เครื่องวิเคราะห์สเปคตรัมควบคุมด้วยคอมพิวเตอร์

COMPUTER CONTROLED SPECTRUM ANALYZER

โดย

นายจักรพงศ์ สำลีว่อง

นายธีรชัย สว่างวารี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

เครื่องวิเคราะห์สเปคตรัมควบคุมด้วยคอมพิวเตอร์

COMPUTER CONTROLED SPECTRUM ANALYZER

โดย

นายจักรพงศ์ สำลีว่อง 40013042

นายธีรชัย สว่างวารี 40013052

อาจารย์ที่ปรึกษา

ดร. สุทธิชัย นพนาคีพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

เครื่องวิเคราะห์สเปคตรัมควบคุมด้วยคอมพิวเตอร์
COMPUTER CONTROLED SPECTRUM ANALYZER

โดย นายจักรพงศ์ สำลีว่อง 40013042
นายธีรชัย สว่างวารี 40013052

อาจารย์ที่ปรึกษา ดร. สุทธิชัย นพนาคีพงษ์

บทคัดย่อ

เครื่องวิเคราะห์สเปคตรัมนั้นเป็นเครื่องมือที่ใช้วิเคราะห์สัญญาณความถี่ต่างๆ เช่นสัญญาณความถี่วิทยุ(Radio Frequency)เป็นต้น ซึ่งในโครงงานนี้จะใช้คอมพิวเตอร์ควบคุมการวิเคราะห์สัญญาณโดยมีลักษณะเป็นการ์ดวัดเสียบต่อกับคอมพิวเตอร์ส่วนบุคคล และควบคุมการทำงานทั้งหมด รวมทั้งแสดงผลด้วยคอมพิวเตอร์ การทำงานของเครื่องวิเคราะห์สเปคตรัมนี้จะประกอบด้วยส่วนใหญ่ๆ 2 ส่วน คือ ฮาร์ดแวร์(Hardware) และ ซอฟแวร์ (Software) โดยฮาร์ดแวร์คือการ์ดที่ทำหน้าที่แปลงสัญญาณอนาล็อกไปเป็นสัญญาณดิจิตอลและติดต่อสื่อสารกับคอมพิวเตอร์ได้ ส่วนซอฟแวร์ใช้หลักการแปลงสัญญาณอินพุทด้วยวิธีฟาสต์ฟูเรียร์ทรานส์ฟอร์ม (Fast Fourier Transform) สามารถวิเคราะห์สเปคตรัมที่มีความถี่ต่างๆได้ และจะเป็นเครื่องวิเคราะห์สเปคตรัมที่ราคาประหยัด เมื่อเทียบกับเครื่องวิเคราะห์สเปคตรัมทั่วไป สามารถวิเคราะห์สัญญาณที่มีความถี่ตั้งแต่ 10 กิโลเฮิร์ต ถึง 5 เม็กกะเฮิร์ต

ABSTRACT

A frequency analyzer is an auxiliary tool that is used for analyzing frequency such as radio frequency etc. That is we will use the computer to control frequency analysis by using a computer card as a medium to communicate with the computer and control all of operation. In addition an analytical result will be displayed on the computer scene. The computer controlled spectrum analyzer operation can be separate into two pat. The first one is a hardware card is used to transform an analog signal to be a digital signal and communicate with computer. The second one is a software that using fourier transtorm technique, finally advantage of using computer as the frequency analyzer bring us into a save way.It can be used in frequency range of 10 kHz to 5 MHz.

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องวิเคราะห์สเปคตรัมควบคุมด้วยคอมพิวเตอร์

COMPUTER CONTROLED SPECTRUM ANALYZER

ผู้จัดทำ

1. นายจักรพงศ์ สำลีว่อง 40013042

2. นายธีรชัย สว่างวารี 40013052

..... อาจารย์ที่ปรึกษา

(ดร. สุทธิชัย นพนาคีพงษ์)

สารบัญ

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

สเปคตรัมอะนาไลเซอร์เป็นเครื่องมือวัดที่สำคัญตัวหนึ่งในงานด้านการสื่อสารและอีเลคทรอนิกส์ทั่วไป หากย้อนไปดูประวัติของเครื่องมือวัดตัวนี้จะเห็นการพัฒนาอย่างต่อเนื่องจนถึงปัจจุบันได้มีการใช้ไมโครโปรเซสเซอร์เข้ามาช่วยประมวลผล เปลี่ยนชุดแสดงผลเป็นระบบดิจิตอลปรับปรุงชุดออสซิลเลเตอร์ให้เป็นแบบ Synthesizer และอื่นๆอีกมาก ทำให้สเปคตรัมอะนาไลเซอร์ใช้งานได้ง่ายขึ้น และมีความเที่ยงตรงสูง

สเปคตรัมอะนาไลเซอร์เป็นเครื่องมือที่ทำหน้าที่แยกกระจายองค์ประกอบของสัญญาณที่วัดออกจากกันตามความถี่ และแสดงผลขององค์ประกอบนั้นๆบนจอภาพ โดยมีแกนนอน(แกน X)แสดงความถี่ ส่วนแกนตั้ง (แกน Y) จะแสดงค่าความแรง(Amplitude) ของตัวองค์ประกอบแต่ละตัวในปัจจุบันสเปคตรัมอะนาไลเซอร์มี 2 ระบบคือ Super Heterodyne และ FFT ระบบ Super Heterodyne เป็นระบบที่ใช้ทั่วไปในปัจจุบันสเปคตรัมอะนาไลเซอร์ ซึ่งใช้ระบบนี้จะแยกออกเป็น 2 ชนิดตามย่านความถี่ที่ใช้งานคือ ย่าน RF(Radio Frequency) และย่าน Microwave ส่วนระบบ FFT นั้นจากความก้าวหน้าของเทคโนโลยีดิจิตอล ระบบนี้จะอ่านสัญญาณอินพุทซึ่งเป็นอนาล็อก และเปลี่ยนสัญญาณเป็นดิจิตอลด้วย A-D Converter จากนั้นจะใช้กรรมวิธี Fast Fourier Transform (FFT) หาค่า Amplitude เทียบกับความถี่ หรือค่า Phase เทียบกับความถี่ออกมาแสดงผล

เนื่องจากเครื่องสเปคตรัมอะนาไลเซอร์ ในปัจจุบันจะมีราคาสูงมากจึงมีการพัฒนานำคอมพิวเตอร์(Computer) ซึ่งจะมีราคาถูกกว่าเครื่องสเปคตรัมอะนาไลเซอร์มาใช้งานโดยเฉพาะเพื่อเป็นเครื่องสเปคตรัมอะนาไลเซอร์ที่วิเคราะห์สัญญาณต่างๆ โดยการสร้างการ์ด(Card)มีลักษณะเฉพาะขึ้นมาเพื่อใช้งานกับคอมพิวเตอร์ โดยตัวการ์ดนี้สามารถแปลงสัญญาณอนาล็อก (Analog Signal) ไปเป็นสัญญาณดิจิตอล(Digital Signal) ที่ความถี่สูงๆได้ แล้วประกอบกับส่วนของ โปรแกรมที่ทำการเปลี่ยนข้อมูลได้ซึ่งเป็นค่าแอมปลิจูด (Amplitude) ให้อยู่ในรูปจำนวนเชิงซ้อน (Complex Number) โดยกำหนดให้ข้อมูลที่ได้เป็นข้อมูลจริง (Real Number) และกำหนดให้ข้อมูลจินตภาพ(Imaginary) มีค่าเป็น “0” โดยข้อมูลที่ได้จะเป็นค่าแอมปลิจูดเชิงเวลา(Time Domain) แล้วทำการเปลี่ยนข้อมูลเชิงเวลาให้อยู่ในรูปของข้อมูลเชิงความถี่(Frequency Domain) โดยใช้วิธีการของฟาสต์ฟูเรียร์ทรานส์ฟอร์มที่เรียกว่าบัตเตอร์ฟลาย (Butterfly) จากนั้นนำข้อมูลที่ได้ไปแสดงผลออกทางจอคอมพิวเตอร์

จะเห็นได้ว่าโครงงานนี้เทียบเท่ากับเครื่องสเปคตรัมอะนาไลเซอร์เครื่องหนึ่งที่ใช้คอมพิวเตอร์เป็นตัวควบคุมโดยผ่านทางสล็อต(Slot)คอมพิวเตอร์

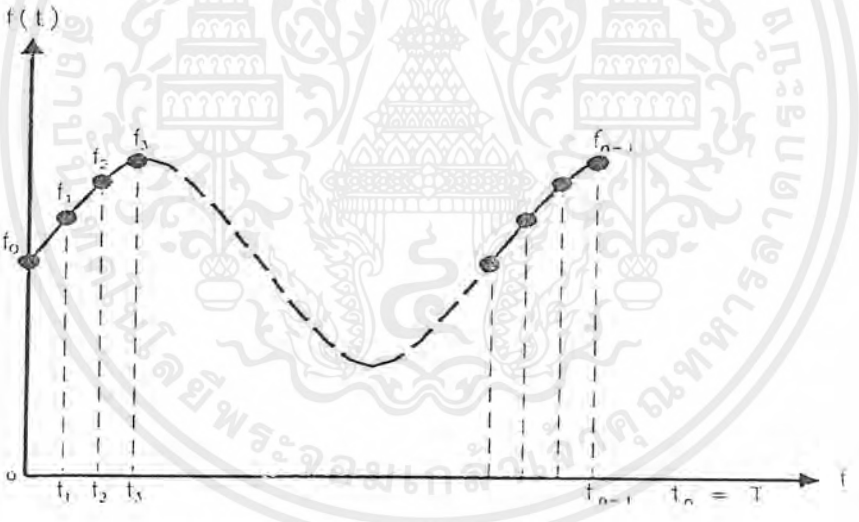
บทที่ 2
ทฤษฎีหรือหลักการ

2.1 ทฤษฎีฟาสต์ฟูเรียร์ทรานส์ฟอร์ม (FFT)

การคำนวณฟูเรียร์เพื่อให้สามารถคำนวณได้เร็วขึ้นโดยจะใช้ทฤษฎีฟาสต์ฟูเรียร์ทรานส์ฟอร์ม ดังนั้นจึงจำต้องทำการศึกษาและทำความเข้าใจทฤษฎีดีสครีสทรานส์ฟอร์มเสียก่อน ซึ่งจะเห็นได้ว่าฟาสต์ฟูเรียร์ทรานส์ฟอร์มนั้นเป็นการพัฒนามาจากดีสครีสทรานส์ฟอร์มจะทำให้คำนวณได้เร็วกว่าประมาณ 10 เท่า

2.1.1 ทฤษฎีดีสครีทฟูเรียร์ทรานส์ฟอร์ม (THE DISCRETE FOURIER TRANSFORM)

ดีสครีทฟูเรียร์ทรานส์ฟอร์มเป็นการแปลงฟูเรียร์แบบไม่ต่อเนื่อง โดยข้อมูลที่สุ่มเข้ามาจะต้องเริ่มจากค่า "0" จนถึงค่า "T" ดังรูปที่ 2.1 โดยให้ค่า N เป็นจำนวนการสุ่มข้อมูลทั้งหมด โดยมี เวลา t_n เป็นเวลาในการสุ่มข้อมูล และค่า f_n เป็นค่าของฟังก์ชั่นต่อเนื่องตามการสุ่มของ t_n ใดๆ



รูปที่ 2.1 ดีสครีทฟูเรียร์ทรานส์ฟอร์ม

โดยข้อมูลจะอยู่ที่ตำแหน่ง $n = 0,1,2,3,\ldots\ldots, N-1$ สามารถเขียนสมการดีสครีสฟูเรียร์ทรานส์ฟอร์มได้ดังนี้

$$F_k = \sum_{n=0}^{N-1} f_n e^{-jk\omega_0 n} \qquad \text{โดยที่ค่า } k = 0 \text{ ถึง } N-1 \tag{2.1}$$

ดังนั้นจะได้ค่าอินเวิสฟูเรียร์ทรานส์ฟอร์มดังนี้

$$f_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k e^{jk\omega_0 n} \quad \text{โดยที่ค่า } k = 0 \text{ ถึง } N-1 \tag{2.2}$$

โดยค่า $\omega_0 = 2\pi/N$ และการแปลงค่าดีสครีทฟูเรียร์ทรานส์ฟอร์มจากสมการ (2.1) ต้องทำการคำนวณเป็นจำนวน N^2 โดยข้อมูลต้องอยู่ในรูปของจำนวนเชิงซ้อนเพื่อให้คอมพิวเตอร์คำนวณได้ง่าย จากสมการ (2.1) สามารถใช้ทฤษฎีของอูเลอร์(Euler's) มาคำนวณได้จากสูตรดังนี้

$$e^{\pm ia} = \cos a \pm j \sin a$$

แทนสูตรของอูเลอร์ลงในสมการ (2.1) และ (2.2) จะได้

$$F_k = \sum_{k=0}^{N-1} [f_n \cos(k\omega_0 n) - jf_n \sin(k\omega_0 n)] \tag{2.3}$$

$$f_n = \frac{1}{N}\sum_{k=0}^{N-1} [f_k \cos(k\omega_0 n) + jF_k \sin(k\omega_0 n)] \tag{2.4}$$

จากสมการ (2.3) และ (2.4) สามารถนำไปเขียนโปรแกรมเพื่อใช้ในการคำนวณหาค่าฟูเรียร์ต่อไป

ตัวอย่างต่อไปนี้เป็นการสุ่ม(Sampling) สัญญาณดังรูปที่ 2.1 ให้เป็นสัญญาณที่ไม่ต่อเนื่องเพื่อนำขนาดของแซมเปิ้ลมาคำนวณ โดยจะให้ Sequence ของสัญญาณ h_m = [3,4,2,2] และหาSpectrum ของสัญญาณนั้นจะได้ $N = 4$

จากสูตรการแปลงฟูเรียร์ของสัญญาณดีสครีท

$$H_k = \frac{1}{N}\sum_{m=0}^{N-1} h_m e^{jmk2\pi/N} \qquad ; k = 0,1,2,3,\ldots\ldots,N-1$$

$$h_m = \sum_{m=0}^{N-1} h_m e^{jmk2\pi/N} \qquad ; m = 0,1,2,3,\ldots\ldots,N-1$$

แทนค่าต่างๆจะได้

$$H_k = \frac{1}{4}\sum_{m=0}^{3} h_m e^{-jmk2\pi/N}$$

$$H_{(0)} = ¼ \sum_{m=0}^{3} h_m \, e^{0}$$

$$= ¼ [h(0) + h(1) + h(2) + h(3)]$$

$$= ¼ [3+4+2+2] = 2.75$$

$$H_{(1)} = ¼ \sum_{m=0}^{3} h_m e$$

$$= ¼ [h(0)\, e^{-j0} + h(1)\, e^{-j\pi/2} + h(2)\, e^{-j\pi} + h(3)\, e^{-j3\pi/2}]$$

$$= ¼ [3.1+4.(-j)+2.(-1)+2.(j)]$$

$$= ¼ [3-j4-2+j2] = ¼ [1-j2]$$

$$H_{(2)} = ¼ \sum_{m=0}^{3} h_m e^{-jm\pi}$$

$$= ¼ [h(0)\, e^{-j0} + h(1)\, e^{-j\pi} + h(2)\, e^{-j2\pi} + h(3)\, e^{-j3\pi}]$$

$$= ¼ [3.1+4.(-1)+2.(1)+2.(-1)]$$

$$= ¼ [3-4+2-2] = -0.25$$

$$H_{(3)} = 1/4 \sum_{m=0}^{3} h_m e^{-j3\pi/2}$$

$$= ¼ [h(0)\, e^{-j0} + h(1)\, e^{-j3\pi/2} + h(2)\, e^{-j3\pi} + h(3)\, e^{-j9\pi}]$$

$$= ¼ [3.1+4.(j)+2.(-1)+2.(-j)]$$

$$= ¼ [3+j4-2-j2] = ¼ [1+2j]$$

เพราะฉะนั้นสเปคตรัม คือ

$$|H_{(0)}| = 2.75$$

$$|H_{(1)}| = ¼ |(1^2 + (-2)^2)^{1/2}| = 0.56$$

$$|H_{(2)}| = |-0.25| = 0.25$$

$$|H_{(3)}| = ¼ |(1^2 + 2^2)^{1/2}| = 0.56$$

2.1.2 ทฤษฎีฟาสต์ฟูเรียร์ทรานส์ฟอร์มและวิธีการแบบบัตเตอร์ฟลาย

ฟาสต์ฟูเรียร์ทรานส์ฟอร์มเป็นการพัฒนามาจากดีสครีทฟูเรียร์ทรานส์ฟอร์มเพื่อให้มีการคำนวณได้เร็วขึ้นซึ่งดีสครีทฟูเรียร์ทรานส์ฟอร์มต้องการข้อมูลที่ใช้ในการคำนวณเป็นจำนวณ N^2 แต่ฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะใช้ข้อมูลในการคำนวณประมาณ $N \log_2 N$

ฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะคำนวณเร็วกว่าดีสครีทฟูเรียร์ทรานส์ฟอร์มประมาณ 10 เท่า โดยจะใช้วิธีการแบบบัตเตอร์ฟลาย หรือวิธีการของคูเล่ย์ตูคีย์(Cooley-Tukey Algorithm)

สมมุติให้ $N = 2^M$ โดย M เป็นค่าจริงใดๆจะได้

$F_k = H(k), f_n = h(m)$ จะได้สมการใหม่ดังนี้

$$H(k) = \sum_{m=0}^{N-1} h(m) e^{-j(2\pi/N)km} \qquad \text{โดย } k = 0,1,2,3,\ldots,N-1 \tag{2.5}$$

ถ้าให้ $e^{-j(2\pi/N)km} = W^{km}$ ได้สมการใหม่ดังนี้

$$H(k) = \sum_{m=0}^{N-1} h(m) W^{mk} \tag{2.6}$$

ซึ่งแบ่งสมการ (2.6) ได้เป็น 2 ส่วนดังนี้

$$H(k) = \sum_{m=0}^{N/2-1} h(2m) W_N^{mk} + \sum_{m=0}^{N/2-1} h(2m+1) W_N^{(2N+1)k} \tag{2.7}$$

N แทนขนาดความยาวของการสุ่มตัวอย่างและแทนสมการทั้งสองด้วยสมการคู่ และสมการคี่ได้สมการใหม่คือ

$$H(k) = \sum_{m=0}^{N/2-1} h_{ev}(m) W_{N/2}^{mk} + \sum_{m=0}^{N/2-1} h_{od}(m) W_{N/2}^{mk} \tag{2.8}$$

หรือ $$H(k) = H_{ev}(k) + W_{N/2}^{k} H_{0d}(k) \tag{2.9}$$

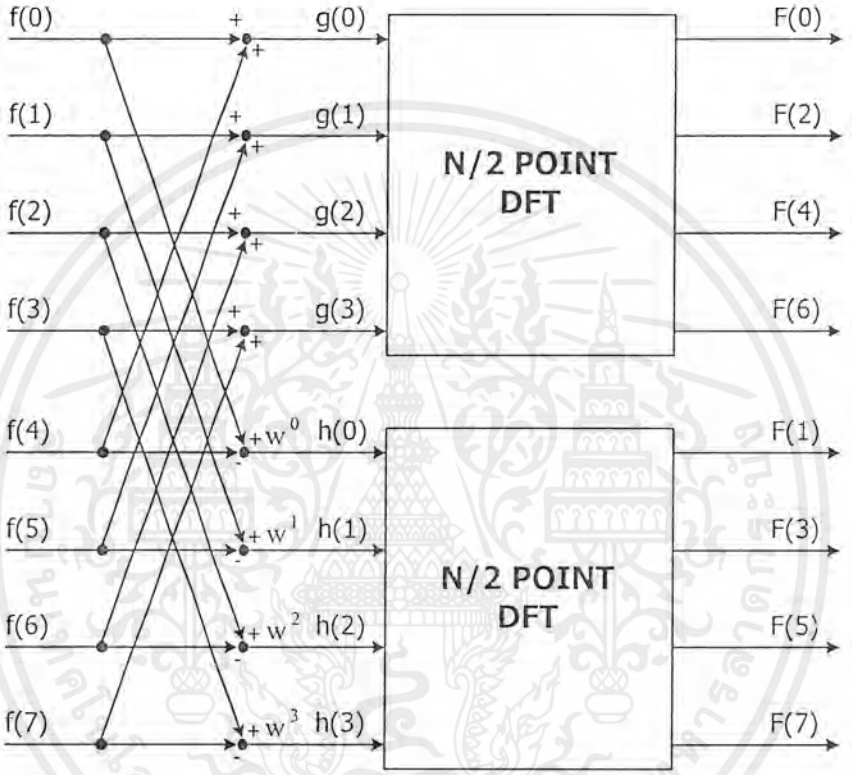
ในการนี้จะใช้ $N/2$ จุดเท่านั้นเพื่อคำนวณหาค่า $H(k)$ โดย k ต้องมีค่าตั้งแต่ 0 ถึง N -1 อย่างไรก็ตามคุณสมบัติของสมการคู่และสมการคี่หาได้จาก

$$H_{ev}(k) = H_{ev}\left(k - \frac{N}{2}\right) \rightarrow for \frac{N}{2} \le k \le (N-1) \tag{2.10}$$

ผลลัพธ์ของการแปลงฟูเรียร์แบบไม่ต่อเนื่องสามารถทำซ้ำๆ จนกระทั่งเหลือข้อมูลแค่ตัวเดียว โดยต้องมีข้อมูลอินพุท 2 ตัวเพื่อใช้ในการคำนวณตลอดเวลา

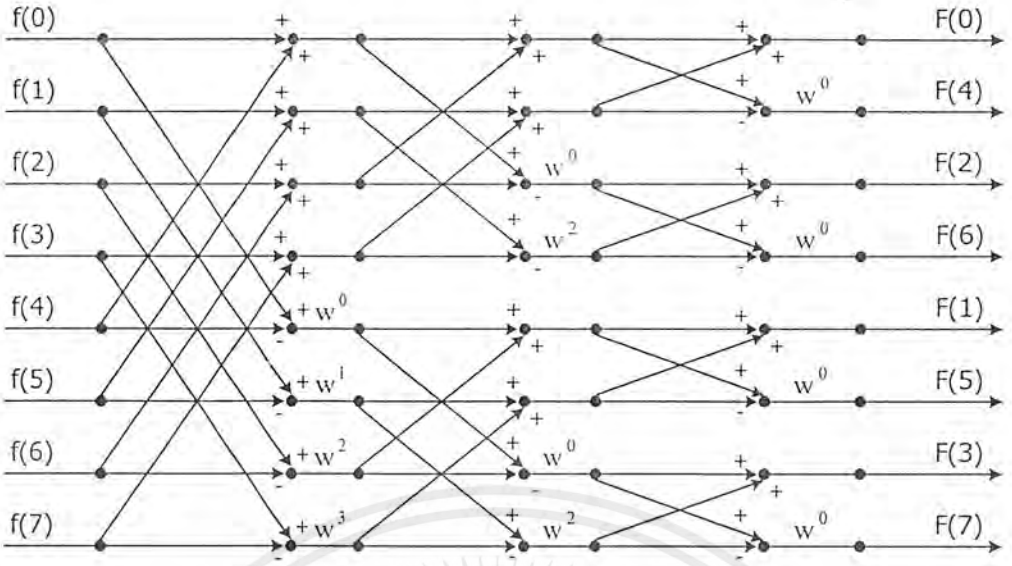
$$\begin{aligned} \Lambda(k) &= \lambda(0) + \lambda(1) e^{-j2\pi k/2} && \text{for all} \\ &= \lambda(0) + \lambda(1) && \text{for } k \text{ even} \\ &= \lambda(0) - \lambda(1) && \text{for } k \text{ odd.} \quad \text{ตามลำดับ} \end{aligned}$$

สำหรับการหาค่าฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะใช้แค่ 2 จุดในการคำนวณหาค่าของฟาสต์ฟูเรียร์ทรานส์ฟอร์มที่สมบูรณ์จะต้องมีการคูณโดยใช้แฟคเตอร์ที่เหมาะสมคือ ค่า W โดยค่า W จะต้องเริ่มจาก W^0 ถึง $W^{N/2-1}$ ในรูปที่ 2.2 แสดงกราฟของฟาสต์ฟูเรียร์ทรานส์ฟอร์ม 8 จุดโดยต้องคำนวณหาค่าต่างๆ ตามวิธีการของบัตเตอร์ฟลาย (Butterflies)ให้เหมาะสม



รูปที่ 2.2 การหาค่า 8 จุดฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

สำหรับการเริ่มต้นคำนวณต้องการทราบค่าของ N ที่จะใช้ในการคำนวณ จากรูปที่ 2.2 จะใช้ค่า $N = 8$ และจะได้ค่า g_n และ h_n ที่คำนวณได้เป็นจำนวน $N/2$ จุดโดยมีค่า W^n มาคูณกับค่า h_n ดังรูปที่ 2.1 และจะคำนวณในลำดับต่อไปดังรูปที่ 2.3



รูปที่ 2.3 การหาค่า 8 จุดฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

จากรูปที่ 2.3 เมื่อคำนวณช่วงแรกเสร็จจะคำนวณช่วงที่ 2 ต่อ โดยมีวิธีการคำนวณดังรูปและจะใช้ข้อมูลการคำนวณทั้งหมดประมาณ $N\log_2(N)$ สามารถหาจำนวนของบัตเตอร์ฟลาย(Number of butterflies)ได้ดังนี้

จำนวนของบัตเตอร์ฟลาย = $\frac{N}{2}\log_2(N)$

โดย $N/2$ = จำนวนแถวของบัตเตอร์ฟลาย (มี 2 อินพุท)

$\log_2(N)$ = จำนวนหลักของบัตเตอร์ฟลาย

แต่ละบัตเตอร์ฟลายต้องมี 2 อินพุทเท่านั้น

2.1.3 คุณสมบัติและเทคนิคช่วยแปลงฟูเรียร์อย่างเร็ว

สิ่งจูงใจในการเสาะหาวิธีการ (Motivation to search for an algorithm)

เนื่องจากผลการทำ FFT จะได้

$$H\left(\frac{N}{2}+l\right)=\overline{H}\left(\frac{N}{2}-l\right) \quad ; \quad l = 1,2,\ldots\ldots,\frac{N}{2}-1$$

เมื่อ $\overline{H}(k)$ เป็น complex conjugate ของ $H(k)$

กุญแจสำคัญในการพัฒนาวิธีการ (Key to develop the algorithm)

จาก m ซึ่งเป็นเลขฐานสิบ สามารถเขียนในรูปเลขฐานสองได้เป็น

$$m = m_{n-1}2^{n-1} + m_{n-2}2^{n-2} + \ldots + m_1 2^1 + m_0 2^0$$

โดย $m_v = 0$ หรือ 1 เมื่อ v = 0,1,2,..., n - 1

ทำนองเดียวกันกับ k เป็นเลขฐานสิบ เขียนให้อยู่ในรูปเลขฐานสองจะได้

$$k = k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + \ldots + k_1 2^1 + m_0 2^0$$

โดย $k_v = 0$ หรือ 1 เมื่อ $v = 0,1,...,N-1$

ถ้าเราให้ $\tilde{h}(m)$ เป็นลำดับ (Sequence) ของข้อมูลที่ตัวแปรเป็นเลขฐานสองที่ใช้แทน $h(m)$ ซึ่งตัวแปรเป็นเลขฐานสิบ เราจะได้ว่า

$$h(m) = \tilde{h}(m_{n-1}2^{n-1} + m_{n-2}2^{n-2} + ... + m_1 2^1 + m_2 2^0) \tag{2.11}$$

หรือ $h(m) = \tilde{h}(m_{n-1}, m_{n-2}, ...m_1, m_0)$

จากสมการที่ (2.11) พบว่า

$$\sum_{m=0}^{N-1} h(m)W^{km} = \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}.... \sum_{m_{n-1}=0}^{1} \tilde{h}(m_{n-1}, m_{n-2}, ..., m_0)W^{k[m_{n-1}2^{n-1}+m_{n-2}2^{n-2}+...+m_0 2^0]} \tag{2.12}$$

ตัวอย่างในกรณีที่ $N = 4$ หรือ $n = \log_2 4 = 2$ ดังนั้นสมการที่ (2.12) เขียนได้เป็น

$$\sum_{m_0=0}^{1}\sum_{m_1=0}^{1} \tilde{h}(m_1, m_0)W^{k[2m_1+m_0]} = \sum_{m_0=0}^{1}\left\{\tilde{h}(0, m_0)W^{km_0} + \tilde{h}(1, m_0)W^{k(2+m_0)}\right\}$$

$$= \sum_{m_0=0}^{1} \tilde{h}(0, m_0)W^{km_0} + \sum_{m_0=0}^{1} \tilde{h}(1, m_0)W^{k(2+m_0)}$$

$$= \tilde{h}(0,0) + \tilde{h}(0,1)W^{k} + \tilde{h}(1,0)W^{2k} + \tilde{h}(1,1)W^{3k} \tag{2.13}$$

ดังนั้นสมการที่ (2.13) จะได้ว่า

$$\sum_{m_0=0}^{1}\sum_{m_1=0}^{1} \tilde{h}(m_1, m_0)W^{k[2m_1+m_0]} = \sum_{m=0}^{3} h(m)W^{km}$$ นั่นเอง

การพัฒนาวิธีการ (Development of the Algorithm)

สมมุติกรณีที่ $N = 8$ เป็นพื้นฐานเบื้องต้นในการอธิบายขบวนการพัฒนา FFT กล่าวคือ

$$H(k) = \frac{1}{8}\sum_{m=0}^{7} h(m)W^{km} \quad ; \quad \text{โดย} \quad k = 0,1,...,7 \tag{2.14}$$

เมื่อ $W = e^{-j\frac{2\pi}{8}} = e^{-j\frac{\pi}{4}}$ และให้ m เขียนอยู่ในรูปเลขฐานคือ

$$m = m_2 2^2 + m_1 2^1 + m_0 2^0 \tag{2.15}$$

จากสมการที่ (2.15) เอา 8 คูณตลอดจะได้

$$8H(k) = \sum_{m=0}^{7} h(m)W^{km} \tag{2.16}$$

เมื่อแปลงเทอมทางขวามือของสมการที่ (2.16) ให้ตัวแปรอยู่ในรูปของเลขฐานสองจะได้ว่า

$$8H(k) = \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\sum_{m_2=0}^{1} \tilde{h}(m_2, m_1, m_0)W^{k[4m_2+2m_1+m_0]}$$

$$= \sum_{m_0=0}^{1}\sum_{m_1=0}^{1}\sum_{m_2=0}^{1} \tilde{h}(m_2, m_1, m_0)W^{4km_2}W^{2km_1}W^{km_0} \tag{2.17}$$

จากสมการที่ (2.17) เราให้ summation ของตัวแปร m_2 ในสูตรเป็น M_2 นั้นคือ

$$M_2 = \sum_{m_2=0}^{1} \tilde{h}(m_2, m_1, m_0) W^{4km_2}$$

แต่เนื่องจาก $W^4 = \left[e^{-j\frac{\pi}{4}} \right]^4 = e^{-j\pi} = -1$ และแทน k ด้วยเลขฐานสองจะได้

$$M_2 = \sum_{m_2} \tilde{h}(m_2, m_1, m_0)(-1)^{m_2[4k_2+2k_1+k_0]}$$

จาก $(-1)^{m_2[4k_2+2k_1]} = 1, M_2$ สามารถเขียนได้ใหม่เป็น

$$M_2 = \sum_{m_2=0}^{1} \tilde{h}(m_2, m_1, m_0)(-1)^{k_0 m_2} \tag{2.18}$$

summation ของสมการที่ (2.18) เป็นการแทนค่า m_2 ด้วย 0 และ 1 ดังนั้นสมการที่ (2.18) นี้จะมีตัวแปรที่ไม่ทราบค่าคือ k_0, k_1 และ m_0 ดังนั้น M_2 จะเป็นฟังก์ชั่นของ k_0, m_1 และ m_0 จึงเขียนใหม่ได้

$$M_2 = \tilde{h}_1(k_0, m_1, m_0) \tag{2.19}$$

แทนค่าสมการที่ (2.19) ลงในสมการ (2.17)

$$8H(k) = \sum_{m_0=0}^{1} \sum_{m_1=0}^{1} \tilde{h}_1(k_0, m_1, m_0) W^{2km_1} W^{km_0} \tag{2.20}$$

summation ของตัวแปร m_1 ของสมการที่ (2.20) สมมุติให้เป็น M_1 นั่นคือ

$$M_1 = \sum_{m_1=0}^{1} \tilde{h}_1(k_0, m_1, m_0) W^{2km_1}$$

ทำการแทนตัวแปร k ซึ่งเป็นเลขฐานสิบให้อยู่ในรูปของเลขฐานสอง

$$M_1 = \sum_{m_1=0}^{1} \tilde{h}_1(k_0, m_1, m_0)(-j)^{(4k_2+2k_1+k_0)m_1} \tag{2.21}$$

โดยที่จากสมการที่ (2.21) นั้น $W^2 = \left[e^{-j\frac{\pi}{4}} \right]^2 = -j$ และเทอม $(-j)^{4k_2m_1} = 1$

ดังนั้น M_1 เขียนใหม่ได้เป็น

$$M_1 = \sum_{m_1} \tilde{h}_1(k_0, m_1, m_0)(-j)^{(2k_1+k_0)m_1} \tag{2.22}$$

จากสมการที่ (2.22) เมื่อทำการแปรค่า m_1 เป็น 0 และ 1 ดังนั้นเทอม M_1 จะเป็นตัวแปรของ k_0, k_1 และ m_0 นั่นคือสมการ (2.22) เขียนใหม่ได้เป็น

$$M_1 = \tilde{h}_2(k_0, k_1, m_0) \tag{2.23}$$

แทน (2.23) ลงใน (2.20) จะได้

$$8H(k) = \sum_{m_0} \tilde{h}_2(k_0, k_1, m_0) W^{m_0 k}$$

ในทำนองเดียวกันกับ M_2 และ M_1 เมื่อให้ M_0 เป็นผลการทำ summation ของตัวแปร m_0 นั้นคือ

$$M_0 = \sum_{m_0=0}^{1} \tilde{h}_2(k_0, k_1, m_0) W^{m_0(4k_2+2k+k_0)}$$

$$= \sum_{m_0=0}^{1} \widetilde{h}_2(k_0,k_1,m_0)\left(\frac{1-j}{\sqrt{2}}\right)^{[4k_2+2k_1+k_0]m_0} \tag{2.24}$$

หลังจากทำการแปรค่า m_0 ไปแล้วจะพบว่า M_0 เป็นฟังก์ชั่นของ k_0, k_1 และ k_2 ซึ่งจะแทนด้วย

$$M_0 = \widetilde{h}_3(k_0,k_1,k_2)$$

ดังนั้นจากสมการที่ (2.17) จะได้คำตอบว่า

$$8H(k) = 8\widetilde{H}(k_2,k_1,k_0) = \widetilde{h}_3(k_0,k_1,k_2) \tag{2.25}$$

จากสัมประสิทธิ์ $(-1),(-j)$ และ $\frac{(1-j)}{\sqrt{2}}$ ค่าเหล่านี้จะเป็นรากของ $e^{-j2\pi}$ นั้นเองกล่าวคือ

$(-1),(-j)$ และ $\frac{(1-j)}{\sqrt{2}}$ เป็นรากที่ 2,4 และ 8 ของ unity ซึ่งจะใช้สัญลักษณ์แทนด้วย

$A_2r = e^{-j\frac{2\pi}{2^r}}$ เมื่อ $r = 1,2,...\log_2 N$

โดย A_2r จะมีคุณสมบัติดังต่อไปนี้

(i) $A_2r = W\frac{N}{2^r}$ เมื่อ $W = e^{-j\frac{2\pi}{N}}$

(ii) $(A_2r)^{\lambda+l} = -(A_2r)^l$ เมื่อ $r = 1,2,...,\log_2 N; l = 0,1,...,2^{r-1}; \lambda = 2^{r-1}$

(iii) $(A_N)^{\frac{N}{2}} = 1$

จากสมการที่ (2.19) เขียนอยู่ในเทอม A_2r จะได้ว่า

$$\widetilde{h}_1(k_0,m_1,m_0) = \sum_{m_2=0}^{1} \widetilde{h}(m_2,m_1,m_0)A_2^{k_0m_2}$$

นั่นคือ $$\widetilde{h}_1(k_0,m_1,m_0) = \widetilde{h}(0,m_1,m_0) + \widetilde{h}(1,m_1,m_0)A_2^{k_0} \tag{2.26}$$

สมการที่ (2.26) นี้ k_0 จะมีค่าไม่เป็น 0 ก็เป็น 1 จะได้ว่าค่า k_0 แต่ละค่าจะให้ 4 สมการคือ

กรณีที่ 1 $k_0 = 0$ จะได้

$$\widetilde{h}_1(0,0,0) = \widetilde{h}(0,0,0) + \widetilde{h}(1,0,0) \Rightarrow h_1(0) = h(0) + h(4)$$
$$\widetilde{h}_1(0,0,1) = \widetilde{h}(0,0,1) + \widetilde{h}(1,0,1) \Rightarrow h_1(1) = h(1) + h(5)$$
$$\widetilde{h}_1(0,1,0) = \widetilde{h}(0,1,0) + \widetilde{h}(1,1,0) \Rightarrow h_1(2) = h(2) + h(6)$$
$$\widetilde{h}_1(0,1,1) = \widetilde{h}(0,1,1) + \widetilde{h}(1,1,1) \Rightarrow h_1(3) = h(3) + h(7)$$

กรณีที่ 2 $k_0 = 1$ จะได้

$$\widetilde{h}_1(1,0,0) = \widetilde{h}(0,0,0) + A_2\widetilde{h}(1,0,0) \Rightarrow h_1(4) = h(0) - h(4)$$
$$\widetilde{h}_1(1,0,1) = \widetilde{h}(0,0,1) + A_2\widetilde{h}(1,0,1) \Rightarrow h_1(5) = h(1) - h(5)$$
$$\widetilde{h}_1(1,1,0) = \widetilde{h}(0,1,0) + A_2\widetilde{h}(1,1,0) \Rightarrow h_1(6) = h(2) - h(6)$$
$$\widetilde{h}_1(1,1,1) = \widetilde{h}(0,1,1) + A_2\widetilde{h}(1,1,1) \Rightarrow h_1(7) = h(3) - h(7)$$

สมการที่ (2.22) เมื่อแทน $(-j)$ ด้วยเทอม A_4 จะได้ว่า

$$\widetilde{h}_2(k_0,k_1,m_0) = \widetilde{h}_1(k_0,0,m_0) + \widetilde{h}_1(k_0,1,m_0)A_4^{(2k+k_0)}$$

จากสมการที่ (2.22) fixed ค่า k_0, k_1 แล้วทำการแปร m_0 ไปแต่ละครั้งจะได้ 2 สมการดังนี้

กรณีที่ 1 $(k_1,k_0) = (0,0)$

$$\widetilde{h}_2(0,0,0) = \widetilde{h}_1(0,0,0) + \widetilde{h}_1(0,1,0) \Rightarrow h_2(0) = h_1(0) + h_1(2)$$
$$\widetilde{h}_2(0,0,1) = \widetilde{h}_1(0,0,1) + \widetilde{h}_1(0,1,1) \Rightarrow h_2(1) = h_1(1) + h_1(3)$$

กรณีที่ 2 $(k_1, k_0) = (0,1)$

$$\widetilde{h}_2(1,0,0) = \widetilde{h}_1(1,0,0) + A_4\widetilde{h}_1(1,1,0) \Rightarrow h_2(4) = h_1(4) + A_4 h_1(6)$$
$$\widetilde{h}_2(1,0,1) = \widetilde{h}_1(1,0,1) + A_4\widetilde{h}_1(1,1,1) \Rightarrow h_2(5) = h_1(5) + A_4 h_1(7)$$

กรณีที่ 3 $(k_1, k_0) = (1,0)$

$$\widetilde{h}_2(0,1,0) = \widetilde{h}_1(0,0,0) + A_4^2\widetilde{h}_1(0,1,0) \Rightarrow h_2(2) = h_1(0) - h_1(2)$$
$$\widetilde{h}_2(0,1,1) = \widetilde{h}_1(0,0,1) + A_4^2\widetilde{h}_1(0,1,1) \Rightarrow h_2(3) = h_1(1) - h_1(3)$$

กรณีที่ 4 $(k_1, k_0) = (1,1)$

$$\widetilde{h}_2(1,1,0) = \widetilde{h}_1(1,0,0) + A_4^3\widetilde{h}_1(1,1,0) \Rightarrow h_2(6) = h_1(4) - A_4 h_1(6)$$
$$\widetilde{h}_2(1,1,1) = \widetilde{h}_1(1,0,1) + A_4^3 h_1(1,1,1) \Rightarrow h_2(7) = h_1(5) - A_4 h_1(7)$$

ในที่สุดสมการที่ (2.25) ซึ่งเป็นการทำ iteration ครั้งที่ 2 โดย $r = 3$ เขียนค่าสัมประสิทธิ์ในเทอมของ A_8 จะได้ว่า

$$\widetilde{h}_3(k_0, k_1, k_2) = \widetilde{h}_2(k_0, k_1, 0) + \widetilde{h}_2(k_0, k_1, 1)A_8^{(4k_2+2k_1+k_0)}$$

จะได้ว่า

กรณีที่ 1 $(k_2, k_1, k_0) = (0,0,0)$

$$\widetilde{h}_3(0,0,0) = \widetilde{h}_2(0,0,0) + \widetilde{h}_2(0,0,1) \Rightarrow h_3(0) = h_2(0) + h_2(1)$$

กรณีที่ 2 $(k_2, k_1, k_0) = (0,0,1)$

$$\widetilde{h}_3(1,0,0) = \widetilde{h}_2(1,0,0) + A_8\widetilde{h}_2(1,0,1) \Rightarrow h_3(4) = h_2(4) + A_8 h_2(5)$$

กรณีที่ 3 $(k_2, k_1, k_0) = (0,1,0)$

$$\widetilde{h}_3(0,1,0) = \widetilde{h}_2(0,1,0) + A_8^2\widetilde{h}_2(0,1,1) \Rightarrow h_3(2) = h_2(2) + A_8^2 h_2(3)$$

กรณีที่ 4 $(k_2, k_1, k_0) = (0,1,1)$

$$\widetilde{h}_3(1,1,0) = \widetilde{h}_2(1,1,0) + A_8^3\widetilde{h}_2(1,1,1) \Rightarrow h_3(6) = h_2(6) + A_8^3 h_2(7)$$

กรณีที่ 5 $(k_2, k_1, k_0) = (1,0,0)$

$$\widetilde{h}_3(0,0,1) = \widetilde{h}_2(0,0,0) + A_8^4\widetilde{h}_2(0,0,1) \Rightarrow h_3(1) = h_2(0) - h_2(1)$$

กรณีที่ 6 $(k_2, k_1, k_0) = (1,0,1)$

$$\widetilde{h}_3(1,0,1) = \widetilde{h}_2(1,0,0) + A_8^5\widetilde{h}_2(1,0,1) \Rightarrow h_3(5) = h_2(4) - A_8 h_2(5)$$

กรณีที่ 7 $(k_2, k_1, k_0) = (1,1,0)$

$$\widetilde{h}_3(0,1,1) = \widetilde{h}_2(0,1,0) + A_8^6\widetilde{h}_2(0,1,1) \Rightarrow h_3(3) = h_2(2) - A_8^2 h_2(3)$$

กรณีที่ 8 $(k_2, k_1, k_0) = (1,1,1)$

$$\widetilde{h}_3(1,1,1) = \widetilde{h}_2(1,1,0) + A_8^7\widetilde{h}_2(1,1,1) \Rightarrow h_3(7) = h_2(6) - A_8^3 h_2(7)$$

ทำ bit reversal จะได้ดังตารางข้างล่างนี้

Decimal Number	Binary Representation	Bit-Reversed Representation	Decimal Equivalent
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

ตารางที่ 2.1 Bit-reversed order for $N = 8$

หลังจากทำ Bit reversal แล้วจะได้ว่า

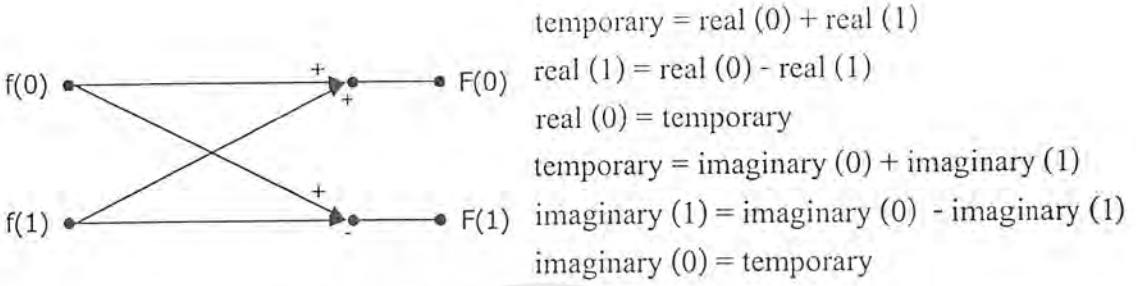
$$h_3(0) = 8H(0)$$
$$h_3(4) = 8H(1)$$
$$h_3(2) = 8H(2)$$
$$h_3(6) = 8H(3)$$
$$h_3(1) = 8H(4)$$
$$h_3(5) = 8H(5)$$
$$h_3(3) = 8H(6)$$
$$h_3(7) = 8H(7)$$

2.1.4 การคำนวณโดยใช้คอมพิวเตอร์

ในการคำนวณ โดยใช้คอมพิวเตอร์จะต้องทำการแปลงฟังชั่นเอ็กโปแนนท์เชียลให้เป็นฟังก์ชั่นตรีโกณมิติจากสูตร

$$e^{-ja} = \cos a \pm i \sin a$$

เนื่องจากโปรแกรมที่ใช้เขียนไม่รู้จักฟังชั่นเอ็กโปแนนท์เชียล ในการคำนวณจะยกตัวอย่างมาแค่บัตเตอร์ฟลายเดียว และจะต้องมีการเก็บค่าไว้ชั่วคราว ดังรูปที่ 2.4



รูปที่ 2.4 หลักการคำนวณโดยใช้บัตเตอร์ฟลายเดียว

จากรูปที่ 2.4 จะได้ว่า $F(0) = f(0) + f(1)$

$F(1) = f(0) + f(1)$

โดยข้อมูลที่ได้จะมีทั้งค่าจริงและค่าจินตภาพ

2.1.5 ตัวอย่างการคำนวณหาฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

ตัวอย่างที่ 1 จะเป็นการหาค่า 16 จุดฟาสต์ฟูเรียร์ทรานส์ฟอร์ม โดยมีการสุ่มข้อมูล 16 จุดและมีค่าสัญญาณดังนี้

$$x(n) = \cos[2\pi(4n/16)]$$

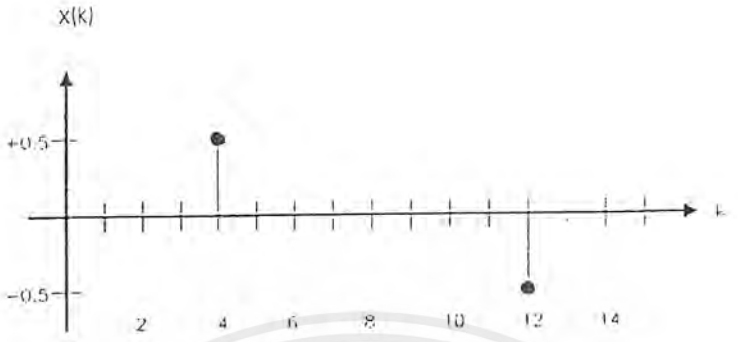
จะเปรียบเทียบจำนวนบัตเตอร์ฟลายแบบดีสครีทฟูเรียร์ทรานส์ฟอร์ม ที่ใช้ในการคำนวณ

Length of Transform (N)	DFT Operations (N^2)	FFT Operations $N \log_2 (N)$
8	64	24
16	256	64
32	1024	160
64	4096	384
128	1,6384	896

ตารางที่ 2.2

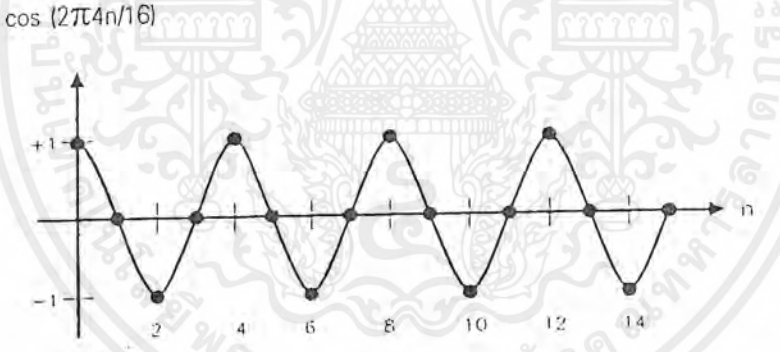
เปรียบเทียบจำนวนการคำนวณระหว่างดีสครีทฟูเรียร์ทรานส์ฟอร์มและฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

จากตัวอย่างมีค่าแอมปลิจูด "1" และสัญญาณเป็นค่าจริง ส่วนค่าจิตภาพเป็น "0" การสุ่มข้อมูลดูได้จากรูปที่ 2.5 ซึ่งมีจำนวนข้อมูล 16 จุดตั้งแต่ $x(0) - x(15)$



รูปที่ 2.5 อินพุท 16 จุดฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

โดยค่าเอาท์พุทที่ได้จะแสดงในรูปที่ 2.5 ซึ่งมีค่าอยู่ที่ $k = 4$ โดยมีขนาดแอมปลิจูด 0.5 และที่ $k = 12$ มีขนาดแอมปลิจูด -0.5



รูปที่ 2.6 เอาท์พุทของ 16 จุดฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

สำหรับสัญญาณที่ได้ในรูปของความถี่โดยใช้ฟูเรียร์ทรานส์ฟอร์ม จะได้

$$X(f) = \int_{-\infty}^{\infty} \cos(2\pi f_0 t) e^{-j2\pi ft}\, dt$$

แปลงให้อยู่ในรูปเอ็กโปเนนท์เชียล (Exponential) ได้

$$X(f) = \frac{1}{2}\int_{-\infty}^{+\infty} \exp[j2\pi(f_0 - f)t]dt - \frac{1}{2}\int_{-\infty}^{+\infty} \exp[-j2\pi(f_0 + f)t]dt$$

ผลลัพธ์จะได้ที่ $k = 4$ มีขนาดแอมปลิจูด 0.5 และที่ $k = 12$ มีขนาดแอมปลิจูด -0.5 จะมีค่า $f = f_0$ และที่ $k = 12$ จะมีค่า $f = -f_0$ ตามรูปที่ 2.5 เช่นเดียวกัน

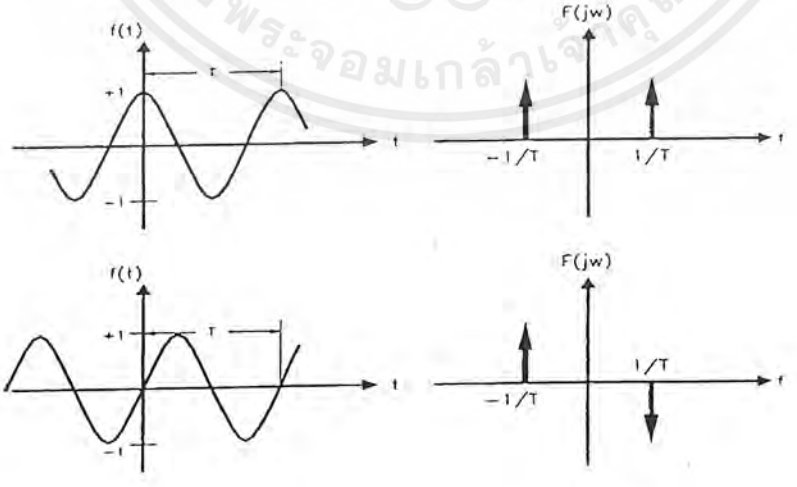
ตัวอย่างที่ 2 จงหาค่าความถี่จากสัญญาณต่อไปนี้

$$f(t) = \cos(3t) + \sin(10t)$$

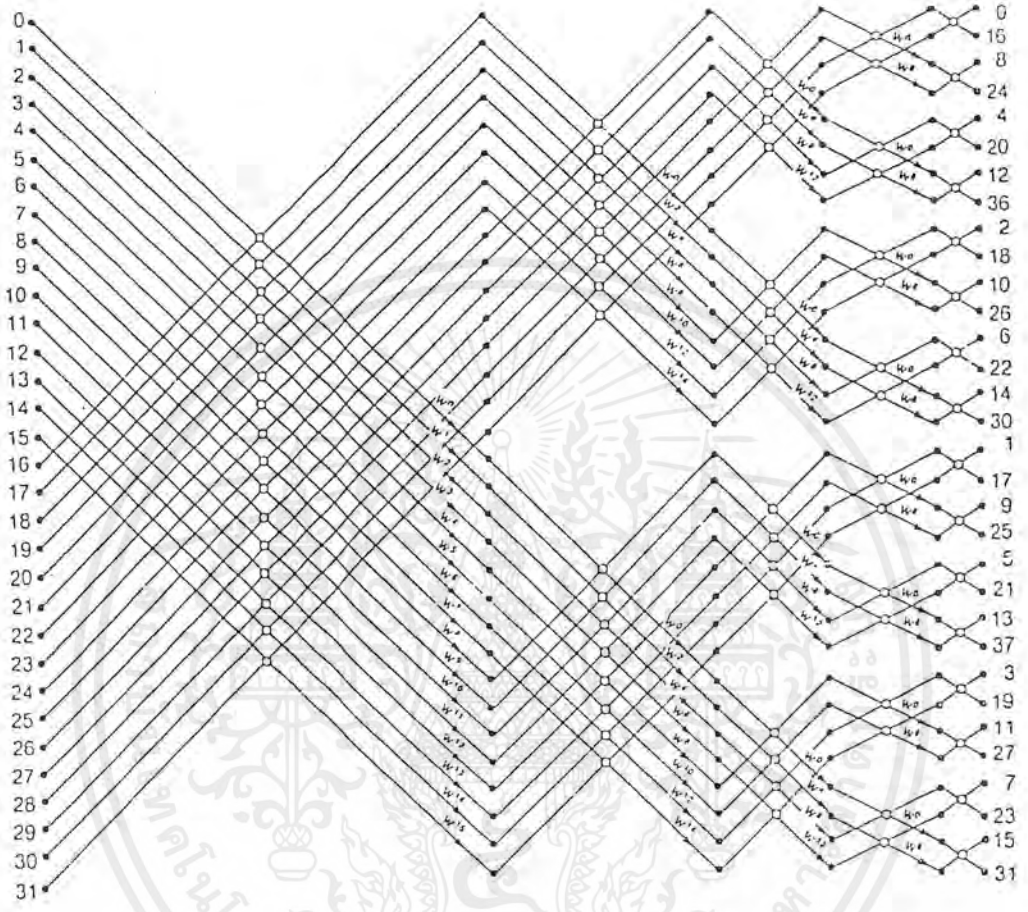
มีการสุ่มข้อมูลมา 32 จุด เวลาที่ใช้ในการสุ่มข้อมูลทั้งหมด $= 2\pi / 32$ โดยมีการสุ่มข้อมูลดังรูปที่ 2.7

รูปที่ 2.7 การสุ่มข้อมูลต่างๆจากฟังก์ชั่น cos(3t) + sin(10t)

เมื่อทำการวิเคราะห์โดยใช้วิธีฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะได้ค่า โคซายน์(cosine) มีค่าเป็นบวกอยู่ที่ตำแหน่ง +/- 1/T โดยเป็นค่าจริงอยู่บนแกนความถี่และฟังก์ชั่นซายน์(sine) จะได้ค่าบวกและลบอยู่ที่ตำ แหน่ง -1/T และ 1/T โดยจะเป็นค่าจินตภาพอยู่บนแกนความถี่ดังรูปที่ 2.8



รูปที่ 2.8 ฟังก์ชั่นซายน์,โคซายน์และผลลัพธ์ที่ได้จากฟูเรียร์ทรานส์ฟอร์ม



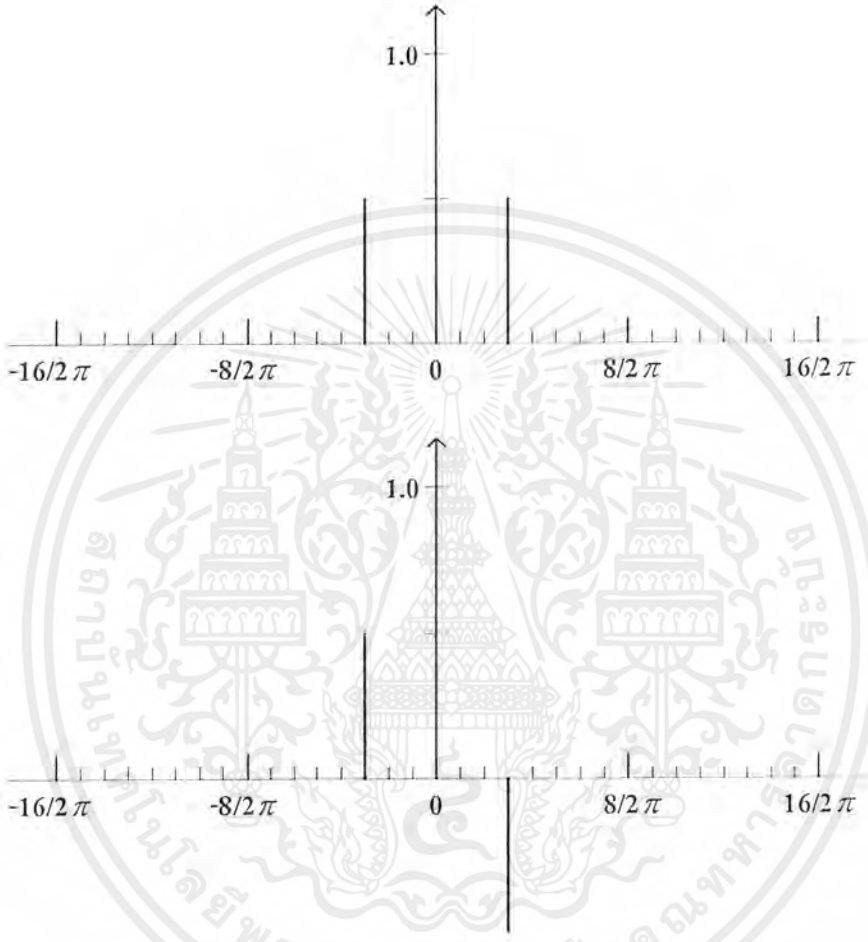
รูปที่ 2.9 วิธีการหาฟาสต์ฟูเรียร์ทรานส์ฟอร์ม 32 จุด

เมื่อคำนวณโดยใช้วิธีฟูเรียร์ทรานส์ฟอร์มจะได้ข้อมูลต่างๆดังรูปที่ 2.10

Index	f(t)	Real	Imaginary	Frequency
0	1.000	-0.000	0.000	0
1	1.755	-0.000	0.000	$1/2\pi$
2	-0.324	0.000	-0.000	$2/2\pi$
3	-0.578	0.500	0.000	$3/2\pi$
4	0.029	-0.000	-0.000	$4/2\pi$
5	-1.363	-0.000	-0.000	$5/2\pi$
6	-1.631	-0.000	-0.000	$6/2\pi$
7	0.368	-0.000	-0.000	$7/2\pi$
8	-0.000	-0.000	-0.000	$8/2\pi$
9	-0.368	-0.000	-0.000	$9/2\pi$
10	1.631	0.000	-0.500	$10/2\pi$
11	1.363	-0.000	0.000	$11/2\pi$
12	-0.293	-0.000	0.000	$12/2\pi$
13	0.578	-0.000	0.000	$13/2\pi$
14	0.324	-0.000	0.000	$14/2\pi$
15	-1.755	-0.000	0.000	$15/2\pi$
16	-1.000	-0.000	0.000	$-16/2\pi$ (Nyquist Frequency)
17	0.092	-0.000	-0.000	$-15/2\pi$
18	-1.090	-0.000	-0.000	$-14/2\pi$
19	-0.188	-0.000	-0.000	$-13/2\pi$
20	1.707	-0.000	-0.000	$-12/2\pi$
21	0.598	-0.000	-0.000	$-11/2\pi$
22	0.217	0.000	0.500	$-10/2\pi$
23	1.479	-0.000	0.000	$-9/2\pi$
24	0.000	-0.000	0.000	$-8/2\pi$
25	-1.479	-0.000	0.000	$-7/2\pi$
26	-0.217	-0.000	0.000	$-6/2\pi$
27	-0.598	-0.000	0.000	$-5/2\pi$
28	-1.707	-0.000	0.000	$-4/2\pi$
29	0.188	0.500	-0.000	$-3/2\pi$
30	1.090	0.000	0.000	$-2/2\pi$
31	-0.092	-0.000	-0.000	$-1/2\pi$

ตารางที่ 2.3 ข้อมูลต่างๆที่ได้จากการคำนวณโดยวิธีฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

จากข้อมูลที่ได้ที่ตำแหน่ง $k = 16 - 32$ จะเป็นค่าความถี่ทางด้านลบ และที่ตำแหน่ง $k = 0$-15 จะเป็นค่าความถี่ทางด้านบวก โดยผลลัพธ์ที่ได้จะแสดงในรูปที่ 2.10 ซึ่งมีค่าจริงเป็นบวกที่ $f = \pm 3/2\pi$ และค่าจินตภาพเป็นค่าบวกและค่าลบที่ $f = -10/2\pi$ และ $f = 10/2\pi$ ตามลำดับ



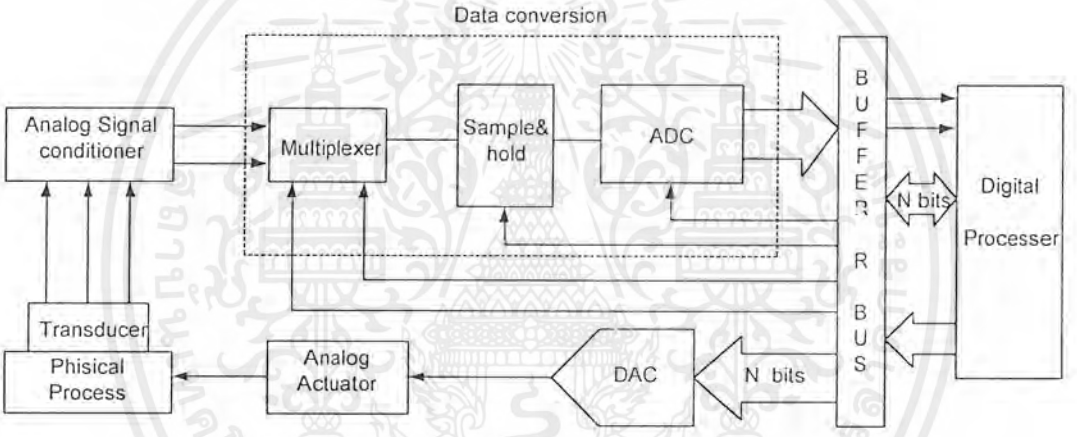
รูปที่ 2.10 การพล็อตค่าต่างๆ บนแกนความถี่

ในการหาค่าฟาสต์ฟูเรียร์ทรานส์ฟอร์มโดยใช้วิธีการแบบบัตเตอร์ฟลายนั้นจะใช้วิธีหารหาค่าต่างๆ ตามรูปที่ 2.4 โดยจะต้องรู้ว่าสุ่มสัญญาณมากี่จุดและต้องหาค่า W มาให้ครบทุกค่า ซึ่งจะเป็นวิธีการที่ยุ่งยาก เพราะว่าจะมีบัตเตอร์ฟลายหลายตัว

2.2 การแปลงสัญญาณอนาล็อกเป็นดิจิตอล (ANALOG TO DIGITAL CONVERSION)

รูปแบบสัญญาณไฟฟ้าที่เราพบเห็นและคุ้นเคยในชีวิตประจำวันจะอยู่ในรูปของสัญญาณที่ต่อเนื่องหรือที่เรียกว่า สัญญาณอนาล็อก (Analog Signal) ซึ่งเดิมการนำเอาสัญญาณไฟฟ้าดังกล่าวมาประมวล (Process) เพื่อให้มีรูปแบบที่เหมาะสมจะกระทำในแบบอนาล็อกนั่นเอง แต่เมื่อเทคนิคและ

อุปกรณ์การประมวลสัญญาณทางสัญญาณดิจิตอลได้รับการพัฒนาขึ้นมา เนื่องจากพบว่าในรูปแบบดิจิตอล การประมวล เก็บ สื่อสาร และการนำเสนอกระทำได้ง่ายและอย่างมีประสิทธิภาพมากกว่า ดังนั้นการเปลี่ยนรูปแบบของสัญญาณ (conversion) จึงได้มีความจำเป็นขึ้นมา ในรูปที่ 2.11 เป็นตัวอย่างแสดงระบบควบคุมที่ใช้ในการประมวลข้อมูลในระบบดิจิตอลในระบบที่ยกมาเป็นตัวอย่างนี้การเปลี่ยนแปลงทางกายภาพในลักษณะใดๆก็ตาม (physical process) เช่น ความดัน อุณหภูมิ ฯลฯ จะถูกเปลี่ยนเป็นสัญญาณไฟฟ้าที่มีความต่อเนื่อง (สัญญาณอนาล็อก) โดยการใช้ตัวทรานสดิวเซอร์ที่มีคุณสมบัติเหมาะสมกับรูปแบบทางกายภาพนั้น สัญญาณไฟฟ้าจะถูกปรับให้อยู่ในรูปและขนาดที่เหมาะสมก่อนโดย Analog signal conditioner ซึ่งอาจจะเป็นวงจรขยาย หรือ ฟิลเตอร์ เป็นต้น ADC จะทำหน้าที่เปลี่ยนรูปแบบของสัญญาณจากอนาล็อกเป็นดิจิตอล ตัวประมวลผลทางดิจิตอล (Digital processors) เช่น คอมพิวเตอร์จะจัดการกับข้อมูลเพื่อนำเสนอหรือถูกเปลี่ยนกลับมาอยู่ในรูปแบบอนาล็อกโดย DAC เพื่อป้อนกลับไปควบคุม Physical Process



รูปที่ 2.11 ระบบควบคุมที่มีการประมวลข้อมูลแบบดิจิตอล

ในระบบที่มีข้อมูลที่ต้องประมวลในเวลาเดียวกันหลายข้อมูล หาก ADC ทำงานได้เร็วพอจะไม่จำเป็นต้องใช้ ADC หลายๆตัวทำงานแยกกันสำหรับข้อมูลแต่ละชุด แต่จะใช้วิธีแบ่งเวลา (timesharing) โดยวิธี multiplexing (รูปที่ 2.11) วงจร sampling and hold (S/H) จะสุ่ม (sample) ขนาดของสัญญาณอนาล็อกมาและเก็บ(hold) ไว้ชั่วขณะเพื่อรอให้ ADC รับเปลี่ยนให้เป็นสัญญาณดิจิตอลจนเรียบร้อยแล้วค่อยสุ่มสัญญาณใหม่ ทั้งนี้เพื่อที่ไม่จำเป็นต้องใช้ ADC ที่มีความเร็วสูงราคาแพง ข้อมูลดิจิตอลจะถูกส่งต่อไปยัง System bus และถูกประมวลโดย Processor ผลของการประมวลจะถูกส่งกลับออกมาเพื่อเปลี่ยนกลับเป็นสัญญาณอนาล็อกโดยDACเพื่อไปควบคุมกิจกรรมทางกายภาพของระบบผ่าน Analog actuator

วงจร A/D มีด้วยกันหลายแบบ แต่ที่นิยมใช้กันแพร่หลายมี 3 แบบคือ

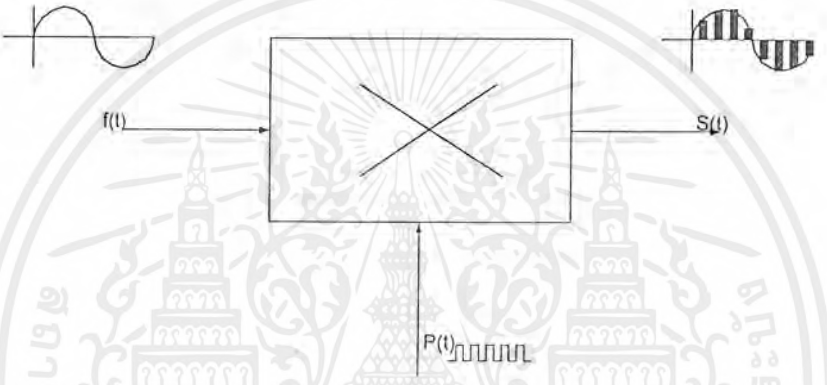
- แบบสโลปคู่(dual slope)
- แบบแปลงแรงดันเป็นความถี่(V to F converter)
- แบบประมาณทีละบิต(successive approximation)

2.2.1 รูปแบบการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

การแปลงสัญญาณอนาล็อกเป็นดิจิตอลประกอบด้วยขบวนการสำคัญ ๆ 3 ส่วนคือ

1) การแซมปลิ้ง (Sampling)
2) การควอนไตซิ่ง (Quantizing)
3) การเข้ารหัส (Encoder)

ซึ่งส่วนประกอบที่สำคัญที่สุดคือการแซมปลิ้ง เพราะความผิดพลาดของสัญญาณดิจิตอลที่แปลงมาจากสัญญาณอนาล็อกนั้นจะมีมากหรือน้อยขึ้นอยู่กับความสัมพันธ์ของความถี่แซมปลิ้งกับความถี่สูงสุดของสัญญาณอนาล็อก โดยมีทฤษฎีการแซมปลิ้ง (Sampling Theory) ซึ่งความสัมพันธ์ตามทฤษฎีการแซมปลิ้งดูได้จากรูปที่ 2.12



รูปที่ 12 รูปแสดงการแซมปลิ้ง

จากรูปจะได้ $S(t) = P(t)*f(t)$

เมื่อดู P(t) ซึ่งเป็นพัลส์สี่เหลี่ยมและถ้านำมาเขียนสมการฟูเรียร์ (Fourier) จะได้

$$P(t) = DC + a_0\cos \varpi_0 t + a_1\cos 3\varpi_0 t + a_2\cos 5\varpi_0 t + \ldots$$

ซึ่ง P(t) ประกอบด้วยความถี่พื้นฐานรวมกับฮาโมนิคที่เป็นเลขคี่ไปจนถึง ∞ และถ้านำค่าของ P(t) คูณด้วย f(t) จะได้ S(t)

$$S(t) = f(t)*DC + (a_0\cos \varpi_0 t)*f(t) + (a_1\cos 3\varpi_1 t)*f(t) + \ldots$$

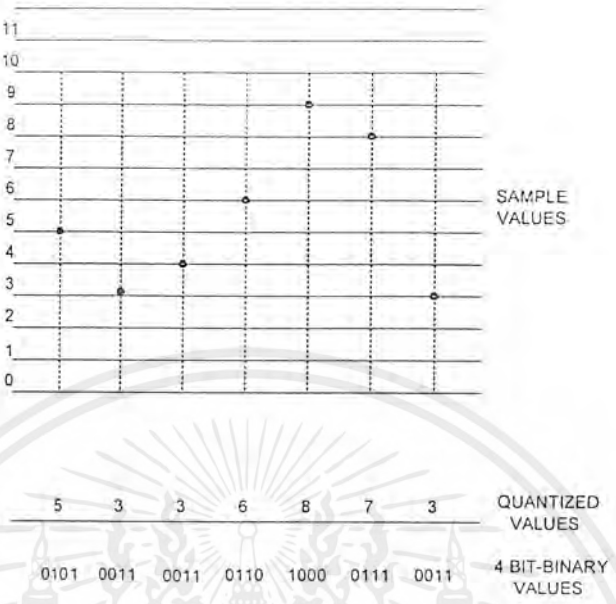
เมื่อพิจารณาดูเทอมที่ 2 จะพบว่ามีรูปแบบเหมือนแอมปลิจูดมอดูเลชั่น (AM)

โดยถ้า $f(t) = B\cos \varpi_m t$

$$f(t)*a_0\cos \varpi_0 t = (Ba_0/2)\cos(\varpi_0 - \varpi_m)t - (Ba_0/2)\cos(\varpi_0 + \varpi_m)t$$

ซึ่งความถี่ ϖ_0 ที่ใช้สำหรับการแซมปลิ้งและการดีเทคสัญญาณที่ได้คืนมาจะใช้วงจรกรองความถี่ต่ำ (Low Pass Filter) กรองเฉพาะ f(t)*DC ออกมาเท่านั้น ซึ่งถ้าค่า ϖ_0 มีค่าน้อยกว่า 2 เท่าของ ϖ_m แล้วจะทำให้มีความถี่ซึ่งเป็นผลต่างของ $\varpi_0 - \varpi_m$ เข้ามาแทรกใน f(t)*DC ด้วย ซึ่งจะมีผลทำให้สัญญาณที่ดีเทคกลับคืนมามีความผิดพลาด ดังนั้นจึงต้องทำการเลือก ϖ_0 หรือความถี่แซมปลิ้งให้มีค่ามากกว่า 2 เท่าของความถี่สูงสุดก่อนที่จะมีการแซมปลิ้ง หรือ ϖ_0 ซึ่งสัญญาณที่ได้ออกมาจากการแซมปลิ้งนั้น

จะเรียกว่า"พัลส์แอมปลิจูดมอดูเลชั่น (Pulse Amplitude Modulation :PAM)"

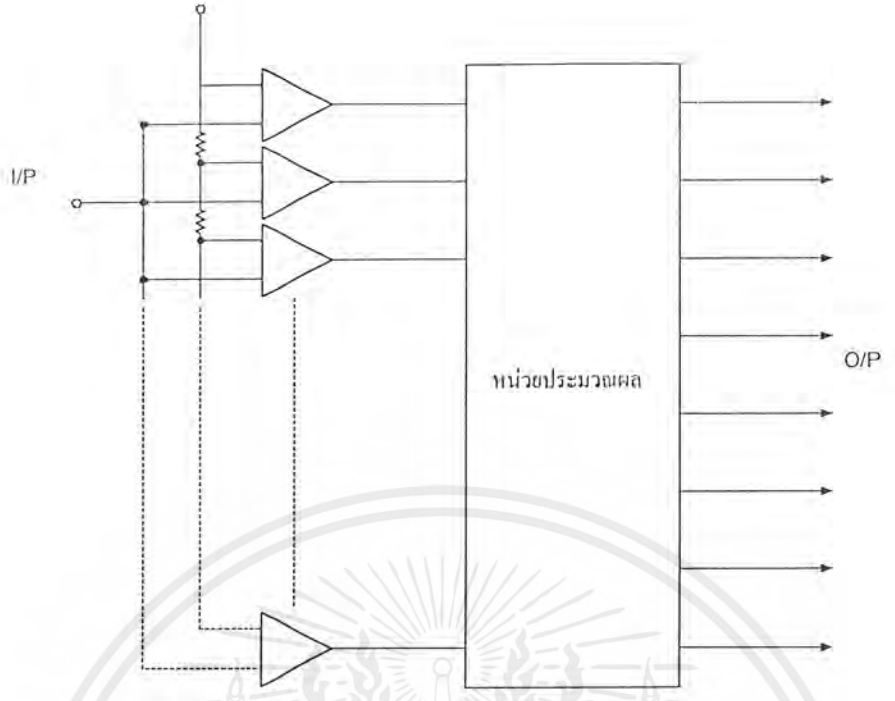


รูปที่ 2.13 รูปแสดงลักษณะสัญญาณของภาคควอนไตซิ่งและการเข้ารหัส

ภาคควอนไตซิ่ง (Quantizing) เป็นการจัดระบบของสัญญาณ PAM ซึ่งอาจจะมีระดับของสัญญาณที่ไม่แน่นอนให้ไปอยู่ในระดับที่แน่นอน ซึ่งในขั้นตอนนี้จะมีความผิดพลาดจากการจัดระดับอยู่ เรียกว่า ควอนไตซิ่งเออเรอร์ หรือสัญญาณรบกวนควอนไตซิ่ง (Quantizing Error , Quantizing Noise) ซึ่งจะมีค่ามากหรือน้อยขึ้นอยู่กับระดับของสัญญาณที่จะแบ่ง ซึ่งลักษณะของสัญญาณภาคควอนไตซิ่งและข้อผิดพลาดของสัญญาณเมื่อได้รับควอนไตซิ่งแล้วก็จะนำไปเข้าสู่วง-จรเข้ารหัสให้เป็นสัญญาณดิจิตอลซึ่งจะมีค่า 2 ระดับคือ 0 กับ 1 เท่านั้น ถ้าการส่งข้อมูลเป็นแบบ 8 บิท เพราะฉะนั้นจะส่งข้อมูลได้ถึง $2^8 = 256$ ระดับ โดยกำหนดให้ระดับต่ำสุดของสัญญาณควอนไตซิ่ง (Quantizing Signal) เท่ากับ 1111 1111 ดังนั้นข้อมูลที่ออกมาจะเป็นสัญญาณดิจิตอลที่มีค่าตามระดับที่ตั้งไว้

2.2.2 แฟลชอนาล็อกทูดิจิตอลคอนเวอร์เตอร์ (Flash A/D)

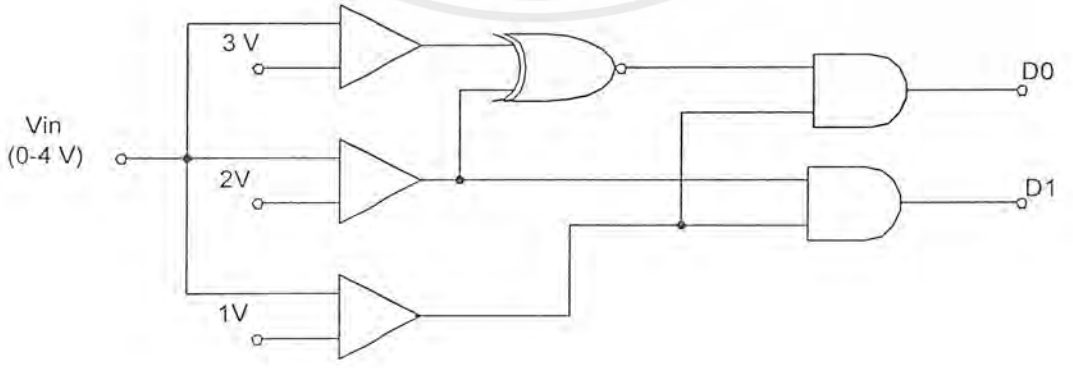
แฟลชอนาล็อกทูดีจิตอลคอนเวอร์เตอร์ คือ วงจรที่เปลี่ยนสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอล (A/D Converter) ที่มีความเร็วสูงในการเปลี่ยน เนื่องจากแฟลสเอทูดีจะใช้การโปรแกรมเอาท์พุทไว้ก่อนแล้ว ส่วนวงจรเปรียบเทียบในใช้ออปแอมป์ที่มีจำนวนเท่ากับจำนวนของสัญญาณเอาท์พุทที่เกิดขึ้นเป็นสัญญาณดิจิตอล 8 บิท ซึ่งจะต้องใช้ออปแอมป์ $2^8 = 256$ ตัวโดยที่แต่ละตัวจะมีแรงดันอ้างอิงที่เป็นค่าคงที่อยู่ค่าหนึ่งตามระดับของสัญญาณดังรูปที่ 2.14



รูปที่ 14 รูปแสดงวงจร Flash A/D

เมื่อมีสัญญาณอินพุทเข้ามาสัญญาณที่เข้ามาก็จะถูกส่งต่อไปยังออปแอมป์ทุกตัวเพื่อทำการเปรียบเทียบกับแรงดันอ้างอิงของแต่ละตัว ถ้าสัญญาณอินพุทที่เข้ามาไปตรงกับออปแอมป์ตัวใดก็จะทำให้มีสัญญาณออกมาที่เอาท์พุทส่งไปให้วงจรประมวลผล เพื่อที่จะจัดหาค่าของสัญญาณดิจิตอล สัญญาณเอาท์พุทให้ได้ตามค่าของสัญญาณอินพุทที่ส่งเข้ามา ซึ่งการเปลี่ยนสัญญาณในรูปแบบนี้ไม่ต้องใช้วงจรนับแล้วป้อนกลับมาเปรียบเทียบทีละค่า จึงทำให้ความเร็วในการเปลี่ยนสัญญาณสูงมาก

บางครั้งจะเรียกแฟลชเอทูดีว่า Open Loop Converter เนื่องจากไม่มีสัญญาณป้อนกลับไปยังอินพุท ตัวอย่างดังรูปที่ 2.15



รูปที่ 2.15 แสดงหลักการของ Open Loop Converter

จากรูปที่ 2.15 จะใช้โลจิกเกทเป็นวงจรที่เปลี่ยนระดับของสัญญาณอินพุทให้เป็นสัญญาณดิจิตอล โดยเริ่มจากเมื่อมีอินพุท 0 โวลท์ เข้ามาจะทำให้เอาท์พุททุกตัวของคอมพาราเตอร์ (Comparator) เป็น 0 หมด และเมื่อผ่านเอ็กคลูซีฟนอร์เกท (Exclusive-Nor Gate) จะมีผลทำให้เอาท์พุทเป็น 1 ซึ่งจะไปเข้า AND เกทเบอร์ 1 เป็น 0 $D_0 = 0$ และ AND เบอร์ 2 จะมีเอาท์พุทเป็น 00 ดังนั้น $D_1 = 1$ เพราะฉะนั้นระดับ 0 โวลท์เอาท์พุทของเอทูดีจะเท่ากับ 00 เมื่อ $V_{in} = 1$ โวลท์จะทำให้เอาท์พุทของคอมพาราเตอร์ตัวที่ 1 เป็น 1 ไป AND กับเอาท์พุทของเอ็กคลูซีฟนอร์เกทซึ่งเป็น 1 ทำให้ได้ $D_0 = 1$ ส่วน D_1 ได้จากการ AND กันของเอาท์พุทคอมพาราเตอร์ตัวที่ 1 กับตัวที่ 2 เป็น 1 0 ดังนั้นเอาท์พุท $D_1 = 0$ เพราะฉะนั้นที่ระดับ $V_{in} = 2$ โวลท์จะทำให้เอาท์พุทของคอมพาราเตอร์ตัวที่ 1 กับตัวที่ 2 เป็น 1 ทำให้เอาท์พุทของ $D_1 = 1$ และ $D_2 = 0$ เพราะฉะนั้นเอาท์พุทของเอทูดีจึงเท่ากับ 1 0 และเมื่อ $V_{in} = 3$ โวลท์จะทำให้เอาท์พุทของคอมพาราเตอร์ทุกตัวเป็น 1 ดังนั้นเมื่อผ่านโลจิกเกทจะได้เอาท์พุทเป็น 1 1

ข้อดีของวงจรแฟลชเอทูดีคือ มีความสามารถในการเปลี่ยนสัญญาณได้เร็วมาก

ข้อเสียของวงจรแฟลชเอทูดีคือ จะต้องใช้จำนวนของคอมพาราเตอร์จำนวนมากเป็น 2 เท่าเมื่อต้องการเพิ่ม 1 บิท หรือถ้าเป็นสมการจะได้จำนวนออปแอมป์เท่ากับ 2^{N-1} ตัวโดย N คือจำนวนบิท ดังนั้นเมื่อเราใช้สัญญาณดิจิตอล 8 บิท เราจะต้องใช้คอมพาราเตอร์ถึง 256 ตัว ซึ่งจะทำให้อุปกรณ์มีราคาสูงมาก

ในโครงงานนี้จึงใช้แฟลชเอทูดีเบอร์ CA3318C ที่มีค่าควอนไตซิ่ง 256 ระดับ (8 บิท) และมีความถี่ในการแซมปลิ้งสูงถึง 15 MHz

2.3 วงจรตามศักดาสัญญาณ (Voltage Follower)

วงจรในรูปที่ 2.16 จะปฏิบัติตามศักดาสัญญาณ นั่นคือศักดาสัญญาณออกจะเปลี่ยนแปลงตามสัญญาณเข้าทุกประการ

รูปที่ 2.16 แสดงวงจรตามศักดาสัญญาณ

บางครั้งอาจจะเรียกวงจรดังกล่าวนี้ว่า เป็นวงจรค่าขยายสัญญาณหนึ่ง(Unity gain amplifier)หรือวงจรสะเทินสัญญาณ(Buffer amplifier) สัญญาณเข้า Ei จะจ่ายเข้าโดยตรงยังขาสัญญาณเข้า (+) และเนื่องจากศักดาระหว่างขาสัญญาณเข้า (+) และ (-) จะถือได้ว่าเป็น 0 และจากรูปที่ 2.16 เราจะเห็นว่าขาสัญญาณออกได้ต่ออยู่กับขาสัญญาณเข้า (-) ดังนั้นศักดาสัญญาณออกเท่ากับศักดาสัญญาณเข้าทั้งระดับและเครื่องหมายส่วนค่าขยายสัญญาณเท่ากับหนึ่ง

$$Vo = Ei$$

$$A_{CL} = Vo / Ei = 1$$

อิมพีแดนซ์จุดสัญญาณเข้า ซึ่งมองเข้าไปที่จุดสัญญาณเข้า (+) จะมีค่าสูงสุดมาก(หลายเม็กกะโอห์ม)ดังนั้นจุดสัญญาณออกและจุดสัญญาณเข้าจึงถูกแยกออกจากกันอย่างอิสระ

2.4 ช่วงตอบสนองต่อความถี่ของออปแอมป์

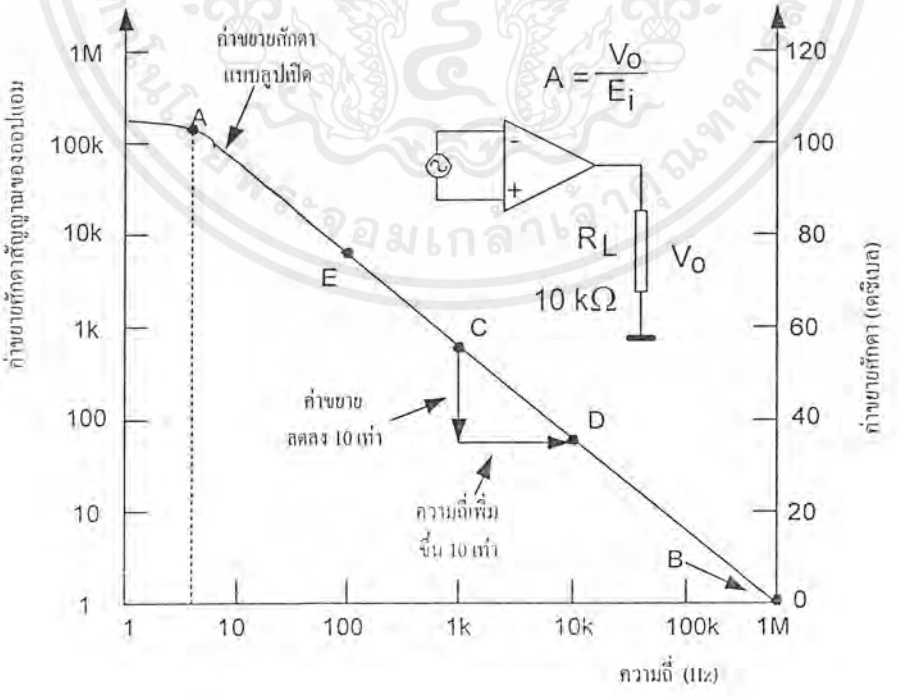
2.4.1 การชดเชยความถี่ภายในวงจรออปแอมป์

ออปแอมป์หลายแบบมีการชดเชยความถี่ภายในตัวออปแอมป์ นั่นคือผู้ผลิตได้สร้างความจุไฟฟ้าค่าเล็กประมาณ 30 ไมโครฟาราดไว้ในตัวออปแอมป์ และความจุไฟฟ้าชดเชยความถี่ดังกล่าวนี้จะคอยกันไม่ให้ออปแอมป์ ออสซิลเลทที่ความถี่สูง การออสซิลเลทนี้จะทำได้โดยการลดค่าขยายแบบลูปเปิดของออปแอมป์ที่ความถี่สูง มิเช่นนั้นแล้วที่ความถี่สูงๆจะมีจุด ซึ่งทั้งค่าขยายและค่าเลื่อนเฟส และระดับสัญญาณป้อนกลับมีเพียงพอที่จะทำให้ออปแอมป์ออสซิลเลทได้

จากทฤษฎีของวงจรเบื้องต้น เราทราบว่ารีแอคแตนซ์ของความจุไฟฟ้าจะลดลงเมื่อความถี่เพิ่มขึ้น ในลักษณะ $X_C = 1/(2\pi f_c)$นั่นคือเมื่อความถี่เพิ่มขึ้น 10 เท่ารีแอคแตนซ์จะลดลง 10 เท่าความถี่ที่เปลี่ยนไป 10 เท่าจะเรียกว่าหนึ่งดีเขต ผู้ผลิตจะแสดงความสัมพันธ์ระหว่างค่าขยายศักดาสัญญาณแบบลูปเปิด และความถี่ในลักษณะของกราฟ ซึ่งจะเรียกว่ากราฟการตอบสนองต่อความถี่สำหรับสัญญาณระดับต่ำ

2.4.2 กราฟการตอบสนองต่อความถี่ภายในออปแอมป์

รูปที่ 2.17 เช่นออปแอมป์แบบ 741 และ 747 ที่ความถี่ต่ำมากๆ (ต่ำกว่า 0.1 เฮิร์ต) ค่าขยายศักดาสัญญาณแบบลูปเปิดจะมีค่าสูงมาก โดยทั่วไปจะมีค่าประมาณ 200,000 เท่าหรือ 106 เดซิเบล ในกรณีที่ผู้ผลิตไม่ได้แสดงกราฟการตอบสนองต่อความถี่ เขามักจะแจ้งค่าขยายศักดาสัญญาณดังกล่าวนี้แทน



รูปที่ 2.17 กราฟการตอบสนองต่อความถี่

จุด A ในรูปที่ 2.17 จะเป็นจุดแยกความถี่(Break frequency point) ณ. จุดนี้ค่าขยายศักดาสัญญาณจะมีค่าประมาณ 0.707 เท่าของค่าที่ความถี่ต่ำมากๆดังนั้น ค่าขยายศักดาสัญญาณที่จุด A จะเท่ากับ 0.707x 200,000 หรือประมาณ 140,000 เท่า

จุด C และ D แสดงว่าค่าขยายสัญญาณจะลดลง 10 เท่า เมื่อความถี่เพิ่มขึ้น 10 เท่าหรืออาจจะกล่าวได้ว่าค่าขยายสัญญาณจะลดลง 1 ดีเขต ต่อความถี่เพิ่มขึ้น 1 ดีเขตแกนขวามือในรูปที่10แสดงค่าขยายศักดาสัญญาณในหน่วยเดซิเบล ดังนั้นจะเห็นได้ว่าค่าขยายสัญญาณจะลดลงในอัตรา 20 เดซิเบลต่อ 1 ดีเขตของการเพิ่มของสัญญาณ หรือในบางกรณี เราอาจจะใช้ในหน่วยออคเทป(2 เท่า) ได้ว่าค่าขยายศักดาสัญญาณจะลดลงในระดับ 6 เดซิเบลต่อความถี่เพิ่มขึ้น 1 ออคเทปแบนด์วิดเมื่อค่าขยายเท่ากับหนึ่ง

จุด B แทนกราฟของรูปที่ 2.17 เป็นจุดกำหนดแบนวิด ณ จุดค่าขยายเท่ากับหนึ่งของออปแอมป์นั่นคือจะอยู่ ณ จุดความถี่ที่ค่าขยายศักดาสัญญาณแบบลูปเปิดเท่ากับหนึ่ง

ในบางกรณีผู้ผลิตจะไม่แจ้งแบนด์วิดค่าขยายหนึ่ง หรือกราฟของการตอบสนองต่อความถี่ แต่จะแจ้งเวลาตอบสนองต่ำสุดในการเพิ่มระดับสัญญาณออก เมื่อค่าขยายสัญญาณลูปเปิดเท่ากับหนึ่ง (Transient response rise time at unity gain) สำหรับออปแอมป์แบบ 741 จะมีเวลาตอบสนองดังกล่าวอยู่ในช่วง 0.25 ไมโครวินาที ถึง 0.8 ไมโครวินาที ดังนั้นแบนด์วิด B จะคำนวณได้จาก

$$B = 0.35/(\text{Rise Time})$$

โดย B หน่วยเป็นเฮิร์ต และเวลาเพิ่มระดับหรือ Rise Time มีหน่วยเป็นวินาที เราจะได้นิยามของการเพิ่มระดับให้รัดกุมกว่านี้ในหัวข้อย่อยที่ 2.4.3

ตัวอย่างที่ 1 ถ้าออปแอมป์แบบ 741 มีเวลาเพิ่มระดับหรือ Rise Time เท่ากับ 0.35 ไมโครวินาที จงคำนวณหาแบนด์วิดเมื่อค่าขยายเท่ากับหนึ่ง

วิธีทำ

จากสมการเราจะได้ $B = 0.35/0.35\ \mu\text{sec}$

$= 1\ MHz$

ตัวอย่างที่ 2 จงคำนวณหาค่าขยายศักดาสัญญาณแบบลูปเปิด ณ จุดความถี่ 100 กิโลเฮิร์ตสำหรับออปแอมป์ในตัวอย่างที่ 1

วิธีทำ จากรูปที่ 10 จะเห็นได้ว่าเมื่อความถี่ลดลง 10 เท่าเนื่องจากค่าขยายสัญญาณเท่ากับหนึ่ง เมื่อความถี่เท่ากับ 1 เม็กกะเฮิร์ตดังนั้น ณ จุดความถี่ 100 กิโลเฮิร์ต ค่าขยายศักดาสัญญาณจะเท่ากับ 10

ตัวอย่างที่ 3 จงคำนวณหาค่าขยายศักดาสัญญาณแบบลูปเปิด สำหรับออปแอมป์ซึ่งมีแบนด์วิดเมื่อมีค่าขยายสัญญาณเท่ากับหนึ่ง เท่ากับ 15 เม็กกะเฮิร์ต ณ จุดความถี่ 1 กิโลเฮิร์ต

วิธีทำ จากตัวอย่างที่ 2 เราอาจสรุปได้ว่า เมื่อหารแบนด์วิดค่าขยายหนึ่งด้วยความถี่ของสัญญาณเราจะได้ค่าขยายลูปเปิด ณ ความถี่นั้นๆ นั่นคือ

ค่าขยายลูปเปิด ณ f = แบนด์วิด / ความถี่ของสัญญาณเข้า

ดังนั้นในกรณีปัจจุบันนี้ เราจะได้ค่าขยายศักดาสัญญาณลูปเปิด ณ ความถี่ 1 กิโลเฮิร์ต เท่ากับ

$$1.5\ MHz / 1\ kHz = 1500$$

โดยทั่วไปสำหรับออปแอมป์ราคาถูก ผู้ผลิตจะกำหนดค่าขยายศักดาสัญญาณแบบลูปเปิด โดยเฉลี่ยเท่ากับ 200,000 แต่ออปแอมป์แต่ละตัวอาจจะมีค่าขยายลดลงได้ 10 เท่า คือเท่ากับ 20,000 โดยทั่วไปผู้ผลิตจะรับรองว่าไม่มีออปแอมป์ตัวใด ค่าขยายต่ำกว่า 20,000 ค่าขยายศักดาสัญญาณ แบบลูปเปิดเพียง 20,000 ก็เพียงพอต่อการใช้งานในกรณีทั่วๆไปแล้ว

2.4.3 เวลาเพิ่มระดับ

ถ้าสมมุติว่าระดับศักดาสัญญาณเข้าของวงจรตามสัญญาณ ซึ่งมีค่าขยายศักดาสัญญาณเท่ากับหนึ่งเปลี่ยนระดับอย่างรวดเร็วในลักษณะสัญญาณรูปสี่เหลี่ยมเช่น อาจจะเปลี่ยนระดับจากศูนย์โวลท์ถึง + 20 มิลลิโวลท์ในเวลาศูนย์วินาที แต่ในทางปฏิบัติ การเปลี่ยนระดับของสัญญาณเข้าอาจจะใช้เวลาหลายนาโนวินาที สัญญาณออกของออปแอมป์ก็ควรเปลี่ยนระดับจากศูนย์โวลท์ไปเป็น +20 มิลลิโวลท์เช่นกัน แต่ลักษณะการเปลี่ยนระดับของสัญญาณออกจะล้าหลังศักดาสัญญาณเข้า ดังนั้นเราจึงกำหนดนิยามของเวลาของการเพิ่มระดับไว้ว่าคือเวลาที่ศักดาสัญญาณออกใช้ในการเพิ่มระดับจากค่า 10 เปอร์เซนต์ของระดับสุดท้ายจนถึง 90 เปอร์เซ็นต์ ของระดับสุดท้ายสำหรับออปแอมป์แบบ741 มีเวลาเพิ่มระดับเท่ากับ 0.35ไมโครวินาที นั่นคือศักดาสัญญาณออกจะใช้เวลา 0.35 ไมโครวินาที ในการเพิ่มระดับจาก 2 มิลลิโวลท์ ถึง 18 มิลลิโวลท์

2.4.4 อัตราสลูและศักดาสัญญาณออก

นิยามของอัตราสลู

อัตราสลูของออปแอมป์จะเป็นตัวกำหนดอัตราสูงสุด ซึ่งศักดาสัญญาณออกจะเปลี่ยนระดับได้ สำหรับออปแอมป์แบบใช้งานทั่วไป เช่นแบบ 741 อัตราสลูจะเท่ากับประมาณ 0.5 โวลท์/ไมโครวินาที นั่นคือระดับศักดาออกจะเปลี่ยนค่าได้สูงสุดเพียง 0.5 โวลท์ในเวลา 1 ไมโครวินาที อัตราสลูจะขึ้นอยู่กับหลายสิ่ง คือค่าขยายสัญญาณของวงจรค่าความจุไฟฟ้าชดเชย และแม้กระทั่งว่าระดับศักดาสัญญาณเข้ากำลังเพิ่มหรือลดระดับ อัตราสลูค่าต่ำสุดจะเกิดขึ้น เมื่อค่าขยายสัญญาณของวงจรเท่ากับหนึ่ง ดังนั้นผู้ผลิตออปแอมป์มักจะแจ้งอัตราสลู เมื่อค่าขยายเท่ากับหนึ่ง

สาเหตุที่จำกัดอัตราสลู

โดยทั่วไป ภายในออปแอมป์ (หรือบางกรณีอาจจะอยู่ภายนอก) จะต้องมีความจุไฟฟ้าอย่างน้อยหนึ่งตัวเพื่อป้องกันการออสซิลเลทของออปแอมป์ วงจรของออปแอมป์ซึ่งต่ออยู่กับความจุไฟฟ้าดังกล่าวนี้ จะจ่ายกระแสผ่านความจุไฟฟ้าได้สูงสุดค่าหนึ่งขึ้นอยู่กับลักษณะการออกแบบของวงจร อัตราส่วนของกระแสสูงสุด I และค่าความจุไฟฟ้าชดเชย จะเท่ากับอัตราสลู ตัวอย่างเช่น ออปแอมป์แบบ 741 จะจ่ายกระแสสูงสุดเพื่อเพิ่มหรือลดประจุของความจุไฟฟ้าชดเชยซึ่งมีค่าประมาณ 30 พิโกฟาราด เท่ากับ 15 ไมโครแอมป์ดังนั้น

$$\begin{aligned}\text{อัตราสลู} &= (\text{ระดับเปลี่ยนของศักดาสัญญาณออก}) / \text{เวลา} \\ &= I/C = 15\ \mu A / 30pF \\ &= 0.5\ V/\mu sec\end{aligned}$$

จากสมการจะเห็นได้ว่า ถ้าต้องการอัตราสลูที่เร็วขึ้น ออปแอมป์จะต้องมีความสามารถจ่ายกระแสสูงสุดได้เพิ่มขึ้น หรือความจุไฟฟ้าชดเชย จะต้องลดลง เช่นในออปแอมป์ AD 518 ซึ่งมีอัตราสลูเท่ากับ 80 โวลท์/ไมโครวินาที จะต้องมี I = 400 ไมโครแอมป์และ C = 50 พิโกฟาราด

ตัวอย่างที่ 4 ถ้าศักดาสัญญาณเข้าเปลี่ยนแปลงอย่างทันทีทันใด เท่ากับ 10 โวลท์ สำหรับวงจรขยายศักดาสัญญาณแบบกลับศักดา ซึ่งมีค่าขยายเท่ากับหนึ่ง ถ้าออปแอมป์ที่ใช้เป็นแบบ 741 จงคำนวณหาเวลาที่ศักดาสัญญาณออกต้องการในการเพิ่มระดับขึ้น 10 โวลท์

วิธีทำ

$$\text{อัตราสลู} = (\text{ระดับเปลี่ยนของศักดาสัญญาณออก}) / \text{เวลา}$$

$$0.5\text{V}/1\mu\text{sec} = 10\text{ V} / t$$

ดังนั้นจะได้

$$t = (10\text{ V} \times 1\ \mu\text{sec}) / 0.5\text{ V}$$

$$= 20\ \mu\text{sec}$$

การจำกัดอัตราสลูสำหรับสัญญาณรูปซายน์

ในวงจรตามสัญญาณของภาพที่ 2.16 มี Ei เป็นสัญญาณเข้ารูปซายน์ ซึ่งมีระดับศักดาสูงสุดเท่ากับ Ep อัตราการเปลี่ยนระดับสูงสุด Ei ขึ้นอยู่กับความถี่ f และระดับศักดาสูงสุด Ep เท่ากับ $2\pi fEp$ ถ้าอัตราดังกล่าวมีค่าสูงสุดกว่าอัตราสลูของออปแอมป์ ผลจะปรากฏว่าศักดาของสัญญาณออกจะเปลี่ยนรูปไปจากสัญญาณเข้า คือเกิดดิสทรอชั่นขึ้น ทั้งนี้เพราะศักดา Vo จะพยายามตามศักดา Ei แต่ Vo จะเปลี่ยนระดับได้ค่าสูงสุดเท่ากับอัตราสลูเท่านั้น ดังนั้นถ้า $2fEp$ มีค่าสูงเท่าอัตราสลูมากขึ้นเท่าใด ผลจะปรากฏว่า Vo จะมีลักษณะใกล้เคียงกับสัญญาณรูปสามเหลี่ยมมากขึ้นเท่านั้น ความถี่สูงสุด (f_{max}) ซึ่งเราจะได้สัญญาณออกซึ่งมีระดับสูงสุดเท่ากับ Vop และไม่มีดิสทรอชั่นจะหาได้จาก

$$f_{max} = (\text{อัตราสลู}) / 6.28 \times \text{Vop}$$

โดยที่ f_{max} มีหน่วยเป็นเฮิร์ต, Vop มีหน่วยเป็นโวลท์ และอัตราสลูมีหน่วยเป็นโวลท์ต่อไมโครวินาที

ตัวอย่างที่ 5 ถ้าอัตราสลูของออปแอมป์แบบ 741 เท่ากับ 0.5 โวลท์/ไมโครวินาที จงหาความถี่สูงสุดซึ่งจะได้สัญญาณออกแบบ ไม่มีดิสทรอชั่นสำหรับระดับศักดาสูงสุดดังนี้ (a) 10 โวลท์ (b) 1 โวลท์

วิธีทำ (a) จากสมการ

$$f_{max} = (1 \times 0.5\text{V}/\mu\text{sec}) / (6.28 \times 13\text{V})$$

$$= 8\ kHz$$

(b) จากสมการ

$$f_{max} = 80\ kHz$$

ตัวอย่างที่ 6 ระดับอัตราสัญญาณออกสูงสุดของออปแอมป์แบบ 741 จะเท่ากับ 13 โวลท์ ถ้าใช้ไฟจ่ายซึ่งมีระดับศักดา ± 15 โวลท์ ระดับศักดาสัญญาณออกดังกล่าวนี้จะเรียกว่าเป็นระดับสัญญาณออก

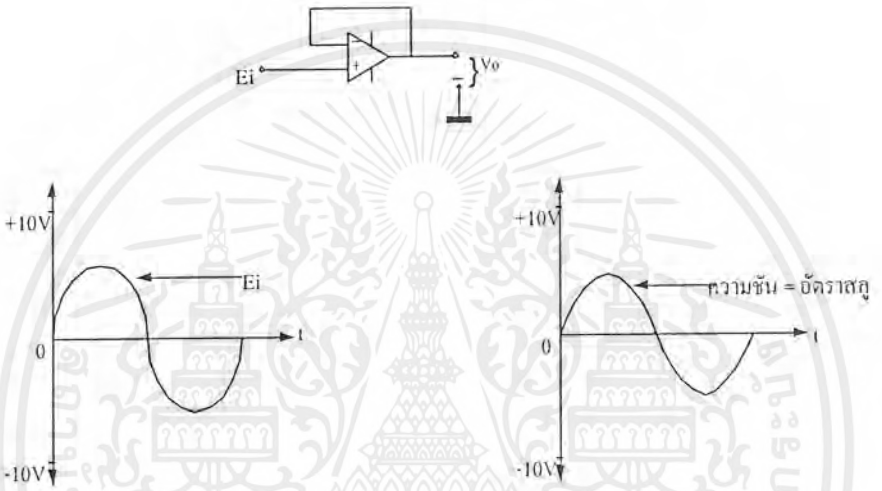
กำลังสูงสุด จงคำนวณหาความถี่สูงสุดซึ่งออปแอมป์จะจ่ายสัญญาณออกกำลังสูงสุดได้ โดยไม่มีดิสทรอชั่น

วิธีทำ

$$f_{max} = (1 \times 0.5V/\mu sec) / (6.28 \times 13V)$$

$$= 6\ kHz$$

ตัวอย่างที่ 5 และ ตัวอย่างที่ 6 แสดงว่าอัตราสลูจะเป็นจำนวนจำกัดขีดความถี่สูงสุดสำหรับสัญญาณระดับสูง เมื่อระดับสัญญาณลดลง ขีดความถี่สูงสุด ซึ่งถูกจำกัดโดยอัตราสลูจะเพิ่มขึ้น



รูปที่ 2.18 ตัวอย่างผลของอัตราสลูซึ่งมีต่อสัญญาณออก

จากที่ได้กล่าวมาแล้วในหัวข้อก่อนว่าขีดความถี่สูงสุดสำหรับสัญญาณระดับต่ำจะเพิ่มขึ้น ถ้าค่าขยายศักดาแบบลูปปิดลดลง ดังนั้นในการปฏิบัติงานของออปแอมป์ในแต่ละกรณี ขีดความถี่สูงสุดซึ่งถูกจำกัดโดยอัตราสลูและแบนด์วิดสัญญาณระดับต่ำ จะต้องคำนวณเปรียบเทียบกัน ขีดความถี่อันต่ำที่คำนวณได้จะเป็นขีดความถี่ซึ่งจะจำกัดความถี่สูงสุดของการใช้งาน โดยทั่วไปอัตราสลูจะเป็นจำนวนจำกัดความถี่สูงสุด เมื่อสัญญาณมีระดับสูง และการตอบสนองต่อสัญญาณระดับต่ำจะเป็นจำนวนจำกัดความถี่สูงสุดเมื่อสัญญาณมีระดับต่ำ

2.5 ตำแหน่งขาต่าง ๆ บนสล็อท ISA

ภายใน IBM PC ได้มีการออกแบบให้สามารถเพิ่มเติมวงจรอินเตอร์เฟสเข้าไปภายหลังได้โดยผ่านสล็อทที่มีอยู่บนเมนบอร์ด สำหรับสล็อทมาตรฐานแรกบนเครื่องคอมพิวเตอร์ส่วนบุคคลคือสล็อทแบบ ISA ซึ่งแต่ละสล็อทมีจำนวนขาทั้งสิ้น 64 ขา แบ่งเป็น 2 ข้าง ๆ ละ 32 ขา ตำแหน่งขาของที่อยู่ด้านซ้ายของสล็อทจะเลือกโดยใช้อักษร B นำหลขหน้าของตำแหน่งของขา เช่นขา B10 คือขาทางด้านซ้ายของ

สล็อทขาที่ 10 (นับจากทางด้านซ้ายของเครื่อง) ส่วนตำแหน่งของขาที่อยู่ทางด้านขวาของสล็อทจะเลือกโดยใช้อักษร A นำหน้าเลขตำแหน่งของขา เช่น A24 คือขาทางด้านขวาของสล็อทขาที่ 24 (นับจากทางด้านท้ายของเครื่อง)

แต่ขาของสล็อทเหล่านี้เชื่อมต่อกับเส้นสัญญาณต่าง ๆ บนเมนบอร์ด ทำให้การสร้างวงจรอินเตอร์เฟสกับเครื่องคอมพิวเตอร์ส่วนบุคคลสามารถกระทำได้สะดวก เส้นสัญญาณที่เชื่อมต่อกับขาสล็อทเหล่านี้ประกอบด้วย เส้นสัญญาณของแอดเดรส (Address Bus) , บัสข้อมูล (Data Bus) , บัสควบคุมสำหรับการเขียนและการอ่านข้อมูลจากหน่วยความจำหรือพอร์ท I/O , เส้นสัญญาณสำหรับการขออินเตอร์รัพของวงจรอินเตอร์เฟส , เส้นสัญญาณสำหรับขอ DMA (Direct Memory Access) , สัญญาณฐานเวลา (Timing Signal) ต่าง ๆ ที่ใช้ในระบบเส้นสัญญาณแสดงการรีเฟชหน่วยความจำและสัญญาณสำหรับตรวจสอบความผิดพลาด (I/O Check) นอกจากเส้นสัญญาณเหล่านี้แล้วสล็อท ISA ยังสามารถเชื่อมต่อกับแหล่งจ่ายไฟต่าง ๆ คือ $\pm$5 V(DC) , $\pm$12 V(DC) ตำแหน่งขาต่าง ๆ บนสล็อท ISA แสดงไว้ในรูปที่ 2.19

ชื่อสัญญาณ	ตำแหน่งขาบนสล็อท		ชื่อสัญญาณ
	ด้านลายทองแดง	ด้านติดตั้งอุปกรณ์	
GND	B01	A01	I/OCHCK
RESET	B02	A02	D7
+5V	B03	A03	D6
IRQ9	B04	A04	D5
-5V	B05	A05	D4
DREQ2	B06	A06	D3
-12V	B07	A07	D2
OWS	B08	A08	D1
+12V	B09	A09	D0
GND	B10	A10	I/O CHRD
MEMW	B11	A11	AEN
MEMR	B12	A12	A19
IOWC	B13	A13	A18
IORC	B14	A14	A17
DACK3	B15	A15	A16
DRQ3	B16	A16	A15

ชื่อสัญญาณ	ตำแหน่งขาบนสล็อท		ชื่อสัญญาณ
	ด้านลายทองแดง	ด้านติดตั้งอุปกรณ์	
DACK1	B17	A17	A14
DREQ1	B18	A18	A13
DACK0	B19	A19	A12
CLK	B20	A20	A11
IRQ7	B21	A21	A10
IRQ6	B22	A22	A09
IRQ5	B23	A23	A08
IRQ4	B24	A24	A07
IRQ3	B25	A25	A06
DACK2	B26	A26	A05
TC	B27	A27	A04
ALE	B28	A28	A03
+5V	B29	A29	A02
OSC	B30	A31	A01
GND	B31	A32	A00

รูปที่ 2.19 แสดงตำแหน่งขาบนสล็อทของ ISA

2.5.1 รายละเอียดเกี่ยวกับสัญญาณต่าง ๆ บนสล็อท ISA

สัญญาณต่าง ๆ บนสล็อท ISA มีหลายสัญญาณ แต่ในที่นี้จะกล่าวถึงเฉพาะสัญญาณต่าง ๆ ที่จำเป็นที่ใช้เกี่ยวกับการอ่าน/เขียนข้อมูลกับหน่วยความจำและพอร์ท I/O เท่านั้น

$A_0 - A_{19}$ (Address Bus : ขา A31 – A12)

ขาสัญญาณทั้ง 20 สัญญาณนี้เป็นเอาท์พุท ซึ่งใช้สำหรับกำหนดแอดเดรสของหน่วยความจำหรืออุปกรณ์ I/O ที่ CPU ต้องการติดต่อ โดยสัญญาณที่ A_0 จะมีค่านัยสำคัญต่ำที่สุด (LSB) และ A_{19} จะมีนัยสำคัญสูงที่สุด (MSB) สำหรับค่าแอดเดรส $A_0 - A_{19}$ นี้ในระหว่างกระบวนการอ่าน/เขียนข้อมูลกับหน่วยความจำหรืออุปกรณ์ I/O ถูกกำหนดโดย CPU แต่ในระหว่างกระบวนการ DMA จะถูกกำหนดโดย DMA-Controller (ระหว่างนี้ CPU จะถูกตัดออกจากระบบ) จะเห็นได้ว่าแอดเดรส 20 เส้นสามารถอ้างอิงแอดเดรสของหน่วยความจำได้ถึง 1 *M* byte คือตั้งแต่ 0FC00-0FFFFF สำหรับการอ้างแอดเดรสพอร์ทI/O จะเลือกใช้เพียง 16 เส้น คือ A_0 - A_{15} ซึ่งจะทำให้อ้างแอดเดรสของพอร์ทเพียง 10 เส้นคือ $A_0 - A_9$ และค่าแอดเดรสที่ใช้งานจะต้องอยู่ในช่วง 0200 – 03FF

$D_0 - D_7$ (Data Bus : ขา A9 – A2)

ขาสัญญาณนี้เป็นแบบ Bi-Directional ซึ่งต่ออยู่กับบัสข้อมูลของระบบเพื่อทำหน้าที่ในการส่งผ่านข้อมูลระหว่างพอร์ท I/O กับเครื่องคอมพิวเตอร์ส่วนบุคคล โดยบิท D_0 จะมีค่านัยสำคัญต่ำที่สุด (LSB) และบิท D_7 จะมีค่านัยสำคัญสูงที่สุด (MSB)

IOW(I/O Write : ขา B13)

ขาสัญญาณนี้เป็นเอาท์พุทซึ่งแอคทิฟที่ลอจิก 0 ซึ่งถูกสร้างขึ้นโดย 8080 Bus Controller เพื่อแสดงว่าไซเคิลที่เกิดขึ้นเป็นไซเคิลของการเขียนข้อมูลลงบนพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนแอดเดรสบัสนั้นรับข้อมูลไปเก็บไว้ และขา LOW นี้จะแอคทิฟในอีกกรณีหนึ่งคือ กรณี DMA Controller จะทำการส่งสัญญาณ LOW เอง โดยที่ค่าแอดเดรสที่อยู่บนบัสแอดเดรสจะเป็นค่าของหน่วยความจำที่พอร์ท I/O ที่ DMA ต้องการจะอ่านข้อมูล

MEMW (Memory Write : ขา B11)

ขานี้เป็นขาเอาท์พุทซึ่งจะแอคทิฟที่ลอจิก 0 ซึ่ง 8080 Bus Controller สร้างขึ้นในระหว่างไซเคิลในการเขียนข้อมูลลงในหน่วยความจำที่แอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้นทำการรับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ โดยทั่วไปหน่วยความจำจะรับข้อมูลในช่วงขอบขาขึ้นของสัญญาณ MEMW สำหรับในกระบวนการ DMA นั้น MEMW ก็จะแอคทิฟเช่นเดียวกัน

MEMR (Memory Read : ขาB12)

ขานี้เป็นขาเอาท์พุทจะแอคทิฟที่ลอจิ 0 ในระหว่างไซเคิลของการอ่านข้อมูลจากหน่วยความจำของ CPU เพื่อให้อ่านหน่วยความจำที่มีแอดเดรสตรงกับแอดเดรสบนแอดเดรสนั้นทำการส่งข้อมูลออกมายังบัสข้อมูลสำหรับในระหว่างกระบวนการ DMA นั้น MEMR ก็จะแอคทิฟเช่นกัน

AEN (Address Enable : ขาA11)

สัญญาณนี้เป็นเอาท์พุทที่ใช้ในการแสดงว่าไซเคิลที่เกิดขึ้นในช่วงเวลาที่สัญญาณ AEN แอคทิฟที่โลจิก 1 นั้นเป็นไซเคิลของกระบวนการ DMA

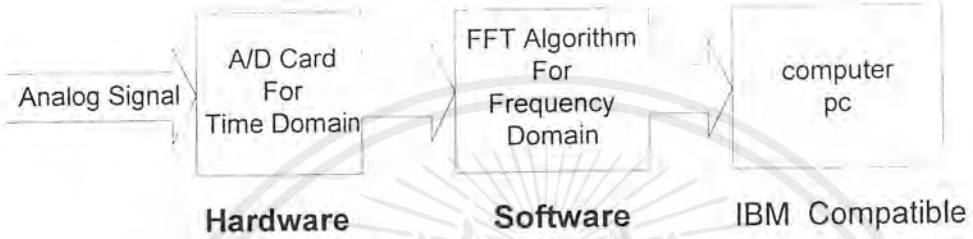
AGD (Ground Address : ขาB1 , B10 และB31)

ขาทั้งสามนี้จะต่อเข้ากับกราวด์ของระบบ

บทที่ 3
หลักการทำงานและการคำนวณ

3.1 หลักการทำงานพื้นฐาน

สามารถสรุปหลักการทำงานเป็นบล๊อคไดอะแกรม(Block Diagram)ได้คร่าวๆดังรูป 3.1



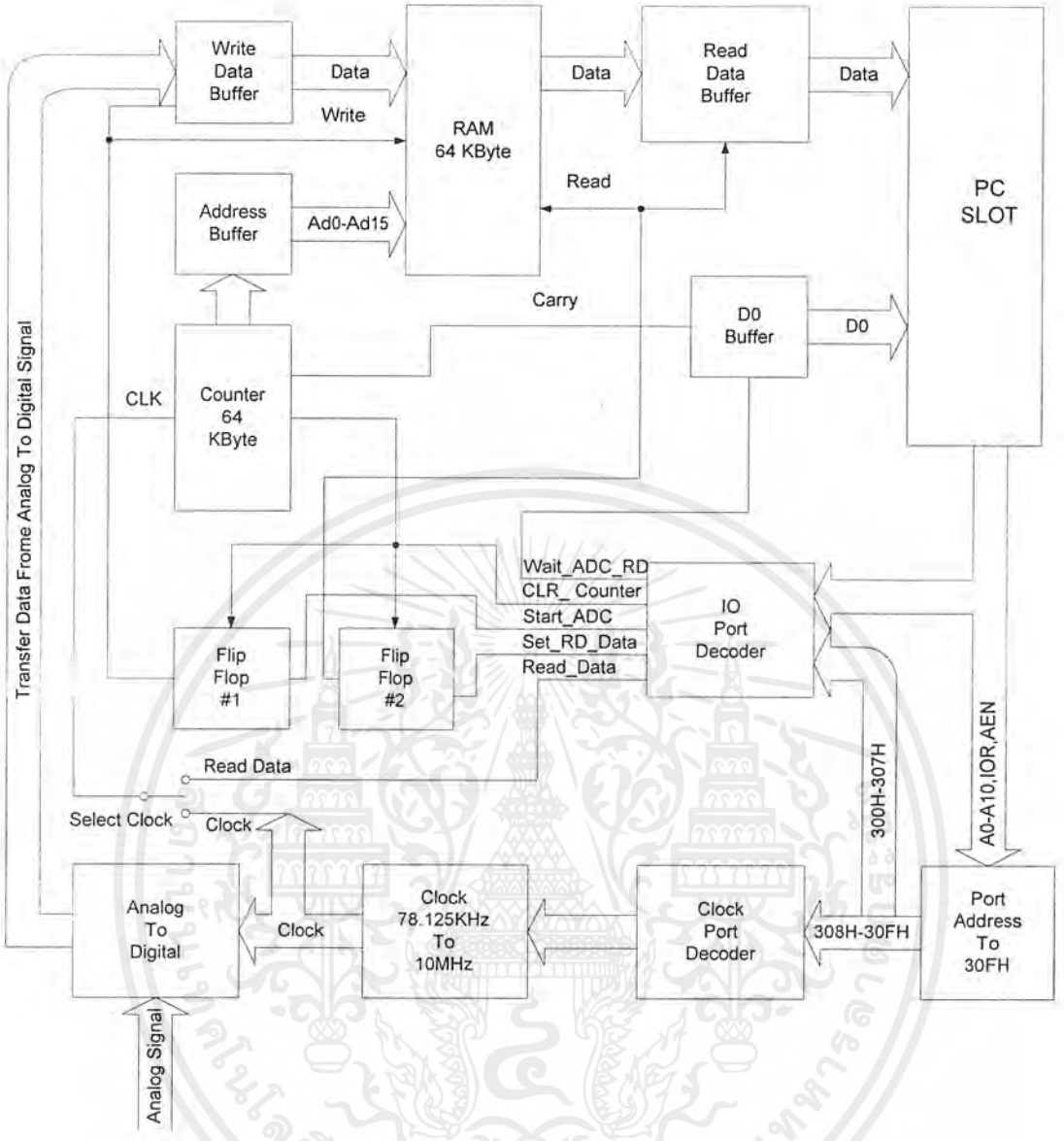
รูปที่ 3.1 บล๊อคไดอะแกรม

จากบล๊อคไดอะแกรมสามารถแบ่งได้เป็น 2 ส่วนใหญ่ๆคือ ฮาร์ดแวร์(Hardware) และ ซอฟแวร์(Software)โดยฮาร์ดแวร์คือการ์ดที่จะต้องทำขึ้นมาเพื่อที่จะแปลงค่าสัญญาณจากอนาล็อกไปเป็นสัญญาณแบบดิจิตอลและติดต่อสื่อสารกับคอมพิวเตอร์ได้ โดยค่าที่แปลงได้จากอนาล็อกเป็นดิจิตอลจะได้ข้อมูลเป็นค่าแอมปลิจูดในเชิงเวลา หลังจากนั้นคอมพิวเตอร์จะนำค่าแอมปลิจูดที่ได้ไปเปลี่ยนให้อยู่ในรูปแบบของจำนวนเชิงซ้อนโดยจะให้ค่าแอมปลิจูดเป็นค่าจริงและให้ค่าจินตภาพมีค่าเท่ากับศูนย์ จากนั้นจะแปลงค่าจำนวนเชิงซ้อนที่ได้ในเชิงเวลาให้อยู่ในเชิงความถี่โดยใช้วิธีการแบบบัตเตอร์ฟลายเป็นตัวคำนวณและนำค่าที่ได้ไปแสดงผลที่จอของคอมพิวเตอร์และคำนวณหาความถี่ต่อไป

โดยในบทนี้จะอธิบายส่วนของฮาร์ดแวร์และซอฟแวร์ โดยซอฟแวร์จะมีการคำนวณหาค่าของฟาสต์ฟูเรียร์ทรานส์ฟอร์มและการแสดงผลที่คำนวณได้ให้อยู่ในรูปความถี่

3.2 หลักการทำงานของวงจร

จากบล๊อคไดอะแกรมของฮาร์ดแวร์ตามรูปที่ 3.1 มาทำการแยกเป็นบล๊อคเพื่อจะได้รู้การทำงานของส่วนต่างๆและเป็นแนวทางในการออกแบบฮาร์ดแวร์ต่อไปซึ่งพอที่จะอธิบายได้เป็นบล๊อคไดอะแกรมดังรูปที่ 3.2



รูปที่ 3.2 บล๊อคไดอะแกรมหลักการทำงานของวงจร

รูปที่ 3.2 บล๊อคไดอะแกรมหลักการทำงานของวงจร พอที่จะอธิบายหน้าที่การทำงานของแต่ละบล็อกได้คร่าวๆดังนี้โดยเริ่มจาก

สล๊อตคอมพิวเตอร์ (PC Slot)

โดยทั่วไปสล็อตของคอมพิวเตอร์จะทำหน้าที่ติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์กับการ์ดที่ทำขึ้นมาส่วนใหญ่คอมพิวเตอร์จะมีพอร์ท(Port)ว่างอยู่ช่วงหนึ่งจะใช้พอร์ทที่ว่างอยู่สำหรับการควบคุมการทำงานของส่วนต่างๆของการ์ดได้ เช่นใช้พอร์ทหมายเลข 0300H-030FH และใช้แอดแดรส A0-A10 ของคอมพิวเตอร์เป็นตำแหน่งในการควบคุมพอร์ทโดยใช้ร่วมกับขา IOR และ AEN ของคอมพิวเตอร์

การอ้างตำแหน่งพอร์ท(Port Address)

พอร์ทแอดเดรสตำแหน่ง 0300H-030FH จะถูกแบ่งเป็น 2 กลุ่มใหญ่ๆคือ พอร์ทแอดเดรสตำแหน่ง 0300H-0307H และพอร์ทแอดเดรสตำแหน่ง 0308H-030FH โดย พอร์ทแอดเดรสตำแหน่ง 0300H-0307H ใช้ในการอ่านข้อมูลและเขียนข้อมูลเพื่อควบคุมการทำงานต่างๆ และพอร์ทแอดเดรสตำแหน่ง 0308H-030FH จะใช้ในการเลือกสัญญาณนาฬิกาโดยสามารถเลือกสัญญาณนาฬิกาได้ทั้งหมด 8 พอร์ทแอดเดรสด้วยกัน

พอร์ทเลือกความถี่ของสัญญาณนาฬิกา (Clock Port Decoder)

ทำหน้าที่เลือกความถี่ของสัญญาณนาฬิกาเพื่อป้อนให้กับเอทูดีและตัวนับโดยเลือกได้ทั้งหมด 8 ความถี่โดยจะใช้พอร์ทแอดเดรสตำแหน่ง 0308H-030FH

อินพุท/เอาท์พุท พอร์ท (I/O Port Decoder)

เป็นพอร์ทที่ใช้ควบคุมการติดต่อระหว่างฮาร์ดแวร์กับคอมพิวเตอร์ โดยอินพุท/เอาท์พุทพอร์ทจะสร้างเอาท์พุทมา 8 เส้น แต่จะใช้เพียง 5 เส้นเท่านั้นโดยมีสัญญาณจากขาต่างๆ ดังนี้

1.เคลียร์ตัวนับ(CLR_Counter)จะเป็นตัวเคลียร์ส่วนของตัวนับ(Counter) เพื่อให้เริ่มนับที่ตำแหน่งแอดเดรส 0000H-FFFFH และเคลียร์ฟลิบ-ฟลอบ(Flip-Flop) ทั้ง 2 ตัวให้มีสถานะเป็นศูนย์

2.เลือกการเขียนข้อมูล(Start_Counter) หน้าที่เปิดบัฟเฟอร์การเขียนข้อมูลที่บล๊อคของตัวป้องกันการเขียนข้อมูล(Wait Data Buffer)และทำหน้าที่เขียนข้อมูลที่แปลงได้จากเอทูดี(ADC) ไปเก็บไว้ในหน่วยความจำให้เต็ม 64 กิโลไบท์(*k* bytes)

3.รอการเขียนข้อมูล(Wait_ADC_RD) ทำหน้าที่รอการเขียนข้อมูลจากเอทูดี ว่ามีข้อมูลเต็มหน่วยความจำหรือยังโดยใช้ร่วมกับตัวนับ โดยตัวนับทำการนับถึง 64 กิโลไบท์แล้วตัวนับจะส่งแคร์รี่ออกมาเป็นสัญญาณนาฬิกา 1 ลูก เพื่อบอกให้คอมพิวเตอร์รู้ว่ามีการเขียนข้อมูลในหน่วยความจำเต็ม 64 กิโลไบท์ โดยจะส่งสัญญาณเส้นนี้มาตลอดเวลาเพื่อคอยตรวจสอบว่ามีการเขียนข้อมูลลงในหน่วยความจำเต็มหรือยัง เมื่อข้อมูลเต็ม 64 กิโลไบท์แล้วคอมพิวเตอร์จะหยุดส่งสัญญาณเส้นนี้ทันที

4.เลือกการอ่านข้อมูล(Set_RD_Data)เป็นตัวเลือกการอ่านข้อมูลโดยอ่านข้อมูลจากหน่วยความจำเพื่อนำไปเก็บไว้ในคอมพิวเตอร์และเป็นตัวเปิดเกทให้กับตัวป้องกันการอ่านข้อมูล(Read Data Buffer) โดยจะส่งสัญญาณนาฬิกามาแค่ลูกดียว

5.การอ่านข้อมูล (Read_Data) ทำหน้าที่เป็นสัญญาณนาฬิกาให้กับตัวนับเพื่ออ้างตำแหน่งแอดเดรสในการอ่านข้อมูลจากหน่วยความจำไปเก็บไว้ในคอมพิวเตอร์

ตัวผลิตสัญญาณนาฬิกา(Clock)

เป็นตัวกำเนิดสัญญาณนาฬิกาโดยจะได้ความถี่ของสัญญาณนาฬิกาจากการออสซิลเลเตอร์ (Oscillator) ซึ่งผลิตความถี่ 20 เม็กกะเฮิร์ตโดยผ่านวงจรนับ 8 บิท จะได้สัญญาณนาฬิกา 8 ความถี่คือ ความถี่สูงสุดของสัญญาณ 10 เม็กกะเฮิร์ต และค่อยๆหาร 2 ไปเรื่อยๆจนได้ความถี่ต่ำสุดของสัญญาณ 78.125 กิโลเฮิร์ต สาเหตุที่ต้องมีการเลือกสัญญาณนาฬิกาหลายความถี่ต่ำๆไม่สามารถแปลงข้อมูลจากอนาล็อกเป็นดิจิตอลได้ครบ 1 คาบ เพราะมีหน่วยความจำในการเก็บข้อมูลจำกัด(ประมาณ 64 กิโลไบท์) โดยข้อมูลที่ได้อาจจะยังไม่ครบ 1 คาบ หน่วยความจำอาจจะเต็มแล้ว จึงต้องมีการเลือกสัญญาณนาฬิกา

เพื่อใช้ในการสุ่มค่าอินพุทและสัญญาณนาฬิกาที่ได้ก็จะนำไปเป็นสัญญาณนาฬิกาให้กับตัวนับเพื่อใช้ในการอ้างตำแหน่งแอดเดรสด้วย

เอทูดี(ADC)

เอทูดีจะทำหน้าที่แปลงจากสัญญาณอนาล็อกไปเป็นสัญญาณดิจิตอลโดยส่งข้อมูลที่ได้ไปเก็บไว้ที่หน่วยความจำโดยมีการแปลงค่าตลอดเวลา

ตัวป้องกันการเขียนข้อมูล(Write Data Buffer)

ตัวป้องกันการเขียนข้อมูลทำหน้าที่เป็นตัวกันชนระหว่าง เอทูดี กับหน่วยความจำโดยมี ฟลิบ-ฟลอบ 1 เป็นตัวเปิดเกทให้กับตัวป้องกันการเขียนข้อมูลเพื่อเป็นการถ่ายเทข้อมูลจากเอทูดีไปสู่หน่วยความจำ

ตัวเลือกสัญญาณนาฬิกา(Select Clock)

ตัวเลือกสัญญาณนาฬิกาจะเป็นตัวเลือกสัญญาณนาฬิการะหว่างสัญญาณนาฬิกาของตัวผลิตสัญญาณนาฬิกากับสัญญาณนาฬิกาจากการอ่านข้อมูลของอินพุท/เอาท์พุท พอร์ท

ฟลิบ-ฟลอบ 1 (Flip-Flop 1)

ฟลิบ-ฟลอบ 1 ทำหน้าที่เป็นตัวจำสภาวะให้กับสัญญาณการเขียนข้อมูลซึ่งมาจาก อินพุท/เอาท์พุท พอร์ทและทำหน้าที่เปิดเกทของตัวป้องกันการเขียนข้อมูลเพื่อให้ข้อมูลจากเอทูดีผ่านไปหน่วยความจำและเป็นสัญญาณการเขียนข้อมูลให้กับหน่วยความจำ

ฟลิบ-ฟลอบ 2 (Flip-Flop 2)

ฟลิบ-ฟลอบ 2 ทำหน้าที่เป็นตัวจำสภาวะให้กับสัญญาณการอ่านข้อมูลซึ่งมาจาก อินพุท/เอาท์พุท พอร์ท และทำหน้าที่เปิดเกทของตัวป้องกันการอ่านข้อมูลเพื่อให้ข้อมูลจากหน่วยความจำไปเก็บไว้ในคอมพิวเตอร์โดยผ่านทางสล็อตของคอมพิวเตอร์

ตัวนับ(Counter)

ตัวนับจะทำหน้าที่เป็นตัวนับเพื่ออ้างตำแหน่งแอดเดรสของหน่วยความจำทั้งการอ่านข้อมูลและการเขียนข้อมูล

เมื่อรู้หน้าที่ต่างๆของแต่ละบล็อกแล้วก็สามารถอธิบายบล็อกไดอะแกรมทั้งหมดให้สัมพันธ์กันเพื่อจะนำมาสร้างเป็นวงจรต่อไป เริ่มต้นด้วยต้องการหาค่าสเปคตรัมของสัญญาณหนึ่งโดยต้องการเลือกสัญญาณนาฬิกาเพื่อใช้ในการสุ่มข้อมูลให้เหมาะสมกับสัญญาณอินพุทโดยจะใช้คอมพิวเตอร์ในการเลือกความถี่ของสัญญาณนาฬิกาจะถอดรหัสได้ทั้งหมด 8 พอร์ท โดยจะใช้ขาต่างๆของคอมพิวเตอร์เป็นตัวถอดรหัสโดยนำขาที่มาถอดรหัสดังนี้คือขาแอดเดรส A0-A10 สัญญาณ IOR และAEN และจะได้สัญญาณนาฬิกาในความถี่ต่างๆ ตามตาราง 3.1

หมายเลขพอร์ท	ความถี่
0308H	10*MHz*
0309H	5*MHz*
030AH	2.5*MHz*
030BH	1.25*MHz*
030CH	625*kHz*
030DH	312.5*kHz*
030EH	156.25*kHz*
030FH	78.125*kHz*

ตาราง 3.1 แสดงหมายเลขพอร์ทที่ใช้เลือกความถี่สัญญาณนาฬิกา

จะเลือกความถี่ของสัญญาณนาฬิกามา 1 ค่า เพื่อใช้เป็นสัญญาณนาฬิกาของตัวนับ และเป็นสัญญาณนาฬิกาให้กับการสุ่มข้อมูลของเอทูดี และคอมพิวเตอร์จะอ้างตำแหน่งพอร์ทที่ใช้การอ้างข้อมูล การเขียนและการอ่านข้อมูลได้อีก 8 พอร์ท แต่จะใช้เพียง 5 พอร์ท ซึ่งอยู่ในส่วนของ อินพุท/เอาท์พุท พอร์ท ดังนี้

หมายเลขพอร์ท	สัญญาณ
0300H	เคลียร์ตัวนับ
0302H	เขียนข้อมูล
0302H	รอการเขียนข้อมูล
0303H	เลือกการอ่านข้อมูล
0304H	อ่านข้อมูล

ตาราง 3.2 แสดงหมายเลขพอร์ทที่ใช้เลือกสัญญาณควบคุม

หลักการทำงานเริ่มจากเมื่อคอมพิวเตอร์ได้ความถี่ของสัญญาณนาฬิกา 1 ค่าแล้วคอมพิวเตอร์ก็จะส่งสัญญาณเคลียร์ตัวนับมาที่พอร์ทแอดเดรส "0300H" เพื่อทำการเคลียร์ตัวนับ เพื่อให้มีค่าเริ่มต้นที่

แอดเดรส "0000H" และให้ฟลิบ-ฟลอบทั้ง 2 ตัวมีค่าเป็น "1"จึงไม่มีการเปิดเกทในการอ่านข้อมูลของหน่วยความจำ

จากนั้นคอมพิวเตอร์ก็จะส่งสัญญาณการเขียนข้อมูลออกมาที่พอร์ทแอดเดรส 0301H เพื่อไปทริกฟลิบ-ฟลอบ 1ทำให้ฟลิบ-ฟลอบ 1 เปลี่ยนสถานะจาก "1" ไปเป็น "0" เพื่อไปเปิดเกทให้ตัวป้องกันการเขียนข้อมูล ทำให้มีข้อมูลจาก เอทูดี ซึ่งแปลงจากอนาล็อกไปเป็นดิตอลผ่านไปสู่หน่วยความจำขณะเดียวกันก็จะเปิดสัญญาณการเขียนข้อมูลของหน่วยความจำด้วยแต่ต้องรอตัวนับเริ่มนับเสียก่อน

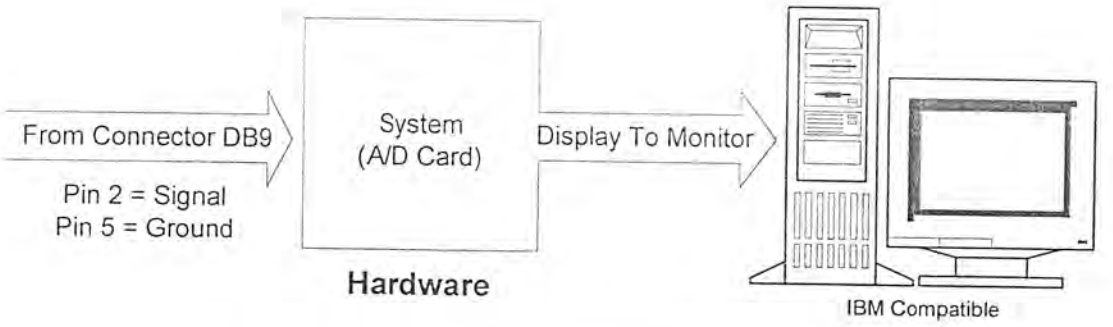
ในขณะเดียวกันตัวเลือกสัญญาณนาฬิกาจะถูกเลือกมาที่สัญญาณนาฬิกาทำให้มีสัญญาณนาฬิกาลูกแรกผ่านเข้ามาที่ตัวนับทำให้ตัวนับเริ่มนับที่ตำแหน่งแอดเดรส "0001H" เพื่ออ้างตำแหน่งแอดเดรสในการเขียนข้อมูลลงในหน่วยความจำของข้อมูลที่แปลงจากเอทูดีในตำแหน่งแรกก็ถูกเขียนลงในหน่วยความจำแล้วคอมพิวเตอร์ก็จะส่งสัญญาณรอการเขียนข้อมูลมาที่พอร์ทแอดเดรส "0032H" เพื่อเช็คดูว่าขา D0 ของคอมพิวเตอร์เป็น 1 หรือยัง ถ้ายังก็ให้ตัวนับทำการนับต่อไป

เมื่อสัญญาณนาฬิกาลูกที่ 2 เข้ามาตัวนับก็เริ่มเพิ่มค่าอีก 1 ค่าเพื่อการอ้างตำแหน่งแอดเดรส 0002H และข้อมูลที่แปลงจาก เอทูดี ในตำแหน่งที่ 2 ก็จะเขียนลงไป อีกค่าหนึ่งของสัญญาณนาฬิกาและคอมพิวเตอร์ก็จะส่งสัญญาณรอการเขียนข้อมูลมาที่พอร์ทแอดเดรส "0032H" เพื่อเช็คดูว่าขา D0 ของคอมพิวเตอร์เป็น 1 หรือยังถ้ายังก็ให้ตัวนับทำการนับต่อไป

จนกระทั่งสัญญาณนาฬิกาลูกที่ 64 กิโลไบท์เข้ามาทำให้ตัวนับ นับไปถึงตำแหน่งที่ 64 กิโลไบท์และมีการเขียนข้อมูลลงในหน่วยความจำเต็ม ตัวนับก็จะหยุดนับโดยอัตโนมัติ พร้อมจะส่งสัญญาณพัลซ์มาหนึ่งลูก ขณะเดียวกันคอมพิวเตอร์ก็จะส่งสัญญาณรอการเขียนข้อมูลมาที่พอร์ทแอดเดรส "0032H"เพื่อเช็คดูว่าขา D0 ของคอมพิวเตอร์เป็น 1 หรือยัง เนื่องจากตอนนี้มีพัลซ์ลูก เข้ามาที่ขา D0

ทำให้คอมพิวเตอร์รู้ว่าขณะนี้มีการเขียนข้อมูลลงในหน่วยความจำเต็ม 64 กิโลไบท์แล้วคอมพิวเตอร์ก็จะส่งสัญญาณเคลียร์ตัวนับออกมาที่พอร์ทแอดเดรส "0300H" เพื่อเคลียร์ฟลิบ-ฟลอบ ทั้ง 1 ตัวและเคลียร์ตัวนับด้วย

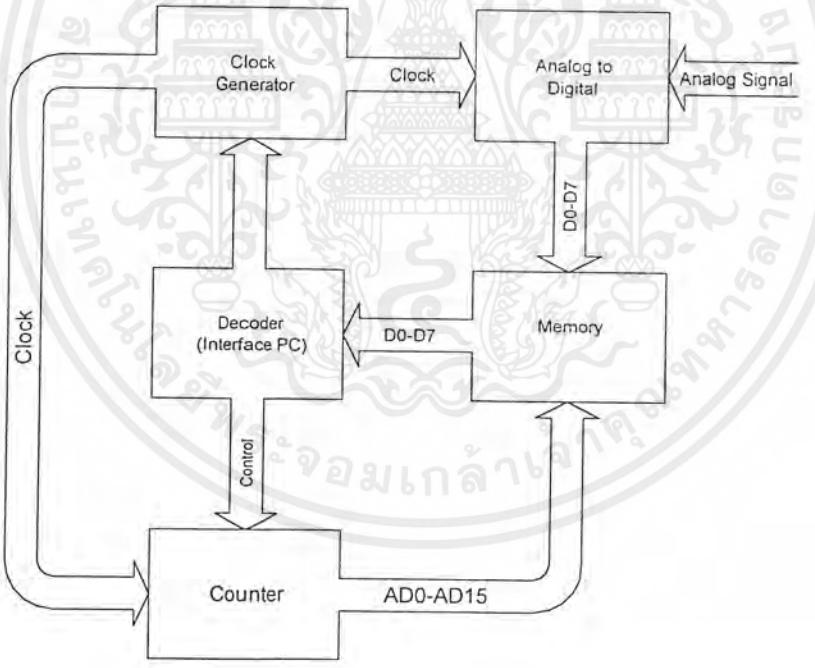
จากนั้นคอมพิวเตอร์ก็จะเริ่มส่งสัญญาณเลือกการอ่านข้อมูลมาที่พอร์ทแอดเดส "0303H"เพื่อเลือกข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์ เมื่อส่งสัญญาณการอ่านข้อมูลมาทำให้ ฟลิบ-ฟลอบ 2 เปิดเกทมีสัญญาณ "0" ไปที่ขา RD ของหน่วยความจำและเปิดเกทของตัวป้องกันการอ่านข้อมูลของหน่วยความจำ และคอมพิวเตอร์จะส่งสัญญาณการอ่านข้อมูลมาที่พอร์ทแอดเดรส "0304H"ขณะเดียวกันตัวเลือกสัญญาณนาฬิกาก็จะไปปิดสวิทที่ขาการอ่านข้อมูลเพื่อให้เป็นสัญญาณการอ่านข้อมูลเป็นสัญญาณนาฬิกาให้กับตัวนับเมื่อส่งสัญญาณการอ่านข้อมูลมา 1 ครั้งตัวนับก็จะนับ 1 ครั้งทำให้แอดเดรสเลื่อนไป 1 ตำแหน่งเพื่ออ่านข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์ 1 ค่าจากนั้นคอมพิวเตอร์ก็จะส่งสัญญาณการอ่านข้อมูลมาตลอดทำให้การอ่านข้อมูลมาเก็บไว้ที่หน่วยความจำของคอมพิวเตอร์ จากนั้นนำข้อมูลอ่านได้ไปทำการแปลงให้อยู่ในรูปความถี่ต่อไป และสามารถสรุปฮาร์ดแวร์ทั้งหมดดังรูปที่ 3.3 ซึ่งเป็นการ์ดที่ทำขึ้นมาเพื่อติดต่อกับคอมพิวเตอร์



รูปที่ 3.3 การ์ดแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

ระบบ (การ์ดอนาล็อกเป็นดิจิตอล)

สัญญาณอินพุทที่เข้ามาทางคอนเน็คเตอร์ จะถูกแปลงเป็นสัญญาณดิจิตอลด้วย ADC ขนาด 8 Bit การ์ดส่วนนี้แบ่งเป็น 5 ส่วนใหญ่ๆดังรูปที่ 3.4



รูปที่ 3.4 บล๊อคไดอะแกรมแสดง การ์ดแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

จากรูปที่ 3.4 มีทั้งหมด 5 บล๊อคคือ เอทูดี(Analog to Digital) , หน่วยความจำ(Memory) , วงจรผลิตสัญญาณนาฬิกา(Clock Generator) , วงจรถอดรหัส(Decoder) , วงจรนับ(Counter) จะอธิบายทีละบล๊อคได้ดังนี้

เอทูดี(Analog to Digital)

ในส่วนของบล๊อคนี้จะทำหน้าที่ในการแปลงสัญญาณอนาล็อกไปเป็นดิจิตอล โดยใช้สัญญาณนาฬิกาของวงจรผลิตสัญญาณนาฬิกาเป็นสัญญาณนาฬิกาในการสุ่มข้อมูลและข้อมูลที่ได้จะจะถูกส่งต่อไปให้บล๊อคหน่วยความจำ

หน่วยความจำ(Memory)

จะทำหน้าที่ในการเก็บข้อมูลที่แปลงได้จากสัญญาณอนาล็อกไปเป็นดิจิตอลโดยจะมีวงจรนับเป็นตัวกำหนดแอดเดรสในการเก็บข้อมูล

วงจรผลิตสัญญาณนาฬิกา(Clock Generator)

เป็นตัวกำเนิดสัญญาณนาฬิกาให้แก่วงจรนับและเอทูดี โดยสัญญาณนาฬิกาของวงจรผลิตสัญญาณนาฬิกานี้จะผลิตสัญญาณนาฬิกาได้ทั้งหมด 8 ความถี่ เพื่อใช้ในการเลือกสัญญาณในการสุ่มสัญญาณ

วงจรถอดรหัส(Decoder)

เป็นตัวถอดรหัสใช้ในการควบคุมวงจรนับ หน่วยวามจำและเลือกความถี่ของวงจรผลิตความถี่ของสัญญาณนาฬิกาและวงจรถอดรหัสนี้จะควบคุมการอ่านและเขียนข้อมูลจากหน่วยความจำเพื่อส่งข้อมูลที่ได้ไปให้คอมพิวเตอร์ต่อไป

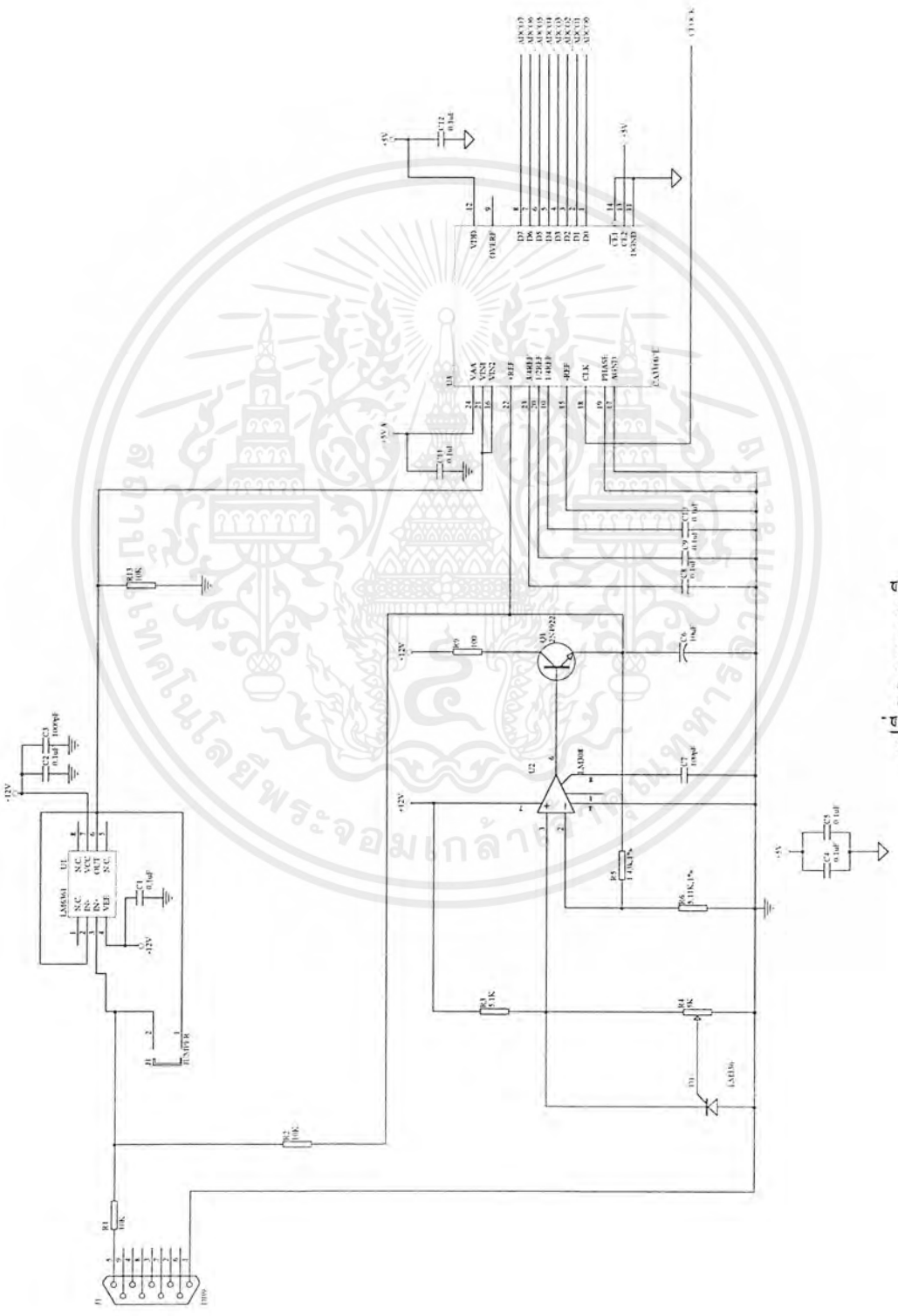
วงจรนับ(Counter)

ใช้สำหรับอ้างแอดเดรสของหน่วยความจำ โดยการนับแต่ล่ะครั้งจะถูกควบคุมโดยสัญญาณนาฬิกาของวงจรผลิตสัญญาณนาฬิกาและวงจรถอดรหัส

จากการ์ดอนาล็อกในรูปที่ 3.4 แบ่งได้ 5 ส่วนใหญ่ๆ คือวงจรเอทูดี , วงจรหน่วยความจำ , วงจรผลิตสัญญาณนาฬิกา , วงจรถอดรหัส และวงจรนับ

3.2.1 วงจรเอทูดี

เป็นการแปลงสัญญาณจากอนาล็อกเป็นดิจิตอล โดยใช้ไอซีเอทูดี 8 บิทเบอร์ CA 3318 เป็นตัวแปลงสัญญาณโดยมีการสุ่มสัญญาณที่เฉพาะไฟบวกไม่เกิน 6.4 โวลท์ เท่านั้นเมื่อเทียบกับกราวด์ มิฉะนั้นข้อมูลที่แปลงมาได้จะผิดพลาด เป็นวงจรดังรูปที่3.5แบ่งได้เป็น3ส่วนดังนี้คือวงจรอ้างอิงแรงดัน(Voltage Reference) , วงจรป้องกันสัญญาณ(Analog Buffer) , วงจรเอทูดี(ADC) อธิบายทีละส่วนดังนี้



รูปที่ 3.5 วงจรเอทูดี

วงจรอ้างอิงแรงดัน

จะทำหน้าที่ปรับระดับแรงดันเพื่อเป็นจุดอ้างอิงให้กับ U3(CA3318) โดยมี D1(LM336) ทำหน้าที่ปรับค่าโวลเตจ โดยมี R3,R4 เป็นวงจรโวลเตจดีไวเดอร์ ในที่นี้จะปรับค่า D1ให้ได้ประมาณ 5 โวลท์ เพื่อเป็นอินพุทให้กับ U2(LM308) ซึ่งเป็นออปแอมป์ ต่อวงจรในลักษณะนอนอินเวิสติ้ง โดยมี R5,R6 เป็นตัวแบ่งแรงดัน เพื่อให้ได้เอาท์พุทประมาณ 6.4 โวลท์และเอาท์พุทที่ได้จะไปเข้าขาเบสของ Q1 (2N4922)ซึ่ง Q1 จะทำหน้าที่เพิ่มกระแสให้เอาท์พุทโดยมี C6 เป็นตัวกรองไฟให้เรียบอีกที เอาท์พุทที่ได้จะไปต่อกับขา +REF ของ CA3318 และ R13($10\,k\,\Omega$)

จากการต่อแบบนอนอินเวิสติ้งแอมปลิฟลาย(Non Inverting Amplifier) สามารถคำนวณค่า R5,R6 เพื่อให้ได้เอาท์พุทประมาณ 6.4 โวลท์ ได้จากสมการ

$$Vout/Vin = 1 + R5/R6$$

กำหนดให้ Vin = 5 โวลท์

Vout = 6.4 โวลท์

R 5 = 1.43 $k\,\Omega$ 1%

แทนค่าต่างๆในสมการ จะได้

$$6.4 / 5 = 1 + 1.43\ k\,\Omega / R6$$

$$1.28 - 1 = 1.43\ k\,\Omega / R6$$

$$R6 = 1.43\ k\,\Omega / 0.28$$

$$R6 = 5.11\ k\,\Omega$$

สรุปได้ว่าถ้าต้องการเอาร์ทพุทโวลท์เตจประมาณ 6.4 โวลท์ จะต้องเลือก

R5 = 1.43 $k\,\Omega$ 1%

R6 = 5.11 $k\,\Omega$ 1%

ถ้ายังไม่ได้ 6.4 โวลท์ สามารถปรับค่า R4 ซึ่งเป็นความต้านทานแบบปรับค่าได้เพื่อปรับค่าเอาท์พุทให้ได้ดังต้องการเพื่อป้อนให้ขา 22 (+REF) ของU3(CA3318) และเป็นขาอินพุทให้กับ R13($10\,k\,\Omega$)

วงจรป้องกันเอทูดี

ทำหน้าที่เป็นตัวกันชนระหว่างอินพุทกับตัวเอทูดีและเนื่องจากเอทูดีต้องการอินพุทเป็นสัญญาณไฟบวกเท่านั้น แต่สัญญาณที่ทำการวัดจะต้องมีทั้งช่วงบวกและช่วงลบ จึงต้องมีการดัดแปลงสัญญาณอินพุท ก่อนที่จะส่งสัญญาณผ่านวงจรป้องกันสัญญาณเพื่อให้สัญญาณรับได้ทั้งช่วงบวกและช่วงลบจึงต้องยกระดับสัญญาณอินพุทที่ได้ไปอีกครึ่งหนึ่งโดยใช้ขา +REF ของ U3(CA3318) ต่อวงจรระหว่าง R1 และ R13 ทำให้เอาท์พุทที่ได้ก่อนที่จะมาถึงขา 3 ของ U1(LM6361) ถูกยกระดับลงไปครึ่งหนึ่งจากสัญญาณอินพุทแต่จะยกระดับสัญญาณดีซีโวลท์เตจไปประมาณ 3.2 โวลท์โดย C1 ,C2 ,C3 และ C4 เป็นตัวกรองแรงดันไฟ +12 โวลท์ และ –12 โวลท์ ให้เรียบและขา 3 (IN) ของ U1(LM6361) จะเป็นสัญญาณอินพุทของสัญญาณอนาล็อกและจะได้เอาท์พุทออกที่ขา 6 (OUT) ของ U1 (LM6361) โดยสัญญาณที่ได้จะ

มีการผิดเพี้ยนน้อยที่สุด ค่าเอาท์พุทที่ได้จะป้อนให้วงจรเอทูดีโดยต่อเข้าที่ขา Vin1 , Vin2 ตามวงจร เนื่องจากเอทูดีที่ใช้ไม่มีแรงดันโวลท์เตจที่ใช้อ้างอิง จึงต้องสร้างแรงดันโวลท์เตจอ้างอิงขึ้นมาเพื่อเป็นจุดอ้างอิงสำหรับการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลและแรงดันโวลท์เตจที่ใช้อ้างอิงมีค่าประมาณ 6.4 โวลท์

วงจรเอทูดี

ทำหน้าที่แปลงสัญญาณอนาล็อกเป็นดิจิตอลโดยขา Vin1 , Vin2 เป็นอินพุท ขา 22(+REF)เป็นแรงดันโวลท์เตจอ้างอิงให้กับ ADC(CA3318) และขา REF อื่นๆไม่ใช้ก็ต่ออนุกรมกับตัวเก็บประจุลงกราวด์เพื่อไม่ให้มีสัญญาณไปรบกวน ADC(CA3318) รวมไปถึงขา -REF ,Phase และ AGND ก็ต่อกราวด์ด้วยเหมือนกัน

ขา Phase ต่อลงกราวด์ แสดงว่ามีการสุ่มสัญญาณนาฬิกาแบบไม่สมดุลย์ ถ้าต่อขา Phase กับไฟ +5 โวลท์ จะเป็นการสุ่มสัญญาณแบบสมดุล

ขา CLK ของเอทูดีต่อมาจากวงจรผลิตสัญญาณนาฬิกา

ขา CE1 และ CE2 จะถูกอีนาเบิล(Enable) ตลอดเวลาทำให้มีการแปลงข้อมูลตลอดเวลา

D0-D7 ไปต่อกับหน่วยความจำเพื่อนำค่าข้อมูลที่ได้ไปเขียนลงหน่วยความจำต่อไป

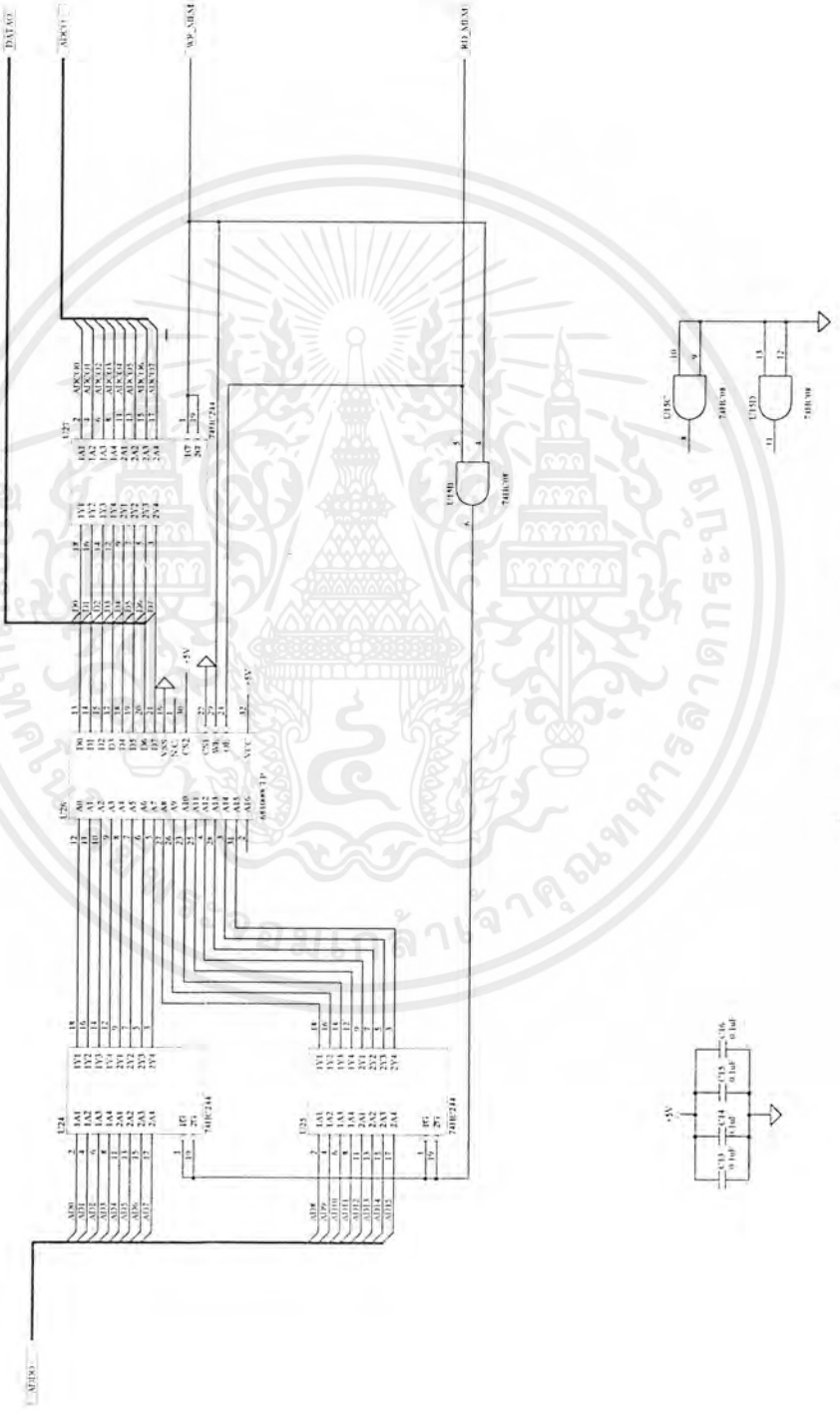
ขา OVER FLOW ไม่ใช้งานก็ปล่อยลอยไว้

หมายเหตุ

อนาล็อกกราวด์และดิจิตอลกราวด์ต้องแยกออกจากกันให้ชัดเจนเพื่อไม่ให้มีสัญญาณรบกวนจากสัญญาณอนาล็อกไปกวนส่วนของดิจิตอลและจะนำกราวด์ทั้งสองมารวมกันอีกครั้งก่อนที่ต่อเข้ากับสล็อตของคอมพิวเตอร์

3.2.2 วงจรหน่วยความจำ

วงจรหน่วยความจำทำหน้าที่เขียนข้อมูลที่ได้จากการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลและอ่านข้อมูลจากหน่วยความจำไปให้คอมพิวเตอร์ตามการเขียนโปรแกรมโดยมี U24(74HC244), U25 (74HC244) ทำหน้าที่เป็นตัวป้องกันแอดเดรสและ U27(74HC244) ทำหน้าที่เป็นตัวกันชนการเขียนข้อมูล และ U26(KM681000cpl) เป็นหน่วยความจำขนาด 128 κ byte เท่านั้น



รูปที่ 3.6 วงจรหน่วยความจำ

หลักการทำงานเริ่มจากคอมพิวเตอร์ส่งสัญญาณมาเคลียร์วงจรนับทำให้พอร์ทการเขียนข้อมูล (WR_MEM)และพอร์ทการอ่านข้อมูล(RE_MEM) เป็น"1"ทั้งคู่ทำให้ U24(74HC244),U25(74HC244), U27(74HC244)เป็นไฮอิมพีแดนซ์ทำให้ไม่มีการอ่านข้อมูลและเขียนข้อมูลมาที่หน่วยความจำ

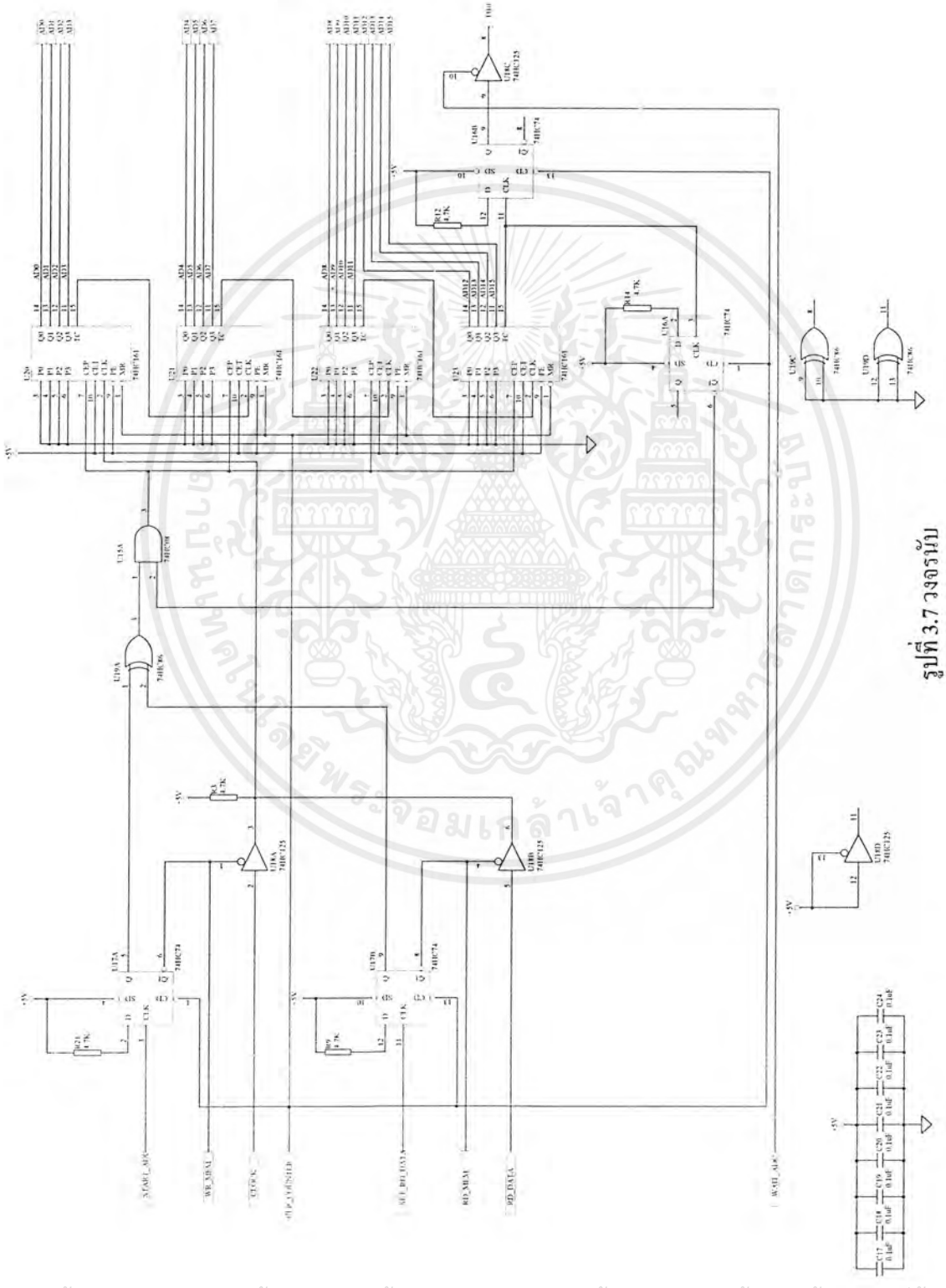
เมื่อต้องการเขียนข้อมูลมาที่หน่วยความจำจะต้องส่งสัญญาณมาที่พอร์ทการเขียนข้อมูลเป็น"0" ทำให้ U27(74HC244)หรือตัวป้องกันการเขียนข้อมูลเปิดเกท และเป็นผลทำให้ขา WE ของหน่วยความจำถูกอีเนเบิล แต่พอร์ทการเขียนข้อมูลยังเป็น "1"อยู่ ทำให้ขา OE ของหน่วยความจำถูกดิสเซเบิ้ลและขา 4,5 ของ U15(74HC08) เป็นอินพุทของแอนท์เกท(And Gate) ซึ่งทั้งสองขาต่ออยู่กับพอร์ทการเขียนข้อมูล และพอร์ทการเขียนข้อมูลทำให้ได้เอาท์พุทที่ขา 6 ของ U15(74HC08)เป็น "0" ทำให้ U24 ,U25,U27 (74HC244) เปิดเกท จึงมีการเขียนข้อมูลลงในหน่วยความจำโดยตำแหน่งที่มีการเขียนข้อมูลถูกเลื่อนตำแหน่งแอดเดรสโดยวงจรนับ โดยข้อมูลถูกเขียนจากเอทูดีมายังพอร์ทข้อมูล DA(0-7) เพื่อทำการเขียนข้อมูลลงในหน่วยความจำ

เมื่อมีการเขียนข้อมูลจนเต็ม 64 K byte จะต้องทำการเคลียร์วงจรนับอีกทำให้พอร์ทการเขียนข้อมูลและพอร์ทการข้อมูลเป็น "1" ทั้งคู่ทำให้ U24, U25, U27(74HC244) เป็นไฮอิมพีแดนท์ และไม่มีการเปิดหน่วยความจำ

เมื่อต้องการอ่านข้อมูลจากหน่วยความจำเพื่อนำข้อมูลไปเก็บไว้ในหน่วยความจำของคอมพิวเตอร์ คอมพิวเตอร์จะต้องทำการส่งสัญญาณการอ่านข้อมูลเป็น "0"และพอร์การเขียนข้อมูลเป็น "1" ทำให้ U27(74HC244) หรือตัวป้องกันข้อมูลเปิดเกท เป็นผลทำให้ขา WE ของหน่วยความจำถูกดิสเซเบิลแต่พอร์ทการอ่านข้อมูลเป็น "0"ทำให้ขา OE ของหน่วยความจำถูกอีนาเบิลและขา 4, 5 ของU15 (74HC08) เป็นอินพุทของแอนท์เกท ซึ่งทั้ง 2 ขาต่ออยู่กับพอร์ทการเขียนข้อมูลและพอร์ทการอ่านข้อมูล ทำให้ได้เอาท์พุทที่ขา 6 ของ U15(74HC08) เป็น "0" ทำให้ U24, U25(74HC244)เปิดเกททำให้มีการอ่านข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์ โดยตำแหน่งที่มีการอ่านข้อมูลถูกเลื่อนแอดเดรสโดยวงจรนับ

3.2.3 วงจรนับ

วงจรนับทำหน้าที่เป็นตัวนับเพื่อเลือกตำแหน่งแอดเดรสในการอ่านและเขียนหน่วยความจำและคอยตรวจสอบว่ามีการนับตำแหน่งของแอดเดรสเต็ม 64 กิโลไบท์



รูปที่ 3.7 วงจรนับ

จากรูปที่ 3.7 จะเห็นว่าวงจรมีไอซีเบอร์ 74HC161 ซึ่งเป็นวงจรนับ 4 บิท จะทำการนับได้ก็ต่อเมื่อขา ENP และ ENT ทั้งคู่เป็น "1" เท่านั้น และวงจรนี้สามารถอ้างตำแหน่งแอดเดรสได้ทั้งหมด 64 กิโลไบท์ จากวงจรขา A, B, C, D ของ 74HC161(U20-U23) ต่อกราวด์เพื่อให้เริ่มต้นนับที่แอดเดรส "0000"และไม่ใช้ขา LOAD จึงต่อไฟแบบ +5V และ ENT ต่อไฟ +5 V จากวงจรทั้งหมดอธิบายได้ดังนี้

1. คอมพิวเตอร์จะส่งสัญญาณนาฬิกามา 1 ลูก ที่พอร์ทเคลียร์ตัวนับ(CLR_Coounter) ทำให้มีการเคลียร์ฟลิบ-ฟลอบและเคลียร์วงจรนับทำให้

1.1 ขา 5(U17) เป็น "0" และขา 6(U17)เป็น "1"

1.2 ขา 9(U17) เป็น "0" และขา 8(U17)เป็น "1"

1.3 U18ปิดเกทจึงไม่มีสัญญาณนาฬิกามาที่วงจรนับ

1.4 ขา 1และขา 2 ของ(U19)เป็น "0" ทำให้ขา 3(U19) เป็น "0"

1.5 ขา 6(U16) เป็น "1" ต่อเข้าอินพุทที่ขา 2(U15)

1.6 ขา 3(U15) เป็น "0" ทำให้วงจรนับไม่ทำงาน

1.7 ขา 9(U18) เป็น "0"

2. คอมพิวเตอร์จะส่งสัญญาณมาที่พอร์ทเลือกสัญญาณนาฬิกา(Set_Clock)ซึ่งอยู่ที่วงจรผลิตสัญญาณนาฬิกา ทำให้มีการส่งสัญญาณนาฬิกามาที่พอร์ทสัญญาณนาฬิกา(Clock) แต่ก็ยังไม่มีการเปิดเกทของ U18(74HC125) ทำให้ไม่มีสัญญาณนาฬิกาเข้ามาในวงจรนับ

3. เมื่อคอมพิวเตอร์ต้องการเขียนข้อมูลจากเอทูดี คอมพิวเตอร์จะส่งสัญญาณมาที่พอร์ทการเขียนข้อมูล(Start_ADC)โดนส่งสัญญาณนาฬิกามา 1 ลูก ทำให้

3.1 ขา 5(U17)เป็น "1"และขา 6(U17) เป็น "0"

3.2 ขา 9(U17)เป็น "0"และขา 8(U17)เป็น "1"

3.3 U18 เปิดเกททำให้มีสัญญาณนาฬิกาเข้ามาที่วงจรนับและส่ง "0" ไปพอร์ตเขียนหน่วยความจำทำให้หน่วยความจำ (WR_MEM) เพื่อเปิดตัวป้องกันข้อมูลและเปิดขา WR ของหน่วยความจำในวงจรหน่วยความจำ ทำให้หน่วยความจำถูกอีนาเบิลจึงมีการเขียนข้อมูลลงหน่วยความจำได้

3.4 ขา 1(U19) เป็น "1" และขา 6 (U17) เป็น "0" ทำให้ขา 3(U19)เป็น "1"

3.5 ขา 6(U16) เป็น "1" ต่อเข้ามาที่อินพุทที่ขา 2(U15)

3.6 ขา 3(U15) เป็น "1" ทำให้วงจรนับ (U20-U23) เริ่มทำงาน

3.7 เมื่อมีสัญญาณนาฬิกามา 1 ลูก ก็จะเริ่มนับตำแหน่งแอดเดรสแรกทำให้ความจำชี้แอดรสแรกและมีการเขียนข้อมูลจากเอทูดีไปสู่หน่วยความจำแอดเดรส 0000H

3.8 ขา 15 ของ U23 เป็น "0" ทำให้ขา 9(U16) เป็น "0"

3.9 คอมพิวเตอร์จะทำการเช็คทันทีว่ามีการเขียนข้อมูลจากเอทูดีไปสู่หน่วยความจำเต็ม 64 กิโลไบท์หรือยังโดยส่งสัญญาณมา 1 ลูก ที่พอร์ทรอการเขียนข้อมูล (WAIT_ADC_RD) ทำให้ U18 เปิดเกทเอาร์พุทออกมาที่ขา 8(U18) เป็น "0" ค่าที่ได้จึงถูกส่งออกไปที่คอมพิวเตอร์ทำให้คอมพิวเตอร์รู้ว่ามีการเขียนข้อมูลไปตำแหน่งใดบ้างแล้ว

3.10 เมื่อมีสัญญาณนาฬิกาลูกที่ 2 เข้ามาวงจรนับก็จะอ้างตำแหน่งแอดเดรสถัดไป

3.11 มีการเขียนข้อมูลจากเอทูดีไปหน่วยความจำในตำแหน่งแอดเดรสถัดไป

3.12 ไปทำที่หัวข้อ 3.9

3.13 มีสัญญาณนาฬิกาลูกถัดไปเข้ามาก็เหมือนข้อ 3.12 จนกระทั่งสัญญาณนาฬิกาลูกที่ 64 กิโลไบท์ เข้ามาทำให้ขา 15(U23) มีสัญญาณเป็นพัลซ์มา 1 ลูกทำให้ขา 9(U16)เป็น "1"

3.14 คอมพิวเตอร์จะส่งสัญญาณนาฬิกามาหนึ่งลูกที่พอร์ทรอการเขียนข้อมูลทำให้ U18 เปิดเกทมีเอาท์พุทออกมาที่ขา 8(U18) เป็น "1" ค่าที่ได้จึงถูกส่งไปที่คอมพิวเตอร์ ทำให้คอมพิวเตอร์รู้ว่ารู้ว่ามีการเขียนข้อมูลเต็ม 64 กิโลไบท์แล้ว

4. คอมพิวเตอร์จะส่งสัญญาณนาฬิกามา 1 ลูกที่พอร์ทเคลียร์ตัวนับ(CLR_Counter) ทำให้มีการเคลียร์ฟลิบ-ฟลอบและเคลียร์วงจรนับทำให้ทำให้สถานะต่างๆเหมือนข้อ 1

5. คอมพิวเตอร์ทำการอ่านข้อมูลจากหน่วยความจำไปเก็บไว้ที่คอมพิวเตอร์โดยคอพิวเตอร์จะส่งสัญญาณมาที่พอร์ทเลือกการอ่านข้อมูล(Set_RD_DATA)โดยส่งสัญญาณมา 1 ลูก ทำให้

5.1 มีสัญญาณมาที่ขา 11 (CLK,U17B)

5.2 ขา 9(U17) เป็น "1" และขา 8(U17) เป็น "0"

5.3 U18Bเปิดเกทส่ง "0" ไปยังพอร์ทการอ่านหน่วยความจำ (RD_MEM) เพื่อเปิดตัวป้องกันข้อมูลและปิดขา OE ของหน่วยความจำทำให้หน่วยความจำอีนาเบิลการอ่านข้อมูล

5.4 ขา 1(U19) เป็น "0" และขา 2 (U19)เป็น "1" ทำให้ขา 3 (U19) เป็น "1"

5.5 ขา 6(U16) เป็น "1" ต่อเข้าอินพุท ที่ขา 2(U15)

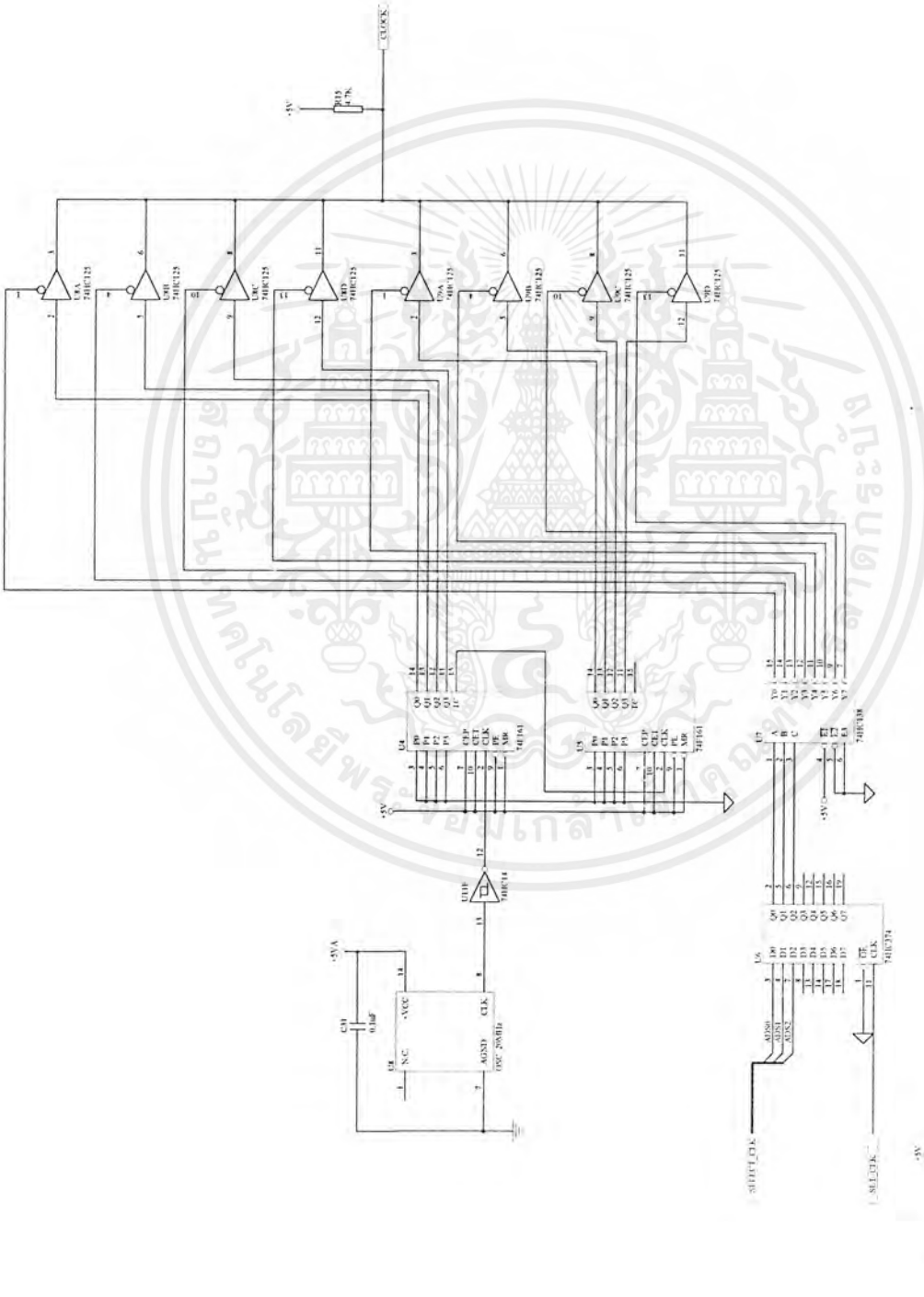
5.6 ขา 3 (U15) เป็น "1" ทำให้วงจรนับเริ่มทำงาน

5.7 คอมพิวเตอร์ส่งสัญญาณนาฬิกามา 1 ลูกที่พอร์ทการอ่านข้อมูลทำให้มีสัญญาณนาฬิกาเข้ามาที่วงจรนับทำให้วงจรนับเริ่มนับและมีการอ่านข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์ในตำแหน่งแอดเดรส "000"

5.8 เมื่อมีสัญญาณนาฬิกามาที่พอร์ทการอ่านข้อมูลลูกที่ 2 วงจรนับก็เริ่มทำการนับแอดเดรสตำแหน่งแอดเดรสถัดไป และมีการอ่านข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์ ในตำแหน่งแอดเดรส "0002"

5.9 เมื่อมีสัญญาณนาฬิกามาที่พอร์ทการอ่านข้อมูลลูกถัดไปเข้ามาในวงจรนับก็จะอ้างแอดเดรสถัดไปและมีการอ่านข้อมูลจากหน่วยความจำไปสู่คอมพิวเตอร์เป็นเช่นนี้ไปเรื่อยๆ จนกระทั่งมีการอ่านข้อมูลจากหน่วยความจำจนหมด

3.2.4 วงจรผลิตสัญญาณนาฬิกา จะเป็นตัวกำเนิดสัญญาณนาฬิกาดังรูปที่ 3.8



รูปที่ 3.8 วงจรผลิตสัญญาณนาฬิกา

จากรูปที่ 3.8 จะมีออสซิลเลเตอร์ผลิตความถี่ประมาณ 30 เม็กกะเฮิร์ต เป็นตัวกำเนิดสัญญาณนาฬิกาและป้อนให้กับ U4(74HC161) ซึ่งทำหน้าที่เป็นวงจรนับมีขนาด 4 บิท แล้วนำตัวทดไปป้อนให้ขาสัญญาณนาฬิกาของ U5(74HC161) ซึ่งเป็นวงจรนับ 4 บิทเช่นกันจึงเปรียบเสมือนทั้ง 2 ตัวต่ออนุกรมกันทำให้ได้วงจรนับ 8 บิท โดยขา A,B,C,D ทั้ง 2 ตัวเป็นค่าเริ่มต้นในการนับโดยต่อทั้ง 4 ขาลงกราวด์เพื่อเริ่มต้นในการนับที่ตำแหน่งแอดเดรส "0000H" ส่วนขา CLR,LOAD ไม่ได้ใช้ต่อ +5 โวลท์ส่วนขา ENT,ENP ต้องเป็น "1"ทั้งคู่ มิฉะนั้นเมื่อมีสัญญาณนาฬิกาเข้ามาก็จะไม่มีการนับ โดยจะได้สัญญาณนาฬิกาที่ขาเอาท์พุทต่างๆที่มีความถี่ต่างกันดังตารางที่ 3.3

หมายเลขขา	ความถี่
ขา 14,U4	10*MHz*
ขา 13,U4	5*MHz*
ขา 12,U4	2.5*MHz*
ขา 11,U4	1.25*MHz*
ขา 14,U5	625*kHz*
ขา 13,U5	312.5*kHz*
ขา 12,U5	156.25*kHz*
ขา 11,U5	78.125*kHz*

ตารางที่ 3.3 ความถี่ของสัญญาณนาฬิกาที่ขาต่างๆ

เอาท์พุททุกขาจะไปต่อเข้ากับ U8,U9(74HC125) ซึ่งทำหน้าที่เป็นตัวกันชนแต่ก็ยังไม่มีเอาท์พุทออกมาได้จนกว่าจะมีการถอดรหัสโดย U7(74HC138) ซึ่งทำหน้าที่ในการเปิดเกททั้ง 8 ตัวเพื่อจะเลือกสัญญาณนาฬิกาโดนมี U6(74HC374) ทำหน้าที่เป็นตัวจำข้อมูล(Latch) เพื่อเอาค่าที่ได้ไปป้อนให้เอาท์พุทให้ขา 74HC138 เพื่อนำไปถอดรหัสต่อ โดย U6 จะต่อใช้งานเพียง 3 ขา เพื่อใช้จำข้อมูล 3 ค่าคือ AD0,AD1,AD2 ของคอมพิวเตอร์แต่จะต้องมีสัญญาณนาฬิกาเข้ามาที่ขา 11(U6) มิฉะนั้นข้อมูลที่ทำการจำไว้ก็จะไม่สามารถผ่านออกมาได้ทำให้การถอดรหัสผิดไปก็ได้ การทำงานของวงจรนี้จะมีการสร้างสัญญาณนาฬิกาได้ 8 ค่ามารออยู่ที่ตัวกันชน U8,U9 และจะเปิดเกทเพื่อเลือกสัญญาณนาฬิกามาค่าหนึ่งโดย

1. ส่งค่า AD0,AD1,AD2 มาที่ U6(74HC374) โดยสามารถเลือกได้ 8 พอร์ท
2. ส่งสัญญาณนาฬิกามาที่พอร์ทเลือกสัญญาณนาฬิกา (Set Clock)เพื่อป้อนให้ขา CLK ของ U6
3. มีข้อมูลออกมาที่ขา Q0,Q1,Q2 ของ U6
4. นำ Q0 ต่อขา A ของ U7,Q1 ต่อขา B ของ U7,Q2 ต่อขา C ของ U7

5. ทำให้ Y0-Y7 "ON" เส้นใดเส้นหนึ่งเท่านั้น จนกว่าจะมีการเปลี่ยนค่า AD0-AD2

6. จะได้สัญญาณนาฬิกา 1 ความถี่เพื่อป้อนให้ส่วนอื่นต่อไป

3.2.5 วงจรถอดรหัส

เป็นวงจรถอดรหัสเพื่อควบคุมการทำงานต่างๆดังรูปที่ 3.9

วงจรถอดรหัสเป็นวงจรควบคุมการทำงานต่างๆโดยผ่านทางสล็อตคอมพิวเตอร์ จากขา B1-B31 จะต่อเพียงบางขาเพื่อป้อนเป็นแหล่งจ่ายไฟให้กับวงจรทั้งหมดโดยผ่านทางสล็อตคอมพิวเตอร์ โดยมีขาต่างๆที่นำออกมาถอดรหัสเนื่องจากคอมพิวเตอร์มีพอร์ทที่ว่างอยู่ตั้งแต่แอดเดรส "3000H-301F" จึงใช้พอร์ทที่ส่งออกมาถอดรหัสเพื่อไม่ให้ชนกับส่วนอื่นๆ ของคอมพิวเตอร์โดยมี U12(74HC30) ทำหน้าที่เป็นตัวเข้ารหัส 8 อินพุท 1 เอาท์พุทเพื่อเลือกตำแหน่งแอดเดรสออกไปหนึ่งค่าเท่านั้นโดยมีอินพุทต่างๆ ดังต่อไปนี้

1. AD10 (คอมพิวเตอร์)ต่อผ่านน๊อตเกท (U11) ไปเข้าที่ขา 1 ของ U12
2. AD9 (คอมพิวเตอร์) ต่อขา 2 ของ U12
3. AD8 (คอมพิวเตอร์) ต่อขา 3 ของ U12
4. AD7 (คอมพิวเตอร์) ต่อผ่านน๊อตเกท (U11) ไปเข้าที่ขา 4 ของ U12
5. AD6 (คอมพิวเตอร์) ต่อผ่านน๊อตเกท (U11) ไปเข้าที่ขา 5 ของ U12
6. AD7 (คอมพิวเตอร์) ต่อผ่านน๊อตเกท (U11) ไปเข้าที่ขา 6 ของ U12
7. ขา 11,12 ของ U12 ต่อ +VCC

เพราะฉะนั้น เอาท์พุทของ U12 (ขา18) จะเป็น "0" ได้ก็ต่อเมื่อ

AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
0	1	1	0	0	0	X	X	X	X	X

ตาราง 3.4 การถอดรหัสที่เอาท์พุทของ U12

จะได้เอาท์พุทที่ขา 8 ของ U12 เป็นเบอร์พอร์ทหมายเลข 0300H-031FH แต่จะนำไปใช้เพียง 16 พอร์ท คือ 0300H-030FH ก็เพียงพอสำหรับการถอดรหัสแล้ว เอาท์พุทที่ได้จะต่อเข้าขา G2B ของ U14 (74HC138) โดยมีขา IOR ของคอมพิวเตอร์ต่อเข้าที่ขา G2A ของ U14(74HC138) และขา AEN กับ AD9 ของคอมพิวเตอร์ต่อเข้ากับอินพุทของ U19 และเอาท์พุทที่ได้ต่อเข้าที่ขา G1 ของ U14(74HC138) เอาขา AD3,AD4 ของคอมพิวเตอร์ต่อเข้าขา A,B ของ U14 และขา C ของ U14 ต่อกราวด์

เหตุที่นำ AEN มาใช้ ก็เพื่อป้องกันการถอดรหัสผิดพลาดเพราะถ้า AEN เป็น "1"จะทำให้พอร์ทที่ถอดรหัสมาผิด ถ้า AEN เป็น "0" ก็จะทำให้แอดเดรสที่นำมาถอดรหัสไม่มีการถอดรหัสผิด เมื่อ AEN เป็น "0" และ AD9 เป็น "1" ทำให้ขา G1 ของ U14 เป็น "1" ด้วย ส่วนขา IOR ของคอมพิวเตอร์ต่อขา G2A เพื่อให้มีการอ่านข้อมูลจากภายนอกได้อย่างเดียว เพราะฉะนั้น U14 จะถอดรหัสค่าต่างๆได้ดังนี้

แอดเดรสของคอมพิวเตอร์

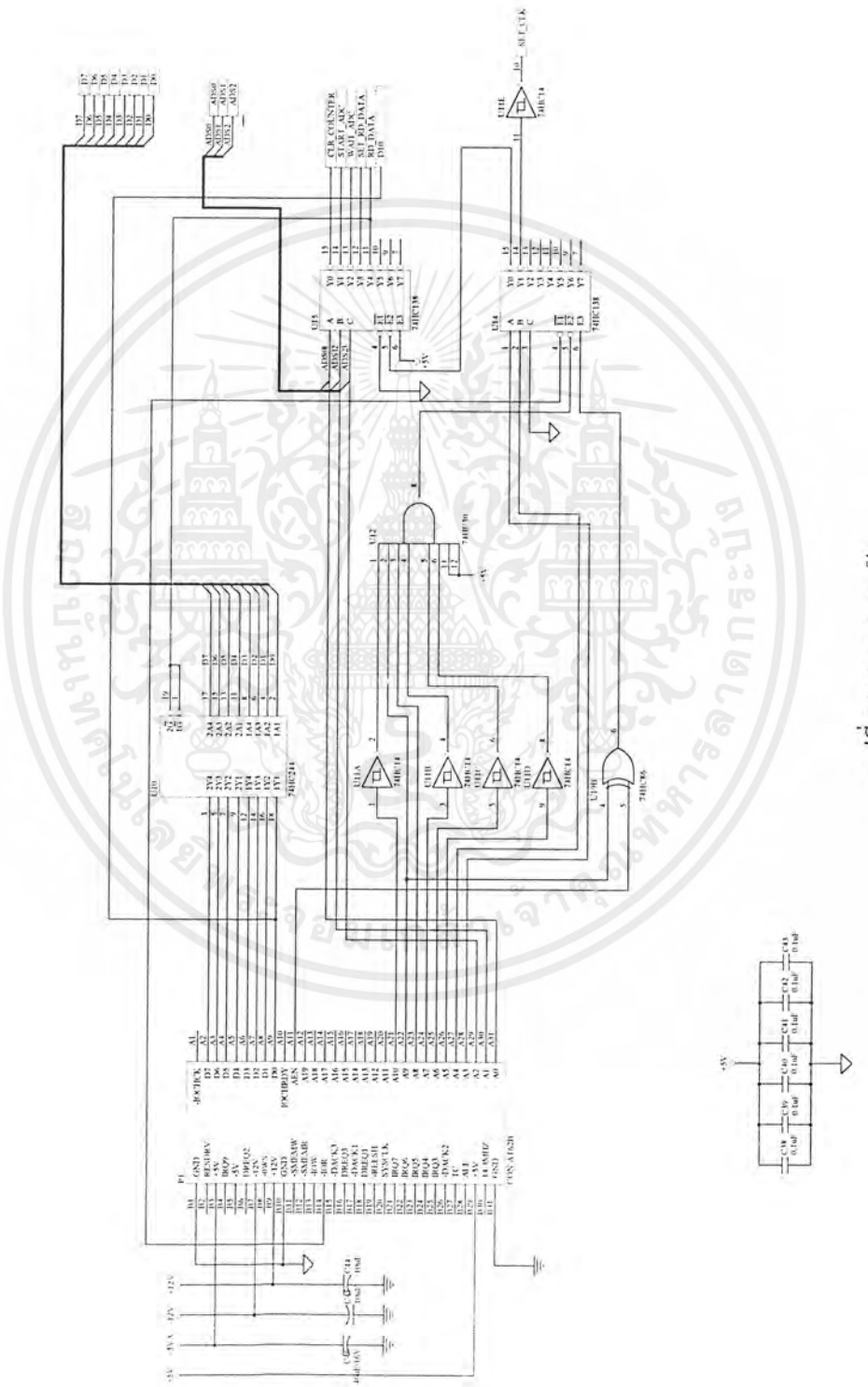
10	9	8	7	6	5	4	3	2	1	0	O/P	แอดเดรส
0	1	1	1	0	0	0	0	X	X	X	Y0	0300-0307H
0	1	1	1	0	0	0	1	X	X	X	Y1	0308-0301H

ตาราง 3.5 การถอดรหัสของ U14

ขา Y1 ของ U14 จะไปผ่านน๊อตเกทเพื่อทำการเลือกพอร์ทสัญญาณนาฬิกาเพื่อเลือกสัญญาณนาฬิกาความถี่ค่าต่างๆ เหตุที่ต้องผ่านน๊อตเกทเพราะขา CLK ของ U6 จะทำงานที่สัญญาณนาฬิกาที่ขอบขาขึ้นเท่านั้นจึงต้องมีการกลับเกทโดยใส่น๊อตเกทดังวงจรเพื่อใช้เรียกพอร์ท 0308H-030FH ส่วนขา Y0 ของ U14 จะไปเข้าที่ขา G2B ของ U13 ส่วน G2A ต่อกราวด์ G1 ต่อ +5V เมื่อ G2Bเป็น "0"หมายความว่าคอมพิวเตอร์ต้องการจะถอดรหัสพอร์ทหมายเลข 0300H-0307H เพื่อเลือกสัญญาณควบคุมการอ่านข้อมูลและเขียนข้อมูลลงบนหน่วยความจำ โดยขา A B C ของ U13 จะต่อ AD0, AD1,AD2 ตามลำดับ โดยจะมีการถอดรหัสของ U13 ได้ดังนี้

10	9	8	7	6	5	4	3	2	1	0	O/P	หมายเลขพอร์ท	หน้าที่
0	1	1	0	0	0	0	0	0	0	0	Y0	0300H	เคลียร์วงจรนับ
0	1	1	0	0	0	0	0	0	0	1	Y1	0301H	เขียนข้อมูล
0	1	1	0	0	0	0	0	0	1	0	Y2	0302H	รอการเขียนข้อมูล
0	1	1	1	0	0	0	0	0	1	1	Y3	0303H	เลือกอ่านข้อมูล
0	1	1	0	0	0	0	1	1	0	0	Y4	0304H	อ่านข้อมูล

ตารางที่ 3.6 การถอดรหัสของ U12



รูปที่ 3.9 วงจรถอดรหัส

ส่วนอีก 3 พอร์ทไม่ได้ใช้และ U10(74HC244)ทำหน้าที่เป็นตัวป้องกันการเขียนข้อมูลจะมีหน้าที่ในการอ่านข้อมูลจากหน่วยความจำของวงจรมาเก็บไว้ในหน่วยความจำของคอมพิวเตอร์เท่านั้น

ส่วนพอร์ท DI0 ต่อเข้าที่ขา SD0 ของคอมพิวเตอร์เพื่อทำหน้าที่ตรวจสอบการเขียนข้อมูลจากเอฟูดีไปสู่หน่วยความจำ เพื่อทำให้คอมพิวเตอร์ทราบว่ามีการเขียนข้อมูลลงหน่วยความจำเต็มหรือยัง

สรุปการทำงานของวงจร

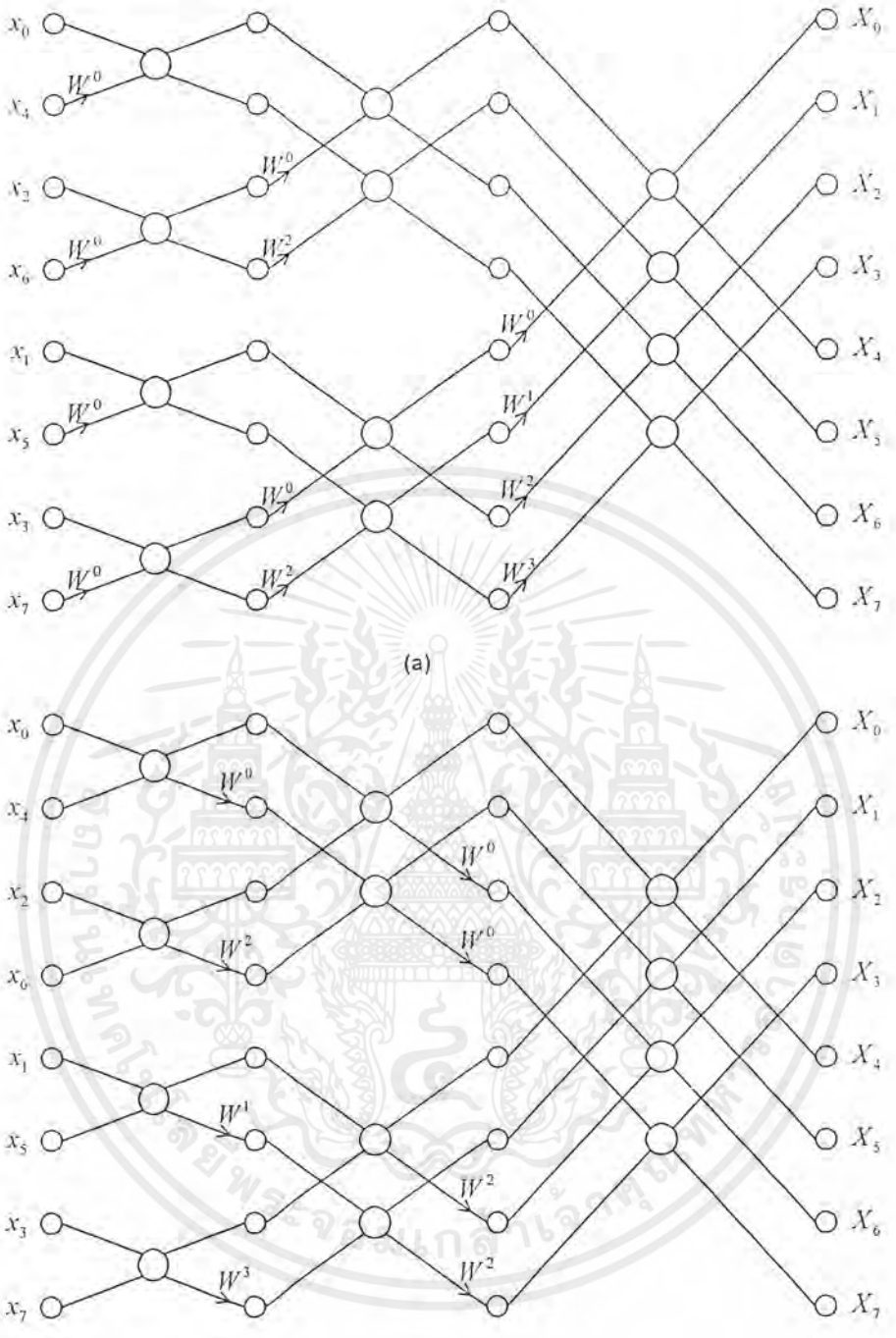
การทำงานของวงจรนี้จะเป็นวงจรของการถอดรหัสเพื่อเลือกการเขียนข้อมูลจากเอฟูดีไปสู่หน่วยความจำและอ่านข้อมูลจากหน่วยความจำเข้าไปเก็บไว้ในคอมพิวเตอร์ โดยจะถูกควบคุมโดยโปรแกรมคอมพิวเตอร์

3.3 หลักการเขียนโปรแกรมฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

ในการคำนวณหาฟาสต์ฟูเรียร์ทรานส์ฟอร์ม ($N\log_2(N)$) และของดีสครีทฟูเรียร์ทรานส์ฟอร์ม คือ N^2 (N = ความยาวในการคำนวณ) จะแตกต่างในเรื่องของความเร็วในการคำนวณ โดยฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะทำการคำนวณได้เร็วกว่า แต่ฟาสต์ฟูเรียร์ทรานส์ฟอร์มมีข้อจำกัดที่จะต้องนำมาพิจารณาด้วยคือ

1. วิธีการหาฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะใช้ข้อมูลในการคำนวณแค่ 2 ตัวในบัตเตอร์ฟลายหนึ่งตัวและมีความยาวในการคำนวณเท่ากับข้อมูลทั้งหมดยกกำลังสอง (N^2) โดยการคำนวณต้องเริ่มจากค่า 0 จนถึงค่า N
2. เมื่อมีการคำนวณฟาสต์ฟูเรียร์ทรานส์ฟอร์มต้องมีการกลับบิทเมื่อคำนวณเสร็จแล้ว (นำข้อมูลที่คำนวณแล้วมาเรียงใหม่ตามลำดับ)

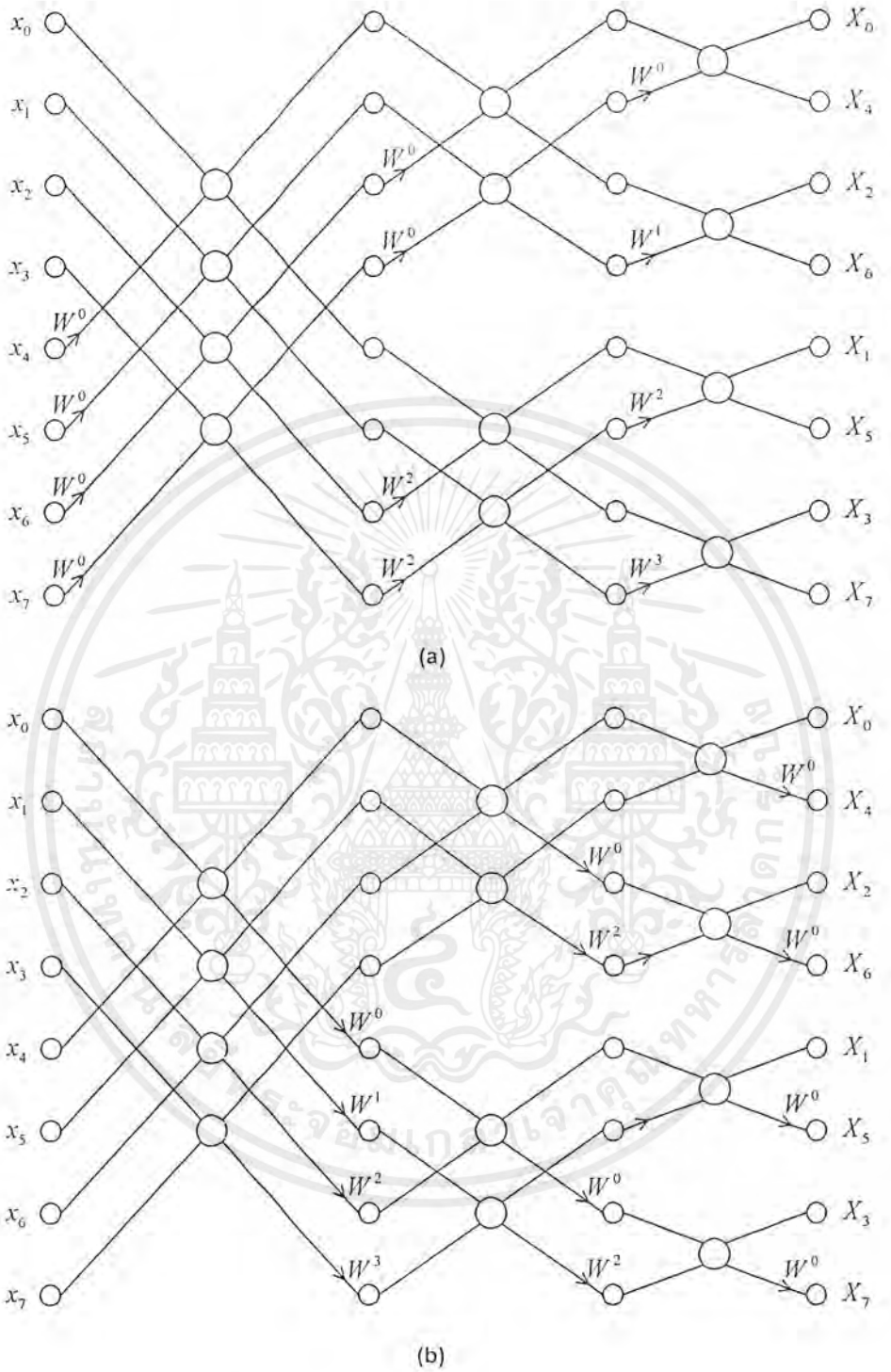
ทั้งสองข้อต่างเป็นข้อบกพร่องของการหาฟาสต์ฟูเรียร์ทรานส์ฟอร์มที่ยุ่งยาก แต่มีวิธีการหาฟาสต์ฟูเรียร์ทรานส์ฟอร์มอยู่ 2 ทางเรียกว่าเดซิเมชั่นอินไทม์ (Decimation in Time)และ เดซิเมชั่นอินฟรีเควนซี่ (Decimation in Frequency) ซึ่งวิธีการทั้ง 2 นี้เป็นพื้นฐานของฟาสต์ฟูเรียร์ที่ใช้ในการคำนวณในแต่ละจุดของฟาสต์ฟูเรียร์ทรานส์ฟอร์มและทั้งสองรูปแบบนี้ก็มีข้อแตกต่าง ดังรูปที่ 3.10 (บิทอินพุทกลับด้านส่วนเอาท์พุทจะปกติ) และในรูปที่ 3.11 จะแสดง (บิทอินพุทปกติแต่เอาท์พุทกลับด้าน)



รูปที่ 3.10 กราฟ 8 จุด ฟาสต์ฟูเรียร์ทรานส์ฟอร์ม (อินพุทกลับบิท)

(a) กราฟเดซิเมชั่นอินไทม์ (Decimation in Time Flowgraph)

(b) กราฟเดซิเมชั่นอินฟรีเควนซี่ (Decimation in Frequency Flowgraph)

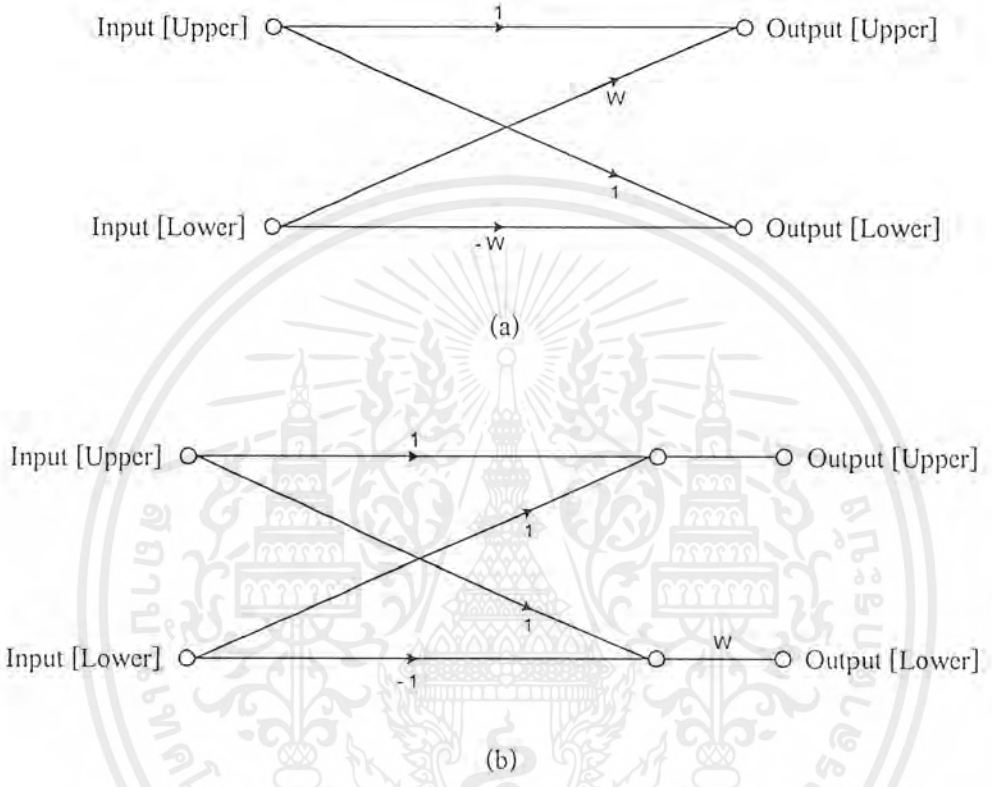


รูปที่ 3.11 กราฟ 8 จุด ฟาสต์ฟูเรียร์ทรานส์ฟอร์ม (อินพุทไม่กลับบิท)

(a) กราฟเดซิเมชั่นอินไทม์ (Decimation in Time Flowgraph)

(b) กราฟเดซิเมชั่นอินฟรีเควนซี่ (Decimation in Frequency Flowgraph)

จากกราฟทั้ง 4 สามารถแบ่งบัตเตอร์ฟลายได้ 2 ชนิด คือ กราฟเดซิเมชั่นอินไทม์และกราฟเดซิเมชั่นอิฟรีเควนซี่ซึ่งมี 2 แบบคือแบบกลับบิทและไม่กลับบิทที่เอาท์พุท เพื่อให้ง่ายขึ้นในการทำเข้าใจจากกราฟจะนำมาอธิบายแค่บัตเตอร์ฟลายเดียว (2 อินพุท 2 เอาท์พุท) ดังรูปที่ 3.12



รูปที่ 3.12 บัตเตอร์ฟลายเดียว

(a) เดซิเมชั่นอินไทม์ มีการคูณด้วยค่า W ก่อนทำการบวกและลบ

(b) เดซิเมชั่นอินฟรีเควนซี่ มีการบวกและลบก่อนทำการคูณด้วยค่า W

โดยสมการของ เดซิเมชั่นอินไทม์บัตเตอร์ฟลาย คือ

$$\text{Output (Upper)} = \text{Input (Upper)} + W * \text{Input (Lower)}$$

$$\text{Output (Lower)} = \text{Input (Upper)} - W * \text{Input (Lower)}$$

สมการของ เดซิเมชั่นอินฟรีเควนซี่บัตเตอร์ฟลาย คือ

$$\text{Output (Upper)} = \text{Input (Upper)} + \text{Input (Lower)}$$

$$\text{Output (Lower)} = W * [\text{Input (Lower)} - \text{Input (Lower)}]$$

ในหลักการเขียนโปรแกรมนี้จะใช้วิธีการแบบเดซิเมชั่นอินฟรีเควนซี่บัตเตอร์ฟลายและจะต้องมีการเก็บค่าข้อมูลที่ได้ให้คุ้มค่าเพื่อประหยัดหน่วยความจำในการเก็บข้อมูลโดยจะต้องนำข้อมูลที่คำนวณได้ไปเก็บไว้ที่ตำแหน่งข้อมูลเดิมและทำการคำนวณเช่นนี้ไปเรื่อยๆ จนกระทั่งคำนวณเสร็จ ในการคำนวณแบบเดซิเมชั่นอินฟรีเควนซี่ จะมีข้อมูลที่ต้องการดังนี้

(1) การชี้ตำแหน่งขาบน (Upper Leg) ของอินพุท

(2) ความแตกต่างระหว่างขาบน (Upper Leg) และขาล่าง (Lower Leg)

(3) ค่าของ W (การเก็บค่า W อาร์เรย์)

ตัวแปรที่ใช้ในการคำนวณฟาสต์ฟูเรียร์ทรานส์ฟอร์ม

x : ชี้ค่าเริ่มต้นของคำสั่งจำนวนเชิงซ้อนซึ่งรอรับข้อมูลของอินพุทและเอาร์พุท

I : การอ้างถึงตำแหน่งขาบนใน x อาร์เรย์

le : การอ้างถึงความแตกต่างระหว่างขาบนและขาล่าง

wptr : เป็นจุดที่ใช้เก็บค่าของ W

n : เป็นจำนวนจุดต่างๆที่ใช้คำนวณ

m : เป็นตัวยกกำลังของ n

temp : เก็บค่าจำนวนเชิงซ้อนที่เป็นค่าจริงไว้ชั่วคราว

tm : เก็บค่าจำนวนเชิงซ้อนที่เป็นค่าจินตภาพไว้ชั่วคราว

จากรูปที่ 3.12 สามารถเขียนโปรแกรมของเดซิเมชั่นอินฟรีเควนซี่บัตเตอร์ฟลายได้ดังนี้

```
xi = x + i
xip = xi + le
temp.real = xi->real + xip->real
temp.imag = xi->imag + xip->imag
tm.real   = xi->real - xip->real
tm.imag   = xi->imag - xip->imag
xip->real = tm.real*u.real - tm.imag*u.imag
xip->imag = tm.real*u.imag + tm.imag*u.real
*xi = temp
```

โดย xi แทนตำแหน่งอินพุทขาบน

xip แทนตำแหน่งอินพุทขาล่าง

temp แทนตำแหน่งเอาท์พุทขาบนชั่วคราวซึ่งมีค่าจริงและค่าจินตภาพ

temp แทนตำแหน่งเอาท์พุทขาล่างก่อนคูณกับค่า W ซึ่งมีค่าจริงและค่าจินตภาพ

การหาค่าเอาท์พุทขาล่างจะหามาจากสมการ

(a + jb) * (c + jd) = ac + jad + jbc – bd

= (ac –bd) + j(ad + bc)

เอาท์พุทขาล่าง = (tm.teal + j tm.imag) (wptr.real + j wptr.imag)

เอาท์พุทขาล่าง = (tm.real * wptr.real) + (tm.real * j wptr.imag) +

(j tm.imag * wptr.real) – (tm.imag * wptr.imag)

เอาท์พุทค่าจริงขาล่าง = (tm.real * wptr.real) – (tm.imag * wptr.imag)

เอาท์พุทค่าจินตภาพขาล่าง = (tm.real * wptr.imag) + (tm.imag * wptr.real)

ที่บรรทัดสุดท้ายของโปรแกรมจะใช้ตัวแปร temp เก็บค่าเอาท์พุทขาบนไว้ชั่วคราว และจะเปลี่ยนเป็นตัวแปร xi อีกครั้งเพื่อใช้ในการคำนวณครั้งต่อไป

ในการคำนวณบัตเตอร์ฟลายหลายตัวจะยกตัวอย่างการเขียนโปรแกรมในรูปที่ 3.11(b) กราฟเดซิเมชั่นอินฟรีเควนซี่ การหาบัตเตอร์ฟลายในแนวตั้งจะมีจำนวนแถวทั้งหมดเท่ากับ ($\log_2 N$) แต่ละแถวของบัตเตอร์ฟลายจะต้องคำนึงถึงค่า W ด้วย ในแต่ละแถวต้องมีค่า W ในบัตเตอร์ฟลายแรกเหมือนกัน แต่ในบัตเตอร์ฟลายที่สองจะมีค่าของ W แต่ละตัวอาจจะเหมือนเดิมหรือไม่ก็ได้ และโครงสร้างของบัตเตอร์ฟลายที่มีค่า W เหมือนกันจะใช้การวนลูป (Loop) ดังโปรแกรมที่เพิ่มดังต่อไปนี้

```
for (i = j ; i<n; i= I + 2*lc)
        {
                xi = x + i
                xip = xi + lc
                temp.real = xi->real + xip->real
                temp.imag = xi->imag + xip->imag
                tm.real   = xi->real - xip->real
                tm.imag   = xi->imag - xip->imag
                xip->real = tm.real*u.real - tm.imag*u.imag
                xip->imag = tm.real*u.imag + tm.imag*u.real
                *xi = temp
        }
```

ในลูปที่กล่าวถึงจะชี้ค่าเริ่มต้นที่ขาบนโดยค่า j และทำต่อไปโดยใช้ระยะห่างระหว่างขาบนและขาล่างของบัตเตอร์ฟลาย โดยสามารถตรวจสอบจากกราฟโดยมีจำนวนของ W เหมือนกัน และทำการคำนวณในแต่ละแถวจะเพิ่มโปรแกรมลูปต่อไปนี้

```
for (j=1; j<le; j++)
        {
            for (i = j ; i<n; i= l + 2*le)
                {
                    xi = x + i
                    xip = xi + le
                    temp.real = xi->real + xip->real
                    temp.imag = xi->imag + xip->imag
                    tm.real  = xi->real - xip->real
                    tm.imag  = xi->imag - xip->imag
                    xip->real = tm.real*u.real - tm.imag*u.imag
                    xip->imag = tm.real*u.imag + tm.imag*u.real
                    *xi = temp
                }
            wptr = wptr + windex
        }
```

ในการวนลูปแต่ละครั้งจะชี้ตำแหน่งขาบนโดยใช้ค่า j เพิ่มตำแหน่งการหาบัตเตอร์ฟลายในแต่ละแถว โดยลูปภายในจะสนใจทุกๆ ค่าของบัตเตอร์ฟลายโดยใช้ค่า j เป็นจุดเริ่มต้นเช่นกัน และจะเพิ่มค่า W (wptr) โดยใช้ windex ค่าของ windex จะเปลี่ยนไปแต่ละแถว

ในลำดับสุดท้ายจะเพิ่มลูปจนถึงแถวที่ m โดยเพิ่มลูปนอกเข้าไปดังนี้

```
le = n
windex = 1;
for (l=0; l<m; l++)
    {
        le = le/2
        wptr = w
        for( i=0; i<n; i=l +  2*le)
            {
                xi  = x+i
                xip = xi+le
                temp.real = xi->real + xip->real
                temp.imag = xi->imag + xip->imag
```

```
                xip->real = xi->real - xip->real
                xip->imag = xi->imag - xip->imag
                    *xi = temp
                }
            wptr = wptr + windex
        }
    winddex = 2 *  dindex
    }
```

ค่า w จะชี้จุดเริ่มต้นของอาร์เรย์ W ก่อนจะผ่านไปแต่ละแถว

ค่า le เป็นระยะทางระหว่างขาบนและขาล่างของบัตเตอร์ฟลาย

ค่า *wptr เป็นค่าเริ่มต้นของเฟคเตอร์ W โดยจะเพิ่มไปเรื่อยๆ จนจบแถวแรกโดยใช้ค่า windex = 1 และเมื่อคำนวณบัตเตอร์ฟลายในแถวที่สองค่า windex ก็จะเพิ่มเป็น 2 เท่าและถ้าเพิ่มการคำนวณบัตเตอร์ฟลายในแถวต่อไปค่า windex ก็จะเพิ่มจากเดิมเป็น 2 เท่า จะเป็นเช่นนี้ไปเรื่อยๆ จนจบการคำนวณ จากโปรแกรมข้างต้นสามารถนำไปแก้ไขให้เป็นโปรแกรมฟาสต์ฟูเรียร์ทรานส์ฟอร์มที่สมบูรณ์และมีการแสดงผลการคำนวณที่ได้ไปพล็อตที่คอมพิวเตอร์

บทที่ 4

การทดลองและผลการทดลอง

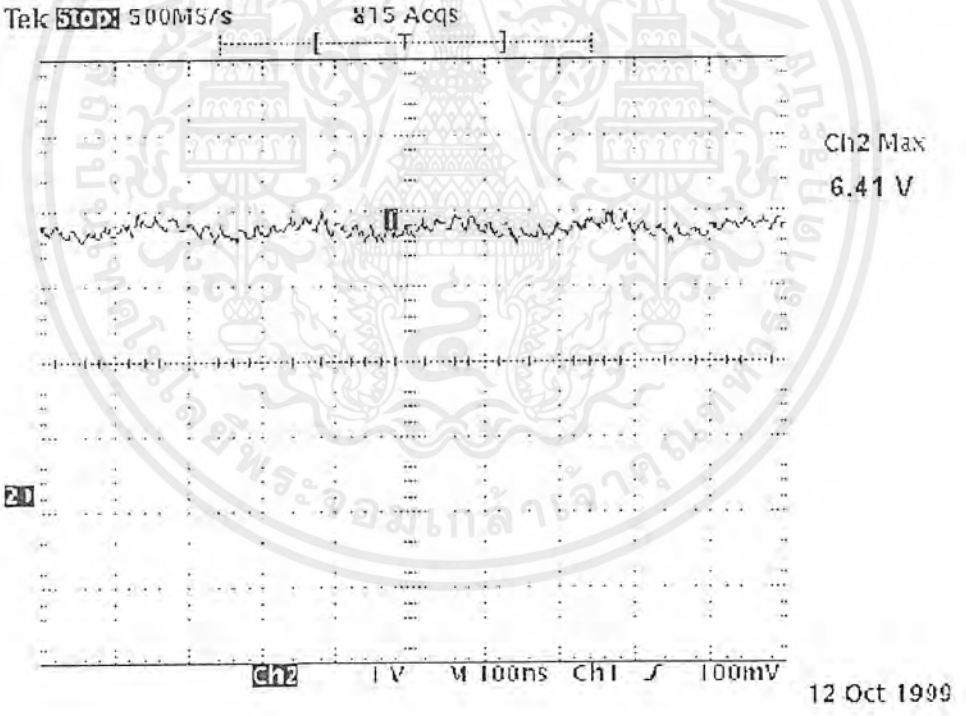
เราจะทำการทดลองออกเป็น 2 ลักษณะคือทดลองวงจรโดยการทดสอบสั่งงานทางโลจิก(Logic) และ การทดลองวงจรโดยการทดสอบสั่งงานด้วยคอมพิวเตอร์

4.1 การทดลองวงจรโดยการทดสอบสั่งงานทางโลจิก

4.1.1 การทดลองวงจรเอทูดี

วงจรอ้างอิงแรงดัน

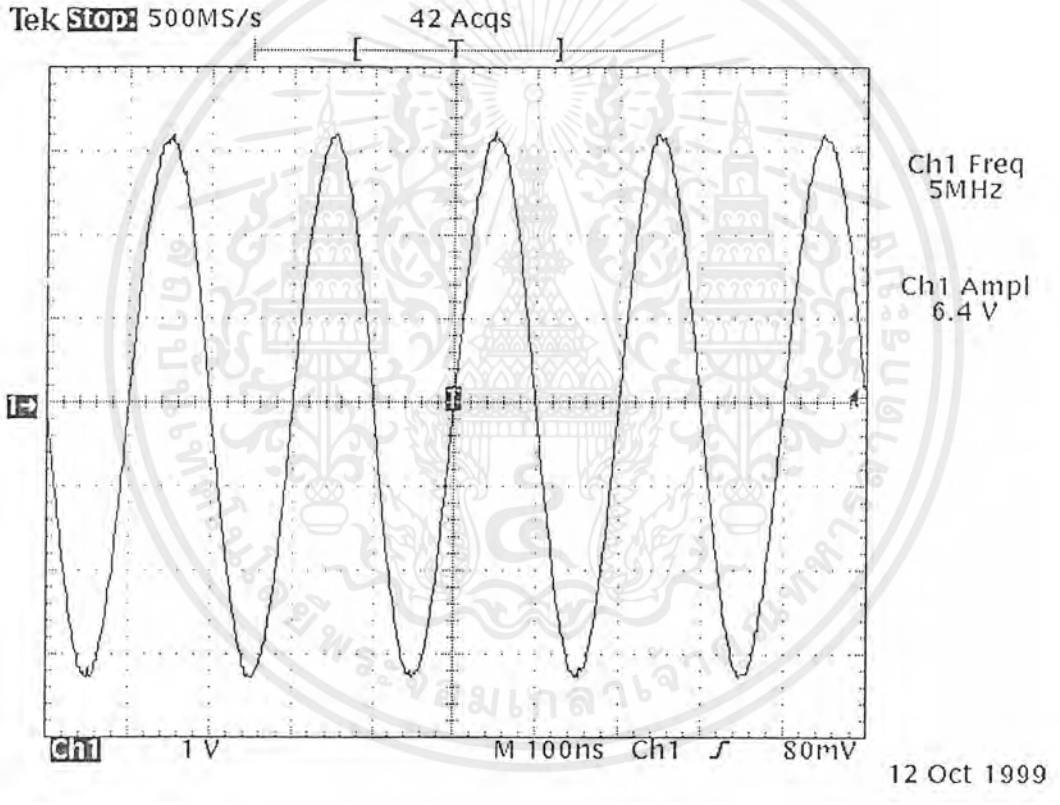
1. ต่อวงจรลงบน แผงทดลอง
2. ป้อนแรงดัน +12 โวลท์
3. ให้ทำการปรับค่า R4 จนกระทั่งวัดที่ขาอีมิเตอร์ของ Q1 ได้ประมาณ 6.4 โวลท์
4. ทำการใช้สโคปวัดที่ขาอีมิเตอร์ของ Q1 จะเกิดสัญญาณรบกวนเล็กน้อย



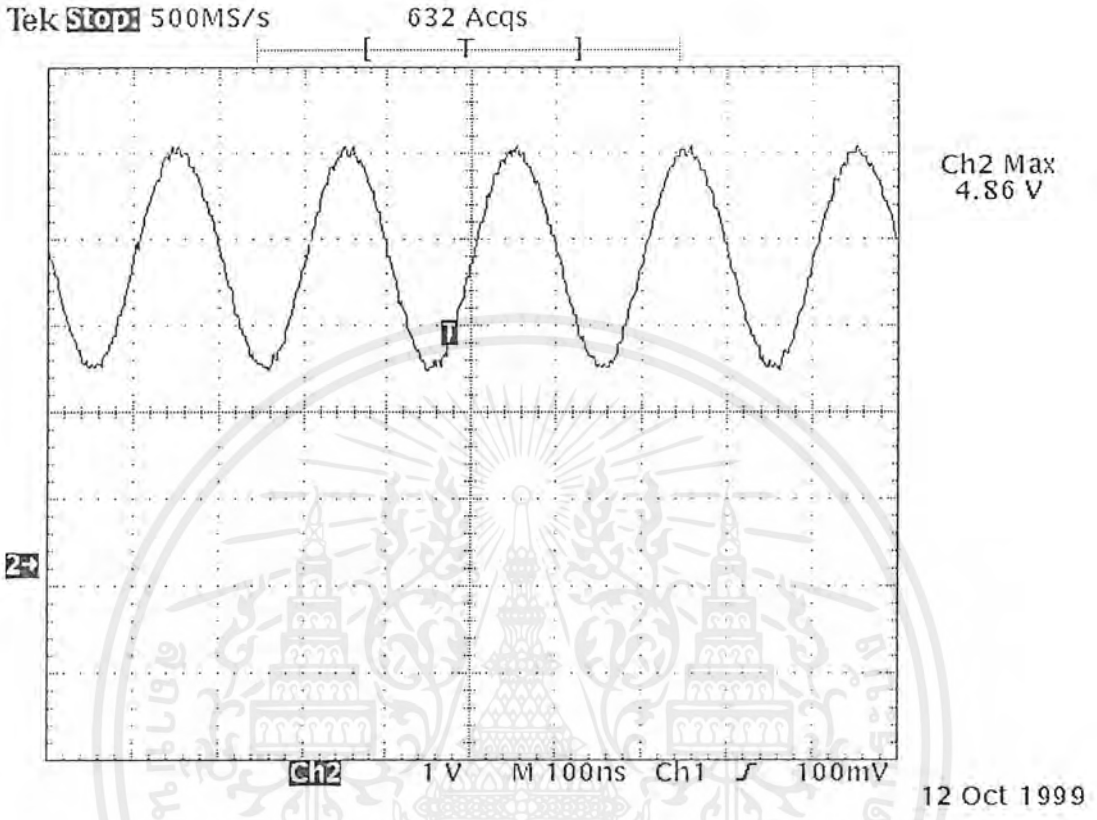
รูปที่ 4.1 สัญญาณที่ขา อีมิเตอร์ของ Q1

ตัวป้องกันเอทูดี

1. ต่อวงจรลงบนแผงทดลอง
2. ป้อนแหล่งจ่ายไฟ +12 โวลท์และ -12 โวลท์
3. ป้อนสัญญาณไซน์เวฟ ที่ขา R1 ประมาณ 6.4 โวลท์ ที่ความถี่ 5 *MHz*
4. จะได้สัญญาณที่ขา 6 ของ U1 โดยที่สัญญาณที่ได้จะมีค่าขนาดลดลงครึ่งหนึ่งของสัญญาณอินพุทและสัญญาณถูกยกระดับขึ้น 3.2 โวลท์
5. จากการคำนวณค่าอัตราสลูลองจ่ายสัญญาณความถี่มากกว่าที่กำหนดประมาณ 7.5 *MHz* สังเกตสัญญาณแล้วไม่เกิดความเพี้ยนจากสัญญาณเดิม แต่ลองจ่ายสัญญาณที่ความถี่ 10 *MHz* สังเกตดูสัญญาณเกิดความเพี้ยนจากสัญญาณเดิม



รูปที่ 4.2 สัญญาณอินพุท

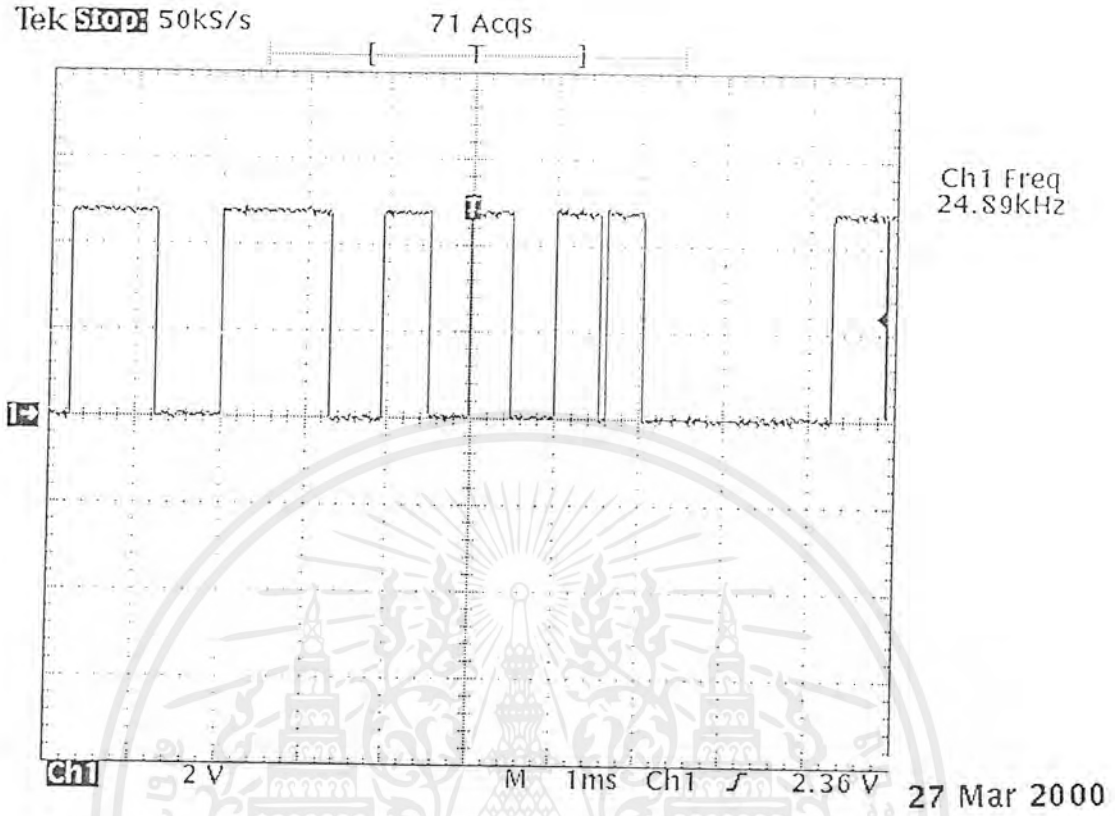


รูปที่ 4.3 สัญญาณเอาท์พุทของตัวป้องกันเอทูดี

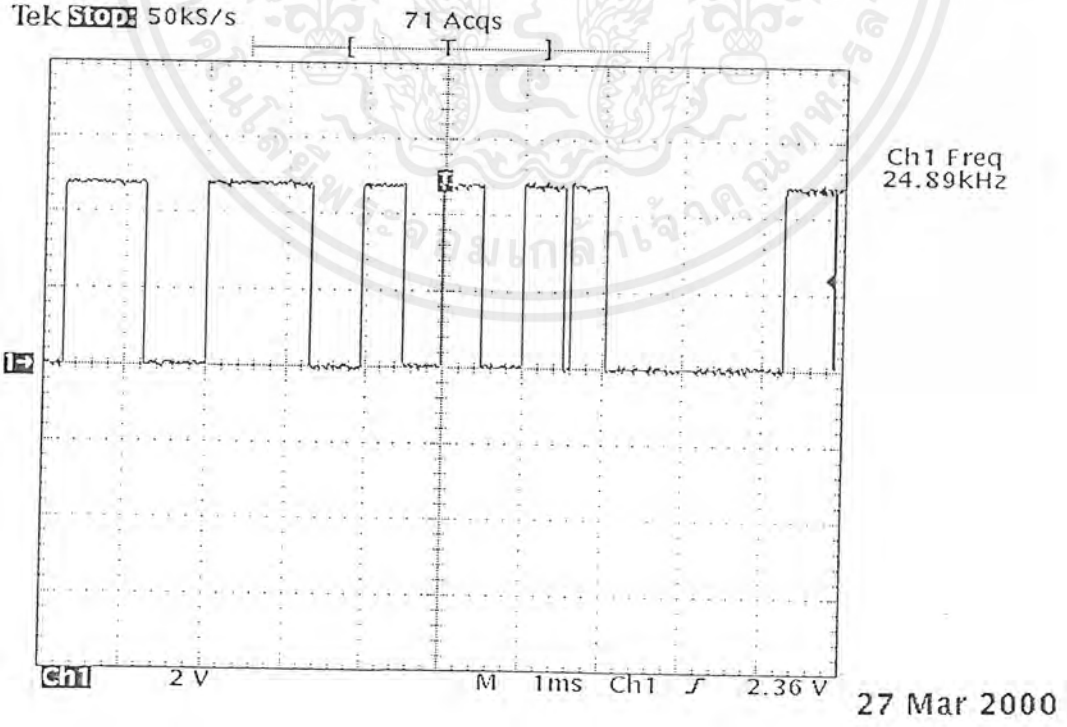
วงจรเอทูดี

1. ต่อวงจรลงบนแผงทดลองโดยทำการต่อวงจรทั้งหมดของวงจรเอทูดี
2. ป้อนแล่งจ่ายไฟ +5 โวลท์
3. ทำการจ่าย CLOCK ให้กับวงจรที่ขา18ของ U3
4. ใช้สโคปทำการทดสอบดูว่าที่ขา 1-8 ของ U3 มีสัญญาณเกิดการเปลี่ยนแปลงหรือไม่
5. ถ้ามีการเปลี่ยนแปลงแสดงว่าวงจรเอทูดีทำงาน มีการเปลี่ยนแปลงทั้ง 8 ขา

สัญญาณที่วัดได้จาก สัญญาณบิตที่ 1 และ สัญญาณบิตที่ 8 ของเอทูดี



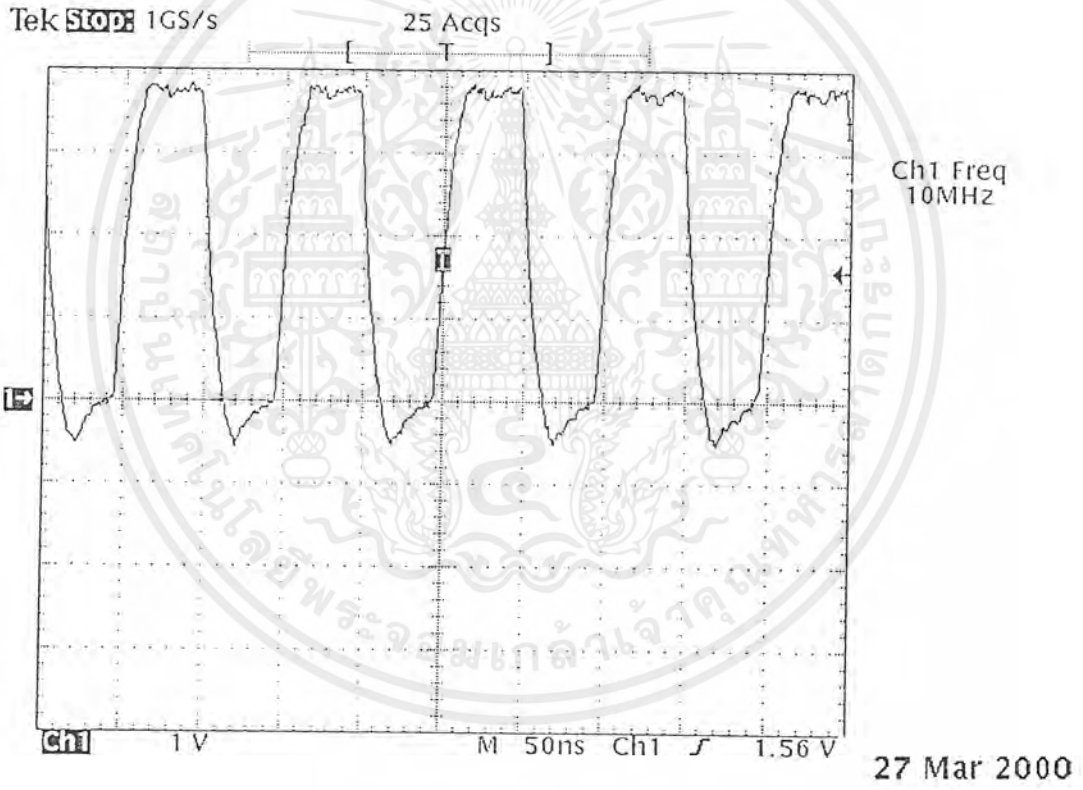
รูปที่ 4.4 สัญญาณบิตที่ 1



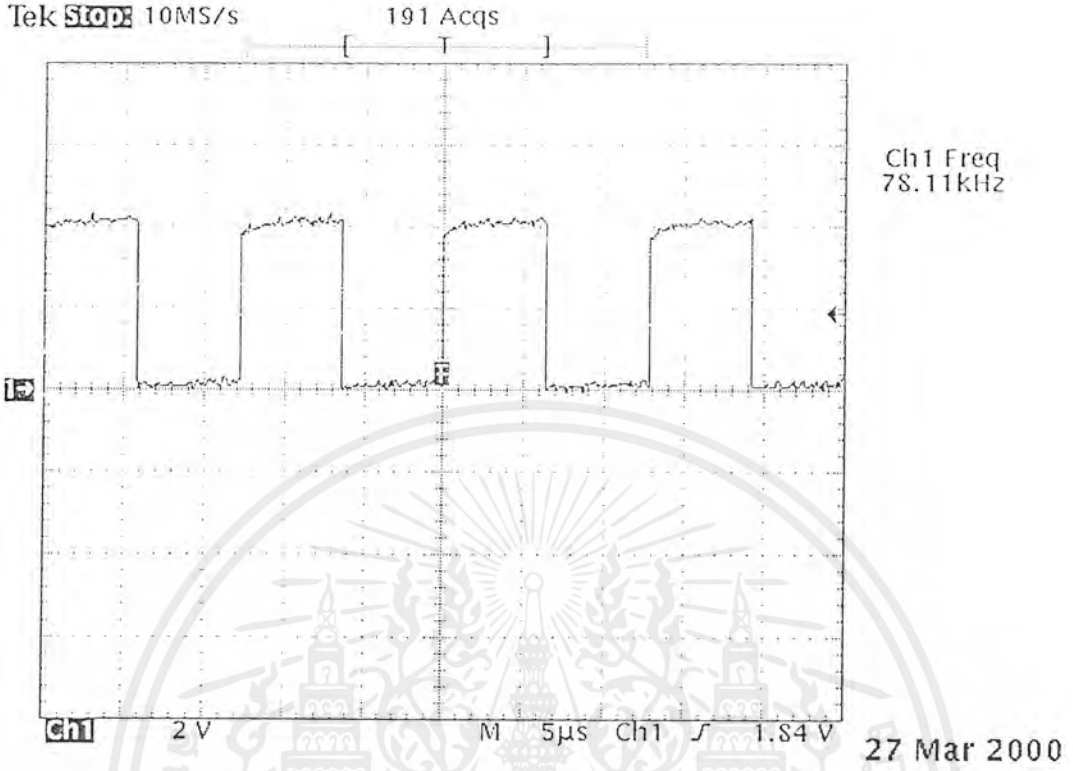
รูปที่ 4.5 สัญญาณบิตที่ 8

4.1.2 การทดลองวงจรผลิตสัญญาณนาฬิกา

1. ต่อวงจรลงบนแผงทดลอง
2. ป้อนแหล่งจ่ายไฟ +5 โวลท์
3. จับสัญญาณเอาท์พุทที่ขา U4 และ U5 จะได้ความถี่สัญญาณนาฬิกาตามที่ต้องการ 8 ความถี่
4. ทำการทริก D0, D1, D2 เป็น "0" และ ที่ขา 11 เป็น "1" ของ U6 พร้อมกัน
5. จะได้สัญญาณที่ขา 3 ของ U9 เป็นClock เท่ากับ 10 MHz
6. ทำการทริก D0, D1, D2 เป็น "0" และ ที่ขา 11 เป็น "1" ของ U6 พร้อมกัน
7. จะได้สัญญาณที่ขา 3 ของ U9 เป็น CLOCK เท่ากับ 78.125 *kHz*



รูปที่ 4.6 สัญญาณจาก CLOCK ความถี่ 10 *MHz*



รูปที่ 4.7 สัญญาณจาก CLOCK ความถี่ 78.12 *KHz*

จากการทดลองวัดสัญญาณทั้ง 8 ได้ค่าสัญญาณประมาณดังนี้

ค่าที่ต้องกำหนดที่ขาต่างๆของ U6(74HC374)				เอาท์พุท
D2	D1	D0	ขา11ทำการทริก	
0	0	0	1	10 *MHz*
0	0	1	1	5.001 *MHz*
0	1	0	1	2.5 *MHz*
0	1	1	1	1.25 *MHz*
1	0	0	1	625.01 *kHz*
1	0	1	1	312.5 *kHz*
1	1	0	1	156.23 *kHz*
1	1	1	1	78.11 *kHz*

ตารางที่ 4.1 แสดงความถี่ที่ได้จากวงจรผลิตความถี่

4.1.3 การทดลองวงจรนับ

1. ต่อวงจรลงบนแผงทดลอง
2. ป้อนแหล่งจ่ายไฟ +5 โวลท์
3. ต่อ LED ที่ขา 6 และขา 8 ของ U17 เพื่อแสดงสถานะการWRITE และการ READ ของ MEMORY
4. ทำการป้อนสัญญาณ พัลซ์ 1 ลูกที่ขา CLR_COUNTER
5. ป้อนสัญญาณ พัลซ์ 1 ลูกที่ขา 3 (START_ADC) ของ U17
6. สังเกต LED ที่ขา 6 ของ U17 จะติด แสดงสถานะการเขียนข้อมูลลงบนหน่วยความจำ
7. ป้อนสัญญาณเป็น "1" ที่ขา 10(WAIT_ADC_READY)ของ U18 เพื่อรอการนับครบ FFFFH
8. จ่าย CLOCK ให้ที่ขา 2 ของ U18 จะได้การนับที่เอาท์พุทเริ่มจาก 0000H-FFFFH
9. สังเกตเมื่อนับครบแล้ว จะหยุดนับและ LED ที่ขา 8 ของ U18 จะติดขึ้นบอกว่าการนับถึงตำแหน่งที่ FFFFH แล้ว
10. ทำการป้อนสัญญาณ พัลซ์ 1 ลูกที่ขา CLR_COUNTER เพื่อทำการเคลียร์ เกทต่างๆและ LED ทุกตัว จะดับเพื่อพร้อมเริ่มการอ่านข้อมูล
11. ป้อนสัญญาณพัลซ์ 1 ลูกที่ขา 11 (START_RD_DATA)ของ U17 เพื่อทำการเริ่มอ่านข้อมูลหรือทำการนับแอดเดรส
12. สังเกต LED ที่ขา 8 ของ U17 จะติด แสดงสถานะการอ่านข้อมูลจากบนหน่วยความจำ
13. ทำการเริ่มนับโดยการป้อนพัลซ์ที่ละลูกที่ขา 5(READ_DATA)ของ U18
14. แอดเดรสจะเริ่มทำการนับขึ้น แต่การทดลองจะทำการป้อนขบวนพัลซ์แทนเพื่อความรวดเร็วการสังเกตการนับของแอดเดรส
15. สังเกตเมื่อนับครบแล้ว จะหยุดนับและ LED ที่ขา 8 ของ U18 จะติดขึ้นบอกว่าการนับถึงตำแหน่งที่ FFFFH แล้ว

4.1.4 ทดลองวงจรถอดรหัส

1. ต่อวงจรลงบนแผงทดลอง
2. ทำการป้อนแรงดันไฟ +5 โวลท์
3. ป้อนค่าต่างๆดังตารางเพื่อทดลองการถอดรหัสของ U13 และ U14
4. โดยการป้อนแรงดันที่ขาต่างๆของ U14 ดังนี้และสังเกต LED ที่ขา 14 และขา 15 ได้ดังตาราง 4.2

ADDRESS											output(U14)	
IOR	AEN	10	9	8	7	6	5	4	3	8,U12	Y0	Y1
0	1	0	1	1	0	0	0	0	0	0	0	1
0	1	0	1	1	0	0	0	1	0	0	1	0

ตารางที่ 4.2 แสดงสัญญาณที่ขาเอาต์พุทของ U14

5. ทำการป้อนแรงดันที่ขาต่างๆของ U13 ดังนี้และสังเกต LED ที่ขา 15, 14, 13, 12, 11 ได้ดังนี้

ป้อน IOR = 0, AEN = 1, A10, A9 = 1, A8 = 1, A7 = 0, A6 = 0, A 5 = 0, A4 = 0,A3 = 0

ADDRESS													(U13)
IOR	AEN	10	9	8	7	6	5	4	3	2	1	0	'ON'
0	1	0	1	1	0	0	0	0	0	0	0	0	Y0
0	1	0	1	1	0	0	0	0	0	0	0	1	Y1
0	1	0	1	1	0	0	0	0	0	0	1	0	Y2
0	1	0	1	1	0	0	0	0	0	0	1	1	Y3
0	1	0	1	1	0	0	0	0	0	1	0	0	Y4

ตารางที่ 4.3 แสดงเอาท์พุทของ U13

4.2 การทดลองวงจรโดยการทดสอบสั่งงานด้วยคอมพิวเตอร์

ใส่อุปกรณ์ทั้งหมดลงบนแผ่นวงจรเพื่อทดสอบดูว่าวงจรที่ทำขึ้นนั้นสามารถใช้ติดต่อคอมพิวเตอร์ได้หรือไม่ โดยจะใช้การเขียนโปรแกรมช่วยในการทดลอง โดยการเขียนโปรแกรมควบคุมการทำงานทั้งหมดของวงจรและจะใช้คอมพิวเตอร์ควบคุมการทำงานต่างๆ ของการ์ดที่ทำขึ้นมาโดยมีรายละเอียดดังนี้

4.2.1 การทดลองติดต่อกับคอมพิวเตอร์

การทำงานของโปรแกรมจะมีการเลือกอยู่ 2 โหมด(mode) คือดีบั๊กโหมด (Debug mode) และโหมดปกติ (Normal mode) ถ้าเลือกดีบั๊กโหมดจะเป็นการทำงานที่ละขั้นตอนแล้วจะกดคีย์เพื่อเลือกว่าจะทดสอบส่วนใดเพื่อจะได้วิเคราะห์ดูว่าที่ออกแบบมาสามารถติดต่อกับคอมพิวเตอร์ได้หรือเปล่า ส่วนโหมดปกติจะทำการทดสอบเพื่อดูการติดต่อกับคอมพิวเตอร์โดยทำงานเป็นขั้นตอนจนจบ

ผลการทดลองโปรแกรมโดยใช้ดีบั๊กโหมด

เมื่อทำการต่อการ์ดเข้ากับสล็อตคอมพิวเตอร์ แล้วทดลองป้อนอินพุทเป็นสัญญาณซายน์เวฟ ที่ประมาณ 50 กิโลเฮิร์ต ขนาดแอมปลิจูดประมาณ 3 โวลท์ แล้วทำการรันโปรแกรม (run program)โดยพิมพ์ spectrum/d จะมีการเขียนข้อมูลจากเอทูดีลงหน่วยความจำและอ่านข้อมูลจากหน่วยความจำไปหาคอมพิวเตอร์แล้วนำค่าทำได้แสดงผลที่จอคอมพิวเตอร์ดังนี้

Spectrum Analyzer

[DebugMode]

1	- Clear Counter	F1	- Display data buffer
2	- Set Clock	F2	- Display FFT
3	- Start ADC	F9	- Read data from file
4	- Wait ADC to be ready	F10	- Create data
5	- Set Read data		
6	- Read ADC data		
?	- Help Menu	ESC	- Exit

โดยต้องเลือกกดคีย์ ต่างๆ เพื่อดีบั๊กวงจรทีละส่วนตามการกดหมายเลขคีย์ โดยจะใช้โจ๊กโพลป (Logic Probe) จับสัญญาณต่างๆ ที่การ์ดตามการกดคีย์ต่างๆ โดยอาจกดคีย์ดังต่อไปนี้

1.) กดคีย์ "1" เพื่อเคลียร์ตำแหน่งการนับแอดเดรสและเคลียร์ฟลิบ – ฟลอบของการ์ด
2.) กดคีย์ "2" เพื่อเลือกความถี่สัญญาณนาฬิกาให้กับวงจรนับและวงจรเอทูดี
3.) กดคีย์ "3" เพื่อเลือกการเขียนข้อมูลจากเอทูดีเพื่อไปเก็บในหน่วยความจำ
4.) กดคีย์ "4" เพื่อเช็คว่ามีการเขียนข้อมูลจากเอทูดีไปเก็บในหน่วยความจำเต็ม 64 กิโลไบท์ หรือยังถ้าเต็มก็จะหลุดมาที่โปรแกรมหลัก
5.) กดคีย์ "1" เพื่อเคลียร์ตำแหน่งการนับแอดเดรสและเคลียร์ฟลิบ – ฟลอบของการ์ด
6.) กดคีย์ "5" เพื่อเลือกการอ่านข้อมูลจากหน่วยความจำ
7.) กดคีย์ "6" เพื่ออ่านข้อมูลจากหน่วยความจำของการ์ดเพื่อไปเก็บไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์ และเป็นสัญญาณนาฬิกาให้กับวงจรนับ
8.) กดคีย์ "F1" เพื่อแสดงผลข้อมูลที่แปลงได้จากเอทูดี
9.) กดคีย์ "F2" เพื่อแสดงผลของการคำนวณที่ได้จากการทำฟาสต์ฟูเรียร์ทรานส์ฟอร์มทั้งค่าจริงและจินตภาพ
10.) กดคีย์ "F9" เพื่ออ่านข้อมูลจากไฟล์ Input.dat โดยการป้อนข้อมูลจากภายนอกเมื่อรู้ค่าการแซมปลิ้ง(sampling)ที่จุดต่างๆ เป็นการทดสอบโดยไม่ต้องติดต่อกับการ์ด
11.) กดคีย์ "F10" เพื่อสร้างฐานข้อมูลก่อนจะนำไปเข้าคำนวณ
12.) กดคีย์ "?"กลับไปยังเมนูหลักใหม่
13.) กดคีย์ "ESC" เพื่อออกจากโปรแกรมดีบั๊ก

เมื่อกดคีย์ "F1" แสดงผลของข้อมูลที่แปลงได้จากเอทูดีที่จอคอมพิวเตอร์จะมีค่าต่างๆ ณ ตำแหน่งแอดเดรสต่างๆดังนี้ (จะนำมาแสดงประมาณ 1 หน้า) และจะนำข้อมูลเหล่านี้ไปคำนวณหาฟาสต์ฟูเรียร์ทรานส์ฟอร์มต่อไป

Address 0280: BB 7B BB FB BB FB BB 7A-BB 7B BB FA BB FB BB 5A

Address 0290: BB 7B BB FB BB FB BB 7A-BB 7B BB 7A BB FB BB 5A

Address 02A0: BB FB BB FB BB FB BB 7A-BB 7B BB 7A FB FB FC 5A

Address 02B0: BB 7B BC FA BB 7B BB 5A-BB 7B BB 7A BB FB BB 5A

Address 02C0: BB 79 BB FA BB 7B BB 7A-BB 7B BB FA BB FB BB 5A

Address 02D0: BB 7B BB FB BB FB BB 5A-BB 79 BB 7A BB FB BB 5A

Address 02E0: BB 7B BB FB BB 7B BB 7A-BB 79 BB 7A BB FB BB 5A

Address 02F0: BB 79 BB FB BB 7B BB 7A-BB 79 BB 7A BB FB BC 5A

Address 0300: BC FD BB FB BB 7B BB 7A-BC FD BB 7A BB FB BB 5A

Address 0310: BB 79 BC FA BC FE BC FA-BC FD BB 7A FE FE BB 5A

Address 0320: BB 79 BB FB BC FE BC FA-BC FD BD FA BC FE BC 5A

Address 0330: BC FD BC FA BE FE BC FA-BC FD BB 79 BD FE BC 5A

Address 0340: BC FD BB FB BC FE BB 7A-BE FE BC FA BE FE BC 59

Address 0350: BC FD BC F9 BD FE BC F9-BC FD BC FA BE FE BC 59

Address 0360: BC FD BC FE BC FE BC F9-BC FD BC FA BE FE BC 59

Address 0370: BC FD BC F9 BD FE BC F9-BC FD BC F9 BD FE BC 59

Address 0380: BC FD BC F9 BD FE BC F9-BC FD BC F9 BD FE BC 59

Address 0390: BC FD BC F9 BD FD BC F9-BC FD BC F9 BD FE BC 59

Address 03A0: BC FD BC F9 BD FE BC F9-BC FD BC F9 BD FE BC 59

Address 03B0: BC FD BC F9 BD FD BC F9-BC FD BC F9 BD FE BC 59

Address 03C0: BC FD BD F9 BD FD BD F9-BC FD BD F9 BD FE BC 59

Address 03D0: BC FD BD F9 BD FD BD F9-BC FD BD FA BC FD BD 59

Address 03E0: BC FD BD F9 BD FD BD F9-BC FD BD F9 BD FD BD 59

Address 03F0: BD FD BD FD BD FD BD F9-BD FD BD F9 BD FD BD 59

4.2.2 การทดลองโปรแกรมโดยใช้โหมดปกติ

1.) เสียบการ์ดเข้าที่สล็อตคอมพิวเตอร์

2.) ต่อสัญญาณที่จะวัดเข้าที่คอนเน็คเตอร์ DB9

3.) พิมพ์ "Spectrum"แล้วกด "Enter"

4.) โชว์ เมนูหลัก

4.1 กด "**Esc**" เพื่อออกจากโปรแกรม

4.2 กด "**F2**" เพื่อทำการขยายย่านวัดให้กว้างขึ้น

4.3 กด "**F3**" เพื่อทำการลดย่านวัดให้แคบลง

4.4 กด "**F4**" เป็นการอ่านข้อมูลจากไฟล์ Input.dat โดยการป้อนค่าเพื่อการทดสอบ

5.) ถ้าผลการทดลองไม่สามารถวัดได้ให้กด "Esc"เพื่อกลับสู่เมนูหลัก

6.) เพื่อให้ได้สเปคตรัมที่ละเอียดขึ้นให้ทำการเปลี่ยนขยายย่านการวัดโดยกด "F2"

7.) ถ้าต้องการลดย่านการวัดกด "F3" เพื่อคำนวณและแสดงผลที่จอคอมพิวเตอร์

8.) เลื่อนตำแหน่ง "Mouse" ไปตำแหน่งที่ต้องการวัดค่าความถี่

9.) จะแสดงค่าความถี่และค่าแอมปลิจูดที่จอคอมพิวเตอร์

10.) ถ้าจะทำการวัดสัญญาณใหม่ให้ออกไปที่เมนูหลัก โดยกด "Esc" เพื่อกลับเมนูหลัก

11.) ทำตามข้อ 6-9

12.) กด "Esc"เพื่อออกจากโปรแกรม

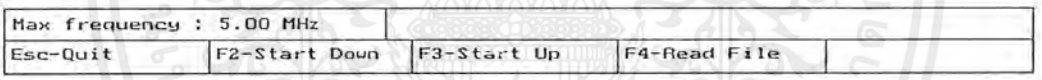
สรุปผลการทำงาน

เมื่อนำการ์ดมาต่อร่วมกับคอมพิวเตอร์สามารถแปลงข้อมูลจากสัญญาณอนาล็อกมาเป็นสัญญาณดิจิตอลโดยใช้การเขียนโปรแกรมและสามารถควบคุมการทำงานต่างๆ ตามการกดคีย์ ซึ่งจะมีผลการทดลองนำไปใช้งานจริง ทำให้สามารถวิเคราะห์สัญญาณต่างๆได้ เป็นอย่างดี

จากการทดลองวัดรูปคลื่นซายน์ เห็นได้ว่าจะปรากฏเป็นสเปคตรัมแนวดิ่ง 1 แท่ง ที่ตรงกับตำแหน่งความถี่สัญญาณที่ทำการวัดนั้นๆบนแกนราบเพียงแห่งเดียวเพราะเป็นคลื่นรูปซายน์บริสุทธิ์ เมื่อทำการวัดคลื่นรูปสี่เหลี่ยม ซึ่งคลื่นรูปสี่เหลี่ยมนั้นจะประกอบด้วยคลื่นรูปซายน์จำนวนมากหรือกล่างอีกนัยหนึ่งว่า คลื่นรูปสี่เหลี่ยมประกอบด้วยคลื่นรูปซายน์ที่เป็นคลื่นความถี่ฟันดาเมนตัล (Fundamental) จำนวน 1 คลื่น และคลื่นรูปซายน์ที่มีความถี่เท่ากับฮาร์มอนิกคี่ (odd) ซึ่งมีความถี่เป็น 1 เท่า, 3 เท่า, 5 เท่า... จนถึงจำนวนนับอนันต์ และเมื่อทำการวัดคลื่นรูปสามเหลี่ยม(Sawtooth wave) จะได้สเปคตรัมที่ประกอบไปด้วยฮาร์มอนิกลำดับคู่(even) และลำดับคี่ (odd) ซึ่งเป็นการรวมของรูปคลื่นซายน์ฟันดาเมนตัลกับฮาร์มอนิกที่ 2, 3, 4 ...

4.2.3 การทดลองนำไปวัดสัญญาณ

เมื่อรัน(Run)โปรแกรมด้วยโหมดปกติจะแสดงหน้าจอคอมพิวเตอร์ มีเมนูให้เลือกใช้งานต่างๆ ดังนี้



รูปที่ 4.8 แสดงเมนูหลักของโปรแกรม

โดยมีการแสดงค่าความถี่สูงสุดของสัญญาณที่จะทำการวัดและมีการกดคีย์ต่างๆดังนี้

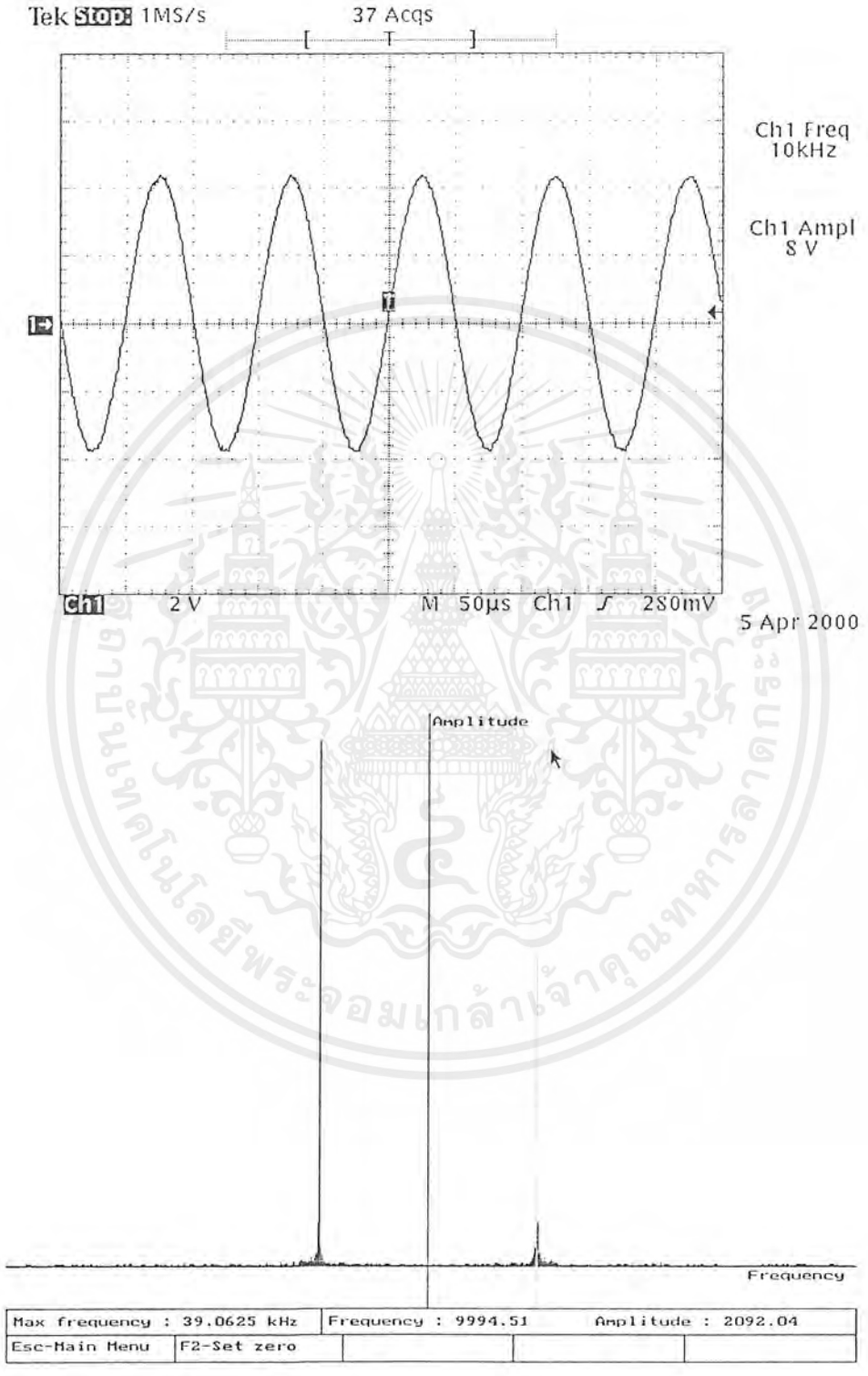
กด "F2" เพื่อเลือกการขยายขอบเขตการวัดที่กว้างขึ้น

กด "F3" เพื่อเลือกการลดขอบเขตการวัดให้แคบลง

กด "F4" เพื่อทำการคำนวณและแสดงผล โดยการอ่านจากไฟล์ Input.dat ซึ่งได้จากการป้อนค่าจากภายนอกเมื่อรู้ค่า ณ จุดต่างๆ ของสัญญาณ

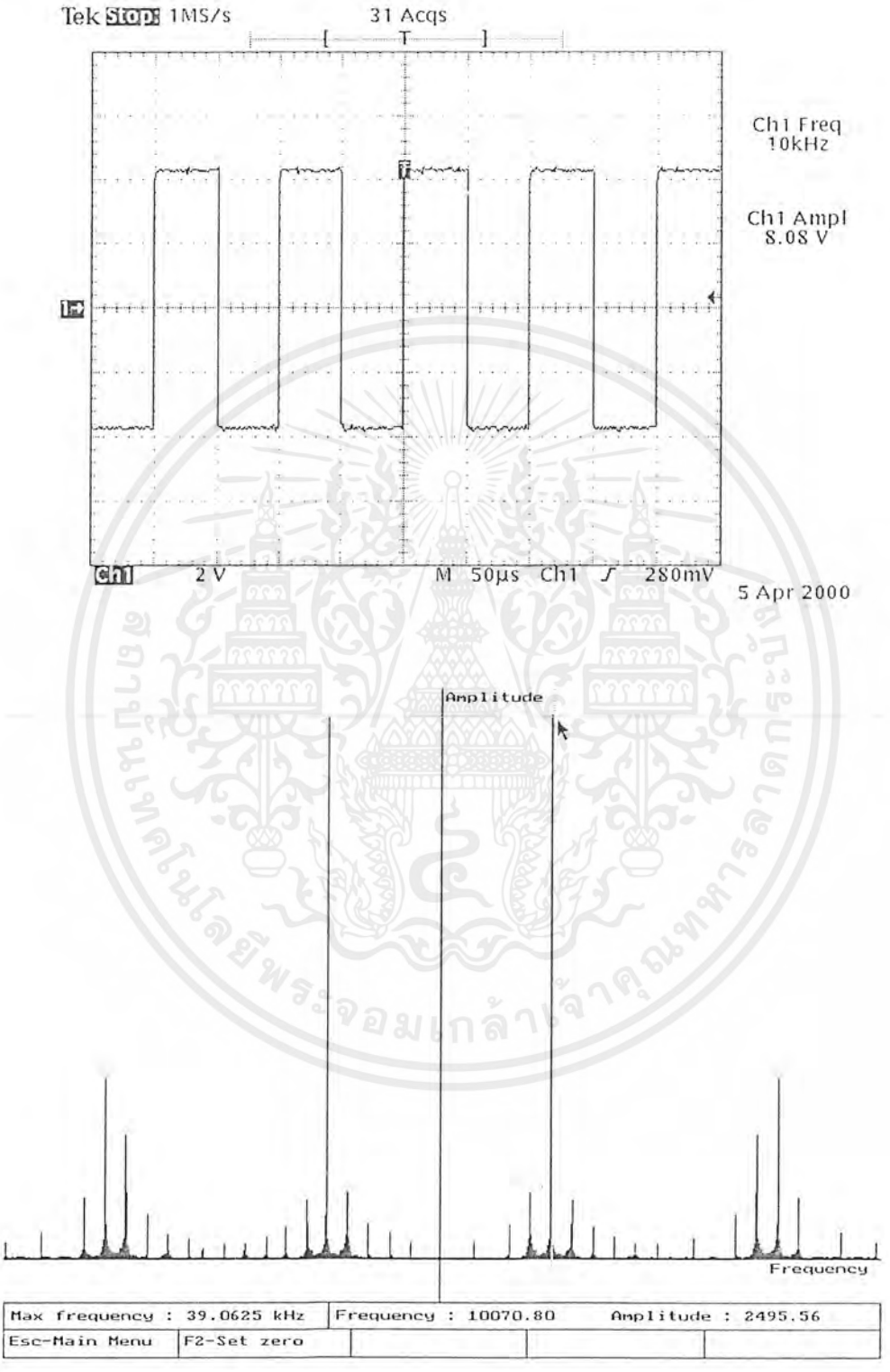
กด "Esc" เพื่อออกจากโปรแกรม

ผลการวัดสัญญาณซายน์ที่ความถี่ 10 กิโลเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



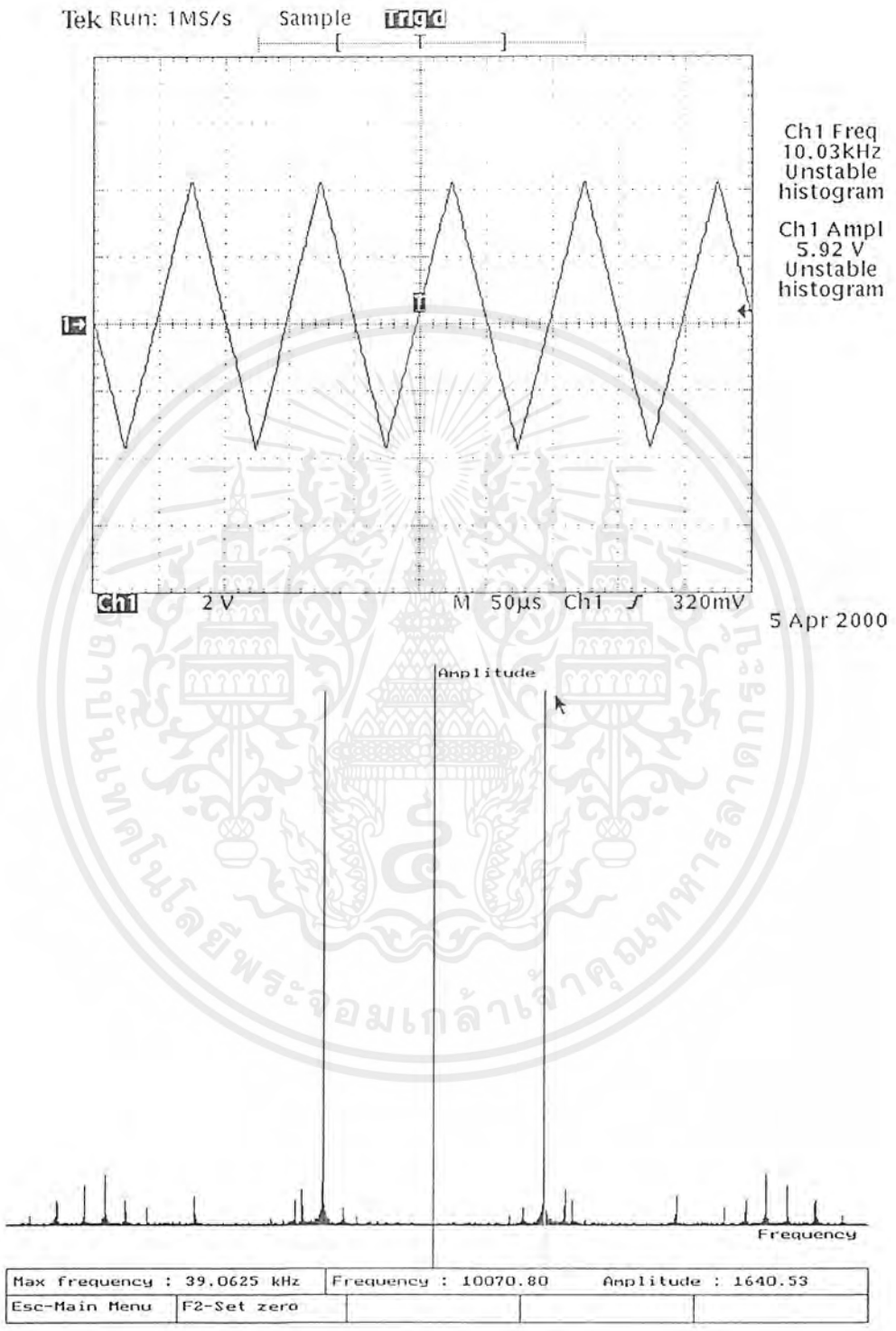
รูปที่ 4.9 แสดงผลการวัดสัญญาณซายน์ 10 กิโลเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณสี่เหลี่ยมที่ความถี่ 10 กิโลเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



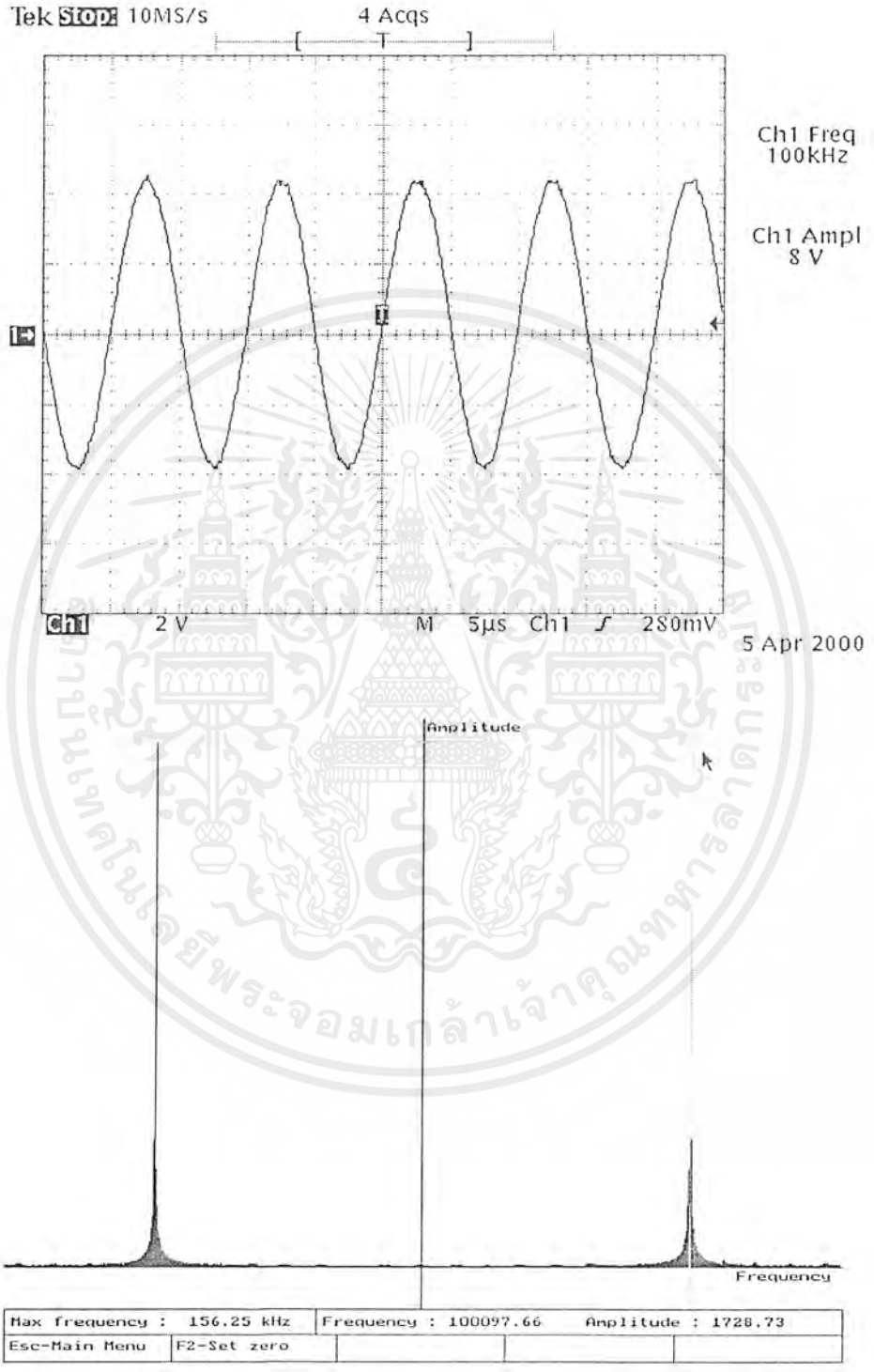
รูปที่ 4.10 แสดงผลการวัดสัญญาณสี่เหลี่ยม 10 กิโลเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณสามเหลี่ยมที่ความถี่ 10 กิโลเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



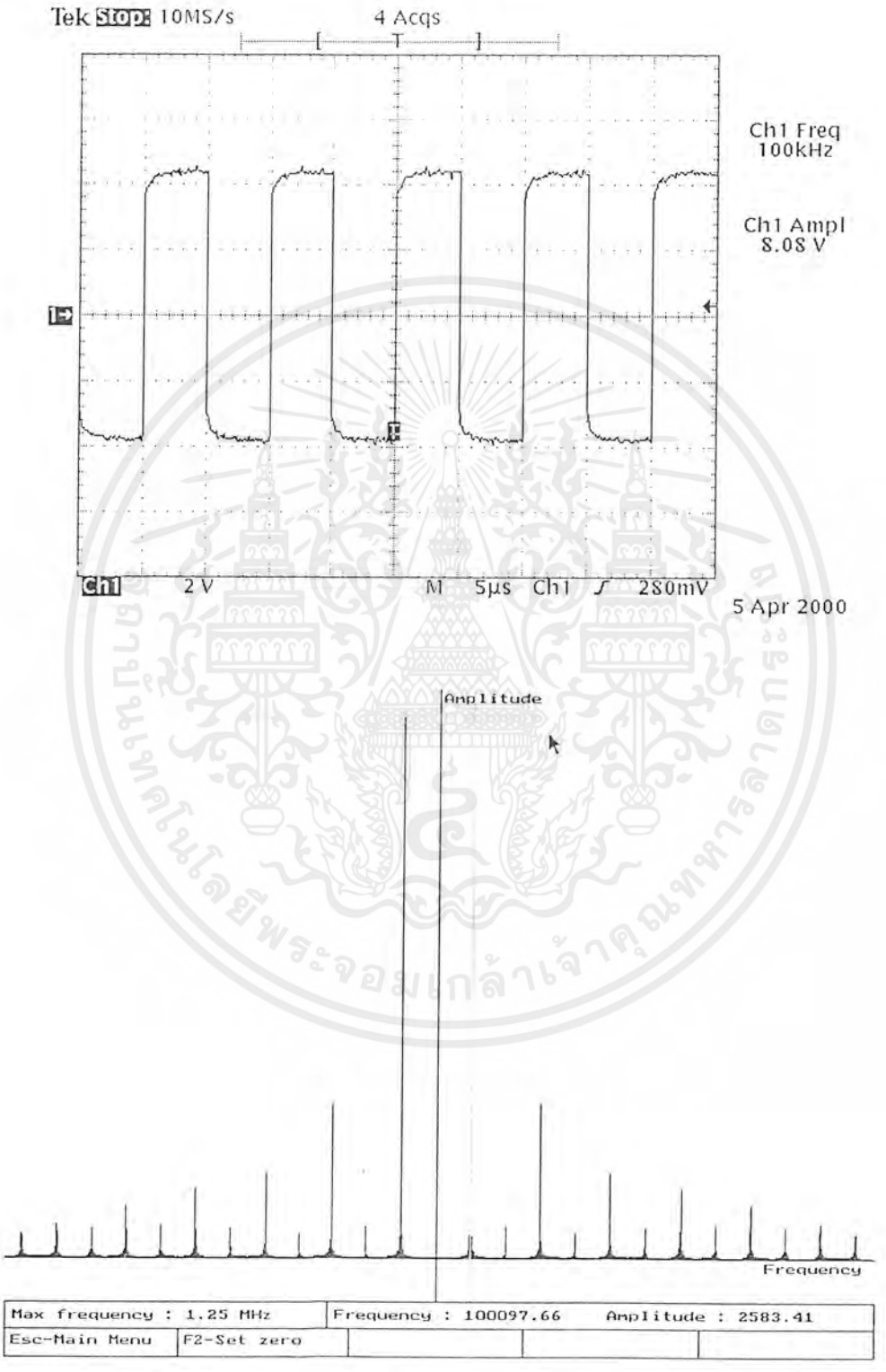
รูปที่ 4.11 แสดงผลการวัดสัญญาณสามเหลี่ยม 10 กิโลเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณซายน์ที่ความถี่ 100 กิโลเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



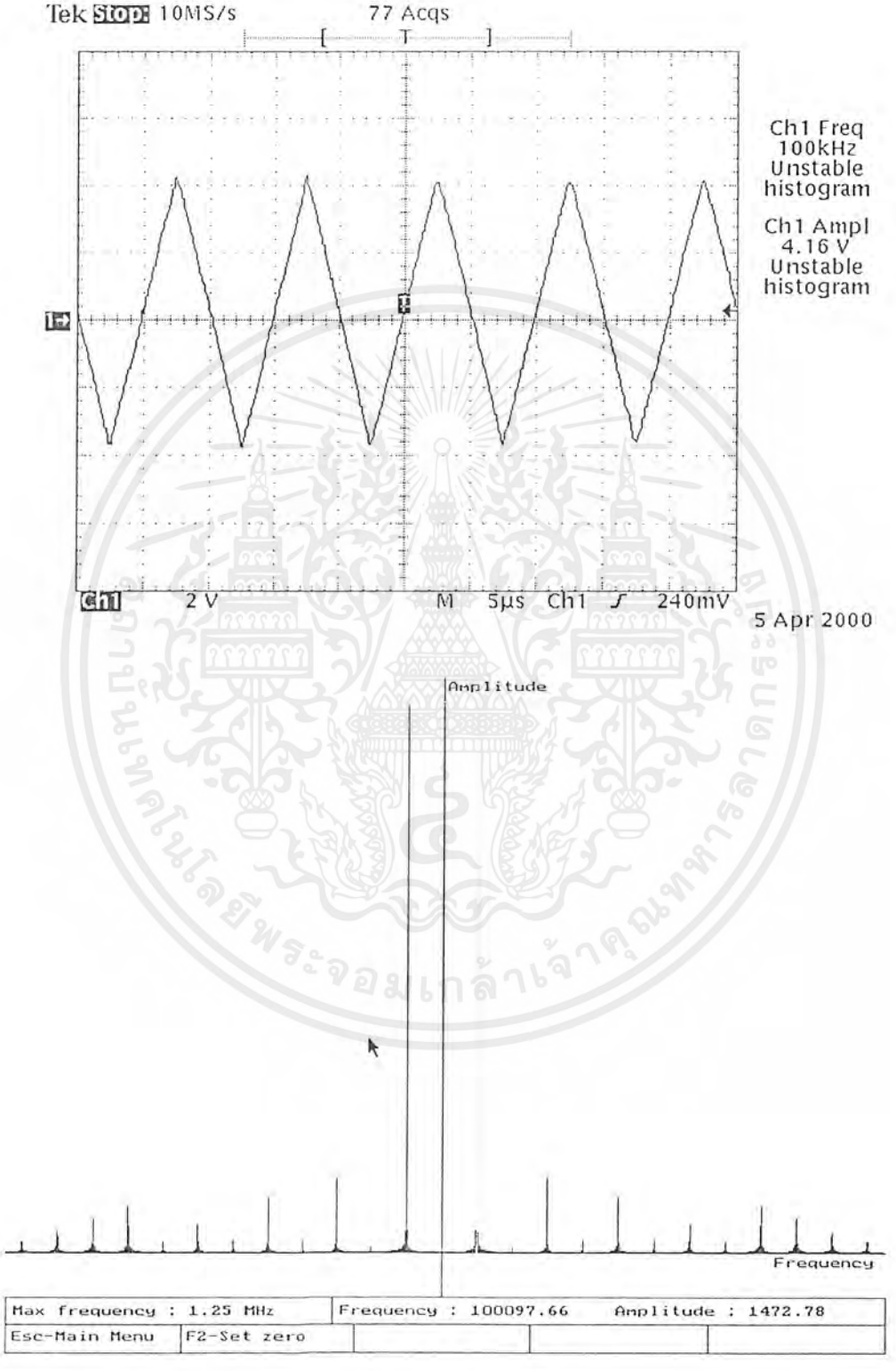
รูปที่ 4.12 แสดงผลการวัดสัญญาณซายน์ 100 กิโลเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณสี่เหลี่ยมที่ความถี่ 100 กิโลเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



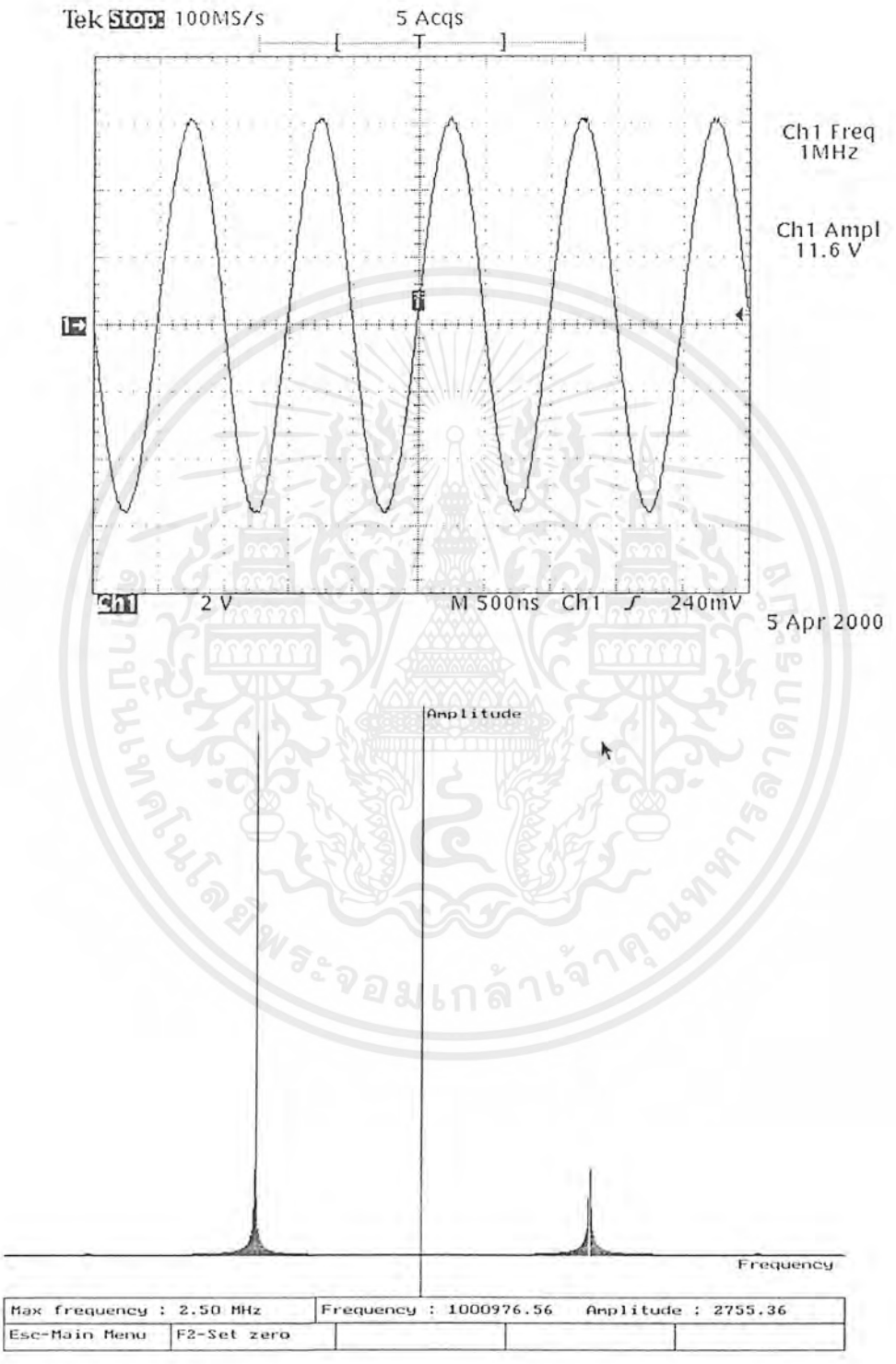
รูปที่ 4.13 แสดงผลการวัดสัญญาณสี่เหลี่ยม 100 กิโลเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณสามเหลี่ยมที่ความถี่ 100 กิโลเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



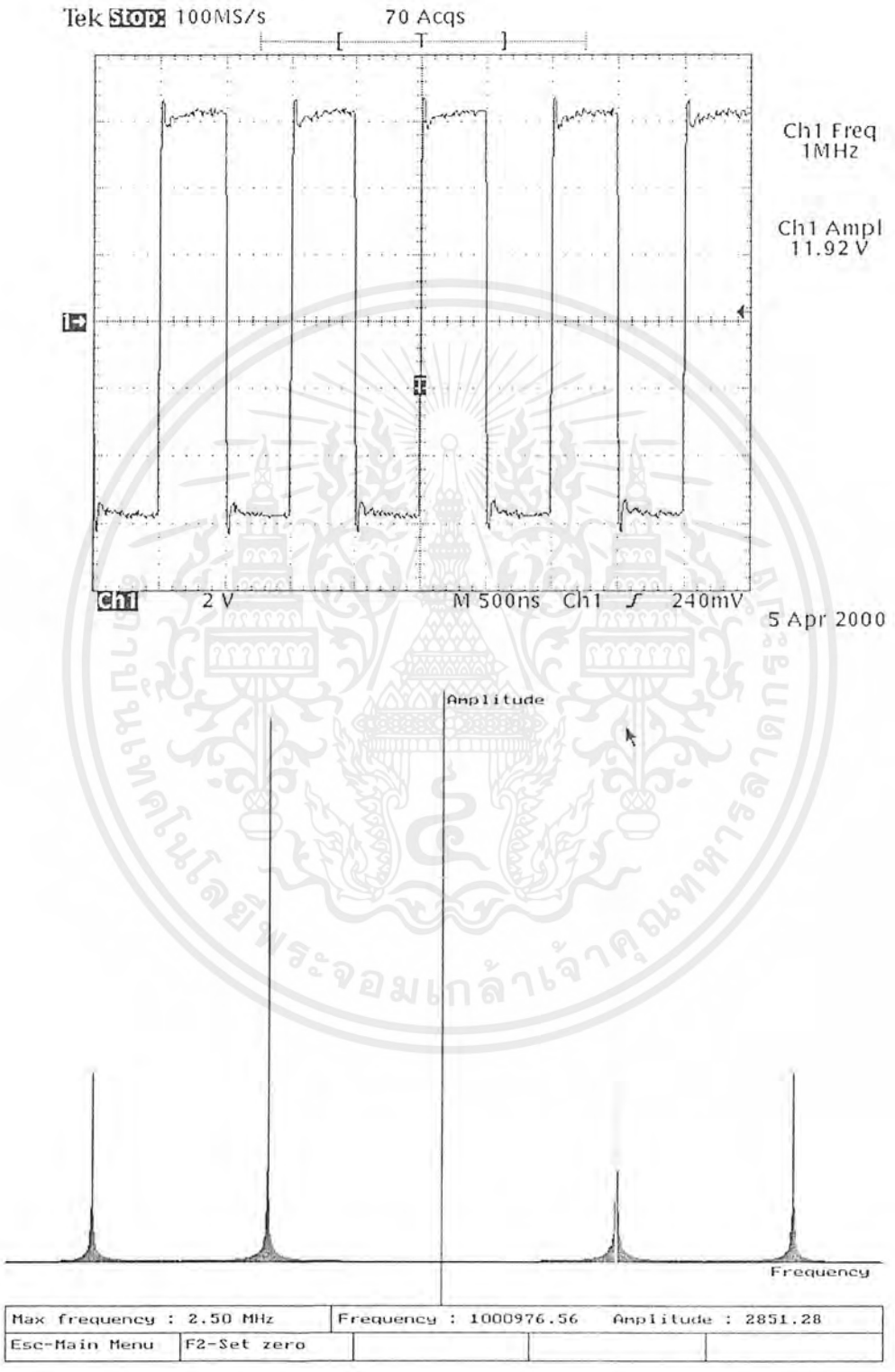
รูปที่ 4.14 แสดงผลการวัดสัญญาณสามเหลี่ยม 100 กิโลเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณซายน์ที่ความถี่ 1 เม็กกะเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



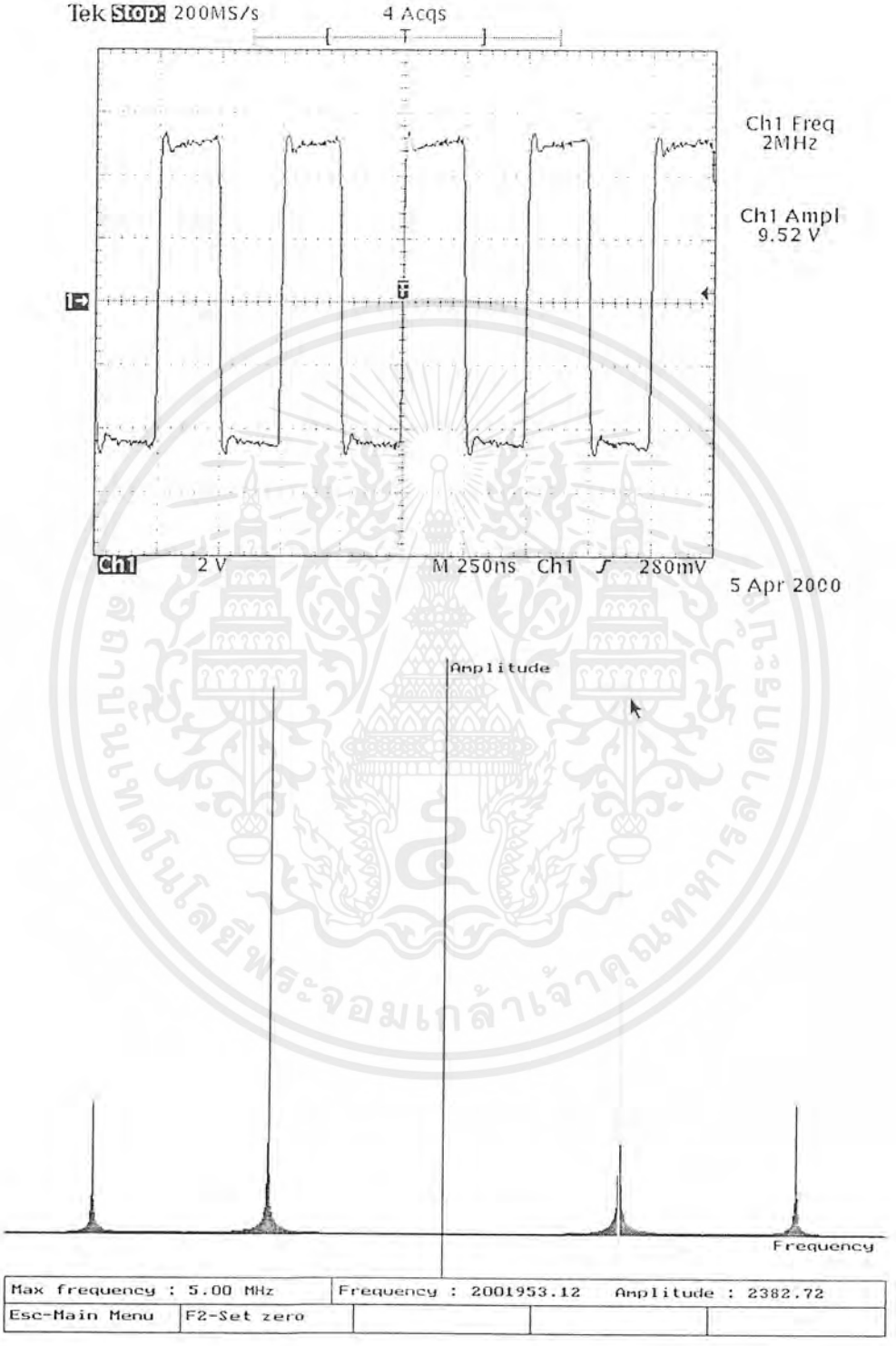
รูปที่ 4.15 แสดงผลการวัดสัญญาณซายน์ 1 เม็กกะเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณสี่เหลี่ยมที่ความถี่ 1 เม็กกะเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



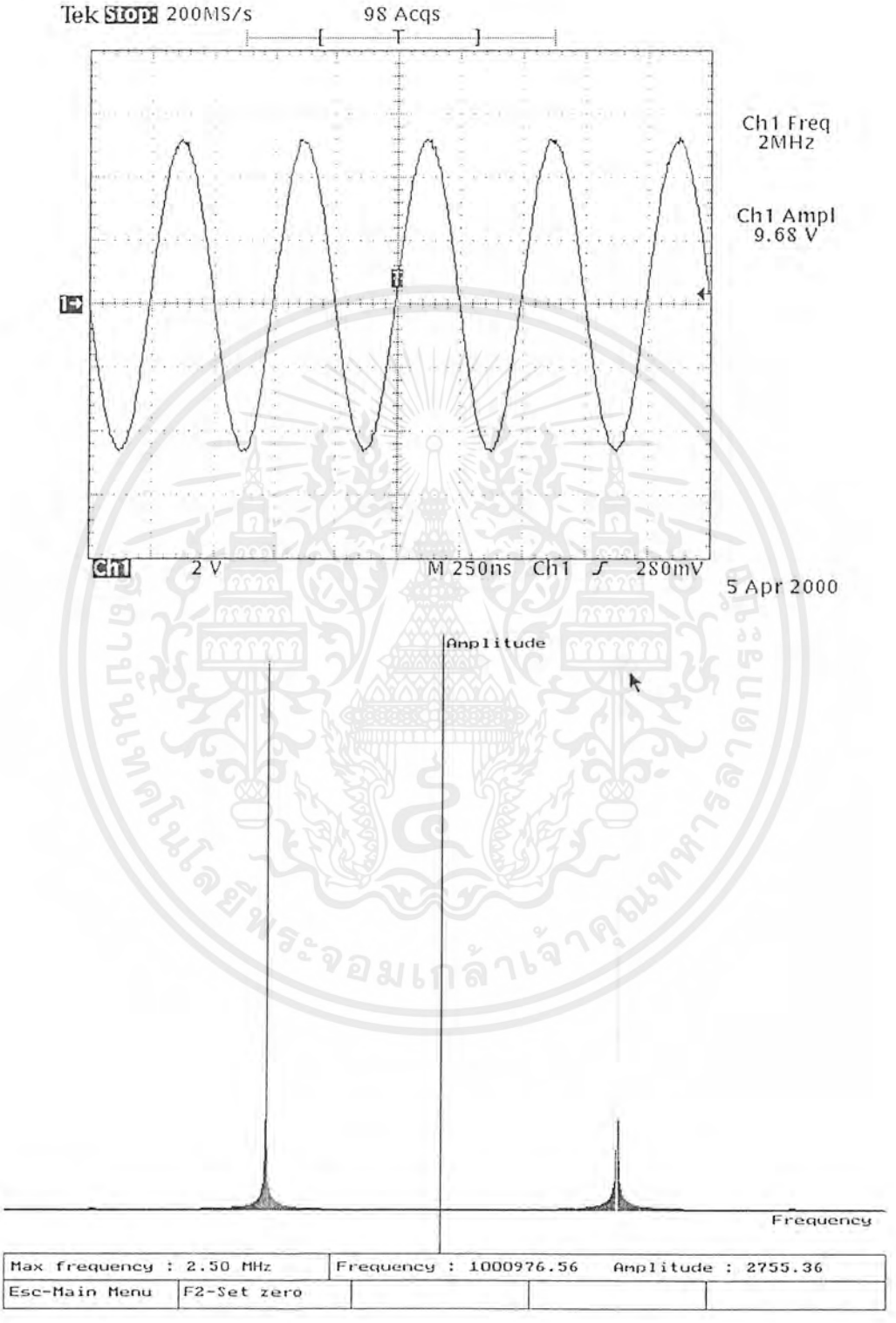
รูปที่ 4.16 แสดงผลการวัดสัญญาณสี่เหลี่ยม 1 เม็กกะเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณสี่เหลี่ยมที่ความถี่ 2 เม็กกะเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



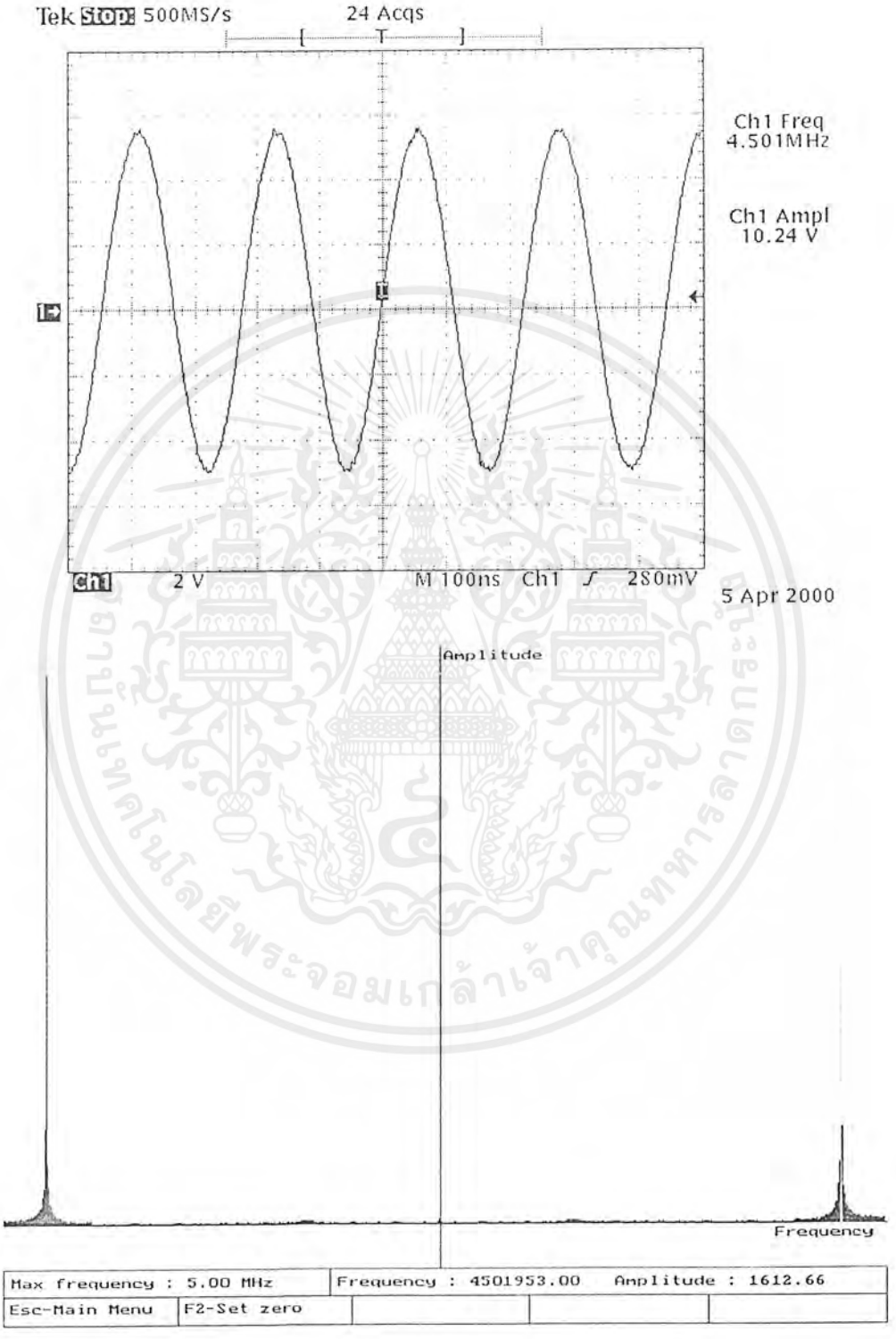
รูปที่ 4.17 แสดงผลการวัดสัญญาณสี่เหลี่ยม 2 เม็กกะเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณซายน์ที่ความถี่ 2 เม็กกะเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



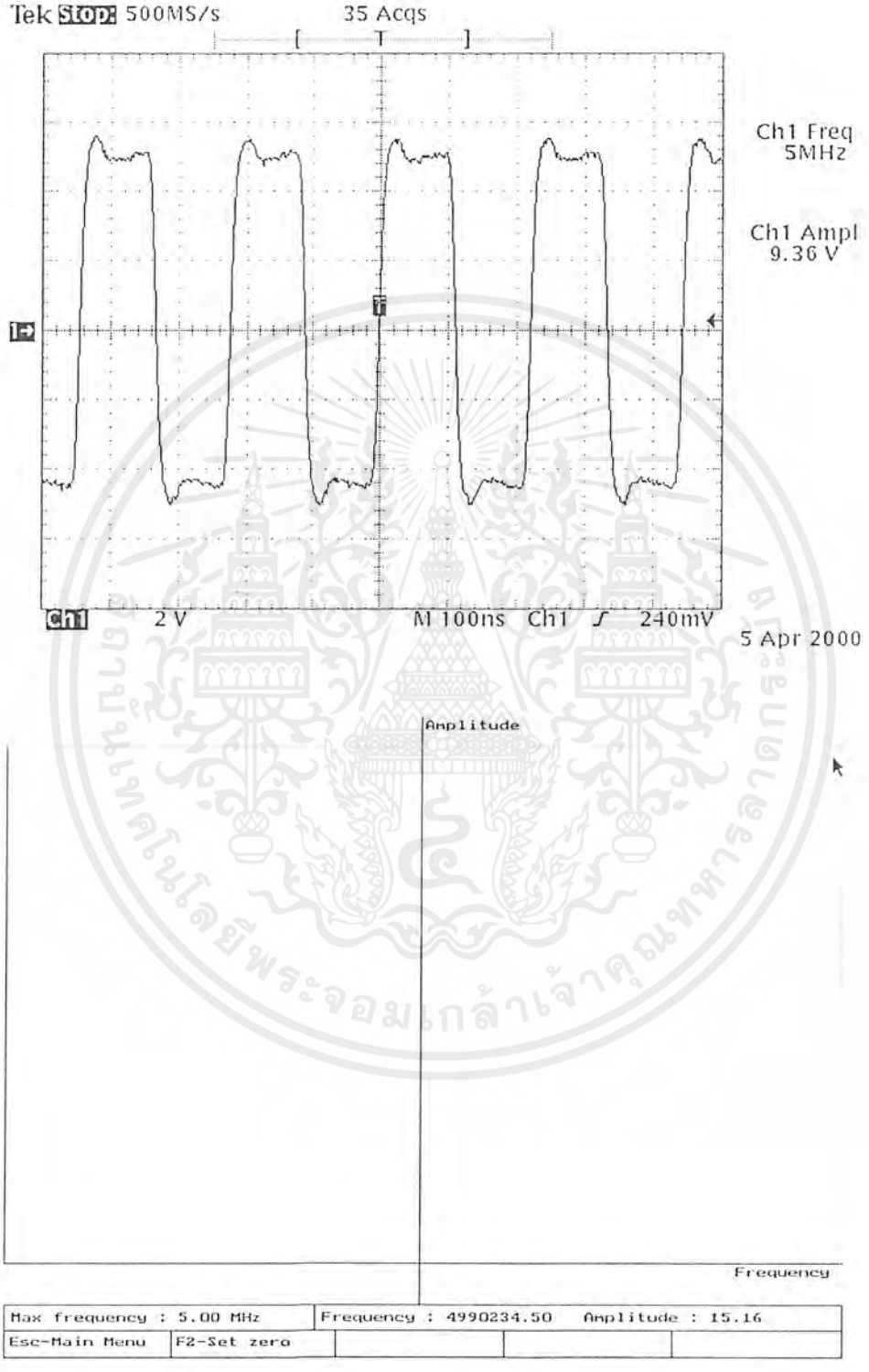
รูปที่ 4.18 แสดงผลการวัดสัญญาณซายน์ 2 เม็กกะเฮิร์ต เทียบกับอินพุท

ผลการวัดสัญญาณซายน์ที่ความถี่ 4.5 เม็กกะเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



รูปที่ 4.19 แสดงผลการวัดสัญญาณซายน์ 4.5 เม็กกะเฮิร์ต เทียบกับอินพุท

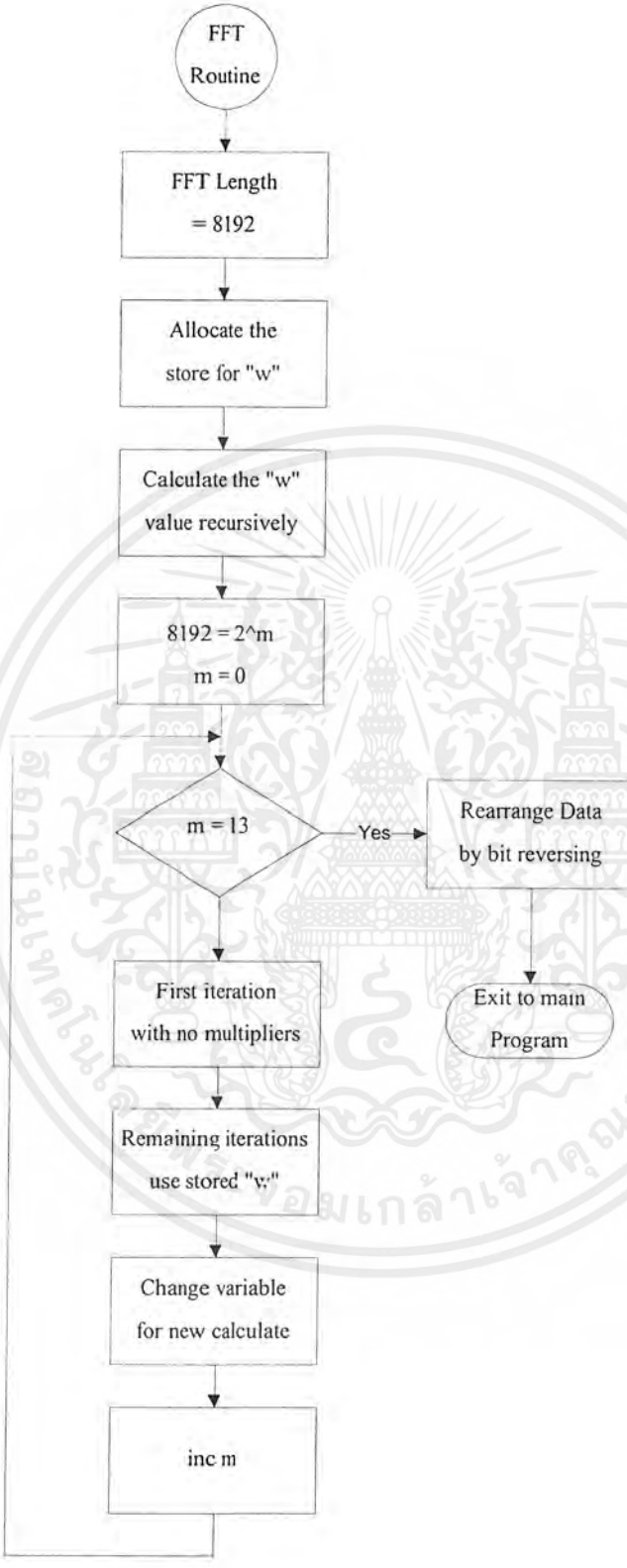
ผลการวัดสัญญาณสี่เหลี่ยมที่ความถี่ 5 เม็กกะเฮิร์ต เทียบกับสัญญาณอินพุทที่ป้อน



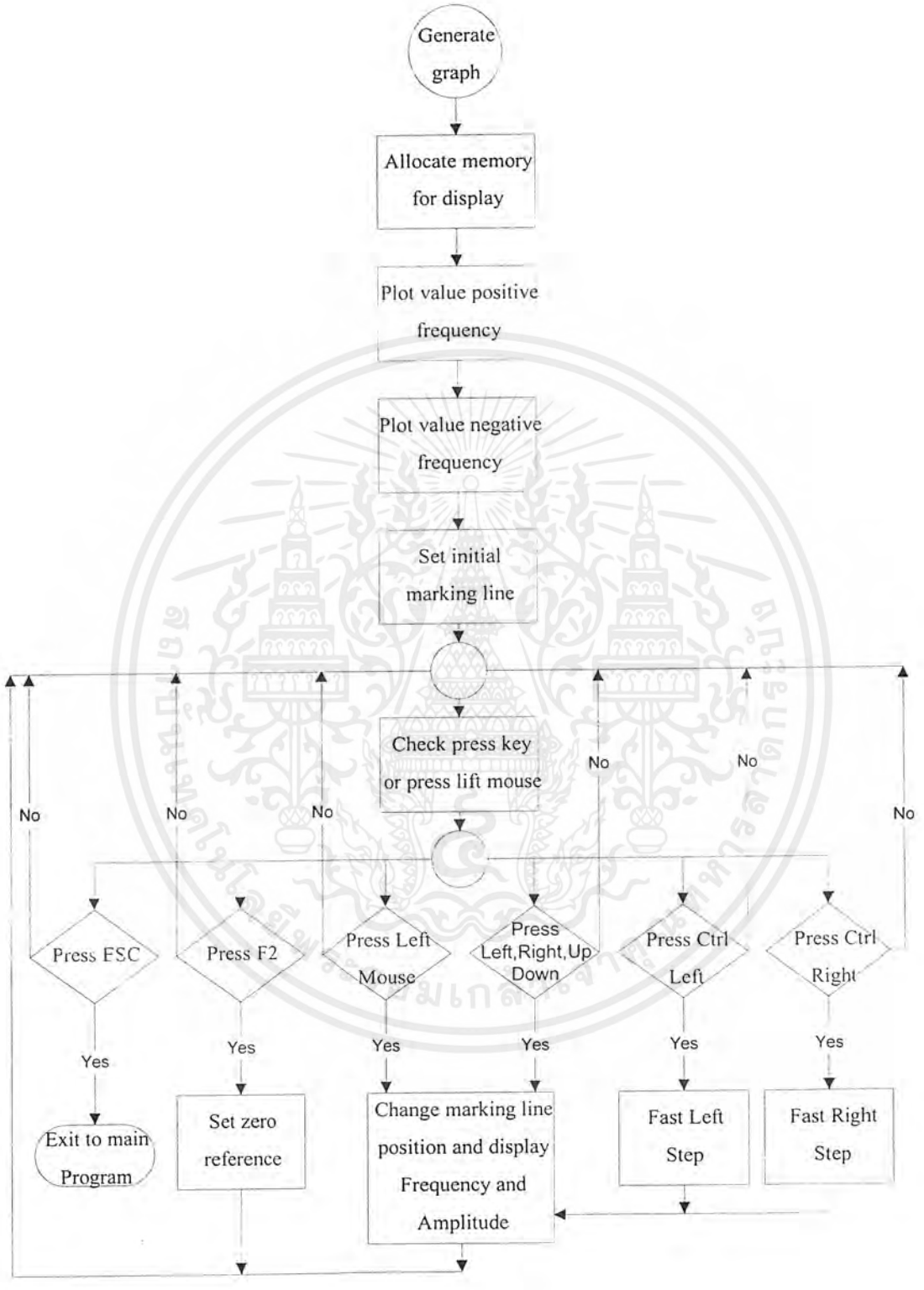
รูปที่ 4.20 แสดงผลการวัดสัญญาณสี่เหลี่ยม 5 เม็กกะเฮิร์ต เทียบกับอินพุท

4.3 แผนภาพ (Flowchart) การทำงานของโปรแกรม

รูปที่ 4.21 แสดงแผนภาพการทำงานของโปรแกรมสเปคตรัมอะนาไลเซอร์



รูปที่ 4.22 แผนภาพแสดงการทำงานส่วนโปรแกรมโมดูล (module) ฟาสต์ฟูเรียร์ทรานส์ฟอร์ม



รูปที่ 4.23 แผนภาพสร้างกราฟแสดงผล

บทที่ 5
บทวิจารณ์และสรุปผลการทดลอง

จากการทดลองเกิดปัญหาจากการทดลองหลายอย่างด้วยกัน อันเนื่องมาจาก เช่น การวัดสัญญาณที่ยากลำบากเนื่องจากการใช้พอร์ทในการอินเตอร์เฟสแบบสล๊อต ISA ซึ่งวางบนเมนบอร์ดซึ่งมีสัญญาณไฟรั่วไหลมากมายดังนั้น การทดลองจึงใช้สล๊อต ISA ที่เป็นอเนกประสงค์ต่อออกมาภายนอก และจากการออกแบบแผ่นวงจรมีการผิดพลาดหลายอย่าง เนื่องจากกราวด์ที่ต้องแยกให้ถูกต้องคือ กราวด์อนาล็อก และกราวด์ดิจิตอล ถ้าแยกไม่ถูกต้องจะเกิดการกวนกันของสัญญาณทั้งสองทำให้สัญญาณเกิดการผิดเพี้ยนได้

อุปกรณ์ที่ใช้เป็นอนาล็อกบัฟเฟอร์คือออปแอมป์ที่ใช้ต้องมีอัตราสลูสูงๆ และอัตราการเปลี่ยนของแรงดันทางเอาท์พุทต้องมากคือ 12 โวลท์ ซึ่งในตลาดหายาก ดังนั้นจากการใช้งานจึงใช้วงจร RELAY มาทำการเลือกสัญญาณทางอินพุทเพื่อให้ค่าที่คำนวณได้ถูกต้อง ด้านการเขียนโปรแกรมประสบปัญหาคือขาดประสบการณ์ในการเขียนโปรแกรมภาษา ซี แต่ภาษา ซี มีข้อเด่นหลายอย่างทำให้สามารถนำมาประยุกต์ใช้งานได้ เช่น มีฟังก์ชั่นไว้สำหรับการใช้งานเฉพาะซึ่งผู้ใช้สามารถเรียกใช้งานได้ง่าย หากจะพัฒนาก็คือการพัฒนาให้การใช้งานได้สะดวกขึ้น จะเห็นว่าส่วนของโมดูลการคำนวณฟาสต์ฟูเรียร์ทรานส์ฟอร์มจะคงรูปเดิม สามารถนำมาใช้ได้เลย โดยการรับอินพุทเข้าไปและทำการพล็อต(Plot)ค่าต่างๆออกมา ซึ่งผลจากการทดลองวัดสัญญาณต่างๆก็จะได้สเปคตรัมของสัญญาณนั้นๆ ทำให้ทราบว่าสัญญาณนั้นประกอบขึ้นจากสัญญาณความถี่ต่างๆกันและมีขนาดที่ต่างกัน

แนวการพัฒนาต่อคือ เพิ่มย่านวัดให้สูงขึ้น เขียนโปรแกรมให้ใช้งานได้สะดวกขึ้น และอ่านค่าความถี่ละเอียดขึ้น แสดงขนาดเป็นหน่วยเดซิเบล(dB) ให้โปรแกรมทำงานและอ่านค่าเป็นอัตโนมัติ มีการเก็บภาพที่ได้จากการวัด ทำงานเป็นเวลาปัจจุบัน(Real Time)

ภาคผนวก

ภาคผนวก ก

โปรแกรม (Source Code) เครื่องวิเคราะห์สเปคตรัม

```
/*************** Computer Controled Spectrum Analyzer  ****************/
#include <stdio.h>
#include <bios.h>
#include <dos.h>
#include <conio.h>
#include <alloc.h>
#include <string.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <graphics.h>
#include "funcmse.h"

typedef struct {
      float real, imag;
} COMPLEX;

#define PORT_CLEARCOUNTER     0x0300
#define PORT_STARTADC         0x0301
#define PORT_WAITADCREADY     0x0302
#define PORT_SETREADDATA      0x0303
#define PORT_READDATA         0x0304
#define PORT_SETCLOCK         0x0308

#define clear_counter()       inportb(PORT_CLEARCOUNTER)
#define set_clock(setclock)   inportb(setclock)
#define start_ADC()           inportb(PORT_STARTADC)
#define wait_ADC_ready()      inportb(PORT_WAITADCREADY)
#define set_read_data()       inportb(PORT_SETREADDATA)
#define read_ADC_data()       inportb(PORT_READDATA)

#define ESC                   0x011B
#define F1                    0x3B00
#define F2                    0x3C00
#define F3                    0x3D00
#define F4                    0x3E00
#define F5                    0x3F00
#define F6                    0x4000
#define F7                    0x4100
#define F8                    0x4200
#define F9                    0x4300
#define F10                   0x4400
#define SPACE                 0x3920
#define LEFT                  0x4B00
#define RIGHT                 0x4D00
#define UP                    0x4800
#define DOWN                  0x5000
#define CTRL_LEFT             0x7300
#define CTRL_RIGHT            0x7400

#define PI                    3.14159265

/* #define AMOUNT_SAMPLE     8192  */
/* #define N_CONSTANT        13    */
#define AMOUNT_SAMPLE        1024
#define N_CONSTANT           10
```

```
#define NORMAL_XSTEP            1
#define FAST_XSTEP              AMOUNT_SAMPLE/20

#define TMP_SAMPLE              2048
#define TMP_N                   11

int _port_setclock = 0x0308;

unsigned char far *data_buffer;
COMPLEX far *data_complex;
int maxx,maxy;
char *menu_msg[] =
{
      "Esc-Quit",
      "F2-Start Down",
      "F3-Start Up",
      "F4-Read File",
      ""
};

char *graph_msg[] =
{
      "Esc-Main Menu",
      "F2-Set zero",
      "","",""
};

/*************************************************************************/
void read_data(void);

/*************************************************************************/
void wait_anykey(void)
{
      while(!kbhit());
      bioskey(0);
}

int get_N(int nn)
{
    int cnt = 0;

    while ((nn = nn/2) >= 1)
        cnt++;
    return cnt;
}

/*************************************************************************/
void save_screen(void far *buf[4])
{
      unsigned size;
      int ystart=0, yend, yincr, block;

      yincr = (maxy+1) / 4;
      yend = yincr;
      size = imagesize(0, ystart, maxx, yend);
      /* get byte size of image */
```

```
        if (size != 0xFFFF)
        {
              for (block=0; block<=3; block++)
              {
                    if ((buf[block] = farmalloc(size)) == NULL)
                    {
                        closegraph();
                     printf("Error: not enough heap space in save_screen().\n");
                        exit(1);
                    }

                    getimage(0, ystart, maxx, yend, buf[block]);
                    ystart = yend + 1;
                    yend += yincr + 1;
              }
        }
        else
        {
              closegraph();
              printf("Error: image size is too large.\n");
              exit(1);
        }
}

void restore_screen(void far *buf[4])
{
        int ystart=0, yend, yincr, block;

        yincr = (maxy+1) / 4;
        yend  = yincr;

        for (block=0; block<=3; block++)
        {
              putimage(0, ystart, buf[block], COPY_PUT);
              /* farfree(buf[block]); */
              ystart = yend + 1;
              yend += yincr + 1;
        }
}

void save_scrn(void far *buf[1], int x1, int y1, int x2, int y2)
{
        unsigned size;

        size = imagesize(x1, y1, x2, y2);
        if (size != 0xFFFF)
        {
              if ((buf[0] = farmalloc(size)) == NULL)
              {
                    closegraph();
              printf("Error: not enough heap space in save_screen().\n");
                    exit(1);
              }
              getimage(x1, y1, x2, y2, buf[0]);
        }
        else
        {
```

```
            closegraph();
            printf("Error: image size is too large.\n");
            exit(1);
      }
}

void restore_scrn(void far *buf[1], int x1, int y1)
{
      putimage(x1, y1, buf[0], COPY_PUT);
      farfree(buf[0]);
}

/*************************************************************************/
/* Function for set window coordinate of graphic mode                    */
void set_window(int x1,int y1,int x2,int y2,int ops,int clip,int status)
{
      switch (ops)
      {
         case 0:setviewport(x1,y1,x2,y2,clip);break;
         case 1:rectangle(x1,y1,x2,y2);setviewport(x1+1,y1+1,x2-1,y2-
1,clip);break;
      }
      clearviewport();
      if (status == 0)
            setviewport(0,0,maxx,maxy,clip);
}

/*************************************************************************/
float get_max_real(COMPLEX *num, int n)
{
      COMPLEX *num_ptr;
      int i, amount;
      float max;

      num_ptr = num;
      max = num_ptr->real;
      amount = 1<<n;
      for (i=0;i<amount;i++)
      {
            if (max < num_ptr->real)
                  max = num_ptr->real;
            num_ptr++;
      }
      return max;
}

float get_max_imag(COMPLEX *num, int n)
{
      COMPLEX *num_ptr;
      int i,amount;
      float max;
      num_ptr = num;
      max = num_ptr->imag;
      amount = 1<<n;
      for (i=0;i<amount;i++)
      {
            if (max < num_ptr->imag)
```

```
                max = num_ptr->imag;
            num_ptr++;
        }
        return max;
}

/*************************************************************************/
/* fft - In-place radix 2 decimation in frequency FFT  Requires pointer
to complex array, x and power of 2 sizeof FFT,m (size of FFT = 2^m)
Places FFT output on top of input COMPLEX array. void fft(COMPLEX *x,
int m)                                                                  */
/*************************************************************************/
void fft(COMPLEX *x,int m)

        static COMPLEX *w;          /* used to store the w complex array */
        static int mstore=0;         /* stores m for future reference */
        static int n = 1;            /* length of fft stored for future */
        COMPLEX u,temp,tm;
        COMPLEX *xi,*xip,*xj,*wptr;
        int i,j,k,l,le,windex;
        float arg,w_real,w_imag,wrecur_real,wrecur_imag,wtemp_real;

        if(m != mstore)
        {
            /* free previously allocated storage and set new m */
            if(mstore !=0) free(w);
            mstore = m;
            if(m == 0) return;              /* if m=0 then done */

            /* n = 2^m = fft length */
            n=1 << m;
            le= n/2;

            /* allocate the storage for w */
            w = (COMPLEX far *) farcalloc(le-1,sizeof (COMPLEX));
            if (!w)
            {
                printf ("Unable to allocate complex W array \n");
                exit(1);
            }

            /* calculate the w values recursively */
            arg = 4.0 * atan(1.0)/le;     /* PI/le calculation */
            wrecur_real = w_real = cos(arg);
            wrecur_imag = w_imag = -sin(arg);
            xj = w;
            for (j=1 ; j<le ; j++)
            {
                xj->real = wrecur_real;
                xj->imag = wrecur_imag;
                xj++;
                wtemp_real  = wrecur_real*w_real - wrecur_imag*w_imag;
                wrecur_imag = wrecur_real*w_imag + wrecur_imag*w_real;
                wrecur_real = wtemp_real;
            }
        }
```

```
/* start fft */
      le = n;
      windex = 1;
      for (l=0; l<m; l++)
      {
            le = le/2;

            /* first iteration with no multiplies */
            for( i=0; i<n; i=i+ 2*le)
            {
                  xi  = x+i;
                  xip = xi+le;
                  temp.real = xi->real + xip->real;
                  temp.imag = xi->imag + xip->imag;
                  xip->real = xi->real - xip->real;
                  xip->imag = xi->imag - xip->imag;
                  *xi = temp;
            }

            /* remaining iterations use stored w */
            wptr = w + windex - 1;
            for (j=1; j<le; j++)
            {
                  u = *wptr;
                  for (i = j ; i<n; i= i+2*le)
                  {
                        xi = x + i;
                        xip = xi + le;
                        temp.real = xi->real + xip->real;
                        temp.imag = xi->imag + xip->imag;
                        tm.real   = xi->real - xip->real;
                        tm.imag   = xi->imag - xip->imag;
                        xip->real = tm.real*u.real - tm.imag*u.imag;
                        xip->imag = tm.real*u.imag + tm.imag*u.real;
                        *xi = temp;
                  }
                  wptr = wptr + windex;
            }
            windex = 2*windex;
      }

      /* rearrange data by bit reversing */
      j=0;
      for (i=1; i<(n-1); i++)
      {
            k = n/2;
            while(k<=j)
            {
                  j = j-k;
                  k = k/2;
            }
            j = j + k;
            if (i < j)
            {
                  xi = x+i;
                  xj = x+j;
```

```
                temp = *xj;
                *xj = *xi;
                *xi = temp;
            }
        }
    }

    void fft_gmode(COMPLEX *x,int m)
    {
        set_window(0,0,maxx,maxy-40,0,1,1);
        outtextxy(5,5,"Calculating Fast Fourier Transform <FFT>...");

        fft(x,m);

        outtextxy(5,15,"Ready...");
        delay(100);
        set_window(0,0,maxx,maxy-40,0,1,0);
    }

    /*************************************************************************/
    int read_datafile(void)
    {
        register unsigned i;
        int tmp;
        float ftmp;
        unsigned char far *buffer;
        COMPLEX far *complx;
        FILE *fileInput;
        int cnt = 0;

      if ((fileInput = fopen("input.dat", "r")) != NULL)
      {
          complx = data_complex;
          while (!feof(fileInput))
          {
              fscanf(fileInput, "%X", &tmp);
              ftmp = (float)tmp/0xFFFF;
              cnt++;
              complx->real = ftmp;
              complx->imag = 0;
              complx++;
          }
          fclose(fileInput);
          return cnt-1;
      }
      else
          return -1;
    }

    int wait_datafile(void)
    {
        int iRetVal;

      printf("Read Data... ");
      iRetVal = read_datafile();
      printf("OK\n");
      return iRetVal;
```

```
}

void display_fft(int n)
{
      register i;
      int ch;
      COMPLEX far *complx;

      clrscr();
      complx = data_complex;
      for (i=0; i<(1<<n); i++)
      {
            printf("Index %d: ", i);
            printf("%12.5f  %12.5f\n",complx->real,complx->imag);
            complx++;
            if (kbhit())
            {
                  ch = bioskey(0);
                  switch (ch)
                  {
                        case ESC:
                              return;

                        case SPACE:
                              printf("\n-- more --");
                              bioskey(0);
                              printf("\r          \r");
                              break;
                  }
            }
        if ((i != 0)&&(i%22 == 0))
        {
            printf("\n-- more --");
            bioskey(0);
            printf("\r          \r");
        }
      }
}



void show_debug_menu(void)
{
      printf("\n");
      puts("1   - Clear counter          F1   - Display data buffer");
      puts("2   - Set clock              F2   - Display FFT");
      puts("3   - Start ADC              F9   - Read data from file");
      puts("4   - Wait ADC to be ready   F10  - Create data");
      puts("5   - Set read data");
      puts("6   - Read ADC data");
      puts("?   - Help Menu              ESC  - Exit");
}

void debug_mode(void)
```

```
}

int set_tmp_data(void)
{
      unsigned int i;
      float time, angle1, angle2;
      COMPLEX far *complx;

    printf("Create Example data... ");
      complx = data_complex;
      time = 1/(10E6/(1<<(_port_setclock-0x308)));
      for (i=0; i<TMP_SAMPLE;i++)
        {
            /* time = i*(1/(10E6/(1<<_port_setclock-0x0308)));  */
             angle1 = i*(1000*2*PI*time);
            /* complx->real = sin(100*time)+cos(200*time); */
            complx->real = sin(angle1);
            complx->imag = 0;
            complx++;
        }
    printf("OK\n");
      return TMP_N;
}

void display_data(int n)
{
      register i, j;
      int ch;
      unsigned char far *buffer;
    int total_sample = 1<<n;

      buffer = data_buffer;
      for (i=0; i<total_sample/16; i++)
      {
            cprintf("\r\nAddress %04X: ", i*16);
            for (j=0; j<16; j++)
            {
                  cprintf("%02X", *buffer++);
                  if (j != 7) putch(' ');
                  else putch('-');
            }
            if (kbhit())
            {
                  ch = bioskey(0);
                  switch (ch)
                  {
                        case ESC:
                              return;

                        case SPACE:
                              printf("\n-- more --");
                              bioskey(0);
                              printf("\r          \r");
                              break;
                  }
            }
      }
```

```
}

int set_tmp_data(void)
{
      unsigned int i;
      float time, angle1, angle2;
      COMPLEX far *complx;

    printf("Create Example data... ");
      complx = data_complex;
      time = 1/(10E6/(1<<(_port_setclock-0x308)));
      for (i=0; i<TMP_SAMPLE;i++)
        {
            /* time = i*(1/(10E6/(1<<_port_setclock-0x0308)));  */
             angle1 = i*(1000*2*PI*time);
            /* complx->real = sin(100*time)+cos(200*time); */
            complx->real = sin(angle1);
            complx->imag = 0;
            complx++;
        }
    printf("OK\n");
      return TMP_N;
}

void display_data(int n)
{
      register i, j;
      int ch;
      unsigned char far *buffer;
    int total_sample = 1<<n;

      buffer = data_buffer;
      for (i=0; i<total_sample/16; i++)
      {
            cprintf("\r\nAddress %04X: ", i*16);
            for (j=0; j<16; j++)
            {
                  cprintf("%02X", *buffer++);
                  if (j != 7) putch(' ');
                  else putch('-');
            }
            if (kbhit())
            {
                  ch = bioskey(0);
                  switch (ch)
                  {
                        case ESC:
                              return;

                        case SPACE:
                              printf("\n-- more --");
                              bioskey(0);
                              printf("\r          \r");
                              break;
                  }
            }
      }
```

```
}

void display_fft(int n)
{
      register i;
      int ch;
      COMPLEX far *complx;

      clrscr();
      complx = data_complex;
      for (i=0; i<(1<<n); i++)
      {
            printf("Index %d: ", i);
            printf("%12.5f  %12.5f\n",complx->real,complx->imag);
            complx++;
            if (kbhit())
            {
                  ch = bioskey(0);
                  switch (ch)
                  {
                        case ESC:
                              return;

                        case SPACE:
                              printf("\n-- more --");
                              bioskey(0);
                              printf("\r          \r");
                              break;
                  }
            }
        if ((i != 0)&&(i%22 == 0))
        {
            printf("\n-- more --");
            bioskey(0);
            printf("\r          \r");
        }
      }
}



void show_debug_menu(void)
{
      printf("\n");
      puts("1   - Clear counter          F1   - Display data buffer");
      puts("2   - Set clock              F2   - Display FFT");
      puts("3   - Start ADC              F9   - Read data from file");
      puts("4   - Wait ADC to be ready   F10  - Create data");
      puts("5   - Set read data");
      puts("6   - Read ADC data");
      puts("?   - Help Menu              ESC  - Exit");
}

void debug_mode(void)
```

```
  int ch;
int n = N_CONSTANT, nn;

  clrscr();
  puts("\n[DebugMode]\n");
  show_debug_menu();
  for (;;)
  {
        printf("\n> ");
    ch = bioskey(0);
    switch (ch & 0xFF)
        {
        case '1':
                    puts("Counter cleared.");
                    clear_counter();
                    break;

        case '2':
                    puts("Clock set.");
                    set_clock(_port_setclock);
                    break;

              case '3':
                    puts("ADC started.");
                    start_ADC();
                    break;

              case '4':
                    puts("Waiting for ADC to be ready");
                    puts("Press ESC to abort...");
                    while ((wait_ADC_ready() & 0x01) == 0 )
                    {
                          if (kbhit())
                          {
                                if ((ch = bioskey(0)) == ESC)
                                {
                                      puts("Abort...");
                                      break;
                                }
                          }
                    }
                    break;

              case '5':
                    puts("Read data set.");
                    set_read_data();
                    break;

              case '6':
                    puts("Reading data...");
                    read_data();
                    break;

              case '?':
                    show_debug_menu();
                    break;
```

```
        }

        switch (ch)
        {
            case F1:
                display_data(n);
                break;

            case F2:
                display_fft(n);
                break;

            case F9:
                if ((nn = wait_datafile()) != -1)
                {
                    n = get_N(nn);
                    fft(data_complex,n);
                }
                break;

            case F10:
                n = set_tmp_data();
                fft(data_complex,n);
                break;

            case ESC:
                puts("");
                return;
            }
        }
}

/*************************************************************************/
void read_data(void)
{
      register unsigned i;
      unsigned char tmp;
      unsigned char far *buffer;
      COMPLEX far *complx;

      buffer = data_buffer;
      complx = data_complex;
      for (i=0; i<AMOUNT_SAMPLE; i++)
      {
            tmp = read_ADC_data();
            *buffer++ = tmp;
     /* complx->real = tmp*6.4/255-3.2; change raw data to true value */
            complx->real = 2*6.4/255*tmp-6.4;
            complx->imag = 0;
            complx++;
      }
}

/* Wait data from A/D converter */
int wait_data(void)
{
      int ch;
```

```
	set_window(0,0,maxx,maxy-40,0,1,1);
	outtextxy(5,5,"Waiting for ADC ready, press ESC to abort, F10 to
ignore...");

	clear_counter();
	 set_clock(_port_setclock);
	start_ADC();
	while ((0x01 & wait_ADC_ready()) == 0)
	{
		if (kbhit())
		{
			ch = bioskey(0);
			switch (ch)
			{
				case ESC:
					outtextxy(5,15,"Terminated...");
					return 0;

				case F10:
					break;
			}
		}
	}
	clear_counter();
	set_read_data();
	 read_data();

	outtextxy(5,15,"Ready...");
	delay(100);
	set_window(0,0,maxx,maxy-40,0,1,0);
	return N_CONSTANT;
}

/*************************************************************************/
void show_main_menu(void)
{
	clrscr();
	printf("Spectrum Analyzer version 2000 [KMIT'L] .\n\n");
	printf("ESC    - Quit\n");
	printf("F2     - Wait Data\n");
	printf("F3     - Display Data\n");
	printf("F4     - Find FFT\n");
	printf("F5     - Display FFT\n");
	printf("F6     - Display Graph\n");
	printf("F10    - Sample Data\n");
	printf("Up,Down- Change Sampling frequency
[%04X]\n",_port_setclock);
}

void display_gscreen(void)
{
	int i, x_start;

	Cursor_Off();
	set_window(0,maxy-20,maxx,maxy,1,1,0);
	for (i=0;i<5;i++)
```

```
        {
                x_start = i*maxx/5;
                line(x_start,maxy-20,x_start,maxy);
                outtextxy(x_start+5,maxy-15,graph_msg[i]);
        }
        Cursor_On();
}

int check_gbutton(int x, int y)
{
        int i;

        for (i=1;i<=5;i++)
                if (x>(i-1)*maxx/5 && x<i*maxx/5)
                        if (y>maxy-20 && y<maxy)
                                return i;
        if (x>=0 && x<=maxx)
                if (y>=0 && y<=maxy-40)
                        return 6;
        return 0;
}

void display_xy(float freq, float amplitude)
{
        char x_txt[50],y_txt[50];
        struct viewporttype viewinfo;

        Cursor_Off();
        getviewsettings(&viewinfo);

        sprintf(x_txt,"Frequency : %1.2f",freq);
        sprintf(y_txt,"Amplitude : %1.2f",amplitude);
        set_window(maxx-400,maxy-39,maxx-200,maxy-20,1,1,1);
        outtextxy(5,5,x_txt);
        set_window(maxx-200,maxy-39,maxx,maxy-20,1,1,1);
        outtextxy(5,5,y_txt);

        setviewport(viewinfo.left,viewinfo.top,viewinfo.right,viewinfo.bot
tom,viewinfo.clip);
        Cursor_On();
}

#define NO_YSTEP        100
#define SPACE_YDOT      20
void gen_graph(int n)
{
        int mpos_x, mpos_y;
        int button_no;
        int max_gx, max_gy;
        int y_start;

        int i,ch,update_flag = 0;
        int old_color;
        float x_step,y_step;
        int total_sample;
        float total_time;
        int x1,y1,x2,y2;
```

```
        float max_real, max_imag;
        float tmp;

        int xpos, ypos, old_xpos, old_ypos;
        float xaxis,yaxis;
        int f_step;
        int normal_ystep;

        float far *f_axis;
        int far *idx, *pos_idx;
        COMPLEX far *complx;
        void far *hbuf[1], *vbuf[1];

        display_gscreen();
        Cursor_Off();
        if ((f_axis = (float far *)farcalloc(AMOUNT_SAMPLE,sizeof(float)))
== NULL)
        {
                closegraph();
                puts("Can't allocate memory\n");
                exit(1);
        }
      if ((idx = (int far *)farcalloc(AMOUNT_SAMPLE,sizeof(int))) == NULL)
        {
                closegraph();
                puts("Can't allocate memory\n");
                exit(1);
        }
if ((pos_idx = (int far *)farcalloc(AMOUNT_SAMPLE,sizeof(int))) == NULL)
        {
                closegraph();
                puts("Can't allocate memory\n");
                exit(1);
        }

        max_gx = maxx;
        max_gy = maxy-40;
        y_start = max_gy-30;
        set_window(0,0,max_gx,max_gy,0,1,1);
        total_sample = 1 << n;
        total_time = (1/(10E6/(1<<_port_setclock-0x0308)))*total_sample;
        /* total_time = (1/8e3)*total_sample; */
        /* find maximum amplitude of complex number of FFT */
        max_real = fabs(get_max_real(data_complex,n));
        max_imag = fabs(get_max_imag(data_complex,n));

        x_step = (float)max_gx/total_sample;
        y_step = (float)(y_start-SPACE_YDOT)/(max(max_real,max_imag));
        /* Draw X axis  */
        line(0,y_start,max_gx,y_start);
        outtextxy(max_gx-80,y_start+3,"Frequency");
        /* Draw Y axis */
        line(max_gx/2,0,max_gx/2,max_gy);
        outtextxy(max_gx/2+3,3,"Amplitude");

        for (i=0;i<total_sample/2;i++)
        {
```

```
        *(idx+i) = i;
        *(f_axis+i+total_sample/2) = i/total_time;
        *(pos_idx+i+total_sample/2) = (max_gx/2)+*(idx+i)*x_step;
}
for (i=total_sample/2;i<total_sample;i++)
{
        *(idx+i) = -(total_sample-i);
        *(f_axis+i-total_sample/2) = -(total_sample-i)/total_time;
        *(pos_idx+i-total_sample/2) = (max_gx/2)+*(idx+i)*x_step;
}

/* draw_fft(x_step,y_step,idx,max_gx,y_start); */
complx = data_complex;
old_color = getcolor();
setcolor(YELLOW);
/* Draw amplitude of each result FFT  */
for (i=0;i<total_sample;i++)
{
        x1 = (max_gx/2)+(*(idx+i)*x_step);
        x2 = x1;
        y1 = y_start;
        y2 = (y_start)-fabs(complx->real)*y_step;
        line(x1,y1,x2,y2);
        y2 = (y_start)-fabs(complx->imag)*y_step;
        line(x1,y1,x2,y2);
        complx++;
}
setcolor(old_color);

/* set initial marking value */
xpos = max_gx/2;
ypos = y_start;
old_xpos = xpos;
old_ypos = ypos;
save_scrn(hbuf,xpos,0,xpos,max_gy);
save_scrn(vbuf,0,ypos,max_gx,ypos);
/* Draw X-Y marking */
old_color = getcolor();
setcolor(LIGHTBLUE);
line(xpos,0,xpos,max_gy);       /* Y-axis */
line(0,ypos,max_gx,ypos);       /* X-axis */
setcolor(old_color);
xaxis = 0;
yaxis = 0;
f_step = total_sample/2;
if(max(max_real,max_imag)< NO_YSTEP)
        normal_ystep = 1;
else
        normal_ystep = max(max_real,max_imag)/NO_YSTEP;
display_xy(xaxis,yaxis);
Cursor_On();
while (ch != ESC)
{
        if (Leftb_Pressed())
        {
                Wait_On(LEFTB);
                Mouse_Position(&mpos_x, &mpos_y);
```

```
        button_no = check_gbutton(mpos_x,mpos_y);
        switch (button_no)
        {
              case 1:
                    ch = ESC;
                    break;

              case 2:
                    xpos = max_gx/2;
                    ypos = y_start;
                    xaxis = 0;
                    yaxis = 0;
                    f_step = total_sample/2;
                    break;

              case 6:
                    tmp = (float) total_sample/max_gx;
                    if (mpos_x < xpos)
                          f_step = f_step-(xpos-mpos_x)*tmp;
                    else
                          f_step = f_step+(mpos_x-xpos)*tmp;
                    xpos = *(pos_idx+f_step);
                    xaxis = *(f_axis+f_step);

                    if (mpos_y < ypos)
                          yaxis = yaxis+(ypos-mpos_y)/y_step;
                    else
                          yaxis = yaxis-(mpos_y-ypos)/y_step;
                    ypos = y_start-yaxis*y_step;
                    break;
        }
        update_flag = 1;
}

if (kbhit())
{
        ch = bioskey(0);
        switch (ch)
        {
              case F2:
                    xpos = max_gx/2;
                    ypos = y_start;
                    xaxis = 0;
                    yaxis = 0;
                    f_step = total_sample/2;
                    break;

              case F3:
                    x_step = (float)max_gx/(total_sample/2);
                    for (i=0;i<total_sample/2;i++)
*(pos_idx+i+total_sample/2) = (max_gx/2)+*(idx+i)*x_step;
                    for (i=total_sample/2;i<total_sample;i++)
*(pos_idx+i-total_sample/2) = (max_gx/2)+*(idx+i)*x_step;
                    complx = data_complex;
                    old_color = getcolor();
                    setcolor(YELLOW);
                    /* Draw amplitude of each result FFT */
```

```
        for (i=0;i<total_sample;i++)
        {
            x1 = (max_gx/2)+(*(idx+i)*x_step);
            x2 = x1;
            y1 = y_start;
            y2 = (y_start)-fabs(complx->real)*y_step;
            line(x1,y1,x2,y2);
            y2 = (y_start)-fabs(complx->imag)*y_step;
            line(x1,y1,x2,y2);
            complx++;
        }
        setcolor(old_color);

    /* set initial marking value */
        xpos = max_gx/2;
        ypos = y_start;
        old_xpos = xpos;
        old_ypos = ypos;
        save_scrn(hbuf,xpos,0,xpos,max_gy);
        save_scrn(vbuf,0,ypos,max_gx,ypos);
    /* Draw X-Y marking */
        old_color = getcolor();
        setcolor(LIGHTBLUE);
    line(xpos,0,xpos,max_gy);    /* Y-axis */
    line(0,ypos,max_gx,ypos);    /* X-axis */
        setcolor(old_color);
    xaxis = 0;
    yaxis = 0;

    f_step = total_sample/2;
    if(max(max_real,max_imag) < NO_YSTEP)
      normal_ystep = 1;
    else
      normal_ystep = max(max_real,max_imag)/NO_YSTEP;
      display_xy(xaxis,yaxis);
      Cursor_On();
                break;

            case F4:
                break;

            case LEFT:
                if (f_step-NORMAL_XSTEP >= 0)
                {
                    f_step-=NORMAL_XSTEP;
                    xpos = *(pos_idx+f_step);
                    xaxis = *(f_axis+f_step);
                }
                break;

            case CTRL_LEFT:
                if (f_step-FAST_XSTEP >= 0)
                {
                    f_step-=FAST_XSTEP;
                    xpos = *(pos_idx+f_step);
                    xaxis = *(f_axis+f_step);
                }
```

```
                        break;

                case RIGHT:
                        if (f_step+NORMAL_XSTEP <= total_sample-1)
                        {
                                f_step+=NORMAL_XSTEP;
                                xpos = *(pos_idx+f_step);
                                xaxis = *(f_axis+f_step);
                        }
                        break;

                case CTRL_RIGHT:
                        if (f_step+FAST_XSTEP <= total_sample-1)
                        {
                                f_step+=FAST_XSTEP;
                                xpos = *(pos_idx+f_step);
                                xaxis = *(f_axis+f_step);
                        }
                        break;

                case UP:
                        if (ypos > 0)
                        {
                                yaxis+=normal_ystep;
                                ypos = y_start-yaxis*y_step;
                        }
                        break;

                case DOWN:
                        if (ypos < max_gy)
                        {
                                yaxis-=normal_ystep;
                                ypos = y_start-yaxis*y_step;
                        }
                        break;
        }
        update_flag = 1;
}

if (update_flag)
{
        update_flag = 0;
        Cursor_Off();
        old_color = getcolor();
        setcolor(LIGHTBLUE);
        restore_scrn(hbuf,old_xpos,0);
        restore_scrn(vbuf,0,old_ypos);
        old_xpos = xpos;
        old_ypos = ypos;
        save_scrn(hbuf,xpos,0,xpos,max_gy);
        save_scrn(vbuf,0,ypos,max_gx,ypos);
        line(xpos,0,xpos,max_gy);
        line(0,ypos,max_gx,ypos);
        setcolor(old_color);
        display_xy(xaxis,yaxis);
        Cursor_On();
}
```

```
        }
        farfree(idx);
        farfree(pos_idx);
        farfree(f_axis);
        farfree(hbuf[0]);
        farfree(vbuf[0]);
        set_window(0,0,max_gx,max_gy,0,1,0);
}

void display_rate(int port)
{
        char txt[50];
        struct viewporttype viewinfo;

        Cursor_Off();
        switch (port)
        {
                case 0x0308: sprintf(txt,"Max frequency : 5.00 MHz");break;
                case 0x0309: sprintf(txt,"Max frequency : 2.50 MHz");break;
                case 0x030A: sprintf(txt,"Max frequency : 1.25 MHz");break;
                case 0x030B: sprintf(txt,"Max frequency :625.00 kHz");break;
                case 0x030C: sprintf(txt,"Max frequency :312.50 kHz");break;
                case 0x030D: sprintf(txt,"Max frequency :156.25 kHz");break;
                case 0x030E: sprintf(txt,"Max frequency :78.125 kHz");break;
                case 0x030F:sprintf(txt,"Max frequency :39.0625 kHz");break;
        }

        getviewsettings(&viewinfo);
        set_window(0,maxy-39,maxx-400,maxy-20,1,1,1);
        outtextxy(5,5,txt);
        setviewport(viewinfo.left,viewinfo.top,viewinfo.right,viewinfo.bot
tom,viewinfo.clip);
        Cursor_On();
}

void display_mscreen(void)
{
        int i, x_start;

        Cursor_Off();
        set_window(0,0,maxx,maxy-40,0,1,0);
        set_window(0,maxy-39,maxx,maxy-20,1,1,0);
        set_window(0,maxy-20,maxx,maxy,1,1,0);
        for (i=0;i<5;i++)
        {
                x_start = i*maxx/5;
                line(x_start,maxy-20,x_start,maxy);
                outtextxy(x_start+5,maxy-15,menu_msg[i]);
        }
        Cursor_On();
}

int check_mbutton(int x, int y)
{
        int i;

        for (i=1;i<=5;i++)
```

```
            if (x>(i-1)*maxx/5 && x<i*maxx/5)
                  if (y>maxy-20 && y<maxy)
                        return i;
      return 0;
}

/*************************************************************************/
void main(int argc, char *argv[])
{
      int ch;
    int n = N_CONSTANT, nn;
      int mpos_x, mpos_y, button_no;
      int gdriver = DETECT, gmode, errorcode; /*request auto detection*/
    FILE *fileInput;

      if (argc > 2)  /* check that user pass correct parameter or not */
      {
            puts("Usage:  SPECTRUM  [/d for debugging]");
            exit(0);
      }
      /* allocate memory for keeping true value */
      if ((data_buffer = (unsigned char far *)farmalloc(AMOUNT_SAMPLE))
== NULL)
      {
            puts("Can't allocate data buffer\n");
        exit(1);
      }
   /* allocate memory for keeping value to be used in FFT calculation */
      if ((data_complex = (COMPLEX far *)farcalloc(AMOUNT_SAMPLE,sizeof
(COMPLEX))) == NULL)
      {
            printf ("Can't allocate complex array\n");
        exit(1);
      }
      if (strnicmp(argv[1], "/d", 2) == 0) debug_mode();/* if using
debug mode */
    else
    {
      if ((fileInput = fopen(argv[1], "r")) != NULL)
        {
            printf("This feature is not support now!\n");
            exit(1);
        }
            else
            {
          /* initialize graphics mode */
            initgraph(&gdriver, &gmode, "");
          /* read result of initialization */
            errorcode = graphresult();
          if (errorcode != grOk)      /* an error occurred */
            {
               printf("Graphics error: %s\n", grapherrormsg(errorcode));
               printf("Press any key to halt:");
               getch();
            exit(1);                  /* return with error code */
            }
          /* trap_prt_scrn(); */
```

```
/* find maximum x value and maximum y value of monitor  */
  maxx = getmaxx();
  maxy = getmaxy();

  display_mscreen();            /* show main screen */
  display_rate(_port_setclock);
  Mouse_Reset();
  Cursor_On();                  /* cursor mouse is on */
  /* loop for checking key-pressed and mouse-pressed */
  while (ch != ESC)
  {
  if (Leftb_Pressed())/* check left button of mouse */
  {
  Wait_On(LEFTB);/* wait until left button is released */
  Mouse_Position(&mpos_x, &mpos_y);/* find mouse position */
  button_no = check_mbutton(mpos_x,mpos_y);
  switch (button_no)
  {
                        case 1:
                              ch = ESC;
                              break;

                        case 2:
                              n = wait_data();
                              fft_gmode(data_complex,n);
                              gen_graph(n);
                              display_mscreen();
                              display_rate(_port_setclock);
                              if (_port_setclock < 0x030F)
                                    _port_setclock++;
                              else
                                    _port_setclock = 0x0308;
                              display_rate(_port_setclock);
                              break;

                        case 3:
                              if (_port_setclock > 0x0308)
                                    _port_setclock--;
                              else
                                    _port_setclock = 0x030F;
                              display_rate(_port_setclock);
                              n = wait_data();
                              fft_gmode(data_complex, n);
                              gen_graph(n);
                              display_mscreen();
                              display_rate(_port_setclock);
                              break;

            case 4:
                if ((nn = read_datafile()) != -1)
                {
                    n = get_N(nn);
                    fft_gmode(data_complex,n);
                    gen_graph(n);
                }
                              display_mscreen();
                              display_rate(_port_setclock);
```

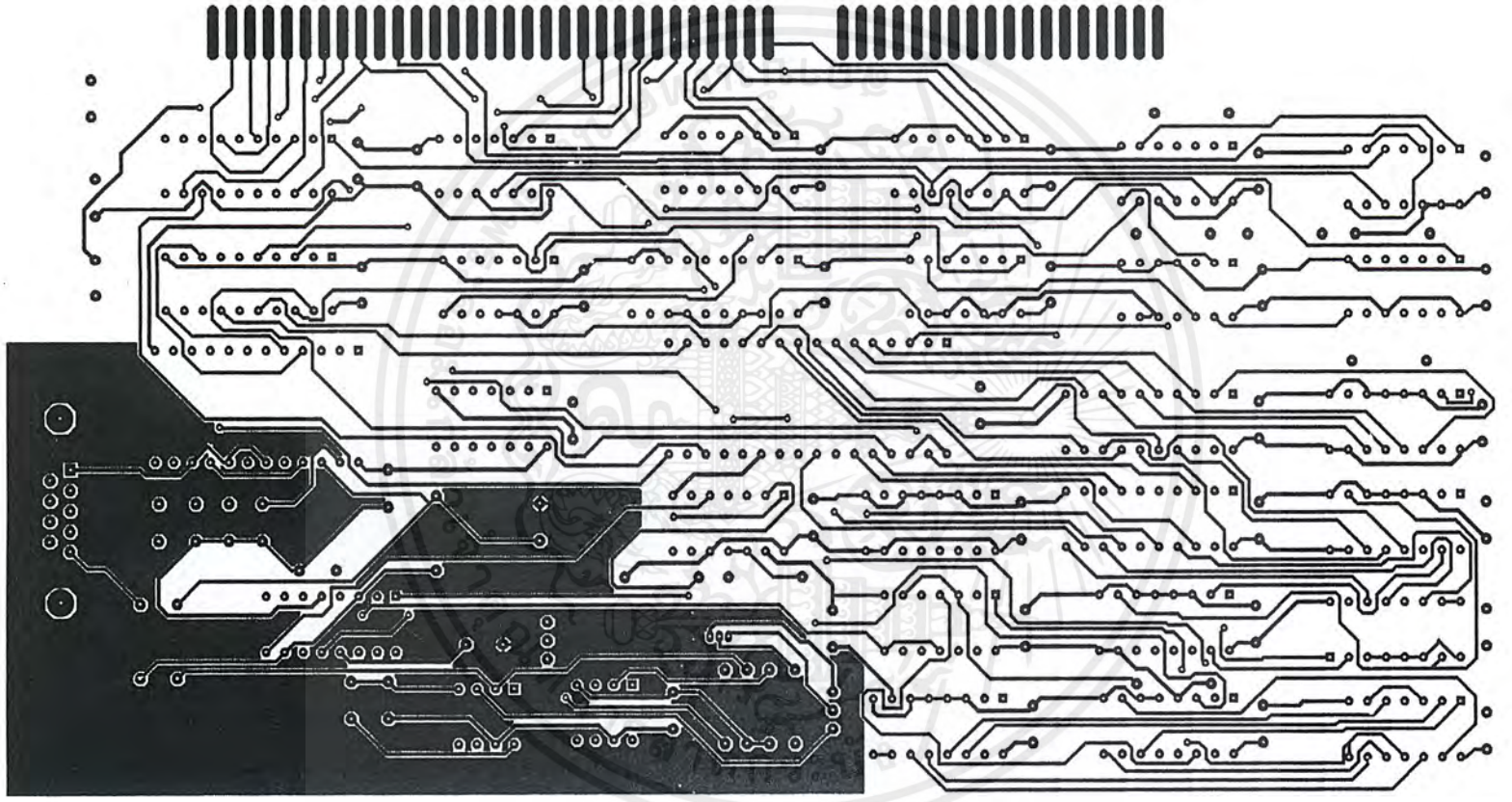
```
                                        break;

                                case 5:
                                      n = set_tmp_data();
                                    fft_gmode(data_complex,n);
                                    gen_graph(n);
                                    display_mscreen();
                                    display_rate(_port_setclock);
                                    break;
                                      }
                                }

                                if (kbhit())            /* check key pressed */
                                {
                                      ch = bioskey(0);
                                      switch (ch)
                                      {
                                          case F2:
                                                if (_port_setclock > 0x0308)
                                                      _port_setclock--;
                                                else
                                                      _port_setclock = 0x030F;
                                                display_rate(_port_setclock);
                                                break;

                                          case F3:
                                                n = wait_data();
                                                fft_gmode(data_complex,n);
                                                gen_graph(n);
                                                display_mscreen();
                                                display_rate(_port_setclock);
                                                break;

                                                case F4:
                                              if ((nn = read_datafile()) != -1)
                                                {
                                                      n = get_N(nn);
                                                  fft_gmode(data_complex,n);
                                                      gen_graph(n);
                                                }
                                                display_mscreen();
                                                display_rate(_port_setclock);
                                                break;
                                      }
                                }
                          }
                          closegraph();
                    }
              }
              farfree(data_complex);          /* release memory */
              farfree(data_buffer);           /* release memory */
        /* release_prt_scrn(); */
        }
```

```
/*****************************************************************************/
/*                                FUNCMSE                                    */
/*              Compose of function for interface with mouse                 */
/*****************************************************************************/
#define NOT_MOVED  0
#define MSE_RIGHT  1
#define MSE_LEFT   2
#define MSE_DOWN   3
#define MSE_UP     4
#define LEFTB      1
#define RIGHTB     2

 void Cmouse(int *fnum,int *arg2,int *arg3,int *arg4);
 int Mouse_Reset(void);
 void Cursor_On(void);
 void Cursor_Off(void);
 int Leftb_Pressed(void);
 void Mouse_Position(int *x,int *y);
 int Rightb_Pressed(void);
 void Set_Mouse_Position(int x,int y);
 void Mouse_Motion(int *deltax,int *deltay);
 void Wait_On(int button);
/*****************************************************************************/
void Cmouse(fnum,arg2,arg3,arg4)
  int *fnum,*arg2,*arg3,*arg4;
{
  union REGS inr,outr;
  inr.x.ax=*fnum;
  inr.x.bx=*arg2;
  inr.x.cx=*arg3;
  inr.x.dx=*arg4;
  int86(0x33,&inr,&outr);
  *fnum=outr.x.ax;
  *arg2=outr.x.bx;
  *arg3=outr.x.cx;
  *arg4=outr.x.dx;
}

/*****************************************************************************/
/* Function number 0                                                         */
/* reset mouse and move mouse cursor to the centre of screen                 */
/*****************************************************************************/
int Mouse_Reset(void)
{
  int fnum=0,arg2;
  Cmouse(&fnum,&arg2,NULL,NULL);
  if (fnum==0)
      printf("Mouse Driver is not installed");
  return(fnum);
}

/*****************************************************************************/
/* Function number 1                                                         */
/* display mouse cursor                                                      */
/*****************************************************************************/
void Cursor_On(void)
{
```

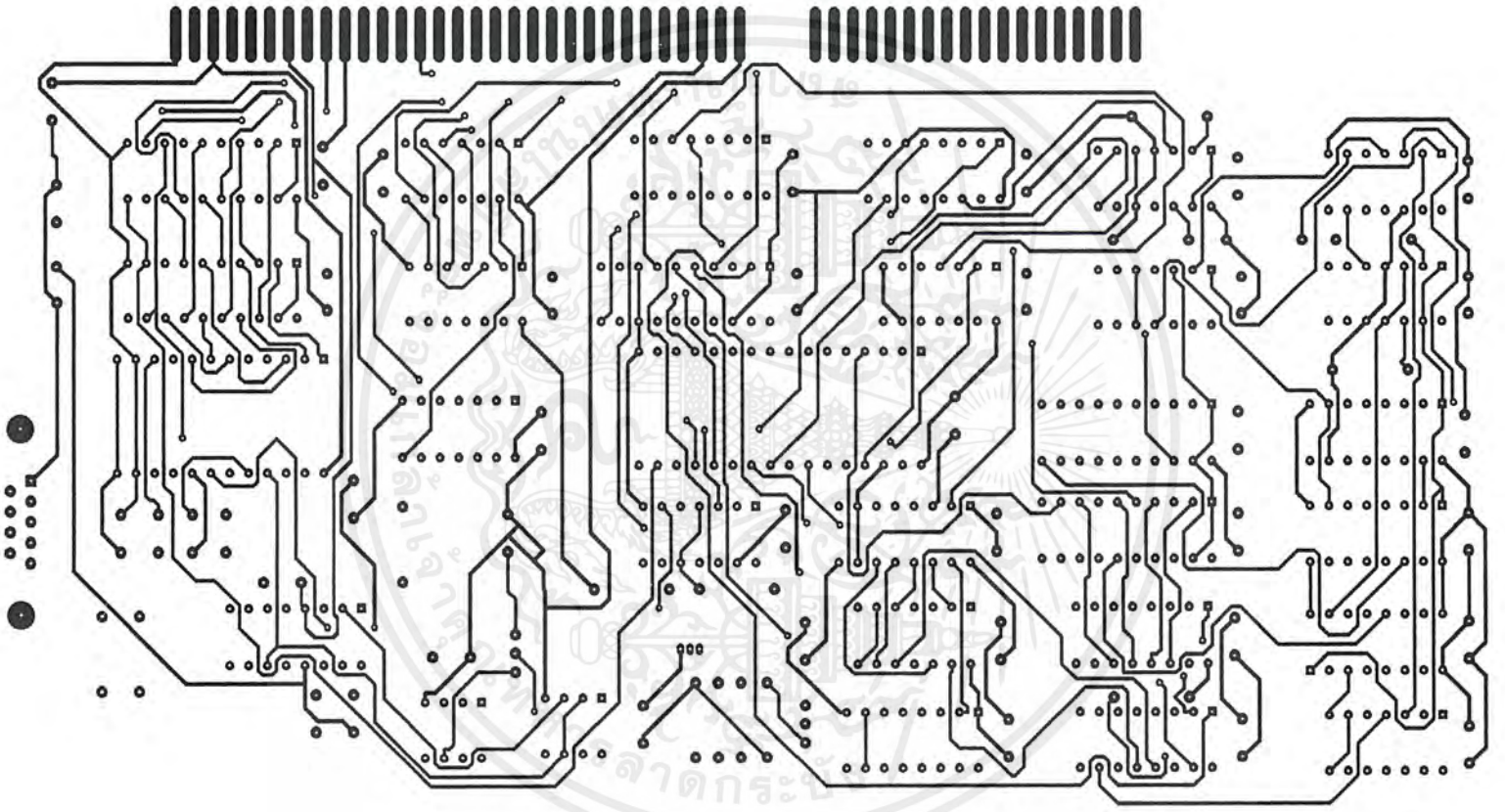
```
  int fnum=1;
  Cmouse(&fnum,&fnum,&fnum,&fnum);
}

/*************************************************************************/
/* Function number 2                                                     */
/* not display mouse cursor                                              */
/*************************************************************************/
void Cursor_Off(void)
{
  int fnum=2;
  Cmouse(&fnum,&fnum,&fnum,&fnum);
}

/*************************************************************************/
/* Function number 3                                                     */
/* check left button of mouse is pressed or not                          */
/* if left button is pressed will return 1                               */
/* else return 0                                                         */
/*************************************************************************/
int Leftb_Pressed(void)
{
  int fnum=3,arg2;
  Cmouse(&fnum,&arg2,&fnum,&fnum);
  return (arg2&1);
}


/*************************************************************************/
/* Function number 3                                                     */
/* get mouse cursor position                                             */
/*************************************************************************/
void Mouse_Position(x,y)
  int *x,*y;
{
  int fnum=3,arg3,arg4;
  Cmouse(&fnum,&fnum,&arg3,&arg4);
  *x=arg3;
  *y=arg4;
}

/*************************************************************************/
/* Function number 3                                                     */
/* check right button of mouse is pressed or not                         */
/* if right button is pressed will return 2                              */
/* else return 0                                                         */
/*************************************************************************/
int Rightb_Pressed(void)
{
  int fnum=3,arg2;
  Cmouse(&fnum,&arg2,&fnum.&fnum);
  return (arg2&2);
}

/*************************************************************************/
/* Function number 4                                                     */
/* set position of mouse cursor                                          */
```

```
/*************************************************************************/
void Set_Mouse_Position(x,y)
  int x,y;
{
  int fnum=4;
  Cmouse(&fnum,&fnum,&x,&y);
}

/*************************************************************************/
/* Function number 11                                                    */
/* check mouse is moved or not                                           */
/* if not move return 0                                                  */
/* if move right return 1                                                */
/* if move left return 2                                                 */
/* if move down return 3                                                 */
/* if move up return 4                                                   */
/*************************************************************************/
void Mouse_Motion(deltax,deltay)
  int *deltax,*deltay;
{
  int fnum=11,arg2,arg3,arg4;
  Cmouse(&fnum,&arg2,&arg3,&arg4);
  if (arg3>0) *deltax=MSE_RIGHT;
  else if (arg3<0) *deltax=MSE_LEFT;
         else *deltax=NOT_MOVED;
  if (arg4>0) *deltay=MSE_DOWN;
  else if (arg4<0) *deltay=MSE_UP;
         else *deltay=NOT_MOVED;
}

/*************************************************************************/
/* Function for waiting until mouse button is release                    */
/*************************************************************************/
void Wait_On(button)
  int button;
{
  if (button==LEFTB)
      while(Leftb_Pressed());
  else
      while(Rightb_Pressed());
}
```

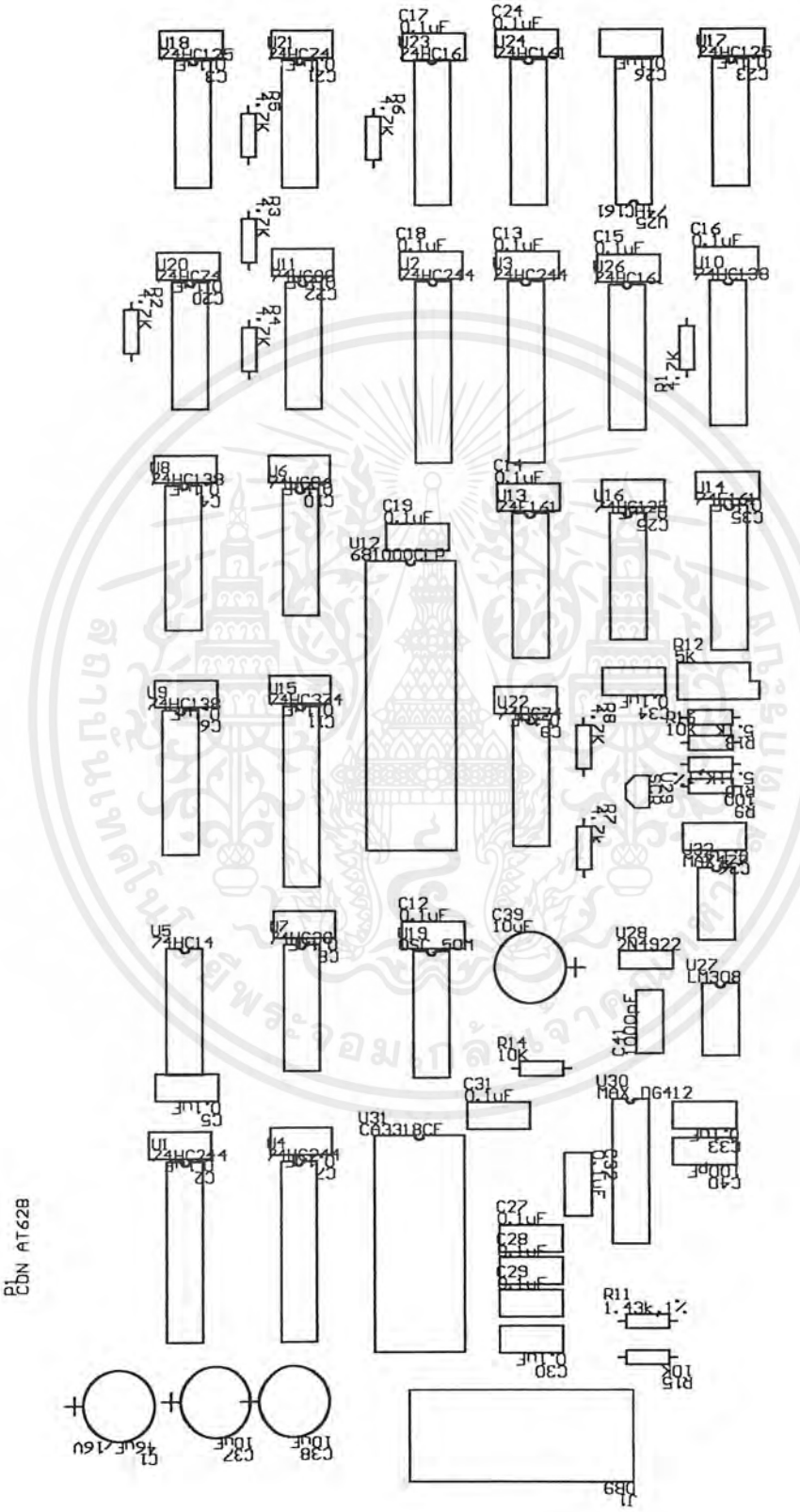
ภาคผนวก ข
แสดงลายปริ้นท์และการวางอุปกรณ์



รูปแสดงลายปริ้นท์เท่าของจริงด้านบน



รูปแสดงลายปริ้นท์เท่าของจริงด้านล่าง



รูปแสดงการวางอุปกรณ์

ภาคผนวก ค

DATA CHEET

intersil

CA3318

August 1997

CMOS Video Speed, 8-Bit, Flash A/D Converter

Features

- CMOS Low Power with SOS Speed (Typ)..... 150mW
- Parallel Conversion Technique
- 15MHz Sampling Rate (Conversion Time).....67ns
- 8-Bit Latched Three-State Output with Overflow Bit
- Accuracy (Typ)..... ±1 LSB
- Single Supply Voltage..... 4V to 7.5V
- 2 Units in Series Allow 9-Bit Output
- 2 Units in Parallel Allow 30MHz Sampling Rate

Applications

- TV Video Digitizing (Industrial/Security/Broadcast)
- High Speed A/D Conversion
- Ultrasound Signature Analysis
- Transient Signal Analysis
- High Energy Physics Research
- General-Purpose Hybrid ADCs
- Optical Character Recognition
- Radar Pulse Analysis
- Motion Signature Analysis
- µP Data Acquisition Systems

Description

The CA3318 is a CMOS parallel (FLASH) analog-to-digital converter designed for applications demanding both low power consumption and high speed digitization.

The CA3318 operates over a wide full scale input voltage range of 4V up to 7.5V with maximum power consumption depending upon the clock frequency selected. When operated from a 5V supply at a clock frequency of 15MHz, the typical power consumption of the CA3318 is 150mW.

The intrinsic high conversion rate makes the CA3318 ideally suited for digitizing high speed signals. The overflow bit makes possible the connection of two or more CA3318s in series to increase the resolution of the conversion system. A series connection of two CA3318s may be used to produce a 9-bit high speed converter. Operation of two CA3318s in parallel doubles the conversion speed (i.e., increases the sampling rate from 15MHz to 30MHz).

256 paralleled auto balanced voltage comparators measure the input voltage with respect to a known reference to produce the parallel bit outputs in the CA3318.

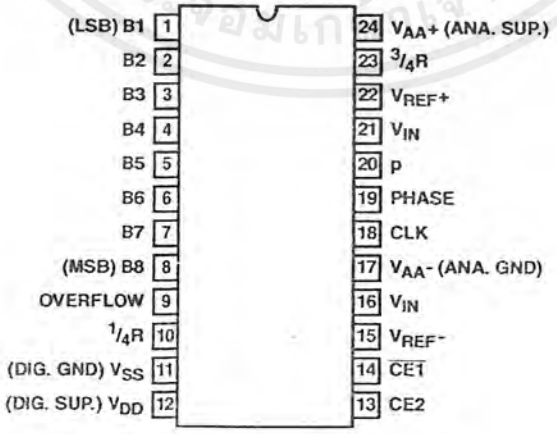
255 comparators are required to quantize all input voltage levels in this 8-bit converter, and the additional comparator is required for the overflow bit.

Ordering Information

PART NUMBER	LINEARITY (INL, DNL)	SAMPLING RATE	TEMP. RANGE (°C)	PACKAGE	PKG. NO.
CA3318CE	±1.5 LSB	15MHz (67ns)	-40 to 85	24 Ld PDIP	E24.6
CA3318CM	±1.5 LSB	15MHz (67ns)	-40 to 85	24 Ld SOIC	M24.3
CA3318CD	±1.5 LSB	15MHz (67ns)	-40 to 85	24 Ld SBDIP	D24.6

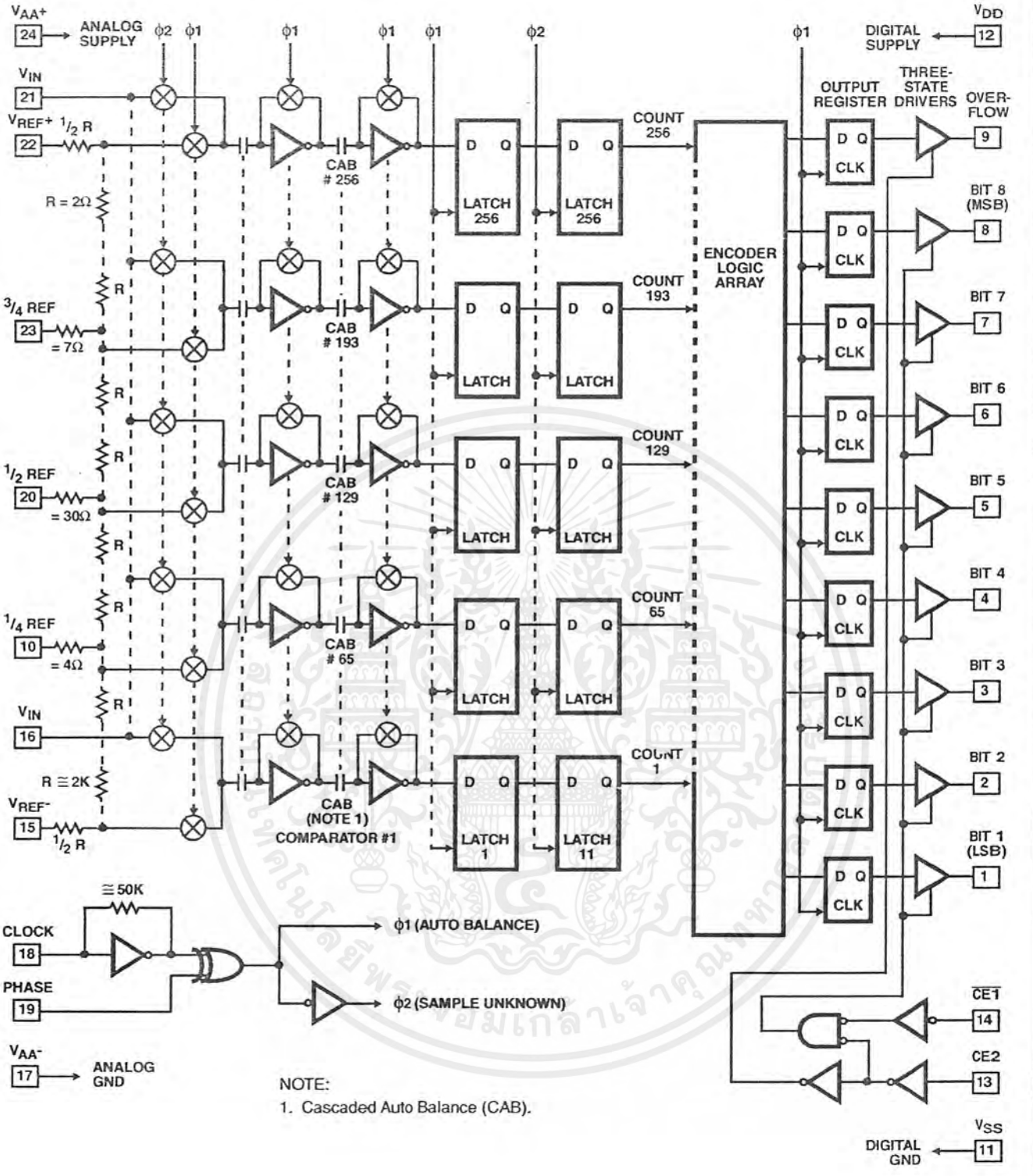
Pinout

CA3318
(PDIP, SBDIP, SOIC)
TOP VIEW



File Number **3103.1**

Functional Block Diagram



NOTE:

1. Cascaded Auto Balance (CAB).

Absolute Maximum Ratings

DC Supply Voltage Range (V_{DD} or V_{AA}+) ..... -0.5V to +8V
(Referenced to V_{SS} or V_{AA}- Terminal, Whichever is More Negative)
Input Voltage Range
CE2 and $\overline{CE1}$ ..... V_{AA}- -0.5V to V_{DD} + 0.5V
Clock, Phase, V_{REF}-, $^1/_2$ Ref ..... V_{AA}- -0.5V to V_{AA}+ + 0.5V
Clock, Phase, V_{REF}-, $^1/_4$ Ref ..... V_{SS}- -0.5V to V_{DD} + 0.5V
V_{IN}, $^3/_4$ REF, V_{REF}+ ..... V_{AA}- -0.5V to V_{AA}- + 7.5V
Output Voltage Range, ... V_{SS} - 0.5V to V_{DD} + 0.5V
Bits 1-8, Overflow (Outputs Off)
DC Input Current ..... ±20mA
Clock, Phase, $\overline{CE1}$, CE2, V_{IN}, Bits 1-8, Overflow

Thermal Information

Thermal Resistance (Typical, Note 1)	θ_{JA} (°C/W)	θ_{JC} (°C/W)
SBDIP Package	60	22
PDIP Package	60	N/A
SOIC Package	75	N/A

Maximum Junction Temperature
Ceramic Package ..... 175°C
Plastic Packages ..... 150°C
Maximum Storage Temperature Range ..... -65°C to 150°C
Maximum Lead Temperature (Soldering 10s) ..... 265°C
(SOIC - Lead Tips Only)

Operating Conditions

Operating Voltage Range (V_{DD} or V_{AA}+) ... 4V (Min) to 7.5V (Max)
Recommended V_{AA}+ Operating Range ..... V_{DD} ±1V
Recommended V_{AA}- Operating Range ..... V_{SS} ±1V
Operating Temperature Range (T_A) ..... -40°C to 85°C

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

At 25°C, V_{AA}+ = V_{DD} = 5V, V_{REF}+ = 6.4V, V_{REF}- = V_{AA}- = V_{SS}, CLK = 15MHz, All Reference Points Adjusted, Unless Otherwise Specified

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
SYSTEM PERFORMANCE					
Resolution		8	-	-	Bits
Integral Linearity Error		-	-	± 1.5	LSB
Differential Linearity Error		-	-	+1, -0.8	LSB
Offset Error, Unadjusted	V_{IN} = V_{REF}- + $^1/_2$ LSB	-0.5	4.5	6.4	LSB
Gain Error Unadjusted	V_{IN} = V_{REF}+ - $^1/_2$ LSB	-1.5	0	1.5	LSB
DYNAMIC CHARACTERISTICS					
Maximum Input Bandwidth	(Note 1) CA3318	2.5	5.0	-	MHz
Maximum Conversion Speed	CLK = Square Wave	15	17	-	MSPS
Signal to Noise Ratio (SNR) $=\frac{RMS\,Signal}{RMS\,Noise}$	f_S = 15MHz, f_{IN} = 100kHz	-	47	-	dB
	f_S = 15MHz, f_{IN} = 4MHz	-	43	-	dB
Signal to Noise Ratio (SINAD) $=\frac{RMS\,Signal}{RMS\,Noise+Distortion}$	f_S = 15MHz, f_{IN} = 100kHz	-	45	-	dB
	f_S = 15MHz, f_{IN} = 4MHz	-	35	-	dB
Total Harmonic Distortion, THD	f_S = 15MHz, f_{IN} = 100kHz	-	-46	-	dBc
	f_S = 15MHz, f_{IN} = 4MHz	-	-36	-	dBc
Effective Number of Bits (ENOB)	f_S = 15MHz, f_{IN} = 100kHz	-	7.2	-	Bits
	f_S = 15MHz, f_{IN} = 4MHz	-	5.5	-	Bits
Differential Gain Error	Unadjusted	-	2	-	%
Differential Phase Error	Unadjusted	-	1	-	%
ANALOG INPUTS					
Full Scale Range, V_{IN} and (V_{REF}+) - (V_{REF}-)	Notes 2, 4	4	-	7	V
Input Capacitance, V_{IN}		-	30	-	pF
Input Current, V_{IN}, (See Text)	V_{IN} = 5V, V_{REF}+ = 5V	-	-	3.5	mA
REFERENCE INPUTS					
Ladder Impedance		270	500	800	Ω

Electrical Specifications At 25°C, $V_{AA}+ = V_{DD} = 5V$, $V_{REF}+ = 6.4V$, $V_{REF}- = V_{AA}- = V_{SS}$, CLK = 15MHz, All Reference Points Adjusted, Unless Otherwise Specified **(Continued)**

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
DIGITAL INPUTS					
Low Level Input Voltage, V_{OL}					
$\overline{CE1}$, CE2	Note 4	-	-	$0.2V_{DD}$	V
Phase, CLK	Note 4	-	-	$0.2V_{AA}$	V
High Level Input Voltage, V_{IN}					
$\overline{CE1}$, CE2	Note 4	$0.7V_{DD}$	-	-	V
Phase, CLK	Note 4	$0.7V_{AA}$	-	-	V
Input Leakage Current, I_I (Except CLK Input)	Note 3	-	±0.2	±5	µA
Input Capacitance, C_I		-	3	-	pF
DIGITAL OUTPUTS					
Output Low (Sink) Current	$V_O = 0.4V$	4	10	-	mA
Output High (Source) Current	$V_O = 4.5V$	-4	-6	-	mA
Three-State Output Off-State Leakage Current, I_{OZ}		-	±0.2	±5	µA
Output Capacitance, C_O		-	4	-	pF
TIMING CHARACTERISTICS					
Auto Balance Time (ϕ1)		33	-	∞	ns
Sample Time (ϕ2)	Note 4	25	-	500	ns
Aperture Delay		-	15	-	ns
Aperture Jitter		-	100	-	ps
Data Valid Time, t_D	Note 4	-	50	65	ns
Data Hold Time, t_H	Note 4	25	40	-	ns
Output Enable Time, t_{EN}		-	18	-	ns
Output Disable Time, t_{DIS}		-	18	-	ns
POWER SUPPLY CHARACTERISTICS					
Device Current ($I_{DD} + I_A$) (Excludes I_{REF})	Continuous Conversion (Note 4)	-	30	60	mA
	Auto Balance (ϕ1)	-	30	60	mA

NOTES:

1. A full scale sine wave input of greater than $f_{CLOCK}/2$ or the specified input bandwidth (whichever is less) may cause an erroneous code. The -3dB bandwidth for frequency response purposes is greater than 30MHz.
2. V_{IN} (Full Scale) or $V_{REF}+$ should not exceed $V_{AA}+ + 1.5V$ for accuracy.
3. The clock input is a CMOS inverter with a 50kΩ feedback resistor and may be AC coupled with $1V_{P-P}$ minimum source.
4. Parameter not tested, but guaranteed by design or characterization.

Timing Waveforms

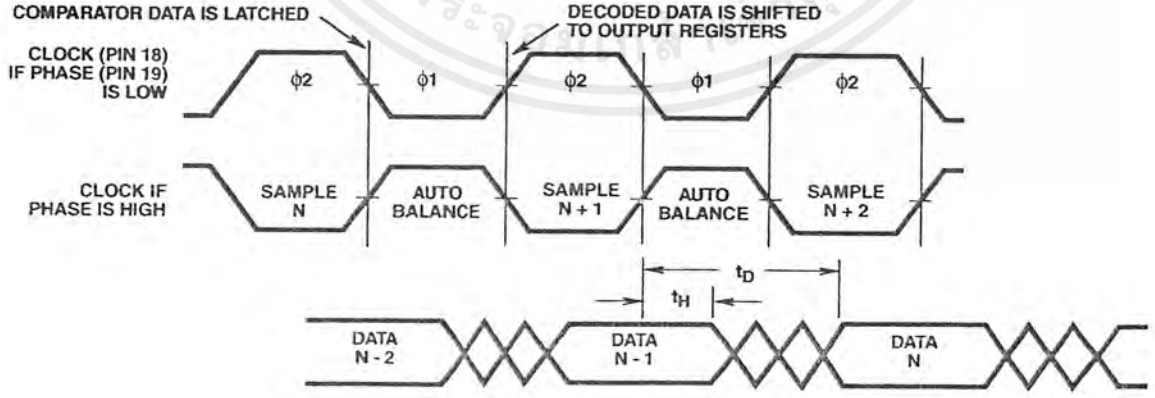


FIGURE 1. INPUT TO OUTPUT TIMING DIAGRAM

FAIRCHILD
SEMICONDUCTOR

April 1988
Revised July 1999

74F161A • 74F163A
Synchronous Presettable Binary Counter

General Description

The 74F161A and 74F163A are high-speed synchronous modulo-16 binary counters. They are synchronously presettable for application in programmable dividers and have two types of Count Enable inputs plus a Terminal Count output for versatility in forming synchronous multi-stage counters. The 74F161A has an asynchronous Master-Reset input that overrides all other inputs and forces the outputs LOW. The 74F163A has a Synchronous Reset input that overrides counting and parallel loading and allows the outputs to be simultaneously reset on the rising edge of the clock. The 74F161A and 74F163A are high-speed versions of the 74F161 and 74F163.

Features

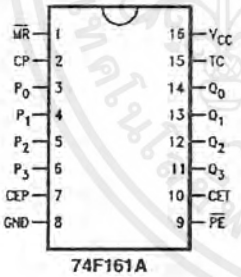
- Synchronous counting and loading
- High-speed synchronous expansion
- Typical count frequency of 120 MHz

Ordering Code:

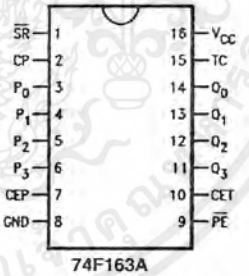
Order Number	Package Number	Package Description
74F161ASC	M16A	16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-012, 0.150 Narrow
74F161ASJ	M16D	16-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
74F161APC	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300 Wide
74F163ASC	M16A	16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-012, 0.150 Narrow
74F163ASJ	M16D	16-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
74F163APC	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300 Wide

Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

Connection Diagrams

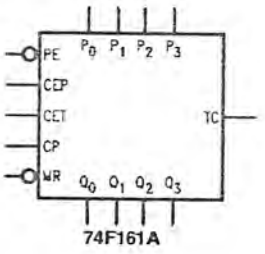


74F161A

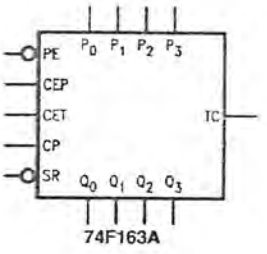


74F163A

Logic Symbols

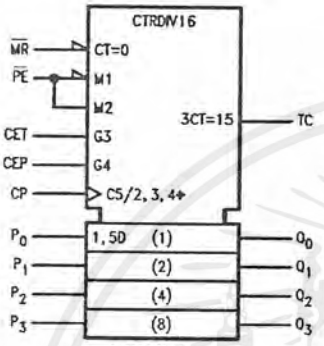


74F161A



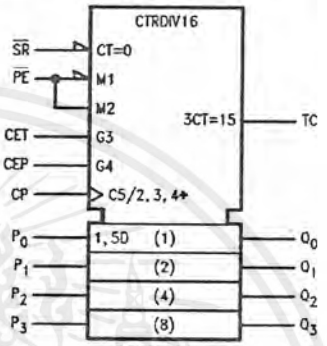
74F163A

IEEE/IEC



74F161A

IEEE/IEC



74F163A

Unit Loading/Fan Out

Pin Names	Description	U.L. HIGH/LOW	Input I_{IH}/I_{IL} Output I_{OH}/I_{OL}
CEP	Count Enable Parallel Input	1.0/1.0	20 µA/–0.6 mA
CET	Count Enable Trickle Input	1.0/2.0	20 µA/–1.2 mA
CP	Clock Pulse Input (Active Rising Edge)	1.0/1.0	20 µA/–0.6 mA
$\overline{MR}$ (74F161A)	Asynchronous Master Reset Input (Active LOW)	1.0/1.0	20 µA/–0.6 mA
$\overline{SR}$ (74F163A)	Synchronous Reset Input (Active LOW)	1.0/2.0	20 µA/–1.2 mA
P_0–P_3	Parallel Data Inputs	1.0/1.0	20 µA/–0.6 mA
$\overline{PE}$	Parallel Enable Input (Active LOW)	1.0/2.0	20 µA/–1.2 mA
Q_0–Q_3	Flip-Flop Outputs	50/33.3	–1 mA/20 mA
TC	Terminal Count Output	50/33.3	–1 mA/20 mA

Functional Description

The 74F161A and 74F163A count in modulo-16 binary sequence. From state 15 (HHHH) they increment to state 0 (LLLL). The clock inputs of all flip-flops are driven in parallel through a clock buffer. Thus all changes of the Q outputs (except due to Master Reset of the 74F161A) occur as a result of, and synchronous with, the LOW-to-HIGH transition of the CP input signal. The circuits have four fundamental modes of operation, in order of precedence: asynchronous reset (74F161A), synchronous reset (74F163A), parallel load, count-up and hold. Five control inputs—Master Reset ($\overline{MR}$, 74F161A), Synchronous Reset ($\overline{SR}$, 74F163A), Parallel Enable ($\overline{PE}$), Count Enable Parallel (CEP) and Count Enable Trickle (CET)—determine the mode of operation, as shown in the Mode Select Table. A LOW signal on $\overline{MR}$ overrides all other inputs and asynchronously forces all outputs LOW. A LOW signal on $\overline{SR}$ overrides counting and parallel loading and allows all outputs to go LOW on the next rising edge of CP. A LOW signal on $\overline{PE}$ overrides counting and allows information on the Parallel Data (P_n) inputs to be loaded into the flip-flops on the next rising edge of CP. With $\overline{PE}$ and $\overline{MR}$ ('F161A) or $\overline{SR}$ (74F163A) HIGH, CEP and CET permit counting when both are HIGH. Conversely, a LOW signal on either CEP or CET inhibits counting.

The 74F161A and 74F163A use D-type edge triggered flip-flops and changing the $\overline{SR}$, $\overline{PE}$, CEP and CET inputs when the CP is in either state does not cause errors, provided that the recommended setup and hold times, with respect to the rising edge of CP, are observed.

The Terminal Count (TC) output is HIGH when CET is HIGH and the counter is in state 15. To implement synchronous multi-stage counters, the TC outputs can be used with the CEP and CET inputs in two different ways. Please refer to the 74F568 data sheet. The TC output is subject to decoding spikes due to internal race conditions and is therefore not recommended for use as a clock or asynchronous reset for flip-flops, counters or registers.

Logic Equations: Count Enable = $CEP \bullet CET \bullet \overline{PE}$

$TC = Q_0 \bullet Q_1 \bullet Q_2 \bullet Q_3 \bullet CET$

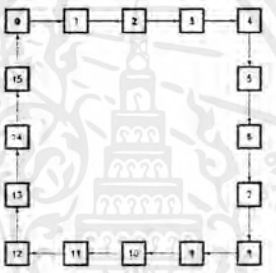
Mode Select Table

$\overline{SR}$ (Note 1)	$\overline{PE}$	CET	CEP	Action on the Rising Clock Edge (⌐)
L	X	X	X	Reset (Clear)
H	L	X	X	Load ($P_n \rightarrow Q_n$)
H	H	H	H	Count (Increment)
H	H	L	X	No Change (Hold)
H	H	X	L	No Change (Hold)

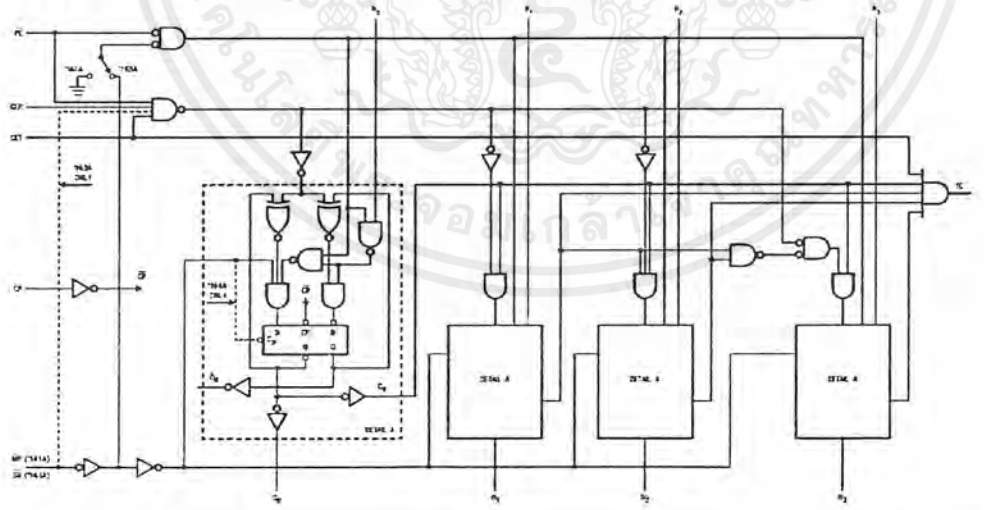
H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

Note 1: For 74F163A only

State Diagram



Block Diagram



Absolute Maximum Ratings(Note 2)

Storage Temperature	−65°C to +150°C
Ambient Temperature under Bias	−55°C to +125°C
Junction Temperature under Bias	−55°C to +150°C
V_{CC} Pin Potential to Ground Pin	−0.5V to +7.0V
Input Voltage (Note 3)	−0.5V to +7.0V
Input Current (Note 3)	−30 mA to +5.0 mA
Voltage Applied to Output in HIGH State (with V_{CC} = 0V)	
Standard Output	−0.5V to V_{CC}
3-STATE Output	−0.5V to +5.5V
Current Applied to Output in LOW State (Max)	twice the rated I_{OL} (mA)
ESD Last Passing Voltage (Min)	4000V

Recommended Operating Conditions

Free Air Ambient Temperature	0°C to +70°C
Supply Voltage	+4.5V to +5.5V

Note 2: Absolute maximum ratings are values beyond which the device may be damaged or have its useful life impaired. Functional operation under these conditions is not implied.

Note 3: Either voltage limit or current limit is sufficient to protect inputs.

DC Electrical Characteristics

Symbol	Parameter		Min	Typ	Max	Units	V_{CC}	Conditions
V_{IH}	Input HIGH Voltage		2.0			V		Recognized as a HIGH Signal
V_{IL}	Input LOW Voltage				0.8	V		Recognized as a LOW Signal
V_{CD}	Input Clamp Diode Voltage				−1.2	V	Min	I_{IN} = −18 mA
V_{OH}	Output HIGH Voltage	10% V_{CC}	2.5			V	Min	
		5% V_{CC}	2.7					
V_{OL}	Output LOW Voltage	10% V_{CC}			0.5	V	Min	I_{OL} = 20 mA
I_{IH}	Input HIGH Current				5.0	µA	Max	V_{IN} = 2.7V
I_{BVI}	Input HIGH Current Breakdown Test				7.0	µA	Max	V_{IN} = 7.0V
I_{CEX}	Output HIGH Leakage Current				50	µA	Max	V_{OUT} = V_{CC}
V_{ID}	Input Leakage Test		4.75			V	0.0	I_{ID} = 1.9 µA All Other Pins Grounded
I_{OD}	Output Leakage Circuit Current				3.75	µA	0.0	V_{IOD} = 150 mV All Other Pins Grounded
I_{IL}	Input LOW Current				−0.6	mA	Max	V_{IN} = 0.5V (CEP, CP, $\overline{MR}$, P_0–P_3)
					−1.2	mA	Max	V_{IN} = 0.5V (CET, $\overline{PE}$, $\overline{SR}$)
I_{OS}	Output Short-Circuit Current		−60		−150	mA	Max	V_{OUT} = 0V
I_{CC}	Power Supply Current			37	55	mA	Max	

National Semiconductor

December 1994

LM108/LM208/LM308 Operational Amplifiers

General Description

The LM108 series are precision operational amplifiers having specifications a factor of ten better than FET amplifiers over a −55°C to +125°C temperature range.

The devices operate with supply voltages from ±2V to ±20V and have sufficient supply rejection to use unregulated supplies. Although the circuit is interchangeable with and uses the same compensation as the LM101A, an alternate compensation scheme can be used to make it particularly insensitive to power supply noise and to make supply bypass capacitors unnecessary.

The low current error of the LM108 series makes possible many designs that are not practical with conventional amplifiers. In fact, it operates from 10 MΩ source resistances, introducing less error than devices like the 709 with 10 kΩ sources. Integrators with drifts less than 500 μV/sec and analog time delays in excess of one hour can be made using capacitors no larger than 1 μF.

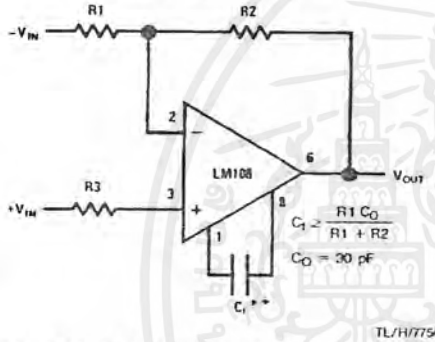
The LM108 is guaranteed from −55°C to +125°C, the LM208 from −25°C to +85°C, and the LM308 from 0°C to +70°C.

Features

- Maximum input bias current of 3.0 nA over temperature
- Offset current less than 400 pA over temperature
- Supply current of only 300 μA, even in saturation
- Guaranteed drift characteristics

Compensation Circuits

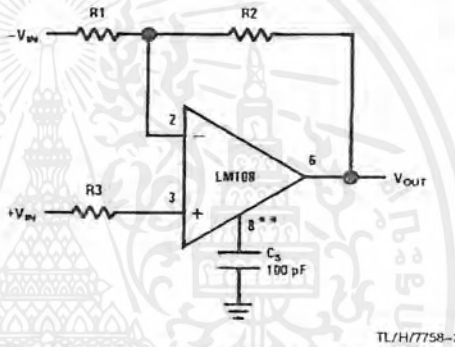
Standard Compensation Circuit



TL/H/7758–1

**Bandwidth and slew rate are proportional to $1/C_1$

Alternate* Frequency Compensation

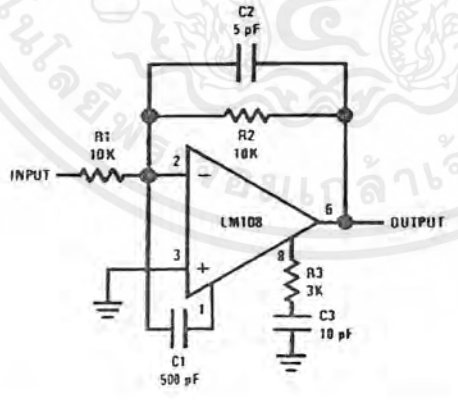


TL/H/7758–2

*Improves rejection of power supply noise by a factor of ten.

**Bandwidth and slew rate are proportional to $1/C_S$

Feedforward Compensation



TL/H/7758–3

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

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.
(Note 5)

	LM108/LM208	LM308
Supply Voltage	±20V	±18V
Power Dissipation (Note 1)	500 mW	500 mW
Differential Input Current (Note 2)	±10 mA	±10 mA
Input Voltage (Note 3)	±15V	±15V
Output Short-Circuit Duration	Continuous	Continuous
Operating Temperature Range (LM108)	−55°C to +125°C	0°C to +70°C
(LM208)	−25°C to +85°C	
Storage Temperature Range	−65°C to +150°C	−65°C to +150°C
Lead Temperature (Soldering, 10 sec)		
DIP	260°C	260°C
H Package Lead Temp		
(Soldering 10 seconds)	300°C	300°C
Soldering Information		
Dual-In-Line Package		
Soldering (10 seconds)	260°C	
Small Outline Package		
Vapor Phase (60 seconds)	215°C	
Infrared (15 seconds)	220°C	
See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" for other methods of soldering surface mount devices.		
ESD Tolerance (Note 6)	2000V	

Electrical Characteristics (Note 4)

Parameter	Condition	LM108/LM208			LM308			Units
		Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	$T_A = 25°C$		0.7	2.0		2.0	7.5	mV
Input Offset Current	$T_A = 25°C$		0.05	0.2		0.2	1	nA
Input Bias Current	$T_A = 25°C$		0.8	2.0		1.5	7	nA
Input Resistance	$T_A = 25°C$	30	70		10	40		MΩ
Supply Current	$T_A = 25°C$		0.3	0.6		0.3	0.8	mA
Large Signal Voltage Gain	$T_A = 25°C$, $V_S = \pm 15V$ $V_{OUT} = \pm 10V$, $R_L \geq 10 k\Omega$	50	300		25	300		V/mV
Input Offset Voltage				3.0			10	mV
Average Temperature Coefficient of Input Offset Voltage			3.0	15		6.0	30	µV/°C
Input Offset Current				0.4			1.5	nA
Average Temperature Coefficient of Input Offset Current			0.5	2.5		2.0	10	pA/°C
Input Bias Current				3.0			10	nA
Supply Current	$T_A = +125°C$		0.15	0.4				mA
Large Signal Voltage Gain	$V_S = \pm 15V$, $V_{OUT} = \pm 10V$ $R_L \geq 10 k\Omega$	25			15			V/mV
Output Voltage Swing	$V_S = \pm 15V$, $R_L = 10 k\Omega$	±13	±14		±13	±14		V

Electrical Characteristics (Note 4) (Continued)

Parameter	Condition	LM108/LM208			LM308			Units
		Min	Typ	Max	Min	Typ	Max	
Input Voltage Range	$V_S = \pm 15V$	±13.5			±14			V
Common Mode Rejection Ratio		85	100		80	100		dB
Supply Voltage Rejection Ratio		80	96		80	96		dB

Note 1: The maximum junction temperature of the LM108 is 150°C, for the LM208, 100°C and for the LM308, 85°C. For operating at elevated temperatures, devices in the H08 package must be derated based on a thermal resistance of 160°C/W, junction to ambient, or 20°C/W, junction to case. The thermal resistance of the dual-in-line package is 100°C/W, junction to ambient.

Note 2: The inputs are shunted with back-to-back diodes for overvoltage protection. Therefore, excessive current will flow if a differential input voltage in excess of 1V is applied between the inputs unless some limiting resistance is used.

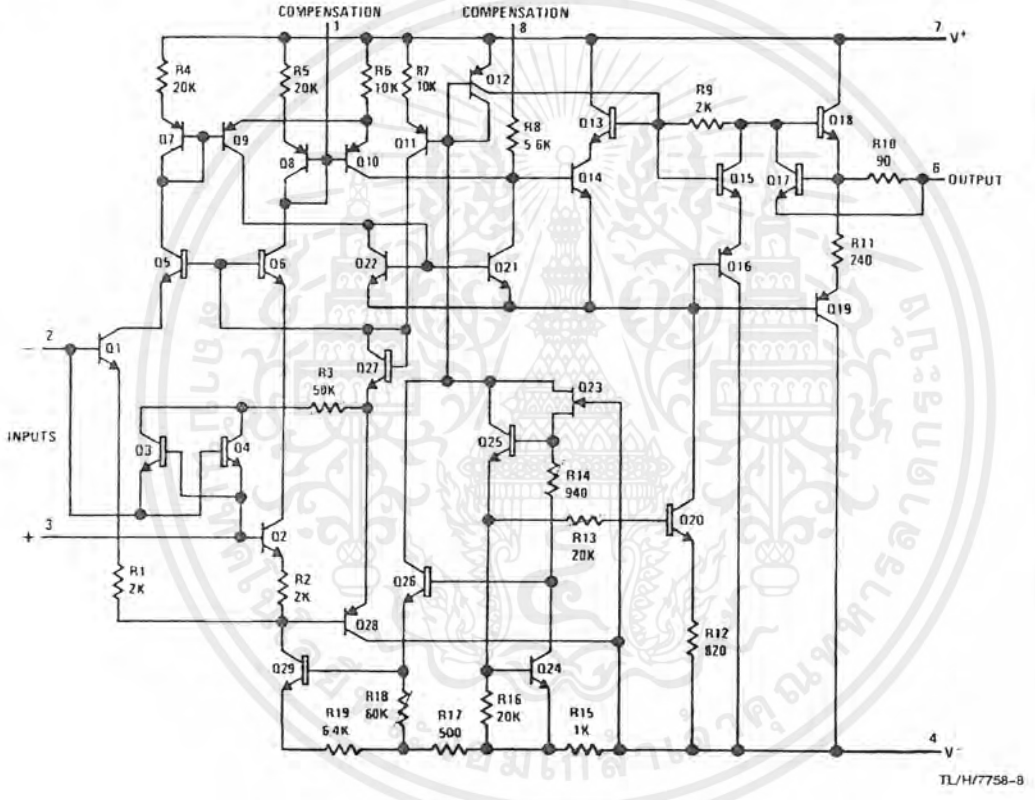
Note 3: For supply voltages less than ±15V, the absolute maximum input voltage is equal to the supply voltage.

Note 4: These specifications apply for $\pm 5V \leq V_S \leq \pm 20V$ and $-55°C \leq T_A \leq +125°C$, unless otherwise specified. With the LM208, however, all temperature specifications are limited to $-25°C \leq T_A \leq 85°C$, and for the LM308 they are limited to $0°C \leq T_A \leq 70°C$.

Note 5: Refer to RETS108X for LM108 military specifications and RETs 108AX for LM108A military specifications.

Note 6: Human body model, 1.5 kΩ in series with 100 pF.

Schematic Diagram



128K x8 bit Low Power CMOS Static RAM

FEATURES

- Process Technology: TFT
- Organization: 128K x8
- Power Supply Voltage: 4.5~5.5V
- Low Data Retention Voltage: 2V(Min)
- Three state output and TTL Compatible
- Package Type: 32-DIP-600, 32-SOP-525, 32-TSOP1-0820F/R

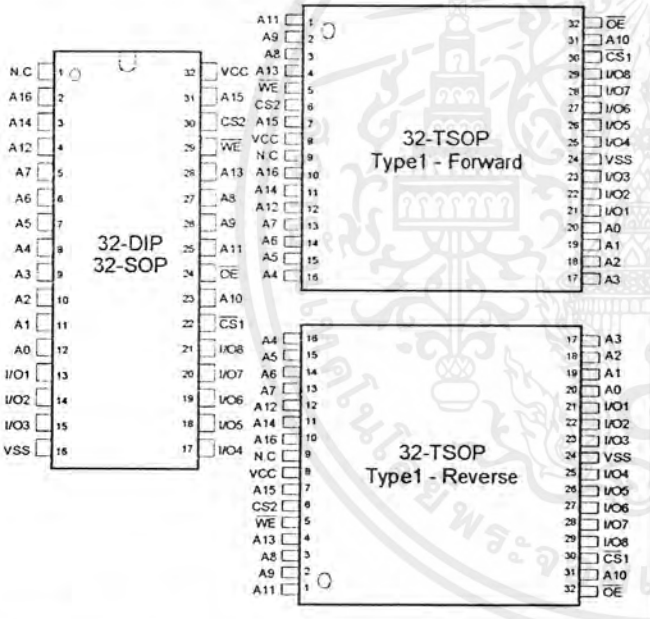
GENERAL DESCRIPTION

The KM681000C families are fabricated by SAMSUNG's advanced CMOS process technology. The families support various operating temperature ranges and have various package types for user flexibility of system design. The families also support low data retention voltage for battery back-up operation with low data retention current.

PRODUCT FAMILY

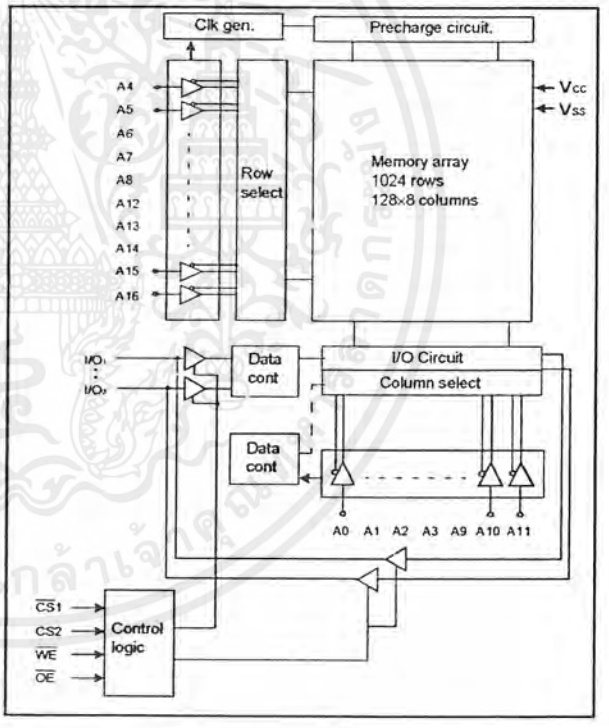
Product Family	Operating Temperature	Vcc Range	Speed	Power Dissipation		PKG Type
				Standby (I_{SB1}, Max)	Operating (I_{CC2}, Max)	
KM681000CL KM681000CL-L	Commercial(0~70°C)	4.5~5.5V	55/70ns	50μA 10μA	60mA	32-DIP, 32-SOP 32-TSOP1-F/R
KM681000CLI KM681000CLI-L	Industrial(-40~85°C)		70ns	50μA 15μA		32-SOP 32-TSOP1-F/R

PIN DESCRIPTION



Name	Function	Name	Function
$\overline{CS_1}$,CS_2	Chip Select Inputs	I/O_1~I/O_8	Data Inputs/Out-
$\overline{OE}$	Output Enable	Vcc	Power
$\overline{WE}$	Write Enable	Vss	Ground
A_0~A_{16}	Address Inputs	N.C	No Connection

FUNCTIONAL BLOCK DIAGRAM



SAMSUNG ELECTRONICS CO., LTD. reserves the right to change products and specifications without notice.

PRODUCT LIST

Commercial Temperature Products(0~70°C)		Industrial Temperature Products(-40~85°C)	
Part Name	Function	Part Name	Function
KM681000CLP-5	32-DIP, 55ns, L-pwr	KM681000CLGI-7	32-SOP, 70ns, L-pwr
KM681000CLP-7	32-DIP, 70ns, L-pwr	KM681000CLGI-7L	32-SOP, 70ns, LL-pwr
KM681000CLP-5L	32-DIP, 55ns, LL-pwr		
KM681000CLP-7L	32-DIP, 70ns, LL-pwr	KM681000CLTI-7L	32-TSOP1-F, 70ns, LL-pwr
KM681000CLG-5	32-SOP, 55ns, L-pwr	KM681000CLRI-7L	32-TSOP1-R, 70ns, LL-pwr
KM681000CLG-7	32-SOP, 70ns, L-pwr		
KM681000CLG-5L	32-SOP, 55ns, LL-pwr		
KM681000CLG-7L	32-SOP, 70ns, LL-pwr		
KM681000CLT-5L	32-TSOP1-F, 55ns, LL-pwr		
KM681000CLT-7L	32-TSOP1-F, 70ns, LL-pwr		
KM681000CLR-5L	32-TSOP1-R, 55ns, LL-pwr		
KM681000CLR-7L	32-TSOP1-R, 70ns, LL-pwr		

FUNCTIONAL DESCRIPTION

$\overline{CS_1}$	CS_2	$\overline{OE}$	$\overline{WE}$	I/O Pin	Mode	Power
H	X[1]	X[1]	X[1]	High-Z	Deselected	Standby
X[1]	L	X[1]	X[1]	High-Z	Deselected	Standby
L	H	H	H	High-Z	Output Disable	Active
L	H	L	H	Dout	Read	Active
L	H	X[1]	L	Din	Write	Active

1. X means don't care(Must be in high or low status.)

ABSOLUTE MAXIMUM RATINGS[1]

Item	Symbol	Ratings	Unit	Remark
Voltage on any pin relative to Vss	V_{IN}, V_{OUT}	-0.5 to 7.0	V	-
Voltage on Vcc supply relative to Vss	V_{CC}	-0.5 to 7.0	V	-
Power Dissipation	P_D	1.0	W	-
Storage temperature	T_{STG}	-65 to 150	°C	-
Operating Temperature	T_A	0 to 70	°C	KM681000CL
		-40 to 85	°C	KM681000CLI
Soldering temperature and time	T_{SOLDER}	260°C, 10sec (Lead Only)	-	-

1. Stresses greater than those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. Functional operation should be restricted to recommended operating condition. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

National Semiconductor

May 1998

LM136-2.5/LM236-2.5/LM336-2.5V Reference Diode

General Description

The LM136-2.5/LM236-2.5 and LM336-2.5 integrated circuits are precision 2.5V shunt regulator diodes. These monolithic IC voltage references operate as a low-temperature-coefficient 2.5V zener with 0.2Ω dynamic impedance. A third terminal on the LM136-2.5 allows the reference voltage and temperature coefficient to be trimmed easily.

The LM136-2.5 series is useful as a precision 2.5V low voltage reference for digital voltmeters, power supplies or op amp circuitry. The 2.5V make it convenient to obtain a stable reference from 5V logic supplies. Further, since the LM136-2.5 operates as a shunt regulator, it can be used as either a positive or negative voltage reference.

The LM136-2.5 is rated for operation over −55°C to +125°C while the LM236-2.5 is rated over a −25°C to +85°C temperature range.

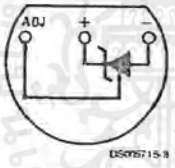
The LM336-2.5 is rated for operation over a 0°C to +70°C temperature range. See the connection diagrams for available packages.

Features

- Low temperature coefficient
- Wide operating current of 400 µA to 10 mA
- 0.2Ω dynamic impedance
- ±1% initial tolerance available
- Guaranteed temperature stability
- Easily trimmed for minimum temperature drift
- Fast turn-on
- Three lead transistor package

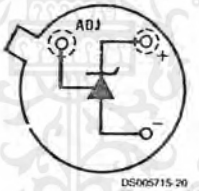
Connection Diagrams

TO-92
Plastic Package



Bottom View
Order Number LM236Z-2.5, LM236AZ-2.5, LM336Z-2.5 or LM336BZ-2.5
See NS Package Number Z03A

TO-46
Metal Can Package



Bottom View
Order Number LM136H-2.5, LM136H-2.5/883, LM236H-2.5, LM136AH-2.5, LM136AH-2.5/883 or LM236AH-2.5
See NS Package Number H03H

Absolute Maximum Ratings (Note 1)

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

Reverse Current	15 mA
Forward Current	10 mA
Storage Temperature	−60°C to +150°C
Operating Temperature Range (Note 2)	
LM136	−55°C to +150°C
LM236	−25°C to +85°C
LM336	0°C to +70°C
Soldering Information	
TO-92 Package (10 sec.)	260°C
TO-46 Package (10 sec.)	300°C
SO Package	
Vapor Phase (60 sec.)	215°C
Infrared (15 sec.)	220°C

See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" (Appendix D) for other methods of soldering surface mount devices.

Electrical Characteristics (Note 3)

Parameter	Conditions	LM136A-2.5/LM236A-2.5 LM136-2.5/LM236-2.5			LM336B-2.5 LM336-2.5			Units
		Min	Typ	Max	Min	Typ	Max	
Reverse Breakdown Voltage	T_A=25°C, I_R=1 mA							
	LM136, LM236, LM336	2.440	2.490	2.540	2.390	2.490	2.590	V
	LM136A, LM236A, LM336B	2.465	2.490	2.515	2.440	2.490	2.540	V
Reverse Breakdown Change With Current	T_A=25°C, 400 µA≤I_R≤10 mA		2.6	6		2.6	10	mV
Reverse Dynamic Impedance	T_A=25°C, I_R=1 mA, f = 100 Hz		0.2	0.6		0.2	1	Ω
Temperature Stability (Note 4)	V_R Adjusted to 2.490V							
	I_R=1 mA, *Figure 2*							
	0°C≤T_A≤70°C (LM336)					1.8	6	mV
	−25°C≤T_A≤+85°C (LM236H, LM236Z)		3.5	9				mV
	−25°C ≤ T_A ≤ +85°C (LM236M)		7.5	18				mV
	−55°C≤T_A≤+125°C (LM136)		12	18				mV
Reverse Breakdown Change With Current	400 µA≤I_R≤10 mA		3	10		3	12	mV
Reverse Dynamic Impedance	I_R=1 mA		0.4	1		0.4	1.4	Ω
Long Term Stability	T_A=25°C ±0.1°C, I_R=1 mA, t = 1000 hrs		20			20		ppm

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. Electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 2: For elevated temperature operation, T_j max is:

LM136	150°C
LM236	125°C
LM336	100°C

Thermal Resistance	TO-92	TO-46	SO-8
θ_{ja} (Junction to Ambient)	180°C/W (0.4" leads) 170°C/W (0.125" lead)	440°C/W	165°C/W
θ_{jc} (Junction to Case)	n/a	80°C/W	n/a

Note 3: Unless otherwise specified, the LM136-2.5 is specified from −55°C ≤ T_A ≤ +125°C, the LM236-2.5 from −25°C ≤ T_A ≤ +85°C and the LM336-2.5 from 0°C ≤ T_A ≤ +70°C.

Note 4: Temperature stability for the LM336 and LM236 family is guaranteed by design. Design limits are guaranteed (but not 100% production tested) over the indicated temperature and supply voltage ranges. These limits are not used to calculate outgoing quality levels. Stability is defined as the maximum change in V_{ref} from 25°C to T_A (min) or T_A (max).

RECOMMENDED DC OPERATING CONDITIONS[1)]

Item	Symbol	Min	Typ	Max	Unit
Supply voltage	V_{CC}	4.5	5.0	5.5	V
Ground	V_{SS}	0	0	0	V
Input high voltage	V_{IH}	2.2	-	V_{CC}+0.5[2)]	V
Input low voltage	V_{IL}	-0.5[3)]	-	0.8	V

Note
1. Commercial Product : T_A=0 to 70°C and Industrial Product : T_A=-40 to 85°C, otherwise specified.
2. Overshoot : V_{CC}+3.0V for≤30ns pulse width.
3. Undershoot : -3.0V for≤30ns pulse width.
4. Overshoot and undershoot are sampled, not 100% tested.

CAPACITANCE[1)] (f=1MHz, T_A=25°C)

Item	Symbol	Test Condition	Min	Max	Unit
Input capacitance	C_{IN}	V_{IN}=0V	-	6	pF
Input/Output capacitance	C_{IO}	V_{IO}=0V	-	8	pF

1. Capacitance is sampled not, 100% tested.

DC AND OPERATING CHARACTERISTICS

Item		Symbol	Test Conditions		Min	Typ	Max	Unit
Input leakage current		I_{LI}	V_{IN}=Vss to Vcc		-1	-	1	μA
Output leakage current		I_{LO}	$\overline{CS_1}$=V_{IH} or CS_2=V_{IL} or $\overline{OE}$=V_{IH} or $\overline{WE}$=V_{IL}, V_{IO}=Vss to Vcc		-1	-	1	μA
Operating power supply current		I_{CC}	I_{IO}=0mA, $\overline{CS_1}$=V_{IL}, CS_2=V_{IH}, V_{IN}=V_{IH} or V_{IL}, Read		-	5	10	mA
Average operating current		I_{CC1}	Cycle time=1μs, 100% duty, I_{IO}=0mA, $\overline{CS_1}$≤0.2V, CS_2≥Vcc-0.2V, V_{IN}≤0.2V or V_{IN}≥Vcc-0.2V	Read	-	2	5	mA
				Write		20	35	
		I_{CC2}	Cycle time=Min, 100% duty, I_{IO}=0mA, $\overline{CS_1}$=V_{IL}, CS_2=V_{IH}, V_{IN}=V_{IL} or V_{IH}		-	45	60	mA
Output low voltage		V_{OL}	I_{OL}=2.1mA		-	-	0.4	V
Output high voltage		V_{OH}	I_{OH}=-1.0mA		2.4	-	-	V
Standby Current(TTL)		I_{SB}	$\overline{CS_1}$=V_{IH}, CS_2=V_{IL}, Other input=V_{IL} or V_{IH}		-	-	3	mA
Standby Current (CMOS)	KM681000CL	I_{SB1}	$\overline{CS_1}$≥Vcc-0.2V, CS_2≥Vcc-0.2V or CS_2≤0.2V Other input =0~Vcc	Low Power	-	1	50	μA
	KM681000CL-L			Low Low Power	-	0.3	10	
	KM681000CLI			Low power	-	1	50	
	KM681000CLI-L			Low Low Power	-	0.3	15	

National Semiconductor

May 1999

LM6161/LM6261/LM6361
High Speed Operational Amplifier

General Description

The LM6161 family of high-speed amplifiers exhibits an excellent speed-power product in delivering 300 V/µs and 50 MHz unity gain stability with only 5 mA of supply current. Further power savings and application convenience are possible by taking advantage of the wide dynamic range in operating supply voltage which extends all the way down to +5V.

These amplifiers are built with National's VIP™ (Vertically Integrated PNP) process which provides fast PNP transistors that are true complements to the already fast NPN devices. This advanced junction-isolated process delivers high speed performance without the need for complex and expensive dielectric isolation.

Features

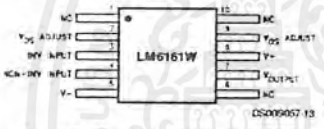
- High slew rate 300 V/µs
- High unity gain freq 50 MHz
- Low supply current 5 mA
- Fast settling 120 ns to 0.1%
- Low differential gain <0.1%
- Low differential phase 0.1°
- Wide supply range 4.75V to 32V
- Stable with unlimited capacitive load
- Well behaved; easy to apply

Applications

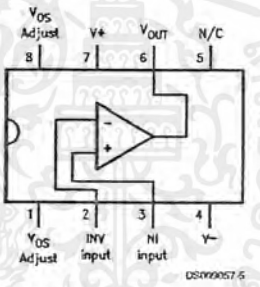
- Video amplifier
- High-frequency filter
- Wide-bandwidth signal conditioning
- Radar
- Sonar

Connection Diagrams

10-Lead Flatpak



See NS Package Number W10A



See NS Package Number J08A, N08E or M08A

Temperature Range			Package	NSC Drawing
Military $-55^{\circ}C \le T_A \le +125^{\circ}C$	Industrial $-25^{\circ}C \le T_A \le +85^{\circ}C$	Commercial $0^{\circ}C \le T_A \le +70^{\circ}C$		
	LM6261N	LM6361N	8-Pin Molded DIP	N08E
LM6161J/883 5962-8962101PA		LM6361J	8-Pin Ceramic DIP	J08A
	LM6261M	LM6361M	8-Pin Molded Surface Mt.	M08A
LM6161WG/883 5962-8962101XA			10-Lead Ceramic SOIC	WG10A
LM6161W/883 5962-8962101HA			10-Pin Ceramic Flatpak	W10A

VIP™ is a trademark of National Semiconductor Corporation

National Semiconductor

December 1994

LM108/LM208/LM308 Operational Amplifiers

General Description

The LM108 series are precision operational amplifiers having specifications a factor of ten better than FET amplifiers over a −55°C to +125°C temperature range.

The devices operate with supply voltages from ±2V to ±20V and have sufficient supply rejection to use unregulated supplies. Although the circuit is interchangeable with and uses the same compensation as the LM101A, an alternate compensation scheme can be used to make it particularly insensitive to power supply noise and to make supply bypass capacitors unnecessary.

The low current error of the LM108 series makes possible many designs that are not practical with conventional amplifiers. In fact, it operates from 10 MΩ source resistances, introducing less error than devices like the 709 with 10 kΩ sources. Integrators with drifts less than 500 μV/sec and analog time delays in excess of one hour can be made using capacitors no larger than 1 μF.

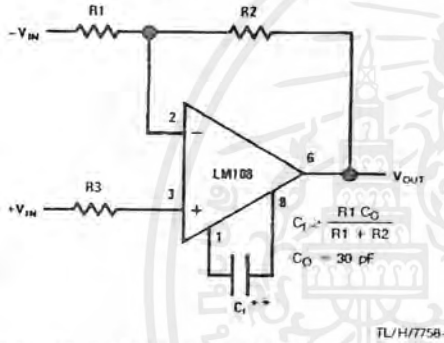
The LM108 is guaranteed from −55°C to +125°C, the LM208 from −25°C to +85°C, and the LM308 from 0°C to +70°C.

Features

- Maximum input bias current of 3.0 nA over temperature
- Offset current less than 400 pA over temperature
- Supply current of only 300 μA, even in saturation
- Guaranteed drift characteristics

Compensation Circuits

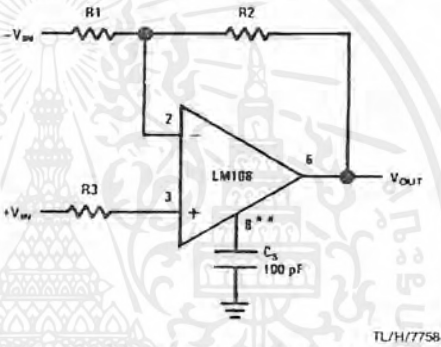
Standard Compensation Circuit



TL/H/7758–1

**Bandwidth and slew rate are proportional to $1/C_f$

Alternate* Frequency Compensation

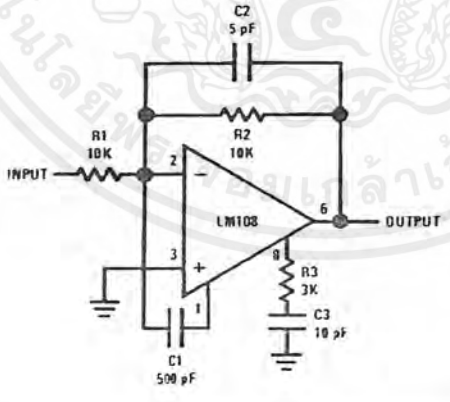


TL/H/7758–2

*Improves rejection of power supply noise by a factor of ten.

**Bandwidth and slew rate are proportional to $1/C_s$

Feedforward Compensation



TL/H/7758–3

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

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดีต้องขอขอบคุณ อาจารย์ ดร. สุทธิชัย นพนาคีพงษ์ ที่ให้คำแนะนำในการทำโปรเจ็ก และขอขอบคุณ พี่เศรษฐกร กาเมือง(ปริญญาโทภาคโทรคมนาคม) ที่ให้คำชี้แนะและให้ยืมอุปกรณ์ในการทดลอง ขอบคุณพี่จรัล ปิ่นทองที่คอยให้คำปรึกษา สุดท้ายขอบคุณเพื่อนๆ ทุกคน ที่คอยให้กำลังใจ

หากมีข้อผิดพลาดประการใดทางผู้จัดทำยินดีรับคำชี้แนะและขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ
นายจักรพงศ์ สำลีว่อง
นายธีรชัย สว่างวารี

บรรณานุกรม

1. ธันวา ศรีประโมง. "**การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม**" พิมพ์ครั้งที่ 3 . กรุงเทพมหานคร: โครงการตำราวิชาการมหาวิทยาลัยเทคโนโลยีมหานคร,2521
2. มนต์ชัย เทียนทอง. "**การโปรแกรมภาษาซี**" พิมพ์ครั้งที่ 1. กรุงเทพมหานคร:คณะครุศาสตร์อุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ,2535
3. สิทธิชัย โภไคยอุดม. "**วงจรขยายสัญญาณโอเปอร์เรชั่นแนล**" พิมพ์ครั้งที่ 1.กรุงเทพมหานคร:คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2523
4. ธานินทร์ ถาวรศาสนวงศ์. "**การอินเตอร์เฟส IBM/ PC**" พิมพ์ครั้งที่1.กรุงเทพมหานคร:โครงการตำราเรียน Physics Center,2528
5. Lonnie C. Ludeman. "**Fundamental of Digital Signal Processing**" New York:Harper & Row,Published,1986
6. Paul M. Embree & Bruce Kimble. "**C Language Algorithms for Digital Signal Processing**"New Jersey:Englewood Cliffs,1991