Copyright © 1984, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

DATA ACQUISITION AND DATA REDUCTION FOR MMX

by

B. T. Archer III

Memorandum No. UCB/ERL M84/85

13 October 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DATA ACQUISITION AND DATA REDUCTION FOR MMX

Branch T. Archer III

October 13, 1984

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley, CA 94720

Contents

1 INTRODUCTION

This document describes the data acquisition system and data reduction system developed as part of the UC Berkeley Ten Meter Multiple Experiment research effort. There are three major divisions of this document. The first describes codes in use on the local computer system, called S–100 (S–100 refers to the 100-pin standard bus used by the various cards of the system). The second division describes some general purpose analog electronics available for use in conjunction with the digital data acquisition system. The third major division deals with data reduction facilities available on the National Magnetic Fusion Energy Computation Center (NMFECC) network, to which the group has access.

There are several sections of this document titled PROGRAMMER'S NOTES. These sections provide information which is helpful to the person in charge of maintaining this system, and can be ignored by most users of this data reduction facility.

PROGRAMMER'S NOTES

This document is called MMXDR.TEX and was processed by the LATEX document-preparation program on the NMFECC "ccc" machine. The sources are stored both on the ccc machine, and also disk 1C of the S–100 system.

2 CODES RESIDING ON S–100

This section describes the codes which are available on the S–100 computer system. The primary function of these codes is acquisition of MMX data from the bank of digitizers via the CAMAC interface, and subsequent transmission of the raw data to NMFECC for analysis. Also available is a library of subroutines which simplify the writing of codes for local data reduction.

2.1 MMX MASTER PROGRAM

As the flexibility and complexity of our computer system has increased, it has become desirable to provide an orderly system for transferring control between executable codes. This function is provided by the code MMX.CMD.

When the code MMX.CMD is executed, it automatically opens and reads the text file CHAIN.LST. This text file is created with any standard editor, and consists of the names of the files to which the user would like to chain. DIGIT.CMD (Section 2.2) is automatically included as the first option. In the following example, CHAIN.LST has been set up to chain to MFE and MAVTC.

```
SW MFE
MAVTC
```

Note that codes which run on the 8085 must be preceded by SW, so the operating system knows that the object code is intended for execution by the 8085 processor.

If the file CHAIN.LST is not found on the default disk, the default chaining list is simply RUNTRAN. Once a code has been executed from MMX, the called code may return to MMX by calling the chain routine. The following FORTRAN source statement accomplishes this function:

```
CALL CHAIN('MMX$')
```

It may prove desirable to have the code MMX run automatically when you log in. This is achieved by simply creating a .SUB file which contains the text MMX followed by a carriage return, and is called LOGIN.SUB:

Contents of LOGIN.SUB.

```
MMX
```

PROGRAMMER'S NOTES

MMX.CMD is written in PASCAL, as this was the only 8086 language available at the time. The source code is MMX.PAS of user 3, disk A. If re-written, C would be the language of choice for the re-write.

2.2 DATA ACQUISITION

Data from MMX is displayed on the MicroAngelo graphics monitor and stored to disk via the code DIGIT.CMD. DIGIT is executed by a command line of the following form:

DIGIT *filename*

If *filename* is present, then DIGIT reads the indicated file and automatically enters FILE I/O mode. Whenever DIGIT is exited, the most recently used header is written to a file called HEADER. When DIGIT is next executed, it reads this file and uses it for the default header.

There are two lists of selections in DIGIT. The first one is reproduced in Figure 1, where *filename* is replaced by the most recently used file name. A description of each command follows.

A: This brings you to the FILE I/O selections of Figure 2 discussed later.

B: To set module parameters enter the slot # of the module to be set up. A list of options appears for the sampling rate, pre-trigger samples, and record length. Enter three single digits separated by spaces. The LeCroy 8837F's differ slightly in operation from the Transiac modules The hardware is unable to detect their presence when in sampling mode. This causes a problem since the code checks to see if a module exists before it allows it to be set up or used in a header. Thus the module must be in readout mode before attempting these operations.

```
*** DIGITIZER CONTROLLER ***

(A) - FILE I/O
(B) - SET UP A MODULE
(C) - ENABLE TRIGGERS
(D) - DISPLAY WAVEFORMS
(E) - SAVE ALL WAVEFORMS - filename
(F) - HALT SAMPLING
(G) - CHAIN TO MMX
(H) - INCREMENT FILE
(I) - EDIT SHOT LOG

   ENTER OPTION:
```

Figure 1: Main selection list for DIGIT.CMD

C: The user is asked for verification of the command so that data will not be unwitting erased. If the user replies with 'y' (or 'Y'), then all the modules are put into sampling mode by the computer.

D: After selecting this option, enter the slot number of the channel to be displayed. The channel is displayed on the MicroAngelo which corresponds to the user's terminal location (one MicroAngelo is associated with each terminal). Enter 'R' or 'L' to scroll right or left on the display. The Transiac 2001 is displayed in 8K blocks numbered 0–3. Enter 'B' followed by a digit 0–3 to look at the corresponding block. Enter ESC to look at another channel. When asked for the channel to display, entering '0' followed by RETURN exits the user from this option.

E: The data is saved into the file set up via the FILE I/O commands (option A).

F: The last used file name is incremented in the standard way to indicate the next shot number. For example AUG1301 becomes AUG1302. All header parameters remain unchanged.

G: This causes DIGIT to chain to MMX (Section 2.1) from which one may then chain to any desired program.

H: The next shot number is generated, according to the standard indicated in Section 4.5.

I: The CED editor (Section 2.3) is invoked to edit the shot log. The name of the shot log is obtained from the current file name by appending 'LOG.LOG' to the first 5 characters of the current file name. The string "SHOT=*shotname*" is placed in the

```
*** FILE I/O ***

(A) - READ AN OLD HEADER
(B) - OPEN A NEW FILE
(C) - DELETE A FILE
(D) - RENAME A FILE
(E) - INSPECT DIRECTORY
(F) - PRINT OUT HEADER
(G) - DISPLAY DATA FILE

   ENTER OPTION:
```

Figure 2: FILE I/O Selection list for DIGIT.CMD

yank buffer, so that when CED executes that string may be placed in the shot log by entering ^Y.

ESC: This is the command for exiting DIGIT. The user is first given a chance to return to the system, in case the ESC key was hit inadvertently.

Selecting option 'A' of the main selection list produces a FILE I/O selections, reproduced in Figure 2. Hitting the ESC key returns to the main selection list. The FILE I/O commands should all be self-explanatory. The only point that has seemed to cause confusion is the command to open a file. The code requests the slot #'s that will be used for the shot. These must be entered one per line, terminated by a '0'. Then the user is asked to give names to each slot to be used on the shot. Each name is a maximum of 7 characters long.

PROGRAMMER'S NOTES

DIGIT.CMD is written in DeSmet C. The source is divided into two parts, DIGIT.C and DIGIT1.C. The object code must be linked to CAMAC.C, MA.C, and MMXLIB.A (compliation instructions are contained in the source DIGIT.C). CAMAC.C handles the CAMAC interface. MA.C handles the MicroAngelo graphics interface. A library of assembly level functions called MMXLIB.A is written for linkage to the C code. All these codes reside on disk 1B.

2.3 CED editor

CED (C editor) is a screen-oriented editor intended primarily for editing the shot log in conjunction with data acquisition on the MMX. Advanced features of the WYSE terminal are used to minimize time-consuming terminal I/O. The limit for text length is 32K, which

is sufficient for any reasonable shot log. There is also a 24K 'yank buffer'. The yank buffer is a buffer of text which is not presently in the file, but can be inserted into the text at a the cursor position by ∧Y (Control-Y) or ESC K commands (see below). CED is invoked with a command line of the following form:

CED *filename yankbuffer*

If no command line arguments are present, the program prompts for a filename. If two arguments are present, the second argument is text which will be automatically placed in the yank buffer. Once the filename is entered, the file is read and the name of the file is displayed on the top line of the WYSE monitor for easy reference. The name remains there until a different file is read. Once a file is read, all characters typed are entered at the current cursor position with the exception of control characters and characters followed by an ESCape character. These characters comprise editing commands which are detailed below. Mnemonics are indicated in boldface where possible.

Control Commands

Some control characters already have special meanings. For example, typing Control-H is the same as hitting the BACK SPACE key. These equivalences are noted in the following list.

∧*A:* Move to the beginning of the current line.

∧*B:* Move Back one character.

∧*D:* Delete character at current cursor position.

∧*C:* Insert Channel information is inserted into the text in the standard shot log format (Section 4.4) at the current cursor position. This is a sequence of lines of the form CHAN=*channame*. The channel information is obtained from the file HEADER, which is automatically generated by DIGIT upon exit.

∧*E:* Move to the End of the current line.

∧*F:* Move Forward one character.

∧*H:* Same as ∧B or the BACK SPACE key.

∧*I:* Equivalent to TAB.

∧*J:* Join the current line with the next. Equivalent to LINE FEED.

∧*K:* Kill the text between the cursor and the end of line. The killed text is added to the yank buffer.

∧L: Send text to Line printer. ∧L is entered at two different cursor positions. Upon entry of the second marker, the intervening text is printed on the line printer. This capability allows one to avoid printing an entire file when only a small portion is actually of interest.

∧M: Make a new line. If the cursor is in the middle of a line, the current character becomes the start of the new line. Equivalent to RETURN or ENTER.

∧N: Move to the Next line.

∧O: Omit the previous character. Equivalent to RUB OUT.

∧P: Move to the Previous line.

∧R: Initiate a Reverse search.

∧S: Initiate a forward Search. ∧R and ∧S prompt for a search string. Entry of the string is terminated by the ESC character. If ESC is entered as the first character, the most recently used string is echoed back to the console and used for the search. While entering the search string, ∧O, ∧H, or BACK SPACE all erase the last character entered. ∧X erases the entire string. After entry of the string, the cursor is positioned at the first occurence of the string before (∧R) or after (∧S) the current cursor position.

∧U: Universal argument. This causes the next character to be repeated by Count times. Count begins at 4. Continuing to enter ∧U multiplies Count by 4 each time, mod 1000. This is useful for moving several lines or characters at a time, or for insertion of many of the same character (for example, a row of stars). If a more accurate Count is desired (i.e., a Count which cannot be achieved by the algorithm described), enter the desired count after striking ∧U. This is useful for locating a certain line in a file, such as a line that causes a compilation error.

∧V: View next page (cursor advances 23 lines).

∧Y: Yank all text from the yank buffer to the current cursor position. The yank buffer itself is unmodified by this command.

∧Z: Zip to the end of the text.

Escape Commands

Following the ESCape character with a character other than those listed will cause the terminal to beep and no further action will be taken.

ESC C: Copy text into the yank buffer. Entering this command at two different locations adds the intervening text (including the characters at the cursor positions) into the yank buffer. The text remains the same.

ESC G: Go to the marker set by the ESC S command.

ESC K: UnKill the last line put into the yank buffer. The line is inserted at the cursor position, and removed from the yank buffer. This command is especially useful for retrieving accidentally killed lines of text.

ESC L: Clear the Line printer marker (see ∧L above).

ESC M: Move text into the yank buffer. This is similar to ESC C, but the text that is moved into the yank buffer is removed from the text buffer.

ESC P: Display the Position (line and column) of the cursor in the file. This information is displayed at the bottom of the screen.

ESC R: Replace a text string. Prompts are given for a search string and a replacement string. The strings are entered with the same editing features as are ∧S and ∧R. At each occurence of the search string, the cursor is moved to the string. The user then enters a character indicating the desired action. 'A' or 'a' means to abort the replacement. 'G' or 'g' means to make all replacements without further queries. 'S' means to skip the current replacement and query for the next replacement. Any other character means to make the current replacement and proceed to the next.

ESC S: Set a marker at the current text position. The cursor will be moved to this position by an ESC G command.

ESC V: View the previous screen. The cursor is moved backward 23 lines.

ESC Y: This command voids the Yank buffer.

File Handling Commands

These commands handle file manipulations. Each consists of the control character ∧*X* followed by another control character.

∧*X* ∧ *C:* Same as ∧*X* ∧ *Q*.

∧*X* ∧ *D:* Chain to the digitizer controller program DIGIT.

∧*X* ∧ *Q:* Quit. If the current text has been modified, the user will be given the opportunity to return back to the editor.

∧*X* ∧ *R:* Read a text file. If the current file has been modified, the user will be given the opportunity to abort the READ command. If the command is not aborted, a prompt is given for the file name to be opened.

∧*X* ∧ *S:* Save the current file with using the same file name as that used on the read. The previous file is renamed to have extension .BAK.

∧*X* ∧ *W:* Write the current file to disk. This is the same as ∧*X* ∧ *S*, but a prompt for the file name is issued. The file name entered can be different from the name of the input file.

PROGRAMMER'S NOTES

CED.CMD is implemented in DeSmet C. The source code consists of the modules CED.C and CED1.C. A few functions have been implemented in an assembly level module called CEDASM.A. Linkage instructions are given in the source CED.C.

2.3.1 S–100 Data Storage

This section details the format for raw data files on the S–100 system.

DATE,TIME: 6 one-byte integers. These integers denote the month, day of month, last two digits of the year, the hour of the day (24 hour time), minute, and second the file was created.

FILE NAME: 8 characters. The 8 characters of the file name are stored here, padded with spaces.

FILE EXTENSION: 3 characters. The 3 character file extension is stored here, padded with spaces.

NAME: 7 characters for each of 32 channels.

Now there are 6 arrays of 32 one-byte integers. A given position in the array describes the parameters for a single digitizer channel. See Section 2.7 for details on the interpretation of these codes.

SLOT NUMBERS: Slot numbers of the channels being used. Unused positions are filled with zeroes.

SAMPLING INTERVALS: These are the digitizer sampling codes 0–7.

RECORD LENGTH: These are the codes for the number of points in the data record (0–7).

PRE-TRIGGER SAMPLES: Codes for the number of pre-triger samples (0–7).

DEVICE TYPE: Codes for the device type (0–3)

CRATE NUMBER: The crate associated with the channel.

The raw data for each channel follows. The raw data consists of a sequence of 8-bit data bytes for each channel indicated in the arrays mentioned above. 64-byte data records are padded to 128 bytes, the block length for our system.

2.4 S–100 TO MFE DATA TRANSMISSION

Transmission of MMX Transiac data to the NMFECC network is accomplished via two programs: *MFE* on the S–100 system and *receive* on the MFE network. The data sending procedure begins by executing the MFE program on the user number under which the data resides. A list of options appears on the console (Fig. 3). The password is not protected

```
*** MFE PROGRAM ***

(1) - COMMUNICATE
(2) - SEND A TRANSIAC FILE
(3) - RE-ENTER LOGIN DATA
(4) - (UNUSED)
(5) - SEND A TEXT FILE
(6) - RECEIVE A TEXT FILE
(7) - SEND ALL TRANSIAC FILES WITH SPECIFIED NAME

LOGIN DATA ENTERED: x 1111 x 818sbl ^Zabcdef
```

Figure 3: Main Selection List for MFE Link

so that you can be sure it is entered correctly. If there is an error, it can be re-entered using option (3). One may then select option (1), which allows the user to communicate with the MFE network as though he were at a normal terminal. The communication rate is limited to 4800 baud (half the normal rate of 9600 baud), however, because of the overhead associated with terminal I/O and other functions of the multi-user S–100 system. On selecting option (1), it takes a couple of seconds for the system to respond with the **SPEED CHANGE COMPLETE** message, as the PDP-11 must be slowed down to the 4800 baud rate. After this message is received, the user may automatically send the login line by typing ^L (Control-L). Once logged onto the system, one needs to execute the *receive* program. If this program is not on the user's local file space, it may be retrieved from filem via the command line:

```
filem rds 1235 .s100 receive
```

Once the *receive* program is ready for data, it resonds with the ampersand character ('&'). Now the user types ^Q (Control-Q) to return to the list of options. Again, there is some delay as the system is sped back up to 9600 baud for data transmission. Now the user should verify that his login line is correct, then select the type of file send desired. Option (2) sends a single Transiac file. Option (7) allows entry of a file name containing wild cards (* or ?). All files satisfying this designation under the current user number are sent.

Once data transmission has begun, an ampersand appears on the console each time a line of data to MFE has been acknowledged by the *receive* program. The user may enter ^D (Control-D) to detach the console from the job and do other tasks at the same console. Once ^D is entered, information which would normally be written to the console is instead written to a file on the default drive called MFE.LOG. At any time execution can be called back to the console by typing `ATTACH MFE`.

If the S–100 system does not receive a prompt back from MFE for 3 minutes after sending

a line of data, the S–100 system will attempt to log onto the system using the login data entered previously. Obviously, if this information is incorrect, the system will never be able to log on. When the login procedure succeeds, the program *receive* is automatically executed, and transmission resumes with the file on which transmission failed. On normal completion of the transmission of data, the list of options will re-appear. To exit the program, simply hit the *ESC* key.

Option (4) sends text files to the text editor TEDI or TRIXGL. Its operation is slightly different from that of the Transiac data file options. To use option 4, first use communiate mode to set the tabs for TEDI as desired. Then put TEDI into insert mode and wait for the ampersand prompt. Lastly, exit communicate mode and enter '4'.

PROGRAMMER'S NOTES

The source for the *receive* program is called *creceive* and is written in CAL. The source resides in directory .s100 under user 1235. This code was written before I knew of some of the powerful facilities provided by the MFE library **baselib**. If there is occasion to revise this code, it could be substantially simplified by use of the library routines *zmovechr*, *zciatob*, etc. Also, calls to **fortlib** could be avoided altogether by appropriate substitution of **baselib** routines.

2.5 S–100 UTILITY PACKAGE

UTILITY.REL is a package of utility routines which perform a number of housekeeping functions. The codes are written in assembly level and are designed for linkage to FORTRAN. The FORTRAN data type declarations appropriate for the routines described below are as follows:

```
INTEGER IDRIVE,IDAT,IOFF
INTEGER*1 CHAR,ROW,COL,FLAG
INTEGER*1 FILIN(14),FILOUT(11),DATIM(6),DATA(8192)
```

CONIN reads a character from the default console. If no character is available, the routine waits for one to be typed. A flag is provided which permits some preliminary processing of the character.

FLAG VALUE	INTERPRETATION
0	No conversion
1	Convert to upper case
2	Convert to a one byte number
3	Accept yes or no response only (y or n, upper or lower case)

Coding example:

```
CALL CONIN(CHAR,FLAG)
```

```
      IF (CHAR.EQ.90) WRITE (1,1000)
1000  FORMAT ('The character Z was entered')
```

This example also indicates that the character typed can be determined by comparison with a decimal value. This is inconvenient and makes the code less readable, but the MicroSoft F80 compiler does not make character comparisons correctly.

WYSMOV moves the cursor to the desired location on the WYSE console. The console is 24 rows × 80 columns.

```
      CALL WYSMOV(10,20)
      WRITE (1,1000)
1000  FORMAT ('+LINE 10, COLUMN 1)
```

Unfortunately FORTRAN insists on inserting a carriage return before any formatted write statements, so that the column is reset to one as soon as the WRITE statement is executed. The '+' sign in the example above at least prevents the normal line feed from also being inserted.

WYSDMP dumps the contents of the console to the printer. To use it first move the cursor to the desired row and column of the end of the dump using WYSMOV, then call WYSDMP.

```
      CALL WYSMOV(10,80)
      CALL WYSDMP
```

In this example, all the text from (1,1) to (10,80) inclusive is sent to the printer.

PARSE extracts file name and drive information from the array FILIN. The routine places the parsed file name into the array FILOUT, sets the variable IDRIVE to the drive number.

DRIVE LETTER	IDRIVE
NONE	0
A...E	1...5

Coding example:

```
C Read in a CP/M-type file name; e.g.  'B:TEST.DAT' or 'X.FOR', etc.
      READ (1,1010) FILIN
1010  FORMAT (14A1)
Assume FILIN='B:TEST.DAT'
      CALL PARSE(FILIN,FILOUT,IDRIVE)
Now FILOUT='TEST␣␣␣␣␣DAT' and IDRIVE=2
```

CHAIN enables one program to execute another. CHAIN requires a single argument which is a string of characters containing the file name terminated by a '$' character. If the program to be chained to is a .COM file, the argument must begin with 'SW␣' (where ␣ denotes a space).

```
CALL CHAIN('MAVTC$')
CALL CHAIN('SW C:DISPLAY$')
```

DATIME returns the system data and time in a 6 element INTEGER*1 array. Only the last two digits of the year are returned, and the hours are returned in 24 hour time.

```
CALL DATIME(DATIM)
```

INT2 is a FORTRAN function which converts unsigned byte values into unsigned 16-bit values. If this conversion is not done on Transiac data values, then values above 127 will be interpreted as negative numbers. This function must be declared EXTERNAL in the the program segment (main program or subroutine) which attempts to use the function. The following program fragment might be used to subtract an offset IOFF from a Transiac data value before further manipulation.

```
EXTERNAL INT2
IDAT=INT2(DATA(I))-IOFF
```

2.6 FORTRAN ⇔ MicroAngelo Interface

The code MAPRNT.CMD file is executed at boot time. The function of this code is to load the macros for plotting various point styles (see routine POINT in the UTILITY.REL documentation) onto the μAngelo board. Also routines for dumping the screen onto the printer, and for encoding the screen for sending to MFE (see MAVTC.CMD) are loaded onto the μAngelo board.

UTILITY.REL is a package of assembly level subroutines which handle the FORTRAN interface to the S-100 μAngelo graphics board. The available routines are described below. In all cases, when a variable is indicated as INTEGER*1, a two byte integer may be used (type INTEGER), but the most significant byte (high byte) is ignored.

The μAngelo monitor is mapped as a Cartesian grid which is numbered from 0 to 511 on the X-axis and 0 to 479 along the y-axis. IX and IY represent pixel locations, and are of type INTEGER.

There is also an alphanumeric mode which is used on the LABEL routine. In alphanumeric mode the screen consists of 40 rows numbered 0-39 from top to bottom, and 85 columns numbered 0-84 from left to write. ROW and COL are row and column numbers, respectively. They are of type INTEGER*1.

In the following descriptions the variable MODE indicates a 1 or 2-byte integer. There are 3 possible MODE values:

MODE VALUE	EFFECT
0	Turn off relevant pixels.
1	Turn on relevant pixels.
2	Complement relevant pixels.

The first thing a program which is going to use graphics routines should do is call the routine GPHINT. This routine initializes the graphics board.

```
CALL GPHINT
```

ERASE clears the μAngelo screen (i.e., turns all pixels off).

```
CALL ERASE
```

LABEL writes an alphanumeric string at (ROW,COL). The string must terminate with a '$' sign.

```
LOGICAL*1 LBL(3)
DATA LBL/'H','I','$'/
CALL LABEL(10,0,LBL)
CALL LABEL(20,30,'This is a label$')
```

TRACK turns the tracking cross off (FLAG=0) or turns the tracking cross on (FLAG=1) at (IX,IY).

```
IX=25
IY=45
CALL TRACK(IX,IY,1)
CALL TRACK(25,45,0)
```

REGION affects the coordinates of a rectangular region according to the MODE parameter. (IXLO,IYLO) identifies the lower left coordinate of the region; (IXHI,IYHI) identifies the upper right coordinate.

```
CALL REGION (IXLO,IYLO,IXHI,IYHI,MODE)
CALL REGION (50,50,100,100,2)
```

MOVE moves the μAngelo graphics cursor to (IX,IY).

```
CALL MOVE (IX,IY)
```

RMOVE moves the graphics cursor by an offset (IX,IY) relative to the last positioning of the cursor by MOVE, RMOVE, VECTOR, or RELVEC.

```
CALL RMOVE(IX,IY)
```

VECTOR affects the coordinates between the current location to location (IX,IY) according to the MODE value.

```
CALL VECTOR (IX,IY,MODE)
```

RELVEC affects the coordinates between the current location and the offset value (IX,IY) according to the MODE value.

```
CALL RELVEC (IX,IY,MODE)
```

POINT plots a point at location (IX,IY) with a point style indicated by the variable STYLE.

STYLE VALUE	STYLE PLOTTED
0	•
1	□
2	△
3	×
4	◇

```
INTEGER*1 STYLE
IX=5
IY=40
STYLE=0
CALL POINT(IX,IY,STYLE)
CALL POINT(30,50,2)
```

Plot a character at the current cursor location and advance cursor, or set the character plotting mode. FLAG=0 ⇒ plot character; FLAG=1 ⇒ set mode.

```
      INTEGER*1 CHAR,FLAG
      DATA CHAR,FLAG/'P',0/
C plots 'p' at current location
      CALL PLTCHR(CHAR,FLAG)
C plot 'X' at current location
      CALL PLTCHR('X',0)
C set plotting mode to 2 (see sec.  4.7 in uAngelo manual)
      CALL PLTCHR(2,1)
```

2.7 FORTRAN ⇔ Digitizer Interface

TRANSIAC.REL contains routines which control the Transiac data acquistion channels. The routines are callable from FORTRAN, and are used primarily for more specialized data reduction operations than the simple display of raw data provided by DIGIT.CMD.

The following INTEGER*1 variables are used for examples in the FORTRAN calling sequences:

MODNUM	module slot #
SMPCOD	sampling interval code
RECLEN	record length code
PRETRG	pre-trigger samples code
DEVTYP	device type
DATA	an array to hold raw Transiac data
BLKNUM	a flag for which 8K block to read out into the DATA array

MODNUM indicates the slot number of the digitizer channel. It can range in value from 1–32.

SMPCOD is a code which determines the interval between sampling points for the digitizers. There are at present three different types of digitizers, and each may have a sampling code ranging from 0–7. The relationship between DEVTYP, SMPCOD and the sampling rate is given in the following table. The "Sampling rates" column gives the sampling rates corresponding to the sampling codes 0–6. For each digitizer, code 7 corresponds to an external clock.

DIGITIZER	Sampling rates (μsec)
Transiac 2008	0.05, 0.10, 0.20, 0.50, 1.00, 2.00, 5.00
Transiac 2001	0.01, 0.02, 0.05, 0.10, 0.20, 0.50, 1.00
LeCroy 8837F	.03125, .0625, .125, .25, .50, 1.00, 2.00

In FORTRAN a real array such as the following will give the desired conversion:

```
      REAL SMPTIM(3,8)
      DATA SMPTIM/0.05,0.10,0.20,0.50,1.00,2.00,5.00,0.,
     $ 0.01,0.02,0.05,0.10,0.20,0.50,1.00,0.,
     $ .03125,.0625,.125,.25,.50,1.00,2.00,0./
      RATE=SMPTIM(DEVTYP,SMPCOD+1)
```

RECLEN is converted into a number of data points via the following relationship:

DIGITIZER	Number of data points
Transiac 2008	NMPNTS=2**(13−RECLEN)
Transiac 2001	NMPNTS=2**(15−RECLEN)
LeCroy 8837F	NMPNTS=1024*(RECLEN+1)

Be sure to make NMPNTS an INTEGER*4 variable if the Transiac 2001 module is to be used, as its maximum number of data points (32768) will be intepreted as a negative number if a normal INTEGER variable is used.

PRETRG is converted into the number of pre-trigger data points as follows:

```
IF (DEVTYP.EQ.1) MODVAL=32
IF (DEVTYP.EQ.2) MODVAL=128
NUMPTG=PRETRG*(NMPNTS/8)
IF (DEVTYP.LE.2) NUMPTG=NUMPTG-MOD(NUMPTG,MODVAL)
```

If the 2001 is going to be used, NUMPTG should be declared INTEGER*4.

DEVTYP currently has one of 4 possible values.

DEVTYP	Meaning
0	No module
1	Transiac 2008
2	Transiac 2001
3	LeCroy 8837F

TRNCHK is a routine which verifies that a module exists and is powered up in the slot number indicated by MODNUM. The value DEVTYP is returned as previously indicated.

```
CALL TRNCHK(MODNUM,DEVTYP)
```

TRNINT is a routine which sends a 'start sampling' trigger to the Transiac indicated by MODNUM ,putting it in the sampling mode. The green CVT light on the given TRANSIAC module will light up.

```
CALL TRNINT(MODNUM)
```

TRNSET and **TRNRD** are the routines whereby the sampling parameters are set and read, respectively. The user may consult the TRANSIAC 2008 manual for details on the meanings of these parameters. The current settings of each parameter for the TRANSIAC 2008 units appear on the face of each unit as a 3 digit binary code represented by LED's. The 2001 module has no LED's. The user must be careful not to set the 2001 unit to 'Local' and expect to exercise control over the settings from the computer console. The following code fragment sets the parameters of module #MODNUM with the TRNSET call. Then TRNRD reads the parameters back out again.

```
CALL TRNSET(MODNUM,SMPINT,RECLEN,PRETRG)
CALL TRNRD (MODNUM,SMPINT,RECLEN,PRETRG,DEVTYP)
```

TRNDAT reads data out from a specified channel into an INTEGER*1 array. The array must be large enough to hold the maximum amount of data that will ever be sent into it. A dimension of 8192 is safest. Because of memory limitations in our S-100 system, the Transiac 2001 module requires special treatment for reading out more than 8K of data. The data is read out in 8K blocks, numbered from 0 to 3. Set the BLKNUM parameter according to whick 8K block is desired. When the module is a 2008 always set BLKNUM=0.

```
CALL TRNDAT(MODNUM,DATA,BLKNUM)
```

Because the FORTRAN compiler uses 2's complement arithmetic, data in the array must be first converted to INTEGER form before manipulation, so that data points greater than 127 will not be interpreted as negative numbers. This is accomplished by always operating on the array values with the INT2 function before performing computations. INT2 is available in the UTILITY module (Section 2.5).

3 SCREEN ROOM HARDWARE

3.1 AMPLIFIER PANEL PINOUTS

The following table details the pin designations for the rack-mounted amplifier panels in the screen room. The pins are numbered from 1 to 22, bottom to top. There are two physical rows of pins on the connectors, but they are electrically connected.

PIN #	PIN DESCRIPTION
6	Power Supply Ground
8	$+12V$
9	Signal Output Ground
10	V_{out}
12	$-12V$
14	V_{in}
15	Signal Input Ground

3.2 TEN X SIGNAL AMPLIFIERS

These linear signal amplifiers plug into the amplifier panel of Section 3.1. The amplifiers have a voltage gain of 10 from DC to a 3dB bandwidth of 15 MHz. The amplifiers are designed primarily for use in conjunction with the Transiac 2008 digitizers. The high frequency limit of 15 MHz is well above the 10 MHz Nyquist frequency implied by digitization at 20 MHz, the top speed for our 2008 digitizers. The amplifier input impedance is 50Ω and the modules will directly drive the 50Ω signal input of the Transiac modules. The input impedance can be changed by the simple substitution of a different resistor at the op amp input, and units which have a value different from 50Ω will so labelled. If a gain of less than 10 is desired, the *input* should be attenuated. Attenuating the output means that the amplifier will have to work harder, and the bandwidth of the amplifier could be effectively reduced.

CIRCUIT DESCRIPTION

The circuit used is the 10X Buffer Amplifier depicted in the National Semiconductor *LINEAR* catalog under applications for the LH0032 Operational Amplifier with three minor modifications (Fig. 4).

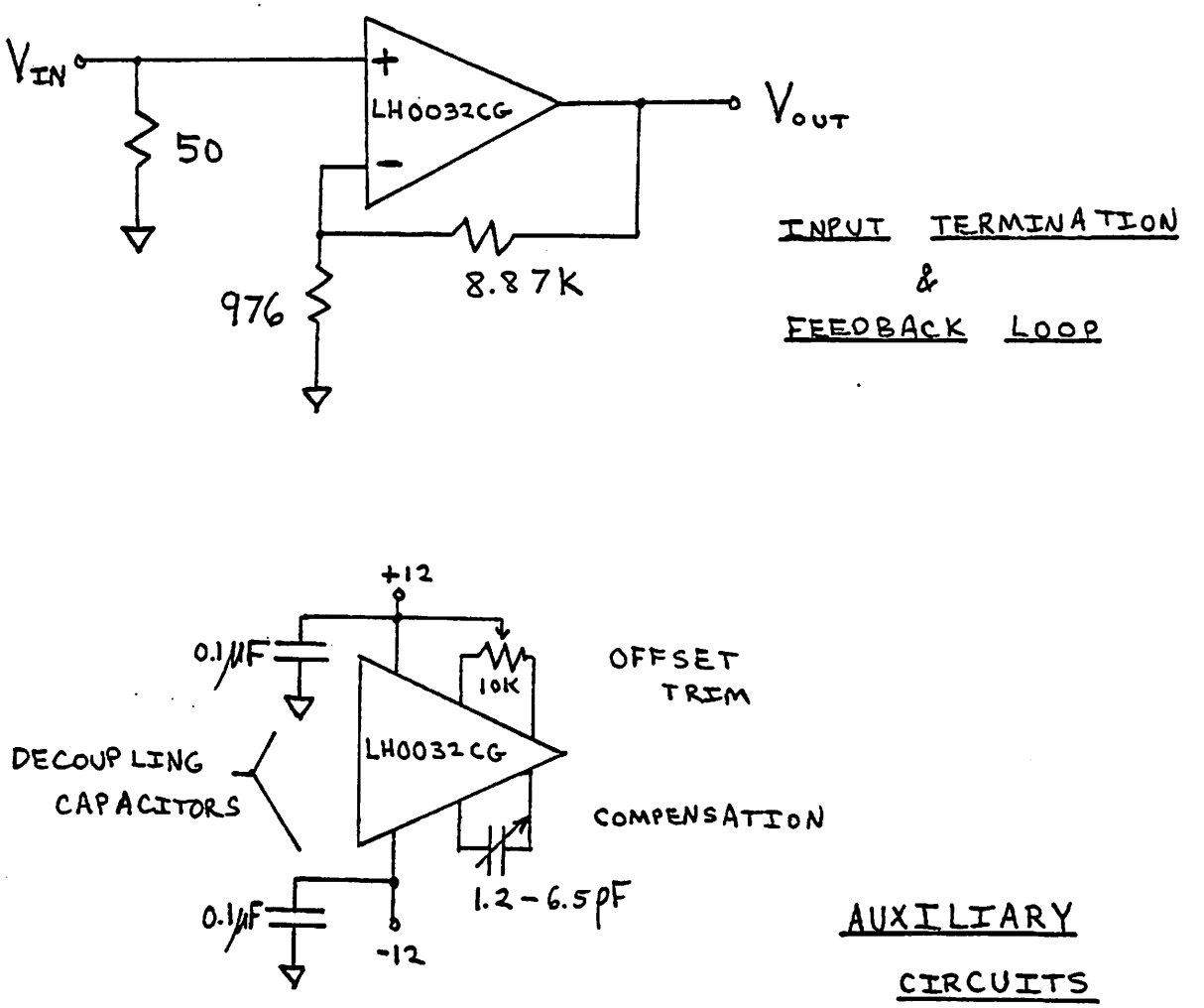


Figure 4: Circuit for 10X Signal Amplifiers

1. The 5pf compensation capacitor has been replaced by a trimming capacitor.
2. Both power supplies are bypassed by .1 μF monolithic capacitors.
3. The input is terminated by 50Ω.

OPTIMIZATION ADJUSTMENT

Circuit adjustments are to be made with a flat blade screwdriver. The handle should be of plastic or other insulating material, so that the stray capacitance introduced by touching the screwdriver to the circuit will be small. Two simple adjustments are required to optimize the performance of each amplifier.

1. With the circuit powered up, ground the input. Load the output with 50Ω and measure the output voltage. There should be less than $2mV_{p-p}$ ripple voltage. The DC portion of the output is called the input offset voltage, and should be set to zero by adjustment of the 10K trimpot.
2. Now apply a $50mV_{p-p}$ 10 MHz sine wave to the input and monitor the output. Adjust the trimming capacitor for a gain of 10.

POWER CONSUMPTION

For a full scale output voltage ($\approx .5V_{p-p}$), each amplifier module consumes something less than 20 ma for each of two supplies at $\pm$ 15 V. A supply for several modules should be chosen accordingly.

3.3 DATA COMPRESSION AMPLIFIERS

Purpose

These amplifiers are designed to overcome the limited dynamic range of the transient digitizers for MMX. Their logarithmic transfer characteristic compresses the high end of the input signal range relative to the low end, thus expanding the range of signals which can be digitized.

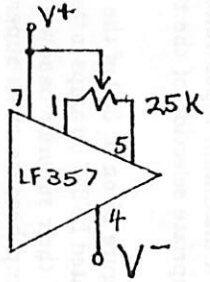
Electrical Characteristics

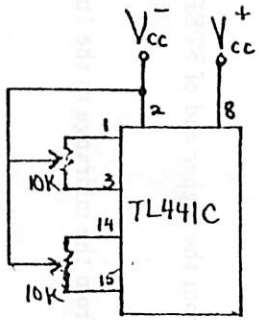
INPUT IMPEDANCE:	10K
OUTPUT IMPEDANCE:	$\ll 50\Omega$
TRANSFER CHARACTERISTIC:	$V_{out} = 0.148 log_{10} V_{in} + .405$ Volts
OPERATING RANGE:	$1mV \leq V_{in} \leq 6V$

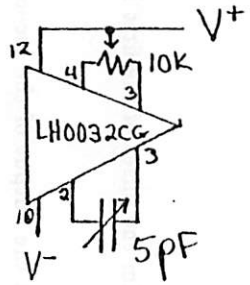
Circuit Adjustment

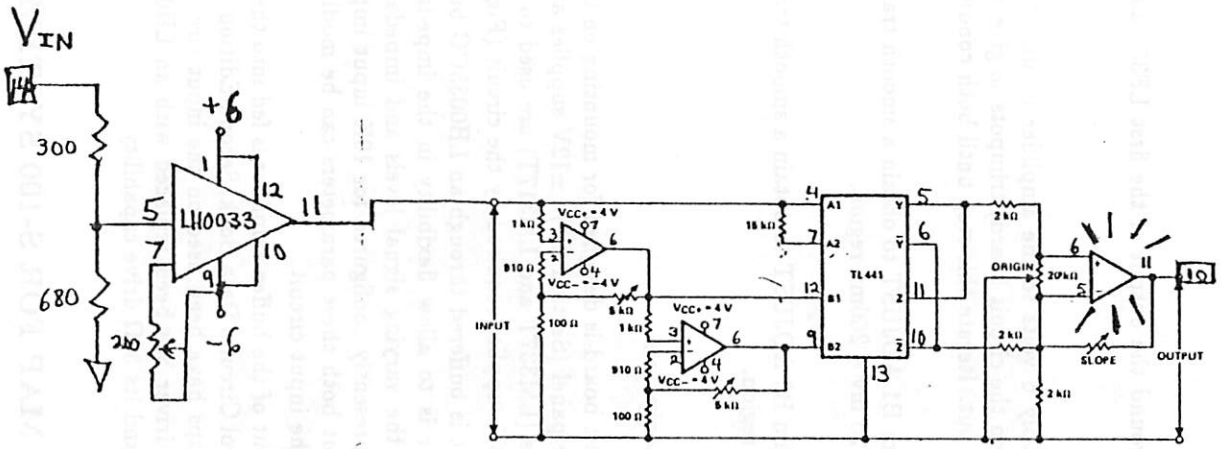
STEP I: Ground the input of the input buffer (LH0033CG) and trim the offset to 0V.

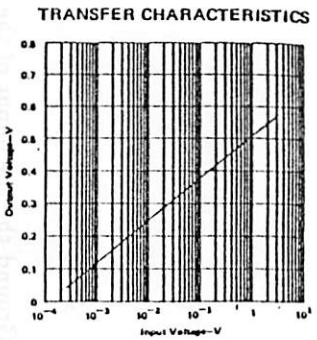
AUXILIARY CIRCUITS











NOTES: A. Inputs are limited by reducing the supply voltages for the input amplifiers to ±4 V.
B. The gains of the input amplifiers are adjusted to achieve smooth transitions.

FIGURE 15–LOGARITHMIC AMPLIFIER WITH INPUT VOLTAGE RANGE GREATER THAN 80 dBV

Figure 5: Data Compression Amplifier Circuit

STEP II: Ground the the ouput of the LH0033CG and trim the offset of the first LF357 to 0V.

STEP III: Ground the output of the first LF357 and trim the output of the second LF357 to 0V.

STEP IV: Apply 6 volts to the amplifier input. Adjust the ORIGIN and/or SLOPE (see labels on the circuit board) trimpots to give $\approx$ 500mv out. Now apply 1 volt to get $\approx$ 408mv out. Iterate this step until both conditions are met.

STEP V: Trim B1 ADJUST to obtain a smooth transition from the higher end of STEP IV to the 20 mv – 200mv region.

STEP VI: Trim B2 ADJUST to obtain a smooth transition from the midrange to the 1mv – 50 mv region.

Circuit description

The circuit board is designed for mounting on the 22-pin edge-card connector found on the amplifier panel (Section 3.1). $\pm$12V supplies are available from the power supply, and IC regulators (LM337T and LM317T) are used to step down the available voltages to the $\pm$4V and $\pm$6V supplies required by the circuit (Fig. 5).

The input is buffered through an LH0033CG buffer amplifier IC. The primary purpose of this buffer is to allow flexibility in the impedance and attenuation characteristics to accomodate the varying signal levels and impedances which may be encountered. The circuits are presently configured for 10K input impedance, and a maximum input voltage of 6 volts, but both these parameters can be modified by appropriate selection of the two resistors of the input circuit.

The output of the buffer amplifier is fed into the circuit of Figure 15 on p. 330 of the TI Linear Control Circuits Data Book, Second Edition. Uncompensated LF357 op amps instead of 741 op amps have been used on the input circuit because of their superior bandwidth. The output driver has been replaced with an LH0032CG op amp because of its superior bandwidth, and its 50Ω drive capability.

3.4 I/O MAP FOR S-100 SYSTEM

The I/O usage for the S-100 system is given in the following chart.

PORT (HEX)	DESCRIPTION
10—17	Interfacer III I/O ports
40—4F	CAMAC crate controller interface [a]
50—5F	System Support I/O ports [b]
C0—C3	Floppy disk controller I/O ports
C8—C9	Hard disk controller (Disk 2)
E0—E1	MicroAngelo graphics board (Screen room)
E2—E3	MicroAngelo graphics board (Room 117)
F0	Selector channel for Disk 2
FD—FE	Memory manager/processor swap control port

[a] This area is a reserved area under MP/M 8-16. See p. 71 of the MP/M 8-16 reference manual.
[b] See p. 19 of System Support 1 User's Manual for a detailed description of these ports.

4 CODES RESIDING AT NMFECC

This section describes data reduction facilities which are available on the NMFECC network, and which are useful to users of the Ten Meter Multiple Mirror experiment. All these facilities reside in *filem* under user 1235. They are in the directory *.mmx*, unless otherwise specified. The canonical example of the usage of this package of routines is the program *plngprb1* in directory *.mmxinp*, which reduces Langmuir Probe data. See this code for examples of most of the items you see discussed here.

PROGRAMMER'S NOTES

The following convention has been used to name the source, intermediate, and executable files:

1. Source codes which must run through PRECOMP begin with 'p'. The output from PRECOMP begins with 'f'.
2. Source codes written for CFT begin with 'f'.
3. Source codes written for CAL (Cray Assembly Language) begin with 'c'.
4. The binary output from CFT begins with 'b'.
5. Executable codes and libraries have no special beginning character.
6. Version numbers are indicated by a single digit for the last character of the name.

4.1 GENERALIZED PLOTTING PACKAGE

The code *mmxplt1* forms the heart of this data reduction system. Figure 6 schematically indicates the flow of data from the experiment to the final hardcopy plots.

mmxplt1 can plot either raw data produced on the S-100 system by DIGIT.CMD, or reduced data which conforms to the standard indicated in Section 4.3. The input file for *mmxplt1* defaults to *mmxin*, and a sample input file resides under directory *.mmxinp*.

The plotting package used by *mmxplt1* is called *DISSPLA*. This package generates an error file with the name *disoutx*, where the letter *x* is replaced by the channel name under which the job was run (i.e., a–e). If no errors occurred, this file will be three lines long and contain some bookkeeping information about how much output was produced. If errors did occur, the file will contain helpful diagnostic information.

The plot file output by *mmxplt1* is a standard FR80 file with the name **f3mmxp0x**. If this file already exists, the subsequent file names are generated in the standard way (e.g., **f3mmxq0x** is the second file generated). The file should be plotted with a netout command of the following form:

```
netout f3mmxp0x size=(10.,10.)
```

4.1.1 MMXIN

This section describes the input file *mmxin* in detail. The philosophy of the formulation of this input file is to keep it as simple as possible. Most of the information about the shot and the resulting plotted output is best handled through the shot log. In this way a permanent record of the shot is maintained, along with the parameters that affect the output.

The input file is in a namelist format. The namelist is divided into blocks by the dollar sign delimiter ('$'). Blocks are read one at a time, and a plot file is generated according to the given input parameters. This procedure repeats until the variable *job* is set to 'end'.

JOB – Execution of *mmxplt1* halts when job = 'end'.

SHOTLOG – The name of the shot log file. If this file is not specified in a given namelist block, the name will default to the first 5 characters of SHOTBEG, with the letters 'log' appended.

SHOTBEG, SHOTEND – These two file names designate the names of the first and last shots to be analyzed. They may have the same name, if data is to be plotted for only one shot.

PRCPLT – This array holds up to 20 file names which contain data in the standard pre-processed data format (see section 4.3).

RAWPLT – This array of channel names denotes the channels for which the raw data will be plotted.

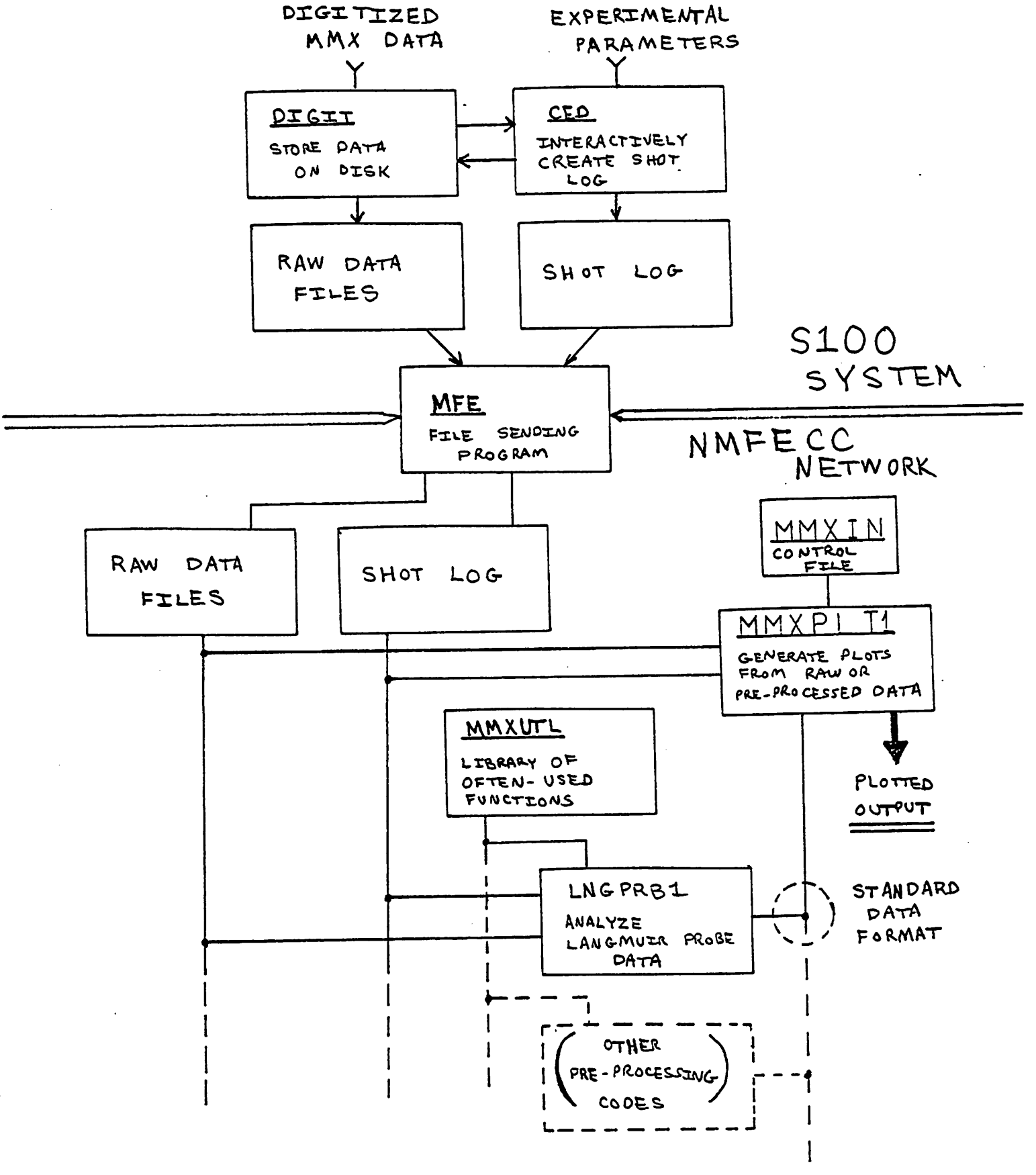


Figure 6: Plotting MMX Data

TIMMIN, TIMMAX – These variables set the plotting window (in μsec) for data to be plotted. These values set the window both for pre-processed data which has $TIMFLG = 1$ (see Section 4.3), and for raw data.

HDRFLAG – Setting this flag to -1 defeats the production of the standard header for each shot. This header normally lists all the digitizer channels used for the shot, and any comments for the shot in the shot log.

MSGFLG – This flag provides for re-direction of error messages. If MSGFLG = 0, error messages go to the terminal, else to the file *mmxerr*.

SAMPLE INPUT:

```
shotlog='aug09'
shotbeg='aug0901'
shotend='aug0913'
rawplt='lpm01s' 'lpm01b' 'iacm23'
timmin=0.
timmax=200.
prcplt='iap'
$
rawplt='lpm01s' 'lpm01b' 'iacm23'
timmin=500.
timmax=1000.
$
job = 'end'
$
```

4.1.2 MMXNEWS

The file *mmxnews* contains a log of new features and changes in this data reduction system. The entries will be dated, and in reverse chronological order. This file is available under user 1235, directory *.mmx*.

4.2 MMX UTILITIES ON MFE

The library file *mmxutl* contains a number of powerful utility routines which greatly simplify reduction of data from the MMX experiment. These routines may be called from any CFT-compiled FORTRAN code by linking to this library in the link statement.

Example:

```
rcft i=fcode,b=bcode,lib=mmxutl / t v
```

NOTE: MMXUTL has been built using the BUILD option `lib=('f','b')`, so that any codes which are linked to MMXUTL do not require explicit linkage to fortlib or baselib.

This means that the source fcode of the previous example may have made calls to fortlib and baselib, and still not require that they be mentioned in the compilation line.

In this section the variable ISLOTP refers to the position of a given channel in the raw data file, rather than the actual slot number of the channel. For example, the first digitizer used might be a Transiac 2008 in slot 12. In this case $ISLOTP = 1$. This usage proves to be the more natural one for data reduction programs.

4.2.1 LOGOPN – Open log file

Calling sequence:

```
call logopn(shotlog)
```

Example call:

```
call logopn('aug03log')
```

This function opens the shotlog and reads it in preparation for subsequent calls to IXTRACT. The routine may be called multiple times with the same file *shotlog* with no ill effect. If the indicated file is already open, the routine simply returns.

4.2.2 IXTRACT - Extract information from log

Calling sequence:

```
ier=ixtract(shot,chan,var,type,value,len)
```

Example call:

```
ier=ixtract('AUG0301','LPM01S','BIAS','REAL',bias,len)
```

Entry:
- SHOT – shot name
- CHAN – channel name or 0
- VAR – variable name or comment #
- TYPE – 'REAL', 'INTEGER', 'ALPHA', OR 'COMMENT'

SHOT/CHAN processing:

SHOT	CHAN	DESIRED ACTION.
shot	*chan*	variable must reside in channel *chan* after shot *shot*
shot	0	first occurrence after shot *shot* returned

Exit:
- VALUE – returned value
- LEN – character length of value, if TYPE='ALPHA' or 'COMMENT'

This function extracts variables and comments from the shot log file. The shot log must have been opened by a call to LOGOPN. NOTES:

- Entry parameters are converted to upper case by the routine.

- Shot log entries of type REAL may one of the following suffixes: M, K, m, u, or p. These suffixes imply multipliers of 1.e6, 1.e3, 1.e-3, 1.e-6, and 1.e-12, respectively. Thus a shot log entry of RTERM = 50K will yield a value for rterm of 50,000. when requested by a call to ixtract.
- Type ALPHA must be enclosed in single quotes in the shot log.
- Type COMMENT is a line beginning with an asterisk ('*').
- COMMENTs can only be associated with shots, not channels. COMMENT's are printed out on the shot header, so pre-processing codes will not normally need to access them.

4.2.3 IPSNFILE – Position a raw data file in preparation for a data read

Calling sequence:

```
ier=ipsnfile(lun,chnnam,islotp)
```

Entry: LUN – Logical unit number of the raw data file to be positioned
CHNNAM – Channel name to which file is to be positioned
ISLOTP – Slot position to which file is to be positioned
HEADER – The common block *header* must be declared (see section 4.6)

This call positions the indicated raw data file for a subsequent read statement of the form

```
      read(lun,1000) (idata(i),i=1,nmpnts)
1000  format (16i4)
```

If $1 \leq ISLOTP \leq 32$ then ISLOTP designates the slot position. Otherwise, CHNNAM is used to locate the proper slot position.

4.2.4 IPLTBNDS – Compute plotting bounds for a channel

Calling sequence:

```
ier=ipltbnds(chnnam,islotp,timmin,timmax,ibegin,iend)
```

Example call:

```
call mmxerr(ipltbnds(name(i),0,timmin,timmax,ibegin,iend)
```

Entry: CHNNAM – Name of the desired data channel
ISLOTP – Slot position of the destired data channel, or 0 if not known
TIMMIN – Minimum time of interest (in μsec)
TIMMAX – Maximum time of interest (in μsec)
HEADER – The common block *header* must be declared (see section 4.6)

Exit: IBEGIN – First data point to fall within the specified interval

IEND Last data point to fall withing the specified interval

If the channel is set to external clock, the currently open shotlog is searched for a variable called EXTINT, which specifies the external sampling interval (in μsec). This routine accounts for pre-trigger samples when calculating IBEGIN and IEND.

4.2.5 ICLCPRM – Calculate digitizer parameters for a channel

Calling sequence:

```
ier=iclcprm(chnnam,islotp,smpint,nmpnts,pttime,nmpt)
```

Entry: CHNNAM – Name of the desired data channel
ISLOTP – Slot position of the destired data channel, or 0 if not known
HEADER – The common block *header* must be declared (see section 4.6)

Exit: ISLOTP – Slot position for the channel, if 0 on entry
SMPINT – Sampling interval for the channel, in μsec
NMPNTS – Number of points for the channel
PTTIME – The time of the first sample, in μsec
NMPT – Number of pre-trigger samples

If $1 \leq ISLOTP \leq 32$ then ISLOTP designates the slot position. Otherwise, CHNNAM is used to locate the proper slot position. If the channel is set to external clock, the currently open shotlog is searched for a variable called EXTINT, which specifies the external sampling interval (in μsec).

4.2.6 ICLCNPT – Calculate number of points in a digitizer channel

Calling format:

```
nmpnts=iclcnpt(islotp)
```

Entry: ISLOTP – Slot position for which NMPNTS is to be calculated
HEADER – The common block *header* must be declared (see section 4.6)

Exit: NMPNTS – The number of points in the indicated channel

If $1 \leq ISLOTP \leq 32$ then ISLOTP designates the slot position. Otherwise, CHNNAM is used to locate the proper slot position.

4.2.7 ICOMPARE – Compare digitizer parameters

Calling format:

```
ier=icompare(chan1,chan2,...)
```
Example call:
```
ier=icompare('LPM01S','LPM01B','LPM01T')
```
Entry: CHANx – Names of channels to be compared. As many as 8 names may be specified.
HEADER – The common block *header* must be declared (see section 4.6)

Exit: IER – A zero is returned if the channels to be compared have the same sampling interval, pre-trigger setting, and number of digitization points. Otherwise the appropriate error number is returned (see section 4.7)

This routine assists in identifying errors in the experimental setup. For example, suppose one expects 4 digitizer channels to contain Langmuir probe data with the same settings. It is wise to call this routine prior to beginning data analysis in order to ensure that the settings are indeed identical for all 4 channels.

4.2.8 NAMEDCD – Decode variable or channel name

Calling sequence:
```
ier=namedcd(chnnam,namtyp,lstrng,chncod,len)
```
Example call:
```
call mmxerr(namedcd(name(i),'CHAN',lstrng,chncod,len)
```

Entry: CHNNAM – Name of the desired data channel
NAMTYP – Type of name to be decoded. 'CHAN' indicates a channel name, and 'VAR' indicates a variable name.
HEADER – The common block *header* must be declared (see section 4.6)

Exit: LSTRNG – This 64 character (8 word) field is returned with a value that may be supplied to MMXPLT1 as a plotting label.
CHNCOD – Type of channel. This field is intended to indicate whether a pre-processing code should process a given data channel. This field can be modified to return whatever users find convenient. The conventions now employed are as indicated in section 4.5.
LEN – Length of LSTRNG, in characters.

The usage of 'VAR' is only partially implemented. Eventually, variables such as TPVOLT will be converted into θ-Pinch Voltage when plotted by MMXPLT1.

4.2.9 CNVDATE – Convert numeric date to expanded format

Calling sequence:
```
call cnvdate(datim,date,len)
```

Entry: DATIM – Integer array containing the date in standard MMX format (i.e., Year, Month, Day)

Exit: DATE – 3 word integer array which will contain the converted date
LEN – Number of characters returned in DATE

Date is returned in an "expanded" format. For example, if the first three values of DATIM are 6, 1, and 59, then DATE will return as 'June 1, 1959', and LEN will be set to 12. This routine is used by MMXPLT1 to expand the date and print it on the header. It is included here with the thought that others may find it useful.

4.2.10 INCFIL – Increment file name

Calling sequence:

```
call incfil(filename)
```

Example call:

```
call incfil(shot)
```

This subroutine is for incrementing the standard MMX file name. The 6th and 7th digits of the file name *shot* are treated as a two digit integer and incremented. Other characters are left unchanged. For example, the file name AUG0301 would return as AUG0302.

4.2.11 AMNMX,IMNMX – Calculate minimum and maximum array values

Calling sequence:

```
call amnmx(array,nmpnts,amin,amax)
call immmx(iarray,nmpnts,imin,imax)
```

Entry: ARRAY, IARRAY – real or integer array to be scanned
NMPNTS – Number of points to be scanned

Exit: AMIN, IMIN – Minimum value of real or integer array
AMAX, IMAX – Maximum value of real or integer array

4.2.12 TOUPPER, TOLOWER – Convert to upper or lower case

Calling sequences:

```
call toupper(ivar,numcnv)
call tolower(ivar,numcnv)
```

Example:

```
call toupper(shot,8)
```

Entry: NUMCNV – Number of characters to convert
IVAR – Variable to be converted

Exit: IVAR – Result of conversion

These routines convert characters from lower to upper case, or *vice versa*. Only alphabetic characters are affected; others are left unchanged. Note that improper specification of the number of characters could cause variables following IVAR in memory to be mistakenly affected. Failure to provide exactly two arguments will cause a call to MMXERR with error number 900, and no conversion will take place. TOLOWER is especially in converting file names to lower case before usage in a FORTLIB "OPEN" call. This is because the CRAY operating system distinguishes between upper- and lower-case letters for file names. Hence the file "aug0301" will not be recognized if the OPEN call is made with the name "AUG0301", which is the normal designation in the shot log.

4.2.13 MMXERR – Handle error conditions

Calling sequences:

```
call mmxerr(numerr)
call mmxerr(numerr,message)
call mmxerr(numerr,message,msglen)
call mmxerr(numerr,message,msglen,routine)
```

Example:

```
call mmxerr(101)
call mmxerr(999,'This is an error$')
call mmxerr(301,namchn,8)
call mmxerr(302,chnnam,8,'calcdns')
```

Entry: NUMERR – Error number (see section 4.7)
MESSAGE – Message to be printed along with the error
MSGLEN – Length of MESSAGE (i.e., the number of characters)
ROUTINE – Routine name where the error occurred

This subroutine aids in handling the most commonly occurring errors in the manipulation of MMX data. If no arguments are passed, an 'invalid error' message is written to the error device and the routine returns. If numerr=0, the routine returns without any message. Otherwise, the routine writes the error to the error device. Providing the fourth argument ROUTINE will greatly assist in the debugging process and is good programming technique.

4.2.14 RDRERR – Re-direct error message output.

Calling sequence:

```
call rdrerr(msglun)
```
Entry: MSGLUN – Logical unit number for re-direction of error messages.

Normally error messages will be written to Logical Unit Number 1, which is assumed to be connected to the terminal. If the user desires error messages to be written to a disk file instead, the disk file should be opened, and its Logical Unit Number used as the argument in a call to RDRERR.

PROGRAMMER'S NOTES

MMXUTL is built from a number of different source codes, which are listed below. Following the name of each source code is a brief description of the function of the routines in the source file, and then the names of the routines which reside in that source file.

1. fmulog – Manipulate the shot log: **logopn, ixtract.**
2. pmudig – Handle digitizer parameters: **ipltbnds, iclcprm, ipsnfile, iclcnpt, icompare.**
3. *fmunmdcd* – Decode channel names into the form required by *mmxplt1:* **namedcd.**
4. *fmumnmx* – Calculate minimum and maximum array values: **amnmx, imnmx.**
5. *cmummxe* – Error handler: **mmxerr, rdrerr**
6. *cmumisc* – Miscellaneous: **cnvdate, incfil, toupper, and tolower.**

Note that the last two source files begin with 'c', thus denoting CAL source files.

4.3 FORMAT FOR PRE-PROCESSED DATA

MMXPLT1 can plot either raw data or pre-processed data files. The pre-processed data file consists of 16 lines of header information followed by a variable number of lines of numbers to be plotted. The sequence is repeated as many times as necessary. The following list enumerates exactly the format of the header. Each item of the list is preceded by the line number(s) associated with that item. FORTRAN-type format designations are given where appropriate.

NOTE ON LABELS:

Each label indicated below is a maximum of 64 characters long, and is terminated with a '$'. Upper case letters are obtained by enclosing the letters in parentheses. For fancier effects, consult the DISSPLA graphics package documentation, Part B, Section 24. Subscripts, superscripts, underlining, Greek letters, etc. are all possibilities here. The escape character used by *mmxplt1* to generate instruction strings is the backslash character ('\').

[1.] SHOT NAME – This field determines which shot MMXPLT1 will associate with this plot.

```
format (a8)
```

[2.] PLOT NAME – Label for the name of the plot to be generated.

[3.] X LABEL – Label for X-axis.

[4.] Y LABEL – Label for Y-axis.

[5.] NMPNTS, X DATA TYPE, XMIN, XMAX, Y DATA TYPE, YMIN, YMAX

```
format (i8,a8,2e12.5,a8,2e12.5)
```

DATA TYPE = 'AUTO', 'DETERMIN', or 'TRUNCATE'
AUTO – Automatically generates axis from min,max values
TRUNCATE – Truncates values outside the domain [min,max]
DETERMIN – Determine appropriate values for the data provided

[6.] WNDWADV, PLTTYP, MRKFLG, ISYM, TIMFLG, FRMADV

```
format (6i2)
```

WNDWADV – flag for advancing the plot window for packed plots
- 0 –do not advance to next window
- 1 –advance to next window

PLTTYP – determines the type of plot to be generated
- 1 – rectangular, packed format (4 plots/page)
- 2 – polar
- 3 – rectangular, large (1 plot/page)

MRKFLG – flag for whether the plot should be labelled with markers
- 1 – connect the points of the plot and use markers
- 0 – connect the points of the plot and do not use markers
- -1 – do not connect the points of the plot and do use markers

ISYM – symbol type for markers, if used
- 1–15–use the marker type ISYM
- (see DISSPLA manual, section 5.3)
- *otherwise* – use default marker

TIMFLG – flag for truncation of x-axis data to conform to values given for TIMMIN, and TIMMAX in *mmxin*

FRMADV – flag for whether the frame should be advanced prior to creating the plot
- 0 – do not advance
- 1 – do advance

[7.–12.] PLOT LABELS – 6 lines of labels. These labels appear to the left of the plot.

[13.–16.] 4 lines reserved for future use

[17.– ?] DATA – The pre-processed data begins at this point. X or θ values are first, followed by Y or R values. If an axis is specified as 'AUTO' (see item 5), then that data is omitted.

```
format (6e11.4)
```

4.4 SHOT LOG INFORMATION

A facility has been prepared for the use of a "shot log" in conjunction with the existing data acquisition system for the Multiple Mirror Experiment. This log is to be prepared as the shots are taken, and information entered into the log as one would enter information in a notebook. The shot log should be named with the extension ".LOG" so that the MFE transmission program will not mistake the file for a raw data file. The shotlog is quite flexible, allowing for comments, and a wide and expandable range of experimental parameters. Extraction of information from this log is made simple by the use of the routines included in the library file *mmxutl* of section 4.2.

The log is a written in a very lax format, making for maximum flexibity for a changing experimental environment. See the shot log at the end of this section for a sample log for the experiment. The following restrictions apply to the format of the log.

- The shots are listed in reverse order, the last shot being first. This ordering greatly simplifies extraction of information from the log. To print out the log in forward order, execute the program *swaplog* and use *netout* to print the resulting file *fwdlog*.
- The shot log should be in upper-case characters.
- Each shot begins with the phrase `SHOT=`*shotname*, where *shotname* is the 8 character name of the raw data file for the shot. This phrase must appear at the beginning of the line.
- Variables associated with a channel follow the phrase `CHAN=`*channame*, where *channame* is the 7 character name of the channel, as designated in the raw data file. This phrase must appear at the beginnning of the line.
- There is never a separation between a variable and the following '=' sign.
- Comments are lines which begin with a '*'. To avoid confusing the routine ixtract, avoid using the '=' sign in a comment, unless preceded by a space.
- Variables which are not re-assigned for a shot are given the most recently stated value when called for by *ixtract*.
- M, K, m, u, n, and p multipliers can be used, as indicated in the documentation for the *ixtract* routine.

SAMPLE SHOT LOG

```
SHOT=AUG1302
CHAN=LPM78S ADB=0.
* Note that all other parameters remain unchanged.
SHOT=AUG1301
* This is the first shot of the day.
CHAN=8MMTIM ADB=14 TRANSIAC=3 OFFSET=-.256
CHAN=8MMPHS GAIN=10.  TRANSIAC=4 OFFSET=-.255 RTERM=15K
    LDTERM=50.  FREQPOT=20.7 AMPLPOT=17.0
CHAN=LPM78S ADB=6.  RTERM=5K BIAS=60.
```

4.5 STANDARD MMX NOTATION

This data reduction system relies heavily on the channel names to designate what information is in a data channel, both in the shot log, and in the utility routines. This method makes for a more readable and less error-prone system. Consistency amongst MMX experimentalists in naming of variables and channels will facilitate the data-taking process, and simplify the sharing of pre-processing codes amongst users. The following table contains a proposed standard that will expanded to accomodate whatever diagnostics are being utilized. In the table, Channel Name is the name to be used for all references to the channel (i.e., in the shot log, in the raw data header, and in the pre-processing codes.) In this column, xxx represents the location of the diagnostic. The first character is either 'T' for a mirror throat, or 'M' for a mirror midplane. Then for midplanes, the next 2 digits indicate the adjacent mirror designations. For throats, only one digit is required to indicate the appropriate mirror throat. The letter 'P' here stands for "port designator". Its value may be N, S, T, or B (North, South, Top, or Bottom). CHNCOD is a code returned by the *mmxutl* routine NAMEDCD.

Channel Name	Description	CHNCOD
LPxxxP	Langmuir Probe	LP
EPxxxP	Emissive Probe	EP
SPxxxP	Swept Probe	SP
XRAYn	X-Ray channel 'n'	XR
FGMxx	Fast Ion Gauge	FG
IABxxxP	ICRF B-Field Probe	IAB
IAPxxx	ICRF Antenna phase	IAP
IAVxxx	ICRF Antenna voltage	IAV
IACxxx	ICRF Antenna current	IAC
8MMTIME	8mm Interferometer timing	8MM
8MMPxxx	8mm Interferometer phase	8MM

The following table summarizes the notation for shot names.

Character position(s)	Value	Description
1–3	JAN ··· DEC	First 3 letters of the month
4–5	01–31	Two digit day of the month
6–7	00–99	Two digit shot number
6–8	LOG	Shotlog for the indicated date
8	(*blank*)	Plasma shot
8	F	Vacuum shot with fields
8	N	Null shot (no plasma or fields)
8	R	Reference shot
8	C	Calibration shot

Note that this convention does not distinguish between years, but that information is in the raw data header, so this should not be a problem.

A list of suggested variable names for the shot log is included here to help provide notation consistency. Most are just suggestions, but it is necessary to use the name EXTINT if you want the mmxutl routines to access properly the external sampling interval for a channel.

VARIABLE	DESCRIPTION
EXTINT	External sampling interval (in μsec)
RTERM	Termination resistance
RCM	Radius of probe tip location
BIAS	Bias voltage for Langmuir probe
ADB	Attenuation in dB
AMP	10X amplifier number
CABLE	Cable designator for channel (e.g. 'L1')
LINEDRVR	Line driver number
OFFSET	Digitizer offset

4.6 STANDARD RAW DATA HEADER

The PRECOMP processor is a great programming aid, reducing both programming development time and debugging time. See the program *plngprb1* for an example of the usage of cliches. The file *header* gives a standard header cliche which may be used by PRECOMP. This header may be included in the program by a text header. Alternatively, PRECOMP will automatically search for a file named *header* when it comes to a *use* directive in the FORTRAN source file, if no cliche named header is present in the source file. The inclusion of the *header* common block is necessary for a program to execute properly the following *mmxutl* routines: *ipsnfile*, *iclcprm*, *ipltbnds*, and *iclcnpt*.

```
c     STANDARD RAW DATA HEADER
      cliche header
      integer datim(6),filnam,filext,name(32),slot(32),smpcod(32),
     $ reclen(32),pretrg(32),devtyp(32),crate(32),reserv(64)
```

```
      common/header/
     $ datim,filnam,filext,name,slot,smpcod,
     $ reclen,pretrg,devtyp,crate,reserv
      endcliche header
```

The header can be read by the following FORTRAN code.

```
c     Assume unit lun has already opened.
      read (lun,1000) datim,filnam,filext,name,
     $ slot,smpcod,reclen,pretrg,devtyp,crate,reserv
1000  format (6i2,/,a8,a3,/,4(8a8,/),7(32i2,/),32i2)
```

4.7 STANDARD ERROR MESSAGES

Many of the routines in the *mmxutl* library are functions which return an error code *ier* (see, for example, ICLCPRM). If the error code is 0, the function executed normally. Otherwise an error code is returned according to the following list. The error code can be displayed by the user code by a call to *mmxerr*.

100 Control file not found.

101 Data file not found.

102 Cannot open log file.

110 Data file read error.

112 Shot log read error.

122 Shot log too long.

132 Cannot close log file.

SHOT LOG ERRORS

200 Shot log not open.

211 Invalid shot specification.

212 Invalid channel specification.

213 Invalid variable specification.

220 Shot log not found.

221 Variable not found for channel.

224 Comment not found for shot.

230 Multiple declaration of file name.

231 Multiple declaration of channel name.

232 Multiple declaration of variable.

250 Variable conversion error.

251 Invalid variable type request.

252 Possible garbage after variable.

DIGITIZER PARAMETER ERRORS

300 Non-existent slot number.

301 External clock not specified.

302 Channel not found in shot.

303 Unrecognized device type.

304 Too many points in channel.

305 Invalid slot position.

ERRORS FROM ICOMPARE

320 Sampling intervals don't match.

321 Record lengths don't match.

322 Pre-trigger samples don't match.

323 Devices don't match.

ERRORS FROM NAMEDCD

400 Invalid namtyp for namedcd.

410 Channel name not recognized.

411 Variable name not recognized.

415 Invalid axial position designation.

420 Invalid port designation.

900 Wrong number of arguments

PROGRAM-DEFINABLE ERROR MESSAGE

999

ACKNOWLEDGEMENTS

This work was supported by DOE Contract No. DE-ATO3-76ET53059. Thanks to H. Meuth, M. Liebermann, R. Mett, and A. Lichtenberg for useful discussions on the structure of this system.