ดิจิตอลฟิลเตอร์

DIGITAL FILTER

โดย

นายณรินทร์ชัย เพียรสุขมณี

นายวีรยุทธ ยังคล้าย

นายสุทัศน์ ว่องไว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ดิจิตอลฟิลเตอร์

DIGITAL FILTER

โดย

นายณรินทร์ชัย	เพียรสุขมณี	รหัส 35103007
นายวีรยุทธ	ยังคล้าย	รหัส 35103027
นายสุทัศน์	ว่องไว	รหัส 35103035

อาจารย์ที่ปรึกษา

ผู้ช่วยศาสตราจารย์อภินันท์ มัณยานนท์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ดิจิตอลฟิลเตอร์

ผู้จัดทำ

1. นายณรินทร์ชัย เพียรสุขมณี รหัส 35103007
2. นายวีรยุทธ ยังคล้าย รหัส 35103027
3. นายสุทัศน์ ว่องไว รหัส 35103035

.....อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์อภินันท์ มัณยานนท์)

ดิจิตอลฟิลเตอร์

DIGITAL FILTER

โดย นายณรินทร์ชัย เพียรสุขมณี รหัส 35103007

นายวีรยุทธ ยังคล้าย รหัส 35103027

นายสุทัศน์ ว่องไว รหัส 35103035

อาจารย์ที่ปรึกษา ผ.ศ.อภินันท์ มัณยานนท์

บทคัดย่อ

จุดมุ่งหมายของปริญญานิพนธ์ฉบับนี้ คือ ใช้เป็นโปรแกรมสำหรับโปรแกรมเมอร์หรือผู้สนใจในระบบดิจิตอลโปรเซสซิ่ง ได้ใช้ศึกษาและแก้ปัญหาในการประมวลผลสัญญาณดิจิตอล (ดีเอสพี) หรือในงานที่เกี่ยวข้อง ซึ่งในโปรแกรมนี้จะแสดงให้เห็นถึง การฟิลเตอร์แบบต่าง ๆ คือ โลว์พาสส์ฟิลเตอร์ ไฮพาสส์ฟิลเตอร์ ย่านหยุดความถี่หรือแบนด์สตอปฟิลเตอร์ แบนด์พาสส์ฟิลเตอร์ ค่าตอบสนองทางความถี่ ฯลฯ และสามารถที่จะนำไปประยุกต์ใช้ในงานทางด้านวิศวกรรม ซึ่งได้แก่ เรดาร์ อุลตร้าซาวด์ อิมเมจโปร เซสซิ่ง การประมวลผลทางด้านการสื่อสาร และด้านประมวลสัญญาณทางด้านเสียง

ABSTRACT

The purpose of this thesis is to programming language provide a tool so that a program -mer can easily solve a problem involving the manipulation of some type of information. Based on this definition, the purpose of a DSP program is to manipulate a signal (special kind of information) in such a way that the program solves a signal processing problem. This program showed that for example lowpass,highpass,bandstop,bandpass filter and frequency response. Some examples of the applications of DSP to engineering problems are Radar,Ultrasound,Image processing ,communications signal processing and Speech signal processing

สารบัญ

หน้า

บทที่ 1
บทนำ

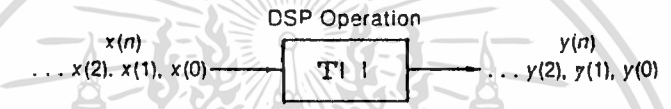
วงจรกรองสัญญาณดิจิตอลมีอยู่ด้วยกัน 2 ประเภท ได้แก่ ประเภทเอฟไออาร์ และประเภทไอไอาร์แต่ ในปริญญานิพนธ์ฉบับนี้จะกล่าวถึงเฉพาะประเภทเอฟไอาร์ ซึ่งมีข้อได้เปรียบวงจรกรองสัญญาณประเภทไอไอาร์อยู่ 2 ประการ ประการแรกเราสามารถจะออกแบบให้มีการตอบสนองความถี่ของเฟสมีความเป็นเชิงเส้นได้โดยแท้จริง และประการที่สอง วงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์จะเสถียรเสมอ เนื่องจากโครงสร้างปราศจากส่วนป้อนกลับ (feedback path) ขณะที่วงจรกรองสัญญาณดิจิตอลประเภทไอไอาร์ จะได้เปรียบตรงที่ว่าด้วยข้อกำหนดเดียวกันจะใช้ลำดับที่ต่ำกว่าวงจรกรองสัญญาณประเภทเอฟไอาร์มาก ซึ่งจะหมายถึงว่า การทำงานของวงจรกรองสัญญาณดิจิตอลประเภทไอไอาร์จะเร็วกว่าและใช้หน่วยความจำที่น้อยกว่ามาก ทำให้ประหยัดค่าใช้จ่าย

การออกแบบวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์ สามารถกระทำได้หลายวิธี แต่ในที่นี้จะพิจารณาเพียงสองวิธี วิธีแรกเรียกว่าวิธีหน้าต่าง (window method) ซึ่งมีหลักการ สำคัญอยู่ตรงที่ว่าเราทราบค่าตอบสนองค่าหนึ่ง $\{h_D(n)\}$ ของวงจรที่เราปรารถนาหรือไม่ และค่าตอบสนองดังกล่าวมีจำนวนที่สิ้นสุดหรือไม่ หากเราทราบค่าและจำนวนที่สิ้นสุดของ $\{h_D(n)\}$ เราก็จะออกแบบได้โดยง่าย หากเราทราบค่าของ $\{h_D(n)\}$ แต่ทว่าจำนวนของ $\{h_D(n)\}$ มีถึงอนันต์เราจำเป็นต้องทอนให้เหลือจำนวนที่สิ้นสุดเพียง N จำนวนโดยที่ N เป็นจำนวนเต็มที่สิ้นสุดเช่นกัน และเราเขียนค่าตอบสนองค่าหนึ่งที่ทอนลงแล้วด้วยสัญญลักษณ์ $\{h_D(n)\}$ การทอนลงนี้เราจำเป็นต้องใช้ฟังก์ชันหน้าต่างช่วย วิธีการนี้เรียกว่า วิธีหน้าต่าง ปกติแล้ว $\{h_D(n)\}$ จะได้มาจากข้อกำหนดของการตอบสนองความถี่ของขนาด $|H_D(e^{j\omega})|$ และค่าตอบสนองเฟสที่เหมาะสมซึ่ง $\{h_D(n)\}$ ก็คืออินเวิสฟูเรียร์ทรานสฟอร์มของ $|H_D(e^{j\omega})|$ นั่นเอง ในทางปฏิบัตินั้นเราอาจหาอินเวิสฟูเรียร์ทรานสฟอร์มในรูปของสูตรสำเร็จได้ จึงเป็นการสะดวก แต่บ่อยครั้งที่เราไม่อาจหาในลักษณะสูตรสำเร็จ ดังนั้นจึงจำเป็นต้องใช้วิธีฟูเรียร์ทรานสฟอร์มไม่ต่อเนื่อง (Discrete Fourier Tranform)ช่วยในการหา $\{h_D(n)\}$ จาก $|H_D(e^{j\omega})|$ เราเรียกวิธีการนี้โดยย่อ ๆ ว่า วิธีดีเอฟที จากคำย่อภาษาอังกฤษ DFT จากคำเต็ม Discrete Fourier Transform ทำให้ได้วิธีที่สองในการออกแบบวงจรกรองสัญญาณดิจิตอลแบบเอฟไอาร์ โดยวิธีนี้เรียกว่า วิธีดีเอฟทีผสมหน้าต่าง

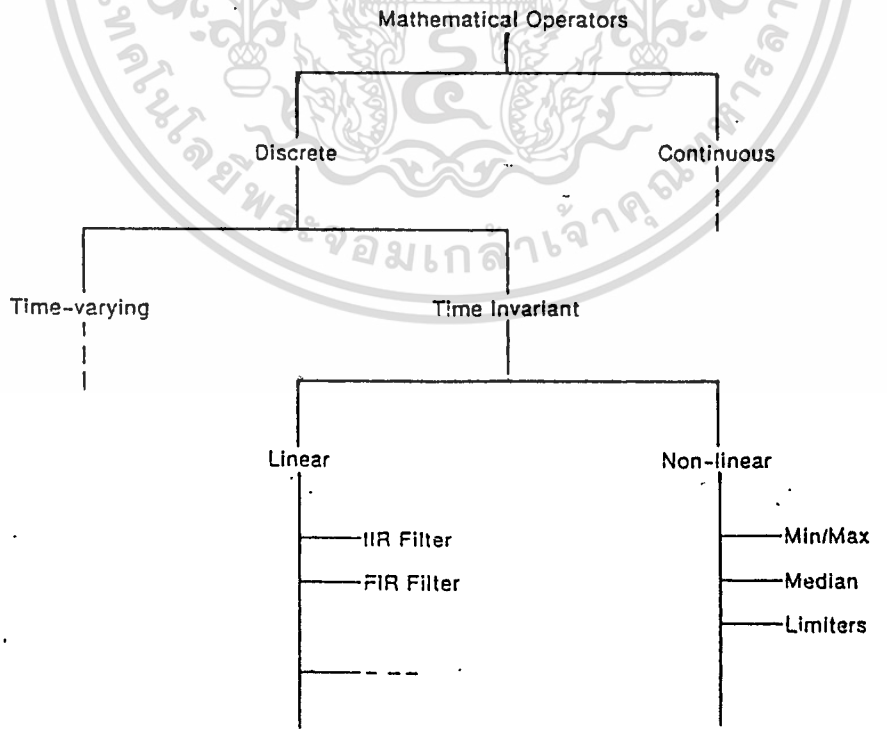
โปรแกรมการแสดงผลของวงจรกรองสัญญาณดิจิตอลประเภทเอฟไออาร์นี้ เขียนขึ้นโดยใช้โปรแกรมภาษาซี ดังนั้น ผู้ที่ต้องการศึกษาควรจะมีความรู้ในด้านภาษาซีพอสมควร และมีความรู้ในด้านคณิตศาสตร์ ซึ่งได้แก่ฟูเรียร์ทรานสฟอร์ม, แซดทรานสฟอร์ม, ดิสครีทฟูเรียร์ทรานสฟอร์ม, ฟาสฟูเรียร์ทรานสฟอร์ม รวมทั้ง ศึกษาวิชาการประมวลผลสัญญาณดิจิตอล หรือดิจิตอล ซิกแนล โปรเซสซิ่ง มาบ้าง ซึ่งในปริญญานิพนธ์นี้สูตรต่าง ๆ ได้สรุปมาเป็นสูตรสำเร็จสำหรับใช้ในการศึกษาเรียบร้อยแล้ว ดังนั้น ผู้ที่ต้องการศึกษารายละเอียดและที่มามากกว่านี้สามารถศึกษาได้จากหนังสือต่าง ๆ ที่อ้างอิงอยู่ท้ายปริญญานิพนธ์นี้

บทที่ 2
พื้นฐานการประมวลผลสัญญาณดิจิตอล

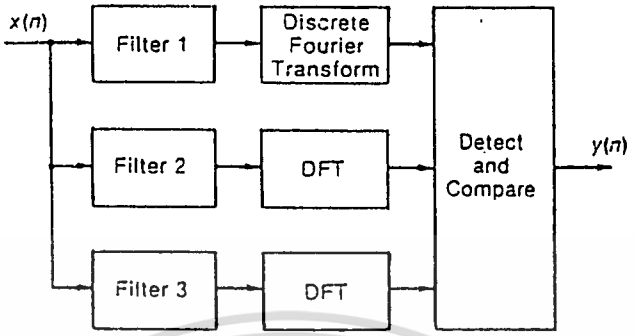
การประมวลผลสัญญาณดิจิตอล จะเริ่มต้นด้วยการแบ่งระดับสัญญาณที่เป็นดิสครีทไทม์ ให้อยู่ในรูปลำดับ(sequence)ของค่าดิจิตอล รูปที่ 2.1 แสดงตัวอย่างของตัวกระทำ(operator)ของการประมวลผลสัญญาณดิจิตอล โดยลำดับอินพุทแทนด้วย x(n), ตัวกระทำแทนด้วย $T\{\ \}$, และ ลำดับเอาท์พุทแทนด้วย y(n) ตัวกระทำแบ่งได้เป็นลิเนียร์และนอนลิเนียร์



รูปที่ 2.1 ตัวกระทำ ดีเอสพี



รูปที่ 2.2 โครงสร้างแบบต้นไม้ของตัวกระทำ



รูปที่ 2.3 ระบบ ดีเอสพี

2.1 ลำดับ (SEQUENCES)

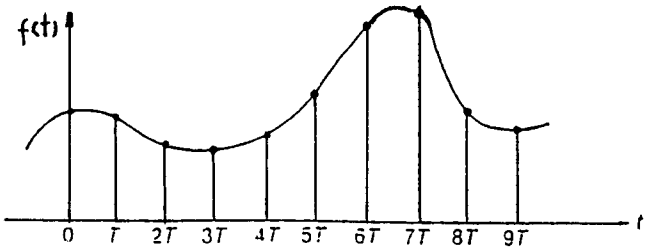
รูปที่ 2.4 แสดงตัวอย่างของฟังก์ชันของเวลาที่ต่อเนื่องซึ่งมีการแซมเปิ้ลเป็นช่วงของคาบเวลา T ผลของการสุ่มที่ได้เรียกว่า ลำดับ ลำดับที่ได้จากสัญญาณที่ต่อเนื่อง จะมีคุณลักษณะที่สำคัญดังนี้

(1) สัญญาณที่สุ่ม จะมีค่าที่จุดดิสครีทที่สิ้นสุดในเวลาที่สุ่ม

(2) สัญญาณที่สุ่มได้แต่ละอัน จะมีช่วงเวลาของแต่ละอันที่สุ่มได้

(3) สัญญาณเป็นการแบ่งระดับ ซึ่งเป็นการกำหนดระดับของแอมปลิจูดของดิสครีท ความเที่ยงตรงของสัญญาณจะขึ้นอยู่กับการกระทำทางคณิตศาสตร์ของคอมพิวเตอร์

เพื่อที่จะเข้าใจลักษณะผลของการสร้างตัวกระทำของ ดีเอสพี คุณลักษณะต่าง ๆ เหล่านี้จะถูกนำมาใช้ในการพิจารณา



รูปที่ 2.4 การแซมปลิ้ง

2.1.1 ฟังก์ชันการสุ่ม(The Sampling Function)

ฟังก์ชันการสุ่ม(sampling function) จะเป็นกุญแจที่นำไปสู่การเดินทางระหว่างโลกของเวลาที่ต่อเนื่อง (continuous time) และเวลาที่ไม่ต่อเนื่อง (discrete time) ซึ่งมันจะมีชื่อเรียกหลาย ๆ ชื่อ ได้แก่ ดิแร็คเดลต้าฟังก์ชัน(Dirac delta function),ชิฟติ้งฟังก์ชัน(Sifting function) ,ซิงกูลาร์ลิตี้ฟังก์ชัน(Singularity function) และ แซมปลิ้งฟังก์ชัน(Sampling function) ซึ่งมันจะมีคุณสมบัติดังต่อไปนี้

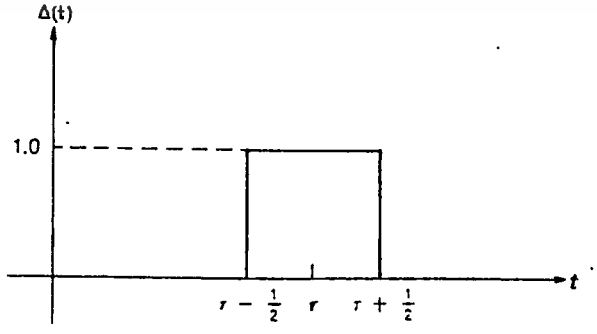
คุณสมบัติที่ 1 $$\int_{-\infty}^{\infty} f(t)\,\delta(t-\tau)dt = f(\tau) \qquad .....(2.1)$$

คุณสมบัติที่ 2 $$\int_{-\infty}^{\infty} \delta(t-\tau)dt = 1 \qquad .....(2.2)$$

จากสมการข้างบน τ แทนค่าของจำนวนจริง

เพื่อจะดูว่าฟังก์ชันเป็นแซมปลิ้งฟังก์ชันในอุดมคติ(Ideal Sampling Function) หรือไม่ ขั้นแรกจะพิจารณาแซมปลิ้งฟังก์ชัน,Δt ดังรูปที่ 2.5 ซึ่งมีความกว้างของพัลซ์เป็น 1 หน่วยและมีแอมปลิจูดเป็น 1 ซึ่งมันจะมีคุณสมบัติตรงกับ (2.2) ของแซมปลิ้งฟังก์ชัน และเมื่อนำ Δt คูณด้วยฟังก์ชันที่ถูกสุ่ม Δt แซมปลิ้งฟังก์ชันจะไม่มีค่าเวลาเดียว แต่จะมีช่วงระหว่าง -1/2 ถึง +1/2 ผลก็คือ คุณสมบัติข้อที่ 2.1จะได้เป็น

$$\int_{-\infty}^{\infty} f(t)\Delta(t-\tau)dt = \int_{\tau-\frac{1}{2}}^{\tau+\frac{1}{2}} f(t)dt \qquad .....(2.3)$$



รูปที่ 2.5 แซมปลิ้งฟังก์ชันจริง

2.1.2 การใช้ฟังก์ชันการสุ่ม(Use of the Sampling Function)

รูปที่ 2.6 แสดงถึงกระบวนการแซมปลิ้งโดยการใช้ แซมปลิ้งฟังก์ชันอุดมคติ (Ideal Sampling Function) ที่ช่วงแซมเปิ้ลของเวลา T ผลลัพธ์ที่ได้สามารถเขียนได้เป็น

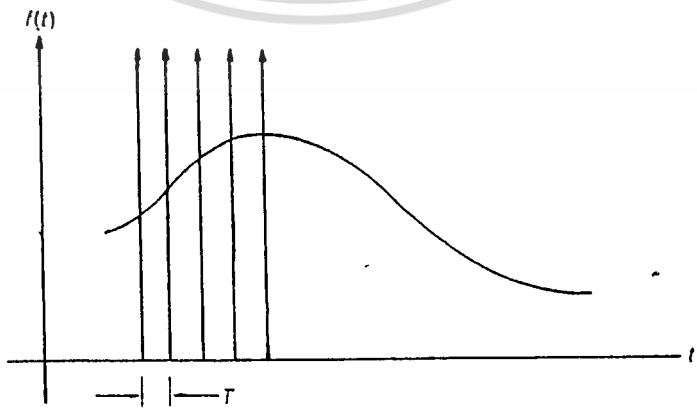
$$Xs(t) = \sum_{n=-\infty}^{\infty} x(t)\delta(t-nT) \qquad \text{.....(2.4)}$$

รูปคลื่นที่ได้จากกระบวนการนี้ ถ้าเป็น แซมปลิ้งฟังก์ชัน ในอุดมคติจะมีขนาดของแอมปลิจูดที่ไม่สิ้นสุดและความกว้างเป็นศูนย์ แล้วนำไปคูณกับรูปคลื่นเวลาที่ต่อเนื่อง

สิ่งที่จะเน้นก็คือ Xs(t) คือ รูปคลื่นเวลาที่ต่อเนื่องที่เกิดจากการแทนที่ของกลุ่มของสัญญาณเวลาที่ต่อเนื่องที่ไม่สิ้นสุด X(t)δ(t-nT) ซึ่งเขียนได้เป็น

$$Xs(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t-nT) \qquad \text{.....(2.5)}$$

เมื่อแซมปลิ้งฟังก์ชันกำหนดให้ตัวคูณไม่เป็นศูนย์ที่ค่า t=nT ในสมการสุดท้ายลำดับ x(nT) จะมีรูปร่างเป็นกลุ่มของจำนวนลำดับ ดังรูปที่ 2.6



รูปที่ 2.6 แซมปลิ้งฟังก์ชัน ในอุดมคติ

2.1.3 สเปคตรัมของสัญญาณสุ่ม(The Spectrum of The Sampled Signal)

เมื่อใช้ทฤษฎีของฟูเรียร์ทรานสฟอร์ม สเปคตรัมความถี่ของรูปคลื่นเวลาที่ต่อเนื่อง x(t) สามารถเขียนได้เป็น

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}\,dt \qquad \text{.....(2.6)}$$

และรูปคลื่นเวลาสามารถเขียนสมการในเทอมของสเปคตรัมได้เป็น

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}\,df \qquad \text{.....(2.7)}$$

เมื่อสมการนี้ถูกต้องสำหรับทุก ๆ ฟังก์ชันที่ต่อเนื่องของเวลา,x(t) จะได้ค่าของ Xs(t) ที่ถูกต้องด้วยคือ

$$Xs(f) = \int_{-\infty}^{\infty} Xs(t)e^{-j2\pi ft}\,dt \qquad \text{.....(2.8)}$$

โดยการแทน Xs(t) ด้วยแซมปลิ้งฟังก์ชันจะได้

$$Xs(t) = \int_{-\infty}^{\infty}\left[\sum_{-\infty}^{\infty} x(t)\delta(t-nT)\right]e^{-j2\pi ft}\,dt \qquad \text{.....(2.9)}$$

ลำดับ(order) ของการซัมเมชั่นและการอินทีเกรตสามารถเชื่อมโยงกับคุณสมบัติที่ 2.1 ของแซมปลิ้งฟังก์ชัน ได้เป็น

$$Xs(f) = \sum_{n=-\infty}^{\infty} x(nT)e^{-j2\pi fnT} \qquad \text{.....(2.10)}$$

สมการนี้คือรูปแบบที่แท้จริงของการนำเสนอในเรื่องของอนุกรมฟูเรียร์(Fourier Series) ของ $Xs(f)$ ซึ่งเป็นพีรีออดิกฟังก์ชัน(periodic function) ของความถี่ที่มีคาบ 1/T ค่าสัมประสิทธิ์(coefficient) ของอนุกรมฟูเรียร์ คือ $X(nT)$ และมันสามารถคำนวณได้จากรูปแบบของการอินทีกรัลต่อไปนี้

$$X(nT) = T\int_{-1/2T}^{1/2T} X_s(f)e^{j2\pi fnT}df \qquad .....(2.11)$$

สองสมการสุดท้ายเป็นคู่อนุกรมฟูเรียร์ ซึ่งใช้ในการคำนวณไทม์ซิกแนล หรือฟรีเควนซี่สเปคตรัม ในเทอมของคู่สมาชิกตรงกันข้าม สังเกตว่าการใช้ที่เมื่อเกิดปัญหา สัญญาณ $Xs(t)$ จะถูกนำออกไปและลำดับ $X(nT)$ จะถูกนำมาใช้แทนที่

2.4.1 ความสัมพันธ์ระหว่างสเปคตรัมของสัญญาณต่อเนื่องและดิสครีทไทม์ (The Relationship of the Spectrums of the Continuous and Discrete Time Signals)

โดยการคำนวณสมการที่ 2.7 ที่ t=nT และทำการปรับค่าผลลัพธ์ให้เท่ากับสมการที่ 2.11 ทางขวามือ จะทำให้ได้ความสัมพันธ์ระหว่างสเปคตรัม 2 อัน คือ

$$x(nT) = \int_{-\infty}^{\infty} X(f)e^{j2\pi fnT}df = T\int_{-1/2T}^{1/2T} X_s(f)e^{j2\pi fnT}df \qquad .....(2.12)$$

สมการที่ 2.7 ทางขวามือสามารถแสดงได้โดยจัดอยู่ในรูปของ

$$x(nT) = \sum_{m=-\infty}^{\infty} \int_{\frac{2m-1}{2T}}^{\frac{2m+1}{2T}} X(f)e^{j2\pi fnT}df \qquad .....(2.13)$$

โดยการเปลี่ยนตัวแปรไปเป็น $\lambda = f - \frac{m}{T}$ (จะได้ $f = \lambda + \frac{m}{T}$ และ $df = d\lambda$) ทำให้ได้

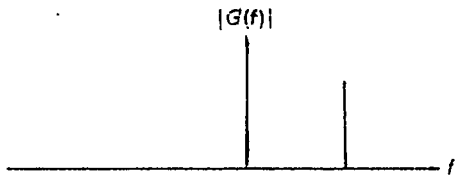
$$x(nT) = \sum_{m=-\infty}^{\infty} \int_{-1/2T}^{1/2T} X(\lambda + \frac{m}{T})e^{j2\pi\lambda nT}e^{j2\pi\frac{m}{T}nT}d\lambda \qquad .....(2.14)$$

ทำการย้ายชัมเมชั่นภายในอินทีกรัล จำไว้ว่าที่ $e^{j2\pi mn}$ (สำหรับทุก ๆ เลขจำนวนเต็ม m และ n) จะมีค่าเท่ากับหนึ่ง และทุก ๆ ตัวภายในอินทีกรัลจะมีส่วนที่เหมือนกับสมการที่ 2.11 ซึ่งจะได้ความสัมพันธ์ดังนี้

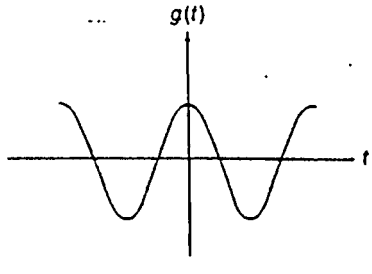
$$X_s(f) = \sum_{m=-\infty}^{\infty} X(f + \frac{m}{T}) \qquad .....(2.15)$$

สมการที่ 2.15 แสดงถึงสเปคตรัมความถี่เวลาที่สุ่ม ซึ่งเท่ากับ การรวมที่สิ้นสุดของการเลื่อน (shift) ของสเปคตรัมความถี่เวลาที่ต่อเนื่องที่ทับอยู่บนแต่ละอัน การเลื่อนของรูปสัญญาณมีค่าเท่ากับแซมเปิ้ลฟรีเควนซี่ 1/T ซึ่งมันเป็นสิ่งที่น่าสนใจที่จะตรวจสอบเงื่อนไขภายใต้สเปคตรัม 2 อันที่เท่ากับอันอื่น ๆ ที่น้อยที่สุดของการจำกัดย่านความถี่ ในกรณีที่ไม่มีสเปคตรัมเป็นส่วประกอบของความถี่ที่มากกว่า 1/2T ในการเริ่มต้นของรูปคลื่นเวลาที่ต่อเนื่องสเปคตรัม 2 อัน จะเท่ากันที่เหนือย่านความถี่ f=-1/2T ถึง f=+1/2T ซึ่งสเปคตรัมเวลาที่สุ่มจะเกิดการซ้ำซึ่งเหมือนกับกลุ่มของคาบแอมปลิจูดสำหรับทุก ๆ ความถี่ ขณะที่สเปคตรัมเวลาที่สุ่มเป็นศูนย์สำหรับทุก ๆ ความถี่ที่อยู่นอกเหนือย่านที่กำหนด

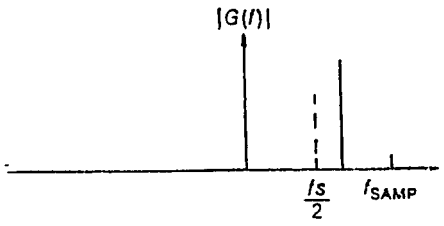
กฎเกณฑ์ของไนควิสท์แซมปลิ้ง(Nyquist sampling) เป็นพื้นฐานที่อยู่บนการเบี่ยงเบนที่ต้องนำเสนอและอ้างถึงรูปคลื่นเวลาที่ต่อเนื่อง เมื่อแซมเปิ้ลที่ความถี่มากกว่า 2 เท่าของความถี่สูงสุดที่ประกอบอยู่ในสเปคตรัม สามารถทำการสร้างใหม่ให้สมบูรณ์ได้จากรูปคลื่นที่สุ่ม ในทางกลับกัน ถ้ารูปคลื่นเวลาที่ต่อเนื่องเป็นแซมเปิ้ลที่ความถี่ต่ำกว่า 2 เท่าของค่าความถี่สูงสุดจะทำให้เกิดปรากฏการณ์ที่เรียกว่า อะเลียซิ่ง(aliasing) ถ้าสัญญาณเวลาที่ต่อเนื่อง ถูกสร้างได้ใหม่จากการที่เกิดอะเลียซิ่ง การบิดเบือน(distortion)จะถูกนำมาพิจารณาผล และขนาดของการบิดเบือนขึ้นอยู่กับขนาดของการอะเลียซิ่ง รูปที่ 2.7(a) ถึง (c) แสดงสเปคตรัมของสัญญาณที่สุ่มที่ปราศจากการอะเลียซิ่ง และการอะเลียซิ่ง รูปที่ 2.8(a) ถึง 2.8(c) แสดงการสร้างรูปคลื่นใหม่ของสัญญาณที่อะเลียซิ่ง



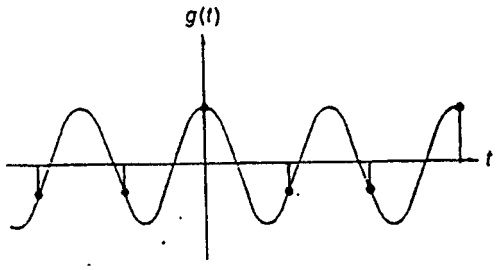
(a) อินพุทที่เป็นสเปคตรัม



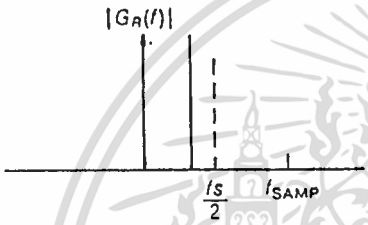
(a) อินพุทที่เป็นสัญญาณเวลาที่ต่อเนื่อง



(b) สเปคตรัมที่สุ่ม

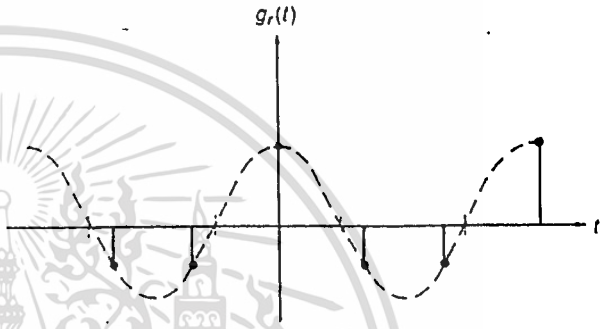


(b) สัญญาณที่สุ่ม



(c) สเปคตรัมที่สร้างขึ้นมาใหม่

รูปที่ 2.7 การอะเลียซิ่ง ในโดเมนความถี่



(c) สัญญาณที่สร้างขึ้นมาใหม่

รูปที่2.8 การอะเลียซิ่ง ในโดเมนเวลา

2.1.5 ลำดับอิมพัลส์ (Impulse Sequence)

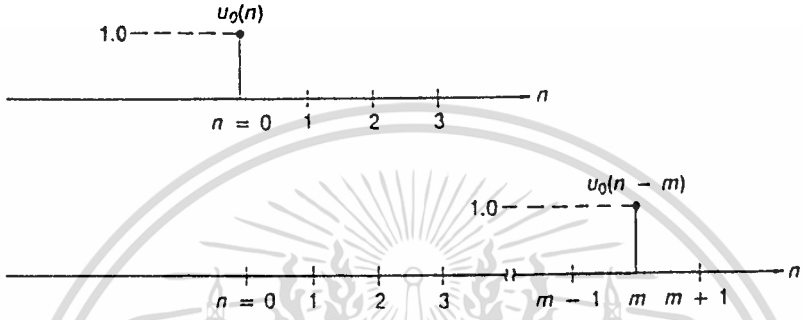
ลำดับอิมพัลส์(impulse sequence) มีลักษณะเป็นเส้นเดียวดังรูป และเป็นลำดับสำคัญ จึงเรียกว่า ลำดับอิมพัลส์ และแทนด้วย $u_0(n)$ ลำดับอิมพัลส์เป็นส่วนประกอบของจำนวนที่จำกัด เป็นค่า 0 ที่ทุก ๆค่า n ยกเว้นที่ n=0. ที่ n=0 ลำดับอิมพัลส์จะมีแอมปลิจูดของแซมเปิ้ลเท่ากับ 1 และลำดับใด ๆ สามารถเลื่อนตำแหน่งโดยการลบค่าคงที่จากค่าดัชนี (index) รูปที่ 2.9 แสดงถึงลำดับอิมพัลส์ และการเลื่อนตำแหน่ง $u_0(n-m)$ ลำดับอิมพัลส์เป็นส่วนที่คู่กับแซมปลิ้งฟังก์ชัน

คุณสมบัติของลำดับอิมพัลส์ สามารถเขียนให้อยู่ในรูปแบบที่คล้ายคลึงกับคุณสมบัติของแซมปลิ้งฟังก์ชัน ดังนี้

คุณสมบัติที่ 1 $$\sum_{n=-\infty}^{\infty} x(n)u_0(n-m) = x(m) \qquad \text{.....(2.16)}$$

คุณสมบัติที่ 2 $$\sum_{n=-\infty}^{\infty} u_0(n-m) = 1 \qquad \text{.....(2.17)}$$

อิมพัลส์ฟังก์ชัน และ คุณสมบัติที่ 2.16 จะใช้ในส่วนของตัวกระทำที่เป็นลิเนียร์และไม่ขึ้นกับเวลา (linear time invariant operator)



รูปที่ 2.9 การเลื่อนของลำดับอิมพัลส์

2.2 ตัวกระทำที่เป็นลิเนียร์และไม่ขึ้นกับเวลา (LINEAR TIME INVARIANT OPERATORS)

ตัวกระทำของ ดีเอสพี ที่ใช้มากที่สุด คือ ลิเนียร์(linear) และไทม์อินวาเรียนท์ (time-invariant) (LTI) ซึ่งเป็นลิเนียร์และไม่ขึ้นกับค่าของเวลา ซึ่งจะมีคุณสมบัติเป็นดังนี้

กำหนดให้ x(n) เป็นลำดับที่สิ้นสุด และ $T\{\ \}$ ตัวกระทำใน n space เขียนได้เป็น

$$y(n) = T\{x(n)\} \qquad \text{.....(2.18)}$$

ถ้า

$$x(n) = ax_1(n) + bx_2(n) \qquad \text{.....(2.19)}$$

เมื่อ a และ b คือค่าคงที่ตามลำดับไปจนถึง n ถ้า $T\{\ \}$ คือ ตัวกระทำที่เป็นลิเนียร์

$$y(n) = aT\{x_1(n)\} + bT\{x_2(n)\} \qquad \text{.....(2.20)}$$

คุณสมบัติของไทม์อินวาเรียนท์ หมายถึงว่าถ้า

$$y(n) = T\{x(n)\} \qquad \text{.....(2.21)}$$

เมื่อมีการเลื่อน จะทำให้มีการตอบสนองที่เหมือนกันคือ

$$y(n-m) = T\{x(n-m)\} \qquad \text{.....(2.22)}$$

ในทางอื่น ๆ คุณสมบัตินี้ถ้า x(n) เป็น พีรีออดิก ด้วยคาบเวลา N เช่น

$$x(n+N) = x(n) \qquad \text{.....(2.23)}$$

เมื่อ $T\{\ \}$ เป็นตัวกระทำไทม์อินวาเรียนท์ ใน n สเปซ(space) จะได้

$$T\{x(n)\} = T\{x(n+N)\} \qquad \text{.....(2.24)}$$

ต่อไปคุณสมบัติ LTI ของตัวกระทำ $T\{\ \}$ จะใช้สำหรับหาที่มาของสมการและวิธีของการคำนวณลำดับอิมพัลส์ สามารถใช้แทน x(n) ในแบบดิฟเฟอเรน

$$x(n) = \sum_{m=-\infty}^{+\infty} x(m)u_0(n-m) \qquad \text{.....(2.25)}$$

ที่เป็นเช่นนี้เพราะว่า

$$u_0(n-m) = \begin{cases} 1, n=m \\ 0, otherwise \end{cases} \qquad \text{.....(2.26)}$$

ลำดับอิมพัลส์ จะแสดงการแซมปลิ้งหรือชิฟติ้งฟังก์ชันบนฟังก์ชัน x(m), โดยการใช้ตัวแปร m ทำการเลือกเฟ้นหรือกลั่นกรองและหาค่า x(n) ที่ต้องการอันเดียว ซึ่งขณะนี้การนำเสนอของ x(n) คือการแทนลงไปในตัวกระทำของสมการที่ 2.18

$$y(n) = T\left\{ \sum_{m=-\infty}^{+\infty} x(m)u_0(n-m) \right\} \qquad \text{.....(2.27)}$$

โดยการนำ $T\{\ \}$ กลับคืนบนฟังก์ชันของ n และการใช้คุณสมบัติ

$$y(n) = \sum_{m=-\infty}^{+\infty} x(m)T\left\{u_0(n-m)\right\} \qquad \text{.....(2.28)}$$

ตัวกระทำทุก ๆ ตัวจะมีกลุ่มของเอาท์พุทซึ่งมันก็คือการตอบสนองเมื่อลำดับอิมพัลส์ ที่ใช้กับอินพุท เราจะแทนการตอบสนองของอิมพัลส์นี้ด้วย h(n) ซึ่งจะได้ว่า

$$h(n) = T\left\{u_0(n)\right\} \qquad \text{.....(2.29)}$$

การตอบสนองของอิมพัลส์ นี้ก็คือ ลำดับ ที่มีความหมายพิเศษสำหรับ $T\{\ \}$ เมื่อมันคือลำดับซึ่งปรากฏที่เอาท์พุทของบล็อค $T\{\ \}$ในรูปที่ 2.1 เมื่อลำดับอิมพัลส์ถูกใช้ที่อินพุท โดยไทม์อินวาเรียนท์ ซึ่งได้ค่าที่ถูกต้องเป็น

$$h(n-m) = T\left\{u_0(n-m)\right\} \qquad \text{.....(2.30)}$$

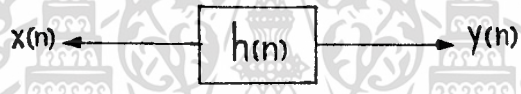
ดังนั้น

$$y(n) = \sum_{m=-\infty}^{+\infty} x(m)h(n-m) \qquad \text{.....(2.31)}$$

สมการที่ 2.31 ที่ y(n)จะมีค่าเท่ากับการคอนโวลูชั่นของ x(n) ด้วยลำดับอิมพัลส์ h(n) โดยการแทน m=n-p ลงในสมการ 2.31 รูปแบบเสมือนจะได้เป็น

$$y(n)=\sum_{p=-\infty}^{+\infty}h(p)x(n-p) \qquad \text{.....(2.32)}$$

ซึ่งต้องจำไว้ว่าที่ m และ n คือ ดัมมี่ วาริเอเบิ้ล(dummy variable) และใช้สำหรับจุดประสงค์ของการซัมเมชั่นเท่านั้น จากสมการต้องทำการหาที่มาเพื่อให้ชัดแจ้งที่การตอบสนอง อิมพัลส์ที่สมบูรณ์ของลักษณะคุณสมบัติของตัวกระทำ $T\{\ \}$ และสามารถใช้เลเบิ้ลบล็อค (label block) แทนตัวกระทำ ในรูปที่ 2.10



รูปที่ 2.10 การตอบสนองอิมพัลส์ที่ใช้แทนตัวกระทำ

2.3 การคอนโวลูชั่น (CONVOLUTION)

จากทฤษฎีที่แสดงถึง ระบบดิสครีทไทม์ลิเนียร์ชิฟอินวาเรียนท์ (discrete time linear shift invariant) สามารถอธิบายได้โดยการตอบสนองยูนิตแซมเปิ้ล (unit sample response) h(n). h(n) จะทำการจัดเตรียมข้อมูลทั้งหมดที่จำเป็นในการหาค่าการตอบสนองไปยังอินพุท

ถ้า x(n) คืออินพุทไปถึงระบบดิสครีทไทม์ลิเนียร์ชิฟอินวาเรียนท์ ที่กำหนดโดย $T\{\ \}$ เมื่อ y(n) เป็น เอาท์พุท จะได้เป็น

$$y(n)=\sum_{k=-\infty}^{+\infty}x(k)h(n-k)=\sum_{k=-\infty}^{+\infty}x(n-k)h(k) \qquad \text{.....(2.33)}$$

เมื่อ $h(n)=T\{\delta(n)\}$

สมการ 2.33 ข้างบนจะใช้บ่อยในการวิเคราะห์ระบบดิสครีทไทม์ลิเนียร์ชิฟอินวาเรียนท์ เพื่อความสะดวกจะใช้ตัวกระทำคอนโวลูชั่น เป็น * ซึ่งจะได้

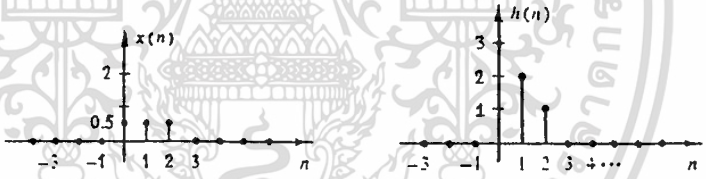
$$x(n)*h(n)\underline{\Delta}\sum_{k=-\infty}^{\infty}x(k)h(n-k)=\sum_{k=-\infty}^{\infty}x(n-k)h(k) \qquad .....(2.34)$$

ด้วยเครื่องหมายนี้เราจะแสดงทฤษฎีข้างต้นด้วยการตอบสนอง y(n) ของระบบลิเนียร์ชิฟอินวาเรียนท์ แสดงได้โดยการตอบสนองอิมพัลส์h(n) ไปยังอินพุท x(n) ซึ่งก็คือ

$$y(n)=x(n)*h(n)=h(n)*x(n) \qquad .....(2.35)$$

จากสมการ (2.35) เอาท์พุทเป็นการคอนโวลูชั่นของอินพุทกับระบบการตอบสนองอิมพัลส์

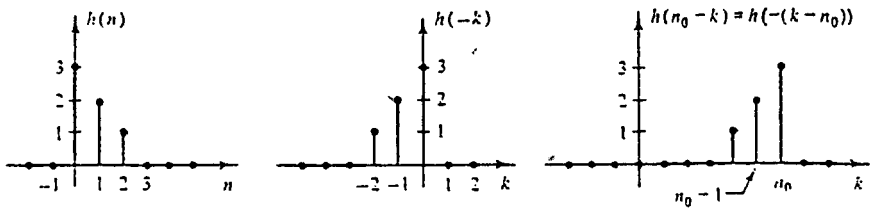
ตัวอย่าง ให้ x(n) เป็นอินพุทของระบบลิเนียร์ชิฟอินวาเรียนท์ ที่แสดงถึงการตอบสนองยูนิตแซมเปิ้ล h(n) และเรียก y(n) ว่าเป็นการตอบสนองเอาท์พุท สำหรับ x(n) และ h(n) ที่ให้ข้างล่าง จงหาเอาท์พุท y(n)



เอาท์พุทของระบบสามารถหาได้โดยการคอนโวลูชั่นอินพุท x(n) ด้วย h(n) ซึ่งจะได้

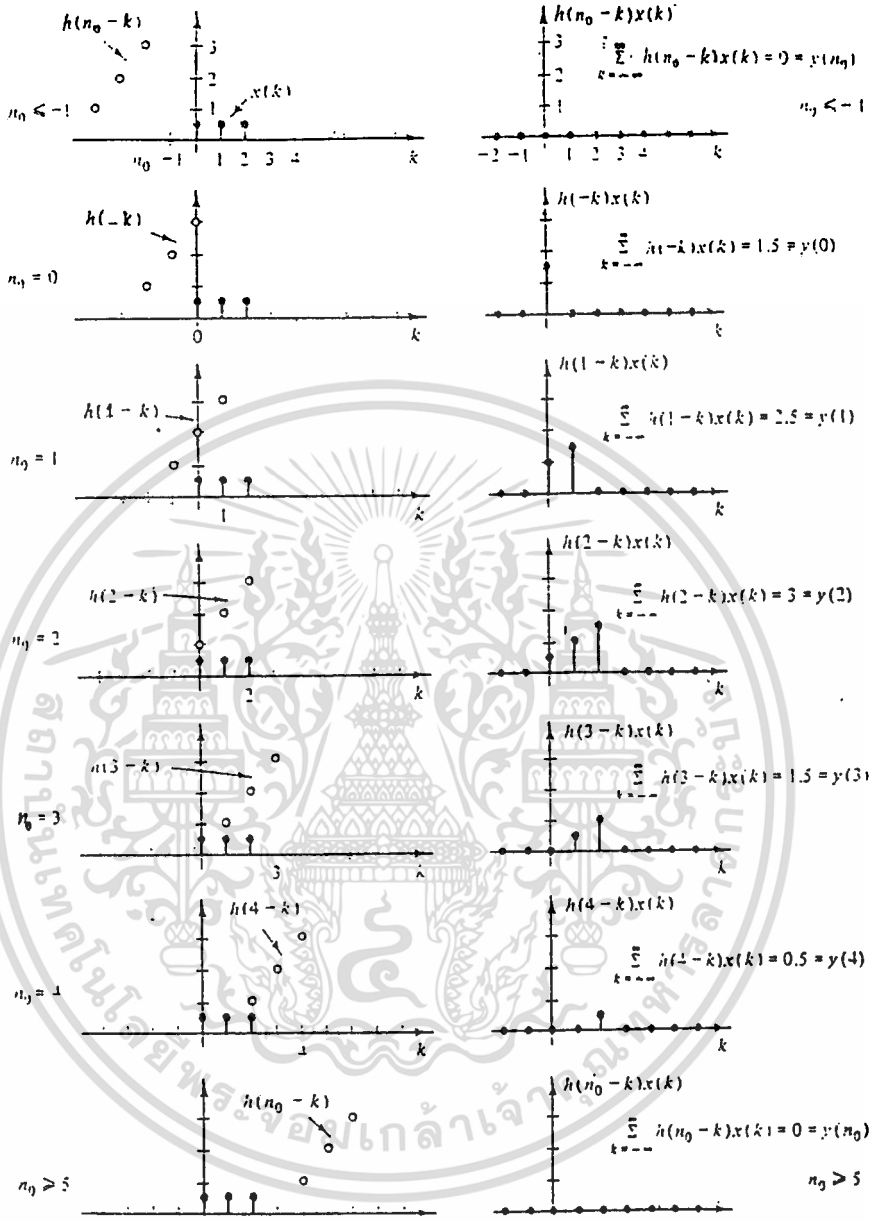
$$y(n)=x(n)*h(n)=\sum_{k=-\infty}^{\infty}x(k)h(n-k) \qquad .....(2.36)$$

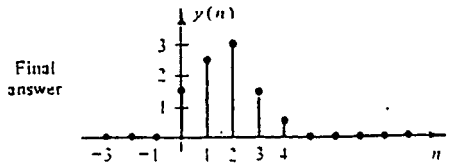
ซึ่งใช้การหาการคอนโวลูชั่นที่เป็นการรวมทางกราฟฟิคคอล (graphical) ดังรูปที่ 2.11



ลำดับเริ่มต้น พับกลับ พับกลับและเลื่อนตำแหน่ง

รูปที่ 2.11 ขั้นตอนในการพับและการเลื่อนตำแหน่งของการคอนโวลูชั่น





รูปที่ 2.12 ตัวอย่างของการคอนโวลูชั่นโดยการใช้วิธีการรวมทางกราฟฟิคคอล

2.4 ลักษณะเชิงเหตุ (CAUSALITY)

ในการพรรณาทางคณิตศาสตร์ของลำดับ และ ตัวกระทำ การที่จะสมมติที่การตอบสนองอิมพัลส์ ของตัวกระทำอาจจะสอดคล้องกับค่าซึ่งปรากฏก่อนหน้านี้ที่ใช้อินพุทใด ๆ ซึ่งเป็นรูปแบบที่นิยมมากที่สุดของสมการและเหมาะสมสำหรับการพัฒนาของทฤษฎีถึงจุดนี้ อย่างไรก็ดี, เพื่อให้เกิดความชัดแจ้งไม่มีระบบทางกายภาพที่จะสามารถสร้างเอาท์พุทได้ ในการตอบสนองถึงอินพุทซึ่งยังไม่มีการใช้ เมื่อตัวกระทำดีเอสพี และ ลำดับ มีพื้นฐานในระบบทางกายภาพ มันใช้ได้มากกว่าถึงการจำกัดการพิจารณาถึงสับเซต(subset) ของตัวกระทำ และ ลำดับซึ่งสามารถมีค่าจริงในโลกของความเป็นจริง

ขั้นแรกในการนำเสนอลำดับที่เหมาะสม คือความรู้ที่ลำดับใด ๆ ดังนั้น มันจึงต้องสมมติที่อีลีเมนท์(element) ใด ๆ ของลำดับในระบบที่เป็นจริงได้ ซึ่งค่าดัชนีเวลามีค่าน้อยกว่าศูนย์มีค่าของศูนย์ ลำดับที่เริ่มที่เวลามากกว่านี้สามารถจะยังคงนำเสนอโดยปราศจากจำนวนของค่าที่เริ่มต้นที่เป็นศูนย์ด้วย อย่างไรก็ดี ค่าที่ถูกต้องที่สุดของลำดับใด ๆ ต้องเป็นค่าของ n ซึ่งมีค่ามากกว่าหรือเท่ากับศูนย์ ลำดับที่อ้างถึงและตัวกระทำ เราเรียกว่า คอซอลลิตี้ (CAUSALITY) เมื่อมันยอมให้ลำดับที่อ้างถึงทั้งหมดมีความชัดแจ้ง ลำดับที่มีค่าแล้วสำหรับเวลาที่สิ้นสุด ดังนั้น ความสัมพันธ์ของการคอนโวลูชั่นสำหรับตัวกระทำคอซอล จะกลายเป็น

$$y(n)=\sum_{m=0}^{\infty}h(m)x(n-m) \qquad \text{.....(2.37)}$$

รูปแบบนี้เป็นไปตามธรรมชาติเมื่อ การตอบสนองอิมพัลส์ คือ ลำดับ และไม่มีค่าสำหรับ m ที่น้อยกว่าศูนย์

2.4.1 สมการดิฟเฟอเรน (DIFFERENCE EQUATIONS)

ดิสครีทไทม์,ลิเนียร์,คอซอล,ตัวกระทำไทม์อินวาเรียนท์ สามารถพรรณาในทฤษฎีโดยใช้สมการดิฟเฟอเรนลำดับ N

$$\sum_{m=0}^{N-1}a_m y(n-m)=\sum_{p=0}^{N-1}b_p x(n-p) \qquad \text{.....(2.38)}$$

เมื่อ x(n) เป็นตัวช่วยสำหรับตัวกระทำ และ y(n) คือผลลัพธ์หรือเอาท์พุทของตัวกระทำ สมการจะยังคงสมบูรณ์โดยทั่วไปถ้าสัมประสิทธิ์ทั้งหมดเป็นนอมอลไลซ์(normalized) โดยค่าของ a_0 ทำให้ได้

$$y(n)+\sum_{m=1}^{N-1}a_m y(n-m)=\sum_{p=0}^{N-1}b_p x(n-p) \qquad \text{.....(2.39)}$$

และรูปแบบเสมือนคือ

$$y(n)=\sum_{p=0}^{N-1}b_p x(n-p)-\sum_{m=1}^{N-1}a_m y(n-m) \qquad \text{.....(2.40)}$$

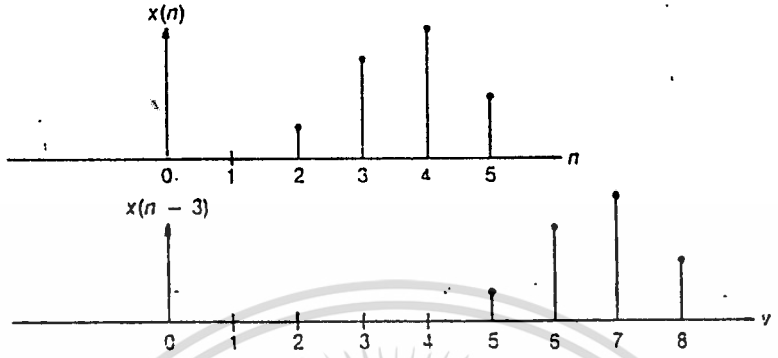
หรือ

$$\begin{aligned} y(n)&=b_0x(n)+b_1x(n-1)+b_2x(n-2)+\ldots \\ &+b_{N-1}x(n-N+1)-a_1y(n-1)-a_2y(n-2) \\ &-\ldots-a_{N-1}y(n-N+1). \end{aligned} \qquad \text{.....(2.41)}$$

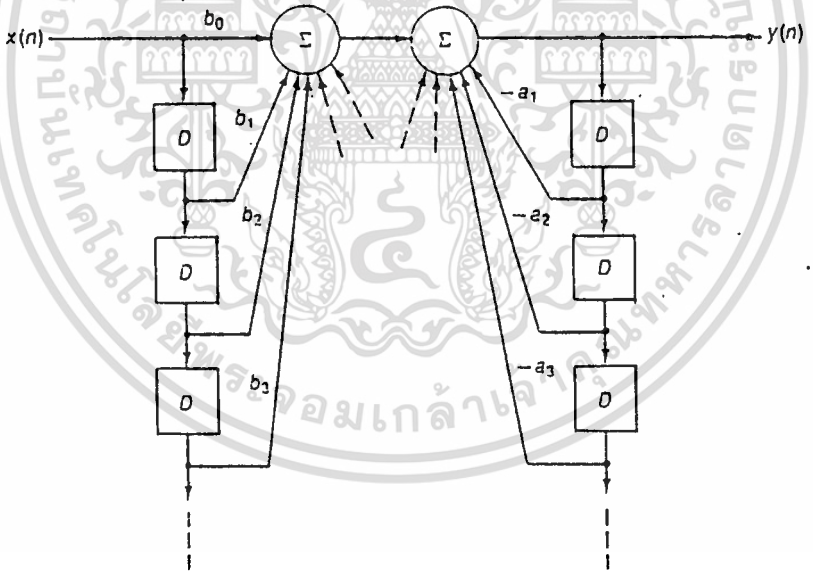
การนำเสนอตัวกระทำ อาจจะต้องการค่า N ที่สูงมาก ๆ และตัวกระทำ ที่ความซับซ้อน N อาจจะมีค่าไปจนถึงอินฟินิตี้ ในทางปฏิบัติค่าของ N จะถูกเก็บไว้ภายในซึ่งจัดการโดยคอมพิวเตอร์และทำการประมาณค่าบ่อยครั้งในของส่วนตัวกระทำ จนถึงทำ N ที่ยอมรับขนาดด้วย ซึ่งตัวมันจะหมายเหตุที่จำนวนของสัมประสิทธิ์ a_m ที่ไม่เป็นศูนย์อาจจะไม่เท่ากับจำนวนสัมประสิทธิ์ b_n ที่ไม่เท่ากับศูนย์

ในสมการที่ 2.39 และ 2.40 เทอม y(n-m) และ x(n-p) เป็นการเลื่อนหรือหน่วงของฟังก์ชัน y(n) และ x(n) ตามลำดับ สำหรับรูปที่ 2.13 แสดงถึงลำดับ x(n) และ x(n-3) ซึ่งเป็นการหน่วงลำดับ ไป 3 แซมเปิ้ลพีเรียด การใช้คุณสมบัติการหน่วงและสมการที่ 2.41 โครงสร้างหรือ โฟลวกราฟ (flow graph) สามารถสร้างได้สำหรับรูปแบบทั่ว ๆ ไป ของตัวกระทำดิสครีทไทม์ โครงสร้างนี้แสดงดังรูปที่ 2.14 บล็อคแต่ละอันเป็นส่วนหน่วง(delay) ด้วย

อัตราการขยายที่เท่ากับหนึ่ง สัมประสิทธิ์แสดงในส่วนต่อไปซึ่งเป็นขาของโฟลวกราฟ ซึ่งมันทั้งหลายใช้อยู่ วงกลมที่ล้อมรอบเครื่องหมายซัมเมชั่น ($\sum$) ที่เป็นส่วนของตัวบวก



รูปที่ 2.13 การเลื่อนของลำดับ



รูปที่ 2.14 โครงสร้างโฟลวกราฟ ของตัวกระทำลิเนียร์

2.4.2 การพรรณาถึงแซด-ทรานสฟอร์มของตัวกระทำที่เป็นลิเนียร์

(The z-transform Description of Linear Operators)

ลิเนียร์ทรานสฟอร์ม ซึ่งใช้ประโยชน์ในการวิเคราะห์ดิสครีทไทม์ ด้วย ลาปลาซ ทรานสฟอร์มไปจนถึงการวิเคราะห์เวลาที่ต่อเนื่อง ทรานสฟอร์ม นี้เรียกว่า แซด ทรานสฟอร์ม และนิยามได้โดย

$$Z\{x(n)\} = \sum_{n=0}^{\infty} x(n)z^{-n} \qquad \text{.....(2.42)}$$

เมื่อ สัญญลักษณ์ $Z\{\ \}$ แทนรูปแบบของแซดทรานสฟอร์ม และ Z ในสมการเป็นจำนวนเชิงซ้อน (complex number) คุณสมบัติที่สำคัญที่สุดอันหนึ่งของแซดทรานสฟอร์ม คือมันจะสัมพันธ์กับการหน่วงเวลาในลำดับ เพื่อที่จะแสดงคุณสมบัตินี้ทำการหาลำดับ x(n) ด้วยแซดทรานสฟอร์ม

$$Z\{x(n)\} = X(z) = \sum_{n=0}^{\infty} x(n)z^{-n} \qquad \text{.....(2.43)}$$

ส่วนการเลื่อนของลำดับ นี้มีแซดทรานสฟอร์มเป็น

$$Z\{x(n-p)\} = \sum_{n=0}^{\infty} x(n-p)z^{-n} \qquad \text{.....(2.44)}$$

เมื่อ p เป็นตัวเลขจำนวนเต็มเสมอและ x(n) เป็น 0 เมื่อ n<0

$$Z\{x(n-p)\} = \sum_{n-p=0}^{\infty} x(n-p)z^{-n} \qquad \text{.....(2.45)}$$

ขณะนี้เราจะทำการแทนค่าตัวแปร m=n-p และจะได้ n=m+p ดังนั้น

$$Z\{x(n-p)\} = \sum_{m=0}^{\infty} x(m) z^{-(m+p)}$$

$$= z^{-p} \sum_{m=0}^{\infty} x(m) z^{-m} \qquad \text{.....(2.46)}$$

แต่เมื่อทำการเปรียบเทียบซัมเมชั่น ในสมการสุดท้ายถึงสมการ 2.35 สำหรับแซดทรานสฟอร์มข่อง x(n) จะได้ว่า

$$Z\{x(n-p)\} = z^{-p} Z\{x(n)\} = z^{-p} X(z) \qquad \text{.....(2.47)}$$

เราสามารถใช้คุณสมบัตินี้ของแซดทรานสฟอร์ม ถึงสมการทั่ว ๆ ไป สำหรับตัวกระทำ LTI จะได้ดังนี้

$$Z\left\{ y(n) + \sum_{p=1}^{\infty} a_p y(n-p) \right\} = z^{-p} Z\left\{ \sum_{q=0}^{\infty} b_q x(n-q) \right\} \qquad \text{.....(2.48)}$$

เมื่อแซดทรานสฟอร์ม เป็นลิเนียร์ทรานสฟอร์มที่ใช้คุณสมบัติการกระจาย และ การจัดหมู่และเราสามารถเขียนสมการใหม่ได้เป็น

$$Z\{y(n)\} + \sum_{p=1}^{\infty} a_p Z\{y(n-p)\} = \sum_{q=0}^{\infty} b_q Z\{x(n-q)\} \qquad \text{.....(2.49)}$$

โดยการใช้คุณสมบัติการเลื่อน ของแซดทรานสฟอร์ม (สมการที่ 2.47)

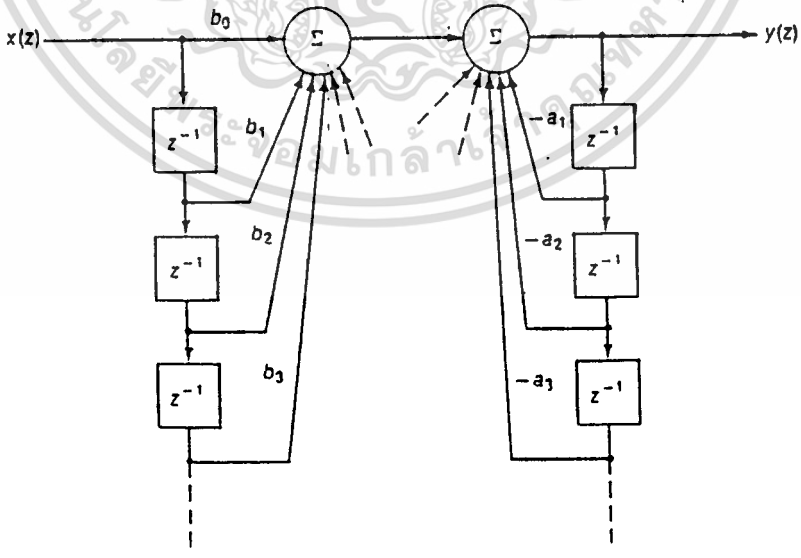
$$Y(z)+\sum_{p=1}^{\infty}a_p z^{-p}Y(z)=\sum_{q=0}^{\infty}b_q z^{-q}X(z)$$

$$Y(z)\left[1+\sum_{p=1}^{\infty}a_p z^{-p}\right]=X(z)\left[\sum_{q=0}^{\infty}b_q z^{-q}\right] \qquad \text{.....(2.50)}$$

สุดท้ายนี้ เราจะจัดการกับสมการ 2.50 ด้วย ทรานสเฟอร์ฟังก์ชัน ใน โดเมนแซดทรานสฟอร์ม

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{q=0}^{\infty}b_q z^{-q}}{1+\sum_{p=1}^{\infty}a_p z^{-p}} \qquad \text{.....(2.51)}$$

โดยการใช้สมการ 2.49 รูปที่ 2.14 สามารถเขียนได้ใหม่ในรูปโดเมนแซดทรานสฟอร์ม และโครงสร้างนี้แสดงในรูปที่ 2.15 โฟลวกราฟ จะเหมือนกันถ้าเข้าใจการคูณด้วย z^{-1} ในโดเมนท์รานสฟอร์ม เสมือนกับการหน่วงของช่วงเวลาการแซมปลิ้งหนึ่งครั้งในโดเมนเวลา



รูปที่ 2.15 โครงสร้างโฟลวกราฟ สำหรับแซดทรานสฟอร์ม ของตัวกระทำ

2.4.3 ทรานสเฟอร์ฟังก์ชันที่เป็นโดเมนความถี่ของตัวกระทำ

(Frequency Domain Transfer Function of an Operator)

ทำการหาฟูเรียร์ทรานสฟอร์มของสมการ 2.30 ทั้งสองข้าง เพื่อให้อยู่ในรูปของ โดเมนความถี่ ผลลัพธ์ที่ได้จะเป็นดังนี้

$$F\{y(n)\} = \sum_{m=0}^{\infty} h(m) F\{x(n-m)\} \qquad \text{.....(2.52)}$$

โดยการใช้คุณสมบัติข้อที่หนึ่งของฟูเรียร์ทรานสฟอร์ม

$$F\{x(n-m)\} = e^{-j2\pi fm} F\{x(n)\} \qquad \text{.....(2.53)}$$

จากสมการที่ 2.48 จะทำให้ได้

$$Y(f) = \sum_{m=0}^{\infty} h(m) e^{-j2\pi fm} X(f) \qquad \text{.....(2.54)}$$

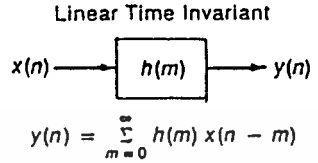
หรือทำหารทั้งสองข้างด้วย $X(f)$

$$\frac{Y(f)}{X(f)} = \sum_{m=0}^{\infty} h(m) e^{-j2\pi fm} \qquad \text{.....(2.55)}$$

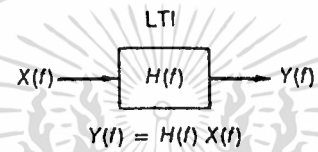
ซึ่งจะจดจำได้ง่ายในรูปฟูเรียร์ทรานสฟอร์ม ของอนุกรม h(m) ซึ่งเขียนได้ใหม่เป็น

$$\frac{Y(f)}{X(f)} = H(f) = F\{h(m)\} \qquad \text{.....(2.56)}$$

รูปที่ 2.16 แสดงบล็อคไดอะแกรมโดเมนเวลา ของสมการ 2.56 และรูปที่ 2.17 แสดงบล็อคไดอะแกรมฟูเรียร์ทรานสฟอร์ม(หรือโดเมนความถี่) และสมการโดเมนความถี่ พรรณาถึงตัวกระทำลิเนียร์ จะใช้บ่อยมากที่สุดซึ่งมันจะแสดงแอมปลิจูดและพลอตมุมเฟส ที่ฟังก์ชันของตัวแปร f (บางครั้งจะนอมอลไลซ์ ด้วยอัตราแซมปลิ้ง ,1/T)



รูปที่ 2.16 บล็อคไดอะแกรมที่เป็นโดเมนเวลา ของระบบ LTI



รูปที่ 2.17 บล็อคไดอะแกรมที่เป็นโดเมนความถี่ ของระบบ LTI

2.4.4 ความสัมพันธ์ของแซด-ทรานสฟอร์มต่อการตอบสนองความถี่ (Relationship of the z -transform to the Frequency Response)

กลับไปดูคู่ฟูเรียร์ทรานสฟอร์ม

$$X_s(f) = \sum_{n=-\infty}^{\infty} x(nT) e^{-j2\pi fnT} \qquad \text{.....(2.57)}$$

และ $$x(nT) = \int_{-\infty}^{\infty} X_s(f) e^{j2\pi fnT} df \qquad \text{.....(2.58)}$$

เพื่อที่จะ สังเกตได้ง่ายเราจะนอมอลไลซ์ค่าของ T คาบเวลาของรูปคลื่นแซมปลิ้ง ให้มีค่าเท่ากับ 1 ขณะนี้เปรียบเทียบสมการที่ 2.57 จะได้สมการสำหรับแซดทรานสฟอร์ม ของ x(n) จะได้ดังนี้

$$X(z)=\sum_{n=0}^{\infty}x(n)z^{-n} \qquad .....(2.59)$$

สมการที่ 2.57 และ 2.59 มีค่าเท่ากันสำหรับลำดับ x(n) ซึ่งเป็นคอซอล (เช่น x(n)=0 สำหรับทุก ๆ n<0) ถ้า z เป็นกลุ่มของ

$$z=e^{j2\pi f} \qquad .....(2.60)$$

พล็อต โลกัส ของค่าสำหรับ z ในคอมเพล็ก เพลน(comlex plane) อธิบายได้โดยสมการที่ 2.60 ที่แสดงในรูปที่ 2.18 ค่าที่พลอต ได้จะเป็นวงกลมรัศมีหนึ่งหน่วย ดังนั้นแซดทรานสฟอร์มของลำดับคอซอล ,x(n) เมื่อหาค่าบนวงกลมหนึ่งหน่วยในคอมเพล็ก เพลน จึงเปรียบเสมือนกับการนำเสนอลำดับในรูปของโดเมนความถี่ ซึ่งเป็นคุณสมบัติหนึ่งของแซดทรานสฟอร์ม ซึ่งทำให้มันใช้ประโยชน์ได้มากสำหรับการวิเคราะห์สัญญาณดิสครีท

รูปที่ 2.18 วงกลมหนึ่งหน่วยใน z-plane

สรุปเกี่ยวกับตัวกระทำลิเนียร์

ที่กล่าวมาแล้วทั้งหมดเกี่ยวกับตัวกระทำลิเนียร์ สามารถสรุปได้ดังนี้

(1) การตอบสนองอิมพัลส์ h(m)

สมการคือ $$y(n)=\sum_{m=0}^{\infty}h(m)x(n-m)$$

(2) ทรานสเฟอร์ฟังก์ชันในโดเมนความถี่ H(f)

สมการคือ $$Y(f)=H(f)X(f)$$

(3) สมการดิฟเฟอเรน

สมการคือ $$y(n)=\sum_{q=0}^{\infty}b_q x(n-q)-\sum_{p=1}^{\infty}a_p y(n-p)$$

(4) แซด ทรานสฟอร์ม ทรานสเฟอร์ฟังก์ชัน H(z)

สมการคือ $$Y(z)=H(z)X(z)$$

ในแต่ละสมการที่นำเสนอจะใช้ประโยชน์ในการศึกษาระบบดิสครีทไทม์ เพื่อที่จะเข้าใจความสัมพันธ์ซึ่งเป็นกุญแจที่สำคัญอันหนึ่งในการที่จะออกแบบระบบ ดีเอสพี ได้อย่างมีประสิทธิภาพ

2.5 ดิจิตอลฟิลเตอร์(DIGITAL FILTERS)

ตัวกระทำที่เป็นลิเนียร์ซึ่งมีการแสดงและวิเคราะห์ในส่วนก่อนหน้านี้ คือ ดิจิตอลฟิลเตอร์ แนวคิดหลักของการฟิลเตอร์เป็นแบบอะนาลอกระหว่างการแสดงทางกายภาพของซิฟเตอร์ไปเป็นการแสดงของตัวกระทำที่เป็นลิเนียร์บนลำดับ เมื่อตัวกระทำเป็นรูปแบบในโดเมนของความถี่ ฟิลเตอร์จะยอมให้ความถี่อินพุทที่แน่นอนผ่านไปโดยไม่มีการเปลี่ยนแปลงไปออกยังเอาท์พุทและ จะทำการกันความถี่อื่น ๆไม่ให้ออกไปยังเอาท์พุท โดยปกติแล้วการแสดงใด ๆ จะมีผลการตอบสนองในโดเมนของเวลา รูปแบบของตัวกระทำที่เป็นลิเนียร์นี้จะเปิดกว้างในการวิเคราะห์และทำให้เข้าใจการแสดงของระบบดิจิตอลเพิ่มขึ้น

ดิจิตอลฟิลเตอร์แบ่งได้เป็น 2 รูปแบบ สำหรับสมการดิฟเฟอเรนเชี่ยลของตัวกระทำทั่ว ๆ ไป จะเขียนได้เป็น

$$y(n)=\sum_{q=0}^{Q-1} b_q x(n-q)-\sum_{p=1}^{p-1} a_p y(n-P) \qquad \text{.....(2.61)}$$

สังเกตว่าที่เครื่องหมายซัมอินฟินิตี้ จะถูกแทนด้วยเครื่องหมายซัมฟินิตี้ ซึ่งมีความจำเป็นในลำดับของการฟิลเตอร์ที่เป็นจริงในทางกายภาพ

ชนิดแรกของดิจิตอลฟิลเตอร์จะมี a_p เท่ากับศูนย์สำหรับทุก ๆ ค่า p ฟิลเตอร์ชนิดนี้เรียกว่า Finite Impulse Responce(FIR) Filter ซึ่งมีการตอบสนองของอิมพัลส์ที่มีจำนวนสิ้นสุด ฟิลเตอร์ชนิดนี้อาจเรียกอีกอย่างหนึ่งได้ว่า moving average(or MA) ซึ่งเอาท์พุทจะเป็นค่าน้ำหนักเฉลี่ยของค่าอินพุท

$$y(n)=\sum_{q=0}^{Q-1} b_q x(n-q) \qquad \text{.....(2.62)}$$

สมการนี้เป็นหน้าต่างของน้ำหนัก (b_q) ซึ่งทำให้ได้ค่า Q ใหม่สุดของ x(n) และทำการรวมค่าเหล่านี้เพื่อสร้างออกที่เอาท์พุท

ชนิดที่สองของดิจิตอลฟิลเตอร์ก็คือ Infinite Impulse Response(IIR) Filter ฟิลเตอร์ชนิดนี้เรียกอีกอย่างหนึ่งว่า autoregressive (AR) ฟิลเตอร์ และนิยมเขียนในรูปแบบ ARMA ฟิลเตอร์ ในกรณี AR b_q ทั้งหมดสำหรับ q=1 ถึง Q-1 ถูกปรับให้เป็นศูนย์

$$y(n) = x(n) - \sum_{p=1}^{p-1} a_p y(n-p) \qquad \text{.....(2.63)}$$

สมการ 2.63 ข้างบนเป็นสมการของ ARMA ฟิลเตอร์

2.5.1 วงจรกรองแบบเอฟไออาร์(FIR Filters)

สมการทั่วไปของวงจรกรองแบบเอฟไออาร์ คือ

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q) \qquad \text{.....(2.64)}$$

ทำการเปรียบเทียบสมการนี้กับความสัมพันธ์ของการคอนโวลูชั่นของตัวกระทำที่เป็นลิเนียร์

$$y(n) = \sum_{m=0}^{\infty} h(m) x(n-m) \qquad \text{.....(2.65)}$$

จะพบว่าที่สัมประสิทธิ์ในวงจรกรองเอฟไออาร์ จะเหมือนกับส่วนของลำดับการตอบสนองอิมพัลส์ ถ้าการตอบสนองอิมพัลส์นี้มีความยาวที่สิ้นสุด

$$b_q = h(q) \qquad \text{for q=0,1,2,3,.....,Q-1} \qquad \text{.....(2.66)}$$

หมายถึงว่าถ้าอันหนึ่งเป็นลำดับการตอบสนองอิมพัลส์ของตัวกระทำ ที่เป็นลิเนียร์กับการตอบสนองที่สิ้นสุด สามารถที่จะเขียนสัมประสิทธิ์ของวงจรกรองเอฟไออาร์ได้โดยทันที อย่างไรก็ดี สิ่งที่จะกล่าวถึงในส่วนนี้ก็คือ ทฤษฎีการฟิลเตอร์ ซึ่งขั้นแรกจะดูตัวกระทำที่เป็นลิเนียร์จากจุด

โดเมนของความถี่ อันหนึ่งที่ต้องการมากที่สุดในการตอบสนองโดเมนของความถี่และกล่าวถึงเพื่อหาค่าสัมประสิทธิ์ของวงจรกรองเอฟไอาร์

ซึ่งจำนวนของวิธีในการหาค่าสัมประสิทธิ์ของวงจรกรองเอฟไอาร์ ที่มีการตอบสนองในโดเมนของความถี่ วิธีที่นิยมที่สุดในการออกแบบวงจรกรองเอฟไอาร์มี 2 วิธี คือ

1.ใช้ ดีเอฟที บนการตอบสนองความถี่ที่แซมเปิ้ล วิธีนี้ต้องการการตอบสนองความถี่ของฟิลเตอร์เป็นแซมเปิ้ลที่ช่วงของความถี่ 1/T เมื่อ T เป็นเวลาระหว่างแซมเปิ้ลในระบบ ดีเอสพี ดิสครีทฟูเรียร์ทรานสฟอร์มผกผันจะถูกใช้ในการตอบสนองแซมเปิ้ล เพื่อสร้างการตอบสนองอิมพัลส์ของฟิลเตอร์ให้ได้ผลที่ดีที่สุด ปกติจะสำเร็จถ้าหน้าต่างราบเรียบซึ่งใช้ตอบสนองความถี่ก่อนใช้รูปแบบของ ดีเอฟที ผกผัน

2.วิธีการประมาณค่าต่ำสุด-สูงสุด ที่ดีที่สุด โดยการใช้วิธีการโปรแกรมแบบลิเนียร์ ซึ่งเขียนขึ้นมาโดยโปรแกรมเมอร์

การตอบสนองความถี่ของวงจรกรองเอฟไอาร์สามารถตรวจสอบได้ โดยการใช้ทรานสเฟอร์ฟังก์ชัน สำหรับตัวกระทำที่เป็นลิเนียร์โดยทั่ว ๆ ไป จะได้

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{q=0}^{Q-1}b_q z^{-q}}{1+\sum_{p=1}^{p-1}a_p z^{-p}} \qquad .....(2.67)$$

สังเกตที่เครื่องหมายซัมเมชั่น ที่สิ้นสุดเพื่อทำการฟิลเตอร์ เมื่อวงจรกรองเอฟไอาร์ a_p ทั้งหมดเท่ากับ 0 สมการจะกลายเป็น

$$H(z)=\frac{Y(z)}{X(z)}=\sum_{p=1}^{Q-1}b_q z^{-q} \qquad .....(2.68)$$

ฟูเรียร์ทรานสฟอร์มหรือการตอบสนองความถี่ของทรานสเฟอร์ฟังก์ชันจะได้ $z=e^{j2\pi f}$ ซึ่งจะได้

$$H(f) = H(z)\Big|_{z=e^{j2\pi f}} = \sum_{q=0}^{Q-1} b_q e^{-j2\pi fq} \quad \text{.....(2.69)}$$

สมการนี้เป็นโพลิโนเมียลในกำลังของ z^{-1} หรือผลบวกของผลคูณ(sum of product) ของรูปแบบ

$$H(z) = b_q + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \ldots + b_{Q-1} z^{-(Q-1)} \quad \text{.....(2.70)}$$

ส่วนสำคัญของวงจรกรองเอฟไออาร์ที่เป็นโพลิโนเมียล สามารถทำให้อยู่ในรูปของผลคูณของผลบวก (product of sum) จาก

$$H(z) = \prod_{m=0}^{M-1} (z^{-2} + \alpha_m z^{-1} + \beta_m) \prod_{n=0}^{N-1} (z^{-1} - \gamma_n) \quad \text{.....(2.71)}$$

สมการนี้สำหรับทรานสเฟอร์ฟังก์ชันจะทำให้ได้ค่า z^{-1} ที่แน่นอน ซึ่งจะทำให้ H(z) กลายเป็นศูนย์ ที่จุดนี้เป็นรากของสมการควอดราติค

$$0 = z^{-2} + \alpha_m z^{-1} + \beta_m \quad \text{.....(2.72)}$$

ซึ่งโดยทั่วไปคู่ของคอมเพล็กคอนจูเกต และค่า γ_n จะเป็นศูนย์

2.5.2 ลิเนียร์เฟสในวงจรกรองเอฟไออาร์(Linear Phase in FIR Filters)

ในการใช้งานในทางการสื่อสารและอิมเมจโปรเซสซิ่ง ส่วนมากจำเป็นที่จะต้องมีการฟิลเตอร์ ซึ่งทรานสเฟอร์ฟังก์ชันจะแสดงคุณลักษณะของเฟสซึ่งเปลี่ยนลิเนียร์กับเปลี่ยนความถี่ คุณลักษณะนี้เป็นสิ่งสำคัญ เพราะว่ามันเป็นความสัมพันธ์ของการแปลงเฟสซึ่งจะทำให้สัญญาณที่ผ่านวงจรฟิลเตอร์มีการผิดเพี้ยนน้อยที่สุด ลักษณะที่ใช้ประโยชน์กันมากของวงจรกรองเอฟไออาร์ เป็นความสัมพันธ์ง่าย ๆ ของสัมประสิทธิ์,b_q ผลของการฟิลเตอร์สามารถรับประกันได้ถึงการตอบสนองที่เป็นลิเนียร์เฟส สืบเนื่องจากความสัมพันธ์ซึ่งลิเนียร์เฟสฟิลเตอร์ดังนี้

ความสัมพันธ์ลิเนียร์เฟสกับความถี่จะได้

$$H(f) = |H(f)|e^{j[\alpha f+\beta]} \quad \text{.....(2.73)}$$

เมื่อ α และ β เป็นค่าคงที่ ถ้าทรานสเฟอร์ฟังก์ชันของฟิลเตอร์สามารถแยกได้เป็น ฟังก์ชันที่เป็นส่วนจริงของ f คูณด้วยเฟสแฟคเตอร์ $e^{j[\alpha f+\beta]}$,ทรานสเฟอร์ฟังก์ชันนี้จะแสดงลิเนียร์เฟส

ทำการหาทรานสเฟอร์ฟังก์ชันวงจรกรองเอฟไออาร์จะได้

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \ldots + b_{Q-1} z^{-(Q-1)} \quad \text{.....(2.74)}$$

และแทน z ด้วย $e^{j2\pi f}$ จะได้การตอบสนองความถี่เป็น

$$H(f) = b_0 + b_1 e^{-j2\pi f} + b_2 e^{-j2\pi(2f)} + \ldots + b_{Q-1} e^{-j2\pi(Q-1)f} \quad \text{.....(2.75)}$$

คูณข้างนอกสมการทั้งหมดด้วยตัวคูณ $\exp[-j2\pi(Q-1)f/2]$ และ ζ เท่ากับ $(Q-1)/2$ จะได้

$$H(f) = e^{-j2\pi\zeta f}\left\{b_0 e^{j2\pi\zeta f} + b_1 e^{j2\pi(\zeta-1)} + b_2 e^{j2\pi(\zeta-2)f} + \ldots + b_{Q-2} e^{-j2\pi(\zeta-1)f} + b_{Q-1} e^{-j2\pi\zeta f}\right\} \quad \text{.....(2.76)}$$

ทำการรวมสัมประสิทธิ์กับคอมเพล็กคอนจูเกตเฟสและทำการแทนลงในวงเล็บจะได้

$$H(f) = e^{-j2\pi\zeta f}\left\{\left[b_0 e^{j2\pi\zeta f} + b_{Q-1} e^{-j2\pi\zeta f}\right] + \left[b_1 e^{j2\pi(\zeta-1)f} + b_{Q-2} e^{-j2\pi(\zeta-1)f}\right] + \left[b_2 e^{j2\pi(\zeta-2)f} + b_{Q-3} e^{-j2\pi(\zeta-2)f}\right] + \ldots\right\} \quad \text{.....(2.77)}$$

ถ้าแต่ละคู่ของสัมประสิทธิ์ข้างในวงเล็บคือกลุ่มที่มีค่าดังนี้

$$b_0 = b_{Q-1}$$
$$b_1 = b_{Q-2}$$
$$b_2 = b_{Q-3} \quad \text{ไปเรื่อย ๆ} \quad \text{.....(2.78)}$$

แต่ละเทอมในวงเล็บจะกลายเป็นฟังก์ชันโคไซน์และความสัมพันธ์ของลิเนียร์เฟสที่ได้ ซึ่งเป็นคุณลักษณะทั่วไปของสัมประสิทธิ์วงจรกรองเอฟไออาร์

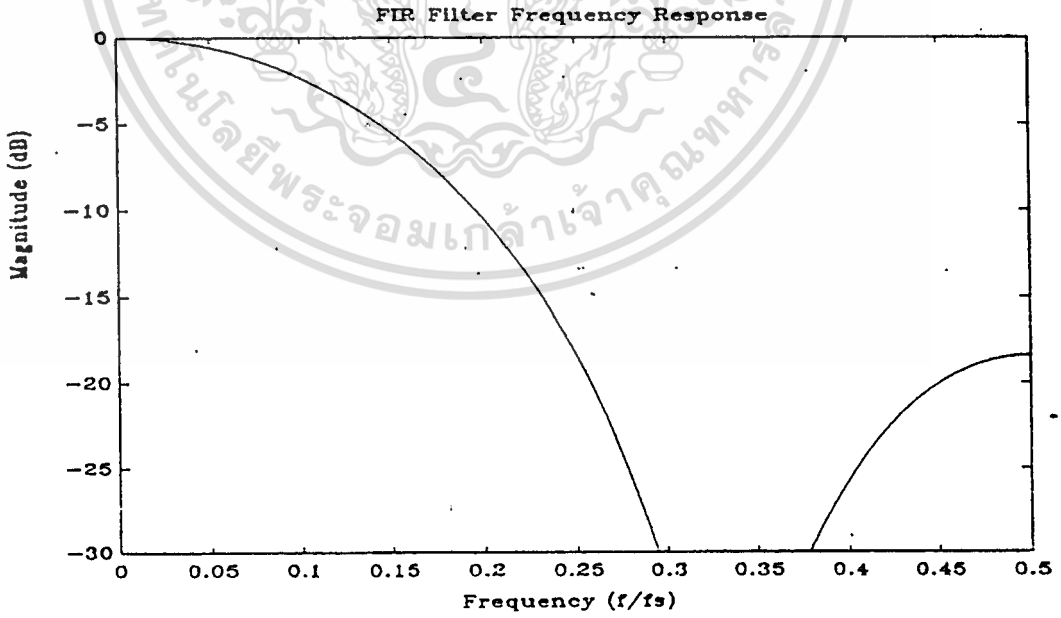
2.5.3 ตัวอย่างการตอบสนองของวงจรกรองเอฟไออาร์(Example of an FIR Filter Response)

ตัวอย่างของการตอบสนองความถี่ของวงจรกรองเอฟไออาร์กับสัมประสิทธิ์ทำได้ง่ายมาก โดยการหาด้วยสมการ MA

$$y(n) = 0.11 * x(n) + 0.22 * x(n-1) + 0.34 * x(n-2) + 0.22 * x(n-3) + 0.11 * x(n-4)$$

.....(2.79)

การตอบสนองของวงจรกรองเอฟไออาร์นี้แสดงดังรูปที่ 2.19 ซึ่งจะยอมให้ความถี่ต่ำผ่านและไม่ให้ผ่านในย่านหยุดความถี่ ซึ่งเป็นคุณลักษณะของดิสครีทไทม์ฟิลเตอร์ทั่ว ๆ ไป



รูปที่ 2.19 การตอบสนองของเอฟไออาร์ โลว์พาสส์

2.5.6 รายละเอียดของฟิลเตอร์(Filter Specifications)

ในส่วนก่อนหน้านี้ ลักษณะของฟิลเตอร์ทั่ว ๆ ไป จะอยู่ในรูปของโดเมนของความถี่ทั้งแอมปลิจูดและการตอบสนองเฟส ที่ฟังก์ชันของความถี่ ในรูปที่ 2.20 แสดงลักษณะการตอบสนองของโลว์พาสส์ฟิลเตอร์ อัตราขยายฟิลเตอร์จะมีค่านอมอลไลซ์ที่เป็น 1.0 ที่ความถี่ต่ำ รูปจะแสดงเทอมที่สำคัญที่สุดกับคุณลักษณะของฟิลเตอร์

ในย่านที่ฟิลเตอร์ยอมให้สัญญาณอินพุทผ่านไปยังเอาท์พุทโดยไม่มีการลดทอน จะเรียกว่า พาสส์แบนด์ ในโลว์พาสส์ฟิลเตอร์ ในย่านพาสส์แบนด์จะเริ่มจากความถี่ $\omega = 0$ ไปจนถึงย่านทรานซิชั่นแบนด์ซึ่งเขียนแทนด้วยความถี่ ω_p ในรูปที่ 2.20 ทรานซิชั่นแบนด์เป็นย่านที่ฟิลเตอร์มีความราบเรียบที่เปลี่ยนจากให้สัญญาณผ่านเป็นหยุดสัญญาณที่จะผ่าน ที่ปลายของย่านทรานซิชั่นแบนด์จะปรากฏที่ย่านหยุดความถี่ ω_s สตอปแบนด์เป็นย่านของความถี่ที่อยู่เลยขึ้นไปเป็นย่านที่ไม่ให้สัญญาณผ่าน ซึ่งฟิลเตอร์จะทำการลดทอนสัญญาณด้วยแฟคเตอร์ที่กำหนดให้ ชนิดของฟิลเตอร์จะกำหนดด้วยค่าพารามิเตอร์ดังนี้

(1) พาสส์แบนด์ริปเปิ้ล เป็น 2δ

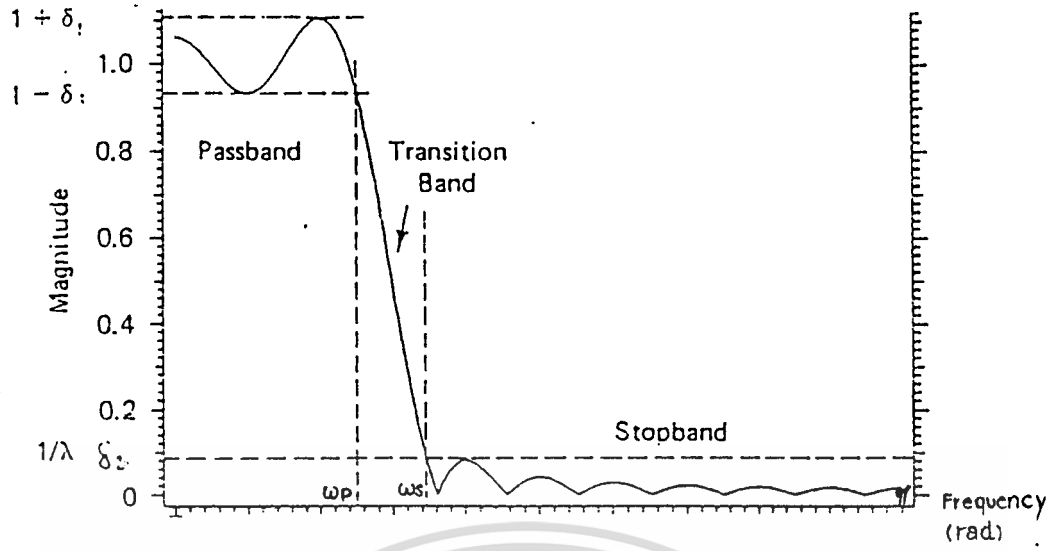
(2) การลดทอนในสตอปแบนด์ เป็น $1/\lambda$

(3) ความถี่เริ่มต้นและหยุดในย่านทรานซิชั่น เป็น ω_p และ ω_s

(4) ความกว้างย่านทรานซิชั่น เป็น $\omega_s - \omega_p$

(5) ความถี่คัทออฟ เป็นความถี่ที่ซึ่งอัตราการขยายฟิลเตอร์มีค่าน้อยกว่าอัตราขยายในย่านพาสส์แบนด์ ซึ่งอาจจะเป็น -1 dB,-3 dB

โปรแกรมคอมพิวเตอร์ จะคำนวณสัมประสิทธิ์ฟิลเตอร์จากพารามิเตอร์ที่ตอบสนองขนาดโดเมนของความถี่ โดยการใช้รายการข้างบนหรือเปลี่ยนแปลงบางอย่างที่โปรแกรมอินพุท



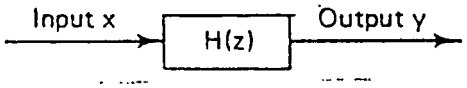
รูปที่ 2.20 ขนาดการตอบสนองของ นอร์มอลไลซ์ โลว์พาสส์ ฟิลเตอร์

2.5.7 โครงสร้างของวงจรฟิลเตอร์(Filter Structures)

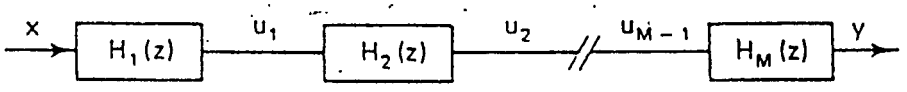
จากสมการ FIR สำหรับฟิลเตอร์ที่จำนวนของโครงสร้างที่สนับสนุนที่สามารถใช้ได้แต่ละโครงสร้าง แม้ว่าการเทียบเคียงทางคณิตศาสตร์อาจจะสร้างผลลัพธ์ที่แตกต่างซึ่งนำไปสู่ความไม่เที่ยงตรงในคอมพิวเตอร์หรือจุดประสงค์พิเศษที่ใช้

รูปที่ 2.21 แสดงถึงโครงสร้าง 3 โครงสร้างสำหรับการสนับสนุนฟิลเตอร์ อันแรกเป็นรูปแบบโดยตรงของทรานสเฟอร์ฟังก์ชัน โครงสร้างนี้ใช้สมการ Z ทรานสฟอร์ม ของฟิลเตอร์ทรานสเฟอร์ฟังก์ชันและมีการหน่วงและทำการคูณสัมประสิทธิ์โดยตรง

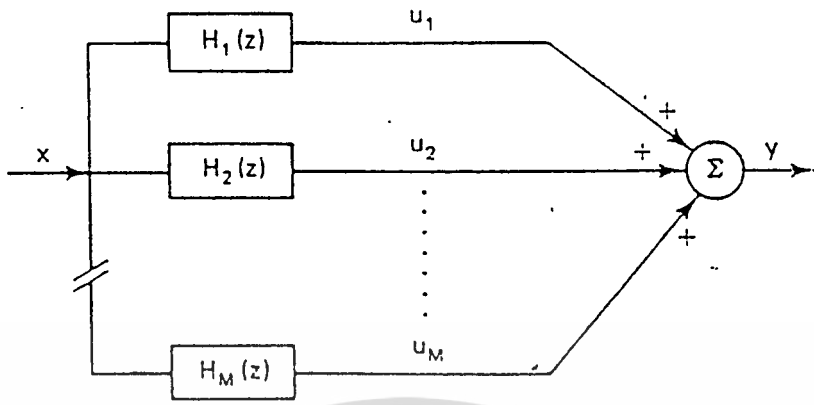
รูปแบบโดยตรงของฟิลเตอร์สามารถเปลี่ยนให้อยู่ในรูปคาสเคด โดยการคูณทรานสเฟอร์ฟังก์ชันเข้าไปในกลุ่มของฟังก์ชันซึ่งจะสร้างทรานสเฟอร์ฟังก์ชันทั้งหมด ในลักษณะที่คล้ายคลึงกัน ส่วนขยายของพาร์เชี่ยล แฟรกชั่น(Partial Fraction) สามารถกระทำได้บนทรานสเฟอร์ฟังก์ชันไปเป็นกลุ่มของฟังก์ชันซึ่งทำการรวมทรานสเฟอร์ฟังก์ชันทั้งหมด ส่วนขยายพาร์เชี่ยล แฟรกชั่นนี้จะต่ออยู่ข้างหน้ารูปแบบที่ขนานกันของดิจิตอลฟิลเตอร์ ดังรูปที่ 2.21 (c)



(a) ทรานสเฟอร์ฟังก์ชันที่เป็น ไดเร็ค ฟอร์ม(Direct form)



(b) ทรานสเฟอร์ฟังก์ชันที่เป็น คาสเคด ฟอร์ม(Cascade form)



(c) ทรานสเฟอร์ฟังก์ชันที่เป็นพาราแลล ฟอร์ม(Parallel form)

รูปที่ 2.21 โครงสร้างของดิจิตอลฟิลเตอร์

2.6 ดิสครีทฟูเรียร์ทรานสฟอร์ม (THE DISCRETE FOURIER TRANSFORM)

เราได้ทำการใช้ฟูเรียร์ทรานสฟอร์มหลาย ๆ ครั้ง ในการพัฒนาคุณลักษณะของลำดับ (sequence)และตัวกระทำลิเนียร์ (linear operator) ฟูเรียร์ทรานสฟอร์มของ คอซอลซีเควน (causal sequence) คือ

$$F\{x(n)\} = X(f) = \sum_{n=0}^{\infty} x(n)e^{-j2\pi fn} \qquad \text{.....(2.80)}$$

เมื่อคาบเวลาการแซมเปิ้ลมีค่าเป็นนอร์มอลไลซ์ไปจนถึง 1 (T=1) ถ้าลำดับเป็นช่วงที่จำกัด (ซึ่งต้องถูกต้องในการใช้ในคอมพิวเตอร์) เมื่อ

$$X(f) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi fn} \qquad \text{.....(2.81)}$$

เมื่อรูปคลื่นแซมเปิ้ลในโดเมนของเวลา คือ ความยาวแซมเปิ้ล N ใช้ฟูเรียร์ทรานสฟอร์มผกผันจะได้

$$F^{-1}\{X(f)\} = x(n) = \int_{-1/2}^{1/2} X(f) e^{-j2\pi fn} df \qquad .....(2.82)$$

เมื่อ X(f) เป็นคาบกับ 1/T =1 อินทีกรัลสามารถหาคาบเต็มได้ทั้งหมด ดังนั้น

$$x(n) = \int_{0}^{1} x(f) e^{-j2\pi fn} df \qquad .....(2.83)$$

2.6.1 รูปแบบของดีเอฟที(Form of the DFT)

สิ่งที่จะกล่าวถึงในที่นี้อีกครั้ง ก็คือความเที่ยงตรงของฟูเรียร์ทรานสฟอร์มแต่มันมีปัญหามากในการใช้งานเป็นดิจิตอล การแปรเปลี่ยนความถี่เป็นค่าที่ต่อเนื่อง,ไม่เป็นดิสครีท ที่กล่าวมาทั้งหมดเป็นปัญหาที่เราจะต้องประมาณทั้งเวลาและความถี่ ให้แทนอยู่ในรูปของสัญญาณ

การสร้าง ดิสครีทฟูเรียร์ทรานสฟอร์ม (DFT) เราต้องใช้การแซมเปิ้ลของรูปคลื่นความถี่ การแซมปลิ้งนี้ในโดเมนของความถี่เป็นเสมือนกับการคอนโวลูชั่นในโดเมนของเวลา ซึ่งจะได้รูปคลื่นเวลาเป็น

$$h_1(t) = \sum_{r=-\infty}^{\infty} \delta(t - rT) \qquad .....(2.84)$$

การสร้างรูปคลื่นแซมเปิ้ลในโดเมนของเวลาซึ่งซ้ำกับคาบ T ,T นี้จะเท่ากันไปจนถึง T ที่ใช้ข้างบนในลำดับของโดเมนเวลา ต่อไปโดยการใช้จำนวนแซมเปิ้ลเดียวกันใน 1 คาบ ของรูปคลื่นโดเมนความถี่ที่ซ้ำกันใน 1 คาบของรูปคลื่นโดเมนเวลา คู่ ดีเอฟที เป็นค่าที่ได้ซึ่งเป็นการประมาณที่เป็นการเปลี่ยนคู่ฟูเรียร์ทรานสฟอร์มที่ต่อเนื่อง รูปแบบของดิสครีทฟูเรียร์ทรานสฟอร์มจะได้เป็น

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad \text{.....(2.85)}$$

และฟูเรียร์ทรานสฟอร์มผกผันคือ

$$x(n) = \frac{1}{N} \sum_{K=0}^{N-1} X(k) e^{-j2\pi kn/N} \qquad \text{.....(2.86)}$$

2.6.2 คุณสมบัติของดีเอฟที(Properties of the DFT)

ในส่วนนี้จะอธิบายถึงคุณสมบัติบางประการของ ดีเอฟที ทฤษฎีการแซมปลิ้ง ได้กล่าวไว้ว่าความถี่ต้องไม่สูงกว่า 1/2T ซึ่งแทนด้วย X(k), อย่างไรก็ดี ค่าของ k ถึง N-1 ซึ่งตอบสนองถึงความถี่ที่ใกล้และเท่ากับความถี่ที่แซมปลิ้ง 1/T ซึ่งหมายถึงว่าลำดับจริง ค่าของ k จาก N/2 ถึง N-1 หรืออีกนัยหนึ่งในความเป็นจริงแอมปลิจูดของค่า X(k) นี้ก็คือ

$$|X(k)| = |X(N-k)|, \text{ สำหรับ } k = N/2 \text{ ถึง } N-1 \qquad \text{.....(2.87)}$$

ดีเอฟที เป็นลิเนียร์ทรานสฟอร์มซึ่งก็คือ Z ทรานสฟอร์ม ดังนั้นจึงได้ความสัมพันธ์ดังนี้ถ้า

$$x(n) = \alpha a(n) + \beta b(n) \qquad \text{.....(2.88)}$$

เมื่อ α และ β เป็นค่าคงที่จะได้

$$X(k) = \alpha A(k) + \beta B(k) \qquad \text{.....(2.89)}$$

เมื่อ A(k) และ B(k) เป็น ดีเอฟที ของฟังก์ชันเวลา a(n) และ b(n) ตามลำดับ

ดีเอฟที จะแสดงด้วยลักษณะที่คล้ายคลึงกันภายใต้การเลื่อนเวลาที่

แซด ทรานสฟอร์ม ถ้า X(k) เป็นดีเอฟที ของ x(n) เมื่อ

$$DFT\{x(n-p)\} = \sum_{n=0}^{N-1} x(n-p)e^{-j2\pi kn/N} \quad \text{.....(2.90)}$$

ขณะนี้จะได้ตัวแปรใหม่ คือ m=r-p ดังนั้น n=m+p ซึ่งจะทำให้

$$DFT\{x(n-p)\} = \sum_{m=-p}^{m=N-1-p} x(m)e^{-j2\pi kn/N}e^{-j2\pi kp/N} \quad \text{.....(2.91)}$$

ซึ่งเปรียบเทียบได้กับสมการต่อไปนี้

$$DFT\{x(n-p)\} = e^{-j2\pi kp/N}X(k) \quad \text{.....(2.92)}$$

สิ่งที่ควรจำสำหรับ ดีเอฟที เรามีการสมมติที่ลำดับ x(m) ไปบนการซ้ำที่ถาวรมันมีค่าบนคาบเวลา n=0 ถึง N-1 ดังนั้นความหมายของ negative time argument เขียนง่าย ๆ ได้เป็น

$$x(-p) = x(N-p) \quad \text{สำหรับ } p = 0 \text{ ถึง } N-1 \quad \text{.....(2.93)}$$

2.6.5 ฟาสฟูเรียร์ทรานสฟอร์ม(The Fast Fourier Transform)

ฟาสฟูเรียร์ทรานสฟอร์ม หรือ เอฟเอฟที คือ สัมประสิทธิ์ที่ใช้สำหรับการคำนวณลำดับของดีเอฟที มันถูกนำไปใช้ประโยชน์ในการคำนวณค่าที่จำนวนมากคือค่าที่ซ้ำใน ดีเอฟที ไปจนถึงคาบเวลาของดิสครีทฟูเรียร์:$e^{-j2\pi kn/N}$ รูปแบบของ ดีเอฟที คือ

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \quad \text{.....(2.94)}$$

กำหนดให้ $W^{nk} = e^{-j2\pi kn/N}$ เราจะเขียนใหม่ได้เป็น

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{nk} \qquad \text{.....(2.95)}$$

ขณะนี้ $W^{(N+qN)(k+rN)} = W^{nk}$ สำหรับทุก ๆ q,r ซึ่งเป็นตัวเลขที่เป็นคาบของฟูเรียร์ต่อไป เราทำการแตกดีเอฟที ออกเป็นสองส่วน คือ

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_N^{2nK} + \sum_{n=0}^{n/2-1} x(2n+1) W_N^{(2n+1)k} \qquad \text{... (2.96)}$$

เมื่อซับสคริบบนฟูเรียร์แสดงถึงขนาดของลำดับ

ถ้าเราแสดง อี ลีเม้นท์ คู่ของซีเควน x(n) โดย x_{ev} และอิลิเมนท์ที่เป็นคี่ โดย x_{od} สมการจะเขียนใหม่ได้เป็น

$$X(k) = \sum_{n=0}^{n/2-1} x_{ev}(n) W_{N/2}^{k} \sum_{n=0}^{N/2-1} x_{od}(n) W_{N/2}^{nk} \qquad \text{.....(2.97)}$$

ขณะนี้เราจะได้สมการสองสมการในรูปแบบของดีเอฟที ซึ่งเราสามารถเขียนได้เป็น

$$X(k) = X_{ev}(k) + W_{N/2}^{k} X_{od}(k) \qquad \text{.....(2.98)}$$

สังเกตว่าเราจะใช้ ดีเอฟที ที่ N/2 เท่านั้นในการคำนวณหาค่าของ X(k) เมื่อดัชนี k ต้อง มีค่าถึง N-1 อย่างไรก็ดีคุณสมบัติของ ดีเอฟที คู่และคี่จะใช้ทั้งคู่

$$X_{ev}(k) = X_{ev}(k - \frac{N}{2}) \quad \text{สำหรับ } \frac{N}{2} \le k \le N-1 \qquad \text{.....(2.99)}$$

กระบวนการของการหาร ดีเอฟที ให้เป็นคู่หรือคี่สามารถทำซ้ำได้จนกระทั่งค่าทางด้านซ้ายมือเป็น 1 กับ DFT สองจุดเท่านั้น

$$\Delta(k) = \lambda(0) + \lambda(1)e^{-j2\pi k/2}$$

สำหรับทุกค่า K

$$= \lambda(0) + \lambda(1) \qquad \text{เมื่อ k เป็น คู่}$$

$$= \lambda(0) - \lambda(1) \qquad \text{เมื่อ k เป็น คี่}$$

.....(2.100)

ดังนั้น สำหรับ 2 จุดของ ดีเอฟที ที่ไม่มีการคูณจึงเป็นที่ต้องการ โดยการบวกและลบเท่านั้น การคำนวณ ดีเอฟที ที่สมบูรณ์ จะยังคงต้องการคูณของเฉพาะ 2 จุด ของ ดีเอฟที โดยส่วนประกอบที่สมควรของ W จะมีย่านเริ่มจาก W^0 ถึง $W^{N/2-1}$ รูปที่ 2.22 แสดง โฟลวกราฟ ของ 32 จุด เอฟเอฟที ที่สมบูรณ์ เราสามารถคำนวณและเก็บไว้ในการคำนวณถึง เอฟเอฟที อัลกอริทึม

สำหรับจุดเริ่มต้นของ ดีเอฟที,N complex จะคูณเป็นค่าที่ต้องการสำหรับแต่ละ N ค่าของ k แม้ว่า N-1 จะทำการเพิ่มของแต่ละค่า k

ใน เอฟเอฟที แต่ละฟังก์ชันของรูปแบบ

$$\lambda(0) \pm W^p \lambda(1) \qquad \text{.....(2.101)}$$

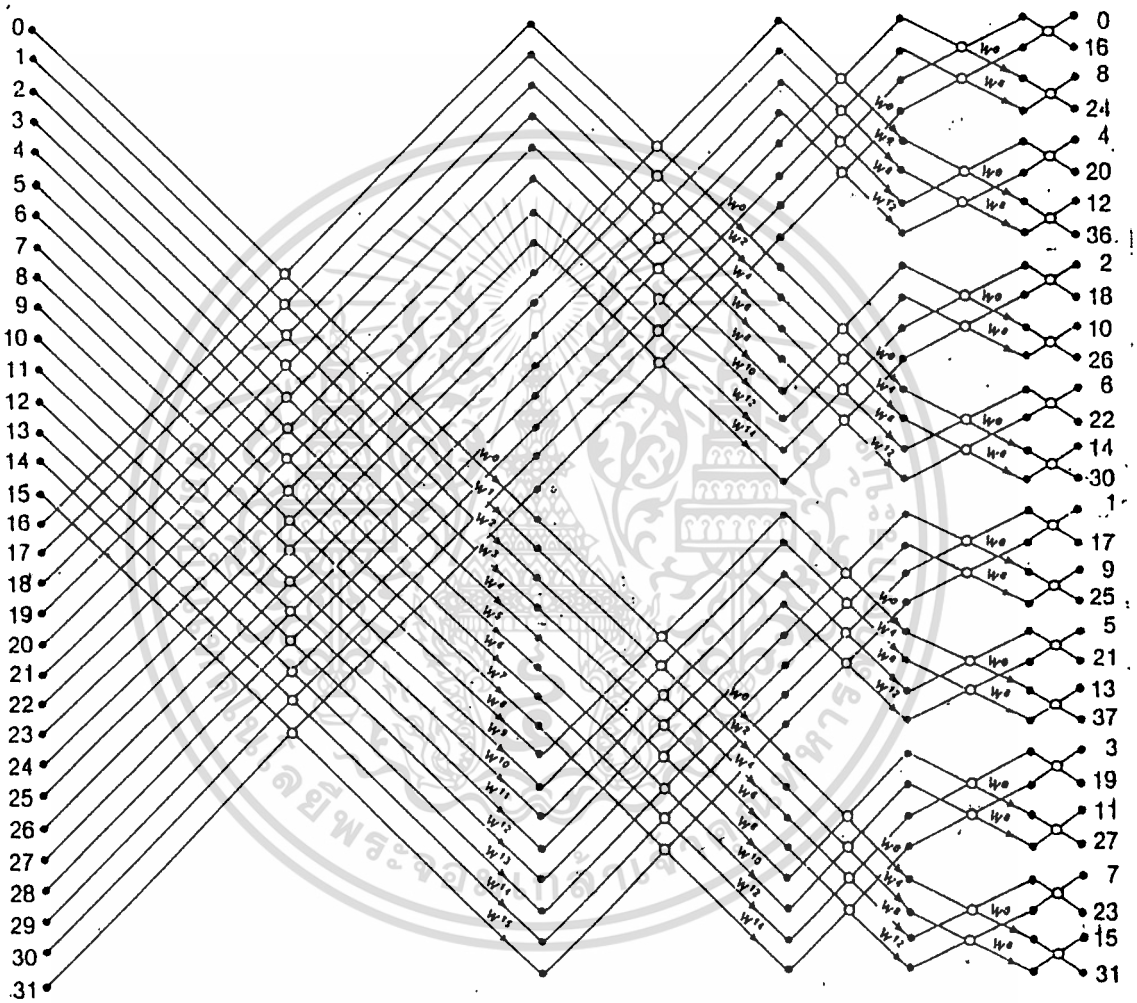
(เรียกว่า บัตเตอร์ฟลาย ซึ่งดูจากรูปร่างลักษณะของ โฟลวกราฟ) ต้องการ คูณ 1 และ บวก 2 จากโฟลวกราฟ ในรูปที่ 2.22 เราสามารถสร้างจำนวนของบัตเตอร์ฟลายเป็น

$$\text{จำนวนของบัตเตอร์ฟลาย} = \frac{N}{2}\log_2(N) \qquad \text{.....(2.102)}$$

ที่เป็นเช่นนี้เพราะว่าในแถว N/2 ของบัตเตอร์ฟลาย(เมื่อแต่ละบัตเตอร์ฟลายมี 2 อินพุท) และมีค่า $\log_2(N)$ ในหลักของบัตเตอร์ฟลาย

ตาราง 1.1 การเปรียบเทียบจำนวนของบัตเตอร์ฟลาย ใน ดีเอฟที และ เอฟเอฟที

ความยาวของ TRANSFORM (N)	DFT Operations (N^2)	FFT Operations N LOG_2 (N)
8	64	24
16	256	64
32	1024	160
64	4096	348
128	16384	896
256	65536	1024
512	262144	4608
1024	1048576	10240
2048	4194304	22528



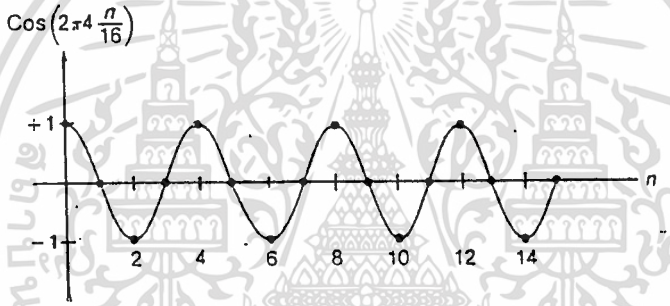
รูปที่ 2.22 32-point, radix 2, in-place FFT

2.6.6 ตัวอย่างของเอฟเอฟที(An Example of the FFT)

เพื่อที่จะช่วยให้ขยายความเข้าใจได้มากขึ้นในการวิเคราะห์สเปคตรัมด้วย เอฟเอฟที ในที่นี้จะแสดงตัวอย่างง่าย ๆ โดย สัญญาณอินพุทมี 16 จุด ดังนี้

$$x(n) = \cos\left[2\pi(4n/16)\right] \quad \text{.....(2.103)}$$

อ้างเหตุผลของโคไซน์ที่มีการเขียนผิดปกติเพื่อเป็นความถี่ของรูปคลื่นเมื่อทำการโปรเซสด้วย 16 จุด เอฟเอฟที แอมปลิจูดของสัญญาณมีค่าเป็น 1 และเป็นสัญญาณจริง ๆ ส่วนของจินตภาพจะมีค่าแอมปลิจูดเป็น 0 รูปที่ 2.23 แสดง 16 แซมเปิล ซึ่งประกอบด้วย x(0) ถึง x(15)



รูปที่ 2.23 1 อินพุท ต่อ 16 จุด เอฟเอฟที

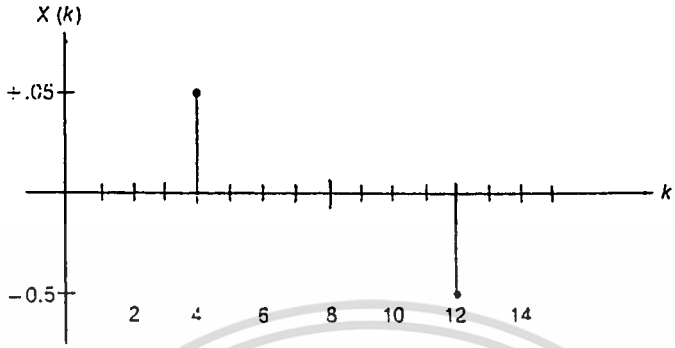
ด้วยสัญญาณ 16 จุด เอฟเอฟที นี้ จะสร้างเอาท์พุทได้ง่ายมาก ๆ เอาท์พุทนี้แสดงดังรูปที่ 2.24 ซึ่งมันมีแซมเปิ้ลที่ k=4 และมีแอมปลิจูด 0.5 และมีแซมเปิ้ลที่ k=12 และมีแอมปลิจูด -0.5 เราสามารถดูได้ว่าทำไมจึงเป็นเช่นนั้น โดยอ้างถึงจากข้างหลังไปยังฟูเรียร์ทรานสฟอร์มต่อเนื่อง รูปคลื่นโคไซน์ของความถี่ฟูเรียร์ทรานสฟอร์ม จะเป็น

$$X(f) = \int_{-\infty}^{+\infty} \cos(2\pi f_0 t) e^{-j2\pi ft}\, dt \quad \text{.....(2.104)}$$

ทำการแทนโคไซน์ด้วยเอ็กโปเนนเชี่ยล

$$X(f) = \frac{1}{2}\int_{-\infty}^{\infty} \exp\left[j2\pi(f_0 - f)t\right]dt - \frac{1}{2}\int_{-\infty}^{\infty} \exp\left[-j2\pi(f_0 + f)t\right]dt \quad \text{.....(2.105)}$$

สามารถแสดงการอินทีกรัล ใน 2 อินทีกรัล ข้างบนอินทีเกรตเป็น 0 โดยปราศจาก อาร์กูเม้นท์ ของเอ็กโปเนเชี่ยล เป็น 0 ถ้าอาร์กูเม้นท์ของ เอ็กโปเนนเชี่ยล เป็น 0



รูปที่ 2.24 เอาท์พุท ของ 16 จุด เอฟเอฟที

ผลก็คือ 2 แซมเปิ้ล อันหนึ่งที่ $f=f_0$ และที่ $f=-f_0$ ซึ่งเป็น เดลต้า ฟังก์ชัน ในโดเมน ความถี่ พื้นฐานบนผลอันนี้ และลำดับอิมพัลส์ เป็น ดิจิตอล อะนาลอก ของ เดลต้า ผลของ เอฟเอฟที ดูเหมือนจะมีเหตุผลมากกว่า อธิบายได้ว่า ทำไม k=12 ซึ่งเสมือนเป็น $f=-f_0$ เมื่อย้อนหลังไปดูการพัฒนาของ ดีเอฟที มันจำเป็นที่จุดหนึ่งของสเปคตรัมความถี่ จะกลายเป็นพีรีออดิก กับคาบเวลา f_sแม้ว่าใน ดีเอฟที มีค่าเป็นบวกเล็กน้อยเท่านั้น การรวมทั้งสองเป็นอันเดียว จะแสดงไว้ใน รูปที่ 2.24

บทที่ 3

การออกแบบวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์ด้วยวิธีหน้าต่าง

ฟังก์ชันระบบของวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์เขียนเป็นคณิตศาสตร์ ได้ว่า

$$H_{FIR}(z) = \sum_{n=-N_p}^{N_p} b_k z^{-k} \qquad .....(3.1)$$

หากทราบค่าสัมประสิทธิ์ b_k, $-N_F<k<N_p$ ก็สามารถนำไปสร้างเป็นวงจรโดยใช้โครงสร้างแบบนันรีเคอร์ซีฟ จากคุณสมบัติของวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์ เราทราบดีว่าสัมประสิทธิ์ดังกล่าวและค่าตอบสนองค่าหนึ่งนั้นมีค่าเท่ากัน หรือ

$$h(n) = b_n \quad \text{สำหรับ } -N_F<n<N_p \qquad .....(3.2)$$

ซึ่งนั่นก็คือ เราสามารถสร้างวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์ได้โดยตรงจากค่าตอบสนองค่าหนึ่ง

โดยปกติแล้ว การออกแบบวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์ให้มีการตอบสนองความถี่ของเฟสมีความเป็นเชิงเส้น ซึ่งกระทำได้โดยมีเงื่อนไขว่าค่าตอบสนองค่าหนึ่งที่ปรารถนา $\{h_D(n)\}$ จะต้องมีลักษณะสมมาตร(symmetry) กล่าวคือ $\{h_D(n)\}$ เป็นจำนวนที่สิ้นสุดและสมมาตรในช่วง $-N_F<n<N_p$ โดยที่ $N_F = N_p$ (อันเนื่องจากความสมมาตร) หาก $\{h_D(n)\}$ มีลักษณะดังกล่าวมานี้ จะสามารถสร้างวงจรกรองสัญญาณดิจิตอลประเภทเอฟไอาร์ที่มีค่าตอบสนองค่าหนึ่งตรงตาม $\{h_D(n)\}$ ที่กำหนดมาให้ได้อย่างแม่นยำ โดยทำให้สัมประสิทธิ์ b_n เท่ากับ $h_D(n)$ หรือนั่นคือ

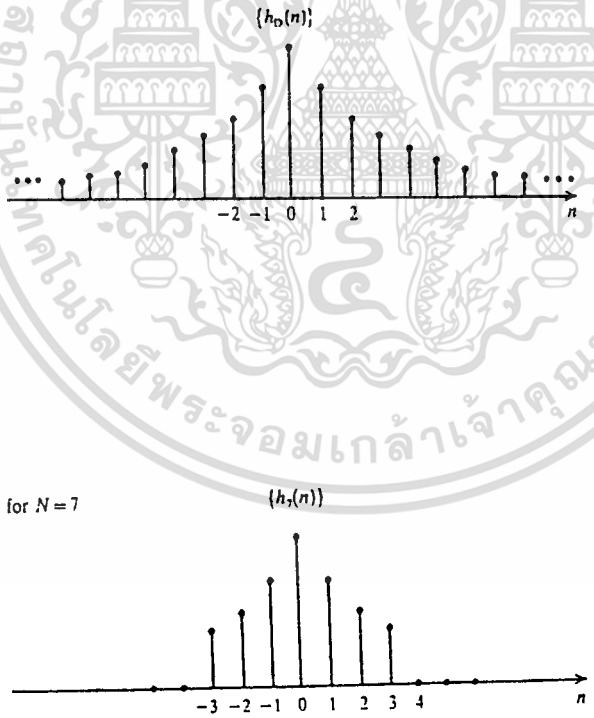
$$b_n = h_D(n) \quad \text{โดยที่ } -N_F<n<N_p \qquad .....(3.3)$$

แต่ในทางปฏิบัติมักจะพบปัญหาตรงที่ว่าข้อกำหนดทางอุดมคติจะทำให้ $\{h_D(n)\}$ มีจำนวนอนันต์หรือนั่นคือจำนวนที่ไม่สิ้นสุด ตัวอย่าง เช่น วงจรกรองสัญญาณโลว์พาสส์อุดมคติ

จะมี $\{h_D(n)\}$ จำนวนอนันต์ เป็นต้น โดยหลักการแล้วจะพิสูจน์ได้เสมอว่า $\{h_D(n)\}$ จะมีจำนวนอนันต์เสมอหากค่าตอบสนองความถี่ของขนาด $|H_D(e^{j\omega})|$ หรืออนุพันธ์ใดๆ ของ $|H_D(e^{j\omega})|$ มีความไม่ต่อเนื่อง (discontinuity) กรณีที่ $\{h_D(n)\}$ มีจำนวนถึงอนันต์ ดังนี้เราจำเป็นต้องตัดทอนจำนวนให้เหลือเป็นจำนวน N สิ้นสุด และเขียนค่าตอบสนองค่าหนึ่งที่ตัดทอนแล้วนี้ว่า $\{h_N(n)\}$

จากการที่ต้องการให้ การตอบสนองความถี่ของเฟสมีความเป็นเชิงเส้นเสมอทำให้ค่าตอบสนองค่าหนึ่งต้องมีความสมมาตรคู่ (even symmetry)หรือความสมมาตรคี่ (antisymmetry) รอบจุดใดจุดหนึ่งในแกนเวลา เพื่อไม่ให้ซับซ้อนเกินไปเราลองพิจารณากรณีที่ $\{h_D(n)\}$ สมมาตรรอบจุด n=0 หรือนั่นคือ

$$h_D(n) = h_D(-n) \qquad \text{.....(3.4)}$$



รูปที่ 3.1 เปรียบเทียบค่าตอบสนองค่าหนึ่ง $\{h_D(n)\}$ ซึ่งมีจำนวนอนันต์ และค่าตอบสนองค่าหนึ่ง $\{h_N(n)\}$ จำนวน 7 ค่าซึ่งเป็นการประมาณ $\{h_D(n)\}$

ในการตัดทอนให้เหลือจำนวน N ต้องรักษาให้ $\{h_D(n)\}$ ที่ได้รับยังคงมีความสมมาตรรอบๆ จุด n=0 ด้วย ดังแสดงในรูปที่ 3.1 ทำให้เราเขียนค่าตอบสนองค่าหนึ่งจำนวน N ดังกล่าวโดยที่ N เป็นเลขจำนวนเต็มคี่ได้ดังนี้

$$h_N = \begin{cases} h_D(n) & \text{สำหรับ } -(N-1)/2 \le n \le (N-1)/2 \\ 0 & \text{สำหรับ N อื่นๆ} \end{cases} \qquad .....(3.5)$$

หากประสงค์จะให้วงจรกรองสัญญาณมีลักษณะเชิงเหตุ(causal) เราต้องหน่วง $\{h_D(n)\}$ ตามแกนเวลาด้วยปริมาณ (N-1)/2 เพื่อให้ $\{h_D(n)\}$ เริ่มต้นที่ n=0 ต้องตั้งข้อสังเกตด้วยว่าหาก N เป็นเลขจำนวนเต็มคี่แล้วจุดสมมาตรจะตรงกับตำแหน่งค่าที่ถูกสุ่มค่าใดค่าหนึ่งพอดี แต่หาก N เป็นเลขจำนวนเต็มคู่ แล้วจุดสมมาตรจะอยู่ตรงตำแหน่งระหว่างค่าที่ถูกสุ่มสองค่า

เพื่อประโยชน์ของการวิเคราะห์ทางคณิตศาสตร์ จะเปรียบเทียบการตัดทอนเสมือนหนึ่งการคูณค่าตอบสนองค่าหนึ่งที่เราปรารถนาซึ่งมีจำนวนอนันต์ $\{h_D(n)\}$ ด้วยค่าหน้าต่าง ซึ่งมีสัญญลักษณ์ทางคณิตศาสตร์ว่า $\{\omega(n)\}$ ดังนั้นเราเขียนคณิตศาสตร์ของการได้มาซึ่ง $\{h_D(n)\}$ ว่า

$$h_N(n) = h_D(n)\omega(n) \quad \text{สำหรับทุกค่าของ } n \qquad .....(3.5)$$

$$\omega(n) = \omega(-n) \neq 0 \quad \text{สำหรับ } |n| \le (N-1)/2$$

$$= 0 \quad \text{สำหรับ } |n| > (N-1)/2 \qquad ....(3.6)$$

$\{\omega(n)\}$ มีจำนวนที่สิ้นสุดคือจำนวน N ค่าและสมมาตรรอบจุด n=0 เพื่อจะยังคงรักษาความสมมาตรซึ่งมีอยู่เดิมใน $\{h_D(n)\}$

จึงพอสรุปการออกแบบด้วยวิธีหน้าต่างว่า จากค่าตอบสนองค่าหนึ่งจำนวนอนันต์ที่ปรารถนา $\{h_D(n)\}$ เราต้องเลือกค่า $\{\omega(n)\}$ ที่จะทำให้ได้ค่าตอบสนองค่าหนึ่งจำนวนสิ้นสุด $\{h_D(n)\}$ ซึ่งมีค่าตอบสนองความถี่ของขนาดตรงตามเงื่อนไขของข้อกำหนด เนื่องจากโครงสร้างของวงจร กรองสัญญาณประเภทเอฟไออาร์จะใช้จำนวนหน่วยตัวคูณมากขึ้น เมื่อ N มีค่ามากขึ้น ฉะนั้นเราจึงปรารถนาที่จะให้ $\{\omega(n)\}$ มีจำนวนที่น้อยเพื่อลดจำนวนหน่วยตัวคูณลงซึ่งจะเป็นการประ หยัดค่าใช้จ่ายในการสร้างวงจร

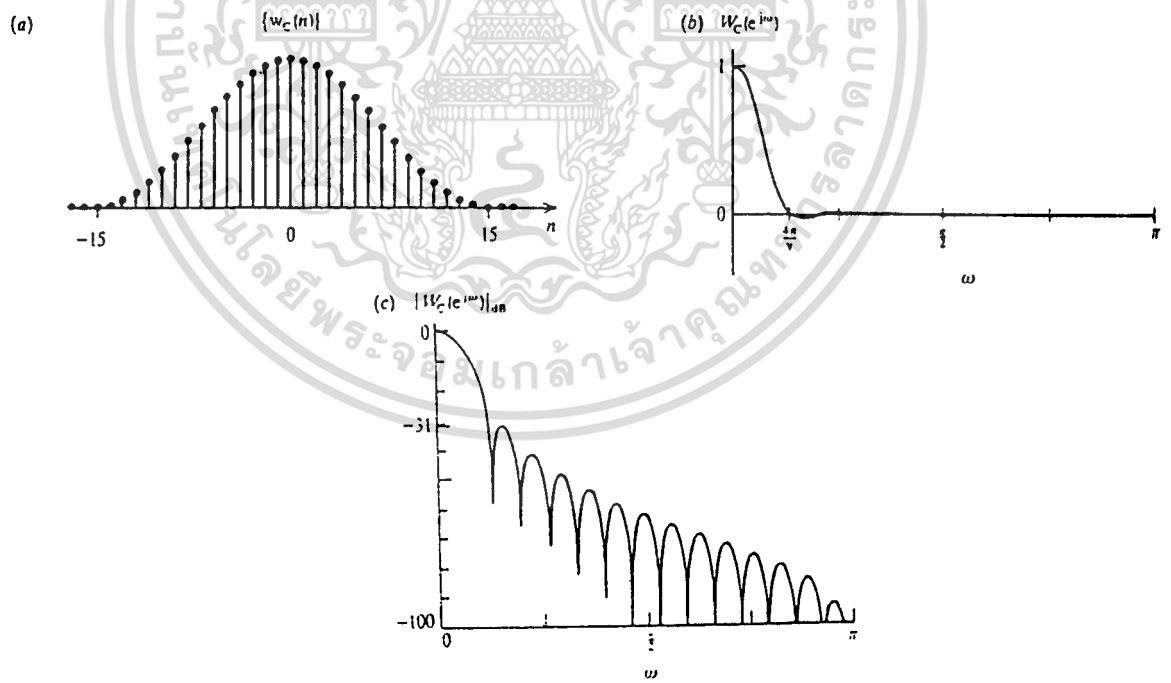
ประเภทที่สองเรียกว่า หน้าต่างแฮมมิ่ง (Hamming Window)

ประเภทที่สามซึ่งเรียกว่า หน้าต่างแบล็คแมน (Blackman Window)

เราจะเริ่มจากการพิจารณาหน้าต่างแฮนนิ่ง ซึ่งมีสัญญลักษณ์ {ωc(n)} และนิยามว่า

$$\omega_c(n) = \begin{cases} 0.5+0.5\cos(2\pi n/(N-1)) & -(N-1)/2 \le n \le (N-1)/2 \\ 0 \quad \text{สำหรับค่าอื่น ๆ ของ N} & \end{cases} \qquad .....(3.10)$$

รูปที่ 3.3 แสดงค่าของหน้าต่างนี้และสเปคตรัม $\omega_c(e^{j\omega})$ ของมันเมื่อ N=31 ซึ่งลอนข้างเคียงแรก (ซึ่งมีระดับสูงกว่าข้างเคียงอื่น ๆ) จะอยู่ที่ -31 ดีบี นั่นคือ ต่ำกว่ากรณีหน้าต่างสามเหลี่ยม 6 ดีบี และเนื่องจากความกว้างของลอนหลักจะมีค่า $8\pi/N$ ซึ่งเท่ากับกรณีหน้าต่างสามเหลี่ยม เพราะฉะนั้นหน้าต่างแฮนนิ่งจึงเป็นที่นิยมมากกว่าหน้าต่างสามเหลี่ยม



รูปที่ 3.3 หน้าต่างเร็สโคไซน์ (แฮนนิ่ง) (ซึ่งมี N=31)

a) ค่าของหน้าต่างในแกนเวลา

b) สเปคตรัม

c) ขนาดของสเปคตรัมเป็นดีบี

เราสามารถจะศึกษาว่าลอนข้างมีระดับลดลงไปได้ จากการพิจารณาว่า $\{\omega_c(n)\}$ นั้นเป็นผลคูณระหว่าง ค่าเร็สโคไซน์ซึ่งมีจำนวนอนันต์กับค่าหน้าต่างสี่เหลี่ยมหรือนั่นคือ

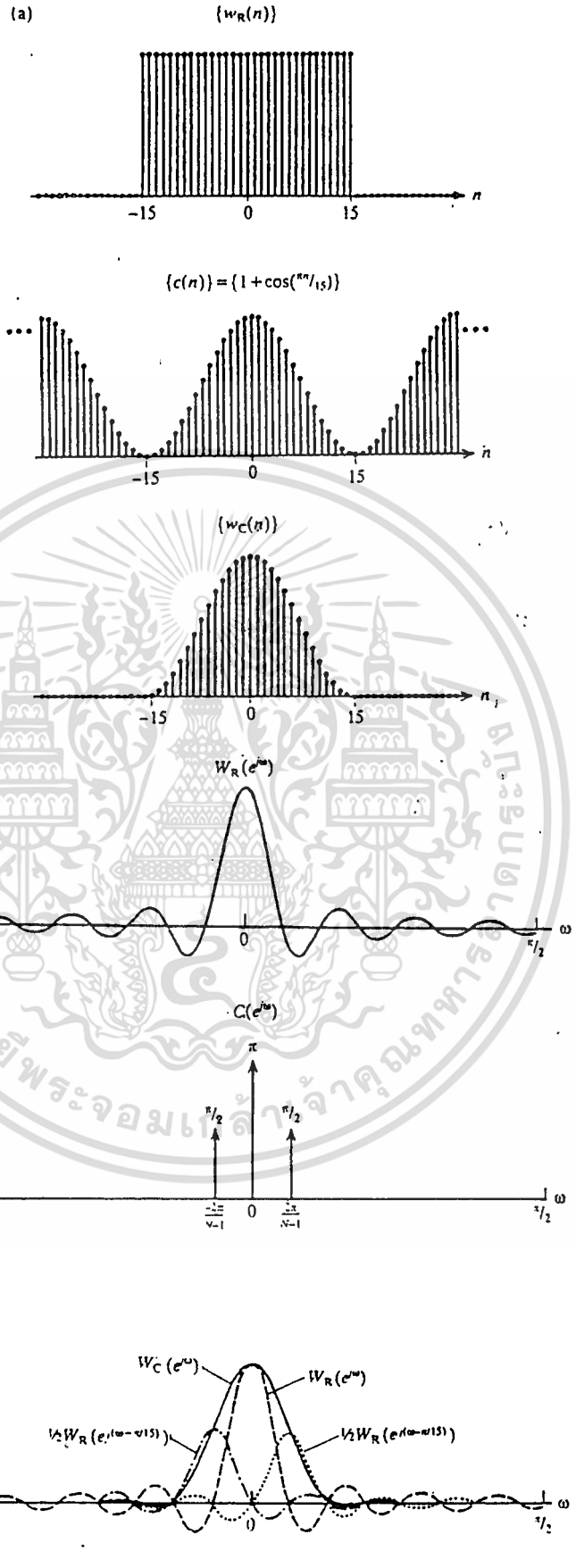
$$\omega_c(n) = [0.5+0.5\cos\{(2\pi n/(N-1))\}]\omega_R(n) \quad \text{สำหรับทุกค่าของ } n$$
$$= c(n)\omega_R(n) \quad \text{สำหรับทุกค่าของ } n \quad .....(3.11)$$

โดยที่ $\{c(n)\}$ เป็นค่าเร็สโคไซน์จำนวนอนันต์ดังแสดงในรูปที่ 3.4(a) การคูณในแกนเวลาตามความสัมพันธ์ ของสมการ (3.11) นี้ ก็คือ การคอนโวลูชั่นวงกลมของฟูเรียร์ทรานส์ฟอร์มของฟังก์ชันทั้งสองในแกนความถี่นั่นเองเนื่องจาก $\{c(n)\}$ ประกอบด้วยค่าคงที่และค่าโคไซน์ที่ความถี่ (N-1)

ดังนั้น ฟูเรียร์ทรานส์ฟอร์มจึงสามารถเขียนให้อยู่ในรูปของฟังก์ชันเด็ลต้าดิแร็ค (Dirac delta function) 3 ฟังก์ชัน ดังนี้

$$C(e^{j\omega}) = \pi\delta(\omega)+0.5\pi\delta(\omega-2\pi/(N-1))+0.5\pi\delta(\omega+2\pi/(N-1)) \quad \text{สำหรับ } -\pi\leq\omega\leq\pi \quad .....(3.12)$$

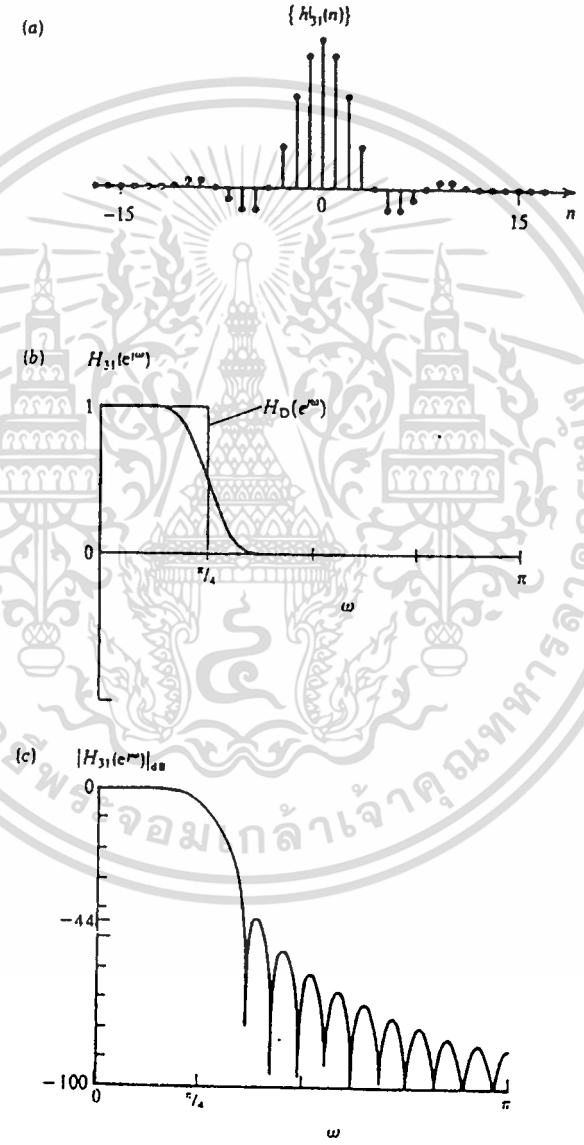
เมื่อเราทำการคอนโวลว์ $\omega_R(e^{j\omega})$ และ$C(e^{j\omega})$ กลไกที่เกิดขึ้นคือสเปคตรัม $\omega_R(e^{j\omega})$ จะไปปรากฏที่ตำแหน่งของอิมพัลส์ทั้งสาม และค่าของแต่ละสเปคตรัมจะถูกปรับไปตามน้ำหนักของแต่ละอิมพัลส์ ผลรวมของสเปคตรัมทั้งสามนี้จะให้สเปคตรัมของหน้าต่างแฮนนิ่ง $\omega_c(e^{j\omega})$ ตามต้องการดังแสดงในรูปที่ 3.4(b) การที่สเปคตรัม $\omega_R(e^{j\omega})$ เคลื่อนไปปรากฏที่ตำแหน่งของอิมพัลส์ทั้งสามนั้น เราจะสังเกตว่าส่งผลให้เกิดการหักล้างระหว่างลอนข้างเคียงของสเปคตรัมทั้งสาม ทำให้ผลลัพธ์สุดท้ายคือ $\omega_c(e^{j\omega})$ มีลอนข้างเคียงที่ระดับของขนาดลดลงยิ่งขึ้น



รูป3.4 การตีความหน้าต่างเร็สโคไซน์หรือหน้าต่างแฮนนิ่ง

a) แกนเวลา b) แกนความถี่

รูปที่ 3.5 แสดงผลการประยุกต์หน้าต่างแฮนนิ่งเพื่อออกแบบวงจรกรองสัญญาณดิจิตอลประเภทเอ็ฟไอาร์ให้มีค่าตอบสนองความถี่ ของขนาดใกล้เคียงโลว์พาสส์อุดมคติมากที่สุด ข้อที่เราสังเกตได้ชัดเจนคือ ย่านหยุดความถี่ที่มีความสามารถลดทอนสัญญาณสูงขึ้น กล่าวคือ ค่าสูงสุดของขนาดในย่านหยุดความถี่จะประมาณ -44 ดีบี ที่ความถี่สูงขึ้นไปความสามารถลดทอนสัญญาณก็จะยิ่งสูงขึ้นไปอีก



รูป 3.5 ผลจากการประยุกต์หน้าต่างเร็สโคไซน์ (ซึ่งมี N=31)กับค่าตอบสนองค่าหนึ่งของโลว์พาสส์อุดมคติ

a) ค่าตอบสนองค่าซึ่งมีจำนวนสิ้นสุดกล่าวคือ N=31

b) การตอบสนองค่าของทรานส์เฟอร์ฟังก์ชันเทียบกับอุดมคติ

c) การตอบสนองความถี่ของขนาดเป็นดีบี

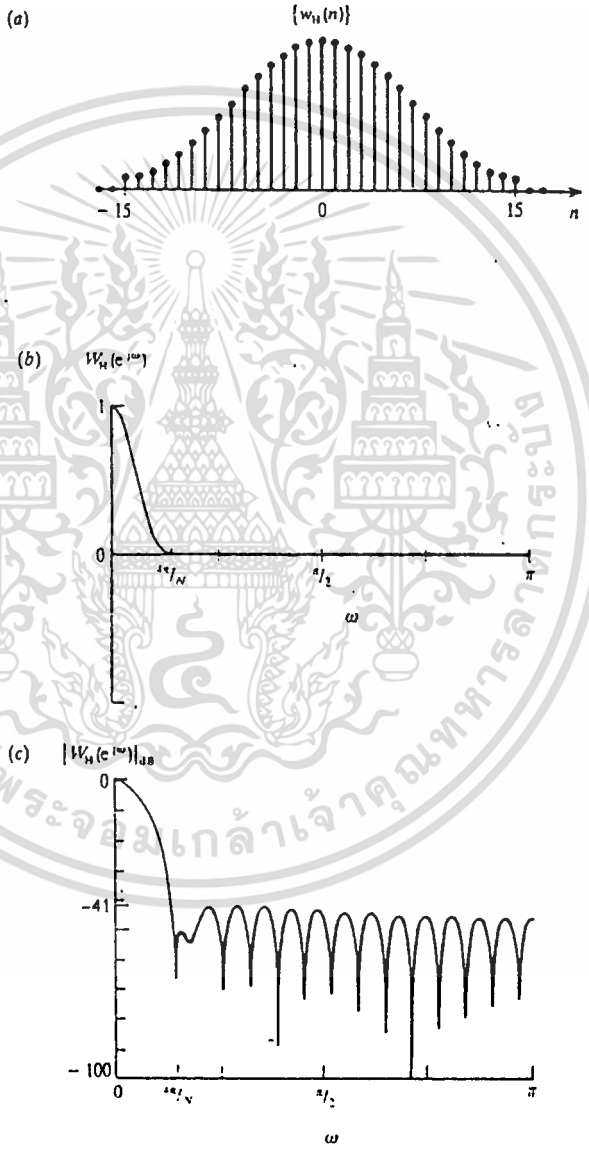
หน้าต่างต่อไปนี้ คือ หน้าต่างแฮมมิ่งซึ่งสามารถลดระดับของลอนข้างเคียงแรกไปได้อีก ทั้งนี้กระทำได้โดยการเพิ่มค่าคงตัวให้แก่ หน้าต่างโคไซน์ หน้าต่างแฮมมิ่งมีสัญญลักษณ์ว่า $\{\omega_H(n)\}$ และมีนิยามว่า

$$\omega_H(n) = \begin{cases} 0.54+0.46\cos(2\pi n/(N-1) & \text{สำหรับ } -(N-1)/2 \le n \le (N-1)/2 \\ 0 \quad \text{สำหรับค่าอื่นๆ ของ } N & \end{cases} \quad .....(3.13)$$

การที่ระดับของลอนข้างเคียงแรกลดลงไปมากกว่ากรณีหน้าต่างแฮนนิ่งนั้น เราสามารถจะอธิบายได้จากการที่สเปคตรัมของหน้าต่างสี่เหลี่ยม $\omega_R(e^{j\omega})$ เคลื่อนและถูกปรับค่าตามที่แสดงในรูปที่ 3.4(b) ในกรณีของแฮนนิ่งนั้น ค่าฟังก์ชันเด็ลต้าดิแร็คที่ $\omega=0$ และ $2/(N-1)$ มีอัตราส่วน 2:1 (กล่าวคือ 0.5: 0.5/2) ซึ่งส่งผลให้ลอนข้างเคียงแรกของ $\omega_c(e^{j\omega})$ มีระดับอยู่ที่ ประมาณ -31 ดีบี โดยการเปลี่ยนอัตราส่วนนี้เป็น 2.35:1 (กล่าวคือ 0.54:0.46/2) จึงทำให้ระดับของลอนข้างเคียงของ $\omega_c(e^{j\omega})$ สามารถลดลงไปอีก และกลายเป็นหน้าต่างแฮมมิ่งนั่นเอง ค่าหน้าต่างแฮมมิ่งและสเปคตรัม $\omega_H(e^{j\omega})$ ของมันกรณี N=31 แสดงอยู่ในรูปที่ 3.6 ซึ่งขนาดหรือระดับของลอนข้างเคียงแรกจะลดลงไปอยู่ที่ -41 ดีบี หรือนั่นคือต่ำกว่าของหน้าต่างแฮนนิ่งอยู่ 10 ดีบี อย่างไรก็ดีข้อเสียเปรียบอื่นเมื่อเทียบกับหน้าต่างแฮนนิ่ง กล่าวคือ ลอนข้างเคียงที่ความถี่สูงของหน้าต่างแฮนนิ่งมีระดับลดลงตามลำดับ ขณะที่ลอนข้างเคียงที่ความถี่สูงของหน้าต่างแฮมมิ่งมีระดับที่เกือบคงที่ รูปที่ 3.7 แสดงผลการประยุกต์ หน้าต่างแฮมมิ่งในการออกแบบวงจรกรองสัญญาณดิจิตอลประเภทเอ็ฟไอาร์ ให้มีค่าตอบสนองความถี่ของขนาดใกล้เคียงโลว์พาสส์อุดมคติมากที่สุด ข้อที่สังเกตได้คือ ลอนข้างเคียงแรกมียอดอยู่ที่ -51 ดีบี หรือนั่นคือต่ำกว่ากรณีที่ใช้หน้าต่างแฮนนิ่งอยู่ -7 ดีบี อย่างไรก็ดีเมื่อความถี่สูงขึ้น ความสามารถที่จะกำจัดสัญญาณจะมิได้เพิ่มขึ้นเท่ากับ ในกรณีวงจรกรองสัญญาณที่ออกแบบโดยหน้าต่างแฮนนิ่ง

เราสังเกตว่าความสามารถกำจัดสัญญาณในย่านหยุดความถี่ของความถี่ ของวงจรกรองสัญญาณ นั้นจะขึ้นอยู่กับระดับของลอนข้างเคียงของฟังก์ชันหน้าต่าง แม้ว่าหน้าต่างสามารถจะให้การลดทอนถึง -51 ดีบี ดังแสดงในรูปที่ 3.7 ก็ตาม แต่ก็อาจจะยังมิเพียงพอสำหรับงานบางประเภท ฉะนั้นหน้าต่างที่มีระดับลอนข้างเคียงต่ำกว่าของหน้าต่างแฮมมิ่งจึงเป็นสิ่ง

จำเป็น หน้าต่างแฮมมิ่งสามารถลดระดับลอนข้างเคียงได้ โดยยังรักษาความกว้างของลอนหลักอยู่ที่ $4\pi/N$ ต่อไปเราจะศึกษาหน้าต่างซึ่งระดับของลอนข้างเคียงสามารถลดลงได้อีกแต่จำเป็นต้องขยายความกว้างของลอนหลัก ตัวอย่างเช่น หน้าต่างแบล็คแมนที่จะกล่าวต่อไปนี้

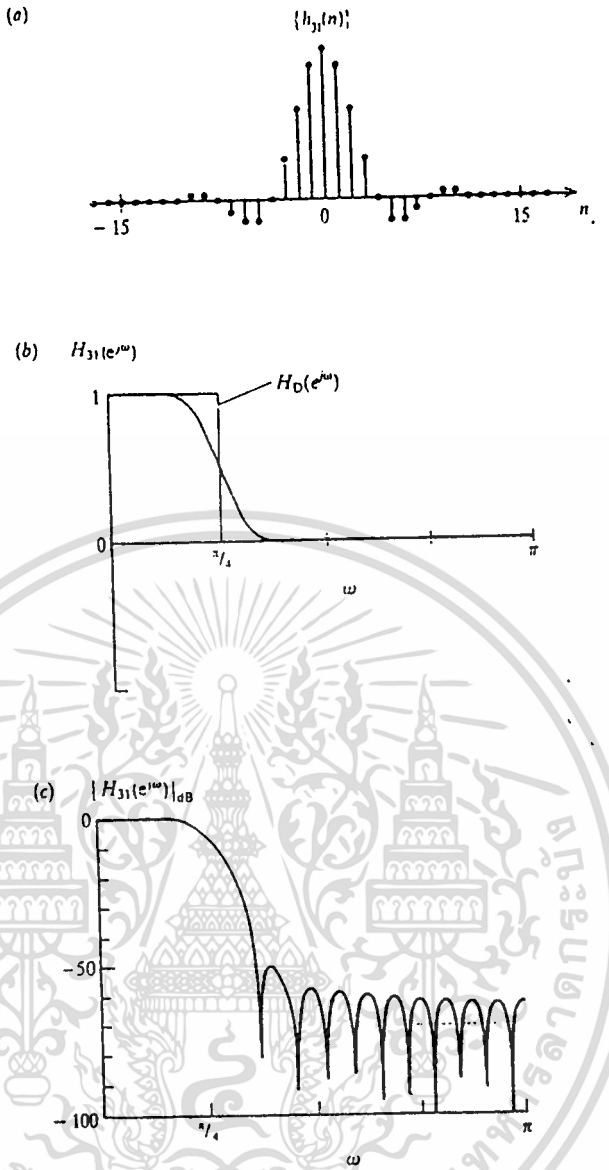


รูป3.6 หน้าต่างแฮนนิ่ง (N=31)

a) ค่าของหน้าต่างในแกนเวลา

b) แกนความถี่

c) ขนาดของสเปคตรัมเป็นดีบี



รูป3.7 ผลจากการประยุกต์หน้าต่างแฮนนิ่งซึ่ง N=31 กับค่าตอบสนองค่าหนึ่งของโลว์

a) ค่าตอบสนองค่าซึ่งมีจำนวนสิ้นสุดกล่าวคือ N=31

b) การตอบสนองความถี่ของทรานส์เฟอร์ฟังก์ชันเทียบกับอุดมคติ

c) การตอบสนองความถี่ของขนาดเป็นดีบี

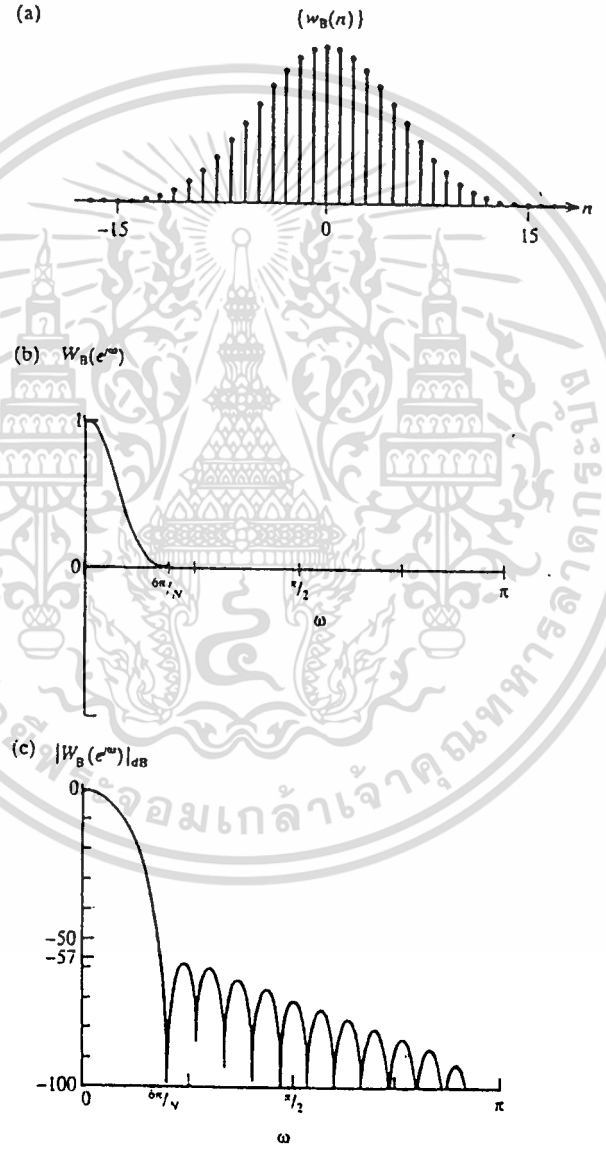
หน้าต่างแบล็คแมนสามารถลดระดับของลอนข้างเคียงต่อไปอีก โดยเพิ่มจำนวนโคไซน์มากขึ้น ดังนี้

$$\omega_B(n) = 0.42+0.5\cos(2\pi n/(N-1)+0.80\cos(2\pi n/(N-1))$$

สำหรับ $-(N-1)/2 \le n \le (N-1)/2$

$= 0$ สำหรับ n อื่นๆ .....(3.14)

ดังแสดงในรูปที่ 3.8 โดยที่ขนาดของลอนข้างเคียงแรกถูกลดลงไปอยู่ที่ -57 ดีบี แต่ความกว้างของลอนหลักเพิ่มไปเป็น $12\pi/N$ ซึ่งเป็นตัวอย่างของการแลกเปลี่ยนระหว่างขนาดหรือระดับของลอนข้างเคียงและความกว้างของลอนหลัก



รูป 3.8 หน้าต่างแบล็คแมน (N=31)

a) ค่าของหน้าต่าง

b) แกนความถี่

c) ขนาดของสเปคตรัมเป็นดีบี

การลดลงของระดับของลอนข้างเคียง สามารถอธิบายได้ในทำนองเดียวกับที่เคยอธิบายมาแล้วในกรณีของหน้าต่างแฮนนิ่งกล่างคือ เราสามารถเขียนหน้าต่างแบล็คแมนในรูปผลคูณระหว่างค่าจำนวนอนันต์ของแบล็คแมนและค่าของหน้าต่างสี่เหลี่ยมดังนี้

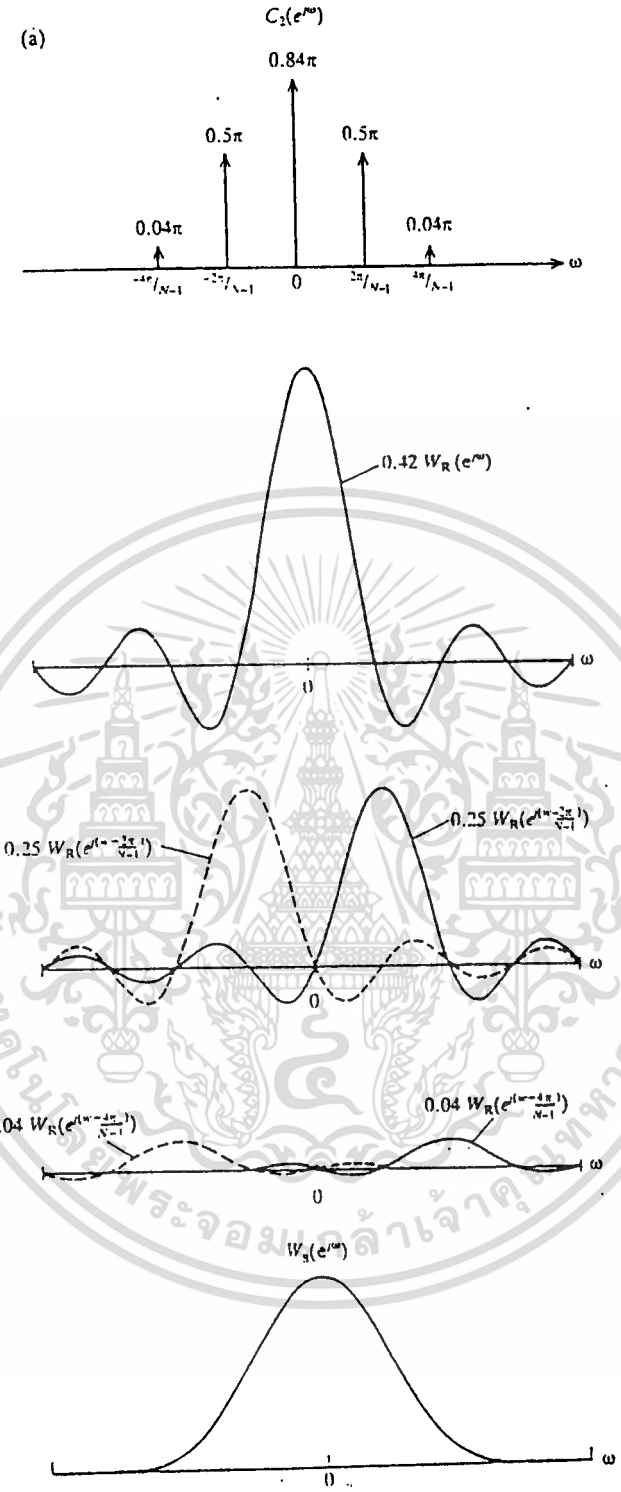
$$\omega_B(n) = [0.42+0.5\cos(2\pi n/(N-1))+0.08\cos(4\pi n/(N-1))]\omega_R(n)$$
$$= C_2(n)\omega_R(n) \qquad \text{สำหรับทุกค่าของ } n \qquad .....(3.15)$$

โดยที่ $\{\omega_R(n)\}$ เป็นค่าหน้าต่างสี่เหลี่ยมจำนวน N ค่า และ $\{C_2(n)\}$ เป็นค่าจำนวนอนันต์ของแบล็คแมน ดังที่เคยทราบมาแล้วว่าการคูณระหว่างสัญญาณสองสัญญาณในแกนเวลาก็คือ การคอนโวลูชั่นของสเปคตรัมของสัญญาณทั้งสองในแกนความถี่ เนื่องจาก $\{C_2(n)\}$ ประกอบ ด้วยค่าคงที่หนึ่งค่าและฟังก์ชันโคไซน์สองตัวที่ความถี่ $\omega=2\pi/(N-1)$ และ $\omega=4\pi/(N-1)$ เราจึงเขียนฟูเรียร์ทรานส์ฟอร์มของสมการที่ (3.15) ได้ว่า

$$C_2(e^{j\omega}) = 0.84\ \pi\delta(\omega)+0.5\ \pi\delta(\omega-2\pi/(N-1))+0.5\pi\ (\omega+2\pi/(N-1))$$
$$+0.04\pi\delta(\omega-4\pi/(N-1))+0.04\pi\delta(\omega+4\pi/(N-1))$$
$$-\pi\le\omega\le\pi \qquad .....(3.16)$$

ซึ่งแสดงอยู่ในรูปที่ 3.9 (a) ส่วนรูปที่ 3.9 (b) แสดงการที่ $\omega_R(e^{j\omega})$ เคลื่อนไปอยู่ที่ตำแหน่งของอิมพัลส์ต่าง ๆ และถูกปรับค่าตามน้ำหนักของค่าอิมพัลส์นั้น และรูปที่ 3.9 (c) แสดงสเปคตรัมของหน้าต่างแบล็คแมน $\omega_B(e^{j\omega})$ ซึ่งเป็นผลรวมของบรรดา $\omega_R(e^{j\omega})$ ที่ถูกปรับค่าและปรากฏที่ตำแหน่งอิมพัลส์ต่าง ๆ ในรูปที่ 3.9 (b) เราจะสังเกตว่าความกว้างของลอนหลักที่กว้างขึ้นก็เนื่องมาจาก $\omega_R(e^{j\omega})$ ที่ถูกปรับค่าและปรากฏที่ $w=\pm4\pi/(N-1)$ แต่ในขณะเดียวกันก็จะช่วยลดระดับของลอนข้างเคียงด้วย

รูปที่ 3.10 แสดงผลจากการประยุกต์หน้าต่างแบล็คแมนในการออกแบบวงจรกรองสัญญาณเอ็ฟไออาร์ให้มีค่าตอบสนองความถี่ใกล้เคียงโลว์พาสส์อุดมคติมากที่สุด ระดับของลอนข้างเคียงจะลดลงไปอยู่ที่ -74 ดีบี แต่ว่าเกิดขึ้นเมื่อ $\omega/2$ การที่ย่านเปลี่ยนแปลงมีขนาดกว้างกว่าที่พบในตัวอย่างที่ใช้หน้าต่างอื่นที่ผ่านมา ก็เพราะว่าหน้าต่างแบล็คแมนมีลอนหลักที่กว้างกว่า หน้าต่างอื่น ๆ ที่กล่าวมาแล้วนั่นเอง

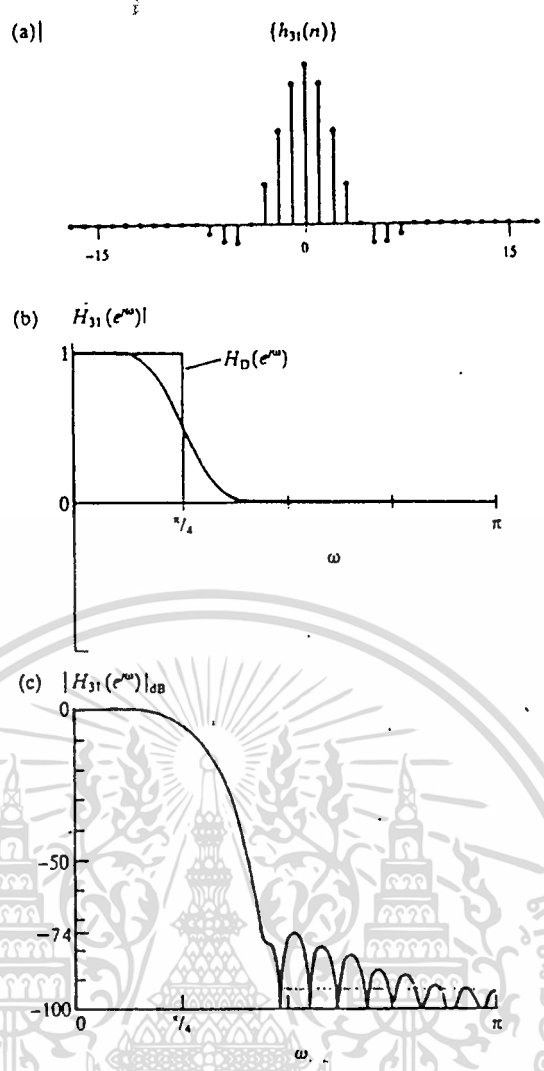


รูป 3.9 การตีความหน้าต่างแบล็คแมนในแกนความถี่

a) สเปคตรัมของฟังก์ชันแบล็คแมนซึ่งมีค่าถึงอนันต์

b) สเปคตรัมของหน้าต่างสี่เหลี่ยมที่ถูกปรับค่า และเคลื่อนไปอันเนื่องจากการคอนโวลูชั่น

c) สเปคตรัมของหน้าต่างแบล็คแมน ซึ่งเป็นผลบวกของแต่ละสเปคตรัม



รูป 3.10 ผลจากการประยุกต์หน้าต่างแบล็คแมนซึ่ง N=31

a) ค่าตอบสนองค่าหนึ่งซึ่งมีจำนวนสิ้นสุด

b) การตอบสนองความถี่ของทรานส์เฟอร์ฟังก์ชันเทียบกับอุดมคติ

c)การตอบสนองความถี่ของขนาดเป็นดีบี

ตารางที่ 1 คุณสมบัติสเปคตรัมของหน้าต่างจำนวน N ค่า

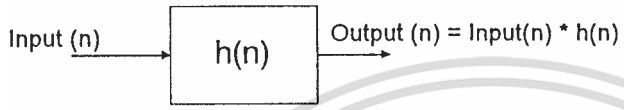
หน้าต่าง	ความกว้างของลอนหลัก	ระดับสูงสุดของลอนข้างเคียงเป็นดีบี
สี่เหลี่ยม	$4\pi/N$	-13
บาร์เล็ทท์	$8\pi/N$	-25
แฮนนิ่ง	$8\pi/N$	-31
แฮมมิ่ง	$8\pi/N$	-41
แบล็คแมน	$12\pi/N$	-57

บทที่ 4

หลักการทำงานและโครงสร้างของโปรแกรม

4.1 แนวความคิด

ฟังก์ชั่นที่เป็นหัวใจสำคัญที่สุดของโปรแกรมดิจิตอลฟิลเตอร์ของรายงานนี้คือ ฟังก์ชั่นที่ทำหน้าที่เลียนแบบการทำงานของการคอนโวลูชั่นของ 2 สัญญาณ ดังรูปที่ 4.1



รูป 4.1

จากสมการดิฟเฟอเรนเชี่ยลอีเควชั่น

$$y(n) = \sum b_q * x(n-q) - \sum a_p * y(n-p) \quad q = 0..Q-1,\ p = 0..P-1$$

เนื่องจากเป็นฟิลเตอร์แบบ FIR นั่นคือไม่มีส่วนป้อนกลับเพราะฉะนั้นจึงเหลือสมการ

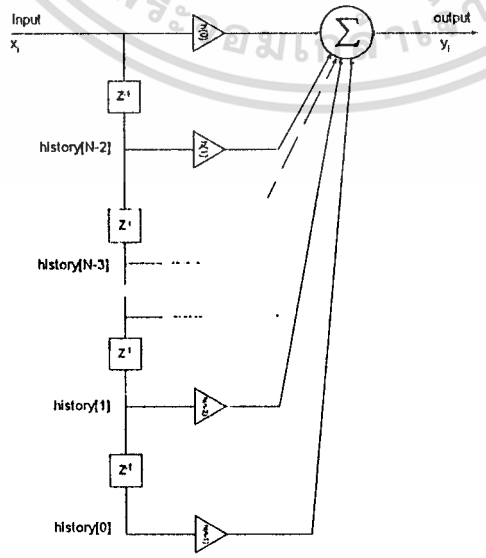
$$y(n) = \sum b_q * x(n-q) \quad q = 0..Q-1$$

ในรูปแบบ Z-transform

$$Y(z) = X(z) * \sum b_q * z^{-q} \quad q= 0..Q-1$$

$$H(z) = \frac{Y(z)}{X(z)} = \sum b_q * z^{-q}$$

สามารถเขียนสมการที่ได้อยู่ในรูปแบบของไดเร็กฟอร์มวัน (Direct Form I) ได้ดังรูป 4.2



รูป 4.2

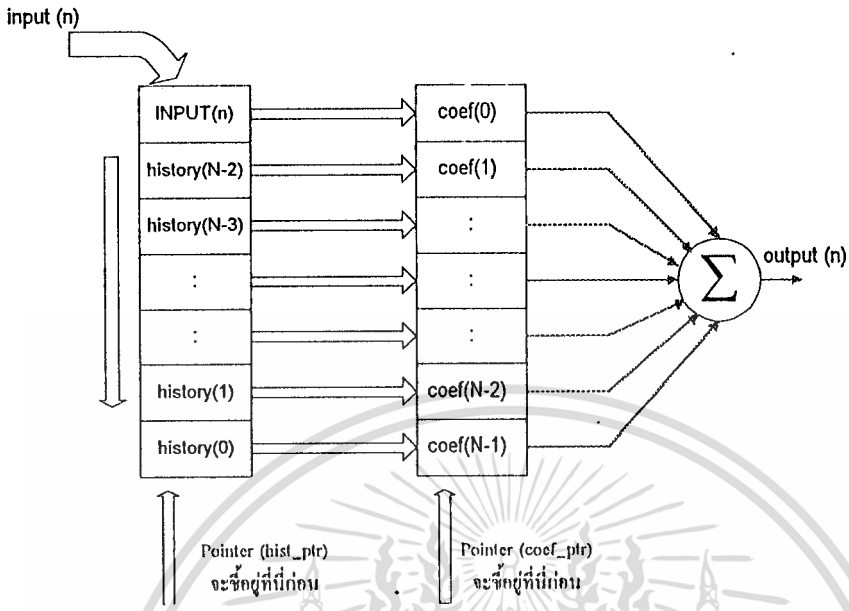
```
/*
float fir_filter(float input,FILTER *fir)
    float input         new float input sample
    FILTER *fir         pointer to FILTER structure
Returns float value giving the current output.
Allocation errors cause an error message and a call to exit.
*/
float fir_filter(input,fir)
    float input;         /* new input sample */
    FILTER *fir;         /* pointer to FILTER structure */
{
    int i;
    float *hist_ptr,*hist1_ptr,*coef_ptr;
    float output;

/* allocate history array if not already allocated */
    if(!fir->history) {
        fir->history = (float *) calloc(fir->length-1,sizeof(float));
        if(!fir->history) {
      print_error ("Unable to allocate history array in fir_filter");
            exit(1);
        }
    }

    hist_ptr = fir->history;
    hist1_ptr = hist_ptr;                    /* use for history update
*/
    coef_ptr = fir->coef + fir->length - 1; /* point to last coef */

/* form output accumulation */
    output = *hist_ptr++ * (*coef_ptr--);
    for(i = 2 ; i < fir->length ; i++) {
        *hist1_ptr++ = *hist_ptr;            /* update history array
*/
        output += (*hist_ptr++) * (*coef_ptr--);
    }
    output += input * (*coef_ptr);           /* input tap */
    *hist1_ptr = input;                      /* last history */

    return(output);
}
```



รูป 4.3

ขั้นตอนการทำงานของฟังก์ชั่น (ดูจากรูป 4.3) นี้คือ :

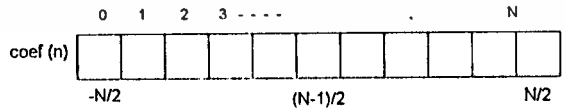
1. ประกาศตัวแปรต่าง ๆ ที่เกี่ยวข้อง
2. จองหน่วยความจำสำหรับเก็บค่า history ของสัญญาณ [เสมือนเก็บค่าของสัญญาณที่มาก่อนหน้านี่เอาไว้]
3. นำข้อมูลที่อยู่ใน history แรกสุดคูณกับโคเอฟฟิเซียน (coefficient) ตัวท้ายสุด
4. บวกตำแหน่ง history ไป 1 ลบตำแหน่งโคเอฟฟิเซียนไป
5. วนลูปคูณกันไปจนครบ
6. นำอินพุทที่รับเข้ามาใหม่คูณกับโคเอฟฟิเซียนตัวแรกสุด
7. นำค่าอินพุทที่เข้าใหม่เก็บไว้ใน history ตัวท้ายสุด
8. เอาท์พุทที่ได้คือผลของการคอนโวลูชั่น

4.2 ฟังก์ชั่นที่ทำหน้าที่ในการออกแบบฟิลเตอร์

```
int clpf (unsigned fp, unsigned N, float *coef)
{
      float wdp, wsampling, twopi, ratio, pi;
      unsigned  i, a, b, oe;
      twopi = 8*atan(1);
      pi = twopi / 2;
      ratio = (float) fp/sys.sampling;
      wdp = twopi*ratio;
      oe = odd_even (N);
      if (oe == ODD)  a = b = (N-1) / 2;
      else  a = b = N/2;
      for (i = 0; i < b; i++,a--)   {
            coef[i] = 1/(pi*a)*sin(a*wdp);
      }
      if (oe == ODD) {
            coef[i] = (float) wdp / pi;
            i++;
      }
      a = 1;
      for (;i < N; i++,a++)   {
            coef[i] = 1/(pi*a)*sin(a*wdp);
      }
      return 1;
}
```

จากทฤษฎีที่ได้กล่าวมาแล้วจะได้ฟูเรียร์ซีรีย์ของไอเดียโลพาสฟิลเตอร์คือ

$$h(n) = 1/(2pi*n)*\sin(2pi*n*fp/fs)$$

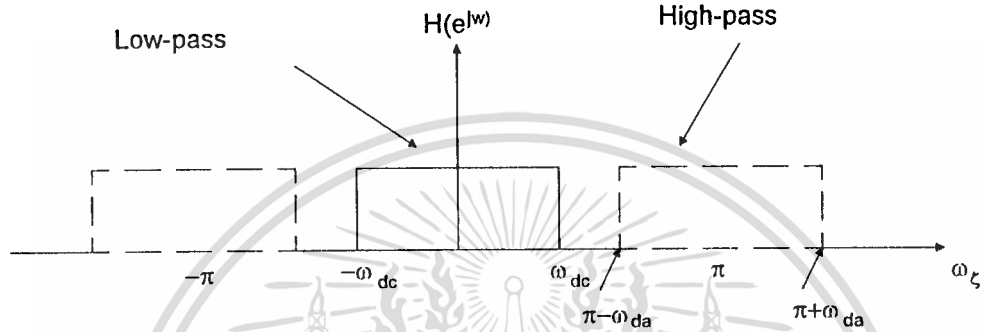


รูป 4.4

จากรูป 4.4 เนื่องจากอาเรย์จะเริ่มต้นจากจุด 0 โดยเริ่มต้นตัวแปร a=-(N-1)/2 และคำนวนค่าโคเอฟฟิเซียนออกมาเก็บในตำแหน่งอาเรย์ที่ 0 และจะลดค่า a ไปทีละ 1 ตำแหน่ง พร้อมคำนวนใส่ค่าโคเอฟฟิเซียนเพิ่มไปเรื่อย ๆ จนถึงตำแหน่ง (N-1)/2 ซึ่ง ณ. จุดนี้ คือจุดที่ทำให้สมการ

ได้ค่าอนันต์ฉะนั้นถ้าใช้ L'Hopital เข้าไปจะได้ coef(j) = w / pi และก็จะหาในส่วนที่ a เป็นบวก จนถึง (N-1)/2

ในส่วนการออกแบบ HPF, BPF, BSF ก็ใช้วิธีการ ชิพฟรีเควนซี่ (shift frequency) เช่น ต้องการ่รออกแบบไฮพาสท์ฟิลเตอร์จากโลพาสท์ฟิลเตอร์



รูป 4.5

จากรูป 4.5 เราจะเริ่มต้นออกแบบจากการตอบสนองความถี่ต่ำก่อนโดยกำหนดให้ $w_{dc} = w_{da}$ หรือ $H_{lp}(e^{jwd})$ นั่นคือถ้าเราให้ $w_{dc} = w_{da}$ และเลื่อนผลตอบสนองทางความถี่ของโลว์พาสท์ฟิลเตอร์โดยใช้ p ก็จะได้คุณลักษณะของไฮพาสท์ออกมา

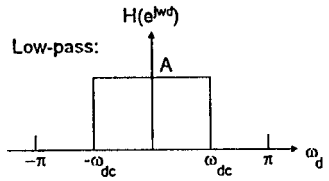
$$H_{lp}(e^{jwd}) = H_{lp}(e^{j(wd-p)}) = S\, h_{lp}(n) * e^{j(wd-p)} \qquad n = -w \,..\, a$$

$$= S\, h_{lp}(n) * e^{jnp} * e^{-jnwd}$$

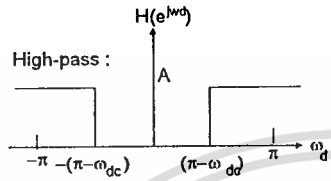
นั่นคือ

$$h_{hp}(n) = h_{lp}(n) * e^{jnp} = (-1)^n * h_{lp}(n)$$

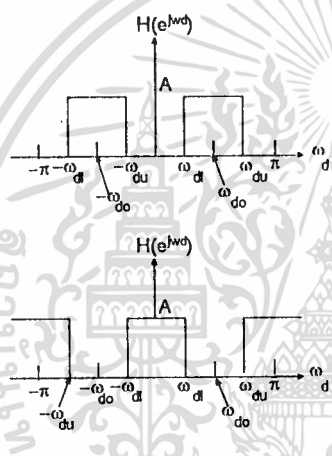
ข้อมูลที่ได้จากการออกแบบโคเอฟฟิเซียนจะถูกเก็บไว้ในชื่อต่างๆ อย่างอัตโนมัติ เช่นถ้า LPF ก็จะใช้ชื่อ "DLPF.FIR" ในส่วนฟิลเตอร์ชนิดอื่น ๆ ก็เพียงแต่เปลี่ยนเป็น "Dxxx.FIR" เท่านั้น



$$h_{lp}(n) = \frac{A \sin(n\,\omega_{dc})}{\pi n} \qquad n = \ldots -1, 0, 1, 2,$$



$$h_{hp}(n) = (-1)^n h_{lp}(n) \qquad n = \ldots -1, 0, 1, 2$$



$$h_{bp}(n) = [2\cos(n\,\omega_{do})]\, h_{lp}(n)$$

$$\omega_{du} - \omega_{dl} = 2\omega_{dc}$$

$$\omega_{do} = \frac{\omega_{du} + \omega_{dl}}{2}$$

$$h_{bs}(0) = A - h_{bp}(0)$$

$$h_{bs}(n) = -h_{bp}(n)$$

$$\omega_{du} - \omega_{dl} = 2\omega_{dc}$$

$$\omega_{do} = \frac{\omega_{du} + \omega_{dl}}{2}$$

รูป 4.6 แสดงตาราง Transform ต่าง ๆ

4.3 การรับข้อมูลจากซาวน์การ์ด

เนื่องจากต้องการให้เกิดความยืดหยุ่นสูงในการพัฒนาโปรแกรม และสามารถนำเอาโปรแกรมไปใช้งานยังสถานที่ต่าง ๆ สำหรับผู้ที่มีความสนใจ การที่จะพัฒนาฮาร์ดแวร์ขึ้นเพื่อรองรับข้อมูลอินพุทในรูปแบบต่าง ๆ จะไม่เป็นการสะดวกต่อการนำไปใช้งาน และในปัจจุบันเครื่องคอมพิวเตอร์ส่วนใหญ่ (PC compatible) จะติดตั้งอุปกรณ์ซาวน์การ์ดมาพร้อม จึงเป็นการสะดวกและรวดเร็วถ้าเราสามารถใช้ทรัพยากรที่มีอยู่ให้เกิดประโยชน์ได้สูงสุด จึงได้เขียนโปรแกรมให้มีความสามารถในการอ่านข้อมูลจากซาวน์การ์ดได้โดยตรง หรือจะอ่านจากไฟล์ที่ได้ผ่านการบันทึกอยู่ในรูปแบบ (*.voc) ก็ได้

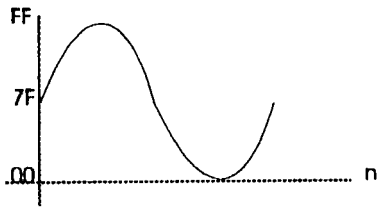
ทั้งนี้ข้อมูลที่ผ่านการประมวลจากโปรแกรมก็สามารถนำออกลำโพงของซาวน์การ์ดเพื่อเปรียบเทียบความแตกต่างระหว่างก่อนและหลังประมวลผล(ยกเว้นข้อมูลที่สร้างจาก wave generator จะไม่สามารถ เซฟ (save) ได้ แต่ก็สามารถทำได้โดยเรียกใช้โปรแกรม sweep.exe ในการสร้างข้อมูลแบบ *.voc ออกมาและใช้ฟังก์ชั่น โหลด (load) ของโปรแกรมฟิลเตอร์เรียกข้อมูลมาประมวลผลต่อไป แต่เนื่องจากทั้งโปรแกรมเป็นเพียง ทูล(tools) ที่ช่วยเหลือเท่านั้นจึงยังมีข้อผิดพลาดในการทำงานบ้าง ฉะนั้นอาจจะใช้งานได้ไม่เต็มที่นัก)

4.4 รูปแบบรหัสข้อมูลที่ได้จากซาวน์การ์ด

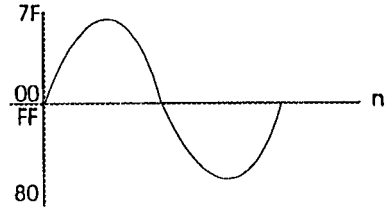
ไฟล์ที่ได้จากการบันทึกจากไมโครโฟนหรือ ไลน์อิน (Line in) จะถูกบันทึกอยู่ในขนาด 8 บิต การที่จะนำข้อมูลขนาด 8 บิตนี้ไปประมวลผลโดยตรงจะได้ผลลัพธ์ที่ไม่ถูกต้องเนื่องจากค่าโคเอฟฟิเซียน ที่คำนวนได้เป็นเลข โฟลติงพอยท์ (floating point) ถ้าใช้วิธีปัดเศษ (round) เลข โฟลติงพอยท์ให้เหลือเพียง 8 บิต ก็จะทำให้เกิดข้อผิดพลาดขึ้นมากมาย แต่ได้ความเร็วในการประมวลผลสูงขึ้น แต่ถ้าใช้วิธีการแปลงข้อมูล 8 บิตจากซาวน์การ์ดนี้ไปอยู่ในรูปโฟลติงพอยท์จะได้ผลลัพธ์ที่ถูกต้องกว่าแต่ใช้เวลาในการโปรเซสสูงและใช้หน่วยความจำจำนวนมาก ซึ่งใน โครงงานนี้ได้ใช้วิธีหลังเพื่อรักษาความถูกต้องเอาไว้ให้มากที่สุด วิธีการแปลงก็โดยอาศัยหลักการคิดที่ว่าสัญญาณอินพุทต่าง ๆ ที่ป้อนเข้ามาก่อนที่จะถูกซาวน์การ์ดจัดระดับก็จะอยู่ในระดับ 0V - 5V

เพราะฉะนั้นการแปลงข้อมูล 5V ให้อยู่ในระดับ 256 ระดับนั่นคือ 1 ระดับจะมีค่าเท่ากับ 5/256 = 0.01953125 V เช่นถ้ามีข้อมูลของซาวน์การ์ดเป็น 0FH ก็จะแปลงได้เป็น 15*0.01953125 = 0.29296875 V

ทำการแปลงแบบนี้กับข้อมูลที่อ่านได้ทั้งหมดซะก่อนจึงจะนำไปประมวลผลต่อได้จึงเป็นเหตุผลหนึ่งว่าทำไมจึงไม่สามารถทำฟิลเตอร์แบบ เรียลไทม์ (real time) ได้ เนื่องจากต้องเสียเวลาในการแปลงและต้องดำเนินการทางคณิตศาสตร์เป็นเลขโฟลติงพอยท์ (32 bits) นี่เอง



รูป 4.7 แสดงไฟล์ข้อมูลที่ได้จากซาวน์การ์ด



รูป 4.8 แสดงข้อมูลที่ผ่านการแปลงเป็น 2's คอมพลีเมนท์แล้ว

```
void far *read_data (VOC_FILE *voc)
{
      char far *buffer;
      unsigned long x;
      if (tbuffer != NULL) farfree (tbuffer);
      if (pbuffer != NULL) farfree (pbuffer);
      buffer = farmalloc (voc->length_data-2);
      if (buffer == NULL)
            {
            print_error ("Out of memory for large data");
            return NULL;
            }
      tbuffer = farcalloc (voc->length_data-2, sizeof (float));
      if (tbuffer == NULL)
            {
            print_error ("Out of memory for large data");
            return NULL;
            }
      pbuffer = farcalloc (voc->length_data,sizeof (float));
      if (pbuffer == NULL)
            {
            print_error ("Out of memory for large data");
            return NULL;
            }
      fread ((void far*) buffer, voc->length_data-2, 1, voc->fp);
      for (x=0;x < voc->length_data-2;x++) {
            tbuffer[x] = (buffer[x] - 0x80) * STEP;     <-----------
ส่วนที่ทำหน้าที่แปลง
      }
      farfree (buffer);
      return (tbuffer)
}
```

4.5 โปรแกรมที่ใช้ช่วยในการพัฒนา

เพื่อความสะดวกในการตรวจสอบและเพื่อการพัฒนาโปรแกรมได้รวดเร็วจึงได้เขียนโปรแกรม เพื่อช่วยในการสร้างไฟล์ข้อมูลออกมาในรูปของ (*.voc) โดยจะทำการสร้างรูปคลื่นต่าง ๆ ที่ต้องการออกมาให้ โดยมี 3 โปรแกรมคือ

1. stgen.exe ทำการสร้างรูปคลื่น ไซน์ (sine) เพียง 1 ลูกจากความถี่ที่เรากำหนดให้

2. genvoc.exe ทำการสร้างผลบวกของรูปคลื่นไซน์โดยสามารถกำหนดจำนวนลูกคลื่นและความถี่ของแต่ละลูกได้

3. sweep.exe จะทำการสร้างความถี่ตั้งแต่ 0 Hz จนถึงความถี่ sampling/2

ส่วนโปรแกรมสำเร็จรูปที่ใช้ในการจำลองต่าง ๆ ก็มีหลายโปรแกรมเช่น MCAD, SIGSYS และอื่น ๆ ในส่วนของโปรแกรม sigsys ก็ดูในส่วนคู่มือการใช้งานสามารถดูได้จากส่วนอ้างอิงของโครงงาน

4.6 การทำงานของฟังก์ชั่น SWEEP

เนื่องจากเราต้องการให้ฟังก์ชั่นทำหน้าที่ผลิตความถี่ตั้งแต่ 0 ถึง fs/2 ถ้าพิจารณาฟังก์ชั่นที่ทำหน้าที่ผลิตความถี่เดี่ยวๆ กันก่อน

$$a = \sin\left(\frac{\pi}{2} \times \frac{f}{fs} \times n\right)$$

f : คือความถี่ที่เราต้องการ

fs : คือแซมบลิ้งเรท

n : ลำดับ

ถ้าเราต้องการให้ฟังก์ชั่นนี้ผลิตความถี่เดียวก็ให้อัตราส่วน f/fs เท่ากับค่าคงที่ แต่ถ้าเราต้องการให้ฟังก์ชั่นนี้ผลิตความถี่ออกมาหลาย ๆ ความถี่ก็เพียงแต่ให้ f เปลี่ยนแปลงไปตามลำดับที่เปลี่ยนแปลงตามเวลา โดยที่ให้ลำดับเริ่มต้นจาก 0, 1, 2, ... ก็จะได้ f เปลี่ยนแปลงไป เพื่อป้องกันการผิดเพี้ยนความถี่ f จึงไม่ควรเกิด fs/2

4.7 ตัวแปรต่าง ๆ ที่เกี่ยวข้องกับระบบ

```
typedef struct {
      unsigned long sampling;          เก็บค่า sampling rate
      unsigned long length;            เก็บความยาวของ block ของข้อมูล
      unsigned int softwaregen;        ใช้บอกว่าเป็นข้อมูลที่สร้างขึ้นมา
      unsigned int time_freq;          ใช้บอกว่าต้องการประมวลผลทาง
                                       ด้านใด
      unsigned int best_run;
      unsigned int ptime_freq;
      unsigned int methord;            ใช้กำหนดว่าเป็นฟิลเตอร์แบบใด
                                       (default FIR)
      unsigned int type_fil;           ใช้บอกว่าเป็นฟิลเตอร์ชนิดใด
      int  zoom;                       ใช้บอกว่าเป็นการ zoom in , zoom out
      int  zoom_value;                 ใช้บอกว่า zoom ด้วยค่าเท่าใด
      char *gtitle;          /* title for graphics mode */
      float *coef;           pointer ชี้ไปที่เก็บ coefficient
} System;                    ตัวแปรโครงสร้างของระบบ


typedef struct filter {
       unsigned int length;    /* size of filter */
       float far *history;     /* pointer to history in filter */
       float far *coef;        /* pointer to coefficients of filter */
} FILTER;
```

float far *tbuffer; เป็นตัวชี้ (pointer) จะใช้ชี้ว่าข้อมูลจากซาวน์การ์ดและแปลงเรียบร้อยแล้วถูกเก็บอยู่ที่ใด

float far *pbuffer; เป็นตัวชี้จะใช้ชี้ว่าข้อมูลที่ผ่านการประมวลผลถูกเก็บอยู่ที่ใด

4.8 หลักการและเทคนิคการใช้ซาวน์การ์ด

จะอ้างอิงถึง ซาว์ดบลัสเตอร์ (SOUND BLASTER) เป็นหลักเนื่องจากเป็นซาวน์การ์ดที่ได้รับความนิยมสูงสุดในขณะนี้ หลังจากที่เราจัดการติดตั้งซาวน์การ์ดของเราแล้วส่วนใหญ่ก็จะมีซับไดเร็กทรอรี่ \SB โผล่ออกมาให้เห็น ซึ่งถ้าดูในนั้นแล้วก็จะเห็นว่ามี \SB\DRV\CT-VOICE.DRV อยู่ด้วยไดรเวอร์ ตัวนี้แหละที่เราจะต้องติดต่อผ่านเพื่อควบคุมการทำงานของซาวน์การ์ดอีกที

วิธีการโหลดไดร์เวอร์นี้ก็มีหลักการอยู่ที่ว่ามันจะอยู่เซกเมนท์ใด ๆ ก็ได้ แต่ต้องมีออฟเซ็ทอยู่ที่ 0x0000 เท่านั้นเมื่อเราสามารถโหลดไดร์เวอร์ตัวนี้ได้สำเร็จแล้วก็เพียงแต่เรียกใช้ฟังก์ชั่นต่าง ๆ ที่มีให้เท่านั้น

4.9 รูปแบบไฟล์ *.VOC

รูปแบบไฟล์มาตราฐานของซาวน์บลัสเตอร์นั้น เสียงต่าง ๆ จะถูกบันทึกอยู่ในรูปแบบ *.VOC ซึ่งมีรายละเอียดสำคัญอยู่ 2 ส่วน

1. เฮดเดอร์บล็อก (Header Block)
2. ดาต้าบล็อก (Data Block)

เฮดเดอร์บล็อก : จะประกอบไปด้วย

- ไอเด็นติไฟล์เออร์ (identifier)
- หมายเลขเวอร์ชั่น (version)
- พอทย์เตอร์ที่ชี้ไปที่ดาต้าบล็อกแรก
- เช็คโค้ด (Check code) ใช้ตรวจสอบว่าเป็นรูปแบบ *.voc จริงหรือไม่

ดาต้าบล็อก : จะแบ่งเป็นซับบล็อก (sub block) ย่อย ๆ ซึ่งความหมายของซับบล็อกแต่ละชุดจะบรรจุข้อมูลต่าง ๆ เช่น วอยซ์ดาต้า (voice data), ไซด์เร็นซ์ (silence), มาร์คเกอร์(marker), แอสกี้เทกซ์ (ASCII text),รีพีต (repeat) และ เอนรีพีต (end repeat)

* ได้ประกาศเฮด์เดอร์ไว้แล้วในไฟล์ sbcvoice.h โดยเช็คโค้ดที่ประกาศไว้จะต้องนำเอาเวอร์ชั่น + 1234H = 1129H ก็แสดงว่าเป็นรูปแบบ *.voc จริง

ในไบต์แรกของซับบล็อกจะเรียก บล็อกไธย์ (block type) (BLKTYPE) ถัดมาอีก 3 ไบต์เรียกบล็อกเรนท์ (block length) (BLKLEN) และข้อมูลถัดจาก BLKLEN อาจจะเป็น เสียง, หรืออัตราการแซมบลิ้ง, ไทม์คอนสแตนท์ (Time Constant) ก็ขึ้นอยู่กับความหมายของบล็อกไธย์นั้น ๆ ถ้า

BLKTYPE = 0 : เทอร์มิเนเตอร์ (Terminator) เป็นการชี้ให้เราทราบว่าข้อมูลเสียงของไฟล์นั้น ๆ หมดแล้ว

BLKTYPE = 1 : วอทซ์ดาต้า

01	BLKLEN	TC	PACK	VOICE DATA .....

BLKLEN : ความยาวของวอทซ์ดาต้า (voice data = BLKLEN - 2)

TC : ไทม์คอนสแตนท์ (TC = 256 - (1000000/sampling rate))

PACK : = 0 ข้อมูลวอทซ์ธรรมดาไม่ได้ แพ็ค(pack) เอาไว้

= 1 4 bit packed

= 2 2.6 bit packed

= 3 2 bit packed

BLKTYPE = 2 : วอทซ์คอนทินูเอชั่น (Voice Continuation)

2	BLKLEN	VOICE DATA .....

BLKTYPE = 3 : ไซด์เร็นซ์

03	BLKLEN	PERIOD	TC

จะแทนที่ข้อมูลของเสียงช่วงเงียบด้วยข้อมูลเพียง 2 ไบต์ พีเรียด(period) ส่วน BLKLEN จะถูกเซตให้มีค่า 3 ตลอดเวลา

BLKTYPE = 4 : มาร์คเกอร์

04	BLKLEN	MARKER

BLKLEN ของชนิดนี้จะถูกเซตให้มีค่า 2 เสมอ

มาร์คเกอร์จะมีค่าได้ตั้งแต่ 0x0001 - 0xFFFE โดย 0x0000 และ 0xFFFF จะถูกสงวนไว้ใช้กับ CT-VOICE ไดร์เวอร์

จะใช้เพื่อเป็นสัญลักษณ์ในระหว่างที่เราให้ข้อมูลวอทซ์ดาต้าออกทางเอาท์พุทอยู่นั้น เนื่องจากค่ามาร์กเกอร์จะถูกนำไปเก็บไว้ในตัวแปร สเตตัสเวิร์ด(status word) ทำให้เราสามารถตรวจสอบได้ว่าตอนนี้ซาวด์การ์ดใช้ข้อมูลถึงไหนแล้ว

BLKTYPE = 5 : แอสกี้เทกซ์

05	BLKLEN	ASCII TEXT	NULL

ใช้เก็บข้อความ เช่น ชื่อเพลง

BLKTYPE = 6 : **รีพีทลูป** (Repeat Loop)

06	BLKLEN	COUNT

จะใช้เป็นตัวบอกจุดเริ่มต้นที่จะลูปข้อมูลระหว่างจุดนี้กับจุดเอนลูป (end loop) โดยจะลูปเป็นจำนวน COUNT + 1 ครั้ง และมีค่าได้ตั้งแต่ 0x0001 - 0xFFFE ถ้าเราเซตให้มีค่า 0xFFFF จะทำให้เกิดการลูปไม่รู้จบ

BLKTYPE = 7 : **เอนรีพีทลูป** (End Repeat Loop)

07	BLKLEN

เป็นจุดบอกการสิ้นสุดของข้อมูลที่จะลูปซึ่งโดยปกติ BLKLEN จะถูกเซตเป็น 0 เสมอ

4.10 สเตตัสเวิร์ด

จะใช้เพื่อรายงานสภาวะต่าง ๆ ที่เกิดขึ้น โดยสรุปจะเปลี่ยนแปลงเมื่อ

1. จะเป็น 0 ระหว่างการอินทิเชียลไลท์เซชั่น (initialization)
2. เป็น 0xFFFFH ระหว่างการเริ่มต้นเกี่ยวกับอิทพุท/เอาท์พุทวอทซ์
3. เป็น 0 เมื่อบัฟเฟอร์ (buffer) ของข้อมูลเต็มหรือใช้หมดแล้วในกรณีเอาท์พุท
4. เปลี่ยนตามค่ามาร์คเกอร์ที่อ่านได้จากข้อมูลในซับบล็อค

สำหรับโปรแกรมเมอร์ภาษาซี ถ้าเราต้องการที่จะตรวจสอบตัวแปรนี้เราจำเป็นต้องเพิ่ม directive #pragma loop_opt(off) ไว้เหนือโปรแกรมย่อย ที่เราจะตรวจสอบเพื่อบอกยกเลิกการ**ลูปออฟติไมท์** (loop optimize) ของคอมไพล์เลอร์ (complier)

4.11 ฟังก์ชั่นของ CT-VOICE.DRV

จะมีให้ 16 ฟังก์ชั่น ต่าง ๆ ดังนี้

1. unsigned ctvm_version() จะรีเทอร์เวอร์ชั่น (return version) ของไดร์เวอร์มาให้โดย เมเจอร์เวอร์ชั่นนับเบอร์ (Major version number) จะอยู่ใน HIWORD, และ ไมเนอร์เวอร์ชั่นนับเบอร์ (Minor version number) จะอยู่ใน LOWORD

2. void ctvm_set_port_addr (unsigned port) เมื่อเราต้องการเปลี่ยนค่า พอล์ต (port) ของซาวท์การ์ดโดย

SBPRO และ SB20 มีพอล์ตให้คือ 220H, 240H

SBMCV และ SB มีพอล์ตให้คือ 210H, 220H, 230H, 240H, 250H และ 260H

โดยปกติทางโรงงานจะติดตั้งให้มี แอดเดรส (address) 220H

3. void ctvm_set_interrupt (unsigned interruptnum) สำหรับเซทค่า อินเทอรัป (interrupt) เพื่อประโยชน์ในการใช้ ดีเอ็มเอ (DMA) ของซาว์ทการ์ดโดย

SBPRO มีอินเทอรัปให้เลือกได้คือ 2, 5, 7, 10

SB20, SBMCV, SB มีอินเทอรัปให้เลือกได้คือ 2, 3, 5, 7

โดยปกติทางโรงงานจะติดตั้งให้เป็นอินเทอรัปนับเบอร์ 7

4. int ctvm_init () จะทำการตรวจเช็คซาวท์การ์ดและติดตั้งระบบต่าง ๆ ให้แก่การ์ดภายหลังจากที่ทำการติดตั้งสำเร็จ ดีเอซี (DAC) ลำโพง จะถูกเซตให้ ออน (on)

return value 0: ถ้าทำการอินทิเชียลไลท์เซชั่นสำเร็จจะคืนค่า 0 มาให้แต่ถ้าไม่ใช่ก็หมายถึง

1: Incorrect driver version

2: I/O read/write failure

3: interrupt for DMA failure

ควรเรียกใช้ฟังก์ชั่นนี้ทุกครั้งหลังจากการเรียกใช้ฟังก์ชั่น ctvm_set_interrupt และ ctvm_set_port_addr

5. void ctvm_speaker (int on_off_flag) ถ้าให้ on_off_flag = 0 จะทำให้ ดีเอซี (ลำโพงเป็นออฟ) แต่ถ้าเป็น 1 ดีเอซี จะออน เราควรจะออฟมันก่อนในขณะที่เรากำลังใช้ฟังก์ชั่น เร็กคอร์ด (record) และก่อนออกจากโปรแกรมของเราควรจะเซตให้ ดีเอซีออฟ ซะก่อนด้วยอย่าลืม

6. void ctvm_set_status_word (unsigned *ct_voice_status) ใช้สำหรับตรวจสอบการทำงานของซาวท์การ์ดในกรณีต่าง ๆ (ถูกเรียกหลัง ctvm_init () อัตโนมัติแล้ว)

7. int ctvm_input (char far *buffer_ptr, unsigned long buffer_size, unsigned sampling_rate)

ฟังก์ชั่นนี้จะทำการบันทึกเสียงโดยมีรายละเอียดตัวแปรต่าง ๆ ดังนี้

sampling_rate : SBPRO 4000 Hz ถึง 44100 Hz mono
SB20 4000 Hz ถึง 15000 Hz mono
SBMCV 4000 Hz ถึง 13000 Hz mono
SB 4000 Hz ถึง 13000 Hz mono

return value 0: successful
x: non-zero fail

ฟังก์ชั่นนี้จะใช้ดีเอ็มเอในการโอนถ่ายข้อมูล และจะเซตให้ สเตตัสเวิร์ด เป็น 0xFFFFH และส่งการควบคุมให้โปรแกรมของเราทันที (ยังรับอินพุตอยู่เบื้องหลัง) แต่ฟังก์ชั่นนี้จะถูก เทอร์มิเนตเมื่อบัฟเฟอร์ เต็มหรือ ถูกเรียกโดยฟังก์ชั่น ctvm_stop () ซึ่งเมื่อเกิดเหตุการณ์เหล่านี้ขึ้นก็จะทำให้สเตตัสเวิร์ด ถูกเซตเป็น 0x0000 โดยทันที

8. int ctvm_output (char far * buffer_ptr) จะนำข้อมูลของเราออกลำโพงโดยเราให้ตำแหน่งของข้อมูลของเราไป และ buffer_ptr ที่ส่งไปให้ต้องชี้ไปที่ดาต้าบล็อคฟอร์แมต หลังจากเรียกฟังก์ชั้นนี้แล้วมันจะคือการควบคุมให้แก่เราทันที (เสียงยังออกไปเรื่องขณะที่เราทำงานอื่นอยู่) และ สเตตัสเวิร์ดจะถูกเซตให้เป็น 0xFFFFH ทันที และสเตตัสเวิร์ดจะถูกเซตเป็น 0x0000 เมื่อเราเรียกใช้ฟังก์ชั่น ctvm_stop () หรือข้อมูลนำมาใช้หมดแล้ว ในระหว่างที่เสียงไปปรากฏที่เอาท์พุทนั้นค่า สเตตัสเวิร์ดก็จะถูกเปลี่ยนแปลงไปตามค่ามาร์คเกอร์ของดาต้าบล็อคนั้น ๆ

return value 0: zero (successful)
x: non-zero (fail) อาจเกิดขึ้นเมื่อมีขบวนการอื่น ๆ กำลังทำงานอยู่

9. void ctvm_stop (void) จะทำการหยุด ทั้งอินพุทและเอาท์พุทที่มีอยู่ขณะนั้น และทำให้ค่า สเตตัสเวิร์ดเป็น 0x0000

10. int ctvm_pause (void) จะมีผลให้ลำโพงหยุดทำงานไป และ สเตตัสเวิร์ดจะเก็บค่าสุดท้ายที่ทำงานค้างอยู่เอาไว้

return value 0: zero if successfully

x: non-zero ชี้ให้เห็นว่าตอนนั้นไม่มีวอท์ซเอาร์พุทที่ทำงานอยู่

11. int ctvm_continue (void) จะทำให้ผลการของ หยุดชั่วขณะ (pause) ยกเลิกไปและให้วอทซ์เอ้าทพุธต่อไป

return value 0: zero if successfully

x: non-zero ไม่ได้ถูกหยุดชั่วขณะเอาไว้

12. int ctvm_break_loop (int break_mode) ในรูปแบบ *. VOC เราสามารถสั่งให้ วอทซ์ (ดาต้า) ของเรามีการเล่นซ้ำ ๆ กันได้โดยไม่ต้องอัดเสียงซ้ำ ๆ กันหลาย ๆ ครั้ง ซึ่งฟังก์ชั่นนี้จะทำการหยุดลูปนั้นซะ (ในกรณีที่เรามีดนตรีแบบเล่นไปเรื่อย ๆ และเราต้องการหยุดเสียงนี้สักครู่เพื่อทำการอะไรบางอย่าง) ตัวแปร break_mode จะมีอยู่ 2 กรณีคือ

break_mode = 1: จะทำการหยุดเสียที่วนลูปนั้นทันทีทันใด และจะนำเอาซับบล็อค (ข้อมูลชุดต่อไปจากลูปนี้) ต่อไปออกเอาท์พุทแทน

break_mode = 0: จะรอให้การทำงานของเสียงที่ลูปนั้นจบของมันก่อนที่จะออกนำข้อมูลชุดถัดไปมาออกเอาท์พุทต่อ

return value 0:zero if successful

1: voice output ไม่ได้มีการลูป

13. void ctvm_terminate (void) ถ้าเราสามารถเรียกฟังก์ชั่น ctvm_init () สำเร็จก็จะต้องเรียก ฟังก์ชั่นนี้ควบคู่กันไปด้วยเพื่อจะทำการรีอินทิเซียลไลท์เซชั่นการ์ด (reinitializes card) และลำโพงก็จะถูกเซตให้ออฟ

14. int ctvm_get_cardtype (void)

return value 1: SB and SBMCV

2: SBPRO

3: SB20

15. unsigned long ctvm_get_ADC_range (int mode) จะคืนค่าแซมบลิ้งเรทที่ ซาวน์การ์ดนั้น ๆ สนับสนุนในการบันทึกเสียงให้ ตัวแปร mode ถ้าเป็น 0 คือ mono และ 1 ถ้าเป็น stereo ค่าที่ return จะอยู่ใน HIWORD กับ LOWORD และให้นำค่าที่รับได้นั้นไปคูณด้วย 10 ก่อน จึงจะได้แซมบลิ้งเรทจริง ๆ

16. unsigned long ctvm_get_DAC_range (int mode) จะคือค่า แซมบลิ้งเรท ที่ sound card นั้น ๆ สนับสนุนในการ play voice ให้ ตัวแปร mode ถ้าเป็น 0 คือ mono และ 1 ถ้าเป็น stereo ค่าที่ return จะต้องใน HIWORD กับ LOWORD และให้นำค่าที่รับได้นั้นไปคูณด้วย 10 ก่อนจึงจะได้แซมบลิ้งเรทจริง ๆ

4.12 คำแนะนำในการพัฒนาโปรแกรมต่อไป

โปรแกรมนี้มีปัญหาเกี่ยวกับหน่วยความจำเต็มอยู่บ่อย ๆ เนื่องจากใช้วิธีการจองหน่วยความจำจาก ฮีพ (heap) ฉะนั้นถ้าต้องการให้ปัญหาเหล่านี้หมดไปควรจะทำการจองหน่วยความจำโดยใช้ฟังก์ชั่นของ ดอส (DOS) ซึ่งสามารถจองได้มากแต่ก็จะมีปัญหาในเรื่องการจัดการพอสมควร ในส่วนการพัฒนาความสามารถของโปรแกรมนี้ อาจจะพัฒนาในส่วนของ IIR เพิ่มเติม เนื่องจากมีความรวดเร็วกว่าและใช้ฟังก์ชั่นเดิมได้เพียงแต่ต้องออกแบบส่วนออกแบบฟิลเตอร์ชนิด IIR เพิ่มเติมเท่านั้น

การพัฒนาโปรแกรมให้มีความสามารถทางด้านเรียลไทม์ก็ยังสามารถที่จะกระทำได้โดยการทำการประมวลผลเพียง 8 บิต และใช้หลักการ ฟาสท์คอนโวลูชั่น (Fast Convolution) เข้ามาช่วย ถึงแม้ว่าโปรแกรมนี้ก็มีความสามารถทางฟาสท์คอนโวลูชั่นแต่เนื่องจากโครงสร้างยังไม่ดีเพียงพอจึงทำงานได้ช้ากว่าที่ควรจะเป็นและยังทำงานได้ไม่ถูกต้องนัก

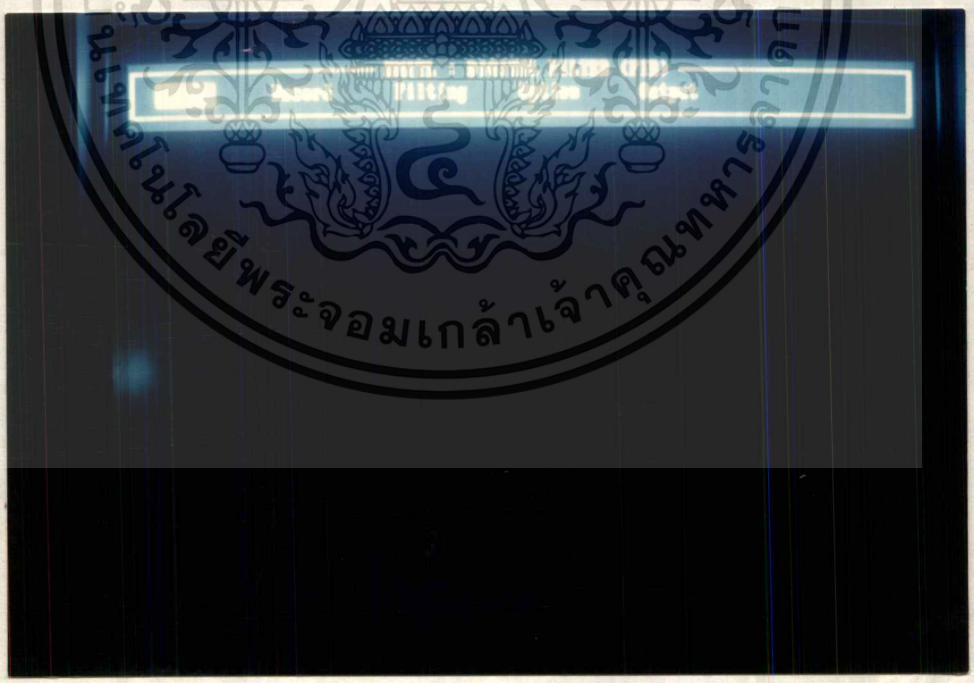
บทที่ 5

หลักการใช้งานซอร์ฟแวร์

ซอร์ฟแวร์จะใช้ระบบอินเตอร์เฟส แบบพูล ดาวน์ เมนู โดยมีส่วนประกอบต่าง ๆ ดังนี้

- File ใช้เกี่ยวกับการเปิด (Open),เก็บ(Save) ไฟล์ต่าง ๆ
- Record ใช้สำหรับทำการบันทึกข้อมูลหรือสัญญาณจากไมค์
- Filtering ใช้สำหรับกำหนดว่าจะนำข้อมูลที่ได้มาทำการประมวลผล(process) แบบใด (LPF,HPF)
- Option ใช้กำหนดคุณสมบัติเพิ่มเติมต่าง ๆ
- Output ใช้สำหรับแสดงผลออกทางจอแสดงผล หรือการ์ดเสียง(Sound Card)

ซึ่งจะใช้รูปแบบไฟล์(File format) นามสกุล *.VOC ของครีเอทีฟ รูปแบบของซอร์ฟแวร์ แสดงดังรูปที่ 5.1



รูปที่ 5.1 ภาพรายละเอียดส่วนต่าง ๆ ของซอร์ฟแวร์

หัวข้อต่าง ๆ ประกอบไปด้วยส่วนย่อยต่าง ๆ ดังนี้

FILE

แสดงดังรูปที่ 5.2

รูปที่ 5.2 ภาพแสดงเมนูย่อยของไฟล์

- Load เมื่อเลือกมาที่ Load แล้วกด Enter จะใช้สำหรับ Load ข้อมูลเสียง *.VOC โดยจะต้องป้อนชื่อและนามสกุลเต็มพร้อมทั้ง Path ด้วย

- Save ใช้สำหรับเก็บข้อมูลที่ผ่านการประมวลผลแล้ว

- Info ใช้สำหรับดูข้อมูลของ File ที่ Load มาว่ามีข้อมูลจำเพาะอะไรบ้าง ซึ่งเป็นข้อมูลของ VOC Data ซึ่งจะให้ใส่ชื่อ File,Time Constant และ Sampling Frequency

RECORD

แสดงดังรูปที่ 5.3

รูปที่ 5.3 ภาพแสดงเมนูย่อยของ record

- Sampling Rate เมื่อเลือกมาที่ตำแหน่งนี้แล้วกด Enter โปรแกรมจะให้ใส่ค่าแซมปลิ้ง ที่ต้องการซึ่งใช้สำหรับกำหนดความถี่ในการแซมปลิ้ง ข้อมูล (Default = 1K Sampling/s) นั่นคือจะตอบสนองความถี่ได้ถูกต้องที่ fs/2 และค่าแซมปลิ้งนี้จะต้องไม่ต่ำกว่า 400 Sampling/s ถ้าต้องการใช้กับการ์ดเสียง

- Record ในกรณีที่ต้องการบันทึกเสียงเพื่อเก็บไว้ในการวิเคราะห์ ก็สามารถใช้ฟังก์ชันนี้ในการบันทึกได้ โดยความยาวจะขึ้นอยู่กับความจุของฮาร์ดดิสก์ ค่าแซมปลิ้งจะต้องกำหนดไว้ด้วย

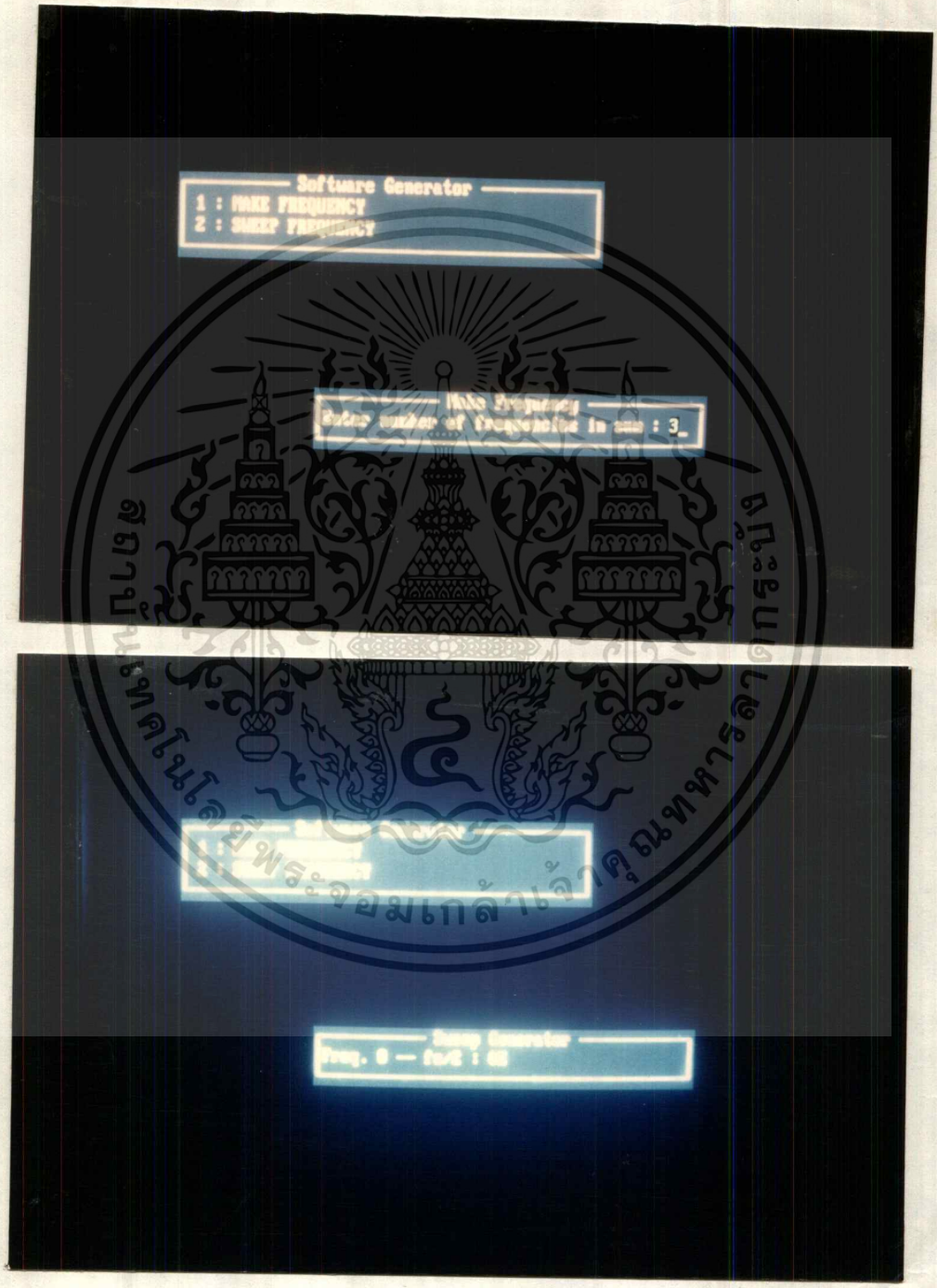
- Software Generator ในการวิเคราะห์เพื่อศึกษาคุณสมบัติต่าง ๆ ของฟิลเตอร์ถ้าใช้เสียงที่ได้จากการบันทึกผ่านสื่อต่าง ๆ จะไม่เป็นการสะดวกและให้ผลที่ชัดเจน

ดังนั้นจึงได้สร้างฟังก์ชันที่จะสร้างความถี่ต่าง ๆ ตามที่ต้องการออกมา 2 ฟังก์ชัน ซึ่งประกอบด้วยฟังก์ชันดังนี้

1. Make Frequency

2. Sweep Frequency

ดังแสดงในรูปที่ 5.4 (a),(b)



รูปที่ 5.4 แสดงเมนูย่อยของ software generator

(a) Make Frequency

(b) Sweep Frequency

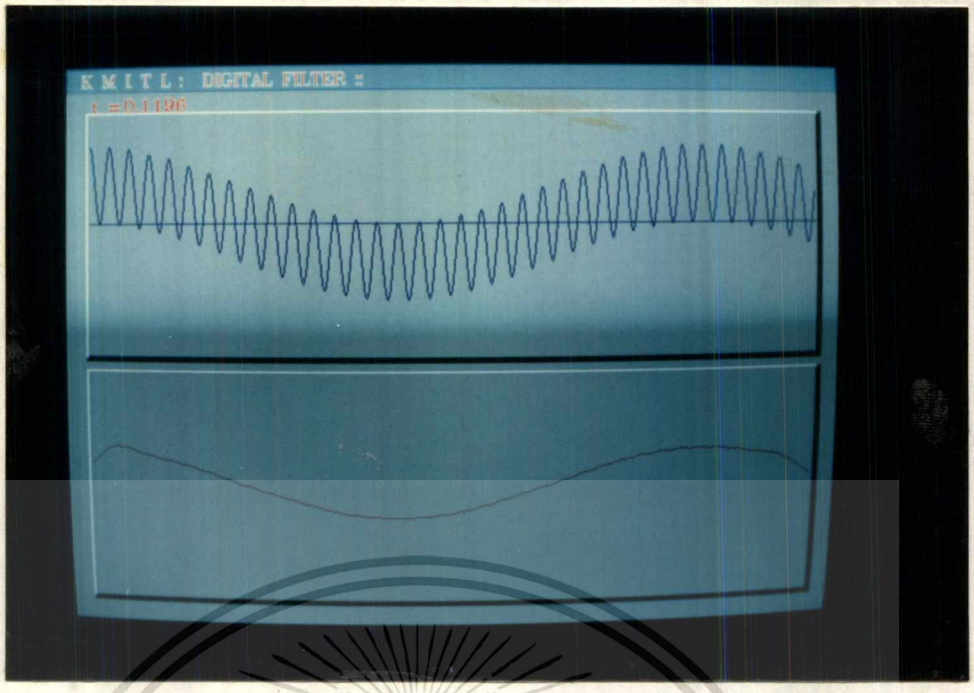
Make Frequency

ฟังก์ชันนี้จะสามารถสร้างความถี่ได้ไม่จำกัดจำนวน โดยฟังก์ชันจะสอบถามเราว่าต้องการสร้างความถี่จำนวนกี่ความถี่

เมื่อทราบจำนวนที่ต้องการแล้วก็จะให้เราป้อนความถี่ที่ต้องการเป็นตัว ๆ ไป จนครบที่เรากำหนด และโปรแกรมก็จะนำความถี่ต่าง ๆ มารวมเข้าด้วยกัน เพื่อสร้างเป็นรูปคลื่นใหม่

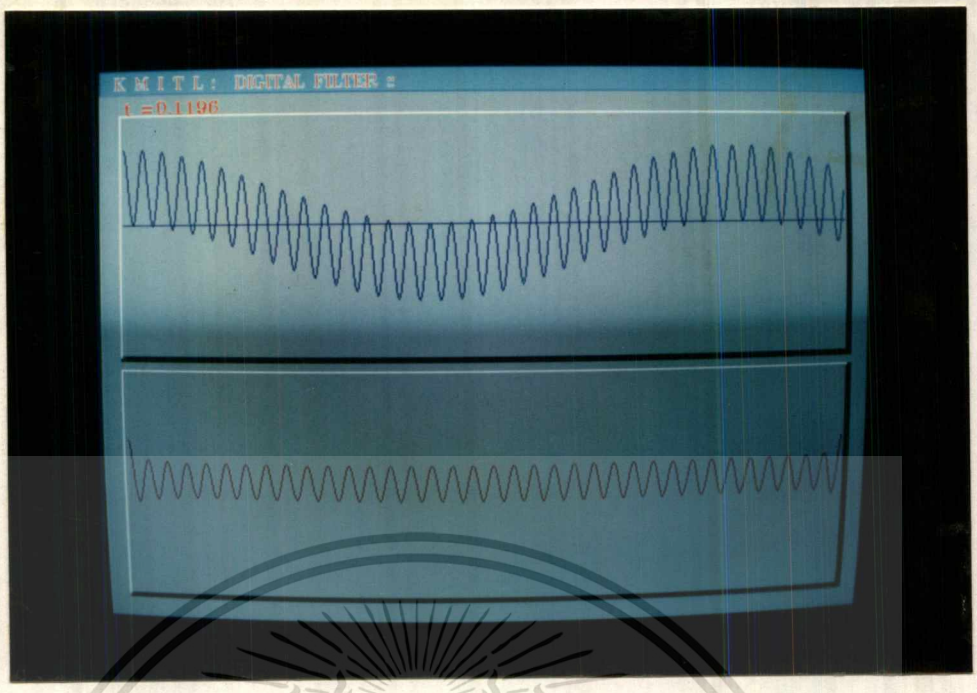
นั่นคือ เราสามารถลองทดสอบว่าฟิลเตอร์ที่เราออกแบบออกมานั้น สามารถตัดความถี่ที่เรา ไม่ต้องการออกได้หรือไม่ ตัวอย่างเช่น สร้างความถี่ขึ้นมา 2 ตัว คือ 10Hz และ 300 Hz และทำการออกแบบฟิลเตอร์ แบบ โลว์พาสส์ฟิลเตอร์ ให้คัทออฟ ที่ความถี่ 100 Hz โดยใช้สัมประสิทธิ์ที่ 99

รูปที่ 5.5 ภาพแสดงการออกแบบโลว์พาสส์ ฟิลเตอร์



รูปที่ 5.6 ภาพหน้าต่างบนเป็นรูปคลื่นที่เกิดจากความถี่ 10 และ 300 Hz รวมกัน หน้าต่างล่างเป็นรูปคลื่นความถี่ 10 Hz ที่ผ่านการฟิลเตอร์แล้ว

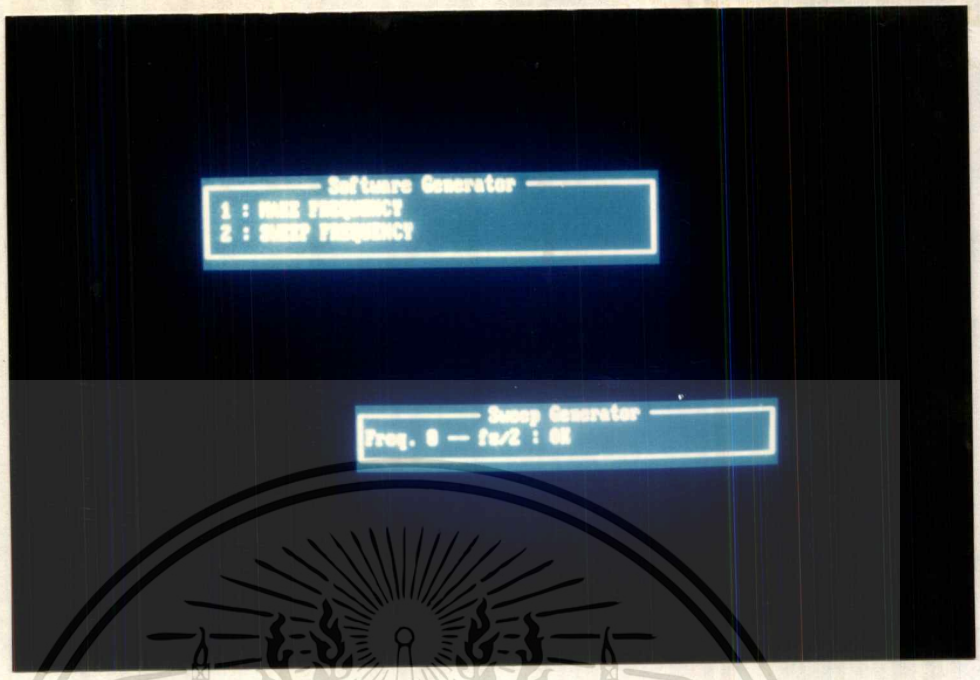
เมื่อทำการออกแบบเสร็จเรียบร้อยแล้ว ก็ไปยังเมนู Output เลือก Display รูปคลื่นที่ได้จะแสดงดังรูปที่ 5.6 ซึ่งจะเห็นว่าหน้าต่างแรกแสดงคลื่นที่เราสร้างขึ้น ส่วนเมนู 2 แสดงรูปคลื่นที่ถูกฟิลเตอร์แล้ว หรือในทางกลับกันถ้าต้องการส่วน HPF ก็ไปเลือกหัวข้อ High Pass Filter ในเมนู Filter ในที่นี้กำหนดให้ใช้ความถี่คัทออฟ ที่ 300 Hz และใช้สัมประสิทธิ์ที่ 99 เสร็จแล้วเลือก Display ใน เมนู Output ก็จะได้หน้าต่าง แสดงส่วนความถี่สูงออกมา ดังแสดงในรูปที่ 5.7



รูปที่ 5.7 ภาพหน้าต่างบนแสดงความถี่ 10 และ 300 Hz รวมกัน
หน้าต่างล่างแสดงความถี่ 300 Hz ที่ผ่านการฟิลเตอร์แล้ว

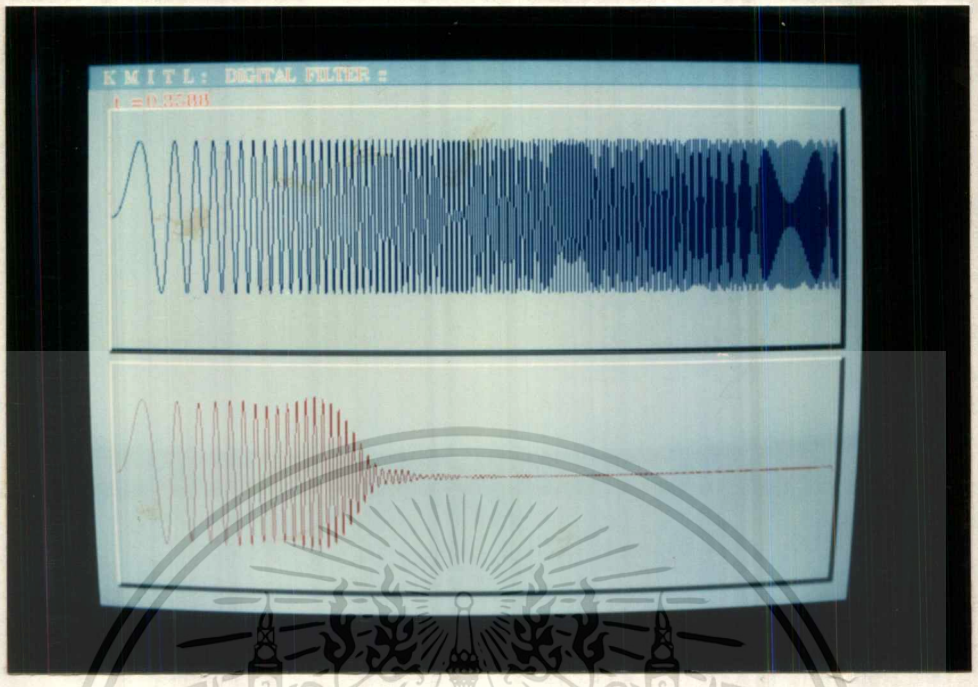
Sweep Frequency

ฟังก์ชันนี้จะสร้างความถี่ตั้งแต่ 0 ถึง fs/2 ออกมา เพื่อแสดงให้เห็นได้อย่างคร่าว ๆ ว่า ระบบมีการตอบสนองความถี่อย่างไร ซึ่งเมื่อทำการออกแบบฟิลเตอร์แบบต่าง ๆ ดังตัวอย่างที่กล่าวมาข้างต้นเรียบร้อยแล้ว ให้เลือกไปที่ตำแหน่ง Sweep Generator ด้วยทุกครั้ง โดยให้มีข้อความขึ้นที่หน้าจอดังรูปที่ 5.8

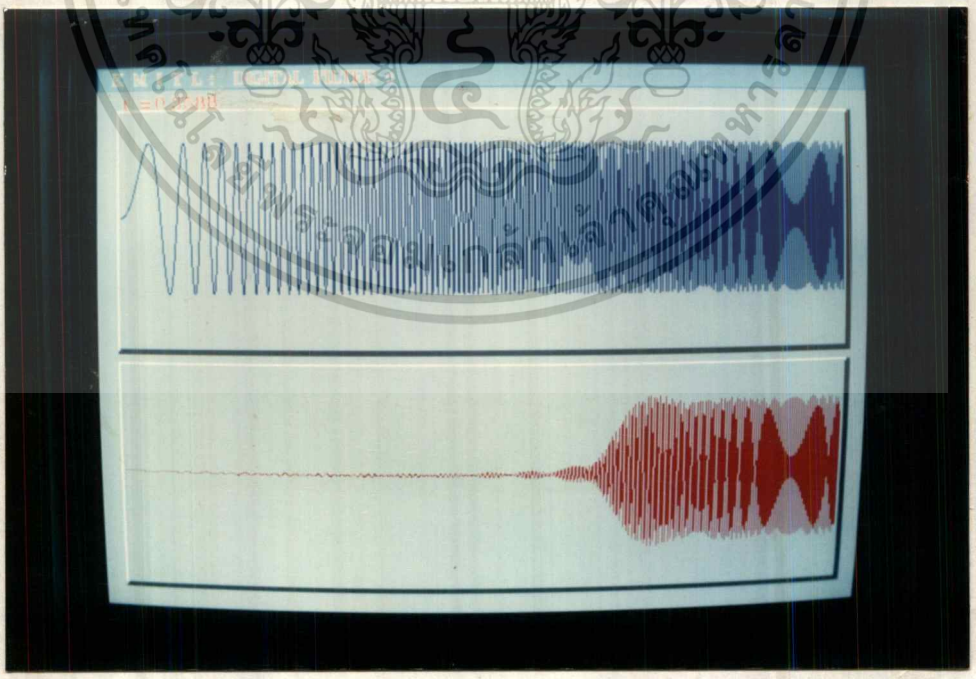


รูปที่ 5.8 ภาพแสดง Sweep Generator

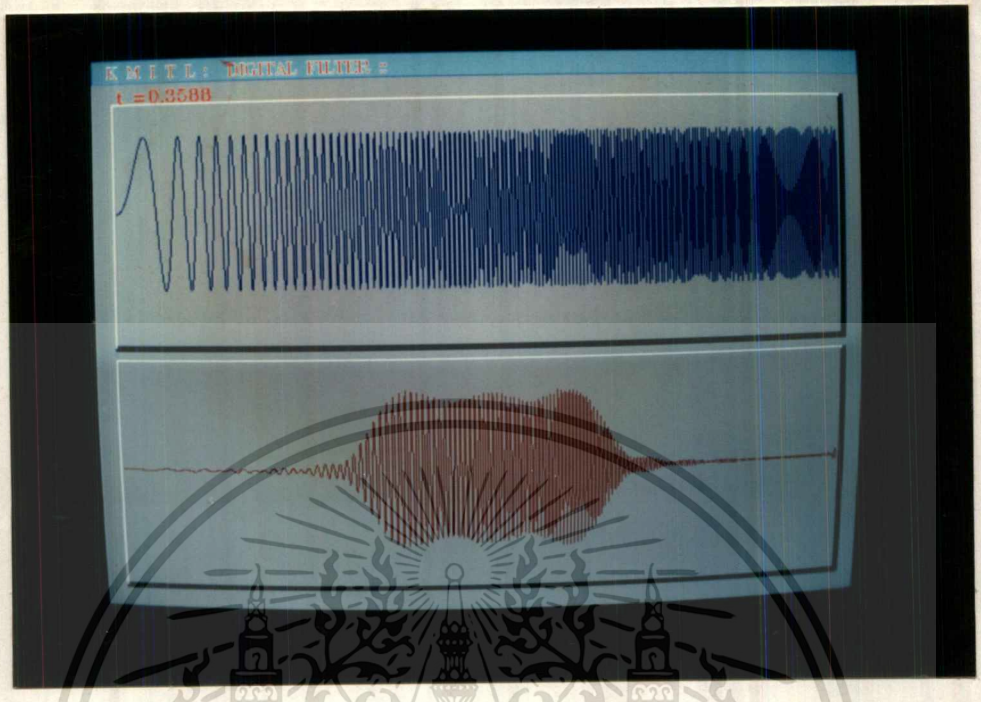
ตัวอย่างเช่นต้องการออกแบบฟิลเตอร์แบบต่าง ๆ ก็ให้เลือกความถี่แซมปลิ้ง แล้วป้อนค่าที่ต้องการลงไป เสร็จแล้วเลือกมาที่ตำแหน่ง Sweep Generator กด Enter ต่อมาเลือกไปที่เมนูการฟิลเตอร์แล้วเลือกการฟิลเตอร์แบบต่าง ๆ ที่ต้องการ ซึ่งจะมีให้เลือก 4 แบบ ได้แก่ LPF,HPF ,BPF,BSF ต่อมาเลือก ไปที่ User Design Filter แล้วทำการป้อนค่าความถี่และสัมประสิทธิ์ที่ต้องการออกแบบ ซึ่งจะได้รูปคลื่นการตอบสนองที่แสดงออกมาของฟิลเตอร์แบบต่าง ๆ ดังรูปที่ 5.9 (a),(b),(c),(d)



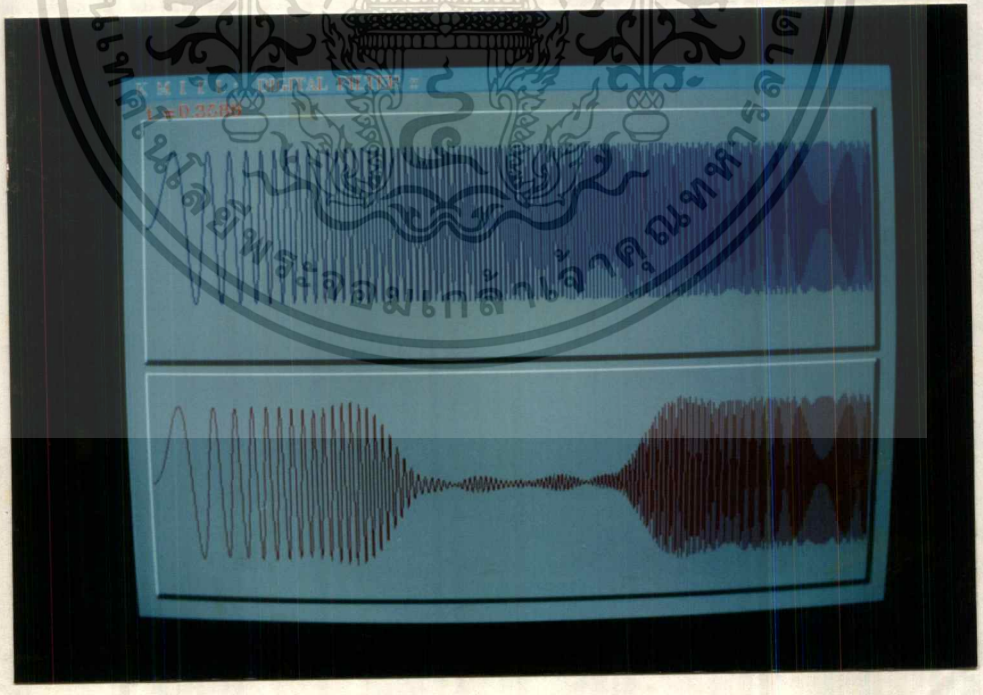
(a) ภาพการตอบสนองของโลว์พาสส์ ฟิลเตอร์



(b) ภาพการตอบสนองของไฮพาสส์ ฟิลเตอร์



(c) ภาพการตอบสนองของแบนด์ พาสส์ ฟิลเตอร์



(d) ภาพการตอบสนองของแบนด์ สตอป ฟิลเตอร์

รูปที่ 5.9 ภาพแสดงการตอบสนองของฟิลเตอร์แบบต่าง ๆ

FILTERING

ฟังก์ชันนี้ประกอบด้วย

- Low Pass Filter
- High Pass Filter
- Band Pass Filter
- Band Stop Filter
- User Design Filter

ซึ่งแสดงดังรูปที่ 5.10

รูปที่ 5.10 ภาพแสดงเมนูย่อยของฟิลเตอร์

Low Pass Filter : ในกรณีที่ต้องการได้เอาท์พุท แบบ LPF ก็เพียงแต่กด Enter ที่ฟังก์ชันนี้เท่านั้น

High Pass Filter : ในกรณีที่ต้องการได้เอาท์พุท แบบ HPF ก็เพียงแต่กด Enter ที่ฟังก์ชันนี้เท่านั้น

Band Pass Filter : ในกรณีที่ต้องการได้เอาท์พุท แบบ BPF ก็เพียงแต่กด Enter ที่ฟังก์ชันนี้เท่านั้น

Band Stop Filter : ในกรณีที่ต้องการได้เอาท์พุทแบบ BSF ก็เพียงแต่กด Enter ที่ฟังก์ชันนี้เท่านั้น

User Design Filter : ฟังก์ชันนี้จะสามารถให้เราออกแบบได้ไม่ว่าจะเป็น LPF,HPF,BPF, BSF แต่เราจะต้องทำการเลือกว่าจะออกแบบฟิลเตอร์ชนิดใดซะก่อนโดยการกดเลือกชนิดของฟิลเตอร์ที่ต้องการก่อนแล้ว จึงกดตัวเลือกนี้ ["User Design Filter"] ซึ่งจะมีค่า Default เป็นการออกแบบ เช่น ถ้าเลือกออกแบบ HPF

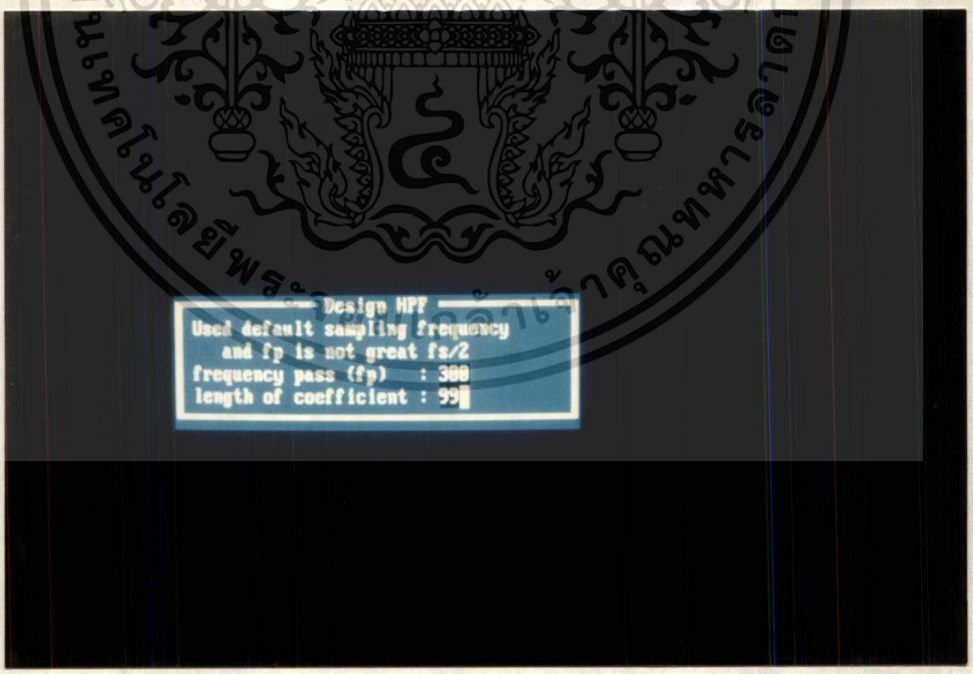
(1) ไปที่เมนู Record ก่อนเพื่อเลือกค่า Sampling Rate [ใส่ 5000]

(2) เลือก Frequency Software Generate และเลือก Sweep Frequency

(3) ไปเมนู Filtering เลือก "High Pass Filter"

(4) ไปเลือก "User Design Filter"

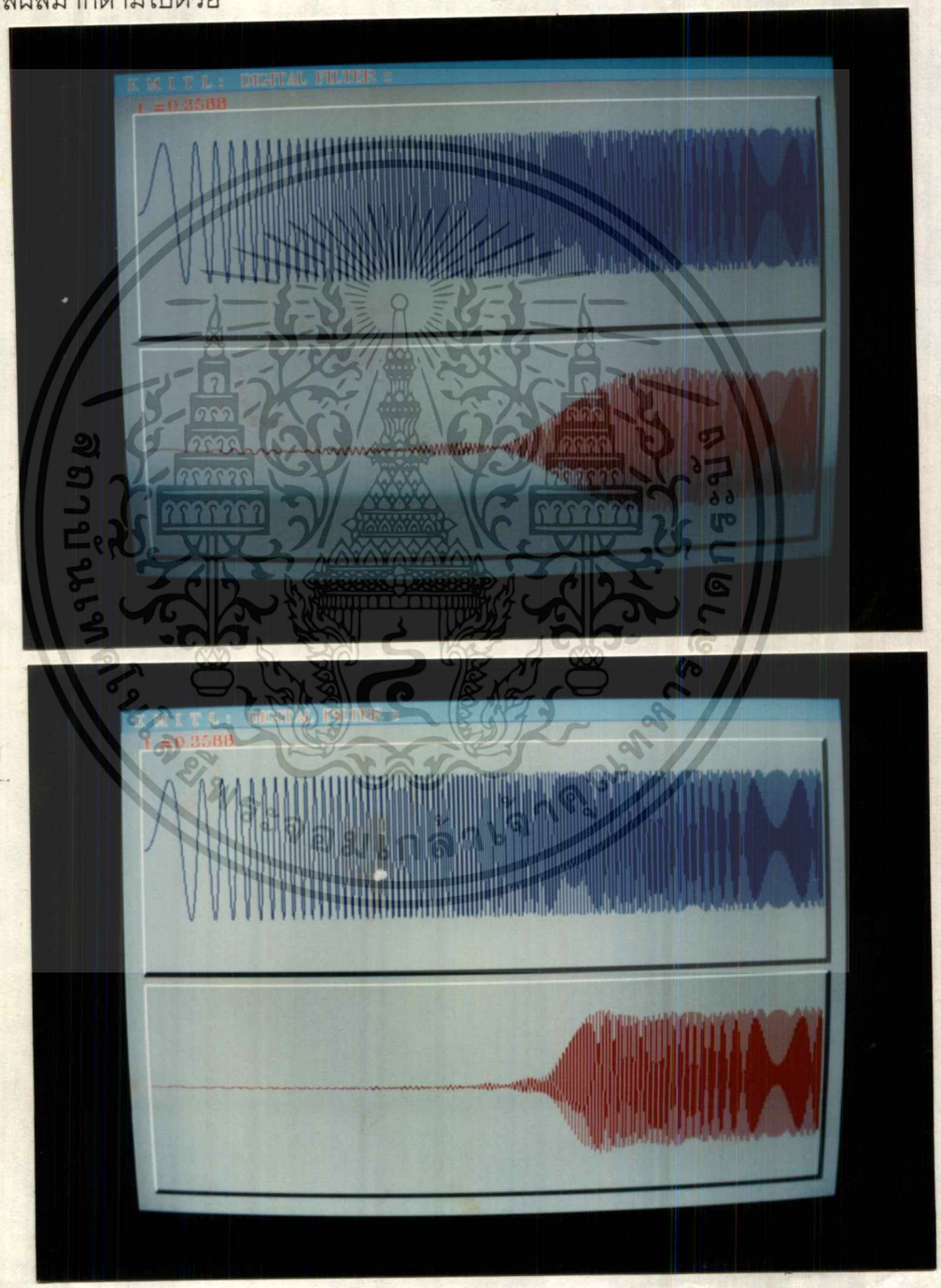
ซึ่งแสดงดังรูปที่ 5.11



รูปที่ 5.11 ภาพแสดงการเลือกออกแบบไฮพาสส์ ฟิลเตอร์

ฟังก์ชันจะให้เราใส่ค่าความถี่ (เป็น Hz) โดยความถี่ที่ใส่จะได้มากที่สุดไม่เกิน fs/2 [ป้อน 540 Hz]

ต่อมาฟังก์ชันจะให้เราป้อนจำนวนสัมประสิทธิ์ ซึ่งกำหนดสูงสุดไม่เกิน 99 ตัว [ป้อน 35] โดยถ้าป้อนค่ายิ่งมากก็จะมีผลให้ได้ค่าคัท ออฟ ที่คมมากยิ่งขึ้น แต่จะใช้เวลาในการประมวลผลมากตามไปด้วย



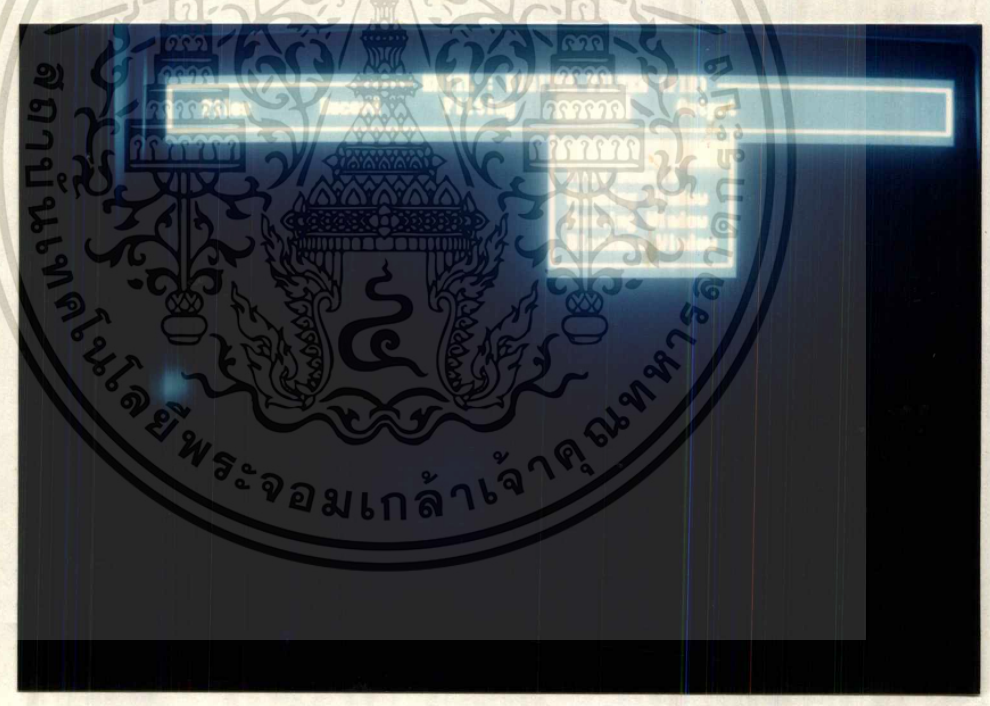
รูปที่ 5.12 ภาพแสดงการใช้สัมประสิทธิ์ต่างกัน

OPTION

ประกอบไปด้วยส่วนย่อย ๆ ดังนี้

- Time Domain
- Frequency Domain
- Hamming Window
- Hanning Window
- Blackman Window

ซึ่งแสดงดังรูปที่ 5.13



รูปที่ 5.13 ภาพแสดงเมนูย่อยของ Option

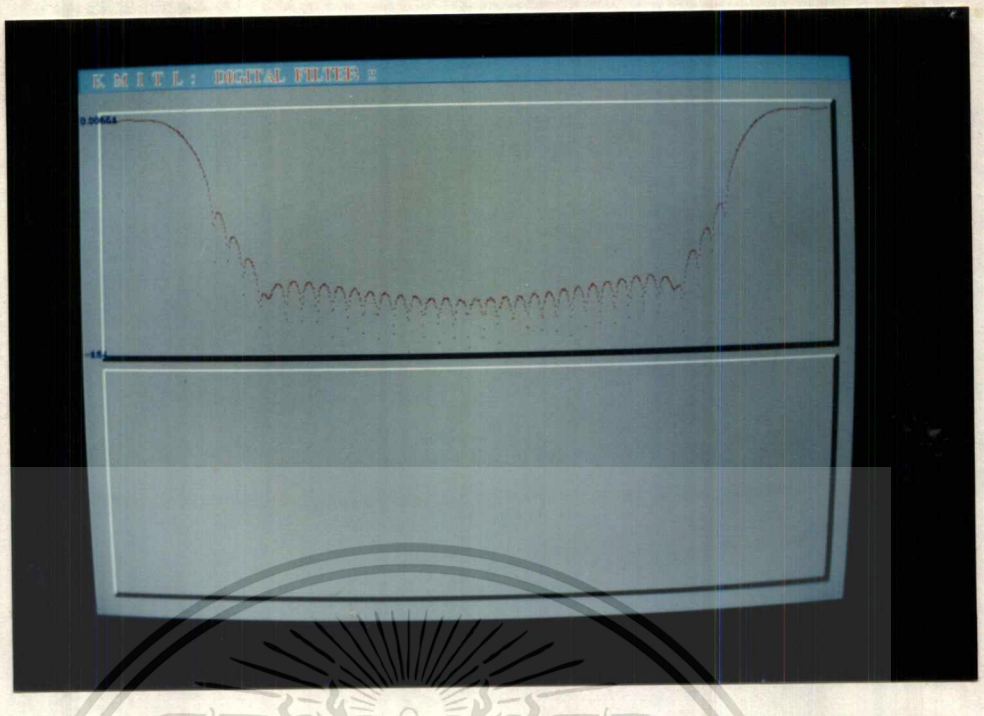
Time Domain จะใช้ในการเลือกว่าจะทำการประมวลผลสัญญาณในมิติของเวลา หรือความถี่

Frequency Domain จะใช้ในการเลือกว่าจะทำการประมวลผล [คอนโวลูชั่น] ในมิติของความถี่ แต่เนื่องจากโครงสร้างโปรแกรมและความซับซ้อนของ Algorithm (FFT) จึงทำให้ได้ผลลัพธ์ที่ไม่ถูกต้องทุกกรณีหรือใช้เวลาในการประมวลผลนานกว่าทางโดเมนเวลา ซึ่งในทางทฤษฎีแล้วการประมวลผล โดยใช้โดเมนความถี่ จะต้องเร็วกว่าโดเมนเวลามาก ฉะนั้นจึงยังต้องทำการแก้ไข โครงสร้างให้มีความรวดเร็วกว่านี้อีกมาก

Hamming Window เป็นการฟิลเตอร์แบบต่าง ๆ ที่ต้องการให้มีการฟิลเตอร์แบบผ่านหน้าต่าง Hamming ซึ่งสามารถทำได้โดยเมื่อเลือกการฟิลเตอร์ที่ต้องการแล้วก็เลือกมายังตำแหน่ง Hamming แล้วกด Enter ตัวอย่างดังแสดงในรูปที่ 5.14

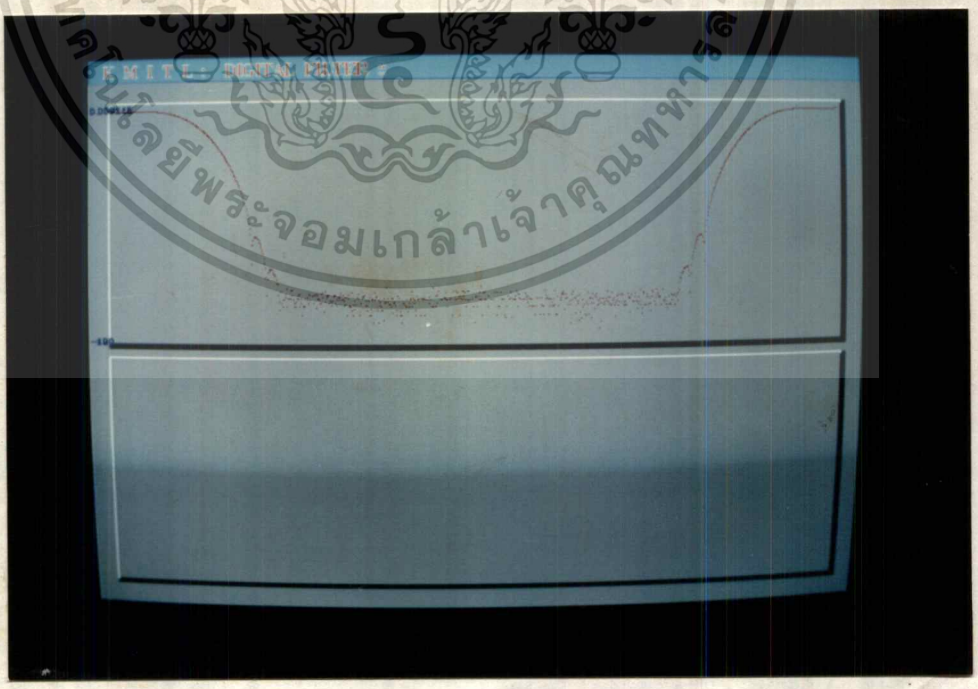
รูปที่ 5.14 ภาพแสดง Filter Response ของ Hamming Window

Hanning Window เป็นการฟิลเตอร์แบบต่าง ๆ ที่ต้องการให้มีการฟิลเตอร์แบบผ่านหน้าต่าง Hanning ซึ่งสามารถทำได้โดยเมื่อเลือกการฟิลเตอร์แบบที่ต้องการแล้วก็เลือกมายังตำแหน่ง Hanning แล้วกด Enter ตัวอย่างดังแสดงในรูปที่ 5.15



รูปที่ 5.15 ภาพแสดง Filter Response ของ Hanning Window

Blackman Window เป็นการฟิลเตอร์แบบต่าง ๆ ที่ต้องการให้มีการฟิลเตอร์แบบผ่านหน้าต่าง Blackman ซึ่งสามารถทำได้โดยเมื่อเลือกการฟิลเตอร์แบบที่ต้องการแล้วก็เลือกมายังตำแหน่ง Blackman แล้วกด Enter ตัวอย่างดังแสดงในรูปที่ 5.16



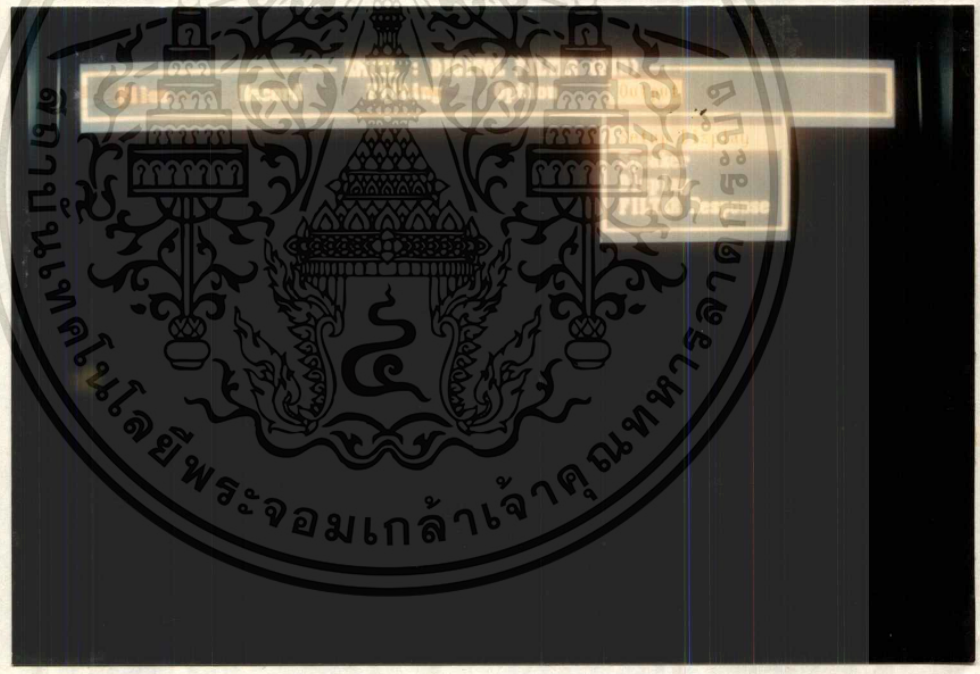
รูปที่ 5.16 ภาพแสดง Filter Response ของ Blackman Window

OUTPUT

ประกอบด้วยส่วนย่อยต่าง ๆ ดังนี้

- Setup Display
- Speaker
- Display
- FILTER Response

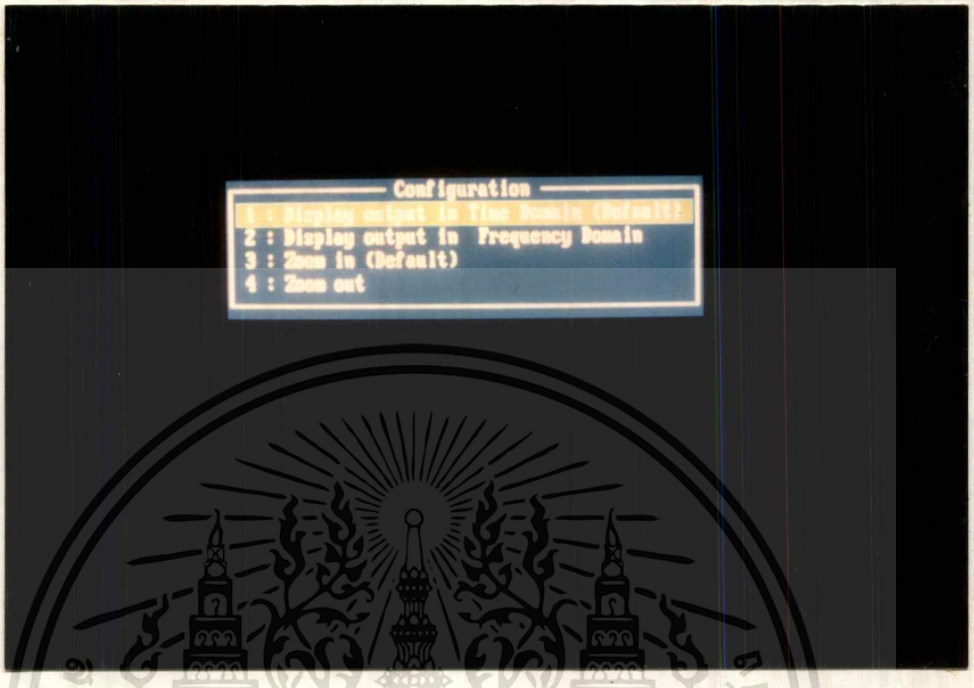
ดังแสดงในรูปที่ 5.17



รูปที่ 5.17 ภาพแสดงเมนูย่อยของ Output

- Setup Display

ประกอบด้วยส่วนย่อยต่าง ๆ ดังแสดงในรูปที่ 5.18



รูปที่ 5.18 ภาพแสดงส่วนต่าง ๆ ของ Setup Display

- Display Output in Time Domain (Default) จะเป็นการเซ็ทให้โปรแกรมแสดงผลอยู่ในรูปของลำดับ

- Display Output in Frequency Domain จะเป็นการเซ็ท ให้ฟังก์ชันแสดงสเปคตรัมของสัญญาณอินพุท กับ เอาท์พุท แต่สำหรับส่วนนี้ยังไม่เสร็จเรียบร้อยดีทำงานไม่ถูกต้องเท่าที่ควร ควรปรับปรุง

- Zoom In ในกรณีที่ต้องการเข้าไปดูรายละเอียดของสัญญาณ ว่ามีการเปลี่ยนแปลงอย่างไรในช่วงหนึ่ง ๆ ก็เพียงแต่กด Enter ตามจำนวนที่ต้องการ ฟังก์ชันนี้ยังทำงานได้ไม่ยืดหยุ่นดีนัก มันจะดูได้เฉพาะช่วงต้น ๆ ของสัญญาณเท่านั้น ซึ่งการพัฒนาขั้นต่อไปน่าที่จะดูได้ทุกช่วงของสัญญาณ

- Zoom Out ในกรณีที่ต้องการดูรายละเอียดอย่างคร่าว ๆ ก็ให้เลือกฟังก์ชันนี้ มันจะทำให้เราสามารถแสดงข้อมูลได้จำนวนมากใน 1 หน้าจอ

SPEAKER ฟังก์ชันจะนำไฟล์ที่เรากำหนดออก [*.VOC] Output ของการ์ดเสียง

DISPLAY จะแสดงผลข้อมูลของอินพุทไฟล์ หรือตัวกำเนิดรูปคลื่น (Wave Generator) กับข้อมูลที่ผ่านการประมวลผลแบบต่าง ๆ

FILTER RESPONSE จะใช้ในการดูการตอบสนองของฟิลเตอร์ ที่เราได้ทำการออกแบบเอาไว้ เช่น ดูการตอบสนอง Filter Response ของ Hamming Window (ยังทำงานได้ไม่เต็มที่เนื่องจากมีปัญหาเรื่องหน่วยความจำ)

โปรแกรม

DIGITAL FILTER (FIR)

```
/* ----------------------------------------------------------------
      DESIGN.C : For DSP (FILTER FIR) program !
      update time              10:21:37p
                  date           23/5/94
      ----------------------------------------------------------------
-- */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <math.h>
#include <graphics.h>
#include "twindow.h"
#include "keys.h"
#include "sbcvoice.h"
#include "project.h"
#include "filter.h"
#include "get.h"
#include "dft.h"
#define XAREA 598
#define     EVEN  0
#define  ODD      1
extern System sys;
extern VOC_FILE *voc;
extern float far *tbuffer;
extern float far *pbuffer;
extern FILTER *fil_ptr;
extern FILTER fir_lpf;
extern FILTER fir_hpf;
extern FILTER fir_bpf;
extern FILTER fir_bsf;
extern float fir_lpf35[35];


static float transform_coef[MAX_COEF];
FILTER fir_transform     = {35, NULL, transform_coef};
float dlpf[MAX_COEF];

#include "design.i"
void User_design (void)
{
      WINDOW *interface;
   interface = establish_window (10, 10, 6, 40);
   set_border (interface, 1);
   set_colors (interface, ALL, BLUE, GREEN, BRIGHT);
      switch (sys.type_fil)
   {
      case LPF:   set_title (interface, " Design LPF ");
                                 display_window (interface);
                                    design_lpf (interface);
                                    break;

      case HPF:   set_title (interface, " Design HPF ");
                                    display_window (interface);
                  design_hpf (interface);
                                    break;
      case BPF:   set_title (interface, " Design BPF ");
                  display_window (interface);
                  design_bpf (interface);
                                    break;
      case BSF:   set_title (interface, " Design BSP ");
                                    display_window (interface);
                  design_bsf (interface);
                                    break;
   }
```

```
    }
    delete_window (interface);
}

void design_lpf (WINDOW *inter)
{
      unsigned fp, N;
    unsigned i;
    wprintf (inter, " Used default sampling frequency \n");
    wprintf (inter, "    and fp is not great fs/2 \n");
    wprintf (inter, " frequency pass (fp)    : \n");
    wprintf (inter, " length of coefficient : ");
    set_cursor_type (5);
    cursor (36, 13); scanf ("%d",&fp);
    cursor (36, 14); scanf ("%d",&N);
    set_cursor_type (-1);
    i = odd_even (N);
       if (i == EVEN) N += 1;
       clpf (fp, N, transform_coef);
    fir_transform.length = N;
       fil_ptr = &fir_transform;
    sys.type_fil = DLPF;
    save_design (inter, "DLPF.FIR", transform_coef, N);
}

void design_hpf (WINDOW *inter)
{
      unsigned fp, N;
    unsigned i;
    double a;

    wprintf (inter, " Used default sampling frequency \n");
    wprintf (inter, "    and fp is not great fs/2 \n");
    wprintf (inter, " frequency pass (fp)    : \n");
    wprintf (inter, " length of coefficient : ");
    set_cursor_type (5);
    cursor (36, 13); scanf ("%d",&fp);
       cursor (36, 14); scanf ("%d",&N);
    set_cursor_type (-1);
    i = odd_even (N);
    if (i == EVEN) N += 1;

    fp = sys.sampling/2 - fp;

       clpf (fp, N, transform_coef);
//    a = (N-1)/2;
       a = 1;
    for (i=0;i<N;i++,a++)
       transform_coef[i] = pow (-1,a)*transform_coef[i];

    fir_transform.length = N;
       fil_ptr = &fir_transform;
    sys.type_fil = DHPF;
       save_design (inter, "DHPF.FIR" , transform_coef, N);
}


void design_bpf (WINDOW *inter)
{
      unsigned fu, fl, N, fp;

      double twopi, fo;
    unsigned  i;
    int a;
```

```
   wprintf (inter, " Used default sampling frequency \n");
   wprintf (inter, " frequency low  (fl)   : \n");
   wprintf (inter, " frequency high (fu)   : \n");
   wprintf (inter, " length of coefficient : ");
   set_cursor_type (5);
   cursor (36, 12); scanf ("%d",&fl);
   cursor (36, 13); scanf ("%d",&fu);
   cursor (36, 14); scanf ("%d",&N);
   set_cursor_type (-1);
   i = odd_even (N);
   if (i == EVEN) N += 1;

      fp = (fu - fl) / 2;
   fo = (fu + fl) / 2;

      clpf (fp, N, transform_coef);

   twopi = 8*atan(1);
   twopi = twopi * (fo/sys.sampling);
   a = (N-1)/2;
   a = -a;
   for (i=0; i<(N+1)/2; i++, a++)   {
      transform_coef[i] = (2*cos(a*twopi))*transform_coef[i];
   }
   a = 1;
   for (; i < N; i++, a++)    {
         transform_coef[i] = (2*cos(a*twopi))*transform_coef[i];
      }

   fir_transform.length = N;
      fil_ptr = &fir_transform;
      sys.type_fil = DBPF;
      save_design (inter, "DBPF.FIR" , transform_coef, N);
}

void design_bsf (WINDOW *inter)
{
      unsigned fu, fl, N, fp;

      double twopi, fo;
   unsigned  i;
   int a;

   wprintf (inter, " Used default sampling frequency \n");
   wprintf (inter, " frequency low  (fl)   : \n");
   wprintf (inter, " frequency high (fu)   : \n");
   wprintf (inter, " length of coefficient : ");
   set_cursor_type (5);
   cursor (36, 12); scanf ("%d",&fl);
   cursor (36, 13); scanf ("%d",&fu);
   cursor (36, 14); scanf ("%d",&N);
      set_cursor_type (-1);
   i = odd_even (N);
   if (i == EVEN) N += 1;

      fp = (fu - fl) / 2;
   fo = (fu + fl) / 2;

      clpf (fp, N, transform_coef);

   twopi = 8*atan(1);
   twopi = twopi * (fo/sys.sampling);
   a = (N-1)/2;
   a = -a;
```

```
   for (i=0; i<(N-1)/2; i++, a++)    {
      transform_coef[i] = (2*cos(a*twopi))*transform_coef[i];
      transform_coef[i] = -transform_coef[i];
   }
      transform_coef[i] = 1 - transform_coef[i];
      i++;
      a = 1;
      for (; i < N; i++, a++) {
            transform_coef[i] = (2*cos(a*twopi))*transform_coef[i];
            transform_coef[i] = -transform_coef[i];
      }


      fir_transform.length = N;
      fil_ptr = &fir_transform;
      sys.type_fil = DBSF;
      save_design (inter, "DBSF.FIR" , transform_coef, N);
}


int clpf (unsigned fp, unsigned N, float *coef)
{
      float wdp, wsampling, twopi, ratio, pi;
      unsigned  i, a, b, oe;
      twopi = 8*atan(1);
      pi = twopi / 2;
      ratio = (float) fp/sys.sampling;
      wdp = twopi*ratio;
      oe = odd_even (N);
      if (oe == ODD)  a = b = (N-1) / 2;
      else  a = b = N/2;
      for (i = 0; i < b; i++,a--)    {
            coef[i] = 1/(pi*a)*sin(a*wdp);
      }
      if (oe == ODD) {
            coef[i] = (float) wdp / pi;
            i++;
      }
      a = 1;
      for (;i < N; i++,a++)    {
            coef[i] = 1/(pi*a)*sin(a*wdp);
      }
      return 1;
}

int odd_even (unsigned n)
{
      unsigned a;
      a = n & 0x0001;
      if (a == ODD) return ODD;
      else return EVEN;
}

int save_design (WINDOW *inter, char *name, float *data, unsigned N)
{
      FILE *fp;
      unsigned i;
      static char str[25];
      fp = fopen (name, "w");
      if (fp == NULL)
            {
                  print_error ("can't save data (FILTER COEFF)");
                  return -1;
            }
      for (i=0; i < N; i++)    {
```

```
            strset (str, 0);
            gcvt (data[i], 10, str);
            strcat (str,"\n");
            fwrite (str, 1, strlen(str), fp);
      }
   fclose (fp);
   return 1;
}
#define YAREA 200
vdid Vlpf (void)
{
   COMPLEX *filt;
   FILTER *fil;
   float a, *signal,zoom;
   double tempflt;
   char str[25],str2[15];
   unsigned m, fft_length, x;
   int flag=1, i;
   float min, max, dbmin;
   switch (sys.type_fil)
      {
      case LPF:  fil = &fir_lpf;    break;
      case HPF:  fil = &fir_hpf; break;
      case BPF:  fil = &fir_bpf; break;
      case BSF:  fil = &fir_bsf; break;
      default : fil = &fir_transform; break;
      }
      m = log2(sys.sampling);
   fft_length = 1<<m;
   filt = (COMPLEX *) calloc (fft_length, sizeof (COMPLEX));
      if (filt == NULL)
            {
                  print_error ("Out of memory for function
VLPF");
                  flag = 0;
            }
      signal = calloc (fft_length, sizeof (float));
            if (signal == NULL)
                  {
                  print_error ("Out of memory for function
VLPF");
                  flag = 0;
         }

      if (flag != 0)
      {
      for (i = 0; i < fil->length; i++)
            filt[i].real = fil->coef[i];

            fft (filt, m);
      a = fil->length * fil->length;
            for (i = 0; i < fft_length; i++)
            {
            tempflt = filt[i].real * filt[i].real;
            tempflt += filt[i].imag * filt[i].imag;
            signal[i] = 10*log10(MAX(tempflt, 1e-19));
          }
      Graphics_interface ();
      setusercharsize (1, 4, 1, 4);
    setcolor (EGA_BLUE);
    i = 0;
      max_min (&max, &min, signal, fft_length);
 /*   for (x = 0; x < 200; x=x+10, i = i - 10)    {
      itoa (i, str, 10);
            outtextxy (2, 40+x, str);
```

```
   } */
      gcvt (max, 3, str); gcvt(min, 3, str2);
      if (min < 0) {dbmin = min; min = -min;};
      zoom = (float) ((float) min/(float)YAREA);
            for (x=0; x < fft_length; x++)
      {
      a = (float) x / ((float) fft_length/XAREA);
            putpixel (a+20, 45-(signal[x]/zoom), EGA_RED);
         if (signal[x] == dbmin) outtextxy (2, 35-(signal[x]/zoom),
str2);
         if (signal[x] == max) outtextxy (2, 40-(signal[x]/zoom),
str);
      }
   get_char();
   free (filt);
   free (signal);
   closegraph ();
   }

}

void max_min (float *max, float *min, float *buf, unsigned length)
{
   register i;
   *max = *min = buf[0];
      for (i = 1; i < length; i++)
   {
      if (*max < buf[i]) *max = buf[i];
      if (*min > buf[i]) *min = buf[i];
   }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
#include "twindow.h"
#include "dft.h"
void print_error (char *);
/*****************************************************************
**

DFT.C - SOURCE CODE FOR DISCRETE FOURIER TRANSFORM FUNCTIONS

fft     In-place radix 2 decimation in time FFT
log2    Base 2 logarithm

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

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

fft - In-place radix 2 decimation in time FFT

Requires pointer to complex array, x and power of 2 size of FFT, m
(size of FFT = 2**m).  Places FFT output on top of input COMPLEX
array.

int fft(COMPLEX *x, int m)

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

int fft(x,m)
    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;

    double 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 *) calloc(le-1,sizeof(COMPLEX));
```

```
            if(!w) {
                    closegraph ();
                    print_error ("Error : Out of memory for FFT")
;
                    return (-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 = (float)wrecur_real;
            xj->imag = (float)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;
        }
      }
    return 1;
}

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

log2 - base 2 logarithm

Returns base 2 log such that i = 2**ans where ans = log2(i).
if log2(i) is between two values, the larger is returned.

int log2(unsigned int x)

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

int log2(x)
    unsigned int x;
{
    unsigned int mask,i;

    if(x == 0) return(-1);          /* zero is an error, return -1 */

    x--;                            /* get the max index, x-1 */

       for(mask = 1 , i = 0 ; ; mask *= 2 , i++) {
        if(x == 0) return(i);       /* return log2 if all zero */
        x = x & (~mask);            /* AND off a bit */
    }
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "filter.h"
void print_error (char *);
/*************************************************************************
******

FILTER.C - Source code for filter functions

      fir_filter_array   FIR filter float array of data
      fir_filter         FIR filter floats sample by sample (real
time)

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

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

fir_filter_array - Filter float array of data

Requires FILTER structure for coefficients.
Input and output arrays are of equal length
and are allocated by the calller.

void fir_filter_array(float *in,float *out,int in_len,FILTER *fir)

    in          float pointer to input array
    out         float pointer to output array
    in_len      length of input and output arrays
    FILTER *fir   pointer to FILTER structure

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

void fir_filter_array(in,out,in_len,fir)
    float far *in,*out;
    unsigned long in_len;
    FILTER *fir;
{
    int i,j,coef_len2,acc_length;
    float acc;
    float far *in_ptr,*data_ptr,*coef_start,*coef_ptr,*in_end;

/* set up for coefficients */
    coef_start = fir->coef;
    coef_len2 = (fir->length + 1)/2;

/* set up input data pointers */
    in_end = in + in_len - 1;
    in_ptr = in + coef_len2 - 1;

/* initial value of accumulation length for startup */
    acc_length = coef_len2;

    for(i = 0 ; i < in_len ; i++) {

/* set up pointers for accumulation */

        data_ptr = in_ptr;
        coef_ptr = coef_start;

/* do accumulation and write result */
```

```
        acc = (*coef_ptr++) * (*data_ptr--);
        for(j = 1 ; j < acc_length ; j++)
            acc += (*coef_ptr++) * (*data_ptr--);
        *out++ = acc;

/* check for end case */

        if(in_ptr == in_end) {
            acc_length--;       /* one shorter each time */
            coef_start++;       /* next coefficient each time */
        }

/* if not at end, then check for startup, add to input pointer */

        else {
            if(acc_length < fir->length) acc_length++;
            in_ptr++;
        }
    }
}

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

fir_filter - Perform fir filtering sample by sample on floats

Requires FILTER structure for history and coefficients.
Returns one output sample for each input sample.  Allocates history
array if not previously allocated.

float fir_filter(float input,FILTER *fir)

    float input         new float input sample
    FILTER *fir         pointer to FILTER structure

Returns float value giving the current output.
Allocation errors cause an error message and a call to exit.

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

float fir_filter(input,fir)
    float input;            /* new input sample */
    FILTER *fir;            /* pointer to FILTER structure */
{
    int i;
    float *hist_ptr,*hist1_ptr,*coef_ptr;
    float output;

/* allocate history array if not already allocated */
    if(!fir->history) {
        fir->history = (float *) calloc(fir->length-1,sizeof(float));
        if(!fir->history) {
                        print_error ("Unable to allocate history
array in fir_filter");
            exit(1);
        }
    }

    hist_ptr = fir->history;
    hist1_ptr = hist_ptr;                       /* use for history update
*/
    coef_ptr = fir->coef + fir->length - 1; /* point to last coef */
```

```
/* form output accumulation */
    output = *hist_ptr++ * (*coef_ptr--);
    for(i = 2 ; i < fir->length ; i++) {
        *hist1_ptr++ = *hist_ptr;              /* update history array
*/
        output += (*hist_ptr++) * (*coef_ptr--);
    }
    output += input * (*coef_ptr);             /* input tap */
    *hist1_ptr = input;                        /* last history */

    return(output);
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

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

GET.C - Source code for user input functions

    get_string      get string from user with prompt
    get_int         get integer from user with prompt and range
    get_float       get float from user with prompt and range

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

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

get_string - get string from user with prompt

Return pointer to string of input text, prompts user with string
passed by caller.  Indicates error if string space could not be
allocated.  Limited to 80 char input.

char *get_string(char *prompt_string)

    prompt_string  string to prompt user for input

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

char *get_string(char *title_string)
{
    char *alpha;                                /* result string pointer
*/

    alpha = (char *) malloc(80);
    if(!alpha) {
            print_error ("String allocation error in get_string");
            return (NULL);
       }
       printf("%s ",title_string);
       gets(alpha);

       return(alpha);
}
/*************************************************************************
******

get_int - get integer from user with prompt and range

Return integer of input text, prompts user with prompt string
and range of values (upper and lower limits) passed by caller.

int get_int(char *title_string,int low_limit,int up_limit)

    title_string  string to prompt user for input
    low_limit     lower limit of acceptable input (int)
    up_limit      upper limit of acceptable input (int)

*************************************************************************
****/
```

```
int get_int(title_string,low_limit,up_limit)
    char *title_string;
    int low_limit,up_limit;
{
    int i,error_flag;
    char *get_string();              /* get string routine */
    char *cp,*endcp;                 /* char pointer */
    char *stemp;                     /* temp string */

/* check for limit error, low may equal high but not greater */
    if(low_limit > up_limit) {
            print_error ("Limit error, lower > upper");
            return (-1);
    }

/* make prompt string */
    stemp = (char *) malloc(strlen(title_string) + 60);
    if(!stemp) {
            print_error ("String allocation error in get_int");
            return (-1);
    }
    sprintf(stemp,"%s [%d...%d]",title_string,low_limit,up_limit);

/* get the string and make sure i is in range and valid */
    do {
        cp = get_string(stemp);
        i = (int) strtol(cp,&endcp,10);
        error_flag = (cp == endcp) || (*endcp != '\0'); /* detect
errors */
        free(cp);                                   /* free string
space */
    } while(i < low_limit || i > up_limit || error_flag);

/* free temp string and return result */
    free(stemp);
    return(i);
}
/*****************************************************************
******

get_float - get float from user with prompt and range

Return double of input text, prompts user with prompt string
and range of values (upper and lower limits) passed by caller.

double get_float(char *title_string,double low_limit,double up_limit)

    title_string  string to prompt user for input
    low_limit     lower limit of acceptable input (double)
    up_limit      upper limit of acceptable input (double)

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

double get_float(title_string,low_limit,up_limit)
    char *title_string;
    double low_limit,up_limit;
{
    double x;
    int error_flag;
    char *get_string();              /* get string routine */
    char *cp,*endcp;                 /* char pointer */
    char *stemp;                     /* temp string */

/* check for limit error, low may equal high but not greater */
```

```
    if(low_limit > up_limit) {
              print_error ("Limit error, lower > upper");
              return (-1);
    }

/* make prompt string */
    stemp = (char *) malloc(strlen(title_string) + 80);
    if(!stemp) {
              print_error ("String allocation error in get_float");
              return (-1);
    }

    sprintf(stemp,"%s
[%1.2g...%1.2g]",title_string,low_limit,up_limit);

/* get the string and make sure x is in range */
    do {
        cp = get_string(stemp);
        x = strtod(cp,&endcp);
        error_flag = (cp == endcp) || (*endcp != '\0'); /* detect
errors */
        free(cp);                                       /* free string
space */
    } while(x < low_limit || x > up_limit || error_flag);

/* free temp string and return result */
    free(stemp);
    return(x);
}
```

```
#ifdef __TINY__
#error GRAPHICS will not run in the tiny model.
#endif
#define XAREA  606
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include <fcntl.h>
#include <io.h>
#include <graphics.h>

void Initialize (void);
void StatusLine (char *msg);
void Pause ();
void Screen_GUI ();

int   GraphDriver;                                  /* The Graphics
device driver */
int   GraphMode;                                    /* The Graphics
mode value */
int   MaxX, MaxY;                                   /* The maximum
resolution of the screen */
int   MaxColors;                                    /* The maximum
# of colors available */
int   ErrorCode = 0;                                /* Reports any
graphics errors */
struct palettetype palette;                 /* Used to read palette
info */
extern char msg[] = " K M I T L :  DIGITAL FILTER :: ";

void Initialize (void)
{
      GraphDriver = DETECT;
      initgraph (&GraphDriver, &GraphMode, "");
      ErrorCode = graphresult ();
      if (ErrorCode != grOk)
            {
                  printf (" Graphics System Error:%s\n",
                                          grapherrormsg(ErrorCode));
                  exit (1);
            }
      getpalette(&palette);                                   /* Read
the palette from board */
      MaxColors = getmaxcolor () + 1;                   /* Read
maximum colors */
      MaxX = getmaxx ();
      MaxY = getmaxy ();
}

/*    ERASESTR: Used to erase a portion of the screen before a new
string */
/*    is written to the screen or simply to remove a string from the
screen */

void erasestr (int xloc, int yloc, char *str)
{
      struct textsettingstype textinfo;
      int xdim, ydim;
      void *textimage;
      gettextsettings (&textinfo);
      switch (textinfo.direction)
      {
```

```
            case HORIZ_DIR: xdim = textwidth (str);
                                    ydim = textheight (str);
                                    xloc--; break;
            case VERT_DIR:  ydim = textwidth (str);
                                    xdim = textheight (str);
                                    yloc++; break;
      }

      switch (textinfo.horiz)
      {
            case  LEFT_TEXT:                        break;
            case  CENTER_TEXT:      xloc -= xdim / 2;       break;
            case  RIGHT_TEXT:       xloc -= xdim;
break;
      }

      switch (textinfo.vert)
      {
            case  BOTTOM_TEXT:      yloc -= ydim;
break;
            case  CENTER_TEXT:      yloc -= ydim / 2;       break;
            case  TOP_TEXT:
            break;
      }

      while (xloc < 0) {xloc++; xdim--;}                    /* if
position is offscreen */
      while (yloc < 0) {yloc++; ydim--;}                    /* move
back to valid area */
      textimage = malloc (imagesize(xloc, yloc, xdim, ydim));
      getimage (xloc, yloc, xloc+xdim, yloc+ydim, textimage);
      putimage (xloc, yloc, textimage, XOR_PUT);
      free (textimage);
}

/*          GPRINTF: Used like PRINTF except the output is sent to
the screen   */
/*    in graphics mode at the specified co-ordinate. Depending on
text    */
/*    direction, the xloc or yloc coordinate is returned offset
according */
/*                                                          to the
string (font) height                                */

void gprintf (int *xloc, int *yloc, char *fmt, ...)
{
      va_list argptr;
      /* argument list pointer */
      char str[140];
            /* buffer to build string into */
      struct textsettingstype textinfo;             /* text
settings information */
      int pos_adj = *xloc;
      /* default position for adjust */
      va_start(argptr, fmt);                                /*
initialize va_functions */
      vsprintf (str, fmt, argptr);                          /* add
elements to str buffer */
      gettextsettings (&textinfo);                          /* check
graphic text settings */
      outtextxy (pos_adj, *yloc, str);                      /* write
string in graphics mode */
      switch (textinfo.direction)                           /* adjust
offset for next string */
      {
```

```
            case HORIZ_DIR: *yloc += textheight (str) + 2;  break;
            case VERT_DIR:  *xloc += textheight (str) + 2;  break;
      };
      va_end(argptr);
}

/*          GPRINTC: Used like GRPINTF except the area where the text
will be    */
/*          written to the screen is first reset to the background
color.          */

void gprintc (int *xloc, int *yloc, char *fmt,...)
{
      va_list argptr;
      char str[140];
      struct textsettingstype textinfo;
      int pos_adj = *xloc;                                          /*
default position for adjust */

      va_start(argptr, fmt);                                  /* initialize
va_funtions */
      vsprintf (str, fmt, argptr);                            /* add elements
to str buffer */
      gettextsettings (&textinfo);                            /* check
graphic text settings */
      erasestr (*xloc, *yloc, str);                           /* erase the
area first */
      outtextxy (pos_adj, *yloc, str);
      switch (textinfo.direction)
      {
            case HORIZ_DIR : *yloc += textheight (str) + 2; break;
            case VERT_DIR  : *xloc += textheight (str) + 2; break;
      };
      va_end(argptr);
}

/*          GPRINTXY : Used like GPRINTC except the area where screen
coordinates */
/*                     are passed by value rather than passed by address
pointer */
void gprintxy (int xloc, int yloc, char *fmt,...)
{
      va_list argptr;
      char str[140];
      struct textsettingstype textinfo;
      va_start(argptr, fmt);
      vsprintf (str, fmt, argptr);
      gettextsettings (&textinfo);
      erasestr (xloc, yloc, str);
      outtextxy (xloc, yloc, str);
      va_end (argptr);
}

void Pause ()
{
      static char msg[] = "Press any key...";
      StatusLine (msg);
      while (kbhit ()) getch ();
      getch ();
}

void StatusLine (char *msg)
{
      int height;
      setcolor (MaxColors - 1);
```

```
        settextstyle (DEFAULT_FONT, HORIZ_DIR, 1);
        settextjustify (CENTER_TEXT, TOP_TEXT);
        height = textheight ("H");
        rectangle (0, MaxY-(height+4), MaxX, MaxY);
        outtextxy (MaxX/2, MaxY-(height+2), msg);
}

void Graphics_interface ()
{
        struct palettetype palette;
        struct textsettingstype textinfo;
        unsigned int maxx;
        int color;
        unsigned int x1, x2, y1, y2;
        Initialize ();
        settextstyle (TRIPLEX_FONT, HORIZ_DIR, 0);
        setusercharsize (1, 2, 1, 2);

        getpalette(&palette);
        maxx = getmaxx ();
        setbkcolor (EGA_LIGHTGRAY);
        setfillstyle (SOLID_FILL, EGA_LIGHTBLUE);
        bar (0, 0, maxx, 20);
        setfillstyle (SOLID_FILL, EGA_LIGHTCYAN);
        bar (1, 1, maxx-1, 19);
        setcolor (EGA_LIGHTRED);
        outtextxy (5, 1, msg);

        x1 = 17; x2 = 624; y1 = 38; y2 = 245;
        setfillstyle (SOLID_FILL, EGA_WHITE);
        bar (x1, y1, x2, y2);
        setfillstyle (SOLID_FILL, EGA_DARKGRAY);
        bar (x1+1, y1+2, x2, y2);
        bar (x1, y2-2, x1+3, y2);              /* little */
        bar (x2-3, y1, x2, y1+7);      /* little */
        setfillstyle (SOLID_FILL, EGA_LIGHTGRAY);
        bar (x1+1, y1+1, x2-5, y2-5);

        x1 = 17; x2 = 624; y1 = 253; y2 = 460;
        setfillstyle (SOLID_FILL, EGA_WHITE);
        bar (x1, y1, x2, y2);
        setfillstyle (SOLID_FILL, EGA_DARKGRAY);
        bar (x1+1, y1+2, x2, y2);
        bar (x1, y2-2, x1+3, y2);              /* little */
        bar (x2-3, y1, x2, y1+7);      /* little */
        setfillstyle (SOLID_FILL, EGA_LIGHTGRAY);
        bar (x1+1, y1+1, x2-5, y2-5);

}
```

```
/* ------------------------------------------------------------------
------ */
/*  @@ Usage                                          *** TC Version
***     */
/*
*/
/*  Copyright (c) Creative Technology Pte Ltd, 1991. All rights
reserved.   */
/*
*/
/*   char far *LoadDriver (char *szDrvName)
*/
/*
*/
/*   DESCRIPTION:
*/
/*       Loads driver into memory with the driver name specified. The
*/
/*       driver is always loaded to the offset 0 of a segement.
*/
/*
*/
/*   ENTRY:
*/
/*       szDrvName :- Driver name to be loaded.
*/
/*
*/
/*   EXIT:
*/
/*       Pointer to the loaded driver if successful, else returns
NULL      */
/*
*/
/* ------------------------------------------------------------------
------ */

#include  <dir.h>
#include  <stdlib.h>
#include  <io.h>
#include  <fcntl.h>
#include  <dos.h>
#include  <string.h>

char far *LoadDriver (char *szDrvName)
{
    char        far *lpDrvPtr = 0 ;
    char        far *lpPtr ;
    char        szDrvFile[100] ;
    char        *pPtr ;
    int         Handle = 1, NotDone = 1;
    unsigned    wDrvSize, wTemp, wDrvSeg ;
    struct ffblk     stFile ;


    /* set the default file mode to binary mode */
    _fmode = O_BINARY;

    /* locate driver through environment parameter */
    if ((pPtr = getenv("SOUND")) != 0)
    {
        strcat(strcpy(szDrvFile,pPtr),"\\DRV\\") ;
        strcat(szDrvFile,szDrvName);

        /* NotDone set to 0, if found */
```

```
        NotDone = findfirst(szDrvFile,&stFile,0) ;
    }

    /* locate driver in current directory */
    if (NotDone)
    {
        strcpy(szDrvFile,szDrvName) ;

        /* NotDone set to 0, if found */
        NotDone = findfirst(szDrvFile,&stFile,0) ;
    }

    if (NotDone)
        printf("Driver file does not exist.\n") ;
    else
    {
        if ((Handle=_open(szDrvFile,O_RDONLY)) == -1)
            printf("Open %s error.\n",szDrvFile) ;
        else
        {
            wDrvSize = (unsigned) filelength(Handle) ;

            if ( allocmem((unsigned)((wDrvSize + 15) >> 4),&wDrvSeg)
== -1 )
            {
                lpDrvPtr = (char far *)((unsigned long)wDrvSeg << 16)
;
                lpPtr = lpDrvPtr;

                if ( DosReadDrv(Handle,lpPtr,wDrvSize,&wTemp) == 0 )
                {
                    freemem(wDrvSeg) ;
                    lpDrvPtr = 0 ;
                }
            }
            else
                printf("Memory allocation error.\n");

            _close(Handle) ;
        }
    }

    return(lpDrvPtr) ;
}

/* ---------------------------------------------------------------------
------ */
/*  @@ Usage
*/
/*
*/
/*   DosReadDrv (int Handle, char far *Buffer,
*/
/*                unsigned wLen, unsigned *lpByteRead)
*/
/*
*/
/*   DESCRIPTION:
*/
/*       Read driver to buffer using DOS interrupt 0x21 function 0
x3F.      */
/*
*/
```

```
/*    ENTRY:
*/
/*        Handle :- File handle to read.
*/
/*        Buffer :- Buffer to write to.
*/
/*        wLen   :- Number of byte to read.
*/
/*        lpByteRead :- pointer to number of byte actually read.
*/
/*
*/
/*    EXIT:
*/
/*        Byte read if successful, else returns 0.
*/
/*
*/
/* ------------------------------------------------------------------
------ */

DosReadDrv (int Handle, char far *Buffer, unsigned wLen, unsigned *
wByteRead)
{
    union REGS regs;
    struct SREGS segregs;


    regs.h.ah = 0x3f ;
    regs.x.bx = Handle;
    regs.x.dx = FP_OFF(Buffer);
    regs.x.cx = wLen;
    segregs.ds = FP_SEG(Buffer);

    intdosx(&regs, &regs, &segregs);

    if(regs.x.cflag)    /* error */
        *wByteRead = 0;
    else
        *wByteRead = regs.x.ax ;


    return(*wByteRead);
}
```

```
/* ------------------------------------------------------------------
-----
      MRECORD.C : For DSP (FILTER FIR) program !
      update time              21:57
                  date              15/5/94
      ------------------------------------------------------------
-------- */
#include <stdio.h>
#include <alloc.h>
#include <math.h>
#include  <io.h>
#include  <string.h>
#include  <stdlib.h>
#include  <dos.h>
#include  <bios.h>
#include  <fcntl.h>
#include "twindow.h"
#include "keys.h"
#include "project.h"
#include "sbcvoice.h"


extern System sys;
extern VOC_FILE *voc;
extern float far *tbuffer;
extern float far *pbuffer;

void Sampling (void)
{
      WINDOW *sam;
      unsigned int s;
      sam = establish_window (20, 15, 3, 25);
      set_colors(sam, ALL, BLUE, WHITE, BRIGHT);
      set_title (sam, " Sampling ");
      display_window (sam);
      wprintf (sam, "fs = ");
      cursor (27, 16);
      scanf ("%d",&s);
      sys.sampling = (long) s;
      delete_window (sam);
}

int Software_gen ()
{

      WINDOW *mn,*choose;
      double arg,twopi;
      unsigned int num,freq[20];
      int i,j;
      int s = 0;
      char **cp;
      char *titles[] = {
            " 1 : MAKE FREQUENCY ",
            " 2 : SWEEP FREQUENCY ",
            0
      };

      release ();

      sys.softwaregen = ON;
      sys.length = sys.sampling;
      cursor(0, 25);
      tbuffer = farcalloc (sys.length,sizeof (float));
      if (tbuffer == NULL)
            {
```

```
            print_error ("Out of memory for generator");
            return 2;
            }
      pbuffer = farcalloc (sys.length,sizeof (float));
      if (pbuffer == NULL)
            {
                  print_error ("Out of memory for large data");
                  return NULL;
            }
      mn = establish_window(5, 5, 4, 45);
      set_title(mn, " Software Generator ");
      set_colors(mn, ALL, BLUE, WHITE, BRIGHT);
      set_colors(mn, ACCENT, GREEN, WHITE, BRIGHT);
      display_window(mn);
      cp = titles;

      while (*cp)
            wprintf(mn, "\n%s", *cp++);
      while (1)   {
      s = get_selection(mn, s+1, "12");
            if (s==0) break;
            if (s==FWD || s == BS) {
                  s = 0;
                  continue;
            }
            choose = establish_window (20, 15, 3, 42);
            set_colors(choose, ALL, BLUE, WHITE, BRIGHT);
            switch (s)  {
                  case 1:  {
                                    display_window (choose);
                                    set_title (choose," Make
Frequency ");
                                    wprintf (choose, "Enter number of
frequencies in sum : ");
                                    cursor (58, 16);
                                    scanf ("%d", &num);
                                    clear_window (choose);
                                    cursor (21, 16);
                                    printf ("Enter frequency #");

                                    for (i=0;i < num; i++)  {
                                          cursor (38, 16);
                                          printf ("%d:",i+1);
                                          scanf ("%d",&freq[i]);
                                          cursor (40, 16); printf ("
");
                                    }
                                          set_cursor_type(-1);
                                    twopi = 8.0*atan(1.0);
                                    for (j=0; j < num; j++)   {
                                          arg = twopi*((double) freq
[j]/(double) sys.sampling);
                                    for (i=0; i < sys.sampling; i++)
                                          tbuffer[i] += 2.5*(cos
((double) i*arg));
                                    }
                                    for (i=0; i < sys.sampling; i++)
                                          tbuffer[i] = tbuffer[i]/
num;
                                    s--;
                                    break;
                                    }

                  case 2:
                                    display_window (choose);
```

```
Generator ");
sys.sampling;
((double) i * (double) i *arg));
: OK");
                                    set_title (choose, " Sweep
                                    arg = 2.0 * atan(1.0) / (double)
                                    for (i=0; i < sys.sampling; i++)
                                          tbuffer[i] = 2.5*(sin
                                    wprintf (choose, "Freq. 0 -- fs/2
                                    get_char ();
                                    s--;
                                    break;
            }
            delete_window (choose);
      }
      delete_window(mn);
      clear_message ();
}
```

```
/* ------------------------------------------------------------------
      RDWINDOW.C : For DSP (FILTER FIR) program !
------------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <mem.h>
#include <conio.h>
#include <math.h>
#include <graphics.h>
#include <io.h>
#include <fcntl.h>
#include <dos.h>
#include <bios.h>
#include <ctype.h>
#include "twindow.h"
#include "keys.h"
#include "sbc.h"
#include "sbcvoice.h"
#include "project.h"
#include "filter.h"
#include "filter.i"
#include "dft.h"

#define XAREA 598
#include "Loaddrv.c"

extern System sys;
extern VOC_FILE *voc;
extern float far *tbuffer;
extern float far *pbuffer;
extern char far * near ctvdsk_drv;

void recored (char *name);
void OutputFile (char *szFilename);
void PlayVoiceInBkgnd (void);
void RecordUntilStopped (void);

/* Function MENU for OUTPUT */
FILTER *fil_ptr = &fir_lpf;

#include "rdwindow.i"

void Speaker (void)
{
      WINDOW *wnd;
      char name[50];
      cursor(0, 25);
      wnd = establish_window(10, 10, 3, 45);
      set_title(wnd, " Speaker Output for sound card ");
      set_colors(wnd, ALL, BLUE, WHITE, BRIGHT);
      set_colors(wnd, ACCENT, GREEN, WHITE, BRIGHT);
      display_window(wnd);
      wputchar (wnd, ':');
      wcursor (wnd, 1, 0);
      set_cursor_type (1);
      scanf ("%s", name);
      set_cursor_type (-1);
      voice (name);
      delete_window (wnd);
}

void voice (char *name)
{
      extern  char far * near ctvdsk_drv;
```

```
        /* Retrieve the BLASTER environment settings */
        if ( ! GetEnvSetting() )
        {
         if (sbc_check_card() & 4)
         {
             if (sbc_test_int())
             {
                   if (sbc_test_dma() >= 0)
                    {
                     if ((ctvdsk_drv =
                         LoadDriver_Disk("CTVDSK.DRV")) != 0 )
                     {
                         if ( !ctvd_init(16) )
                         {
                          OutputFile(name) ;
                          ctvd_terminate() ;
                         }
                         else
                          ShowError() ;
                     }
                    }
                    else
                     print_error ("Error on DMA channel.");
             }
             else
              print_error ("Error on interrupt.");
         }
         else
             print_error ("Sound Blaster Card not found or wrong I/O
             settings.") ;
        }
 else
        print_error ("BLASTER environment not set or incomplete or
             invalid.");
}

/* ------------------------------------------------------
    @@ Usage
    OutputFile (char *szFilename)

    DESCRIPTION:
      Output voice with the filename specified.

    ENTRY:
      szFilename :- filename to be output.

    EXIT:
      None.

------------------------------------------------------------------ */

void OutputFile (char *szFilename)
{
    int     Handle ;

    /* Open and play the voice file */
    if ( (Handle=_open(szFilename, O_RDONLY)) != -1 )
    {
        ctvd_speaker(1) ;
              if ( ctvd_output(Handle) == NO_ERROR )
        {
            PlayVoiceInBkgnd () ;
            if ( ctvd_drv_error () )
                ShowError () ;
            else
```

```
                print_error ("Voice output ended.") ;
              }
         else
             ShowError () ;
         _close (Handle) ;
     }
        else
              print_error ( "Open file error") ;
}

/* ------------------------------------------------------------ */
/*  @@ Usage                                                    */
/*                                                              */
/*   PlayVoiceInBkgnd (void)                                    */
/*                                                              */
/*   DESCRIPTION:                                               */
/*   Control voice plaing at the background using keyboard.     */
/*                                                              */
/*   ENTRY:                                                     */
/*       None.                                                  */
/*                                                              */
/*   EXIT:                                                      */
/*       None.                                                  */
/*                                                              */
/* ------------------------------------------------------------ */

#pragma loop_opt(off)    /* turn off loop optimiaztion */
void PlayVoiceInBkgnd (void)
{
       /* Polls for hot key while playing voice file */
    while ( ct_voice_status )
    {
        if ( bioskey(1) )
        {
            switch ( bioskey(0) & 0xff )
            {
                case 0x1b : ctvd_stop () ;
                            print_error ("Voice stopped.") ;
                           break ;

                case ' '  : ctvd_pause () ;
                            print_error ( "Pause...\n"
                            "Pressed any key to continue." );
                            bioskey(0) ;
                            ctvd_continue () ;
                           break ;

                case 'b'  :
                case 'B'  : ctvd_break_loop(1) ;
                              print_error( "Break-out takes place
                                    immediately") ;
                             break ;

                case 'n'  :
                case 'N'  : ctvd_break_loop(0) ;
                              print_error( "Break-out takes place
                                    after the current"
                                    " loop finishes" ) ;
                           break ;
            }
        }
    }
}
#pragma loop_opt()
```

```
/* ------------------------------------------------------------------ */
/*  @@ Usage                                                          */
/*                                                                    */
/*   ShowError (void)                                                 */
/*                                                                    */
/*  DESCRIPTION:                                                      */
/*   Display error occurred during the process of voice I/O.          */
/*                                                                    */
/*   ENTRY:                                                           */
/*   None.                                                            */
/*                                                                    */
/*   EXIT:                                                            */
/*       None.                                                        */
/*                                                                    */
/* ------------------------------------------------------------------ */

void ShowError (void)
{
    int     Err ;
    char    str[25];

       /* Show the driver error and the DOS extended error code */
       Err = ctvd_drv_error() ;

       itoa (Err, str, 10);
       print_error(strcat ("Driver error = ", str)) ;

       Err = ctvd_ext_error();
       itoa (Err, str, 10);
       if ( Err != 0 )
             print_error (strcat ("DOS error = ", str)) ;
}


void Recordv (int h, int v)
{
      WINDOW *wnd;
      char name[50];
      wnd = establish_window (20, 15, 3, 50);
      set_colors (wnd, ALL, BLUE, WHITE, BRIGHT);
      set_title (wnd, " Record to file. ");
      display_window (wnd);
      wputchar (wnd, ':');
      wcursor (wnd, 1, 0);
      set_cursor_type (5);
      scanf ("%s", name);
      wcursor (wnd, 1, 0);
      set_cursor_type (-1);
      recored (name);
      get_char ();
      delete_window (wnd);
}
```

```
void recored (char *name)
{

       if ( ! GetEnvSetting() )
       {
              if (sbc_check_card() & 4)
              {
                     if (sbc_test_int())
                     {
                            if (sbc_test_dma() >= 0)
                            {
                                   if ((ctvdsk_drv =
                     LoadDriver_Disk("CTVDSK.DRV")) != 0)
                                   {
                                          if ( !ctvd_init(6) )
                                          {
                                           RecordFile(name) ;
                                           ctvd_terminate() ;
                                          }
                                          else
                                           ShowError() ;
                                   }
                            }
                            else
                             print_error("Error on DMA channel.");
                     }
                     else
                      print_error("Error on interrupt.");
              }
              else
             print_error("Sound Blaster Card not found or wrong I/O
settings.") ;
       }
       else
       print_error("BLASTER environment not set or incomplete or
invalid.");
}

/* ------------------------------------------------------------ */
/*  @@ Usage                                                    */
/*                                                              */
/*   RecordFile (char *szFilename)                              */
/*                                                              */
/*   DESCRIPTION:                                               */
/*       Record voice with the filename specified.              */
/*                                                              */
/*   ENTRY:                                                     */
/*       szFilename :- filename to be output.                   */
/*                                                              */
/*   EXIT:                                                      */
/*       None.                                                  */
/*                                                              */
/* ------------------------------------------------------------ */

void RecordFile (char *szFilename)
{
       int     Handle ;

       /* create and record a voice file */
       if ( (Handle=_creat(szFilename, 0)) != -1 )
       {
        ctvd_speaker(0) ;
            if (sys.sampling < 4000) sys.sampling = 4000;
        if ( ctvd_input(Handle, sys.sampling) == NO_ERROR )
```

```
                {
                    RecordUntilStopped() ;

                    if ( ctvd_drv_error() )
                           ShowError() ;
                          else
                           print_error ("Voice record ended.") ;
                }
                else
                          ShowError () ;

                _close (Handle) ;
        }
        else
                print_error ("Create file error") ;
}



/* ------------------------------------------------------- */
/*  @@ Usage                                               */
/*                                                         */
/*   RecordUntilStopped (void)                             */
/*                                                         */
/*   DESCRIPTION:                                          */
/*   Starts voice recording. Press ESC key to terminate the
     recording.                                            */
/*                                                         */
/*   ENTRY:                                                */
/*       None.                                             */
/*                                                         */
/*   EXIT:                                                 */
/*       None.                                             */
/*                                                         */
/* ------------------------------------------------------- */

#pragma loop_opt(off)   /* turn off loop optimiaztion */
void RecordUntilStopped (void)
{
        /* Polls for hot key while playing voice file */
        while ( ct_voice_status )
        {
                if ( bioskey(1) )
                {
                    /* check for ESC key */
                    if ( (bioskey(0) & 0xff) == 0x1b )
                           ctvd_stop () ;
                }
        }

        print_error (" Voice stopped.") ;
}
#pragma loop_opt()
```

```
/* ------------------------------------------------------------
      SAVE.C : For DSP program !
------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <share.h>
#include <string.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <dos.h>
#include <mem.h>
#include "filter.h"
#include "twindow.h"
#include "keys.h"
#include "sbcvoice.h"
#include "project.h"

#include "save.i"

#define BUFFER 0x20000L
#define OFFSET 32
#define NOCHECKBUFFER
//#define TEST_SAVE 1
void Save (void)
{
   int _flag = 0;
   char name[50];
   char far *lpVoiceBuf, *lpBuf;
   char huge *lpBuf_h;
   float far *stbuffer;       /* float pointer to voice quantizer */
   float far *spbuffer;       /* float pointer to process */
   unsigned statust, statusp;
   unsigned long *l, length, flag=1, num=1;
   WINDOW *wnd;
#ifndef TEST_SAVE
      if (voc == NULL) _flag = 1;
#endif
   if (_flag != 1)
      {
         wnd =  establish_window (20, 10, 3, 40);
      set_border (wnd, 1);
         set_colors (wnd, ALL, BLUE, WHITE, BRIGHT);
      set_title (wnd, " Save file name. ");
         display_window (wnd);
            wputchar (wnd, ':');
         wcursor (wnd, 1, 0);
      scanf ("%s", name);
      set_cursor_type (-1);
#ifdef TEST_SAVE
      strcpy (voc->name, "hello.voc");
#endif
      if ( (lpVoiceBuf = LoadFile(voc->name)) != 0)
         {
            /* success */
            lpBuf_h = (char far *) lpVoiceBuf;
                  lpBuf_h += ((VOCHDR far *) lpBuf_h) ->
                              voice_offset;
            do {
                  switch (lpBuf_h[0])
                  {
                        case 0x01:  {
                              /* yes == blocktype 1*/
                              gotoxy (1,1); printf ("%d ", num);
```

```
                                num++;
                                lpBuf_h = lpBuf_h + 1;
                                /* shift lpBuf_h to blklen */
                                l =  (unsigned long *) lpBuf_h;
                                /* pointer to block len */
                                length = *l & 0x00FFFFFF;
                                /* length is length of block len */
                                length = length - 2L;
                                lpBuf_h = lpBuf_h+5;
                                /* pointer to data voice */
                                lpBuf = (char far *) lpBuf_h;
                                stbuffer = quantizer (lpBuf, &statust,
                                           length);
                                spbuffer = convolution (stbuffer, &
                                            statusp, length);
                                if (spbuffer == NULL) flag = 0;
                                else requantizer (lpBuf, spbuffer,
                                      length);
                           break;
                        }
                case 0x00:  flag = 0; break;
                           }
                           lpBuf_h = lpBuf_h+length;
                           if (statust == NO_ERROR)
                              freemem (FP_SEG(stbuffer));
                           if (spbuffer != NULL) {
                           if (statusp == NO_ERROR)
                           freemem (FP_SEG(spbuffer));
                }
                } while (flag != 0);
                lpBuf = lpVoiceBuf;
                lpBuf += ((VOCHDR far *) lpBuf) -> voice_offset;
                SaveVoiceFile (name, lpBuf);
        freemem (FP_SEG(lpVoiceBuf));
                }
                else {
                /* insuccess */
                     print_error ("Can't open file for funtion save.c");
                }
                delete_window (wnd);
        }
}
```

```
/* ------------------------------------------------------------
      STWINDOW.C : For DSP program !
------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <graphics.h>
#include "twindow.h"
#include "keys.h"
#include "sbcvoice.h"
#include "project.h"
#define XAREA  598
#define ROUND(a)  (((a) < 0) ? (int)((a)-0.5) : (int)((a)+0.5))
#define STEP2 0.0390625
extern System sys;
extern VOC_FILE *voc;
extern float far *tbuffer;
extern float far *pbuffer;

char *titles[] = {
      " 1 : ÑÇ²¥°Ã·Ò Time Domain (Default)",
      " 2 : ÑÇ²¥°Ã·Ò Frequency Domain ",
      " 3 : Zoom in (Default)",
      " 4 : Zoom out ",
      0
};

void St_window (void)
{
      int s = 0;
      static int i=0;
      static int o=0;
      static WINDOW *mn,*choose;
      char **cp;

      cursor(0, 25);

      mn = establish_window(5, 5, 6, 45);
      set_title(mn, " ¡ÍÈ·²¡ÍÁÑÇ²¥°Ã¢Ê¥ WINDOW ");
      set_colors(mn, ALL, BLUE, WHITE, BRIGHT);
      set_colors(mn, ACCENT, GREEN, WHITE, BRIGHT);
      display_window(mn);
      cp = titles;

      while (*cp)
            wprintf(mn, "\n%s", *cp++);
      while (1)   {
      s = get_selection(mn, s+1, "12345");
            if (s==0) break;
            if (s==FWD || s == BS) {
                  s = 0;
                  continue;
            }
            choose = establish_window (20, 15, 3, 25);
            set_colors(choose, ALL, BLUE, WHITE, BRIGHT);
            switch (s)  {
                  case 1:  {
                        set_title (choose, "Select Time Domain");
                                    display_window (choose);
                                    wprintf (choose, "          OK");
                                    get_char ();
                                    hide_window (choose);
                                    sys.time_freq = TIME;
                                    s--;
```

```
                                        break;
                                        }
                        case 2:  set_title (choose, "Select Frequency
                                        Domain");
                                        display_window (choose);
                                        wprintf (choose, "          OK");
                                        get_char ();
                                        hide_window (choose);
                                        sys.time_freq = FREQ;
                                        s--;
                                        break;
                        case 3:
                                        if (sys.zoom == ZOOMOUT) i = 0;
                                        sys.zoom = ZOOMIN;
                                        i += 1;
                                        sys.zoom_value = i;
                                        set_title (choose, " ZOOM IN ");
                                        display_window (choose);
                                        wprintf (choose, "          %
                                        d",sys.zoom_value);
                                        get_char ();
                                        hide_window (choose);
                                        s--;
                                        break;
                        case 4:
                                        if (sys.zoom == ZOOMIN) o = 0;
                                        sys.zoom = ZOOMOUT;
                                        o += 1;
                                        sys.zoom_value = o;
                                        if (sys.zoom_value >=
//
sys.sampling/XAREA)
                                        if (sys.zoom_value >=
sys.length/XAREA)
                                        {
                                                sys.zoom_value = 1;
                                                o = 0;
                                        }
                                        set_title (choose, " ZOOM OUT ");
                                        display_window (choose);
                                        wprintf (choose, "          %
d",sys.zoom_value);
                                        get_char ();
                                        hide_window (choose);
                                        s--;
                                        break;
                }
                delete_window (choose);
        }
        get_char ();
        delete_window (mn);
}

#ifndef TEST
void Display (void)
{
   float loop, ypos;
        register i;
        register x;
        unsigned step, op;
        float far *lpBuf;
//      FILTER *fir_ptr;
        double time;
        char str[25];
        int dummy;
        lpBuf = (float far *) tbuffer;
```

```
      if (sys.softwaregen != ON) {
            if (voc == NULL) dummy = -1;
            if (tbuffer == NULL) dummy = -1;
      }
      if (sys.sampling < XAREA) {dummy = -1;}
      if (dummy != -1)  {
      Graphics_interface ();

      outtextxy (22,22, "t = ");
      if (sys.zoom == ZOOMIN) {
            loop = XAREA/sys.zoom_value;
            time = (double) (XAREA) / (sys.zoom_value * sys.sampling)
;
            gcvt (time, 5, str); outtextxy (50,21,str);

            setcolor (EGA_BLUE);
            line (20, 130, 618, 130);
                        for (x=0; x < loop; x++)
                        {
                              ypos = lpBuf[x];
                              i = x*sys.zoom_value;
//   12 / 7 / 94 remark ¹Õéà»ç¹ÊèÇ¹·Õè¨Ð¤Ó¢Õé¹ÁÒãËÁè ·ÐË¹éÒ·Õèá¡¡ÒÃ
plot
// ÍÍ¡à»ç¹ +, - â´Âà´ç´¢Ò´ áµèÂÑ§·Ø§Ò¹ãÁèã´é´Õ¹Ñ¡
/*                            if (x==0) moveto (i+20, 130-(ypos/0.3))
; */
/*                            if (ypos > 0)
                                    lineto (i+20, 130 - (ypos/0.3));
                              if (ypos < 0)
                                    lineto (i+20, 130 - (ypos/0.3));
*/
// µéÍ§¡ÒÃãËéÁÑ¹ã¹éÊÀÒ¾à´ÔÁã»¡èÍ¹
                              if (x==0) moveto (i+20, 130 - (ypos /
STEP2));
                              lineto (i+20, 130-(ypos/STEP2));

                        }
                              setcolor (EGA_RED);
                        if (sys.time_freq == TIME) {
                              Display4 (loop);
//                  adjust (&op);
                              for (x=0; x < loop; x++)
                              {
                                    i = x*sys.zoom_value;
                                    if (x==0) moveto (i+20, 350-
(pbuffer[x]/STEP2));
                                    lineto (i+20, 350 - (pbuffer[x]/
STEP2));
                              }
                        } else if (sys.time_freq == FREQ) {
//                                  Display5 (loop);
                                    Fastcon (loop);
                              for (x=0; x < loop; x++)
                              {
                                    i = x*sys.zoom_value;
                                    if (x==0) moveto (i+20, 420-
                                    (pbuffer[x]/STEP2));
                                    lineto (i+20, 420 - (pbuffer[x]/
                                    STEP2));
//                                  putpixel (i+20, 420 - (pbuffer[x]
                                    /STEP2),EGA_BLUE);
                              }
                        }
      } else /* if == ZOOMOUT */
      {
```

```
                loop = XAREA*sys.zoom_value;
                time = (double) (XAREA * sys.zoom_value) / sys.sampling;
                gcvt (time, 5, str); outtextxy (50,21,str);
                        setcolor (EGA_BLUE);
                        for (x=0; x < loop; x++)
                        {
                              ypos = lpBuf[x];
                              i = x/sys.zoom_value;
                              if (x==0) moveto (i+20, 130-
                              (ypos/STEP2));
                              lineto (i+20, 130 - (ypos/STEP2));
                        }
                        setcolor (EGA_RED);
                        if (sys.time_freq == TIME) {
                                    Display4 (loop);
                                    adjust (&op);
                                    for (x=0; x < loop; x++)
                                    {
                                          i = x/sys.zoom_value;
                                          if (x==0) moveto (i+20,
                                          350-(pbuffer[x]/STEP2));
                                          lineto (i+20, 350 -
                                          (pbuffer[x]/STEP2));
                                    }
                        } else if (sys.time_freq == FREQ) {
//                                  Display5 (loop);
                                    Fastcon (loop);
                                    for (x=0; x < loop; x++)
                                    {
                                          i = x/sys.zoom_value;
                                          if (x==0) moveto (i+20,
                                          420-(pbuffer[x]/STEP2));
                                          lineto (i+20, 420 -
                                          (pbuffer[x]/STEP2));
//                                        putpixel (i+20, 420 -
                                    (pbuffer[x]/STEP2), EGA_BLUE);
                                    }
                        }
        }
        get_char ();
        closegraph ();
        }
}
#endif

int min_max (float *max, float *min)
{
   register i;
   *max = *min = pbuffer[0];
      for (i = 1; i < sys.length; i++)
   {
      if (*max < pbuffer[i]) *max = pbuffer[i];
      if (*min > pbuffer[i]) *min = pbuffer[i];
   }
}

void adjust (unsigned *op)
{
        switch (sys.type_fil)
   {

        case DHPF:
        case HPF:
        case BPF:
        case DBPF:  *op = 60; break;
```

```
        case LPF:
        case BSF:
        case DLPF:
        case DBSF:  *op = 0;      break;

        default: *op = 0;
        }
}
```

```
/* ------------------------------------------------------------
      DESIGN.I : For DSP (FILTER FIR) program !
   ------------------------------------------------------------ */
/* Function MENU for FILTER TYPE */
void Lpf (void)
{
      fil_ptr = &fir_lpf;
   sys.type_fil = LPF;
}
void Hpf (void)
{
      fil_ptr = &fir_hpf;
   sys.type_fil = HPF;
}
void Bpf (void)
{
      fil_ptr = &fir_bpf;
   sys.type_fil = BPF;
}

void Bsf (void)
{
      fil_ptr = &fir_bsf;
   sys.type_fil = BSF;
}

/* Function MENU for OPTION */
void Time_domain (void)
{
      sys.time_freq = TIME;
}

void Freq_domain (void)
{
      sys.time_freq = FREQ;
}



void hamming (void)
{
      int    i;
      double ham,factor;
   unsigned n;
   FILTER *dummy;
   switch (sys.type_fil)
   {
      case LPF:   dummy = &fir_lpf; break;
      case HPF:   dummy = &fir_hpf; break;
      case BPF:   dummy = &fir_bpf; break;
      case BSF:   dummy = &fir_bsf; break;
      case DLPF:
      case DHPF:
      case DBSF:
            case DBPF:  dummy = &fir_transform; break;
      default: dummy = &fir_lpf;
   }
   fil_ptr = &fir_transform;
      n = fil_ptr->length;
      factor = 8.0*atan(1.0)/(n-1);
      for (i = 0 ; i < n ; i++){
       ham = 0.54 - 0.46*cos(factor*i);
       fil_ptr->coef[i] = dummy->coef[i] * ham;
      }
}
```

```
void hanning (void)
{
      int    i;
      double han,factor;
   unsigned n;
   FILTER *dummy;
   switch (sys.type_fil)
   {
      case LPF:   dummy = &fir_lpf; break;
      case HPF:   dummy = &fir_hpf; break;
      case BPF:   dummy = &fir_bpf; break;
      case BSF:   dummy = &fir_bsf; break;
      case DLPF:
      case DBPF:
      case DBSF:
      case DHPF:  dummy = &fir_transform; break;
      default: dummy = &fir_lpf;
   }
   fil_ptr = &fir_transform;
      n = fil_ptr->length;
      factor = 8.0*atan(1.0)/(n-1);
      for (i = 0 ; i < n ; i++){
       han = 0.5 - 0.5*cos(factor*i);
       fil_ptr->coef[i] = dummy->coef[i] * han;
      }
}
void blackman (void)
{
  unsigned   n,i;
  double black,factor;
  FILTER *dummy;

   switch (sys.type_fil)
   {
      case LPF:   dummy = &fir_lpf; break;
      case HPF:   dummy = &fir_hpf; break;
      case BPF:   dummy = &fir_bpf; break;
      case BSF:   dummy = &fir_bsf; break;
      case DLPF:
      case DHPF:
      case DBSF:
      case DBPF:  dummy = &fir_transform; break;
      default: dummy = &fir_lpf;
   }
   fil_ptr = &fir_transform;
      n = fil_ptr->length;
      factor = 8.0*atan(1.0)/(n-1);
        for (i=0; i<n; ++i){
       black = 0.42 - 0.5*cos(factor*i) + 0.08*cos(2*factor*i);
       fil_ptr->coef[i] = dummy->coef[i] * black;
      }
}
```

```
/* dft.h - function prototypes and structures for dft and fft
functions */

/* COMPLEX STRUCTURE */

typedef struct {
    float real, imag;
} COMPLEX;

/* function prototypes for dft and inverse dft functions */

    extern int  fft(COMPLEX *,int);
    extern int log2(unsigned int);
```

```
/*  FILTER.H */
/* FILTER INFORMATION STRUCTURE FOR FILTER ROUTINES */

typedef struct filter {
       unsigned int length;        /* size of filter */
       float far *history;             /* pointer to history in filter
*/
       float far *coef;                /* pointer to coefficients of
filter */
} FILTER;

/* filter function prototypes */

       extern void fir_filter_array(float *,float *,unsigned
long,FILTER *);

/* note: if you have a compiler that passes floats then change
                the two double declarations to float.  This problem
is
                a result of using the old style function
declarations. */
       extern float fir_filter(double,FILTER *);
       extern float iir_filter(double,FILTER *);
```

```
/* get.h - prototypes for get.c functions and common macros */

    extern char *get_string(char *);
    extern int get_int(char *,int,int);
    extern double get_float(char *,double,double);

/* MIN, MAX, ROUND macros */

#define MAX(a,b)     (((a) > (b)) ? (a) : (b))
#define MIN(a,b)     (((a) < (b)) ? (a) : (b))
#define ROUND(a)  (((a) < 0) ? (int)((a)-0.5) : (int)((a)+0.5))
```

```
#define MAX_COEF        100
#define NOT_SUPPORT 0xFFFF
#define STEP       0.01953125
#define NO_ERROR   0
#ifndef ON
      #define ON   1
#endif

#ifndef OFF
      #define OFF 0
#endif
#define ZOOMIN          1
#define ZOOMOUT    2
#define XAREA      598
#define _FIR       0x0001
#define _IIR       0x0002

#define LPF        0x0001
#define HPF        0x0002
#define BPF        0x0003
#define BSF        0x0004
#define DLPF    0x0005
#define DHPF       0x0006
#define DBPF       0x0007
#define DBSF       0x0008
#define TIME       0x0004
#define FREQ       0x0005

#define RUNTIME    0x8000
#define BESTTIME   0x0800

typedef struct {
                    unsigned long sampling;
                    unsigned long length;
                    unsigned int softwaregen;
                    unsigned int time_freq;
                    unsigned    int best_run;
                    unsigned int ptime_freq;
                    unsigned int methord;
                    unsigned int type_fil;
                    int                 zoom;
                    int                 zoom_value;
                    char *gtitle; /* title for graphics mode */
                    float *coef;
                } System;

typedef struct
                {
                    char id[20] ;
                    unsigned   voice_offset ;
                    unsigned   version ;
                    unsigned   check_code ;
                    unsigned int sampling ;
                    unsigned int tc;
                    unsigned int pack;
                    unsigned long length_file;
                    unsigned long length_data;
                    char *name;
                    FILE *fp;
                } VOC_FILE;

typedef struct
                {
                    char blktype[1];
                    char blklen[3];
```

```
                    unsigned int tc_pack;
                } BLOCKTYPE1;

typedef struct
                {
                    char blktype[1];
                    char blklen[3];
                } BLOCKTYPE2;

void voice (char *name);
char far *LoadFile (char *szFilename);
int DosRead (int Handle, char far *Buffer,
             unsigned wLen, unsigned *wByteRead);
int OutputVoice (char far *lpBuf);

void Recordv (int , int );
void RecordFile (char *szFilename);
void RecordUntilStopped (void);
void ShowError (void);

void Display2 (float);
void Display3 (float ,int x);
void Display4 (float);
int min_max (float *max, float *min);
void adjust (unsigned *op);
int Fastcon (float);
unsigned Display5 (float);

/* external function for this program */
void Graphics_interface (void);

/* General Function */
VOC_FILE *open_read_voc (char *file_name);
void readblk (VOC_FILE *voc,BLOCKTYPE1 *blkbuf);
void readblk2 (VOC_FILE *voc,BLOCKTYPE2 *blkbuf);
void release (void);
void far *read_data (VOC_FILE *voc);

/* Function Prototypes */
SaveVoiceFile_A(char *,char far *) ;
RecordVoice (char *szFilename) ;
Recording (char far *lpBuf,long lBufSize) ;
WriteToFile (int Handle, char far *lpBuf, long lSize) ;
DosWrite (int Handle, char far *Buffer,
       unsigned wLen, unsigned *wByteWritten) ;
void release (void);

void design_lpf (WINDOW *);
void design_hpf (WINDOW *);
void design_bpf (WINDOW *);
void design_bsf (WINDOW *inter);

int save_design (WINDOW *inter, char *name, float *data, unsigned N);
int clpf (unsigned fc, unsigned N, float *coef);
int odd_even (unsigned n);
void max_min (float *max, float *min, float *buf, unsigned);

void Save (void);
```

ภาคผนวก

SIGSYS
A Signals and Systems Interpreter
TUTORIAL

Description. SIGSYS is an interpreter that has been designed to generate, manipulate and display time and frequency signals. A wide and flexible range of operations is supported.

Contents

seven digits precision. The precision of the display may be changed to 15 digits with the command

```
format long
```

and changed back to seven digits with

```
format short
```

Also, by default SIGSYS displays complex numbers in Cartesian form. The display may be changed to polar form by the command

```
format polar
```

In this format, the complex number $1 + j$ is shown as

```
1.414214*exp(j*0.7853982)
```

The display may be changed back to Cartesian form by the command

```
format cartesian
```

The single command

```
format
```

reverts the display to default.

2.3. Exercise. Define the variables a and b as in 2.1, and display them in polar and Cartesian format and with long and short precision. ■

3. Elementary Data Management and Command Line Editing

At any time during a session, the command

```
who
```

shows a list of the variables that have been created so far, together with their types. The command

```
clear
```

clears all variables, while

```
clear a,x,A
```

clears a, x and A only. It is useful to know that a session may be peacefully ended at any time by typing

```
exit
```

and that SIGSYS accepts the keyboard interrupts

⟨CTRL⟩S	halt execution,
⟨CTRL⟩Q	resume execution,
⟨CTRL⟩C	cancel the current operation.

⟨CTRL⟩S means that the CTRL key is pressed together with the (lower case) s key.

At any point during a session all variables and their values may be saved on disk by typing for instance

```
save data
```

This creates a file on disk called DATA.DAS. The extension .DAS is added by SIGSYS, and no extension should be specified. If the file name "data" is omitted, the default file name SIGSYS is used. In file names no distinction is made between upper and lower case characters. Saved data may be retrieved by typing

```
load data
```

where any existing SIGSYS data file name (with extension .DAS) may be used instead of "data". The variables in the data file that is loaded are merged with the existing variables. Variables with duplicate names are overwritten by the file that is loaded.

3.1. Exercise. Display the variables you have created so far and clear some of them. Save your variables, exit SIGSYS and enter it again, and load the saved data. ■

The rest of this section may be skipped on first reading.

If no more memory is available for storing variables, SIGSYS displays an error message. Free space may be created by

clearing variables. The stored variables may be reorganized to make more efficient use of the available space by the command

```
pack
```

This command first saves all variables on disk in a temporary file, then clears all variables, next reloads the variables from file and finally erases the temporary file.

The effectiveness of the ⟨CTRL⟩c command is increased by issuing the DOS command BREAK ON before starting SIGSYS (or by issuing the shell command !BREAK ON from SIGSYS—see 18.) This facility slows down execution; it may be switched off by the DOS command BREAK OFF.

By default, .DAS files are written to and read from the current disk and sub-directory. The command chdir ("change directory") may be used to change this. Thus,

```
chdir c:\sigsys\data
```

changes the default directory to C:\SIGSYS\DATA. The same may be done *before* starting SIGSYS by the command (from DOS)

```
SET SIGSYS=C:\SIGSYS\DATA
```

where of course any other valid path name may be used.

SIGSYS has a simple but effective facility for command line editing. While typing a command line, the cursor may be moved back and forth by the ← and → keys, and the ⟨INS⟩, ⟨DEL⟩ and ⟨BACKSPACE⟩ keys have their usual functions. By means of the ↑ and ↓ keys commands that recently have been issued may be recalled and displayed. They may be re-issued, possibly after editing, by pressing the ⟨ENTER⟩ key.

3.2. Exercise. Experiment with the command line editing facility. ■

4. Operations Among Scalars

The following operations among scalars are defined

a^b	exponentiation,
-a	unary minus,
a*b	multiplication,
a/b	division,
a+b	addition,
a-b	subtraction,
a mod b	reduction of a modulo b

The operations apply both to real and complex numbers. If a and b are complex, the modulo operation is applied to the real and imaginary parts separately and returns a complex number, that is, a mod b is the complex number with real part Re (a) mod Re (b) and imaginary part Im (a) mod Im (b)

The scalars may be entered by name or by value. Operations may be combined to algebraic expressions. The usual precedence rules apply. Parentheses may be used to change the natural precedences.

4.1. Exercise. To compute the root to the base 10 of *j* type

```
j^(1/10)
```
■

4.2. Exercise. Compute

```
11+12j mod 3+4j
```

See what the outcome of `5 mod 0` is. ○ ■

Typing an expression without assigning it to a variable produces the result directly on the screen. The result is stored in a variable named ans. The previous value of ans is overwritten.

4.3. Example. Typing

```
2+5
ans/7
```

results in successive displays of 7 and 1. After completion, ans has the value 1. ■

5. Functions of Scalars

A number of standard functions have been implemented. They are the following

`real(a)`	real part,
`imag(a)`	imaginary part,
`abs(a)`	absolute value,
`arg(a)`	argument,
`conj(a)`	complex conjugate,
`exp(a)`	exponential,
`log(a)`	natural logarithm,
`log10(a)`	logarithm to the base 10,
`sqrt(a)`	square root,
`sin(a)`	sine,
`cos(a)`	cosine,
`tan(a)`	tangent,
`asin(a)`	arcsine,
`acos(a)`	arccosine,
`atan(a)`	arctangent,
`sinc(a)`	`sin(a)/a`,
`sign(a)`	sign,
`floor(a)`	largest integer less than or equal to a,
`ceil(a)`	smallest integer greater than or equal to a,
`round(a)`	nearest integer to a,
`step(a)`	1 for $a \geq 0$, 0 otherwise,
`ramp(a)`	a for $a \geq 0$, 0 otherwise,
`rect(a)`	1 for $-1/2 \leq a < 1/2$, 0 otherwise,
`trian(a)`	$1-\|a\| \leq 1$, for $\|a\| \leq 1$, 0 otherwise,
`sat(a)`	a for $\|a\| \leq 1$, `sign(a)` otherwise.

The operations are all defined for real and complex argument a. If the argument a is complex, the functions `sign`, `floor`, `ceil`, `round`, `step`, `ramp`, `rect`, `trian` and `sat` operate on the real and imaginary parts separately, and return a complex number. Thus, `sign(0.1-5*j)`, for instance, returns the complex number `1-j`.

5.1. Exercise. Take `a=pi/2` and `b=exp(j*a)`. Check that the real part, the imaginary part, the absolute value and the argument of `b` are as expected. Check some of the other functions as well. ■

Note. Complex scalars with imaginary part equal to zero are automatically converted to real scalars.

6. Creating Time Signals

A time signal is handled by the program as a finite-time discrete-time signal. A signal is represented by the following attributes:

The *increment*, which is the distance between the sampling instants.

The *domain*, characterized by two real numbers, denoting the lower and the upper limit of the time axis on which the signal is defined. The difference between the upper and lower limit is an integral multiple of the increment.

A *list* of real or complex numbers, representing the signal values at the sampling instants.

A direct way of creating a time signal is by using the `cat` function. `Cat` is a contraction of *concatenation*. The command

```
x=cat(2,3,1)
```

creates a real signal `x`, whose increment is 1, whose domain is {0, 2} and that takes the successive values 2, 3, and 1. The `cat` function uses the increment `inc`, which is a permanent variable with default value 1. The list of arguments of the `cat` function (which may have arbitrary length) specifies the successive signal values. The first signal value is placed at the time origin. An exclamation mark may be used to change the location of the time origin. Thus,

```
y=cat(5,4!3,2,1)
```

creates a signal with increment 1, domain $[-1, 3]$ and successive values $y(-1) = 5$, $y(0) = 4$, $y(1) = 3$, $y(2) = 2$ and $y(3) = 1$.

6.1. Exercise. Create the signals x and y. Display them. ∎

The cat function may not only be used to define signals from their sampled values, but also to concatenate signals that already exist. The general format of the command is

```
u=cat(a,b,...,m!n,...,z)
```

Here a, b, ..., z are either scalars or signals. The signals must all have the same increment. The new signal is formed by concatenation of the signal and scalar values. The exclamation mark indicates the location of the origin. If m is a scalar, this signal value is placed at the origin. If m is a signal, the origin of m becomes the origin for u. If no exclamation mark is present, the signal starts at time 0.

6.2. Exercise. Define x=cat(2,3,1) and y=cat(5,4!3,2,1) as before. Check that cat(x,y!) is the same as cat(2,3,1,5,4!3,2,1) and that cat(x,y) is cat(2,3,1,5,4,3,2,1). What is cat(x!y)? ∎

Once a signal has been defined, four functions may be used to extract some of its attributes:

inc(x)	returns the increment of x,
num(x)	returns the number of samples of x,
low(x)	returns the lower limit of the domain of x,
up(x)	returns the upper limit of the domain of x.

6.3. Exercise. Verify that inc(y), num(y), low(y) and up(y), with y as in Exercise 6.2, result in the expected answers. ∎

There are two important permanently defined time signals, named dot and dotplus. In Section 9 it is explained how they may be used to generate a wide class of signals. The signal dot has increment inc (with default value 1), two-sided domain

$$\left[-\frac{\text{num}}{2}\,\text{inc},\ \left(\frac{\text{num}}{2}-1\right)\text{inc}\right]$$

(provided num is even), and signal values specified by $\text{dot}(t) = t$. Here num is another permanent variable with default value 10. Displaying dot results in something like

```
signal specifiers
num=10
inc=1
domain=[-5,4]
signal data
signal axis          signal value
   -5                    -5
   -4                    -4
   -3                    -3
   -2                    -2
   -1                    -1
    0                     0
    1                     1
    2                     2
    3                     3
    4                     4
```

If num is odd, the signal dot has domain

$$\left[-\frac{\text{num}-1}{2}\,\text{inc},\ \frac{\text{num}-1}{2}\,\text{inc}\right].$$

The signal dotplus has increment inc, right one-sided domain

[0, (num-1)inc],

and signal values specified by dotplus(t) = t.

It will be seen in Section 9 that the main function of the signals dot and dotplus is to define a time axis. A more flexible way of doing this is with the command a:b:c, with a, b and c three real numbers with b positive and c ≥ a. The command

```
x=a:b:c
```

creates a signal with increment b and domain [a, c] such that x(t) = t. If necessary, a is rounded up and c is rounded down so that a and c are integral multiples of b. The real scalars a, b and c may be called by name or by value. Thus, the command

```
x=2:0.5:4
```

creates a signal x with increment 0.5, domain [2, 4] and successive values y(2) = 2, y(2.5) = 2.5, y(3) = 3, y(3.5) = 3.5 and y(4) = 4.

6.4. Exercise. Display dot, dotplus, and the signal 2:0.5:4. ∎

A further permanent time signal is delta. It has the same increment and domain as dot (and hence is two-sided), while its signal values are given by delta(0) = 1/inc and delta(t) = 0 for $t \neq 0$.

Besides dot, dotplus, and delta another permanent time signal is noise. The command

```
z=noise
```

generates a signal z with increment inc and two-sided time axis, whose successive values equal random numbers. The successive random signal values are generated as rand/$\sqrt{\text{inc}}$ (see the description of the permanent scalar rand). The command

```
z=noiseplus
```

generates a similar random signal but on a right one-sided time axis with increment inc and one-sided domain.

6.5. Exercise. Generate the noise signal z. See what happens if you change the distribution. ∎

Note. Complex signals that have their imaginary part equal to zero are automatically converted to real signals.

7. Changing the Default Values

The scalars j and pi are permanent and cannot be changed. In addition to the permanent variables inc and num that we already met there is a third permanent variable, called INC. INC is the increment for frequency signals, which are discussed in Section 12. The default value of inc is 1, that of num is 10 and that of INC is 0.1. The default values of inc and num can be changed by simple assignment statements such as

```
inc=.01
```

or

```
num=100
```

Whenever a new value is assigned to inc, num or INC, the new values for inc, num and INC are all displayed.

The meaning of the permanent variables is the following. That of inc, the increment for time signals, is clear. The variable num defines the number of sampling instants of the standard time signals dot and dotplus. Together, inc and num define the time axes of the standard time signals dot and dotplus and via these of all signals derived from dot and dotplus. How this is done is discussed in Section 9. Given inc and num, the domain of dot is taken (more or less) symmetrically with respect to the origin (imitating a two-sided time axis) and that of dotplus at the right-

hand side of the origin (imitating a right one-sided time axis).

7.1. Exercise. Change `inc` to .01 and `num` to 100. Verify that the lower limit `low(dot)` and the upper limit `up(dot)` and those of `dotplus` are as expected. Try also `num = 101`. ■

There are three `set` commands that may be used to set some of the defaults automatically. They are mainly useful when simultaneously working with time signals and their Fourier transforms (see Section 17). The commands are the following

`set INC`	make `INC` equal to `1/(num*inc)`,
`set inc`	make `inc` equal to `1/(num*INC)`,
`set num`	round up `num` to the next integral power of 2.

8. Operations Among and on Signals

The following pointwise operations among signals are defined:

`x^y`	pointwise exponentiation,
`-x`	pointwise unary minus,
`x*y`	pointwise multiplication,
`x/y`	pointwise division,
`x+y`	pointwise addition,
`x-y`	pointwise subtraction,
`x mod y`	pointwise reduction of x modulo y.

In each of these cases x and y must have the same increments. The domain of the signal that results from any of the binary operations is the smallest closed interval that includes the domains of the operands. Signal values of the operands outside their domains are set equal to zero. If x or y is a scalar (called by name or by value), it is taken as a constant signal. If x and y are complex-valued, then the modulo reduction operates on the real and imaginary parts separately, as in the case of scalars.

8.1. Exercise. Define `x=cat(2,3,1)` and `y=cat(5,4!3,2,1)` as before. Compute `x + y` and verify the result. Also check some of the other operations from the list. Is `dot mod 4` what you expect it to be? How about `(-1)*dot`? ■

Two important binary operations between signals are not defined pointwise. They are

`x**y`	fast convolution,
`x oo y`	fast cross-correlation.

The spaces before and after `oo` are important, like those before and after `mod`. The signals x and y need have the same increments. Both x and y are taken to be zero outside their domains. The domains of the convolution and the cross-correlation of x and y are made equal to their supports.

The fast convolution `x**y` and cross-correlation `x oo y` use the fast Fourier transform as an intermediate step. When the command `x**y` is invoked, first the discrete Fourier transforms of the signals x and y are computed, next the transforms are multiplied, and finally the inverse discrete Fourier transform of the product is calculated. Before the transforms are determined the signals x and y are padded with sufficiently many zeros. At the end of the computation the domain of the result `x**y` is trimmed to the correct interval. For the correlation `x oo y` a similar algorithm is employed.

The commands

`x*.*y`	direct convolution,
`x o.o y`	direct cross-correlation,

also perform convolution and cross-correlation, respectively, but do it directly by first computing the convolution or cross-correlation sum applied to the lists of signal values, and multiplying the result by the (common) increment of x and y. For long signals (over 100 samples or more) fast convolution and correlation are much quicker than the direct algorithms.

8.2. Exercise. Define `x=cat(2,3,1)` and `y=cat(5,4!3,2,1)` as before. Compute their convolution z using the fast and direct algorithms. Verify that `x**y = y**x`. ■

8.3. Exercise. Set num equal to a fairly large number (e.g. num = 256), and create some time signal x (e.g. `x=dot`). Compute the convolution of x with itself by the fast and the direct algorithms, and compare the computation times as well as the results. ■

Two further commands are available to compute the cyclical convolution of two time signals. They are

`x cc y`	fast cyclical convolution,
`x c.c y`	direct cyclical convolution,

Before computing the cyclical convolution of two signals, any parts of the signals defined for negative times are truncated. Next, for positive times the shortest signal is padded with zeros so that the signals have the same length. The two resulting signals are the starting point for the direct cyclical convolution. For fast cyclical convolution the signals are further padded with zeros to a length consisting of a number of samples that is an integral power of 2, and the fast Fourier transform is used. As a result, fast and direct cyclical convolution may not produce the same outcomes.

9. Functions and Functionals of Signals

All the functions that are introduced in Section 5 also work when the argument a is a signal. The evaluation is pointwise. Thus, assuming that inc = 1 and num = 10,

```
x=sin(dot)
```

creates a signal x with increment 1, domain [−5, 4] and signal values sin(−5), sin(−4), · · · , sin(0), · · · , sin(4). In this way, using the standard functions of Section 5 and the permanent time signal dot, a variety of well-known elementary time signals may be created, such as real and complex harmonic signals, steps, ramps, and pulses.

9.1. Exercise. Verify that

```
num=100; inc=1/num
period=up(dot)-low(dot)+inc
x=sin(2*pi*dot/period)
```

creates a single period of the sinusoid. Plot the signal using the command

```
plot x
```

Plotting signals is discussed in Section 13. ■

9.2. Exercise. Check that

```
num=20; inc=1
z=rect((dot-2)/4)
```

generates a rectangular pulse starting at time 0 and ending at time 3. ■

9.3. Exercise. Generate a step of size 10 starting at time 2 on the same time axis as in 9.2. ■

The following unary operations on signals are not pointwise:

`int(z)`	integral,
`der(z)`	derivative,
`sum(x)`	sum (synonymous with int),
`dif(z)`	difference (synonymous with der),
`del(z)`	delay by one increment,
`inv(z)`	inverse (z real only),
`sort(z)`	sort,
`hist(z)`	histogram (z real only).

These operations are explained in what follows.

If z is a signal, then `x=int(z)` is another signal representing the running integral of z, while `y=der(z)` is its derivative. The signal `int(z)` is obtained by simple Euler integration and `der(z)` by taking differences and dividing by the increment. The signals `sum(z)` and `dif(z)` represent the corresponding operations for discrete-time signals but actually are synonyms for `int` and `der`. The signal `x=del(z)` is defined on the same axis as z and given by $x(t) = z(t - iz)$ with t ranging over the domain of z and `iz` the increment of z.

9.4. Exercise. Let z be the rectangular pulse of Exercise 9.2 and look at the signals `int(z)`, `der(z)`, and `del(z)`. ■

Suppose that z is a real, monotonically increasing or decreasing signal. Then the command `x=inv(z)` creates a signal x that is the inverse of z taken as a map from its domain to its range. The domain of x is `[min(z), max(z)]` (see below for the definition of max and min), while the number of increments of x equals that of z. If z is not monotonically increasing or decreasing, then it is truncated to the initial monotonically increasing or decreasing part. The inverse x of z is computed on a uniformly sampled signal axis, and its values are obtained by linear interpolation.

9.5. Exercise. Compute `x=inv(dot)` and verify the result. Also, compute `z=exp(0:0.1:1)` and `x=inv(z)`. Define `e=exp(1)` and compare x with the signal `y=log(1:(e-1)/10:e)`. Explain. ■

The command `sort(z)` returns a signal on the same axis as z but whose signal values are rearranged in order of increasing magnitude. Duplicate signal values are retained. If z is complex, then the real and imaginary parts are sorted separately.

If z is a real signal, then the command `h=hist(z)` generates a histogram of z. The number of sample points of the signal h according to a well-known rule of thumb is chosen as $10 \log_{10}(\text{num})$, rounded to the nearest integer. Its domain is `[min(z), max(z)]` (see below), and $h(t)$ *is* the number of signal values of z in the interval $[t - ih/2, t + ih/2)$, with `ih` the increment of h.

9.6. Exercise. Generate the signal x of Exercise 9.1. Compute and plot `sort(x)` and `hist(x)`. ■

The following functionals may be used to determine certain signal characteristics:

`mean(z)`	airthmetic mean,
`rms(z)`	root mean square value,
`ampl(z)`	maximum of the magnitude of z,
`max(z)`	maxima of real and imaginary parts of z,
`argmax(z)`	values at which real and imaginary parts of z take their first maximum,
`min(z)`	minima of real and imaginary parts of z,
`argmin(z)`	values at which real and imaginary parts of z

	take their first minimum,
`zero(z)`	values at which real and imaginary parts of z first cross zero,
`<x,y>`	inner product of the signals x and y.

If the signal z is complex, then `mean(z)`, `max(z)`, and `min(z)` are computed as complex numbers whose real parts are the mean, maximum, and minimum of the real part of z and whose imaginary parts are the mean, maximum, or minimum of the imaginary part of z, respectively.

Similarly, the numbers `argmax(z)`, `argmin(z)`, and `zero(z)` are computed as complex numbers indicating the times or frequencies at which the real and imaginary part of z take their maximum or minimum or cross zero, respectively. If several maxima, minima, or zero crossings exist the *smallest* time or frequency is taken. The values of `argmin(z)` and `argmax(z)` are taken at sampling instants but those of `zero(z)` are obtained by linear interpolation of adjacent sampling instants.

9.7. Exercise. Generate x as in 9.1 and check whether `mean(x)`, `rms(x)`, `ampl(x)`, `max(x)`, `argmax(x)`, `min(x)`, `argmin(x)`, and `zero(x)` are as expected. Define `period` as in 9.1, and let

```
y= exp(j*2*pi*dot/period)
```

Check if `mean(y)`, `rms(y)`, `ampl(y)`, `max(y)`, `argmax(y)`, `min(y)`, `argmin(y)`, and `zero(y)` are as expected. How about ⟨x,y⟩? ∎

10. Pointwise Evaluation and Composition of Signals

Let x be an existing signal and t a real scalar. Then the command

```
x(t)
```

results in the evaluation and display of the signal x at time t. If the time t does not coincide with a sampling instant, then `x(t)` is evaluated by linear interpolation between adjacent sampling instants. Thus,

```
dotplus(1.6)
```

displays the value 1.6.

The same construction may be used in an assignment. If x is an existing signal, t a real scalar and c a scalar, then the command

```
x(t)= c
```

assigns the value c to x at time t. If t is not a sampling instant, then it is rounded to the nearest sampling instant. If x does not exist, then the command is rejected. If t is outside the domain of x then the domain is automatically expanded. Missing signal values are set equal to zero.

If x is again an existing signal and a another existing real signal, then the command

```
x(a)
```

results in the composition of the two signals as follows. The increment and the domain of the signal `x(a)` that is created are the increment and the domain of a. The successive signal values of `x(a)` are evaluated as $x(a)(t) = x(a(t))$ with t ranging over the time axis of a. If for some value of t the value of $a(t)$ does not coincide with a sampling instant of x, then the value of $x(a(t))$ is obtained by linear interpolation between adjacent sampling instants. If for some value of t the value of $a(t)$ is outside the domain of x, then $x(a(t))$ is set equal to zero.

If x is a scalar (called either by name or by value), then $x(a(t))$ is set equal to x for each t in the domain of a. Thus,

```
y=1(dot)
```

results in a signal y that has the constant value 1 on the axis of dot.

The composition operation is mainly intended to change time scales on existing signals and to shift them. Suppose that x is the rectangular pulse generated in Exercise 9.2. Then,

```
x(dot/2); x(2*dot)
```

generates two rectangular pulses that have double the length respectively half the length of the original pulse. The command

```
x(dot-8)
```

shifts the pulse so that it starts at time 8 rather than at time 0.

Another use of composition is to generate periodic signals. If x is an existing signal and P some positive real scalar, then

```
x(dotplus mod P)
```

generates a periodic repetition of the part of x defined on [0, P).

10.1. Exercise. Define x as in Exercise 9.2. Look at the signals

```
x(dot)
x(dotplus)
x(dot/2)
x(2*dot)
x(dot-8)
x(dot mod 6)
(1+j)(dot)
```

and explain what you find. ∎

11. Change of Domain of a Signal

Suppose that x is an existing signal. It is possible to change the domain of x by one of the following four commands:

```
y=x[a:b]
y=x(a:b)
y=x[a:b)
y=x(a:b)
```

Here a and b are real scalars. The commands generate a signal y that equals x but whose domain is extended or restricted to the interval delimited by a and b, which is taken (half) open or (half) closed as indicated. Thus

`(a`	sets the lower limit of the domain equal to the smallest sampling instant greater than a,
`[a`	sets the lower limit of the domain equal to the smallest sampling instant greater than or equal to a,
`b)`	sets the upper limit of the domain equal to the largest sampling instant less than b,
`b]`	sets the upper limit of the domain equal to the largest sampling instant less than or equal to b.

If the domain of y is extended beyond that of x, then the missing signal values are set equal to zero.

If in any of the four commands a or b is replaced with ., then the *current* lower or upper limit is taken. By way of example, let `x=cat(1,2!1)`. Then

```
x=x(-5:5)
```

makes x equal to `cat(0,0,0,1,2!1,0,0,0)`, but

```
x=x[-5:5]
```

results in x being equal to `cat(0,0,0,0,1,2!1,0,0,0,0)`. The command

```
x=x(.:.)
```

reduces x to `cat(2!)`, while

```
x=x[.:.]
```

leaves x unchanged.

11.1. Exercise. Letting `x=cat(1,2!1)` as above, verify that

```
z=x[low(dot):up(dot)]
```

leads to the same result as

```
z=x(dot)
```

■

11.2. Exercise. Generate and plot the signal x that is given by

$$x(t) = \begin{cases} 1 & \text{for } t < 0, \\ \cos(2\pi t/P) & \text{for } t \geq 0, \end{cases}$$

on the default time axis dot with inc = 1, num = 20 and $P = 10$. ■

12. Creating and Manipulating Frequency Signals

Frequency signals are only distinguished from time signals by their default increment, which equals INC. Frequency signals may be created directly using the CAT function. This function works precisely the same way as the cat function except that it assigns INC to the increment of the signals it creates.

Frequency signals may be generated indirectly by using the standard frequency signals DOT and DOTPLUS in the same way as dot and dotplus are used for time signals. DOT has increment INC (default value 1/10), domain

$$\left[-\frac{\text{num}}{2}\text{INC}, \left(\frac{\text{num}}{2} - 1\right)\text{INC}\right]$$

if num is even and

$$\left[-\frac{\text{num} - 1}{2}\text{INC}, \frac{\text{num} - 1}{2}\text{INC}\right]$$

if num is odd, and signal values specified by $\text{DOT}(f) = f$ with f ranging over all sampling frequencies in the domain. DOTPLUS has increment INC, domain

$$[0, (\text{num} - 1)\text{INC}]$$

and signal values similarly defined by $\text{DOTPLUS}(f) = f$.

The permanent frequency signal DELTA is defined on the time axis of DOT and given by $\text{DELTA}(0) = 1/\text{INC}$, $\text{DELTA}(f) = 0$ for $f \neq 0$.

All functions and operations defined for time signals also work for frequency signals.

12.1. Exercise. Create the frequency signal z defined by

$$Z(f) = \frac{1}{1 + j2\pi f}, \quad 0 \leq f < 10,$$

with increment INC = .05. Compute the magnitude and argument of z as a function of frequency. ■

13. Graphic Display of Signals

The command

```
plot x
```

with x the name of a signal or a signal expression, provides a plot of the signal. The real and imaginary parts are plotted in different frames. If the signal is real, then the frame for the imaginary part is not displayed and the whole screen is used for the real part.

When the command `plot` is invoked, the screen is changed to graphic mode and the normal text display disappears. The text display may be recalled by pressing the ⟨ESC⟩ key. Typing

```
plot
```

without argument restores the last plot, if available.

A hardcopy dump of the graphics

screen on a printer connected to the computer may be made by holding down the shift key and pressing the ⟨PRTSCR⟩ key.

Scaling of the plots is automatic, the axes are labeled by default, and the sample points of the plot are connected by straight lines. Several signals may be displayed in the same frame by a command such as

```
plot x,y,z
```

If the format status is polar, the plot command provides plots of the magnitude and argument of x rather than of the real and imaginary parts.

13.1. Exercise. Generate a single period of a sine and two periods of a cosine as

```
inc=.01; num=101
x=sin(2*pi*dotplus)
y=sin(4*pi*dotplus)
z=x+j*y
```

Display the signals by

```
plot x
⟨ESC⟩ plot y
⟨ESC⟩ plot z
⟨ESC⟩ plot x,y
```

Look at some of the plots in polar format (see 2). ■

The command

```
polarplot x
```

provides a plot (in a single frame) of the imaginary part of x versus its real part. Several plots can be made in the same frame as for the plot command.

13.2. Exercise. Compute z as in Exercise 13.1 and give the command

```
polarplot z
```

Note that actually the signal x is plotted against the signal y. ■

Curves in a plot may be marked with a number or letter. For instance,

```
plot x"x"
```

marks the plot of x at regular distances with an "x," while

```
plot x"1",y"2"
```

marks the plot of x with "1" and that of y with "2." The command

```
plot x"."
```

plots x point by point, without connecting lines. The command format

```
plot x"|"
```

finally, is meant for displaying discrete-time signals, and plots each point as a vertical bar located at the correct horizontal coordinate whose height equals the vertical coordinate.

13.3. Exercise. Display the signals of Exercise 13.1 with different markings and as discrete-time signals. ■

If a color monitor is available, curves may be plotted in color. For instance, the command

```
plot x"/r"
```

plots the signal x in red. The available colors are

/b	blue
/c	cyan
/g	green
/r	red
/y	yellow

Default is white.

Once a plot is on the screen, it may be relabeled, titled, or a grid may be drawn in. After reverting to text mode the command

```
title "Legend"
```

writes "Legend" at the top of the current plot, which may be recalled by the command `plot`. The commands

```
hlabel "legend"
vlabel "legend"
vlabelt "legend"
vlabelb "legend"
```

overwrite the default labels beneath the horizontal axis, next to the vertical axes, next to the vertical axis of the top plot and that of the bottom plot, respectively. The commands

```
text x,y,"legend"
textt x,y,"legend"
textb x,y,"legend"
```

write "legend" in both frames, the top frame and the bottom frame, respectively, at the location (x, y), where (0, 0) is the lower left corner of the frame and (1, 1) the upper right.

The command

```
grid
```

draws grid lines on the current plot. The grid is extinguished by repeating the command.

The command

```
haxis xmin,xmax
```

freezes the left-hand and right-hand side ends of the horizontal scale at xmin and xmax, respectively. This command has to be given *before* issuing a plot command, and cannot be used to change the scale of the current plot. Similarly, the commands

```
vaxis ymin,ymax
vaxist ymin,ymax
vaxisb ymin,ymax
```

freeze the scales for both plots, the top plot or the bottom plot, respectively. The commands

```
axis
haxis
vaxis
vaxist
vaxisb
```

display *all* scale settings, respectively those for the horizontal and vertical axes separately. The commands

```
set axis
set haxis
set vaxis
set vaxist
set vaxisb
```

revert all scalings to automatic, respectively those for the horizontal and vertical axes separately.

13.4. Exercise. Compute z as in Exercise 13.1 and see what the effect is of typing

```
plot z
⟨ESC⟩ haxis 0,.5
plot z
⟨ESC⟩ hlabel "time"
vlabelt "x"
vlabelb "y"
haxis -2,2
vaxis -2,2
polarplot z
```

Put the legend "Lissajoux figure" both at the top of the plot and somewhere inside the frame by using the commands title and text. Add a grid as well. ■

The command

```
plotfile ⟨file name⟩
```

finally, creates a plot file of the current plot (i.e., the last that appeared on the screen). The file name should not have an extension; SIGSYS gives it the extension .PLT. If no file name is given, the default name SIGSYS.PLT is assigned. The file may be printed from DOS by the command copy SIGSYS.PLT prn: assuming that the file name is SIGSYS.PLT.

14. Entering Polynomials

Polynomials may be entered by using the indeterminate variable sigma. Typing

```
p=1.5+(2+3*j)*sigma+sigma^2
```

defines the polynomial

$$p = 1.5 + (2 + 3j)\sigma + \sigma^2.$$

Displaying p results in something like

```
(1.5+j*0)*σ^0+
(2+j*3)*σ^1+
(1+j*0)*σ^2
```

Note that the indeterminate variable is displayed as σ, which does not exist on the keyboard. The indeterminate variable `sigma` may be redefined, for instance to the handier `s`, by typing

```
indeter=s
```

The display remains in terms of σ, however. The current name of the indeterminate variable may be displayed by simply typing

```
indeter
```

If p has been defined as a polynomial, the command

```
p[n]=c
```

with n an integer and c a real or complex scalar, sets the coefficient of the term with the nth power equal to c. If n is greater than the degree of the polynomial, the degree is automatically increased.

If the leading coefficient of a polynomial equals zero, then its degree is automatically decreased by 1. This is repeated until the leading coefficient differs from zero. Similarly, if all the coefficients of a complex polynomial have zero imaginary part, then the polynomial is converted to a real polynomial.

14.1. Exercise. Enter the polynomial $p = 1 + 2\sigma + 3\sigma^2 + 4\sigma^4 + \sigma^5$. Change the leading coefficient to 0 and see how the degree of the polynomial decreases to 4. ■

15. Operations Among Polynomials

The following operations among polynomials are defined:

`p^n`	(with n an integer) exponentiation,
`-p`	unary minus,
`p*q`	multiplication of polynomials,
`p/a`	(with a scalar) division by a scalar,
`p+q`	addition of polynomials,
`p-q`	subtraction of polynomials.

15.1. Exercise. Compute the polynomials $p_1 = (1 + \sigma)^{10}$, $p_2 = (1 - \sigma)^{10}$ and determine $p = p_1 + p_2$. ■

16. Functions and Evaluation of Polynomials

For polynomials there are three attribute functions, namely,

`degree(p)`	the degree of p,
`roots(p)`	the roots of p,
`p[n]`	(with n an integer) the coefficient of the term with power n.

The roots `roots(p)` of p are returned as a *signal* with increment 1 and domain [0, `degree(p)` − 1]. The roots are given in order of increasing magnitude.

Two further functions convert polynomials into other polynomials:

`p'`	replaces the indeterminate σ with −σ,
`conj(p)`	replaces the coefficients with their complex conjugates.

Finally, the command

```
p(a)
```

with a a scalar, returns the value of the polynomials at σ = a. When a is a signal p(a) is a signal that is obtained by pointwise substitution of the signal a.

16.1. Exercise. Enter the polynomials $p = 1 + \sigma + \sigma^2$ and $q = 1 + \sigma^4$. Compute the roots of p and q. Determine the leading coefficient of the product polynomial pq. Compute and plot the frequency signal Z defined by

$$Z(f) = \frac{1 + \sigma + \sigma^2}{1 + \sigma^4}$$

with $\sigma = j2\pi f$, where f ranges over [0, 2) with increment INC = .05. *Hint:* Set INC = .05, num = 2/INC,

```
s=j*2*pi*DOTPLUS
```

and simply compute Z=p(s)/q(s).

17. Fourier Transformation

SIGSYS provides the following Fourier transforms:

fft	Fast Fourier Transform,
ifft	inverse Fast Fourier Transform,
fc	Fourier coefficients,
fs	Fourier series,
ddft	discrete-to-discrete Fourier transform,
iddft	inverse discrete-to-discrete Fourier transform,
cdft	continuous-to-discrete Fourier transform,
icdft	inverse continuous-to-discrete Fourier transform,
dcft	discrete-to-continuous Fourier transform,
idcft	inverse discrete-to-continuous Fourier transform
ccft	continuous-to-continuous Fourier transform,
iccft	inverse continuous-to-continuous Fourier transform.

The various transforms differ only with respect to

The *domain* of the signals they operate on, and
a *scaling factor*.

All transforms are based on the Fast Fourier Transform. One consequence of this is that all signals that are transformed need be defined on a number of sample points that is an integral power of 2. The domain of signals that are defined on a different number of sample points is automatically extended; missing signal values are set equal to zero.
Two types of domain are used:

"One-sided." The domain of the signal is

$$[0, (N - 1)T],$$

where N is the number of sample points of the signal for nonnegative times, and T the increment of the signal. Signal values for negative times are discarded. If N is not an integral power of 2 it is increased to the next integral power of 2 and the signal is supplemented with zeros.
"Two-sided." The domain of the signal is

$$\left[-\frac{N}{2}T, \left(\frac{N}{2}-1\right)T\right],$$

where N is the number of sample points of the signal and T its increment. If N is not an integral power of 2 it is increased to one and the signal is supplemented with zeros.

The various transforms are based on the formula

$$\hat{x}(f) = c \sum_{t \in D} \hat{x}(t) e^{-j2\pi ft}, \qquad f \in \hat{D},$$

while the inverse transforms are given by

$$x(t) = \hat{c} \sum_{f \in \hat{D}} \hat{x}(f) e^{j2\pi ft}, \qquad t \in D.$$

D is the domain of the time-signal and $\hat{D}$ that of the transform, while c and $\hat{c}$ are constants. Given a time signal with N sample points and increment T, the increment of its transform is $F = 1/NT$. Given a transform with N sample points and increment F, the increment of the inversely transformed time signal is $T = 1/NF$. D and $\hat{D}$ both contain N points, and are one- or two-sided depending on the transformation.

For the transform pair `fft/ifft` (Fast Fourier Transform and its inverse) the constants are $c = 1$ and $\hat{c} = 1/N$, according to the usual convention. Both D and $\hat{D}$ are one-sided.

For the transform pair `fc/fs` (Fourier coefficients/Fourier series) the constants are $c = 1/N$ and $\hat{c} = 1$, so that the transform `fc` generates the Fourier coefficients (in the alternative form of 6.4.5) and the inverse transform `fs` reconstructs the time signal from the Fourier coefficients. D is one-sided and $\hat{D}$ two-sided.

For all remaining Fourier transform pairs the constants are $c = T$ and $\hat{c} = F$. The domains and coefficients of all transform pairs are indicated in the table.

We look at an example. Suppose that a time signal `z` is generated by the sequence of commands

```
num=200; inc=.01
z=rect(dot)
```

The time signal is a rectangular pulse with duration from $-.5$ to $.5$, defined on the time axis $[-1, 1)$, with increment 0.01 and 200 sample points. The command

```
Z=ccft(z)
```

first expands the number of sample points from 200 to 256, then extends the domain two-sidedly from $[-1, 1)$ to $[-1.28, 1.28)$ while padding with zeros, and finally computes the continuous-to-continuous Fourier transform (approximating the integral by a sum) on 256 points, with increment $F = 1/(256 \cdot 0.01) = 0.390625$, on the two-sided frequency domain $[-128F, 128F) = [-50, 50)$. The subsequent command

```
zz=iccft(Z)
```

ATTRIBUTES FOR THE VARIOUS FOURIER TRANSFORM PAIRS

Transform	Inverse	Name	c	$\hat{c}$	D	$\hat{D}$
`fft`	`ifft`	Fast Fourier Transform	1	$1/N$	one-sided	one-sided
`fc`	`fs`	Fourier coefficients	$1/N$	1	one-sided	two-sided
`ddft`	`iddft`	DDFT	T	F	one-sided	one-sided
`dcft`	`idcft`	DCFT	T	F	two-sided	one-sided
`cdft`	`icdft`	CDFT	T	F	one-sided	two-sided
`ccft`	`iccft`	CCFT	T	F	two-sided	two-sided

leaves the number of sample points and the domain of z unaffected, and returns zz as the inverse CCFT (approximating again the integral by a sum) on 256 points, with increment $T = 1/NF = 0.01$, on the two-sided domain $[-1.28, 1.28)$. Because the original signal z has its support entirely inside its domain, the signal zz is identical to the original signal z.

We continue the example. Given the time signal z as originally defined, the command

```
Z=fc(z)
```

discards the signal values at negative times, changes the number of sample points from 100 to 128 (the next integral power of 2 up from 100), redefines the domain of the signal to $[0, 1.28)$ and supplements the signal with zeros. The command considers the resulting signal as a single period of a periodic signal and computes its Fourier coefficients. The coefficients are stored as a frequency signal with 128 sample points, frequency increment $F = 1/(128 \cdot 0.01) = 0.78125$ and domain $[-64F, 64F) = [-50, 50)$. The command

```
zz=fs(Z)
```

sums the Fourier series whose coefficients are stored in Z on 128 sample points, with time increment $T = 1/(128 \cdot 0.78125) = 0.01$ on the one-sided domain $[0, 1.28)$. The signal zz equals the original signal z only for nonnegative times.

17.1. Exercise. Go through the steps indicated above and verify the results. ∎

18. Macros and Shell Escape

SIGSYS may handle macros. Macros should be prepared as ASCII files, and given the extension .MAS. Suppose that MACRO1.MAS is a macro file name. When the command

```
macro1
```

is given (without the extension, and no distinction is made between upper and lower case characters), SIGSYS searches the disk for a file of that name with the extension .MAS, reads it line by line, and presents each line to the interpreter as a command line. A macro can call other macros.

Macros are searched for in the default directory, or the directory set externally or by the `chdir` command as explained in Section 3.

Macros may be internally documented using the % character. Any text on a command line following the character % up to the end of the line or the first separator ; is ignored, including the % character itself. The % may be preceded by any number of spaces.

By way of example, a macro that shows a Lissajoux figure follows.

```
%Macro DEMO
%Plot of a Lissajoux figure
num=200; inc=1/num
%Add 1 point
%to obtain a complete figure:
num=num+1
x=sin(2*pi*dotplus*4)
y=sin(2*pi*dotplus*5)
z=x+j*y
polarplot z
```

The command

```
pause
```

may be inserted in a macro file. When this command is encountered, execution stops, and resumes when the ⟨ENTER⟩ key is pressed.

A useful facility in creating and modifying macros is the *shell escape*

```
!
```

A single ! on a command line shifts control to the "shell," in this case DOS, while keeping SIGSYS resident. In this mode, an editor may be invoked to create or modify a macro. After finishing with editing, typing

```
exit
```

(from DOS) returns control to SIGSYS. The escape ! may also be followed directly on the command line by a DOS command. Thus,

```
!edit macro1.mas
```

invokes an editor called "edit" to work on the file MACRO1.MAS (note that DOS automatically converts lower case letters to capitals.) After exiting from the editor control is automatically returned to SIGSYS. As another example, the command

```
!dir
```

displays the contents of the current directory.

Note that because SIGSYS stays resident, the memory available for any program invoked via the shell escape is much less than it would normally be.

18.1. Exercise. Create the macro file DEMO1.MAS that will display a Lissajoux figure, and run it. ∎

19. Integration of Differential Equations

SIGSYS offers a unique facility for the integration of a set of differential equations. Consider for instance solving the Volterra equation

$$\dot{x} = x(y - 1),$$

$$\dot{y} = y(1 - x),$$

on the interval [0, 10] with an integration step size of .04 and initial conditions $x(0) = 2$, $y(0) = 2$. This is accomplished by the following sequence of commands, which are best collected and executed as a macro.

```
x=2; y=2
integrate from 0 to 10 stepsize .04
@x=x*(y-1)
@y=y*(1-x)
write x,y
end
```

Before entering the integration loop the initial conditions need be set. This is done in the first line. Inside the integration loop the notation @ may be read as the differential operator. For the interpreter the symbol @ marks the variables that need be integrated.

The `write` command indicates the signals whose values have to be retained during integration and stored upon completion. After completion of the command, x and y are signals with domain [0, 10] and increment .04 containing the solution of the differential equation.

By default, integration is performed with the standard second-order Runge-Kutta method. Modification of the `integrate` command to

```
integrate RKx ...
```

with $x = 1$, $x = 2$ or $x = 4$ results in the use of the standard first-, second-, or fourth-order Runge-Kutta method, while

```
integrate AB ...
```

uses the second-order Adams-Bashforth method, initialized with a single second-order Runge-Kutta step.

If `stepsize` is not specified, the current value of `inc` is used as a default for the step size. If neither the integration range nor the step size are specified, integration is performed on the default time axis `dotplus`. Thus, the second command line may for instance be modified to

```
integrate from 0 to 10
```

or to

```
integrate AB
```

In the example, upon exit of the integration loop x and y, which are initially defined as scalars, are overwritten by the signals included in the `write` command. If this command is omitted, then x and y on exit have scalar values equal to their values at the final time.

The integration loop may also involve auxiliary variables and externally

defined signals. For instance, the differential equation

$$\dot{x} = -x + u,$$

with u an existing time signal, may be integrated on the time axis dotplus as

```
x=0; t=0
integrate
@t=1
@x=-x+u(t)
write x
end
```

By integrating t along with x, each time the right-hand side of the differential equation for x is called the external input u is evaluated at the correct time instant.

Finally, consider the integration of the differential equation

$$\frac{dw}{dt} = \alpha(u - w^2).$$

where u is given by

$$u = k(w_o - w).$$

Here w might represent the normalized speed of a car, and u the throttle position, which follows by proportional feedback of the difference $w_o - w$ between a constant reference speed w_o and the actual speed w. We integrate this as follows:

```
alpha=.1; k=1; wo=0.9
w=0.2
integrate from 0 to 100 stepsize 1
u=k*(wo-w)
@w=alpha*(u-w*w)
write u,w
end
```

Note that the command u=k*(wo-w) must precede the integration command to ensure that @w is evaluated for the correct value of u. Upon completion of the command, the solutions for both u and w are stored over the entire time interval.

19.1. Exercise. Try the three examples presented in this section. ■

The integration loop may prematurely be terminated with the break command. If we modify the last example to

```
alpha=.1; k=.1; wo=.9; w=.2
integrate from 0 to 100 stepsize 1
u=k*(wo-w)
@w=alpha*(u-w*w)
if w>.55 break end
write u,w
end
```

integration is stopped as soon as the speed w exceeds the value 0.55.

20. Iteration

Sets of iterative equations may conveniently be solved by using the iterate command. Suppose by way of example that we wish to solve the simultaneous difference equations

$$x_1(n+1) = x_2(n),$$

$$x_2(n+1) = x_1(n) + x_2(n),$$

for $n = 0, 1, 2, \cdots, 20$, with the initial conditions $x_1(0) = 0$, $x_2(0) = 1$. This may be done as follows in a way that is similar to integration:

```
x1=0; x2=1
iterate from 0 to 20
@x1=x2
@x2=x1+x2
write x1,x2
end
```

The iteration loop is repeated as many times as indicated in the from ··· to statement. Each time the loop is executed the variables marked with @ are temporarily stored and given their definitive values *after* the loop has been completed once. The values of the variables specified in the write statement are retained during the iteration process and stored as signals on the

axis defined by the `from ··· to` statement with increment one. If the `from ··· to` part is omitted, iteration is performed on the axis 0, 1, ··· , num.

Like integration, iteration may be interrupted by using the `break` command. Modification of the example to

```
num=20
x1=0; x2=1
iterate
@x1=x2
@x2=x1+x2
if x1>1000 break end
write x1,x2
end
```

causes the iteration process to stop as soon as `x1` exceeds the value 1000.

20.1. Exercise. See how this iteration works. ∎

21. Command Flow Control

SIGSYS has several facilities for command flow control. They are mainly useful within macros. The first is the `if` command, which takes the form

```
if ⟨condition⟩
⟨compound statement⟩
end
```

The four basic elements of this statement may be separated by spaces, commas, or newlines. An example is

```
if real(x)>0 x=0 end
```

In ⟨condition⟩ the following comparators may be used:

==	equal to,
>	greater than,
>=	greater than or equal to,
<	less than,
<=	less than or equal to,
~=	not equal to

Furthermore, the logical operators

&	"and,"
|	"or," and
~	"not"

are permitted.

The ⟨compound statement⟩ may encompass several statements. An example is

```
if x>0 & y~=0
x=1; y=0
end
```

The `if` statement may include an `else` option, in the form

```
if ⟨condition⟩
⟨compound statement⟩
else
⟨compound statement⟩
end
```

By way of example consider

```
if x>=0 x=1
else x=-1
end
```

More flow control is possible with a `while` loop. It has the general form

```
while ⟨condition⟩
⟨compound statement⟩
end
```

The ⟨compound statement⟩ is repeatedly executed until ⟨condition⟩ becomes false. Thus, in

```
t=0
while t<10
t=t+1
end
```

the value of `t` is increased from 0 to 10. `While` loops may be interrupted with the `break` command. In

```
t=0; x=1
while t<10
t=t+1; x=2*x
if x>1000 break end
end
```

the loop is broken when x becomes larger than 1000.

Finally, a `for` loop may be executed. It has the general form

```
for ⟨scalar variable⟩
in ⟨signal variable⟩
⟨compound statement⟩
end
```

In this command, the ⟨compound statement⟩ is repeated as many times as ⟨signal variable⟩ has sample points, with ⟨scalar variable⟩ successively taking the values assumed by ⟨signal variable⟩. For instance,

```
for t in dot
z(t)=x(t)+x(t)
end
```

is a way of adding the time signals x and y on the time axis `dot`. A more useful example is

```
x=0(0:1:100)
for t in 0:10:100
x(t)=1
end
```

which creates x as a comb on the signal axis 0:1:100 with period 10. As a final example consider

```
x=0
for v in x
X=X+abs(v)
end
```

The final value of x is the sum of the absolute values taken by the signal x.

Also a `for` loop may be interrupted with the `break` command.

21.1 Exercise. Test the various examples in this section. ∎

22. User-Defined Functions and Procedures

SIGSYS allows the user to define functions. The general form is

```
function ⟨name⟩ ((a. list))
⟨compound statement⟩
return ⟨expression⟩
end
```

where "a. list" stands for "argument list." We look at an example. The series of commands

```
function crt(x)
return x^(1/3)
end
```

defines a function similar to `sqrt` that may be used to compute cube roots of scalars and signals. Calling `crt(8)` returns the value 2. Once defined, a function may be called as often as desired. Another example is

```
function Max(x,y)
if real(x)>=real(y)
 r=real(x)
 else r=real(y) end
if imag(x)>=imag(y)
 i=imag(x)
 else i=imag(y) end
return r+j*i
end
```

The function `Max` computes the maximum of the real and imaginary parts of two scalars or signals.

All variables within a function definition are local. This means that externally defined variables are not available.

Besides functions, also *procedures* may be defined. The general form is

```
procedure ⟨name⟩ ((a. list))
⟨compound statement⟩
end
```

where again "a. list" stands for "argument list." The arguments in the list may be input or output parameters. For instance, after defining

```
procedure swap(x,y)
u=x; x=y; y=u
end
```

the command

```
swap(a,b)
```

causes the values of a and b to be interchanged.

22.1 Exercise. Define and test the functions crt and Max and the procedure swap. ■

23. Export and Import of Signals

Signals may be exported as ASCII files using the commands

```
export ⟨s. name⟩ as ⟨f. name⟩
```

where "s. name" stands for "signal name" and "f. name" for "file name." The file name should have no extension: SIGSYS gives it the extension .DAT. If

```
as ⟨f. name⟩
```

is omitted, the file name is the same as the signal name. The file is written to the current directory (see 3.)

SIGSYS writes the signal data in the file in three column format, with the first column the signal axis, the second column the real part, and the third column the imaginary part of the signal values. Thus, part of a .DAT file could look like

0	1	1
0.1	1.5	0.5
0.2	3	1
0.3	4.5	1
0.4	6	1.5

If the signal is real, the third column is omitted.

Exported .DAT files may be used as input to other programs, such as text processing and graphical programs. The converse of the export command is the command

```
import ⟨f. name⟩ as ⟨s. name⟩
```

with "f. name" standing for "file name", and "s. name" for "signal name." No extension should be specified; SIGSYS looks for a file with the extension .DAT. If

```
as ⟨s. name⟩
```

is omitted, the signal name is the file name. The import command converts a two or three column ASCII file as created by the export command or otherwise to a real or complex signal.

The export and import commands may be modified to

lexport	export in list format
limport	import from list format

In files in list format the first column (i.e., the signal axis), is omitted. Imported files in list format are defined as signals on a one-sided time axis with the default increment inc.

24. Help and Diary Facilities and Diagnostics

SIGSYS offers an on-line help facility. Typing

```
help
```

displays a list of all available commands, one of which may be selected by using the cursor keys. By pressing the ⟨RETURN⟩ key, information concerning the command is shown on the left-half part of the screen. Additional related items may be selected with the cursor and displayed on the right-half part of the screen by again using the ⟨RETURN⟩ key. The help command may be terminated by using the ⟨ESC⟩ key. Help may alternatively be invoked, even in the middle of typing a command line, by using the

```
⟨ALT⟩H
```

combination (i.e., by using the H key while pressing the ⟨ALT⟩ key).

The diary facility makes it possible to keep a record of all text that appears on the screen. Typing

```
diary (file name)
```

opens a file with the specified name, and all text that subsequently appears on the screen is written to the file. Plots are not recorded. The file name should be given without extension; SIGSYS gives the file the extension .LOG. If (file name) is omitted, then the default name SIGSYS.LOG is used. The command

```
diary off
```

temporarily suspends recording the screen text. Recording may be resumed by typing

```
diary on
```

The command

```
diary close
```

closes the file.

If during the compilation of a SIGSYS command line an error is detected, an arrow appears below the command line, approximately indicating the location of the error in the command line. Moreover, a concise error message is shown.

25. Synopsis

Syntax element	Meaning	Section
→, ←, ↑, ↓	command line editing	3
!	origin mark for concatenation	6
!	shell escape	18
!BREAK OFF	turn off DOS abort feature	3
!BREAK ON	turn on DOS abort feature	3
&	logical "and"	21
%	comment follows	18
'	mirror polynomial	16
( )	precedence	4
(:)	change domain	11
(:]	change domain	11
*	multiply	4, 8, 17
**	fast convolution	8
.	direct convolution	8
+	add	4, 8, 15
−	unary minus, subtract	4, 8, 15
/	divide	4, 8, 15
::	define signal axis	6
;	command separator	2
<	less than	21
< , >	inner product	9
<ALT> H	invoke the help utility	24
<BACKSPACE>	delete preceding character	3
<CTRL> C	cancel current operation	3
<CTRL> Q	resume execution	3
<CTRL> S	halt execution	3
<DEL>	delete current character	3
<INS>	toggle insert mode	3
<PRTSCR>	print screen	13
<=	less than or equal to	21
=	assigment operator	2
==	equal to	21

Syntax element	Meaning	Section
>	greater than	21
>=	greater than or equal to	21
@	differential operator, store temporarily	19, 20
abs	absolute value	5
acos	arccosine	5
ampl	amplitude	9
ans	value of last unassigned variable	4
arg	argument	5
argmax	argument of maximum	9
argmin	argument of minimum	9
asin	arcsine	5
atan	arctangent	5
axis	return settings for plot scales	13
BREAK OFF	turn off DOS abort feature (external command)	3
BREAK ON	turn on DOS abort feature (external command)	3
break	break a loop	19, 20, 21
CAT	concatenate frequency signals	12
cat	concatenate time signals	6
cc	fast cyclical convolution	8
c.c	direct cyclical convolution	8
ccft	CCFT	17
cdft	CDFT	17
ceil	round up	5
chdir	change directory	3, 18
clear	clear all or some variables	3
conj	complex conjugate	5, 16
cos	cosine	5
dcft	DCFT	17
ddft	DDFT	17
degree	degree of polynomial	16
del	delay by one increment	9
DELTA	unit frequency impulse	12
delta	unit time impulse	6
der	derivative	9
diary	start a diary	24
diary close	close the diary	24
diary off	suspend writing to the diary	24
diary on	resume writing to the diary	24
dif	difference	9
DOT	permanent frequency signal	12
dot	permanent time signal	6
DOTPLUS	permanent frequency signal	12
dotplus	permanent time signal	6
else	else	21
end	end of statement	19, 20, 21, 22
exit	terminate SIGSYS, exit from DOS shell	3, 18
exp	exponential	5
export	export signal	23
fft	Fast Fourier Transform	17
floor	round down	5
for	for loop	21
format	revert to default formats	2

Syntax element	Meaning	Section
format cartesian	Cartesian format	2
format long	long precision	2
format polar	polar format	2
format short	short precision	2
function	define function	22
grid	insert grid	13
haxis	freeze or show horizontal scale	13
help	help utility	24
hist	histogram	9
hlabel	label horizontal axis	13
iccft	inverse CCFT	17
icdft	inverse CDFT	17
idcft	inverse DCFT	17
iddft	inverse DDFT	17
if	if	21
ifft	inverse FFT	17
imag	imaginary part	5
import	import signal file	23
INC	default increment frequency signal	7, 12
inc	default increment time signal	6
indeter	redefine or display indeterminate variable	14
integrate	integrate differential equation	19
int	integrated signal	9
inv	inverse	9
iterate	iterate	20
j	$\sqrt{-1}$	2
lexport	export in list format	23
limport	import in list format	23
load	load a data file	3
log	natural logarithm	5
log10	logarithm to the base 10	5
low	lower limit of domain	5
max	maximum value	9
mean	mean value	9
min	minimum value	9
mod	reduction modulo	4, 8
noise	white noise	6
noiseplus	white noise	6
num	number of sample points	6
o.o	direct correlation	8
oo	fast correlation	8
pack	reorganize memory	3
pause	pause	18
pi	π	2
plot	plot a signal	13
plotfile	create a plot file	13
polarplot	polar plot of a signal	13
procedure	define procedure	22
ramp	ramp function	5
rand N	change distribution to normal	2
rand seed	set seed random number generator	2
rand U	change distribution to uniform	2

Syntax element	Meaning	Section
rand	random number	2
real	real part	5
rect	rectangular pulse	5
return	return function value	22
rms	root mean square value	9
roots	roots of polynomial	16
round	round to nearest integer	5
sat	saturation function	5
save	save all data	3
SET SIGSYS=	set default directory (external command)	3, 18, 23
set axis	unfreeze all scalings	13
set haxis	unfreeze horizontal scaling	13
set INC	set increment frequency signal	7
set inc	set increment time signal	7
set num	set number of sample points	7
set vaxisb	unfreeze vertical scaling of the bottom plot	13
set vaxist	unfreeze vertical scaling of the top plot	13
set vaxis	unfreeze vertical scaling	13
sigma	default indeterminate variable	14
sign	signum function	5
sin	sine	5
sinc	sinc function	5
sort	sort function	9
sqrt	square root	5
step	step function	5
stepsize	integration step size	20
sum	summed equal	9
tan	tangent	5
text	insert text in plot	13
textb	insert text in bottom plot	13
textt	insert text in top plot	13
title	display title	13
trian	triangular function	5
up	upper limit of domain	6
vaxisb	freeze or show vertical axis of the bottom plot	13
vaxist	freeze or show vertical axis of the top plot	13
vlabelb	label vertical axis of the bottom plot	13
vlabelt	label vertical axis of the top plot	13
vaxis	freeze or show vertical scales	13
vlabel	label vertical axes	13
while	while	21
who	display variables and free memory space	3
write	store signal in integration or iteration loop	19, 20
zero	zero crossing	9
[:)	change domain	11
[:]	change domain	11
[ ]	extract a coefficient of a polynomial	16
^	exponentiation	4, 8, 15
|	logical "or"	21
~=	not equal to	21

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่ง ของผู้ช่วยศาสตราจารย์อภินันท์ มัณยานนท์ อาจารย์ที่ปรึกษาปริญญานิพนธ์ ซึ่งท่านได้ให้คำแนะนำและข้อคิดเห็นต่าง ๆ ของการทำปริญญานิพนธ์มาด้วยดีตลอด รวมทั้งเพื่อนๆ ทุกคนที่คอยให้กำลังใจตลอดมา จึงขอขอบพระคุณมา ณ ที่นี้ด้วย

ท้ายนี้ ผู้ทำปริญญานิพนธ์ใคร่ขอกราบขอบพระคุณ บิดา-มารดา ซึ่งสนับสนุนในด้านการเงิน และให้กำลังใจแก่ผู้ทำปริญญานิพนธ์เสมอมาจนสำเร็จการศึกษา

ณรินทร์ชัย เพียรสุขมณี
วีรยุทธ ยังคล้าย
สุทัศน์ ว่องไว

เอกสารอ้างอิง

[1] ดร.ไพรัช ธัชยพงษ์ "การประมวลสัญญาณดิจิตอล" ตอนการออกแบบวงจรกรองดิจิตอล
: สำนักพิมพ์ บริษัท เอดิสัน เพรส โปรดักส์ จำกัด,กันยายน 2535

[2] นิรันดร์ คำประเสริฐ "คณิตศาสตร์วิศวกรรมไฟฟ้า 1"
: สำนักพิมพ์ สถาบันเทคโนโลยีพระจอมเกล้า ธนบุรี

[3] ผศ.ดร.เอก ไชยสวัสดิ์ "การโพรเซส สัญญาณ ดิจิตอล"
: สำนักพิมพ์ สถาบันเทคโนโลยีพระจอมเกล้า ธนบุรี

[4] ธันวา ศรีประโมง "การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม",2537

[5] Roman Kuc "Introduction to DIGITAL SIGNAL PROCESSING"
: McGRAW-HILL INTERNATIONAL EDITIONS,1982

[6] Paul M.Embree,Bruce Kimble "C LANGUAGE ALGORITHMS FOR DIGITAL SIGNAL PROCESSING"
: Prentice-Hall International Editions,1991

[7] Huibert Kwakernaak,Raphael Sivan "MODERN SIGNALS AND SYSTEMS"
: Prentice-Hall Internationals Editions,1991

[8] Poularikas,Seely "SIGNALS AND SYSTEMS"
:PWS-KENT Publishing Company,1991

[9] Lonnie C. Ludeman "FUNDAMENTAL OF DIGITAL SIGNAL PROCESSING"
:Harper & Row,Publishers

[10] Creative Labs "THE DEVELOPER KIT" for sound blaster series
:Creative Labs,1991

[11] Herbert Schildt "TURBO C/C++: The Complete Reference,Second Edition"
:McGRAW HILL INTERNATIONAL EDITIONS,1992

[12] Al Stevens "TURBO C: Memory Resident Utilities,Screen I/O and Programming Techniques"
:TECH PUBLICATION

[13] C.Britton Rorabaugh "Digital Filter Designer's Handbook"
:McGRAW HILL,1993