CIMAG2

THE COMPUTER PROGRAM
TO GENERATE COLOR IMAGES

G. Dural
S. Smithberger
J.D. Young

The Ohio State University
ElectroScience Laboratory
Department of Electrical Engineering
Columbus, Ohio 43212

Technical Report 718048-7
Contract No. N00014-86-K-0202
November 1987

Department of the Navy
Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217
NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.
This report describes a user interactive FORTRAN program, CIMAG2, to be used to produce color images using either measured or calculated scattered field data. The program provides the processing of either the frequency or time domain data, and produces a 2-D image of the target of interest. Color plots of the images are done by another program called CLRPL which can also be accessed while running CIMAG2. The report includes the user's and programming manuals, a listing of the commands in the 'HELP' library and all the FORTRAN subroutines.
# TABLE OF CONTENTS

I. INTRODUCTION ............................................. 1

II. CIMAG2 USER'S MANUAL .................................. 2

2.1 PUTTING THE DATA TOGETHER .............................. 3

2.1.1 Appending Data Files ................................. 3
2.1.2 Processing Routines Now in Use ....................... 6

2.1.2.1 RDPL Frequency Domain Read ....................... 6
2.1.2.2 REA Time Domain Read ............................... 8
2.1.2.3 FTREA or FFREA ................................. 8
2.1.2.4 RDFL Frequency Domain Read ....................... 6
2.1.2.5 REA Time Domain Read ............................... 8
2.1.2.6 FTREA or FFREA ................................. 8
2.1.2.7 RDFL Frequency Domain Read ....................... 6
2.1.2.8 REA Time Domain Read ............................... 8

2.1.3 Summary of Data Processing ........................... 11

2.2 PROCEDURE DEFINITIONS .................................. 12

2.2.1 What are Procedure Definitions? ....................... 12
2.2.2 How to User the PROC Command ........................ 12

2.3 CREATING AN IMAGE ........................................ 16

2.3.1 Reading In the Data ................................... 16
2.3.2 The CMI Command ..................................... 17
2.3.3 The IMG Command ..................................... 17
2.3.4 Output the Image ..................................... 19

2.4 THE LOG AND FILE COMMANDS ............................. 19

2.4.1 Creating the History of an Image ..................... 19
2.4.2 Regenerating Results (the File Command) ............ 20

2.5 QUICK REFERENCE FOR SOME COMMANDS OF GENERAL USE . 21

---

Justification

Distribution/ Availability Codes

Dist Special

iii
III. CIMAG2 PROGRAMMER’S MANUAL ........................................ 22
   3.1 Linking CIMAG2 .................................................. 22
   3.2 Process Control Structure .................................... 23
   3.3 Installing a New Routine .................................... 25
   3.4 Listing of the Linking Commands ......................... 28

REFERENCES .................................................................... 29

APPENDICES

A  CIMAG2 HELP LIBRARY ................................................. 30
B  THE COLOR IMAGING PROGRAM ‘CIMAG2’ ....................... 43
C  CIMAG2 LINKING SUBROUTINES .................................. 85
D  COLOR IMAGING PROGRAM CLRPL ................................. 97
I. INTRODUCTION

CIMAG2 is a user interactive FORTRAN program which can be used to produce the color images of the targets using either measured or simulated scattered field data. The program is also capable of processing either frequency or time domain data prior to the image processing.

The original version of the program is written by Dr. J. Young and the new version is written and modified by S. Smithberger and G. Dural. An access for the 'file read' routines of the program 'FTRAN' [1] is provided for the users who already processed their data by FTRAN.

Color images are displayed on a Tektronix 429 color CRT display and hard copy unit via the computer program CLRPL (Appendix D). CIMAG2 accesses the program CLRPL via the 'DCL commands' facility of the program.

Theory related to the imaging procedure is described in detail in another report entitled "Polarimetric ISAR Imaging Using Either Measured or Calculated Transient Signatures" [2].

Chapter II contains the User's Manual of the program. The Programmer's Manual is included in Chapter III. Chapter IV covers the conclusions. Program listings and a list for the 'HELP' library are contained in the appendices.
II. CIMAG2 USER’S MANUAL

This manual shows the user how to use the CIMAG2 program. It can be read in bits and pieces as the user finds a need for information but it is recommended that the user take the time to sit down and read all of the information given here. CIMAG2 is a very sophisticated program into which a lot of automation has been installed. The basic routines can be used to get perfectly usable image, but the user will be able to handle more data more efficiently if he takes the time to learn the automatic features of the program. Not only will he save time in generating the images but he will also find it easier to keep track of what he has done. It is also recommended that the user read this manual while sitting in front of a terminal in order to get a better grasp and feel for what is going on. There are many examples of command sequences in this manual which can be used for illustrative purposes.

This manual stresses processes in which a series of commands have to be used rather than the individual commands themselves. If the user needs more information about the commands available, he can do the following:

- get into the program by typing
  $RUN user2:[DURAL.CIMAG2]CIMAG2

- type the command
  $HELP

At this point the computer will list a menu of commands used by the CIMAG2 program. The user simply types the command that he is interested in and the computer will give him the information that he seeks. When he is finished he simply keeps hitting the <RETURN> until he gets back
to the ‘<’ prompt. This command is most useful when one is in the middle of doing something and can’t remember what a command is called or can’t remember what a certain command does.

2.1 PUTTING THE DATA TOGETHER

2.1.1 Appending Data Files

Some of the data used by the CIMAG2 program will come in chunks. In other words, sometimes the data for a given look angle couldn’t be taken all at once so it was necessary to make more than one data file in order to get all the available data. If we want to use all of the data that is available or at least more than one data file for a given look angle then we will have to append the data files together. This seems simple enough since a given element of the array is assigned a certain frequency. However, many times the files may overlap. One file may contain data from 1.5-6.5 GHz and the other may contain data from 6-12 GHz. When we put the two files together we don’t want the overlapping part of the files to add up so we will have to use what is called the FGT command which puts a trapezoidal gate around the information that we are trying to get to and in effect chops off the information that we don’t want. After the files have been properly gated then we will use the COM (combine) command to put the files together. The following is a general procedure for putting two files together. The text following the exclamation points are comments to inform the reader what is going on. The exclamation marks and comments are not to be typed into the program.
A frequency domain read command
This command will be covered in the following sections.
The store in buffer command

A good way of deciding which elements of the array should be modified is through the use of the TYP command. This command will list any range of the array elements and their respective values.

Put the contents of buffer 1 in the main buffer.

This is the gating command

The array element where the cutoff begins.

This is quite a bit since we want a sharp cutoff.

This element is in the region that is never used close to DC.

A very slow roll-off

It is stored in buffer 3 in so that if an error was made we can try again.

We will use the combine command to put the two files together but the files have to be converted to the time domain before we can do that.

Inverse Fast Fourier transform does this conversion.
COM
STORAGE BUFFER NUMBER(USE 0 TO FINISH): 4  ! Combine the two
MULTIPLIER = 1.0  ! files.
STORAGE BUFFER NUMBER(USE 0 TO FINISH): 5
MULTIPLIER = 1.0
STORAGE BUFFER NUMBER(USE 0 TO FINISH): 0  ! A "0" means we're
! finished.
! At this point it is recommended
! to plot the waveform and see
! whether it is continuous. We
! don't want a spike or null to
! occur at the point we combine
! two files.

SB6

! At this point we should check
! the frequency domain again to
! make sure we did the gating
! right.
! If data overlapped then there
! will be a spike where the two
! files were joined. If there
! was a hole left in the data
! then a null will appear where
! the two files were joined.

RB6
FFT

! Fast Fourier Transform converts
! the time domain back to the
! frequency domain.

SB7

! This command is pretty well
! self explanatory.

RB6
WRI

...
2.1.2 Processing Routines Now In Use

These are the processes that are currently being used on the data before an image is generated.

2.1.2.1 Read Commands

There are four read commands of the two types of reads to be performed by this program. The data that is taken from the radar range is stored in a frequency domain format. This is the raw data. For imaging purposes we will be using the (filtered) time domain waveform. CIMAG2 can only store time domain waveforms. Most of the time the user will process a frequency domain waveform, multiply it with frequency transform it to the time domain, and then store it. Later we will come back and read in several of these time domain waveforms and use them to make an image. With this procedure in mind we will now explain the four read commands -- two of which (FTREA, FFREA) are used to read the files processed by FTRAN [1].

RDFL Frequency Domain Read

The RDFL command was designed to read in the calibrated frequency domain files from the ESL database. The following are questions the that the program will ask and what they mean to the user:

FREQUENCY SAMPLING(1) OR .1KL SAMPLING(0)?
FREQUENCY INCREMENT IN MHZ
If the user chooses to use frequency sampling then the program will ask for the frequency increment to be used for each sample.

INPUT MAJOR AXIS DIMENSION IN INCHES

If the user chooses to use .1KL sampling then CIMAG2 will ask this question.

SELECT THE TYPE OF INTERPOLATION
INPUT 0 --> TWO-POINT INTERPOLATION ; NO SMOOTHING
INPUT 1 --> INTERPOLATION AND SMOOTHING USING A COSINE WINDOW

Most of the time there will be no need for smoothing since the user will more than likely be using data that has already been calibrated. Usually the data is smoothed in the calibration phase but if the data has not been sufficiently smoothed, then the facilities are here to do more smoothing.

ASSUMED INPUT AMPLITUDE IS IN DB/SQUARE CM
NORMALIZE TO: SQ CM(1), SQ M(2), PI*L*L/4(0)?

TYPE DATA FILE NAME

This is obviously the file name of the data. If the data file is not in the current directory be sure to fully specify the file name. The program will then type out the header of the file so that the user can be sure he has the right file.
REA Time Domain Read

The REA command is a much simpler command. It simply asks for the file name and then types out the header. This command is only intended to read files that were output from the CIMAG2 program therefore it has been taken for granted that the file was stored in whatever form it was needed. However, the file can be processed more after it has been read in.

FTREA or FFREA

These are the time and Frequency domain read commands for the files processed by the program FTRAN [1]. Since FTRAN is capable of reading either 750 or 11/23 type data, the user must be careful about the format of the data file. Default is 11/23. The FTY command can be used to switch from one format to another.

FTY

To control the format of the data file the user should type the command FTY. The program then will ask about the file type. Enter T for 750, and F for 11/23 type. Default is 11/23 type when the program is started unless no FTY command is used then.

2.1.2.2 IFF Inverse Fourier Transform

The INVERSE FOURIER TRANSFORM command converts a data file from the frequency domain to the time domain. This command
must be performed before the file can be used for an image and before the file can be output using the WRI command.

2.1.2.3 ROT Rotate

The ROTATE command takes all the elements of a time domain file and moves them either in the positive direction or the negative direction. When an Inverse Fourier Transform is used to convert a waveform from the frequency domain to the time domain a waveform is created which repeats throughout time. The 4096 element array is in effect a time window which shows one complete iteration of this waveform. The main pulse of a time domain waveform may occur around the zero time position. When this happens part of the pulse will be plotted at the beginning of the plot and part of the pulse will be plotted at the end of the plot. Using this command we can move the pulse to some other part of the plot (usually to element 1024) so that the entire pulse can be plotted in one area of the plot.

2.1.2.4 WND Windowing

The frequency domain files that we are using for data are band limited. On either side of the valid data that lies within this bandwidth is a null value. This creates a very sharp change in the amplitude of the waveform where the data starts and ends. The technique of Fourier Transform assumes that the frequency domain waveform is continuous. The sharp changes in the data file are very discontinuous. Thus when these discontinuities are
transformed into the time domain they cause an oscillation of the time domain waveform. The WINDOW command reduces this problem by convolving the frequency domain waveform with a cosine. This greatly reduces the change in amplitude that occurs at the beginning and the end of the valid data thereby reducing the oscillation that results in the time domain.

2.1.2.5 DCV Downconversion

Since the bandwidth of the data that we are using lies in a region well above DC, many of the resulting time domain waveforms take the appearance of a modulated waveform much like that received from a radio station. To get rid of this effect we are currently doing just what the radio receivers do: move the center of the bandwidth to DC. That is what the DOWNCONVERT command does. It respond with the number that represents the middle of the bandwidth.

eg. :
If we are using a bandwidth of 1-12 GHz, then the center of the bandwidth is 6.5 GHz. \(((12-1)/2 +1)\) This corresponds to element 650.

2.1.2.6 MJV Multiply by jw

Multiplying by \(jw\) in the frequency domain is the same as differentiation in the time domain. Another way of thinking of it is to say that it will shift the phase of the waveform ninety
degrees and act as a high pass filter. One can see that the higher the frequency the higher the amplitude.

2.1.2.7 MVR Multiply by v

This command is used as a high pass filter. The higher the frequency is the the bigger the multiplying factor will be. It is a required step in data processing with the algorithm described in [2].

2.1.2.8 WRI Write

The WRITE command is the only way to output a file. Many times after the user has processed a file he will want to save it. Then when he wants to use it again he simply reads it in and the file can be used in a pre-processed form. This command can only write out time domain files so somewhere along the line the file will have had to been transformed using the IFF command. If the user wishes to save the file and then later when he reads it in perform some more processing in the frequency domain, he simply reads it in in the time domain and then performs a FFT command which will convert it back to the frequency domain.

2.1.3 Summary Of Data Processing

Here is the sequence of commands that we are currently using to process the data:
1 RDFL
   - frequency sampling every ten degrees
     (Different sampling intervals can be used to scale the image.)
   - no smoothing
   - normalized to square meters

2 WND
   - use a Hanning Window

3 MWR

4 DCV

5 IFF

6 ROT  (Optional)
   - usually by 1024

7 WRI

2.2 PROCEDURE DEFINITIONS

2.2.1 What Are Procedure Definitions?

Most of the time when an image is made many data files will be needed. This means that many data files will have to be processed using the same command sequences with the same parameters. This is time consuming, redundant, costly, and bothersome. CIHAG2 has a way to get around this problem. A procedure definition is a series of commands that are to be performed on a number of data files. It will take a list of input files and a list of output files, process each file and then put the results in the corresponding output file. This allows the user to go through the procedure once and then let the computer do all the work.
2.2.2 How to Use the PROC Command

The PROC command is a very powerful command but the user must take great care in using it. It is a good idea to go through the first data file the regular way to make sure the method of data processing gives the results that you are looking for. Then you can go ahead and define a procedure with an input list and an output list.

COMPARE THE RESULTS WITH THE FILE MADE THE REGULAR WAY TO MAKE SURE THE PROCEDURE IS DOING WHAT YOU THINK IT IS DOING!!

It is very possible to think you have defined one thing when in fact you have defined something else. This may take some time but not as much time as processing all the files manually or redoing an image because you have used data that is trash.

Let's say that you have already done the test case and you know the exact sequence of commands that you want to perform on each data file. This is the sequence of events that will occur when you define the procedure:
Do you have a procedure definition file for this process already? (Y or N)

! If you answer this question with a Y then you will be asked for a filename. If you answer N then you will be given the following prompt.

Enter the process using regular commands and NAME.DAT for a filename. For the new filename use NEWNAME.DAT. When finished defining the process use the command DONE.
(Warning: a filename must be listed for each time it is used.)

! At this point the user will type in something that looks like the following. Notice how the filenames NAME.DAT and NEWNAME.DAT have been used. The lines that the user input have been marked with a '*'.

<>RDPL

FREQUENCY SAMPLING(1) OR .1KL SAMPLING(0)?
  * 1
  FREQUENCY INCREMENT IN MHZ
  * 10.
  SELECT THE TYPE OF INTERPOLATION
  INPUT 0 --> TWO-POINT INTERPOLATION ; NO SMOOTHING
  INPUT 1 --> INTERPOLATION AND SMOOTHING USING A COSINE WINDOW
  * 0
  ASSUMED INPUT AMPLITUDE IS IN DB/SQUARE CM
  NORMALIZE TO: SQ CM(1), SQ M(2), PI*L*L/4(0)?
  * 2
  TYPE DATA FILE NAME
  * NAME.DAT
  Dummy file for procedure definition
  a 6 in. sphere
  NL1200 FF= 1000 IN= 9 frequency domain
  *
  <>WND
  INPUT HARMONICS; START, END, TYPE OF WINDOW
  TYPE: 0=HANNING, 1=HAMMING, 2=GAUSSIAN, 0,N,M,=TEST
  * 100,1200,0
  *
  <>DCV
  INPUT THE HARMONIC NUMBER TO BE MOVED TO DC
* 650
* <>MWR
* <>IFF
*<>ROT
* ROTATE BY INCREMENTS OF:
* 1024
* <>WRI
* FILE NAME ?
* NEWNAME.DAT
* DO YOU WANT TO CHANGE THE FILE HEADER Y=1, N=0
* 0
* <>DONE
* Do you wish to save this procedure definition?(Y or N)
* Y
* Filename:
* PROC.DEF
* Do you have a data list file?(Y or N)
* N
* Enter the list of data files, following each with <RETURN>. When finished type the word DONE.
* DATA1.DAT
* DATA2.DAT
* DATA3.DAT
* DONE
* Do you wish to save this data list?(Y or N)
* Y
* Filename:
* INPUT.DAT
* Is there an output filename list?(Y or N)
* N
* Enter a list of the output file names in the order they are to be used. When finished type DONE.
* OUT1.DAT
* OUT2.DAT
* OUT3.DAT
* DONE
* Do you wish to save this list?(Y or N)
* Y
* Filename:
* OUTPUT.DAT
* Your data is being processed.

At this point the computer processes the data according to what you have told it to do. CHECK THE OUTPUT TO MAKE SURE IT IS WHAT YOU WANTED!!
2.3 CREATING AN IMAGE

The main objective of this program is to produce an image of the target on a computer monitor. The idea is that we can get this image to the point where we can identify the target with the image. This is the sequence of events used to form an image on the screen:

- read in and store all the data files needed for the image in the buffers
- CMI command
- IMG command
- $RUN USER2:[DURAL.IMAGE]CLRPL

2.3.1 Reading In the Data

Assuming that the data has been processed using some method based on methods presented earlier in this manual filtered and stored in the time domain, we can use a REA (or FTREA command if data are not processed by CIMAG2) to read in each data file and then we can use the SBF command to store all the data files in separate buffers. The SHO_BUF command can be used to give a listing of all the buffers and their contents. However the user will have to remember the polarization, buffer#, look angle, and center element for each data file so he will probably want to keep track of these things on a sheet of paper while going through this process.
2.3.2 The CMI Command

The CMI command sets up the data structures for the rest of the imaging processes. First the program will ask the user for the number of files in a given polarization. It will look like this:

NUMBER OF VV TIME DOMAIN WAVEFORMS TO BE USED?

VV stands for vertical polarization. The user should answer with an integer. Then it will ask the user these questions for each of the files for that given polarization:

BUFFER NUMBER FOR VV FILE #

LOOK ANGLE IN DEGREES FOR FILE #

CENTER ELEMENT NUMBER FOR FILE #

The center element of the file will be zero unless the user has used the rotate command on the data. After all the information for a given polarization has been accumulated, then CIMAG2 will repeat the sequence for HH (horizontal) and HV (cross polarization).

2.3.3 The IMG Command

The IMG combines all the data down into a hundred by hundred matrix. These are the questions that will be asked:
SIZE OF THE IMAGE, (1 TO 4096) = ?

Generally a good value for this is around 300. The user shouldn't really use any bigger value than this since the current resolution is only hundred by hundred. When a higher resolution device is connected up to this software, it might be better to get a larger window.

LOOK ANGLE OF THE IMAGE, DEGREES?

The computer is able to spin the image that is on the screen so that it may be easier to see certain things but this doesn't really have any real affect on the image.

POLARIZATION OF THE IMAGE, (1 = VV, 2 = HH, 3 = HV, 4 = ALL)

This is self explanatory.

After this command has finished executing then the final array is ready to be imaged. After the image is generated the computer asks,

DO YOU WANT TO STORE THE IMAGE? Y = 1, N = 0

If the answer is "1" then the computer asks for the name of the data file to store the image and the frequency increment for the frequency domain signal (usually 10 MHz) which is used for calculating the time axes in the plot.
2.3.4 OUTPUT THE IMAGE

In order to output the image the user should enter

$RUN USER2:[DURAL.IMAGE]CLRPL

2.4 THE LOG & FILE COMMANDS

2.4.1 Creating the History of An Image

The sequence of events necessary to create a given image can be long and complex. Many times it is advantageous to keep track of exactly what has been done to a file or an image. This can be especially helpful if an error occurs or you want to reevaluate a couple of different procedures. The history of the procedure can also be used to regenerate results. This is especially useful in the case of images. The image can be reproduced much faster using automated techniques rather than manually entering all the commands that are necessary. The final reason for generation of a history is that it is easy to do and if the user can take advantage of a history with little effort then why not do it?

To make the history is a simple matter. The user simply types the command LOG before he types in the procedure. CIMAG2 will prompt for a filename in which to dump the history. Then the user uses the program the same way he would if there were no history being kept. When he is finished with the procedure he types the command STO_LOG.

This closes the history file given by the LOG command.
2.4.2 Regenerating Results (the FILE command)

As was mentioned in section 5.1 the history of a procedure can be used to regenerate the results of that procedure. To do this the user will use the FILE command. The user then types in the name of the history and the computer will take over and perform the procedure stored in the history file. When the computer is finished executing this file it will give the user this message:

CONTROL HAS RETURNED TO THE TERMINAL

At this point the user is free again to do as he wishes.

If the user would like to see the normal prompts that the computer outputs to the screen for each command so that he can follow the execution of the history file he can use the ECHO command before he uses the FILE command and the computer will output this information.
2.5 QUICK REFERENCE FOR SOME COMMANDS OF GENERAL USE

CLR - clears the main buffer
CLR_BUF - clears all the buffers
CMI - initialize buffers for imaging
COM - combine command adds time domain waveforms together
DCV - downconversion
DEF - sets the default directory for input files
DEL - deletes a file from the directory
ECHO - sets the echo for the FILE command
FFREA - FTRAN Read Command (Frequency)
FFT - fast fourier transform
FGT - trapezoidal gate
FILE - executes a history file
FTREA - FTRAN Read Command (Time)
FTY - File type for the input format
IFF - inverse fourier transform
IMG - sets up the overall parameters for image
LOG - creates a log or history file
MJW - multiply by jw
NO_ECHO - stops the echo for the FILE command
PLO - plot command plots a data file on one of the plotting devices
PROC - procedure definition
RBF - read buffer
RDFL - read frequency domain file
REA - read time domain file
RLB - relabel the file header
ROT - rotate a time domain file
SBF - store in buffer
SHO_BUF - sho the contents of all the buffers
STO_LOG - stop the log or history
TYP - type the value of all the elements of a data file
WND - window
WRI - write time domain data file
III. CIMAG2 PROGRAMMER'S MANUAL

CIMAG2 is a very large, non-trivial program. It also has some distinct methods of control. It is the aim of this manual to familiarize the user with the methods and data structures used in CIMAG2. If more information is needed the Vax Fortran Manual will have more detailed information on file handling. The purpose of this manual is to inform the reader of names and structures unique to this program. It is suggested that the programmer read the CIMAG2 User's Manual first. This will familiarize the programmer with the commands that have been implemented and give him background for the discussions that follow.

3.1 Linking CIMAG2

CIMAG2 is a very complicated program. It is divided up into many smaller programs and data files. The data files not only include data files created by users but also data files for such structures as error messages, help libraries etc. For some of these files it is necessary to link them with the program but others must be linked. As a result the linking for CIMAG2 has become rather cumbersome. To get around this the linking for the program has been placed in the command file USER2:[DURAL.CIMAG2]CIMAG2.COM. This means that there are only two commands necessary to convert a new version of CIMAG2.FOR into an execution file:
The first command compiles the main program and creates a listing file. The second command executes the linking command file. A listing of CIMAG2.COM is given in Section (3-4). It is recommended that programmers first make changes to the NEWMAG.FOR file first. Then after the new routine has been debugged and tested NEWMAG.FOR can be copied over into CIMAG2.FOR. This means that a working version of the program will always be in CIMAG2. NEWMAG can be linked the same way that CIMAG2 is through the use of user2:[DURAL.CIMAG2]NEWMAG.COM. This command file links NEWMAG in the same a way shown below.

$FOR/LIST NEWMAG
$@NEWMAG

A listing of NEWMAG.COM can also be found in Section (3-4).

3.2 Process Control Structure

The Vax treats files and devices the same way. In other words the input from the terminal looks like a file to the Vax. This fact been used in a few of the most powerful commands in CIMAG2; namely the FILE, PROC, LOG, and STO_LOG commands. We can transfer control from the terminal to an input file and back again. This gives the user the ability to use processes that
have already been defined and define new processes himself. Here are the basics of how this process control is implemented in CIMAG2.

All devices and files that are to be used are assigned what is called Logical Unit Numbers (LU#). Each file or device will be referenced by this number in read and write statements. (When a programmer writes new software for CIMAG2 he should not use any ACCEPT or TYPE statements in his code. These two statements will defeat the purpose of this whole method of control.) Here are the three basic control structures:

```plaintext
COM UNIT ,  ! LU# for the command input
IOUT ,      ! LU# for the program output
LOG_UNIT ,  ! LU# for the log file
```

COM_UNIT and IOUT can be set to either the terminal or a file. The LOG_UNIT can either be set to a file or to the null device. Using these three structures the I/O for a given routine would be written like this:

```plaintext
READ (COM_UNIT,*) X
WRITE (LOG_UNIT,*) X
.
.
WRITE (IOUT,*) X
```

Notice how the input was immediately written to the LOG_UNIT. All input should be done this way. This allows the routine to be used in procedure definitions and logging files. The format "*" was used in this example but any numbered format
may be used just as in any Fortran programming. For these structures to be used in this way they must be assigned values that correspond to the various devices and files. Here are some variable names that are assigned:

- TERM_UNIT, ! LU# for the terminal
- FILE_UNIT, ! LU# for the command file
- NULL_UNIT, ! LU# of null device
- STO_UNIT ! LU# for buffer storage

Using these variables this is the way that the structures are initialized:

```
COM_UNIT = TERM_UNIT
IOUT = TERM_UNIT
LOG_UNIT = NULL_UNIT
```

Initially the input is coming from the terminal, the output is going to the terminal, and we are logging to the null unit which means we really aren't logging anywhere.

### 3.3 Installing a New Routine

In order for a routine to be used by this program it must be written in a particular format. This format is outlined in section 3.2. PROCESS CONTROL.

Once the routine is properly formatted it will probably need to change values of some of the variables included in common blocks. The following is a list of the files in which the common blocks used by this program are stored:

- MAGCMN.FOR Contains all the working variables and arrays of the program.
MAGCMN2.FOR  The program control variables.
HEADER.CMN  All the variables that define the data header fields.
FTRN.FOR  Common variables used in FTRAN read routine.

You may also want to include something in the helps. All the helps are in a file called:

CIMAG2.HLP

To convert this into a library file, enter:

LIBRARY/CREATE/HELP CIMAG2.HLB CIMAG2.HLP

CIMAG2.HLB

(Content of the existing HELP file is listed in Appendix A).

There is another utility this program uses called the message utility. If you wish to use this utility to generate error messages, the existing message file is:

CIMAGMSG.MSG

When the utility is ran the output will be put into a file named:

CIMAGMSG.OBJ
More details on this utility are included in the Vax manuals.
3.4 Listing of the Linking Command Files

This command procedure links the modules for the CIMAG2 program.

```
$ !
$ ! LINK\NOTRACE
  CIMAG2       !Main program
  INTER,      !Frequency Domain Read
  RDFLE,      !Frequency Domain Read (called by INTER)
  FORT,       !Fourier Transform
  DDMPB,      !BSC Read
  CIMAGMSG,   !Program Error Messages
  'GRPIILIB',  !Plotting
  'PLOTOLD2'  !Plotting
$!
$ EXIT
```

This command procedure links the modules for the NEWMAG program.

```
$ !
$ ! LINK\NOTRACE
  NEWMAG      !Main program
  INTER,      !Frequency Domain Read
  RDFLE,      !Frequency Domain Read (called by INTER)
  FORT,       !Fourier Transform
  DDMPB,      !BSC Read
  CIMAGMSG,   !Program Error Messages
  'GRPIILIB',  !Plotting
  'PLOTOLD2'  !Plotting
$!
$ EXIT
```
REFERENCES

[1] Dominek, A., Personal Communication, The Ohio State University ElectroScience Laboratory, Columbus, Ohio.

APPENDIX A

CIMAG2 HELP LIBRARY

1 DCL Commands

The user may use DCL commands while he is still in the program by simply typing "$" before the command he wants to use. In this way he may use the $DIR command to see the files that are available in a given directory, run a calibration program and then use this new data with the data that he already had in the program, etc...

Warning:
The user will not be able to use the $SET DEFAULT command due to the way in which the DCL commands are enabled. The default will be set to whatever the default was when the user entered the program. This means he will to fully specify directory names if he wishes to use other directories. When in the program however he may use the DEF command to set the default for the program's read statements.

2 SSPAWN

If the user uses the SSPAWN command he can in effect suspend the program and open up a new terminal. From here he can do anything he wants. When he is through he simply logs out and then he will find himself back in the program in the same spot he left.

1 BSC

This routine reads data from a basic scattering code calculation.

1 CHANGE

This command allows the user to change the value of any harmonic in a file. This was originally intended for use in creating test files but the user may find other uses for it.

2 Parameters

The computer will ask:

Which harmonic do you wish to change?
(Answer with an integer between 1 and 4096 representing the number of the harmonic you wish to change.)

Then it will give you some information and then ask you for a new value:
Current Value: [The harmonic number] [The current value]
New Value:
   (Answer with a DECIMAL NUMBER. This is most important.
    If the user forgets to put in the decimal point then
    the program will give erroneous results.)

   It is a good idea to use the TYP command to check the
   file to make sure that the results turn out to be what they
   were intended to be.

1 CREATE
   Creates a blank time domain file in the main buffer.
   The values of all the harmonics will be zero.

2 PARAMETERS
   The program will ask the user for a three line header.
   The user simply types in each line of the header finishing each
   with a carriage return. When all three lines have been typed in
   the program will give the user the command prompt.

1 CLR
   Clears the main buffer by setting all values to zero.

1 CLR_BUF
   Clears all buffers including the main buffer.

1 CMI
   Routine to set up the parameters for a two dimensional
   image.

1 COM
   The combine command calculates any linear combination
   of the data in any number of the storage buffers and puts the
   result into the main buffer.

   IMPORTANT NOTE: TIME domain waveforms are LINEAR, so combine
   is ADDITION or SUBTRACTION. FREQUENCY spectra
   are LOGARITHMIC, so combine is MULTIPLICATION
   or DIVISION of spectra. (Same as convolution
   or deconvolution in the time domain.)

2 Parameters
   It starts by clearing the main buffer. Then it asks:
   STORAGE BUFFER NUMBER(USE 0 TO FINISH):
   (reply with a buffer number n or 0 to finish
   the combination process and CARRIAGE RETURN)
MULTIPLIER =
(reply with a floating point number a and
CARRIAGE RETURN).
Then the main buffer will be:
BUF(0)=a1*BUF(n1)+a2*BUF(n2)+a3*BUF(n3)+........

1 DATA
This command puts the user in the Database program.
This is a separate program designed to find data files using
the header. It has its own HELP command once the user gets
into the program. When the user wishes to return to the CIMAG2
program he simply types EXIT and he will find himself wherever
he left off before he entered the Database.

1 DCV
Routine to downconvert a spectrum to a dual-polarity
envelope. The user chooses the harmonic number to call his
"center frequency" which gets converted to the DC term.
The routine works on a frequency spectrum in the main
buffer and returns the result to the main buffer.

1 DEF
Sets the default directory to a new default for the
program's input read statements. This gives the user the
ability to input data files from anywhere in the computer
without having to fully specify devices and directories every
time.

2 Parameters
It will ask:

DEFAULT?

The user should answer the same way that he would
for the DCL command:

Device:[directory]
eg:
USER1:[STEVIE.747]

1 DEL
Deletes a data file in VAX memory device.

2 Parameters

ENTER THE FILE NAME
(reply with up to 50 characters, CARRIAGE
RETURN)
The differentiate command calculates the derivative of the TIME DOMAIN waveform in main buffer and places the result in main buffer.

Divides the frequency spectrum in main buffer by j*2*pi*f (equivalent to integration in the time domain) and places the result back in main buffer.

For use with a command file. If the user wishes to have the commands displayed as the command file executes them he may do so by using this command. To turn the echo off again he simply uses the NO ECHO command.

Exit Command

This command performs the "fast Fourier Transform" on a 4096 point time domain waveform that resides in the main buffer (buffer 0). It then places the resulting 2048-harmonic spectrum back in the main buffer.

This frequency spectrum gating routine performs a trapezoidal gate on the log amplitude components (0<n<2048) while leaving the phase spectral components (2049<n<4096) unchanged. It modifies the spectrum in main buffer.

Parameters

HARMONIC FOR START OF HIGH FREQUENCY CUTOFF
(reply with integer "a", CARRIAGE RETURN)

HIGH FREQUENCY ROLL-OFF IN DB PER HARMONIC
(reply with a floating point value, CARRIAGE RETURN)

HARMONIC FOR START OF LOW FREQUENCY CUTOFF
(reply with integer "b", CARRIAGE RETURN)

LOW FREQ ROLLOFF IN DB PER HARMONIC
(reply with a floating point value, CARRIAGE RETURN)

The amplitude spectrum in main buffer is unchanged between harmonics a and b. It is attenuated at the rates specified for harmonics above and below those values.
1 FILE

This command gives control of the program to a specified command file. When the new file takes control it puts the old command device on a stack. When the command file is finished it may then return control to the old device or it may give control to another command file by using the FILE command itself. However it must eventually pass control back to the device that gave it control. When the user can again enter commands he will be given the prompt:

CONTROL HAS RETURNED TO THE TERMINAL

2 Command_files

The command files are just what the name implies: a list of commands. The file however must also include everything the user would normally type in. So all of the prompts must be answered such as:

BUFFER#?

The easiest way to do this is through the use of the LOG command.

2 Parameters

FILE?: This is asking for the name of the new command file.

1 FFREA

Reads a data file written by [DOMI.DAT]FTRN. (FTRN REA Command)-Frequency Domain.

1 FRD

Reads a frequency domain data file set in the old standard bands of 1-2, 2-4, 4-8, 8-12 GHz into the main buffer in place of its present contents. See Bill Leeper for further info.

1 FTREA

Reads a data file written by [DOMI.DAT]FTRN. (FTRN REA Command)-Time Domain

1 FTY

File type control for the data processed by FTRAN.

T=750 P=11/23
(Program initially sets to 11/23)

1 GAT

This command performs a trapezoidal gate of the data in the main buffer.
2 Parameters

START GATE OPEN (SAMPLE NO.)
(reply with integer "a", 1≤a≤4096)
START GATE CLOSE AT SAMPLE
(reply with integer "b", b>a, 1≤b≤4096)
RAMP LENGTH IN SAMPLES =
(reply with integer "c")

Then if the main buffer is X(n), and the result to be placed in the main buffer is X'(n):

X'(n) = 0 for n≤a
X'(n) = X(n)*(n-a)/c for a<n≤(a+c)
X'(n) = X(n) for (a+c)<n<b
X'(n) = X(n)*(n-b)/c for b<n≤(b+c)
X'(n) = 0 for (b+c)<n<4096

1 GRID

Creates a time domain file that has a pulse at every given number of nanoseconds. When this file is imaged it effectively gives the user a time scale.

WARNING: Due to the resolution of the system a pulse will usually have to be several harmonics wide, depending on the widow, in order for it to be picked up. It is a good idea to use the TYP command to see how many pulses you should see in a given window to make sure that they were all picked up in the scan.

2 Parameters

How many nanoseconds per division?
(Answer with a DECIMAL number. Make sure the decimal is included.)

Type in header: (Three lines)
(Type in the three header lines ending each with a CARRIAGE RETURN.)

How many harmonics wide should the line be?
(Answer with an integer. Remember this system is set up for continuous wave forms and not pulses so the pulse may have to be several harmonics wide; i.e., I have found 7 harmonics to work best for a window size of 300. As the window gets smaller fewer harmonics will be needed.)
1 IFF

This command performs the inverse of the "fast Fourier Transform" on a 2048-harmonic spectrum in the main buffer and places the resulting 4096-point time domain waveform back in the main buffer.

1 IMG

Routine to form the image for one polarization from time domain waveforms as set up by the CHI command.

1 INT

The integrate command calculates the integral of the TIME DOMAIN waveform in main buffer and places the result in main buffer.

1 LOG

Logs the user input into a command file. This is a good way to build command files for the FILE command. You can stop logging with the STO_LOG command.

2 Parameters

LOG FILE?: Give a name for the file to which the log is to be sent.

2 Command_files

The log command is the easiest way to make command files for the FILE command. It automatically takes all the input from the input device and makes a file out of it.

If the user wishes to put comments in the log file he may do so by using '!' as the first character of a command. The program will ignore it but it will be logged into the log file. This is a very good way of identifying what a given command file does or even what a section of one does.

The only other thing to remember when making command files is that it is a good idea to set the default in the very first line through the use of the DEF command. In this way a given command file can be run regardless of what the default was set at previously.
There are two manuals for the CIMAG2 program:

- CIMAG2 USER'S MANUAL USER.TXT
- CIMAG2 PROGRAMMER'S MANUAL PROG.TXT

The user's manual gives information on the procedures used to create images and manipulate data. The programmer's manual gives information that might be helpful to someone wanting to make changes or additions to the program. To get a copy of either text the user simply needs to use either the NEC command to get a copy from the spinwriter or the PRINT command to get a copy from the printer. This can be done from inside the program or from the monitor. (See DCL_Commands)

\[ \text{e.g.:} \]
\[ \$\text{LASER USER.TXT} \]

1 MJW

Multiplies the frequency spectrum in main buffer by \( j \times 2\pi f \) (equivalent to differentiation in the time domain) and places the result back in main buffer.

1 MWR

Multiplies the frequency domain spectrum in main buffer by \( 2\pi f \) and places the result back in the main buffer.

1 NO_ECHO

Turns off the echo that was enabled by the ECHO command.

1 NOR

This normalize command calculates the mean of the 4096 point waveform in the main buffer and shifts the main buffer waveform so its mean is zero.

1 PIM

This routine plots an isometric view of a single polarization two-dimensional target image on the plot device in isometric form with no shadowing.

1 PLO

The PLOT command allows the user to plot any one of the buffers on one of the plotting devices listed.
2 Parameters

WAVEFORM BUFFER NUMBER?
(reply with the number of the buffer in which the desired waveform is stored)

DO YOU WANT A NEW WINDOW? (1=Y,0=N)
(this allows you to choose what section of the wave you want to look at or you may look at the whole thing. If your answer is 1 then it will ask the for the range of data numbers you want to look at (1≤a≤4096). If your answer is 0 then it will default to the window that you used last.)

DO YOU WANT NEW AXES?
(this allows you to fix the labeling for the graph)

INPUT TITLE FOR PLOT?
(this is the title for the top of the plot.
Type in any title desired.)

Then it gives you a list of devices on which you may output the plot. Just type the number of the device which you wish to use.

1 PROC

This operation allows the user to specify a process to be performed on a group of files. Then it will ask for the list of data files and a list of the output filenames desired. It will then run each of the data files through the defined process and make output files as requested.

2 Procedure

The user is given explicit instructions throughout the procedure. There are only a few points that need to be stressed here:

- The filename NAME.DAT will be used for all input data files. This has to be typed in capital letters.

- The filename NEWNAME.DAT is used for all output file names. It also has to be typed in capital letters.
When making either of the filename lists if a given file is used more than once it has to be listed more than once.

- Filenames are used in the order that they appear in the lists.
- Whenever the user finishes defining a process or a list he/she will type the word "DONE" in capital letters and the procedure will move on to the next phase.

**WARNING:** This is a very powerful command but the user must be very careful when using it so that the output files will have the desired content and not some other content without the users knowledge.

1 **PSM**

Point smooth command fits a cubic curve to points surrounding a small bad region of time waveform or spectrum for the data in main buffer.

2 Parameters

For **TIME** waveforms it asks:
- FIRST BAD POINT INTEGER=
  (reply with the integer for the first point to be replaced, CARRIAGE RETURN)
- LAST BAD POINT INTEGER=
  (reply with the integer for the last point to be replaced. This can be the same as the first point)

For **FREQUENCY** spectrum in main buffer, it asks:
- FIRST BAD HARMONIC
  (reply with a harmonic # between 0 and 2048, CARRIAGE RETURN)
- LAST BAD HARMONIC
  (reply with a second harmonic #, perhaps same as first)

For both cases, the bad section of data is replaced in main buffer with values which join surrounding values with continuous slope.

1 **RBF**

This command reads the contents of a storage buffer into the main buffer.
2 Parameters

BUFFER#: There are 35 buffers. This number is used to specify which one is to be used.

1 RDFL

Reads a modern frequency domain file into the main buffer in place of its present contents. It asks:

INPUT

1 REA

Reads a time domain file in "Jon Young format" from a VAX storage device into the main buffer (destroys what was in main buffer before).

2 Parameters

INPUT FILE ?
(reply with up to 50 characters for a VAX file name)

WAVEFORM NUMBER?
(reply with an integer, since "Jon Young format" files will accept more than one waveform per file; normally 1 is used with those files which have only one waveform in them)

If the name is unrecognized or if any other error happens, the routine aborts and the "ERROR IN COMMAND" message is printed.

1 RLB

The re-label command. It will print out the first line of the old title block and then wait for the user to type in the new line.

OLD TITLE BLOCK:
Whatever the line is

NEW TITLE BLOCK:
The user then simply types in whatever the new line is to be.

1 ROT

Rotates the data in main buffer by "a" increments to the right(+a) or left(-a), with values shifted beyond 0 or 4096 appearing on the other end of the waveform. It asks:

ROTATE BY INCREMENTS OF:
(reply with "a", CARRIAGE RETURN)

1 SBF

This command stores the contents of the main buffer into one of the storage buffers.
2 Parameters
BUFFER#: There are 35 buffers. This number is used to specify which one is to be used.

1 SCT
This smooth cutoff tail routine attaches a shifted cosine amplitude spectrum attenuation function to the spectrum in main buffer to eliminate Gibbs phenomenon in the time waveform.

2 Parameters
INPUT THE STARTING AND END POINT FOR THE FREQUENCY ROLLOFF
(reply with a pair of integers, "a", "b", 0 ≤ a, b, ≤ 4096)
The spectrum is unchanged for n ≤ a. It is -100 db for n ≥ b. And for a < n < b, the spectrum has a cosine rolloff.

1 SHO_BUF
This command shows the contents of all of the buffers that have anything in them. The buffer number is given for each entry. The domain of the file is also given (time or freq). This is the format:

BF#  DOMAIN
DESCRIPTION ...

1 STO_LOG
The stop logging command stops the input from going to the logging file. It assumes that the user was writing to a logging device.

1 TYP
Type a portion of data in a buffer. This data is typed out in numerical form so that the user can see the actual data values of individual points.

2 Parameters
START AT ELEMENT NUMBER:
(reply with an integer from 1 to 4096, CARRIAGE RETURN)
END AT ELEMENT NUMBER:
(reply with a second integer, 1 to 4096, CARRIAGE RETURN)
BUFFER NUMBER(0=MAIN):
(reply with a buffer integer #, 0 to 8, CARRIAGE RETURN)
The specified portion of the specified buffer is then typed out on the CRT screen.
1. **WND**

   This routine attaches a smooth cutoff tail to the spectrum in order to reduce Gibb's Phenomenon.

   **Parameters**

   INPUT HARMONICS;START,END,TYPE OF WINDOW

   TYPE: O-HANNING, I-HAMMING, 2-GAUSSIAN, O,N,M-TEST

   (reply with a pair of integers, "a", "b", 0 ≤ a, b ≤ 4096 and then 0, 1, or 2 depending on the type of rolloff desired)

2. **WRI**

   Writes a time domain waveform in the main buffer to a specified VAX memory device in "Jon Young format". The data in main buffer is not changed.

   **Parameters**

   FILE NAME ?

   (reply with a name up to 50 characters, CARRIAGE RETURN)

3. **YSH**

   Shifts the data in buffer up(+) or down(-) by Y units.

   **Parameters**

   YSHIFT IN UNITS(-2048 ≤ Y ≤ 2048)=

   (reply with the shift value, CARRIAGE RETURN)
APPENDIX B

THE COLOR IMAGING PROGRAM 'CIMA2'

CIMAG2

CHARACTER*80 COMMAND

INCLUDE 'USER2:[DURAL.CIMA2]FTRN.FOR'
INCLUDE 'USER2:[DURAL.CIMA2]MAGCMN.FOR'
INCLUDE 'USER2:[DURAL.CIMA2]MAGCMN2.FOR'
INCLUDE 'USER2:[DURAL.CIMA2]HEADER.CMN'

EQUIVALENCE (HEADER,HEAD(1,1)),(LINE1(1),HEADER(1))
EQUIVALENCE (FLTP,IDS(1)),(HEADER(61),LINE3(1)),(HEADER(31),LINE2(1))

INTEGER*4 FOR_RETCODE
INTEGER*4 FOR_EOF / -1 /
INTEGER*4 RETCODE
INTEGER*4 LIBSGET_LUN

CALL START

RETCODE = LIBSGET_LUN( TERM_UNIT )
OPEN( FILE='.*:', +
        UNIT=TERM_UNIT ,
        STATUS='NEW' )

RETCODE = LIBSGET_LUN( NULL_UNIT )
OPEN( FILE='NL: ', +
        UNIT=NULL_UNIT ,
        STATUS='NEW' )

COM_UNIT=TERM_UNIT
IOUT=TERM_UNIT
LOG_UNIT=NULL_UNIT

DO WHILE (COMMAND .NE. 'EXI')

IF (COM_UNIT .EQ. TERM_UNIT) THEN

IOUT=TERM_UNIT
WRITE( UNIT=IOUT, FMT=5)
ELSE
IF (ECHO) THEN
IOUT=TERM_UNIT
ELSE
IOUT=NULL_UNIT
END IF
END IF
READ ( UNIT=COM_UNIT , FMT=10 , IOSTAT=FOR_RETCODE ) COMMAND

IF (FOR_RETCODE .EQ. FOR_EOF) THEN
CALL BACK
IF (PROC_FLAG) THEN
END IF
CALL PROC3
END IF

43
ELSE
CALL STR$UPCASE(COMMAND, COMMAND)  ! convert to uppercase
C
C not equl to
+  IF ((COMMAND .NE. 'STO_LOG') .AND. COMMAND .NE. 'FILE') .AND. COMMAND .NE. 'DONE') THEN
C run command file
+  WRITE(UNIT=LOG_UNIT, FMT=10) ! log COMMAND

C
IF (COMMAND .EQ. 'BSC') THEN  ! execute the command
CALL BSC
ELSE IF (COMMAND .EQ. 'CARD') THEN
CALL CARDC
ELSE IF (COMMAND .EQ. 'CHANGE') THEN
CALL CHANGE
ELSE IF (COMMAND .EQ. 'CLR') THEN
CALL CLR
ELSE IF (COMMAND .EQ. 'CLR_BUF') THEN
CALL CLRBUF(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'CMI') THEN
CALL CMI
ELSE IF (COMMAND .EQ. 'COM') THEN
CALL COM(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'CREATE') THEN
CALL CREATE(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'DATA') THEN
CALL LIB$SPAWN('RUN USER2:' //''DURALJDBMANAGER') ! use the database
ELSE IF (COMMAND .EQ. 'DCV') THEN
CALL DCV
ELSE IF (COMMAND .EQ. 'DEF') THEN
CALL DEFFER  ! set def command
ELSE IF (COMMAND .EQ. 'DEL') THEN
CALL DEL
ELSE IF (COMMAND .EQ. 'DIF') THEN
CALL DIF
ELSE IF (COMMAND .EQ. 'DJW') THEN
CALL DJW
ELSE IF (COMMAND .EQ. 'DONE') THEN
CALL PROC2
ELSE IF (COMMAND .EQ. 'DWR') THEN
CALL DWR
ELSE IF (COMMAND .EQ. 'ECHO') THEN  ! shows command
ECHO=.TRUE. ! being executed.
ELSE IF (COMMAND .EQ. 'FFT') THEN
CALL FFT
ELSE IF (COMMAND .EQ. 'FGT') THEN
CALL FGT
ELSE IF (COMMAND .EQ. 'FILE') THEN
CALL FILE
ELSE IF (COMMAND .EQ. 'FRD') THEN
CALL FRDC
ELSE IF (COMMAND .EQ. 'FTREA') THEN
CALL FTREA
ELSE IF (COMMAND .EQ. 'FFREA') THEN
CALL FFREA
ELSE IF (COMMAND .EQ. 'FTY') THEN
CALL FTY
ELSE IF (COMMAND .EQ. 'GAT') THEN

CALL GAT
ELSE IF (COMMAND .EQ. 'GRID') THEN
CALL GRID(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'HELP') THEN
CALL LIBSSPAWN('HELP/PAGE/LIBRARY=USER2:'
//'[DURAL.CIMAG2]CIMAG2.HLB') ! call help files
ELSE IF (COMMAND .EQ. 'IFF') THEN
CALL IFF
ELSE IF (COMMAND .EQ. 'IMG') THEN
CALL IMG
ELSE IF (COMMAND .EQ. 'INT') THEN
CALL INTEG
ELSE IF (COMMAND .EQ. 'LOG') THEN
CALL LOGGER
ELSE IF (COMMAND .EQ. 'MJW') THEN
CALL MJW
ELSE IF (COMMAND .EQ. 'MWR') THEN
CALL MWR
ELSE IF (COMMAND .EQ. 'NO_ECHO') THEN
ECHO=.FALSE.
ELSE IF (COMMAND .EQ. 'NOR') THEN
CALL NOR
ELSE IF (COMMAND .EQ. 'PIM') THEN
CALL PIM
ELSE IF (COMMAND .EQ. 'PLO') THEN
CALL PLO
ELSE IF (COMMAND .EQ. 'PROC') THEN ! process data
CALL PROC
ELSE IF (COMMAND .EQ. 'PSM') THEN
CALL PSM
ELSE IF (COMMAND .EQ. 'READ') THEN
CALL REA
ELSE IF (COMMAND .EQ. 'RB') THEN
CALL RB
ELSE IF (COMMAND .EQ. 'RDPL') THEN
CALL RDPL(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'REA') THEN
CALL REA(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'RBF') THEN
CALL RBF(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'RB') THEN
CALL RB(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'РB') THEN
CALL RBF(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'ROT') THEN
CALL ROT
ELSE IF (COMMAND .EQ. 'SBF') THEN
CALL SBF(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'SB') THEN
CALL SB(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'SHO_BUF') THEN
CALL SHOBUF
ELSE IF (COMMAND .EQ. 'STO_LOG') THEN
CALL STOLOG
ELSE IF (COMMAND .EQ. 'TYP') THEN
CALL TYP(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'WND') THEN
CALL WND
ELSE IF (COMMAND .EQ. 'WRI') THEN
CALL WRI(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND .EQ. 'YSH') THEN
CALL YSH
ELSE IF (COMMAND .EQ. '2') THEN
CONTINUE ! clause on the old REA command
ELSE IF (COMMAND(1:1) .EQ. '$') THEN ! If a
CALL LIBSSPAWN( COMMAND ) ! execute the DCL
C command
ELSE IF (COMMAND(1:1) .EQ. 'I') THEN 1 if
C command file comment
ELSE IF (COMMAND .NE. 'EXI') THEN 1 If not exit
C commands are not coming from the terminal
WRITE(IOUT,10) COMMAND 1 output the
C comment
END IF
IF (COMMAND .NE. 'EXIT') THEN 1 If exit
C then it must be undefined
WRITE(IOUT,*)'?'
END IF

STOP 1 do a fortran stop
END

SUBROUTINE CON_FILE(HEAD, LINE1, LINE2, LINE3)

INCLUDE 'USER2:(DURAL.CIMAG2]FTRN.FOR'
INCLUDE 'USER2:(DURAL.CIMAG2]MAGCMN.FOR'
INCLUDE 'USER2:(DURAL.CIMAG2]HEADER.CMN'
INCLUDE 'USER2:(DURAL.CIMAG2]NSGBLK.FOR'

INTEGER*4 SYS$ALLOC ,
        SYS$DALLOC 1 allocate routine
+ INTEGER*4 FOR RETCODE
INTEGER*4 RETCODE 1 fortran return code
INTEGER*4 SS$NORMAL ,
        SS$DEVALLOC 1 device already allocated to another process

PARAMETER (SS$NORMAL = '00000001'X)
PARAMETER (SS$DEVALLOC = '00000840'X)

THE INITIALIZATION ROUTINE

EXTERNAL CTRL C HDLR
CALL LIB$ESTABLISH(CTRL C HDLR) 1 Initialize ...
ENTRY START
FTYPE=F
ECHO=.FALSE.
PROC FLAG=.FALSE.
DEFINE FLAG=.FALSE.
ISIZE=4096
IS=ISIZE/2.
FSIZE=0.
IHDSZ=256
IDSSZ=1
PI=3.14159
IZ=12
ISM2=(ISIZE/2)-2
NBUF=30

46
NOW CONSTRUCT FFT SINE TABLE

CALL FORTRAN(A,IZ,5,0,IFERR)
TYPE *,'Instructions on CIMAG2 available through the command HELP'
RETURN

ENTRY FFT

DO 20 N=1,ISIZE
    A(N)=CMPLX(1.,0.)*P(N,1)
    CALL FORTRAN(A,IZ,5,-2,IFERR)
    IF(IFERR.NE.0) THEN
        CALL LIB$SIGNAL(MAG_COM)
        RETURN
    END IF
    RT2=SQR(2.)
    X=CABS(A(1))
    IF(X.LT.1.E-20) X=1.E-20
    P(1,1)=10.24*(10.+.20.*ALOG10(X))
    IS=ISIZE/2.
    P(IS+1,1)=512.*ATAN2(AIMAG(A(1)),REAL(A(1)))/PI
    PMAX=1.E-20
    DO 21 N=2,IS
        NN=IS+N
        X=CABS(A(N))
        IF(X.LT.1.E-20) X=1.E-20
        P(N,1)=10.24*(10.+.20.*ALOG10(X*RT2))
        IF(P(N,1).GT.PMAX) PMAX=P(N,1)
        X=CABS(A(N))
        IF(X.EQ.0)P(NN,1)=0
        P(NN,1)=512.*ATAN2(AIMAG(A(N)),REAL(A(N)))/PI
    END DO

ENTRY IFFT

RETURN
IS = ISIZE / 2.
A(1) = CMPLX(COS(P(IS+1,1)), SIN(P(IS+1,1))) * 10.**(P(IS+1,1) * 0.05)
DO 30 N = 2, IS
P(N,1) = P(N,1) / 10.24 - 10.
P(IS+N,1) = PI * P(IS+N,1) / 512.
A(N) = CMPLX(COS(P(IS+N,1)), SIN(P(IS+N,1))) * RRT2 * 10.**(P(N,1) * 0.05)
A(IS+1) = CMPLX(0., 0.)
DO 31 N = 2, IS
NN = ISIZE + 2 - N
A(NN) = CMPLX(REAL(A(N)), -AIMAG(A(N)))
CALL FORT(A, 12, 5, 2, IFERR)
IF(IFERR.NE.0) THEN
   CALL LIB$SIGNAL(MAG_COM)
   PRINT an error message
END IF
DO 32 N = 1, ISIZE
NN = ISIZE + 1 - N
P(NN,1) = REAL(A(NN))
RETURN
ENTRY FTY
WRITE(*,'(A)') 'Enter file type for FTRAN read routines'
WRITE(*,'(A)') T = '750 F = ' 11/23'
READ(COM_UNIT, *, FTYPE)
WRITE(LOG_UNIT, *, FTYPE)
RETURN
ENTRY FTRA COMAND (FREQUENCY DOMAIN)
ENTRY FTRA
FLTP = 1
IF(.NOT.FTYPE) CALL FREA(INPFILE, AMPL, PHS)
IF(FTYPE) CALL FREU(INPFILE, AMPL, PHS)
RETURN
ENTRY FTRA COMAND (TIME DOMAIN)
ENTRY FFTRA
FLTP = 0
IF(.NOT.FTYPE) CALL FREA(INPFILE, AMPL, PHS)
IF(FTYPE) CALL FREU(INPFILE, AMPL, PHS)
RETURN
DIFFERENTIATE ROUTINE
MULT BY JW IN THE FREQ DOMAIN
ENTRY MJW
SN = 1.
GOTO 40
INTEGRATE ROUTINE
DIVIDE BY JW IN THE FREQ DOMAIN
ENTRY DJW
SN = -1.
40 IS = ISIZE / 2.
IF (FLTP.NE.1.) THEN
   CALL LIB$SIGNAL(MAG_COM)
END IF
DO 41 N=2,1S
  NN=1S+N
  P(NN,1)=P(NN,1)+SN*256.
  P(N,1)=P(N,1)+10.24*SN+20.*ALOG10(FLOAT(N-1))
  P(1,1)=-100
  P(IS+1,1)=0.
RETURN

FILTER ROUTINE
MULT BY W IN THE FREQ DOMAIN
ENTRY MWR
SN=1.
GOTO 45

FILTER ROUTINE
DIVIDE BY W IN THE FREQ DOMAIN
ENTRY DWR
SN=-1.

IS=ISIZE/2.
IF (FLTGE.NE.1.) THEN
  CALL LIB$SIGNAL(MAG_CON)
END IF
DO 46 N=2,1S
  P(N,1)=P(N,1)+10.24*SN+20.*ALOG10(FLOAT(N-1))
  P(1,1)=-100
  P(IS+1,1)=0.
RETURN

GATE COMMAND
TRAPEZOIDAL GATE OF TIME DOMAIN WAVEFORM
ENTRY GAT

FORMAT (15)
WRITE (IOUT,50)
FORMAT(' START GATE OPEN(SAMPLE NO.)-')
READ (COMUNIT,49) IOP
WRITE(IOUT,51)
FORMAT(' START GATE CLOSE AT SAMPLE ')
READ (COMUNIT,49) ICL
WRITE(IOUT,52)
FORMAT(' RAMP LENGTH IN SAMPLES -')
READ (COMUNIT,49) IRMP
WRITE (LOGUNIT,49) IRMP
IF(IRMP.EQ.0)IRMP=1
DO 53 N=1,ISIZE
  F=0.
  RMP=FLOAT(IRMP)
  NUM=N-IOP
  FNUM=FLOAT(NUM)
  IF(FNUM.GT.IOP)F=FNUM/RMP
  IF(FNUM.GT.1.)F=1.
  NUM=N-ICL
  FNUM=FLOAT(NUM)
  IF(FNUM.GT.ICL)F=1.-FNUM/RMP
  IF(FNUM.LT.0.)F=0.

49
53   P(N,1)=P(N,1)*F
RETURN

**TYPE COMMAND**

TYPES A BUFFER WAVEFORM SEGMENT
ENTRY TYP(HEAD, LINE1, LINE2, LINE3)
WRITE(IOUT,81)
81   FORMAT(' START AT ELEMENT NUMBER:','$')
READ (COM_UNIT,86) NP
WRITE (LOG_UNIT,86) NP
WRITE(IOUT,82)
82   FORMAT(' END AT ELEMENT NUMBER:','$')
READ (COM_UNIT,86) NE
WRITE (LOG_UNIT,86)
83   FORMAT(' BUFFER NUMBER(0-MAIN):','$')
READ (COM_UNIT,86) NNB
WRITE (LOG_UNIT,86)
IF(NNB.LT.1) THEN
   CALL LIB$SIGNAL(MAGCOM)
   RETURN
END IF
IF(NNB.GT.NBUF) THEN
   CALL LIB$SIGNAL(MAGCOM)
   RETURN
ENDIF
WRITE(IOUT,117) (HEAD(I,NNB),I-1,30)
WRITE(IOUT,117) (HEAD(I,NNB),I-31,60)
WRITE(IOUT,117) (HEAD(I,NNB),I-61,90)
WRITE( IOUT,85)(N,P(N,NNB),P(N+1,NNB),N-NP,NE,2)
85   FORMAT(I5,2F10.2)
86   FORMAT(I5)
RETURN

**BUFFER STORE ROUTINES**

STORE A WAVE OR SPECTRUM IN ONE OF 30 TEMP LOCS
ENTRY SBF(HEAD, LINE1, LINE2, LINE3)
WRITE(IOUT,87)
87   FORMAT(I8,' BUFFER #','$')
READ (COM_UNIT,93) NB
WRITE (LOG_UNIT,93)
ENTRY SB(HEAD, LINE1, LINE2, LINE3)
IF (NB .GT. 35) THEN
   CALL LIB$SIGNAL(MAGCOM)
   RETURN
ENDIF
DO 90 I = 1, 30
BUFFERS(NB,I) = LINE1(I)
90   CALL BUFSTR(IDHSZ,IDSsz,ISIZE,NB,HEAD,P,IDS)
93   FORMAT(I5)
RETURN

**BUFFER READ ROUTINES**

RETRIEVES WAVE OR SPECTRUM AND ITS HEADING AND ITS DESCRIPTIVE PARAMETERS INTO THE MAIN BUFFER
ENTRY RBF(HEAD, LINE1, LINE2, LINE3)
WRITE (IOUT,88)
FORMAT(IX,' BUFFER #',$)
READ (COM UNIT,94) NB
WRITE (LOG_UNIT,94) NB

ENTRY RB(HEAD, LINE1, LINE2, LINE3)
DOMAIN = DOM(NB)
CALL BUFRD(IHDSZ, IDSSZ, ISIZE, NB, READ, P, IDS)
FORMAT (15)
RETURN

COMBINING ROUTINE
COMBINES FILES FROM TEMP BUFFERS, EACH MULTIPLIED BY A
FACTOR, INTO THE MAIN BUFFER. HEADING AND DESCRIPTIVE PAR
OF THE LAST WAVEFORM WILL BE IN THE MAIN BUFFER

ENTRY COM(HEAD, LINE1, LINE2, LINE3)
DO 101 I=1, ISIZE
    P(I,1)=0
    WRITE(IOUT,103)
FORMAT(' STORAGE BUFFER NO.(USE 0 TO FINISH):')
READ (COM UNIT,111) IB
WRITE (LOG_UNIT,111) IB
    IF(IB.LT.1.OR.IB.GT.NBUF)GO TO 110
    DOMAIN = DOM(IB)
    IB=IB+1
    WRITE(IOUT,104)
FORMAT(' MULTIPLIER=')
READ (COM UNIT,108) AMULT
WRITE (LOG_UNIT,108) AMULT
DO 105 J=1, IHDSZ
    HEAD(J,1)=HEAD(J,IB)
DO 107 J=1, ISIZE
    P(J,1)=P(J,IB)+AMULT*P(J,IB)
    IDS(1)=IDS(IB)
GO TO 102
108 FORMAT (F10.2)
110 CONTINUE
111 FORMAT (I5)
RETURN

ROUTINE TO READ FROM DISK

ENTRY REA(HEAD, LINE1, LINE2, LINE3)
DOMAIN='TIME'
WRITE(IOUT,112)
FORMAT(' INPUT FILE ? ',$)
READ (COM UNIT,114) FILNM
WRITE (LOG_UNIT,114) FILNM
IF((PROC_FLAG).AND.(FILNM .EQ.'NAME.DAT')) THEN
    I make a dummy time domain
    P(1,1)=10
END DO
    IDS(1)=0
WRITE(IOUT,113)
RETURN
113 FORMAT(' This is a dummy file for PROC definition.')
114 FORMAT(A50)
OPEN(UNIT=1, ERR=115, FILE=FILNM, TYPE='OLD', READONLY,
     IOSTAT= FOR_READCODE, DEFAULTFILE=DEF, FORM='UNFORMATTED')
IPGF=0
GOTO 116
115 IF (FOR_READCODE .EQ. 29) THEN
END IF
CALL LIBSSIGNAL(MAG_FILENOTFOU) ! print error
RETURN
ELSE
CALL LIBSSIGNAL(MAG_COM) ! Print an error message
RETURN
END IF
READ (1, IOSTAT=FOR_RETCODE, END=162) (HEAD(I,1), I=1,256)
IF (FOR_RETCODE.EQ. 35) THEN
CALL LIBSSIGNAL(MAG_INPFOR)
RETURN
END IF
WRITE(IOUT,117) LINE1
WRITE(IOUT,117) LINE2
WRITE(IOUT,117) LINE3
FORMAT(X,30A2)
DO 77812 J=1,ISIZE
P(J,1)=0.
READ(1,END=162) (P(J,1),J=1,ISIZE)
IDS(1)=0
GO TO 164
77812
WRITE(TERM_UNIT,*) 'ERROR END OF FILE ENCOUNTERED AT ',I
CLOSE(UNIT=1)
FORMAT (I5)
% RETURN
C
C
C
C
C
TO DELETE A FILE ON DISK
ENTRY DEL
WRITE(IOUT,*) 'ENTER THE FILE NAME'
READ (COM_UNIT,114) FILNM
WRITE (LOG_UNIT,114) FILNM ! log the input
OPEN(UNIT=I,ERR=166,FILE=FILNM,TYPE='OLD',
+ DEFAULTFILE=DEF,DISP='DELETE')
CLOSE(UNIT=I)
GOTO 167
166 CALL LIBSSIGNAL(MAG_COM) ! Print an error message
GOTO 167
RETURN

C
C
C
C
C
 THIS ROUTINE PLOTS A WAVEFORM FROM A BUFFER
ENTRY PLO
WRITE (IOUT,186)
READ (COM_UNIT,187) ISEG1
WRITE (LOG_UNIT,187) ISEG1
ISEG=ISEG1+1
WRITE(IOUT,*)'DO YOU WANT A NEW WINDOW?(1=Y,0=NO)' READ (COM_UNIT,*) IANS
WRITE (LOG_UNIT,*) IANS ! log the input
IF (IANS.NE.1) GO TO 18500
WRITE(IOUT,*)'WINDOW VALUES ARE NOW:' WRITE (IOUT,4247) WIN1,WIN2,WIN3,WIN4
WRITE(IOUT,*)'START ELEMENT FOR PLOT(0=NO CHANGE):' READ (COM_UNIT,*) IANS1
WRITE (LOG_UNIT,*) IANS1 ! log the input
IF(IANS1.NE.0) IWIN1=IANS1
WRITE(IOUT,*)'END ELEMENT FOR PLOT (0=NO CHANGE):' READ (COM_UNIT,*) IANS1
WRITE(IOUT,*)'NOW PLOTTING WAVEFORM FROM BUFFER:' WRITE (IOUT,183) (P(J,1),J=1,ISIZE)
WRITE(IOUT,*)'IF YOU HAVE AN ERRONEOUS WINDOW:
+ WRITE(IOUT,*)'WRITE (LOG_UNIT,*) IANS1
+ IWIN1=IANS1
WRITE(IOUT,*)'END ELEMENT FOR PLOT (0=NO CHANGE):' READ (COM_UNIT,*) IANS1
WRITE (LOG UNIT,*), IANS1 ! log the input
IF(IANS1.NE.0) IWIN2=IANS1
WRITE(IOUT,*), 'MAX PLOT VALUE (+1024 NORMAL, 0=NO CHANGE):'
READ (COM_UNIT,*), ANS
WRITE (LOG UNIT,*), Ilog the input
IF(ANS.NE.0) WIN=ANS
WRITE(IOUT,*), 'MIN PLOT VALUE (-1024 NORMAL,0=NO CHANGE):'
READ (COM_UNIT,*), ANS
WRITE (LOG UNIT,*), Ilog the input
IF(ANS.NE.'U) WIN=ANS
WIN1=FLOAT(IWIN1)
WIN2=FLOAT(IWIN2)
18500 WRITE(IOUT,*), 'DO YOU WANT NEW AXES?(1=Y, 0=N)'
READ (COM_UNIT,*), ANS
WRITE (LOG UNIT,*), Ilog the input
IF(ANS.NE.0) GO TO 18600
WRITE(IOUT,*), 'INPUT Y-AXIS MAX,MIN, AND TIK MARK INTERVAL'
READ (COM_UNIT,*), YMAX,YMIN,YTMI
WRITE (LOG UNIT,*), Ilog the input
WRITE(IOUT,*), 'Y AXIS LEGEND'
READ (COM UNIT,18601), YLAB
WRITE (LOG UNIT,18601) YLAB
WRITE(IOUT,*), 'INPUT X-AXIS MAX,MIN, AND TIK MARK INTERVAL'
READ (COM_UNIT,*), XMAX,XMIN,XTMI
WRITE (LOG_UNIT,*), Ilog the input
WRITE(IOUT,*), 'X AXIS LEGEND'
READ (COM_UNIT,18601), XLAB
WRITE (LOG UNIT,18601) XLAB
WRITE(IOUT,*), 'INPUT TITLE FOR PLOT'
READ (COM UNIT,18601), TITLE
WRITE (LOG UNIT,18601) TITLE
18600 FORMAT((A68))
4247 FORMAT(' STRT','F8.2,' END ','F8.2,' MIN Y','F8.2,' MAX Y','F8.2)
YC=WIN4-WIN3
YZERO=WIN3+YINC/2.
XINC=WIN2-WIN1
IXINC=INT(XINC)
C DRAW THE CURVE USING A SQUARE 7 IN PLOT
CALL VPlOTS(0,0,0)
PSIZE=7.
DO 18610 I=1,IXINC
X=I*PSIZE/XINC
Y=PSIZE*(P(IWIN1+I,ISEG)-YZERO)/YINC
18610 CALL PLOT(X,Y,2)
CALL PLOT(0,-PSIZE/2.,-3)
CALL PFFAXIS(0.,0.,XMAX,XMIN,0.,0.,XDS,XSP,1,1.1)
CALL PFFAXIS(0.,0.,0.,0.,0.,0.,0.,0.,-1,1.1)
CALL PTEXT(0.5,PSIZE+0.5,0.1,TITLE,0.)
CALL PLOT(0.9,999)
CALL PLOTNOW(IMG)
IF(IMG.EQ.0) THEN
CALL LIBSSIGNAL(MAG_COM) ! Print an error message
RETURN
END IF
186 FORMAT(IX,'WAVEFORM BUFFER NO.? (0=MAIN)')
187 FORMAT(15)
THIS ROUTINE READS DATA FROM A BASIC SCATTERING CODE CALCULATION

ENTRY BSC
CALL BSCREA(P, COM_UNIT)
FLTP=1.
RETURN

THIS ROUTINE ATTACHES A SMOOTH CUTOFF TAIL TO THE SPECTRUM IN ORDER TO REDUCE THE GIBBS PHENOMENON

FROM LEEPER'S COSINE ROLLOFF 18FEB83 IN [LEEPER.X]DUMPZ.FOR

ENTRY WND
WRITE(IOUT,*) 'INPUT HARMONICS;START,END,TYPE OF WINDOW'
WRITE(IOUT,*) 'TYPE: 0=HANNING,1=HAMMING,2=GAUSSIAN, 0,N,M,=TEST'
WRITE(LOG_UNIT,*) IFS,IFE,ITYP
READ(LOG_UNIT,*) IFS,IFE,ITYP
WRITE(LOG_UNIT,*) IFS,IFE,ITYP
MAKE THE INPUT
IF(IFS.GE:IFE) THEN
CALL LIB$SIGNAL(MAG_CON)
I
PRINT AN ERROR MESSAGE
RETURN
END IF
IF(ITYP.EQ.0) THEN
WRITE(IOUT,*) 'INPUT ALPHA'
END IF
IF(ITYP.EQ.2) THEN
READ(COM_UNIT,*) ALPHA
END IF
IF(ITYP.EQ.2) THEN
WRITE(IOUT,*) 'INPUT ALPHA'
WRITE(LOG_UNIT,*) ALPHA
I
LOG THE INPUT
PI=3.1415927
NW=IFE-IFS+1
XN2=NW/2.
IF(XN2.NE.INT(XN2)) GO TO 178
IF(IFS.EQ.IFE) GO TO 180
GO TO 179
179 IN2=XN2
IF(IN2.NE.INT(IN2)) GO TO 190
(IF(IFS.EQ.0) GO TO 190
P(I-15,1)=0.
189 P(I,1)=102.4
IF(ILS=IFE) GO TO 179
190 DO 198 I=IFS,IFE
N=I-IFS
WRITE(IOUT,*) I
WFACT=0.5*(1.-COS(2*PI*N/NW))
*HANNING
IF(ITYP.EQ.1) WFACT=0.54-.46*COS(2*PI*N/NW)
*HAMMING
IF(ITYP.NE.2) GO TO 195
N=N-1N2
AVAR=-0.5*(ALPHA*N/N2)**2
WFACT=EXP(AVAR)
195 IF(WFACT.EQ.0) WFACT=3.E-6
P(I,1)=P(I,1)/10.24-10.
=LOG(ORIGINAL VOLTAGE)
P(I,1)=10.**(P(I,1)/20.)
=LINEAR VOLTAGE
54
P(l,l)=P(I,1)*WEIGHT WITH SHIFTED COSINE
198
P(I,1)=10.24*(10.+20.*ALOG10(P(I,1)))
IFE=IFE+1
IFS=IFS-1
DO 199 I=IFE,IFS
199 P(I,1)=-1024.
DO 188 I=1,IFS
188 P(I,1)=-1024.
RETURN
C*******FILL REMAINING P(K,l) WITH -110dB
C
C
C
C
C
C
C
C
C
C
C
C
GATE IN THE FREQ DOMAIN ROUTINE:FGT
STARTS AT HARMONIC N
ATTENUATES AT X DB PER HARMONIC
C
ENTRY FGT
WRITE (IOUT,191)

191 FORMAT(' HARMONIC FOR START HIGH FREQ CUTOFF ',$)
READ (COM UNIT,200) IHAR
WRITE (LOG UNIT,200) IHAR
WRITE (IOUT,192)

192 FORMAT(' HIGH FREQ ROLL-OFF IN DB PER HARMONIC',$)
READ (COM UNIT,201) XDB
WRITE (LOG UNIT,201) XDB
WRITE (IOUT,193)

193 FORMAT(' HARMONIC FOR START OF LOW FREQ CUTOFF ',$)
READ (COM UNIT,200) IHAR
WRITE (LOG UNIT,200) IHAR
WRITE (IOUT,194)

194 FORMAT(' LOW FREQ ROLLOFF IN DB PER HARMONIC ',$)
READ (COM UNIT,201) YDB
WRITE (LOG UNIT,201) YDB
DO 195 I=IHAR,ISIZE
195 P(I,1)=P(I,1)-10.24*XDB*FLOAT(I-IHAR)
IF(P(I,1).LT.-1024.) P(I,1)=-1024.
CONTINUE
DO 196 I=1,ILMAR
196 P(I,1)=P(I,1)-10.24*YDB*FLOAT(IHAR-I)
IF(P(I,1).LT.-1024.) P(I,1)=-1024.
CONTINUE
200 FORMAT(I5)
201 FORMAT(F10.2)
RETURN
C

C
TIME DOMAIN INTEGRATE ROUTINE
C
ENTRY INTEG
IF(IDS(1).NE.0) THEN
CALL LIB$SIGNAL(MAG_COM)
END IF
PIZ=2.*PI/ISIZE
P(1,1)=P(1,1)*PIZ
DO 210 N=2,ISIZE
210 P(N,1)=P(N-1,1)+P(N,1)*PIZ
RETURN
C
TIME DOMAIN DIFFERENTIATE ROUTINE

ENTRY DIF
IF(IDS(1).NE.0) THEN
   CALL LIB$SIGNAL(MAG_COM)
   RETURN
END IF
PE=P(ISIZE,1)
DO 220 N=2,ISIZE
   NN=ISIZE+2-N
   P(NN,1)=(P(NN,1)-P(NN-1,1))*ISIZE/(2.*PI)
P(1,1)=(P(1,1)-PE)*ISIZE*0.5/PI
RETURN

STATEMENT 550 32583

RELIABLE COMMAND

ENTRY RLB
WRITE (IOUT,225)
WRITE (IOUT,231) ITITLE
225 FORMAT(/,' OLD TITLE BLOCK: ')
WRITE (IOUT,230)
230 FORMAT(/,' NEW TITLE BLOCK: ')
WRITE (LOG_UNIT,231) ITITLE
231 FORMAT(50A1)
RETURN

NORMALIZING TIME DOMAIN DATA TO ZERO MEAN

ENTRY NOR
RMEAN=0.
DO 3111 I=1,ISIZE
   RMEAN=P(I,1)+RMEAN
   RMEAN=RMEAN/ISIZE
DO 3112 I=1,ISIZE
   P(I,1)=P(I,1)-RMEAN
RETURN

CLEAR THE MAIN BUFFER

ENTRY CLR
DOMAIN = ' '
LINE1(1) = 0
DO 3113 I=1,ISIZE
   P(I,1)=0.
RETURN

POINT SMOOTH ROUTINE

ENTRY PSM
IF(FLTP.EQ.1) GO TO 4051
WRITE (IOUT,4001)

56
4001 FORMAT(' FIRST BAD POINT INTEGER= ',$)
READ (COM UNIT,4003) IQ
WRITE (LOG UNIT,4003) IQ
WRITE (IOUT,4002)
        : log the input

4002 FORMAT(' LAST BAD POINT INTEGER= ',$)
4003 FORMAT(15)
READ (COM UNIT,4003) IR
WRITE (LOG UNIT,4003) IR
        : log the input

IF(IQ.LT.3) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IR.LT.3) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IQ.GT.ISIZE-2) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IR.GT.ISIZE-2) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IR.LT.IQ) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF

PSQ=P(IQ-1,1)
PSN=IR-IQ-2
PSS1=P(IQ-1,1)-P(IQ-2,1)
PSS2=P(IR+2,1)-P(IR+1,1)
PSD1=P(IR+1,1)-P(IQ-1,1)
PSD2=(PSS2*PSN+PSN*(2.*PSS1+PSS2)/PSN
PSD=(PSS2*PSN1)/(PSN*PSN)-2.*PSD1/(PSN*PSN*PSN)
PSA=PSQ
PSB=PSS1
IPSN=IR-IQ+1
DO 4010 I=1,IPSN
4010 P(IQ+IPSN-1,1)=PSA+PSB*I+PSC*I+PSD*I*I
GO TO 4075

4051 WRITE (IOUT,4052)
4052 FORMAT( ' FIRST BAD HARMONIC= ',$)
READ (COM UNIT,4003) IQ
WRITE (LOG UNIT,4003) IQ
WRITE (IOUT,4053)
        : log the input

4053 FORMAT( ' LAST BAD HARMONIC= ',$)
READ (COM UNIT,4003) IR
WRITE (LOG UNIT,4003) IR
        : log the input

IF (IQ.LT.3) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IR.LT.3) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IQ.GT.ISIM2) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IR.GT.ISIM2) THEN
    CALL LIB$SIGNAL(MAG_COM)
END IF
IF(IR.LT.IQ) THEN
        : Print an error message

Print an error message
CALL LIB$SIGNAL(MAG_Com)
RETURN
END IF
RT2=SQRT(2.)
DO 4056 N=IQ-2,IR+2
NN=IS+N
AMP=(P(N,1)/10.24)-10.
PHAS=PI*P(NN,1)/512.
A(N)=CMPLX(COS(PHAS),SIN(PHAS))*RT2*10.*((AMP*.05)
4056 WRITE(IOUT,*) N,AMP,PHAS,A(N)
ASQ=A(IQ-1)
ASN=IR-IQ+2
ASS1=A(IQ-1)-A(IQ-2)
ASS2=A(IR+2)-A(IR+1)
ASDI=A(IR+1)-A(IQ-1)
ASC=3.*ASDI/(ASN*ASN)-(2.*ASS1+ASS2)/ASN
ASD=(ASS2+ASS1)/(ASN*ASN)-2.*ASDI/(ASN*ASN*ASN)
ASA=ASQ
ASB=ASS1
WRITE(IOUT,*) ASA,ASB,ASC,ASD
IASN=IR-IQ+1
DO 4060 I=1,IASN
IN=IQ-1+I
FI=FLOAT(I)
A(IN)=ASA+ASB*FI+ASC*FI*FI+ASD*FI*FI*FI
WRITE(IOUT,*) A(IN)
X=CABS(A(IN))
IF(X.LT.1.E-20)X=1.E-20
P(IN,1)=10.24*(10.+20.*ALOG(X*RT2))
IF(X.EQ.0) P(IN+IS,1)=0.
4060 IF(X.NE.0)P(In+IS,1)=512.*ATAN2(AMAG(A(IN)),REAL(A(IN)))/PI
4075 CONTINUE
RETURN

C

YSHIFT COMMAND

ENTRY YSH
WRITE (IOUT,251)
251 FORMAT( 'YSHIFT IN UNITS(-2048<Y<2048)=',$)
READ (COM UNIT,252) IY
WRITE (LOG UNIT,252) IY
! log the input
252 FORMAT (I5)
Y=FI*AT(IY)
DO 255 I=1,ISIZE
255 F(I,1)=P(I,1)+Y
RETURN

C

ROTATE COMMAND

ENTRY ROT
WRITE (IOUT,260)
260 FORMAT ( 'ROTATE BY INCREMENTS OF:',$)
READ (COM UNIT,261) IX
WRITE (LOG UNIT,261) IX
! log the input
261 FORMAT (I5)
DO 265 I=1,ISIZE
J=I+IX
58
ENTRY DCV
IF (FLTP.EQ.0) THEN
   CALL LIB$SIGNAL(MAG_COM) ! Print an error message
   RETURN
END IF
WRITE(IOUT,*) 'INPUT THE HARMONIC NUMBER TO BE MOVED TO DC'
READ (COM UNIT,*) IHRDC
WRITE (LOG UNIT,*) IHRDC ! log the input
IF (IHRDC.GT.IS) THEN
   CALL LIB$SIGNAL(MAG_COM) ! Print an error message
   RETURN
END IF
DO 5110 N=1,ISIZE
Q(N)=P(N,1)
DO 5120 N=1,ISIZE
Q(N+IS)=PI*Q(N+IS)/512.
DO 5190 N=1,ISIZE
M=N-1
BCX=0.
CCX=0.
P(N,1,1)=0.
IF(IHRDC+M.GT.IS) GO TO 5130
BCX=CMPLX(COS(Q(IHRDC+M+IS)),SIN(Q(IHRDC+M+IS)))*10.**Q(IHRDC+M)*.05
5130 IF(IHRDC- M.LT.1) GO TO 5140
AEXP=10.**(-0.05*Q(IHRDC-M))
CCX=CMPLX(COS(Q(IHRDC-M+IS)),SIN(Q(IHRDC-M+IS)))*AEXP
5140 DCX=BCX+CCX
X=CARB(DCX)
IF(X.LT.1.E-20) X=1.E-20
P(N,1)=10.24*(10.+20.*ALOG10(X))
IF(X.GT.1.E-20) GO TO 5190
CONTINUE
RETURN
END

ENTRY CMI
WRITE(IOUT,*) 'NUMBER OF VV TIME DOMAIN WAVEFORMS TO BE USED?'
READ (COM UNIT,*) NUMB(1)
WRITE (LOG UNIT,*) NUMB(1)
DO 5510 I=1,NUMB(1)
WRITE (IOUT,5503) I 1 log the input

5503 FORMAT( ' BUFFER NUMBER FOR VV FILE # ',I5)
READ (COM UNIT,*) BUFN(1,I)
WRITE (LOG UNIT,*) BUFN(1,I)
BUFN(1,I)=I-BUFN(1,I)
WRITE (IOUT,5505) I 1 log the input

5505 FORMAT( ' LOOK ANGLE IN DEGREES FOR FILE # ',I5)
READ (COM UNIT,*) ANG(1,I)
WRITE (LOG UNIT,*) ANG(1,I)
ANG(1,I)=ANG(1,I)*PI/180.
WRITE (IOUT,5507) I 1 log the input

5507 FORMAT( ' CENTER ELEMENT NUMBER FOR FILE # ',I5)
READ (COM UNIT,*) CNTR(1,I)
WRITE (IOUT,*) CNTR(1,I)
5510 WRITE (LOG UNIT,*) CNTR(1,I) 1 log the input
WRITE (IOUT,*) 'NUMBER OF HH TIME DOMAIN WAVEFORMS TO BE USED?'
READ (COM UNIT,*) NUMB(2)
WRITE (LOG UNIT,*) NUMB(2)
DO 5520 I=1,NUMB(2)
WRITE (IOUT,5513) I 1 log the input

5513 FORMAT( ' BUFFER NUMBER FOR HH FILE # ',I5)
READ (COM UNIT,*) BUFN(2,I)
WRITE (LOG UNIT,*) BUFN(2,I)
BUFN(2,I)=I-BUFN(2,I)
WRITE (IOUT,5505) I
READ (COM UNIT,*) ANG(2,I)
WRITE (LOG UNIT,*) ANG(2,I)
ANG(2,I)=ANG(2,I)*PI/180.
WRITE (IOUT,5507) I
READ (COM UNIT,*) CNTR(2,I)
WRITE (IOUT,*) CNTR(2,I)
5520 WRITE (LOG UNIT,*) CNTR(2,I) 1 log the input
WRITE (IOUT,*) 'NUMBER OF HV TIME DOMAIN WAVEFORMS TO BE USED?'
READ (COM UNIT,*) NUMB(3)
WRITE (LOG UNIT,*) NUMB(3)
DO 5530 I=1,NUMB(3)
WRITE (IOUT,5523) I 1 log the input

5523 FORMAT( ' BUFFER NUMBER FOR HV FILE # ',I5)
READ (COM UNIT,*) BUFN(3,I)
WRITE (LOG UNIT,*) BUFN(3,I)
BUFN(3,I)=I-BUFN(3,I)
WRITE (IOUT,5505) I
READ (COM UNIT,*) ANG(3,I)
WRITE (LOG UNIT,*) ANG(3,I)
ANG(3,I)=ANG(3,I)*PI/180.
WRITE (IOUT,5507) I
READ (COM UNIT,*) CNTR(3,I)
WRITE (IOUT,*) CNTR(3,I)
5530 WRITE (LOG UNIT,*) CNTR(3,I) 1 log the input
RETURN

ROUTE TO FORM THE IMAGE FOR ONE POLARIZATION FROM TIME DOMAIN WAVEFORMS AS SET UP BY THE CMI COMMAND

ENTRY IMG

WRITE(IOUT,*) 'SIZE OF THE IMAGE,(1 TO 4096)=?' 1 get the window size
READ (COM UNIT,*) IMSIZE
WRITE (LOG UNIT,*) IMSIZE 1 log the input
FIMINC=FLO.XT(IMSIZE/100) 1 the size of the increment
PNUM=FIMINC
IF (MOD(PNUM,2) .EQ. 0) THEN 1 figure out the number of harmonics to be averaged for an array value

60
PNUM=PNUM+1  ! This number must be odd
END IF
IF (PNUM .LT. 3) THEN  ! when PNUM is 3 the weights of the other
   PNUM=3
END IF
WRITE(IOUT,*) 'LOOK ANGLE OF THE IMAGE, DEGREES?'  ! get the
C rotation angle of the image
READ (COM UNIT,*) ALANGL
WRITE (LOG UNIT,*) ALANGL  ! log the input
BLANGL=ALANGL*PI/180.
WRITE(IOUT,*) 'POLARIZATION OF THE IMAGE, (1=VV, 2=HH, 3=HV, 4=ALL)'  ! which
C polarizations will be used?
READ (COM UNIT,*) IIIPOL
WRITE (LOG UNIT,*) IIIPOL
IF(IIIPOL.NE.4) GO TO 5660  ! if not using all polarizations then
   IIIPOL=IIIPOL
DO 5699 I=1,100  ! just do the individual pol.
   XX=(I-50.5)*FIMINC
   DO 5680 J=1,100  ! figure out the X coordinate
      YY=(J-50.5)*FIMINC
      XX=X*COS(BLANGL)+Y*SIN(BLANGL)
   END DO
5680   YY=X*SIN(BLANGL)+Y*COS(BLANGL)  ! the Y coordinate
5699   ARRAY(IIIPOL,I,J)=0.  ! clear the image array
DO 5670 K=1,NUMB(IIIPOL)
   XXX=XX*COS(ANG(IIIPOL,K))+YY*SIN(ANG(IIIPOL,K))  ! the X coordinate
   YYY=XX*SIN(ANG(IIIPOL,K))+YY*COS(ANG(IIIPOL,K))  ! the Y coordinate
   PVALUE=YYY+CNTR(IIIPOL,K)  ! the point of intersection
IF(IIPOINT.LT.0) IIPOINT=IIPOINT+ISIZE  ! the time domain waveform
C repeats throughout time
 IF(IIPOINT.GT.ISIZE) IIPOINT=IIPOINT-ISIZE
   VALUE=0
   DO ELEMENT=(IIPOINT-((PNUM-1)/2)),(IIPOINT+((PNUM-1)/2))  ! take
      C the points around the center point
      ELEM=ELEM
      WEIGHT=1+COS((2*(ELEM-IIPOINT)*PI)/(PNUM-1))  ! use a weighted
      C shifted cosine average
      IF(ELEM.LT.0) ELEM=ELEM+ISIZE
      IF(ELEM.GT.ISIZE) ELEM=ELEM-ISIZE
      PVALUE=WEIGHT*P(ELEM,BUFN(IIIPOL,K))
      VALUE=VALUE+PVALUE
   END DO
5670   CONTINUE
5660   CONTINUE  ! the time domain waveform
END DO
VALUE=VALUE/(PNUM-1)
ARRAY(IIIPOL,I,J)=ARRAY(IIIPOL,I,J)+VALUE/NUMB(IIIPOL)
WRITE(IOUT,*)'Do you want to store the image? Y=1, N=0'  ! the time domain waveform
WRITE(LOU,*)ANS
IF(ANS.EQ.0)RETURN
WRITE(IOUT,'Enter the output file name')
READ(COM UNIT,10)FNAME
WRITE(LOG UNIT,10)FNAME
10 FORMAT(A41)
WRITE(IOUT,'Enter the freq. increment of the signal in MHz')
READ(COM UNIT,*) FER
WRITE(LOG UNIT,*)FER
PER=1./FER*1000  ! Period of the time signal
OPEN(UNIT=31, FILE=FNAME, TYPE='NEW',
     FORM='UNFORMATTED')

WRITE(31) IMSIZE, PER
DO 99 I=1,100
DO 99 J=1,100
WRITE(31) ARRAY(IPOL, I, J)
99 CONTINUE
RETURN

THIS ROUTINE PLOTS AN ISOMETRIC VIEW OF A SINGLE POLARIZATION
TWO-DIMENSIONAL TARGET IMAGE ON THE PLOT DEVICE IN ISOMETRIC
FORM WITH NO SHADOWING

ENTRY PIM
WRITE(IOUT, 'TYPE IN A POLARIZATION INDEX(1=VV, 2=HH, 3=HV)'
READ (COM UNIT, *) IIPOL
WRITE (LOG UNIT, *) IIPOL
IF(IIPOL.GT.3) THEN
    CALL LIB$SIGNAL(MAG_COM)
    RETURN
END IF
IF(IIPOL.LT.1) THEN
    CALL LIB$SIGNAL(MAG_COM)
    RETURN
END IF
CALL VPLOTS(0, 0, 0)
CALL PLOT(4., 5., -3)
DO 5790 I=1, 100
DO 5790 J=1, 100
X=(J-20.5)*0.1
Y=(I-20.5)*0.1
XX=0.866*(X+Y)
YY=0.5*(Y-X)+ARRAY(IIPOL, I, J)/256.
IF(J.EQ.1) CALL PLOT(XX, YY, 3)
IF(J.NE.1) CALL PLOT(XX, YY, 2)
5790 CONTINUE
CALL PLOT(0, 0, 999)
CALL PLOTNOW(IMSG)
RETURN

ENTRY FRDC
CALL FRD(P, COM_UNIT)
FLTP=1
RETURN

THIS COMMAND READS NEW MEASUREMENT

ENTRY RDFL(HEAD, LINE1, LINE2, LINE3)

62
CALL FRD(P, EFLAG)
IF (EFLAG) THEN
   IF an error occurred
   EFLAG = .FALSE.
   I reset error flag
ELSE
   FLTP = 1
END IF
RETURN

READ THE CARD FILE
ENTRY CARDC
CALL CARD(P)
RETURN

SHOW BUFFER ROUTINE:
   This routine shows the contents of
   all of the buffers.
ENTRY SHO_BUF
DO I=1,40
   IF (BUFFERS(I,1) .NE. 0) THEN
      WRITE(IOUT,6105) (I, DOM(I))
      WRITE(IOUT,6110) (BUFFERS(I,J), J=1,30)
   END IF
END DO
6105 FORMAT('0', I4, 3X, A4)
6110 FORMAT(4X, 60A2)
END

SUBROUTINE FRD(P, COM UNIT)
COMPLEX BA(201), KLA, CA(2049)
CHARACTER*20 INFILE
INTEGER*4 COM UNIT
REAL FA(2049), P(2048, 11)
DO 3410 I=1, 1024
   P(I+1024, 1) = 0.
3410 P(I, 1) = 1024.
nn.
   V
7825 WRITE(IOUT,*) 'INPUT MAJOR AXIS DIMENSION (INCHES)'
READ (COM UNIT, *) DLE
WRITE (LOG_UNIT, *) DLE
FLOW = 100.
FHIGH = 0.
IBASE = 0
DLE = DLE * 2.54
TWOP = ATAN(1.) * 8.
WRITE(IOUT,*) '*** TYPE RETURN IF THERE IS NO SUCH FILE ***'
WRITE(IOUT,*) 'NOW INPUT THE 1-2G FILE NAME'
READ (COM UNIT, 7823) INFILE
WRITE (LOG_UNIT, 7823) INFILE
7823 FORMAT(A20)
IF(INFILE.EQ.' ') GO TO 3350

63
IF(FLOW.GT.1.)FLOW=1.
IF(FHIGH.LT.2.)FHIGH=2.
CALL TDATA(BA,INFILE)
DO 3360 I=1,201
CA(I)=BA(I)
3360 FA(I)=(1.+((I-1)/200.)*TWOPI*DLE/3.
IBASE=IBASE+200
WRITE(IOUT,*) ‘INPUT THE 2-4G FILE NAME’
READ (COM UNIT,7823) INFILE
WRITE (LOG UNIT,7823) INFILE
LOG THE INPUT
IF(INFILE.EQ.’ ’)GO TO 3380
IF(FLOW.GT.2.)FLOW=2.
IF(FHIGH.LT.4.)FHIGH=4.
CALL TDATA(BA,INFILE)
DO 9210 I=1,201
CA(I+IBASE)=BA(I)
9210 FA(I+IBASE)=(2.+((I-1)/100.)*TWOPI*DLE/3.
IBASE=IBASE+200
WRITE(IOUT,*) ‘INPUT THE 4-8G FILE NAME’
READ (COM UNIT,7823) INFILE
WRITE (LOG UNIT,7823) INFILE
LOG THE INPUT
IF(INFILE.EQ.’ ’)GO TO 3390
IF(FLOW.GT.4.)FLOW=4.
IF(FHIGH.LT.8.)FHIGH=8.
CALL TDATA(BA,INFILE)
DO 9220 I=1,201
CA(I+IBASE)=BA(I)
9220 FA(I+IBASE)=(4.+((I-1)/50.)*TWOPI*DLE/3.
IBASE=IBASE+200
WRITE(IOUT,*) ‘INPUT THE 8-12G FILE NAME’
READ (COM UNIT,7823) INFILE
WRITE (LOG UNIT,7823) INFILE
LOG THE INPUT
IF(INFILE.EQ.’ ’)GO TO 3400
IF(FLOW.GT.8.)FLOW=8.
IF(FHIGH.LT.12.)FHIGH=12.
CALL TDATA(BA,INFILE)

NOTE THAT THIS DO LOOP RUNS 201 TIMES
DO 9230 I=1,201
CA(I+IBASE)=BA(I)
9230 FA(I+IBASE)=(8.+((I-1)/50.)*TWOPI*DLE/3.
3400 I=1
IF(FLOW.GE.FHIGH)RETURN
KLOW=INT(TWOPI*FLOW*DLE/3.)+1
KHIGH=INT(TWOPI*FHIGH*DLE/3.)
DO 7720 K=KLOW,KHIGH
7720 IF(K.GE.FA(I).AND.K.LE.FA(I+1))GO TO 7730
7730 IF(K.GT.1024)GO TO 7777
PK,1) =-1024.*ATAN2(AIMAG(KLA),REAL(KLA))/TWOPI
7777 RETURN
END
C
C
SUBROUTINE BUFSTR(IHDSZ,IDSSZ,ISIZE,IB,HEAD,P,IDS)
INTEGER*2 HEAD(256,31)
DIMENSION P(4096,31),IDS(31)
IC=IB+1
DO 10 I=1,IHDSZ
10 HEAD(I,IC)=HEAD(I,1)
DO 20 I=1,ISIZE
20 P(I,IC)=P(I,1)

64
SUBROUTINE BUFRD(IHDSZ, IDSSZ, isize, IB, HEAD, P, IDS)
INTEGER*2 HEAD(256, 31)
DIMENSION P(4096, 31), IDS(31)
IC = IB + 1
DO 10 I = 1, IHDSZ
   10 HEAD(I, 1) = HEAD(I, IC)
DO 20 I = 1, isize
   20 P(I, 1) = P(I, IC)
IDS(1) = IDS(IC)
RETURN
END

SUBROUTINE TDATA(CA, INFILE)
COMPLEX*8 CA(201)
CHARACTER*20 INFILE
INTEGER*2 BUFF(1024)
REAL*4 AM(201), PH(201)
EQUIVALENCE (AM(1), BUFF(53)), (PH(1), BUFF(455))
INCLUDE 'SYS$LIBRARY: FORIOSDEF'
OPEN(UNIT=19, NAME=INFFILE, STATUS='OLD', DEFAULTFILE=DEFIOSTAT-IERR, ERR=8100)
   810 DO 20 I = 1, 1024, 256
       20 READ (19, 30) (BUFF(K), K = I, J)
   30 FORMAT (256A2)
DO 70 I = 1, 201
   70 CA(I) = CMPLX (AM(I), PH(I)), AM(I) * SIN(PH(I))
GO TO 80
8100 IF (IERR.EQ.FORIOS_FILENOTFOU) THEN
WRITE(TERM_UNIT, 1112) INFFILE
   1112 FORMAT ('FILE: ', A20, ' DOES NOT EXIST,// , ENTER FILENAME AGAIN')
ELSE IF (IERR.EQ.FORIOS_FILNAMSPE) THEN
WRITE(TERM_UNIT, *) 'FILE: ', INFFILE, ' WAS BAD, ENTER NEW FILENAME'
ELSE
WRITE(TERM_UNIT,*) 'UNRECOVERABLE ERROR, CODE = ', IERR
STOP
ENDIF
GO TO 810
80 CLOSE (UNIT=19, DISP='SAVE')
RETURN
END

CONTROLLER
This subroutine controls the input and output devices of the program.

SUBROUTINE CONTROLLER
INCLUDE 'USER2:DURAL.CIMAG2)FTRN.FOR'
INCLUDE 'USER2:DURAL.CIMAG2)MAGCMN2.FOR'
Include 'USER2:DURAL.CIMAG2)MAGCMN.FOR'
INCLUDE 'USER2:DURAL.CIMAG2)HEADER.CMN'
INCLUDE 'USER2:DURAL.CIMAG2)MSBLK.FOR' ! error message common block
INTEGER*4 FORRETCODE
INTEGER*4 RETCODE
INTEGER*4 SS$NORMAL
INTEGER*4

LIB$GET LUN, LIB$FREE_LUN

INTEGER*4 COM STACK(20)
INTEGER*4 TOP COM / 0 /
CHARACTER*40 DEF STACK(20)
INTEGER*4 TOP_DEF / 0 /
CHARACTER*70 NAME

PARAMETER (SS$NORMAL = '00000001'X)

WR1:

ROUTINE TO WRITE TO DISK

ENTRY WR1(HEAD, LINE1, LINE2, LINE3)
WRITE(IOUT,2)
2 FORMAT(' FILE NAME ? ',$)
READ (COM_UNIT,4) FILNM
WRITE (LOG_UNIT,4) FILNM
4 FORMAT(A5O-T
OPEN(UNIT=1,ERR=6,FILE=FILNM,TYPE='NEW',
FORM='UNFORMATTED')
GOTO 8
6 CALL LIB$SIGNAL(MAG_CON)
WRITE(IOUT,* ) 'DO YOU WANT TO CHANGE THE FILE HEADER Y=1, N=0'
READ (COM_UNIT,*) IYN
WRITE (LOG_UNIT,* ) IYN
IF(IYN.EQ.1) CALL RLBD
WRITE(1) (HEAD(I,1),I=1,256)
WRITE(1) (P(I,1),I=1,ISIZE)
CLOSE(UNIT=1)
RETURN

FILE:

This command gives control of the program to a command file. The output however will still go wherever IOUT is set.

ENTRY FILE

RETCODE=LIB$GET_LUN( FILE_UNIT )  I get the LU$ for the command file
IF (RETCODE .NE. SS$NORMAL) THEN  I if no errors occurred
   CALL LIB$STOP( $VAL(RETCODE) )
END IF
IF(PROC_FLAG)THEN
   NAME='USER2:[DURAL.CIMAG2]PROC2.DAT'
ELSE
   WRITE(IOUT,10) ! prompt for file name
   READ(COM_UNIT,15) NAME
END IF

66
OPEN (UNIT=FILE UNIT, FILE=NAME", DEFAULTFILE=DEF, IOSTAT=FOR RETCODE, STATUS=’OLD’)

IF (FOR RETCODE .EQ. 29) THEN
    CALL LIB$SIGNAL(MAG_FILNOTFOU)
    RETURN
ELSE IF (FOR RETCODE .NE. 0) THEN
    CALL LIB$SIGNAL(MAG_CON)
    PRINT AN ERROR MESSAGE
    RETURN
END IF

TOP_COM = TOP_COM + 1
COM_STACK( TOP_COM ) = COM_UNIT

TOP_DEF = TOP_DEF + 1
DEF_STACK( TOP_DEF ) = DEF

COM_UNIT = FILE_UNIT

FORMAT(IX,’COMMAND FILE?’,
       FORMAT(A)
       FORMAT(IX,’ERROR IN RETRIEVING LU#’)
RETURN

RETURN

BACK:
   This command gives control back to the unit that had control before this one took over.

ENTRY BACK

IF (COM_UNIT .EQ. TERM_UNIT) THEN
   IF terminal is command input
      RETURN
   THEN ignore this command
END IF

FILE UNIT = COM_UNIT
CLOSE(UNIT = FILE_UNIT)

COM_UNIT = COM_STACK( TOP_COM )
   pop the last com_unit off the stack
TOP_COM = TOP_COM - 1

DEF = DEF_STACK( TOP_DEF )
TOP_DEF = TOP_DEF - 1

RETCODE = LIBS$FREE LU( FILE_UNIT )
   give LU# back to system
IF (RETCODE .NE. $S$ NORMAL) THEN
   IF error
      CALL LIBS$STOP( %VAL( RETCODE ) )
      THEN stop and give reason
END IF

IF (COM_UNIT .EQ. TERM UNIT) THEN
   IF the terminal has become
   the command unit
      WRITE(„IOUT,25)"
      THEN tell the user
25
   FORMAT(IX,’CONTROL HAS RETURNED TO THE TERMINAL’)
RETURN
DEFS: This command sets the default for all the file I/O.

ENTRY DEFFER
WRITE(IOUT,30)
READ(COM_UNIT,35) DEF
WRITE(LOG_UNIT,35) DEF
IF (PROC_FLAG) THEN
  DEF='USER2:[DURAL.CIMAG2]' ! if defining a procedure
ENDIF

ENTRY LOG
This command copies all the user input into a command file.

ENTRY STOLOG
Stop logging.

LOG: This command copies all the user input into a command file.

STO_LOG: Stop logging.

END

68
SUBROUTINE COMPLEX_COM:
This subroutine contains all of the complex commands. These complex commands use the other
command routines in combinations to perform more complex operations.

SUBROUTINE COMPLEX_COM
INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'

REAL SEC 1 number of nanoseconds per division
INTEGER*4 INC 1 increment
INTEGER*4 GWID 1 width of a grid line
INTEGER*4 PROC_UNIT 1 LU# for definition file
INTEGER*4 LIST_UNIT 1 LU# for list file
INTEGER*4 NEW_UNIT 1 LU# for list of output file names
INTEGER*4 PROC2_UNIT 1 The command procedure file
INTEGER*4 FOR_RETCODE 1 fortran return code
INTEGER*4 PROCSTAT 1 status of a read from def. file
INTEGER*4 LISTSTAT 1 status of a read from list file
INTEGER*4 NEWSTAT 1 status of a read from output list file
INTEGER*4 FOR_EOF / -1 / 1 end of file code

CHARACTER*80 OPER 1 input data from the definition file
CHARACTER*1 REPLY 1 answer to Y or N question
CHARACTER*70 NAME 1 name of the logging file
CHARACTER*70 NEWNAME 1 new file name
CHARACTER*75 COMMAND 1 command sent to a spawned procedure
CHARACTER*70 PROCNAME 1 definition file name
CHARACTER*70 LISTNAME 1 data list file
CHARACTER*70 OUTNAME 1 output list name

CLR_BUF:
This routine clears all the buffers

ENTRY CLR_BUF(HEAD, LINE1, LINE2, LINE3)
CALL CLR 1 clear the main buffer
DO NB=1,40 1 do for all the buffers
   IF (BUFFERS(NB,1).NE.0) THEN 1 if buffer is not empty
      CALL SB(HEAD, LINE1, LINE2, LINE3) 1 SB# routine
   END IF
END DO
RETURN

GRID:
This makes a time domain file with a harmonic
value of 100 at every given number of nanoseconds
in the main buffer.

ENTRY GRID(HEAD, LINE1, LINE2, LINE3)
WRITE(10,10) 1 prompt for the number of nanoseconds between lines
READ(COM_UNIT,20) SEC 1 The number of nanoseconds between lines
CALL CREATE(HEAD, LINE1, LINE2, LINE3) 1 Create a new time domain
C file in the main buffer

TEM=10.0

INC=SEC*41: The size of the increment (41 harmonics = 1 nanosecond)

WRITE(IOUT,30) ! The current harmonic to be changed

READ(COM UNIT,*1) GWID ! The width value

DO WHILE(N.LT.4096) ! For all the harmonics in the time domain file:

DO I=1,GWID ! The line will have a width of GWID

CALL GRD ! Make the line wide enough

END DO

N=N+1

END DO

N=N-GWID ! Figure out the increment to the next set of harmonics

N=N+INC

END DO

10 FORMAT(' How many nanoseconds per division?')

10 FORMAT(F10.2)

READ(COM UNIT,33) REPLY ! get answer

IF (REPLY.EQ. 'Y') THEN ! if yes then

WRITE(IOUT,34) ! ask for filename

READ(COM UNIT,33) PROCNAME ! read in file name

GOTO 58

ELSE

DEFINE FLAG=.TRUE. ! set definition flag

WRITE(IOUT,35) ! instruction prompt

WRITE(IOUT,36)

WRITE(IOUT,37)

WRITE(IOUT,38)

WRITE(IOUT,39)

WRITE(IOUT,40)

WRITE(IOUT,41)

WRITE(IOUT,42)

CALL LOGGER ! make a definition file of the commands

PROCNAME= 'USER2:[DURAL.CJAM2]PROC.DAT' ! make the

PROC: This command allows the user to specify a list of files to be processed and how these files are to be processed. This facilitates the processing of large amounts of data.

ENTRY PROC

PROC_FLAG=.TRUE. ! tell the rest of the

C program that Proc is being used

WRITE(IOUT,31) ! does the user have a def. file already?

WRITE(IOUT,32) ! get answer

READ(COM UNIT,33) REPLY

IF (REPLY.EQ. 'Y') THEN ! if yes then

WRITE(IOUT,34) ! ask for filename

READ(COM UNIT,33) PROCNAME ! read in file name

GOTO 58

ELSE

DEFINE FLAG=.TRUE. ! set definition flag

WRITE(IOUT,35) ! instruction prompt

WRITE(IOUT,36)

WRITE(IOUT,37)

WRITE(IOUT,38)

WRITE(IOUT,39)

WRITE(IOUT,40)

WRITE(IOUT,41)

WRITE(IOUT,42)

CALL LOGGER ! make a definition file of the commands

PROCNAME= 'USER2:[DURAL.CJAM2]PROC.DAT' ! make the

c definition file

END IF

12 FORMAT(' Do you have a procedure definition file')

13 FORMAT( ' for this process already?(Y or N)')

14 FORMAT( 'Filename:')

15 FORMAT( 'Enter the process using regular commands and')

16 FORMAT( 'NAME.DAT for a filename. For the new')

17 FORMAT( 'filename use NEWNAME.DAT. When finished')

18 FORMAT( 'defining the process use the command DONE. ')

19 FORMAT( 'Warning: a filename must be listed for ')

20 FORMAT( 'each time it is used. ')

RETURN

ENTRY PROC2 ! reentry after DONE command

CALL STO LOG ! close up the definition file

DEFINE FLAG=.FALSE. ! reset definition flag

WRITE(IOUT,55) ! ask whether he wants to save def.
READ(COM_UNIT,56)  REPLY
IF(REPLY .EQ. 'Y') THEN
WRITE(IOUT,57)
READ(COM_UNIT,56) NAME
COMMAND(1:33) = 'SCOPY USER2:[DURAL.CIMAG2]PROC.DAT' ! the first half of the command
COMMAND(34:75) = NAME ! the filename that the copy is output to
CALL LIB$SPAWN( COMMAND ) ! spawn a process to copy the file
END IF

FORMAT(' Do you wish to save this procedure definition?(Y or N)')
FORMAT(' Filename:')
WRITE(IOUT,59)  REPLY
IF(REPLY .EQ. 'Y') THEN
WRITE(IOUT,60)
READ(COM_UNIT,75) LISTNAME
CALL OPENER(LIST_UNIT,LISTNAME)
CLOSE(LIST_UNIT) ! close the unit
ELSE
LISTNAME = 'USER2:[DURAL.CIMAG2]LIST.DAT'
CALL OPENER(LIST_UNIT,LISTNAME)
WRITE(IOUT,61)
WRITE(IOUT,65)
WRITE(IOUT,70)
READ(COM_UNIT,75) NAME
DO WHILE(NAME .NE. 'DONE')
WRITE(LIST_UNIT,75)
NAME
READ(COM_UNIT,75) NAME
END DO
CLOSE(LIST_UNIT)
WRITE(IOUT,71)
READ(CON_UNIT,75)
REPLY
IF(REPLY .EQ. 'Y') THEN
WRITE(IOUT,60) ! task for file name
COMMAND(1:34) = 'SCOPY USER2:[DURAL.CIMAG2]LIST.DAT'
COMMAND(35:75) = NAME ! the filename that the copy is output to
CALL LIB$SPAWN( COMMAND ) ! spawn a process to copy the file
END IF

FORMAT(' Do you have a data list file?(Y or N)')
FORMAT(' Filename:')
WRITE(IOUT,59)
READ(COM_UNIT,75) LISTNAME
CALL OPENER(LIST_UNIT,LISTNAME)
CLOSE(LIST_UNIT)
WRITE(IOUT,60)
READ(COM_UNIT,75) REPLY
IF(REPLY .EQ. 'Y') THEN
WRITE(IOUT,60)
READ(COM_UNIT,75) OUTNAME
CALL OPENER(NEW_UNIT,OUTNAME)
WRITE(IOUT,80)
WRITE(IOUT,85)
ELSE
OUTNAME = 'USER2:[CIMAG2]NEWLIST.DAT'
CALL OPENER(NEW_UNIT,OUTNAME)
WRITE(IOUT,80)
WRITE(IOUT,85)
END IF

WRITE(IOUT,76)  REPLY
READ(COM_UNIT,75) REPLY
IF(REPLY .EQ. 'Y') THEN
WRITE(IOUT,77)
READ(COM_UNIT,75) OUTNAME
CALL OPENER(NEW_UNIT,OUTNAME)
CLOSE(NEW_UNIT)
ELSE
OUTNAME = 'USER2:[CIMAG2]NEWLIST.DAT'
CALL OPENER(NEW_UNIT,OUTNAME)
WRITE(IOUT,80)
WRITE(IOUT,85)
END IF

WRITE(IOUT,76)  REPLY
READ(COM_UNIT,75) REPLY
IF(REPLY .EQ. 'Y') THEN
WRITE(IOUT,77)
READ(COM_UNIT,75) OUTNAME
CALL OPENER(NEW_UNIT,OUTNAME)
CLOSE(NEW_UNIT)
ELSE
OUTNAME = 'USER2:[CIMAG2]NEWLIST.DAT'
CALL OPENER(NEW_UNIT,OUTNAME)
WRITE(IOUT,80)
WRITE(IOUT,85)
END IF
WRITE(IOUT,90)
READ(COM_UNIT,95) NAME           \* initial read
DO WHILE(NAME .NE. 'DONE')
   WRITE(NEW_UNIT,95) NAME        \* put the new name in file
   READ(COM_UNIT,95) NAME         \* read next file name
END DO
CLOSE(NEW UNIT)
WRITE(IOUT,78)
READ(COM_UNIT,75) REPLY
IF (REPLY .EQ. 'Y') THEN
   WRITE(IOUT,77)
   READ(COM_UNIT,75) NAME        \* the filename
   IF (LISTSTAT .EQ. FOREOF) THEN
      CALL LIB$SIGNAL(MAGCON)
      ELSE IF (LISTSTAT .EQ. 'NEWNAME.DAT') THEN
      END IF
   END IF
   IF(OPER .EQ. 'NAME.DAT') THEN
      CALL LIB$SIGNAL(MAGCOM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC_UNIT,PROCNAME)
OPEN( UNIT=LIST_UNIT,
     + FILE=LISTNAME,
     + STATUS='OLD')
OPEN( UNIT=NEW_UNIT,
     + FILE=OUTNAME,
     + STATUS='OLD')
NAME='USER2:[DURAL.CIMAG2]PROC2.DAT' \* The command procedure
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME          \* Initial read from
C data file
DO WHILE(LISTSTAT .NE. FOR_EOF) \* do while there is still data
   READ( PROC_UNIT , 100 , IOSTAT=PROCSTAT ) OPER   \* initial read
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF) \* do entire definition
      IF( OPER .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( OPER .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
CALL OPENER(PROC2_UNIT,NAME)
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
DO WHILE(LISTSTAT .NE. FOR_EOF)
   READ( PROCBODY , 100 , IOSTAT=PROCSTAT ) PCBODY
C from definition
   DO WHILE(PROCSTAT .NE. FOR_EOF)
      IF( PROCSTAT .EQ. 'NAME.DAT' ) THEN
      END IF
      IF(LISTSTAT .EQ. FOR_EOF) THEN \* if there is
C no filename print error
         CALL LIB$SIGNAL(MAG_COM)
      END IF
END IF
READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
ELSE IF( PROCSTAT .EQ. 'NEWNAME.DAT' ) THEN \* if output
   READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C file is needed
   IF(NEWSTAT .EQ. FOR_EOF) THEN \* if error
      CALL LIB$SIGNAL(MAG_COM)
   END IF
END IF
WRITE(IOUT,97)
**SUBROUTINE OPENER(L_UNIT,NAME)**

This routine opens up a file and gets a LU#

```fortran
INTEGER*4 FOR RETCODE
INTEGER*4 RETCODE
INTEGER*4 SSS_NORMAL
```

END IF
OPER= NEWNAME ! give command procedure the

END IF
WRITE( PROC2_UNIT,100) OPER
READ( PROC_UNIT,100, IOSTAT=PROCSTAT) OPEN ! start definition over
+ FILE=PROCNAME ,
+ STATUS='OLD')

CLOSE( PROC_UNIT)
CLOSE( LIST_UNIT)
CLOSE( NEW_UNIT)
CLOSE( PROC2_UNIT)

CALL LIB$FREE_LUN( PROC_UNIT )
CALL LIB$FREE_LUN( LIST_UNIT )
CALL LIB$FREE_LUN( NEW_UNIT )
CALL LIB$FREE_LUN( PROC2_UNIT )

CALL FILE ! execute the command procedure
RETURN

ENTRY PROC3
IF (PROCNAME .EQ. 'USER2:[DURAL.CIMAG2]PROC.DAT') THEN ! delete all
OPEN(UNIT=1,FILE='PROC.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)
END IF
IF (LISTNAME .EQ. 'USER2:[DURAL.CIMAG2]LIST.DAT') THEN
OPEN(UNIT=1,FILE='LIST.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)
END IF
IF (OUTNAME .EQ. 'USER2:[DURAL.CIMAG2]NEWLIST.DAT') THEN
OPEN(UNIT=1,FILE='NEWLIST.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)
END IF
OPEN(UNIT=1,FILE='PROC2.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)

PROC_FLAG=.FALSE. ! turn off flag
RETURN

END

SUBROUTINE OPENER(L_UNIT,NAME)

ENTRY PROC3
IF (PROCNAME .EQ. 'USER2:[DURAL.CIMAG2]PROC.DAT') THEN ! delete all
OPEN(UNIT=1,FILE='PROC.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)
END IF
IF (LISTNAME .EQ. 'USER2:[DURAL.CIMAG2]LIST.DAT') THEN
OPEN(UNIT=1,FILE='LIST.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)
END IF
IF (OUTNAME .EQ. 'USER2:[DURAL.CIMAG2]NEWLIST.DAT') THEN
OPEN(UNIT=1,FILE='NEWLIST.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)
END IF
OPEN(UNIT=1,FILE='PROC2.DAT',TYPE='OLD',
+ DEFAULTFILE='USER2:[DURAL.CIMAG2]',DISP='DELETE')
CLOSE(UNIT=1)

PROC_FLAG=.FALSE. ! turn off flag
RETURN

END
INTEGER*4 L_UNIT
CHARACTER*70 NAME
PARAMETER (SS$_NORMAL = '00000001'X)

RETCODE=LIB$GET_LUN( L_UNIT )
IF (RETCODE .EQ. SS$_NORMAL) THEN
  OPEN ( UNIT=L_UNIT ,
    FILE=NAME ,
    STATUS='UNKNOWN')
ELSE
  WRITE(IOUT,55)
  CALL LIB$STOP(%VAL(RETCODE))
END IF

SUBROUTINE OUTSIDER:
These are routines are more basic commands 
such as those found in COM_FILE.

SUBROUTINE OUTSIDER

CREATE:
Creates a blank time domain file with a 
header and stores it in the main buffer.

ENTRY CREATE(HEAD, LINE1, LINE2, LINE3)

WRITE(IOUT,*) 'Type in the header:(three lines)'
READ(COM UNIT,10) LINE1
READ(COM UNIT,10) LINE2
READ(COM UNIT,10) LINE3

DO I=1,ISIZE
  P(I,1)=0
END DO
IDS(1)=0
DOMAIN='TIME'
RETURN

CHANGE:
This routine changes a given harmonic in a data
```fortran
ENTRY CHANGE
WRITE(IOUT,30) ! Prompt for the harmonic to be changed
READ(COM UNIT,*) N
WRITE(IOUT,40) (N,P(N,1)) ! Read in the value for the harmonic
WRITE(IOUT,50) I ! Prompt for the new value
ENTRY GRID
P(N,1)-TEMP ! Put the new value in the specified harmonic
FORMAT(X,'Which harmonic do you wish to change?')
30 FORMAT(' Current Value: ',I5,F10.2)
40 FORMAT(' New Value: ') ! The new value
50 FORMAT(F10.2)
60 FORMAT(1E5)
RETURN
END

C FTRAN READ COMMANDS (BY A. DOMINEK)

THE CALLING PROGRAM MUST CONTAIN THE FOLLOWING
COMMON BUFFER,NDIM,ANST,AINC
DIMENSION AMPL(5000),PHS(5000)
BYTE BUFFER(35000)
INTEGER*2 INPFILE(15)
USER SUPPLIES THE FILE NAME
IN FREQUENCY DOMAIN
AMPL CONTAINS AMPLITUDE OF DATA IN DB
PHS CONTAINS PHASE OF DATA IN DEGREES
NDIM IS THE NUMBER OF FREQUENCY SAMPLES
ANST IS THE FREQUENCY (MHZ) OF THE FIRST SAMPLE
AINC IS THE DELTA FREQUENCY (MHZ)
IN TIME DOMAIN
AMPL CONTAINS THE AMPLITUDE OF TIME WAVEFORM
PHS CONTAINS ZERO
NDIM IS THE NUMBER OF TIME SAMPLES = 4096
ANST IS THE STARTING TIME =1/DF/2 (DF=DELTA FREQUENCY)
AINC IS THE DELTA TIME * 1.E5 =1/DF/4096*1.E5

SUBROUTINE FREA(INPFILE,AMPL,PHS)

PROGRAM NAME :USER1:[DOMI][REW FOR
THIS PROGRAM READS BACKSCATTER DATA FILES STORED
ON VAX DISCS. WITH 11/23 FORMAT

INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
include 'USER2:[DURAL.CIMAG2]FTRN.FOR'

INTEGER*2 LIN1(30),LIN2(30),PARAM(30)
REAL*4 AP(10000)

DEFINE BUFFER STRUCTURE

EQUIVALENCE(LIN1(1),BUFFER(1)),(LIN2(1),BUFFER(61))
1,(PARAM(1),BUFFER(121)),(AP(1),BUFFER(361))
EQUIVALENCE(LINE1(1),LIN1(1)),(LINE2(1),LIN2(1))
1,(LINE3(1),PARAM(1))
```
READ A FILE

CALL TTR(INPFILE)
TYPE 105,LIN1
TYPE 105,LIN2
TYPE 105,PARM

GET NUMERICAL INFORMATION FROM THE THIRD LINE
OF THE HEADER

CALL DDCDE

DIVIDE AN AMP-PHASE ARRAY INTO
AN AMP ARRAY AND A PHASE ARRAY

DO 199 NN=1,NDIM
AMPL(NN)=AP(2*NN-1)
AMPL(NN)=35.
199
PHS(NN)=AP(2*NN)

CARRY INFORMATION TO 'CIMAG' (TIME DOMAIN ONLY)
DO 1 I=1,NDIM
P(I,1)=AMPL(I)

CHECK FOR BAD DATA POINTS, I.E., AMPL(I).GT.995

CALL ERRF(AMPL,PHS)
RETURN
END

SUBROUTINE REU(INPFILE,AMPL,PHS)
THIS PROGRAM READS BACKSCATTER DATA FILES STORED
ON VAX DISKS. WITH 750 FORMAT

INCLUDE USER2: [DURAL.CIMAG2]MAGCN2.FOR
INCLUDE USER2: [DURAL.CIMAG2]JAGCN2.FOR
INCLUDE USER2: [DURAL.CIMAG2]FTRN.FOR

BYTE PARAM(60)
INTEGER*2 L1(30),L2(30),L3(30)
* AL*4 AP(10000)
EQUIVALENCE (L3(1),PARAM(1))
FORMTALF(1),BUFFER(1)),(L2(1),BUFFER(61))
1,(L3(1),BUFFER(121)),(AP(1),BUFFER(361))
WRITE(IOUT,2)
2 FORMAT('S','ENTER DATA FILE NAME: ')READ(COM UNIT,10)INPFILE
10 FORMAT(15A2)
WRITE(LOG UNIT,10)INPFILE
INPFILE(15)=0
OPEN(UNIT=8,NAME=INPFILE,TYPE='OLD',FORM='UNFORMATTED',
READONLY,ERR=1)
READ(8)L1
READ(8)L2
READ(8)L3
TYPE 100,L1
TYPE 100,L2
TYPE 100,L3
100 FORMAT(X,30A2)
DECODE(4,4,PARAM(3),ERR=1001)NDIM
DECODE(S,5,PARAM(11),ERR=1002)IANST
IANST=FLOAT(IANST)
DECODE(S,5,PARAM(20),ERR=1003)IAINC
IAINC=FLOAT(IAINC)

4 FORMAT(14)
5 FORMAT(15)
DO 200 I=1,NDIM
READ(8) AMPL(I),PHS(I)
 IO=2*I-1
 IE=2*I
200 AP(IO)=AMPL(I)
 AP(IE)=PHS(I)
CLOSE (UNIT=8,DISP='SAVE')
CALL ERRF(AMPL,PHS,NDIM)
GO TO 99

1001 WRITE(IOUT,*) ' DECODE ERROR NDIM'
GO TO 99
1002 WRITE(IOUT,*) ' DECODE ERROR ANST'
GO TO 99
1003 WRITE(IOUT,*) ' DECODE ERROR AINC'

RETURN
END

SUBROUTINE TTR(INPFILE)
BYTE TBUFF(512)
INCLUDE 'SYSSLIBRARY:FORIOSDEF'
INCLUDE 'USER2:[DURAL.CIAG2]MAGCMN.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
INCLUDE 'USER2:[DURAL.CIAG2]FTRN.FOR'

WRITE( 1OUT, 1111)
1111 FORMAT(1X,' ENTER DATA FILE NAME:')
READ(COM_UNIT,2222) INPFILE
2222 FORMAT(1A2)
WRITE(LOG_UNIT,2222) INPFILE
INPFILE(15)=0
IF(IB.EQ.1)LEN=512-9*4
IF(IB.GT.1)LEN=512-26*4
OPEN(UNIT=8,NAME=INPFILE,READONLY,TYPE='OLD',IOSTAT=IERR,ERR=8100)
8106 SET BLOCK LENGTH IN BYTES
IF(IB.EQ.1)LEN=512-9*4
IF(IB.GT.1)LEN=512-26*4
READ A BLOCK OF 512 BYTES
80 FORMAT(512A1)
STORE A BLOCK INTO THE BUFFER ACCORDING TO ITS LENGTH
85 I=1,LEN
BUFFER(ICNT+1)=TBUFF(I)
IB=IB+1
ICNT=ICNT+LEN
GO TO 82
90 DO 86 I=1,LEN
86 BUFFER(ICNT+1)=TBUFF(I)
ELIMINATE BLANK SPACES IN BETWEEN EACH CHARACTER
IN A FILE HEADER
77
DO 40 I=1,180
  BUFFER(I)=BUFFER(2*I-1)
  GO TO 331
40
  IF(IERR.EQ.FORSIOS_FILENOTFOU) THEN
    WRITE(IOUT,1112) INPFILE
    1112 FORMAT( ' FILE ',15A2, ' WAS NOT FOUND',/, ' $',
    1 ' ENTER FILENAME AGAIN: ')
  ELSE IF (IERR.EQ.FORSIOS_FILENAMSPE) THEN
    WRITE(6,1113) INPFILE
    1113 FORMAT( ' FILE ',15A2, ' WAS BAD, ENTER NEW FILENAME: ')
  ELSE
    TYPE *, 'UNRECOVERABLE ERROR, CODE=', IERR
    STOP
  ENDIF
  GO TO 6
331
  CLOSE(UNIT=8,DISP='SAVE')
  RETURN
END

SUBROUTINE DDCDE
  INTEGER*4 IMIN,IINC,NDIM
  INCLUDE 'USER2: [DURAL.CIMAG2] MAGCMN.FOR'
  INCLUDE 'USER2:[DURAL.CIMAG2] MAGCMN2.FOR'
  INCLUDE 'USER2: [DURAL.CIMAG2] FTRN.FOR'
C
C NO OF DATA POINTS IS STORED IN FOUR CHARACTERS, AND
STATING ANGLE AND ANGLE INC. IN 5 CHARACTERS
C
CHARACTER*3 CNL1
CHARACTER*4 CNL
CHARACTER*5 CFF,CINC
CHARACTER*1 ECH,TCAS
DATA ECH,ZERO/'-','0'/
EQUIVALENCE (BUFFER(123),CNL1),(BUFFER(131),CFF),(BUFFER(140),CINC)
EQUIVALENCE (BUFFER(123),TCAS),(BUFFER(124),CNL1)
C
C CONVERT CHARACTERS INTO THEIR NUMERICAL EQUIVALENTS
C
  IF(ECH.EQ.TCAS) THEN
    DECODE(3,102,CNL1,ERR=9)NDIM
  ELSE
    DECODE(4,101,CNL,ERR=9)NDIM
  END IF
  100 FORMAT(15)
  101 FORMAT(14)
  102 FORMAT(13)
  103 DECODE(5,100,CFF,ERR=99)IMIN
  104 DECODE(5,140,CINC,ERR=999)IINC
  ANST=FLOAT(IMIN)
  AINC=FLOAT(IINC)
  RETURN
C
9  WRITE (6,200) 'NDIM'
  READ (5,*,ERR=91) NDIM
  GO TO 103
C
99  WRITE (6,200,ERR=991) 'ANST'
  READ(5,*) ANST
  GO TO 104
C
999 WRITE (6,200,ERR=9991) 'AINC'
  READ(5,*) AINC
  RETURN
SUBROUTINE ERRF(AMPL,PHS)
COMPLEX C1,C2,CD
INCLUDE 'USER2:[DURAL.CIMAG2]KAGCMN.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]M4AGCMN2.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
DO 1 I=1,NDIM
IF(PHS(I).GT.995.) WRITE(6,2) X,AMPL(I),PHS(I)
1 CONTINUE
2 FORMAT(IX,16ERROR AT DATA PT,114.4HMAG-,1F10.4,4HPHS-,1F10.4)
C CHECK LEFT HAND END POINT
IF(AMPL(I).GT.100.) THEN
DO 200 I=2,NDIM
IF(AMPL(I).LE.100.,AND. AMPL(I+1).LE.100.) THEN
C1=CMPLX(A1*COSD(PHS(I)),A1*SIND(PHS(I)))
A2=10.**(AMPL(I+1)/20.)
C2=CMPLX(A2*COSD(PHS(I+1)),A2*SIND(PHS(I+1)))
CD=C1-C2
RD-REAL(CD)
AD-AIMAG(CD)
DO 212 II=I-1,1-1
RC-REAL(C1)+RD*II
AC-AIMAG(C1)+AD*II
AMPL(II)=20.*LOG10(SQRT(RC*RC+AC*AC))
PHS(II)=ATAN2D(AC,RC)
212 CONTINUE
GO TO 211
ELSE
END IF
ELSE
END IF
C CHECK RIGHT HAND END POINT
211 IF(AMPL(NDIM).GT.100.) THEN
DO 220 I=1,NDIM
J=NDIM-I
IF(AMPL(J).LE.100.,AND. AMPL(J-1).LE.100.) THEN
A1=10.**(AMPL(J)/20.)
C1=CMPLX(A1*COSD(PHS(J)),A1*SIND(PHS(J)))
A2=10.**(AMPL(J-1)/20.)
C2=CMPLX(A2*COSD(PHS(J-1)),A2*SIND(PHS(J-1)))
CD=C1-C2
RD-REAL(CD)
AD-AIMAG(CD)
DO 222 II=J+1,NDIM
RC=REAL(C1)+RD*(II-J)
AC=AIMAG(C1)+AD*(II-J)
AMPL(II)=20.*LOG10(SQRT(RC*RC+AC*AC))
PHS(II)=ATAN2D(AC,RC)
222 CONTINUE
END IF
END IF
END

SUBROUTINE ERRF(/,'$','HAVING PROBLEMS READING HEADER. ENTER ',A4, '
1 MANUALLY: ')
CONTINUE
GO TO 221
ELSE
END IF
CONTINUE
ELSE
END IF

CHECK INTERIOR POINTS
DO 230 I=2,NDIM-1
IF(AMPL(I).GT.100.) THEN
DO 240 K=I+1,NDIM
IF(AMPL(K).LE.100.) THEN
A1=10.**((AMPL(K)-1)/20.)
C1=CMPLX(A1*COSD(PHS(I-1)),A1*SIND(PHS(I-1)))
A2=10.**((AMPL(I))/20.)
C2=CMPLX(A2*COSD(PHS(K)),A2*SIND(PHS(K)))
CD=(C1-C2)/(K-I+1)
RD=REAL(CD)
AD=AIMAG(CD)
DO 241 II=I,K-1
RC=REAL(C1)+RD*(II-I+1)
AC=AIMAG(C1)+AD*(II-I+1)
AMPL(II)=20.*LOG10(SQRT(RC*RC+AC*AC))
PHS(II)=ATAN2D(AC,RC)
CONTINUE
GO TO 230
ELSE
END IF
CONTINUE
ELSE
END IF
CONTINUE
RETURN
END
COMMON BLOCKS

INTEGER*2 HEAD(256, 31), LINE1(30), LINE2(30), LINE3(30)
INTEGER*2 BUFFERS(400, 30)
INTEGER*4 NB, ISIZE
INTEGER*4 NNUM, ELEMENT, ELEM
REAL VALUE, PVALUE, WEIGHT
LOGICAL EFLAG
LOGICAL ECHO
CHARACTER*4 DOMAIN, DOM(40)
CHARACTER*50 INAME, JNAME, FNAME
CHARACTER*60 TITLE, XLAB, YLAB
CHARACTER*40 FNAME
DIMENSION P(4096, 31), IDS(31), S(1024), Q(4096), ARRAY(3, 100, 100)
DIMENSION NUMB(3), ANG(3, 400), BUFN(3, 400), CNTR(3, 400)
DIMENSION CARRAY(4, 8), CLRTAB(6), ARRAY2(3, 100, 100)
COMPLEX A(4096), CCI, CCX, BCX, DCX
COMPLEX BA(201), KLA, CA(2049), ASQ, ASS1, ASS2, ASD1, ASC, ASD, ASA, ASB
CHARACTER*20 INFILE
REAL FA(2049)
BYTE MACRO(128)
INTEGER*2 BFILE(6)

COMMON /BLK1/ INAME, JNAME, FILNM, TITLE, XLAB, YLAB, P, IDS, S, Q,
+ ARRAY, NUMB, ANG, BUFN, CNTR, CARRAY, CLRTAB,
+ ASS2, ASD1, ASC, ASD, ASA, ASB, INFILE, FA,
+ MACRO, BFILE, BUFFERS, DOMAIN, DOM, NB, ECHO,
+ ARRAY2, ISIZE, FNAME

The main common block
MAGCMN2.FOR
THIS IS THE PROGRAM CONTROLS COMMON BLOCK

INTEGER*4
+ COM_UNIT , 1 LU$ for the command input
+ IOUT , 1 LU$ for the program output
+ TERM_UNIT , 1 LU$ for the terminal
+ FILE_UNIT , 1 LU$ for the command file
+ LOG_UNIT , 1 LU$ for the log file
+ NULL_UNIT , 1 LU$ of null device for logging routine
+ STO_UNIT , 1 LU$ for buffer storage

CHARACTER*40 DEF 1 default directory
LOGICAL PROC_FLAG
LOGICAL DEFINE_FLAG

COMMON/BLK2/COM_UNIT, IOUT, TERM_UNIT, FILE_UNIT,
+ DEF, LOG_UNIT, NULL_UNIT, PROC_FLAG,
+ DEFINE_FLAG
COMMON BUFFER, NDIM, ANST, AINC, FTYPE
DIMENSION AMPL(5000), PHS(5000)
BYTE BUFFER(35000)
INTEGER*2 INFILE(15)
LOGICAL FTYPE
DEFINE THE TAPE HEADER FIELDS

BYTE ITITLE(50) ITITLE
INTEGER*2 IHED(6) IDATE AND TIME AS MONTH,DAY,YEAR
HOURS,MINUTES,SECONDS

BYTE ITARG(38) ITARGET LABEL
INTEGER*2 IANG(3) ISTARTING ANGLE, ANGLE INCREMENT FOR ROTATION
AND NUMBER OF ANGLES IN WHOLE FILE

BYTE ITYPE(6) IACOUSTIC DATA TYPE PCW,PLFM
INTEGER*2 IPARAM(6) IFREQUENCY IN KHZ, SAMPLE INTERVAL, INTERVAL
AND NUMBER OF PINGS AT A GIVEN ANGLE

INTEGER*2 IDEANG IELEVATION/DECLINATION ANGLE
INTEGER*2 IPULTH IPULSE LENGTH - DIGITS ONLY
INTEGER*2 IPUNIT IUNITS FOR PULSE LENGTH
INTEGER*2 IMODFW IMODULATION BANDWIDTH FOR PLFM
INTEGER*2 ISTRING ISOUCE-TARGET RANGE IN METERS * 100
INTEGER*2 ISRRNG ISOUCE-RECEIVER RANGE IN METERS * 100
INTEGER*2 IXMTVL IRMS TRANSMIT VOLTAGE * 100
INTEGER*2 IRCVGN IRECEIVER GAIN IN DB * 10
INTEGER*2 IFTRBW IRECEIVE FILTER 3-DB BW IN KHZ
BYTE IPROJ(20) IPROJECTOR DESCRIPTION
INTEGER*2 ITRV IXMIT LEVEL OF PROJECTOR IN DB/MICRO PA/V*10
BYTE IHYD(20) IRECEIVER DESCRIPTION
INTEGER*2 IRSRS IRECEIVER SENSITIVITY IN DB/V/MICRO PA*10
INTEGER*2 IDATR INUMBER OF BIOMATION SAMPLES IN A SINGLE PING

INTEGER*2 HEADER(256)
COMMON /HEADER/ ITITLE,IHED,ITARG,IANG,ITYPE,IPARAM,IDEANG,IPULTH,
IPUNIT,IMODFW,ISTRING,ISRRNG,IXMTVL,IRCVGN,IFTRBW,
IPROJ,ITRV,IHYD,IRRS,IDATR,IOOTHER
EQUIVALENCE (HEADER,ITITLE)
APPENDIX C

CIMAG2 LINKING SUBROUTINES

SUBROUTINE FRD1(P,EFLAG)
LOGICAL EFLAG
INTEGER*2 HEAD(256)
REAL P(4096,7),AM(2049),PH(2049),FA(2049)
COMMON/HEADER/HEAD
ISIZE=4096
ISZ=ISIZE/2.
TYPE *, 'FREQUENCY SAMPLING(1) OR .1KL SAMPLING(0) ?'
READ (COM UNIT,*) IFS
WRITE (LOG UNIT,*) IFS I
IF (IFS.EQ.0) TYPE *, 'INPUT MAJOR AXIS DIMENSION IN INCHES'
READ (COM UNIT,*) DLE
WRITE (LOG UNIT,*) DLE I
IF (IFS.EQ.1) TYPE *, 'FREQUENCY INCREMENT IN MHZ'
READ (COM UNIT,*) DLE
WRITE (LOG UNIT,*) DLE I
IF (IFS.EQ.1 ) DLE = 590.551/(PI*DLE)
TYPE *, 'SELECT THE TYPE OF INTERPOLATION'
TYPE *, 'INPUT 0 --- TWO-POINT INTERPOLATION ; NO SMOOTHING'
TYPE *, 'INPUT 1 --> INTERPOLATION AND SMOOTHING USING A COSINE WINDOW'
READ (COM UNIT,*) IS
WRITE (LOG UNIT,*) IS I
IF (IS.GT.2.OR.IS.LT.0) GO TO 5
DLE = DLE*2.54/100.*PI-4.*ATAN(1.)
RLOW = TWOPR = TWOPR + FLOW*DLE/300.
RHIGH = TWOPR + FHIGH*DLE/300.
RINC = 1
WD = TWOPR + FINC*DLE/300.*6.
DO 5 I=IS2,IS2
P(I,1)=0
DO 10 I=I,NP
FA(I)=(FLOW*(I-1)*FINC)*TWOPR*DLE/300.
CALL GP(INORM,IS,AM,PH,FA,NP,WD,DLE,LOW,HIGH,RINC,P)
RETURN
IF (EFLAG) THEN I
IF an error occurred
END IF
TWOPI = 8.*ATAN(1.)
LOW = TWOPI*FLOW*DLE/300.
HIGH = TWOPI*FHIGH*DLE/300.
RINC = 1
WD = TWOPI + FINC*DLE/300.*6.
DO 5 I=IS2,IS2
P(I,1)=0
DO 10 I=I,NP
FA(I)=(FLOW*(I-1)*FINC)*TWOPR*DLE/300.
CALL GP(INORM,IS,AM,PH,FA,NP,WD,DLE,LOW,HIGH,RINC,P)
RETURN
END

SUBROUTINE GP(INORM,IS,AM,PH,FA,NP,WD,DLE,LOW,HIGH,RINC,P)
REAL*4 P(2049),FA(2049),P(4096,7),AM(2049),PH(2049)
COMPLEX*8 R,CA(2049)
PI = 4. * ATAN(1.)
DO 111 I=1,NP
ATMP = PH(I)/180.*PI
CA(I) = CMPLX(ATMP*COS(PTMP),ATMP*SIN(PTMP))
TWOPR = PI*2.
I=1
DO 7740 RF=LOW,HIGH, RINC
7740 IF(RF.GE.FA(I).AND.RF.LE.FA(I+1)) GO TO 7730

85
IF(I.GT.NP)GO TO 4912
GO TO 7740
7730
R=R+10.
IF(K.GT.2048)GO TO 4912
IF(IS.EQ.I)CALL INTER(R,CA,FA,NP,NS,I,WD,RF)
IF(INORM.EQ.0)R=R*0.02/(SQRT(PI)*DLE)
P(K+2048,I)=0.24.*ATAN2(AIMAG(R),REAL(R))/TWOPI
CONTINUE
4912 RETURN
C
C SUBROUTINE INTER(R,CA,FA,ICNT,NS,I,WD,RF)
COMPLEX*8 R,CA(2049)
REAL*4 FA(2049)
IS=I
XTMP=0.
YTMP=0.
RFL=RF-WD/2.
RFH=RF+WD/2.
20 IF(IS.GT.ICNT)GO TO 10
IF(FA(IS).GT.RFH)GO TO 10
T=FA(IS)-RF
HAMM=54.46+COS(3.1415926*T/WD)
XTMP=XTMP+HAMM*REAL(CA(IS))
YTMP=YTMP+HAMM*AIMAG(CA(IS))
WEI=WEI+HAMM
IS=IS+1
GO TO 20
10 IS=I-1
15 IF(IS.LT.1)GO TO 30
IF(FA(IS).LT.RFL)GO TO 30
T=FA(IS)-RF
HAMM=54.46+COS(3.1415926*T/WD)
XTMP=XTMP+HAMM*REAL(CA(IS))
YTMP=YTMP+HAMM*AIMAG(CA(IS))
WEI=WEI+HAMM
IS=IS-1
GO TO 15
30 R=CMPLX(XTMP/WEI,YTMP/WEI)
RETURN
END
C
C SUBROUTINE PCHR(DPH,FB,FINC,PH)
PI=3.1415926
30 IF(DPH.LT.PI)GO TO 20
DPH=DPH+PI
GO TO 30
20 IF(DPH.GT.-PI)GO TO 40
DPH=DPH-PI
GO TO 20
40 DO 10 I=1,201
FA=FB+(I-1)*FINC
10 PH(I)=PH(I)-(DPH*FA/FB)
RETURN
END
C
C SUBROUTINE EPH(PHM,PH,NS,NF,NI)
REAL*4 PH(201)
WEI=0
PHM=0
WD=2.*(NF-NS)
DO 10 I=NS,NF,NI
HAMM=.54+.46*COS(3.1415926*(I-NS)/WD)
WEI=WEI+HAMM
10
PHM=PHM+HAMM*PH(I)
PHM=PHM/WEI
RETURN
END
SUBROUTINE RDFLE(A,P,NP,RMIN,FMAX,FINC,COM_UNIT,EFLAG)

CHARACTER*1 YN
LOGICAL EFLAG
COMMON BUFF
BYTE BUFF(20000)
INTEGER*2 LINE1(30),LINE2(30),PARAM(30),OFILE(15)
INTEGER*4 COM_UNIT
CHARACTER*70 INFILE
REAL*4 AP(4096),A(2049),P(2049)
INTEGER*2 HEADR(256)
COMMON/HEADER/HEADR

DEFINE BUFFER STRUCTURE

CALL TR(INFILE.,IB,EFLAG)
IF ( EFLAG ) THEN I
  if error has occurred
  RETURN
END IF

TYPE 105,LINE1
TYPE 105,LINE2
TYPE 105,PARAM

FORMAT(X,30A2)

PUT INFO IN HEADR BLOCK FOR TRANSFER TO CALLING PROG

DO 108 I=1,30
  HEADR(I)=LINE1(I)
  HEADR(30+I)=LINE2(I)
  HEADR(60+I)=PARAM(I)

GET NUMERICAL INFORMATION FROM THE THIRD LINE
OF THE HEADER

CALL DCDE(NP,FMIN,FINC,EFLAG)
IF ( EFLAG ) THEN I
  if error return
  RETURN
END IF

DIVIDE AN AMP-PHASE ARRAY INTO
AN AMP ARRAY AND A PHASE ARRAY

DO 199 NN=1,NP
  A(NN)=AP(2*NN-1)
  P(NN)=AP(2*NN)
  FORMAT(A1)
  FORMAT(1X,5(2F2.3,L;))
  FMAX-FMIN+(NP-1)*FINC
  RETURN
END
SUBROUTINE TR(INFILE,IB,EFLAG)
INCLUDE 'MAGCMN2.FOR'
LOGICAL EFLAG
INTEGER*2 IBUFF(10000)
CHARACTER*70 INFILE
INTEGER*4 FOR_RETCODE
COMMON BUFF
BYTE BUFF(20000),TBUFF(1500)
EQUIVALENCE (BUFF(I),IBUFF(I))
INCLUDE 'SYS$LIBRARY:FORIOSDEF'
WRITE(6,5)
5 FORMAT(IX,'TYPE DATA FILE NAME')
READ (COM UNIT,10) INFILE
WRITE (LOG UNIT,10) INFILE
IF ((INFILE .EQ. 'NAME.DAT') .AND. DEFINE FLAG) THEN
   INFILE='USER2:[DURAL.CIMAG2]NAME.DAT'
   IF (FOR_RETCODE .EQ. 29) THEN
      CALL LIB$SIGNAL(MAGFILNOTFOU)
      EFLAG = .TRUE.
      RETURN
   ELSE
      CALL LIB$SIGNAL(MAG_CON)
      EFLAG = .TRUE.
      RETURN
   END IF
END IF
10 FORMAT( A )
IB=1
ICNT=0
OPEN(UNIT=8,NAME=INFILE,TYPE='OLD',READONLY,IOSTAT=FOR_RETCODE,
     DEFAULTFILE=DEF,ERR=81)
GOTO 82
81 IF (FOR_RETCODE .EQ. 29) THEN
   CALL LIB$SIGNAL(MAG_FILNOTFOU)
   EFLAG = .TRUE.
   RETURN
ELSE
   CALL LIB$SIGNAL(MAG_CON)
   RETURN
END IF
C
C SET BLOCK LENGTH IN BYTES
C 82 IF(IB.EQ.1)LEN=512-9*4
   IF(IB.GT.1)LEN=512-26*4
C
C READ A BLOCK OF 512 BYTES
C 80 READ(8,80,END=90)(TBUFF(I),I=1,512)
80 FORMAT(512AI)
C
C STORE A BLOCK INTO THE BUFFER ACCORDING TO ITS LENGTH
C DO 85 I=1,LEN
   BUFF(ICNT+I)=TBUFF(I)
   ICNT=ICNT+LEN
   GO TO 82
85 DO 86 I=1,LEN
   BUFF(ICNT+I)=TBUFF(I)
86 DO 86 I=1,LEN
C
C ELIMINATE BLANK SPACES IN BETWEEN EACH CHARACTER
C IN A FILE HEADER
C DO 40 I=1,180
   BUFF(I)=BUFF(2*I-1)
40 CLOSE(UNIT=8,DISP='SAVE')
RETURN
C
C SUBROUTINE DCDE(NP,FMIN,FINC,EFLAG)

89
COMMON BUFF
BYTE BUFF(20000)
INTEGER*4 IMIN,IINC,NP

NO OF DATA POINTS IS STORED IN THREE CHARACTERS, AND
STARTING FREQ AND FREQ INC. IN 5 CHARACTERS

CHARACTER*4 CNL
CHARACTER*5 CFF,CINC
INTEGER*4 FOR.RETCODE    ! fortran return code
LOGICAL EFLAG
INCLUDE 'USER2:[DURAL.CIMAG2]MSGBLK.FOR'    ! error messages

EQUVALENCE (BUFF(123),CNL),(BUFF(131),CFF),(BUFF(140),CINC)

CONVERT CHARACTERS INTO THEIR NUMERICAL EQUIVALENTS

    if an equal sign appears as the first character
    change it into a blank.

    IF( CNL(1:1).EQ.'=' ) CNL(1:1) = ' '
    READ(UNIT=CNL, FMT=100, IOSTAT=FOR.RETCODE, ERR=110) NP
    FORMAT(15)
    READ(UNIT=CFF, FMT=100, IOSTAT=FOR.RETCODE, ERR=110) IMIN
    READ(UNIT=CINC, FMT=100, IOSTAT=FOR.RETCODE, ERR=110) IINC
    IMIN-IMIN
    IINC-IINC

110 IF (FOR.RETCODE.EQ.64) THEN
    CALL LIB$SIGNAL(MAG_INPFOR)
    EFLAG = .TRUE.
    ELSE
    CALL LIB$SIGNAL(MAG_COMP)
    EFLAG = .TRUE.
    END IF
RETURN
END
SUBROUTINE FORT(A,M,S,IFS,IFERR)

DIMENSION A(1),S(1),K(15)

IF(M)2,2,3

3 IF(M-14) 5,5,2

2 IFERR=1

1 RETURN

5 IFERR=0

N=2**M

IF( IABS(IFS) - 1 ) 200,200,10

WE ARE DOING TRANSFORM ONLY. SEE IF PRE-COMPUTED

S TABLE IS SUFFICIENTLY LARGE

10 IF( N-NP )20,20,12

12 IFERR=1

GO TO 200

C SCRAMBLE A, BY SANDE'S METHOD

20 K(1)=2*N

DO 22 L=2,M

22 K(L)=K(L-1)/2

DO 24 L=M,13

24 K(L+1)=2

C BINARY SORT

K1=K(14)

K2=K(13)

K3=K(12)

K4=K(11)

K5=K(10)

K6=K(9)

K7=K(8)

K8=K(7)

K9=K(6)

K10=K(5)

K11=K(4)

K12=K(3)

K13=K(2)

K14=K(1)

N2=K(1)

I,J=2

DO 30 J1=2,K1,2

DO 30 J2=J1,K2,K1

DO 30 J3=J2,K3,K2

DO 30 J4=J3,K4,K3

DO 30 J5=J4,K5,K4

DO 30 J6=J5,K6,K5

DO 30 J7=J6,K7,K6

DO 30 J8=J7,K8,K7

DO 30 J9=J8,K9,K8

DO 30 J10=J9,K10,K9

DO 30 J11=J10,K11,K10

DO 30 J12=J11,K12,K11

DO 30 J13=J12,K13,K12

DO 30 J14=J13,K14,K13

IF( I,J1)28,30,30

28 T=A(I,J-1)

A(I,J-1)=A(J1)

A(J1)=T

T=A(I,J)

A(I,J)=A(J1)

A(J1)=T

91
30 1J=1J+2
IF (IPS<32,2,36)
C DOING FOURIER ANALYSIS, SO DIV. BY N AND CONJUGATE.
32 FN = N
DO 34 I=1,N
A(2*I-1) = A(2*I-1)/FN
34 A(2*I-1) = A(2*I-1)/FN
C SPECIAL CASE- L=1
36 DO 40 I=1,N,2
T = A(2*I-1)
A(2*I-1) = T + A(2*I+1)
A(2*I+1) = T - A(2*I+1)
A(2*I) = T + A(2*I+2)
40 A(2*I+2) = T - A(2*I+2)
IF (M-1) 2,1,50
C SET FOR L=2
50 LEXP = 2
C LEXP = 2**(L-1)
LEXP = 8
C LEXP = 2**(L+1)
NPL = 2**-MT
C NPL = NP* 2**-L
60 DO 130 L=2,M
C SPECIAL CASE- J=0 DO 80 I=2,N2,LEXP
I1 = I + LEXP1
I2 = I1 + LEXP1
I3 = I2 + LEXP1
T = A(I-1)
A(I-1) = T + A(I2-1)
A(I2-1) = T - A(I2-1)
T = A(I1)
A(I1) = T + A(I2)
A(I2) = T - A(I2)
T = A(I3)
TI = A(I3-1)
A(I3-1) = A(I1-1) - T
A(I3-1) = A(I1) - TI
A(I1-1) = A(I1-1) + T
80 A(I1) = A(I1) + TI
IF (L-2) 120,120,90
90 KLAST = N2 - LEXP
JJ = NPL
DO 110 J=4,LEXP1,2
NPL = NPL - J
UR = S(NPJJ)
UI = S(JJ)
ILAST = J + KLAST
DO 100 J = J, ILAST, LEXP
I1 = I + LEXP1
I2 = I1 + LEXP1
I3 = I2 + LEXP1
T = A(I2-1) * UR - A(I2) * UI
TI = A(I2-1) * UI + A(I2) * UR
SET = A(I-1)
A(I2) = A(I-1) - TI
A(I1) = A(I1) + TI
A(I) = A(I) * TI
T = A(I3-1) * UR - A(I3) * UI
TI = A(I3-1) * UI + A(I3) * UR
SET = A(I1-1)
A(I3) = A(I1) - TI
A(I1-1) = A(I1-1) + TI
100 A(I1) = A(I1) + TI
C END OF I LOOP
DOING FOURIER ANALYSIS. REPLACE A BY CONJUGATE.

DO 150 I=1,N
150 A(2*I) =-A(2*I)
GO TO 1

RETURN

MAKE TABLE OF S(J)=SIN(2*PI*J/NP),J=1,2,...,NT-1,NT=NP/4

NP=N
MP=M
NT=N/4
MT=M-2
IF(MT) 260,260,205

THETA=PI/2**(L+1) FOR L=1

JSTEP = NT
JDIF = NT/2

JDIF = 2**(MT-L) FOR L=1
S(JDIF) = SIN(THETA)
IF (MT-2)260,220,220
DO 250 L=2,MT
THETA = THETA/2.
JSTEP2 = JSTEP
JSTEP = JDIF
JDIF = JDIF/2
S(JDIF)=SIN(THETA)
JC1=NT-JDIF
S(JC1)=COS(THETA)
JLAST=NT-JSTEP2
IF(JLAST-JSTEP)250,230,230
DO 240 J=JSTEP,JLAST,JSTEP
JC=NT-J
JD=J+JDIF
240 S(JD)=S(J)*S(JC1)+S(JDIF)*S(JC)
CONTINUE
250 CONTINUE
260 IF(IFS)20,1,20
END
SUBROUTINE BSCREA(P,COM UNIT)
COMPLEX ET(1024),EP(1024)
REAL P(4096,9),DT(1024),DP(1024)
CHARACTER*20 INFILE
INTEGER*4 CON
INCLUDE 'SYS$LIBRARY:FORIOSDEF'
PI=3.14159265
TYPE *, INPUT FILE NAME =
READ (COM UNIT,7823) INFILE
7823 FORMAT(A216)
OPEN(UNIT=19,NAME=INFILE,STATUS='OLD',FORM='UNFORMATTED'
2,IOSTAT=IERR,ERR=-8100)
DO 3410 I=1,2048
P(I+2048,1)=0.
3410 P(I,1)=1024.
READ(19) ET
READ(19) EP
READ(19) IB,IE,IS,PLS,PLI
FAC=10.*BLOG10(4.*PI)
DO 40 I=IB,IE,IS
DT(I)=20.*BLOG10(BABS(ET(I)))+FAC
DP(I)=20.*BLOG10(BABS(EP(I)))+FAC
ETM--200.
EPM--200.
ETA=0.
EPA=0.
IT=0
DO 11 I=IB,IE,IS
IF(DT(I).GT.ETM) ETM=DT(I)
IF(DP(I).GT.EPM) EPM=DP(I)
IT=IT+1
ETA=ETA+DT(I)
EPA=EPA+DP(I)
11 CONTINUE
ETA=ETA/IT
EPA=EPA/IT
PLE=PLS/(IE-1)*PLI
TYPE 1,PLS,PLE,PLI
1 FORMAT(' START-',F10.5,5X,'END-',F10.5,5X,'STEP-',F10.5)
TYPE 3,ETM,EPM
3 FORMAT(' ETM-',F10.5,5X,'EPM-',F10.5)
TYPE 2,ETA,EPA
2 FORMAT(' ETA-',F10.5,5X,'EPA-',F10.5)
TYPE 4
4 FORMAT(' ET-',F10.5,5X,'EP-',F10.5,'IPOL-',$',I9)
READ (COM UNIT,*) IPOL
DLE=0.03/(2.*PI*PLI)
TYPE *, 'L=',DLE
TYPE *, 'NORMALIZE TO: SQ CM(1), SQ M(2), PI*L*L/4(0)?'
READ (COM_UNIT,*) INORM
DFAC=0.
IF(INORM.EQ.0) DFAC=-10.*BLOG10(0.25*PI*DLE*DLE)
IF(INORM.EQ.1) DFAC=20.
KLOW=1.5+PLS/PLI
DO 7720 K=I,IE,IS
K=KLOW+I-1
IF(IPOL.EQ.1) THEN
P(K,1)=10.24*(10.+DFAC+DT(I))
P(K+2048,1)=512.*BTAN2(AIMAG(ET(I)),REAL(ET(I)))/PI
ELSE
P(K,1)=10.24*(10.+DFAC+DP(I))
P(K+2048,1)=512.*BTAN2(AIMAG(EP(I)),REAL(EP(I)))/PI
ENDIF

7720 CONTINUE
GO TO 80

8100 IF(IERR.EQ.FOR$IOS_FILENOTFOUND)THEN
TYPE 1112,INFILE
1112 FORMAT(' FILE - ,A20,' DOES NOT EXIST',//
2,' ENTER FILENAME AGAIN')
ELSE IF (IERR.EQ.FOR$IOS_FILENAMEPRESENT)THEN
TYPE *,FILE:' ,INFILE,' WAS BAD, ENTER NEW FILENAME'
ELSE
TYPE *,UNRECOVERABLE ERROR, CODE =',IERR
STOP
ENDIF
GO TO 810

80 CLOSE(UNIT=19,DISP='SAVE')
RETURN
END
INCLUDE 'ESL_ESLROOT:GRF11LIB]BFILES.FOR'
.TITLE CIMAG_MESSAGES Program error messages
.FACILITY CIMAG,283/PREFIX=MAG_

Error Messages
.SEVERITY ERROR
.COM "error in command"
.INPFOR "error in input format"
.FILNOTFOU "file not found in this directory"
.END
APPENDIX D

COLOR IMAGING PROGRAM CLRPL

THIS PROGRAM DEFINES A PLOTTING SOFTWARE CODE TO PLOT IMAGE OF AN OBJECT WITH POLARIZATION INFORMATION

IMAGE DATA SHOULD BE CODED BY A 100*100 MATRIX AND IT SHOULD BE CONTAINED IN THE MATRIX 'ARRAY(100,100)'

DIMENSION ICOL(100,100),ICOLA(100)
DIMENSION GX(5),GY(5),CBX(5),CBY(5),XP(5),YP(5),XT(10),YT(10)
DIMENSION ARRAY(100,100),COORD(5)
INTEGER COLI ,WKSTID
REAL*4 L
LOGICAL FINISHED
CHARACTER*40 FILNM
CHARACTER*7 NUMB
C
C THE GX MATRIX IS THE X COORDINATES OF THE COLOR GRID
DATA GX/.30,.85,.85,.30,.30/
C
C THE GY MATRIX IS THE Y COORDINATES OF THE COLOR GRID
DATA GY/.12,.12,.67,.67,.12/
C
C THE CBX MATRIX IS THE X COORDINATES OF THE COLOR BOX
DATA CBX/.30,.85,.85,.30,.30/
C
C THE CBY MATRIX IS THE Y COORDINATES OF THE COLOR BOX
DATA CBY/.70,.70,.72,.72,.70/
C
C THE XT MATRIX IS THE X COORDINATES OF THE TIK MARKS
DATA XT/.3,.44,.57,.71,.85,.26,.26,.26,.26,.26/
C
C THE YT MATRIX IS THE Y COORDINATES OF THE TIK MARKS ON THE Y AXIS
DATA YT/.08,.08,.08,.08,.08,.12,.26,.40,.54,.67/

PI=4.*ATAN(1.)
WKK=412505
WKSTID=1
C
VMTO=--V Value of Magnitude Top Center Line
VMTCL=CBY(3)+.02
C
ENTER THE ARRAY TO BE PLOTTED
WRITE(6,*)'ENTER FILE NAME'
READ(5,100) FILNM
100 FORMAT(A40)
C
POLARIZATION DEPENDENT PLOT?
WRITE(6,*) 'DO YOU NEED TO USE POLARIZATION AS A PARAMETER?'
WRITE(6,*)'Y=1 N=0'
READ(5,*) POL
IF(POL.EQ.1) THEN
C
ENTER THE POLARIZATION
WRITE(6,*)'ENTER POLARIZATION (VP=1,HP=2)'
READ(5,*) PL
ELSE
END IF

OPEN(UNIT=1,FILE=FILENM,STATUS='OLD',FORM='UNFORMATTED')
ENTER THE IMAGE SIZE AND PERIOD OF THE TIME SIGNAL
READ(1) IMGSZ, PER
ENTER THE IMAGE ARRAY

DO 1000 I=1,100
DO 1000 J=1,100
READ(1) ARRAY(I,J)
ARRAY(I,J)=ABS(ARRAY(I,J))
CONTINUE

NORMALIZE THE ARRAY VALUES
CALL SEARCH(ARRAY,AMAX,AMIN)
WRITE(6,'(E10.4)') 'MAX=', AMAX, 'MIN=', AMIN
WRITE(6,'(A10)') 'ENTER THE DESIRED MAX., AND, MIN.'
READ(5,'(F10.4)') ANMAX, ANMIN
DO 20 I=1,100
DO 20 J=1,100
IF (ARRAY(I,J).GT.ANMAX) ARRAY(I,J)=ANMAX
IF (ARRAY(I,J).LT.ANMIN) ARRAY(I,J)=ANMIN
ARRAY(I,J)=(ARRAY(I,J)-ANMIN)/(ANMAX-ANMIN)
CONTINUE

OPEN GKS ERROR FILE
CALL GOPKS(6,5000)

FIND CONNECT ID
999 CALL GKHGCW('ESL 4129', JERROR, KCONID)
IF (JERROR.NE.0) THEN
WRITE(6,'(A10)') 'Can not be a connection ID'
WRITE(6,'(A10)') 'Would you like to wait ? (Y=1)'
READ(5,'(A1)') ANS
IF (ANS.NE.1) THEN
STOP
ELSE
WRITE(6,'(A10)') 'Enter 1 when ready'
READ(5,'(A1)') ANS
GO TO 999
END IF
END IF

OPEN WORKSTATION #1
CALL GOPWK(1,KCONID,KWK)

ACTIVATE WORKSTATION #1
CALL GACWK(1)

SET THE WORKSTATION WINDOW/VIEWPORT-FULLSCREEN
GET MAX X AND Y KUNITS=0
CALL GQDSP(KWK,KERROR,KUNITS,XSIZE, YSIZE,KRAX, KRAY)
CALL GSWKWN(1,0.,1.0.,YSIZE/ XSIZE)
CALL GSWKVP(1,0.,.343,0.,.274)

GENERATE THE COLOR INDICES

IF (POL.EQ.0) THEN
DO 3 COLI=1,100
L=5
S=1
3 CONTINUE
IF(COLI.LE.5)L=0
H=95+2.6*COLI
CALL HLSRGB(H,L,S,R,G,B)
CALL GSCR(1,1+COLI,R,G,B)
ICOLA(COLI)=1+COLI
3 CONTINUE
ELSE
DO 4 COLI=1,100
L=.5
S=1
FC=25*(4.8)**(COLI/100.)
IF(COLI.LE.5) L=0
IF(PL.EQ.1) H=225-FC
IF(PL.EQ.2) H=110-FC
IF(PL.EQ.2.AND.COLI LT.25.AND.COLI GT.5)L=.75
IF(H.LT.0) H=H+360
CALL HLSRGB(H,L,S,R,G,B)
CALL GSCR(1,1+COLI,R,G,B)
ICOLA(COLI)=1+COLI
4 CONTINUE
END IF
C C GENERATE COLOR CODE FOR X AND Y COORDINATED RCS LEVELS
DO 30 I=1,100
DO 30 J=1,100
ICOL(I,J)=ARRAY(I,J)*99+2
30 CONTINUE
C PLOT COLOR LABEL USING CELL ARRAY
CALL GCRSG(SEG)
CALL GCA(CBX(1),CBY(1),CBX(3),CBY(3),100,1,1,1,100,1,ICOLA)
CALL GSPLCI(1)
CALL GPL(5,CBX,CBY)
C C LABEL COLOR BAR
CALL GSTXAL(2,3)
CALL GSTXP(0)
CALL GSCMXP(1.25)
CALL GSCMSXP(1.)
CALL GTX(GX(1),YMTCL,'0.0')
CALL GTX(.44,YMTCL,'0.25')
CALL GTX(.57,YMTCL,'0.50')
CALL GTX(.71,YMTCL,'0.75')
CALL GTX(GX(2),YMTCL,'1.0')
CALL GSCMXP(1.)
CALL GSCMSXP(1.)
C PLOT COLOR MATIX USING CELL ARRAY
TFST=10
IF (TEST.FQ.0) GO TO 21
CALL GCA(GX(1),GY(1),GX(3),GY(3),100,100,1,1,100,100,1,ICOL)
CALL GSPLCI(1)
CALL GPL(5,GX,GY)
C C DRAW GRID LINE ADJACENT TO EACH AXES
XP(1)=.30
XP(2)=.85
YP(1)=.08
YP(2)=.08
CALL GPL(2,XP,YP)
XP(1)=.26
XP(2)=.26
YP(1)=.12
YP(2)=.67
99
CALL GPL(2,XP,YP)

C PRINT THE TIK MARKS

CALL GSMK(2)
CALL GSMKSC(1)
CALL GSPMCI(1)
CALL GPM(10,XT,YT)

C SHOW THE ASPECT ANGLES
XP(1)=.57
XP(2)=.57
YP(1)=.16
YP(2)=.19
CALL GPL(2,XP,YP)
XP(1)=.565
XP(2)=.57
XP(3)=.575
YP(1)=.185
YP(2)=.19
YP(3)=.185
CALL GPL(3,XP,YP)
XP(1)=.80
XP(2)=.77
YP(1)=.40
YP(2)=.40
CALL GPL(2,XP,YP)
XP(1)=.775
XP(2)=.77
XP(3)=.775
YP(1)=.405
YP(2)=.4
YP(3)=.395
CALL GPL(3,XP,YP)
CALL GTX(.57,.14,'0 DEG.'
CALL GTX(.57,.02,'TIME IN NANOSECONDS'
CALL GSTXAL(2,1)
CALL GSTXP(0)
CALL GSCUP(1,0.)
CALL GTX(.82,.4,'90 DEG.'
CALL GTX(.20,.40,'TIME IN NANOSECONDS'

C FIGURE OUT THE COORDINATES ON EACH AXES
CALL MARKS(IMGSZ,PER,COORD)

C PRINT THE COORDINATES
X=.30
CALL GSCHUP(0.,1.)
DO 40 I=1,5
WRITE(NUMB,FMT='(F6.2)')COORD(I)
CALL GTX(X,.06,NUMB)
40 X=X+.14
CALL GSCHUP(1.,0.)
Y=.12
DO 50 I=1,5
WRITE(NUMB,FMT='(F5.2)')COORD(I)
CALL GTX(.23,Y,NUMB)
50 Y=Y+.14

COLOR 1 = WHITE, 0 = BLACK (ON PLOTTER, REVERSED ON SCREEN)

CALL GCLSG(SEG)

WRITE(6,*)'Enter return to finish.......
READ(5,1)FINISHED
FORMAT(A1)  
CALL GCLRWK(WKSTID,1) !Clears the screen on TEXTRONIX  
C  
DEACTIVATE THE WORK STATION  
CALL GDARK(1)  
C  
CLOSE THE WORK STATION  
CALL GCLWK(1)  
C  
CLOSE THE SYSTEM  
CALL GCLKS  
STOP  
END  

THIS ROUTINE MAKES THE TRANSFORMATION BETWEEN COLOR SYSTEMS  
SUBROUTINE HLSRGB(H,L,S,R,G,B)  
REAL H,L,S,R,G,B,M1,M2  
IF (L.LE.0.5) THEN  
   M1=2*L-M2  
ELSE  
   M1=2*L-M2  
8  
END IF  
M2=L*(1+S)  
IF (L.LE.0.5) THEN  
   M1=2*L-M2  
ELSE  
   M1=2*L-M2  
END IF  
B=rgb_value(M1,M2,H+120)  
R=rgb_value(M1,M2,H)  
G=rgb_value(M1,M2,H-120)  
RETURN  
END  

FUNCTION rgb_value(N1,N2,HUE)  
REAL rgb_value,N1,N2,HUE  
IF (HUE.GT.360) THEN  
   HUE=HUE-360  
END IF  
IF (HUE.LT.0) THEN  
   HUE=HUE+360  
END IF  
IF (HUE.LT.60) THEN  
   rgb_value=N1+(N2-N1)*HUE/60  
ELSE IF (HUE.LT.180) THEN  
   rgb_value=N2  
ELSE IF (HUE.LT.240) THEN  
   rgb_value=N1+(N2-N1)*(240-HUE)/60  
ELSE  
   rgb_value=N1  
END IF  
RETURN  
END  

THIS ROUTINE FINDS THE MAX., AND MIN. OF A 100*100 ARRAY  
SUBROUTINE SEARCH(ARRAY,AMAX,AMIN)  
DIMENSION ARRAY(100,100)  
AMAX=-1000  
AMIN=1000  
DO 1 I=1,100  
   DO 1 J=1,100  
   END
IF(ARRAY(I,J).GT.AMAX)AMAX=ARRAY(I,J)
IF(ARRAY(I,J).LT.AMIN)AMIN=ARRAY(I,J)
CONTINUE
RETURN
END

THIS ROUTINE CALCULATES THE DIVISIONS ON THE AXES

SUBROUTINE MARKS(IMGSZ,PER,COORD)
  DIMENSION COORD(5)
  SIZE=PER*IMGSZ/4096.
  COORD(1)=SIZE/-2.
  COORD(2)=SIZE/-4.
  COORD(3)=0
  COORD(4)=SIZE/4.
  COORD(5)=SIZE/2.
RETURN
END
END

DATE

FILMED

6-88

DTIC