FILE

Institute of
Oceanographic Sciences
Deacon Laboratory

INTERNAL DOCUMENT No. 352

Sonic Buoy - GCAT Raw Data Logger handbook

C H Clayson

1995

Natural Environment Research Council

INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

INTERNAL DOCUMENT No. 352

Sonic Buoy - GCAT Raw Data Logger handbook

C H Clayson

1995

Wormley
Godalming
Surrey GU8 5UB UK
Tel +44-(0)428 684141
Telex 858833 OCEANS G
Telefax +44-(0)428 683066

DOCUMENT DATA SHEET

AUTHOR	PUBLICATION DATE
CLAYSON, C H	1995

TITLE

Sonic Buoy - GCAT Raw Data Logger handbook.

REFERENCE

Institute of Oceanographic Sciences Deacon Laboratory, Internal Document, No. 352, 67pp.
(Unpublished manuscript)

ABSTRACT

The GCAT Raw Data Logger was developed as part of the Sonic Buoy development program; it was required as an on-board partial back-up system to the (previously untried) VHF Radio Telemetry System for obtaining raw sonic anemometer data.

It monitors the sonic anemometer transmit and receive lines to determine the start and end of the sections of data used by the Sonic Processor and logs a single record to a 4 Mbyte Flash EEPROM PCMCIA card at intervals of 48 hours.

This document describes in detail the design and operation of the GCAT Raw Data Logger; it is intended to serve the combined purposes of documenting and design and acting as a guide to operating the system and recovering the data.

KEYWORDS

ISSUING ORGANISATION

Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB. UK.

Director: Colin Summerhayes DSc

Telephone Wormley (0428) 684141
Telex 858833 OCEANS G.
Facsimile (0428) 683066

Copies of this report are available from: ***The Library,*** *PRICE* *£0.00*

Index

1. INTRODUCTION

The Sonic Raw Data Logger is a PC-based processing system, using DSP Designs Ltd. GCAT 3000 processor board and GCAT 2000 peripherals board, mounted on a motherboard, BMPPROC2, of IOSDL design. The system is mounted within a diecast box which also contains a +12V power supply for the Radio Modem.

The Sonic Raw Data Logger is designed to acquire a 10 minute sample of raw Sonic anemometer data at approximately noon on odd-numbered (Julian) days; the anemometer sampling is controlled by the Sonic Processor, running FFTC2 and FASTCOM software. The Raw Data Logger sample is synchronised with a 10 minute record period of the Sonic Processor. The data are stored on a 4 Mbyte Series 1 PCMCIA Flash Card in the standard FASTCOM raw data file format.

2. FUNCTIONAL DESCRIPTION

The main functions of the Sonic Raw Data Logger are as follows:

a) to idle for a 2 day period until just before noon (if the first time after boot up, wait until the first odd Julian day at least 2 days after boot up)

b) to start logging anemometer raw data to the flash card when the first transmit command has been sent to the anemometer after a change to prompted mode

c) to stop logging data when the change is made back to unprompted mode

d) to repeat functions a) to c) until the flash card is full

The above functions are achieved by the application RAWLOG.EXE which is held in EPROM (ROM Disk drive) on the GCAT 3000 board. The ROM Disk also holds DOS version 5.0, AUTOEXEC.BAT and CONFIG.SYS files, and the application SETTIME.EXE. The last application is run upon boot-up before the main application RAWLOG, allowing setting of the hardware Real Time Clock via the COM1 port.; this is described in more detail in Section 3.2, below. Anemometer data are received via the GCAT 3000 COM2 port, anemometer commands are received via the GCAT 2000 COM1 port.

3. SOFTWARE

3.1 Overview

The Sonic Raw Data Logger software is embedded in the GCAT 3000 in a 512k EPROM; this contains MS-DOS 5.0 system files, COMMAND.COM, AUTOEXEC.BAT and CONFIG.SYS, and the applications SETTIME.EXE and RAWLOG.EXE. The EPROM also includes a suitable ROMDISK.DRV driver (the DSP-supplied RD_512_T.DRV) and BIOS (the DSP-supplied 124011.B02). These make the ROM Disk the A: drive (the boot drive) and the B: drive is a PCMCIA drive (although we address the PCMCIA directly by memory mapping, in practice);

there is no C: drive. This version of the BIOS is used to prevent error messages from occurring during boot up, due to the lack of a floppy drive. During development a BIOS was used having A: as the floppy drive, with the software under development on the floppy.

The process for programming the EPROM is described in the DSP Designs Ltd. document "Instructions for producing a ROM Disk for the GCAT-3000".

After boot up, the application SETTIME.EXE is run; this allows synchronisation of the GCAT clock with the clock of an external PC, running the BASIC program SONTIM.BAS and with its COM1 port connected to the GCAT COM1 port. This external PC is normally a battery-powered Husky Hunter 16 (running GWBASIC under DOS).

After the completion or time-out of the application SETTIME, the application RAWLOG.EXE runs; this is the main data acquisition control program with the functions described above. The application RAWLOG remains running continuously until terminated by a key press, a manual reset, or by a system failure. A system failure, such as a processor crash, will result in the watch dog timer rebooting the system.

3.2 SETTIME - Application for Clock Synchronisation

The application is built from the object files SETTIME.OBJ and SETTIM.OBJ. The former is produced by compiling the 'C' code SETTIME.C; the latter is produced by assembling the assembly code SETTIM.ASM. The library SLIBCE.LIB is used when linking. A listing of the source code is given in Appendix A.

When the application is run, the message "Date: DD/MM/YY Time: HH:mm:SSQ" is prepared, where

YY is Year, e.g. (19)93
MM is Month (01 - 12)
DD is Day of the Month (01 - 31)
HH is Hour (00 - 23)
mm is Minute (00 - 59)
SS is Second (00 - 59)
and Q is a terminator. The date and time are derived from the system clock.

The application then outputs the Date/Time message via the COM1 port. (on the GCAT 2000 board); the port is set up for 2400 baud (8 bits data, 1 stop bit, no parity). The application then waits for a Date/Time message terminated by a line feed (character 10) from the external PC (if present). If none is received within at set interval, the application times out. Otherwise, the external PC's Date/Time message is decoded and used to set the GCAT's Real Time and system clocks, using DOS DATE and TIME calls. The application then outputs a message in the above format (using the received Date/Time) to the external PC via the COM1 port.

Note that the SETTIME application is only effective if the GCAT RAM has previously been set up by entering the time/date in the SET UP screen upon power up; SET UP is entered by pressing the F2 key repeatedly during boot up. This makes the connection of a keyboard and VDU essential when first powering up the system. Thereafter SETTIME can be used to alter the time/date by re-booting, using the manual reset push-button, with an external PC running SONTIM.BAS attached to the COM1 port.

This version (2) of the application SETTIME is specific to the GCAT system, although a similar application (but using the COM2 port) has been produced for the DSP ECAT system.

3.3 RAWLOG - Application for Control of Sonic Raw Data Logging

The application is built from the object files RAWLOG.OBJ and FLASH5.OBJ; the former is produced by compiling the 'C' source code RAWLOG.C; the latter is produced by assembling the assembly code FLASH5.ASM, this contains functions used for writing to the PCMCIA Flash Card. The library SLIBCE.LIB is used when linking. The commands for carrying out the above processes are:

```
masm /MX flash5;
```

to produce the object code FLASH5.OBJ, followed by:

```
nmake raw
```

where RAW is the make file, consisting of the following lines:

```
rawlog.exe : rawlog.c flash5.obj

    QCL /AS /Zr /c rawlog.c

    LINK /M /ST:8000 rawlog flash5, rawlog.exe,,slibce.lib
```

A listing of the C source code is given in Appendix B.1 and the assembly code is given in Appendix B.2

When the application is run, the following initialisation steps are carried out:

the COM ports are set up

the time zone is set to GMT

a check is made for the presence of the Flash Card and, if present, the last Flash Card directory entry is read and pointers are initialised

the Julian day number for the first record is calculated

a number of flags are initialised

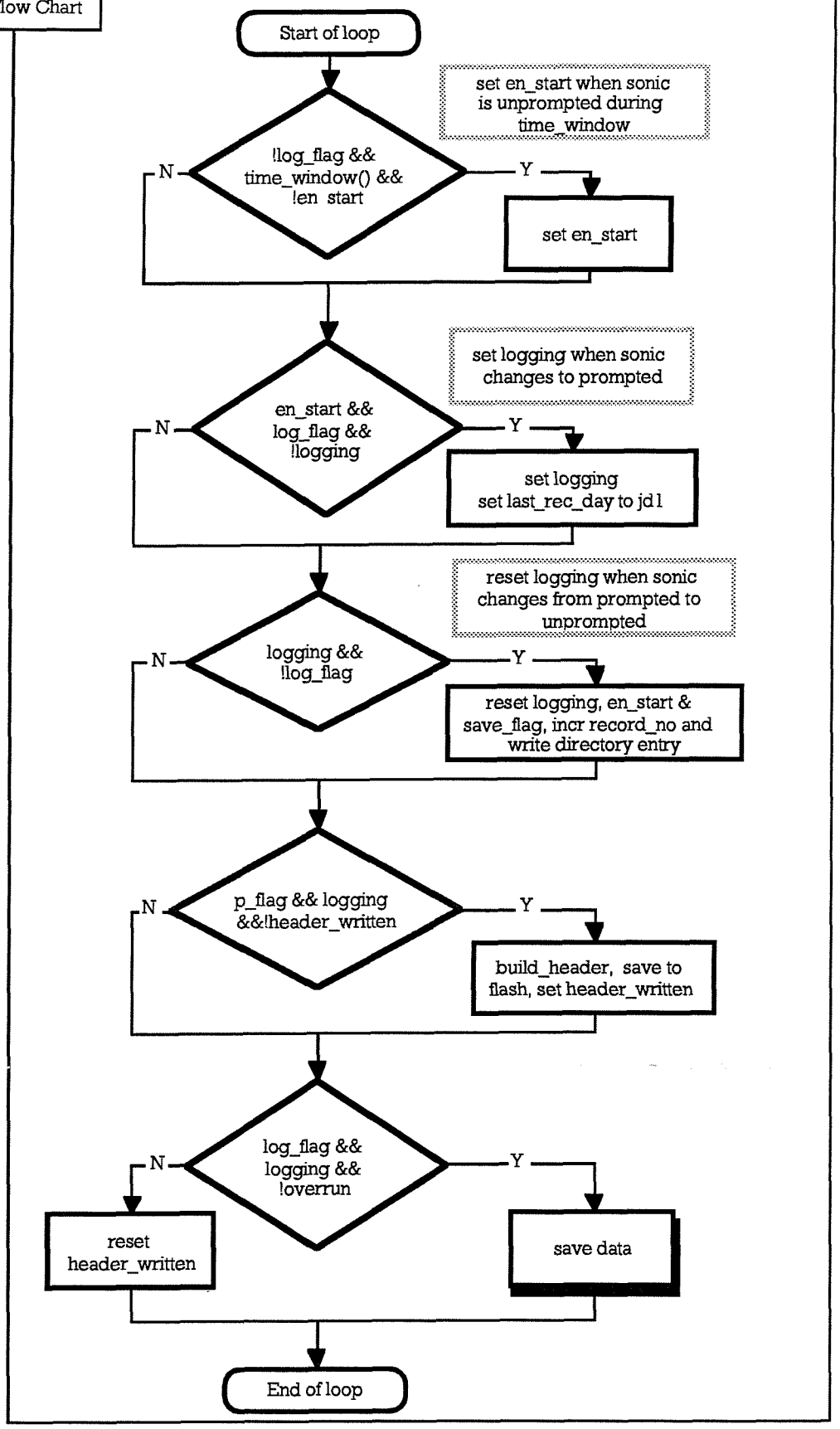
A continuous loop is then entered, this loop will terminate if no flash card space is available or if a key is pressed. In this loop:

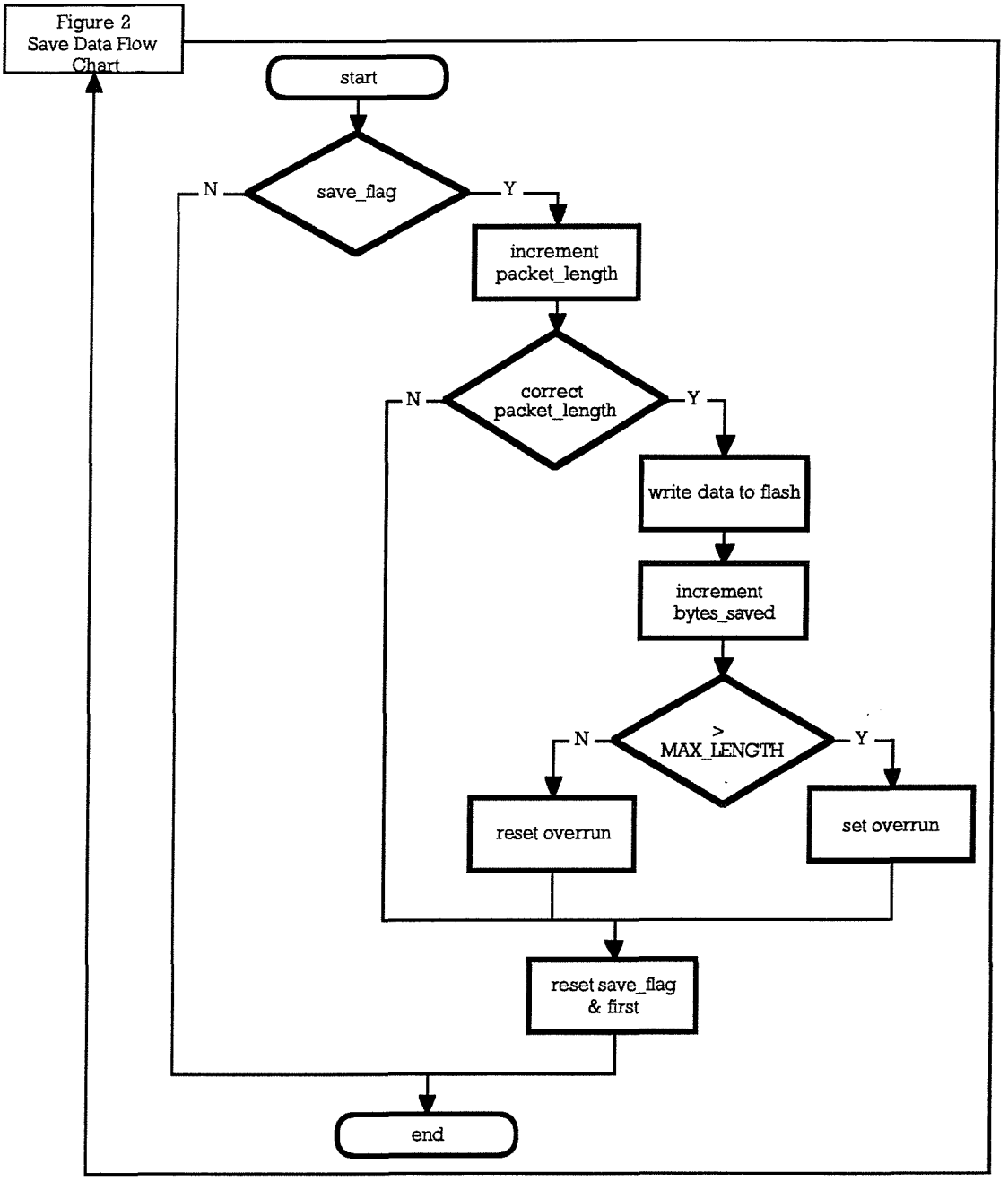
the clock is read and if the seconds count has changed, the watch dog circuit is triggered by pulsing the speaker (this will normally cause an audible 1 second "tick")

a check is made whether the date and time lie within the window for a new "record", i.e. the day is correct (as defined in Section 2, above) and the time is in the range 11:55 and 12:09

if the above time window is fulfilled and the anemometer Prompted (2 Ps) and Transmit Block (2 Ts) commands are detected, logging of received data commences and continues until the Unprompted command (2 Us) is received. The logic is rather more complicated than this, to cater for eventualities; the full logic is shown in the flow diagrams, Figures 1 and 2. These should be read in conjunction with the RAWLOG.C source code in Appendix B.1.

Figure 1
Loop Flow Chart





The watchdog trigger is inhibited during the data acquisition period so that, if the end of record (Unprompted command) is not detected, the data collection will be terminated by a re-boot.

4. HARDWARE

4.1 General

The GCAT 2000 and 3000 boards are mounted on the BMPPROC2 motherboard in a sealed diecast aluminium alloy box. A'Lemo connector, GR1, supplies 24V dc to the dc-dc converters on the motherboard. The anemometer RS232 Tx and Rx lines (optically isolated) for the raw data logger and for the radio modem are input to the box via a Lemo connector, GR2, which

also carries the +5V supplies for the opto-isolators; the Tx and Rx lines for the radio modem are chained through to a similar Lemo, GR3, for connection to the radio modem (see Section 5).

A general assembly drawing and parts list are given in Appendix C.

4.2 Circuit Descriptions

4.2.1 BMPPROC2 Motherboard

The BMPPROC2 motherboard is a general purpose board design which is only part filled for this application. An on-board DC-DC converter produces a +5 Volt stabilised supply at up to 1 Amp from the (nominally) 24 volt input from the battery distribution system (DC-DC Converter Box). This supply is conservatively rated for the Raw Logger system, even when the keyboard is plugged in. The board includes the standard IOSDL watchdog circuit, as developed for the 1802 Microboard System; the time-out period is selectable by jumper on a pin header. The watchdog can also be disabled from resetting the GCAT by removing a jumper.

The board includes a 12V 800mA supply, not shown in the circuit diagram, for the radio modem.

The circuit diagram, PCB tracking and silk screen plots and a parts list are given in Appendix D.

4.2.2 GCAT Boards

The GCAT 3000 and 2000 boards are standard items, but with the applications software in a ROM Disk (512k EPROM, type 27C040-10). The processor runs at 7.2 MHz (determined by the version of the BIOS included in the EPROM.

5. WIRING

The wiring within the unit is relatively simple, consisting of input 24V power connections from Lemo connector GR1 to the BMPPROC2 motherboard, anemometer Tx/Rx signal connections from Lemo GR2 to the motherboard and to Lemo GR3 and, finally 12V power connections from the motherboard to Lemo GR3 for the Radio Modem. The individual connections are listed in the table below.

Lemo	Pin	Function	Destination	Wire Colour
GR1	1	0V	SK1 Pin 2	White/Black
GR1	2	+24V	SK1 Pin 1	Red/Brown
GR2	1	+5V GCAT RAW	SK2 Pin 4	Yellow/Red
GR2	2	Sonic Tx	SK1 Pin 3	Red/Green
GR2	3	Sonic Rx	SK1 Pin 4	Orange/Brown

GR2	4	0V GCAT RAW I/P	SK2 Pin 3	White/Red
GR2	5	+5V HF RAW I/P	SK2 Pin 4	Yellow
GR2	6	Sonic Tx	GR3 Pin 6	Red
GR2	7	Sonic Rx	GR3 Pin 7	Orange
GR2	8	0V HF RAW I/P	SK2 Pin 3	White
GR3	1	N/C		
GR3	2	N/C		
GR3	3	N/C		
GR3	4	N/C		
GR3	5	+12V HF RAW O/P	SK2 Pin 1	Red/Brown
GR3	6	Sonic Tx	GR2 Pin 6	Red/
GR3	7	Sonic Rx	GR2 Pin 7	Orange/
GR3	8	0V HF RAW I/P	SK2 Pin 2	White/Brown

6. OPERATIONAL

The Radio Modem can be disabled, if required, by unplugging the orange plug-in terminal block leading to the Lemo connector GR3 or by directly unplugging the cable to the Radio Modem.

6.1 Procedures to power up system and set in the correct time

The Sonic Raw Data Logger and Radio Modem systems are both powered via the same cable to this unit; it is not possible to power up the Radio Modem without powering up the Raw Data Logger, unless a separate supply/cable are used.

Plug in a suitable keyboard (set for XT PC and NOT AT) and a suitable VDU (with TTL RGB interface and NOT analogue; this may require some adjustment of the setting switches on the keyboard and VDU). Plug in the (orange) PCB connector SK1 to PL1 on the motherboard.

Power up the DC-DC Converter Box from a 24V supply or battery pack and plug in the cable from the DC-DC Converter Box to Lemo GR1. The GCAT should bleep and the "DSP Designs .....etc." message should be displayed on the VDU. Keep pressing the keyboard F2 key as the memory check is made and the machine should then run its "SET-UP" routine, displaying a configuration screen.

The time must then be entered by using the ⇐ and ⇒ arrow keys to highlight the Hours, Minutes, Seconds, Year, Month and Day of the month positions on this screen and entering the required values. In the case of the Month, use the (coloured) + and - keys in the numerical keypad area to adjust the months (these keys can also be used to adjust the other entries, if desired). When the required settings have been entered, pressing the F10 key will,

simultaneously, exit from the set-up and enter the set time and date into the GCAT Real Time Clock. Note that, if the highlight remains on the last parameter altered, pressing F10 may not have any effect, so always move the highlight to another parameter after setting the last alteration. For exact time setting, move the highlight from the Seconds setting at exactly the time which has been entered on the screen (down to the last second). Do not take too long over the set up process, or the watchdog timer (if enabled by the jumper) may re-boot the system.

IMPORTANT Note that, if the set-up process is not carried out as described above, subsequent use of an external PC or Husky, running SONTIM.BAS, will NOT set the Real Time Clock correctly when the application SETTIME runs after a re-boot.

The boot up process will then continue with the SETTIME application being run; this is followed, a short interval later, by the NEWFORM application.

If it is necessary to correct the clock time by use of an external PC or Husky, running SONTIM.BAS, carry out the following steps:

disconnect the IDC ribbon cable connector from BMPPROC2 H1 (COM1) - this runs to the 8 way Lemo GR2

plug the special ribbon cable, labelled "Husky to Formatter", into the Husky or PC 25 way COM1 port (use a 25 to 9 way adaptor if necessary) and into the H1 (COM1 port) connector

Set the PC Date/Time, using the DOS TIME and DATE commands, run the program SONTIM.BAS under GWBasic or QBasic and wait for the "Ready" prompt - this involves the following steps for the Husky:

press the red PWR key to turn the machine on

at the C:\ prompt, enter DATE

- the machine then displays its current date which can be accepted, by pressing RETURN, or modified by keying in a new date with the same format and then pressing RETURN

enter TIME

- the machine then displays its current time which can be accepted, by pressing RETURN, or modified by keying in a new time with the same format and then pressing RETURN

enter GWBASIC

enter LOAD "SONTIM"

enter CLS

enter RUN

wait for "READY FOR DATA" to appear at the top of the screen

press the reset button (labelled RESET) next to the VDU connector on the Raw Data Logger BMPPROC2 motherboard; this will cause a re-boot. When the SETTIME application runs on the GCAT, the message

Date: DD/MM/YY Time: HH:mm:SS

should appear on the PC/Husky display, where:

DD = Day of the month (0 - 31)

MM = Month (1 - 12)

YY = Year, e.g. (19)93

HH = Hour (00 - 23)

mm = Minutes (00 - 59)

SS = Seconds (00 - 59)

- the displayed values being for the initial GCAT Date/Time.

This should be followed shortly by another message of the same format, showing the new time set in to the GCAT from a similar format message sent from the PC/Husky to the GCAT. The GCAT will, after a short pause, run the RAWLOG application.

Remove the ribbon cable from the GCAT COM1 port H1 and reconnect the ribbon cable from Lemo GR2. Disconnect the VDU and keyboard connectors from the motherboard.

6.2 Erasure of the FlashCard prior to use in the GCAT PCMCIA socket

Although the application RAWLOG will examine the FlashCard when it runs (see program description, above) and will append data to any existing entries, it is best to start any prolonged logging session with an erased card. There are two ways in which this may be achieved. The first is to use the Thincard PCMCIA drive and software, installed in a PC. For example, using this with the Tandon 386 S3869, insert the FlashCard in the drive slot and enter:

c:

cd c:\thincard

er

This runs the batch file ER.BAT, which simply contains

tcerase -card IMC004 e:

This will erase the complete FlashCard; NB there are no precautionary checks before erasure commences. Note that the FlashCard drive has been defined as the E: drive in the THINCARD installation process.

The card can also be erased, starting from a base address by including -base address in the above command (see also the THINCARD User Guide).

Alternatively, one can run the IOSDL application FLASH2.EXE in the GCAT development system. To do this:

connect the development system to a keyboard (XT PC - type and NOT AT-type), a RGB TTL VDU and a +5V 2A supply

insert a bootable disk containing the FLASH2.EXE application

switch on the +5V supply to boot up the system

run FLASH2.EXE by entering FLASH2 at the A:\ prompt, the VDU will then display

Erase Card? <Y/N> (press y or Y)

Enter Start Chip and Finish Chip (0-15) (separated by comma):

(enter number of chips to be erased, separated by a comma, e.g. 0,4)

the required chips will then be erased; this takes a while, during which progress messages will be displayed on the VDU. Note that the directory is in chip 0 and data are in chips 1 - 15 inclusive (256k per chip for the 4 Mbyte IMC004 FlashCard)

The partial erasure allowed by FLASH2 is useful when a card has only been used for a short test, e.g. when only chips 0 and 1 need erasure; this can save a few minutes and is better for the card than a total erasure.

6.3 Recovery of data from the FlashCard

At present, this can only be done via the THINCARD drive installed in the Tandon or another PC. Insert the FlashCard in the THINCARD drive slot and then enter

```
c:
cd c:\thincard
t
```

This runs the batch file T.BAT, which contains:

```
tcread -size 0x400000 e: test
read test
```

The whole card is read into a 4 Mbyte file c:\thincard\test and the application READ is then run to allow examination of this file. It is obviously necessary to ensure that space is available for a file of this length on the hard disk before commencing (or that an existing file TEST exists in the c:\thincard directory and that the contents of this file are no longer required). The application READ allows examination of the file TEST, 256 bytes at a time. After the file has been examined, it can be copied to another directory or drive, under an informative name.

Since a 4 Mbyte file is unwieldy for some purposes, an application was written to allow it to be split into four 1 Mbyte files. This application is called 4MTO1M.EXE The resulting files are suffiexed .1MG, .2MG, .3MG, .4MG

Data are subsequently recovered from the file TEST by reading each (sequential) directory entry and using the contents to find the related file of data. Software to decompose the entire contents of a TEST file into a number of individual FASTCOM-format files has yet to be written, but would be quite straightforward.

7. SPECIFICATION

7.1 Supplies

The Sonic Raw Data Logger requires a 24 Volt supply at 60 mA

The Radio Modem requires a 24 Volt supply at approximately 105 mA average

7.2 Power Consumption

The consumption including the DC-DC converters is typically 1.45 Watts at a primary bus supply voltage of +24 Volts, this includes the quiescent consumption of the Radio Modem DC-DC converter, with the Radio Modem disconnected.

7.3 Data Storage and Output

The raw Sonic data are stored on a Series 1 PCMCIA Flash Card in FASTCOM-format as described in Appendix E.

The application outputs diagnostic data to a VDU, if connected, during its operation. When the application is run, diagnostic information regarding the Flash Card Status is produced; the most likely message of any importance is:

**********Flash Card not inserted**********

If this appears, insert the Flash Card and re-boot by pressing the reset push-button.

Other (unlikely) catastrophic error messages are:

Exiting program, COMS error

- if this appears, there was a problem in initialising the COM ports.

Error in setting TZ

- if this appears, there was an error in setting the Time Zone (highly unlikely).

When a record is logged the following sequence of output messages should appear:

Window on

- beginning of time window has occurred

Enabled

- set during unprompted period within time window

H

- when the FASTCOM-format Header is written at the start of the record

R

- when mode changes to unprompted at end of the record

APPENDIX A SOURCE CODE FOR SETTIME

Appendix A.1 C Source Code

```
/*********************************SETTIME.C*****************************\
        Version 2.0 for GCAT (includes port enable function in SETTIM.ASM)
        This program is for inclusion in the autoexec.bat for the
        sonic buoy sonic processor. It allows the dsp processor
        clock to be reset at boot up time by connecting a PC
        running the GWBasic program settime.bas to the COM1 port.
        The DSP time is then set to the PC time.
        If the PC is not connected, this program times out.
        The autoexec then runs the sonic acq/processing prog fftc2.

\***************************************************************************/

#include <stdio.h>
#include<stdlib.h>
#include <dos.h>
#include <bios.h>
#include<string.h>

extern void uart_on(void);
extern void uart_off(void);

main()
{
char rsout[45];
char dum[10];
char stbuf[35];
char dum1[10];

int n;
long loop_ctr;

struct dosdate_t date;
struct dostime_t time;

unsigned status, data;

int ch, ch_hit, port = 0;   /* port = 0 for COM1, =1 for COM2 */
                            /* NB for COM2 set to 1/2 req'd baud rate */

uart_on();                  /* enable GCAT ports */

/* initialise com1 port, 2400 baud, 8bit data, no parity, 1 stop bit */

data = (unsigned) (_COM_CHR8 | _COM_STOP1 | _COM_NOPARITY | _COM_2400);

_bios_serialcom(_COM_INIT, port, data);

_dos_getdate(&date);
_dos_gettime(&time);

    strcpy(rsout, "Date: ");

    itoa(date.day, stbuf, 10);
    strcat(rsout, stbuf);
    strcat(rsout, "/");

    itoa(date.month, stbuf, 10);
    strcat(rsout, stbuf);
```

```
        strcat(rsout, "/");

        itoa(date.year - 1900, stbuf, 10);
        strcat(rsout, stbuf);

        strcat(rsout, " Time: ");

        itoa(time.hour, stbuf, 10);
        strcat(rsout, stbuf);
        strcat(rsout, ":");

        itoa(time.minute, stbuf, 10);
        strcat(rsout, stbuf);
        strcat(rsout, ":");

        itoa(time.second, stbuf, 10);
        strcat(rsout, stbuf);
        strcat(rsout, "Q");

    printf("Sending %s to COM%d\n", rsout, port + 1);
    loop_ctr = 0L;

    for (ch = 0; ch < strlen(rsout); ch++)
        {
        do
            {
            status = 0x2000 & _bios_serialcom(_COM_STATUS, port, 0);
            loop_ctr++;
            }
        while ( (status != 0x2000) && (loop_ctr < 100) );

        if(_bios_serialcom(_COM_SEND, port, rsout[ch]) > 0x7fff)
            {
            exit(0);
            }
        if ((status & 0x8000) == 0x8000)
            {
            printf("RS232 COM%d timed out\n", port + 1);
            break;
            }
        }

    ch = 0;
    loop_ctr = 0L;

    do
        {
        status = 0x100 & _bios_serialcom(_COM_STATUS, port, 0);

        if (status == 0x100)
            {
            ch_hit = 0xff & _bios_serialcom(_COM_RECEIVE, port, 0);
            printf("%c", ch_hit);
            if (ch_hit == 68)           /* capital D */
                {
                ch = 0;
                }
            stbuf[ch] = (char) ch_hit;
            ch++;
            }
        loop_ctr++;
        }
    while ( (ch_hit != 10) && (loop_ctr < 100000L) );
```

```
stbuf[ch] = 0;

printf("\n%s\n", stbuf);

date.month = 10 * (stbuf[5] - 48) + stbuf[6] - 48;
date.day = 10 * (stbuf[8] - 48) + stbuf[9] - 48;
date.year =1900 + 10 * (stbuf[13] - 48) + stbuf[14] - 48;
time.hour = 10 * (stbuf[21] - 48) + stbuf[22] - 48;
time.minute = 10 * (stbuf[24] - 48) + stbuf[25] - 48;
time.second = 10 * (stbuf[27] - 48) + stbuf[28] - 48;

if (loop_ctr < 100000)
    {
    if (_dos_setdate(&date) != 0)
        {
        printf("Error in date set\n");
        }
    if (_dos_settime(&time) != 0)
        {
        printf("Error in time set\n");
        }

    strcpy(rsout, "Date: ");

    itoa(date.day, stbuf, 10);
    strcat(rsout, stbuf);
    strcat(rsout, "/");

    itoa(date.month, stbuf, 10);
    strcat(rsout, stbuf);
    strcat(rsout, "/");

    itoa(date.year - 1900, stbuf, 10);
    strcat(rsout, stbuf);

    strcat(rsout, "  Time: ");

    itoa(time.hour, stbuf, 10);
    strcat(rsout, stbuf);
    strcat(rsout, ":");

    itoa(time.minute, stbuf, 10);
    strcat(rsout, stbuf);
    strcat(rsout, ":");

    itoa(time.second, stbuf, 10);
    strcat(rsout, stbuf);
    strcat(rsout, "Q");

    printf("Sending %s to COM%d\n", rsout, port + 1);

    for (ch = 0; ch < strlen(rsout); ch++)
        {
        do
            {
            status = 0x2000 & _bios_serialcom(_COM_STATUS, port, 0);
            }
        while (status != 0x2000);

        _bios_serialcom(_COM_SEND, port, rsout[ch]);
        if ((status & 0x8000) == 0x8000)
            {
```

```
                printf("RS232 COM%d timed out\n", port + 1);
                break;
                }
            }
        }
uart_off();
}
```

Appendix A.2 Assembly Code

```
;*******************************SETTIM.ASM*********************************
;    Assembly Code functions used to enable GCAT ports
;
;        for use in conjunction with SETTIME.C
;
;        assemble using MASM /MX SETTIM;  to give SETTIM.OBJ
;
;        and then link with SETTIME.OBJ and SLIBCE.LIB to give SETTIME.EXE

;    Author CHC      Date 23/08/1993
;*********************************************************************

enable_uartclock    equ         030CH
disable_uartclock   equ         0304H
enable_rs232        equ         030EH
disable_rs232       equ         0306H


                        ;******** PUBLICS ********

public      _uart_on
public      _uart_off


assume      cs:_TEXT
assume      ds:_DATA

_DATA       segment byte public 'DATA'
            dummy   dw  ?

_DATA       ends

_TEXT       segment word public 'CODE'

; NB this macro is not universal and is only correct for regmem == AX
; See Appendix A of CHIPS Superstate R Interface Guide for general case
; also, see CHIPS Programmer's reference Manual pp 2-12 to 2-19 incl.

LFEAT       MACRO     regmem
            DB      0FEH
            DB      0F8H
            ENDM

; NB this macro is not universal and is only correct for regmem == AL
; See Appendix A of CHIPS Superstate R Interface Guide for general case
; also, see CHIPS Programmer's reference Manual pp 2-12 to 2-19 incl.

STFEAT      MACRO     regmem, sdata
            DB      0FEH
            DB      0F0H
            DB      sdata
```

```
        ENDM


_uart_on    PROC

        push    BP
        mov     BP,SP

        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    SS
        push    DS
        push    ES

        ; first select utility register by setting PS4 low

        mov     AH, 8EH         ; set PS4 address (low byte) to Util Reg low byte
        mov     AL, 00H
        LFEAT   AX

        mov     AH, 8FH         ; set PS4 address (high byte) to Util Reg high byte
                                ; OR'd with 0f8h (enable writes - 16 addresses)
        mov     AL, 0FBH
        LFEAT   AX

        mov     AH, 8CH         ; set PS4 fn selector to "active low chip select"
        mov     AL, 64H
        LFEAT   AX

        ; Utility Register is now selected

        mov     DX, enable_uartclock
        mov     AL, 0           ; (byte written is immaterial)
        out     DX, AL
        mov     DX, enable_rs232
        mov     AL, 0
        out     DX, AL

        mov     AH, 8CH         ; set PS4 to "input" for safety
        mov     AL, 0
        LFEAT   AX

        ; Utility Register is now deselected

        pop     ES
        pop     DS
        pop     SS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX

        mov     SP, BP
        pop     BP
        ret
```

```
_uart_on    ENDP


_uart_off   PROC

            push    BP
            mov     BP,SP

            push    AX
            push    BX
            push    CX
            push    DX
            push    SI
            push    DI
            push    SS
            push    DS
            push    ES

            ; first select utility register by setting PS4 low

            mov     AH, 8EH         ; set PS4 address (low byte) to Util Reg low byte
            mov     AL, 00H
            LFEAT   AX

            mov     AH, 8FH         ; set PS4 address (high byte) to Util Reg high byte
                                    ; OR'd with 0f8h (enable writes - 16 addresses)
            mov     AL, 0FBH
            LFEAT   AX

            mov     AH, 8CH         ; set PS4 fn selector to "active low chip select"
            mov     AL, 64H
            LFEAT   AX

            ; Utility Register is now selected

            mov     DX, disable_uartclock
            mov     AL, 0           ; (byte written is immaterial)
            out     DX, AL
            mov     DX, disable_rs232
            mov     AL, 0
            out     DX, AL

            mov     AH, 8CH         ; set PS4 to "input" for safety
            mov     AL, 0
            LFEAT   AX

            ; Utility Register is now deselected

            pop     ES
            pop     DS
            pop     SS
            pop     DI
            pop     SI
            pop     DX
            pop     CX
            pop     BX
            pop     AX

            mov     SP, BP
            pop     BP
            ret

_uart_off   ENDP
```

```
_TEXT    ends

         end
```

APPENDIX B SOURCE CODE FOR RAWLOG

Appendix B.1 C Source Code

```
/*****************************RAWLOG.C*****************************
* Sonic Buoy Raw Data Logging system, using GCAT + Flashcard
* vn 1.0
* Acquires raw data asynchronously from Sonic Sensor which is
* under control of the ECAT Sonic Processor
*
* Saves raw data to 4 MByte Flashcard in FASTCOM format
* For SWALES
*
* Compile using make file raw (uses flash5.obj)
*
* Author CHC
* Started 17/08/1993
*
*************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<float.h>
#include<math.h>
#include<time.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<bios.h>
#include<dos.h>

#include"coms.c"                    /* Port & UART register definitions */

#define COMMAND_PORT        1   /* COM1 i/p only for monitoring commands
                                   - IRQ4 driven */
#define DATA_PORT           2   /* COM2 i/p only for receiving data
                                   - IRQ3 driven */
#define TRUE                1
#define FALSE               0
#define DISPLAY             1

/* FLASHCARD settings */
#define DIRECTORY_START     0L  /* normally 0L, set higher for dud card */
#define DIR_CHIP            0   /* normally 0, set higher for dud card */
#define DATA_START          262144L     /* normally 262144L,
                                   set higher for dud card */
#define HEADER_LENGTH       44
#define MAX_LENGTH          123144L     /* limit to overrun of record length */

/************************FUNCTION DECLARATIONS************************/
/* Functions in FLASH5.ASM */
```

```
extern int pcmcia_save(unsigned, unsigned, unsigned, char *);
extern void chip_erase(unsigned);
extern unsigned long seek_end(int);             /* for start-up/re-start only */
extern int read_header(unsigned, unsigned);     /* for start-up/re-start only */
extern void progsupply_on(void);
extern void progsupply_off(void);
extern int card_detect(void);
extern void bankswitch_disable(void);

/* Functions in this file */
void readclock(int);
void read_hhmm(void);
void clean_up(void);
int init_coms(void);
int com_init(int, unsigned, unsigned, unsigned);
int ser_putc(int, char *);
int ser_getc(int);
int flash_save(char *, unsigned long, unsigned long);
int directory_entry(unsigned, unsigned, unsigned long, unsigned, char *);
void data_save(int);
void build_header(void);
int time_window(void);
void wdog(void);

/* Interrupt Handlers and addresses of default handlers */
void interrupt far our_irq3_handler(void);
void (interrupt far *old_irq3_handler)();
void interrupt far our_irq4_handler(void);
void (interrupt far *old_irq4_handler)();

/**************GLOBAL VARIABLES******************/
char a[512], display_buffer[80], header[64], header_contents[40], julian[10];
char data_buffer[1024];
int hl, ml, sl, dl, nl, yyl, i, jdl, start_day;
int command_flag = 0, full_flag = 0, log_flag = 0, save_flag = 0;
int flash_full = 0, logging = 0, last_char = 0, en_start = 0;
int packet_length = 0, p_flag = 0, first = 1, overrun = 0;
int last_rec_day = 0, wdog_mask = 0;

unsigned old_ints;
unsigned header_block, header_startptr;
unsigned reclen, record_no = 0, segment, seg_ptr, start_block, start_offset;

unsigned long header_reclength, locn;
unsigned long dir_ptr = 0, fl_ptr, old_fl_ptr, bytes_saved;

main()
{
int header_written, n, s11;
union REGS regs;

/* turn on Flashcard Programming Supply VPP */
progsupply_on();

/* the following CREG gets/writes for test purposes only
. . . . delete down to "start of real stuff" */
regs.h.ah = 0x14;
regs.h.bh = 0x0f;                       /* F8680 UART config */
regs.h.al = 0;
regs.h.bl = 0;                          /* get creg */
int86(0x1f, &regs, &regs);

#if DISPLAY == TRUE
```

```
    {
    printf("CREG 0Fh before init: %x\n\r", regs.h.al);
    }
#endif

/* normally returns 0x0f, i.e. COM2, int active low, enabled */

/* get PC/CHIP and 82C710 Options */
regs.h.ah = 0x08;
regs.h.bl = 0;                          /* return options */
int86(0x1f, &regs, &regs);

#if DISPLAY == TRUE
    {
    printf("PC/CHIP Options: %x\n\r", regs.h.al);
    }
#endif

/* normally returns 0x02, i.e. drive B is PCMCIA */

#if DISPLAY == TRUE
    {
    printf("82C710 Options: %x\n\r", regs.h.ah);
    }
#endif

/* normally returns 0xec, i.e. XT IDE, FDC, par and ser ports enabled


/*************************** real stuff starts here ************************/

/* set up the COM ports */
if (init_coms() == 0)
    {
    #if DISPLAY == TRUE
        {
        printf("Exiting program, COMS error\n\r");
        }
    #endif
    exit(0);
    }

/* need to set timezone to GMT */
if (putenv("TZ=GMT") == -1)
    {
    #if DISPLAY == TRUE
        {
        printf("Error in setting TZ\n\r");
        }
    #endif
    return 0;
    }
tzset();

/* In case of startup due to re-boot or with unerased FlashCard */
header_contents[0] = 255;
header_contents[1] = 0;

n = card_detect() & 0xff;

#if DISPLAY == TRUE
    {
    printf("SDATA 0A: %02x\n\r", n);
```

```
    }
#endif


if (n & 0x0c)                       /* Card Detect lines bits 2&3 should be low */
    {
    #if DISPLAY == TRUE
        {
        printf("***************Flash Card not inserted***************\n\r");
        }
    #endif

    flash_full = 1;
    }
else
    {
    flash_full = 0;
    /* Find last directory entry */
    locn = seek_end(DIR_CHIP);

    #if DISPLAY == TRUE
        {
        printf("Flash dir ptr:%lx\n\r", locn);
        }
    #endif

    segment = (unsigned) (locn >> 16);
    seg_ptr = (unsigned) (locn & 0xffff);
    dir_ptr = locn;
    if (locn == DIRECTORY_START)
        {
        #if DISPLAY == TRUE
            {
            printf("Virgin FlashCard\n\r");
            }
        #endif


        fl_ptr = DATA_START;
        old_fl_ptr = fl_ptr;
        }
    else
        {
        /* Flashcard has data/directory entries, so must adjust for these
            by setting pointers and loading n_saves bins */
        if (seg_ptr == 0)
            {
            seg_ptr = 65504;
            }
        else
            {
            seg_ptr -= 32;
            }

        #if DISPLAY == TRUE
            {
            printf("Last Directory Entry:- Segment %x, Offset %x\n\r",
                                    segment, seg_ptr);
            }
        #endif

        /* Read the directory entry */
        read_header(segment, seg_ptr);      /* result in header_contents[] */
```

```
            strcpy(display_buffer, "");
            for (n = 0; n < 32; n++)
                {
                sprintf(julian, "%02x ", header_contents[n] & 0xff);
                strcat(display_buffer, julian);
                if (n == 15)
                    {
                    strcat(display_buffer, "\n\r");
                    }
                }
            strcat(display_buffer, "\n\r");
            #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
            #endif

            /* Calculate Flash Pointer (fl_ptr ) for 1st free byte on Card */
            header_block = (unsigned) header_contents[8] & 0xff;
            header_startptr = (unsigned) header_contents[9] & 0xff;
            header_startptr += (( (unsigned) header_contents[10] & 0xff) << 8);
            header_reclength = (unsigned long) header_contents[11] & 0xff;
            header_reclength += (( (unsigned long) header_contents[12] & 0xff) << 8L);

            fl_ptr = 65536L * header_block + header_startptr + header_reclength + 65536L;
            old_fl_ptr = fl_ptr;

            #if DISPLAY == TRUE
                {
                printf("Last Record:- Block %x, Offset %x, \
                                    Length %lx\n\rFlash data ptr %lx\n\r",
                    header_block, header_startptr, header_reclength, fl_ptr);
                }
            #endif

            } /* end of else (not a virgin flashcard) */
        } /* end of else (locn not 0x40000) */


    readclock(1);
    if (jd1 < 363)                          /* jd1 runs from 0->364 in non-leap year */
        {
        start_day = jd1 + 2;
        }
    else
        {
        start_day = jd1 - 363;              /* NB Jan 1st -> jd1 = 0 */
        }
    log_flag = 0;
    save_flag = 0;
    header_written = 0;

    /*******************START OF CONTINUOUS LOOP******************/
    while (!kbhit() && (flash_full == 0))
        {
        readclock(0);
        if (s11 != s1)
            {
            wdog();
            s11 = s1;
            }

        if ( !log_flag && time_window() && !en_start )
```

```
/* log_flag set by 2 'P's + 2 'T's, reset by 2 'U's */
    {
    #if DISPLAY == TRUE
        {
        printf("Enabled\n");
        }
    #endif
    en_start = 1;                   /* set en_start when unpr during time_window */
    last_rec_day = jd1;
    wdog_mask = 1;                  /* inhibit wdog trigger until end of record */
    }
if (en_start && log_flag && !logging)
    {
    logging = 1;                    /* set logging when unpr->pr during time_window */
    first = 1;
    bytes_saved = 0;
    }
if (logging && !log_flag)           /* end of prompted data logging */
    {
    #if DISPLAY == TRUE
        {
        printf("R\n");
        }
    #endif
    logging = 0;
    en_start = 0;
    save_flag = 0;
    wdog_mask = 0;
    record_no++;

    /* make directory entry */
    start_block = (unsigned) (old_fl_ptr >> 16);
    start_offset = (unsigned) (old_fl_ptr - (start_block << 16) );
    reclen = (unsigned) (fl_ptr - old_fl_ptr);
    old_fl_ptr = fl_ptr;

    while ( !directory_entry(start_block, start_offset, reclen, record_no, julian)
                        && (dir_ptr < (DIRECTORY_START + 262144L) ) )
        {
        dir_ptr += 32;              /* allow full length of directory entry gap */
        }


    if (dir_ptr >= DIRECTORY_START + 262144L)
        {
        flash_full = 1;
        }
    }

if (p_flag && logging && !header_written)
    {
    build_header();
    #if DISPLAY == TRUE
        {
        printf("H");
        }
    #endif
    data_save(HEADER_LENGTH);
    header_written = 1;
    }


if ( log_flag && logging && !overrun)
```

```
    /* log_flag is set by IRQ4 handler when 2 'P's  + 2 'T's rxd
                and reset by IRQ4 handler when 2 'U's rxd */
        {
        if (save_flag)                  /* set by IRQ3 handler when 2nd EOT byte read */
                                        /* reset when data written to Flashcard */
                                        /* or by reading char other than 2nd EOT byte */
            {
            packet_length++;
            /* printf("%d ", packet_length); */

            if (!(div(packet_length - 6, 10).rem))
                {
                if ( (a[1] = (char) 0x81) && (packet_length < 513) )
                    {
                    for (n = 4; n < packet_length; n++)
                        {
                        data_buffer[n - 4] = a[n];      /* misses out SOT and rec no. */
                        }
                    data_save(packet_length - 6);
                    bytes_saved += (packet_length - 6);
                    if (bytes_saved > MAX_LENGTH)
                        {
                        overrun = 1;
                        }
                    else
                        {
                        overrun = 0;
                        }
                    }
                }

            save_flag = 0;
            /* i = 0; */
            first = 0;
            }
        }
    else
        {
        header_written = 0;
        }
    }
/*********************END OF CONTINUOUS LOOP*********************/

clean_up();
return 0;
}

/*********************START OF FUNCTION DEFINITIONS***************/

/*******************READCLOCK gets system time & date**************/
void readclock(int d_enable)
    {
    struct tm *tmnow;
    time_t tnow;

    time(&tnow);
    tmnow = gmtime(&tnow);
    h1 = tmnow->tm_hour;
    m1 = tmnow->tm_min;
    s1 = tmnow->tm_sec;
    d1 = tmnow->tm_mday;
    n1 = tmnow->tm_mon + 1;
    yy1 = tmnow->tm_year;
```

```
    jd1 = tmnow->tm_yday;
    #if DISPLAY == TRUE
        {
        if (d_enable == 1)
            {
            printf("date %02d/%02d/%02d: time %02d:%02d:%02d\n\r",
                                d1, n1, yy1, h1, m1, s1);
            }
        }
    #endif
    }

/**************** CLEAN_UP resets system for exit ***************/
void clean_up(void)
{
int n;

/* reset UART GPO2s to disable interrupts */
n = inp(COM1_BASE + MODEM_CONTR_REG);
outp(COM1_BASE + MODEM_CONTR_REG, n & 0xf7);
n = inp(COM2_BASE + MODEM_CONTR_REG);
outp(COM2_BASE + MODEM_CONTR_REG, n & 0xf7);

/* reset interrupt enables in UART IERs */
/* NB include COM1 for ARGOS XON detection */

outp(COM1_BASE + INT_ENABLE_REG, 0);
outp(COM2_BASE + INT_ENABLE_REG, 0);

/* read every UART register to clear any interrupts pending */
for (n = 0; n < 7; n++)
    {
    inp(COM1_BASE + n);
    inp(COM2_BASE + n);
    }

/* Restore old interrupt masks */
outp(0x21, old_ints);

/* restore default interrupt handlers */
_disable();
_dos_setvect(INT_NO3, old_irq3_handler);
_dos_setvect(INT_NO4, old_irq4_handler);
_enable();

/* disable memory bank switch registers */
bankswitch_disable();

/* turn off VPP */
progsupply_off();

}


/************ INIT_COMS sets up COMS H/Ware & S/Ware **********/
int init_coms(void)
{
int n;

unsigned imask = IRQ3 & IRQ4;

/****************** Set up baud rate etc ************************/
```

```
/* NB if COM2, set up for 2400 baud rate as xtal is 3.6864 MHz
            if COM1, set up for 4800 baud rate as xtal is 1.8432 MHz */
if (com_init(COMMAND_PORT, BAUD_4800, 0, CHRS_8 | STOP_1 | NOPARITY) == NULL)
    {
    #if DISPLAY == TRUE
        {
        printf("Initialised COM%d Port\n\r", COMMAND_PORT);
        }
    #endif

    }
else
    {
    #if DISPLAY == TRUE
        {
        printf("Failed to initialise COM%d Port\n\r", COMMAND_PORT);
        }
    #endif

    return 0;
    }

if (com_init(DATA_PORT, BAUD_2400, 0, CHRS_8 | STOP_1 | NOPARITY) == NULL)
    {
    #if DISPLAY == TRUE
        {
        printf("Initialised COM%d Port\n\r", DATA_PORT);
        }
    #endif

    }
else
    {
    #if DISPLAY == TRUE
        {
        printf("Failed to initialise COM%d Port\n\r", DATA_PORT);
        }
    #endif

    return 0;
    }

/******************* Now set up interrupt handlers ******************/
outp(0x20, 0x10);
outp(0x21, 0x08);
outp(0x21, 0x10);               /* set to 10 to enable multiple ints from same channel */
outp(0x20, 0x20);

outp(0x20, 0x68);               /* enables special mask mode */

old_ints = inp(0x21) | 0xb8;
/* old_ints = 0xb8; temp ***************************************/

#if DISPLAY == TRUE
    {
    printf("Old Int Mask register Contents: %x\n\r", old_ints);
    }
#endif

n = old_ints & imask;           /* enables IRQ 3 & 4  (ints 11 & 12) */
outp(0x21, n);
n = inp(0x21);
```

```
#if DISPLAY == TRUE
    {
    printf("New Int Mask register Contents: %x\n\r", n);
    }
#endif

/* save existing int handlers */
old_irq3_handler = _dos_getvect(INT_NO3);
old_irq4_handler = _dos_getvect(INT_NO4);

/* load new int handlers */
_disable();
_dos_setvect(INT_NO3, our_irq3_handler);
_dos_setvect(INT_NO4, our_irq4_handler);
_enable();

/* enable interrupts for Rx (not Tx or Modem) in UARTs */
outp(COM1_BASE + INT_ENABLE_REG, RX_DATA_AVAIL_EN | RX_ERR_EN);
outp(COM2_BASE + INT_ENABLE_REG, RX_DATA_AVAIL_EN | RX_ERR_EN);

/* read UART registers to clear any interrupts pending */
for (n = 0; n < 7; n++)
    {
    inp(COM1_BASE + n);
    inp(COM2_BASE + n);
    }

/* set GPO2 to enable required interrupts via PAL to IRQ lines */

outp(COM1_BASE + MODEM_CONTR_REG, 0x08);
outp(COM2_BASE + MODEM_CONTR_REG, 0x08);

return 1;
}


/*************** COM_INIT sets up UARTS for COM Ports **************/
int com_init(int port, unsigned bauds,
                                unsigned int_enable_data, unsigned line_control_data)
    {
    unsigned base_address, n;
    switch(port)
        {
        case 1:
            base_address = COM1_BASE;
            break;
        case 2:
            base_address = COM2_BASE;
            break;
        default:
            return -1;
            break;
        }

    /* set baud rate by loading divisor latches */
    outp(base_address + LINE_CONTROL_REG, DLAB);
    outp(base_address + DIV_LATCH_LSREG, bauds & 0xff);
    outp(base_address + DIV_LATCH_MSREG, (bauds & 0xff00) >> 8);
    /* set word length, start/stop bits, parity */
    outp(base_address + LINE_CONTROL_REG, line_control_data & 0x7f);
    /* set any interrupt criteria */
    outp(base_address + INT_ENABLE_REG, int_enable_data);
    return 0;
```

```
    }

/**********INTERRUPT HANDLER FOR COM1 (Command) INTERRUPT
HANDLING**********/
void interrupt far our_irq4_handler()
    {
    int m, n = 0;
    _enable();

    m = inp(COM1_BASE + INT_IDENT_REG) & 0x07;
    do                          /* added do-while 11/8/93 to stop int latching high */
        {
        switch(inp(COM1_BASE + INT_IDENT_REG) & 0x07)
            {
            case RX_DATA_AVAIL:
                n = inp(COM1_BASE + RX_BUFF_REG);
                break;
            case RX_ERR:
                inp(COM1_BASE + LINE_STATUS_REG);
                n = 253;
                break;
            case MODEM_STATUS:
                inp(COM1_BASE + MODEM_STAT_REG);
                n = 254;
                break;
            case TXHR_EMPTY:
                n = 254;
                break;
            case INT_PENDING:
                n = 255;
                break;
            default:
                n = 255;
                break;
            }
        if (n == 80)
            {
            if (command_flag == 80)          /* P already received */
                {
                p_flag = 1;
                }
            else
                {
                command_flag = 80;
                }
            }
        if ( (n == 84) && (p_flag == 1) )
            {
            if (command_flag == 84)          /* T already received */
                {
                log_flag = 1;
                i = 0;
                }
            else
                {
                command_flag = 84;
                save_flag = 0;
                }
            }
        if (n == 85)
            {
            if (command_flag == 85)          /* U already received */
                {
```

```
                log_flag = 0;
                p_flag = 0;
                }
            else
                {
                command_flag = 85;
                }
            }
        }
    while ( (m = (inp(COM1_BASE + INT_IDENT_REG) & 0x07)) != INT_PENDING);

    outp(0x20, 0x20);                   /* non-specific EOI ? in do-while */
    _chain_intr(old_irq4_handler);      /* other sources of int handled */
    }
/*********END OF INTERRUPT HANDLER FOR COM1 INTERRUPT HANDLING*******/

/****INTERRUPT HANDLER FOR COM2 (Data) INTERRUPT HANDLING****/
void interrupt far our_irq3_handler()
    {
    int m, n = 0;
    _enable();

    m = inp(COM2_BASE + INT_IDENT_REG) & 0x07;
    do                              /* added do-while 11/8/93 to stop int latching high */
        {
        switch(m)
            {
            case RX_DATA_AVAIL:
                n = inp(COM2_BASE + RX_BUFF_REG);
                break;
            case RX_ERR:
                inp(COM2_BASE + LINE_STATUS_REG);
                n = 0x99;
                break;
            case MODEM_STATUS:
                inp(COM2_BASE + MODEM_STAT_REG);
                n = 256;
                break;
            case TXHR_EMPTY:
                n = 256;
                break;
            case INT_PENDING:
                n = 256;
                break;
            default:
                n = 255;
                break;
            }                       /* end of switch(m) */
        if (n < 256)
            {
            if ( (n == 0x81) && (last_char == 0x81) )
                {
                i = 1;
                }
            if ( (n == 0x82) && (last_char == 0x82) )
                {
                save_flag = 1;
                packet_length = i;
                }
            else
                {
                save_flag = 0;
                }
```

```
                a[i] = (char) n;
                last_char = n;
                i++;
                i &= 0x3ff;                     /* restrict for buffer length 1024 */
                }
            else
                {
                a[i] = 0;
                last_char = 0;                  /* if error */
                i++;
                i &= 0xff;
                }

            }
        while ( (m = (inp(COM2_BASE + INT_IDENT_REG) & 0x07)) != INT_PENDING);

        outp(0x20, 0x20);                       /* non-specific EOI 20, 20 */

        _chain_intr(old_irq3_handler);          /* other sources of int handled */
        }
/*********END OF INTERRUPT HANDLER FOR COM2 INTERRUPT HANDLING*******/


/*************** FLASH_SAVE writes data to FLASH EEPROM Card ****************/
int flash_save(char * s_buffer, unsigned long flash_pointr,
                                        unsigned long nbytes)
/* address of 1st byte to be saved, flash pointer (0 - 4 MB)
            and number of bytes to be written to flash */
{
unsigned block, b_ptr;

if (nbytes == 0)
    {
    exit(0);
    }
do
    {
    block = (unsigned) (flash_pointr >> 16);
    b_ptr = (unsigned) (flash_pointr - (block << 16) );

    if (block > 63)
        {
        #if DISPLAY == TRUE
            {
            printf("Out of Storage Space\n\r");
            }
        #endif

        /* exit(0); */
        full_flag = 1;
        return(0);
        }
    if( ( (unsigned long) b_ptr + nbytes) > 65536 )
        {
        if (pcmcia_save((unsigned) (65535 - b_ptr), block, b_ptr, s_buffer) == 0)
            {
            flash_pointr += (unsigned long) (65536 - b_ptr);
            nbytes -= (unsigned long) (65536 - b_ptr);
            s_buffer += (unsigned long) (65536 - b_ptr);
            }
        else
            {
            #if DISPLAY == TRUE
```

```
                {
                printf("Failed\n\r");
                }
            #endif

            return(0);
            }
        }
    else
        {
        if (pcmcia_save((unsigned) nbytes - 1, block, b_ptr, s_buffer) == 0)
            {
            flash_pointr += nbytes;
            nbytes = 0;
            }
        else
            {
            #if DISPLAY == TRUE
                {
                printf("Failed\n\r");
                }
            #endif

            return(0);
            }
        }
    } while (nbytes > 0);

return(1);                          /* returns 1 if OK, 0 if failure */
}


/*************** DIRECTORY_ENTRY creates and writes an entry ***************/
int directory_entry(unsigned start_block, unsigned start_offset,
                    unsigned long reclen, unsigned record_no, char * jul_start)
/* need to change a lot of this */
{
char dir_entry[35];
char *ptr;
char dummy[10];

int ch;

time_t tnow;
struct tm *gmt;


time(&tnow);
gmt = gmtime(&tnow);

strcpy( dir_entry, "v");
sprintf(dummy, "%03d", 1 + gmt->tm_yday);
strcat( dir_entry, dummy);
sprintf(dummy, "%02d", gmt->tm_hour);
strcat( dir_entry, dummy);
sprintf(dummy, "%02d", gmt->tm_min);
strcat( dir_entry, dummy);
dir_entry[8] = (char) (start_block & 0xff);
ptr = (char *) &start_offset;
dir_entry[9] = *ptr++;
dir_entry[10] = *ptr;
ptr = (char *) &reclen;
dir_entry[11] = *ptr++;
```

```
dir_entry[12] = *ptr++;
ptr = (char *) &record_no;
dir_entry[13] = *ptr++;
dir_entry[14] = *ptr;
dir_entry[15] = 0;

for (ch = 0; ch < 16; ch++)
     {
     dir_entry[16 + ch] = dir_entry[ch];
     }

if (flash_save(&dir_entry[0], dir_ptr, 32L) == 1)
     {
     dir_ptr += 32L;
     return 1;
     }
else
     {
     return 0;
     }
}


/****************DATA_SAVE writes data to Flashcard*********************/
void data_save(int data_length)
{
int ch;

while ( !flash_save(data_buffer, fl_ptr, data_length) && (fl_ptr < 4194284L) )
     {
     fl_ptr += data_length;                    /* allow full length to ensure no overwrite */
     }


if (fl_ptr >= 4194284L)
     {
     logging = 0;
     flash_full = 1;
     }
else
     {
     fl_ptr += data_length;
     }

}

/******************BUILD_HEADER creates FASTCOM-type header***************/
void build_header(void)
     {
     readclock(0);
     sprintf(data_buffer,
                    "Mode 1\nAnalog 1\nTime %02d:%02d:%02d Date %02d/%02d/%02d\n",
                         h1, m1, s1, d1, n1, yy1);
     }


/**********TIME_WINDOW returns 1 during selected 1/4 hour every 2 days******/
int time_window(void)
     {
     readclock(0);
     /* allow for 70 days operation (odd only) and end-of-year case */
     if ( ((jd1 >= start_day) || ((jd1 - start_day) < -290)) && div(jd1, 2).rem )
          {
```

```
        if ( (jd1 != last_rec_day)  && !en_start
                    && ( ((h1 == 11) && (m1 > 54)) || ((h1 == 12) && (m1 < 10)) ) )
            {
            #if DISPLAY == TRUE
                {
                printf("Window on\n");
                }
            #endif
            return 1;
            }
        else
            {
            return 0;
            }
        }
    else
        {
        return 0;
        }

    /* following lines for test purposes, may give 2 recs per 3hrs,
    depending on window timing relative to sonic record */
    /*
    if ( !(div(h1, 3).rem) && (m1 < 15) && !en_start)
        {
        #if DISPLAY == TRUE
            {
            printf("Window on\n");
            }
        #endif
        return 1;
        }
    else
        {
        return 0;
        }
    */
    /* down to here */
    }

/************ WDOG sends a beep to speaker to trigger watchdog ***********/
void wdog(void)
    {
    /*  to give a single cycle o/p on spkr */
    unsigned n, status;
    if (!wdog_mask)
        {
        status = inp(0x61);
        outp(0x61, status | 3);          /* speaker on */
        for (n = 0; n < 200; n++);
        status = inp(0x61);
        outp(0x61, status & ~3);         /* speaker off */
        /*  gives a short beep (long enough to trigger wdog) */
        }
    }
```

APPENDIX B.2 Assembly Code FLASH5.ASM

```
; Name   FLASH5.ASM
;
; Function: GCAT - drivers for PCMCIA Flash EEPROM Card
;
; assemble using masm /MX flash5;
;
; developed from DSP code, with extra functions
; uses LFEAT AX instructions which are not recognised by MASM.
; therefore Macro is defined to insert the bytes FE F8
;
; chc IOSDL 1/2/93


; Miscellaneous Equates

exitfn          equ     04ch            ; function code for exit from program
cr              equ     0dh             ; ASCII carriage return
lf              equ     0ah             ; ASCII line feed
EPROM           equ     0f000h          ; address of BIOS EPROM
FLASH           equ     0e000h          ; segment of mapped PCMCIA flash EEPROM

; Utility Register Equates

CSUTIL_BASE     equ     300h            ; default I/O address
VPP_OFF_PORT    equ     CSUTIL_BASE + 05h
VPP_ON_PORT     equ     CSUTIL_BASE + 0dh

; INT 1F Equates

GET_SET_CREG    equ     14h             ; Int 1f function to set/get CREG
SET_CREG        equ     1               ; set CREG
GET_CREG        equ     0

; CREG Equates

PS4_SELECTOR    equ     8ch             ; PS4 function selector
PS4_ALOW        equ     8eh             ; PS4 address low
PS4_AHIGH       equ     8fh             ; PS4 address high

WRITE_16        equ     0f8h            ; enable writes - 16 addresses
SELECT_CS_LOW   equ     64h             ; active low chip select
SELECT_INPUT    equ     0               ; pin is an input


; see Chips and Technologies F8680 PC/CHIP Programmer's Reference Manual
; pp 3-54 to 3-55  for Bank Switch Register programming

BSHI            equ     0afh            ; hi byte BSR for mapped 64k segment
BSHI_VAL        equ     0ch             ; to set to 48MB (CardB)
BSLO            equ     0a3h            ; lo byte BSR for mapped 64k segment
BSLO_VAL        equ     0               ; A2 maps to segment C000
                                        ; A3 maps to segment E000

; 28F020 Flash EEPROM Commands

CMD_READ                equ     0
CMD_ERASE               equ     20h
CMD_ERASE_VERIFY        equ     0a0h
```

```
CMD_SETUP_PROGRAM       equ     040h
CMD_PROGRAM_VERIFY      equ     0c0h
CMD_RESET               equ     0ffh
CMD_IDENTIFY            equ     90h

public          _chip_erase
public          _pcmcia_save
public          _seek_end
public          _read_header
public          _progsupply_on
public          _progsupply_off
public          _card_detect
public          _bankswitch_disable

extrn           _data_buffer:BYTE
; extrn         _card_ptr:dword
extrn           _header_contents:BYTE
; extrn         _main_ds:WORD


assume          cs:_TEXT, ds:_DATA

_DATA           SEGMENT BYTE PUBLIC 'DATA'
dummy           DW      ?
answ_ax         DW      ?

_DATA  ENDS


_TEXT           segment word public 'CODE'


; NB this macro is not universal and is only correct for regmem == AX
; See Appendix A of CHIPS Superstate R Interface Guide for general case
; also, see CHIPS Programmer's reference Manual pp 2-12 to 2-19 incl.

LFEAT           MACRO           regmem
                DB              0FEH
                DB              0F8H
                ENDM

; NB this macro is not universal and is only correct for regmem == AL
; See Appendix A of CHIPS Superstate R Interface Guide for general case
; also, see CHIPS Programmer's reference Manual pp 2-12 to 2-19 incl.

STFEAT          MACRO           regmem, sdata
                DB              0FEH
                DB              0F0H
                DB              sdata
                ENDM

;*******************************************************************************
;
; _chip_erase
;
;  procedure to erase a single flash EEPROM chip in the PCMCIA Card
;
;*******************************************************************************
_chip_erase     PROC

                push            bp
                mov             bp, sp
```

```
                push    ax
                push    bx
                push    cx
                push    dx              ; save registers being used
                push    si
                push    di
                push    ss
                push    ds
                push    es

                mov     bx, WORD PTR [BP+4]   ; chip number
                mov     cl, 4
                shl     bx, cl          ; multiply by 16
                mov     cx, bx          ; no. of chip (0-15) x 16
                cmp     cx, 0f0h        ; CX used in Memory_map
                jg      Argument_Error1

                mov     ax, FLASH       ; destination of the data
                mov     es, ax

                call    VPP_ON          ; switch on VPP

                call    Erase_All       ; erase device - see fig 6 of 28F020
                                        ; data sheet
                jnz     Erase_Error     ; jump on error

                jmp     Exit1

Erase_Error:
                jmp     Exit1

Argument_Error1:
                jmp     Exit1

Exit1:
                call    VPP_OFF         ; switch off VPP

                pop     es
                pop     ds
                pop     ss
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                mov     sp, bp
                pop     bp
                ret

_chip_erase     ENDP

;*************************************************************************
;
; _read_header
;
;  procedure to read 32 bytes of directory information into the
;  global string  _header_contents
;  NB relies on a directory entry not crossing a segment boundary
;
;  unsigned arguments SEGMENT  and OFFSET/PTR are passed by calling code
;
;  Returns 1 if successful, 0 if called with out-of-range segment
```

```
;
;*****************************************************************************
_read_header  PROC

              push      bp
              mov       bp, sp
;             push      ax
              push      bx
              push      cx
              push      dx              ; save registers being used
              push      si
              push      di
              push      ss
              push      ds
              push      es

              mov       bx, WORD PTR [BP+4]   ; segment (0-63)
              mov       cl, 2
              shl       bx, cl          ; mult by 4
              mov       cx, bx          ; (CX used in Memory_map)
              cmp       cx, 0fch
              jg        Argument_Error3       ; out of range

              mov       ax, FLASH
              mov       es, ax          ; set up ES as mapped Flash segment

              call      Memory_map      ; set up memory map

              mov       bx, WORD PTR [BP+6]   ; seg_ptr (offset)
                                              ; es:bx points to start of header
              call      Read_Cmd        ; issue read command

              mov       cx, 32
              mov       di, offset _header_contents

Head_loop:
              mov       al, BYTE PTR es:[bx]
              mov       BYTE PTR [di], al
              inc       bx
              inc       di
              loop      Head_loop

              mov       BYTE PTR [di], 0      ; string terminator
              mov       ax, 1           ; flag for OK
              jmp       Tidy_up

Argument_Error3:
              mov       ax, 0           ; flag for failure

Tidy_up:
              pop       es
              pop       ds
              pop       ss
              pop       di
              pop       si
              pop       dx
              pop       cx
              pop       bx
;             pop       ax
              mov       sp, bp
              pop       bp
              ret
```

```
_read_header   ENDP

;*****************************************************************************
;
; _pcmcia_save
;
;  procedure to write LENGTH bytes, start in 64k segment SEG at pointer PTR
;
;  (unsigned arguments passed in the above order at [BP+4], [BP+6], [BP+8])
;  and source data start address passed at [BP+10] (far address i.e. 4 bytes)
;
;*****************************************************************************
_pcmcia_save  PROC
              push      bp
              mov       bp, sp

;             push      ax
              push      bx
              push      cx
              push      dx
              push      si
              push      di
              push      ss
              push      ds
              push      es


              mov       bx, WORD PTR [BP+6]   ; no. of Flash Segment (0-63)
              mov       cl, 2
              shl       bx, cl          ; mult by 4
              mov       cx, bx          ; (CX used in Memory_map)
              cmp       cx, 0fch
              jg        Argument_Error2

              mov       dx, WORD PTR [BP+4]   ; no. of bytes to write less 1

              mov       bx, WORD PTR [BP+8]   ; es:bx will point to
                                              ; start byte in Flash

              mov       ax, FLASH
              mov       es, ax          ; set up ES as mapped Flash segment

;             call      VPP_ON
              call      Memory_map    ; set up memory map

              call      Program_Set   ; program device - see fig 5 of 28F020
                                      ; data sheet
              jnz       Program_Error         ; jump on error

;             mov       dx, offset M_Program_OK
;             call      Print_Message         ; print OK

              mov       ax, 0         ; return value for OK
              jmp       Exit2

Program_Error:
;             mov       dx, offset M_Program_Error
;             call      Print_Message
              mov       ax, 2         ; return value for prog error
              jmp       Exit2

Argument_Error2:
```

```
;               mov     dx, offset M_Arg_Error2
;               call    Print_Message
                mov     ax, 1           ; return value for segment call error
                jmp     Exit2


Exit2:

;               call    VPP_OFF         ; switch off VPP

                pop     es
                pop     ds
                pop     ss
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
;               pop     ax
                mov     sp, bp
                pop     bp
                ret

_pcmcia_save    ENDP

;*****************************************************************************
;
; _seek_end
;
;  procedure to find 1st free byte in card (starting at chip number
;  is passed to routine)
;  result (long) returned to calling program. AX = Ptr, DX = Segment
;
;*****************************************************************************
_seek_end       PROC

                push    bp
                mov     bp, sp

;               push    ax
                push    bx
                push    cx
;               push    dx
                push    si
                push    di
                push    ss
                push    es
                push    ds


                mov     bx, WORD PTR [BP+4]   ; chip number
                mov     cl, 4
                shl     bx, cl          ; multiply by 16
                mov     cx, bx          ; no. of chip (0-15) x 16
                cmp     cx, 0f0h        ; CX used in Memory_map
                jg      Card_end

                mov     ax, FLASH       ; destination of the data
                mov     es, ax

; temp code to read Identification Codes
;               call    VPP_ON          ; switch on VPP
;               call    Memory_map      ; map segment to FLASH (C000)
```

```
;               xor     bx, bx
;               call    Identify
;               mov     al, es:[0]
;               call    Print_Hex
;               mov     al, es:[1]
;               call    Print_Hex
;               call    Read_Cmd
;               call    VPP_OFF         ; switch off VPP

Seg_search:
                mov     dx, cx
                push    cx
                mov     cl, 2
                shr     dx, cl          ; DX = Segment number (0 - 63)
                pop     cx

                call    Memory_map      ; map segment to FLASH (C000)

                call    Find_FF         ; search a 64k segment for FF

                cli
                mov     ax, _DATA
                mov     ds, ax          ; ensure DS is for this module
                sti
                mov     ax, answ_ax     ; AX = offset within card segment DX
                jz      Found

                add     cx, 4           ; set CX for next segment
                cmp     cx, 0fch
                jg      Card_end
                jmp     Seg_search

Card_end:
                mov     ax, 0           ; returned pointer for failure
                mov     dx, 40h         ; returned segment (normal range 0-63)

Found:

                pop     ds
                pop     es
                pop     ss
                pop     di
                pop     si
;               pop     dx
                pop     cx
                pop     bx
;               pop     ax
                mov     sp, bp
                pop     bp
                ret

_seek_end       ENDP

;*******************************************************************************
;
; Find_FF
;
;  finds first occurrence of byte==FF in a segment
;
;     If successful, returns pointer in AX with Z flag set
;     If FF not found, returns with Z flag reset
```

```
;
; ***************************************************************************
Find_FF         PROC
                push            ax
                push            bx
                push            di

                mov             bx, 0           ; set ptr to start of segment
                call            Read_Cmd        ; issue read command

Ptr_loop:

                cli
                mov             ax, _DATA
                mov             ds, ax          ; ensure DS is for this module
                sti
                mov             answ_ax, bx
                mov             al, BYTE PTR es:[bx]    ; read data
;               call            Print_letter    ; temp testing
                cmp             al, 0ffh        ; data == FF?
                je              Located
                cmp             bx, 0ffe0h
                je              End_seg
                add             bx, 32
                jmp             Ptr_loop

Located:

;               cli
;               mov             ax, _DATA
;               mov             ds, ax
;               mov             bx, answ_ax
;               mov             ax, bx
;               xchg            ah, al
;               call            Print_hex
;               xchg            ah, al
;               call            Print_hex
;               mov             ax, main_ds
;               mov             ds, ax
;               sti

;               mov             di, offset _card_ptr
;               mov             WORD PTR [di], bx
;               mov             WORD PTR [di + 2], dx
                call            Memory_Restore          ; NEW!!! reset Bank Switching
;               call            Print_Hex
                xor             ax, ax          ; set Z flag for success
                mov             ax, bx
                jmp             End_label

End_seg:
                call            Memory_Restore          ; NEW!!! reset Bank Switching
                inc             ax                      ; reset Z flag for failure

End_label:
                pop             di
                pop             bx
                pop             ax
                ret
Find_FF         ENDP

; ***************************************************************************
;
```

```
; Print_Hex
;
;    Prints a byte in al as 2 hex chars
;
;*****************************************************************************
Print_Hex       PROC
                push            ax
                push            cx

                mov             ah, al
                and             al, 0f0h
                mov             cl, 4
                shr             al, cl
                add             al, 30h
                cmp             al, 3ah
                jl               Dec_Char1
                add             al, 7

Dec_Char1:
                call            Print_letter      ; 1st hex character
                mov             al, ah
                and             al, 0fh
                add             al, 30h
                cmp             al, 3ah
                jl               Dec_Char2
                add             al, 7

Dec_Char2:
                call            Print_letter      ; 2nd hex character
                mov             al, 20h
                call            print_letter      ; print space

                pop             cx
                pop             ax
                ret
Print_Hex       ENDP

;*****************************************************************************
;
; Memory_Map
;
;    sets up PC/Chip address map registers to put the 1st 64k of
;    the PCMCIA flash EEPROM at C000h
;
;*****************************************************************************
Memory_Map  PROC

                push            ax
                push            bx
                push            ds

;               cli

;               mov             bh, 0ch           ; CREG for bank switch enable
;               mov             bl, SET_CREG
;               mov             al, 0             ; value to reset enable
;               mov             ah, GET_SET_CREG
;               int             1fh               ; call Superstate code
                mov             ah, 0ch
                mov             al, 0
                LFEAT           ax

;               mov             bh, BSHI          ; hi byte for mapped 64k segment
```

```
;               mov     bl, SET_CREG
;               mov     al, BSHI_VAL    ; value to write to it
;               mov     ah, GET_SET_CREG
;               int     1fh             ; call Superstate code
                mov     ah, BSHI
                mov     al, BSHI_VAL
                LFEAT   ax

;               mov     bh, BSLO        ; lo byte for mapped 64k segment
;               mov     bl, SET_CREG
;               mov     al, BSLO_VAL    ; value to write to it
;               add     ax, cx
;               mov     ah, GET_SET_CREG
;               int     1fh             ; call Superstate code
                mov     ah, BSLO
                mov     al, BSLO_VAL
                add     ax, cx
                LFEAT   ax

;               mov     bh, 0ch         ; CREG for bank switch enable
;               mov     bl, SET_CREG
;               mov     al, 1           ; value to set enable
;               mov     ah, GET_SET_CREG
;               int     1fh             ; call Superstate code
                mov     ah, 0ch
                mov     al, 1
                LFEAT   ax

;               sti

                pop     ds
                pop     bx
                pop     ax
                ret
Memory_Map      ENDP

;*****************************************************************************
;
; Memory_Restore
;
;  disables Bank Switching
;
;*****************************************************************************
Memory_Restore          PROC

                push    ax
                push    bx
                push    ds

;               cli

;               mov     bh, 0ch         ; CREG for bank switch enable
;               mov     bl, SET_CREG
;               mov     al, 0           ; value to reset enable
;               mov     ah, GET_SET_CREG
;               int     1fh             ; call Superstate code
                mov     ah, 0ch
                mov     al, 0
                LFEAT   ax

;               sti

                pop     ds
```

```
                pop             bx
                pop             ax
                ret
Memory_Restore                  ENDP

;*****************************************************************************
;
; Erase_All
;
;  Uses algorithm in 28F020 data sheet to erase the chip
;     returns with Z flag set if OK
;
;*****************************************************************************
Erase_All       PROC

                push            cx

                mov             ax, 0

Chip_seg:                       ; loop to program 48*64k segments to 0
                push            ax
                call            Memory_map

                push            cx
                call            Program_Zeros
                pop             cx
                pop             ax
                jnz             E_Error

                add             cx, 4           ; for next 64k
                add             ax, 1
                cmp             ax, 4
                je              All_done
                jmp             Chip_seg

All_done:
                mov             cx, 0           ; cx is PLSCNT in data sheet
EA1:
                inc             cx
                cmp             cx, 3000        ; tried 3000 times?
                jz              E_Error         ; yes- quit
                call            Erase           ; issue erase command
                call            Erase           ; twice to enable erase
                mov             ax, 10000   ; 10ms
                call            Delay           ; wait a while
                mov             bx, 0           ; address of bottom of EEPROM
EA2:
                call            Erase_Verify ; issue erase verify command
                mov             ax, 6
                call            Delay           ; wait 6us
                mov             al, es:[bx]     ; read data
                cmp             al, 0ffh        ; data = ff?
                jnz             EA1             ; no - jump
                inc             bx              ; next address
                jnz             EA2             ; no - next byte

;               mov             al, "E"
;               call            Print_Letter    ; status report

                cmp             bh, 0           ; gone all the way around?
                jnz             EA2

                call            Read_Cmd        ; issue read command
```

```
                xor             ax, ax          ; set Z flag to show success
                pop             cx
                ret

E_Error:
                inc             cx              ; clear Z flag to show failure
                pop cx
                ret
Erase_All       ENDP

; ***************************************************************************
;
; Program_Set
;
;  Uses algorithm in 28F020 data sheet to write to the chip
;     bx points to 1st write address
;     and dx is the number of bytes to be written
;     returns with Z flag set if OK
;
; ***************************************************************************
Program_Set     PROC
                push            cx
                push            di

;               mov             di, offset _data_buffer     ; ds:di is start of buffer
                                                            ; to be written
                mov             di, WORD PTR [BP+10]    ; address of start of source data
PA1:
                mov             cx, 0           ; cx is PLSCNT in data sheet
PA2:
                inc             cx
                cmp             cx, 26          ; tried enough times?
                jz              P_Error         ; yes - fail
                call            Setup_Program           ; set up for programming


                mov             al, ds:[di]     ; get byte from data source

                mov             es:[bx], al     ; write byte to Flash EEPROM
                mov             ax, 10
                call            Delay           ; wait 10us
                call            Program_Verify          ; issue program verify command
                mov             ax, 6
                call            Delay           ; wait 6us
                mov             ah, es:[bx]     ; read data from EEPROM
                mov             al, ds:[di]     ; get byte from data_buffer
                cmp             al, ah          ; compare with source
                jnz             PA2             ; jump if data not correct

;               mov             al, "P"
;               call            Print_Letter

                inc             di              ; next location in data_buffer
                inc             bx              ; next address to write
                jc              Overrun
                cmp             dx, 0
                je              PA3
                dec             dx              ; no. remaining to be written less 1
                jmp             PA1             ; if any remaining, loop
PA3:
                call            Read_Cmd        ; issue read command
                xor             ax, ax          ; set Z flag to show success
```

```
                pop     di
                pop     cx
                ret

P_Error:
                inc     cx              ; clear Z flag to show failure
                pop     di
                pop     cx
                ret

Overrun:
;               mov     dx, offset M_Overrun
;               call    Print_Message

                inc     cx              ; clear Z flag to show failure
                pop     di
                pop     cx
                ret
Program_Set     ENDP

;*****************************************************************************
;
; Program_Zeros
;
;  Uses algorithm in 28F020 data sheet to fill chip with 0
;     returns with Z flag set if OK
;
;*****************************************************************************
Program_Zeros           PROC

                mov     bx, 0           ; point to start of EEPROM

PZ1:
                mov     cx, 0           ; cx is PLSCNT in data sheet
PZ2:
                inc     cx
                cmp     cx, 26          ; tried enough times?
                jz      P_Error_Z       ; yes - fail
                call    Setup_Program           ; set up for programming
                mov     al, 0           ; get byte to program
                mov     es:[bx], al     ; write data to EEPROM
                mov     ax, 10
                call    Delay           ; wait 10us
                call    Program_Verify          ; issue program verify command
                mov     ax, 6
                call    Delay           ; wait 6us
                mov     al, es:[bx]     ; read data from EEPROM
                cmp     al, 0           ; compare with source
                jnz     PZ2             ; jump if data not correct
                inc     bx              ; next memory address
                cmp     bl, 0           ; done whole block
                jnz     PZ1             ; nop - loop

;               mov     al, "Z"
;               call    Print_Letter

                cmp     bh, 0           ; gone all the way around?
                jnz     PZ1             ; no - loop

                call    Read_Cmd        ; issue read command
                xor     ax, ax          ; set Z flag to show success
```

```
                ret

P_Error_Z:
                inc             cx              ; clear Z flag to show failure

                ret
Program_Zeros                   ENDP

;*************************************************************************
;
; Read_Cmd
;
;  issues read command to EEPROM
;
;*************************************************************************
Read_Cmd        PROC
                push            ax
                mov             al, CMD_READ
                mov             es:[bx], al     ; issue command
                pop             ax
                ret
Read_Cmd        ENDP


Identify        PROC
                push            ax
                mov             al, CMD_IDENTIFY
                mov             es:[bx], al     ; issue command
                pop             ax
                ret
Identify        ENDP

;*************************************************************************
;
; Erase
;
;  issues erase command to EEPROM
;
;*************************************************************************
Erase           PROC
                push            ax
                mov             al, CMD_ERASE
                mov             es:[bx], al     ; issue command
                pop             ax
                ret
Erase           ENDP


;*************************************************************************
;
; Erase_Verify
;
;  issues erase verify command to EEPROM
;     bx must contain address
;
;*************************************************************************
Erase_Verify    PROC
                push            ax
                mov             al, CMD_ERASE_VERIFY
                mov             es:[bx], al     ; issue command
                pop             ax
                ret
Erase_Verify    ENDP
```

```
;*************************************************************************
;
; Setup_Program
;
;  issues setup program command to EEPROM
;
;*************************************************************************

Setup_Program                   PROC
                push            ax

                mov             al, CMD_SETUP_PROGRAM
                mov             es:[bx], al         ; issue command
                pop             ax
                ret
Setup_Program                   ENDP

;*************************************************************************
;
; Program_Verify
;
;  issues program verify command to EEPROM
;
;*************************************************************************
Program_Verify                  PROC
                push            ax
                mov             al, CMD_PROGRAM_VERIFY
                mov             es:[bx], al         ; issue command
                pop             ax
                ret
Program_Verify                  ENDP

;*************************************************************************
;
; Delay
;
;  ax contains the number of microseconds to delay
;     !!! very crude - uses program loop
;
;*************************************************************************
Delay           PROC
                cmp             ax, 0               ; count = 0?
                jz              DL1                 ; yes - exit
                nop
                nop
                nop
                nop
                dec             ax
                jmp             Delay
DL1:
                ret
Delay           ENDP

;*************************************************************************
;
; VPP_ON
;
;  turns on VPP
```

```
;
;*************************************************************************
VPP_ON          PROC
                push            ax
                push            bx
                push            dx

;               cli
                call            Enable_CSUTIL           ; enable PS4 to be CSUTIL pin
; access Utility Register to turn on VPP

                mov             dx, VPP_ON_PORT         ; turn on VPP
                out             dx, al                  ; data is ignored
;               sti

                mov             ax, 50000
                call            Delay                   ; wait 50ms for VEE to turn on
                call            Disable_CSUTIL          ; disable CSUTIL

                pop             dx
                pop             bx
                pop             ax
                ret
VPP_ON          ENDP

;*************************************************************************
;
; VPP_OFF
;
;  turns of VPP
;
;*************************************************************************
VPP_OFF         PROC
                push            ax
                push            bx
                push            dx

;               cli
                call            Enable_CSUTIL           ; enable PS4 to be CSUTIL pin

; access Utility Register to turn on VPP

                mov             dx, VPP_OFF_PORT        ; turn off VPP
                out             dx, al                  ; data is ignored
;               sti

                call            Disable_CSUTIL          ; disable CSUTIL
                pop             dx
                pop             bx
                pop             ax
                ret
VPP_OFF         ENDP

;*************************************************************************
;
; progsupply_on
;
;  turns on VPP (for external calls)
;
;*************************************************************************
_progsupply_on                  PROC
```

```
                push    bp
                mov     bp, sp
                push    ax
                push    bx
                push    cx
                push    dx                      ; save registers being used
                push    si
                push    di
                push    ss
                push    ds
                push    es


;               cli
                call    Enable_CSUTIL           ; enable PS4 to be CSUTIL pin
; access Utility Register to turn on VPP

                mov     dx, VPP_ON_PORT         ; turn on VPP
                out     dx, al                  ; data is ignored
;               sti

                mov     ax, 50000
                call    Delay                   ; wait 50ms for VEE to turn on
                call    Disable_CSUTIL          ; disable CSUTIL

                pop     es
                pop     ds
                pop     ss
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                mov     sp, bp
                pop     bp
                ret

_progsupply_on          ENDP

;*************************************************************************
;
; progsupply_off
;
;  turns of VPP (for external calls)
;
;*************************************************************************
_progsupply_off         PROC

                push    bp
                mov     bp, sp
                push    ax
                push    bx
                push    cx
                push    dx                      ; save registers being used
                push    si
                push    di
                push    ss
                push    ds
                push    es
```

```
;               cli
                call            Enable_CSUTIL           ; enable PS4 to be CSUTIL pin

; access Utility Register to turn on VPP

                mov             dx, VPP_OFF_PORT        ; turn off VPP
                out             dx, al                  ; data is ignored
;               sti

                call            Disable_CSUTIL          ; disable CSUTIL

                pop             es
                pop             ds
                pop             ss
                pop             di
                pop             si
                pop             dx
                pop             cx
                pop             bx
                pop             ax
                mov             sp, bp
                pop             bp
                ret

_progsupply_off                 ENDP


;*****************************************************************************
;
; Enable_CSUTIL
;
;  enable access to Utility Register by setting PS4 pin
;  to be an active low chip select
;
;*****************************************************************************
Enable_CSUTIL                   PROC
;               mov             bh, PS4_ALOW
;               mov             bl, SET_CREG
;               mov             al, CSUTIL_BASE and 0ffh        ; ls bits of address
;               mov             ah, GET_SET_CREG
;               int             1fh                     ; call Superstate code
                mov             ah, PS4_ALOW
                mov             al, CSUTIL_BASE and 0ffH
                LFEAT           ax

;               mov             bh, PS4_AHIGH
;               mov             bl, SET_CREG
;               mov             al, (CSUTIL_BASE and 300h)/256          ; MS address
;               or              al, WRITE_16            ; add other bits
;               mov             ah, GET_SET_CREG
;               int             1fh                     ; call Superstate code
                mov             ah, PS4_AHIGH
                mov             al, (CSUTIL_BASE and 300H)/256
                or              al, WRITE_16
                LFEAT           ax

;               mov             bh, PS4_SELECTOR
;               mov             bl, SET_CREG
;               mov             al, SELECT_CS_LOW       ; active low CS
;               mov             ah, GET_SET_CREG
;               int             1fh                     ; call Superstate code
                mov             ah, PS4_SELECTOR
                mov             al, SELECT_CS_LOW
```

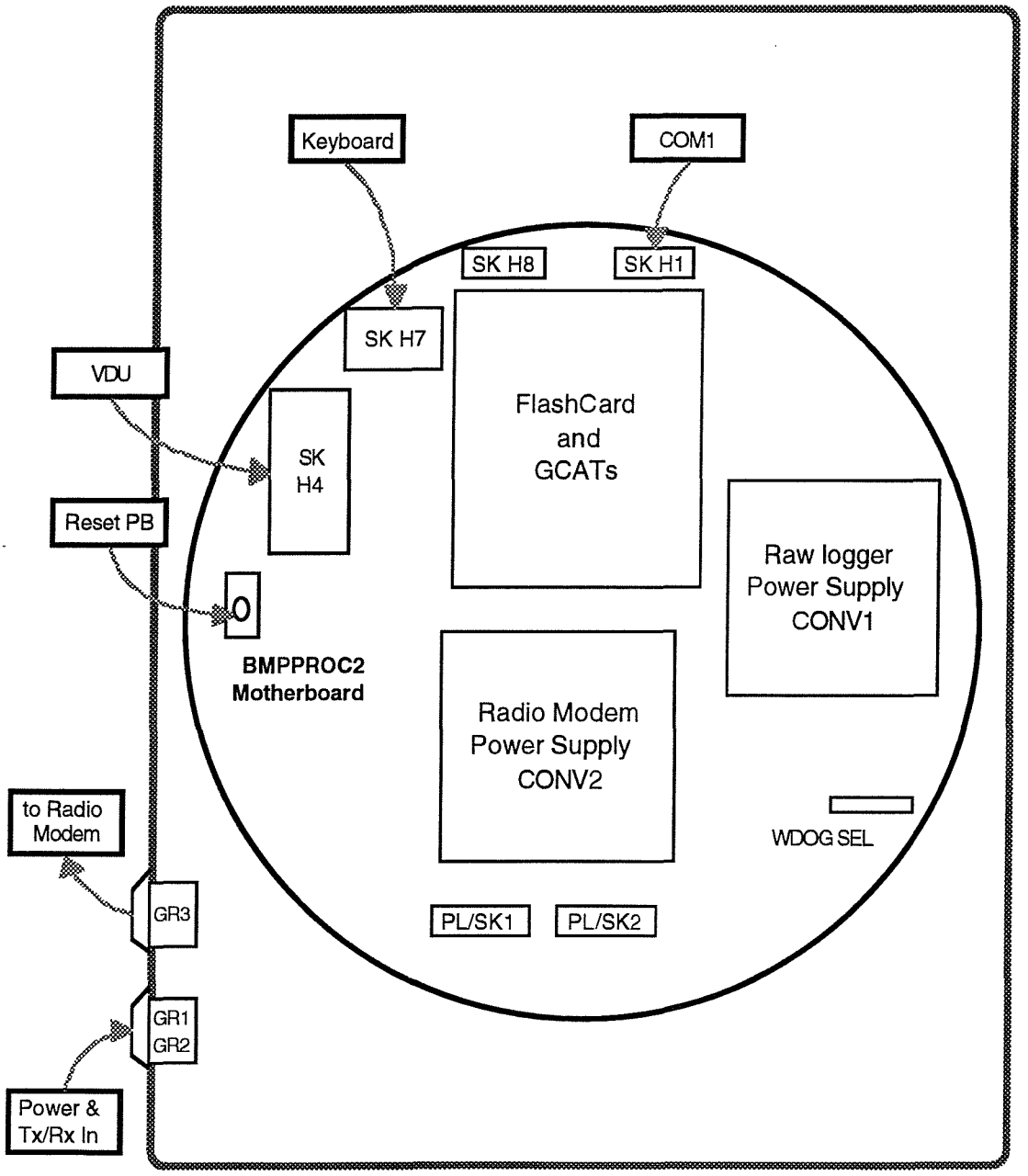
```
                LFEAT           ax

                ret
Enable_CSUTIL                   ENDP

;*************************************************************************
;
; Disable_CSUTIL
;
;  now disable access to Utility Register incase software crashes
;  and writes to it
;
;*************************************************************************
Disable_CSUTIL                  PROC
;               mov             bh, PS4_SELECTOR
;               mov             bl, SET_CREG
;               mov             al, SELECT_INPUT    ; set to input - pullup
;               mov             ah, GET_SET_CREG    ; resistor holds it high
;               int             1fh                     ; call Superstate code
                mov             ah, PS4_SELECTOR
                mov             al, SELECT_INPUT
                LFEAT           ax

                ret
Disable_CSUTIL                  ENDP

;*************************************************************************
;
; Check_Key_Press
;
;  uses MS_DOS interrupt to check for a key
;     zero flag is set if no key pressed
;
;*************************************************************************
Check_key_press                 PROC
                push            ax
                push            dx

                mov             ah, 06h         ; console input call
                mov             dl, 0ffh        ; input
                int             21h             ; see if key is pressed
                                                ; zero flag is set if no key was pressed
                pop             dx
                pop             ax
                ret
Check_key_press                 ENDP

;*************************************************************************
;
; Print_message
;
;  uses MS_DOS interrupt to print message
;     ds:dx points to message
;
;*************************************************************************
Print_Message   PROC
                push            ds
                push            ax

                mov             ax, cs
```

```
                mov             ds, ax          ; ds=cs to point to text

                mov             ah, 9h          ; string output
                int             21h             ; DOS call

                pop              ax
                pop              ds
                ret
Print_Message   ENDP

;*************************************************************************
;
; Print_Letter
;
;  uses MS_DOS interrupt to print letter in AL
;
;*************************************************************************
Print_Letter    PROC
                push            ax
                push            dx
                mov             dl, al

                mov             ah, 2h          ; character output
                int             21h             ; DOS call

                pop             dx
                pop             ax
                ret
Print_Letter    ENDP

;*************************************************************************
;
; _card_detect
;
;  uses STFEAT to read SDATA 0A for PCMCIA interface status
;
;*************************************************************************
_card_detect    PROC

                push            bp
                mov             bp, sp

                push            bx
                push            cx
                push            dx                      ; save registers being used
                push            si
                push            di
                push            ss
                push            ds
                push            es

                STFEAT          al, 0ah

                pop             es
                pop             ds
                pop             ss
                pop             di
                pop             si
                pop             dx
                pop             cx
                pop             bx
```

```
                mov             sp, bp
                pop             bp
                ret

_card_detect    ENDP

;*****************************************************************************
;
;_bankswitch_disable
;
;  disables Bank Switching
;
;*****************************************************************************
_bankswitch_disable             PROC

                push            ax
                push            bx
                push            ds

;               cli

;               mov             bh, 0ch         ; CREG for bank switch enable
;               mov             bl, SET_CREG
;               mov             al, 0           ; value to reset enable
;               mov             ah, GET_SET_CREG
;               int             1fh             ; call Superstate code
                mov             ah, 0ch
                mov             al, 0
                LFEAT           ax

;               sti

                pop             ds
                pop             bx
                pop             ax
                ret
_bankswitch_disable             ENDP

;*****************************************************************************

_TEXT           ends

                end
```

APPENDIX C GENERAL ASSEMBLY

Figure 3.



C.1 Parts List

1 off Diecast Box		RS Components Ltd. 506-930
1 off Motherboard	BMPPROC2	see Appendix D
1 off 3 way bulkhead connector	GR1	Lemo Series 3, ERA 303 CNL

2 off 8 way bulkhead connector	GR2, GR3	Lemo Series 3, ERA 308 CNL
4 off 12 mm M3 spacers		RS Components Ltd. 222-402
1 off chassis (mounting plate)		make to suit (no drawing)

APPENDIX D BMPPROC2

D.1 Motherboard for GCAT and AMPRO Minimodules™

The board circuit diagram and component layout are shown in Figures 4 and 5, respectively. Modifications to the standard circuit are listed below:

D1 is replaced by a link and D2 is omitted

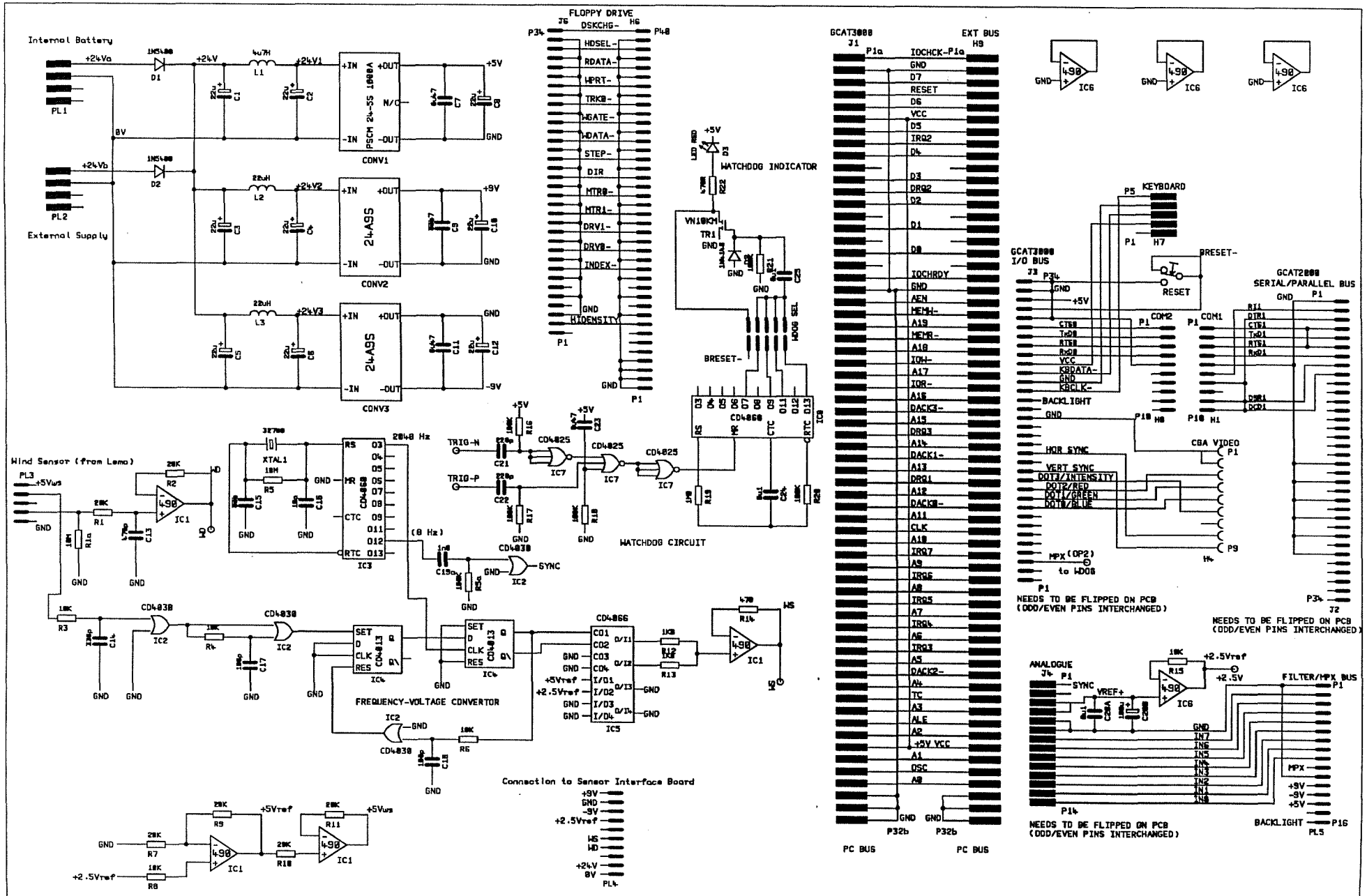
+9V and -9V Supplies are omitted

Wind Sensor (frequency to voltage converter) Circuit is omitted

Analogue reference Circuits are omitted

Analogue Filter/MPX Bus connector is omitted

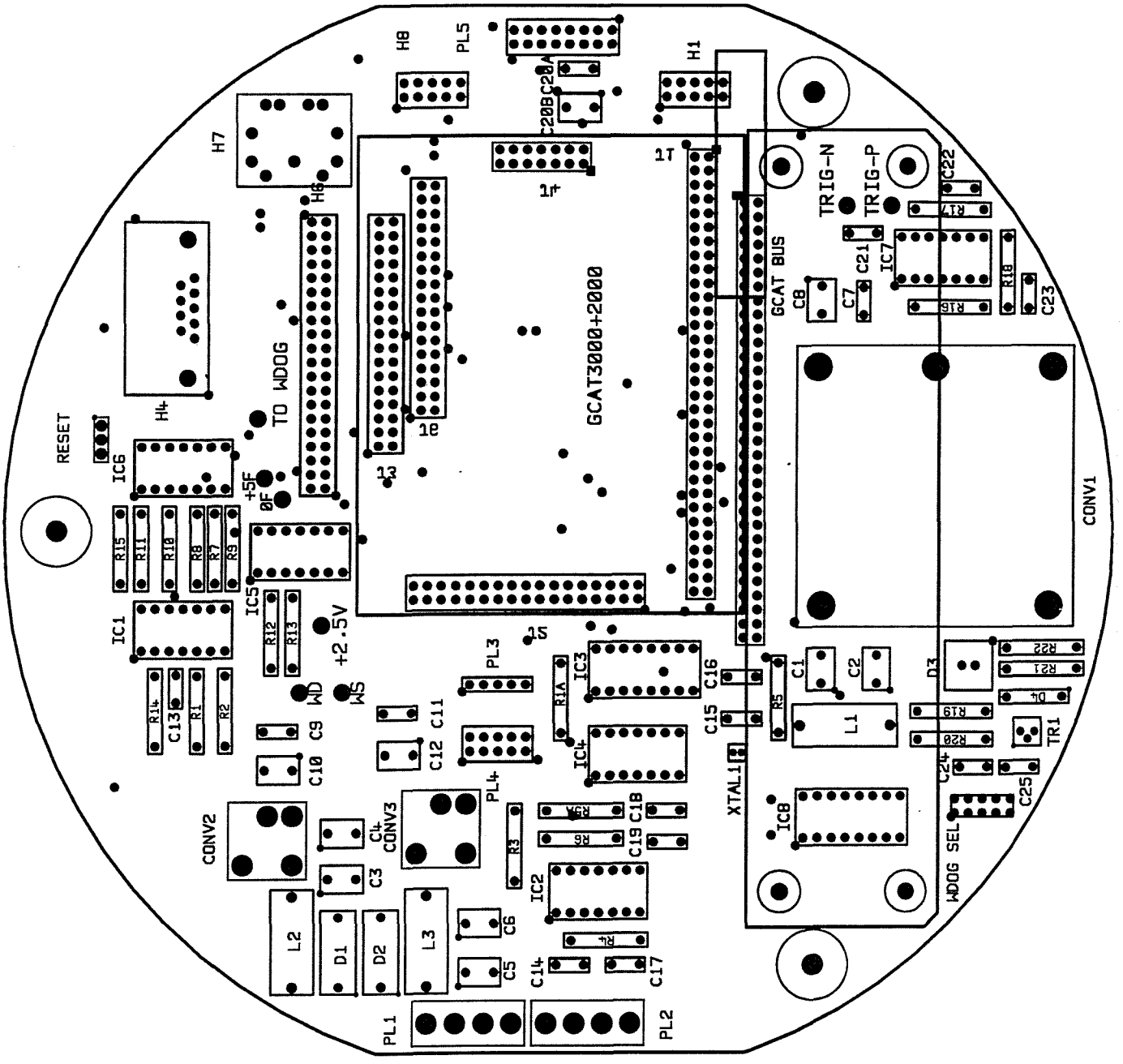
An additional convertor for the Radio Modem supply (CONV2) is mounted on the board as shown in figure 3.



INSTITUTE OF OCEANOGRAPHIC SCIENCES, DEACON LABORATORY, BROOK ROAD, WORMLEY, GODALMING, SURREY GU8 5UB, ENGLAND	FILENAME BMPPROC2.DGM	Sonic Buoy Motion Package Processor Board	SHEET 1 OF 1 / ISSUE 2 / DATE 02/06/1993.

Figure 4 BMPPROC2 Circuit Diagram

Figure 5 BMPPROC2 Component Layout



D.2 Parts List

Alphabetically ordered List of Parts with Silk References and Descriptions

1 off PCB	BMPPROC2	Motherboard manufactured to IOSDL artwork BMPPROC2.ART
2 off CTANT#15U	C1, C2,	Capacitor Tantalum 35V Farnell 100-907
2 off CMKS2#0U47	C7, C23	Capacitor Polycarbonate Farnell 143-684
1 off CTANT#22U	C8	Capacitor Tantalum 25V Farnell 100-892
2 off CFKC2#220P	C21, C22	Capacitor Polycarbonate Farnell 147-661
2 off CMKS2#0U1	C24, C25	Capacitor Polycarbonate Farnell 143-680
1 off DC24-5S	CONV1	24V i/p to 5V @ 2A o/p DC-DC Converter KRP Power Source UK, LPD 10/33 - 5S2000A
1 off DC24-12S	CONV2	24V i/p to 12V @ 0.8A o/p DC-DC Converter KRP Power Source UK, LPD 24-12S800A
1 off LED-0.2-RED	D3	Red LED Farnell 213-664
1 off 1N4148	D4	Small Signal Diode Farnell 1N4148
1 off IDC10	H1	Male PCB mounting IDC header Farnell 145-057
1 off D9SKT-RT	H4	90° PCB-mounting 9 way D Socket Farnell 150-738
1 off IDC40	H6	Socket Double Row 40 way 0.1" part of an M20-9833206
1 off DIN5PIN-RT	H7	PCB-mounting DIN 5 way Socket Farnell 148-505
1 off CD4025	IC7	Triple 3 i/p NOR gate Farnell CD4025BCN

1 off CD4060	IC8	Oscillator/Divider
		Farnell CD4060BCN
1 off GCAT64CON	J1	Socket Double Row 64 way 0.1"
		M20-9833206
2 off IDC34	J2, J3	Socket Double Row 34 way 0.1"
		M20-9833706
1 off IDC34	J6	Male PCB mounting IDC header
		Farnell .609-3427
1 off IRF#4U7H	L1	R.F.Choke
		Farnell 177-508
2 off PCCON4	PL1, PL2	Top Entry PCB Header - Open End 4 way
		Farnell 151-985
related parts *		
* 2 off	SK1, SK2	Free Plug 4 way
		Farnell 151-969
1 off 470uF/16V	Cx (between PL2 1&2)	Farnell 294-457
5 off RMFW25#100K	R16, R17, R18, R20, R21	Resistor 1/4W Metal Film
		Farnell SFR25 100K
1 off RMFW25#1M0	R19	Resistor 1/4W Metal Film
		Farnell SFR25 1M
1 off RMFW25#470R	R22	Resistor 1/4W Metal Film
		Farnell SFR25 470R
1 off SPDTBIASED	RESET	Push Button Switch Miniature
		Farnell 150-543
2 off TP	TRIG-N, TRIG-P	pads for patching Watchdog i/p
1 off VN10KM	TR1	Low Power MOSFET
		Farnell VN10KM
1 off PATCH5	WDOGSEL	Pin Header Straight Double Row 10 way
		part of Farnell 148-195

APPENDIX E FORMAT OF PCMCIA DIRECTORY AND DATA FILES

The PCMCIA filing system is a non-standard system developed in the absence (at the time) of a commercially available filing system for Flash EPROM PCMCIA cards. The Directory Area begins at relative address 0 and occupies the first 256 kbytes. The Data Area occupies the remainder of the 4 Mbytes.

Each directory entry consists of 32 bytes, which consist of a date/time stamp, data start address and length information; remaining bytes are used for additional information, in this case for a duplicate of the first 16 bytes. The format is shown below:

vjjjhhmmbfflrr0vjjjhhmmbfflrr0

where:

v = start character

jjj = (decimal) Julian day number (range 1 to 366)

hh = (decimal) hours (range 00 to 23)(date/time of directory entry)

mm = (decimal) minutes (range 00 to 59)

b = (binary) block number (range 0 to 63)

(card space consists of 64 blocks, each of 64k bytes, numbered 0 to 63)

ff = (binary) offset relative to the above block in bytes (range 0 to 65535)

ll = (binary) file length in bytes (range 0 to 65535)

rr = (binary) record number (range 0 to 65535)

0 = terminating character

In the above definitions, 'b' and 'ff' both refer to the relative start address of the file,

e.g. b = 6, ff = 3060 refer to a relative start address of (6 - 1) * 65536 + 3060 = 330740.

The data area starts at block 4, offset 0 and ends at block 63, offset 65535.

Each FASTCOM-format file begins with a 44 character header of the following format:
Mode<SP>1<LF>Analog<SP>1<LF>Time<SP>hh:mm:ss<SP>Date<SP>dd/nn/yy<LF>
where:

hh = (decimal) hours (range 00 to 23)

mm = (decimal) minutes (range 00 to 59)

ss = (decimal) seconds (range 00 to 59)

dd = (decimal) day of the month (range 00 to 31)

nn = (decimal) month (range 1 to 12)

yy = (decimal) year (range 00 to 99, year = 19yy)

This header is followed by anemometer raw data, consisting of (file length - 44) / 10 samples; each sample consists of 5 x (2 byte binary numbers), which represent U, V, W, C, H (three components of wind speed, velocity of sound and buoy heading).

Due to the change from unprompted data to prompted data, it is found that the block of data received by the Raw Data Logger after the Prompted and the first Transmit Block commands is garbled; this is, therefore, discarded. The length of file may not, therefore, amount to exactly 44 + 12288 bytes.