is obviously a need for the CPU to determine which device requires servicing. *Figure 18* and the accompanying software is a method of determining which of 7 ADC0801 converters has completed a conversion ($\overline{INTR}$ asserted) and is requesting an interrupt. This circuit allows starting the A/D converters in any sequence, but will input and store valid data from the converters with a priority sequence of A/D 1 being read first, A/D 2 second, etc., through A/D 7 which would have the lowest priority for data being read. Only the converters whose INT is asserted will be read.

The key to decoding circuitry is the MM74LS373, 8-bit D type flip-flop. When the Z-80 acknowledges the interrupt, the program is vectored to a data input Z-80 subroutine. This subroutine will read a peripheral status word from the MM74LS373 which contains the logic state of the $\overline{INTR}$ outputs of all the converters. Each converter which initiates an interrupt will place a logic "0" in a unique bit position in the status word and the subroutine will determine the identity of the converter and execute a data read. An identifier word (which indicates which A/D the data came from) is stored in the next sequential memory location above the location of the data so the program can keep track of the identity of the data entered.

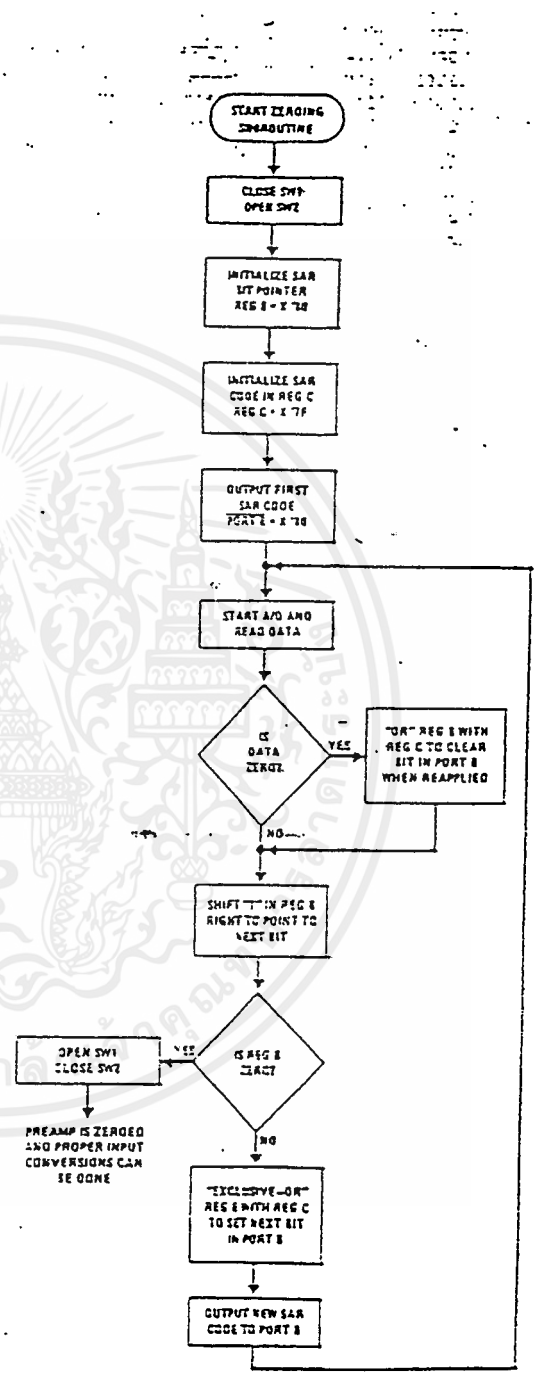


FIGURE 16. Flow Chart for Auto-Zero Routine

```
3000   3E90     MVI 90
3002   03E7     Out Control Port          ; Program PPI
3004   2601     MVI H 01      Auto-Zero Subroutine
3006   7C       MOV A,H
3007   03E6     OUT C                     ; Close SW1, open SW2
3009   0680     MVI B 80                  ; Initialize SAR bit pointer
3008   3E7F     MVI A 7F                  ; Initialize SAR code
300D   4F       MOV C,A       Return
300E   03E5     OUT B                     ; Port B = SAR code
3010   31AA30   LXI SP 30AA  Start:      ; Dimension stack pointer
3013   03E4     OUT A                     ; Start A-D
3015   FB       IE
3016   00       NOP           Loop        ; Loop until INT asserted
3017   C31630   JMP Loop
301A   7A       MOV A,D       Auto-Zero
3018   C600     ADI 00                    ; Test A-D output data for zero
3010   CA2D30   JZ Set C
3020   78       MOV A,B       Shift B
3021   F600     ORI 00                    ; Clear carry
3023   1F       RAR                       ; Shift "1" in B right one place
3024   FE00     CPI 00                    ; Is B zero? If yes last
3026   CA3730   JZ Done                   ; approximation has been made
3029   47       MOV B,A
302A   C33330   JMP New C
302D   79       MOV A,C       Set C
302E   80       ORA B                     ; Set bit in C that is in same
302F   4F       MOV C,A                   ; position as "1" in B
3030   C32030   JMP Shift B
3033   A9       XRA C         New C       ; Clear bit in C that is in
3034   C30030   JMP Return                ; same position as "1" in B
3037   47       MOV B,A       Done        ; then output new SAR code.
3038   7C       MOV A,H                   ; Open SW1, close SW2 then
3039   EE03     XRI 03                    ; proceed with program. Preamp
303B   03E6     OUT C                     ; is now zeroed.
303D   --       --            Normal

                 Program for processing
                 proper data values
3C30   03E4     IN A          Read A-D Subroutine  ; Read A-D data
3C3F   EEFF     XRI FF                    ; Invert data
3C41   57       MOV D,A
3C42   78       MOV A,B                   ; Is B Reg = 0? If not stay
3C43   E6FF     ANI FF                    ; in auto zero subroutine
3C45   C21A30   JNZ Auto-Zero
3C48   C33D30   JMP Normal
```

Note: All numerical values are hexadecimal representations.

FIGURE 17. Software for Auto-Zeroed Differential A/D

5-3 Multiple A/D Converters in a Z-80 Interrupt Driven Mode (Continued)

The following notes apply:

1) It is assumed that the CPU automatically performs a RST 7 instruction when a valid interrupt is acknowledged (CPU is in interrupt mode 1). Hence, the subroutine starting address of X0038.

2) The address bus from the Z-80 and the data bus to the Z-80 are assumed to be inverted by bus drivers.

3) A/D data and identifying words will be stored in sequential memory locations starting at the arbitrarily chosen address X 3E00.

4) The stack pointer must be dimensioned in the main program as the RST 7 instruction automatically pushes the PC onto the stack and the subroutine uses an additional 6 stack addresses.

5) The peripherals of concern are mapped into I/O space with the following port assignment:

HEX PORT ADDRESS	PERIPHERAL
00	MM74C374 8-bit flip-flop
01	A/D 1
02	A/D 2
03	A/D 3
04	A/D 4
05	A/D 5
06	A/D 6
07	A/D 7

This port address also serves as the A/D identifying word in the program.

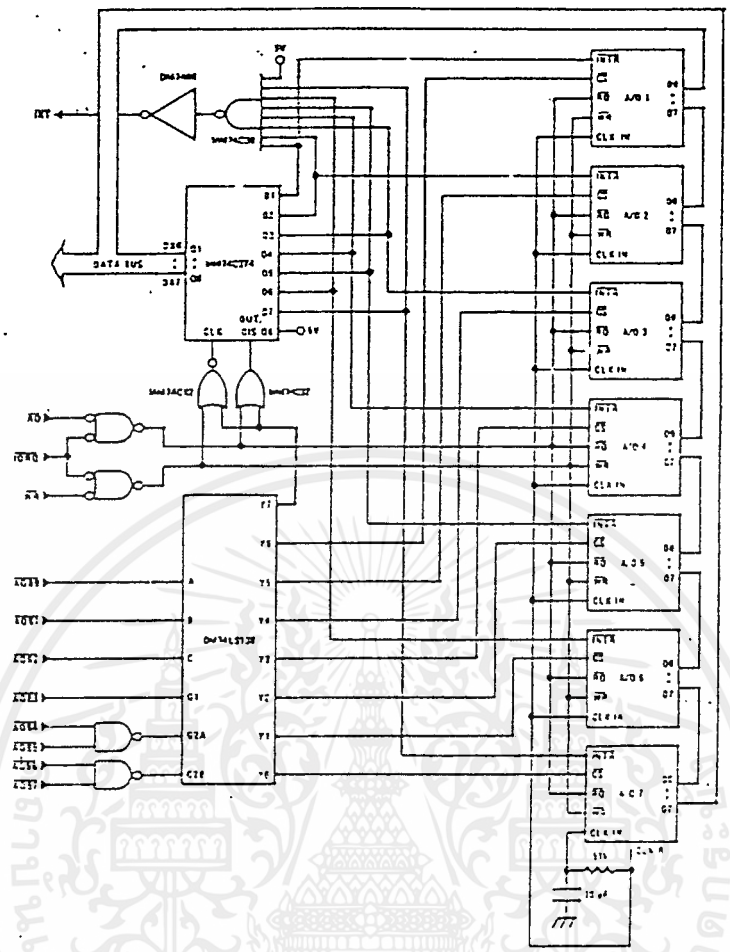


FIGURE 18. Multiple A/D's with Z-80 Type Microprocessor

INTERRUPT SERVICING SUBROUTINE

LOC	OBJ CODE	SOURCE STATEMENT	COMMENT
0038	E5	PUSH HL	; Save contents of all registers affected by
0039	C5	PUSH BC	; the subroutine.
003A	F5	PUSH AF	; Assumed INT mode 1 earlier set.
003B	21 00 3E	LD IHL X3E00	; Initialize memory pointer where data will be stored.
003E	0E 01	LD C,ACT	; C register will be port ADDR of A/D converters.
0040	D300	OUT X00,A	; Load peripheral status word into 8 bit latch
0042	DB00	IN A, X00	; Load status word into accumulator
0044	47	LD B,A	Save the status word.
0045	78	TEST LD A,B	; Test to see if the status of the A/D's have
0046	FE 00	CP, X00	; been checked. If so, exit subroutine.
0048	CA 60 00	JPZ, DONE	
0048	78	LD A,B	; Test a single bit in status word by looking for
004C	1F	RRA	; a "1" to be rotated into the CARRY (an INT
004D	47	LD B,A	; is looked as a "1"). If CARRY is set then load
004E	DA 5500	JPC, LOAD	; contents of A/D at port ADDR in C register.
0051	0C	NEXT INC C	; If CARRY is not set, increment C register to point
0052	C3 4500	JP,TEST	; to next A/D, then test next bit in status word.
0055	ED 78	LOAD IN A, IC	; Read data from interrupting A/D and invert
0057	EE FF	XOR FF	; the data.
0059	77	LD IHL),A	; Store the data.
005A	2C	INC L	
005B	71	LD IHL),C	; Store A/D identifier (A/D port ADDR)
005C	2C	INC L	
005D	C3 51 00	JP,NEXT	; Test next bit in status word.
0060	F1	DONE POP AF	; Reestablish all registers as they were
0061	C1	POP BC	; before the interrupt.
0062	E1	POP HL	
0063	C9	RET	; Return to original program

DALLAS SEMICONDUCTOR

DS1202, DS1202S
Serial Timekeeping Chip

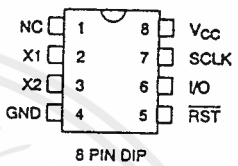
FEATURES

- Real time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap year compensation
- 24 x 8 RAM for scratchpad data storage
- Serial I/O for minimum pin count
- 2.0-5.5 volt full operation
- Uses less than 300 nA at 2 volts
- Single-byte or multiple-byte (burst mode) data transfer for read or write of clock or RAM data
- 8-pin DIP or optional 16-pin SOIC for surface mount
- Simple 3-wire interface
- TTL-compatible (V_{CC} = 5V)
- Optional industrial temperature range -40°C to +85°C

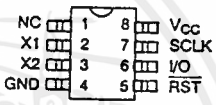
ORDERING INFORMATION

DS1202	8–pin DIP
DS1202S	16–pin SOIC
DS1202S8	8–pin SOIC

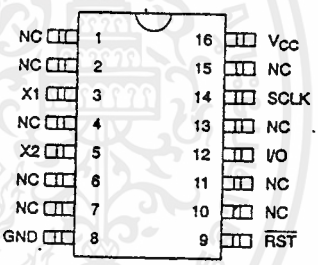
PIN ASSIGNMENT



8 PIN DIP



8–PIN SOIC
(208 mil)



16–PIN SOIC

PIN DESCRIPTION

NC	– No Connection
X1, X2	– 32.768 KHz Crystal Input
GND	– Ground
$\overline{RST}$	– Reset
I/O	– Data Input/Output
SCLK	– Serial Clock
V_{CC}	– Power Supply Pin

DESCRIPTION

The DS1202 Serial Timekeeping Chip contains a real time clock/calendar and 24 bytes of static RAM. It communicates with a microprocessor via a simple serial interface. The real time clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. Interfacing the DS1202 with a microprocessor is simplified by using synchronous serial communication. Only three wires are required to communicate with the clock/RAM: (1) $\overline{RST}$ (Reset), (2) I/O (Data line), and (3) SCLK (Serial clock). Data can be transferred to and from the clock/

RAM one byte at a time or in a burst of up to 24 bytes. The DS1202 is designed to operate on very low power and retain data and clock information on less than 1 microwatt.

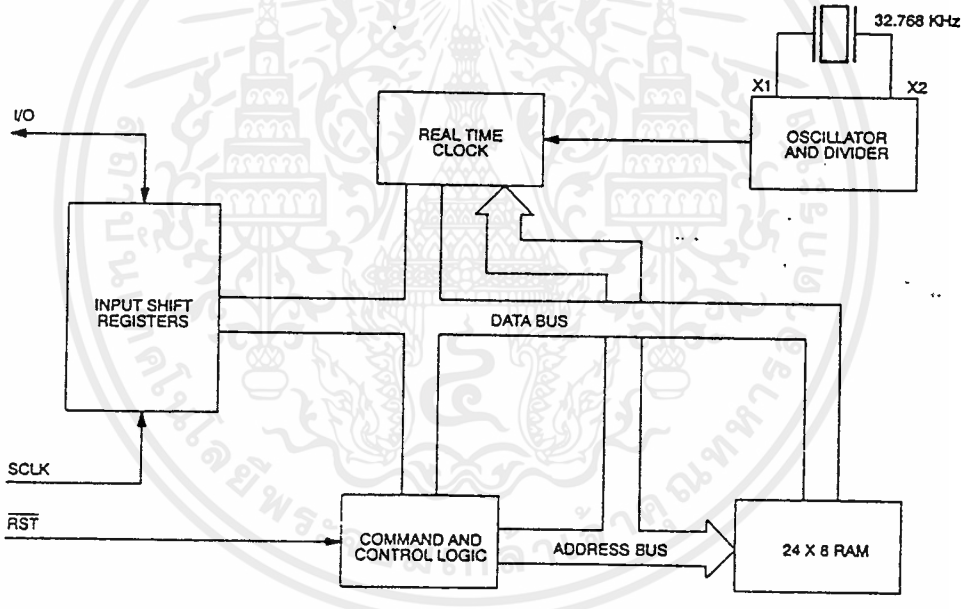
OPERATION

The main elements of the Serial Timekeeper are shown in Figure 1: shift register, control logic, oscillator, real time clock, and RAM. To initiate any transfer of data, $\overline{RST}$ is taken high and eight bits are loaded into the shift register providing both address and command information. Data is serially input on the rising edge of the SCLK. The first eight bits specify which of 32 bytes will be accessed, whether a read or write cycle will take place, and whether a byte or burst mode transfer is to occur. After the first eight clock cycles have occurred which load the command word into the shift register, additional clocks will output data for a read or input data for a write. The number of clock pulses equals eight plus eight for byte mode or eight plus up to 192 for burst mode.

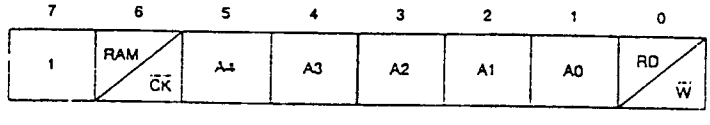
COMMAND BYTE

The command byte is shown in Figure 2. Each data transfer is initiated by a command byte. The MSB (Bit 7) must be a logic 1. If it is zero, further action will be terminated. Bit 6 specifies clock/calendar data if logic 0 or RAM data if logic 1. Bits one through five specify the designated registers to be input or output, and the LSB (Bit 0) specifies a write operation (input) if logic 0 or read operation (output) if logic 1. The command byte is always input starting with the LSB (bit 0).

DS1202 BLOCK DIAGRAM Figure 1



ADDRESS/COMMAND BYTE Figure 2



REGISTER SUMMARY

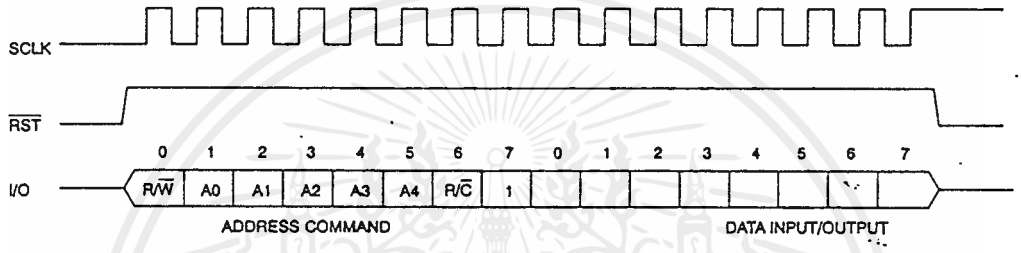
A register data format summary is shown in Figure 4.

CRYSTAL SELECTION

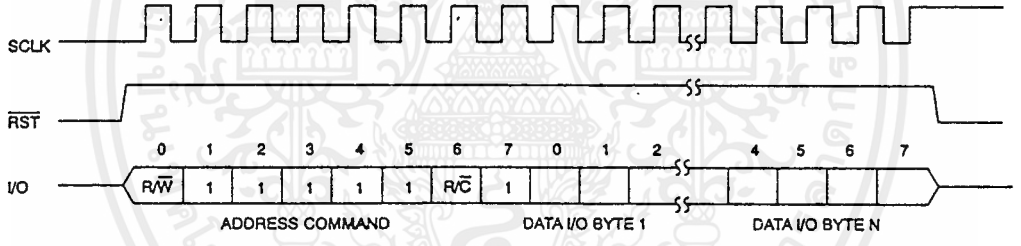
A 32.768 KHz crystal, Daiwa Part No. DT26S, Seiko Part No. DS-VT-200 or equivalent, can be directly connected to the DS1202 via pins 2 and 3 (X1, X2). The crystal selected for use should have a specified load capacitance (CL) of 6 pF. The crystal is connected directly to the X1 and X2 pins. There is no need for external capacitors or resistors. Note: X1 and X2 are very high impedance nodes. It is recommended that they and the crystal be guard–ringed with ground and that high frequency signals be kept away from the crystal area. For more information on crystal selection and crystal layout considerations, please consult Application Note 58, "Crystal Considerations with Dallas Real Time Clocks". Crystals can be ordered from Dallas Semiconductor. Order part number DS9032.

DATA TRANSFER SUMMARY Figure 3

SINGLE BYTE TRANSFER



BURST MODE TRANSFER



FUNCTION	BYTE N	SCLK n
CLOCK	8	72
RAM	24	200

RESET AND CLOCK CONTROL

All data transfers are initiated by driving the $\overline{RST}$ input high. The $\overline{RST}$ input serves two functions. First, $\overline{RST}$ turns on the control logic which allows access to the shift register for the address/command sequence. Second, the $\overline{RST}$ signal provides a method of terminating either single byte or multiple byte data transfer. A clock cycle is a sequence of a falling edge followed by a rising edge. For data inputs, data must be valid during the rising edge of the clock and data bits are output on t he falling edge of clock. All data transfer terminates if the $\overline{RST}$ input is low and the I/O pin goes to a high impedance state. Data transfer is illustrated in Figure 3.

DATA INPUT

Following the eight SCLK cycles that input a write command byte, a data byte is input on the rising edge of the next eight SCLK cycles. Additional SCLK cycles are ignored should they inadvertently occur. Data is input starting with bit 0.

DATA OUTPUT

Following the eight SCLK cycles that input a read command byte, a data byte is output on the falling edge of the next eight SCLK cycles. Note that the first data bit to be transmitted occurs on the first falling edge after the last bit of the command byte is written. Additional SCLK cycles retransmit the data bytes should they inadvertently occur so long as $\overline{RST}$ remains high. This operation permits continuous burst mode read capability. Data is output starting with bit 0.

BURST MODE

Burst mode may be specified for either the clock/calendar or the RAM registers by addressing location 31 decimal (address/command bits one through five = logical one). As before, bit six specified clock or RAM and bit 0 specifies read or write. There is no data storage capacity at locations 8 through 31 in the Clock/Calendar Registers or locations 24 through 31 in the RAM registers. When writing to the clock registers in the burst mode, the first eight registers must be written in order for the data to be transferred.

However, when writing to RAM in burst mode it is not necessary to write all 24 bytes for the data to transfer. Each byte that is written to will be transferred to RAM regardless of whether all 24 bytes are written or not.

CLOCK/CALENDAR

The clock/calendar is contained in eight write/read registers as shown in Figure 4. Data contained in the clock/calendar registers is in binary coded decimal format (BCD).

CLOCK HALT FLAG

Bit 7 of the seconds register is defined as the clock halt flag. When this bit is set to logic 1, the clock oscillator is stopped and the DS1202 is placed into a low-power standby mode with a current drain of not more than 100 nanoamps. When this bit is written to logic 0, the clock will start.

AM-PM/12-24 MODE

Bit 7 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).

WRITE PROTECT REGISTER

Bit 7 of write protect register is the write protect bit. The first seven bits (bits 0-6) are forced to zero and will always read a zero when read. Before any write operation to the clock or RAM, bit 7 must be zero. When high, the write protect bit prevents a write operation to any other register.

CLOCK/CALENDAR BURST MODE

The clock/calendar command byte specifies burst mode operation. In this mode the eight clock/calendar registers can be consecutively read or written (see Figure 4) starting with bit 0 of address 0.

RAM

The static RAM is 24 x 8 bytes addressed consecutively in the RAM address space.

RAM BURST MODE

The RAM command byte specifies burst mode operation. In this mode, the 24 RAM registers can be consecutively read or written (see Figure 4) starting with bit 0 of address 0.

REGISTER ADDRESS/DEFINITION Figure 4

REGISTER ADDRESS
A. CLOCK

	7	6	5	4	3	2	1	0
SEC	1	0	0	0	0	0	0	RD/$\overline{W}$
MIN	1	0	0	0	0	0	1	RD/$\overline{W}$
HR	1	0	0	0	0	1	0	RD/$\overline{W}$
DATE	1	0	0	0	0	1	1	RD/$\overline{W}$
MONTH	1	0	0	0	1	0	0	RD/$\overline{W}$
DAY	1	0	0	0	1	0	1	RD/$\overline{W}$
YEAR	1	0	0	0	1	1	0	RD/$\overline{W}$
CONTROL	1	0	0	0	1	1	1	RD/$\overline{W}$
CLOCK BURST	1	0	1	1	1	1	1	RD/$\overline{W}$

REGISTER DEFINITION

00–59	CH	10 SEC	SEC			
00–59	0	10 MIN	MIN			
01–12 00–23	12/24	0	10 A/P	HR	HR	
01–28/29 01–30 01–31	0	0	10 DATE	DATE		
01–12	0	0	0	10 M	MONTH	
01–07	0	0	0	0	0	DAY
0–99	10 YEAR	YEAR				
	WP	FORCED TO ZERO				

B. RAM

RAM 0	1	1	0	0	0	0	0	RD/$\overline{W}$
RAM 23	1	1	1	0	1	1	1	RD/$\overline{W}$
RAM BURST	1	1	1	1	1	1	1	RD/$\overline{W}$

RAM DATA 0

RAM DATA 23

AC ELECTRICAL CHARACTERISTICS

(0°C to 70°C; V_{CC} = 2.0 to 5.5V*)

PARAMETER	SYMBOL		MIN	TYP	MAX	UNITS	NOTES
Data to CLK Setup	t_{DC}	V_{CC}=2V	200			ns	7
		V_{CC}=5V	50				
CLK to Data Hold	t_{CDH}	V_{CC}=2V	280			ns	7
		V_{CC}=5V	70				
CLK to Data Delay	t_{CDD}	V_{CC}=2V			800	ns	7, 8, 9
		V_{CC}=5V			200		
CLK Low Time	t_{CL}	V_{CC}=2V	1000			ns	7
		V_{CC}=5V	250				
CLK High Time	t_{CH}	V_{CC}=2V	1000			ns	7, 12
		V_{CC}=5V	250				
CLK Frequency	f_{CLK}	V_{CC}=2V			0.5	MHz	7, 12
		V_{CC}=5V	DC		2.0		
CLK Rise and Fall	t_R, t_F	V_{CC}=2V			2000	ns	
		V_{CC}=5V			500		
RST to CLK Setup	t_{CC}	V_{CC}=2V	4			µs	7
		V_{CC}=5V	1				
CLK to RST Hold	t_{CCH}	V_{CC}=2V	1000			ns	7
		V_{CC}=5V	250				
RST Inactive Time	t_{CWH}	V_{CC}=2V	4			µs	7
		V_{CC}=5V	1				
RST to I/O High Z	t_{CDZ}	V_{CC}=2V			280	ns	7
		V_{CC}=5V			70		

*Unless otherwise noted.

ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to Ground	–0.3V to +7.0V
Operating Temperature	0°C to 70°C
Storage Temperature	–55°C to +125°C
Soldering Temperature	260°C for 10 seconds

* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS

(0°C to 70°C)

PARAMETER	SYMBOL		MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{CC}		2.0		5.5	V	1
Logic 1 Input	V_{IH}		2.0		V_{CC}+0.3	V	1
Logic 0 Input	V_{IL}	V_{CC}=2.0V	–0.3		+0.3	V	1
		V_{CC}=5V	–0.3		+0.8		

DC ELECTRICAL CHARACTERISTICS

(0°C to 70°C; V_{CC} = 2.0 to 5.5V*)

PARAMETER	SYMBOL		MIN	TYP	MAX	UNITS	NOTES
Input Leakage	I_{LI}				+500	µA	6
I/O Leakage	I_{LO}				+500	µA	6
Logic 1 Output	V_{OH}	V_{CC}=2V	1.6			V	2
		V_{CC}=5V	2.4				
Logic 0 Output	V_{OL}	V_{CC}=2V			0.4	V	3
		V_{CC}=5V			0.4		
Active Supply Current	I_{CC}	V_{CC}=2V			.4	mA	5
		V_{CC}=5V			1.2		
Timekeeping Current	I_{CC1}	V_{CC}=2V			0.3	µA	4
		V_{CC}=5V			1		
Leakage Current	I_{CC2}	V_{CC}=2V			100	nA	10
		V_{CC}=5V			100		

*Unless otherwise noted.

CAPACITANCE

(t_A = 25°C)

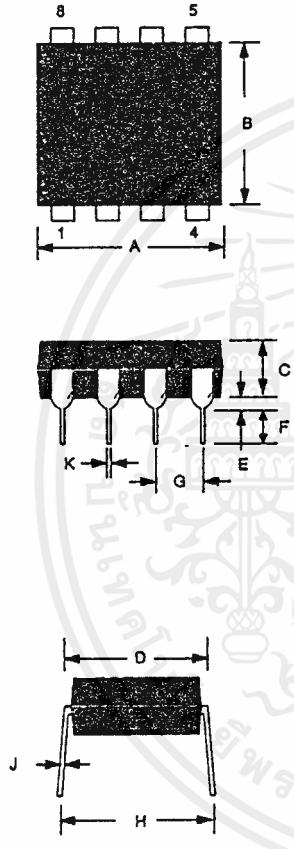
PARAMETER	SYMBOL	CONDITION	TYP	MAX	UNITS	NOTES
Input Capacitance	C_I		5		pF	
I/O Capacitance	$C_{I/O}$		10		pF	
Crystal Capacitance	C_X		6		pF	

10. I_{CC2} is specified with $\overline{RST}$, I/O, and SCLK open. The clock halt flag must be set to logic one (oscillator disabled).

11. At power–up, $\overline{RST}$ must be at a logic 0 until $V_{CC} \geq 2$ volts. Also, SCLK must be at a logic 0 when $\overline{RST}$ is driven to a logic one state.

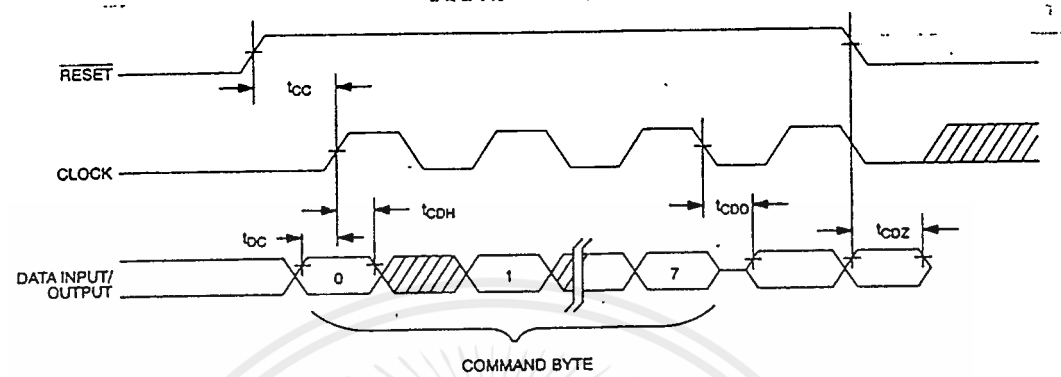
12. If t_{CH} exceeds 100 ms with $\overline{RST}$ in a logic one state, then I_{CC} may briefly exceed I_{CC} specification.

DS1202 SERIAL TIMEKEEPER 8–PIN DIP

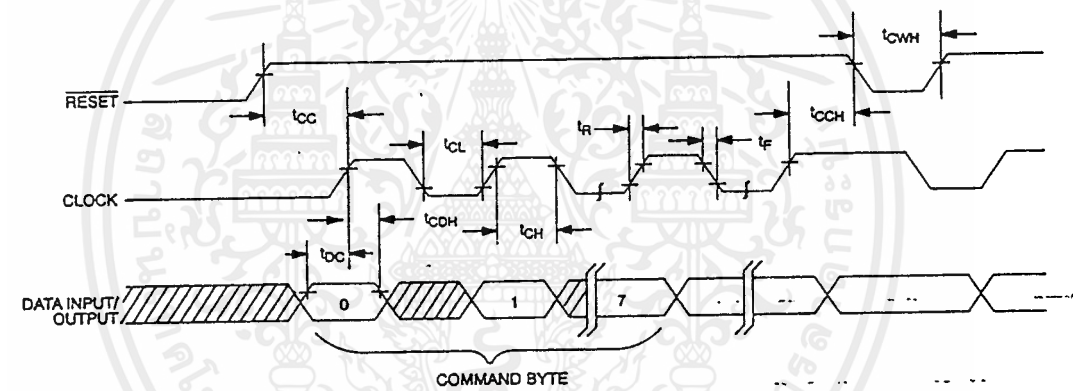


PKG	8–PIN	
DIM	MIN	MAX
A IN. MM	.360	.400
B IN. MM	.240	.260
C IN. MM	.120	.140
D IN. MM	.300	.325
E IN. MM	.015	.040
F IN. MM	.110	.140
G IN. MM	.090	.110
H IN. MM	.320	.370
J IN. MM	.008	.012
K IN. MM	.015	.021

TIMING DIAGRAM: READ DATA TRANSFER Figure 5



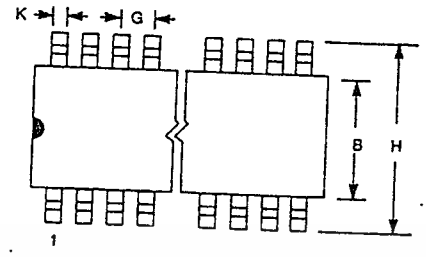
TIMING DIAGRAM: WRITE DATA TRANSFER Figure 6

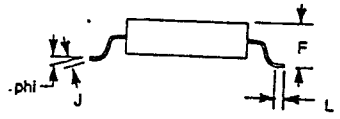


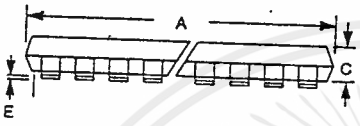
NOTES:

1. All voltages are referenced to ground.
2. Logic one voltages are specified at a source current of 1 mA at V_{CC}=5V and .4 mA at V_{CC}=2V, V_{OH}=V_{CC} for capacitive loads.
3. Logic zero voltages are specified at a sink current of 4 mA at V_{CC}=5V and 1.5 mA at V_{CC}=2V.
4. I_{CC1} is specified with I/O open, $\overline{RST}$ set to a logic 0, and clock halt flag=0 (oscillator enabled).
5. I_{CC} is specified with the I/O pin open, $\overline{RST}$ high, SCLK=2 MHz at V_{CC}=5V; SCLK=500 KHz, V_{CC}=2V and clock halt flag=0 (oscillator enabled).
6. $\overline{RST}$, SCLK, and I/O all have 40KΩ pulldown resistors to ground.
7. Measured at V_{IH}=2.0V or V_{IL}=0.8V and 10 ms maximum rise and fall time.
8. Measured at V_{OH}=2.4V or V_{OL}=0.4V.
9. Load capacitance = 50 pF.

DS1202S SERIAL TIMEKEEPER 16–PIN SOIC

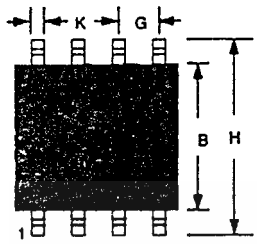


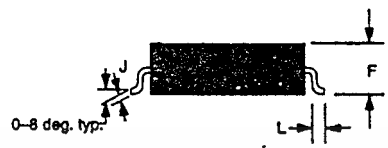




PKG	16-PIN	
DIM	MIN	MAX
A IN. MM	0.500 12.70	0.511 12.99
B IN. MM	0.290 .7.37	0.300 7.65
C IN. MM	0.089 2.26	0.095 2.41
E IN. MM	0.004 0.102	0.012 0.30
F IN. MM	0.094 2.38	0.105 2.68
G IN. MM	0.050 BSC 1.27 BSC	
H IN. MM	0.398 10.11	0.416 10.57
J IN. MM	0.009 0.229	0.013 0.33
K IN. MM	0.013 0.33	0.019 0.48
L IN MM	0.016 0.406	0.040 1.20
phi	0°	8°

DS1202S8 8–PIN SOIC 200 MIL





PKG	8-PIN	
DIM	MIN	MAX
A IN. MM	0.203 5.16	0.213 5.41
B IN. MM	0.203 5.16	0.213 5.41
C IN. MM	0.070 1.78	0.074 1.88
E IN. MM	0.004 0.102	0.010 0.390
F IN. MM	0.074 1.88	0.84 2.13
G IN. MM	0.050 BSC 1.27 BSC	
H IN. MM	0.302 7.67	0.318 8.07
J IN. MM	0.006 0.152	0.010 0.254
K IN. MM	0.013 0.33	0.020 0.508
L IN. MM	0.19 4.83	0.030 0.762