

AD-A191 847

CIMAG2: THE COMPUTER PROGRAM TO GENERATE COLOR IMAGES

1/2

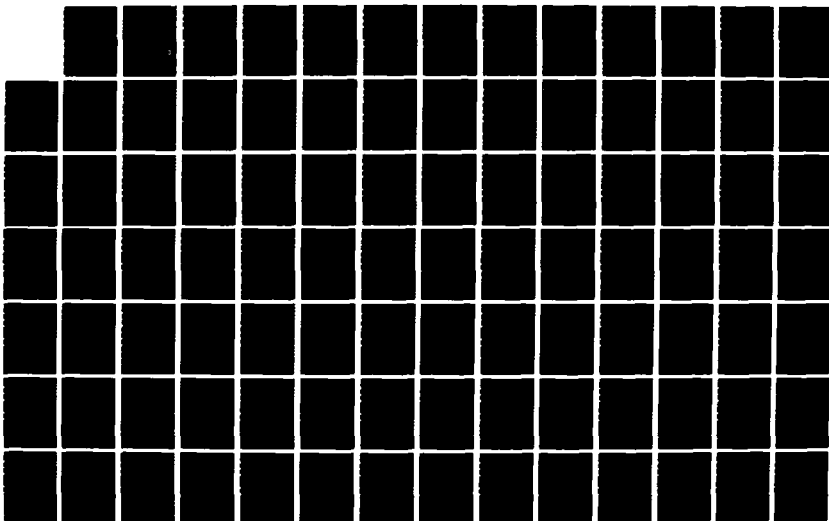
(U) OHIO STATE UNIV COLUMBUS ELECTROSCIENCE LAB

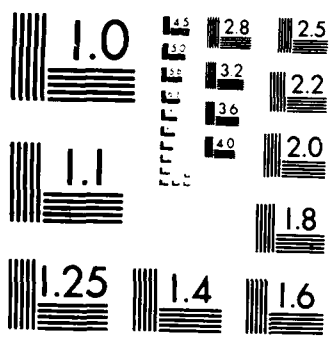
G DURAL ET AL NOV 87 ESL-718048-7 N00014-86-K-0202

UNCLASSIFIED

F/G 12/5

NL







AD-A191 847

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

E.L.
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

DTIC
ELECTE
MAR 07 1988
S *caH* D

DTIC
Approved for public release
Distribution is unlimited

88 3 01 151

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.

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2.	3. Recipient's Accession No.
4. Title and Subtitle CIMAG2: The Computer Program to Generate Color Images			5. Report Date November 1987
7. Author(s) G. Dural, S. Smithberger, J.D. Young			6.
9. Performing Organization Name and Address The Ohio State University ElectroScience Laboratory 1320 Kinnear Road Columbus, Ohio 43212			8. Performing Organization Rept. No. 718048-7
12. Sponsoring Organization Name and Address Department of the Navy Office of Naval Research 800 N. Quincy Street Arlington, VA 22217			10. Project/Task/Work Unit No.
15. Supplementary Notes			11. Contract(C) or Grant(G) No. (C) N00014-86-K-0202 (G)
16. Abstract (Limit: 200 words) 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.			13. Type of Report & Period Covered Technical
17. Document Analysis			14.
a. Descriptors			
b. Identifiers/Open-Ended Terms			
c. COSATI Field/Group			
18. Availability Statement Approved for public release; distribution is unlimited.		19. Security Class (This Report) Unclassified	21. No. of Pages 106
		20. Security Class (This Page) Unclassified	22. Price

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	Read Commands	6
	RDFL Frequency Domain Read	6
	REA Time Domain Read	8
	FTREA or FFREA.....	8
	FTY	8
2.1.2.2	IFF Inverse Fourier Transform	8
2.1.2.3	ROT Rotate	9
2.1.2.4	WND Windowing	9
2.1.2.5	DCV Downconversion	10
2.1.2.6	MJW Multiply by jw	10
2.1.2.7	MWR Multiply by w	11
2.1.2.8	WRI Write	11
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



19		ton For
20		GRA&I ☒
21		Unannounced ☐
		Justification ☐
By _____		
Distribution/ _____		
Availability Codes		
Avail and/or		
Dist	Special	
A-1		

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.

```

<>RDFL          ! A frequency domain read command
...            ! This command will be covered in
               ! the following sections.
<>SBF           ! The store in buffer command
  BUFFER # 1
<>RDFL
...
<>SBF
  BUFFER # 2

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

<>RB1           ! Put the contents of buffer 1 in
               ! the main buffer.
<>FGT           ! This is the gating command
  HARMONIC FOR START OF HIGH FREQUENCY CUTOFF ! The array element
    598                                           ! where the cutoff
                                           ! begins.
  HIGH FREQUENCY ROLL-OFF IN DB PER HARMONIC ! This is quite a
    50.0                                           ! bit since we
                                           ! want
                                           ! a sharp cutoff.
  HARMONIC FOR START OF LOW FREQUENCY CUTOFF ! This element is
    1                                           ! in the region
                                           ! that is never
                                           ! used close
                                           ! to DC.
  LOW FREQ ROLLOFF IN DB PER HARMONIC ! A very slow roll-off
    0.01

<>SB3           ! 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.

<>RB2
<>IFF           ! Inverse Fast Fourier transform
               ! does this conversion.

<>SB4
<>RB3
<>IFF
<>SB5

```

```

<>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
..
  <>RB6
  <>WRI
  ...
                                              ! This command is pretty well
                                              ! self explanatory.

```

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 VND 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 MJW 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 MWR Multiply by w

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. CIMAG2 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:

<>PROC

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 '*'.

```
<>RDFL
  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:

\$FOR/LIST CIMAG2

\$@CIMAG2

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:

```
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:

```
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
    'GRP11LIB',           -!Plotting
    'PLOTOLD2'            -!Plotting
$ !
$ EXIT

$ !
$ !   This command procedure links the modules for the NEWMAG program.
$ !
$ LINK\NOTRACE
    NEWMAG,
    INTER,                -!Frequency Domain Read
    RDFLE,                -!Frequency Domain Read (called by INTER)
    FORT,                 -!Fourier Transform
    DDMPB,                -!BSC Read
    CIMAGMSG,             -!Program Error Messages
    'GRP11LIB',           -!Plotting
    'PLOTOLD2'            -!Potting
$ !
$ EXIT
```

REFERENCES

- [1] Dominek, A., Personal Communication, The Ohio State University ElectroScience Laboratory, Columbus, Ohio.
- [2] Dural, G. and J.D. Young, "Polarimetric ISAR Imaging Using Either Measured or Calculated Transient Signatures," Technical Report 718048-6, The Ohio State University ElectroScience Laboratory, Department of Electrical Engineering, generated under Contract No. N00014-86-K-0202, for Department of the Navy, Office of Naval Research, Arlington, Virginia, October 1987.

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 \$SPAWN

If the user uses the \$SPAWN 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:

$BUF(0)=a1*BUF(n1)+a2*BUF(n2)+a3*BUF(n3)+\dots$

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 it's 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)

1 DIF

The differentiate command calculates the derivative of the TIME DOMAIN waveform in main buffer and places the result in main buffer.

1 DJW

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.

1 ECHO

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.

1 EXI

Exit Command

1 FFT

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.

1 FGT

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

2 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 F=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", $1 \leq a \leq 4096$)
START GATE CLOSE AT SAMPLE
 (reply with integer "b", $b > a, 1 \leq b \leq 4096$)
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 $a < n \leq (a + c)$
 $X'(n) = X(n)$ for $(a + c) \leq n \leq b$
 $X'(n) = X(n) * (n - b) / c$ for $b < n \leq (b + c)$
 $X'(n) = 0$ for $(b + c) \leq n \leq 4096$

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 CMI 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.

1 MANUALS

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)

e.g.
\$LASER USER.TXT

1 MJW

Multiplies the frequency spectrum in main buffer by $j \cdot 2 \cdot \pi \cdot f$ (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 $2 \cdot \pi \cdot f$ 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 FREQ ROLLOFF

(reply with a pair of integers,
"a", "b", $0 \leq a, b, \leq 4096$)

The spectrum is unchanged for $n \leq a$. It is -100 db for $n \geq b$. And for $a \leq n \leq b$, 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.

2 Parameters

INPUT HARMONICS;START,END,TYPE OF WINDOW

TYPE:0=HANNING,1=HAMMING,2=GAUSSIAN, 0,N,M=TEST
(reply with a pair of integers,
"a", "b", $0 \leq a, b, \leq 4096$ and then 0, 1, or 2
depending on the type of rolloff desired)

1 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.

2 Parameters

FILE NAME ?

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

1 YSH

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

2 Parameters

YSHIFT IN UNITS($-2048 < Y < 2048$)=

(reply with the shift value, CARRIAGE RETURN)

APPENDIX B THE COLOR IMAGING PROGRAM 'CIMAG2'

```

C      CIMAG2
C
C      CHARACTER*80 COMMAND                                ! the command
C
C      INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
C      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
C      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'           ! common blocks
C      INCLUDE 'USER2:[DURAL.CIMAG2]HEADER.CMN'
C
C      EQUIVALENCE (HEADER,HEAD(1,1)),(LINE1(1),HEADER(1))
C      EQUIVALENCE (FLTP,IDS(1)),(HEADER(61),LINE3(1)),(HEADER(31),LINE2(1))
C
C      INTEGER*4 FOR_RETCODE                                ! fortran return code
C      INTEGER*4 FOR_EOF / -1 /                             ! end of file code
C      INTEGER*4 RETCODE                                    ! return code for RTL routines
C      INTEGER*4 LIB$GET_LUN                                ! 'get LU#' RTL routine
C
C      CALL START                                           ! initialize all the variables
C
C      RETCODE = LIB$GET_LUN( TERM_UNIT )
C      OPEN( FILE='A:',                                     ! open terminal for I/O
+         UNIT=TERM_UNIT ,
+         STATUS='NEW' )
C
C      RETCODE = LIB$GET_LUN( NULL_UNIT )                   ! null unit for logging
C      OPEN( FILE='NL:',
+         UNIT=NULL_UNIT ,
+         STATUS='NEW' )
C
C      COM_UNIT=TERM_UNIT                                  ! get commands from terminal initially
C      IOUT=TERM_UNIT                                       ! send output to the terminal
C      LOG_UNIT=NULL_UNIT                                   ! log to null unit
C
C      DO WHILE (COMMAND.NE. 'EXI')                         ! do until exit command
C      IF (COM_UNIT.EQ. TERM_UNIT) THEN                     ! if waiting for a command
C from terminal      IOUT=TERM_UNIT                        ! then all prompts go to terminal
C                     WRITE( UNIT=IOUT, FMT=5 )
C      ELSE
C      IF (ECHO) THEN                                       ! if using a command file
C      IOUT=TERM_UNIT                                       ! output can either be
C      ELSE                                                ! displayed on terminal
C      IOUT=NULL_UNIT                                       ! or not be displayed at all
C      END IF
C      END IF
C      READ ( UNIT=COM_UNIT , FMT=10 ,                     ! read the command
+         IOSTAT=FOR_RETCODE ) COMMAND
C
C      IF (FOR_RETCODE.EQ. FOR_EOF) THEN                   ! on end of file get old
C control device    CALL BACK
C                     IF (PROC_FLAG) THEN                 ! if coming back from a procedure
C                     CALL PROC3                          ! finish off the PROC procedure
C                     END IF

```

```

ELSE
C
CALL STR$UPCASE( COMMAND , COMMAND ) ! convert to uppercase
C
C
C      IF ((COMMAND .NE. 'STO_LOG') .AND.      ! if command is
C not equal to      (COMMAND .NE. 'FILE') .AND.      ! stop log or
C      +      (COMMAND .NE. 'DONE')) THEN
C run command file      WRITE( UNIT=LOG_UNIT , FMT=10 ) ! log
C      +      COMMAND
C the input
C      +      END IF
C
C      IF (COMMAND .EQ. 'BSC') THEN      ! execute the command
CALL BSC
C
C      ELSE IF (COMMAND .EQ. 'CARD') THEN
CALL CARD
C
C      ELSE IF (COMMAND .EQ. 'CHANGE') THEN
CALL CHANGE
C
C      ELSE IF (COMMAND .EQ. 'CLR') THEN
CALL CLR
C
C      ELSE IF (COMMAND .EQ. 'CLR_BUF') THEN
CALL CLR_BUF(HEAD, LINE1, LINE2, LINE3)
C
C      ELSE IF (COMMAND .EQ. 'CMI') THEN
CALL CMI
C
C      ELSE IF (COMMAND .EQ. 'COM') THEN
CALL COM(HEAD, LINE1, LINE2, LINE3)
C
C      ELSE IF (COMMAND .EQ. 'CREATE') THEN
CALL CREATE(HEAD, LINE1, LINE2, LINE3)
C
C      ELSE IF (COMMAND .EQ. 'DATA') THEN
CALL LIB$SPAWN( 'RUN USER2:'
+      //'[DURAL]DBMANAGER') ! use the database
C
C      ELSE IF (COMMAND .EQ. 'DCV') THEN
CALL DCV
C
C      ELSE IF (COMMAND .EQ. 'DEF') THEN
CALL DEFFER      ! set def command
C
C      ELSE IF (COMMAND .EQ. 'DEL') THEN
CALL DEL
C
C      ELSE IF (COMMAND .EQ. 'DIF') THEN
CALL DIF
C
C      ELSE IF (COMMAND .EQ. 'DJW') THEN
CALL DJW
C
C      ELSE IF (COMMAND .EQ. 'DONE') THEN
CALL PROC2
C
C      ELSE IF (COMMAND .EQ. 'DWR') THEN
CALL DWR
C
C      ELSE IF (COMMAND .EQ. 'ECHO') THEN      ! shows command
C file instructions as they are      ECHO=.TRUE.      ! being executed.
C
C      ELSE IF (COMMAND .EQ. 'FFT') THEN
CALL FFT
C
C      ELSE IF (COMMAND .EQ. 'FGT') THEN
CALL FGT
C
C      ELSE IF (COMMAND .EQ. 'FILE') THEN
CALL FILE
C
C      ELSE IF (COMMAND .EQ. 'FRD') THEN
CALL FRDC
C
C      ELSE IF (COMMAND .EQ. 'FTREA') THEN
CALL FTREA
C
C      ELSE IF (COMMAND .EQ. 'FFREA') THEN
CALL FFREA
C
C      ELSE IF (COMMAND .EQ. 'PTY') THEN
CALL PTY
C
C      ELSE IF (COMMAND .EQ. 'GAT') THEN

```

+
C help files

```

CALL GAT
ELSE IF (COMMAND.EQ. 'GRID') THEN
CALL GRID(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND.EQ. 'HELP') THEN
CALL LIB$SPAWN('HELP/PAGE/LIBRARY=USER2:'
//'[DURAL.CIMAG2]CIMAG2.HLB') ! call

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. 'RDFL') THEN
CALL RDFL(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(1:2).EQ. 'RB') THEN
NB = ICHAR( COMMAND(3:3) ) - 48
CALL RB(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND.EQ. 'RLB') THEN
CALL RLB
ELSE IF (COMMAND.EQ. 'ROT') THEN
CALL ROT
ELSE IF (COMMAND.EQ. 'SBF') THEN
CALL SBF(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND(1:2).EQ. 'SB') THEN
NB = ICHAR( COMMAND(3:3) ) - 48
CALL SB(HEAD, LINE1, LINE2, LINE3)
ELSE IF (COMMAND.EQ. 'SHO_BUF') THEN
CALL SHO_BUF
ELSE IF (COMMAND.EQ. 'STO_LOG') THEN
CALL STO_LOG
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. ' 1') THEN
CONTINUE ! clause on the old REA command
ELSE IF (COMMAND(1:1).EQ. '$') THEN ! If a

CALL LIB$SPAWN( COMMAND ) ! execute the DCL

```

C DCL command


```

P(1,1)=(P(1,1)/10.24)-10.
IS=ISIZE/2.
P(IS+1,1)=PI*P(IS+1,1)/512.
A(1)=CMPLX(COS(P(IS+1,1)),SIN(P(IS+1,1)))*10.** (P(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.
30 A(N)=CMPLX(COS(P(IS+N,1)),SIN(P(IS+N,1)))*RRT2*10.** (P(N,1)*.05)
A(IS+1)=CMPLX(0.,0.)
DO 31 N=2,IS
NN=ISIZE+2-N
31 A(NN)=CMPLX(REAL(A(N)),-AIMAG(A(N)))
CALL FORT(A,IZ,S,2,IFERR)
IF(IFERR.NE.0) THEN                                ! Print an error message
CALL LIB$SIGNAL(MAG_COM)
RETURN
END IF
DO 32 N=1,ISIZE
NN=ISIZE+1-N
32 P(NN,1)=REAL(A(NN))
RETURN
C
ENTRY FTY
WRITE(IOUT,*)'Enter file type for FTRAN read routines'
WRITE(IOUT,*)' T=750 F=11/23'
READ(COM UNIT,*)FTYPE
WRITE(LOG UNIT,*)FTYPE
RETURN
C
C FTRAN 'REA' COMMAND (FREQUENCY DOMAIN)
C
ENTRY FTREA
FLTP=1
IF(.NOT.FTYPE) CALL FREA(INPFILE,AMPL,PHS)
IF(FTYPE) CALL REU(INPFILE,AMPL,PHS)
RETURN
C
C FTRAN 'REA' COMMAND (TIME DOMAIN)
C
ENTRY FFREA
FLTP=0
IF(.NOT.FTYPE) CALL FREA(INPFILE,AMPL,PHS)
IF(FTYPE) CALL REU(INPFILE,AMPL,PHS)
RETURN
C
C
C
C
C DIFFERENTIATE ROUTINE
C MULT BY JW IN THE FREQ DOMAIN
C ENTRY MJW
C SN=1.
C GOTO 40
C
C INTEGRATE ROUTINE
C DIVIDE BY JW IN THE FREQ DOMAIN
C ENTRY DJW
C SN=-1.
C
C
C 40 IS=ISIZE/2.
IF (FLTP.NE.1.) THEN
CALL LIB$SIGNAL(MAG_COM)
END IF

```



```

53      P(N,1)=P(N,1)*F
      RETURN
C
C
C
C
C      TYPE COMMAND
      TYPES A BUFFER WAVEFORM SEGMENT
      ENTRY TYP(HEAD, LINE1, LINE2, LINE3)
      WRITE(IOUT,81)
81      FORMAT(' START AT ELEMENT NUMBER:',$)
      READ (COM UNIT,86) NP
      WRITE (LOG_UNIT,86) NP
      WRITE(IOUT,82)
      ! log the input
82      FORMAT(' END AT ELEMENT NUMBER:',$)
      READ (COM UNIT,86) NE
      WRITE (LOG_UNIT,86) NE
      ! log the input
      IF(NE.GE.ISIZE)NE=ISIZE-1
      WRITE(IOUT,83)
83      FORMAT(' BUFFER NUMBER(0=MAIN):',$)
      READ (COM UNIT,86) NNB
      WRITE (LOG_UNIT,86) NNB
      ! log the input
      NNB=NNB+1
      IF(NNB.LT.1) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
          ! Print an error messag
      END IF
      IF(NNB.GT.NBUF) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
          ! Print an error messag
      END IF
      WRITE(IOUT,117) (HEAD(I,NNB),I=1,30)
      WRITE(IOUT,117) (HEAD(I,NNB),I=31,60)
      WRITE(IOUT,117) (HEAD(I,NNB),I=61,90)
      WRITE(IOUT,85)(N,P(N,NNB),P(N+1,NNB),N=NP,NE,2)
85      FORMAT(I5,2F10.2)
86      FORMAT (I5)
      RETURN
C
C
C
C      BUFFER STORE ROUTINES
      STORE A WAVE OR SPECTRUM IN ONE OF 30 TEMP LOCS
      ENTRY SBF(HEAD, LINE1, LINE2, LINE3)
      WRITE (IOUT,87)
87      FORMAT(1X,' BUFFER #',$)
      READ (COM UNIT,93) NB
      WRITE (LOG_UNIT,93) NB
      ! log the input
C
      ENTRY SB(HEAD, LINE1, LINE2, LINE3)
      ! SB# routine
      IF (NB .GT. 35) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      END IF
      DOM(NB)=DOMAIN
      DO 90 I= 1, 30
90      BUFFERS(NB,I) = LINE1(I)
91      CALL BUFSTR(IHDSZ,IDSSZ,ISIZE,NB,HEAD,P,IDS)
93      FORMAT (I5)
      RETURN
C
C
C
C      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)

```



```

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,*) ANS                                    ! log the input
IF (ANS.NE.0) WIN4=ANS
WRITE(IOUT,*) 'MIN PLOT VALUE (-1024 NORMAL,0=NO CHANGE):'
READ (COM UNIT,*) ANS
WRITE (LOG UNIT,*) ANS                                    ! log the input
IF(ANS.NE.0) WIN3=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,*) ANS                                    ! log the input
IF (ANS.EQ.0) GO TO 18600
WRITE(IOUT,*) 'INPUT Y-AXIS MAX,MIN,AND TIK MARK INTERVAL'
READ (COM UNIT,*) YMAX,YMIN,YTMI
WRITE (LOG UNIT,*) YMAX,YMIN,YTMI                        ! log the input
WRITE(IOUT,*) 'Y AXIS LEGEND'
READ (COM UNIT,18601) YLAB
WRITE (LOG UNIT,18601) YLAB                                ! log the input
WRITE(IOUT,*) 'INPUT X-AXIS MAX,MIN,AND TIK MARK INTERVAL'
READ (COM UNIT,*) XMAX,XMIN,XTMI
WRITE (LOG UNIT,*) XMAX,XMIN,XTMI                        ! log the input
WRITE(IOUT,*) 'X AXIS LEGEND'
READ (COM UNIT,18601) XLAB
WRITE (LOG UNIT,18601) XLAB                                ! log the input
18600 WRITE(IOUT,*) 'INPUT TITLE FOR PLOT'
READ (COM UNIT,18601) TITLE
WRITE (LOG UNIT,18601) TITLE                              ! log the input
18601 FORMAT (A60)
4247 FORMAT(' STRT',F8.2,' END ',F8.2,' MIN Y',F8.2,' MAX Y',F8.2)
YINC=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.
CALL PLOT(0.5,5.5,-3)
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)
XDS=(XMAX-XMIN)/PSIZE
XSP=PSIZE*XTMI/(XMAX-XMIN)
CALL FFAXIS(0.,0.,%REF(XLAB),-60,PSIZE,0.,XMIN,XDS,XSP,1,1.,1)
YDS=(YMAX-YMIN)/PSIZE
YSP=PSIZE*YTMI/(YMAX-YMIN)
CALL FFAXIS(0.,0.,%REF(YLAB),60,PSIZE,90.,YMIN,YDS,YSP,1,1.,1)
CALL TEXT(0.5,PSIZE+0.5,0.1,TITLE,0.)
CALL PLOT(0,0,999)
CALL PLOTNOW(IMG)
IF(IMG.EQ.0) THEN
CALL LIB$SIGNAL(MAG_COM)                                ! Print an error message
RETURN
END IF
186 FORMAT (1X,'WAVEFORM BUFFER NO.?(0=MAIN)')
187 FORMAT (I5)
RETURN

```

C
C
C
C


```

C
C
C      TIME DOMAIN DIFFERENTIATE ROUTINE
C
      ENTRY DIF
      IF (IDS(1).NE.0) THEN
          CALL LIB$SIGNAL(MAG_COM)          ! Print an error message
          RETURN
      END IF
      PE=P(ISIZE,1)
      DO 220 N=2,ISIZE
          NN=ISIZE+2-N
220    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

C
C
C      STATEMENT 550 32583

C
C      RELABLE COMMAND

      ENTRY RLB
      WRITE (IOUT,225)
      WRITE (IOUT,231) ITITLE                ! sho the present label
225    FORMAT(/,' OLD TITLE BLOCK: ')
      WRITE (IOUT,230)                      ! prompt
230    FORMAT(/,' NEW TITLE BLOCK: ')
      READ (COM UNIT,231) ITITLE            ! read the new label
      WRITE (LOG UNIT,231) ITITLE           ! log the input
231    FORMAT(50A1)
      RETURN

C
C
C      NORMALIZING TIME DOMAIN DATA TO ZERO MEAN
C
      ENTRY NOR
      RMEAN=0.
      DO 3111 I=1,ISIZE
3111    RMEAN=P(I,1)+RMEAN
          RMEAN=RMEAN/ISIZE
      DO 3112 I=1,ISIZE
3112    P(I,1)=P(I,1)-RMEAN
      RETURN

C
C
C      CLEAR THE MAIN BUFFER

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

C
C
C      POINT SMOOTH ROUTINE

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

```

```

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= ', $)
4003  FORMAT (I5)
      READ (COM UNIT, 4003) IR
      WRITE (LOG UNIT, 4003) IR
! log the input
      IF (IQ.LT.3) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      END IF
      IF (IR.LT.3) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      END IF
      IF (IQ.GT.ISIZE-2) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      END IF
      IF (IR.GT.ISIZE-2) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      END IF
      IF (IR.LT.IQ) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      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)
      PSC=3.*PSD1/(PSN*PSN)-(2.*PSS1+PSS2)/PSN
      PSD=(PSS2+PSS1)/(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*I+PSD*I*I*I
      GO TO 4075
4051  WRITE (IOUT, 4052)
4052  FORMAT(' FIRST BAD HARMONIC= ', $)
      READ (COM UNIT, 4003) IQ
      WRITE (LOG UNIT, 4003) IQ
! log the input
      WRITE (IOUT, 4053)
4053  FORMAT(' LAST BAD HARMONIC= ', $)
      READ (COM UNIT, 4003) IR
      WRITE (LOG UNIT, 4003) IR
! log the input
      IF (IO.LT.3) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
! Print an error message
      END IF
      IF (IR.LT.3) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
! Print an error message
      END IF
      IF (IQ.GT.ISM2) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
      END IF
      IF (IR.GT.ISM2) THEN
          CALL LIB$SIGNAL(MAG_COM)
          RETURN
! Print an error message
      END IF
      IF (IR.LT.IQ) THEN

```



```

      READ (COM UNIT,*) NUMB(1)
      WRITE (LOG UNIT,*) NUMB(1)
      DO 5510 I=1,NUMB(1)
      WRITE (IOUT,5503) I
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
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
5507  FORMAT(' CENTER ELEMENT NUMBER FOR FILE # ',I5)
      READ (COM UNIT,*) CNTR(1,I)
5510  WRITE (LOG UNIT,*) CNTR(1,I)
      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
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)
5520  WRITE (LOG UNIT,*) CNTR(2,I)
      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
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)
5530  WRITE (LOG UNIT,*) CNTR(3,I)
      RETURN

```

C
C
C
C
C
C
C

ROUTINE 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)=?'
      READ (COM UNIT,*) IMSIZE
      WRITE (LOG UNIT,*) IMSIZE
      FIMINC=FLOAT(IMSIZE/100)
      PNUM=FIMINC
      IF (MOD(PNUM,2) .EQ. 0) THEN

```

```

        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                                       ! two numbers are 0
    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,*) IIPOL
    WRITE (LOG UNIT,*) IIPOL                        ! log the input
    IPOL=IIPOL
    IF(IIPOL.NE.4) GO TO 5660                       ! if not using all polarizations then
    DO 5699 I=1,3                                    ! just do the individual pol.
5660    DO 5690 I=1,100                               ! figure out the X coordinate
        X=(I-50.5)*FIMINC
        DO 5680 J=1,100                               ! figure out the Y coordinate
            Y=(J-50.5)*FIMINC
            XX=X*COS(BLANGL)+Y*SIN(BLANGL)            ! the
C   X coordinate taking the rotation angle into account
            YY=-X*SIN(BLANGL)+Y*COS(BLANGL)           ! the Y
C   coordinate taking the rotation angle into account
            ARRAY(IIPOL,I,J)=0.                      ! clear the image array
            DO 5670 K=1,NUMB(IIPOL)
                XXX=XX*COS(ANG(IIPOL,K))+YY*SIN(ANG(IIPOL,K)) ! the X coordinate
                YYY=-XX*SIN(ANG(IIPOL,K))+YY*COS(ANG(IIPOL,K)) ! the Y coordinate
                IPOINT=YYY+CNTR(IIPOL,K)              ! the point of intersection
                IF(IPOINT.LT.0) IPOINT=IPOINT+ISIZE    ! the time domain waveform
C   repeats throughout time
                IF(IPOINT.GT.ISIZE) IPOINT=IPOINT-ISIZE
                VALUE=0
                DO ELEMENT=((PNUM-1)/2), (IPOINT+((PNUM-1)/2)) ! take
C   the points around the center point
                    ELEM=ELEMENT
                    WEIGHT=1+COS((2*(ELEM-IPOINT)*PI)/(PNUM-1)) ! use a weighted
C   shifted cosine average
                    IF(ELEM.LT.0) ELEM=ELEM+ISIZE    ! the time domain waveform
C   repeats throughout time
                    IF(ELEM.GT.ISIZE) ELEM=ELEM-ISIZE
                    PVALUE=WEIGHT*P(ELEM,BUFN(IIPOL,K))
                    VALUE=VALUE+PVALUE
            END DO
            VALUE=VALUE/(PNUM-1)
            ARRAY(IIPOL,I,J)=ARRAY(IIPOL,I,J)+VALUE/NUMB(IIPOL)
5670    CONTINUE
5680    CONTINUE
5690    CONTINUE
5699    CONTINUE
C   WRITE THE IMAGE ARRAY TO AN OUTPUT FILE
    WRITE(IOUT,*) 'Do you want to store the image? Y=1,N=0 '
    READ (COM UNIT,*) ANS
    WRITE(LOG UNIT,*) 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(A40)
C
    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
C

```



```

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
3350 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
3380 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
3390 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
7740 IF(K.GE.FA(I).AND.K.LE.FA(I+1))GO TO 7730
I=I+1
GO TO 7740
7730 IF(K.GT.1024)GO TO 7777
KLA=(CA(I+1)-CA(I))/(FA(I+1)-FA(I))*(K-FA(I))+CA(I)
KLA=KLA*2./(SQRT(TWOPI*.2.)*DLE)
P(K,1)=10.24*(10.+20.*ALOG10(CABS(KLA)))
7720 P(K+1024,1)=1024.*ATAN2(AIMAG(KLA),REAL(KLA))/TWOPI
7777 RETURN
END
C
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
HEAD(I,IC)=HEAD(I,1)
DO 20 I=1,ISIZE
20 P(I,IC)=P(I,1)

```

```

IDS(IC)=IDS(1)
RETURN
END

C
C
C
SUBROUTINE BUFDR(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
HEAD(I,1)=HEAD(I,IC)
DO 20 I=1,ISIZE
P(I,1)=P(I,IC)
IDS(1)=IDS(IC)
RETURN
END

C
C
C
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$LIBRARY:FORIOSDEF'
810 OPEN(UNIT=19,NAME=INFILE,STATUS='OLD',
+      DEFAULTFILE=DEF,IOSTAT=IERR,ERR=8100)
DO 20 I=1,1024,256
J=I+255
20 READ(19,30)(BUFF(K),K=I,J)
30 FORMAT(256A2)
DO 70 I=1,201
AM(I)=10*(AM(I)/20.)
70 CA(I)=CMPLX(AM(I)*COS(PH(I)),AM(I)*SIN(PH(I)))
GO TO 80
8100 IF(IERR.EQ.FOR$IOS FILNOTFOU)THEN
WRITE(TERM UNIT,1112) INFILE
1112 FORMAT(' FILE : ',A20,' DOES NOT EXIST',/,/, ' ENTER FILENAME AGAIN')
ELSE IF (IERR.EQ.FOR$IOS FILNAMSPE)THEN
WRITE(TERM UNIT,*) 'FILE:',INFILE,' 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

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

SUBROUTINE CONTROLLER

C
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]MSGBLK.FOR' ! error message common block

```



```

      OPEN(  UNIT=FILE_UNIT ,           ! open the command file
      +     FILE=NAME ,
      +     DEFAULTFILE=DEF ,
      +     IOSTAT=FOR RETCODE ,
      +     STATUS='OLD')
C
      IF (FOR_RETCODE .EQ. 29) THEN      ! if file not found
      CALL LIB$SIGNAL(MAG_FILNOTFOU)    ! print error
      RETURN
      ELSE IF (FOR_RETCODE .NE. 0) THEN  ! if not normal
      CALL LIB$SIGNAL(MAG_COM)          ! Print an error messag
      RETURN
      END IF
C
      TOP_COM = TOP_COM + 1              ! push the old command unit onto the stack
      COM_STACK( TOP_COM ) = COM_UNIT
C
      TOP_DEF = TOP_DEF + 1              ! push old def on stack
      DEF_STACK( TOP_DEF ) = DEF
C
      COM_UNIT = FILE_UNIT              ! give control to the file
C
C
C
C
10  FORMAT(1X, 'COMMAND FILE?', $)      ! the name prompt
15  FORMAT( A )
20  FORMAT(1X, 'ERROR IN RETRIEVING LU#')
C
      RETURN
C
C
C
C
C
      BACK:
          This command gives control back to
          the unit that had control before this
          one took over.
C
      ENTRY BACK
C
      IF (COM_UNIT .EQ. TERM_UNIT) THEN  ! if terminal is command input
      RETURN                             ! then ignore this command
      END IF
C
      FILE_UNIT = COM_UNIT               ! retrieve file LU#
      CLOSE( UNIT = FILE_UNIT )          ! close the file
C
      COM_UNIT = COM_STACK( TOP_COM )    ! pop the last com_unit off the stack
      TOP_COM = TOP_COM - 1
C
      DEF = DEF_STACK( TOP_DEF )         ! pop old def
      TOP_DEF = TOP_DEF - 1
C
      RETCODE = LIB$FREE LUN( FILE_UNIT ) ! give LU# back to system
      IF (RETCODE .NE. SSS$ NORMAL) THEN ! if error
      CALL LIB$STOP( %VAL( RETCODE ) ) ! then stop and give reason
      END IF
C
      IF (COM_UNIT .EQ. TERM_UNIT) THEN  ! if the terminal has become
      WRITE(IOUT,25)                     ! the command unit
      END IF                             ! then tell the user
25  FORMAT( 1X, 'CONTROL HAS RETURNED TO THE TERMINAL' )
      RETURN
C
C

```

```

C      DEF:
C          This command sets the default for all the
C          file I/O.
C
C      ENTRY DEFFER
C      WRITE(IOUT,30)
C          ! ask user for the default
C      READ(COM_UNIT,35) DEF
C      WRITE (LOG_UNIT,35) DEF
C          ! log the input
C      IF (PROC_FLAG) THEN
C          ! if defining a procedure
C          DEF='USER2:[DURAL.CIMAG2]'
C          ! then all files accessed are
C      in this directory
C      END IF
30      FORMAT(1X, 'DEFAULT?', $)
35      FORMAT(A40)
      RETURN
C
C      LOG:
C          This command copies all the user input
C          into a command file.
C
C      ENTRY LOGGER
C
C      RETCODE=LIB$GET_LUN( LOG_UNIT )
C          ! get the LU# for the log file
C      IF (PROC_FLAG) THEN
C          ! if defining a procedure
C          NAME='USER2:[DURAL.CIMAG2]PROC.DAT' !this is the definition file
C      ELSE
C          WRITE(IOUT,40)
C          ! prompt for file name
C          READ(COM_UNIT,45) NAME
C          ! get the file name
C      END IF
C      IF (RETCODE .EQ. SS$ NORMAL) THEN
C          ! if no errors occurred
C          OPEN(
C              UNIT=LOG_UNIT ,
C              FILE=NAME ,
C              STATUS='UNKNOWN')
C          +
C          +
C      ELSE
C          WRITE(IOUT,50)
C          CALL LIB$STOP(%VAL(RETCODE))
C      END IF
C
C      40      FORMAT(1X, 'LOG FILE?', $)
C          ! the name prompt
C      45      FORMAT( A )
C      50      FORMAT(1X,'ERROR IN RETRIEVING LU#')
C
C      RETURN
C
C      STO_LOG:
C          Stop logging.
C
C      ENTRY STO_LOG
C
C      CLOSE( UNIT=LOG_UNIT )
C          ! close the logging file
C      RETCODE = LIB$FREE_LUN( LOG_UNIT )
C          ! give LU# back to system
C      IF (RETCODE .NE. SS$ NORMAL) THEN
C          ! if error
C          CALL LIB$STOP( %VAL( RETCODE ) ) ! then stop and give reason
C      END IF
C      LOG_UNIT = NULL_UNIT
C          ! log to null device
C      RETURN
C
C      END

```

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' ! controls common block

REAL SEC ! number of nanoseconds per division
 INTEGER*4 INC ! increment
 INTEGER*4 GWID ! width of a grid line
 INTEGER*4 PROC_UNIT ! LU# for definition file
 INTEGER*4 LIST_UNIT ! LU# for list file
 INTEGER*4 NEW_UNIT ! LU# for list of output file names
 INTEGER*4 PROC2_UNIT ! The command procedure file

INTEGER*4 FOR_RETCODE ! fortran return code
 INTEGER*4 PROCSTAT ! status of a read from def. file
 INTEGER*4 LISTSTAT ! status of a read from list file
 INTEGER*4 NEWSTAT ! status of a read from output list file
 INTEGER*4 FOR_EOF / -1 / ! end of file code

CHARACTER*80 OPER ! input data from the definition file
 CHARACTER*1 REPLY ! answer to Y or N question

CHARACTER*70 NAME ! name of the logging file
 CHARACTER*70 NEWNAME ! new file name
 CHARACTER*75 COMMAND ! command sent to a spawned procedure

CHARACTER*70 PROCNAME ! definition file name
 CHARACTER*70 LISTNAME ! data list file
 CHARACTER*70 OUTNAME ! output list name

CLR_BUF:

This routine clears all the buffers

ENTRY CLR_BUF(HEAD, LINE1, LINE2, LINE3)
 CALL CLR ! clear the main buffer
 DO NB=1,40 ! do for all the buffers
 IF (BUFFERS(NB,1) .NE. 0) THEN ! if buffer is not empty
 CALL SB(HEAD, LINE1, LINE2, LINE3) ! 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(10UT,10) ! prompt for the number of nanoseconds between lines
 READ(COM_UNIT,20) SEC ! The number of nanoseconds between lines
 CALL CREATE(HEAD, LINE1, LINE2, LINE3) ! Create a new time domain


```

READ(COM_UNIT,56) REPLY                                ! read in answer
IF(REPLY_.EQ. 'Y') THEN                                ! if he wants to save the def
    WRITE(IOUT,57)                                     ! prompt for a file name
    READ(COM_UNIT,56) NAME                             ! read the filename
    COMMAND(I:33)='$COPY USER2:[DURAL.CIMAG2]PROC.DAT ' ! the first
C half of the command
    COMMAND(34:75)=NAME ! the filename that the copy is output to
    CALL LIB$SPAWN( COMMAND ) ! spawn a process to copy the file
END IF
55 FORMAT(' Do you wish to save this procedure definition?(Y or N)')
56 FORMAT( A )
57 FORMAT(' Filename:')
C
58 WRITE(IOUT,59)                                     ! Ask if there is already a data list
READ(COM_UNIT,75) REPLY                                ! get answer
IF(REPLY_.EQ. 'Y') THEN
    WRITE(IOUT,60)
    READ(COM_UNIT,75) LISTNAME                         ! data list filename
    CALL OPENER(LIST_UNIT,LISTNAME)                   ! get a LU#
    CLOSE(LIST_UNIT)                                  ! close the unit back up
ELSE
    LISTNAME='USER2:[DURAL.CIMAG2]LIST.DAT'
    CALL OPENER(LIST_UNIT,LISTNAME)                   ! open a file
    WRITE(IOUT,61)                                     ! prompt
    WRITE(IOUT,65)
    WRITE(IOUT,70)
    READ(COM_UNIT,75) NAME                             ! read first name in list
    DO WHILE(NAME .NE. 'DONE')                       ! when finished use
C the word DONE
        WRITE(LIST_UNIT,75) NAME ! write name into list file
        READ(COM_UNIT,75) NAME ! read next name
    END DO
    CLOSE(LIST_UNIT)                                  ! close the list file
    WRITE(IOUT,71)                                     ! ask the user if it should be saved
    READ(COM_UNIT,75) REPLY
    IF(REPLY_.EQ. 'Y') THEN
        WRITE(IOUT,60)                                ! ask for file name
        COMMAND(1:34)='$COPY USER2:[DURAL.CIMAG2]LIST.DAT '
C the first half of the command
        COMMAND(35:75)=NAME ! the filename that the copy
C is output to
        CALL LIB$SPAWN( COMMAND ) ! spawn a process to
C copy the file
    END IF
END IF
59 FORMAT(' Do you have a data list file?(Y or N)')
60 FORMAT(' Filename:')
61 FORMAT(' Enter the list of data files, following')
65 FORMAT(' each with <CR>. When finished type the')
70 FORMAT(' word DONE.')
71 FORMAT(' Do you wish to save this data list?(Y or N)')
75 FORMAT( A )
C
C
WRITE(IOUT,76)                                     ! is there an output name file
READ(COM_UNIT,75) REPLY                                ! read answer
IF (REPLY_.EQ. 'Y') THEN
    WRITE(IOUT,77)                                     ! prompt for a filename
    READ(COM_UNIT,75) OUTNAME
    CALL OPENER(NEW_UNIT,OUTNAME)                     ! get a LU#
    CLOSE(NEW_UNIT)
ELSE
    OUTNAME='USER2:[CIMAG2]NEWLIST.DAT' ! list of new file names
    CALL OPENER(NEW_UNIT,OUTNAME) ! open newlist file
    WRITE(IOUT,80) ! prompt
    WRITE(IOUT,85)

```

```

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) ! save?
READ(COM_UNIT,75) REPLY
IF (REPLY .EQ. 'Y') THEN
    WRITE(IOUT,77)
    READ(COM_UNIT,75) NAME
    COMMAND(1:37)='$COPY USER2:[DURAL.CIMAG2]NEWLIST.DAT '
C the first half of the command
    COMMAND(38:75)=NAME ! the filename
C that the copy is output to
    CALL LIB$SPAWN( COMMAND ) ! spawn a process
C to copy the file
    END IF
END IF
76 FORMAT(' Is there an output filename list?(Y or N)')
77 FORMAT(' Filename:')
78 FORMAT(' Do you wish to save this list?(Y or N)')
80 FORMAT(' Enter a list of the output file names in') ! The prompt
85 FORMAT(' the order they are to be used. When')
90 FORMAT(' finished type DONE.')
95 FORMAT( A )
C
C
WRITE(IOUT,97)
97 FORMAT(' Your data is being processed.')
C
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)
C
C
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
C to be processed
    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 ! if the data
C file name is needed then get the name
            IF(LISTSTAT .EQ. FOR_EOF) THEN ! if there is
C no filename print error
                CALL LIB$SIGNAL(MAG_COM)
            END IF
            OPER=NAME
            READ( LIST_UNIT, 100, IOSTAT=LISTSTAT ) NAME
C! read next data entry
        ELSE IF( OPER .EQ. 'NEWNAME.DAT' ) THEN ! if output
C file is needed
            READ( NEW_UNIT, 100 , IOSTAT=NEWSTAT ) NEWNAME
C! read output file name
            IF(NEWSTAT .EQ. FOR_EOF) THEN ! if error
C print message
                CALL LIB$SIGNAL(MAG_COM)

```



```

C      INTEGER*4 L_UNIT                                ! logical unit #
C      CHARACTER*70 NAME                                ! file name
C      PARAMETER (SS$ _NORMAL = '00000001'X)          ! normal return code
C
C      RETCODE=LIB$GET_LUN( L_UNIT )                    ! get the LU# for the list file
C      IF (RETCODE .EQ. SS$ _NORMAL) THEN               ! if no errors occurred
C          OPEN( UNIT=L_UNIT ,                          ! open the list file
C              FILE=NAME ,
C              STATUS='UNKNOWN' )
C      +
C      ELSE
C          WRITE(IOUT,55)                                ! error conditions
C          CALL LIB$STOP(%VAL(RETCODE))
C      END IF
C
C 55  FORMAT(1X,'ERROR IN RETRIEVING LU#')
C      RETURN
C      END
C
C      SUBROUTINE OUTSIDER:
C          These are routines are more basic commands
C          such as those found in COM_FILE.
C
C      SUBROUTINE OUTSIDER
C
C          INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
C          INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
C          INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
C          INCLUDE 'USER2:[DURAL.CIMAG2]HEADER.CMN'
C
C      CREATE:
C          Creates a blank time domain file with a
C          header and stores it in the main buffer.
C
C      ENTRY CREATE(HEAD, LINE1, LINE2, LINE3)
C
C      WRITE(IOUT,*) 'Type in the header:(three lines)' ! prompt
C      READ(COM UNIT,10) LINE1                             ! read in
C      the new header
C      READ(COM UNIT,10) LINE2
C      READ(COM UNIT,10) LINE3
C 10  FORMAT(60A1)
C
C      DO I=1,ISIZE
C          P(I,1)=0                                         ! The original array is '0's
C      END DO
C      IDS(1)=0
C      DOMAIN='TIME'                                       ! This file is in the time domain
C      RETURN
C
C      CHANGE:
C          This routine changes a given harmonic in a data

```

```

C      file.
C
C
C      ENTRY CHANGE
C      WRITE(IOUT,30)                ! Prompt for the harmonic to be changed
C      READ(COM UNIT,*) N             ! Read in the value for the harmonic
C      WRITE(IOUT,40) (N,P(N,1))     ! Give the current value of the harmonic
C      WRITE(IOUT,50)                ! Prompt for the new value
C      READ(COM UNIT,60) TEMP         ! The new value
C      ENTRY GRD
C      P(N,1)=TEMP                   ! Put the new value in the specified harmonic
30    FORMAT(X,'Which harmonic do you wish to change?')
40    FORMAT(' Current Value: ',I5,F10.2)
50    FORMAT(' New Value: ')
60    FORMAT(F10.2)
      RETURN
      END
C
C
C*****
C*****
C      FTRAN READ COMMANDS (BY A. DOMINEK)
C*****
C      THE CALLING PROGRAM MUST CONTAIN THE FOLLOWING
C      COMMON BUFFER,NDIM,ANST,AINC
C      DIMENSION AMPL(5000),PHS(5000)
C      BYTE BUFFER(35000)
C      INTEGER*2 INPFILE(15)
C      USER SUPPLIES THE FILE NAME IN VARIABLE 'INPFILE'
C      IN FREQUENCY DOMAIN
C          AMPL  CONTAINS AMPLITUDE OF DATA IN DB
C          PHS   CONTAINS PHASE OF DATA IN DEGREES
C          NDIM  IS THE NUMBER OF FREQUENCY SAMPLES
C          ANST  IS THE FREQUENCY (MHZ) OF THE FIRST SAMPLE
C          AINC  IS THE DELTA FREQUENCY (MHZ)
C      IN TIME DOMAIN
C          AMPL  CONTAINS THE AMPLITUDE OF TIME WAVEFORM
C          PHS   CONTAINS ZERO
C          NDIM  IS THE NUMBER OF TIME SAMPLES = 4096
C          ANST  IS THE STARTING TIME =1/DF/2 (DF=DELTA FREQUENCY)
C          AINC  IS THE DELTA TIME * 1.E5 =1/DF/4096*1.E5
C
C*****
C*****
C      SUBROUTINE FREA(INPFILE,AMPL,PHS)
C
C      PROGRAM NAME :USER1:[DOMI]REAV.FOR
C      THIS PROGRAM READS BACKSCATTER DATA FILES STORED
C      ON VAX DISKS. WITH 11/23 FORMAT
C
C      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
C      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
C      INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
C
C      INTEGER*2 LIN1(30),LIN2(30),PARAM(30)
C      REAL*4 AP(10000)
C
C      DEFINE BUFFER STRUCTURE
C
C      EQUIVALENCE(LIN1(1),BUFFER(1)),(LIN2(1),BUFFER(61))
C      1,(PARAM(1),BUFFER(121)),(AP(1),BUFFER(361))
C      EQUIVALENCE(LINE1(1),LIN1(1)),(LINE2(1),LIN2(1))
C      1,(LINE3(1),PARAM(1))

```

```

C
C
C      READ A FILE
C
C      CALL TTR(INPFILE)
C      TYPE 105,LIN1
C      TYPE 105,LIN2
C      TYPE 105,PARAM
105    FORMAT(X,30A2)
C
C
C      GET NUMERICAL INFORMATION FROM THE THIRD LINE
C      OF THE HEADER
C
C      CALL DDCDE
C
C
C      DIVIDE AN AMP-PHASE ARRAY INTO
C      AN AMP ARRAY AND A PHASE ARRAY
C
C      DO 199 NN=1,NDIM
C      AMPL(NN)=AP(2*NN-1)
C      IF(AMPL(NN).GT.35)AMPL(NN)=35.
199    PHS(NN)=AP(2*NN)
C
C      CARY INFORMATION TO 'CIMAG' (TIME DOMAIN ONLY)
C      DO 1 I=1,NDIM
1      P(I,1)=AMPL(I)
C
C      CHECK FOR BAD DATA POINTS, I.E., AMPL(I).GT.995
C
C      CALL ERRF(AMPL,PHS)
C      RETURN
C      END
C
C .....
C      SUBROUTINE REU(INPFILE,AMPL,PHS)
C      THIS PROGRAM READS BACKSCATTER DATA FILES STORED
C      ON VAX DISKS. WITH 750 FORMAT
C
C      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
C      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
C      INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
C
C      BYTE PARAM(60)
C      INTEGER*2 L1(30),L2(30),L3(30)
C      REAL*4 AP(10000)
C      EQUIVALENCE (L3(1),PARAM(1))
C      EQUIVALENCE(L1(1),BUFFER(1)),(L2(1),BUFFER(61))
1      1,(L3(1),BUFFER(121)),(AP(1),BUFFER(361))
1      WRITE(IOUT,2)
2      FORMAT('$','ENTER DATA FILE NAME: ')
10     READ(COM UNIT,10) INPFILE
10     FORMAT(15A2)
10     WRITE(LOG UNIT,10)INPFILE
10     INPFILE(15)=0
10     OPEN(UNIT=8,NAME=INPFILE,TYPE='OLD',FORM='UNFORMATTED',
10     READONLY,ERR=1)
10     READ(8) L1
10     READ(8) L2
10     READ(8) L3
10     TYPE 100,L1
10     TYPE 100,L2
10     TYPE 100,L3
100    FORMAT(X,30A2)

```

```

      DECODE(4,4,PARAM(3),ERR=1001)NDIM
      DECODE(5,5,PARAM(11),ERR=1002)IANST
      ANST=FLOAT(IANST)
      DECODE(5,5,PARAM(20),ERR=1003)IAINC
      AINC=FLOAT(IAINC)
4      FORMAT(I4)
5      FORMAT(I5)
      DO 200 I=1,NDIM
      READ(8) AMPL(I),PHS(I)
      IO=2*I-1
      IE=2*I
      AP(IO)=AMPL(I)
      AP(IE)=PHS(I)
200    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'
99     RETURN
      END

```

```

C .....
      SUBROUTINE TTR(INPFILE)
      BYTE TBUFF(512)
      INCLUDE 'SYSSLIBRARY:FORIOSDEF'
      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
      INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
      INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
C
C
      WRITE(IOUT,1111)
1111   FORMAT(1X,' ENTER DATA FILE NAME:')
6      READ(COM UNIT,2222) INPFILE
2222   FORMAT(15A2)
      WRITE(LOG UNIT,2222) INPFILE
      INPFILE(15)=0
      IB=1
      ICNT=0
8106   OPEN(UNIT=8,NAME=INPFILE,READONLY,TYPE='OLD',IOSTAT=IERR,ERR=8100)
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
      READ(8,80,END=90) TBUFF
80     FORMAT(512A1)
C
C      STORE A BLOCK INTO THE BUFFER ACCORDING TO ITS LENGTH
C
      DO 85 I=1,LEN
      BUFFER(ICNT+I)=TBUFF(I)
      IB=IB+1
      ICNT=ICNT+LEN
      GO TO 82
90     DO 86 I=1,LEN
86     BUFFER(ICNT+I)=TBUFF(I)
C
C      ELIMINATE BLANK SPACES IN BETWEEN EACH CHARACTER
C      IN A FILE HEADER
C

```

```

DO 40 I=1,180
100 BUFFER(I)=BUFFER(2*I-1)
GO TO 331
8100 IF(IERR.EQ.FOR$IOS_FILNOTFOU)THEN
WRITE(IOUT,1112) INPFILE
1112 FORMAT(' FILE ',15A2,' WAS NOT FOUND',/, '$',
1 'ENTER FILENAME AGAIN: ')
ELSE IF (IERR.EQ.FOR$IOS_FILNAMSPE)THEN
WRITE(6,1113) INPFILE
1113 FORMAT('$', '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

```

CC

```

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
C STARTING 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), CNL), (BUFFER(131), CFF), (BUFFER(140), CINC)
EQUIVALENCE (BUFFER(123), TCAS), (BUFFER(124), CNL1)

```

C
C
C
C
C
C
C
C
C

CONVERT CHARACTERS INTO THEIR NUMERICAL EQUIVALENTS

```

IF(ECH.EQ.TCAS) THEN
DECODE(3,102,CNL1,ERR=9)NDIM
ELSE
DECODE(4,101,CNL,ERR=9)NDIM
END IF
100 FORMAT(I5)
101 FORMAT(I4)
102 FORMAT(I3)
103 DECODE(5,100,CFF,ERR=99)IMIN
104 DECODE(5,100,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

```

C
200  FORMAT (/, 'S', 'HAVING PROBLEMS READING HEADER. ENTER ', A4,
1' MANUALLY: ')
C
91  WRITE(6,300)
GO TO 9
C
991  WRITE(6,300)
GO TO 99
C
9991 WRITE(6,300)
GO TO 999
C
300  FORMAT(1X, '***INVALID ENTRY***')
C
END
C .....
SUBROUTINE ERRF(AMPL,PHS)
COMPLEX C1,C2,CD
INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]MAGCMN2.FOR'
INCLUDE 'USER2:[DURAL.CIMAG2]FTRN.FOR'
DO 1 I=1,NDIM
IF(PHS(I).GT.995.) WRITE(6,2) I,AMPL(I),PHS(I)
1 CONTINUE
2 FORMAT(1X,16HERROR AT DATA PT,1I4,4HMAG=,1F10.4,4HPHS=,1F10.4)
C CHECK LEFT HAND END POINT
IF(AMPL(1).GT.100.) THEN
DO 200 I=2,NDIM
IF(AMPL(I).LE.100. .AND. AMPL(I+1).LE.100.) THEN
A1=10.**(AMPL(I)/20.)
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=1,I-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
200 CONTINUE
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

```

```

222      CONTINUE
        GO TO 221
        ELSE
220      CONTINUE
        ELSE
        END IF
C      CHECK INTERIOR POINTS
221      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(I-1)/20.)
        C1=CMPLX(A1*COSD(PHS(I-1)),A1*SIND(PHS(I-1)))
        A2=10.**(AMPL(K)/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)
241      CONTINUE
        GO TO 230
        ELSE
240      CONTINUE
        ELSE
        END IF
230      CONTINUE
      RETURN
      END

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC      MAGCMN.FOR
CC      PROGRAM COMMON BLOCKS
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C

      INTEGER*2      HEAD(256,31),LINE1(30),LINE2(30),LINE3(30)
      INTEGER*2      BUFFERS(400,30)
      INTEGER*4      NB, ISIZE
      INTEGER*4      PNUM,ELEMENT,ELEM
      REAL           VALUE,PVALUE,WEIGHT
      LOGICAL        EFLAG                                !error flag
      LOGICAL        ECHO
      CHARACTER*4     DOMAIN, DOM(40)
      CHARACTER*50     INAME,JNAME,FILNM
      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)

C
C
C
C
      COMMON/BLK1/INAME, JNAME,                                ! The main common block
+      FILNM, TITLE, XLAB, YLAB, P, IDS, S, Q,
+      ARRAY, NUMB, ANG, BUFN, CNTR, CARRAY, CLRTAB,
+      A, CCI, CCX, BCX, DCX, BA, KLA, CA, ASQ, ASS1,
+      ASS2, ASD1, ASC, ASD, ASA, ASB, INFILE, FA,
+      MACRO, BFILE, BUFFERS, DOMAIN, DOM, NB, ECHO,
+      ARRAY2, ISIZE,FNAME
C

```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC          MAGCMN2.FOR
CC      THIS IS THE PROGRAM CONTROLS COMMON BLOCK
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
      INTEGER*4
      +      COM_UNIT ,           ! LU# for the command input
      +      IOU_T ,             ! LU# for the program output
      +      TERM_UNIT ,         ! LU# for the terminal
      +      FILE_UNIT ,         ! LU# for the command file
      +      LOG_UNIT ,          ! LU# for the log file
      +      NULL_UNIT ,         ! LU# of null device for logging routine
      +      STO_UNIT            ! LU# for buffer storage
C
C
      CHARACTER*40  DEF           ! default directory
      LOGICAL       PROC_FLAG
      LOGICAL       DEFINE_FLAG
C
      COMMON/BLK2/COM_UNIT, IOU_T, TERM_UNIT, FILE_UNIT,
      +      DEF, LOG_UNIT, NULL_UNIT, PROC_FLAG,
      +      DEFINE_FLAG

```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC      FTRN.FOR
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
COMMON BUFFER,NDIM,ANST,AINC,FTYPE
DIMENSION AMPL(5000),PHS(5000)
BYTE BUFFER(35000)
INTEGER*2 INPFILE(15)
LOGICAL FTYPE
```

```

CCCCCCCC&CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC
CC      DEFINE THE TAPE HEADER FIELDS
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC
      BYTE      ITITLE(50)      !TITLE
      INTEGER*2 IHED(6)         !DATE AND TIME AS MONTH,DAY,YEAR
C                                     ! HOURS,MINUTES,SECONDS
      BYTE      ITARG(38)       !TARGET LABEL
      INTEGER*2 IANG(3)         !STARTING ANGLE, ANGLE INCREMENT FOR ROTATION
C                                     ! AND NUMBER OF ANGLES IN WHOLE FILE
      BYTE      ITYPE(6)        !ACOUSTIC DATA TYPE PCW,PLFM
      INTEGER*2 IPARAM(6)       !FREQUENCY IN KHZ, SAMPLE INTERVAL, INTERVAL
C                                     ! UNITS, AND NUMBER OF PINGS AT A GIVEN ANGLE
      INTEGER*2 IDEANG          !ELEVATION/DECLINATION ANGLE
      INTEGER*2 IPULTH          !PULSE LENGTH - DIGITS ONLY
      INTEGER*2 IPUNIT          !UNITS FOR PULSE LENGTH
      INTEGER*2 IMODFW          !MODULATION BANDWIDTH FOR PLFM
      INTEGER*2 ISTRNG          !SOURCE-TARGET RANGE IN METERS * 100
      INTEGER*2 ISRRNG          !SOURCE-RECEIVER RANGE IN METERS * 100
      INTEGER*2 IXMTVL          !RMS TRANSMIT VOLTAGE * 100
      INTEGER*2 IRCVGN          !RECEIVER GAIN IN DB * 10
      INTEGER*2 IFTRBW          !RECEIVE FILTER 3-DB BW IN KHZ
      BYTE      IPROJ(20)       !PROJECTOR DESCRIPTION
      INTEGER*2 ITRV            !XMIT LEVEL OF PROJECTOR IN DB/MICRO PA/V*10
      BYTE      IHYD(20)        !RECEIVER DESCRIPTION
      INTEGER*2 IRRS            !RECEIVER SENSITIVITY IN DB/V/MICRO PA*10
      INTEGER*2 IDATR           !NUMBER OF BIOMATION SAMPLES IN A SINGLE PING
C
C
C
      INTEGER*2 HEADER(256)
      COMMON /HEADER/ ITITLE,IHED,ITARG,IANGL,ITYPE,IPARAM,IDEANG,IPULTH,
2                      IPUNIT,IMODFW,ISTRNG,ISRRNG,IXMTVL,IRCVGN,IFTRBW,
3                      IPROJ,ITRV,IHYD,IRRS,IDATR,IOTHER
      EQUIVALENCE (HEADER,ITITLE)

```

CIMAG2 LINKING SUBROUTINES

85

```

I=I+1
IF(I.GT.NP)GO TO 4912
GO TO 7740
7730 K=RF*10
IF(K.GT.2048)GO TO 4912
IF(IS.EQ.1)CALL INTER(R,CA,FA,NP,NS,I,WD,RF)
IF(IS.EQ.0)R=(CA(I+1)-CA(I))/(FA(I+1)-FA(I))*(RF-FA(I))+CA(I)
IF (INORM.EQ.0)R=R*0.02/(SQRT(PI)*DLE)
IF (INORM.EQ.2)R=R/100.
P(K,1)=10.24*(10.+20.*ALOG10(CABS(R)))
P(K+2048,1)=1024.*ATAN2(AIMAG(R),REAL(R))/TWOPI
7720 CONTINUE
4912 RETURN
END

```

C
C

```

SUBROUTINE INTER(R,CA,FA,ICNT,NS,I,WD,RF)
COMPLEX*8 R,CA(2049)
REAL*4 FA(2049)
WEI=0
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 PHCR(DPH,FB,FINC,PH)
REAL*4 PH(201)
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)

```

10 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
PHM=PHM+HAMM*PH(I)
PHM=PHM/WEI
RETURN
END

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC          RDFLE
CC          FREQUENCY DOMAIN READ
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC
      SUBROUTINE RDFLE(A,P,NP,FMIN,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(4098),A(2049),P(2049)
      INTEGER*2 HEADR(256)
      COMMON/HEADER/HEADR

C
C      DEFINE BUFFER STRUCTURE
C
      EQUIVALENCE(LINE1(1),BUFF(1)),(LINE2(1),BUFF(61))
      1,(PARAM(1),BUFF(121)),(AP(1),BUFF(361))
      CALL TR(INFILE,IB,EFLAG)
      IF ( EFLAG ) THEN
                                ! if error has occurred
                                ! then return
      RETURN
      END IF
      TYPE 105,LINE1
      TYPE 105,LINE2
      TYPE 105,PARAM
      FORMAT(X,30A2)
105
C
C
C
C
      PUT INFO IN HEADR BLOCK FOR TRANSFER TO CALLING PROG
C
      DO 108 I=1,30
      HEADR(I)=LINE1(I)
      HEADR(30+I)=LINE2(I)
108  HEADR(60+I)=PARAM(I)
C
C
C
C
      GET NUMERICAL INFORMATION FROM THE THIRD LINE
      OF THE HEADER
C
      CALL DCDE(NP,FMIN,FINC,EFLAG)
      IF (EFLAG) THEN
                                ! if error return
      RETURN
      END IF

C
C      DIVIDE AN AMP-PHASE ARRAY INTO
C      AN AMP ARRAY AND A PHASE ARRAY
C
      DO 199 NN=1,NP
      A(NN)=AP(2*NN-1)
199  P(NN)=AP(2*NN)
688  FORMAT(A1)
689  FORMAT(1X,5(2F12.3,1H;))
      FMAX=FMIN+(NP-1)*FINC
      RETURN
      END
C
C

```

```

SUBROUTINE TR(INFILE,IB,EFLAG)
INCLUDE 'MAGCMN2.FOR'
LOGICAL EFLAG                                ! error flag
INTEGER*2 IBUFF(10000)
CHARACTER*70 INFILE
INTEGER*4 FOR RETCODE                        ! fortran return code
INCLUDE 'USER2:[DURAL.CIMAG2]MSGBLK.FOR'    ! error message declarations
COMMON BUFF
BYTE BUFF(20000),TBUFF(1500)
EQUIVALENCE (BUFF(1),IBUFF(1))
INCLUDE 'SYSSLIBRARY:FORIOSDEF'
WRITE(6,5)
5  FORMAT(1X,'TYPE DATA FILE NAME')
READ (COM UNIT,10) INFILE
WRITE (LOG UNIT,10) INFILE
IF ((INFILE.EQ. 'NAME.DAT') .AND. DEFINE_FLAG) THEN    ! if defining
C                                     a procedure then
                                     INFILE='USER2:[DURAL.CIMAG2]NAME.DAT' ! use the dummy file
END IF
10  FORMAT( A )
IB=1
ICNT=0
8106 + OPEN(UNIT=8,NAME=INFILE,TYPE='OLD',READONLY,IOSTAT=FOR_RETCODE,
      + DEFAULTFILE=DEF,ERR=81)
      GOTO 82
81  IF (FOR_RETCODE.EQ. 29) THEN                    ! if file is not found
      CALL LIB$SIGNAL(MAG_FILNOTFOU) ! write out file not found error
      EFLAG = .TRUE.                      ! error flag
      RETURN
    ELSE
      CALL LIB$SIGNAL(MAG_COM)              ! else command error
      EFLAG = .TRUE.                      ! flag the error
      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
    READ(8,80,END=90)(TBUFF(I),I=1,512)
80  FORMAT(512A1)
C
C  STORE A BLOCK INTO THE BUFFER ACCORDING TO ITS LENGTH
C
    DO 85 I=1,LEN
85  BUFF(ICNT+I)=TBUFF(I)
      IR=IR+1
      ICNT=ICNT+LEN
      GO TO 82
    DO 86 I=1,LEN
90  BUFF(ICNT+I)=TBUFF(I)
86
C  ELIMINATE BLANK SPACES IN BETWEEN EACH CHARACTER
C  IN A FILE HEADER
C
    DO 40 I=1,180
40  BUFF(I)=BUFF(2*I-1)
331 CLOSE(UNIT=8,DISP='SAVE')
      RETURN
      END
C
C  SUBROUTINE DCDE(NP,FMIN,FINC,EFLAG)

```


AD-A191 847

CINAG2: THE COMPUTER PROGRAM TO GENERATE COLOR IMAGES

2/2

(U) OHIO STATE UNIV COLUMBUS ELECTROSCIENCE LAB

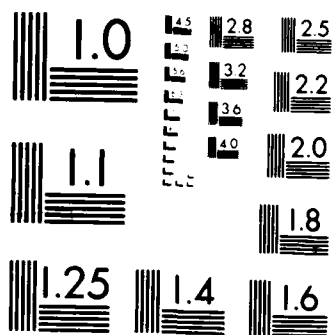
G DURAL ET AL NOV 87 ESL-718848-7 N88014-86-K-0282

UNCLASSIFIED

F/G 12/5

NL





30	IJ=IJ+2	FORT 097
	IF(IFS)32,2,36	FORT 098
C	DOING FOURIER ANALYSIS,SO DIV. BY N AND CONJUGATE.	FORT 099
32	FN = N	FORT 100
	DO 34 I=1,N	FORT 101
	A(2*I-1) = A(2*I-1)/FN	FORT 102
34	A(2*I)=-A(2*I)/FN	FORT 103
C	SPECIAL CASE- L=1	FORT 104
36	DO 40 I=1,N,2	FORT 105
	T = A(2*I-1)	FORT 106
	A(2*I-1) =T + A(2*I+1)	FORT 107
	A(2*I+1)=T-A(2*I+1)	FORT 108
	T=A(2*I)	FORT 109
	A(2*I) = T + A(2*I+2)	FORT 110
40	A(2*I+2)= T - A(2*I+2)	FORT 111
	IF(M-1) 2,1 ,50	FORT 112
C	SET FOR L=2	FORT 113
50	LEXP1=2	FORT 114
C	LEXP1=2**(L-1)	FORT 115
	LEXP=8	FORT 116
C	LEXP=2**(L+1)	FORT 117
	NPL= 2**MT	FORT 118
C	NPL = NP* 2**-L	FORT 119
60	DO 130 L=2,M	FORT 120
C	SPECIAL CASE- J=0	FORT 121
	DO 80 I=2,N2,LEXP	FORT 122
	I1=I + LEXP1	FORT 123
	I2=I1+ LEXP1	FORT 124
	I3 =I2+LEXP1	FORT 125
	T=A(I-1)	FORT 126
	A(I-1) = T +A(I2-1)	FORT 127
	A(I2-1) = T-A(I2-1)	FORT 128
	T =A(I)	FORT 129
	A(I) = T+A(I2)	FORT 130
	A(I2) = T-A(I2)	FORT 131
	T= -A(I3)	FORT 132
	TI = A(I3-1)	FORT 133
	A(I3-1) = A(I1-1) - T	FORT 134
	A(I3) = A(I1) - TI	FORT 135
	A(I1-1) = A(I1-1) +T	FORT 136
80	A(I1) = A(I1) +TI	FORT 137
	IF(L-2) 120,120,90	FORT 138
90	KLAST=N2-LEXP	FORT 139
	JJ=NPL	FORT 140
	DO 110 J=4,LEXP1,2	FORT 141
	NPJJ=NT-JJ	FORT 142
	UR=S(NPJJ)	FORT 143
	UI=S(JJ)	FORT 144
	ILAST=J+KLAST	FORT 145
	DO 100 I= J,ILAST,LEXP	FORT 146
	I1=I+LEXP1	FORT 148
	I2=I1+LEXP1	FORT 149
	I3=I2+LEXP1	FORT 150
	T=A(I2-1)*UR-A(I2)*UI	FORT 151
	TI=A(I2-1)*UI+A(I2)*UR	FORT 152
	A(I2-1)=A(I-1)-T	FORT 153
	A(I2)=A(I) - TI	FORT 154
	A(I-1) =A(I-1)+T	FORT 155
	A(I) =A(I)+TI	FORT 156
	T=-A(I3-1)*UI-A(I3)*UR	FORT 157
	TI=A(I3-1)*UR-A(I3)*UI	FORT 158
	A(I3-1)=A(I1-1)-T	FORT 159
	A(I3) =A(I1)-TI	FORT 160
	A(I1-1)=A(I1-1)+T	FORT 161
100	A(I1) =A(I1) +TI	FORT 162
C	END OF I LOOP	

110	JJ=JJ+NPL	FORT 163
C	END OF J LOOP	FORT 164
120	LEXP1=2*LEXP1	FORT 165
	LEXP = 2*LEXP	FORT 166
130	NPL=NPL/2	FORT 167
C	END OF L LOOP	FORT 168
140	IF(IFS)145,2,1	FORT 169
C	DOING FOURIER ANALYSIS. REPLACE A BY CONJUGATE.	FORT 170
145	DO 150 I=1,N	FORT 171
150	A(2*I) =-A(2*I)	FORT 172
160	GO TO 1	FORT 173
C	RETURN	FORT 174
C	MAKE TABLE OF S(J)=SIN(2*PI*J/NP),J=1,2,...,NT-1,NT=NP/4	FORT 175
200	NP=N	FORT 176
	MP=M	FORT 177
	NT=N/4	FORT 178
	MT=M-2	FORT 179
	IF(MT) 260,260,205	FORT 180
205	THETA=.7853981634	FORT 181
C	THETA=PI/2**((L+1) FOR L=1	FORT 182
210	JSTEP = NT	FORT 183
C	JSTEP = 2**((MT-L+1) FOR L=1	FORT 184
	JDIF = NT/2	FORT 185
C	JDIF = 2**((MT-L) FOR L=1	FORT 186
	S(JDIF) = SIN(THETA)	FORT 187
	IF (MT-2)260,220,220	FORT 188
220	DO 250 L=2,MT	FORT 189
	THETA = THETA/2.	FORT 190
	JSTEP2 = JSTEP	FORT 191
	JSTEP = JDIF	FORT 192
	JDIF = JDIF/2	FORT 193
	S(JDIF)=SIN(THETA)	FORT 194
	JC1=NT-JDIF	FORT 195
	S(JC1)=COS(THETA)	FORT 196
	JLAST=NT-JSTEP2	FORT 197
	IF(JLAST-JSTEP)250,230,230	FORT 198
230	DO 240 J=JSTEP,JLAST,JSTEP	FORT 199
	JC=NT-J	FORT 200
	JD=J+JDIF	FORT 201
240	S(JD)=S(J)*S(JC1)+S(JDIF)*S(JC)	FORT 202
250	CONTINUE	FORT 203
260	IF(IFS)20,1,20	FORT 204
	END	FORT 205


```

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$IOS_FILNOTFOU)THEN
TYPE 1112,INFILE
1112 FORMAT(' FILE : ',A20,' DOES NOT EXIST',//
2,' ENTER FILENAME AGAIN')
ELSE IF (IERR.EQ.FOR$IOS_FILNAMSPE)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:[GRP11LIB]BFILES.FOR'

```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC          CIMAGMSG
CC    PROGRAM ERROR MESSAGES
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
CC

```

```

.TITLE          CIMAG_MESSAGES  Program error messages  ! module name and
C              listing title
.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

```

COLOR IMAGING PROGRAM CLRPL

97

```

ELSE
END IF
C
OPEN(UNIT=1,FILE=FILNM,STATUS='OLD',FORM='UNFORMATTED')
C
ENTER THE IMAGE SIZE AND PERIOD OF THE TIME SIGNAL
READ(1) IMGSZ,PER
C
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))
1000 CONTINUE
C
NORMALIZE THE ARRAY VALUES
C
CALL SEARCH(ARRAY,AMAX,AMIN)
WRITE(6,*)'MAX=',AMAX,'MIN=',AMIN
WRITE(6,*)'ENTER THE DESIRED MAX.,AND,MIN.'
READ(5,*) 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)
20 CONTINUE
C
OPEN GKS ERROR FILE
CALL GOPKS(6,5000)
C
FIND CONNECT ID
999 CALL GKHGCI('ESL 4129',JERROR,KCONID)
IF(JERROR.NE.0) THEN
WRITE(6,*) 'Can not be a connection ID'
WRITE(6,*) 'Would you like to wait ? (Y=1)'
READ(5,*)ANS
IF(ANS.NE.1) THEN
STOP
ELSE
WRITE(6,*) 'Enter 1 when ready'
READ(5,*) ANS
GO TO 999
END IF
END IF
C
OPEN WORKSTATION #1
CALL GOPWK(1,KCONID,KWK)
C
ACTIVATE WORKSTATION #1
CALL GACWK(1)
C
SET THE WORKSTATION WINDOW/VIEWPORT-FULLSCREEN
GET MAX X AND Y
KUNITS=0
CALL GQDSP(KWK,KERROR,KUNITS,XSIZE,YSIZE,KRASX,KRASY)
CALL GSWKWN(1.0.,1.,0.,YSIZE/XSIZE)
CALL GSWKVP(1.0.,.343,0.,.274)
C
GENERATE THE COLOR INDICES
C
IF(POL.EQ.0) THEN
DO 3 COLI=1,100
L=.5
S=1

```

```

        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
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 GSCHXP(1.25)
      CALL GSCHSP(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 GSCHXP(1.)
      CALL GSCHSP(1.)
C
C      PLOT COLOR MATIX USING CELL ARRAY
      TEST=10
      IF (TEST.FQ.0)GO TO 21
      CALL GCA(GX(1),GY(1),GX(3),GY(3),100,100,1,1,100,100,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

```

```

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
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 GSCHUP(1.,0.)
CALL GTX(.82,.4,'90 DEG.')
CALL GTX(.20,.40,'TIME IN NANoseconds')
C
C FIGURE OUT THE COORDINATES ON EACH AXES
CALL MARKS(IMGSZ,PER,COORD)
C
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
C
C COLOR 1 = WHITE, 0 = BLACK (ON PLOTTER, REVERSED ON SCREEN)
C
C CALL GCLSG(SEG)
C
C WRITE(6,*)'Enter return to finish.....'
READ(5,1)FINISHED

```

```

1      FORMAT(A1)
      CALL GCLRWR(WKSTID,1)           !Clears the screen on TEXTRONIX

C      DEACTIVATE THE WORK STATION
      CALL GDAWK(1)

C
C      CLOSE THE WORK STATION
      CALL GCLWR(1)

C      CLOSE THE SYSTEM
      CALL GCLKS
      STOP
      END

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC      THIS ROUTINE MAKES THE TRANSFORMATION BETWEEN COLOR SYSTEMS
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C

```

```

      SUBROUTINE HLSRGB(H,L,S,R,G,B)
      REAL H,L,S,R,G,B,M1,M2
      IF (L .LE. .5) THEN
        M2=L*(1+S)
      ELSE
        M2=L+S-L*S
8      END IF
      M1=2*L-M2
      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

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      THIS ROUTINE FINDS THE MAX., AND MIN. OF A 100*100 ARRAY
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C

```

```

      SUBROUTINE SEARCH(ARRAY,AMAX,AMIN)
      DIMENSION ARRAY(100,100)
      AMAX=-1000
      AMIN=1000
      DO 1 I=1,100
        DO 1 J=1,100

```

```

                IF (ARRAY(I,J).GT.AMAX) AMAX=ARRAY(I,J)
                IF (ARRAY(I,J).LT.AMIN) AMIN=ARRAY(I,J)
1      CONTINUE
        RETURN
        END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      THIS ROUTINE CALCULATES THE DIVISIONS ON THE AXES
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      SUBROUTINE MARKS(IMG SZ,PER,COORD)
        DIMENSION COORD(5)
        SIZE=PER*IMG SZ/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