เครื่องกำเนิดสัญญาณพัลซ์ชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลข

PROGRAMMABLE PULSE GENERATOR BY DIRECT DIGITAL SYNTHESIS METHOD

โดย

นายวิชัย ซื่อสัตย์

นายเสมา ขันนาก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องกำเนิดสัญญาณพัลซ์ชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลข

Programmable Pulse Generator by Direct Digital Synthesis Method

ผู้จัดทำ

1. นายวิชัย ซื่อสัตย์ 38013026
2. นายเสมา ขันนาก 38013039

..... อาจารย์ที่ปรึกษา

(ผศ.เกรียงไกร วงศ์โรจน์กรณ์)

เครื่องกำเนิดสัญญาณพัลซ์ชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลข

Programmable Pulse Generator by Direct Digital Synthesis method

โดย

1. นายวิชัย ซื่อสัตย์ 38013026
2. นายเสมา ขันนาก 38013039

อาจารย์ที่ปรึกษา ผศ.เกรียงไกร วงศ์โรจน์ภรณ์

บทคัดย่อ

โครงงานนี้เป็นการศึกษา วงจรเครื่องกำเนิดสัญญาณที่สามารถควบคุมรูปแบบและความถี่ของสัญญาณได้จาก computer ผ่าน serial port การสร้างสัญญาณจะใช้เทคนิคการสร้างแบบ digital synthesizer มาแทนการสร้างสัญญาณแบบเดิมทาง analog โดยการใช้ computer คำณวณรูปสัญญาณจนได้รูปแบบที่ต้องการแล้ว ข้อมูลของสัญญาณจะส่งจาก computer มาเก็บไว้ใน ram รูปของสัญญาณแต่ละจุดได้จากค่า phase ของสัญญาณซึ่งมีการเพิ่มขึ้นอย่างสม่ำเสมอโดยวงจร phase accumulator ซึ่ง phase นี้ จะส่งไปเทียบค่าที่ตารางรูปสัญญาณที่ได้เก็บค่าไว้แล้ว ค่าที่ได้จาก ram จะถูกเปลี่ยนเป็นสัญญาณ analog ด้วยวงจร DAC

Abstract

This project describes the studying of programmable pulse generator which is controlled by computer via serial port by using the digital technique to synthesize the waveform instead of the analog method. The digital waveform are synthesized by using computer before sending to RAM.Each point of signal is obtained from phase accumulator,these phase will send to RAM and look up waveform table in order to obtain the function values.It will be converted to be analog signal after that it will be sent to DAC.

สารบัญ

	เรื่อง	หน้า
บทที่ 1	บทนำ	1
บทที่ 2	ทฤษฎีและหลักการทำงาน	2
	1. หลักการทำงาน	3
	2. แนวคิดในการกำหนดรูปคลื่น	6
	3. การเปลี่ยนสัญญาณดิจิตอลเป็นอนาล็อก	8
บทที่ 3	การออกแบบวงจร	15
บทที่ 4	การทดลองและผลการทดลอง	38
บทที่ 5	สรุปและวิจารณ์	46
	ภาคผนวก	47
	เอกสารอ้างอิง	

บทที่ 1
บทนำ

เครื่องกำเนิดสัญญาณที่ใช้ในการทดลองด้านอิเลคทรอนิกส์ในปัจจุบันเป็นอุปกรณ์ที่มีความจำเป็นมาก เพื่อเป็นแหล่งกำเนิดสัญญาณรูปแบบต่างๆ ป้อนให้กับวงจรทดลองต่างๆ และใช้เป็นสัญญาณอ้างอิงเทียบกับการทดลอง โดยเครื่องกำเนิดสัญญาณที่ใช้กันอยู่ทั่วไปนั้นมักมีรูปร่างของสัญญาณอยู่ 3 ชนิด คือ SINE WAVE, SQUARE WAVE และ TRIANGLE WAVE รูปสัญญาณทั้ง 3 ชนิด นี้จะถูกสร้างขึ้นด้วยวงจรเฉพาะแบบของแต่ละรูปแบบสัญญาณ ในกรณีที่เราต้องการรูปแบบสัญญาณพิเศษชนิดหนึ่ง เราต้องออกแบบวงจรไฟฟ้าเฉพาะรูปแบบของวงจรนั้น บางครั้งอาจทำได้ยากลำบาก หรือบางครั้งอาจทำไม่ได้เลย ในที่นี้ได้เสนอแนวทางการสร้างรูปสัญญาณไฟฟ้าชนิดต่างๆ ที่เราสามารถกำหนดโปรแกรมรูปสัญญาณได้จากคอมพิวเตอร์ ...

การกำหนดรูปแบบของสัญญาณทำได้โดยอาศัยเทคนิคการสร้างสัญญาณแบบ Digital Synthesizer มาใช้แทนเทคนิคการสร้างสัญญาณแบบ Analog การสร้างสัญญาณประเภทนี้มีความยืดหยุ่นมาก คือสามารถใช้เครื่องคอมพิวเตอร์มาคำนวณสร้างรูปแบบของสัญญาณจนได้รูปร่างตามที่ต้องการแล้ว จึงนำข้อมูลของสัญญาณนั้นมาสร้าง เพื่อต้องการเปลี่ยนสัญญาณเป็นรูปอื่น ก็สามารถคำนวณหาค่าของสัญญาณนั้นใหม่ได้แล้วจึงนำมาสร้าง โดยไม่ต้องเปลี่ยนแปลงวงจรทางด้านฮาร์ดแวร์ใหม่ ความสามารถของโครงงานนี้สามารถกำเนิดสัญญาณรูปคลื่นต่าง ๆ ได้ทุกรูปแบบโดยรูปแบบของสัญญาณนั้นเราสามารถกำหนดรูปแบบได้จากคอมพิวเตอร์ สามารถที่จะเลือกได้ทั้งรูปคลื่น ความถี่ และขนาดของสัญญาณโดยทำการเลือกผ่านทางจอคอมพิวเตอร์ ซึ่งจะมีความถูกต้องค่อนข้างสูงและสามารถทำงานได้อย่างรวดเร็ว โดยเราจะทำการคำนวณรูปแบบของสัญญาณที่ต้องการแล้ว รวมทั้งความถี่และค่าของขนาดสัญญาณต่างๆ ก็จะถูกส่งผ่านพอร์ทอนุกรมต่อไปยังส่วนควบคุมของไมโครคอนโทรลเลอร์เลอร์ 8031 ทำการควบคุมการทำงานทั้งหมดอีกครั้ง โดยเครื่องกำเนิดสัญญาณชนิดที่สามารถควบคุมความถี่ และขนาดของสัญญาณที่ได้จะนำมาสร้างเป็นโครงงานนี้

สามารถทำการกำเนิดสัญญาณได้ทั้งหมดจำนวน 2 ช่องสัญญาณพร้อมๆ กัน โดยที่เราสามารถเลือกว่าจะกำเนิดสัญญาณรูปแบบใดก็ได้จำนวน 2 สัญญาณพร้อมๆ กัน โดยสัญญาณที่เราทำการกำเนิดขึ้นมานั้นสามารถที่จะทำการปรับขนาดของสัญญาณแยกกันระหว่างช่องสัญญาณที่ 1 และ 2 ของสัญญาณได้ และสามารถเพิ่มรูปแบบของสัญญาณจากการเพิ่มในส่วนของซอร์ฟแวร์ ขึ้นมาโดยเราสามารถทำการเขียนโปรแกรมการทำงานเพิ่มเข้าไป

บทที่ 2
ทฤษฎีและหลักการทำงาน

การทำงานของเครื่องกำเนิดความถี่ชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลขนี้แบ่งการทำงานออกเป็น 2 ส่วน คือ

1. ฮาร์ดแวร์
2. ซอร์ฟแวร์ (โปรแกรม)

ส่วนของฮาร์ดแวร์ประกอบด้วย

1.1 วงจรไมโครคอนโทรลเลอร์เบอร์ 8031
1.2 วงจรถอดรหัสตำแหน่ง (Decode Address)
1.3 วงจร Phase Step Size
1.4 วงจร Phase Accumulator
1.5 จงจร RAM Address Gen
1.6 วงจร RAM Data Gen
1.7 วงจร มัลติเพล็ก (Multiplex)
1.8 วงจรหน่วยความจำ
1.9 วงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อก (DAC)
1.10 วงจรสัญญาณนาฬิกาอ้างอิง (Clock Generator)
1.11 วงจรขยายปรับค่าได้

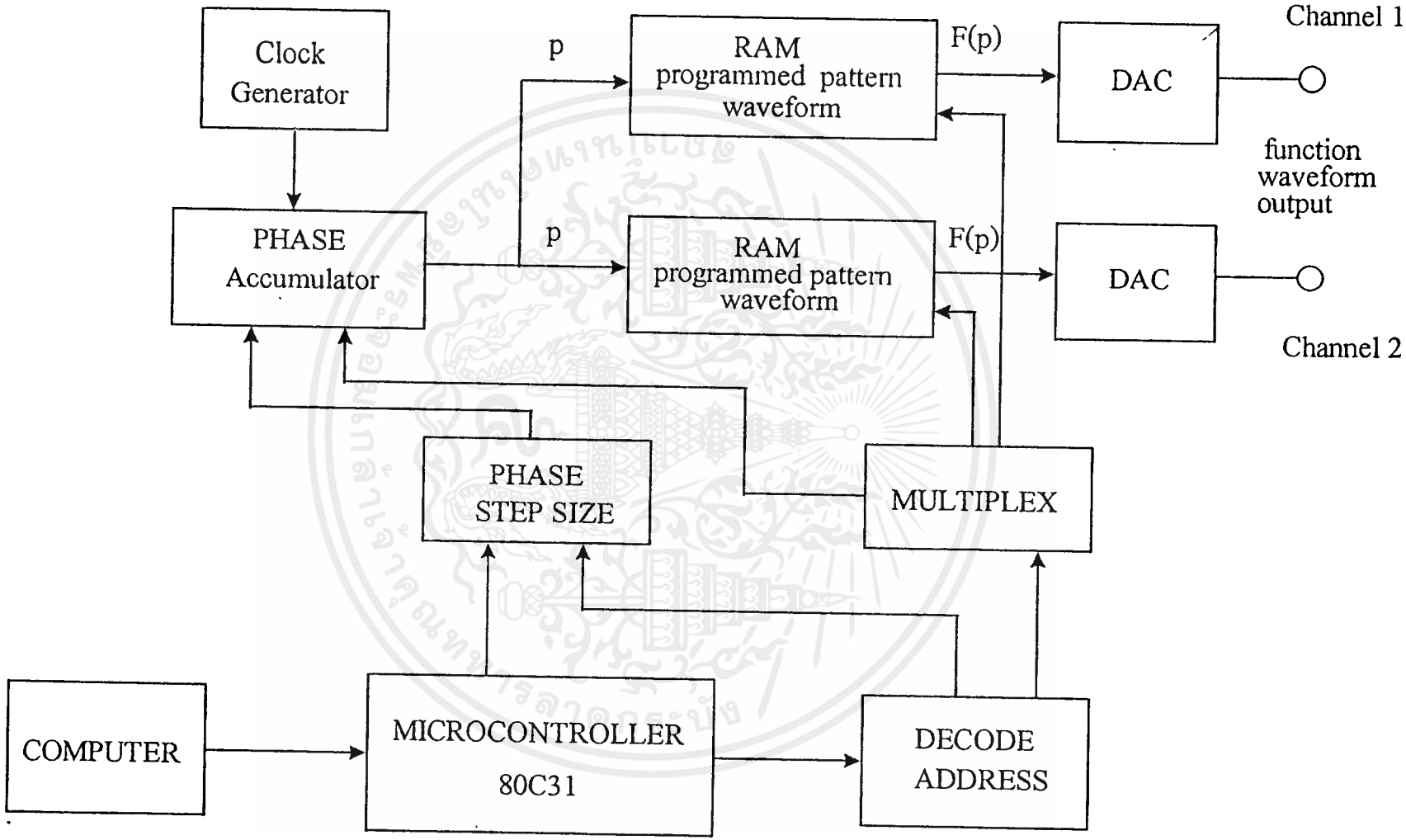
ส่วนประกอบของซอร์ฟแวร์ประกอบด้วย

2.1 ส่วนเลือกรูปแบบของสัญญาณ
2.2 ส่วนปรับความถี่
2.3 ส่วนปรับขนาดของสัญญาณ

รูปของสัญญาณของวงจรกำเนิดความถี่ชนิดนี้ สามารถให้ค่าของ Frequency Bandwidth อยู่ที่ประมาณ 500 KHz ค่าของขนาดสูงสุดของสัญญาณประมาณ 0 - 4 Vp-p

2.1 หลักการทำงาน

วงจรเครื่องกำเนิดสัญญาณชนิดที่โปรแกรมได้ โดยการสังเคราะห์โดยตรงทางเชิงเลขนี้สามารถควบคุมแอมพลิจูด และขนาดของสัญญาณได้นี้ โครงงานนี้จะมีทั้งหมด 2 ช่องสัญญาณ โครงสร้างหลักๆ ของวงจรแสดงดังรูปที่ 1



รูปที่ 1 BLOCK DIAGRAM เครื่องกำเนิดสัญญาณที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลข

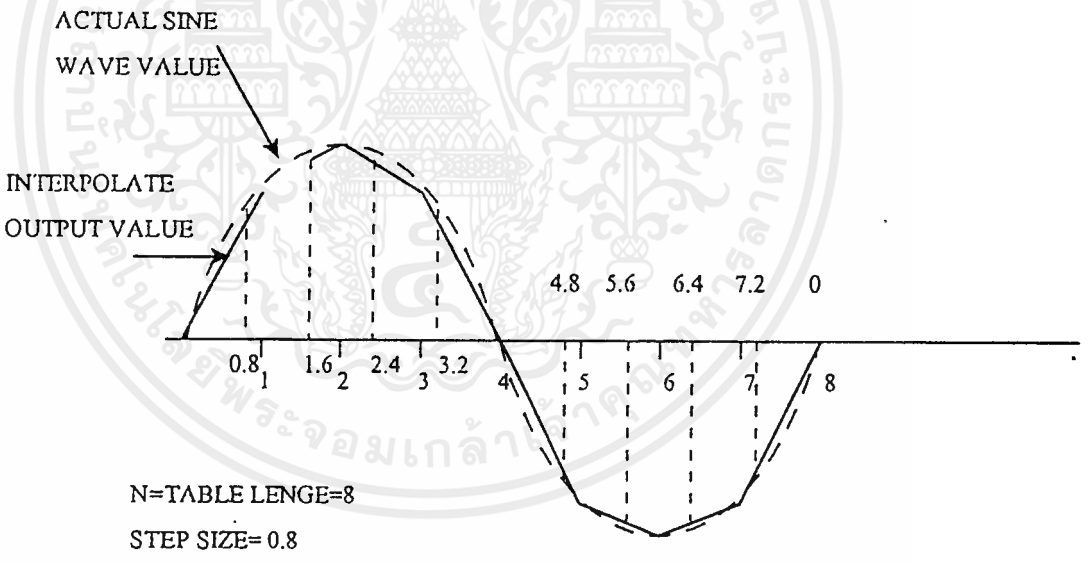
จากบล็อกไดอะแกรม เป็นโครงสร้างหลักๆ ของเครื่องกำเนิดความถี่ชนิดนี้จากรูปจะใช้ไมโครคอนโทรลเลอร์ เบอร์ 8031 เป็นตัวควบคุมการทำงานทั้งหมดของวงจร การทำงานของเครื่องกำเนิดความถี่ชนิดนี้ จะเริ่มจากการทำงานของคอมพิวเตอร์ โดยคอมพิวเตอร์จะทำการคำนวณสร้างรูปแบบของสัญญาณที่ต้องการสร้าง โดยการคำนวณของคอมพิวเตอร์นั้นเกิดขึ้นจากโปรแกรม โดยการทำการแซมปลิ้งสัญญาณ 1 คาบเวลาออกเป็น 8192 จุด เท่ากับค่าตำแหน่งของหน่วยความจำของแรมภายนอกที่ใช้งานคือ เบอร์ 6264 เป็นขนาด 8 Kbyte ซึ่งมีตำแหน่งของหน่วยความจำเท่ากับ 8192 ตำแหน่ง เท่ากับจุดของสัญญาณที่เราทำการแซมปลิ้ง 1 คาบเวลา เมื่อคอมพิวเตอร์คำนวณรูปแบบของสัญญาณได้แล้วก็จะทำการส่งข้อมูลของสัญญาณแต่ละจุด พร้อมกับทั้งค่าความถี่ของรูปคลื่นที่เรากำหนดไว้ในแรมภายนอก คือ RAM1 และ RAM2 และค่าของข้อมูลที่เป็นตัวกำหนดความถี่ของรูปคลื่นกับข้อมูลที่เป็นตัวกำหนดค่าแอมพลิจูดของรูปคลื่น จะเก็บไว้ในตำแหน่งของ แรมภายใน เมื่อเราได้ข้อมูลของรูปคลื่นสัญญาณ 1 คาบเวลา เก็บไว้ในแรมเรียบร้อยแล้ว ในขั้นตอนต่อไปของการสร้างสัญญาณออกมานั้น วงจรกำเนิดสัญญาณนาฬิกาอ้างอิง จะสร้างสัญญาณนาฬิกาอ้างอิง fc ออกมา สัญญาณนาฬิกาอ้างอิงนี้จะป้อนให้กับวงจร Phase Accumulator วงจรนี้จะทำการบวกวนซ้ำด้วยค่าที่ตั้งไว้โดยวงจร Phase Step Size ค่าที่ได้ทำการตั้งไว้โดยวงจร Phase Step Size นี้ ได้รับมาจากข้อมูลที่เป็นตัวกำหนดความถี่ของรูปคลื่นที่เราทำการเก็บตำแหน่งของ Ram ภายใน ผลของการบวกซ้ำจะทำให้เกิดค่าของเฟสสะสม P ค่าเฟสสะสม P นี้ มีลักษณะเพิ่มขึ้นด้วยจำนวนเท่ากันทุกครั้งที่ทำการบวก ตามจังหวะของสัญญาณนาฬิกาอ้างอิง fc ค่าของเฟสสะสม P นี้ จะนำไปชี้ตารางของ Function Waveform ที่ได้จัดเตรียมไว้ล่วงหน้าและจัดเก็บไว้ในแรมแล้ว เมื่อได้เทียบ ชี้ค่าได้เป็นค่าของ Function Waveform Value , F(p) ออกมา ก็จะนำค่า F(p) นี้ซึ่งเป็นค่าทางดิจิตอลมาทำการเปลี่ยนเป็นค่าอนาล็อกโวลท์เตจด้วยวงจร DAC ก็จะได้ค่าของเอาท์พุทโวลท์เตจเป็นสัญญาณรูปแบบตามต้องการ

จากการทำงานของวงจรถ้าเราทำการเปลี่ยนค่าของ Phase Step Size ก็เท่ากับเป็นการเปลี่ยนค่าของAccess time ของหน่วยความจำ เพราะค่าของ Phase Step Size นี้จะเป็นตัวกำหนดค่าเริ่มต้นให้กับวงจรนับวนซ้ำคือวงจร Phase Accumulator ถ้า Phase Step Size มีค่าสูง ทำให้การบวกเพิ่มขึ้นของ Phase Accumulator มีค่า เพิ่มมากขึ้นจึงทำให้ค่าความถี่ที่เป็นได้ออกมานั้นมีค่ามากขึ้น เพราะช่วงเวลาที่จะไปใช้ในการอ่านข้อมูลออกจากหน่วยความจำเร็วขึ้น จึงทำให้ค่าของความถี่ที่ได้นั้นมีค่าสูงขึ้นตามไปด้วย

จากรูปวงจรจะเห็นได้ว่ามีจำนวนของช่องสัญญาณทั้งหมด 2 ช่องสัญญาณเพราะมีส่วนของ RAM1 และ RAM2 และส่วนของวงจรที่ทำหน้าที่แปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก 2 ชุด ดังนั้นเราจึงสามารถกำเนิดรูปคลื่นได้ 2 รูปคลื่นพร้อมกัน หรือสามารถกำเนิดรูปคลื่นจำนวน 2 สัญญาณพร้อมกันก็ได้ในกรณีที่เป็นคนละสัญญาณกัน โดยมีวงจร Phase Accumulator และวงจร Phase Step Size เดียวกัน ทำให้ทั้งสองช่องสัญญาณมีความถี่เดียวกัน

2.2 แนวคิดในการกำเนิดรูปคลื่น

ไดเร็กดิจิตอลซินธิไซเซอร์ (Direct Digital Synthesizer) DDS คือการกำเนิดรูปคลื่นโดยอาศัยวิธีการทางดิจิตอล ซึ่งในเครื่องกำเนิดความถี่แบบทั่วๆ ไปนั้นจะใช้วิธีการทางอนาล็อกเฟสล็อกลูป หรือการกำเนิดรูปคลื่นโดยใช้คริสตอล ในการเก็บข้อมูลของสัญญาณที่เราจะทำการกำเนิดขึ้นมานั้น จะต้องทำการเก็บข้อมูลของสัญญาณให้ครบคาบของสัญญาณ ซึ่งถ้าหากว่าเราเก็บข้อมูลจำนวนมากก็จะทำให้สัญญาณที่จะกำเนิดขึ้นมานั้นมีลักษณะใกล้เคียงกับสัญญาณจริงมาก แต่การเก็บข้อมูลของสัญญาณมากๆ ก็มีข้อเสียคือ จะต้องใช้สัญญาณนาฬิกาจำนวนมากในการกำเนิดสัญญาณ แต่ในโครงงานนี้จึงเลือกการเก็บข้อมูล 8192 จุดต่อสัญญาณ 1 คาบเวลา ดังนั้น สัญญาณ 1 สัญญาณ จะใช้เนื้อที่หน่วยความจำในการเก็บข้อมูล 8192 ไบท์ หรือ 8 Kbyte หรือเท่ากับจำนวนความจุของหน่วยความจำภายนอกที่เราใช้งาน วิธีการเก็บข้อมูลของสัญญาณ 1 คาบเวลาดังรูป



รูปที่ 2 แสดงการเก็บข้อมูลของสัญญาณ 1 คาบเวลา

จากรูปที่ 2 เป็นตัวอย่างการเก็บข้อมูลของสัญญาณซายน์ 1 คาบเวลา ซึ่งการเก็บข้อมูลจะต้องใช้การสุ่ม (Sampling) ข้อมูลบนสัญญาณแต่ละจุดด้วยเวลาที่เท่ากันทุกจุด ดังนั้นจุดบนสัญญาณที่ต้องการเก็บคือ จุดที่

360/จำนวนข้อมูล เช่น ต้องการเก็บข้อมูลของสัญญาณซายน์จำนวน 8192 ค่า ดังนั้นจะต้องทำการเก็บข้อมูลทุก 360/8192 องศา สามารถเขียนเป็นตารางได้ดังนี้

ข้อมูลที่	มุมที่เก็บข้อมูล	ค่าของข้อมูลที่ได้
0	0 * 360/N	S(0) = SIN (0/N)
1	1 * 360/N	S(1) = SIN (360/N)
2	2 * 360/N	S(2) = SIN (720/N)
●		
●		
N-2	(N-2) * 360/N	S(N-2) = SIN (CN-2) * 360/N)
N-1	(N-1) * 360/N	S(N-1) = SIN (CN-2) * 360/N)

หลังจากที่เราได้ข้อมูลมาแต่ละจุดแล้ว จะต้องมาจำการจัดค่า (Quantization) ให้มีค่าเป็นทางดิจิตอล โดยการจัดค่าจะมีได้ทั้งหมด 256 ค่าคือค่า 00H-FFH เมื่อเราได้ข้อมูลที่มีค่าเป็นเลขฐานสองมาแล้ว ก็จะนำค่าข้อมูลเหล่านั้นไปเก็บไว้ในหน่วยความจำโดยการจัดเรียงกันไป คือค่าของข้อมูลจุดแรกบนสัญญาณจะถูกเก็บที่แอดเดรสแรก ค่าของข้อมูลตัวที่สองก็จะถูกเก็บในแอดเดรสถัดไปจนครบหมดทุกตัว

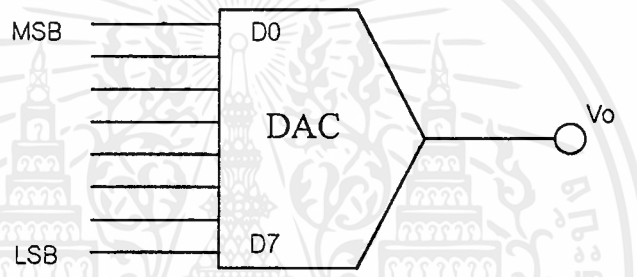
นอกจากสัญญาณซายน์ แล้วเราสามารถสร้างสัญญาณชนิดอื่นได้อีกมาก โดยการใช้โปรแกรมคำนวณหาค่าของสัญญาณต่างๆ โดยการคำนวณหาค่าของสัญญาณแต่ละจุดโดยการใช้หลักการเดียวกับการกำเนิดรูปคลื่นซายน์ ดังที่ได้กล่าวมาเบื้องต้น

2.3 การเปลี่ยนสัญญาณดิจิตอลเป็นอนาล็อค (DAC)

หลักการทำงานของ DAC คือการนำเอากลุ่มของบิท (BIT) จากคอมพิวเตอร์หรืออุปกรณ์ดิจิตอลเปลี่ยนแปลงเป็นระดับแรงดันอนาล็อก เอาท์พุทของ DAC เป็นระดับความแตกต่างของแต่ละบิทของดิจิตอลอินพุท

หลักการพื้นฐานของ DAC

บล็อกไดอะแกรมของ DAC แสดงในรูป เอาท์พุทที่สร้างขึ้นจาก DAC เป็นได้ทั้งแรงดันและกระแส



รูปที่ 3.1 บล็อกไดอะแกรมของ D /A Converter

เอาท์พุทชนิดใดก็ตามของ DAC ที่ผลิตขึ้นมาได้จากวงจรที่นำมาใช้ในการเปลี่ยนดิจิตอลเป็นอนาล็อก จำนวนของความแตกต่างของระดับแรงดันและกระแสที่สร้างขึ้นที่เอาท์พุทของ DAC จะสัมพันธ์กับจำนวนของบิทที่นำมาเปลี่ยนจากสมการ

$$N=2^n$$

เมื่อ N คือ จำนวนของระดับความแตกต่างด้านเอาท์พุทที่สร้างขึ้น และ n คือจำนวนของบิทอินพุทที่นำมาเปลี่ยน

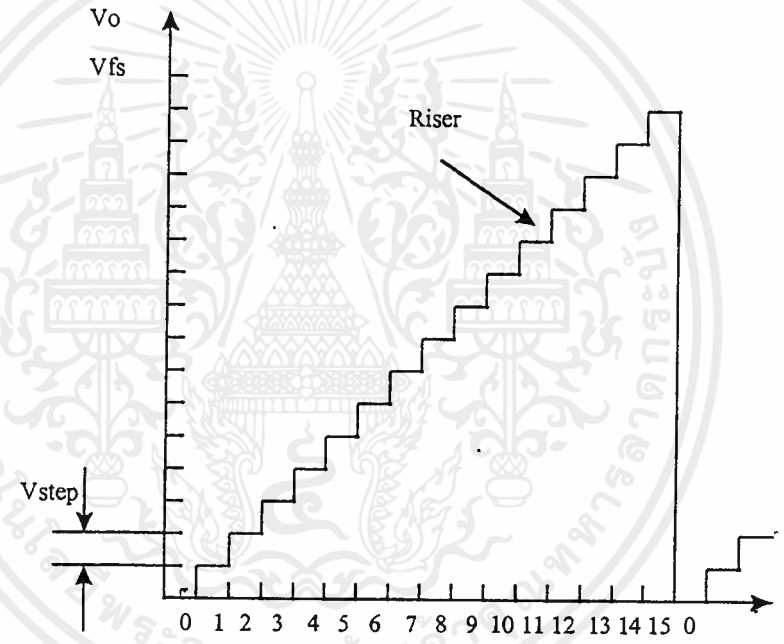
จำนวนของระดับความแตกต่างที่ สร้างขึ้นที่เอาท์พุทของ DAC จะขึ้นอยู่กับขอบเขตการจำแนกของอุปกรณ์ที่ใช้งาน จำนวนบิทของอินท์พุทจะใช้บิทที่สูงที่สุดในการคำนวณ เช่น อินพุทของ DAC จำนวน 10 บิท สามารถเปลี่ยนระดับสัญญาณได้ 1024 ระดับการเปลี่ยนแปลงเป็นรูปอื่นเป็นคุณสมบัติหนึ่งที่สำคัญของ DAC ในการนำไปประยุกต์ใช้งานในหลาย ๆ ด้าน หลักการหนึ่งในการเปลี่ยนแปลงสัญญาณดิจิตอลในรูปของ N ($N=2^n$) และสามารถคิดเป็นในรูปเปอร์เซนต์ได้จากสมการ

$$\text{Percent Resolusion} = (1/2^{n}) \times 100\%$$

เช่น 10 บิท DAC

$$\text{Percent Resolusion} = (1/2^{10}) \times 100\%$$
$$= (1/1024) \times 100\%$$
$$= 0.098\%$$

จากตัวอย่างของเอาท์พุทของ 10 บิท DAC มีความแน่นอน 0.098% ของเอาท์พุท Full Scale ซึ่งคือระดับแรงดันหรือกระแสที่สร้างขึ้นที่เอาท์พุทของ DAC ที่สมมติขึ้นว่าเลข 1 ไบนารี่ที่เป็นอินพุทแต่ละตัวเปลี่ยนแปลงเป็นรูปอื่นไม่ได้จำกัด แต่ในความเป็นจริง DAC ไม่สามารถมีจำนวนถึง Ideal Full Scale เนื่องจากการจำกัดจำนวนของอินพุทตัวอย่างเช่น DAC ที่แสดงในรูป 3.2 มีอินพุท 4 เส้น กราฟของ Vo และอินพุทไบนารี่สำหรับ 4 บิท DAC สามารถสร้างได้ดังแสดงในรูป 3.2



รูป 3.2 คุณสมบัติทรานเฟอร์สำหรับ 4 บิท DAC

จะสังเกตได้ว่ามีระดับความแตกต่างของแรงดันที่เป็นไปได้ 16 ระดับและ 15 ของขาขึ้น ถ้าเป็นเอาท์พุท Full Scale จะมีขอบขาขึ้น 16 ขอบ ซึ่งหมายถึงว่าค่า Vo สูงสุดเอาท์พุทจะไม่ถึง Vfs อีกหนึ่งขั้นขนาดของเอาท์พุทหนึ่งขั้นเรียกว่า 1LSB ของดิจิตอลอินพุทเปลี่ยนสภาวะ การเพิ่มขึ้นของเอาท์พุท (แรงดันหรือกระแส) สำหรับแต่ละขั้นหาได้จากจำนวนของขั้นและ Vfs ซึ่งมีความสัมพันธ์กับดังนี้

$$\text{ขนาดขั้น} = Vfs/2^{n}$$

เมื่อ n คือจำนวนของอินพุทไบนารี่และ Vfs คือแรงดัน Full Scale ของ Ideal DAC เช่น Vo ของ 4 บิท DAC เป็นไปตามทฤษฎีมี Vfs เท่ากับ 10 V และอินพุทไบนารี่ 12 ฐาน 10 ได้ Vo เท่ากับ

$$\text{ขนาดขั้น} = Vfs/2^n$$

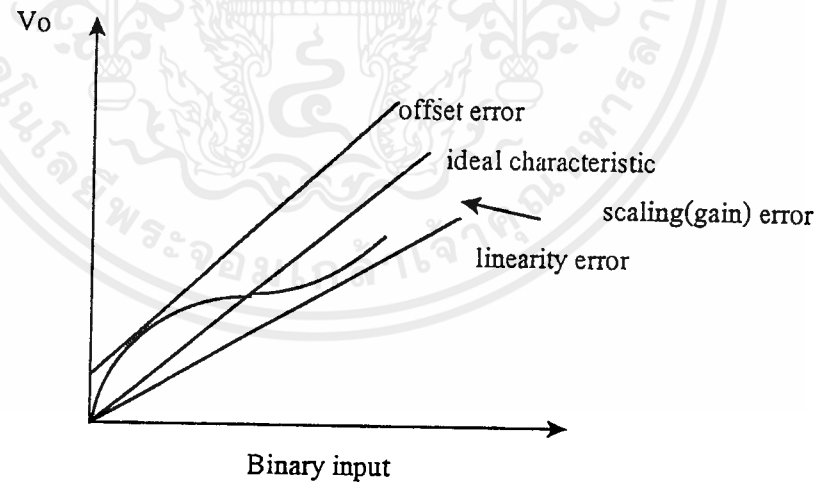
$$= 10V/16$$

$$= 0.625V$$

$$Vo = 0.625V \times 12$$

$$= 7.5\ V$$

การจำแนกของ DAC จะใช้เป็นตัวบอกความเที่ยงตรงของศักย์เพราะว่าการจำแนกเป็นตัวกำหนดข้อจำกัดของความเที่ยงตรงของการเปลี่ยนแปลง อย่างไรก็ตามความเที่ยงตรงและการจำแนกไม่ใช่สิ่งเดียวกัน ตัวอย่างเช่น 16 บิท DAC จะพิจารณาถึงการจำแนกสูงสุด (65536) แต่ไม่ใช่สิ่งจำเป็นที่ถูกต้องในการหาค่า Vo ซึ่งจะหาได้จากค่าอินพุทที่ให้มา ภายใต้เงื่อนไขอุดมคติเอาท์พุทของ DAC จะมีความถูกต้อง +1/2 Vstep (หรือ +1/2 LSB เพราะ 1 STEP = 1 LSB) อย่างไรก็ตามอาจมีความผิดพลาดได้ใน DAC แต่ละชนิดความคลาดเคลื่อนบนเอาท์พุทบนตัวคอนเวอร์เตอร์ แสดงดังรูปที่ 3.3 เป็นรูปผลของการเปลี่ยนแปลงความคลาดเคลื่อนของทรานเฟอร์ฟังก์ชั่นของ DAC อุดมคติ



รูปที่ 3.3 กราฟของ DAC อุดมคติและผลของความคลาดเคลื่อน

OFFSET ERROR เป็นผลที่เกิดขึ้นที่เอาท์พุทของ DAC ไม่เป็น 0 เมื่ออินพุทไบนารี่เป็น 0 ทำให้เกิดค่าคงที่เลื่อนให้ Vo ให้เกิดย่านของไบนารี่อินพุท

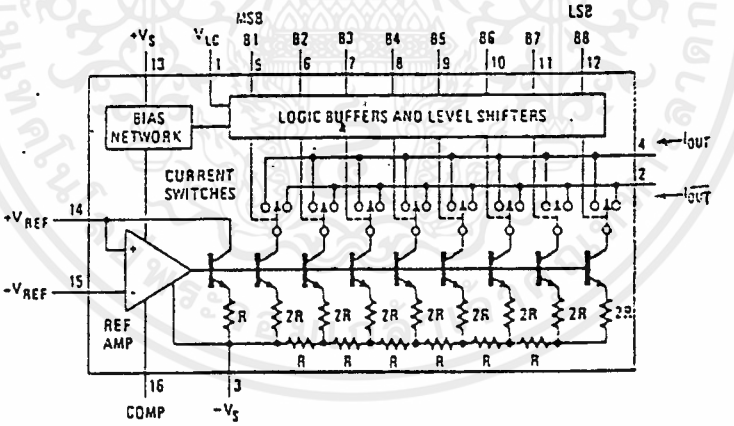
GAIN ERROR หรือเรียกอีกอย่างหนึ่งว่า Scaling Error จะสร้างขนาดขึ้นให้ใหญ่กว่า หรือเล็กกว่าขนาดปกติซึ่งเป็นสาเหตุให้ค่า Vo เบี่ยงเบนจากค่าความเป็นจริงของไบนารี่อินพุท

LINEARITY ERROR เป็นความคลาดเคลื่อนอีกชนิดหนึ่งที่เป็นสาเหตุทำให้ DAC ไม่เป็นเชิงเส้น ตัวอย่างเช่น ถ้าเกณฑ์ของ DAC ไม่คงที่สำหรับไบนารี่เอาท์พุทจะเปลี่ยนแปลงขนาดของขั้นที่สร้างขึ้น

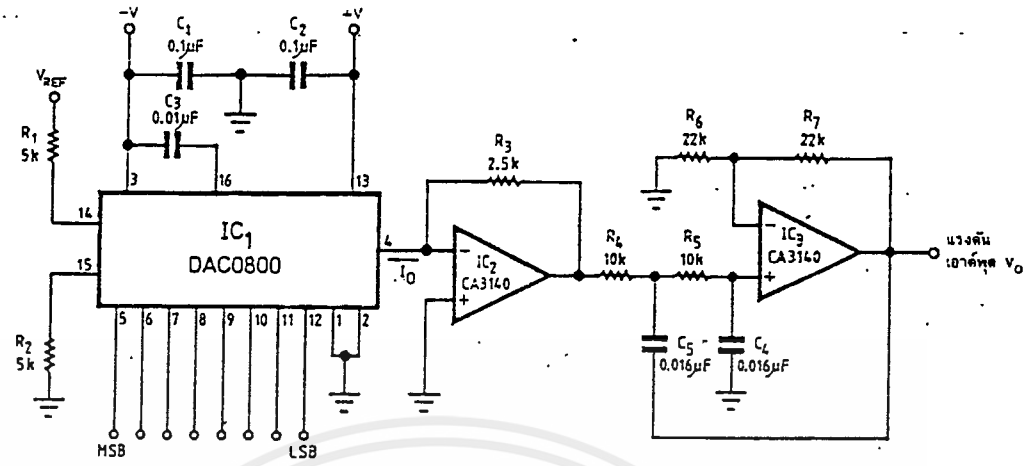
คุณลักษณะของ DAC ที่สำคัญอีกอย่างคือความสัมพันธ์เกี่ยวกับเวลาที่ใช้ในการเปลี่ยนแปลง คุณสมบัตินี้เรียกว่า Setting Time' เป็นการวัดการตอบสนองทางด้านความเร็วของ DAC

ไอซี DAC

ไอซี DAC มีหลายแบบและแตกต่างกัน ตัวอย่างหนึ่งของไอซี DAC ที่ผลิตโดยเนชั่นแนลเซมิคอนดัคเตอร์ คือ DAC 0800 เป็น DAC ชนิด 8 บิท DAC ให้เอาท์พุท เป็นกระแสซึ่งจะมีความสัมพันธ์กับค่าของไบนารี่อินพุทที่ป้อนเข้ามา DAC 0800 มีทั้งหมด 16 ขา เป็นแบบ DIP ดังแสดงในรูป 3.4



รูป 3.4 ไอซี DAC 0800



รูป 3.5 การใช้งานวงจร DAC 0800

วงจร DAC ที่ใช้ไอซี DAC 0800 แสดงดังรูป และกระแสเอาท์พุทของวงจรหาได้จาก

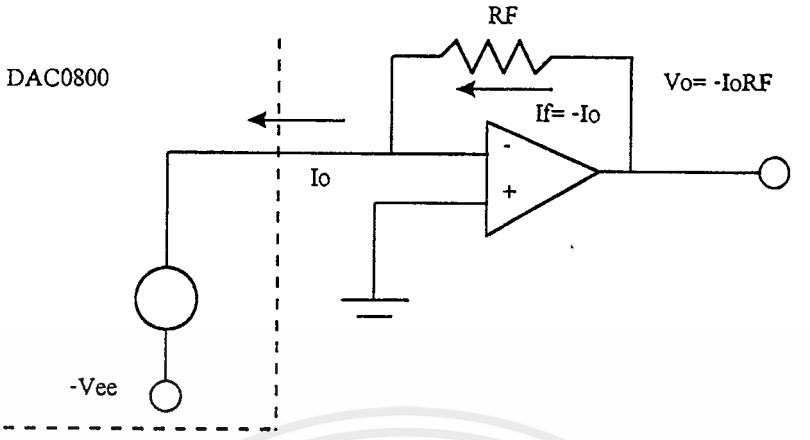
$$Io = -\frac{V_{ref}}{R_{in}} (D_7/2 + D_6/4 + D_5/8 + \ldots\ldots\ldots + D_0/256)$$

เมื่อ Dn = 0 หรือ 1

อินพุท D จะเปลี่ยนแปลงไปมาระหว่าง 0 กับ 1 ซึ่งอินพุทของ DAC สามารถต่อเข้าดาต้าบัสของ CPU ได้ ขาแรงดันอ้างอิงลบ Vref (-) จะต่อลงกราวด์ผ่านความต้านทาน R2 ซึ่งมีค่าเท่ากับ R1 จะช่วยป้องกัน offset Error ขา 16 จะต่อไฟลบ -VEE โดยมีตัวเก็บประจุขั้นไว้ (มีค่าประมาณ 0.001μF) ซึ่งจะช่วยป้องกัน Ringinng และ Over shoot ที่เอาท์พุทของ DAC เอาท์พุท ของ DAC เอาท์พุทของ DAC 0800 จะเป็นระดับกระแสลบ

เราสามารถผ่านเอาท์พุทของ DAC ที่เป็นกระแสให้เป็นแรงดันได้ โดยใช้ออปแอมป์เปลี่ยนกระแสเป็นแรงดัน (I/V) โดยการต่อ IO ของ DAC 0800 ดังรูป 3.6 แสดงการเปลี่ยนกระแสเป็นแรงดัน แรงดันเอาท์พุทของวงจรหาได้จากสมการ

$$VO = -IoR_f$$

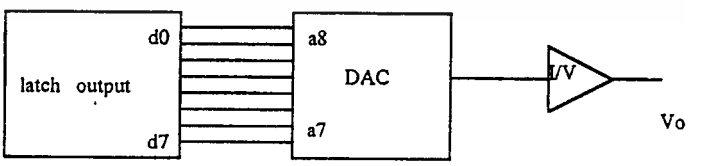


รูป 3.6 ใช้ออปแอมป์เปลี่ยนกระแสเป็นแรงดันจากเอาท์พุทของ DAC 0800

จากรูปเอาท์พุทของ DAC จะเป็นกระแสลบเมื่อผ่านออปแอมป์จะได้แรงดันบวก ซึ่งจะสัมพันธ์กับไบนารี่อินพุท วงจรสมมูลย์ของ DAC 0800 และ I/V คอนเวอร์เตอร์ แสดงดังรูป 3.5 เอาท์พุท VO สามารถหาได้จากสมการนี้

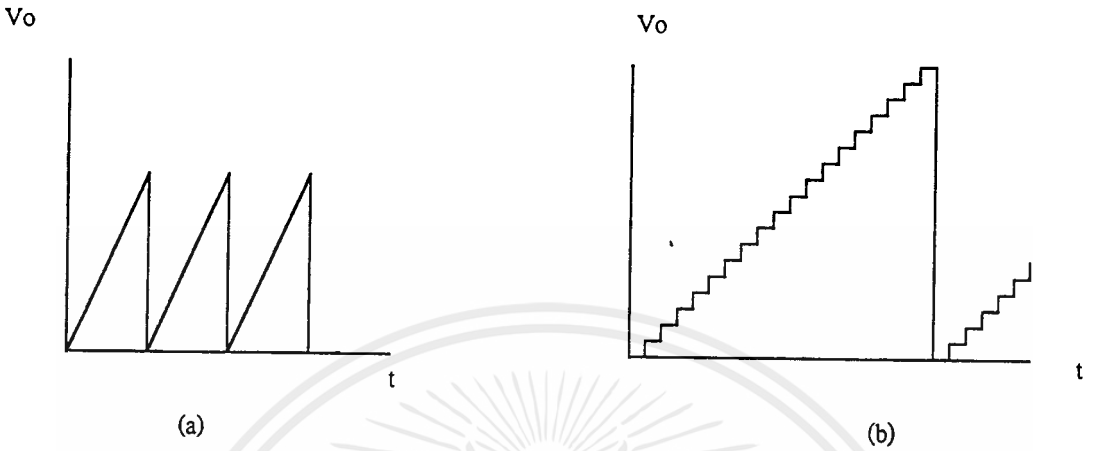
$$VO = \frac{Vref\ RF}{R1} (D7/2 + D6/4 + D5/8 \ldots\ldots + D0/256)$$

สามารถนำเอา DAC 0800 มาประยุกต์ใช้งานได้โดยใช้คอมพิวเตอร์ในการทำเนิดรูปคลื่นสัญญาณ ดังแสดงในรูป 3.7 ซึ่งเป็นบล็อกไดอะแกรมของวงจรที่ใช้กำเนิดรูปคลื่น



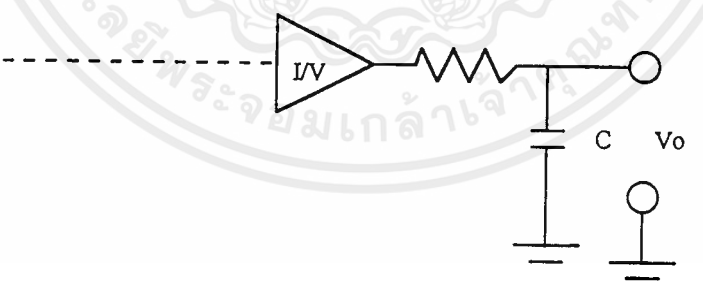
รูป 3.7 การต่อ DAC เข้ากับพอร์ทเอาท์พุท

เช่น ป้อนอินพุทให้ DAC ค่า 00H-FFH จะได้รูปคลื่นดังรูป 3.8 (b)



รูป 3.8 (a) รูปคลื่นฟันเลื่อยอุดมคติ
(b) รูปคลื่นที่ผลิตจาก DAC

และทำให้รูปคลื่นที่ผลิตได้มีลักษณะใกล้เคียงรูปคลื่นในอุดมคติได้โดยการฟิลเตอร์



รูป 3.9 เอาท์พุทของ DAC ผ่านฟิลเตอร์

บทที่ 3

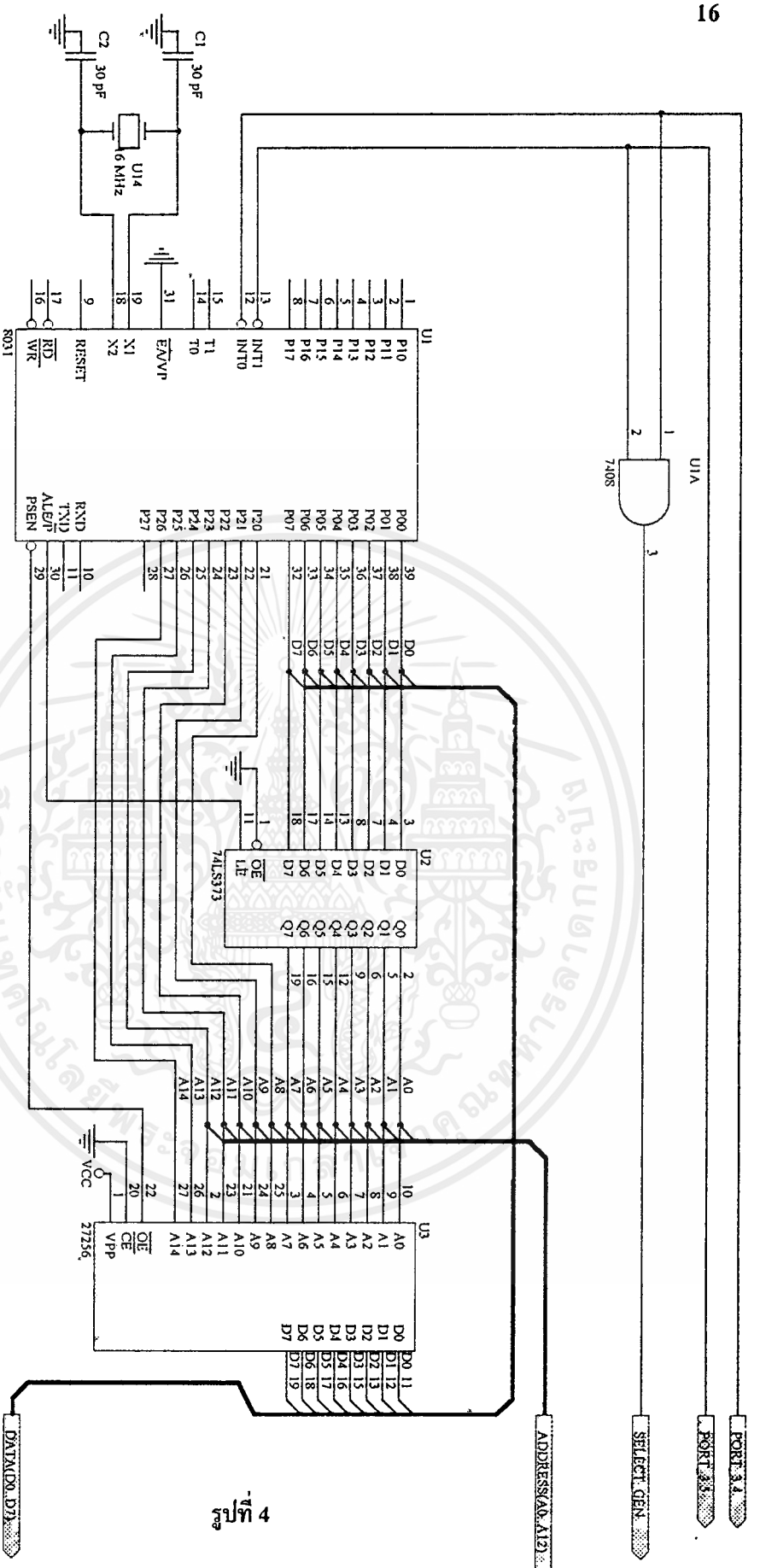
การออกแบบวงจร

จากบล็อกไดอะแกรม สามารถที่จะแยกอธิบายการทำงานได้เป็นส่วนๆ ดังต่อไปนี้

1. วงจรไมโครคอนโทรลเลอร์ เบอร์ 8031

ทำหน้าที่ควบคุมการทำงานทุกๆ อย่างของฮาร์แวร์วงจรกำเนิดความถี่นี้ ซึ่งจะถูกควบคุมการทำงานด้วยโปรแกรมที่ป้อนให้ไมโครคอนโทรลเลอร์ เบอร์ 8031 โครงสร้างของไมโครคอนโทรลเลอร์แสดงดังรูป และติดต่อกับหน่วยความจำข้อมูลภายนอกและการติดต่อกับหน่วยความจำสำหรับโปรแกรมภายนอกจะไม่กล่าวในที่นี้ เพราะว่าได้กล่าวในทฤษฎีเบื้องต้นแล้ว การทำงานของส่วนไมโครคอนโทรเลอร์ เบอร์ 8031 นี้จะทำการรับข้อมูลอนุกรมกับคอมพิวเตอร์ ซึ่งข้อมูลที่ใช้ส่งมาจากเครื่องคอมพิวเตอร์ทั่วไป จะเป็นค่าของรูปสัญญาณ 1 คาบเวลา ที่ได้คำนวณโดยโปรแกรม และค่าของข้อมูลความถี่ ที่จะมาป้อนให้กับส่วนของวงจร phase step size จากรูปจะเห็นได้ว่าส่วนประกอบทางฮาร์ดแวร์นั้นประกอบด้วย

หน่วยความจำสำหรับโปรแกรมภายนอก คือ EPROM 27256 ซึ่งจะทำหน้าที่เก็บโปรแกรมควบคุมการทำงานทุกอย่างของฮาร์ดแวร์ เช่น โปรแกรมที่ทำการติดต่อกันทางพอร์ทอนุกรมกับเครื่องคอมพิวเตอร์ทั่วๆ ไป การทำงานเมื่อคอมพิวเตอร์ส่งค่าของสัญญาณ 1 รูปคลื่นมาให้แล้ว ไมโครคอนโทรลเลอร์ ก็จะทำการเก็บค่าของรูปคลื่นไว้ในหน่วยความจำสำหรับข้อมูลภายนอก คือ RAM1 หรือ RAM2 พร้อมกับข้อมูลที่จะไปป้อนให้กับส่วนของ Phase Step Size ไว้ในหน่วยความจำสำหรับข้อมูลภายในไมโครคอนโทรเลอร์เบอร์ 8031

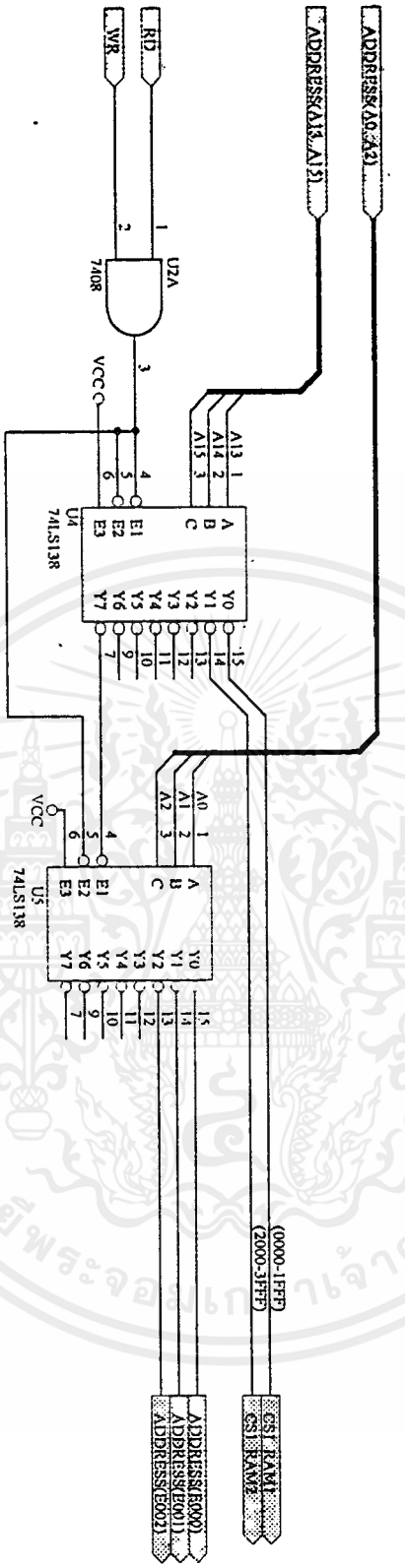


รูปที่ 4

โครงสร้างทางฮาร์แวร์ของไมโครคอนโทรลเลอร์ เบอร์ 8031

2. วงจรถอดรหัสตำแหน่ง (Decode Address)

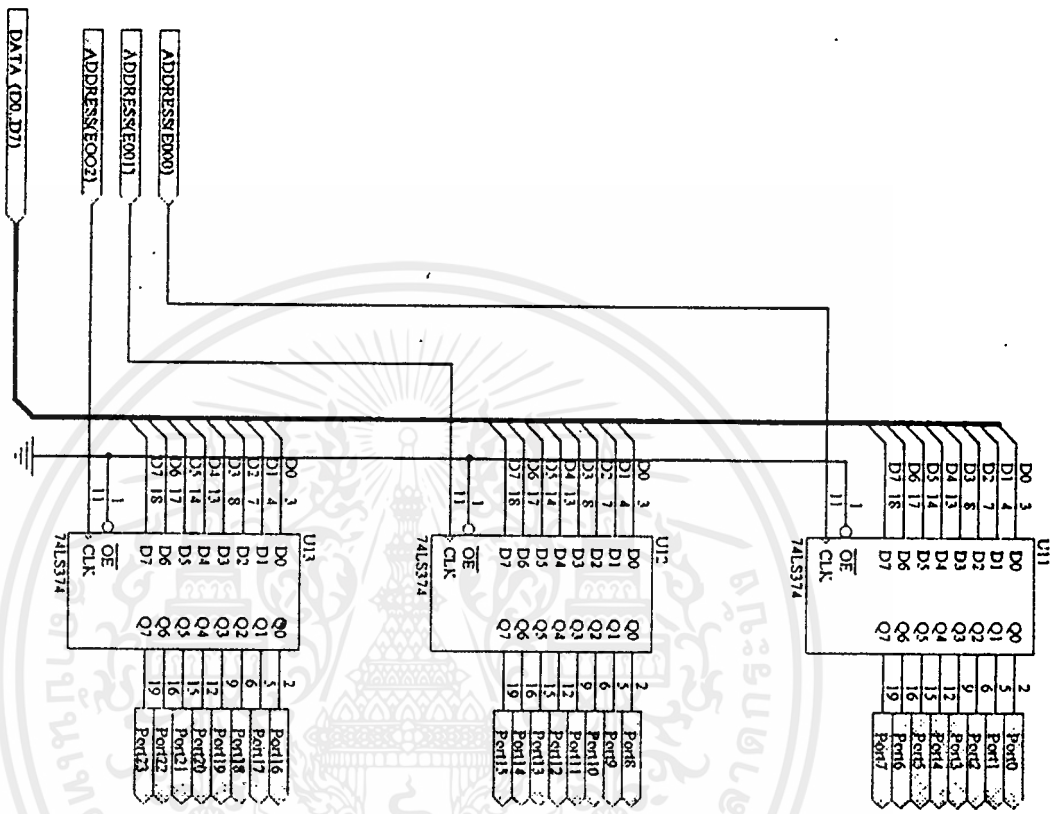
ส่วนของวงจรถอดรหัสตำแหน่งจะทำการถอดรหัสตำแหน่งของหน่วยความจำโดยจะทำงานร่วมกับสัญญาณ RD และ WR ของไมโครคอนโทรลเลอร์ เบอร์ 8031 โดยจะทำการถอดรหัสตำแหน่งของหน่วยความจำของตำแหน่งของหน่วยความจำสำหรับข้อมูลภายนอก โดยตำแหน่งของ RAM1 คือแอดเดรส 0000-1FFF โดยสัญญาณที่ได้คือสัญญาณ CS-RAM1 และตำแหน่งของ RAM2 คือแอดเดรส 2000-3FFF โดยสัญญาณที่ได้คือ CS-RAM2 และจะประกอบด้วยสัญญาณเพื่อไปทำการถอดรหัสของตำแหน่งของ Phase Step Size คือ ตำแหน่งของหน่วยความจำตำแหน่งที่ E000-E002 เพื่อนำไปทำการเลือกตำแหน่งของ Phase Step Size เพื่อที่จะทำการส่งค่าของ Phase Step Size ออกไปแต่ละตำแหน่ง โดยไปทำการเซ็ทวงจรแลตช์ข้อมูล Phase Step Size เพื่อที่จะนำไปทำการกำหนดค่าเริ่มต้นของการบวกวนซ้ำให้กับวงจร Phase Accumulator



รูปที่ 5 วงจรการถอดรหัสตำแหน่ง

3. <u>วงจร Phase Step Size</u>

จากรูปส่วนนี้ประกอบไปด้วย IC 74LS374 U11, U12 และ U13 ซึ่งประกอบกันเป็นวงจรที่ทำหน้าที่แลตช์ข้อมูล ซึ่งวงจรนี้จะทำการสร้างค่า Phase Step Size (Port0-Port23) ไปให้วงจร Phase Accumulator โดยได้รับการควบคุมการสร้างจากวงจรการถอดรหัสตำแหน่ง IC74LS138 U4 และ U5 โดยสร้างสัญญาณ ADDRESS (E000), ADDRESS (E001) และ ADDRESS (E002) มาทำการควบคุมค่า Phase Step Size นี้จะเป็นอินพุทให้กับวงจร Phase Accumulator อีกทีเพื่อที่จะเป็นตัวกำหนดความถี่ของสัญญาณ

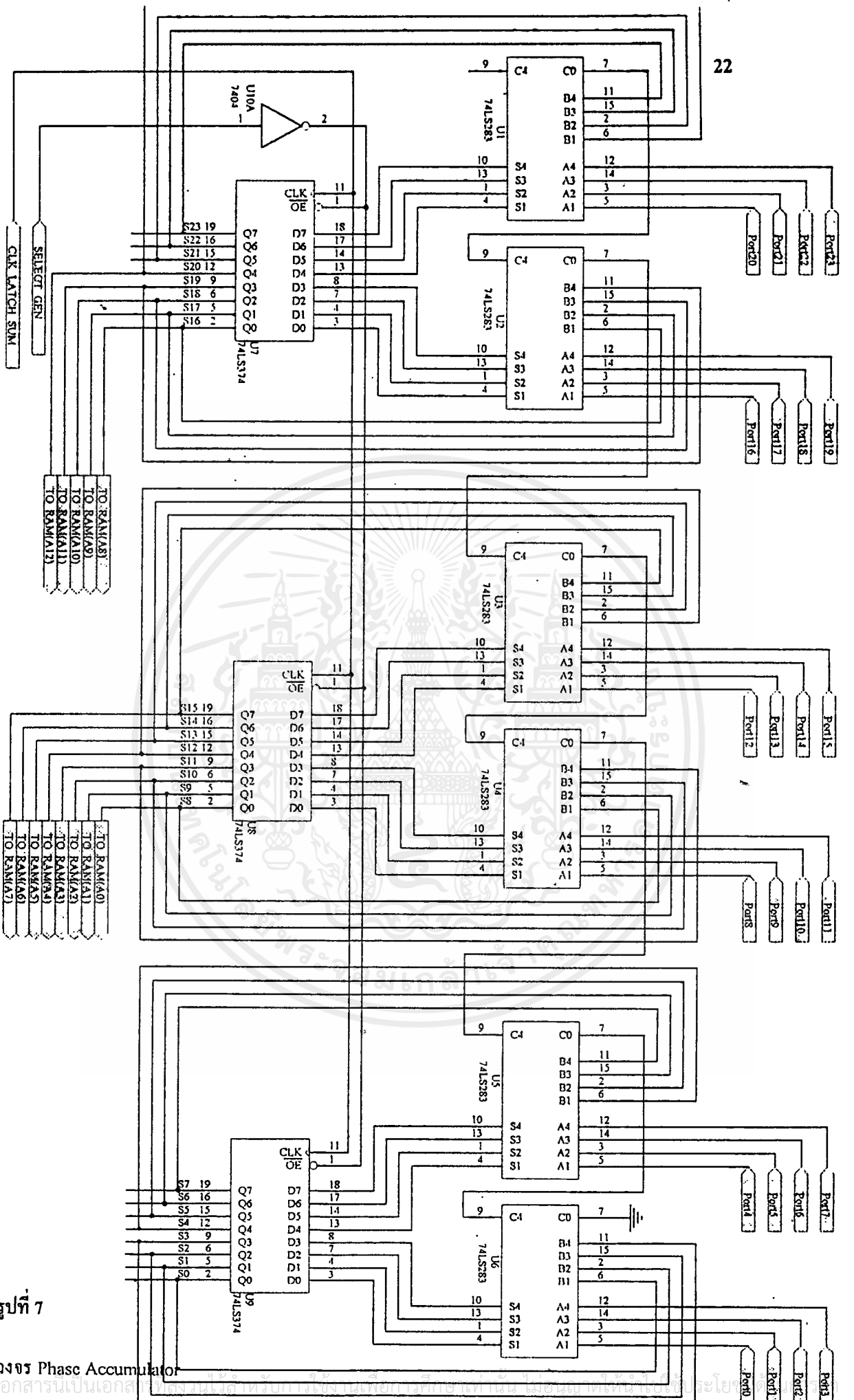


รูปที่ 6 วงจร Phase Step Size

4. <u>วงจร Phase Accumulator</u>

จากรูปวงจรที่ประกอบด้วย IC 74LS283 U1-U6 และ U7, U8 และ U9 จากรูปวงจร Phase Accumulator นี้จะทำการรับค่า Phase Step Size (Port0-Port23) แล้วทำการบอกกับค่าเฟสเดิมที่ทำการสะสมไว้ได้เป็น เฟสสะสมใหม่ที่สัญญาณ SO_S23 ของ 74LS283 นี้ จะถูกควบคุมโดยสัญญาณนาฬิกา CLK_LATCH_SUM จากวงจรกำเนิดสัญญาณอ้างอิง (Clock Generator) เฟสสะสมได้จากวงจรนี้จะใช้สัญลักษณ์ว่า TO_RAM (AO).- TO_RAM (A12) มาทำการป้อนให้กับแอดเดรส AO-A12 ของ RAM1 โดยร่วมกับสัญญาณ SELECT _ GEN เพื่อควบคุมการจ่ายแอดเดรสให้กับ RAM1 โดยร่วมกับสัญญาณ SELECT_GEN นี้ จะทำการเลือกว่าจะทำการติดต่อกับ RAM1 แบบใด โดยถ้าหากว่า สัญญาณของ SELECT_GEN นี้มีสภาวะลอจิกเป็น " 0 " จะทำการติดต่อกับ RAM1 ตามปกติ โดยไมโครคอนโทรลเลอร์ เบอร์ 8031 สามารถ ทำการอ่านค่าหรือ ทำการเขียนข้อมูลได้ตามปกติ แต่ถ้าหากว่าสัญญาณของ SELECT_GEN มีสภาวะลอจิกเป็น " 1 " จะสามารถทำการอ่านข้อมูลออกจาก RAM1 ได้อย่างเดียว โดยตำแหน่งของหน่วยความจำของ RAM1 นั้นจะได้รับสัญญาณมาจากส่วนของวงจร Phase Accumulator หรืออาจกล่าวได้อีกนัยหนึ่งว่า ถ้าหากว่าสัญญาณ SELECT_GEN มีสภาวะลอจิกเป็น " 1 " จะเป็นการกำเนิดสัญญาณความถี่ใด ที่เราทำการเก็บค่าต่าง ๆ ของรูปคลื่น 1 คาบเวลาเอาไว้ใน RAM1 เพราะว่าข้อมูลของ RAM1 จะถูกส่งให้กับวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกเปลี่ยนเป็นค่าสัญญาณความถี่ต่าง ๆ ออกทางเอาท์พุทของวงจร

จากรูปเราจะเห็นได้ว่า มี IC U7404 เป็นวงจรที่ทำหน้าที่การเลือกว่าจะเอาค่าของหน่วยความจำข้อมูลไปป้อนให้กับ RAM1 มาจากที่ใดระหว่างตำแหน่งของหน่วยความจำที่มาจาก Phase Accumulator กับค่าตำแหน่งของหน่วยความจำที่มาจากไมโครคอนโทรลเลอร์ เบอร์ 8031 โดยตรง

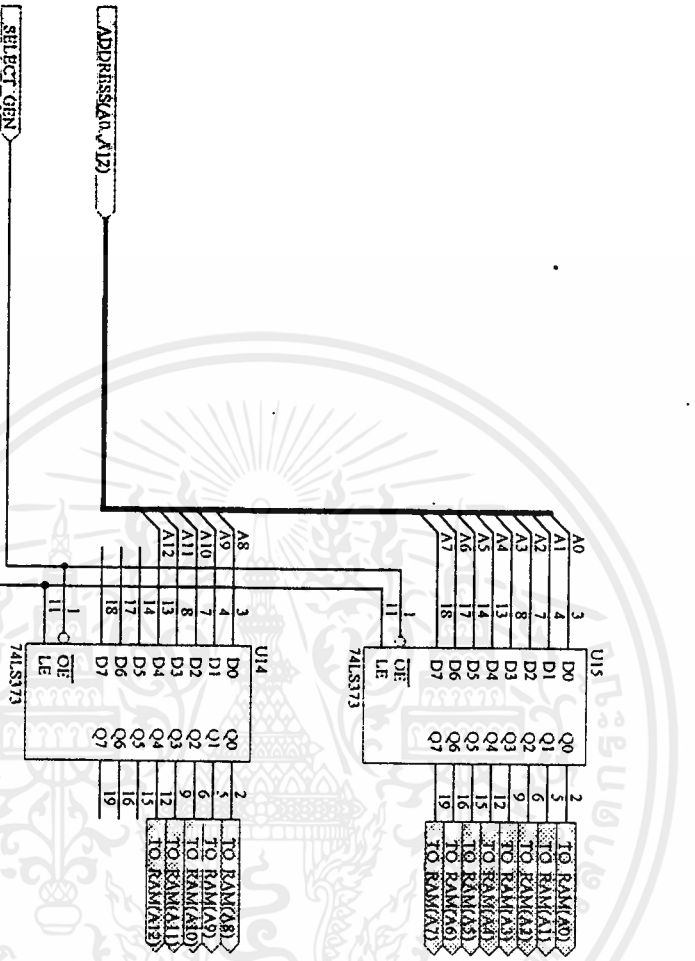


รูปที่ 7

วงจร Phase Accumulator

5. **วงจร RAM Address Generator**

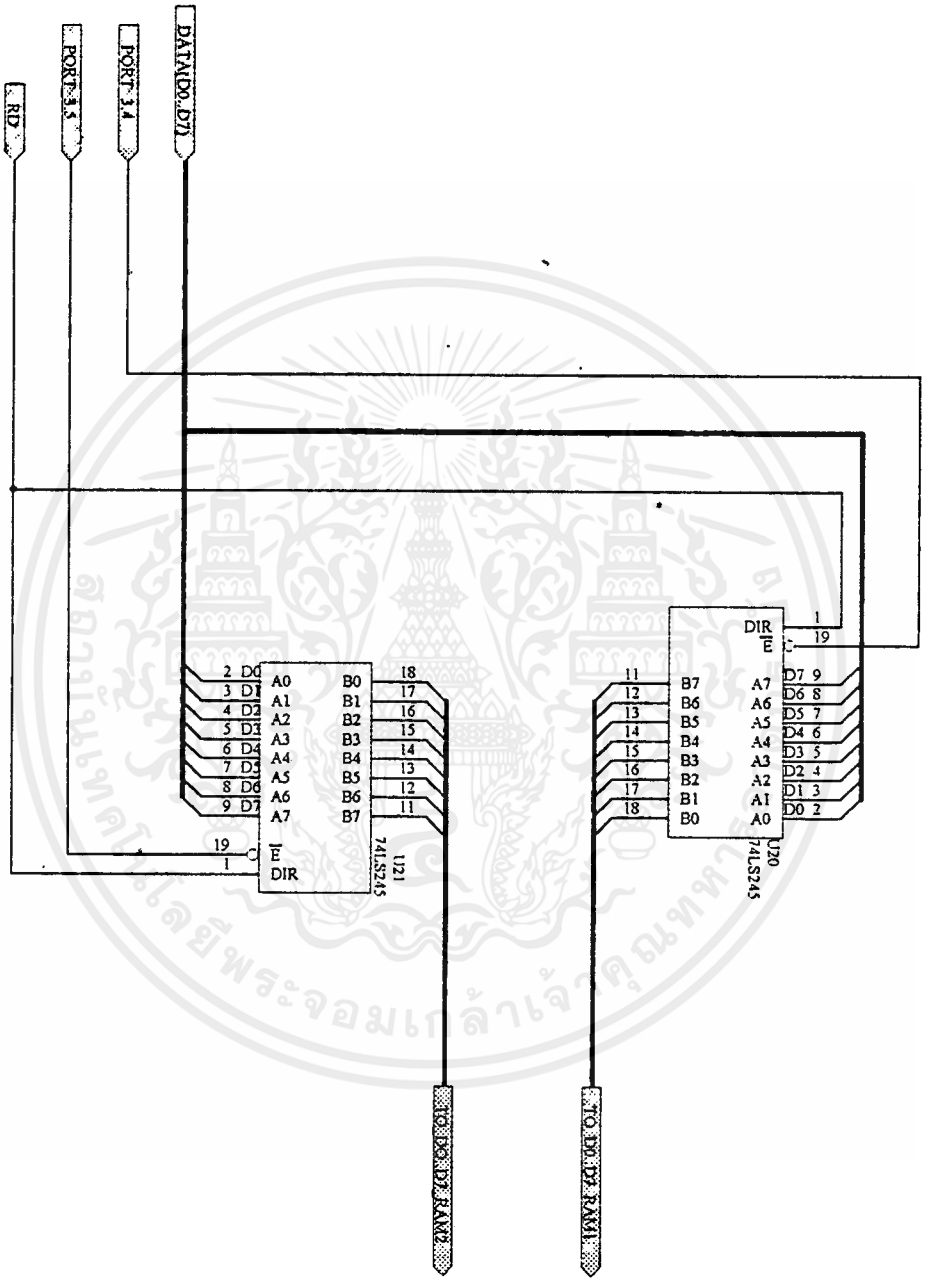
จากรูปของวงจรได้แก่ IC 74LS373 U14 และ U15 วงจรนี้จะทำการสร้างตำแหน่งของหน่วยความจำข้อมูลเพื่อป้อนให้กับ แรม 6264 ในจังหวะที่ต้องการที่จะทำการ Program Pattern ของ function wave form ลงไปใน RAM หรือในจังหวะที่ต้องการอ่านข้อมูลหรือเขียนข้อมูลลงไปใน RAM ตามปกติ โดยตำแหน่งของหน่วยความจำข้อมูล นี้จะได้รับมาโดยตรงจากไมโครคอนโทรลเลอร์ เบอร์ 8031 แต่จังหวะในการแลตช์ของตำแหน่งของหน่วยความจำภายนอกนี้จะอาศัยสัญญาณควบคุมจาก SELECT_GEN



รูปที่ 8 วงจร RAM Address Generator

6. **วงจร RAM Data Generator**

วงจรนี้ประกอบด้วย IC 74LS245 U20 และ U21 ซึ่งเป็นไอซีทำหน้าที่เป็นบัฟเฟอร์ สองทิศทางโดยมีขา E เป็นขาควบคุมการทำงานของ บัฟเฟอร์ ส่วนขา DIR จะเป็นขาสัญญาณใช้ควบคุมทิศทางของบัฟเฟอร์ คือจะให้ข้อมูลเคลื่อนย้ายจากอินพุตด้าน A ไปยังเอาท์พุทด้าน B หรืออินพุตด้าน B ไปยังเอาท์พุตด้าน A โดยการทำงานมีลักษณะการทำงานดังนี้ ถ้าลอจิกที่ขา DIR เป็น 1 ข้อมูลจะส่งผ่านจาก Aไป B แต่ถ้าหากว่าลอจิกที่ขา DIR เป็น " 0 " ข้อมูลจะถูกส่งผ่านจาก B ไปยัง A โดยข้อมูลที่ขา E จะต้องเป็น 0 ตลอดเวลา แต่ถ้าหากว่าเมื่อใดก็ตาม ถ้าลอดจิกที่ขา E เป็น " 1 " จะทำให้เกิดสภาวะอิมพีแดนซ์สูง เสมือนว่า บัสระหว่าง A กับ B แยกขาดออกจากกันวงจรส่วนนี้จะทำหน้าที่ติดต่อกับหน่วยความจำโดยตรง ผ่านทางไมโครคอนโทรลเลอร์ เบอร์ 8031 โดยจะทำหน้าที่ส่งข้อมูลของรูปสัญญาณให้กับ RAM1 โดยตรง โดยผ่านทาง RAM Data Generator เข้าไปสู่แรม 6264 จังหวะการแลตช์ของข้อมูลในวงจรนี้จะมาจากสัญญาณ PORT_3.4 กับสัญญาณ PORT_3.5 โดยมาทำการควบคุมที่ขา E ของ U21 จะต่อเข้ากับขา DO-D7 ของ RAM1 และ U21 จะต่อเข้ากับขา D0-D7 ของ RAM1 และ RAM2 โดยเราจะใช้สัญญาณ PORT 3.5 ในกรณีที่เราต้องกการติดต่อกับ RAM2 โดยจะทำงานร่วมกับสัญญาณ RD จากไมโครคอนโทรลเลอร์ เบอร์ 8031 มาควบคุมการอ่านและเขียนของ RAM อีกทีโดยผ่านทางขา DIR ของ U20 และ U21 แยกจากกันนั้นก็เพราะว่า ในกรณีที่เราทำการเขียนข้อมูลลงไปในหน่วยความจำนั้นสามารถเขียนได้ตามปกติ แต่ถ้าเป็นการอ่านข้อมูลจากหน่วยความจำแล้ว ถ้าหากว่าสัญญาณควบคุม U20 และ U21 เป็นตัวเดียวกันแล้ว ข้อมูลที่ถูกอ่านเข้ามาในไมโครคอนโทรลเลอร์ จาก RAM1 และ RAM2 มาพร้อมกันเลยเกิดการชนกันของข้อมูล แต่เมื่อใช้สัญญาณควบคุม U20 และ U21 คนละตัวกันนั้น สามารถเลือกอ่านข้อมูลจากแรมได้ที่ละตัว ได้โดยไม่เกิดการชนกันของข้อมูลที่อ่านเข้ามา

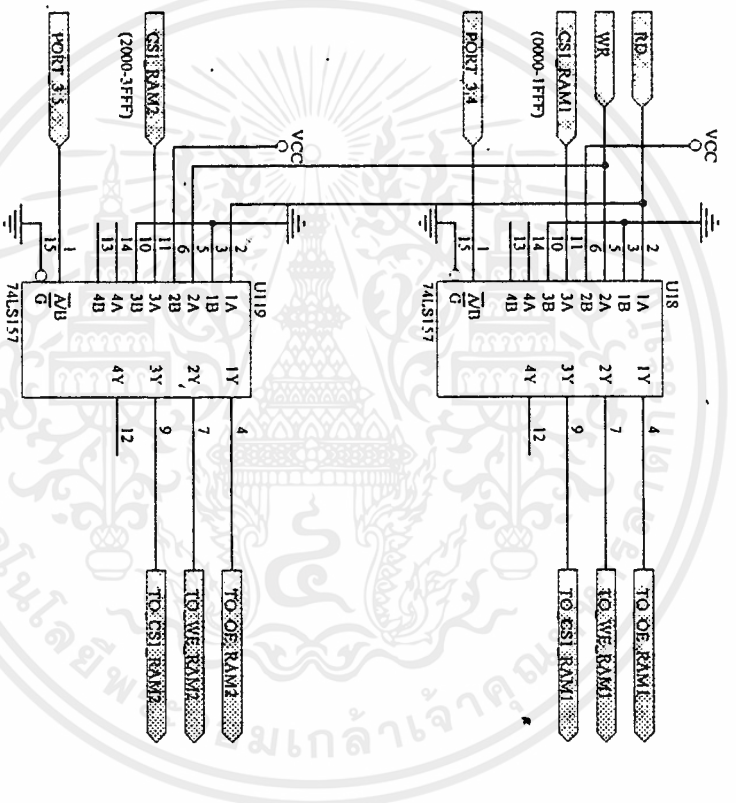


รูปที่ 9 RAM Data Generator

7. <u>วงจรมัลติเพล็ก (MULTIPLEX)</u>

วงจรมัลติเพล็ก แสดงดังรูปจะประกอบด้วย IC 74LS157 U18 และ U19 ซึ่งจะทำหน้าที่ว่าจะเลือกติดต่อกับ RAM1 และ RAM2 ว่าจะทำงานในลักษณะใด จากรูปจะยกตัวอย่างการทำงานของ U18 โดย U18 จะทำหน้าที่ควบคุม RAM1 โดยสัญญาณควบคุม RAM1 ที่มาจากไมโครคอนโทรเลอร์ เบอร์ 8031 นั้นมีสัญญาณ RD, WR และ CS1_RAM1 (0000-IFFF) จะเป็นสัญญาณอินพุทให้กับ 74LS157 และจะมีสัญญาณ PORT_3.4 มีสภาวะลอจิกเป็น " 0 " การทำงานของ RAM1 จะทำการติดต่อกับไมโครคอนโทรลเลอร์เบอร์ 8031 ได้โดยตรง สามารถที่จะทำการเขียนข้อมูลและทำการอ่านข้อมูลจาก RAM1 ได้ตามปกติ แต่ถ้าสัญญาณจาก PORT_3.4 มีสภาวะลอจิกเป็น " 1 " จะเป็นสภาวะที่เราสามารถอ่านข้อมูลออกจาก RAM1 ได้เพียงอย่างเดียว โดยค่าตำแหน่งของหน่วยความจำที่มาชี้ที่ A0-A12 ของ RAM1 นี้มาจากการทำงานของวงจร PHASE ACCUMULATOR ซึ่งจะเป็นการนำค่าของสัญญาณรูปคลื่น 1 คาบเวลาออกมาป้อนให้กับวงจรเปลี่ยนสัญญาณดิจิตอลเป็นอนาล็อกต่อไปซึ่งจะสามารถกำเนิดรูปคลื่นความถี่ต่าง ๆ ออกมา

การทำงานของ 74LS157 นี้เป็นไอซีมัลติเพล็กสัญญาณโดยสัญญาณเข้า 2 ออก 1 โดยจะมีขา A/B เป็นตัวที่ทำหน้าที่เลือกสัญญาณ ถ้าหากว่าขา A/B มีสภาวะลอจิกเป็น " 1 " สัญญาณที่เข้ามาทีอินพุทขา B จะออกไปยังเอาท์พุท Y

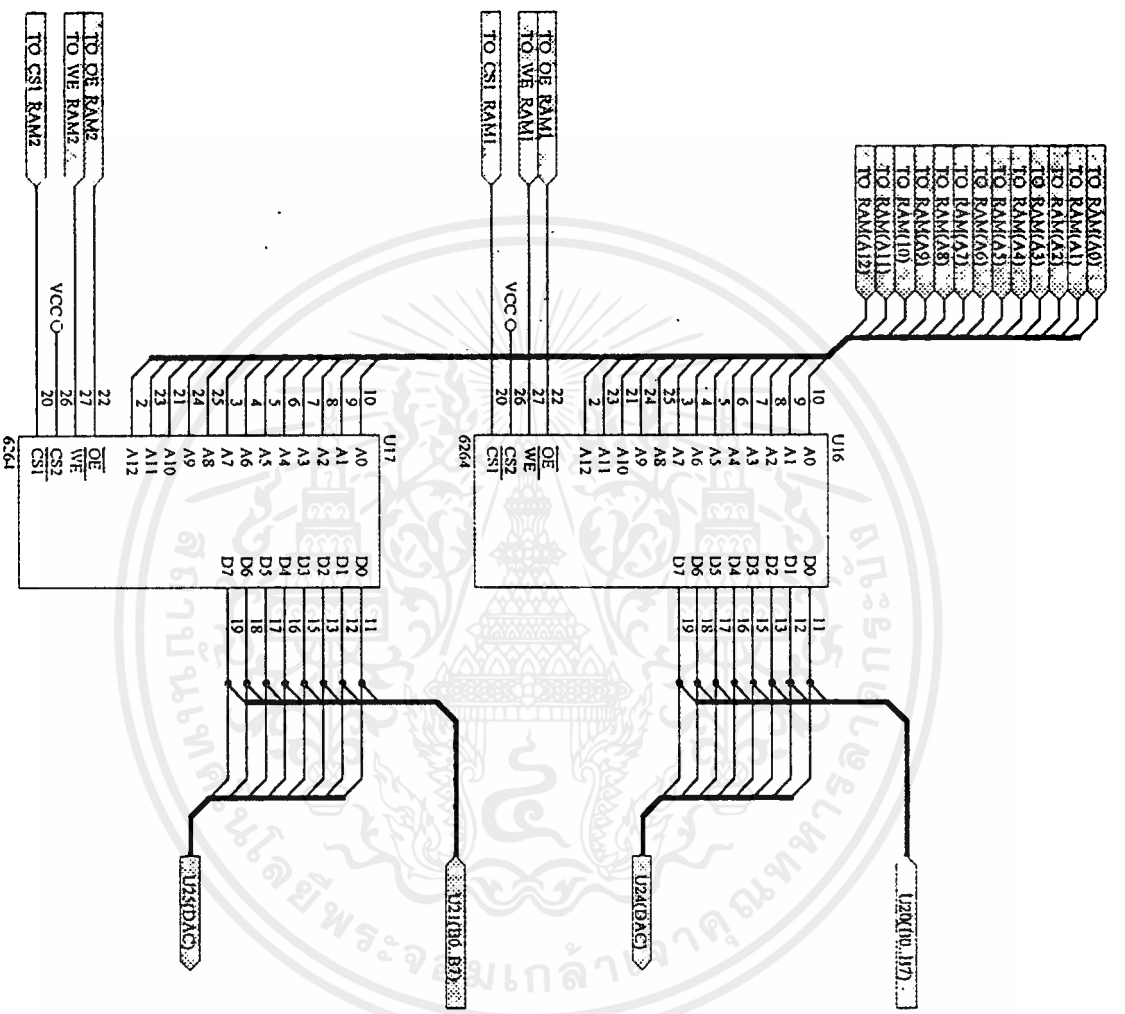


รูปที่ 10 วงจรมัลติเพล็ก (MULTIPLEX)

8. <u>วงจรหน่วยความจำ (RAM)</u>

วงจรนี้จะเป็นส่วนที่ทำหน้าที่เก็บข้อมูลของรูปคลื่น U17 และ U18 ทำหน้าที่เป็นตัวเก็บข้อมูลที่ได้จากการคำนวณ โดยโปรแกรมจากคอมพิวเตอร์นำมาบันทึกลง RAM1 และ RAM2 และจากนั้นก็จะทำหน้าที่อ่านข้อมูลจากหน่วยความจำเพื่อนำมาสร้างเป็นรูปคลื่นที่ต้องการ เราสามารถเปลี่ยนค่าความถี่ของรูปคลื่นโดย ทำการเปลี่ยนค่าของ Phase Step Size ที่มาป้อนให้กับวงจร Phase Accumulator เปลี่ยนไป ทำให้ความเร็วในการนับตำแหน่งของหน่วยความจำเปลี่ยนไป ทำให้เวลาในการเข้าถึงข้อมูล (Access Time) เปลี่ยนไป ส่งผลให้ข้อมูลที่ออกมาจากหน่วยความจำมีความเร็วเปลี่ยนไป เมื่อผ่านวงจร เปลี่ยนสัญญาณดิจิตอลเป็นอนาล็อก ก็ทำให้รูปคลื่นที่ได้มีความถี่เปลี่ยนไป

หน่วยความจำเราใช้สแตติกแรม เบอร์ 16264 ซึ่งมีหน่วยความจำ 8 kbyte เป็นแรมที่มีความเร็วสูง ควบคุมการอ่านข้อมูล และเขียนข้อมูล จากไมโครคอนโทรเลอร์ เบอร์ 8031 โดยการควบคุมผ่านทางวงจรมัลติเพล็กอีกทีหนึ่ง

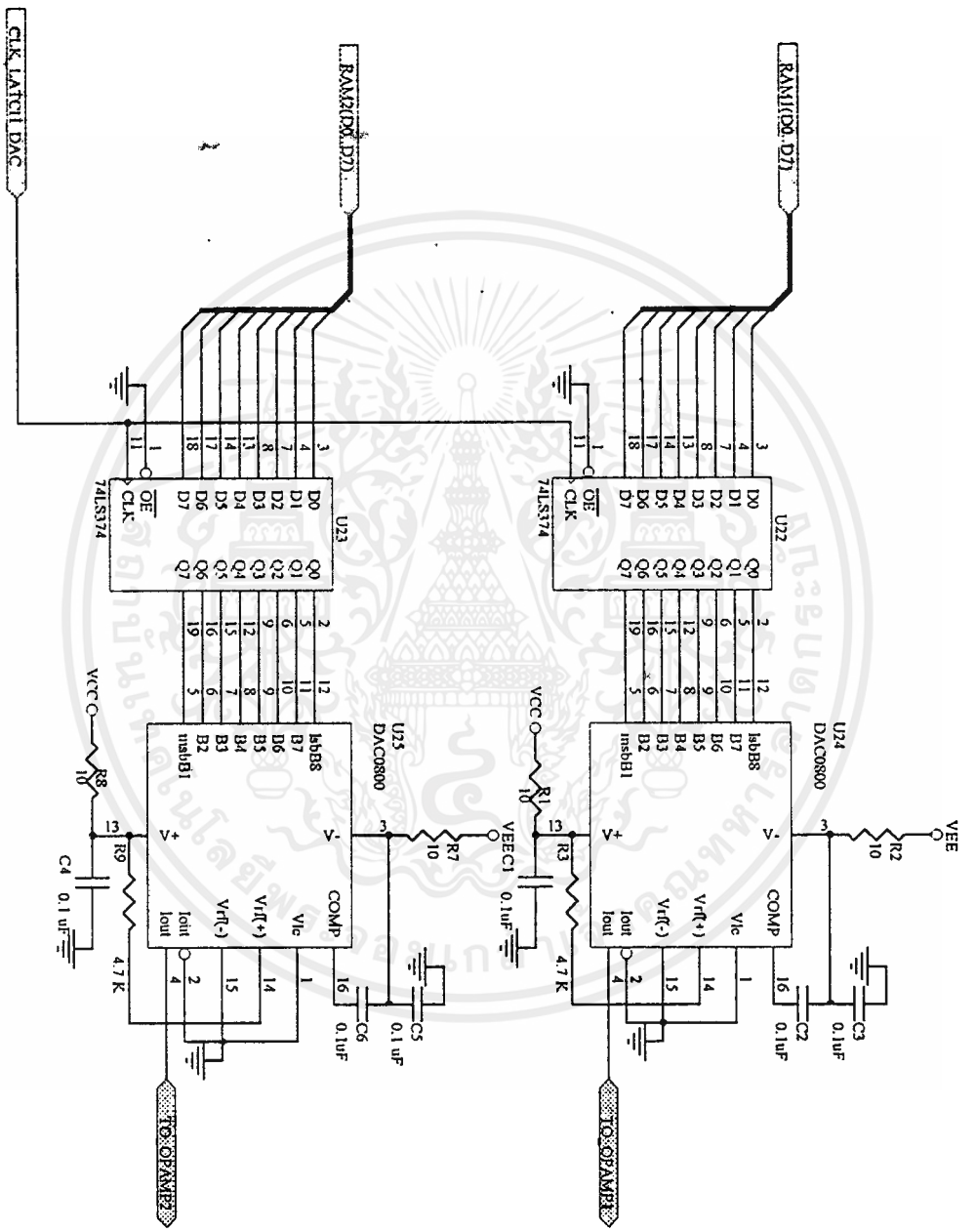


รูปที่ 11 วงจรหน่วยความจำ (RAM)

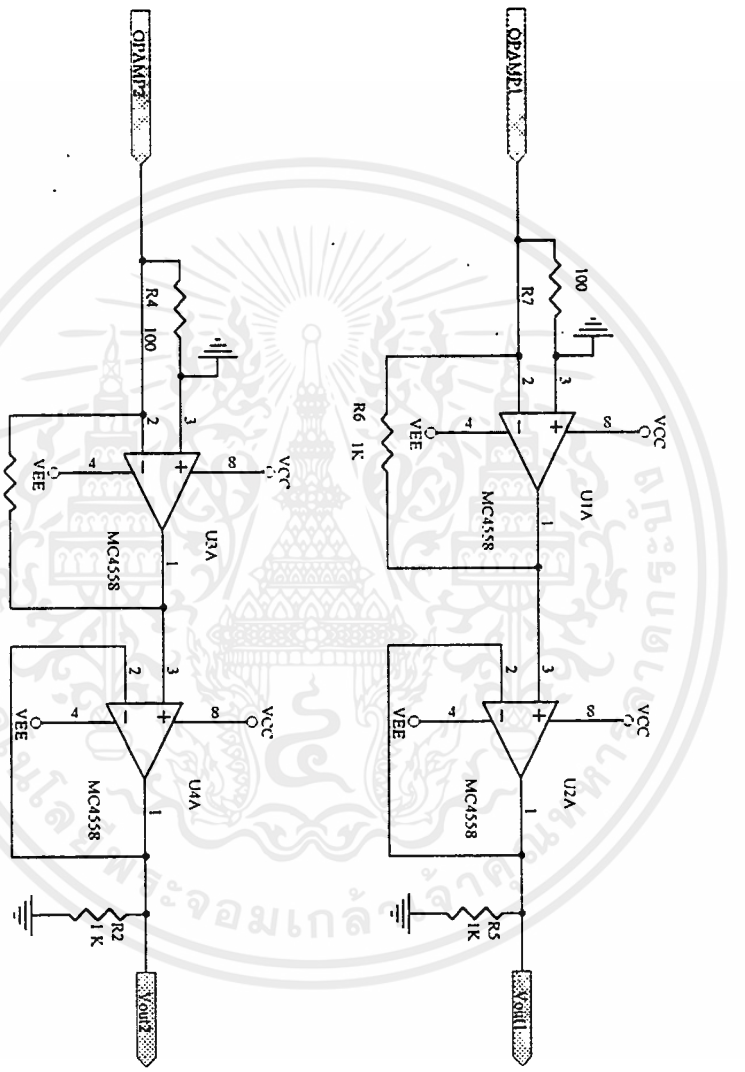
9. <u>วงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อก (DAC)</u>

วงจรนี้ทำหน้าที่แปลงสัญญาณดิจิตอลมาเป็นระดับสัญญาณแรงดันไฟตรง วงจรนี้เราใช้ IC DAC 0800 ซึ่งควบคุมแรงดันเอาท์พุทได้จากไบนารีอินพุทที่ป้อนให้ในการออกแบบใช้ Vref +5v เอาท์พุทที่ได้เป็นกระแส และใช้ IC OP - AMP MC 4558 ทำการเปลี่ยนกระแสให้เป็นแรงดัน วงจรนี้ประกอบด้วย IC 74LS374 U22 และ U23 DAC 0800 U24 และ U25 และ MC4558 ทำงานร่วมกันสัญญาณนาฬิกาควบคุมจังหวะการจ่ายสัญญาณดิจิตอลให้กับวงจร DAC โดยสัญญาณดิจิตอลก็จะนำมาแปลงเป็นกระแส และแปลงกระแสให้เป็นโวล์ทเตจโดย MC 4558 จึงได้สัญญาณความถี่ที่เรากำหนดขึ้นมาออกไปใช้งาน

ความละเอียดของความถี่เอาท์พุทของสัญญาณในระบบ DDS นี้ขึ้นอยู่กับสัญญาณนาฬิกาอ้างอิง และจำนวนบิทในส่วนของ Phase Accumulator ค่าความละเอียดของความถี่เท่ากับ $Fc/2^N$



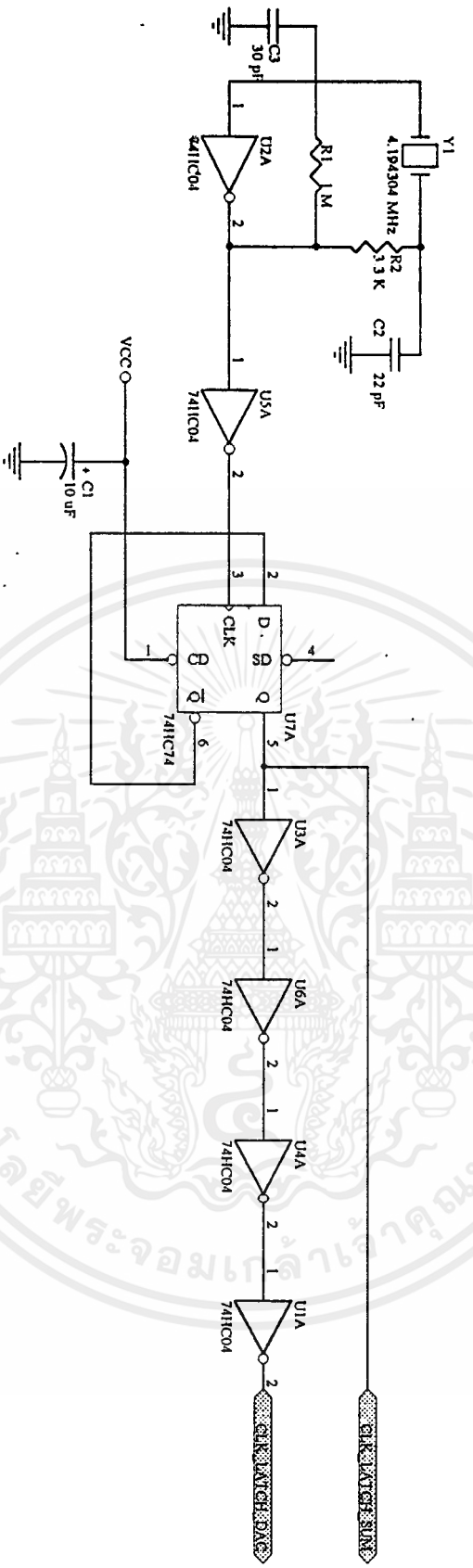
รูปที่ 12 วงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก



รูปที่ 13 วงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก(ต่อ)

10. <u>วงจรกำเนิดสัญญาณนาฬิกาอ้างอิง (Clock Generator)</u>

จากวงจรในรูป IC 74HC04 U2A ร่วมกับคริสตอล ความถี่ 4.1943 MHz , R1,R2,C2,C3 กำเนิดสัญญาณนาฬิกาขึ้นมาแล้วนำไปป้อน 74HC74 U5A เพื่อทำการหารสองความถี่สัญญาณนาฬิกาที่ได้ เป็นสัญญาณ CLK_LATCH_SUM และสัญญาณ CLK_LATCH_DAC สัญญาณ CLK_LATCH_SUM นี้ป้อนให้กับวงจร Phase Accumulator เพื่อควบคุมจังหวะการบวกสัญญาณ CLK_LATCH_SUM นี้จะถูก Delay ด้วย Inverter gate ที่เหลือคือ 74HC04, U3A, U4A และ U1A 4 ตัว ได้เป็นสัญญาณใหม่ เรียกว่าสัญญาณ CLK_LATCH_DAC สัญญาณที่ได้นี้ใช้ควบคุมจังหวะการแลตช์ของดาต้าที่ออกจากหน่วยความจำสู่วงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อก



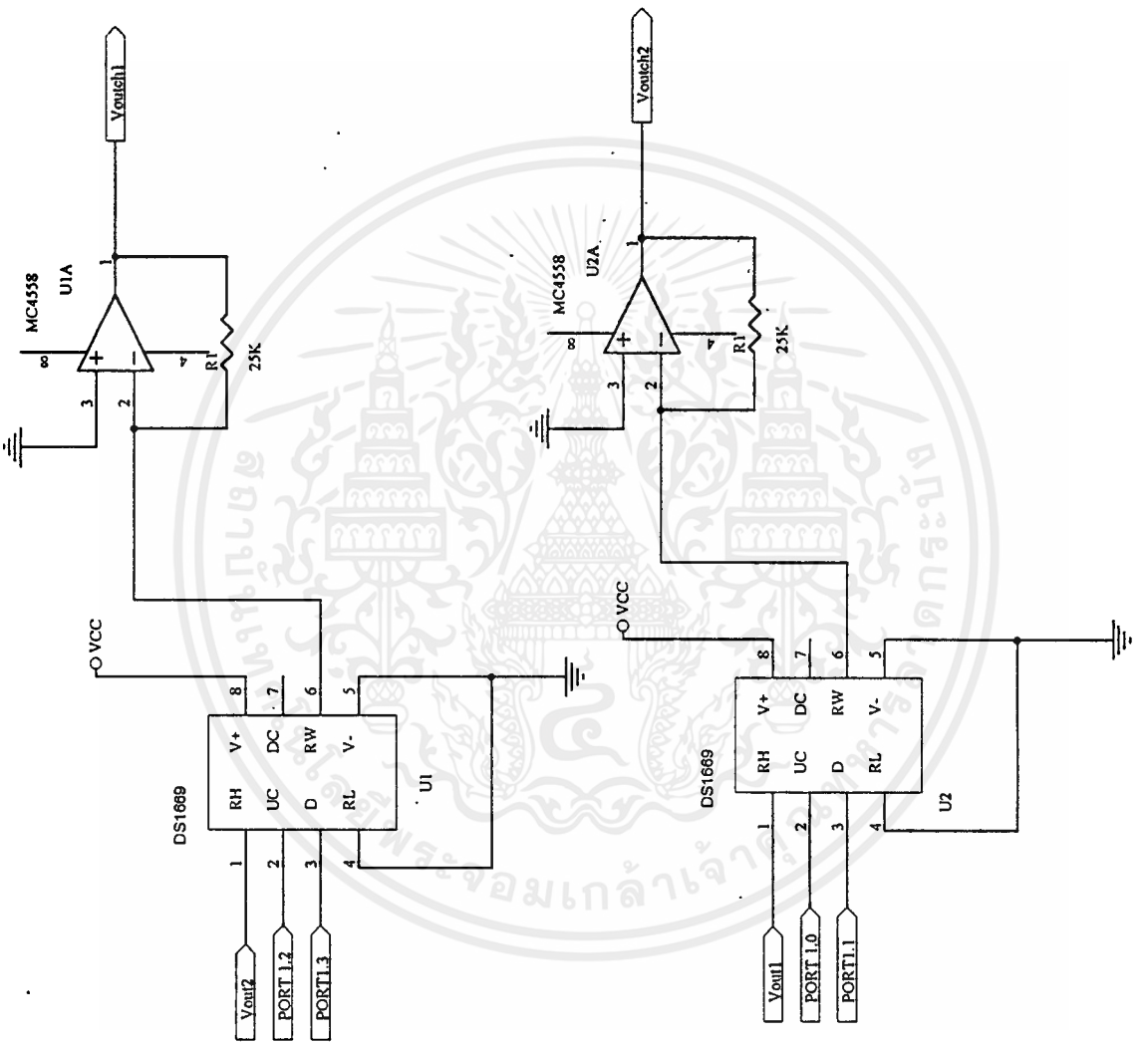
รูปที่ 14 วงจรกำเนิดสัญญาณนาฬิกาอ้างอิง

11. <u>วงจรขยายปรับค่าได้</u>

วงจรขยายปรับค่าได้นี้ อาศัยการทำงานของตัวต้านทานปรับค่าได้ DS1669 ซึ่งเป็นตัวต้านทานแบบปรับค่าได้ สามารถที่จะปรับเปลี่ยนค่าได้จากการปิดหน้าสัมผัสทาง INPUT หรือที่สามารถปรับค่าได้ทางการควบคุมการจ่ายสัญญาณดิจิตอลควบคุมที่ขา INPUT ได้ ในที่นี้เลือกที่จะใช้การปรับค่าทางการป้อนสัญญาณ INPUT ให้กับมัน DS1669 ตัวนี้สามารถเปลี่ยนค่าความต้านทานจากช่วง 0-50K โดยแบ่งออกเป็น 64 ระดับของความต้านทานของตัวมัน โดยการต่อวงจรนี้จะนำมาต่อร่วมกับออปแอมป์ 4558 โดยให้ DS1669 เป็น Rin ส่วน Rf มีค่า 25K ทำให้สัญญาณเอาท์พุทต่ำสุดคือ 0.5 V

จาก $V_o=(R_f/R_{in})*V_{in}$

สัญญาณเอาท์พุทมากที่สุดได้ 4 Vp-p โดยสัญญาณควบคุมความต้านทานของ DS1669 นี้มาจาก P1.0 และ P1.1 สำหรับช่องสัญญาณที่ 1 ,P1.2 และ P1.3สำหรับช่องสัญญาณที่ 2โดยสัญญาณควบคุมนี้ เกิดจากโปรแกรมควบคุมจากคอมพิวเตอร์ทำการคำนวณค่าออกมาแล้วทำการป้อนให้ไมโครคอนโทรลเลอร์อีกทีหนึ่ง



รูปที่ 15 วงจรขยายแบบปรับค่าได้

บทที่ 4

การทดลองและผลการทดลอง

การทดลองการทำงานของเครื่องกำเนิดความถี่ชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลขนี้เป็นการทำงานร่วมกันระหว่างโปรแกรมคำนวณรูปร่างสัญญาณจากคอมพิวเตอร์ ซึ่งจะคำนวณและทำการเลือกรูปแบบของสัญญาณและทำการส่งรูปคลื่นที่ได้ไปให้กับไมโครคอนโทรลเลอร์ทำการสร้างรูปคลื่นความถี่ต่าง ๆ อีกครั้งหนึ่ง

การทดลองแรกจะเป็นการทดลองหาค่า BANDWIDTH ของเครื่องกำเนิดสัญญาณความถี่นี้ในกรณีที่เป็นการกำเนิดสัญญาณซายน์โดยการทดลองจะทำการกำเนิดค่าความถี่แล้วทำการวัดค่าแอมปลิจูดของความถี่นั้น แล้วทำการพิจารณาว่า ณ ความถี่ใดที่ทำให้แอมปลิจูดของสัญญาณมีค่าเท่ากับ 0.707 เท่าของแอมปลิจูดสูงสุดของสัญญาณซายน์ การทดลองหาค่า BANDWIDTH ของเครื่องกำเนิดสัญญาณนั้นจะทำการทดลองเฉพาะสัญญาณซายน์เท่านั้น

การทดลองที่สองเป็นการทดลองกำเนิดสัญญาณรูปคลื่นต่างๆ ที่ความถี่ต่างๆ กัน แล้วสังเกตการเปลี่ยนแปลงของสัญญาณเมื่อความถี่ของสัญญาณเปลี่ยนแปลงไป

ผลการทดลอง

เปรียบเทียบค่าโวล์ทเตจเอาท์พุทของสัญญาณที่ความถี่ต่างๆ

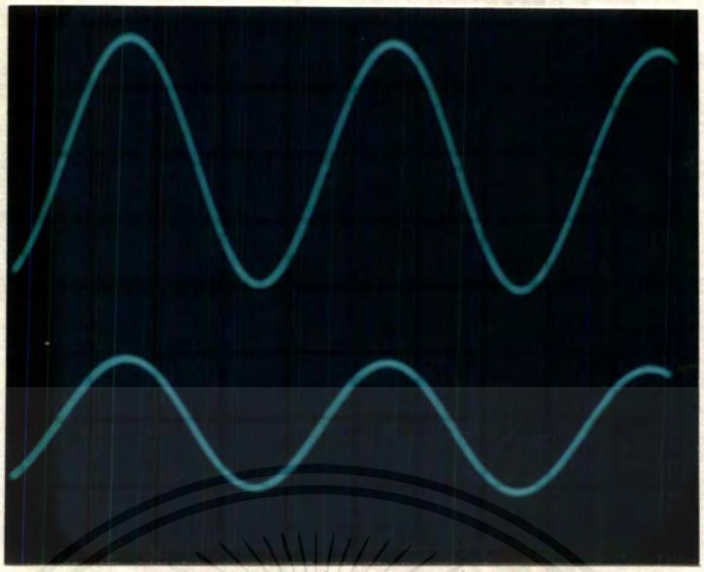
FREQUENCY	VOLTAGE
200 Hz	1
1 kHz	1
20 kHz	1
40 kHz	1
70 kHz	1
150 kHz	0.96
190 kHz	0.92
200 kHz	0.92
220 kHz	0.88
230 kHz	0.82
240 kHz	0.81

FREQUENCY	VOLTAGE
250 kHz	0.81
270 kHz	0.8
280 kHz	0.8
290 kHz	0.76
300 kHz	0.76
320 kHz	0.72
325 kHz	0.71
330 kHz	0.7

จากผลการทดลองแรกจะสังเกตเห็นได้ว่า BANDWIDTH ของสัญญาณซายน์อยู่ที่ความถี่ประมาณ 325 kHz

ผลการทดลองที่สองเป็นการทดลองทำการกำเนิดรูปคลื่นต่าง ๆ ที่ความถี่ต่างกัน และที่ค่าของแอมปลิจูดอยู่ที่ค่า 0.5 - 4 V จากการทดลอง มีผลการทดลองดังนี้

จากผลการทดลองที่สอง เป็นการกำเนิดรูปคลื่นที่ความถี่ต่างๆ กันจะเห็นได้ว่าเมื่อความถี่ของสัญญาณสูงขึ้น ความเพี้ยนของรูปคลื่นจะมากขึ้นเพราะว่าจำนวนจุดของข้อมูลต่อหนึ่งรูปคลื่นมีค่าน้อยลง ทำให้รูปของสัญญาณเพี้ยนไป

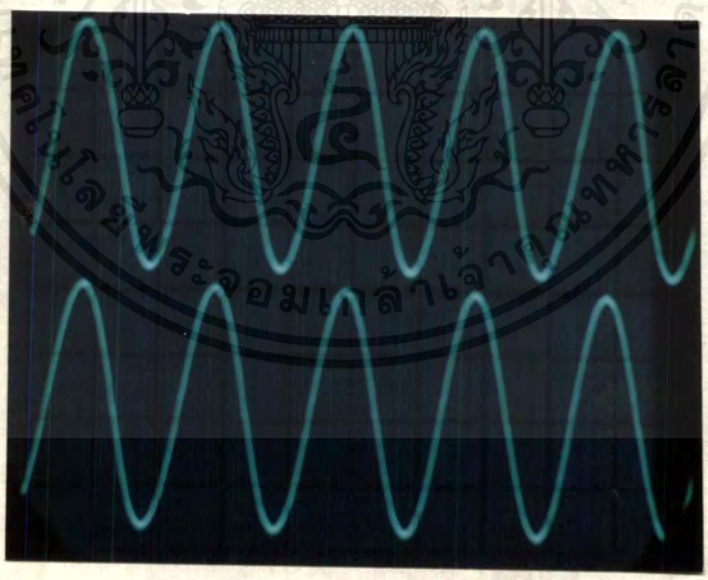


ทำการวัดสัญญาณเอาท์พุทของวงจรทำการเปรียบเทียบค่าโวลท์เตจเอาท์พุทระหว่าง Channel 1 และ Channel 2 ที่ความถี่เดียวกัน Channel 1 = 3.8 Volts , Channel 2 = 2 Volts

FREQUENCY = 5 kHz

TIME/DIV = 50 uS

VOLTS/DIV = 1 Volt

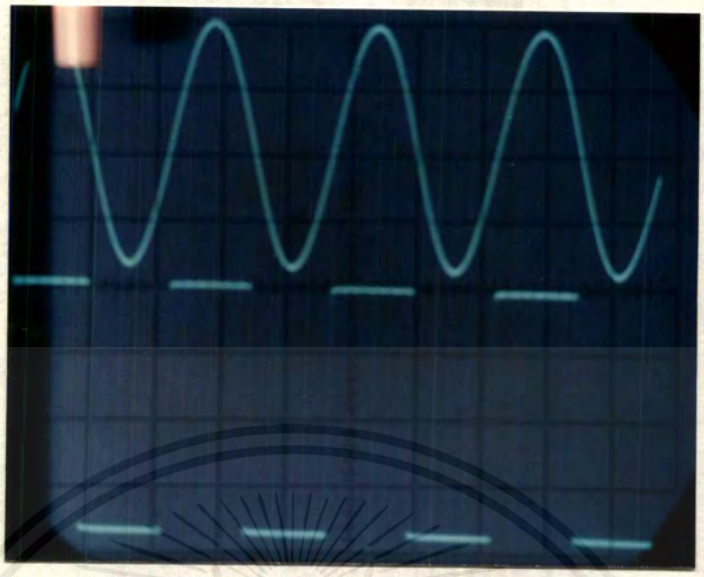


จากรูปแสดงค่าแอมปลิจูดสูงสุดของแต่ละ Channel โดยค่าแอมปลิจูดของ แต่ละ Channel แสดงดังต่อไปนี้ Channel 1 = 3.8 Volts , Channel 2 = 3.8 Volts

FREQUENCY = 10 kHz

TIME/DIV = 50 uS

VOLTS/DIV = 1 Volt

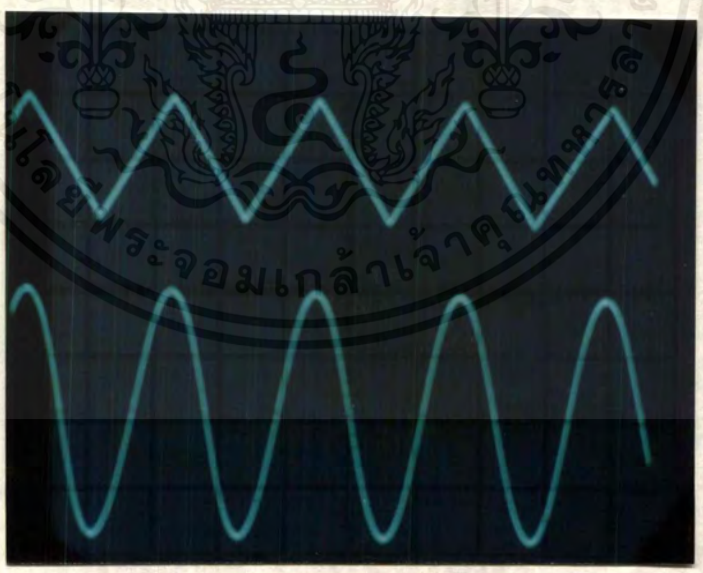


จากรูปแสดงการกำเนิดรูปสัญญาณรูปแบบต่างๆ โดยจากรูป Channel 1 แสดง SINE WAVE Channel 2 แสดง SQUARE WAVE ที่แอมปลิจูดเดียวกัน คือ 3.8 Volts

FREQUENCY = 8 kHz

TIME/DIV = 50 uS

VOLTS/DIV = 1 Volt

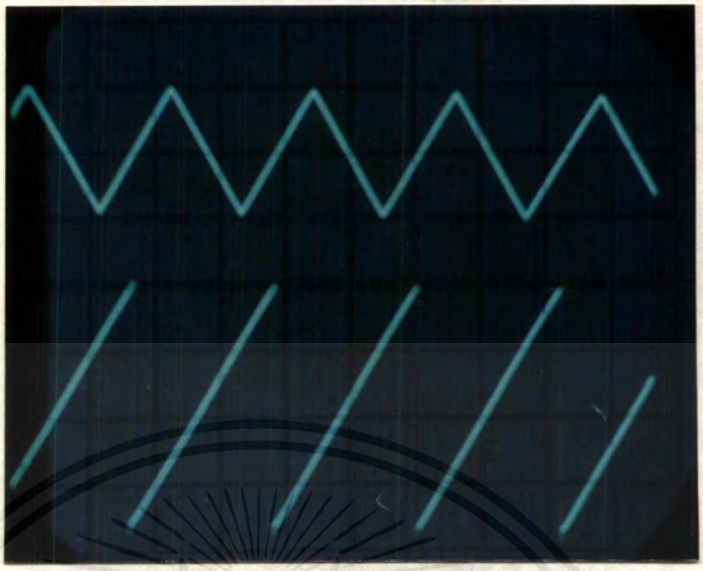


จากรูปแสดงการกำเนิดรูปสัญญาณรูปแบบต่างๆ กัน โดยจากรูป Channel 1 แสดง สัญญาณ TRIANGLE WAVE ที่แอมปลิจูด 2 Volts Channel 2 แสดง SINE WAVE ที่แอมปลิจูด 3.8 Volts

FREQUENCY = 4.2 kHz

TIME/DIV = 0.1 mS

VOLTS/DIV = 1 Volt

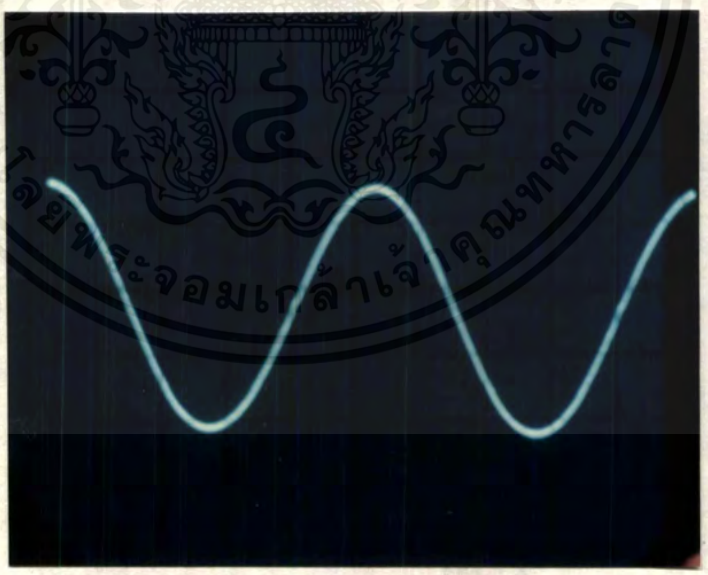


จากรูปแสดง Channel 1 แสดง TRIANGLE WAVE , Channel 2 แสดง SAWTOOTH

FREQUENCY = 4.2 kHz

TIME/DIV = 0.1 mS

VOLTS/DIV = 1 Volt

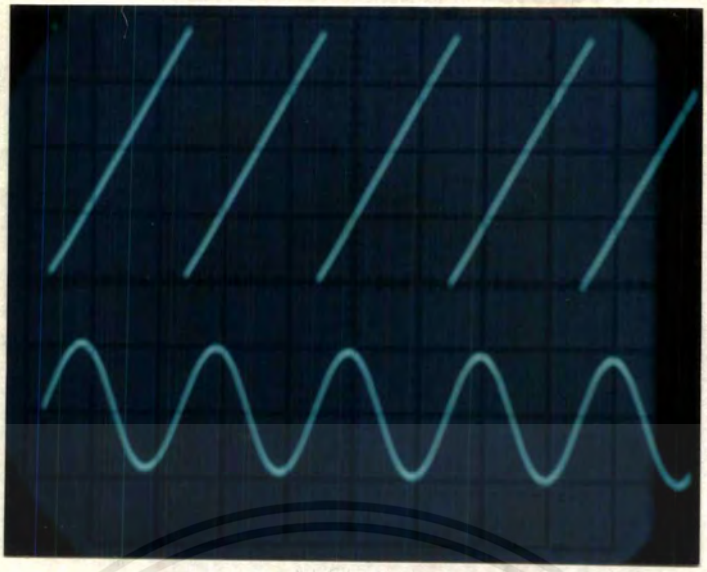


จากรูปเป็นการทดลองการกำเนิดสัญญาณ SINE WAVE ที่ความถี่ต่ำ ประมาณ 400 Hz ค่าแอมปลิจูด 3.8 Volts

FREQUENCY = 400 Hz

TIME/DIV = 0.5 mS

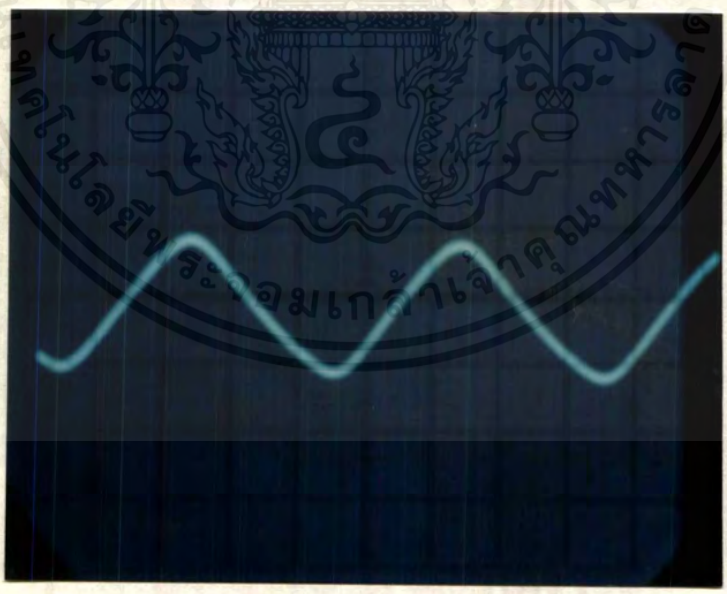
VOLTS/DIV = 1 Volt



FREQUENCY = 100 kHz

TIME/DIV = 5 uS

VOLTS/DIV = 1 Volt

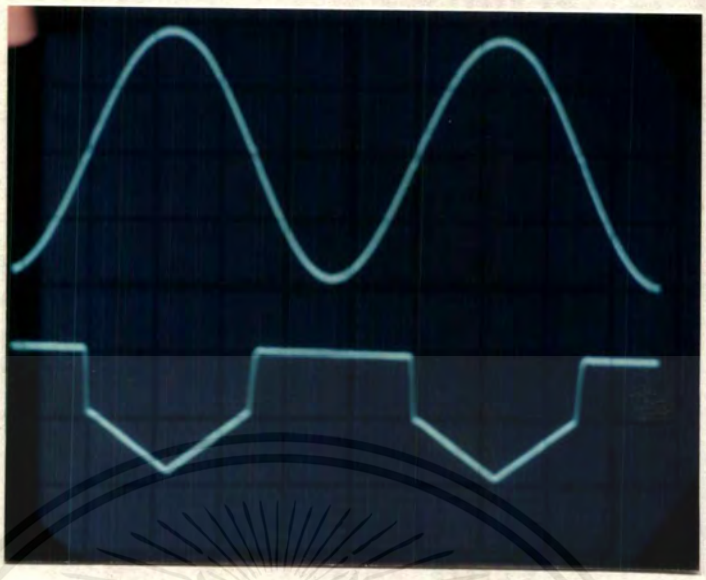


จากรูปเป็นการทดลองการกำเนิดสัญญาณ SINE WAVE ที่ความถี่สูงประมาณ 500 kHz ค่าแอมปลิจูด 2 Volts รูปสัญญาณจะมีความผิดเพี้ยนสูง

FREQUENCY = 500 kHz

TIME/DIV = 0.5 uS

VOLTS/DIV = 1 Volt

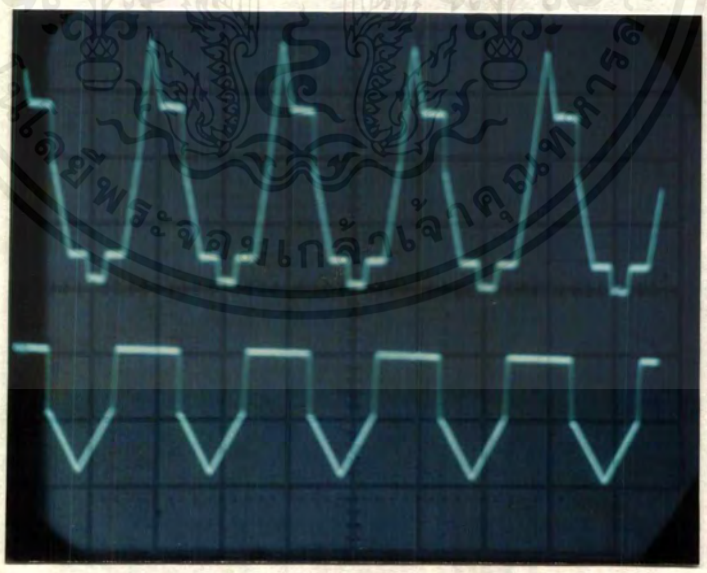


การทดลองการทำงานของโปรแกรม ในส่วนของ PROGRAM PATTERN โดยการทดลองสร้างสัญญาณรูปแบบต่างๆ ที่ความถี่ต่างกัน แสดงดังรูป

FREQUENCY = 20 kHz

TIME/DIV = 10 uS

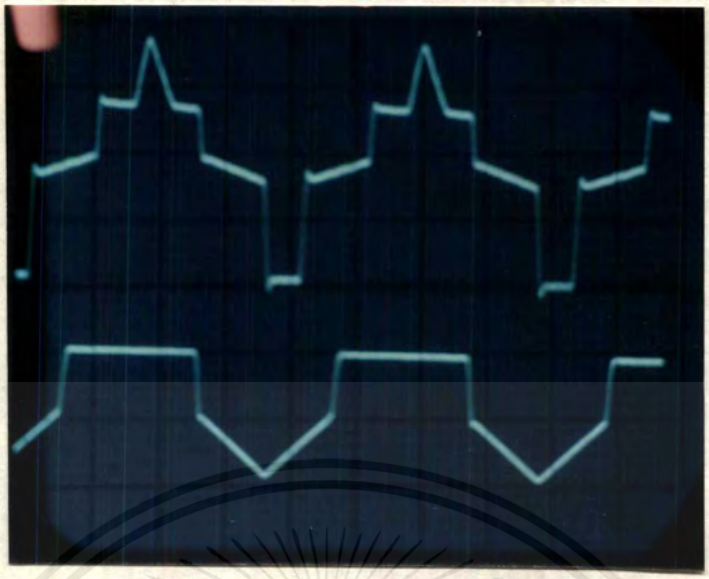
VOLTS/DIV = 1 Volt



FREQUENCY = 25 kHz

TIME/DIV = 20 uS

VOLTS/DIV = 1 Volt

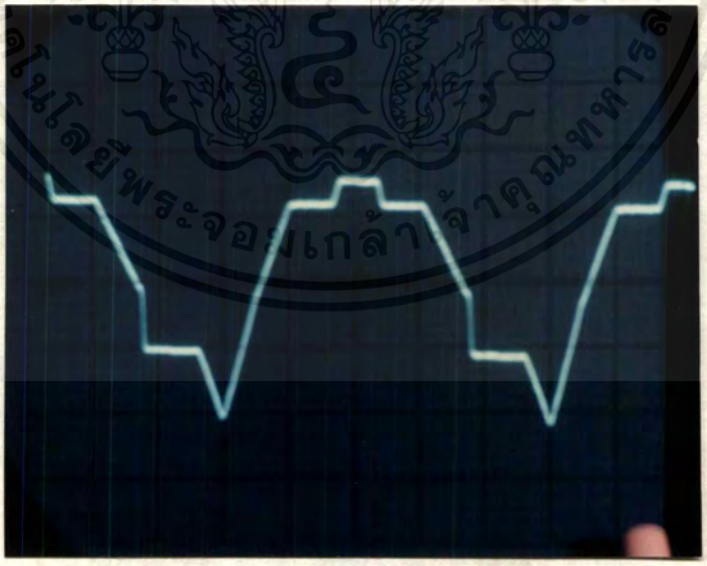


การทดลองการทำงานของโปรแกรม ในส่วนของ PROGRAM PATTERN โดยการทดลองสร้างสัญญาณรูปแบบต่างๆ ที่ความถี่ต่างกัน แสดงดังรูป

FREQUENCY = 25 kHz

TIME/DIV = 10 uS

VOLTS/DIV = 1 Volt



FREQUENCY = 40 kHz

TIME/DIV = 5 uS

VOLTS/DIV = 1 Volt

บทที่ 5
สรุปและวิจารณ์

จากการทดลองสร้างเครื่องกำเนิดสัญญาณชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลข ในโครงงานนี้ได้สร้างเครื่องกำเนิดสัญญาณ SINEWAVE SQUAREWAVE TRIANGLEWAVE SAWTOOTH และสัญญาณอะไรก็ได้ตามที่ต้องการ สามารถทำการออกแบบสัญญาณได้โดยการใช้โปรแกรมคอมพิวเตอร์ในการออกแบบสัญญาณเครื่องกำเนิดสัญญาณชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลขนี้ ได้สร้างสัญญาณรูปแบบต่างๆที่ความถี่ต่างๆกัน โดยเน้นเรื่องความถูกต้องของความถี่เป็นหลักโดยใช้เครื่องคอมพิวเตอร์คำณวณหารูปแบบของสัญญาณและใช้ไมโครคอลโทรลเลอร์เข้ามาควบคุมการสร้างสัญญาณออกมา ทำให้มีความคล่องตัวในการใช้งานมาก และจากการศึกษาค้นคว้าในโครงงานนี้ ก็พอสรุปข้อดีและข้อเสียของเครื่องกำเนิดสัญญาณชนิดที่โปรแกรมได้โดยการสังเคราะห์โดยตรงทางเชิงเลขนี้ ได้ดังต่อไปนี้

ข้อดี

1. สามารถออกแบบสัญญาณที่ต้องสร้างได้โดยง่าย เนื่องมาจากว่าการออกแบบสัญญาณรูปแบบต่างๆเราทำการเปลี่ยนแปลงเฉพาะทางด้าน SOLFWARE เท่านั้น ส่วนวงจรกำเนิดสัญญาณนั้น หรือ HARDWARE นั้นยังคงเดิมอยู่เช่นถ้าเราต้องการเปลี่ยนแปลงจากสัญญาณ SINEWAVE มาอยู่ในรูปสัญญาณ SQUAREWAVE หรือสัญญาณ SAWTOOTH เราก็ทำการเปลี่ยนชุดข้อมูลของสัญญาณไปเท่านั้น ไม่จำเป็นต้องเปลี่ยนแปลงส่วนของอุปกรณ์ที่ใช้ในการผลิตสัญญาณแต่อย่างใด ดังนั้นจึงเป็นการง่ายต่อการออกแบบสัญญาณที่ผลิตขึ้นมามาก

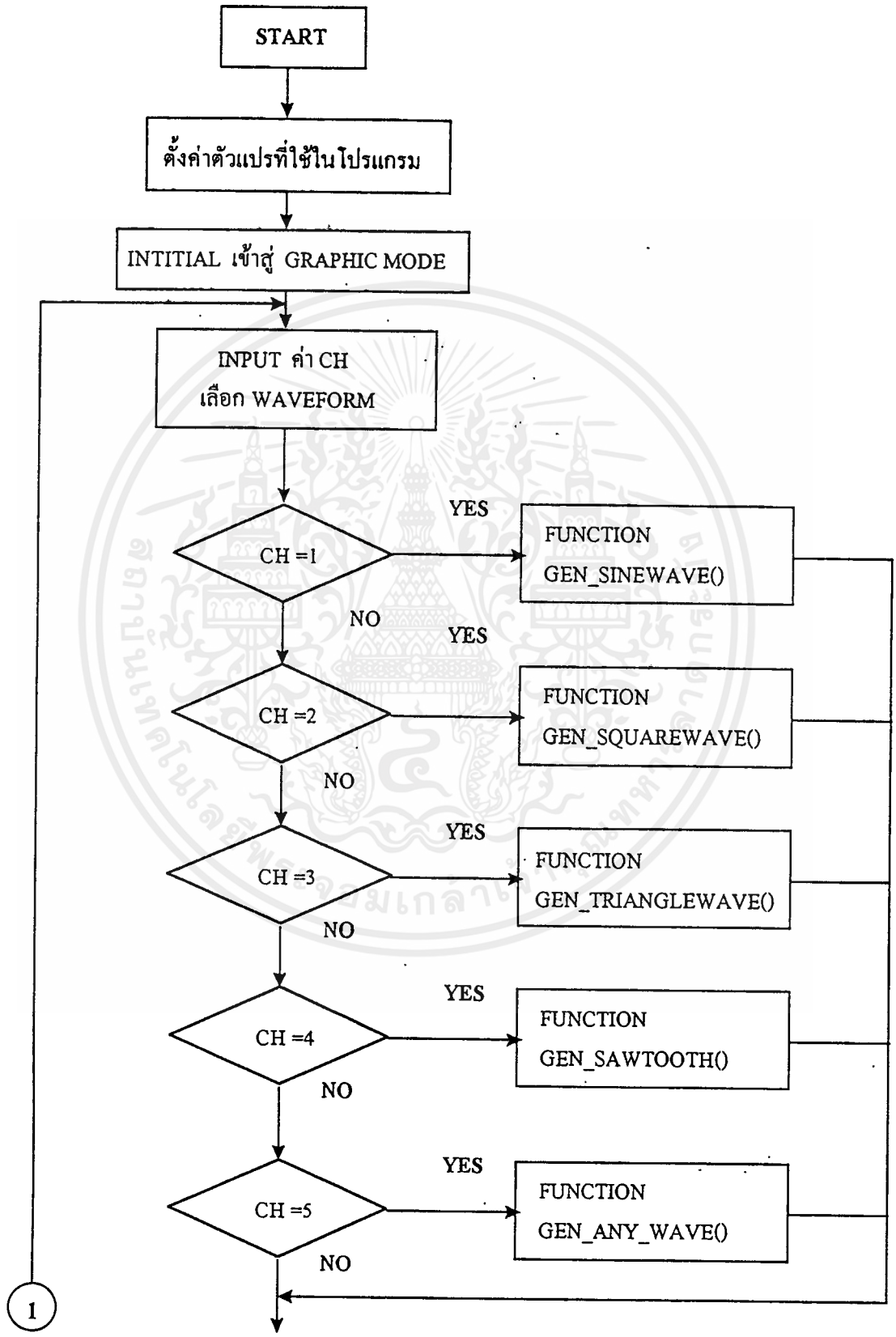
2. สามารถผลิตสัญญาณรูปแบบบางอย่างได้ โดยสัญญาณบางสัญญาณที่เราต้องการผลิตขึ้นมานั้น บางสัญญาณทำได้ยากมาก หรืออาจทำไม่ได้เลยแต่เครื่องกำเนิดสัญญาณนี้สามารถทำการสร้างได้ เนื่องจากเป็นการสร้างสัญญาณจากข้อมูล Didital

ข้อเสีย

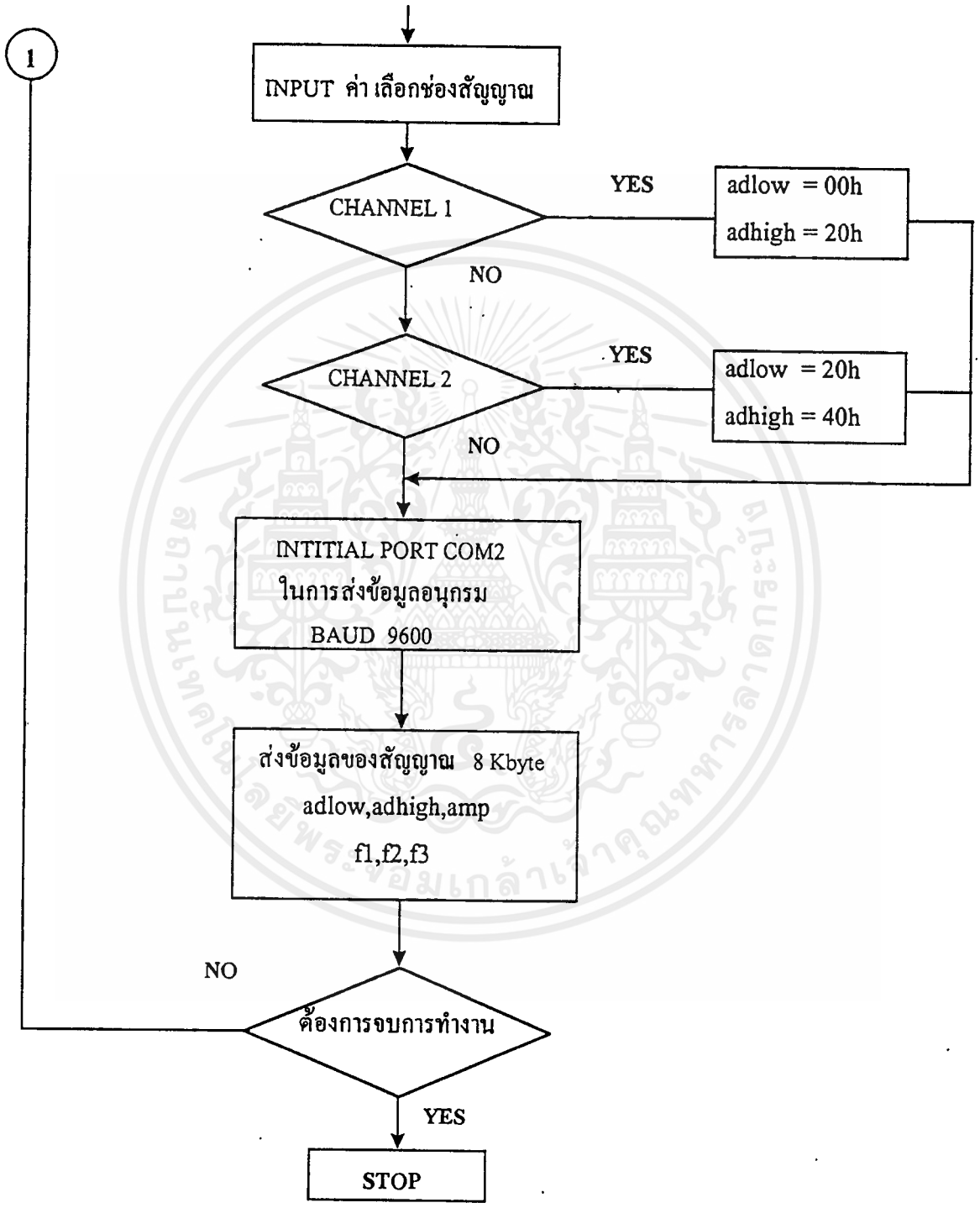
1.ข้อจำกัดทางด้านความถี่ สามารถสร้างความถี่ได้ไม่สูงนัก เพราะที่ความถี่สูงในหนึ่งรูปคลื่นจะมีการSampling น้อยครั้งเมื่อเทียบกับที่ความถี่ต่ำๆ เพราะฉะนั้นเมื่อความถี่สูงๆ รูปร่างของสัญญาณจะมีความเพี้ยนมาก

ภาคผนวก

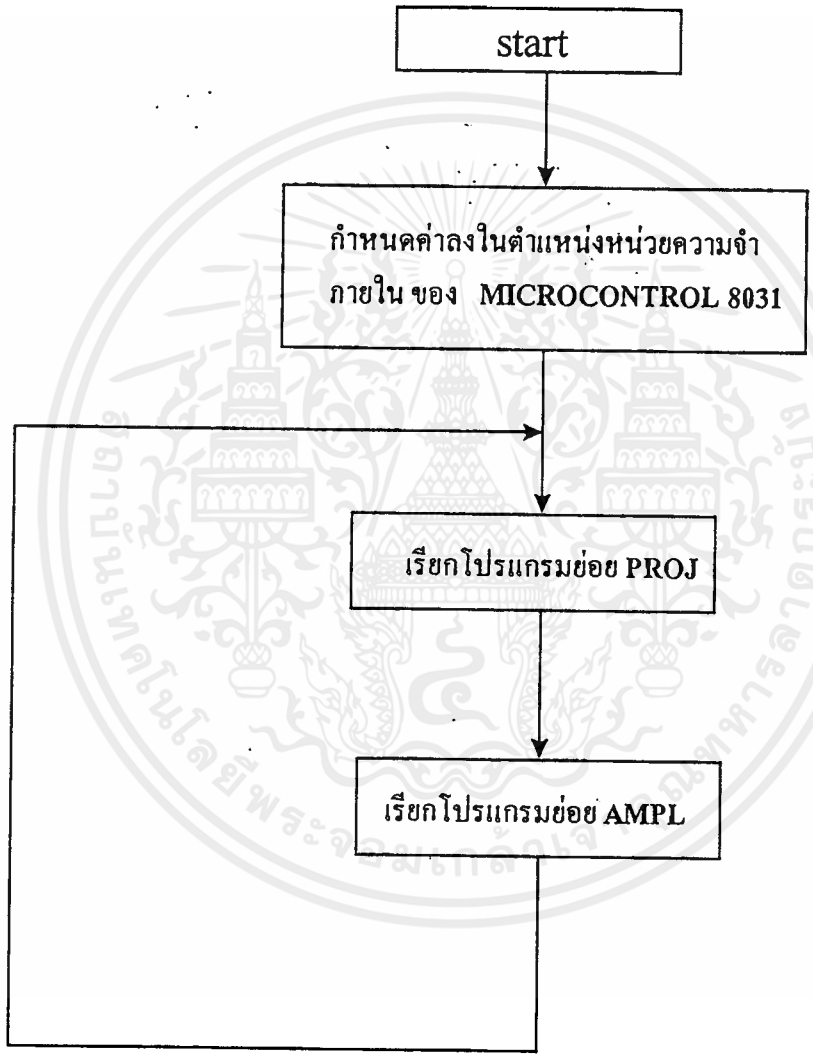
; MAIN PROGRAM ภาษา C



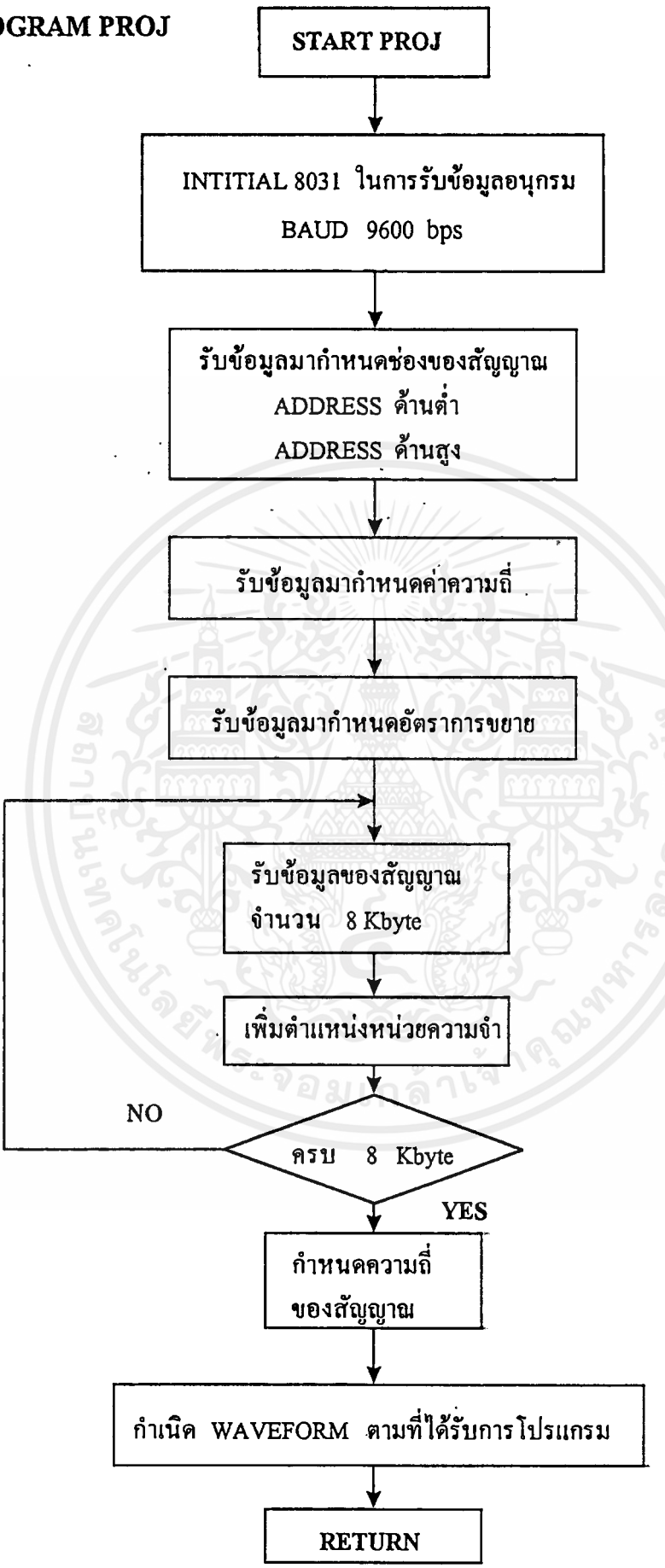
; MAIN PROGRAM ภาษา C (ต่อ)



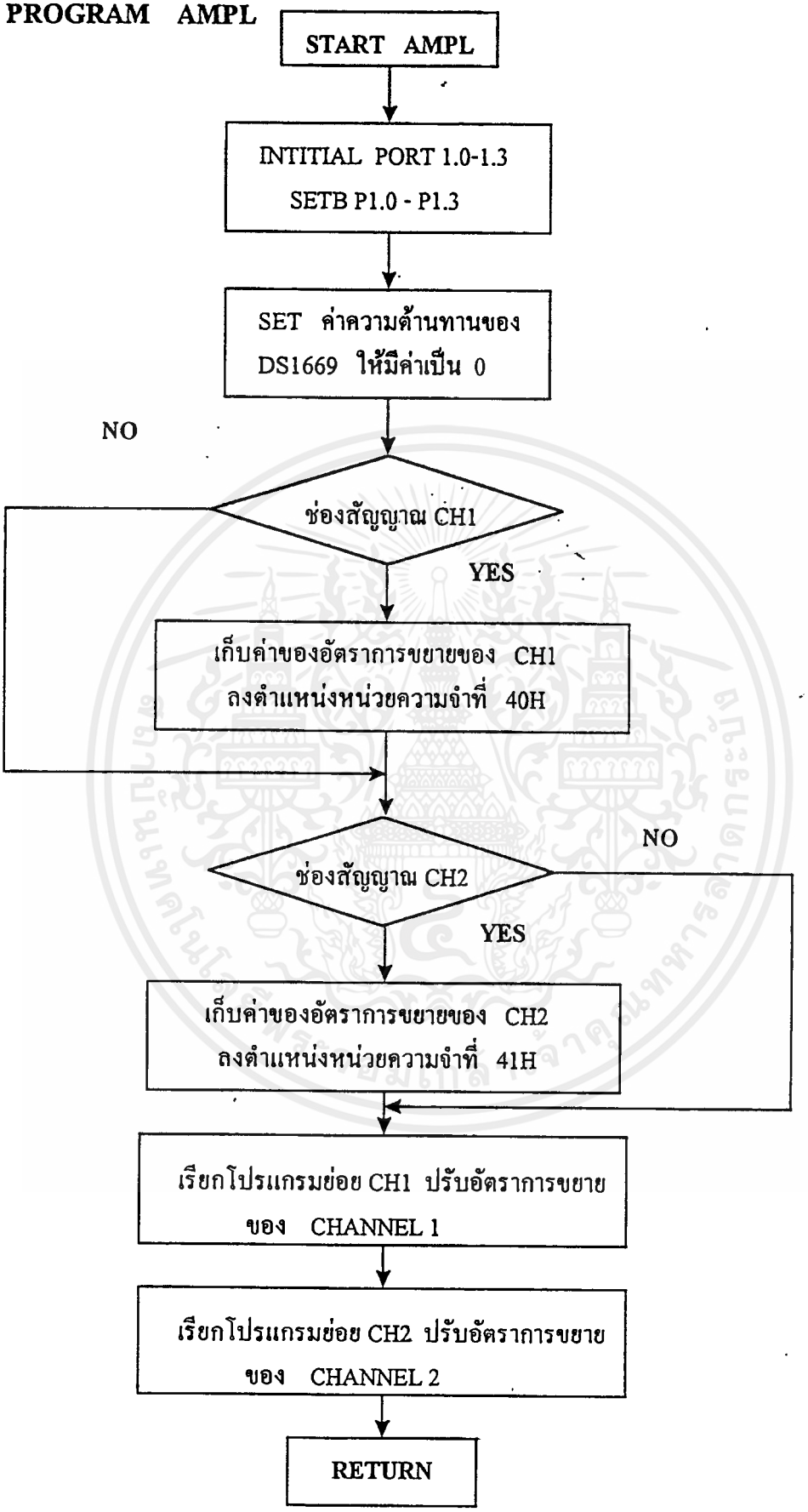
; main program 8031



; SUB PROGRAM PROJ



; SUB PROGRAM AMPL



โปรแกรมภาษา C

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

/*status of mouse botton*/


#define PORT 1
#define MOUSE_LEFT 1
#define MOUSE_MID  4
#define MOUSE_RIGHT 2


struct MOUSETYPE{
       int x;
       int y;
       int status;
       };

unsigned char mousecursor[16][8]={{1,0,0,0,0,0,0,0,},
                                  {1,1,0,0,0,0,0,0,},
                                  {1,15,1,0,0,0,0,0,},
                                  {1,15,15,1,0,0,0,0,},
                                  {1,15,15,15,1,0,0,0,},
                                  {1,15,15,15,1,1,0,0,},
                                  {1,15,15,15,15,1,1,0,},
                                  {1,15,15,15,1,0,0,0,},
                                  {1,15,15,15,1,0,0,0,},
                                  {1,15,1,15,1,0,0,0,},
                                  {1,1,0,1,15,1,0,0,},
                                  {1,0,0,1,15,1,0,0,},
                                  {0,0,0,0,1,15,1,0,},
                                  {0,0,0,0,1,15,1,0,},
                                  {0,0,0,0,0,1,15,1,},
                                  {0,0,0,0,0,1,15,1,}};

unsigned char mousebackground[2048];
unsigned char s[8192];
unsigned char adlow,adhigh;
unsigned char f1,f2,f3,amp;


int MResetFlag=0,MDispFlag=0,
     MouseOLDX=0,MouseOLDY=0;

/*********  Functions ************/

int mouse_installed(void);
```

```
int mouse_uninstalled(void);
void mouse_display(void);
void mouse_hide(void);
void table(void);
void table1(void);
void setregion(int x1,int y1,int x2,int y2);
struct MOUSETYPE getmouse(void);
void setmouse(int x,int y);

void interrupt movemouse1(void);
void interrupt movemouse2(void);
void interrupt (*old0x1c)(void);
void gen_any_wave(void);
void sport(),port_init();
void gen_sinewave(void);
void out_wave(void);
void gen_trianglewave(void);
void gen_sawtooth(void);
void gen_squarewave(void);
void frequency(void);
void freq_amp(void);
void cur_off(void);


 /*  main program */

void main(void)
{
   int gdriver= DETECT,gmode;
   struct MOUSETYPE m;



   int ch,f,u,t;
   int add;

   clrscr();
   cur_off();
   delay(300);

for(;;)
{

 /*  To graphic mode */

   initgraph(&gdriver,&gmode,"");
   if(!mouse_installed())
       {
      closegraph();
      printf("error resetting mouse\n");
      exit(1);
      }
   setmouse(500,200);
   setregion(0,0,630,460);
   mouse_display();



   delay(100);
   setbkcolor(0);
   setcolor(15);
   rectangle(170,45,470,95);
   setfillstyle(1,1);
   floodfill(300,80,15);
```

```
    setcolor(15);
    outtextxy(226,68,"PLEASE SELECT WAVEFORM");
    outtextxy(390,450,"       Click MOUSE_LEFT ");
    outtextxy(390,465,"       SELECT and RUN continue");
    setcolor(15);
    rectangle(210,110,410,145);
    setfillstyle(1,15);
    floodfill(240,135,15);
    setcolor(4);
    rectangle(220,120,420,155);
    setfillstyle(1,4);
    floodfill(230,130,4);
    setcolor(15);
    outtextxy(275,135,"SINE WAVE ");


    setcolor(15);
    rectangle(210,170,410,205);
    setfillstyle(1,15);
    floodfill(240,195,15);
    setcolor(4);
    rectangle(220,180,420,215);
    setfillstyle(1,4);
    floodfill(230,190,4);
    setcolor(15);
    outtextxy(270,195,"SQUARE WAVE");


    setcolor(15);
    rectangle(210,230,410,265);
    setfillstyle(1,15);
    floodfill(240,255,15);
    setcolor(4);
    rectangle(220,240,420,275);
    setfillstyle(1,4);
    floodfill(230,250,4);
    setcolor(15);
    outtextxy(265,255,"TRIANGLE WAVE");


    setcolor(15);
    rectangle(210,290,410,325);
    setfillstyle(1,15);
    floodfill(240,315,15);
    setcolor(4);
    rectangle(220,300,420,335);
    setfillstyle(1,4);
    floodfill(230,320,4);
    setcolor(15);
    outtextxy(280,315,"SAWTOOTH");


    setcolor(15);
    rectangle(210,350,410,385);
    setfillstyle(1,15);
    floodfill(240,375,15);
    setcolor(4);
    rectangle(220,360,420,395);
    setfillstyle(1,4);
    floodfill(240,390,4);
    setcolor(15);
    outtextxy(255,375,"PROGRAM PATTERN ");
    delay(300);
```

```
for(;;)
{
   do
   {

     m=getmouse();
     }
    while(m.status!=MOUSE_LEFT);

        if((m.x>=220)&&(m.x<=415)&&(m.y>=120)&&(m.y<=149))
        {
        setbkcolor(0);


        setcolor(0);
        rectangle(210,110,410,145);
        setfillstyle(1,0);
        floodfill(225,125,0);

        setcolor(15);
        rectangle(230,130,430,165);
        setfillstyle(1,0);
        floodfill(250,155,15);

        setbkcolor(0);
        setcolor(4);
        rectangle(220,120,420,155);
        setfillstyle(1,4);
        floodfill(265,125,4);
        setcolor(15);
        outtextxy(275,135,"SINE WAVE ");
        ch=1;
        break;
        }
       else
       if((m.x>=220)&&(m.x<=415)&&(m.y>=180)&&(m.y<=209))
        {
        setbkcolor(0);


        setcolor(0);
        rectangle(210,170,410,205);
        setfillstyle(1,0);
        floodfill(225,195,0);

        setcolor(15);
        rectangle(230,190,430,225);
        setfillstyle(1,0);
        floodfill(250,215,15);

        setbkcolor(0);
        setcolor(4);
        rectangle(220,180,420,215);
        setfillstyle(1,4);
        floodfill(265,195,4);
        setcolor(15);
        outtextxy(275,195,"SQUARE WAVE ");
        ch=2;
        break;
        }
         else
       if((m.x>=220)&&(m.x<=415)&&(m.y>=240)&&(m.y<=269))
        {
```

```
        setbkcolor(0);


        setcolor(0);
        rectangle(210,230,410,265);
        setfillstyle(1,0);
        floodfill(225,245,0);

        setcolor(15);
        rectangle(230,250,430,285);
        setfillstyle(1,0);
        floodfill(250,265,15);

        setbkcolor(0);
        setcolor(4);
        rectangle(220,240,420,275);
        setfillstyle(1,4);
        floodfill(265,255,4);
        setcolor(15);
        outtextxy(265,255,"TRIANGLE WAVE ");
        ch=3;
        break;
        }
         else
       if((m.x>=220)&&(m.x<=415)&&(m.y>=300)&&(m.y<=329))
        {
        setbkcolor(0);


        setcolor(0);
        rectangle(210,290,410,325);
        setfillstyle(1,0);
        floodfill(225,305,0);

        setcolor(15);
        rectangle(230,310,430,345);
        setfillstyle(1,0);
        floodfill(250,325,15);

        setbkcolor(0);
        setcolor(4);
        rectangle(220,300,420,335);
        setfillstyle(1,4);
        floodfill(265,315,4);
        setcolor(15);
        outtextxy(280,315,"SAWTOOTH  ");
        ch=4;
        break;
        }
         else
       if((m.x>=220)&&(m.x<=415)&&(m.y>=360)&&(m.y<=389))
        {
        setbkcolor(0);


        setcolor(0);
        rectangle(210,350,410,385);
        setfillstyle(1,0);
        floodfill(225,365,0);

        setcolor(15);
        rectangle(230,370,430,405);
        setfillstyle(1,0);
        floodfill(250,385,15);
```

```
        setbkcolor(0);
        setcolor(4);
        rectangle(220,360,420,395);
        setfillstyle(1,4);
        floodfill(265,375,4);
        setcolor(15);
        outtextxy(255,375,"PROGRAM PATTERN ");
        ch=5;
        break;
        }

  }
    setmouse(500,400);
    delay(200);
    do
    {
     m=getmouse();
     }
   while(m.status!=MOUSE_LEFT);
   closegraph();
   cur_off();


/*      case of "ch" to select wave form    */

    switch(ch)
    {


     case 1:gen_sinewave();
          initgraph(&gdriver,&gmode,"");
          table1();
          setregion(0,0,630,460);
          setmouse(600,400);
          setcolor(9);outtextxy(190,60,"SINE WAVE 1 PERIOD");
          outtextxy(450,450,"Click mouse to continue");
          setbkcolor(0);
          f=0;
          t=0;
         for(u=0;u<=8191;u++)
         {
          f++;
          if(f==32)
           {
          putpixel(192+t,367-(int)s[u],4);
          t++;
          f=0;
            }
          }
         delay(200);
         do
         {
          m=getmouse();
         }
         while(m.status!=MOUSE_LEFT);
          mouse_hide();
          mouse_uninstalled();
          closegraph();
          cur_off();
          delay(200);
          initgraph(&gdriver,&gmode,"");
```

```
        frequency();
        closegraph();
        cur_off();
        break;


    case 2:gen_squarewave();
        initgraph(&gdriver,&gmode,"");
        table1();
        setregion(0,0,630,460);
        setmouse(600,400);
        setcolor(9);outtextxy(190,60,"SQUARE WAVE 1 PERIOD");
        outtextxy(450,450,"Click mouse to continue");
        setbkcolor(0);
        f=0;
        t=0;
       for(u=0;u<=8191;u++)
       {
        f++;
        if(f==32)
         {
        putpixel(192+t,367-(int)s[u],4);
        t++;
        f=0;
          }
      }
       delay(200);
       do
       {
        m=getmouse();
       }
       while(m.status!=MOUSE_LEFT);
        mouse_hide();
        mouse_uninstalled();
        closegraph();
        cur_off();
        initgraph(&gdriver,&gmode,"");
        frequency();
        closegraph();
        cur_off();
        break;


   case 3:gen_trianglewave();
        initgraph(&gdriver,&gmode,"");
        table1();
        setregion(0,0,630,460);
        setmouse(600,400);
        setcolor(9);outtextxy(190,60,"TRIANGLE WAVE 1 PERIOD");
        outtextxy(450,450,"Click mouse to continue");
        setbkcolor(0);
        f=0;
        t=0;
       for(u=0;u<=8191;u++)
       {
        f++;
        if(f==32)
         {
        putpixel(192+t,367-(int)s[u],4);
        t++;
        f=0;
          }
       }
       delay(200);
```

```
        do
       {
        m=getmouse();
       }
        while(m.status!=MOUSE_LEFT);

        mouse_hide();
        mouse_uninstalled();

        closegraph();
        cur_off();
        initgraph(&gdriver,&gmode,"");
        frequency();
        closegraph();
        cur_off();
        break;


    case 4:gen_sawtooth();
        initgraph(&gdriver,&gmode,"");
        table1();
        setregion(0,0,630,460);
        setmouse(600,400);
        setcolor(9);outtextxy(190,60,"SAWTOOTH 1 PERIOD");
        outtextxy(450,450,"Click mouse to continue");
        setbkcolor(0);
        f=0;
        t=0;
       for(u=0;u<=8191;u++)
       {
        f++;
        if(f==32)
         {
        putpixel(192+t,367-(int)s[u],4);
        t++;
        f=0;
          }
       }
        delay(200);
        do
       {
        m=getmouse();
       }
        while(m.status!=MOUSE_LEFT);
        mouse_hide();
        mouse_uninstalled();
        closegraph();
        cur_off();
        initgraph(&gdriver,&gmode,"");
        frequency();
        closegraph();
        cur_off();
        break;


    case 5:initgraph(&gdriver,&gmode,"");
        gen_any_wave();
        closegraph();
        cur_off();
        initgraph(&gdriver,&gmode,"");
        frequency();
        closegraph();
        cur_off();
    }
```

```
    delay(200);
    clrscr();
    cur_off();

  /*   to select channel    */
  /*   To graphic mode      */

    initgraph(&gdriver,&gmode,"");
    if(!mouse_installed())
        {
       closegraph();
       printf("error resetting mouse\n");
       exit(1);
       }

    setmouse(500,200);
    setregion(0,0,630,460);
    mouse_display();

    delay(100);
    setbkcolor(0);
    setcolor(15);
    rectangle(170,45,470,95);
    setfillstyle(1,4);
    floodfill(300,80,15);
    setcolor(15);
    outtextxy(226,68,"PLEASE SELECT CHANNEL");
    outtextxy(360,450,"     Click MOUSE_LEFT ");
    outtextxy(360,465,"     To Sending pattern waveform");
    setcolor(15);
    rectangle(210,110,410,145);
    setfillstyle(1,15);
    floodfill(240,135,15);
    setcolor(1);
    rectangle(220,120,420,155);
    setfillstyle(1,1);
    floodfill(230,130,1);
    setcolor(15);
    outtextxy(275,135,"CHANNEL 1 ");


    setcolor(15);
    rectangle(210,170,410,205);
    setfillstyle(1,15);
    floodfill(240,195,15);
    setcolor(1);
    rectangle(220,180,420,215);
    setfillstyle(1,1);
    floodfill(230,190,1);
    setcolor(15);
    outtextxy(275,195,"CHANNEL 2");

    delay(300);
for(;;)
 {
    do
    {

      m=getmouse();
      }
     while(m.status!=MOUSE_LEFT);

        if((m.x>=220)&&(m.x<=415)&&(m.y>=120)&&(m.y<=149))
```

```
        {
        setbkcolor(0);


        setcolor(0);
        rectangle(210,110,410,145);
        setfillstyle(1,0);
        floodfill(225,125,0);

        setcolor(15);
        rectangle(230,130,430,165);
        setfillstyle(1,0);
        floodfill(250,155,15);

        setbkcolor(0);
        setcolor(1);
        rectangle(220,120,420,155);
        setfillstyle(1,1);
        floodfill(265,125,1);
        setcolor(15);
        outtextxy(275,135,"CHANNEL 1 ");
        adlow=0x00;
        adhigh=0x20;
        break;
        }
       else
       if((m.x>=220)&&(m.x<=415)&&(m.y>=180)&&(m.y<=209))
        {
        setbkcolor(0);


        setcolor(0);
        rectangle(210,170,410,205);
        setfillstyle(1,0);
        floodfill(225,195,0);

        setcolor(15);
        rectangle(230,190,430,225);
        setfillstyle(1,0);
        floodfill(250,215,15);

        setbkcolor(0);
        setcolor(1);
        rectangle(220,180,420,215);
        setfillstyle(1,1);
        floodfill(265,195,1);
        setcolor(15);
        outtextxy(275,195,"CHANNEL 2 ");
        adlow=0x20;
        adhigh=0x40;
        break;
        }

 }
   setmouse(500,300);
   delay(200);
   do
   {
    m=getmouse();
    }
  while(m.status!=MOUSE_LEFT);

/*   sending pattern waveform  */
```

```
    closegraph();
    cur_off();
    initgraph(&gdriver,&gmode,"");
    setmouse(500,300);
    setregion(0,0,630,460);
    setcolor(7);
    rectangle(170,200,470,250);
    setfillstyle(1,4);
    floodfill(450,240,7);
    setcolor(15);
    outtextxy(250,220,"Sending pattern...");

    out_wave();

    setcolor(7);
    rectangle(170,200,470,250);
    setfillstyle(1,1);
    floodfill(450,240,7);
    setcolor(15);
    outtextxy(250,220, "  End of Sending...");
    outtextxy(400,430,"click MOUSE_LEFT continue");
    outtextxy(400,450,"      MOUSE_RIGHT to exit ");

    delay(200);
    do
    {
     m=getmouse();
     }
    while(!m.status);
    if(m.status==MOUSE_RIGHT) break;


    mouse_hide();
    mouse_uninstalled();
    closegraph();
   }

  /*  if out of break */

    mouse_hide();
    mouse_uninstalled();
    closegraph();

}


 /*  frequency and amplitude*/


void frequency(void)
{
  unsigned long int i,h,k,dataf,total,t1;
  char ch[6],w[6],w1[6],ha,s;
  float p1,amp1,amp2,total1,Vout;
  float Vin,Rin,Rf,step,resis1;
  int a,d,b,v,data[9],data1[3],t2,p2;
  int a1,b1,d1,resis2;
  for(i=0;i<=28;i++)
 {
  ch[i]=' ';
 }

  setbkcolor(0);
  a=10;
```

```
   b=0;
   setcolor(15);
   outtextxy(380,450,"  Presss any key to continue");
   setcolor(4);
   outtextxy(100,200,"FREQUENCY =                                                    Hz");
   d=246;

/*freaquency*/

  for(;;)
  {
  ch[6]=getch();
  if((ch[6]=='0'||ch[6]=='1'||ch[6]=='2'||ch[6]=='3'||ch[6]==0x0d||
  ch[6]==0x08||ch[6]=='4'||ch[6]=='5'||ch[6]=='6'||ch[6]=='7'||
  ch[6]=='8'||ch[6]=='9'||ch[6]=='.'||ch[6]==0x20))
  {
  setcolor(15);
  if(ch[6]== 0x0d||(b>=6))break;
  if(ch[6]== 0x08)
  {
  a=a-8;
  b=b-1;
  setcolor(0);
  d=246+a;
  if(d>254)
  {
  outtextxy(d,200,"Û");
  a=a-8;
  b=b-1;
  setcolor(15);

  }
   }
  else
{
  outtextxy(198+a,200,ch);
  setcolor(0);
  outtextxy(406+a,200,"Û");
  setcolor(15);
  w[b]=ch[6];
  }
  b=b+1;
  a=a+8;
   }
  }

/*amplitude*/

  setcolor(4);
  outtextxy(100,240,"AMPLITUDE =                                                    Volt");
  setcolor(15);
  outtextxy(504,240,"<Vmax 4 V>");

 d1=240;
 a1=10;
 b1=0;

  for(;;)
  {
  ch[6]=getch();
  if((ch[6]=='0'||ch[6]=='1'||ch[6]=='2'||ch[6]=='3'||ch[6]==0x0d||
  ch[6]==0x08||ch[6]=='4'||ch[6]=='5'||ch[6]=='6'||ch[6]=='7'||
  ch[6]=='8'||ch[6]=='9'||ch[6]=='.'||ch[6]==0x20))
  {
```

```
 setcolor(15);
 if(ch[6]== 0x0d||(b1>=4))break;
 if(ch[6]== 0x08)
 {
 a1=a1-8;
 b1=b1-1;
 setcolor(0);
 d1=246+a1;
 if(d1>254)
 {
 outtextxy(d1,240,"Û");
 a1=a1-8;
 b1=b1-1;
 setcolor(15);
 }
  }
 else
{
 if(b1==1)
{
outtextxy(246+a1,240,".");
}
else
 {
 outtextxy(198+a1,240,ch);
 setcolor(0);
 outtextxy(406+a1,240,"Û");
 setcolor(15);
 w1[b1]=ch[6];
 }
 }
 b1=b1+1;
 a1=a1+8;
  }
 }


/* out frequency*/
for(v=0;v<=b-1;v++)
 {

 switch(w[v])
 {
   case '0':data[v]=0;break;
   case '1':data[v]=1;break;
   case '2':data[v]=2;break;
   case '3':data[v]=3;break;
   case '4':data[v]=4;break;
   case '5':data[v]=5;break;
   case '6':data[v]=6;break;
   case '7':data[v]=7;break;
   case '8':data[v]=8;break;
   case '9':data[v]=9;break;

 }

 }


 t2=b-1;
 dataf=0;
 total=0;
```

```
for(v=0;v<=b-1;v++)
{
 t1=pow(10,(unsigned long int)t2);
 total=(unsigned long int )data[v]*t1;
 dataf=dataf+total;
 t2=t2-1;
}


h=dataf/256;
k=h/256;
f3=k;
f2=h;
f1=dataf;


/* out amplitude */

for(v=0;v<=b1-1;v++)
{

switch(w1[v])
{
  case '0':data1[v]=0;break;
  case '1':data1[v]=1;break;
  case '2':data1[v]=2;break;
  case '3':data1[v]=3;break;
  case '4':data1[v]=4;break;
  case '5':data1[v]=5;break;
  case '6':data1[v]=6;break;
  case '7':data1[v]=7;break;
  case '8':data1[v]=8;break;
  case '9':data1[v]=9;break;


}

}
 Vout=0;
 total1=0;
 amp2=0;
 p2=b1-3;
 amp1=(float)data1[0];
 for(v=2;v<=b1-1;v++)
 {
   p1=pow(10,-(v-1));
   total1=(float)data1[v]*p1;
   amp2=amp2+total1;

 }
 Vout=amp1+amp2;
 if(Vout>=4)Vout=4;

 Vin=1;
 Rf=25000;
 step=45000/63;
 Rin=Vin*Rf/Vout;
 resis1=Rin/step;
 resis2=(int)resis1;
 if(resis1- (float)resis2>=0.5)
 resis2++;
 amp=resis2+1;
}
```

```
/********* generated any waveform*************/

void gen_any_wave(void)
 {
   int x,y,i,j,x3,y3,x4,y4;
   int px,py,p,point[400][50];
   int point_color;
   int addr,loop;
   double data1;
   int data2,ad1,ad2;
   int gdriver = DETECT,gmode;
   struct MOUSETYPE m;

    table();
    setcolor(7);
    rectangle(20,20,80,50);
    setfillstyle(1,4);
    floodfill(25,25,7);
    setcolor(15);
    outtextxy( 36,31,"EXIT");
    setmouse(600,440);
    mouse_installed();
    setmouse(500,460);
    setregion(0,0,630,480);
    mouse_display();

   /* Generate fuction waveform */
   /* set origin of cursor of line*/
    x3=192;
    y3=239;
    loop=0;
    do
      {
       setbkcolor(0);
       m =getmouse();

       if(m.status==MOUSE_RIGHT)
       {
        x3=192;
        y3=m.y;
        }
       }
       while(m.status!=MOUSE_LEFT);
       if((m.x>=192)&&(m.x<=447)&&(m.y>=90)&&(m.y<=389))
      {
       setcolor(14);
       line(x3,y3,m.x,m.y);
      }
      else
      {
       line(x3,y3,x3,y3);
       }
       x4=x3;
       y4=y3;

    for(;;)
    {

    do
      {
       setbkcolor(0);
       m =getmouse();
```

```
      }
       while(m.status!=MOUSE_LEFT);

       if((m.x>20)&&(m.x<80)&&(m.y>20)&&(m.y<50))break;
       else
       if((m.x>=192)&&(m.x<=447)&&(m.y>=90)&&(m.y<=389))

            {
           if((m.x>=x3))
            {
             setbkcolor(0);
             setcolor(14);
             line(x3,y3,m.x,m.y);
             delay(250);
             line(x3,y3,x4,y4);
             y4=y3;
             x4=x3;
             y3=m.y;
             x3=m.x;
            }
           }
       else
       if((m.x>=447)&&(m.x>x3)&&(m.y<=389)&&(m.y>=90))
       {
         if(loop==0)
         {
           setcolor(14);
           line(x3,y3,447,m.y);
           delay(250);
           line(x3,y3,x4,y4);
           loop++;
           x3=m.x;
           }

        }

            else

             {
            line(x4,y4,x4,y4);
             }
        }

      setmouse(400,600);
      delay(100);
    setcolor(7);
    rectangle(20,20,80,50);
    setfillstyle(1,4);
    floodfill(25,25,7);
    setcolor(15);
    outtextxy( 36,31,"EXIT");

/*****************************************************************/
/* get data of waveform*/

       for(px=0;px<=255;px++)
       {
         p=0;
         for(py=90;py<=389;py++)
          {
          point_color= getpixel(192+px,py);
          if((point_color==14))
          {
          point[px][p]=py-89;          /*out of phase 389-py 180 */
```

```
            p++;
            }
            }
          }
        mouse_hide();
        mouse_uninstalled();
        delay(200);
        ad2=0;
        for(px=0;px<=255;px++)
        {
          data1=(double)point[px][0]*255/299;
          data2=(int)data1;
          if(data1-(double)data2>=0.5)
           {
           data2++;
           }
          for(ad1=0;ad1<=31;ad1++)
          {
            s[ad2]=data2;
            ad2++;
            }
          }
}


/*  function  copy wave  to mcs_51*/
void out_wave(void)
  {
       int addr;
       port_init(PORT,227);
       sport(PORT,adlow);
       sport(PORT,adhigh);
       sport(PORT,f1);
       sport(PORT,f2);
       sport(PORT,f3);
       sport(PORT,amp);
       for(addr=0;addr<=8191;addr++)
       {
         sport(PORT,s[addr]);
       }


  }


/************************************/


int mouse_installed(void)
{
   /*reset mouse driver*/
   _AX = 0;
   geninterrupt(0x33);
   if(_AX==0) return 0;
   if(!MResetFlag)
   {
     old0x1c=getvect(0x1c);
     setvect(0x1c,movemouse1);
     MResetFlag =1;
     }
     else
```

```
      return 0;
      getimage(0,0,16,8,mousebackground);
      return(1);
      }


int mouse_uninstalled(void)
  {
    if(MResetFlag)/*check interrupt installed*/
    {
     setvect(0x1c,old0x1c);
     MResetFlag=0;
     return 1;
     }
     else
     return 0;
     }

void mouse_display(void)
  {
    MDispFlag =1;
    }

void mouse_hide(void)
  {
    MDispFlag =0;
    }

void setregion(int x1,int y1,int x2,int y2)
 {
  _CX =x1;
  _DX =x2;
  _AX =0x07;
  geninterrupt(0x33);
  _CX =y1;
  _DX =y2;
  _AX =0x08;
  geninterrupt(0x33);
  }

struct MOUSETYPE getmouse(void)
 {
  struct MOUSETYPE m;
   _AX =0x03;
   geninterrupt(0x33);
   m.status = _BX;
   m.x      = _CX;
   m.y      = _DX;
   return m;
   }

void setmouse(int x,int y)
{
  _CX =x;
  _DX =y;
  _AX =0x04;
  geninterrupt(0x33);
  }


void interrupt movemouse1(void)
 {
  register int x,y,c;
```

```
    _AX =0x03;
    geninterrupt(0x33);
    x= _CX;
    y= _DX;

    if((x!=MouseOLDX)||(y!=MouseOLDY))
    {
     putimage(MouseOLDX,MouseOLDY,
            mousebackground,COPY_PUT);
      MouseOLDY=y;
      MouseOLDX=x;
      getimage(MouseOLDX,MouseOLDY,MouseOLDX+7,
            MouseOLDY+15,mousebackground);
      c=1;
      if(MDispFlag)
      for(y=0;y<16;y++)
        for(x=0;x<8;x++)
        if(c= mousecursor[y][x])
        putpixel(MouseOLDX+x,MouseOLDY+y,15);
      }
      old0x1c();

  }


void interrupt movemouse2(void)
 {
  register int x,y,c;
   _AX =0x03;
   geninterrupt(0x33);
   x= _CX;
   y= _DX;

   if((x!=MouseOLDX)||(y!=MouseOLDY))
   {
     MouseOLDY=y;
     MouseOLDX=x;
     c=4;
     if(MDispFlag)
     for(y=0;y<2;y++)
       for(x=0;x<2;x++)
       if(c!= mousecursor[y][x])
       putpixel(MouseOLDX+x,MouseOLDY+y,5);
     }
     old0x1c();

  }

 void table(void)
 {
   int i,k;
   setcolor(9);
   outtextxy(190,50,"Press and Move Mouse_Left");
   outtextxy(190,60,"Plot waveform  1 period");
   outtextxy(190,410,"Press Mouse_Right to point's start waveform");
   setcolor(15);
   rectangle(191,89,448,390);
   line(192,239,448,239);
   line(319,90,319,390);
   for(k=0;k<=240;k=k+30)
   {
   for(i=0;i<=256;i=i+4)
    {
```

```
     putpixel(191+i,119+k,15);
     }
    }
    for(k=0;k<=224;k=k+32)
    for(i=0;i<=300;i=i+4)
    {
     putpixel(191+k,90+i,15);
     }
    }


 /**********************************/


  void tablel(void)
  {
    int i,k;

    rectangle(191,89,448,390);
    line(192,239,448,239);
    line(319,90,319,390);
    for(k=0;k<=240;k=k+30)
    {
    for(i=0;i<=256;i=i+4)
     {
     putpixel(191+i,119+k,15);
     }
    }
    for(k=0;k<=224;k=k+32)
    for(i=0;i<=300;i=i+4)
    {
     putpixel(191+k,90+i,15);
     }
    }


/* Send data to MCS-51 */


void sport(port,c)
int port;
unsigned char c;
{
     union REGS r;
     r.x.dx = port;
     r.h.al = c;
     r.h.ah = 1;
     int86(0x14,&r,&r);

}


/* initial port to send data from COM2*/


void port_init(port,code)
int port;
unsigned char code;
{
  union REGS r;
  r.x.dx = port;
  r.h.ah = 0;
  r.h.al = code;
  int86(0x14,&r,&r);
```

```
}

/*  Function generated sinewave */

void  gen_sinewave()
 {
   double p=0;
   double S=0;
   double sin();
   double pi=3.141592654;
   int addr=0;
   int bytes=8192;
   clrscr();
   while(addr<8192)
   {
    p=2.0*pi*((double)addr)/((double)bytes);
    S=127.5*(1.0+sin(p));
    s[addr]=((int)S);
    if(S-((double)s[addr])>=0.5)
      s[addr]++;
      addr++;
      }
}

 /*Function generate triangle wave*/

void gen_trianglewave()
 {
   int i,b,data;
   data=128;
   clrscr();
   b=0;
   for(i=0;i<=2047;i++)
     {
       s[i]=data;
       b++;
       if(b==16)
       {
       data++;
       b=0;
       }
      }
      data=255;
   for(i=2048;i<=6143;i++)
     {
     s[i]=data;
     b++;
     if(b==16)
       {
      data--;
      b=0;
          }
      }
     data=0;
     for(i=6144;i<=8191;i++)
     {
      s[i]=data;
      b++;
      if(b==16)
```

```
        {
         data++;
         b=0;
         }
       }
  }

  /*Functiopn to generate sawtooth*/

 void gen_sawtooth()
 {
   int b,i,data;
   data=0;
   b=0;
   clrscr();
   for(i=0;i<=8191;i++)
    {
     s[i]=data;
     b++;
     if(b==32)
     {
       data++;
       b=0;
       }
     }
  }

 /*Function generate square wave*/
 void gen_squarewave()
 {
   int i,data;
   data= 255;
   for(i=0;i<=4095;i++)
   {
     s[i]=data;
     }
     data=0;
   for(i=4096;i<=8191;i++)
     {
      s[i]=data;
      }
 }

 /*  cursor off*/

 void cur_off(void)
  {
     union REGS regs;
     regs.h.ch=0x20;
     regs.h.cl=0;
     regs.h.ah=1;
     int86(0x10,&regs,&regs);
    }
```

โปรแกรมภาษา แอสเซมบลี

```
ORG 0000H

          ;*******  main program ************************

          MOV 40H,#00
          MOV 41H,#03
STATUS:   CALL PROJ
          CALL AMPL
          LJMP STATUS


          ;******** SET Boaud rate RECEIVE*****************

PROJ:     MOV PCON,#00H
          MOV SCON,#50H
          MOV TMOD,#20H
          MOV TH1,#0FDH
          SETB TR1



          ;******* Receive and Set start address of RAM*****


          MOV DPL,#00H
          LCALL RECI
          MOV DPH,A       ; receive adlow
          MOV 30H,A
          LCALL RECI
          MOV 31H,A       ; receive adhigh


          ;*******  Recieve frequency **********************


          LCALL RECI
          MOV 32H,A       ; receive f1
          LCALL RECI
          MOV 33H,A       ; receive f2
          LCALL RECI
          MOV 34H,A       ; receive f3


          ;********  Receive amplitude**********************

          LCALL RECI
          MOV 35H,A       ; receive amp


          ;*******  Receive data of wave form***************


          CLR P3.4
          CLR P3.5
```

```
START:   LCALL RECI
         MOVX @DPTR,A
         INC DPTR
         MOV A,DPH
         CJNE A,31H,START

        ; set frequency

         MOV DPTR,#0E000H
         MOV A,32H
         MOVX @DPTR,A

         MOV DPTR,#0E001H
         MOV A,33H
         MOVX @DPTR,A

         MOV DPTR,#0E002H
         MOV A,34H
         MOVX @DPTR,A

         SETB P3.4
         SETB P3.5

         ;TEST F1 F2 F3

         RET

RECI:    JNB RI,RECI
         CLR RI
         MOV A,SBUF
         MOV P1,A
         RET


   ; sub program to set amplitude

AMPL:
         MOV A,#11111111B
         MOV P1,A


         SETB P1.0
         SETB P1.1
         SETB P1.2
         SETB P1.3

         CLR P1.0
         CLR P1.2
         CALL DELAY
         SETB P1.0
         SETB P1.2

         MOV A,31H
         CJNE A,#20H,JEC1
         MOV A,35H
         MOV 40H,A

JEC1:    CJNE A,#40H,JEC2
         MOV A,35H
         MOV 41H,A

JEC2:
```

```
          CALL CH1
          CALL CH2
          RET

CH1:      MOV A,40H
          CALL DELAX
LOP1:     CLR P1.1
          CALL DELAX
          SETB P1.1
          CALL DELAX
          DEC A
          CJNE A,#0FFH,LOP1

          RET

CH2:      MOV A,41H
          CALL DELAX
LOP2:     CLR P1.3
          CALL DELAX
          SETB P1.3
          CALL DELAX
          DEC A
          CJNE A,#02H,LOP2

          RET


DELAY:    MOV R4,#35H
DELAY3:   MOV R3,#0FFH
DELAY2:   MOV R2,#0FFH
DELAY1:   DJNZ R2,DELAY1
          DJNZ R3,DELAY2
          DJNZ R4,DELAY3
          RET

DELAX:    MOV R4,#03H
DELAX3:   MOV R3,#45H
DELAX2:   MOV R2,#0FFH
DELAX1:   DJNZ R2,DELAX1
          DJNZ R3,DELAX2
          DJNZ R4,DELAX3
          RET

          END
```

หนังสืออ้างอิง

1. สมยศ จุณณะปิยะ , ผศ. " การใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS-51 ",โรงพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

2. สุเจ็ตน์ จันทรังษ์ " Single Chip Microcontroller 8051 " ,โครงการตำราวิชาการมหาวิทยาลัยมหานคร , น. 1-83

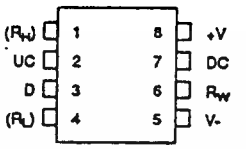
DALLAS SEMICONDUCTOR

DS1668, DS1669, DS1669S
Dallastat™ Electronic Digital Rheostat

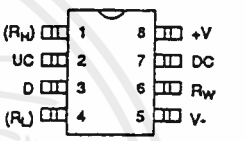
FEATURES

- Replaces mechanical variable resistors
- Available as the DS1668 with manual interface or the DS1669 integrated circuit
- Human engineered interface provides easy control with DS1668
- Electronic interface provided for digital as well as manual control
- Wide differential input voltage range between 4.5 and 8 volts
- Wiper position is maintained in the absence of power
- Low cost alternative to mechanical controls
- Applications include volume, tone, contrast, brightness, and dimmer control
- 8 pin SOIC and 8 pin DIP packages for DS1669
- Standard resistance values for Dallastat
  - DS1668/DS1669–10 = 10KΩ
  - DS1668/DS1669–50 = 50KΩ
  - DS1668/DS1669–100 = 100KΩ

PIN ASSIGNMENT DS1669



8-Pin DIP (300 Mil)
See Mech. Drawing
Sect. 16, Pg. 1

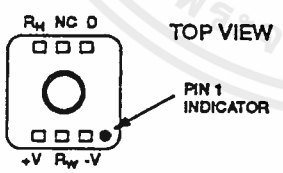


8-Pin SOIC (200 Mil)
See Mech. Drawing
Sect. 16, Pg. 5

PIN DESCRIPTION DS1669

R_H	-	Resistor High End (Option)
R_W	-	Resistor Wiper
R_L	-	Resistor Low End
-V, +V	-	Voltage Inputs
UC	-	Up Contact Input
D	-	Digital Input
DC	-	Down Contact Input
NC	-	No Connect

PIN ASSIGNMENT DS1668



PIN DESCRIPTION DS1668

+V	-	Positive Voltage Input
-V	-	Negative Voltage
R_W	-	Resistor Wiper
D	-	Digital Input
R_H	-	Resistor High End
NC	-	No Connection - Pin Missing

DESCRIPTION

The Dallastat is a digital rheostat or potentiometer which is adjusted to a desired value by a contact closure input. Alternatively, the desired setting can be achieved from a digital source input. When supplied as a 6 pin device, the contact closure is provided on the top of the package. In this configuration (DS1668), -V is connected to R_L on the bottom side of the package, and R_W, +V, D and R_H are single connections on the bottom side of the package. The 6 pin Dallastat is a self contained substitute for rheostat and potentiometer applications. Any time the button on the top of the package is depressed the resistance between pins -V and R_W will increase or decrease provided that a potential of +4.5V to +8V exist between -V and +V inputs. The 8 pin packaged versions of the Dallastat (DS1669) can be used in a similar manner as the 6 pin version with -V connected to R_L; +V connected to a positive source greater than +4.5 volts relative to -V, and a contact closure between the inputs and -V. Under this condition the wiper pin (R_W) provides a variable resistance relative to -V and is increased or decreased based on a sequence of contact inputs between UC, DC, or D, and -V.

Both the DS1668 and DS1669 can also be controlled by a digital input which functions in parallel with a contact closure or instead of contact closure. In addition, the DS1669 can be configured with and up/down two button arrangement.

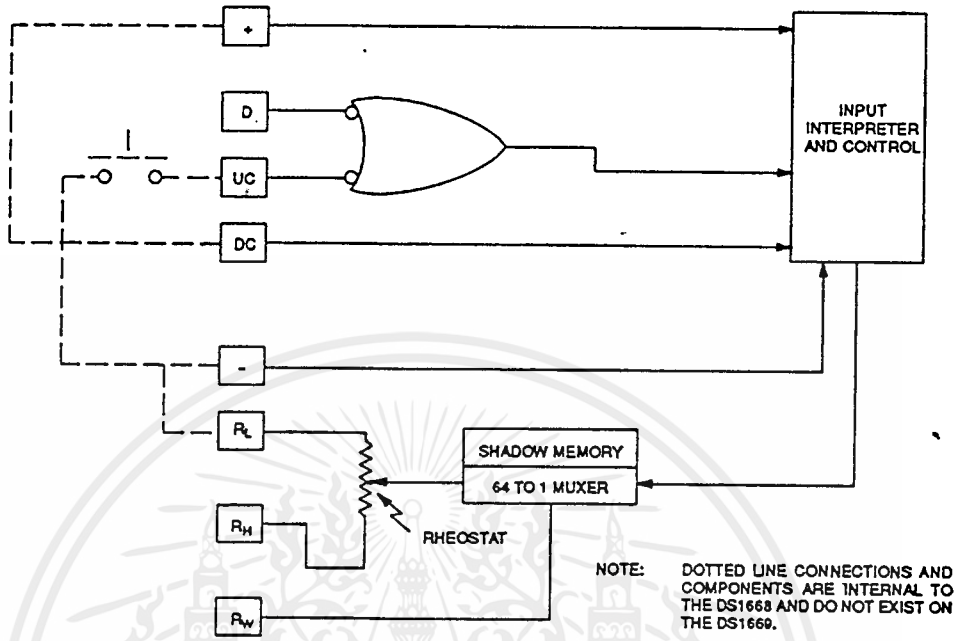
OPERATION

The main elements of the Dallastat are shown in the block diagram of Figure 1. The diagram shows that the rheostat or variable resistor setting is determined by the value of a 64 to 1 muxer which is controlled by the input interpreter. The input interpreter takes a UC, DC, or D input, and sends control information to the multiplexer. The way the interpreter derives the control information is key to the operation of the Dallastat. The dotted lines shown in the block diagram are included in the DS1668 device and serve as a typical application example for the use of the DS1669 DIP and SOIC devices. As shown, a pushbutton contact is between UC and -V and pulls the input of an "OR" gate to the negative supply. Note that "D" assumes a logic high level when not connected. When the input of the OR gate is first connected low, the interpreter sends a pulse to the multiplexer which will either increment or decrement the rheostat wiper position $^1/_{64}$ of the total taper (see flow diagram Figure 2 and 3). Increment or decrement determination is based on prior activity. A single input from contact closure of a duration of greater than 1 ms is sufficient to cause a wiper position change of $^1/_{64}$ of total. Subsequent inputs will increment or decrement $^1/_{64}$ of total for each additional contact closure. However, if the contact input remains active for greater than 1 second, increments/decrements of $^1/_{64}$ of total occur at intervals of 100 ms for as long as the input is active or until the top or bottom of the rheostat taper is reached. Anytime that input activity stops for a period of time greater than 1 second, the next action taken as a result of subsequent input activity will be reversed; i.e., if the Dallastat was incrementing, it will decrement, and if decrement was the prior action, the next action taken will be increment. If input activity is maintained for a period of time such that the upper or lower limits of the rheostat are reached, successive action is in the opposite direction. Total time of movement from one end of the taper to the other requires 64 X 100 ms + 1 second or 7.4 seconds. The DS1669 version of the Dallastat can be configured for two button operation such that the DC input can be used for decrementing and the UC input is then used only for incrementing. Upon power up, the device will internally sense the impedance between the DC input and V+. For this reason, the DC input must be connected to +V when not in use. Otherwise, the DS1669 version of the Dallastat performs as described above with the contact input attached external to the device package. Connection between contact inputs and -V of less than 10K is all that is required to be interpreted as activity. Alternatively, the D input accepts a low going signal of 0.8 volts maximum with respect to -V. The input pulse width must exceed 1 us to guarantee recognition. Successive input pulses can be any length apart provided they are not separated by more than 1 second. As with manual inputs, increment/decrement action reverses if input activity stops for a period of time greater than 1 second. If the D input is held low for more than 1 second, incrementing/decrementing occurs automatically on $^1/_{64}$ of total intervals. The flow chart for electronic control is shown in Figure 2, as the D input acts the same as the UC input. When the DS1669 is used, the rheostat low end and wiper may be connected to voltage sources other than -V or +V. The voltage applied to any rheostat element must not exceed -V - 0.5 volts on the low end or +V + 0.5 volts on the high end. If -V is connected to ground, then all other input voltages are referenced to ground.

NONVOLATILE WIPER SETTINGS

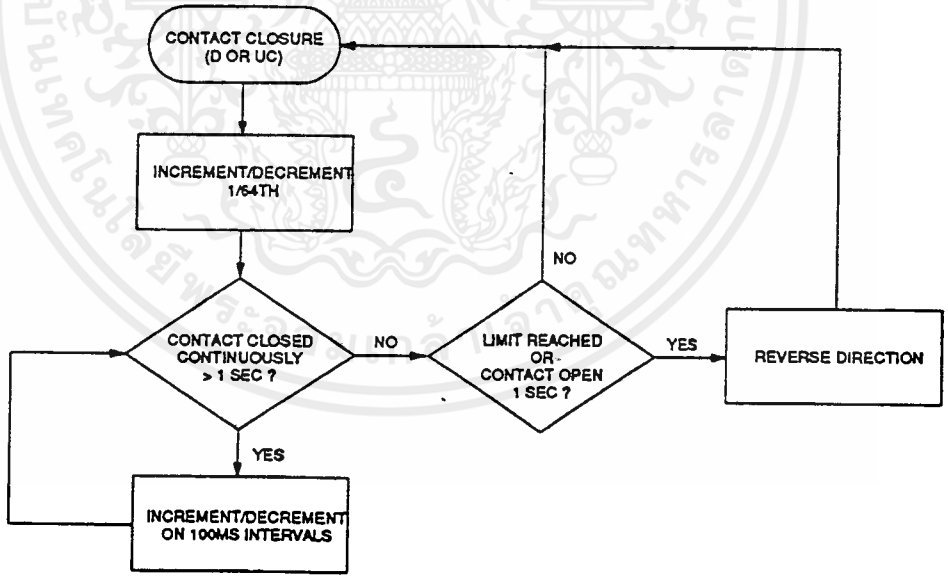
The wiper setting of the Dallastat is maintained in the absence of power in the shadow memory. During normal operation the position of wiper is determined by the multiplexer. The shadow memory is periodically updated by the multiplexer during normal operation. The manner in which an update occurs has been optimized for reliability, durability, and performance and is totally transparent to the user. When power is applied to the Dallastat, the wiper setting is set at the last value recorded in the shadow memory. On an initial power up for the first time, the wiper position may therefore be random. If the Dallastat setting is changed after power is applied, the new value will be stored in the shadow memory after a delay of about 2 seconds. The initial storing of a new value after power up always occurs when the first change is made regardless of when this change occurs after power up. After the initial change, subsequent changes in the Dallastat setting of less than 12.5% are not copied in the shadow memory. Since the Dallastat contains a 64 to 1 multiplexer, a change in the 3 LSB's is not copied into the shadow memory except for change after power up. Changes greater than 12.5% or changes large enough to affect the 4 LSB or greater are always copied into the shadow memory. As on power up, a copy from the multiplexer to shadow memory allows for a 2 second delay to guarantee that the new setting changes are finalized, and all shadow updates are transparent to the user. On power down (loss of power) the shadow memory is not changed and retains the most recent update resulting from a setting change. This value is used to set the Dallastat value on power up. The shadow memory is made with E^2 PROM type memory cells that will accept at least 80 thousand value changes before wearout. If the E^2 PROM cells ever reach a wearout condition, the Dallastat will still continue to operate properly while power is applied, but will return to the last accepted value of the shadow RAM on power up.

DS1668 DALLASTAT™ BLOCK DIAGRAM Figure 1



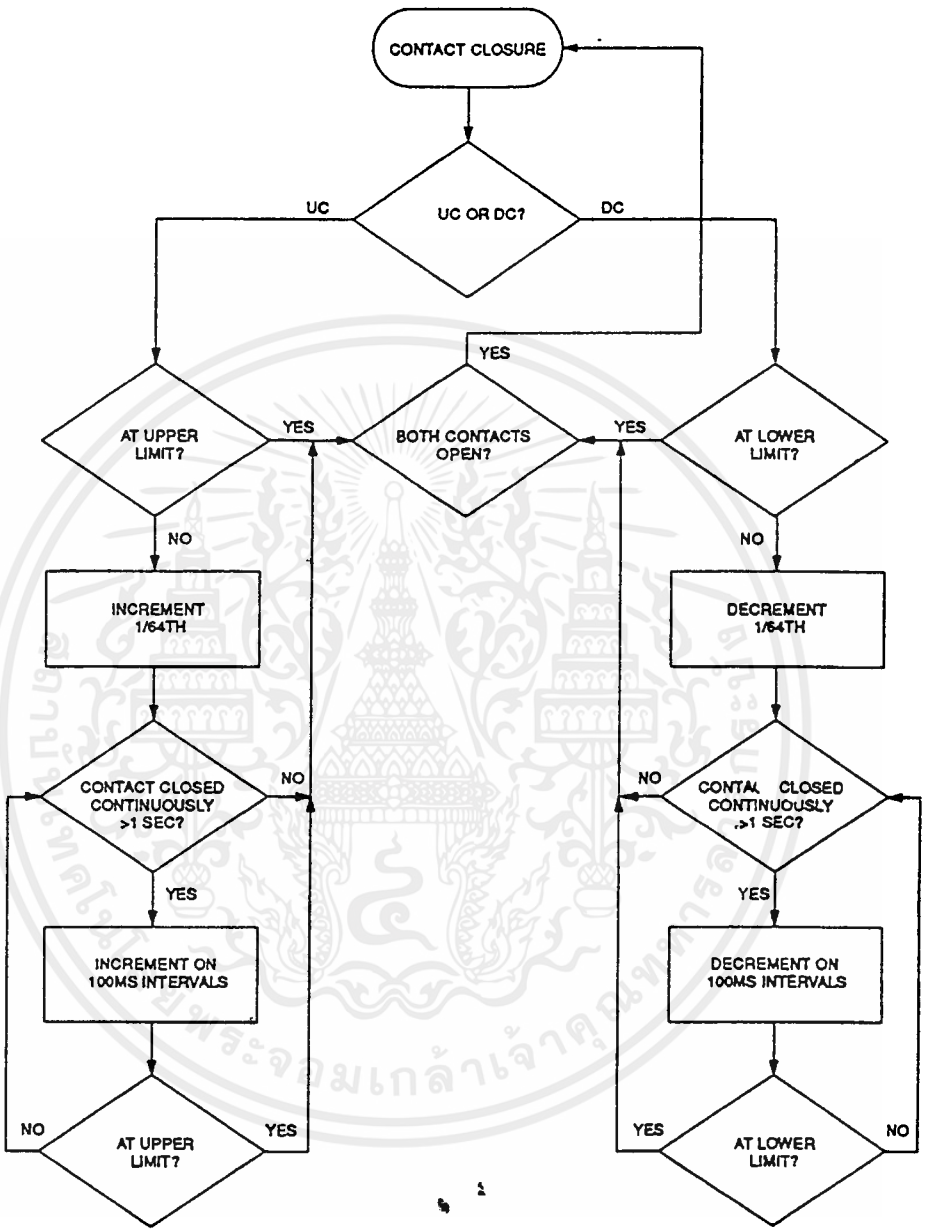
NOTE: DOTTED LINE CONNECTIONS AND COMPONENTS ARE INTERNAL TO THE DS1668 AND DO NOT EXIST ON THE DS1669.

FLOWCHART: ONE BUTTON OPERATION AND ELECTRICAL CONTROL Figure 2



CONTACT OPEN AND CONTACT CLOSURE TIMING IS 1s ± 10%

FLOWCHART: TWO BUTTON OPERATION Figure 3



CONTACT OPEN AND CONTACT CLOSURE TIMING IS 1 sec. ± 10%

ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to -V	-V –0.5V + 8.0V
Operating Temperature	–10°C to +70°C
Storage Temperature	–55°C to 125°C
Soldering Temperature	260°C for 15 seconds

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

RECOMMENDED DC OPERATING CONDITIONS

(-10°C to +70°C)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
+ Supply Voltage	+V	-V + 4.5		-V + 8.0	V	
- Supply Voltage	-V	+V - 8.0		+V - 4.5	V	
Rheostat Inputs	R_H,R_W,R_L	-V - 0.5		+V + 0.5	V	
Logic Input 1	V_{IH}	+2.4			V	1, 2
Logic Input 0	V_{IL}			+0.8	V	1, 2

DC ELECTRICAL CHARACTERISTICS

(-10°C to +70°C -V to +V = 4.5V to 7.0V)

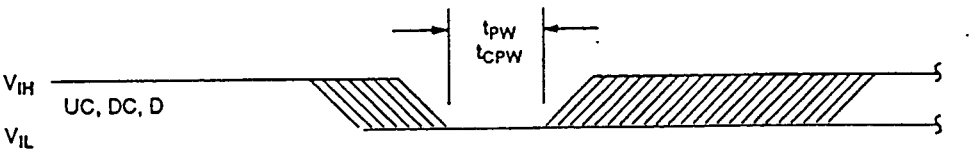
PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
+, - Supply Current	I_{CC1}		1	2	mA	3
Supply Current, Idle State	I_{CC2}			100	nA	9
Wiper Resistance	R_W		500	1000	Ω	
Wiper Current	I_W			2	mA	5
Rheostat Current	I_H, I_L			2	mA	5

AC ELECTRICAL CHARACTERISTICS

(-10°C to +70°C -V to+V = 4.5V to 7.0V)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Input Pulse Width	t_{PW}	1		DC	µs	1, 7, 8
Contact Pulse Width	t_{CPW}	1		DC	ms	1, 7, 8
Capacitance	C_{IN}		5	10	pF	6

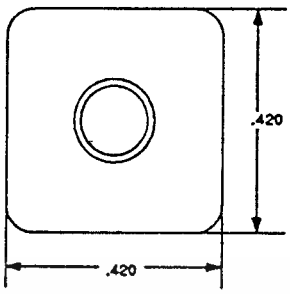
TIMING DIAGRAM

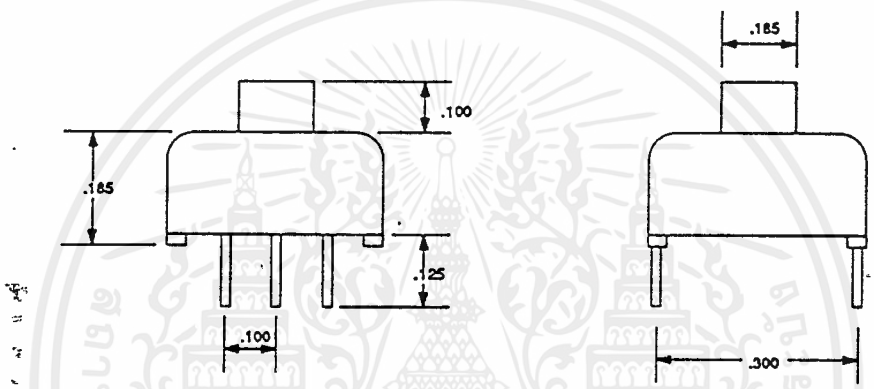


NOTES

1. All inputs; UV, DC, and D are internally pulled up with a resistance of 100KΩ.
2. Input logic levels are referenced to -V.
3. I_{CC} is the internal current that flows between -V and +V.
4. Input leakage applies to contact inputs UC and DC and digital input (D).
5. Wiper current and rheostat currents are the maximum current which can flow in the resistive elements.
6. Capacitance values apply at 25°C.
7. Input pulse width is the minimum time required for an input to cause an increment or decrement. If the UC, DC or D input is held active for longer than 1 second, subsequent increments or decrements will occur on 100 ms intervals until the inputs UC, DC, and/or D is released to V_{IH}.
8. Repetitive pulsed inputs on UC, DC, or D will be recognized as long as the pulse repetition occurs within 1 second of each other. Pulses occurring faster than 1 ms apart may not be recognized as individual inputs but can be interpreted a constant input.
9. Idle state supply current is measured with no pushbutton depressed and with the wiper. R_W tied to a CMOS load.

DS1668 PUSHBUTTON DIMENSIONS





National Semiconductor

DAC0800, DAC0801, DAC0802 8-Bit Digital-to-Analog Converters

General Description

The DAC0800 series are monolithic 8-bit high-speed current-output digital-to-analog converters (DAC) featuring typical settling times of 100 ns. When used as a multiplying DAC, monotonic performance over a 40 to 1 reference current range is possible. The DAC0800 series also features high compliance complementary current outputs to allow differential output voltages of 20 Vp-p with simple resistor loads as shown in *Figure 1*. The reference-to-full-scale current matching of better than ±1 LSB eliminates the need for full-scale trims in most applications while the nonlinearities of better than ±0.1% over temperature minimizes system error accumulations.

The noise immune inouts of the DAC0800 series will accept TTL levels with the logic threshold pin, V_{LC} pin 1 grounded. Simple adjustments of the V_{LC} potential allow direct interface to all logic families. The performance and characteristics of the device are essentially unchanged over the full ± 4.5V to ±18V power supply range; power dissipation is only 33 mW with ±5V supplies and is independent of the logic input states.

The DAC0800, DAC0802, DAC0800C, DAC0801C and DAC0802C are a direct replacement for the DAC-08, DAC-08A, DAC-08C, DAC-08E and DAC-08H, respectively.

Features

- Fast settling output current — 100 ns
- Full scale error — ±1 LSB
- Nonlinearity over temperature — ±0.1%
- Full scale current drift — ±10 ppm/°C
- High output compliance — −10V to +18V
- Complementary current outputs
- Interface directly with TTL, CMOS, PMOS and others
- 2 quadrant wide range multiplying capability
- Wide power supply range — ±4.5V to ±18V
- Low power consumption — 33 mW at ±5V
- Low cost

Typical Applications

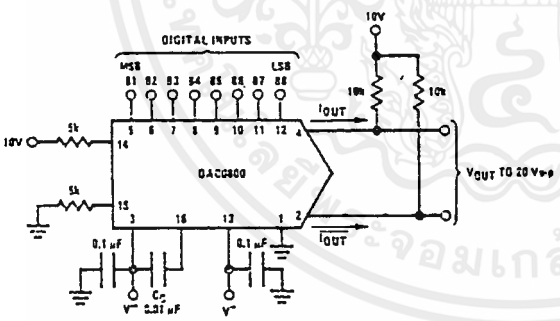
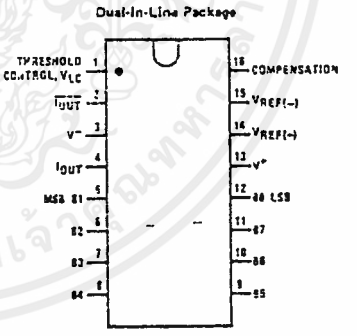


FIGURE 1. ±20 Vp-p Output Digital-to-Analog Converter

Connection Diagram



TOP VIEW

Ordering Information

NON LINEARITY	TEMPERATURE RANGE	ORDER NUMBERS*					
		D PACKAGE (D16C)		J PACKAGE (J16A)		N PACKAGE (N16A)	
±0.1% FS	−55°C ≤ T_A ≤ +125°C	DAC0802LD	DAC-08AQ				
±0.1% FS	0°C ≤ T_A ≤ + 70°C			DAC0802LCJ	DAC-08HQ	DAC0802LCN	DAC-08HP
±0.19% FS	−55°C ≤ T_A ≤ + 125°C	DAC0800LD	DAC-08Q				
±0.19% FS	0°C ≤ T_A ≤ +70°C			DAC0800LCJ	DAC-08EQ	DAC0800LCN	DAC-08EP
±0.39% FS	0°C ≤ T_A ≤ +70°C			DAC0801LCJ	DAC-08CQ	DAC0801LCN	DAC-08CP

*Note. Devices may be ordered by using either order number.

Absolute Maximum Ratings

Supply Voltage	±18V or 36V
Power Dissipation (Note 1)	500 mW
Reference Input Differential Voltage (V14 to V15)	V^- to V^+
Reference Input Common-Mode Range (V14, V15)	V^- to V^+
Reference Input Current	5 mA
Logic Inputs	V^- to V^- plus 36V
Analog Current Outputs	*Figure 24*
Storage Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C

Operating Conditions

	MIN	MAX	UNITS
Temperature (T_A)			
DAC0802L	−55	+125	°C
DAC0800L	−55	+125	°C
DAC0800LC	0	+70	°C
DAC0801LC	0	+70	°C
DAC0802LC	0	+70	°C

Electrical Characteristics (V_S = ±15V, I_{REF} = 2 mA, $T_{MIN} \le T_A \le T_{MAX}$ unless otherwise specified. Output characteristics refer to both I_{OUT} and $\overline{I_{OUT}}$.)

	PARAMETER	CONDITIONS	DAC0802L/ DAC0802LC			DAC0800L/ DAC0800LC			DAC0801LC			UNITS
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
	Resolution		8	8	8	8	8	8	8	8	8	Bits
	Monotonicity		8	8	8	8	8	8	8	8	8	Bits
	Nonlinearity				±0.1			±0.19			±0.39	%FS
t_s	Settling Time	To ±1/2 LSB, All Bits Switched "ON" or "OFF", T_A = 25°C		100	135					100	150	ns
		DAC0800L					100	135				ns
		DAC0800LC					100	150				ns
tPLH, tPHL	Propagation Delay	T_A = 25°C										
	Each Bit			35	60		35	60		35	60	ns
	All Bits Switched			35	60		35	60		35	60	ns
TCIFS	Full Scale Tempco			±10	±50		±10	±50		±10	±80	ppm/°C
V_{OC}	Output Voltage Compliance	Full Scale Current Change < 1/2 LSB, R_{OUT} > 20 MΩ Typ	−10		18	−10		18	−10		18	V
IFS4	Full Scale Current	V_{REF} = 10.000V, R14 = 5.000 kΩ R15 = 5.000 kΩ, T_A = 25°C	1.984	1.992	2.000	1.94	1.99	2.04	1.94	1.99	2.04	mA
IFSS	Full Scale Symmetry	IFS4 − IFS2		±0.5	±4.0		±1	±8.0		±2	±16	µA
IZS	Zero Scale Current			0.1	1.0		0.2	2.0		0.2	4.0	µA
IFSR	Output Current Range	V^- = −5V	0	2.0	2.1	0	2.0	2.1	0	2.0	2.1	mA
		V^- = −8V to −18V	0	2.0	4.2	0	2.0	4.2	0	2.0	4.2	mA
	Logic Input Levels											
V_{IL}	Logic "0"	V_{LC} = 0V			0.8			0.8			0.8	V
V_{IH}	Logic "1"		2.0			2.0			2.0			V
	Logic Input Current	V_{LC} = 0V										
I_{IL}	Logic "0"	−10V ≤ V_{IN} ≤ +0.8V		−2.0	−10		−2.0	−10		−2.0	−10	µA
I_{IH}	Logic "1"	2V ≤ V_{IN} ≤ +18V		0.002	10		0.002	10		0.002	10	µA
V_{IS}	Logic Input Swing	V^- = −15V	−10		18	−10		18	−10		18	V
VTHR	Logic Threshold Range	V_S = ±15V	−10		13.5	−10		13.5	−10		13.5	V
I_{15}	Reference Bias Current			−1.0	−3.0		−1.0	−3.0		−1.0	−3.0	µA
dI/dt	Reference Input Slew Rate	*(Figure 24)*	4.0	8.0		4.0	8.0		4.0	8.0		mA/µs
PSSIFS+	Power Supply Sensitivity	4.5V ≤ V^- ≤ 18V		0.0001	0.01		0.0001	0.01		0.0001	0.01	%/%
PSSIFS−		−4.5V ≤ V^- ≤ 18V		0.0001	0.01		0.0001	0.01		0.0001	0.01	%/%
		I_{REF} = 1 mA										
	Power Supply Current	V_S = ±5V, I_{REF} = 1 mA										
I+				2.3	3.8		2.3	3.8		2.3	3.8	mA
I−				−4.3	−5.8		−4.3	−5.8		−4.3	−5.8	mA
		V_S = 5V, −15V, I_{REF} = 2 mA										
I+				2.4	3.6		2.4	3.6		2.4	3.8	mA
I−				−6.4	−7.8		−6.4	−7.8		−6.4	−7.8	mA
		V_S = ±15V, I_{REF} = 2 mA										
I+				2.5	3.8		2.5	3.8		2.5	3.8	mA
I−				−6.5	−7.8		−6.5	−7.8		−6.5	−7.8	mA
P_D	Power Dissipation	±5V, I_{REF} = 1 mA		33	48		33	48		33	48	mW
		5V, −15V, I_{REF} = 2 mA		108	138		108	136		108	136	mW
		±15V, I_{REF} = 2 mA		135	174		135	174		135	174	mW

Note 1: The maximum junction temperature of the DAC0800, DAC0801 and DAC0802 is 125°C. For operating at elevated temperatures, devices in the dual-in-line J or D package must be derated based on a thermal resistance of 100°C/W, junction to ambient, 175°C/W for the molded dual-in-line N package.