เครื่องบันทึกรูปคลื่นไฟฟ้ากระแสสลับ

นางสาวอรยา ใจใหญ่
นายอำนาจ เจนจิโรจพิพัฒน์

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต
สาขาวิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2540

AC POWER LINE RECORDER

MISS ORASA CHAIYAI

MR.. AMNARD CHENCHIRODPIPAT

A SPECAIL PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE

RQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

DEPARTMENT OF APPLIED PHYSICS

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1997

หัวข้อโครงงานพิเศษ เครื่องบันทึกรูปคลื่นสัญญาณไฟฟ้า

โดย นางสาวอรษา ใจใหญ่

นายอำนาจ เจนจิโรจพิพัฒน์

ภาควิชา ฟิสิกส์ประยุกต์

อาจารย์ที่ปรึกษา ผู้ช่วยศาสตราจารย์วิชิต ศิริโชติ

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับโครงงานพิเศษฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

---------- หัวหน้าภาค

(รองศาสตราจารย์สุรพล รักวิจัย)

---------- ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์วิชิต ศิริโชติ)

---------- กรรมการ

(อาจารย์บดินทร์ ดำรงค์ศักดิ์)

---------- กรรมการ

(อาจารย์สุรวุฒิ กิจสัมพันธ์)

---------- กรรมการ

(อาจารย์ทรงวิทย์ เจริญรวยวัฒนา)

ลิขสิทธิ์ของภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อโครงงานพิเศษ	เครื่องบันทึกรูปคลื่นสัญญาณไฟฟ้า
นักศึกษา	นางสาวอรษา ใจใหญ่
	นายอำนาจ เจนจิโรจพิพัฒน์
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์วิชิต ศิริโชติ
ภาควิชา	ฟิสิกส์ประยุกต์
ปีการศึกษา	2540

บทคัดย่อ

เป็นเครื่องมือที่ใช้บันทึกรูปคลื่นสัญญาณไฟสลับในสายส่งไฟฟ้าโดยตัวบันทึกนี้จะสร้างเป็นวงจรซึ่งประกอบด้วย ไมโครคอนโทรลเลอร์89C51 หน่วยความจำขนาด128kB และตัวแปลงอะนาลอกเป็นดิจิตอลความละเอียดขนาด8บิต โดยระดับแรงดันจากเพาเวอร์ไลน์จะถูกลดระดับลงมาโดยหม้อแปลงและวงจรชมิดท์ทริกเกอร์จะทำหน้าที่เริ่มการจับภาพสัญญาณ โดยอัตราในการแซมปลิ้งของอะนาลอกเป็นดิจิตอลอยู่ในช่วง 100kHz-1460kHz ผลที่เก็บได้จะถูกส่งไปยังคอมพิวเตอร์ทางพอร์ตอนุกรม ตัวบันทึกนี้ออกแบบมาเพื่อใช้ตรวจสอบคุณภาพของไฟกระแสสลับเป็นระยะเวลานาน จะสามารถใช้ได้กับสภาพสัญญาณที่มีการเปลี่ยนแปลงนาน

Special Project Title	AC POWER LINE RECORDER
Name	Miss Orasa Chaiyai
	Mr. Amnard Chenchirodpipat
Special Project Advisor	Asst. Prof. Wichit Sirichote
Department	Applied Physics
Acadamic Year	1998

Abtract

A device used for recording AC power line has been develop. The recorder employs an embedded microcomputer board, 89C51 having a 128kB SRAM and 8-Bit resolution analog-to-digital converter. The AC power line conditioner utilizes a simple step-down transformer including schmittrigger for staring the A/D converter to capture the voltage waveform. The sampling rate of the converter uses 100kHz-1460kHz. The recorded voltage waveform can be transmitted to a PC through a serial port. The recorder is intended to be used for long-term monitoring the quality of AC power line.

กิตติกรรมประกาศ

โครงงานพิเศษนี้สำเร็จลุล่วงได้ด้วยดี เนื่องด้วยความอนุเคราะห์จากหลายๆฝ่ายด้วยกัน ดังนี้

บิดาและมารดา	ผู้ให้กำเนิดและคอยอุปการะ รวมทั้งเป็นผู้ให้กำลังในอย่างดี
อาจารย์ วิชิต ศิริโชติ	ผู้ให้คำปรึกษา ชี้แนวทางในการทำโครงงานและเป็นผู้เร่งให้ทำเสร็จได้ทันเวลา และช่วยเหลือเวลามีปัญหาตลอดเวลา
เพื่อนๆทุกคน	ที่คอยให้กำลังใจและช่วยเหลืออย่างสุดความสามารถเท่าที่กำลังจะทำได้

สารบัญเรื่อง

สารบัญตาราง

สารบัญรูป

บทที่ 1
บทนำ

โลกเราในทุกวันนี้มีการเจริญเติบโตทางด้านเศรษฐกิจและอุตสาหกรรมสูงมาก มีการแข่งขันกันในทุกๆด้าน และนำเครื่องจักรและอุปกรณ์อิเล็กทรอนิกเข้ามาช่วยในการผลิตเพื่อความสะดวก รวดเร็ว และความแม่นยำในการผลิต ซึ่งอุปกรณ์ต่างๆที่นำเข้ามาใช้จำเป็นต้องใช้ไฟฟ้าในการทำงานแทบทั้งสิ้น และอุปกรณ์แต่ละชนิดจะมีความต้องการใช้กระแสแรงดันที่แตกต่างกันออกไป ในการใช้งานเครื่องจักรและอุปกรณ์บางตัวอาจเกิดการดึงกระแสจากแหล่งจ่ายไฟฟ้า หรือก่อให้เกิดแรงดันกระชาก สัญญาณรบกวนต่างๆบนแหล่งจ่ายไฟฟ้า ซึ่งแหล่งจ่ายไฟฟ้านี้เป็นศูนย์รวมของอุปกรณ์ต่างๆ ดังนั้นเมื่อเกิดสัญญาณรบกวนบนแหล่งจ่ายไฟฟ้า ก็จะส่งผลต่ออุปกรณ์ที่นำมาต่อเข้าด้วย

ในประเทศไทยซึ่งมีการเจริญเติบโตทางด้านเศรษฐกิจและอุตสาหกรรมสูงมาก และมีการนำเครื่องจักรเข้ามาช่วยในการผลิตเป็นจำนวนมาก แต่ประเทศไทยไม่มีกฎหมายในการป้องกันสัญญาณรบกวนต่างๆของอุปกรณ์ที่ส่งมายังเพาเวอร์ไลน์ ดังนั้นในสายสัญญาณจึงมีสัญญาณรบกวนต่างๆอยู่เป็นจำนวนมาก ถ้าอุปกรณ์มีการป้องกันที่ไม่ดีพอ จะเกิดการทำงานผิดพลาดหรือเกิดการเสียหายได้

ความสมบูรณ์ของกำลังไฟฟ้า (Power Quality) จึงเข้ามามีบทบาทสำคัญในการป้องกัน หาสาเหตุและทำการแก้ไขให้ระบบมีประสิทธิภาพดีและปลอดภัยยิ่งขึ้น มีสาเหตุที่สำคัญอยู่ 4 ประการในการให้ความสนใจต่อความสมบูรณ์ของกำลังไฟฟ้า คือ

1 โหลดของอุปกรณ์มีความไวต่อการเปลี่ยนแปลงทางไฟฟ้า เพราะอุปกรณ์ในการควบคุมต่างๆจะประกอบด้วยไมโครโปรเซสเซอร์ ซึ่งมีความไวต่อการรบกวนต่างๆมาก

2 การเพิ่มประสิทธิภาพให้แก่อุปกรณ์ต่างๆให้ดียิ่งขึ้น เช่น การทำมอเตอร์ที่สามารถปรับความเร็วการหมุนได้ จะทำให้เกิด ฮาร์โมนิค ค่าต่างๆ ภายในระบบมากขึ้น

3 ผู้ใช้ไฟฟ้าก่อให้เกิดขึ้นเอง เช่น การสับสวิทซ์จะทำให้เกิดแรงดันทรานเซียนส์ได้

4 การผิดพลาดของอุปกรณ์ส่วนหนึ่งในวงจรโครงข่าย ทำให้วงจรรวมผิดพลาดตามไปด้วย

ในโครงงานนี้เราจะทำการศึกษาการเปลี่ยนแปลงของสัญญาณ และช่วงเวลาการเกิดการผิดพลาดจากปรากฏการณ์ต่างๆ เพื่อที่จะนำไปวิเคราะห์และหาแนวทางแก้ไขและป้องกันระบบให้ดียิ่งขึ้น

บทที่2
ทฤษฎีและหลักการ

2.1 ปัญหาความสมบูรณ์ของกำลังไฟฟ้า (Power Quality Problems)

ปัญหาความไม่บริสุทธิ์ของสัญญาณไฟฟ้าเกิดจากสาเหตุหลายประการด้วยกัน และมักจะเกิดจากการเปิดสวิตซ์ของอุปกรณ์ไฟฟ้าทุกชนิด ไม่ว่าสวิตช์อันนั้นจะเป็นสวิตช์ทางด้านกลไกหรือเป็นสวิตช์ที่ใช้อุปกรณ์อิเล็กทรอนิกส์ การเปิดสวิตช์เหล่านี้จะทำให้เกิดแรงดัน spike หรือ แรงดันทรานเซียนท์ ขึ้นในระบบจ่ายกำลังไฟฟ้า ซึ่งอาจจะส่งผลเสียหายให้กับอุปกรณ์ได้ทันทีถ้ามีระดับสูงมากเพียงพอ

เราสามารถอธิบายสภาพความสมบูรณ์ของกำลังไฟฟ้าในรูปของความสมบูรณ์ทางแรงดันเพราะแหล่งจ่ายไฟทั่วไปสามารถควบคุมได้เพียงแรงดัน และนอกจากนี้การเปลี่ยนแปลงของกระแสก็จะทำให้ค่าแรงดันเปลี่ยนแปลงตามด้วย เช่น การไหลของกระแสขณะวงจรเกิดการลัดวงจร ระดับแรงดันจะตกลงหรือหายไป กระแสที่ไหลขณะฟ้าผ่าจะเกิดการกระตุ้นให้ความถี่ของแรงดันเปลี่ยนแปลงไป

สถาบัน International Electrotechnical Comission (IEC) ได้แบ่งการรบกวนสัญญาณไฟฟ้าจากปรากฏการณ์ต่างๆ ตามตาราง 2.1 ซึ่งถ้าเรานำมาแบ่งแยกตามระยะเวลาการเกิด แอมปลิจูดให้เหมาะสมจะได้ตามตาราง 2.2 ซึ่งทำให้เราสามารถหาสาเหตุของปัญหาจากการวัดสัญญาณได้ง่ายขึ้น

Conducted low-frequency phenomena
Harmonics, interharmonics
Signal systems (power line carrier)
Voltage fluctuations
Voltage dips and interruptions
Voltage unbalance
Power frequency variations
Induced low-frequency voltages
dc in ac networks
Radiated low-frequency phenomena
Magnetic fields
Electric fields
Conducted high-frequency phenomena
Induced continuous-wave (CW) voltages or currents
Unidirectional transients
Oscillatory transients
Radiated high-frequency phenomena
Magnetic fields
Electric fields
Electromagnetic fields
Continuous waves
Transients
Electrostatic discharge phenomena (ESD)
Nuclear electromagnetic pulse (NEMP)

ตารางที่ 2.1 แสดงปรากฏการณ์การรบกวนทางคลื่นแม่เหล็กไฟฟ้าต่างๆ ซึ่งจำแนกโดย IEC

Categories	Typical spectral content	Typical duration	Typical voltage magnitude
1.0 Transients			
1.1 Impulsive			
1.1.1 Nanosecond	5-ns rise	<50 ns	
1.1.2 Microsecond	1-μs rise	50 ns–1 ms	
1.1.3 Millisecond	0.1-ms rise	> 1 ms	
1.2 Oscillatory			
1.2.1 Low frequency	<5 kHz	0.3–50 ms	0–4 pu
1.2.2 Medium frequency	5–500 kHz	20 μs	0–8 pu
1.2.3 High frequency	0.5–5 MHz	5 μs	0–4 pu
2.0 Short-duration variations			
2.1 Instantaneous			
2.1.1 Interruption		0.5–30 cycles	<0.1 pu
2.1.2 Sag (dip)		0.5–30 cycles	0.1–0.9 pu
2.1.3 Swell		0.5–30 cycles	1.1–1.8 pu
2.2 Momentary			
2.2.1 Interruption		30 cycles–3 s	<0.1 pu
2.2.2 Sag (dip)		30 cycles–3 s	0.1–0.9 pu
2.2.3 Swell		30 cycles–3 s	1.1–1.4 pu
2.3 Temporary			
2.3.1 Interruption		3 s–1 min	<0.1 pu
2.3.2 Sag (dip)		3 s–1 min	0.1–0.9 pu
2.3.3 Swell		3 s–1 min	1.1–1.2 pu
3.0 Long-duration variations			
3.1 Interruption, sustained		>1 min	0.0 pu
3.2 Undervoltages		>1 min	0.8–0.9 pu
3.3 Overvoltages		>1 min	1.1–1.2 pu
4.0 Voltage unbalance		Steady state	0.5–2%
5.0 Waveform distortion			
5.1 dc offset		Steady state	0–0.1%
5.2 Harmonics	0–100th harmonic	Steady state	0–20%
5.3 Interharmonics	0–6 kHz	Steady state	0–2%
5.4 Notching		Steady state	
5.5 Noise	Broadband	Steady state	0–1%
6.0 Voltage fluctuations	<25 Hz	Intermittent	0.1–7%
7.0 Power frequency variations		<10 s	

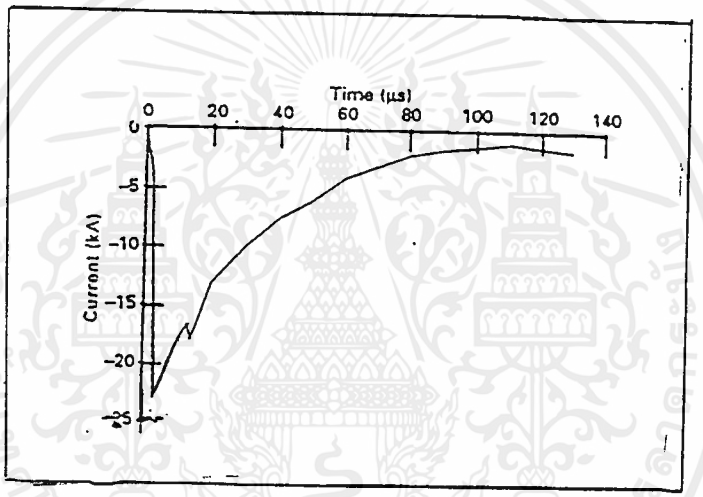
ตารางที่ 2.2 แสดงชนิดและคุณสมบัติของปรากฏการณ์แม่เหล็กไฟฟ้าของระบบกำลัง

2.1.1 การเกิดทรานเซียนท์ (Transients)

มักพบได้บ่อยในวงจร RLC เป็นสัญญาณที่ไม่ต้องการในระบบที่มีการเปลี่ยนแปลงขึ้นลงด้วยความเร็วสูง แต่จะมีอยู่เพียงชั่วขณะหนึ่งเท่านั้น Broadly ได้แบ่งแยกการเกิดทรานเซียนท์ได้เป็น 2 แบบคือ

ทรานเซียนท์แบบกระตุ้น (Impulsive transient)

เป็นการเปลี่ยนแปลงทันทีทันใด เป็นการเปลี่ยนแปลงของกระแสหรือแรงดันหรือทั้งสองอย่างจากสภาวะเสถียร โดยการเปลี่ยนแปลงจะเป็นไปทางเดียว (+ หรือ -)



รูปที่ 2.1 แสดงการเกิดทรานเซียนท์แบบกระตุ้น

ทรานเซียนท์แบบสั่น (Oscillatory transient)

จะคล้ายกับอิมพัลซีพแต่จะแกว่งทั้งซีกบวกและซีกลบของสัญญาณ เป็นการเปลี่ยนขั้วของสัญญาณแบบฉับพลันและยังสามารถแบ่งย่อยลงไปได้อีกคือ

■ สัญญาณเดิมมีความถี่มากกว่า 500 kHz

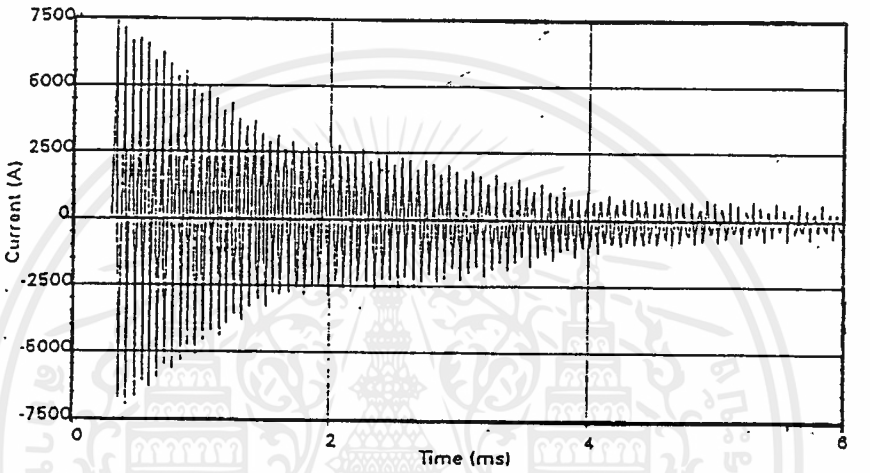
การเกิดทรานเซียนท์ชนิดนี้จะวัดได้ในระดับ 10^{-6} วินาที เป็นแบบทรานเซียนท์ความถี่สูง

■ สัญญาณเดิมมีความถี่ในช่วง 5-500kHz

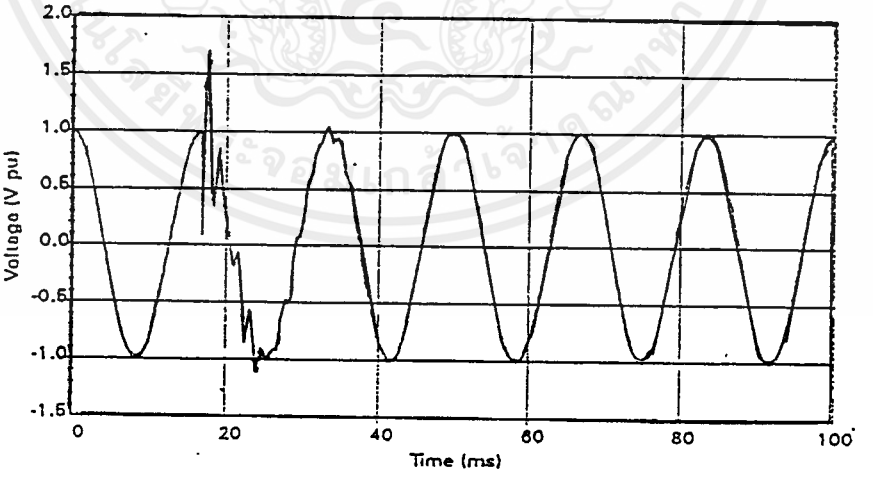
สัญญาณทรานเซียนท์อยู่ในระดับ 10^{-5} วินาที เป็นแบบทรานเซียนท์ความถี่ปานกลาง

■ สัญญาณเดิมมีความถี่ต่ำกว่า500kHz

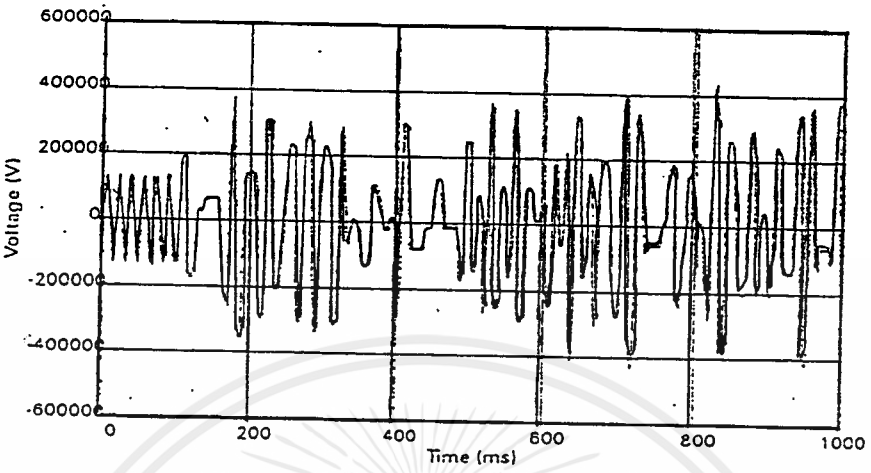
และอยู่ในช่วง 0.3-50 ms จะเป็นแบบทรานเซียนท์ความถี่ต่ำ



รูปที่ 2.2 ทรานเซียนท์แบบสั่นในช่วงความถี่ปานกลาง



รูปที่ 2.3 การสั่นของ ทรานเซียนท์แบบความถี่ต่ำเนื่องจาก
การสะสมพลังงานของตัวเก็บประจุ



รูปที่ 2.4 การสั่นของทรานเซียนท์แบบความถี่ต่ำในหม้อแปลงที่ไม่มีการต่อโหลด

2.1.2 การเปลี่ยนแปลงแรงดันแบบยาวนาน (Long-Duration Voltage Variations)

จะมีการเปลี่ยนความถี่ของกำลังนานกว่า 1 นาที สามารถแบ่งได้เป็นแรงดันเกินและแรงดันตก ซึ่งไม่ได้เกิดจากการทำงานผิดพลาดของระบบ แต่เกิดจากการปรับเปลี่ยนสถานะของโหลดไม่มีเสถียรภาพดีพอ

แรงดันเกิน (Overvoltage)

เป็นการเกิดแรงดันมากกว่าเดิม110เปอร์เซ็นต์ในช่วงเวลาที่ยาวนานกว่า 1 นาที มักเกิดจากการเปิดสวิทซ์ของโหลดขนาดใหญ่ และจะเกิดในระบบที่มีการรักษาระดับแรงดันไม่ดีเท่าที่ควร หรือการวางตำแหน่ง tap ติดของหม้อแปลง

แรงดันตก (Undervoltage)

เป็นการลดลง 90 เปอร์เซ็นต์จากระดับแรงดันเดิมและมีระยะเวลานานกว่า 1 นาที เกิดจากการเปิดใช้งานโหลดขนาดใหญ่

การอินเทอรัพท์ (Sustained interruptions)

เมื่อแรงดันตกลงจนเป็น 0 เป็นเวลานานมากกว่า 1 นาที จะเป็นการเกิดการขัดจังหวะมักเกิดแบบถาวรต้องทำการซ่อมแซม

2.1.3 <u>การเปลี่ยนแรงดันในระยะสั้น</u> (Short-Duration Voltage Variations)

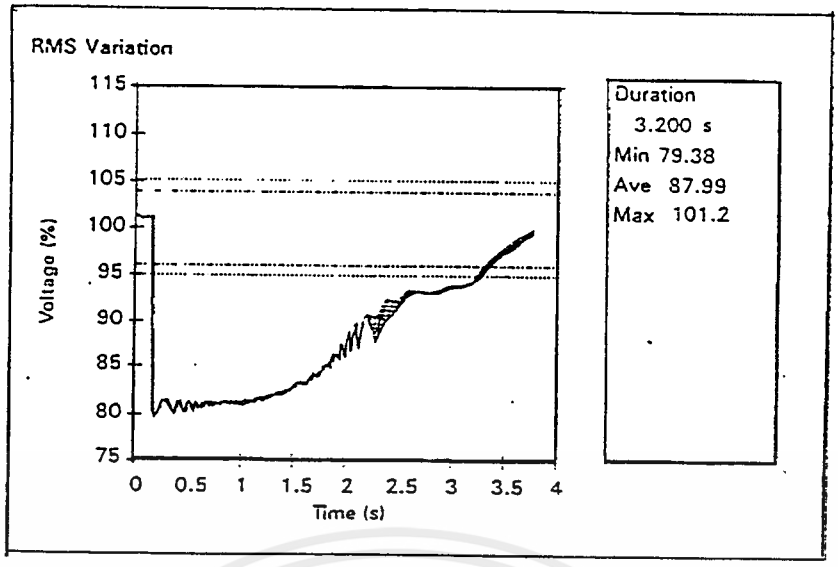
จะประกอบด้วยการ voltage dip และการอินเทอรัพท์ในช่วงสั้นๆ เกิดจากโหลดขนาดใหญ่ ซึ่งต้องการกระแสในการเริ่มต้นทำงานมาก หรือการเชื่อมต่อที่ไม่สมบูรณ์ของสายไฟ การผิดพลาดนี้จะทำให้แรงดันตกลง (sags) หรือเพิ่มขึ้น (swells) หรือหายไป (interruption) จะเกิดอยู่ตลอดเวลาถ้ายังไม่มีการแก้ไขหรือป้องกันการผิดพลาดนี้

การอินเทอรัพท์ (Interruption)

เกิดจากแหล่งจ่ายแรงดันหรือโหลดของกระแสมีค่าลดลงน้อยกว่า 0.1 pu ในช่วงเวลาไม่ถึง 1 นาที อาจเกิดจากการผิดพลาดของอุปกรณ์หรือการผิดพลาดของส่วนควบคุมต่างๆ โดยจะเกิดการผิดพลาดน้อยกว่า 30 รอบของคาบเวลาสัญญาณ

แรงดันตกลง (Sags หรือ Dips)

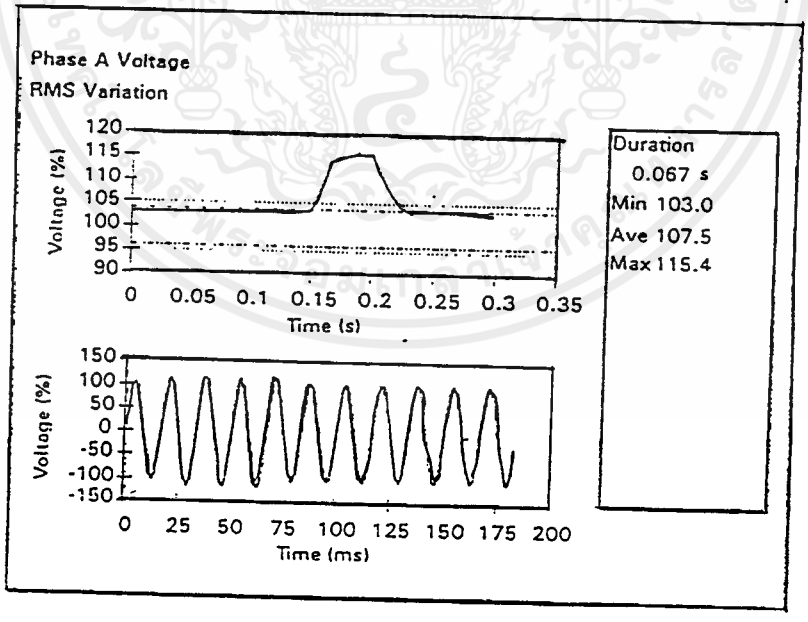
เป็นการลดลงในช่วง 0.1-0.9 pu เป็นระยะเวลา 0.5 รอบ-1นาที มักจะเกิดจากการทำงานผิดพลาดของระบบ แต่ก็สามารถเกิดจากการสะสมพลังงานของโหลดขนาดใหญ่และการทำงานของมอเตอร์



รูปที่ 2.5 แรงดันตกลงเนื่องจากการเริ่มทำงานของมอเตอร์

แรงดันเพิ่ม (Swells)

หมายถึงการเพิ่มของสัญญาณในช่วงระหว่าง 1.1 - 1.8 pu สามารถเกิดขึ้นได้หลายสาเหตุเช่น ในระบบอิมพีแดนซ์และกราวด์



รูปที่ 2.6 การเพิ่มขึ้นของแรงดันชั่วขณะ (Voltage swell)

2.1.4 <u>การผิดเพี้ยนของรูปคลื่น</u> (Waveform Distortion)

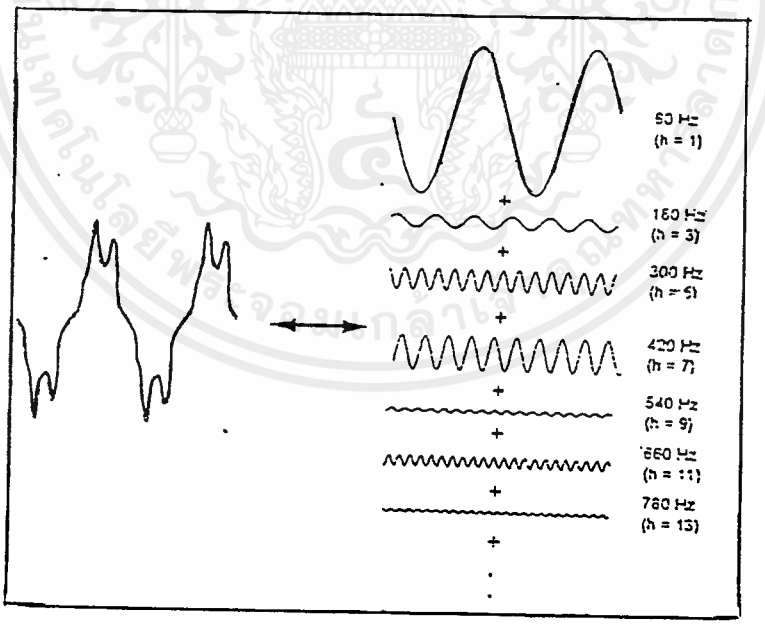
เป็นการเปลี่ยนแปลงรูปคลื่นของสัญญาณจากสัญญาณอุดมคติรูปไซน์ จะแบ่งออกได้เป็น5ชนิดคือ

การยกระดับจากไฟตรง (DC offset)

เป็นการเกิดองค์ประกอบทางแรงดันหรือกระแสของไฟตรงบนระบบไฟสลับ ซึ่งจะเกิดจากการรบกวนของสนามแม่เหล็กโลก หรือเกิดจากวงจรเรคติไฟล์ต่างๆ

ฮาร์โมนิค (Harmonics)

คือสัญญาณแรงดันหรือกระแสรูปไซน์ ซึ่งมีความถี่เป็นจำนวนเต็มเท่าของสัญญาณเดิม โดยการผิดเพี้ยนทางฮาร์โมนิคจะเป็นการรวมคลื่นที่มีฮาร์โมนิคต่างๆ ทำให้เกิดรูปคลื่นผิดเพี้ยนไปจากเดิม มักเกิดกับโหลดของอุปกรณ์ที่ไม่มีความเป็นเชิงเส้น



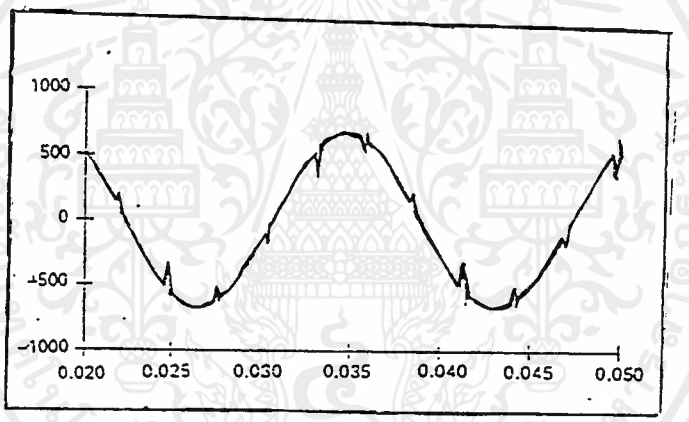
รูปที่ 2.7 การผิดเพี้ยนของสัญญาณ เนื่องจากการรวมของคลื่นความถี่ต่างๆ

อินเตอร์ฮาร์โมนิค (Interharmonics)

เป็นการเกิดฮาร์โมนิคที่มีความถี่ไม่เป็นจำนวนเท่าของจำนวนเต็มโดยจะพบใน frequency converter และการเหนี่ยวนำของมอเตอร์ต่างๆ

Notching

เป็นการรบกวนแรงดันแบบเป็นคาบเวลา ซึ่งเกิดจากการแปลงกระแสไฟสลับให้เป็นกระแสไฟตรงสามารถวิเคราะห์ได้จากการวิเคราะห์ทางฮาร์โมนิค



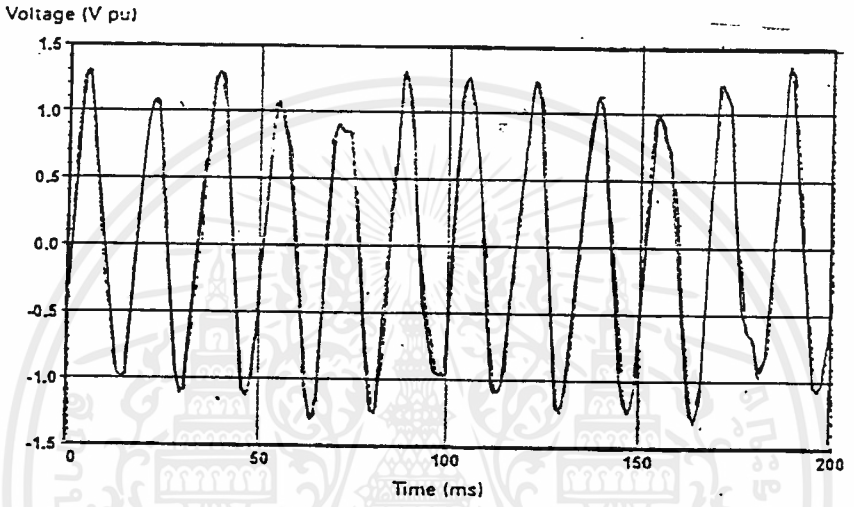
รูปที่ 2.8 การเกิด Notching

สัญญาณรบกวน (Noise)

เป็นสัญญาณรบกวนทางไฟฟ้า มีช่วงความถี่ต่ำกว่า 200kHz มักพบในตัวนำกระแสต่างๆ เกิดได้หลายสาเหตุ มักเกิดจากการวางระบบกราวด์ไม่ดีพอ

2.1.5 การแกว่งของแรงดัน (Voltage Fluctuation)

เป็นการเปลี่ยนระดับแรงดันแบบสุ่มและแบบต่อเนื่องหรือเรียกอีกชื่อว่า "Flicker" มีสาเหตุมาจากการส่งผ่านและการกระจายของสัญญาณในระบบ



รูปที่ 2.9 แสดงการแกว่งของแรงดัน

2.2 สัญญาณรบกวน (Noise)

สัญญาณรบกวนคือสัญญาณอื่นๆ นอกจากสัญญาณที่ต้องการให้ปรากฏที่จุดนั้นๆ การกำจัดสัญญาณรบกวน (Noise Immunity) ก็คือสิ่งที่ใช้วัดความสามารถในการให้สัญญาณรบกวนเข้ามาในระบบได้ถึงระดับหนึ่ง การที่สัญญาณรบกวนจะก่อปัญหาขึ้นมาได้จะต้องประกอบด้วย สิ่งแรกคือ แหล่งกำเนิดของสัญญาณรบกวน สองคือ ตัวรับหรือวงจรที่รับเอาสัญญาณรบกวนเข้าไปสุดท้ายคือ ช่องทาง หรือวิธีการที่สัญญาณรบกวนส่งผ่านจากแหล่งกำเนิดไปยังตัวรับที่ว่า วิธีการที่สัญญาณรบกวนผ่านเข้าไปยังตัวรับได้มีอยู่ 3 วิธีคือ

1 การผ่านโดยตรงทางสายตัวนำที่มีการเชื่อมวงจรถึงกัน

2 การส่งผ่านเนื่องจากอิมพีแดนซ์ร่วม ซึ่งเกิดขึ้นเมื่อมีกระแสของสองวงจรหรือมากกว่าไหลผ่านอิมพีแดนซ์ตัวเดียวกัน

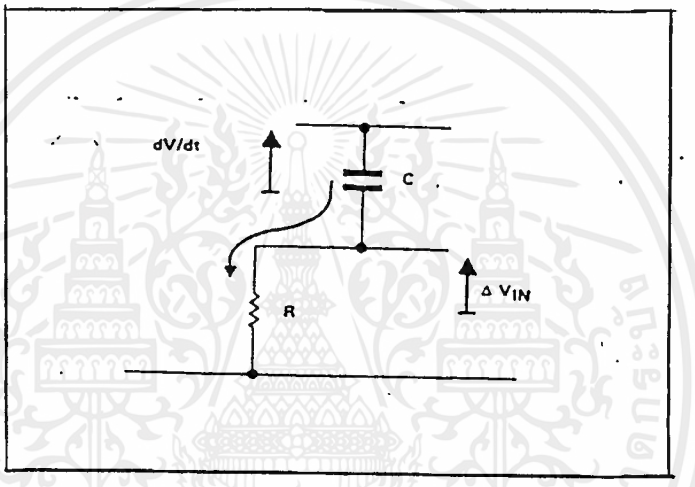
3 การส่งผ่านโดยสนามไฟฟ้าและสนามแม่เหล็ก การแพร่กระจายของสนามไฟฟ้าและสนามแม่เหล็กทำให้เกิดการส่งผ่านสัญญาณรบกวนได้

ระบบจ่ายไฟของอุปกรณ์ทั่วไป จะเป็นตัวรับสัญญาณรบกวนทั้งโดยตรงจากเพาเวอร์ไลน์ โดยวิธีการต่างๆ เช่น การเหนี่ยวนำหรือส่งผ่านด้วยคลื่นแม่เหล็กไฟฟ้าเข้ามาทางสายไฟ และนอกจากนี้ระบบจ่ายไฟอาจเป็นแหล่งกำเนิดสัญญาณรบกวน เนื่องจากการทำงานของวงจรภาคจ่ายไฟ หรือเกิดจากอุปกรณ์ที่ใช้คุณภาพไม่ดี หรือเดินสายจ่ายไฟไปยังวงจรไม่ถูกวิธี

2.2.1 ขอบเขตของสัญญาณรบกวน (noise margin) เป็นตัวบอกถึงระดับสัญญาณรบกวนที่อุปกรณ์ สามารถจะรับได้ก่อนที่จะให้เอาท์พุทที่ผิดพลาดออกมา ขอบเขตของสัญญาณรบกวนแบบ DC หรือความแตกต่างของแรงดันเอาท์พุทกับแรงดันเทรชโฮลด์ที่อินพุทโดยปกติแล้วแรงดันเทรชโฮลด์ (Threshold) ของเกทชนิด TTL จะอยู่ที่ประมาณ 1.4 v และเอาท์พุทของเกทจะเป็น 0.2 v หรือ 3.3 v ในสภาวะ 0 หรือ 1 ตามลำดับ

2.2.2 สาเหตุของสัญญาณรบกวน สัญญาณรบกวนอาจจะเกิดจากแหล่งที่มีการแพร่กระจาย (ของไฟฟ้า) เช่นจากรีเลย์ หรือการรบกวนข้ามช่องจากเกทอื่นๆ การตีกลับของสัญญาณเมื่อใช้สายยาวหรือการเกิดการกระตุก (Spike) ในทางเดินไฟเลี้ยงอันเกิดจากการสับสวิตช์ หรือการต่อเชื่อมจากสายไฟหลัก (Mains) หรืออาจจะเกิดจากสัญญาณ แปลกปลอมที่เข้ามาในอุปกรณ์ทางช่องทางต่ออินพุทหรือเอาท์พุท ซึ่งแต่ละชนิดที่กล่าวมานี้จะมาพิจารณากันต่อไป

■ *การแพร่ด้วยคลื่น* ในอุปกรณ์บางชนิดเช่นรีเลย์หรือมอเตอร์จะมีประจุแพร่กระจาย (Stray Capacitance) ระหว่างจุดกำเนิดนี้กับช่วงต่อระหว่างเกท ระดับอิมพีแดนซ์ที่จุดนี้จะได้รับการกำหนดจากอิมพีแดนซ์ที่เอาท์พุทของเกทตัวที่ขับ นั่นคือ R ถ้ามีแรงดันกระโชก (Transient) เป็นช่วงปรากฏขึ้นผ่านย่านประจุแพร่กระจายนี้ (C) และ อิมพีแดนซ์ (R) ดังที่แสดงดังรูป 2.10 อัตราการพุ่งขึ้น (Rise) จะเป็น dV/dt



รูปที่ 2.10 แรงดันกระชากที่ปรากฏข้ามประจุแพร่กระจาย C และอิมพีแดนซ์ R

กระแสที่ไหลผ่านประจุและอิมพีแดนซ์ที่เอาท์พุทจะเป็นประมาณ C dV/dt = I และถ้ามีขอบเขตของสัญญาณรบกวนอยู่ที่ 400 mV ดังนั้นจะมี $\Delta V_{IN} < 400$ mV ก่อนที่แรงดันกระโชกจะทำให้ลอจิกผลิตเอาท์พุทที่ไม่ถูกต้องออกมา

■ *การรบกวนข้ามช่อง* ครอสทอล์คจะเกิดขึ้นเมื่อมีสัญญาณถูกส่งเข้าไปในระหว่างเกทสองเกท ทำให้เกิดการชักนำกันขึ้นของระดับลอจิกระหว่างเกทอีกสอง เกทที่เหลือ ระดับความแรงของการชักนำนี้ขึ้นอยู่กับระดับอิมพีแดนซ์ที่ช่วงต่อระ

หว่างเกทสองเกทนี้ ถ้าเกททั้งสองใกล้กันอิมพีแดนซ์ก็จะเป็นเช่นเดียวกันกับอิมพีแดนซ์ที่เอาท์พุทของเกทตัวแรก

ในเรื่องของเส้นสายต่างๆนั้น จะต้องมีสิ่งอื่นๆ เข้ามาเกี่ยวข้องอย่างมากมาย อาทิ เช่นสายดิน (Earth) และระบบส่งกำลังไฟฟ้าซึ่งเป็นสิ่งที่สำคัญที่สุด

2.3 การลดทอนสัญญาณ

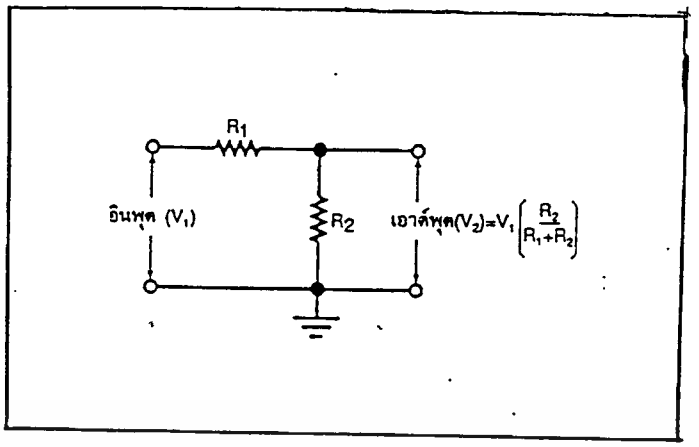
อุปกรณ์อิเล็กทรอนิกที่เราใช้กันอยู่ทั่วไป จะมีความสามารถในการรับแรงดันไฟฟ้าได้ต่างกันไป หรือแรงดันที่มีอยู่มีมากเกินความต้องการ ดังนั้นเราจึงต้องมีวงจรลดทอนสัญญาณเพื่อทำการปรับแต่งให้ได้สัญญาณที่ต้องการ สิ่งจำเป็นที่ต้องทำคือความสามารถในการควบคุมให้มีอัตราการลดทอนได้ตามความต้องการ ขณะที่คงค่าความต้านทานลัพธ์บนขั้วของวงจรลดทอนให้มีค่าคงที่ ซึ่งอาจทำได้โดยเพิ่มตัวต้านทานเข้าไป โดยผลรวมของค่าความต้านทานยังเท่าเดิม ในทางปฏิบัติความต้านทานทั้งทางด้านอินพุทและเอาท์พุทจะมีค่าคงที่ด้วย

ก่อนที่จะออกแบบวงจรที่มีคุณสมบัติข้างต้นได้ รูปที่ 2.11 เป็นรูปแบบง่ายๆ ของวงจรลดทอน ซึ่งประกอบด้วยตัวต้านทาน 2 ตัวคือ R_1 และ R_2 ต่อเป็นวงจรแบ่งแรงดันแรงดันเอาท์พุท V_2 จะแปรค่าแรงดันอินพุท ดังสมการ

$$V_2 = V_1 (R_2 \backslash (R_1+R_2)) \qquad (1)$$

จะเห็นว่าระดับการลดทอนสัญญาณนั้น ถูกกำหนดโดยค่าของตัวต้านทาน R_1 และ R_2 ทั้งสองตัวซึ่งสามารถจัดรูปสมการใหม่เพื่อให้ด้านซ้ายเป็นพจน์ของแรงดันเท่านั้น

$$V_2 \backslash V_1 = (R_2 \backslash (R_1+R_2)) \qquad (2)$$



รูปที่ 2.11 วงจรลดทอนชนิดค่าคงที่

ส่วนอัตราการขยายของวงจรขยาย คืออัตราส่วนของความแรงของสัญญาณด้านเอาท์พุทต่อแรงดันด้านอินพุทของวงจรขยาย ในการหาอัตราการลดทอนจะใช้วิธีเดียวกันนี้ แต่ต่างกันตรงที่อัตราการขยายมีค่ามากกว่า 1 อัตราการลดทอนมีค่าน้อยกว่า 1 ยกตัวอย่างเปรียบเทียบเพื่อให้เห็นภาพชัดเจนยิ่งขึ้นเช่นมีเครื่องขยายชุดหนึ่งป้อนแรงดันอินพุท 0.1 โวลต์แล้วได้แรงดันเอาท์พุทออกมา 1 โวลต์ คำนวณอัตราการขยายได้ดังนี้

อัตราการขยาย = แรงดันเอาท์พุต / แรงดันอินพุท

= 1.0 / 0.1

= 10 เท่า

เราสามารถหาค่าอัตราการขยายและอัตราการลดทอนได้ในอัตราส่วนเชิงตัวเลข แต่กรณีที่อัตราส่วนเชิงตัวเลขมีค่าแตกต่างกันมากๆ จะไม่สะดวกในการเปรียบเทียบค่า ในทางปฏิบัติจะใช้อัตราส่วนลอการิทึมหรืออัตราส่วนในหน่วยเดซิเบลแทน ข้อดีของการใช้อัตราส่วนลอการิทึมคือทำให้สามารถหาผลรวมของอัตราการขยายหรืออัตราการลดทอนของแต่ละระบบที่ต่ออนุกรมกันอยู่ได้ง่ายๆ ค่าเดซิเบลนิยามจากอัตราส่วนของค่ากำลัง 2 ค่าให้เป็น P_1 และ P_2 เขียนเป็นความสัมพันธ์ได้ดังนี้

$$\text{อัตราส่วน (dB)} = 10 \log (P_2/P_1) \qquad (3)$$

โดยปกติทั่วไปแล้ว เราจะวัดค่าในหน่วยแรงดัน (โวลต์) ดังนั้น ถ้าเขียนสูตรในรูปอัตราส่วนของแรงดันได้ย่อมสะดวกกว่า จากสมการข้างต้นจะได้ว่า

อัตราส่วน (dB) = 10 log (P1\P2)

$$P = V_2 / R$$

$$P_2 / P_1 = (V_2^2 / R_2) / (V_1^2 / R_1)$$

$$(V_2^2 / V_1^2) / (R_1 / R_2) = 1$$

เนื่องจาก $R_1 = R_2$; $R_1/R_2 = 1$

อัตราส่วน (dB) = $10\log(V_2^2 / V_1^2)$

= $20\log(V_2 / V_1)$ ...(4)

เงื่อนไขที่ทำให้สมการนี้เป็นจริงขึ้นมาได้คือ R_1 จะต้องมีค่าเท่ากับ R_2

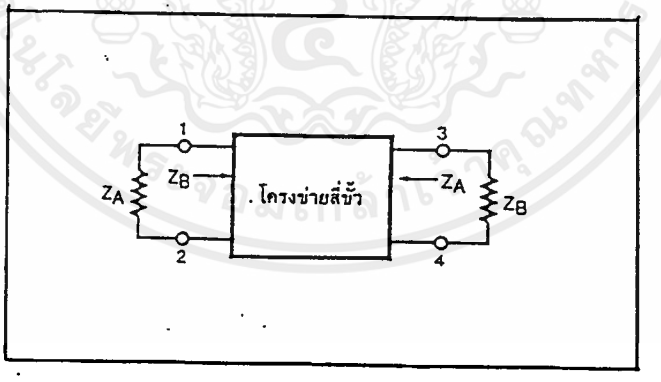
อัตราการขยาย (เดซิเบล)	อัตราส่วนทางแรงดัน (เอาต์พุต/อินพุต)	อัตราส่วนทางกำลัง (เอาต์พุต/อินพุต)
0	1.000	1.00
+3	1.414	2.00
-3	0.707	0.50
+6	2.000	4.00
-6	0.500	0.25
+10	3.160	10.00
+20	10.000	100.00
+30	31.600	1000.00
+40	100.000	100000.00

ตาราง 2.3 อัตราขยายในหน่วยเดซิเบลเมื่อเทียบเป็นอัตราส่วนทางแรงดันและอัตราส่วนทางกำลัง

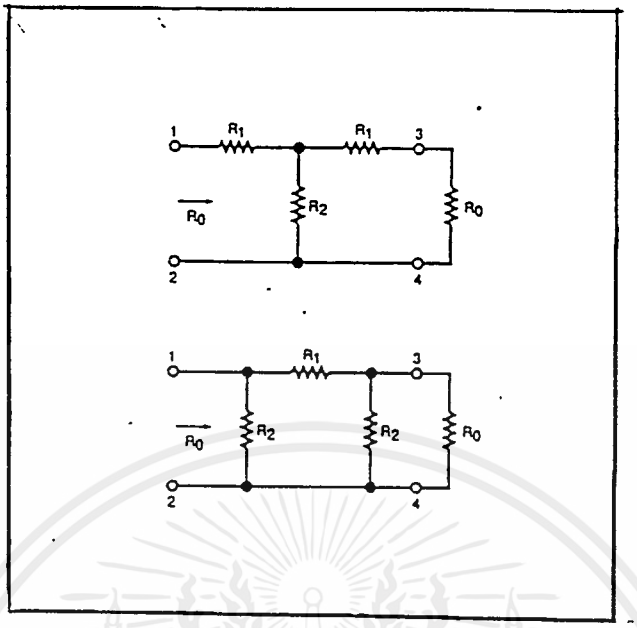
ตารางที่ 2.3 แสดงความสัมพันธ์ระหว่างอัตราส่วนของแรงดันหรือกระแสและอัตราส่วนของกำลังในหน่วยเดซิเบลซึ่งมีทั้งค่าบวกและลบ เราพบว่าอัตราส่วน (dB) ที่มีค่าเป็นลบเมื่อนำศักดาหรือกำลังของค่านี้มาหาส่วนกลับแล้ว จะมีค่าเท่ากับศักดาหรือกำลังของอัตราส่วน(dB) ที่เป็นบวกพอดีซึ่งสามารถใช้วิธีนี้คำนวณหาค่าที่ไม่มีในตารางโดยการใช้ค่าที่มีอยู่เดิมมาช่วย

2.3.1 อิมพีแดนซ์เสมือน (Lterative impedance)

รูปที่2.12 เป็นกล่องใบหนึ่งวงจรภายในประกอบด้วยตัวต้านทานต่อกันเป็นโครงข่ายมีอินพุท 2 ขั้วได้แก่หมายเลข 1 และหมายเลข 2 มีเอาต์พุทอีก 2 ขั้วได้แก่หมายเลข 3 และหมายเลข 4 เราเรียกว่าโครงข่าย 4 ขั้ว (four terminal network) จะกำหนดให้มี 2 ขั้วเป็นเส้นอ้างอิงในที่นี้ใช้หมายเลข 2 และหมายเลข 4 เป็นกราวด์ อินพุทอิมพีแดนซ์จะเป็น Z_a ส่วนเอาท์พุทอิมพีแดนซ์จะมีค่าเป็น Z_b กล่องนี้เองที่นำมาสู่อิมพีแดนซ์เสมือน



รูปที่ 2.12 โครงข่ายสี่ขั้วและอิมพีแดนซ์เสมือน



รูปที่ 2.13 วงจรลดทอนแบบตัวที (T) และตัวพาย (p)

อิมพีแดนซ์เสมือนของโครงข่ายใดๆ คือค่าของอิมพีแดนซ์ที่วัดได้จากขั้วคู่หนึ่งของโครงข่ายนั้น พบว่าค่าที่จะได้เท่ากับค่าอิมพีแดนซ์ซึ่งต่อกับอีกคู่ที่เหลืออยู่พอดี ในรูปที่ 2.12 นี้ขั้วหมายเลข 3 และ 4 มีอิมพีแดนซ์ Z_b ต่ออยู่เมื่อวัดค่าจากหมายเลข 1 กับ 2 ในกรณีที่โครงข่ายนี้สมมาตรกันจะส่งผลให้ $Z_a = Z_b$ จะเรียกค่าอิมพีแดนซ์ที่เท่ากันนี้ว่าคาแรกเตอริสติกอิมพีแดนซ์ (characteristic impedance)

รูปที่ 2.13 เป็นโครงข่ายวงจรลดทอนรูปตัวที (T) และ โครงข่ายรูปตัวพาย (p) โดยทั่วไปแล้วตัวต้านทานเหล่านี้ทำให้เกิดการลดทอนสัญญาณท่ากันในทุกช่วงความถี่ แต่ในความเป็นจริงยังมีข้อจำกัดอยู่กับคุณสมบัติของตัวต้านทานที่นำมาใช้

2.3.2 วงจรลดทอนแบบตัวที (T - type)

การออกแบบเพื่อใช้งานจริงจะต้องกำหนดไว้ก่อนว่าต้องการค่าคาแรดเตอริส ติกอิมพีแดนซ์ และค่าการลดทอนในขนาดเท่าไร แล้วทำการคำนวณหาค่าความต้านทานจากสมการ

$$R_1 = R_0(N-1)/(N+1)$$

$$R_2 = R_0 2n/(n^2-1)$$

ค่าที่คำนวณได้อาจไม่มีขายในท้องตลาดสามารถใช้ค่าใกล้เคียงแทนได้ แต่ค่าผิดพลาดต้องอยู่ภายในระดับที่ยอมรับได้หรืออาจแก้ปัญหาโดยการนำตัวต้านทานที่สามารถหาซื้อได้มาต่ออนุกรมหรือขนานกันให้ได้ค่าตามต้องการ

2.3.3 วงจรลดทอนแบบตัวพาย (P- type)

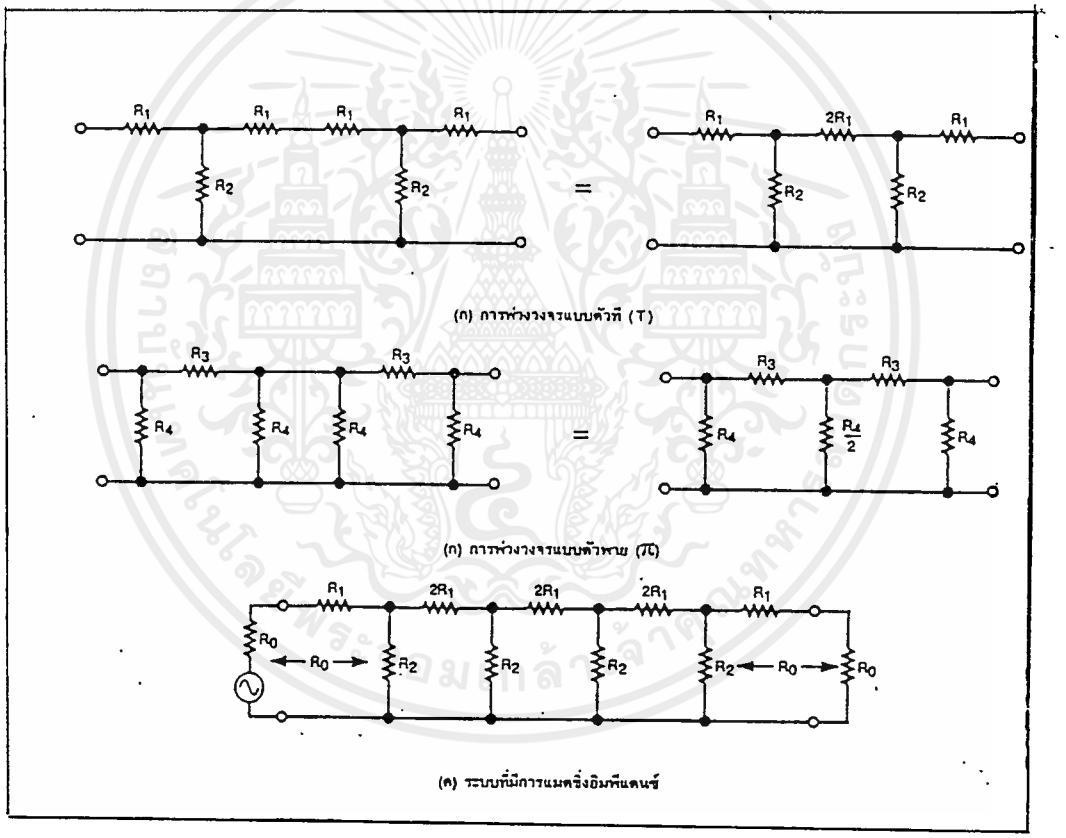
โดยวิธีเดียวกันการออกแบบวงจรลดทอนรูปตัวที สำหรับการหาตัวต้านทานที่ต้องการโดยกำหนดให้

$$R_1 = R_0(n^2-1)/2n$$

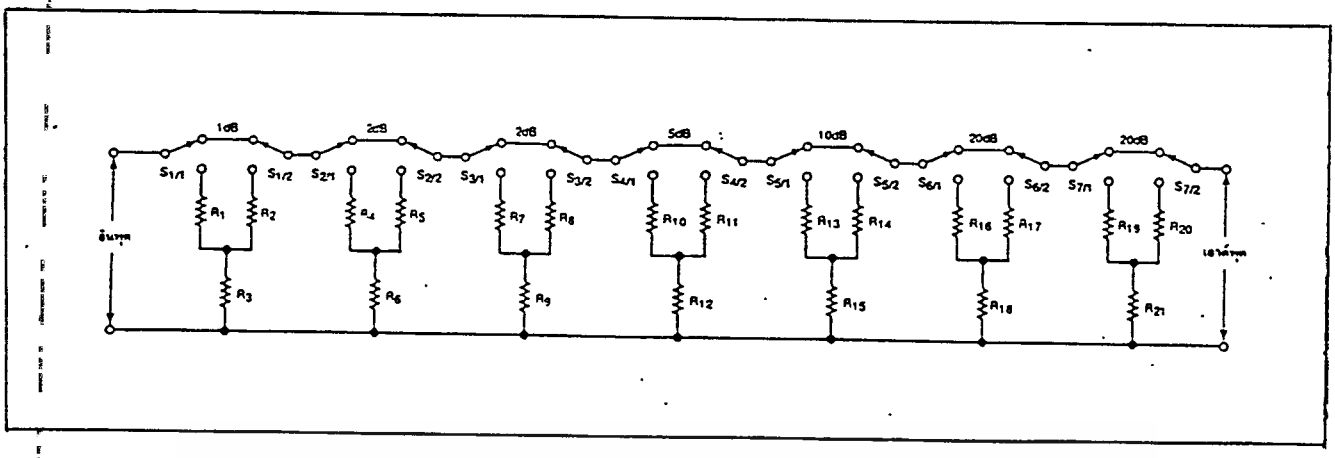
$$R_2 = R_0(n+1)/(n-1)$$

เราสามารถนำเอาวงจรลดทอนที่มีคาแรกเตอริสติกอิมพีแดนซ์เท่ากันมาต่อพ่วงกันหลายๆ ชุดเพื่อเพิ่มอัตราการลดทอนสัญญาณตามต้องการ ซึ่งการจะพบเห็นได้ในวงจรลดทอนที่ปรับค่าได้เป็นชั้นๆ โดยการใช้สวิตช์เลือกระดับการลดทอนตามต้องการ ดังรูปที่ 2.14 เมื่อต่อพ่วงกันแล้วคู่ของ R_1 ที่ต่ออนุกรมกันสามารถยุบเป็นตัวต้านทานตัวเดียวกันที่มีค่าเป็น $2R_1$ ในขณะที่คู่ของ R_4 ที่ต่อขนานกันก็ยุบรวมกันเป็นตัวต้านทานที่มีค่า $R_4/2$ สำหรับค่าอิมพีแดนซ์เสมือนมีค่าคงที่ตลอดโครงข่าย และมีค่าเท่ากับคาแรกเตอริสติกอิมพีแดนซ์วงจรลดทอนแต่ละชุดย่อมต้องแมตซึ่งกันได้ทั้งทางด้านอินพุท

และเอาท์พุท ปัญหาหนึ่งที่เกิดขึ้นคือเมื่อคำนวณค่าของตัวต้านทานได้แล้ว หาซื้อค่าที่ต้องการนั้นมาใช้ไม่ได้จึงต้องใช้วิธีนำค่าที่ใกล้เคียงที่สุดมาใช้งาน แต่ก็จะมีผลต่อความเที่ยงตรงในการใช้งานของวงจรลดทอน อีกปัจจัยหนึ่งที่สำคัญซึ่งมีผลต่อความเที่ยงตรงของวงจรคือ ค่าเปอร์เซ็นต์ความคลาดเคลื่อนของตัวต้านทาน สำหรับงานอิเล็กทรอนิกส์โดยทั่วไปการใช้ตัวต้านทานที่มีเปอร์เซ็นต์ความคลาดเคลื่อน 5 เปอร์เซ็นต์ ก็พอยอมรับได้แล้ว แต่สำหรับวงจรลดทอนสัญญาณที่ต้องการความถูกต้องสูง ต้องใช้ตัวต้านทานแบบฟิล์มโลหะที่มีค่าความผิดพลาด 1 เปอร์เซ็นต์แทน



รูปที่ 2.14 การต่อพ่วงวงจรลดทอนเข้าด้วยกัน



รูปที่ 2.15 วงจรลดทอนชนิดปรับค่าได้เป็นขั้นๆโดยปรับค่าได้ตั้งแต่ 0-60 เดซิเบล

2.4 การตรวจจับระดับศูนย์ (Zero Crossing)

จะใช้ออปแอมป์ซึ่งเป็นวงจรรวมที่ทำจากสารกึ่งตัวนำ สามารถนำไปสร้างวงจรต่างๆได้มากมาย มีอัตราขยายสูงมากสามารถนำมาสร้างวงจรเปรียบเทียบแรงดันได้ โดยมีคุณสมบัติและพารามิเตอร์ทั่วไปดังนี้

1 อินพุทอิมพีแดนซ์ ในทางอุดมคติมีค่าเป็นอนันต์ แต่ความจริงจะอยู่ที่ประมาณ 1 เมกกะโอห์ม

2 เอาท์พุทอิมพีแดนซ์ ตามอุดมคติ จะเป็น 0 แต่ทั่วไปจะอยู่ในช่วง 25 โอห์มขึ้นไป

3 การไบแอสอินพุท จะใช้กระแสต่ำมาก ถ้ายิ่งต่ำเท่าไรจะทำให้การทำงานของออปแอมป์ยิ่งดี

4 แรงดันออฟเซ็ท ในทางอุดมคติแรงดันระหว่างขั้วทั้งสองของออปแอมป์จะต้องเท่ากัน แต่ในความเป็นจริงจะมีข้างหนึ่งมีค่าสูงกว่าเสมอเรียกว่าแรงดันออฟเซ็ท โดยจะแบ่งเป็นทางด้านอินพุทและเอาท์พุท

5 ผลของอุณหภูมิ อุณหภูมิจะทำให้แรงดันออฟเซ็ทเปลี่ยนแปลงไป ซึ่งเราเรียกว่า “dirft”

6 อัตราสลูว์ (slew rate) เป็นค่าที่บอกความไวต่อการเปลี่ยนแปลงระดับสัญญาณของออปแอมป์โดยมีค่าเท่ากับ

$$\text{อัตราสลูว์} = \frac{\text{การเปลี่ยนแปลงสูงสุดของแรงดันเอาท์พุท}}{\text{การเปลี่ยนแปลงเวลา}}$$

$$= \frac{\Delta V_{out}(max)}{\Delta t}$$

7 การตอบสนองความถี่ อัตราขยายออปแอมป์จะลดลงเองใช้ในย่านความถี่สูงทั้งนั้นจะขึ้นอยู่กับออปแอมป์แต่ละตัวและแต่ละชนิดสามารถดูได้ใน data sheet ของออปแอมป์

8 อัตราการลดสัญญาณ common mode เป็นการกำจัดสัญญาณอินพุทที่เข้ามาซึ่งมีเฟสและแอมปลิจูดเท่ากันให้หายไป

จากคุณสมบัติของออปแอมป์ นำมาออกแบบเป็นวงจร เปรียบเทียบแรงดัน แบบ Schmitt trigger ได้ดังนี้

Schmitt trigger วงจรนี้มีลักษณะคล้ายคอมแพเรเตอร์แต่การป้อนกลับแบบบวกทำให้มีสองค่า สมมุติว่าออปแอมป์ได้รับกำลัง ± 20v และสามารถแกว่งได้ ± 18 v ที่เอาท์พุท ตัวต้านทาน R_1 และ R_2 แบ่งเอาท์พุทและสร้างแรงดันจ่ายไปยังขานอนอินเวอร์สติ้งของออปแอมป์

เมื่อเอาท์พุทของรูป 2.16 มีค่าบวกสูงสุด ตัวแบ่งแรงดันจะสร้างค่าด้านบน (UTP)

$$UTP = V_{max}\left(\frac{R_1}{R_1+R_2}\right)$$

$$= +18\ V\left(\frac{2.2\ k\Omega}{2.2\ k\Omega + 10\ k\Omega}\right)$$

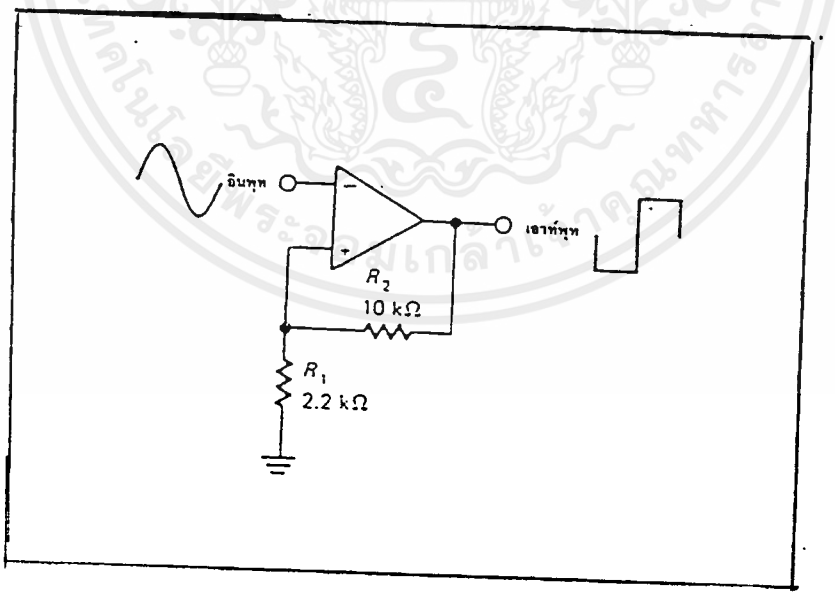
$$= +3.25\ V$$

เมื่อเอาท์พุทของรูป 2.16 มีค่าลบสูงสุด ตัวแบ่งแรงดันจะสร้างค่าด้านล่าง (LTP)

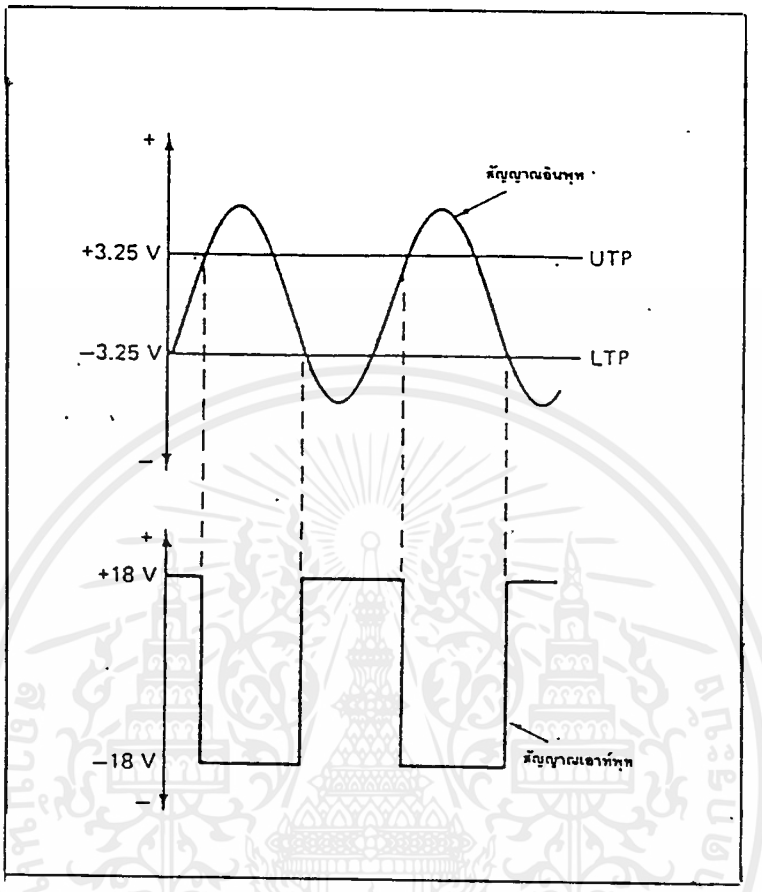
$$LTP = V_{min}\left(\frac{R_1}{R_1+R_2}\right)$$

$$= -18\ V\left(\frac{2.2\ k\Omega}{2.2\ k\Omega + 10\ k\Omega}\right)$$

$$= -3.25\ V$$



รูปที่ 2.16 การใช้ออปแอมป์เป็น Schmitt trigger



รูปที่ 2.17 การทำงานของ Schmitt trigger

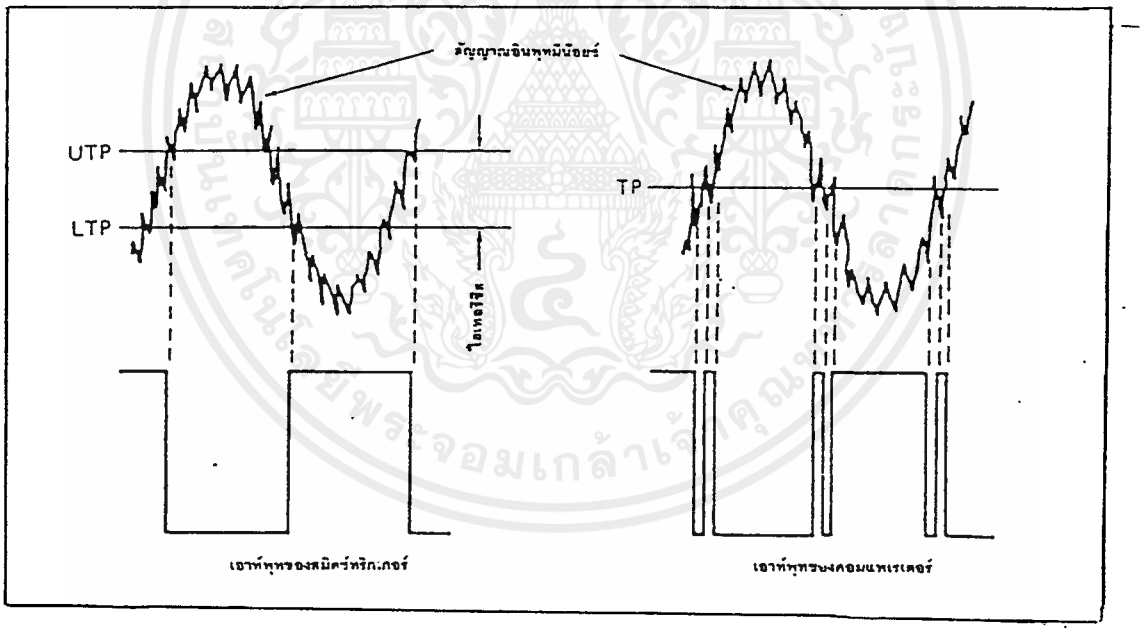
รูป 2.17 แสดงการทำงานของ Schmitt trigger กับสัญญาณอินพุทที่เกินค่าด้านบนและค่าด้านล่าง ขณะที่สัญญาณอินพุทเป็นค่าบวก อินพุทนี้คร่อมค่า +3.25 v ซึ่งค่าด้านบน อินเวอร์สติ้งอินพุทของออปแอมป์ในตอนนี้เป็นบวกมากกว่านอนอินเวอร์สติ้งอินพุท ดังนั้นเอาท์พุทเปลี่ยนเป็น -18 v อย่างรวดเร็ว ต่อมาสัญญาณอินพุทเริ่มไปทางลบและคร่อมค่า -3.25 v ซึ่งเป็นค่าด้านล่าง ในตอนนี้เอาท์พุทของ Schmitt trigger ไปทางบวกโดยมีค่าเป็น +18v ซึ่งเป็นค่าด้านบน (UTP) ความแตกต่างระหว่างค่าสองค่านี้ เรียกว่า ฮิสเทอรีซีส (hyteresis) และในตัวอย่างนี้เท่ากับ

ฮิสเทอรีซีส = UTP - LTP

= + 3.25 - (-3.25)

= 6.5 v

ฮิสเทอรีซีสมีความสำคัญเมื่อมีสัญญาณรบกวนในวงจรดิจิตอลหรือระบบ รูป 2.18 แสดงเหตุผลที่กล่าวมา เอาท์พุทของ Schmitt trigger มีความแตกต่างจากเอาท์พุทของคอมแพเรเตอร์และ Schmitt trigger ฮิสเทอรีซีส สัญญาณรบกวน บนสัญญาณไม่มีผลเสียต่อ Schmitt trigger และเอาท์พุทมีความถี่เดียวกับอินพุท แต่คอมแพเรเตอร์เอาท์พุทมีความถี่สัญญาณอินพุทเนื่องจากสัญญาณรบกวนทำให้เกิดความผิดพลาด สังเกตได้ว่าคอมแพเรเตอร์มีค่าเปรียบเทียบเพียงค่าเดียว (ไม่มี ฮิสเทอรีซีส) สัญญาณรบกวนบนสัญญาณทำให้มีการตกคร่อมกลับ (crossing back) ซึ่งผ่านจุดเปรียบเทียบ ทันทีและมีพัลซ์ปรากฏที่เอาท์พุท

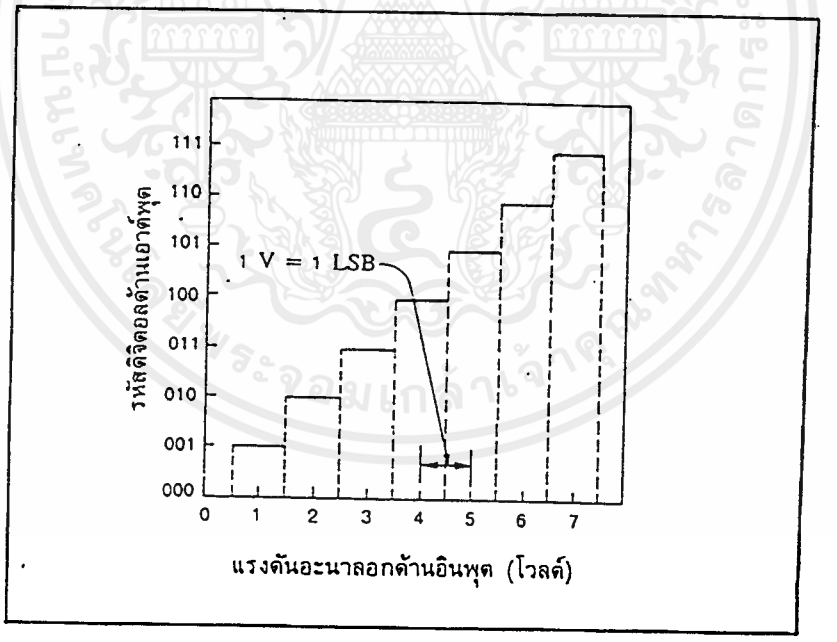


รูป 2.18 การเปรียบเทียบเอาท์พุทของ Schmitt trigger กับเอาท์พุทของคอมแพเรเตอร์เมื่ออินพุทมีค่าสัญญาณรบกวน

2.5 ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอล

กระบวนการต่างๆ ที่เกิดขึ้นตามธรรมชาติส่วนใหญ่หากนำมาแปรค่าเป็นสัญญาณทางไฟฟ้ามักเป็นสัญญาณที่อยู่ในรูปของแรงดันหรือกระแส หรือไม่ก็เป็นลักษณะของความต้านทาน ลักษณะที่ได้จะเป็นสัญญาณอะนาลอก ซึ่งไม่สามารถนำไปใช้กับคอมพิวเตอร์โดยตรงได้จึงจำเป็นต้องมีวงจรแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอล เราเรียกวงจรที่ทำหน้าที่ดังกล่าวว่า เอดีซี

2.5.1 หลักการเบื้องต้นของวงจรเอดีซี หากนำเอาเอดีซีขนาด 3 บิต มาเขียนกราฟคุณสมบัติระหว่างสัญญาณอินพุตกับเอาต์พุต สมมติว่าแรงดันอินพุต V_i เปลี่ยนค่าจาก 0-7 โวลต์ และได้สัญญาณเอาต์พุตที่เป็นสัญญาณดิจิตอลจาก 000 - 111 ดังแสดงในรูปที่ 2.19



รูปที่ 2.19 แสดงกราฟคุณสมบัติของเอดีซีขนาด 3 บิต

2.5.2 ค่าความละเอียดของเอดีซี ค่าความละเอียดของเอดีซีหาได้จากการเปลี่ยนแปลงค่าแรงดันอินพุตแล้วทำให้สัญญาณดิจิตอลเปลี่ยนค่าบิตนัยสำคัญต่ำสุดไป

ความละเอียด = ค่าแรงดันอินพุตต่อบิต = ค่าเต็มสเกลหารด้วย $2^n - 1$

หรือถ้าอ้างถึงเรื่องดีเอซีที่กล่าวมาแล้วจะได้ว่า

$$\text{ความละเอียด} = 2^n$$

ถ้า n คือ จำนวนบิตของวงจร

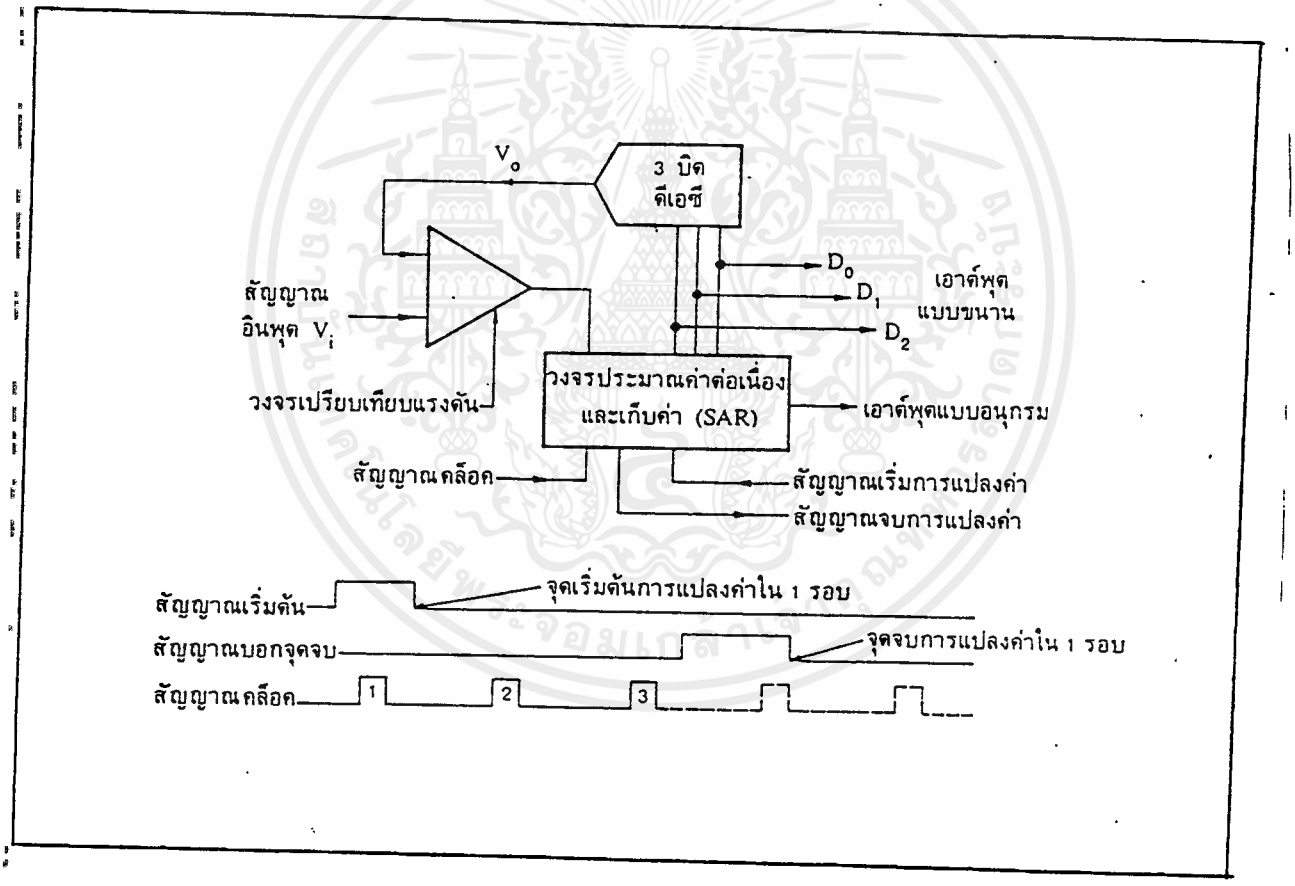
จากรูปที่ 2.19 จะเห็นว่าขณะเอาต์พุตเป็น 001 แรงดันอินพุตมีค่าเท่ากับ 1 โวลต์ ซึ่งค่านี้เกิดจากแรงดันค่าเฉลี่ยของ 0.5 โวลต์ กับ 1.5 โวลต์ หรืออาจกล่าวได้ว่าขณะเอาต์พุตเป็น 001 แรงดันอินพุตถูกกำหนดให้จะอยู่ในช่วง 0.5 โวลต์ถึง 1.5 โวลต์ ซึ่งจะมีค่าผิดพลาดเท่ากับครึ่งบิต ดังนั้นหากต้องการให้ค่าผิดพลาดนี้ลดลงจำเป็นต้องเพิ่มจำนวนบิตให้สูงขึ้น

วิธีการเปลี่ยนสัญญาณอะนาลอกให้เป็นดิจิตอลนั้นมีมากมายหลายแบบ หากแบ่งตามความเร็วที่ใช้ในการเปลี่ยนสัญญาณมี 3 แบบ ดังแสดงคุณสมบัติของแต่ละแบบในตารางที่ 2.4

แบบ	ความเร็ว	ช่วงเวลาแปลงสัญญาณใน 1 รอบ	การใช้งาน
รวบรวมค่า (integrating)	ช้า	มิลลิวินาที	ดีซีโวลต์มิเตอร์
ประมาณค่าต่อเนื่อง (successive approximation)	เร็ว	ไมโครวินาที	สัญญาณเสียง
แฟลช (flash)	เร็วมาก	นาโนวินาที	สัญญาณภาพ

ตารางที่ 2.4 แสดงการเปรียบเทียบเอดีซีแบบต่าง ๆ

2.5.3 เอดีซีแบบประมาณค่าต่อเนื่อง เอดีซีแบบนี้จะประกอบด้วยวงจรเอดีซี วงจรเปรียบเทียบแรงดันและวงจรรีจิสเตอร์ เก็บค่าที่ได้หลังจากการประมาณค่าสัญญาณอินพุตที่รับเข้ามา (SAR = Successive Approximation Register) โดยวงจรรีซิสเตอร์มีขาอินพุต 1 ขาและขาควบคุมอีก 3 ขา คือ ขาแรกเป็นขาสัญญาณบอกจุดเริ่มกระบวนการแปลงค่า ขาที่2 เป็นขาเอาต์พุตบอกจุดจบกระบวนการแปลงค่า ขาที่3 เป็นอินพุตรับสัญญาณคล็อคสำหรับควบคุมกระบวนการแปลงค่าในแต่ละรอบ ขาเอาต์พุตจะให้สัญญาณดิจิตอลมีทั้งแบบอนุกรมและแบบขนาน ดังแสดงในรูปที่ 2.20



รูปที่ 2.20 แสดงแผนผังและรูปคลื่นของวงจรเอดีซีแบบประมาณค่าต่อเนื่อง

การทำงานของวงจร

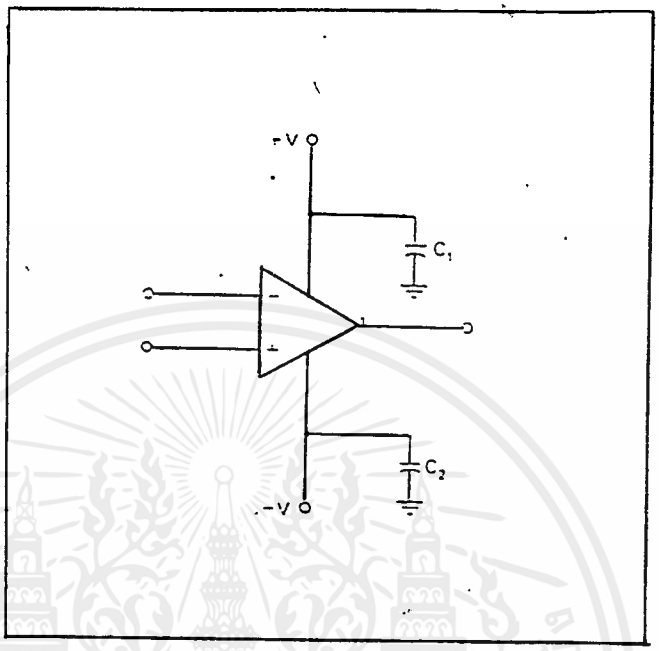
การแปลงค่าในแต่ละรอบจะเริ่มขึ้นเมื่อวงจรได้รับสัญญาณเริ่มต้น วงจรรีจิสเตอร์ส่งค่าดิจิตอลที่ได้ประมาณค่าแล้วออกไปยังวงจรเอดีซี เพื่อทำการแปลงค่าเป็นสัญญาณอะนาลอกส่งกลับมาเปรียบเทียบกับแรงดันอินพุตว่าค่าใดมากกว่ากัน เพื่อนำไปปรับค่าสัญญาณดิจิตอลแต่ละบิตให้ถูกต้องตรงกับค่าที่ได้ป้อนเข้ามาทางอินพุต การเปรียบเทียบเริ่มจากบิตสูงไปยังบิตต่ำเสมอ เมื่อครบทุกบิตวงจร SAR จะส่งสัญญาณสิ้นสุดกระบวนการออกไป และจะได้รับสัญญาณดิจิตอลเอาต์พุตที่สัมพันธ์กับค่าแรงดันอินพุต

2.6 การเพิ่มเสถียรภาพให้แก่วงจรและการป้องกัน

2.6.1 การเพิ่มเสถียรภาพให้แก่วงจร

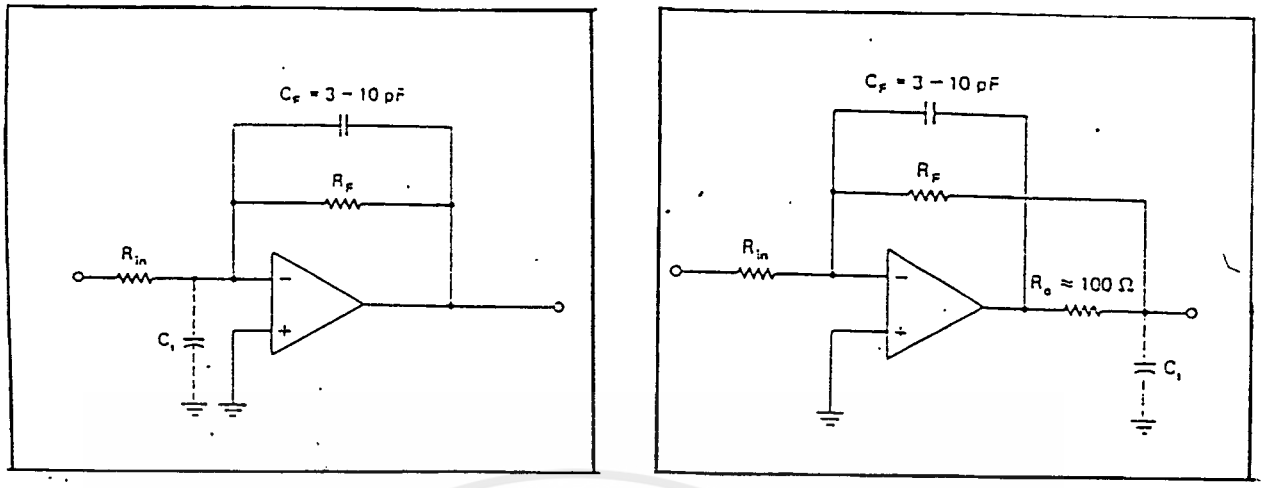
การเพิ่มเสถียรภาพให้แก่วงจรขยายชนิดที่มีการป้อนกลับ หมายถึงการป้องกันไม่ให้วงจรออสซิลเลส (เพิ่มขึ้นจนเราควบคุมไม่ได้) วงจรจะต้องมีอัตราขยายตามที่ถูกออกแบบมาและต้องสามารถลดสัญญาณรบกวนออกไปให้มากที่สุด การเพิ่มเสถียรภาพดังกล่าวอาจทำให้ได้โดยการเลย์เอาท์ (layout) หรือการวางอุปกรณ์แต่ละชนิดในตำแหน่งที่เหมาะสม การต่ออุปกรณ์ต่างๆ โดยใช้สายไฟตัวนำขนาดสั้นที่สุดเท่าที่จะทำได้ และการลดอิมพีแดนซ์ของกราวด์ร่วมให้เหลือน้อยที่สุด เป็นต้น

นอกจากวิธีดังกล่าว การทำให้แหล่งจ่ายไฟฟ้ามีค่าคงที่ตลอดเวลาก็สามารถเพิ่มเสถียรภาพให้แก่ระบบได้ซึ่งทำได้โดยการดีคัปปลิ้งสัญญาณรบกวนดังแสดงในรูป 2.21 ตัวเก็บประจุที่ใช้ควรเป็นชนิดแทนทาลัมและมีขนาด 1 ไมโครฟารัด



รูป 2.21 การเพิ่มเสถียรภาพโดยการดีคัปปลิ้งสัญญาณ

นอกจากการต่อตัวเก็บประจุให้แหล่งจ่ายไฟแล้ว ควรต่อตัวเก็บประจุคร่อมตัวต้านทาน ดังรูป2.21a ด้วยเพื่อลดปัญหาซึ่งเกิดจากตัวเก็บประจุชนิดสเตรย์ในด้านอินพุท นั้นไม่ใช่ตัวเก็บประจุที่เราสามารถเห็นด้วยตา แต่เป็นเพียงตัวเก็บประจุเสมือน ที่ปรากฏอยู่ ณ ตำแหน่งดังในรูป อาจเกิดจากคาปาซิแตนซ์ในสายไฟหรือตัวเก็บประจุภายในออปแอมป์ นอกจากนี้ควรต่อตัวต้านทานที่ภาคเอาท์พุท ดังรูป 2.21b เพื่อช่วยลดปัญหาจาก ที่เอาท์พุท



(a) (b)

รูป 2.22 แสดงการเกิดสเตรย์คาพาซิสแตนซ์ว่างินพุท (a) และ เอาท์พุท (b)

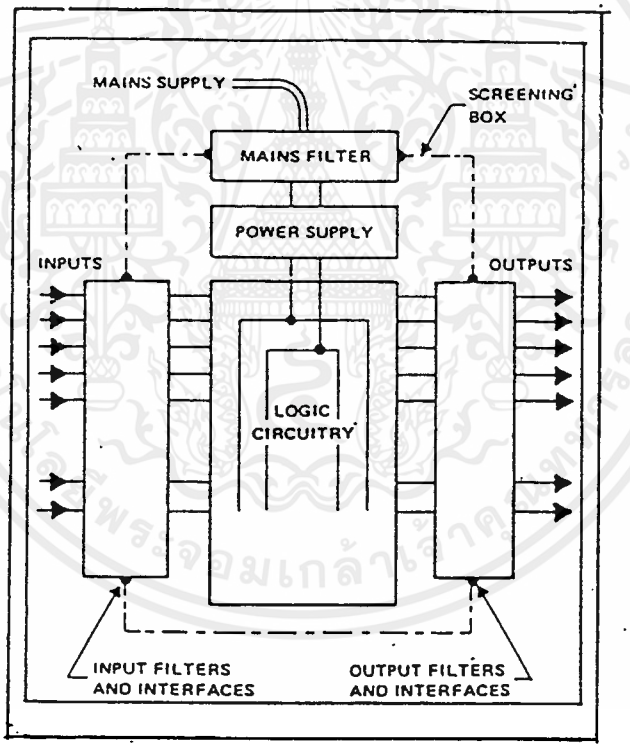
การวางเส้นทางต่างๆในแผ่นวงจรพิมพ์จะต้องให้กว้างมากที่สุดเท่าที่จะกว้างได้ และควรเดินข้ามกันหรือคู่เคียงกันเพื่อให้อิมพีแดนซ์ในระหว่างสายสัญญาณต่ำที่สุดในทางอุดมคติแล้วด้านหนึ่งของแผ่นวงจรนั้นควรจะทำให้เป็นบริเวณดินไป ส่วนสายไฟเลี้ยงนั้นก็อาจจะให้อยู่บนด้านเดียวกันได้เช่นกัน แค่ควรจะต้องมีการต่อเชื่อมหรือกำกับ (couple) อย่างสม่ำเสมอ ดังนั้นพื้นผิวทั้งหมดจึงทำตัวเป็นกราวด์ของไฟ AC และช่วงต่อทั้งหมดก็จะมีอิมพีแดนซ์ที่ต่ำและการรบกวนข้ามช่องก็จะลดลงไป

ไม่ว่าจะใช้กราวด์เพลนหรืออะไรก็ตาม จะต้องมีคาพาซิเตอร์ขนาด 0.1 ถึง 0.01 ต่อคร่อมระหว่างเส้นทางไฟเลี้ยงและเส้นกราวด์ตลอดทุกๆ 5 ตัวของไอซีก็พอ แต่ขอให้เป็นชนิดที่ตอบสนองความถี่สูงได้ดีสักหน่อย (เช่นแบบเซรามิค) สิ่งสำคัญที่จะต้องจดจำก็คือแม้ว่าอัตราของพัลซ์มักจะสูงเสมอ ดังนั้นแผ่นวงจรแต่ละแผ่นจึงควรมีคาพาซิเตอร์ค่าสูงหนึ่งตัว เพื่อจ่ายความแตกต่างในกระแสที่เกทต้องการในสภาวะที่ต่างกัน

การแยกส่วนเส้นทางไฟเลี้ยงกับแผ่นวงจรสามารถทำได้โดยการจ่ายผ่านโช๊ค (choke) ไปยังแผ่นวงจรแต่ละแผ่น อย่างไรก็ตามก็ควรจะระวังเรื่องรีโซแนนซ์ที่จะเกิดขึ้นระหว่างโช๊คและคาพาซิเตอร์กรองกระแสขนาดใหญ่ด้วย

สิ่งหนึ่งที่ควรทราบคืออิมพีแดนซ์ของเส้นสัญญาณแคบๆถึงจะทำให้กว้างขึ้นเป็นสองเท่าก็ไม่สามารถจะลดลงได้มากนัก ดังนั้นสิ่งที่ควรจะทำในกรณีเช่นนี้คือทำเส้นทางเดินสองเส้นให้ขนานกัน แม้ว่าจะใช้เส้นทางไฟเลี้ยงขนาดใหญ่ก็ตามการเดินสายคู่ขนานในลักษณะเส้นทางไฟเลี้ยงแบบวงแหวน (Ring Main) จะให้ผลดีกว่า

ในกรณีที่มีกระแสสูง (AC หรือ DC) ผ่านช่วงต่อตรงขอบเขตของแผ่นวงจรเข้ามา ก็ควรจะเพิ่มจุดสัมผัสให้มากกว่าหนึ่งจุดเข้าไว้โดยเฉพาะกับแผ่นที่จะต้องมีการโอนย้ายบ่อยๆเพราะจุดสัมผัสอาจจะกร่อนไปก็ได้



รูปที่ 2.23 มาตรการต่างๆ ที่ใช้ป้องกันวงจรลอจิกจากสภาพแวดล้อมที่เลวสุด

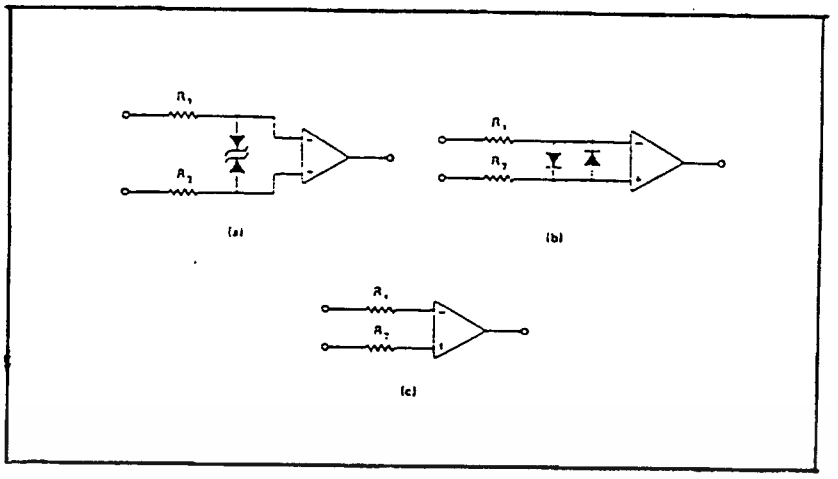
ถ้าเราต้องการใช้งานลอจิกในสถาพที่มีสิ่งรบกวนมากมายก็ควรจะมีการป้องกันเสียก่อน อุปกรณ์ลอจิกทั้งหมดรวมทั้งแหล่งจ่ายไฟนั้นจะอยู่ในตัวกล่องที่เป็นช่องด้วยกัน สายทุกสายที่ผ่านเข้าและออกจากกล่องนี้ควรจะได้รับการกรองเป็นอย่างดีเมื่อผ่านผนังของกล่อง

ระบบที่ดีควรจะเป็นเช่นในรูปที่ 2.22 ซึ่งมีภาคกรองไฟหลัก (Main Filter) อยู่ด้านนอกของกล่องโดยมีเอาท์พุทที่ได้รับการกรองแล้วเดินผ่านผนังของกล่องเข้าไปโดยตรง อินพุทและเอาท์พุททั้งหมดก็จะต้องได้รับการกรองในขณะที่ผ่านเข้าในระบบ การต่อเชื่อมลอจิกที่มีระบบการควบคุมเข้าและภาคขับออกก็ควรจะได้รับการรวมในภาคกรองนี้เช่นเดียวกัน การต่อลอจิกด้วยจุดเชื่อมที่มีอิมพีแดนซ์ต่ำจะทำให้การกวนข้ามช่องและการคัพปลิ้งลดน้อยลงไปวิธีนี้ทำได้โดยใช้อีกด้านหนึ่งของแผ่นวงจรทำเป็นกราวด์เพลน สายไฟเลี้ยงควรจะอยู่ด้านเดียวกับกราวด์เพลนแต่จะต้องมีเส้นกว้างและมีการคัพปลิ้งอย่างสม่ำเสมอในลักษณะนี้ด้านทั้งด้านจะทำตัวเป็นกราวด์ชนิด AC

2.6.2 การป้องกันภาคอินพุท

ออปแอมป์

เช่นเดียวกับอุปกรณ์โซลิดสเตท (solid- state device) ชนิดอื่น ออปแอมป์ก็มีค่าจำกัดสำหรับภาคอินพุทเช่นกัน ซึ่งหมายความถึงทั้งอินพุทแบบดิฟเฟอเรนเชียลและแบบคอมมอนโหมดจากการศึกษาวงจรภายในออปแอมป์ขั้วอินพุทจะทำตัวเสมือนมีซีเนอร์ไดโอดสองตัวต่อกลับหัวกัน ดังนั้นแรงดันที่ตกคร่อมขั้วอินพุททั้งสองจึงมีค่าค่อนข้างคงที่ จนกระทั่งถึงแรงดันพังทลาย (breakdown voltage) ซีเนอร์ไดโอดทั้งสองจะเสมือนเป็นตัวต้านทานที่มีค่าต่ำมาก เป็นผลให้เกิดกระแสจำนวนมากไหลผ่านออปแอมป์ และกระแสนี้จะทำให้ทรานซิสเตอร์ภายใน ออปแอมป์เสียหาย

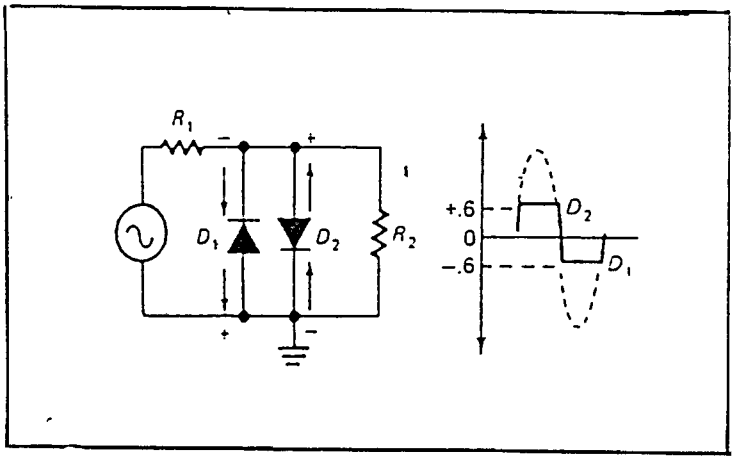


รูปที่ 2.24 แสดงการใช้ไดโอดและตัวต้านทานสำหรับการป้องกันภาคอินพุท

รูป 2.24 แสดงการใช้ตัวต้านทานและไดโอดสำหรับการป้องกันภาคอินพุท รูป 2.24a แสดงการต่อซีเนอร์ไดโอดที่มีแรงดันพังทลายต่ำกว่าแรงดันพังทลายของซีเนอร์ไดโอดภายในออปแอมป์ดังนั้นก่อนที่แรงดันอินพุทจะมีขนาดที่จะทำให้ไดโอดในออปแอมป์เสียหาย ซีเนอร์ไดโอดที่เราต่อก็จะลัดวงจรไปก่อนแล้วจึงไม่มีกระแสลัดวงจรผ่านเข้าสู่ออปแอมป์ ส่วนรูป2.24b แสดงการใช้ไดโอดธรรมดาในการป้องกันแต่อาจให้ผลไม่ดีนัก รูป2.24c แสดงออปแอมป์ที่มีวงจรป้องกันภาคเอาท์พุทอยู่ภายในแล้ว

.A/D Converter

ทำได้โดยการใช้ไดโอดใช้เป็นคลิบเปอร์ (clippers) หรือลิมิตเตอร์ (limiter) โดยการอ้างถึงรูป 2.24 ไดโอด D_1 ตัดสัญญาณอินพุทที่ -0.6 v และ D_2 ตัดที่ +0.6 v สัญญาณมีค่าน้อยเกินไปสำหรับการไบอัสตรงของไดโอดทั้งคู่ ดังนั้นจึงไม่มีผลใดๆ เกิดขึ้น ไดโอดมีความต้านทานสูงเมื่อหยุดการทำงาน แต่อย่างไรก็ตาม สัญญาณขนาดใหญ่ทำให้ไดโอดทำงานและนำไฟฟ้า เมื่อสิ่งนี้เกิดขึ้นแรงดันที่เกินนั้นตกคร่อม R_1 ดังนั้นการแกว่งของเอาท์พุททั้งหมดถูกกำจัด 1.2 v peak-to-peak การกำจัดเช่นนี้อาจจะถูกนำไปใช้เมื่อสัญญาณมีขนาดใหญ่



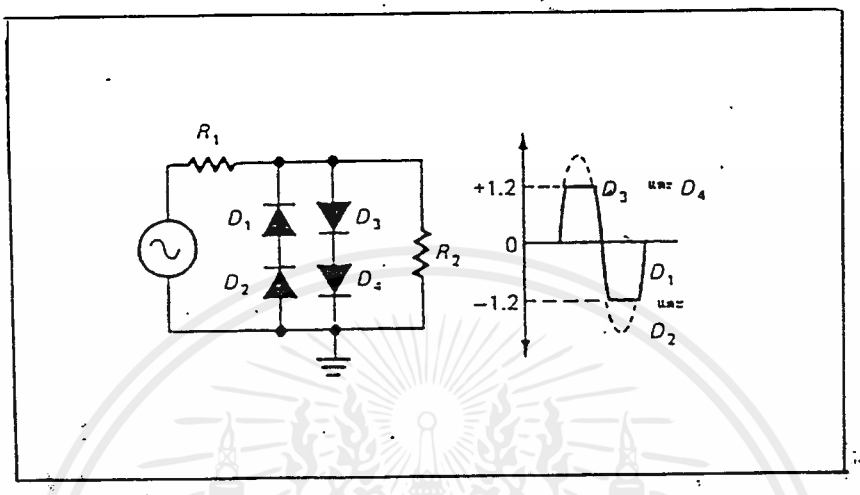
รูปที่ 2.25 ไดโอดถูกนำไปใช้เป็นคลิปเปอร์

คลิบเปอร์สามารถถูกนำไปใช้ในการกำจัดน็อยซ์พัลซ์ (noise pulse)ที่เข้ามารวมกับสัญญาณ ถ้าสัญญาณรบกวนพัลซ์มีมากเกินจุดตัด ดังนั้นจะถูกตัดออกหรือถูกจำกัด และผลของสัญญาณนี้มีสัญญาณรบกวนน้อยกว่าสัญญาณเริ่มต้น

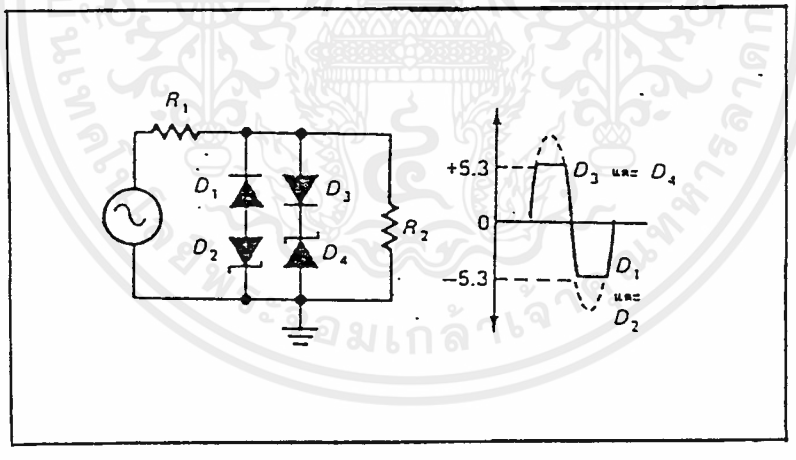
ไดโอด D_2 ตัดส่วนบวกสัญญาณในรูป 2.24 แรงดันสัญญาณ (signal voltage) เริ่มต้นด้วยการเพิ่มจาก 0v ในตอนแรกยังไม่มีอะไรเกิดขึ้น เมื่อแรงดันสัญญาณถึง 0.6v D_2 ทำงานและเริ่มนำไฟฟ้า ความต้านทานของ D_2 น้อยกว่า R_1 มากดังนั้นตัวต้านทาน R_1 รับแรงดันของแหล่งจ่ายที่มีค่าเกิน 0.6 v ต่อมาเมื่อกระแสสลับทางลบเริ่มต้นเข้ามาในตอนแรกยังไม่มีอะไรเกิดขึ้น เมื่อสัญญาณมาถึง -0.6 v D_1 ทำงานและนำไฟฟ้า R_1 รับค่าแรงดันสัญญาณที่เกิน -0.6 v การแกว่งของเอาท์พุททั้งหมดคือความแตกต่างระหว่าง +0.6 v และ -0.6 v หรือ 1.2 v peak- to-peak เจอเมเนียมไดโอดทำงานที่ 0.2 v และมีการแกว่งทั้งหมด 0.4v peal-to-peak ถ้าถูกนำไปใช้ในวงจรคลิบเปอร์

จุดที่ตัดสามารถถูกเปลี่ยนให้แรงดันสูงขึ้นโดยการใช้ไดโอดต่ออนุกรม ตัวอย่างในรูป 2.25 ต้องการ 0.6+0.6 หรือ 1.2v สำหรับการทำให้ D_3 และ D_4 ทำงาน สังเกตได้ว่าจุดที่ตัดทางด้านบวกแสดงบนกราฟที่ +1.2v ในทางที่คล้ายกัน D_1 และ D_2 ทำงานเมื่อการแกว่งของสัญญาณถึง -1.2 v ดังนั้นสัญญาณเอาท์พุทถูกจำกัดการแกว่งทั้งหมด 2.4v peak-to-peak แรงดันที่ถูกตัดสูงกว่านี้ สามารถทำได้โดยการใช้ซีเนอร์ไดโอดซึ่งแสดงในรูป 2.26 สมมุติว่า D_2 และ D_4 คือ 4.7 v ซีเนอร์ สัญญาณทางบวกที่ถูกตัด

ที่+5.3v เพราะ 4.7v ทำให้ D_4 ทำงานและ +0.6v ทำให้ D_3 ทำงานไดโอด D_1 และ D_2 ตัดไฟกระแสสลับทางด้านลบที่ -5.3v สัญญาณเอาท์พุททั้งหมดในรูป 2.25 ถูกจำกัดถึง 10.6v



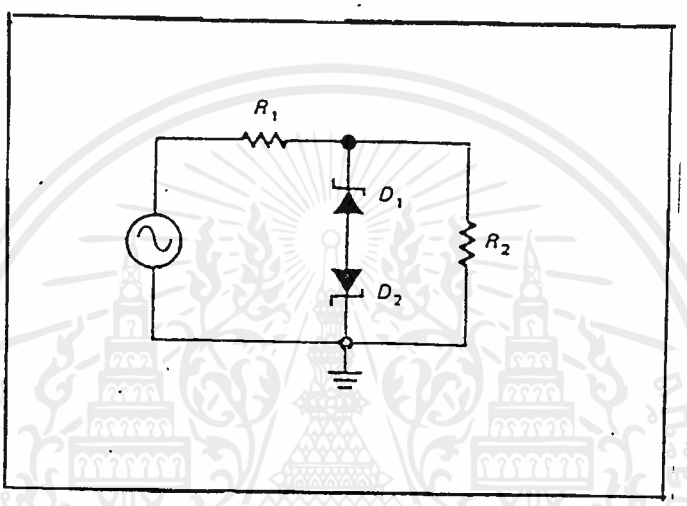
รูปที่ 2.26 การตัดค่าที่สูงขึ้น



รูปที่ 2.27 การใช้ซีเนอร์ไดโอดเพื่อการตัดที่สูงขึ้น

เมื่อซีเนอร์ไดโอดถูกไบแอสตรงจะแสดงเหมือนกับซิลิกอนเรคติฟายเออร์ไดโอดซึ่งเริ่มนำไฟฟ้าที่ 0.6v ดังนั้นวงจรในรูป 2.26 สามารถแก้ไขได้โดยการใช้ซีเนอร์

ไดโอด 2 ตัวต่อกลับกันซึ่งแสดงดังรูป 2.27 ถ้ากระแสกำลังไหลขึ้น ดังนั้นซีเนอร์ไดโอดทางด้านล่างจะรับ 0.6 v และซีเนอร์ทางด้านบนจะรับแรงดันของตัวเอง เมื่อกระแสกำลังไหลลง ซีเนอร์ไดโอดด้านบนจะรับ 0.6v และซีเนอร์ทางด้านล่างจะรับแรงดันของตัวเอง ตัวอย่างเช่น ถ้าใช้วงจร 1N4733 (อุปกรณ์ 5.1 v) 2 ตัว การแกว่งของเอาท์พุทถูกจำกัดถึง 5.1+0.6= 5.7 v peak voltage หรือ 11.4 v pea -to-peak



รูปที่ 2.28 คลิปเปอร์ที่ตัดได้สูงขึ้นโดยการแก้ไขบางส่วน

บทที่3
ขั้นตอนการออกแบบ

ในการออกแบบเครื่องตรวจวัดสัญญาณไฟฟ้ากระแสสลับ จะแบ่งการออกแบบเป็น 2 ส่วนใหญ่ๆคือ ส่วนวงจร และ ส่วนโปรแกรม ซึ่งการออกแบบส่วนวงจรจะรวมไปถึงการเลือกใช้วัสดุอุปกรณ์ต่างๆ เพื่อให้เหมาะสมกับการทำงาน และส่วนโปรแกรมจะเป็นการออกแบบวิธีการประเมินผลจากข้อมูลที่ได้รับ และกระบวนการทางข้อมูลต่างๆเขียนโปรแกรม เพื่อควบคุมการทำงานของเครื่องวัดให้ทำงานได้ถูกต้อง และมีประสิทธิภาพ

3.1 การออกแบบวงจร (Hardware design)

ในการออกแบบวงจรเพื่อใช้งานจะต้องทราบแนวทางและลักษณะการทำงานของวงจร และขั้นตอนการทำงานของวงจร ซึ่งขั้นตอนการทำงานของเครื่องวัดจะเป็นดังนี้

- ตัววัดรูปคลื่นจะมีการเริ่มและหยุดการวัดเป็นช่วงๆ โดยที่การสั่งงานจะเป็นหน้าที่ของไมโครคอนโทรเลอร์ ซึ่งจะใช้การอินเทอรัพท์และไทเมอร์ เข้ามาช่วยกำหนดระยะเวลาการวัด และ การประมวลผล
- การวัดข้อมูลจะวัดมาจากสายเพาวเวอร์ไลน์ ซึ่งเป็นสัญญาณเชิงความถี่ (Analog Signal) และนำมาเปลี่ยนเป็นสัญญาณเชิงตัวเลข เพื่อนำไปใช้ในการประมวลผล และ จัดเก็บเป็นข้อมูลต่อไป
- ข้อมูลที่ได้จะถูกเก็บอยู่ภายในหน่วยความจำข้อมูล (RAM) ซึ่งการเขียนและอ่านข้อมูลจากหน่วยความจำ จะถูกควบโดยไมโครคอนโทรเลอร์ และนอกจากนี้ยังเป็นตัวควบคุมตำแหน่งข้อมูลในหน่วยความจำ ให้มีจังหวะการเขียนอ่านที่เหมาะสม

- นอกจากนี้ข้อมูลภายในหน่วยความจำ สามารถถ่ายโอนข้อมูลจากเครื่องวัดไปเก็บไว้เป็นไฟล์ข้อมูลภายในคอมพิวเตอร์ทั่วไป โดยใช้พอร์ตอนุกรมของคอมพิวเตอร์ และ ไมโครคอนโทรลเลอร์

จากขั้นตอนข้างต้นเราสามารถแยกวงจรออกเป็นส่วนหลักได้ 4 ส่วนคือ ส่วนประมวลผล ส่วนตรวจวัดสัญญาณ ส่วนเก็บข้อมูล และส่วนการติดต่อสื่อสาร ซึ่งส่วนต่างๆสามารถนำมาประกอบเป็นวงจรแสดงได้ดังรูป 3.1 โดยแต่ละส่วนจะมีความสำคัญต่างกันออกไป แต่จะมีการทำงานที่สัมพันธ์กันเพื่อให้การทำงานสามารถทำได้เต็มที่และถูกต้อง การเลือกใช้อุปกรณ์ภายในวงจรก็เป็นหลักสำคัญของการออกแบบวงจรด้วย เพราะการเลือกอุปกรณ์ที่ดีและเหมาะสม ก็จะช่วยทำให้การออกแบบสามารถทำได้ง่ายขึ้น

3.1.1 การเลือกอุปกรณ์ภายในวงจร

การเลือกใช้อุปกรณ์ภายในวงจรที่ดีจะช่วยให้การออกแบบทำได้ง่าย และการทำงานของวงจรสามารถเป็นไปได้ด้วยดี อีกทั้งยังช่วยให้ข้อมูลที่ได้จากการวัดและการประมวลผลมีคุณภาพมากขึ้น โดยมีวิธีการเลือกอุปกรณ์ดังนี้

ส่วนประมวลผล

เป็นส่วนที่มีความสำคัญส่วนหนึ่งของวงจร ทำหน้าที่ควบคุมการทำงาน และจัดระเบียบภายในระบบ เป็นตัวส่งสัญญาณการวัดสัญญาณ การจัดเก็บข้อมูลลงในหน่วยความจำ และยังทำหน้าที่ติดต่อสื่อสารกับคอมพิวเตอร์ เพื่อนำข้อมูลไปเก็บไว้เป็นไฟล์

ในการเลือกตัวประมวลผลเพื่อให้ถูกต้องตามต้องการ เราจะต้องคำนึงถึงคุณสมบัติต่างๆดังต่อไปนี้

1. ความเร็ว ต้องมีความเร็วในการทำงานที่มากกว่า และสอดคล้องกับการทำงานของตัวแปลงสัญญาณเอดีซี เพราะเป็นตัวสั่งงานการเปลี่ยนตำแหน่งของหน่วย

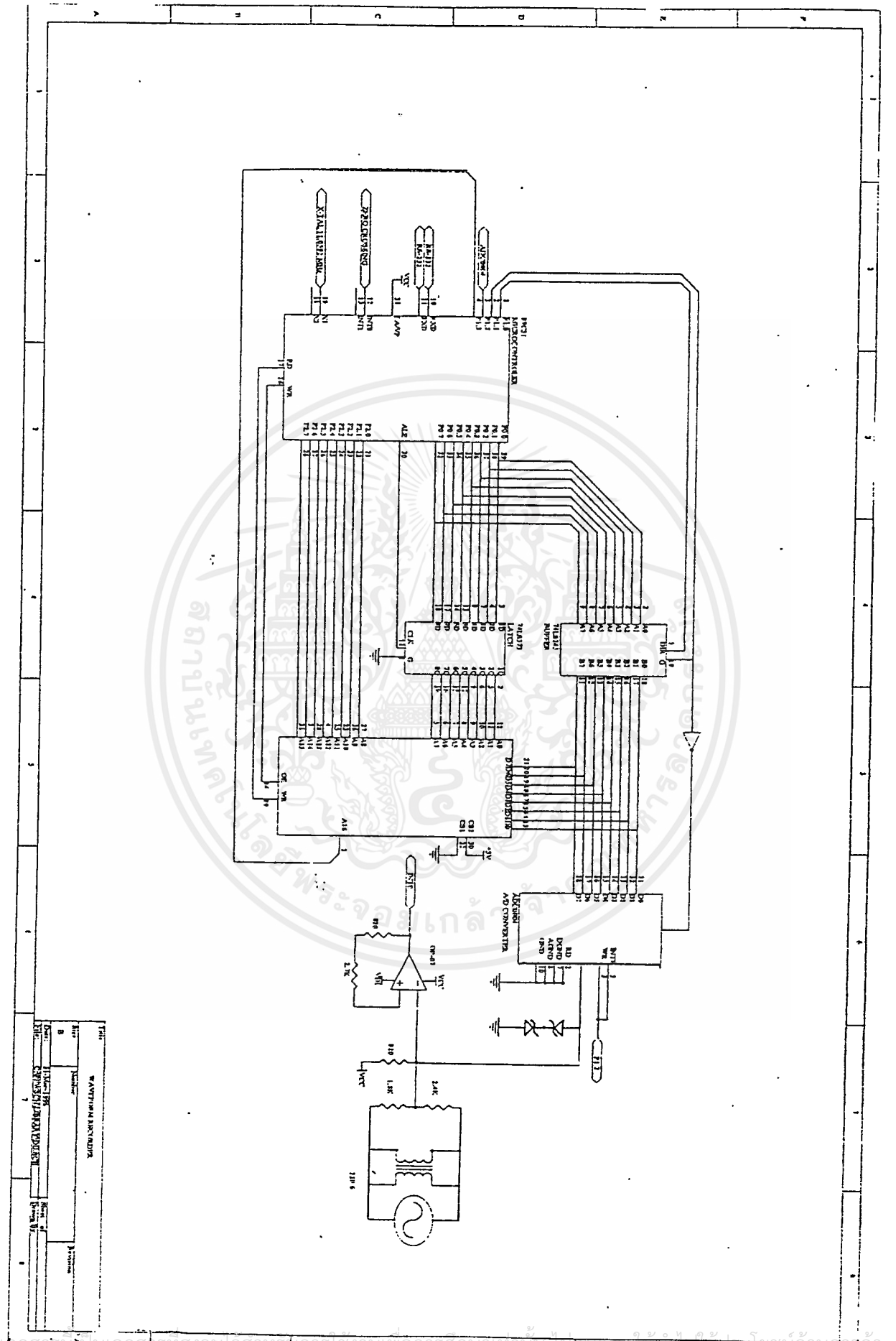
ความจำเพื่อใช้ในการเก็บข้อมูล (ถ้าต้องการใช้ตัวแปลงเอดีซีที่มีความเร็วสูงมาก เราต้องออกแบบส่วนเลือกตำแหน่งของหน่วยความจำต่างหาก) นอกจากนี้ยังทำหน้าที่ในการประมวลผลข้อมูลที่เก็บภายในหน่วยความจำ ถ้ามีความเร็วในการทำงานสูงก็สามารถวัดสัญญาณข้อมูลได้อย่างต่อเนื่องยิ่งขึ้น

2. จำนวนพอร์ตในการทำงาน ภายในเครื่องวัดสัญญาณไฟกระแสสลับ จะมีส่วนที่ต้องควบคุมมากมายเช่น เลือกตำแหน่งของหน่วยความจำ ควบคุมการทำงานของการแปลงสัญญาณ การเชื่อมต่อกับคอมพิวเตอร์ เป็นต้น ถ้าเลือกชนิดที่มีขาใช้งานน้อยจะต้องต่ออุปกรณ์เพิ่มเติม หรือถ้ามากไปก็จะมีราคาแพงขึ้น ดังนั้นต้องเลือกให้พอดีกับการใช้งาน

3. การนำไปใช้งาน สามารถนำไปใช้งานได้ง่ายไม่ต้องต่ออุปกรณ์เพิ่มเติมมากมายนัก และสามารถต่อเข้ากับอุปกรณ์อื่นได้ไม่ยุ่งยาก เพราะถ้าต้องต่ออุปกรณ์เพิ่มเติมมากก็ต้องใช้พื้นที่มากวงจรรวมก็จะใหญ่ตามไปด้วย และยิ่งมีอุปกรณ์มากก็จะเปลืองพลังงานในการทำงานมาก

4. ราคาเหมาะสม ไม่แพงเกินควร เหมาะสมการหน้าที่การใช้งานและสภาวะเศรษฐกิจ

จากหลักการข้างต้นนี้ ไมโครคอนโทรลเลอร์ 89C51 เหมาะสมมาก คือเป็นไมโครคอนโทรลเลอร์ชิปเดี่ยวที่มีการประเมินผลขนาด 8 บิต มีหน่วยความจำโปรแกรมชนิด PEROM ขนาด 4K bytes ภายในตัวทำให้ไม่ต้องต่อหน่วยความจำโปรแกรมภายนอกเพิ่มเติม สามารถอ้างหน่วยความจำได้ถึง 64 K bytes มีพอร์ตการใช้งานอยู่ 4 พอร์ตเป็นแบบ 2 ทิศทาง สามารถรับสัญญาณอินเทอรัพท์ได้ 5 แหล่ง และมีไทเมอร์ให้ใช้งานอยู่ 2 ช่อง มีแหล่งกำเนิดความถี่ภายใน มีความเร็วในการทำงานประมาณ 1 ไมโครวินาทีต่อคำสั่ง (ทั้งนี้ขึ้นอยู่กับแหล่งกำเนิดความถี่ภายในและคำสั่ง) อีกทั้งยังมีราคาที่ถูกและหาซื้อง่าย มีสัญญาณการทำงานในระดับทีทีแอลทำให้การเชื่อมต่อกับอุปกรณ์ทั่วไปทำได้ง่าย ซึ่งคุณสมบัติต่างๆเหล่านี้เหมาะสมที่จะนำมาใช้มาก

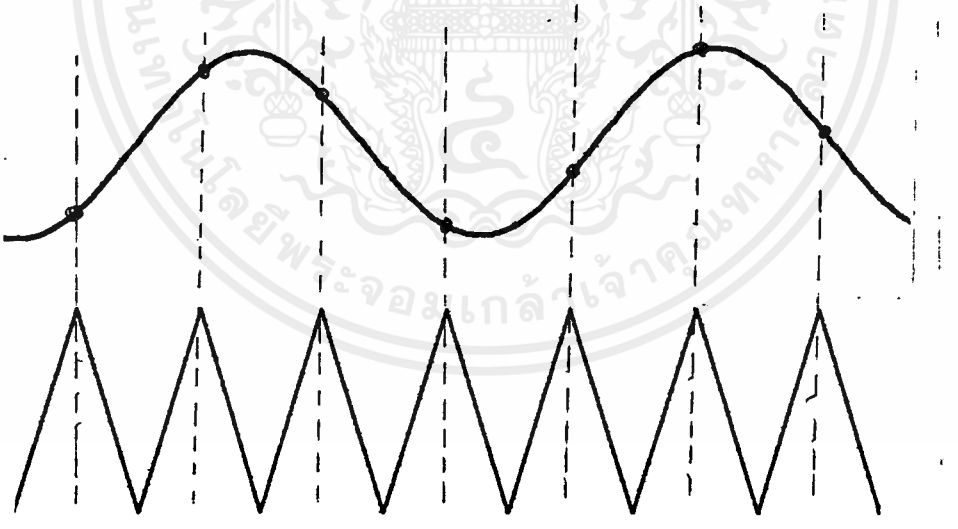


รูปที่ 3.1 แสดงวงจรต่างๆภายใน waveform recoder

ส่วนตรวจวัดสัญญาณ

เป็นส่วนที่ทำหน้าที่วัดสัญญาณไฟสลับที่เข้ามา แล้วนำไปแปลงสัญญาณโดยวิธีการต่างๆเป็นสัญญาณเชิงตัวเลขสำหรับการประมวลผล โดยจะใช้ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอล (A/D Converter) เป็นตัวเปลี่ยนสัญญาณ ซึ่งข้อมูลที่วัดได้จะมีความละเอียดถูกต้องมากน้อยเพียงใดนั้นจะขึ้นกับส่วนนี้ ถ้าตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลมีคุณสมบัติที่ดีคุณภาพของสัญญาณที่วัดได้ก็จะดีตามไปด้วย ทั้งนี้ขึ้นกับคุณสมบัติต่อไปนี้

1. **ความเร็วในการแปลงสัญญาณ** เป็นปัจจัยหนึ่งที่จะช่วยให้สามารถทำการวัดข้อมูลทำได้ละเอียดและถูกต้อง โดยความเร็วในการแปลงสัญญาณจะทำให้การวัดสัญญาณทางเวลาหรือความถี่ทำได้ดีเช่น การเปลี่ยนแปลงแบบรานเซียนท์แบบสั่น มีการเปลี่ยนแปลงอยู่ในช่วงความถี่ 0.5 - 5 MHz หรือใช้เวลา 5 - 20 ไมโครวินาที ถ้าตัวแปลงสัญญาณมีความเร็วที่ต่ำเกินไปก็จะไม่สามารถวัดสัญญาณเหล่านี้ได้เลย



รูปที่ 3.1.1 แสดงสัญญาณการแซมปลิ้งสัญญาณและสัญญาณข้อมูล

2. ความละเอียดในการแปลง เป็นส่วนที่ทำให้ค่าแรงดันที่วัดได้ถูกต้อง ถ้าความละเอียดสูงจะสามารถวัดการเปลี่ยนแปลงสัญญาณระดับต่ำๆเช่น สัญญาณรบกวน ได้ ในทางกลับกันถ้าความละเอียดต่ำเกินไปก็จะไม่สามารถวัดการเปลี่ยนแปลงได้เลย

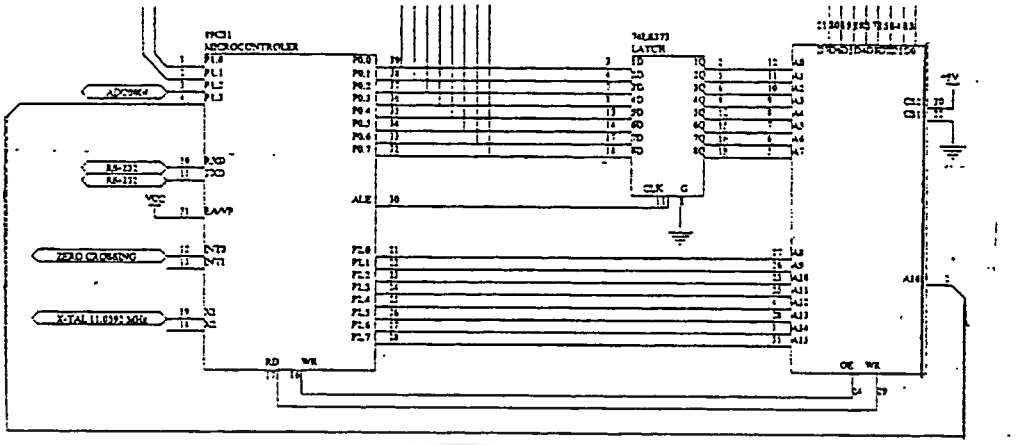
ADC0804 เป็นตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลแบบประมาณค่าต่อเนื่อง (successive-approximation) สามารถทำงานกับไมโครคอนโทรลเลอร์ทั่วไปหรือทำงานเพียงลำพังได้ มีความละเอียด 8 บิตและเวลาในการแปลงประมาณ 100 ไมโครวินาที สามารถรับแรงดันอินพุทได้ไม่เกิน 0-5V. แรงดันอ้างอิงมีค่าเท่ากับ 2.5 V. และใช้สัญญาณแซมปลิ้งได้ในช่วงความถี่ช่วง 100kHz - 1460kHz และมีขา RD , WR , CS และ INTR เพื่อง่ายต่อการสั่งงาน

3.1.2 การออกแบบส่วนต่างๆภายในวงจร

เมื่อเราได้อุปกรณ์ตามต้องการแล้ว ขั้นต่อไปคือการนำส่วนต่างๆในวงมาต่อเข้าด้วยกัน เนื่องจากในอุปกรณ์บางตัวมีระดับสัญญาณที่แตกต่างกัน บางชนิดต้องใช้สัญญาณภายนอกมาสั่งงานหรือควบคุมให้มีการทำงานเป็นไปตามต้องการ ซึ่งการออกแบบจะแบ่งเป็นส่วนต่างๆดังนี้

การเชื่อมต่อไมโครคอนโทรลเลอร์และหน่วยความจำ

ในการการเชื่อมต่อไมโครคอนโทรลเลอร์ 89C51 และหน่วยความจำเข้าด้วยกันจะใช้พอร์ต 0 ในการเลือกตำแหน่งของหน่วยความจำและรับส่งข้อูลกับหน่วยความจำและพอร์ต 2 สำหรับตำแหน่งของหน่วยความจำที่สูงขึ้น โดยไมโครคอนโทรลเลอร์จะเป็นตัวส่งสัญญาณการเขียนและอ่านไปยังหน่วยความจำ และเนื่องจากพอร์ต 0 ถูกใช้งาน 2 หน้าที่ ดังนั้นเราจะใช้การแลตช์มาช่วยในการเลือกหน้าที่ โดยจะใช้ตัวแลตช์เบอร์ 74LS373 มาช่วย ซึ่งการต่อจะเป็นดังรูปที่ 3.1.2.1

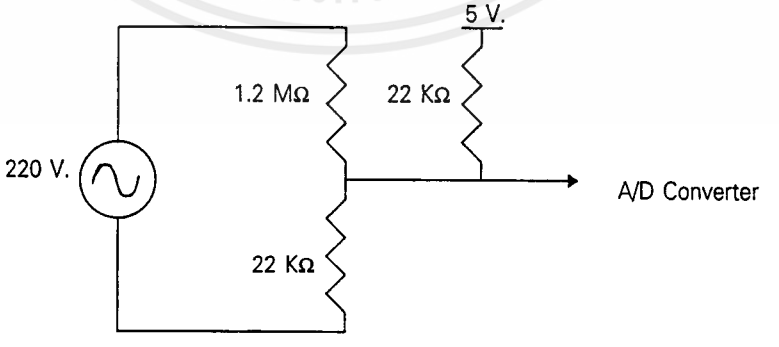


รูปที่ 3.1.2.1 การเชื่อมต่อไมโครคอนโทรลเลอร์และหน่วยความจำ

วงจรส่วนตรวจวัด

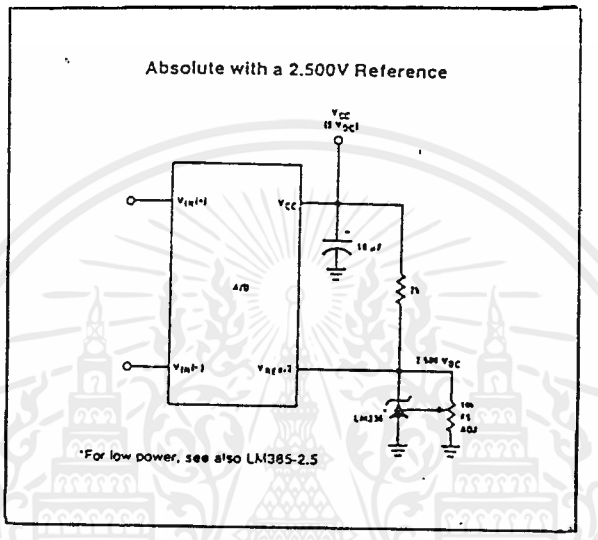
จะใช้ ADC0804 เป็นตัวแปลงข้อมูล ซึ่งง่ายต่อการเชื่อมต่อกับไมโครคอนโทรลเลอร์และมีอัตราการแปลงที่สูง โดยจะมีโดยวงจรส่วนต่างๆจะแสดงดังนี้

แรงดันอินพุท เนื่องจากสัญญาณที่เราจะทำการวัดเป็นสัญญาณขนาด 220V. ซึ่งมีค่าที่สูงมาก ดังนั้นเราจึงต้องทำการลดทอนสัญญาณลงมา โดยขั้นแรกจะใช้วงจรแบ่งแรงดันลดค่าแรงดันลงมาให้ได้ค่าประมาณ 4Vp-p เนื่องจาก ADC ใช้ได้กับแรงดันในช่วง 0-5V. เท่านั้น ดังนั้นจึงต้องต่อตัวต้านทานเพื่อแบ่งแรงดันจากไฟเลี้ยงวงจรมาเป็นแรงดันออฟเซท ซึ่งวงจรจะแสดงได้ดังรูป



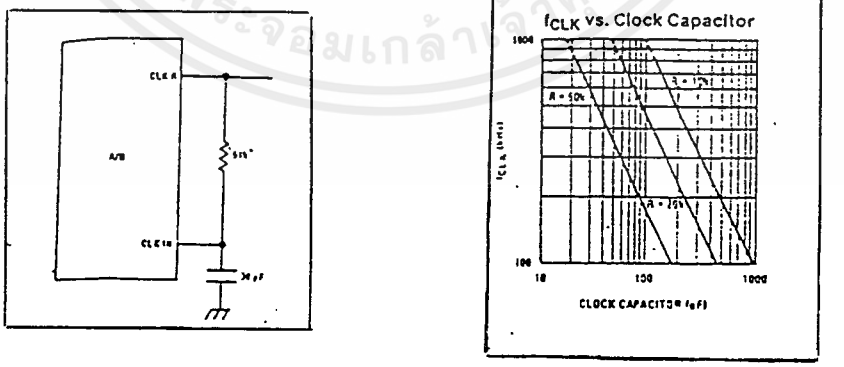
รูปที่ 3.1.2.2 การแปลงระดับสัญญาณโดยวิธีการแบ่งแรงดันและออฟเซท

แรงดันอ้างอิง ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอล ADC0804 จะต้องมีแรงดันอ้างอิงเพื่อใช้เปรียบเทียบค่าสำหรับการแปลง โดย ADC0804 ต้องการแรงดันอ้างอิงขนาด 2.5V ซึ่งจะใช้วงจรแบ่งแรงดันทำการแบ่งแรงดันจากแหล่งจ่ายไฟลงมา และใช้เรคกูเรเตอร์ LM336 เป็นตัวรักษาระดับของแรงดันอ้างอิง ซึ่งจะประกอบเป็นวงจรได้ตามรูป



รูปที่ 3.1.2.3 การต่อแรงดันอ้างอิงโดยใช้วงจรแบ่งแรงดัน และ LM 336

สัญญาณนาฬิกา ใช้การต่อ R และ C กับ ADC เพื่อใช้แหล่งกำเนิดสัญญาณภายใน ซึ่งความสัมพันธ์ของ R และ C ดูได้จากรูป 3.6 โดยค่า R และ C ที่ใช้จะมีค่าเท่ากับ 10 K และ 100pF ตามลำดับ

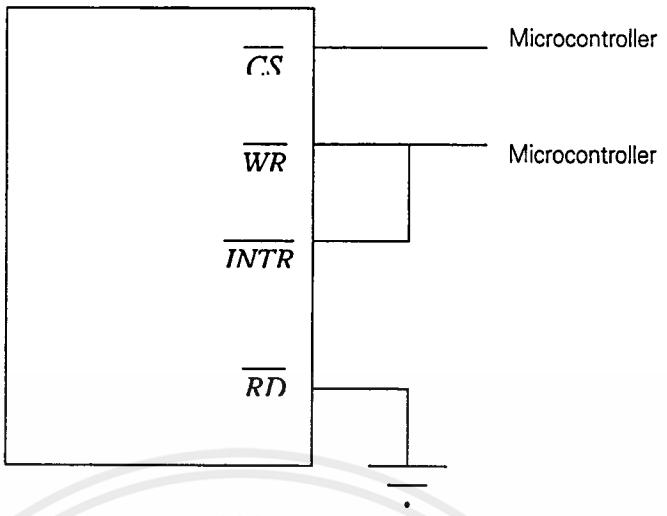


รูปที่ 3.1.2.4 การต่อสัญญาณนาฬิกาให้กับ ADC 0804 และ กราฟแสดงความสัมพันธ์ระหว่าง R และ Cต่อความที่ในการแซมปลิ้ง

สัญญาณควบคุมต่างๆ การทำงานของ ADC0804 นั้น ต้องมีสัญญาณมาควบคุมการแปลงค่าหรือการส่งผ่านค่า โดยADC0804จะมีขาในการบอกสถานะและสั่งงานดังนี้

1. $\overline{CS}$ เป็นขาที่ใช้เลือกชิพ เมื่อขานี้มีแรงดันศูนย์ ADC0804 ก็สามารถทำงานได้
2. $\overline{RD}$ เนื่องจากภายใน ADC0804 จะมีรีจิสเตอร์เก็บข้อมูลที่แปลงได้ไว้ภายในตัวเอง ซึ่งข้อมูลนี้จะออกมาก็ต่อเมื่อสัญญาณที่ขา $\overline{RD}$ มีระดับแรงดันเท่ากับศูนย์
3. $\overline{WR}$ เป็นขาสั่งเริ่มการทำงานของ ADC0804 ในการแปลงค่าสัญญาณที่เข้ามา เมื่อสถานะเดิมของขา $\overline{WR}$ มีสภาวะลอจิกสูง การเริ่มการทำงานสามารถทำได้โดยการจ่ายจ่ายพัลซ์ศูนย์ให้กับขา $\overline{WR}$
4. $\overline{INTR}$ ใช้บอกสถานะการทำงานของตัวแปลง โดยที่ขาอินเทอรัพท์นี้จะส่งสัญญาณพัลซ์ศูนย์ ออกมาเมื่อการแปลงข้อมูลสิ้นสุด
5. *ขาข้อมูล 8 บิต* เป็นขาข้อมูลที่ได้จากการแปลง จะมีสภาวะอิมพีแดนซ์สูงขณะที่ขา $\overline{RD}$ มีค่าลอจิกหนึ่ง

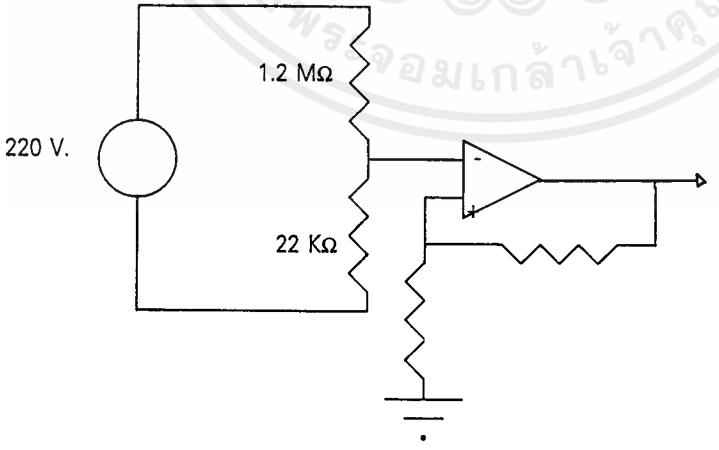
ในการต่อขาสัญญาณของ ADC0804 นั้น จะต่อให้ทำงานเป็นแบบอิสระ (Free Running) จะใช้ขา $\overline{INTR}$ ต่อเข้ากับขา $\overline{WR}$ ซึ่งสัญญาณจากขา $\overline{INTR}$ นี้จะส่งเป็นพัลซ์ออกมาเมื่อการแปลงสัญญาณนั้นเสร็จสิ้น โดยสัญญาณนี้จะส่งไปยังขา $\overline{WR}$ เพื่อกระตุ้นการทำงานของ ADC0804ต่อไปเรื่อยๆ ดังนั้น ADC0804 จะทำงานได้ตลอดเวลาโดยไม่ต้องส่งสัญญาณควบคุมทุกครั้ง และใช้ขา $\overline{CS}$ เป็นตัวควบคุมการทำงานของ ADC0804 การเริ่มการทำงานจะต้องส่งสัญญาณพัลซ์ศูนย์ที่ขา $\overline{WR}$ ก่อน หลังจากนั้น ADC0804 จะทำงานต่อไปโดยใช้สัญญาณการสั่งงานจากตัวมัน ด้วยวิธีนี้จะช่วยประหยัดเวลาการทำงานลงอย่างมาก



รูปที่ 3.1.2.5 แสดงการต่อขาต่างๆของ ADC0804 เพื่อใช้งาน

วงจร zero crossing

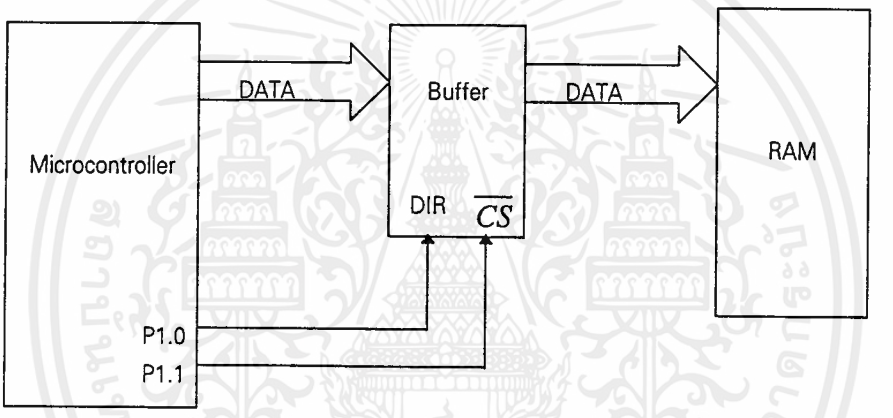
การจำกัดระยะเวลาการวัดข้อมูลให้ได้ 1 ลูกคลื่นนั้น สามารถทำได้โดยใช้วงจรนี้ ซึ่งสัญญาณอินพุตจะนำจากพาวเวอร์ไลน์ และถูกลดระดับลงมาด้วยวงจรแบ่งแรงดัน แล้วต่อเข้ากับวงจรตรวจระดับศูนย์ซึ่งสร้างจากออปแอมป์ที่ต่อเป็นวงจรป้อนกลับแบบบวก (Positive Feedback) การคำนวณได้แสดงไว้ในภาคทฤษฎี สัญญาณที่ได้ออกมาจะออกมาทั้งฝั่งบวกและลบ ดังนั้นจะใช้ ไดโอด เป็นตัวจำกัดทิศทาง สัญญาณที่ได้จะนำไปเป็นสัญญาณอินเทอรัพท์ให้กับไมโครคอนโทรลเลอร์ ซึ่งวงจรสามารถแสดงดังรูป



รูปที่ 3.1.2.6 การต่อวงจร Zero Crossing

วงจรช่วยการทำงานต่างๆ จะเป็นส่วนช่วยในการทำงานของวงจรหลักให้สามารถทำงานได้ โดยส่วนนี้จะแบ่งเป็น

บัฟเฟอร์ จะใช้ในการแยกข้อมูลในส่วนประมวลผลและส่วนตรวจวัดออกจากกัน โดยจะใช้เบอร์ไอซี 74HCT245 ซึ่งเป็นบัฟเฟอร์แบบ 8 บิตแบบ 2ทิศทาง และมีสถานะอิมพีแดนซ์สูงเมื่อไม่ใช้งาน สภาวะของบัฟเฟอร์จะมีขาในการสั่งงานคือขา $\overline{CS}$ ใช้ในการเลือกไอซีใช้งาน และขา *DIR* ใช้กำหนดทิศทางการทำงานของไอซี ในการควบคุมจะใช้ไมโครคอนโทรลเลอร์ควบคุม ซึ่งจะต่อดังรูป



รูปที่ 3.1.2.7 แสดงการต่อบัฟเฟอร์กับไมโครคอนโทรลเลอร์

RS-232 ใช้ในการแปลงระดับสัญญาณลอจิกของ 89C51 ให้เป็นระดับของคอมพิวเตอร์เพื่อใช้ในการสื่อสารแบบอนุกรม โดยระดับสัญญาณ -3V ถึง -20V จะแทนลอจิก 1 และระดับ +3V ถึง +20Vจะแทนลอจิก 0

3.2 การออกแบบโปรแกรม (Software design)

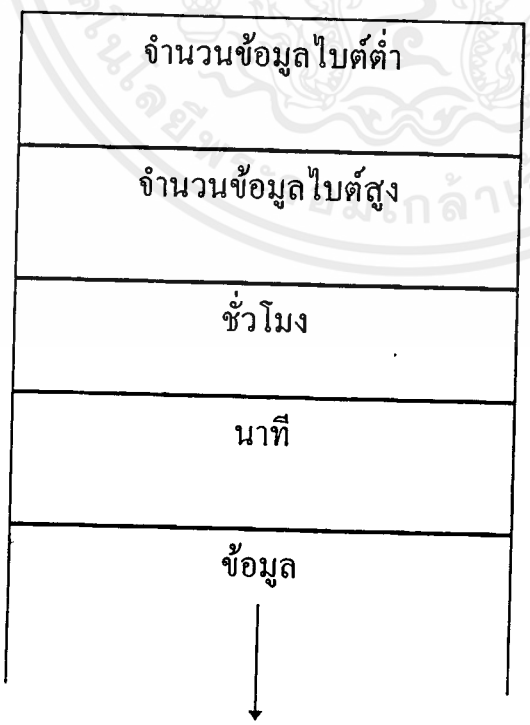
ในการเขียนโปรแกรมนั้นจะต้องทราบถึงขั้นตอนการทำงานของวงจร ซึ่งได้กล่าวไว้ในหัวข้อการออกแบบวงจรแล้ว นอกจากนี้จะต้องทราบไทมิ่งไดอะแกรมการทำงาน การใช้สัญญาณควบคุมการเริ่มการทำงานของอุปกรณ์ที่ใช้งาน เพื่อให้วงจรมีการทำงานที่ถูกต้อง โดยในการควบคุมวงจรนั้นจะใช้ไมโครคอนโทรลเลอร์ 89C51 ควบคุม ซึ่งภายในจะมี PEROM สำหรับเก็บข้อมูลโปรแกรมการทำงานของวงจร ในการทำงานของโปรแกรมจะเป็นขั้นตอนดังนี้

1. การเริ่มทำงานของวงจรจะต้องตั้งค่ารีจิสเตอร์ต่างๆที่จะใช้งานภายในไมโครคอนโทรลเลอร์ก่อน เช่น ไทเมอร์ อินเทอรัพท์ ค่าสถานะการทำงานเริ่มต้นของอุปกรณ์เสียก่อน โดยขาสั่งงานของไมโครคอนโทรลเลอร์จะแบ่งเป็น
   - ขา p1.0 ใช้ควบคุมทิศทางการทำงานของบัฟเฟอร์
   - ขา p1.1 เลือกการติดต่อของไมโครคอนโทรลเลอร์กับหน่วยความจำ (ลอจิก 0) และ ตัวแปลงอะนาลอกเป็นดิจิตอลกับหน่วยความจำ (ลอจิก 1)
   - ขา p1.2 ใช้ในการส่งสัญญาณเริ่มการทำงานให้กับตัวแปลงอะนาลอกเป็นดิจิตอล โดยสถานะทั่วไปจะเป็นลอจิก 1 การส่งสัญญาณจะส่งเป็นพัลซ์ศูนย์ให้กับขา $\overline{WR}$ ของตัวแปลง
   - ขา p1.3 เนื่องจากหน่วยความจำที่ใช้มีขนาด 128K แต่ 89C51 สามารถอ้างได้เพียง 64K ดังนั้นจึงใช้ขานี้มาช่วยในการอ้างตำแหน่ง
   - ขา $\overline{RD}$ และ $\overline{WR}$ เป็นขาสัญญาณการอ่าน/เขียนของหน่วยความจำซึ่งจะส่งออกอัตโนมัติโดยคำสั่ง MOVX

2. ในการรับคำสั่งเริ่มวัดจะทำโดยการรับสัญญาณอินเทอรัพท์จากวงจรตรวจระดับศูนย์เข้ามา โปรแกรมจะสั่งเริ่มการวัดสัญญาณโดยการส่งพัลซ์ไปยังขา

p1.2 และส่งลอจิก 1 ที่ขา p1.1 เพื่ออนุญาตการติดต่อระหว่างตัวแปลงอะนาลอกเป็นดิจิตอลกับหน่วยความจำ สำหรับการเก็บข้อมูล

3. ในระหว่างการเก็บข้อมูลตัวแปลงอะนาลอกเป็นดิจิตอลจะส่งสัญญาณบอกการแปลงข้อมูลเสร็จสิ้นแต่ละครั้ง เพื่อให้ไมโครคอนโทรลเลอร์ทำการเปลี่ยนตำแหน่งหน่วยความจำในการเก็บข้อมูล
4. ระยะเวลาการรับสัญญาณจะควบคุมด้วยไทเมอร์ โดยจะตั้งให้ระยะเวลาเท่ากับ 40*mS* หรือ 2 ลูกคลื่นเพื่อทำการตัดให้เหลือ 1 ลูกคลื่นต่อไป
5. เมื่อไทเมอร์นับครบจะส่งสัญญาณอินเทอรัพท์บอกหน่วยประมวลผล ไมโครคอนโทรลเลอร์จะส่งลอจิก 0 ที่ขา p1.1 เพื่อหยุดการทำงานของตัวแปลง และให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำได้สำหรับการประเมินผลในขั้นตอนต่อไป
6. ในการประเมินผลข้อมูลจะเริ่มจากการตัดรูปคลื่นให้เหลือเพียงรูปไซน์รูปเดียวเท่านั้น จากนั้นจะนำข้อมูลที่ได้ไปนับจำนวนเพื่อใส่รายละเอียดของการวัดไว้ภายในหน่วยความจำอีกครั้ง ซึ่งจะมีรูปแบบของข้อมูลที่เก็บอยู่ภายในหน่วยความจำจะแสดงดังรูป



รูปที่ 3.2.1 แสดงรูปแบบการเก็บข้อมูลภายในหน่วยความจำ

บทที่ 4
ผลการวิจัย

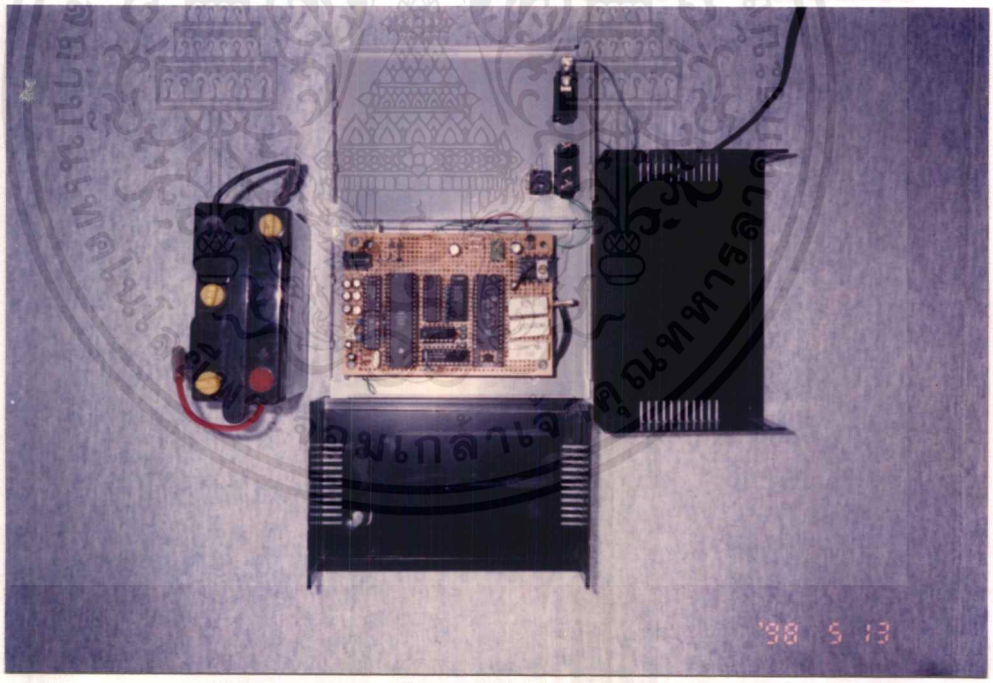
ก่อนที่จะทำการทดลองควรจะทำความรู้จักวิธีการใช้เครื่องวัดสัญญาณไฟสลับและโปรแกรมที่ใช้ร่วมกับเครื่องก่อนดังนี้

4.1 วิธีการใช้งานเครื่องวัดสัญญาณไฟสลับ

ภายในเครื่องนี้มีแบตเตอรี่อยู่ภายในสำหรับจ่ายไฟเลี้ยงภายในวงจร ดังนั้นจึงไม่ต้องต่อไฟเลี้ยงจากภายนอก สามารถเคลื่อนย้ายได้โดยข้อมูลในการวัดไม่สูญหาย โดยข้อมูลที่วัดได้ จะสามารถถ่ายโอนไปยังคอมพิวเตอร์เพื่อวาดเป็นภาพได้ และเครื่องนี้ยังสามารถตั้งฟังก์ชั่นการทำงานของเครื่องได้โดยคอมพิวเตอร์ การติดต่อกับคอมพิวเตอร์นั้นจะสามารถติดต่อได้ทั้งพอร์ต COM1 และพอร์ต COM2 ของคอมพิวเตอร์ การใช้งานเครื่องวัดนี้จะแบ่งออกเป็นการใช้งานตัวเครื่องและการใช้งานทางโปรแกรมซึ่งเป็นดังนี้

1.การใช้งานเครื่องวัด การใช้เครื่องวัดนี้สามารถทำได้ง่าย โดยตัวเครื่องแสดงได้ดังรูป

รูปที่ 4.1.1 แสดงตัวเครื่องวัดสัญญาณไฟสลับ



รูปที่ 4.1.2 ภาพแสดงภายในตัวเครื่องวัดสัญญาณไฟสลับ

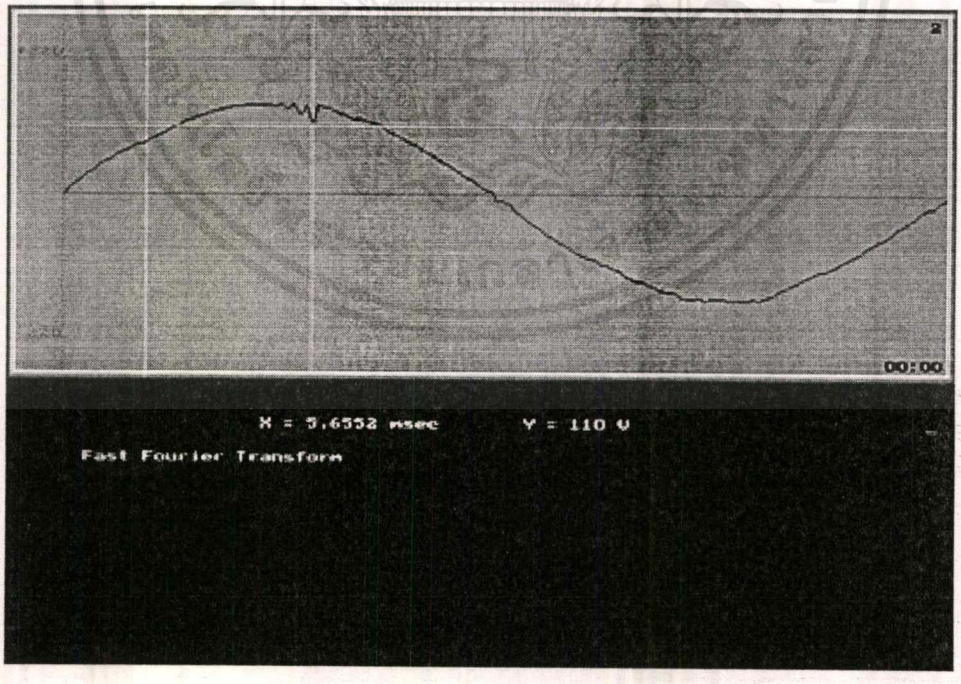
ฟังก์ชั่นการทำงานของโปรแกรมจะแสดงได้ดังนี้

2.1 ฟังก์ชั่นทางด้านขนาด จะเป็นการเปลี่ยนขนาดของรูปกราฟเพื่อความสะดวกในการสังเกตการเปลี่ยนแปลงของสัญญาณ ซึ่งแต่ละขนาดจะมีคุณสมบัติต่างๆ กันไป

D ⟹ Divide Size ขนาดเล็ก เป็นการนำข้อมูลที่ได้มาวาดเป็นขนาดย่อยหลายขนาด ซึ่งแสดงดังรูป 4.3 สามารถเลื่อนดูรูปอื่นๆได้ด้วยแป้นลูกศรจากแป้นพิมพ์ รูปที่สังเกตจะมีความสว่างกว่ารูปอื่นซึ่งรูปนี้สามารถขยายเป็นขนาดอื่นได้

A ⟹ Actual Size เป็นขนาดปกติของสัญญาณ รายละเอียดต่างๆจะคล้ายกับขนาดเล็ก แต่จะแตกต่างกันเพียงแค่ขนาดเท่านั้น

W ⟹ Wide Size เป็นการขยายขนาดจากขนาดปกติทางด้านกว้างออกไป ไม่สามารถเปลี่ยนไปดูภาพอื่นได้ (ถ้าจะเปลี่ยนภาพต้องเปลี่ยนเป็นขนาดอื่นก่อน) แต่จะมีการแกนบอกขนาดของสัญญาณทางด้านแรงดัน (แกนY) และเวลา (แกน X) ซึ่งสามารถเลื่อนดูค่าที่ตำแหน่งอื่นได้ด้วยแป้นลูกศร แสดงได้ดังรูป



รูปที่ 4.1.4 แสดงภาพขนาดกว้างของโปรแกรมโดยที่มีค่าบอกแกน X และ Y

2.2 ฟังก์ชั่นทางข้อมูล เป็นการเปิดปิดและเขียนไฟล์ข้อมูล โดยโปรแกรมนี้จะใช้กับไฟล์ข้อมูลเป็นรูปแบบเฉพาะ ซึ่งคำสั่งต่างๆจะแบ่งเป็น

L $\Rightarrow$ Load เป็นการถ่ายโอนข้อมูลจากเครื่องวัดขึ้นมาบนคอมพิวเตอร์ ซึ่งจะเก็บไว้ในรูปของไฟล์ชื่อ PRJ-XXXX.RWS (XXXX คือตัวเลขจำนวนเต็ม 0-9) โดยในการนำค่าเข้ามาโปรแกรมจะใช้วิธีตรวจสอบจำนวนข้อมูลของหน่วยความจำว่าเป็นศูนย์จึงจะหยุดการถ่ายโอนข้อมูล หลังจากนั้นตำแหน่งของหน่วยความจำภายในเครื่องวัดจะถูกตั้งค่าเป็น 0000 เพื่อเก็บค่าใหม่

O $\Rightarrow$ Open ใช้ในการเปิดดูข้อมูลที่เก็บมาจากการ Load เมื่อกดปุ่ม " O " บนแป้นพิมพ์จะมีหน้าต่างขึ้นมาให้ใส่หมายเลขข้อมูล (XXXX) ลงไป ถ้าข้อมูลนั้นไม่มีจะมีเสียงบอกข้อผิดพลาดดังขึ้น

2.3 ฟังก์ชั่นทางสถานะของเครื่องวัด ภายในเครื่องวัดนี้จะมีการเก็บข้อมูลต่างๆที่ช่วยในการประเมินผลไว้เช่น เวลา ค่าสัญญาณรบกวน เป็นต้น ค่าที่แตกต่างกันข้อมูลที่ได้ก็จะเปลี่ยนแปลงไปจากเดิม ฟังก์ชั่นทางสถานะของเครื่องวัดจะมีดังนี้

S $\Rightarrow$ Status ดูสถานะของเครื่องวัดสัญญาณว่าสถานะที่เครื่องทำงานอยู่นั้นเป็นอย่างไร เวลา จำนวนข้อมูลที่วัดได้ และค่าอ้างอิงมีค่าเท่าไร เพื่อใช้สังเกตและแก้ไขให้มีความถูกต้องต่อไป

H $\Rightarrow$ Threshold Value เป็นการตั้งค่าเทรชโฮลด์หรือค่าอ้างอิงที่ใช้ในการเปรียบเทียบข้อมูล ซึ่งค่านี้จะมีประโยชน์ในการช่วยขจัดสัญญาณที่ไม่ต้องการเช่น สัญญาณรบกวนต่างๆ ออกไป โดยค่ายิ่งมากการเก็บรูปคลื่นก็จะเก็บเพียงรูปที่มีการเปลี่ยนแปลงสูงเท่านั้น ดังนั้นควรเลือกค่าที่เหมาะสม ค่าที่ตั้งจะอยู่ในช่วง 0-255

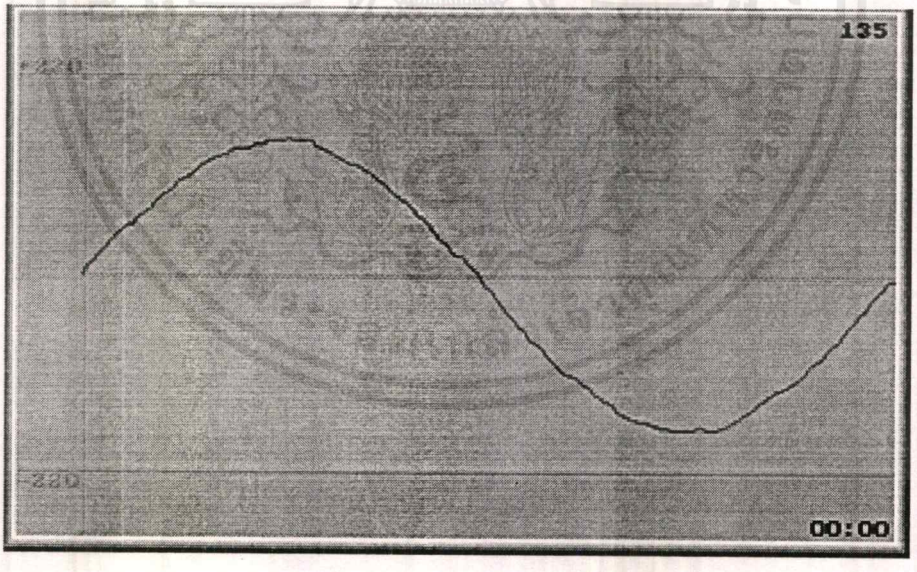
T $\Rightarrow$ Timer ใช้ตั้งนาฬิกาภายในเครื่องให้ตรงกับเวลาจริงขณะการวัด โดยสามารถตั้งเวลาได้ตั้งแต่ช่วงวินาทีจนถึงชั่วโมง

ฟังก์ชั่นต่างๆข้างต้นนี้จะเป็นการติดตั้งและใช้งานภายในเครื่องนี้ จะเป็นส่วนช่วยในการทำงานของเครื่องให้ครบถ้วนและสมบูรณ์และถูกต้องยิ่งขึ้น

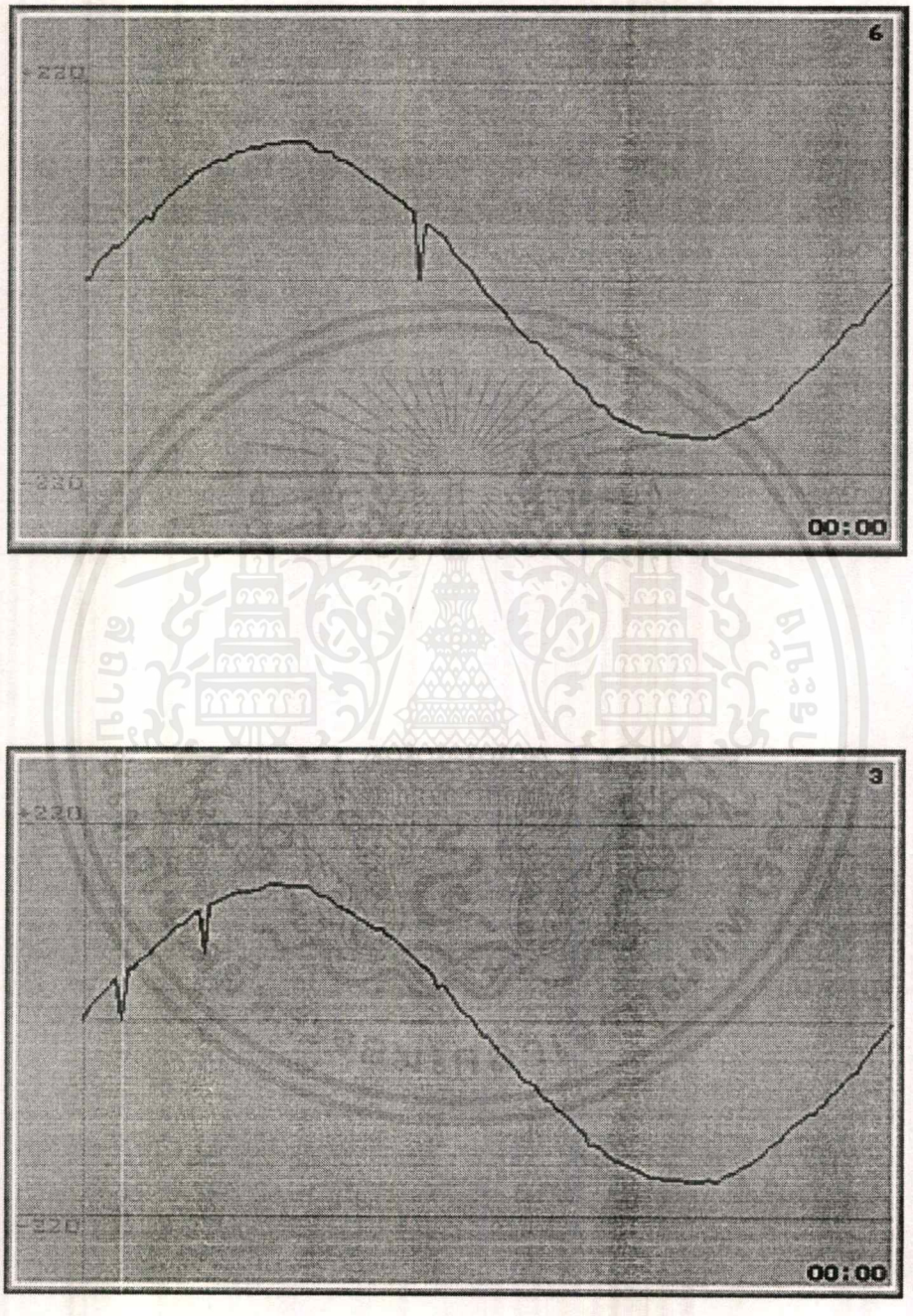
4.2 การทดลองและผลการทดลอง

การทดลองจะทำโดยการติดตั้งเครื่องวัดสัญญาณไว้บริเวณที่มีเครื่องจักรต่างๆที่จะทำการสังเกต แล้วนำเครื่องนี้มาถ่ายข้อมูลเก็บไว้เป็นผลการทดลองเพื่อทำการวิเคราะห์ต่อไป

ในหน่วยความจำของเครื่องจะมีข้อมูลที่เป็นแหล่งอ้างอิงอยู่ภายใน ซึ่งข้อมูลชุดนี้จะเป็นข้อมูลอ้างอิงในการเปรียบเทียบกับข้อมูลอื่น โดยการเปรียบเทียบจะเปรียบเทียบแบบไบต์ต่อไบต์ และใช้ค่าเทรชโฮลด์เป็นตัวกำหนดความแตกต่างของข้อมูล ซึ่งข้อมูลชุดนี้สามารถแสดงได้ดังรูป

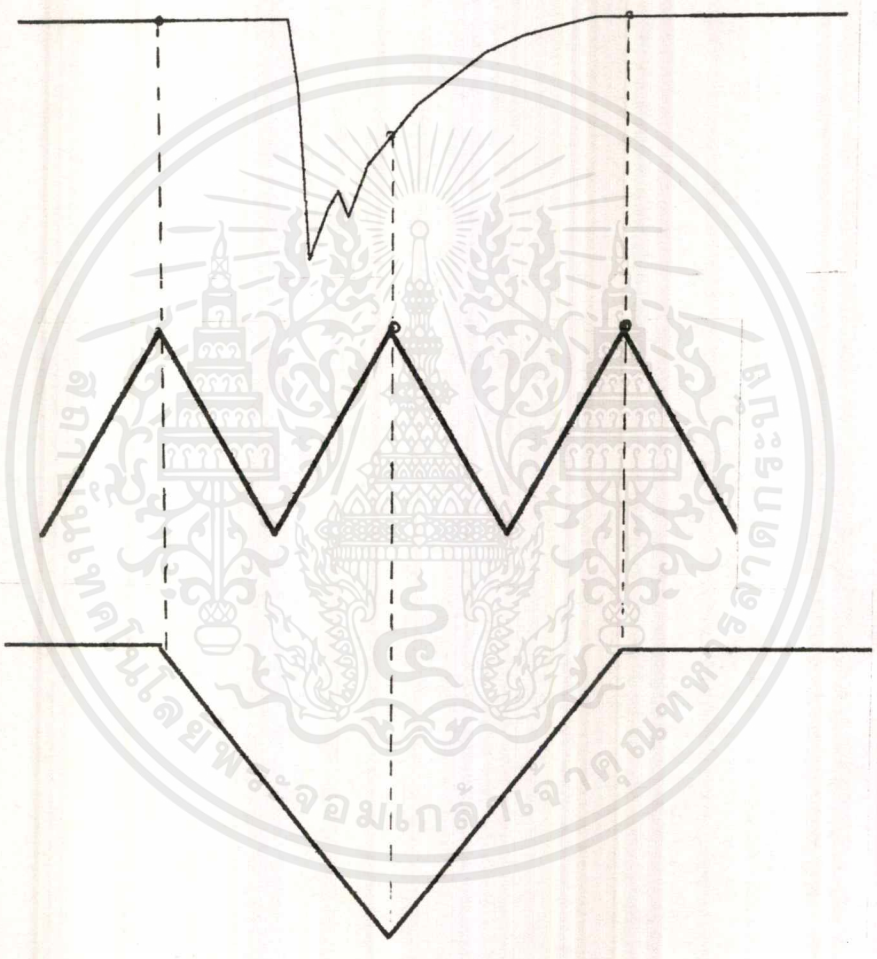


รูปที่ 4.2.1 แสดงรูปคลื่นที่วัดได้ขณะที่ยังไม่มีการทำงานของอุปกรณ์ต่างๆ



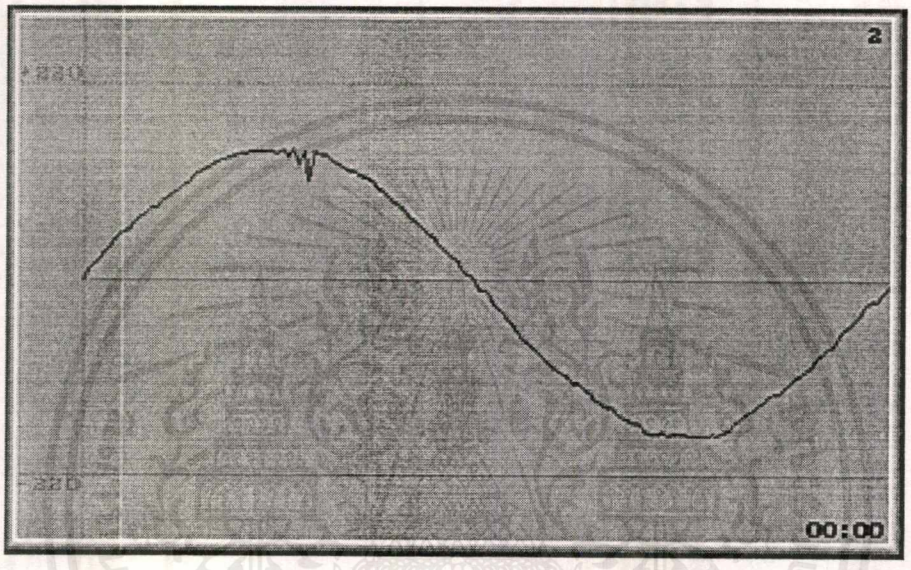
รูปที่ 4.2.2 ผลจากการทำงานของมอเตอร์ไฟฟ้าต่อสัญญาณไฟฟ้าสลับ

จากรูปที่ 4.2.2 แสดงให้เห็นถึงผลของการทำงานมอเตอร์ไฟฟ้าต่อสัญญาณไฟฟ้าสลับ จะเห็นว่าเกิดแรงดันทรานเซียนท์ขึ้นบนสัญญาณ รูปแบบที่ได้นั้นจะแตกต่างกับทฤษฎี ทั้งนี้เนื่องมาจากความสามารถในการแปลงค่าของตัวแปลงนั้นมีความเร็วที่ต่ำเกินไป ซึ่งทำให้รูปที่ได้นั้นจะเป็นดังรูปที่ 4.2.3

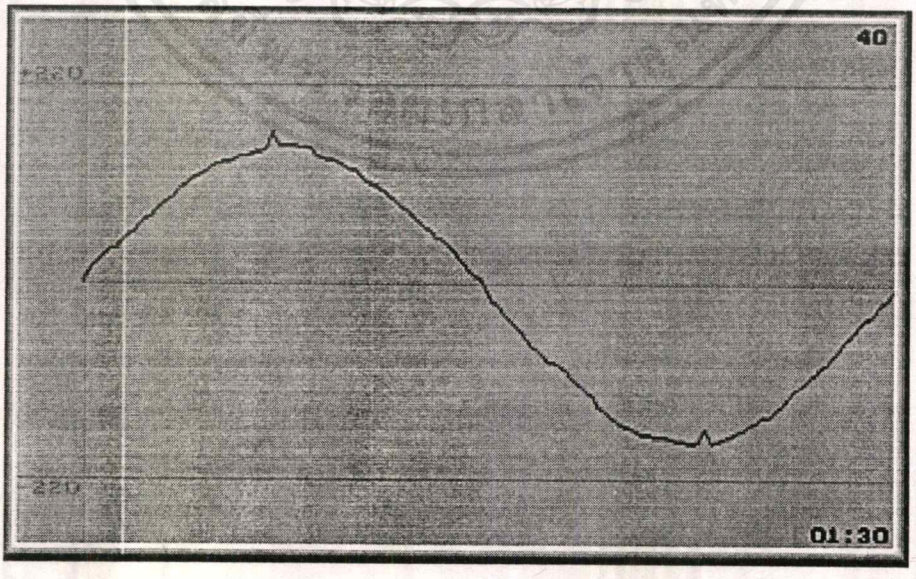


รูปที่ 4.2.3 แสดงผลของความเร็วในการแปลงต่อสัญญาณที่เกิดขึ้น

นอกจากนี้การทำงานของคอมพิวเตอร์ยังมีผลต่อสัญญาณไฟฟ้าด้วย สังเกตจากการตรวจวัดสัญญาณในขณะที่มีการใช้งานและไม่ใช้คอมพิวเตอร์ ซึ่งผลที่วัดได้จะเป็นดังรูป

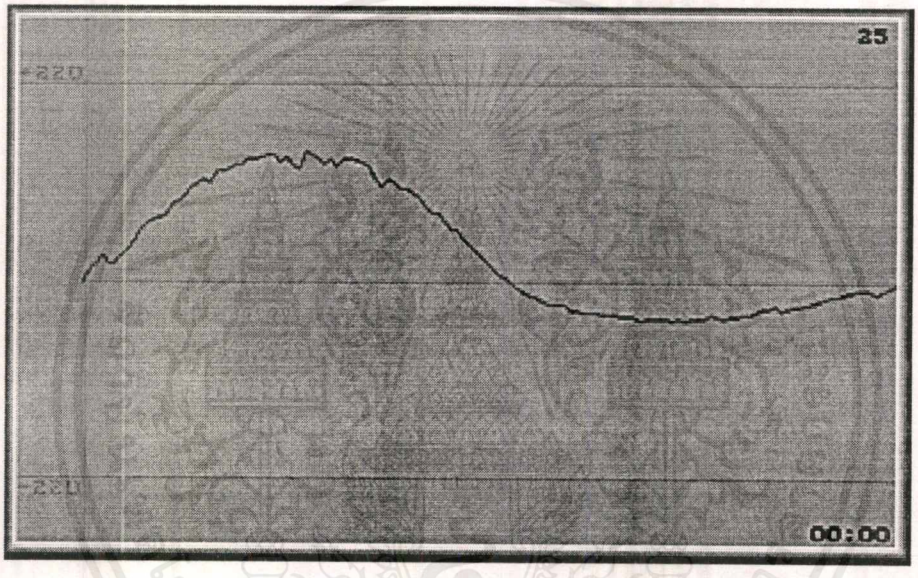


รูปที่ 4.2.4 สัญญาณขณะที่มีการเปิดคอมพิวเตอร์



รูปที่ 4.2.5 สภาพสัญญาณขณะใช้งานคอมพิวเตอร์

และในรูปที่ 4.2.6 จะเป็นผลมาจากการปิดสวิตช์ของหัววัดสัญญาณ โดยจะทำเหมือนกับสภาวะไฟตก จะเห็นว่าสัญญาณค่อยๆลดลงตามลำดับแต่สัญญาณที่วัดได้นี้อาจจะไม่ถูกต้องซึ่งตามจริงควรจะเป็นการเปลี่ยนแปลงของทรานเซียนท์แบบสั่น แต่จะเกิดปัญหาจากคุณภาพของตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลตามที่ได้กล่าวไว้ข้างต้น



รูปที่ 4.6 ผลของการสับสวิทช์ปิดสัญญาณไฟสลับ

บทที่5

สรุปผลการศึกษาและข้อเสนอแนะ

จากผลที่ได้เครื่องวัดสัญญาณไฟสลับนี้สามารถตรวจจับสัญญาณที่เกิดการเพี้ยนหรือมีการเปลี่ยนแปลงขณะเครื่องจักรอยู่ในสภาวะต่างๆโดยเครื่องนี้สามารถตรวจจับสัญญาณที่มีความเร็วในการเปลี่ยนแปลงไม่มากนัก (มากกว่า 0.5ms. ขึ้นไป) และในการเก็บสัญญาณจะไม่สามารถเก็บสัญญาณต่อเนื่องกันได้ เนื่องจากขั้นตอนในการประมวลผลต่างๆที่ต้องใช้เวลาพอสมควรในการทำงาน (เก็บไป1ลูกคลื่นแล้วเว้นไป2 ถึง 3ลูกคลื่น) ทำให้ไม่สามารถสังเกตสัญญาณที่มีการเปลี่ยนแปลงต่อเนื่องกัน หรือถ้าเกิดการเปลี่ยนแปลงของสัญญาณระหว่างประเมินผล จะไม่สามารถตรวจวัดได้ ทั้งนี้ขึ้นอยู่กับชนิดของตัวแปลงสัญญาณ และ ตัวประเมินผลด้วย ถ้าทั้งสองตัวนี้มีประสิทธิภาพสูง สัญญาณที่วัดได้ก็จะมีความถูกต้องและแม่นยำสูงตามไปด้วย

เราสามารถเพิ่มประสิทธิภาพการทำงานของเครื่องนี้ได้ โดยอาจจะเปลี่ยนชนิดของตัวแปลงเอดีซีไปใช้เอดีซีที่มีความเร็วสูงขึ้น เราก็จะสามารถวัดการเปลี่ยนแปลงที่มีความเร็วสูงได้ หรือถ้าเราใช้เอดีซีที่มีความละเอียดสูงเราก็จะสามารถตรวจวัดการเปลี่ยนแปลงที่มีค่าต่างๆได้ และนอกจากนี้ถ้าเราเปลี่ยนจากการใช้แบตเตอรี่มาเป็นแบบ NiCd ก็จะช่วยให้สามารถพกพาได้สะดวกยิ่งขึ้น

ภาคผนวก ก
โปรแกรมการทำงาน

โปรแกรมการทำงานของเครื่องวัดสัญญาณไฟสลับ

```
      ORG     0000H

ALE     EQU     8EH

   ;DEFINE REGISTER FOR PROGRAM
 L_ADD  EQU        2FH          ; LOW   BYTE
ADDRESS
 H_ADD  EQU     30H       ; HIGH BYTE ADDRESS
 ADD16  EQU     31H       ; 16th BIT  ADDRESS
 MSEC   EQU     32H
 SEC    EQU     33H
 HOUR   EQU     34H       ; STORAGE HOURS
 MINT   EQU     35H       ; STORAGE MINUTE
 NUM_1   EQU     36H       ; HIGH BYTE DATA
 NUM_2   EQU     37H       ; LOW  BYTE DATA
 MS_40   EQU    -50000     ; 2'Complement of 20 mS

;********** ASCII CODE ***********
      BELL        EQU     07H
      BS           EQU     08H
      LF          EQU     0AH
      FS          EQU     0CH
      CR          EQU     0DH
      EOS         EQU     10H
      SPACE       EQU     20H

LJMP    50H             ; GOTO MAIN PROGRAM

;******** Interrupt Vector ********
ORG     0003H
JMP         EXT_0   ; EXTERNAL_0   INTERRUPT
VECTOR
ORG     000BH
JMP         TIME_0 ; TIMER_0        INTERRUPT
VECTOR

ORG     0023H
JMP         TXRX     ; SERIAL PORT INTERRUPT
VECTOR

;#### Duty of General register ####
;     R0 = Status of A/D (0:Running 1:stopping)
;     R1 = Calculate number of data ( HIGH_BYTE )
;     R2 = Calculate number of data ( LOW_BYTE  )
;     R3 = General Register
;     R4 = To Compare data & General Register
;     R5 = To Compare data & General Register
;     R6 = To Compare data & General Register
;     R7 = General Register
;    P1.1 = Enable/Disable A/D & Buffer ( 0:A/D[D]
                [E]Buff 1:A/D[E] [D]Buff)
;    P1.2 = Not Use
;    P1.3 = 16th Address of RAM
;    P1.4 = Direction of Buffer ( 0:Write 1:Read )

;***********************************
;********** MAIN PROGRAM ********
;***********************************
      ORG     50H
      MOV     HOUR,#02H
      MOV     MINT,#22H
      ORG     60H
MAIN:  MOV     SP,#50H
      CLR     P1.3              ; A16 CONTROL
      CLR       P1.1                    ; A/D
CONTROL(DISABLE)
      ORL     ALE,#1          ; DISABLE ALE
      MOV     TH0,#HIGH MS_40
      MOV     TL0,#LOW MS_40
      MOV     IE,#00011011B
```

```
    MOV   ??D,#00100001B
    MOV   SCON,#01010010B
    MOV   TH1,#0FDH        ; SET BAUDRATE
9600
    SETB  TR1
    SETB  IT0

  ; Send title to computer
    MOV   DPTR,#TITLE
    CALL  SEND_CR
    CALL  SEND_FS
    CALL  SEND_STR

  ; Setup begining status of register
    MOV   NUM_1,#0
    MOV   NUM_2,#0
    MOV   H_ADD,#0
    MOV   L_ADD,#0
    MOV   ADD16,#0
    MOV   DPTR,#0
    MOV   R0,#1

    NOP

BEGIN: MOV   R1,DPH
    MOV   R2,DPL
    CALL  START
    MOV   H_ADD,DPH
    MOV   L_ADD,DPL
    MOV   ACC.3,P1.3
    MOV   ADD16,A
    SETB  IE.7
    CJNE  R0,#0,$
AD_CLK: MOVX  @DPTR,A
    CALL  INC_AD
    CJNE  R0,#1,AD_CLK
    CLR   IE.7
    CALL  TIMER
    CALL  ONLY_ONE
    CALL  WR_NDATA
    CALL  COMPARE
    JMP   BEGIN

;***************************************
;********** INTERRUPT ROUTINE **********
;***************************************

;======  EXTERNAL  INTERRUPT  ROUTINE
=======
EXT_0:  SETB   TR0
        CLR    IE.0
        RETI

;======= TIMER_0  INTERRUPT ROUTINE =======
TIME_0: CJNE   R0,#0,RUNNING
        CLR    P1.1
        CLR    TF0
        CLR    TR0
        MOV    TH0,#HIGH MS_40
        MOV    TL0,#LOW MS_40
        SETB   TR0
        MOV    R0,#1
        RETI

RUNNING: SETB  P1.1
        CLR    TF0
        CLR    TR0
        MOV    TH0,#HIGH MS_40
        MOV    TL0,#LOW MS_40
        SETB   TR0
        CLR    P1.2
        SETB   P1.2
        MOV    R0,#0
OUT_LOOP:RETI
```

```
;=====  SERIAL  PORT  INTERRUPT  ROUTINE
======
TXRX:   JNB    RI,TX
        CLR    RI
        PUSH   ACC
        PUSH   DPH
        PUSH   DPL
        MOV    A,SBUF
RX_0:   CJNE   A,#'L',RX_1
        JMP    LOAD
RX_1:   CJNE   A,#'T',RX_2
        JMP    SET_TIME
RX_2:   CJNE   A,#'N',RX_7
        POP    DPL
        POP    DPH
        POP    ACC
        MOV    DPTR,#0
        RETI
RX_7:   POP    DPL
        POP    DPH
        POP    ACC
TX:     RETI

;****************************************
;********** GENERAL ROUTINE ************
;****************************************


;============ ADDRESS ROUTINE ============
;---------- INC & DEC ADDRESS ----------
INC_AD: INC    DPTR
        MOV    R3,DPL
        CJNE   R3,#0,OUTL
        MOV    R3,DPH
        CJNE   R3,#0,OUTL
        CPL    P1.3
OUTL:   RET
DEC_AD: DEC    DPL
        MOV    R3,DPL
        CJNE   R3,#0,OUTL
        MOV    R3,DPH
        CJNE   R3,#0,OUTL
        CPL    P1.3
        RET


;========== DATA PROCESS LOOP ============
;------- CONVERT BINARY TO ASCII CODE -------
BIN_ASCI:       ANL    A,#0FH
                MOV    R3,A
                CLR    C
                SUBB   A,#10
                JNC    STEXT
                MOV    A,R3
                ADD    A,#30H
                RET
STEXT:  MOV     A,R3
        ADD     A,#37H
        RET

;----- CONVERT ASCII TO BINARY CODE -----
ASCI_BIN:       CLR    C
                MOV    R3,A
                SUBB   A,#41H
                JNC    RTEXT
                MOV    A,R3
                CLR    C
                SUBB   A,#30H
                RET
RTEXT:  MOV     A,R3
        SUBB    A,#37H
        RET

;---------- DATA INFORMATION ----------
```

```
START:  CLR   P1.4
        MOV   A,#0
        CALL  INC_AD
        MOVX  @DPTR,A
        CALL  INC_AD
        MOV   A,HOUR
        MOVX  @DPTR,A
        CALL  INC_AD
        MOV   A,MINT
        MOVX  @DPTR,A
        CALL  INC_AD
        SETB  P1.4
        RET

;---------- WRITE DATA NUMBER ----------
WR_NDATA:   CLR   P1.4
            CLR   C
            MOV   A,DPL
            MOV   DPL,R2
            SUBB  A,#4
            JNC   DAT0
            DEC   DPH
            CLR   C
DAT0:   SUBB  A,R2
        MOV   R2,A
        JNC   DAT1
        CLR   C
        DEC   DPH
DAT1:   MOV   A,DPH
        MOV   DPH,R1
        SUBB  A,R1
        MOV   R1,A
        JNC   WR_NUM
        CPL   P1.3
WR_NUM:     MOV   A,R2
            MOVX  @DPTR,A
            CALL  INC_AD
            MOV   A,R1
            MOVX  @DPTR,A
            MOV   DPH,H_ADD
            MOV   DPL,L_ADD
            MOV   A,ADD16
            MOV   P1.3,ACC.3
            RET

ONLY_ONE:   MOV   DPH,H_ADD
            MOV   DPL,L_ADD
            MOV   A,ADD16
            MOV   P1.3,ACC.3
            SETB  P1.4
            MOV   R4,#0
            MOV   R5,#0
            CLR   C
            MOV   R7,#0
ON_1:       MOVX  A,@DPTR
            INC   R7
            CALL  INC_AD
            SUBB  A,#7FH
            JNC   ON_1
            CLR   C
            MOV   A,#8
            SUBB  A,R7
            JNC   ON_1

ON_2:       CLR   C
            MOVX  A,@DPTR
            CALL  INC_AD
            MOV   R6,A
            SUBB  A,#7FH
            JC    ON_2
            MOV   R7,#0
ON_3:       CALL  MOVE
            MOVX  A,@DPTR
            INC   R7
            CALL  INC_AD
            MOV   R6,A
```

```
                SUBB    A,#7FH
                JNC     ON_3
                CLR     C
                MOV     A,#22
                SUBB    A,R7
                JNC     ON_3
                MOV     R7,#0
ON_4:           CLR     C
        CALL    MOVE
        INC     R7
        MOVX    A,@DPTR
        CALL    INC_AD
        MOV     R6,A
        SUBB    A,#7FH
        JC      ON_4
        MOV     A,#4
        SUBB    A,R7
        JNC     ON_4
        MOV     DPH,H_ADD
        MOV     DPL,L_ADD
        MOV     A,ADD16
        MOV     P1.3,ACC.3
        RET

MOVE:           MOV     ACC.3,P1.3
        PUSH    ACC
        PUSH    DPH
        PUSH    DPL
        MOV     DPH,H_ADD
        MOV     DPL,L_ADD
        MOV     A,ADD16
        MOV     P1.3,ACC.3
        CLR     P1.4
        MOV     A,R6
        MOVX    @DPTR,A
        SETB    P1.4
        CALL    INC_AD
        MOV     H_ADD,DPH
        MOV     L_ADD,DPL
        MOV     ACC.3,P1.3
        MOV     ADD16,A
        POP     DPL
        POP     DPH
        POP     ACC
    MOV     P1.3,ACC.3
    RET

TIMER:  INC     MSEC
        MOV     A,MSEC
        CJNE    A,#10,OUT
        CLR     MSEC
        INC     SEC
        MOV     A,SEC
        CJNE    A,#60,OUT
        CLR     SEC
        INC     MINT
        MOV     A,MINT
        CJNE    A,#60,OUT
        CLR     MINT
        INC     HOUR
        MOV     A,HOUR
        CJNE    A,#24,OUT
        CLR     HOUR
OUT:    RET

;========== INTERFACE ROUTINE ==========
;---------- SEND ASCII CODE ----------
SEND_ASC:       MOV     B,A
                SWAP    A
                CALL    BIN_ASCI
                CALL    SEND
                MOV     A,B
                CALL    BIN_ASCI
```

```
                    CALL   SEND
                    RET
;-------- DATA TRANSFER ROUTINE --------
SEND:   JNB    TI,$
        CLR    TI
        MOV    SBUF,A
        RET

RECV:   JNB    RI,$
        CLR    RI
        MOV    A,SBUF
        RET
RECVE:  JNB    RI,$
        CLR    RI
        MOV    A,SBUF
        CALL   SEND
        RET

;---------- SEND STRING LOOP ----------
SEND_STR:       PUSH   ACC
SD_STR1:        CLR    A
                MOVC   A,@A+DPTR
                CJNE   A,#EOS,SD_STR2
                POP    ACC
                RET
SD_STR2:        CALL   SEND
                INC    DPTR
                JMP    SD_STR1
COMPARE:        SETB   P1.4
                MOV    NUM_1,R1
                MOV    NUM_2,R2
                MOV    R4,NUM_1
                MOV    R5,NUM_2
                CLR    C
                MOV    A,R2
                ADD    A,#4
                JNC    COMP1
                INC    R1
                INC    R5
                INC    NUM_1
COMP0:                  CALL   DEC_AD
                MOV    R1,NUM_1
                MOV    R2,NUM_2
COMP1:          MOVX   A,@DPTR
                MOV    R7,A
                PUSH   DPH
                PUSH   DPL
                MOV    ACC.3,P1.3
                PUSH   ACC
                CJNE   R1,#0,COMP2
                INC    R1
COMP2:          CALL   DEC_AD
                DJNZ   R2,COMP2
                DJNZ   R1,COMP2
                MOVX   A,@DPTR
                CLR    C
                SUBB   A,R7
                JNC    COMP3
                MOVX   A,@DPTR
                XCH    A,R7
                SUBB   A,R7
COMP3:          SUBB   A,#3
                JNC    STORAGE
                DJNZ   R5,COMP4
                DJNZ   R4,COMP4
                CALL   DEC_AD
                CALL   DEC_AD
                CALL   DEC_AD
                CALL   DEC_AD
                MOV    H_ADD,DPH
                MOV    L_ADD,DPL
                MOV    ACC.3,P1.3
                MOV    ADD16,A
COMP4:          POP    ACC
                MOV    P1.3,ACC.3
                POP    DPL
```

```
                POP     DPH
                JMP     COMP0
COMP5:          POP     ACC
                MOV     P1.3,ACC.3
                POP     DPL
                POP     DPH
                MOV     DPH,H_ADD
                MOV     DPL,L_ADD
                MOV     A,ADD16
                MOV     P1.3,ACC.3
                RET

STORAGE:        POP     ACC
                MOV     P1.3,ACC.3
                POP     DPL
                POP     DPH
                MOV     DPH,H_ADD
                MOV     DPL,L_ADD
                MOV     A,ADD16
                MOV     P1.3,ACC.3
                RET

;--------- SEND GENERAL CODE ---------
SEND_CR:        MOV     A,#CR
                CALL    SEND
                MOV     A,#LF
                CALL    SEND
                RET
SEND_FS:        MOV     A,#FS
                CALL    SEND
                RET

STOP:           CLR     P1.1
                CLR     TR0
                CLR     TF0
                CLR     P1.4
                MOV     DPH,H_ADD
                MOV     DPL,L_ADD
                MOV     A,ADD16
                MOV     P1.3,ACC.3
                CALL    DEC_AD
                CALL    DEC_AD
                CALL    DEC_AD
                MOV     A,#0
                MOVX    @DPTR,A
                CALL    DEC_AD
                MOVX    @DPTR,A
                RET

;-- DOWN LOAD DATA FORM 89C51 TO
COMPUTER
LOAD:           CALL    STOP
                MOV     DPTR,#0
                CLR     P1.3
                CLR     IE.7
                SETB    P1.4

LOAD_0:         MOVX    A,@DPTR
                MOV     R2,A
                CALL    SEND
                CALL    INC_AD
                MOVX    A,@DPTR
                MOV     R1,A
                CALL    SEND
                ORL     A,R2
                JZ      END_LOAD
                CJNE    R2,#0,CHK_LD1
                JMP     CHK_LD2
CHK_LD1:        INC     R1
CHK_LD2:        CALL    INC_AD
                MOVX    A,@DPTR
                CALL    SEND
                CALL    INC_AD
                MOVX    A,@DPTR
```

```
                CALL    SEND
LOAD_1:         CALL    INC_AD
                MOVX    A,@DPTR
                SUBB    A,#3
                CALL    SEND
LOAD_2:         DJNZ    R2,LOAD_1
                DJNZ    R1,LOAD_1
                CALL    INC_AD
                JMP     LOAD_0
END_LOAD:       POP     DPL
                POP     DPH
                POP     ACC
                MOV     DPTR,#0
                CLR     P1.3
                MOV     R0,SP
                MOV     @R0,#60H
                RETI
;========== SETUP CLOCK OF RWS ==========
SET_TIME:       POP     DPL
                POP     DPH
                POP     ACC
                CALL    STOP
                MOV     IE,#00011011B
                CLR     TR0
                CLR     TF0
                MOV     DPTR,#SETTING
                CALL    SEND_STR
TIME_IN1:       CALL    ST_LOOP
                MOV     HOUR,A
                MOV     A,#':'
                CALL    SEND
TIME_IN2:       CALL    ST_LOOP
                MOV     MINT,A
                MOV     R0,SP
                MOV     @R0,#BEGIN
                MOV     TH0,#HIGH MS_40
                MOV     TL0,#LOW MS_40
                MOV     R0,#1
                RETI
ST_LOOP:        CALL    CLR_TIME
                MOV     B,#10
                MUL     AB
                MOV     B,A
                CALL    CLR_TIME
                ADD     A,B
                RET
CLR_TIME:       MOV     A,#SPACE
                CALL    SEND
CLR_T1:         MOV     A,#BS
                CALL    SEND
                CALL    RECVE
                PUSH    ACC
                CLR     C
                SUBB    A,#41H
                JNC     CLR_T1
                POP     ACC
                CALL    ASCI_BIN
                RET
TITLE:          DB      FS,CR,LF," **********"
                DB      "**********"
                DB      CR,LF," Welcome to AC-Power
Line "
                DB      "Recorder Program"
                DB      CR,LF,"                by"
                DB      CR,LF,"    Miss Orasa Chaiyai & Mr.
                        A.Jenjirodpipat"
                DB      ,LF,"**********
**********",CR,LF,EOS
WAIT:   DB      CR,LF," PRESS ANY KEYS TO
                CONTINUED.",CR,LF,EOS
CHOICE: DB      CR,LF,"   upLoad data.   set Timer."
                DB      CR,LF,"         Stop load.",EOS
SETTING: DB     FS,CR,LF," Enter time (EX 09:58) =
                  ",EOS

                END
```

โปรแกรมจอมอนิเตอร์

```
#pragma inline
#include <alloc.h>
#include <bios.h>
#include <conio.h>
#include <dos.h>
#include <fstream.h>
#include <graphics.h>
#include <io.h>
#include <iostream.h>
#include <mem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned char BYTE;
class ComPort {
    private:
            int ComNumber;
    public:
            ComPort(int, int);
            int DataReady(void);
            void Init(int, int, int, int, char);
            char Read(void);
            void Write(char);
};
ComPort::ComPort(int c = 1, int arg = _COM_9600 | _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY)
{
    ComNumber = c - 1;
    bioscom(_COM_INIT, arg, ComNumber);
}
void ComPort::Init(int c, int baud = 9600, int data = 8, int stop = 1, char parity = 'N')
{
    ComNumber = c - 1;
    bioscom(_COM_INIT, 0xe3, ComNumber);
```

```
}

int ComPort::DataReady(void)
{
    return bioscom(_COM_STATUS, 0, ComNumber) & 0x100;
}

char ComPort::Read(void)
{
    return bioscom(_COM_RECEIVE, 0, ComNumber) & 0xff;
}

void ComPort::Write(char c)
{
    bioscom(_COM_SEND, c, ComNumber);
}

class Windows {
private:
    int x1, y1, x2, y2;
public:
    int color, width;
    void Init(int, int, int, int, int, int);
    void Draw(void);
    void Clear(void);
    void View(int, int);
    int Xlen(void);
    int Ylen(void);
};

#define RestoreView() setviewport(0, 0, 639, 479, 1)

void Frame(int x1, int y1, int x2, int y2, int color, int width)
{
    int i, w = width << 1;
    setfillstyle(SOLID_FILL, color);
    bar(x1, y1, x2, y2);
    setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
```

```
    for (i=0; i<width; i++) {
            setcolor(color + 1);
            rectangle(x1+i, y1+i, x2-i, y2-i);
            setcolor(color + 2);
            rectangle(x1+w+i, y1+w+i, x2-w-i, y2-w-i);
    }
}

void Windows::Init(int xx1, int yy1, int xx2, int yy2, int c, int w)
{
    x1 = xx1;
    y1 = yy1;
    x2 = xx2;
    y2 = yy2;
    color = c;
    width = w;
}

void Windows::Draw(void)
{
    RestoreView();
    Frame(x1, y1, x2, y2, color, width);
}

void Windows::Clear(void)
{
    RestoreView();
    setfillstyle(SOLID_FILL, 0);
    bar(x1, y1, x2, y2);
}
void Windows::View(int xoffset = 0, int yoffset = 32767)
{
    int w = width * 3;
    if (yoffset == 32767) yoffset = xoffset;
    setviewport(x1+w+xoffset, y1+w+yoffset, x2-w-xoffset, y2-w-yoffset, 1);
}
int Windows::Xlen(void)
{
```

```
    return x2 - x1 - width * 6 + 1;
}
int Windows::Ylen(void)
{
    return y2 - y1 - width * 6 + 1;
}

char *RwsHex(int rwsnum)
{
    static char *rwsname = "PRJ-0000.RWS";
    int i;
    for (i=7; i>3; i--) {
            rwsname[i] = rwsnum % 10 + '0';
            rwsnum /= 10;
    }
    return rwsname;
}

void InitScreen(void)
{
    int gdriver = VGA, gmode = VGAHI;
    registerbgidriver(EGAVGA_driver);
    registerbgifont(triplex_font);
    initgraph(&gdriver, &gmode, "");
    setrgbpalette(1, 0, 0, 48);
    setrgbpalette(2, 0, 0, 32);
    setrgbpalette(3, 0, 0, 63);
    setrgbpalette(4, 32, 32, 32);
    setrgbpalette(5, 16, 16, 16);
    setrgbpalette(20, 48, 48, 48);
    setrgbpalette(7, 0, 0, 0);
    setrgbpalette(56, 48, 48, 48);
    setrgbpalette(57, 32, 32, 32);
    setrgbpalette(58, 63, 63, 63);
    setrgbpalette(60, 32, 10, 10);
    setrgbpalette(61, 40, 40, 40);
}
```

```
Windows MenuWin, TextWin, LoadWin, GraphWin[8];
BYTE huge *GraphData;

BYTE huge *GraphSeek(int Number)
{
    unsigned i, len;
    unsigned long ptr;

    if (Number < 0) return NULL;
    i = 0;
    ptr = 0;
    len = GraphData[ptr];
    len |= (unsigned)GraphData[ptr + 1] << 8;
    while (i++ < Number && len) {
            ptr += len + 4;
            len = GraphData[ptr];
            len |= (unsigned)GraphData[ptr + 1] << 8;
    }
    if (!len) return NULL;
    return &GraphData[ptr];
}

unsigned x, y;
char *bchar, text[2];

void GoXY(int px, int py)
{
   TextWin.View(6);
   setcolor(15);
   setfillstyle(SOLID_FILL, TextWin.color);
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
   putimage(x << 3, y << 3, bchar, 0);
   x = px;
   y = py;
   px = x << 3;
   py = y << 3;
   getimage(px, py, px + 7, py + 7, bchar);
   outtextxy(px, py, "_");
```

```
}

void Print(char c, char col = 15)
{
   unsigned px, py;

   TextWin.View(6);
   setcolor(col);
   setfillstyle(SOLID_FILL, TextWin.color);
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
   putimage(x << 3, y << 3, bchar, 0);
   switch (text[0] = c) {
          case 8:
             if (x) x--;
             break;
          case 10:
             if (y < 21)
                   y++;
             else {
                   bar(0, 0, 610, 175);
                   x = y = 0;
             }
             break;
          case 11:
             x = y = 0;
             break;
          case 12:
             bar(0, 0, 610, 175);
             x = y = 0;
             break;
          case 13:
             x = 0;
             break;
          default: {
             px = x << 3;
             py = y << 3;
             bar(px, py, px + 7, py + 7);
             outtextxy(px, py, text);
```

```
                if (x < 76) x++;
            }
        }
        px = x << 3;
        py = y << 3;
        getimage(px, py, px + 7, py + 7, bchar);
        outtextxy(px, py, "_");
    }

    void Print(char *s, char col = 15)
    {
        unsigned i, l;

        l = strlen(s);
        for (i=0; i<l; i++) Print(s[i], col);
    }

    void ShowGraph(int WinNum, int GraphNum, unsigned offset = 65535)
    {
        Windows Win;
        BYTE huge *Graph;
        int w, h, x, bx, by, count, hour, minute, len, dec, sign;
        char *s;
        static char time[] = "00:00";
        static char number[6];

        Win = GraphWin[WinNum];
        Graph = GraphSeek(GraphNum);
        if (!Graph) {
            h = Win.color;
            Win.color = 1;
        }
        Win.Draw();
        Win.View();
        if (!Graph) {
            Win.color = h;
            return;
        }
```

```
count = (Graph[0] | (unsigned)Graph[1] << 8);
w = Win.Xlen();
h = Win.Ylen();
setcolor(13);
line(32, h / 2, 32 + w - 1, h / 2);
line(0, h / 4, 32 + w - 1, h / 4);
line(0, h * 3 / 4, 32 + w - 1, h * 3 / 4);
line(32, 0, 32, h - 1);
settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
settextjustify(RIGHT_TEXT, TOP_TEXT);
outtextxy(32, 1, "+5V");
outtextxy(32, h / 4 + 2, "+2V");
outtextxy(32, h * 3 / 4 - 8, "-2V");
outtextxy(32, h - 8, "-5V");
settextjustify(LEFT_TEXT, TOP_TEXT);
hour = Graph[2];
minute = Graph[3];
if (Win.color == 4)
        setcolor(6);
else
        setcolor(12);
w -= 33;
count--;
moveto(32, ((unsigned long)Graph[4] ^ 255) * h / 256);
putpixel(getx(), gety(), getcolor());
for (x=1; x<=count; x++) {
        line(getx(), gety(), bx = 32 + (unsigned long)x * w / count, by = (unsigned long)(Graph[x+4] ^ 255) * h / 256);
        moveto(bx, by);
}
w += 33;
settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
if (Win.color == 4)
        setcolor(11);
else
        setcolor(0);
time[0] = hour / 10 + '0';
time[1] = hour % 10 + '0';
time[3] = minute / 10 + '0';
```

```
        time[4] = minute % 10 + '0';
        outtextxy(w - 42, h - 10, time);
        settextjustify(RIGHT_TEXT, TOP_TEXT);
        outtextxy(w - 4, 4, itoa(GraphNum, number, 10));
        settextjustify(LEFT_TEXT, TOP_TEXT);
        if (offset < 65535) {
                w -= 33;
                setcolor(14);
                moveto(32 + (unsigned long)offset * w / count, 0);
                linerel(0, h -1);
                moveto(32, (unsigned long)(Graph[offset+4] ^ 255) * h / 256);
                linerel(w - 1, 0);
                GoXY(20, 1);
                Print("X = ");
                s = ecvt((double)offset * 20 / count, 5, &dec, &sign);
                for (x=0; x<dec; x++) Print(s[x]);
                if (!dec) Print('0');
                Print('.');
                for (; x < 5 - !dec; x++) Print(s[x]);
                Print(" msec");
                GoXY(42, 1);
                Print("Y = ");
                Print(itoa(Graph[offset+4] - 127, number, 10));
                Print("  ");
        }
}

void MenuWindows(void)
{
    int i;
    static char *MenuText[] = {
                "[L] Load",
                "[A] Actual",
                "[W] Wide",
                "[D] Divide",
                "[O] Open",
                "[T] Set Time",
                "[Q] Quit"
```

```
    };
    MenuWin.Draw();
    MenuWin.View();
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 3);
    setcolor(14);
    for (i=0; i<7; i++) {
        outtextxy(15, 8 + i * 35, MenuText[i]);
    }
}

void DivideWindows(int GraphNum, int CurrentWin)
{
    int i;
    for (i=0; i<6; i++) {
        if (i == CurrentWin) {
            GraphWin[i].color = 8;
            GraphWin[i].width = 2;
        } else {
            GraphWin[i].color = 4;
            GraphWin[i].width = 1;
        }
        ShowGraph(i, GraphNum + i);
        if (i == CurrentWin) {
            GraphWin[i].color = 4;
            GraphWin[i].width = 1;
        }
    }
}

void GraphWindows(int GraphNum, int CurrentWin, unsigned offset = 0)
{
    switch (CurrentWin) {
        case 6:
            GraphWin[7].Clear();
            MenuWindows();
            ShowGraph(CurrentWin, GraphNum);
            break;
        case 7:
```

```
                ShowGraph(CurrentWin, GraphNum, offset);
                break;
            default: {
                GraphWin[7].Clear();
                MenuWindows();
                DivideWindows(GraphNum, CurrentWin);
            }
    }
}

void TextWindows(void)
{
    TextWin.Draw();
    TextWin.View(6);
    settextstyle(0, HORIZ_DIR, 1);
    setcolor(15);
}

void MainScreen(int GraphNum, int CurrentWin, unsigned offset)
{
    cleardevice();
    MenuWindows();
    TextWindows();
    GraphWindows(GraphNum, CurrentWin, offset);
}

void beep(int time = 100)
{
    sound(600);
    delay(time);
    nosound();
}

void main(int argc, char *argv[])
{
    ComPort COM(1);
    fstream rwsfile;
    char key, *vscn;
```

```
unsigned long count;
unsigned px, py, i, j, len, offset, CurrentWin;
int RwsNum, GraphNum, GraphMax;

if (argc > 1) {
        switch (argv[1][0]) {
           case '1': COM.Init(1);
                     break;
           case '2': COM.Init(2);
                     break;
           default: {
                    printf("uses: XTRA 1  --- for COM1\n");
                    printf("uses: XTRA 2  --- for COM2\n");
                    exit(1);
           }
        }
}
InitScreen();
GraphData = new huge BYTE[131072];
vscn = new char[imagesize(64, 0, 447, 271)];
bchar = new char[imagesize(0, 0, 7, 7)];
if (!(GraphData && vscn && bchar)) {
        closegraph();
        cout << "Out of memory\n";
        delete GraphData;
        delete vscn;
        delete bchar;
        exit(1);
}
MenuWin.Init(458,   0, 639, 271, 1, 2);          // Menu Windows
TextWin.Init(  0, 280, 639, 479, 1, 2);          // Text Windows
LoadWin.Init( 50, 100, 400, 160, 8, 2);          // Load Windows
GraphWin[0].Init(  0,   0, 149, 136, 4, 1);      // Divide Windows #0
GraphWin[1].Init(  0, 136, 149, 271, 4, 1);      // Divide Windows #1
GraphWin[2].Init(149,   0, 298, 136, 4, 1);              // Divide Windows #2
GraphWin[3].Init(149, 136, 298, 271, 4, 1);      // Divide Windows #3
GraphWin[4].Init(298,   0, 447, 136, 4, 1);      // Divide Windows #4
GraphWin[5].Init(298, 136, 447, 271, 4, 1);              // Divide Windows #5
```

```
GraphWin[6].Init(  0,   0, 447, 271, 8, 2);                    // Actual Windows
GraphWin[7].Init(  0,   0, 639, 271, 8, 2);          // Wide Windows
RwsNum = GraphNum = GraphMax = offset = x = y = 0;
CurrentWin = 6;
key = 0;
GraphData[0] = GraphData[1] = 0;
text[1] = 0;
MainScreen(GraphNum, CurrentWin, offset);
getimage(0, 0, 7, 7, bchar);
Print(12);
while (key != 'Q') {
        if (COM.DataReady()) Print(COM.Read());
        if (kbhit()) {
           key = getch();
           key -= (key >= 'a' && key <= 'z') * 32;
           switch (key) {
                  case 0:
                     if (CurrentWin < 6) {
                           GraphWin[CurrentWin].color = 4;
                           GraphWin[CurrentWin].width = 1;
                           ShowGraph(CurrentWin, GraphNum + CurrentWin);
                           setfillstyle(SOLID_FILL, 4);
                           switch (getch()) {
                              case 72:
                                    if (CurrentWin) CurrentWin--;
                                    else if (GraphNum) {
                                       GraphWin[6].View(-6);
                                       for (i=0; i<128; i+=64) {
                                              px = (i == 64) * 21;
                                              getimage(0, 0, 383 - px, 271, vscn);
                                              putimage(64 + px, 0, vscn, 0);
                                              bar(0, 0, 63 + px, 271);
                                       }
                                       GraphNum -= 2;
                                       CurrentWin = 1;
                                       ShowGraph(0, GraphNum);
                                    }
                                    break;
```

```
case 75:
        if (CurrentWin > 1) CurrentWin -= 2;
        else if (GraphNum > 1) {
           GraphWin[6].View(-6);
           for (i=0; i<128; i+=64) {
                 px = (i == 64) * 21;
                 getimage(0, 0, 383 - px, 271, vscn);
                 putimage(64 + px, 0, vscn, 0);
                 bar(0, 0, 63 + px, 271);
           }
           GraphNum -= 2;
           ShowGraph(1 - CurrentWin, GraphNum + 1 - CurrentWin);
        }
        break;
case 77:
        if (GraphNum + CurrentWin + 2 >= GraphMax) break;
        if (CurrentWin < 4) CurrentWin += 2;
        else if (GraphNum + (CurrentWin & 4) + 2 < GraphMax) {
           GraphWin[6].View(-6);
           for (i=0; i<128; i+=64) {
                 px = (i == 64) * 21;
                 getimage(64 + px, 0, 447, 271, vscn);
                 putimage(0, 0, vscn, 0);
                 bar(384 - px, 0, 447, 271);
           }
           GraphNum += 2;
           if (GraphNum + CurrentWin >= GraphMax) CurrentWin--;
           ShowGraph(9 - CurrentWin, GraphNum + 9 - CurrentWin);
        }
        break;
case 80:
        if (GraphNum + CurrentWin + 1 >= GraphMax) break;
        if (CurrentWin < 5) CurrentWin++;
        else if (GraphNum + 6 < GraphMax) {
           GraphWin[6].View(-6);
           for (i=0; i<128; i+=64) {
                 px = (i == 64) * 21;
                 getimage(64 + px, 0, 447, 271, vscn);
```

```
                        putimage(0, 0, vscn, 0);
                        bar(384 - px, 0, 447, 271);
                    }
                    GraphNum += 2;
                    CurrentWin = 4;
                    ShowGraph(5, GraphNum + 5);
                }
                break;
            default:;
        }
        GraphWin[CurrentWin].color = 8;
        GraphWin[CurrentWin].width = 2;
        ShowGraph(CurrentWin, GraphNum + CurrentWin);
    } else if (CurrentWin == 6) {
        switch (getch()) {
            case 75:
                if (GraphNum) GraphNum--;
                break;
            case 77:
                if (GraphNum + 1 < GraphMax) GraphNum++;
                break;
            default:;
        }
        ShowGraph(6, GraphNum);
    } else {
        switch (getch()) {
            case 75:
                if (offset) offset-- ;
                else beep();
                break;
            case 77:
                count = GraphSeek(GraphNum) - GraphData;
                px = GraphData[count] | (unsigned)GraphData[count + 1] << 8;
                if (offset < --px) {
                    offset++;
                } else beep();
                break;
            case 73:
```

```
                    if (offset > 64) offset -= 64;
                    else {
                        offset = 0;
                        beep();
                    }
                    break;
                case 81:
                    count = GraphSeek(GraphNum) - GraphData;
                    px = GraphData[count] | (unsigned)GraphData[count + 1] << 8;
                    if (offset < --px - 64) offset += 64;
                    else {
                        offset = px;
                        beep();
                    }
                    break;
                default:;
            }
            ShowGraph(7, GraphNum, offset);
        }
        break;
    case 'A':
        Print(12);
        if (CurrentWin < 6) GraphNum += CurrentWin;
        GraphWindows(GraphNum, CurrentWin = 6);
        break;
    case 'D':
        Print(12);
        GraphWindows(GraphNum &= 0xfffe, CurrentWin = GraphNum & 1);
        break;
    case 'W':
        if (CurrentWin < 6) GraphNum += CurrentWin;
        Print(12);
        GoXY(5, 4);
        Print("Fast Fourier Transform", 14);
        GoXY(10, 6);
        Print(" 2   2");
        GoXY(10, 7);
        Print("X + Y");
```

```
        GraphWindows(GraphNum, CurrentWin = 7, offset = 0);
        break;
    case 'O':
        LoadWin.Draw();
        LoadWin.View(60, 20);
        setcolor(1);
        setfillstyle(SOLID_FILL, LoadWin.color);
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
        outtext("Number of rws file: ");
        px = getx();
        py = gety();
        len = px;
        i = 0;
        outtext("_");
        while ((key = getch()) != 13 || px == len) {
            if ((key >= '0' && key <= '9') && px < len + 32) {
                i = i * 10 + key - '0';
                text[0] = key;
                bar(px, py, px + 7, py + 7);
                outtextxy(px, py, text);
                px += 8;
                outtextxy(px, py, "_");
            } else if (key == 8 && px > len) {
                i /= 10;
                px -= 8;
                bar(px, py, px + 15, py + 7);
                outtextxy(px, py, "_");
            }
        }
        rwsfile.open(RwsHex(i), ios::in | ios::nocreate | ios::binary);
        if (rwsfile) {
            rwsfile.seekg(0, ios::end);
            count = rwsfile.tellg();
            rwsfile.seekg(0, ios::beg);
            rwsfile.read((BYTE *)GraphData, count);
            rwsfile.close();
            GraphMax = count = 0;
            len = GraphData[count];
```

```
            len += (unsigned)GraphData[count + 1] << 8;
            while (len) {
               GraphMax++;
               count += len + 4;
               len = GraphData[count];
               len += (unsigned)GraphData[count + 1] << 8;
            }
         } else {
            LoadWin.Draw();
            LoadWin.View(50, 12);
            setcolor(1);
            moveto(15, 0);
            outtext("Not found ");
            outtext(RwsHex(i));
            outtextxy(0, 16, "Press any key to continue...");
            beep(400);
            getch();
         }
         if (CurrentWin < 6) CurrentWin = 0;
         GraphWindows(GraphNum = 0, CurrentWin, offset = 0);
         break;
      case 'L':
         LoadWin.Draw();
         LoadWin.View();
         settextstyle(TRIPLEX_FONT, HORIZ_DIR, 1);
         setcolor(0);
         outtextxy(70, 10, "Loading Please Wait...");
         GraphMax = count = 0;
         COM.Write(key);
         len = GraphData[count++] = COM.Read();
         len |= (unsigned)(GraphData[count++] = COM.Read()) << 8;
         while (len) {
            if (count + len < 130000) {
               GraphMax++;
               GraphData[count++] = COM.Read();
               GraphData[count++] = COM.Read();
               while (len, len--) GraphData[count++] = COM.Read();
               len = GraphData[count++] = COM.Read();
```

```
                        len |= (unsigned)(GraphData[count++] = COM.Read()) << 8;
                    } else {
                        GraphData[count-1] = 0;
                        GraphData[count-2] = 0;
                        len = 0;
                    }
                }
                rwsfile.open(RwsHex(RwsNum++), ios::out | ios::noreplace | ios::binary);
                while (!rwsfile) {
                    rwsfile.open(RwsHex(RwsNum++), ios::out | ios::noreplace | ios::binary);
                }
                rwsfile.write(GraphData, count);
                rwsfile.close();
                if (CurrentWin < 6) CurrentWin = 0;
                GraphWindows(GraphNum = 0, CurrentWin, offset = 0);
                break;
            case 'C':
            case 'T':
            case 'N':
            case 'R':
                COM.Write(key);
                break;
            case 'Q':
                break;
            default:;
        }
    }
    closegraph();
    delete GraphData;
    delete vscn;
    delete bchar;
    while (kbhit()) getch();
}
```

ภาคผนวก ข
รายละเอียดอุปกรณ์

ARCHITECTURAL OVERVIEW OF THE MHS C51 FAMILY

INTRODUCTION

The MHS C51 microcontroller family is based on the 80C51 core which features are :

- 8-bit CPU optimized for control applications
- Extensive boolean processing (single-bit logic) capabilities
- 64 K Program Memory address space
- 64 K Data Memory address space
- 4 K bytes of on chip Program Memory
- 128 bytes of on chip data RAM
- 32 bidirectionnal and individually addressable I/O lines
- two 16-bit timers/counters
- Full duplex UART
- 6 sources / 5-vector interrupt structure with 2 priority levels
- on chip clock oscillators

The basic architectural structure of the MHS C51 microcontroller family is shown in figure 1.

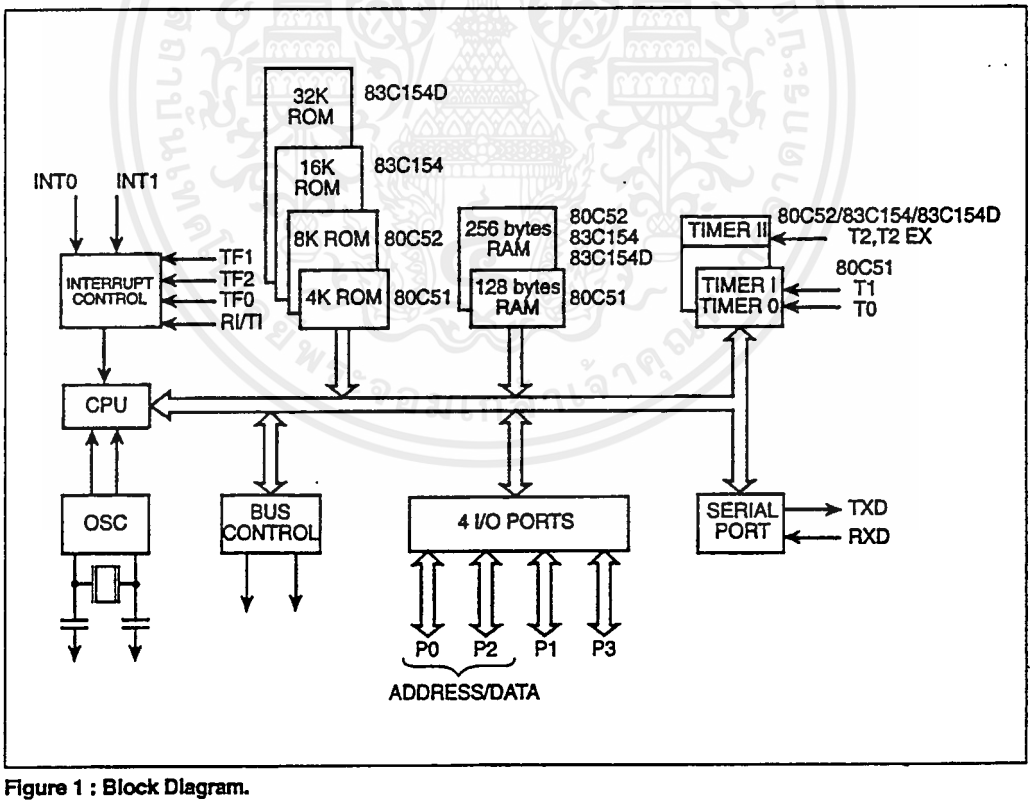


Figure 1 : Block Diagram.

Each device of the MHS C51 family is listed in Table 1.

DEVICE NAME	ROMLESS VERSION	ROM BYTES	RAM BYTES	16-BIT TIMERS	SPEED UP TO	PROCESS
80C51	80C31	4 K	128	2	20 MHz	CMOS
80C52	80C32	8 K	256	3	20 MHz	CMOS
83C154	80C154	16 K	256	3	20 MHz	CMOS
83C154D	–	32 K	256	3	16 MHz	CMOS
80C51µ	80C31µ	4 K	128	2	42 MHz	SCMOS
80C52µ	80C32µ	8 K	256	3	42 MHz	SCMOS
83C154µ	80C154µ	16 K	256	3	42 MHz	SCMOS
83C154Dµ	–	32 K	256	3	36 MHz	SCMOS
80C51PX	–	–	128/256	2/3	12 MHz	CMOS

Table 1 : MHS C51 Family of Microcontrollers.

80C51/80C31

The 80C51 is the CMOS version of the 8051. Functionally, it is fully compatible with the 8051, but being CMOS it draws less current than its HMOS counterpart. To further exploit the power savings available in CMOS circuitry, two reduced power modes are added ;

- Software-invoked Idle Mode, during which the CPU is turned off while the RAM and other onchip peripherals continue operating. In this mode, current draw is reduced to about 15 % of the current drawn when the device is fully active.
- Software-invoked Power Down Mode, during which all on-chip activities are suspensed. The on-chip RAM continues to hold its data. In this mode the device typically draws less than 10 µA.

Although the 80C51 is functionally compatible with its HMOS counterpart, specific differences between the two types of devices must be considered in the design of an application circuit if one wishes to ensure complete interchangeability between the HMOS and CMOS devices.
The ROMless version of the 80C51 is the 80C31.
80C51µ and 80C31µ are the submicron, CMOS (SCMOS) versions of the same devices.

80C52/80C32

The 80C52 is an enhanced 80C51. It is produced with CMOS technology, and is compatible with the 80C51. Its enhancements over the 80C51 are as follows :

- 256 bytes of on-chip RAM
- Three timer/counters
- 6-source interrupt structure
- 8 K bytes of on-chip Program ROM

The ROMless version of the 80C52 is the 80C32.
80C52µ and 80C32µ are the submicron, CMOS (SCMOS) versions of the same devices.

83C154/80C154

The 83C154 is an enhanced 80C52. It is produced with CMOS technology, and is compatible with the 80C51 and 80C52. Its enhancements over the 80C51 are as follows :

- 256 bytes of on-chip data RAM
- Three timer/counters (included watchdog and 32 bits timer/counters)
- 6 source interrupt structure
- Serial reception error detection
- New modes of power reduction consumption
- Programmable impedance port
- 16 K bytes of on-chip ROM
- Asynchronous Counter/Serial port mode during power-down

The ROMless version of the 83C154 is the 80C154.
83C154µ and 80C154µ are the submicron, CMOS (SCMOS) versions of the same devices.

83C154D

The 83C154D is an enhanced 80C154. It is produced with CMOS technology, and is compatible with the 83C154. Its enhancements over the 80C51 are as follows :

- 256 bytes of on-chip data RAM

- Three timer/counters (included watchdog and 32 bits timer/counters)
- 6 source interrupt structure
- Serial reception error detection
- New modes of power reduction consumption
- Programmable impedance port
- 32 K bytes of on-chip ROM
- Asynchronous Counter/Serial port mode during power-down

83C154Dμ is the submicron CMOS (SCMOS) version of the some device.

80C51PX

The MHS 80C51PX are PIGGY BACK devices for all the MHS microcontrollers. 4 Different versions are available :

- 80C51P4 for the 80C51 (4 K ROM)
- 80C51P8 for the 80C52 (8 K ROM)
- 80C51P16 for the 80C154 (16 K ROM)
- 80C51P32 for the 80C154D (32 K ROM)

MEMORY ORGANIZATION IN MHS C51 DEVICES

LOGICAL SEPARATION OF PROGRAM AND DATA MEMORY

All MHS C51 devices have separated address spaces for program and Data Memory, as shown in figure 2. The logical separation of Program and Data Memory allows the Data Memory to be accessed by 8-bit addresses, which can be more quickly stored and manipulated by an 8-bit CPU. Nevertheless, 16-bit Data Memory addresses can also be generated through the DPTR register.

Program Memory can only be read, not written to. There can be up to 64 K bytes of program Memory. In the ROM versions of these devices the lowest 4 K, 8 K, 16 K or 32 K bytes of Program Memory are provided on-chip. Refer. to Table 1 for the amount of on-chip ROM, on each device. In the ROMless versions all Program Memory is external. The read strobe for external Program Memory is the signal PSEN (Program Store Enable).

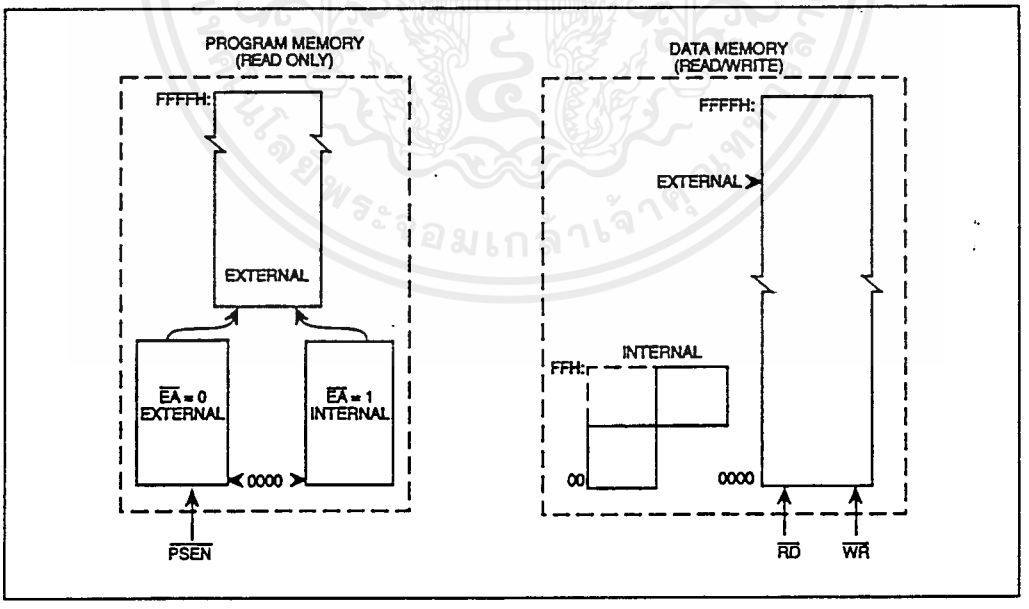


Figure 2 : MHS C51 Memory Structure.

Data Memory occupies a separate address space from Program Memory. Up to 64 K bytes of external RAM can be addressed in the external Data Memory space. The CPU generates read and write signals, RD and WR, as needed during external Data Memory accesses. External Program Memory and external Data Memory may be combined if desired by applying the RD and PSEN signals to the inputs of an AND gate and using the output of the gate as the read strobe to the external Program/Data memory.

PROGRAM MEMORY

Figure 3 shows a map of the lower part of the Program Memory. After reset, the CPU begins execution from location 0000H.

As shown in Figure 3, each interrupt is assigned a fixed location in Program Memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose Program Memory.

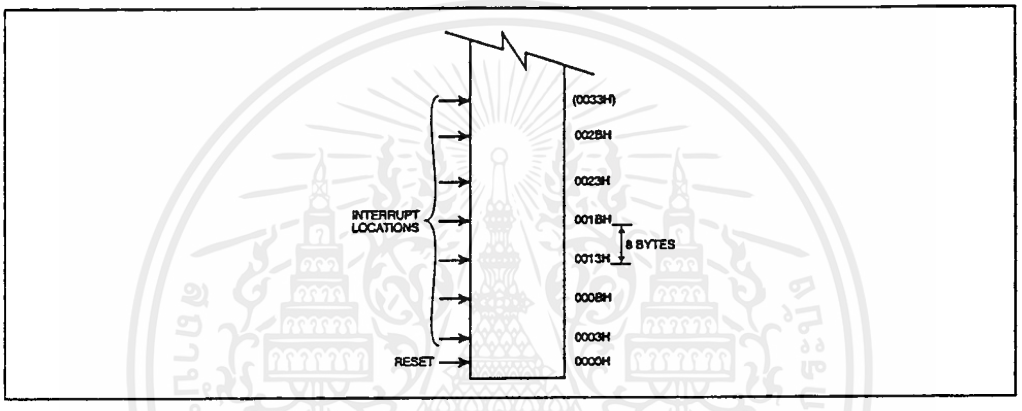


Figure 3 : MHS C51 Program Memory.

The interrupt service locations are spaced at 8-byte intervals : 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

The lowest 4 K (or 8 K in the 80C52 or 16 K in the 83C154 or 32 K in the 83C154D) bytes of Program Memory can be either in the on-chip ROM or in an external ROM. This selection is made by strapping the EA (External Access) pin to either V_{CC} or V_{SS}. In the 80C51 and its derivatives, if the $\overline{EA}$ pin is strapped to V_{CC}, then program fetches to addresses 0000H through 0FFFH are directed to the internal ROM. Program fetches to addresses 1000H through FFFFH are directed to external ROM.

In the 80C52, $\overline{EA} = V_{CC}$ selects addresses 0000H through 1FFFH to be internal, and addresses 2000H through FFFFH to be external.

In the 83C154, $\overline{EA} = V_{CC}$ selects addresses 0000H through 3FFFH to be internal, and addresses 4000H to FFFFH to be external.

In the 83C154D, $\overline{EA} = V_{CC}$ selects addresses 0000H through 7FFFH to be internal and addresses 8000H to FFFFH to be external.

If the $\overline{EA}$ pin is strapped to V_{SS}, then all program fetches are directed to external ROM. The ROMless parts must have this pin externally strapped to V_{SS} to enable them to execute from external Program Memory.

The read strobe to external ROM, $\overline{PSEN}$, is used for all external program fetches. $\overline{PSEN}$ is not activated for internal program fetches.

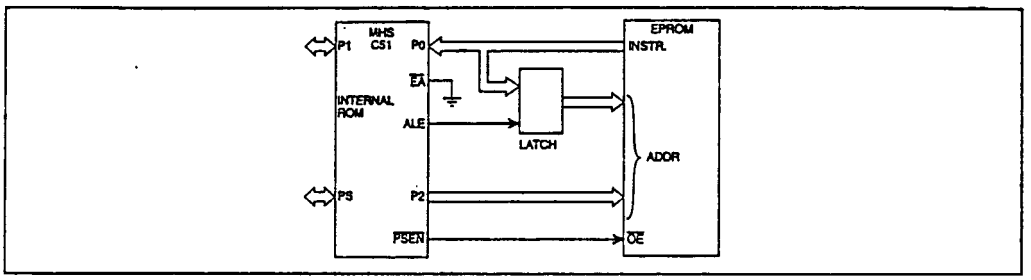


Figure 4 : Executing from External Program Memory.

The harware configuration for external program execution is shown in figure 4. Note that 16 I/O lines (Ports 0 and 2) are dedicated to bus functions during external Program Memory fetches. Port 0 (P0 in Figure 4) serves as a multiplixed address/data bus. It emits the low byte of the Program Counter (PCL) as an address, and then goes into a float state awaiting the arrival of the code byte from the Program Memory. During the time that the low byte of the Program Counter is valid on P0, the signal ALE (Address Latch Enable) clocks this byte into an address latch. Meanwhile, Port 2 (P2 in Figure 4) emits the high byte of Program Counter (PCH). Then PSEN strobes the EPROM and the code byte is read into the microcontroller.

Program Memory addresses are always 16 bits wide, even though the actual amount of Program Memory used may be less than 64 K bytes. External program execution sacrifices two of the 8-bit ports, P0 and P2, to the function of addressing the Program Memory.

DATA MEMORY

The right half of Figure 2 shows the internal and external Data Memory spaces available to the MHS C51 user. Figure 5 shows a hardware configuration for accessing up to 2 K bytes of external RAM. The CPU in this case is executing from internal ROM. Port 0 serves as multiplexed address/data bus to the RAM, and 3 lines of Port 2 are being used to page the RAM. The CPU generates RD and WR signals as needed during external RAM accesses.

There can be up to 64 K bytes of external Data Memory. External Data Memory addresses can be either 1 or 2 bytes wide. One-byte address is often used in conjunction with one or more other I/O lines to page the RAM, as shown in Figure 5. Two-byte addresses can also be used, in which case the address byte is emitted at Port 2.

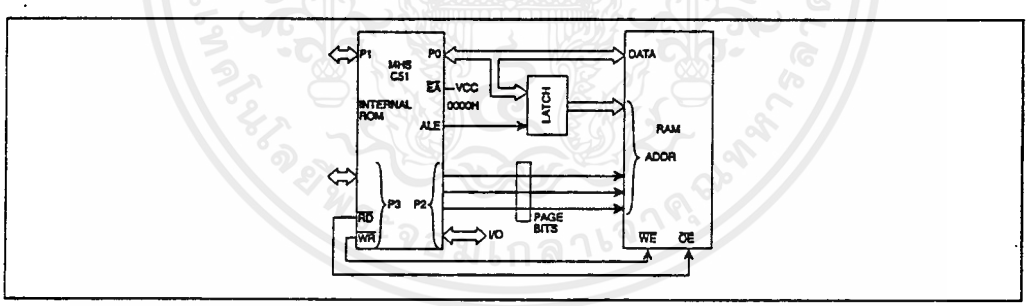


Figure 5 : Accessing External Data Memory. If the Program Memory is external, the other bits of P2 are available as I/O.

Internal Data Memory is mapped in figure 6. The memory space is shown divided into three blocks, which are generally referred to as the lower 128, the Upper 128, and SFR space.

Internal Data Memory addresses are always one byte wide, which implies an address space of only 256 bytes. However, the addressing modes for internal RAM can in fact accomodate 384 bytes, using a simple trick. Direct addresses higher than 7FH access one memory space, and indirect addresses higher than 7FH access a different memory space. Thus figure 6 shows the Upper 128 and SFR space occupying the same block of addresses, 80H through FFH, although they are physically separate entities.

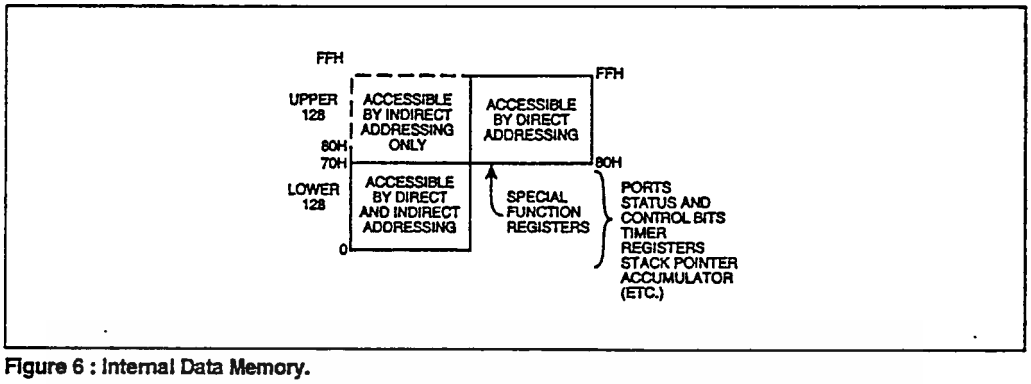


Figure 6 : Internal Data Memory.

The Lower 128 bytes of RAM are present in all MHS C51 devices as mapped in Figure 7. The lowest 32 bytes are grouped into 4 banks of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) select which register bank is in use. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing.

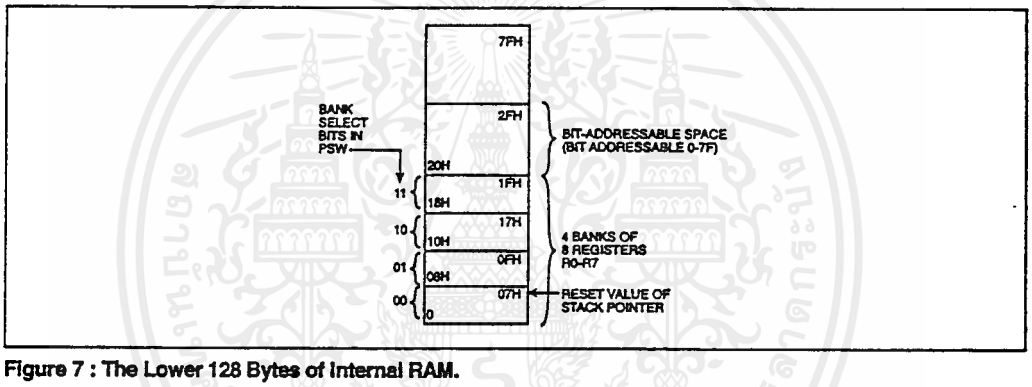


Figure 7 : The Lower 128 Bytes of Internal RAM.

The next 16 bytes above the register banks form a block of bit-addressable memory space. The MHS-C51 instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH.

All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing. The Upper 128 (Figure 8) can only be accessed by indirect addressing. The Upper 128 bytes of RAM are not implemented in the 80C51 but are in the 80C52, 83C154 and 83C154D.

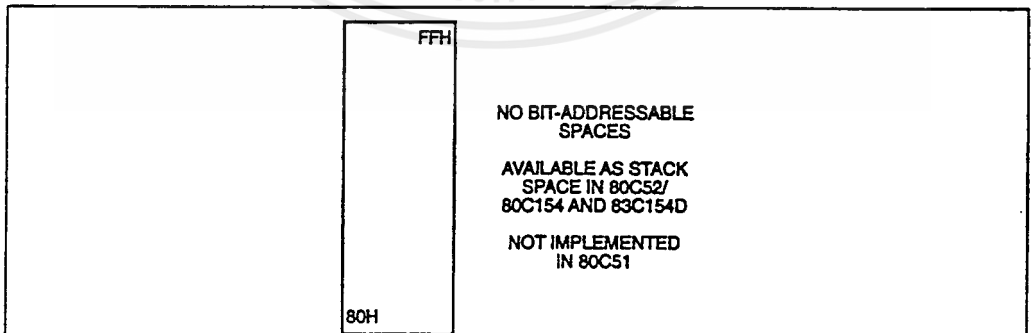


Figure 8 : The Upper 128 Bytes of Internal RAM.

Figure 9 gives a brief look at the Special Function Register (SFR) space. SFRs include the Port latches, timers, peripheral controls, etc. These registers can only be accessed by direct addressing. In general, all MHS C51 microcontrollers have the same SFRs as the 80C51, and at the same addresses in SFR space. However, enhancements to the 80C51 have additional SFRs that are not present in the 80C51, nor perhaps in other proliferation of the family.

Sixteen addresses in SFR space are both byte-and bit-addressable. The bit-addressable SFRs are those whose address ends in 0, 8 or 9. The bit addresses in this area are 80H through FFH.

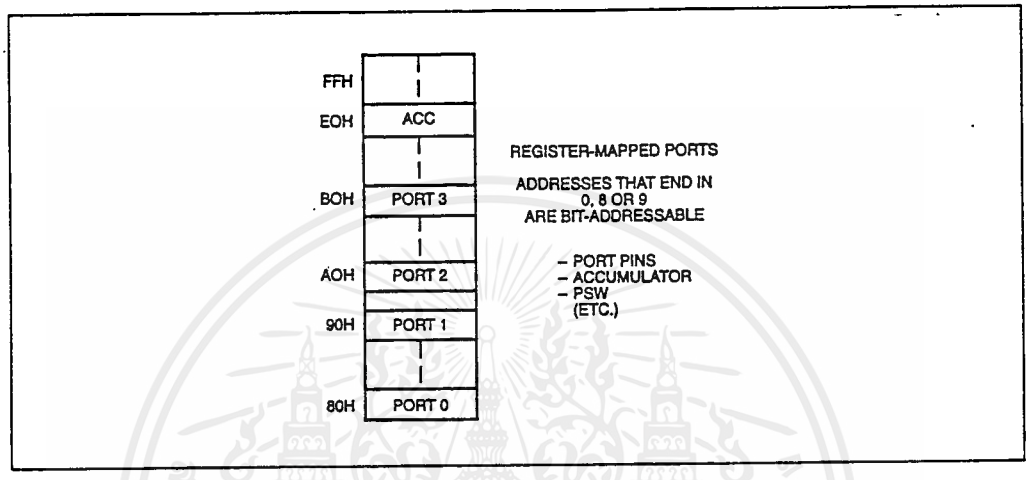


Figure 9 : SFR Space.

THE MHS C51 INSTRUCTION SET

All members of the MHS C51 family execute the same instruction set. (except code A5H, skip opcode in MHS C51/C52). The MHS C51 instruction set is optimized for 8-bit control applications. It provides a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. The instruction set provides extensive support for one-bit variables as a separate data type, allowing direct bit manipulation in control and logic systems that require Boolean processing.

An overview of the MHS C51 instruction set is presented below, with a brief description of how certain instructions might be used.

PROGRAM STATUS WORD

The Program Status Word (PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown in Figure 10, resides in SFR space. It contains the Carry bit, the Auxiliary Carry (for BCD operations), the two register bank select bits, the Overflow flag, a parity bit, and two user-definable status flags.

The Carry bit, other than serving the functions of a Carry bit in arithmetic operations, also serves as the "Accumulator" for a number of Boolean operations.

The bits RS0 and RS1 are used to select one of the four register banks shown in Figure 7. A number of instructions refer to these RAM locations as R0 through R7. The selection of which of the four banks is being referred to is made on the basis of the bits RS0 and RS1 at execution time.

The parity bit reflects the number of 1 s in the Accumulator : P = 1 if the Accumulator contains an odd number of 1 s, and P = 0 if the Accumulator contains an even number of 1 s. Thus the number of 1 s in the Accumulator plus P is always even.

Two bits in the PSW are uncommitted and may be used as general purpose status flags.

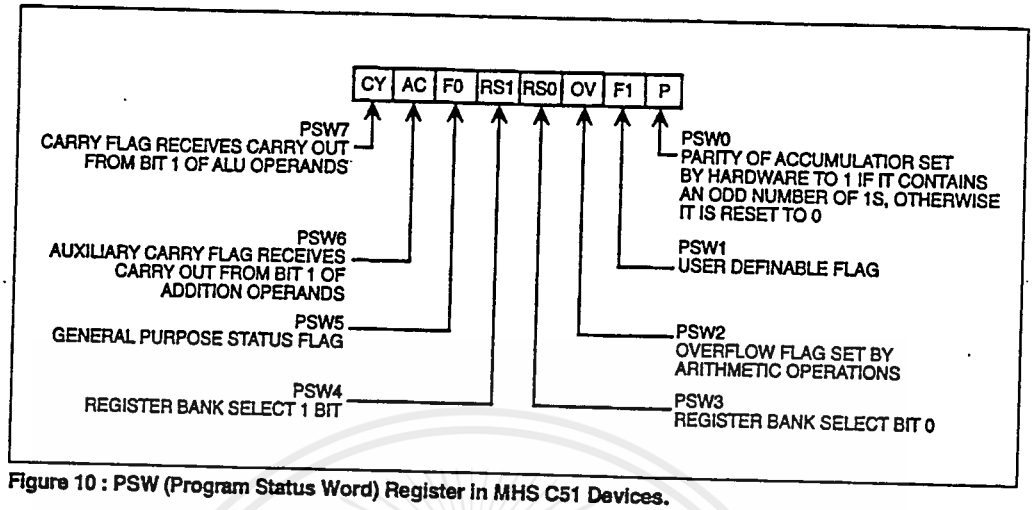


Figure 10 : PSW (Program Status Word) Register in MHS C51 Devices.

ADDRESSING MODES

The addressing modes in the MHS C51 instruction set are as follows :

Direct addressing

In direct addressing the operand is specified by an 8-bit address field in the instruction. Only 128 Lowest bytes of internal Data RAM and SFRs can be directly addressed.

Indirect addressing

In indirect addressing the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed.

The address register for 8-bit addresses can be R0 or R1 of the selected register bank, or the Stack Pointer. The address register for 16-bit addresses can only be the 16-bit "data pointer" register, DPTR.

Register Instructions

The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code efficient, since this mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank is accessed. One of four banks is selected at execution time by the two bank select bits in the PSW.

Register-specific Instructions

Some instructions are specific to a certain register. For example, some instructions always operate on the Accumulator, or Data Pointer, etc., so no address byte is needed to point to it. The opcode itself does that. Instructions that refer to the Accumulator as A assemble as accumulator-specific opcodes.

Immediate constants

The value of a constant can follow the opcode in Program Memory. For example,

MOV A, # 100

loads the Accumulator with the decimal number 100. The same number could be specified in hex digits as 64H.

Indexed addressing

Only Program Memory can be accessed with indexed addressing, and it can only be read. This addressing mode is intended for reading look-up tables in Program Memory. A 16-bit base register (either DPTR or the Program Counter) points to the base of the table, and the Accumulator is set up with the table entry number. The address of the table entry in Program Memory is formed by adding the Accumulator data to the base pointer.

Another type of indexed addressing is used in the "case jump" instruction. In this case the destination address of a jump instruction is computed as the sum of the base pointer and the Accumulator data.

ARITHMETIC INSTRUCTIONS

The menu of arithmetic instructions is listed in Table 2. The table indicates the addressing modes that can be used with each instruction to access the <byte> operand. For example, the ADD A, <byte> instruction can be written as :

ADD A, 7FH (direct addressing)
ADD A, @ R0 (indirect addressing)
ADD A, R7 (register addressing)
ADD A, # 127 (immediate constant)

MNEMONIC	OPERATION	ADDRESSING MODES				EXECUTION TIME (μs)
		Dir	Ind	Reg	Imm	
ADD A, <byte>	A = A + <byte>	X	X	X	X	1
ADDC A, <byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A, <byte>	A = A – <byte> – C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A – 1	Accumulator only				1
DEC <byte>	<byte> = <byte> – 1	X	X	X		1
MUL AB	B:A = B × A	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

Table 2 : A list of the MHS C51 Arithmetic Instructions.

The execution times listed in Table 2 assume a 12 MHz clock frequency. All of the arithmetic instructions execute in 1 μs except the INC DPTR instruction, which takes 2 μs, and the Multiply and Divide instructions, which take 4 μs.

Note that any byte in the internal Data Memory space can be incremented or decremented without going through the Accumulator.

One of the INC instructions operates on the 16-bit Data Pointer. The Data Pointer is used to generate 16-bit addresses for external memory, so being able to increment it in one 16-bit operation is a useful feature.

The MUL AB instruction multiplies the Accumulator by the data in the B register and puts the 16-bit product into the concatenated B and Accumulator registers.

The DIV AB instruction divides the Accumulator by the data in the B register and leaves the 8-bit quotient in the Accumulator, and the 8-bit remainder in the B register.

Oddly enough, DIV AB finds less use in arithmetic "divide" routines than in radix conversions and programmable shift operations. An example of the use of DIV AB in a radix conversion will be given later. In shift operations, dividing a number by 2^n shifts its n bits to the right. Using DIV AB to perform the division completes the shift in 4 μs leaves the B register holding the bits that were shifted out.

The DA A instruction is for BCD arithmetic operations. In BCD arithmetic ADD and ADDC instructions should always be followed by a DA A operation, to ensure that the result is also in BDC. Note that DAA will not convert a binary number to BCD. The DA A operation produces a meaningful result only as the second step in the addition of two BCD bytes.

LOGICAL INSTRUCTIONS

Table 3 shows the list of MHS C51 logical instructions. The instructions that perform Boolean operations (AND, OR, Exclusive OR, NOT) on bytes perform the operation on a bit-by-bit basis. That is, if the Accumulator contains 00110101B and <byte> contains 01010011B, then

ANL A, <byte>

will leave the Accumulator holding 00010001B.
The addressing modes that can be used to access the <byte> operand are listed in Table 3. Thus, the ANL A, <byte> instruction may take any of the forms.

ANL A, 7FH (direct addressing)
ANL A, @ R1 (indirect addressing)
ANL A, R6 (register addressing)
ANL A, # 53H (immediate constant)

All of the logical instructions that are Accumulator specific in 1 µs (using a 12 MHz clock). The others take 2 µs.

MNEMONIC	OPERATION	ADDRESSING MODES				EXECUTION TIME (µs)
		Dir	Ind	Reg	Imm	
ANL A, <byte>	A = A AND <byte>	X	X	X	X	1
ANL <byte>, A	<byte> = <byte> AND A	X				1
ANL <byte>, # data	<byte> = <byte> AND # data	X				2
ORL A, <byte>	A = A OR <byte>	X	X	X	X	1
ORL <byte>, A	<byte> = <byte> OR A	X				1
ORL <byte>, # data	<byte> = <byte> OR # data	X				2
XRL A, <byte>	A = A XOR <byte>	X	X	X	X	1
XRL <byte>, A	<byte> = <byte> XOR A	X				1
XRL <byte>, # data	<byte> = <byte> XOR # data	X				2
CLR A	A = 00H	Accumulator only				1
CLP A	A = NOT A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

Table 3 : A list of the MHS C51 Logical Instructions.

Note that Boolean operations can be performed on any byte in the internal Data Memory space without going through the Accumulator. The XRL <byte>, # data instruction, for example, offers a quick and easy way to invert port bits, as in

XRL P1, #0FFH

If the operation is in response to an interrupt, not using the Accumulator saves the time and effort to stack it in the service routine.

The Rotate instructions (RLA, RLCA, etc.) shift the Accumulator 1 bit to the left or right. For a left rotation, the MSB rolls into the LSB position. For a right rotation, the LSB rolls into the MSB position.

The SWAP A instruction interchanges the high and low nibbles within the Accumulator. this is a useful operation in BCD manipulations. For example, if the Accumulator contains a binary number which is known to be less than 100, it can be quickly converted to BCD by the following code :

```
MOV   B, #10
DIV   AB
SWAP  A
ADD   A,B
```

Dividing the number by 10 leaves the tens digit in the low nibble of the Accumulator, and the ones digit in the B register. The SWAP and ADD instructions move the tens digit to the high nibble of the Accumulator, and the ones digit to the low nibble.

DATA TRANSFERS

Internal RAM

Table 4 shows the menu of instructions that are available for moving data around within the internal memory spaces, and the addressing modes that can be used with each one. With a 12 MHz clock, all of these instructions execute in either 1 or 2 µs.

The MOV <dest>, <src> instruction allows data to be transfered between any two internal RAM or SFR locations without going through the Accumulator. Remember the Upper 128 bytes of data RAM can be accessed only by indirect, and SFR space only by direct addressing.

Note that in all MHS C51 devices, the stack resides in on-chip RAM, and grows upwards. The PUSH instruction first increments the Stack Pointer (SP), then copies the byte into the stack. PUSH and POP use only direct addressing to identify the byte being saved or restored, but the stack itself is accessed by indirect addressing using the SP register. This means the stack can go into the Upper 128, if they are implemented, but not into SFR space.

MNEMONIC	OPERATION	ADDRESSING MODES				EXECUTION TIME (µs)
		Dir	Ind	Reg	Imm	
MOV A, <src>	A = <src>	X	X	X	X	1
MOV <dest>, A	<dest> = A	X	X	X		1
MOV <dest>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR, # data 16	DPTR = 16-bit immediate constant				X	2
PUSH <src>	INC SP : MOV"@SP", <scr>	X				2
POP <dest>	MOV <dest>, "@SP" : DEC SP	X				2
XCH A, <byte>	ACC and <byte> Exchange Data	X	X	X		1
XCHD A, @Ri	ACC and @ Ri exchange low nibbles		X			1

Table 4 : A list of the MHS C51 Data Transfer Instructions that Access Internal Data Memory Space.

The Upper 128 are not implemented in the 80C51, nor in their ROMless. With these devices, if the SP points to the Upper 128 PUSHed bytes are lost, and POPped bytes are indeterminate.

The Data Transfer instructions include a 16-bit MOV that can be used to initialize the Data Pointer (DPTR) for look-up tables in Program Memory, or for 16-bit external Data Memory accesses.

The XCH A, <byte> instruction causes the Accumulator and addressed byte to exchange data.
The XCHD A, @ Ri instruction is similar, but only the low nibbles are involved in the exchange.

The see how XCH and XCHD can be used to facilitate data manipulations, consider first the problem of shifting an 8-digit BCD number two digits to the right. Figure 11 shows how this can be done using direct MOVs, and for comparison how it can be done using XCH instructions. To aid in understanding how the code works, the contents of the registers that are holding the BCD number and the content of the Accumulator are shown alongside each instruction to indicate their status after the instruction has been executed.

After the routine has been executed, the Accumulator contains the two digits that were shifted out on the right. Doing the routine with direct MOVs uses 14 code bytes and 9 µs of execution time (assuming a 12 MHz clock). The same operation with XCHs uses less code and executes almost twice as fast.

	2A	2B	2C	2D	2E	ACC
MOV A,2EH	00	12	34	56	78	78
MOV 2EH, 2DH	00	12	34	56	56	78
MOV 2DH, 2CH	00	12	34	34	56	78
MOV 2CH, 2BH	00	12	12	34	56	78
MOV 2BH, # 0	00	00	12	34	56	78

(a) Using direct MOVs : 14 bytes, 9 µs

	2A	2B	2C	2E	2E	ACC
CLR A	00	12	34	56	78	00
XCH A,2BH	00	00	34	56	78	12
XCH A,2CH	00	00	12	56	78	34
XCH A,2DH	00	00	12	34	78	56
XCH A,2EH	00	00	12	34	56	78

(b) Using XCHs : 9 bytes, 5 µs

Figure 11 : Shifting a BCD Number Two Digits to the Right.

	2A	2B	2C	2D	2E	ACC
MOV R1,# 2EH	00	12	34	56	78	XX
MOV R0,# 2DH	00	12	34	56	78	XX
loop for R1 = 2EH :						
LOOP : MOV A, @R1	00	12	34	56	78	78
XCHD A, @R0	00	12	34	58	78	76
SWAP A	00	12	34	58	78	67
MOV @R1, A	00	12	34	58	67	67
DEC R1	00	12	34	58	67	67
DEC R0	00	12	34	58	67	67
CJNE R1, #2AH, LOOP						
loop for R1 = 2DH :	00	12	38	45	67	45
loop for R1 = 2CH :	00	18	23	45	67	23
loop for R1 = 2BH :	08	01	23	45	67	01
CLR A	08	01	23	45	67	00
XCH A,2AH	00	01	23	45	67	08

Figure 12 : Shifting a BCD Number One Digit to the Right.

To right-shift by an odd number of digits, a one-digit shift must be executed. Figure 12 shows a sample of code that will right-shift a BCD number one digit, using the XCHD instruction. Again, the contents of the registers holding the number and of the Accumulator are shown alongside each instruction.

First, pointers R1 and R0 are set up to point to the two bytes containing the last four BCD digits. Then a loop is executed which leaves the last byte, location 2EH, holding the last two digits of the shifted number. The pointers are decremented, and the loop is repeated for location 2DH. The CJNE instruction (Compare and Jump if Not Equal) is a loop control that will be described later.

The loop is executed from LOOP to CJNE for R1 = 2EH, 2DH, 2CH and 2BH. At that point the digit that was originally shifted out on the right has propagated to location 2AH. Since that location should be left with 0s, the lost digit is moved to the Accumulator.

External RAM

Table 5 shows a list of the Data Transfer instructions that access external Data Memory. Only indirect addressing can be used. The choice is whether to use a one-byte address, @Ri, where Ri can be either R0 or R1 of the selected register bank, or a two-byte address, @DPTR. The disadvantage to using 16-bit addresses if only a few K bytes of external RAM are involved is that 16-bit addresses use all 8 bits of Port 2 as address bus. On the other hand, 8-bit addresses allow one to address a few K bytes of RAM, as shown in Figure 5, without having to sacrifice all of Port 2.

All of these instructions execute in 2 µs, with a 12 MHz clock.

Note that in all external Data RAM accesses, the Accumulator is always either the destination or source of the data.

The read and write strobes to external RAM are activated only during the execution of a MOVX instruction. Normally these signals are inactive, and in fact if they're not going to be used at all, their pins are available as extra I/O lines. More about that later.

ADDRESS WIDTH	MNEMONIC	OPERATION	EXECUTION TIME (µs)
8 bits	MOVX A, @ Ri	Read external RAM @ Ri	2
8 bits	MOVX @ Ri, A	Write external RAM @ Ri	2
16 bits	MOVX A, @ DPTR	Read external RAM @ DPTR	2
16 bits	MOVX @ DPTR, A	Write external RAM @ DPTR	2

Table 5 : A list of the MHS C51 Data Transfer Instructions that Access External Data Memory Space.

Lookup Tables

Table 6 shows the two instructions that are available for reading lookup tables in Program Memory. Since these instructions access only Program Memory, the lookup tables can be read, not updated. The mnemonic is MOVC for "move constant".

If the table access is to external Program Memory, then the read strobe is $\overline{PSEN}$.

The first MOVC instruction in Table 6 can accomodate a table of up to 256 entries, numbered 0 through 255. The number of the desired entry is loaded into the Accumulator, and the Data Pointer is set up to point to beginning of the table. Then

MOVC A, @A + DPTR

copies the desired table entry into the Accumulator.

The other MOVC instruction works the same way, except the Program Counter (PC) is used as the table base, and the table is accesses through a subroutine. First the number of the desired entry is loaded into the Accumulator, and the subroutine is called :

```
MOV     A, ENTRY_NUMBER
CALL    TABLE
```

The subroutine "TABLE" would look like this :

```
TABLE : MOVC A, @A + PC
        RET
```

The table itself immediately follows the RET (return) instruction in Program Memory. This type of table can have up to 255 entries; numbered 1 through 255. Number 0 can not be used, because at the time the MOVC instruction is executed, the PC contains the address of the RET instruction. An entry numbered 0 would be the RET opcode itself.

MNEMONIC	OPERATION	EXECUTION TIME (µs)
MOVC A, @A + DPTR	Read Pgm Memory at (A + DPTR)	2
MOVC A, @A + PC	Read Pgm Memory at (A + PC)	2

Table 6 : The MHS C51 Lookup Table Read Instructions.

BOOLEAN INSTRUCTIONS

MHS C51 devices contain a complete Boolean (single-bit) processor. The internal RAM contains 128 addressable bits, and the SFR space can support up to 128 other addressable bits. All of the port lines are bit-addressable, and each one can be treated as a separate single-bit port. The instructions that access these bits are not just conditional branches, but a complete menu of move, set, clear, complement, OR and AND instructions. These kinds of bit operations are not easily obtained in other architectures with any amount of byte-oriented software.

The instruction set for the Boolean processor is shown in Table 7. All bit accesses are by direct addressing. Bit addresses 00H through 7FH are in the Lower 128, and bit addresses 80H through FFH are in SFR space.

MNEMONIC	OPERATION	EXECUTION TIME (µs)
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND (NOT bit)	2
ORL C,bit	C = C OR bit	2
ORL C,/bit	C = C OR (NOT bit)	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	bit = NOT bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1 ; CLR bit	2

Table 7 : A list of the MHS C51 Boolean Instructions.

Note how easily an internal flag can be moved to a port pin :

```
MOV     C, FLAG
MOV     P1.0, C
```

In this example, FLAG is the name of any addressable bit in the lower 128 or SFR space. An I/O line (the LSB of Port 1, in the case) is set or cleared depending on whether the flag bit is 1 or 0.

The Carry bit in the PSW is used as the single-bit Accumulator of the Boolean processor. Bit instructions that refer to the Carry bit as C assemble as Carry-specific instructions (CLR C, etc). The Carry bit also has a direct address, since it resides in the PSW register, which is bit-addressable.

Note that the Boolean instruction set includes ANL and ORL operations, but not the XRL (Exclusive OR) operation. An XRL operation is simple to implement in software. Suppose, for example, it is required to form the Exclusive OR of two bits :

C = bit1 XRL bit2

The software to do that could be as follows :

```
MOV     C, bit1
JNB     bit2, OVER
CPL     C
OVER : (continue)
```

First, bit 1 is moved to the Carry. If bit 2 = 0, then C now contains the correct result. That is, bit 1 XRL bit2 = bit1 if bit2 = 0. On the other hand, if bit2 = 1 C now contains the complement of the correct result. It need only be inverted (CPL C) to complete the operation.

This code uses the JNB instruction, one of a series of bit-test instructions which execute a jump if the addressed bit is set (JC, JB, JBC) or if the addressed bit is not set (JNC, JNB). In the above case, bit2 is being tested, and if bit2 = 0 the CPL C instruction is jumped over.

JBC executes the jump if the addresed bit is set, and also clears the bit. Thus a flag can be tested and cleared in one operation.

All the PSW bits are directly addressable, so the Parity bit, or the general purpose flags, for example, are also available to the bit-test instructions.

Relative offset

The destination address for these jumps is specified to the assembler by a label or by an actual address in Program Memory. However, the destination address assembles to a relative offset byte. This is a signed (two's complement) offset byte which is added to the PC in two's complement arithmetic if the jump is executed.

The range of the jump is therefore - 128 to + 127 Program Memory bytes relative to the first byte following the instruction.

JUMP INSTRUCTIONS

Table 8 shows the list of unconditional jumps.

MNEMONIC	OPERATION	EXECUTION TIME (μs)
JMP addr	Jump to addr	2
JMP @A + DPTR	Jump to A + DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

Table 8 : Unconditional Jumps in MHS C51.

The table lists a single "JMP addr" instruction, but in fact there are three - SJMP, LJMP, AJMP - which differ in the format of the destination address. JMP is a generic mnemonic which can be used if the programmer does not care which way the jump is encoded.

The SJMP instruction encodes the destination address as relative offset, as described above. The instruction is 2 bytes long, consisting of the opcode and the relative offset byte. The jump distance is limited to range of – 128 to + 127 bytes relative to the instruction following the SJMP.

The LJMP instruction encodes the destination address as a 16-bit constant. The instruction is 3 bytes long, consisting of the opcode and two address bytes. The destination address can be anywhere in the 64K Program Memory space.

The AJMP instruction encodes the destination address as an 11-bit constant. The instruction is 2 bytes long, consisting of the opcode, which itself contains 3 of the 11 address bits, followed by another byte containing the low 8 bits of the destination address. When the instruction is executed, these 11 bits are simply substituted for the low 11 bits in the PC. The high 5 bits stay the same. Hence the destination has to be within the same 2K block as the instruction following the AJMP.

In all cases the programmer specifies the destination address to the assembler in the same way : as a label or as a 16-bit constant. the assembler will put the destination address into the correct format for the given instruction. If the format required by the instruction will not support the distance to the specified destination address, a "Destination out of range" message is written, into the list file.

The JMP @ A + DPTR instruction supports case jumps. The destination address is computed at execution time as the sum of the 16-bit DPTR register and the Accumulator. Typically, DPTR is set up with the address of a jump table, and the Accumulator is given an index to the table. In a 5-way branch, for example, an integer 0 through 4 is loaded into the Accumulator.

The code to be executed might be as follows :

```
MOV     DPTR, # JUMP_TABLE
MOV     A, INDEX_NUMBER
RL      A
JMP     @ A + DPTR
```

The RLA instruction converts the index number (0 through 4) to an even number on the range 0 through 8, because each entry in the jump table is 2 bytes long :

```
JUMP_TABLE :
        AJMP CASE_0
        AJMP CASE_1
        AJMP CASE_2
        AJMP CASE_3
        AJMP CASE_4
```

Table 8 shows a single "CALLaddr" instruction, but there are two of them - LCALL and ACALL - which differ in the format in which the subroutine address is given to the CPU. CALL is a generic mnemonic which can be used if the programmer does not care which way the address is encoded.

The LCALL instruction uses the 16-bit address format, and the subroutine can be anywhere in the 64K Program Memory space. The ACALL instruction uses the 11-bit format, and the subroutine must be in the same 2K block as the instructon following the ACALL.

In any case the programmer specifies the subroutine address to the assembler in the same way : as a label or as a 16-bit constant. The assembler will put the address into the correct format for the given instructions.

Subroutines should end a RET instruction, which returns execution following the CALL.

RETI is used to return from an interrupt service routine. The only difference between RET and RETI is that RETI tells the interrupt control system that the interrupt in progress is done. If there is no interrupt in progress at the time RETI is executed, then the RETI is functionnally identical to RET.

Table 9 shows the list of conditional jumps available to the MHS C51 user. All of these jumps specify the destination address by the relative offset method, and so are limited to a jump distance of - 128 to + 127 bytes from the instruction following the conditional jump instruction. Important to note, however, the user specifies to the assembler the actual destination address the same way as the other jumps : as a label or a 16-bit constant.

MNEMONIC	OPERATION	ADDRESSING MODES				EXECUTION TIME (µs)
		DIR	IND	REG	IMM	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A ≠ 0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNZ A,<byte>,rel	Jump if A = <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> ≠ #data		X	X		2

Table 9 : Conditional Jumps in MHS C51 Devices.

There is no Zero bit in the PSW. The JZ and JNZ instructions test the Accumulator data for that condition.

The DJNZ instruction (Decrement and Jump if Not Zero) is for loop control. To execute a loop N times, load a counter byte with N and terminate the loop with DJNZ to the beginning of the loop, as shown below for N = 10 :

```
        MOV     COUNTER, # 10
LOOP :  (begin loop)
          *
          *
          *
        (end loop)
          DJNZ    COUNTER, LOOP
        (continue)
```

The CJNE instruction (Compare and Jump if Not Equal) can also be used for loop control as in Figure 12. Two bytes are specified in the operand field of the instruction. The jump is executed only if the two bytes are not equal. In the example of Figure 12, the two bytes were the data in R1 and the constant 2AH. The initial data in R1 was 2EH. Every time the loop was executed, R1 was decremented, and the looping was to continue until the R1 data reached 2AH.

Another application of this instruction is in "greater than, less than" comparisons. The two bytes in the operand field are taken as unsigned integers. If the first is less than the second, then the Carry bit is set (1). If the first is greater than or equal to the second, then the Carry bit is cleared.

CPU TIMING

All MHS C51 microcontrollers have an on-chip oscillator which can be used if desired as the clock source for the CPU. To use the on-chip oscillator, connect a crystal or ceramic resonator between the XTAL1 and XTAL2 pins of the microcontroller, and capacitors to ground as shown in Figure 13.

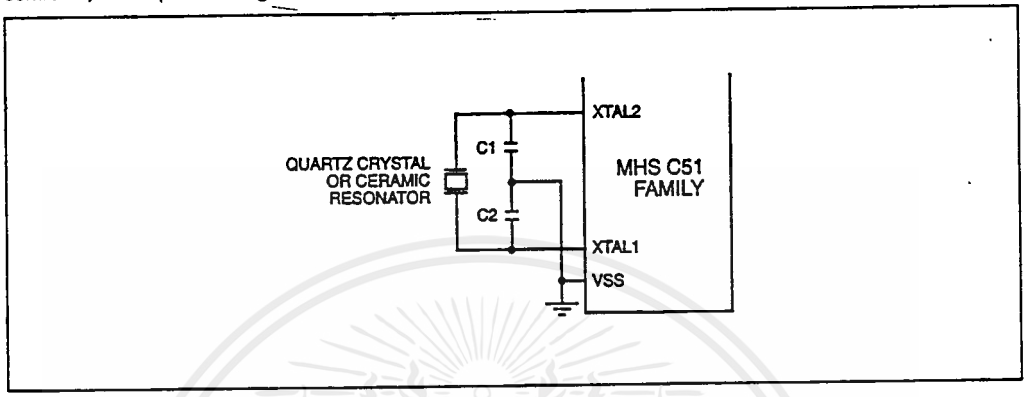


Figure 13 : Using the On-Chip Oscillator.

Examples of how to drive the clock with an external oscillator are shown in Figure 14. In the MHS C51 devices the signal at the XTAL1 pin drives the internal clock generator. If only one pin is going to be driven with the external oscillator signal, make sure it is the right pin.

The internal clock generator defines the sequence of states that make up the MHS C51 machine cycle.

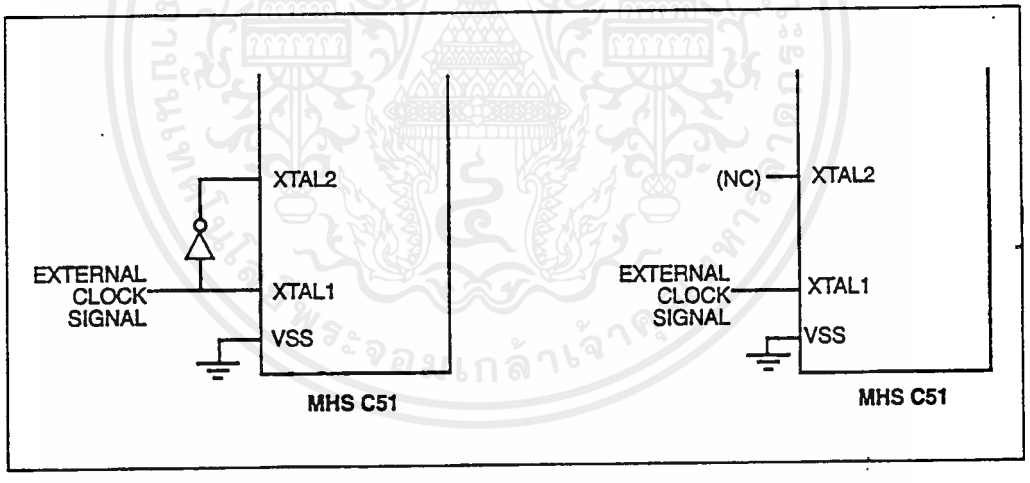


Figure 14 : Using an External Clock.

MACHINE CYCLES

A machine cycle consists of a sequence of 6 states, numbered S1 through S6. Each state time lasts for two oscillator periods. Thus a machine cycle takes 12 oscillator periods or 1 µs if the oscillator frequency is 12 MHz.

Each state is divided into a Phase 1 half and a Phase 2 half. Figure 15 shows the fetch/execute sequences in states and phases for various kinds of instructions. Normally two program fetches are generated during each machine cycle, even if the instruction being executed doesn't require it. If the instruction being executed doesn't need more code bytes, the CPU simply ignores the extra fetch, and the Program Counter is not incremented.

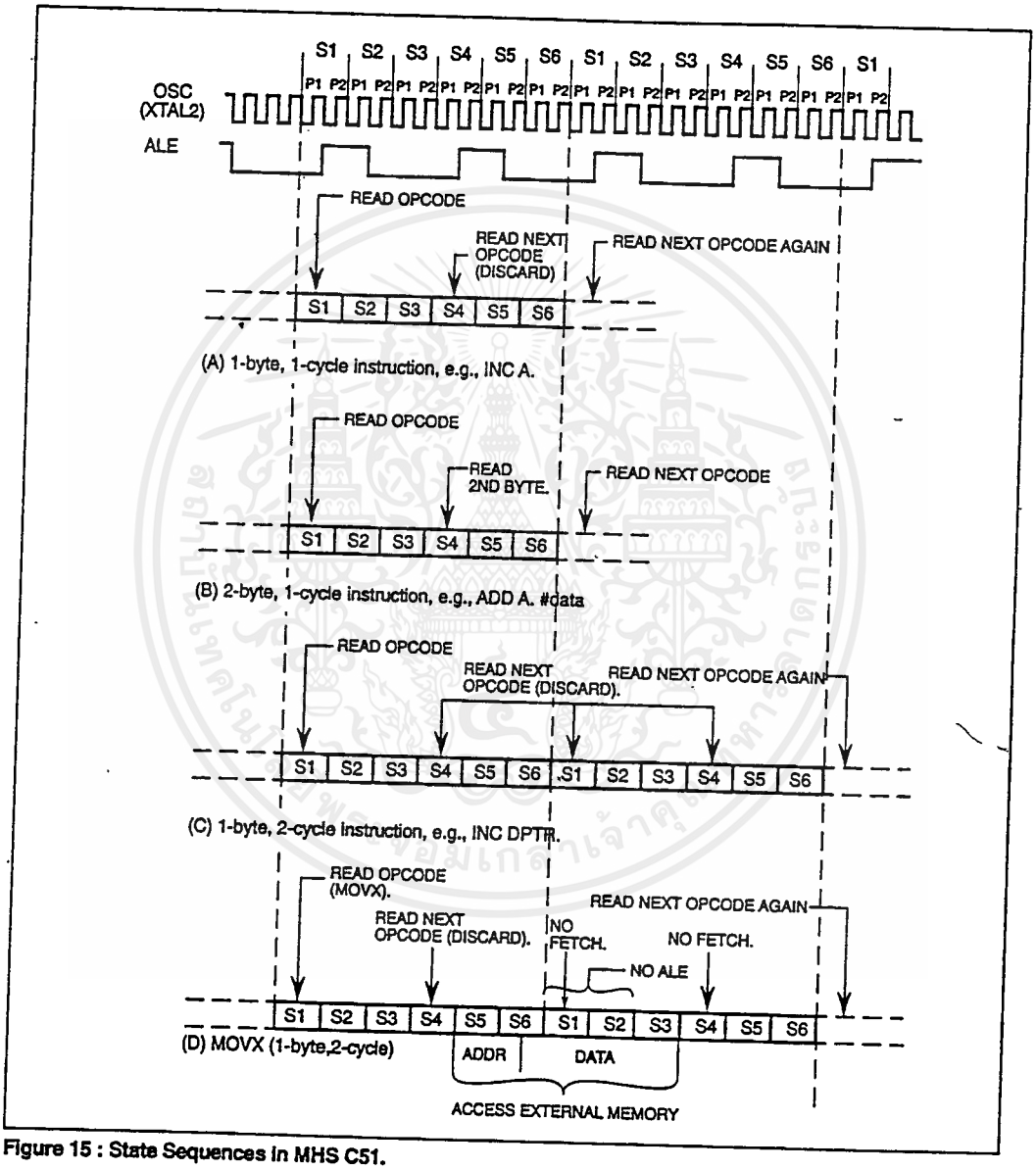


Figure 15 : State Sequences in MHS C51.

Execution of a one-cycle instruction (Figure 15A and B) begins during State 1 of the machine cycle, when the opcode is latched into the Instruction Register. A second fetch occurs during S4 of the same machine cycle. Execution is completed at the end of State 6 of this machine cycle.

The MOVX instructions take two machine cycles to execute. No program fetch is generated during the second cycle of a MOVX instruction. This is the only time program fetches are skipped. The fetch/execute sequence for MOVX instructions is shown in Figure 15 (D).

The fetch/execute sequences are the same whether the Program Memory is internal or external to the chip. Execution times do not depend on whether the Program Memory is internal or external.

Figure 16 shows the signals and timing involved in program fetches when the Program Memory is external. If Program Memory is external, then, the Program Memory read strobe $\overline{PSEN}$ is normally activated twice per machine cycle, as shown in Figure 16 (A).

If an access to external Data Memory occurs, as shown in Figure 16 (B), two $\overline{PSEN}$s are skipped, because the address and data bus are being used for the Data Memory access.

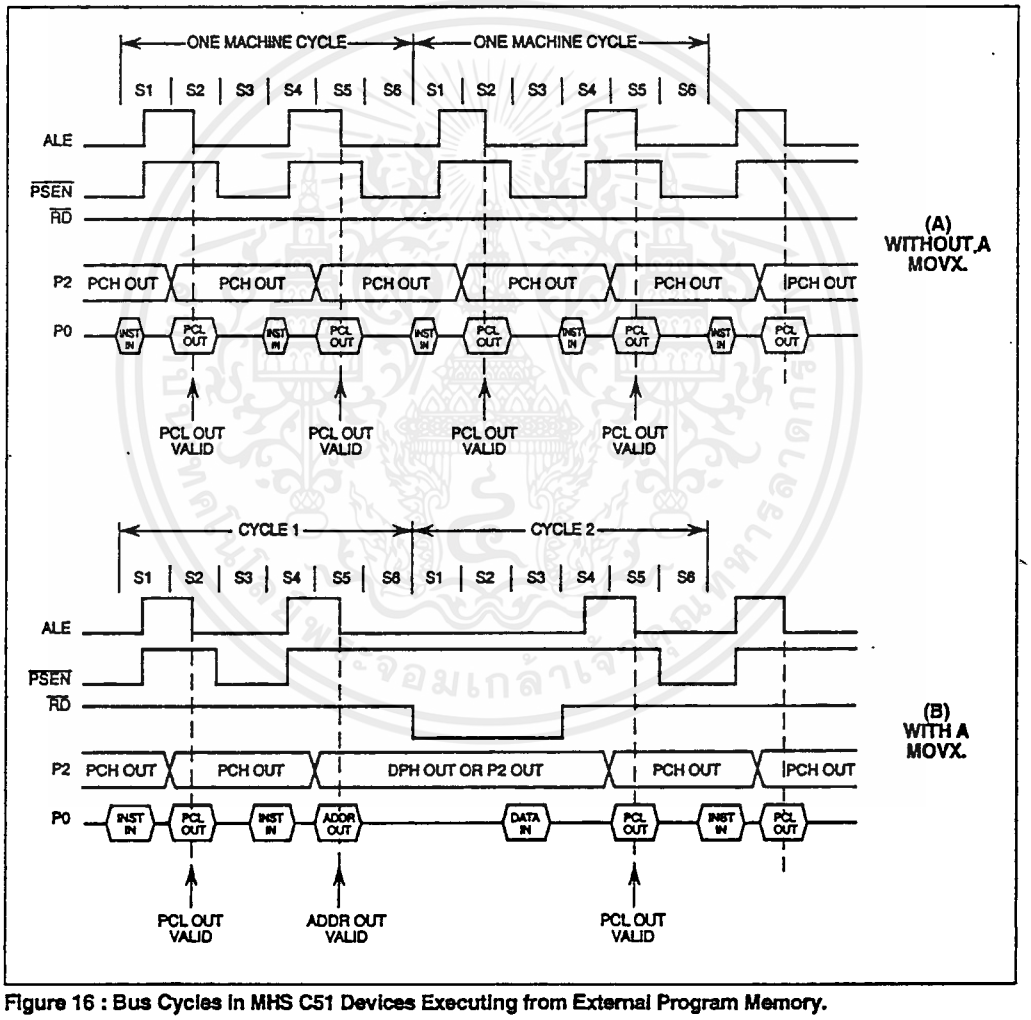


Figure 16 : Bus Cycles in MHS C51 Devices Executing from External Program Memory.

Note that a Data Memory bus cycle takes twice as much time as a Program Memory bus cycle. Figure 16 shows the relative timing of the addresses being emitted at ports 0 and 2, and of ALE and PSEN. ALE is used to latch the low address byte from P0 into the address latch.

When the CPU is executing from internal Program Memory, PSEN is not activated, and program addresses are not emitted. However, ALE continues to be activated twice per machine cycle and so is available as a clock output signal. Note, however, that one ALE is skipped during the execution of the MOVX instruction.

INTERRUPT STRUCTURE

The 80C51 and his ROMless version provide 5 interrupt sources : 2 external interrupts, 2 timer interrupts, and the serial port interrupt. the 80C52, 83C154 and 83C154D and their ROMless version provide these 5 plus a sixth interrupt that is associated with the third timer/counter which is present in those devices.

What follows is an overview of the interrupt structure for these devices. More detailed information for specific members of the MHS C51 family is provided in the chapters of this handbook that describe the specific devices.

Interrupt Enables

Each of the interrupt source can be individually enabled or disabled by setting or clearing a bit in the SFR named IE (Interrupt Enable). This register also contains a global disable bit, which can be cleared to disable all interrupts at once. Figure 17 shows the IE register for the 80C51, 80C52 and 83C154 or the 83C154D.

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1,each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.

Figure 17 : IE (Interrupt Enable) Register in the 80C51, 80C52, 83C154 and 83C154D.

Interrupt priorities

Each interrupt source can also be individually programmed to one of two priority level by setting or clearing a bit in the SFR named IP (Interrupt Priority). Figure 18 shows the IP register in the 80C51, 80C52, *83C154 and 83C154D.

A low-priority interrupt can be interrupted by a high-priority interrupt, but not by another low-priority interrupt.

A high-priority interrupt can't be interrupted by any other interrupt source.

If two interrupt requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence.

(MSB)							(LSB)
PCT	X	PT2	PS	PT1	PX1	PT0	PX0

Symbol	Position	Function
PCT	IP.7	83C154/C154D only. Priority interrupt circuit control bit. The priority register contents are valid and priority assigned interrupts can be processed when this bit is "0". When the bit is "1", the priority interrupt circuit is stopped, and interrupts can only be controlled by the interrupt enable register (IE).
–	IP.6	reserved
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PX1	IP.2	defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

Figure 18 : IP (Interrupt Priority) Register in the 80C51, 80C52, 83C154 and 83C154D.

Figure 19 shows, for the 80C51, 80C52, 83C154 and 83C154D, how the IE and IP registers and the polling sequence work to determine which interrupt will be serviced.

In operation, all the interrupt flags are latched into the interrupts control system during State 5 of every machine cycle. The samples are polled during the following machine cycle. If the flag for an enabled interrupt is found to be set (1), the interrupt system generates an LCALL to the appropriate location in Program Memory, unless some other condition blocks the interrupt. Several conditions can block an interrupt, among them that an interrupt of equal or higher priority level is already in progress.

The hardware-generated LCALL causes the contents of the Program Counter to be pushed onto the stack, and reloads the PC with the beginning address of the service routine. As previously noted (Figure 3), the service routine for each interrupt begins at a fixed location.

Only the Program Counter is automatically pushed onto that stack, not the PSW or any other register. Having only the PC be automatically saved allows the programmer to decide how much time to spend saving which other registers. This enhances the interrupt response time, albeit at the expense of increasing the programmer's burden of responsability. As a result, many interrupt functions that are typical in control applications-toggling a port pin, for example, or reloading a timer, or unloading a serial buffer can often be completed in less time than it takes other architectures to commence them.

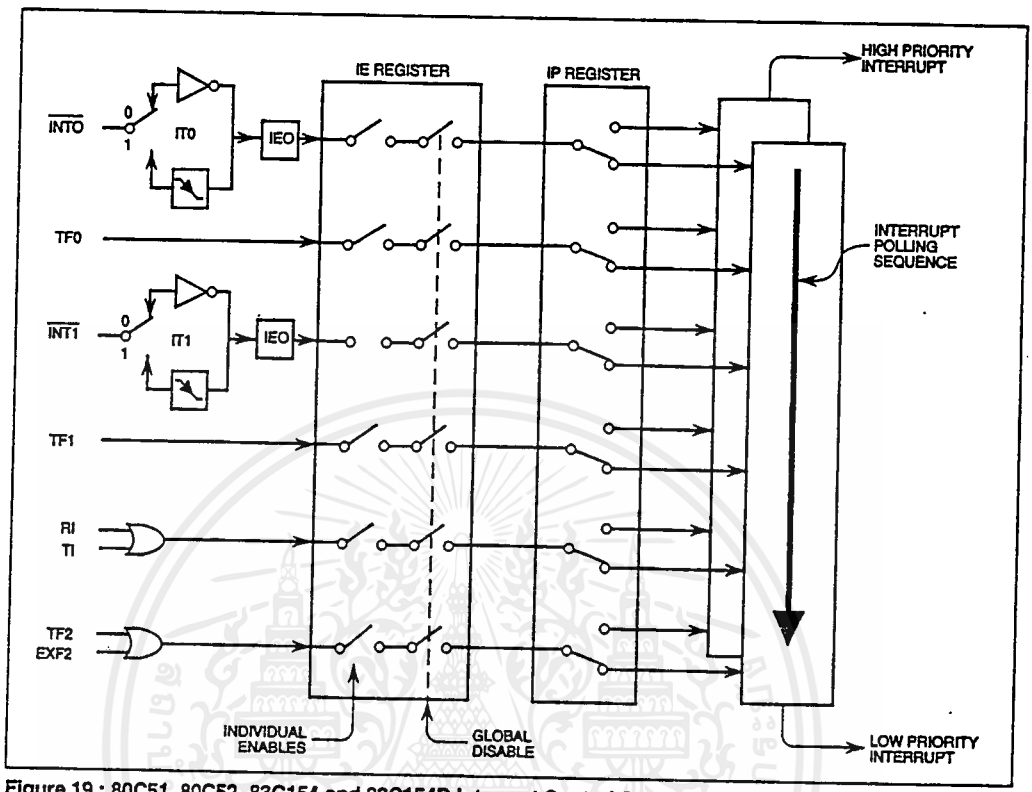


Figure 19 : 80C51, 80C52, 83C154 and 83C154D Interrupt Control System.

Simulating a third priority level in software

Some applications require more than the two priority levels that are provided by on-chip hardware in MHS C51 devices. In these cases, relatively simple software can be written to produce the same effect as a third priority level. First, interrupts that are to have higher priority than 1 are assigned to priority 1 in the IP (Interrupt Priority) register. The service routines for priority 1 interrupts that are supposed to be interruptible by "priority 2" interrupts are written to include the following code :

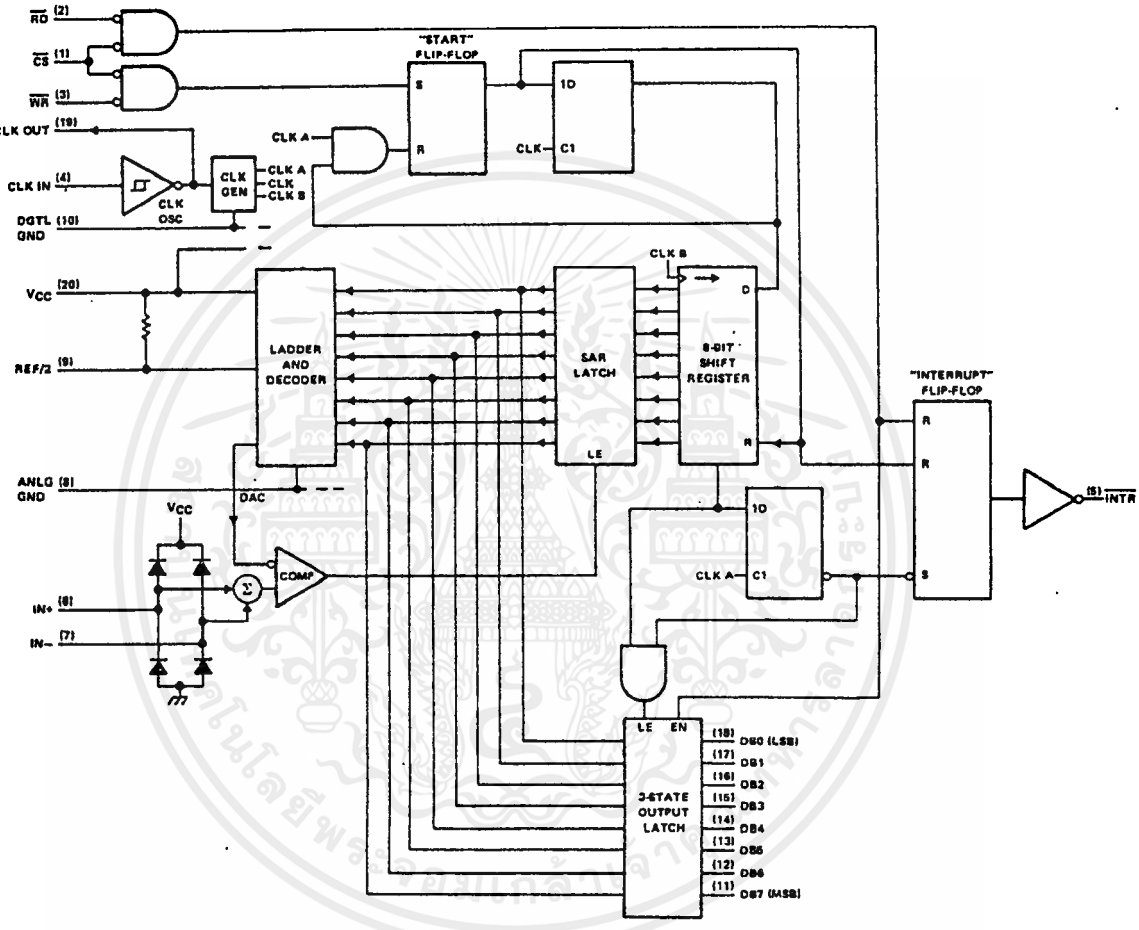
```
        PUSH    IE
        MOV     IE, # MASK
        CALL    LABEL
              **********
        (execute service routine)
              ********
        POP     IE
        RET
LABEL:  RETI
```

As soon as any priority 1 interrupt is acknowledged, the IE (Interrupt Enable) register is redefined so as to disable all but "priority 2" interrupts. Then, a CALL to LABEL executes the RETI instruction, which clears the priority 1 interrupt-in-progress flip-flop. At this point any priority 1 interrupt that is enabled can be serviced, but only "priority 2" interrupts are enabled.

POPping IE restores the original enable byte. Then a normal RET (rather than another RETI) is used to terminate the service routine. The additional software adds 10 µs (at 12 MHz) to priority 1 interrupts.

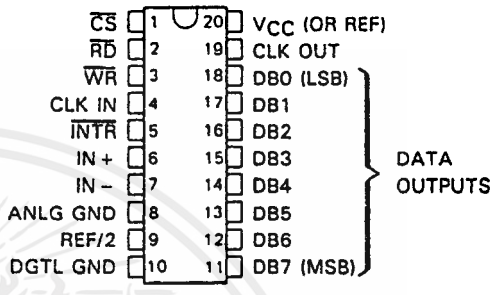
ADC0804I, ADC0804C
8-BIT ANALOG-TO-DIGITAL CONVERTER WITH DIFFERENTIAL INPUTS

functional block diagram (positive logic)



- 8-Bit Resolution
- Ratiometric Conversion
- 100-μs Conversion Time
- 135-ns Access Time
- No Zero Adjust Requirement
- On-Chip Clock Generator
- Single 5-V Power Supply
- Operates with Microprocessor or as Stand-Alone
- Designed to be Interchangeable with National Semiconductor and Signetics ADC0804

N DUAL-IN-LINE PACKAGE
(TOP VIEW)



description

The ADC0804 is a CMOS 8-bit successive-approximation analog-to-digital converter that uses a modified potentiometric (256R) ladder. The ADC0804 is designed to operate from common microprocessor control buses, with the three-state output latches driving the data bus. The ADC0804 can be made to appear to the microprocessor as a memory location or an I/O port. Detailed information on interfacing to most popular microprocessors is readily available from the factory.

A differential analog voltage input allows increased common-mode rejection and offset of the zero-input analog voltage value. Although a reference input (REF/2) is available to allow 8-bit conversion over smaller analog voltage spans or to make use of an external reference, ratiometric conversion is possible with the REF/2 input open. Without an external reference, the conversion takes place over a span from V_{CC} to analog ground (ANLG GND). The ADC0804 can operate with an external clock signal or, with an additional resistor and capacitor, can operate using an on-chip clock generator.

The ADC0804I is characterized for operation from −40°C to 85°C. The ADC0804C is characterized for operation from 0°C to 70°C.

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

Supply voltage, V_{CC} (see Note 1) . . . . . 6.5 V
Input voltage range: $\overline{CS}$, $\overline{RD}$, $\overline{WR}$ . . . . . −0.3 V to 18 V
other inputs . . . . . −0.3 V to V_{CC} + 0.3 V
Output voltage range . . . . . −0.3 V to V_{CC} + 0.3 V
Operating free-air temperature range: ADC0804I . . . . . −40°C to 85°C
ADC0804C . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds . . . . . 260°C

NOTE 1: All voltage values are with respect to digital ground (DGTL GND) with DGTL GND and ANLG GND connected together (unless otherwise noted).

recommended operating conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		4.5	5	6.3	V
Voltage at REF/2, $V_{REF/2}$ (see Note 2)		0.25	2.5		V
High-level input voltage at $\overline{CS}$, $\overline{RD}$, or $\overline{WR}$, V_{IH}		2		15	V
Low-level input voltage at $\overline{CS}$, $\overline{RD}$, or $\overline{WR}$, V_{IL}				0.8	V
Analog ground voltage (see Note 3)		−0.05	0	1	V
Analog input voltage (see Note 4)		−0.05		V_{CC}+0.05	V
Clock input frequency, f_{clock} (see Note 5)		100	640	1460	kHz
Duty cycle for f_{clock} ≥ 640 kHz (see Note 5)		40		60	%
Pulse duration clock input (high or low) for f_{clock} < 640 kHz, $t_{w(CLK)}$ (see Note 5)		275	781		ns
Pulse duration, $\overline{WR}$ input low (start conversion), $t_{w(WR)}$		100			ns
Operating free-air temperature, T_A	ADC0804I	−40		85	°C
	ADC0804C	0		70	

NOTES: 2. The internal reference voltage is equal to the voltage applied to REF/2, or approximately equal to one-half of the V_{CC} when REF/2 is left open. The voltage at REF/2 should be one-half the full-scale differential input voltage between the analog inputs. Thus, the differential input voltage when REF/2 is open and V_{CC} = 5 V is 0 to 5 V. VREF/2 for an input voltage range from 0.5 V to 3.5 V (full-scale differential voltage of 3 V) is 1.5 V.
3. These values are with respect to DGTL GND.
4. When the differential input voltage ($V_{IN+} - V_{in-}$) is less than or equal to 0 V, the output code is 0000 0000.
5. Total unadjusted error is specified only at an f_{clock} of 640 kHz with a duty cycle of 40% to 60% (pulse duration 625 ns to 937 ns). For frequencies above this limit or pulse duration below 625 ns, error may increase. The duty cycle limits should be observed for an f_{clock} greater than 640 kHz. Below 640 kHz, this duty cycle limit can be exceeded provided $t_{w(CLK)}$ remains within limits.

electrical characteristics over recommended operating free-air temperature range, V_{CC} = 5 V, f_{clock} = 640 kHz, REF/2 = 2.5 V (unless otherwise noted)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	All outputs	V_{CC} = 4.75 V, I_{OH} = −360 μA	2.4			V
		DB and $\overline{INTR}$	V_{CC} = 4.75 V, I_{OH} = −10 μA	4.5			
V_{OL}	Low-level output voltage	Data outputs	V_{CC} = 4.75 V, I_{OL} = 1.6 mA			0.4	V
		$\overline{INTR}$ output	V_{CC} = 4.75 V, I_{OL} = 1 mA			0.4	
		CLK OUT	V_{CC} = 4.75 V, I_{OL} = 360 μA			0.4	
V_{T+}	Clock positive-going threshold voltage			2.7	3.1	3.5	V
V_{T-}	Clock negative-going threshold voltage			1.5	1.8	2.1	V
$V_{T+} - V_{T-}$	Clock input hysteresis			0.6	1.3	2	V
I_{IH}	High-level input current				0.005	1	μA
I_{IL}	Low-level input current				−0.005	−1	μA
I_{OZ}	Off-state output current		V_O = 0			−3	μA
			V_O = 5 V			3	
I_{OHS}	Short-circuit output current	Output high	V_O = 0, T_A = 25°C	−4.5	−6		mA
I_{OLS}	Short-circuit output current	Output low	V_O = 5 V, T_A = 25°C	9	16		mA
I_{CC}	Supply current plus reference current		REF/2 open, $\overline{CS}$ at 5 V, T_A = 25°C		1.9	2.5	mA
$R_{REF/2}$	Input resistance to reference ladder		See Note 6	1	1.3		kΩ
C_i	Input capacitance (control)				5	7.5	pF
C_o	Output capacitance (DB)				5	7.5	pF

operating characteristics over recommended operating free-air temperature range, V_{CC} = 5 V, $V_{REF/2}$ = 2.5 V, f_{clock} = 640 kHz (unless otherwise noted)

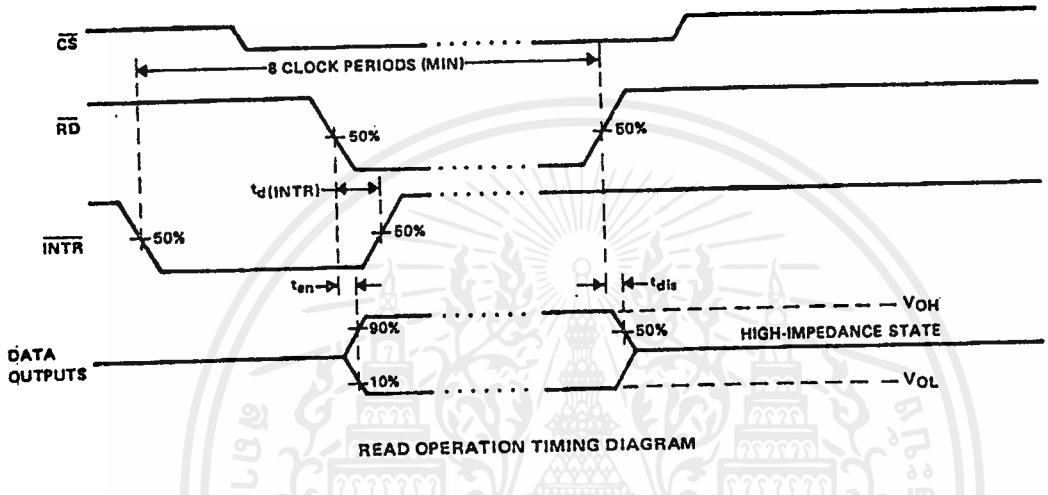
PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
	Supply-voltage-variation error (See Notes 2 and 7)	V_{CC} = 4.5 V to 5.5 V		±1/16	±1/8	LSB
	Total unadjusted error (See Notes 7 and 8)	$V_{REF/2}$ = 2.5 V			±1	LSB
	DC common-mode error (See Note 8)			±1/16	±1/8	LSB
t_{en}	Output enable time	C_L = 100 pF		135	200	ns
t_{dis}	Output disable time	C_L = 10 pF, R_L = 10 kΩ		125	200	ns
$t_{d(INTR)}$	Delay time to reset $\overline{INTR}$			300	450	ns
t_{conv}	Conversion cycle time (See Note 9)	f_{clock} = 100 kHz to 1.46 MHz	65½		72½	clock cycles
	Conversion time		103		114	μs
CR	Free-running conversion rate	$\overline{INTR}$ connected to $\overline{WR}$, $\overline{CS}$ at 0 V			8827	conv/s

†All typical values are at T_A = 25°C.

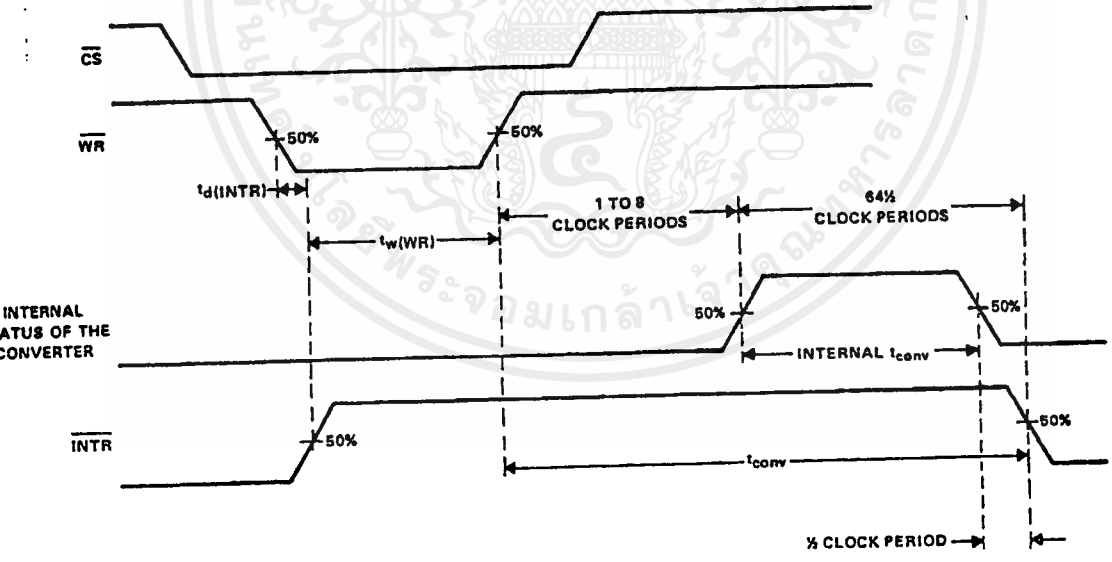
NOTES: 2. The internal reference voltage is equal to the voltage applied to REF/2, or approximately equal to one-half of the V_{CC} when REF/2 is left open. The voltage at REF/2 should be one-half the full-scale differential input voltage between the analog inputs. Thus, the differential input voltage when REF/2 is open and V_{CC} = 5 V is 0 to 5 V. $V_{REF/2}$ for an input voltage range from 0.5 V to 3.5 V (full-scale differential voltage of 3 V) is 1.5 V.

6. The resistance is calculated from the current drawn from a 5-V supply applied to pins 8 and 9.
7. These parameters are specified for the recommended analog input voltage range.
8. All errors are measured with reference to an ideal straight line through the end-points of the analog-to-digital transfer characteristic.
9. Although internal conversion is completed in 64 clock periods, a $\overline{CS}$ or $\overline{WR}$ low-to-high transition is followed by 1 to 8 clock periods before conversion starts. After conversion is completed, part of another clock period is required before a high-to-low transition of $\overline{INTR}$ completes the cycle.

timing diagrams



READ OPERATION TIMING DIAGRAM



WRITE OPERATION TIMING DIAGRAM

PRINCIPLES OF OPERATION

The ADC0804 contains a circuit equivalent to a 256-resistor network. Analog switches are sequenced by successive approximation logic to match an analog differential input voltage ($V_{in+} - V_{in-}$) to a corresponding tap on the 256-resistor network. The most-significant bit (MSB) is tested first. After eight comparisons (64 clock periods), an 8-bit binary code (1111 1111 = full scale) is transferred to an output latch and the interrupt ($\overline{INTR}$) output goes low. The device can be operated in a free-running mode by connecting the $\overline{INTR}$ output to the write ($\overline{WR}$) input and holding the conversion start ($\overline{CS}$) input at a low level. To ensure start-up under all conditions, a low-level $\overline{WR}$ input is required during the power-up cycle. Taking $\overline{CS}$ low anytime after that will interrupt a conversion in process.

When the $\overline{WR}$ input goes low, the ADC0804 successive approximation register (SAR) and 8-bit shift register are reset. As long as both $\overline{CS}$ and $\overline{WR}$ remain low, the ADC0804 remains in a reset state. One to eight clock periods after $\overline{CS}$ or $\overline{WR}$ makes a low-to-high transition, conversion starts.

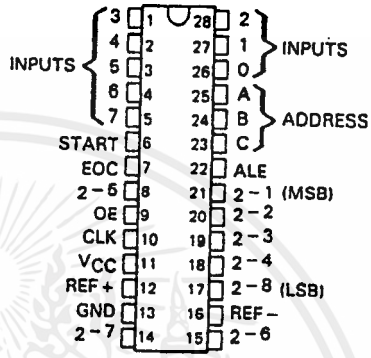
When the $\overline{CS}$ and $\overline{WR}$ inputs are low, the start flip-flop is set and the interrupt flip-flop and 8-bit register are reset. The next clock pulse transfers a logic high to the output of the start flip-flop. The logic high is ANDed with the next clock pulse, placing a logic high on the reset input of the start flip-flop. If either $\overline{CS}$ or $\overline{WR}$ have gone high, the set signal to the start flip-flop is removed, causing it to be reset. A logic high is placed on the D input of the 8-bit shift register and the conversion process is started. If the $\overline{CS}$ and $\overline{WR}$ inputs are still low, the start flip-flop, the 8-bit shift register, and the SAR remain reset. This action allows for wide $\overline{CS}$ and $\overline{WR}$ inputs with conversion starting from one to eight clock periods after one of the inputs goes high.

When the logic high input has been clocked through the 8-bit shift register, completing the SAR search, it is applied to an AND gate controlling the output latches and to the D input of a flip-flop. On the next clock pulse, the digital word is transferred to the three-state output latches and the interrupt flip-flop is set. The output of the interrupt flip-flop is inverted to provide an $\overline{INTR}$ output that is high during conversion and low when the conversion is completed.

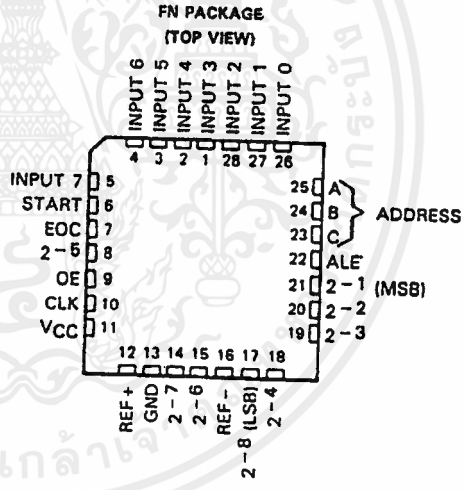
When a low is at both the $\overline{CS}$ and $\overline{RD}$ inputs, an output is applied to the DB0 through DB7 outputs and the interrupt flip-flop is reset. When either the $\overline{CS}$ or $\overline{RD}$ inputs return to a high state, the DB0 through DB7 outputs are disabled (returned to the high-impedance state). The interrupt flip-flop remains reset.

- Total Unadjusted Error . . . ±0.75 LSB Max for ADC0808 and ±1.25 LSB Max for ADC0809
- Resolution of 8 Bits
- 100 μs Conversion Time
- Ratiometric Conversion
- Monotonicity Over the Entire A/D Conversion Range
- No Missing Codes
- Easy Interface with Microprocessors
- Latched 3-State Outputs
- Latched Address Inputs
- Single 5-V Supply
- Low Power Consumption
- Designed to be Interchangeable with National Semiconductor ADC0808, ADC0809

N
DUAL-IN-LINE PACKAGE
(TOP VIEW)



FN PACKAGE
(TOP VIEW)



Description

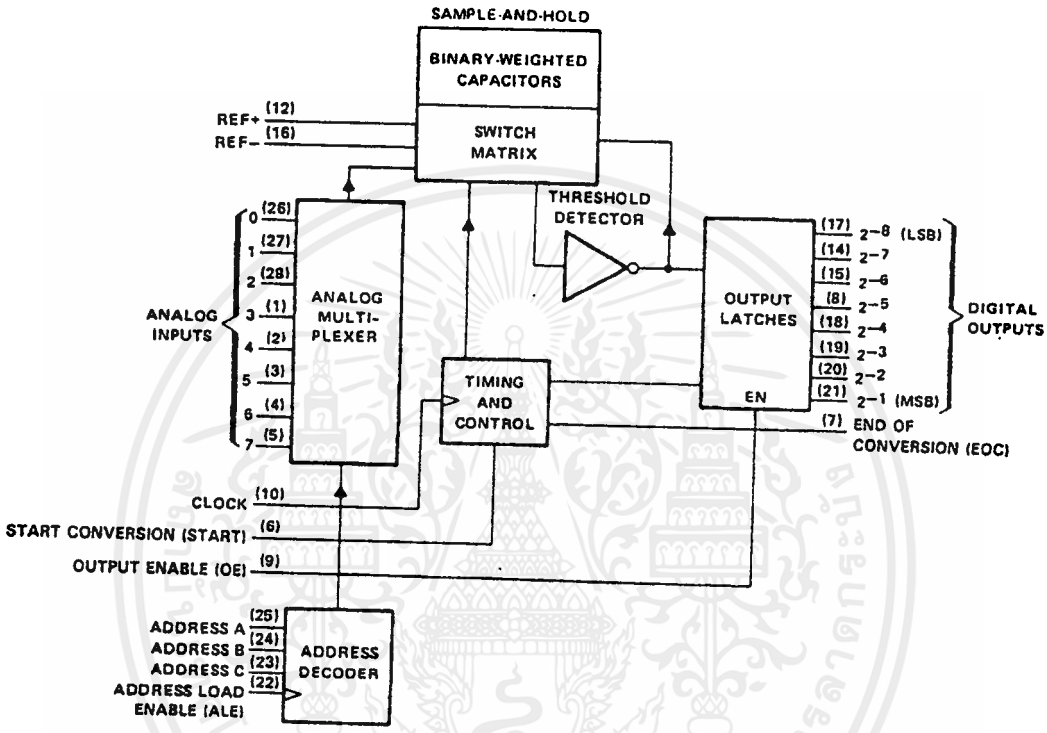
The ADC0808 and ADC0809 are monolithic CMOS devices with an 8-channel multiplexer, an 8-bit analog-to-digital (A/D) converter, and microprocessor-compatible control logic. The 8-channel multiplexer can be controlled by a microprocessor through a 3-bit address decoder with address load to select any one of eight single-ended analog switches connected directly to the comparator. The 8-bit A/D converter uses the successive-approximation conversion technique featuring a high-impedance threshold detector, a switched-capacitor array, a sample-and-hold, and a successive-approximation register (SAR). Detailed information on interfacing to most popular microprocessors is readily available from the factory.

The comparison and converting methods used eliminate the possibility of missing codes, nonmonotonicity, and the need for zero or full-scale adjustment. Also featured are latched 3-state outputs from the SAR and latched inputs to the multiplexer address decoder. The single 5-V supply and low power requirements make the ADC0808 and ADC0809 especially useful for a wide variety of applications. Ratiometric conversion is made possible by access to the reference voltage input terminals.

The ADC0808 and ADC0809 are characterized for operation from −40°C to 85°C.

ADC0808, ADC0809
CMOS ANALOG-TO-DIGITAL CONVERTERS
WITH 8-CHANNEL MULTIPLEXERS

functional block diagram (positive logic)

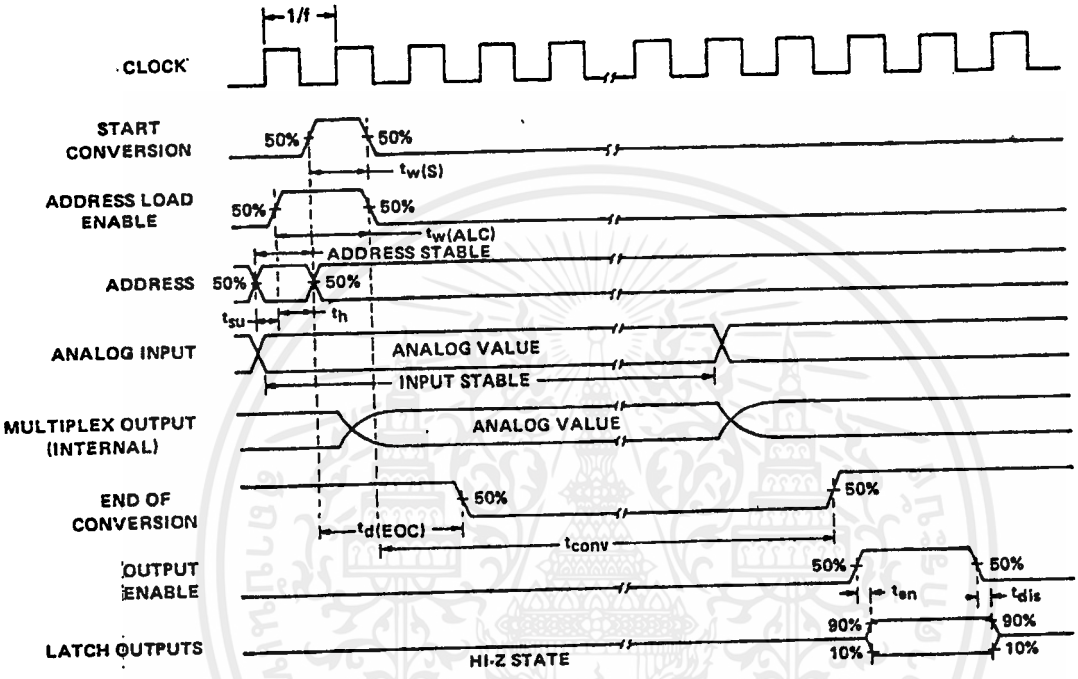


MULTIPLEXER FUNCTION TABLE

INPUTS				SELECTED
ADDRESS			ADDRESS	ANALOG
C	B	A	STROBE	CHANNEL
L	L	L	↑	0
L	L	H	↑	1
L	H	L	↑	2
L	H	H	↑	3
H	L	L	↑	4
H	L	H	↑	5
H	H	L	↑	6
H	H	H	↑	7

H = high level, L = low level
↑ = low-to-high transition

operating sequence



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

Supply voltage, V_{CC} (see Note 1) . . . . . 6.5 V
Input voltage range: control inputs . . . . . −0.3 to 15 V
all other inputs . . . . . −0.3 V to V_{CC} + 0.3 V
Operating free-air temperature range . . . . . −40°C to 85°C
Storage temperature range . . . . . −65°C to 150°C
Case temperature for 10 seconds: FN package . . . . . 260°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds: N package . . . . . 260°C

NOTE 1: All voltage values are with respect to network ground terminal.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.5	5	6	V
Positive reference voltage, V_{ref+} (see Note 2)		V_{CC}	V_{CC}+0.1	V
Negative reference voltage, V_{ref-}		0	−0.1	V
Differential reference voltage, $V_{ref+} - V_{ref-}$		5		V
High-level input voltage, V_{IH}	V_{CC}−1.5			V
Low-level input voltage, V_{IL}			1.5	V
Operating free-air temperature, T_A	−40		85	°C

NOTE 2: Care must be taken that this rating is observed even during power-up.

electrical characteristics over recommended operating free-air temperature range. V_{CC} = 4.75 V to 5.25 V (unless otherwise noted)

total device

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage		I_O = −360 µA	V_{CC}−0.4			V
V_{OL}	Low-level output voltage	Data outputs	I_O = 1.6 mA			0.45	V
		End of conversion	I_O = 1.2 mA			0.45	
I_{OZ}	Off-state (high-impedance-state) output current		V_O = V_{CC}			3	µA
			V_O = 0			−3	
I_I	Control input current at maximum input voltage		V_I = 15 V			1	µA
I_{IL}	Low-level control input current		V_I = 0			−1	µA
I_{CC}	Supply current		f_{clock} = 640 kHz		0.3	3	mA
C_I	Input capacitance, control inputs		T_A = 25°C		10	15	pF
C_O	Output capacitance, data outputs		T_A = 25°C		10	15	pF
	Resistance from pin 12 to pin 16				1000		kΩ

analog multiplexer

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_{on}	Channel on-state current (see Note 3)	V_I = V_{CC},	f_{clock} = 640 kHz			2	µA
		V_I = 0.1 V,	f_{clock} = 640 kHz			−2	
I_{off}	Channel off-state current	V_{CC} = 5 V, T_A = 25°C	V_I = 5 V		10	200	nA
			V_I = 0		−10	−200	
		V_{CC} = 5 V	V_I = 5 V			1	µA
			V_I = 0			−1	

†Typical values are at V_{CC} = 5 V and T_A = 25°C.
NOTE 3: Channel on-state current is primarily due to the bias current into or out of the threshold detector, and it varies directly with clock frequency.

timing requirements, V_{CC} = V_{ref+} = 5 V, V_{ref-} = 0 V (unless otherwise noted)

	PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_{clock}	Clock frequency		10	640	1280	kHz
t_{conv}	Conversion time	See Note 4	90	100	116	μs
$t_{w(s)}$	Pulse duration, START		200			ns
$t_{w(ALE)}$	Pulse duration, ALE		200			ns
t_{su}	Setup time, ADDRESS		50			ns
t_h	Hold time, ADDRESS		50			ns
t_d	Delay time, EOC	See Notes 4 and 5	0		14.5	μs

operating characteristics, T_A = 25°C, V_{CC} = V_{ref+} = 5 V, V_{ref-} = 0 V, f_{clock} = 640 kHz (unless otherwise noted)

	PARAMETER	TEST CONDITIONS	ADC0808			ADC0809			UNIT
			MIN	TYP†	MAX	MIN	TYP†	MAX	
k_{SVS}	Supply voltage sensitivity	V_{CC} = V_{ref+} = 4.75 V to 5.25 V, T_A = −40°C to 85°C, See Note 6		±0.05			±0.05		%/V
	Linearity error (see Note 7)			±0.25			±0.5		LSB
	Zero error (see Note 8)			±0.25			±0.25		LSB
	Total unadjusted error (See Note 9)	T_A = 25°C		±0.25	±0.5		±0.5		LSB
		T_A = −40°C to 85°C			±0.75			±1.25	
		T_A = 0°C to 70°C						±1	
t_{en}	Output enable time	C_L = 50 pF, R_L = 10 kΩ		80	250		80	250	ns
t_{dis}	Output disable time	C_L = 10 pF, R_L = 10 kΩ		105	250		105	250	ns

†Typical values for all except supply voltage sensitivity are at V_{CC} = 5 V, and all are at T_A = 25°C.

NOTES: 4. Refer to the operating sequence diagram.
5. For clock frequencies other than 640 kHz, $t_{d(EOC)}$ maximum is 8 clock periods plus 2 μs.
6. Supply voltage sensitivity relates to the ability of an analog-to-digital converter to maintain accuracy as the supply voltage varies. The supply and V_{ref+} are varied together and the change in accuracy is measured with respect to full-scale.
7. Linearity error is the maximum deviation from a straight line through the end points of the A/D transfer characteristic.
8. Zero error is the difference between 00000000 and the converted output for zero input voltage; full-scale error is the difference between 1111.1111 and the converted output for full-scale input voltage.
9. Total unadjusted error is the maximum sum of linearity error, zero error, and full-scale error.

PRINCIPLES OF OPERATION

The ADC0808 and ADC0809 each consists of an analog signal multiplexer, an 8-bit successive-approximation converter, and related control and output circuitry.

multiplexer

The analog multiplexer selects 1 of 8 single-ended input channels as determined by the address decoder. Address load control loads the address code into the decoder on a low-to-high transition. The output latch is reset by the positive-going edge of the start pulse. Sampling also starts with the positive-going edge of the start pulse and lasts for 32 clock periods. The conversion process may be interrupted by a new start pulse before the end of 64 clock periods. The previous data will be lost if a new start of conversion occurs before the 64th clock pulse. Continuous conversion may be accomplished by connecting the End-of-Conversion output to the start input. If used in this mode an external pulse should be applied after power up to assure start up.

converter

The CMOS threshold detector in the successive-approximation conversion system determines each bit by examining the charge on a series of binary-weighted capacitors (Figure 1). In the first phase of the conversion process, the analog input is sampled by closing switch S_C and all S_T switches, and by simultaneously charging all the capacitors to the input voltage.

In the next phase of the conversion process, all S_T and S_C switches are opened and the threshold detector begins identifying bits by identifying the charge (voltage) on each capacitor relative to the reference voltage. In the switching sequence, all eight capacitors are examined separately until all 8 bits are identified, and then the charge-convert sequence is repeated. In the first step of the conversion phase, the threshold detector looks at the first capacitor (weight = 128). Node 128 of this capacitor is switched to the reference voltage, and the equivalent nodes of all the other capacitors on the ladder are switched to REF−. If the voltage at the summing node is greater than the trip-point of the threshold detector (approximately one-half the V_{CC} voltage), a bit is placed in the output register, and the 128-weight capacitor is switched to REF−. If the voltage at the summing node is less than the trip point of the threshold detector, this 128-weight capacitor remains connected to REF+ through the remainder of the capacitor-sampling (bit-counting) process. The process is repeated for the 64-weight capacitor, the 32-weight capacitor, and so forth down the line, until all bits are counted.

With each step of the capacitor-sampling process, the initial charge is redistributed among the capacitors. The conversion process is successive approximation, but relies on charge redistribution rather than a successive-approximation register (and reference DAC) to count and weigh the bits from MSB to LSB.

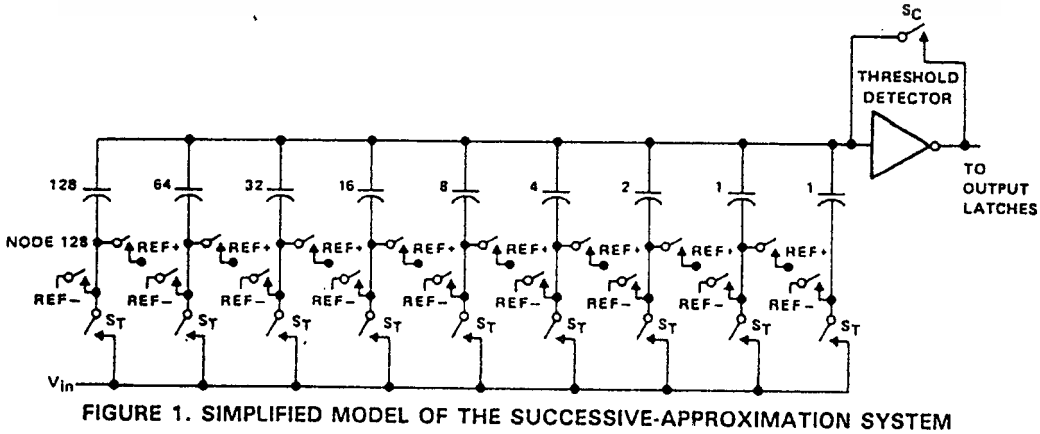


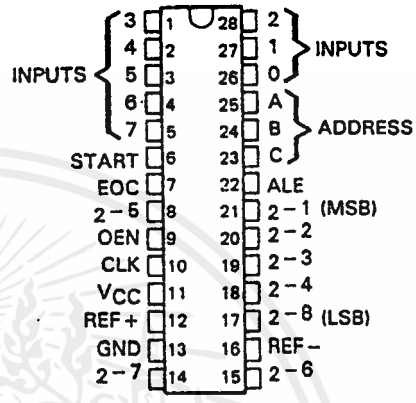
FIGURE 1. SIMPLIFIED MODEL OF THE SUCCESSIVE-APPROXIMATION SYSTEM

ADC0808M
CMOS ANALOG-TO-DIGITAL CONVERTER WITH 8-CHANNEL MULTIPLEXER

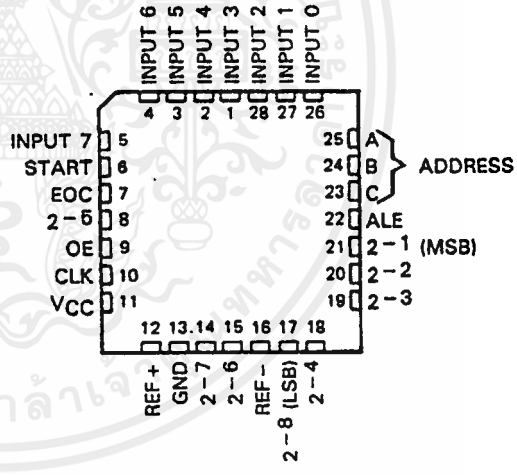
D2642, NOVEMBER 1986—REVISED MAY 1988

- Total Unadjusted Error . . . ±0.75 LSB Max
- Resolution of 8 Bits
- 100 μs Conversion Time
- Ratiometric Conversion
- Monotonous Over the Entire A/D Conversion Range
- No Missing Codes
- Easy Interface with Microprocessors
- Latched 3-State Outputs
- Latched Address Inputs
- Single 5-Volt Supply
- Low Power Consumption
- Designed to be Interchangeable with National Semiconductor ADC0808CJ

J
DUAL-IN-LINE PACKAGE
(TOP VIEW)



FK PACKAGE
(TOP VIEW)



description

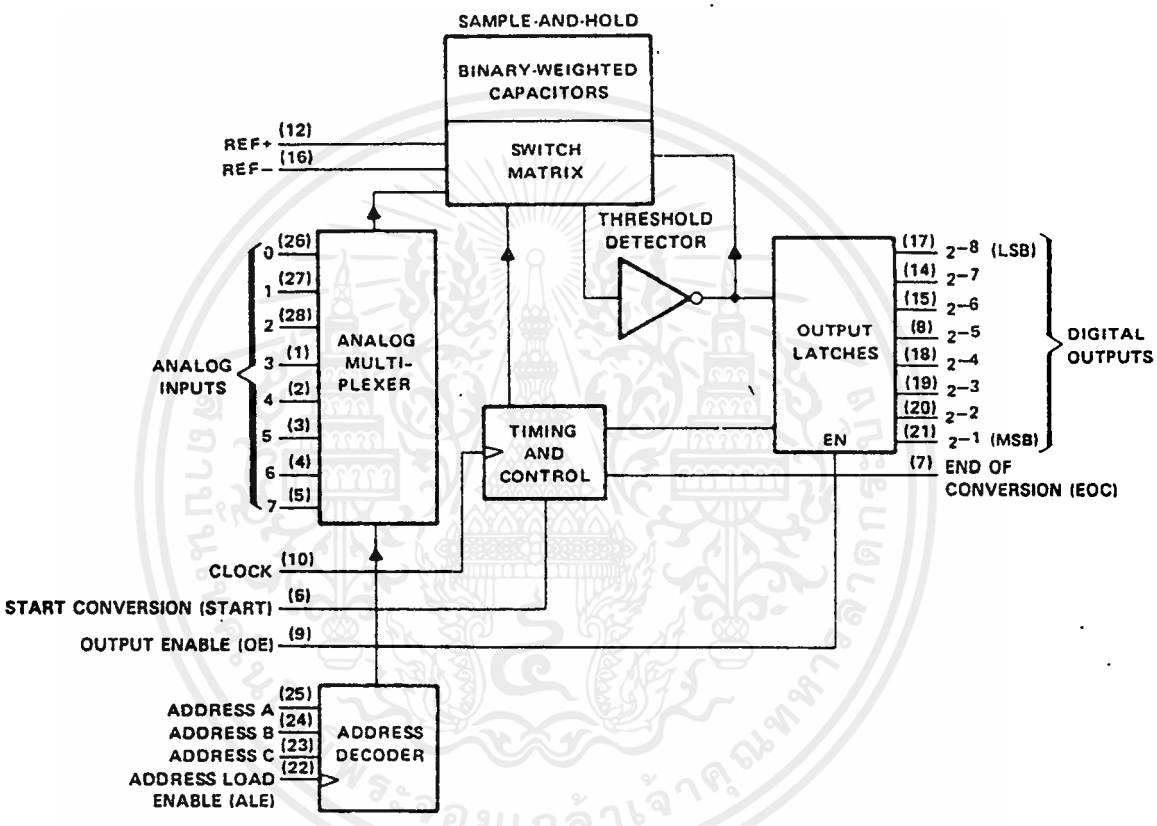
The ADC0808M is a monolithic CMOS device with an 8-channel multiplexer, an 8-bit analog-to-digital (A/D) converter, and microprocessor-compatible control logic. The 8-channel multiplexer can be controlled by a microprocessor through a 3-bit address decoder with address load to select any one of eight single-ended analog switches connected directly to the comparator. The 8-bit A/D converter uses the successive-approximation conversion technique featuring a high-impedance threshold detector, a switched capacitor array, a sample-and-hold, and a successive-approximation register (SAR). Detailed information on interfacing to most popular microprocessors is readily available from the factory.

The comparison and converting methods used eliminate the possibility of missing codes, nonmonotonicity, and the need for zero or full-scale adjustment. Also featured are latched 3-state outputs from the SAR and latched inputs to the multiplexer address decoder. The single 5-volt supply and low power requirements make the ADC0808M especially useful for a wide variety of applications. Ratiometric conversion is made possible by access to the reference voltage input terminals.

The ADC0808M is characterized for operation over the full military temperature range of −55°C to 125°C.

ADC0808M
CMOS ANALOG-TO-DIGITAL CONVERTER WITH 8-CHANNEL MULTIPLEXER

functional block diagram (positive logic)

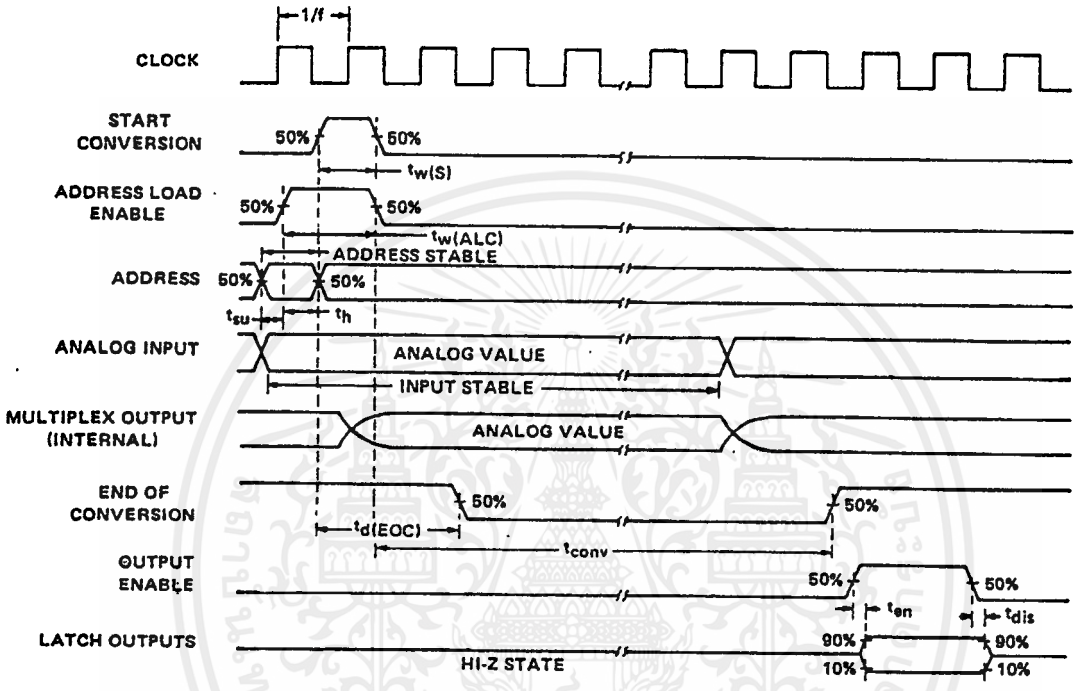


MULTIPLEXER FUNCTION TABLE

INPUTS				SELECTED
ADDRESS			ADDRESS	ANALOG
C	B	A	STROBE	CHANNEL
L	L	L	↑	0
L	L	H	↑	1
L	H	L	↑	2
L	H	H	↑	3
H	L	L	↑	4
H	L	H	↑	5
H	H	L	↑	6
H	H	H	↑	7

H = high level, L = low level
↑ = low-to-high transition

operating sequence



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

Supply voltage, V_{CC} (see Note 1) . . . . . 6.5 V
Input voltage range: control inputs . . . . . −0.3 to 15 V
all other inputs . . . . . −0.3 V to V_{CC} + 0.3 V
Operating free-air temperature range . . . . . −55°C to 125°C
Storage temperature range . . . . . −65°C to 150°C
Case temperature for 60 seconds: FK package . . . . . 260°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds: J package . . . . . 300°C

NOTE 1: /All voltage values are with respect to network ground terminal.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.5	5	6	V
Positive reference voltage, V_{ref+} (see Note 2)		V_{CC}	V_{CC}+0.1	V
Negative reference voltage, V_{ref-}		0	−0.1	V
Differential reference voltage, $V_{ref+} - V_{ref-}$		5		V
High-level input voltage, V_{IH}	V_{CC} − 1.5			V
Low-level input voltage, V_{IL}			1.5	V
Start pulse duration, $t_{w(S)}$	200			ns
Address load control pulse duration, $t_{w(ALC)}$	200			ns
Address setup time, t_{su}	50			ns
Address hold time, t_h	50			ns
Clock frequency, f_{clock}	10	640	1280	kHz
Operating free-air temperature, T_A	−55		125	°C

NOTE 2: Care must be taken that this rating is observed even during power-up.

electrical characteristics over recommended operating free-air temperature range, V_{CC} = 4.5 V to 5.5 V (unless otherwise noted)

total device

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage		I_O = −360 μA	V_{CC}−0.4			V
V_{OL}	Low-level output voltage	Data outputs	I_O = 1.6 mA			0.45	V
		End of conversion	I_O = 1.2 mA			0.45	
I_{OZ}	Off-state (high-impedance-state) output current		V_O = V_{CC}			3	μA
			V_O = 0			−3	
I_I	Control input current at maximum input voltage		V_I = 15 V			1	μA
I_{IL}	Low-level control input current		V_I = 0			−1	μA
I_{CC}	Supply current		f_{clock} = 640 kHz		0.3	3	mA
C_I	Input capacitance, control inputs		T_A = 25°C		10		pF
C_O	Output capacitance, data outputs		T_A = 25°C		10		pF
	Resistance from pin 12 to pin 16				1000		kΩ

analog multiplexer

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_{on}	Channel on-state current (see Note 3)	V_I = V_{CC},	f_{clock} = 640 kHz			2	μA
		V_I = 0,	f_{clock} = 640 kHz			−2	
I_{off}	Channel off-state current	V_{CC} = 5 V, T_A = 25°C	V_I = 5 V		10	200	nA
			V_I = 0		−10	−200	
		V_{CC} = 5 V	V_I = 5 V			1	μA
			V_I = 0			−1	

† Typical values are at V_{CC} = 5 V and T_A = 25°C.

NOTE 3: Channel on-state current is primarily due to the bias current into or out of the threshold detector, and it varies directly with clock frequency.

timing characteristics, V_{CC} = V_{ref+} = 5 V, V_{ref-} = 0 V, T_A = 25°C (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_{clock}	Clock frequency		10	640	1280	kHz
t_{conv}	Conversion time	See Notes 4 and 5 and Figure 1	90	100	116	μs
t_{enH}	Enable time, high	See Figure 1		150	360	ns
t_{enL}	Enable time, low	See Figure 1		90	250	ns
t_{dis}	Output disable time	See Figure 1		200	405	ns
$t_{w(s)}$	Pulse duration, START		200			ns
$t_{w(ALE)}$	Pulse duration, ALE		200			ns
t_{su}	Setup time, ADDRESS		50			ns
t_h	Hold time, ADDRESS		50			ns
$t_{d(EOC)}$	Delay time, EOC	See Notes 4 and 6 and Figure 1	0		14.5	μs

NOTES: 4. Refer to the operating sequence diagram.

5. For clock frequencies other than 640 kHz, t_{conv} is 57 clock cycles minimum and 74 clock cycles maximum.

6. For clock frequencies other than 640 kHz, $t_{d(EOC)}$ maximum is 8 clock cycles plus 2 μs.

operating characteristics, T_A = 25°C, V_{CC} = V_{ref+} = 5 V, V_{ref-} = 0 V, f_{clock} = 640 kHz (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
k_{SVS}	Supply voltage sensitivity	V_{CC} and V_{ref+} = 4.5 V to 5.5 V, T_A = −55°C to 125°C, See Note 7		±0.05		%/V
	Linearity error (see Note 8)			±0.25		LSB
	Zero error (see Note 9)			±0.25		LSB
	Total unadjusted error (see Note 10)	T_A = 25°C		±0.25	±0.5	LSB
		T_A = −55°C to 125°C			±0.75	

† Typical values for all except supply voltage sensitivity are at V_{CC} = 5 V, and all are at T_A = 25°C.

NOTES: 7. Supply voltage sensitivity relates to the ability of an analog-to-digital converter to maintain accuracy as the supply voltage varies. The supply and V_{ref+} are varied together and the change in accuracy is measured with respect to full-scale.
8. Linearity error is the maximum deviation from a straight line through the end points of the A/D transfer characteristic.
9. Zero error is the difference between 00000000 and the converted output for zero input voltage; full-scale error is the difference between 11111111 and the converted output for full-scale input voltage.
10. Total unadjusted error is the maximum sum of linearity error, zero error, and full-scale error.

PARAMETER MEASUREMENT INFORMATION

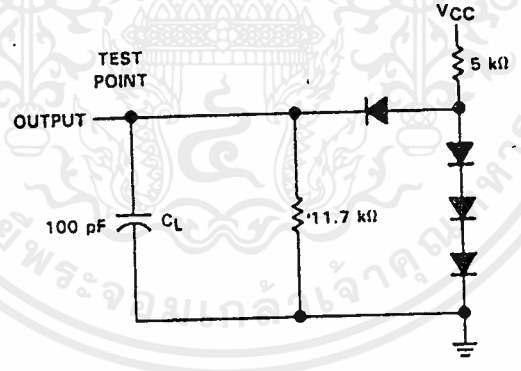


FIGURE 1. TEST CIRCUIT

PRINCIPLES OF OPERATION

The ADC0808M consists of an analog signal multiplexer, an 8-bit successive-approximation converter, and related control and output circuitry.

multiplexer

The analog multiplexer selects 1 of 8 single-ended input channels as determined by the address decoder. Address load control loads the address code into the decoder on a low-to-high transition. The output latch is reset by the positive-going edge of the start pulse. Sampling also starts with the positive-going edge of the start pulse and lasts for 32 clock periods. The conversion process may be interrupted by a new start pulse before the end of 64 clock periods. The previous data will be lost if a new start of conversion occurs before the 64th clock pulse. Continuous conversion may be accomplished by connecting the End-of-Conversion output to the start input. If used in this mode an external pulse should be applied after power up to assure start up.

converter

The CMOS threshold detector in the successive-approximation conversion system determines each bit by examining the charge on a series of binary-weighted capacitors (Figure 2). In the first phase of the conversion process, the analog input is sampled by closing switch S_C and all S_T switches, and by simultaneously charging all the capacitors to the input voltage.

In the next phase of the conversion process, all S_T and S_C switches are opened and the threshold detector begins identifying bits by identifying the charge (voltage) on each capacitor relative to the reference voltage. In the switching sequence, all eight capacitors are examined separately until all 8 bits are identified, and then the charge-convert sequence is repeated. In the first step of the conversion phase, the threshold detector looks at the first capacitor (weight = 128). Node 128 of this capacitor is switched to the reference voltage, and the equivalent nodes of all the other capacitors on the ladder are switched to REF−. If the voltage at the summing node is greater than the trip-point of the threshold detector (approximately one-half the V_{CC} voltage), a bit is placed in the output register, and the 128-weight capacitor is switched to REF−. If the voltage at the summing node is less than the trip point of the threshold detector, this 128-weight capacitor remains connected to REF+ through the remainder of the capacitor-sampling (bit-counting) process. The process is repeated for the 64-weight capacitor, the 32-weight capacitor, and so forth down the line, until all bits are counted.

With each step of the capacitor-sampling process, the initial charge is redistributed among the capacitors. The conversion process is successive approximation, but relies on charge redistribution rather than a successive-approximation register (and reference DAC) to count and weigh the bits from MSB to LSB.

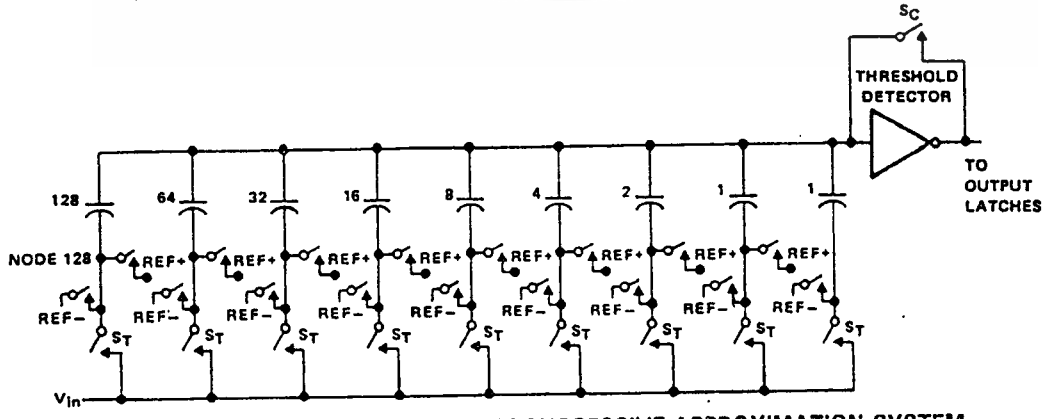


FIGURE 2. SIMPLIFIED MODEL OF THE SUCCESSIVE-APPROXIMATION SYSTEM

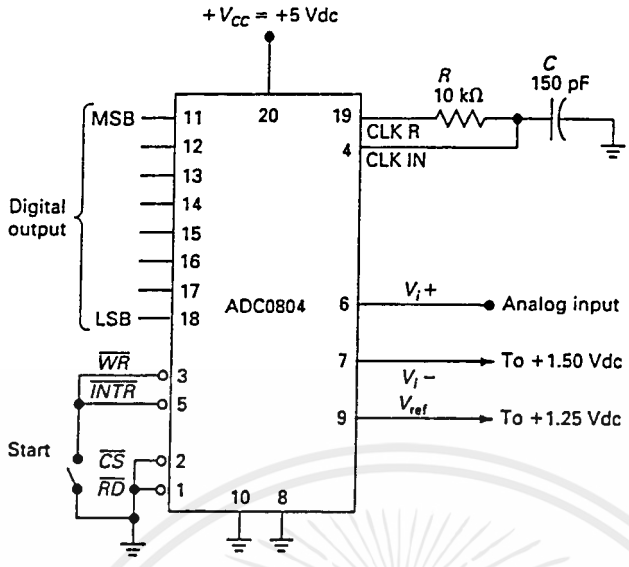
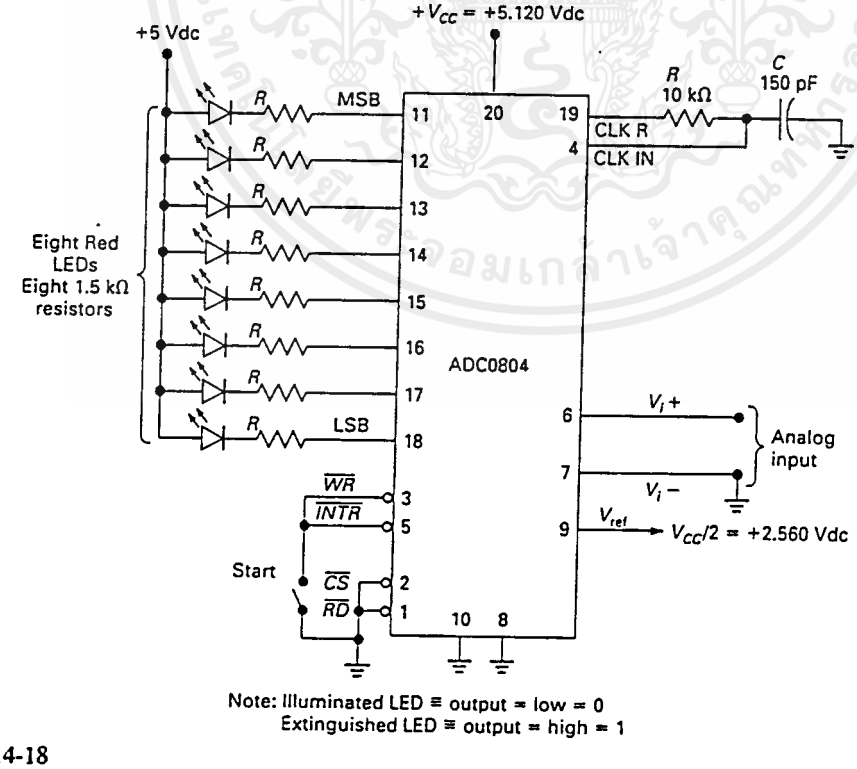


Fig. 14-16



Note: Illuminated LED ≡ output = low = 0
Extinguished LED ≡ output = high = 1

Fig. 14-18

เอกสารอ้างอิง

1. Roger C. Dugan, ***Electrical Power System Quality***, pp. 1-38, McGraw-Hill, Inc. 1996
2. Willis J. Tomkins, ***Interfacing Sensors to the IBM PC***, pp. 128-141, Prentice-Hall, New Jersey, Inc. 1988
3. แอดวานซ์เอ็นจิเนียริ่งกรุ๊ป, ***อิเล็กทรอนิกส์เบื้องต้น***, หน้า 42-44, 255-257, ฟิสิกส์เซ็นเตอร์, 2536
4. ผ.ศ.สมเกียรติ ศุภเดช, ***ทฤษฎีและการออกแบบวงจรพัลซ์***, หน้า 67-73, อิเล็กทรอนิกส์เวิร์ด, 2526

ประวัติผู้จัดทำโครงงานพิเศษ

นางสาวอรษา ใจใหญ่ เกิดวันที่ 8 สิงหาคม พ.ศ.2517 สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนสตรีวัดระฆัง จากนั้นเข้าศึกษาต่อในระดับปริญญาตรี สาขาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และสำเร็จการศึกษาในปี 2540

นายอำนาจ เจนจิโรจพิพัฒน์ เกิดวันที่ 21 กันยายน พ.ศ.2519 สำเร็จการศึกษาระดับมัธยมศึกษาตอนต้นและตอนปลายจากโรงเรียนวัดสุทธิวราราม จากนั้นเข้าศึกษาต่อในระดับปริญญาตรี สาขาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และสำเร็จการศึกษาในปี 2540