Application Software of the Future-Filter Design with Gem

1985

Alan D. George
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd
University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
APPLICATION SOFTWARE OF THE FUTURE —
FILTER DESIGN WITH GEM

BY

ALAN DALE GEORGE
B.S., University of Central Florida, 1982

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

Fall Term
1985
ABSTRACT

As the use of computers in engineering design as well as other areas increases, it becomes more imperative that the application software used be as simple, convenient, and powerful as possible. The engineer is not interested in the internal workings of the computer or its operating system. It is the design itself that takes precedence. The filter design package developed for this project, known as FILTER, is such an application. With FILTER, coupled with the Digital Research Graphics Environment Manager, the engineer is led through the analog and digital filter design phase on a personal computer with carefully designed interactive computer graphics requiring little or no computer knowledge.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Signals</td>
<td>1</td>
</tr>
<tr>
<td>Filters</td>
<td>2</td>
</tr>
<tr>
<td>The Approach</td>
<td>3</td>
</tr>
<tr>
<td>2. THE GRAPHICS ENVIRONMENT MANAGER (GEM)</td>
<td>4</td>
</tr>
<tr>
<td>Brief History of GEM Type Systems</td>
<td>4</td>
</tr>
<tr>
<td>An Introduction to GEM</td>
<td>5</td>
</tr>
<tr>
<td>The Desktop, Mouse, and Icons</td>
<td>5</td>
</tr>
<tr>
<td>Drop-Down Menus</td>
<td>7</td>
</tr>
<tr>
<td>Windows</td>
<td>7</td>
</tr>
<tr>
<td>3. USING THE PACKAGE</td>
<td>9</td>
</tr>
<tr>
<td>Analog Design</td>
<td>9</td>
</tr>
<tr>
<td>Analog Analysis</td>
<td>11</td>
</tr>
<tr>
<td>Digital Transformation</td>
<td>11</td>
</tr>
<tr>
<td>Digital Analysis</td>
<td>12</td>
</tr>
<tr>
<td>Desk</td>
<td>13</td>
</tr>
<tr>
<td>File</td>
<td>13</td>
</tr>
<tr>
<td>4. THE DEVELOPMENT</td>
<td>14</td>
</tr>
<tr>
<td>5. COMPUTER DESCRIPTION</td>
<td>20</td>
</tr>
<tr>
<td>6. DESIGN EXAMPLE</td>
<td>22</td>
</tr>
<tr>
<td>7. CONCLUSION</td>
<td>24</td>
</tr>
</tbody>
</table>

Appendix

FILTER SOURCE LISTING | 26 |

LIST OF REFERENCES | 52 |
CHAPTER 1
INTRODUCTION

Signals

The study of signals is important to almost every field of science and engineering. Examples include acoustics, biomedical engineering, communications, control systems, radar, physics, seismology, and telemetry. Signals can be classified into two classes, namely continuous-time signals and discrete-time signals.

A continuous-time signal is a signal that exists and is defined at every instant of time, like a continuous function. Examples of continuous-time signals include the waveform of a voltage or current. On the other hand, a discrete-time signal is a signal that is only defined at certain discrete instances of time, such as every second, minute, or hour. Examples of discrete-time signals include the balance of a checking account at a bank or the daily rainfall amount for a particular place.

Both continuous-time and discrete-time signals can be represented by a unique function of frequency referred to as the frequency spectrum of that signal. This frequency
spectrum is a description of the frequency content of the signal.

**Filters**

Filtering is a process used to modify, reshape, or manipulate in any fashion the frequency spectrum of a signal. It can include amplifying or attenuating a range of frequency components, rejecting or isolating certain frequency components, or a combination of the two. The filtering process is used for such things as reducing the signal contamination due to noise, reducing the signal distortion caused by transmission problems, separating mixed signals, demodulating signals, and limiting to or stopping a certain frequency band range.

The filter designing process used in this project begins with the analog filter design based on classical analog filter approximation techniques such as the Butterworth, Chebyshev, and Bessel approximations. A series of routines are available to analyze the analog filter produced to ensure the desired results are realized. If a digital filter is desired, a series of transformation routines are available to translate the analog filter to a similar digital filter, such as the bilinear transform. These transformation techniques are based on a user-defined sample frequency or sample period. A series of routines are then available to analyze
the digital filter produced to ensure the desired results are realized.

The Approach

What makes this project unique is the ease of use of the FILTER package. This user-friendliness is achieved with carefully designed interactive graphics software and interaction with the Graphics Environment Manager. With this approach, the computer knowledge needed by the designer is greatly decreased.
CHAPTER 2
THE GRAPHICS ENVIRONMENT MANAGER (GEM)

Brief History of GEM Type Systems

During the 1970s, a group of computer professionals at the Xerox Palo Alto Research Center (PARC) did research into possible ways to improve the interface between computers and people. From this research came the first implementations of human and computer interaction based on images instead of command languages. Their computer, the Xerox Star, was truly the first of its kind. It incorporated the many tools associated with a graphic object oriented system. However, the price of $20,000 for the system left it out of reach of the average computer user.

The next major step in bringing a truly user-oriented computer environment to the average user came in the early 1980s, when Apple Computer Incorporated introduced their object-oriented computer system, the Lisa. However, it too was priced out of the reach of the average computer user. Recognizing this, Apple developed and released the Macintosh
computer system, a system very similar to the Lisa but at a much more affordable price. While the Macintosh computer was one of the first real user-oriented personal computers of its kind and thus a breakthrough, it was seen that the environment on this computer had shortcomings, among them portability between computers, color graphics, and speed.

Then, in 1984, Digital Research Incorporated (DRI) released the Graphics Environment Manager (GEM), a product with a graphic object oriented environment similar to its predecessors, but with some distinct new advantages. These included color graphics capability, portability between some of the most powerful and popular personal computers (including the IBM PC), speed, and cost. This GEM environment would not only be available on faster and more powerful new machines, but also older and more established computers.

An Introduction to GEM

The GEM user environment can be divided into three areas: the desktop, mouse and icons; drop-down menus; and windows.

The Desktop, Mouse, and Icons

When the GEM environment is first loaded (the details of which are dependent on the machine being used), the desktop
will appear filling the screen. On the desktop will appear three icons, which are graphical representations of objects available for use. Two of the icons are pictures of filing cabinet drawers, and they represent disk drives. The third icon is a picture of a trash can. It is this icon that allows the user to dispose of unwanted information.

To manipulate objects on the desktop, there is a device connected to the computer called a mouse. The mouse is a small box with two buttons on the top and a small ball underneath, connected by a wire to a port in the computer. To use the mouse, it must be placed on a flat surface. Then, as the mouse is moved around the surface, a small black arrow moves accordingly around the GEM desktop on the computer display. This arrow, referred to as the mouse pointer, is used to point to objects to be acted upon.

There are three fundamental operations that can be performed on icons with the mouse: selection, activation, and dragging. To select an icon, simply move the mouse until the mouse pointer is positioned on the icon desired. Then press the left hand button on the mouse. The icon is thus selected, signified by the icon changing color. To activate an icon, use the same procedure as selecting, except press the button twice in quick succession instead of once (this is know as double-clicking). To drag or move an icon about the desktop, use the same procedure as selecting,
except hold the button down. With the button held down, move the mouse to the desired new position, and release the button to drop the icon there.

Drop-Down Menus

Across the top of the GEM desktop are the names of the menus available. To see the operations available within a particular menu, simply point the mouse to that menu. When the mouse pointer comes in contact with the menu name, the menu’s contents will drop down below the menu name for viewing. To select one of the menu items, simply position the mouse pointer over the item and press the left button, similar to selecting an icon. The menu items can appear in two forms, available and unavailable. A menu item is available when it appears dark, and is unavailable when it appears very light. In this project, items for future implementation appear unavailable, but are included for completeness.

Windows

With most conventional computer environments, information is entered and displayed about the entire screen. With GEM, this activity takes place through objects called windows. Windows can be considered small rectangular screens within the computer screen itself. Windows were designed to
improve data input/output (I/O) organization, control, and convenience. Windows are created by many different actions, for example, activation of an icon and selection of a menu item.

Each window is made up of any combination of the following components: the window name, the move bar, the close box, the full box, the size box, the vertical scroll area, and the horizontal scroll area. The window name is located at the top center of the window, and consists of an alphanumeric name for the window. The move bar is a shaded rectangle across the top of the window and is used for moving the window about the GEM desktop. The close box is located at the upper left corner of the window and is used to close the window. The full box is located at the upper right corner of the window and is used to expand the size of the window to full screen automatically. The size box is located at the bottom right corner of the window and is used to change the size of the window. The vertical scroll area is located along the right edge of the window, and it is used to move information vertically in the window. The horizontal scroll area is located across the bottom of the window and is used to move information horizontally in the window. These items are all activated by selecting them with the mouse and dragging when appropriate.
CHAPTER 3
USING THE PACKAGE

The filter design application software package designed and developed for this research project consists of four main menus: Analog Design; Analog Analysis; Digital Transformation; and Digital Analysis. In addition, two support menus are included: Desk and File. To display the contents of any of these menus in the FILTER package, simply drop down the appropriate menu on the FILTER desktop.

**Analog Design**

The first main menu of the FILTER package is the Analog Design menu. In this menu, the user has the capability of choosing analog prototype filter designs based on classical approximation techniques, or the option of entering a user-defined analog filter transfer function. This menu consists of five items: Butterworth filter design; Chebyshev filter design; Inverted Chebyshev filter design; Bessel filter design; and user-entered filter transfer functions.
The Butterworth filter design is based on the classical Butterworth filter, which is characterized by maximally flat gain in the passband. The Chebyshev and Inverted Chebyshev filter designs are based on the classical Chebyshev filters, also known as the Chebyshev Type I and Type II filters, which are characterized by ripple in the passband and stopband respectively, and improved cutoff characteristics. The Bessel filter design is based on the classical Bessel filter, which is characterized by maximally flat delay. In all of these designs, the user is prompted for the type of filter (high pass or low pass), the passband radian frequency, the maximum attenuation at the passband, the stopband radian frequency, the minimum attenuation at the stopband, and the order of the filter. Upon entering these values, the analog filter is instantaneously generated, ready for analog analysis.

The user-entered analog filter transfer function item is used to enter a predetermined analog filter transfer function into the FILTER package for subsequent Analog Analysis, Digital Transformation, and Digital Analysis. The analog transfer function is entered in the following coefficient type polynomial form:

\[ H(s) = K_a \cdot \frac{a_m s^m + a_{m-1} s^{m-1} + \ldots + a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} + \ldots + b_1 s + b_0} \]
Analog Analysis

In the Analog Analysis menu, the user has the capability of displaying frequency and time response plots of the analog filter previously designed or entered. This menu consists of four plotting routines: a magnitude versus radian frequency plot; a phase versus radian frequency plot; an impulse response versus time plot; and a step response versus time plot. With the magnitude plot, the user has the option of displaying either the voltage gain ($|H(j\omega)|$) or the power gain ($|H(j\omega)|^2$). Both frequency response plots request the user for the starting and ending frequencies to display.

Digital Transformation

In the Digital Transformation menu, the user has the capability of choosing common analog to digital filter transformation algorithms or the option of entering a user-defined digital filter transfer function. This menu consists of five items: Bilinear Transformation with exact cutoff frequency match; Bilinear Transformation without exact cutoff frequency match; Impulse Invariant Transformation; and Step Invariant Transformation. With all of these transformations, the user is prompted for the desired sample period or sample frequency. Upon entering this value, the digital filter is instantaneously generated, ready for digital analysis.
The user-entered digital filter transfer function item is used to enter a predetermined digital filter transfer into the FILTER package for subsequent Digital Analysis. The digital transfer function is entered in the following coefficient type polynomial form:

\[ H(z) = K_d \cdot \frac{c_m z^m + c_{m-1} z^{m-1} + \ldots + c_1 z + c_0}{d_n z^n + d_{n-1} z^{n-1} + \ldots + c_1 z + c_0} \]

**Digital Analysis**

In the Digital Analysis menu, the user has the capability of displaying frequency and time response plots of the digital filter previously designed or entered. This menu consists of four plotting routines: a magnitude versus radian frequency plot; a phase versus radian frequency plot; a pulse response versus time plot; and a step response versus time plot. With the magnitude plot, the user has the option of displaying either the voltage gain (\(|H(e^{j\omega T})|\)) or the power gain (\(|H(e^{j\omega T})|^2\)). Both frequency response plots request the user for the starting and ending frequencies to display. Both time response plots graph the first 100 samples.
Desk

In the Desk menu, the user has access to the FILTER information item as well as numerous system desktop utilities. The FILTER information simply consists of general information about the package, such as the author and date of release. The utilities present are dependent on the computer being used, but normally consist of such items as printer configuration, modem port configuration, and the system control panel, all of which allow the user to change system specific setup characteristics without having to exit the FILTER package.

File

In the File menu, the user has the capability of storing and retrieving both analog and digital filters. Upon selecting either the save or load item, the user is prompted whether to use the analog or the digital filter in memory. Then a special file selector box appears allowing the user to examine the disk directory before choosing a name for the file.
CHAPTER 4
THE DEVELOPMENT

The development of this project began with the development of a design philosophy. The philosophy chosen was based on ease of use. It was desired to create a package whereby practically anyone could design and analyze filters, both analog and digital. In addition, it was desired to implement this package on a microcomputer, thereby making it more accessible to the average user. Once this philosophy was chosen, the next step was to choose a machine. After careful analysis of the market, the Atari 520ST computer was chosen, based on price, performance, user environment, and availability. In fact, the GEM environment included with the computer is ideally suited for applications emphasizing ease of use.

Once the machine and the environment had been chosen, the next step was to choose the programming language with which to implement the software. At the time, five languages were available. These included BASIC, LOGO, C, FORTH, and 68000 Assembly language. The C language was chosen for its
speed, portability, power, and compatibility with the GEM-callable libraries. The next step was learning the language itself. C is one of the more conventional languages, not unlike PASCAL, FORTRAN, BASIC, or PL/I. It includes all of the most important structured programming control constructs such as 'while' statements with testing at either beginning or end, 'if-then-else' testing statements that can act upon blocks of code, a 'case' type statement similar to 'if-then-else' but more convenient for numerous testing values, and a 'for' iterative loop type statement. Due to its strong emphasis on functions, C is almost inherently modularized. It also contains numerous unusual facilities, among them pointer data types and bit-level functions, that make it well-suited for system applications. However, clear and well-written code can be sacrificed if care is not taken to ensure these facilities are not overly used.

The next step involved in development was learning the complexities of making C code executable on the 520ST. To compile a C program, the C preprocessor must first be called to process C macros in the code. Next the C parser is called to parse the C code output of the preprocessor. Then the C assembly language code generator is called to translate the parser output into 68000 assembly language. At this point, it is possible to edit the assembly code and make
modifications, an ideal situation for speed optimization of key routines. Then the 68000 assembler is called to assemble the code into machine code, after which the linker is called to link the various object code modules and resolve symbol table references. Then finally the relocation utility is called to take the linker relocatable output code and translate it into absolute addresses. It is at this point that the code is in executable image form.

The next step was to master some of the many capabilities of GEM programming, which consist of three key parts: the Resource Construction Set, the Virtual Device Interface, and the Application Environment Services.

The GEM Resource Construction Set (RCS) is a utility for designing and developing resources such as drop-down menus, dialog boxes, and alert boxes. These resources can be saved in files in a tree structure, to be referenced from the application source code. For this application, the root of the tree is the FILTER menu bar. Each of the items available from this menu are branches from this root.

The GEM Virtual Device Interface (VDI) is a graphics architecture made up of device drivers and functions. These include control functions to initialize the graphics workstation and set default values, output functions for graphics primitives, attribute functions to determine output qualities such as color and style, raster functions for logic
operations on raster areas, input functions, inquire functions to check the device settings, and escape functions for accessing special device capabilities.

The GEM Application Environment Services (AES) consists of numerous routines for interacting with the application program. These include the application library for setting up the application, the event library for servicing application interrupts, the menu library for menu maintenance, the object library for object maintenance, the form library for form application management, the graphics library for window and mouse graphics options, the scrap library for data interchange between applications, the file selector library for application and disk directory interaction, the window library for window management, the resource library for communicating with resources from the RCS, and the shell library for invoking other applications.

The final development step before implementation was deriving the filter design and analysis algorithms. While based on standard textbooks, all of the algorithms used in this project were derived by the author. While some of these algorithms may not be as optimized for speed as the others available, they have helped with a more thorough understanding of the subject. A detailed analysis of all of the many algorithms involved would be beyond the scope of
this research paper. However, one such typical algorithm, analog frequency response, is shown for completeness.

The analog transfer function is expressed in the following form:

$$H(s) = K_a \frac{a_m s^m + a_{m-1} s^{m-1} + \ldots + a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} + \ldots + b_1 s + b_0}$$

Substituting in "jw" for "s" gives:

$$H(jw) = K_a \frac{a_m (jw)^m + a_{m-1} (jw)^{m-1} + \ldots + a_1 (jw) + a_0}{b_n (jw)^n + b_{n-1} (jw)^{n-1} + \ldots + b_1 (jw) + b_0}$$

Then define:

$$A = a_0 - a_2 w^2 + a_4 w^4 - \ldots \quad , \quad B = a_1 w^1 - a_3 w^3 + a_5 w^5 - \ldots$$

$$C = b_0 - b_2 w^2 + b_4 w^4 - \ldots \quad , \quad D = b_1 w^1 - b_3 w^3 + b_5 w^5 - \ldots$$

Which gives:

$$H(jw) = K_a \frac{A + jB}{C + jD}$$

Multiplying numerator and denominator by (C-jD) gives:

$$H(jw) = K_a \frac{A + jB}{C + jD} \frac{C - jD}{C - jD}$$
\[ \frac{AC + BD}{C^2 + D^2} + \frac{BC - AD}{C^2 + D^2} = K_a \cdot \frac{1}{C^2 + D^2} + \frac{1}{C^2 + D^2} \]

Then define:

\[ X = K_a \cdot \frac{AC + BD}{C^2 + D^2}, \quad Y = K_a \cdot \frac{BC - AD}{C^2 + D^2} \]

Which gives:

\[ H(jw) = X + jY \]

Therefore:

\[ |H(jw)| = \left( X^2 + Y^2 \right)^{1/2} = \text{magnitude} \]

\[ \angle H(jw) = \tan^{-1}\left( \frac{Y}{X} \right) = \text{phase} \]
CHAPTER 5
COMPUTER DESCRIPTION

The computer chosen for the implementation of this project was the Atari 520ST. This machine was chosen for its user environment, performance, availability, and cost.

The Atari 520ST computer was first announced in January of 1985, and became widely available by mid 1985. It is based on one the most popular 16/32-bit microprocessors on the market, the Motorola 68000, with a 16-bit external data bus, 32-bit internal registers, and a 24-bit address bus.

Some of the standard features of the Atari 520ST are: 512K of Random Access Memory (RAM); a floppy disk interface; a hard disk interface capable of 1.33 megabytes per second transfer rates; a Centronics-standard parallel port; an RS-232 serial port; a video and audio interface; an incoming and outgoing Musical Instrument Digital Interface (MIDI); a 128K Read Only Memory (ROM) cartridge slot; internal sockets for 192K of ROM; three graphics modes consisting of a 640x400 pixel high-resolution monochrome mode, a 640x200 medium-resolution, 4-color mode, and a 320x200 low-resolution,
16-color mode; 9-bit color registers enabling 512 color possibilities; a 3-channel sound chip; a full keyboard with full keypad, function keys, and cursor keys; a 2-button mouse input device; choice of a monochrome or RGB analog color monitor; the operating system with GEM; and the BASIC and LOGO programming languages.

The configuration chosen includes one monochrome monitor capable of 640x400 pixel resolution, one 700K double-sided 3.5-inch disk drive, one 350K single-sided 3.5-inch disk drive, and a dot matrix graphics printer. Software used consisted of a C compiler, an assembler, a linker, an editor, various utilities, and a full set of GEM libraries.

When considering the performance of the system, based on its microprocessor, support chips, and I/O throughput, the many options that come standard with the machine, and its incredibly low cost ($800 monochrome, $1000 color), it is clear that the Atari 520ST is one of the most outstanding computers on the market. No other computer on the market even close to this price range can offer such capabilities.
A third-order low pass Butterworth digital filter is to be designed. The passband is to be at 100 radians per second, with a maximum attenuation of 0.4 dB. The stopband is to be at 400 radians per second, with a minimum attenuation of 12.5 dB. Based on these specifications, the design was made using FILTER.

The design begins by choosing the type of analog filter to use, in this case Butterworth. When prompted for low or high pass, low pass is chosen. When prompted for the desired passband radian frequency, 100 is entered. When prompted for the desired maximum attenuation there, 0.4 is entered. When prompted for the stopband radian frequency, 400 is entered. When prompted for the minimum attenuation there, 12.5 is entered. When prompted for the desired order of the filter, 3 is entered. At this point, the computer generates the analog filter.

The next step is to analyze the results of the analog design. For this, the analog magnitude plot item is chosen.
When prompted for either the voltage or gain plot, the voltage plot is chosen. When prompted for the beginning radian frequency for plotting, 0 is chosen. When prompted for the ending radian frequency for plotting, 1000 is chosen, after which the plot appears. Satisfied with the results, the display function item is chosen to output the transfer function. The result given by FILTER is:

\[
H(s) = K \times \frac{1}{s^3 + 295.319223s^2 + 43606.748580s + 3219479.322433}
\]

\[K = 3219479.084014\]

The next step is to perform the transformation from analog to digital. In this case, the Bilinear Transformation without matching will be used. When prompted for either the sample period T or sample radian frequency \(\omega_{\text{sample}}\), the sample frequency is chosen, with a value of 2000. At this point, the computer generates the digital filter.

Next, to begin analyzing the digital filter, the digital magnitude plot is chosen, with a starting value of 0 and an ending value of 1000. Satisfied with this plot, the pulse and step response plots are chosen. Satisfied with these plots, the digital display function item is chosen to output the transfer function, which in this case is:

\[
H(z) = 0.007878 \times \frac{z^3 + 3z^2 + 3z + 1}{z^3 - 2.095293z^2 + 1.556829z - 0.398515}
\]
CHAPTER 7
CONCLUSION

This paper has presented the development of an application software package for analog and digital filter design. This package, coupled with the GEM user environment, increases design productivity by simplifying computer operations so that the emphasis of the designer can be placed on the design.

This package can be used for the entire digital filter design process, from analog design by classical approximation, to analog analysis, to digital transformation, to digital analysis. The package is also useful by just making use of the separate functions available, such as simply analyzing a user-defined analog or digital filter.

Further development and expansion of this package will continue into the future. Some of the possible improvements are: the completion of the currently unavailable items; the addition of more analog design approximation techniques, such as the Elliptic approximation; and the addition of more.
analog and digital analysis techniques. Emphasis will continue to be placed on ease of use and convenience.

The package developed for this project marks a breakthrough in microcomputer-based filter design. Productivity is greatly increased by minimizing design time, and as seen in the design example, this design time is reduced considerably compared to conventional design techniques, due to a greater emphasis on ease of use. Analog and digital filters can be designed and analyzed in a short period of time, and by users with little, if any, computer knowledge. As the use of computers in engineering as well as other areas increases, it becomes imperative that the application software be as simple, convenient, and powerful as possible. It seems clear that these goals will be realized in the future with software similar to FILTER and the GEM environment.
APPENDIX

FILTER SOURCE LISTING
/* Infinite Impulse Response Digital Filter Design Package */
/* By Alan D. George, for Research Project towards M.S. in */
/* Computer Engineering, Fall Semester, 1985. */

/**** INCLUDE FILES ****/
#include <stdio.h>
#include <vdibind.h>
#include <obdefs.h>
#include <define.h>
#include <gemdefs.h>
#include <osbind.h>
#include "filter.h"

/**** DEFINES ****/
#define WI_KIND (CLOSER|NAME)
#define NO_WINDOW (-1)
#define MIN_WIDTH (2*gl_wbox)
#define MIN_HEIGHT (3*gl_hbox)
#define MAX_ORDER 20
#define PI 3.141592654
#define XORIGIN 35
#define XMID 335
#define XEND 635
#define YORIGIN 380
#define YMID 210
#define YEND 40
#define LP 1
#define HP 2
#define BP 3
#define BS 4
#define ANALOG 1
#define DIGITAL 2

/* EXTERNAL VARIABLES */
extern int gl_apid;

/* GLOBAL VARIABLES */
int gl_hchar, gl wchar;
int gl_wbox, gl hbox; /* system sizes */
int phys_handle; /* physical workstation handle */
int handle; /* virtual workstation handle */
int wi_handle; /* window handle */
int top_window;                /* handle of topped window */
int xdesk,ydesk,hdesk,wdesk;
int xold,yold,hold,wold;
int xwork,ywork,hwork,wwork;  /* desktop and work areas */
int msgbuff[8];               /* event message buffer */
int keycode;                  /* keycode returned by */
int butdown;                  /* event-keyboard */
int ret;                      /* dummy return variable */
int hidden;                   /* current state of cursor */
int fulled;                   /* current state of window */
int contrl[12],intin[128];
int ptsin[128],intout[128];  /* storage used for bindings */
int work_in[11];             /* Input-GSX parm. array */
int work_out[57];             /* Output-GSX parm. array */
int pxyarray[10];            /* input point array */

long menu_addr;               /* FILTER menu address variable */
int openwindow;               /* T or F. If window is open */
int snum_order;               /* order of analog TF numerator */
int sden_order;               /* order of analog TF denominator */
int znum_order;               /* order of digital TF numerator */
int zden_order;               /* order of digital TF denom. */
double snum[MAX_ORDER+1];     /* analog TF numerator polynomial */
double sden[MAX_ORDER+1];     /* analog TF denom. polynomial */
double k;                     /* analog TF constant multiplier */
double znum[MAX_ORDER+1];     /* digital TF num. polynomial */
double zden[MAX_ORDER+1];     /* digital TF denom. polynomial */
double K_digital;             /* digital TF constant mult. */
double T;                     /* sample period T */
double Amax;                  /* maximum attenuation value */
double Amin;                  /* minimum attenuation value */
double w_pass;                /* passband radian frequency */
double w_stop;                /* stopband radian frequency */

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

main()
{
    int event, exitcall, i;
    exitcall=FALSE;
    openwindow=FALSE;
    clr_analog();
    clr_digital();

    appl_init();
    phys_handle=graf_handle( &gl_wchar, &gl_hchar, &gl_wbox,
&gl_hbox);
open_vwork();
graf_mouse(ARROW, 0x01);
rsrc_load("FILTER.RSC");
rsrc_gaddr(0, DPFDMENU, &menu_addr);
menu_bar(menu_addr, 1);
disable();

do {
  
  event = evnt_mesag(msgbuff);
  wind_update(TRUE);

  if (msgbuff[0] == MN_SELECTED) {
    
    if (msgbuff[4] == DESKINFO)
      form_alert(1, "[FILTER|Analog/Digital Filter

Design|By Alan D. George|M.S. Engineering Project|Fall
Semester, 1985][ OK ]");

    if (msgbuff[4] == LOADF) && (openwindow==FALSE))
      do_loadf();
    if (msgbuff[4] == SAVEF) && (openwindow==FALSE))
      do_savef();
    if (msgbuff[4] == QUIT)
      exitcall=TRUE;

    if (msgbuff[4] == BW) && (openwindow==FALSE))
      do_bw();
    if (msgbuff[4] == CHEBY) && (openwindow==FALSE))
      do_cheby();
    if (msgbuff[4] == BESSEL) && (openwindow==FALSE))
      do_bessel();
    if (msgbuff[4] == AUSER) && (openwindow==FALSE))
      do_auser();

    if (msgbuff[4] == AMAG) && (openwindow==FALSE))
      do_freqresp(ANALOG);
    if (msgbuff[4] == ADISPLAY) && (openwindow==FALSE))
      do_display(ANALOG);

    if (msgbuff[4] == BILINW) && (openwindow==FALSE))
      do_bilinear(BILINW);
    if (msgbuff[4] == BILINWO) && (openwindow==FALSE))
      do_bilinear(BILINWO);
    if (msgbuff[4] == DUSER) && (openwindow==FALSE))
      do_duser();

    if (msgbuff[4] == DMAG) && (openwindow==FALSE))
      do_freqresp(DIGITAL);
if ((msgbuff[4] == PULSE) && (openwindow==FALSE))
do_timeresp(PULSE);
if ((msgbuff[4] == DSTEP) && (openwindow==FALSE))
do_timeresp(DSTEP);
if ((msgbuff[4] == DDISPLAY) && (openwindow==FALSE))
do_display(DIGITAL);

menu_tnormal(menu_addr,msgbuff[3],1);
}
if (msgbuff[0] == WM_CLOSED)
close_window();

wind_update(FALSE);
} while (exitcall==FALSE);

menu_bar(menu_addr,0);
v_clsvwk(handle);
appl_exit();

/* TURN OFF THE MOUSE DISPLAY */
hide_Mouse() {
  if(!hidden) {
    graf_mouse(M_OFF,0x0L);
    hidden=TRUE;
  }
}

/* TURN ON THE MOUSE DISPLAY */
show_Mouse() {
  if(hidden) {
    graf_mouse(M_ON,0x0L);
    hidden=FALSE;
  }
}

/* OPEN A VIRTUAL WORKSTATION */
open_vwork() {
  int i;
  for(i=0;i<10;work_in[i++]=1);
  work_in[10]=2;
handle=phys_handle;
\_v\_opnvwk(work\_in,\&handle,work\_out);
}

/* OPEN A NEW WINDOW WITH THE NAME PASSED */
open\_window(wind\_name)
char wind\_name[];
{
wind\_get(0, WF\_WORKXYWH, \&xdesk, \&ydesk, \&wdesk, \&hdesk);
wi\_handle=wind\_create(WI\_KIND, xdesk, ydesk, wdesk, hdesk);
wind\_set(wi\_handle, WF\_NAME,wind\_name,0,0);
graf\_growbox( xdesk+wdesk/2, ydesk+hdesk/2, gl\_wbox,
\&gl_hbox, xdesk, ydesk, wdesk, hdesk);
wind\_open(wi\_handle,xdesk,ydesk,wdesk,hdesk);
wind\_get( wi\_handle, WF\_WORKXYWH, \&xwork, \&ywork, \&wwork, \&hwork);
openwindow=TRUE;
}

/* DISABLE MENU ITEMS CURRENTLY NOT YET IMPLEMENTED */
disable()
{
menu\_ienable(menu\_addr,ICHEBY,0);
menu\_ienable(menu\_addr,APHASE,0);
menu\_ienable(menu\_addr,IMPULSE,0);
menu\_ienable(menu\_addr,ASTEP,0);
menu\_ienable(menu\_addr,IMPINV,0);
menu\_ienable(menu\_addr,STEPINV,0);
menu\_ienable(menu\_addr,DPHASE,0);
}

do\_bw()
/* Find the Butterworth Filter Transfer Function, based */
/* on user defined specifications. */
{
int i,j,n,type;
double sigma, omega, sin(), cos(), a2, a1, a0;
double sqrt(), pow(), tmp[MAX\_ORDER+1], tmp1, tmp2, e;
clr\_analog();
hide\_mouse();
open\_window(" Butterworth Filter Design ");
clear\_window();
type=prompt_inputs();
printf("\nWhat is the desired order for the filter? ");
scanf("%d",&n);

snum_order=0;
sden_order=0;
snum[0]=1;
sden[0]=1;
k=1;
for ( j=1 ; j<=(int) n/2 ; j++ ) {
    /* Find Conjugate Pole Pairs */
    sigma=sin((2*j-1)*PI/(2*n));
    omega=cos((2*j-1)*PI/(2*n));
a2=1;
a1=2*sigma;
a0=sigma*sigma+omega*omega;
    for ( i=0 ; i<=sden_order ; i++ ) tmp[i]=sden[i];
    sden_order=quad_mult(tmp,sden_order,a2,a1,a0,sden);
}
if ( ((float) n/2) != ((int) n/2) ) {
    /* If n odd, add real pole */
    for ( i=0 ; i<=sden_order ; i++ ) tmp[i]=sden[i];
a2=0;
a1=1;
a0=1;
    sden_order=quad_mult(tmp,sden_order,a2,a1,a0,sden);
}
tmp1=10*e0;
tmp2=0.1*Amax;
tmp1=pow(tmp1,tmp2)-1;
e=sqrt(tmp1); /* e = sqrt(10^(.1*Amax)-1) */
tmp1=1/((double) n);
tmp2=pow(e,tmp1);
tmp1=tmp2/w_pass; /* Denormalize by replacing s with */
denormalize(tmp1); /* s * e^(1/n) / w_pass . */
if (type==HP) cnvt_hp();

close_window();
show_mouse();
}

do_cheby()

/* Find the Chebyshev Filter Transfer Function, based on */
/* user defined specifications. */
{
    int i,j,n,type;
    double sigma,omega,sin(),cos(),a2,a1,a0,tmp[MAX_ORDER+1];
}
double e, a, sqrt(), pow(), sinh_a, cosh_a, exp(), tmp1, tmp2;
double log(), sinh(), cosh(), asinh();
clr_analog();
hide_mouse();
open_window("Chebyshev Filter Design");
clear_window();
type=prompt_inputs();
printf("\nWhat is the desired order for the filter? ");
scanf("%d", &n);

snum_order=0;
sden_order=0;
snum[0]=1;
sden[0]=1;

tmpl1=10e0;
tmp2=0.1*Amax;
tmpl1=pow(tmpl1, tmp2) - 1;
e=sqrt(tmpl1); /* e = sqrt(10^(.1*Amax)-1) */
tmpl1=1/e;
tmp2=asinh(tmpl1);
a=tmp2/n; /* a = asinh(1/e)/n */
sinh_a=sinh(a);
cosh_a=cosh(a);

for ( j=1 ; j<=(int) n/2 ; j++ ) {
    /* Find Conjugate Pole Pairs */
    sigma=sin((2*j-1)*PI/(2*n)) * sinh_a;
    omega=cos((2*j-1)*PI/(2*n)) * cosh_a;
    a2=1;
    a1=2*sigma;
    a0=sigma*sigma+omega*omega;
    for ( i=0 ; i<sden_order ; i++ ) tmp[i]=sden[i];
    sden_order=quad_mult(tmp, sden_order, a2, a1, a0, sden);
}
if ( ((float) n/2) != ((int) n/2) ) {
    /* If n odd, add real pole */
    for ( i=0 ; i<sden_order ; i++ ) tmp[i]=sden[i];
    a2=0;
    a1=1;
    a0=sinh_a;
    sden_order=quad_mult(tmp, sden_order, a2, a1, a0, sden);
    /* if n is odd, then set K so that mag is 1 at d.c. */
    k=sden[0];
}
else {
    /* if n is even, then set K so that mag is 1/sqrt(1+e^2) */
    /* at d.c. */
    tmpl1=e*e+1;
    tmp2=sqrt(tmpl1);
do_bessel()

/* Find the Bessel Filter Transfer Function, based on user defined specifications. */
{
    int i, n, type;
    clr_analog();
    hide_mouse();
    open_window(" Bessel Filter Design ");
    clear_window();
    type=prompt_inputs();
    printf("\nWhat is the desired order for the filter? ");
    scanf("%d", &n);
    snum_order=0;
    sden_order=n;
    snum[0]=1;
    sden[n]=1;
    for ( i=1 ; i<=n ; i++ )
        sden[n-i]={(sden[n-i+1]*(n+i)*(n+1-i))/((2*i));
    k=sden[0];
    if (type==HP) cnvt_hp();
    close_window();
    show_mouse();
}

prompt_inputs()
{
    int type;
    type=form_alert(1,"[1][Please choose the|kind of Filter Desired][ _LP | HP ]");
    printf("\nWhat is the desired passband radian frequency ? ");
scanf("%f", &w_pass);
printf("\nWhat is the desired maximum attenuation there
[A_max, in dB] ? ");
scanf("%f", &A_max);
printf("\nWhat is the desired stopband radian frequency ? ");
scanf("%f", &w_stop);
printf("\nWhat is the desired minimum attenuation there
[A_min, in dB] ? ");
scanf("%f", &A_min);
return(type);
}

cnvt_hp() {
    /* Convert Low Pass to High Pass, which is simply */
    /* replacing s with 1/s. This can be done by changing */
    /* the numerator from s^0 to s^n, and reversing the order */
    /* of the denominator coefficients. */
    int i;
    double tmp[MAX_ORDER+1];
    snum_order=sden_order;
    snum[0]=0;
    snum[snum_order]=1;
    for ( i=0 ; i<=sden_order ; i++ ) tmp[i]=sden[i];
    for ( i=0 ; i<=sden_order ; i++ ) sden[i]=tmp[sden_order-i];
}

denormalize(constant)
double constant;
/* Routine to denormalize a normalized prototype filter */
/* contained in Snum[]/Sden[]. This is done by replacing */
/* s with alpha*s in the numerator and denominator of the */
/* transfer function. */
{ double tmp1;
/* s <- constant*s in numer. */
poly_replace(snum,snum_order,constant);
/* s <- constant*s in denom. */
poly_replace(sden,sden_order,constant);
tmpl=1/sden[sden_order];
k=k*tmpl; /* divide highest coeff. out of denom. */
poly_cmult(sden,sden_order,tmp1);
}

do_auser()
{
    int i;

    clr_analog();
    hide_mouse();
    open_window(" User Defined Analog Filter Transfer Function ");
    clear_window();
    printf("\nWhat is the Order of the Numerator? ");
    scanf("%d",&snum_order);
    printf("\nWhat is the Order of the Denominator? ");
    scanf("%d",&sden_order);

    clear_window(); /* ENTER THE NUMERATOR COEFFICIENTS */
    for ( i=0 ; i<=snum_order ; i++) {
        printf("\nEnter the coefficient for the Numerator s^%d term: ",i);
        scanf("%f",&snum[i]);
    }

    clear_window(); /* ENTER THE DENOMINATOR COEFFICIENTS */
    for ( i=0 ; i<=sden_order ; i++) {
        printf("\nEnter the coefficient for the Denominator s^%d term: ",i);
        scanf("%f",&sden[i]);
    }

    clear_window();
    printf("\nWhat is the constant K value? ");
    scanf("%f",&k);

    close_window();
    show_mouse();
}

clr_analog()
{
    int i;

    snum_order=0;
    sden_order=0;
    for ( i=0 ; i<=MAX_ORDER ; i++ ) {
        snum[i]=0; /* set numerator coefficients to zero */


```c

sden[i]=0; /* " denominator " " " */
}


clear_window()
{
    int temp[4];
    vs_curaddress(handle,3,0);
    vsf_interior(handle,2);
    vsf_style(handle,8);
    vsf_color(handle,0);
    temp[0]=xwork;
    temp[1]=ywork;
    temp[2]=xwork+wwork-1;
    temp[3]=ywork+hwork-1;
    v_bar(handle,temp);
}

close_window()
{
    wind_close(wi_handle);
    graf_shrinkbox( xwork+wwork/2, ywork+hwork/2, gl_wbox,
    gl_hbox, xwork, ywork, wwork, hwork);
    wind_delete(wi_handle);
    openwindow=FALSE;
}

draw_line(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
    int pxy[4];
    pxy[0]=x1;
    pxy[1]=y1;
    pxy[2]=x2;
    pxy[3]=y2;
    v_pline(handle,2,pxy);
}


do_bilinear(type)
int type;
```
/* Convert Analog Filter to Digital based upon the */
/* Bilinear Transform at the designated sample period T. */
/* See standard text for details. */
{
    int i,m,n,sample_type;
double wrk[MAX ORDER+1],wrk1[MAX ORDER+1],wrk2[MAX ORDER+1];
double wrk3[MAX ORDER+1],num[MAX ORDER+1],den[MAX ORDER+1];
double a2,a1,a0,tmp1,tmp2,wsample,wcutoff,tan();

clr digital();
sample_type=form_alert(1,"[l]Please choose the form
of sampling time that you wish to enter: | (T = sample period
in seconds) | (wsample = sample radian freq) |
hide_mouse();
if (type==BILINW)
    open_window(" Bilinear Transformation with match at
cutoff ");
else
    open_window(" Bilinear Transformation without match at
cutoff ");
clear_window();
if (sample_type==1) {
    printf("\nWhat is the sample period T desired? ");
    scanf("%f", &T);
}
else {
    printf("\nWhat is the sample radian frequency desired? ");
    scanf("%f", &wsample);
    wsample=wsample/(2*PI);
    T=1/wsample;
}
if (type==BILINW) {
    printf("\nWhat is the cutoff radian frequency desired for
matching? ");
    scanf("%f", &wcutoff);
}
m=snum order;
n=sden_order;
for ( i=0 ; i<=m ; i++ ) {
    num[i]=snum[i];
    wrk[i]=0;
}
for ( i=0 ; i<=n ; i++ ) den[i]=sden[i];
K_digital=k;
/* if bilinear w/o match, use factor of 2/T */
if (type==BILINWO)
    tmp1=2/T;
else {
    /* if bilinear w/ exact match, use factor of */
    /* wcutoff*cot(wcutoff*T/2) */
}
tmpl = wcutoff * T / 2;
tmpl2 = tan(tmp1);
tmpl = wcutoff / tmp2;

/* put factor in transfer function */
poly_replace(num, m, tmpl);
poly_replace(den, n, tmpl);

a2 = 0;
a1 = 1;
for (i = 0; i <= m; i++) {
    a0 = -1; /* z-1 term */
    quad_power(a2, a1, a0, m-i, wrk1); /* (z-1)^(m-i) */
    a0 = 1; /* z+1 term */
    quad_power(a2, a1, a0, i, wrk2); /* (z+1)^i */
    /* wrk3[] = wrk1[] * wrk2[] */
    poly_pmult(wrk1, m-i, wrk2, i, wrk3);
    /* wrk3[] = wrk3[] * num[m-i] */
    poly_cmult(wrk3, m, num[m-i]);
    /* wrk[] = wrk[] + wrk3[] */
    poly_padd(wrk, m, wrk3, m, wrk);
}
quad_power(a2, a1, a0, n, wrk1); /* (z+1)^n */
/* znum[] = wrk1[] * wrk[] */
znum_order = poly_pmult(wrk1, n, wrk, m, znum);
for (i = 0; i <= n; i++) wrk[i] = 0;

for (i = 0; i <= n; i++) {
    a0 = -1; /* z-1 term */
    quad_power(a2, a1, a0, n-i, wrk1); /* (z-1)^(n-i) */
    a0 = 1; /* z+1 term */
    quad_power(a2, a1, a0, i, wrk2); /* (z+1)^i */
    /* wrk3[] = wrk1[] * wrk2[] */
    poly_pmult(wrk1, n-i, wrk2, i, wrk3);
    /* wrk3[] = wrk3[] * den[n-i] */
    poly_cmult(wrk3, n, den[n-i]);
    /* wrk[] = wrk[] + wrk3[] */
    poly_padd(wrk, n, wrk3, n, wrk);
}
quad_power(a2, a1, a0, m, wrk1); /* (z+1)^m */
/* zden[] = wrk1[] * wrk[] */
zden_order = poly_pmult(wrk1, m, wrk, n, zden);
tmpl = 1 / zden[zden_order];
K_digital = K_digital * tmpl;
/* divide highest coeff. out of denominator */
poly_cmult(zden, zden_order, tmpl);

close_window();
show_mouse();
do_duser()
{
    int i;

    clr_digital();
    hide_mouse();
    open_window(" User Defined Digital Filter Transfer Function ");
    clear_window();
    printf("\nWhat is the Order of the Numerator? ");
    scanf("%d", &znum_order);
    printf("\nWhat is the Order of the Denominator? ");
    scanf("%d", &zden_order);

    clear_window(); /* ENTER THE NUMERATOR COEFFICIENTS */
    for (i=-1; i<=znum_order; i++) {
        printf("\nEnter the coefficient for the Numerator z\^%d term: ", i);
        scanf("%f", &znum[i]);
    }

    clear_window(); /* ENTER THE DENOMINATOR COEFFICIENTS */
    for (i=-1; i<=zden_order; i++) {
        printf("\nEnter the coefficient for the Denominator z\^%d term: ", i);
        scanf("%f", &zden[i]);
    }

    clear_window();
    printf("\nWhat is the constant K value? ");
    scanf("%f", &K_digital);

    close_window();
    show_mouse();
}

do_freqresp(mode)
int mode;

    /* Analog/Digital Frequency Response - Magnitude Plot */
    {
        double wbegint, wendt, w, wtmp, calc_amag(), calc_dmag();
        int xpos, ypos, last_xpos, last_ypos, type, i;
        char c_wend[20], c_wbegin[20], c_wtmp[20];

        type = form_alert(1, "[1][Please choose the kind of Magnitude Gain Plot Desired: ][ Voltage | Power ]");
        if (mode == ANALOG) {
if (type==1) open_window(" Analog Filter Magnitude Voltage Gain Plot \(|H(jw)|\) ");
else open_window(" Analog Filter Magnitude Power Gain Plot \(|H(jw)|^2\) ");
}
else {
if (type==1) open_window(" Digital Filter Magnitude Voltage Gain Plot \(|H(e^{j\omega T})|\) ");
else open_window(" Digital Filter Magnitude Power Gain Plot \(|H(e^{j\omega T})|^2\) ");
}
clear_window();
printf("\nWhat beginning value of \(w\) (omega) would you like [rad/sec]? ");
scanf("%f",&wbegin);
printf("\nWhat ending value of \(w\) (omega) would you like [rad/sec]? ");
scanf("%f",&wend);
clear_window();

/* Draw x axis and y axis, 2 lines wide */
draw_line(XORIGIN-1,YORIGIN,XORIGIN-1,YEND);
draw_line(XORIGIN,YORIGIN,XORIGIN,YEND);
draw_line(XORIGIN,YORIGIN+1,XEND,YORIGIN+1);
draw_line(XORIGIN,YORIGIN,XEND,YORIGIN);  
draw_line(XORIGIN-4,YEND,YORIGIN-4,YEND);
draw_end_point();
draw_line(XORIGIN-4,YEND+1,YORIGIN-4,YEND+1);
draw_end_point();
on y-axis *
  draw_line(XORIGIN-4,YMID,YORIGIN-4,YMID);
draw_mid_point();
draw_line(XORIGIN-4,YMID+1,YORIGIN-4,YMID+1);
don y-axis *
  draw_line(XEND,YORIGIN-4,YEND,YORIGIN-4);
draw_end_point();
don x-axis *
  draw_line(XEND-1,YORIGIN-4,YEND-1,YORIGIN-4);
draw_end_point();
don x-axis *
  draw_line(XMID,YORIGIN-4,XMID,YORIGIN-4);
draw_mid_point();
don x-axis *
  draw_line(XMID-1,YORIGIN-4,XMID-1,YORIGIN-4);

/* Draw 10 divisions on both the x and y axes */
for (i=XORIGIN; i<=XEND; i+=(XEND-XORIGIN)/10)
  draw_line(i,YORIGIN-2,i,YORIGIN+2);
for (i=YEND; i<=YORIGIN; i+=(YORIGIN-YEND)/10)
  draw_line(XORIGIN-2,i,XORIGIN+2,i);

v_gtext(handle,5,YEND+12,"1.0"); /* put on y-axis labels */
v_gtext(handle,5,YMID+5,"0.5");
v_gtext(handle,5,YORIGIN,\"0.0\")

wtmp=(wend+wbeargin)/2;
etoa(wbegin,c_wbegin,2);
etoa(wend ,c_wend ,2);
etoa(wtmp ,c_wtmp ,2);
v_gtext(handle,XORIGIN,YORIGIN+15,c_wbegin);
    /* put on
x-axis begin label */
v_gtext(handle,XMID-30,YORIGIN+15,c_wtmp ); /* put on
x-axis mid label */
v_gtext(handle,XEND-55,YORIGIN+15,c_wend ); /* put on
x-axis end label */

last_xpos=XORIGIN;
last_ypos=YORIGIN;
xpos=XORIGIN;
for ( w=wbegin ; w<=wend ; w+=(wend-wbegin)/(XEND-XORIGIN)
        ) {
    if ( mode==ANALOG) m=calc_amag(w);
else
    m=calc_dmag(w);
    if ( m>1 ) m=1;
    if ( type==2 ) m=m*m;  /* Make it Power Gain if Desired */
ypos=YORIGIN-(m*(YORIGIN-YEND));
draw_line(last_xpos,last_ypos,xpos,ypos);
last_xpos=xpos;
last_ypos=ypos;
xpos++;}
show_mouse();

double calc_amag(w) /* CALCULATE ANALOG MAGNITUDE */
double w; /* HELD IN SNUM[]/SDEN[]. */
{
    int toggle,i;
    double a,b,c,d,x,y,mag,pow(),sqrt();
    a=0;
toggle=1;
    for ( i=0 ; i<=snum_order ; i+=2 ) {
        a += snum[i] * pow(w, ((double) i) ) * toggle;
toggle *= -1;
    }
c=0;
toggle=1;
    for ( i=0 ; i<=sdem_order ; i+=2 ) {
        c += sden[i] * pow(w, ((double) i) ) * toggle;
toggle *= -1;
    }
b=0;
toggle=1;
for ( i=1 ; i<=snum_order ; i+=2 ) {
    b += snum[i] * pow(w, ((double) i)) * toggle;
    toggle *= -1;
}
d=0;
toggle=1;
for ( i=1 ; i<=sden_order ; i+=2 ) {
    d += sden[i] * pow(w, ((double) i)) * toggle;
    toggle *= -1;
}
x = k * (a*c+b*d)/(c*c+d*d);
y = k * (b*c-a*d)/(c*c+d*d);
mag = sqrt(x*x+y*y);
return(mag);

double calc_dmag(w) /* CALCULATE DIGITAL MAGNITUDE */
double w; /* HELD IN ZNUM[]/ZDEN[]. */
{
    int i;
double numreal,numimag,denreal,denimag,mag,sin(),cos(),sqrt();
doubletmpl1,tmp2,tmp3;
    numreal=0;
    numimag=0;
    for ( i=0 ; i<=znum_order ; i++ ) {
        tmpl1=i*w*T;
        tmp2=cos(tmpl1);
        tmp3=sin(tmpl1);
        numreal += znum[i]*tmp2;
        numimag += znum[i]*tmp3;
    }
    numreal *= K_digital;
    numimag *= K_digital;

    denreal=0;
    denimag=0;
    for ( i=0 ; i<=zden_order ; i++ ) {
        tmpl1=i*w*T;
        tmp2=cos(tmpl1);
        tmp3=sin(tmpl1);
        denreal += zden[i]*tmp2;
        denimag += zden[i]*tmp3;
    }
    tmpl1= numreal*numreal+numimag*numimag;
tmp2= denreal*denreal+denimag*denimag;
tmp3 = tmp1/tmp2;
mag = sqrt(tmp3);
return(mag);
}

do_timeresp(mode)
int mode;

/* Display Pulse/Step Time Response */
{
  int i, m, n, nT, xpos, ypos, last_xpos, last_ypos, ystart;
  double pl, p2, y[100], max_value;

  hide_mouse();
  if (mode==PULSE) {
    open_window(" Digital Pulse Response h(nT) ");
    ystart=YMID; /* for pulse response, put x-axis in middle of screen */
  }
  else {
    open_window(" Digital Step Response h1(nT) "); /* for step response, put x-axis at bottom of screen */
    ystart=YORIGIN;
  }
  clear_window();

  /* Draw x axis and y axis, 2 lines wide */
  draw_line(XORIGIN-1,YORIGIN,XORIGIN-1,YEND);
  draw_line(XORIGIN,YORIGIN,XORIGIN,YEND);
  draw_line(XORIGIN,ystart+1,XEND,ystart+1);
  draw_line(XORIGIN,ystart ,XEND,ystart );

  draw_line(XORIGIN-4 ,YEND ,XORIGIN+4 ,YEND ); /* draw end point */
  draw_line(XORIGIN-4 ,YEND+1 ,XORIGIN+4 ,YEND+1 ); /* on y-axis */
  draw_line(XORIGIN-4 ,YMID ,XORIGIN+4 ,YMID ); /* draw mid point */
  draw_line(XORIGIN-4 ,YMID+1 ,XORIGIN+4 ,YMID+1 ); /* on y-axis */
  draw_line(XORIGIN-4 ,YORIGIN ,XORIGIN+4 ,YORIGIN ); /* draw begin pt */
  draw_line(XORIGIN-4 ,YORIGIN+1 ,XORIGIN+4 ,YORIGIN+1 ); /* on y-axis */

  draw_line(XEND ,ystart-4 ,XEND ,ystart+4 ); /*
  draw_end_point */
  draw_line(XEND-1 ,ystart-4 ,XEND-1 ,ystart+4 ); /* on x-axis */
draw line(XMID,ystart-4,XMID,ystart+4); /*
draw mid point */
draw line(XMID-1,ystart-4,XMID-1,ystart+4); /*
on x-axis */

/* draw 10 divisions on both the x and y axes */
for ( i=XORIGIN; i<=XEND; i+= (XEND-XORIGIN)/10 )
draw line(i,ystart-2,i,ystart+2);
for ( i=YEND; i<=YORIGIN; i+=(YORIGIN-YEND)/10 )
draw line(XORIGIN-2,i,XORIGIN+2,i);

v_gtext(handle,XORIGIN,YORIGIN+15,"n=0"); /* put on
x-axis begin label */
v_gtext(handle,XMID-15,YORIGIN+15,"n=50"); /* put on
x-axis mid label */
v_gtext(handle,XEND-40,YORIGIN+15,"n=100"); /* put on
x-axis end label */

v_gtext(handle,5,ystart,"0.0"); /* put on y-axis label */

m=znum_order;
n=zden_order;
max_value=0;
for ( nT=0; nT<=99; nT++ ) {
    pl=0;
    for ( i=0; i<=m; i++ )
        if ( ((nT+i-n)==0 & mode==PULSE) || ((nT+i-n)>=0 &
             mode==DSTEP) )
            pl += znum[i];
    pl=(pl*K digital)/zden[n];
    p2=0;
    for ( i=0; i<=(n-1); i++ )
        if ( (nT+i-n)>=0 )
            p2 += zden[i]*y[nT+i-n];
    p2=p2/zden[n];
    y[nT]=pl-p2;
    if (y[nT]>max_value) max_value=y[nT];
}
xpos=XORIGIN;
for ( i=0; i<=99; i++ ) {
    if (mode==PULSE)
        ypos=YMID - ( (y[i]/max_value) * (YORIGIN-YEND) )/2;
    else
        ypos=YORIGIN-((y[i]/max_value) * (YORIGIN-YEND));
draw line(xpos,ystart,xpos,ypos);
xpos+=6;
}
show_mouse();
do display(mode)
int mode;

/* Display the ANALOG/DIGITAL transfer function in */
/* coefficient form. */
{
    int i;

    hide_mouse();
    if (mode==ANALOG) open_window(" Analog Filter Transfer Function Display ");
    else open_window(" Digital Filter Transfer Function Display ");
    clear_window();
    if (mode==ANALOG) {
        printf("\n=> Constant K value: %f",k);
        for ( i=0 ; i<=max(snum_order,sden_order) ; i++ ) {
            printf("\n%sd",i);
            printf(" term: Num=%f",snum[i]);
            printf(" , Den=%f",sden[i]);
        }
    } else {
        printf("\n=> Constant K value: %f",K_digital);
        for ( i=0 ; i<=max(znum_order,zden_order) ; i++ ) {
            printf("\nz%sd",i);
            printf(" term: Num=%f",znum[i]);
            printf(" , Den=%f",zden[i]);
        }
    }
    show_mouse();
}

clr_digital()
{
    int i;

    znum_order=0;
    zden_order=0;
    for ( i=0 ; i<=MAX_ORDER ; i++ ) {
        znum[i]=0; /* set numerator coefficients to zero */
        zden[i]=0; /* " denominator */
    }
}
/* MATH LIBRARY ROUTINES */
/* tan, sinh, cosh, tanh, asinh, acosh, atanh */

/* Tangent Function */
double tan(x)
double x;
{
  double y,sin(),cos(),tmpl,tmp2;

  tmpl=sin(x);
  tmp2=cos(x);
  y=tmpl/tmp2;
  return(y);
}

/* Hyperbolic Sine Function */
double sinh(x)
double x;
{
  double y,exp(),tmpl,tmp2;

  tmpl=exp(x);
  tmp2=exp(-x);
  y=(tmpl-tmp2)/2;
  return(y);
}

/* Hyperbolic Cosine Function */
double cosh(x)
double x;
{
  double y,exp(),tmpl,tmp2;

  tmpl=exp(x);
  tmp2=exp(-x);
  y=(tmpl+tmp2)/2;
  return(y);
}

/* Hyperbolic Tangent Function */
double tanh(x)
double x;
{
double y,sinh(),cosh(),tmp1,tmp2;

tmp1=sinh(x);
tmp2=cosh(x);
y=tmp1/tmp2;
return(y);
}

/* Inverse Hyperbolic Sine Function */
double asinh(x)
double x;
{
    double y,log(),sqrt(),tmp1,tmp2;
    tmp1=x*x+1;
tmp2=x+sqrt(tmp1);
y=log(tmp2);
return(y);
}

/* Inverse Hyperbolic Cosine Function */
double acosh(x)
double x;
{
    double y,log(),sqrt(),tmp1,tmp2;
    tmp1=x*x-1;
tmp2=x+sqrt(tmp1);
y=log(tmp2);
return(y);
}

/* Inverse Hyperbolic Tangent Function */
double atanh(x)
double x;
{
    double y,log(),tmp1,tmp2;
    tmp1=(1+x)/(1-x);
tmp2=log(tmp1);
y=tmp2/2;
return(y);
}
#define MAX_ORDER 20

quad mult(in,in_order,a2,a1,a0,out)
double in[],a2,a1,a0,out[];
int in_order;

/* Routine to multiply a quadratic by a polynomial. The form of the */
/* quadratic is a2*s^2+a1*s+a0, and the polynomial is in the array */
/* array in[], of size in_order, set up so that the index is */
/* of the term. For example, in[2] is the coefficient to go with s^2. */
{
  int i;
in[in_order+2]=0;
in[in_order+1]=0;

  for ( i=0 ; i<=(in_order+2) ; i++ ) {
    if (i>=2) out[i]=a2*in[i-2]+a1*in[i-1]+a0*in[i];
    if (i==1) out[i]=a1*in[i-1]+a0*in[i];
    if (i==0) out[i]=a0*in[i];
  }

  if (out[in_order+2]!=0)
    if (out[in-order+1]!=0)
      return(in_order);
  return(in_order+2);
}

poly_replace(a,a_order,alpha)
double a[],alpha;
int a_order;

/* Replace s with alpha*s (or z) in the polynomial */
/* a[0],a[1],... */
{
  int i;
double pow();

  for ( i=0 ; i<=a_order ; i++ ) a[i]*=pow(alpha,((double) i));
}
poly_cmult(poly,order,constant)
double poly[],constant;
int order;

/* Multiply the polynomial poly[0],poly[1],... by the constant. */
{
    int i;
    for ( i=0 ; i<=order ; i++ ) poly[i]*=constant;
}

poly_padd(a,a_order,b,b_order,out)
double a[],b[],out[];
int a_order,b_order;

/* Add 2 polynomials together... out[] = a[] + b[] */
{
    int i,tmp1;

    tmp1=max(a_order,b_order);
    for ( i=0 ; i<=tmp1 ; i++ ) {
        if ( i>a_order ) out[i]=b[i];
        else if ( i>b_order ) out[i]=a[i];
        else out[i]=a[i]+b[i];
    }
    return(tmp1);
}

poly_pmult(a,a_order,b,b_order,out)
double a[],b[],out[];
int a_order,b_order;

/* Multiply 2 polynomials together... out[] = a[] * b[] */
{
    int i,j,order;

    order=a_order+b_order;
    for ( i=0 ; i<=order ; i++ ) out[i]=0;
    for ( i=0 ; i<=a_order ; i++ )
        for ( j=0 ; j<=b_order ; j++ )
            out[i+j] += a[i]*b[j];
    return(order);
quad_power(a2,a1,a0,pwr,out)
 double a2,a1,a0,out[];
 int pwr;

 /* evaluate out[]=( a2*x^2 + a1*x + a0 )^pwr */
{  
 int i,j,order;
 double in[MAX_ORDER+1];

 out[0]=a0;
 out[1]=a1;
 out[2]=a2;
 if (a2 != 0)  order=2;
 else if (a1 != 0)  order=1;
   else  order=0;
 for ( i=2 ; i<=pwr ; i++ ) {  
   for ( j=0 ; j<=order ; j++ )  in[j]=out[j];
   order=quad_mult(in,order,a2,a1,a0,out);
 }
 if (pwr==0)  {  
   out[0]=1;
   order=0;
 }
 return(order);
}
LIST OF REFERENCES


