A/D & D/A DEMONSTRATION

โดย

นาย ป่านคม พัฒนพีระเดช
นาย ภัทรชัย อุปริพุทธิกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมระบบควบคุม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

A/D & D/A DEMONSTRATION

โดย

นาย ป่านคม พัฒนพีระเดช รหัส 38014303

นาย ภัทรชัย อุปริพุทธิกุล รหัส 38014365

อาจารย์ที่ปรึกษา

อาจารย์ สว่าง เลิศถิรสุนทร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมระบบควบคุม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

ปริญญานิพนธ์ ปีการศึกษา 2541

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง A/D & D/A DEMONSTRATION

ผู้จัดทำ 1. นาย ปานคม พัฒนพีระเดช รหัส 38014303

2. นาย ภัทรชัย อุปริพุทธิกุล รหัส 38014365

.....อาจารย์ที่ปรึกษา

(อาจารย์ สว่าง เลิศธีรสุนทร)

A/D & D/A DEMONSTRATION

นาย ปานคม พัฒนพีระเดช รหัส 38014303
นาย ภัทรชัย อุปริพุทธิกุล รหัส 38014365

อาจารย์ที่ปรึกษา อ. สว่าง เลิศถิรสุนทร
ปีการศึกษา 2541

บทคัดย่อ

ในปัจจุบันนี้อุปกรณ์ต่าง ๆ ภายในโรงงานอุตสาหกรรมที่ใช้สำหรับตรวจวัดปริมาณและควบคุมการทำงานของระบบต่าง ๆ ได้ถูกออกแบบมาให้สามารถใช้คอมพิวเตอร์เป็นตัวควบคุม เก็บข้อมูล และทำการประมวลผล แต่เนื่องจากคอมพิวเตอร์เป็นอุปกรณ์ที่รับส่งข้อมูลที่อยู่ในรูปแบบดิจิตอล ขณะที่อุปกรณ์ภายนอกทำงานและรับสัญญาณควบคุมในรูปแบบอนาล็อก ดังนั้นจึงจำเป็นจะต้องเชื่อมต่อคอมพิวเตอร์เข้ากับอุปกรณ์ภายนอกโดยการใช้ การตัวแปลงสัญญาณ A/D และ D/A

ซึ่งในปริญญานิพนธ์ฉบับนี้จัดทำขึ้นเพื่อเป็นสื่อการสอนให้เกิดความเข้าใจมากขึ้นในหลักการทำงานของ A/D และ D/A โดยได้ทำการออกแบบและทำการจัดสร้างการ์ดวงจรอินเตอร์เฟสกับคอมพิวเตอร์ โดยภายในประกอบด้วยส่วนวงจร A/D และ D/A ซึ่งจะทำการควบคุมและแสดงผลการทำงานของ A/D และ D/A โดยใช้โปรแกรมภาษา C

Abstract

Today, almost industrial instruments used in control system that have been designed to be controlled, recorded and processed by computer. By the way, computer is processing in digital system but industrial instrument is working in analog system. They can work together by using A/D and D/A circuit.

This Thesis has been made to demonstrate how A/D and D/A can convert Analog to Digital and convert Digital to Analog. We have designed and made Interface Card with inside have A/D and D/A circuit. This Interface Card will be connected to computer and use C language to control and show how A/D and D/A can convert Analog to Digital and convert Digital to Analog

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดี ก็เพราะบุคคลต่างๆหลายท่าน โดยทางผู้จัดทำขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ของผู้จัดทำ ที่ได้ให้การอุปการะอย่างดีเยี่ยม อีกทั้งคอยเป็นกำลังใจให้แก่ผู้จัดทำตลอดเวลา

ขอกราบขอบพระคุณ อ.สว่าง เลิศถิรสุนทร อ.ที่ปรึกษาที่มีความเมตตา ได้กรุณาแนะนำ แก่ผู้จัดทำมาโดยตลอด พร้อมทั้ง คณาจารย์ทุกท่านที่ได้ประสิทธิประสาทความรู้แก่ผู้จัดทำ ผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ด้วย

ขอขอบคุณเพื่อน ๆ ทุกคน ที่ให้ยืมอุปกรณ์ ให้คำแนะนำ ด้วยดีตลอดการทำปริญญานิพนธ์ฉบับนี้

คณะผู้จัดทำ

สารบัญ

สารบัญรูปภาพ

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของโครงงาน

เนื่องมาจากในระบบงานควบคุมต่าง ๆ จะต้องมีการนำเอาสัญญาณจากอุปกรณ์ตรวจจับ (Sensor) มาเข้ายังอุปกรณ์ควบคุม (Controller) เพื่อที่จะทำการวิเคราะห์สัญญาณและส่งต่อออกไปเพื่อทำการควบคุมระบบต่อไป แต่ในการส่งสัญญาณจากอุปกรณ์ตรวจจับเข้าสู่อุปกรณ์ควบคุมนั้น อุปกรณ์ตรวจจับจะทำการส่งสัญญาณออกมาเป็น สัญญาณอนาล็อก (Analog Signal) แต่สำหรับตัวอุปกรณ์ควบคุมนั้น ซึ่งในระบบงานปัจจุบันอาจจะเป็น Microcontroller หรือ คอมพิวเตอร์ เพราะใช้งานง่ายและมีความยืดหยุ่นสูง จะรับสัญญาณอินพุท (Input) เป็นสัญญาณดิจิตอล (Digital) และเช่นเดียวกันในการที่จะส่งสัญญาณจากอุปกรณ์ควบคุมออกไปเพื่อที่จะควบคุมระบบ (Plant) ใดระบบหนึ่งนั้น อุปกรณ์ควบคุมจะส่งสัญญาณออกไปควบคุมเป็นดิจิตอล แต่ระบบจะรับสัญญาณที่จะเข้ามาควบคุมเป็นอนาล็อก ดังนั้นจากสาเหตุดังกล่าวข้างต้น จึงทำให้จะต้องมีอุปกรณ์อย่างใดอย่างหนึ่งที่จะทำหน้าที่แปลงสัญญาณทั้งระหว่างขาเข้าและออกจากอุปกรณ์ควบคุม นั่นคือจะต้องมี Analog to Digital Converter (ADC) ทำหน้าที่แปลงสัญญาณด้านขาเข้าอุปกรณ์ควบคุม และมี Digital to Analog Converter (DAC) ทำหน้าที่แปลงสัญญาณด้านขาออก

สำหรับโครงงานนี้ จัดทำขึ้นเพื่อสาธิตการทำงานของ A/D และ D/A เนื่องจาก ในระหว่างการแปลงสัญญาณจากอนาล็อกเป็นดิจิตอลของ A/D และ การแปลงสัญญาณจากดิจิตอลเป็นอนาล็อกของ D/A นั้น ไม่สามารถดูและพิจารณาขั้นตอนของการแปลงสัญญาณของ A/D และ D/A ซึ่งจะกระทำอยู่ภายในไอซีได้จะสามารถดูได้เฉพาะสัญญาณอินพุทและเอาท์พุทเท่านั้น โครงงานนี้จึงจัดทำขึ้นเพื่อแสดงรูปสัญญาณในระหว่างขั้นตอนการทำงานของ A/D และ D/A โดยใช้คอมพิวเตอร์ (Personal Computer) เป็นตัวแสดงผล

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อสาธิตการหลักการทำงานของ A/D และ D/A
2. เพื่อให้เกิดความเข้าใจมากขึ้น ในวิธีการ Sampling และ Hold
3. เพื่อศึกษาหลักการติดต่ออุปกรณ์ภายนอกกับคอมพิวเตอร์
4. สามารถออกแบบ Card แปลงสัญญาณ A/D และ D/A ที่ใช้ร่วมกับคอมพิวเตอร์ได้
5. เพื่อจะได้ศึกษาหลักการและแนวคิดในการเขียนโปรแกรม เพื่อใช้ในการควบคุมและแสดงผล และจะได้นำไปประยุกต์ใช้งานอื่น ๆ ต่อไปได้ตามวัตถุประสงค์
6. เพื่อที่จะนำไปเป็นสื่อการเรียนการสอนให้กับผู้ที่สนใจเรื่องการติดต่อกับคอมพิวเตอร์และหลักการทำงานของ A/D และ D/A

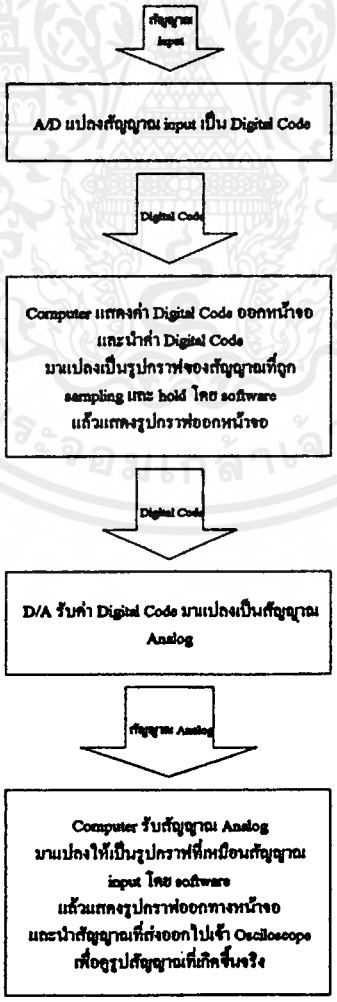
7. เพื่อที่จะเรียนรู้และแก้ไขปัญหาที่เกิดขึ้นในขณะทำการทดลอง

1.3 ขอบเขตของโครงงาน

เพื่อทำการสร้างระบบจำลองการทำงานของ ADC และ DAC โดยที่จะแสดงผลออกมาทางจอ (Monitor) ของคอมพิวเตอร์ โดยมีขั้นตอนการทำงานดังนี้

1. รับสัญญาณอินพุทเป็นอนาล็อกจากภายนอกเป็นค่า -5 - +5 Volt และ 0 – 5 Volt
2. แสดงรูปสัญญาณอินพุทที่ถูก Sampling โดย A/D (โดยมี Software เป็นตัวคำนวณสร้างรูปกราฟขึ้นมา)
3. แสดงรูปสัญญาณอินพุทที่ถูก Sampling ทำการ Hold
4. แสดงค่าของสัญญาณอินพุทที่ถูก A/D แปลงเป็นดิจิตอล ออกมาในรูปแบบตาราง
5. แสดงรูปสัญญาณเอาท์พุทที่จะถูกส่งออกมาและทำการ Hold
6. แสดงรูปสัญญาณเอาท์พุทที่ถูก D/A แปลงออกมาเป็นอนาล็อกโดยจะใช้ออสซิโลสโคปเป็นตัวแสดงผล

โดยสามารถแสดงขั้นตอนเบื้องต้นเป็นบล็อคไดอะแกรม ได้ดังรูปที่ 1.1



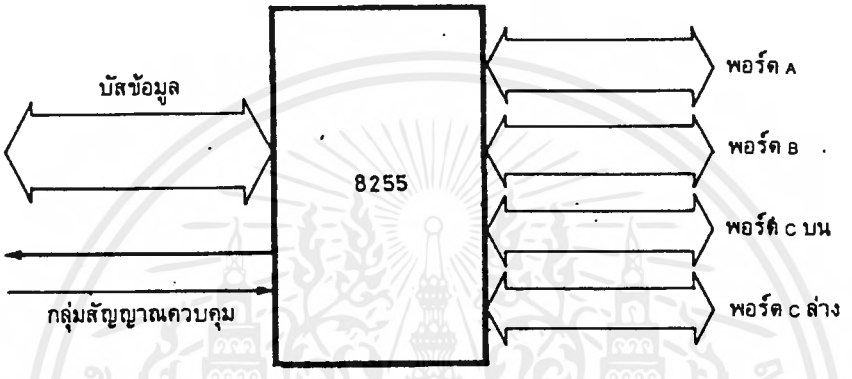
รูปที่ 1.1 แสดงบล็อคไดอะแกรมขั้นตอนการทำงานเบื้องต้นของโครงงาน

บทที่ 2
ทฤษฎี และ หลักการ

2.1 8255 พอร์ตข้อมูลแบบขนาน

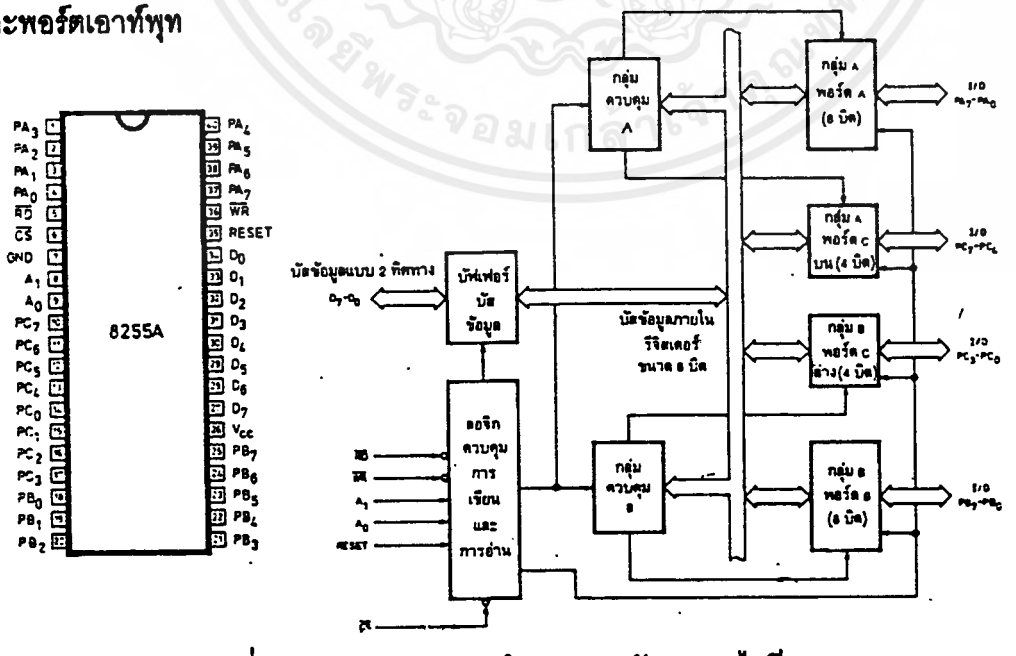
2.1.1 โครงสร้างพื้นฐานของ 8255

8255 เป็นไอซีที่มี 40 ขา ที่ต่อเป็นพอร์ตให้ไมโครโปรเซสเซอร์ ได้ 3 พอร์ต โดยมีโครงสร้างพื้นฐานแสดงดังรูปที่ 2.1



รูปที่ 2.1 แผนผังโครงสร้างของไอซี 8255

การเรียก พอร์ตของ 8255 จะเรียกพอร์ตต่างๆว่า พอร์ต A, พอร์ต B และพอร์ต C โดย พอร์ต C แยกเป็น 2 ส่วน คือ พอร์ต C ล่าง และ พอร์ต C บน โดยทุกพอร์ตเป็นได้ทั้งพอร์ตอินพุท และพอร์ตเอาท์พุท



รูปที่ 2.2 แผนผังวงจรภายในและการจัดขาของไอซี 8255

รูปที่ 2.2 เป็นแผนผังภายในของไอซีและการจัดวางขาของไอซี 8255 การทำงานของ วงจรจะใช้สัญญาณควบคุมจากไมโครโปรเซสเซอร์มาควบคุมการทำงาน โดยไมโครโปรเซสเซอร์จะส่งคำสั่งมาโปรแกรมการทำงานหรือกำหนดรูปแบบของพอร์ตให้เป็นอินพุทหรือเอาท์พุทได้

2.1.2 ขาต่าง ๆของ 8255

ขาทั้ง 40 ขาของไอซีประกอบด้วย

■ D0 – D7 เป็นขาที่ข้อมูลอินพุทเอาท์พุทจะต้องผ่านเข้าออกจากส่วนนี้ D0 – D7 จึงต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์ เพื่อให้ไมโครโปรเซสเซอร์สามารถอ่านหรือเขียนข้อมูลจากพอร์ตผ่านทางบัสนี้

■ $\overline{CS}$ (สัญญาณเลือกชิป) ขานี้เป็นขาอินพุทที่จะรับสัญญาณจากภายนอกเพื่อเลือกชิป 8255 โดยเมื่อขานี้เป็น "0" จะทำให้ 8255 ต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์ เพื่อให้ไมโครโปรเซสเซอร์เขียนหรืออ่านข้อมูลจากพอร์ตได้

■ $\overline{RD}$ (สัญญาณการอ่าน) เป็นสัญญาณอินพุทที่ต้องส่งมาจากซีพียูเมื่อสัญญาณที่ขานี้เป็น "0" และสัญญาณ $\overline{CS}$ เป็น "0" ด้วย ไอซี 8255 จะทำตัวให้ซีพียูอ่านข้อมูลจากบัสขณะที่เป็นพอร์ตอินพุท

■ $\overline{WR}$ เป็นสัญญาณกวรเขียน จะแอคทีฟเมื่อสัญญาณ $\overline{WR}$ และสัญญาณ $\overline{CS}$ เป็น "0" สัญญาณนี้จะมาจากซีพียูเมื่อต้องการเขียนข้อมูลลงบนพอร์ตที่กำหนด

■ A0 –A1 (สัญญาณแอดเดรส) ลอจิกของสัญญาณทั้งสองจะถอดรหัสออกเป็น 4 รหัส เพื่อกำหนดรีจิสเตอร์ภายในที่เชื่อมต่อกับพอร์ตอินพุทเอาท์พุทของ 8255

■ RESET (สัญญาณรีเซท) เป็นสัญญาณที่ส่งจากภายนอกเข้ามาทำการรีเซท 8255 เพื่อเคลียสถานะต่างๆของ 8255 เมื่อ 8255 ได้รับการรีเซท ก็จะกลับเข้าสู่โหมดอินพุทหรือทุกพอร์ตที่เป็นพอร์ตอินพุท

■ PA0 – PA7 เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 ที่ชื่อพอร์ต A การเลือกพอร์ตจะเลือกโดยสัญญาณแอดเดรส A0 – A1

■ PB0 – PB7 เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 ที่ชื่อพอร์ต B การเลือกพอร์ตจะเลือกโดยสัญญาณแอดเดรส A0 – A1

■ PC0 – PC7 เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 ที่ชื่อพอร์ต C การเลือกพอร์ตจะเลือกโดยสัญญาณแอดเดรส A0 – A1 พอร์ต C นี้แบ่งเป็นสองกลุ่ม คือ กลุ่ม PC0 - PC3 และกลุ่ม PC4 - PC7

2.1.3 รีจิสเตอร์ภายในของ 8255

การใช้งาน 8255 จะต้องส่งรหัสควบคุม (control code) เข้าไปยังพอร์ตข้อมูลควบคุมเพื่อควบคุมการทำงานของ 8255 โดยแต่ละพอร์ตจะเสมือนเป็นรีจิสเตอร์ ซึ่งสามารถอ่านและเขียนข้อมูลได้ สัญญาณของขาควบคุมที่ประกอบกันจะแสดงดังตาราง 2.1

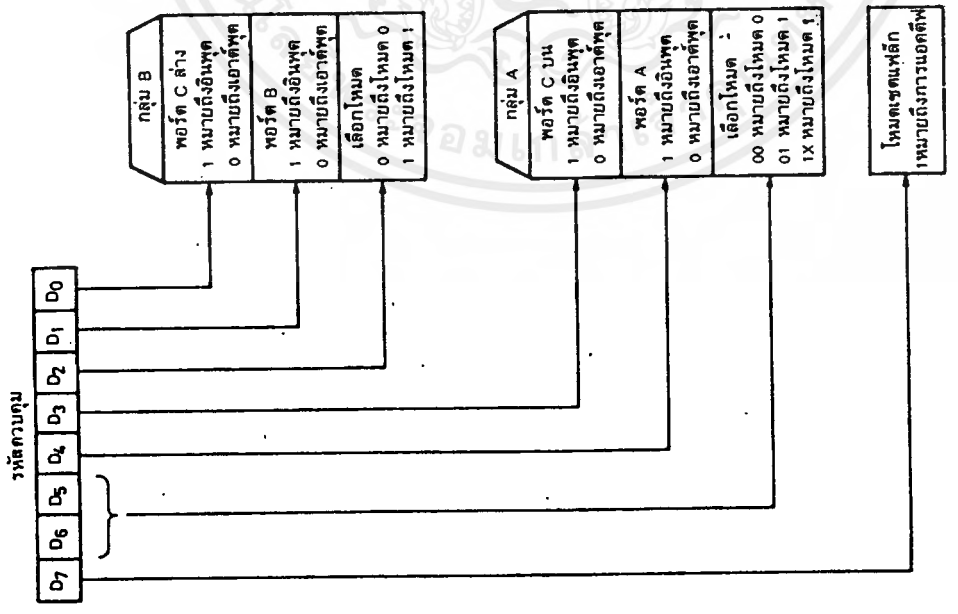
ตารางที่ 2.1 สัญญาณควบคุมการกระทำของ 8255

$\overline{RD}$	$\overline{WR}$	$\overline{A1}$	$\overline{A0}$	ความหมาย
1	0	0	0	เขียนพอร์ต A ซึ่งเป็นข้อมูล
0	1	0	0	อ่านพอร์ต A ซึ่งเป็นข้อมูล
1	0	0	1	เขียนพอร์ต B ซึ่งเป็นข้อมูล
0	1	0	1	อ่านพอร์ต B ซึ่งเป็นข้อมูล
1	0	1	0	เขียนพอร์ต C ซึ่งเป็นข้อมูล
0	1	1	0	อ่านพอร์ต C ซึ่งเป็นข้อมูล
1	0	1	1	เขียนข้อมูล ซึ่งเป็นรหัสควบคุม
0	1	1	1	อ่านเข้ามา ซึ่งไม่มีความหมายใด

ในการควบคุมการทำงานของ 8255 จะมีอยู่หลายโหมด แต่ละโหมดจะแตกต่างกันไป การโปรแกรมให้ 8255 ทำงานจะทำได้ 3 โหมด คือ โหมด0 โหมด1 และโหมด2

2.1.4 รหัสควบคุม

การกำหนดโหมดการทำงานของ 8255 นั้น จะต้องส่งข้อมูลคำสั่งเข้าไปโปรแกรมในพอร์ตควบคุมของ 8255 แต่ละบิทของข้อมูลที่ส่งไปจะมีความหมายในตัวเอง ลักษณะความหมายของแต่ละบิทในรหัสควบคุมแสดงได้ดังรูป 2.3



รูปที่ 2.3 แสดงความหมายของบิทต่าง ๆ ในรหัสควบคุม

การโปรแกรม 8255 คือ การให้ค่ารหัสบิทต่างๆเข้าไปในรหัสควบคุมแล้วส่งไปยังรีจิสเตอร์ของพอร์ตควบคุม ความหมายของบิทต่างๆมีดังนี้

- บิท D7 เป็นบิทที่แสดงรหัสคำสั่งควบคุม ถ้าบิทนี้เป็น "1" หมายถึงรหัสควบคุมนี้จะมีผลต่อการเปลี่ยนแปลงการเซตโหมดต่างๆของ 8255

- บิท D6 และ D5 เป็นการเลือกโหมดของพอร์ต A ซึ่งมี 3 โหมด คือ โหมด0 โหมด1 และโหมด2

- บิท D4 ถ้ามีค่าเป็น "0" หมายถึงการกำหนดพอร์ต A เป็นเอาท์พุท ถ้ามีค่าเป็น "1" จะหมายถึงการกำหนดให้พอร์ต A เป็นอินพุท

- บิท D3 เป็นบิทที่บอกถึงการเซตพอร์ต C บน ถ้าเป็น "0" จะทำให้พอร์ต C บน เป็นเอาท์พุท ถ้าเป็น "1" จะเป็นอินพุท

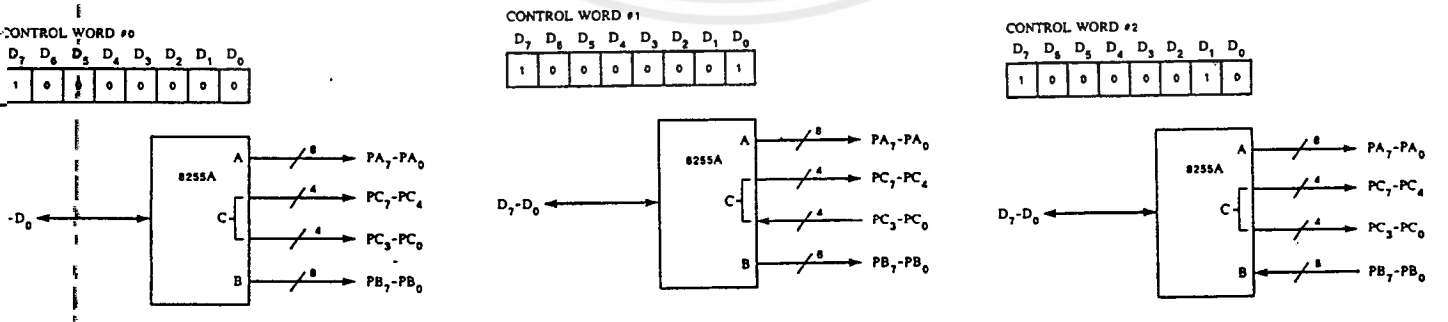
- บิท D2 เป็นบิทที่บอกถึงการเซตโหมดของพอร์ต B ถ้าเป็น "0" หมายถึง เลือกพอร์ต B เป็น โหมด 0 และถ้าเป็น "1" หมายถึงหารเลือกโหมด 1

- บิท D1 ถ้ามีค่าเป็น "0" หมายถึงการกำหนดพอร์ต B เป็นเอาท์พุท ถ้ามีค่าเป็น "1" จะหมายถึงการกำหนดให้พอร์ต B เป็นอินพุท

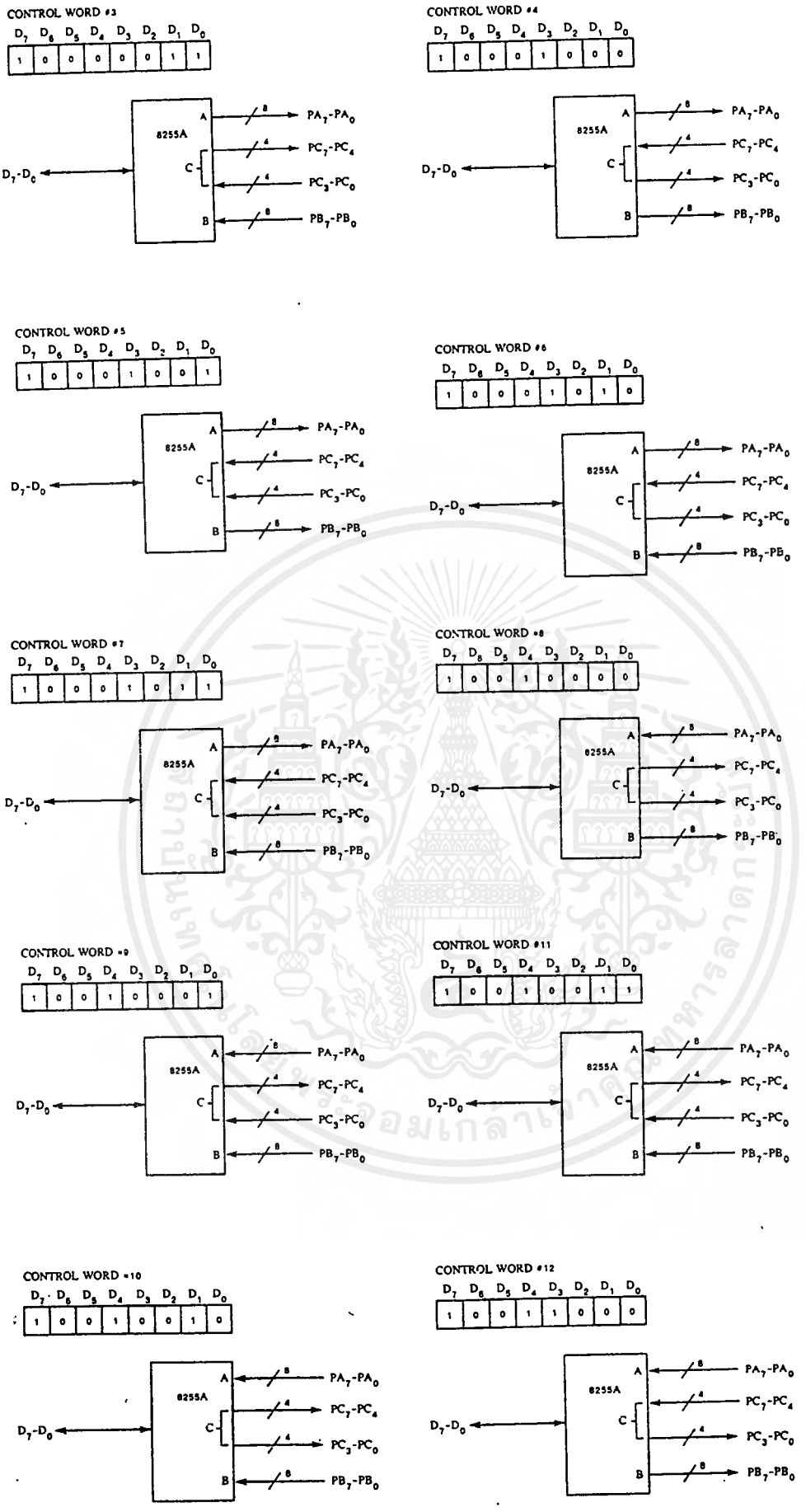
- บิท D0 ถ้ามีค่าเป็น "0" หมายถึงการกำหนดพอร์ต C ล่าง เป็นเอาท์พุท ถ้ามีค่าเป็น "1" จะหมายถึงการกำหนดให้พอร์ต C ล่าง เป็นอินพุท

2.1.5 การทำงานในโหมด 0

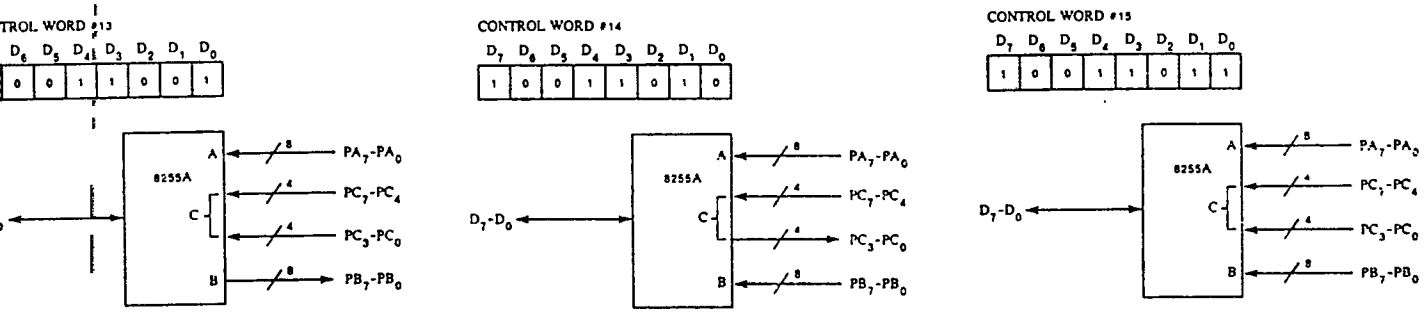
โหมด 0 เป็นโหมดที่กำหนดให้พอร์ตทุกตัวบน 8255 เป็นพอร์ตอินพุทหรือเอาท์พุทแบบพื้นฐาน รูปแบบความเป็นไปได้จึงมีทั้งสิ้น 16 รูปแบบตามลักษณะของพอร์ต A พอร์ต B พอร์ต C บน และพอร์ต C ล่าง ลักษณะของรหัสควบคุมแต่ละแบบจะเป็นดังรูป 2.4



รูปที่ 2.4 ลักษณะของรหัสควบคุมแบบต่าง ๆ ในโหมด 0



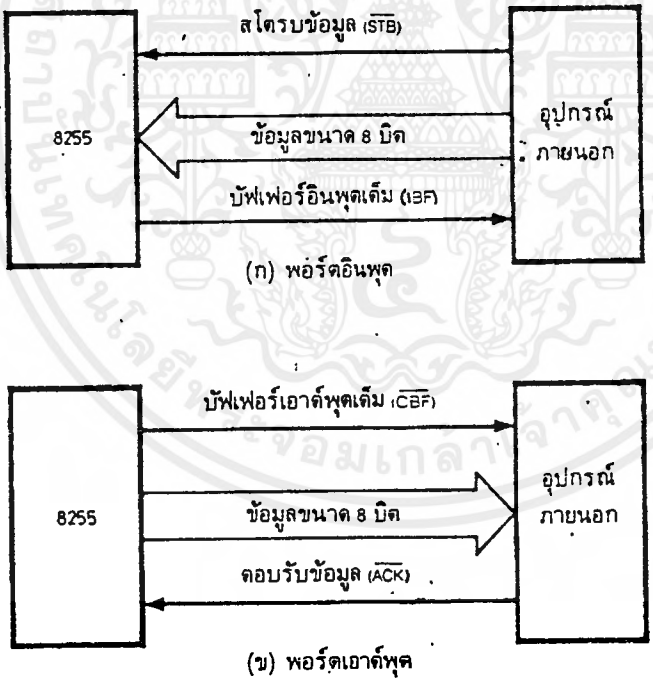
รูปที่ 2.4 (ต่อ) ลักษณะของรหัสควบคุมแบบต่าง ๆ ในโหมด 0



รูปที่ 2.4 (ต่อ) ลักษณะของรหัสควบคุมแบบต่าง ๆ ในโหมด 0

2.1.6 การทำงานในโหมด 1

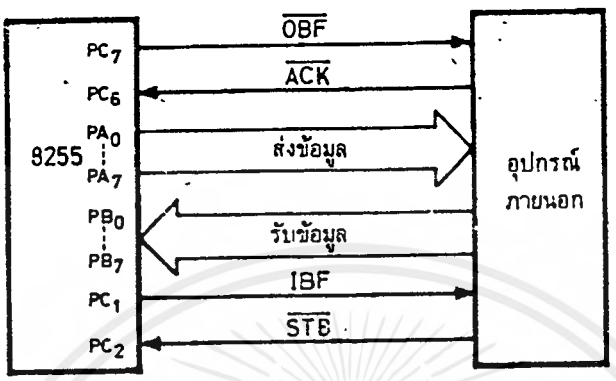
การทำงานของ 8255 ในโหมด1 เป็นโหมดที่ทำให้อินพุทเอาท์พุทมีการตรวจสอบสัญญาณ (handshaking) โดยใช้อินพุทเอาท์พุทของพอร์ต A และ พอร์ต B เป็นหลัก และใช้พอร์ต C บนเป็นตัวตรวจสอบสัญญาณ (handshake) ของพอร์ต A ส่วนพอร์ต C ล่างเป็นตัวตรวจสอบสัญญาณของพอร์ต B การจัดสัญญาณต่างๆ เหล่านี้แสดงดังรูปที่ 2.5



รูปที่ 2.5 โครงสร้างตัวตรวจสอบสัญญาณของพอร์ตอินพุทและพอร์ตเอาท์พุท

แนวความคิดของการใช้พอร์ตอินพุทเอาท์พุทโดยมีตัวตรวจสอบสัญญาณก็เพื่อให้มีการซิงโครไนซ์ระหว่างอุปกรณ์ภายนอกที่ทำงานได้ช้ากับการทำงานของคอมพิวเตอร์ที่ทำงานได้เร็ว เช่น เครื่องพิมพ์ทำงานได้ช้า เมื่อคอมพิวเตอร์ส่งอักษรตัวแรกมา เครื่องพิมพ์รับตัวอักษรและ

กำลังจะพิมพ์ คอมพิวเตอร์ก็ส่งอักษรตัวที่ 2 ตัวที่ 3 ตามมา ทำให้การประมวลผลของอุปกรณ์เครื่องพิมพ์ทำงานไม่ทัน ซึ่งอาจทำให้ข้อมูลสูญหาย ดังนั้นเครื่องพิมพ์จึงส่งสัญญาณบอกคอมพิวเตอร์ว่า "อย่าเพิ่งส่งเพราะยังไม่พร้อมจะรับ" ดังรูป 2.5 จะใช้ PA0 - PA7 เป็นเอาท์พุท และ PB0 - PB7เป็นอินพุท โดยมีพอร์ต C เป็นตัวตรวจสอบสัญญาณ ดังรูป 2.6



รูปที่ 2.6 วงจรการต่อ 8255 ในโหมด 1

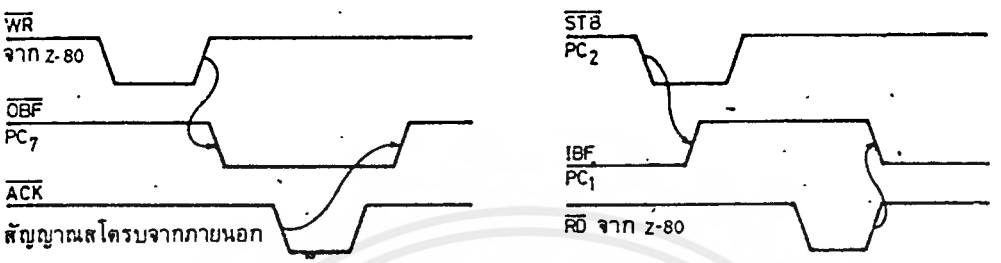
เมื่อโปรแกรม 8255 เป็นโหมด1 แล้ว ตัว8255 จะให้พอร์ต C เป็นสัญญาณควบคุม โดยแต่ละบิทของพอร์ต C เป็นไปตามตาราง 2.2

ตารางที่ 2.2 หน้าที่ของสัญญาณต่างๆ ของพอร์ต C ในการทำงานเป็นตัวตรวจสอบสัญญาณเมื่อ 8255 ทำงานในโหมด 1

ขา	กรณีอินพุต	กรณีอาต์พุต
PC_0	$INTR_B$	$INTR_B$
PC_1	IBF_B	$\overline{OBF}_B$
PC_2	$\overline{STB}_B$	$\overline{ACK}_B$
PC_3	$INTR_A$	$INTR_A$
PC_4	$\overline{STB}_A$	I/O
PC_5	IBF_A	I/O
PC_6	I/O	$\overline{ACK}_A$
PC_7	I/O	$\overline{OBF}_A$

โดยปกติ 8255 จะให้สัญญาณอินเตอร์รัพต์ไปบอก CPU ด้วย สัญญาณอินเตอร์รัพต์ของ 8255 จะเกิดขึ้นที่ PC0 และ PC3 โดยที่เมื่อบัฟเฟอร์พร้อมแล้วและต้องการให้ CPU ส่งอินพุทหรือเอาท์พุทมาที่บัฟเฟอร์ สัญญาณอินเตอร์รัพต์จะเกิดขึ้น

โครงสร้างการตรวจสอบสัญญาณของ 8255 แสดงด้วยสัญญาณทางไฟฟ้าได้ดังรูปที่ 2.7



(ก) เมื่อเป็นพอร์ตเอาต์พุต (ข) เมื่อเป็นพอร์ตอินพุต

รูปที่ 2.7 แผนผังเวลาการรับและส่งข้อมูลโดยใช้ตัวตรวจสอบสัญญาณ

สังเกตว่า การทำงานของ 8255 จะเกี่ยวข้องกับสัญญาณ $\overline{RD}$ และ $\overline{WR}$ ซึ่งจะทำให้สัญญาณควบคุมมีสถานะเปลี่ยนแปลงไป ทำให้การตรวจสอบสัญญาณซึ่งกันและกัน เป็นวิธีการรับส่งที่มีประสิทธิภาพ เช่น ในกรณีอินพุท เมื่ออุปกรณ์ภายนอกต้องการส่งข้อมูลให้ CPU ก็จะส่งข้อมูลแบบขนานเข้ามาพร้อมทั้งสโตรบ ($\overline{STB}$) บอก 8255 ซึ่ง 8255 จะนำข้อมูลนั้นไปเก็บไว้ที่รีจิสเตอร์ภายในก่อน แล้วส่งสัญญาณ (IBF) บอกว่าบัฟเฟอร์ยังเก็บค่าเอาไว้เต็มอยู่ไม่สามารถข้อมูลที่จะส่งเข้ามาใหม่ได้ จนเมื่อ CPU อ่านข้อมูลจากรีจิสเตอร์ไปแล้ว สัญญาณ (IBF) ก็จะเป็น "0" เพื่อบอกว่าส่งข้อมูลใหม่มาได้

ในทำนองเดียวกัน สำหรับพอร์ตเอาท์พุท เมื่อ CPU ส่งข้อมูออกมาทางพอร์ตเอาท์พุทให้กับ 8255 ตัว8255 ก็จะรับไว้ในรีจิสเตอร์ภายใน พร้อมทั้งส่งสัญญาณออกไปบอกอุปกรณ์ภายนอก ($\overline{OBF}$) ว่า มารับข้อมูลที่เอาท์พุทบัฟเฟอร์ อุปกรณ์ภายนอกเมื่อรับทราบและพร้อมจะอ่านก็จะส่งสัญญาณตอบ ($\overline{ACK}$) เพื่อแสดงว่าอ่านข้อมูลไปแล้ว

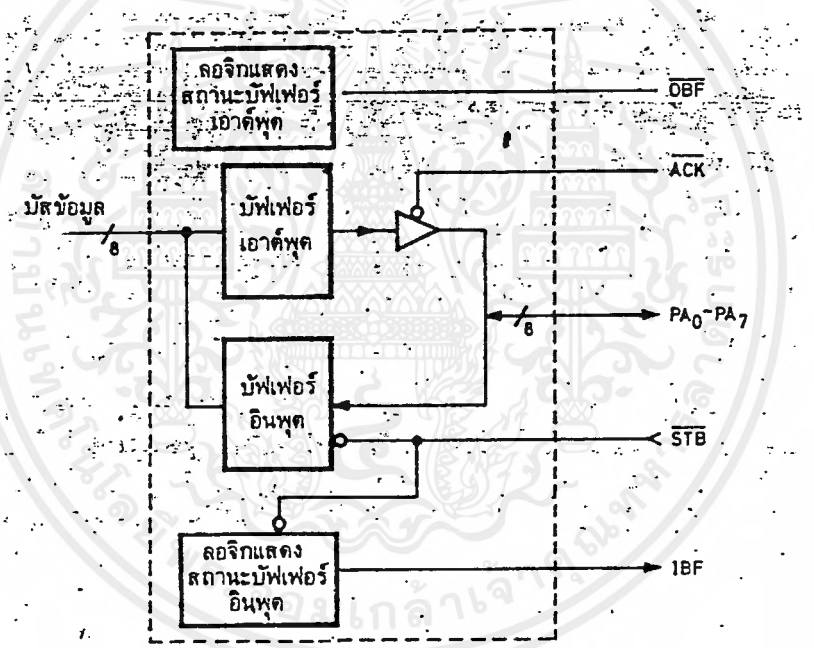
2.1.7 การทำงานในโหมด 2

การทำงานในโหมด 2 จะทำได้เฉพาะพอร์ต A โดยจะใช้พอร์ต A ทำหน้าที่แบบสองทิศทาง คือสามารถเป็นได้ทั้งพอร์ตอินพุทและเอาท์พุท โดยโครงสร้างของพอร์ต A ทั้งอินพุทเอาท์พุทมีตัวตรวจสอบสัญญาณทั้งคู่ ส่วนพอร์ต C จะทำหน้าที่เป็นสัญญาณตรวจสอบ โดยมีสัญญาณแต่ละขาดังตารางที่ 2.3

ตารางที่ 2.3 หน้าที่ของพอร์ต C ในโหมด 2

พอร์ต c	ความหมาย
PC_0	I/O
PC_1	I/O
PC_2	I/O
PC_3	$INTR_A$
PC_4	$\overline{STB}_A$
PC_5	IBF_A
PC_6	$\overline{ACK}_A$
PC_7	$\overline{OBF}_A$

ส่วนโครงสร้างของพอร์ต A ที่ทำงานแบบ 2 ทิศทาง สามารถแสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 โครงสร้างของพอร์ต A ที่ทำงานแบบพอร์ต 2 ทิศทาง

สังเกตว่า เมื่อโปรแกรมพอร์ต A เป็นโหมด 2 แล้ว พอร์ต B จะต้องโปรแกรมเป็นโหมด 0 หรือโหมด 1 ก็ได้ ซึ่งก็ทำงานแบบแยกอิสระอีก ในการใช้งานพอร์ตแบบ 2 ทิศทางนี้ ใช้ได้กับงานบางประเภท เช่นใช้ในการรับส่งข้อมูลของพอร์ตมาตรฐานบางประเภทเช่น IEEE 488 หรือใช้เชื่อมโยงระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ในการรับส่งข้อมูลสลับกันไปและกลับ

2.2 การแปลงอนาล็อกเป็นดิจิตอล (Analog to Digital Conversion)

2.2.1 แนะนำ

ในการที่จะให้ไมโครคอมพิวเตอร์รับรู้สัญญาณอนาล็อกได้ จะต้องมีการแทนสัญญาณอนาล็อกระดับค่าหนึ่ง ๆ ด้วยค่าของตัวเลขไบนารี่ เราเรียกว่าการแปลงอนาล็อกเป็นดิจิตอล ในตัวแปลง A/D บางชนิดจะมีการใช้ตัวแปลงเป็นดิจิตอล (D/A Conversion) อยู่ภายในตัวแปลงด้วย วิธีการใช้ในการแปลง A/D มีหลายวิธี ที่มีใช้ในปัจจุบันอาจแยกได้ 4 วิธีดังนี้

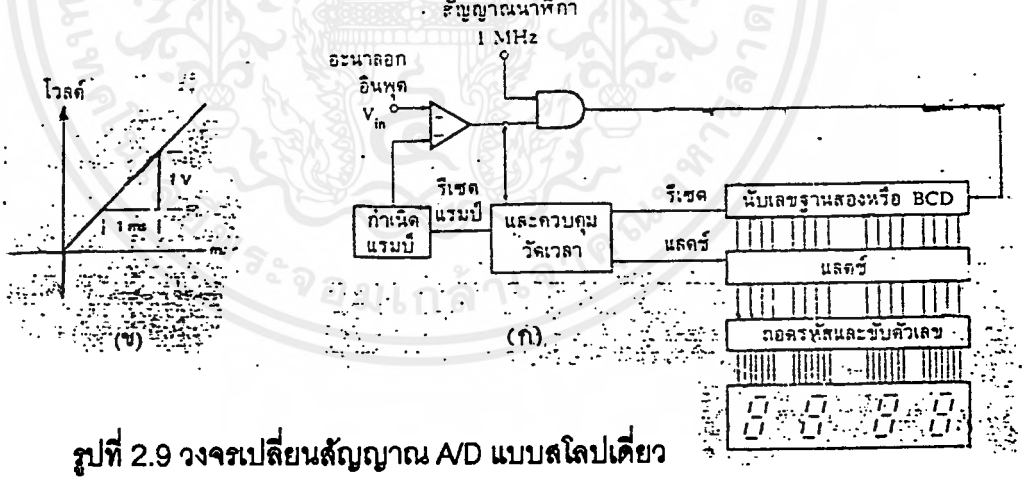
1. แบบอินทิเกรท (Integrating)
2. แบบขนาน (Flash)
3. แบบไบนารี่แรมพ์ (Binary Ramp)
4. แบบประมาณค่าอย่างต่อเนื่อง (Successive Approximation)

2.2.2 A/D Converter แบบที่ใช้การอินทีเกรท

วงจรเปลี่ยนสัญญาณ A/D ที่ใช้เทคนิคการอินทีเกรทสัญญาณมี 3 แบบ คือ

2.2.1.1 <u>แบบสโลปเดี่ยวหรือแบบแรมป์</u> (Single Ramp หรือ Single Slope A/D Converter)

วงจร A/D แบบนี้แสดงไว้ในรูป 2.9 ประกอบด้วยวงจรกำเนิดสัญญาณแรมป์,วงจรเปรียบเทียบ,วงจรนับ BCD หรือ นับเลขฐานสอง



รูปที่ 2.9 วงจรเปลี่ยนสัญญาณ A/D แบบสโลปเดี่ยว

(ก) แสดงบล็อกไดอะแกรม (ข)ความชันของสัญญาณแรมป์

เมื่อเริ่มทำการเปลี่ยนสัญญาณ สัญญาณแรมป์และวงจรนับจะถูกรีเซตให้เป็น 0 แรงดันอนาล็อกถูกป้อนไปยังวงจรเปรียบเทียบทางขาอินพุทแบบไม่กลับ เมื่อแรงดันอินพุทที่ขานี้เป็นบวกมากกว่าที่ขาอินพุทแบบกลับ วงจรเปรียบเทียบก็จะให้เอาท์พุทระดับเป็น "High" ทำให้แอนด์เกตปล่อยสัญญาณนาฬิกาผ่านไปยังวงจรนับได้ และทำให้เริ่มเกิดสัญญาณแรมป์

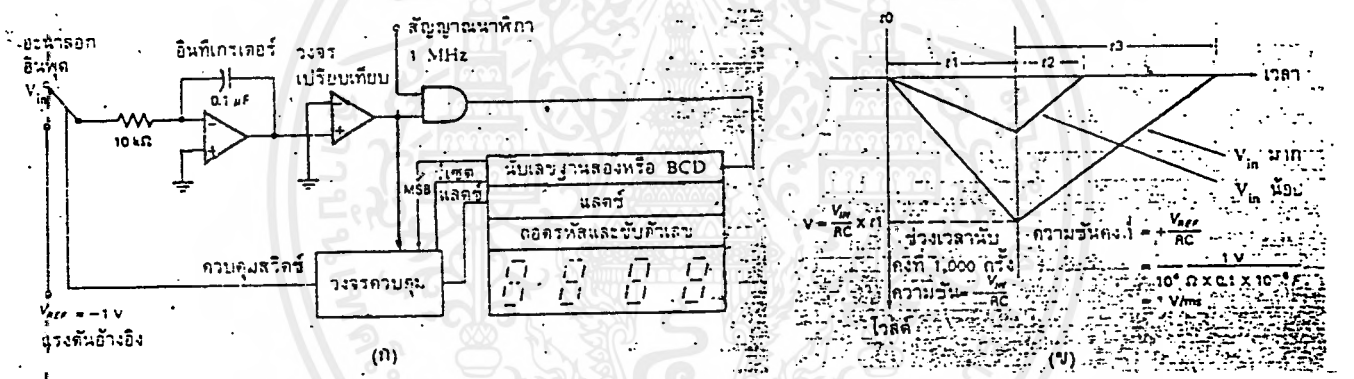
สัญญาณแรมป์มีแรงดันเป็นบวกขึ้นเรื่อย ๆ จนมากกว่าระดับแรงดันอินพุท เอาท์พุทของวงจรเปรียบเทียบก็ตกลงมาเป็นระดับ "Low" ส่งผลทำให้แอนด์เกตไม่มีสัญญาณผ่านไปให้วงจรนับ

วงจรนับจะหยุดนับและเก็บค่าไว้ที่วงจรแลตช์ จากนั้นจึงทำการรีเซตวงจรนับและวงจรกำเนิดสัญญาณแรมป์

ข้อเสียของวงจรแบบนี้คือไม่ควรใช้กับงานที่ต้องการความถูกต้องสูง เนื่องจากการเปลี่ยนแปลงในแหล่งกำเนิดสัญญาณแรมป์ขึ้นกับอุณหภูมิและผลตอบสนองต่อสัญญาณอินพุททำให้ไม่มีความคงที่ ดังนั้นจึงมีการปรับปรุงให้ดีขึ้นเป็นแบบสโลปคู่ (dual-slope)

2.2.1.2 <u>แบบสโลปคู่</u> (Dual-Slope A/D Converter)

ดังรูปที่ 2.10 แสดงบล็อกไดอะแกรมของวงจร A/D แบบสโลปคู่ ซึ่งวงจรส่วนใหญ่คล้ายกับแบบสโลปเดี่ยว แต่มีสวิตช์ที่อินพุทเพิ่มขึ้นเพื่อทำการเลือกระหว่างแรงดันอินพุทกับแรงดันอ้างอิง (วงจรเปรียบเทียบต่อขาสัญญาณอินพุทกลับกันกับแบบสโลปเดี่ยว)



รูปที่ 2.10 วงจรเปลี่ยนสัญญาณ A/D แบบสโลปคู่

(ก) แสดงบล็อกไดอะแกรม (ข) เอาท์พุทของวงจรอินทีเกรเตอร์เมื่อเทียบกับเวลา

ส่วนประกอบของวงจรมี วงจรกำเนิดสัญญาณแรมป์หรือวงจรอินทีเกรเตอร์ ซึ่งจะกำเนิดสัญญาณแรมป์ที่เป็นเชิงเส้น แต่มีสโลปเป็นลบ ซึ่งวงจรเปรียบเทียบก็จะได้รับแรงดันเป็นลบเข้ามาแล้วให้เอาท์พุทเป็นบวกทำการเปิดแอนด์เกตให้สัญญาณนาฬิกาผ่านเข้าสู่วงจรนับ วงจรนับจะนับไปจนถึงค่าคงที่ที่กำหนดไว้ แล้วสับสวิตช์ต่อเข้ากับแรงดันอ้างอิง ซึ่งจะให้แรมป์ที่สโลปเป็นบวกจนมีค่าเป็น 0 วงจรนับจะเก็บค่าที่เวลานั้นไว้ ซึ่งค่าในวงจรนับจะเป็นสัดส่วนโดยตรงกับแรงดันอินพุท

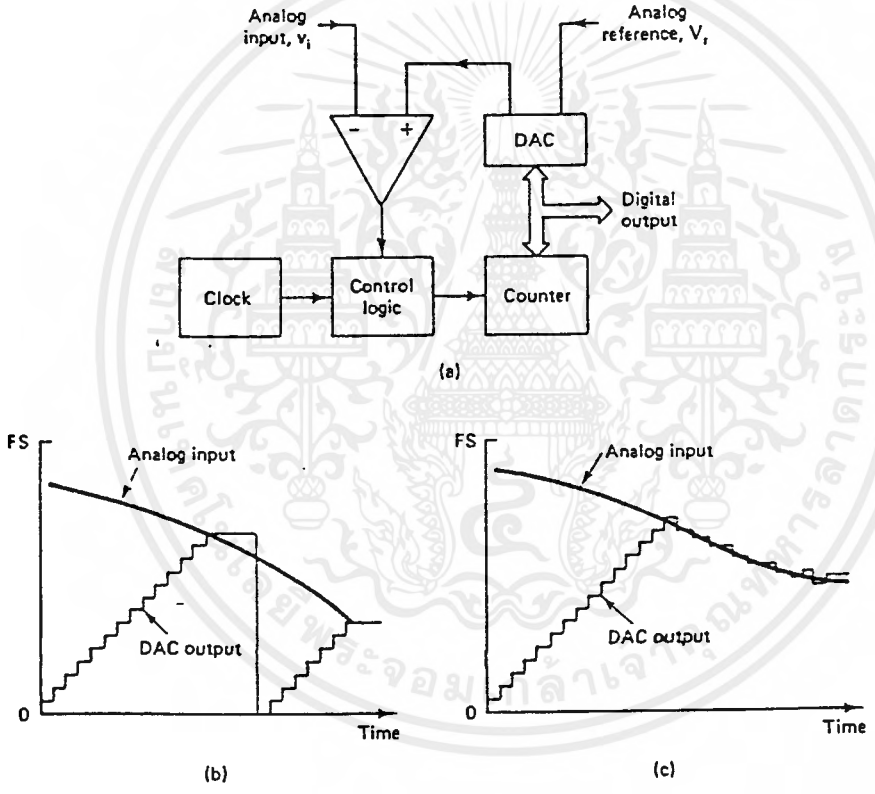
ข้อดีของวงจรแบบนี้ คือ ความถูกต้องสูง,ราคาถูก,เสถียรภาพทางด้านอุณหภูมิดี ข้อเสียคือความเร็วต่ำ ในการเปลี่ยนสัญญาณ 1 ครั้งอาจใช้เวลาถึง 100 ms

2.2.1.3 แบบชาร์จบาลานซ์ (Charge Balance A/D Converter)

วงจรแบบชาร์จบาลานซ์นี้คล้ายกับแบบสโลปคู่ แต่แทนที่จะให้อินพุท สวิตช์ไปมาระหว่างแรงดันที่ไม่รู้ค่ากับแรงดันอ้างอิง ก็ทำการแทรกพัลส์ของกระแสอ้างอิงมาตรงๆ ที่จุดรวม (Summing Point) ของวงจรอินทีเกรเตอร์ในช่วงเวลาคงที่ โดยที่จำนวนพัลส์จะเป็นสัดส่วนโดยตรงกับแรงดันอินพุทที่ไม่รู้ค่า

2.2.2 A/D Converter แบบที่ใช้วงจรนับและวงจร D/A ประกอบกัน

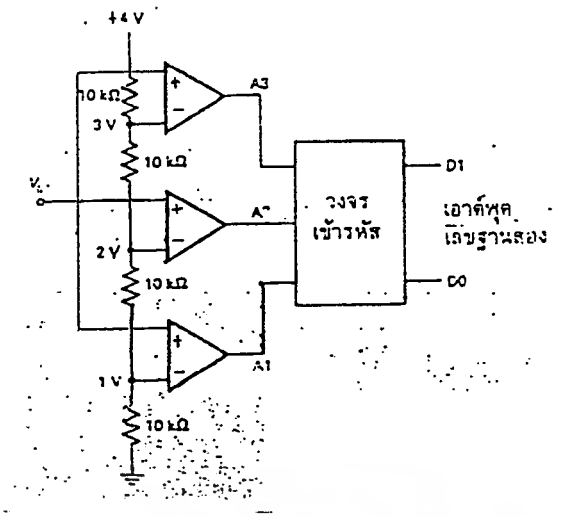
รูปที่ 2.11 แสดงบล็อกไดอะแกรมของวงจร โดยที่มีหลักการเช่นเดียวกันกับแบบสโลปเดี่ยว เพียงแต่สโลปของวงจรนี้ถูกสร้างโดยวงจรนับและวงจร DAC โดยแรมป์นี้จะมีลักษณะเป็นขั้นบันไดเล็ก ๆ จำนวนมากนั่นเอง



รูปที่ 2.11 แสดง A/D แบบวงจรนับและวงจร D/A ประกอบกัน
(a) แสดงบล็อกไดอะแกรม (b) แสดงรูปคลื่นแบบวงจรนับ (c) แสดงรูปคลื่นแบบ tracking

2.2.3 A/D converter แบบใช้วงจรเปรียบเทียบขนานหรือแบบ "แฟลช" (Parallel Comparator Simultaneous "Flash" A/D converter)

วงจร A/D แบบนี้ใช้วงจรเปรียบเทียบที่ต่อขนานกัน ดังรูป 2.12 ประกอบด้วยออปแอมป์ที่ต่อเป็นวงจรเปรียบเทียบ และตัวต้านทานต่อไว้เพื่อแบ่งแรงดันที่ขาอินพุทแบบกลับให้มีขนาดต่าง ๆ



รูปที่ 2.12 แสดงการต่อวงจร Parallel Comparator A/D Converter

จากหลักการของวงจรเปรียบเทียบทั่วไป เมื่อแรงดันอินพุทที่ขาอินพุทแบบไม่กลับ มีค่าสูงกว่าที่ขาอินพุทแบบกลับ เอาท์พุทจะได้ค่าสูง ดังตาราง 2.4 แสดงความสัมพันธ์ระหว่างแรงดันอินพุทที่เป็นอะนาลอกกับเอาท์พุทที่เป็นดิจิตอล

ตารางที่ 2.4 ตารางแสดงความสัมพันธ์ระหว่างอินพุทที่เป็นอนาล็อกกับเอาพุทที่เป็นดิจิตอล

แรงดันอินพุต V_{in} (โวลท์)	เอาต์พุตของวงจรเปรียบเทียบ			เอาต์พุตเลขฐานสอง	
	A1	A2	A3	D1	D0
0 – 1	0	0	0	0	0
1 – 2	1	0	0	0	1
2 – 3	1	1	0	1	0
3 – 4	1	1	1	1	1

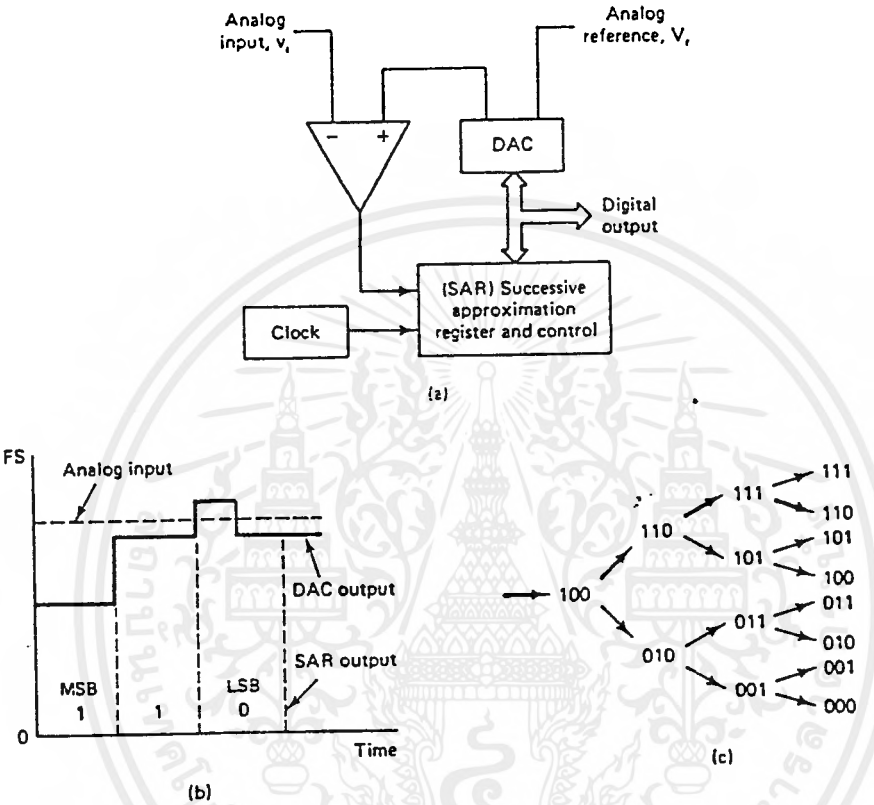
เมื่อต้องการวงจรที่มีความละเอียดสูงขึ้น จำเป็นต้องใช้วงจรเปรียบเทียบมากขึ้น เป็น 2^N-1 เมื่อ N แทนจำนวนบิทหรือความละเอียดที่ต้องการ

ข้อเสียของวงจรแบบนี้ คือจะใช้ตัวเปรียบเทียบมาก เช่น ใช้ 255 ตัว เมื่อต้องการความละเอียด 8 บิท

ข้อดีของวงจรแบบนี้ คือมีความเร็วสูงมาก คือใช้เวลาในการแปลงได้เร็วในระดับนาโนวินาที

2.2.4 A/D Converter แบบใช้การประมาณค่า (Successive Approximation A/D Converter)

ประกอบด้วยวงจร D/A ซึ่งจะแปลง Votlage เปรียบเทียบ มาเปรียบเทียบกับสัญญาณ Input โดย ค่าที่เปรียบเทียบแต่ละครั้งจะเป็น เอาท์พุทของ ค่าดิจิตอลแต่ละบิท จาก บิทสูงไปบิทต่ำ โดยจะมีพัลส์จาก Clock คอยสั่งให้เปรียบเทียบทีละบิท ถ้า A/D มี 8 บิท ก็จะต้องการพัลส์ 8 ลูก เพื่อทำการเปรียบเทียบค่านั้นๆ



รูปที่ 2.13 แสดง Successive Approximation A/D Converter
(a) แสดงบล็อกไดอะแกรม (b) แสดงรูปคลื่นของวงจร (c) แสดง logic ของการทำงาน

จากรูปที่ 2.13 แสดงตัวอย่างการทำงานของ A/D ขนาด 3 บิท โดยที่พัลส์ลูกแรก ค่าจาก D/A จะมีค่าเท่ากับค่ากลางของ Voltage Reference (คือ 100) ซึ่ง น้อยกว่า Input ค่าในรีจิสเตอร์ภายในจึงเป็น "1" เมื่อมีพัลส์ลูกที่ 2 D/A จะเพิ่มค่าขึ้นเป็น 110 ซึ่งน้อยกว่า Input อยู่อีก ค่าบิทที่ 2 จึงเป็น "1" เมื่อมีพัลส์ลูกที่ 3 D/A จะเพิ่มค่าเป็น 111 มากกว่า Input ค่าบิทที่ 3 จึงเป็น "0" สังเกตว่า ความจริงสัญญาณ input จะมีค่ามากกว่า 110 แต่น้อยกว่า 111 ซึ่ง D/A ขนาด 3 บิท มีความละเอียดไม่พอที่จะเปรียบเทียบได้ ต้องทำการเพิ่มขนาดบิทของ D/A จึงจะได้ Output ที่มีค่าใกล้เคียงกับ Input ขึ้น

ข้อดีของวงจรแบบนี้ คือ ความละเอียดของวงจรขึ้นกับบิท ถ้าต้องการให้ เอาท์พุทละเอียดมากก็เพิ่มขนาดของ A/D และเวลาที่ใช้ในการแปลงคงที่ทุกๆค่า

2.3 การแปลงดิจิตอลเป็นอนาล็อก (Digital to Analog Conversion)

2.3.1 แนะนำ

การแปลงดิจิตอลเป็นอนาล็อก จะใช้วงจรหรืออุปกรณ์ที่มีหน้าที่แปลงสัญญาณดิจิตอลซึ่งอาจเป็นแรงดันหรือกระแส ให้เป็นสัญญาณอนาล็อกที่เป็นสัดส่วนกับสัญญาณดิจิตอลที่ป้อนเข้าไปเป็นอินพุทของวงจร เราสามารถเขียนสมการเอาท์พุทของการแปลงดิจิตอลเป็นอนาล็อก ได้ดังนี้

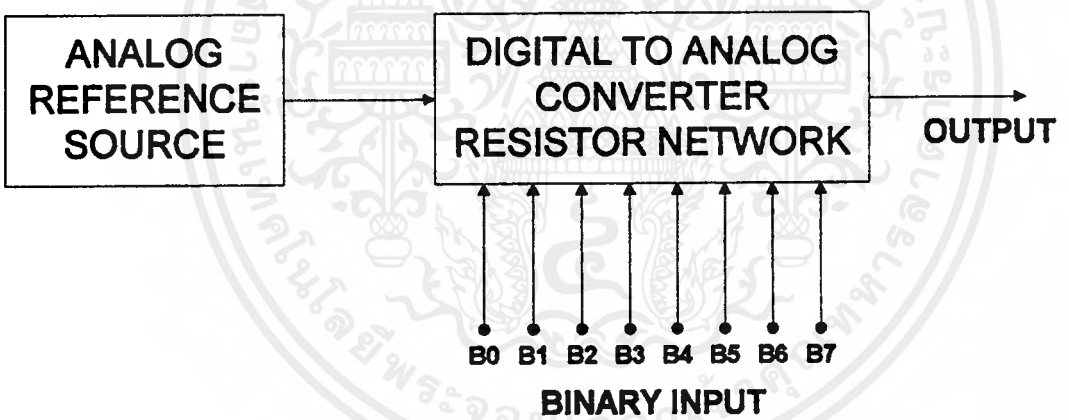
$$X = K \times A \times B$$

โดยที่ X = เป็นค่าแรงดันหรือกระแสทางด้านเอาท์พุท (อนาล็อก)

A = ค่าอ้างอิงอนาล็อก (เป็นแรงดันหรือกระแสก็ได้)

B = จำนวน (ค่า) ของตัวเลข Binary

K = ค่าคงที่จะมีค่าเป็น 1 เสมอ



รูปที่ 2.14 แสดงส่วนประกอบพื้นฐานของการแปลงดิจิตอลเป็นอนาล็อก

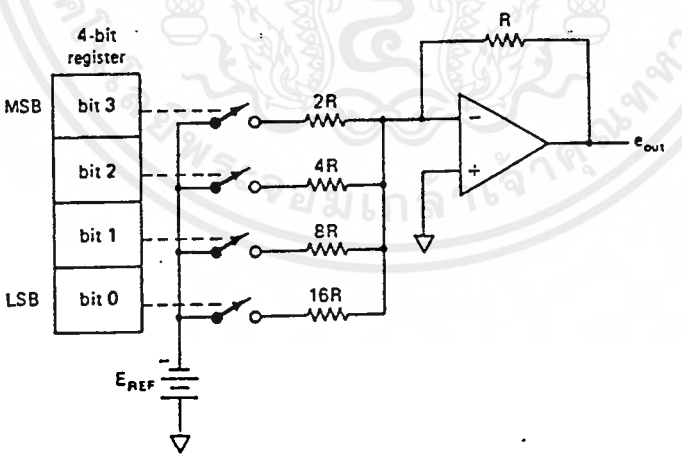
จากรูปที่ 2.14 แสดงส่วนประกอบต่างๆ ของระบบการแปลงดิจิตอลเป็นอนาล็อก โดยที่แหล่งจ่ายอ้างอิงของอนาล็อกจะมีค่าคงที่อยู่ค่าหนึ่ง อาจจะเป็นแหล่งจ่ายแรงดันหรือกระแสที่มีความเที่ยงตรงสูง จุดประสงค์เพื่อใช้เปรียบเทียบกับอินพุท เพื่อที่จะสร้างแรงดันหรือกระแสที่เอาท์พุท

การแปลงดิจิตอลเป็นอนาล็อก จะแบ่งตามการใช้ตัวต้านทาน ซึ่งจะมีการใช้ตัวต้านทานโดยทั่ว ๆ ไปอยู่ 2 ลักษณะคือ แบบใช้ตัวต้านทานแบ่งน้ำหนัก (Binary weighted Resistor Ladder) และแบบใช้ตัวต้านทานขั้นบันได (R-2R Ladder)

2.3.2 การแปลงดิจิตอลเป็นอนาล็อกแบบใช้ตัวต้านทานแบ่งน้ำหนัก (Binary weighted Resistor Ladder)

การใช้ตัวต้านทานแบบนี้ จะเป็นการแปลงข้อมูลดิจิตอลให้เป็นอนาล็อกโดยตรง จะมีตัวต้านทานต่ออนุกรมอยู่กับแรงดันอ้างอิง (V_{ref}) โดยจะมีอิเลคโทรนิคสวิทช์ใช้ในการ เปิด – ปิด สัญญาณ ตามสถานะของสัญญาณดิจิตอลซึ่งมีได้ 2 ระดับ คือ ลอจิก "0" กับ "1" การต่อลักษณะนี้ค่าของตัวต้านทานจะมีค่าในแต่ละน้ำหนักที่คูณด้วย 2 ตลอดคือ จะมีตั้งแต่ R, 2R, 4R,... จนถึง nR หรือเขียนเป็นสมการได้ว่า $2^{(n-1)}R$ โดยที่ n คือจำนวนบิตของเลข Binary ที่จะทำการแปลง

เมื่อตัวต้านทานถูกต่อลง Ground จะไม่มีกระแสไหลผ่านตัวต้านทานที่ต่ออยู่ แต่ถ้าตัวต้านทานตัวนั้นต่ออยู่กับแรงดันอ้างอิง (แทนด้วย ระดับลอจิกที่ตรงข้ามกับการต่อตัวต้านทานลง Ground) จะมีกระแสไหลผ่านตัวต้านทานตัวนั้น ดังนั้นถ้าหากตัวต้านทานค่า R และ 2R ถูกต่อกับแรงดันอ้างอิงพร้อมกัน ก็จะมีกระแสไหลผ่านตัวต้านทาน R มีค่าเท่ากับ V_{ref}/R และกระแสไหลผ่าน 2R มีค่าเท่ากับ $V_{ref}/2R$ ค่ากระแสทั้ง 2 จะไหลมารวมกันที่จุดผสม (Summing Point) ดังรูปที่ 2.15



รูปที่ 2.15 แสดงวงจร D/A Converter แบบ Binary Weighted Resistor Ladder

จากรูปที่ 2.15 สามารถเขียนสมการได้ดังนี้

$$I_{out} = \frac{V_{ref}}{R} \cdot [\sum_{i=1}^{n} \frac{b_i}{2^{(i-1)}}]$$

โดยที่ I_{out} = กระแสเอาท์พุท (สัญญาณอนาล็อก) มีหน่วยเป็น Ampere (A)

b_i = ค่าตัวเลข Binary อินพุท ("0" หรือ "1")

R = ค่าตัวต้านทานตัวแรกที่มีค่าต่ำที่สุด (เป็นค่า R ของ บิทนัยสำคัญสูงสุด (MSB – Most Significant Bit))

จากรูปที่ 2.15 จะเห็นได้ว่าตัวต้านทานที่ทำให้กระแสไหลได้สูงสุด คือ R ดังนั้นตัวต้านทาน R จะเป็นตัวต้านทานของบิทนัยสำคัญสูงสุด (MSB – Most Significant Bit) และจะยังเห็นได้ว่าค่าของตัวต้านทานเพิ่มขึ้นทีละมาก ๆ เช่น ถ้าหากมีการแปลงสัญญาณดิจิตอลขนาด 8 บิท แล้วถ้าใช้ค่า R = 10kΩ แล้วตัวต้านทานตัวที่ 8 จะมีค่าเท่ากับ $2^{(8-1)}R = 128R = 128 \times 10k\Omega = 1280k\Omega$ หรือ $1.28M\Omega$ จะทำให้มีกระแสไหลผ่านตัวต้านทานตัวนี้น้อยมากและจากธรรมชาติที่ว่าตัวต้านทานค่ามากจะสร้างได้ยากและยังมีผลกระทบจากสิ่งแวดล้อมภายนอกอีก คือความร้อนจะทำให้ค่าความต้านทานเปลี่ยนไป (โดยเฉพาะตัวต้านทานที่มีค่ามาก ๆ) ทำให้ความละเอียดและแม่นยำของการแปลง D/A แบบนี้ลดลง ดังนั้นจากข้อจำกัดที่ว่าเมื่อมีจำนวนบิทสูงขึ้นจะทำให้ต้องใช้ตัวต้านทานที่มีค่ามาก ๆ ซึ่งหาได้ยากและสร้างได้ยากรวมถึงยังมีผลกระทบต่อความแม่นยำในการแปลง เพราะฉะนั้นเมื่อมีจำนวนบิทสูงขึ้นจึงได้ทำการเปลี่ยนไปใช้แบบตัวต้านทานขั้นบันได (R-2R Ladder) ซึ่งจะมีการใช้ตัวต้านทานเพียง 2 ค่าเท่านั้น ซึ่งเป็นการขจัดปัญหาที่กล่าวมาข้างต้นได้ ดังจะกล่าวในหัวข้อถัดไป

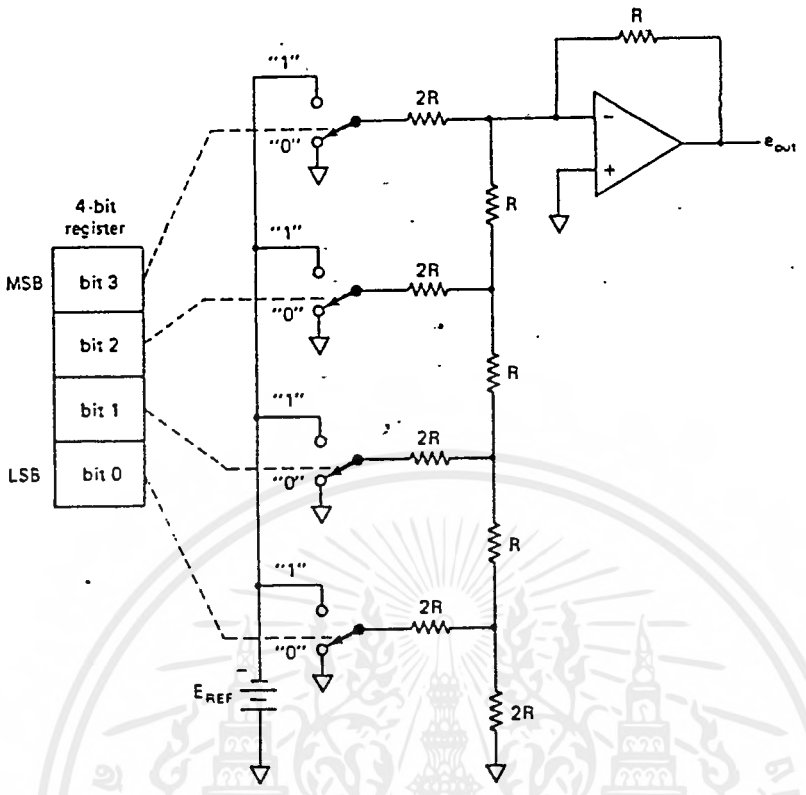
2.3.3 การแปลงดิจิตอลเป็นอนาล็อกแบบตัวต้านทานขั้นบันได (R-2R Ladder D/A Converter)

จะมีรูปวงจรดังแสดงในรูปที่ 2.16 โดยที่การใช้การแปลง D/A แบบ R-2R Ladder นี้ใช้แก้ปัญหาของแบบแรก (Binary weighted Resistor Ladder) ที่ต้องใช้ความต้านทานหลายค่าซึ่งอาจจะหาค่าได้ไม่ตรงกับที่ต้องการ มาใช้ตัวต้านทานเพียง 2 ค่าเท่านั้นในการแปลงแบบนี้ คือค่า R และ 2R

ดังนั้นจากรูปที่ 2.16 วงจรสามารถเขียนเป็นสมการได้ดังนี้คือ

$$I_{out} = \frac{V_{ref}}{R} \cdot [\sum_{i=1}^{n} \frac{b_i}{2^{(i-1)}}]$$

ซึ่งเป็นสมการเดียวกันกับแบบ Binary weighted Resistor Ladder



รูปที่ 2.16 แสดงวงจร D/A converter แบบ R-2R Ladders

2.3.4 คุณสมบัติและข้อกำหนดของตัวแปลง D/A (D/A Characteristics and Specification)

สำหรับตัวแปลงสัญญาณดิจิตอลเป็นอนาล็อกนี้ จะมีคุณสมบัติที่สำคัญดังต่อไปนี้

2.3.4.1. <u>ความละเอียด</u> (Resolution)

คุณสมบัติประการแรก ของ D/A นี้คือความละเอียดของความสามารถในการเปลี่ยนแปลง พิจารณาบิทเลขฐานสองที่ป้อนเข้าที่อินพุทของตัวแปลง D/A ถ้าอินพุทเป็นเลขฐานสอง 8 ตัว จะแสดงว่ามีระดับของเอาท์พุทที่เป็นไปได้เท่ากับ $2^8 = 256$ ดังนั้นค่า Resolution ของ D/A ตัวนี้คือ 1 ใน 256 ในบางครั้งอาจจะกล่าวในรูปของเปอร์เซ็นต์ เช่น Resolution ของ D/A ขนาด 8 บิท คือ ประมาณ 0.39%

2.3.4.2. <u>ค่าเต็มพิกัดของแรงดันเอาท์พุท</u> (Full Scale Output Voltage)

คุณสมบัติประการที่สองของ D/A คือค่าเต็มพิกัดของแรงดันเอาท์พุท คือค่าแรงดันสูงสุดที่ D/A สามารถให้ออกมาได้เมื่ออินพุทดิจิตอลมีค่าลอจิกเป็น "1" หมดทุกบิท

2.3.4.3. ความเที่ยงตรง (Accuracy)

ข้อกำหนดความเที่ยงตรงสำหรับ D/A คือการเปรียบเทียบระหว่าง เอาท์พุทที่เกิดขึ้นจริงกับเอาท์พุทที่คาดหวังไว้ (มาจากการคำนวณ) โดยที่จะมีการกำหนดค่าไว้เป็นเปอร์เซ็นต์ เช่น D/A มีแรงดันเต็มพิกัดของเอาท์พุท 10V และความเที่ยงตรง 0.2% ดังนั้นค่าผิดพลาดสูงสุดของเอาท์พุทใด ๆ เป็น 10V x 0.002 = 20 mV สำหรับ D/A ในอุดมคตินั้นจะมีค่าความผิดพลาดสูงสุดไม่มากกว่า ½ ของบิทนัยสำคัญต่ำสุด (LSB – Least Significant Bit)

2.3.4.4. ความเป็นเชิงเส้น (Linearity)

คือการวัดค่าความเบี่ยงเบนของเอาท์พุทที่เป็นพิสัยจากเส้นตรงที่ตัวแปลงสัญญาณไม่มีการทำงานเลยไปจนถึงทำงานครบทุกบิท ซึ่งค่าเบี่ยงเบนในอุดมคติของเอาท์พุทจากเส้นตรง ควรมีไม่มากกว่า +½ ของค่า LSB

2.3.4.5. ค่าเวลาในการแปลง (Setting Time)

คือเมื่อเกิดการเปลี่ยนแปลงค่า Binary ที่ทำการป้อนเป็นอินพุทเข้ามาในแต่ละบิทของ D/A เอาท์พุทจะเปลี่ยนแปลงเป็นสัญญาณอนาล็อกค่าใหม่ได้ถูกต้อง จะต้องใช้เวลาค่าหนึ่ง ซึ่งขึ้นกับข้อกำหนดของเวลาในการแปลงของตัว D/A ค่าเวลาที่เอาท์พุทใช้ไปภายใน ½ LSB ของค่าสุดท้ายจะเรียกว่า ค่าเวลา "Setting time" สำหรับค่าการเปลี่ยนแปลงเอาท์พุทเต็มพิกัด คุณสมบัตินี้สำคัญเพราะถ้าตัวแปลงสัญญาณทำงานที่ความถี่สูง ๆ มันอาจจะไม่มีเวลาจะตั้งค่าก่อนที่มันจะสวิตช์ไปสู่อีกสถานะหนึ่ง

2.4 การจัดแอดเดรสสำหรับพอร์ท I/O ใน IBM/PC

ในหัวข้อนี้จะกล่าวถึงวิธีการอ้างและใช้งานในแอดเดรสต่าง ๆ ของพอร์ท I/O ที่ใช้งานอยู่ใน IBM/PC

2.4.1 การอ้างแอดเดรสของพอร์ท I/O

ในการควบคุมและตรวจสอบสภาวะการทำงาน รวมทั้งการอ่านข้อมูลจากอุปกรณ์ที่เป็นชิพชัพพอร์ทหรือการ์ดต่าง ๆ ที่มีในระบบ IBM/PC นั้น จะกระทำโดยผ่านทางพอร์ท I/O ของระบบ ดังนั้นในการที่จะใช้งานหรือควบคุมการทำงานของอุปกรณ์เหล่านี้ จึงจำเป็นต้องศึกษาถึงวิธีการควบคุมพอร์ท I/O ต่าง ๆ ของระบบด้วย และเนื่องจากการควบคุมหรือติดต่อกับพอร์ทเหล่านี้ ต้องกระทำโดยการอ้างถึงแอดเดรสของพอร์ท I/O เหล่านั้นโดยตรง เราจึงจำเป็นต้องศึกษาถึงหลักการอ้างแอดเดรสของ 8088 ใน IBM/PC ด้วย

สำหรับแอดเดรสของพอร์ท I/O ต่าง ๆ นั้น จะเป็นแอดเดรสที่ถูกสร้างขึ้นโดย 8088 ซึ่งแอดเดรสเหล่านี้เป็นแอดเดรสที่จัดไว้สำหรับพอร์ท I/O เหล่านั้นโดยเฉพาะ คือแยกจากแอดเดรสของหน่วยความจำโดยเด็ดขาด ส่วนการส่งข้อมูลให้กับพอร์ทเหล่านี้ จะทำได้โดยการใช้คำสั่ง OUT

ของ 8088 ส่งข้อมูลนั้นไปยังแอดเดรสของพอร์ทที่ต้องการ และสำหรับการตรวจสอบหรือการอ่านข้อมูลจากพอร์ท ก็จะทำได้โดยการใช้คำสั่ง IN ของ 8088 อ่านข้อมูลจาดแอดเดรสของพอร์ทที่ต้องการ เช่นกัน

ภายในไมโครโปรเซสเซอร์เบอร์ 8088 นี้จะมีแอดเดรสสำหรับใช้กับพอร์ท I/O อยู่ทั้งสิ้น 65,536 หรือ 64K แอดเดรส (ในขณะที่มีแอดเดรสสำหรับหน่วยความจำอยู่ 1 Mbyte) ซึ่งทำให้การอ้างแอดเดรสของพอร์ท I/O ที่ทำงานร่วมกับ 8088 นั้น ต้องใช้จำนวนเส้นแอดเดรสในบัสแอดเดรสทั้งสิ้น 16 เส้น คือ A0 - A15 แต่สำหรับใน IBM/PC นี้ถูกออกแบบมาให้ใช้เส้นแอดเดรสเฉพาะ 10 เส้นล่าง คือ A0 - A9 เท่านั้น ดังนั้นในการอ้างแอดเดรสของพอร์ทของอุปกรณ์หรือชิพซัพพอร์ทใดๆ ที่ใช้ร่วมกับ IBM/PC จึงใช้จำนวนเส้นแอดเดรสเพียง 10 เส้นด้วย โดยเส้นแอดเดรสที่เหลือ คือ A10 - A15 นั้น จะไม่ถูกนำไปใช้งาน แต่ค่าแอดเดรสบนเส้นแอดเดรสเหล่านี้ยังคงเปลี่ยนแปลงตามค่าแอดเดรสของพอร์ทที่กำหนดไว้ในคำสั่ง OUT หรือ IN อยู่ด้วย เพียงแต่ไม่ได้ถูกนำมาดีโค้ดร่วมกับแอดเดรส A0 - A9 เท่านั้น ตัวอย่างเช่นในการใช้คำสั่ง OUT ส่งข้อมูลไปยังพอร์ทที่ตรงกับแอดเดรส 0010H นั้น จะให้ผลเหมือนกับการส่งข้อมูลไปยังพอร์ทที่ตรงกับแอดเดรส 0410H, 0810, 0C10H, ทั้งนี้เนื่องจากแอดเดรส 6 บิทบนไม่ได้ถูกใช้งานจึงทำให้การเปลี่ยนแปลงค่าแอดเดรสบนเส้นแอดเดรส A10 - A15 นั้นไม่ทำให้เกิดความแตกต่างใด ๆ ขึ้น

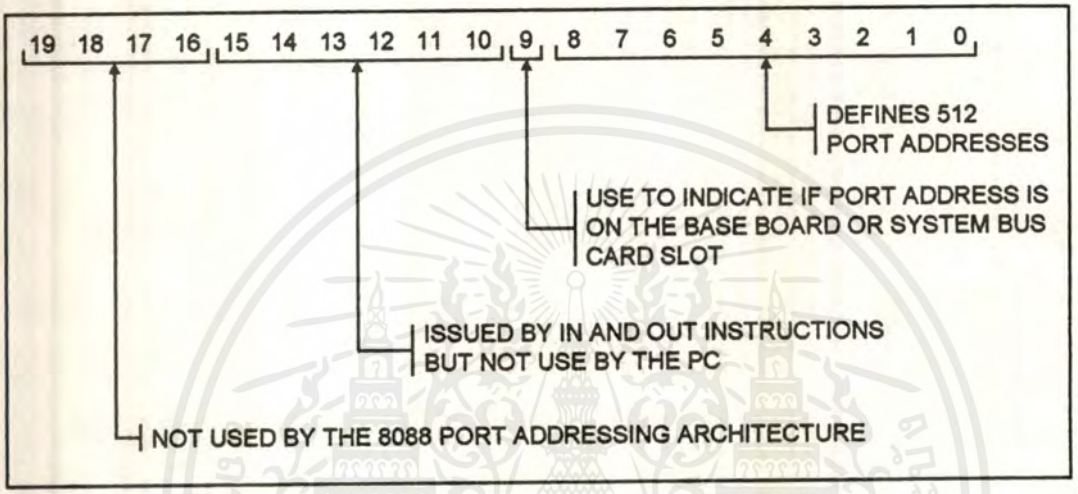
เนื่องจากใน IBM/PC ได้ใช้งานเส้นแอดเดรสเพียง 10 เส้น (คือ A0 - A9) ดังนั้นจึงสามารถที่จะอ้างแอดเดรสของพอร์ทได้สูงสุดเพียง 1024 พอร์ท (จากจำนวน 64K พอร์ท) เท่านั้น นอกจากนี้ในกรณีที่เป็นการอ่านข้อมูลจากพอร์ทของ IBM/PC ข้อมูลบิท A9 จะถูกจัดให้มีหน้าที่ในการแบ่งพอร์ททั้ง 1024 พอร์ทออกเป็น 2 ส่วน (ส่วนละ 512 พอร์ท) อีกด้วย กล่าวคือถ้าข้อมูลในบิท A9 เป็น "0" แล้ว เราจะทำการอ่านข้อมูลได้เฉพาะจากพอร์ทอุปกรณ์ หรือชิพซัพพอร์ท ต่าง ๆ ที่อยู่บนเมนบอร์ด (Main Board) ของ IBM/PC เช่น 8253 - 5, 8237 - 5 หรือ 8259A เท่านั้น แต่ถ้าข้อมูลในบิท A9 นี้เป็น "1" ก็จะทำการอ่านข้อมูลได้เฉพาะจากพอร์ทที่อยู่บนการ์ดต่าง ๆ เท่านั้น

จากที่ได้กล่าวมานั้น จะสรุปได้ว่าพอร์ทบน IBM/PC ทั้ง 1024 พอร์ท ถูกแบ่งออกเป็น 2 กลุ่ม โดยที่กลุ่มแรกเป็นกลุ่มของพอร์ทที่อยู่บนเมนบอร์ด และกลุ่มที่สองเป็นกลุ่มที่จัดเตรียมไว้สำหรับพอร์ทที่อยู่บนการ์ดต่าง ๆ

สำหรับในกรณีของการส่งข้อมูลให้กับพอร์ททั้ง 1024 พอร์ท เราสามารถที่จะเลือกส่งไปยังพอร์ทใด ๆ ใน IBM/PC ได้ ดังนั้นการเลือกแอดเดรสสำหรับพอร์ทที่อยู่บนการ์ดจึงสามารถทำได้โดยสะดวก แต่อย่างไรก็ตามสิ่งหนึ่งที่จะต้องคำนึงถึงก็คือ ถ้าแอดเดรสที่เลือกให้กับพอร์ทนี้ ตรงกับค่าแอดเดรสเดิมที่มีอยู่บนเมนบอร์ดแล้ว เมื่อเราทำการส่งข้อมูลให้กับพอร์ทที่อยู่ในตำแหน่งแอดเดรสนี้ก็จะเท่ากับเป็นการส่งข้อมูลให้กับทั้งพอร์ทที่อยู่บนเมนบอร์ด และพอร์ทที่อยู่บนการ์ด

ด้วย ซึ่งในกรณีเช่นนี้อาจก่อให้เกิดความผิดพลาดขึ้นได้เช่นกัน ดังนั้นในการกำหนดค่าแอดเดรสให้กับพอร์ทที่ถูกสร้างขึ้นบนการ์ดต่าง ๆ จึงควรใช้ค่าแอดเดรสที่แอดเดรสบิท A9 มีค่าเป็น "1" คือ แอดเดรส 0FE00H จนถึง 0FFFFH เท่านั้น (แอดเดรสบิท A10 - A15 ไม่ถูกใช้ในการดีโค้ด แต่เพื่อความสะดวกจึงกำหนดให้มีค่าเป็น "1" ในฐานสองทั้งหมด แต่ในการใช้งานจริงอาจเปลี่ยนให้แอดเดรส A10 - A15 แต่ละบิทมีค่าเป็น "1" หรือ "0" ก็ได้)

สำหรับรูปที่ 2.17 นี้ จะแสดงถึงการใช้งานแอดเดรสบิทต่าง ๆ ในการอ้างแอดเดรสของพอร์ทใน IBM/PC



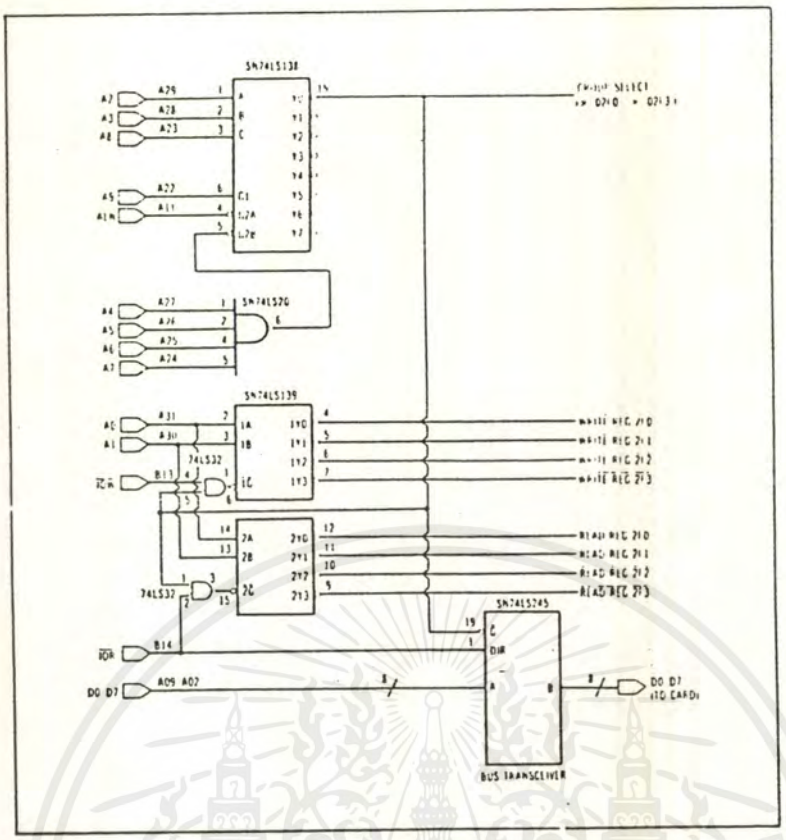
รูปที่ 2.17 การใช้แอดเดรสบิทต่าง ๆ ในการอ้างแอดเดรสของพอร์ทใน IBM/PC

2.4.2 เทคนิคในการดีโค้ดแอดเดรสสำหรับพอร์ท I/O

ในหัวข้อต่าง ๆ ที่ผ่านมาข้างต้นนั้น ได้กล่าวถึงการอ้างแอดเดรสต่าง ๆ ของพอร์ท I/O ใน IBM/PC สำหรับในหัวข้อนี้จะกล่าวถึงวิธีต่าง ๆ ที่ใช้ในการดีโค้ดแอดเดรสต่าง ๆ ให้เป็นไปตามที่เราต้องการ

2.4.2.1 การดีโค้ดแบบ Fixed

วิธีการดีโค้ดแบบนี้เป็นวิธีที่ง่ายและสะดวกในการดีโค้ดแอดเดรส หรือกลุ่มของแอดเดรสของพอร์ท I/O ซึ่งวิธีนี้เป็นการกำหนดจำนวนของแอดเดรสที่เราต้องการใช้ จากนั้นจึงทำการเลือกบล็อกของแอดเดรสที่ยังไม่ถูกใช้งานโดยการ์ดหรือวงจรอินเทอร์เฟสอื่น ๆ (บล็อกของแอดเดรสที่เลือกต้องมีจำนวนแอดเดรสเพียงพอกับจำนวนแอดเดรสที่เราต้องการใช้งาน) แล้วจึงออกแบบวงจรที่ทำการดีโค้ดแอดเดรสที่เราต้องการ สำหรับตัวอย่างวงจรที่ใช้ในการดีโค้ดแอดเดรสในแบบนี้จะแสดงดังรูปที่ 2.18



รูปที่ 2.18 ตัวอย่างวงจรดีโค้ดแอดเดรสแบบ Fixed

จากรูปจะเห็นได้ว่า วงจรที่ใช้นี้เป็นวงจรที่สามารถทำการดีโค้ดแอดเดรสได้ 8 กลุ่ม โดยแต่ละกลุ่มจะมีจำนวนแอดเดรส 4 แอดเดรส ซึ่งแอดเดรสทั้ง 8 กลุ่มจะแสดงได้ดังตารางข้างล่าง

กลุ่ม	แอดเดรส
0 (Y0)	02F0H - 02F3H
1 (Y1)	02F4H - 02F7H
2 (Y2)	02F8H - 02FBH
3 (Y3)	02FCH - 02FFH
4 (Y4)	03F0H - 03F3H
5 (Y5)	03F4H - 03F7H
6 (Y6)	03F8H - 03FBH
7 (Y7)	03FCH - 03FFH

สำหรับในตัวอย่างนี้จะเลือกใช้การดีโค้ดแอดเดรสในกลุ่ม 0 (เริ่มจากแอดเดรส 02F0H จนถึง 02F3H) คือ ใช้สัญญาณเอาท์พุท (สัญญาณ $\overline{\text{GROUPSELECT}}$) จากขา Y0 (ขา 15 ของ 74LS138 ไปทำการ OR กับสัญญาณ $\overline{\text{IOR}}$ และ $\overline{\text{IOW}}$ เพื่อสร้างเป็นสัญญาณอีนาเบิลวงจรดีโค้ด

(74LS139) แอดเดรสอีด 4 แอดเดรส ซึ่งแบ่งเป็น 2 ชุด คือชุดที่เป็น $\overline{\text{WRITE REG}}$ ซึ่งจะแอคทีฟ (ลอจิก "0") เมื่อ CPU ต้องการจะส่งข้อมูลให้กับวงจรภายนอก (สัญญาณ $\overline{\text{IOW}}$ แอคทีฟ) และชุดที่เป็น $\overline{\text{READ REG}}$ ซึ่งจะแอคทีฟเมื่อ CPU ต้องการจะอ่านข้อมูลจากวงจรภายนอก (สัญญาณ $\overline{\text{IOR}}$ แอคทีฟ) สัญญาณ $\overline{\text{WRITE REG}}$ และ $\overline{\text{READ REG}}$ นี้ โดยทั่วไปจะนำไปเป็นสัญญาณสโตรป (Strobe) ให้กับวงจรภายนอกที่เกี่ยวข้อง เพื่อให้สามารถส่งหรือรับข้อมูลจาก CPU ได้ในช่วงเวลาที่เหมาะสม นอกจากนี้สัญญาณ $\overline{\text{GROUPSELECT}}$ ยังถูกนำไปใช้ในการอีนาเบิลบัฟเฟอร์ 74LS245 ด้วย เพื่อให้ CPU สามารถส่งหรือรับข้อมูลจากภายนอกได้เมื่อแอดเดรสในกลุ่มนี้ถูกเลือก สำหรับทิศทางของข้อมูลจะถูกควบคุมโดยสัญญาณ $\overline{\text{IOR}}$ ส่วนสัญญาณ AEN จะถูกนำมามาใช้ในการดิสเอเบิลวงจรดีโค้ด โดยถ้าสัญญาณ AEN เป็น "1" ซึ่งเป็นช่วงเวลาของขบวนการ DMA นั้น 74LS138 จะถูกดิสเอเบิลทันที ทั้งนี้ก็เพื่อป้องกันความผิดพลาดที่อาจเกิดขึ้น เนื่องจากการดีโค้ดแอดเดรสของพอร์ทในระหว่างขบวนการ DMA นั่นเอง (ในระหว่างนี้แอดเดรสบนบัสแอดเดรสจะเป็นแอดเดรสของหน่วยความจำคือ สัญญาณ $\overline{\text{MEMW}}$ หรือ $\overline{\text{MEMR}}$ จะแอคทีฟ แต่ในขณะเดียวกันสัญญาณ $\overline{\text{IOR}}$ หรือ $\overline{\text{IOW}}$ ก็จะแอคทีฟด้วย ดังนั้นถ้าไม่ดิสเอเบิลวงจรดีโค้ดไว้แล้ว อาจจะทำให้วงจรดีโค้ดคิดว่าแอดเดรสบนบัสแอดเดรสเป็นแอดเดรสของพอร์ท I/O ก็ได้)

ในการดีโค้ดแอดเดรสของพอร์ท I/O เราจะต้องคำนึงถึงช่วงเวลาของสัญญาณที่เกิดขึ้นในกระบวนการอ่าน หรือเขียนข้อมูลลงบนพอร์ท I/O ดังนี้

1. ในช่วงเริ่มต้นของบัสไซเคิลเกี่ยวกับพอร์ท I/O นั้น ถ้าสัญญาณจากวงจรดีโค้ดมีการหน่วงเวลา (Delay) มากเกินไป อาจจะทำให้สัญญาณดีโค้ดนี้เกิดขึ้นหลังจากที่สัญญาณ $\overline{\text{IOR}}$ หรือ $\overline{\text{IOW}}$ แอคทีฟ แลพเนื่องจากค่าแอดเดรสบนบัสแอดเดรสนั้นเปลี่ยนแปลงได้ตลอดเวลา ดังนั้นก่อนที่ค่าแอดเดรสที่ถูกต้องจะถูกส่งออกมาบนบัสแอดเดรสนั้น วงจรดีโค้ดจะได้รับค่าแอดเดรสอื่น ๆ อยู่ ซึ่งถ้าหากวงจรดีโค้ดมีการหน่วงเวลามากเกินไปแล้ว สัญญาณดีโค้ดแอดเดรสที่ไม่ถูกต้องนี้อาจจะถูกหน่วงเวลาจนเกิดขึ้นในช่วงเวลาที่สัญญาณ $\overline{\text{IOR}}$ หรือ $\overline{\text{IOW}}$ เกิดขึ้นแล้วก็ได้ ทำให้ข้อมูลบนบัสข้อมูลนั้นถูกส่งไปยังพอร์ทไม่ถูกต้อง สำหรับใน IBM/PC จะถูกออกแบบให้การหน่วงเวลาในวงจรดีโค้ดนั้นมีค่าไม่เกิน 92 nanosec.

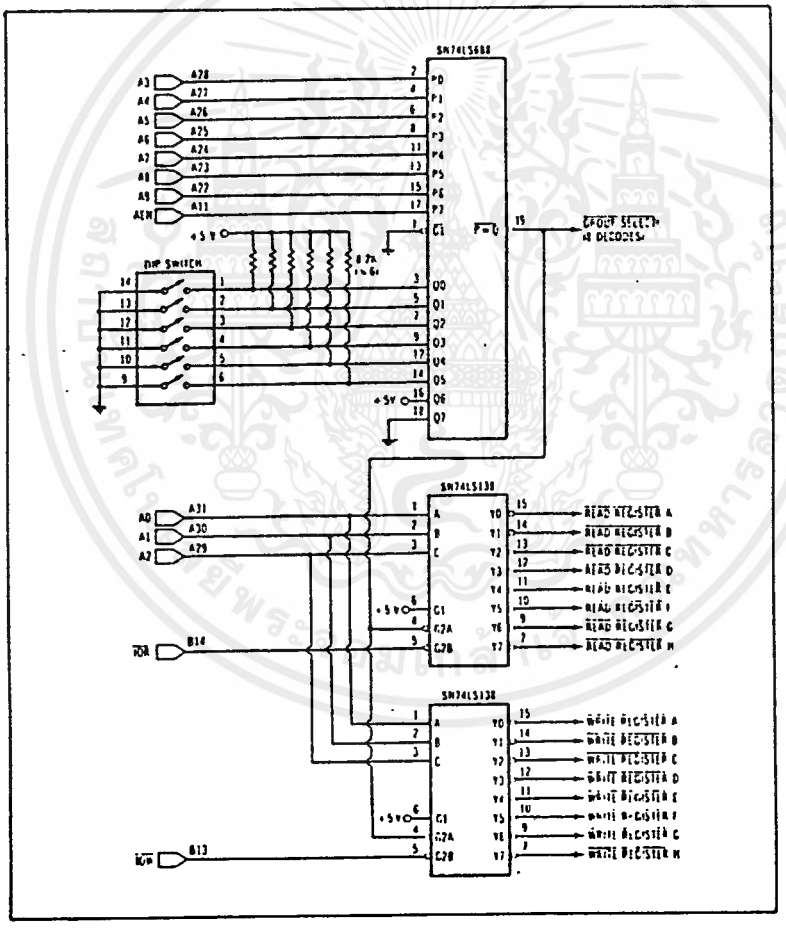
2. ในช่วงท้ายของบัสไซเคิลในการเขียนข้อมูลลงบนพอร์ท I/O นั้น ถ้าสัญญาณ $\overline{\text{IOW}}$ มีการหน่วงเวลาออกไป และวงจรดีโค้ดมีความเร็วในการทำงานสูงแล้ว อาจจะทำให้ข้อมูลในบัสไซเคิลนี้ถูกส่งให้กับพอร์ท I/O ที่มีแอดเดรสตรงกับค่าแอดเดรสในบัสไซเคิลต่อไปก็ได้สำหรับใน IBM/PC สัญญาณ $\overline{\text{IOW}}$ จะมีการหน่วงเวลาไปไม่เกิน 200 nanosec.

อย่างไรก็ตามช่วงเวลาที่ต้องสนใจมากอีกช่วงเวลาหนึ่งก็คือ ช่วงเวลาระหว่างขอบขาขึ้นของสัญญาณ $\overline{\text{IOW}}$ กับช่วงเวลาที่ข้อมูลที่ถูกต้องถูกส่งออกมาบนบัสข้อมูล ถ้าสัญญาณ $\overline{\text{IOW}}$ ถูกหน่วงเวลาไปเกินกว่า 120 nanosec. แล้ว อาจจะทำให้พอร์ท I/O ได้รับข้อมูลที่ไม่ถูกต้องก็ได้ และ

สำหรับสัญญาณ $\overline{IOR}$ นั้น ถ้ามีการหน่วงเวลาเกิดขึ้นแล้ว ก็จะทำให้ความเร็วในการอ่านข้อมูลถูกลดลง

2.4.2.2 การดีโค้ดโดยใช้สวิตช์เลือก

การดีโค้ดในแบบ Fixed ที่ได้กล่าวไว้ในหัวข้อที่กล่าวมานั้น มีข้อเสียอยู่บางประการ คือ แอดเดรสที่เราเลือกใช้งานไว้นั้น อาจซ้ำกับแอดเดรสของการ์ดอื่นที่เรานำมาเพิ่มเข้าไปในระบบในภายหลังก็ได้ ซึ่งกรณีเช่นนี้เราต้องแก้ไขวงจร เพื่อหลีกเลี่ยงไปใช้แอดเดรสอื่นที่ยังว่างอยู่ และไม่ถูกใช้งานโดยการ์ดที่จะเพิ่มเข้าไปใหม่ ซึ่งยุ่งยากและต้องเสียเวลามากขึ้น ปัญหาเช่นนี้เราสามารถแก้ไขได้โดยใช้วงจรดีโค้ดที่สามารถเปลี่ยนแปลงค่าแอดเดรสได้ โดยเพียงแต่เปลี่ยนตำแหน่งของสวิตช์ (ในที่นี้คือ DIP Switch) ที่เซ็ทไว้ในวงจรเท่านั้น ดังรูปที่ 2.19



รูปที่ 2.19 ตัวอย่างวงจรดีโค้ดโดยใช้สวิตช์เลือก

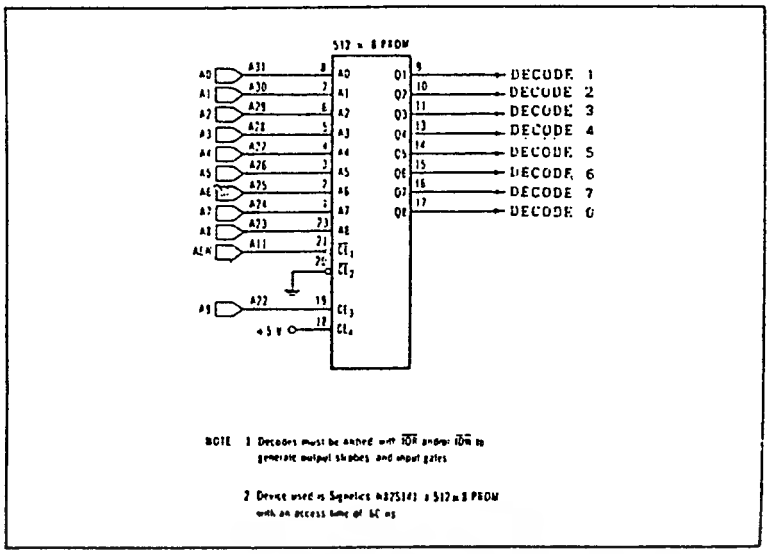
จากรูปเป็นวงจรที่ทำการดีโค้ดกลุ่มแอดเดรสขนาด 8 แอดเดรส ซึ่งการเลือกกลุ่มแอดเดรสที่จะทำการดีโค้ดนี้ จะทำได้โดยการเซ็ท DIP Switch ที่ขา Q0 - Q5 ของ 74LS688

สำหรับหน้าที่ของ 74LS688 นี้ จะทำการเปรียบเทียบค่าอินพุท 2 ชุด ที่ถูกส่งเข้ามาทางขา P0 - P7 และขา Q0 -Q7 ถ้าอินพุททั้ง 2 ชุดนี้เท่ากันแล้ว เอาท์พุทที่ขา $\overline{P=Q}$ จะให้เอาท์พุทเป็นลอจิก "0" จากในวงจรขา P0 - P6 ของ 74LS688 ต่อกับแอดเดรสบิท A3 - A9 ในขณะที่ขา Q0 - Q5 ต่อกับความต้านทางที่หน้าที่เป็น Pull Up (รักษาระดับแรงดันให้เป็นลอจิก "1" ไว้ในกรณีที่ไม่มีอินพุทใด ๆ เข้ามา) และขา Q0 - Q5 นี้จะต่อกับปลายข้างหนึ่งของ DIP Switch ด้วย ส่วนปลายอีกข้างหนึ่งของ DIP Switch นั้นจะต่อลง Ground (ลอจิก "0") ไว้ ดังนั้นถ้าเราทำการ "ON" DIP Switch ที่ต่อกับขาใดขานั้น ก็จะได้รับลอจิก "0" ในขณะที่ถ้า DIP Switch ที่ต่อกับขาใดถูก "OFF" ขานั้นก็จะได้รับลอจิก "1" และเนื่องจากอินพุทที่ขา P0 - P5 นั้น ต้องเปลี่ยนแปลงตามไปด้วย จึงจะทำให้เอาท์พุทของ 74LS688 แอคทีฟได้ ทำให้เราสามารถเปลี่ยนค่าแอดเดรสที่ต้องการจะดีโค้ดได้ง่ายกว่าวิธีดีโค้ดแบบ Fixed สำหรับขา Q6 นั้น จะต่อกับลอจิก "1" (+5 V) และขา P6 ต่อกับแอดเดรสบิท A9 ในกรณีเช่นนี้จึงเท่ากับเป็นการบังคับให้แอดเดรสที่จะทำการดีโค้ดได้นั้น จะต้องมีแอดเดรสบิท A9 เป็น "1" เท่านั้น ส่วนขา P7 จะต่อกับสัญญาณ AEN โดยมีขา Q7 ต่อกับลอจิก "0" การต่อในลักษณะนี้ก็เพื่อป้องกันไม่ให้ 74LS688 ทำการดีโค้ดในระหว่างกระบวนการ DMA นั่นเอง เอาท์พุทจากขา $\overline{P=Q}$ ของ 74LS688 นี้จะถูกนำไปใช้ในการอีนาเบิล 74LS138 ทำหน้าที่ในการดีโค้ดแอดเดรสของกลุ่มแอดเดรสที่เราเลือก (โดยใช้ DIP Switch ดังที่ได้กล่าวในตอนต้น)

วงจรในลักษณะนี้เราสามารถจะนำไปใช้เป็นการดีโค้ดในแบบ Fixed ได้โดยการนำเอา DIP Switch ออก จากนั้นถ้าอินพุทใดต้องการลอจิก "0" จึงจะใช้ตัวนำเชื่อมต่อระหว่างขั้วทั้งสองแทนการเซ็ท DIP Switch ให้ "ON" แต่ถ้าอินพุทใดต้องการลอจิก "1" ก็ปล่อยขั้วทั้งสองนั้นไว้

2.4.2.3 การดีโค้ดโดยใช้ PROM

การดีโค้ดในแบบต่าง ๆ ที่กล่าวมาแล้วนั้น เป็นการดีโค้ดในลักษณะที่แอดเดรสของพอร์ทต่าง ๆ อยู่รวมกันเป็นกลุ่ม แต่ในบางกรณีพอร์ทที่เราใช้งานนั้นมีแอดเดรสแยกกันอย่างเป็นอิสระ เช่นในการนำเอาหน้าที่การทำงานอยู่บนการ์ดต่าง ๆ มารวมไว้บนการ์ดเพียงการ์ดเดียว และมีความจำเป็นต้องคงค่าแอดเดรสของพอร์ทเดิม (ที่อยู่บนการ์ดเดิม) ไว้ด้วย ทำให้ไม่สามารถใช้การดีโค้ดในแบบต่าง ๆ ที่ผ่านมาได้ เนื่องจากการใช้วิธีการดีโค้ดในแบบที่ผ่านมานั้นจะทำให้ต้องใช้อุปกรณ์ที่ทำการดีโค้ดนั้นมากเกินไป ในกรณีเช่นนี้เราจำเป็นต้องใช้การดีโค้ดอีกแบบหนึ่ง ซึ่งจะได้กล่าวถึงในหัวข้อนี้ คือการดีโค้ดโดยใช้ PROM (Programmable Read Only Memory) ดังในรูปที่ 2.20



รูปที่ 2.20 ตัวอย่างวงจรดีโค้ดโดยใช้ PROM

จากรูปข้างต้นเป็นวิธีการง่าย ๆ แบบหนึ่งในการดีโค้ดโดยใช้ PROM ซึ่งจะเห็นได้ว่าเราใช้เส้นแอดเดรส A0 - A8 ของระบบต่อเข้ากับเส้นแอดเดรส A0 - A8 ของ PROM และใช้บัสข้อมูลทั้ง 8 ของ PROM คือ Q1 - Q8 เป็นเอาท์พุท สำหรับใช้เป็นสัญญาณดีโค้ดให้กับพอร์ทต่าง ๆ 8 พอร์ท อย่างไรก็ตามสัญญาณดีโค้ดทั้ง 8 เส้น คือ DECODE1 - DECODE8 นี้ ยังคงต้องนำไป OR กับสัญญาณ $\overline{IOR}$ หรือ $\overline{IOW}$ ก่อนที่จะนำไปอีนาเบิลพอร์ทที่มีแอดเดรสตรงกับแอดเดรสที่ป้อนให้กับ PROM นั้น

จากที่ได้กล่าวมานั้นจะเห็นได้ว่าส่วนของวงจรดีโค้ดนั้น จะมี PROM เพียงตัวเดียวเท่านั้น ซึ่ง PROM ที่จะนำมาใช้งานนี้จะต้องถูกโปรแกรมมาก่อนแล้ว โดยข้อมูลที่โปรแกรมให้กับแอดเดรสต่าง ๆ ของ PROM นั้น จะต้องสัมพันธ์กับสัญญาณดีโค้ดที่เราต้องการ กล่าวคือเราจะต้องทราบเสียก่อนว่าค่าแอดเดรสของพอร์ททั้ง 8 ที่เราต้องการจะดีโค้ดนั้นมีแอดเดรสใดบ้าง แล้วจึงกำหนดว่าพอร์ทใดจะใช้สัญญาณดีโค้ดเส้นใด จากนั้นจึงโปรแกรมข้อมูลให้กับ PROM โดยแอดเดรสใดถ้าต้องการให้สัญญาณดีโค้ดใดแอคทีฟ (ในที่นี้จะกำหนดให้สัญญาณดีโค้ดแอคทีฟที่ลอจิก "0") ก็กำหนดให้ข้อมูลในบิทที่ตรงกับสัญญาณดีโค้ดนั้นเป็น "0" เช่นถ้าเรากำหนดให้แอดเดรสของพอร์ท ที่เราต้องการจะดีโค้ดเป็น 0393H และเลือกใช้สัญญาณ DECODE 5 เราก็จะต้องทำการโปรแกรม ให้แอดเดรส 0193H ของ PROM (เหตุที่แอดเดรสของ PROM เป็น 0193H แทนที่จะเป็นแอดเดรส 0393H เหมือนกับแอดเดรสของพอร์ท ก็เพราะแอดเดรสของ PROM มีเพียง 9 บิท คือ A0 - A8 เท่านั้น ส่วนบิท A9 จะถูกต่อเข้ากับ PROM ภายหลังเพื่ออีนาเบิล PROM เมื่อข้อมูลในบิท A9 นี้เป็น "1" เท่านั้น) มีข้อมูลในบิท Q5 (ถ้าเริ่มนับจากบิท D0 ก็คือบิท D4) เป็น "0" ส่วนบิทอื่น ๆ นั้นมีค่าเป็น "1" ทั้งหมด ดังนั้นการโปรแกรมแอดเดรส 0193H ของ PROM จึงต้องโปรแกรม

ด้วย ข้อมูล 0EFH เป็นต้นสำหรับข้อมูลในแอดเดรสอื่น ๆ ที่นอกเหนือจากแอดเดรสทั้ง 8 ที่กำหนดแล้ว จะต้องโปรแกรมให้ข้อมูลทุกบิทเป็น "1" ทั้งหมด ซึ่งก็คือโปรแกรมด้วยข้อมูล 0FFH นั่นเอง

ตัวอย่างเช่น

ถ้าแอดเดรสของพอร์ททั้ง 8 ที่เราต้องการจะดีโค้ดเป็น 024A, 02B5, 0317, 0361, 0382, 03A8, 03C4 และ 03DB ในฐานสิบหกตามลำดับ โดยกำหนดให้สัญญาณจากการดีโค้ดแอดเดรสเหล่านี้เป็นสัญญาณ DECODE1 จนถึง DECODE8 ตามลำดับแล้ว (เช่นสัญญาณจากการดีโค้ด 024AH ก็คือสัญญาณ DECODE1 และสัญญาณจากการดีโค้ด 02B5H ก็คือสัญญาณ DECODE2 เป็นต้น) เราจะต้องทำการโปรแกรม PROM ให้มีข้อมูลที่สัมพันธ์กับเอาท์พุทที่เราต้องการ ดังนี้

1. แอดเดรสของพอร์ทเป็นแอดเดรสที่บิท A9 ถูกใช้งานร่วมด้วย โดยในบิทนี้ต้องมีข้อมูลเป็น "1" ในขณะที่แอดเดรสของ PROM จะมีเพียง 9 บิทคือ A0 - A8 เท่านั้น เราจึงจัดแอดเดรสของ PROM เมื่อเทียบกับพอร์ทดังนี้

แอดเดรสของพอร์ท (บิท A9 ถูกใช้งาน)	แอดเดรสของ PROM (เฉพาะบิท A0 - A8)
024A	04A
02B5	0B5
0317	117
0361	161
0382	182
03A8	1A8
03C4	1C4
03D4	1DB

2. ข้อมูลที่จะโปรแกรมให้กับแอดเดรสทั้ง 8 ของ PROM จะต้องสัมพันธ์กับเอาท์พุทที่ต้องการ เช่นถ้ามีการอ้างถึงแอดเดรสของพอร์ท 02B5H แล้ว PROM จะต้องให้เอาท์พุทที่มีลอจิก "0" ที่ขา Q2 (DECODE2) ส่วนเอาท์พุทที่ขาอื่นต้องเป็น "1" ดังนั้นจึงต้องมีการโปรแกรมให้แอดเดรส 00B5H ของ PROM มีข้อมูลเป็น 1111 1101 (ฐานสอง) หรือ 0FDH เป็นต้น สำหรับแอดเดรสอื่น ๆ นอกเหนือจากแอดเดรสทั้ง 8 นี้แล้วจะต้องถูกโปรแกรมให้มีข้อมูลเป็น 0FFH (ฐานสิบหก) ทั้งหมด ดังนี้

แอดเดรส PROM (ฐานสิบหก)	ข้อมูล (ฐานสิบหก)
000 - 049	0FF
04A	0FE
04B - 0B4	0FF
0B5	0FD
0B6 - 116	0FF
117	0FB
118 - 160	0FF
161	0F7
162 - 181	0FF
182	0EF
183 - 1A7	0FF
1A8	0DF
1A9 - 1C3	0FF
1C4	0BF
1C5 - 1DA	0FF
1DB	07F
1DC - 1FF	0FF

อย่าไรก็ตามสิ่งสำคัญอีกสิ่งหนึ่งที่ต้องคำนึงถึงเสมอเมื่อใช้วิธีดีโค้ดแบบนี้ก็คือ PROM ที่ใช้นั้นจะต้องใช้เวลาทำงานน้อยกว่า 92 nanosec. ด้วย

บทที่ 3
การออกแบบ และ วงจรโดยรวม

3.1 วงจร Interface กับเครื่องคอมพิวเตอร์

3.1.1 หลักการติดต่อพอร์ท I/O ของ IBM PC

จากทฤษฎีในบทที่ 2 ได้กล่าวถึงการติดต่อพอร์ท I/O ของ IBM PC แล้วครั้งหนึ่งแต่เพื่อความเข้าใจมากขึ้นจึงขอกล่าวเพิ่มเติมในบทนี้ดังนี้ ในการที่จะทำการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอก (I/O Device) นั้นเครื่องคอมพิวเตอร์จะทำโดยการติดต่อผ่านทางพอร์ท I/O ซึ่งพอร์ท I/O นั้นจะมีแอดเดรสโดยเฉพาะ คือแยกจากแอดเดรสของหน่วยความจำโดยเด็ดขาด ภายในเครื่องคอมพิวเตอร์นั้นจะมีแอดเดรสใช้สำหรับพอร์ท I/O อยู่ทั้งหมด 64K (65536) แอดเดรส โดยที่มีแอดเดรสบัสที่ใช้ติดต่อทั้งหมด 16 เส้น แต่ในเครื่อง IBM PC นั้นจะถูกออกแบบมาให้ใช้แอดเดรสบัสได้เฉพาะ 10 เส้นล่างเท่านั้น คือ A0 – A9 (A10 – A15 นั้นจะไม่ถูกนำไปใช้งาน) ดังนั้นจึงสามารถอ้างแอดเดรสได้สูงสุดเพียง 1024 แอดเดรส จาก 64K ซึ่งพอร์ททั้ง 1024 พอร์ทนี้จะถูกแบ่งออกเป็น 2 ส่วน ส่วนละ 512 พอร์ท ดังที่จะสามารถแสดงพอร์ททั้ง 1024 ได้ดังตารางที่ 3.1

ตารางที่ 3.1 ตารางแสดงแอดเดรสพอร์ท I/O ของ IBM PC

Hex range	Usage	
000-00F	DMA chip 8237A-5	
020-021	Interrupt 8259A	
040-043	Timer 8253-5	
060-063	PPI 8255A-5	Assigned to
080-083	DMA page registers	system board
0Ax	NMI mark register	Components
0Cx	Reserved	
0Ex	Reserved	
100-1FF	Not usable	
200-20F	Game control	
210-217	Expansion unit	
220-24F	Reserved	
278-27F	Reserved	
2F0-2F7	Reserved	
2F8-2FF	Asynchronous communications (2)	
300-31F	Prototype card *	
320-32F	Fixed disk	
378-37F	Printer	Assigned to
380-38C	SDLC communications	feature card
380-389	Binary synchronous communications (2)	Ports
3A0-3A9	Binary synchronous communications (1)	
3B0-3BF	IBM monochrome display/printer	
3C0-3CF	Reserved	
3D0-3DF	Color/graphics	
3E0-3EF	Reserved	
3F0-3F7	Diskette	
3F8-3FF	Asynchronous communications (1)	

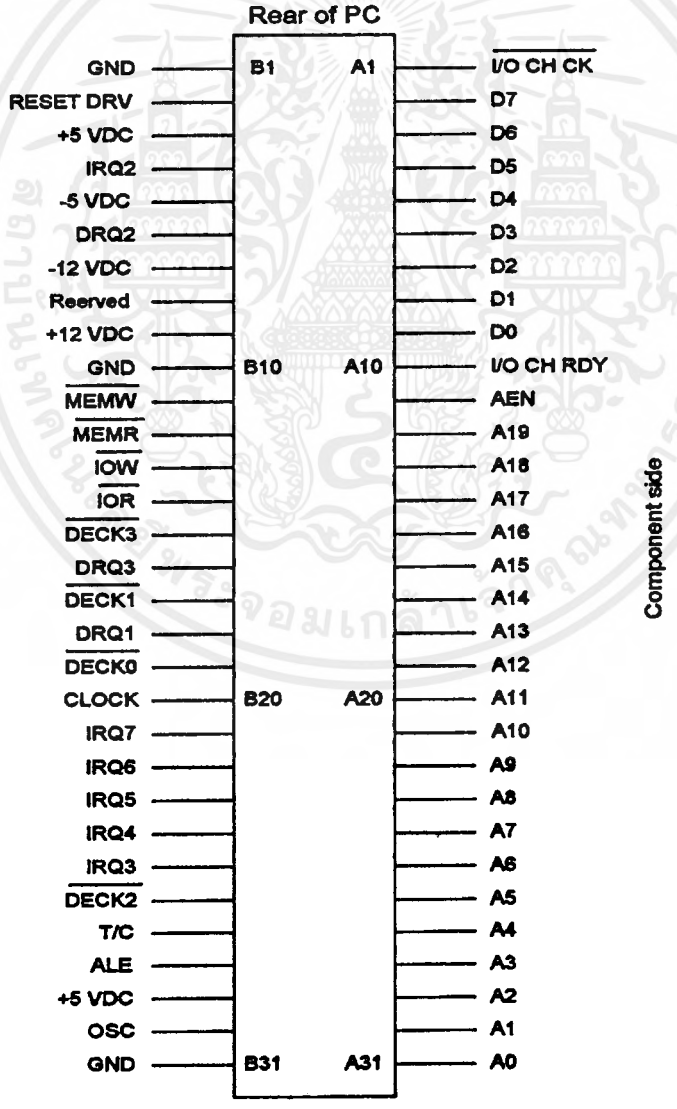
จากตารางที่ 3.1 นั้นจะเห็นได้ว่าพอร์ท 1024 พอร์ทถูกแบ่งเป็น 2 ส่วน นั่นคือ

◆ ส่วนที่ 1 แอดเดรสมีค่า 000H – 1FFH เป็นส่วนของพอร์ทที่อยู่บน Main Board

◆ ส่วนที่ 2 แอดเดรสมีค่า 200H – 3FFH เป็นส่วนของพอร์ทที่จะใช้งานอยู่บน Card ที่ใช้เสียบใน Slot ต่าง ๆ ของ IBM PC

ซึ่งทั้ง 2 ส่วนนี้จะถูกกำหนดการแบ่งด้วยข้อมูลในบิท A9 คือถ้า A9 มีข้อมูลเป็น "0" จะทำการติดต่อกับพอร์ทในส่วนที่ 1 (มีแอดเดรส 000H – 1FFH) เท่านั้น (ติดต่อกับ Main Board เท่านั้น) แต่ถ้า A9 มีข้อมูลเป็น "1" ก็จะทำการติดต่อกับพอร์ทในส่วนที่ 2 (มีแอดเดรส 200H – 3FFH) เท่านั้น (ติดต่อกับ Card ที่เสียบอยู่ใน Slot เท่านั้น)

สำหรับในโครงงานนี้จะทำการเลือกใช้แอดเดรสพอร์ทช่วง 300H – 31FH ซึ่งจากตารางที่ 3.1 เป็นช่วงของ Prototype Card มาใช้เป็นแอดเดรสสำหรับที่จะใช้ในการอ้างถึง Interface Card ต่อไป และส่วนของ Slot ที่จะใช้สำหรับต่อกับ Card ต่าง ๆ นั้นสามารถแสดงได้ดังรูปที่ 3.1



รูปที่ 3.1 IBM PC System Bus (Slot ของ IBM PC)

จากรูปที่ 3.1 จะขออธิบายส่วนประกอบของสล็อตเสริม (Expansion slots) เฉพาะส่วนที่นำไปใช้งานเท่านั้น ว่าแต่ละขามีหน้าที่อย่างไรบ้าง ดังจะอธิบายได้ดังนี้

◆ RESET DRV (ขา B2)

ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งจะแอคทีฟ (ลอจิก "1") ในช่วงที่เราเริ่มจ่ายไฟให้กับระบบ และจะยังคงแอคทีฟไปจนกว่าระบบต่างๆภายใน IBM/PC จะพร้อมที่จะทำงานได้ จากนั้นสัญญาณนี้ก็จะเปลี่ยนกลับเป็นลอจิก "0" นอกจากนี้ในระหว่างการทำงานของ IBM/PC ถ้าระดับของแรงดันแหล่งจ่ายไฟตกลง สัญญาณนี้ก็จะถูกทำให้แอคทีฟเช่นกัน

◆ A0 – A19 (Address Bus; ขา A31 – A12)

สัญญาณทั้ง 20 ขานี้เป็นเอาท์พุท ซึ่งใช้สำหรับกำหนดแอดเดรสของหน่วยความจำหรืออุปกรณ์ I/O ที่ 8088 ต้องการติดต่อด้วย โดยที่สัญญาณ A0 จะมีนัยสำคัญต่ำสุด (LSB - Least Significant Bit) และ A19 จะมีนัยสำคัญสูงสุด (MSB - Most Significant Bit) สำหรับค่าแอดเดรสบนบัสแอดเดรส A0 – A19 นี้ จะถูกกำหนดโดย 8088 ในระหว่างขบวนการอ่าน/เขียนข้อมูลลงในหน่วยความจำหรืออุปกรณ์ I/O แต่ในช่วงของขบวนการ DMA นั้น DMA-Controller จะเป็นผู้กำหนดค่าแอดเดรสบนบัสแอดเดรสเอง (ในระหว่างนี้ 8088 จะถูกตัดออกจากระบบ)

◆ D0 – D7 (Data Bus; ขา A9 – A2)

ขาสัญญาณนี้จะเป็นแบบ Bi-Directional ซึ่งต่อกับบัสข้อมูลของระบบ เพื่อทำหน้าที่ในการส่งผ่านข้อมูลระหว่างพอร์ท I/O กับ IBM/PC โดยบิท D0 จะมีนัยสำคัญต่ำสุด (LSB) และบิท D7 จะมีนัยสำคัญสูงสุด (MSB)

◆ $\overline{\text{IOR}}$ (I/O Read; ขา B14)

ขาสัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ที่สร้างขึ้นโดย 8288 Bus Controller เพื่อใช้ในการแสดงว่าบัสที่เกิดขึ้นนี้ เป็นบัสไซเคิลของการอ่านข้อมูลจากพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้นส่งข้อมูลออกมาบนบัสข้อมูล โดยข้อมูลจะต้องถูกส่งออกมาบนบัสข้อมูลก่อนขอบขาขึ้นของสัญญาณ IOR ประมาณ 30 ns เพื่อให้มั่นใจได้ว่า 8088 สามารถรับข้อมูลได้ถูกต้อง

◆ $\overline{\text{IOW}}$ (I/O Write; ขา B13)

ขาสัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ซึ่งถูกสร้างขึ้นโดย 8288 Bus Controller เพื่อใช้แสดงว่าบัสไซเคิลที่เกิดขึ้นนี้เป็นบัสไซเคิลของการเขียนข้อมูลลงบนพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้น รับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้

◆ AEN (Address Enable; ขา A11)

สัญญาณนี้เป็นเอาท์พุทที่ใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นในช่วงเวลาที่สัญญาณ AEN แอคทีฟ (ลอจิก "1") นั้น เป็นบัสไซเคิลของขบวนการ DMA

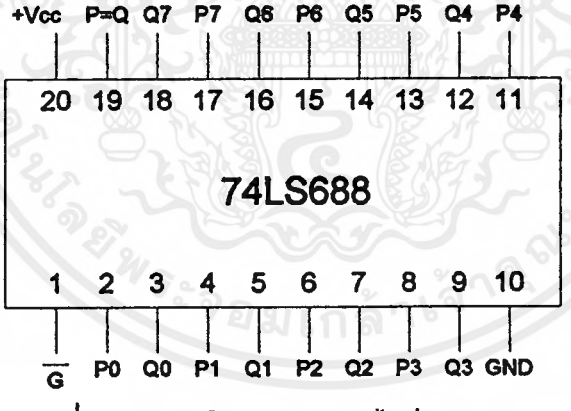
◆ **บัสของแหล่งจ่ายไฟของระบบ**

แหล่งจ่ายไฟของระบบจะประกอบด้วยแหล่งจ่ายไฟ DC +5V,+12V,-5V,-12V ซึ่งประกอบไปด้วยขา B3, B29, B9, B5 และ B7 ส่วนสัญญาณ GND จะประกอบไปด้วยขา B1, B10และ B31 ซึ่งขาทั้งสามนี้จะต่อกับกราวด์ (Ground) ของระบบ

3.1.2 เทคนิคการดีโค้ดแอดเดรสสำหรับพอร์ท I/O

จากทฤษฎีในบทที่ 2 นั้นได้กล่าวเทคนิคการดีโค้ดแอดเดรสสำหรับพอร์ท I/O ไว้หลายแบบด้วยกันได้แก่การดีโค้ดแบบ Fixed, แบบใช้สวิตช์เลือก และแบบใช้ PROM จะขอกล่าวถึงเฉพาะหลักการที่นำมาใช้ในการออกแบบในปริญญานิพนธ์นี้อีกครั้งเพื่อที่จะได้เกิดความเข้าใจมากขึ้นนั่นคือการดีโค้ดแบบใช้สวิตช์เลือก

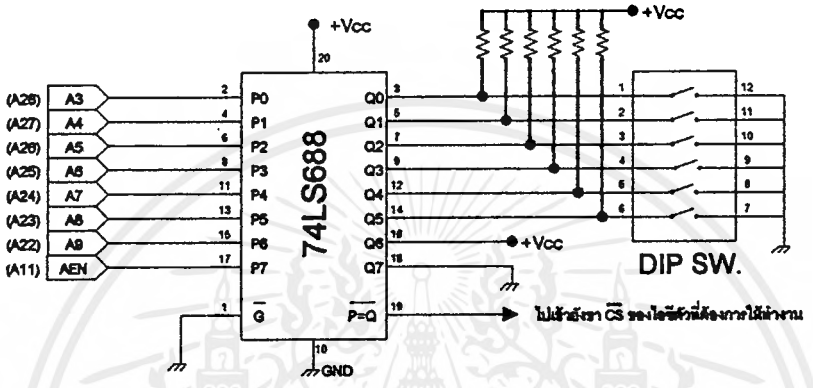
เทคนิคการดีโค้ดใช้สวิตช์เลือก จะเป็นการดีโค้ดแบบ Fixed ซึ่งอาจจะทำให้ได้แอดเดรสซ้ำกับ Card อื่นที่นำมาเสียบไว้ในระบบจะสามารถแก้ไขได้โดยการสับ DIP SW. เปลี่ยนแอดเดรสไปใช้ในตำแหน่งที่ว่างอยู่ ซึ่งในโครงงานนี้ตำแหน่งแอดเดรสของพอร์ทที่จะเลือกใช้จะอยู่ที่ 300H – 31FH การทำการดีโค้ดแบบนี้นั้นจะใช้ไอซีเปรียบเทียบระหว่าง 2 อินพุท 2 ชุดที่ถูกส่งเข้ามาโดยใช้ไอซี 74LS688 (8 bit comparator) ซึ่งเป็น Exclusive-OR ดังที่จะสามารถแสดงตำแหน่งของขาต่าง ๆ ได้ดังรูปที่ 3.2



รูปที่ 3.2 แสดงตำแหน่งขาของไอซี 74LS688

มีหลักการทำงานดังนี้ พิจารณารูปที่ 3.3 ไอซี 74LS688 จะทำการเปรียบเทียบค่าอินพุท 2 ชุดที่ถูกส่งเข้ามาทางขา P0 – P7 และขา Q0 – Q7 ถ้าอินพุท 2 ชุดนี้เท่ากันแล้วเอาท์พุทที่ขา P=Q จะมีค่าเป็น logic "0" จากวงจรขา P0 – P6 ต่อกับแอดเดรสบิท A3 – A9 ในขณะที่ขา Q0 – Q5 ต่อกับ R Pull up และต่ออยู่กับ DIP SW. ส่วนปลายอีกข้างหนึ่งของ DIP SW. ต่อลง GND (logic "0") ดังนั้นถ้า ON SW. ตัวใดขานั้นก็จะรับ logic "0" ในทางตรงกันข้ามถ้า OFF SW ตัวใดขานั้นก็จะรับ logic "1" และเนื่องจากอินพุทที่ขา P0 – P5 (A3 – A9) ต้องมีค่าเท่ากับอินพุทที่ขา Q0 – Q5 ดังนั้นถ้าเปลี่ยนแปลงค่าที่ Set ไว้ที่ DIP SW. ก็จะทำให้แอดเดรส A3 – A5 จะต้องเปลี่ยนแปลง

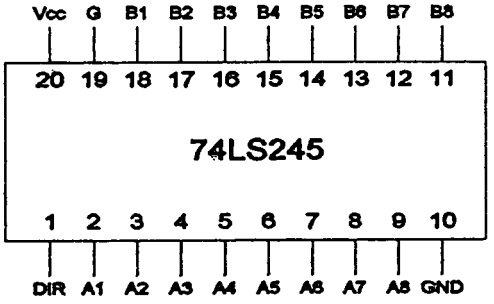
ตามไปด้วยเพื่อให้เอาท์พุทของ 74LS688 ทำงานได้ สำหรับขา Q6 จะต่อกับ logic "1" และขา P6 จะต่ออยู่กับแอดเดรสบิท A9 ซึ่งจะเป็นการบังคับให้ A9 เป็น "1" เท่านั้น และขา P7 จะต่ออยู่กับ AEN โดยที่ขา Q7 ต่ออยู่กับ logic "0" เพื่อที่จะเป็นการป้องกันไม่ให้เกิดการดีโค้ดในระหว่างขบวนการ DMA จากนั้นจะนำขาเอาท์พุทของ 74LS688 (ขา $\overline{P=Q}$) ไปต่อเข้ากับสัญญาณ Chip Select ของอุปกรณ์ที่ต้องการให้ทำงานที่แอดเดรสนี้ ส่วนในการที่จะเปลี่ยนค่าแอดเดรสนั้นสามารถทำได้ง่าย โดยการเปลี่ยนตำแหน่งของสวิตช์ (DIP SW.)



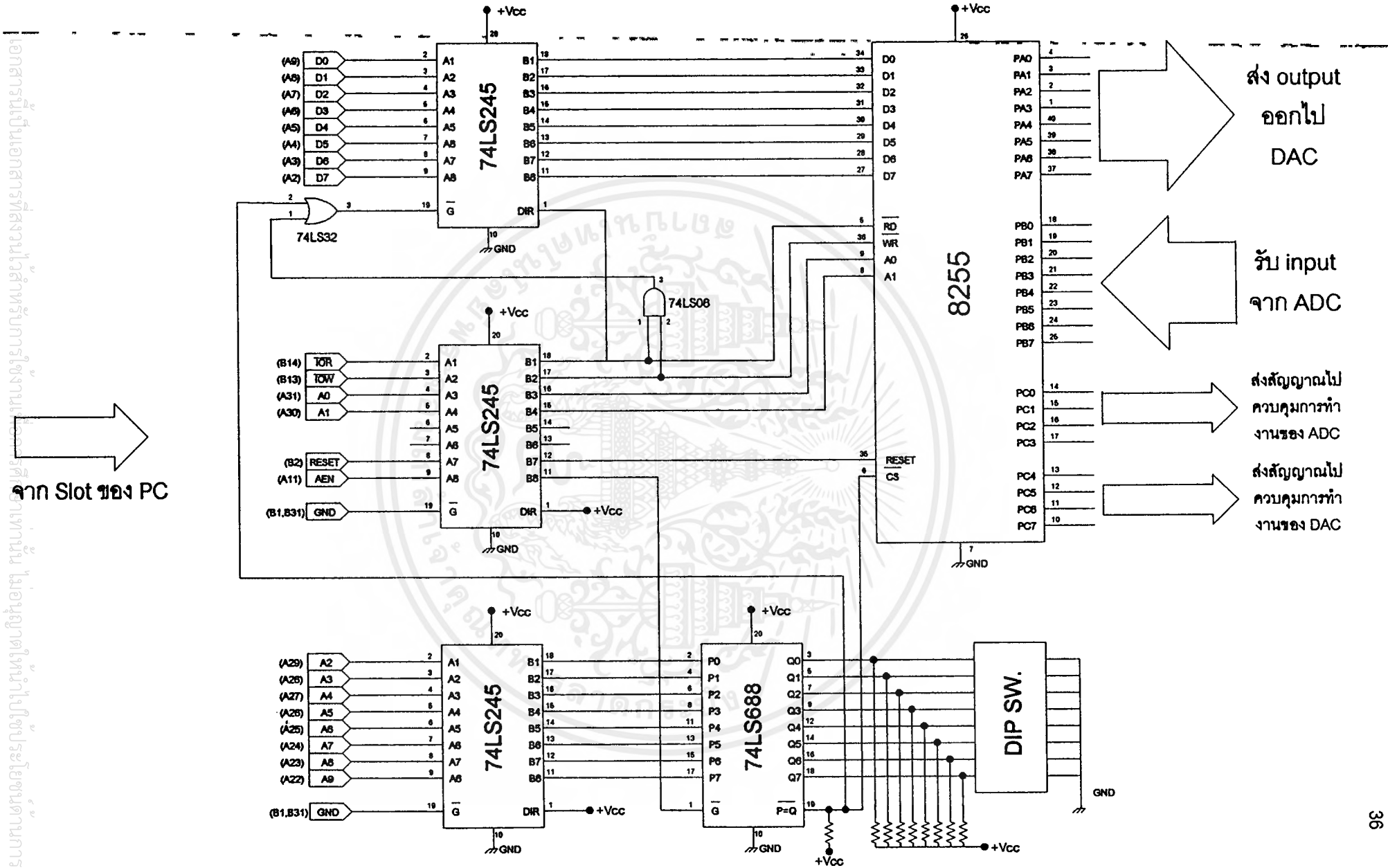
รูปที่ 3.3 วงจรดีโค้ดโดยใช้สวิตช์เลือก

3.1.3 INTERFACE CARD

การที่จะต่อ 8255 เข้ากับคอมพิวเตอร์นั้นต้องใช้วงจรชุดดีโค้ดซึ่งในที่นี้เลือกแบบใช้สวิตช์เลือก โดยมีส่วนประกอบคือ DIP SW. และไอซี 74LS688 โดยมีหลักการทำงานตามที่ได้กล่าวมาแล้วในหัวข้อ 3.1.2 ขาที่เอาท์พุทของ 74LS688 จะถูกต่อเข้ากับขา CS ของ 8255 นอกจากนี้ 74LS688 ยังถูกควบคุมการทำงานโดยคอมพิวเตอร์ โดยจะนำสัญญาณ AEN มาต่อที่ขา 1 ของ 74LS688 ซึ่งจะ active low ซึ่งรูปวงจรรวมของ INTERFACE CARD จะถูกแสดงไว้ในรูปที่ 3.4 โดยภายในวงจรจะมีไอซีเพิ่มขึ้นมา 1 ตัวนั่นคือ 74LS245 (Buffer 2 ทิศทาง) ซึ่งจะสามารถแสดงรูปของ 74LS245 ได้ดังรูปที่ 3.5



รูปที่ 3.5 แสดงตำแหน่งขาของไอซี 74LS245



รูปที่ 3.4 แสดงการต่อวงจรพอร์ท 8255 ของ Interface card

3.1.4 วงจรการทำงานของ Interface card

ภายในการ์ดจะประกอบด้วยวงจร 2 ส่วน คือ

1. วงจรดีโค้ด

2. ส่วนของ CHIP SUPPORT

วงจรดีโค้ด

จะใช้ไอซี เบอร์ 74LS688 จะทำการดีโค้ดสัญญาณแอดเดรสจาก Slot ของคอมพิวเตอร์มาเพื่อใช้เลือกแอดเดรสให้กับ 8255 สัญญาณที่จะนำมาใช้ในการดีโค้ดมี A0-A9 แต่ 8255 ต้องการสัญญาณ จาก A0 และ A1 ไปควบคุมรีจิสเตอร์ภายในตัวมัน ดังนั้นสัญญาณที่สามารถนำมาใช้ได้จึงมีแค่ A2 - A9 ท่านั้น ส่วน A0 - A1 นั้นจะถูกต่อเข้ากับขา A0 และ A1 ของ 8255 โดยตรง

จากรวงจรในรูปที่ 3.4 จะสังเกตว่าจะนำแต่สัญญาณ A0 - A9, AEN, $\overline{IOR}$, $\overline{IOW}$, RESET มาใช้เท่านั้น ส่วนสัญญาณอื่นๆไม่นำมาใช้

การทำงานของ 74LS688 ดังที่ได้กล่าวมาแล้วนั้น ไอซีนี้จะทำการเปรียบเทียบสัญญาณอินพุท 2 ส่วน คือ P0 - P7 และ Q0 - Q7 จะให้เอาท์พุทออกมาเป็น logic "0" เมื่อ $\overline{P=Q}$ สัญญาณ Q จะถูกต่ออยู่กับ DIP SW. ในการเลือกแอดเดรสของการ์ดเลือกโดยตั้งสวิตช์ ซึ่งในโครงงานนี้จะใช้แอดเดรสช่วง 300H - 31FH เมื่อมีแอดเดรสค่า 300H มาเข้าที่อินพุทของ 74LS688 ที่ขาเอาท์พุทจะให้เอาท์พุทเป็น "0" ทำให้ $\overline{CS}$ ของ 8255 ทำงาน 8255 จะต่อ Data Bus ของตัวมันเองเข้ากับ Data Bus ของ คอมพิวเตอร์ สำหรับแอดเดรสที่ใช้มี 4 ตำแหน่ง คือ

300H ควบคุมการทำงานของพอร์ต A

301H ควบคุมการทำงานของพอร์ต B

302H ควบคุมการทำงานของพอร์ต C

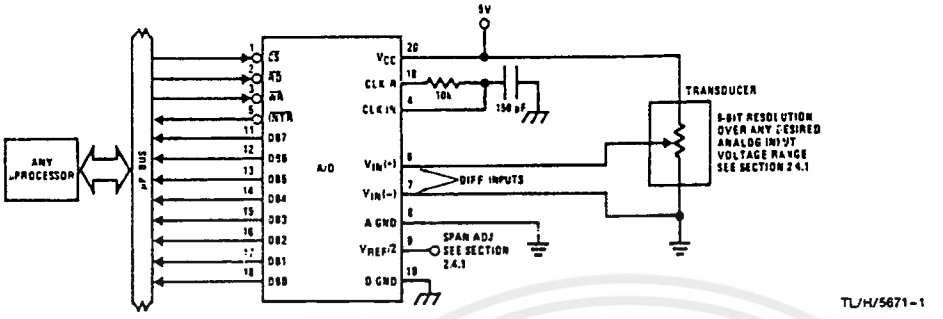
303H ส่ง Control Word ให้กับ 8255

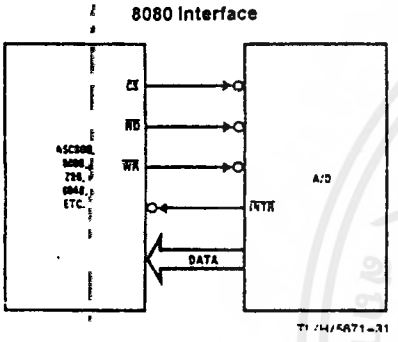
การใช้งาน 8255 Interface card จะต้องเริ่มต้นที่ตั้งแอดเดรสของ Card ก่อนโดยที่จะทำการตั้งที่ DIP SW. ซึ่งการตั้งแอดเดรสนั้น ต้องตั้งบิท A8 และ A9 เป็น "1" เพื่อให้ได้ค่าเป็น 300H สำหรับ A0, A1 จะนำไปใช้ในการ Set Control Word ให้กับ 8255 โดยที่จะใช้โปรแกรมเป็นตัวตั้งค่า สำหรับค่าแอดเดรส A2 – A9 จะทำการตั้งดังต่อไปนี้

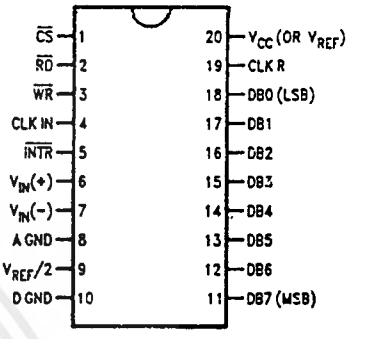
A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	0	0	0	0	0	0	X	X

3.2 การเลือกไอซี A/D มาใช้งานในวงจร (เลือก ADC0805) และวงจรที่ใช้งาน

ADC0805 เป็นไอซีวงจร A/D 8 บิท แบบ Successive Approximation มีวงจรแลตช์ภายในซึ่งเหมาะสำหรับต่อเข้ากับ Data Bus ของไมโครโปรเซสเซอร์ มี Conversion Time = 110 µs







Error Specification (Includes Full-Scale, Zero Error, and Non-Linearity)			
Part Number	Full-Scale Adjusted	$V_{REF}/2 = 2.500\ V_{DC}$ (No Adjustments)	$V_{REF}/2$ = No Connection (No Adjustments)
ADC0801	±¼ LSB		
ADC0802		±½ LSB	
ADC0803	±½ LSB		
ADC0804		±1 LSB	
ADC0805			±1 LSB

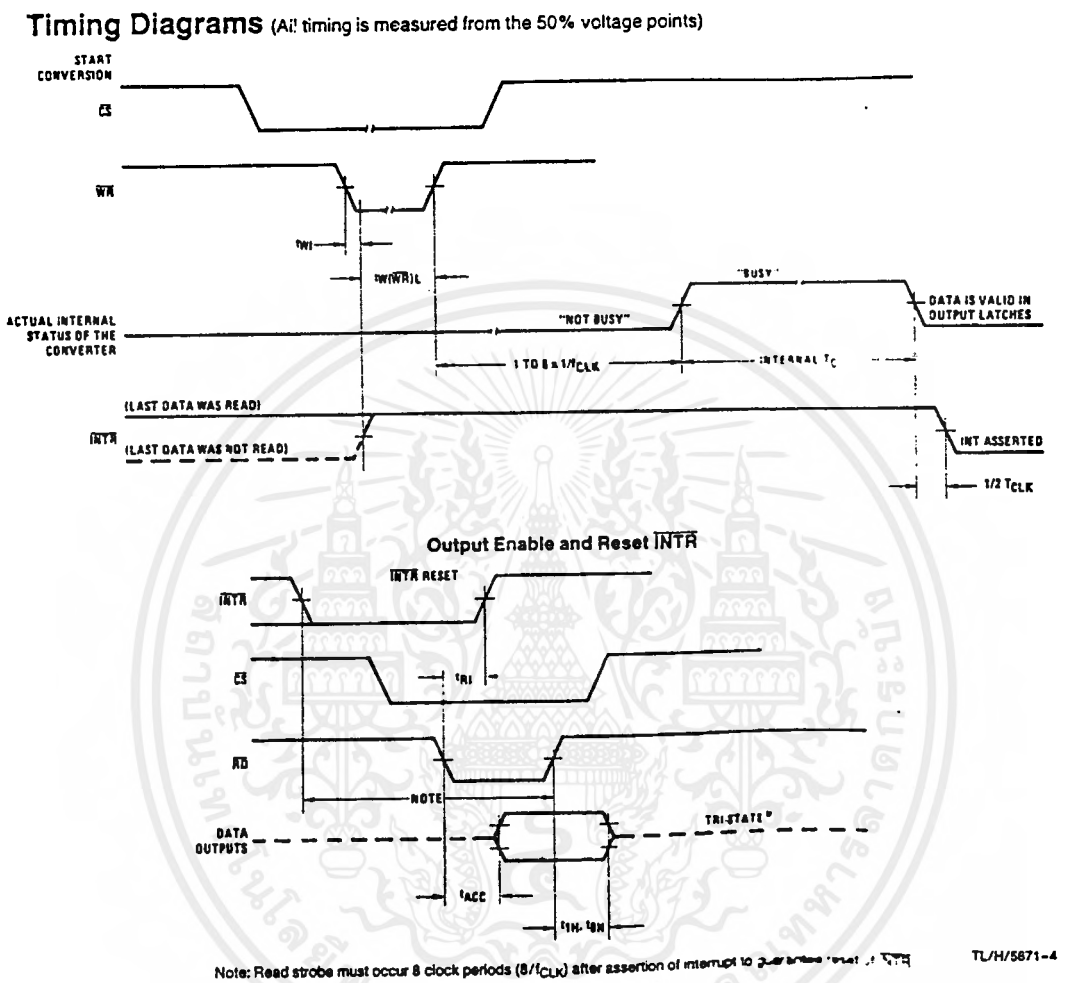
รูปที่ 3.6 แสดงลักษณะการต่อ ADC0805 เข้ากับไมโครโปรเซสเซอร์

ความหมายของขาต่างๆ ของ ADC0805

- $\overline{CS}$ Chip Select (Active Low) ไมโครโปรเซสเซอร์จะต้องให้ค่า "0" กับขานี้ เมื่อต้องการติดต่อกับ ADC0805
- $\overline{WR}$ WRITE (Active Low) ถ้าสัญญาณเป็น "0" ADC0805 จะทำการ Sampling ค่ามาเก็บไว้
- $\overline{RD}$ READ (Active Low) ถ้าสัญญาณเป็น "0" ADC0805 จะส่งค่าที่ทำการแปลงเป็น Digital แล้วให้กับไมโครโปรเซสเซอร์
- V_{CC} Digital Supply Voltage เป็นขาที่ต่อกับแหล่งจ่ายไฟให้กับ A/D ซึ่งมีขนาด 5V
- $V_{Ref/2}$ ต่อกับค่า $V_{Ref/2}$ ซึ่ง A/D จะใช้เพื่อนำไปเปรียบเทียบกับ input แต่สำหรับ ไอซี ADC0805 ไม่จำเป็นต้องต่อขา $V_{Ref/2}$ นี้
- DB0-DB7 เป็นเอาท์พุทพอร์ทของข้อมูล ต่อเข้ากับ DATA BUS ของไมโครโปรเซสเซอร์ โดย DB0 เป็น LSB (Least Significant Bit) และ DB7 เป็น MSB (Most Significant Bit)
- A GND และ D GND เป็น Ground ของวงจร

- $V_{in}(+)$ และ $V_{in}(-)$ เป็นขา Input
- CLK R และ CLK IN ต่อกับวงจรให้กำเนิดสัญญาณ Clock ของ A/D

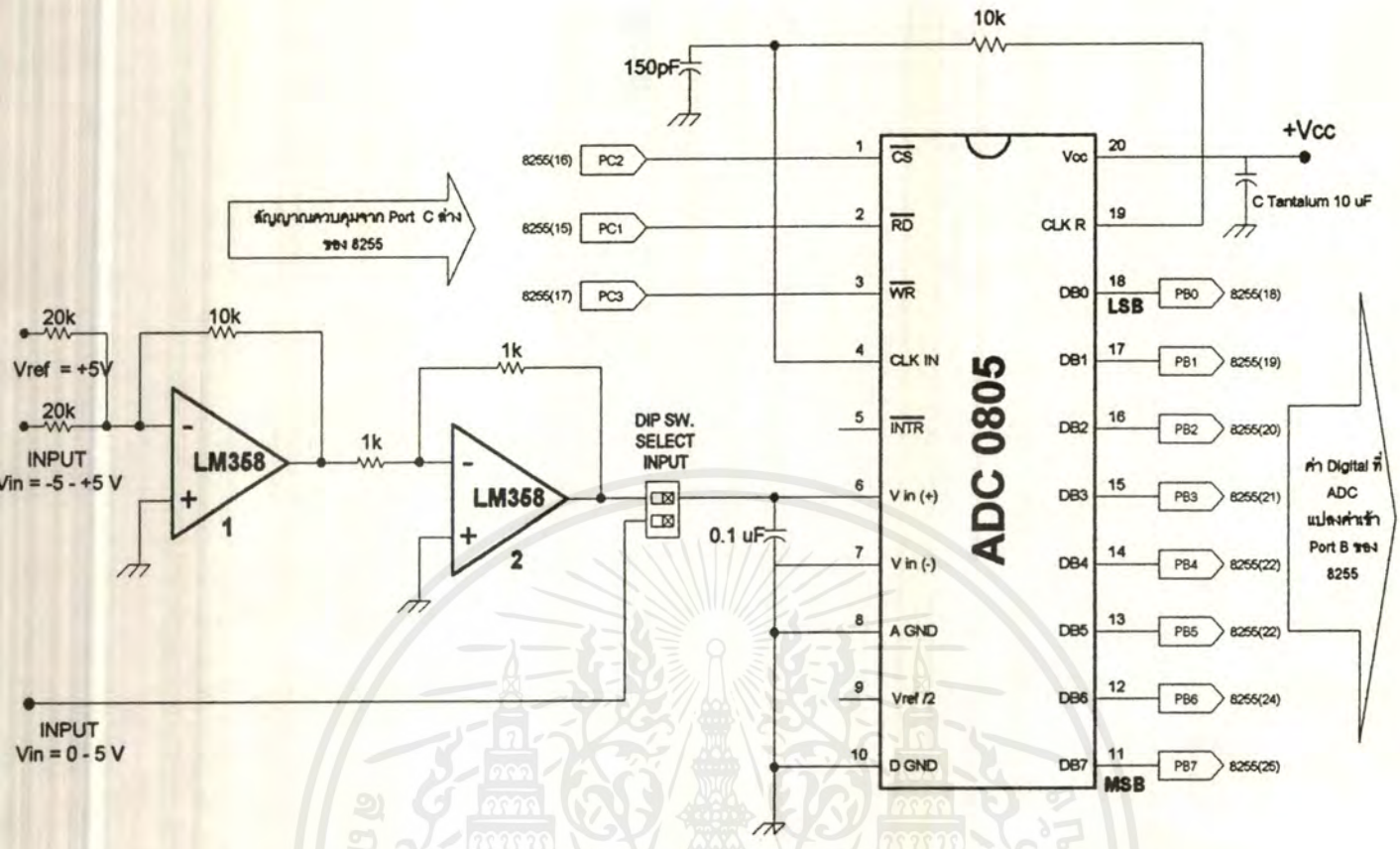
Timing Diagram ของ ADC0805



รูปที่ 3.7 แสดง Timing Diagram ของ ADC0805

การต่อวงจรใช้งานของ ADC0805

จากรูปที่ 3.6 ได้นำเอา ADC0805 ไปทำการต่อใช้งานโดยที่จะทำการต่อลงบน Interface Card ซึ่งจะติดต่อกับคอมพิวเตอร์โดยผ่านทางไอซี 8255 โดยจะต่อวงจรดังรูปที่ 3.8 และวงจรนี้เป็นส่วนหนึ่งของ Interface Card ซึ่งจะมีส่วนไปติดต่อกับรูปที่ 3.4 (ส่วนสัญญาณควบคุม และส่วนข้อมูลเข้าคอมพิวเตอร์)



รูปที่ 3.8 แสดงวงจรการนำไปใช้งานของ ADC0805

จากวงจรข้างต้นนี้ได้มีการบันทึกข้อมูลเพื่อที่จะนำไปใช้ช่วยในการเขียนโปรแกรมดังตาราง ที่ 3.2 และ 3.3

ตารางที่ 3.2 แสดงผลการแปลงค่าของ ADC0805 เมื่อใส่อินพุท 0-5 Volt

Input Voltage	Digital Output from ADC 0805 MSB LSB
0.00	0000 0000
0.25	0000 1100
0.50	0001 1001
0.75	0010 0110
1.00	0011 0011

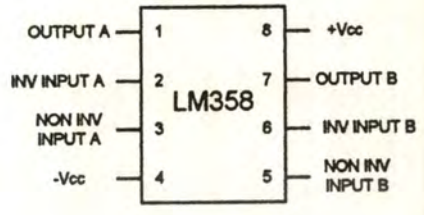
ตารางที่ 3.2 (ต่อ) แสดงผลการแปลงค่าของ ADC0805 เมื่อใส่อินพุท 0-5 Volt

Input Voltage	Digital Output from ADC 0805 MSB LSB
1.25	0011 1111
1.50	0100 1100
1.75	0101 1001
2.00	0110 0101
2.25	0111 0010
2.50	0111 1111
2.75	1000 1011
3.00	1001 1000
3.25	1010 0101
3.50	1011 0010
3.75	1011 1110
4.00	1100 1011
4.25	1101 0111
4.50	1110 0100
4.75	1111 0001
5.00	1111 1111

ตารางที่ 3.3 แสดงผลการแปลงค่าของ ADC0805 เมื่อใส่อินพุท –5 - +5 Volt

Input Voltage	Digital Output from ADC 0805 MSB LSB
-5.05	0000 0000
-4.50	0000 1111
-4.00	0001 1011
-3.50	0010 1000
-3.00	0011 0101
-2.50	0100 0001
-2.00	0100 1110
-1.50	0101 1011
-0.50	0111 0100
0.00	1000 0000
0.50	1000 1101
1.00	1001 1000
1.50	1010 0110
2.00	1011 0010
2.50	1011 1111
3.00	1100 1100
3.50	1101 1000
4.00	1110 0101
4.50	1111 0001
5.00	1111 1111

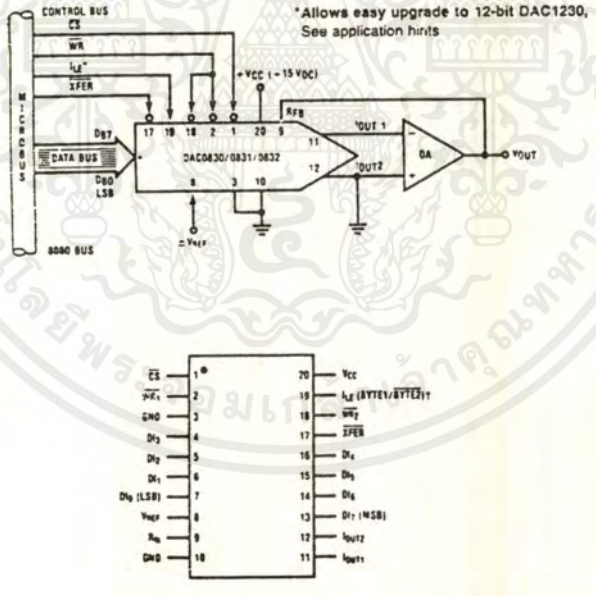
จากวงจรในรูปที่ 3.8 นั้นได้มีการใช้ไอซี LM358 มาเป็นวงจร Zero-span (จากรูปที่ 3.8 นั้นออปแอมป์ LM358 ตัวที่ 1 จะทำหน้าที่ปรับ Zero-span ส่วนตัวที่ 2 นั้นจะทำหน้าที่เป็น Inverter) ทำขึ้นเพื่อที่จะทำการปรับค่าอินพุทให้สามารถรับค่าได้ทั้ง –5 - +5 Volt และ 0 - 5 Volt ดังนั้นจึงได้แสดงรูปการจัดวางขาของไอซี LM358 ไว้ดังรูปที่ 3.9



รูปที่ 3.9 แสดงขาของไอซี LM358

3.3 การเลือกไอซี D/A มาใช้งานในวงจร (เลือก DAC0832) และวงจรที่ใช้งาน

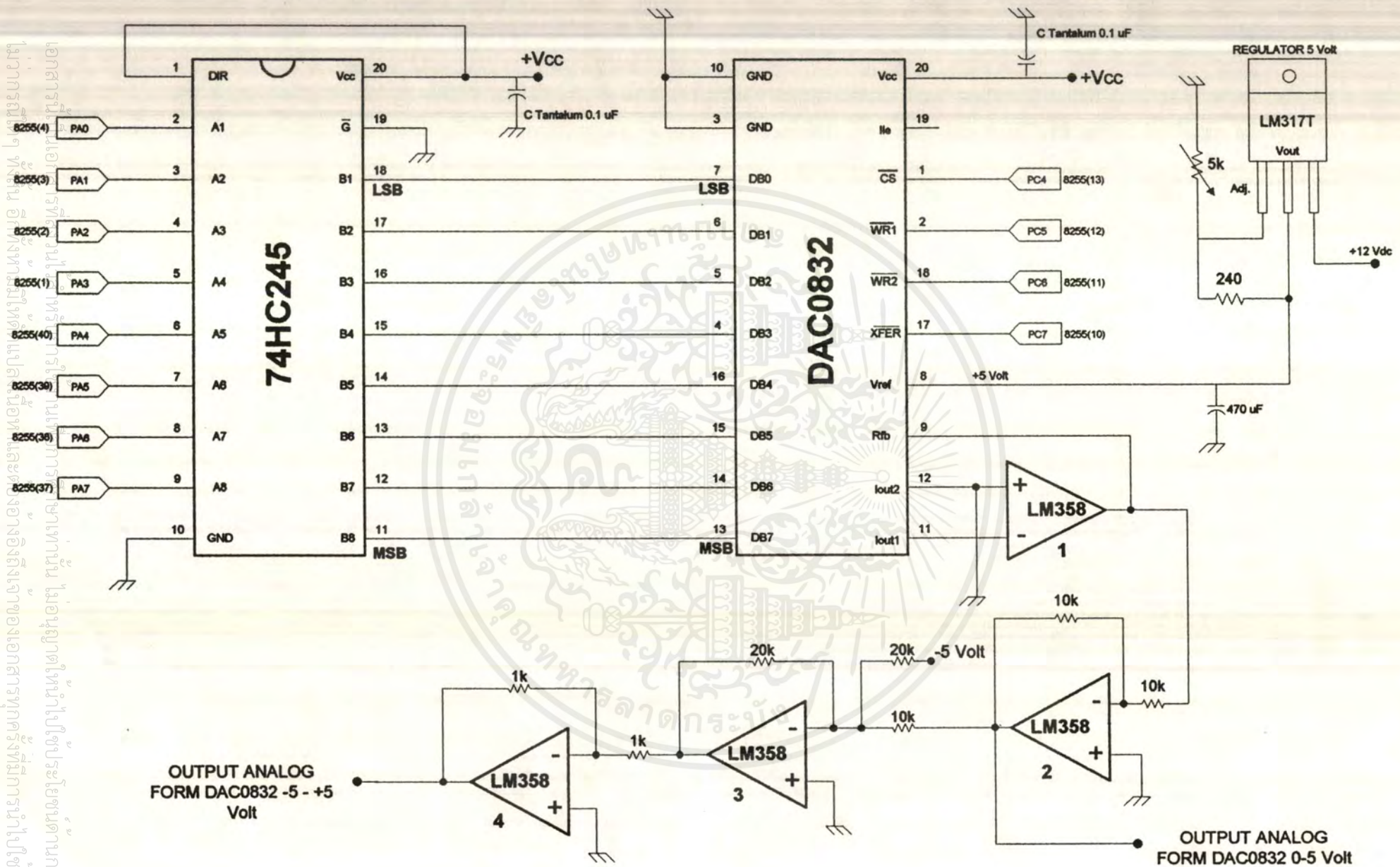
DAC0832 เป็นไอซีวงจร D/A 8 บิท ที่ใช้สำหรับต่อกับไมโครโปรเซสเซอร์แบบต่าง ๆ โดยที่จะต้องมี V_{ref} ต่อจากภายนอก และเป็น D/A แบบ R-2R Ladder



รูปที่ 3.10 แสดงวงจรการใช้งานของ DAC0832

ความหมายของขาต่าง ๆของ DAC0832

- V_{REF} Reference Voltage Input ต่อกับแหล่งจ่ายไฟภายนอกเพื่อใช้เป็น Reference ของวงจร R-2R Ladder โดยมีค่าได้ระหว่าง +10V ถึง –10V



รูปที่ 3.11 แสดงวงจรการนำไปใช้งานของ DAC0832

- Vcc — เป็นขาที่ต่อกับแหล่งจ่ายไฟ โดยเลือกค่าระหว่าง +5 Vถึง +15V
- I_{OUT1} และ I_{OUT2} — เป็นขา output ของวงจร
- GND — เป็นขา Ground ของวงจร
- $DI_0 - DI_7$ — เป็น ขา input 8 บิท ของวงจร
- R_{fb} — Feedback Resistor เป็น shunt ของ ออปแอมพ์ เพื่อทำให้มีค่า Voltage ออกมา
- $\overline{CS}$ — Chip Select (active low) เป็นขาที่ใช้เลือกให้ chip ทำงาน
- ILE — Input Latch Enable (active high) ใช้ร่วมกับ $\overline{CS}$ เพื่อใช้ในการควบคุม WR_1
- $\overline{WR_1}$ — ถ้ามีค่าเป็น "0" DI จะถูกคงค่าไว้ใน Input Latch ถ้ามีค่าเป็น "1" จะทำการ Latch ค่าใหม่เข้ามา
- $\overline{WR_2}$ — (Active Low) ใช้ร่วมกับ $\overline{XFER}$ เพื่อให้ค่าที่อยู่ใน Input Latch ส่งไปที่ DAC Register
- $\overline{XFER}$ — Transfer Control Signal (Active Low) ใช้เพื่อควบคุม $\overline{WR_2}$

การต่อวงจรใช้งานของ DAC0832

จากรูปที่ 3.10 ได้นำเอา DAC0832 ไปทำการต่อใช้งานโดยที่จะทำการต่อลงบน Interface card ซึ่งจะติดต่อกับคอมพิวเตอร์โดยผ่านทางไอซี 8255 โดยจะต่อวงจรดังรูปที่ 3.11 และวงจรนี้เป็นส่วนหนึ่งของ Interface card ซึ่งจะมีส่วนไปติดต่อกับรูปที่ 3.4 (ส่วนสัญญาณควบคุม และส่วนข้อมูลออกจากคอมพิวเตอร์)

จากวงจรรูปที่ 3.11 นี้ได้มีการบันทึกข้อมูลเพื่อที่จะนำไปใช้ช่วยในการเขียนโปรแกรมดังตารางที่ 3.4

ตารางที่ 3.4 แสดงผลการแปลงค่าของ DAC0832

Binary Input B1 B2 B3 B4 B5 B6 B7 B8	Voltage output (0-5 Volt)	Voltage output (-5 - +5 Volt)
1 1 1 1 1 1 1 1	5.00	5.03
1 1 1 1 1 1 1 0	4.99	5.02
1 1 1 1 1 1 0 0	4.99	4.95
1 1 1 1 1 0 0 0	4.92	4.80
1 1 1 1 0 0 0 0	4.76	4.48
1 1 1 0 1 0 0 0	4.60	4.16
1 1 1 0 0 0 0 0	4.44	3.83
1 1 0 0 1 0 0 0	3.96	2.87
1 1 0 0 0 0 0 0	3.80	2.55
1 0 1 0 1 0 0 0	3.33	1.58

ตารางที่ 3.4 (ต่อ) แสดงผลการแปลงค่าของ DAC0832

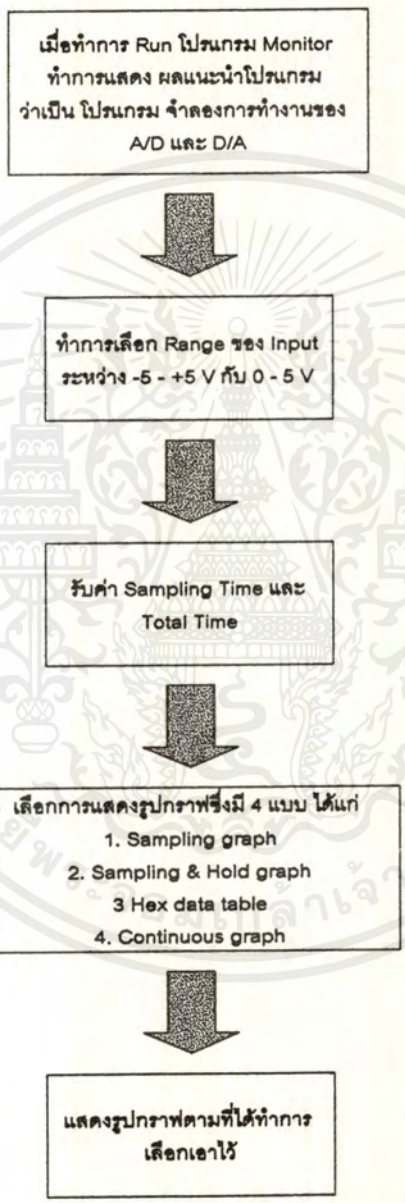
Binary Input B1 B2 B3 B4 B5 B6 B7 B8	Voltage output (0-5 Volt)	Voltage output (-5 - +5 Volt)
1 0 1 0 0 0 0 0	3.17	1.26
1 0 0 0 0 0 0 0	2.53	-0.02
0 1 1 1 1 1 1 1	2.50	-0.09
0 1 0 0 1 1 1 1	1.56	-1.99
0 0 1 1 1 1 1 1	1.25	-2.62
0 0 1 1 0 0 0 0	0.95	-3.27
0 0 1 0 0 0 0 0	0.63	-3.88
0 0 0 1 1 1 1 1	0.63	-3.89
0 0 0 1 0 1 0 0	0.39	-4.38
0 0 0 1 0 0 0 0	0.31	-4.53
0 0 0 0 1 1 1 1	0.31	-4.56
0 0 0 0 1 0 0 0	0.15	-4.85
0 0 0 0 0 1 1 1	0.14	-4.89
0 0 0 0 0 0 1 1	0.05	-5.05
0 0 0 0 0 0 1 0	0.03	-5.09
0 0 0 0 0 0 0 1	0.02	-5.13
0 0 0 0 0 0 0 0	0.00	-5.17

จากรูปที่ 3.11 จะเห็นได้ว่าวงจรที่ใช้งานจริงนั้นจะมีส่วนของ Regulator ประกอบอยู่ด้วย โดยที่จะทำหน้าที่เป็นตัวจ่าย V_{ref} ให้แก่ DAC0832 ซึ่ง Regulator ที่ใช้นี้จะใช้ไอซีเบอร์ LM317T ซึ่งส่วนของวงจรนั้นสามารถ ดูได้จากรูปที่ 3.11 และในวงจรนี้นั้นยังมีการใช้วงจร Zero-span โดยใช้ Op-Amp โดยใช้ Op-amp เบอร์ LM358 เช่นเดียวกับวงจร ADC (โดยที่จากรูป 3.11 Op-Amp LM358 ตัวที่ 3 จะทำหน้าที่เป็นตัวปรับ Zero-span และ Op-Amp LM358 ตัวที่ 4 จะทำหน้าที่เป็นตัว Inverter)

บทที่ 4
การออกแบบซอฟท์แวร์และฮาร์ดแวร์

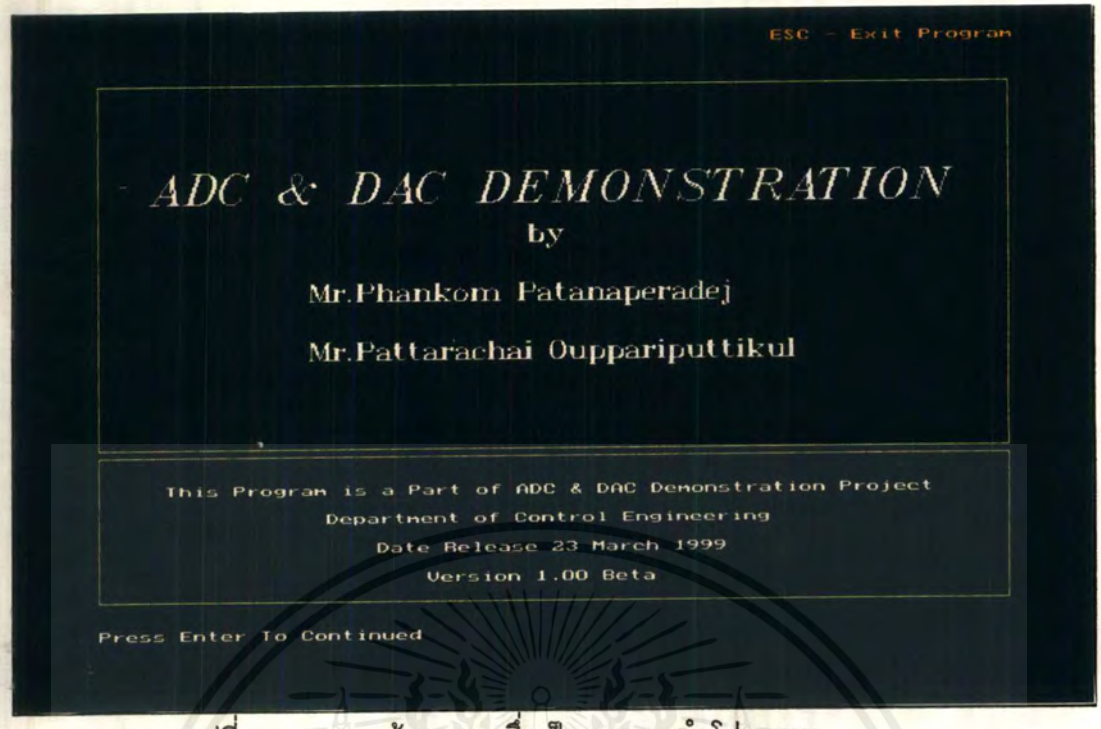
4.1 การออกแบบด้านซอฟท์แวร์

ในโครงงานนี้ เราใช้ภาษาซี ในการควบคุมการทำงานของเครื่องและ ควบคุมการทำงานของการ์ด ซึ่งจะมีโฟล์วชาร์ตของโปรแกรมดังนี้



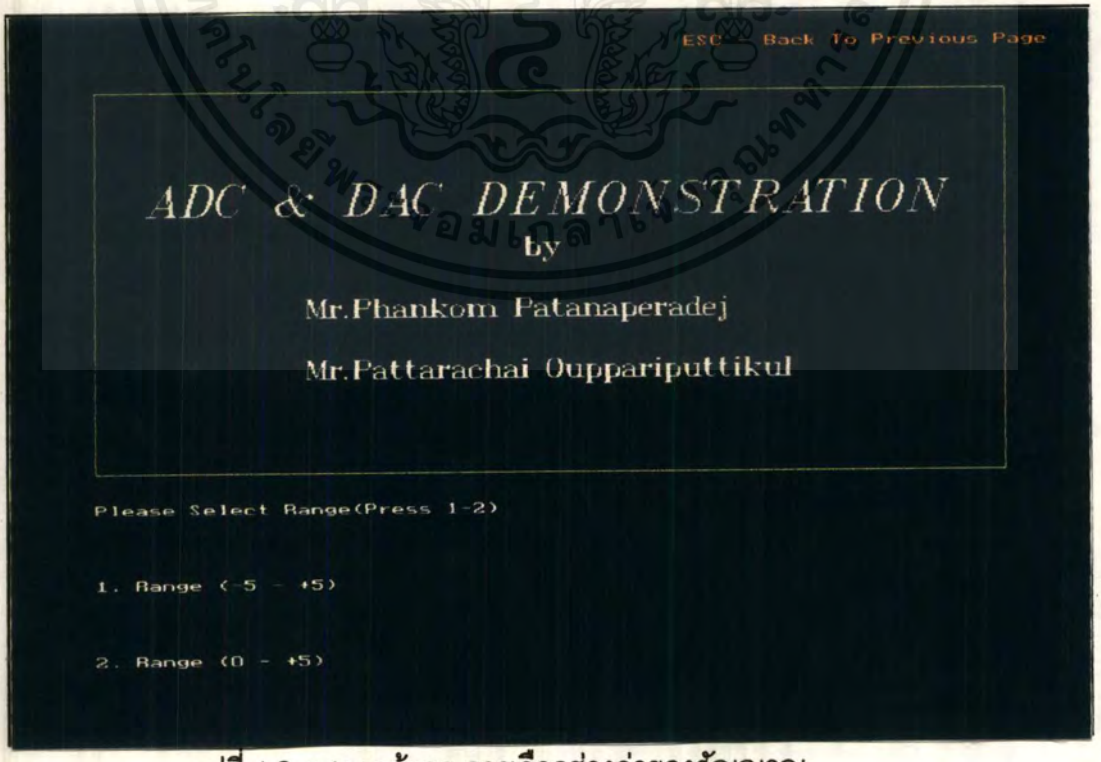
4.1.1 การใช้งานโปรแกรม

เมื่อทำการ RUN โปรแกรมแล้ว โปรแกรมจะแสดงหน้าจอแรก ซึ่งเป็นหน้าจอแนะนำโปรแกรม จะต้องกดปุ่ม ENTER เพื่อแสดงหน้าจอต่อไป หรือ กดปุ่ม ESC เพื่อออกจากโปรแกรมกลับเข้าสู่ระบบปฏิบัติการดังรูปที่ 4.1



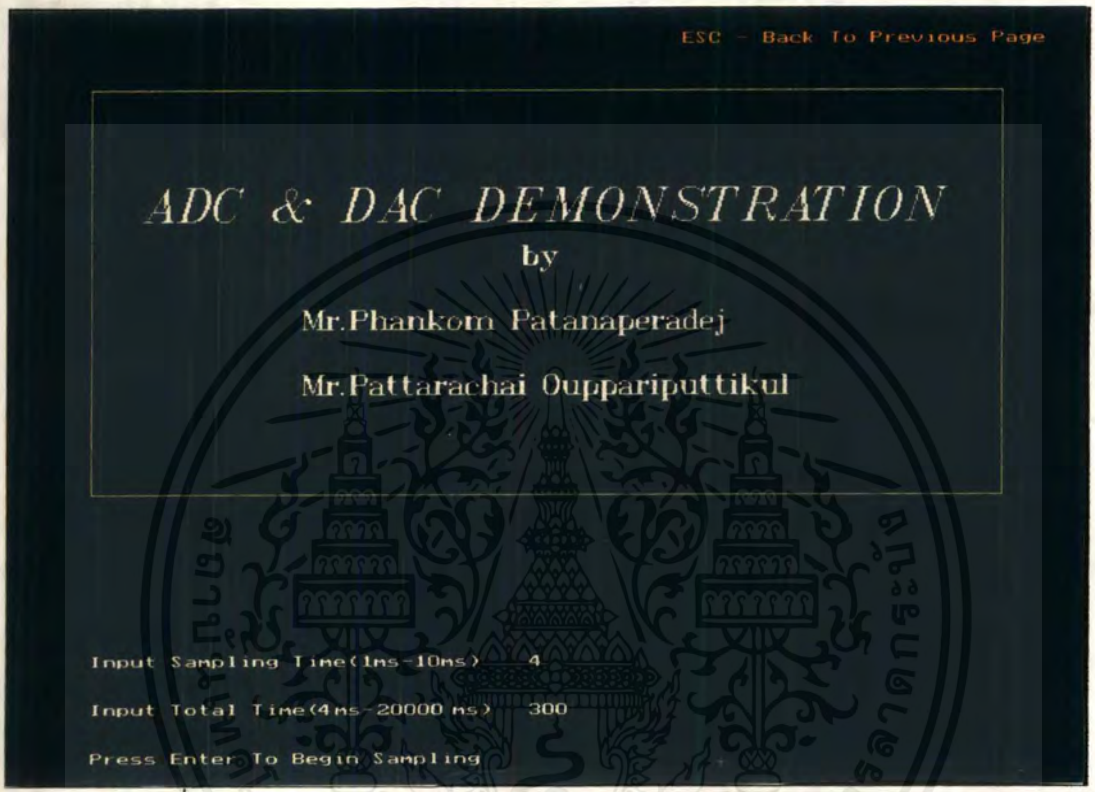
รูปที่ 4.1 แสดงหน้าจอแรก ซึ่งเป็นการแนะนำโปรแกรม

หน้าจอต่อมาจะเป็นหน้าจอสำหรับให้ผู้ใช้งานโปรแกรมเลือกช่วงของค่าสัญญาณ Input ระหว่าง –5V ถึง +5V หรือ 0V ถึง +5V โดยการพิมพ์ตัวเลือก 1 หรือ 2 ลงไป โปรแกรมก็จะทำงานตามที่เลือก โดยถ้าต้องการกลับไปหน้าจอแรก ก็สามารถทำได้โดยการกดปุ่ม ESC ดังรูปที่ 4.2



รูปที่ 4.2 แสดงหน้าจอ การเลือกช่วงค่าของสัญญาณ

หน้าจอต่อมาจะให้ผู้ใช้ พิมพ์ค่า Sampling Time และ ค่าเวลาทั้งหมดที่ต้องการ Sampling (Total Time) เพื่อที่จะนำค่าทั้ง 2 ค่า ไปใช้ในการ Sampling สัญญาณ โปรแกรมจะยังไม่ทำการ Sampling สัญญาณ จนกว่าผู้ใช้จะกดปุ่ม ENTER ซึ่งจะเป็นการบอกให้โปรแกรมเริ่มทำการ Sampling ตั้งแต่เวลาที่กดปุ่ม ENTER ไปจนถึงเวลาทั้งหมดที่ต้องการ (Total Time) ถ้าต้องการกลับไปหน้าจอที่ผ่านมาก็สามารถทำได้โดยการกดปุ่ม ESC ดังรูปที่4.3



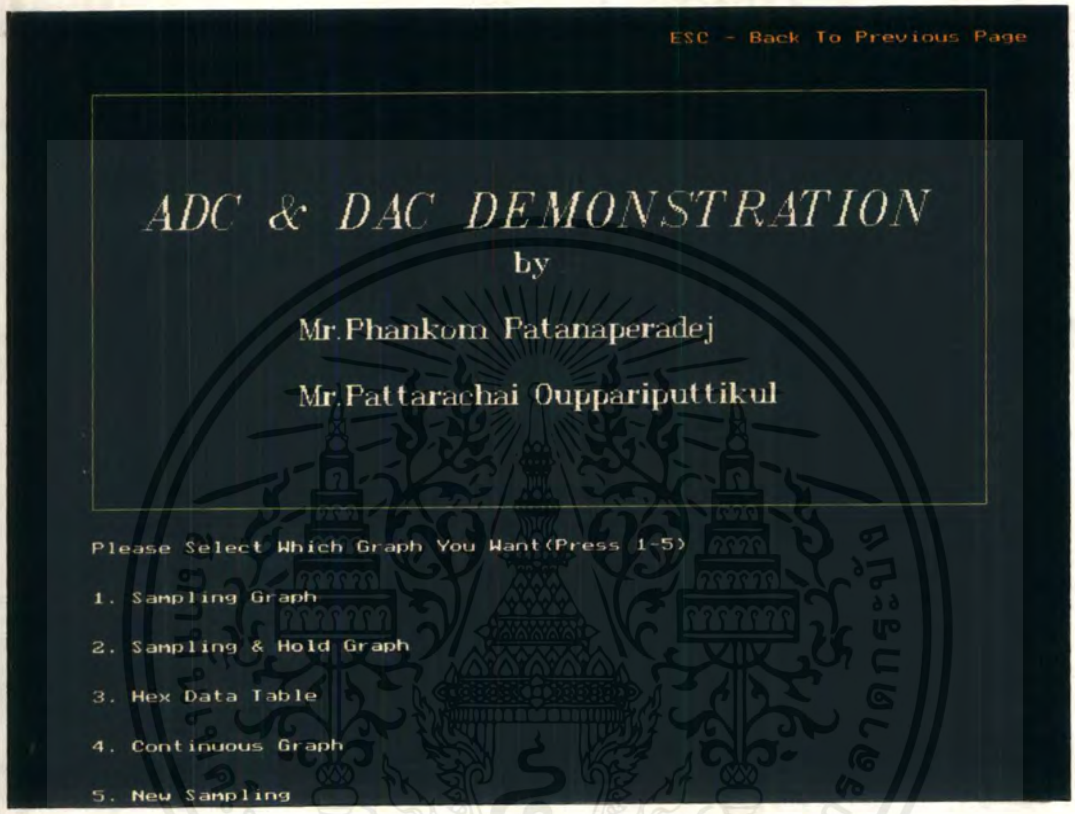
รูปที่ 4.3 แสดงหน้าจอ การรับค่า Sampling time และ Total time

หลังจากที่โปรแกรมทำการ Sampling ค่าตามเวลาที่ต้องการมาได้ ค่าที่ถูก Sampling มาจะถูกเก็บไว้ หน้าจอขึ้น MENU ให้เลือกว่า จากสัญญาณที่ทำการ Sampling มานั้น ผู้ใช้ต้องการดูรูปกราฟในลักษณะไหน โดยผู้ใช้โปรแกรมสามารถเลือกได้โดยการพิมพ์ตัวเลือก 1-4 เพื่อเลือกการดูรูปกราฟ 4 แบบ ดังนี้

1. Sampling Graph
2. Sampling & Hold Graph
3. Hex Data Table
4. Continuous Graph

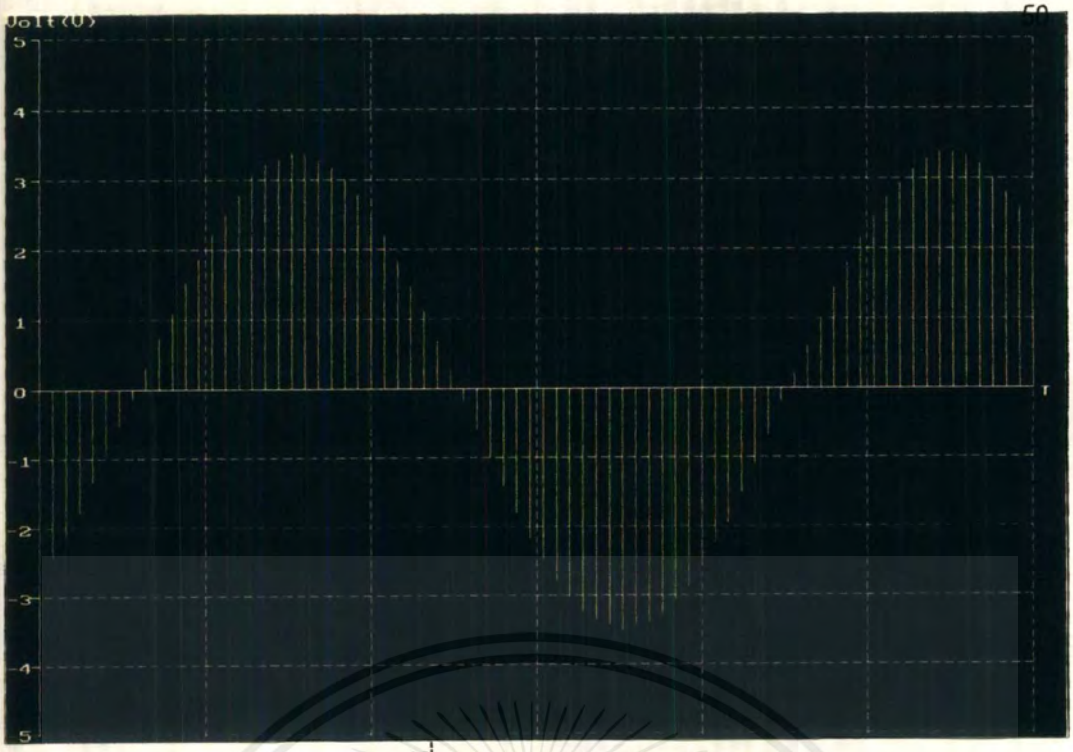
ซึ่งจะอธิบายลักษณะของกราฟต่างๆในตอนต่อไป

สำหรับ ตัวเลือก 5 นั้นมีไว้เพื่อต้องการที่จะกลับไปทำการ Sampling สัญญาณใหม่อีกครั้ง และถ้าต้องการกลับไปหน้าจอที่ผ่านมาก็สามารถทำได้โดยการกดปุ่ม ESC หน้าจอเลือกลักษณะรูปกราฟแสดงดังรูปที่ 4.4



รูปที่ 4.4 แสดหน้าจอการเลือกรูปกราฟ

จากหน้าจอที่ผ่านมา ถ้าเลือกตัวเลือกที่ 1 จะแสดงรูป Sampling Graph ซึ่งเป็นกราฟที่ ADC ทำการ Sampling ค่าสัญญาณ Sine Wave ตาม Sapling time ที่ผู้ใช้ป้อนให้ กราฟจะมีความละเอียดและคล้ายรูปสัญญาณจริงขึ้น ถ้าใช้ Sampling time น้อยลง จากรูปสัญญาณตัวอย่างที่เราทำการ Sampling เป็นสัญญาณ Sine Wave จะได้ผลดังรูปที่ 4.5

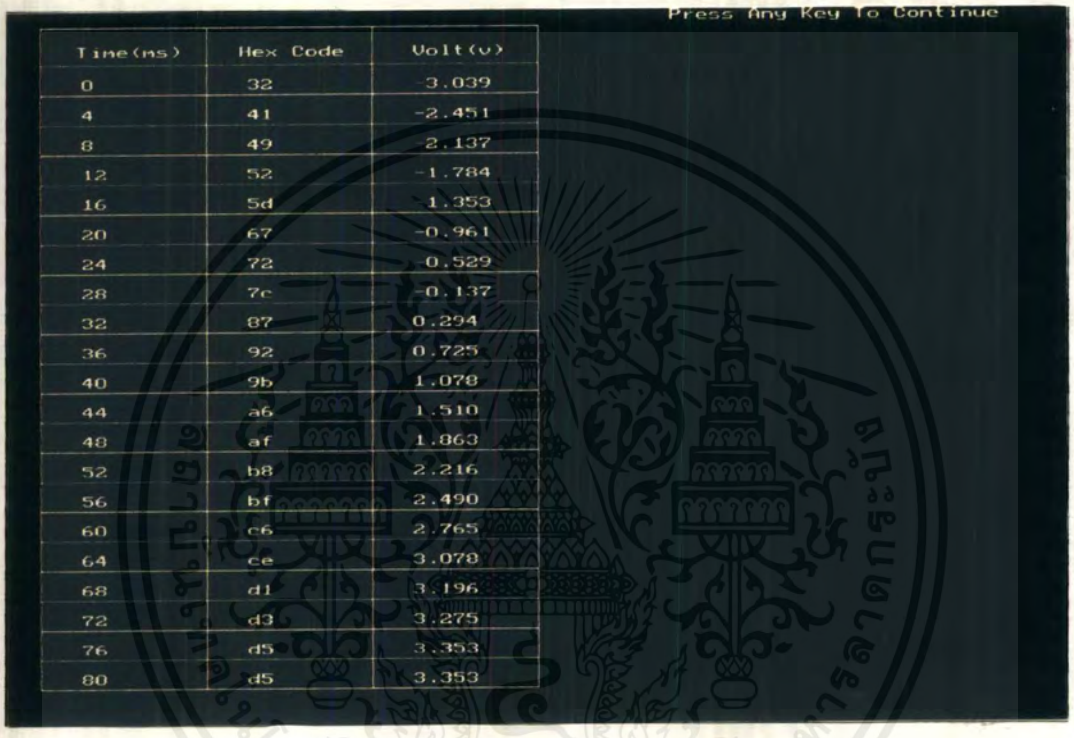


รูปที่ 4.5 Sampling Graph

ถ้าเลือกตัวเลือกที่ 2 จะแสดงรูป Sampling & Hold Graph ซึ่งจะแสดงการ Hold ค่า ของ ADC กระบวนการนี้สำหรับ ADCจริงๆมีไว้เพื่อให้เวลา ADC ทำการแปลงค่าสัญญาณจาก อนาล็อกมาเป็นดิจิตอล ซึ่งจากกราฟ เราจะจำลองการ Hold ค่าด้วยเส้นกราฟสีแดง ส่วนสัญญาณ Sampling จะเป็นเส้นสีเหลือง ดังรูป 4.6

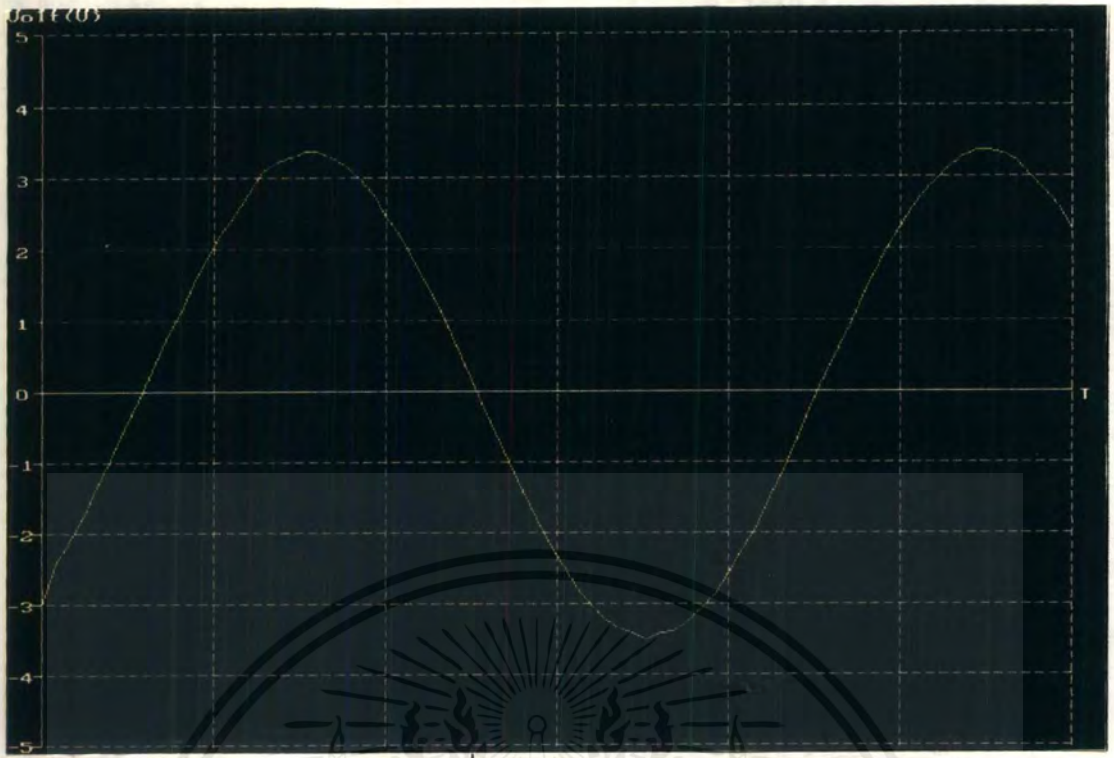
รูปที่ 4.6 Sampling & Hold Graph

ถ้าเลือกตัวเลือกที่ 3 จะแสดง Hex Data Table จากการ Hold ADC จะทำการแปลงค่าอนาล็อกมาเป็นดิจิตอล ณ เวลาที่ Sampling นั้นๆ ค่าดิจิตอลที่ได้จากการ Hold จะถูกแสดงใน Hex Data Table ซึ่งจะแสดงค่าดิจิตอล (เลขฐาน 16) ณ เวลา Sampling นั้นๆ ออกมา พร้อมทั้งแสดงค่า อนาล็อกของสัญญาณที่เข้ามาเป็น Voltage อีกด้วย ดังรูปที่ 4.7



รูปที่ 4.7 Hex Data Table

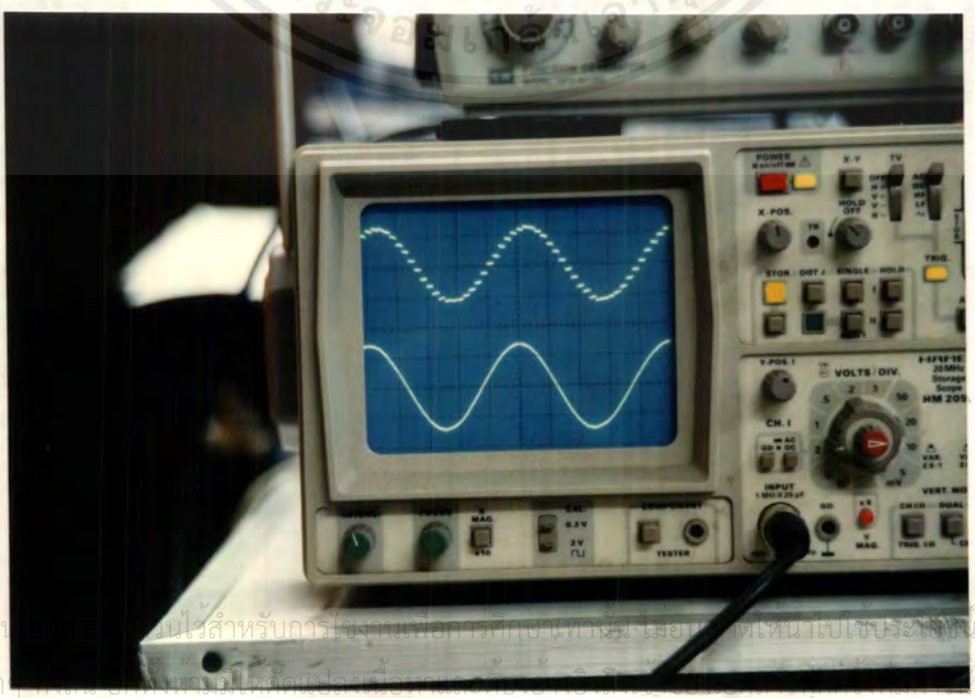
และถ้าเลือกตัวเลือกที่ 4 จะแสดง Continuous Graph ซึ่งหลังจากที่เราได้ค่าดิจิตอลมาแล้ว เราก็ทำการป้อนสัญญาณดิจิตอลนั้น เข้า DAC เพื่อทำการจำลองรูปสัญญาณเดิมขึ้นมาใหม่ รูปสัญญาณที่ได้หลังจากผ่าน DAC แล้วจะมีลักษณะคล้ายกับรูปสัญญาณก่อนที่จะถูก ADC Sampling ดังรูปที่ 4.8



รูปที่ 4.8 Continuous Graph

โดยเมื่อดูรูปกราฟแต่ละรูปเสร็จแล้ว โปรแกรมจะกลับมายังหน้าจอเลือกลักษณะรูปกราฟอีกครั้ง เพื่อให้ผู้ใช้สามารถเลือกดูลักษณะรูปกราฟอื่น ๆ ของสัญญาณที่ถูก Sampling เดิม โดยค่าของสัญญาณจะถูกเก็บไว้และสามารถเรียกขึ้นมาดูจนกว่าจะทำการ Sampling ใหม่ จะได้รูปกราฟใหม่

ส่วนสัญญาณ Output จาก D/A ที่จะถูกส่งออกมาจากคอมพิวเตอร์นั้นจะสามารถตรวจจับได้โดยการใช้ Oscilloscope ตรวจวัดออกมา ซึ่งรูปของสัญญาณที่ Oscilloscope นั้นจะสามารถแสดงได้ดังรูปที่ 4.9



รูปที่ 4.9 แสดงสัญญาณที่ถูก D/A ส่งออกมาจากคอมพิวเตอร์ และตรวจจับด้วย Oscilloscope

สำหรับการออกจากโปรแกรม ผู้ใช้โปรแกรมก็เพียงแค่กดปุ่ม ESC จนกว่าจะกลับมายังหน้าจอแรก และกดปุ่ม ESC อีกครั้งเพื่อกลับไปสู่ระบบปฏิบัติการ

4.2 การออกแบบด้านฮาร์ดแวร์

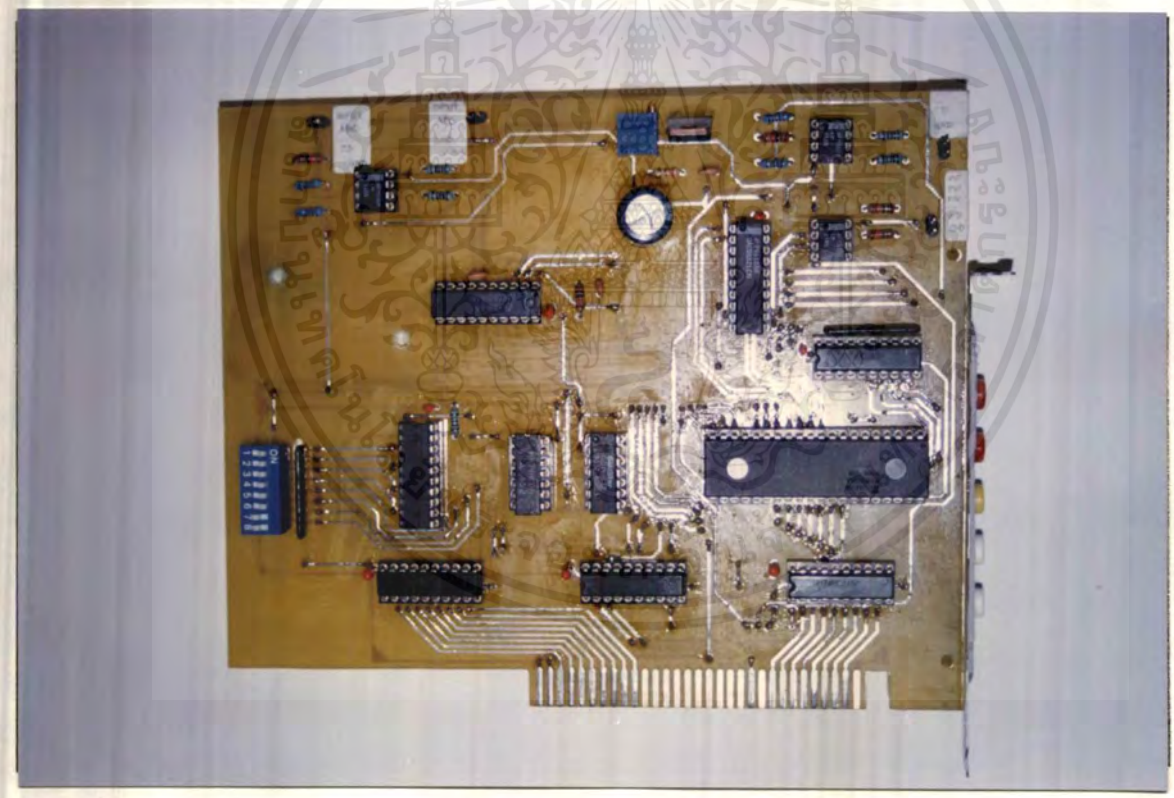
สำหรับการออกแบบด้านฮาร์ดแวร์นั้นจะประกอบไปด้วยวงจร 3 ส่วน ซึ่งได้กล่าวมาแล้วในบทที่ 3 ซึ่งอาจจะทำการแบ่งแยกออกเป็นส่วน ๆ ได้ดังนี้ คือ

1. ส่วนพอร์ท I/O ที่ใช้สำหรับติดต่อ รับ-ส่งข้อมูล นั่นก็คือส่วนของ 8255 บนการ์ดอินเตอร์เฟส

2. ส่วนแปลงอนาล็อกเป็นดิจิตอล (ส่วนของ ADC0805)

3. ส่วนแปลงดิจิตอลเป็นอนาล็อก (ส่วนของ DAC0832)

ซึ่งทั้ง 3 ส่วนนี้ได้ทำการต่อรวมกันบนการ์ดเดียว โดยที่จะแสดงรูปของวงจรที่ทำการต่อเรียบร้อยแล้วไว้ในรูปที่ 4.10



รูปที่ 4.10 แสดงการ์ดวงจร A/D D/A ที่ต่อวงจรเสร็จเรียบร้อยแล้ว

บทที่ 5
สรุปและวิจารณ์

5.1 สรุปผลการดำเนินงาน

ในการทำปริญญานิพนธ์นี้ ได้ทำชุดจำลองการทำงานของการแปลงสัญญาณอนาล็อกเป็นดิจิตอลและการแปลงสัญญาณดิจิตอลให้เป็นอนาล็อก โดยแสดงผลการจำลองการทำงานออกที่คอมพิวเตอร์ ซึ่งในการทำงานครั้งนี้สิ่งที่ได้ทำประกอบไปด้วยส่วนประกอบ 2 ส่วนคือ Hardware และ Software

โดยในส่วนของ Hardware นั้นได้ทำการ์ด A/D D/A เพื่อที่จะ รับ-ส่ง ข้อมูลเข้าสู่คอมพิวเตอร์ แล้วนำไปใช้ในการคำนวณค่าและแสดงผล

ในส่วนของ Software นั้นได้ทำการเขียนโปรแกรมภาษา C เพื่อที่จะใช้ในการควบคุมการทำงานของการ์ด A/D D/A และใช้ในการแสดงผลการจำลองการทำงานของ A/D และ D/A

5.2 ปัญหาที่เกิดขึ้นและแนวทางแก้ไข

สำหรับปัญหาที่พบบ่อย คือ สัญญาณรบกวนจากภายนอกขณะทำการ Sampling ซึ่งมีผลทำให้กราฟที่แสดงมีความคลาดเคลื่อนไปจากจริงบ้าง

นอกจากนี้ ถ้าสัญญาณที่ต้องการจะ Sampling มีค่าคาบน้อยกว่า 1ms จะมีผลทำให้รูปกราฟที่ได้ผิดจากความเป็นจริง ดังนั้นสัญญาณที่จะทำการ Sampling ควรจะมีความถี่น้อยกว่า 1 K Hz

5.3 แนวทางการพัฒนาในอนาคต

ควรจะพัฒนาให้รับอินพุทได้ช่วงกว้างกว่าที่ทำอยู่ และ หากเปลี่ยนแนวทางการเขียนโปรแกรมให้สามารถ RUN บน Windows ได้ จะทำให้รูปกราฟที่ออกมามีความสวยงามมากกว่า แต่อาจจะก่อให้เกิดปัญหากับการ Sampling เนื่องจาก Windows อาจจะทำให้การเวลาของการ Sampling ช้าลง

นอกจากนี้ ควรจะดัดแปลงให้สามารถรับสัญญาณที่มี ความถี่มากกว่า 1 K Hz ซึ่งอาจทำได้โดยใช้โปรแกรมภาษาอื่นที่สามารถสร้างฐานเวลาได้ต่ำกว่า 1 ms หรือ อาจใช้ฐานเวลาจากภายนอก

บรรณานุกรม

ธานินทร์ ถาวรศาศนวงศ์, ทินกร ดุ๊ก ; การอินเทอร์เฟส IBM PC ; หจก. สำนักพิมพ์ฟิสิกส์เซ็นเตอร์ , 2536

ไมโครโปรเซสเซอร์ Z-80 , บริษัทซีเอ็ดยูเคชั่น จำกัด

วารสาร Semiconductor เล่มที่ 103 ประจำเดือนธันวาคม 2533 หน้า 302-309

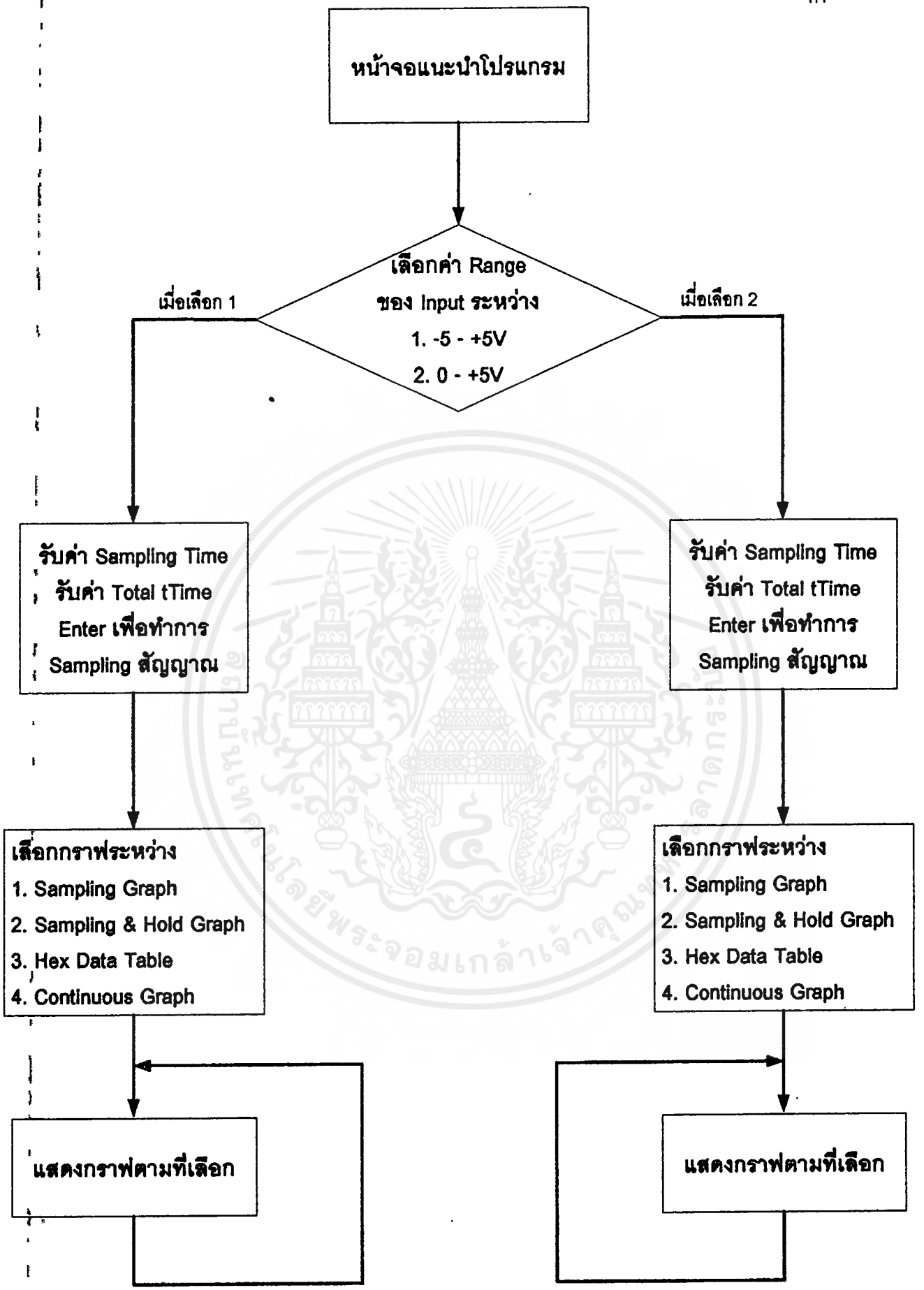
Willis J. Tompkins , John G. Webster ; Interfacing Sensor To The IBC PC

Data Acquisition Databook ; National Semiconductor , 1995

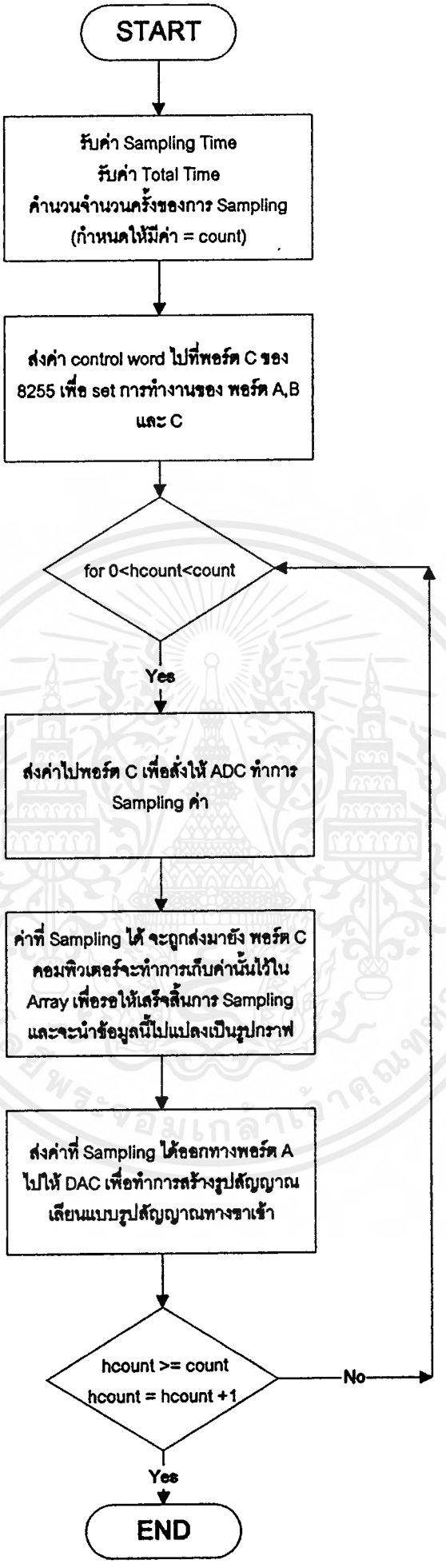
ECG Semiconductors Master Replacement Guide

ภาคผนวก ก

Program



รูปที่ 1ก Flow Chart แสดงรายละเอียดการทำงานของโปรแกรม



รูปที่ 2n Flow Chart แสดงขั้นตอนการทำงานของโปรแกรมในการเก็บ และส่งค่า

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>
#include<stdlib.h>
#include<math.h>

unsigned fstime,stime,ftime;
int gdriver = DETECT,gmode;
int count,store[5010],hcount,rcount,page;

unsigned getnum(int xx,int yy,int page);
void del(int xx1,int xx2,int yy,int col);
void escmenu();
void menu();
void choosea();
void chooseb();
void about();
void range();
void tinputa();
void tinputb();
void wgraph();
void wgraph1();
void discretea(int count,int hcount,int rcount,unsigned stime,int store[5010]);
void discreteb(int count,int hcount,int rcount,unsigned stime,int store[5010]);
void holda(int count,int hcount,int rcount,unsigned stime,int store[5010]);
void holdb(int count,int hcount,int rcount,unsigned stime,int store[5010]);
void table();
void dataa(int count,int store[5010],int hcount,int stime);
void datab(int count,int store[5010],int hcount,int stime);
void signala(int count,int hcount,int rcount,unsigned stime,int store[5010]);
void signalb(int count,int hcount,int rcount,unsigned stime,int store[5010]);
void pageone();
void pagetwo();
void pagethreea();
void pagethreeb();
void pagefoura();
void pagefourb();
void main()
        {
```

```
                clrscr();
                initgraph(&gdriver,&gmode,"c:\\tc\\bgi");
                textbackground(BLACK);
                pageone();
        }



unsigned getnum(int xx,int yy,int page)
        {
          char ch;
          char* cc;
          int nub=0;
                do
                 {
                 ch=getch();
                 if ((ch!='\r')&&(ch!='\b')&&(ch!=27))
                  {
                   cc[nub]=ch; cc[nub+1]='\0'; nub++;
                  }
                 if ((ch=='\b')&&(nub>0))
                  {
                  nub=nub-1; cc[nub]='\0';
                  }
                 if((ch==27)&&(page==2))
                  {
                   cleardevice();
                   pageone();
                  }
                 if((ch==27)&&(page==3))
                  {
                   cleardevice();
                   pagetwo();
                  }
                 if((ch==27)&&(page==4))
                  {
                   cleardevice();
                   pagetwo();
                  }
                 if((ch==27)&&(page==5))
                  {
```

```
                cleardevice();
                pagethreea();
               }
              if((ch==27)&&(page==6))
               {
                cleardevice();
                pagethreeb();
               }
              del(xx,xx+480,yy,BLACK);
              outtextxy(xx,yy,cc);
             }
            while(ch!='\r');
            if((nub==0)&&(page==2))
              {
               cleardevice();
               pagetwo();
              }
            if((nub==0)&&(page==3))
              {
               cleardevice();
               pagethreea();
              }
            if((nub==0)&&(page==4))
              {
               cleardevice();
               pagethreeb();
              }
            if((nub==0)&&(page==5))
              {
               cleardevice();
               pagefoura();
              }
            if((nub==0)&&(page==6))
              {
               cleardevice();
               pagefourb();
              }
            nub=atoi(cc);
            return(nub);
}
```

```
void del(int xx1,int xx2,int yy,int col)
        {
                setfillstyle(1,col);
                bar(xx1,yy-8,xx2,yy+8);
        }

void escmenu()
        {
                cleardevice();
                setcolor(14);
                line(50,50,590,50);
                line(50,300,590,300);
                line(50,50,50,300);
                line(590,50,590,300);
                setcolor(WHITE);
                settextstyle(7,0,4);
                outtextxy(80,100,"ADC & DAC DEMONSTRATION");
                settextstyle(1,0,1);
                outtextxy(305,140,"by");
                outtextxy(175,180,"Mr.Phankom Patanaperadej");
                outtextxy(175,220,"Mr.Pattarachai Ouppariputtikul");
                settextstyle(0,0,1);
                setcolor(12);
                outtextxy(450,15,"ESC - Exit Program");
                setcolor(WHITE);
        }

void menu()
        {
                cleardevice();
                setcolor(14);
                line(50,50,590,50);
                line(50,300,590,300);
                line(50,50,50,300);
                line(590,50,590,300);
                setcolor(WHITE);
                settextstyle(7,0,4);
                outtextxy(80,100,"ADC & DAC DEMONSTRATION");
                settextstyle(1,0,1);
                outtextxy(305,140,"by");
                outtextxy(175,180,"Mr.Phankom Patanaperadej");
```

```
                outtextxy(175,220,"Mr.Pattarachai Ouppariputtikul");
                settextstyle(0,0,1);
                setcolor(12);
                outtextxy(400,15,"ESC - Back To Previous Page");
                setcolor(WHITE);
        }

void about()
        {
                setcolor(14);
                line(50,305,590,305);
                line(50,305,50,405);
                line(590,305,590,405);
                line(50,405,590,405);
                setcolor(WHITE);
                settextstyle(0,0,1);
                outtextxy(90,325,"This Program is a Part of ADC & DAC Demonstration Project");
                outtextxy(185,345,"Department of Control Engineering");
                outtextxy(215,365,"Date Release 23 March 1999");
                outtextxy(245,385,"Version 1.00 Beta");
                outtextxy(50,425,"Press Enter To Continued");
        }

void range()
        {
                int select1;
                menu();
                outtextxy(50,320,"Please Select Range(Press 1-2)");
                outtextxy(50,370,"1. Range (-5 - +5)");
                outtextxy(50,420,"2. Range (0 - +5)");
                select1 = getnum(300,320,2);
                cleardevice();
                switch(select1)
                 {
                   case 1:{
                             pagethreea();
                             break;
                            }
                   case 2:{
                             pagethreeb();
                             break;
```

```
                    }
                default:range();
               }
        }

void tinputa()
        {
                char ststime[2],stfstime[10],r;
                outtextxy(50,400,"Input Sampling Time(1ms-10ms)");
                stime = getnum(310,400,3);
                if ((stime>=1)&&(stime<=10))
                 {
                  outtextxy(50,430,"Input Total Time(");
                  sprintf(ststime,"%u",stime);
                  outtextxy(185,430,ststime);
                  if(stime <10)
                   {
                    outtextxy(195,430,"ms-");
                    fstime = stime*5000;
                    sprintf(stfstime,"%u",fstime);
                    outtextxy(220,430,stfstime);
                    if(fstime<10000)
                     outtextxy(255,430,"ms)");
                     else
                      outtextxy(265,430,"ms)");
                     }
                    else
                     {
                      outtextxy(205,430,"ms-");
                      fstime =stime*5000;
                      sprintf(stfstime,"%u",fstime);
                      outtextxy(230,430,stfstime);
                      outtextxy(275,430,"ms)");
                     }
                   ftime = getnum(310,430,3);
                   if ((ftime>=stime)&&(ftime<=fstime))
                   {
                   outtextxy(50,460,"Press Enter To Begin Sampling");
                   while((r!=27)&&(r!='\r'))     r=getch();
                   if(r==27)
                    {
```

```
                    pagetwo();
                   }
                  }
                  else pagethreea();
                 }
                else pagethreea();
        }


void tinputb()
        {
                char ststime[2],stfstime[10],r;
                outtextxy(50,400,"Input Sampling Time(1ms-10ms)");
                stime = getnum(310,400,4);
                if ((stime>=1)&&(stime<=10))
                 {
                  outtextxy(50,430,"Input Total Time(");
                  sprintf(ststime,"%u",stime);
                  outtextxy(185,430,ststime);
                  if(stime <10)
                   {
                    outtextxy(195,430,"ms-");
                    fstime = stime*5000;
                    sprintf(stfstime,"%u",fstime);
                    outtextxy(220,430,stfstime);
                    if(fstime<10000)
                     outtextxy(255,430,"ms)");
                    else
                     outtextxy(265,430,"ms)");
                   }
                  else
                   {
                    outtextxy(205,430,"ms-");
                    fstime =stime*5000;
                    sprintf(stfstime,"%u",fstime);
                    outtextxy(230,430,stfstime);
                    outtextxy(275,430,"ms)");
                   }
                  ftime = getnum(310,430,4);
                  if ((ftime>=stime)&&(ftime<=fstime))
                   {
```

```
                    outtextxy(50,460,"Press Enter To Begin Sampling");
                    while((rl='\r')&&(rl=27))   r=getch();
                    if(r==27)
                     {
                      pagetwo();
                     }
                    }
                   else pagethreeb();
                  }
                 else pagethreeb();
        }

void choosea()
        {
                 int select;
                 menu();
                 outtextxy(50,320,"Please Select Which Graph You Want(Press 1-5)");
                 outtextxy(50,350,"1. Sampling Graph");
                 outtextxy(50,380,"2. Sampling & Hold Graph");
                 outtextxy(50,410,"3. Hex Data Table");
                 outtextxy(50,440,"4. Continuous Graph");
                 outtextxy(50,470,"5. New Sampling");
                 select = getnum(415,320,5);
                 cleardevice();
                 switch(select)
                  {
                    case 1:{
                              discretea(count,hcount,rcount,stime,store);
                              getch();
                              pagefoura();
                              break;
                             }
                    case 2:{
                              holda(count,hcount,rcount,stime,store);
                              getch();
                              pagefoura();
                              break;
                             }
                    case 3:{
                              dataa(count,store,hcount,stime);
                              getch();
```

```
                    pagefoura();
                    break;
                   }
            case 4:{
                    signala(count,hcount,rcount,stime,store);
                    getch();
                    pagefoura();
                    break;
                   }
            case 5:pagethreea();
                    break;
            default :choosea();
           }
    }

void chooseb()
    {
        int select;
        menu();
        outtextxy(50,320,"Please Select Which Graph You Want(Press 1-5)");
        outtextxy(50,350,"1. Sampling Graph");
        outtextxy(50,380,"2. Sampling & Hold Graph");
        outtextxy(50,410,"3. Hex Data Table");
        outtextxy(50,440,"4. Continuous Graph");
        outtextxy(50,470,"5. New Sampling");
        select = getnum(415,320,6);
        cleardevice();
        switch(select)
         {
           case 1:{
                    discreteb(count,hcount,rcount,stime,store);
                    getch();
                    pagefourb();
                    break;
                   }
           case 2:{
                    holdb(count,hcount,rcount,stime,store);
                    getch();
                    pagefourb();
                    break;
                   }
```

```
                case 3:{
                        datab(count,store,hcount,stime);
                        getch();
                        pagefourb();
                        break;
                       }
                case 4:{
                        signalb(count,hcount,rcount,stime,store);
                        getch();
                        pagefourb();
                        break;
                       }
                case 5:pagethreeb();
                        break;
                default :chooseb();
               }
        }

void wgraph()
        {
                setcolor(WHITE);
                line(20,15,20,475);
                line(20,245,620,245);
                line(15,15,20,15);
                line(15,61,20,61);
                line(15,107,20,107);
                line(15,153,20,153);
                line(15,199,20,199);
                line(15,245,20,245);
                line(15,291,20,291);
                line(15,337,20,337);
                line(15,383,20,383);
                line(15,429,20,429);
                line(15,475,20,475);
                settextstyle(0,0,1);
                outtextxy(5,13,"5");
                outtextxy(5,59,"4");
                outtextxy(5,105,"3");
                outtextxy(5,151,"2");
                outtextxy(5,197,"1");
                outtextxy(5,243,"0");
```

```
                outtextxy(0,289,"-1");
                outtextxy(0,335,"-2");
                outtextxy(0,381,"-3");
                outtextxy(0,427,"-4");
                outtextxy(0,472,"-5");
                outtextxy(0,0,"Volt(V)");
                outtextxy(625,245,"T");
                setcolor(7);
                outtextxy(22,12,"-----------------------------------------------------------");
                outtextxy(22,58,"-----------------------------------------------------------");
                outtextxy(22,104,"-----------------------------------------------------------");
                outtextxy(22,150,"-----------------------------------------------------------");
                outtextxy(22,196,"-----------------------------------------------------------");
                outtextxy(22,288,"-----------------------------------------------------------");
                outtextxy(22,334,"-----------------------------------------------------------");
                outtextxy(22,380,"-----------------------------------------------------------");
                outtextxy(22,426,"-----------------------------------------------------------");
                outtextxy(22,472,"-----------------------------------------------------------");
                settextstyle(0,1,1);
                outtextxy(125,14,"-----------------------------------------");
                outtextxy(225,14,"-----------------------------------------");
                outtextxy(325,14,"-----------------------------------------");
                outtextxy(425,14,"-----------------------------------------");
                outtextxy(525,14,"-----------------------------------------");
                outtextxy(625,14,"-----------------------------------------");
                setcolor(WHITE);
                settextstyle(0,0,1);
        }

void wgraph1()
                {
                        setcolor(WHITE);
                        line(20,15,20,475);
                        line(20,475,620,475);
                        line(15,15,20,15);
                        line(15,61,20,61);
                        line(15,107,20,107);
                        line(15,153,20,153);
                        line(15,199,20,199);
                        line(15,245,20,245);
                        line(15,291,20,291);
```

```
                line(15,337,20,337);
                line(15,383,20,383);
                line(15,429,20,429);
                line(15,475,20,475);
                outtextxy(5,13,"5");
                outtextxy(5,105,"4");
                outtextxy(5,197,"3");
                outtextxy(0,289,"2");
                outtextxy(0,381,"1");
                outtextxy(0,472,"0");
                outtextxy(0,0,"Volt(V)");
                outtextxy(625,472,"T");
                setcolor(7);
                outtextxy(22,12,"----------------------------------------------------
");
                outtextxy(22,58,"----------------------------------------------------
");
                outtextxy(22,104,"----------------------------------------------------
");
                outtextxy(22,150,"----------------------------------------------------
");
                outtextxy(22,196,"----------------------------------------------------
");
                outtextxy(22,243,"----------------------------------------------------
");
                outtextxy(22,288,"----------------------------------------------------
");
                outtextxy(22,334,"----------------------------------------------------
");
                outtextxy(22,380,"----------------------------------------------------
");
                outtextxy(22,426,"----------------------------------------------------
");
                settextstyle(0,1,1);
                outtextxy(125,14,"----------------------------------------");
                outtextxy(225,14,"----------------------------------------");
                outtextxy(325,14,"----------------------------------------");
                outtextxy(425,14,"----------------------------------------");
                outtextxy(525,14,"----------------------------------------");
                outtextxy(625,14,"----------------------------------------");
                setcolor(WHITE);
```

```
                    settextstyle(0,0,1);
                }

void discretea(int count,int hcount,int rcount,unsigned stime,int store[5010])
        {
                int x[5010],y[5010],tcount;
                float hexo,grapho;
                for(hcount = 0;hcount<count;hcount++)
                 {
                 hexo = ((0.039215686)*store[hcount]) - 5;
                 grapho =((-46)*hexo)+245 ;
                 rcount=rcount+(2*stime);
                 if(rcount>600) rcount=rcount-600;
                 y[hcount] = int(grapho);
                 x[hcount] = (rcount+20);
                 }
                wgraph();
                setcolor(14);
                tcount = 0;
                for(hcount = 0;hcount<count;hcount++)
                 {
                 if(x[hcount]<tcount)
                   {
                   setcolor(WHITE);
                   outtextxy(400,0,"Press Any Key To Continued");
                   getch();
                   cleardevice();
                   wgraph();
                   setcolor(14);
                   }
                 line(x[hcount],y[hcount],x[hcount],245);
                 tcount=x[hcount];
                 }
        }

void discreteb(int count,int hcount,int rcount,unsigned stime,int store[5010])
        {
                int x[5010],y[5010],tcount;
                float hexo,grapho;
                for(hcount = 0;hcount<count;hcount++)
                 {
```

```
        {
          if((x[hcount+1]>x[hcount])&&(x[hcount+1]!=620))
                {
                  setcolor(14);
                  line(x[hcount],y[hcount],x[hcount],245);
                  setcolor(4);
                  line(x[hcount],y[hcount],x[hcount+1],y[hcount]);
                  setcolor(2);
                  line(x[hcount+1],y[hcount],x[hcount+1],245);
                  setcolor(14);
                  line(x[hcount+1],y[hcount+1],x[hcount+1],245);

                }
          else if((x[hcount+1]>x[hcount])&&(x[hcount+1]==620))
                {
                  setcolor(14);
                  line(x[hcount],y[hcount],x[hcount],245);
                  setcolor(4);
                  line(x[hcount],y[hcount],x[hcount+1],y[hcount]);
                  setcolor(2);
                  line(x[hcount+1],y[hcount],x[hcount+1],245);
                  setcolor(14);
                  line(x[hcount+1],y[hcount+1],x[hcount+1],245);

                }
          else
                {
                  setcolor(4);
                  line(x[hcount],y[hcount],620,y[hcount]);
                  setcolor(WHITE);
                  outtextxy(400,0,"Press Any Key To Continue");
                  getch();
                  cleardevice();
                  wgraph();
                  setcolor(4);
                  line(20,y[hcount],x[hcount+1],y[hcount]);
                  setcolor(2);
                  line(x[hcount+1],y[hcount],x[hcount+1],245);
                }
        }
}
```

```
                hexo = ((0.01953125)*store[hcount]) ;
                grapho =((-92)*hexo)+475 ;
                rcount=rcount+(2*stime);
                if(rcount>600) rcount=rcount-600;
                y[hcount] = int(grapho);
                x[hcount] = (rcount+20);
                }
              wgraph1();
              setcolor(14);
              tcount = 0;
              for(hcount = 0;hcount<count;hcount++)
                {
                if(x[hcount]<tcount)
                  {
                  setcolor(WHITE);
                  outtextxy(400,0,"Press Any Key To Continued");
                  getch();
                  cleardevice();
                  wgraph1();
                  setcolor(14);
                  }
                line(x[hcount],y[hcount],x[hcount],475);
                tcount=x[hcount];
                }
        }

void holda(int count,int hcount,int rcount,unsigned stime,int store[5010])
        {
              int x[5010],y[5010];
              float hexo,grapho;
              for(hcount = 0;hcount<count;hcount++)
                {
                hexo = ((0.039215686)*store[hcount]) - 5;
                grapho =((-46)*hexo)+245 ;
                rcount=rcount+(2*stime);
                if(rcount>600) rcount=rcount-600;
                y[hcount] = int(grapho);
                x[hcount] = (rcount+20);
                }
              wgraph();
              for(hcount = 0;hcount<count-1;hcount++)
```

```
void holdb(int count,int hcount,int rcount,unsigned stime,int store[5010])
        {
                int x[5010],y[5010];
                float hexo,grapho;
                for(hcount = 0;hcount<count;hcount++)
                 {
                 hexo = ((0.01953125)*store[hcount]);
                 grapho =((-92)*hexo)+475 ;
                 rcount=rcount+(2*stime);
                 if(rcount>600) rcount=rcount-600;
                 y[hcount] = int(grapho);
                 x[hcount] = (rcount+20);
                 }
                wgraph1();
                for(hcount = 0;hcount<count-1;hcount++)
                 {
                  if((x[hcount+1]>x[hcount])&&(x[hcount+1]!=620))
                        {
                          setcolor(14);
                          line(x[hcount],y[hcount],x[hcount],475);
                          setcolor(4);
                          line(x[hcount],y[hcount],x[hcount+1],y[hcount]);
                          setcolor(2);
                          line(x[hcount+1],y[hcount],x[hcount+1],475);
                          setcolor(14);
                          line(x[hcount+1],y[hcount+1],x[hcount+1],475);

                        }
                   else if((x[hcount+1]>x[hcount])&&(x[hcount+1]==620))
                        {
                          setcolor(14);
                          line(x[hcount],y[hcount],x[hcount],475);
                          setcolor(4);
                          line(x[hcount],y[hcount],x[hcount+1],y[hcount]);
                          setcolor(2);
                          line(x[hcount+1],y[hcount],x[hcount+1],475);
                          setcolor(14);
                          line(x[hcount+1],y[hcount+1],x[hcount+1],475);

                        }
```

```
                else
                        {
                          setcolor(4);
                          line(x[hcount],y[hcount],620,y[hcount]);
                          setcolor(WHITE);
                          outtextxy(400,0,"Press Any Key To Continue");
                          getch();
                          cleardevice();
                          wgraph1();
                          setcolor(4);
                          line(20,y[hcount],x[hcount+1],y[hcount]);
                          setcolor(2);
                          line(x[hcount+1],y[hcount],x[hcount+1],475);
                        }
                }
        }

void table(void)
        {
                line(20,10,20,455);
                line(120,10,120,455);
                line(220,10,220,455);
                line(320,10,320,455);
                line(20,10,320,10);
                line(20,35,320,35);
                line(20,55,320,55);
                line(20,75,320,75);
                line(20,95,320,95);
                line(20,115,320,115);
                line(20,135,320,135);
                line(20,155,320,155);
                line(20,175,320,175);
                line(20,195,320,195);
                line(20,215,320,215);
                line(20,235,320,235);
                line(20,255,320,255);
                line(20,275,320,275);
                line(20,295,320,295);
                line(20,315,320,315);
                line(20,335,320,335);
                line(20,355,320,355);
```

```
                line(20,375,320,375);
                line(20,395,320,395);
                line(20,415,320,415);
                line(20,435,320,435);
                line(20,455,320,455);
                outtextxy(42,23,"Time(ms)");
                outtextxy(140,23,"Hex Code");
                outtextxy(245,23,"Volt(v)");
        }

void dataa (int count,int store[5010],int hcount,int stime)
        {
                char word[10],number[10],atime[10];
                unsigned j,btime;
                float num;
                table();
                j = 0;
                btime=0;
                for(hcount=0;hcount<count;hcount++)
                 {
                 if(j>20)
                  {
                  j=0;
                  outtextxy(400,0,"Press Any Key To Continue");
                  getch();
                  cleardevice ();
                  table();
                  }
                 num= (((0.039215686)*store[hcount])-5);
                 sprintf(word,"%x",store[hcount]);
                 sprintf(number,"%5.3f",num);
                 btime = (hcount*stime);
                 sprintf(atime,"%u",btime);
                 outtextxy(45,((20*j)+45),atime);
                 outtextxy(145,((20*j)+45),word);
                 outtextxy(245,((20*j)+45),number);
                j = j+1;
                }
        }

void datab (int count,int store[5010],int hcount,int stime)
```

```
{
        char word[10],number[10],atime[10];
        unsigned j,btime;
        float num;
        table();
        j = 0;
        btime=0;
        for(hcount=0;hcount<count;hcount++)
         {
         if(j>20)
          {
          j=0;
          outtextxy(400,0,"Press Any Key To Continue");
          getch();
          cleardevice ();
          table();
          }
         num= ((0.01953125)*store[hcount]);
         sprintf(word,"%x",store[hcount]);
         sprintf(number,"%5.3f",num);
         btime = (hcount*stime);
         sprintf(atime,"%u",btime);
         outtextxy(45,((20*j)+45),atime);
         outtextxy(145,((20*j)+45),word);
         outtextxy(245,((20*j)+45),number);
         j = j+1;
         }
}
void signala(int count,int hcount,int rcount,unsigned stime,int store[5010])
{
        int x[5010],y[5010],delx,dely,xmax,xmin,ymax,ymin;
        float hexo,grapho,divx,yeqx,yeqn;
        for(hcount = 0;hcount<count;hcount++)
         {
         hexo = ((0.039215686)*store[hcount]) - 5;
         grapho =((-46)*hexo)+245 ;
         rcount=rcount+(2*stime);
         if(rcount>600) rcount=rcount-600;
         y[hcount] = int(grapho);
         x[hcount] = (rcount+20);
         }
```

```
                wgraph();
                setcolor(14);
                for(hcount = 0;hcount<count-1;hcount++)
                 {
                  if (x[hcount+1]>x[hcount])
                  {
                   line(x[hcount],y[hcount],x[hcount+1],y[hcount+1]);
                  }
                  else
                  {
                   delx=stime;
                   dely=abs(y[hcount+1]-y[hcount]);
                   divx=(dely/delx);
                   yeqx=((620-x[hcount])*divx);
                   if(y[hcount+1]<=y[hcount])
                    {
                          ymax=int(y[hcount]+yeqx);
                    }
                   else
                   {
                          ymax=int(y[hcount]-yeqx);
                    }
                   line(x[hcount],y[hcount],620,ymax);
                   setcolor(WHITE);
                   outtextxy(400,0,"Press Any Key To Continue");
                   getch();
                   cleardevice();
                   wgraph();
                   setcolor(14);
                   line(20,ymax,x[hcount+1],y[hcount+1]);
                   line(x[hcount+1],y[hcount+1],x[hcount+2],y[hcount+2]);
                  }
                 }
        }

void signalb(int count,int hcount,int rcount,unsigned stime,int store[5010])
        {
                int x[5010],y[5010],delx,dely,xmax,xmin,ymax,ymin;
                float hexo,grapho,divx,yeqx,yeqn;
                for(hcount = 0;hcount<count;hcount++)
                 {
```

```
                hexo = ((0.01953125)*store[hcount]);
                grapho =((-92)*hexo)+475 ;
                rcount=rcount+(2*stime);
                if(rcount>600) rcount=rcount-600;
                y[hcount] = int(grapho);
                x[hcount] = (rcount+20);
                }
               wgraph1();
               setcolor(14);
               for(hcount = 0;hcount<count-1;hcount++)
                {
                if (x[hcount+1]>x[hcount])
                 {
                 line(x[hcount],y[hcount],x[hcount+1],y[hcount+1]);
                 }
                else
                 {
                 delx=stime;
                 dely=abs(y[hcount+1]-y[hcount]);
                 divx=(dely/delx);
                 yeqx=((620-x[hcount])*divx);
                 if(y[hcount+1]<=y[hcount])
                  {
                        ymax=int(y[hcount]+yeqx);
                  }
                 else
                  {
                        ymax=int(y[hcount]-yeqx);
                  }
                 line(x[hcount],y[hcount],620,ymax);
                 setcolor(WHITE);
                 outtextxy(400,0,"Press Any Key To Continue");
                 getch();
                 cleardevice();
                 wgraph1();
                 setcolor(14);
                 line(20,ymax,x[hcount+1],y[hcount+1]);
                 line(x[hcount+1],y[hcount+1],x[hcount+2],y[hcount+2]);
                 }
                }
}
```

```
void pageone()
        {
                char a;
                escmenu();
                about();
                while((a!='\r')&&(a!=27)) a=getch();
                if(a =='\r')
                 {
                  pagetwo();
                 }
                else if(a == 27)
                 {
                  closegraph();
                  exit(0);
                 }
        }

void pagetwo()
        {
                range();
        }

void pagethreea()
        {
                menu();
                tinputa();
                /*calculate count*/
                count =int((ftime/stime));
                count =count +1;
                /*sampling store and convert to graph to array*/
                rcount=-(2*stime);
                outport(0x303,0x82);
                for(hcount = 0;hcount<count;hcount++)
                 {
                  delay(stime);
                  outport(0x302,0x00);
                  outportb(0x302,0x08);
                  store[hcount]=inp(0x301);
                  outport(0x300,store[hcount]);
                 }
                pagefoura();
```

```
}

void pagethreeb()
{
    menu();
    tinputb();
    /*calculate count*/
    count =int((ftime/stime));
    count =count +1;
    /*sampling store and convert to graph to array*/
    rcount=-(2*stime);
    outport(0x303,0x82);
    for(hcount = 0;hcount<count;hcount++)
     {
      delay(stime);
      outport(0x302,0x00);
      outportb(0x302,0x08);
      store[hcount]=inp(0x301);
      outport(0x300,store[hcount]);

     }
    pagefourb();
}

void pagefoura()
{
    choosea();
}

void pagefourb()
{
    chooseb();
}
```

ภาคผนวก ข

Data Sheet

National Semiconductor

ADC0801/ADC0802/ADC0803/ADC0804/ADC0805 8-Bit μP Compatible A/D Converters

General Description

The ADC0801, ADC0802, ADC0803, ADC0804 and ADC0805 are CMOS 8-bit successive approximation A/D converters that use a differential potentiometric ladder—similar to the 256R products. These converters are designed to allow operation with the NSC800 and INS8080A derivative control bus with TRI-STATE® output latches directly driving the data bus. These A/Ds appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed.

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

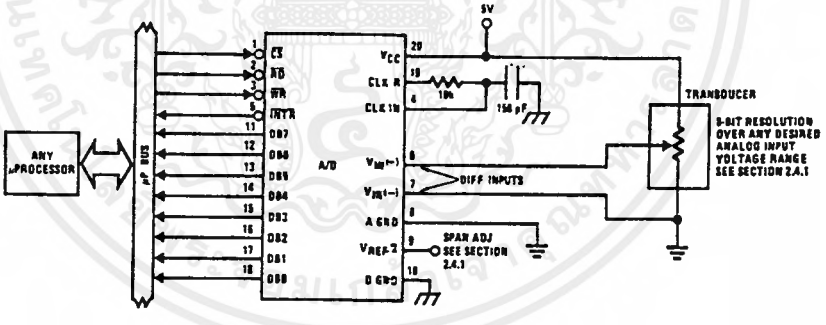
Features

- Compatible with 8080 μP derivatives—no interfacing logic needed - access time - 135 ns
- Easy interface to all microprocessors, or operates "stand alone"
- Differential analog voltage inputs
- Logic inputs and outputs meet both MOS and TTL voltage level specifications
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3" standard width 20-pin DIP package
- 20-pin molded chip carrier or small outline package
- Operates ratiometrically or with 5 V_{DC}, 2.5 V_{DC}, or analog span adjusted voltage reference

Key Specifications

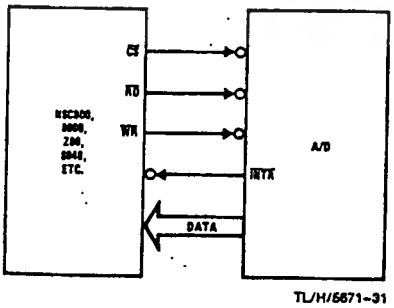
- Resolution — 8 bits
- Total error — ±¼ LSB, ±½ LSB and ±1 LSB
- Conversion time — 100 μs

Typical Applications



TL/H/5671–1

8080 Interface



TL/H/5671–31

Error Specification (Includes Full-Scale, Zero Error, and Non-Linearity)			
Part Number	Full-Scale Adjusted	$V_{REF}/2 = 2.500\ V_{DC}$ (No Adjustments)	$V_{REF}/2$ = No Connection (No Adjustments)
ADC0801	±¼ LSB		
ADC0802		±½ LSB	
ADC0803	±½ LSB		
ADC0804		±1 LSB	
ADC0805			±1 LSB

Absolute Maximum Ratings (Notes 1 & 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC}) (Note 3)	6.5V
Voltage	
Logic Control Inputs	−0.3V to +18V
At Other Input and Outputs	−0.3V to (V_{CC} + 0.3V)
Lead Temp. (Soldering, 10 seconds)	
Dual-In-Line Package (plastic)	260°C
Dual-In-Line Package (ceramic)	300°C
Surface Mount Package	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C
Storage Temperature Range	−65°C to +150°C
Package Dissipation at T_A = 25°C	875 mW
ESD Susceptibility (Note 10)	800V

Operating Ratings (Notes 1 & 2)

Temperature Range	$T_{MIN} \le T_A \le T_{MAX}$
ADC0801/02LJ, ADC0802LJ/883	$-55°C \le T_A \le +125°C$
ADC0801/02/03/04LCJ	$-40°C \le T_A \le +85°C$
ADC0801/02/03/05LCN	$-40°C \le T_A \le +85°C$
ADC0804LCN	$0°C \le T_A \le +70°C$
ADC0802/03/04LCV	$0°C \le T_A \le +70°C$
ADC0802/03/04LCWM	$0°C \le T_A \le +70°C$
Range of V_{CC}	4.5 V_{DC} to 6.3 V_{DC}

Electrical Characteristics

The following specifications apply for V_{CC} = 5 V_{DC}, $T_{MIN} \le T_A \le T_{MAX}$ and f_{CLK} = 640 kHz unless otherwise specified.

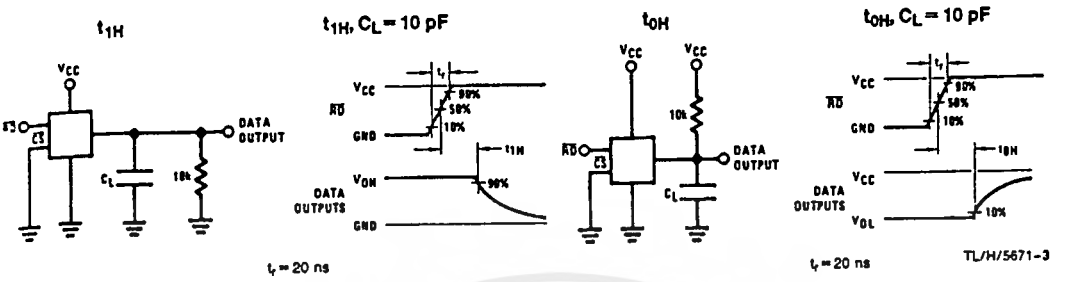
Parameter	Conditions	Min	Typ	Max	Units
ADC0801: Total Adjusted Error (Note 8)	With Full-Scale Adj. (See Section 2.5.2)			±¼	LSB
ADC0802: Total Unadjusted Error (Note 8)	$V_{REF}/2$ = 2.500 V_{DC}			±½	LSB
ADC0803: Total Adjusted Error (Note 8)	With Full-Scale Adj. (See Section 2.5.2)			±½	LSB
ADC0804: Total Unadjusted Error (Note 8)	$V_{REF}/2$ = 2.500 V_{DC}			±1	LSB
ADC0805: Total Unadjusted Error (Note 8)	$V_{REF}/2$-No Connection			±1	LSB
$V_{REF}/2$ Input Resistance (Pin 9)	ADC0801/02/03/05 ADC0804 (Note 9)	2.5 0.75	8.0 1.1		kΩ kΩ
Analog Input Voltage Range	(Note 4) V(+) or V(−)	Gnd−0.05		V_{CC} + 0.05	V_{DC}
DC Common-Mode Error	Over Analog Input Voltage Range		±1/16	±⅛	LSB
Power Supply Sensitivity	V_{CC} = 5 V_{DC} ±10% Over Allowed V_{IN}(+) and V_{IN}(−) Voltage Range (Note 4)		±1/16	±⅛	LSB

AC Electrical Characteristics

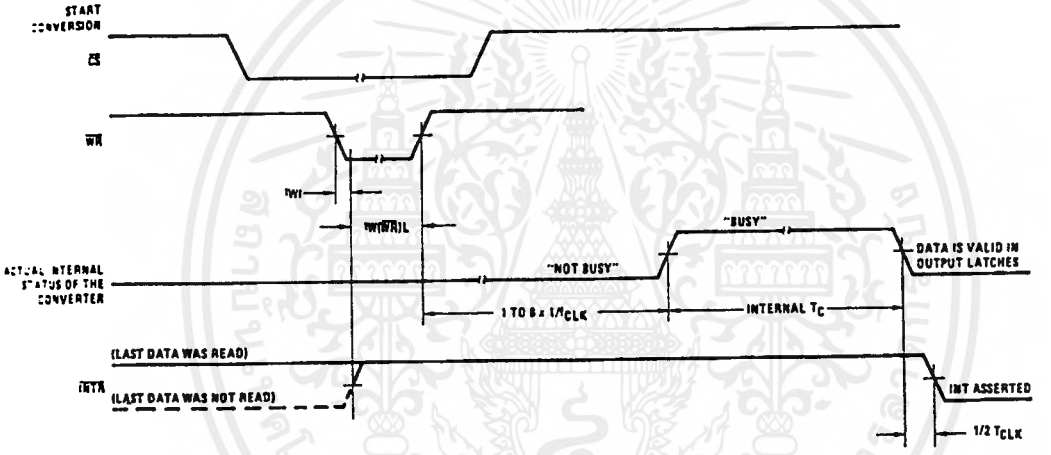
The following specifications apply for V_{CC} = 5 V_{DC} and T_A = 25°C unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
T_C	Conversion Time	f_{CLK} = 640 kHz (Note 6)	103		114	μs
T_C	Conversion Time	(Note 5, 6)	66		73	$1/f_{CLK}$
f_{CLK}	Clock Frequency Clock Duty Cycle	V_{CC} = 5V, (Note 5) (Note 5)	100 40	640	1460 60	kHz %
CR	Conversion Rate in Free-Running Mode	$\overline{INTR}$ tied to $\overline{WR}$ with $\overline{CS}$ = 0 V_{DC}, f_{CLK} = 640 kHz	8770		9708	conv/s
$t_{W(\overline{WR})L}$	Width of $\overline{WR}$ Input (Start Pulse Width)	$\overline{CS}$ = 0 V_{DC} (Note 7)	100			ns
t_{ACC}	Access Time (Delay from Falling Edge of $\overline{RD}$ to Output Data Valid)	C_L = 100 pF		135	200	ns
t_{1H}, t_{0H}	TRI-STATE Control (Delay from Rising Edge of $\overline{RD}$ to Hi-Z State)	C_L = 10 pF, R_L = 10k (See TRI-STATE Test Circuits)		125	200	ns
t_{WI}, t_{RI}	Delay from Falling Edge of $\overline{WR}$ or $\overline{RD}$ to Reset of $\overline{INTR}$			300	450	ns
C_{IN}	Input Capacitance of Logic Control Inputs			5	7.5	pF
C_{OUT}	TRI-STATE Output Capacitance (Data Buffers)			5	7.5	pF
CONTROL INPUTS [Note: CLK IN (Pin 4) is the input of a Schmitt trigger circuit and is therefore specified separately]						
V_{IN} (1)	Logical "1" Input Voltage (Except Pin 4 CLK IN)	V_{CC} = 5.25 V_{DC}	2.0		15	V_{DC}

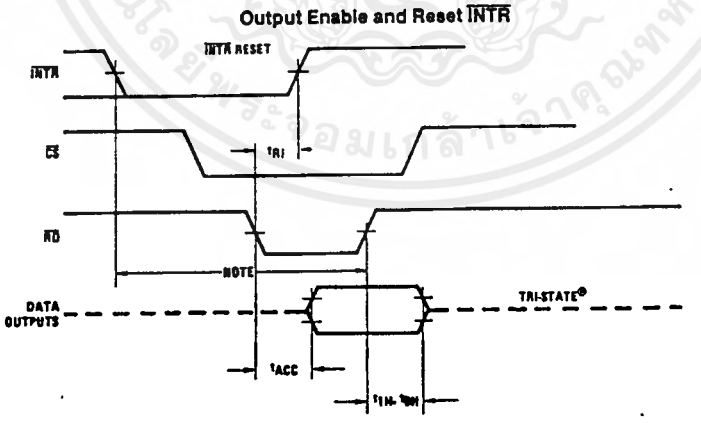
TRI-STATE Test Circuits and Waveforms



Timing Diagrams (All timing is measured from the 50% voltage points)



Output Enable and Reset $\overline{INTR}$



TL/H/5671–4

Note: Read strobe must occur 8 clock periods ($8/f_{CLK}$) after assertion of interrupt to guarantee reset of $\overline{INTR}$.

Functional Description (Continued)

A single point analog ground that is separate from the logic ground points should be used. The power supply bypass capacitor and the self-clocking capacitor (if used) should both be returned to digital ground. Any $V_{REF}/2$ bypass capacitors, analog input filter capacitors, or input signal shielding should be returned to the analog ground point. A test for proper grounding is to measure the zero error of the A/D converter. Zero errors in excess of ¼ LSB can usually be traced to improper board layout and wiring (see section 2 5.1 for measuring the zero error).

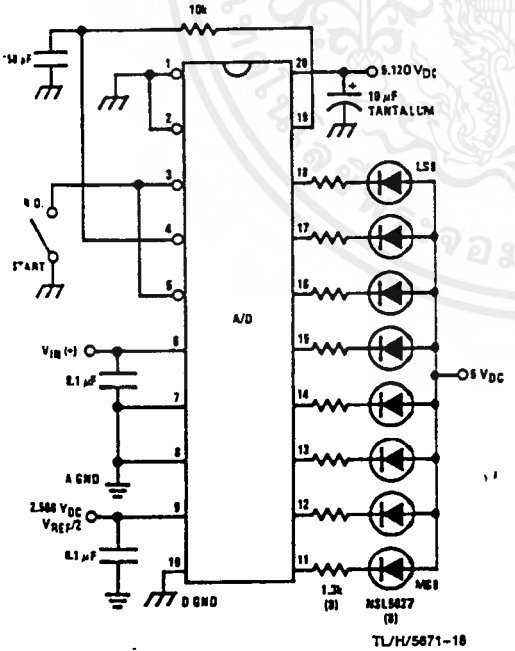
3.0 TESTING THE A/D CONVERTER

There are many degrees of complexity associated with testing an A/D converter. One of the simplest tests is to apply a known analog input voltage to the converter and use LEDs to display the resulting digital output code as shown in *Figure 7*.

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

If a full-scale adjustment is to be made, an analog input voltage of 5.090 V_{DC} (5.120−1½ LSB) should be applied to the $V_{IN}(+)$ pin with the $V_{IN}(-)$ pin grounded. The value of the $V_{REF}/2$ input voltage should then be adjusted until the digital output code is just changing from 1111 1110 to 1111 1111. This value of $V_{REF}/2$ should then be used for all the tests.

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



TL/H/5671–18

FIGURE 7. Basic A/D Tester

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

A basic A/D tester that uses a DAC and provides the error as an analog output voltage is shown in *Figure 8*. The 2 op amps can be eliminated if a lab DVM with a numerical subtraction feature is available to read the difference voltage, "A−C", directly. The analog input voltage can be supplied by a low frequency ramp generator and an X-Y plotter can be used to provide analog error (Y axis) versus analog input (X axis).

For operation with a microprocessor or a computer-based test system, it is more convenient to present the errors digitally. This can be done with the circuit of *Figure 9*, where the output code transitions can be detected as the 10-bit DAC is incremented. This provides ¼ LSB steps for the 8-bit A/D under test. If the results of this test are automatically plotted with the analog input on the X axis and the error (in LSB's) as the Y axis, a useful transfer function of the A/D under test results. For acceptance testing, the plot is not necessary and the testing speed can be increased by establishing internal limits on the allowed error for each code.

4.0 MICROPROCESSOR INTERFACING

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

4.1 Interfacing 8080 Microprocessor Derivatives (8048, 8085)

This converter has been designed to directly interface with derivatives of the 8080 microprocessor. The A/D can be mapped into memory space (using standard memory address decoding for $\overline{CS}$ and the $\overline{MEMR}$ and $\overline{MEMW}$ strobes) or it can be controlled as an I/O device by using the $\overline{I/O\ R}$ and $\overline{I/O\ W}$ strobes and decoding the address bits A0 → A7 (or address bits A8 → A15 as they will contain the same 8-bit address information) to obtain the $\overline{CS}$ input. Using the I/O space provides 256 additional addresses and may allow a simpler 8-bit address decoder but the data can only be input to the accumulator. To make use of the additional memory reference instructions, the A/D should be mapped into memory space. An example of an A/D in I/O space is shown in *Figure 10*.

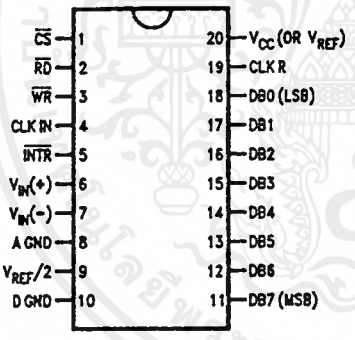
Ordering Information

TEMP RANGE		0°C TO 70°C	0°C TO 70°C	0°C TO 70°C	−40°C TO +85°C
ERROR	±¼ Bit Adjusted				ADC0801LCN
	±½ Bit Unadjusted	ADC0802LCWM	ADC0802LCV		ADC0802LCN
	±½ Bit Adjusted	ADC0803LCWM	ADC0803LCV		ADC0803LCN
	±1Bit Unadjusted	ADC0804LCWM	ADC0804LCV	ADC0804LCN	ADC0805LCN
PACKAGE OUTLINE		M20B—Small Outline	V20A—Chip Carrier	N20A—Molded DIP	

TEMP RANGE		−40°C TO +85°C	−55°C TO +125°C
ERROR	±¼ Bit Adjusted	ADC0801LCJ	ADC0801LJ
	±½ Bit Unadjusted	ADC0802LCJ	ADC0802LJ,
	±½ Bit Adjusted	ADC0803LCJ	ADC0802LJ/883
	±1Bit Unadjusted	ADC0804LCJ	
PACKAGE OUTLINE		J20A—Cavity DIP	J20A—Cavity DIP

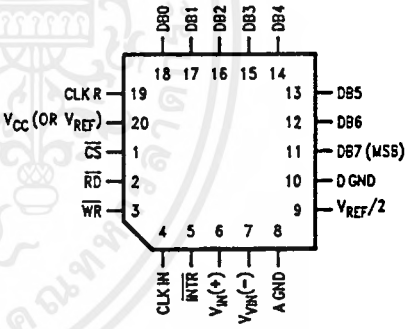
Connection Diagrams

ADC080X
Dual-In-Line and Small Outline (SO) Packages



TL/H/5671–30

ADC080X
Molded Chip Carrier (PCC) Package



TL/H/5671–31

See Ordering Information

National Semiconductor

DAC0830/DAC0831/DAC0832 8-Bit μP Compatible, Double-Buffered D to A Converters

General Description

The DAC0830 is an advanced CMOS/Si-Cr 8-bit multiplying DAC designed to interface directly with the 8080, 8048, 8085, Z80®, and other popular microprocessors. A deposited silicon-chromium R-2R resistor ladder network divides the reference current and provides the circuit with excellent temperature tracking characteristics (0.05% of Full Scale Range maximum linearity error over temperature). The circuit uses CMOS current switches and control logic to achieve low power consumption and low output leakage current errors. Special circuitry provides TTL logic input voltage level compatibility.

Double buffering allows these DACs to output a voltage corresponding to one digital word while holding the next digital word. This permits the simultaneous updating of any number of DACs.

The DAC0830 series are the 8-bit members of a family of microprocessor-compatible DACs (MICRO-DAC™). For applications demanding higher resolution, the DAC1000 series (10-bits) and the DAC1208 and DAC1230 (12-bits) are available alternatives.

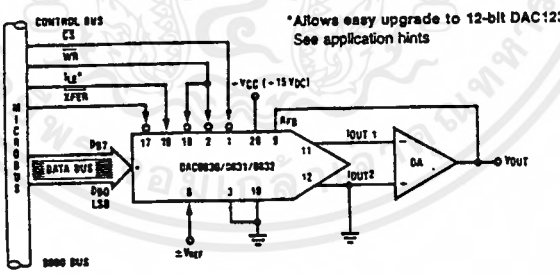
Features

- Double-buffered, single-buffered or flow-through digital data inputs
- Easy interchange and pin-compatible with 12-bit DAC1230 series
- Direct interface to all popular microprocessors
- Linearity specified with zero and full-scale adjust only—NOT BEST STRAIGHT LINE FIT.
- Works with ±10V reference-full 4-quadrant multiplication
- Can be used in the voltage switching mode
- Logic inputs which meet TTL voltage level specs (1.4V logic threshold)
- Operates "STAND ALONE" (without μP) if desired
- Available in 20-pin small-outline or molded chip carrier package

Key Specifications

■ Current settling time	1 μs
■ Resolution	8 bits
■ Linearity (guaranteed over temp.)	8, 9, or 10 bits
■ Gain Tempco	0.0002% FS/°C
■ Low power dissipation	20 mW
■ Single power supply	5 to 15 V_{DC}

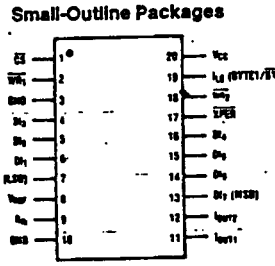
Typical Application



TL/H/5608–1

Connection Diagrams (Top Views)

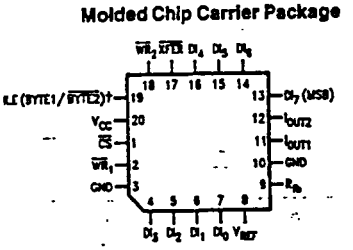
Dual-In-Line and Small-Outline Packages



†This is necessary for the 12-bit DAC1230 series to permit interchanging from an 8-bit to a 12-bit DAC with No PC board changes and no software changes. See applications section.

TL/H/5608–21

Molded Chip Carrier Package



TL/H/5608–22

Absolute Maximum Ratings (Notes 1 & 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	17 V_{DC}
Voltage at Any Digital Input	V_{CC} to GND
Voltage at V_{REF} Input	±25V
Storage Temperature Range	−65°C to +150°C
Package Dissipation at T_A = 25°C (Note 3)	500 mW
DC Voltage Applied to I_{OUT1} or I_{OUT2} (Note 4)	−100 mV to V_{CC}
ESD Susceptability (Note 14)	800V
Lead Temperature (soldering, 10 sec.)	
Dual-In-Line Package (plastic)	260°C
Dual-In-Line Package (ceramic)	300°C
Surface Mount Package	
Vapor Phase (60 sec.)	215°C
Infrared (15 sec.)	220°C

Operating Conditions

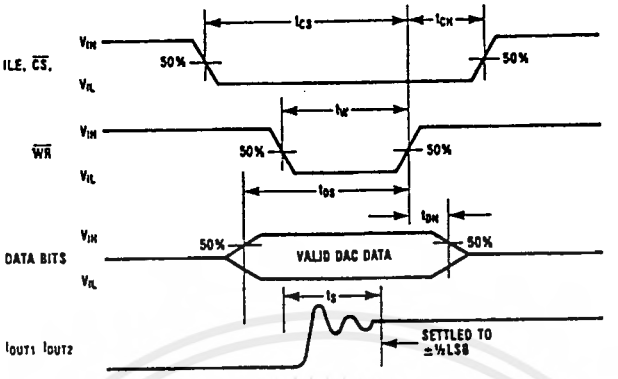
Temperature Range	$T_{MIN} \le T_A \le T_{MAX}$
Part numbers with 'LCN' suffix	0°C to +70°C
Part numbers with 'LCWM' suffix	0°C to +70°C
Part numbers with 'LCV' suffix	0°C to +70°C
Part numbers with 'LCJ' suffix	−40°C to +85°C
Part numbers with 'LJ' suffix	−55°C to +125°C
Voltage at Any Digital Input	V_{CC} to GND

Electrical Characteristics

V_{REF} = 10.000 V_{DC} unless otherwise noted. **Boldface limits apply over temperature, $T_{MIN} \le T_A \le T_{MAX}$.** For all other limits T_A = 25°C.

Parameter	Conditions	See Note	V_{CC} = 4.75 V_{DC} and V_{CC} = 15.75 V_{DC}		V_{CC} = 5 V_{DC} ±5% and V_{CC} = 12 V_{DC} ±5% to 15 V_{DC} ±5%	Limit Units
			Typ (Note 12)	Tested Limit (Note 5)	Design Limit (Note 6)	
CONVERTER CHARACTERISTICS						
Resolution			8	8	8	bits
Linearity Error Max	Zero and full scale adjusted −10V ≤ V_{REF} ≤ +10V	4, 8				
DAC0830LJ & LCJ				**0.05**	**0.05**	% FSR
DAC0832LJ & LCJ				**0.2**	**0.2**	% FSR
DAC0830LCN, LCWM & LCV				0.05	**0.05**	% FSR
DAC0831LCN				0.1	**0.1**	% FSR
DAC0832LCN, LCWM & LCV				0.2	**0.2**	% FSR
Differential Nonlinearity Max	Zero and full scale adjusted −10V ≤ V_{REF} ≤ +10V	4, 8				
DAC0830LJ & LCJ				**0.1**	**0.1**	% FSR
DAC0832LJ & LCJ				**0.4**	**0.4**	% FSR
DAC0830LCN, LCWM & LCV				0.1	**0.1**	% FSR
DAC0831LCN				0.2	**0.2**	% FSR
DAC0832LCN, LCWM & LCV				0.4	**0.4**	% FSR
Monotonicity	−10V ≤ V_{REF} ≤ +10V LJ & LCJ	4		**8**	**8**	bits
	−10V ≤ V_{REF} ≤ +10V LCN, LCWM & LCV			8	8	bits
Gain Error Max	Using Internal R_{fb} −10V ≤ V_{REF} ≤ +10V	7	±0.2	±1	±1	% FS
Gain Error Tempco Max	Using internal R_{fb}		**0.0002**		**0.0006**	% FS/°C
Power Supply Rejection	All digital inputs latched high V_{CC} = 14.5V to 15.5V		0.0002	0.0025		% FSR/
	11.5V to 12.5V		0.0006			
	4.5V to 5.5V		0.013	0.015		
Reference Input Max			**15**	**20**	**20**	kΩ
Reference Input Min			**15**	**10**	**10**	kΩ
Output Feedthrough Error	V_{REF} = 20 Vp-p, f = 100 kHz All data inputs latched low		3			mVp-

Switching Waveform



TL/H/5608–2

Definition of Package Pinouts

Control Signals (All control signals level actuated)

$\overline{CS}$: **Chip Select** (active low). The $\overline{CS}$ in combination with ILE will enable $\overline{WR_1}$.

ILE: **Input Latch Enable** (active high). The ILE in combination with $\overline{CS}$ enables $\overline{WR_1}$.

$\overline{WR_1}$: **Write 1.** The active low $\overline{WR_1}$ is used to load the digital input data bits (DI) into the input latch. The data in the input latch is latched when $\overline{WR_1}$ is high. To update the input latch—$\overline{CS}$ and $\overline{WR_1}$ must be low while ILE is high.

$\overline{WR_2}$: **Write 2** (active low). This signal, in combination with $\overline{XFER}$, causes the 8-bit data which is available in the input latch to transfer to the DAC register.

$\overline{XFER}$: **Transfer control signal** (active low). The $\overline{XFER}$ will enable $\overline{WR_2}$.

Other Pin Functions

DI_0-DI_7: **Digital Inputs.** DI_0 is the least significant bit (LSB) and DI_7 is the most significant bit (MSB).

I_{OUT1}: **DAC Current Output 1.** I_{OUT1} is a maximum for a digital code of all 1's in the DAC register, and is zero for all 0's in DAC register.

I_{OUT2}: **DAC Current Output 2.** I_{OUT2} is a constant minus I_{OUT1}, or $I_{OUT1} + I_{OUT2}$ = constant (I full scale for a fixed reference voltage).

R_{fb}: **Feedback Resistor.** The feedback resistor is provided on the IC chip for use as the shunt feedback resistor for the external op amp which is used to provide an output voltage for the DAC. This on-chip resistor should always be used (not an external resistor) since it matches the resistors which are used in the on-chip R-2R ladder and tracks these resistors over temperature.

V_{REF}: **Reference Voltage Input.** This input connects an external precision voltage source to the internal R-2R ladder. V_{REF} can be selected over the range of +10 to −10V. This is also the analog voltage input for a 4-quadrant multiplying DAC application.

V_{CC}: **Digital Supply Voltage.** This is the power supply pin for the part. V_{CC} can be from +5 to +15V_{DC}. Operation is optimum for +15V_{DC}.

GND: The pin 10 voltage must be at the same ground potential as I_{OUT1} and I_{OUT2} for current switching applications. Any difference of potential (V_{OS} pin 10) will result in a linearity change of

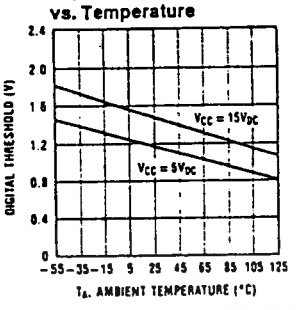
$$\frac{V_{OS} \text{ pin } 10}{3V_{REF}}$$

For example, if V_{REF} = 10V and pin 10 is 9mV offset from I_{OUT1} and I_{OUT2} the linearity change will be 0.03%.

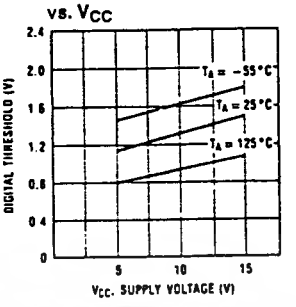
Pin 3 can be offset ±100mV with no linearity change, but the logic input threshold will shift.

Typical Performance Characteristics

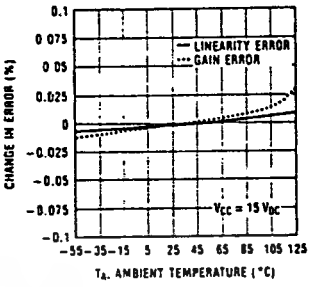
Digital Input Threshold vs. Temperature



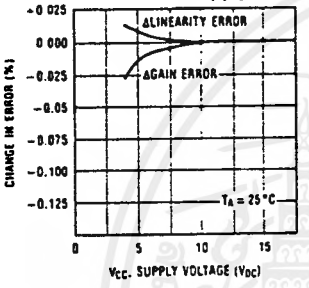
Digital Input Threshold vs. V_{CC}



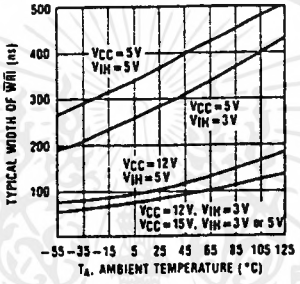
Gain and Linearity Error Variation vs. Temperature



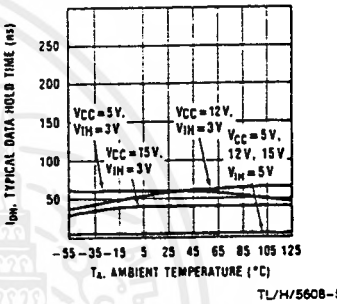
Gain and Linearity Error Variation vs. Supply Voltage



Write Pulse Width



Data Hold Time



TL/H/5608–5

DAC0830 Series Application Hints

These DAC's are the industry's first microprocessor compatible, double-buffered 8-bit multiplying D to A converters. Double-buffering allows the utmost application flexibility from a digital control point of view. This 20-pin device is also pin for pin compatible (with one exception) with the DAC1230, a 12-bit MICRO-DAC. In the event that a system's analog output resolution and accuracy must be upgraded, substituting the DAC1230 can be easily accomplished. By tying address bit A_0 to the ILE pin, a two-byte µP write instruction (double precision) which automatically increments the address for the second byte write (starting with A_0 = "1") can be used. This allows either an 8-bit or the 12-bit part to be used with no hardware or software changes. For the simplest 8-bit application, this pin should be tied to V_{CC} (also see other uses in section 1.1).

Analog signal control versatility is provided by a precision R-2R ladder network which allows full 4-quadrant multiplication of a wide range bipolar reference voltage by an applied digital word.

1.0 DIGITAL CONSIDERATIONS

A most unique characteristic of these DAC's is that the 8-bit digital input byte is double-buffered. This means that the data must transfer through two independently controlled 8-bit latching registers before being applied to the R-2R ladder network to change the analog output. The addition of a second register allows two useful control features. First, any DAC in a system can simultaneously hold the current DAC data in one register (DAC register) and the next data word in the second register (input register) to allow fast updating of the DAC output on demand. Second, and probably more important, double-buffering allows any number of DAC's in a system to be updated to their new analog output levels simultaneously via a common strobe signal.

The timing requirements and logic level convention of the register control signals have been designed to minimize or eliminate external interfacing logic when applied to most popular microprocessors and development systems. It is easy to think of these converters as 8-bit "write-only" memory locations that provide an analog output quantity. All inputs to these DAC's meet TTL voltage level specs and can also be driven directly with high voltage CMOS logic in non-microprocessor based systems. To prevent damage to the chip from static discharge, all unused digital inputs should be tied to V_{CC} or ground. If any of the digital inputs are inadvertantly left floating, the DAC interprets the pin as a logic "1".

1.1 Double-Buffered Operation

Updating the analog output of these DAC's in a double-buffered manner is basically a two step or double write operation. In a microprocessor system two unique system addresses must be decoded, one for the input latch controlled by the $\overline{CS}$ pin and a second for the DAC latch which is controlled by the $\overline{XFER}$ line. If more than one DAC is being driven, *Figure 2*, the $\overline{CS}$ line of each DAC would typically be decoded individually, but all of the converters could share a common $\overline{XFER}$ address to allow simultaneous updating of any number of DAC's. The timing for this operation is shown, *Figure 3*.

It is important to note that the analog outputs that will change after a simultaneous transfer are those from the DAC's whose input register had been modified prior to the $\overline{XFER}$ command.

DAC0830 Series Application Hints (Continued)

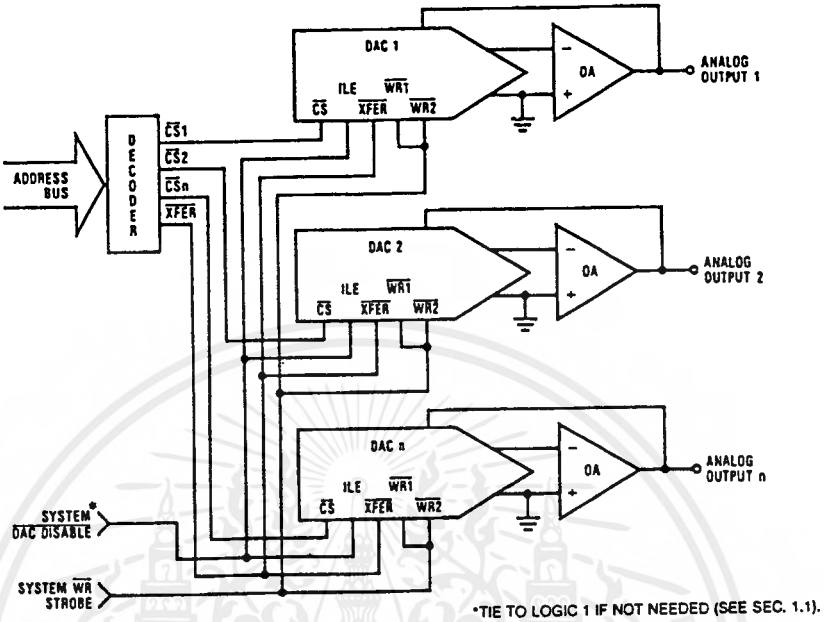


FIGURE 2. Controlling Mutiple DACs

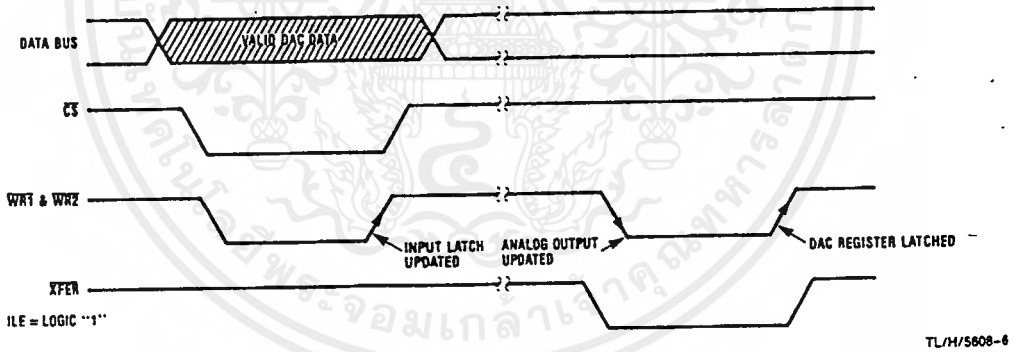


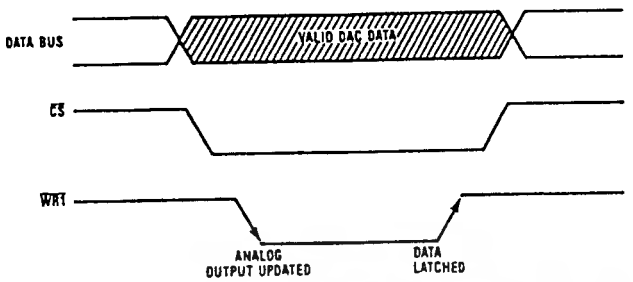
FIGURE 3

The ILE pin is an active high chip select which can be decoded from the address bus as a qualifier for the normal $\overline{CS}$ signal generated during a write operation. This can be used to provide a higher degree of decoding unique control signals for a particular DAC, and thereby create a more efficient addressing scheme.

Another useful application of the ILE pin of each DAC in a multiple DAC system is to tie these inputs together and use this as a control line that can effectively "freeze" the outputs of all the DAC's at their present value. Pulling this line low latches the input register and prevents new data from being written to the DAC. This can be particularly useful in multiprocessing systems to allow a processor other than the one controlling the DAC's to take over control of the data bus and control lines. If this second system were to use the same addresses as those decoded for DAC control (but for a different purpose) the ILE function would prevent the DAC's from being erroneously altered.

In a "Stand-Alone" system the control signals are generated by discrete logic. In this case double-buffering can be controlled by simply taking $\overline{CS}$ and $\overline{XFER}$ to a logic "0", ILE to a logic "1" and pulling $\overline{WR_1}$ low to load data to the input latch. Pulling $\overline{WR_2}$ low will then update the analog output. A logic "1" on either of these lines will prevent the changing of the analog output.

DAC0830 Series Application Hints (Continued)



ILE = LOGIC "1"; $\overline{WR2}$ and $\overline{XFER}$ GROUNDED

FIGURE 4

1.2 Single-Buffered Operation

In a microprocessor controlled system where maximum data throughput to the DAC is of primary concern, or when only one DAC of several needs to be updated at a time, a single-buffered configuration can be used. One of the two internal registers allows the data to flow through and the other register will serve as the data latch.

Digital signal feedthrough (see Section 1.5) is minimized if the input register is used as the data latch. Timing for this mode is shown in *Figure 4*.

Single-buffering in a "stand-alone" system is achieved by strobing $\overline{WR_1}$ low to update the DAC with $\overline{CS}$, $\overline{WR_2}$ and $\overline{XFER}$ grounded and ILE tied high.

1.3 Flow-Through Operation

Though primarily designed to provide microprocessor interface compatibility, the MICRO-DAC's can easily be configured to allow the analog output to continuously reflect the state of an applied digital input. This is most useful in applications where the DAC is used in a continuous feedback control loop and is driven by a binary up-down counter, or in function generation circuits where a ROM is continuously providing DAC data.

Simply grounding $\overline{CS}$, $\overline{WR_1}$, $\overline{WR_2}$, and $\overline{XFER}$ and tying ILE high allows both internal registers to follow the applied digital inputs (flow-through) and directly affect the DAC analog output.

1.4 Control Signal Timing

When interfacing these MICRO-DAC to any microprocessor, there are two important time relationships that must be considered to insure proper operation. The first is the minimum $\overline{WR}$ strobe pulse width which is specified as 900 ns for all valid operating conditions of supply voltage and ambient temperature, but typically a pulse width of only 180ns is adequate if $V_{CC} = 15V_{DC}$. A second consideration is that the guaranteed minimum data hold time of 50ns should be met or erroneous data can be latched. This hold time is defined as the length of time data must be held valid on the digital inputs *after* a qualified (via $\overline{CS}$) $\overline{WR}$ strobe makes a low to high transition to latch the applied data.

If the controlling device or system does not inherently meet these timing specs the DAC can be treated as a slow memory or peripheral and utilize a technique to extend the write strobe. A simple extension of the write time, by adding a wait state, can simultaneously hold the write strobe active and data valid on the bus to satisfy the minimum $\overline{WR}$ pulsewidth. If this does not provide a sufficient data hold time at the end of the write cycle, a negative edge triggered oneshot can be included between the system write strobe and the $\overline{WR}$ pin of the DAC. This is illustrated in *Figure 5* for an exemplary system which provides a 250ns $\overline{WR}$ strobe time with a data hold time of less than 10ns.

The proper data set-up time prior to the latching edge (LO to HI transition) of the $\overline{WR}$ strobe, is insured if the $\overline{WR}$ pulsewidth is within spec and the data is valid on the bus for the duration of the DAC $\overline{WR}$ strobe.

1.5 Digital Signal Feedthrough

When data is latched in the internal registers, but the digital inputs are changing state, a narrow spike of current may flow out of the current output terminals. This spike is caused by the rapid switching of internal logic gates that are responding to the input changes.

There are several recommendations to minimize this effect. When latching data in the DAC, always use the input register as the latch. Second, reducing the V_{CC} supply for the DAC from +15V to +5V offers a factor of 5 improvement in the magnitude of the feedthrough, but at the expense of internal logic switching speed. Finally, increasing C_C (*Figure 8*) to a value consistent with the actual circuit bandwidth requirements can provide a substantial damping effect on any output spikes.