General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by the NASA Center for Aerospace Information (CASI)
SPECIAL REPORT
on the
GRAPHICS AND DATA ACQUISITION
SOFTWARE PACKAGE

Prepared for the NASA/Johnson Space Center
Biomedical Research Laboratories

by William G. Crosier

December 1, 1981

Contract NAS 9-14880

TECHNOLOGY INCORPORATED
LIFE SCIENCES DIVISION
P. O. Box 58827
Houston, Texas 77058

Technology Incorporated
Life Sciences Division
P.O. Box 58827
Houston, Texas 77058
Abstract

A new software package has been developed for use with micro and minicomputers, particularly Digital Equipment Corporation's LSI-11/PDP-11 series. The package has a number of Fortran-callable subroutines which perform a variety of frequently needed tasks for biomedical applications. All routines are well documented, flexible, easy to use and modify, and require minimal programmer knowledge of peripheral hardware. The package is also economical of memory and CPU time. A single subroutine call can perform any one of the following functions: (1) Plot an array of integer values from sampled A/D data; (2) Plot an array of Y values versus an array of X values; (3) Draw horizontal and/or vertical grid lines of selectable type; (4) Annotate grid lines with user units; (5) Get coordinates of user-controlled crosshairs from the terminal for interactive graphics; (6) Sample any analog channel with program selectable gain; (7) Wait a specified time interval; and (8) Perform random access I/O of one or more blocks of a sequential disk file. Several miscellaneous functions are also provided. These routines are modular and easily changed, and are especially applicable for uses in biomedical research laboratories such as NASA's where adaptability is important and software development time is limited. Complete source code listings, example main programs, and sample output are included.
APPROVAL SHEET

for the

GRAPHICS AND DATA ACQUISITION
SOFTWARE PACKAGE

Approved by:

Millard Raschke, Ph.D. (Systems Requirements)
NASA/JSC Neuroscience Laboratory

Edward C. Moseley, Ph.D.
NASA/JSC Medical Sciences Division

Phillip C. Johnson, M.D. (Branch Chief)
NASA/JSC Medical Research Branch

Joseph T. Baker (Section Supervisor)
Technology Incorporated

Harry Hairescher (Project Manager)
Technology Incorporated
PERSONNEL

William G. Crosier - Software Design and Development, Documentation

Kay Elton - Word Processing for Documentation
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td>Approval Sheet</td>
<td>11</td>
</tr>
<tr>
<td>Personnel</td>
<td>111</td>
</tr>
<tr>
<td>Contents</td>
<td>iv</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Sample Main Program</td>
<td>8</td>
</tr>
<tr>
<td><strong>Graphics Subroutines</strong></td>
<td></td>
</tr>
<tr>
<td>GRINIT -- Initialize Graphics Parameters</td>
<td>30</td>
</tr>
<tr>
<td>MPLOT - Draw a Line</td>
<td>33</td>
</tr>
<tr>
<td>COPY - Make a Hard Copy</td>
<td>37</td>
</tr>
<tr>
<td>ERASE - Erase (clear) Screen</td>
<td>39</td>
</tr>
<tr>
<td>CHRSIZ - Set Character Size</td>
<td>41</td>
</tr>
<tr>
<td>GINPUT - Interactive Graphics Input</td>
<td>43</td>
</tr>
<tr>
<td>ARYPLT - Plot an Integer Array</td>
<td>46</td>
</tr>
<tr>
<td>XYPLOT - Plot one Real Array vs. Another</td>
<td>49</td>
</tr>
<tr>
<td>GRID - Draw Grid Lines</td>
<td>53</td>
</tr>
<tr>
<td>ANOTAT - Annotate (Label) Grid with User Units</td>
<td>56</td>
</tr>
<tr>
<td><strong>Data Acquisition and Miscellaneous Subroutines</strong></td>
<td></td>
</tr>
<tr>
<td>BELL - Ring Terminal Bell/Beep</td>
<td>60</td>
</tr>
<tr>
<td>WAIT - Wait for a Specified Time</td>
<td>62</td>
</tr>
<tr>
<td>ISAMPA - Sample an Analog Channel</td>
<td>65</td>
</tr>
<tr>
<td>DISKIO - Perform Random-Access Disk I/O</td>
<td>69</td>
</tr>
</tbody>
</table>
Introduction

Numerous experiments in biomedical research laboratories involve the collection and display of sampled analog data using a mini or microcomputer. Often, however, many experiments which initially seem simple, turn out to involve a considerable investment in programming effort because of the difficulty in finding suitable graphics packages, or in using such devices as analog to digital (A/D) converters in a high level language, such as Fortran. Most data acquisition routines (except for those involving relatively low sampling rates) tend to be application specific, and are written in assembly language by individuals who must have a fairly intimate knowledge both of the particular A/D systems used and of the assembly language of the computer. In addition, biomedical data often seems particularly well suited for graphical displays, because of the complexity of physiological data and also because of the ease of interpretation of graphs by medical or other personnel without an extensive computer or statistical background. Graphics software packages are available from various sources, but most are not particularly well suited for use with microcomputers. Many of the commercially available packages are quite large and have features that are much more extensive than needed in many applications.

Some of the features are used rarely, if at all, but their presence still requires a rather large amount of main memory and disk space in the system. The architecture of some of these packages can be quite complicated, and with many of their routines rather poorly documented, it can be a formidable task for a programmer to remove the extra, unneeded features from such a package without interfering with the basic functions that are required. As a result, these large packages are often only suited for larger computer facilities, where large amounts of memory are available and where numerous users may make use of the myriad of features offered.

Another problem with some graphics packages is that they are not particularly easy to use with real time interactive programs. Although fine for offline plotting applications, they may be too slow or too difficult to use in cases where real time data acquisition and control of an experiment are taking place. In addition, some packages are not sufficiently well
documented, particularly with comments in the subroutines or with examples, to allow them to be used or modified easily by programmers without extensive knowledge of the hardware being used. Sometimes, it can take a naive programmer several weeks to write one fairly simple program, because of the time required to discover the peculiarities of analog to digital converters and graphics terminals or plotters. The plethora of functions available in some packages also seems only to cause confusion among some users, since many of the functions are redundant and do not have to be used at all.

Because of these problems with commercially available software, a new data acquisition graphics package was developed for use at the NASA Johnson Space Center Life Sciences Laboratories. It was designed for use specifically with micro or minicomputers, particularly Digital Equipment Corporation's LSI-11 and PDP-11 series, using the RT-11 operating system. Desired features of the package include the following: (1) It should be relatively small in size so that it can be used easily with microcomputers having limited memory (56K bytes or less). (2) It should include the most frequently needed graphics, analog to digital conversion (A/D), and miscellaneous capabilities. However, extra functions which are not commonly needed or which can be performed with alternate methods, and which require too much memory, should not be included. (3) The package should be modular, flexible, easy to learn, use, and modify. Additionally, use of the package should require only a minimal knowledge of the peripheral hardware. The package should be adaptable to various types of Tektronix terminals or possibly to X-Y plotters. It should also be usable with Fortran main programs. (4) In order to meet the previous objective, the package should be written primarily in a high level language, and should be well commented and otherwise documented as well. (5) Finally, it also should be able to handle moderately fast analog to digital conversion and displays for real-time applications.

With the above objectives in mind, a software package was developed which has the following characteristics. First, it consists of a number of Fortran-callable subroutines which perform all necessary tasks required for interfacing with an A/D system or various types of Tektronix terminals. Only a few of these routines would need to be modified if a different type of terminal or X-Y plotter was used. Further, most of the subroutines themselves
are written in Fortran and all are fully commented so that they can be easily
modified by others, if necessary. In addition, a few other simple
miscellaneous routines were written to perform other tasks used frequently in
laboratory applications. A list of all of the routines in the package is
shown in Table 1.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRINIT</td>
<td>Initialize graphics parameters for other routines</td>
</tr>
<tr>
<td>MPLLOT</td>
<td>Change position (move cursor), go to alphanumeric mode, or draw vector</td>
</tr>
<tr>
<td>COPY</td>
<td>Make hard copy of Tektronix terminal screen &amp; erase it if desired</td>
</tr>
<tr>
<td>ERASE</td>
<td>Erase Tektronix screen without copying</td>
</tr>
<tr>
<td>CHRSIZ</td>
<td>Change character size</td>
</tr>
<tr>
<td>GINPUT</td>
<td>Switch to graphic input mode, display &amp; get coordinates of user-controlled crosshairs</td>
</tr>
<tr>
<td>ARYPLT</td>
<td>Plot an integer array of Y values (A/D samples, etc), with straight lines connecting points, with variable scaling</td>
</tr>
<tr>
<td>XYPLT</td>
<td>Plot an array of real Y values vs an array of real X values, with straight interconnecting lines, with variable scaling</td>
</tr>
<tr>
<td>GRID</td>
<td>Draw grid lines (selectable type) over desired area</td>
</tr>
<tr>
<td>ANOTAT</td>
<td>Label plot axes with user units at some or all grid lines</td>
</tr>
<tr>
<td>BELL</td>
<td>Ring terminal bell/beep (Variable duration &amp; modulation control)</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait a desired period of time (1/60 sec resolution)</td>
</tr>
<tr>
<td>ISAMPA</td>
<td>Sample an A/D channel with selectable gain &amp; channel number</td>
</tr>
<tr>
<td>DISKIO</td>
<td>Perform random access multiblock binary I/O to a sequential disk file</td>
</tr>
</tbody>
</table>
A few of these routines also use subroutines from the Fortran library in the system to access the system line frequency clock, to send characters to the terminal through the terminal handler, to perform double word integer arithmetic, and do disk I/O. These few routines in the package would have to be modified if a different computer or operating system was used. In addition, routines MPLOT, COPY, ERASE, and possibly GRINIT would have to be changed if a different terminal or plotter was used. However, such changes should be fairly straightforward once the characteristics of any particular terminal were determined.

The package will presently support the following hardware on an LSI-11 series microcomputer. First, analog to digital conversion can be performed with LSI-11 compatible A/D systems manufactured by ADAC, Data Translation, or DEC, using different modes. Programmable gain and random channel selection are supported. A throughput rate of 4000 samples per second or more can be accomplished if little or no computation is performed between samples. Furthermore, all of the Tektronix 4000 series graphics terminals can be used with the package. Features of some of these terminals which are supported, but not required, include: (1) 12-bit addressing for X and Y coordinates, (2) variable character sizes and dot-dash line types (using the hardware built into the terminals), (3) graphic input mode for interactive applications (so that a program can determine the positions of user controlled crosshairs), and (4) write-through mode for non-stored (refreshed) displays. Additionally, several different types of X-Y plotters could be used with the package, although routine MPLOT would have to be modified to accommodate the requirements of the particular plotter used.

The remainder of this special report consists of the following. First, some simple example main programs are given in the next section in order to demonstrate the use of, and capabilities of, some of the routines in the package. Graphical outputs from the sample programs are also provided. Following that section, each subroutine in the package is discussed separately, with information on calling conventions, parameters (arguments) to be passed, and any restrictions or additional details concerning its operation. Source listings of each subroutine are also given in each section.
Additional examples and other programs are provided in the references below.

References


INSTALLATION

All of the subroutines in this package are callable by Fortran programs and are designed to be used with DEC's RT-11 operating system. For convenience, some or all of them may be placed in a library on the same or a different disk as the System Subroutine Library (SYSLIB).

For example, in order to create a library called LABLIB containing routines GRINIT, MPLOT, ARYPLT, and ISAMPA, do the following. First, put the source code for each routine in separate files on the default device DK:, each with the name of the subroutine: GRINIT.FOR, MPLOT.FOR, ARYPLT.FOR, and ISAMPA.MAC. Second, compile and assemble these routines with the RT-11 (Version 4) Fortran compiler (Version 02.1 or later) and Macro assembler (Version 4 or later):

FORT/WARN/LIST:DK: (GRINIT,MPLOT,ARYPLT)
MACRO/CR/LIST:DK: ISAMPA

Next, create the actual library file LABLIB.OBJ from the individual object files:

LIBR/CREATE LABLIB
Files? GRINIT,MPLOT,ARYPLT,ISAMPA

The same procedure may also be followed for more subroutines, but you should include only six or fewer file names on any command line. If you want to put routines from more than six files in the library, then include the "/PROMPT" option after the command "LIBR", then specify six or fewer file names per line. After the last line, type two slashes (//) to terminate the file name entry. Refer to the RT-11 System User's Guide for more information.

You will note, in the descriptions of several of the subroutines, that they in turn call other subroutines. For instance, MPLOT and WAIT are called by a number of other routines. Make sure that all of the subroutines which are needed (either directly or indirectly) are included in the library which you build. Otherwise, an undefined global error will occur when you attempt
to link the programs. In addition, several DEC supplied routines from the System Subroutine Library are used, and these must be present in SYSLIB. Such routines include ITTOUR (called by MPLOT, COPY, ERASE, CHRSIZ, BELL, etc.) and GTIM and the Integer*4 routines (called by WAIT). Refer to the descriptions of the appropriate routines in this report for more information.
Sample Main Programs

This section contains several simple programs which demonstrate some of the capabilities of the subroutines in this package. By referring to the program listings and sample output which follow, and running these programs yourself, you should be able to learn how to use the subroutines to perform other similar functions.

The first example program is ADTEST. This uses only the routine ISAMPA from the subroutine package. In addition, ITTINR and IPOKE from the System Subroutine Library are also used. More information on these last two routines is in the DEC RT-11 Version 4 Programmer's Reference Manual.

Program ADTEST does the following. First, it asks the user for the types of analog-to-digital (A/D) and digital-to-analog (D/A) converters present in the system. Second, the program asks which analog input channel should be sampled, and the programmable gain code to use. After this, the program samples the specified channel with subroutine ISAMPA and sends the converted signal back out to the D/A converter(s) in the system where an oscilloscope can be used to monitor the D/A output. The sampling is repeated until the user strikes the Return key. The D/A output resembles a sampled (choppy) version of the input analog signal, with a scale factor dependent on the programmable gain, A/D range jumpers, and D/A range jumpers selected.

Program ADTEST may also be used for diagnostic purposes, since it enables one to quickly check both the A/D and D/A converters plus certain CPU and memory functions, without being particularly difficult to use. In addition, it may be easily modified for other purposes. The program listing and sample output follows.
PROGRAM ADTEST

PURPOSE: TEST A/D & D/A BOARD

METHOD: SAMPLE SELECTED A/D CHANNEL & TRANSFER SAMPLED VALUE TO BOTH D/A'S, AND REPEAT UNTIL INTERRUPTED BY USER. USER MAY INPUT 10 HZ TRIANGLE WAVE FROM SIGNAL GENERATOR, OR ANY OTHER SIGNAL, AND D/A'S SHOULD FOLLOW THE INPUT. SCALING MAY BE DIFFERENT, HOWEVER, DEPENDING ON PROGRAMMABLE GAIN AND VOLTAGE RANGES SELECTED.

WRITTEN BY: WILLIAM G. CROSIER
REVISED: 23 DEC. 1981

SUBROUTINES REQUIRED: ISAMPA

INTEGER DAC1

SET UP DEVICE ADDRESSES

DAC1 = '176760' ! D/A #1 DATA REGISTER

TYPE *'A/D & D/A CONVERTER TEST PROGRAM -- VERSION 3'
TYPE *'ENTER A/D TYPE: 0 FOR ADAC, OR'
TYPE *' 1 FOR DATA TRANSLATION'
ACCEPT **IADTYF
IF (IADTYF .LT. 0 .OR. IADTYF .GT. 1) STOP 'ILLEGAL A/D TYPE'

TYPE *'ENTER D/A TYPE: 0 FOR ADAC, OR'
TYPE *' 1 FOR DATA TRANSLATION'
ACCEPT **IDATYP
IF (IDATYP .LT. 0 .OR. IDATYP .GT. 1) STOP 'ILLEGAL D/A TYPE'

TYPE *'ENTER NUMBER OF D/A CONVERTERS ON BOARD (1-4)'
ACCEPT **NDACS
IF (NDACS .LT. 1 .OR. NDACS .GT. 4) STOP 'ILLEGAL NO. OF DACS'

TYPE *'VERIFY THAT A/D IS JUMPERED FOR BIPOLAR,'
TYPE *'2'S COMPLEMENT OPERATION, WITH CSR ADDRESS'
TYPE *'OF 176770 OCTAL'
TYPE *'ALSO VERIFY THAT D/A IS JUMPERED FOR BIPOLAR'
TYPE *'OPERATION, WITH ADDR OF FIRST D/A OF 176760 OCTAL,'
TYPE *'ENTER -1 FOR INPUT CHANNEL NO. TO STOP.'

REQUEST A/D CONTROL PARAMETERS

5 TYPE 1005
FORMAT( 'OA/D INPUT CHANNEL NO. ?',$)
ACCEPT **ICHAN
IF (ICHAN .LT. 0 .OR. ICHAN .GT. 31) STOP

20 TYPE **PROGRAMMABLE GAIN CODE:'
IF (IADTYF .EQ. 0) TYPE 1020
1020 FORMAT ( '0 = GAIN OF 10' / '1 = GAIN OF 5' /
```fortran
0034   IF (IADTYP .EQ. 1) TYPE 1030
0036   1030 FORMAT (' 0 = GAIN OF 1'/ ' 1 = GAIN OF 2'/
0037     ' 2 = GAIN OF 4'/ ' 3 = GAIN OF 8')
0038   TYPE *,'GAIN CODE DESIRED (0-3)'
0039   ACCEPT *,'IPGNCD'
0040   IF (IPGNCD.LT.0 .OR. IPGNCD.GT.3) GO TO 20
0041   TYPE *,'HIT RETURN KEY TO TERMINATE SAMPLING & CHANGE A/D'
0042   TYPE *,' CHANNEL NO. OR PROGRAMMABLE GAIN.'
0043   DO A/D CONVERSION & OUTPUT DATA TO BOTH D/A'S
0044   UNTIL USER STRIKES RETURN KEY ON TERMINAL.
0045   IF (ITTINR() .GE. 0) GO TO 5 !HAS RETURN KEY BEEN HIT?
0046   IDATA = ISAMPA(ICHAN,IPGNCD,IADTYP) !SAMPLE A/D CHAN.
0047   IF DATA TRANSLATION D/A, CONVERT CODING TO OFFSET BINARY
0048   IF (IADTYP .EQ. 1) IDATA=IDATA+3048
0049   IDAC = DAC1 !D/A CONV. ADDR.
0050   DO 50 K=1,NDACS !FOR EACH D/A,
0051   CALL IPOSE(IDAC,IDATA) ! OUTPUT SAMPLE TO D/A
0052   GO TO 40
0053   END
```

### FORTRAN IV Storage Map for Program Unit ADTEST

Local Variables: .PSECT $DATA, Size = 000022 ( 9, words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC1</td>
<td>I*2</td>
<td>0000000</td>
<td>IADTYP</td>
<td>I*2</td>
<td>0000002</td>
<td>ICHAN</td>
<td>I*2</td>
<td>000010</td>
</tr>
<tr>
<td>IDAC</td>
<td>I*2</td>
<td>000010</td>
<td>IDATA</td>
<td>I*2</td>
<td>000014</td>
<td>IDATYP</td>
<td>I*2</td>
<td>000004</td>
</tr>
<tr>
<td>IPGNCD</td>
<td>I*2</td>
<td>000012</td>
<td>K</td>
<td>I*2</td>
<td>000020</td>
<td>NDACS</td>
<td>I*2</td>
<td>000000</td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPOSE</td>
<td>I*2</td>
<td>ISAMPA</td>
<td>I*2</td>
<td>ITTINR</td>
<td>I*2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RUN ADTEST
A/D & D/A CONVERTER TEST PROGRAM -- VERSION 3
ENTER A/D TYPE: 0 FOR ADAC. OR
1 FOR DATA TRANSLATION
0
ENTER D/A TYPE: 0 FOR ADAC. OR
1 FOR DATA TRANSLATION
0
ENTER NUMBER OF D/A CONVERTERS ON BOARD (1-4)
2
VERIFY THAT A/D IS JUMPERED FOR BIPOLAR.
2'S COMPLEMENT OPERATION WITH CSR ADDRESS
OF 176770 OCTAL
ALSO VERIFY THAT D/A IS JUMPERED FOR BIPOLAR
OPERATION. WITH ADDR OF FIRST D/A OF 176780 OCTAL
ENTER -1 FOR INPUT CHANNEL NO TO STOP
A/D INPUT CHANNEL NO. ?
PROGRAMMABLE GAIN CODE:
GAIN CODE DESIRED (0-3)?
1
GAIN CODE DESIRED (0-3)?
2
GAIN CODE DESIRED (0-3)?
3
HIT RETURN KEY TO TERMINATE SAMPLING & CHANGE A/D
CHANNEL NO. OR PROGRAMMABLE GAIN

A/D INPUT CHANNEL NO. ?
PROGRAMMABLE GAIN CODE:
0 = GAIN OF 10
1 = GAIN OF 5
2 = GAIN OF 2
3 = GAIN OF 1
GAIN CODE DESIRED (0-3)?
1
GAIN CODE DESIRED (0-3)?
2
GAIN CODE DESIRED (0-3)?
3
HIT RETURN KEY TO TERMINATE SAMPLING & CHANGE A/D
CHANNEL NO. OR PROGRAMMABLE GAIN.

A/D INPUT CHANNEL NO. ?
PROGRAMMABLE GAIN CODE:
0 = GAIN OF 10
1 = GAIN OF 5
2 = GAIN OF 2
3 = GAIN OF 1
GAIN CODE DESIRED (0-3)?
1
GAIN CODE DESIRED (0-3)?
2
GAIN CODE DESIRED (0-3)?
3
HIT RETURN KEY TO TERMINATE SAMPLING & CHANGE A/D
CHANNEL NO. OR PROGRAMMABLE GAIN.

A/D INPUT CHANNEL NO. ?
PROGRAMMABLE GAIN CODE:
0 = GAIN OF 10
1 = GAIN OF 5
2 = GAIN OF 2
3 = GAIN OF 1
GAIN CODE DESIRED (0-3)?
1
GAIN CODE DESIRED (0-3)?
2
GAIN CODE DESIRED (0-3)?
3
HIT RETURN KEY TO TERMINATE SAMPLING & CHANGE A/D
CHANNEL NO. OR PROGRAMMABLE GAIN.

A/D INPUT CHANNEL NO. ?
PROGRAMMABLE GAIN CODE:
0 = GAIN OF 10
1 = GAIN OF 5
2 = GAIN OF 2
3 = GAIN OF 1
GAIN CODE DESIRED (0-3)?
1
GAIN CODE DESIRED (0-3)?
2
GAIN CODE DESIRED (0-3)?
3
HIT RETURN KEY TO TERMINATE SAMPLING & CHANGE A/D
CHANNEL NO. OR PROGRAMMABLE GAIN.
A/D INPUT CHANNEL NO. 7-1  Enter -1 to stop.
STOP --
The second sample program is called DEMOGR. This program demonstrates the use of GRINIT, CHRSIZ, ERASE, and MPLOT to plot interesting patterns on the terminal screen. Subroutine ERASE also calls WAIT, and the subroutines also use several routines from the System Subroutine Library (such as ITTOUR, GTIM, and the Integer*4 functions).

Program operation proceeds as follows. First, graphics parameters are initialized by calling routine GRINIT. This routine asks about the type of terminal that is being used. Second, routine CHRSIZ is called to change the character size to #3 (next to the smallest) if a 4014-type terminal is being used. Third, the screen is erased, and the program asks the operator to supply four numbers which control the plot to be produced. The first value requested is N, which is the number of points. N should range between 10 and 30,000. The second value is the Shrinkage Factor (see below). This value should be between 0 and 2. The third number is the Angle Increment, and the fourth is the Line Type (normally 1, 97, 98, 99, 100, 104, or 112). The meaning of these parameters should be clear from the discussion following, and from observing the program's operation. The plot is drawn as follows. First, a point slightly to the right of the center of the terminal's screen is the middle of the plot. From this point, any other point can be defined by a vector with a particular radius and angle, using polar coordinates. The program selects a starting value for the radius, and a starting angle of 0 (relative to horizontal), so that the first point on the plot is to the right of the plot's center. The program then draws N lines, by changing the values for R and the angle each time. If the Shrinkage Factor is 0, then R is always the same (all points will lie on an imaginary circle). If the Shrinkage is 1, then the last point will be at the center and all other points will lie on an imaginary spiral. Other values for the Shrinkage may be used also. After plotting each point on the plot, the program changes the angle by the user-specified Angle Increment. If this value is very small, then a circle or spiral will be drawn. If the value is larger, then a polygon or star may be drawn. The process will continue until N lines have been drawn, and then the user may try a different combination of parameters to produce a new plot. Note that after R and the angle are computed for each point, they are transformed back into rectangular (X and Y) coordinates before the call to MPLOT. The various line types are defined in the section on routine MPLOT.
Refer there for more information. However, the various dotted and dashed lines (types 97-104) can only be produced on a Tektronix 4014 terminal with the enhanced graphics option.

Although this program does not do anything particularly useful, it does demonstrate the use of several graphics subroutines. In particular, it is an example of how MPLOT may be used in a special plotting situation, with little extra programming required. It also shows how a relatively simple program can generate rather intricate plots. The program listing and sample output follow.
PROGRAM DEMOGR

DEMONSTRATION PROGRAM FOR GRAPHICS CAPABILITY
OF TEKTRONIX TERMINALS

COMPILING/LINKING PROCEDURE:
FORT/WARN DEMOGR/LIST
LINK/HAP/DSK:/LIB:SY:FPULIB:SY:WGCLIB DEMOGR

NOTE: /LIB:SY:FPU SHOULD NOT BE USED WHEN YOU WILL
BE RUNNING THIS PROGRAM ON A REGULAR LSI-11
(USE IT ONLY FOR LSI-11/23'S)

RO = 1500.

DEFINE CENTER OF DISPLAY PATTERN (SLIGHTLY TO THE RIGHT
OF THE ACTUAL CENTER OF THE SCREEN)

IXCNTR = 2500
IYCNTR = 1550

INITIALIZE GRAPHICS PARAMETERS

CALL GRINIT(-1,0,0)
CALL CHRSIZ(3)
CALL ERASE

TYPE *, 'Enter N (10-30000), SHRINKAGE (0-2),'
TYPE *, 'ANGLE INC (1-360), and LINE'
TYPE *, 'TYPE (1,97,99,100,104,112)'
ACCEPT *,N,RFACTR,ANGINC,ITYPE

CHANGE ANGLE INC FROM DEGREES TO RADIANS
ANGINC = ANGINC * 3.141593 / 180.

MOVE TO RIGHT SIDE (STARTING POINT)

CALL MPlot(I XCNTR+IFIX(RO), IYCNTR, 0)

DO 50 K=1,N
DRAW LINE TO A POINT AT RADIUS SLIGHTLY LESS
THAN PREVIOUS VALUE (DETERMINED BY SHRINKAGE)
AND AT AN ANGLE OF ANGINC COUNTERCLOCKWISE
FROM THE PREVIOUS POINT

ANGLE = FLOAT(K) * ANGINC
R = RO *(1 - RFACTR * FLOAT(K)/FLOAT(N))
IX = R * COS(ANGLE) + IXCNTR
IY = R * SIN(ANGLE) + IYCNTR
CALL MPlot(IX, IY, ITYPE)

CONTINUE
CALL MPlot(0,300,-1)
PAUSE 'Hit Return Key'
CALL ERASE
GO TO 20
END
### FORTRAN IV Storage Map for Program Unit DEMOGR

**Local Variables, .PSECT $DATA, Size = 000050 (20 words)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGINC</td>
<td>R*4</td>
<td>000016</td>
<td>ANGLE</td>
<td>R*4</td>
<td>000026</td>
<td>ITYPE</td>
<td>I*2</td>
<td>000022</td>
</tr>
<tr>
<td>IX</td>
<td>I*2</td>
<td>000036</td>
<td>IXCNTR</td>
<td>I*2</td>
<td>000004</td>
<td>IY</td>
<td>I*2</td>
<td>000040</td>
</tr>
<tr>
<td>IYCNTR</td>
<td>I*2</td>
<td>000006</td>
<td>K</td>
<td>I*2</td>
<td>000024</td>
<td>N</td>
<td>I*2</td>
<td>000010</td>
</tr>
<tr>
<td>R</td>
<td>R*4</td>
<td>000032</td>
<td>RFACTR</td>
<td>R*4</td>
<td>000012</td>
<td>RO</td>
<td>R*4</td>
<td>000000</td>
</tr>
</tbody>
</table>

**Subroutines, Functions, Statement and Processor-Defined Functions:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHRSIZ</td>
<td>R*4</td>
<td>COS</td>
<td>R*4</td>
<td>ERASE</td>
<td>R*4</td>
<td>FLOAT</td>
<td>R*4</td>
<td>GRINIT</td>
<td>R*4</td>
</tr>
<tr>
<td>IFIX</td>
<td>I*2</td>
<td>MPLLOT</td>
<td>I*2</td>
<td>SIN</td>
<td>R*4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RUN DEMOGR

What is the model no. for the terminal you are using? (Don’t enter dash numbers. Example: If you have a 4014-1, just enter 4014) 74014

What is the model no. for the hard copy unit you are using? (If none is connected to your terminal, enter 0.) 74631

Does your terminal have the enhanced graphics option? (Can it draw dotted and dashed lines?) (Y=Yes) Y
The third sample program is GRTEST. This demonstrates a more realistic plotting situation, although dummy data was used for this example. The program uses GRINIT, CHRSIZ, GRID, ANOTAT, XYPILOT, MPLLOT, COPY, and BELL. Several other routines are also called by these. Refer to the descriptions of each subroutine for more details.

The operation of this program should be fairly clear from the program listing and the sample output. However, a few points are worth noting. First, when you use ANOTAT, you do not have to label every grid line. Every other vertical grid line was labelled in this example by specifying 10 horizontal segments for GRID, and 5 for ANOTAT. Second, when drawing a single line (such as the "50% of max." line here), two calls to MPLLOT are usually required. The first call, with the third argument set to 0, moves the current position to one end of the line (without drawing anything on the screen). The second call to MPLLOT, with the third argument set to a positive number, draws the line to the coordinates of the specified end point. In addition, when labelling the plot axes or performing other alphanumeric I/O, you should first call MPLLOT with the third argument set to -1, then use a formatted write (or type) statement, with a "+" in the first print position in order to disable carriage control on that line. Otherwise, the line may be printed at a location different from the one you specified.

Finally, the calls to routine BELL demonstrate how a program may provide some auditory feedback to the operator, perhaps to verify that certain data is acceptable or unacceptable (using different sounds), or to let a user who is away from the terminal know that some action is needed. The program listing and sample output follow.
PROGRAM GRTEST

C REASON: I TEST CERTAIN FEATURES OF TEKTRONIX
C GRAPHICS PACKAGE

C COMILING/LINKING PROCEDURE:
C FORT/WARN GRTEST/LIST
C LINK/MAK:DK:/LIB:SY:WGCLIB/LIB:SY:FPUS GRTEST

NOTE: THE /LIB:SY:FPUS SHOULD NOT BE USED UNLESS YOU
C WILL BE RUNNING THE PROGRAM ON AN LSI-11/23.
C DO NOT USE /LIB:SY:FPUS WITH A REGULAR LSI-11.

REAL X(30), Y(30)

DEFINE LIMITS OF PLOTTING AREA ON TERMINAL SCREEN
DATA IL, IR, IB, IT / 400, 3900, 1000, 2000 /

DUMMY X & Y VALUES TO PLOT
DATA X / 0.005, 0.012, 0.019, 0.028, 0.032, 0.040, 0.043,
0.050, 0.055, 0.064, 0.076, 0.081, 0.095, 0.100, 0.104,
0.109, 0.116, 0.122, 0.125, 0.134, 0.143, 0.151, 0.158,
0.166, 0.170, 0.175, 0.183, 0.190, 0.195 /

DATA Y / 0.0, 1.0, 1.15, 2.5, 4.5, 42.0, 83.0, 135.0, 120.0, 178.0, 185.0,
205.0, 197.0, 210.0, 225.0, 222.0, 265.0, 308.0, 332.0, 322.0, 345.0,
387.0, 460.0, 405.0, 418.0, 382.0, 360.0, 375.0, 347.0, 357.0 /

INITIALIZE PARAMETERS FOR GRAPHICS ROUTINES (GET INFO.
C FROM USER)
CALL GRINIT(-1.0, 0.0)

SET CHARACTER SIZE TO #3 (NEXT TO SMALLEST)
CALL CHRSIZ(3)

DRAW HORIZ. & VERT. GRID LINES
CALL GRID (10*5, IL, IR, IB, IT, 97)

LABEL (ANDOTATE) GRID LINES WITH NUMERICAL USER UNITS
CALL ANDOTAT (5*5, IL, IR, IB, IT, 0.0, 0.2, 0.0, 500.0)

PLOT THE DATA (CONNECT POINTS WITH STRAIGHT LINES)
CALL XYPLOT (X, Y, 30, IL, IR, IB, IT, 0.0, 0.2, 0.0, 500.0, 1)

GET MAX. Y VALUE & DRAW HORIZ. L INE ON PLOT
AT LEVEL CORRESPONDING TO HALF THAT VALUE
YMAX = 0.
DO 50 K = 1, 30
   IF (YMAX .LE. Y(K)) YMAX = Y(K)
10 CONTINUE

IIHALF = (0.5*YMAX) / 500.0 * (IT-IB) + IB
MOVE TO LEFT SIDE OF PLOT AT PROPER Y DISTANCE UP
CALL MPLT (IL, IIHALF, 0)

DRAW HORIZ. L INE TO RIGHT SIDE OF PLOT
CALL MPLOT (IR,IYHALF,99)

MOVE TO POSITION JUST ABOVE THE LINE WE Drew

CALL MPLOT (IX,IYHALF+10,-1)

LABEL THE LINE

TYPE 60

FORMAT ('+50% of max.')

NOTE THAT YOU MUST USE A + TO DISABLE CARRIAGE CONTROL
IN THE FORMAT. OTHERWISE, A LINE FEED WOULD BE SENT TO
THE TERMINAL BEFORE TYPING THE LINE.

CALL MPLOT (IL+1350,1B-200,-1)

TYPE 70

FORMAT ('+STIMULUS DURATION (msec.)')

CHANGE TO CHARACTER SIZE 82 (SECOND LARGEST)

CALL CHRSIZ (2)

CALL MPLOT (IL+600,IT+30,-1)

TYPE 80

FORMAT ('+RESPONSE AMPLITUDE VS. STIMULUS DURATION')

MOVE TO BELOW BOTTOM OF PLOT

CALL MPLOT (0,1B-300,-1)

MAKE A HARD COPY OF THE TERMINAL SCREEN

CALL COPY (1)

MAKE SOME NOISES TO ALERT THE OPERATOR IN CASE
HE/SHE WENT TO SLEEP.

CALL BELL (20,8)

CALL BELL (200,1)

CALL BELL (5,60)

STOP

END
FORTRAN IV Storage Map for Program Unit GRTEST

Local Variables, .PSECT $DATA, Size = 000420 (136 words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IB</td>
<td>I*2</td>
<td>000364</td>
<td>IL</td>
<td>I*2</td>
<td>000360</td>
<td>IR</td>
<td>I*2</td>
<td>000362</td>
</tr>
<tr>
<td>IT</td>
<td>I*2</td>
<td>000366</td>
<td>IX</td>
<td>I*2</td>
<td>000402</td>
<td>IYHALF</td>
<td>I*2</td>
<td>000400</td>
</tr>
<tr>
<td>K</td>
<td>I*2</td>
<td>000376</td>
<td>YMAX</td>
<td>R*4</td>
<td>000372</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local and COMMON Arrays:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>Size</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>R*4</td>
<td>$DATA</td>
<td>000000</td>
<td>000170 (60) (30)</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>R*4</td>
<td>$DATA</td>
<td>000170</td>
<td>000170 (60) (30)</td>
<td></td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANOTAT</td>
<td>R*4</td>
<td>BELL</td>
<td>R*4</td>
<td>CHRSIZ</td>
<td>R*4</td>
<td>COPY</td>
<td>R*4</td>
</tr>
<tr>
<td>GRINIT</td>
<td>R*4</td>
<td>MPLOT</td>
<td>I*2</td>
<td>XYPLOT</td>
<td>R*4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RUN GRTES
What is the model no. for the terminal you are using? (Don't enter
dash numbers. Example: If you have a 4014-1, just enter 4014) 74014
What is the model no. for the hard copy unit you are using?
(If none is connected to your terminal, enter 0.) 74631
Does your terminal have the enhanced graphics option?
(Can it draw dotted and dashed lines?) (Y=Yes) Y

RESPONSE AMPLITUDE VS. STIMULUS DURATION

<table>
<thead>
<tr>
<th>RESPONSE AMPLITUDE</th>
<th>STIMULUS DURATION (msec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.000</td>
</tr>
<tr>
<td>400</td>
<td>0.040</td>
</tr>
<tr>
<td>300</td>
<td>0.080</td>
</tr>
<tr>
<td>200</td>
<td>0.120</td>
</tr>
<tr>
<td>100</td>
<td>0.160</td>
</tr>
<tr>
<td></td>
<td>0.200</td>
</tr>
</tbody>
</table>
The fourth sample program is DISKRW. This routine demonstrates the use of subroutine DISKIO to read and write data to disk files. Refer to the program listing and sample output below. Note that this program is written specifically for use with the DEC RT-11 operating system, and is probably not adaptable to other systems. DISKIO calls a number of RT-11-specific subroutines, as indicated in its description, later in this document. DISKIO was written primarily to facilitate binary input and output with variable record length to random (not necessarily sequential) blocks in a disk file, using a Fortran main program. The normal Fortran I/O normally requires that direct access binary (unformatted) records all be of the same length. Records in sequential files may be of variable length, but are inconvenient and inefficient to access in a random-access fashion.

In general, the normal DEC Fortran I/O methods, using an OPEN statement followed by READ or WRITE statements, are preferrable if their restrictions are not a problem. Otherwise, you may use subroutine DISKIO. However, records which have been written with DISKIO may only be read using the same routine, and not with the usual Fortran I/O methods. Basically, DISKIO allows more efficient I/O than Fortran in many cases, both in terms of time and also storage space, but its files are not compatible with those produced by the usual Fortran I/O routines.

Operation of program DISKRW and DISKIO should be more clear after referring to the program listing, sample output, and output file dumps which follow.
PROGRAM DISKRW

* PROGRAM TO VERIFY OPERATION & DEMONSTRATE USE OF SUBROUTINE DISKIO
* FOR WRITING DATA TO & READING IT FROM DISK FILES.
* SEE COMMENTS IN DISKIO FOR MORE INFO.

* WRITTEN BY: WILLIAM G. CROSIER
* DATE: 11 JUNE 1980

* COMPILED/LINKING SEQUENCE:
* FORT/NOSWAP DISKRW/LIST
* FORT DISKIO/LIST
* LINK/MAP: DK: DISKRW,DISKIO

INTEGER BUFR1(1024),BUFR2(1024),IERR
BYTE FILNAM(12)
DO 20 K=1,12
FILNAM(K)=0

10 TYPE 30
30 FORMAT ('ENTER COMPLETE FILE NAME FOR DISK I/O IN THE FORMAT',
@' 'DEV'FILNAME'EXT',' WHERE: 'DEV' IS THE 3-CHARACTER DEVICE CODE'/
@' 'FILNAME' IS THE 6-CHARACTER BASIC FILE NAME'/
@' 'EXT' IS THE 3-CHARACTER EXTENSION/FILE TYPE DESIRED (OPT.)'/
@' EACH PORTION OF THE NAME SHOULD BE THE EXACT LENGTH SPECIFIED'/
@' WITH SPACES ADDED, IF NECESSARY, TO FORM THE PROPER LENGTH'/
@' DO NOT USE A COLON OR PERIOD TO SEPARATE PORTIONS OF THE FILE',
@' NAME. ' EXAMPLE: 'DK A12345DAT' /' FILE NAME ? 'Y*$)
ACCEPT 409 FILNAM

40 TYPE *,'ENTER SIZE OF FILE (IN NO. OF 256-WORD BLOCKS)'
ACCEPT *,NBLK

45 TYPE *, 'ENTER NO. OF WORDS TO WRITE & READ (1-1024)'
ACCEPT *,NWRDS

5 TYPE *, 'ENTER BLOCK NO. OF FILE WHERE I/O IS TO START'
6 TYPE *, '(0=START AT 1ST BLOCK, 1=START AT 2ND, ETC.)'
7 ACCEPT *,IBLK

8 TYPE *, 'LEAVE FILE OPEN AFTER I/O (1=YES) ? '
9 ACCEPT *,LOPEN

IMODE = -1
DO 50 K=1,NWRDS
BUFR1(K)=K

50 TYPE*,'NOW WRITING DATA TO DISK'
CALL DISKIO(FILNAM,IMODE,BUFR1,NWRDS,IBLK,NBLK,IERR)

IF (IERR.NE.0) TYPE*, 'ERROR CODE',IERR,'DURING WRITE'
C FOR EXPLANATION OF ERROR CODES, SEE COMMENTS IN DISKIO
IMODE = 3
IF (LOPEN .EQ. 1) IMODE = -3

TYPE *, 'NOW READING DATA FROM DISK'
CALL DISKIO(FILNAM,IMODE,BUFR2,NWRDS,IBLK,NBLK,N Dummy,IERR)

IF (IERR.NE.0) TYPE *, 'ERROR CODE',IERR,' DURING READ'

TYPE 60, NWRDS:(BUFR2(K)+K=1,NWRDS)

FORMAT ('DATA READ FROM DISK SHOULD BE CONSECUTIVE INTEGERS'
@' FROM 1 THROUGH',IS .. ');'/ (10I7))
FORTRAN IV  V02.1-1 Thu 01-Oct-81 15:12:58 PAGE 002

0036 IF (LOPEN .EQ. 1) GO TO 45
0038 TYPE **MORE (1=YES) ? ' 
0039 ACCEPT **MORE
0040 IF (MORE .EQ. 1) GO TO 10
0042 STOP
0043 END

FORTRAN IV  Storage Map for Program Unit DISKRW

Local Variables, .PSECT $DATA, Size = 010050 ( 2068. words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBLK</td>
<td>I*2</td>
<td>010034</td>
<td>IERR</td>
<td>I*2</td>
<td>010024</td>
<td>IRMODE</td>
<td>I*2</td>
<td>010042</td>
</tr>
<tr>
<td>IWMODE</td>
<td>I*2</td>
<td>010040</td>
<td>K</td>
<td>I*2</td>
<td>010026</td>
<td>LOPEN</td>
<td>I*2</td>
<td>010036</td>
</tr>
<tr>
<td>MORE</td>
<td>I*2</td>
<td>010046</td>
<td>NBLK</td>
<td>I*2</td>
<td>010030</td>
<td>NDUMMY</td>
<td>I*2</td>
<td>010044</td>
</tr>
<tr>
<td>NWRDS</td>
<td>I*2</td>
<td>010032</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local and COMMON Arrays:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>----Size----</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFR1</td>
<td>I*2</td>
<td>$DATA</td>
<td>000000</td>
<td>004000 (1024.) (1024)</td>
<td></td>
</tr>
<tr>
<td>BUFR2</td>
<td>I*2</td>
<td>$DATA</td>
<td>004000</td>
<td>004000 (1024.) (1024)</td>
<td></td>
</tr>
<tr>
<td>FILNAM</td>
<td>L*1</td>
<td>$DATA</td>
<td>010000</td>
<td>000014 (6.) (12)</td>
<td></td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USKIO</td>
<td>R*4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RUN DISK

ENTER COMPLETE FILE NAME FOR DISK I/O IN THE FORMAT "DEMFILNEXX".
WHERE "DEV" IS THE 3-CHARACTER DEVICE CODE
"FILNX" IS THE 6-CHARACTER BASIC FILE NAME
"EXT" IS THE 3-CHARACTER EXTENSION/FILE TYPE DESIRED (OPT.)
EACH PORTION OF THE NAME SHOULD BE THE EXACT LENGTH SPECIFIED.
WITH SPACES ADDED, IF NECESSARY, TO FORM THE PROPER LENGTH.
DO NOT USE A COLON OR PERIOD TO SEPARATE PORTIONS OF THE FILE NAME.
EXAMPLE: DK A1234SDAT
FILE NAME: DV1A0002DAT
ENTER SIZE OF FILE (IN NO. OF 256-WORD BLOCKS)
2
ENTER NO. OF WORDS TO WRITE & READ (1-1024)
32
ENTER BLOCK NO. OF FILE WHERE I/O IS TO START
(0=START AT 1ST BLOCK, 1=START AT 2ND, ETC.)

LEAVE FILE OPEN AFTER I/O (1=YES) ?
1
NOW WRITING DATA TO DISK
NOW READING DATA FROM DISK

DATA READ FROM DISK (SHOULD BE CONSECUTIVE INTEGERS
FROM 1 THROUGH 32):

DATA READ FROM DISK (SHOULD BE CONSECUTIVE INTEGERS
FROM 1 THROUGH 48):

MORE (1=YES) ?
0
STOP --
.DUMP/NOASCII DV1·A00002 DAT
### Di1:00002.DAT/N

<table>
<thead>
<tr>
<th>BLOCK NUMBER</th>
<th>000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>010/</td>
<td>000001 000002 000003 000004 000005 000006 000007 000008</td>
</tr>
<tr>
<td>020/</td>
<td>000011 000012 000013 000014 000015 000016 000017 000018</td>
</tr>
<tr>
<td>030/</td>
<td>000021 000022 000023 000024 000025 000026 000027 000028</td>
</tr>
<tr>
<td>050/</td>
<td>000031 000032 000033 000034 000035 000036 000037 000038</td>
</tr>
<tr>
<td>090/</td>
<td>000041 000042 000043 000044 000045 000046 000047 000048</td>
</tr>
<tr>
<td>100/</td>
<td>000051 000052 000053 000054 000055 000056 000057 000058</td>
</tr>
<tr>
<td>140/</td>
<td>000061 000062 000063 000064 000065 000066 000067 000068</td>
</tr>
<tr>
<td>160/</td>
<td>000071 000072 000073 000074 000075 000076 000077 000078</td>
</tr>
<tr>
<td>200/</td>
<td>000081 000082 000083 000084 000085 000086 000087 000088</td>
</tr>
<tr>
<td>220/</td>
<td>000091 000092 000093 000094 000095 000096 000097 000098</td>
</tr>
<tr>
<td>240/</td>
<td>000101 000102 000103 000104 000105 000106 000107 000108</td>
</tr>
<tr>
<td>260/</td>
<td>000111 000112 000113 000114 000115 000116 000117 000118</td>
</tr>
<tr>
<td>300/</td>
<td>000121 000122 000123 000124 000125 000126 000127 000128</td>
</tr>
<tr>
<td>320/</td>
<td>000131 000132 000133 000134 000135 000136 000137 000138</td>
</tr>
<tr>
<td>340/</td>
<td>000141 000142 000143 000144 000145 000146 000147 000148</td>
</tr>
<tr>
<td>360/</td>
<td>000151 000152 000153 000154 000155 000156 000157 000158</td>
</tr>
<tr>
<td>400/</td>
<td>000161 000162 000163 000164 000165 000166 000167 000168</td>
</tr>
<tr>
<td>420/</td>
<td>000171 000172 000173 000174 000175 000176 000177 000178</td>
</tr>
<tr>
<td>440/</td>
<td>000181 000182 000183 000184 000185 000186 000187 000188</td>
</tr>
<tr>
<td>460/</td>
<td>000191 000192 000193 000194 000195 000196 000197 000198</td>
</tr>
<tr>
<td>500/</td>
<td>000201 000202 000203 000204 000205 000206 000207 000208</td>
</tr>
<tr>
<td>520/</td>
<td>000211 000212 000213 000214 000215 000216 000217 000218</td>
</tr>
<tr>
<td>540/</td>
<td>000221 000222 000223 000224 000225 000226 000227 000228</td>
</tr>
<tr>
<td>560/</td>
<td>000231 000232 000233 000234 000235 000236 000237 000238</td>
</tr>
<tr>
<td>600/</td>
<td>000241 000242 000243 000244 000245 000246 000247 000248</td>
</tr>
<tr>
<td>620/</td>
<td>000251 000252 000253 000254 000255 000256 000257 000258</td>
</tr>
<tr>
<td>640/</td>
<td>000261 000262 000263 000264 000265 000266 000267 000268</td>
</tr>
<tr>
<td>660/</td>
<td>000271 000272 000273 000274 000275 000276 000277 000278</td>
</tr>
<tr>
<td>700/</td>
<td>000281 000282 000283 000284 000285 000286 000287 000288</td>
</tr>
<tr>
<td>720/</td>
<td>000291 000292 000293 000294 000295 000296 000297 000298</td>
</tr>
<tr>
<td>740/</td>
<td>000301 000302 000303 000304 000305 000306 000307 000308</td>
</tr>
<tr>
<td>760/</td>
<td>000311 000312 000313 000314 000315 000316 000317 000318</td>
</tr>
</tbody>
</table>

27
GRAPHICS
SUBROUTINES
Subroutine GRINIT

This routine initializes the various parameters used by other graphics routines, and should be called before any of the other graphics routines. It is not needed if only the data acquisition routines ISAMPA, DAQISR, DAQ or miscellaneous routines BELL, WAIT, or DISKIO are used.

Information on the parameters or arguments to be passed to GRINIT appears in the program listing below. When writing a particular program, if you will always be using it with the same type of terminal and hard copy unit, then you may specify their model numbers in the argument list so that GRINIT may define its internal parameters. However, if you may use different types of terminals, then you should set the first argument to -1 so that GRINIT will query the user at execution time about the type of terminal and hard copy unit to be used. The program listing follows.
SUBROUTINE GRINIT(ITERM, IHDCP, IENH)

C
C WRITTEN BY: WILLIAM G. CROSIER
C REVISED: 28 SEPT. 1981
C
C PURPOSE: INITIALIZE PARAMETERS DEFINING TERMINAL
CHARACTERISTICS FOR THE VARIOUS GRAPHICS ROUTINES
C ARGUMENTS:
C ITERM = MODEL NUMBER OF TERMINAL BEING USED.
C IF =-1, THEN USER WILL BE QUERIED DURING PROGRAM
EXECUTION ABOUT TERMINAL & HARD COPY UNIT.
C IHDCP = MODEL NUMBER OF HARD COPY UNIT BEING USED
C SET IHDCP=0 IF NO HARD COPY UNIT IS BEING USED
C IENH = FLAG FOR ENHANCED GRAPHICS OPTION FOR TERMINAL.
C SET IENH=1 IF YOU HAVE A 4014/4015 WITH THIS OPTION.
C OTHERWISE, SET IENH=0
C
C NOTE: IHDCP & IENH ARE BOTH IGNORED IF ITERM=-1. IN THIS CASE,
THEN THE USER WILL BE ASKED TO SUPPLY THE APPROPRIATE
INFO. FOR THE TERMINAL WHEN THIS ROUTINE IS CALLED.
C
0002 LOGICAL*1 ANSWER
0003 COMMON /GRFCOM/ MCHRSZ,LSIZE,IWIDTH,IHEIGHT,IENHAN,
@ TIMERA,TIMHDC
C
0004 ITERM1 = ITERM
0005 IHDCP1 = IHDCP
0006 IENH1 = IENH
0007 IF (ITERM1 .GT. 0) GO TO 60
C
C QUERY USER ABOUT TYPE OF TERMINAL & HARD COPY UNIT BEING USED
0009 TYPE 20
0010 20 FORMAT (' What is the model no. for the terminal you are '
@ 'using? (Don’t enter / dash numbers. Example '
@ 'If you have a 4014-1, just enter 4014) ?',@)
0011 ACCEPT *,ITERM1
0012 TYPE 30
0013 30 FORMAT (' What is the model no. for the hard copy unit'
@ 'you are using? (If none is connected to'
@ 'your terminal, enter 0.) ?',@)
0014 ACCEPT *, IHDCP1
0015 IF (ITERM1.NE.4014 .AND. ITERM1.NE.4015) GO TO 60
0017 TYPE 40
0018 40 FORMAT (' Does your terminal have the enhanced graphics'
@ 'option?' / ' (Can it draw dotted and dashed',
@ 'lines?) (Y=Yes/ N?)
0019 ACCEPT 50, ANSWER
0020 50 FORMAT (A1)
0021 IENH1 = 0
0022 IF (ANSWER .EQ. ‘Y’) IENH1 = 1
C
0024 60 MCHRSZ = 1
C IF USING A 4014/4015 TERMINAL, MULTIPLE CHARACTER SIZES ARE AVAIL.
0025 IF (ITERM1.NE.4014 .AND. ITERM1.NE.4015) MCHRSZ = 0

C SET TIME TO ALLOW FOR ERASING SCREEN = 1.5 SEC.
0027 TIMERA = 1.5

C SET DEFAULT HARD COPY TIME = 20 SEC.
0028 TIMHDC = 20.0

0029 IF (IHDCP1 .LE. 0) TIMHDC=0 !NO HARD COPY UNIT AVAIL.
0031 IF (IHDCP1 .EQ. 4631) TIMHDC=10.0 !10 SEC FOR MODEL 4631

0033 IENHAN = IENH1
0034 RETURN
0035 END

C

FORTRAN IV Storage Map for Program Unit GRINIT

Local Variables, .PSECT $DATA, Size = 000016 ( 7. words)

Name Type Offset Name Type Offset Name Type Offset
ANSWER L*1 000006 IENH I*2 @ 000004 IENH1 I*2 000014
IHDCP I*2 @ 000002 IHDCP1 I*2 000012 ITERM I*2 @ 000000
ITERM1 I*2 000010

COMMON Block /GRFCOM/, Size = 000022 ( 9. words)

Name Type Offset Name Type Offset Name Type Offset
MCHRSZ I*2 000000 LSIZE I*2 000002 IWIDTH I*2 000004
IMIGHT I*2 000006 IENHAN I*2 000010 TIMEKA R*4 000012
TIMHDC R*4 000016
Subroutine MPLOT

This is the basic line-drawing subroutine. It is used by all of the other graphics routines which draw lines or move the cursor, and it may be called directly by the user's program also.

The arguments or parameters for this routine are described in the program listing below. Refer there for more information. In addition, there are further details, concerning the various line types available in a 4014 with enhanced graphics, in the Tektronix manuals for the 4014 terminal. If you are not using a 4014 or 4015, then GRINIT will cause all requests for line types other than 1 (normal solid line) to be responded to as if line type 1 was specified. In this way, you do not need to worry about the presence of the optional enhanced graphics hardware in your terminal when you write the software, because if the hardware is not present then solid lines will be drawn regardless of the line type you specify with parameter IPEN.

Besides drawing a line, MPLOT can also be used to move the cursor immediately before plotting by setting parameter IPEN to 0. Also, it can be used to move the cursor before typing alphanumeric data by setting IPEN to -1. Refer to the third sample main program (GRTEST) in an earlier section of this report for examples of this. The second sample program (DEMOGR) may also be helpful when using MPLOT.

If you are not using a 4014/4015 terminal with enhanced graphics, then you will have only 1024 point horizontal resolution horizontally and 780 point resolution vertically, but the same coordinates will still refer to the same point on the screen. Thus, an X coordinate of 2048 and a Y coordinate of 1560 will always refer to the center of the screen, regardless of the type of Tektronix terminal you are using.

If you want to modify this routine for use with another type of terminal or with an X-Y plotter, then several parts of the subroutine will need to be changed. First, Tektronix requires that the X and Y coordinates be split up and sent in a particular sequence. This sequence, under the heading "Draw Vector" in the program listing, would probably need to be changed for a
different terminal or plotter. Second, many plotters require a delay after transmitting the coordinates for a new point so that the pen has time to move. For this purpose, you can use a call to subroutine WAIT. If an analog X-Y recorder was to be used rather than a digital plotter, then the X and Y coordinates would need to be sent to the plotter via two IPOKE calls (refer to the System Subroutine Library routines) to a pair of digital-to-analog converters. The outputs of these may have to be slowed down with a pair of matched R-C networks in order to avoid too-rapid changes in the pen position, if the recorder so requires. Using this routine with a digital plotter should be much simpler, however. Many digital plotters also have built-in character generators for drawing the standard ASCII character set also, while analog X-Y recorders do not have this capability.

Note that when using a line type of 112 (for the write-thru mode), the lines which are drawn are faint, and do not store on the CRT screen. Normally, when using this line type you should refresh the display by repeating the plotting of the appropriate lines. Ideally, the lines should be redrawn at least 50 times per second in order to prevent flickering and to make the display easier to see. However, slower refresh rates may be necessary if a large number of lines have to be redrawn with each repetition. In addition, static displays (with line types other than 112) may be combined with dynamic displays (line Type 112) at the same time. That way you only have to refresh the lines whose positions change.

One problem when using MPLOT (and any routines which call it) has appeared in some systems when using the RT-11 Version 4 Foreground/Background monitor. The problem appears to be in the RT-11 terminal handler (TT.SYS) or in the System Subroutine Library routine ITTOUR, since the difficulty has never appeared with the RT-11 Single Job monitor. If your plots do not come out correctly but appear garbled, try running your program under the Single Job monitor (RT11SJ) rather than the Foreground/Background one (RT11FB).

MPLOT requires the following routine from the System Subroutine Library:

ITTOUR

A listing of MPLOT follows.
SUBROUTINE MPLT(IX, IY, IPEN)

AUTHOR: WILLIAM G. CROSIER TECHNOLOGY INCORPORATED
REVISED 25 SEPT. 1981
BASED ON ROUTINE WRITTEN BY CHUCK MANN

PURPOSE: TO DRAW LIGHT OR DARK VECTORS FROM THE CURSOR'S (PEN'S) PRESENT LOCATION TO THE COORDINATES PASSED.

ARGUMENTS:
IX = X COORDINATE IN TEKTRONIX SCREEN UNITS (0 TO 4095)
IY = Y COORDINATE IN TEKTRONIX SCREEN UNITS (0 TO 3120)
IPEN CONTROLS PLOTTING AS FOLLOWS:
IPEN = 0 MOVES POSITION TO (IX, IY) WITHOUT DRAWING A LINE (DARK VECTOR)
IPEN < 0 MOVES POSITION & CHANGES TO ALPHA MODE
IPEN > 0 DRAWS A VISIBLE LINE TO (IX, IY)
IF IPEN = 1, THEN NORMAL SOLID LINE
IF IPEN = 97, THEN DOTTED LINE (ON 4014)
   = 98, THEN DOT-DASH LINE
   = 99, THEN SHORT DASH LINE
   = 100, THEN LONG DASH LINE
   = 104, THEN DEFOCUSED Z-AXIS (SLIGHTLY WIDER LINE)
   = 112, THEN WRITE-THRU (NON-STORE)
(THE VARIOUS DOTTED & DASHED LINES WILL ONLY BE PRODUCED ON A 4014 TERMINAL WITH ENHANCED GRAPHICS OPTION.)

COMMON /GRFCOM/ MCHRSZ, LSIZE, IWIDTH, IHEIGHT, IENHAN, @ TIMERA, TIMHDC
DATA LSIZE, IWIDTH, IHEIGHT, IENHAN, TIMERA, TIMHDC / 156, 86, 0, 10, 0, 1.5 /
DATA MINX, MAXX, MINY, MAXY / 0, 4095, 0, 3120 /
DATA IPEN1 / 0 /

IX1 = IX
IY1 = IY
IF (IX .LT. MINX) IX1 = MINX
IF (IX .GT. MAXX) IX1 = MAXX
IF (IY .LT. MINY) IY1 = MINY
IF (IY .GT. MAXY) IY1 = MAXY
IF (IPEN .GT. 0) GO TO 10

DRAW (DARK VECTOR (MOVE POSITION))
IF (ITTOUR('35), NE, 0) GO TO 100 !SEND GS CHAR.
IF (IPEN .LE.) OR (IENHAN .EQ. 0) OR (IPEN .EQ. IPEN1)
   GO TO 30
C CHANGE TO SELECTED PLOTTING MODE IF USING TERMINAL WITH
C ENHANCED GRAPHICS

0022 LINTYP = IPEN
0023 IF (IPEN .LT. 96) LINTYP = 96 'SOLID LINE?'         
0025 20 IF (ITTOUR(27).NE.0) GO TO 20 'SEND ESC
0027 25 IF (ITTOUR(LINTYP).NE.0) GO TO 25 'SET LINE TYPE
0029 30 CONTINUE
0030 IPEN1 = IPEN 'RESET PREV. VALUE

C DRAW VECTOR

C SEND HIGH ORDER Y BYTE
ICH='40+IY1/128
0032 105 IF (ITTOUR(ICH).NE.0) GO TO 105
C SEND EXTRA BYTE (2 LSB'S OF X & Y)
0034 116 ICH='40 +((IX1.AND.'3)*4) + (IX1.AND.'3)
0035 108 IF (ITTOUR(ICH).NE.0) GO TO 108
C SEND LOW ORDER Y BYTE
0037 110 ICH='140+((IY1/4).AND.'37)
0038 110 IF (ITTOUR(ICH).NE.0) GO TO 110
C SEND HIGH ORDER X BYTE
0040 115 ICH='40+IX1/128
0041 115 IF (ITTOUR(ICH).NE.0) GO TO 115
C SEND LOW ORDER X BYTE
0043 120 ICH='100+((IX1/4).AND.'37)
0044 120 IF (ITTOUR(ICH).NE.0) GO TO 120
0046 IF (IPEN .GE. 0) GO TO 900 'RESET PREV. VALUE
C
C CHANGE TO ALPHA MODE IF IPEN IS LESS THAN 0
0048 200 IF (ITTOUR(*37).NE.0) GO TO 200
0050 900 RETURN

FORTRAN IV Storage Map for Program Unit MPLOT

Local Variables, .PSECT @DATA; Size = 000030 ( 12, words)

Name Type Offset Name Type Offset Name Type Offset
ICH I*2 000026 IPEN I*2 @ 000004 IPEN1 I*2 000016
IX I*2 @ 000000 IX1 I*2 000020 IY I*2 @ 000002
IY1 I*2 000022 LINTYP I*2 000024 MAXX I*2 000010
MAXY I*2 000014 MINX I*2 000006 MINY I*2 000012

COMMON Block /GRFCOM/, Size = 000022 ( 9, words)

Name Type Offset Name Type Offset Name Type Offset
CHRSZ I*2 000000 LSIZE I*2 000002 IWIDTH I*2 000004
HIGHT I*2 000006 IENHAN I*2 000010 TIMERA R*4 000012
MHDC R*4 000010

Subroutines, Functions, Statement and Processor-Defined Functions:

Name Type Name Type Name Type Name Type Name Type
ITTOUR I*2
Subroutine COPY

This short routine causes a hard copy unit (if one is present) to make a paper copy of whatever appears on the Tektronix terminal screen. After waiting for a fixed length of time, or until the user strikes the Return key (depending on the parameter IFLAG), the routine returns to the calling program. More details are given in the program listing.

COPY requires the following subroutines:

ERASE
WAIT
ITTOUR (from the System Subroutine Library)

A listing of routine COPY follows.
SUBROUTINE COPY(IFLAG)

PURPOSE: TO MAKE A HARD COPY OF THE TEKTRONIX SCREEN.
      IF: IFLAG=0, WAIT FOR USER TO TYPE <CR>, THEN
          ERASE SCREEN.
          =1, RETURN AFTER MAKING COPY (DON'T
          WAIT FOR RESPONSE, DON'T ERASE)
          =2, DON'T WAIT FOR RESPONSE AFTER MAKING
          COPY, BUT ERASE SCREEN.

COMMON /GRFCOM/, MCHRSZ,LSIZE,IWIDTH,IHEIGHT,IENHAN,
                   @ TIMERA,TIMHDC

MAKE HARD COPY OF THE SCREEN IF HARD COPY UNIT IS AVAIL.

IF (TIMHDC .LE. 0) GO TO 50
0005 230 IF (ITTOUT(27).NE.0) GO TO 230 !SEND ESC
0007 235 IF (ITTOUT(23).NE.0) GO TO 275 !SEND ETB

WAIT FOR SCREEN TO COPY
CALL WAIT (TIMHDC,0) !WAIT TIMHDC SEC.

WAIT FOR USER TO RESPOND
PAUSE 'HIT RETURN KEY TO CONTINUE'

CLEAR THE SCREEN
CALL ERASE
RETURN
END

FORTRAN IV Storage Map for Program Unit COPY

Local Variables, .FSCE $DATA, Size = 000002 ( 1, words)

Name Type Offset Name Type Offset Name Type Offset
IFLAG I*2 @ 000000

COMMON Block /GRFCOM/, Size = 000022 ( 9, words)

Name Type Offset Name Type Offset Name Type Offset
MCHRSZ I*2 000000 LSIZE I*2 000002 IWIDTH I*2 000004
IHEIGHT I*2 000006 IENHAN I*2 000010 TIMERA R*4 000012
TIMHDC R*4 000016

Subroutines, Functions, Statement and Processor-Defined Functions:

Name Type Name Type Name Type Name Type Name Type
ERASE R*4 ITTOUT I*2 WAIT R*4
Subroutine ERASE

This simple routine is used to erase or clear the Tektronix terminal screen. It sends an escape followed by a form feed character, waits a short time for the erase process to finish, then returns to the calling program.

ERASE requires the following subroutines:

WAIT

ITTOUR (from the System Subroutine Library)

A listing of routine ERASE follows.
SUBROUTINE ERASE

ERASE (CLEAR) THE SCREEN ON THE TEKTRONIX TERMINAL

COMMON /GRFCOM/ MCHRSZ, LSIZE, IWIDTH, IHIGHT, IENHAN, TIMERA, TIMHDC

IF (ITTOUR(27) .NE. 0) GO TO 10 !SEND ESC
IF (ITTOUR(12) .NE. 0) GO TO 20 !SEND FF
CALL WAIT(TIMERA,0) !WAIT TIMERA SEC.

RETURN

END

FORTRAN IV Storage Map for Program Unit ERASE

COMMON Block /GRFCOM/, Size = 000022 ( 9, words)

Name Type Offset Name Type Offset Name Type Offset
MCHRSZ I*2 000000 LSIZE I*2 000002 IWIDTH I*2 000004
IHIGHT I*2 000006 IENHAN I*2 000010 TIMERA R*4 000012
TIMHDC R*4 000016

Subroutines, Functions, Statement and Processor-Defined Functions:

Name Type Name Type Name Type Name Type Name Type
ITTOUR I*2 WAIT R*4
Subroutine CHRSIZ

This subroutine can be used to set the character size for those Tektronix terminals with multiple sizes available (the 4014/4015). This routine should be called before the other graphics routines which print alphanumeric characters, such as ANOTAT (or XYPlot with ICODE equal to 0). It can also be used before regular type statements in order to change to smaller or larger characters for page headings, etc. Once CHRSIZ is called, the new character size stays in effect until CHRSIZ is called again with a different size specified, even if the screen is erased or if you change between alpha and graph modes. The parameter/argument ISIZE sets the character size as described in the program listing.

CHRSIZ requires the following subroutines from the System Subroutine Library:

ITTCUR

A listing of CHRSIZ follows.
SUBROUTINE CHRSIZ(ISIZE)

Routine to set character size on Tektronix 4014 screen

Argument ISIZE controls character size as follows:
- ISIZE = 1 selects largest characters
- ISIZE = 2 selects medium-large char.
- ISIZE = 3 selects medium-small char.
- ISIZE = 4 selects smallest char.

LOGICAL*1 ICODE(4)
INTEGER ISIZE,IW(4),IH(4)
COMMON /GRFCOM/ MCHRSZ,LSIZE,IWIDTH,IHEIGHT,IENHAN,
@ TIMERA,TIMHDC

DATA MCHRSZ / 1 /
DATA ICODE / '8','9','1','2' /
DATA IW / 56,51,34,31 /
DATA IH / 88,82,53,48 /

IF (ISIZE.LT.1 .OR. ISIZE.GT.4 .OR. MCHRSZ.EQ.0) GO TO 99
IF (ITTOUR(27).NE.0) GO TO 20 !Send escape
IF (ITTOUR(ICODE(ISIZE)).NE.0) GO TO 30 !Send char.
IWIDTH = IW(ISIZE) !Width of a char.
IHEIGHT = IH(ISIZE) !Height of a char.
LSIZE = ISIZE

RETURN

END

FORTRAN IV Storage Map for Program Unit CHRSIZ

Local Variables: .PSECT $DATA, Size = 000026 ( 11. words)
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISIZE</td>
<td>I*2</td>
<td>@ 000000</td>
<td>ISIZE</td>
<td>I*2</td>
<td>@ 000000</td>
<td>ISIZE</td>
<td>I*2</td>
<td>@ 000000</td>
</tr>
</tbody>
</table>

COMMON Block /GRFCOM/, Size = 000022 ( 9. words)
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCHRSZ</td>
<td>I*2</td>
<td>000000</td>
<td>LSIZE</td>
<td>I*2</td>
<td>000002</td>
<td>IWIDTH</td>
<td>I*2</td>
<td>000004</td>
</tr>
<tr>
<td>IHEIGHT</td>
<td>I*2</td>
<td>000006</td>
<td>IENHAN</td>
<td>I*2</td>
<td>000010</td>
<td>TIMERA</td>
<td>R*4</td>
<td>000012</td>
</tr>
<tr>
<td>TIMHDC</td>
<td>R*4</td>
<td>000016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local and COMMON Arrays:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Section Offset</th>
<th>------Size------</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICODE</td>
<td>L*1</td>
<td>$DATA</td>
<td>000002</td>
<td>000004</td>
</tr>
<tr>
<td>IHEIGHT</td>
<td>I*2</td>
<td>$DATA</td>
<td>000010</td>
<td>000010</td>
</tr>
<tr>
<td>IW</td>
<td>I*2</td>
<td>$DATA</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>ITTOUR</td>
<td>I*2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statements and Processor-Defined Functions:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITTOUR</td>
<td>I*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

42
Subroutine GINPUT

This routine is used with those terminals such as the 4010 and 4014, having user-controlled crosshairs for interactive graphics input from the user. Each time this routine is called, the terminal switches to Graphics Input mode and displays the user-controlled crosshairs. The position of these two lines (one horizontal and one vertical) may be changed by the operator by turning a pair of thumbwheels by the keyboard. The crosshair lines are dim and do not store on the terminal screen. By using this routine, the program can allow the user to select a particular point on the display (perhaps one previously plotted with ARYPLT or XYPLT) and can then easily get the coordinates of that point from the user. The subroutine does this by waiting until the user has positioned the crosshairs as desired. Then he or she can strike a single key on the keyboard and then the Return key. When this is done, the terminal automatically transmits the first character that was struck along with the X and Y coordinates of the crosshairs to the computer. Subroutine GINPUT then returns the ASCII equivalent of the character and the crosshair coordinates to the calling program through the argument list.

Each time GINPUT is called, a single character and the coordinates of one point are sent to the program. The character struck does not appear on the screen. (The terminal "bypass" circuitry keeps this from happening.) The character transmitted can be used as a code for the program to instruct it what function should be performed next, or it can be used for any other purpose by the calling program. Although both the X and Y coordinates are always passed to the program, frequently only one or the other is needed. In that case the user can be instructed to ignore the crosshair line that is not needed (for example, the horizontal one), and use only the other one to pick out the feature of interest on the screen. This can speed up the program's operation by cutting in half the amount of time the operator must spend in positioning the crosshairs. This is especially useful when one is selecting a number of points from a plot of sampled analog data (such as may be produced by ARYPLT) versus time, for instance, since only the vertical crosshair need be used to point out the times of interest. The program can then look up the Y coordinate from the sampled data point in memory which corresponds to that time or X coordinate.
Although the graphical output routines in this package can plot points with 4096 point resolution if a 4014/4015 terminal with the enhanced graphics option is used, the Tektronix terminals are all limited to 1024 point resolution for graphical input. Therefore, in detailed plots, it may be impossible to always select the exact point of interest with the GINPUT routine. In general, your program should take the coordinates returned by GINPUT and find the point which you plotted that is closest to the returned coordinates. Even then, it may be impossible to always select a single given point from among several very closely spaced points. Usually, however, the error in selecting among closely spaced points is not noticeable.

Routine GINPUT requires the following subroutine from the System Subroutine Library:

ITTOUR

The listing for GINPUT follows.
SUBROUTINE GINPUT(ICHAR, IX, IY)

SWITCH TO GRAPHIC INPUT MODE, DISPLAY USER-CONTROLLED CROSSHAIRS. WAIT FOR USER TO TYPE IN A CHARACTER, & RETURN THAT CHARACTER (ICHAR), AND THE X AND Y COORDINATES OF THE CROSSHAIRS TO THE CALLING PROGRAM.

INTEGER ICHAR, HIGHX, LOWX, HIGHY, LOWY

SWITCH TO GRAPHIC INPUT MODE, DISPLAY CROSSHAIR

IF (ITTOUR(27) .NE. 0) GO TO 20 !SEND ESCAPE
IF (ITTOUR(26) .NE. 0) GO TO 30 !SEND SUB

WAIT UNTIL USER HITS A TERMINAL KEY, THEN GET THAT CHARACTER AND THE CROSSHAIR ADDRESS

ACCEPT 50, ICHAR, HIGHX, LOWX, HIGHY, LOWY

FORMAT (5A1)

IX = 4 * ((HIGHX .AND. '37) * 32) + (LOWX .AND. '37)
IY = 4 * ((HIGHY .AND. '37) * 32) + (LOWY .AND. '37)

RETURN

END

FORTRAN IV Storage Map for Program Unit GINPUT

Local Variables, .PSECT $DATA, Size = 000016 (7, words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGHX</td>
<td>I*2</td>
<td>000006</td>
</tr>
<tr>
<td>IX</td>
<td>I*2</td>
<td>@ 000002</td>
</tr>
<tr>
<td>LOWY</td>
<td>I*2</td>
<td>@ 000014</td>
</tr>
<tr>
<td>HIGHY</td>
<td>I*2</td>
<td>000012</td>
</tr>
<tr>
<td>ICHAR</td>
<td>I*2</td>
<td>@ 000000</td>
</tr>
<tr>
<td>IY</td>
<td>I*2</td>
<td>@ 000004</td>
</tr>
<tr>
<td>LOWX</td>
<td>I*2</td>
<td>000010</td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITTOUR</td>
<td>I*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Subroutine ARYPLT

This routine can be used to plot an integer array of Y values with equal increments in the X (horizontal) direction. It is particularly well suited for plotting sampled data such as from an analog-to-digital converter, with the samples equally spaced in time. All points are connected with straight solid lines. For integer data, ARYPLT provides a significant savings in memory usage over XYPLOT, at least if a large number of points are plotted. This is so because real arrays in PDP-11 Fortran require two words to store each element, while integer arrays only require one word per element. In addition, a separate array of X coordinates must be passed to XYPLOT, but is not needed for ARYPLT since ARYPLT generates the X coordinates automatically.

The arguments or parameters are described briefly in the program listing. In addition, you should note that all of the arguments except for YSCALE are of integer type. If IYOFST is set to 1560, then positive values in IARRAY will be plotted above the middle of the screen and negative values will be plotted below the middle. IYOFST can be set larger or smaller in order to move the plot up or down, respectively. YSCALE can be set to a value of 1.0 for many applications, if the elements of IARRAY are within a range of about -1000 to +1000, but YSCALE can be changed to adjust the vertical scale factor of the plot.

Several calls to ARYPLT can be made without erasing the screen, with different values for IYOFST and/or LEFT and RIGHT, in order to plot several curves on the screen together.

Routine ARYPLT requires the following subroutine, in addition to those from the Fortran library:

MPLOT

A listing of ARYPLT follows.
SUBROUTINE ARYPLT(IARRAY,N,IYOFST,YSCALE,LEFT,RIGHT)

ROUTINE TO PLOT AN INTEGER ARRAY OF Y VALUES
IN *IARRAY* ON THE TEKTRONIX TERMINAL.

THE X-COORDINATE IS AUTOMATICALLY GENERATED BY THIS
ROUTINE SO THAT *N* POINTS ARE PLOTTED WITH A CONSTANT
INCREMENT IN X FROM X='LEFT' TO X='RIGHT'.

ARGUMENTS:
IARRAY = (INTEGER) ARRAY OF Y VALUES TO BE PLOTTED
N = NO. OF VALUES OF IARRAY TO USE
     (STARTING WITH IARRAY(1))
IYOFST = (INTEGER) OFFSET (IN TEK. UNITS) TO BE ADDED
     TO EACH Y VALUE AFTER MULTIPLYING BY YSCALE
YSCALE = (REAL) SCALE FACTOR BY WHICH TO MULTIPLY
     EACH Y VALUE.
LEFT = (INTEGER) X-COORDINATE FOR LEFT SIDE OF PLOT
RIGHT = (INTEGER) X-COORDINATE FOR RIGHT SIDE OF PLOT
     BOTH LEFT & RIGHT ARE IN TEKTRONIX SCREEN UNITS

FOR IYOFST=0, YSCALE=1.0, LEFT=0, & RIGHT=4095, THE
VALUES IN IARRAY WILL BE PLOTTED ACROSS THE ENTIRE WIDTH
OF THE SCREEN, WITH VALUES OF 0 IN IARRAY PLOTTED AT
THE EXTREME BOTTOM & VALUES OF 3070 PLOTTED AT THE TOP
OF THE SCREEN. USUALLY IT IS BEST TO AVOID PLOTTING ALL
THE WAY TO THE EDGES OF THE SCREEN, ESPECIALLY IF HARD
COPIES ARE DESIRED.

AUTHOR: WILLIAM G. CROSIER
DATE: JULY 1980
REVISED: DEC. 1980

INTEGER IARRAY(N),IYOFST,LEFT,RIGHT,MIN,MEN,IX,IY
REAL YSCALE,DELTAX
DELTAX = FLOAT(RIGHT-LEFT)/FLOAT(N-1) !X INCREMENT
IX = LEFT
IY = IFIX(FLOAT(IARRAY(1))*YSCALE) + IYOFST
CALL MPLT(IX,IY,0)

MOVE TO POSITION OF FIRST POINT
DRAW LINES BETWEEN EACH OF THE N POINTS
DO 20 I=2,N
IX = LEFT + DELTAX*FLOAT(I-1)
IY = IFIX(FLOAT(IARRAY(I))*YSCALE) + IYOFST
CALL MPLT(IX,IY,1)
20 CONTINUE
RETURN
END
### FORTRAN IV Storage Map for Program Unit ARYPLT

Local Variables, .PSECT $DATA, Size = 000052 (21 words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELTAX</td>
<td>RX4</td>
<td>000026</td>
<td>I</td>
<td>I*2</td>
<td>000032</td>
<td>IX</td>
<td>I*2</td>
<td>000022</td>
</tr>
<tr>
<td>IY</td>
<td>I*2</td>
<td>000024</td>
<td>IYOFST</td>
<td>I*2</td>
<td>@ 000004</td>
<td>LEFT</td>
<td>I*2</td>
<td>@ 000010</td>
</tr>
<tr>
<td>MAX</td>
<td>I*2</td>
<td>000020</td>
<td>MIN</td>
<td>I*2</td>
<td>000016</td>
<td>N</td>
<td>I*2</td>
<td>@ 000002</td>
</tr>
<tr>
<td>RIGHT</td>
<td>I*2</td>
<td>@ 000012</td>
<td>YSCALE</td>
<td>RX4</td>
<td>@ 000006</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local and COMMON Arrays:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>Size</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IARRAY</td>
<td>I*2</td>
<td>$DATA</td>
<td>000000</td>
<td>****</td>
<td>(**) (N)</td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT</td>
<td>RX4</td>
<td>IFIX</td>
<td>I*2</td>
<td>MPLDT</td>
<td>I*2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Subroutine XYPLLOT**

This is a general purpose plotting routine for plotting a real array of Y values versus a real array of X values. If the parameter ICODE is set to 0, then only the points themselves are plotted (with asterisks "*"), and if ICODE is some other value, then the points are connected with straight lines. More information on the arguments/parameters passed to XYPLLOT is given in the program listing. Note that N, L, R, R, T, and ICODE are all of integer type, but that all other arguments are real.

An example of the use of XYPLLOT, along with GRID and ANOTAT, is given in the third sample program (GRTEST) in another section of this report. Please refer there for more information on a typical use of XYPLLOT.

Routine XYPLLOT uses the following subroutine, in addition to those from the Fortran Library:

**MPLOT**

The listing for XYPLLOT follows.
SUBROUTINE XYPLOT(X,Y,N,L,R,B,T,XMIN,XMAX,YMIN,YMAX,ICODE)

C ROUTINE TO PLOT AN ARRAY OF Y VALUES VERSUS AN ARRAY OF X VALUES
C ON A TEKTRONIX TERMINAL. THE POINTS ARE CONNECTED WITH STRAIGHT
C LINES.

C ARGUMENTS:
C X = ARRAY OF X-COORDINATE VALUES TO PLOT (REAL)
C Y = ARRAY OF Y-COORDINATES (REAL)
C N = NUMBER OF PAIRS OF X-Y VALUES TO PLOT (INTEGER)
C L = LEFT BOUNDARY OF PLOTTING AREA ON SCREEN (INTEGER)
C R = RIGHT PLOT BOUNDARY (INTEGER)
C B = BOTTOM PLOT BOUNDARY (INTEGER)
C T = TOP PLOT BOUNDARY (INTEGER)
C XMIN = X-VALUE CORRESPONDING TO LEFT SIDE OF PLOTTING AREA (REAL)
C XMAX = X-VALUE CORRESPONDING TO RIGHT SIDE OF PLOTTING AREA (REAL)
C YMIN = Y-VALUE CORRESPONDING TO BOTTOM SIDE OF PLOTTING AREA (REAL)
C YMAX = Y-VALUE CORRESPONDING TO TOP SIDE OF PLOTTING AREA (REAL)
C ICODE = CODE FOR CONTROLLING TYPE OF LINES TO DRAW BETWEEN
C POINTS (INTEGER). IF ICODE IS POS THEN LINES WILL BE
C CLIPPED IF THEY WOULD EXTEND PAST THE PLOT BOUNDARIES.
C IF ICODE IS 0 OR NEG THEN THE LINES MAY EXTEND PAST THE
C BOUNDARIES. THE ABSOLUTE VALUE OF ICODE DETERMINES THE
C TYPE OF LINES TO DRAW (SEE ROUTINE M PLOT).
C IF ICODE IS 0, THEN THE POINTS ARE PLOTTED WITH ASTERISKS
C (*), BUT NO LINES ARE DRAWN BETWEEN THEM.

EXAMPLES:
ICODE=0 WILL PLOT ASTERISKS WITH NO CONNECTING LINES.
ICODE=1 WILL DRAW NORMAL SOLID LINES BETWEEN POINTS
ICODE=-97 WILL DRAW DOTTED LINES BETWEEN POINTS, &
WILL ALLOW THE LINES TO EXTEND OUTSIDE THE
DESIGNATED PLOT BOUNDARIES.

NOTE: ARGUMENTS L, R, B, & T ARE ALL OF INTEGER TYPE AND ARE
IN TEKTRONIX SCREEN UNITS. L & R MUST BE BETWEEN 0 AND 4095.
B & T MUST BE BETWEEN 0 AND 3120. THESE PARAMETERS DETERMINE
THE PORTION OF THE SCREEN TO BE USED FOR A PLOT.
ARGUMENTS XMIN, XMAX, YMIN, & YMAX ARE ALL REAL AND ARE IN USER
UNITS (SAME AS IN ROUTINE ANOTAT). THEY MAY BE IN ANY RANGE
DESIRED.

AUTHOR: WILLIAM G. CROSIER
DATE: FEB. 1981

INTEGER N,L,R,B,T,ICODE,IX,IX,IXOFST,IXOFST
REAL X(N),Y(N),XMIN,XMAX,YMIN,YMAX,XSCALE,YSCALE

COMMON /GRFCOM/ MCHARSZ,LSIZE,IWIDTH,HIHGHT,IENHAN,TIMERA,
@ TIMHDC

CALCULATE SCALE FACTORS TO CONVERT X & Y VALUES
FROM USER UNITS INTO TEKTRONIX SCREEN UNITS

XSCALE = FLOAT(R-L) / (XMAX-XMIN)
ORIGIN = IN OF POOR QUALITY

FORTRAN IV  V02.1-1  Wed 23-Dec-81 10:23:08  PAGE 002

0006 IXOFST = L - XMIN*SCALE
0007 YSCALE = FLOAT(T-B) / (YMAX-YMIN)
0008 IYOFST = B - YMIN*YSCALE

C
IF ICODE IS NOT 0, DRAW LINES BETWEEN EACH OF THE N POINTS

C
DO 50 I=1,N
0010 IX = IFIX(FLOAT(X(I)) * XSCALE) + IXOFST
0011 IY = IFIX(FLOAT(Y(I)) * YSCALE) + IYOFST
0012 IF (ICODE .LE. 0) GO TO 30

C PREVENT PLOT FROM EXTENDING PAST DESIRED BOUNDARIES
0014 IF (IX .LT. L) IX=L
0016 IF (IX .GT. R) IX=R
0018 IF (IY .LT. B) IY=B
0020 IF (IY .GT. T) IY=T
0022 30 CONTINUE

C IF ICODE=0, PLOT POINTS ONLY WITHOUT CONNECTING LINES
0023 IF (ICODE .EQ. 0) GO TO 40
0025 ITYPE = 0
0026 IF (I.GT.1) ITYPE = IABS(ICODE)
0028 CALL MPLOT(IX,IY,ITYPE)
0029 GO TO 50
0030 40 CALL MPLOT(IX-IWIDTH/2,IY-IHEIGHT/3,-1)
0031 TYPE 45
0032 45 FORMAT ('+*')
0033 50 CONTINUE
0034 RETURN
0035 END
FORTRAN IV  
Store a Map for Program Unit XYPL TO

Local Variables. .PSECT $DATA, Size = 000100 ( 32. words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>I*2</td>
<td>@ 000012</td>
<td>I</td>
<td>I*2</td>
<td>000054</td>
<td>ICODE</td>
<td>I*2</td>
<td>@ 000026</td>
</tr>
<tr>
<td>ITYPE</td>
<td>I*2</td>
<td>000056</td>
<td>IX</td>
<td>I*2</td>
<td>000034</td>
<td>IXDFST</td>
<td>I*2</td>
<td>000040</td>
</tr>
<tr>
<td>IY</td>
<td>I*2</td>
<td>000036</td>
<td>IYOFST</td>
<td>I*2</td>
<td>000042</td>
<td>L</td>
<td>I*2</td>
<td>000006</td>
</tr>
<tr>
<td>N</td>
<td>I*2</td>
<td>@ 000004</td>
<td>R</td>
<td>I*2</td>
<td>@ 000010</td>
<td>T</td>
<td>I*2</td>
<td>@ 000014</td>
</tr>
<tr>
<td>XMAX</td>
<td>R*4</td>
<td>@ 000020</td>
<td>XMIN</td>
<td>R*4</td>
<td>@ 000016</td>
<td>XSCALE</td>
<td>R*4</td>
<td>000044</td>
</tr>
<tr>
<td>YMAX</td>
<td>R*4</td>
<td>@ 000024</td>
<td>YMIN</td>
<td>R*4</td>
<td>@ 000022</td>
<td>YSCALE</td>
<td>R*4</td>
<td>000050</td>
</tr>
</tbody>
</table>

COMMON Block /GrFCOM/, Size = 000022 ( 9. words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCHR SZ</td>
<td>I*2</td>
<td>000000</td>
<td>LSIZE</td>
<td>I*2</td>
<td>000002</td>
<td>IWIDTH</td>
<td>I*2</td>
<td>000004</td>
</tr>
<tr>
<td>IHIGHT</td>
<td>I*2</td>
<td>000006</td>
<td>IENHAN</td>
<td>I*2</td>
<td>000010</td>
<td>TIMERA</td>
<td>R*4</td>
<td>000012</td>
</tr>
<tr>
<td>TIMHDC</td>
<td>R*4</td>
<td>000016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local and COMMON Arrays:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>------Size------ Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>R*4</td>
<td>@ $DATA</td>
<td>000000</td>
<td>**** ( **) (N)</td>
</tr>
<tr>
<td>Y</td>
<td>R*4</td>
<td>@ $DATA</td>
<td>000002</td>
<td>**** ( **) (N)</td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT</td>
<td>R*4</td>
<td>IABS</td>
<td>I*2</td>
<td>IFIX</td>
<td>I*2</td>
<td>MPLT</td>
<td>I*2</td>
</tr>
</tbody>
</table>

52
**Subroutine GRID**

This subroutine is used to draw horizontal and/or vertical grid lines over a specified portion of the terminal screen. All of the arguments/parameters are of integer type, and their functions should be clear from reading the program listing below and the third example program (GRTEST) in another section of this report. The values for arguments LEFT, RIGHT, BOTTOM, and TOP should normally be the same in the calls to XYPLOT (or ARYPLT), GRID, and ANOTAT. If NXDIV is set equal to 0, then no vertical grid lines will be drawn. Similarly, if NYDIV is set equal to 0, then no horizontal grid lines will be drawn. ICODE specifies the line type, as described in the section on routine MPLOT.

Routine GRID uses the following subroutine, in addition to those from the Fortran Library:

**MPLOT**

The program listing for GRID follows.
SUBROUTINE GRID(NXDIV, NYDIV, LEFT, RIGHT, BOTTOM, TOP, ICODE)
THIS ROUTINE DRAWS GRID LINES OVER A DESIRED PORTION
OF THE SCREEN ON A TEKTRONIX TERMINAL.

ARGUMENTS (ALL ARE INTEGERS):
NXDIV = # OF X-AXIS DIVISIONS = # OF VERT. LINES - 1
NYDIV = # OF Y-AXIS DIVISIONS = # OF HORIZ. LINES - 1
LEFT = LEFT BOUNDARY OF PLOTTING AREA IN TEK. SCREEN UNITS (MIN. X VALUE)
RIGHT = RIGHT BOUNDARY (MAX. X)
BOTTOM = LOWER BOUNDARY (MIN. Y)
TOP = UPPER BOUNDARY (MAX Y)
ICODE = DESIGNATES TYPE OF LINES TO DRAW
(SEE ROUTINE MPLOT OR TEKTRONIX MANUAL)
EX.: ICODE = 1 -- DRAW SOLID LINES
ICODE = 97 -- DRAW DOTTED LINES

AUTHOR: WILLIAM G. CROISIER
DATE: 9 DEC. 1980

INTEGER NXDIV, NYDIV, LEFT, RIGHT, BOTTOM, TOP, ICODE, IX, IY
REAL DELTA

DRAW HORIZ. LINES
IF (NYDIV .LE. 0) GO TO 80
CALL MPLOT (LEFT, BOTTOM, 0)
CALL MPLOT (RIGHT, BOTTOM, ICODE)
DELTA = FLOAT (TOP - BOTTOM) / FLOAT (NYDIV)
DO 50 K = 1, NYDIV
IY = BOTTOM + IFIX (DELTA * FLOAT (K))
CALL MPLOT (LEFT, IY, 0)
CALL MPLOT (RIGHT, IY, ICODE)
50 CONTINUE

DRAW VERTICAL LINES
IF (NXDIV .LE. 0) GO TO 999
CALL MPLOT (LEFT, BOTTOM, 0)
CALL MPLOT (LEFT, TOP, ICODE)
DELTA = FLOAT (RIGHT - LEFT) / FLOAT (NXDIV)
DO 100 K = 1, NXDIV
IX = LEFT + IFIX (DELTA * FLOAT (K))
CALL MPLOT (IX, BOTTOM, 0)
100 CONTINUE
CALL MPLOT (IX, TOP, ICODE)
999 RETURN
EMEND
FORTRAN IV  
Storage Map for Program Unit GRID

Local Variables, .PSECT $DATA; Size = 000044 ( 18, words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOTTOM</td>
<td>I*2</td>
<td>000010</td>
<td>DELTA</td>
<td>R*4</td>
<td>000022</td>
<td>ICODE</td>
<td>I*2</td>
<td>000014</td>
</tr>
<tr>
<td>IX</td>
<td>I*2</td>
<td>000016</td>
<td>IY</td>
<td>I*2</td>
<td>000020</td>
<td>K</td>
<td>I*2</td>
<td>000026</td>
</tr>
<tr>
<td>LEFT</td>
<td>I*2</td>
<td>000004</td>
<td>NXDIV</td>
<td>I*2</td>
<td>000000</td>
<td>NYDIV</td>
<td>I*2</td>
<td>000002</td>
</tr>
<tr>
<td>RIGHT</td>
<td>I*2</td>
<td>000006</td>
<td>TOP</td>
<td>I*2</td>
<td>000012</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOT</td>
<td>R*4</td>
<td>IFIX</td>
<td>I*2</td>
<td>MPLOT</td>
<td>I*2</td>
</tr>
</tbody>
</table>
Subroutine ANOTAT

This subroutine is used to anotate (label) the horizontal and/or vertical plot axes with numerical user units at some or all grid lines (previously produced by routine GRID). Most of the arguments or parameters for this routine are the same as for GRID and XYPLOT, and you may refer to the listings of those routines and of ANOTAT itself (given below) for more information. In addition, the third sample main program (GRTEST) in another section of this report gives an example of how these routines can be used. You may note that the values for NXDIV and NYDIV may be different in the call to ANOTAT from what they were in the call to GRID. This can be done if you want to label only every second, fifth, etc., grid line with ANOTAT.

Be sure to call routine CHRSIZ at least once in your program before calling ANOTAT. This is necessary so that ANOTAT will know what the current character size is, so that it can position the numerical units for the axes properly. In addition, make sure that arguments L and B are large enough so that ANOTAT will not attempt to type the numerical units to the left of, or below, the allowable plotting area. This may especially be a problem if one of the two larger character sizes are being used. If L or B are too small, then the numbers may be typed over the plot axes and be difficult to read.

ANOTAT requires the following subroutine, in addition to those from the Fortran Library:

MPLOT

A listing of ANOTAT follows.
SUBROUTINE ANOTAT(NXDIV, NYDIV, L, R, B, T, XMIN, XMAX, YMIN, YMAX)

C ROUTINE TO ANOTATE PLOT AXES WITH USER UNITS AT GRID LINES.
C IF ROUTINE GRID IS ALSO USED, THE FIRST 6 ARGUMENTS SHOULD BE
C IDENTICAL TO THOSE USED IN GRID. IN THAT CASE, THE GRID LABELS
C WILL BE TYPED JUST OUTSIDE THE PLOTTING AREA.
C
C ARGUMENTS:
C FIRST 6--SAME AS FOR GRID (ALL INTEGER):
C NXDIV=X-AXIS DIVISIONS = # VERT. LINES - 1
C NYDIV=Y-AXIS DIVISIONS = # HORIZ LINES - 1
C L = LEFT BOUNDARY OF PLOT IN TEKTRONIX SCREEN UNITS
C R = RIGHT BOUNDARY
C B = BOTTOM BOUNDARY
C T = TOP BOUNDARY
C XMIN = MIN. X VALUE TO BE TYPED AT BOTTOM LEFT (REAL)
C XMAX = MAX. X VALUE TO BE TYPED AT BOTTOM RIGHT (REAL)
C YMIN = MIN. Y VALUE TO BE TYPED AT BOTTOM LEFT (REAL)
C YMAX = MAX. Y VALUE TO BE TYPED AT TOP LEFT (REAL)
C
C IF NXDIV IS GREATER THAN 0, THEN NXDIV+1 NUMBERS (STARTING WITH
C XMIN & ENDING WITH XMAX) ARE TYPED BELOW VERT. GRID LINES (DRAWN
C SEPARATELY WITH ROUTINE GRID) WITH AN F6.0 FORMAT JUST BELOW PLOT.
C IF NYDIV IS GREATER THAN 0, THEN NYDIV+1 NUMBERS (STARTING WITH
C YMIN & ENDING WITH YMAX) ARE TYPED TO THE LEFT OF HORIZ. GRID
C LINES (DRAWN WITH GRID) WITH AN F6.0 FORMAT.
C
C AN F6.0 FORMAT WILL BE USED UNLESS THE MIN. X OR Y VALUE IS
C GREATER THAN -9.99 AND THE MAX VALUE IS LESS THAN 99.9, IN
C WHICH CASE AN F6.3 FORMAT WILL BE USED INSTEAD.
C
C AUTHOR: WILLIAM G. CROSIER
C DATE: 16 DEC. 1980

INTEGER NXDIV, NYDIV, L, R, B, T, IX, IY
REAL XMIN, XMAX, YMIN, YMAX, DELTA, DELTAY, DELTAX
COMMON /GRFCOM/ MCHRSZ, LSIZE, IWIDTH, IHIGH, IENHAN,
               TIMERA, TIMHDC

DATA FMT /4H('+' +4H,F6.0),4HO) /

LABEL Y (VERTICAL) AXIS

IF (NYDIV .LE. 0) GO TO 80
FMT(3) = '0)
IX = L - 6*IWIDTH
DELTA = FLOAT(T-B) / FLOAT(NYDIV)
DELTA = (YMAX-YMIN) / FLOAT(NYDIV)
DO 50 K=1, NYDIV+1
IY = B + IFIX((DELTA*FLOAT(K-1)) - IFIX(0.2*IHIGH))
MOVE TO DESIRED POSITION TO LEFT OF AXIS, ALPHA MODE
CALL MPLOT(IX, IY, -1)
Y = YMIN + DELTAY * FLOAT(K-1)

57
FORTRAN IV

0018 TYPE FMT, Y
0019 CONTINUE
C
C 0020 IF (NXDIV .LE. 0) GO TO 999
0022 FMT(3) = '0',
C
C 0023 IY = B - 1.2*IHIGHT
0024 DELTA = FLOAT(R-L) / FLOAT(NXDIV)
0025 DELTA1 = (XMAX-XMIN) / FLOAT(NXDIV)
0026 DO 100 K=1,NXDIV+1
C
C 0027 IX = L + IFIX(DELTA*FLOAT(K-1)) - 4*IWIDTH
C
C 0028 CALL MPLOT(IXPIYP-1)
0029 X = XMIN + DELTAI * FLOAT(n-1)
C
C 0032 TYPE FMT, X
0033 CONTINUE
C
0034 RETURN
0035 END

FORTRAN IV

Storage Map for Program Unit ANOTAT

Local Variables, .PSEC $DATA, Size = 000132 ( 45. words)

Name Type Offset Name Type Offset Name Type Offset
B I*2 @ 000010 DELTA R*4 000044 DELTA1 R*4 000050
IX I*2 @ 000040 IY I*2 000042 K I*2 000064
L I*2 @ 000004 NXDIV I*2 000000 NYDIV I*2 000002
R I*2 @ 000006 T I*2 000012 X R*4 000054
XMAX R*4 @ 000016 XMIN R*4 000014 Y R*4 000060
YMAX R*4 @ 000022 YMIN R*4 @ 000020

COMMON Block /GRFCOM/ , Size = 000022 ( 9. words)

Name Type Offset Name Type Offset Name Type Offset
MCHRSZ I*2 @ 000000 LSIZE I*2 000002 IWIDTH I*2 000004
IHIGHT I*2 000006 IENHAN I*2 000010 TIMERA R*4 000012
TIMHDC R*4 000016

Local and COMMON Arrays:

Name Type Section Offset -----Size----- Dimensions
FMT R*4 $DATA 000024 000014 ( 6.) (3)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name Type Name Type Name Type Name Type
FLOAT R*4 IFIX I*2 MPLOT I*2

58
DATA ACQUISITION
AND MISCELLANEOUS
SUBROUTINES
Subroutine BELL

This subroutine is used to make the terminal beep or ring its bell. The sound produced depends on the particular terminal. The duration of the sound, as well as its modulation, can be controlled with the two parameters or arguments NUMBER and IDELAY. Examples of the use of this routine are given in the third sample main program (GRTEST), in another section of this report.

Routine BELL requires the following subroutines, in addition to those from the Fortran Library:

WAIT
ITTOUR (from the System Subroutine Library)

The listing for subroutine BELL follows.
SUBROUTINE BELL(NUMBER, IDelay)

RING TERMINAL BELL/BEEP WITH VARIABLE DURATION & MODULATION CONTROL

NUMBER = NO. OF BELL CHARACTERS TO TRANSMIT (CONTROLS DURATION)
IDELAY = NO. OF 1/60 SEC INCREMENTS TO WAIT BETWEEN BELLS (CONTROLS MODULATION & PERCEIVED FREQUENCY).

IDELAY CAN BE 0 FOR A CONTINUOUS TONE, WITH DURATION CONTROLLED BY NUMBER, OR IDELAY CAN BE A POSITIVE INTEGER TO PRODUCE A BUZZING SOUND OR DISCRETE BEEPS.

NOTE: SOUND IS DEPENDENT ON THE TERMINAL & ON ITS BAUD RATE SETTING.

DO 100 K=1, NUMBER
        IF (IDELAY .LT. 1) GO TO 40
        DELAY = FLOAT(IDELAY) / 60.0
        CALL WAIT(DELAY, 0)
        40 IF (ITTOUR(7) .NE. 0) GO TO 40 !SEND BELL

CONTINUE
RETURN

END

FORTRAN IV Storage Map for Program Unit BELL

Local Variables, .PSECT $DATA, Size = 000012 ( 5. words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELAY</td>
<td>R*4</td>
<td>000006</td>
<td>IDELAY</td>
<td>I*2</td>
<td>@ 000002</td>
</tr>
<tr>
<td>NUMBER</td>
<td>I*2</td>
<td>@ 000000</td>
<td>K</td>
<td>I*2</td>
<td>000004</td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT</td>
<td>R*4</td>
<td>ITTOUR</td>
<td>I*2</td>
<td>WAIT</td>
<td>R*4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Subroutine WAIT

This routine uses the RT-11 system line frequency clock to time a waiting period. The user simply passes the routine a real value in argument/parameter SEC which specifies the duration, in seconds, of the waiting period. The normal functions of the line time clock are not affected. When the waiting period has elapsed, then control is returned to the calling program. If you want to be able to terminate the wait prematurely (in less than "SEC" seconds), then an interrupt service routine can be used to set the argument IABORT equal to a non-zero value. You may want to do this in a real-time experiment control program, if something happened during a programmed wait or if you pressed a button on a control panel, for example. If you do not need to prematurely terminate the waiting period, then set IABORT equal to 0. An example of the use of this routine can be seen in the listing for subroutine ERASE. This routine calls WAIT after sending the command to erase the terminal screen, so that the terminal will have enough time (normally TIMERA is 1.5 seconds) to completely clear the screen.

Subroutine WAIT may also be used to time the periods between data acquisition samples, if the sampling rate is fairly slow (60 Hz or slower). The accuracy and resolution of the programmed wait is 1/60 = 0.017 second, since that is the time between cycles of the line frequency. There should be no cumulative time error between waiting periods, however, as long as not too many CPU operations are performed, so that the long term accuracy of the times measured should be quite good. As an example of using WAIT to control data acquisition, the following could be used to get 1000 samples of signals on analog channel 3 and 5 with the samples taken every 1/30 = 0.033 second:

```
INTEGER IDATA(1000),IDATB(1000)
IPGNC = 2     !PROGRAMMABLE GAIN CODE
DO 10 K=1,1000 !DO 1000 TIMES
   IDATA(K) = ISAMPA(3,IPGNC,0) !SAMPLE CHANNEL 3
   IDATB(K) = ISAMPA(5,IPGNC,0) !SAMPLE CHANNEL 5
   CALL WAIT(0.033,0)
10 CONTINUE
```
At the end of 33 seconds, 1000 samples would be collected from Channel 3 and stored in array IDATA, while 1000 samples from Channel 5 would be in array IDATB. If numerous computations such as averaging of many samples are performed between waiting periods, then the period may actually be longer than desired, since the computations make take more than 1/60 of a second.

If you have a 50 HZ line frequency system clock, rather than the 60 HZ usually used in the United States, be sure to change this line in the program:

```
from:  TICKS = SEC*60. + 0.5
to:    TICKS = SEC*50. + 0.5
```

Do this only if you have a 50 HZ line frequency clock.

Routine WAIT requires the following subroutines from the System Subroutine Library:

- **GTIM** (Time of Day in clock ticks past midnight)
- **JADD** (Integer*4 addition)
- **JAFIX** (Real*4 to Integer*4 conversion)
- **JCMP** (Integer*4 compare)
- **JJCVT** (Interchange halves of Integer*4 variable)

A listing of WAIT follows.
SUBROUTINE WAIT(SEC, IABORT)

C LINE TIME CLOCK WAIT ROUTINE
C WRITTEN BY: WILLIAM G. CROSIER
C SEC = NUMBER OF SECONDS (REAL, NOT INTEGER) TO WAIT
C USES LINE TIME CLOCK FOR TIMING CONTROL.
C THE J--- SUBROUTINES USE HERE PERFORM INTEGER*4 ARITHMETIC.
C RESOLUTION & ACCURACY = APPROX. 0.017 = 1/60 SECOND
C THIS ROUTINE DOES NOT AFFECT NORMAL FUNCTIONS OF LTC.
C IF PARAMETER IABORT BECOMES NON-ZERO DURING
C THE WAITING PERIOD (IF SET BY AN INTERRUPT ROUTINE),
C THEN THE WAIT IS IMMEDIATELY TERMINATED.
C IF THIS FEATURE IS NOT NEEDED, USE A VALUE OF 0 FOR IABORT

INTEGER*4 ITIM1, ITIM2, IDELTA
CALL GTIM(ITIM1)  !STORE CURRENT TIME IN ITIME
CALL JCVT(ITIM1)  !INTERCHANGE WORDS
TICKS = SEC*60. + 0.5  !CONVERT SEC TO CLOCK TICKS
CALL JAFIX(TICKS, IDELTA)  !CONVERT TO INTEGER
CALL JADD(ITIM1, IDELTA, ITIM1)  !CALCULATE STOP TIME
CALL GTIM(ITIM2)  !GET CURRENT TIME OF DAY
CALL JCVT(ITIM2)  !INTERCHANGE WORDS
IF(IABORT .NE. 0) GO TO 99  !CHECK FOR ABORT
IF(JCMP(ITIM2, ITIM1) .LT. 0) GO TO 10
99 RETURN
10 END

FORTran IV Storage Map for Program Unit WAIT

Local Variables, .PSECT $DATA, Size = 000024 ( 10. words)

Name Type Offset Name Type Offset Name Type Offset
IABORT I*2 @ 000002 IDELTA I*4 000014 ITIM1 I*4 000004
ITIM2 I*4 000010 SEC R*4 @ 000000 TICKS R*4 000020

Subroutines, Functions, Statement and Processor-Defined Functions:

Name Type Name Type Name Type Name Type
GTIM R*4 JADD I*2 JAFIX I*2 JCMF I*2 JCVT I*2
**Subroutine ISAMPA**

This is a Fortran-callable subroutine, written in MACRO Assembly Language, for sampling an analog signal with an analog-to-digital (A/D) converter. It was written in MACRO so that it can execute as quickly as possible, but still be usable with FORTRAN programs. With this routine, sampling rates of several hundred samples per second can easily be achieved, even with some computations performed between samples. For accurate control of the time intervals between samples, you may use routine WAIT if the sampling rate is 60 Hz or slower. Otherwise, you should use a programmable clock/timer such as the KW-11P. If ISAMPA is called by a MACRO interrupt service routine for the KW-11P, make sure that the routine uses the normal PDP-11 Fortran calling conventions for passing arguments, etc.

If accurate control of the sampling rate is not a requirement, but you need to sample a large number of values in a certain time period and average them in order to reduce noise effects, you may use the following procedure. First, get the current time of day (in seconds past midnight) with the RT-11 system routine SECNDS, or wait until an appropriate external event occurs. Second, call ISAMPA, convert the returned sampled value to real or double precision, and add it to a real or double precision variable used as an accumulator. Repeat the sampling and accumulating until either you have enough samples, or until enough time has elapsed. (Use the SECNDS subroutine again.) Finally, divide by the number of samples collected. A real variable (rather than integer) should be used if you are using a 12-bit A/D converter and are adding together more than 16 samples, because a 16-bit integer accumulator can be overflowed by adding more than 16 12-bit values together if each of them are near full scale. Generally, no error message will occur if this happens, since PDP-11 Fortran does not check for an overflow on an integer add operation. Similarly, you should use a double precision accumulator if you add together more than about 2000 samples, because you can drop bits when doing so with Real*4 arithmetic.

For examples of how ISAMPA may be used, refer to the discussions for subroutine WAIT and for the first sample main program (ADTEST), in another section of this report.
If you are using a DEC ADV-11A A/D converter, then you must mask out the four most significant bits, since DEC uses them for other purposes. In addition, DEC's ADV-11A converters can only be used with an offset binary format, so that a value of 4000 (octal) or 2048 (decimal) must be subtracted from the sampled value in order to convert it to the normal two's complement coding. The following will mask out the 4 MSB's and convert the value to 2's complement:

\[ I = (\text{'7777 .AND. ISAMPA(ICHAN,0,1)}) - '4000 \]

This is necessary only with DEC A/D boards. Note also, in the above example, that a value of 0 must be used for the second argument (IPGMC0D), since the DEC boards do not have programmable gain.

ISAMPA requires no subroutines.
A listing of ISAMPA follows.
TITLE ISANPA

INTERFACE FUNCTION ISANPA

INPUT: SINGLE AUDIO CHANNEL I/O USING PROGRAMMABLE
output gains normalized to full data translation or DEC.

ANALOG INTERFACE BOARD.

USE: I = ISANPA1(chan,IPGNCU,MODIFY)
WHERE ICHAN CAN BE FROM 0 TO 31 (15 ON SOME BOARDS)
AND IPGNCU IS 0,1,2, OR 3
10. FOR DATA TRANSITION A/D'S, 0 GIVES LOW, 5
11. (UNITY) GAIN, AND 3 GIVES HIGHEST GAIN (8 OR
12. 10, DEPENDING ON BOARD USED),
13. FOR ADAC A/D'S, 0 GIVES HIGHEST GAIN (8 OR 10),
14. AND 3 GIVES LOWEST GAIN (UNITY).
15. IF THE BOARD YOU ARE USING DOES NOT HAVE
16. PROGRAMMABLE GAIN (DEC ONE'S DO NOT),
17. THEN YOU SHOULD SPECIFY 0 FOR IPGNCU.
18. AND MOD1FY IS 0 OR 1
19. 0 IS USED FOR ADAC A/D BOARDS. 1
20. 1 IS USED FOR DATA TRANSLATION OR DEC.

VALUE RETURNED BY THE FUNCTION (IN RO) IS THE INTEGER
22. NUMBER OF COUNTS FROM THE A/D CONVERTER (-2048 FOR NEG.
23. FULL SCALE; 2048 FOR POS. FULL SCALE). ONLY THE LOW
24. ORDER 12 BITS ARE VALID IF A DEC A/D BOARD IS USED.

NOTE: THE A/D BOARD CSR ADDRESS SHOULD BE SET
26. TO 176770 (DECD)

WRITTEN BY: WILLIAM U. CROSIER

DATE: 2 MAY 1981

.GLOBAL ISANPA
ADCSR=176770
ADDA=ADCSR+12

ISANPA: MOV @R5,RO
LOAD R5 GAIN CODE IN RO

ASL RO
LSHIFT LEFT 2 BITS

ASL RO

MOV #ADCSR,R1
STORE A/D CSR ADDR. IN R1

TST R1
TEST A/D TYPE

BNE DTDEC
IF NOT TYPE 0

ASL RO
ADAC BOARD: 6 SHIFT 1 MORE BIT

MOV R1,(R1)
PUT gains code in A/D CSR

MOV R2(RS)+1(R1)
LOAD CH 2 START CONVERSION

BR GETBAT
GET FIRST SAMPLED VALUE

DTDEC:
DATA TRANSLATION OR DEC BOARD

MOV @2(RS),R1
PUT CHAN 2 IN A/D CSR HI BYTE

MOV R0,(R1)
PUT GAIN CODE IN A/D CSR

INC (R1)
SET BIT 0 (START CONVERSION)

GETBAT: MOV #ADDA,RO

LOOP: TSTB (R1)
IA/D CONVERSION DONE?

BPL LOOP
WAIT TILL FINISHED

MOV (R0),RO

RTS PC
RETURN

.END
## SYMBR TABLE

- GDSK = 176770
- RMPDATA = 176772
- ARS = 000000 000
- CDRS = 000004 001
- ERRORS DETECTED = 0

Virtual Memory Used: 8192 Words (32 Pages)  
Dynamic Memory Available For: 64 Pages

## ISAMPA CROSS REFERENCE TABLE (CREF V04.00)

<table>
<thead>
<tr>
<th>Module</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGDCR</td>
<td>1-34#</td>
</tr>
<tr>
<td>AGDATA</td>
<td>1-35#</td>
</tr>
<tr>
<td>RDEC</td>
<td>1-42#</td>
</tr>
<tr>
<td>RITDAT</td>
<td>1-46#</td>
</tr>
<tr>
<td>ISAMPA</td>
<td>1-33#</td>
</tr>
<tr>
<td>LOOP</td>
<td>1-52#</td>
</tr>
</tbody>
</table>
Subroutine DISKIO

This routine is used to read or write binary unformatted data, in a random fashion, to a sequential disk file. Variable length records are supported (in multiple of 256 integer words), and the routine is somewhat more economical of memory and CPU time than the normal Fortran disk I/O, since the transfer takes place directly from the arrays in the user's program, rather than through intermediate buffers. Any binary data can be transferred, regardless of whether the calling program treats it as logical, integer, real, or string data. The principal restrictions on the data is that it must all be in contiguous memory locations, and that only multiples of 256 words should be transferred normally through each call to DISKIO. Generally, the data should all be placed in a COMMON block in order to force the compiler to place it all in contiguous locations, unless it is in a single array.

The arguments/parameters for this routine are discussed in the program listing below. The file name passed in argument FILNAM can be any valid RT-11 file name. Note, however, that it must be exactly 12 characters (bytes) long, with no colons or periods within it to separate the device name or file type/suffix. Trailing spaces are allowed, however, at the end of each portion of the file name in order to make the device name identifier exactly 3 characters long and the main part of the file name exactly 6 characters long. A null or zero byte should follow the 12 character file name (as in the MACRO assembler ASCIZ construction).

Data written by this routine can only be read back with the same routine, and not by Fortran READ statements. In addition, this subroutine was designed to work only with the RT-11 operating system.

For more information and examples of the use of DISKIO, refer to the fourth sample main program, DISKRW, in another section of this document.

Refer to the program listing which follows for more information.
SUBROUTINE DISKIO(FILNAM, MODE, BUFFER, NWRDS, IBLK, NBLK, IERR)

PURPOSE: READ OR WRITE BINARY DATA TO A DISK FILE

WRITTEN BY: WILLIAM G. CROSIER

DATE: 11 JUNE 1980

ARGUMENTS:

FILNAME=ARRAY CONTAINING ASCII FILE NAME (12 CHAR.)
(FILNAM IS IGNORED WHEN MODE WAS NEG. ON THE LAST CALL TO DISKIO.)

EX: D XITEST07DAT
    DK MYFILE
    DK FILE7

MODE=+1 OR -1 TO CREATE A NEW FILE & WRITE OUT DATA TO IT
(+1 OR -1 WILL CAUSE ANY FILE WITH THE SAME NAME WHICH PREVIOUSLY EXISTED TO BE DELETED WHEN THE NEW FILE IS CLOSED)
=2 OR -2 TO MODIFY AN EXISTING FILE (OVERWRITE ALL OR PART)
=3 OR -3 TO READ DATA FROM AN EXISTING FILE
IF MODE IS POSITIVE, THE FILE IS CLOSED AFTER THE I/O.
IF MODE IS NEGATIVE, THE FILE IS NOT CLOSED, SO THAT THE NEXT CALL TO DISKIO WILL NOT REQUIRE RE-OPENING THE FILE.
(THE NEXT CALL WILL ALSO IGNORE FILNAM SINCE THE PREVIOUSLY SPECIFIED NAME WILL BE USED AGAIN.)
IF AN ERROR OCCURS WHEN MODE IS NEG., THE NEXT I/O MAY NOT BE VALID SINCE THE FILE MAY NOT BE OPENED PROPERLY. TO AVOID THIS PROBLEM, DO A READ OPERATION WITH MODE=3 TO CLOSE THE FILE IF AN ERROR OCCURS WHEN MODE IS NEG.
NOTE: ALL FILES ARE UNCONDITIONALLY CLOSED WHEN THE PROGRAM TERMINATES, REGARDLESS OF WHETHER MODE WAS POS. OR NEG. ON THE LAST CALL.

BUFFER=AREA IN MEMORY WHERE DATA IS TO BE TRANSFERRED TO/FROM

NWRDS=NO. OF INTEGER WORDS TO READ FROM OR WRITTEN INTO BUFFER
(SHOULD BE A MULTIPLE OF 256)

IBLK=STARTING BLOCK NO. IN FILE WHERE DATA TRANSFER IS TO OCCUR

NBLK=NO. OF 256-WORD INTEGER BLOCKS TO ALLOCATE FOR A NEW FILE
(NBLK IS IGNORED EXCEPT WHEN MODE=1 OR -1)

IERR=ERROR CODE RETURNED BY DISKIO
=0 MEANS NO ERRORS OCCURRED
=1 MEANS QUEUE ELEMENT FAILURE OCCURRED
=2 MEANS NO I/O CHANNEL WAS AVAILABLE
=3 MEANS HANDLER FOR SPECIFIED DEVICE CAN'T BE LOADED
=4 MEANS FILE ALLOCATION FAILED WHEN CREATING FILE
=5 MEANS A DATA OUTPUT ERROR OCCURRED
=6 MEANS A FILE LOOKUP FAILURE OCCURRED (COULD NOT FIND FILE):
C =7 MEANS A DATA INPUT ERROR OCCURRED
C THIS ROUTINE TAKES CARE OF OPENING & CLOSING THE FILE FOR
C EACH DATA TRANSFER, SETTING QUEUE ELEMENTS APPROPRIATELY,
C GETTING AN I/O CHANNEL, FETCHING THE DEVICE HANDLER, CREATING
C THE FILE ENTRY, & DOING THE ACTUAL DATA I/O
C
0002 BYTE FILNAM(12)
0003 INTEGER MODE,BUFFER,NWRDS,IBLK,NBLK,IFILE(4),FLAG,IERR,ICHAN,
@IPMODE
0004 COMMON /DISCOM/IPMODE
0005 DATA IPMODE /0/
0006 IERR=0
0007 IF (IPMODE.LT.0) GO TO 60 !FILE LEFT OPEN?
C CONVERT FILE NAME TO RADIX-50
0009 CALL IRAI150(12,FILNAM,IFILE)
0010 IF (IPMODE .NE. 0) GO TO 20
C FIRST TIME ROUTINE HAS BEEN CALLED, SO SET QUEUE ELEMENTS
0012 IF (IOSET(2) .EQ. 0) GO TO 20
0014 IERR=1 !ERROR-QUEUE ELEMENT FAILURE
0015 GO TO 999
0016 20 ICHAN=IGETC(IDUMMY) !GET AN I/O CHANNEL
0017 IF (ICHAN .GE. 0) GO TO 30 !ERR-NO CHAN. AVAIL.
0019 IERR=2 !ERROR-NO CHAN. AVG.
0020 GO TO 99
0021 30 IF (IFETCH(IFILE(1)) .EQ. 0) GO TO 40 !FETCH DEVICE HANDLE.
0023 IERR=3 !ERROR-CANNOT LOAD HANDLER
0024 GO TO 90
0025 40 IF (IABS(MODE) .GT. 1) GO TO 50
C CREATE NEW FILE ENTRY
0027 IF (IENTER(ICHAN,IFILE,NBLK).GE.0) GO TO 60
0029 IERR=4 !ERROR-FILE ALLOCATION FAILED
0030 GO TO 90
0031 60 IF (IABS(MODE) .EQ. 3) GO TO 70
C WRITE OUT DATA FROM BUFFER
0033 IF (IWRITT(NWRDS,BUFFER,IBLK,ICHAN).GE.0) GO TO 90
0035 IERR=5 !ERROR-DATA OUTPUT
0036 GO TO 90
C FIND EXISTING FILE
0037 50 IF (LOOKUP(ICHAN,IFILE).GE.0) GO TO 60
0039 IERR=6 !ERR IN FILE LOOKUP
0040 GO TO 90
C READ DATA INTO BUFFER
0041 70 IF (IREADW(NWRDS,BUFFER,IBLK,ICHAN).GE.0) GO TO 90
0043 IERR=7 !ERR IN READING DATA
0044 90 IF (MODE .LT. 0) GO TO 99 !LEAVE CHAN. OPEN?
0046 CALL CLOSEC(ICHAN) !CLOSE THE I/O CHANNEL
0047 CALL IFREEC(ICHAN) !$FREE IT
0048 99 IPMODE=MODE
0049 999 RETURN
0050 END
FORTRAN IV Storage Map for Program Unit DISKIO

Local Variables, _PSECT $DATA, Size = 000034 ( 14. words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER</td>
<td>I*2</td>
<td>@ 000004</td>
<td>FLAG</td>
<td>I*2</td>
<td>000026</td>
<td>IBLK</td>
<td>I*2</td>
<td>@ 000010</td>
</tr>
<tr>
<td>ICHAN</td>
<td>I*2</td>
<td>000030</td>
<td>IDUMMY</td>
<td>I*2</td>
<td>000032</td>
<td>IERR</td>
<td>I*2</td>
<td>@ 000014</td>
</tr>
<tr>
<td>MODE</td>
<td>I*2</td>
<td>@ 000002</td>
<td>NBLK</td>
<td>I*2</td>
<td>@ 000012</td>
<td>NURDS</td>
<td>I*2</td>
<td>@ 000006</td>
</tr>
</tbody>
</table>

COMMON Block /DISCOM/, Size = 000002 ( 1. words)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPMODE</td>
<td>I*2</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local and COMMON Arrays.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Section</th>
<th>Offset</th>
<th>------Size------</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILNAM</td>
<td>L*1</td>
<td>@ $DATA</td>
<td>000000</td>
<td>000014</td>
<td>( 6.) (12)</td>
</tr>
<tr>
<td>TFILE</td>
<td>I*2</td>
<td>$DATA</td>
<td>000016</td>
<td>000010</td>
<td>( 4.) (4)</td>
</tr>
</tbody>
</table>

Subroutines, Functions, Statement and Processor-Defined Functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSEC</td>
<td>R*4</td>
<td>IABS</td>
<td>I*2</td>
<td>IENTER</td>
<td>I*2</td>
<td>IFETCH</td>
<td>I*2</td>
</tr>
<tr>
<td>ICETC</td>
<td>I*2</td>
<td>IQSET</td>
<td>I*2</td>
<td>IRAD50</td>
<td>I*2</td>
<td>IREADW</td>
<td>I*2</td>
</tr>
<tr>
<td>LOOKUP</td>
<td>I*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>