เครื่องแสดงผลระยะไกล

Remote Line Monitoring

โดย

นายกร ลิบวาณิชย์

นายเกษม ชุมสงค์

นางสาวจันทณี อาชาพานิช

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขาวิชาเทคโนโลยีอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

หัวข้อปริญญานิพนธ์	**เครื่องแสดงผลระยะไกล**
	Remote Line Monitoring
ชื่อนักศึกษา	**นายกร ลิบวาณิชย์**
	นายเกษม ชุมสงค์
	นางสาวจันทณี อาชาพานิช
อาจารย์ที่ปรึกษา	**อาจารย์กฤดากร กล่อมการ**

คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหลาดกระบัง อนุมัติปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตร์บัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

.....ประธานกรรมการ

( )

.....กรรมการ

( )

.....กรรมการ

( )

.....กรรมการ

( )

.....กรรมการ

( )

ลิขสิทธิ์ของคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เครื่องแสดงผลระยะไกล

โดย นายกร ลิบวาณิชย์ รหัส ๓๗๐๑๓๓๒๖
นายเกษม ชุมสงค์ รหัส ๓๗๐๑๓๓๒๘
นางสาวจันทณี อาชาพานิช รหัส ๓๗๐๑๓๓๓๒

อาจารย์ที่ปรึกษา อาจารย์กฤดากร กล่อมการ

ปีการศึกษา ๒๕๓๕

บทคัดย่อ

วัตถุประสงค์ของปริญญานิพนธ์ฉบับนี้เพื่อเป็นการศึกษาการส่งสัญญาณระหว่างอุปกรณ์ที่อยู่ระยะไกลโดยสายนำสัญญาณ RS 232 นำสัญญาณผ่านพอร์ทสื่อสารเพื่อแสดงผลผ่านทางคอมพิวเตอร์โดยโปรแกรม Visual Basic V.3

ในส่วนของตัวอุปกรณ์เป็นการศึกษาวงจรอนาลอก/ดิจิตอล และ Z80180

ในโครงงานนี้จะเป็นการนำค่าของสัญญาณที่วัดได้ (0-5 V.) มาแสดงที่จอภาพ ซึ่งจะนำไปประยุกต์ใช้กับงานทางด้านเครื่องวัดต่อๆ ได้ต่อไป

REMOTE MONITORING

BY	MR.KORN LIBVANICH.	37013326
	MR.KASEM CHUMSONG.	37013329
	MISS.CHANTANEE ACHAPANICH.	37013332

ADVISOR MR.KITDAKORN KLOMKARN.

YEAR 1996

ABSTRACT

This project was designed for studied about signal transmission betaween terminal equipment by RS-232 transmission line passed to the serial port of the computer to monitor by the Visual Basic V.3 programing.

In the equipment was interested in the Analog to Digital circuit and Z80180. This project about signal transmission was send from the sensor(0-5 V.) to the monitor. It also could be used for apply to work with the instument job.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปได้ด้วยดี จากความช่วยเหลืออย่างดี ของอาจารย์กฤดากร กล่อมการ อาจารย์ที่ปรึกษาซึ่งท่านได้ให้คำแนะนำ และข้อคิดเห็นต่างๆของการทำปริญญานิพนธ์ด้วยดีตลอด

ขอขอบคุณความช่วยเหลือจากเพื่อนๆทุกคนในด้านต่างๆและสำคัญที่สุดขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และพี่ชาย ซึ่งเป็นผู้ที่ให้ความช่วยเหลืออย่างยิ่งทั้งกำลังใจ และทุนทรัพย์ในการทำปริญญานิพนธ์

คณะผู้จัดทำ

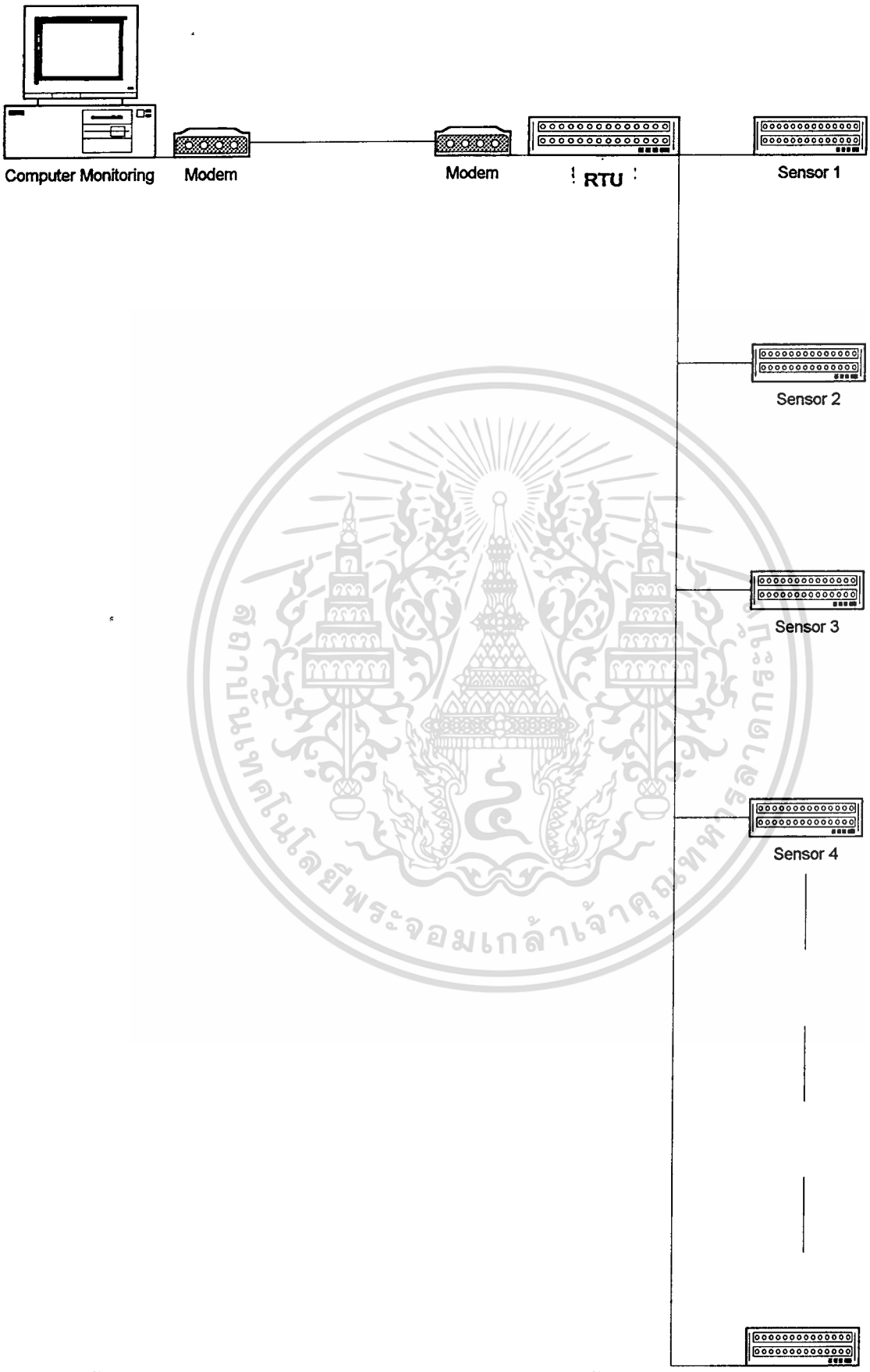
สารบัญ

หน้า

2. ส่วนของคอมพิวเตอร์แสดงผลของสัญญาณที่ถูกส่งมากจากส่วนของเทอร์มินอล

ซึ่งส่วนของคอมพิวเตอร์แสดงผลนี้จะติดต่อกับส่วนของเทอร์มินอล โดยผ่านทางมาตรฐาน RS-232 แล้วจะแสดงผลโดยใช้ Visual Basic ในการเขียนโปรแกรม

ซึ่ง Visual Basic จึะมีสภาพแวดล้อมสำหรับการพัฒนาโปรแกรมบน Windows ประกอบด้วยเครื่องมือต่าง ๆ ครบถ้วน ไม่ว่าจะเป็นส่วนของการออกแบบ (User Interface),ส่วนออกแบบเมนู (Menu Designer), การสร้างรายงาน (Report Writer), อิดิเตอร์สำหรับป้อนโปรแกรม และ Debugger เพื่อตรวจหาข้อผิดพลาดในการเขียนโปรแกรม องค์ประกอบเหล่านี้นับว่าเอื้ออำนวยต่อการทำงานของโปรแกรมเมอร์อย่างมาก



รูปที่ 1.1 รูปแสดงลักษณะของโครงงานโดยรวม

บทที่ 2

โครงสร้างของเทอร์มินอล (Terminal) และทฤษฎี

โครงสร้างของเทอร์มินอล (Terminal)

โครงสร้างในส่วนของเทอร์มินอล (Terminal) จะเป็นส่วนที่อ่านสัญญาณจากเครื่องมอวัด เพื่อส่งออกไปแสดงผลยังจอคอมพิวเตอร์

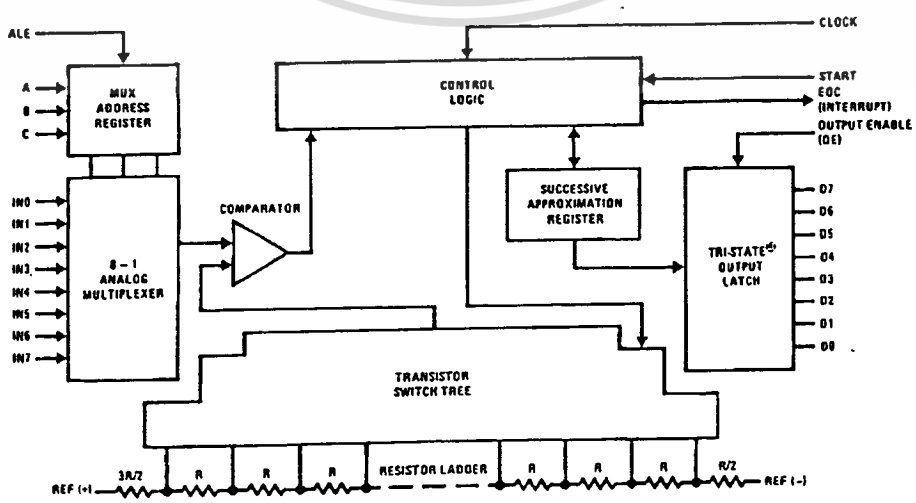
ในส่วนของเทอร์มินอล (Terminal) นี้จะใช้วงจร A/D คอนเวอร์เตอร์ แปลงสัญญาณแรงดันซึ่งเป็นสัญญาณอนาลอกจากเอาท์พุทของเครื่องมือวัด ให้เป็นรหัสสัญญาณดิจิตอล และส่งรหัสสัญญาณดิจิตอลต่อไปยัง CPU (Computer Processing Unit) เพื่อควบคุมการอ่านรหัสสัญญาณดิจิตอลดังกล่าว และส่งออกทางพอร์ตอนุกรมเพื่อที่จะได้แสดงผลยังจอคอมพิวเตอร์ ดังแสดงรูปบล็อกไดอะแกรม (รูปที่ 2.2)

ต่อจากนี้จะกล่าวถึงทฤษฎีที่จะใช้ในส่วนของเทอร์มินอล (Terminal)

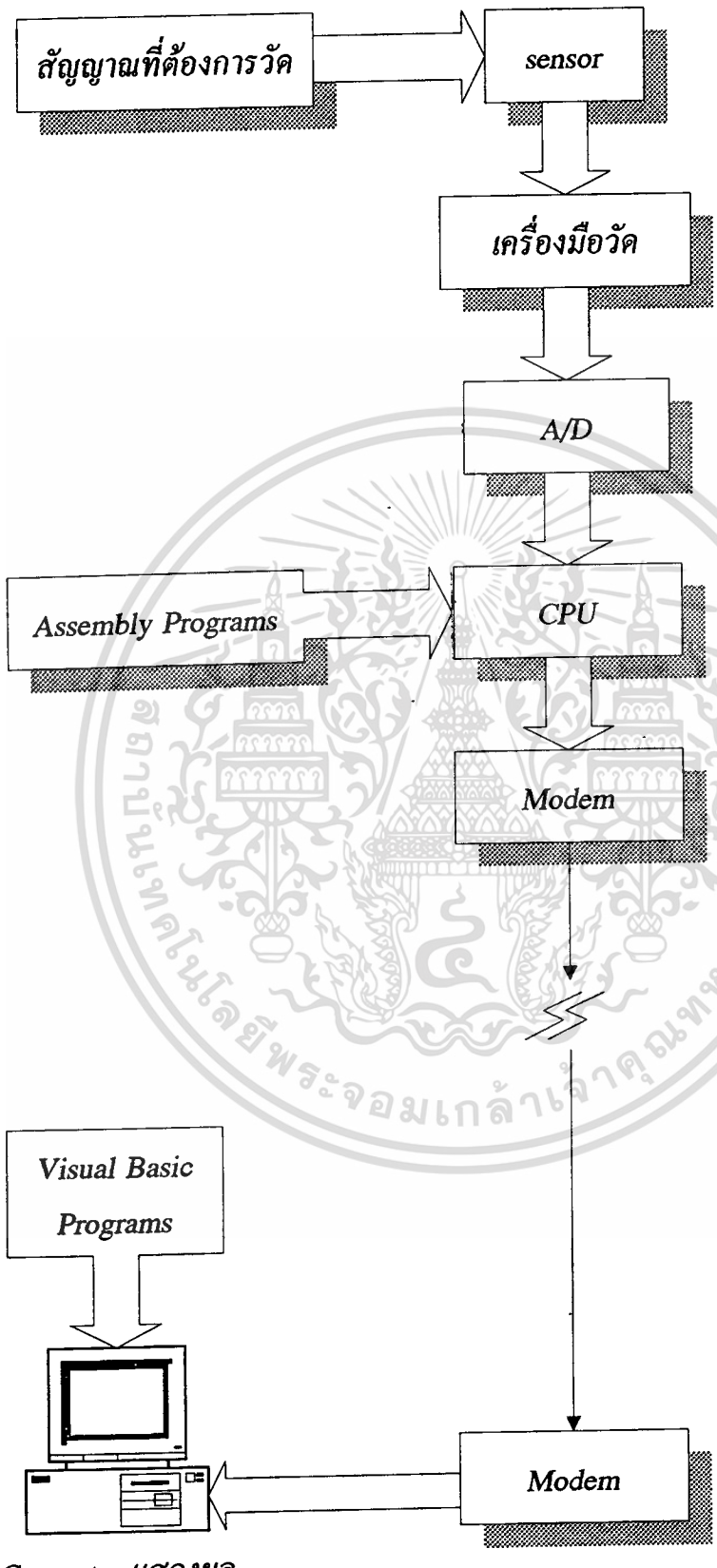
A/D คอนเวอร์เตอร์

เป็นวิธีการแปลงค่าแรงดันอนาลอกเป็นทางดิจิตอล ในปัจจุบัน A/D คอนเวอร์เตอร์มีหลายวิธี แต่ในโครงงานนี้ใช้วิธีซัคเซสซีฟ แอปพรอทซิเมชัน (Successive Approximation) ซึ่งแสดงดังรูป

ทฤษฎีการทำงานของวงจร A/D คอนเวอร์เตอร์ โดยวิธีซัคเซสซีฟ แอปพรอทซิเมชัน (Successive Approximation)



รูป 2.1 รูปแสดง A/D คอนเวอร์ทเตอร์วิธีซัคเซสซีฟ แอปพรอทซิเมชัน



รูปที่ 2.2 รูปแสดงบล็อกไดอะแกรมในส่วนของเทอร์มินอล(Terminal)

ซัคเซสซีฟ แอปพรอทซิเมชั่น (Successive Approximation) คือ มีวงจรควบคุม , SAR รีจิสเตอร์ (Successive Approximation Register) และบัฟเฟอร์รีจิสเตอร์ เมื่อการแปลงสัญญาณสิ้นสุด เราได้รับข้อมูลดิจิตอลเอ้าท์พุทผ่านทางบัฟเฟอร์รีจิสเตอร์ คือ D_0 ถึง D_7

เมื่อเริ่มต้นการแปลงข้อมูลสัญญาณ เริ่มต้นการแปลง (start of conversion) จะเป็น "0" SAR รีจิสเตอร์จะถูกเคลียร์ และ V_{out} จะมีค่าเป็นศูนย์ เมื่อสัญญาณเริ่มต้นการแปลงเป็น "1" การแปลงข้อมูลก็จะเริ่มต้นโดยการนับเพิ่มครั้งละ 1 บิท จากการเซ็ทค่าที่ MSB (Most Significant Bit) ก่อนดังนี้

ในระหว่างพัลซ์แรกของสัญญาณนาฬิกา วงจรควบคุมจะส่ง "1" ของ MSB มาที่ SAR รีจิสเตอร์ ซึ่งทำให้เอ้าท์พุทของ SAR รีจิสเตอร์ มีค่าเท่ากับ 10,000,000 ทันทีที่ค่าดิจิตอลเอ้าท์พุทนี้เกิดขึ้น A/D คอนเวอร์เตอร์
ก็จะแปลงค่าได้ V_{out} เท่ากับ 128/255 เท่า ถ้าค่า V_{out} ที่ได้นี้มากกว่าค่าอนาลอกอินพุท (V_{in}) คอมพาราเตอร์จะส่งสัญญาณเอ้าท์พุทที่เป็นลบ ไปวงจรควบคุมเพื่อรีเซ็ท MSB ในทางตรงข้าม ถ้า V_{out} น้อยกว่าค่าอนาลอคอิทพุท(V_{in}) MSB ก็จะยังคงเซ็ทค่าอยู่

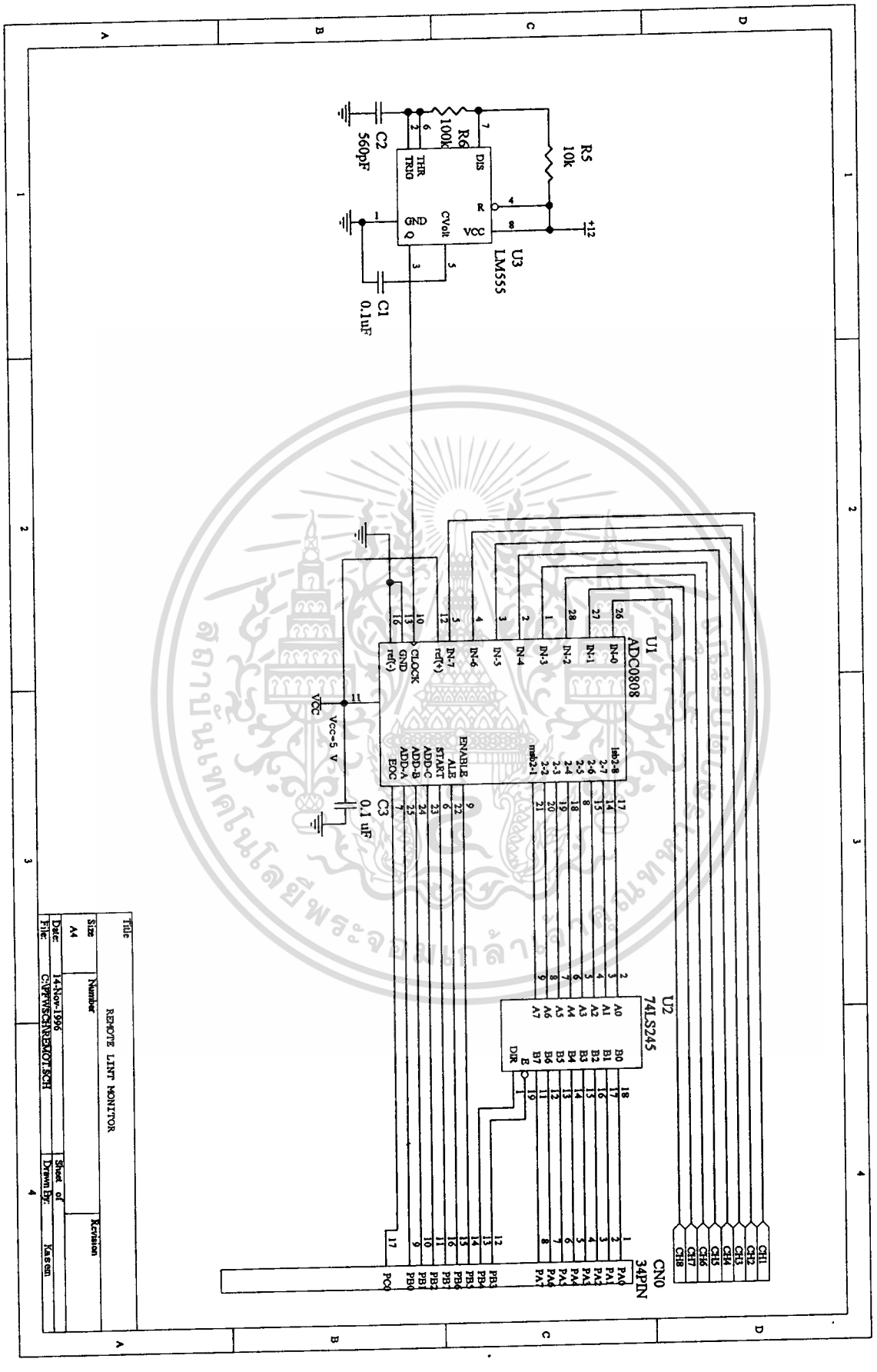
ในกรณีที่ MSB ยังไม่ถูกรีเซ็ท ค่าที่อยู่ใน SAR รีจิสเตอร์ปัจจุบันคือ 10,000,000 ในช่วงของพัลซ์สัญญาณนาฬิกาต่อไปก็จะเซ็ทค่าบิท ที่ต่อจาก MSB หรือ บิท D_6 ก็จะได้ดิจิตอลเอ้าท์พุทเท่ากับ 11,000,000 V_{out} ก็จะเพิ่มขึ้นเป็น 192/255 เท่า ถ้า V_{out} มากกว่าค่าอนาลอกอินพุท (V_{in}) สัญญาณเอ้าท์พุทที่เป็นลบจากคอมพาราเตอร์ก็จะรีเซ็ทค่าบิท D_6 แต่ถ้า V_{out} น้อยกว่าค่าอนาลอกอินพุท บิท D_6 ก็จะยังคงอยู่

การเพิ่มค่าจำนวนบิท และทำการทดสอบค่าด้วยคอมพาราเตอร์จะถูกกระทำทุก ๆ พัลซ์สัญญาณนาฬิกา ดังนั้นการแปลงข้อมูลจะสิ้นสุดหลังจาก 8 พัลซ์สัญญาณนาฬิกา เมื่อสิ้นสุดการแปลงข้อมูลแล้ว วงจรควบคุมจะส่งสัญญาณ "0" บอกว่าสิ้นสุดการแปลงข้อมูล (end of coversion) ไปให้บัฟเฟอร์ รีจิสเตอร์ ข้อมูลดิจิตอลเอ้าท์พุท ก็จะถูกเก็บอยู่ในบัฟเฟอร์ รีจิสเตอร์ ถึงแม้ว่าจะเริ่มวงจรการแปลงข้อมูลใหม่อีกครั้งหนึ่ง

ข้อดีของวิธีซัคเซสซีฟ แอปพรอทซิเมชั่น นี้คือ ในด้านความเร็วจะเห็นได้ว่าใช้เวลาเพียงแค่ n พัลซ์สัญญาณนาฬิกา ก็จะให้ค่า n บิท เรสโซลูชั่นของค่าอนาลอกซึ่งจะดีกว่าวิธีเคาน์เตอร์อย่างมากมาย

การทำงานของวงจร A/D คอนเวอร์เตอร์ที่ใช้ในโครงงานนี้

วงจร A/D นี้ใช้ไอซีเบอร์ ADC0808 เป็นตัวแปลงสัญญาณอนาลอกเป็นดิจิตอลจากรูปสัญญาณอินพุทจะถูกป้อนเข้าสู่ขา 26 (IN0) , 27 (IN1) , 28 (IN2) , 1 (IN3) , 2 (IN4) , 3 (IN5)4 (IN6) , 5 (IN7) ซึ่งเป็นขาอินพุทของสัญญาณช่องที่ 1 ถึง 8 ตามลำดับ สำหรับสัญญาณ Clock ที่ป้อนให้แก่ไอซี ADC0808 นั้นจะสร้างจากไอซี LM555 ซึ่งเอาท์พุทที่ได้ออกมาจะมีความถี่ 500 Mhz (ที่ขาที่ 3) ถูกป้อนให้กับขาที่ 10 (CLK) ของไอซี ADC0808 ส่วน Vref (+) นั้นใช้ไฟบวก 5 โวลต์ ต่อที่ขา 12 ของไอซี ADC0808 และ Verf(-) ที่ขา 16 ต่อลง Ground ไอซี ADC0808 นั้นจะทำงานได้ก็ต่อเมื่อสัญญาณที่ขา Start , ALE , OE เป็น "1" และขาที่ 23 (ADD-C) , 24 (ADD+B) , 25 (ADD-A) จะเป็นตัวเลือก Channnel ที่จะอ่านตั้งแต่ Channel ที่ "0-7" (000-111) ตามลำดับ ADC0808 นั้นจะรับค่าสัญญาณอนาลอก ทางด้านอินพุทแปลงสัญญาณดิจิตอลออกทางด้านเอาท์พุทที่ขา D0-D7 และจะให้สัญญาณที่ขา 7 (EOC) เป็น "1" เพื่อจะบอกว่าสัญญาณอนาลอกถูกแปลงเป็นสัญญาณดิจิตอลเรียบร้อยแล้ว สัญญาณก็จะผ่านเข้าไปที่ไอซี 74LS245 ซึ่งเป็น Buffer สองทางขั้นกลางระหว่าง Port A ของไอซี 8255 กับเอาท์พุทของ ADC0808 เมื่อสัญญาณ DIR เป็น "1" และ E เป็น "0" สัญญาณที่ไอซี 74LS245 ที่ขา "2-9" (A0-A7) จะถูกเชื่อมต่อกับขา "11-18" (B7-B0) สัญญาณดิจิตอลก็จะไปปรากฏอยู่ที่ Port 8255 เพื่อให้ CPU อ่านค่าเข้าไปประมวลผลต่อไป

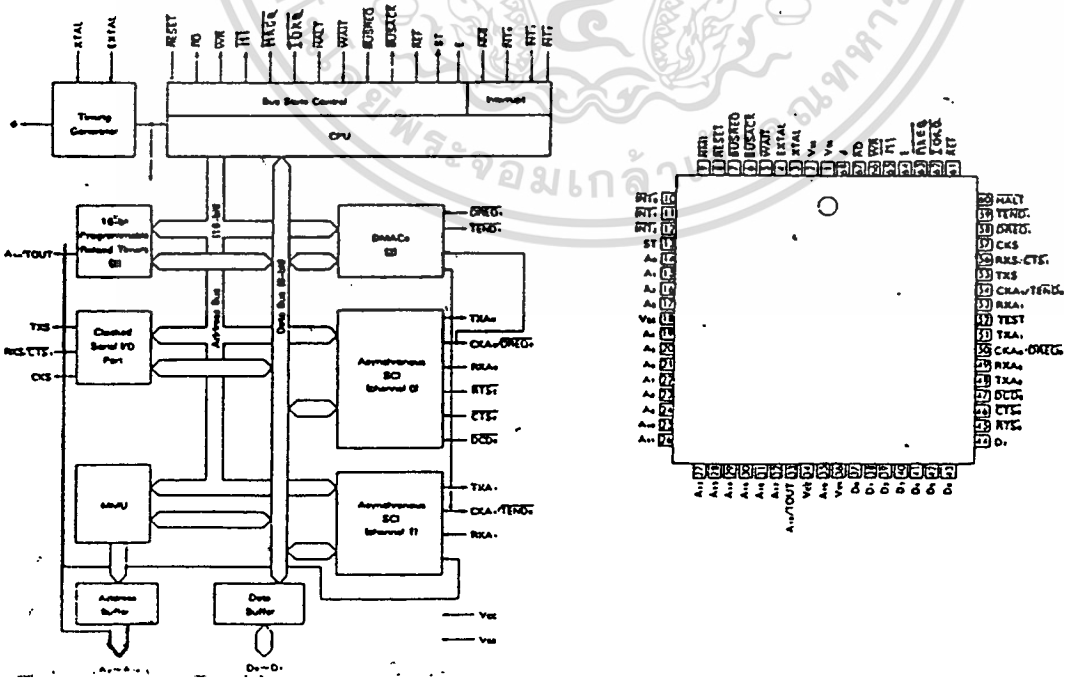


รูปที่ 2.3 รูปแสดงวงจร Analog to Digital.

CPU (COMPUTER PROCESSING UNIT)

ในโครงงานนี้ใช้ Z80180 เพราะเป็น CPU ที่มีความสามารถสูงที่ได้รวม Chip สำคัญอื่น ๆ ไว้ใน CPU Chip เดี่ยวจึงทำให้มีลักษณะคล้ายกับ CPU ที่ใช้ในงาน Control ในจำพวก "Single Chip" ซึ่งเป็นข้อดีคือ ระบบเล็กราคาถูกและด้านโปรแกรมก็สะดวกอย่างมาก เพราะคำสั่งที่ใช้มีมาก และตรงไปตรงมาเพราะ CPU Z80180 เป็น Supper CompatZ80 คือคำสั่งทั้งหมดยังเป็น Z80 และได้เพิ่มชุดคำสั่งขึ้นมาเพื่อเพิ่มความสะดวกในการใช้งาน

ดูระบบ Micro Controller "Single Chip" แล้ว Z80180 จะด้วยกว่าตรงที่ไม่มี ROM,RAM และ PORT แต่ถ้าเป็นในงานควบคุมส่งสัญญาณออกทางพอร์ตอนุกรม Z80180 กับ Chip Micro Controller ไม่ต่างกันเลย เพราะความต้องการเนื้อที่ในการเก็บข้อมูลมาก และ Port มากตาม จึงทำให้ต้องต่อเพื่อภายนอกขึ้น จึทำให้การทำงานคล่องตัวกว่ามาก เพราะภายใน Z80180 ประกอบด้วยเป็น CMOS, Oscilator ในตัว RUN ที่ 10 Mhz, MMU Chip อ้าง Memory ได้ 1 MBYTE, DMA 2 Chanel. Port สื่อสาร UART 2 Chanel. Clock Serial I/O, 16 Bit Timer Counter และเกี่ยวกับ Port สื่อสารสามารถทำ Multi Processor Communication ซึ่งโครงสร้างของ Chip นี้จะเป็นดังรูปที่ 2.4



รูปที่ 2.4 รูปแสดงบล็อกไดอะแกรมของ Z80180 และแสดงรูปรางของ Z80180

ขาการใช้งาน

A0-A19	ADDRESS BUS ระหว่าง RESET จะเป็น HIGHT IMPEDANCE
$\overline{\text{BUSAK}}$	BUS ACKNOWLEDGE เป็นขา OUTPUT ACTIVE LOW ทำงานก็ต่อเมื่อ Z80180 ตอบสนองต่อการขอ BUS ของ $\overline{\text{BUSRQ}}$ และจะทำให้ BUS ข้อมูล BUS ADDRESS และญญาณCONTROL บางเส้นเป็น HIGH IMPEDANCE
$\overline{\text{BUSRQ}}$	BUS REQUEST เป็นขา INPUT ACTIVE LOW ซึ่งจะมีความสำคัญสูงกว่า NMI โดยจะมีการตรวจสอบสัญญาณนี้ทุก ๆ การสิ้นสุดของ MACHINE CYCLE
CKA0 , CKA1A	SYNCHRONOUS CLOCK 0 และ 1 เป็นขาสัญญาณ CLOCKแบบ 2 ทิศทาง คือ จะใช้เป็นขา INPUT หรือ OUTPUT ก็ได้
CKS	SERIAL CLOCK เป็นขา CLOCK 2 ทิศทางของ CSI/0
CLOCK	เป็นขา OUTPUT โดยจะเป็นครึ่งหนึ่งของ X'TAL หรือ CLOCK OUT เช่น X'TAL 12 Mhz Z80180 จะ RUN ที่ 6 Mhz
$\overline{\text{CTS0}}$-$\overline{\text{CTS1}}$	CLEAR TO SEND 0 และ 1 เป็นขา INPUT ACTIVE
LOW	ใช้ในการควบคุม MODEM
D0-D7	DATA BUS เป็นแบบ 2 ทิศทาง
$\overline{\text{DCD0}}$	DATA CARRIER DETECT 0 เป็นขา INPUT ACTIVE
LOW	ใช้ควบคุมในการติดต่อกับ MODEM ของ ASCI CHANEL 0
$\overline{\text{DREQ0}}$-$\overline{\text{DREQ1}}$	DMA REQUEST 0 และ 1 เป็นขา INPUT ACTIVE LOW ใช้ในการขอ DMA และขานี้จะโปรแกรมได้ว่าให้ตรวจสัญญาณที่ขอบหรือระดับได้
E	ENABLE CLOCK เป็นขา OUTPUT ACTIVE HIGH ซึ่งใช้ซิงค์การทำงานกับอุปกรณ์ภายนอก ระหว่างการทำงานเกี่ยวกับ BUS และใช้เชื่อมต่อกับอุปกรณ์ในตระกูล 68XX และ 80XX

$\overline{\text{HALT}}$	เป็นขา OUTPUT ACTIVE LOW จะทำงานเมื่อทำคำสั่ง HALT หรือ SLP
$\overline{\text{INT0}}$	MASKABLE INTERRUPT 0 เป็นขา INPUT ACTIVE LOW สัญญาณที่ขานี้จะถูกตรวจทุก ๆ การสิ้นสุดของคำสั่ง
$\overline{\text{INT1}}$, $\overline{\text{INT2}}$	เช่นเดียวกับ INTO แต่มีระดับความสำคัญรองลงมาตามลำดับ
$\overline{\text{IORQ}}$	เป็นขา OUTPUT เพื่อบอกว่ากำลังติดต่อกับ I/O หรือขา $\overline{\text{IOE}}$ ใน 64180
$\overline{\text{M1}}$	MACHINE CYCLE 1 เป็นขา OUTPUT ACTIVE LOW จะทำงานเมื่อ FETCH OP-CODE หรือเป็นขา LIR ของ 64180
$\overline{\text{NMI}}$	NON MASKABLE INTERRUPT เป็นขา INPUT ACTIVE LOW ขานี้จะตอบรับการ INTERRUPT เสมอ โดยไม่สามารถหยุดด้วย SOFTWARE
$\overline{\text{RD}}$	เป็นขาที่ใช้ทำการอ่านข้อมูลจาก MEMORY หรือ I/O
$\overline{\text{RFSH}}$	เป็นขาที่ให้ ADRESS LOW (A0-A7) ไป EFRESH DYNAMIC RAM หรือ ขา $\overline{\text{REF}}$ ของ 64180
$\overline{\text{RTSO}}$	REQUEST TO SEND เป็นขา OUTPUT ACTIVE LOW ขานี้ใช้โปรแกรม สัญญาณควบคุมโมเด็มของ ASCI CANEL 0
RXA0 , RXA1	RECEIVE DATA 0 และ 1 เป็นขารับสัญญาณจาก SERIAL PORT ของ ASCI
RXS	CLOCK SERIAL RECEIVE DATA เป็นขารับญญาณSERIAL ของ CSIO
ST	STATUS เป็นขา OUTPUT ACTIVE HIGH ใช้แสดงสถานะการทำงานของ CPU โดยร่วมกับ $\overline{\text{M1}}$ และ $\overline{\text{HALT}}$ ดังตาราง :-

ST	$\overline{HALT}$	$\overline{M1}$	OPERATION
0	1	0	CPU operation (1st op-code fetch)
1	1	0	CPU operation (2nd op-code and 3rd op-code fetch)
1	1	1	CPU operation (MC except for op-code fetch)
0	X	1	DMA operation
0	0	0	HALT mode
1	0	1	SLEEP mode (including SYSTEM STOP mode)

$\overline{TEND0}$-$\overline{TEND1}$ TRANSFER END 0 และ 1 เป็นขา OUTPUT ACTIVE LOW ใช้แสดงถึงว่าทำ DMA สิ้นสุดลงแล้ว

TOUT TIMER OUT ใช้กำเนิดพัลส์จาก PRT CHANEL 1

TXA0 , TXA1 TRANSMIT DATA 0 และ 1 เป็นขาส่งข้อมูล SERIAL ของ ASCI

TXS CLOCK SERIAL TRANSMIT DATA เป็นขาส่งข้อมูล

SERIAL ของ CSIO

$\overline{WAIT}$ ขา INPUT ACTIVE LOW จะถูกตรวจที่ขอบขาของ CLOCK ลูกที่ 2 ของทุก ๆ MACHINE เพื่อเป็นการรอให้อุปกรณ์ภายนอก ทำงานให้ทันกับการทำงานของ CPU

WR ใช้สำหรับการส่งข้อมูลไปยัง I/O หรือ MEMORY

X'TAL เป็นขาที่ใช้ต่อกับ X'TAL

ขาที่ MULTIPLEX

A18/TOUT ระหว่าง RESET จะเป็น A18 แต่ถ้ามีการเลือก SET BIT TOC1 หรือ TOC0 ใน TIMER CONTROL REGISTOR (TCR) ก็จะทำหน้าที่เป็น TOUT

CKAO/$\overline{\text{DREQO}}$ ระหว่าง RESET ขานี้จะเป็น CKAO แต่ถ้า DM1 หรือ SM1 ใน DMA MODE REGISTOR (DMODE) ถูก SET เป็น 1 จะเป็นขา DREQO CKA1/TENDOระหว่าง RESET จะเป็นขา CKA1 แต่ถ้า BIT CKA1D ใน ASCI ถูก SET จะเป็นขา $\overline{\text{TENDO}}$

RXS/$\overline{\text{CTS1}}$ ระหว่าง RESET ขานี้จะเป็นขา RXS ถ้า BIT CTS1E ใน ASCI ถูก SET จะเป็นขา $\overline{\text{CTS1}}$

จาก MAP I/O ภายในจะเห็นว่าโปรแกรม Z80 เก่าที่เรามีอยู่อาจจะมีการส่ง PORT ซ้ำกับ I/O ภายใน ทำให้โปรแกรมเดิมทำงานไม่ได้ สามารถแก้ไขได้โดยการโปรแกรมย้าย MAP I/O ภายใน โดยการ CONTROL BIT ใน REGISTOR I/O ICR ADDRESS 3FH ซึ่งสามารถย้ายไปที่ใดก็ได้ภายใน 256 ตำแหน่ง ดังนี้ :-

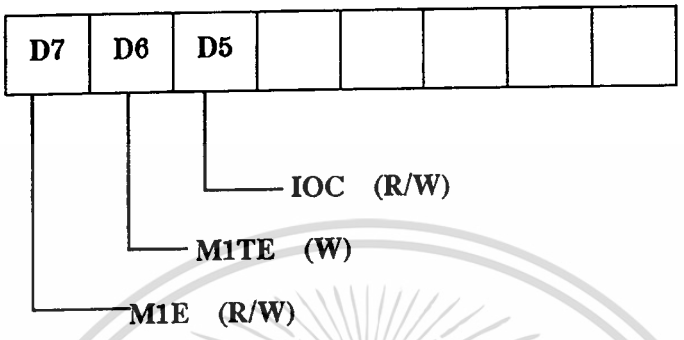
BIT	7	6	5	4	3	2	1	0
	IOA7	IOA6	IOSTP					

และการโปรแกรมจะเป็นดังนี้ :-

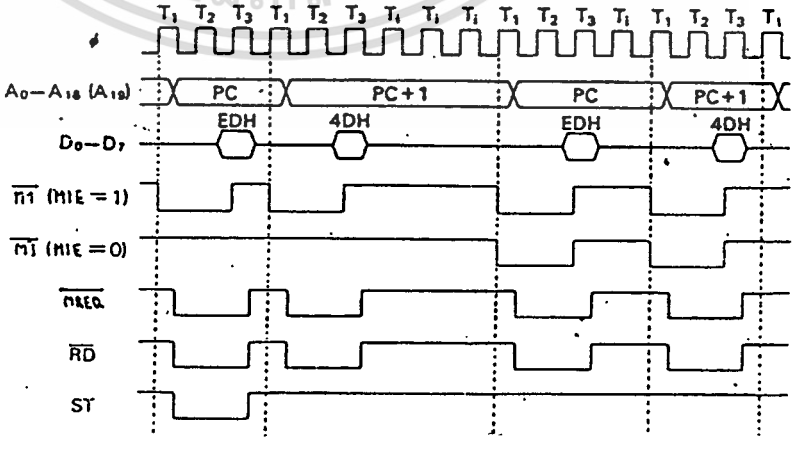
IOA7	IOA6	ช่วง ADDRESS I/O
0	0	0000 - 003FH
0	1	0040 - 007FH
1	0	0080 - 00BFH
1	1	00C0 - 00FFH

OPERATION MODE

Z80180 สามารถกำหนดการทำงานให้เหมือน 64180 ได้ โดยการ SET BIT CONTROL MODE CONTROL REGISTOR (OMCR I/O ADDRESS 3EH)



M1E (M1 ENABLE) ระหว่าง RESET BIT นี้จะเป็น 1 M1 OUTPUT จะเป็น LOW เมื่อ FETCH OPCODE และเนื่องจากการทำคำสั่ง RETI ของ Z80180 จะถูกกระทำ 2 ครั้ง ใน 1 คำสั่ง จึงทำให้เกิด M1 ขึ้น 2 ครั้งด้วยอันอาจจะทำให้เกิด INTERRUPT เข้ามาได้เมื่อยังทำไม่หมดคำสั่ง ด้วยเหตุนี้ BIT M1E จะถูกSET เป็น 0 สำหรับ Z80180 เพื่อให้ M1 ถูกทำงานปกติ คือเมื่อทำคำสั่ง RETI จะมี M1 เพียงครั้งเดียวดังรูป :-

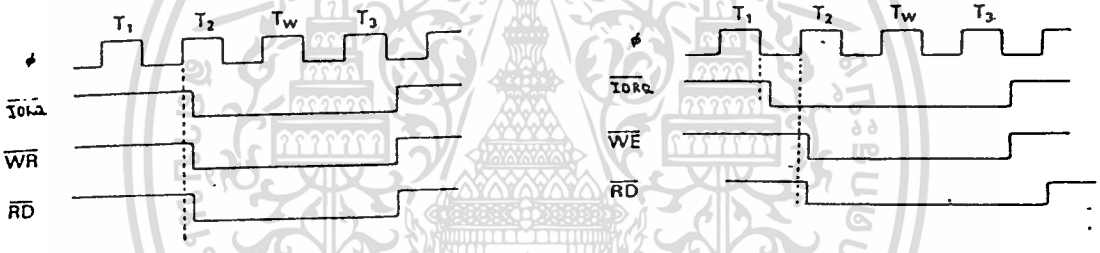


รูปที่ 2.5 รูปแสดงไทมมิ่งไดอะแกรม

M1TE (MA TEMPORARY ENABLE) ใช้กับการต่อ INTERFACE กับ Z80 PIO

IOC เป็น BIT ใช้ควบคุม TIMING ของ IORQ และ RD ให้เหมือน

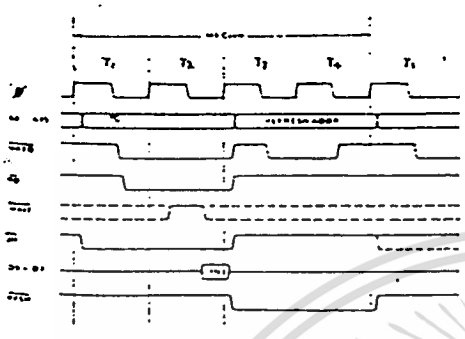
Z80 หรือ 64180 โดยถ้า BIT นี้ถูก SET เป็น 1 TIMING จะเป็นของ 64180 คือ IORQ และ RD จะ ACTIVE ที่ขอบขาลงของ T1 แต่ถ้า BIT นี้เป็น 0 TIMING จะเป็นของ Z80 คือ จะ ACTIVE ที่ขอบขาขึ้นของ T2 เพื่อให้ใช้อุปกรณ์สนับสนุนของ Z80 ได้ ระหว่าง RESET BIT นี้จะเป็น 1 ดังรูป :-



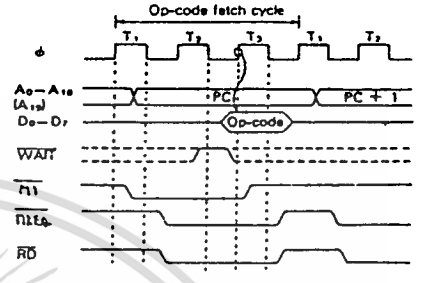
รูปที่ 2.6 I/ORead WriteCycle WhenIOC=0 **รูปที่**2.7I/O Read Write Cycle When IOC = 1

เกี่ยวกับ TIMING

ให้ดูรายละเอียดในคู่มือฉบับภาษาอังกฤษ แต่กล่าวสรุปได้ว่า Z80180 ใช้เวลาในการทำคำสั่งใน 1 MACHINE CYCLE น้อยกว่า Z80 อยู่ 1 T STATE คือใช้เวลาใน 1 MACHINE CYCLE เพียง 3 T STATE ในขณะที่ Z80 ใช้ 4 T STATE จะเห็นได้ว่าในขณะที่ให้ Z80180 RUN ความถี่เดียวกันกับ Z80 CPU Z80180 ก็ยังให้ความเร็วกว่า Z80 ถึงอีก 25% แต่ในขณะเดียวกัน Z80180 ยังสามารถต่อ CLOCK สูงกว่า Z80 ได้มากกว่า 1 เท่า จึงทำให้ความเร็วในการทำงานของ Z80180 ดีกว่ามาก ดูรูปเปรียบเทียบ T STATE ของ Z80 กับ Z80180



รูปที่ 2.8 รูปแสดง T State ของ Z80



รูปที่ 2.9 รูปแสดง T State ของ Z80180

WAIT STATE GENERATOR

Z80180 ทำงานด้วยความถี่ที่สูงขึ้นจึงอาจทำให้ MEMORY หรือ I/O ทำงานไม่ทัน จึงต้องมีสัญญาณมาเป็นตัวช่วยกำหนดความพร้อมระหว่าง CPU กับอุปกรณ์ภายนอกนั่นก็คือ สัญญาณ WAIT ซึ่ง Z80 นั้นจะต้องให้อุปกรณ์ภายนอกส่งสัญญาณนี้มาให้แต่ Z80180 ยังสามารถให้โปรแกรมจำนวน WAIT STATE เพื่อเพิ่มเข้าไปในขณะที่ CPU ปฏิบัติคำสั่งหรือทำ DMA ด้วย

การโปรแกรมจะใช้ 4 BIT ของ DMA/WAIT CONTROL REGISTOR (DCNTL I/O ADDRESS 32H)

BIT	7	6	5	4
	MWI1	IWI0	MWI1	IWI0

BIT 7 , 6 MWI1 , MWI0 (MEMORY WAIT INSERTION)

จะทำการเพิ่มจาก 0-3 WAIT STATE ของการเข้าถึง MEMORY โดยการโปรแกรม

NWI1	MWI0	จำนวน WAIT STATE
0	0	0
0	1	1
1	0	2
1	1	3

BIT 5 , 4 IWI1 , IWI0 (I/O WAIT INSERTION)

จะทำการเพิ่ม WAIT STATE ให้กับ I/O ภายนอกจาก 1-6 ดังตาราง

IWI1	IWI0	I/O ภายนอก	INTO
0	0	1	2
0	1	2	4
1	0	3	5
1	1	4	6

จะเห็นว่า WAIT STATE ของ I/O มากกว่า MEMORY อยู่หนึ่ง T STATE เพราะขณะเข้าถึง I/O ปกติ WAIT STATE จะถูกเพิ่มขึ้น 1 อยู่แล้ว ดังนั้นเมื่อเพิ่ม WAIT STATE เข้าไปก็จะรวมกับที่มีอยู่ปกติ และส่วน INTO ก็เช่นเดียวกัน ขณะเกิด INTO ปกติจะมี WAIT STATE อยู่ 2 WAIT STATE อยู่แล้ว และขณะที่ RESET BIT CONTROL WAIT STATE ทั้ง 4 จะเป็น 1 ทั้งหมด คือ อยู่ใน MODE ของ MAX WAI T STATE

HALT และ LOW POWER MODE

มีด้วยกัน 4 MODE คือ

HALT MODE — โดยทำคำสั่ง 76H จะทำให้ CPU หยุดทำคำสั่ง แต่การทำงานต่าง ๆ ของ CPU ยังทำปกติ การออกจาก HALT โดย RESET หรือ INTERRUPT

SLEEP MODE — โดยการทำคำสั่ง SLP ซึ่ง CPU จะหยุด CLOCK ภายใน ทำให้ ADDRESS เป็น HIGH , DATA BUS เป็น TRISTATE , DRAM REFRESH INTERNAL DMAC หยุดทำงานการออกจาก SLEEP MODE โดยการ RESET หรือ INTERRUPT

IOSTOP MODE — ใช้หยุดการทำงานของ CHIP ภายใน คือ ASCI , CSI/O และ PRT โดยการ SET BIT ใน I/O

CONTROL REGISTOR (ICR I/O ADDRESS 3FH) เป็น 1 และจะทำให้ทำงานต่อก็ RESET หรือโปรแกรมให้ BIT ใน ICR เป็น 0

SYSTEM STOP MODE เป็นการรวมกันของ IOSTOP กับ SLEEP MODE โดยการ SET BIT ใน ICR แล้วตามด้วยคำสั่ง SLP จะทำให้ IO ภายในหยุดทำงานและ CPU หยุดทำงานเพื่อเป็นการประหยัดพลังงานซึ่งใน MODE นี้ CPU จะกินกระแสเพียง 7.5 MA ในขณะปกติจะกินกระแสประมาณ 35 MA เมื่อจะออกจาก SYSTEM STOP MODE ก็โดยการ RESET หรือ INTERRUPT จากภายนอก

REGISTOR CONTROL

CBAR : COMMON/BANK AREA REGISTOR (I/O ADDRESS 3AH) ใช้กำหนดพื้นที่ของ LOGICAL ที่เป็น COMMON AREA 0 , BANK AREA และ COMMON AREA 1

BIT	7	6	5	4	3	2	1	0
	CA 3	CA 2	CA 1	CA 0	BA 3	BA 2	BA 1	BA 0

CA 3 - CA 0 เป็นตั๋วกำหนด ADDRESS เริ่มต้นของ COMMON AREA 1

BA 3 - BA 0 เป็นตัวกำหนดจุด ADDRESS สุดท้ายของพื้นที่ BANKAREA ที่ต่อจากจุดเริ่มต้นของ COMMON AREA A

CBR : COMMON BASE REGISTOR (I/O ADDRESS 38H) เป็น REGISTOR I/O 8 BIT

BBR : BANK BASE REGISTOR (I/O ADDRESS 39H) ใช้กำเนิด PHYSICAL BANK AREA ดังนั้นเราอาจจะกล่าวได้ว่าใน LOGICAL ส่วนมากจะถูกจัดเป็น

COMMON AREA 1 STACK
BANK AREA USER PROGRAM
COMMON AREA 0 MONITOR PROGRAM

โดยให้ส่วนของ AREA 1 และ MONITOR คงที่ ส่วนของ BANK ให้ย้ายไปที่ใด ๆ ก็ได้ใน 1 MBYTE จะเป็นการขยายพื้นที่ของการใช้งาน โดยใช้เนื้อที่ของ STACK เป็นการกระทำกับตัวแปรหรือ DATA อื่นใน 64K อื่น ๆ เมื่อเรามอง LOGICAL 64K ออกเป็น PAGE ๆ ใน 1 MBYTE แต่ข้อเสียในการจัดแบบนี้ จะทำให้ใช้ BANK AREA ได้ไม่เต็มที่ เช่น เราต้องการใช้ RAM ถึง 32K เต็ม เช่น ให้ ROM MONITOR อยู่ที่ 0000-7FFFH และ RAM เริ่มแต่ 8000-FFFFH ซึ่งจะเห็นว่า RAM ในส่วนนี้จะต้องเป็น STACK ด้วยเมื่อเรายึด BANK ออกไปที่ PHYSICAL อื่นก็จะไม่สามารถใช้ได้ถึง 32K เช่นมี RAM ที่ตำแหน่ง 18000-1FFFFH อีกเราจะใช้ได้แค่ 24K เพราะพอเราอ้างที่ 0F000 แทนที่ข้อมูลจะถูกกระทำที่ 1F000H ก็จะมากระทำที่ 0F000H แทนตามที่กำหนด AREA 1 ไว้ใน LOGICAL เราจึงอาจแบ่ง MAP เป็นลักษณะกว้าง ๆ ดังนี้ :-

USER PROGRAM (AREA 1)
SYSTEM (AREA 0)

CBAR = 80 H

โดยกำหนดให้ AREA 1 เป็นส่วน USER PROGRAM ส่วน SYSTEM เป็นของ COMMON AREA 0 ดังนั้น เมื่อเราให้ AREA 1 เริ่มที่ 8000H ก็จะใช้ RAM ได้ถึง 32K เต็ม ส่วน SYSTEM ก็คือ ของ AREA 0 ซึ่งเป็น ส่วน MONITOR แต่ในส่วนนี้เรา

ได้กำหนดไว้ถึง 32K คือ จาก 7FFF ลงไปถึง 0000H ซึ่งใน SYSTEM เราอาจจะใส่ RAM ไว้ใน ADDRESS ช่วงนี้เพื่อเป็นเนื้อที่ของ STACK ก็จะทำให้เราย้ายเนื้อที่ของการใช้งานได้เต็ม

สรุป

1) ระหว่าง RESET LOGICAL ใน CBAR จะถูกกำหนดด้วยค่า 0F0H
2) ให้กำหนด MAP ADDRESS ของ LOGICAL ก่อนที่ CBAR (3AH)
3) BBR และ CBR จะเป็นตัวกำหนดตำแหน่งของข้อมูลในการใช้งานจริงในพื้นที่ 1 MBYTE (PHYSICAL ADDRESS)
4) การคิดค่า PHYSICAL ADDRESS คือ นำค่าใน BBR หรือ CBR คูณด้วย 1000H แล้วบวกด้วย LOGICAL ของพื้นที่นั้น ๆ

INTERRUPT

มีด้วยกัน 12 INTERRUPT แบ่งเป็น 4 INTERRUPT ภายนอกและ8 INTERRUPT ภายใน โดยมีลำดับความสำคัญจากมากไปหาน้อย ดังนี้ TRAP (ภายใน) , (ภายนอก) NMI , INT0 , INT1 , INT2 , (ภายใน) TIMER 0,TIMER 1, DMA CHANEL 0, DMACHANEL 1, CLOCK SERIAL , ASCI CHANEL 0 และ ASCI CHANEL 1

REGISTOR และ FLAG ที่ใช้ควบคุมการ INTERRUPT

INTERRUPT VECTOR LOW (IL) , INTERRUPT VECTOR HIGHT (I) , INTERRUPT TRAP CONTROL (ITC) และ FLAG IEF1 , IEF2 โดยที่ FLAG IEF1 จะใช้ในการ ENABLE INTERRUPT ภายในทั้งหมดยกเว้น TRAP INTERRUPT

VECTOR LOW REGISTOR (IL I/O ADDRESS 33H)

ใช้เป็น VECTOR TABLE BYTE ต่ำ ของ INTERRUPT ภายนอก INT1 , INT2 และ INTERRUPT ภายในทั้งหมดยกเว้น " TRAP " โดย 3 BIT สูงของ IL สามารถโปรแกรมได้ แต่ 5 BIT หลัง จะถูก FIX ดังรูป :-

Interrupt Source	Priority	IL			Fixed Code				
		b7	b6	b5	b4	b3	b2	b1	b0
INT 1	Highest	.	.	.	0	0	0	0	0
INT 2		.	.	.	0	0	0	1	0
PRT channel 0		.	.	.	0	0	1	0	0
PRT channel 1		.	.	.	0	0	1	1	0
DMA channel 0		.	.	.	0	1	0	0	0
DMA channel 1		.	.	.	0	1	0	1	0
CSI/O		.	.	.	0	1	1	0	0
ASCI channel 0		.	.	.	0	1	1	1	0
ASCI channel 1	Lowest	.	.	.	1	0	0	0	0

. Programmable

ดังนั้นการ INTERRUPT ส่วนใหญ่จะเป็น MODE 2 คือนำค่าใน I และ IL หรือจากอุปกรณ์ที่ของ INTERRUPT ในกรณี INTO มาประกอบกันเป็น ADDDRESS ที่จะเก็บข้อมูลที่จะกระโดดไป เช่น I = 10 H และ IL = 40 H และใน ADDRESS 1040H มีข้อมูลที่ 00H, 60H ตามลำดับ เมื่อเกิด INTERRUPT ขึ้นก็จะกระโดดไปทำโปรแกรมที่ตำแหน่ง 6000H นั่นเอง

DYNAMIC RAM REFRESH CONTROL

Z80180 ให้ ADDRESS A0-A7 สำหรับ DYNAMIC RAM และยังสามารถโปแกรมเวลาในการ REFRESH โดยการโปรแกรมที่ RCR

REFRESH CONTROL REGISTOR (RCR ADDRESS I/O 36 H)

BIT	7	6	5	4	3	2	1	0
	REFE	REFW	-	-	-	-	CYC1	CYD0

REFE : REFRESH ENABLE เมื่อเป็น 0 จะ DISABLE แต่ถ้าเป็น 1 จะให้สัญญาณ REFRESH ระหว่าง RESET จะเป็น 1

REFW : REFRESH WAIT เป็น 0 จะให้สัญญาณ REFRESH ทุก ๆ 2 CLOCK ถ้าเป็น 1

CYD1 , CYC0 : CYCLE INTERVAL ใช้กำหนดช่วงเวลาในการ REFRESH
เช่น กรณี DYNAMIC RAM จะต้อง REFRESH 128 ครั้ง ทุก ๆ 2
ms (หรือ 256 ครั้ง ทุก ๆ 4 ms) เพราะฉะนั้นสัญญาณ REFRESH
แต่ ละครั้งจะต้องไม่น้อยกว่า หรือ เท่ากับ 15.625 us จากตาราง ค่าที่
ขีดเส้นใต้เป็นค่าโปรแกรมที่เหมาะสมกับ CLOCK ที่ใช้ในระบบ

CYC1	CYC0	Insertion interval	Time interval			
			: 8 MHz	6 MHz	4 MHz	2.5 MHz
0	0	10 states	1.25 us	1.66 us	2.5 us	4.0 us
0	1	20 states	2.5 us	3.3 us	5.0 us	8.0 us
1	0	40 states	5.0 us	6.6 us	10.0 us	16.0 us
1	1	80 states	10.0 us	13.3 us	20.0 us	32.0 us

DMA CONTROLLER (DMAC)

มีด้วยกัน 2 CHANEL เพื่อเป็นการเพิ่มความเร็วในการ TRANFER ข้อมูล โดยการกระทำไม่ต้องผ่าน CPU โดยมีความสามารถดังนี้

MEMORY ADDRESS SPACE โดยสามารถกำหนดตำแหน่ง SOURCE และ DESTINATION ที่ใดก็ได้ใน 1024 K BYTE

I/O ADDRESS STACE กำหนดที่ใดก็ได้ใน 64 KBYTE ทั้งSOURCE และ DESTINATION

TRANSFER LENGTH ใช้เป็น COUNTER ในการ TRANSFER ได้เป็น BLOCK ๆ ละ 64 K BYTE

$\overline{\text{DREQ}}$ เป็นขา INPUT จะตรวจจับที่ระดับ หรือขอบของสัญญาณ

$\overline{\text{TEND}}$ เป็นขา OUTPUT เพื่อบอกกับอุปกรณ์ภายนอกว่าทำ DMA หมด BLOCK แล้ว

TRANSFER RATE การ TRANSFER แต่ละครั้งจะเกิดทุก ๆ 6 CLOCK และ WAIT STATE สามารถเพิ่ม

เข้าไปใน DMA (O) = 6 Mhz อัตราการ TRANSFER จะสูงถึง 1 M BYTE ใน 1 วินาที (ไม่มี WAIT STATE)

ความสามารถของแต่ละ CHANEL

CHANEL 0 สามารถ TRANSFER MEMORY←→MEMORY, MEMORY←→I/O MEMORY←→MEMORY I/O MAP และสามารถให้ ADDRESS ให้ ADDRESS ในการ TRANSFER เพิ่ม , ลด หรือให้คงที่ได้ การ TRANSFER จะเป็นแบบ CYCLE STEAL (ขโมยเวลาเป็นช่วงได้) คือ เมื่อ TRANSFER ครบ 1 หรือ 2 BYTE ก็จะคืน BUS ให้ UP จนอุปกรณ์พร้อมก็จะ TRANSFER ข้อมูลต่อ ทำอย่างนี้สลับกันไปจนหมด BLOCK ใช้สำหรับอุปกรณ์ที่ทำงานช้า และ BURST (TRANSFER แบบต่อเนื่อง) คือ ทำจนจบ BLOCK จึงจะคืน BUS ให้ UP

CHANEL 1 จะใช้กับ MEMORY←→I/O โดย MEMORY ADDRESS เพิ่มหรือลดได้

ASYNCRONOUS SERIAL COMMUNICATION INTERFACE (ASCI) มีด้วยกัน 2 CHANEL

TSR 0 , 1 เป็น SHIFT REGISTOR ที่รับข้อมูลจาก TRANSMIT DATA REGISTOR (TDR) แล้วนำข้อมูลนั้น SHIFT ออกที่ขา TXA

TDR 0 , 1 (I/O ADDRESS 06H , 07H) เป็น REGISTOR ที่ใช้ส่ง DATA ออกไปที่ขา TXA โดยการนำข้อมูลใน TDR ส่งไปที่ TSR เมื่อ TSR ว่างลง และสามารถที่จะเขียนข้อมูลเข้าไปที่

TDR ได้อีกในขณะที่ TSR กำลัง SHIFT ข้อมูลออกไปที่ขา TXA

RSR 0 , 1 เป็น REGISTOR ที่รับข้อมูลจาก RXA PIN เมื่อรับเต็ม BUFFER แล้ว ก็จะ SHIFT ไปที่ RDR ถ้า RSR ไม่ว่าง เมื่อมีการรับข้อมูล BYTE ต่อไปเข้าอีก จะเกิดข้อมูลทับซ้อนขึ้น จะทำให้เกิดการผิดพลาด และผลของการผิดพลาดก็จะแสดงที่ REGISTOR สถานะ REGISTOR นี้ไม่สามารถโปรแกรมได้

RDR 0 , 1 (I/O ADDRESS 08H , 09H)คือ REGISTOR ที่ใช้เก็บข้อมูลที่รับเข้ามาจากRXA PIN และในขณะที่ RDR กำลังบรรจุข้อมูลที่รับเข้ามาจาก RSR ข้อมูล BYTE ถัดไปสามารถรับเข้ามาต่อได้

STAT 0 , 1(I/O ADDRESS 04H , 05H) แต่ละ CHANEL จะมี REGISTOR ใช้สำหรับตรวจสอบการสื่อสารเกี่ยวกับการผิดพลาด และสถานะสัญญาณ CONTROL MODEM การ ENABLE และ DISABLE ASCI ดังรูป :-

BIT	7	6	5	4	3	2	1	0
STAT 0	RDRF	OVRN	PE	FE	RIE	DCDO	TDRE	TIE

BIT	7	6	5	4	3	2	1	0
STAT 1	RDRF	OVRN	PE	FE	RIE	CTSIE	TDRE	TIE

RDRF ; RECEIVE DATA REGISTOR FULL จะถูก SET เป็น 1 เมื่อข้อมูลที่รับเข้ามา ถูกส่งเข้ามาที่ RDR เรียบร้อยแล้ว (ครบ BYTE) แต่ถ้าการรับเกิด ERROR ขึ้น RDRF ก็จะถูก SET ค้าง และข้อมูลที่ผิดนั้นก็จะถูกส่งมาที่ RDR และคงอยู่ ดังนั้นจะต้องทำการ CLEAR FLAG ERROR

RDRF จะถูก CLEAR เป็น 0 เมื่ออ่าน RDR , DCD0 เป็น HIGHT สำหรับ CHANEL 0 , IOSTOP และการ RESET

OVRN : OVERUN ERROR จะเป็น 1 เมื่อ RDR เต็มและ RSR เต็มแล้วยังมีการรับข้อมูลเข้ามาอีกจะถูก CLEAR ได้ เมื่อ EFR BIT ใน CNTLA เป็น 0, DCDO เป็น HIGHT IOSTOPและ RESET

PE : PARITY ERROR เป็น 1 เมื่อข้อมูลที่รับเข้ามา PARITY ผิด และCLEAR ได้ เช่นเดียวกับ OVERUN

FE : FRAMING ERROR เมื่อข้อมูลที่รับเข้ามารูปแบบผิดไปจากที่กำหนด BIT FE จะถูก SET เป็น 1 และการ CLEAR เช่นเดียวกับ OVERUN

RIE : RECEIVE INTERRUPT ENABLE เมื่อเป็น 1 จะอนุญาตให้ ASCI ทำการขอ INTERRUPT ได้ เมื่อ RDRF , OVRN , PE หรือ FE ถูก SET เป็น 1 ด้วยเมื่อนั้น ASCI ก็จะให้สัญญาณ INTERRUPT สำหรับ ASCI HANEL 0 INTERRUPT สามารถเกิดขึ้นโดยการเปลี่ยนแปลงที่ขารับสัญญาณ INPUT ภายนอกที่ขา DCD0 จาก LOW เป็น HIGHT และ RIE จะถูก CLEAR เป็น 0 ระหว่าง RESET

$\overline{\text{DCD0}}$: DATA CARRIER DETECT BIT นี้จะถูก SET เป็น 1 เมื่อขา INPUT $\overline{\text{DCD0}}$ เป็น HIGH และจะถูก CLEAR เป็น 0 จากการอ่าน STAT 0 ครั้งแรก จากนั้นขา INPUT $\overline{\text{DCD0}}$ จะถูกเปลี่ยนจาก HIGHT เป็น LOWและระหว่าง RESET เมื่อ $\overline{\text{DCD0}}$ เป็น 1 ส่วนของภาครับจะไม่ทำงาน

CTSIE : CHANEL 1 CTS ENABLE ที่ CHANEL 1 มีขา INPUT $\overline{\text{CTS1}}$ ภายนอก ซึ่ง MULTIPLEX กับ RXS เมื่อ SET BIT นี้เป็น 1จะถูกเลือกเป็นขา $\overline{\text{CTS1}}$

TDRE : TRANSMIT DATA REGISTOR EMPTY เป็นตัวบอกว่าข้อมูลพร้อมที่จะส่งได้หรือไม่ ถ้าเป็น 1 คือ พร้อมที่จะส่งข้อมูลให้เขียนข้อมูลเข้าไปที่ TDR ได้ และเมื่อมีการเขียนข้อมูลเข้าไปที่ TDR ก็จะทำให้ TDRE เป็น 0 และข้อมูลใน TDR ก็จะถูกส่งให้ TSR จน TDR ว่างลง TDRE ก็จะกลับเป็น 1 อีกครั้ง

TIE : TRANSMIT INTERRUPT ENABLE เมื่อเป็น 1 จะอนุญาตให้ ASCI ใช้การส่งแบบ INTERRUPT ได้ โดยที่ TDRE ต้องเป็น 1 ด้วย TIE จะถูก CLEAR เป็น 0 ระหว่าง RESETCNTLA 0 , 1 (I/O ADDRESS 00H-01H) เป็น REGISTOR กำหนดการทำงานประกอบด้วย

BIT	7	6	5	4	3	2	1	0
CNTLA 0	MPE	RE	TE	RTS0	MPBR/ EFR	MOD2	MOD1	MOD0

BIT	7	6	5	4	3	2	1	0
CNTLA 0	MPE	RE	TE	CKAID	MPBR/ EFR	MOD2	MOD1	MOD0

MPE : MULTIMPROCESSOR MODE ENABLE ใช้ ENABLE ในการสื่อสารแบบไมโครโปรเซสเซอร์ร่วมจากเมื่อมีการเลือก MODE การสื่อสารแล้ว (MP = 1 ใน CNTLB) ในการสื่อสารแบบนี้ FORMAT ของการการสื่อสารจะมี BIT พิเศษเพิ่มเข้ามาเรียกว่า MPB BIT ซึ่งBIT นี้จะถูกใช้ในการตรวจสอบหรือใช้งาน เมื่อ ENABLE MPE ให้เป็น 1 และถ้า MPB = 1เมื่อนั้น ภาครับของ MULTIPROCESSOR จะทำงาน คือ RDRF และ ERROR FLAG จะทำงานแต่ถ้า MBP = 0 ASCI จะไม่สนใจข้อมูล BYTE นั้น ถ้า MPE = 0 จะไม่สามารถทำการสื่อสารแบบไมโครโปรเซสเซอร์ร่วมได้ แม้จะ SET MP เป็น 1 แล้วก็ตาม

RE : RECEIVER ENABLE ถ้าเป็น 1 จะ ENABLE การรับของ ASCI แต่ถ้าเป็น 0 จะ DISABLE การรับ แต่ RDRF และ ERROR FLAG จะไม่ถูก RESET ตาม

TE : TRANSMIT ENABLE เป็น 1 จะ ENABLE การส่ง ถ้าเป็น 0 จะ DISABLE แต่ TDRE FLAG จะไม่ถูก RESET ตาม

$\overline{\text{RTSO}}$: REQUEST TO SEND CHANEL 0 เป็น BIT ที่ให้ผลเช่นเดียวกับขา OUTPUT $\overline{\text{RTS0}}$ คือ ถ้า BIT นี้เป็น 1 ขา OUTPUT $\overline{\text{RTS0}}$ ก็จะเป็น 1

ถ้า BIT นี้เป็น 0 ขา OUTPUT ก็เป็น 0 RTSO BIT นี้ จะถูก SET เป็น 1 ระหว่าง RESET

CKA1D: CKA1 CLOCK DISABLE ซึ่งขา CKA1 จะ MULTIPLEX กับ $\overline{\text{TEND0}}$ เมื่อ BIT นี้เป็น 1 จะเลือกเป็นขา $\overline{\text{TEND0}}$ แต่ถ้าเป็น 0 ก็จะเป็นขา CLOCK ของASCI CHANEL 1 BIT นี้จะเป็น 0 ระหว่าง RESET

MPBR/EFR MULTIPROCESSOR BIT RECEIVE / ERROR FLAG เมื่อ BIT นี้ถูกอ่าน จะใช้ดู MPB BIT ในกรณีที่พบไมโครโปรเซสเซอร์ที่จะทำการติดต่อกันแล้ว และจะ DISABLE ไมโครโปรเซสเซอร์ตัวอื่น ๆ ก็โดยการส่ง MPB BIT ให้เป็น 0 เมื่ออ่านจะได้รู้ว่า MPB เป็น 0 จริง แต่ถ้าเขียน 0 ให้ BIT นี้จะเป็นการ RESET ERROR FLAG ในการรับ

MOD 2 , 1 , 0 : ASCI DATA FORMAT MODE 2 , 1 , 0 โดย

MOD 2 = 0 = 7 BIT , 1 = 8 BIT MOD1 = 0 NOPARITY ,

1 = PARITY ENABLE

MOD 0 = 0 1 STOP BIT , 1 = 2 STOP BIT สรุปได้ดังตาราง

MOD 2	MOD 1	MOD 0	DATA FORMAT
0	0	0	START + 7 BIT DATA + 1 STOP
0	0	1	START + 7 BIT DATA + 2 STOP
0	1	0	START + 7 BIT DATA + PARITY + 1 STOP
0	1	1	START + 7 BIT DATA + PARITY + 2 STOP
1	0	0	START + 8 BIT DATA + 1 STOP
1	0	1	START + 8 BIT DATA + 2 STOP
1	1	0	START + 8 BIT DATA + PARITY + 1 STOP
1	1	1	START + 8 BIT DATA + PARITY + 2 STOP

ASCI CONTROL REGISTOR B 0 , 1 (CNTLE 0 , 1 I/O ADDRESS 02H , 03H) ประกอบด้วย

BIT	7	6	5	4	3	2	1	0
	MPBT	MP	CTS/PS	PEO	DR	SS2	SS1	SS0

MPBT : MULTIPROCESSOR BIT TRANSMIT ใช้ส่ง MPB BIT โดยถ้า MPBT = 1 เมื่อนั้น MPB BIT = 1 และ MPBT = 0 MPB ก็ = 0 ด้วย ระหว่าง RESET ไม่สามารถกำหนดได้

MP : MULTIPROCESSOR MODE ถ้าเป็น 1 จะเป็นการ SET การติดต่อแบบไมโครโปรเซสเซอร์ร่วม โดยใช้ FORMAT ของ MOD 2 กับ MOD 0 โดยยกเว้น MOD 1 ดังนี้

START BIT + 7 หรือ 8 DATA BIT + MPB BIT + 1 หรือ 2 STOP BIT ระหว่าง RESET MP จะเป็น 0

$\overline{\text{CTS}}$/PS: CLEAR TO SEND / PRESCALE เมื่ออ่าน BIT นี้จะใช้แสดงสถานะของขา INPUT $\overline{\text{CTS}}$ ภายนอก ถ้าขา $\overline{\text{CTS}}$ เป็น HIGH ภาคส่งของ ASCI จะไม่ทำงาน แต่ถ้าเขียนเข้าไปที่ BIT นี้จะเป็นการกำหนด BAUD RATE BIT นี้เป็น 0 ระหว่าง RESET

PEO : PARITY EVEN ODD BIT นี้จะไม่มีผลต่อการ ENABLEหรือDISABLE ของ PARTITY (MOD 1 ฯฯ CNTLA) แต่จะใช้เลือกว่าเมื่อมีการ ENABLE PARITY ใน MOD 1 จะให้ PARITY คู่หรือคี่ ถ้า PEO = 0 คือ คู่ แต่ถ่า = 1 คือ คี่

DR : DIVIDE RATIO ใช้กำหนด BAND RATE BIT นี้จะเป็น 0 ระหว่าง RESET

SS2 , 1 , 0 : SOURCE / SPEED SELECT 2 , 1 , 0 ใช้กำหนด CLOCK ว่าจะให้เป็นภายใน หรือภายนอก (โดยภายนอก คือ ขา CLOCK CKA) และเป็นตัวกำหนด BAUD RATE ด้วย ระหว่าง RESET ทั้ง 3 BIT นี้จะเป็น 1 คือเป็นการใช้ CLOCK จากภายนอกนั่นเอง ซึ่งจากที่กล่าวมาในการกำหนด BAUD RATE จึงมีด้วยกันหลายตัว

CLOCK SERIAL I/O PORT (CSI/O)

มี 1 CHANEL ซึ่งเป็น SYNCHRONOUR SERIAL I/O PORT โดยใช้ได้เฉพาะเป็น HALF-DUPLEX เท่านั้น และ DATA ถูกำหนดเป็น 8 BIT โดย CLOCK ที่ใช้ในการซิงค์เลือกได้ว่าจะใช้จาก SYSTEM CLOCK หรือ CLOCK ภายนอกที่ขา (CKS) ก็ได้ ซึ่ง CSI/O ประกอบไปด้วย 2 REGISTOR คือ :-

CSI/O TRANSMIT / RECEIVE DATA REGISTOR (TRDR I/O ADDRESS 0BH) ใช้ในการส่งและรับข้อมูล โดยระบบต้องเป็น HALF-DUPLEX (คือการส่งและรับจะเกิดพร้อมกันไม่ได้)

CSI/O CONTROL / STATUS REGISTOR (CNTR I/O DDRESS0AH) เป็นตัวบอกสถานะและ CONTROL CSI/O ประกอบด้วย

BIT	7	6	5	4	3	2	1	0
	EF	EIE	RE	TE	-	SS2	SS1	SS0

EF : END FLAG เป็น 1 เมื่อ CSI/O รับหรือส่งข้อมูลครบ 8 BIT แล้ว ซึ่งถ้า EIE ถูก SET เป็น 1 ไว้ ก็จะทำให้ CSI/O ขอ INTERRUPT ได้ ระหว่าง RESET BIT นี้จะเป็น 0 และ ใน IOSTOP MODE ด้วย

EIE : END INTERRUPT ENABLE เมื่อเป็น 1 จะเป็นการ ENABLE INTERRUPT และจะ INTERRUPT เมื่อ EF = 1 ด้วย ระหว่าง RESET EIE = 0

RE : RECEIVE ENABLE ภาครับจะทำงานเมื่อ RE = 1 โดยข้อมูลจากขา RSX จะถูก SHIFT เข้ามาที่ TRDR โดยข้อมูลที่เข้าทาง RXS จะซิงค์กับสัญญาณ CLOCK ซึ่งจะเป็นภายในหรือ ภายนอก โดยการเลือก ถ้าเป็นภายในสัญญาณ CLOCK ที่ทำการหารแล้วก็จะออกที่ขา CKS ด้วย หรือถ้าเป็นภายนอก ขา CKS ก็จะเป็นตัวรับสัญญาณ CLOCK เพื่อใช้ซิงค์นั่นเอง หลังจากที่ CSI/O รับข้อมูลครบ 8 BIT แล้วก็จะ CLEAR RE โดยอัตโนมัติ และ RE กับ TE จะต้องไม่เป็น 1 พร้อมกัน

TE : TRANSMIT ENABLE ลักษณะการทำงานเช่นเดียวกับ RE แต่ TE นี้จะใช้ในการส่ง

SS2 , 1 , 0 : SPEED SELECT 2 , 1 , 0 ใช้เลือก CLOCK ในการรับส่งดังรูป :-

SS2	SS1	SS0	DIVIDE RATIO	BAUD RATE
0	0	0	- 20	(200000)
0	0	1	- 40	(100000)
0	1	0	- 80	(50000)
0	1	1	- 160	(25000)
1	0	0	- 320	(12500)
1	0	1	- 640	(6250)
1	1	0	- 1280	(3125)
1	1	1	EXTENAL CLOCK INPUT (LESS THAN -20)	

( ) BAUD RATE ที่แสดง ที่ 0 = 4 Mhz

หลังจาก RESET CKS PIN จะถูกเลือกเป็นขา INPUT เพราะ BIT SS2 , 1 , 0 เป็น 1

PROGRAMMABLE RELOAD TIMER (PRT)

มีด้วยกัน 2 CHANEL เป็น 16 BIT PROGRAMMABLE RELOAD TIMER และสำหรับ CHANEAL 1 มีขา OUTPUT สามารถให้สัญญาณได้ทั้ง 2 CHANEL ประกอบด้วย :-

TIMER DATA REGISTOR (TMDR : I/O ADDRESS - CHO ; ODH , OCH , CH1 ; 15H , 14H) เป็น REGISTOR 16 BIT ใช้กำหนด TIMER โดย ADDRESS I/O สูงเก็บค่า TIMER ค่าสูงระหว่าง RESET TMDRO และ TMDR1 จะเป็น 0FFFFH โดย TMDR จะนับลง 1 ครั้งทุก ๆ 20 CLOCK SYSTEM เมื่อ TMDR นับลงเป็น 0 ค่าใน RELOAD จะถูก LOAD มาให้ TMDR โดยอัตโนมัติ การอ่านใน TMDR อ่านได้เลยโดยไม่ต้องหยุด PRT แต่ถ้าเป็นการเขียนต้องหยุด PRT ก่อน

TIMER RELOAD REGISTOR (RLDR : I/O ADDRESS - CHO ; 0FH , 0EH , CH1 ; 17H , 16H) ใช้ LOAD ค่าที่อยู่ใน RLDR ไปให้ TMDR เมื่อ TMDR ลดลงเป็น 0

TIMER CONTROL REGISTIOR (TCR : I/O ADDRESS 10H) เป็น REGISTOR ใช้แสดงสถานะและ CONTROL ดังนี้ :-

BIT	7	6	5	4	3	2	1	0
	TIF 1	TIF 0	TIE 1	TIE 0	TOC 1	TOC 0	TDE 1	TDE 0

TIE : TIMER INTERRUPT FLAG 1 เมื่อ TMDR 1 ลดลงเป็น 0 TIF จะถูก SET เป็น 1 และ ถ้า TIE 1 = 1 ก็จะทำให้เกิด INTERRUPT ขึ้นได้ TIF จะถูก CLEAR เป็น 0 ก็ต่อเมื่อทำ การอ่านค่าใน TCR กับอ่านค่าใน BYTE HIGH หรือ LOW ของ LOW ของ TMDR 1 ระหว่าง RESET TIF = 0

TIE 0 : TIMER INTERRUPT FLAG หลักการเช่นเดียวกับ TIF 1

TIF 1 , 0: TIMER INTERRUPT ENABLE 0 , 1 เมื่อ SET เป็น 1 จะอนุญาตให้ INTERRUPT ได้ ระหว่าง RESET 2 BIT นี้จะเป็น 0

TOC 1 , 0: TIMER OUTPUT CONTROL 2 BIT นี้ใช้ควบคุมขา OUTPUT ของ PRT 1 โดยถ้าทั้ง 2 BIT นี้เป็น 0 จะเป็นการใช้งาน A18 นอกนั้นจะเป็นการกำหนดให้ TOUT เป็น HIGH , LOW หรือ TOGGLE ดังตาราง ระหว่าง RESET 2 BIT นี้จะเป็น 0

TOC 1	TOC 0	OUTPUT
0	0	ADRESS A18
0	1	TOGGLE
1	0	0
1	1	1

TDE 1 , 0 : TIMER DOWN COUNT ENABLE เมื่อ SET เป็น 1 ก็คือให้เริ่มทำการนับ TMDR ได้ แต่ถ้าเป็น 0 การนับจะหยุดทำงาน ะหว่าง

RESET BIT ทั้งคู่นี้จะเป็น 0 การคำนวณเวลาเราทราบแล้วว่า TIMER จะนับลงทุก ๆ 20 CLOCK ดังนั้นถ้า X'TAL ที่ใช้ในงานคือ 12 Mhz ความถี่ที่ RUN บน BOARD = 6 Mhz หาค่า TIMER คือ

T = 1/F = 0.1666 US ต่อ 1 CLOCK

20 CLOCK = 0.1666 X 20 = 3.333 US

นั่นก็คือ TIMER นับลง 1 ครั้ง ทุก ๆ 3.333 US ที่ X'TAL 12 Mhz เช่นให้ TMDR มีค่า = 1 และ SET FLAG INTERRUPT ไว้ ก็จะทำให้ PRT เกิดการ INTERRUPT ทุก ๆ 6.666 US เพราะการนับลงจะนับจากตัวเองลงก่อนคือ 1 แล้วก็ 0 จึงเท่ากับ 2 ครั้งนั่นเอง และถ้าให้กำเนิดสัญญาณสแควที่ TOUT ก็จะทำให้เกิดการ TOGGLE กันทุก ๆ จำนวนที่ให้นับ เช่น จากตัวอย่างข้างบนก็จะเป็น HIGHT 6.666 US และ LOW 6.666 US ดังนั้น 1 ลูก สัญญาณจะประมาณ 13 US หรือความถี่จะต่ำลงเท่าหนึ่งของค่าเวลา ที่คิดจากจำนวนครั้งในการนับค่าก็ได้

SECONDARY BUS INTERFACE

E CLOCK OUTPUT TIMING เป็นสัญญาณ BUT ที่ 2 เพื่อใช้เชื่อมต่อ INTERFACE เป็นไปได้ง่ายกับอุปกรณ์ PERIPHERAL ในตระกูลอื่น ๆ เช่น 68XX และ 80XX และเป็นสัญญาณที่ทำให้ระบบเกิดความน่าเชื่อถือในการใช้งาน เพราะจะติดต่อกับอุปกรณ์ก็ต่อเมื่อมีสัญญาณที่ขานี้ จากรูปขณะที่ $\overline{\text{MREQ}}$ หรือ $\overline{\text{IORQ}}$ จะเกิดขึ้นช่วงที่ T STATE แรกซึ่งยังไม่ใช่ช่วงของ DATA ที่อ่านหรือเขียน จึงทำให้อาจเกิดข้อมูลผิดพลาดกับอุปกรณ์ภายนอก แต่สัญญาณจะให้สัญญาณ ACTIVE HIGH เมื่อมีการจ่ายหรือรับ DATA เท่านั้น

FREE RUNNIG COUNTER (18H)

เป็น REGISTOR I/O ที่ใช้อ่านได้อย่างเดียวใช้สำหรับการ REFRESH DYNAMIC RAM ซึ่งเป็น COUNTER นับลง 8 BIT (A0-A7) แบบอิสระโดยจะนับลง 1 ครั้ง ทุก ๆ 10 CLOCK ถ้าเกิดมีการเขียนข้อมูลไปที่ REGISTOR นี้ จะทำให้ช่วงเวลาของการ REFRESH DYNAMIC RAM , BAUDRATE ของ ASCI และ CSI/O ไม่แน่นอน (คือ จะไม่ถูกรับปรกันว่าตรงตามที่กำหนดในคู่มือ)

ถึงแม้อยู่ใน IO STOP MODE ก็ตาม FRERUNNIG COUNTER นี้ก็ยังนับอยู่อย่างต่อเนื่องซึ่งในขณะ RESET จะมีค่าเป็น 0FFH

<u>คำสั่งเพิ่มเติม 12 คำสั่ง</u>

SLP เมื่อใช้คำสั่งนี้ CPU จะหยุดทำงานบางอย่างทำให้ใช้กำลังต่ำ

MLT MULTPLY ใช้สำหรับคูณเลข 8 BIT 2 จำนวน โดยผลลัพธ์จะเป็น 16 BIT โดย REGISOR ที่ใช้ในการคูณอาจจะเป็น BC , DE , HL หรือ SP โดยผล ลัพธ์จะได้ที่ REGISTOR คู่นั่น

<u>OTIM , OTIMR , OTDM , OTDMR - BLOCK I/O</u>

เป็นคำสั่ง OUT PORT เป็น BLOCK ของ PORT ADDRESS ต่ำ A0-A7 เท่านั่น คือ จะทำการ OUT ข้อมูลเป็น BLOCK โดยที่ PORT เพิ่มหรือลดตามจำนวนข้อมูล โดยใช้ HL เป็นตัวชี้ข้อมูลที่จะ OUT ออกไป และ C เป็น NUMBER PORT ในคำสั่ง OTIM และ OTDM ก็คือจะเพิ่มค่า HL ที่ชี้ขึ้นเป็นหนึ่ง หรือลดลง 1 ตามลำดับด้วย PORT เพิ่มขึ้นหรือลดลงด้วยและค่า B จะลดลง 1 ซึ่ง B จะเป็น COUNTER ในการส่ง DATA ส่วน OTIMR และ OTDMR จะมีลักษณะเช่นเดียวกับ OTIM และ OTDM เพียงแต่จะทำการส่งข้อมูลเพิ่มขึ้นหรือลดลง และ PORT NUMBER เพิ่มขึ้นหรือลดลงตามค่า B จนกระทั้ง B = 0

TSTIO m ใช้สำหรับ TEST I/O PORT คือ จะทำการอ่านค่า PORT ที่กำหนดโดย REGISTOR C เข้ามาแล้วทำการ AND กับ DATA 8 BIT ที่ต้องการ โดยที่ค่าข้อมูลที่ IN เข้ามานั่นไม่เปลี่ยนแปลง แต่จะให้ผลที่ FLAG และ PORT ที่ IN เข้ามาจะเป็นเฉพาะ ADDRESS ต่ำ A0-A7 เท่านั้น สามารถเปรียบเทียบเป็นโปรแกรมได้ดังนี้ :-

```
XOR  A                  LD C , NUMBER PORT
IN   A , (PORT)         TSTIO 70H
LD   B , A              JP Z , OK
LD   A , 70H
AND  B
JP   Z , OK
```

TST g - TEST REGISTOR โดยค่าที่กำหนดใน REGISTOR จะ AND กับ ACCUMULATOR ซึ่งจะทำให้มีผลต่อ FLAG ตามคำสั่ง AND แต่ค่าใน ACCUMULATOR และ REGISTOR ไม่เปลี่ยนแปลงเช่น ตัวอย่าง :-

```
LD   A , 7          LD A , 7
LD   C , A          TST B
AND  B              JR Z , OK
LD   A , C
JR   Z , OK
```

TST m - TEST IMMEDIATE เช่นเดียวกับ REGISTOR เพียงแต่ข้อมูลเป็น DATA โดยตรงที่ AND กับ ACCUMULATOR

TST (HL) - TEST MEMORY คือ จะนำค่าใน MEMORY ที่ถูกชี้โดย HL AND กับ ACCUMULATOR โดยค่าทั้ง 2 ไม่เปลี่ยนแปลงแต่ให้ผลการกระทำที่ FLAG

INO g (m) - INPUT , IMMEDIATE I/O IN ค่าจาก PORT 8 BIT (A0-A7) มายัง REGISTORใด ๆ ก็ได้ A , BC , DE , HL OUTO (m) , g - OUTPUT , IMMEDIATE I/O OUT ค่าจาก REGISTOR ใด ๆ ไปยัง PORT 8 BIT (A0-A7) REGISTOR ก็มี A , BC , DE , HL

CODE คำสั่งใหม่

MNEMONIC	OPCODE
MLT BC	ED 4C
MLT DE	ED 5C
MLT HL	ED 6C
MLT SP	ED 7C
IN0 A,(n)	ED 38 n
IN0 B,(n)	ED 00 n
IN0 C,(n)	ED 08 n
IN0 D,(n)	ED 10 n
IN0 E,(n)	ED 18 n

IN0 H,(n)	ED 20 n
IN0 L,(n)	ED 28 n
OUT0 (n),A	ED 39 n
OUT0 (n),B	ED 01 n
OUT0 (n),C	ED 09 n
OUT0 (n),D	ED 11 n
OUT0 (n),E	ED 19 n
OUT0 (n),H	ED 21 n
OUT0 (n),L	ED 29 n
OTIM	ED 83
OTIMR	ED 93
OTDM	ED 8B
OTDMR	ED 9B
TSTIO n	ED 74 n
SLP	ED 76
TST A	ED 3C
TST B	ED 04
TST C	ED 0C
TST D	ED 14
TST E	ED 1C
TST H	ED 24
TST L	ED 2C
TST n	ED 64 n
TST (HL)	ED 34

n = DATA OR NUMBER PORT 8 BIT

โครงสร้างของ CPU Z80180 ในโครงงานนี้

ส่วนของ CPU

ใช้ CPU Z80180 ของบริษัท ZILOG โดยเป็น SUPPER SET ของ CPU Z80 ใช้คำสั่งของ Z80 ได้ทั้งหมดและยังเพิ่มอีก 12 คำสั่งใช้งาน เช่น คำสั่งคูณ (MLT), TST g (คำสั่ง TEST BIT ใน REGISITOR) เป็นต้น ในบอร์ด Z80180 เลือกใช้ความเร็วขนาด 6 MHZ
แต่ก็สามารถใช้กับความถี่ 6.144 MHZ ได้ ทำให้การทำงาน 1 คำสั่ง ใช้เวลาเพียง 0.48 USEC เท่านั้น ปัญหาที่ตามมาก็คือจำเป็นต้องใช้ EPROM หรือ RAM ที่มี ACCESS TIME ต่ำ ๆ ได้เท่านั้น หรือซึ่งในข้อนี้ทาง ZILOG ได้ออกแบบให้แก้ไขจุดนี้ได้โดยสามารถ SET WAIT STATE ภายในตัว CPU ในการ READ, WRITE MEMORY หรือในการ READ, WRITE I/O DEVICE ได้ด้วย

ส่วนของ MEMORY

CP-JR180 สามารถต่อกับหน่วยความจำได้สูงสุด 128K BYTE ON BOARD โดยใช้ไอซี DECODE แบ่งหน่วยความจำเป็นช่วง ๆ ได้ 8 ช่วง ช่วงละ 32K BYTE ใช้ A18, A16, A17 มาเข้ายัง IC เบอร์ 74LS138 และใช้สัญญาณควบคุมการติดต่อกับหน่วยความจำ MREQ และ A19 มาควบคุมขา G2A, G2B เพื่อให้เกิดสัญญาณ CS เฉพาะการติดต่อกับหน่วยความจำเท่านั้น

SOCKET U2 สามารถใส่ EPROM ขนาด 64K BYTE (27512) หรือ 32K BYTE (27256) โดยใช้ JUMPER J1 เป็นตัวเลือกเบอร์ EPROM และใช้ DIODE เบอร์ 1N 4148 2 ตัว ต่อในลักษณะ AND GATE ให้ DECODE ได้ 2 ช่วง ADDRESS U2 นี้หน่วยความจำเริ่มจาก 0000H ถึง FFFFH

SOCKET U3 สามารถใส่ RAM ขนาด 32K BYTE (62256) หรือ 8K BYTE(6264) ได้โดยใช้ JUMPER J4 เป็นตัวเลือกเบอร์ RAM ในวงจรส่วนนี้สามารถใส่ BATTERY ขนาด 3 VOLTS เพื่อ BACKUP ข้อมูลใน RAM ได้ด้วย โดยใช้ MOSFET เบอร์ BS170 เป็นส่วนกันสัญญาณรบกวนจากการปิดเปิดระบบไฟไม่ให้เข้าไปรบกวน ขา CS ของ RAM เพื่อผลการ BACKUP ที่ดี U3 นี้หน่วยความจำเริ่มจาก 10000H ถึง 17FFFH

SOCKET U4 สามารถใส่ RAM หรือ ROM ได้โดยใช้ JUMPER SW เป็นตัวเลือกเบอร์ไอซีที่เราจะใส่ และใช้ JUMPER J3 เป็นตัวประกอบในกรณีใช้ RAM และก็ต้องการ BACKUP ข้อมูลด้วย U4 นี้หน่วยความจำเริ่มจาก 18000 H ถึง 1FFFFH

ส่วนของ PORT

ในส่วนนี้เราใช้ไอซี DECODE PORT U7 74LS138 เป็นตัว DECODE แบ่งช่วง ADDRESS PORT โดยใช้สัญญาณควบคุม I/O DEVICE (IORQ) และ M1 มาควบคุมเพื่อให้เกิดสัญญาณ CS ขึ้นเฉพาะในการติดต่อกับส่วน I/O เท่านั้น

8255 PORT เราใช้ไอซี PORT ใช้งานประจำบอร์ดโดยเป็น PORT ขนาด 8 BIT 3 PORT ใช้งานโดยมีตำแหน่ง ADDRESS ดังนี้

PORT A = 80H

PORT B = 81H

PORT C = 82H

CONTROL PORT = 83H

ส่วนของ POWER ON RESET และ WATCH DOG

ใช้ไอซีของบริษัท DALLAS SEMICONDUCTOR เบอร์ DS1232 เป็นวงจร POWER ON RESET และ WATCH DOG เพิ่มความมั่นใจในบอร์ดยิ่งขึ้น โดยในส่วนของ POWER ON RESET นั่นจะทำการ RESET CUP เมื่อ VOLT มากกว่าหรือต่ำกว่า 4.75 VOLTS และจะหน่วงเวลาในกรณี POWER ON ประมาณ 250 USEC ถึง 1 SEC และในส่วนของ WATCH DOC นั้น สามารถเลือกระยะเวลาในการทริกได้ด้วย JUMPER TD โดยเลือกเวลาในการทริก WATCH DOG

นอกจากนี้ยังสามารถ SET ให้วงจร WATCH DOG ทำงานหรือไม่ทำงานได้ด้วยการ SET JUMPER WDT และถ้า SET WATCH DOG ทำงานแล้วจะต้องทริก PORT EOH โดยจะใช้คำสั่ง IN หรือ OUT PORT ก็ได้ตามระยะเวลาที่เลือกเพื่อไม่ให้วงจร WATCH DOG นั้นทำการ RESET CPU

ส่วนของ SERIAL PORT

ASYNCHRONOUS SERIAL COMMUNICATION (ASCI) นี้จะประกอบอยู่ในชิปของ Z80180 ทั้ง 2 PORT ซึ่งแยกกันอย่างอิสระ สามารถติดต่อสื่อสารกัน

ได้แบบ FULL DUPLEX ทั้ง 2 PORT ตามมาตรฐานของ UART (UNIVERSAL ASYNCHRONOUS RECEIVER / TRANSMITTER)

- FULL - DU, LEX COMMUNICATION
- 7 OR 8 BIT DATA LENGTH
- PROGRAM CONTROLLED 9TH DATA BIT FOR MULTIPROCESSOR COMMUNICATION
- 1 OR 2 STOP BIT
- ODD, EVEN, NO PARITY
- PARITY, OVER RUN, FRAMING ERROR DETECTION
- SPEED TO 38.4K BITS PER SECOND
- MODEM CONTROL SIGNAL - CHANNEL
- PROGRAMMABLE INTERRUPT CONDITION ENABLE AND DISABLE
- OPERATION WITH ON - CHIP DMAC

การติดต่อสื่อสารข้อมูล (RS 232C)

โดยปกติไมโครคอมพิวเตอร์จะมีพอร์ตที่เป็นแบบอนุกรม เรียกชื่อกันว่า RS 232C อยู่ในตัวเองอยู่แล้ว หลายเครื่องไม่มีมากับเครื่อง อย่างเช่น IBM PC จำเป็นจะต้องมีการ์ดที่เรียกว่า อะซิงโครนัสอะแดปเตอร์ (Asynchronous Communication Adapter) มาเสียบใส่

พอร์ต RS 232 นี้ทำหน้าที่รับและส่งข้อมูลในแบบอนุกรมเรียกว่า Universal Asynchronous Adapter เหตุที่มีชื่อเรียกว่า RS 232C ก็เนื่องจากสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิคส์ของอเมริกาหรือ EIA ได้กำหนดมาตรฐานของอุปกรณ์การสื่อสารแบบอนุกรมเอาไว้ภายใต้ชื่อว่า RS 232C ความจริงมาตรฐานของการส่งข้อมูลแบบอนุกรมมีหลายมาตรฐาน แต่ที่นิยมกันมากที่สุดสำหรับไมโครคอมพิวเตอร์ก็คือ RS 232C

หน้าที่สำคัญของการสื่อสารแบบอะซิงโครนัส ก็คือ

รับสัญญาณ

1. เปลี่ยนสัญญาณเข้ามาแบบอนุกรมให้เป็นแบบขนาน
2. ตรวจสอบความผิดพลาดของสัญญาณที่รับ
3. ตัดสต๊อปบิต และพาริตี้บิตออก
4. ส่งสัญญาณให้ซีพียู รู้ว่ารับสัญญาณไว้แล้ว

ส่งสัญญาณ

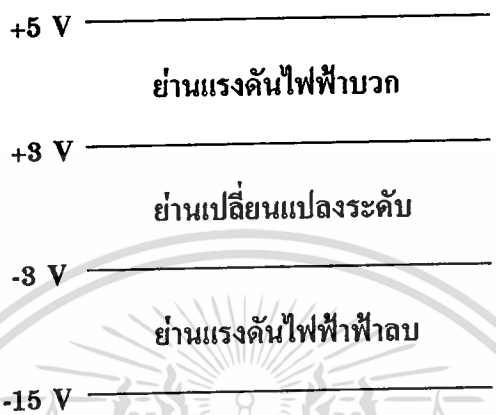
1. เปลี่ยนสัญญาณแบบขนานจากซีพียู ค่อยทยอยส่งออกเป็นแบบอนุกรม
2. เพิ่มสต๊อปบิต และพาริตี้
3. เพิ่มสัญญาณควบคุมโมเด็มที่ต่อเชื่อม (ถ้ามี)

ลักษณะของสัญญาณ RS 232C

เพื่อเป็นหลักประกันว่าข้อมูลถูกส่งออกไปอย่างถูกต้อง และอุปกรณ์ถูกควบคุมอย่างถูกต้อง จำเป็นต้องมีข้อตกลงกันในเรื่องของสัญญาณที่ใช้ มาตรฐาน RS 232C กำหนดย่านของแรงดันไฟฟ้าในสัญญาณเพื่อสนองจุดประสงค์ข้างบน ดังแสดงในตาราง และรูป

ตาราง

มาตรฐานของการใช้แรงดันไฟฟ้า			
แรงดันไฟฟ้า	สถานภาพลอจิก	สถานภาพของสัญญาณ	ฟังก์ชันในการควบคุม
บวก	1	สเปซ	ออน
ลบ	0	มาร์ค	ออฟ



รูปที่ 2.10 รูปย่านของแรงดันไฟฟ้าที่ใช้ในสัญญาณ RS 232C

สำหรับไมโครคอมพิวเตอร์บางเครื่อง ใช้แต่สัญญาณลอจิกออกมาเป็นสัญญาณของ RS 232C เลย อย่างเช่น อะซิงโครนัสอะแคปเตอร์ของ IBM PC ในกรณีเช่นนี้ระยะทางของสายที่เชื่อมต่อ อาจจะไปได้สั้นกว่า 50 ฟุต ดังที่กล่าวเอาไว้เนื่องจากระดับของกราวนด์เปลี่ยนแปลงไป อันเนื่องจากการสูญเสียเสียไปในความต้านทานของสาย

การกำหนดจุดข้อต่อของ RS 232C

ในทางฟิสิกส์แล้ว มาตรฐานของ RS 232C กำหนดข้อต่อแบบ DB-25 แต่ละขาของข้อต่อกำหนดไว้ดังรูป อย่างไรก็ตามผู้ผลิตไมโครคอมพิวเตอร์ อาจจะใช้ข้อต่อชนิดอื่นที่นอกเหนือไปจาก DB-25 ยกตัวอย่างเช่น FUJITSU F-8 , IBM AT , IBM JR เป็นต้น ตัวเมียของข้อต่อควรจะอยู่ที่โมเด็ม ขณะที่ตัวผู้ควรจะอยู่ที่ Asynchronous Communication Adapter หรือที่ตัวไมโครคอมพิวเตอร์เอง อย่างไรก็ตามผู้ผลิตหลายรายไม่ได้ทำตามกฎเกณฑ์ที่ว่านี้

สัญญาณต่าง ๆ ถูกมอบหมายให้ทำหน้าที่ดังนี้

Transmit Data (TD ขาที่ 2)

เป็นสัญญาณที่ส่งออกจาก DTE (หรือตัวไมโครคอมพิวเตอร์) ไปยังโมเด็ม หรือต่อเข้าโดยตรงกับไมโครคอมพิวเตอร์ตัวอื่น หรือเครื่องพิมพ์ เมื่อไม่มีสัญญาณส่งออกสถานภาพของลอจิกที่ขานี้จะมีค่าเท่ากับ"1" หรือเทียบเท่ากับสต๊อปบิต

Receive Data (RD ขาที่ 3)

เป็นทางของสัญญาณเข้าไปยัง DTE หรือไมโครคอมพิวเตอร์เมื่อไม่สัญญาณรับเข้ามา ขานี้จะมีสถานภาพทางลอจิก เป็น "1"

Request To Send (RTS ขาที่4)

ใช้สำหรับส่งสัญญาณไปยังโมเด็ม หรือเครื่องพิมพ์เป็นการเรียกร้องที่จะส่งสัญญาณมาทางขา 2 สัญญาณนี้ใช้คู่กับ CTS หรือ Clear to send อุปกรณ์รับหากได้รับสัญญาณ RTS จะตรวจสอบตัวเองว่าพร้อมจะรับสัญญาณได้หรือยัง หากพร้อมที่จะรับก็ส่งสัญญาณออกไปที่สาย CTS

Clear To Send (CTS ขาที่ 5)

ดังอธิบายไว้ใน RTS เมื่อสัญญาณนี้อยู่ในสถานะออฟ (negative voltage หรือลอจิก "1") หมายความว่า อุปกรณ์รับกำลังบอกว่าพร้อมที่จะรับข้อมูลแล้ว

Data Set Ready (DSR ขาที่ 6)

เมื่อสัญญาณสายนี้อยู่ในสถานะออน (หรือลอจิก 0) เป็นการบอกไมโครคอมพิวเตอร์หรือฝ่ายส่งว่า โมเด็มต่อเข้ากับสายโทรศัพท์เรียบร้อยแล้ว และพร้อมที่จะส่งได้แล้ว โมเด็มที่มีการหมุนหมายเลขอัตโนมัติจะส่งสัญญาณนี้ไปบอกให้คอมพิวเตอร์รู้ว่าต่อโทรศัพท์ได้สำเร็จแล้ว

Signal Ground (SG ขาที่ 7)

SG ทำหน้าที่เป็นระดับแรงดันอ้างอิงสำหรับทุก ๆ สายของสัญญาณ จะมีแรงดันเป็น "0" เมื่อเทียบกับสัญญาณตัวอื่น

Carrier Detect (CD ขาที่ 8)

โมเด็มจะส่งสัญญาณที่อยู่ในสถานะออน (ลอจิก "0") ไปบอกไมโครคอมพิวเตอร์เมื่อได้รับสัญญาณจากโมเด็มของอีกฝ่ายหนึ่ง สัญญาณนี้จะนำไปจุด LED บอกว่าได้รับสัญญาณจากโมเด็มอีกฝ่ายหนึ่งแล้ว ไฟ LED จะอยู่บนหน้าปัดของโมเด็มเอง

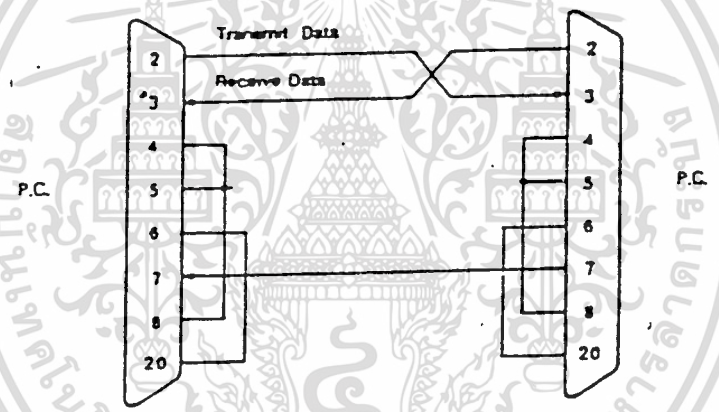
Data Terminal Ready (DTR ขาที่ 20)

คอมพิวเตอร์เปิดสัญญาณสายนี้ให้ออน (ลอจิก "0") เมื่อพร้อมที่จะติดต่อกับโมเด็ม โมเด็มส่วนมากจะไม่รายงานสถานภาพของตัวเอง (CD , USR และ CTS) ให้คอมพิวเตอร์รู้หากคอมพิวเตอร์ไม่เปิดสัญญาณ DTR

Ring Indicator (RI ขาที่ 22)

สัญญาณที่ใช้ในโมเด็มที่เป็นระบบตอบได้อัตโนมัติ (Auto-answer) สัญญาณนี้จะออนเมื่อมีสัญญาณกระดิ่งมา และออฟระหว่างเสียงดังของกระดิ่ง

วิธีการต่อ RS 232C เข้าระหว่างเครื่องคอมพิวเตอร์โดยตรงมีอยู่หลายวิธีตามแต่ขบวนการที่จะใช้ ถ้าไม่ต้องการมีการตรวจสอบสัญญาณกันก็ต่อ RD เข้า TD ของอีกเครื่องหนึ่ง สายกราวนด์ต่อถึงกันดังรูป ก็สามารถใช้งานถ่ายโอนข้อมูลได้แล้ว

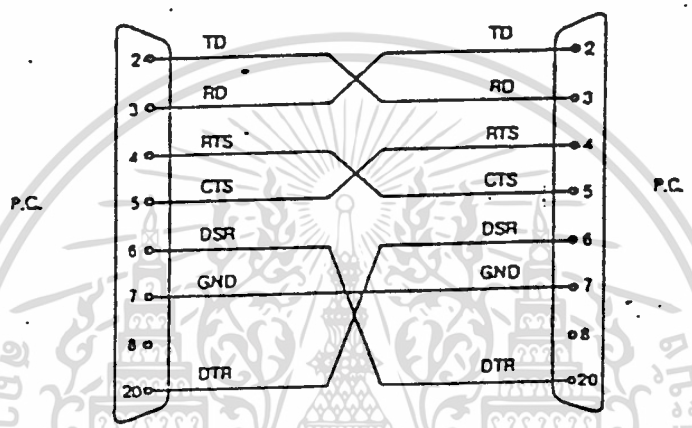


รูปที่ 2.11 รูปการต่อไมโครคอมพิวเตอร์ผ่าน RS 232C แบบง่าย ๆ

ปกติ CS ที่ให้บริการเกี่ยวกับพอร์ต RS 232C จะส่งสัญญาณ RTS หรือ Request to sent ออกมาที่ขา 4 ก่อน เมื่อ CS หรือ clear to send ที่ขา 5 เป็นลอจิก "1" (หรือไฟลบ) จึงจะเริ่มทำการส่งข้อมูลที่โอเปอเรเตอร์บอกให้ส่งออกไปที่ขา 2 ในกรณีที่เป็นการต่อแบบง่าย ๆ ในรูป จึงถือว่าเป็นการหลอกคอมพิวเตอร์โดยเอาขา 4 RTS ต่อเข้ากับขา 5 หรือ CTS เพื่อให้คอมพิวเตอร์ส่งข้อมูลได้ทันทีโดยไม่ต้องการความเรียบร้อยของฝ่ายรับ สำหรับขา 6 Data Set Ready ต่อเข้ากับขาที่ 20 Data Terminal Ready ก็ทำนองเดียวกัน โดยปกติคอมพิวเตอร์จะถามอุปกรณ์ที่มาต่อพ่วงกับ RS 232C ว่าพร้อมที่จะส่งแล้วหรือยัง โดยส่งสัญญาณถามที่ขา 20 และรอคำตอบที่ขา 6 นี้ก็เป็นการหลอกคอมพิวเตอร์เหมือนกัน คือ ถามที่ขา 20 ก็ได้รับคำตอบกลับที่ขา 6 ทันที

ในการต่อแบบนี้ฝ่ายรับจะต้องรอรับอยู่ก่อนแล้ว ก่อนที่ฝ่ายส่งจะเป็นผู้ส่ง ไม่เช่นนั้นข้อมูลที่ส่งออกมาหายแน่ ๆ เพราะฝ่ายส่งไม่ได้ตรวจสอบความเรียบร้อยของฝ่ายรับก่อน

เราอาจจะต่อสายให้มีการตรวจสอบสัญญาณโต้ตอบ (Hand Shake) ที่ดีกว่านี้ ในรูป



รูปที่ 2.12 รูปการต่อไมโครคอมพิวเตอร์ผ่าน RS 232C แบบมี Hand Shake

ในกรณีเช่นนี้จะมีการโต้ตอบที่ดีขึ้น เมื่อฝ่ายรับยังไม่พร้อมที่จะรับก็จะยังไม่มีสัญญาณ RTS ออกมา (พอร์ตอนุกรมยังไม่เปิด หรือ OPEN "COM1" ในภาษาเบสิกยังไม่ถูกเอกซีคิ้ว) ฝ่ายส่งซึ่งถือเอา RTS ของฝ่ายรับเป็น CTS ก็จะยังไม่ส่ง

บทที่ 3
ทฤษฎี และการออกแบบโปรแกรม

ทฤษฎีการออกแบบโปรแกรมเชิงภาพ (Event Driven)

ใน Visual Basic นั้น การพัฒนาโปรแกรมจะใช้หลักของภาพและการมองเห็น โดยเริ่มจากออกแบบวินโดวส์ย่อย หรือที่ใน Visual Basic เรียกว่า ฟอร์ม (Form) ในฟอร์มจะประกอบด้วยสิ่งต่าง ๆ ที่เราจะทำงานด้วย หรือเรียกว่า Object เช่น ข้อความ , ช่วงรับข้อความ , Scroll Bar หรือปุ่ม (button) เมื่อกำหนดสิ่งเหล่านี้ครบตามความต้องการแล้ว จึงระบุว่าองค์ประกอบแต่ละอย่างจะทำงานอย่างไร โดยเขียนโปรแกรมย่อย ๆ ปะเข้าไปกับ object เหล่านี้ ที่ต้องทำแบบนี้ ก็เพราะว่าการทำงานบนวินโดวส์ เป็นแบบ EVENT DRIVEN คือขึ้นกับเหตุการณ์ (Event) ซึ่งจะยุ่งยากมากหากใช้การเขียนโปรแกรมแบบ การสั่งงานตามลำดับ เนื่องจากเป็นระบบแบบ Multitasking คือ Windows จะต้องจัดการกับทุกแอพพลิเคชั่น ที่ทำงานในขณะนั้นทั้งหมดไปพร้อม ๆ กัน ตัวอย่างเช่น ในขณะที่โปรแกรมแสดงหน้าจอในการรับอินพุท ผู้ใช้อาจพิมพ์ข้อมูลเข้าหรืออาจจะเลื่อนเมาส์ไปคลิ๊กที่อื่น ทำให้ยากที่จะเขียนโปรแกรมแบบโครงสร้างธรรมดา ในการดักเส้นทางการทำงานในการรับอินพุทว่า จะเกิดอะไรขึ้น ตรงไหนได้ จึงต้องใช้รูปแบบการโปรแกรมในลักษณะ EVENT DRIVEN โดยที่ object แต่ละตัวจะมีเหตุการณ์เกิดขึ้นได้หลายอย่าง โดยสามารถโปรแกรมสั่งงานให้คอยดักหรือทำงานตามเฉพาะเหตุการณ์ที่สนใจได้ เช่น ถ้าสิ่งที่สนใจ (object) คือปุ่มควบคุม มีการทำงานในลักษณะที่คลิ๊ก หรือดับเบิลคลิ๊ก ก็ระบุได้ว่า หากมีการคลิ๊กที่ปุ่มควบคุมนี้ โปรแกรมจะต้องทำอย่างไร ส่วนเหตุการณ์อื่นที่ไม่ได้ระบุก็จะไม่มีผลต่อ object นั้น

นอกจาก object จะมีการตอบสนองต่อเหตุการณ์ต่าง ๆ ที่กำหนดแล้ว ทุก object จะมีลักษณะ หรือคุณสมบัติ (Properly) ของตัวเองเช่น ช่วงรับข้อความ (Text Box), ข้อความนั้น ๆ , ความกว้าง , ความสูง , สี โดยเราสามารถอ้างถึงหรือเปลี่ยนคุณสมบัติเหล่านี้ได้ ขณะที่โปรแกรมทำงานอยู่ เป็นต้นว่า หากไม่มีการป้อนข้อมูลจะให้แสดงด้วยสีหนึ่ง หรืออาจไม่ต้องแสดงเลย

สำหรับเหตุการณ์เฉพาะที่จะเกิดกับ object ก็คือ method และ object แต่ละแบบก็อาจจะมี method ที่แตกต่างกันออกไป เช่น ถ้าต้องการสั่งให้เลื่อนตำแหน่งของข้อความ (Label) ก็จะมีกระบวนการ หรือ method ชื่อ Move ของ Label เพิ่งทำงานนี้ โดยสั่งว่า

Label Move = ตำแหน่งที่จะย้ายไป หรือการสั่งพิมพ์ก็มี method ซึ่ง Print เป็นต้น ถ้าพูดไปแล้ว method นี้คล้าย ๆ กับ คำสั่งที่ใช้ได้กับ object แต่ละชนิดนั่นเอง

โดยสรุปแล้ว รูปแบบของหลักการใน Visual Basic ก็คือ เริ่มจากออกแบบจอภาพ และเขียนโปรแกรมสำหรับแต่ละ Event ปะเข้าไปยัง object ต่าง ๆ ให้ทำงานตามเหตุการณ์ที่เกิดขึ้น โดยทุก object จะมีคุณสมบัติเฉพาะ ที่สามารถเปลี่ยนแปลงได้

การเขียนโปรแกรมสื่อสารแบบอนุกรมภายใต้วินโดวส์

วินโดวส์ให้ฟังก์ชันภายในมากกว่าดอสมาก ฟังก์ชันเหล่านี้บางครั้งทำให้การเขียนโปรแกรมง่ายขึ้น แม้ว่าความซับซ้อนของสภาวะแวดล้อมบนวินโดวส์ จะมากกว่า ประโยชน์ที่ได้จากฟังก์ชันก็ตาม โปรแกรมเมอร์สามารถเรียก ฟังก์ชัน API (APPLICATION RPOGRAMMING INTERFEC) โดยที่วินโดวส์ มีให้จำนวนหนึ่งสำหรับการสื่อสารแบบอนุกรม ซึ่งมีรูปแบบการประกาศตัวแปร และฟังก์ชัน เหล่านี้เป็นมาตรฐานของวินโดวส์อยู่แล้ว เหล่านี้คือ

ฟังก์ชัน	หน้าที่
BuildCommDCB	ใส่รหัสควบคุมอุปกรณ์ในบล็อกคุมอุปกรณ์
ClearCommBreak	ยกเลิกสถานะการหยุดรอที่มาจากอุปกรณ์
CloseComm	ปิดการสื่อสารกับอุปกรณ์เมื่อส่งข้อมูลในบัฟเฟอร์ไปแล้ว
EscapeCommFunction	บอกอุปกรณ์ให้ฟังก์ชันพิเศษ
FlushComm	ล้างตัวอักษรที่มาจากอุปกรณ์ทิ้ง
GetCommError	ใส่สถานะการสื่อสารลงในบัฟเฟอร์
GetCommEventMask	ดึงค่า แล้วเคลียร์อีเวนต์มาสก์
GetCommState	ก๊อบปี้ค่าจากบล็อกคุมอุปกรณ์มาใส่บัฟเฟอร์
OpenComm	เริ่มการสื่อสารกับอุปกรณ์
ReadComm	อ่านข้อมูลหนึ่งไบต์จากอุปกรณ์มาใส่บัฟเฟอร์
SetCommBreak	ตั้งค่าสถานะการหยุดของอุปกรณ์
SetCommEvent ask	ดึงค่า และตั้งค่าอีเวนต์มาสก์
SetCommState	ตั้งค่าอุปกรณ์ให้เป็นไปตามสถานะที่กำหนดในดีไวซ์คอนโทรล

TransmitCommChart	บล็อก
UngetCommChar	ใส่ตัวอักษรไปต้นคิวของการส่ง
Write Comm	กำหนดตัวอักษรต่อไปที่จะถูกอ่านเขียนข้อมูลหนึ่งไบต์จากบัฟเฟอร์ไปยังอุปกรณ์

ฟังก์ชันการสื่สารแบบอนุกรม

การเปิดพอร์ต

การเปิดพอร์ตใช้ฟังก์ชัน OpenComm() โดยมีไวยากรณ์ดังนี้

```
int OpenComm(lpComName, sInQueue, wOutQueue)
LPSTR lpComName;
WORD wInQueue, wOutQueue;
```

lpComName ชี้ไปยังชื่อพอร์ตที่ต้องการเปิด (เช่น "COM1") wInQueue และ wOutQueue เป็นขนาดของคิวรับข้อมูล (receive queue) และคิวส่งข้อมูล (transmitt queue) ที่ต้องการจอง ขนาดที่ควรจะจองให้กับคิวเหล่านี้ขึ้นอยู่กับวิธีการถ่ายโอนข้อมูล

ถ้าผลลัพธ์จาก OpenComm เป็นจำนวนบวก แสดงว่าการทำงานสำเร็จ ค่าที่คืนกลับมาจะนำไปใช้เป็นแฮนเดิล (handle) เพื่อเข้าถึงพอร์ตลักษณะนี้คล้ายกับฟังก์ชัน Open() ซึ่งเป็นที่คุ้นเคยของนักเขียนโปรแกรมภาษาซีอยู่แล้ว ถ้าผลลัพธ์เป็นจำนวนลบ สามารถนำผลลัพธ์ไปใช้ตรวจสอบสาเหตุที่การทำงานล้มเหลวได้ โดยค่าที่คืนกลับมาจะมีรหัสตามที่นิยามไว้ในวินโดวส์

การปรับตั้งพอร์ต

เมื่อเปิดพอร์ตแล้วก็สามารถตั้งค่าพารามิเตอร์ เช่น อัตราบอดได้ โดยการใช้ฟังก์ชัน SetCommState() ซึ่งมีไวยากรณ์ดังนี้

```
int SetCommState (lpDCB)
DCB FAR *lpDCB;
```

lpDCB เป็นพอยน์เตอร์ (pointer) ไปยังสตรักเจอร์ (structure) ชนิด DCB (Device Control Block) ซึ่งเป็นที่เก็บพารามิเอตร์ที่เหมาะสม สตรักเจอร์นี้มีโครงสร้างตามตาราง ถ้าการเรียกสำเร็จจะคืนค่าศูนย์ หรือคืนค่าลบถ้าล้มเหลว

ตาราง สตรักเจอร์ DCB

คำสั่ง	ฟิลด์	คำบรรยาย
BYTE	Id;	หมายเลขอุปกรณ์ที่คืนมาโดย OpenComm()
UNIT	Baud Rate;	อัตราบอด
BYTE	Byte Size;	ความยาวเวิร์ด (4 ถึง 8 บิต)
BYTE	Parity;	พาริตี้ นิยามไว้เป็น EVENPARITY, MARKPARITY, NOPARITY, ODDPARITY หรือ SPACEPARITY
BYTE	Stop Bits;	จำนวนของบิตจบ นิยามไว้เป็น ONESTOPBIT, ONE5 STOPBIT หรือ TWOSTOPBITS
UINT	Ils Timeout;	เวลา (มิลลิวินาที) ที่รอสัญญาณ Carrier Detect (CD)
UINT	Cts Timeout;	เวลาที่รอสัญญาณ CTS
UINT	Dsr Timeout;	เวลาที่รอสัญญาณ DSR
UINT	f Binary : 1;	ถ้าเป็นหนึ่ง คือภาวะไบนารี เมื่อพบ E of Char ที่นิยามไว้ข้างล่างจะถือว่าเป็นการสิ้นสุดข้อมูล
UINT	f Rts Disable : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะไม่เปิดสัญญาณ RTS
UINT	f Parity : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะรายงานข้อผิดพลาดทางพาริตี้
UINT	f Out x Cts Flow : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะทำการส่งขณะที่ CTS เป็นไฮเท่านั้น
UINT	f Out x Dsr Flow : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะทำการส่งขณะที่ DSR เป็นไฮเท่านั้น
UINT	f Dummy : 2;	สงวนไว้
UINT	f Ddtr Disable : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะไม่เปิด DTR หรือทำการปิดมันเมื่อพอร์ตถูกปิด
UINT	f Out X : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะหยุดการส่งชั่วคราว เมื่อได้ X off Char และส่งต่อไปเมื่อได้รับ X on Char
UINT	f In X : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะส่ง X off Char เมื่อบัฟเฟอร์ของมันเกือบเต็ม และส่ง X on Char เมื่อมันพร้อมที่จะรับข้อมูลเพิ่มขึ้น
UINT	f Pe Char : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะแทนที่ตัวอักษรได้แล้วเกิดความผิดพลาดทางพาริตี้ Pe Char ซึ่งนิยามไว้ข้างล่าง
UINT	f Null : 1;	ถ้าเป็นหนึ่ง จะไม่สนใจตัวอักษร Null (ASCII 0) ที่รับได้

UINT	f Ch Evt : 1;	จะส่งสัญญาณแจ้งเหตุการณ์ (event signal) ไปให้โฮสต์โปรแกรม ถ้าได้รับอักษรที่นิยามให้เป็น Evt Char
UINT	f Dtr Flow : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะใช้สัญญาณ DTS สำหรับโฟลว์คอนโทรล
UINT	f Rts Flow : 1;	ถ้าเป็นหนึ่ง วินโดวส์จะใช้สัญญาณ RTS สำหรับโฟลว์คอนโทรล
UINT	f Dummy 2: 1;	สงวนไว้
char	X on Char;	ตัวอักษรที่ใช้เพื่อร้องขอข้อมูลเพิ่มขึ้น
char	X off Char;	ตัวอักษรที่ใช้เพื่อหยุดการส่งข้อมูล
UINT	X on Lim;	จำนวนตัวอักษรน้อยที่สุด ที่ต้องอยู่ในบัฟเฟอร์ข้อมูลเข้าก่อนที่วินโดวส์จะส่ง X off Char ถ้า f In X เป็นหนึ่ง
UINT	X off Lim;	จำนวนตัวอักษรมากที่สุด ที่ต้องอยู่ในอินบัฟเฟอร์ ก่อนที่วินโดวส์จะส่ง X on Char , f In X เป็นหนึ่ง
char	Pe Char;	ตัวอักษรที่ใช้แทนตัวอักษรที่รับเข้ามาแล้ว เกิดความผิดพลาดทางพาริตี้ ถ้า f Pe Char เป็นหนึ่ง
char	E of Char;	ตัวอักษรที่ถือว่าเป็นตัวบอกจุดสิ้นสุดของข้อมูลถ้า fBinary เป็นหนึ่ง
char	Evt Char;	ตัวอักษรที่จะกระตุ้นสัญญาณแจ้งเหตุการณ์ไปยัง โฮสต์โปรแกรม ถ้า f Ch Evt เป็นหนึ่ง
UINT	Tx Delay;	ไม่ใช้

ค่าที่ตั้งไว้ในปัจจุบันสามารถตรวจสอบโดยฟังก์ชัน Get Comm State()

nCid เป็นหมายเลขอุปกรณ์ที่คืนกลับมาโดย Open comm() 1pDCB เป็นพอยน์เตอร์ไปยังสตรักเจอร์ DCB เป็นที่ซึ่งวินโดวส์จะใส่ค่าปัจจุบันลงไป ถ้ารหัสที่คืนกลับมาเป็นลบแสดงว่าการเรียกใช้ล้มเหลว ซึ่งอาจมีสาเหตุจากหมายเลขอุปกรณ์ไม่ถูกต้อง

วิธีปกติของการตั้งค่าพารามิเตอร์ คือเปิดพอร์ต อ่านพารามิเตอร์ในขณะนั้นโดยใช้ Get Comm State() แก้ไขพารามิเตอร์ภายใน DCB ตามความจำเป็น และเรียกใช้ Set Comm State()

ยังมีฟังก์ชันหนึ่ง คือ Build comm DCB() ซึ่งจะสร้าง DCB ที่มีพารามิเตอร์จำกัด 1p Def เป็นพอยน์เตอร์ไปยังสตริงที่มีรูปแบบเดียวกับที่ใช้ในคำสั่ง MOD ของดอส 1p DCB เป็นพอยน์เตอร์ไปยังสตรักเจอร์ DCB

การอ่านจากพอร์ตอนุกรม

หลังจากทำการเปิดอุปกรณ์อนุกรมและตั้งพารามิเตอร์ การอ่านข้อมูลทำได้โดยใช้ฟังก์ชัน Read Comm()
ที่วินโดวส์เตรียมไว้ให้ตามไวยากรณ์

nCid เป็นหมายเลขอุปกรณ์ที่ได้จาก Open Comm() 1p Buf เป็นพอยน์เตอร์ไปยังบัฟเฟอร์สำหรับข้อมูล nSize เป็นจำนวนสูงสุดของไบต์ ที่จะอ่าน

วินโดวส์จะไม่คอยจนกระทั่งได้รับข้อมูลจำนวน nSize ไบต์ แต่มันจะคืนค่าจำนวนไบต์ที่มีอยู่ในบัฟเฟอร์ขณะที่ใช้ฟังก์ชันนี้ ถ้าเกิดข้อผิดพลาด Read Comm() จะคืนค่าเป็นจำนวนลบ

การเขียนไปยังพอร์ตอนุกรม

ฟังก์ชัน Write comm() ทำหน้าที่ส่งตัวอักษรไปยังอุปกรณ์อนุกรมโดยมีไวยากรณ์ดังนี้

```
int Write Comm (nCid, lpBuf, nSize)
int nCid;
LPSTR lpBuf;
int nSize;
```

nCid เป็นหมายเลขอุปกรณ์ที่ได้จาก Open Comm() 1pBuf เป็นพอยน์เตอร์ไปยังบัฟเฟอร์บรรจุข้อมูลที่จะส่ง nSize เป็นจำนวนไบต์ที่จะทำการส่ง

วินโดวส์จะคืนค่าจำนวนตัวอักษรที่ถูกส่งไปจริง ซึ่งอาจน้อยกว่าจำนวนที่ร้องขอไป ถ้าไม่มีที่ว่างในคิวส่งข้อมูลเพียงพอ ฟังก์ชันจะไม่รอจนกระทั่งตัวอักษรถูกส่งไปหมดแล้วจึงคืนกลับมา มันเพียงแค่ใส่ตัวอักษรลงในบัฟเฟอร์สำหรับการส่งด้วยอัตราที่เหมาะสม และตามวิธีการแฮนด์เช็คกิ้งที่ใช้ ถ้ามีข้อผิดพลาด Write Comm()
จะคืนค่าเป็นจำนวนลบ

การอ่านข้อผิดพลาดและสถานะ

การหาว่ามีข้อผิดพลาดใดเกิดขึ้นใช้ฟังก์ชัน Get Comm Error() ซึ่งข้อมูลเกี่ยวกับสถานะของพอร์ตด้วยเช่นกัน

การเรียก Get Comm Error() ทำให้แฟล็กภายในของวินโดวส์ถูกเคลียร์ด้วย เพื่อจะได้สามารถทำการสื่อสารได้ต่อไป เมื่อมีข้อผิดพลาดเกิดขึ้น การเรียกฟังก์ชันการสื่อสารจะล้มเหลวจนกว่า Get Comm Error() จะเรียกใช้ตามไวยากรณ์ดังนี้

```
int Get Comm Error (nCid, lpStat)
int nCid;
COMSTAT FAR  *lpStat;
```

nCid เป็นหมายเลขอุปกรณ์ที่คืนมาโดย Opcn Comm() lpstat เป็นพอยน์เตอร์ไปยังสตรักเจอร์ชนิด
COMSTAT ซึ่งจะเก็บสถานะให้อุปกรณ์เมื่อออกจาก Get Comm Error() ค่าที่คืนกลับมาเป็นค่าที่เกิดจากการ OR ของค่าต่อไปนี้

CE_BREAK	ได้รับสัญญาณเบรก
CE_CTSTO	ไทม์เอาต์ขณะที่กำลังรอสัญญาณ CTS
CE_DSRTO	ไทม์เอาต์ขณะที่กำลังรอสัญญาณ DTR
CE_FRAME	เกิดความผิดพลาดทางเฟรม
CE_MODE	พารามิเตอร์ nCid ไม่ถูกต้อง
CE_OVERRUN	เกิดความผิดพลาดโอเวอร์รัน
CE_RLSDTO	ไทม์เอาต์ขณะที่กำลังรอสัญญาณ CD
CE_RXOVER	บัฟเฟอร์รับข้อมูลเกิดโอเวอร์โฟลว์หรือได้รับตัวอักษรหลังจากการรับ E of Char
CE_RXPARITY	เกิดความผิดพลาดทางพาริตี้
CE_TXFULL	บัฟเฟอร์ส่งข้อมูลเต็ม

สตรักเจอร์ COMSTAT บอกให้ทราบสถานะปัจจุบันของพอร์ตสื่อสาร โดยมีโครงสร้างของสตรักเจอร์

แฮนด์เช็คกิ้งและสัญญาณเบรก

ในทางทฤษฎี สถานะปัจจุบันของแฮนด์เช็คกิ้งสามารถถูกตรวจสอบได้ โดยการใช้ฟังก์ชันGet Comm Error() และดูที่สตรักเจอร์ COMSTAT อย่างไรก็ตาม เนื่องจากบัก (bug) ในวินโดวส์ 3.0 ซึ่งเกิดกับวินโดวส์ 3.1 ด้วย ค่าที่คืนมาสำหรับสัญญาณแฮนด์เช็คกิ้ง

(DTR, CTS, RI และ CD) จะไม่ถูกต้องไมโครซอฟต์ได้เปิดเผยวิธีที่สามารถใช้เพื่อตรวจสอบค่าแท้จริง โดยใช้ค่าพอยน์เตอร์ไปยังไบต์ที่เก็บสำเนาของ รีจิสเตอร์แสดงสถานะโมเด็ม เมื่อได้ค่าพอยน์เตอร์นี้แล้วก็สามารถตรวจสอบแต่ละบิตภายในรีจิสเตอร์ แสดงสถานะโมเด็ม

การเปลี่ยนแฮนด์เช็คกิ้ง

ฟังก์ชัน Escape Comm Function() สามารถใช้เพื่อเปลี่ยนสถานะแฮนด์เช็คกิ้ง และบอกให้วินโดวส์เปลี่ยนสัญญาณทางฮาร์ดแวร์ สถานะของซอฟต์แวร์แฮนด์เช็คกิ้งได้ โดยมีไวยากรณ์ดังนี้

```
int Escape Comm Function (nCid, nFunc)
int nCid;
int nFunc:
```

nCid เป็นหมายเลขของอุปกรณ์สื่อสารที่ได้จาก Open Comm() nFunc เป็นตัวเลขที่บอกฟังก์ชันที่ต้องการให้ทำงาน ฟังก์ชันที่มีได้แก่

CLRDTR	ปิดสัญญาณ DTR
CLRRTS	ปิดสัญญาณ RTS
RESETDEV	รีเซตอุปกรณ์ถ้าทำได้
SETDTR	เปิดสัญญาณ DTR
SETRTS	เปิดสัญญาณ RTS
SETXOFF	จำลองการทำงานเมื่อได้รับสัญญาณ XOFF (หยุดส่ง)
SETXON	จำลองการทำงานเมื่อได้รับสัญญาณ XON (ส่งต่อ)

การใช้งานฟังก์ชัน Escape Comm Function() ที่มักใช้กันคือบังคับให้สายสัญญาณ DTR เป็นโล ชั่วขณะหนึ่ง เพื่อทำให้โมเด็มเลิกการเชื่อมต่อ วิธีนี้จะทำงานได้ก็ต่อเมื่อโมเด็มถูกเซตให้วางสายเมื่อ DTR เปลี่ยนเป็นโล

การจัดการสัญญาณเบรก

หน้าที่หลักของสัญญาณเบรก คือ ใช้เพื่อขัดจังหวะโปรแกรมที่กำหนดงานบนคอมพิวเตอร์ระยะไกล วินโดวส์มีฟังก์ชัน Set Comm Break() เพื่อเปิดสัญญาณเบรก และฟังก์ชัน Clear comm Break() เพื่อปิดสัญญาณเบรก ทั้งสองฟังก์ชันต้องการหมายเลขอุปกรณ์ที่ได้จาก Set Comm() เป็นอาร์กิวเมนต์ (argument)

การใช้งานตามปกติมีลักษณะดังนี้

```
Set Comm Break (comdev);
Waitmillsecs(250); // รอเป็นเวลา x มิลลิวินาที
Clear comm Break (comdev);
```

การปิดพอร์ตอนุกรม

เมื่อเสร็จจากการใช้พอร์ตสื่อสารอนุกรมแล้วการปิดพอร์ตทำด้วยฟังก์ชัน Close comm() nCid เป็นหมายเลขอุปกรณ์สื่อสารที่ได้จากฟังก์ชัน Open Comm()ฟังก์ชันอื่น ๆ วินโดวส์มีฟังก์ชันที่ทำงานเกี่ยวกับการสื่อสารแบบอนุกรมอีกจำนวนมาก ดังจะได้อธิบายในหัวข้อต่อไปนี้

การล้างบัฟเฟอร์

บางครั้งเมื่อต้องการทิ้งตัวอักษรที่อยู่ในคิวรับข้อมูล หรือคิวส่งข้อมูล ก็มีฟังก์ชันที่ทำหน้าที่นี้ คือ Flush Comm() โดยมีไวยากรณ์ดังนี้

```
int Flush Comm (nCid, nQueue)
int nCid;
int nQueue;
```

nCid เป็นหมายเลขอุปกรณ์สื่อสารที่คืนมาโดย Open Comm() nQueue บอกว่าต้องการ ล้างคิวที่รับข้อมูล (nQueue = 0) หรือคิวส่งข้อมูล (nQueue = 1)

การส่งตัวอักษรด่วน

สมมุติว่าได้ส่งตัวอักษรจำนวนหนึ่งให้กับวินโดวส์เพื่อส่งออกไป แล้วมีตัวอักษรตัวหนึ่งที่ ต้องการให้ถูกส่งออกไปก่อนตัวอักษรที่อยู่ในคิวส่งข้อมูล เช่น ในกรณีที่จัดการกับ XON/XOFF ด้วยตัวเอง (แทนที่จะให้วินโดวส์ทำให้) ต้องมีการส่ง XOFF ให้เร็วที่สุดเท่าที่เป็นไปได้ ลักษณะนี้เป็นที่ของฟังก์ชัน Transmit Comm Char() ซึ่งจะนำตัวอักษรตัวหนึ่งไปใส่ไว้ที่ต้นคิว โดยมีไวยากรณ์ดังนี้

```
int Transmit Comm Char (nCid, cChar)
int nCid;
char cChar;
```

nCid เป็นหมายเลขอุปกรณ์สื่อสารที่คืนมาโดย Open Comm() cChar เป็นตัวอักษรที่ต้องการส่ง

ตัวอักษรที่ไม่ได้รับ

วินโดวส์มีฟังก์ชันหนึ่งชื่อว่า Un Get Comm Char() ใช้ในการใส่ตัวอักษรตัวหนึ่งลงในคิว รับข้อมูล เพื่อให้มันเป็นตัวอักษรตัวแรกที่ถูกอ่าน เมื่อเรียกใช้ Read comm() ในครั้งต่อไป โดย สามารถใส่ตัวอักษรลงในคิวได้เพียงตัวเดียวเท่านั้น และฟังก์ชันนี้ไม่ทำงานในวินโดวส์ 3.1 เนื่องจาก

บักในเวอร์ชันนั้น

การสื่อสารแบบกระตุ้นด้วยเหตุการณ์

ฟังก์ชันที่กล่าวถึงในบทนี้เพียงพอสำหรับการสื่อสารแบบอนุกรมภายในวินโดวส์ เมื่อทำการเปิดพอร์ตแล้ว สามารถใช้ Write Comm() เมื่อต้องการส่งข้อมูล และเรียกใช้ Read Comm() ในทุกรอบของโปรแกรมหลัก เพื่อดูว่ามีข้อมูลเข้าหรือไม่ อย่างไรก็ตาม มีอีกวิธีหนึ่งที่มีประสิทธิภาพสูงกว่า คือการเขียนโปรแกรมสื่อสารแบบกระตุ้นด้วยเหตุการณ์ (event-driven communications) โดยขอให้วินโดวส์ส่งเมจเสจ (message) มาให้เมื่อมีบางสิ่งเกิดขึ้น วิธีนี้ค่อนข้างเหมือนกับการเกิดโปรแกรมแบบกระตุ้นด้วยอินเตอร์รัปต์

วิธีขอให้วินโดวส์สร้างเหตุการณ์มีสองวิธี ขึ้นอยู่กับประเภทของเหตุการณ์ที่ต้องการกระตุ้น ให้เกิด เมสเสจ ได้แก่ การใช้ฟังก์ชัน Enable Comm Notification() ตามลำพัง หรือใช้ฟังก์ชันนี้ร่วมกัน Set Comm-Event Mask() ทั้งสองวิธีจะได้รับการอธิบายในหัวข้อต่อไป

ถ้าตั้งใจที่จะโปรแกรมการสื่อสารแบบกระตุ้นด้วยเหตุการณ์ ควรจะระวังเหตุการณ์ทางการสื่อสารที่ทราบว่าเป็นสาเหตุให้วินโดวส์ทำงานพลาดที่อัตราบอดสูง ๆ และบางครั้งเป็นสาเหตุให้ระบบถูกบูต

การแจ้งการรับหรือส่งข้อมูล

ฟังก์ชัน Enable Comm Notification() สามารถใช้เพื่อร้องขอการแจ้งเตือนเมื่อได้รับตัวอักษร หรือตัวอักษรถูกส่งออกไป โดยมีไวยากรณ์ดังนี้

```
BOOL Enable Comm Notification (nCid, hwnd, cb Write Notify, cb Out Queue)

int nCid;

HWND hwnd;
```

int cb Write Notify;

int cb Out Queue;

nCid เป็นหมายเลขอุปกรณ์สื่อสารที่ได้จาก Open Comm() hwnd เป็นแฮนเดิลของหน้าต่างที่ต้องการรับเมสเสจ cb Write Notify เป็นจำนวนตัวอักษรที่ต้องการให้มีก่อนที่วินโดวส์จะส่งเมสเสจมาบอกว่าได้รับตัวอักษร ถ้าค่านี้เป็น -1 หมายถึงไม่ต้องการให้แจ้งเตือนการรับตัวอักษร cb Out Queue เป็นจำนวนตัวอักษรน้อยที่สุดที่จะอยู่ในคิวส่งข้อมูล เมื่อวินโดวส์ส่งเมสเสจมาให้ ถ้าค่านี้ -1 หมายถึงไม่ต้องการให้แจ้งเตือนเมื่อคิวส่งข้อมูลว่างลงถึงค่าหนึ่ง

การแจ้งเหตุการณ์อื่น

ถ้าต้องการรับเมสเสจเมื่อเกิดเหตุการณ์อื่นทางการสื่อสาร สามารถใช้ฟังก์ชัน Enable Comm - Notification() ร่วมกับฟังก์ชัน Set Comm Event Mask() ตามไวยากรณ์ดังนี้

WORD AR *Set Comm Event Mask (nCid, nEvt Mask)

int nCid;

int nEvt Mask;

nCid เป็นหมายเลขอุปกรณ์การสื่อสารที่ได้จาก Open Comm() nEvt Mask() เป็นเวิร์ดที่ประกอบด้วยบิตที่บอกว่าเหตุการณ์ใดที่ต้องการให้วินโดวส์แจ้งเตือน โดยกำหนดไว้ดังต่อไปนี้

EV_BREAK การรับสัญญาณเบรก

EV_CTS CTS เปลี่ยนสถานะ

EV_DSR DSR เปลี่ยนสถานะ

EV_ERR เกิดความผิดพลาดทางเฟรม พาริตี้ หรือ โอเวอร์รัน

EV_RING ตรวจจับ Ring indicator ได้

EV_RSLD Carrier Detect เปลี่ยนสถานะ

EV_RXCHAR ได้รับตัวอักษรหนึ่งตัว

EV_RXFLAG ได้รับตัวอักษรที่กำหนดให้เป็นEvt Char ในสตรักเจอร์ DCB

EV_TXEMPTY ตัวอักษรทั้งหมดในคิวส่งข้อมูลถูกส่งออกไป

รหัสที่คืนมาจาก Set Comm Event Mask() เป็นพอยน์เตอร์ไปยังอินทีเจอร์ (interger) ซึ่งแต่ละบิตถูกเซตตามเหตุการณ์ที่เกิดขึ้น ถ้าใช้ฟังก์ชันนี้ควรเรียกใช้ Enable Comm Notification() โดยให้ cb Write Notify และ cb Out Queue เป็นศูนย์

การรับเมสเสจเกี่ยวกับการสื่อสาร

เมื่อเกิดเหตุการณ์ที่ขอให้แจ้งเตือน เมสเสจจะถูกส่งมาให้และสามารถอ่านได้จากลูปของโปรแกรม หลักด้วยวิธีปกติ คือใช้ Get Message() หรือ Peek Massage() ประเภทของเมสเสจจะเป็น WM_COMMNOTIFY วิธีจัดการกับเมสเสจจะขึ้นอยู่กับว่าได้เซตอีเวนต์มาสก์ (event mask) ด้วยฟังก์ชัน
Set Comm Event Mask() หรือไม่

การจัดการเมสเสจด้วยอีเวนต์มาสก์

ถ้าใช้อีเวนต์มาสก์ เวิร์ดล่างของพารามิเตอร์ 1Param ที่เกี่ยวข้องกับเมสเสจนั้นจะมีบิต CN_EVENT ถูกเซต การหาว่าเหตุการณ์ใดเป็นตัวกระตุ้นเมสเสจ และรีเซตมาสก์เพื่อรับการแจ้งเตือนครั้งต่อไป ใช้ฟังก์ชันGet Comm Event Mask()

ฟังก์ชัน Get Comm Event Mask() รีเซตแฟล็กเพื่อตอบรับว่าได้รับเมสเสจแล้ว ถ้าเราไม่เรียกใช้ฟังก์ชันนี้ เราจะไม่ได้รับการแจ้งเตือนเหตุการณ์ที่เกิดขึ้นในภายหลัง ลักษณะนี้คล้ายคลึงกับการตอบรับ อินเตอร์รัปต์

เมื่อได้รับเมสเสจแล้ว จะเข้าสู่ลูปเพื่อทำการอ่านพอร์ตสื่อสารจนกระทั่งได้รับเมสเสจอื่น จากนั้นจึงตรวจสอบข้อผิดพลาดและดูที่สตรักเจอร์ COMSTAT ซึ่งคืนมาจาก Get Comm Error() ถ้าคิวรับข้อมูลไม่ว่างจะสร้างเหตุการณ์หลอกขึ้นมา โดยการส่งเมสเสจไปที่หน้าต่างของเราก่อนที่จะออกจากโปรแกรมส่วนนี้

การจัดการเมสเสจโดยไม่ใช้อีเวนต์มาสก์

ถ้าไม่ใช้อีเวนต์มาสก์ เวิร์ดล่างของพารามิเตอร์ 1Param ที่เกี่ยวข้องกับเมสเสจจะมีบิต CN_RECEIVE หรือ CN_TRANSMIT ถูกเซต เพื่อบ่งบอกว่ามีเหตุการณ์เกิดขึ้น เมื่อได้รับเมสเสจแล้ว จะเข้าสู่ลูปเพื่ออ่านพอร์ตสื่อสารจนกระทั่งได้รับเมสเสจอื่น จากนั้นจึงตรวจสอบข้อผิดพลาดและดูที่สตรักเจอร์ COMSTAT ซึ่งคืนมาจาก Get Comm Error() ถ้าคิวรับข้อมูลไม่ว่างจะสร้างเหตุการณ์หลอกขึ้นมาโดยการส่งเมสเสจไปยังหน้าต่างของเราก่อนที่จะออกจากโปรแกรมส่วนนี้

การออกแบบโปรแกรมในโครงงานนี้

1. ในการออกแบบโปรแกรมในส่วนของการสื่อสารข้อมูลนั้น ตามหลักการโปรแกรมบนระบบปฏิบัติการ WINDOWS นั้น ผู้ออกแบบสามารถจัดการในส่วนของ APPLICATION เท่านั้น ในส่วนของระบบของ WINDOWS โดยการเรียกใช้งานผ่าน port สื่อสารนั้นจะมีความเกี่ยวข้องกับเรียกใช้ function เป็นมาตรฐาน เรียกว่า API Function API (APPLICATION PROGRAMMING INTERFACE) ซึ่งการเรียกใช้ API Function นั้นมีมาตรฐานการประกาศเรียกใช้อยู่เพื่อใช้จัดการกับการโปรแกรมที่ต้องยุ่งเกี่ยวกับระบบของ WINDOWS คล้ายกับการเรียก funiction int (Interrupt) บน DOS โดยมาตรฐานการประกาศ API ต่าง ๆ ได้มีอยู่ในส่วนของโปรแกรมแล้วจึงไม่ขอกล่าวซ้ำ

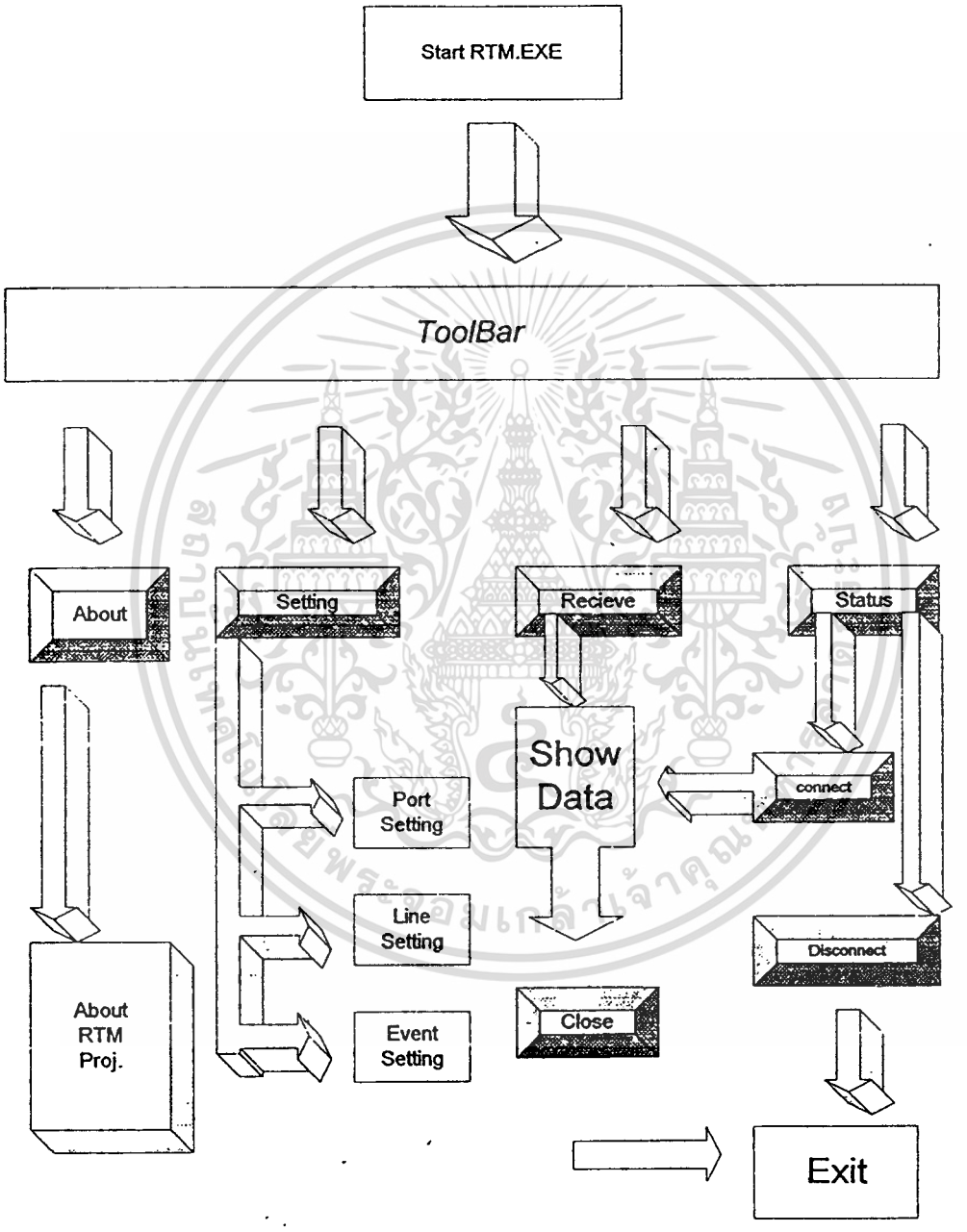
2. ในส่วนของอัตราการสื่อสารนั้น ผู้จัดได้ตั้งไว้เป็นค่า Default แต่มีการทำเผื่อสำหรับในการติดต่อผ่านโมเด็ม เพื่อสำหรับปรับเปลี่ยนค่าได้ ซึ่งต้องรอให้การพัฒนาโมเด็มสำเร็จ จึงจะทำให้ Software ในส่วนนี้จึงแล้วเสร็จได้ตามลำดับ

3. ในส่วนของการแยกช่วงสัญญาณเพื่อแยกตรวจรับข้อมูลที่ส่งมาแต่ละ Terminal เนื่องจากเป็นการสื่อสารแบบ Serial Communication จึงสามารถส่งข้อมูลได้ไม่จำกัด และการแยกช่วงสัญญาณในส่วนของโปรแกรมตัวนี้ เนื่องมาจากรูปแบบการโปรแกรมในส่วนของการตรวจรับเป็นลักษณะ array จึงสามารถเพิ่มช่องสัญญาณได้ แต่ในส่วน Terminal มีการจัดส่ง 8 channel เพื่อเป็น Prototype จึงจัดแสดง 8 channel ถ้าหากมีส่วนมีส่วน Terminal เพิ่มขึ้น โปรแกรมก็สามารถรองรับการขยายได้เพิ่มขึ้นในอนาคต

4. ส่วนการโปรแกรมเชิงกราฟฟิกแบบกราฟ มีการนำ Control Graph มาประยุกต์ใช้ โดยเรียกไฟล์ Graph VBX เข้าโปรเจ็กคอนโทรล graph จะเป็นคอนโทรล หรือเครื่องมือในการจัดการกับกราฟในรูปแบบต่าง ๆ ของ VB โดยในการ update ข้อมูลต่าง ๆ ของ Control นี้ใช้ mehtod ที่ชื่อ Quickdata มาใช้สำหรับเรียกข้อมูลที่ส่งเข้ามาจาก Terminal เพื่อแสดงผลในรูปแบบกราฟ

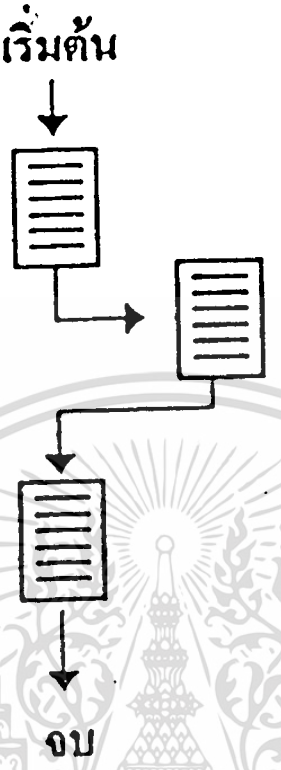
5. สำหรับการทำ function หรือการทำ Subrontine นั้นเกิดจากการเรียกใช้งานที่เมนูบาร์ View ตรง New Procedure เพื่อการเพิ่มเติม function ย่อยหรือ Subroutine ย่อย

6. ในการสร้างตัวแปรแบบ array สำหรับ VB ทำโดยการเลือก object ที่ต้องการทำให้เป็น array แล้วเลือก เมนู Edit เลือก Copy VB จะถามว่าเราต้องการทำ array หรือเปล่า ให้ตอบ yes แล้ว VB จะทำให้เอง จึงเลือก Plate ตามที่ต้องการได้

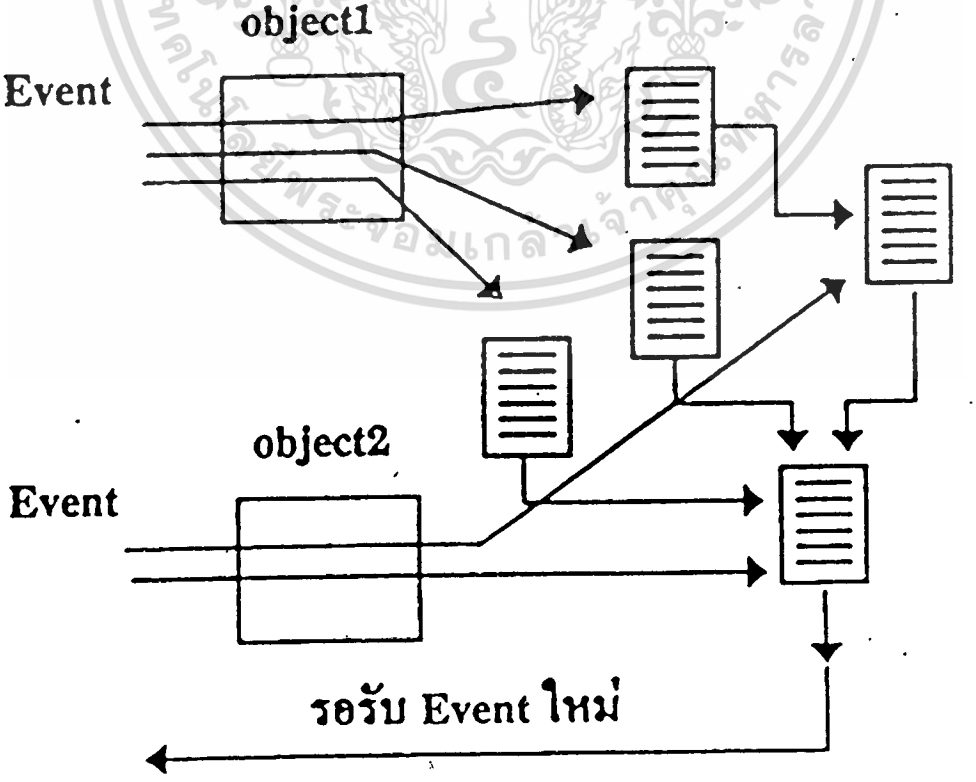


รูปที่ 3.1 รูปแสดงบล็อกไดอะแกรม โปรแกรม RTM.EXE

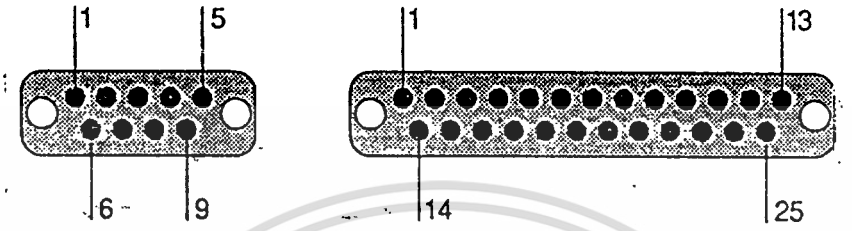
แสดงการเขียนโปรแกรมแบบธรรมดาเปรียบเทียบกับการเขียนโปรแกรมแบบ Event-Driven.



รูปที่ 3.2 แสดงการเขียนโปรแกรมแบบธรรมดา

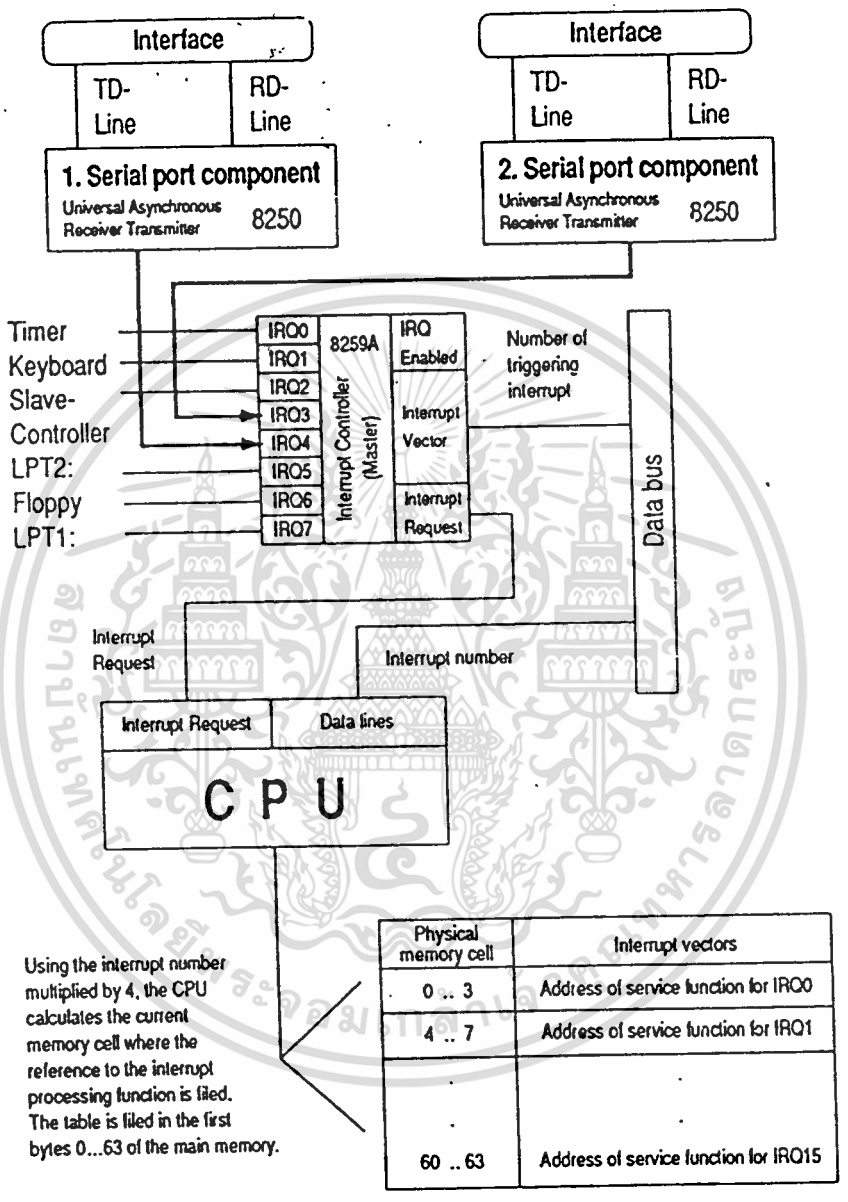


รูปที่ 3.3 การเขียนโปรแกรมแบบ Event-Driven

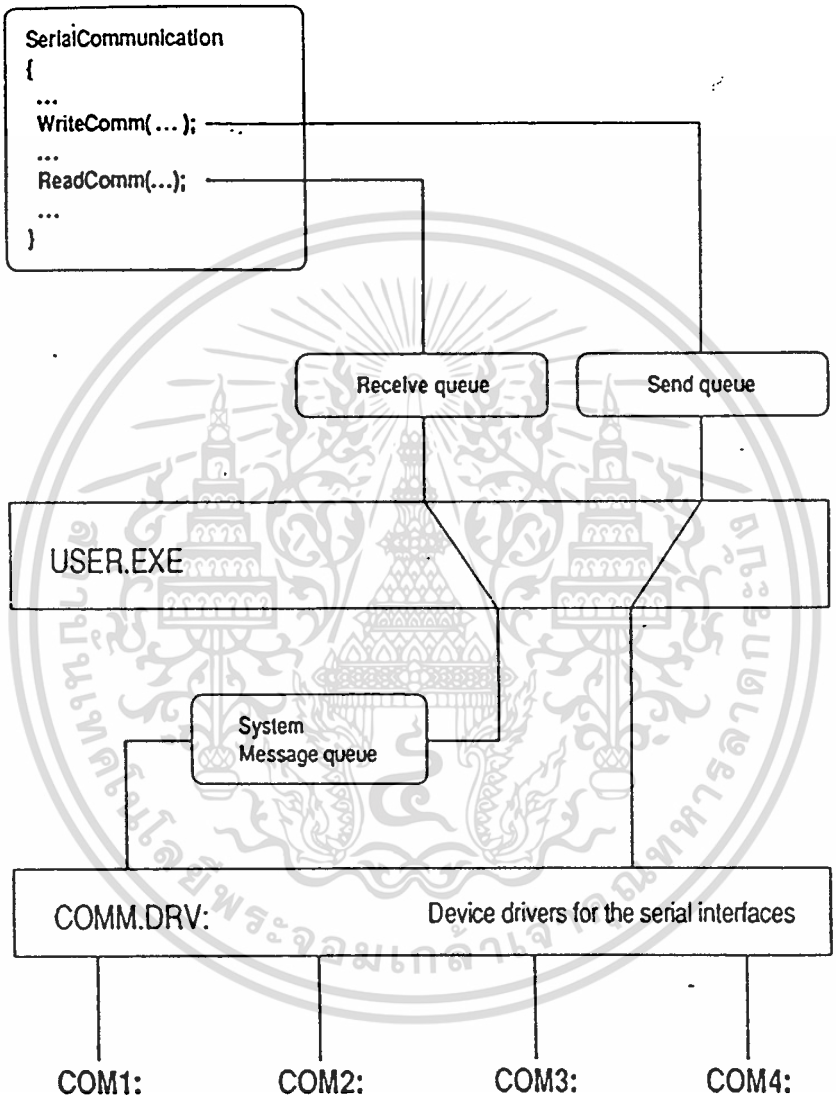


9-pin	EIA label	Signal label		25-pin
1	DCD	Data carrier detect		8
2	RD	Receive data		3
3	TD	Transmit data		2
4	DTR	Data terminal ready		20
5	GND	Signal ground		7
6	DSR	Data set ready		6
7	RTS	Request to send data		4
8	CTS	Clear to send		5
9	RI	Ring indicator		22

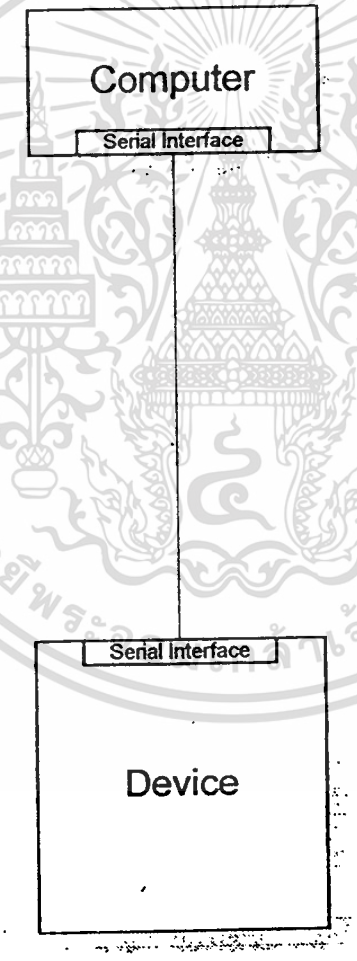
รูปที่ 3.4 รูปแสดงส่วน Pin Layout ของ RS-232



รูปที่ 3.5 รูปแสดงการทำงาน Interrup บนเครื่อง PC



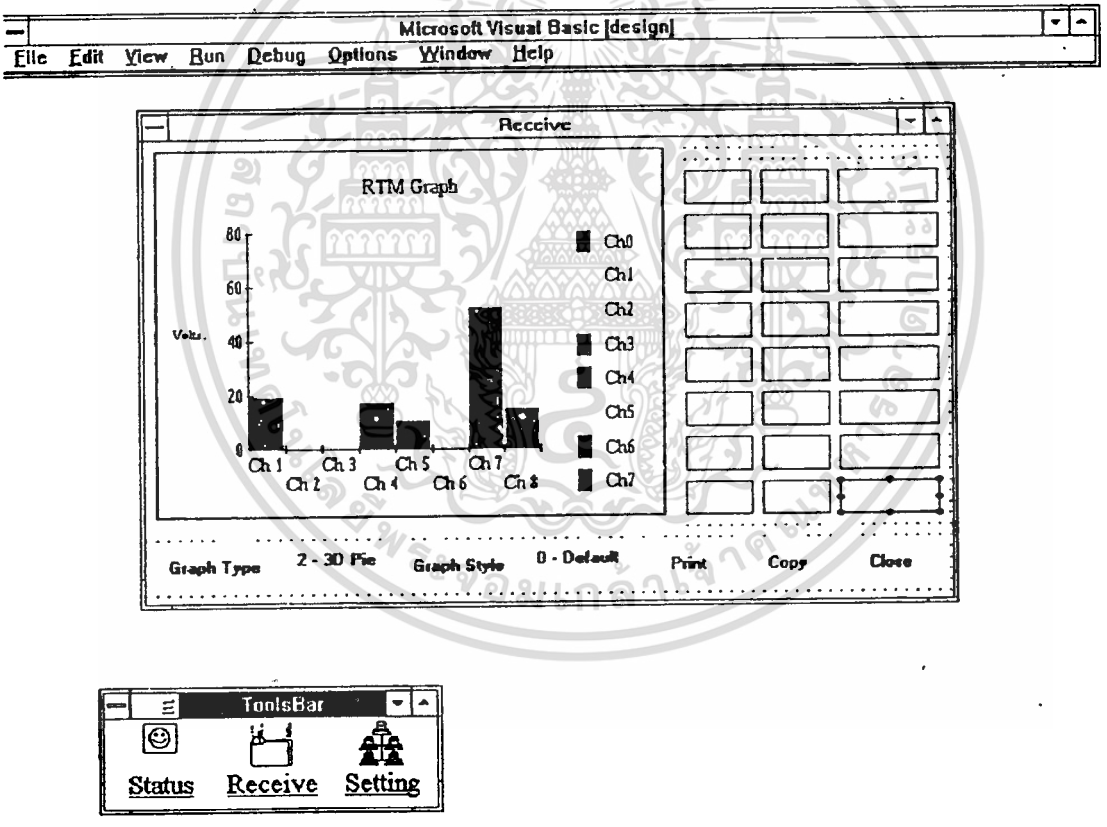
รูปที่ 3.6 รูปแสดงการสื่อสารแบบอนุกรม



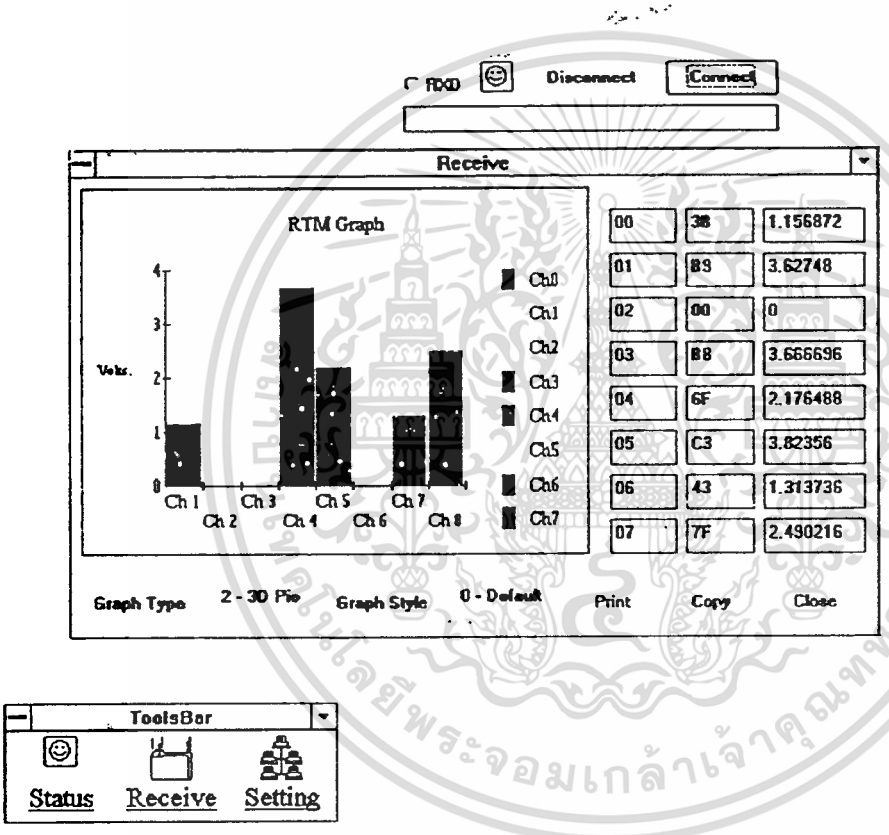
รูปที่ 3.7 Serial Communication theory states that all communication take place over one wire

บทที่ 4

ผลการทดลอง

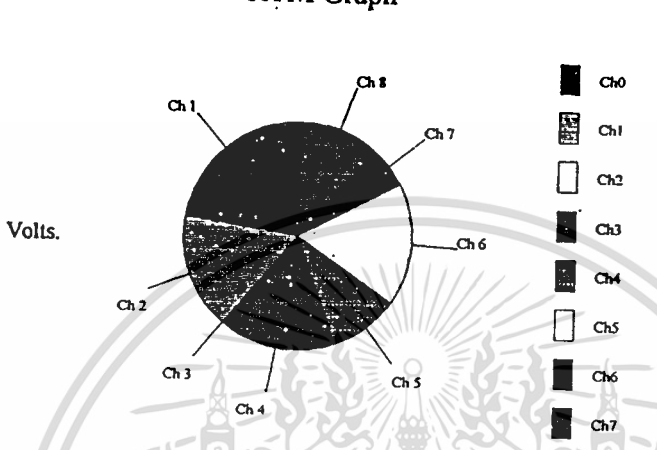


รูปที่ 4.1 รูปแสดงผลการทดลองหน้าจอมอนิเตอร์ก่อนรับสัญญาณ

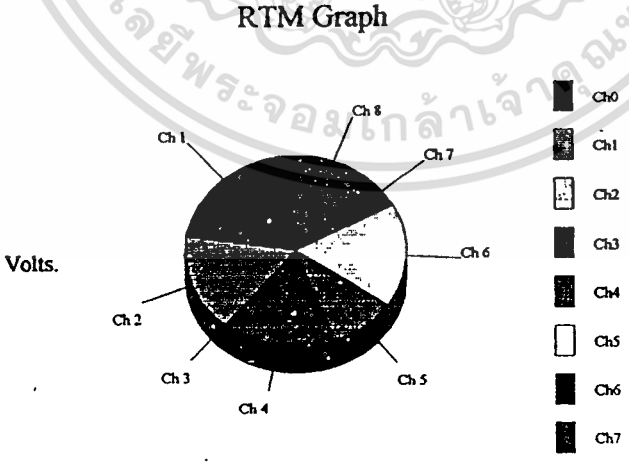


รูปที่ 4.2 รูปแสดงผลการทดลองหลังจากได้รับสัญญาณ

แสดงผลการทดลองลักษณะกราฟแบบต่างๆ หลังการรับสัญญาณ Graph Style = Defalut.

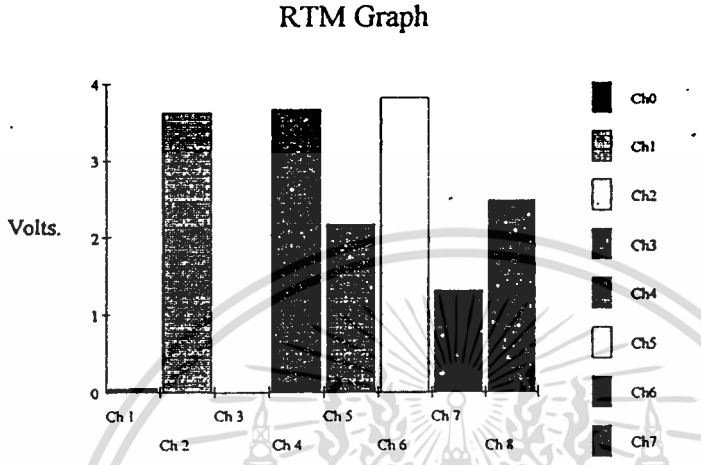


รูปที่ 4.3 Graph Type = 2D Pie

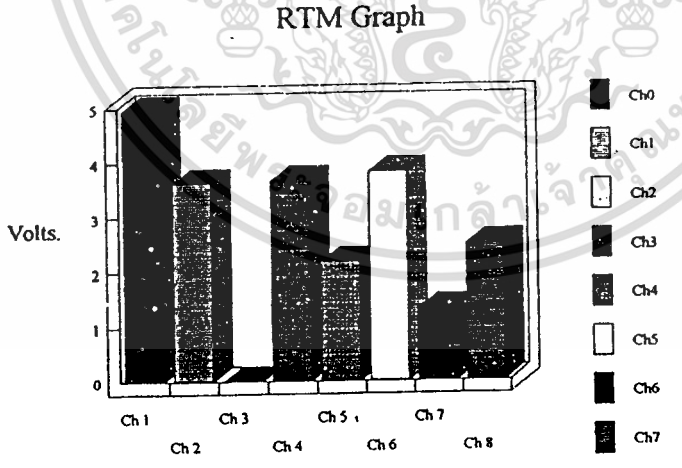


รูปที่ 4.4 Graph Type 3D Pie

แสดงผลการทดลองลักษณะกราฟแบบต่างๆ หลังการรับสัญญาณ Graph Style = Defalut.

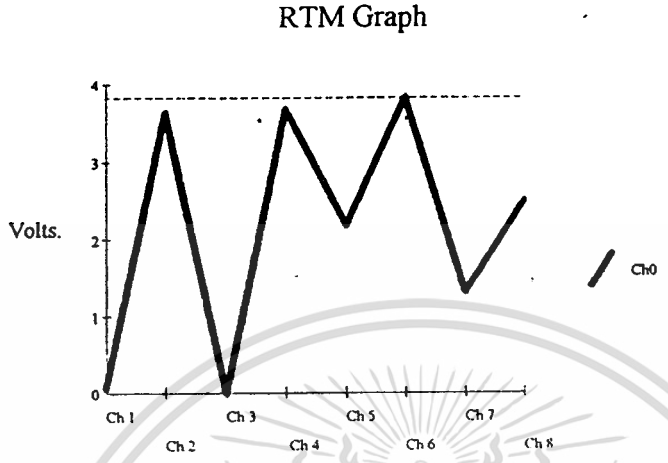


รูปที่ 4.5 Graph Type = 2D Bar

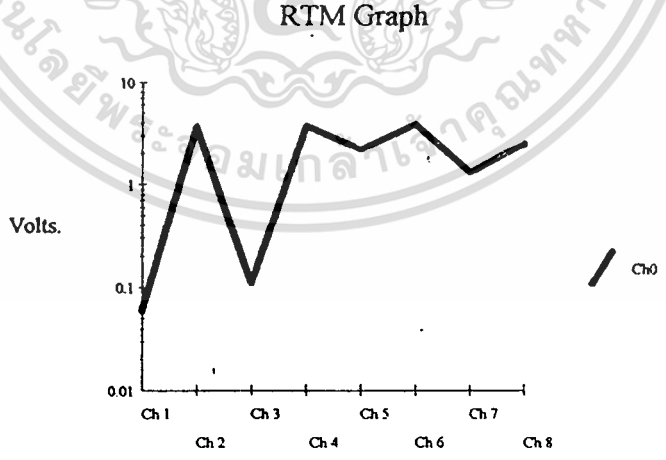


รูปที่ 4.6 Graph Type 3D Bar

แสดงผลการทดลองลักษณะกราฟแบบต่างๆ หลังการรับสัญญาณ Graph Style = Defalut.

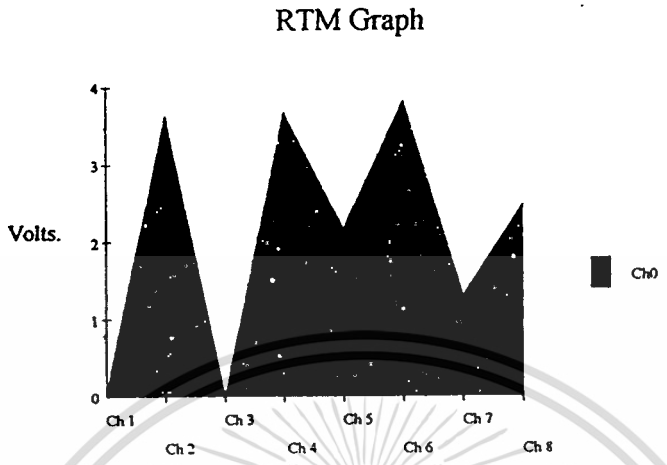


รูปที่ 4.7 Graph Type = Line

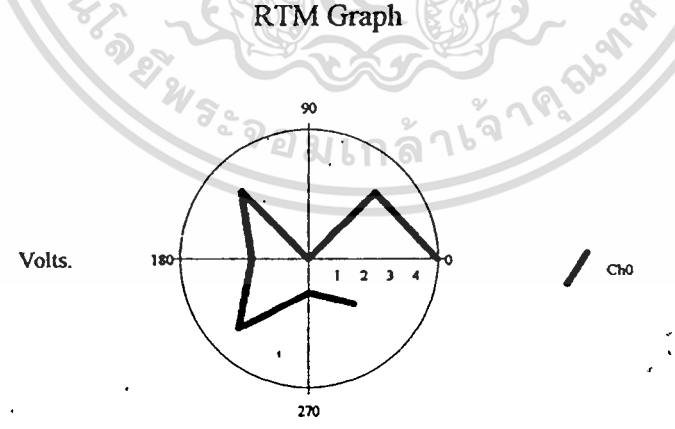


รูปที่ 4.8 Graph Type Log/Line

แสดงผลการทดลองลักษณะกราฟแบบต่างๆ หลังการรับสัญญาณ Graph Style = Defalut.



รูปที่ 4.9 Graph Type = Area



รูปที่ 4.10 Graph Type Area

เอกสารอ้างอิง

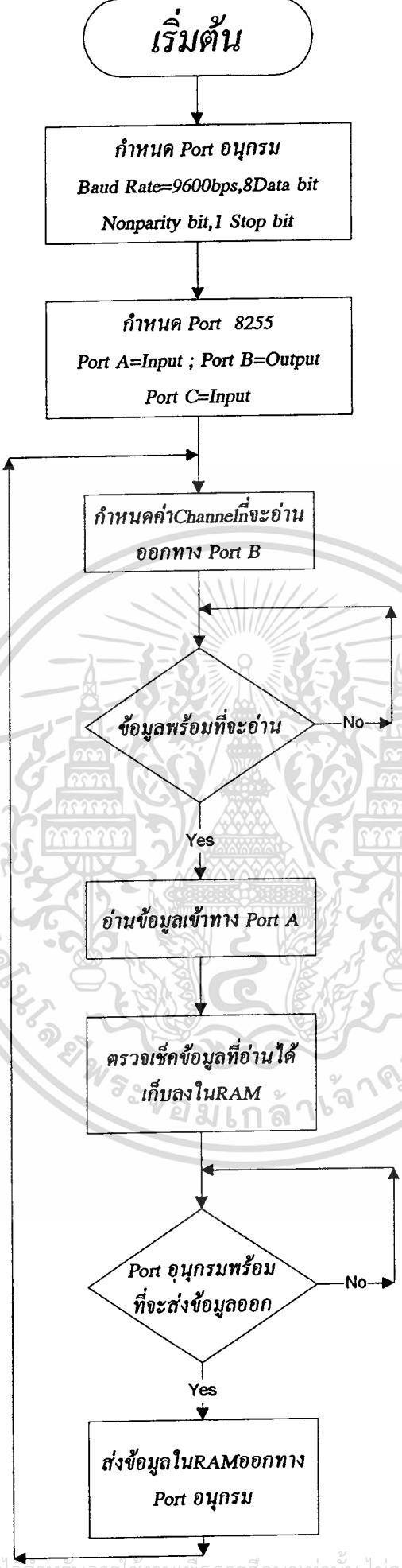
1. รศ.ยืน ภู่วรวรรณ,"ไมโครโปรเซสเซอร์ ไมโครคอมพิวเตอร์",ภาควิชาวิศวกรรมไฟฟ้า,คณะวิศวกรรศาสตร์,มหาวิทยาลัยเกษตรศาสตร์.
2. ผศ.ประทีป บัญญัตินพรัตน์,"การเขียนโปรแกรมภาษาแอสเซมบลี้ Z80,ภาควิชาวิศวกรรมคอมพิวเตอร์,คณะวิศวกรรมศาสตร์,สถาบันเทคโนโลยี่พระจอมเกล้า,วิทยาเขตเจ้าคุณทหารลาดกระบัง,กรุงเทพฯ.
3.จิรพัฒน์ จันทร์เจิดศักดิ์,วีระนพนิราพาธ,การเขียนโปรแกรมบนไมโครซอฟวินโดวส์,บริษัทซีเอ็ดยูเคชั่นจำกัด.
4. ยืน ภู่วรวรรณ และ สุรศักดิ์ สงวนพงษ์,"โปรแกรมคอมพิวเตอร์ ภาษาแอสแซมบลี้ 8086/8088,กรุงเทพฯ,บริษัท ซีเอ็ดยูเคชั่น จำกัด,พิมพ์ครั้งที่ 2/2537.
5. "คู่มือไอซี CMOS 4000 SERIES",กรุงเทพฯ,บริษัท ซีเอ็ดยูเคชั่น จำกัด,พิมพ์ครั้งที่ 3/2530.
6. "Liner Integrated Circuit",Nationnal Semiconductor Corporation,1982.
7. Peter W. Gofton,Mastering Serial Communication,SYBEX.
8. Steve Potts,Michael Mckelvy,Edward B.Toupin,Michael Marchuk,James A.Dooley Joseph Armitage,Elisabeth Boonin,Andrew Dean,Steven List,S.RamaRama Chandran J.D Evans,Jr.,Brain Blackman,Jon Oelschlaeger,Dr.David Fullerton,Visual Basic Expert Solutions,QUE.
9. Mark Warhol,The Art of Programing with Visual Basic,Wiley.
10. John Clark Craig, Visual Basic Workshop, Microsoft Press.
11. Phil Feldman, Roger Jennings, Barry Seymour, Bob Edison, Pam Palmer, Steve Gillmor,Jack Pesso,QUE.
12. Peter Wilken, Dirk Hone kanop, Windows System Programing, Abacus.

ภาคผนวก

ภาคผนวก ก.

โปรแกรม

1. แสดงโปรแกรมของเครื่อง Terminal ซึ่งใช้ในการทำงานของเครื่องโดยใช้โปรแกรม Assembler ของ Z80180.
2. แสดงโปรแกรมในส่วนของคอมพิวเตอร์แสดงผลซึ่งเป็นส่วนของ Monitoring โดยใช้โปรแกรม Visual Basic.



รูปแสดงขั้นตอนการทำงานของโปรแกรมควบคุมการทำงานของ Remote Terminal.

```
000001 0000              ; ***********************************
000002 0000              ; *  SOFT-WARE Z80180 FOR REMOTE *
000003 0000              ; ***********************************
000004 0000              ;
000005 0000              ;ROM monitor program 00000-07FFFH
000006 0000              ;RAM for use 10000,17EBFH
000007 0000              ;RAM stack for user & system 17EC0-18000H
000008 0000              ;PORT 8255 (A=input,B=output,C=input)
000009 0000              ;
000010 0000                    .ORG 0000H
000011 0000              ;
000012 0000              ;*** REGTER I/O Z80180 ***
000013 0000              ;
000014 0007                    .EQU TDR1,7      ;ASCI CHANEL1 TX
000015 0009                    .EQU RDR1,9      ;RX
000016 0005                    .EQU STAT1,5     ;STATUS
000017 0001                    .EQU CNTLA1,1    ;DATA FORMT
000018 0003                    .EQU CNTLB1,3    ;BAUD RATE
000019 0000              ;
000020 003A                    .EQU CBAR,3AH    ;SET LOGICAL
000021 0039                    .EQU BBR,39H     ;BANK BASE
000022 0038                    .EQU CBR,38H     ;COMMON AREA 1
000023 0000              ;
000024 FEC0                    .EQU STACK,0FEC0H
000025 0083                    .EQU PCTRL,83H   ;PORT CONRTOL 8255
000026 0080                    .EQU PDATA,80H   ;PORT A OF 8255 (INPUT)
000027 0081                    .EQU PDATB,81H   ;PORT B OF 8255 (OUTPUT)
000028 0082                    .EQU PDATC,82H   ;PORT C OF 8255 (OUTPUT)
000029 0000              ;
000030 0000              ; ******************************
000031 0000              ; *      POWER PU DELAY        *
000032 0000              ; ******************************
000033 0000              ;
000034 0000 AF                 START:  XOR A            ;POWER UP
000035 0001 00                 START1: NOP
000036 0002 3D                 DEC A
000037 0003 C20100             JP NZ,START1
000038 0006 C30900             JP INIT
000039 0009              ;
000040 0009              ;***********************************
000041 0009              ;*       INITIAL PARAMETER         *
000042 0009              ;***********************************
000043 0009              ;
000044 0009 3EF8          INIT:        LD A,0F8H        ;LOGICAL ADDRESS
000045 000B ED393A                     OUT0 (CBAR),A
000046 000E 3E08                       LD A,8H
000047 0010 ED3939                     OUT0 (BBR),A     ;PSYSICAL BANK 18000H
000048 0013 ED3938                     OUT0 (CBR),A     ;PSYICAL COMMON AREA1
000049 0016              ;
```

```
000050 0016 31C0FE              LD SP,STACK      ;LOAD STACK FOR CALL
PROGRAM
000051 0019 CD9601              CALL ASCSET      ;SET SERIAL PORT 1
000052 001C             ;
000053 001C 3E99                LD A,99H         ;SET 8255 A,B,C (IN,OUT,OUT)
000054 001E D383                OUT (PCTRL),A
000055 0020             ;
000056 0020             ; ***************************
000057 0020             ; *      MAIN PROGRAM       *
000058 0020             ; ***************************
000059 0020             ;
000060 0020 3E48          MAIN: LD A,48H
000061 0022 320190              LD (9001H),A
000062 0025 3E31                LD A,31H
000063 0027 320290              LD (9002H),A
000064 002A 3E00                LD A,00H
000065 002C D381                OUT (81H),A
000066 002E 3ED0                LD A,0D0H        ;MUTIPLEX CH1
000067 0030 D381                OUT (81H),A
000068 0032 3E10                LD A,10H
000069 0034 D381                OUT (81H),A
000070 0036 CDC101              CALL ENABLE
000071 0039 3E30                LD A,30H
000072 003B D381                OUT (81H),A
000073 003D DB80                IN A,(80H)       ;INPUT CH1
000074 003F CDC901              CALL HEXASCII
000075 0042 3A0191              LD A,(9101H)
000076 0045 320390              LD (9003H),A
000077 0048 3A0291              LD A,(9102H)
000078 004B 320490              LD (9004H),A
000079 004E             ;
000080 004E 3E48                LD A,48H
000081 0050 320590              LD (9005H),A
000082 0053 3E32                LD A,32H
000083 0055 320690              LD (9006H),A
000084 0058 3E01                LD A,01H
000085 005A D381                OUT (81H),A
000086 005C 3ED1                LD A,0D1H        ;MUTIPLEX CH2
000087 005E D381                OUT (81H),A
000088 0060 3E11                LD A,11H
000089 0062 D381                OUT (81H),A
000090 0064 CDC101              CALL ENABLE
000091 0067 3E31                LD A,31H
000092 0069 D381                OUT (81H),A
000093 006B DB80                IN A,(80H)       ;INPUT CH2
000094 006D CDC901              CALL HEXASCII
000095 0070 3A0191              LD A,(9101H)
000096 0073 320790              LD (9007H),A
000097 0076 3A0291              LD A,(9102H)
000098 0079 320890              LD (9008H),A
```

```
000099 007C             ;
000100 007C 3E48                LD A,48H
000101 007E 320990              LD (9009H),A
000102 0081 3E33                LD A,33H
000103 0083 320A90              LD (900AH),A
000104 0086 3E02                LD A,02H
000105 0088 D381                OUT (81H),A
000106 008A 3ED2                LD A,0D2H       ;MUTIPLEX CH3
000107 008C D381                OUT (81H),A
000108 008E 3E12                LD A,12H
000109 0090 D381                OUT (81H),A
000110 0092 CDC101              CALL ENABLE
000111 0095 3E32                LD A,32H
000112 0097 D381                OUT (81H),A
000113 0099 DB80                IN A,(80H)      ;INPUT CH3
000114 009B CDC901              CALL HEXASCII
000115 009E 3A0191              LD A,(9101H)
000116 00A1 320B90              LD (900BH),A
000117 00A4 3A0291              LD A,(9102H)
000118 00A7 320C90              LD (900CH),A
000119 00AA             ;
000120 00AA 3E48                LD A,48H
000121 00AC 320D90              LD (900DH),A
000122 00AF 3E34                LD A,34H
000123 00B1 320E90              LD (900EH),A
000124 00B4 3E03                LD A,03H
000125 00B6 D381                OUT (81H),A
000126 00B8 3ED3                LD A,0D3H       ;MUTIPLEX CH4
000127 00BA D381                OUT (81H),A
000128 00BC 3E13                LD A,13H
000129 00BE D381                OUT (81H),A
000130 00C0 CDC101               CALL ENABLE
000131 00C3 3E33                LD A,33H
000132 00C5 D381                OUT (81H),A
000133 00C7 DB80                IN A,(80H)      ;INPUT CH4
000134 00C9 CDC901              CALL HEXASCII
000135 00CC 3A0191              LD A,(9101H)
000136 00CF 320F90              LD (900FH),A
000137 00D2 3A0291              LD A,(9102H)
000138 00D5 321090              LD (9010H),A
000139 00D8             ;
000140 00D8 3E48                LD A,48H
000141 00DA 321190              LD (9011H),A
000142 00DD 3E35                LD A,35H
000143 00DF 321290              LD (9012H),A
000144 00E2 3E04                LD A,04H
000145 00E4 D381                OUT (81H),A
000146 00E6 3ED4                LD A,0D4H       ;MUTIPLEX CH5
000147 00E8 D381                OUT (81H),A
000148 00EA 3E14                LD A,14H
```

```
000149 00EC D381        OUT (81H),A
000150 00EE CDC101      CALL ENABLE
000151 00F1 3E34        LD A,34H
000152 00F3 D381        OUT (81H),A
000153 00F5 DB80        IN A,(80H)      ;INPUT CH5
000154 00F7 CDC901      CALL HEXASCII
000155 00FA 3A0191      LD A,(9101H)
000156 00FD 321390      LD (9013H),A
000157 0100 3A0291      LD A,(9102H)
000158 0103 321490      LD (9014H),A
000159 0106             ;
000160 0106 3E48        LD A,48H
000161 0108 321590      LD (9015H),A
000162 010B 3E36        LD A,36H
000163 010D 321690      LD (9016H),A
000164 0110 3E05        LD A,05H
000165 0112 D381        OUT (81H),A
000166 0114 3ED5        LD A,0D5H       ;MUTIPLEX CH6
000167 0116 D381        OUT (81H),A
000168 0118 3E15        LD A,15H
000169 011A D381        OUT (81H),A
000170 011C CDC101      CALL ENABLE
000171 011F 3E35        LD A,35H
000172 0121 D381        OUT (81H),A
000173 0123 DB80        IN A,(80H)      ;INPUT CH6
000174 0125 CDC901      CALL HEXASCII
000175 0128 3A0191      LD A,(9101H)
000176 012B 321790      LD (9017H),A
000177 012E 3A0291      LD A,(9102H)
000178 0131 321890      LD (9018H),A
000179 0134             ;
000180 0134 3E48        LD A,48H
000181 0136 321990      LD (9019H),A
000182 0139 3E37        LD A,37H
000183 013B 321A90      LD (901AH),A
000184 013E 3E06        LD A,06H
000185 0140 D381        OUT (81H),A
000186 0142 3ED6        LD A,0D6H       ;MUTIPLEX CH7
000187 0144 D381        OUT (81H),A
000188 0146 3E16        LD A,16H
000189 0148 D381        OUT (81H),A
000190 014A CDC101      CALL ENABLE
000191 014D 3E36        LD A,36H
000192 014F D381        OUT (81H),A
000193 0151 DB80        IN A,(80H)      ;INPUT CH7
000194 0153 CDC901      CALL HEXASCII
000195 0156 3A0191      LD A,(9101H)
000196 0159 321B90      LD (901BH),A
000197 015C 3A0291      LD A,(9102H)
000198 015F 321C90      LD (901CH),A
```

```
000199 0162                ;
000200 0162 3E48                LD A,48H
000201 0164 321D90              LD (901DH),A
000202 0167 3E38                LD A,38H
000203 0169 321E90              LD (901EH),A
000204 016C 3E07                LD A,07H
000205 016E D381                OUT (81H),A
000206 0170 3ED7                LD A,0D7H       ;MUTIPLEX CH8
000207 0172 D381                OUT (81H),A
000208 0174 3E17                LD A,17H
000209 0176 D381                OUT (81H),A
000210 0178 CDC101              CALL ENABLE
000211 017B 3E37                LD A,37H
000212 017D D381                OUT (81H),A
000213 017F DB80                IN A,(80H)      ;INPUT CH8
000214 0181 CDC901              CALL HEXASCII
000215 0184 3A0191              LD A,(9101H)
000216 0187 321F90              LD (901FH),A
000217 018A 3A0291              LD A,(9102H)
000218 018D 322090              LD (9020H),A
000219 0190 CDB201              CALL SENT
000220 0193 C32000              JP MAIN
000221 0196                ;
000222 0196                ; *** SET ASCI CHANEL1 TX,RX,8,N,1,9600, AT X'TAL 12.488
***
000223 0196                ;
000224 0196 3E64           ASCSET: LD A,64H
000225 0198 ED3901              OUT0 (CNTLA1),A  ;TX,RX8BIT,1 STOP
000226 019B 3E02                LD A,2           ;9600 BAUD AT X'TAL 12.488
000227 019D ED3903              OUT0 (CNTLB1),A
000228 01A0 3E08                LD A,8           ;RIE ENABLE TO KEYBOARD
000229 01A2 ED3905              OUT0 (STAT1),A
000230 01A5 C9                  RET
000231 01A6                ;
000232 01A6 ED1805         SENTOUT: IN0 E,(STAT1)
000233 01A9 CB4B                BIT 1,E
000234 01AB CAA601              JP Z,SENTOUT
000235 01AE ED3907              OUT0 (TDR1),A
000236 01B1 C9                  RET
000237 01B2 210190         SENT:   LD HL,9001H
000238 01B5 7E             SENT1:  LD A,(HL)
000239 01B6 CDA601              CALL SENTOUT
000240 01B9 23                  INC HL
000241 01BA 7D                  LD A,L
000242 01BB FE21                CP 21H
000243 01BD C2B501              JP NZ,SENT1
000244 01C0 C9                  RET
000245 01C1                ;
000246 01C1 DB82           ENABLE: IN A,(82H)
000247 01C3 CB47                BIT 0,A
```

```
000248 01C5 CAC101          JP Z,ENABLE
000249 01C8 C9              RET
000250 01C9 320091  HEXASCII: LD (9100H),A
000251 01CC 3A0091          LD A,(9100H)
000252 01CF CB2F            SRA A
000253 01D1 CB2F            SRA A
000254 01D3 CB2F            SRA A
000255 01D5 CB2F            SRA A
000256 01D7 CDE701          CALL ASCII0
000257 01DA 320191          LD (9101H),A
000258 01DD 3A0091          LD A,(9100H)
000259 01E0 CDE701          CALL ASCII0
000260 01E3 320291          LD (9102H),A
000261 01E6 C9              RET
000262 01E7 E60F    ASCII0: AND 0FH
000263 01E9 FE0A            CP 0AH
000264 01EB DAF001          JP C,ASCII1
000265 01EE CE07            ADC A,07H
000266 01F0 C630    ASCII1: ADD A,30H
000267 01F2 C9              RET
000268 01F3                 .END
```

```
    Type CommstateDCB
    Id          As String * 1   ' Port Id from OpenComm
    BaudRate    As Integer      ' Baud Rate
    ByteSize    As String * 1   ' Data Bit Size (4 to 8)
    Parity      As String * 1   ' Parity
    StopBits    As String * 1   ' Stop Bits
    RlsTimeOut  As Integer      ' Carrier Detect Time "CD"
    CtsTimeOut  As Integer      ' Clear-to-Send Time
    DsrTimeOut  As Integer      ' Data-Set-Ready Time
    ModeControl As Integer      ' Mode Control Bit Fields
    XonChar     As String * 1   ' XON character
    XoffChar    As String * 1   ' XOFF character
    XonLim      As Integer      ' Min characters in buffer before XON is sent
    XoffLim     As Integer      ' Max characters in buffer before XOFF is send
    PeChar      As String * 1   ' Parity Error Character
    EofChar     As String * 1   ' EOF/EOD character
    EvtChar     As String * 1   ' Event character
    TxDelay     As Integer      ' Reserved/Not Used
End Type

Declare Function OpenComm Lib "user" (ByVal a As String, ByVal b As Integer, ByVal c As
Integer) As Integer
Declare Function CloseComm Lib "user" (ByVal a As Integer) As Integer

Declare Function WriteComm Lib "user" (ByVal a As Integer, ByVal b As String, ByVal c As
Integer) As Integer
Declare Function ReadComm Lib "user" (ByVal a As Integer, ByVal b As String, ByVal c As
Integer) As Integer

Declare Function GetCommEventMask Lib "user" (ByVal a As Integer, ByVal b As Integer)
As Integer
```

```
Declare Function SetCommEventMask Lib "user" (ByVal a As Integer, ByVal b As Integer)
As Integer

Declare Function SetCommState Lib "user" (b As CommstateDCB) As Integer
Declare Function GetCommState Lib "user" (ByVal a As Integer, b As CommstateDCB) As
Integer

Declare Function RemoveMenu Lib "User" (ByVal hMenu As Integer, ByVal nPosition As
Integer, ByVal wFlags As Integer) As Integer
Declare Function GetSystemMenu Lib "User" (ByVal hWnd As Integer, ByVal Action As
Integer) As Integer


' COMM OPEN Error Numbers

Global Const IE_BADID = -1       ' Invalid or unsupported id
Global Const IE_OPEN = -2        ' Device Already Open
Global Const IE_NOPEN = -3       ' Device Not Open
Global Const IE_MEMORY = -4      ' Unable to allocate queues
Global Const IE_DEFAULT = -5     ' Error in default parameters
Global Const IE_HARDWARE = -10   ' Hardware Not Present
Global Const IE_BYTESIZE = -11   ' Illegal Byte Size
Global Const IE_BAUDRATE = -12   ' Unsupported BaudRate


' COMM EVENT MASK

Global Const EV_RXCHAR = &H1
Global Const EV_RXFLAG = &H2
Global Const EV_TXEMPTY = &H4
Global Const EV_CTS = &H8
Global Const EV_DSR = &H10
Global Const EV_RLSD = &H20
```

```
Global Const EV_BREAK = &H40
Global Const EV_ERR = &H80
Global Const EV_RING = &H100
Global Const EV_PERR = &H200
Global Const EV_ALL = &H3FF

Global CommHandle As Integer
Global CommDeviceNum As Integer

Global CommPortName As String
Global PostPortName As String

Global CommEventMask As Integer
Global PostEventMask As Integer

Global CommState As CommstateDCB
Global PostState As CommstateDCB

Global CommRBBuffer As Integer
Global PostRBBuffer As Integer

Global CommTBBuffer As Integer
Global PostTBBuffer As Integer

Global CommReadInterval As Integer
Global PostReadInterval As Integer
'Global channel As String
Global to_detect As String
'Global reduce As Integer
'Global temp As String
'Global ch As String
'Global level As String
```

MODULE.BAS

```
Sub CenterDialog (A_Form As Form)

    Dim cLeft As Integer
    Dim cTop As Integer

    cLeft = (Screen.Width - A_Form.Width) / 2
    cTop = (Screen.Height - A_Form.Height) / 2

    A_Form.Move cLeft, cTop

End Sub
Sub Delay (amount As Single)

    t! = Timer

    While t! + amount > Timer
    Wend

End Sub
Sub detect ()
channel$ = Left$(a$, 4)
a$ = Right$(a$, reduce)
reduce = reduce - 4
temp$ = Left$(channel$, 2)
ch$ = Right$(temp$, 1)
level$ = Right$(channel$, 2)
End Sub
Function ReadCommPort (ReadAmount As Integer) As String

    Dim ApiErr As Integer
    Dim EventMask As Integer
```

```
Dim Found As Integer

If ReadAmount < 1 Then
    ReadCommPort = ""
    Exit Function
End If

EventMask = CommEventMask
ApiErr = GetCommEventMask(CommHandle, EventMask)

If ApiErr And EV_RXCHAR Then
    Buffer$ = Space$(ReadAmount)
    ApiErr = ReadComm(CommHandle, Buffer$, Len(Buffer$))

    If ApiErr < 0 Then
        'UpdateCaption " ReadCOMM API FAILED! (ERR " + Str$(ApiErr) + ")", 3
        Buffer$ = ""
    Else
        Buffer$ = Left$(Buffer$, ApiErr)

        ' Expand CR to CR/LF for "Text" box display

        Found = 1
        Do
            Found = InStr(Found, Buffer$, Chr$(13))
            If Found Then
                Buffer$ = Left$(Buffer$, Found) + Chr$(10) + Right$(Buffer$, Len(Buffer$) -
Found)
                Found = Found + 1
            End If
        Loop While Found
    End If
```

```
End If

If (ApiErr And EV_RXFLAG) And (CommEventMask And EV_RXFLAG) Then
End If

If (ApiErr And EV_TXEMPTY) And (CommEventMask And EV_XFLAG) Then
End If

If (ApiErr And EV_CTS) And (CommEventMask And EV_CTS) Then
End If

If (ApiErr And EV_DSR) And (CommEventMask And EV_DSR) Then
End If

If (ApiErr And EV_RLSD) And (CommEventMask And EV_RLSD) Then
End If

If (ApiErr And EV_BREAK) And (CommEventMask And EV_BREAK) Then
End If

If (ApiErr And EV_ERR) And (CommEventMask And EV_ERR) Then
End If

If (ApiErr And EV_PERR) And (CommEventMask And EV_PERR) Then
End If

If (ApiErr And EV_RING) And (CommEventMask And EV_RING) Then
    'UpdateCaption " Receive Window: RING! ", 0
    Beep
End If

    ReadCommPort = Buffer$
```

```
End Function
Sub WriteCommPort (Send$)

    ApiErr% = WriteComm(CommHandle, Send$, Len(Send$))

    If ApiErr% < 0 Then
      ' UpdateCaption " WriteComm API Failed! (ERR " + Str$(ApiErr%) + ")", 2
    End If

End Sub
```

FRMRECEI.FRM

```
Dim graphdata(7) As Double
Sub Clear_text ()
    For i = 0 To 15
     Text1(i).Text = " "
    Next i

End Sub
Sub detect (b As String, ByVal count As Integer, channel As String, dBmV As String)
    block$ = Left$(b$, 4)
    channel$ = Left$(block$, 2)
    dBmV$ = Right$(block$, 2)
    b$ = Right$(b$, count)
End Sub
Function hex2dec (ByVal a$) As Double
Dim d As Double
d = Val("&H" + a$)
d = d * .019608
hex2dec = d
End Function
Sub Initialize ()

    IndexTrans = 0
    IndexReceive = 0

    CommHandle = -1
    CommDeviceNum = -1

    ' Default Port Settings
    CommPortName = "COM1:"
    CommState.BaudRate = 9600
```

```
    CommState.ByteSize = Chr$(8)
    CommState.Parity = Chr$(0)
    CommState.StopBits = Chr$(0)

    ' Default Line Settings

    CommRBBuffer = 2048
    CommTBBuffer = 2048
    CommState.RlsTimeOut = 0
    CommState.CtsTimeOut = 0
    CommState.DsrTimeOut = 0
    CommEventMask = &H3FF
    CommReadInterval = 500

    ' Post-Poned settings

    PostRBBuffer = CommRBBuffer
    PostTBBuffer = CommTBBuffer
    PostEventMask = CommEventMask
    PostState = CommState
    PostReadInterval = CommReadInterval
    PostPortName = CommPortName
End Sub
Sub Updatelabels ()
    'Update graph type description
    Select Case graph1.GraphType
    Case 1
      lblType.Caption = "1 - 2D Pie"
    Case 2
      lblType.Caption = "2 - 3D Pie"
    Case 3
      lblType.Caption = "3 - 2D Bar"
```

```
Case 4
    lblType.Caption = "4 - 3D Bar"
Case 6
    lblType.Caption = "6 - Line"
Case 7
    lblType.Caption = "7 - Log/Lin"
Case 8
    lblType.Caption = "8 - Area"
Case 9
    lblType.Caption = "9 - Scatter"
Case 10
    lblType.Caption = "10 - Polar"
End Select

'Update graph style description
Select Case graph1.GraphType
Case 1, 2
    Select Case graph1.GraphStyle
    Case 0
        lblStyle.Caption = "0 - Default"
    Case 1
        lblStyle.Caption = "1 - No label lines"
    Case 2
        lblStyle.Caption = "2 - Colored labels"
    Case 3
        lblStyle.Caption = "3 - Colored labels, no lines"
    Case 4
        lblStyle.Caption = "4 - % labels"
    Case 5
        lblStyle.Caption = "5 - % labels, no lines"
    Case 6
        lblStyle.Caption = "6 - % colored labels"
```

```
        Case 7
            lblStyle.Caption = "7 - % colored labels, no lines"
        End Select
    Case 3, 4
        Select Case graph1.GraphStyle
        Case 0
            lblStyle.Caption = "0 - Default"
        Case 1
            lblStyle.Caption = "1 - Horizontal"
        Case 2
            lblStyle.Caption = "2 - Stacked"
        Case 3
            lblStyle.Caption = "3 - Horizontal stacked"
        Case 4
            lblStyle.Caption = "4 - Stacked %"
        Case 5
            lblStyle.Caption = "5 - Horizontal stacked %"
        Case 6
            lblStyle.Caption = "6 - Z-clustered"
        Case 7
            lblStyle.Caption = "7 - Horizontal Z-clustered"
        End Select
    Case 6, 7, 10
        Select Case graph1.GraphStyle
        Case 0
            lblStyle.Caption = "0 - Default"
        Case 1
            lblStyle.Caption = "1 - Symbols"
        Case 2
            lblStyle.Caption = "2 - Sticks"
        Case 3
            lblStyle.Caption = "3 - Sticks and symbols"
```

```
        Case 4
            lblStyle.Caption = "4 - Lines"
        Case 5
            lblStyle.Caption = "5 - Lines and symbols"
        Case 6
            lblStyle.Caption = "6 - Lines and sticks"
        Case 7
            lblStyle.Caption = "7 - Lines, sticks, and symbols"
        End Select
    Case 8
        Select Case graph1.GraphStyle
        Case 0
            lblStyle.Caption = "0 - Default"
        Case 1
            lblStyle.Caption = "1 - Absolute"
        Case 2
            lblStyle.Caption = "2 - Percentage"
        End Select
    Case 9
        lblStyle.Caption = "0 - Default"
    End Select

    'Force graph to redraw
    'graph1.DrawMode = 2

End Sub
Sub btnclose_Click ()
    Unload frmrecei

End Sub
Sub cmdCopy_Click ()
    'Copy graph to Clipboard
```

```
    graph1.DrawMode = 4
    MsgBox "Graph is now copied to Clipboard"

End Sub
Sub cmdPrint_Click ()
    'Send graph to printer
    graph1.DrawMode = 5

End Sub
Sub cmdStyle_Click ()
    'Increment style based on type
    Select Case graph1.GraphType
    Case 1 To 4, 6, 7, 10
        graph1.GraphStyle = (graph1.GraphStyle + 1) Mod 8
    Case 8
        graph1.GraphStyle = (graph1.GraphStyle + 1) Mod 3
    Case 9
        graph1.GraphStyle = 0
    End Select

    'Update type and style descriptions
     Updatelabels

End Sub
Sub cmdType_Click ()
    'Increment type of graph, skipping Gantt and HLC
    Do
        graph1.GraphType = (graph1.GraphType Mod 11) + 1
    Loop While graph1.GraphType = 5 Or graph1.GraphType = 11

    'Set default graph style
    graph1.GraphStyle = 0
```

```
    'Update type and style descriptions
    Updatelabels

End Sub
Sub Form_Load ()
   If WindowState < 0 Then Exit Sub
   Move (Screen.Width - Width) / 2, (Screen.Height - Height) / 2
   Initialize
   Clear_text
End Sub
Sub Receive_text_KeyPress (Keyascii As Integer)
 Keyascii = 0
End Sub
Sub Receive_Timer_Timer ()
 Dim stab As String
Dim scrlf As String
Dim stestdata As String
Dim V As Double
stab = Chr$(9)
scrlf = Chr$(13) + Chr$(10)
a$ = ReadCommPort(1024)
bit = 36
Index = 0

If Len(a$) > 0 Then
    Receive_Text.SelStart = 0
    Receive_Text.SelLength = Len(Receive_Text.Text) + 1
    Receive_Text.SelText = a$
End If
Do
    bit = bit - 4
```

```
detect a$, bit, ch$, dat$
If ch$ = "00" Then
   Index = 0
   Text1(Index).Text = ch$
   Index = Index + 1
   Text1(Index).Text = dat$
   V = hex2dec(dat$)
   graphdata(0) = V
   Text2(Index).Text = V
   Index = Index + 1
   ElseIf ch$ = "01" Then Index = 2
      Text1(Index).Text = ch$
      Index = Index + 1
      Text1(Index).Text = dat$
      V = hex2dec(dat$)
      graphdata(1) = V
      Text2(Index).Text = V
      Index = Index + 1
      ElseIf ch$ = "02" Then
         Index = 4
         Text1(Index).Text = ch$
         Index = Index + 1
         Text1(Index).Text = dat$
         V = hex2dec(dat$)
         graphdata(2) = V
         Text2(Index).Text = V
         Index = Index + 1
         ElseIf ch$ = "03" Then
            Index = 6
            Text1(Index).Text = ch$
            Index = Index + 1
            Text1(Index).Text = dat$
```

```
V = hex2dec(dat$)
graphdata(3) = V
Text2(Index).Text = V
Index = Index + 1
ElseIf ch$ = "04" Then
    Index = 8
    Text1(Index).Text = ch$
    Index = Index + 1
    Text1(Index).Text = dat$
    V = hex2dec(dat$)
    graphdata(4) = V
    Text2(Index).Text = V
    Index = Index + 1
    ElseIf ch$ = "05" Then
        Index = 10
        Text1(Index).Text = ch$
        Index = Index + 1
        Text1(Index).Text = dat$
        V = hex2dec(dat$)
        graphdata(5) = V
        Text2(Index).Text = V
        Index = Index + 1
        ElseIf ch$ = "06" Then
            Index = 12
            Text1(Index).Text = ch$
            Index = Index + 1
            Text1(Index).Text = dat$
            V = hex2dec(dat$)
            graphdata(6) = V
            Text2(Index).Text = V
            Index = Index + 1
            ElseIf ch$ = "07" Then
```

```
                Index = 14
                Text1(Index).Text = ch$
                Index = Index + 1
                Text1(Index).Text = dat$
                V = hex2dec(dat$)
                graphdata(7) = V
                Text2(Index).Text = V
                Index = Index + 1

End If
    Index = 0
    X = X + 1
Loop Until X = 8
stestdata = Str$(graphdata(0)) & stab & Str$(graphdata(1)) & stab & Str$(graphdata(2)) & stab
& Str$(graphdata(3)) & stab & Str$(graphdata(4)) & stab & Str$(graphdata(5)) & stab &
Str$(graphdata(6)) & stab & Str$(graphdata(7)) & scrlf
graph1.QuickData = stestdata
graph1.DrawMode = 2
End Sub
```

FRMSTATUS.FRM

```
Sub btnconnect_Click ()
    image1.Visible = False
    image2.Visible = True

  CommTBBuffer = PostTBBuffer
  CommRBBuffer = PostRBBuffer
  CommPortName = PostPortName
  CommHandle = OpenComm(CommPortName, CommRBBuffer, CommTBBuffer)

  If CommHandle = -2 Then
     result% = MsgBox("Port already Open!" + Chr$(13) + "Do you want to use it anyway ?",
4 + 16 + 256, "Communication Sampler II")

     If result% = 6 Then
          ApiErr% = CloseComm(0)
          CommHandle = OpenComm(CommPortName, CommRBBuffer, CommTBBuffer)
     End If

  End If

  If CommHandle < 0 Then
     result% = MsgBox("OpenComm() API Failed! (ERR " + Str$(CommHandle) + ")", .5)
  Else

    frmrecei.Show
    CommEventMask = PostEventMask
    CommDeviceNum = Val(Mid$(CommPortName, 4, 1))
    ApiErr% = SetCommEventMask(CommHandle, CommEventMask)
    PostState = CommState
    'CommState.Id = Chr$(CommHandle)
```

```
    ApiErr% = SetCommState(CommState)


    'DisplayQBOpen CommState, CommPortName, CommRBBuffer, CommTBBuffer,
CommReadInterval

        CommReadInterval = PostReadInterval
        frmrecei.Receive_Timer.Interval = CommReadInterval

End If


End Sub
Sub btndisconnect_Click ()
    frmrecei.Receive_Timer.Interval = 0
    ApiErr% = CloseComm(CommHandle)
    CommHandle = -1
    CommDeviceNum = -1
    image1.Visible = True
    image2.Visible = False
    Unload frmrecei
    Unload frmstatus


End Sub
Sub Form_Load ()


   image1.Visible = True
   image2.Visible = False
End Sub
```

```
detect a$, bit, ch$, dat$
If ch$ = "00" Then
   Index = 0
   Text1(Index).Text = ch$
   Index = Index + 1
   Text1(Index).Text = dat$
   V = hex2dec(dat$)
   graphdata(0) = V
   Text2(Index).Text = V
   Index = Index + 1
   ElseIf ch$ = "01" Then Index = 2
      Text1(Index).Text = ch$
      Index = Index + 1
      Text1(Index).Text = dat$
      V = hex2dec(dat$)
      graphdata(1) = V
      Text2(Index).Text = V
      Index = Index + 1
      ElseIf ch$ = "02" Then
         Index = 4
         Text1(Index).Text = ch$
         Index = Index + 1
         Text1(Index).Text = dat$
         V = hex2dec(dat$)
         graphdata(2) = V
         Text2(Index).Text = V
         Index = Index + 1
         ElseIf ch$ = "03" Then
            Index = 6
            Text1(Index).Text = ch$
            Index = Index + 1
            Text1(Index).Text = dat$
```

```
    PostEventMask = CommEventMask
    PostState = Commstate
    PostReadInterval = CommReadInterval
    PostPortName = CommPortName
End Sub
Sub btnAbout_Click ()
    frmAbout.Show

End Sub
Sub btnreceive_Click ()
   frmrecei.Show
End Sub
Sub btnsetting_Click ()
    frmmdcmst.Show

End Sub
Sub btnstatus_Click ()
    frmstatus.Show
End Sub
Sub Form_Load ()
  Initialize
End Sub
```

ภาคผนวก ข.

DATA SHEET

WINAPI.TXT: Windows API Declarations and Constants for VB

Article ID: Q73694

The information in this article applies to:

- Standard and Professional Editions of Microsoft Visual Basic for Windows, versions 2.0 and 3.0
- Microsoft Visual Basic programming system for Windows, version 1.0

SUMMARY

The file WINAPI.TXT supplies declarations for Microsoft Visual Basic programmers who want to call Windows API routines.

To obtain WINAPI.TXT, download WINAPI.EXE, a self-extracting file, from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for WINAPI.EXE
  Display results and download

- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download WINAPI.EXE

- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get WINAPI.EXE

To use WINAPI.TXT, you need a reference for Windows API calls, such as the documentation provided with the Microsoft Windows Software Development Kit (SDK). If you don't have a reference manual for Windows API calls, you can obtain the Visual Basic add-on kit number 1-55615-413-5, "Microsoft Windows Programmer's Reference" and Online Resource (which includes WINAPI.TXT on disk), available from Microsoft for a fee.

MORE INFORMATION

WINAPI.TXT is an ASCII text file containing the functions and constants in the Microsoft Windows 3.0 API, declared in the format used by Microsoft Visual Basic.

To use WINAPI.TXT, you must have the book "Microsoft Windows Programmer's Reference" for Windows version 3.0 (published by Microsoft Press, 1990), or you must have the reference manuals provided with the Microsoft Windows SDK.

WINAPI.TXT includes the following:

- External procedure declarations for all the Microsoft Windows API functions that can be called from Visual Basic.
- Global constant declarations for all the constants used by the Microsoft Windows API.
- Type declarations for the user-defined types (structures) used by the Microsoft Windows API.

WINAPI.TXT is too large to be loaded directly into a Visual Basic module. Attempting to load it directly into Visual Basic will cause an "Out of Memory" error message.

WINAPI.TXT is also too large for the Notepad editor supplied with Microsoft Windows, but it can be loaded by Microsoft Write. To use WINAPI.TXT, load it into an editor (such as Microsoft Write) that can handle large files. Copy the declarations you want and paste them into the global module in your Visual Basic application.

NOTE: Some of the Windows API declarations are very long. Some editors will wrap these onto a second line, and will copy them as multiple lines rather than a single line. Declarations in Visual Basic cannot span lines, so if you paste these as multiple lines, Visual Basic will report an error. If an error occurs, you can either adjust the margins in the editor before copying or remove the line break after pasting.

The global module is the recommended place for the declarations that you copy from the WINAPI.TXT file; however, you can place the external procedure declarations in the Declarations section of any form or module. You can also place the constant declarations anywhere in any module or form code if you remove the Global keyword. Type declarations must be placed in the global module.

Once you have pasted the declaration for a Windows API routine (as well as any associated constant and type declarations) into your application, you can call that routine as you would call any Visual Basic procedure.

For more information about declaring and calling external procedures, see Chapter 23, "Extending Visual Basic," in "Microsoft Visual Basic: Programmer's Guide."

WARNING: Visual Basic cannot verify the data you pass to Microsoft Windows API routines. Calling a Microsoft Windows API routine with an invalid argument can result in unpredictable behavior: your application, Visual Basic, or Windows may crash or hang. When experimenting with Windows API routines, save your work often.

Additional reference words: 1.00 2.00 3.00 softlib
KBCategory: kbprg kbfile
KBSubcategory: APrgOther

Using the ADC0808/ ADC0809 8-Bit μP Compatible A/D Converters with 8-Channel Analog Multiplexer

National Semiconductor
Application Note 247
Larry Wakeman
September 1980

INTRODUCTION

The ADC0808/ADC0809 Data Acquisition Devices (DAD) implement on a single chip most the elements of the standard data acquisition system. They contain an 8-bit A/D converter, 8-channel multiplexer with an address input latch, and associated control logic. These devices provide most of the logic to interface to a variety of microprocessors with the addition of a minimum number of parts.

These circuits are implemented using a standard metal-gate CMOS process. This process is particularly suitable to applications where both analog and digital functions must be implemented on the same chip.

These two converters, the ADC0808 and ADC0809, are functionally identical except that the ADC0808 has a total unadjusted error of ±½ LSB and the ADC0809 has an unadjusted error of ±1 LSB. They are also related to their big brothers, the ADC0816 and ADC0817 expandable 16 channel converters. All four converters will typically do a conversion in ~100 μs when using a 640 kHz clock, but can convert a single input in as little as ~50 μs.

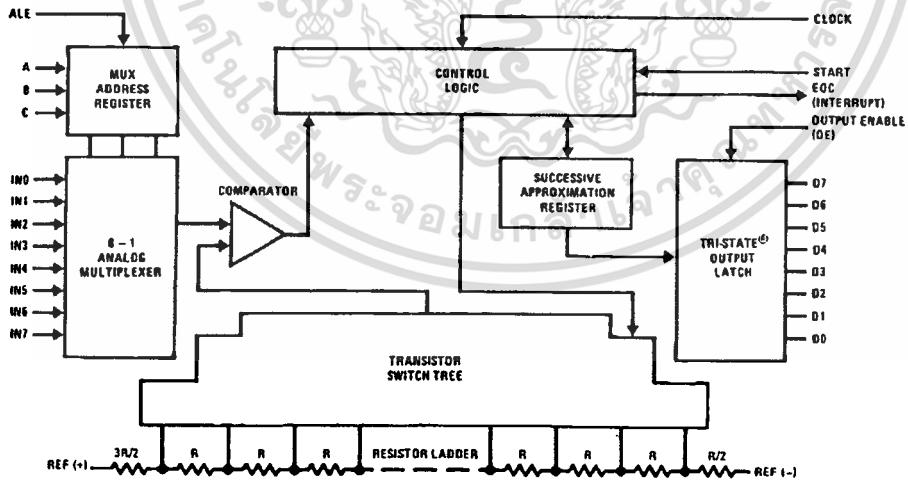
1.0 FUNCTIONAL DESCRIPTION

The ADC0808/ADC0809, shown in *Figure 1*, can be functionally divided into 2 basic subcircuits. These two subcircuits are an analog multiplexer and an A/D converter. The multiplexer uses 8 standard CMOS analog switches to provide for up to 8 analog inputs. The switches are selectively turned on, depending on the data latched into a 3-bit multiplexer address register.

The second function block, the successive approximation A/D converter, transforms the analog output of the multiplexer to an 8-bit digital word. The output of the multiplexer goes to one of two comparator inputs. The other input is derived from a 256R resistor ladder, which is tapped by a MOSFET transistor switch tree. The converter control logic controls the switch tree, funneling a particular tap voltage to the comparator. Based on the result of this comparison, the control logic and the successive approximation register (SAR) will decide whether the next tap to be selected should be higher or lower than the present tap on the resistor ladder. This algorithm is executed 8 times per conversion, once every 8 clock periods, yielding a total conversion time of 64 clock periods.

When the conversion cycle is complete the resulting data is loaded into the TRI-STATE® output latch. The data in the output latch can then be read by the host system any time before the end of the next conversion. The TRI-STATE capability of the latch allows easy interface to bus oriented systems.

The operation of these converters by a microprocessor or some control logic is very simple. The controlling device first selects the desired input channel. To do this, a 3-bit channel address is placed on the A, B, C input pins; and the ALE input is pulsed positively, clocking the address into the multiplexer address register. To begin the conversion, the START pin is pulsed. On the rising edge of this pulse the internal registers are cleared and on the falling edge the start conversion is initiated.



TL/H/5623–1

FIGURE 1. ADC0808/ADC0809 Functional Block Diagram

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

RRD-B30M115/Printed in U. S. A.

As mentioned earlier, there are 8 clock periods per approximation. Even though there is no conversion in progress the ADC0808/ADC0809 is still internally cycling through these 8 clock periods. A start pulse can occur any time during this cycle but the conversion will not actually begin until the converter internally cycles to the beginning of the next 8 clock period sequence. As long as the start pin is held high no conversion begins, but when the start pin is taken low the conversion will start within 8 clock periods.

The EOC output is triggered on the rising edge of the start pulse. It, too, is controlled by the 8 clock period cycle, so it will go low within 8 clock periods of the rising edge of the start pulse. One can see that it is entirely possible for EOC to go low before the conversion starts internally, but this is not important, since the positive transition of EOC, which occurs at the end of a conversion, is what the control logic is looking for.

Once EOC does go high this signals the interface logic that the data resulting from the conversion is ready to be read. The output enable (OE) is then raised high. This enables the TRI-STATE outputs, allowing the data to be read. *Figure 2* shows the timing diagram.

2.0 ANALOG INPUTS

2.1 Ratiometric Inputs

The arrangement of the REF(+) and REF(−) inputs is intended to enable easy design of ratiometric converter systems. The REF inputs are located at either end of the 256R resistor ladder and by proper choice of the input voltages several applications can be easily implemented.

Figure 3 shows a typical input connection for ratiometric transducers. A ratiometric transducer is a conversion device whose output is proportional to some arbitrary full-scale value. In other words, the transducer's absolute output value is of no particular concern but the ratio of the output to the full-scale is of great importance. For example, the potentiometric displacement transducers of *Figure 3* have this feature. When the wiper is at midscale, the output voltage is $V_O = V_F \times$ (Wiper Displacement) $= V_F \times 0.5$. This enables the use of much less accurate and less expensive references. The important consideration for this reference is noise. The reference must be "glitch free" because a voltage spike during a conversion cycle could cause conversion inaccuracies.

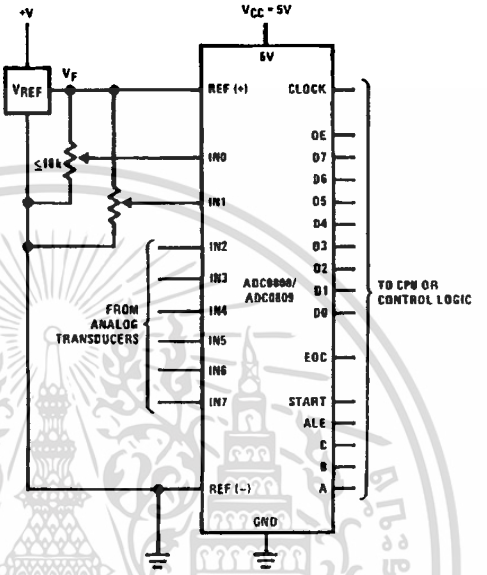
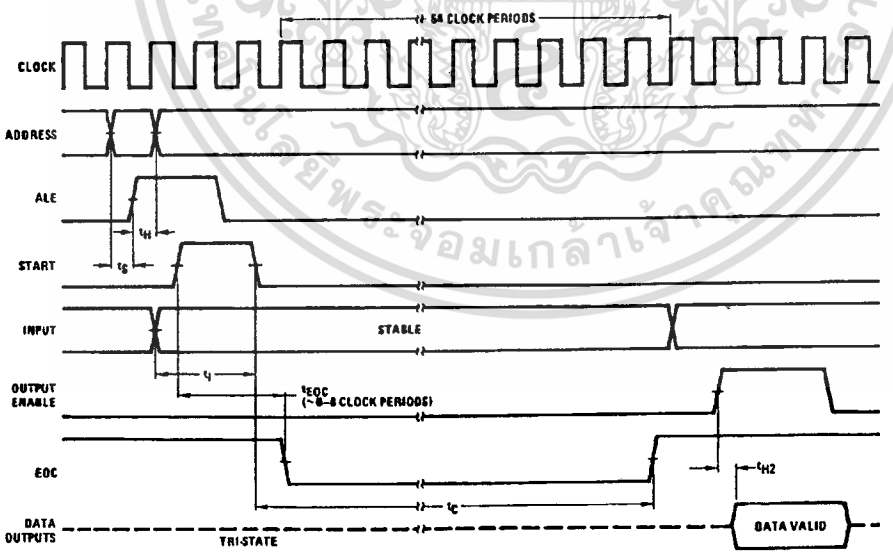


FIGURE 3. Ratiometric Converter with Separate Reference



TL/H/5623–2

FIGURE 2. ADC0808/ADC0809 Timing Diagram

Since highly accurate references aren't required it is possible to use the system power supply as a reference, as shown in *Figure 4*. If the power supply is to be used in this manner supply noise must be kept to a minimum to preserve conversion accuracy. If possible the supply should be well bypassed and separate reference and supply PC board traces, originating as close as possible to the power supply or regulator, should be used. This is illustrated in *Figure 4*. External accessibility of both ends of the resistor ladder enables several variations on these basic connections, and are shown in *Figures 5* and *6*. The magnitude of the reference voltage, $V_{REF} = REF(+) - REF(-)$, can be varied from about $-0.5V$ to V_{CC}, but the center voltage must be maintained within $\pm 0.1V$ of $V_{CC}/2$. This constraint is due to the design of the transistor switch tree, which could malfunction if the offset from center scale becomes excessive. Variation of the reference voltage can sometimes eliminate the need for external gain blocks to scale the input voltage to a full-scale range of 5V.

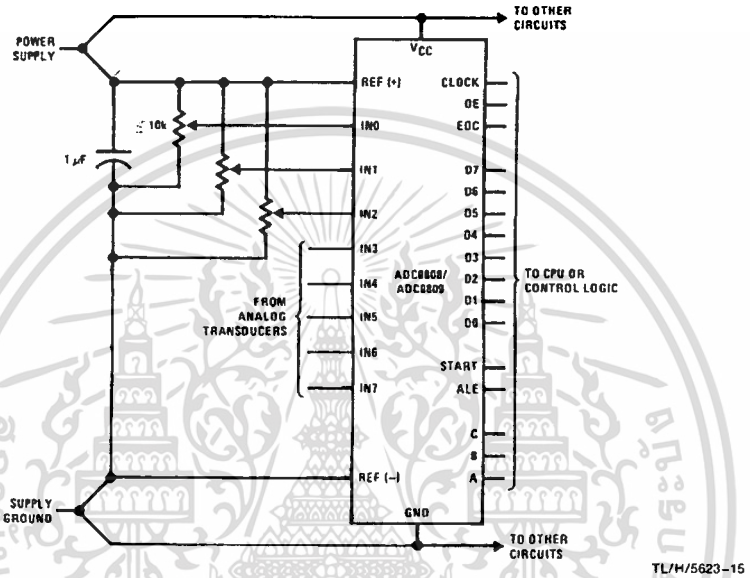


FIGURE 4. Ratiometric Converter with Power Supply Reference

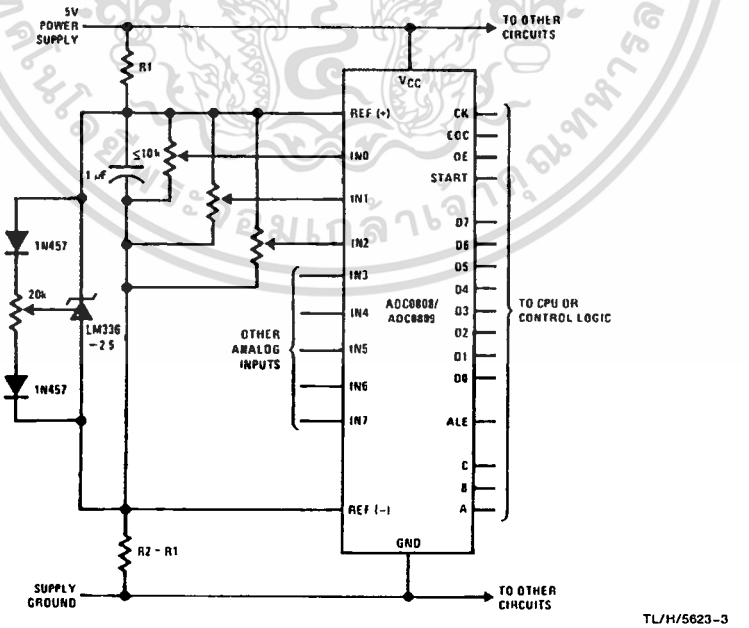


FIGURE 5. Mid-Supply Centered Reference Using LM336 2.5V Reference

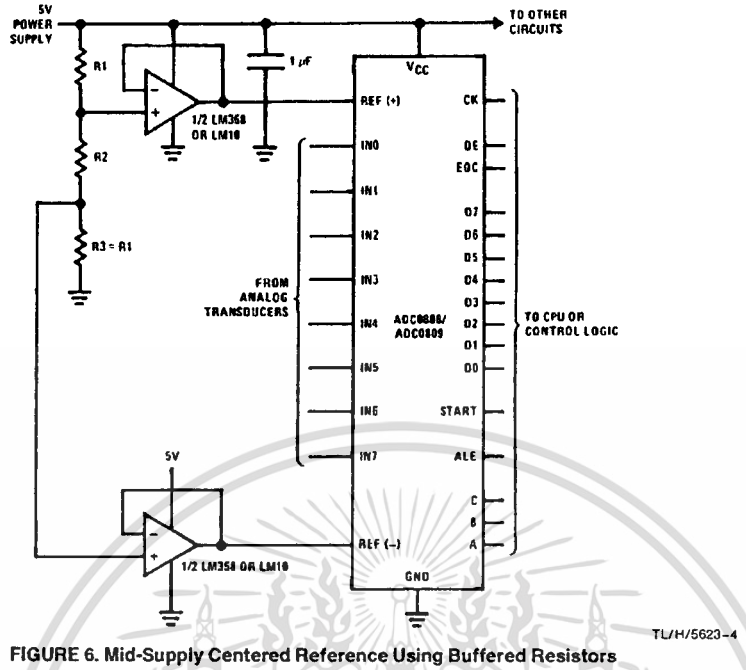


FIGURE 6. Mid-Supply Centered Reference Using Buffered Resistors

Figure 5 shows a center referencing technique, using two equal resistors to symmetrically offset an LM336 2.5V reference, from both supplies. The offset from either supply is:

$$V_{OFF} = \frac{V_{CC} - V_{REF}}{2} = 1.25V$$

These resistors should be chosen so that they limit current through the LM336 to a reasonable value, say 5 mA. The total resistor current is:

$$I_R = I_{REF} + I_{LADDER} + I_{TRAN}$$

where I_{LADDER} is the 256R ladder current, I_{TRAN} is the current through all the transducers, and I_{REF} is the current through the reference. R1 and R2 should be well matched and track each other over temperature.

For odd values of reference voltage, the reference could be replaced by a resistor, but due to loading and temperature problems, these resistors should be buffered to the REF(+) and REF(−) inputs, *Figure 6*. The power supply must be well bypassed as supply glitches would otherwise be passed to the reference inputs. The reference voltage magnitude is:

$$V_{REF} = V_{DD}\left(\frac{R2}{2R1 + R2}\right) \text{ For } R3 = R1$$

There are several op amps that can be used for buffering this ladder. Without adding another supply, an LM358 could be used if the REF(+) input is not to be set above 3.5V. The LM10 can swing closer to the positive supply and can be used if a higher $V_{REF(+)}$ voltage is needed.

As the REF(+) to REF(−) voltage decreases the incremental voltage step size decreases. At 5V one LSB represents ~20 mV, but at 1V, one LSB represents ~4 mV.

As the reference voltage decreases, system noise will become more significant so greater precaution should be enforced at lower voltages to compensate for system noise; i.e., adequate supply and reference bypassing, and physical as well as electrical isolation of the inputs.

2.2 Absolute Analog Inputs

The ADC0808/ADC0809 may have been designed to easily utilize ratiometric transducers, but this does not preclude the use of non-ratiometric inputs. A second type of input is the absolute input. This is one which is independent of the reference. This implies that its *absolute* numerical voltage value is very critical, and to accurately measure this voltage the accuracy of the reference voltage becomes equally critical. The previous designs can be modified to accommodate absolute input signals by using a more accurate reference. In *Figure 4* the power supply reference could be replaced by LM336-5.0 reference. R1 and R2 of *Figure 6*, and R1 and R3 of *Figure 7* may have to be made more accurately equal.

In some small systems it is possible to use the precision reference as the power supply as shown in *Figure 7*. An unregulated supply voltage > 5V is required, but the LM336-5.0 functions as both a regulator and reference. The dropping resistor R must be chosen so that, for the whole range of supply currents needed by the system, the LM336-5.0 will stay in regulation. As in *Figure 4* separate supply and reference traces should be used to maintain a noiseless supply.

If the system requires more power, an op amp can be used as shown in *Figure 8* to isolate the reference and boost the supply current capabilities. Here again, a single unregulated supply is required.

2.3 Differential Inputs

Differential measurements can be obtained by playing a little software trick. This simply involves sequentially converting two channels then subtracting the two results. For example, if the difference voltage between channel 1 and 2 is required, merely convert channel 1 and read the result. Then convert channel 2, input the result, and subtract it from the first result. (See *Figure 9.*) When using this procedure, both input signals must be stable throughout both conversion times or the end result will be incorrect. One way to get around this is to use two sample/holds which are sampled at the same time.

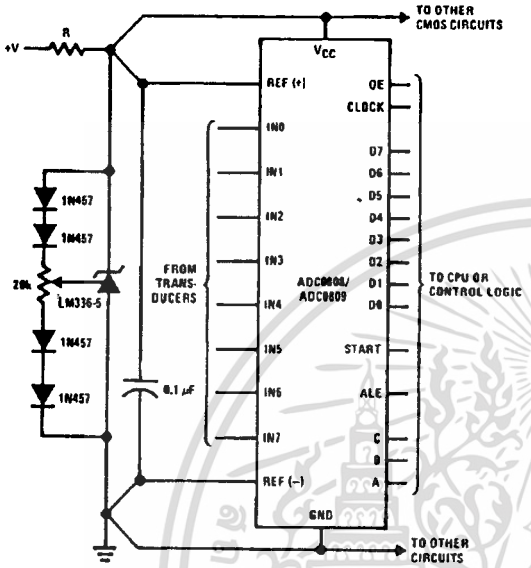


FIGURE 7. Precision Reference used as a Power Supply

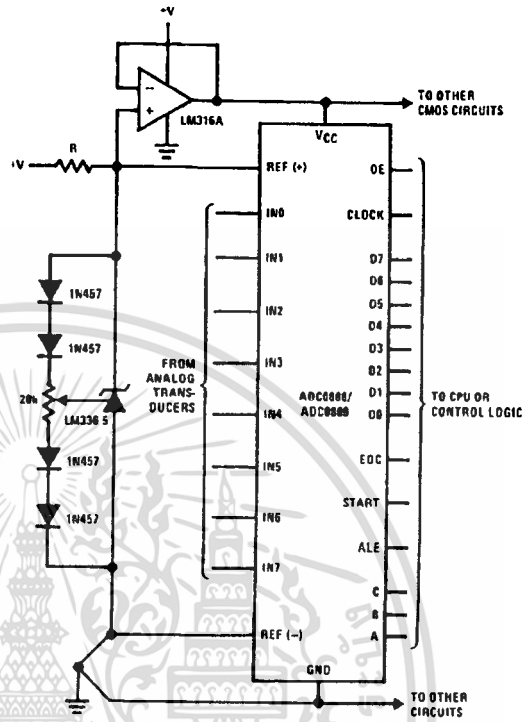
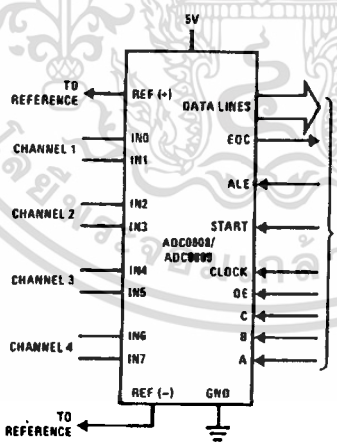


FIGURE 8. Precision Reference Buffered for Power Supply

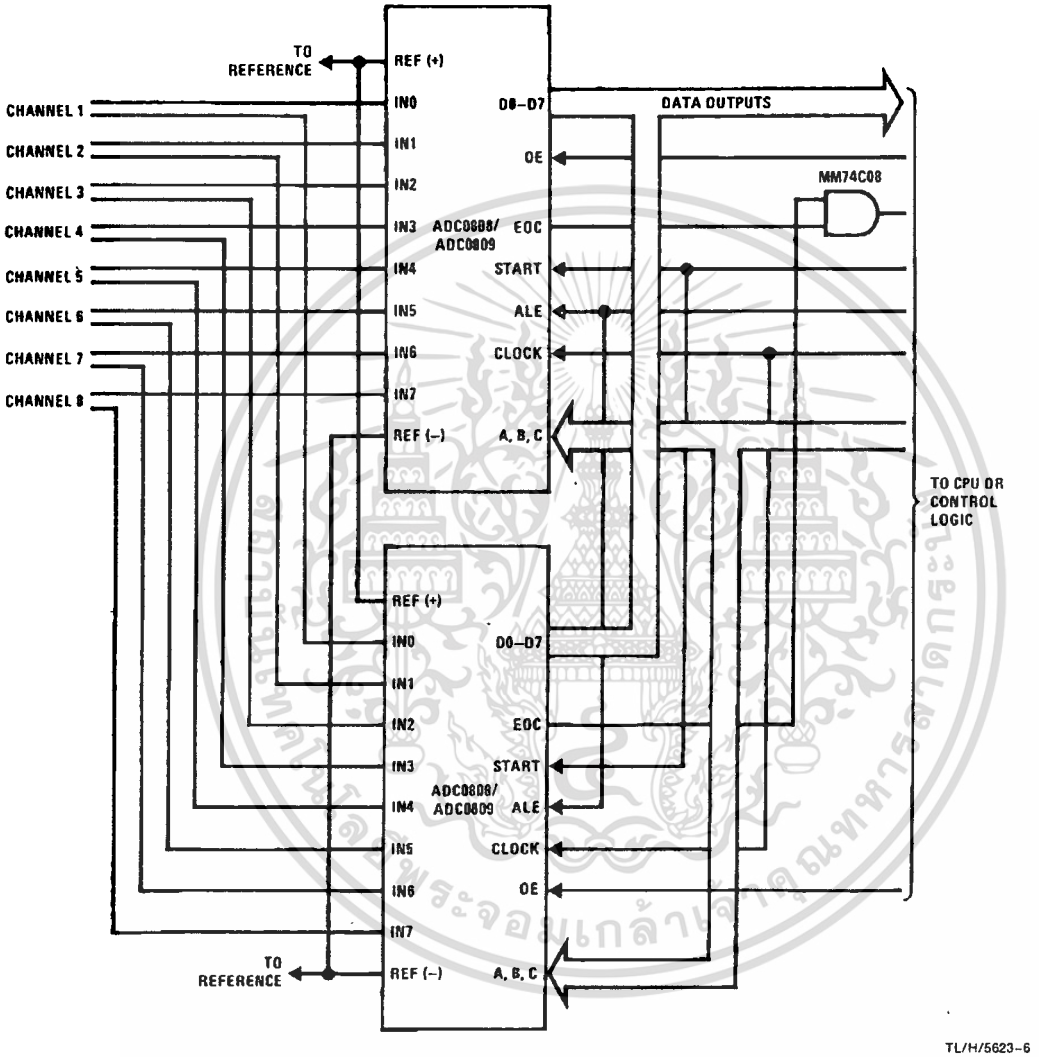


TL/H/5623–6

FIGURE 9. Software Controlled Differential Converter

A second method is to use two chips to convert a differential channel, *Figure 10*. Typically each channel 1 would be connected to opposite sides of the differential input. Both converters are started simultaneously. When both converters' EOC outputs go high the output of the AND gate will go high indicating that the data is ready to be read.

The circuit in *Figure 10* can be slightly modified to provide increased data throughput by using two converters in a parallel data acquisition scheme. *Figure 11* shows this circuit in which all the input channels are connected in pairs through LF398 monolithic sample/holds. Under normal operation a sample/hold is accessed through an MM74C42 which will pulse an MM74C221, generating a sample pulse. After a sample/hold is done sampling the signal, the appropriate channel is started. If this process is alternated between two converters the sample rate can be doubled.

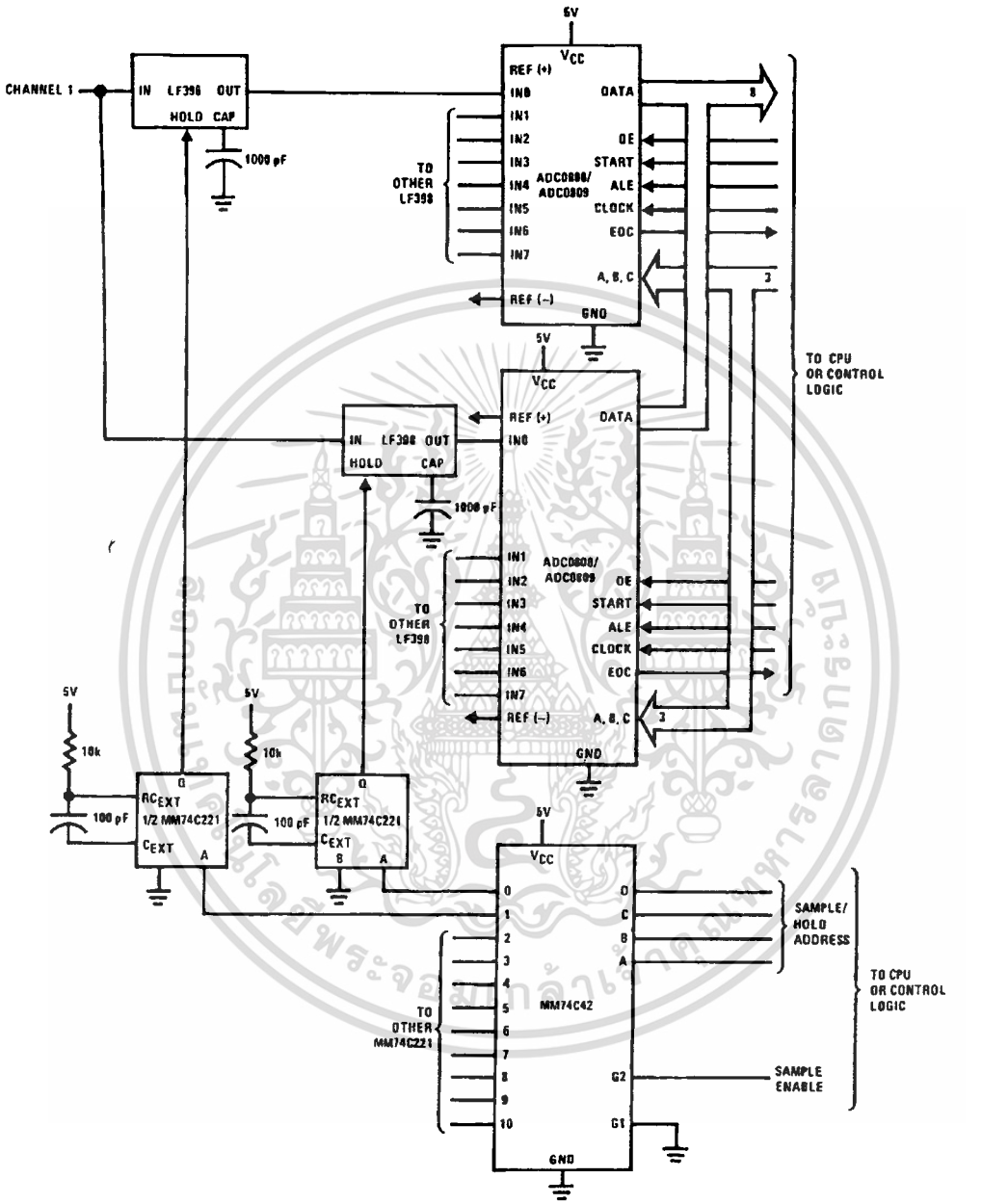


TL/H/5623–6

FIGURE 10. Dual Converter Differential Circuit

2.4 Analog Input Considerations

Analog inputs into the ADC0808/ADC0809 can handle any input signal that is maintained within the supply limits, but some careful consideration must be given to the output impedance of the transducer or buffer. Using transducers with large source impedances can cause errors due to comparator input currents.



TL/H/5623–7

FIGURE 11. Parallel Data Acquisition with Sample/Holds

To understand the nature of these currents a short discussion of comparator operation is required. *Figure 12* shows a simplified model of the comparator and multiplexer. This comparator alternately samples the input voltage and the ladder voltage. As it samples the input, C_C and C_P are charged up to the input voltage. It then samples the ladder and discharges the capacitor. The net charge difference is determined by a modified inverter chain and results in a 1 or 0 state at the output.

Eight samples are made per conversion, resulting in eight spikes of varying magnitude on the input.

If the source resistance is large, it adds to the RC time constant of the switched capacitor which will inhibit the input from settling properly, causing errors. As one might expect, the maximum source resistance allowable for accurate conversions is inversely proportional to clock frequency. This resistance should be $\leq$1 kΩ at 1.2 MHz and $\leq$2 kΩ at 640 kHz. If a potentiometer-type ratiometric transducer is used it should be $\leq$5 kΩ at 1.2 MHz and $\leq$10 kΩ at 640 kHz.

If large source impedances are unavoidable ($\geq$2 kΩ at 640 kHz), the transient errors can be reduced by placing a bypass capacitor $\geq$0.1 μF from the analog inputs to ground. This will reduce the spikes to a small average current which will cause some error as well, but this can be much less than the error otherwise incurred. The maximum voltage error for a potentiometer input with a bypass capacitor added is:

$$V_{ERR} \approx \left[\frac{R_{POT}}{5}(I_{IN})\frac{Ck}{640\ kHz}\right]V$$

where R_{POT} = total potentiometer resistance; I_{IN} = maximum input current at 640 kHz, 2 μA; and Ck = clock frequency.

For standard buffer source impedance the maximum error is:

$$V_{ERR} = \left[I_{IN}R_S\left(\frac{Ck}{640\ kHz}\right)\right]V$$

where R_S = buffer source resistance; I_{IN} = the maximum input current at 640 kHz, 2 μA; and Ck = clock frequency.

3.0 MICROPROCESSOR INTERFACING

The ADC0808/ADC0809 converters were designed to interface to most standard microprocessors with very little external logic, but there are a few general requirements which must be considered to ensure proper converter operation. Most microprocessors are designed to be TTL compatible and, due to speed and drive requirements, incorporate many TTL circuits. The data outputs of the ADC0808/ADC0809 are capable of driving one standard TTL load which is adequate for most small systems, but for larger systems extra buffering may be necessary. The EOC output is not quite as powerful as the data outputs, but normally it is not bussed like the data outputs.

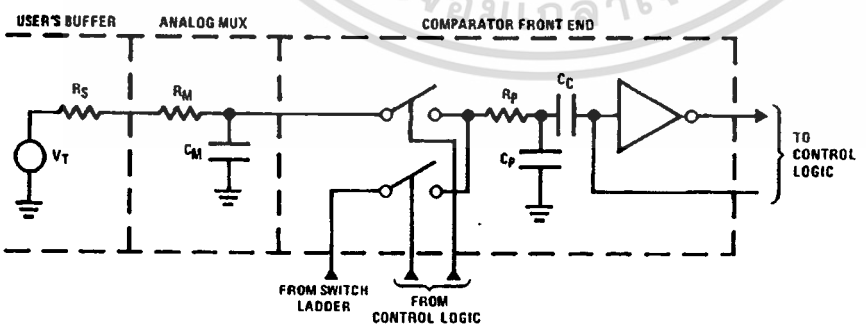
The converter inputs are standard CMOS compatible inputs. When TTL outputs are connected to any of the digital inputs a pull-up resistor should be tied from the TTL output to V_{CC}, ~ 5 kΩ. This will ensure that the TTL will pull-up above 3.5V.

Usually the converter clock will be derived from the microprocessor system clock. Some slower microprocessor clocks can be used directly, but at worst a few divider stages may be necessary to divide microprocessor clock frequencies above 1.2 MHz to a usable value.

The timing of the START and ALE pulses relative to channel selection and signal stability can be critical. The simplest approach to microprocessor interfaces usually ties START and ALE together. When these lines are strobed the address is strobed into the address register and the conversion is started. The propagation delay from ALE to comparator input of the selected input signal is about ~3.0 μs (input source resistance $<<$1 kΩ). If the start pulse is very short the comparator can sample the analog input before it is stable. When using a slow clock $\leq$500 kHz the sample period of the comparator input is long enough to allow this delay to settle out.

If the ADC0808/ADC0809 clock is >500 kHz, a delay between the START and ALE pulses is required. There are three basic methods to accomplish this. The first possibility is to design the microprocessor interface so that the START and ALE inputs are separately accessible. This is simple if some extra address decoding is available. Separate accessibility of the START and ALE pins allows the microprocessor, via software, to set the delay time between the START and ALE pulses.

If extra decoding is not available, then START and ALE could be tied together. To obtain the proper delay, the microprocessor would cause START/ALE to be strobed twice by executing the load and start instruction twice. The first time this instruction is executed, the new channel address is loaded and the conversion is started. The second execution of this instruction will reload the same channel address and restart the conversion. But since the multiplexer address register contents are unchanged the selected analog input will have already settled by the time the second instruction is issued. Actual implementations of these ideas are shown in following sections.



TL/H/5623–8

FIGURE 12. Analog Multiplexer and Comparator Input Model

A third possibility when ALE and START are tied together is to stretch the microprocessor derived ALE/START pulse by inserting a one-shot at these inputs and creating a positive pulse ≥3 μs. Since ALE loads the multiplexer register on the positive going edge of the pulse and START begins the conversion on the falling edge, the width of the pulse sets the ALE to START delay time.

Most microprocessor interfaces would be designed such that a START pulse is issued by a memory or I/O write instruction, although a memory or I/O read can be used. The ALE strobe on the other hand, requires a write by the CPU when A, B, and C are connected to the data bus, and could use a read instruction if A, B, and C are connected to the address bus, but the software could get confusing. The logic to derive the OE strobe must be connected to the microprocessor so that a memory or I/O read instruction will cause OE to be pulsed. A read is required since the ADC0808/ADC0809 data must be read.

3.1 Interfacing to the 8080

The simplest interface would contain no address decoding, which may seem unreasonable; but if the system ports are I/O mapped, up to 8 of them can be connected to the CPU with no decoding. Each of the 8 I/O address lines would serve as a simple port enable line which would be gated with read and write strobes to select a particular port. This scheme is shown in *Figure 13*. A7 is the address line used and, whenever it is zero and an I/O read or write is low, the port is accessed. This implementation shows A, B, C connected to D0, D1, D2 causing the information on the data bus to select the channel, but A, B, and C could be connected to the address bus, with a loss of only 3 ports. Both decoding schemes are tabulated in *Figure 14*. (Remember A, B, C inputs are only valid when selecting a channel to convert, and are not used to read data.)

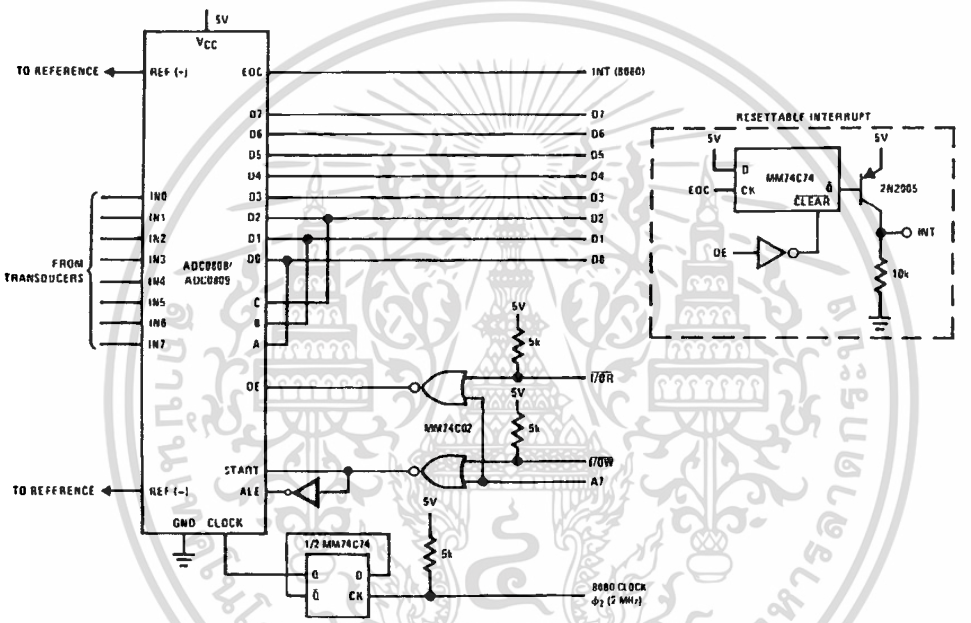


FIGURE 13. Minimum 8080/8224/8228 Interface

TL/H/5623–9

A7	A6	A5	A4	A3	A2	A1	A0	D2	D1	D0	Output Port Description
1	1	1	1	1	1	1	0	X	X	X	Spare Port
1	1	1	1	1	1	0	1	X	X	X	Spare Port
1	1	1	1	1	0	1	1	X	X	X	Spare Port
1	1	1	1	0	1	1	1	X	X	X	Spare Port
1	1	1	0	1	1	1	1	X	X	X	Spare Port
1	1	0	1	1	1	1	1	X	X	X	Spare Port
1	0	1	1	1	1	1	1	X	X	X	Spare Port
0	1	1	1	1	1	1	1	0	0	0	Channel 0 Port
0	1	1	1	1	1	1	1	0	0	1	Channel 1 Port
0	1	1	1	1	1	1	1	0	1	0	Channel 2 Port
0	1	1	1	1	1	1	1	0	1	1	Channel 3 Port
0	1	1	1	1	1	1	1	1	0	0	Channel 4 Port
0	1	1.	1	1	1	1	1	1	0	1	Channel 5 Port
0	1	1	1	1	1	1	1	1	1	0	Channel 6 Port
0	1	1	1	1	1	1	1	1	1	1	Channel 7 Port

FIGURE 14a. Write Address Decoding for 8080 Output Ports (A, B, C Connected to D0, D1, D2)

A7	A6	A5	A4	A3	A2	A1	A0	Output Port Description
0	1	1	1	1	0	0	0	Channel 0 Port
0	1	1	1	1	0	0	1	Channel 1 Port
0	1	1	1	1	0	1	0	Channel 2 Port
0	1	1	1	1	0	1	1	Channel 3 Port
0	1	1	1	1	1	0	0	Channel 4 Port
0	1	1	1	1	1	0	1	Channel 5 Port
0	1	1	1	1	1	1	0	Channel 6 Port
0	1	1	1	1	1	1	1	Channel 7 Port
1	1	1	1	0	X	X	X	Spare Port
1	1	1	0	1	X	X	X	Spare Port
1	1	0	1	1	X	X	X	Spare Port
1	0	1	1	1	X	X	X	Spare Port

X = don't care

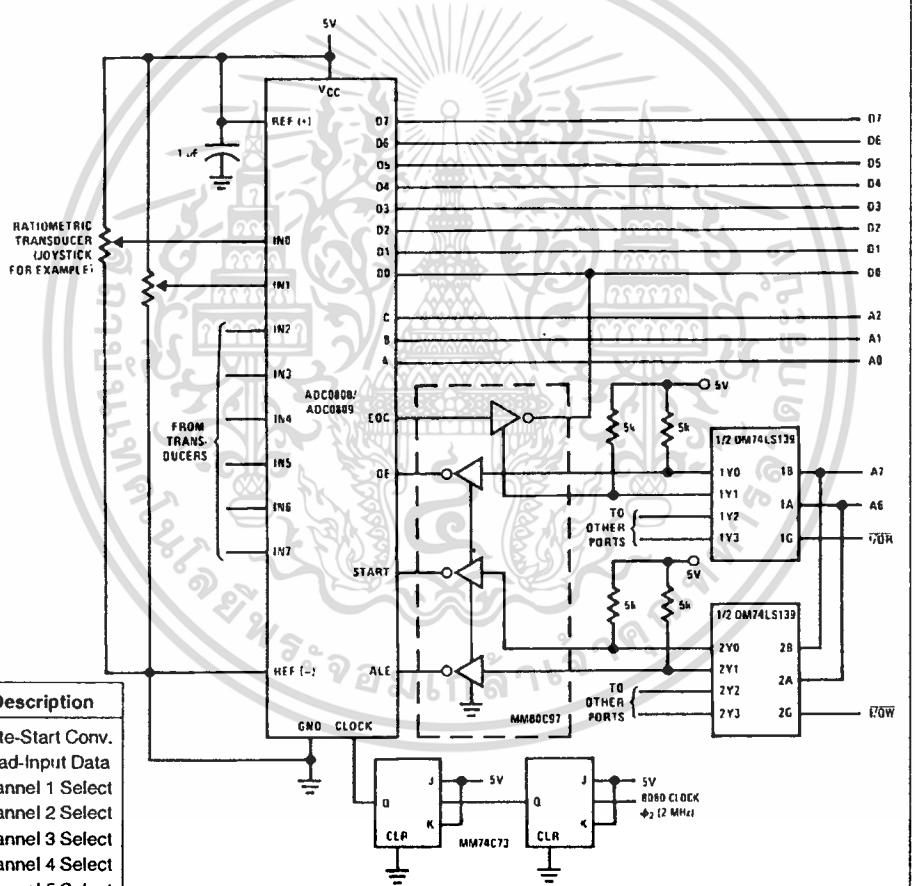
FIGURE 14b. Modified Write Address Decoding for 8080 Output Ports (A, B, C Connected to A0, A1, A2)

Two LSTTL NOR gates are used to generate the ADC0808/ADC0809 read/write strobes. When the 8080 writes to the ADC0808/ADC0809 the ALE and START inputs are strobed, loading and starting the conversion. When the CPU reads the ADC0808/ADC0809 the OE input is taken high, and the data outputs are enabled.

Figure 13 implements a simple interrupt concept where EOC is tied directly to the 8080 interrupt input. When the INS8228 is used and the INTA pin is tied high through a 1 kΩ resistor, the interrupt will cause a restart, RST, instruction to be executed, which will then cause a jump to a restart vector and execution of the interrupt routine. If a very simple multi-interrupt system is desired, a wire OR'ed configuration employing resettable latches as shown in *Figure 13's* inset can be used. In this simple design the MM74C74 is reset when the ADC0808/ADC0809 data is read. If more complicated interrupt structures are required, then an interrupt controller is usually the best solution.

The I/O port address structure for *Figure 13's* implementation is shown in *Figure 14a*. If the A, B, C inputs are tied to A0, A1, A2 inputs the port structure is as shown in *Figure 14b*. The later method makes each channel look like a separate port address, whereas if A, B, C are tied to the data bus the ADC0808/ADC0809 looks like one start conversion port address whose channel is selected by the 3-bit status word written to it on the data bus.

Figure 15 shows a slightly more complex interface, where the address is partialy decoded by a DM74LS139, dual 2-4 line decoder which creates the read and write strobes to operate the converter. This design interfaces to the processor in a polled type of interface. An MM80C97 TRI-STATE buffer is used to buffer the EOC line to the data bus, as well as provide the correct level for the START, ALE, and OE pulses. The converter clock is a divided 8080 system clock.



Address A7–A0	Description
0 0 X X X X X X	Write-Start Conv.
0 0 X X X X X X	Read-Input Data
0 1 X X X 0 0 0	Channel 1 Select
0 1 X X X 0 0 1	Channel 2 Select
0 1 X X X 0 1 0	Channel 3 Select
0 1 X X X 0 1 1	Channel 4 Select
0 1 X X X 1 0 0	Channel 5 Select
0 1 X X X 1 0 1	Channel 6 Select
0 1 X X X 1 1 0	Channel 7 Select
0 1 X X X 1 1 1	Channel 8 Select
0 1 X X X X X X	Read-Input EOC

FIGURE 15. 8080/8224/8228 Interface Using Partial Decoding

Typically, the software to use *Figure 15* would first select the desired channel by writing the channel address to the ALE port address, 01XXXCBA, where X = don't care, and CBA is the channel address. Next the conversion is started by writing to the START address, 00XXXXXX. Now the processor must wait a few instruction cycles to allow EOC to fall. Once EOC falls, its status can be checked by reading the EOC line, address 01XXXXXX. When the EOC line is detected high again (a low on DO), the data can be read by accessing the OE port, address 00XXXXXX. As in the previous example the A, B, C inputs can be tied to D0, D1, D2 rather than A0, A1, A2, so that the information on the data bus selects the channel to be converted. *Figure 15* can be connected in an interrupt mode by incorporating the interrupt flip-flop of *Figure 13*.

A few typical utility routines to operate the ADC0808/ADC0809 application in *Figure 13* are shown in *Figure 16a*. These routines assume that the resettable interrupt flip-flop is used. *Figure 16b* illustrates some typical polled I/O routines for *Figure 15*. Notice that in *Figure 16a* the OUT START1 instruction is executed twice to allow the analog input signal to settle as discussed earlier.

```
;
;
; START CONVERSION (A, B, C CONNECTED TO D0, D1, D2)
;
CHANN1                  EQU     7
START1                  EQU     7FH
DATA                    EQU     7FH
;
START:                  LDA     CHANN1      ; LOAD CHANNEL ADDRESS INTO ACE
                        OUT     START1      ; STORE IT TO ADC0808/ADC0809 AND START
                        OUT     START1      ; RESTART ADC0808/ADC0809 TO ACCOUNT FOR
;                                           ; MULTIPLEXER DELAY
                        EI                  ; ENABLE INTERRUPTS IF NOT ALREADY
                        —       —           ; PROCESS PROGRAM
;
; INTERRUPT HANDLER ROUTINE
;
INTRP:                  IN      DATA        ; READ DATA AND RESET INTERRUPT
                        —       —           ; PROCESS DATA
                        EI                  ; ENABLE INTERRUPTS IF DESIRED
                        RET                 ; RETURN TO MAIN PROGRAM
```

FIGURE 16a. Typical 8080 Resettable Interrupt I/O Routines

```
;
; START CONVERSION (A, B, C CONNECTED TO A0, A2, A3) AND POLL EOC
; (FIGURE 15)
SELECT          EQU     40H         ; SELECT CHANNEL 0
START           EQU     00H         ; START CONVERTER
EOCIN           EQU     40H         ; READ EOC
DATA            EQU     00H         ; READ DATA
START:          OUT     SELECT      ; SELECT CHANNEL
                OUT     START       ; START CONVERSION
                NOP                 ; INSERT INSTRUCTIONS TO WAIT 0-8
                NOP                 ; CLOCK PERIODS OF ADC0808/ADC0809 CLOCK
                NOP                 ; FOR EOC TO DROP (8NOPs MINIMUM)
                NOP
                NOP
;
; READ AND TEST EOC
;
STATUS:         IN      EOCIN       ; INPUT EOC BIT
                ANI     01H         ; MASK OUT OTHER BITS
                JZ      READY       ; IF INPUT BIT IS ZERO JUMP READY
                —       —           ; ELSE CONTINUE EXECUTING PROGRAM
; OR
; CONTINUOUS POLLING ROUTINE
;
STAT 2:         IN      EOCIN       ; INPUT EOC STATUS BIT
                ANI     01H         ; MASK OUT ALL BITS BUT D0
                JNZ     STAT 2      ; JUMP TO TRY AGAIN IF NOT READY
READY:          IN      DATA        ; IF READY INPUT DATA
                —       —           ; CONTINUE EXECUTING PROGRAM
```

FIGURE 16b. Typical Polled I/O Routines for ADC0808/ADC0809

The application in *Figure 17* uses a 6-bit bus comparator and a few gates to decode a read and write strobe. Viewed from the CPU this interface looks like a bidirectional data port whose address is set by the logic levels on the T_n inputs of the DM8131 comparator. When data is written to the ADC0808/ADC0809 the 3 least significant bits on the address bus define the channel to be converted. The rest of the bits are decoded to provide the START and ALE strobes. When the conversion is completed EOC sets the interrupt flip-flop, and when the data is read the interrupt is reset.

Both the decoder and the bus comparator methods of address decoding have their own advantages. Bus comparators will more completely decode addresses but are capable of only a limited number of port strobes. Decoders, on the other hand, provide less decoding but more port strobes. There is a trade off for minimum parts systems as far as which route to go, and it will depend on the CPU and type of system.

3.2 Interfacing to the 6800

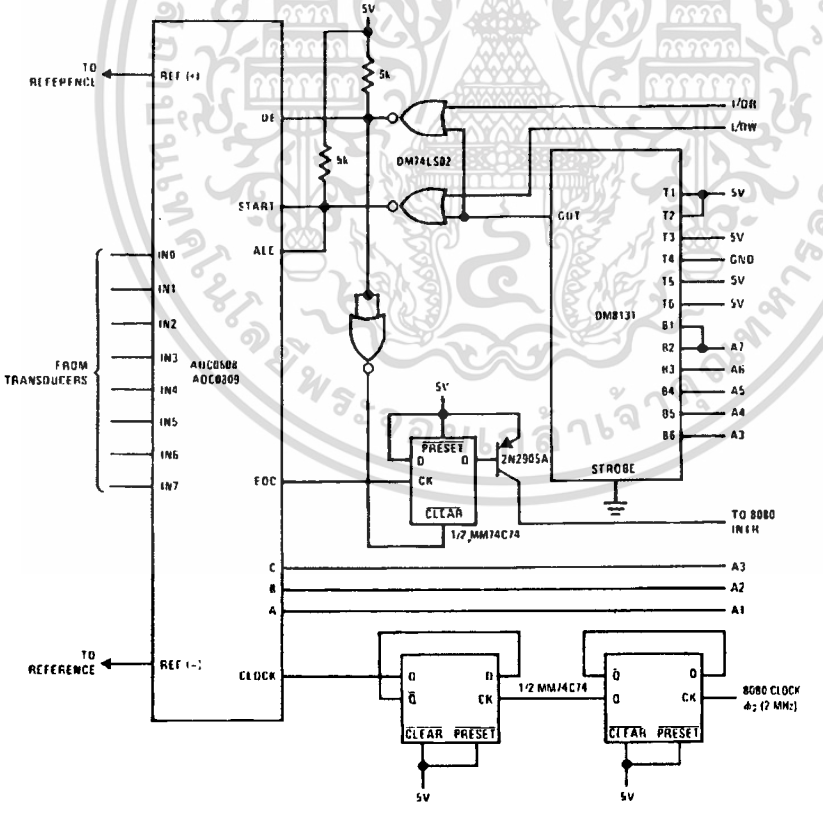
The ADC0808/ADC0809 easily interface to more than one microprocessor. The 6800 can also be used to control the converter. The 6800 has no separate I/O address space so all I/O transfers must be memory mapped. In general more address decoding logic is required to ensure that the I/O ports don't overlap existing memory. For small systems a partial address decoding scheme is shown in *Figure 18*. Generally, if several ports are desired, a small block of memory would be set aside, as is accomplished by the DM8131. *Figure 18* also illustrates a typical 6800 interrupt scheme using a flip-flop and open collector transistor. The interrupt is reset when the data is read. If more ports are needed, a decoder could be added as shown in *Figure 19*. *Figure 19* also illustrates a polled I/O mode using TRI-STATE buffer to gate EOC onto the data bus. As with the INS8080 the A, B, C inputs of the ADC0808/ADC0809 can be connected to the address bus or the data bus.

The 6800 differs from the INS8080 in that the 6800 has a single read/write (R/$\overline{W}$) strobe and a valid memory address (VMA), whereas the INS8080 has separate read and write strobes (I/$\overline{OR}$ and I/$\overline{OW}$). Normally, to obtain a read pulse, VMA, R/$\overline{W}$ and ϕ_2 are gated together and, for a write R/$\overline{W}$ is inverted. ϕ_2 is the 6800 phase 2 system clock. Also notice that the 6800 I$\overline{NT}$ interrupt input is active low. This enables a standard wired-OR open collector design to be implemented.

Figure 20 illustrates some typical 6800 software utility routines for either polled or interrupt interfaces. Again notice double start instructions.

3.3 Z80 Interface

Interfacing the Z80 to the ADC0808 is much the same as interfacing to an 8080/8224/8228 CPU group. CPU instruction timing is very similar, except the read/write control signals are slightly different. Instead of the I/$\overline{OW}$ write strobe there is the I$\overline{OREQ}$ and $\overline{WR}$ and instead of I/$\overline{OR}$, I$\overline{OREQ}$ and $\overline{RD}$ are supplied.



TL/H/5623–11

FIGURE 17. Interrupt-Type 8080/8224/8228 Interface Using 6-Bit Bus Comparator

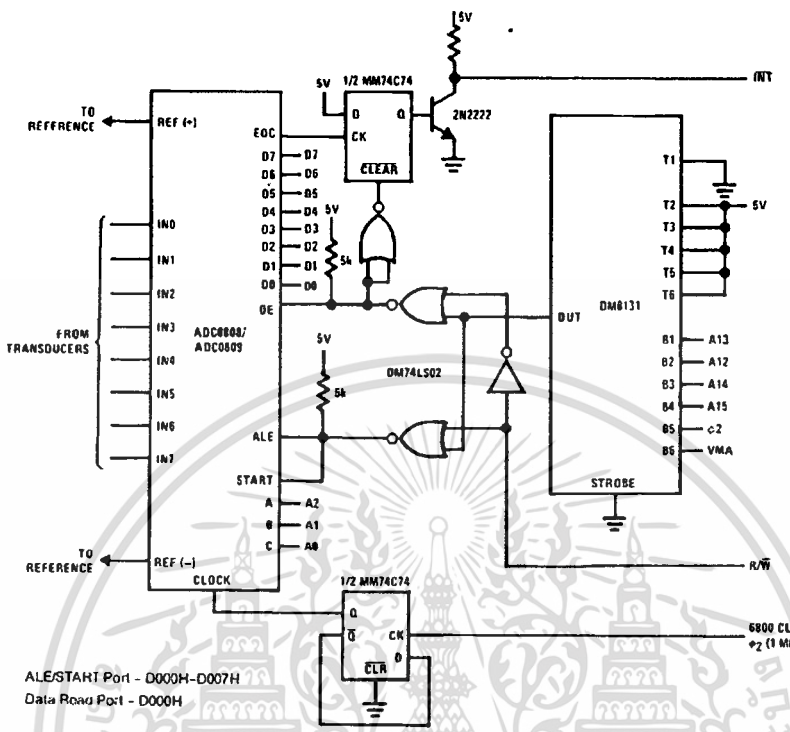


FIGURE 18. Typical 6800 Interface with Partial Address Decoding

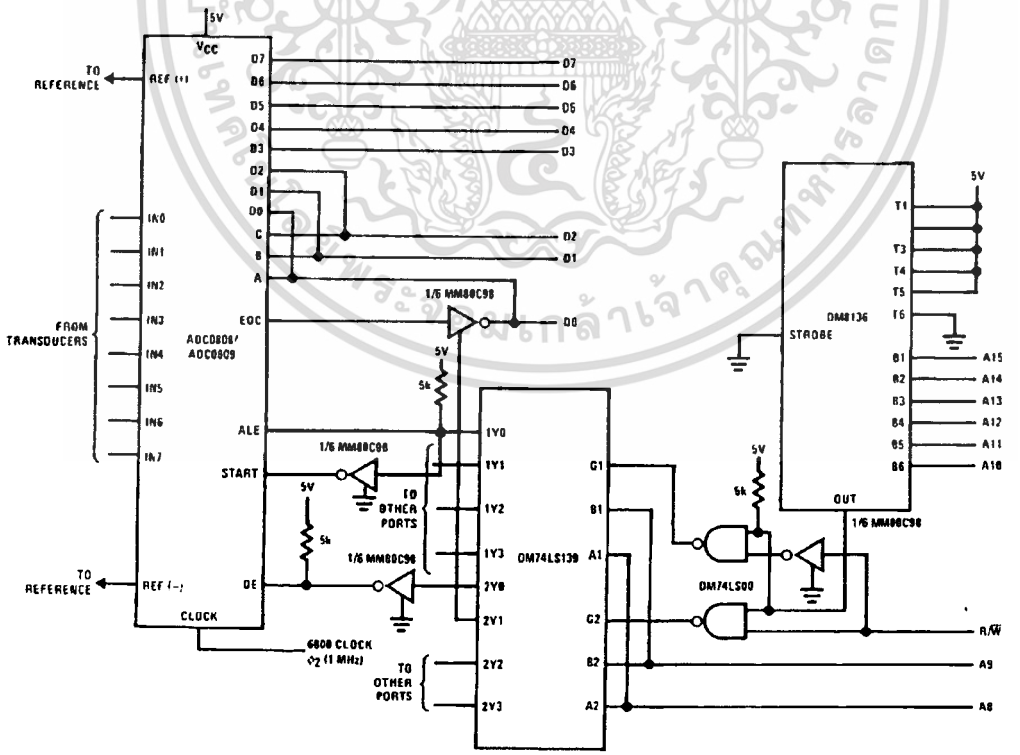


FIGURE 19. Full Decoded 6800 Interface Address

TL/H/5623–12

```
*
*UTILITY ROUTINES FOR ADC0808/ADC0809 INTERFACE
*
*
*LOAD AND START CONVERSION (FIGURE 18)
*
STATUS      EQU     $D800     START ADDRESS FOR CHANNEL 0
DATA        EQU     $D800     CONVERTER DATA ADDRESS
*
*
*
START       STA     STATUS    SELECT CHANNEL 0 AND START
            STA     STATUS    DO AGAIN TO LET INPUTS SETTLE
            LDX     #VECTOR   LOAD INTERRUPT VECTOR ADDRESS
            STX     $FFF8     STORE IT
            — —               EXECUTE MISC PROGRAM
            — —
            — —
            CLI               ENABLE INTERRUPT IF NOT ALREADY
            — —               EXECUTE MISC PROGRAM
            WAI               WAIT FOR INTERRUPT
*
*INTERRUPT HANDLER (FIGURE 18)
*
VECTOR      LDAA    DATA      LOAD DATA RESET INTERRUPT
            CLI               ENABLE INTERRUPTS (OPTION)
            — —               EXECUTE PROGRAM
            RTI               RETURN TO MAIN PROGRAM
*
*START AND TEST CONVERSION POLLED MODE (FIGURE 19)
*
DATA2       EQU     $F800     CONVERTER DATA ADDRESS
CHANN2      EQU     02        CHANNEL 2 ADDRESS
EOCIN       EQU     $F900     EOC INPUT PORT
START2      LDAA    CHANN2    LOAD A ACCUMULATOR
            STAA    STATUS    LOAD ADDRESS AND START
            NOP               WAIT
            STAA    STATUS    RESTART TO LET MUX SETTLE
            NOP               8 NOPS TO WAIT FOR EOC
            — —               TO GO LOW
            LDAA    EOCIN     LOAD EOC STATUS BIT
            ANDA    01        MASK BITS 1–7
            BEQ     READY     IF A = 0 THEN CONVERTER DONE
*
*
            — —               EXECUTE MISC PROGRAM
*
*CONTINUOUS POLLING OF EOC (FIGURE 19)
*
*
POLLIT      LDAA    EOCIN     LOAD EOC STATUS
            ANDA    CHANN2    MASK MSBs
            BNE     POLL IT   IT ACC≠0 NOT READY, LOOP
READY LDAA              DATA  ELSE READ DATA
            — —               CONTINUE PROGRAM
            — —
```

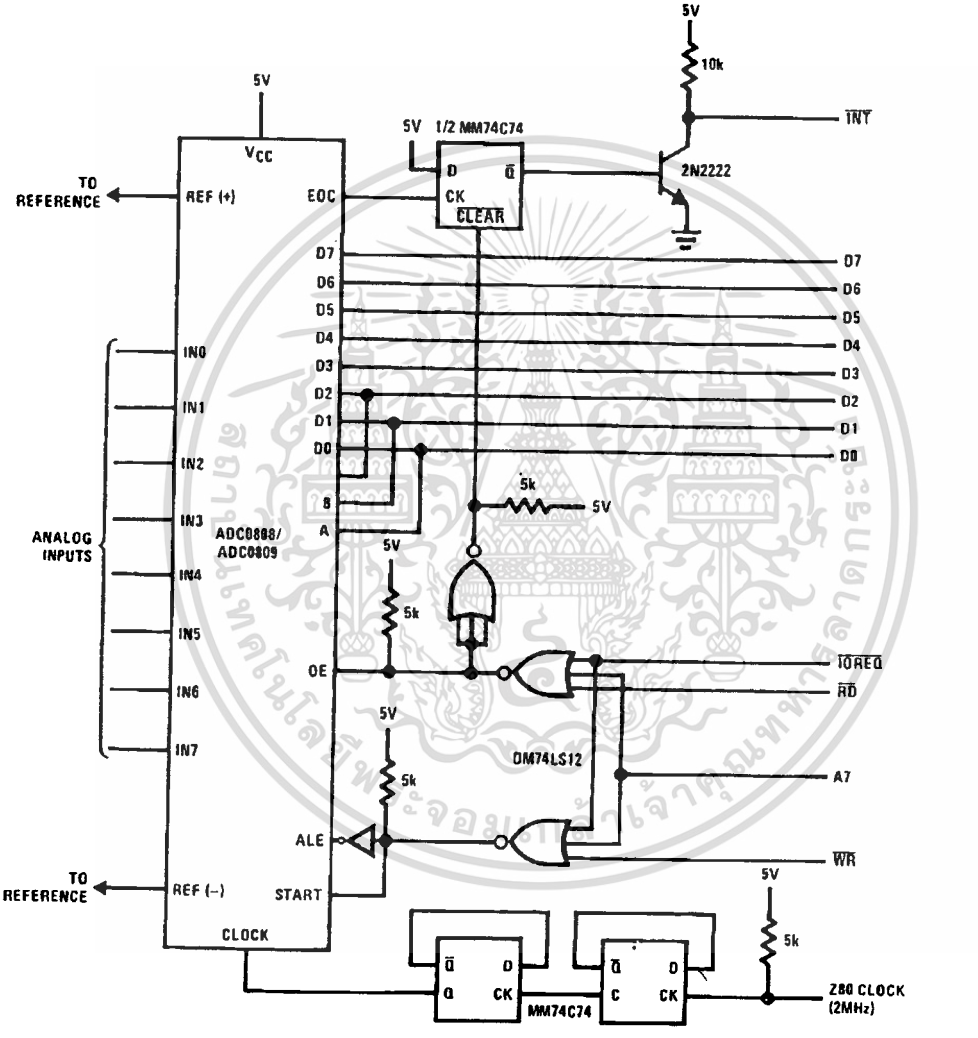
FIGURE 20. Typical I/O Routines for ADC0808/ADC0809 and 6800 Interface

Figure 21 shows a very simple Z80 interface, which is similar to the INS8080 interface of *Figure 13*, except that the interrupt flip-flop design is closer to the 6800 designs. This is because the Z80 $\overline{INT}$ is active low as is the 6800, but the INS8080 INT is active high.

Figure 22 shows a fully decoded bus comparator design where the DM8131 decodes 5 address bits and the $\overline{IOREQ}$ I/O request strobe. Two NOR gates gate the $\overline{RD}$ and $\overline{WR}$ strobes for ALE, START and OE inputs.

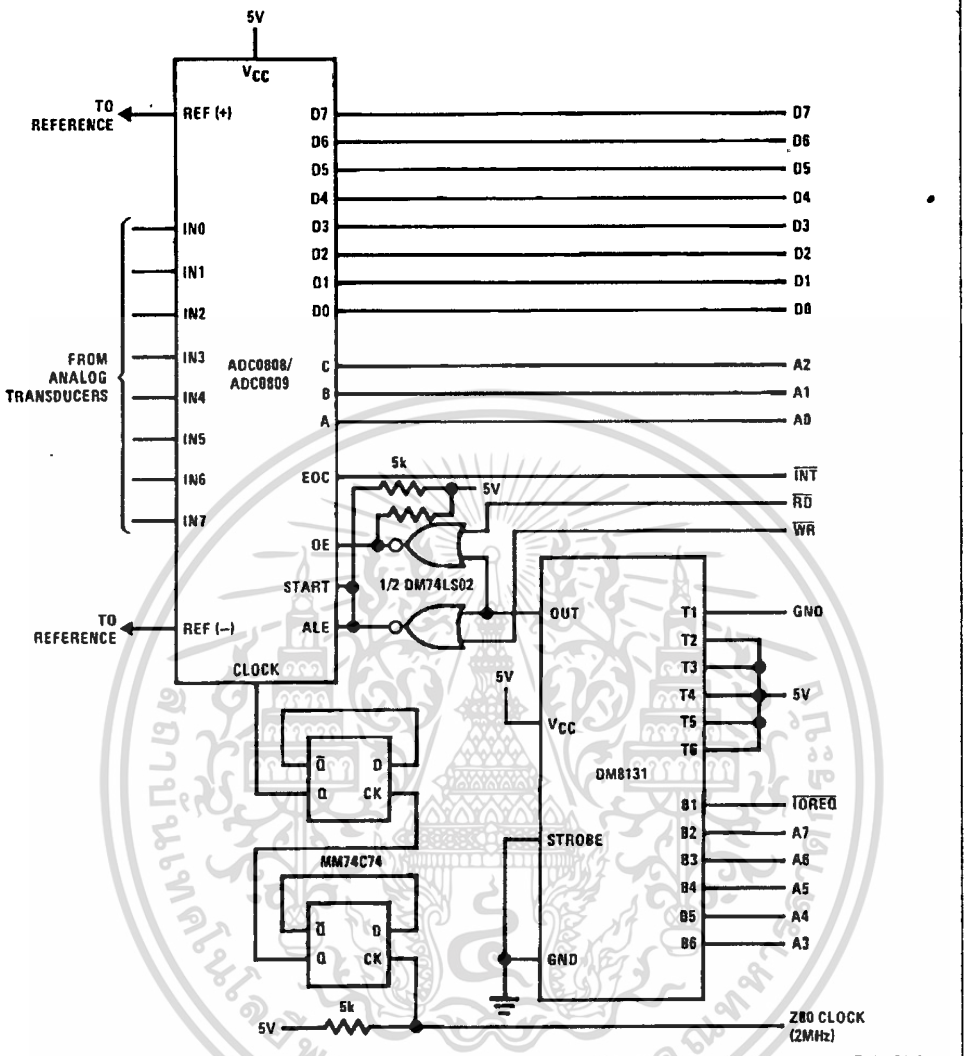
4.0 CONCLUSION

Both the ADC0808 and the ADC0809 can be easily used in microprocessor controlled environments. Many sophisticated medium throughput applications can be handled with a minimum of extra hardware, but additional hardware can increase flexibility and simplify software. Putting both the multiplexer and A/D on the same chip frees the designer from matching multiplexers and A/Ds to implement a 7 or 8-bit accurate system. Design time and overall system cost can be reduced by using these low cost converters.



TL/H/5623–13

FIGURE 21. Simple Z80 Interface



TL/H/5623–14

FIGURE 22. Z80 Partial Decoding Interface

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408