1988

Computer aided design of linear phase finite impulse response digital filters

Wentao Lyou
University of Central Florida

COMPUTER AIDED DESIGN OF LINEAR PHASE FINITE IMPULSE RESPONSE DIGITAL FILTERS

BY
WENTAO LYOU
B.S.E.E., Feng Chia University, 1984

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Summer Term
1988

ABSTRACT

This research report is an investigation of techniques for designing linear phase finite-duration impulse response (FIR) filters. The software package, CADFIR, developed for this project can be used to design linear phase FIR digital filters via three well-known design techniques: the window method, the frequency sampling method, and the optimal filter design method. CADFIR has options for designing such standard filters as lowpass, highpass, bandpass, and bandstop filters as well as multipassband-stopband filters.

The fundamental properties of FIR filters are discussed and design algorithms associated with the three FIR filter design techniques are developed. Design examples are included and the procedure for using CADFIR is demonstrated.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

The study of signals is encountered in almost every field of science and engineering, e.g., in acoustics, communication, control systems, electronics and physics. Signals can be classified into two classes, namely continuous-time signal and discrete-time signal. Both types of signals can be represented by a unique function of frequency referred to as the frequency spectrum of that signal. This frequency spectrum provides a description of the frequency content of the signal. Filtering is a process used to modify or manipulate the frequency spectrum of an original signal in order to meet some desirable criteria. There are two general motivations for filtering. One is to improve the quality of the original signal; the other is to process or extract information from the original signal.

A digital filter is a digital system that is used for filtering discrete-time signals. The terms IIR (infinite-duration impulse response) and FIR (finite-duration impulse response) define the two types of digital filters. One class of digital filters that is of primary interest involves those filters that possess exactly linear phase. This report restricts itself to the design and analysis of

this class of digital filters, i.e., linear phase FIR filters. Before discussing the concepts of FIR filters, a system overview of the CADFIR package is presented in this chapter.

CADFIR Overview

CADFIR is a collection of FIR filter analysis and design software routines for execution on an IBM-PC or a truly compatible machine; it includes the following files:

CADFIR.EXE: CADFIR package main program

WINDOW.EXE: Windowed filter design routine

FRESAMP.EXE: Frequency sampling filter design routine

OPTIMAL.EXE: Optimal filter design routine

PLOT.EXE: Graph display routine

Hardware and software requirements for using CADFIR are listed in Appendix A.

To start CADFIR, you have to log on to the disk drive (A hard disk drive is recommended because CADFIR requires a lot of file assessments) or directory that contains the above files. Then type "CADFIR" at the DOS (Disk Operating System) prompt and press the ENTER key. When CADFIR is invoked, it first displays its title. After a key being pressed, the main screen window (see Figure 1) appears. Look closely at the main screen window; it consists of three parts: the "main menu," the "view port" and the "guide line." The "main menu" contains the commands

of CADFIR and a cursor bar to highlight one of the menu options. You can use the LEFT and RIGHT arrow keys to move the cursor bar to the desired option and then select it with the ENTER key. The "view port" is used to display other information after the CADFIR commands are invoked. If you just start to use CADFIR, the "guide line" is very helpful; it shows what you should and/or can do at each step of the design procedure.

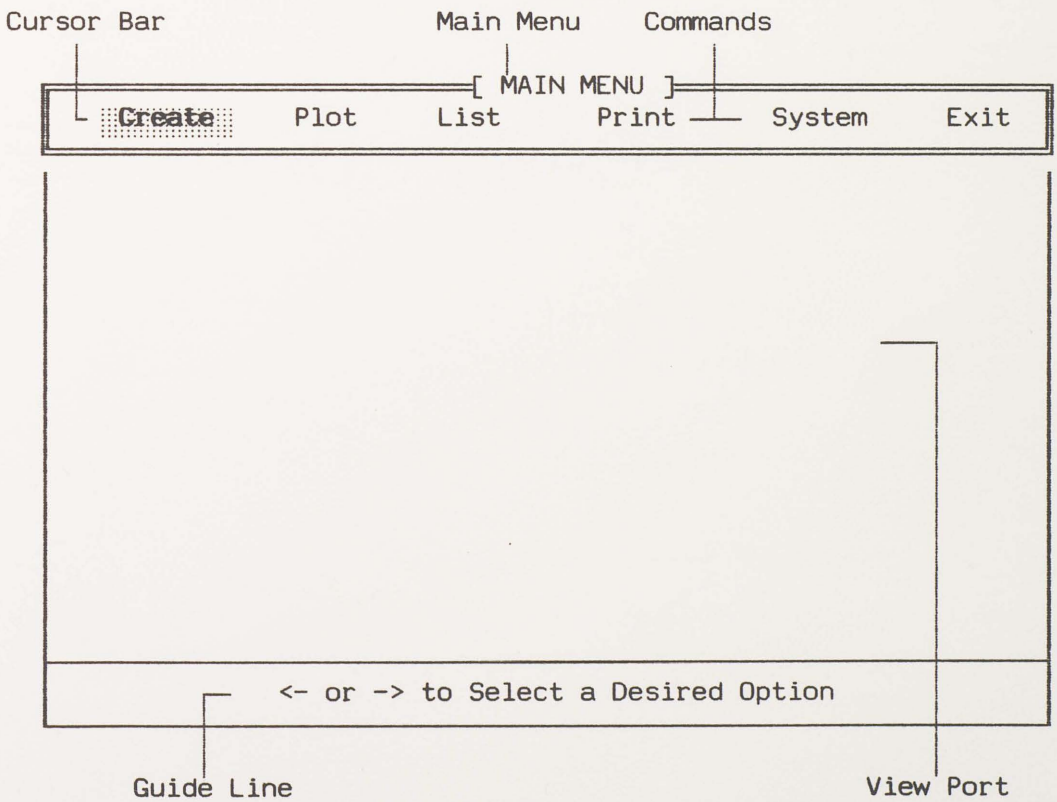


Figure 1. Main Screen Window of CADFIR.

CADFIR Menu Commands

The main menu contains the major commands that you will use to create new designs and review old designs. The six options include Create, Plot, List, Print, System and Exit. All of these commands are described as follows:

Command Create

Function Create a new filter design.

Description When you select Create, you will see some submenus and a data entry window sequentially displayed in the view port. You may use the UP and DOWN arrow keys to select one of the submenu options when that submenu is active (i.e., a cursor bar appears) and you can enter the filter specifications when the data entry window is active. Figure 2 shows an example of the submenus and data entry window for designing a FIR filter via the window method. After all the filter specifications have been entered, CADFIR will ask for a name for the present design. Because CADFIR will create data files after the design phase is complete, it needs names for those files. The design name you entered is used as the prefix of those file names. The data files follow the convention shown on the next page in terms of

their assigned extensions and definitions; the extensions designate the types of the files,

DesignName.IMP: Impulse Response Coefficients,

DesignName.FRQ: Frequency Response Coefficients.

As you see a message "Working" appears, the design is in processing. The design phase is not finished until the main menu is active.

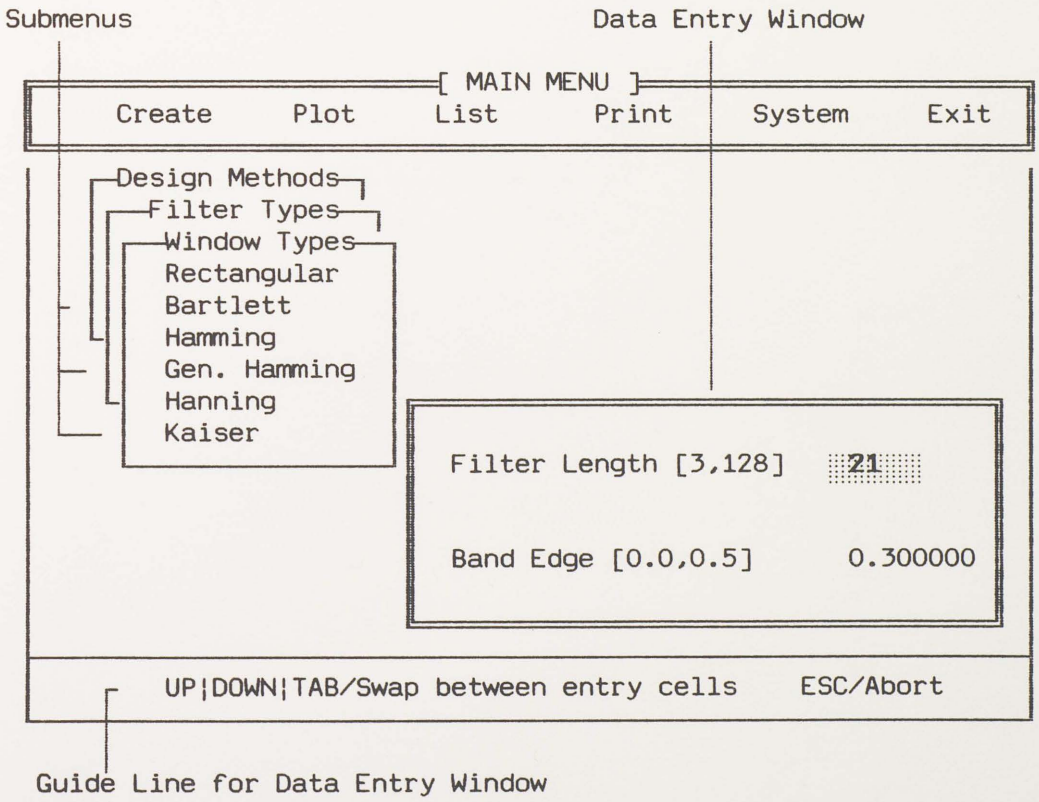


Figure 2. An Example of the CADFIR Submenus and Data Entry Window.

Command Plot

Function Plot the impulse or frequency response of an existing design on the screen.

Description When you select Plot, two submenus (shown in Figure 3) will appear. From the first submenu, you can choose a type of graph (e.g., impulse response, magnitude of frequency response or phase of frequency response). At the second submenu a list of existing data files, associated with the type you selected, is displayed to let you pick out the data file which you intent to plot.

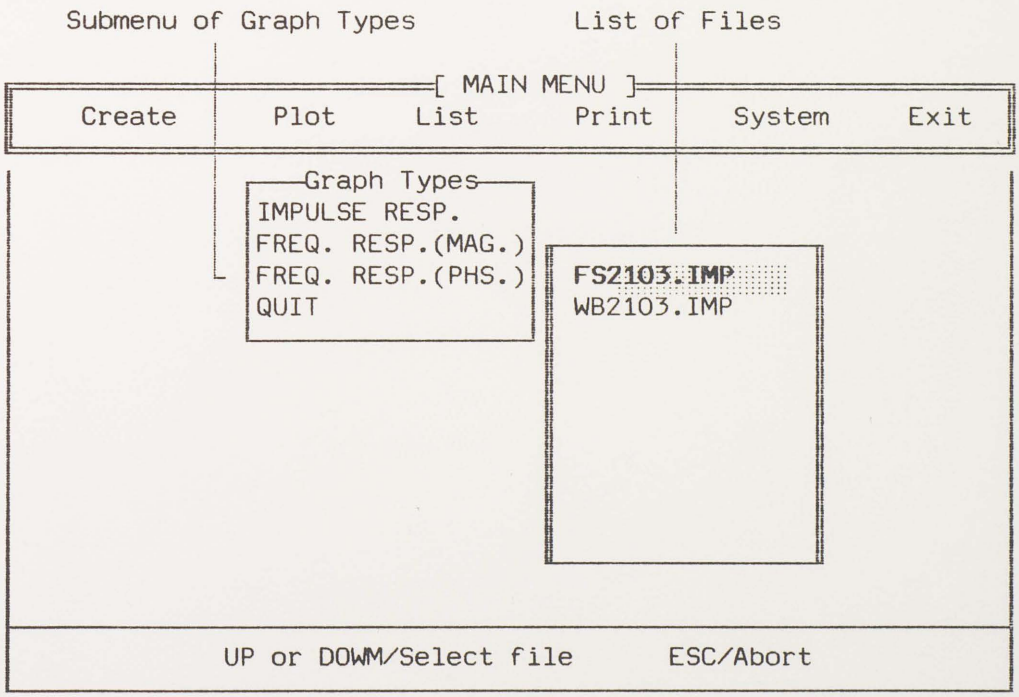


Figure 3. Submenus of Plot Command.

Command List

Function List a data file on the screen.

Description After you select List, two more options should be made from two submenus, that is, you have to select a file type and then a file name to specify the data file to be listed. Figure 4 shows an example of these submenus; in this case the file type is already selected to be the frequency response data file.

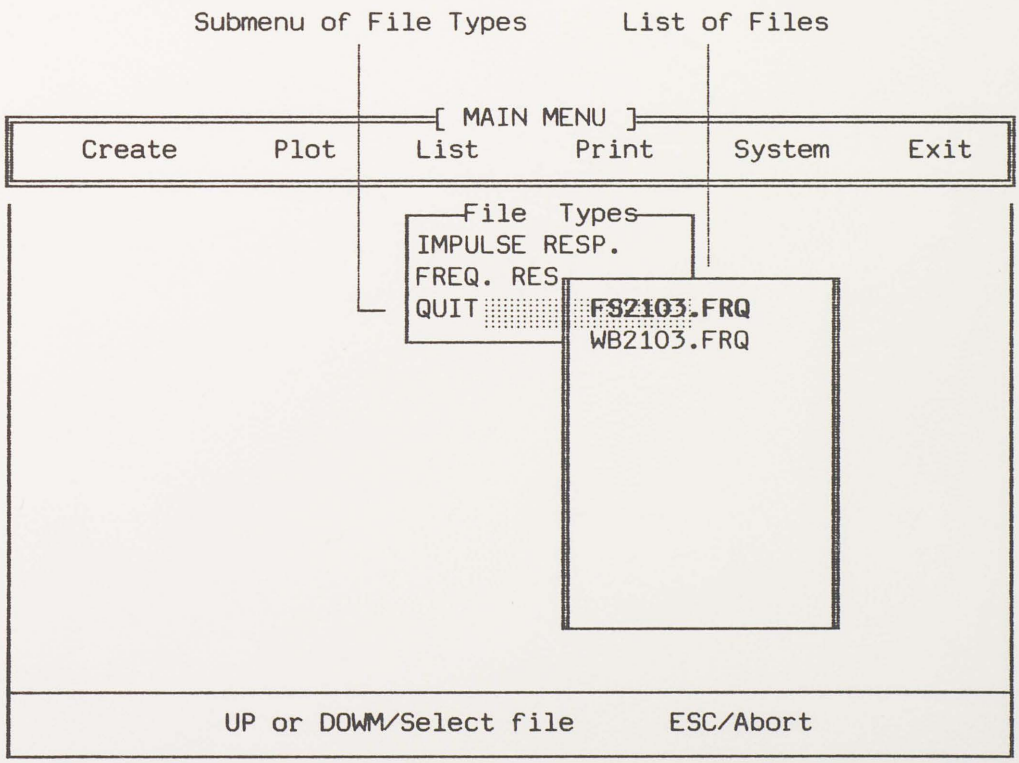


Figure 4. Submenus of List Command.

Command Print

Function Print a data file.

Description Same procedure as the List command, two options are required to specify the file you want to print. If the printer is not ready for some reason (e.g., power off, not on line, out of paper, etc.), CADFIR will prompt you to get the printer ready then retry again.

Command System

Function Invoke DOS.

Description This option lets you leave CADFIR temporarily and you will back to the DOS prompt. To return to CADFIR, you just type "exit" at the DOS prompt and then press the ENTER key. Yet, this command can be executed only if the DOS system file COMMAND.COM is in the current working directory or in those directories specified by the DOS PATH environment.

Command Exit

Function Exit back to DOS system.

Description To avoid unintentionally selecting this option, CADFIR will ask you to confirm that you really wish to exit. You may press "Y" or "y" to exit or press "N" or "n" to stay.

In the above discussions some knowledge about DOS may be helpful but not critical. What makes CADFIR unique is its ease of use. If you have not used an IBM-PC before or have no idea about DOS, the only important step you have to know is how to start CADFIR.

CHAPTER II

PROPERTIES OF FINITE-DURATION IMPULSE RESPONSE (FIR) FILTERS

Digital filters with a finite-duration impulse response possess certain properties from the point of view of filter design. For example, the questions of stability and realization never arise since FIR filters are always stable and, with an appropriate finite delay, can always be made realizable. Furthermore, as shown later in this chapter, FIR filters can be designed so that their frequency responses have an exactly linear phase.

The duration or sequence length of the impulse response of FIR filters is, by definition, finite; therefore, the output can be written as a finite convolution sum (Oppenheim and Schafer 1975).

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \tag{2.1}$$

With a change of index variables,

$$y(n) = \sum_{m=0}^{N-1} h(n-m)x(m) \tag{2.2}$$

where x(n) is the input and h(n) is the length-N impulse response.

The system function of an FIR filter is given by the z transform of h(n) as

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n} \tag{2.3}$$

The frequency response of a filter is found by setting $z=e^{jw}$, which gives (2.3) the form

$$H(w) = \sum_{n=0}^{N-1} h(n)e^{-jwn} \tag{2.4}$$

where w is frequency in radians per second. Actually the exponent should be $-jwTn$, where T is the time interval between the integer steps of n (the sampling interval). In (2.4), T = 1 is assumed. For convenience, throughout this paper the notation H(w) rather than $H(e^{jw})$ is used to denote the frequency response of the digital filter.

This frequency response function (2.4) is complex-valued and consists of a magnitude and a phase. Even though the impulse response is a function of the discrete variable n, the frequency response is a function of the continuous frequency variable w and is periodic with period 2Π. This periodicity is easily shown as follows:

$$H(w+2\Pi) = \sum_{n=0}^{N-1} h(n)e^{-j(w+2\pi)n} \tag{2.5}$$

$$= \sum_{n=0}^{N-1} h(n)e^{-jwn}e^{-j2\pi n}$$

$$= H(w)$$

Frequency is denoted by w in radians per second or by f in hertz (cycles per second). These quantities are related by $w = 2\Pi f$.

The discrete Fourier transform (DFT) can be used to evaluate the frequency response at certain frequencies. The DFT (Rabiner and Gold 1975) of the length-N impulse response h(n) is defined as

$$C(k) = \sum_{n=0}^{N-1} h(n)e^{-j2\pi nk/N} \quad ,k = 0,1,....,N-1 \qquad (2.6)$$

When compared to (2.4), (2.6) gives

$$C(k) = H(w)\big|_{w=2\pi k/N} = H(2\Pi k/N) \quad ,k = 0,1,....,N-1 \qquad (2.7)$$

This states that the DFT of h(n) gives N samples of the frequency response function H(w). This sampling at N points may not give enough detail; therefore, more samples are needed. Any number of equally spaced samples can be found with the DFT by simply appending L-N zeros to h(n) and taking an L-length DFT (Willams 1986).

Particularly useful FIR filters are those with linear phase shift. If the real and imaginary parts of H(w) are given by

$$H(w) = R(w)+jI(w)$$

the magnitude and phase are defined by

$$M(w) = |H(w)| = \sqrt{R^2+I^2}$$

$$\phi(w) = \arctan(I/R)$$

Thus

$$H(w) = M(w)e^{j\phi(w)} \qquad (2.8)$$

Mathematical problems arise, however, because M(w) is not analytic and ø(w) is not continuous. These problems are solved by introducing the real-valued amplitude function A(w) that may be positive or negative, and the frequency response is written as

$$H(w) = A(w)e^{j\theta(w)} \tag{2.9}$$

where A(w) is called the amplitude to distinguish it from the magnitude M(w), and θ(w) is the continuous version of ø(w). A(w) is a real, analytic function related to the magnitude by

$$A(w) = \pm M(w) \tag{2.10}$$

or $|A(w)| = M(w)$. With this definition, A(w) can be made analytic and θ(w) can be made continuous. These quantities are much easier to work with than M(w) and ø(w). The relationships of A(w) and M(w) and of θ(w) and ø(w) are shown in Figure 3.

To develop the characteristics and properties of linear phase filters, a general linear form for the phase function is assumed:

$$\theta(w) = K_1 + K_2 w \tag{2.11}$$

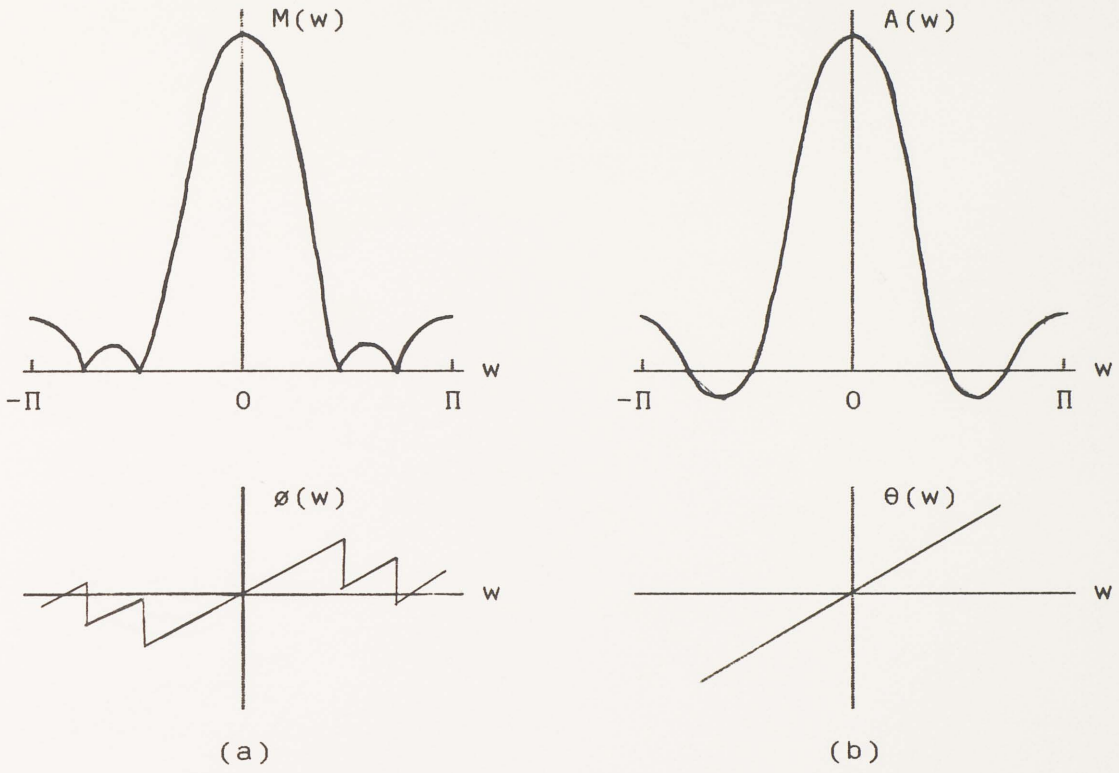


Figure 5. The Magnitude and Amplitude of an Example Linear Phase FIR Filter, (a) Magnitude and Phase (b) Amplitude and Phase.

Equation (2.4) gives the frequency response function of a length-N FIR filter as

$$H(w) = \sum_{n=0}^{N-1} h(n)e^{-jwn}, \tag{2.12}$$

$$H(w) = e^{-jwM} \sum_{n=0}^{N-1} h(n)e^{jw(M-n)}, \tag{2.13}$$

and

$$H(w) = e^{-jwM}\{ h_0e^{jwM} + h_1e^{jw(M-1)} + \ldots + h_{N-1}e^{jw(M-N+1)} \} \tag{2.14}$$

Equation (2.14) can be rewritten in the form

$$H(w) = A(w)e^{j(K1+K2w)} \tag{2.15}$$

if M (not necessarily an integer) is defined by

$$M = \frac{N - 1}{2} \tag{2.16}$$

or, equivalently,

$$M = N-M-1$$

Equation (2.14) then becomes

$$\begin{aligned} H(w) = e^{-jwM}\{ & (h_0 + h_{N-1})\cos(wM) \\ & + j(h_0 - h_{N-1})\sin(wM) \\ & + (h_1 + h_{N-2})\cos(w(M-1)) \\ & + j(h_1 - h_{N-2})\sin(w(M-1) \\ & + ...\}. \end{aligned} \tag{2.17}$$

Equation (2.17) can be expressed in the form of (2.15), where A(w) is real, in two ways: $K_1 = 0$ or $K_1 = \Pi/2$. The first case requires a special even symmetry in h(n) of the form h(n) = h(N-n-1) which gives

$$H(w) = A(w)e^{-jMw} \tag{2.18}$$

where A(w) is a real-valued function of w and e^{-jMw} gives the linear phase. When N is odd,

$$A(w) = \sum_{n=0}^{M-1} 2h(n)\cos[w(M-n)] + h(M) \tag{2.19}$$

With a change of variables,

$$A(w) = \sum_{n=1}^{M} 2h(M-n)\cos(wn) + h(M) \tag{2.20}$$

When N is even,

$$A(w) = \sum_{n=0}^{N/2-1} 2h(n)\cos[w(M-n)] \tag{2.21}$$

With a change of variables,

$$A(w) = \sum_{n=1}^{N/2} 2h(N/2-n)\cos[w(n-1/2)]. \qquad (2.22)$$

When $K_1 = \Pi/2$ in (2.15), an odd symmetry of the form

$$h(n) = -h(N-n-1) \qquad (2.23)$$

is required. H(w) then becomes

$$H(w) = jA(w)e^{-jMw}$$

where A(w) is defined as

for N odd,

$$A(w) = \sum_{n=0}^{M-1} 2h(n)\sin[w(M-n)] \qquad (2.24)$$

for N even,

$$A(w) = \sum_{n=0}^{N/2-1} 2h(n)\sin[w(M-n)] \qquad (2.25)$$

From the previous discussion, it can be seen that there are four possible types of FIR filters leading to the linear phase of (2.15).

Type 1: The impulse response has odd length and is even symmetrical about its midpoint n = M = (N-1)/2, which requires h(n) = h(N-n-1) and gives (2.19) and (2.20).

Type 2: The impulse response has even length and is even symmetrical about M, but M is not an integer. Therefore, there is no h(n) at the point of symmetry, but it satisfies (2.21) and (2.22).

Type 3: The impulse response has odd length and the odd symmetry of (2.23), given an imaginary multiplier for the linear phase form in (2.24).

Type 4: The impulse response has even length and the odd symmetry of (2.23), and satisfies (2.25).

The four types of linear phase FIR filters with the symmetries for odd and even lengths are shown in Figure 4.

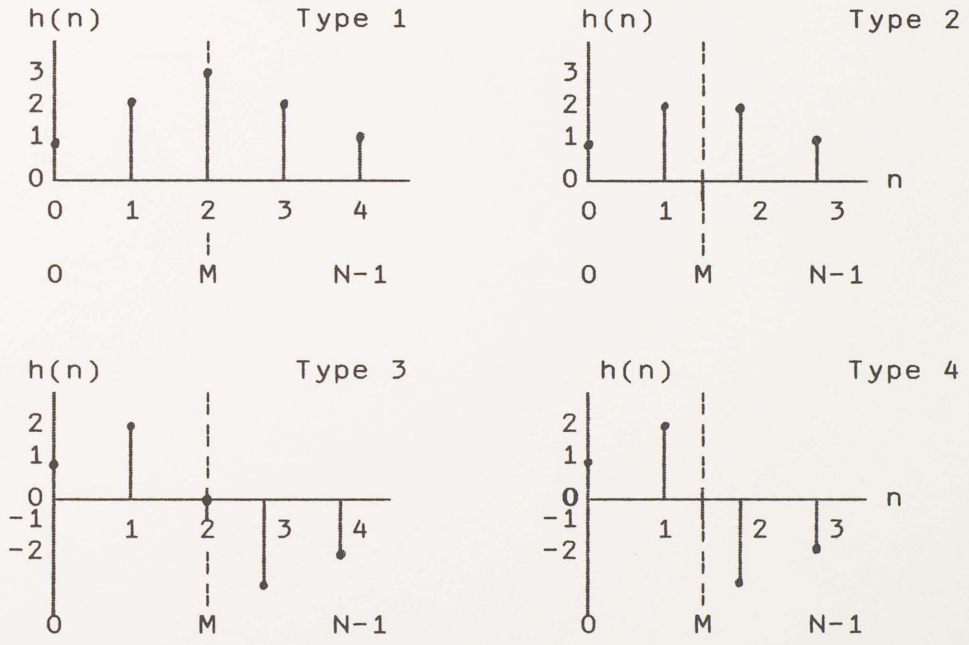


Figure 6. Examples of Impulse Responses for Linear Phase FIR Filters.

To analyze or design linear phase FIR filters, the characteristics of A(w) must to be known. The most important characteristics are concluded in Table 1; Figure 5 illustrates examples of amplitude functions for odd and even length linear phase filters A(w).

TABLE 1

CHARACTERISTICS OF A(W) FOR LINEAR PHASE

Type 1:	Odd length, even symmetrical h(n)	
	A(w) is even about w = 0	A(w) = A(-w)
	A(w) is even about w = Π	A(w+Π) = A(Π-w)
	A(w) is periodic with period 2Π	A(w+2Π) = A(w)
Type 2:	Even length, even symmetrical h(n)	
	A(w) is even about w = 0	A(w) = A(-w)
	A(w) is odd about w = Π	A(w+Π) = -A(Π-w)
	A(w) is periodic with period 4Π	A(w+4Π) = A(w)
Type 3:	Odd length, odd symmetrical h(n)	
	A(w) is odd about w = 0	A(w) = -A(-w)
	A(w) is odd about w = Π	A(w+Π) = -A(Π-w)
	A(w) is periodic with period 2Π	A(w+2Π) = A(w)
Type 4:	Even length, even symmetrical h(n)	
	A(w) is odd about w = 0	A(w) = -A(-w)
	A(w) is even about w = Π	A(w+Π) = A(Π-w)
	A(w) is periodic with period 4Π	A(w+4Π) = A(w)

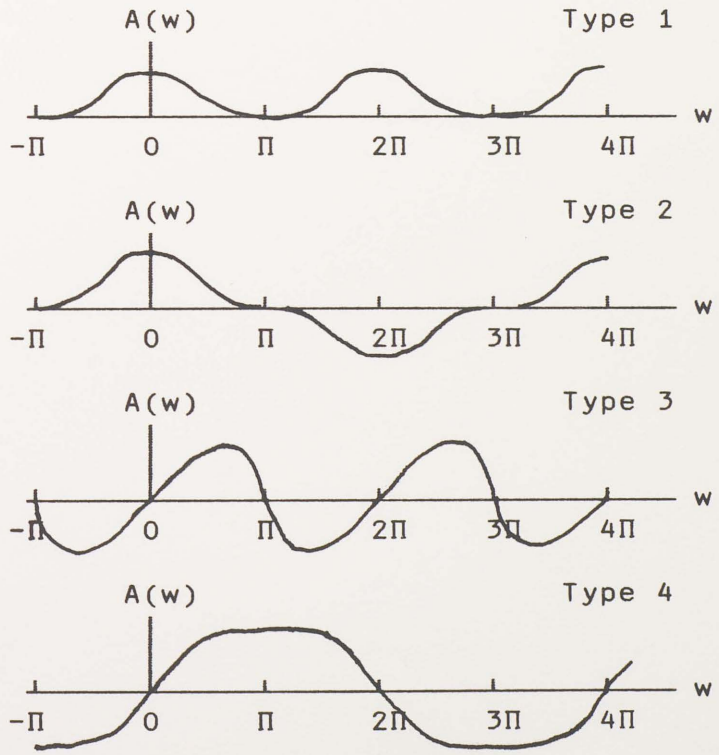


Figure 7. Examples of Amplitude Functions for FIR Filters.

These characteristics reveal several inherent features that are extremely important to filter design. For types 3 and 4, A(0) = 0 for any choice of filter coefficients h(n), which is undesirable for a lowpass filter. Types 2 and 3 always have A(Π) = 0, which is undesirable for a highpass filter. In addition to the linear phase characteristic representing a time shift, types 3 and 4 give a constant 90 degree phase shift, desirable for a differentiator or a Hilbert transformer (Rabiner and Gold 1975).

CHAPTER III

FIR FILTER DESIGN TECHNIQUES

There are essentially three well-known classes of design techniques for linear phase FIR filters: the window method, the frequency sampling and the optimal filter design method. In this chapter, each one of these methods is discussed.

Design Technique One - The Window Method

The periodic frequency response H(w) and its corresponding impulse response sequence are related by the following Fourier transform pair:

$$H(w) = \sum_{n=-\infty}^{\infty} h(n)e^{-jw} \tag{3.1}$$

$$h(n) = \frac{1}{2\Pi} \int_{0}^{2\Pi} H(w)e^{jw}dw \tag{3.2}$$

Unfortunately, two difficulties are encountered when using equation (3.1) and (3.2) to define a FIR filter. First, the filter impulse response is infinite in duration since the summation in (3.1) extends to $\pm\infty$. Second, the filter is unrealizable because the impulse response begins at $-\infty$. Therefore, no finite amount of delay can make the impulse response realizable (i.e., causal).

The most straightforward approach for remedying the former problem is to obtain a finite-duration impulse response sequence by symmetrically truncating the infinite-duration impulse response sequence. However, direct truncation of the sequence leads to the well-known Gibbs phenomenon which manifests itself as a fixed percentage overshoot and ripple before and after an approximated discontinuity in the frequency response. An example of the Gibbs phenomenon of an ideal finite-duration lowpass filter which possess an 8.9% maximum ripple is shown in Figure 8.

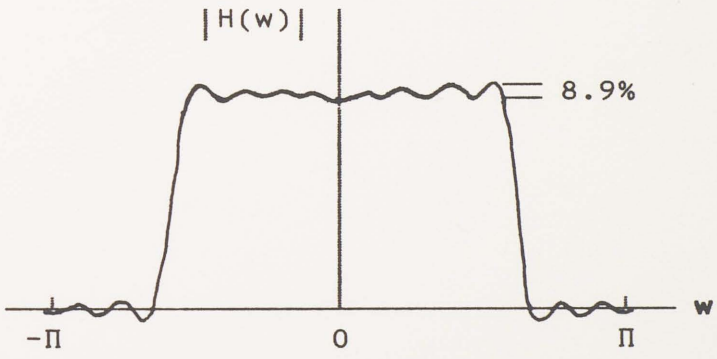


Figure 8. An Example of Gibbs Phenomenon.

In general, all frequency selective filters are ideally discontinuous at the boundaries between bands (see Figure 9); simple truncation of the impulse response will often yield an unacceptable FIR design. This problem can be solved by introducing a finite-duration weighting sequence w(n), called a window, to modify the infinite-duration impulse response sequence h(n).

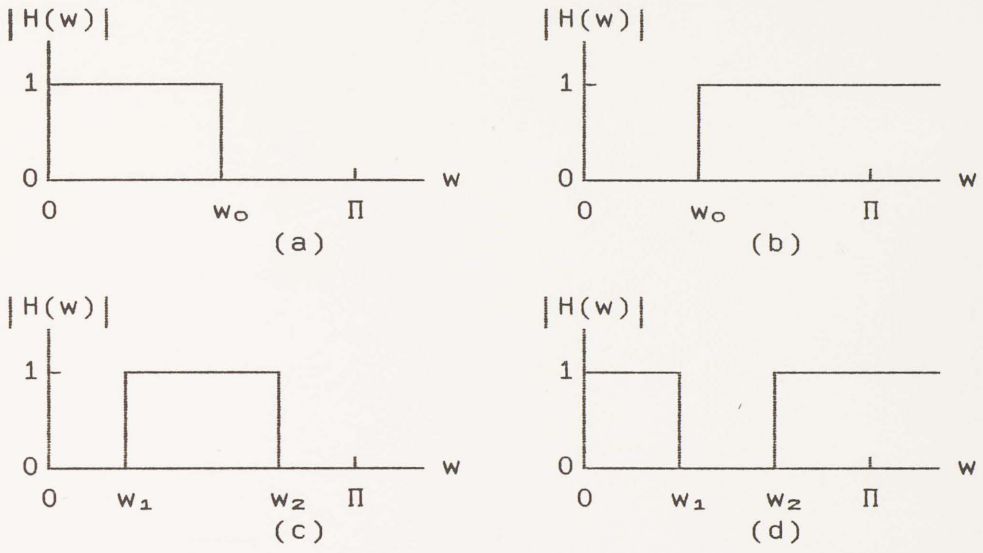


Figure 9. Frequency Response of Ideal (a) Lowpass (b) Highpass (c) Bandpass (d) Bandstop Filters.

The technique of windowing seeks to reduce the Gibbs phenomenon by multiplying the infinite-duration impulse response sequence by a smooth finite-duration window. If the window is denoted as $w(n)$, for $-N_1 \leq n \leq N_2$, and the impulse response of the desired ideal digital filter as $h_d(n)$, for $-\infty < n < \infty$, then the impulse response of the windowed filter is given as

$$h(n) = \begin{cases} h_d(n)w(n), & -N_1 \leq n \leq N_2 \\ 0, & \text{elsewhere} \end{cases} \qquad (3.3)$$

It is known that multiplication of two functions in the time domain corresponds to convolution of the Fourier transforms of the two signals. If $H(w)$, $H_d(w)$, and $W(w)$ represent the Fourier transforms of $h(n)$, $h_d(n)$, and $w(n)$,

respectively, the time-domain truncation operation given in (3.3) is described in the frequency domain by

$$H(w) = \frac{1}{2\Pi} \int_{-\pi}^{\pi} H_d(\theta)W(w-\theta)d\theta = H_d(w)*W(w) \qquad (3.4)$$

For example, if $H_d(w)$ represents an ideal lowpass filter with cutoff frequency w_o and w(n) is a rectangular window positioned about the origin, the H(w) is as shown in Figure 10.

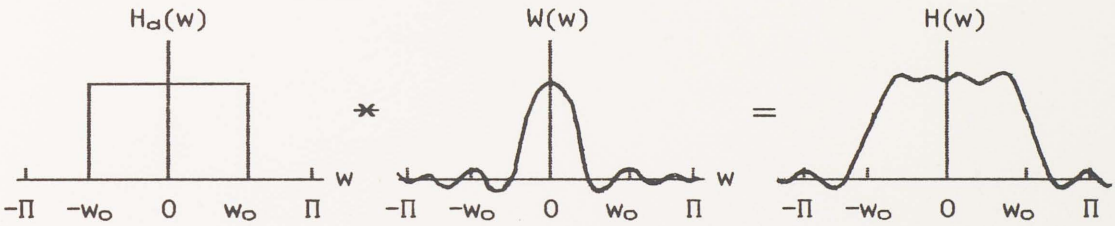


Figure 10. Frequency Response of a Windowed Lowpass Filter.

In general, the frequency response of the window consists of a main lobe representing the middle of the frequency response and various side-lobes located on both sides of the main lobe. The major effect of windowing is the discontinuity in $H_d(w)$ becomes a transition band between pass band and stop band. It is clear that the width of the transition band depends on the width of the main lobe of W(w). A second effect is that the ripple from the side-lobes of W(w) produces errors (ripples) in the frequency response for all w. It is impossible to simultaneously narrow the main lobe and minimize the

side-lobes. Thus, the design of FIR filter by using the window method is obviously an iterative process whereby various window are tried-the results specify the solution.

The ideal filter impulse response sequence $h_d(n)$ in (3.3) can be derived from the frequency response $H_d(w)$ by using the inverse Fourier transform. The $h_d(n)$ of lowpass, highpass, bandpass and bandstop filters are given as follows:

Lowpass filter:

$$h_d(n) = \begin{cases} \sin[2\Pi f_c(n-M)]/\Pi(n-M), & n \neq M \\ 2f_c, & n = M \end{cases} \qquad (3.5)$$

Highpass filter:

$$h_d(n) = \begin{cases} -\sin[2\Pi f_c(n-M)]/\Pi(n-M), & n \neq M \\ 1 - 2f_c, & n = M \end{cases} \qquad (3.6)$$

Bandpass filter:

$$h_d(n) = \begin{cases} 2\{\sin[\Pi(f_H-f_L)(n-M)]\cos[\Pi(f_H+f_L)(n-M)]\} \\ \quad /\Pi(n-M), & n \neq M \\ 2(f_H-f_L), & n = M \end{cases} \qquad (3.7)$$

Bandstop filter:

$$h_d(n) = \begin{cases} -2\{\sin[\Pi(f_H-f_L)(n-M)]\cos[\Pi(f_H+f_L)(n-M)]\} \\ \quad /\Pi(n-M), & n \neq M \\ 1 - 2(f_H-f_L), & n = M \end{cases} \qquad (3.8)$$

where in (3.5)-(3.8),

$M = (N-1)/2$ and $0 \leq n \leq N-1$;

f_c is the normalized filter cutoff frequency for lowpass and highpass filters, $0 \leq f_c \leq 0.5$;

f_H and f_L are the normalized cutoff frequencies for bandpass and bandstop filters, $0 \leq f_L < f_H \leq 0.5$.

The windows used in the filter design package (i.e., CADFIR) are Rectangular, Bartlett, Hamming, Hanning, Generalized Hamming and Kaiser windows. These are defined mathematically as follows (assuming they are symmetrical about n = 0 and N is odd):

Rectangular Window:

$$w(n) = \begin{cases} 1.0, & |n| \leq \tfrac{1}{2}(N-1) \\ 0.0, & \text{elsewhere} \end{cases} \qquad (3.9)$$

Bartlett Window:

$$w(n) = \begin{cases} 1 - |2n|/(N-1), & |n| \leq \tfrac{1}{2}(N-1) \\ 0.0, & \text{elsewhere} \end{cases} \qquad (3.10)$$

Hamming Window:

$$w(n) = \begin{cases} 0.54 + 0.46\cos[2n\Pi/(N-1)], & |n| \leq \tfrac{1}{2}(N-1) \\ 0.0, & \text{elsewhere} \end{cases} \qquad (3.11)$$

Hanning Window:

$$w(n) = \begin{cases} 0.5 + 0.5\cos[2n\Pi/(N-1)], & |n| \leq \tfrac{1}{2}(N-1) \\ 0.0, & \text{elsewhere} \end{cases} \qquad (3.12)$$

Generalized Hamming Window:

$$w(n) = \begin{cases} \alpha + (1-\alpha)\cos[2n\Pi/(N-1)], & |n| \leq \tfrac{1}{2}(N-1) \\ 0.0, & \text{elsewhere} \end{cases} \qquad (3.13)$$

where α is a variable parameter, $0.0 \leq \alpha \leq 1.0$

Kaiser Window:

$$w(n) = \begin{cases} \dfrac{I_o(\beta\sqrt{1 - [2n/(N-1)]^2})}{I_o(\beta)}, & |n| \le \tfrac{1}{2}(N-1) \\ 0.0, & \text{elsewhere} \end{cases} \qquad (3.14)$$

where β is a parameter related to the desired minimum stop band attenuation (ATT). It can be determined by the following equations (Kaiser 1974):

$$\beta = \begin{cases} 0.1102(ATT-8.7), & ATT > 50 \\ 0.5842(ATT-21)^{0.4} + 0.07886(ATT-21), & 21 < ATT \le 50 \\ 0.0, & ATT \le 21 \end{cases} \qquad (3.15)$$

$I_o(x)$ is the modified Bessel function of the first kind and zeroth order (Kaiser 1974). It is defined as

$$I_o(x) = 1 + \sum_{k=1}^{\infty} \left[\frac{(x/2)^k}{k!} \right]^2 \qquad (3.16)$$

Description of the Design Algorithm

In order to design FIR windowed digital filters, the following specifications are required:

1. Type of filter - i.e., lowpass, highpass, bandpass or bandstop filter.
2. Type of window - i.e., Rectangular, Bartlett, Hamming, Hanning, Generalized Hamming, Kaiser.
3. Filter cutoff frequencies.
4. Window duration and/or parameter - i.e., α or ATT.

From the given filter specifications the ideal impulse response $h_d(n)$ is calculated from one of the formulas in (3.5)-(3.8) and the window coefficients w(n) can be obtained by using the window functions in (3.9)-(3.14). Then the windowed filter impulse response h(n) in (3.3) and its frequency response in terms of DFT coefficients are computed as the final outputs. Figure 11 provides a summary of the overall flowchart of the windowed filter design algorithm.

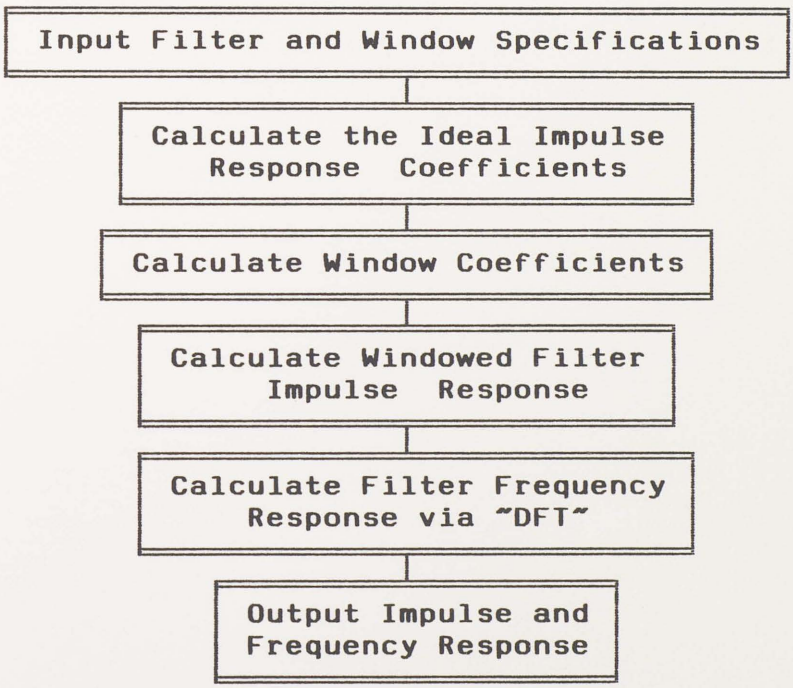


Figure 11. Overall Flowchart of the Windowed Filter Design Algorithm.

A Design Example

In this example a length-21 lowpass filter is designed with ideal cutoff frequency of 0.3. A Hamming is used for this case. To design this filter with CADFIR the following steps are required.

1. Select command "Creat" from the main menu.
2. Select design method "Window."
3. Select the filter type "Lowpass."
4. Select the window type "Hamming."
5. Enter 21 for the window duration and 0.3 for the filter band edge.
6. Enter the design name.

The resulting filter coefficients (impulse response) are

h[000] = +1.2146571249e-010 = h[020]

h[001] = -3.4482360352e-003 = h[019]

h[002] = +3.9255940355e-003 = h[018]

h[003] = +7.2064464912e-003 = h[017]

h[004] = -2.0073669031e-002 = h[016]

h[005] = -8.1989431910e-010 = h[015]

h[006] = +5.1626771688e-002 = h[014]

h[007] = -5.0540167838e-002 = h[013]

h[008] = -8.5330463946e-002 = h[012]

h[009] = +2.9591500759e-001 = h[011]

h[010] = +6.0000002384e-001 = h[010]

Plots of the impulse response and frequency response are shown in Figure 12. Note that the frequency response of this filter has smooth pass band and stop band but a wide transition band.

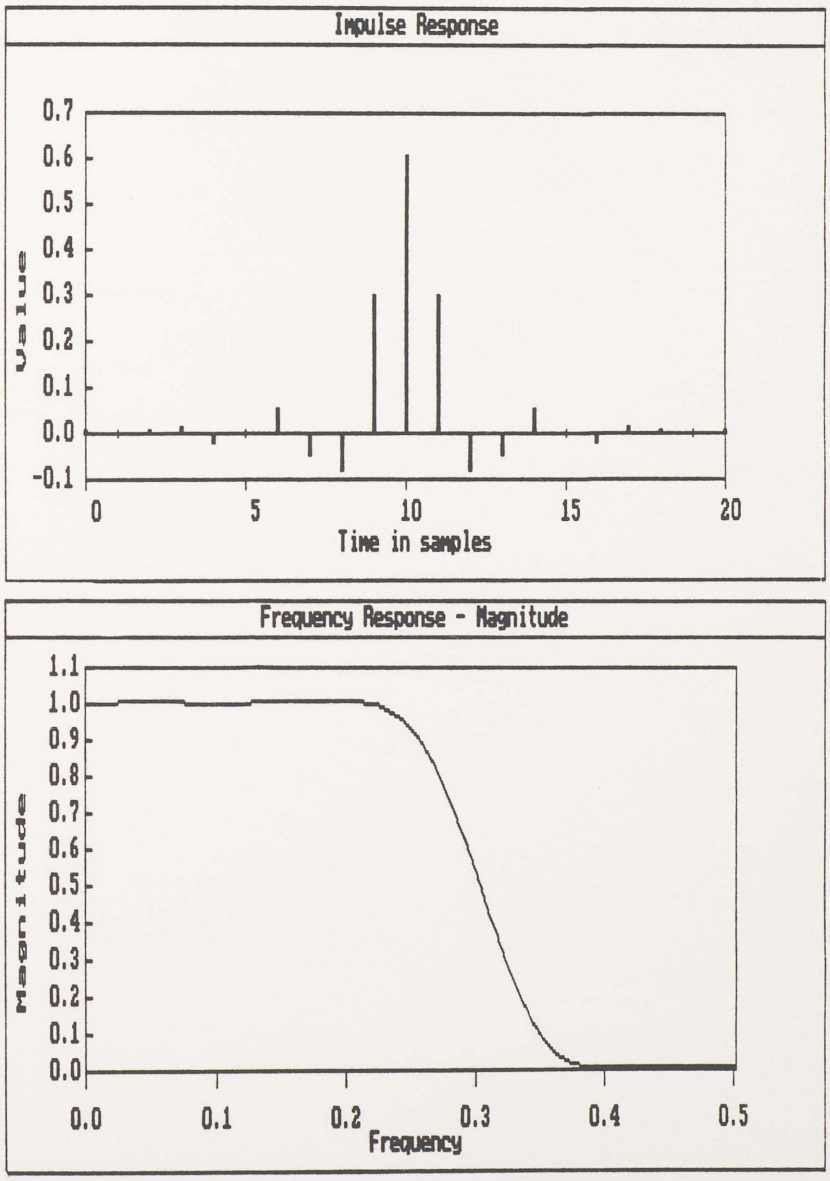


Figure 12. Plots of Impulse Response and Frequency Response (magnitude) of a Windowed Filter Design Example.

Design Technique Two - Frequency Sampling

It is known that the DFT of impulse response gives samples of the frequency response, and the inverse DFT (IDFT) of samples of a desired frequency response will give the impulse response. Thus, if we are given equally spaced samples of a desired frequency response, then IDFT can be used to solve the impulse response.

For N equally spaced frequency response samples of H(w), i.e.,

$$H(k) = H(w)\Big|_{w=2\pi k/N} = \sum_{n=0}^{N-1} h(n)e^{-j2\pi nk/N}$$

the length-N FIR filter coefficients are given by the IDFT as

$$h(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(k)e^{j2\pi nk/N} \qquad (3.17)$$

When H(w) is linear phase, (3.17) may be simplified by the formulas in Chapter II for the four types of linear phase FIR filters. For example, the frequency response for the type 1 filter with odd N and a frequency sample at w = 0 is defined by (2.19) as

$$A(w) = \sum_{n=0}^{M-1} 2h(n)\cos[w(M-n)] + h(M) \qquad (3.18)$$

where M = (N-1)/2.

Let w = 2Πk/N, then (3.18) can be rewritten as

$$A(k) = \sum_{n=0}^{M-1} 2h(n)\cos[2\pi k(M-n)/N] + h(M) \qquad (3.19)$$

Using the amplitude function A(w), defined in (2.18), of the form (3.19) and the IDFT (3.17) gives the impulse response

$$h(n) = \frac{1}{N} \sum_{k=0}^{N-1} e^{-j2\pi Mk/N} A(k) e^{j2\pi nk/N}$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} A(k) e^{j2\pi k(n-M)/N} \qquad (3.20)$$

Because h(n) is real, A(k) = A(N-k) and (3.20) becomes

$$h(n) = \frac{1}{N} \left[A(0) + \sum_{k=1}^{M} 2A(k)\cos[2\Pi(n-M)k/N] \right] \qquad (3.21)$$

Only M of the h(n) need to be calculated since h(n) is even symmetry, i.e., h(n) = h(N-n-1).

A similar development applied to (2.21), the case for N even, gives the frequency samples

$$A(K) = \sum_{n=0}^{N/2-1} 2h(n)\cos[2\Pi(M-n)k/N] \qquad (3.22)$$

The design formula becomes

$$h(n) = \frac{1}{N} \left[A(0) + \sum_{k=1}^{N/2-1} 2A(k)\cos[2\Pi(n-M)k/N] \right] \qquad (3.23)$$

which is of the same form as (3.21), except that the upper limit of summation is changed for even N and A(N/2) = 0.

For the odd symmetric filters, type 3 and 4, similar developments can easily be performed, with the result closely related to the discrete sine transform.

For type 3 the analysis and design formulas are

$$A(K) = \sum_{n=0}^{M-1} 2h(n)\sin[2\Pi(M-n)k/N], \tag{3.24}$$

$$h(n) = \frac{1}{N}\left[\sum_{k=1}^{M} 2A(k)\sin[2\Pi(M-n)k/N]\right]. \tag{3.25}$$

For type 4

$$A(K) = \sum_{n=0}^{N/2-1} 2h(n)\sin[2\Pi(M-n)k/N], \tag{3.26}$$

$$h(n) = \frac{1}{N}\left[\sum_{k=1}^{N/2-1} 2A(k)\sin[2\Pi(M-n)k/N] + A(N/2)\sin(\Pi(M-n))\right]. \tag{3.27}$$

The scheme just described uses frequency samples at

$$w = 2\Pi k/N, \qquad k = 0,1,2....,N-1. \tag{3.28}$$

which are N equally spaced samples starting at w = 0. Another possible frequency sampling scheme allows design formulas which have no samples at w = 0 but uses N equally spaced samples located at

$$w = (2k+1)\Pi/N \qquad k = 0,1,2....,N-1. \tag{3.29}$$

The analysis and design formulas based on this scheme are given as follows:

For type 1

$$A(k) = \sum_{n=0}^{M-1} 2h(n)\cos[2\Pi(M-n)(k+\tfrac{1}{2})/N] + h(M), \tag{3.30}$$

$$h(n) = \frac{1}{N}\left[\sum_{k=0}^{M-1} 2A(k)\cos[2\Pi(n-M)(k+\tfrac{1}{2})/N] + A(M)\cos\Pi(n-M)\right]. \quad (3.31)$$

For type 2

$$A(K) = \sum_{n=0}^{N/2-1} 2h(n)\cos[2\Pi(M-n)(k+\tfrac{1}{2})/N], \quad (3.32)$$

$$h(n) = \frac{1}{N}\left[\sum_{k=0}^{N/2-1} 2A(k)\cos[2\Pi(M-n)(k+\tfrac{1}{2})/N]\right]. \quad (3.33)$$

For type 3

$$A(k) = \sum_{n=0}^{M-1} 2h(n)\sin[2\Pi(M-n)(k+\tfrac{1}{2})/N], \quad (3.34)$$

$$h(n) = \frac{1}{N}\left[\sum_{k=0}^{M-1} 2A(k)\sin[2\Pi(n-M)(k+\tfrac{1}{2})/N]\right]. \quad (3.35)$$

For type 4

$$A(k) = \sum_{n=0}^{M-1} 2h(n)\sin[2\Pi(M-n)(k+\tfrac{1}{2})/N], \quad (3.36)$$

$$h(n) = \frac{1}{N}\left[\sum_{k=0}^{M-1} 2A(k)\sin[2\Pi(n-M)(k+\tfrac{1}{2})/N]\right]. \quad (3.37)$$

The importance of the second scheme lies in the additional flexibility that is given to the design method to specify the desired frequency response at a second possible set of frequencies. Thus a given band edge frequency may be much closer to a sample defined by the second scheme than to a sample defined by the first scheme.

Description of the Design Algorithm

To design a FIR filter via frequency sampling method the following specifications are needed:

1. Type of filter - i.e., lowpass, highpass, bandpass or bandstop filter.
2. Sampling scheme - i.e, $w = 2k\Pi/N$ or $w = (2k+1)\Pi/N$
3. Filter cutoff frequencies and duration.

From the given filter specifications the desired frequency response samples (equally spaced) are set to be 1 for the pass band and 0 for the stop band. Then the design formulas developed earlier are used to obtain the impulse response h(n), and the frequency response coefficients are evaluated by DFT. Figure 13 provides an overall flowchart of the frequency sampling design algorithm.

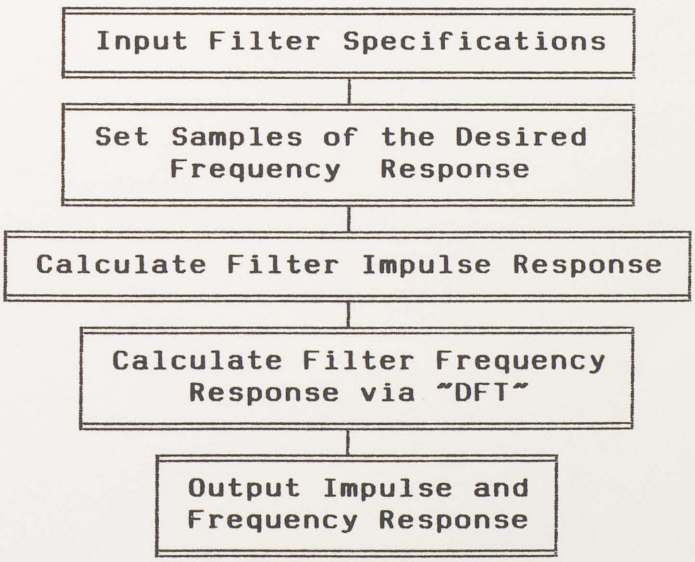


Figure 13. Overall Flowchart of the Frequency Sampling Design Algorithm.

A Design Example

A length-21 lowpass filter is designed with an ideal cutoff frequency of 0.3. The first sampling scheme is used for this case. To design this filter with CADFIR the following steps are required.

1. Select command "Create" from the main menu.
2. Select design method "Frequency Sampling."
3. Select the filter type "Lowpass."
4. Select the first sampling scheme.
5. Enter 21 for the filter length and 0.3 for the filter band edge.
6. Enter the design name.

The resulting filter coefficients (impulse response) are

h[000] = +2.6899982244e-002 = h[020]

h[001] = -4.7619048506e-002 = h[019]

h[002] = +7.6242899522e-003 = h[018]

h[003] = +4.7619048506e-002 = h[017]

h[004] = -4.7619048506e-002 = h[016]

h[005] = -2.0635873079e-002 = h[015]

h[006] = +8.4296479821e-002 = h[014]

h[007] = -4.7619041055e-002 = h[013]

h[008] = -1.0988502204e-001 = h[012]

h[009] = +2.9741445184e-001 = h[011]

h[010] = +6.1904764175e-001 = h[010]

Plots of the frequency response and impulse response are shown in Figure 14. Note the ripples in the pass band and the stop band near the band edge. The overshoot is caused by the discontinuity, between the pass band and stop band, of the desired frequency response samples.

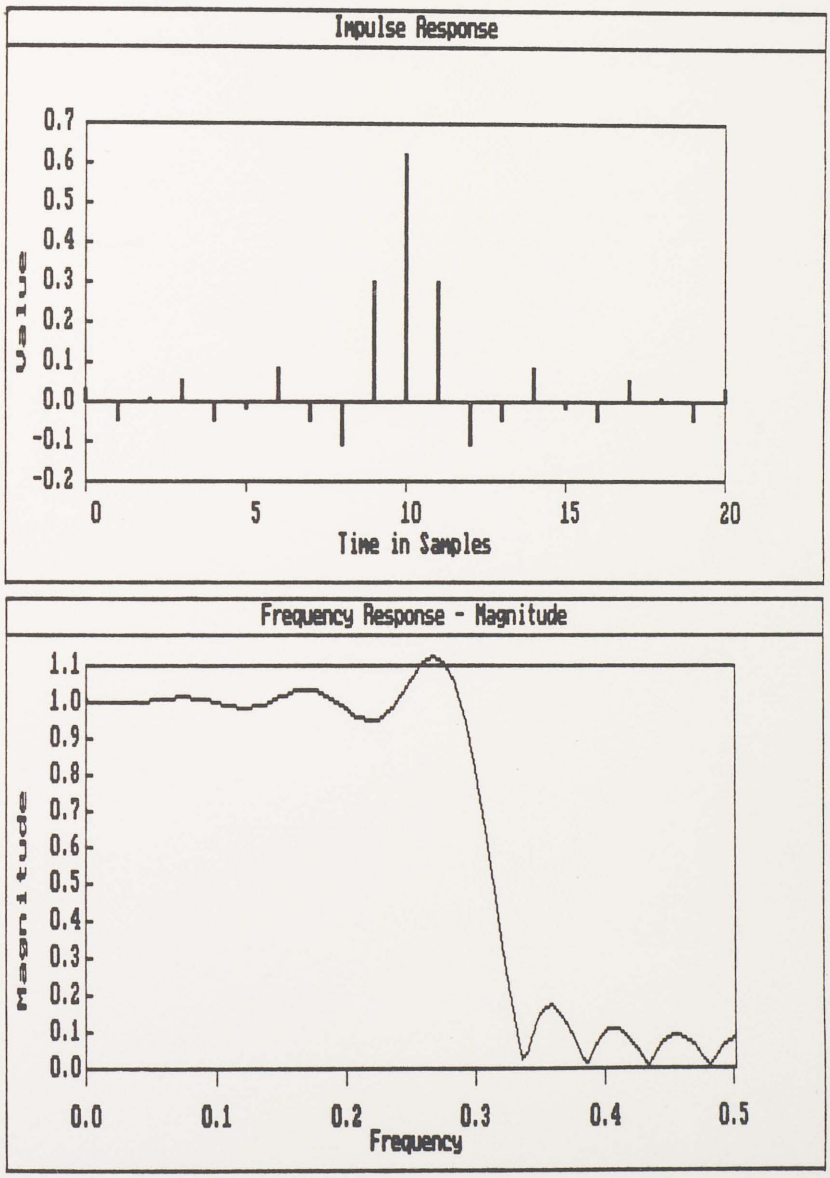


Figure 14. Plots of Impulse Response and Frequency Response (Magnitude) of the Example Designed by Frequency Sampling Method.

Design Technique Three - Optimal Filter Design

The term optimal filter means that a linear phase FIR filter has a frequency response with a minimum value for the maximum error (ripple). A well-known approximation technique that can be used to minimize the maximum error over a set of frequencies is called Chebyshev approximation. Several optimization procedures have been developed to solve the Chebyshev approximation problem (Rabiner and Gold 1975). In this project, the design approach is based on the algorithm developed by Parks and McClellan (Parks and McClellan 1972). The algorithm applies the Remez exchange algorithm to minimize the weighted Chebyshev error in approximating a desired ideal frequency response H(w).

As indicated in Chapter I, there are four types of linear phase FIR filters. All four types have a frequency response

$$H(f) = (j)^m A(f) e^{-j((N-1)/2)2\pi f} \qquad m = 0,1 \qquad (3.38)$$

where A(f) is a real-valued positive or negative function. The frequency variable f, with units of cycles per second or hertz, is used along with a normalized unit sampling rate. The relation between this frequency variable and the radian frequency w is $w = 2\pi f$.

It is shown that A(f) can always be written as a weighted sum of cosines for all four types of linear phase filters (McClellan and Parks 1973). These formulas can be derived from equations (2.19), (2.21), (2.24) and (2.25)

with the use of appropriate trigonometric identities. The specific form of A(f) is given in Table 2.

TABLE 2

APPROXIMATION FUNCTIONS FOR LINEAR PHASE FILTERS

	EVEN SYMMETRY (m=0) $h(n) = h(N-n-1)$	ODD SYMMETRY (m=1) $h(n) = -h(N-n-1)$
ODD LENGTH	$A(w) = \sum_{k=0}^{\frac{1}{2}(N-1)} a_k \cos 2\pi kf$ $a_0 = h(\frac{1}{2}(N-1))$ $a_k = h(-k+\frac{1}{2}(N-1))$ $k = 1,\ldots,\frac{1}{2}(N-1)$	$A(w) = \sin 2\pi f \sum_{k=0}^{\frac{1}{2}(N-3)} c_k \cos 2\pi kf$ $c_0 - \frac{1}{2}c_2 = 2h(\frac{1}{2}(N-3))$ $c_{\frac{1}{2}(N-5)} = 4h(1)$ $c_{\frac{1}{2}(N-3)} = 4h(0)$ $c_{k-1} - c_{k+1} = 2h(-k+\frac{1}{2}(N-1))$ $k = 2,\ldots,\frac{1}{2}(N-5)$
EVEN LENGTH	$A(w) = \cos\pi f \sum_{k=0}^{\frac{1}{2}N-1} b_k \cos 2\pi kf$ $b_0 + \frac{1}{2}b_1 = 2h(\frac{1}{2}N-1)$ $b_{\frac{1}{2}N-1} = 4h(0)$ $b_{k-1} + b_k = 4h(-k+\frac{1}{2}(N-1))$ $k = 2,\ldots,\frac{1}{2}N-1$	$A(w) = \sin\pi f \sum_{k=0}^{\frac{1}{2}N-1} d_k \cos 2\pi kf$ $d_0 - \frac{1}{2}d_1 = 2h(\frac{1}{2}N-1)$ $d_{\frac{1}{2}N-1} = 4h(0)$ $d_{k-1} - d_k = 4h(-k+\frac{1}{2}(N-1))$ $k = 2,\ldots,\frac{1}{2}N-1$

In general, the frequency responses of ideal filters (see Figure 9) are real-valued (no phase shift), exact unity in the pass band and exact zero for the entire stop band. It is impossible for a causal FIR filter to have the same frequency response as the ideal filters. However, an acceptable frequency response (shown in Figure 15) should have the following characteristics:

1. Linear phase.

2. A width Δf transition band between the pass band and stop band.
3. A ripple from unity of A(f) in the pass band of $\pm\delta_1$.
4. A ripple from zero of A(f) in the stop band of $\pm\delta_2$.

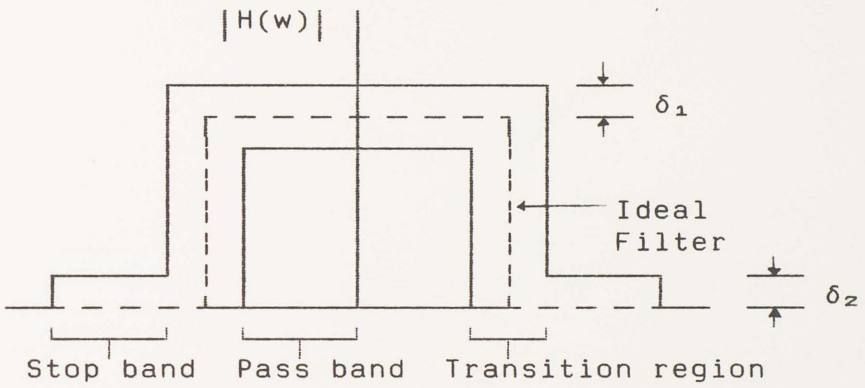


Figure 15. Frequency Response of an Acceptable Lowpass Filter.

A more general filter design problem could have several pass bands and several stop bands. Further, some of these bands may be more important than others; therefore, different weights could be put in different bands to choose the relative size of the ripples. The multiple bands are assumed to make up a compact subset of frequency band [0.0,0.5]. The compact subset F in most applications is the union of closed intervals (corresponding to frequency bands) and discrete frequency points. These requirements for a good linear phase filter are summarized in the following statement of the approximation problem for linear phase design.

The approximation problem for linear phase filter design can be stated as follow:

Given

A compact subset F of [0,0.5].

A desired magnitude response D(f), continuous on F.

A positive weight function W(f), continuous on F.

The form of A(f),

$$A(f) = Q(f) \sum_{k=0}^{r-1} \alpha(k)\cos(2\Pi kf) \tag{3.39}$$

We want to minimize the maximum absolute weighted error, defined as

$$||E(f)|| = \max_{f \in F} W(f)|D(f)-A(f)| \tag{3.40}$$

over $\alpha(k)$ in (3.39) by choice of A(f).

Each of the four types of linear phase filters in Table 2 is described by (3.39), where, by definition,

$$Q(f) = \begin{cases} 1, & \text{odd length, even symmetry} \\ \cos\Pi f, & \text{even length, even symmetry} \\ \sin 2\Pi f, & \text{odd length, odd symmetry} \\ \sin\Pi f, & \text{even length, odd symmetry} \end{cases} \tag{3.41}$$

and

$$\alpha(k) = \begin{cases} a_k, & \text{odd length, even symmetry} \\ b_k, & \text{even length, even symmetry} \\ c_k, & \text{odd length, odd symmetry} \\ d_k, & \text{even length, odd symmetry} \end{cases} \tag{3.42}$$

If we define the sum of cosines in (3.39) as

$$P(f) = \sum_{k=0}^{r-1} \alpha(k)\cos(2\Pi kf) \qquad (3.43)$$

Thus, (3.39) becomes $A(f) = Q(f)P(f)$ and the error function can be written in the form

$$E(f) = W(f)[D(f)-Q(f)P(f)]. \qquad (3.44)$$

By factoring $Q(f)$ out of (3.44), given

$$E(f) = W(f)Q(f)[D(f)/Q(f)-P(f)] \qquad (3.45)$$

Equation (3.45) is valid, if we are careful to omit those endpoints where $Q(f) = 0$. Letting $D'(f) = D(f)/Q(f)$ and $W'(f) = W(f)Q(f)$, then an equivalent approximation problem would be to minimize the quantity

$$||E(f)|| = \max_{f \in F'} W'(f)|D'(f)-P(f)| \qquad (3.46)$$

by choice of the coefficients of $P(f)$. The set F is replaced by $F' = F$ - [endpoints where $Q(f) = 0$].

The problem stated above, which minimizes the maximum error over a set of frequencies, is the Chebyshev approximation problem for designing FIR filters. This problem leads directly to a characterization of the optimal filter in terms of the alternation theorem (Rabiner, McClellan, and Parks 1975).

Alternation Theorem: If P(f) is a linear combination of r cosine functions-that is, if

$$P(f) = \sum_{k=0}^{r-1} \alpha(k)\cos(2\Pi kf)$$

then a necessary and sufficient condition that P(f) be the unique, best weighted Chebyshev approximation to a continuous function D'(f) on F is that the weighted error function E(f) = W'(f)•[D'(f)-P(f)] exhibit at least r+1 extremal frequencies in F', i.e., there must exist r+1 points f_m in F' such that $f_o < f_1 < \ldots < f_r$, with

$$E(f_m) = -E(f_{m+1}), \qquad m = 0,1,\ldots,r-1 \qquad (3.47)$$

and

$$|E(f_m)| = \max_{f \in F'} E(f) \qquad m = 0,1,\ldots,r \qquad (3.48)$$

The alternation theorem means that the best Chebyshev approximation must necessarily have an equiripple error function E(f). It also states that there is a unique best approximation for a given set of frequencies, filter length, and weight function W'(f). Now, the problem of designing the filter has been reduced to the problem of finding the extremal frequencies. It was proved that the Remez exchange algorithm is valuable in finding the extremal frequencies (Parks and McClellan 1972).

The Remez exchange algorithm makes use of the fact that it is always possible to make the error function

$$E(f) = W'(f)[D'(f) - P(f)] \tag{3.49}$$

take on the values $\pm\delta$ for any given set of r+1 extremal frequencies f_m, m = 0,..., r. In other words, the set of equations

$$W'(f)[D'(f)-P(f)] = (-1)^m\delta, \quad m = 0,.....,r \tag{3.50}$$

or in matrix form (Note that $P(f) = \sum_{k=0}^{r-1} \alpha(k)\cos(2\Pi kf)$)

$$\begin{bmatrix} 1 & \cos2\Pi f_0 & \dots & \cos[(r-1)2\Pi f_0] & \frac{1}{W'(f_0)} \\ 1 & \cos2\Pi f_1 & & & \\ \vdots & & & & \\ 1 & \cos2\Pi f_r & & & \frac{(-1)^r}{W'(f_r)} \end{bmatrix} \begin{bmatrix} \alpha(0) \\ \alpha(1) \\ \vdots \\ \alpha(r-1) \\ \delta \end{bmatrix} = \begin{bmatrix} D'(f_0) \\ D'(f_1) \\ \vdots \\ D'(f_r) \end{bmatrix} \tag{3.51}$$

has a unique solution for δ.

Given a dense grid of frequency points to form the compact subset F of [0.0,0.5], the Remez exchange algorithm consists of the following basic steps:

1. Initialize a trial set of extremal frequencies f_0, f_1,.....,f_r by selecting r+1 frequencies, which are equally spaced in the pass band(s) and stop band(s) of the filter, on the grid of frequencies.
2. Solve the equations in (3.50) or (3.51). This solution has an error that oscillates with

amplitude δ_k on the trial set of frequencies for the k-th iteration.

3. Interpolate to find the frequency response on the entire grid of frequencies.
4. Search over the entire grid of frequencies to see if (and where) the magnitude of the error in (3.49), i.e., $|E(f)|$, is larger than the magnitude of δ_k found in step 2.
5. If the maximum value of the error magnitude found in step 4 equals δ_k, stop. If not, take the r+1 frequencies where the error attains its maximum magnitude as the new trial set of extremal frequencies and go to step 2.

In step 2, direct solving (3.50) or (3.51) is both difficult and inefficient. A more efficient approach is to calculate δ using (Rabiner, McClellan, and Parks 1975, eq. 22), i.e.,

$$\delta = \frac{\sum_{m=0}^{r} a_m D'(f_m)}{\sum_{m=0}^{r} \frac{(-1)a_m}{W'(f_m)}} \qquad (3.52)$$

where

$$a_m = \prod_{\substack{m=0 \\ m \neq k}}^{r} \frac{1}{(x_m - x_k)} \qquad (3.53)$$

and $x_k = \cos 2\pi f_k$.

In step 3, P(f) can be interpolated by using (Rabiner, McClellan, and Parks 1975, eq. 25), i.e.,

$$P(f) = \frac{\sum_{m=0}^{r-1} \left[\frac{\beta_m}{x - x_m} \right] C_m}{\sum_{m=0}^{r-1} \left[\frac{\beta_m}{x - x_m} \right]} \qquad (3.54)$$

where

$$C_m = D'(f_m) - (-1)^m \frac{\delta}{W'(f_m)} , \qquad (3.55)$$

$$\beta_m = \prod_{\substack{m=0 \\ m \neq k}}^{r-1} \frac{1}{(x_m - x_k)} , \qquad (3.56)$$

and $x_k = \cos 2\pi f_k$.

After the extremal frequencies are found, the P(f) can be interpolated at a set of equally spaced frequencies. Then the inverse DFT can be used to calculate the coefficients of the best approximation (i.e., $\alpha(k)$).

After $\alpha(k)$ are found the filter impulse response coefficients can be easily computed by using the formulas listed in Table 2.

Description of the Design Algorithm

To design an optimal FIR filter the following specifications are needed:

1. Filter length.
2. The frequency bands, specified by upper and lower cutoff frequencies.
3. The desired frequency response D(f) in each band - i.e., 1 for pass band and 0 for stop band.
4. A positive weight value W(f) in each band.

From the given specifications, the set F of frequencies is made by a dense grid with the number of frequency points equal 16 times the filter length. Both D(f) and W(f) are evaluated on this grid. Then the auxiliary approximation problem is set up by forming D'(f) and W'(f) as described earlier, and an initial guess of the extremal frequencies is made by taking r+1 (r is the number of cosine basis functions) equally spaced frequencies values. Next, the Remez exchange is used to perform the calculation of the best approximation equivalent problem. Finally, the appropriate formulas in Table 2 are applied to recover the impulse response from the coefficients of the best cosine approximation obtained by the Remez exchange algorithm.

Figure 16 provides a summary of the overall flowchart of the optimal filter design algorithm.

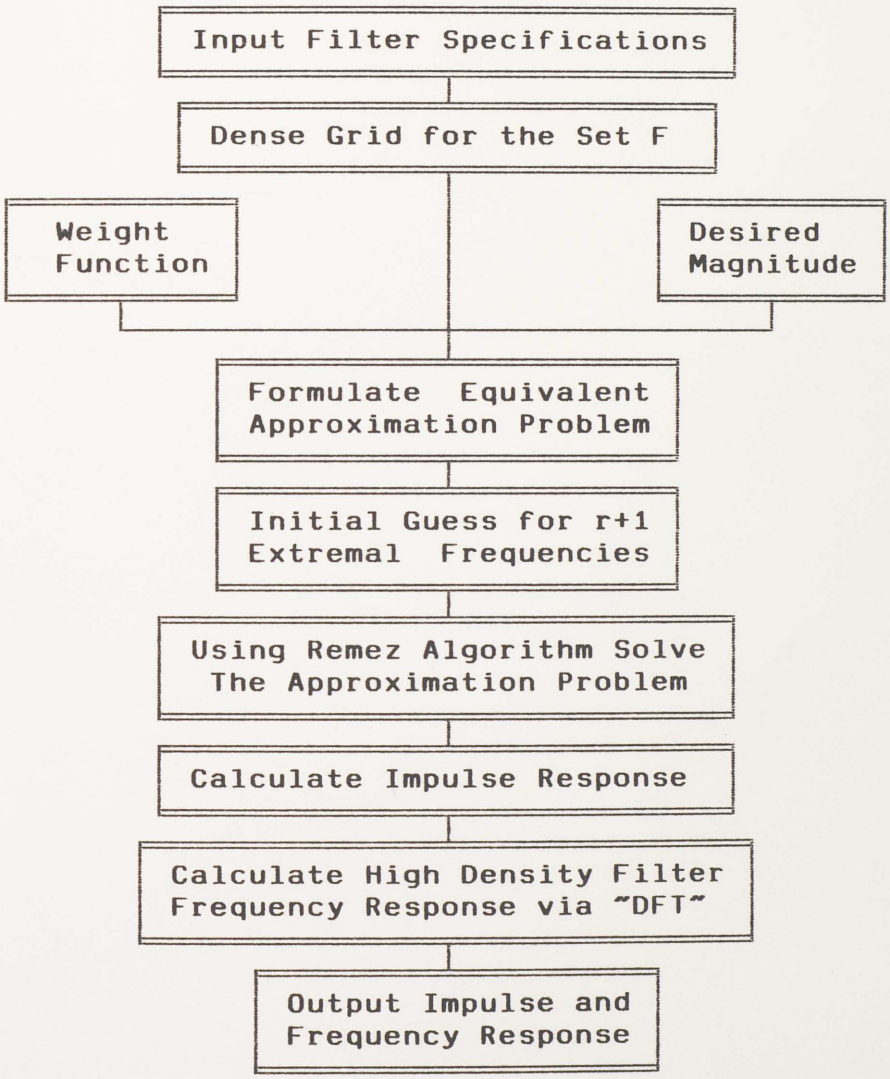


Figure 16. Overall Flowchart of the Optimal Filter Design Algorithm.

A Design Example

In this design of a length-21 lowpass filter, the pass band was set with a frequency range from 0.0 to 0.27. The stop band was specified to begin at 0.33, and the errors in both bands were given the same weight as 1. To design this filter with CADFIR the following steps are required.

1. Select command "Create" from the main menu.
2. Select design method "Optimal."
3. Select the number of bands "2."
4. Enter filter length and the band edges as specified above.
5. Enter the design name.

The resulting filter coefficients (impulse response) are

h[000] = -4.4596185908e-003 = h[020]

h[001] = -3.4870199859e-002 = h[019]

h[002] = +1.7714956775e-002 = h[018]

h[003] = +1.9718796015e-002 = h[017]

h[004] = -4.0848415345e-002 = h[016]

h[005] = +7.5142197602e-005 = h[015]

h[006] = +6.8993128836e-002 = h[014]

h[007] = -5.9293776751e-002 = h[013]

h[008] = -9.1351397336e-002 = h[012]

h[009] = +3.0105113983e-001 = h[011]

h[010] = +5.9990268946e-001 = h[010]

Plots of the frequency response and impulse response are shown in Figure 17. Note that the ripples in the pass band and the stop band have equal magnitudes because we set the same weight for both bands.

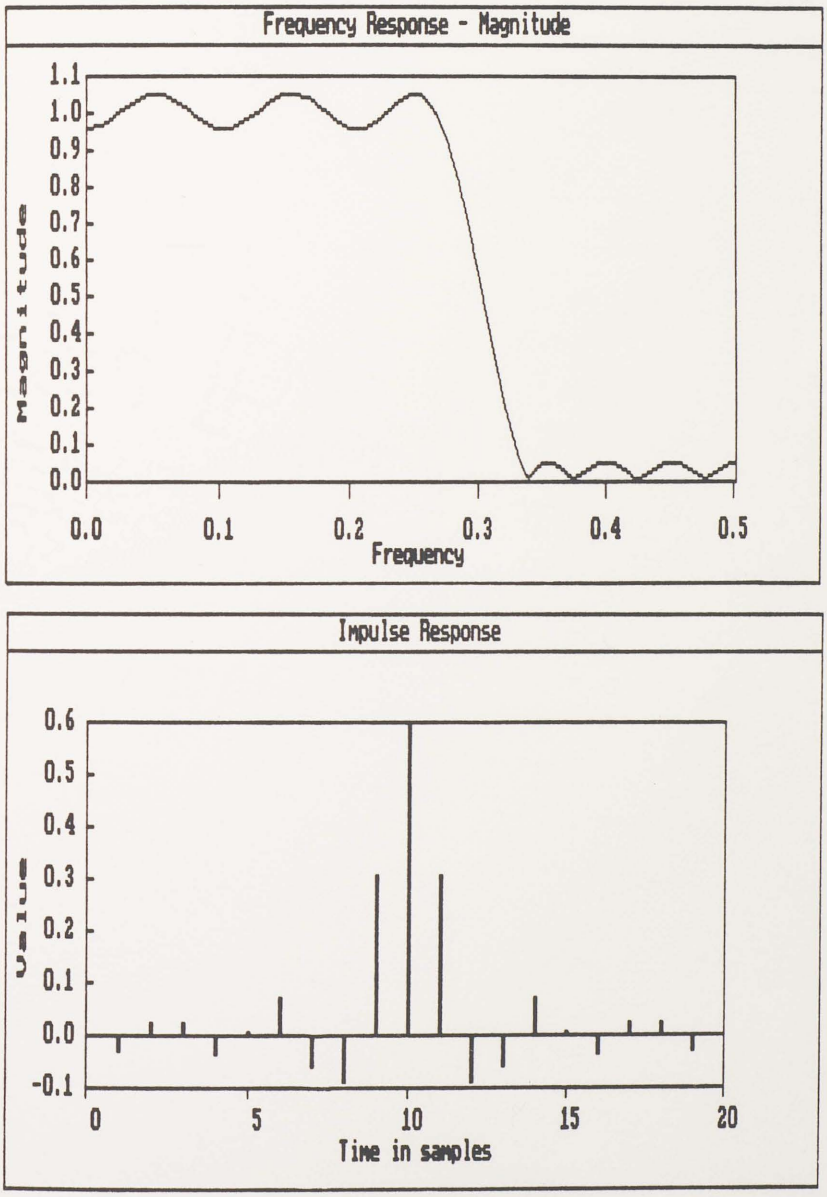


Figure 17. Plots of Impulse Response and Frequency Response (Magnitude) of an Optimal Filter Design Example.

CHAPTER IV

CONCLUSIONS AND SUMMARY

The purpose of this research paper was to present an informative overview of the finite-duration impulse response (FIR) filters design techniques and to implement these design techniques via a personal computer. Attempting to accomplish this task, information concerned with FIR filters was initially presented. The fundamental concepts were developed to investigate the properties of FIR filters; the characteristics of linear phase FIR filters are the major concern of this report. The three well-known design techniques were discussed in order to develop the programming algorithms associated with each design method.

The software package, CADFIR, developed for this project is designed for ease of use. It can be used to design linear phase FIR filters via three well-known design techniques, i.e., the window method, the frequency sampling method, and the optimal filter design method. Furthermore, CADFIR has options for designing such standard filters as lowpass, highpass, bandpass, and bandstop filters as well as multipassband-stopband filters.

Summarizing, each design technique has its advantages and disadvantages. The frequency sampling method is fast and simple; unfortunately, it gives the least control over the total frequency response. The use of windows is a simple method of controlling the oscillations or overshoots of the frequency response, but the design results are not optimal according to any known criterion. The optimal filter design method minimizes the Chebyshev error, but the design algorithm can be slow. Therefore, the designer has the responsibility to choose an appropriate design method to implement a specific filter. CADFIR provides three design methods, that make the design of FIR linear phase filters more convenient and easy.

APPENDIX A

HARDWARE AND SOFTWARE REQUIREMENTS FOR USING CADFIR

1. CADFIR will run on the IBM-PC family of computers, including the XT and AT, along with other IBM compatible computers. It requires a minimum of 384K of RAM and will operate under DOS version 2.1 or higher.

2. CADFIR will display plots in graphics mode if you have a CGA or EGA graphics card, but a graphics card is not required.

3. All printers in the IBM line or other printers supported with an IBM compatible mode can be used.

APPENDIX B

DESIGN EXAMPLES

Example 1

Design Specifications:

Design technique: Window

Filter type: Highpass

Window type: Hanning

Band edge: 0.25

Window(Filter) duration: 27

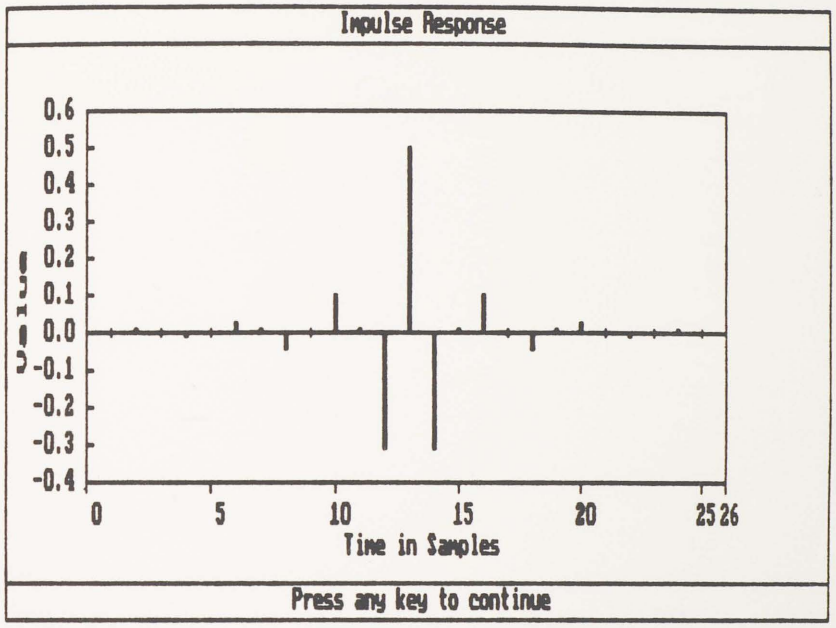
Design Results:

Impulse response coefficients:

```
h[000] =  -3.0694974703e-004  = h[026]
h[001] =  -6.2650516719e-011  = h[025]
h[002] =  +3.1565998215e-003  = h[024]
h[003] =  -1.6667220937e-009  = h[023]
h[004] =  -1.0011133738e-002  = h[022]
h[005] =  -5.4088311607e-009  = h[021]
h[006] =  +2.2736420855e-002  = h[020]
h[007] =  +7.7340900351e-010  = h[019]
h[008] =  -4.5641940087e-002  = h[018]
h[009] =  -1.1294429392e-008  = h[017]
h[010] =  +9.4529092312e-002  = h[016]
h[011] =  +1.3224819462e-008  = h[015]
h[012] =  -3.1431952119e-001  = h[014]
h[013] =  +5.0000000000e-001  = h[013]
```

Plots of impulse response and frequency response:

(See Figure B.1)



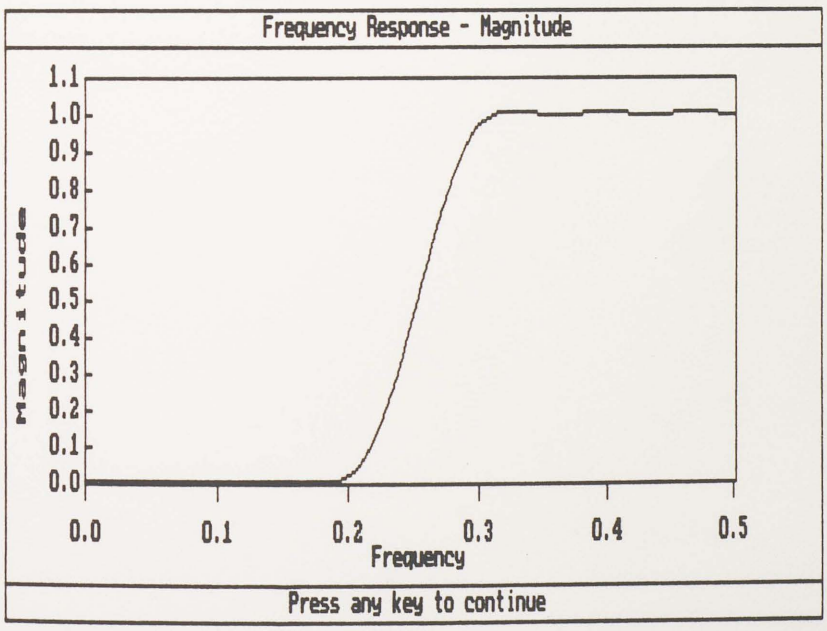


Figure B.1 Plots of Impulse Response and Frequency Response of Example 1.

Example 2

Design Specifications:

Design technique: Window

Filter type: Bandpass

Window type: Kaiser

Band edge: 0.18; 0.39

Window(Filter) duration: 44

ATT: 60 db

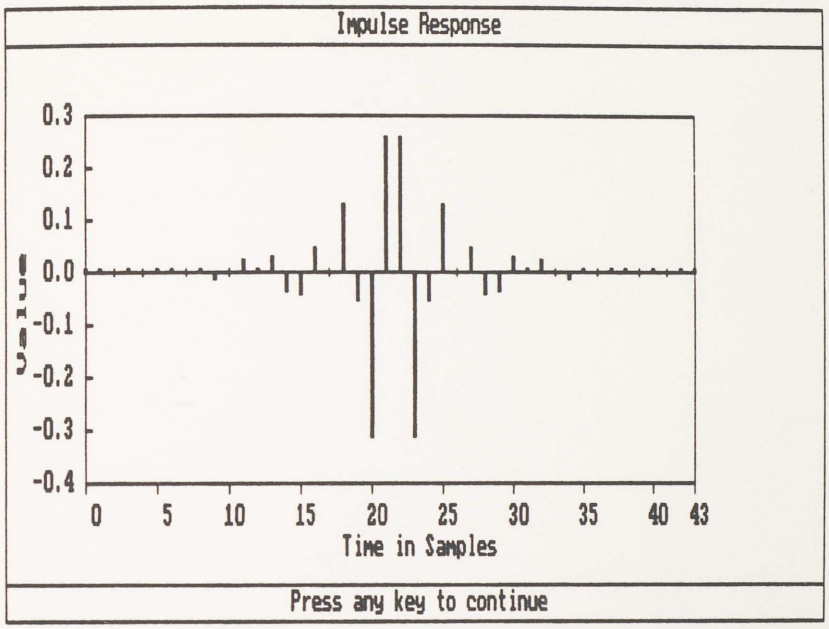
Design Results:

Impulse response coefficients:

h[000] =	+4.1965144919e-004	= h[043]
h[001] =	+5.3148821462e-004	= h[042]
h[002] =	-5.5035349214e-004	= h[041]
h[003] =	+1.5488504141e-004	= h[040]
h[004] =	-3.8841802161e-003	= h[039]
h[005] =	+1.8841509009e-003	= h[038]
h[006] =	+5.4995114915e-003	= h[037]
h[007] =	-1.1155875400e-003	= h[036]
h[008] =	+4.3848296627e-003	= h[035]
h[009] =	-1.6915790737e-002	= h[034]
h[010] =	-4.1784448549e-003	= h[033]
h[011] =	+1.9026627764e-002	= h[032]
h[012] =	+1.6443768982e-004	= h[031]
h[013] =	+2.7313204482e-002	= h[030]
h[014] =	-3.8893729448e-002	= h[029]
h[015] =	-4.2193900794e-002	= h[028]
h[016] =	+4.1637390852e-002	= h[027]
h[017] =	-4.4037029147e-003	= h[026]
h[018] =	+1.2563303113e-001	= h[025]
h[019] =	-5.7238914073e-002	= h[024]
h[020] =	-3.1460207701e-001	= h[023]
h[021] =	+2.5750818849e-001	= h[022]

Plots of Impulse response and frequency response:

(See Figure B.2)



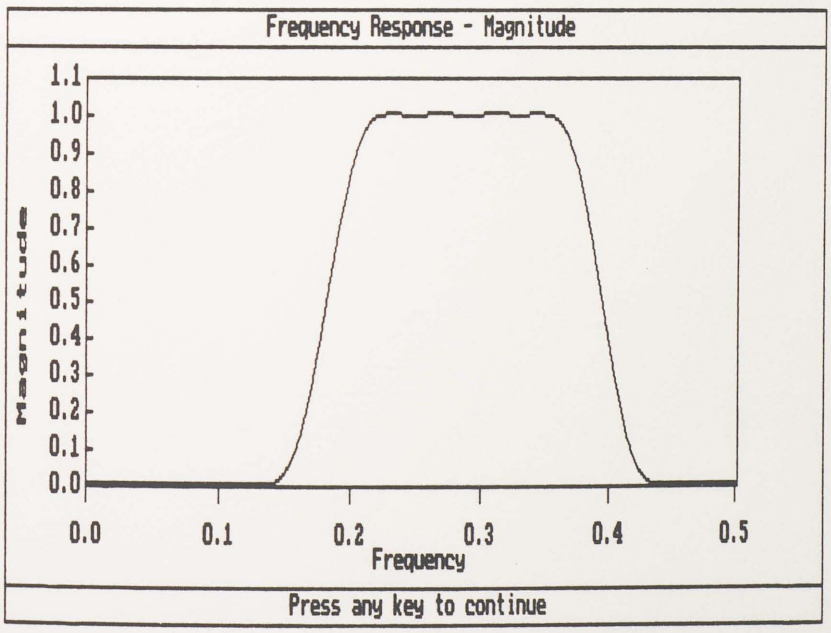


Figure B.2 Plots of Impulse Response and Frequency Response of Example 2.

Example 3

Design Specifications:

Design technique: Window

Filter type: Bandstop

Window type: Generalize Hamming

Band edge: 0.18; 0.39

Window(Filter) duration: 43

Alpha: 0.42

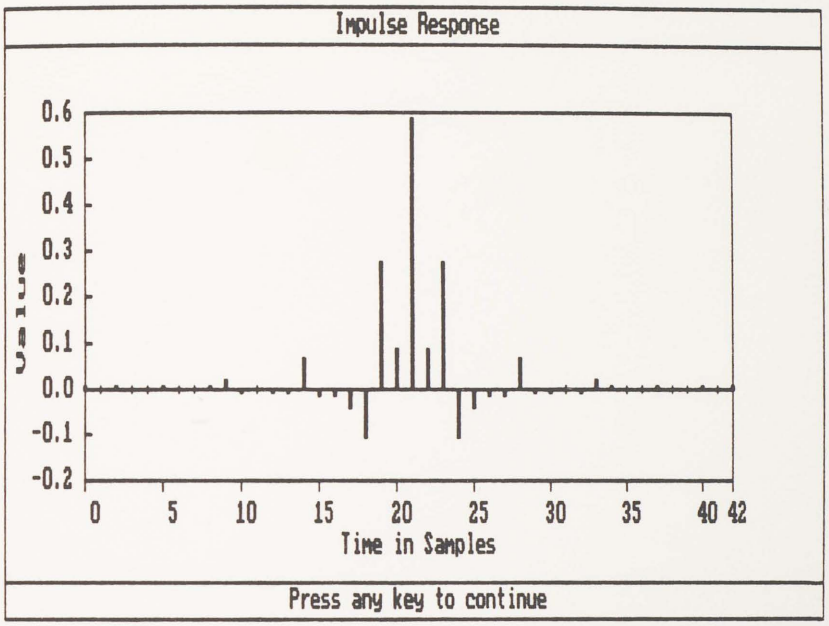
Design Results:

Impulse response coefficients:

h[000]	=	+4.6371710487e-003	=	h[042]
h[001]	=	-8.8761019288e-004	=	h[041]
h[002]	=	+1.2160648475e-004	=	h[040]
h[003]	=	-1.5828013420e-003	=	h[039]
h[004]	=	-1.2164711952e-003	=	h[038]
h[005]	=	+1.7306346854e-004	=	h[037]
h[006]	=	-1.7595439567e-004	=	h[036]
h[007]	=	-1.1055185460e-003	=	h[035]
h[008]	=	+2.1326965652e-003	=	h[034]
h[009]	=	+1.3498867862e-002	=	h[033]
h[010]	=	-1.1923047714e-002	=	h[032]
h[011]	=	-5.3577567451e-003	=	h[031]
h[012]	=	-1.2073969468e-002	=	h[030]
h[013]	=	-7.9556396231e-003	=	h[029]
h[014]	=	+6.4253143966e-002	=	h[028]
h[015]	=	-1.5034667216e-002	=	h[027]
h[016]	=	-1.4999148436e-002	=	h[026]
h[017]	=	-4.3947990984e-002	=	h[025]
h[018]	=	-1.1250970513e-001	=	h[024]
h[019]	=	+2.7177849412e-001	=	h[023]
h[020]	=	+8.4565639496e-002	=	h[022]
h[021]	=	+5.8000004292e-001	=	h[021]

Plots of impulse response and frequency response:

(See Figure B.3)



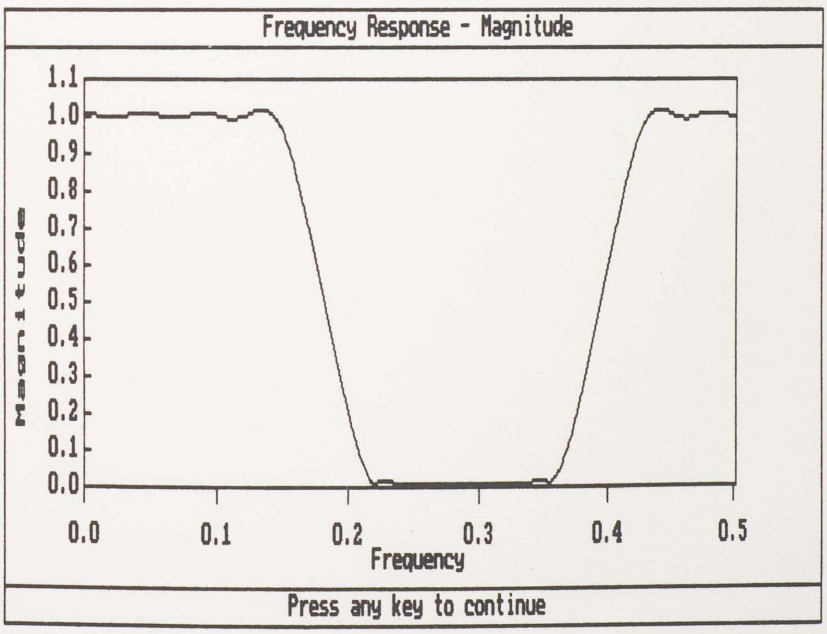


Figure B.3 Plots of Impulse Response and Frequency Response of Example 3.

Example 4

Design specifications:

Design technique: Frequency sampling

Filter type: Bandpass

Band edge: 0.18; 0.39

Filter duration: 44

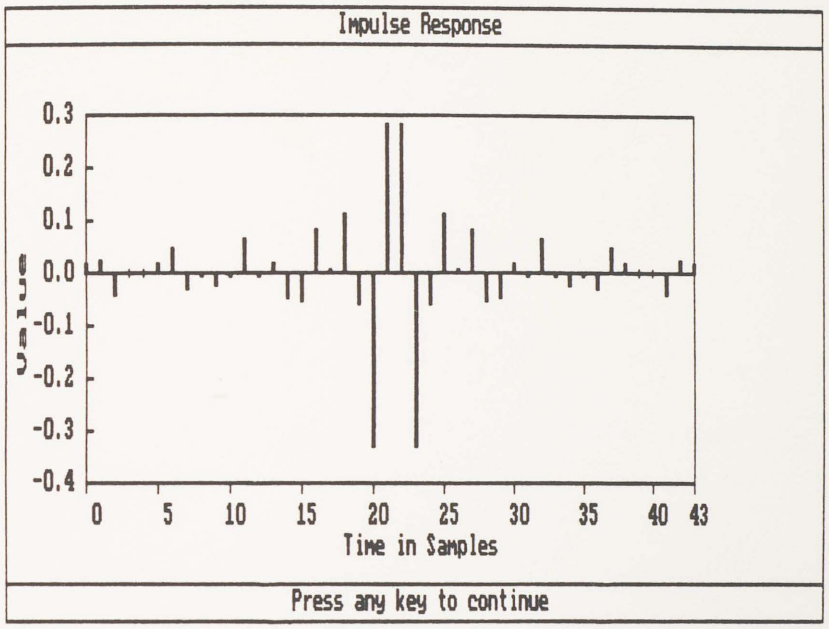
Design Results:

Impulse response coefficients:

h[000] =	+1.2376403436e-002	= h[043]
h[001] =	+1.7960371450e-002	= h[042]
h[002] =	-4.3730925769e-002	= h[041]
h[003] =	-1.0034228908e-003	= h[040]
h[004] =	-3.3632791601e-003	= h[039]
h[005] =	+1.3313328847e-002	= h[038]
h[006] =	+4.1647419333e-002	= h[037]
h[007] =	-3.0967338011e-002	= h[036]
h[008] =	-6.0021127574e-003	= h[035]
h[009] =	-2.6545291767e-002	= h[034]
h[010] =	-6.2201838009e-003	= h[033]
h[011] =	+6.2141895294e-002	= h[032]
h[012] =	-1.0963723063e-002	= h[031]
h[013] =	+1.4569840394e-002	= h[030]
h[014] =	-4.8593081534e-002	= h[029]
h[015] =	-5.7767454535e-002	= h[028]
h[016] =	+7.7595762908e-002	= h[027]
h[017] =	+1.8231627764e-003	= h[026]
h[018] =	+1.1007609963e-001	= h[025]
h[019] =	-6.1864886433e-002	= h[024]
h[020] =	-3.3373525739e-001	= h[023]
h[021] =	+2.7925264835e-001	= h[022]

Plots of impulse response and frequency response:

(See Figure B.4)



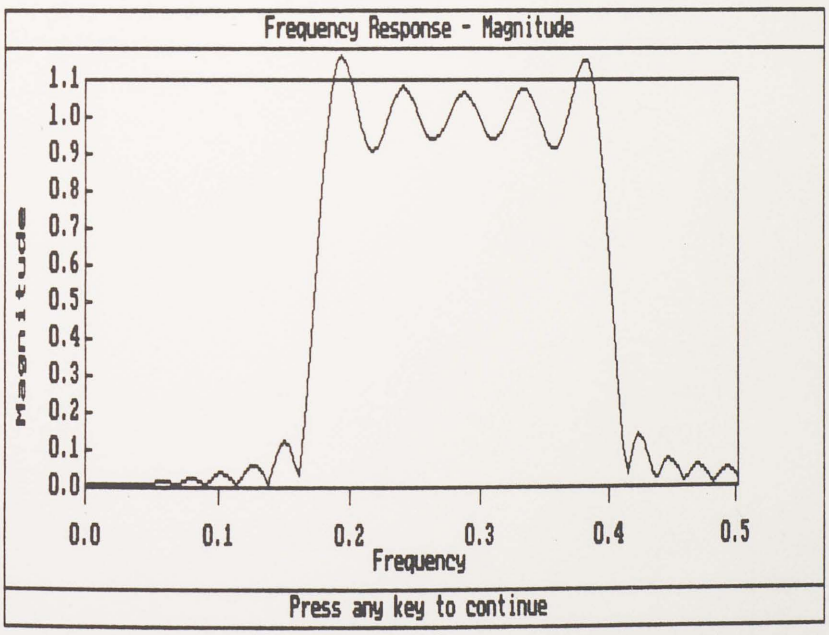


Figure B.4 Plots of Impulse Response and Frequency Response of Example 4.

Example 5

Design specifications:

Design technique: Frequency sampling

Filter type: Bandstop

Band edge: 0.18; 0.39

Filter duration: 43

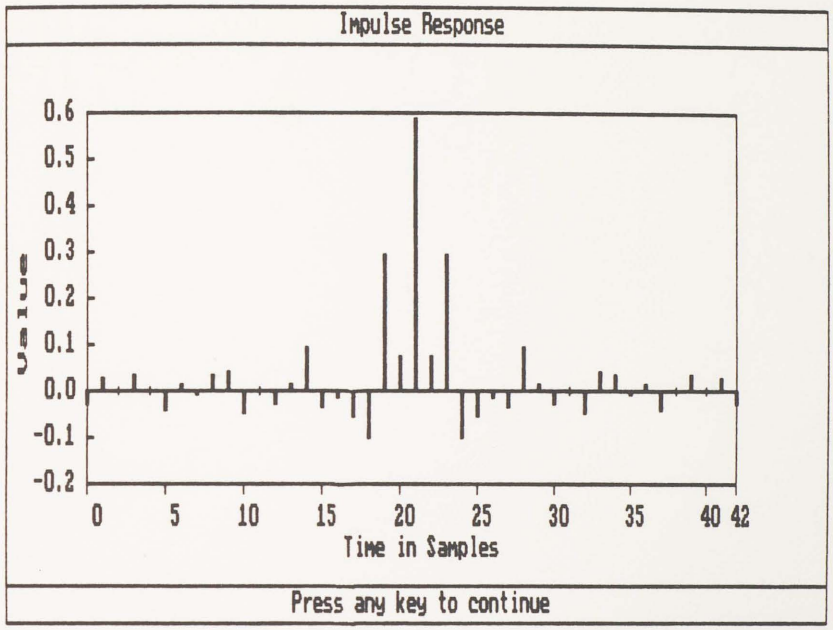
Design Results:

Impulse response coefficients:

h[000]	=	-2.8177499771e-002	=	h[042]
h[001]	=	+2.2515008226e-002	=	h[041]
h[002]	=	-1.1147640180e-003	=	h[040]
h[003]	=	+3.1739365309e-002	=	h[039]
h[004]	=	-1.7649733927e-003	=	h[038]
h[005]	=	-4.3869510293e-002	=	h[037]
h[006]	=	+8.6819864810e-003	=	h[036]
h[007]	=	-9.8808743060e-003	=	h[035]
h[008]	=	+3.0510483310e-002	=	h[034]
h[009]	=	+3.5185389221e-002	=	h[033]
h[010]	=	-4.7576904297e-002	=	h[032]
h[011]	=	-5.0789788365e-003	=	h[031]
h[012]	=	-2.7113515884e-002	=	h[030]
h[013]	=	+7.8697167337e-003	=	h[029]
h[014]	=	+9.0459242463e-002	=	h[028]
h[015]	=	-3.6070223898e-002	=	h[027]
h[016]	=	-1.5004671179e-002	=	h[026]
h[017]	=	-5.8846794069e-002	=	h[025]
h[018]	=	-1.0256068408e-001	=	h[024]
h[019]	=	+2.8866538405e-001	=	h[023]
h[020]	=	+7.0735134184e-002	=	h[022]
h[021]	=	+5.8139532804e-001	=	h[021]

Plots of impulse response and frequency response:

(See Figure B.5)



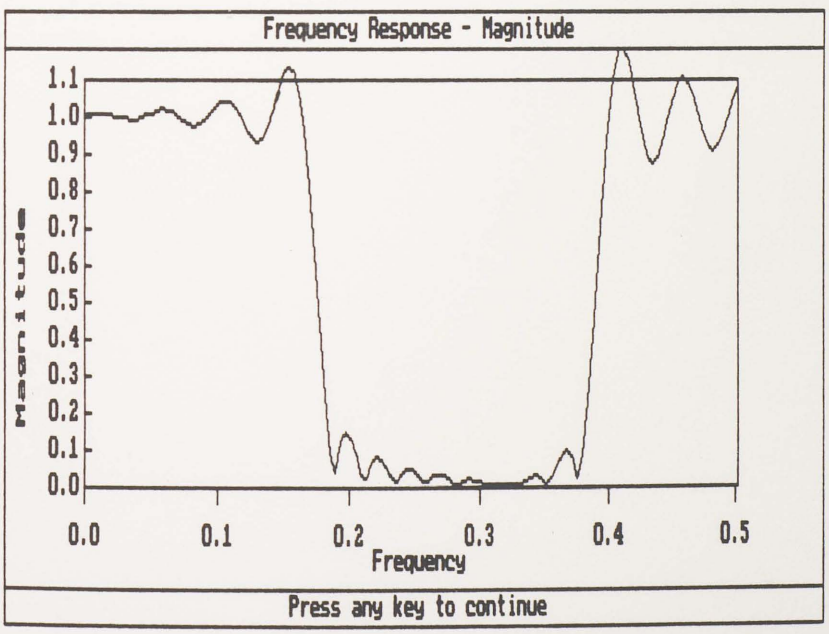


Figure B.5 Plots of Impulse Response and Frequency Response of Example 5.

Example 6

Design Specifications:

Design technique: Optimal

Filter bands: 3

Band edges/Weight: Band 1 [0.0, 0.1]/10

Band 2 [0.15, 0.35]/1

Band 3 [0.425, 0.5]/10

Filter duration: 32

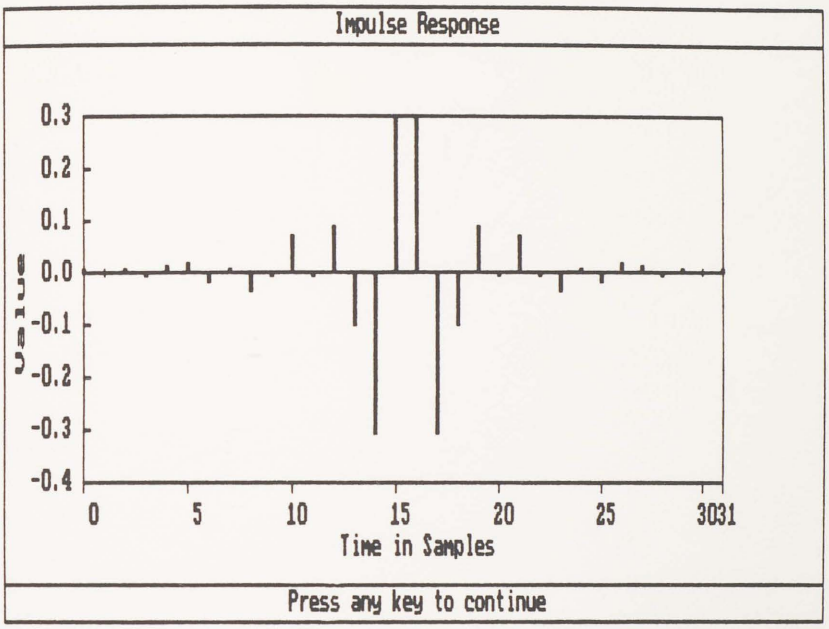
Design Results:

Impulse response coefficients:

h[000]	=	+2.3803768272e-005	= h[031]
h[001]	=	-4.2945374735e-003	= h[030]
h[002]	=	+4.4738319702e-003	= h[029]
h[003]	=	-6.2177525833e-003	= h[028]
h[004]	=	+9.4416113570e-003	= h[027]
h[005]	=	+1.4423212968e-002	= h[026]
h[006]	=	-1.8821891397e-002	= h[025]
h[007]	=	+5.3134290501e-003	= h[024]
h[008]	=	-3.7510462105e-002	= h[023]
h[009]	=	-6.9556869566e-003	= h[022]
h[010]	=	+7.0442378521e-002	= h[021]
h[011]	=	-6.5589807928e-003	= h[020]
h[012]	=	+8.7522782385e-002	= h[019]
h[013]	=	-1.0117103904e-001	= h[018]
h[014]	=	-3.0966731906e-001	= h[017]
h[015]	=	+2.9920503497e-001	= h[016]

Plots of Impulse response and frequency response:

(See Figure B.6)



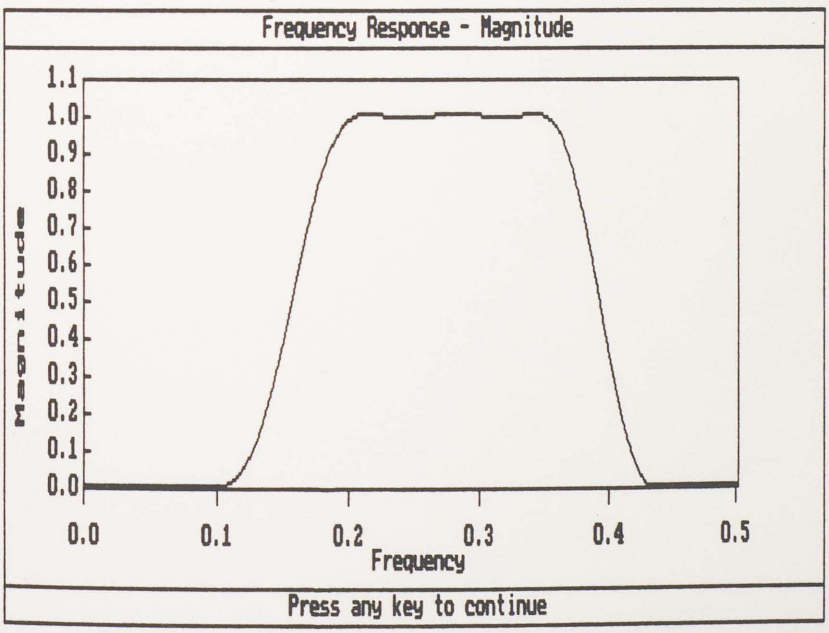


Figure B.6 Plots of Impulse Response and Frequency Response of Example 6.

Example 7

Design Specifications:

Design technique: Optimal

Filter bands: 2

Band edges/Weight: Band 1 [0.0, 0.33]/1

Band 2 [0.17, 0.5]/1

Filter duration: 20

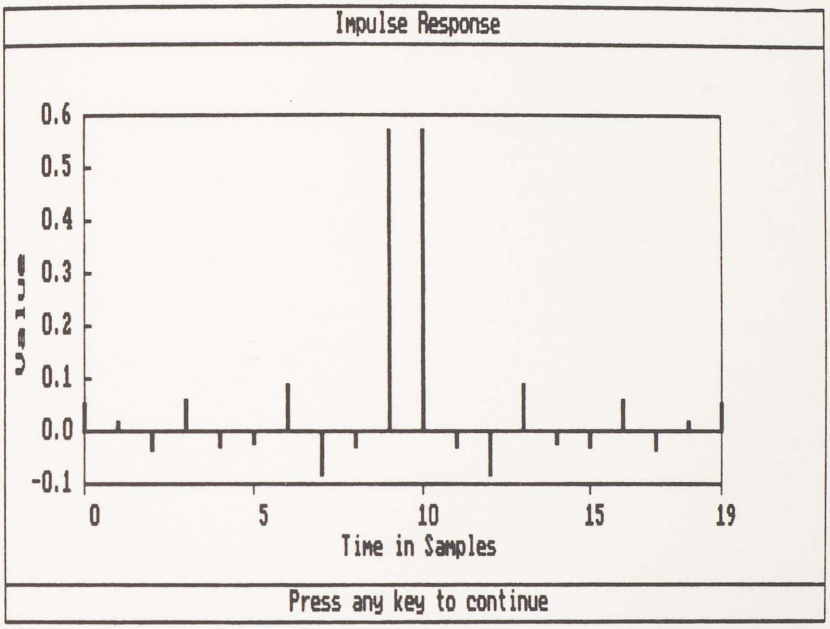
Design Results:

Impulse response coefficients:

h[000]	=	+4.8411231488e-002	= h[019]
h[001]	=	+1.3537425548e-002	= h[018]
h[002]	=	-3.9344053715e-002	= h[017]
h[003]	=	+5.3151816130e-002	= h[016]
h[004]	=	-3.1608257443e-002	= h[015]
h[005]	=	-2.5162726641e-002	= h[014]
h[006]	=	+8.3330653608e-002	= h[013]
h[007]	=	-8.6372196674e-002	= h[012]
h[008]	=	-3.4074477851e-002	= h[011]
h[009]	=	+5.6718868017e-001	= h[010]

Plots of impulse response and frequency response:

(See Figure B.7)



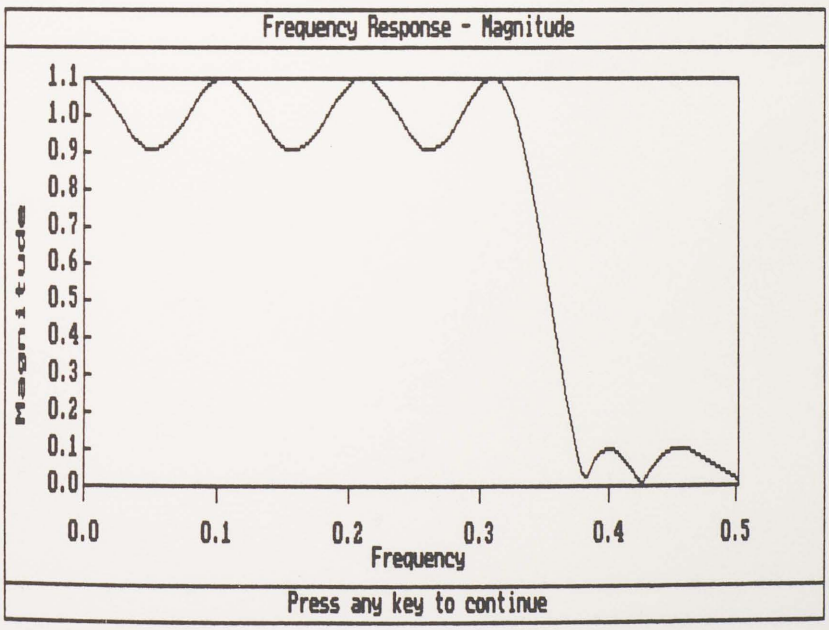


Figure B.7 Plots of Impulse Response and Frequency Response of Example 7.

Example 8

Design Specifications:

Design technique: Optimal

Filter bands: 2

Band edges/Weight: Band 1 [0.0, 0.17]/1

Band 2 [0.21, 0.5]/1

Filter duration: 37

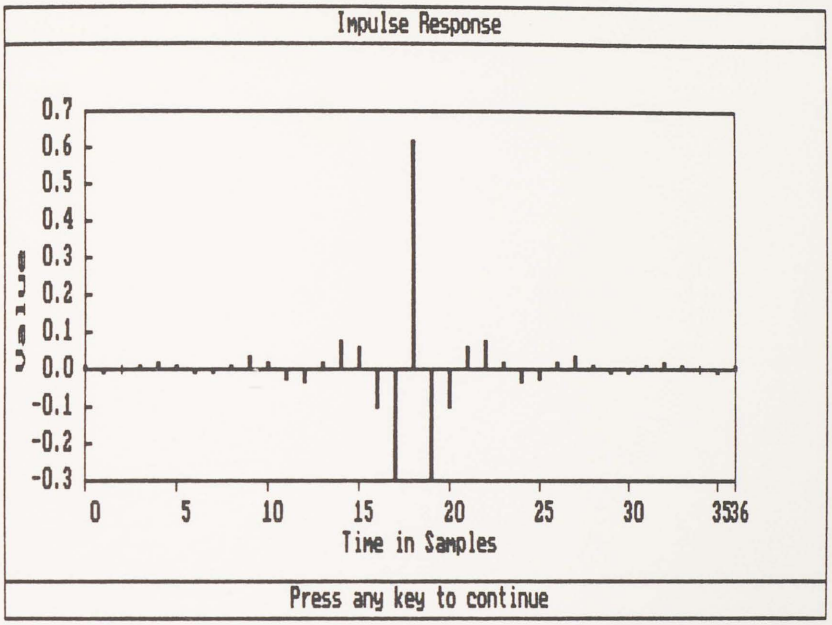
Design Results:

Impulse response coefficients:

h[000] =	+1.9141206285e-003	= h[036]
h[001] =	-1.0486555286e-002	= h[035]
h[002] =	-6.1963154003e-003	= h[034]
h[003] =	+4.1637071408e-003	= h[033]
h[004] =	+1.1371149682e-002	= h[032]
h[005] =	+3.0356489588e-003	= h[031]
h[006] =	-1.3734713197e-002	= h[030]
h[007] =	-1.5283850022e-002	= h[029]
h[008] =	+7.0165814832e-003	= h[028]
h[009] =	+2.7112836018e-002	= h[027]
h[010] =	+1.1950018816e-002	= h[026]
h[011] =	-2.9090648517e-002	= h[025]
h[012] =	-4.1451111436e-002	= h[024]
h[013] =	+9.1355592012e-003	= h[023]
h[014] =	+7.4285961688e-002	= h[022]
h[015] =	+5.2547756582e-002	= h[021]
h[016] =	-1.0012545437e-001	= h[020]
h[017] =	-2.9856505990e-001	= h[019]
h[018] =	+6.0993951559e-001	= h[018]

Plots of impulse response and frequency response:

(See Figure B.8)



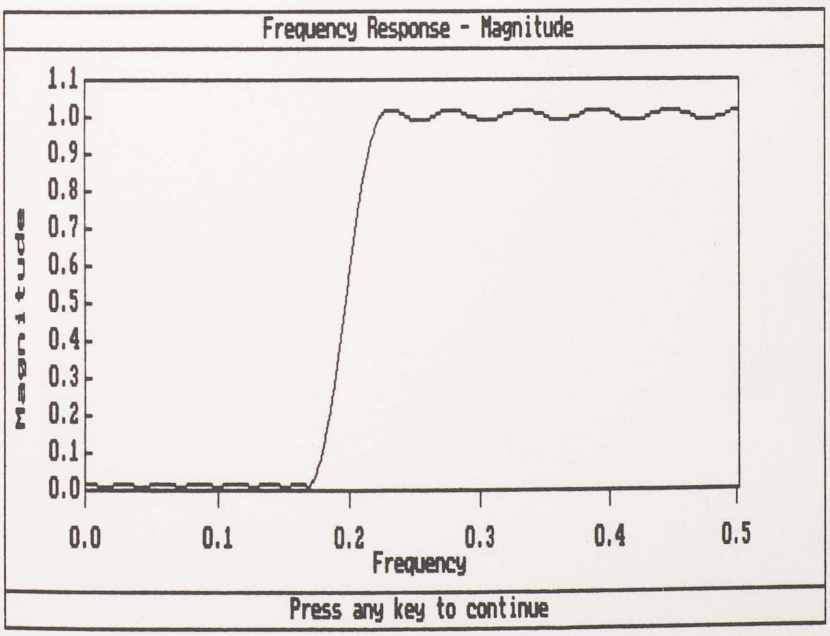


Figure B.8 Plots of Impulse Response and Frequency Response of Example 8.

Example 9

Design Specifications:

Design technique: Optimal

Filter bands: 3

Band edges/Weight: Band 1 [0.0, 0.1]/10

Band 2 [0.14, 0.37]/1

Band 3 [0.42, 0.5]/10

Filter duration: 33

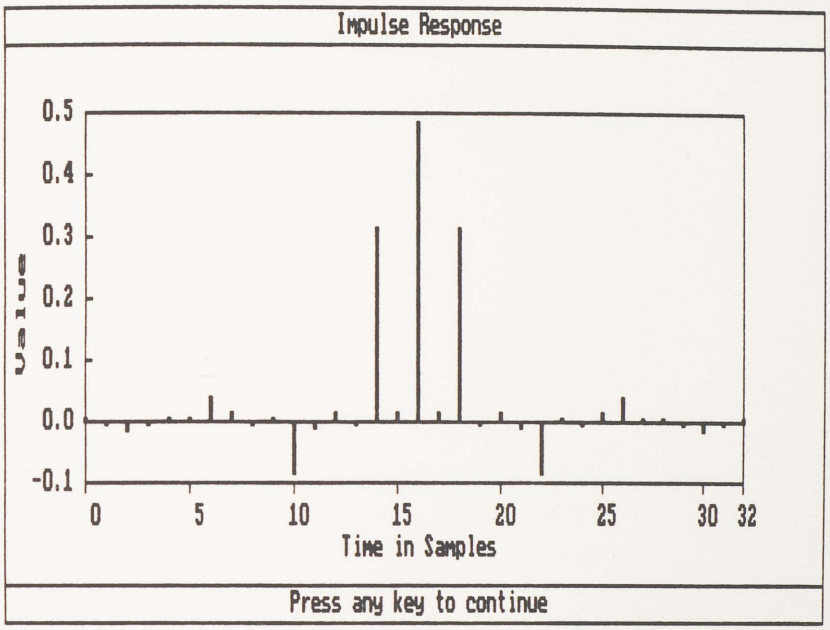
Design Results:

Impulse response coefficients:

h[000] =	+4.1235890239e-003	= h[032]
h[001] =	-8.8336439803e-003	= h[031]
h[002] =	-1.5807975084e-002	= h[030]
h[003] =	-9.2285722494e-003	= h[029]
h[004] =	+4.6741366386e-003	= h[028]
h[005] =	+1.8658206100e-003	= h[027]
h[006] =	+3.5324033350e-002	= h[026]
h[007] =	+1.2315007858e-002	= h[025]
h[008] =	-9.6339331940e-003	= h[024]
h[009] =	+2.0628520288e-003	= h[023]
h[010] =	-8.5898689926e-002	= h[022]
h[011] =	-1.4480249025e-002	= h[021]
h[012] =	+1.4566914178e-002	= h[020]
h[013] =	-7.9581877217e-003	= h[019]
h[014] =	+3.1094589829e-001	= h[018]
h[015] =	+1.2898549438e-002	= h[017]
h[016] =	+4.8341211677e-001	= h[016]

Plots of impulse response and frequency response:

(See Figure B.9)



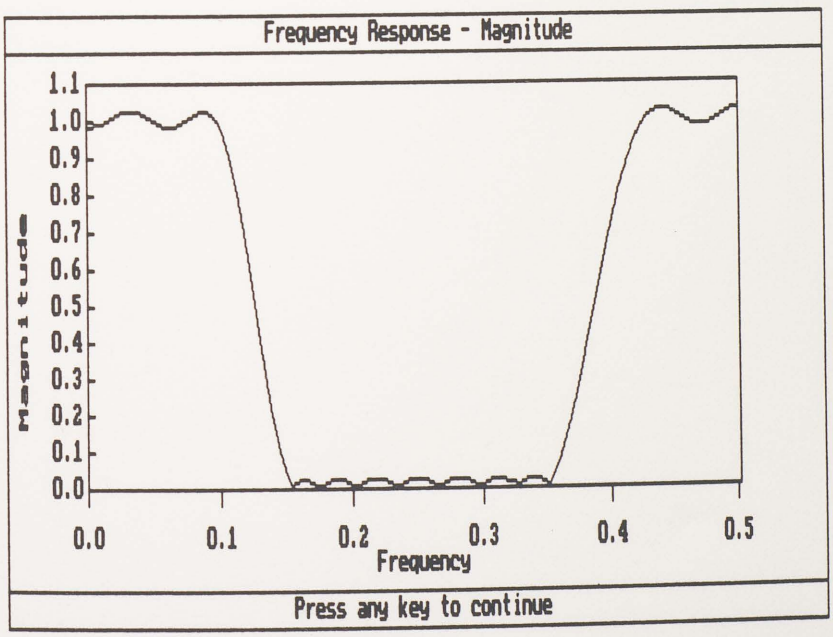


Figure B.9 Plots of Impulse Response and Frequency Response of Example 9.

APPENDIX C

CADFIR PACKAGE SOURCE LISTING

```
/*************************************************************************/
/*                                                                       */
/*  Computer-Aided Design of Linear Phase FIR Digital Filters            */
/*  By WENTAO LYOU, for Research Project towards M.S. in                 */
/*  Engineering, Summer Semester, 1988                                   */
/*                                                                       */
/*************************************************************************/

/*                                                                   */
/************************ MAIN FILE ****************************/
/*                                                                   */

/* INCLUDE FILES */
#include <window.h1>
#include <stdlib.h>
#include <stdio.h>
#include <stat.h>
#include <dir.h>
#include <dos.h>
#include <process.h>
#include <tcdef.h>
#include <key.h>
#include <video.h>
#include <window.h>
#include <print.h>

/* DATA TYPE DEFINITION */
typedef struct _fname {
            char name[15];
            struct _fname *nxt;
            struct _fname *pre;
        } fname;

/* DEFINES */
#define cell "0.0000 "
#define Dcell "0 "
#define Wcell "1 "

/* FUNCTION PROTOTYPES */
void  Initiation(void);
void  Mainmenu(void);
void  sampinput(void);
void  wininput(void);
void  optinput(void);
int   getkey();
char  *getdesigname();
char  *current_dir(char *path);
fname *getdir(char *mask);
void  listfilename(fname *fn, int direct);
fname *pickfile(char *mask);
void  listfile(char *filename);
```

```
int   ptrready();
void  printfile(char *filename);
void  statusline(char *str);
int   exit_confirm();
void  prog_exit();

/* GLOBAL  VARIABLES */
int   i,j, row, col;
int   whandle[6];
int   filenum;
int   *sbuf;
int   status;
char  curdir[MAXPATH];
char  *OLDPATH, *NEWPATH="PATH=";

/* MAIN ROUTINE */
main()
{
    Initiation();
    Mainmenu();
}

/* TITLE PAGE DISPLAY ROUTINE */
void Initiation()
{
    int pitch;

    videoinit();
    sbuf=ssave();
    readcur(&row,&col);
    clrscr();
    cursoroff();
    setcbrk(ON);
    pitch = (machid()==IBMPCAT ? 15000 : 10000);

    for (i=0;i<6;i++) {
        whandle[i]=wopen(i,i,i+19,i+74,3,LCYAN|_BLUE);
        if (!whandle[i]) return;
            sound_(pitch-((i+1)*1500),2);
    }

    wprints(3,30,LRED|_BLUE,">>>> CADFIR <<<<");
    wprints(5,27,YELLOW|_BLUE,"Computer-Aided Design of");
    wprints(6,22,YELLOW|_BLUE,"FIR Linear-Phase Digital Filters");
    wprints(9,36,WHITE|_BLUE,"by");
    wprints(10,32,LGREEN|_BLUE,"Wentao Lyou");
    wprints(11,30,LMAGENTA|_BLUE,"EECS Department");
    wprints(12,25,LTGRAY|_BLUE,"University of Central Florida");
    wprints(16,28,LCYAN|_BLUE|BLINK,"Press any key to begin");
    while (!kbhit());
    clearkeys();
```

```
    wprints(13,28,_BLUE,"                          ");

    for (i=0;i<10;i++) {
        wscroll(1,UP);
        delay_(1);
    }
    for (i=5;i>=0;i--)
        wclose();
    current_dir(curdir);
}

/* MAIN MENU MANIPULATE ROUTINE */
void Mainmenu()
{

    int   *sbuff;
    char  ch, *ptype="  ";
    char  path[MAXPATH];
    fname *fn;

    initwindows(DOUBLE,BLUE,LCYAN,BLUE,LCYAN);
    clrscr();
    cursoroff();
    scrncolor(LTGRAY,BLACK);
    makewindow(1,1,1,3,80,"[ MAIN MENU ]",NOSHADOW);
    initchoicesh(1,"Create,Plot,List,Print,System,Exit");
    initwindows(SINGLE,BLUE,LCYAN,BLUE,LCYAN);
    statusline("<- or -> to Select a command");
    while (1) {
        ch = getchoice(1);
        switch(ch) {
            case 1 : makewindow(2,4,1,9,18,"Design Methods", SHADOW);
                     initchoicesv(2,"FREQ. SAMPLING,WINDOWING,
                                  OPTIMAL,QUIT");
                     statusline("UP or DOWN to select a filter
                                 design method");
                     ch=getchoice(2);
                     switch (ch) {
                         case 1  : sampinput();
                                   break;
                         case 2  : wininput();
                                   break;
                         case 3  : optinput();
                         default : break;
                     }
                     wclose();
                     removewindow(2);
                     break;
            case 2 : makewindow(2,4,18,9,36,"Graph Types",SHADOW);
                     initchoicesv(2,"IMPULSE RESP., FREQ. RESP.
                                   (MAG),FREQ. RESP.(PHS),QUIT");
```

```
            statusline("UP or DOWN to select a graph
                        type");
            ptype=itoa(getchoice(2),ptype,10);
            switch (*ptype) {
                case '1' : fn=pickfile("*.imp");
                           break;
                case '2' :
                case '3' : fn=pickfile("*.frq");
                default  : break;
            }
            if (fn!=NULL && *ptype<='3') {
                strcpy(path,curdir);
                strcat(path,"plot.exe");
                status=spawnl(P_WAIT,path,path,ptype,
                              fn->name,NULL);
            }
            cursoroff();
            removewindow(2);
            break;
case 3 :    makewindow(2,4,31,9,45,"File Types",SHADOW);
            initchoicesv(2,"IMPULSE RESP.,
                         FREQ. RESP.,DFT COEFFS.,QUIT");
            statusline("UP or DOWN to select a file
                        type");
            ch=getchoice(2);
            switch (ch) {
                case 1 : fn=pickfile("*.imp");
                         break;
                case 2 : fn=pickfile("*.frq");
                         break;
                case 3 : fn=pickfile("*.dft");
                case 4 : break;
            }
            if (fn!=NULL && ch<=3)
                listfile(fn->name);
            removewindow(2);
            break;
case 4 :    makewindow(2,4,44,9,58,"File Types",SHADOW);
            initchoicesv(2,"IMPULSE RESP.,
                         FREQ. RESP.,DFT COEFFS.,QUIT");
            statusline("UP or DOWN to select file type");
            ch=getchoice(2);
            switch (ch) {
                case 1 : fn=pickfile("*.imp");
                         break;
                case 2 : fn=pickfile("*.frq");
                         break;
                case 3 : fn=pickfile("*.dft");
                case 4 : break;
            }
            if (fn!=NULL && ch<=3)
```

```
                    printfile(fn->name);
                removewindow(2);
                break;
        case 5 :  sbuff=ssave();
                cursoron();
                system("CLS");
                printf("Type EXIT to return CADFIR");
                system("COMMAND.COM");
                srestore(sbuff);
                cursoroff();
                break;
         case 6 : if (exit_confirm())
                    prog_exit();
         default: break;
      }
   statusline("<- or -> to select a command");
   }
}

/* FREQUENCY SAMPLING DESIGN INPUT ROUTINE */
void sampinput()
{
   char *ft="  ";
   char *loweg="0.000000";
   char *upeg="0.000000";
   char *filen="3    ";
   char *desname;
   char path[MAXPATH];

   /* Get Filter Type */
   makewindow(3,5,2,10,19,"Filter Types",SHADOW);
   initchoicesv(3,"Low Pass,High Pass,Band Pass,Band Stop");
   statusline("UP or DOWN to select a filter type");
   ft=itoa(getchoice(3),ft,10);

   /* Get filter parameters */
   do {
      whandle[0]=wopen(10,20,20,79,1,LCYAN|_BLUE);
      if (!whandle[0])
         return;
      wtextattr(LMAGENTA|_BLUE);
      statusline("UP|DOWN|TAB/Swap between cells    ESC/Abort");
      wprints(2,2,LCYAN|_BLUE,"Filter Length [3,128]");
      if (*ft=='2'||*ft=='4')
         wprints(2,38,YELLOW|_RED,"Odd Length Only");
      if (winpdef(2,30,filen,'#')) return;
      wprints(5,2,LCYAN|_BLUE,"Band Edge [0.0,0.5]");
      if (winpdef(5,30,loweg,'9')) return;
      if (*ft>'2') {
         wprints(4,30,LCYAN|_BLUE,"Lower");
         wprints(4,40,LCYAN|_BLUE,"Upper");
```

```
            if (winpdef(5,40,upeg,'9')) return;
        }
        wtextattr(LGREEN|_BLUE);
        cursoron();
        i = winpread();
        cursoroff();
        if (i) return;  /* ESC pressed */
        desname=getdesigname();
        strcpy(path,curdir);
        strcat(path,"fresamp.exe");
        status=spawnl(P_WAIT,path,path,ft,filen,loweg,upeg,desname,
                      NULL);
        wclose();
    } while(status==100);
}

/* WINDOW FILTER DESIGN INPUT ROUTINE */
void wininput()
{
    char *ft="  ";
    char *wt="  ";
    char *loweg="0.000000";
    char *upeg="0.000000";
    char *filen="3    ";
    char *wpar="0.00000";
    char *desname;
    char path[MAXPATH];

    /* Get Filter Type */
    makewindow(3,5,2,10,19,"Filter Types",SHADOW);
    initchoicesv(3,"Low Pass,High Pass,Band Pass,Band Stop");
    statusline("UP or DOWN to select a filter type");
    ft=itoa(getchoice(3),ft,10);

    /* Get Window Type */
    makewindow(4,6,3,13,20,"Window Types",SHADOW);
    initchoicesv(4,"Rectangular,Bartlett,Hamming,Gen. Hamming,
                 Hanning,Kaiser");
    statusline("UP or DOWN to select a window type");
    wt=itoa(getchoice(4),wt, 10);

    /* Get filter\window parameters */
    do {
        whandle[0]=wopen(10,20,20,79,1,LCYAN|_BLUE);
        if (!whandle[0])
            return;
        wtextattr(LMAGENTA|_BLUE);
        statusline("UP|DOWN|TAB/Swap between cells    ESC/Abort");
        wprints(2,2,LCYAN|_BLUE,"Filter Length [3,128]");
        if (*ft=='2'||*ft=='4')
            wprints(2,38,YELLOW|_RED,"Odd Length Only");
```

```
        if (winpdef(2,30,filen,'#')) return;
        wprints(5,2,LCYAN|_BLUE,"Band Edge [0.0,0.5]");
        if (winpdef(5,30,loweg,'9')) return;
        if (*ft>'2') {
            wprints(4,30,LCYAN|_BLUE,"Lower");
            wprints(4,40,LCYAN|_BLUE,"Upper");
            if (winpdef(5,40,upeg,'9')) return;
        }
        if (*wt=='4') {
            wprints(7,2,LCYAN|_BLUE,"Alpha[0.0,1.0]");
            wpar="0.000000";
            if (winpdef(7,30,wpar,'9')) return;
        }
        if (*wt=='6') {
            wprints(7,2,LCYAN|_BLUE,"Attenuation(db)");
            wpar="0       ";
            if (winpdef(7,30,wpar,'9')) return;
        }
        wtextattr(LGREEN|_BLUE);
        cursoron();
        i = winpread();
        cursoroff();
        if (i) return;  /* ESC pressed */
        desname=getdesigname();
        strcpy(path,curdir);
        strcat(path,"window.exe");
        status=spawnl(P_WAIT,path,path,ft,wt,
                      filen,loweg,upeg,wpar,desname,NULL);
        wclose();
    } while(status==100);
}

/* OPTIMAL FILTER DESIGN INPUT ROUTINE */
void optinput()
{
    char *Nband="   ";
    char *filen="3    ";
    char *loweg[10]={cell,cell,cell,cell,cell,cell,cell,cell,
                     cell,cell};
    char *upeg[10]={cell,cell,cell,cell,cell,cell,cell,cell,
                    cell,cell};
    char *des[10]={Dcell,Dcell,Dcell,Dcell,Dcell,Dcell,Dcell,
                   Dcell,Dcell,Dcell};
    char *wt[10]={Wcell,Wcell,Wcell,Wcell,Wcell,Wcell,Wcell,
                  Wcell,Wcell,Wcell};
    char *desname;
    char path[MAXPATH];
    int  handle;
    FILE *fp;

    makewindow(4,6,3,8,25,"No. of Bands",SHADOW);
```

```
    initchoicesh(4,"2,3,4,5,6,7,8,9,10");
    statusline("<- or -> to choose the number of bands");
    Nband=itoa(getchoice(4)+1,Nband,10);

    do {
        whandle[0]=wopen(10-atoi(Nband),10,13+atoi(Nband),79,1,
                         LCYAN|_BLUE);
        if (!whandle[0])
            return;
        wtextattr(LMAGENTA|_BLUE);
        statusline("UP|DOWN|TAB/Swap between cells   ESC/Abort");
        wprints(1,2,LCYAN|_BLUE,"Filter Length [3,128]");
        if (winpdef(1,26,filen,'#')) return;
        wprints(2,2,LCYAN|_BLUE,"Band No. Lower Edge   Upper Edge
                Desired Value   Weight");
        for (i=1; i<=atoi(Nband); i++) {
            wgotoxy(2*i+1,1);
            wprintf("%4d",i);
            if (winpdef(2*i+1,14,loweg[i-1],'9')) return;
            if (winpdef(2*i+1,27,upeg[i-1],'9')) return;
            if (winpdef(2*i+1,43,des[i-1],'#')) return;
            if (winpdef(2*i+1,55,wt[i-1],'#')) return;
        }
        wtextattr(LGREEN|_BLUE);
        cursoron();
        i = winpread();
        cursoroff();
        if (i) return;  /* ESC pressed */
        if (!fexist("tmp")) {
            handle = creat("tmp", S_IREAD|S_IWRITE);
            fp = fdopen(handle, "wt");
        }
        else
            fp =fopen("tmp","wt");
        for (i=0;i<atoi(Nband);i++)
            fprintf(fp,"%d  %f  %f  %f  %f\n", i, atof(loweg[i]),
                    atof(upeg[i]),atof(des[i]), atof(wt[i]));
        fclose(fp);
        desname=getdesigname();
        strcpy(path,curdir);
        strcat(path,"optimal.exe");
        status=spawnl(P_WAIT,path,path,Nband,filen,desname,NULL);
        wclose();
    } while(status==100);
}

/* PROMPT USER ENTER DESIGN NAME */
char *getdesigname()
{
    char *dn="           ";
    char *wn="           ";
```

```
    char ans;
    int  filexist;

    whandle[1]=wopen(10,18,12,62,1,WHITE|_GREEN);
    if (!whandle[0])
        return(NULL);
    cursoron();
    statusline("Enter the name of this design");
    while(1) {
        clrwin(11,19,11,59);
        wgotoxy(0,0);
        winputsf(dn,"' Enter Design Name : '!RE! X !R! XXXXXXX");
        strcpy(wn,dn);
        strcat(wn,".*");
        filexist=fexist(wn);
        if (filexist==1) {
            beep();
            wgotoxy(0,34);
            wprints(0,2,WHITE|_GREEN,"Duplicated Name,Overwrite(Y/N)?");
            ans = wgetchf("YyNn");
            if (ans=='y'||ans=='Y')
                break;
        }
        else
            break;
    }
    wclose();
    cursoroff();
    return(dn);
}

/* GET A PRESSED KEY */
int getkey()
{
    int c;
    static union REGS rg;

    while(1) {
        rg.h.ah = 1;
        int86(0x16, &rg, &rg);
        if (rg.x.flags & 0x40)  {
            int86(0x28, &rg, &rg);
            continue;
        }
        rg.h.ah =0;
        int86(0x16, &rg, &rg);
        if (rg.h.al==0)
            c = rg.h.ah|256;
        else
            c = rg.h.al;
    break;
```

```
    }
    return c;
}

/* GET CURRENT DIRECTORY */
char *current_dir(char *path)
{
    strcpy(path, "X:\\");
    path[0] = 'A'+getdisk();
    getcurdir(0,path+3);
    if (path[3]!='')
       strcpy(path+strlen(path),"\\");
    return(path);
}

/* GET DIRECTORY FILE NAMES */
fname *getdir(char *mask)
{
    short  done;
    struct ffblk ffb;
    fname  *fn, *pre, *ret;

    done = findfirst(mask,&ffb,0);
    if (done)
        return(NULL);
    fn = ret = (fname *)malloc(sizeof(fname));
    fn->pre = NULL;
    filenum = 0;
    while (!done) {
        strcpy(fn->name,ffb.ff_name);
        pre = fn;
        fn = (fname *)malloc(sizeof(fname));
        fn->pre = pre;
        pre->nxt = fn;
        done = findnext(&ffb);
        filenum += 1;
    };
    free(fn);
    pre->nxt = NULL;
    return (ret);
}

/* LIST DIRECTORY FILE NAMES */
void listfilename(fname *fn, int direct)
{
    int i;

    clrwin(6,31,15,46);

    if (direct==UP) {
        for (i=9;i>=0;i--) {
```

```
        wprints(i,1,YELLOW|_BLUE,fn->name);
        fn=fn->pre;
        if (!fn) return;
    }
    }

    if (direct==DOWN) {
        for (i=0;i<10;i++) {
        wprints(i,1,YELLOW|_BLUE,fn->name);
        fn=fn->nxt;
        if (!fn) return;
    }
    }
}

/* FILE PICK OUT SUBMENU */
fname *pickfile(char *mask)
{
    fname *fn, *dfn;
    int   i, sel, pos=0, fsel=0;

    fn=getdir(mask);
    if (fn==NULL) {
    wopen(11,25,14,55,2,WHITE|_RED);
        wprints(0,7,WHITE|_RED,"No exist design");
    wprints(1,4,YELLOW|_RED|BLINK,"Press a key to return");
    waitkey();
    return(NULL);
    }

    statusline("UP or DOWM/Select file      ESC/Abort");
    wopen(5,30,16,47,2,YELLOW|_BLUE);
    listfilename(fn,DOWN);
    wprints(0,1,WHITE|_RED,fn->name);
    while(1) {
        sel=getkey();
        wprints(pos,1,YELLOW|_BLUE,fn->name);
        switch (sel) {
            case UPKEY  : if (fsel>0) {
                              fn=fn->pre;
                  if (pos!=0)
                  pos-=1;
                              fsel -= 1;
                              if (pos==0)
                  listfilename(fn,DOWN);
                      wprints(pos,1,WHITE|_RED,fn->name);
                          }
              wprints(pos,1,WHITE|_RED,fn->name);
                          break;
        case DOWNKEY  : if (fsel<filenum-1) {
                             fn = fn->nxt;
```

```
                if (pos!=9)
                    pos+=1;
                            fsel += 1;
                if (pos==9)
                    listfilename(fn,UP);
                wprints(pos,1,WHITE|_RED,fn->name);
                        }
                 else
                            wprints(pos,1,WHITE|_RED,fn->name);
                          break;
            case CR       : wclose();
                    return(fn);
        case ESC      : wclose();
                    return(NULL);
            default       : wprints(pos,1,WHITE|_RED,fn->name);
                    break;
            }
        }
}

/* LIST A FILE ON SCREEN */
void listfile(char *filename)
{
    char path[MAXPATH];

    if (filename==NULL)
    return;
    strcpy(path,curdir);
    strcat(path,filename);
    statusline("Press any key to progess");
    initwindows(SINGLE,LTGRAY,BLACK,LTGRAY,BLACK);
    makewindow(3,4,1,24,80,path,NOSHADOW);
    filetowindow(3,filename,1,1);
    windwrite(3,"--Press a key to return--",19,50);
    waitkey();
    initwindows(SINGLE,BLUE,LCYAN,BLUE,LCYAN);
}

/* TEST FOR PRINT READY */
int ptrready()
{
    int i, ptr, prtnum;
    prtnum=((biosequip()|0xc000)>>14);
    for (i=0;i<=prtnum;i++) {
    if (biosprint(2,0,i)==0x90) {
        ptr=biosprint(1,0,i);
            return(ptr);
    }
    }
    return(0);
}
```

```
/* PRINT A FILE */
void printfile(char *filename)
{
    int  linenum=0, page=1;
    FILE *file;
    char *Page="   ", line[80];

    if (filename==NULL)
    return;

    if (!ptrready()) {
    beep();
    statusline("!!! Please get the printer ready,
               then press a key !!!");
    wopen(11,25,14,55,2,WHITE|_RED);
        wprints(0,6,WHITE|_RED,"Printer not ready");
    wprints(1,2,YELLOW|_RED|BLINK,"Press a key to try again");
        waitkey();
    if (!ptrready()) {
       beep();
       return;
    }
    wclose();
    }

    file = fopen(filename,"r");
    lcrlf();
    lprints(filename);
    lprints("  Page ");
    lprintc('1');
    lcrlf(); lcrlf();

    wopen(12,25,14,55,2,WHITE|_GREEN);
    wprints(0,9,WHITE|_GREEN|BLINK,"Printing....");
    while (getc(file)!=EOF) {
         linenum += 1;
     if (linenum%54==0){
            linenum = 0;
        lprintc(FF);
            page += 1;
            Page = itoa(page,Page,10);
        lcrlf();
        lprints(filename);
        lprints("  Page  ");
        lprints(Page);
        lcrlf(); lcrlf();
     }
         fgets(line,80,file);
     lprints(line);
    }
```

```
    lprintc(FF);
    fclose(file);
    wclose();
}

/* DISPLAY STATUS LINE */
void statusline(char *str)
{
    int halfstrlen;

    fill(24,0,24,79,' ',WHITE|_RED);
    halfstrlen=strlen(str)/2;
    printsd(24,40-halfstrlen,WHITE|_RED,str);

}

/* PROMPT USER CONFIRM EXIT */
int exit_confirm()
{
    int answer;

    whandle[0]=wopen(12,15,14,65,1,WHITE|_RED);
    statusline("Y|y => YES     N|n => NO");
    cursoron();
    wgotoxy(0,40);
    wprints(0,5,WHITE|_RED,"Are you sure you want to exit(Y/N)?");
    answer = wgetchf("YyNn");
    if (answer=='Y'||answer=='y') {
    wclose();
    return(1);
    }
    wclose();
    cursoroff();
    return(0);
}

/* PROGRAM EXIT */
void prog_exit()
{
    clearkeys();
    srestore(sbuf);
    gotoxy_(row,0);
    puts("ALL DONE");
    gotoxy_(row,col);
    exit(0);
}
```

```
/*-------------------------------------------------------*/
/*  Program for designing FIR filter via window method   */
/*  by WENTAO LYOU, Summer term, 1988                    */
/*                                                       */
/*                                                       */
/*  Filter type : Lowpass, Highpass, Bandpass and Bandstop */
/*  Window type : Rectangular, Bartlett, Hamming,        */
/*                Generalize hamming, Hanning and Kaiser */
/*  Maximum filter length : 128                          */
/*-------------------------------------------------------*/

/*----- INCLUDE FILES -----*/
#include <stdlib.h>
#include <math.h>
#include <tcdef.h>
#include <window.h>

/*-- FILTER TYPES --*/
typedef enum { lowpass, highpass, bandpass, bandstop }
             filters;

/*-- WINDOW TYPES --*/
typedef enum { rectangular, bartlett, hamming, genhamming, hanning,
             kaiser }  windows;

/*-- WINDOW DATA TYPE --*/
typedef struct {
            filters type;
            float   fc1, fc2;
        } fdata;

/*-- WINDOW DATA TYPE --*/
typedef struct {
           windows  type;
           float    par;
        } wdata;

/*-- WINDOWED FILTER DATA TYPE --*/
typedef struct {
            fdata   fil;
            wdata   win;
            float   pb_fc1, pb_fc2, sb_fc1, sb_fc2;
            float   pb_rip1, pb_rip2, sb_rip1, sb_rip2;
        } WFilter;

/*-----GLOBAL VARIABLES -----*/
int     Nfilt, halfN;
int     ODD;
float   Ledge, Uedge;
float   HH[65];
float   WIN[65];
```

```
float   FFT[257];
float   MAG[129];
float   PHS[129];
WFilter WF;

/*----- DEFINES -----*/
#define PI        3.14159265358979224
#define PI2       6.28318530717958534
#define FC1       WF.fil.fc1
#define FC2       WF.fil.fc2
#define FTYPE     WF.fil.type
#define WTYPE     WF.win.type
#define PARA      WF.win.par


/*----- FUNCTION PROTOTYPE -----*/
void    rec_window(void);
void    bar_window(void);
void    ham_window(void);
double  Io(double x);
void    kai_window(void);
void    calwincoef(void);
void    calimpcoef(void);
void    swapcoeff(void);
int     validinputs(void);

/*-- MAIN ROUTINE --*/
main(int argc, char *argv[])
{
   char *designame;

   FTYPE=atoi(argv[1])-1;
   WTYPE=atoi(argv[2])-1;
   Nfilt=atoi(argv[3]);
   FC1=atof(argv[4]);
   FC2=atof(argv[5]);
   PARA=atof(argv[6]);
   designame=argv[7];
   halfN=(Nfilt+1)/2;
   ODD=Nfilt%2;
   if (!validinputs())
       exit(100);
   wopen(12,25,14,55,0,WHITE|_GREEN);
   wprints(0,10,WHITE|_GREEN|BLINK,"Working...");
   calwincoef();
   calimpcoef();
   swapcoeff();
   fft();
   waitkey();
   savedesign(designame);
   wclose();
```

```
}

/*-- RECTANGULAR WINDOW FUNCTION --*/
void rec_window()
{
    int   i;
    for (i=1; i<=halfN; i++)
        WIN[i] = 1.0;
}

/*-- BARTLETT WINDOW FUNCTION --*/
void bar_window()
{
    int   i;
    float n;

    for (i=1; i<=halfN; i++) {
        n = (ODD ? (float)i-1.0 : (float)i-0.5);
        WIN[i] = 1.0 - (n/halfN);
    }
}

/*-- GENERALIZED HAMMING WINDOW FUNCTION --*/
void ham_window()
{
    int   i, hn;
    float n, nn, alpha, gamma;

    if (WTYPE == hanning) {
        PARA = 0.5;
        nn = (float)Nfilt+1.0;
        hn = halfN+1.0;
    }
    else {
        if (WTYPE == hamming)
            PARA = 0.54;
        nn = (float)Nfilt-1.0;
        hn = halfN;
    }
    alpha = PARA;
    gamma = 1.0-PARA;
    for (i = 1; i <= hn; i++) {
        n = (ODD ? (float)i-1.0 : (float)i-0.5);
        WIN[i] = alpha+gamma*cos(2.0*PI*n/nn);
    }
}

/*-- MODIFIED BESSEL FUNCTION OF THE FIRST KIND AND ZEROTH ORDER --*/
double Io(double x)
{
    int   i;
```

```
    float y=x/2.0, t=1.0e-08, e=1.0, de=1.0, ri, sde;

    for (i = 1; i <= 25; i++) {
        ri = (float)i;
        de = de*(y/ri);
        sde = de*de;
        e = e + sde;
        if (e*t-sde > 0.0)
            break;
     }
    return(e);
}

/*-- KAISER WINDOW FUNCTION --*/
void kai_window()
{
    int    i;
    double Io_beta;
    float  n, sn, s2n, s, beta;

    if (PARA>50.0)
        beta = 0.1102*(PARA-8.7);
    else
        if (PARA>=20.96)
            beta = 0.58417*pow(PARA-20.96,0.4)+0.07886*(PARA-20.96);
        else
            beta = 0.0;
    Io_beta = Io(beta);
    sn = ((float)Nfilt-1.0)*((float)Nfilt-1.0);
    for (i = 1; i <= halfN; i++) {
        n = (ODD ? (float)i-1.0 : (float)i-0.5);
        s2n = 4*n*n;
        s = beta*sqrt(1.0 - (s2n/sn));
        WIN[i] = Io(s)/Io_beta;
    }
}

/*-- CALCULATE WINDOW COEFFICIENTS --*/
void calwincoef()
{
    switch (WTYPE) {
        case rectangular  : rec_window();
                            break;
        case bartlett     : bar_window();
                            break;
        case hamming      :
        case genhamming   :
        case hanning      : ham_window();
                            break;
        case kaiser       : kai_window();
                            break;
```

```
    }
}

/*-- CALCULATE WINDOWED FILTER COEFFICIENTS --*/
void calimpcoef()
{
    int    i;
    float Fc, n, npi, nwc;

    /* calculate ideal impulse sequence */
    if (FTYPE == lowpass || FTYPE == highpass)
        Fc = FC1;
    else
        Fc = FC2 - FC1;
    if (ODD)
        HH[1] = 2.0 * Fc;
    for (i = ODD+1; i <= halfN; i++) {
         n = (ODD ? (float)i-1.0 : (float)i-0.5);
         npi = n*PI;
         if (FTYPE == lowpass || FTYPE == highpass) {
             nwc = n*PI2*Fc;
             HH[i] = sin(nwc)/npi;
         }
         else {
             nwc = n*PI*Fc;
             HH[i] =  2.0*(sin(nwc)/npi)*cos(npi*(FC1 + FC2));
         }
    }

    /* calculate windowed filter coefficients */
    if (FTYPE == lowpass || FTYPE == bandpass) {
        for (i = 1; i <= halfN; i++)
            HH[i] = HH[i]*WIN[i];
    }
    else {
        HH[1] = 1.0-HH[1]*WIN[1];
        for (i = 2; i <= halfN; i++)
            HH[i] = -HH[i]*WIN[i];
    }

}

/*-- SWAP COEFFICIENTS --*/
void swapcoeff()
{
    int   i;
    float temp;

    for (i = 1; i <= (halfN+1)/2; i++) {
        temp = HH[i];
        HH[i] = HH[halfN-i+1];
```

```
        HH[halfN-i+1] = temp;
    }
}

/*-- CHECK FOR VALID INPUTS --*/
int validinputs()
{

    int errN=0, errF=0, errP=0;
    char *errmsg[4] = {"",
                       "Invalid Filter Length",
                       "Invalid Band Edge",
                       "Invalid Window Parameter"};
    if (Nfilt>128 || Nfilt<3)
        errN=1;
    if ((FTYPE==1||FTYPE==3)&&(!ODD))
        errN=1;
    if (((FTYPE==2||FTYPE==4)&&(FC2<FC1)) || (FC1<0.0||FC1>0.5) ||
          (FC2<0.0||FC2>0.5))
        errF=2;
    if (WTYPE==3 && (PARA<0.0 || PARA > 1.0))
        errP=3;
    if (errN||errF||errP) {
        beep();
        wopen(18,0,23,30,1,WHITE|_RED);
        wprints(0,2,WHITE|_RED,errmsg[errN]);
        wprints(1,2,WHITE|_RED,errmsg[errF]);
        wprints(2,2,WHITE|_RED,errmsg[errP]);
        wprints(3,2,YELLOW|_RED|BLINK,"Press a key to continue");
        wclose();
        return(0);
    }
    return(1);
}
```

```
/*-------------------------------------------------------------*/
/*  Program for designing FIR filter via frequency sampling  */
/*  by WENTAO LYOU, Summer term, 1988                        */
/*                                                           */
/*                                                           */
/*  Filter type : Lowpass, Highpass, Bandpass and Bandstop   */
/*  Maximum filter length : 128                              */
/*-------------------------------------------------------------*/

/*----- INCLUDE FILES -----*/
#include <stdlib.h>
#include <math.h>
#include <tcdef.h>
#include <window.h>

/*-- FILTER TYPES --*/
typedef enum { lowpass, highpass, bandpass, bandstop }
               filters;

/*-- FILETR DATA TYPE --*/
typedef struct {
            filters type;
            float   fc1, fc2;
        } fdata;

/*-- FREQUENCY SAMPLING FILTER DATA TYPE --*/
typedef struct {
            fdata   fil;
            float   pb_fc1, pb_fc2, sb_fc1, sb_fc2;
            float   pb_rip1, pb_rip2, sb_rip1, sb_rip2;
        } FSFilter;

/*----- GLOBAL VARIBALS -----*/
int      Nfilt, halfN;
int      ODD;
float    Ledge, Uedge;
float    HH[65];
float    FFT[257];
float    MAG[129];
float    PHS[129];
FSFilter FSF;

/*----- DEFINES -----*/
#define PI2        6.28318530717958534
#define FC1        FSF.fil.fc1
#define FC2        FSF.fil.fc2
#define FTYPE      FSF.fil.type
#define FRE        FFT

/*----- FUNCTION PROTOTYPES -----*/
void    sampling(void);
```

```
void    calimpcoef(void);
void    extimpcoef(void);
void    calfilchars(void);
int     validinputs(void);

/*--  MAIN ROUTINE  --*/
main(int argc, char *argv[])
{
   char *designame;

   /* get input data */
   FTYPE=atoi(argv[1])-1; /* input filter type */
   Nfilt=atoi(argv[2]); /* input filter length */
   FC1=atof(argv[3]); /* input filter bandedge 1 */
   FC2=atof(argv[4]); /* input filter bandedge 2
                         (for Bandpass & Bandstop) */
   designame=argv[5]; /* input design name to save results */
   halfN=(Nfilt+1)/2;/* calculate half filter length */
   ODD=Nfilt%2;

   if (!validinputs())
       exit(100);
   wopen(12,25,14,55,0,WHITE|_GREEN);
   wprints(0,10,WHITE|_GREEN|BLINK,"Working...");
   sampling();
   calimpcoef();
   extimpcoef();
   fft();
   savedesign(designame);
   wclose();
}

/*-- INITIALIZE SAMPLES VALUES => 1 FOR PASSBAND, 0 FOR STOPBAND --*/
void sampling()
{
    int    i, j=0, M1, N1, Nh;
    float  des[2], deltF, tmp, fup;

    switch (FTYPE) {
        case lowpass  :
        case bandstop : des[0] = 1.0;
                        des[1] = 0.0;
                        break;
        case highpass :
        case bandpass : des[0] = 0.0;
                        des[1] = 1.0;
                        break;
    }
    if (FTYPE==lowpass||FTYPE==highpass)
        FC2 = 0.5;
```

```
    M1 = halfN+1;
    Nl = Nfilt*FC1+1;
    Nh = ((FTYPE==lowpass||FTYPE==highpass) ? M1 : Nfilt*FC2+1);
    for (i=1; i<=Nl; i++)
        FRE[++j] = des[0];
    for (i=Nl+1; i<=Nh; i++)
        FRE[++j] = des[1];
    if (FTYPE==bandpass||FTYPE==bandstop)
        for (i=Nh+1; i<=M1; i++)
            FRE[++j] = des[0];
}

/*-- CALCULATE IMPULSE RESPONSE COEFFICIENTS --*/
void calimpcoef()
{
    int   i, j;
    float M, htmp;
    double w;

    M = (Nfilt+1)/2.0;
    w = PI2/Nfilt;

    for (j=1; j<=halfN; j++) {
        htmp = FRE[1]/2.0;
        for (i=2; i<=halfN; i++)
            htmp = htmp + FRE[i]*cos(w*(M-j)*(i-1.0));
        HH[j] = HH[Nfilt-j+1] = (2.0*htmp)/Nfilt;
    }
}

/*-- EXTEND THE COEFFICIENTS TO FULL FILTER LENGTH --*/
void extimpcoef()
{
    int   i;
    float temp;

    for (i = 1; i <= halfN; i++) {
        temp = HH[i];
        HH[i] = HH[halfN-i+1];
        HH[halfN-i+1] = temp;
    }
}


/*-- CHECK THE VALIDITY OF INPUT FILTER PARAMETERS --*/
int validinputs()
{
    int errN=0, errF=0;
    char *errmsg[2] = {"Invalid Filter Length",
```

```
                    "Invalid Band Edge(s)"};
    if (Nfilt>128 || Nfilt<3)
        errN= 0;
    if ((FTYPE==1||FTYPE==3)&&(!ODD))
        errN=0;
    if (((FTYPE==2||FTYPE==4)&&(FC2<FC1))
        || (FC1<0.0||FC1>0.5)||(FC2<0.0||FC2>0.5))
        errF=1;
    if (errN||errF) {
        beep();
        wopen(18,0,22,30,1,WHITE|_RED);
        wprints(0,2,WHITE|_RED,errmsg[errN]);
        wprints(1,2,WHITE|_RED,errmsg[errF]);
        wprints(3,2,YELLOW|_RED|BLINK,"Press a key to continue");
        waitkey();
        wclose();
        return(0);
    }
    return(1);
}
```

```
/*-------------------------------------------------------------*/
/*  Program for designing optimal FIR filter                   */
/*  by WENTAO LYOU, Summer term, 1988                          */
/*                                                             */
/*                                                             */
/*  Filter type : Lowpass, Highpass, Bandpass, Bandstop        */
/*                and Multibands.                              */
/*  Maximum filter length : 128                                */
/*-------------------------------------------------------------*/

/*----- INCLUDE FILES -----*/
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <tcdef.h>
#include <window.h>

/*-- FILTER PARAMETERS DATA TYPE --*/
typedef  struct {
             float ledge;
             float uedge;
             float des;
             float wight;
             float dev;
         } bandpar;

/*-- FILTER DATA TYPE --*/
typedef  struct {
             int      nband;
             bandpar  band[11];
         } optfilter;

/*----- FUNCTION PROTOTYPE -----*/
int    setgrid(void);
void   initguess(int gridN, float iextremal[]);
void   remez(int gridN, float initextremal[], float alpha[]);
void   impresp(float alpha);
double Dev(double x[], int k, int n, int m);
double evfreresp(float gd, int n, double ad[], double x[], double y[]);
void   error(void);


/*----- GLOBAL VARIABLES -----*/
int   Nfilt;
int   halfN;
int   ODD;
float HH[65];
float FFT[257];
float MAG[129];
float PHS129];
optfilter  ofil;
```

```
float      G[1046], D[1046], W[1046];

/*----- DEFINES -----*/
#define   PI         3.14159265358979224
#define   PI2        6.28318530717958534
#define   gridden    16
#define   BANDNUM (ofil.nband)
#define   LEDGE   (ofil.band[lband].ledge)
#define   UEDGE   (ofil.band[lband].uedge)
#define   wight   (ofil.band[lband].wight)
#define   desir   (ofil.band[lband].des)
#define   DEV     (ofil.band[lband].dev)

/*-- MAIN ROUTINE --*/
main(int argc, char *argv[])
{
   char *designame;
   int  i, lband=0, gridN, initextremal[67];
   float  alpha[67];
   float le, ue, de, wt;
   FILE *fp;

   BANDNUM=atoi(argv[1]);
   Nfilt=atoi(argv[2]);
   designame=argv[3];

   fp=fopen("tmp","rt");
   for (i=0;i<BANDNUM;i++)  {
       lband += 1;
       fscanf(fp,"%*c %f %f %f %f\n",&le,&ue,&de,&wt);
       LEDGE=le;
       UEDGE=ue;
       desir=de;
       wight=wt;
   }
   fclose(fp);

   ODD=Nfilt%2;
   halfN = (ODD ? 1+(Nfilt/2) : Nfilt/2);

   wopen(12,25,14,55,0,WHITE|_GREEN);
   wprints(0,10,WHITE|_GREEN|BLINK,"Working...");
   gridN = setgrid();
   initguess(gridN, initextremal);
   remez(gridN, initextremal, alpha);
   impresp(alpha);
   fft();
   savedesign(designame);
}

/*-- DENSE GRID --*/
```

```
int   setgrid()
{
    int    j = 1, lband = 1, gridN;
    float  deltaf, temp, fup;
    double change;

    deltaf = 0.5/(gridden*halfN);
    grid[j] = (NEG && (LEDGE < deltaf) ? deltaf : LEDGE);

    do {
        fup = UEDGE;
        do {
            temp = G[j];
            G[++j] = temp+deltaf;
         }  while (G[j] <= fup);

         G[j-1] = fup;
         if (++lband > ofil.nband)
             break;
         G[j] = LEDGE;
    }  while(lband <= ofil.nband);

    gridN = j-1;
    if (!ODD) {
        for (j = 1; j <= gridN; j++) {
            change = cos(PI*G[j]);
            D[j] = D[j]/change;
            W[j] = W[j]*change;
        }
    }
    return (gridN);
}

/*-- INITIAL GUESS EXTREAMAL FREQUENCIES --*/
void initguess(int gridN, float iextremal[])
{
    float temp;
    int   j;

    temp = (float)(gridN-1)/(float)halfN;
    for (j = 1; j <= halfN; j++) {
        initext[j] = (int  )((j-1)*temp+1);

        }
    initext[halfN+1] = gridN;
}

/*-- CALCULATE DEVIATION --*/
double Dev(double x[], int k, int n, int m)
{
  int   j, l;
```

```
  double dd = 1.0, q;

  q = x[k];
  for (l = 1; l <= m; l++) {
      for (j = l; j <= n; j += m) {
          if (j != k)
             dd = 2.0*dd*(q-x[j]);
      }
  }
  return(1.0/dd);
}

/*-- EVALUATE FREQUENCY RESPONSE --*/
double evfreresp(float gd, int n, double ad[], double x[], double y[])
{
  double  p = 0.0, d = 0.0, c, xf;
  int   j;

  xf = cos(PI2*gd);
  for (j = 1; j <= n; j++) {
      c = xf - x[j];
      c = ad[j]/c;
      d = d+c;
      p = p+c*y[j];
  }
  return(p/d);
}

/*-- REMEZ EXCHANGE TO GET APPROXIMATE FREQUENCY RESPONSE --*/
void remez(int gridN, float initext[], float alpha[])
{
    double ad[67], x[67], y[67];
    double a[66], p[66],q[66];
    double dtemp, dnum, dden, dev, devl = -1.0, err;
    double xt, xt1;
    float  y1, comp = 0.0, kn;
    float  fsh = 1.0e-6, gtemp, deltaf, aa, bb, ft, xe;
    int    itrmax=25, niter=0, nu, nut, nut1, nz=halfN+1, nzz=halfN+2;
    int    jet, k1, knz, ynz, kup, klow=0, change=1;
    int    done, search = 1, luck, i=1, j, k;
    int    nm1=halfN-1, cn = 2*halfN-1, kkk = 0;

    do {
        initext[nzz] = gridN+1;
        done = 6;
        if (++niter <= itrmax) {
            for (i = 1; i <= nz; i++) {
                j = initext[i];
                x[i] = cos(PI2*G[j]);
            }
            jet = ((halfN-1)/15)+1;
```

```
for (i = 1; i <= nz; i++)
    ad[i] = Dev(x, i, nz, jet);
k = 1;
dnum = 0.0;
dden = 0.0;
for (i = 1; i <= nz; i++) {
    j = initext[i];
    dnum = dnum+ad[i]*D[j];
    dden = dden+(k*ad[i])/W[j];
    k = -k;
}
dev = dnum/dden;

nu = (dev > 0.0 ? -1 : 1);
dev = -(float)nu*dev;
k = nu;
for (i = 1; i <= nz; i++) {
    j = initext[i];
    y[i] = D[j]+k*dev/W[j];

    k = -k;
}
if (dev <= devl) {
    error();
    break;
}

devl = dev;
change = 0;
k1 = initext[1];
knz = initext[nz];
klow = 0;
nut = -nu;
i = 1;

do {
    done = 0;
    if (i == nzz)
        ynz = comp;
    if (i < nzz) {
        kup = initext[i+1];
        j = initext[i]+1;
        nut = -nut;
        if (i == 2)
            y1 = comp;
        comp = dev;
        if (j >= kup) {
            j -= 1;
            search = 2;
        }
        else {
```

```
            err = evfreresp(G[j], nz, ad, x, y);
            err = (err-D[j])*W[j];
            dtemp = err*nut-comp;
            if (dtemp <= 0.0) {
                j -= 1;
                search = 2;
            }
            else {
                comp = nut*err;
                search = 1;
            }
        }
    }
    else {
        if (i == nzz) {
            if (k1 > initext[1])
                k1 = initext[1];
            if (knz < initext[nz])
                knz = initext[nz];
            nut1 = nut;
            nut = -nu;
            j = 0;
            kup = k1;
            comp = ynz*1.00001;
            luck = 1;
            search = 5;
        }
        if (i > nzz) {
            if (luck > 9) {
                search = 0;
                done = 4;
            }
            else {
                if (comp > y1)
                    y1 = comp;
                k1 = initext[nzz];
                search = 6;
            }
        }
    }
    do {
        switch (search) {
            case 1 : if (++j < kup) {
                        err = evfreresp(G[j], nz, ad,x,y);
                        err = (err-D[j])*W[j];
                        dtemp = err*nut-comp;
                        if (dtemp <= 0.0)
                            done = 1;
                        else
                            comp = nut*err;
                        break;
```

```
            }
            else
                done = 1;
            break;
case 2 :    j -= 1;
            if (j <= klow) {
                j = initext[i]+1;
                if (change > 0)
                    done = 1;
                else
                    search = 4;
                break;
            }
            err = evfreresp(G[j], nz, ad, x, y);
            err = (err-D[j])*W[j];
            dtemp = err*nut-comp;
            if (dtemp > 0.0) {
                comp = nut*err;
                search = 3;
                break;
            }
            if (change != 0)
                done = 2;
            break;
case 3 :    if (--j > klow) {
                err = evfreresp(G[j],nz,ad,x,y);
                err = (err-D[j])*W[j];
                dtemp = err*nut-comp;
                if (dtemp <= 0.0)
                    done = 3;
                else
                    comp = nut*err;
            }
            else
                done = 3;
            break;
case 4 :    j += 1;
            if (j >= kup) {
               done = 2;
               break;
            }
            err = evfreresp(G[j], nz, ad, x, y);
            err = (err-D[j])*W[j];
            dtemp = err*nut-comp;
            if (dtemp > 0.0) {
                comp = nut*err;
                search = 1;
            }
            break;
case 5 :    j += 1;
            if (j >= kup) {
```

```
                    luck = 6;
                    search = 6;
                    break;
                }
                err = evfreresp(G[j], nz, ad, x, y);
                err = (err-D[j])*W[j];
                dtemp = err*nut-comp;
                if (dtemp > 0.0) {
                    comp = nut*err;
                    j = nzz;
                    search = 1;
                }
                break;

        case 6 : j = gridN+1;
                klow = knz;
                nut = -nut1;
                comp = y1*1.00001;
                search = 7;
                break;
        case 7 : j -= 1;
                if (j <= klow) {
                    done = 5;
                    break;
                }
                err = evfreresp(G[j], nz, ad, x, y);
                err = (err-D[j])*W[j];
                dtemp = err*nut-comp;
                if (dtemp > 0.0) {
                    comp = nut*err;
                    j = nzz;
                    luck += 10;
                    search = 3;
                }
        default: break;
    }
}  while (done == 0);
switch (done) {
    case 1 : initext[i++] = j-1;
             klow = j-1;
             change += 1;
             break;
    case 2 : klow = initext[i++];
             break;
    case 3 : klow = initext[i];
             initext[i++] = j+1;
             change += 1;
             break;
    case 4 : kn = initext[nzz];
             for (i = 1; i <= halfN; i++)
                 initext[i] = initext[i+1];
```

```
                        initext[nz] = kn;
                        break;
                case 5 : if (luck != 6) {
                             for (i = 1; i <= halfN; i++)
                                 initext[nzz-i] = initext[nz-i];
                             initext[1] = k1;
                        }
                        else
                             done = (change > 0 ? 5 : 6);
                default: break;
            }
        } while (done <= 3);
    }
} while ( done != 6);

if (done == 6) {
    gtemp = G[1];
    x[nzz] = -2.0;
    deltaf = 1.0/cn;
    j = 1;
    if ((G[1] < 0.01 && G[gridN] >0.49) || halfN <= 3)
        kkk = 1;
    else {
        dtemp = cos(PI2*G[1]);
        dnum = cos(PI2*G[gridN]);
        aa = 2.0/(dtemp-dnum);
        bb = -((dtemp+dnum)/(dtemp-dnum));
    }
    for (i = 1; i <= halfN; i++) {
        ft = (i-1)*deltaf;
        xt = cos(PI2*ft);
         if (kkk == 0) {
            xt = (xt-bb)/aa;
            xt1 = sqrt(1.0-xt*xt);
            ft = atan2(xt1,xt)/PI2;
        }
        do {
            xe = x[j];
            if (xt > xe) {
                if (xt-xe < fsh)
                    a[i] = y[j];
                else {
                    G[1] = ft;
                    a[i] = evfreresp(G[1], nz, ad, x, y);
                }
                break;
            }
            if (xe-xt < fsh) {
                a[i] = y[j];
                break;
            }
```

```
            j += 1;
        } while(1);
        j = (j > 1 ? j-1 : j);
    }
    G[1] = gtemp;
    dden = PI2/cn;
    for (i = 1; i <= halfN; i++) {
        dtemp = 0.0;
        dnum = (i-1)*dden;
        if (nm1 >= 1) {
            for (k = 1; k <= nm1; k ++)
                dtemp = dtemp+a[k+1]*cos(dnum*k);
        }
        dtemp = a[1]+2.0*dtemp;
        alpha[i] = dtemp;
    }
    for (i = 2; i <= halfN; i++)
        alpha[i] = (2.0*alpha[i])/cn;
    alpha[1] = alpha[1]/cn;
    if (kkk != 1) {
        p[1] = 2.0*bb*alpha[halfN]+alpha[nm1];
        p[2] = 2.0*aa*alpha[halfN];
        q[1] = alpha[halfN-2]-alpha[halfN];
        for (i = 2; i <= nm1; i++) {
            if (i == nm1) {
                aa = 0.5*aa;
                bb = 0.5*bb;
            }
            p[i+1] = 0.0;
            for (k = 1; k <= i; k++) {
                a[k] = p[k];
                p[k] = 2.0*bb*a[k];
            }
            p[2] += 2.0*aa*a[1];
            for (k = 1; k <= i-1; k++)
                p[k] += q[k]+aa*a[k+1];
            for (k = 3; k <= i+1; k++)
                p[k] += aa*a[k-1];
            if (i != nm1) {
                for (k = 1; k <= i; k++)
                    q[k] = -a[k];
                q[1] += alpha[nm1-i];
            }
        }
        for (i = 1; i <= halfN; i++)
            alpha[i] = p[i];
    }
    if (halfN <= 3) {
        alpha[halfN+1] = 0.0;
        alpha[halfN+2] = 0.0;
    }
```

```
    }
}

/*-- CALCULATE IMPULSE RESPONSE --*/
void impresp(float alpha)
{

    int   nz=halfN+1, nm1=halfN-1, j;

    if (ODD) {
        for (j = 1; j <= nm1; j++) {
            HH[j] = 0.5*alpha[nz-j];
        }
        HH[halfN] = alpha[1];
    }
    else {
        HH[1] = 0.25*alpha[halfN];
        for (j = 2; j <= nm1; j++) {
            HH[j] = 0.25*(alpha[nz-j]+alpha[halfN+2-j]);
        }
        HH[halfN] = 0.5*alpha[1]+0.25*alpha[2];
    }

}

/*-- DISPLAY ERROR MESSAGE --*/
void error()
{
    wprints(0,5,WHITE|_RED|BLINK,"Failure converge");
}
```

```
/*-------------------------------------------------------------*/
/*  Porgram to plot Frequency Response and Impulse Response  */
/*  by WENTAO LYOU, Summer term, 1988                        */
/*-------------------------------------------------------------*/

/*----- INCLUDE FILES -----*/
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>

/*----- DEFINE -----*/
#define halfN    (Nfilt+1)/2

struct PTS {
      int x, y;
};

/*-----GLOBAL VARIABLES -----*/
int    GraphDriver;
int    GraphMode;
int    MaxX, MaxY;
int    MaxColors;
int    ErrorCode;
double AspectRatio;
struct palettetype palette;
int    Nfilt;
float  HH[65];
float  FFT[257];
float  MAG[129];
float  PHS[129];



/*----- FUNCTION PROTOTYPES -----*/
void   initiplot(void);
float  maxone(int n, float a[]);
float  minone(int n, float a[]);
float  absf(float a);
void   graphwindow(char *header);
void   plotimp(float H[]);
void   statusline(char *msg );
void   drawborder(void);
int    gprintf(int *xloc, int *yloc, char *fmt, ... );
void   loaddata(int dtype, char *filename);

/*--  MAIN ROUTINE  --*/
main(int argc,char *argv[])
{
```

```
    int plottype,i;
    char *designame;

    plottype = atoi(argv[1]);
    designame = argv[2];
    switch (plottype) {
      case 1 : loaddata(plottype, designame);
               initplot();
               plotimp(HH);
               break;
      case 2 : loaddata(plottype, designame);
               initplot();
               plotmag(MAG);
               break;
      case 3 : loaddata(plottype, designame);
               initplot();
               plotphs(PHS);
               break;
      default: break;
    }
    closegraph();
    exit(1);
}

/*--  INITIALIZE GRAPHIC SYSTEM  --*/
void initplot(void)
{
    int xasp, yasp;

    GraphDriver = DETECT;
    initgraph(&GraphDriver, &GraphMode, "");
    ErrorCode = graphresult();
    if (ErrorCode != grOk) {
        printf(" Graphics System Error: %s\n",
                grapherrormsg( ErrorCode ) );
        exit(1);
    }

    getpalette( &palette );
    MaxColors = getmaxcolor() + 1;

    MaxX = getmaxx();
    MaxY = getmaxy();

    getaspectratio(&xasp, &yasp);
    AspectRatio = (double)xasp/(double)yasp;
}

/*--  FUNCTION TO FIND MAXIMUM VALUE AMOMG ARRAY ELEMENTS --*/
float maxone(int n, float a[])
{
```

```
    int i;
    float temp = 0.0;

    for (i=1; i<=n; i++)
        temp = (a[i] > temp ? a[i] : temp);
    return(temp);
}

/*--  FUNCTION TO FIND MINIMUM VALUE AMOMG ARRAY ELEMENTS --*/
float minone(int n, float a[])
{
    int i;
    float temp = 1.0;

    for (i=1; i<=n; i++)
        temp = (a[i] < temp ? a[i] : temp);
    return(temp);
}


/*--  FUNCTION TO FIND ABSOLUTE VALUE  --*/
float absf(float a)
{
    return((a>0 ? a : -a));
}

/*--  PLOT FREQUENCY RESPONSE (MAGNITUDE) --*/
void plotmag(float P[])
{
    int w, height, width, style, step, Vscale, Hscale;
    int i, wc, hc, sx, sy, x, y, ix, iy;
    float unit;
    struct viewporttype vp;
    char buffer[40];

    setgraphmode(getgraphmode());
    graphwindow("Frequency Response - Magnitude");

    Vscale = (MaxY < 200 ? 12 : 24);
    Hscale = (MaxX < 320 ? 2 : 4);

    getviewsettings(&vp );

    wc = (vp.right-vp.left)/2;
    hc = (vp.bottom-vp.top)/2;

    sx = vp.left+60;
    sy = vp.bottom-45;

    setlinestyle(0, 1, NORM_WIDTH);
    rectangle(sx, sy-11*Vscale, sx+127*Hscale, sy);
```

```
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    height = textheight("H");
    width = textwidth("W");

    outtextxy(wc, sy+3*height, "Frequency");

    w = 127*Hscale/5;
    for (i=0; i<=5; i++) {
        x = sx+w*i;
        if (i>=1) x = x+1;
        y = sy+2*height;
        line(x, sy, x, sy+5);
        gprintf(&x, &y, "%3.1f",(float)i/10.0);
    }

    iy = sy-Vscale*P[1]*10;
    moveto(sx,iy);

    for (i=1; i<=127; i++) {
        x = sx+Hscale*i;
        y = sy-Vscale*P[i+1]*10;
        lineto(x, y);
    }

    for (i=0; i<=11; i++) {
        y = sy-Vscale*i;
        x = sx-2*width;
        line(sx, y, sx+5, y);
        gprintf(&x, &y, "%3.1f",(float)i/10.0);
    }

    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(DEFAULT_FONT, VERT_DIR, 1);
    outtextxy(sx-50, hc, "Magnitude");

    settextjustify(LEFT_TEXT, TOP_TEXT);
    statusline("Press any key to continue");
    getch();
    restorecrtmode();
}

/*--  PLOT FREQUENCY RESPONSE (PHASE) --*/
void plotphs(float P[])
{
    int w, height, width, style, step, Vscale, Hscale;
    int i, wc, hc, sx, sy, x, y, ix, iy;
    float unit;
    struct viewporttype vp;
    char buffer[40];
```

```
setgraphmode(getgraphmode());
graphwindow("Frequency Response - Phase");

Vscale = (MaxY < 200 ? 3 : 6);
Hscale = (MaxX < 320 ? 2 : 4);

getviewsettings( &vp );

wc = (vp.right-vp.left)/2;
hc = (vp.bottom-vp.top)/2;

sx = vp.left+60;
sy = MaxY/2-20;

setlinestyle(0, 1, NORM_WIDTH );
rectangle(sx, sy-20*Vscale, sx+127*Hscale, sy+20*Vscale);
line(sx, sy, sx+127*Hscale, sy);

settextjustify(CENTER_TEXT, CENTER_TEXT);
settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
height = textheight("H");
width = textwidth("W");

outtextxy(wc, sy+2*height+20*Vscale, "Frequency");

w = 127*Hscale/5;
for (i=0; i<=5; i++) {
    x = sx+w*i;
    if (i>=1) x = x+1;
    y = sy+height+20*Vscale;
    line(x, sy+20*Vscale, x, sy+20*Vscale+3);
    gprintf(&x, &y, "%3.1f",(float)i/10.0);
}

iy = sy-Vscale*P[1]*10;
moveto(sx,iy);

for (i=1; i<=127; i++) {
    x = sx+Hscale*i;
    y = sy-Vscale*P[i+1]*10;
    lineto(x, y);
}

for (i=0; i<=20; i=i+4) {
    y = sy-Vscale*i;
    x = sx-3*width;
    iy = sy+Vscale*i;
    line(sx, y, sx+5, y);
    line(sx, iy, sx+5, iy);
    gprintf(&x, &y, "%3.1f",(float)i/10.0);
```

```
        gprintf(&x, &iy, "%3.1f",(float)-i/10.0);
    }

    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(DEFAULT_FONT, VERT_DIR, 1);
    outtextxy(sx-50, hc, "Phase");

    settextjustify(LEFT_TEXT, TOP_TEXT);
    statusline("Press any key to continue");
    getch();
    restorecrtmode();
}

/*--  PLOT IMPULSE RESPONSE --*/
void plotimp(float H[])
{
    int  plotT, plotB, plotH, Vscale, Hscale;
    int  textH, centX, centY, baseX, baseY;
    int  i, w, h, x, x1, y, y1;
    struct viewporttype vp;

    setgraphmode(getgraphmode());
    graphwindow("Impulse Response");
    plotT = absf(maxone(halfN,H))*10+1;
    plotB = absf(minone(halfN,H))*10+1;
    plotH = plotT+plotB;

    Vscale = (MaxY < 200 ? 120/plotH : 260/plotH);
    Hscale = (MaxX < 320 ? 256/(Nfilt-1) : 512/(Nfilt-1));

    getviewsettings(&vp);
    centX = (vp.right-vp.left)/2;
    centY = (vp.bottom-vp.top)/2;
    baseX = vp.left+60;
    baseY = vp.bottom-45;

    setlinestyle(0, 1, NORM_WIDTH );
    rectangle(baseX,baseY-plotH*Vscale,baseX+(Nfilt-1)*Hscale,baseY);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    settextjustify(RIGHT_TEXT, CENTER_TEXT);

    x = baseX - textwidth("W");
    for (i=-plotB; i<=plotT; i++) {
        y = baseY-Vscale*(i+plotB);
        if (i==0) {
            line(baseX, y, baseX+(Nfilt-1)*Hscale, y);
            y1 = y;
        }
        line(baseX, y, baseX+5, y);
        gprintf(&x, &y, "%3.1f",(float)i/10);
    }
```

```
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(DEFAULT_FONT, VERT_DIR, 1);
    outtextxy(baseX-50, centY, "Value");

    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    textH = textheight("H");

    w = (Nfilt>64 ? 10 : 5);
    for (i=0; i<=Nfilt-1; i++) {
        if (!(i%w) || (i==Nfilt-1)) {
            x = baseX+i*Hscale;
            y = baseY+textH+3;
            line(x, baseY, x, baseY+3);
            gprintf(&x, &y, "%3d", i);
        }
    }
    outtextxy(centX, baseY+2*textH+5, "Time in Samples");

    setlinestyle(SOLID_LINE, 1, THICK_WIDTH );
    for (i=1; i<=halfN; i++) {
        x = baseX+Hscale*(i-1);
        x1 = baseX+Hscale*(Nfilt-i);
        h = y1-Vscale*H[i]*10;
        line(x, y1, x, h);
        line(x1, y1, x1, h);
    }
    statusline("Press any key to continue");
    getch();
    restorecrtmode();
}

/*--  PLOT A GRAPHICS WINDOW --*/
void graphwindow(char *header )
{
  int height;

  cleardevice();

  setviewport(0, 0, MaxX, MaxY, 1);
  if (GraphDriver == CGA) {
      setbkcolor(WHITE);
      setfillstyle(EMPTY_FILL, BLACK);
  }
  else {
      setcolor(MaxColors - 1);
      setfillstyle(SOLID_FILL, BLUE);
  }
  setviewport(0, 0, MaxX, MaxY, 1);
  height = textheight("H");
  bar(0, 0, MaxX, height+4);
```

```
  rectangle(0, 0, MaxX, height+4);
  settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
  settextjustify(CENTER_TEXT, TOP_TEXT);
  outtextxy(MaxX/2, 2, header);
  setviewport(0, height+4, MaxX, MaxY-(height+4), 1);
  drawborder();
  setviewport(1, height+5, MaxX-1, MaxY-(height+5), 1);

}

/*--  DISPLAY A STATUS LINE --*/
void statusline(char *msg )
{
  int height, scolor;

  setviewport(0, 0, MaxX, MaxY, 1);
  settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
  settextjustify(CENTER_TEXT, TOP_TEXT);
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
  if (MaxColors < 2) {
      if (GraphDriver == CGA)
          setbkcolor(MaxColors-1);
      else
          setcolor(MaxColors-1);
      setfillstyle(EMPTY_FILL, BLACK);
  }
  else {
      setcolor(MaxColors - 1);
      setfillstyle(SOLID_FILL, LIGHTRED);
  }
  height = textheight( "H");
  bar( 0, MaxY-(height+4), MaxX, MaxY);
  rectangle(0, MaxY-(height+4), MaxX, MaxY);
  outtextxy(MaxX/2, MaxY-(height+2), msg );
  setviewport(1, height+5, MaxX-1, MaxY-(height+5), 1);

}

/*--  DWAW A WINDOW BOX --*/
void drawborder()
{
  struct viewporttype vp;
  setcolor(MaxColors - 1);
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
  getviewsettings(&vp);
  rectangle(0, 0, vp.right-vp.left, vp.bottom-vp.top);
}

/*-- GRAPHICS PRINTF  --*/
int gprintf(int *xloc, int *yloc, char *fmt, ...)
```

```
{
  va_list  argptr;
  char str[140];
  int cnt;

  va_start(argptr, format);
  cnt = vsprintf(str, fmt, argptr);
  outtextxy(*xloc, *yloc, str);
  *yloc += textheight("H") + 2;
  va_end(argptr);
  return(cnt);
}

/*-- LOAD DATD FILE --*/
void loaddata(int ptype, char *filename)
{
    int i, j, k;
    FILE *fp;

    fp = fopen(filename, "rt");
    rewind(fp);
    switch (ptype) {
        case 1 :  fscanf(fp,"%*s %*s %*s\n");
                  fscanf(fp,"%*s %*s %d\n", &Nfilt);
                  for (i=1; i<=halfN; i++){
                      fscanf(fp,"%*s %*c %f %*c %*s\n", &HH[i]);
                  }
                  break;
        case 2 :
        case 3 :  fscanf(fp, "%*s %*s\n\n %*s %*c %*s %*c\n\n");
                  for (i=1; i<=128; i++) {
                      fscanf(fp,"%*s %*c %f %*c %*s %*s %*c %f
                             %*1c %*s\n",&MAG[i], &PHS[i]);
                  }
        default : break;
      }
      fclose(fp);
}
```

```
/*-------------------------------------------------------*/
/*  Routine to create data files associated with a design  */
/*  by WENTAO LYOU, Summer term, 1988                      */
/*-------------------------------------------------------*/

/*----- INCLUDE FILES -----*/
#include <stdio.h>
#include <stat.h>
#include <window.h>
#include "filter.h"

/*----- EXTERNA VARIABLES -----*/
extern short Nfilt;
extern short halfN;
extern float HH[65];
extern float FFT[257];
extern float MAG[129];
extern float PHS[129];

/*-----GLOBAL VARIABLE -----*/
char *fileext[] = {
    ".LST", ".IMP", ".DFT", ".FRQ"
};


int savedesign(char *fname)
{
    FILE  *fp;
    char  *fn="              ";
    short c, i, j, handle;

    wsize(14,75);
    for ( j=1; j<= 3; j++) {
        strcpy(fn, fname);
        strcat(fn, fileext[j]);
        handle = creat(fn, S_IREAD|S_IWRITE);
        fp = fdopen(handle, "wt");
        if (fp == NULL) {
            clrwin(13,26,13,74);
            wprints(2,2,WHITE|_GREEN,"MSGNOOPEN");
            return(0);
        }
        else {
            clrwin(13,26,13,74);
            wgotoxy(0,15);
            wprintf("Saving %s", fn);
            switch (j) {
              case 1:  fprintf(fp, " Impulse Response Coefficients\n");
                       fprintf(fp, "   Filter Length    %d\n", Nfilt);
                       for (i=1; i<=halfN; i++)
                         fprintf(fp,"h[%03d] =  %+18.10e  = h[%03d]\n",
```

```
                                    i-1, HH[i], Nfilt-i);
                        break;
                case 2: fprintf(fp, "DFT Coefficients\n\n");
                        fprintf(fp, "Real Part :                    ");
                        fprintf(fp, "       Imagenary Part :\n\n");
                        for (i=1; i<=128; i++) {
                          fprintf(fp,"  X[%03d] = %+16.8e = X[%03d]   ",
                                    i-1, FFT[2*i-1], 256-i);
                          fprintf(fp,"  Y[%03d] = %+16.8e = -Y[%03d]\n",
                                    i-1, FFT[2*i], 256-i);
                        }
                        break;
                case 3: fprintf(fp, "Frequency Response\n\n");
                        fprintf(fp, "Magnitude :                    ");
                        fprintf(fp, "       Phase :\n\n");
                        for (i=1; i<=128; i++) {
                          fprintf(fp," M[%03d] = %+16.8e = M[%03d]  ",
                                    i-1, MAG[i], 256-i);
                          fprintf(fp," P[%03d] = %+16.8e = P[%03d]\n",
                                    i-1, PHS[i], 256-i);
                        }
                  default: break;
                }
                fflush(fp);
                fclose(fp);
            }
        }
        return(1);

}
```

```
/*-------------------------------------------------------*/
/*  Routines to calculate 256-point DFT coefficients by  */
/*  using a Radix-Eight FFT alogrithm (GERGLAND 1969).   */
/*                                                       */
/*  by WENTAO LYOU, Summer term, 1988                    */
/*-------------------------------------------------------*/

/*----- INCLUDE FILES -----*/
#include <math.h>
#include "filter.h"

/*----- EXTERNAL GLOBAL VARIABLES -----*/
extern short Nfilt;
extern short halfN;
extern float HH[65];
extern float FFT[257];
extern float MAG[129];
extern float PHS[129];

/*----- DEFINES -----*/
#define  p7   0.7071067812
#define  c22  0.9238795325
#define  s22  0.3826834323

/*----- FUNCTION PROTOTYPES -----*/
void fft();
void r4tr();
void r8tr();
void ord1();
void ord2();
void calmagphs();

/* MAIN FFT ROUTINE TO CALCULATE 256-POINTS DFT */
void fft()
{
    int i,nn=4, iter=64;

    for (i=1; i<= halfN ; i++)  {
      FFT[i] = HH[i];
      FFT[Nfilt-i+1]=HH[i];
    }
    r4tr(iter, &FFT[0], &FFT[iter], &FFT[iter*2], &FFT[iter*3]);
    for (i = 1; i <= 2; i++) {
      nn = nn*8;
      iter = 256/nn;
      r8tr(iter, nn, &FFT[0], &FFT[iter], &FFT[iter*2], &FFT[iter*3],
           &FFT[iter*4], &FFT[iter*5], &FFT[iter*6], &FFT[iter*7],
           &FFT[0], &FFT[iter], &FFT[iter*2], &FFT[iter*3],
           &FFT[iter*4], &FFT[iter*5], &FFT[iter*6], &FFT[iter*7]);
    }
    ord1(&FFT[0]);
```

```
    ord2(&FFT[0]);
    calmagphs();
}

/* RADIX 4 ITERATION SUBROUTINE */
void r4tr(int i, float *b0, float *b1, float *b2, float *b3)
{
    short k;
    float r0, r1;

    for (k=1; k<=i ; k++) {
        r0 = *(b0+k)+*(b2+k);
        r1 = *(b1+k)+*(b3+k);
        *(b2+k) = *(b0+k)-*(b2+k);
        *(b3+k) = *(b1+k)-*(b3+k);
        *(b0+k) = r0+r1;
        *(b1+k) = r0-r1;
    }
}

/* RADIX 8 ITERATION SUBROUTINE */
void r8tr(int i, int nn, float *br0, float *br1, float *br2,
          float *br3, float *br4, float *br5, float *br6, float *br7,
          float *bi0, float *bi1, float *bi2, float *bi3, float *bi4,
          float *bi5, float *bi6, float *bi7)
{
    short j, j0, j1, j2, j3, j4, j5, j6, j7, j8, j9, j10, j11, j12;
    short j13, j14, ji=3, j1=2, jr=2, jthet, jlast;
    short i8, k, k0, k1, th2;
    float t0, t1, t2, t3, t4, t5, t6, t7;
    float tr0, tr1, tr2, tr3, tr4, tr5, tr6, tr7;
    float ti0, ti1, ti2, ti3, ti4, ti5, ti6, ti7;
    float c1, c2, c3, c4, c5, c6, c7;
    float s1, s2, s3, s4, s5, s6, s7;
    float piovn, arg, pr, pi;
    short l[16];

    l[1] = nn/8;
    for (k=2; k<=15 ; k++) {
        l[k-1] = (l[k-1] < 2 ? 2 : l[k-1]);
        if (l[k-1] > 2)
            l[k] = l[k-1]/2;
        else
            l[k] = 2;
    }
    piovn = PI/(float)nn;
    for (j1=2; j1<=l[15]; j1 += 2) {
     for (j2=j1; j2<=l[14]; j2 += l[15]) {
      for (j3=j2; j3<=l[13]; j3 += l[14]) {
       for (j4=j3; j4<=l[12]; j4 += l[13]) {
        for (j5=j4; j5<=l[11]; j5 += l[12]) {
```

```
for (j6=j5; j6<=l[10]; j6 += l[11]) {
 for (j7=j6; j7<=l[9]; j7+= l[10]) {
  for (j8=j7; j8<=l[8]; j8+= l[9]) {
   for (j9=j8; j9<=l[7]; j9+= l[8]) {
    for (j10=j9; j10<=l[6]; j10 += l[7]) {
     for (j11=j10; j11<=l[5]; j11 += l[6]) {
      for (j12=j11; j12<=l[4]; j12 += l[5]) {
       for (j13=j12; j13<=l[3]; j13 += l[4]) {
        for (j14=j13; j14<=l[2]; j14 += l[3]) {
         for (jthet=j14; jthet<=l[1]; jthet += l[2]) {
           th2 = jthet-2;
           if (th2 <= 0) {
             for (k=1; k<=i ; k++) {
               t0 = *(br0+k)+*(br4+k);
               t1 = *(br1+k)+*(br5+k);
               t2 = *(br2+k)+*(br6+k);
               t3 = *(br3+k)+*(br7+k);
               t4 = *(br0+k)-*(br4+k);
               t5 = *(br1+k)-*(br5+k);
               t6 = *(br2+k)-*(br6+k);
               t7 = *(br3+k)-*(br7+k);
               *(br2+k) = t0-t2;
               *(br3+k) = t1-t3;
               t0 = t0+t2;
               t1 = t1+t3;
               *(br0+k) = t0+t1;
               *(br1+k) = t0-t1;
               pr = p7*(t5-t7);
               pi = p7*(t5+t7);
               *(br4+k) = t4+pr;
               *(br7+k) = pi+t6;
               *(br6+k) = t4-pr;
               *(br5+k) = pi-t6;
             }
             if (nn-8>0) {
               k0 = i*8+1;
               k1 = k0+i-1;
               for (k=k0; k<=k1; k++) {
                 pr = p7*(*(bi2+k)-*(bi6+k));
                 pi = p7*(*(bi2+k)+*(bi6+k));
                 tr0 = *(bi0+k)+pr;
                 ti0 = *(bi4+k)+pi;
                 tr2 = *(bi0+k)-pr;
                 ti2 = *(bi4+k)-pi;
                 pr = p7*(*(bi3+k)-*(bi7+k));
                 pi = p7*(*(bi3+k)+*(bi7+k));
                 tr1 = *(bi1+k)+pr;
                 ti1 = *(bi5+k)+pi;
                 tr3 = *(bi1+k)-pr;
                 ti3 = *(bi5+k)-pi;
                 pr = tr1*c22-ti1*s22;
```

```
      pi = ti1*c22+tr1*s22;
      *(bi0+k) = tr0+pr;
      *(bi6+k) = tr0-pr;
      *(bi7+k) = pi+ti0;
      *(bi1+k) = pi-ti0;
      pr = -tr3*s22-ti3*c22;
      pi = tr3*c22-ti3*s22;
      *(bi2+k) = tr2+pr;
      *(bi4+k) = tr2-pr;
      *(bi5+k) = pi+ti2;
      *(bi3+k) = pi-ti2;
    }
  }
  else {
    arg = th2*piovn;
    c1 = (float)cos(arg);
    s1 = (float)sin(arg);
    c2 = c1*c1-s1*s1;
    s2 = c1*s1+c1*s1;
    c3 = c1*c2-s1*s2;
    s3 = c2*s1+s2*c1;
    c4 = c2*c2-s2*s2;
    s4 = c2*s2+c2*s2;
    c5 = c2*c3-s2*s3;
    s5 = c3*s2+s3*c2;
    c6 = c3*c3-s3*s3;
    s6 = c3*s3+c3*s3;
    c7 = c3*c4-s3*s4;
    s7 = c4*s3+s4*c3;
    i8 = i*8;
    j0 = jr*i8+1;
    k0 = ji*i8+1;
    jlast = j0+i-1;
    for (j=j0; j<=jlast; j++) {
      k = k0+j-j0;
      tr1 = *(br1+j)*c1-*(bi1+k)*s1;
      ti1 = *(br1+j)*s1+*(bi1+k)*c1;
      tr2 = *(br2+j)*c2-*(bi2+k)*s2;
      ti2 = *(br2+j)*s2+*(bi2+k)*c2;
      tr3 = *(br3+j)*c3-*(bi3+k)*s3;
      ti3 = *(br3+j)*s3+*(bi3+k)*c3;
      tr4 = *(br4+j)*c4-*(bi4+k)*s4;
      ti4 = *(br4+j)*s4+*(bi4+k)*c4;
      tr5 = *(br5+j)*c5-*(bi5+k)*s5;
      ti5 = *(br5+j)*s5+*(bi5+k)*c5;
      tr6 = *(br6+j)*c6-*(bi6+k)*s6;
      ti6 = *(br6+j)*s6+*(bi6+k)*c6;
      tr7 = *(br7+j)*c7-*(bi7+k)*s7;
      ti7 = *(br7+j)*s7+*(bi7+k)*c7;
      t0 = *(br0+j)+tr4;
      t1 = *(bi0+k)+ti4;
```

```
    tr4 = *(br0+j)-tr4;
    ti4 = *(bi0+k)-ti4;
    t2 = tr1+tr5;
    t3 = ti1+ti5;
    tr5 = tr1-tr5;
    ti5 = ti1-ti5;
    t4 = tr2+tr6;
    t5 = ti2+ti6;
    tr6 = tr2-tr6;
    ti6 = ti2-ti6;
    t6 = tr3+tr7;
    t7 = ti3+ti7;
    tr7 = tr3-tr7;
    ti7 = ti3-ti7;
    tr0 = t0+t4;
    ti0 = t1+t5;
    tr2 = t0-t4;
    ti2 = t1-t5;
    tr1 = t2+t6;
    ti1 = t3+t7;
    tr3 = t2-t6;
    ti3 = t3-t7;
    t0 = tr4-ti6;
    t1 = ti4+tr6;
    t4 = tr4+ti6;
    t5 = ti4-tr6;
    t2 = tr5-ti7;
    t3 = ti5+tr7;
    t6 = tr5+ti7;
    t7 = ti5-tr7;
    *(br0+j) = tr0+tr1;
    *(bi7+k) = ti0+ti1;
    *(bi6+k) = tr0-tr1;
    *(br1+j) = ti1-ti0;
    *(br2+j) = tr2-ti3;
    *(bi5+k) = ti2+tr3;
    *(bi4+k) = tr2+ti3;
    *(br3+j) = tr3-ti2;
    pr = p7*(t2-t3);
    pi = p7*(t2+t3);
    *(br4+j) = t0+pr;
    *(bi3+k) = pi+t1;
    *(bi2+k) = t0-pr;
    *(br5+j) = pi-t1;
    pr = -p7*(t6+t7);
    pi = p7*(t6-t7);
    *(br6+j) = t4+pr;
    *(bi1+k) = pi+t5;
    *(bi0+k) = t4-pr;
    *(br7+j) = pi-t5;
}
```

```
                        jr += 2;
                        ji -= 2;
                        if (ji-jl <= 0) {
                          ji = 2*jr-1;
                          jl = jr;
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}

/* IN-PLACE REORDERING SUBROUTINE 1 */
void ord1(float *b)
{
    short j, k=4, k1=2, n=256;
    float t;

    for (j=4; j<=n; j+=2) {
        if (k > j) {
            t = *(b+j);
            *(b+j) = *(b+k);
            *(b+k) = t;
        }
        k -= 2;
        if (k <= k1) {
            k = 2*j;
            k1 = j;
        }
    }
}

/* IN-PLACE REORDERING SUBROUTINE 2 */
void ord2(float *b)
{
    short j1, j2, j3, j4, j5, j6, j7, j8, j9, j10, j11, j12;
    short j13, j14, j15, j16, j17, j18, ji, ij;
    short k, n=256;
```

```
short l[20];
float t;
l[1] = n;
for (k=2; k<=8 ; k++)
    l[k] = l[k-1]/2;
for (k=9; k<=20 ; k++)
    l[k] = 2;
ij = 2;
for (j1=2; j1<=l[19]; j1 += 2) {
 for (j2=j1; j2<=l[18]; j2 += l[19]) {
  for (j3=j2; j3<=l[17]; j3 += l[18]) {
   for (j4=j3; j4<=l[16]; j4 += l[17]) {
    for (j5=j4; j5<=l[15]; j5 += l[16]) {
     for (j6=j5; j6<=l[14]; j6 += l[15]) {
      for (j7=j6; j7<=l[13]; j7 += l[14]) {
       for (j8=j7; j8<=l[12]; j8 += l[13]) {
        for (j9=j8; j9<=l[11]; j9 += l[12]) {
         for (j10=j9; j10<=l[10]; j10 += l[11]) {
          for (j11=j10; j11<=l[9]; j11 += l[10]) {
           for (j12=j11; j12<=l[8]; j12 += l[9]) {
            for (j13=j12; j13<=l[7]; j13 += l[8]) {
             for (j14=j13; j14<=l[6]; j14 += l[7]) {
              for (j15=j14; j15<=l[5]; j15 += l[6]) {
               for (j16=j15; j16<=l[4]; j16 += l[5]) {
                for (j17=j16; j17<=l[3]; j17 += l[4]) {
                 for (j18=j17; j18<=l[2]; j18 += l[3]) {
                  for (ji=j18; ji<=l[1]; ji += l[2]) {
                      if (ij < ji) {
                          t = *(b+ij-1);
                          *(b+ij-1) = *(b+ji-1);
                          *(b+ji-1) = t;
                          t = *(b+ij);
                          *(b+ij) = *(b+ji);
                          *(b+ji) = t;
                      }
                      ij += 2;
                  }
                 }
                }
               }
              }
             }
            }
           }
          }
         }
        }
       }
      }
     }
    }
   }
  }
 }
}
```

```
        }
      }
    }
  }
}

/* CALCULATE MAGNITUDE AND PHASE */
void calmagphs()
{
    int i, j, k;

    FFT[2] = 0.0;
    for (i=1; i<=128; i++) {
        j = 2*i;
        k = j-1;
        MAG[i] = (float)sqrt(FFT[k]*FFT[k]+FFT[j]*FFT[j]);
        if (FFT[k] != 0.0 )
            PHS[i] = -(float)atan(FFT[j]/FFT[k]);
        else
            PHS[i] = -(FFT[k] > 0.0 ? PI/2 : -PI/2);
    }
}
```

LIST OF REFERENCES

Bergland, Glenn D. "A Radix-Eight Fast Fourier Transform Subroutine for Real-Valued Series." IEEE Transactions on Audio and Electroacoustics 17 (June 1969): 138-144.

IEEE, ed. Programs for Digital Signal Processing. NEW YORK: IEEE Press, 1979.

Jackson, Leland B. Digital Filters and Signal Processing. Norwell, MA.: Kluwer Acadmic Publishers, 1986.

Kaiser, J. F. "Nonrecursive Digital Filter Design Using the I_o-Sinh Window Function." Proceeding 1974 IEEE International Symposium on Circuits and Systems, San Francisco, April 1974, 20-23.

Oppenheim, Alan V., and Schafer, Ronald W. Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

Parks, Thomas W., and McClellan, James H. "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase." IEEE Transactions on Circuit Theory 19 (March 1972): 189-194.

Rabiner, Lawrence R. "Techniques for designing Finite-Duration Impulse-Response Digital Filters." IEEE Transactions on Communication Technology 19 (April 1971): 188-195.

Rabiner, Lawrence R., and Cold, Bernard. Theory and Application of Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

Rabiner, Lawrence R., McClellan, James H., and Parks, Thomas W. "FIR Digital Filter Design Technique Using Weighted Chebyshev Approximation." Proceeding IEEE 63 (April 1975): 595-610.

Stevens, AL. TURBO C Memory-Resident Utilities, Screen I/O and Programming Techniques. Portland, OR.: Management Information Source, 1987.

Williams, Charles S. Designing Digital Filters. Prentice-Hall Information and System Sciences Series, ed. Thomas Kailath. Englewood Cliffs, N.J.: Prentice-Hall, 1986.