Experiment 3

Digital Filters

The major goal of this experiment is to learn how to implement using real-time hardware the discrete-time filtering techniques usually presented in a typical required Electrical Engineering undergraduate course on signals and sytems and a Senior elective course on digital signal processing. Discrete-time filtering is often called digital filtering. In the process, you will learn more about the TMS320C30 and the EVM.

3.1 Discrete-Time Convolution and Frequency Responses

The output y[n] of a linear, time-invariant, discrete-time system (LTI) can be computed by convolving its input x[n] with its unit pulse response h[n]. The equation for this discrete-time convolution is

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{3.1}$$

The **z**-transform of a discrete-time convolution is the product of the transforms, that is,

$$Y(z) = \sum_{n=-\infty}^{\infty} y[n]z^{-n} = X(z)H(z) \tag{3.2}$$

The response of an LTI system to a sinusoid after the transients have become negligible is called its sinusoidal steady-state response. To determine this response, let the input be the sampled complex sinusoid

$$x[n] = Ce^{j\omega nT}$$

According to (3.1), the output is

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]Ce^{j\omega(n-k)T} = Ce^{j\omega nT}\sum_{k=-\infty}^{\infty} h[k]e^{-j\omega kT} = x[n]H(z)|_{z=e^{j\omega T}} \tag{3.3}$$

Thus, the output is a sinusoid at the same frequency as the input but with its amplitude scaled by the complex number

$$H^*(\omega) = H(z)|_{z=e^{j\omega T}} \tag{3.4}$$

The function $H^*(\omega)$ is called the *frequency response* of the system. The function $A(\omega) = |H^*(\omega)|$ is called the *amplitude response* of the system and $\theta(\omega) = \arg H^*(\omega)$ is called its *phase response*. Notice that all of these responses are periodic as functions of ω with period $\omega_s = 2\pi/T$. In polar form

$$H^*(\omega) = A(\omega)e^{j\theta(\omega)} \tag{3.5}$$

so, according to (3.3), the output can be expressed as

$$y[n] = CA(\omega)e^{j[\omega nT+\theta(\omega)]} \tag{3.6}$$

When the input is the real sinusoid

$$x[n] = C\cos(\omega nT + \phi) = \Re e\{Ce^{j\phi}e^{j\omega nT}\}$$

the ouput is

$$y[n] = \Re e\{H^*(\omega)Ce^{j\phi}e^{j\omega nT}\} = CA(\omega)\cos[\omega nT + \theta(\omega) + \phi]$$

In otherwords, the system scales the magnitude of the sinusoidal input by the amplitude response and shifts its phase by the phase response. This is the basis for digital filtering.

3.2 Finite Duration Impulse Response (FIR) Filters

3.2.1 Block Diagram for Most Common Realization

If the unit pulse response is identically zero outside the set of integers $\{0, 1, \cdots, N-1\}$, the convolution (3.1) becomes

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] = \sum_{k=n-N+1}^{n} x[k]h[n-k] \tag{3.7}$$

A filter of this type is called an N-tap *finite duration impulse response* (FIR) filter, *nonrecursive* filter, *transversal* filter, or *moving average* filter. A block diagram for the most common method of implementing FIR filters is shown in Fig. 3.1 on page 42. It consists of a delay line represented by the chain of blocks labelled z^{-1} and a set of taps into the delay line with weights equal to the unit pulse response samples.

The block diagram of an FIR shown in Fig. 3.1 could represent the physical layout of a hardware implementation or just the structure of a software algorithm. In a software implementation, the delay line would just be an array in memory. Entering a new sample into an array by shifting the entire array is inefficient. In the exercises described below, you will learn how to implement the "shift register" by forming a circular array using the C mod operator (%) and the hardware circular addressing capabilities of the TMS320C30.

3.2.2 Two Methods for Finding the Filter Coefficients to Achieve a Desired Frequency Response

Two programs for designing digital filters are included in the directory C:\DIGFIL. Both methods design filters with exactly linear phase which is a reason FIR filters are sometimes preferred over IIR filters.

Historically, the first method for designing digital filters was the *Fourier series and window function* method. (See [II.C.14, Chapter 8] for the theory.) The program **WINDOW.EXE** implements this method. It was taken from the IEEE Press book, *Programs for*

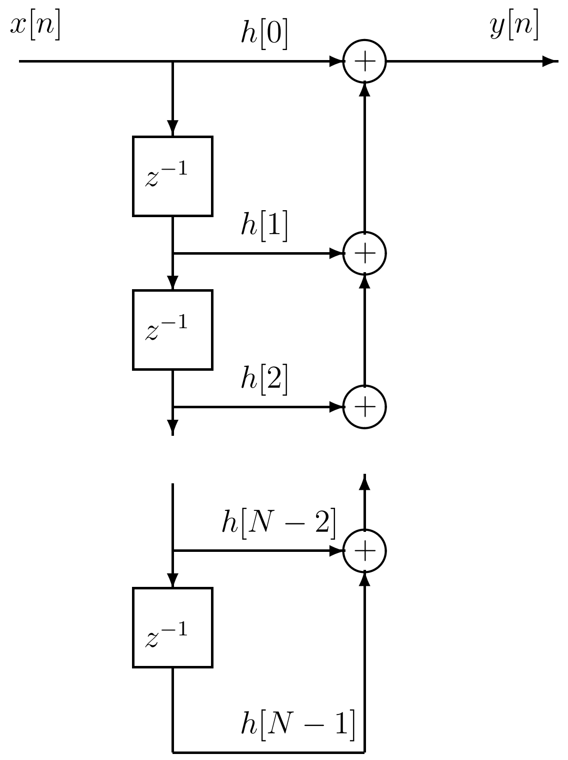

Fig. 3.1. Type 1 Direct Form Realization

Digital Signal Processing [II.C.5] and modified to make it more user friendly. The program presents a selection of seven different window types. The Hamming window (3) and Kaiser window (6) are the ones you will most likely find best. Five different filter types are available: (1) lowpass, (2) highpass, (3) bandpass, (4) bandstop, and (5) bandpass Hilbert transform. To use this program, first copy it to your working directory and then type **WINDOW**. As an example of how to use the program, suppose the sampling rate is 8 kHz, and a 21 tap bandpass filter with a lower cutoff frequency of 1 kHz and an upper cutoff frequency of 3 kHz is desired using the Hamming window. The program screen output and typical responses are shown below. The text written by the program is in capital letters and the user responses are shown in lower case letters but they can be entered as either capital or lower case letters. The lower case functions after window types 3, 4, and 5 are displayed by the program.

```
ENTER NAME OF LISTING FILE: junk.lst
ENTER FILENAME FOR COEFFICIENTS: junk.cof
ENTER SAMPLING FREQUENCY IN HZ: 8000
       WINDOW TYPES
    1     RECTANGULAR WINDOW
    2     TRIANGULAR WINDOW
    3     HAMMING WINDOW   0.54 + 0.46 cos(theta)
    4     GENERALIZED HAMMING WINDOW alpha + (1-alpha) cos(theta)
    5     HANNING WINDOW   0.5 + 0.5 cos(theta)
    6     KAISER (I0-SINH) WINDOW
    7     CHEBYSHEV WINDOW

       FILTER TYPES
    1     LOWPASS FILTER
    2     HIGHPASS FILTER
    3     BANDPASS FILTER
    4     BANDSTOP FILTER
    5     BANDPASS HILBERT TRANSFORM

ENTER FILTER LENGTH, WINDOW TYPE, FILTER TYPE: 21,3,3
SPECIFY LOWER, UPPER CUTOFF IN HZ: 1000,3000

CREATE (FREQUENCY,RESPONSE) FILE (Y OR N)?  y
ENTER FILENAME: junk.dat
LINEAR (L) OR DB (D) SCALE ?: d
```

The `LISTING FILE` is where the number of taps, filter type, window type, sampling frequency, and filter coefficients are written. The `FILENAME FOR COEFFICIENTS` has one entry per line. The first line is the number of coefficients. The remaining lines are the coefficients in order of increasing index and in floating point format. This file is useful for using the coefficients in another program. The `(FREQ, RESPONSE)` file is a listing of the amplitude response of the filter on a linear or dB scale. Each line contains a pair of numbers consisting of the frequency and corresponding amplitude response. The frequency increment is automatically selected to show the ripples in the amplitude response. If the program **GRAPHER** is going to be used to plot the response, this file should be given the extension **DAT**.

The second program for designing digital filters is **REMEZ87.EXE**. It is a modified version of the program in the IEEE book, *Programs for Digital Signal Processing.* The program was developed by J. McClellan and T. Parks who were at Rice University at the time. It uses the Remez algorithm to design filters that are optimum in the Chebyshev sense, that is, the maximum absolute error is minimized and causes the error to be equal ripple. The program can design (1) multiple passband/stopband filters, (2) differentiators, and (3) Hilbert transform filters.

Use of **REMEZ87** will be demonstrated by an example. Suppose the sampling rate is 8 kHz and a 21 tap bandpass filter with a passband extending from 1000 to 3000 Hz is desired. Using the conventions of this program, three bands must be specified. They are: (1) a lower stopband, (2) the passband, and (3) an upper stopband. Let the lower stopband edges extend from 0 to 500 Hz, the passband edges extend from 1000 to 3000 Hz, and the upper stopband edges extend from 3500 to 4000 Hz. Values must be specified for the amplitude in each of the bands. Let the values in the two stopbands be 0 and the value in the passband be 1. Also, weights for each band must be specified. The weight values scale the error in each band. Since the algorithm generates an equal ripple weighted error, larger weights result in bands with smaller unweighted ripple. For this example let the bands be equally weighted with the value 1.

REMEZ87 uses a variable `GRID DENSITY` to determine the frequency increment for computing the frequency response error with larger numbers corresponding to closer spaced frequencies. Values in the range of 16 to 32 seem to work well with little difference observed in the results. The smaller number requires less computation and uses smaller arrays. The program computes the frequency response of the resulting filter and asks the user to enter the lower and upper frequency limits to use for the response. Program screen prompts are in capital letters. User responses can be in either upper or lower case. The prompts and responses for this example are shown below.

```
ENTER LISTING FILENAME: junk.lst
ENTER COEFFICIENT STORAGE FILENAME: junk.cof
LINEAR OR DB AMPLITUDE SCALE FOR PLOTS? (L OR D): d
ENTER SAMPLING FREQUENCY (HZ): 8000
ENTER START AND STOP FREQUENCIES IN HZ FOR
   RESPONSE CALCULATION (FSTART,FSTOP): 0,4000

FILTER TYPES AVAILABLE:
  1 MULTIPLE PASSBAND/STOPBAND FILTER
  2 DIFFERENTIATOR
  3 HILBERT TRANSFORM

ENTER: FILTER LENGTH, TYPE, NO. OF BANDS, GRID DENSITY: 21,1,3,32
ENTER THE BAND EDGES (FREQUENCIES IN HERTZ)
0,500,1000,3000,3500,4000
SPECIAL USER DEFINED AMPLITUDE RESPONSE(Y/N)? n
SPECIAL USER DEFINED WEIGHTING FUNCTION(Y/N)? n

ENTER (SEPARATED BY COMMAS):
  1. VALUE FOR EACH BAND FOR MULTIPLE PASS/STOP BAND FILTERS
  2. SLOPES FOR DIFFERENTIATOR (GAIN = Ki*f -> SLOPE = Ki
      WHERE Ki = SLOPE OF i-TH BAND, f IN HERTZ)
  3. MAGNITUDE OF DESIRED VALUE FOR HILBERT TRANSFORM
0,1,0
ENTER WEIGHT FOR EACH BAND. (FOR A DIFFERENTIATOR
 THE WEIGHT FUNCTION GENERATED BY THE PROGRAM FOR THE i th
  BAND IS WT(i)/f WHERE WT(i) IS THE ENTERED BAND WEIGHT AND
  f IS IN HERTZ.)
1,1,1

STARTING REMEZ ITERATIONS
DEVIATION =     .159436E-03
   .
   .
   .
CALCULATING IMPULSE RESPONSE
CALCULATING FREQUENCY RESPONSE

CREATE (FREQ,RESPONSE) FILE (Y OR N)? y
ENTER FILENAME: junk.dat
```

The files requested by **REMEZ87** are essentially the same as for **WINDOW**. However, the `LISTING` file contains more information, such as, the frequencies where the peak errors occur, a frequency response listing in linear and dB form, and a crude plot of the response used in the days when only line printers without graphics capabilities were available.

REMEZ87 asks if you want a SPECIAL USER DEFINED AMPLITUDE RESPONSE or a SPECIAL USER DEFINED WEIGHTING FUNCTION. You can write your own special subroutines for a special desired amplitude response and/or weighting function and link them into the main program. No special functions are included in this version of the program.

3.2.3 Using Circular Buffers to Implement FIR Filters

For an N-tap FIR filter with coefficients nonzero only for indices in the set $\{0, \dots, N-1\}$, the convolution sum (3.1) becomes

$$y[n] = \sum_{k=n-(N-1)}^{n} x[k]h[n-k] = x[n-(N-1)]h[N-1] + \cdots + x[n-1]h[1] + x[n]h[0] \quad (3.8)$$

Notice that the oldest input sample $x[n-(N-1)]$ is multiplied by the impulse response sample $h[N-1]$ with the largest index and the newest sample $x[n]$ is multiplied by the impulse response sample $h[0]$ with the smallest index. This equation is shown schematically in Fig. 3.1 where the N required signal samples are shown stored in a delay line. In a software implementation, the delay line represents an array in memory. Entering the newest sample into the "delay line" by shifting the elements in the entire array is inefficient for a software implementation and a better approach is to use *circular buffers*. Circular buffers can be implemented in C by using the mod operator %. The TMS320C30 has hardware support that can be accessed by assembler instructions to implement circular buffers even more efficiently.

The concept of a circular buffer is illustrated in Fig. 3.2 on page 46. The filter coefficients are stored in the N element array `hr` in reverse order, that is, $\texttt{hr}[k] = h[N-1-k]$ for $k = 0, \dots, N-1$. A variable, *oldest*, points to the location in the circular buffer array that contains the oldest shift register sample. When a new sample is received at time n, it is written over the sample at location *oldest*. In a physical implementation using a shift register, this sample would be shifted out of the end of the shift register when the new one is shifted in. Next, the variable *oldest* must be incremented modulo N to point to the new oldest sample. Notice that when *oldest* initially has the value $N-1$, it becomes 0 when incremented modulo N. Thus, data samples are written into the array in a circular fashion. Finally, the filter output can be calculated as

$$y[n] = \sum_{k=0}^{N-1} \texttt{hr}[k]\texttt{xcirc}[(oldest + k) \bmod N] \quad (3.9)$$

```
         .word  _hr           ; Base address of filter coefficients
         .word  _xcirc        ; Base address of circular buffer

;  Now begin convolution function instructions

         .text
         .global  _convol     ; Make entry point available to C.

_convol  LDI N,BK             ; Set circular buffer block size.
         LDF 0.0,R2           ; Clear convolution sum.
         LDI @ADDRESSES+0,AR0 ; Make AR0 point to hr[0].
         LDI @ADDRESSES+1,AR1 ; Make AR1 point to xcirc[0].
         ADDI @_oldest,AR1    ; Make AR1 point to xcirc[oldest].

         MPYF3 *AR0++,*AR1++%,R0 ; Put hr[0]*xcirc[oldest] in R0.

         RPTS RC1             ; Repeat next instruction RC1+1 times.
         MPYF3 *AR0++,*AR1++%,R0 ; put hr[i]*xcirc[(oldest+i)%N] in R0.
||       ADDF3 R0,R2,R2       ; Accumulate previous product in R2.

         ADDF3 R0,R2,R0       ; Add final product to R2 and return
                              ; result to C in R0.
         RETS                 ; Return to C
         .END
```

3.3 Infinite Duration Impulse Response (IIR) Filters

A filter with an impulse response, $h(n)$, that has infinite duration is known as an IIR filter. When $h(n)$ is the sum of damped exponentials, its **z**-transform, $H(z)$, which is also called its transfer function, is a rational function of z. That is, it is the ratio of two finite degree polynomials. We will use a rational function of the form

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} = \frac{B(z)}{A(z)} \tag{3.10}$$

3.3.1 Realizations for IIR Filters

Rational transfer functions can be realized in many ways. Three common realizations will be described below. The first realization will be called a **type 0 direct form**. The ratio of the **z**-transforms of the filter output and input is

$$\frac{Y(z)}{X(z)} = H(z) = \frac{B(z)}{A(z)} \tag{3.11}$$

Cross multiplying gives

$$Y(z)A(z) = X(z)B(z) \quad \text{or} \quad Y(z)\left(1 + \sum_{k=1}^{M} a_k z^{-k}\right) = X(z)\sum_{k=0}^{N} b_k z^{-k} \tag{3.12}$$

Taking all except the $Y(z)$ term to the righthand side yields

$$Y(z) = \sum_{k=0}^{N} b_k X(z) z^{-k} - \sum_{k=1}^{M} a_k Y(z) z^{-k} \tag{3.13}$$

The time domain equivalent is the difference equation

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{M} a_k y[n-k] \tag{3.14}$$

This equation shows how to compute the current filter output from the current and N past inputs and M past outputs. A filter implemented in this way is also called a *recursive* filter since past outputs are used to calculate the current output. It is called a direct form because the coefficients in the transfer function appear directly in the difference equation.

Another realization which we will call a **type 1 direct form** is based on observing that (3.11) can be rearranged into the cascade form

$$Y(z) = \frac{X(z)}{A(z)}B(z) = V(z)B(z) \tag{3.15}$$

where

$$V(z) = X(z)\frac{1}{A(z)} \tag{3.16}$$

This is illustrated in Fig. 3.3.

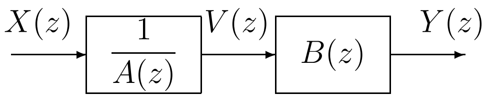

Fig. 3.3. First Step in Finding Type 1 Direct Form Realization

The intermediate signal $v[n]$ can be computed using the direct form 0 realization

$$v[n] = x[n] - \sum_{k=1}^{M} a_k v[n-k] \tag{3.17}$$

Then, the output can be computed as

$$y[n] = \sum_{k=0}^{N} b_k v[n-k] \tag{3.18}$$

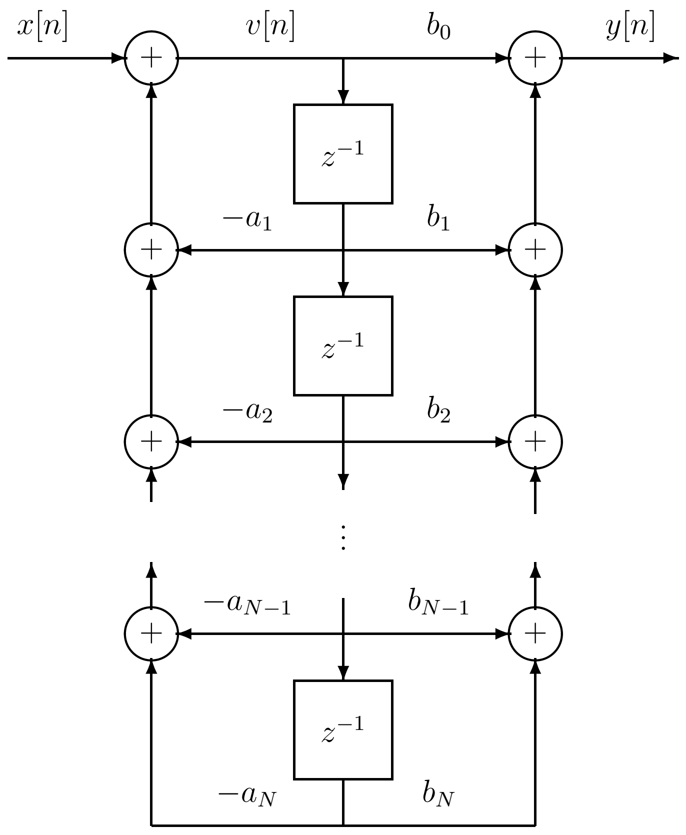

Fig. 3.4. Type 1 Direct Form Realization

A block diagram for these equations is shown in Fig. 3.4 on page 57 where it is assumed that $M = N$. This form requires less storage than the type 0 direct form.

Another realization called the **type 2 direct form** can be found by rearranging (3.12). For simplicity, let $M = N$. Then

$$Y(z) = b_0X(z) + \sum_{k=1}^{N}[b_kX(z) - a_kY(z)]z^{-k} \tag{3.19}$$

A block diagram for this realization is shown in Fig. 3.5 on page 58. It requires essentially the same storage and arithmetic as direct form 1.

3.3.2 A Program for Designing IIR Filters

The program, **C:\DIGFIL\IIR.EXE**, designs IIR filters by using the bilinear transformation [II.C.14, pp. 212-219] with a Butterworth, Chebyshev, inverse Chebyshev, or elliptic analog prototype filter. It can design lowpass, highpass, bandpass, or bandstop filters. The form of the resulting filter is a cascade (product) of sections, each with a second order numerator and denominator with the leading constant terms normalized to 1, possibly a first order section normalized in the same way, and an overall scale factor. These second order

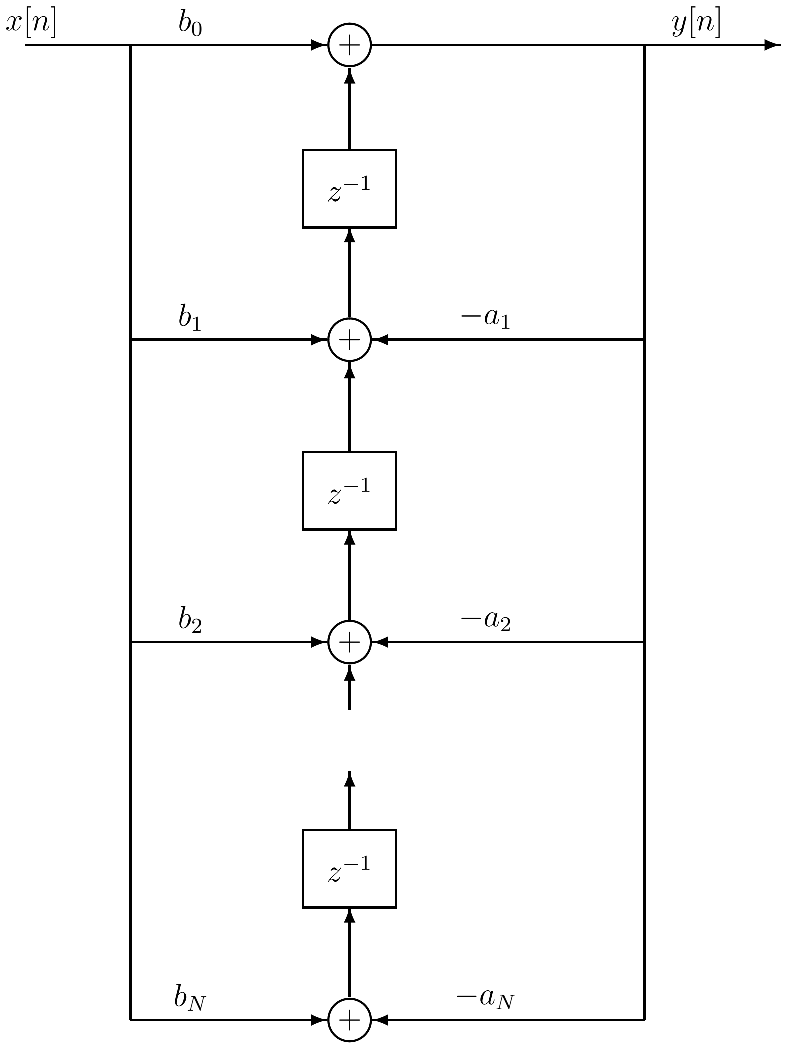

Fig. 3.5. Direct Form 2 Realization

sections are also know as biquads. The sections can be realized by any of the three direct forms described above or other structures that can be found in DSP books.

Care must be taken to prevent overflows and underflows when digitial filters are implemented with fixed point DSP's. This problem is significantly reduced with floating point DSP's. Sometimes the overall scale factor generated by **IIR** is quite small and to maintain numerical accuracy it should be split among the different sections.

An example of how to use **IIR** is shown below. The program prompts are shown in upper case letters and the user responses in lower case letters or numbers. In this example, a bandpass filter is designed based on an elliptic analog prototype filter. The nominal lower stopband extends from 0 to 600 Hz, the passband extends from 1000 to 2000 Hz, and the upper stopband extends from 3000 to 4000 Hz. The questions and answers are explained more fully after the dialog.

```
SAVE RESULTS IN A FILE (Y OR N): y
```

```
ENTER LISTING FILENAME: junk.lst
ENTER 1 FOR ANALOG, 2 FOR DIGITAL: 2
ENTER SAMPLING RATE IN HZ: 8000
ENTER NUMBER OF FREQS TO DISPLAY: 100
ENTER STARTING FREQUENCY IN HZ: 0
ENTER STOPPING FREQUENCY IN HZ: 4000
ENTER 1 FOR BW, 2 FOR CHEBY, 3 FOR ICHEBY, 4 FOR ELLIPTIC: 4
ENTER 1 FOR LOWPASS, 2 FOR HP, 3 FOR BP, OR 4 FOR BR: 3
ENTER F1,F2,F3,F4 FOR BP OR BR FREQS: 600,1000,2000,3000
ENTER PASSBAND RIPPLE AND STOPBAND ATTENUATION IN +DB: 0.2,40

ELLIPTIC FILTER ORDER =      4

CREATE FREQ, LINEAR GAIN FILE (Y,N)? n
CREATE FREQ, DB GAIN FILE (Y,N)? Y
ENTER FILENAME: junkdb.dat
CREATE FREQ, PHASE FILE (Y,N)? n
CREATE FREQ, DELAY FILE (Y,N)? y
ENTER FILENAME: JUNKDEL.DAT
```

The first line of the dialog asks if you want to save the results in in a disk file. If the answer is Y or y, you are prompted for the name of a file. If the answer is N or n, the results appear on the screen (usually too fast to be read). The program computes the frequency response of the designed filter at the number of points specified which are equally spaced over the range of frequencies selected. You are then prompted for the type of analog prototype filter desired and the frequency selectivity type of the digital filter. In the case of a bandpass (BP) filter, four critical frequencies, $F1 < F2 < F3 < F4$, must be entered. The frequency $F1$ is the upper edge of the lower stopband, $F2$ is the lower edge of the passband, $F3$ is the upper edge of the passband, and $F4$ is the lower edge of the upper stopband. In the case of an elliptic filter, you are then prompted for the desired maximum passband ripple and the minimum stopband attenuation. The program then computes the order of the required analog lowpass prototype filter which in this example is 4. The actual order of the digital filter is double this number for bandpass and band reject filters. The user is given the option of choosing the filter order or letting **IIR** choose the order for some of the other prototype filters. Finally you are prompted for the types of frequency response files you wish to generate which can then be plotted with your favorite graphing program.

The `RESULTS` file for this example is shown below. First, the z-plane zeros and poles are displayed in rectangular form. Then they are shown in polar form. The radius is the magnitude of the pole or zero and the frequency is $f_s\theta/(2\pi)$ where θ is the angle and f_s is the sampling frequency. Notice, that for this bandpass filter, the zeros are all exactly on the unit circle with frequencies in the stop bands. The pole frequencies are in the passband.

The coefficients of the numerators and denominators of the second order sections are given and they can be realized by the direct forms. It is shown in many DSP books that it is computationally better to realize an IIR filter by splitting it into low order sections rather than by one high order section.

Finally, the amplitude response on a linear scale, the amplitude response on a dB scale, the phase response, and the envelope delay are listed for the chosen range. This data also appears in separate files if selected in the dialog.

```
                      DIGITAL BANDPASS ELLIPTIC FILTER

                           FILTER ORDER =  8

                            Z PLANE

           ZEROS                              POLES

         .977149 +- j  .212554      .173365 +- j  .761580
         .902015 +- j  .431705     -.028463 +- j  .919833
        -.538154 +- j  .842847      .683010 +- j  .651915
        -.873779 +- j  .486323      .482595 +- j  .656484

         RADIUS        FREQUENCY         RADIUS       FREQUENCY

       .100000E+01     .272712E+03     .781063E+00     .171502E+04
       .100000E+01     .568352E+03     .920273E+00     .203939E+04
       .100000E+01     .272351E+04     .944190E+00     .970348E+03
       .100000E+01     .335335E+04     .814782E+00     .119288E+04

             4 CASCADE STAGES, EACH OF THE FORM:

F(z) = ( 1 + B1*z**(-1) + B2*z**(-2) ) / ( 1 + A1*z**(-1) + A2*z**(-2) )

         B1           B2                A1           A2
     -1.954298     1.000000           -.346731      .610059
     -1.804029     1.000000            .056927      .846903
      1.076307     1.000000          -1.366019      .891495
      1.747559     1.000000           -.965191      .663870

  SCALE FACTOR FOR UNITY GAIN IN PASSBAND:   1.8000479016654E-002


                     FREQUENCY RESPONSE

     FREQUENCY      GAIN        GAIN (dB)      PHASE      DELAY (SEC)

          .0000   2.1048E-03  -5.3536E+01      .00000   .13458E-03
        40.0000   2.0557E-03  -5.3741E+01     -.03385   .13493E-03
        80.0000   1.9093E-03  -5.4382E+01     -.06789   .13600E-03
       120.0000   1.6681E-03  -5.5556E+01     -.10228   .13780E-03
```