Dynamic Stability Instrumentation System (DSIS)
Volume II: Operation Software Manual

Taumi S. Daniels and Thomas L. Jordan

NASA Langley Research Center
Hampton, VA 23681

NASA TM-109041
April 1996

Table of Contents

Abstract

This document is a software manual describing the operational control of the Dynamic Stability Instrumentation System (DSIS). The system, developed at Langley Research Center in 1992, is used to determine aerodynamic stability coefficients from forced-oscillation wind tunnel tests. Two additional documents describe the DSIS hardware and use of the system. A single program performs all data acquisition, system control, and data display and storage. Source code for this program, written in the C programming language for a personal computer (PC) platform, is described at a functional level. Instructions are also provided to enable users to modify this software.

Manual Conventions

Different types of objects used within this document are listed below along with examples of their appearance. Program and command names appear in **boldface**, file names and arguments that you need to replace with a value appear in *italics*, and options and environment variables appear in UPPERCASE. Optional arguments to commands appear within [square brackets]. Although some of these entries to the computer appear in UPPERCASE, the user does not have to type them that way.

Meaning	Format	Example
Command argument	Italics	*XMA.C*
Computer output	Courier	`Stability`
Directory	Italics	*\SOURCE*
Drive	Capital	C:
Environment variable	Capitals	LIB
Filename	Italics	*XMA.C*
Function name	Bold	**dynstat()**
Program name	Bold	**DSP.EXE**
Program listings	Courier	`Stability`
Variable name	Italics	*keyswork*
Keyboard Key names	In Quotes	"Enter"

Introduction

This document, the second volume of a three-volume set, is intended as a software manual for operational control of the Dynamic Stability Instrumentation System (DSIS). This all-digital system, which utilizes dual-digital signal processors, replaces an obsolete analog version based on electro-mechanical resolvers. In addition to the Synchronous Demodulation algorithm that was implemented on the analog version, the new system also implements an algorithm based on the Fast Fourier Transform. Descriptions of these algorithms are provided elsewhere [1][2]. The first volume of the three-volume set describes the DSIS hardware, including electrical schematics. The third volume describes how to use the system to obtain the desired aerodynamic damping coefficients.

The main program, **DSP.EXE**, is created from C programming language source code files. Each of the source code files is described on a functional level and is listed in one of the appendices. Information is provided to enable the user to modify these programs. However, no attempt should be made to modify any part of the software without a good understanding of the C programming language, MS-DOS, and the DSIS hardware [3][4].

Table 1 lists the hardware that is controlled by the software. A text editor is needed to modify the source code files. A compiler is used to convert the source code files into object modules, and then a linker is used to combine all of the object modules into an executable program. Additional details are provided on utility software such as a text editor and a “make” program. Before running the software, some directory and file maintenance is required.

Software Description

A description of the **DSP.EXE** source code that outlines program flow from a functional basis is provided. C programming language functions are similar to FORTRAN or BASIC subroutine structures.

DSP.EXE

The main program for the DSIS is "**DSP.EXE**," and is created from six source code files, which are listed below. Source code files are listed in appendices A-G, except for appendix C which lists format information for configuration files. Each source code file contains a specific set of functions. On the PC, the most recent version of the program resides in the C:*DSP* subdirectory. More than one version of this program exists, as a result of the software development process.

DSP.EXE Source Code Files:

DSPHEAD.H	*XMA.C*
XAG.C	*XHP.C*
XRS.C	*XTZ.C*

Notice that the names of the six source code files for the **DSP.EXE** program are created to reveal functional contents. For example, the source code file *XAG.C* contains all functions whose names begin with the letter "A" through those beginning with the letter "G." Two exceptions are *XMA.C*, which contains the main program structure and *DSPHEAD.H*, which contains variable declarations and other compiler information.

Directory information is listed in table 2. This table gives locations of source code files, along with all other necessary software programs. List 1, “Function Listings by Source Code File for **DSP.EXE**,” provides more information.

Function Descriptions

Descriptions of individual functions are now given in the order that the functions appear in the source code files.

DSPHEAD.H

This source code header file is listed in appendix A. Include directives (indicated by "#include") enable the compiler and linker to find common functions. Macros (indicated by "#define") are defined to make programming tasks easier. All variable, function, and certain file names are declared as external and given a global visibility. Only a few of the functions use local variables. Only a few functions are declared external because these particular functions are referenced by a calling function prior to being defined.

XMA.C

This source code file is listed in appendix B, and contains the function **main(),** where program execution starts.

main()

Program flow is outlined in figure 1. In the figure, function names are listed in the order that they are executed. An

indent is used to indicate that the subsequent function(s) are called by the preceding function. Function names within braces are called conditionally depending upon software flags or user input.

The first action taken by the program is to restore the status of the system software based upon values saved during the previous run. Calibration constants from *CALIVAL.DAT*, wind tunnel parameters from *TESTVAL.DAT*, configuration settings from *DCONFIG.DAT*, sensitivities from *SENS.DAT*, and tare test results from *TAREVAL.DAT* are restored to their respective variables. These text files are listed in appendix C with format description information. The next step is to initialize the DAS-20 board. This multipurpose input and output (I/O) board is commercially available and is used to measure the model attitude signal voltage and to update and monitor the hardware switches that determine *datacode*.

The IEEE-488 board is initialized followed by an update of the gain and phase correction factors from the disk file *GPHSPLIN.MAT*. The 2001 DMM and PSC8000 signal conditioners are initialized, and program flow jumps to the function **setnew()**. Upon return, execution of the main program loop starts. Program flow is controlled by user input at this time. Depending on user input, one of four primary data acquisition and display functions will be called from this main loop. These four functions are **calibrate()**, **dynstat()**, **fftcontrol()**, and **plot()**. The first three will be referred to as “real-time” functions. The source code listings refer to the signal conditioners as “programmable amplifiers.”

ac_calibrate()

The user can depress the "C" key to execute this function while one of the real-time functions is executing. The hardware should be rewired, before this action is taken, to perform an AC calibration. Refer to the section entitled "AC Calibration" in Volume III: User Manual. This key provides optional measurement of a particular simulated balance signal via the 2001 DMM, with an alternative of user input from the keyboard. Either value is stored to a disk file along with all of the force and torque readings. These other readings are continuously being updated by one of the real-time functions, i.e., **dynstat()**.

XAG.C

This source code file, listed in appendix D, contains general functions with names beginning from "a" to "g".

aoa()

After clearing the personal computer monitor screen, this function generates a simple display with three values, continuously updated until the user depresses the "Enter" key. These values are computed model attitude, labelled "AoA;" model attitude voltage measured by the DAS-20 board; and the standard deviation of the model attitude voltage readings. The “F9" key, if depressed during real-time execution, will cause **keycommand()** to call this function. This function can be used to monitor and calibrate the model attitude measurements.

autozero_read()

A DC offset voltage is added by each of the PSC8000 programmable signal conditioners to the balance signal voltages prior to amplification. This function reads the offset from one of the programmable signal conditioners. See the description for the “A” key in list 2 for user input that determines the "active" signal conditioner.

caldatacode()

Three different types of calibration data are recorded and must be "marked" by a code in order to distinguish between them. This function prompts the user to select a number representing a calibration data code, which is stored in program variable *calcode*. The three types of calibration data are defined in table 3, which also lists information on another data marker, *datacode*. The “F5" key, if depressed during real-time execution, will cause **keycommand()** to call this function.

calibrate()

DC calibrations of the balance signals are performed using this function, the first of the four real-time functions. A real-time function is one that is executed on the PC while "live" data is being acquired and processed by the DSP microcomputers. This function updates results taken (alternately) from each of the two DSP microcomputers and displays

computed values onto the monitor. The number of data points stored is counted and regression results (computed elsewhere) are also displayed. The user can type the special keys while this function is executed.

dynstat()

Reduction of tare and wind-on data is completed by this second real-time function. The DSP microcomputers preprocess the displacement, torque, and secondary signals using a synchronous demodulation algorithm. This function uploads the DSP results alternately from each microcomputer and, displays real and imaginary components and phase angles, on the monitor. Frequency information is determined by hardware tachometer within the PC. Model attitude information is updated. Standard deviations of all of the displayed values are also updated. The number of averages and the display of those values are determined by the user. Averages are always being computed, regardless of whether they are actually displayed on the monitor. The user can use the special keys while this function is executing.

fftcontrol()

This third real-time function performs the same tasks as the **dynstat()** function, with a few differences. The DSP microcomputers preprocess the displacement, torque, and secondary signals using a Fast Fourier Transform algorithm. Peak values that vary with frequency (for both real and imaginary terms) from the DSP microcomputers are sent to the PC where they are corrected for FFT window gain error, then averaged to determine standard deviations. Frequency information is determined by the hardware tachometer. Phase angles are computed from the averaged values. Averaging is handled the same way as in **dynstat()**. The user can use the special keys while this function is executed.

filestore()

A main menu is created by function **setnew()**, and selection of option 9, “Change filename for Dynamic Stability Data Storage,” causes execution of this function. The user input is prompted for the name of a disk file. Two file names are created, one for calibration data and one for formatted tare or wind-on data. The first file name has the letter "C" appended to the beginning of the user input name (for calibration data), while the second name has the letter "F" appended to the beginning (for formatted data). This function can also be accessed by depressing the “F8" key during real-time execution.

findaverage()

In the source code listing in appendix D, notice that this function has two calling parameters. The first is the pointer or variable name of the most recent data value to be averaged, while the second is a pointer or array name of past values of that data. The function updates the array and computes a running average called *finalave*. It then computes a standard deviation on the averaged values called *deviation*.

gaininterp()

This function corrects AC gain variation over frequency, an error associated with the analog to digital (A/D) converters on the DSP microcomputer boards. Calibration data is loaded from GPHSPLIN.MAT, a text file that resides in the same subdirectory as **DSP.EXE**.

getdatacode()

This function combines the status of two binary input port bits on the DAS-20 board with two program variables to determine a special code value. This variable *datacode* is used to mark certain types of data and can take on values 1, 2, 3, 6, 7, and 9. A second data marker, calcode, is also used but not affected by this function. Once determined, the *datacode* value is sent to an LED display on the front panel of the Motor Speed Control Chassis. Refer to table 3 for more information on *datacode*.

getindepvar()

During calibrations of the balance signals, either attitude is adjusted or weights are applied to the strain gage balance. The user's known reference is the angle or the weight, both measured independently. This function prompts the user for that independent data point. The adjustment to attitude is made by elevating the entire sting while the locking pin maintains the oscillating balance at a fixed position. Weights are applied using weight pans and special calibration

hardware while the sting is fixed at 0 (zero) degrees attitude.

gnormalize()

If selected by the user, a correction of A/D drift error is performed by this function. The function corrects AC gains by comparing values measured by the 2001 Digital Multimeter with A/D values using the HP 3245 Universal Source (function generator) as a voltage standard. The “F4" key, if depressed during real-time execution, will cause this function to be called from **keycommand()**. Daily use of this feature is recommended at the beginning of that day's testing.

XHP.C

This source code file is listed in appendix E.

help()

A list of all the keys with descriptions is displayed on the monitor by this function when the user depresses the “F1" key. Refer to list 2, “Key Function Assignments,” for more information on the “F1" key.

keycommand()

This function controls user input to the main program infinite loop. The functions **calibrate()**, **dynstat()**, and **fftcontrol()** are all programmed with “while” loop stuctures with extra code that enables certain keys to be sensitive. If the user depresses any key while one of these functions is executing, program flow jumps to this function, where the keyboard input is determined. Typically, the user will depress one of the keys, and **keycommand()** will perform a particular activity as listed in list 2, which uses the same indentation format as figure 1. If the depressed key does not match one of the keys, then program flow returns to the calling function. The variable *keyswork* is also used to determine program flow information. See table 4 for definitions of *keyswork* values.

leastsq()

This function calculates a linear least squares regression on input data. The three array names *ndata*, *tempind*, and *tempdep*, are the number of data points, independent data array pointer, and dependent data array pointer, respectively. They are the three input parameters used for this function. Regression slope *lscoffa* and y-intercept *lscoffb* are global output variables. A third output, *lsstdev*, is the standard error of the estimates. The code for this function is derived from a similar program given in reference 5.

newx() and **newy()**

These two functions separately scale the input x and y coordinates to fit data to plots on the monitor screen. Floating point values from spectral data (frequency and magnitude) passed up from the DSP microcomputers are converted to integer pixel coordinates.

output()

This function updates real-time data from either **calibrate()**, **dynstat()**, or **fftcontrol()** onto the monitor. In addition, it updates the status of hardware switches that define the data marker datacode. If datacode = 9, then **output()** computes the sensitivities. If datacode = 3, then tare values are computed. If datacode = 7, then **output()** computes a set of wind-on coefficients (three different sets). In each case, the **output()** will then display the computed results across the middle of the monitor screen. This space is left blank on the displays generated by each of the three functions **calibrate()**, **dynstat()**, and **fftcontrol()**. If datacode is any other value than one of the three mentioned above, then this space is cleared or blanked out. Screen colors are yellow for text and white for data.

parameters()

Data reduction parameters and other constants are computed by this function. These parameters are listed in table 5. The last four items in the table are used to compute aerodynamic coefficients.

plot()

This is the fourth real-time data display function. It takes FFT values from the DSP microcomputers and computes magnitude spectra for a user selected channel (displacement, torque, or secondary). Computed results are scaled and plotted onto the monitor. The update rate for plots is limited by the long input buffer lengths required for the lowpass digital filters prior to the FFT computation on the DSP microcomputers.

ppcpk()

Two variables, *ppc* and *pk*, are computed by this function but are not used by any other function nor saved to any disk file.

printfile()

A user selected disk file is sent to the printer by this function. This function is intended to be used as a utility and to prevent the user from terminating the software merely to print a data file. Tare and wind-on results are sent to the printer as each data point is taken. The printer buffer prints a new page when it becomes full. Thus, this function is not needed to obtain hardcopy during or immediately after tare or wind-on tests. The “F12" key, if depressed during real-time execution, will cause **keycommand()** to call this function.

XRS.C

This source code file is listed in appendix F.

readaoa()

Model attitude is measured by an inertial sensor whose output is conditioned by electronics, and then sent to the main interface box. The conditioned signal is then digitized by an A/D converter on the DAS-20 multifunction board. This function reads the DAS-20 board and calculates the model attitude using constants entered by the user. Wind-on conditions are sensed and model attitude is corrected for flow angle error. Also, inverted model information from the user is utilized to correct the reported model attitude.

read_gain()

This function reads the gain setting of one of the signal conditioners. The three signal conditioners, one of which is "active" based on user input, are connected to the PC via the IEEE-488 bus.

realimag()

This function is used by **calibrate()** and **fftcontrol()** to compute real and imaginary components of data from the DSP microcomputers. The torque and secondary channels are resolved with respect to the displacement channel (which is defined to have 0° phase angle). Standard deviations of the real and imaginary results are also computed.

refresh()

This function prints text labels for the numeric data that the **output()** function displays on the monitor.

scale()

This function draws the axes and prints the labels for the FFT plots generated by the **plot()** function.

setnew()

After performing hardware and software initialization, the **DSP.EXE** program calls this function to display 14 user-selected options from the main menu, defined in table 6. The selected option is executed. The function **setnew()** uses variable *p* to determine the DSP microcomputer algorithm currently being executed. See table 7 for definitions of variable *p* values. This function is also called when the user depresses the "ESC" key while **calibrate()**, **dynstat()**, or **fftcontrol()** are executing. This main menu is the first and last display seen by the user. The function listing in appendix F contains additional information in the comments (/* */) fields.

setsen()

Sensitivities are values used in balance signal data reduction. These values may be computed from actual data (datacode = 9) or entered by hand from the keyboard. Refer to "Sensitivities Test Procedure" in Volume III: User Manual for a description of how these values are computed. When the user depresses the "F6" key, **keycommand()** calls this function which prompts the user to enter the sensitivities.

setup_plot()

This function prompts the user for calibration constants and XY plotter scaling parameters. It is called from **setnew()** when the user selects main menu option 10, "Plot Tare or Wind-on axes on X-Y plotter." Also, based on user input, this function commands the XY plotter to draw and label axes for tare or wind-on plots. The calibration constants that are requested are listed in table 8. Notice the first column is the name of the requested term, while the last three columns are descriptions of these terms for the three types of testing. These terms are computed from DC balance calibration data.

startover()

When the user depresses the "ESC" key, this function resets certain variables to zero and calls the **setnew()** function to display the main menu. Refer to table 9, "Variables Affected by Function **Startover()**," for more information.

switchboard()

The PC includes two identical DSP microcomputer boards, each equipped with two A/D channels. Board 1 samples data from the displacement and torque channels; Board 2 samples the same displacement channel and the secondary channel. All of the real-time functions **calibrate()**, **dynstat()**, and **fftcontrol()** access and retrieve data from each DSP board alternately using this function.

XTZ.C

This source code file is listed in appendix G.

takecaldat()

This function is used for diagnostic purposes and is not utilized by the software. It is normally called by the function **takedata()** to perform hardware calibrations using the 2001 DMM as a reference. Notice, however, that the call to this function has been commented out of **takedata()**.

tareplot()

This function plots a tare data point on each of the two plots on the XY plotter. The data points are computed and scaled by this function based on user input from main menu option 10, "Plot Tare or Wind-on axes on X-Y plotter."

takedata()

When the "F10" key is depressed , **keycommand()** calls this function. The current *calcode* and *datacode* values are used by **takedata()** to determine how to process the current data point. The function computes coefficients, stores the current data point to disk, and plots the data point on the XY plotter. Refer to table 3 for a description of **takedata()** actions for particular *calcode* and *datacode* values.

The **takecaldat()** and **takedata()** functions use the variable *typecal* to determine which channel is being calibrated. See table 10 for definitions of this variable.

taredata()

After the user has conducted a tare test, the data are ready for manipulation as provided by this function. The user depresses the "F7" key to call this function and then responds to the submenu options "amend," "omit," "compute regression," or "load alternate data set." Before a regression is computed, the user should amend or omit any data points as required for the previously completed tare test. The option "compute regression" is selected next. The computed regression line will be drawn on the tare plot. These steps are repeated after loading the other tare data set.

testcond()

This function corresponds to main menu option 8, “Record test numbers and conditions.” It prompts the user to enter values for the test condtions or parameters listed below. The function calls **parameters()** to compute other wind tunnel parameters based on user input. Refer to table 11, “Variables for Main Menu Option 8,” and Appendix C, “Formats for Configuration Files,” for more information.

windonplot()

When datacode = 7, **takedata()** calls this function to command the XY plotter to plot a user-selected coefficient versus model attitude. Two variables used to configure this function, *woplotcode* and *wosymbolc*, are defined in Table 12. These variables can be changed by the user from main menu option 10, “Plot Tare or Wind-on axes on XY plotter.”

Source Code Modification

Editing the Source Code

The **DSP.EXE** program is created by compiling and linking C programming language source code files. The format of these source code files is ASCII text. A text editor is needed to modify the files prior to compiling and linking. The user should modify these programs as required by using a text editor such as **QEDIT.EXE**, which resides on the DSIS PC. Refer to table 2 for the location of this editor on the PC. The choice of text editor is left to the user.

For example, to modify the *XMA.C* source code file, locate the current version of this file. Change the subdirectory to C:*DSP* using the DOS “cd” command.

```
cd dsp
```

Now execute the text editor. At the DOS prompt type:

```
qedit xma.c
```

The monitor will display the top of the *XMA.C* text file.

Note that directory and file information is provided across the top of the monitor. Either the mouse or arrow keys can be used to position the cursor to the desired location. Use comment fields (/* */) to note changes to the source code.

Once editing has been completed, exit the text editor by depressing the “F10" key to save the file, the “F7" key to terminate this editing session, and the “ESC” key to quit the editor program.

Compile and Link

After the source code has been modified by the user, it should be compiled and linked. The following commands are specific to Microsoft C Version 5.2, although newer versions or other C compilers can be used.

A special utility called **MAKE.EXE** should be used to perform these tasks[5]. This utility is provided on the DSIS PC and combines both tasks into one step.

The first step is compilation using the **CL.EXE** program. Source code files, denoted with a ".C" extension, are compiled into object modules. The information in the header file is used by the compiler at this time.

The linker program **LINK.EXE** then combines all of the object modules, denoted by a ".OBJ" extension, into an executable program. External libraries are used at this time to include any functions not located in the object files. Thus, when successfully completed, the **MAKE** utility creates an executable file, denoted by a ".EXE" extension, such as **DSP.EXE**.

The **MAKE** utility uses a separate text file for information on which files to compile, to link, and which libraries to use. An example of this text file is listed in appendix H, and is called *haste*.

Thus, at the DOS prompt, type:

```
make haste
```

The entire process will take several minutes to complete. Each step in the process will be evident from the information printed on the monitor. The DOS prompt will reappear when completed.

An explanation of the *haste* file is now given. Notice that the format of the first three lines is repeated for all source code files and the final executable file. The **make** utility compares the time/date stamp on the target file and source file in order to determine if the user has made any changes to the source code. If so, the command on the next line is executed; otherwise it is skipped. Comment lines begin with a "#" sign and are ignored by the **make** utility.

The **make** utility identifies the object file name listed on the first line as a target for compilation of the matching source code file name listed on the same line. The next line starts with the **Cl.EXE** command for the make utility to execute to create the object file. The "/AL" switch tells the compiler to use the large memory model, and the "/c" switch tells the compiler to compile only (not link). If any errors occur during compilation, an error file will be created by the redirection symbol ">" instead of appearing on the monitor. In this instance the file will be *comp$ma.err*. Note that error file names match source and object file names.

The last three lines create the **DSP.EXE** file. The target is identified first, as before, as **DSP.EXE**, followed by the object files that are checked for time differences. No comment is given. The command **LINK.EXE** is used with several fields separated by commas. The first field contains all of the object files (minus extension) in the first field separated by "+" signs. The next field is the name of the newly created executable file. The third field is blank, while the fourth field contains all of the external libraries required. As before, a redirection symbol is used to create a file. All **LINK.EXE** errors will be placed into a disk file named *link$az.err*.

Software Preparation

Before running any of the three software programs, some directory and file maintenance is required. The **DSP.EXE** program requires that several other specific files reside within the same subdirectory. These files are listed in list 3, "Necessary Files for Run-Time Execution." Any older versions of the source code should be saved in different subdirectories. Refer to figure 2, "PC Directory Tree," for information on directory structure, and table 2, "PC Directory Information" for locations of specific files. After completing a calibration, tare data run, or wind-on run, the data files should be moved or copied to another directory or floppy disk.

References

1. Kilgore, Robert A.: *Wind-Tunnel Measurements of the Aerodynamic Damping and Oscillatory Stability in Pitch of a Sphere at Mach Numbers From 0.02 to 4.63*, NASA-TM-X-50348, August 1963.

2. Tripp, John. S.; and Tcheng, Ping: *Dynamic Stability Test Instrumentation Using Digital Signal Processing*

Techniques. 18th Aerospace Ground Testing Conference, AIAA-94-2583, Colorado Springs, Colorado., June 20-23, 1994.

3. *Microsoft C for the MS-DOS Operating System Run-Time Library Reference*, Part Number 048-014-104. Microsoft Corp., 1987.

4. *User Guide for the DAS-20 A/D & D/A Acquisition Boards*, Revision C. Keithley Metrabyte Corp., Feb. 1991.

5. Press, William H.; Flannery, Brian P.; Teukolsky, Saul A.; and Vetterling, William T.: *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, Cambridge, Massachusetts, 1988, pp. 523-528.

6. *CodeView and Utilities Software Development Tools for the MS-DOS Operating System*, Part Number 016-014-042. Microsoft Corp., 1987.

Appendix A

DSPHEAD.H

Header File Listing:

```
/* dsphead.h
   Header file for compiling the dspx.c files. */

#include <stdio.h>
#include <dsputil.h>        /* Header for commercial DSP board software */
#include <float.h>
#include <math.h>
#include <graph.h>
#include <conio.h>
#include <time.h>
#include <ieeeio.h>
#include <ctype.h>
#include <fcntl.h>
#include <io.h>
#include <bios.h>

/* defined macros... */
/* AT = places text at specified position on screen */
/* DATA = set color for output of numbers (bright white) */
/* GO = outputs string to screen */
/* HIDE = set color for clearing output field (black) */
/* OUT prints the information following it to a buffer string */
/* TEXT = set color for output of text (bright yellow) */

#define      AT       _settextposition
#define      CLS      _clearscreen(_GCLEARSCREEN);
#define      DATA     _settextcolor(16);
#define      BLUEBAK   _setbkcolor(_BLUE);
#define      BLACKBAK   _setbkcolor(_BLACK);
#define      CYANTXT   _settextcolor(11);
#define      ENDY     440           /* y-pixel boundary of graph */
#define      FL       fflush(stdin); /* clears data from input stream */
#define      GO       _outtext(buffer);
#define      HIDE     _settextcolor(0);
#define      HIGHY     130           /* Scaling factor for y-axis */
#define      MAXPNTS   500        /* max number FFT points to plot */
#define      OUT      sprintf(buffer,
#define      OUT_XY    fprintf(stdaux,            /* x-y plotter */
#define      PI       3.14159265
#define      SCALE     54.18539922   /* 20*log10(1024*.5) */
#define      SD       fprintf(formated,     /* Store formated data */
#define      SDCAL     fprintf(calfile,    /* Store calibration data */
#define      STORE     fprintf(filename,     /* Store numerical data */
#define      STORE_AC   fprintf(ac_cal,
#define      TEXT     _settextcolor(14);
#define      VALUES    12         /* # values upload from FFT data */
#define      WINDON    datacode==6 || datacode==7
#define      TARE     datacode==2 || datacode==3
#define      SENSITIV   datacode==1 || datacode==9
#define      XORG     90         /* x-pixel value of graph origin */
#define      YORG     80         /* y-pixel value of graph origin */
#define      ESC      27         /* The following are key codes...*/
#define      Space     32            /* ...for the specified key */
#define      Enter     13
#define      F1       59
#define      F2       60
#define      F3       61
#define      F4       62
#define      F5       63
#define      F6       64
#define      F7       65
#define      F8       66
#define      F9       67
```

```
#define      F10      68
#define      F11      133
#define      F12      134

                      /* global variables in alphabetical order */
extern char accalfile[12];
extern char *acfile;             /* filename for ac cal Data Storage */
extern float acnorm;                        /* Keithley reading */
extern long addr[2];            /* DSP address for data from dynstat */
extern long addr_plot[3];                /* DSP addresses fft data */
extern long addr_realf[2];              /* DSP addresses of "realf" */
extern float alphabias;                          /* Alpha bias */
extern float alphadelt;                         /* Delta Alpha */
extern float alphaoff;                         /* Alpha offset */
extern float alphasens;     /* AoA accelerometer const. Alpha sensitivity */
extern int  ampchan;                /* Amplifier channel indicator */
extern int  amplif;                    /* Amplifier on-off switch*/
/* extern float ampratio;    */             /* Ratio of peak amplitudes */
/* extern float ampratiosave; */             /* Averaged amplitude ratio */
extern float amr;           /* Secondary channel corollary to kspring */
extern float angleav[2][121];       /* averaging arrays for phase angles */
extern float anorm0[2];               /* Gain normalization factors */
extern float anorm1[2];
extern float tanorm0[2];
extern float tanorm1[2];
extern float afreq;
extern float AoAval;                          /* Angle of Attack */
extern float aoavolts;       /* Raw voltage reading from angle of attack */
extern float array[VALUES];          /* Buffer array, for dsp_up_array */
extern float atten[6];       /* brings voltage to scale of highest gain */
/* extern float avamprat[2][121]; */       /* averaging arrays for ampratio */
extern float avaoavolts[121];          /* averaging array for aoavolts */
extern int  aveorimed;         /* Control: averaged or immediate data */
extern int  NAVE;             /* # of data values in running average */
extern float aveaveang[2][121];     /* Arrays for storing averages of... */
/* extern float aveaveamp[2][121]; */     /* ...averages for finding st. dev. */
extern float aveavepeak0[2][121];
extern float aveavepeak2[2][121];
extern int  avecount;       /* Counter for preloading averaging arrays */
extern float avimag[2][121];      /* averaging array for imag components */
extern float avpeak[2][8][121];       /* averaging arrays for all peaks */
extern float avreal[2][121];      /* averaging array for real components */
extern int  b;           /* Differentiates between boards/their data */
extern long baddr[2];                /* I/O addresses of DSP boards */
extern int  beentoaoa;     /* Switch tells program it has just read AoA */
extern float bmr;           /* Secondary channel corollary to inertia */
extern char buffer[80];                     /* Output text string */
extern int  c;            /* Differentiates between channels to plot */
extern float c1,c2,c3,c4;   /* values to compute aerodynamic coefficients */
extern float calib[3];                   /* calibration constants */
extern int  calibnum;                     /* Calibration number */
extern float calindep[121]; /* Data entered by hand, used in leastsq funct. */
extern float calarray0[121];      /* storage for calibrate function data */
extern float calarray1[121];      /* storage for calibrate function data */
extern int  calcode;     /* Display code for specifying type of cal data */
extern int  calcode_old;  /* used to restore calcode after shunt resistor */
extern float calcoffa0;           /* display and storage ls 'a' value */
extern float calcoffa1;           /* display and storage ls 'b' value */
extern float calcoffb0;           /* display and storage ls 'a' value */
extern float calcoffb1;           /* display and storage ls 'b' value */
extern char cal_name[20];       /* storage for name of calibration type */
extern int  calpoint;                    /* auto timed cal counter */
extern int  calpoint_d;            /* Calibration data point counter */
extern int  calpoint_s;           /* Calibration shunt point counter */
extern int  calpoint_z;            /* Calibration zero point counter */
extern int  calres;                 /* Calibration resistor switch */
extern float ctare;                            /* Tare damping */
extern float cf[2];                     /* Gain correction values */
extern float cd[2];
extern float cforce[2];               /* Corrected force and moment */
```

```
extern float cdisp[2];                    /* Corrected displacement */
extern long dacvalue;                /* Returned hex value of offset */
extern int  data[3];                  /* Data i/o from DAS-20 board */
extern int  datacode;                /* Data code from DAS-20 board */
/* extern float devampratio;  */     /* standard deviation of amplitude ratio */
extern float devaoa;                /* Standard deviation of aoavolts */
extern float devaveang;                /* Deviations of the averages */
/* extern float devaveamp; */
extern float devavepeak0,devavepeak2;
extern float dvpk[2][2];  /* Storage for devs of averages for noise record */
extern float deviation;           /* Stores st. dev. from findaverage */
extern float devimag[2];              /* St. Dev. of imag components */
extern float devpeak[2][2];       /* standard deviation of fundamentals */
extern float devreal[2];              /* St. Dev. of real components */
extern float dfreq;                       /* used in gnormalize() */
extern float discal0;       /* Calibration reading, displacement Brd. 0 */
extern float discal0_z[15];       /* 1st val is len, displacement0 zeros */
extern float discal0_s[15];      /* 1st val is len, displacement0 shunts */
extern float discal1;       /* Calibration reading, displacement Brd. 1 */
extern float discal1_z[15];         /* storage for displacement1 zeros */
extern float discal1_s[15];         /* storage for displacement1 shunts */
extern float displm[2];
extern float div[2][2][11];   /* Scaling divisions for all cases of plots */
extern int  e;           /* Differentiates between scaling on x-axis */
extern int  each;                  /* How many FFT points to plot */
extern int  endx;               /* pixel coordinate of last x value */
extern int  error;           /* Error code returned from DAS-20 board */
extern float fft[MAXPNTS*2];         /* Upload array for fft plot data */
extern float flow_angle;           /* correct AoA for UPWT flow angle */
extern float forcem[2];
extern float freqg[101];     /*Frequency column of gain correction table */
extern char fname[13];
extern char *file;                   /* filename for Data Storage */
extern char *format;           /* filename for formatted data storage */
extern char *cformat;               /* filename for cal data storage */
extern float finalave;        /* Stores averaged value from findaverage */
extern float freq;                       /* frequencies of peaks */
extern int  functionkey;      /* User controlled directions for program */
extern float gain[6];             /* Gain of programmable amplifiers */
extern float gain0[2][101];               /* A/D gain table Chan 1 */
extern float gain1[2][101];                    /* Chan 2 */
extern float g0[2];
extern float g1[2];
extern char gaincode;             /* Code letter for amplifier gain */
extern int  highx;                   /* Scaling factor for x-axis */
extern int  i;               /* general use loop counting variable */
extern float imag[2];          /* Imaginary w/ respect to Displacement */
extern float imagsave[2];      /* save averaged values of imag component */
extern int  imax;               /* size of gain interpolation table */
extern float indepvar;       /* variable used to hold hand entered data */
extern float inertia;                              /* Inertia */
extern int  inverted;              /* Model inverted switch for AoA */
extern int  keith;                          /* Keithley flag */
extern int  key;                   /* Key-in for gain correction */
extern int  keyswork;     /* Turns off functionkey sensitivity to plots */
extern float kspring;                        /* Spring constant */
extern int  labelwoplot;    /* Switch, controls labeling of windon plots */
extern int  lastcalpt;      /* Switch, last calibration point was taken */
extern float lscoffa;               /* leastsq() linear coefficient */
extern float lscoffb;              /* leastsq() constant coefficient */
extern float lsstdev;               /* leastsq() standard deviation */
extern float m1;             /* mr/d or other specified coefficients */
extern float m2;             /* mj/wd or "                    */
extern float mach;                     /* Wind tunnel mach number */
extern int  maxx, maxy;       /* Number of pixels in x and y directions */
extern float mjwdtare;                        /* Damping factor */
extern int  mode;                         /* DAS-20 board mode */
extern float momentarm;             /* user entered moment arm length */
extern float offset[5];            /* Amplifier offset from auto-zero */
extern int  offrange;           /* value indicating full scale range */
```

```
extern float omega;                    /* frequency in radians/sec */
extern int  p;                /* Differentiates between DSP programs */
extern float peak[8];          /* magnitudes of real and imaginary data */
extern float peaksave[8];   /* save average values for standard deviation */
extern float phase[2][101];                 /* Phase calibrate table */
extern float ph[2];                          /* Phase correction */
extern float phdif;             /* Phase angle difference in degrees */
extern float phdifm[2];                  /* Uncorrected phase shift */
extern float phdifsave;             /* Average phase angle in degrees */
extern float phifft[2],phideg[2];     /* Phase angles for peak amplitudes */
extern float pk;                             /* (f*cos(eta)/d */
extern float pnt[MAXPNTS];            /* Array for dB levels for plot */
extern float ppc;                        /* (f*sin(eta)/freq*d) */
extern float pressure;                    /* Wind tunnel pressure */
extern char *program;       /* board0 initialize to longest program name */
extern char *program2;                 /* program for 2nd DSP board */
extern int  ptnum;                           /* Point number */
extern float Q;                     /* Dynamic pressure, lb/ft^2 */
extern float reading[6];                /* scaled voltage readings */
extern float real[2];             /* Real w/ respect to Displacement */
extern float realsave[2];      /* save averaged values of real component */
extern float refarea;                    /* Reference wing fsarea */
extern float reflength;                      /* Reference length */
extern float reynolds;                 /* Reynolds number per foot */
extern int  runnum;                             /* Run number */
extern float seccal;               /* Calibration reading, secondary */
extern float seccal_z[15];     /* 1st val is len, secondary zeros storage */
extern float seccal_s[15];    /* 1st val is len, secondary shunts storage */
extern float sen[6];                   /* calculated sensitivities */
extern int  shuntopt;                   /* Shunt resistor option */
extern float sqrt2;
extern float statemp;                    /* Static temperature °R */
extern float standev0;                /* display and storage ls 'std' */
extern float standev1;                /* display and storage ls 'std' */
extern float stdev;         /* St deviation of phase angle differences */
extern float t1;              /* tr/d or other specified coefficient */
extern float t2;              /* tj/wd or "                  */
extern int  tarecount;                 /* Tare Data Point counter */
extern float tarelinreg[2][15]; /*Tare linear regressi. variable 0=up,1=low */
extern float tarex[2][15];     /* Tare plot upper abscissa, omega squared */
extern float tarey[2][15];           /* Tare plot upper ordinate, mr/d */
extern float temperature;                /* Wind tunnel temperature */
extern int  testnum;                            /* Test number */
extern int  textpc[11];                /* col positions of scale # */
extern int  textpr[11];                /* row positions of scale # */
extern int  titlebar;       /* Switch to turn off title bar for datac=7 */
extern double timelapse;
extern float tmp[2][2];     /* Prevents old data from being recalculated */
extern float torcal;                /* Calibration reading, torque */
extern float torcal_z[15];       /* 1st val is len, torque zeros storage */
extern float torcal_s[15];     /* 1st val is len, torque shunts storage */
extern char typecal;           /* Input variable for calibration type */
extern int  u6,u7,u8,u9;    /* Switches,tell user what needs to be input */
extern float updatime;
extern int  value;           /* Counter in error trapping for DAS-20 */
extern float vcalin;                     /* Calibrate Voltage in */
extern float velocity;                     /* Velocity, feet/sec */
extern float viscosity;                            /* Viscosity */
extern float voltage[6];                /* Channel voltage readings */
extern int  windoncount;                 /* Wind-on point counter */
extern int  woplotcode;                     /* Wind-on plot code */
extern int  wosymbolc;             /* Wind-on symbol code for plots */
extern float x;           /* Abscissa for plotting all hard copy plots */
extern float xmin;                     /* Wind-on plot x minimum */
extern float xmax;                     /* Wind-on plot x maximum */
extern float xscale;             /* x-axis scale for hard copy plots */
extern float y;           /* Ordinate for plotting all hard copy plots */
extern float y1scale;         /* y-axis scale for upper hard copy plot */
extern float y2scale;         /* y-axis scale for lower hard copy plot */
extern char ytitle[];                   /* y-axis title for plot */
```

```
extern float zero[6];       /* Zero value for real and imag of channels */
             /* special-type variables for graphics, time, and files */
extern struct videoconfig myscreen;   /* predefined structure for graphics */
extern time_t testtime;                   /* Current date and time */
extern time_t oldtime;
extern FILE *filename;           /* For Dynamic Stability Data Storage */
extern FILE *formated;            /* For Formated Output data storage */
extern FILE *testval;                      /* Configuration file */
extern FILE *fi;              /* Error message file for DAS-20 board */
extern FILE *calfile;               /* Storage for calibration data */
extern FILE *dconfig;             /* plot/average configuration file */
extern FILE *sens;               /* semsitivities configuration file */
extern FILE *calival;               /* Storage for calibration data */
extern FILE *tareval;                    /* Storage for tare data */
extern FILE *fsplin;                   /* Spline gain file pointer */
extern FILE *ac_cal;                    /* AC calibration storage */
                    /* functions referenced before being defined */
extern   output();
extern   windonplot();
extern   ppcpk();
extern   read_gain();
extern   keycommand();
extern   newx();
extern   switchboard();
extern   realimag();
extern   readaoa();
extern   setnew();
extern   getdatacode();
extern   scale();
extern   refresh();
```

Appendix B

First Source File Listing:

```
/*********************************************************************
   xma.c

   first of six files to control all functions of user interface for
   digital Dynamic Stability Data Acquisition and Reduction system.

   Read the file dsphead.h for more information.

   tsd 2/10/93
   td&jt 3/3/93
   tsd 4/19/93
*********************************************************************/

#include "dsphead.h"     /* establishes defines, variables, and functions */
                       /* global variables in alphabetical order */
char accalfile[12]="dspnac.mat";
char *acfile=accalfile;           /* filename for ac cal Data Storage */
float acnorm;                             /* Keithley reading */
long addr[2];                 /* DSP address for data from dynstat */
long addr_plot[3];                     /* DSP addresses fft data */
long addr_realf[2];                   /* DSP addresses of "realf" */
float alphabias;                              /* Alpha bias */
float alphadelt;                             /* Delta Alpha */
float alphaoff;                             /* Alpha offset */
float alphasens;         /* AoA accelerometer const. Alpha sensitivity */
int  ampchan=0;                   /* Amplifier channel indicator */
int  amplif=0;                       /* Amplifier on-off switch */
/* float ampratio; */                     /* Ratio of peak amplitudes */
/* float ampratiosave; */                   /* Averaged amplitude ratio */
float amr;                /* Secondary channel corollary to kspring */
float angleav[2][121];           /* averaging arrays for phase angles */
float anorm0[2]={1.0,1.0};             /* Gain normalization factors */
float anorm1[2]={1.0,1.0};
float tanorm0[2];
float tanorm1[2];
float afreq;
float AoAval;                             /* Angle of Attack */
float aoavolts;           /* Raw voltage reading from angle of attack */
float array[VALUES];               /* Buffer array, for dsp_up_array */
float atten[6];            /* brings voltage to scale of highest gain */
/* float avamprat[2][121]; */            /* averaging arrays for ampratio */
float avaoavolts[121];               /* averaging array for aoavolts */
int  aveorimed;              /* Control: averaged or immediate data */
int  NAVE;                 /* # of data values in running average */
float aveaveang[2][121];          /* Arrays for storing averages of... */
/* float aveaveamp[2][121]; */          /* ...averages for finding st. dev. */
float aveavepeak0[2][121];
float aveavepeak2[2][121];
int  avecount=0;           /* Counter for preloading averaging arrays */
float avimag[2][121];           /* averaging array for imag components */
float avpeak[2][8][121];            /* averaging arrays for all peaks */
float avreal[2][121];           /* averaging array for real components */
int  b=0;               /* Differentiates between boards/their data */
long baddr[2]={0x300,0x320};           /* I/O addresses of DSP boards */
int  beentoaoa;          /* Switch tells program it has just read AoA */
float bmr;                /* Secondary channel corollary to inertia */
char buffer[80];                            /* Output text string */
int  c;                 /* Differentiates between channels to plot */
float c1,c2,c3,c4;        /* values to compute aerodynamic coefficients */
float calib[3];                        /* calibration constants */
int  calibnum;                          /* Calibration number */
float calindep[121];                      /* Data entered by hand */
float calarray0[121];           /* storage for calibrate function data */
float calarray1[121];           /* storage for calibrate function data */
```

```
int  calcode=10;       /* Display code for specifying type of cal data */
float calcoffa0;                /* display and storage ls 'a' value */
float calcoffa1;                /* display and storage ls 'b' value */
float calcoffb0;                /* display and storage ls 'a' value */
float calcoffb1;                /* display and storage ls 'b' value */
char cal_name[20];           /* storage for name of calibration type */
int  calpoint;                         /* used in takecaldat() */
int  calpoint_d;                 /* Calibration data point counter */
int  calpoint_s;                /* Calibration shunt point counter */
int  calpoint_z;                 /* Calibration zero point counter */
int  calres=0;                    /* Calibration resistor switch */
float ctare;                                /* Tare damping */
float cf[2];                         /* Gain correction values */
float cd[2];
float cforce[2];                    /* Corrected force and moment */
float cdisp[2];                       /* Corrected displacement */
long dacvalue=0;                  /* Returned hex value of offset */
int  data[3];                     /* Data i/o from DAS-20 board */
int  datacode;                    /* Data code from DAS-20 board */
/* float devampratio; */           /* standard deviation of amplitude ratio */
float devaoa;                   /* Standard deviation of aoavolts */
float devaveang;                    /* Deviations of the averages */
/* float devaveamp; */
float devavepeak0,devavepeak2;
float deviation;                /* Stores st. dev. from findaverage */
float devimag[2];                   /* St. Dev. of imag components */
float devpeak[2][2];            /* standard deviation of fundamentals */
float devreal[2];                   /* St. Dev. of real components */
float dfreq;                           /* used in gnormalize() */
float discal0;            /* Calibration reading, displacement Brd. 0 */
float discal0_z[15];   /* 1st elem is len. storage for displacement0 zeros */
float discal0_s[15];         /* 1st int. is len., displacement0 shunts */
float discal1;            /* Calibration reading, displacement Brd. 1 */
float discal1_z[15];           /* 1st val is len, displacement1 zeros */
float discal1_s[15];           /* 1st val is len, displacement1 shunts */
                      /* Scaling divisions for all cases of plots */
float div[2][2][11]={{{0.0,-20.0,-40.0,-60.0,-80.0,0.0,6.1,12.2,18.3,24.4,30.5},
       {0.0,-20.0,-40.0,-60.0,-80.0,0.0,12.2,24.4,36.6,48.8,61.0}},
       {{0.0,-20.0,-40.0,-60.0,-80.0,0.0,12.2,24.4,36.6,48.8,61.0},
      {0.0,-20.0,-40.0,-60.0,-80.0,0.0,24.4,48.8,73.2,97.7,122.1}}};
float dvpk[2][2];       /* Storage for devs of averages for noise record */
int  e;                /* Differentiates between scaling on x-axis */
int  each=250;                     /* How many FFT points to plot */
int  endx;                  /* pixel coordinate of last x value */
int  error;               /* Error code returned from DAS-20 board */
float fft[MAXPNTS*2];              /* Upload array for fft plot data */
float flow_angle;                /* correct AoA for UPWT flow angle */
float forcem[2];
float freqg[101];           /*Frequency column of gain correction table*/
float displm[2];
char fname[13]="aaaaaaaa.aaa";
char *file=fname;                    /* filename for Data Storage */
float gain0[2][101];                    /* A/D gain table Chan 1 */
float gain1[2][101];                    /* Chan 2 */
float g0[2];
float g1[2];
char *format;                /* filename for formatted data storage */
char *cformat;                   /* filename for cal data storage */
float finalave;             /* Stores averaged value from findaverage */
float freq;                          /* frequencies of peaks */
int  functionkey;           /* User controlled directions for program */
                           /* Gain of programmable amplifiers */
float gain[6]={1.0,1.0,1.0,1.0,1.0,1.0};
char gaincode;                  /* Code letter for amplifier gain */
int  highx=325;                     /* Scaling factor for x-axis */
int  i;                    /* general use loop counting variable */
float imag[2];               /* Imaginary w/ respect to Displacement */
float imagsave[2];           /* save averaged values of imag component */
int  imax;                   /* Size of gain interpolation table */
float indepvar;            /* variable used to hold hand entered data */
```

```
float inertia;                                  /* Inertia */
int  inverted=0;                 /* Model inverted switch for AoA */
int  keyswork;          /* Turns off functionkey sensitivity to plots */
int  keith;                              /* Keithley flag */
int  key;                        /* Key-in for gain correction */
float kspring;                              /* Spring constant */
int  labelwoplot;         /* Switch, controls labeling of windon plots */
int  lastcalpt;           /* Switch, last calibration point was taken */
float lscoffa;                    /* leastsq() linear coefficient */
float lscoffb;                   /* leastsq() constant coefficient */
float lsstdev;                    /* leastsq() standard deviation */
float m1;                  /* mr/d or other specified coefficients */
float m2;                 /* mj/wd or "                   */
float mach;                          /* Wind tunnel mach number */
int  maxx, maxy;           /* Number of pixels in x and y directions */
float mjwdtare;                             /* Damping factor */
int  mode;                            /* DAS-20 board mode */
float momentarm;                 /* user entered moment arm length */
float offset[5]={0.0,0.0,0.0,0.0,0.0};      /* Amp offset from auto-zero */
int  offrange=0;               /* value indicating full scale range */
float omega;                        /* frequency in radians/sec */
int  p;                   /* Differentiates between DSP programs */
float peak[8];              /* magnitudes of real and imaginary data */
float peaksave[8];        /* save average values for standard deviation */
float phase[2][101];                      /* Phase calibrate table */
float ph[2];                              /* Phase correction */
float phdif;                  /* Phase angle difference in degrees */
float phdifm[2];                     /* Uncorrected phase shift */
float phdifsave;                 /* Average phase angle in degrees */
float phifft[2],phideg[2];         /* Phase angles for peak amplitudes */
float pk;                                  /* (f*cos(eta)/d */
float pnt[MAXPNTS];                 /* Array for dB levels for plot */
float ppc;                             /* (f*sin(eta)/freq*d) */
float pressure;                         /* Wind tunnel pressure */
char *program="search12";     /* board0 initialize to longest prog. name */
char *program2="serch122";              /* program for 2nd DSP board */
int  ptnum;                                 /* Point number */
float Q;                          /* Dynamic pressure, lb/ft^2 */
float reading[6];                     /* scaled voltage readings */
float real[2];                  /* Real w/ respect to Displacement */
float realsave[2];           /* save averaged values of real component */
float refarea;                         /* Reference wing fsarea */
float reflength;                            /* Reference length */
float reynolds;                      /* Reynolds number per foot */
int  runnum;                                  /* Run number */
float seccal;                   /* Calibration reading, secondary */
float seccal_z[15];         /* 1st val is len, secondary zeros storage */
float seccal_s[15];         /* 1st val is len, secondary shunts storage */
float sen[6];                        /* calculated sensitivities */
int  shuntopt;                         /* Shunt resistor option */
float sqrt2=1.414213562;
float statemp;                         /* Static temperature °R */
float standev0;                    /* display and storage ls 'std' */
float standev1;                    /* display and storage ls 'std' */
float stdev;              /* St deviation of phase angle differences */
float t1;                   /* tr/d or other specified coefficient */
float t2;                   /* tj/wd or "                   */
int  tarecount;                      /* Tare Data Point counter */
float tarelinreg[2][15];     /*Tare linear regressi. variable 0=up,1=low */
float tarex[2][15];         /* Tare plot upper abscissa, omega squared */
float tarey[2][15];               /* Tare plot upper ordinate, mr/d */
float temperature;                     /* Wind tunnel temperature */
int  testnum;                               /* Test number */
                                /* col positions of scale # */
int  textpc[11]={6,6,6,6,6,10,22,34,47,59,71};
                                /* row positions of scale # */
int  textpr[11]={6,10,14,19,24,5,5,5,5,5,5};
int  titlebar;           /* Switch to turn off title bar for datac=7 */
double timelapse;
float tmp[2][2];          /* Prevents old data from being recalculated */
```

```
float torcal;                    /* Calibration reading, torque */
float torcal_z[15];           /* 1st val is len, torque zeros storage */
float torcal_s[15];          /* 1st val is len, torque shunts storage */
char typecal;               /* Input variable for calibration type */
int  u6,u7,u8,u9;         /* Switches,tell user what needs to be input */
float updatime;
int  value;                /* Counter in error trapping for DAS-20 */
float vcalin;                        /* Calibrate Voltage in */
float velocity;                        /* Velocity, feet/sec */
float viscosity;                             /* Viscosity */
float voltage[6];                    /* Channel voltage readings */
int  windoncount;                     /* Wind-on point counter */
int  woplotcode;                        /* Wind-on plot code */
int  wosymbolc;                  /* Wind-on symbol code for plots */
float x;               /* Abscissa for plotting all hard copy plots */
float xmin;                          /* Wind-on plot x minimum */
float xmax;                          /* Wind-on plot x maximum */
float xscale;                  /* x-axis scale for hard copy plots */
float y;               /* Ordinate for plotting all hard copy plots */
float y1scale;              /* y-axis scale for upper hard copy plot */
float y2scale;              /* y-axis scale for lower hard copy plot */
char ytitle[]="Amplitude dB V";            /* y-axis title for plot */
float zero[6];            /* Zero value for real and imag of channels */
             /* special-type variables for graphics, time, and files */
struct videoconfig myscreen;       /* predefined structure for graphics */
time_t testtime;                     /* Current date and time */
time_t oldtime;
FILE *filename;               /* For Dynamic Stability Data Storage */
FILE *formated;                 /* For Formated Output data storage */
FILE *fsplin;                       /* Spline gain file pointer */
FILE *testval;                        /* Configuration file */
FILE *fi;                   /* Error message file for DAS-20 board */
FILE *calfile;                    /* Storage for calibration data */
FILE *dconfig;                  /* plot/average configuration file */
FILE *sens;                  /* semsitivities configuration file */
FILE *calival;                    /* Storage for calibration data */
FILE *tareval;                       /* Storage for tare data */
FILE *ac_cal=NULL;                   /* AC calibration storage */

main()
{
 /* Infinite program loop which calls four primary data acquisition
    and display functions, and calls key command interpreter */
 int   qflag;
 char  testbuff[5];
 struct {
     unsigned dummy  : 16;
     unsigned excflag : 1;
     unsigned dummy2 : 7;
 } amp_status;
        /* Read all old configuration settings and constants from files */
                            /* Read old calibration constants */
 calival=fopen("c:\\data\\calival.dat", "r");
 fscanf(calival,"%d\n%f\n",&calibnum,&momentarm);
 fclose(calival);
                     /* Read other settings from last program run */
 testval=fopen("c:\\data\\testval.dat","r");
 fscanf(testval,"%f\n%f\n%f\n%f\n%f\n%f\n%d\n%d\n%f\n%f\n%f\n%f\n%f\n",
   &alphasens,&alphabias,&alphaoff,&alphadelt,&reflength,&refarea,
   &testnum,&runnum,&mach,&pressure,&temperature,&sen[0],&sen[1],&sen[2]);
 fclose(testval);
                          /* read old configuration settings */
 dconfig = fopen("c:\\data\\dconfig.dat", "r");
 fscanf(dconfig,"%d\n%d\n%d\n%d\n%d\n%d\n",&each,&highx,&c,&p,&aveorimed,&NAVE);
 fclose(dconfig);
                              /* Read old sensitivities */
 sens=fopen("c:\\data\\sens.dat", "r");
 fscanf(sens,"%f\n%f\n%f\n%f\n%f\n%f\n%f\n%f\n",&calib[0],&calib[1],
   &calib[2],&afreq,&anorm0[0],&anorm0[1],&anorm1[0],&anorm1[1]);
 fclose(sens);
```

```
tareval=fopen("c:\\data\\tareval.dat", "r");
fscanf(tareval,"%f\n%f\n%f\n%f\n%f\n%f\n",&kspring,&inertia,&ctare,
  &amr,&bmr,&mjwdtare);
fclose(tareval);
tmp[0][0]=0;             /* First values in DSP arrays are zeros... */
tmp[0][1]=0;
tmp[1][0]=0;        /*...do not attempt calculations with these zeros */
tmp[1][1]=0;
  /* Initialize the DAS-20 board. Set the base address, DMA channel, and
  Interrupt Level. It also resets the ADC and Sample Control Queue, sets
 input gain to 1X/Bipolar, selects input channel 0, and resets the timer. */
mode = 0;                                /* initialize mode */
data[0] = 0x340;                              /* base address */
data[1] = 5;                       /* interrupt level-not used */
data[2] = 3;                               /* DMA channel */
value = 1;
while(value < 6)  {               /* try 5 times to talk to board */
   switch(value)  {
   case 1:
   case 2:
   case 3:
   case 4:
    if ((error = das20(mode,data)) !=0) /*call DAS-20*/
      value++;
    else
      value = 7;
   break;
   case 5:
    fi = fopen("DAS20err","a");    /* open file for error code output */
    fprintf(fi, "\nDAS-20 initialization error number ");
    fprintf(fi, "%d\n", error);
    fclose(fi);
    value = 7;
   break;
   }
}
if (ieeeinit()==-1) {                /* Initialize IEEE-488 board */
  printf("Cannot initialize IEEE system.\n");
  exit(1);
}
printf("IEEE-488 board Ready!\n");
if((fsplin=fopen("gphsplin.mat","r")) != NULL) {
  i=0;
  while(fscanf(fsplin,"%f %f %f %f %f %f %f\n",&freqg[i],&gain0[0][i],
   &gain1[0][i],&gain0[1][i],&gain1[1][i],&phase[0][i],&phase[1][i])
   != EOF && i<=100) i++;
  imax=i-1;
  fclose(fsplin);
}
else exit(1);
if(ieeeinit()==-1) {             /* Initialize Keithley IEEE driver */
  printf("Cannot initialize ieee.\n");
  exit(1);
}
ieeewt("hello\n");
ieeerd(buffer);
printf("%s\n",buffer);
ieeewt("status\n");
ieeerd(buffer);
printf("%s\n",buffer);
ieeewt("remote 16\n");
if(ieeewt("output 16;*IDN?\n")==-1) {
  printf("Keithley Unavailable. Enter 1 to continue\n");
  scanf("%d",&key);
  if(key !=1)
   exit(1);
  keith=0;
}
else {
  keith=1;
```

```
  ieeewt("enter 16\n");
  ieeerd(buffer);
  printf("%s\n",buffer);
  ieeewt("output 16;:FUNC 'VOLT:AC';FUNC?\n");
  ieeewt("enter 16\n");
  ieeerd(buffer);
  ieeewt("output 16;:VOLT:AC:NPLC MAX;NPLC?\n");
  ieeewt("enter 16\n");
  ieeerd(buffer);
  ieeewt("output 16;:VOLT:AC:DIG 7;DET LFRM;DET?\n");
  ieeewt("enter 16\n");
  ieeerd(buffer);
  ieeewt("output 16;:VOLT:AC:COUP DC\n");
  ieeewt("output 16;:VOLT:AC:AVER:STAT ON\n");
  ieeewt("output 16;:INIT:CONT ON\n");
  ieeewt("output 16;:CALC:FORM MXB;FORM?\n");
  ieeewt("enter 16\n");
  ieeerd(buffer);
  ieeewt("output 16;:CALC:KMAT:MMF 1.414214\n");
  ieeewt("output 16;:CALC:STAT ON\n");
  ieeewt("output 16;volt:dc:nplc max;dig 7;aver:stat on\n");
}
ieeewt("clear 10\n"); /* Initialize and put the 8000 PSC at bus address 10 into ECHO OFF */
ieeewt("remote 10\n");
ieeewt("output 10;*000HF\n");
qflag=0;
for(i=0;i<3;i++) {
  sprintf(buffer,"output 10;*00%1dAL\n",i);
  ieeewt(buffer);
  ieeewt("enter 10\n");
  ieeescnf("%s",buffer);
  testbuff[4]=buffer[6];       /* adjust buffers */
  testbuff[i]=buffer[7];
  if(testbuff[i] != 'K')
    qflag = 1;
}
for(i=0;i<3;i++) {      /* set level of bridge excitation power supplies */
  OUT"output 10;*00%1dSE55\n",i);/* make sure hardware jumpers are set for 15 volt range */
  ieeewt(buffer);
  OUT"output 10;*00%1dFT1\n",i);     /* and select anti-alias filter #1 */
  ieeewt(buffer);
}
OUT"output 10;*FFFMF\n");         /* make sure monitor relays are off */
ieeewt(buffer);
/* read amplifier gains, attenuations, and offsets */
for(i=0;i<3;i++) {
  ampchan=i;
  read_gain();
  atten[i]=10000./gain[i];
  autozero_read();
}
ampchan=0;
amplif=0;
for(i=0;i<3;i++) {           /* determine if excitation is on or off */
   OUT"output 10;*00%1dSO\n",i);
  ieeewt(buffer);
  ieeewt("enter 10\n");           /* reads value from IEEE device 10 */
  ieeescnf("%*6s%6lx",&amp_status);
  if(amp_status.excflag==0)
   amplif++;
}
if(amplif>0)         /* if any of the amps are exciting its bridge... */
  amplif=0;
else
  amplif=1;
if(qflag == 0)
  printf("\nPSC-8000 Ready!\n");
else
  printf("\n   PSC-8000 Reports an Error.");
system("print /t");           /* Initialize PC --> Laserjet driver */
```

```
       /* Change graphics mode to VGA and initialize graphics options */
 _setvideomode(_VRES16COLOR);          /* Change to graphics VGA mode */
 _getvideoconfig (&myscreen);             /* Get screen information */
 maxx=myscreen.numxpixels -1;    /* Set max number of pixels for x and y */
 maxy=myscreen.numypixels -1;
 CLS                                        /* Clear screen */
 TEXT                        /* Set text color to bright yellow */
 BLACKBAK                       /* Set background color to black */
 _setcolor(14);                /* Set plot graphics to bright yellow */
 setnew();                          /* Main user menu function */
 while (1) {                            /* Infinite program loop */
    if(p<0)                            /* p defined in setnew() */
    plot();              /* Uploads FFT values and plots to screen */
   else if (p==4 || p==5)
    dynstat();                          /* Heterodyne routine */
   else if(p>5)
    calibrate();                        /* Calibration with DC */
   else
    fftcontrol();         /* Works with all versions of fft programs */
   keycommand();           /* Controls program through function keys */
 }     /* end of infinite loop */
}     /* end of main */

ac_calibrate()
{                              /* ac calibration using Keithley */
 CLS
 if(ac_cal==NULL)  {
   ac_cal=fopen(acfile,"a");
 }
 TEXT OUT"\nFile %s Opened",acfile); GO
 if(keith) {
   ieeewt("output 16;calc:data?\n");
   ieeewt("enter 16\n");
   ieeescnf("%f",&vcalin);
   OUT"\n AC calibrate voltage = %8.6f Enter 1 to Accept ",vcalin); GO
   scanf(" %d",&key);
   CLS
 }
 else  {
   OUT"\n ENTER RMS AC Calibrate Voltage "); GO
   scanf("%f",&vcalin);
   CLS
   key=1;
 }
 if(key==1)  {
   STORE_AC"%8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f \
   %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7f %8.7\n",
   freq,vcalin,cforce[0]/cf[0],cdisp[0]/cd[0],cforce[0],cdisp[0],
   cforce[1]/cf[1],cdisp[1]/cd[1],cforce[1],cdisp[1],devpeak[0][0],
   devpeak[0][1],devpeak[1][0],devpeak[1][1],dvpk[0][0],dvpk[0][1],
   dvpk[1][0],dvpk[1][1],phdif);
 }
}
```

Appendix C

Formats for Configuration Files

Configuration files are listed below. Each file lists the names of the variables that are used by the software. These files are formatted with all variables in one column in the order listed. The last file, GPHSPLIN.MAT, is formatted as a two dimensional matrix of values with each column containing all of the values for that particular variable. Columns are formatted from left to right in the order listed below.

File name	variable name	type	typical value
CALIVAL.DAT	*calibnum*	integer	1
	momentarm	float	1.00000
TESTVAL.DAT	*alphasens*	float	0.494500
	alphabias	float	-0.857800
	alphaoff	float	-3.550000
	alphadelt	float	4.180000
	reflength	float	2.800000
	refarea	float	1.117188
	testnum	float	1
	runnum	float	0
	mach	float	0.800000
	pressure	float	14.805000
	temperature	float	114.000000
	sen[0]	float	0.000006
	sen[1]	float	-0.033111
	sen[2]	float	-0.034776
DCONFIG.DAT	*each*	integer	250
	highx	integer	325
	c	integer	0
	p	integer	1
	averorimed	integer	1
	NAVE	integer	20
SENS.DAT	*calib[0]*	float	0.020650
	calib[1]	float	14.150000
	calib[2]	float	14.860000
	afreq	float	0.000000
	anorm0[0]	float	1.000000
	anorm0[1]	float	1.000000
	anorm1[0]	float	1.000000
	anorm1[1]	float	1.000000
TAREVAL.DAT	*kspring*	float	154.0000
	inertia	float	0.220000
	ctare	float	0.020000
	amr	float	739.0000
	bmr	float	0.159000

	mjwdtare	float	-0.511000
GPHSPLIN.MAT	*freqg*	float	2.8999939e+01
	gain0[0][n]	float	9.7341658e-01
	gain0[0][n]	float	9.7421847e-01
	gain0[0][n]	float	9.7549392e-01
	gain0[0][n]	float	9.7487324e-01
	phase[0][n]	float	-1.4000000e-02
	phase[1][n]	float	-3.0000000e-04

Appendix D

Second Source File Listing:

```
/* xag.c */
/* Contains functions whose names begin with a-g. */
#include "dsphead.h"
aoa()
{            /* Provide continuous display of angle of attack,voltage,& st. dev. */
 CLS
 beentoaoa=1;        /* returns program to setnew after aoa is found */
 while(!kbhit()) {
   readaoa();
   findaverage(aoavolts,avaoavolts);             /* average aoavolts */
   devaoa=deviation;                 /* Save deviation of aoavolts */
                          /* Display AoA, aoavoltage and st.dev */
   OUT"Angle of Attack"); AT(2,26); GO
   OUT"AoA = %8.3f°    ",AoAval); AT(5,6); GO
   OUT"AoA voltage reading = %11.5f V   ",aoavolts); AT(7,6); GO
   OUT"Standard Deviation in voltage reading = %11.5f V   ",devaoa);
   AT(9,6); GO
   OUT"Press ENTER to return."); AT(14,24); GO
 }                              /* Jump out of infinite loop */
 getchar();  FL  CLS
}

autozero_read()
{          /* reads offset from selected programmable amplifier, minimum range */
 OUT"output 10;*00%1dAS\n",ampchan);
 ieeewt(buffer);
 ieeewt("enter 10\n");            /* reads value from IEEE device 10 */
 ieeescnf("%*6s%1d%5lx",&offrange,&dacvalue);
/* printf("Range code is %d and offset is %lx",offrange, dacvalue); */
 if ( offrange==0 )
   offset[ampchan]=0.0;
 else if ( offrange==1 )
   offset[ampchan]=.075*(((float)dacvalue/2 - 262144) / 262144 );
 else if ( offrange==2)
   offset[ampchan]=1.66667*(((float)dacvalue/2 - 262144) / 262144 );
 else if ( offrange==4 )
   offset[ampchan]=.45*(((float)dacvalue/2 - 262144) / 262144 );
 else {
   CLS
   AT(15,1);
   OUT"Invalid range returned from amp. Hit Enter. >>>"); GO
   getchar(); FL
 }
}

caldatacode()
{                  /* F5 key response to allow user to change cal data code */
 char  gg;
 CLS
 AT(10,10); OUT"Calibration Data (calcode=12)......1"); GO
 AT(12,10); OUT"Non-Regression Zero (calcode=10)......2"); GO
 AT(16,15); OUT"Enter Choice  >>>"); GO
 gg=getchar();
 FL
 switch(gg)  {
   case '1': calcode=12; break;
   case '2': calcode=10; break;
 }
 CLS
}

calibrate()
{                                            /* Automated calibration procedure */
 /* Display Loop for the DC values from DSP boards to the screen, use the
```

```
  F10 key to take calibration points just as it is with other data. */
/* Now uses dynstat routine in DSP's */
long count=24;                    /* Upload 12 floats = 24 ints */
int row;                 /* Row to output Secondary/Torque data */
keyswork=1;            /* Code used in keycommand() for calibrate() */
TEXT OUT"DC Calibration"); AT(1,1); GO
OUT"Data Point Number "); AT(25,1); GO
OUT"Zero Point Number "); AT(26,1); GO
OUT"Shunt Point Number "); AT(27,1); GO
AT(26,30);             /* remind user of last data entered by hand */
OUT"Last Independent Data:"); GO
OUT"Calcode = "); AT(28,1); GO
OUT"Shunt Option = "); AT(28,30); GO
OUT"Reading (VDC)"); AT(5,18); GO
OUT"Std. Dev. (VDC)"); AT(5,38); GO           /* Set up framework... */
OUT"Last Zero (VDC)"); AT(5,58); GO           /* Set up framework... */
OUT"Displacement(0):"); AT(7,1); GO           /* ...for data */
OUT"Torque:"); AT(9,10); GO
OUT"Displacement(1):"); AT(11,1); GO
OUT"Secondary:"); AT(13,7); GO
if(keith) {
  ieeewt("output 16;func 'volt:dc'\n");
  ieeewt("output 16;CALC:STAT OFF\n");
}
/* Execute data acquisition loop until keyboard is hit, then return
   control to main() and interpret user command using keycommand() */
while(!kbhit())  {
  /* Upload DC values from DSP routine */
  dsp32c_up_array(addr[b],count,(int*)array);
  time(&testtime);                                /* Get current time */
  /* If data is new...*/
  if (tmp[b][0] != array[1] || tmp[b][1] != array[2]) {
   tmp[b][0]=array[1];                               /* temp=new data */
   tmp[b][1]=array[2];
   freq=array[0];
   omega=2*PI*freq;
   if(omega==0.) omega=1.0e-4;
   for(i=4;i<=7;i++)  {        /* Copy real and imaginary of f and d */
    peak[i]=array[i+4];
    if(freq==0.) peak[i]=0.5*peak[i];
   }                     /* average amplitudes to find standard deviation */
   peak[0]=sqrt(peak[4]*peak[4]+peak[5]*peak[5]);
   peak[2]=sqrt(peak[6]*peak[6]+peak[7]*peak[7]);
   gaininterp();                  /* Correct gain for frequency */
   findaverage(peak[0],avpeak[b][0]);
   peaksave[0]=finalave;
   devpeak[b][0]=deviation;
   findaverage(peak[2],avpeak[b][2]);
   peaksave[2]=finalave;
   devpeak[b][1]=deviation;
   for(i=4;i<8;i++)     {
    findaverage(peak[i],avpeak[b][i]); /* Real/Imag f and d in */
    peaksave[i]=finalave;                /* peak[4] - peak[7] */
   }
   peaksave[0]=sqrt(peaksave[4]*peaksave[4]+peaksave[5]*peaksave[5]);
   peaksave[2]=sqrt(peaksave[6]*peaksave[6]+peaksave[7]*peaksave[7]);
   if(aveorimed)   {
    peak[0]=peaksave[0];
    peak[2]=peaksave[2];
   }
   if(array[8]<0) peak[0]=-peak[0]; /* Correct DC sign of torque/secondary*/
   if(array[10]<0) peak[2]=-peak[2]; /* Correct DC sign of displacement */
   cforce[b]=peak[0];
   cdisp[b]=peak[2];
   if(peak[4]==0. && peak[5]==0.) {      /* rudimentary error checking */
    peak[4]=0.0001;
    peak[5]=0.0001;
   }
   if(peak[6]==0. && peak[7]==0.)    {
    peak[6]=0.0001;
```

```
     peak[7]=0.0001;
    }
    phdif=atan2(peak[5],peak[4])-atan2(peak[7],peak[6]);
    phdif=phdif*180./PI-ph[b];
    if (phdif > 180.) phdif=phdif-360.;    /* Keep range of phase angle...*/
     if (phdif < -180.) phdif=phdif+360.;     /*...between -180 and 180 */
    findaverage(peaksave[0],aveavepeak0[b]);     /* Average the averages*/
    dvpk[b][0]=devavepeak0=deviation;            /* Save deviations */
    findaverage(peaksave[2],aveavepeak2[b]);
    dvpk[b][1]=devavepeak2=deviation;
    avecount++;             /* Prevents overwriting of ave. arrays */
    findaverage(phdif, angleav[b]);          /* Find angle deviation */
    phdifsave=finalave;                       /* Save average */
    stdev=deviation;                  /* Save standard deviation */
    if(aveorimed)
       phdif=phdifsave;
    findaverage(phdifsave,aveaveang[b]);       /* Average the average */
    devaveang=deviation;                   /* Save the deviation */
    realimag();
    if (b==0) {                    /* Output DC values to screen */
     row=7;                       /* Board0 is in rows 8 & 10 */
     discal0=peak[2];           /* Displacement calibration reading */
     torcal=peak[0];               /* Torque calibration reading */
    }
    else     {                          /* other board, b=1 */
     row=11;                     /* Board1 is in rows 11 & 13 */
     discal1=peak[2];           /* Displacement calibration reading */
     seccal=peak[0];             /* Secondary calibration reading */
    }
    DATA OUT"%10.6f",peak[2]); AT(row,18); GO     /* Output displacement */
    DATA OUT"%10.6f",devpeak[b][1]); AT(row,38); GO   /* Output disp. std.dev. */
    DATA OUT"%10.6f",peak[0]); AT(row+2,18); GO   /* output other channel */
    DATA OUT"%10.6f",devpeak[b][0]); AT(row+2,38); GO   /* output its std. dev. */
    switchboard();
    DATA OUT"%10.6f",discal0_z[calpoint_z]); AT(7,58); GO
    DATA OUT"%10.6f",discal1_z[calpoint_z]); AT(11,58); GO
    DATA OUT"%10.6f",torcal_z[calpoint_z]); AT(9,58); GO
    DATA OUT"%10.6f",seccal_z[calpoint_z]); AT(13,58); GO
              /* Show user how many calibration points have been taken */
    DATA OUT"%d",calpoint_d+1); AT(25,19); GO
    DATA OUT"%d",calpoint_z+1); AT(26,19); GO
    DATA OUT"%d",calpoint_s+1); AT(27,20); GO
                        /* remind user of last data entered by hand */
    DATA OUT"%13.6f",calindep[calpoint_d]); AT(26,62); GO
    DATA OUT"%d",calcode); AT(28,12); GO            /* output calcode */
    DATA OUT"%d",shuntopt); AT(28,45); GO        /* output shunt option */
    TEXT OUT ctime(&testtime)); AT(1,40); GO           /* Output time */
/*    if(typecal=='5')  */              /* auto timed calibrate code */
/*    {
     timelapse=difftime(testtime,oldtime);
     if(timelapse > updatime*60-1 && calpoint < 1001)
     {
       oldtime=testtime;
       STORE"%f %f %f %f %f \n",vcalin,discal0,seccal,discal1,torcal);
       calpoint++;
     }
     TEXT OUT"Last point autorecorded at "); AT(25,24); GO
     OUT ctime(&oldtime)); AT(25,52); GO
    } */
   }
   if(calpoint_d>0 || calpoint_z>0 || calpoint_s>0)  {    /* tell user what type */
     TEXT OUT"%s Calibration\n",cal_name);
     AT(3,1); GO
    }
   if(lastcalpt && calpoint_d > 3)   {
    AT(15,5);
    OUT"Regression (LLS) Terms for Data Model: y=mx+b.\n\n"); GO
    OUT"%s calibration yields the following results:\n\n",cal_name); GO
    if(typecal=='1')    {
     OUT"m(0) = %13.6f, b(0) =%13.6f, and Std Err(0)=%13.6f\n",calcoffa0,
```

```
        calcoffb0,standev0); GO
     OUT"m(1) = %13.6f, b(1) =%13.6f, and Std Err(1)=%13.6f\n",calcoffa1,
        calcoffb1,standev1); GO
    }
    else      {
     OUT"m = %13.6f, b =%13.6f, and Std Err=%13.6f\n",calcoffa0,
        calcoffb0,standev0); GO
    }
   }
 }
}

dynstat()
{                /* Controls all aspects of synchronous demodulation algorithm */
 long count=24;                            /* Six floats uploaded */
 refresh();                 /* Set up framework for numerical data */
 keyswork=2;             /* keycommand() uses keyswork=2 for dynstat */
 /* Execute data acquisition loop until keyboard is hit, then return
    control to main() and interpret user command using keycommand() */
 while(!kbhit())  {
   dsp32c_up_array(addr[b],count,(int *)array);         /* Upload data */
   /* If data is new...*/
   if (tmp[b][0] != array[1] || tmp[b][1] != array[2])    {
    tmp[b][0]=array[1];                        /* temp=new data */
    tmp[b][1]=array[2];
    freq=array[0];
    omega=2*PI*freq;
    if(omega==0.) omega=1.0e-4;
    for(i=4;i<=7;i++)  {        /* Copy real and imaginary of f and d */
      peak[i]=array[i+4];
      if(freq==0.) peak[i]=0.5*peak[i];
    }                          /* average amplitudes to find standard deviation */
    peak[0]=sqrt(peak[4]*peak[4]+peak[5]*peak[5]);
    peak[2]=sqrt(peak[6]*peak[6]+peak[7]*peak[7]);
    gaininterp();                  /* Correct gain for frequency */
    findaverage(peak[0],avpeak[b][0]);
    peaksave[0]=finalave;
    devpeak[b][0]=deviation;
    findaverage(peak[2],avpeak[b][2]);
    peaksave[2]=finalave;
    devpeak[b][1]=deviation;
    for(i=4;i<8;i++)     {
     findaverage(peak[i],avpeak[b][i]); /* Real/Imag f and d in */
     peaksave[i]=finalave;                /* peak[4] - peak[7] */
    }
    peaksave[0]=sqrt(peaksave[4]*peaksave[4]+peaksave[5]*peaksave[5]);
    peaksave[2]=sqrt(peaksave[6]*peaksave[6]+peaksave[7]*peaksave[7]);
    if(aveorimed)  { /* copies averaged real & imag back into themselves */
      peak[0]=peaksave[0];
      peak[2]=peaksave[2];
      for(i=4; i<8; i++)
        peak[i] = peaksave[i];
    }
    if(freq==0.)    {   /* Correct DC sign of torque/secondary/displacement... */
      peak[0]=peak[4];                /* if looking at DC values, torq, secd */
      peak[2]=peak[6];                /* and disp */
    }
    cforce[b]=peak[0];
    cdisp[b]=peak[2];                  /* compute phase angle of each channel */
                            /* resolve real & imag of torq&second relative to disp */
    if(peak[2] != 0.)    {
     real[b] = (peak[4] * peak[6] + peak[5] * peak[7]) / peak[2];
     imag[b] = (peak[4] * peak[7] - peak[5] * peak[6]) / peak[2];
    }                             /* average real and imag to obtain stan. devs. */
    findaverage(real[b],avreal[b]);
/*    realsave[b] = finalave;  */
    devreal[b] = deviation;
    findaverage(imag[b],avimag[b]);
/*    imagsave[b] = finalave;   */
    devimag[b] = deviation;
```

```
    if(real[b] == 0. && imag[b] == 0.)    /* compute phase angles and correct at freq */
      real[b]=imag[b]=0.001;
    phdif=atan2(imag[b],real[b]) * 180./PI + ph[b];
       /* inherent discontinuity, but keep phase angle between +/- 180 */
    if(phdif > 180.)
     phdif=phdif-360.;
    if(phdif < -180.)
     phdif=phdif+360.;
    findaverage(phdif,angleav[b]);      /* compute average of phase */
    phdifsave=finalave;
    stdev=deviation;
    findaverage(peaksave[0],aveavepeak0[b]);     /* Average the averages*/
    dvpk[b][0]=devavepeak0=deviation;            /* Save deviations */
    findaverage(peaksave[2],aveavepeak2[b]);
    dvpk[b][1]=devavepeak2=deviation;
    avecount++;             /* Prevents overwriting of ave. arrays */
    devaveang=stdev;                   /* Save the deviation */
    readaoa();                        /* find angle of attack */
    findaverage(aoavolts,avaoavolts);      /* find st. dev of aoavolts */
    devaoa=deviation;               /* save st. dev. of aoavolts */
    output();                 /* Output numerical data to screen */
   }
   switchboard();          /* 2nd board can run dynstat (with DSPNET) */
 }
}

fftcontrol()
{                          /* Manipulates and displays all fft program data */
 long count=VALUES*2;         /* how many int to upload from DSP board=
                           twice the number of floats expected */
 refresh();                 /* Set up framework for numerical data */
 keyswork=3;               /* keycommand() uses keyswork=3 for fft */
 /* Execute data acquisition loop until keyboard is hit, then return
   control to main() and interpret user command using keycommand() */
 while(!kbhit())  {
                 /* Upload array of floating points from DSP board */
   dsp32c_up_array(addr_realf[b],count,(int *)array);
   /* If all data is new... */
   if (tmp[b][0] !=array[0] && tmp[b][1] !=array[6])    {
    tmp[b][0]=array[0];                   /* Set new data values */
    tmp[b][1]=array[6];
 /* Compute magnitudes of real and imag values for FFT routines. FFT data
 from the DSP boards is stored in real and imaginary componest in alternating
 memory locations. A DC term, sent when frequency=0 on all zero reading data,
 will be twice it's actual value and is stored only in the real component.
 This has been corrected */
    freq=array[8];               /* Loads frequency from DSP data */
    for(i=0;i<=2;i+=2)     {
       if (p<4)      {
        /* amplitude = sqrt(sum of sqares of real and imag)/# of FFT
                     points * window scaling factor */
          peak[i]=sqrt(array[i*2]*array[i*2]+array[i*2+1]*array[i*2+1]);
          peak[i]=peak[i]/(1024*.5);    /* .5 scaling factor from windows */
          if(freq==0)                      /* If DC correct by 2 */
             peak[i]=peak[i]/2;
       }
    }
    gaininterp();
    for(i=0;i<=2;i+=2)    {
     findaverage(peak[i],avpeak[b][i]); /* average peaks and find st.dev */
     peaksave[i]=finalave;               /* save averaged values */
     if(i==0)
       devpeak[b][0]=deviation;               /* Store deviations */
     if(i==2)
       devpeak[b][1]=deviation;
     /* devpeak [b][0]==Secondary and Torque devpeak[b][1]==Displacement */
    }
    avecount++;       /* Prevents overwriting of averaging array data */
    findaverage(peaksave[0],aveavepeak0[b]);    /* Average the averages */
    dvpk[b][0]=devavepeak0=deviation;            /* Save deviations */
```

```
    findaverage(peaksave[2],aveavepeak2[b]);
    dvpk[b][1]=devavepeak2=deviation;
    /* Now calculate phase angles and averages by computing phase angles
     for FFT at which peak values occur and average, then store frequency */
    for(i=0;i<2;i++)  {        /* Calculate phase angle from FFT data */
      phifft[i]=atan2(array[i*4+1],array[i*4]);
      phideg[i]=phifft[i]*(180./PI);           /* Convert to degrees */
    }
    omega=2*PI*freq;                    /* frequency in rad/sec */
    phdif=phideg[1]-phideg[0];       /* phase difference Disp-Channel */
    phdifm[b]=phdif;          /* Save uncorrected phase difference */
    phdif-=ph[b];          /* Correct phase difference for frequency */
    if (phdif > 180) phdif=phdif-360;    /* Keep range of phase angle...*/
     if (phdif < -180) phdif=phdif+360;     /*...between -180 and 180 */
       findaverage(phdif, angleav[b]);      /* Find ave. and st. dev. */
    phdifsave=finalave;                       /* Save average */
    stdev=deviation;                  /* Save standard deviation */
    findaverage(phdifsave,aveaveang[b]);       /* Average the average */
    devaveang=deviation;                    /* Save the deviation */
    if(aveorimed)  {              /* If user selects averaged data */
         /* Replace immediate data with averages, Changes all data for FFT
        programs from immediate values to averaged */
     for(i=0;i<4;i++)
     peak[i]=peaksave[i]; /* Copy saved averages to immediate addresses */
/*     ampratio=ampratiosave;    */
     phdif=phdifsave;
    /* real[b]=realsave[b]; */
    /* imag[b]=imagsave[b]; */
               /* Save corrected averaged values for gain normalize */
     cforce[b]=peak[0];
     cdisp[b]=peak[2];
    }
    realimag();         /* Calculates real and imaginary and averages */
    ppcpk();                        /* Calculate ppc and pk */
    readaoa();                        /* get angle of attack */
    findaverage(aoavolts,avaoavolts);      /* average to find st. dev. */
    devaoa=deviation;                        /* save st. dev. */
    output();                 /* Outputs numerical data to screen */
   }
   switchboard();           /* Switch to next DSP board, repeat loop */
 }
}

filestore()
{                                          /* Ask user for output file name */
 CLS
 OUT"Please input\n\n Filename for Dynamic Stability Data Storage\n\n ");  AT(3,1); GO
 scanf(" %s",file); FL         /* User inputs numerical data file name */
 sprintf(format,"f%s",file);    /* Name formatted data file for printing */
 sprintf(cformat,"c%s",file);     /* Name calibration file for printing */
 CLS  u9=1;                            /* Turns off white in menu */
}

findaverage(newdata,savedata)
float newdata;                        /* Most recent data value */
float savedata[121];             /* Array of latest values to average */
{              /* Compute running average and standard deviation of argument array */
 int j;                          /* internal for loop counter */
 if(avecount<3)                     /* If new average starting...*/
   for(j=1;j<NAVE+1;j++)                /* load entire average array */
    savedata[j]=newdata;
 for(j=1;j<NAVE+1;j++)               /* move each element down one */
   savedata[j-1]=savedata[j];             /* throw away oldest value */
 savedata[NAVE]=newdata;             /* load new value in last place */
 finalave=0;
 for(j=1;j<NAVE+1;j++)                        /* Add all values */
   finalave=finalave+savedata[j];
 finalave=finalave/NAVE;                  /* Divide by # values */
 /* Calculate standard deviation */
 deviation=0;      /* stdev=sqrt(sum of sqares of errors/# data points) */
```

```
 for(j=1;j<NAVE+1;j++)
   deviation=deviation+(finalave-savedata[j])*(finalave-savedata[j]);
 deviation=sqrt(deviation/NAVE);
}

gaininterp()
{                        /* Correct gain for frequency */
 i=0;
 while(freq > freqg[i] && i<imax) i++;
   if(i == 0)   {
    g0[b]=gain0[b][0];
    g1[b]=gain1[b][0];
    ph[b]=phase[b][0];
   }
   else   {
    dfreq=freqg[i]-freqg[i-1];
    if(dfreq == 0.0) dfreq=1.;
    dfreq=(freq-freqg[i-1])/dfreq;
    g0[b]=(gain0[b][i]-gain0[b][i-1])*dfreq+gain0[b][i-1];
    g1[b]=(gain1[b][i]-gain1[b][i-1])*dfreq+gain1[b][i-1];
    ph[b]=(phase[b][i]-phase[b][i-1])*dfreq+phase[b][i-1];
   }
                   /* Correct force and displacement for AC gain */
   forcem[b]=peak[0];       /* Save uncorrected voltages for calibrate */
   displm[b]=peak[2];
   cf[b]=anorm0[b]/g0[b];
   cd[b]=anorm1[b]/g1[b];
   cforce[b]=cf[b]*forcem[b];
   cdisp[b]=cd[b]*displm[b];
   peak[0]=cforce[b];
   peak[2]=cdisp[b];
   phdif=phdif-ph[b];
   for(i=4;i<=5;i++)   {
    peak[i]=cf[b]*peak[i];           /* Correct real and imag values */
    peak[i+2]=cd[b]*peak[i+2];
   }
}

getdatacode()
{   /* This function reads two channels from the DAS-20 board and two variables from the
program to calculate datacode, updates the LED display on the motor speed control panel. */
 int wind;         /*wind-on*/
 int motor;         /*motor on-off*/
 mode = 14;         /*digital read mode*/
 value = 1;
 while(value < 6) {
   switch(value)   {
    case 1:
    case 2:
    case 3:
    case 4:
     if ((error = das20(mode,data)) !=0) /*call DAS-20*/
       value++;
     else
       value = 7;
    break;
    case 5:
     fi = fopen("DAS20err","a");
     fprintf(fi, "\nDAS-20 digital read error number ");
     fprintf(fi, "%d\n", error);
     fclose(fi);
     value++;
    break;
   }
 }
 if (data[0] == 0)
   wind = motor = 0;
 else if (data[0] == 1)
   motor = 1 + (wind = 0);
 else if (data[0] == 2)
```

```
    wind = 1 + (motor = 0);
  else if (data[0] == 3)
    wind = motor = 1;
  mode = 15;         /*digital write mode*/
  data[0] = amplif * 2 + calres * 1;   /*write data*/
  value = 1;
  while(value < 6) {
    switch(value)   {
     case 1:
     case 2:
     case 3:
     case 4:
     if ((error = das20(mode,data)) !=0) /*call DAS-20*/
      value++;
     else
      value = 7;
     break;
     case 5:
      fi = fopen("DAS20err","a");
      fprintf(fi, "\nDAS-20 digital write error number ");
      fprintf(fi, "%d\n", error);
      fclose(fi) ;
      value++;
     break;
    }
  }
  datacode = amplif * 1 + motor * 2 + wind * 4 + calres * 8;
}

getindepvar()
{        /* Request independent data point from user, used in takecaldat() */
 char  gg;
 char  pp='N';
 CLS
 while(pp=='N') {
/*   if(keith)  {  */  /* commented out for now */
/*  ieeewt("output 16;data?\n");
    ieeewt("enter 16\n");
    ieeescnf("%f",&indepvar);
   }
   else   {   */
    OUT"\n\nEnter numeral value of 'independent variable'.\n"); GO
    OUT"\n   >>>"); GO
    scanf("%f",&indepvar);
    FL
 /*  } */
   OUT"\n\nDC input was %f. Is this correct? (Y or y, N or n)...",
       indepvar); GO
   gg=getchar();
   FL
   pp=(char)toupper(gg);
 }
 CLS
}

gnormalize()
{              /* gain normalization */
 if(keith) {
   ieeewt("output 16;calc:data?\n");
   ieeewt("enter 16\n");
   ieeescnf("%f",&acnorm);
   CLS   OUT"AC Peak Calibrate Volts = %8.6f\n",acnorm);GO
 }
 else  {
   CLS
   OUT"ENTER RMS AC Volts for Gain Normalize\n"); GO
   scanf(" %f",&acnorm);
   acnorm=acnorm*sqrt2;
 }
   /* Correct AC gains from AC voltage standard at some frequency */
```

```
 for(i=0;i<=1;i++)  {
   if(cforce[i] != 0.) tanorm0[i]=acnorm*anorm0[i]/cforce[i];
   if(cdisp[i] != 0.) tanorm1[i]=acnorm*anorm1[i]/cdisp[i];
 }
 OUT"Current Gain Corrections at Freq %8.6f = %8.6f %8.6f %8.6f %8.6f\n",
   afreq,anorm0[0],anorm0[1],anorm1[0],anorm1[1]); GO
 OUT"New Gain Corrections = %8.6f %8.6f %8.6f %8.6f\n",tanorm0[0],
   tanorm0[1],tanorm1[0],tanorm1[1]); GO
 OUT"Enter 2 to Accept, 1 to reset to 1, 0 to discard \n"); GO
   scanf("%d",&key);
 CLS
 if(key==2) {
   for(i=0;i<=1;i++)   {
    anorm0[i]=tanorm0[i];
    anorm1[i]=tanorm1[i];
    afreq=freq;
   }
 }
 else if (key==1)  {
   for(i=0;i<=1;i++)    {
    anorm0[i]=1.;
    anorm1[i]=1.;
   }
 }
}
```

Appendix E

Third Source File Listing:

```
/* file xhp.c */
/* Contains functions whose names begin with h-p */
#include "dsphead.h"

help()
{ /* On-line user help. Print function key information to screen */
 CLS  BLUEBAK  CYANTXT
 OUT"On-line User Information"); AT(1,26); GO
 OUT"F1  On-line Help"); AT(3,7); GO
 OUT"F2  Averaged or Immediate"); AT(5,7); GO
 OUT"F3  Reset Point Counters"); AT(7,7); GO
 OUT"F4  Gain Normalization"); AT(9,7); GO
 OUT"F5  Change Calcode"); AT(11,7); GO
 OUT"F6  Input Sensitivities"); AT(13,7); GO
 OUT"F7  Tare Regression"); AT(15,7); GO
 OUT"F8  Data File Name?"); AT(17,7); GO
 OUT"F9  AoA Display"); AT(19,7); GO
 OUT"F10 Take Data"); AT(21,7); GO
 OUT"F11 Unused"); AT(23,7); GO
 OUT"F12 Select Data File to Print"); AT(25,7); GO
 OUT"A#  Select amplifier (#)"); AT(3,45); GO
 OUT"B#  Bridge power (O)n/of(F)"); AT(5,45); GO
 OUT"G   Set Gain"); AT(7,45); GO
 OUT"S   Shunt calibrate resistor\n"); AT(9,45); GO
 OUT"I   Increment Gain"); AT(11,45); GO
 OUT"U   Un-Shunt calibrate resistor"); AT(13,45); GO
 OUT"D   Decrement Gain"); AT(15,45); GO
 OUT"N   Connect/Disconnect input"); AT(17,45); GO
 OUT"X   Send offset = 0V"); AT(19,45); GO
 OUT"V   Send computed offset"); AT(21,45); GO
 OUT"Z   Amp computes offset"); AT(23,45); GO
 OUT"ESC Return to Main Menu"); AT(25,45); GO
 TEXT OUT"Press ENTER to leave this help screen"); AT(27,20); GO
 getchar(); FL  CLS   BLACKBAK
}

keycommand() /* Lets user control where program goes through function keys */
{
 char   shuntch[4];
 char   gch;
 char   select_char;
 int    qi;
 int    old_ampchan;
 long   *start;
 long   *stop;
 long   off_set;
 float  zero_range;
 functionkey=getch();               /* Read key code from keyboard */
 if (keyswork)     {              /* keyswork= data or calibration */
   if(functionkey=='C' || functionkey=='c')   {
    ac_calibrate();               /* Take an ac calibration point */
    return(0);
   }
 }
 if(functionkey== 'A' || functionkey== 'a')  {
                /* prompts user to select the 'active' amplifier */
          /* amp menu options here */
   do  {
    OUT"Select active amplifier number 0-2 >>>    ");  AT(25,1);  GO
    AT(25,41);   gch=getchar();     FL
    ampchan=(int)gch - 48;
   } while(ampchan < 0 || ampchan > 2);
 OUT"                                  "); AT(25,1);  GO
 return(0);
```

```
 }
 if(functionkey== 'B' || functionkey== 'b') {   /* Bridge Excitation Power Supply On/Off */
   OUT"Select On (O) or Off (F) >>>   ");   AT(25,1);  GO
   AT(25,32);   gch=getchar();   FL
   if(gch=='O'|| gch=='o')   {
     OUT"output 10;*FFFEO\n");
     ieeewt(buffer);
     amplif=1;
     getdatacode();
   }
   if(gch=='F' || gch=='f')    {
     OUT"output 10;*FFFEF\n");
     ieeewt(buffer);
     amplif=0;
     getdatacode();
   }
   OUT"                              ");  AT(25,1);  GO
   return(0);
 }
 if(functionkey== 'S' || functionkey== 's') {
   /* Shunt calibration for sensitivity. Connects a calibration resistor
   across a selected bridge arm in amplifier channel ampchan. ch
   is a 2 character string: +4, +3, +2, +1, 00, -1, -2, -3, or -4.
   See page 72 in Waugh Controls manual. Also update datacode */
   do   {
    OUT"Select: -1 . . . 1   (50K)  +1 . . . 2\n"); AT(21,1); GO
    OUT"      -2 . . . 3  (100K)  +2 . . . 4\n"); GO
    OUT"      -3 . . . 5  (150K)  +3 . . . 6\n"); GO
    OUT"      -4 . . . 7  (200K)  +4 . . . 8 >>>   "); GO
    AT(24,46);    gch=getchar();      FL
    shuntopt=(int)gch-48;
   } while(gch < '0' && gch > '9');
   if(gch=='1') sprintf(shuntch,"-1");
   if(gch=='2') sprintf(shuntch,"+1");
   if(gch=='3') sprintf(shuntch,"-2");
   if(gch=='4') sprintf(shuntch,"+2");
   if(gch=='5') sprintf(shuntch,"-3");
   if(gch=='6') sprintf(shuntch,"+3");
   if(gch=='7') sprintf(shuntch,"-4");
   if(gch=='8') sprintf(shuntch,"+4");
   calres=1;
   getdatacode();
   OUT"output 10;*00%1d%2s\n",ampchan,shuntch);
   ieeewt(buffer);
   calcode=11;
   OUT"                                  \n"); AT(21,1); GO
   OUT"                                  \n"); GO
   OUT"                                  \n"); GO
   OUT"                                      "); GO
   return(0);
 }
 if(functionkey== 'U' || functionkey== 'u') {
   shuntopt=0;           /* Disconnect shunt res.pairs and update datacode */
   calres=0;
   getdatacode();
   sprintf(shuntch,"00");
/*    OUT"output 10;*00%1d%2s\n",ampchan,shuntch); */
   OUT"output 10;*FFF%2s\n",shuntch);
   ieeewt(buffer);
   calcode=10;
   return(0);
 }
 if(functionkey== 'G' || functionkey== 'g') {
   CLS                                /* manually set gain using codes */
   AT(2,15); OUT"Gain Value  Letter Selection"); GO
   AT(4,15); OUT"     1     E or e"); GO
   AT(5,15); OUT"     2     F or f"); GO
   AT(6,15); OUT"     5     G or g"); GO
   AT(7,15); OUT"    10     H or h"); GO
   AT(8,15); OUT"    20     I or i"); GO
```

```
  AT(9,15); OUT"    50     J or j"); GO
  AT(10,15); OUT"   100     K or k"); GO
  AT(11,15); OUT"   200     L or l"); GO
  AT(12,15); OUT"   500     M or m"); GO
  AT(13,15); OUT"   1000     N or n"); GO
  AT(14,15); OUT"   2000     O or o"); GO
  AT(15,15); OUT"   5000     P or p"); GO
  AT(16,15); OUT"  10000     Q or q"); GO
  AT(18,5); OUT"Enter letter designation for desired gain. >>> "); GO
  gch=getchar();    FL
  gaincode=(char)toupper(gch);
  OUT"%c",gaincode); GO
  if(gaincode >= 'E' && gaincode <= 'Q')    {
    OUT"output 10;*00%1dSG%c\n",ampchan,gaincode);
    ieeewt(buffer);
  }
  read_gain();                        /* Find attenuations from gain settings */
  atten[ampchan]=10000./gain[ampchan];
  CLS   refresh();
  return(0);
}
if(functionkey== 'Z' || functionkey== 'z') {
     /* Value of offset DAC is placed in variable offset.
       Sets ONLY function 3 for amplifier selected channel ampchan.
       y       function
       1   Set 15 mV/V range
       2   Set 333 mV/V range
     ** 3   Auto zero command with auto select of range **
       4   Auto zero command with minimum range only
       0 (or any other) Turn Auto zero off. */
  time(start);
  OUT"output 10;*00%1dSA3\n",ampchan);         /* auto select range */
  ieeewt(buffer);/*   MUST wait for amp to go thru cycle   */
  OUT"       WAITING FOR COMPLETION OF AUTOZERO...");  AT(25,1);    GO
  autozero_read();
  do   {;}
  while( time(stop) - (*start) < 11.0 );
  /* OUT"\nHex value of offset: %lx\n",dacvalue); GO
  OUT"Channel %1d autozero setting: %l\n",ampchan,dacvalue); GO
  OUT"Channel %1d offset range code: %d\n",ampchan,offrange); GO
  OUT"Channel %1d DAC value: %10.6f\n\n",ampchan,offset[ampchan]); GO
  OUT"\n    Hit Enter key to end autozero.\n"); GO
  while( getchar() != Enter ) { FL }                    */
  OUT"                                    "); AT(25,1); GO
  return(0);
}
if(functionkey== 'V' || functionkey== 'v') {
  OUT"Answer NO first time thru.\n");   AT(24,1);  GO
  OUT"Send iterative offset adjustment ( y or n )? "); GO
  gch=getchar();
  if( gch == 'Y' || gch == 'y')   {
   if(ampchan==0)             /* displacement offset adjustment */
    offset[0] = offset[0] - (cdisp[0] / gain[0] );
   if(ampchan==1)                 /* torque offset adjustment */
    offset[1] = offset[1] - (cforce[0] / gain[1] );
   if(ampchan==2)               /* secondary offset adjustment */
    offset[2] = offset[2] - (cforce[1] / gain[2] );
  }
  else   {
   if(ampchan==0)                     /* displacement offset */
    offset[0]= ( -1.0 * cdisp[0] );
   if(ampchan==1)                        /* torque offset */
    offset[1]= ( -1.0 * cforce[0] );
   if(ampchan==2)                      /* secondary offset */
    offset[2]= ( -1.0 * cforce[1] );
   offset[ampchan]=offset[ampchan]/gain[ampchan];  /* gain correction */
  }
  if( fabs( offset[ampchan] ) < 0.075 ) {   /* select full scale range */
   zero_range=0.075;
   off_set=524288*(offset[ampchan]/zero_range + 1.0);
```

```
    OUT"output 10;*00%1dSA1%5lX\n",ampchan,off_set);
   }
   else    {
    zero_range=1.666667;
    off_set=524288*(offset[ampchan]/zero_range + 1.0);
    OUT"output 10;*00%1dSA2%5lX\n",ampchan,off_set);
   }
   ieeewt(buffer);
/*    OUT" Offset (hex): %5lX",off_set);
   AT(11,1); GO
   OUT" Offset: %f",offset[ampchan]);   AT(12,1); GO  */
   autozero_read();
/*   OUT"\n    Hit Enter key to return to amp menu.\n"); GO
   gch=getchar();   FL   CLS     */
   OUT"                          \n"); AT(24,1); GO
   OUT"                                    "); GO
   return(0);
 }
 if(functionkey== 'X' || functionkey== 'x') {    /* send command to add zero offset */
   OUT"output 10;*00%1dSA1%5lX\n",ampchan,0.0);
   ieeewt(buffer);
   autozero_read();
   return(0);
 }
 if(functionkey== 'I' || functionkey== 'i')  {  /* Increments gain of selected amplifier */
   read_gain();                    /* read current gain setting */
   if(gaincode!='Q')    {
    (char)gaincode=gaincode + 1;
    OUT"output 10;*00%1dSG%c\n",ampchan,gaincode);
    ieeewt(buffer);
   }
   read_gain();             /* need to update global variable gain */
   /*  Otherwise, amplifier set at maximum gain and/or do nothing */
   /* AT(13,1); OUT"Amplifier set to maximum gain.\n"); GO */
                       /* Find attenuations from gain settings */
   atten[ampchan]=10000./gain[ampchan];
   return(0);
 }
 if(functionkey== 'D' || functionkey== 'd') {    /* Decrements gain of selected amplifier */
   read_gain();                    /* read current gain setting */
   (char)gaincode=gaincode - 1;
   /* Error checking not necessary since gain default setting is E=1.00 */
   OUT"output 10;*00%1dSG%c\n",ampchan,gaincode);
   ieeewt(buffer);
   read_gain();             /* need to update global variable gain */
                       /* Find attenuations from gain settings */
   atten[ampchan]=10000./gain[ampchan];
   return(0);
 }
 if(functionkey== 'N' || functionkey== 'n') {
      /* Connect or Disconnect input from programmable amplifiers and
        update datacode */
   OUT"Connect.....C      Disconnect.....D   >>>"); AT(25,1); GO
   gch=getchar();    FL
   if(gch=='C'|| gch=='c')    {
     OUT"output 10;*FFFIC\n");
     ieeewt(buffer);
   }
   if(gch=='D' || gch=='d')    {
     OUT"output 10;*FFFID\n");
     ieeewt(buffer);
   }
   OUT"                                    ");   AT(25,1); GO
   return(0);
 }
 if (functionkey==0)         /* Must be called twice for function keys */
   functionkey=getch();         /* If Fkey, read keycode from keyboard */
 if (functionkey==ESC) {
   startover();         /* Return to main menu/reinitialize variables */
   return(0);
```

```
 }
 if (functionkey==F1)  {
   help();                        /* On-line user information */
   return(0);
 }
 if (functionkey==F4) {
   gnormalize();                   /* Gain Normalization option */
   return(0);
 }
 if (functionkey==F5) {
   caldatacode();                  /* User changes cal data code */
   return(0);
 }
 if (keyswork==2||keyswork==3)   {     /* If numerical data being taken...*/
   if (functionkey==F2)   {
    if (aveorimed==1)         /* Toggle to averaged or immediate data */
     aveorimed=0;
    else
     aveorimed=1;
    return(0);
   }
   if (functionkey==F6)   {
    setsen();                 /* User input channel sensitivities */
    return(0);
   }
   if (functionkey==F7)   {
    taredata();                   /* Manipulate tare data */
    return(0);
   }
   if (functionkey==F8)   {
    filestore();            /* User input filename for data storage */
    return(0);
   }
   if (functionkey==F9)   {
    aoa();                 /* Continuous angle of attack display */
    return(0);
   }
   if (functionkey==F12)   {
    printfile();                   /* Print formatted data file */
    return(0);
   }
 }
 if (keyswork && functionkey==F10)  {    /* keyswork= data or calibration */
   takedata();             /* Take a data point or calibration point */
   return(0);
 }
 if (keyswork && functionkey==F3)   {    /* keyswork= data or calibration */
   ptnum=0;               /* Reset all point counters, clear screen */
   tarecount=0;
   windoncount=0;
   calpoint_d=0;
   calpoint_s=0;
   calpoint_z=0;
   calcode=10;      /* default to this starting value for cal zero data */
   CLS
 }
}

leastsq(ndata,tempind,tempdep)
float   *tempind,*tempdep;
int     ndata;
{                              /* Computes an unweighted linear least squares fit */
 float  ta,sumyon,est;
 float  sumerr=0.0;
 float  sumx=0.0;
 float  sumy=0.0;
 float  st2=0.0;
 lscoffa=0.0;
 lscoffb=0.0;
 lsstdev=0.0;
```

```
 for (i=1;i<=ndata;i++) {
   sumy += tempind[i];
   sumx += tempdep[i];
 }
 sumyon = sumy/(float)ndata;
 for (i=1;i<=ndata;i++) {
   ta = tempind[i] - sumyon;
   st2 += ta * ta;
   lscoffa += ta * tempdep[i];
 }
 if(lscoffa==0) {
   OUT"Division by zero Error. Hit Enter to return..."); GO
   getchar();   FL   CLS
   return(0);
 }
 lscoffa = st2 / lscoffa;                 /* sensitivity or slope */
 lscoffb = (sumy - sumx * lscoffa ) / (float)ndata; /* constant or offset */
 for (i=1;i<=ndata;i++) {
   est = tempind[i]-(lscoffa*tempdep[i]+lscoffb);
   sumerr += est * est;
 }
 lsstdev = (float)sqrt(sumerr / (float)ndata);  /* standard error of est. */
}

newx (xcoord)
float xcoord;                  /* number to plot on x-axis */
{                        /* Scale x coordinates to screen size and plot frame */
 float tempx;                /* x coordinate for input number */
 tempx = ((float)maxx)/highx;     /* scaling factor to fit to screen */
 tempx = xcoord * tempx +XORG +.5;  /* # to plot*scale+offset and round */
 return((int)tempx);      /* Convert to integer for pixel coordinates */
}

newy (ycoord)
float ycoord;          /* number to plot on y-axis */
{                      /* Scale y coordinates to screen size and plot frame */
 float tempy;        /* y coordinate for input number */
 tempy = ((float) maxy)/HIGHY;     /* scaling factor to fit to screen */
 tempy = YORG-(ycoord * tempy +.5); /* # * scale+inversion and round */
 return( (int)tempy);   /* Convert to integer for pixel coordinates */
}

output()
{                     /* Output numerical program data to screen */
 int column;              /* Torque/Secondary data column placement */
 float dvz;                           /* Voltage difference */
 if(board_addr==0x300)                     /* If first board... */
   column=46;                     /* Print data in Torque column */
 else                                 /* If second board... */
   column=63;                    /* Print data in Secondary column */
            /* Output all computed data to screen in proper location */
 if (b==0)      {         /* Displacement data comes from board0 only */
   DATA OUT"%7.4f",freq); AT(1,66); GO
   OUT"%8.5f ",cdisp[0]); AT(5,13); GO
   OUT"%10.7f",devpeak[b][1]); AT(6,13); GO
   OUT"%8.5f",cdisp[1]); AT(8,13); GO
   OUT"%10.7f",devpeak[b][1]); AT(9,13); GO
   AT(10,13);
   if (aveorimed)    { OUT"%10.7f",devavepeak2); GO }
   else    { HIDE GO }
   time(&testtime);
   TEXT
   if(ampchan==0)
     DATA
   OUT"%5.1f",gain[0]); AT(26,12); GO
   OUT"%8.7f",offset[0]); AT(27,10);  GO
   TEXT
   if(ampchan==1)
     DATA
   OUT"%5.1f",gain[1]); AT(26,27); GO
```

```
    OUT"%8.7f",offset[1]); AT(27,25);  GO
    TEXT
    if(ampchan==2)
      DATA
    OUT"%5.1f",gain[2]); AT(26,42); GO
    OUT"%8.7f",offset[2]); AT(27,40);  GO
    TEXT
/*   OUT"Data code: %d",datacode); AT(27,60); GO */
    OUT"%5.2f",AoAval); AT(28,63); GO
    OUT"%5.2f",devaoa); AT(29,67); GO
  }
  /* Output Torque and Secondary Data */
  DATA OUT"%9.5f",real[b]); AT(5,column); GO
  OUT"%11.7f",devreal[b]); AT(6,column); GO
  OUT"%9.5f",imag[b]); AT(8,column); GO
  OUT"%11.7f",devimag[b]); AT(9,column); GO
  OUT"%9.5f",cforce[b]); AT(11,column); GO
  OUT"%11.7f",devpeak[b][0]); AT(12,column); GO
  AT(13,column);
  if (aveorimed)
    { OUT"%11.7f",devavepeak0); GO }
  else
    { HIDE GO }
  if (aveorimed) {
    DATA OUT"%11.3f",phdifsave);
    AT(15,column); GO
  }
  else {
    DATA OUT"%11.3f",phdif);
    AT(15,column); GO
  }
  OUT"%11.7f",stdev); AT(17,column); GO
  if(b==1) {
    getdatacode();       /* if datacode changes we want to know about it */
    if(datacode==3 || datacode==7 || datacode==9)   {
     /* Perform run data calculations for datacodes 3, 7, and 9 */
     voltage[0]=peak[2];                      /* Store voltages,  disp r */
     voltage[1]=real[0];                        /* torque real */
     voltage[2]=real[1];                      /* secondary real */
     voltage[4]=imag[0];                     /* torque imaginary */
     voltage[5]=imag[1];                  /* secondary imaginary */
     if (datacode==9) {               /* Calculates sensitivities when datacode = 9 */
      voltage[1]=cforce[0];                       /* torque real */
      voltage[2]=cforce[1];                    /* secondary real */
      for(i=0;i<3;i++)     {
        dvz=(voltage[i]-zero[i])* atten[i];
        if(dvz==0.) { dvz=1.; }
        sen[i]=calib[i]/dvz;
        sen[i+3]=sen[i];          /* Copy sensitivities to imaginary */
      }
      TEXT   OUT"Sensitivities"); AT(19,30); GO
      DATA   AT(20,10);  OUT"%8.7f",sen[0]); GO
      AT(20,30);   OUT"%8.5f",sen[1]); GO
      AT(20,50);   OUT"%8.5f",sen[2]); GO
     }              /* end of datacode = 9 */
     if (datacode==3 || datacode==7)    {            /* take readings */
      reading[0]=sen[0] * (voltage[0]-zero[0]) * atten[0];
      reading[1]=sen[1] * (voltage[1]-zero[1]) * atten[1];
      reading[2]=sen[2] * (voltage[2]-zero[2]) * atten[2];
      reading[4]=sen[4] * (voltage[4]-zero[4]) * atten[1];
      reading[5]=sen[5] * (voltage[5]-zero[5]) * atten[2];
      if(reading[0]==0) { reading[0]=1.; }
      if(reading[1]==0) { reading[1]=1.; }
      if(omega==0) { omega=1.; }
      t1=reading[1]/reading[0];
      t2=reading[4]/ (omega * reading[0]);
      m1=reading[2]/reading[0];
      m2=reading[5]/(omega * reading[0]);
     }   /* end of 3 or 7 */
     if (datacode==3)     {      /* Tare calculations when datacode = 3 */
```

```
     TEXT OUT"Disp       T2         T1         M2         M1");  AT(19,7); GO
     DATA  AT(20,3);  OUT"%10.7f",reading[0]); GO
     AT(20,18); OUT"%10.7f",t2); GO
     AT(20,33); OUT"%10.7f",t1); GO
     AT(20,48); OUT"%10.7f",m2); GO
     AT(20,63); OUT"%10.7f",m1); GO
    } /* end of datacode 3 block */
    if (datacode==7)     {          /* Datacode = 7  Wind-on data */
     t1=t1-(kspring-inertia*omega*omega);
     t2=t2-ctare;
     m1=m1-(amr - bmr * omega*omega);
     m2=m2-mjwdtare;
     TEXT
     if (woplotcode == 1 || woplotcode == 2)   {  /* Wind-on plot code */
       OUT"Disp      cmq+cma*     cma-kkcm*   cnq+cna*    cna-kkcnq*\n"); AT(19,4); GO
       t1*=c2;                 /* Scale dimensional quantities */
       t2*=c4;
       m1*=c1;
       m2*=c3;
     }
     else if(woplotcode == 3 || woplotcode == 4)  {       /* Wind-on plot code */
       OUT"Disp     cnr-cosacnb*   cnbcosa   clr-cosaclb*   clbcosa\n");  AT(19,4); GO
       t1*=-c2;                       /* modify parameters */
       t2*=c4;
       m1*=-c2;
       m2*=c4;
     }
     else if(woplotcode > 4)   {
       OUT"Disp     clp+sinaclb*   clbsina   cnp+sinacnb*   cnbsina\n");  AT(19,4); GO
       t1*=c2;                        /* modify parameters */
       t2*=c4;
       m1*=c2;
       m2*=c4;
     }
     DATA  AT(20,1);  OUT"%10.7f",reading[0]); GO
     AT(20,16); OUT"%10.7f",t2); GO
     AT(20,31); OUT"%10.7f",t1); GO
     AT(20,46); OUT"%10.7f",m2); GO
     AT(20,61); OUT"%10.7f",m1); GO
    } /* End datacode 7 block */
   } /* end of 3-7-9 */
   else                    /* any data code other than 3, 7 or 9 */
   {         /* Clear Coefficient Display Area   */
    OUT"                                                    ");  AT(19,1); GO
    OUT"                                                    ");  AT(20,1); GO
   }
 TEXT
 } /* End b=1 block */
} /* End Output block */

parameters()
{ /* Calculates constants needed for reduction of data. All formulas from BASIC program */
 Q = .7 * mach*mach * 144. * pressure * pow((1.+ .2 * mach*mach),-3.5);
 statemp = (temperature + 459.6) * 1./(1. + .2 * mach*mach);
 velocity = mach * 49.01 * sqrt(statemp);
 viscosity = 2.27 * (pow(statemp,1.5) / (statemp + 198.6)) * 1e-08;
 if (mach==0.) { reynolds = 0.; }
 else { reynolds = .000001 * 2. * Q / (viscosity * velocity); }
 if(refarea==0.)   { refarea=0.0001; }                    /* Calculate coefficients */
 if(reflength==0.) { reflength=0.0001; }
 if(Q==0.) { Q=0.0001; }
 c1 = -1./(Q*refarea);
 c2 = c1/reflength;
 c3 = c2 * 2. * velocity;
 c4 = c3/reflength;
}

plot()
{                      /* Output FFT spectra to screen */
 long count;               /* 2*number of floats expected from DSP */
```

```
 long onefl=2;             /* Upload one float to check if new data */
 int locx,locy;               /* Coordinates of pixel to be plotted */
 float prev=0;                          /* Previous data value */
 keyswork=0;             /* keycommand() uses keyswork=0 for plot() */
 if(c==2) { dsp_board_addr (0x320); }        /* Set board address to 2nd board */
 else {    dsp_board_addr(0x300);   }      /* Set board address to 1st board */
 count=each*4;                   /* Set number of values to upload */
 AT(14,33);   _outtext("Generating plot...");         /* Tell user to wait for plot */
 /* Execute plot loop until keyboard is hit, then return
   control to main() and interpret user command using keycommand() */
 while(!kbhit())  {                             /* Upload data from DSP board */
   dsp32c_up_array(addr_plot[c],onefl,(int *)fft);
   if (prev != fft[0])  {   /* If data is new.../* Upload FFT array from DSP program */
    dsp32c_up_array(addr_plot[c],count,(int*)fft);
    prev = fft[0];              /* "previous"=new value, Find db V for all FFT values, DC
values from FFT are twice the actual value */
    pnt[0]=(20*log10(fabs(fft[0])))-SCALE;
/* dB=20*log10(sqrt(sum squares of real and imag)/(# of FFT points*window scaling factor) */
    for(i=1;i<each;i++)
       pnt[i]=10*log10(fft[2*i]*fft[2*i]+fft[2*i+1]*fft[2*i+1])-SCALE;
    CLS                        /* Draw outline/grid and scale graph */
    endx=newx((float)each);    /* endx=pixel coordinate of last x value */
    _setcliprgn(1,1,endx,ENDY);     /* Prevent graphing outside border */
    _rectangle (_GBORDER, XORG,YORG,endx,ENDY);        /* draw border */
    scale();              /* Draw and label scales on x and y axis */
    AT (1,24);                          /* Output title of plot */
    if (c==0)  _outtext("***** Secondary Channel FFT *****");
    else if (c==1)  _outtext("*** Displacement Channel FFT ***");
    else      _outtext("***** Torque Channel FFT *****");
    AT (3,32);  _outtext("Frequency (Hz)");        /* Output x axis title */
    for(i=0;i<15;i++)    {                 /* Output y axis title */
     AT (i+9,1);   OUT"%c",ytitle[i]); GO
    }             /* Tell operator what type of program is generating plot */
    if(p==-9)      OUT"Flattop window @250");
    else if(p==-7)      OUT"Hanning window @125");
    else      OUT"Hanning window @250");
    AT(30,32); GO
    _moveto (XORG,YORG);                      /* Move to origin */
    for(i=0;i<each;i++)     {                /* Draw spectrum plot */
      locx = newx((float)i);       /* Calculate scaled integer pixel..*/
     locy = newy(pnt[i]);           /* ...value for xy coordinates */
     _lineto (locx,locy);          /* Draw line to next coordinates */
    }
   }
 }
}

ppcpk()
{              /* Calculate coefficients
   These aren't currently used anywhere, but are also calculated in
   the DSP program dynstat */
 float phidif=phdif*PI/180.;      /* Phase angle difference in radians */
 if (peak[2] != 0 && freq != 0)  { /* If not dividing by zero...*/
   ppc=peak[0]*sin(phidif)/(2*PI*freq*peak[2]); /* (f*sin(eta)/freq*d) */
   pk=peak[0]*cos(phidif)/peak[2];          /* (f*cos(eta)/d) */
 }
 else {                    /* Else take corrective action */
   ppc=0;
   pk=0;
 }
}

printfile()
{                    /* Print data to laserjet during execution of C program */
 char ch;
 char string[40];                /* DOS command */
 CLS
 OUT"Type 'c' for calibration data file or 'd' tare/wind-on data file.");
 AT(1,5); GO
 OUT"\n\nElse type 'r' to return to main menu."); GO
```

```
 ch=getchar(); FL
 if(ch=='d' || ch=='D') {
   OUT"System copying tare/wind-on data file to printer.\n\n");   GO
   sprintf(string,"print %s",format);  /* Write DOS command string */
   system(string);                 /* DOS system call */
   OUT"Press ENTER to return to menu."); AT(14,1); GO /* User sees results */
   ch=getchar(); FL
 }
 if(ch=='c' || ch=='C') {
   OUT"System copying calibration data file to printer.\n\n");   GO
   sprintf(string,"print %s",cformat);  /* Write DOS command string */
   system(string);                 /* DOS system call */
   OUT"Press ENTER to return to menu."); AT(14,1); GO /* User sees results */
   ch=getchar(); FL
 }
 CLS
}
```

Appendix F

Fourth Source File Listing:

```
/* file xrs.c */
/* Contains functions whose names begin with r and s. */
#include "dsphead.h"
readaoa()
{        /* This function reads the Kearfott output voltage and calculates
             the AoA of the model using constants entered by the operator.*/
 float A1, A2;     /*intermediate calculations for AoA*/
 mode = 3;         /*single analog to digital read*/
 data[0] = 3;       /*set gain/range selection to ±5V*/
 data[1] = 3;       /*set input channel to 3*/
 value = 1;
 while(value < 6) {
   switch(value)    {
    case 1:
    case 2:
    case 3:
    case 4:
     if ((error = das20(mode,data)) !=0) /*call DAS-20*/
       value++;
     else
       value = 7;
    break;
    case 5:
     fi = fopen("DAS20err","a");
     fprintf(fi, "\nDAS-20 analog read error number ");
     fprintf(fi, "%d\n", error);
     fclose(fi);
     value++;
    break;
   }
 }
 /*convert reading to volts*/
 aoavolts = (float)data[0] * 10.0 / 4096;
 A1 = alphasens * (aoavolts - alphabias);
 if(A1>1.0)
   A1=1.0;
 A2 = (180.0 / 3.1415926) * asin(A1);
 AoAval = A2 - alphaoff - alphadelt;
 if(WINDON)
   AoAval=AoAval+flow_angle;
 if (inverted)
   AoAval = -AoAval;
}

read_gain()
{       /* Places into a global variable selected amplifier ampchan's current gain
          setting. A zero gain means that the IEEE scanf returned an invalid gain code. */
 OUT"output 10;*00%1dGS\n",ampchan);
 ieeewt(buffer);
 ieeewt("enter 10\n"); /* reads value from IEEE device 10 */
 ieeescnf("%s",buffer);
 gaincode=buffer[6];
 switch ( gaincode ) {
   case 'E' : gain[ampchan]= 1.0; break;
   case 'F' : gain[ampchan]= 2.0; break;
   case 'G' : gain[ampchan]= 5.0; break;
   case 'H' : gain[ampchan]= 10.0; break;
   case 'I' : gain[ampchan]= 20.0; break;
   case 'J' : gain[ampchan]= 50.0; break;
   case 'K' : gain[ampchan]= 100.; break;
   case 'L' : gain[ampchan]= 200.; break;
   case 'M' : gain[ampchan]= 500.; break;
   case 'N' : gain[ampchan]= 1000.; break;
   case 'O' : gain[ampchan]= 2000.; break;
```

```
   case 'P' : gain[ampchan]= 5000.; break;
   case 'Q' : gain[ampchan]=10000.; break;
   default  : gain[ampchan]=1.0;
 }
}

realimag()
{              /* Compute real and imaginary components with respect to displacement */
 float phidif=( -1. * phdif)*PI/180.; /* Phase angle difference in radians */
 real[b]=cos(phidif)*peak[0];     /* find real component of channel */
 imag[b]=sin(phidif)*peak[0];     /* find imag component of channel */
 imag[b]= (-1.*imag[b]);       /* sign correction ??? */
 findaverage(real[b],avreal[b]);   /* average and find st. dev for real */
 realsave[b]=finalave;          /* save average */
 devreal[b]=deviation;          /* save deviation */
 findaverage(imag[b],avimag[b]);   /* average and find st. dev for imag */
 imagsave[b]=finalave;          /* save average */
 devimag[b]=deviation;          /* save deviation */
}

refresh()
{           /* Set up screen and framework for numerical data */
 AT(1,7);  TEXT           /* Tell user which program is running */
 if(p==1) _outtext("Enhanced FFT");
 if(p==2) _outtext("Stand alone FFT at 250Hz");
 if(p==3) _outtext("Stand alone FFT at 125Hz");
 if(p==4) _outtext("Synchronous Demodulation 0.25 Hz filter");
 if(p==5) _outtext("Synchronous Demodulation 0.10 Hz filter");
 /* Output lables of framework for numerical data */
 OUT"Displacement"); AT(3,13); GO
 OUT"Frequency"); AT(1,53); GO
 OUT"Real (1)"); AT(5,1); GO
 OUT"St. Dev."); AT(6,1); GO
 OUT"Real (2)"); AT(8,1); GO
 OUT"St. Dev."); AT(9,1); GO
 OUT"Real"); AT(5,30); GO
 OUT"St. Dev."); AT(6,30); GO
 OUT"Imaginary"); AT(8,30); GO
 OUT"St. Dev."); AT(9,30); GO
 OUT"Magnitude"); AT(11,30); GO
 OUT"St. Dev."); AT(12,30); GO
 OUT"Secondary"); AT(3,65); GO
 OUT" Torque"); AT(3,47); GO
 OUT"Phase angle"); AT(15,18); GO
 OUT"(Displacement-Channel)"); AT(16,18); GO
 OUT"St. Dev."); AT(17,18); GO
 OUT"Gains: "); AT(26,1); GO
 OUT"Offsets: "); AT(27,1); GO
 OUT"Temp: %5.2f °F, Pressure: %5.2f psi",temperature,pressure); AT(28,1); GO
 OUT"Ptnum = %d",ptnum+1); AT (28,40); GO
 OUT"Mach: %3.2f, Q: %5.4f psi, V: %5.4f ft/sec",mach,Q,velocity); AT(29,1); GO
 OUT"AoA:"); AT(28,58); GO
 OUT"St. Dev.:"); AT(29,58); GO
 OUT"Press F1 for help"); AT(30,60); GO
 TEXT OUT"Averaging %3d,",NAVE); AT(30,1); GO
 if(aveorimed) {   OUT"StDev Avgs"); AT(10,1); GO
   OUT"St. Dev. Avgs."); AT(13,30); GO
   OUT"Averaged data "); AT(30,16); GO
 }
 else {   OUT"         "); AT(10,1); GO
   OUT"Immediate data"); AT(30,16); GO
   OUT"         "); AT(13,30); GO
 }
 OUT"Stored in: %s",file); AT(30,34); GO
}

scale()
{      /* Scales x and y axis of graph for FFT plots, draws gridlines and lables */
 int locy,locx;    /* xy pixel coordinates */
 int s;         /* Differentiates between plots @250Hz and @125Hz */
```

```
 if (each==250) e=0;    /* Set x-scale for correct # of points */
 else e=1;
 if (p==-7) s=0;       /* Set x-scale for correct sampling frequency */
 else s=1;
 _setlinestyle(0x707);          /* Set gridlines mask on line style */
 for (i=1;i<5;i++)  {          /* Mark standard y gridlines */
   locy=newy(i*-20.);
   _moveto(XORG,locy);
   _lineto(endx,locy);
 }
 for (i=1;i<5;i++)   {         /* Mark standard x gridlines */
    locx=newx(i*(float)each/5.);
   _moveto (locx,YORG);
   _lineto (locx,ENDY);
 }
 for (i=0;i<11;i++) {           /* Output xy scale numbers */
   AT (textpr[i],textpc[i]);
   OUT"%5.1f",div[s][e][i]);
   _outtext(buffer);
 }
 _setlinestyle(0xffff);         /* Reset line-mask to solid line */
}

setnew()
{           /* Allows user to configure program. p (program) is the controlling
            variable passed to the main program and DSP initializing functions.
            p is defined as follows: If p=1 thru 5 then p has been input through
            choose() otherwise, p is set in setnew().
   p=1 for Enhanced FFT
   p=2 for Stand Alone FFT @250
   p=3 for Stand Alone FFT @125
   p=4 for Synchronous Demodulation, 0.25 Hz filter
   p=5 for Synchronous Demodulation, 0.10 Hz filter
   p=9 for Calibration
   for the following plots load DSP program corresponding to p+10
     (see above for these programs)
   p=-9 for plotting with flattop window
   p=-8 for plotting with Hanning window @250
   p=-7 for plotting with Hanning window @125  */
 char jc;               /* input character */
 char prog[30],type[30],chan[15],doit[11];     /* hold menu information */
 int j=0;                      /* User input, # of menu selection */
 beentoaoa=0;                     /* Do not call setnew() at end */
 while(!(j==1||j==2||j==11))               /* While no run command */
 {                               /* Print main menu to screen */
   if (p<0)       p+=10;     /* Change p back from plotting to programs */
   if(p==1)    {  /* Prepare to output program and plot type on menu screen for review */
    sprintf(prog,"Enhanced FFT ");
    sprintf(type,"Flattop @250");
   }
   else if(p==2)   {
    sprintf(prog,"Stand Alone FFT @250");
    sprintf(type,"Hanning @250");
   }
   else if(p==3)   {
    sprintf(prog,"Stand Alone FFT @125");
    sprintf(type,"Hanning @125");
   }
   else if(p==4)   {
    sprintf(prog,"Synch. Demod., 0.25 Hz filter");
    sprintf(type,"Hanning @250");
   }
   else if(p==5)   {
    sprintf(prog,"Synch. Demod., 0.10 Hz filter");
    sprintf(type,"Hanning @250");
   }
   else   {
    sprintf(prog,"Calibrate");
    sprintf(type,"Hanning @250");
   }
```

```
if(c==0) sprintf(chan,"Secondary");
else if(c==1) sprintf(chan,"Displacement");
else sprintf(chan,"Torque");
if(aveorimed) sprintf(doit,"Averaged");
else sprintf(doit,"Immediate");
        /* Print menu and current configuration settings to screen */
TEXT OUT"Main Menu"); AT(1,35); GO
OUT"1. Run program"); AT(2,3); GO
OUT"2. Plot FFT to screen"); AT(4,3); GO
OUT"3. Change program"); AT(6,3); GO
OUT"Current program: %s",prog); AT(6,35); GO
OUT"4. Change plot configuration"); AT(8,3); GO
OUT"%s, %s, %d",type,chan,each); AT(8,35); GO
OUT"5. Averaging"); AT(10,3); GO
OUT"%s, %d points",doit,NAVE); AT(10,35); GO
if (!u6) DATA
OUT"6. Set calibration constants"); AT(12,3); GO TEXT
if (!u7) DATA
OUT"7. Input AoA accelerometer constants and reference dimensions");
AT (14,3); GO TEXT
if (!u8) DATA
OUT"8. Record test numbers and conditions"); AT(16,3); GO TEXT
if (!u9) DATA
OUT"9. Change filename for Dynamic Stability Data Storage"); AT(18,3); GO
TEXT
OUT"10. Plot Tare or Wind-on axes on X-Y plotter"); AT(20,3); GO
OUT"11. Angle of Attack continuous display"); AT(22,3); GO
OUT"12. Select data file for printing"); AT(24,3); GO
OUT"13. Reset point counter to 1"); AT(26,3); GO
OUT"0. Exit\n"); AT(28,3); GO
if(!(u6 && u7 && u8 && u9))
{
 OUT"Please update the items shown in ");
 AT(30,15); GO
 DATA OUT"white.");
 AT(30,48); GO TEXT
}
AT(29,3);                    /* Where user answer will appear */
scanf(" %d",&j);                     /* Accept menu selection */
FL   if (j==3)   {
 do        {       /* Lets the user choose which DSP algorithm to run */
  CLS     OUT"Download which program to DSP boards?\n\n\n"); GO
  OUT"Enter number:\n1. Enhanced FFT\n");     GO
  OUT"2. Stand alone FFT sampled @250Hz\n"); GO
  OUT"3. Stand alone FFT sampled @125Hz\n"); GO
  OUT"4. Synchronous Demodulation, 0.25 Hz filter\n"); GO
  OUT"5. Synchronous Demodulation, 0.10 Hz filter\n"); GO
  OUT"6. Calibration\n\n"); GO
  scanf(" %d", &p);      FL      /* Accept user choice */
 } while(p<1||p>6);         /* If input value not valid, try again */
if(p==6)     p=9;              /* calibration p code was 9 in old program */
}
else if(j==4)   {           /* Choose window to plot, how many points, and channel */
 do    {
  CLS     OUT"Choose a plot...\n\n"); GO
  OUT"1. Flattop window @250\n"); GO      /* Program 1. aka j==3 */
  OUT"2. Hanning window @250\n"); GO      /* Program 2. " */
  OUT"3. Hanning window @125\n\n"); GO     /* Program 3. " */
  scanf(" %d",&p); FL
 } while (p<1||p>3);       /* If user input not on menu, try again */
 OUT"How many points in FFT plot?\n"); GO
 OUT"Enter number:\n\n 1. 250\n 2. 500\n\n\n "); GO
 jc=getchar();    FL          /* Get user input */
 if (jc=='1')    {
  each=250;             /* # points to plot */
  highx=325;           /* Necessary scaling factor for x-axis */
 }
 else if (jc=='2')    {
  each=500;             /* # points to plot */
  highx=650;            /* Necessary scaling factor for x-axis */
```

```
 }               /* If user presses ENTER values do not change */
 OUT"Plot which channel?\n\n"); GO
 OUT"Enter:\n\n1. Displacement\n2. Torque\n3. Secondary\n\n"); GO
 jc=getchar();       FL           /* Get user choice */
 if (jc=='3')       c=0;             /* Index for DSP address of channel to plot */
 else if (jc=='1')       c=1;
 else if (jc=='2')       c=2;
/* If user enters another character or presses enter, c does not change */
 CLS                 /* Reset screen for plot */
}
else if (j==5)   {  /* Use averaged data? and set # points in average */
 do    {
  NAVE=2;                /* Initialize # of points in average */
  TEXT  CLS
  OUT"Averaging....\n\n"); GO
  OUT"Enter number:\n\n1. Use immediate data, no averaging\n"); GO
  OUT"2. Use averaged data \n");     GO
  jc=getchar(); FL           /* User input choice from menu above */
 } while (jc<'1' || jc>'2');
 if(jc=='1')     aveorimed=0;
 else if(jc=='2')     aveorimed=1;
 do    {
  CLS     OUT"How many points averaged for data & standard deviation?\n\n"); GO
  OUT"Enter a number from 2 to 120... "); GO
  scanf(" %d", &NAVE); FL       /* User input # of points in average */
 } while(NAVE<2||NAVE>120);         /* Accept a # between 2 and 120 */
}
else if(j==6)   {    /* Input calibration constants */
 CLS    OUT"Please input calibration constants.\n\n"); GO
 OUT" Enter p for value in parentheses.\n\n"); GO
 OUT"\nk displacement (%6.4f): ",calib[0]); GO
 scanf("%f",&calib[0]);       /* Read user input for new values */
 FL                   /* Accept default if letter is entered */
 OUT"\nk torque (%6.4f): ",calib[1]); GO
 scanf("%f",&calib[1]); FL
 OUT"\nk secondary (%6.4f): ",calib[2]); GO
 scanf("%f",&calib[2]); FL
 CLS                  /* Reset screen */
 for(i=0;i<3;i++) {  calib[i+3]=calib[i]; }    /* Copy real calibration const. to imag */
 u6=1;                  /* Turns white off in menu */
}
else if (j==7) { /* User input for aoa constants, see declarations for descriptions */
 CLS    AT(1,1); OUT"Enter p to accept value in parentheses\n\n\n\n"); GO
 OUT"Please input\n\n"); GO
 OUT" AoA Alpha sensitivity in g/volt (%10.6f): ",alphasens); GO
 scanf(" %f",&alphasens); FL
 OUT"\n AoA Alpha bias in volts (%10.6f): ",alphabias); GO
 scanf(" %f",&alphabias); FL
 OUT"\n AoA Alpha offset in degrees (%7.2f): ",alphaoff); GO
 scanf(" %f",&alphaoff); FL
 OUT"\n Delta Alpha -> sting to model in degrees (%5.2f): ",alphadelt); GO
 scanf(" %f",&alphadelt); FL
 OUT"\n Reference length - B in feet (%4.2f): ",reflength); GO
 scanf(" %f",&reflength); FL
 OUT"\n Reference area - S in feet^2 (%6.4f): ",refarea); GO
 scanf(" %f",&refarea); FL
 u7=1;                  /* Turns off the white on the menu */
}
else if (j==8)  {  testcond();  }            /* User input temperature,pressure,mach */
else if (j==9)  {  filestore(); }           /* user input filename for data storage */
else if (j==10) {  setup_plot(); }             /* Plot axes for tares or wind-on */
else if (j==11) {  aoa();        }         /* Angle of Attack continuous display */
else if (j==12) {  printfile();  }           /* Send formatted data file to printer */
else if (j==13) {    ptnum=0;                       /* Reset all point counters */
  windoncount=0;
  tarecount=0;
  calpoint_d=0;               /* Calibration data point counter */
  calpoint_s=0;               /* Calibration shunt point counter */
  calpoint_z=0;               /* Calibration zero point counter */
  calcode=10;
```

```
      CLS
    }
    else if (j==0)   {           /* Reset screen save all settings and terminate program */
     CLS     _setvideomode (_DEFAULTMODE);     /* Returns to normal video mode */
     if(ptnum>1)     {
      OUT"copy c:\\runtime\\%s c:\\data\\%s",format,format);
      system(buffer);
     }
     if(calpoint_d>1)     {
      OUT"copy c:\\runtime\\%s c:\\data\\%s",cformat,cformat);
      system(buffer);
     }
     printf("\nGoodbye...\n");    /* Open calibration file and save configuration settings */
     calival=fopen("c:\\data\\calival.dat","w");
     fprintf(calival,"%d\n%f\n",calibnum,momentarm);
     fclose(calival);                /* Open test file and save configuration settings */
     testval=fopen("c:\\data\\testval.dat","w");
     fprintf(testval,"%f\n%f\n%f\n%f\n%f\n%f\n%d\n%d\n%f\n%f\n%f\n%f\n%f\n%f\n",
       alphasens,alphabias,alphaoff,alphadelt,reflength,refarea,
       testnum,runnum,mach,pressure,temperature,sen[0],sen[1],sen[2]);
     fclose(testval);       /* Open plot and average configuration file and save settings */
     dconfig=fopen("c:\\data\\dconfig.dat","w");
     fprintf(dconfig,"%d\n%d\n%d\n%d\n%d\n%d\n",each,highx,c,p,aveorimed,NAVE);
     fclose (dconfig);       /* Open sensititivies config. file, save settings, close */
     sens=fopen("c:\\data\\sens.dat", "w");
     fprintf (sens,"%f\n%f\n%f\n%f\n%f\n%f\n%f\n%f\n",calib[0],calib[1],
      calib[2],afreq,anorm0[0],anorm0[1],anorm1[0],anorm1[1]);
     fclose(sens);
     tareval=fopen("c:\\data\\tareval.dat", "w");
     fprintf(tareval,"%f\n%f\n%f\n%f\n%f\n%f\n%f\n",kspring,inertia,ctare,
       amr,bmr,mjwdtare);
     fclose(tareval);
     exit (0);                 /* Closes all files and terminates */
    }                  /* If plot requested for heterodyne, default to FFT routine @250 */
    else if(j==2&&(p==4||p==5||p==9))  {  p=2; }
    CLS
  }                           /* Load correct DSP program to "program" variables */
  if(p==1) {
    sprintf(program,"fft");     /* FFT with freq from DSPNET board0 */
    sprintf(program2,"fft2");    /* Same program board1 */
  }
  else if(p==2)  {
    sprintf(program,"search");   /* FFT search at 250 */
    sprintf(program2,"search2");
  }
  else if(p==3)  {
    sprintf(program,"search12");  /* FFT search at 125 */
    sprintf(program2,"serch122");
  }
  else if(p==4)  {
    sprintf(program,"dynstat");   /* Heterodyne DSP 14 sec filters */
    sprintf(program2,"dynstat2");
  }
  else if(p==5)  {
    sprintf(program,"dynstat");  /* Heterodyne DSP long filters */
    sprintf(program2,"dynstat2");
  }
  else if(p==9)  {
    sprintf(program,"dynstat");  /* Calibration DC digital voltmeter */
    sprintf(program2,"dynstat2");
  }                  /* Download DSP programs and initialize boards */
 /* Initializes all functions of and downloads programs to DSP boards. Initialize both boards
 and set current board address to second board. Sets DSP parameters, exits if board not found.
 Initializes DSP board, refer to Ariel manual for more info */
  if (default_addr() ==0) {   exit (1);}
      /* Downloads DSP executable code terminates if DSP program not found */
  if (dsp_dl_exec(program)==0) {   exit(1); }
  dsp_run();                        /* Begins DSP code execution */
  if(p<4)   {                      /* If downloading an FFT program */
    addr_realf[0]=find_label_name((char *)"realf");  /* Find DSP addresses */
```

```
   addr_plot[0]=find_label_name((char *)"fdft0");     /* Secondary data */
   addr_plot[1]=find_label_name((char *)"ddft0");       /* Displ. data */
 }
 else        /* Calibration and Dynstat save upload values in "upload" */
   addr[0]=find_label_name((char *)"upload");
     /* Performs function of default_addr for second and following boards */
 if (find_device_type(0x320)==0) {  exit(2); }
 if (dsp_dl_exec(program2)==0)  {   exit(2); }      /* Downloads executable code */
 dsp_run();
 if (p<4)     {                  /* If downloading an FFT program */
   addr_realf[1]=find_label_name((char *)"realf");   /* Find DSP address */
   addr_plot[2]=find_label_name((char *)"fdft0");       /* Torque data */
 }
 else        /* Calibration and Dynstat save upload values in "upload" */
 addr[1]=find_label_name((char*)"upload");
 parameters();     /* Calculate Dynamic pressure, velocity, and reynolds */
 if (j==2) p-=10;        /* If plot, then change p to control plotting */
 CLS
 if (beentoaoa) { setnew(); }  /* If aoa has been called from setnew, return to menu */
}

setsen()
{      /* User input of channel sensitivities (normally calculated when datacode = 9) */
 CLS OUT"Please input the following sensitivities...\n\n"); GO
 OUT"Enter p for value in parentheses\n\n"); GO
 OUT"Displacement channel sensitivity (%f)... ",sen[0]); GO
 scanf(" %f",sen[0]); FL
 OUT"\nTorque channel sensitivity (%f)... ",sen[1]); GO
 scanf(" %f",sen[1]); FL
 OUT"\nSecondary channel sensitivity (%f)... ",sen[2]); GO
 scanf(" %f",sen[2]); FL
 CLS
}

setup_plot()
{                /* Sets up plotter for tare plots or wind on plots */
 int   nuchar;
 int   dex;
 int   start,incrm,stop;
 CLS  FL  AT(3,5);
 OUT"You can return to the main menu without plotting axes, but first you"); GO
 AT(4,5); OUT"must enter constants and scaling factors."); GO
 AT(7,5); OUT"Type 'w' or 'W' for wind-on, or 'T' or 't' for tare..."); GO
 nuchar=getchar(); FL AT(9,5);
 if(nuchar=='w' || nuchar=='W') {                     /* wind on plot setup here */
   CLS    OUT"Enter p to accept value in parentheses."); GO
   AT(3,15);   OUT"Please enter the following values..."); GO
   AT(5,5);    OUT"Kspring in ft-lb/rad (%f)... ",kspring); GO
   scanf(" %f",&kspring); FL
   AT(7,5);    OUT"\nInertia in slug-ft^2 (%f)... ",inertia); GO
   scanf(" %f",&inertia); FL
   AT(9,5);    OUT"\nCtare (%f)... ",ctare); GO
   scanf(" %f",&ctare); FL
   AT(11,5);   OUT"\nAMR (%f)... ",amr); GO
   scanf(" %f",&amr); FL
   AT(13,5);   OUT"\nBMR (%f)... ",bmr); GO
   scanf(" %f",&bmr); FL
   AT(15,5);   OUT"\n(MJ/WD)tare (%f)... ",mjwdtare); GO
   scanf(" %f",&mjwdtare); FL
   CLS                   /* Output tare constants to screen */
   OUT"Kspring = %9.5f ft-lb/rad   Inertia = %9.5f slug-ft^2",kspring,inertia); AT(17,1); GO
   OUT" Ctare = %9.5f            AMR = %9.5f",ctare,amr);   AT(18,1); GO
   OUT"   BMR = %9.5f       (mj/wd)tare = %9.5f",bmr,mjwdtare);   AT(19,1); GO
   AT(21,5);   OUT"Enter the wind-on plot code (%d)... ",woplotcode); GO
   scanf(" %d",&woplotcode); FL     /* Read in wind-on plot code */
   AT(22,5);   OUT"Enter the wind-on plot symbol code (%d)... ",wosymbolc);   GO
   scanf(" %d",&wosymbolc); FL
   AT(23,5);   OUT"\nEnter the x-scale factor (%f)... ",xscale); GO
   scanf(" %f",&xscale); FL
   AT(24,5);   OUT"\nEnter the upper y-scale factor (%f)... ",y1scale); GO
```

```
scanf(" %f",&y1scale); FL
AT(25,5);   OUT"\nEnter the lower y-scale factor (%f)... ",y2scale); GO
scanf(" %f",&y2scale); FL
AT(26,5);   OUT"\nEnter the minimum x value (%f)... ",xmin); GO
scanf(" %f",&xmin); FL
AT(27,5);   OUT"\nEnter the maximum x value (%f)... ",xmax); GO
scanf(" %f",&xmax); FL
CLS   AT(3,5);   OUT"Turn on X-Y plotter, load paper. Press ENTER to continue."); GO
AT(5,5);   OUT"Else hit 'R' or 'r' to return without plotting axes.\n"); GO
nuchar = getchar();
FL   if(nuchar=='R' || nuchar=='r') {  return(0); }
AT(15,10);   OUT"Now plotting wind-on axes..."); GO
OUT"DT%c","\003');                     /* initialize plotter */
OUT_XY buffer);
OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",100*xscale*xmin,100*xscale*xmax,
 -800*y1scale,200*y1scale);
OUT_XY buffer);             /* plot the upper x-axis */
OUT_XY"SP1;");
OUT_XY"PAPU 0,0;");
start=(int)((xmin+1)*100*xscale);
stop=(int)(xmax*100*xscale);
incrm=(int)(100*xscale);
for(dex=start; dex<stop; dex+=incrm)    {
 OUT "PAPD %10.2f,0;XT;",(float)dex);
 OUT_XY buffer);
}
OUT_XY"PU;");   /* plot the upper y-axis */
OUT_XY"PAPU 0,0;");
start=(int)(-200*y1scale);
stop=(int)((100*y1scale)+1);
incrm=(int)(100*y1scale);
for(dex=start; dex<stop; dex+=incrm)   {
 OUT "PAPD 0,%10.0f;YT;",(float)dex);
 OUT_XY buffer);
}
OUT_XY"PU;");   /* label upper x and y axes */
OUT_XY"SI,.19,.27;");
start=(int)((xmin+1)*100*xscale);
stop=(int)(xmax*100*xscale);
incrm=(int)(100*xscale);
for(dex=start; dex<stop; dex+=incrm)    {
 if(dex!=0)    {
  OUT "PAPU %10.3f,%10.3f;",(float)dex-(15*xscale),-40*y1scale);   OUT_XY buffer);
  OUT "LB%2d%c;PU;",dex/100,'\003');    OUT_XY buffer);
 }
}
OUT_XY"PAPU 0,0;");
start=(int)(-200*y1scale);
stop=(int)((100*y1scale)+1);
incrm=(int)(100*y1scale);
for(dex=start; dex<stop; dex+=incrm)   {
 if(dex!=0)    {
  OUT "PA %10.3f,%10.3f;",-75*xscale,(float)dex-(5*y1scale));     OUT_XY buffer);
  OUT "LB%3.1f%c;PU;",(float)dex/100.0,'\003');     OUT_XY buffer);
 }
}
OUT_XY"PAPU 0,0;");  /* upper labels */
OUT_XY"PAPU %10.3f,%10.3f;",50*xscale,150*y1scale);
if(woplotcode<3)   {
 OUT "LBCMQ+CMA*, /RAD     TEST      RUN%c;PU;","\003');    OUT_XY buffer);
}
if(woplotcode==3 || woplotcode==4)   {
 OUT "LBCNR-COSACNB*, /RAD  TEST      RUN%c;PU;","\003');    OUT_XY buffer);
}
if(woplotcode>4)   {
 OUT "LBCLP+SINACLB*, /RAD  TEST      RUN%c;PU;","\003');    OUT_XY buffer);
}
OUT_XY"PAPU %10.3f,%10.3f;",330*xscale,150*y1scale);
OUT "LB%d%c;PU;",testnum,'\003');   OUT_XY buffer);
OUT_XY"PAPU %10.3f,%10.3f;",470*xscale,150*y1scale);
```

```
  OUT "LB%d%c;PU;",runnum,'\003');   OUT_XY buffer);
/* plot the lower x-axis */
  OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",100*xscale*xmin,100*xscale*xmax,
    -300*y2scale,700*y2scale);   OUT_XY buffer);
  OUT_XY"PAPU 0,0;");
  start=(int)((xmin+1)*100*xscale);
  stop=(int)(xmax*100*xscale);
  incrm=(int)(100*xscale);
  for(dex=start; dex<stop; dex+=incrm)   {
   OUT "PAPD %10.0f,0;XT;",(float)dex);    OUT_XY buffer);
  }
  OUT_XY"PU;");
  OUT_XY"PAPU 0,0;");  /* plot the lower y-axis */
  start=(int)(-200*y2scale);
  stop=(int)((100*y2scale)+1);
  incrm=(int)(100*y2scale);
  for(dex=start; dex<stop; dex+=incrm)   {
   OUT "PAPD 0,%10.0f;YT;",(float)dex);    OUT_XY buffer);
  }
  OUT_XY"PU;");   /* label lower x and y axes */
  start=(int)((xmin+1)*100*xscale);
  stop=(int)(xmax*100*xscale);
  incrm=(int)(100*xscale);
  for(dex=start; dex<stop; dex+=incrm)   {
   if(dex!=0)    {
    OUT "PAPU %10.3f,%10.3f;",(float)dex-(15*xscale),-40*y2scale);     OUT_XY buffer);
    OUT "LB%2d%c;PU;",dex/100,'\003');     OUT_XY buffer);
   }
  }
  OUT_XY"PAPU 0,0;");
  start=(int)(-200*y2scale);
  stop=(int)((100*y2scale)+1);
  incrm=(int)(100*y2scale);
  for(dex=start; dex<stop; dex+=incrm)   {
   if(dex!=0)    {
    OUT "PA %10.3f,%10.3f;",-75*xscale,(float)dex-(5*y2scale));     OUT_XY buffer);
    OUT "LB%3.1f%c;PU;",(float)dex/100.0,'\003');     OUT_XY buffer);
   }
  }
  OUT_XY"PAPU 0,0;");   /* lower labels */
  OUT_XY"PAPU %10.3f,%10.3f;",50*xscale,150*y2scale);
  if(woplotcode==1)   {
   OUT "LBCMA-KKCMQ*, /RAD%c;PU;",'\003');    OUT_XY buffer);
  }
  if(woplotcode==2)   {
   OUT "LBCNQ+CNA*, /RAD%c;PU;",'\003');    OUT_XY buffer);
  }
  if(woplotcode==3)   {
   OUT "LBCNBCOSA+KKCNR*, /RAD%c;PU;",'\003');    OUT_XY buffer);
  }
  if(woplotcode==4)   {
   OUT "LBCLR-COSACLB*, /RAD%c;PU;",'\003');    OUT_XY buffer);
  }
  if(woplotcode==5)   {
   OUT "LBCLBSINA-KKCLP*, /RAD%c;PU;",'\003');    OUT_XY buffer);
  }
  if(woplotcode==6)   {
   OUT "LBCNP+SINACNB*, /RAD%c;PU;",'\003');    OUT_XY buffer);
  }
  OUT_XY"PAPU %10.3f,%10.3f;",150*xscale,-250*y2scale);
  OUT "LBANGLE OF ATTACK, DEGREES%c;PU;",'\003');   OUT_XY buffer);
  OUT_XY"PAPU 0,0;");
} /* end of wind-on set-up */
else if(nuchar=='t' || nuchar=='T')
{
  AT(9,5);   OUT"\nPlease input the following values for tare plot:\n"); GO
  AT(11,5);   OUT"\nx-scale (%7.3f)... \n",xscale); GO
  scanf(" %f",&xscale);
  FL   OUT"\ny-scale upper plot (%7.3f)... \n",y1scale); GO
  scanf(" %f",&y1scale);
```

```
   FL   OUT"\ny-scale lower plot (%7.3f)... \n",y2scale); GO
   scanf(" %f",&y2scale);
   FL   CLS   AT(3,5);   OUT"Turn on X-Y plotter, load paper. Press ENTER to continue."); GO
   AT(5,5);   OUT"Else hit 'R' or 'r' to return without plotting axes.\n"); GO
   nuchar = getchar();
   FL   if(nuchar=='R' || nuchar=='r') {  return(0); }
   AT(15,10);   OUT"Now plotting tare axes..."); GO
   OUT"DT%c","\003');    OUT_XY buffer);                  /* initialize plotter */
   OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-70*y1scale,30*y1scale);
   OUT_XY buffer);
   OUT_XY"SP1;");   /* plot the upper x-axis */
   OUT_XY"PAPU 0,0;");
   for(dex=0; dex<14*xscale; dex+=xscale)   {
    OUT "PAPD %10.3f,0;XT;",(float)dex);    OUT_XY buffer);
   }
   OUT_XY"PU;");
   OUT_XY"PAPU 0,0;");   /* plot the upper y-axis */
   for(dex=-20*y1scale; dex<30*y1scale; dex+=10*y1scale)   {
    OUT "PAPD 0,%10.3f;YT;",(float)dex);    OUT_XY buffer);
   }
   OUT_XY"PU;");
   OUT_XY"SI,.19,.27;");   /* label upper x and y axes */
   for(dex=2*xscale; dex<13*xscale; dex+=2*xscale)   {
    OUT "PAPU %10.3f,%10.3f;",(float)dex-(.3*xscale),-4*y1scale);    OUT_XY buffer);
    OUT "LB%5d%c;PU;",dex,'\003');    OUT_XY buffer);
   }
   OUT_XY"PAPU 0,0;");
   for(dex=-20*y1scale; dex<30*y1scale; dex+=10*y1scale)   {
    OUT "PA %10.3f,%10.3f;",-.8*xscale,(float)dex);    OUT_XY buffer);
    OUT "LB%5d%c;PU;",dex/10,'\003');    OUT_XY buffer);
   }
   OUT_XY"PAPU 0,0;");   /* labels */
   OUT_XY"PAPU 0,%10.3f;",25*y1scale);
   OUT "LBK-IW^2, FT-LB/RAD  TEST      RUN%c;PU;","\003');   OUT_XY buffer);
   OUT_XY"PAPU 0,0;");
   OUT_XY"PAPU %10.3f,%10.3f;",2.8*xscale,25*y1scale);
   OUT "LB%d%c;PU;",testnum,'\003');   OUT_XY buffer);
   OUT_XY"PAPU 0,0;");
   OUT_XY"PAPU %10.3f,%10.3f;",4.2*xscale,25*y1scale);
   OUT "LB%d%c;PU;",runnum,'\003');   OUT_XY buffer);   /* plot the lower x-axis */
   OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-20*y2scale,80*y2scale);
   OUT_XY buffer);
   OUT_XY"PAPU 0,0;");
   for(dex=0; dex<14*xscale; dex+=xscale)   {
    OUT "PAPD %10.3f,0;XT;",(float)dex);    OUT_XY buffer);
   }
   OUT_XY"PU;");
   OUT_XY"PAPU 0,0;");   /* plot the lower y-axis */
   for(dex=-20*y2scale; dex<30*y2scale; dex+=10*y2scale)   {
    OUT "PAPD 0,%10.3f;YT;",(float)dex);    OUT_XY buffer);
   }
   OUT_XY"PU;");
    for(dex=2*xscale; dex<13*xscale; dex+=2*xscale) { /* label lower x and y axes */
    OUT "PAPU %10.3f,%10.3f;",(float)dex-(.3*xscale),-4*y2scale);    OUT_XY buffer);
    OUT "LB%5d%c;PU;",dex,'\003');    OUT_XY buffer);
   }
   OUT_XY"PAPU 0,0;");
   for(dex=-20*y2scale; dex<30*y2scale; dex+=10*y2scale)   {
    OUT "PA %10.3f,%10.3f;",-.8*xscale,(float)dex);    OUT_XY buffer);
    OUT "LB%5d%c;PU;",dex/10,'\003');    OUT_XY buffer);
   }
   OUT_XY"PAPU 0,0;");   /* labels */
   OUT_XY"PAPU 0,%10.3f;",25*y2scale);
   OUT "LBA-BW^2%c;PU;","\003');   OUT_XY buffer);
   OUT_XY"PAPU 0,0;");
   OUT_XY"PAPU %10.3f,%10.3f;",5*xscale,-15*y2scale);
   OUT "LBW^2, RAD/SEC^2%c;PU;","\003');   OUT_XY buffer);
   OUT_XY"PAPU 0,0;");
 } /* end of tare setup block */
} /*end of set up plots */
```

```
startover()
{ /* Resets all variables to to beginning state. Every time the user exits the numerical data
screen with ESC, this function restarts all PC and DSP programs */
 CLS                                    /* Clears screen */
 tmp[0][0]=0;             /* Prevents use of initial zeros from DSP */
 tmp[1][0]=0;
 tmp[0][1]=0;
 tmp[1][1]=0;
 avecount=0;                        /* Resets averaging arrays */
 setnew();               /* Menu program called for user direction */
 if(keith) {   ieeewt("output 16;func 'volt:ac'\n");
   ieeewt("output 16;CALC:STAT ON\n");
 }
}

switchboard()
{ /* Switches PC access to next DSP board. When b=0 program reads from board0: Displacement and
Secondary. When b=1 program reads from board1: Displacement and Torque. b is the index which
differentiates all variables not common to both boards */
 if(b==0)  {  b=1;  }         /* Switch b values */
 else    { b=0;   }
 dsp_board_addr(baddr[b]);  /* Access next DSP board */
}
```

Appendix G

Fifth Source File Listing:

```
/* file xtz.c */
/* Contains functions whose names begin with t-z. */
#include "dsphead.h"
takecaldat()
{ /* Take calibration data: store current data point, call plotting program for calibration
plots, perform linear regression, etc. */
 calpoint++;           /* Increment calibration point counter */
 if (calpoint==1)  {   CLS     /* If first point, input type calibration */
   OUT"Please enter the type of calibration.\n\n"); GO
   OUT"1. Displacement\n"); GO
   OUT"2. Torque\n"); GO
   OUT"3. Secondary Moment\n"); GO
   OUT"4. Secondary Force\n"); GO
   OUT"5. Auto Timed Calibrate\n\n"); GO
   typecal=getchar(); FL
   CLS   if(typecal=='5')   {
    OUT"Enter Update Interval in Minutes\n"); GO
    scanf(" %f",&updatime);
    oldtime=testtime;
   }
 }             /* type of calibration sets variable polled, user inputs and calculations */
 if(keith) {   ieeewt("output 16;data?\n");
   ieeewt("enter 16\n");
   ieeescnf("%f",&vcalin);
 }
 if(typecal=='1');
 if(typecal=='2');
 if(typecal=='3');
 if(typecal=='4');
 if(typecal!='5' || calpoint==1)
 {   CLS
   if(filename==NULL)   {
    filename=fopen(file,"w");
    OUT"File %s Opened\n",file); GO
   }
   if(keith)   {
    OUT"Point No %d. DC Calibrate Voltage= %8.6f Enter 1 to Accept ",
    calpoint,vcalin); GO
    scanf(" %f",&key);
   }
   else   {
    OUT"Point No %d. ENTER DC Calibrate Voltage\n",calpoint); GO
    scanf("%f",&vcalin);
    key=1;
   }
 }
 CLS if(key==1) {         /* some of these vars don't exist */
   STORE"%8.6f %8.6f %8.6f %8.6f %8.6f \n",vcalin,discal0,seccal,discal1,torcal);
 }
}

tareplot()
{   /* Sends tare data to plotter, comments are the corresponding BASIC code for each line */
 tarex[0][tarecount-1]=omega*omega;       /* 4410 X(E)=W2 */
 tarey[0][tarecount-1]=t1;             /*    Y(E)=T1 */
 tarelinreg[0][tarecount-1]=1;          /*    M(E)=1 */
 x=omega*omega;                    /*    X=W2 */
 y=t1;                           /*    Y=T1 */
 /* scale upper tare plot here */    /* 4470-4480 */
 OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-70*y1scale,30*y1scale);
 OUT_XY buffer); /* plot point */           /* 7080-7100 */
 OUT_XY"PAPU %10.3f,%10.3f;",tarex[0][tarecount-1],10*tarey[0][tarecount-1]);
 OUT_XY"SP1;SM*;PAPD %10.3f,%10.3f;DF;PU;",tarex[0][tarecount-1],10*tarey[0][tarecount-1]);
 OUT_XY"PAPU 0,0;");
```

```
 tarex[1][tarecount-1]=omega*omega;       /*    B(E)=W2 */
 tarey[1][tarecount-1]=m1;             /*    C(E)=M1 */
 tarelinreg[1][tarecount-1]=1;          /*    P(E)=1 */
 y=m1;                          /*    Y=M1 */
 /* scale lower tare plot here */    /* 4530-4550 */
 OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-20*y2scale,80*y2scale);
 OUT_XY buffer); /* plot point */
 OUT_XY"PAPU %10.3f,%10.3f;",tarex[1][tarecount-1],10*tarey[1][tarecount-1]);
 OUT_XY"SP1;SM*;PAPD %10.3f,%10.3f;DF;PU;",tarex[1][tarecount-1],10*tarey[1][tarecount-1]);
 OUT_XY"PAPU 0,0;");
 if(tarecount==14) { OUT"Next tare data point is last point!!!\n\n\n"); GO }
}

takedata()
{ /* Store data, perform calculations for coefficients, control wind-on or tare plotting */
 char   label;                        /* User input character */
 char   gg;                          /* user input character */
 char   timestr[8];               /* string to store time in file */
        /* Take a data point from the numerical or calibration programs */
 if (keyswork==1)  {            /* If calibration program running... */
   /* takecaldat();  */               /* auto ac or dc calibration */
/*   return(0); */                   /* do not take other data */
                      /* If first point, input type calibration */
   if (calpoint_d==0 && calpoint_z==0 && calpoint_s==0)   {
    calfile = fopen(cformat,"a");  /* Open calibration file for printing */
    for(i=0;i<=120;i++)  { calarray0[i]=0.0;
     calarray1[i]=0.0;
    }
    for(i=0;i<=15;i++)   { discal0_z[i]=0.0;
     discal1_z[i]=0.0;
     discal0_s[i]=0.0;
     discal1_s[i]=0.0;
     discal0_z[i]=0.0;
     seccal_z[i]=0.0;
     seccal_s[i]=0.0;
     torcal_z[i]=0.0;
     torcal_s[i]=0.0;
    }
    CLS    AT(9,1);    OUT"Displacement Channel 0/1...1\n"); GO
    OUT"Secondary Moment/Force.....2\n"); GO
    OUT"Torque.....3\n"); GO
    OUT"Please enter the type of calibration..."); GO
    typecal=getchar(); FL
    switch(typecal)    {
     case '1': sprintf(cal_name,"Displacement 0/1"); break;
     case '2': sprintf(cal_name,"Secondary M/F"); break;
     case '3': sprintf(cal_name,"Torque"); break;
    }
    SDCAL"%s Calibration\n",cal_name);   /* store cal type in disk file */
    SDCAL ctime(&testtime));       /* date & time and gains in calfile */
    SDCAL"\nGain of Amplifier %3d is %12.1f\n",ampchan,gain[ampchan]);
    SDCAL"Test Number: %d, Calibration Number: %d\n",testnum,calibnum);
    SDCAL"Wind Tunnel Temperature %10.2f\n",temperature);
    SDCAL"Moment Arm Length %13.5f\n\n",momentarm);                 /* column heads */
    SDCAL"DTC SHO   Z0/S0      Z1/S1    D0, M/F, T      D1       INDEP\n");
   } /* end of first cal point loop */
   CLS   OUT"Is this the last data point? (y or Y, n or N)\n"); GO
   OUT"\n   >>>"); GO
   gg=getchar();
   gg=(char)toupper(gg);
   FL  if(gg=='Y') {    lastcalpt=1;  }
   else {    lastcalpt=0;  }
   CLS   if (typecal=='1') {
    if (calcode==10)    {
     calpoint_z++;         /* increment zero data counter */
     discal0_z[calpoint_z]=discal0;  /* save zero data */
     discal1_z[calpoint_z]=discal1;
     SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
         shuntopt,discal0,discal1,0.0,0.0,0.0); /* save to disk */
    }
```

```
 if (calcode==11)    {
  calpoint_s++;     /* increment shunt data counter */
  discal0_s[calpoint_s]=discal0;  /* save shunt data */
  discal1_s[calpoint_s]=discal1;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,discal0,discal1,0.0,0.0,0.0); /* save to disk */
 }
 if (calcode==12)    {
  calpoint_d++;          /* Increment calibration point counter */
  calarray0[calpoint_d]=discal0;  /* save current data point */
  calarray1[calpoint_d]=discal1;
  getindepvar();    /* get independent variable data from user */
  calindep[calpoint_d]=indepvar;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,0.0,0.0,discal0,discal1,indepvar); /* save to disk */
 }
}   /* end of typecal=1 */
if (typecal=='2')   {
 if (calcode==10)    {
  calpoint_z++;     /* increment zero data counters */
  seccal_z[calpoint_z]=seccal;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,seccal,0.0,0.0,0.0,0.0); /* save to disk */
 }
 if (calcode==11)    {
  calpoint_s++;     /* increment shunt data counters */
  seccal_s[calpoint_s]=seccal;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,seccal,0.0,0.0,0.0,0.0); /* save to disk */
 }
 if (calcode==12)    {
  calpoint_d++;          /* Increment calibration point counter */
  calarray0[calpoint_d]=seccal;  /* save current data point */
  getindepvar();    /* get independent variable data from user */
  calindep[calpoint_d]=indepvar;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,0.0,0.0,seccal,0.0,indepvar); /* save to disk */
 }
} /* end of typecal=2 */
if (typecal=='3')   {
 if (calcode==10)    {
  calpoint_z++;     /* increment zero data counters */
  torcal_z[calpoint_z]=torcal;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,torcal,0.0,0.0,0.0,0.0); /* save to disk */
 }
 if (calcode==11)    {
  calpoint_s++;     /* increment shunt data counters */
  torcal_s[calpoint_s]=torcal;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,torcal,0.0,0.0,0.0,0.0); /* save to disk */
 }
 if (calcode==12)    {
  calpoint_d++;          /* Increment calibration point counter */
  calarray0[calpoint_d]=torcal;  /* save current data point */
  getindepvar();    /* get independent variable data from user */
  calindep[calpoint_d]=indepvar;
  SDCAL"%3d %3d %13.6f %13.6f %13.6f %13.6f %13.6f\n",calcode,
      shuntopt,0.0,0.0,torcal,0.0,indepvar); /* save to disk */
 }
} /* end of typecal=3 */

if(lastcalpt && calpoint_d >3 )  {    /* append calfile with statistics */
 SDCAL"Total # of Calibration points: %d\n",calpoint_d);/* in calfile */
 if(typecal=='1')    {
  leastsq(calpoint_d,calindep,calarray0);   /* compute least squares */
  calcoffa0=lscoffa;        /* save regression values for display */
  calcoffb0=lscoffb;                 /* Disp 0 */
  standev0=lsstdev;
  leastsq(calpoint_d,calindep,calarray1);   /* Disp 1 */
```

```
    calcoffa1=lscoffa;
    calcoffb1=lscoffb;
    standev1=lsstdev;
    SDCAL"\nm(0)= %13.6f, b(0)=%13.6f, and Std Err(0)=%13.6f\n",calcoffa0,
       calcoffb0,standev0);
    SDCAL"m(1) = %13.6f, b(1) =%13.6f, and Std Err(1)=%13.6f\n",calcoffa1,
       calcoffb1,standev1);
   }
   else    {
    leastsq(calpoint_d,calindep,calarray0);   /* compute least squares */
    calcoffa0=lscoffa;        /* save regression values for display */
    calcoffb0=lscoffb;                  /* Secondary or Torque */
    standev0=lsstdev;
    SDCAL"\nm= %13.6f, b=%13.6f, and Std Err=%13.6f\n",calcoffa0,
       calcoffb0,standev0);
   }
   fclose(calfile);               /* close calibration data file */
   u6=0;                       /* change main menu colors... */
   u8=0;                       /* to remind user to update... */
   u9=0;                              /* these options. */
  }   /* end of last cal point */
  flushall();        /* flush all stream buffers or send data to file */
  return(0);                        /* Do not take other data */
}     /* end of take cal data */
else {
  ptnum++;                          /* Increment point counter */
  OUT"Taking data\n\n\n"); CLS AT(1,26); GO
  getdatacode();               /* Read data code from DAS-20 board */
  formated = fopen(format,"a");     /* Open formatted file for printing */
  if (ptnum==1)  { /* If first point output header info to formatted data file */
   time(&testtime);               /* Find and output time to file */
   OUT ctime(&testtime));
   SD"%s\n",buffer); GO              /* time is already in buffer */
   SD"DYNAMIC STABILITY UPWT REAL-TIME DATA REDUCTION\n\n");
   OUT"DYNAMIC STABILITY UPWT REAL-TIME DATA REDUCTION\n\n");    GO
   SD"Filename for Dynamic Stability Data Storage: %s\n",file);
   OUT"Filename for Dynamic Stability Data Storage: %s\n",file); GO
        /* Output all user input conditions and calibration constants */
   SD"kd = %9.7f rad/cal sig,  kt = %9.7f ft-lb/cal sig\n",
     calib[0],calib[1]);
   OUT"kd = %9.7f rad/cal sig,  kt = %9.7f ft-lb/cal sig\n",
     calib[0],calib[1]); GO
   SD"ks = %9.7f lb or ft-lb/cal sig\n",calib[2]);
   OUT"ks = %9.7f lb or ft-lb/cal sig\n",calib[2]); GO
   SD"Alpha sensitivity: %7.5f g/volt, ",alphasens);
   OUT"Alpha sensitivity: %7.5f g/volt, ",alphasens); GO
   SD"Alpha bias: %7.5f volts,\n",alphabias);
   OUT"Alpha bias: %7.5f volts,\n",alphabias); GO
   SD"Alpha offset: %5.3f°, ",alphaoff);
   OUT"Alpha offset: %5.3f°, ",alphaoff); GO
   SD"Delta alpha (sting to model): %5.3f°\n",alphadelt);
   OUT"Delta alpha (sting to model): %5.3f°\n",alphadelt); GO
   SD"Reference Length - B: %4.2f feet, ",reflength);
   OUT"Reference Length - B: %4.2f feet, ",reflength); GO
   SD"Reference Area - S: %6.4f ft^2\n\n",refarea);
   OUT"Reference Area - S: %6.4f ft^2\n\n",refarea); GO
                /* Output test and run number to file and screen */
   SD"Test number = %d,     Run number = %d\n\n",testnum,runnum);
   OUT"Test number = %d,     Run number = %d\n\n",testnum,runnum); GO
  }
  if (WINDON && ptnum==1) {      /* If first point and wind-on, output...*/
   SD"Kspring= %9.5f ft-lb/rad, Inertia= %9.5f slug-ft^2",kspring,inertia);
   SD"\nCtare= %9.5f, AMR=%9.5f, BMR= %9.5f, (mj/wd)tare= %9.5f",ctare,amr,bmr,mjwdtare);
   OUT"Kspring= %9.5f ft-lb/rad, Inertia= %9.5f slug-ft^2",kspring,inertia); AT(15,1); GO
   OUT"Ctare= %9.5f, AMR= %9.5f, BMR= %9.5f, (mj/wd)tare= %9.5f",ctare,amr,bmr,mjwdtare);
   AT(17,1); GO
   SD"\nSensitivity 1= %10.7f,  Sensitivity 2= %10.7f",sen[0],sen[1]);
   OUT"Sensitivity 1= %10.7f,  Sensitivity 2= %10.7f",sen[0],sen[1]); AT(19,1); GO
   SD"\nSensitivity 3= %10.7f",sen[2]);
   OUT"Sensitivity 1= %10.7f,",sen[2]); AT(21,1); GO
```

```
            /*...mach #, pressure, temperature, to file and screen. */
    SD"\nMACH = %3.2f, P TOTAL = %5.4f psi, T TOTAL = %2.1f °F",mach,
      pressure,temperature);
    OUT"MACH = %3.2f, P TOTAL = %5.4f psi, T TOTAL = %2.1f °F",mach,pressure,temperature); GO
    SD"\nQ = %4.3f lb/ft^2, VELOCITY = %4.3f ft/sec",Q,velocity);
    OUT"\nQ = %4.3f lb/ft^2, VELOCITY = %4.3f ft/sec",Q,velocity); GO
    SD"REYNOLDS NUMBER PER FOOT = %f E+06\n\n",reynolds);
    OUT"REYNOLDS NUMBER PER FOOT = %f E+06\n\n",reynolds); GO
   }
   if(ptnum==1)           /* Output column headings on 1st point */
    SD"DISPLACEMENT        TORQUE            SECONDARY\n\n");
            /* Output info common to all datacodes to screen and file */
   SD"Point number %d, DATATYPE = %d, %s",ptnum,datacode,ctime(&testtime));
   OUT"Point number %d, DATATYPE = %d, %s",ptnum,datacode,ctime(&testtime));
   GO                        /* Output gains to screen and file */
   SD"GAIN d = %-6.1f   GAIN t = %-6.1f     GAIN s = %-6.1f\n",
    gain[0],gain[1],gain[2]);
   OUT"GAIN d = %-6.1f   GAIN t = %-6.1f     GAIN s = %-6.1f\n",
    gain[0],gain[1],gain[2]); GO
   SD"OFFSET d = %-6.5f   OFFSET t = %-6.5f    OFFSET s = %-6.5f\n",
    offset[0],offset[1],offset[2]);
   OUT"OFFSET d = %-6.5f   OFFSET t = %-6.5f    OFFSET s = %-6.5f\n",
    offset[0],offset[1],offset[2]); GO
   /* Perform data zero calculations for datacodes 2,6, and 1 */
if(datacode == 2 || datacode==6 || datacode==1)   { /* store zeroes */
    zero[0]=peak[2];                              /* disp r */
    zero[1]=cforce[0];                          /* torque real */
    zero[2]=cforce[1];                        /* secondary real */
    zero[3]=0;                       /* displacement imaginary */
    zero[4]=imag[0];                       /* torque imaginary */
    zero[5]=imag[1];                     /* secondary imaginary */
    for (i=0;i<6;i++) { voltage[i]=zero[i]; }   /* copy zeros to volts for storing in file */
    /* Output zero data to file and screen */
    SD"ZERO dr = %7.5f   ZERO tr = %7.5f     ZERO sr = %7.5f\n",peak[2],real[0],real[1]);
    OUT"ZERO dr = %7.5f   ZERO tr = %7.5f     ZERO sr = %7.5f\n",peak[2],real[0],real[1]); GO
    SD"STDEV d = %7.4f  STDEV t = %7.4f  STDEV s=%7.4f\n",
devpeak[0][1],devreal[0],devreal[1]);
    OUT"STDEV d = %7.4f   STDEV t = %7.4f     STDEV s = %7.4f\n",
      devpeak[0][1],devreal[0],devreal[1]); GO
    SD"ZERO di =  0.000   ZERO ti = %7.5f     ZERO si = %7.5f\n",imag[0],imag[1]);
    OUT"ZERO di =  0.000   ZERO ti = %7.5f     ZERO si = %7.5f\n",imag[0],imag[1]); GO
    SD"STDEV d =  0.0000   STDEV t = %7.4f     STDEV s = %7.4f\n\n",
      devimag[0],devimag[1]);
    OUT"STDEV d = 0.0000   STDEV t = %7.4f     STDEV s = %7.4f\n\n",
      devimag[0],devimag[1]); GO
   }                                          /* Output voltage data to file and screen */
   if(datacode == 3 || datacode==7 || datacode==9) {     SD"VOLT dr = %7.5f   VOLT tr = %7.5f
 VOLT sr = %7.5f\n",peak[2],real[0],real[1]);
    OUT"VOLT dr = %7.5f   VOLT tr = %7.5f     VOLT sr = %7.5f\n",peak[2],real[0],real[1]); GO
    SD"STDEV d = %7.4f   STDEV t = %7.4f     STDEV s = %7.4f\n",
      devpeak[0][1],devreal[0],devreal[1]);
    OUT"STDEV d = %7.4f   STDEV t = %7.4f     STDEV s = %7.4f\n",
     devpeak[0][1],devreal[0],devreal[1]); GO
    SD"VOLT di =  0.000   VOLT ti = %7.5f     VOLT si = %7.5f\n",imag[0],imag[1]);
    OUT"VOLT di =  0.000   VOLT ti = %7.5f     VOLT si = %7.5f\n",imag[0],imag[1]); GO
    SD"STDEV d = 0.0000   STDEV t = %7.4f     STDEV s = %7.4f\n\n",
      devimag[0],devimag[1]);
    OUT"STDEV d = 0.0000   STDEV t = %7.4f     STDEV s = %7.4f\n\n",
      devimag[0],devimag[1]); GO
    if (datacode==9) {       /* Output sensitivities to screen and formatted data file */
     SD"SENS 1 = %8.7f SENS 2 = %8.7f SENS 3 = %8.7f\n\n",  sen[0],sen[1],sen[2]);
     OUT"SENS 1 = %8.7f SENS 2 = %8.7f SENS 3 = %8.7f\n\n",  sen[0],sen[1],sen[2]); GO
    }
    if (datacode==3)  (        /* Tare storage when datacode = 3 */
     tarecount++;             /* Increment tare data point counter */
     if(tarecount==1)           /* Output title bar first time only */
       SD"\nTheta   Freq    Disp    Tj/wd   Tr/D    Mj/wd   Mj/d\n\n");
     SD"Tare data point number: %d\n",tarecount);
     OUT"Tare data point number: %d\n",tarecount); GO
     /* Output calculated values for databar */
```

```
     SD"%3.2f  %5.4f  %6.5f  %6.5f  %6.5f  %6.5f  %6.5f\n\n",
       AoAval,freq,reading[0],t2,t1,m2,m1);
     OUT"%3.2f  %5.4f  %6.5f  %6.5f  %6.5f  %6.5f  %6.5f\n\n",
       AoAval,freq,reading[0],t2,t1,m2,m1); GO
     tareplot();             /* Calculate tare constants and plot */
    } /* end of datacode=3 */
    if (datacode==7)  {               /* store and plot when datacode = 7 */
      windoncount++;             /* increment wind-on point counter */
     if (woplotcode == 1 || woplotcode == 2) {    /* Wind-on plot code */
       if(titlebar==0) {   /* Output titlebar (first time only to file) */
        SD"Theta  Freq    Disp   cmq+cma*   cma-kkcm*   cnq+cna*   cna-kkcnq*\n");
        SD" deg   Hz     rad    /rad      /rad     /rad     /rad\n\n");   GO
        titlebar=1;                  /* Title bar switch off */
       }
       OUT"Theta  Freq    Disp   cmq+cma*   cma-kkcm*   cnq+cna*   cna-kkcnq*\n");
       OUT" deg   Hz     rad    /rad      /rad     /rad      /rad\n\n");       GO
     }
     else if(woplotcode == 3 || woplotcode == 4) {   /* Wind-on plot code */
       if(titlebar==0){   /* Output title bar (first time only to file) */
        SD"Theta  Freq    Disp  cnr-cosacnb*   cnbcosa  clr-cosaclb*  clbcosa\n");
        SD"                           +kkcnr*            +kkclr*\n");
        SD"                    /rad      /rad      /rad     /rad\n\n");
        titlebar=1;                 /* Title bar switch off */
       }
       OUT"Theta  Freq    Disp  cnr-cosacnb*   cnbcosa  clr-cosaclb*  clbcosa\n"); GO
       OUT"                          +kkcnr*            +kkclr*\n");   GO
     }
     else     {
       if (titlebar==0)  {  /* Output title bar (first time only to file) */
        SD"Theta  Freq    Disp   clp+sinaclb*  clbsina  cnp+sinacnb*  cnbsina\n");
        SD"                           -kkclp*            -kkcnp*\n");
        SD"                     /rad     /rad       /rad     /rad\n\n");
        titlebar=1;                  /* title bar switch off */
       }
       OUT"Theta Freq    Disp   clp+sinaclb*   clbsina  cnp+sinacnb*  cnbsina\n");  GO
       OUT"                          -kkclp*            -kkcnp*\n");
     }     /* output calculated quantities in data bar to screen and file */
     SD"%3.2f  %5.4f  %6.5f  %6.5f    %6.5f    %6.5f    %6.5f\n\n",
     AoAval,freq,reading[0],t2,t1,m2,m1);
     OUT"%3.2f  %5.4f  %6.5f  %6.5f    %6.5f    %6.5f    %6.5f\n\n",
     AoAval,freq,reading[0],t2,t1,m2,m1); GO
     windonplot();            /* Wind-on calculations and plotting */
    } /* end of dc=7 */
   } /* end of 3-7-9 */
   OUT"\nPress ENTER to return to numerical data screen."); GO
   getchar(); FL          /* Wait for user to look at data */
   CLS
 } /* end of real-time take data */
 fclose(formated);         /* close the formatted data file each time */
 /* Output the time in format which can be read by BASIC re-reduction */
 sscanf(ctime(&testtime),"%*s%*s%*s%8s%*s",timestr);
 /* Store the numerical data to a file for re-reduction by BASIC program */
 /* new format due to addition of offsets */
 filename = fopen(file,"a");             /* Open numerical data file */
 STORE"%d %d %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f 0 %f %f %f %f %f %f %s\n",
 testnum,runnum,datacode,aoavolts,freq,gain[0],gain[1],gain[2],
 offset[0],offset[1],offset[2],voltage[0],voltage[1],voltage[2],voltage[3],
 voltage[4],voltage[5],devpeak[0][1],devreal[1],devreal[0],devimag[1],
 devimag[0],devaoa,mach,pressure,temperature,timestr);
 fclose(filename);
}             /* end of takedata */

taredata()
{ /* This code is taken directly from the BASIC program. This function corresponds to BASIC
lines 6090-7060.   ===== Now called by F7 ======   */
 int loop=1;              /* loop control variable */
 char menu;               /* User menu choice */
 int num;                /* User input number */
 int whichplot;            /* Upper or lower tare data? */
 float f2,f3,f4;           /* Tare linear regression variables */
```

```
float d2,d3,d4;           /* Tare linear regression variables */
int tarenum;             /* Total number of tare points */
float slope;             /* Tare lin. reg. slope */
float yintercept;          /* Tare plot y intercept */
float correlation;         /* Linear regression correlation */
float xintercept;          /* Tare plot x intercept */
float xtemp[15];           /* Working variables for tare data...*/
float ytemp[15];
float linreg[15];
float xamend,yamend;        /* temp. vars. for amended point */
float xtare,ytare;         /* plotting variables */
whichplot=0;                   /* Work first with upper data set */
for(i=0;i<tarecount;i++) {    /* Copy upper data set into working array */
  xtemp[i]=tarex[whichplot][i];
  ytemp[i]=tarey[whichplot][i];
  linreg[i]=tarelinreg[whichplot][i];
}
while(loop) {   /* Menu for manipulating tare data */
  CLS   OUT"\n\n\nPlease choose an option.\n\n"); GO
  OUT"Amend data point..............1\n"); GO
  OUT"Omit data point...............2\n"); GO
  OUT"Compute linear regression.....3\n"); GO
  if(whichplot)   {  OUT"Load upper tare data set......4\n"); GO   }
  else  {    OUT"Load lower tare data set......4\n"); GO   }
  OUT"Exit..........................5\n"); GO
  menu=getchar(); FL    CLS
  tarenum=tarecount;   /* User input amended point data */
  if(menu=='1')   {
   OUT"\nWhich point needs correcting? 0-14... "); GO
   scanf(" %d",&num); FL
   OUT"\n\nPlease input new values:\n"); GO
   OUT"x[%d] = (%f)... ",num,xtemp[num]); GO
   scanf(" %f",&xamend); FL
   OUT"\ny[%d] = (%f)... ",num,ytemp[num]); GO
   scanf(" %f",&yamend); FL
   xtare=xtemp[num];
   ytare=ytemp[num];
   if(whichplot)    {                /* scale lower correction */
    OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-20*y2scale,80*y2scale);
    OUT_XY buffer);
   }
   else    {                /* scale upper correction */
    OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-70*y1scale,30*y1scale);
    OUT_XY buffer);
   }               /* call routine to cross out bad tare point 7120-7160 */
   OUT"PAPU,%10.3f,%10.3f;",xtare,ytare*10);    OUT_XY buffer);
   OUT"SP1;SM#;PAPD,%10.3f,%10.3f;DF;PU;",xtare,ytare*10);
/*   tarecount=num;  */          /* 6270 */
   xtare=xamend;
   ytare=yamend;
   xtemp[num]=xtare;
   ytemp[num]=ytare;               /* re-scale for the point and re-plot it */ /* 6292...*/
   if(whichplot)    {                /* scale lower correction */
    OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-20*y2scale,80*y2scale);
    OUT_XY buffer);
   }
   else    {                /* scale upper correction */
    OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-70*y1scale,30*y1scale);
    OUT_XY buffer);
   }                          /* plot it */
   OUT"PAPU,%10.3f,%10.3f;",xtare,ytare*10);    OUT_XY buffer);
   OUT"SP1;SM*;PAPD,%10.3f,%10.3f;DF;PU;",xtare,ytare*10);    CLS
  }
  if(menu=='2')   { /* User select a point to omit */
   OUT"\nWhich point is to be omitted? 0-%d... ",tarecount); GO
   scanf(" %d",&num); FL
   OUT"\n\nPoint: x[%d] = %f  y[%d] = %f\n\n",num,xtemp[num],num,ytemp[num]); GO
   OUT"Delete? y or n... "); GO
   menu=getchar();    FL
   if (menu=='y' || menu=='Y')    {
```

```
  OUT"\n\nPOINT OMITTED\n\n"); GO
  xtare=xtemp[num];     /* 6370 */
  ytare=ytemp[num];                 /* scale for point and cross out */
  if(whichplot)     {               /* scale lower correction */
    OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-20*y2scale,80*y2scale);
    OUT_XY buffer);
  }
  else     {                /* scale upper correction */
    OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-70*y1scale,30*y1scale);
    OUT_XY buffer);
  }               /* call routine to cross out bad tare point 7120-7160 */
  OUT"PAPU,%10.3f,%10.3f;",xtare,ytare*10);     OUT_XY buffer);
  OUT"SP1;SM#;PAPD,%10.3f,%10.3f;DF;PU;",xtare,ytare*10);   linreg[num]=0;
 }
 CLS
}   /* Perform linear regression of tare data */
if(menu=='3')   { formated = fopen(format,"a");   /* Open formatted file for printing */
 f2=0;
 f3=0;
 f4=0;
 for(i=0;i<tarenum;i++)   {        /* BASIC 6450 */
  f2=f2+xtemp[i]*linreg[i];
  f3=f3+ytemp[i]*linreg[i];
  f4=f4+linreg[i];
 }
 f2/=f4;                       /* 6500 */
 f3/=f4;
 d2=0;
 d3=0;
 d4=0;
 for(i=0;i<tarenum;i++)     {      /* 6550 */
  d2=d2+(xtemp[i]-f2)*(ytemp[i]-f3)*linreg[i];
  d3=d3+pow(xtemp[i]-f2,2)*linreg[i];
  d4=d4+pow(ytemp[i]-f3,2)*linreg[i];
 }
 slope=d2/d3;
 yintercept= f3 - slope*f2;
 correlation = d2/pow(d3*d4,.5);
 xintercept= -yintercept/slope;    /* Output to screen and formatted data file */
 SD"\n\nTare data set %d  ",whichplot);
 OUT"\n\nTare data set %d  ",whichplot); GO
 SD"y = %fx + %f\n",slope,yintercept);
 OUT"y = %fx + %f\n",slope,yintercept); GO
 SD"Correlation factor = %f\n",correlation);
 OUT"Correlation factor = %f\n",correlation); GO
 SD"Crosses x-axis at x = %f\n\n",xintercept);
 OUT"Crosses x-axis at x = %f\n\n",xintercept); GO
 OUT" Press ENTER to return\n"); GO
 getchar();    FL
 fclose(formated);       /* close the formatted data file each time */
 if(whichplot) {               /* upper data set is whichplot=0 */
  amr=yintercept;        /* Save calculations in global variables */
  bmr=slope;                       /* Code to draw lower tare line 6920-7060 */
  OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-20*y2scale,80*y2scale);
  OUT_XY buffer);
  OUT"PAPU 0,%10.3f;",10*yintercept);  OUT_XY buffer);
  xtare=14*xscale;
  ytare=slope*xtare+yintercept;
  if(ytare > 8 * y2scale)  {ytare = 8 * y2scale; xtare = (ytare - yintercept)/slope; }
  else if(ytare < -2*y1scale) { ytare = -2 * y1scale;
    xtare = (-2 * y1scale - yintercept) / slope;     }
  OUT_XY"SP1;");
  OUT "PAPD %10.3f,%10.3f;",xtare,ytare*10);     OUT_XY buffer);
  OUT_XY"PAPU,0,0;");
 }
 else    {
  kspring=yintercept; /* Save calculations in global variables */
  inertia=slope;                       /* Code to draw upper tare line 6780-6910 */
  OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",-1*xscale,14*xscale,-70*y1scale,30*y1scale);
  OUT_XY buffer);
```

```
      OUT"PAPU 0,%10.3f;",10*kspring);     OUT_XY buffer);
      xtare=14*xscale;
      ytare=slope*xtare+yintercept;
      if(ytare > 3*y1scale) { ytare = 3 * y1scale; xtare = (ytare - yintercept)/slope; }
      else if(ytare < -7*y1scale)     { ytare = -7 * y1scale;
        xtare = (-7 * y1scale - yintercept) / slope;     }
      OUT_XY"SP1;");
      OUT "PAPD %10.3f,%10.3f;",xtare,ytare*10);     OUT_XY buffer);
      OUT_XY"PAPU,0,0;");
     }
    }
    /* Switch between upper and lower tare data sets */
    if(menu=='4')   {
     for(i=0;i<15;i++)    {
       tarex[whichplot][i]=xtemp[i];
       tarey[whichplot][i]=ytemp[i];
       tarelinreg[whichplot][i]=linreg[i];
     }
     if(whichplot==0) { whichplot=1; }
     else             { whichplot=0; }
     for(i=0;i<15;i++)    {
      xtemp[i]=tarex[whichplot][i];
      ytemp[i]=tarey[whichplot][i];
      linreg[i]=tarelinreg[whichplot][i];
     }
    }   /* Exit tare data manipulation loop */
    if(menu=='5')  {   loop=0;  }
  }
  for(i=0;i<15;i++)  {          /* Store current data in arrays */
    tarex[whichplot][i]=xtemp[i];
    tarey[whichplot][i]=ytemp[i];
    tarelinreg[whichplot][i]=linreg[i];
  }
 }

 testcond()
 {    /* Allows user to enter temperature, pressure, and other test conditions */
  int gch;
  CLS   OUT"Please enter the following data...\n\n\n"); GO
  OUT"Enter p to accept value in parentheses\n\n\n"); GO
  OUT"Test number (%d)... ",testnum); GO
  scanf(" %d",&testnum); FL
  OUT"\nTare or Wind-on Run number (%d)... ",runnum); GO
  scanf(" %d",&runnum); FL
  OUT"\nCalibration moment arm length (%13.5f)... ",momentarm); GO
  scanf("%f",&momentarm); FL
  OUT"\nCalibration number (%d)... ",calibnum); GO
  scanf("%d",&calibnum); FL
  OUT"\nTemperature in °F (%7.3f)... ",temperature); GO
  scanf(" %f",&temperature); FL       /* Accept user temperature */
  OUT"\nPressure in lb/in^2 (%7.3f)... ",pressure); GO
  scanf(" %f",&pressure); FL         /* Accept user pressure */
  OUT"\nMach number (%5.2f)... ",mach); GO
  scanf(" %f",&mach); FL
  OUT"\nUPWT AoA flow angle correction (%5.2f)... ",flow_angle); GO
  scanf(" %f",&flow_angle); FL
  OUT"\n\nModel inverted? yes=1 no=0 (%d)... ",inverted); GO
  scanf(" %d",&inverted); FL
  OUT"\n\nDo you want to enter sensitivities from the keyboard? "); GO
  gch=getchar();
  if(gch=='Y' || gch=='y') {
    FL   OUT"\n\nSensitivity 1... "); GO
    scanf(" %d",&sen[0]);   FL
    OUT"\n\nSensitivity 2... "); GO
    scanf(" %d",&sen[1]);   FL
    sen[4]=sen[1];
    OUT"\n\nSensitivity 3... "); GO
    scanf(" %d",&sen[2]);   FL
    sen[5]=sen[2];
  }
```

```
 FL   parameters();          /* Calculate dynamic pressure,reynolds #,etc.*/
 CLS   u8=1;               /* Turns off white in main menu */
}

windonplot()    /* xy-plotter routine for datacode=7 BASIC program lines 1370-2430 and
5070-5230 are found in datac7, inputwopc, and here */
{ int   dex; /* now plot some points, BASIC lines 5070-5230 */
   if(windoncount==14) { OUT"Next data point is last point!!!\n\n\n"); GO }
OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",100*xscale*xmin,100*xscale*xmax,
  -800*y1scale,200*y1scale);   OUT_XY buffer);         /* plot coefficient vs. alpha */
 x=AoAval*100;                     /* alpha computed in readaoa() */
 y=100*t2;
 OUT"PAPU %10.3f,%10.3f;",x,y); OUT_XY buffer);
 if(wosymbolc==1) { OUT_XY"SP1;SM*;PAPD %10.3f,%10.3f;",x,y);  }
 if(wosymbolc==2) { OUT_XY"SP1;SM#;PAPD %10.3f,%10.3f;",x,y);  }
 if(wosymbolc==3) { OUT_XY"SP1;SMX;PAPD %10.3f,%10.3f;",x,y);  }
 OUT_XY"DF;PU;PAPU,0,0");
 if(woplotcode==2 || woplotcode==4 || woplotcode==6) { y=100*m2; }
 else {   y=100*t1;  }
 OUT"SC%10.3f,%10.3f,%10.3f,%10.3f;",100*xscale*xmin,100*xscale*xmax,
    -300*y2scale,700*y2scale); OUT_XY buffer);
 OUT_XY"PAPU 0,0;");
 OUT"PAPU %10.3f,%10.3f;",x,y); OUT_XY buffer);
 if(wosymbolc==1) { OUT_XY"SP1;SM*;PAPD %10.3f,%10.3f;",x,y);  }
 if(wosymbolc==2) { OUT_XY"SP1;SM#;PAPD %10.3f,%10.3f;",x,y);  }
 if(wosymbolc==3) { OUT_XY"SP1;SMX;PAPD %10.3f,%10.3f;",x,y);  }
 OUT_XY"DF;PU;PAPU,0,0");
}
```

Appendix H

Make File *Haste*

The **MAKE** utility uses the source file called *haste* which is listed below.

```
xma.obj:      xma.c
# This section contains main()
 cl /AL /c xma.c > comp$ma.err

xag.obj:      xag.c
# This section contains functions a-g
 cl /AL /c xag.c > comp$ag.err

xhp.obj:      xhp.c
# This section contains functions h-p
 cl /AL /c xhp.c > comp$hp.err

xrs.obj:      xrs.c
# This section contains functions r-s
 cl /AL /c xrs.c > comp$rs.err

xtz.obj:      xtz.c
# This section contains functions t-z
 cl /AL /c xtz.c > comp$tz.err

dsp.exe:   xma.obj xag.obj xhp.obj xrs.obj xtz.obj
 link xma+xag+xhp+xrs+xtz+ieeeiol,dsp,,c:\lib\dsputill.lib c:\lib\das20cl.lib; > link$az.err
```

Tables

Table 1. DSIS Hardware Under PC Control

Rack Component	Interface Type
HP3245A Universal Source	IEEE-488
HP3325A Function Generator	IEEE-488
PSC8000 Signal conditioners	IEEE-488
2001 Digital Multimeter	IEEE-488
Main Interface Box	IEEE-488
Motor Speed Control Chassis	IEEE-488
RPM Indicator Chassis	IEEE-488
XY Plotter	RS-232

Table 2. PC Directory Information

Directory	Description	Files or Programs of Interest
C:	Start-up Disk Drive	*AUTOEXEC.BAT AND CONFIG.SYS*
C:\SOURCE	Source Code	*DSPHEAD.H, XMA.C, XAG.C, XHP.C, AND XTZ.C*
C:\BIN *C:\LIB* *C:\INCLUDE*	C Language Compiler and Linker	**CL.EXE** **LINK.EXE** **MAKE.EXE**
C:\UTIL	Utilities	**NC.EXE AND QEDIT.EXE**
C:\DSP	DSIS Software Programs	**DSP.EXE**
C:\DSP32SL *C:\DSP_IRD*	DSP Microcomputer Software Development	
C:\DOS	Operating System	**PRINT.EXE, DIR.EXE**
C:\QB45	QUICKBasic Version 4.5	**QB.EXE**
C:\DATA *C:\CTEST*	Storage for data files	

Table 3. Actions Performed by Function **Takedata()**

Code Type and Value	Data Type	Action taken by **Takedata()**
calcode=10	Calibration Zero	increment zero counter, store data
calcode=11	Calibration Shunt	increment shunt counter, store data
calcode=12	Calibration Data	increment data counter, store data
datacode=1	Sensitivity Zero	increment zero counter, store data
datacode=9	Sensitivity Data	increment data counter, store data
datacode=2	Tare Zero	increment zero counter, store and plot data
datacode=3	Tare Data	increment data counter, store and plot data
datacode=6	Wind-on Zero	increment zero counter, store and plot data
datacode=7	Wind-on Data	increment data counter, store and plot data

Table 4. Definition of Variable *Keyswork*

Value	Calling Function
0	**plot()**
1	**dynstat()**
2	**fftcommand()**
3	**calibrate()**

Table 5. Data Reduction Parameters

Parameter	Variable Name
Dynamic pressure	*Q*
Static wind tunnel temperature	*temperature*
Velocity	*velocity*
Reynolds number	*reynolds*
Reference wing area	*refarea*
Reference length	*reflength*
C1	*c1*
C2	*c2*
C3	*c3*
C4	*c4*

Table 6. Main Menu Options

Option	Description
0	Exit software
1	Run the program selected in option 3
2	Plot a real-time FFT spectra to the screen
3	Select one of three different DSP algorithms
4	Select digital filter parameters
5	Select averaging or immediate mode for data display
6	Input calibration constants
7	Input model attitude constants
8	Input run and test numbers and conditions
9	Input filename for data storage
10	Input Wind-on parameters and initialize XY plotter
11	Display model attitude in real-time
12	Select data file for hard copy print
13	Reset data counters to 1

Table 7. Definition of Variable *P*

Value	Current Mode of Operation
1	Enhanced FFT
2	Stand Alone FFT @250
3	Stand Alone FFT @125
4	Synchronous Demodulation, 0.25 Hz filter
5	Synchronous Demodulation, 0.10 Hz filter
9	Calibration
-9	Plotting with Flattop window
-8	Plotting with Hanning window @ 250 Hz
-7	Plotting with Hanning window @ 125 Hz

Table 8. Calibration Constants Requested by Function **Setup_plot()**

Variable name	Description for Pitch Testing	Description for Yaw Testing	Description for Roll Testing
kspring	spring constant of the balance	spring constant of the balance	torsional spring constant of the balance
inertia	pitch inertia of the model	yaw inertia of the model	roll inertia of the model
ctare	average of values of $T_I/\omega\Theta$ at resonance	average of values of $T_I/\omega\Psi$ at resonance	average of values of $T_I/\omega\Phi$ at resonance
amr	normal force produced by the mechanical spring	rolling moment produced by yaw displacement	yawing moment produced by roll displacement
bmr	mass moment of the model	negative of the product of model inertia	negative of the product of model inertia
tare	average of values of $M_I/\omega\Theta$ at resonance	average of values of $M_I/\omega\Psi$ at resonance	average of values of $M_I/\omega\Phi$ at resonance

Table 9. Variables Affected by Function **Startover()**

Variable and Action	Comment
tmp[0][0] = 0	These variables are used to store values passed up from the DSP boards. A zero value prevents use of initial zeros.
tmp[1][0] = 0	
tmp[0][1] = 0	
tmp[1][1] = 0	
avecount = 0	A zero value here resets the averaging arrays.

Table 10. Definition of Variable *Typecal*

Value	Channel or Type of Calibration
1	Displacement 0 (DSP Board 0 Channel 0)
2	Displacement 1(DSP Board 1 Channel 0)
3	Torque (DSP Board 0 Channel 1)
4	Secondary(DSP Board 1 Channel 1)
5	Automatic Timed Calibration

Table 11. Variables for Main Menu Option 8

Variable	Description
testnum	test number
runnum	tare or wind-on run number
momentarm	calibration moment arm length
calibnum	calibration number
temperature	temperature
pressure	pressure
flow_angle	UPWT model attitude flow angle correction
mach	mach number
inverted	inverted model
sen[0]	displacement sensitivity
sen[1]	torque sensitivity
sen[2]	secondary sensitivity

Table 12. Wind-on Symbols and Plot Codes

wosymbolc		*woplotcode*	
Value	Plot Symbol	Value	Type of Test
1	*	1 or 2	Pitch
2	#	3 or 4	Yaw
3	X	5 or 6	Roll

Lists

List 1. Function Listings by Source Code File for **DSP.EXE**

XMA.C

main()
ac_calibrate()

XAG.C

aoa()
autozero_read()
caldatacode()
calibrate()
dynstat()
fftcontrol()
filestore()
findaverage()
gaininterp()
getdatacode()
getindepvar()
gnormalize()

XHP.C

help()
keycommand()
leastsq()
newx()
newy()
output()
parameters()
plot()
ppcpk()
printfile()

XRS.C

readaoa()
read_gain()
realimag()
refresh()
scale()
setnew()
setsen()
setup_plot()
startover()
switchboard()

XTZ.C

takecaldat()†
tareplot()
takedata()
taredata()
testcond()
windonplot()

List 2. Key Function Assignments

Real-time operational mode utilizes special keys which are listed below. When depressed, they cause program flow to jump to the functions listed.

Key	Function(s) executed
F1	**keycommand()** **help()**
F2	**keycommand()**
F3	**keycommand()**
F4	**keycommand()** **gnormalize()**
F5	**keycommand()** **caldatacode()**
F6	**keycommand()** **setsen()**
F7	**keycommand()** **taredata()**
F8	**keycommand()** **filestore()**
F9	**keycommand()** **aoa()**
F10	**keycommand()** **takedata()** { **getindepvar()** } { **leastsq()** } { **tareplot()** } { **windonplot()** }
F11	(Not used)
F12	**keycommand()** **printfile()**
A	**keycommand()** (Enables the user to select the “active” amplifier)
B	**keycommand()**

(Enables the user to turn strain gage bridge power on or off.)
getdatacode()

C **keycommand()**
ac_calibrate()

D **keycommand()**
(Decrements the currently active amplifier's gain setting)
read_gain()

G **keycommand()**
read_gain()
refresh()

I **keycommand()**
(Increments the currently active amplifier's gain setting.)
read_gain()

N **keycommand()**
(Controls relays inside the signal conditioners to connect or disconnect the input to or from the output.)

S **keycommand()**
(Controls relays to connect shunt resistor pairs across user selectable strain gage bridge arms.)
getdatacode()

U **keycommand()**
(Disconnects all shunt resistor pairs from the strain gage bridge circuits.)
getdatacode()

X **keycommand()**
(Controls the offset circuitry inside the signal conditioners to add 0 V to the input from the strain gage bridge.)
autozero_read()

V **keycommand()**
(Controls the offset circuitry inside the signal conditioners to add a computed offset to the input from the strain gage bridge.)
autozero_read()

Z **keycommand()**
(Controls the offset circuitry inside the signal conditioners to add a self-computed offset to the input from the strain gage bridge.)
autozero_read()

ESC **keycommand()**
startover()

List 3. Necessary Files for Run-Time Execution

The following PC configuration files must be located within the same sub-directory as the **DSP.EXE** file for proper execution.

calival.dat
dconfig.dat
dspnac.mat
gphsplin.mat
sens.dat
testval.dat

The following DSP microcomputer executable files must also be located in the same directory as the **DSP.EXE** file.

dynstat
dynstat2
fft
fft2
search
search2
search12
serch122

Figures

Figure 1. Program Flow by Function

main()	Program flow starts with the function "main."
	Hardware and software initialization occurs here.
setnew()	Operating mode configuration selected here.
{ **testcond()** }	Prompts user for wind tunnel test conditions.
parameters()	Calculates various aerodynamic parameters.
{ **filestore()** }	Prompts user for data storage file name.
{ **setup_plot()** }	Prompt user for scaling parameters for tare and wind-on plots, draw axes.
{ **aoa()** }	Display measured angle of attack.
readaoa()	Measures the angle of attack.
findaverage()	Computes signal statistics.
{ **printfile()** }	Send test results to printer for hardcopy.
{ **parameters()** }	Recalculate various aerodynamic parameters.
{ **plot()** }	Main loop starts here.
scale()	Scale plots for on-screen display.
newx()	Compute abscissa pixel value for input data.
newy()	Compute ordinate pixel value for input data.
{ **dynstat()** }	Synchronous demodulation algorithm implemented here.
refresh()	Refresh on-screen character display.
gaininterp()	Compute gain correction for frequency.
findaverage()+	Computes signal statistics.
readaoa()	Measures the angle of attack.
output()	Display numeric results on-screen.
switchboard()	Select PC access of the other DSP microcomputer.
{ **calibrate()** }	AC calibration routines here.
gaininterp()	Compute gain correction for frequency.
findaverage()+	Computes signal statistics.
realimag()	Compute real and imaginary components with respect to displacement.
switchboard()	Select PC access of the other DSP microcomputer.
{ **fftcontrol()** }	FFT algorithm implemented here.
refresh()	Refresh on-screen character display.
gaininterp()	Compute gain correction for frequency.
findaverage()+	Computes signal statistics.
realimag()	Compute real and imaginary components with respect to displacement.
ppcpk()	Compute extra coefficients.
readaoa()	Measures the angle of attack.
output()	Display numeric results on-screen.
switchboard()	Main loop ends here.
keycommand()	User input here controls program flow.

+ This function is called several times.

Figure 2. PC Directory Tree

```
C:\──┬─ DATA
     ├─ QB45 ──────┬─ EXAMPLES
     │             └─ ADVR_EX
     ├─ TEMPQB
     ├─ BIN
     ├─ CPBKUP
     ├─ CTEST
     ├─ DAS20
     ├─ DOS
     ├─ DSP ─────┬─ ALT
     │           └─ TEMP
     ├─ DSP32SL ───────┬─ BIN
     │                 ├─ INCLUDE
     │                 └─ LIB
     ├─ DSP_IRD ───────┬─ LIB
     │                 ├─ BIN
     │                 ├─ INCLUDE
     │                 └─ DSP3UTIL
     ├─ FASTBACK
     │ ─ INCLUDE ─────── SYS
     ├─ LIB
     ├─ SOURCE ────────┬─ DOC
     │                 ├─ STARTUP ─────── DOS
     │                 ├─ ME ──────────── INI
     │                 └─ DSP
     ├─ UTIL
     ├─ ROOT
     └─ RUNTIME
```