File

Institute of
Oceanographic Sciences
Deacon Laboratory

INTERNAL DOCUMENT No. 339

Sonic Buoy - Formatter handbook

C H Clayson

1994

Natural Environment Research Council

INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

INTERNAL DOCUMENT No. 339

Sonic Buoy - Formatter handbook

C H Clayson

1994

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	1994

TITLE

Sonic Buoy - Formatter handbook.

REFERENCE

Institute of Oceanographic Sciences Deacon Laboratory, Internal Document, No. 339, 125pp. (Unpublished manuscript)

ABSTRACT

The Formatter Processor was developed as part of the Sonic Buoy development program; it was intended primarily as a means of monitoring the operation of the Sonic and Multimet Processors.

It achieves this by capturing messages which are produced by the latter two processors, noting the capture times, aligning and amalgamating the processed data from the two sources and, finally, transmitting the compacted data via the ARGOS polar orbiting and METEOSAT geostationary satellite data collection systems. The compacted data are also saved in a 4 Mbyte Flash EEPROM PCMCIA card.

This document describes in detail the design and operation of the Formatter Processor; it is intended to serve the combined purposes of documenting the design and acting as a guide to operating the system and to 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 Formatter is designed to link the timing and data of the Sonic and Multimet Processors in the Sonic Buoy and to provide back-up storage and satellite telemetry of an abbreviated data set; the latter function is intended for diagnostic checks and as a last resort data back-up. The Formatter is a complete PC-based processing system, using DSP GCAT^{TM} 3000 and 2000 boards and two AMPRO MINIMODULE^{TM} /SSP boards; these boards are mounted on a motherboard BMPPROC2 of IOSDL design. The system is mounted within a tube which also contains the Sonic Processor system.

2. FUNCTIONAL DESCRIPTION

The functions of the Formatter are as follows:

a) to asynchronously receive processed data messages from the Sonic Processor via the Formatter COM3 port; these are output by the Sonic Processor via its COM2 port at 2400 baud at approximately 12, 27, 42 and 57 minutes past the hour (as determined by the Sonic Processor Real Time Clock.

The standard Sonic message format is:

S00YYMMDDHHmmSS00+P.SDMW.WS+NM.WS+EM.WS+VM.WSCME.ANHHH+A.1F+B.BBE +BBBT

where

S00 is the message header

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)

00 is the Date/Time terminator

+P.SD is the mean PSD value

MW.WS is the mean Wind Speed in m/s

+NM.WS is the North Vector Average Wind Speed in m/s

+EM.WS is the East Vector Average Wind Speed in m/s

+VM.WS is the Vertical Vector Average Wind Speed in m/s

CME.AN is the mean Speed of Sound in m/s

HHH is the mean Buoy Heading in degrees

+A.1F is the PSD vs Frequency regression fit a coefficient

+B.BBE+BBB is the PSD vs Frequency regression fit b coefficient in scientific notation

T is the message terminator

Total length 70 bytes

b) to asynchronously receive standard format processed messages from the Multimet Processor via the Formatter COM4 port; these are output by the Multimet Processor via its RS232 port at 2400 baud at approximately 50 seconds past each minute (as determined by the Multimet Processor Real Time Clock.

The standard Multimet message format is:

S00YYMMDDHHmmSS00HX01HX02HX03HX04HX05HX06HX07HX08HX09HX10HX11HX12HX13HX14HX15HX16HX17HX18HX19HX20HX21HX22HX23HX24HX25HX26HX27HX28HX29HX30T

where

S00 is the message header

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)

00 is the Date/Time terminator

followed by 30 channels HXnn of 4 hex bytes each, e.g. HX01 might be 7FE0

T is the message terminator

Total length 138 bytes

c) to check both Sonic and Multimet messages for correct format and to determine the time differences between the Formatter Real Time Clock and the Sonic and Multimet Real Time Clocks.

d) to average data from the Multimet messages whose acquisition times lie within the nominal 10 minute period of Sonic data acquisition (actually 9.83 minutes), after correction of all timing for Real Time Clock differences. In the event of loss of Sonic data, the Multimet data for the interval TT to TT + 10 minutes is used, where TT is 00, 15, 30 or 45 minutes for the previous quarter hour as determined by the Formatter Real Time Clock. For example, if the Sonic message is not received by Formatter time 17:30, Multimet messages for the period 17:15 to 17:25 are averaged. In the case of compass data, the 4-quadrant averaging method used in Multimet is used on the message data values. The number of Multimet messages used is saved for inclusion in output messages.

e) to format Sonic data over the preceding 5 hours (20 x 1/4 hour periods) into a highly packed format for transmission via ARGOS over 4 frames of 32 bytes. The format, which is described in Appendix A, includes the number of Multimet messages used in averaging over the Sonic acquisition period, as in d), above.

f) to format Sonic and averaged Multimet data over each hour (4 x 1/4 hour periods) into an IA5 ASCII format message of 288 bytes length for transmission via METEOSAT at 18 minutes past each hour. The format, which is described in Appendix B, includes housekeeping data, such as Real Time Clock differences. The message is sent to the Meteosat DCP via the Formatter COM6 port at 8 minutes past each hour, in 72 sections of 4 bytes at 300 baud, using a hardware handshake.

g) to respond to asynchronous XON prompts for data from the ARGOS transmitter via the COM5 port (at approximately 2 minute intervals) by sending a frame of 32 bytes at 2400 baud. The 4 frames representing 5 hours of data are sent cyclically, so that a satellite has to be within view for at least 8 minutes to ensure acquisition of the complete data set. However, since satellite passes are typically 2-hourly, there is a strong likelihood of data redundancy.

h) to output diagnostic messages, including all I/O events and messages, via the Formatter COM1 port at 2400 baud.

i) to save the ARGOS and METEOSAT messages on a Flash EEPROM memory card, as a back-up in the event of transmission failures or failures of the EPROM loggers associated with the Sonic and Multimet Processors

3. SOFTWARE

3.1 Overview

The Formatter 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 NEWFORM.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 ROMDisk 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 BIOS is used to prevent error messages from occurring during boot up, due to the lack of a floppy drive. During development a BIOS using A: for the floppy was used 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".

The application SETTIME.EXE is run first when the system has booted up; it is designed to allow the synchronisation of the Formatter clock with the clock of an external PC, running the BASIC program SONTIM.BAS and with its COM1 port connected to the Formatter COM1 port. This external PC is normally a battery-powered Husky Hunter 16 (running DOS).

After the completion of the application SETTIME (which times out if no external PC is connected), the application NEWFORM.EXE is run; this is the actual formatting program with the functions described above. It continues running until terminated by manual intervention (by any keypress, although the keyboard is normally disconnected except during set-up) or by a system failure. A system failure, such as a processor crash or the failure to receive messages from either of the Sonic, Multimet processors for longer than 5 hours, or failure to

receive an XON from the ARGOS transmitter for longer than 5 hours, will result in the watchdog timer rebooting the system. An attempt is made to clear any possible interrupt latch-up by resetting the relevant (Sonic, Multimet or ARGOS) UART interrupt line after no message has been received for, respectively, 20, 2 and 3 minutes. This is flagged by an error message on the COM1 port diagnostic output.

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. Listings of the source code are given in Appendix C.

When it is run, the message "Sending Date: DD/MM/YY Time: HH:mm:SSQ to COM1" is output to the VDU (if present), 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 application then outputs the Date/Time message via the GCAT 2000 COM1 port.; the port is set up for 2400 baud (8 characters, 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 clock, using DOS DATE and TIME calls. Error messages are output to the VDU (if connected) in the event of the calls failing. The application then outputs a confirmatory Date/Time message (using the received Date/Time) to the VDU (if present) and to the external PC via the GCAT 2000 COM1 port.

Note that the SETTIME application is only effective if the GCAT configuration RAM has been set up on power up. Since the GCAT does not have battery-backed RAM, its clock will be set to the default "midnight of January 1st 1980" on power up. The set up software must be entered by pressing the F2 key during boot-up and then setting in the date and time on the set up screen (see operating manual). After escaping from the set up program by pressing F10, the boot-up will continue and SETTIME will be run. Thereafter, the SETTIME method of synchronising the GCAT to an external PC can be initiated by pressing the reset button on the BMPPROC2 board, or by pressing CTRL-ALT-DEL if a keyboard is plugged in.

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

3.3 NEWFORM - Application for Formatting Sonic and Multimet Data

The application is built from the object files NEWFORM1.OBJ and FLASH5.OBJ. NEWFORM1.OBJ is produced by compiling the 'C' code NEWFORM1.C, using the include files FORM_VAR.C and COMS.C. FLASH5.OBJ is produced by assembling the assembly code FLASH5.ASM, using MASM /MX FLASH5. The library SLIBCE.LIB is used when linking and a stack size of 8000h is specified. Listings of the source code are given in Appendix D.

Note that the program can be conditionally compiled to give diagnostic output to either the VDU or via the COM1 port by setting DISPLAY to TRUE or FALSE, respectively; output is normally via the COM1 port to allow monitoring via RS232 on an external PC using, for example, KERMIT.

When it is run the following initialisation steps are carried out:

Turn on the Flashcard Programming Supply, VPP

Read the F8680 UART configuration;

- this normally returns 0x0f, i.e. the GCAT 3000 port is defined as COM2, the interrupt is active low, enabled

Get the PC/CHIP Options;

- this normally returns 0x02, i.e. drive B is PCMCIA

Get the 82C710 Options;

- this normally returns 0xec, i.e. XT IDE, FDC, parallel and serial ports enabled

Disable the COM2 (IRQ3) UART in the 8680

Set up the COM ports (COM1, COM3, COM4, COM5 and COM6) and the related interrupt handlers

Set timezone to GMT

Detect whether a flash card is inserted;

- if absent set flags, otherwise find the last entry in the directory

(send "Virgin FlashCard" to VDU or COM1 if directory is empty)

Set FlashCard directory pointer and data pointer

Initialise some variables (times)

Set timer for speaker frequency and trigger the watchdog circuit (tick)

The program then enters a continuous loop; the functions of this loop are described below and are also illustrated by the flow chart, figure 1.

If a second has elapsed, trigger the watchdog (tick)

If an ARGOS XON has been detected (argos_send_flag is 11h)

- send "ARG-n" to VDU or COM1, where n is the frame to be transmitted

- send the frame to the ARGOS transmitter via COM5 and reset the flag

If an ARGOS XOFF has been detected (transmitter times out after 2 seconds)

- reset the flag, update the time of last ARGOS message

- send XOFF to VDU or COM5

Check if it is time to send Meteosat data to transmitter (8 minutes past the hour)

- if so, send it as 72 chunks of 4 bytes on successive passes through the loop

Check if it is time to output ARGOS and Meteosat messages for diagnostics

(5, 20, 35 or 50 minutes past the hour)

- if so send the messages with headers via COM1 and

write the data to FlashCard

If a complete Multimet message has been received, but not checked

- check it for errors and replace the last byte with an error code

- get the Multimet Clock time from the header time + 1 minute (actually 50 seconds)

- calculate the Multimet clock "error" cf the Formatter, tmet

and write this to VDU or COM1

- limit the range of tmet to ± 9999

- set a flagto show Multimet message has been checked

If a complete Sonic message has been received, but not checked

- check it for errors and replace the last byte with an error code

- get the Sonic Clock time from the header + 12 minutes

- calculate the Sonic clock "error" cf the Formatter, tson

and write this to VDU or COM1

- limit the range of tson to ± 9999

- set a flagto show Sonic message has been checked

Carry out conversion of data if a quarter hour has elapsed or a new Sonic message has been received (see flow chart for a more rigorous definition of this)

If no ARGOS XON has been received for 3 minutes

- try clearing the COM5 UART interrupt line

If no Sonic message has been received for 20 minutes

- try clearing the COM3 UART interrupt line

If no Multimet message has been received for 2 minutes

- try clearing the COM4 UART interrupt line

If any of the 3 ports have failed, as above, for more than 5 hours

- inhibit the watchdog trigger (tick)

The loop can be exited from by pressing a key; the applications then quits, in an ordered manner, by

Disabling the UART interrupt lines

Disabling the UART interrupt enables

Clearing any interrupts pending

Restoring the original interrupt masks and handlers

Disabling the memory bank switch registers (used by the FlashCard software)

Turning off the FlashCard VPP supply

As mentioned above, the watchdog circuit is triggered by the speaker output waveform; the duration and frequency of this waveform are set to give approximately a single pulse on the speaker, which results in an audible "tick". This is useful as an audible indication that the application is running correctly.

4. HARDWARE

4.1 General Description

The BMPPROC2 motherboard is mounted on the end cap of the combined Sonic/Formatter housing, using aluminium spacers. A "stage plate is mounted on a second set of spacers and the Sonic Processor system is suspended from the stage plate on a further set of spacers (see Figure E.1, Appendix E). The two AMPRO Minimodule boards are mounted on insulating pillars off the motherboard and are connected to the GCAT bus via a 64 - way ribbon cable with IDC connectors. The GCAT 3000 and 2000 boards in tandem are plugged directly into sockets on the motherboard. Signal connections from the various Lemo bulkhead connectors in the end cap are made to the boards via a number of IDC ribbon cable connectors and power connections via two part PCB connectors. Connections for keyboard and VDU for testing/set-up purposes are made via standard 5 - way DIN and 9 - way D-type connectors on the motherboard.

4.2 Circuit Descriptions

4.2.1 BMPPROC2 Motherboard

The BMPPROC2 motherboard (see Figure E.2, Appendix E) is a general purpose board design which is only part filled for this application. An on-board DC-DC convertor produces a +5 Volt stabilised supply at up to 1 Amp from the (nominally) 24 volt input from the battery distribution system (DC-DC Convertor Box). This supply is conservatively rated for the Formatter 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 watchdogcan also be disabled from resetting the GCAT by removing a jumper.

4.2.2 GCAT Boards

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

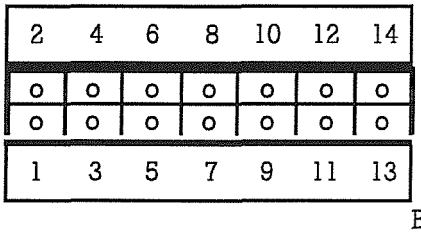
4.2.3 AMPRO MiniModules

The two AMPRO MiniModuleTM /SSP each have two PC-compatible RS232C serial ports and ine PC-compatible parallel port (not used in this application). They use the PC/104 bus, which is a standard for embedded applications. This bus is used (although with opposite sex connectors) by the DSP GCAT design. The required configuration is:

The AMPRO boards are set up by jumpers for use as COM3 - COM6 ports as follows:

W1 - W6 pin layout is:

Function	Port	Base Address	Interrupt	I/O Requirements
Sonic	COM3	3E8 hex	IRQ4	I/P only (no handshake)
Multimet	COM4	2E8 hex	IRQ7	I/P only (no handshake)
Argos	COM5	280 hex	IRQ3	I/P & O/P (no handshake)
Meteosat	COM6	288 hex	None	O/P only (H/W handshake)



Board 1

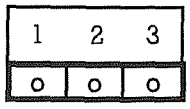
J1 (COM3)	W1 (Base Address)	W4 (Interrupt)
	link 11/12	link 5/6
J2 (COM4)	W2 (Base Address)	W5 (Interrupt)
	link 3/4 and 11/12	link 11/12

Board 2

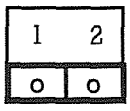
J1 (COM5)	W1 (Base Address)	W4 (Interrupt)
	link3/4, 7/7, 9/10, 11/12 & 13/14	link 3/4
J2 (COM6)	W2 (Base Address)	W4 (Interrupt)
	link 3/4, 7/8, 9/10 & 11/12	no link

On both boards, the J2 interface jumpers are set as follows:

W11 - W14 pin layout is:



W15 - W20 pin layout is:



W11	W12	W13	W14	W15	W16	W17	W18	W19
link 2/3	link 2/3	link 2/3	link 2/3	open	open	open	open	short

The parallel port bidirectional enable/disable W20 is 'don't care' on both boards.

The J1 and J2 connectors have the following pin allocation:

1	2	3	4	5	6	7	8	9	10
DCD	DSR	RxD	RTS	TxD	CTS	DTR	RI	GND	(Test)

5. WIRING

5.1 Lid Connectors

The Formatter shares a common housing with the Sonic Processor. Figure 2, below, shows the layout of the eight Lemo lid connectors.

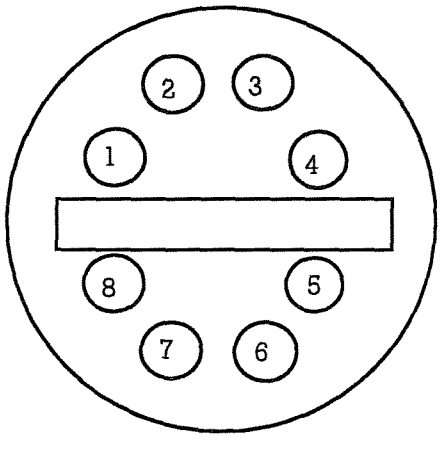


Figure 2 Formatter/Sonic Processor Lid

FS 1	METEOSAT	SERIES 3 5 PIN
FS 2	SONIC SENSOR	SERIES 3 6 PIN
FS 3	MONITOR	SERIES 3 5 PIN
FS 4	RAW DATA O/P	SERIES 3 8 PIN
FS 5	MET SERIAL I/P	SERIES 3 2 PIN
FS 6	SPARE	SERIES 3 5 PIN
FS 7	POWER	SERIES 3 10 PIN
FS 8	ARGOS	SERIES 3 7 PIN

5.2 Wiring from Lid to PCBs

This includes:

a) METEOSAT Lemo connector FS1 to Ampro SSP module A (COM6)

b) ARGOS Lemo connector FS8 to Ampro SSP module A (COM5)

c) POWER Lemo connector FS7 to SK1 on BMP

d) (MONITOR Lemo connector FS3 to SEROPT board SK6)

a) Lid Connector FS1 (Lemo Series 3, 5 pin) to Ampro SSP Module A Connector J2 (IDC10 free socket)

FS1 Pin	Function	J2 Pin
1	DCP 0V, Signal Ground	9
2	DCP Serial i/p, TxD	5
3	(RTS)	4
4	(RxD)	3
5	DCP Input Ready, CTS	6

b) Lid Connector FS3 (Lemo Series 3, 5 pin) to Sonic Processor SEROPT board SK6 (IDC10 free socket)

FS3 Pin	Function	SK6 Pin
1	Isolated Sonic Monitor	5
2	Isolated Multimet Monitor	4
3	Isolated Formatter Monitor	6
4	0V External I/P	10
5	+5V External I/P	1

c) Lid Connector FS7 (Lemo Series 3, 10 pin) to BMPPROC2 Motherboard SK1 (Weidmuller-Klippon 4 way free socket)

FS7 Pin	Function	SK1 Pin
1-8	used for Sonic Processor	
9	Formatter +24V	1
10	Formatter 0V	2

d) Lid Connector FS8 (Lemo Series 3, 7 pin) to Ampro SSP Module A Connector J1 (IDC10 free socket)

FS8 Pin	Function	J1 Pin
1	ARGOS 0V, Signal Ground	9
2	ARGOS Serial i/p, TxD	5
3	ARGOS Serial o/p, RxD	3
4	(CTS)	6
5	(RTS)	4
6-7	n/c	

5.3 Interboard Wiring

This includes:

a) Ampro SSP module B (COMs 3&4) to SEROPT board SK7 (isolated Sonic and Met RS232)

b) BMPPROC2 COM1 port (H1) to SEROPT board SK5 (Monitor o/p for isolation)

c) BMPPROC2 SK1 to Weidmuller-Klippon 4-way free socket on Sonic Motherboard (+5V Formatter)

a) Ampro SSP Module B Connector J1 (IDC10 free socket) to SEROPT board SK7 (IDC10 free socket)

J1 Pin	Function	SK7 Pin
1-2	n/c	
3	Serial i/p, SER 5 (Sonic)	3
4-8	n/c	
9	Signal Ground, 0V Formatter	4

Ampro SSP Module B Connector J2 (IDC10 free socket) to SEROPT board SK7 (IDC10 free socket)

J2 Pin	Function	SK7 Pin
1-2	n/c	
3	Serial i/p, SER 4 (Sonic)	1
4-8	n/c	
9	Signal Ground, 0V Formatter	2

b) BMPPROC2 Motherboard Connector H1 (IDC10 free socket) to SEROPT board SK5 (Molex 4 pin)

H1 Pin	Function	SK5 Pin
2	TxD, FORMATser	4
5	Signal Ground,FORMATgnd	1

c) BMPPROC2 Motherboard Connector SK1 (Weidmuller-Klippon 4-way free socket) to Sonic Motherboard Connector SK1 (Weidmuller-Klippon 4-way free socket)

SK1 Pin	Function	SK1 Pin
3	+5V Formatter	3
4	0V Formatter	4

6. OPERATIONAL

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

The combined Sonic/Formatter systems are removed from their tube by unscrewing the (six off) 4 mm screws which secure the end cap to the tube and withdraw the end cap with the systrems attached.

The Sonic and Formatter systems are both powered via the same cable (C30) to the endcap, but can be individually powered up or down by individual two-part (orange) PCB connectors, as required. To power up the Formatter without powering up the Sonic Processor, disconnect the (orange) PCB connector on the Sonic motherboard. Plug in a suitable keyboard (set for XT PC and NOT AT) and VDU (with TTL RGB interface NOT analogue; this may require some adjustment of the setting switches on the VDU). Plug in the (orange) PCB connector on the Formatter motherboard.

Plug in cable C30 from the DC-DC Convertor box to the endcap Lemo FS7. The GCAT will bleep and the "DSP Designs .....etc." message will 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 set the move highlight to another parameter after setting the last alteration. For exact time setting the F10 key should be pressed exactly at the time which has been entered on the screen (down to the last second). Do not take too long over this 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.

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 H1 (COM1) - this runs to the end cap monitor Lemo FS3

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 Formatter COM1 port

Set the PC Date/Time and 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 on the motherboard next to the VDU connector; 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 NEWFORM application.

Remove the ribbon cable from the GCAT COM1 port H1 and reconnect the ribbon cable from the end cap monitor Lemo.

Note that, in order to monitor the running of the NEWFORM application, it is necessary to connect the COM1 port of a PC running a terminal application (e.g. KERMIT) at 2400 baud either directly to the COM1 port H1 via a suitable cable, e.g. that labelled "Husky to Formatter", or to the end cap MONITOR Lemo FS3. In the latter case, the connection is via an opto-isolator, which requires +5V power via the monitoring cable. The opto-isolator is situated in the Sonic Processor SEROPT board, which does not need power other than the above +5V for this purpose.

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

Although the application NEWFORM 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 a clean 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. 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 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. so that only chips 0 and 1 need erasure; this saves a lot of time 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 4MTO1M.EXE

Data are recovered from the file TEST by reading each (sequential) directory entry and using the contents to find the related block of data. Each block contains the 128 byte Argos message, followed by the 288 byte Meteosat message. A simple QBasic application REPFORM.BAS was written to read these messages and this can form the basis of a more specific application.

7. SPECIFICATION

7.1 Supplies

The supply to the Formatter is nominally 24V d.c.

7.2 Power Consumption

The nominal consumption is 66 mA (keyboard and VDU disconnected), i.e. 1.6 Watts

7.3 Data Storage and Output

The Argos and Meteosat messages are stored on a 4 Mbyte FlashCard, using a non-standard filing system; this uses one chip on the card for directory information (32 bytes per entry, 256 kbytes total, i.e. a maximum of 8k entries, or 85 days duration at 1/4 hourly intervals). The data (messages) are stored on the remaining 15 chips (416 bytes per entry, 3932160 bytes total, i.e. a maximum of 9452 entries, limited to 8192 by the directory space).

Data output via the COM1 port includes the above messages, which are output with a short header at 5, 20, 35 and 50 minutes past each hour. In addition messages are output for diagnostic purposes at the start of most function calls. For example, when a message is received by the Formatter from the Sonic processor, it will be checked by a call to the function MESSAGECHECK; this results in the output via COM1 of a message such as:

```
MCHK-1                  (the 1 in the first line signifies a Sonic message,
                         0 would be a Multimet message)
"S0093091309150100-1.6701.33-00.17+00.12-00.54341.70360-1.54-1.61E-003T"
                        (The string within the parentheses is the message received)
OK                      (the OK signifies that no errors wer found)
```

All statements in brackets are added comments, here and in the following examples.

The function headertime will then be called, giving an output such as:

```
HT-09/13 09:15:01          (part of header converted to MM/DD HH:mm:SS format)
Day 256: date 13/09: time 09:26:24          (Formatter Clock Date/Time stamp)
TSON 1                     (estimated difference between the Sonic and Formatter Real Time
                           Clocks, in minutes)
```

A similar process occurs after reception of a Multimet message, e.g.

```
MCHK-0
"S0093091309020100 1F8A18F50DF3183319111EF51FED200D0000000O000001D7216C210125
7E253E0000000000000000000000000000000000000000000000007A0000T"
OK
HT-09/13 09:02:01
Day 256: date 13/09: time 09:02:56
TMET 0                     (estimated difference between the Multimet and Formatter Real Time
                           Clocks, in minutes)
```

The function CONVERT will be called after a good Sonic message has been acquired or, if Sonic data is absent, on the quarter-hour; this function controls the averaging of the Multimet data over the appropriate period, using other functions. The listing below shows a typical diagnostic output resulting from convert, called after a Sonic message:

```
CON-HT-09/13 09:15:01          (CONVERT gets Sonic headertime)
SDAT:-OK                       (check Sonic message is OK)
HT-09/13 09:16:01              (get HEADERTIME of 1st Multimet message in database)
SHdrSt 367754:MMHdr 367756     (SHdrSt and MMHdr are Sonic and Multimet message start
                               times in minutes past the New Year, corrected for relative
                               clock timing errors)
S-                             ( Multimet is within 10 minute window, so call function SUM)
  437    0  8059  5967  9260  9325  8544  8680  3568 122
HT-09/13 09:17:01              (get HEADERTIME of 2nd Multimet message in database)
SHdrSt 367754:MMHdr 367757     (Multimet is again within 10 minute window, so SUM . .)
S-
  825    0 16161 13124 18509 18638 17094 17363  7136 122
HT-09/13 09:18:01
SHdrSt 367754:MMHdr 367758
S-
 1140    0 24238 19636 27378 24688 25652 26056 10739 122
HT-09/13 09:19:01
SHdrSt 367754:MMHdr 367759
S-
 1420    0 32294 25614 36638 34025 34215 34752 14339 122
HT-09/13 09:20:01
SHdrSt 367754:MMHdr 367760
S-
 1683    0 40394 32695 45911 43377 42782 43449 17939 122
HT-09/13 09:21:01
SHdrSt 367754:MMHdr 367761
S-
 2371    0 48504 40094 55185 52734 51353 52151 21533 122
HT-09/13 09:22:01
SHdrSt 367754:MMHdr 367762
S-
 2597    0 56587 46797 64466 62101 59930 60864 25126 122
HT-09/13 09:23:01
SHdrSt 367754:MMHdr 367763
S-
 3054    0 64639 52662 73781 71499 68515 69582 28715 121
HT-09/13 09:24:01
SHdrSt 367754:MMHdr 367764     (Multimet start time lies outside 10 minute window, not used)
HT-09/13 09:25:01
SHdrSt 367754:MMHdr 367765
```

```
HT-09/13 09:11:01
SHdrSt 367754:MMHdr 367751
HT-09/13 09:12:01
SHdrSt 367754:MMHdr 367752
HT-09/13 09:13:01
SHdrSt 367754:MMHdr 367753
HT-09/13 09:14:01
SHdrSt 367754:MMHdr 367754   (Multimet lies within 10 minute window, so SUM)
S-
 3501    0 72715 59133 83030 80822 77047 78244 32285 122
HT-09/13 09:15:01
SHdrSt 367754:MMHdr 367755
S-
 3871    0 80762 64818 92288 90155 85585 86916 35852 122
N2 10                        (end of summing process, 10 Multimet messages used)
Chan z[0] z[1] z[2] z[3] u[n] (display channel number, calibration coefficients and
                              mean value prior to application of calibration coeffs
                              for required channels)
11 +0.142 +0.0984 +0.00e+000 +0.00e+000  +7.742
10 +0.140 +0.0984 +0.00e+000 +0.00e+000  +0.000
 0 +0.000 +72.0000 +0.00e+000 +0.00e+000  +0.141
 1 +0.000 +72.0000 +0.00e+000 +0.00e+000  +2.088
15 -0.988 +0.0779 +1.50e-006 +9.98e-009 +184.576
14 -1.215 +0.0772 +6.91e-006 +8.80e-010 +180.310
13 -1.042 +0.0769 +7.53e-006 +2.21e-010 +171.170
12 -1.052 +0.0773 +7.48e-006 -8.36e-011 +173.832
 2 +0.000 +4.1000 +0.00e+000 +0.00e+000  +5.624
   0   0   0   0   0   0   0   1   1   1   (function COMP_MEAN sums variance in
                                            compass over N2 values, in degrees**2)
28 +121.900                  (compass channel number and final mean value)
 +0.9  +0.1 +10.2 +150.3 +13.5 +12.9 +12.3 +12.6 +23.1 +121.9   (calibrated values)
Q20 0 Q21 1 Q10 1 Q11 1      (quarter-hour parameters)
UPD:PACK-PPPPPPP             (call function UPDATE, which PACKs data into ARGOS
                              database, inserting PARITY check bits)
MET-FMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMTFMT
                             (call function METEO, which FORMATs data into Meteosat
                              database)
```

When an XON has been received from the Argos transmitter (PTT), a display similar to the following will occur:

```
ARG-1                              (XON received, data transfer using SEND_RS232)
Day 256: date 13/09: time 09:02:48 (time at which XOFF received from Argos PTT)
XOFF
```

At 5, 20, 35 and 50 minutes past the hour, the function RS232OP is called, giving a display similar to the following:

```
ARGOS0905            (header including time of output hhmm)
A36CE0EF14E890E5409E98E71169C0C6ACE812E340AE4CE893628096A0E8F7FF
145A207DECED95DF809E2EF19661409E40E817E340AE66EF18E5A0BEAAE9C7FF
9965E0B67AEF9A69E0DEC8E41BE820B690ED9CEA80CF02E91D6740CEB2EAC7FF
1E6760BECAEF9F6B60DEA8EBA06B60DEE2E9216D60E712F222EAA0DEDEF1C7FF
                     (4 lines of 32 bytes in hex ASCII, see Appendix A for definition)
METEO0905            (header including time of output hhmm)
B01
51005
256
32,-170,013,-001,+001,-005,-159,+126,+125,+117,+120,009,010
33,-162,014,+002,+005,-003,-147,+132,+128,+118,+121,010,009
34,-173,013,+000,+001,-003,-160,+136,+134,+119,+122,010,010
35,-164,014,-005,-003,-005,-146,+137,+136,+120,+123,011,011
231,122,000,-0000,+0001
                     (see Appendix B for message format definition)
```

The function DATA_SAVE is then called to save the databases to FlashCard; a display similar to the following will occur:

```
FLASH                    (function FLASHSAVE called)
Block 6 Ptr 58208        (start address of data saved = 65536*BLOCK + Ptr)
Final Saving 416 bytes, block 6, pointer 58208    (same as above, since data lies within
                                                   same BLOCK)
OK
Block 0 Ptr 14560        (start address of directory entry = 65536*BLOCK + Ptr)
Final Saving 32 bytes, block 0, pointer 14560
OK
```

The OKs signify that no write errors occurred in writing the data and directory entry to the FlashCard.

The other message type is:

MFILL-n

where n lies between 0 and 71; this results when a data transfer of 4 bytes to the Meteosat transmitter (DCP) is made (total 288 bytes).

It should be mentioned that, due to the consequences of interrupts, the above messages may be intermixed as higher priority tasks are performed.

APPENDICES

Appendix A Format of ARGOS messages

The ARGOS database contains four 32 byte message frames, these are cyclically transferred at 1200 baud to the Argos transmitter (PTT) when the latter issues an XON via its RS232 output to the Formatter COM5 port.

An example of the ARGOS database contents is given below as 4 frames of 32 bytes in hex ASCII format.

A36CE0EF14E890E5409E98E71169C0C6ACE812E340AE4CE893628096A0E8F7FF

145A207DECED95DF809E2EF19661409E40E817E340AE66EF18E5A0BEAAE9C7FF

9965E0B67AEF9A69E0DEC8E41BE820B690ED9CEA80CF02E91D6740CEB2EAC7FF

1E6760BECAEF9F6B60DEA8EBA06B60DEE2E9216D60E712F222EAA0DEDEF1C7FF

The 32 bytes of a frame are in a highly packed format, which contains five quarter-hourly sets of values of PSD, MWS, Fit A and Vertical MWS; thus a satellite pass will normally acquire all four frames, i.e. twenty quarter-hourly sets, or 5 hours of data. There is, consequently, a 150% degree of redundancy in the acquisition of data from an assumed 12 satellite passes per day, although not all satellite passes may be at sufficient elevation to acquire all four frames.

We shall denote the five quarter-hourly sets in a frame by suffices a - e

Each quarter-hourly set of data in a message contains:

Q — time of data acquisition period start in quarter-hours since midnight
(this has the range 0 to 95, which can be expressed as a 7 bit binary number
bits Q0 (lsb) to Q6 (msb), with an added even parity bit PQ)

PSD — log10(Power Spectral Density * f^5/3
(this is converted to a 9 bit binary value 000h to 1FFh,
bits PSD0 (lsb) to PSD8 (msb), with added parity bit PPSD,
by taking the remainder of [(100 * (6 + PSD) - 0.5) divided by 512].
This gives a nominal range of -6.00 to -0.89, for 000h to 1FFh,
although secondary ranges, such as -0.88 to +4.23, exist)

MWS — Mean Wind Speed
(this is converted to a 9 bit binary value 000h to 1FFh,
bits MWS0 (lsb) to MWS8 (msb), with added parity bit PMWS,
by taking the remainder of [(10 * MWS + 0.5) divided by 512]

This gives the nominal range of 0.0 to 51.1 m/s, for 000h to 1FFh,
although secondary ranges, such as 51.2 to 102.3, exist)

Fit_A Coefficient 'a', for linear regression fit of PSD vs log10(frequency)
over the frequency range 2 - 4 Hz, PSD = a + b.log10(frequency)
(this is converted to a 9 bit binary value 000h to 1FFh,
bits Fit_A0 (lsb) to Fit_A8 (msb), with added parity bit PFit_A, as for PSD)

V_M Vertical Mean Wind Speed
(this is converted to a 9 bit binary value 000h to 1FFh,
bits V_M0 (lsb) to V)M8 (msb), with added parity bit PV_M,
by taking the remainder of [(50 * V_M + 256.5) divided by 512]
This gives a nominal range of -5.12 to +5.11, for 000h to 1FFh,
although secondary ranges, such as +5.12 to +15.34, exist)

A quarter-hourly set of the above parameters amounts to 48 bits (6 bytes), so that 5 sets amount to 30 bytes, bytes 1 to 30, leaving 2 bytes in the frame free for additional data. These two bytes are used to convey the number of Multimet messages successfully used in the averaging process over the Sonic acquisition period, N2. However, since N2 has the range 0 to 10 (4 bit binary), we can not fit five x 4 bits into 2 bytes and we have to subtract 3 from the N2 values, setting negative values to 0. i.e. the resulting (N2-3) range is 0 to 7 (3 bit binary, bits N22 - N20), leaving one bit (PN2a-e) free for an overall even parity check for the two bytes.

The data format is summarised in tabular form below, showing each byte as one line with the most significant bit to the left, from byte1 to byte32.

PQa	Q6a	Q5a	Q4a	Q3a	Q2a	Q1a	Q0a
PPSDa	PSD8a	PSD7a	PSD6a	PSD5a	PSD4a	PSD3a	PSD2a
PSD1a	PSD0a	PMWSa	MWS8a	MWS7a	MWS6a	MWS5a	MWS4a
MWS3a	MWS2a	MWS1a	MWS0a	PFit_Aa	Fit_A8a	Fit_A7a	Fit_A6a
Fit_A5a	Fit_A4a	Fit_A3a	Fit_A2a	Fit_A1a	Fit_A0a	PV_Ma	V_M8a
V_M7a	V_M6a	V_M5a	V_M4a	V_M3a	V_M2a	V_M1a	V_M0a
PQb	Q6b	Q5b	Q4b	Q3b	Q2b	Q1b	Q0b
PPSDb	PSD8b	PSD7b	PSD6b	PSD5b	PSD4b	PSD3b	PSD2b
PSD1b	PSD0b	PMWSb	MWS8b	MWS7b	MWS6b	MWS5b	MWS4b
MWS3b	MWS2b	MWS1b	MWS0b	PFit_Ab	Fit_A8b	Fit_A7b	Fit_A6b
Fit_A5b	Fit_A4b	Fit_A3b	Fit_A2b	Fit_A1b	Fit_A0b	PV_Mb	V_M8b
V_M7b	V_M6b	V_M5b	V_M4b	V_M3b	V_M2b	V_M1b	V_M0b
PQc	Q6c	Q5c	Q4c	Q3c	Q2c	Q1c	Q0c
PPSDc	PSD8c	PSD7c	PSD6c	PSD5c	PSD4c	PSD3c	PSD2c
PSD1c	PSD0c	PMWSc	MWS8c	MWS7c	MWS6c	MWS5c	MWS4c
MWS3c	MWS2c	MWS1c	MWS0c	PFit_Ac	Fit_A8c	Fit_A7c	Fit_A6c
Fit_A5c	Fit_A4c	Fit_A3c	Fit_A2c	Fit_A1c	Fit_A0c	PV_Mc	V_M8c
V_M7c	V_M6c	V_M5c	V_M4c	V_M3c	V_M2c	V_M1c	V_M0c
PQd	Q6d	Q5d	Q4d	Q3d	Q2d	Q1d	Q0d
PPSDd	PSD8d	PSD7d	PSD6d	PSD5d	PSD4d	PSD3d	PSD2d
PSD1d	PSD0d	PMWSd	MWS8d	MWS7d	MWS6d	MWS5d	MWS4d
MWS3d	MWS2d	MWS1d	MWS0d	PFit_Ad	Fit_A8d	Fit_A7d	Fit_A6d
Fit_A5d	Fit_A4d	Fit_A3d	Fit_A2d	Fit_A1d	Fit_A0d	PV_Md	V_M8d
V_M7d	V_M6d	V_M5d	V_M4d	V_M3d	V_M2d	V_M1d	V_M0d
PQe	Q6e	Q5e	Q4e	Q3e	Q2e	Q1e	Q0e
PPSDe	PSD8e	PSD7e	PSD6e	PSD5e	PSD4e	PSD3e	PSD2e
PSD1e	PSD0e	PMWSe	MWS8e	MWS7e	MWS6e	MWS5e	MWS4e
MWS3e	MWS2e	MWS1e	MWS0e	PFit_Ae	Fit_A8e	Fit_A7e	Fit_A6e
Fit_A5e	Fit_A4e	Fit_A3e	Fit_A2e	Fit_A1e	Fit_A0e	PV_Me	V_M8e
V_M7e	V_M6e	V_M5e	V_M4e	V_M3e	V_M2e	V_M1e	V_M0e
PN2a-e	N22a	N21a	N20a	N22b	N21b	N20b	N22c
N21c	N20c	N22d	N21d	N20d	N22e	N21e	N20e

Table A.1 Bit Map of 32 Byte ARGOS Frame

Software has been produced to carry out the inverse process, decoding a received ARGOS message into the parameters in engineering units; this is described in a separate document.

Appendix B Format of Meteosat messages

The Meteosat database contains 288 bytes in the International Alphabet 5 subset of ASCII; these are transferred at 100 baud to the Meteosat transmitter (DCP) via the Formatter COM6 port in 72 transfers of 4 bytes (to prevent the transfer from slowing the NEWFORM loop execution too much).

An example of the Meteosat database contents is given below:

B01

51005

256

32,-170,013,-001,+001,-005,-159,+126,+125,+117,+120,009,010

33,-162,014,+002,+005,-003,-147,+132,+128,+118,+121,010,009

34,-173,013,+000,+001,-003,-160,+136,+134,+119,+122,010,010

35,-164,014,-005,-003,-005,-146,+137,+136,+120,+123,011,011

231,122,000,-0000,+0001

The format of this message is quite simple; the first three lines are, respectively, the buoy ID, the niminal latitude (51° North) and longditude (005° West) and the Julian Day number (256).

The next four lines include a combination of Sonic and Multimet data in the format:

QQ,+PSD,MWS,N_M,E_M,V_M,FtA,AT1,AT2,ST1,ST2,YW1,YD1

where QQ = Quarter-hours since midnight (range 00 - 96)

+PSD = 100 * log10(Power Spectral Density * f^5/3

MWS = 10 * (Mean Wind Speed in m/s)

N_M = 10 * (North Mean component of Wind Speed in m/s)

E_M = 10 * (East Mean component of Wind Speed in m/s)

V_M = 10 * (Vertical Mean component of Wind Speed in m/s)

FtA = 100 * Coefficient 'a', for linear regression fit of PSD vs log10(frequency)

(over the frequency range 2 - 4 Hz, PSD = a + b.log10(frequency))

AT1 = 10 * (Mean Air Temperature from sensor 1 in °C)

AT2 = 10 * (Mean Air Temperature from sensor 2 in °C)

ST1 = 10 * (Mean Sea Temperature from sensor 1 in °C)

ST1 = 10 * (Mean Sea Temperature from sensor 2 in °C)

YW1 = 10 * (Mean Young AQ1 Wind Speed in m/s)

YD1 = Mean Young AQ1 Wind Direction in degrees.

The final line includes housekeeping data in the format:

BBB,HHH,HSD,+TMET,+TSON

where

BBB = 10 * Mean Battery Voltageon the 24V bus

HHH = Mean Buoy Heading in degrees magnetic

HSD = Standard Deviation of Heading in degrees

+TMET = Time difference between Multimet and Formatter Real Time Clocks

(+ve for Multimet clock ahead of Formatter clock)

+TSON = Time difference between Sonic and Formatter Real Time Clocks

(+ve for Sonic clock ahead of Formatter clock)

Appendix C SETTIME Source Code

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 Formatter and Raw Data GCAT processors. It allows the
        GCATclock to be reset at re-boot time by connecting a PC
        runn ing the GWBasic program SONTIM.BAS to the COM1 port.
        The GCAT Date/Time are then set to the PC Date/Time.
        If the PC is not connected, this program times out.
        The autoexec then runs the main application.

        Note that the GCAT's SETUP must be entered on initial power-up
        (using the F2 key to interrupt the boot sequence) if this method of
        Date/Time setting is to have any effect.
\*************************************************************************/

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

extern void uart_on(void);        /* in SETTIM.ASM */
extern void uart_off(void);       /* in SETTIM.ASM */

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);              /* get Date in the dosdate_t structure date */
_dos_gettime(&time);              /* get Time in the dostime_t structure time */
```

```
        strcpy(rsout, "Date: ");          /* build the Date/Time string for output to the ext PC */

        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++)
        {
        /* wait until transmit holding register is empty (100 tries) */
        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)      /* port timed out */
            {
            exit(0);
            }
        if ((status & 0x8000) == 0x8000)                  /* this never applies, remove it */
            {
            printf("RS232 COM%d timed out\n", port + 1);
            break;
            }
        }

    ch = 0;
    loop_ctr = 0L;
```

```
/* get characters from external PC, if present, until a <LF> is received or time limit reached */
do
    {
    status = 0x100 & _bios_serialcom(_COM_STATUS, port, 0);

    if (status == 0x100)                        /* received data is ready */
        {
        ch_hit = 0xff & _bios_serialcom(_COM_RECEIVE, port, 0);        /* get a character */
        printf("%c", ch_hit);
        if (ch_hit == 68)                       /* capital D i.e. start of message */
            {
            ch = 0;
            }
        stbuf[ch] = (char) ch_hit;              /* message put in stbuf */
        ch++;
        }
    loop_ctr++;
    }
while ( (ch_hit != 10) && (loop_ctr < 100000L) );

stbuf[ch] = 0;                                  /* string terminator */

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

/* load dosdate_t and dostime_t structures with ext PC date/time */
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 a genuine message was received */
    {
    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: ");        /* build new Date/Time string for output to the ext PC */

    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");                    /* omit this Q, it makes ext PC send another
                                                  Date/Time message, which is not serviced */

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

        for (ch = 0; ch < strlen(rsout); ch++)
            {
            /* wait until transmit holding register is empty (100 tries) */
            do
                {
                status = 0x2000 & _bios_serialcom(_COM_STATUS, port, 0);
                }
            while (status != 0x2000);

            _bios_serialcom(_COM_SEND, port, rsout[ch]);
            if ((status & 0x8000) == 0x8000)          /* this never applies, remove it */
                {
                printf("RS232 COM%d timed out\n", port + 1);
                break;
                }
            }
        }
    uart_off();
    }
```

Assembly Code

```
;*******************************SETTIM.ASM*******************************
;    Assembly Code functions used to enable GCAT ports
;
;          for use in conjunction with SETTIME.C
;
;          assemble using MASM /MX 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 D NEWFORM Schematic and Source Code

Figure D.1
NEWFORM Flow Chart

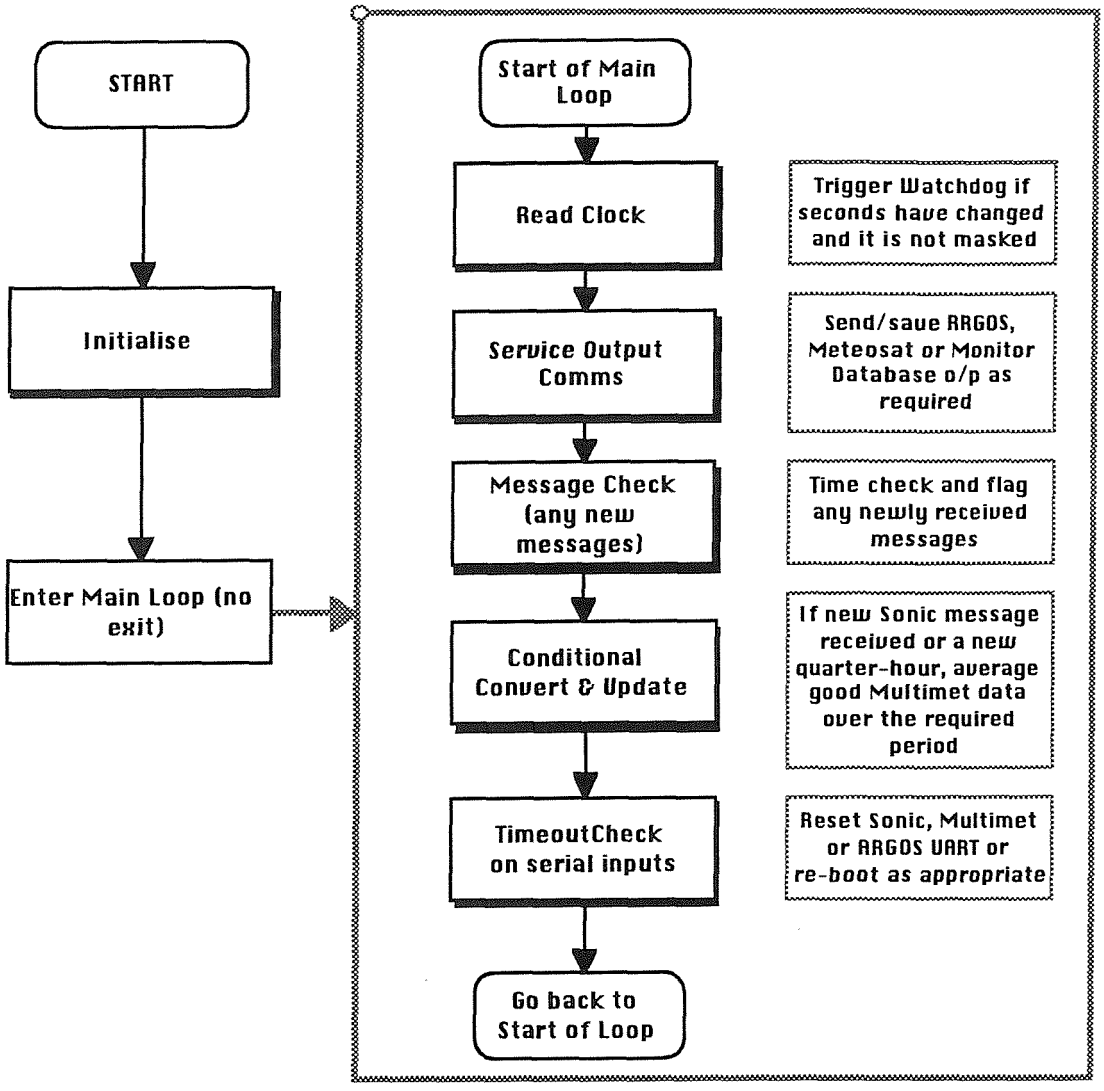


Figure D.2
NEWFORM Flow Chart - Initialise

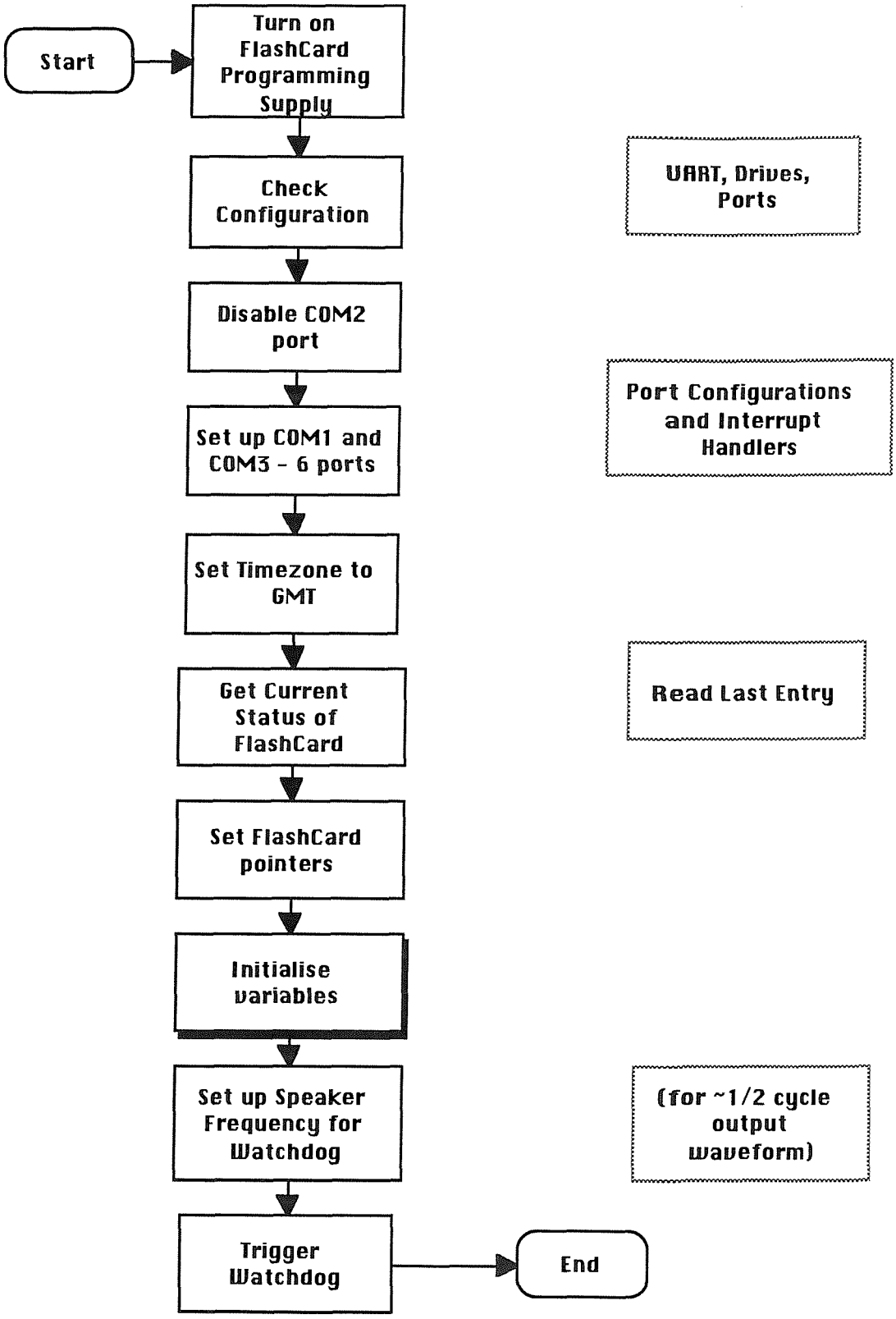


Figure D.3
NEWFORM Flow Chart - Initialise/Initialise variables

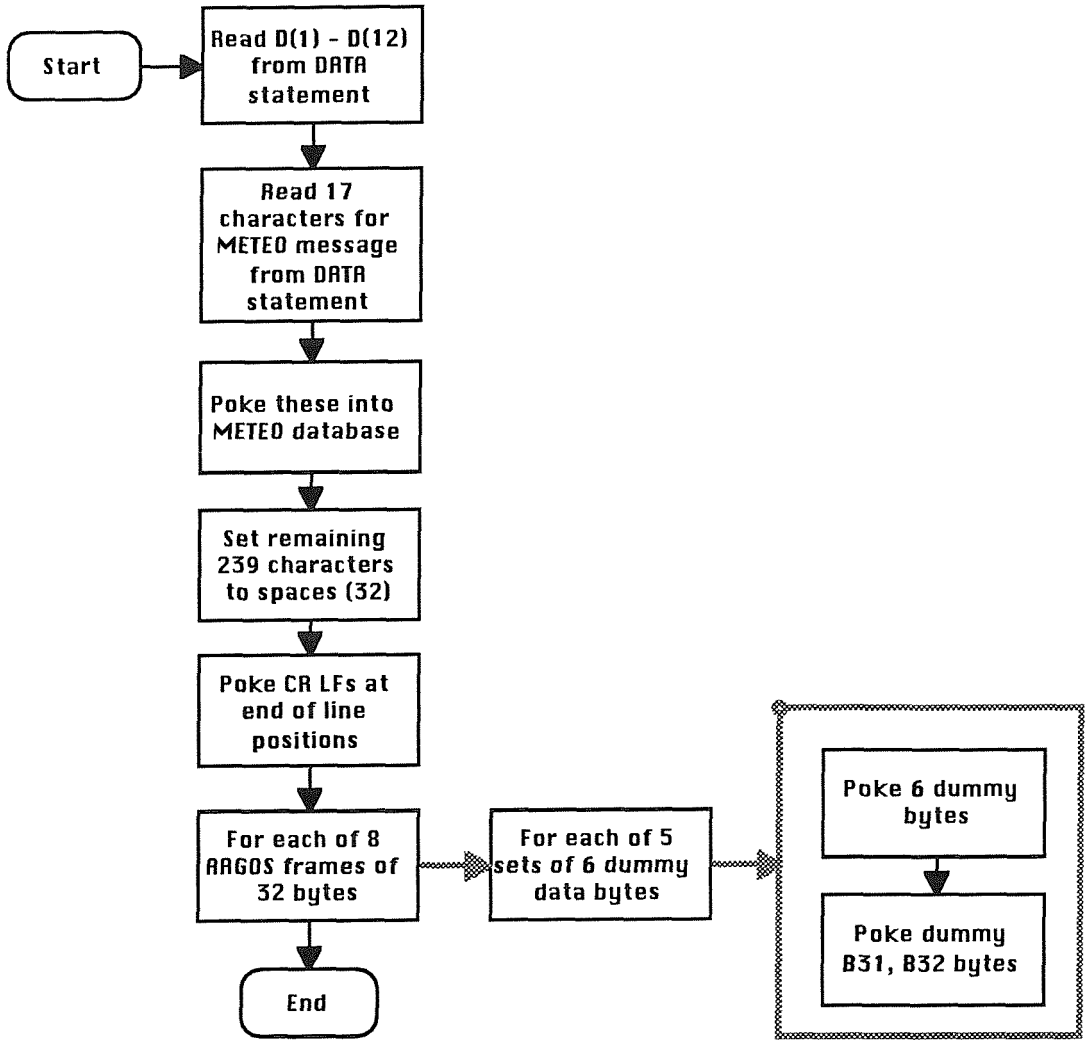


Figure D.4
NEWFORM Flow Chart - Read Clock

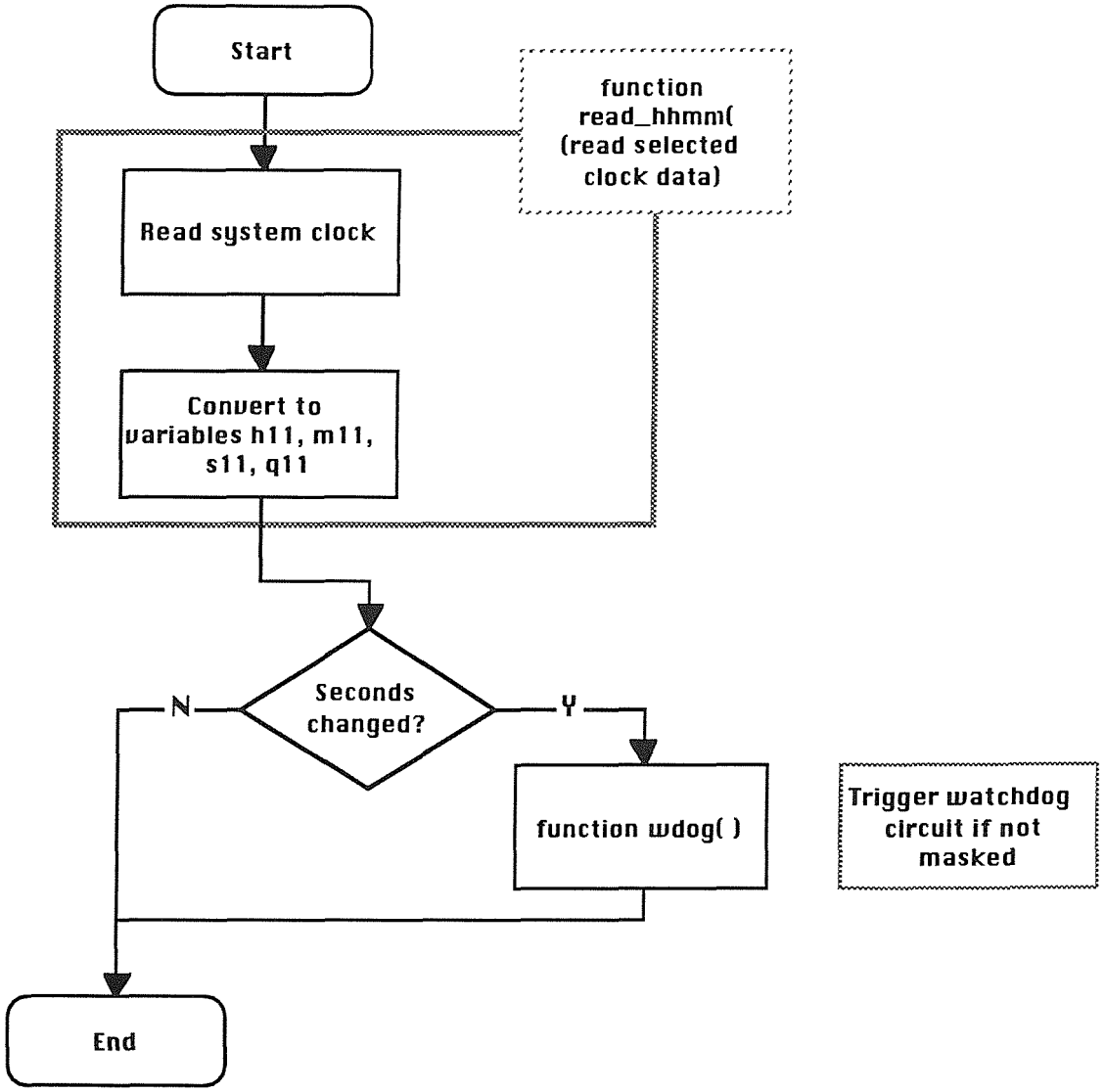


Figure D.5
NEWFORM Flow Chart - Service Output Comms

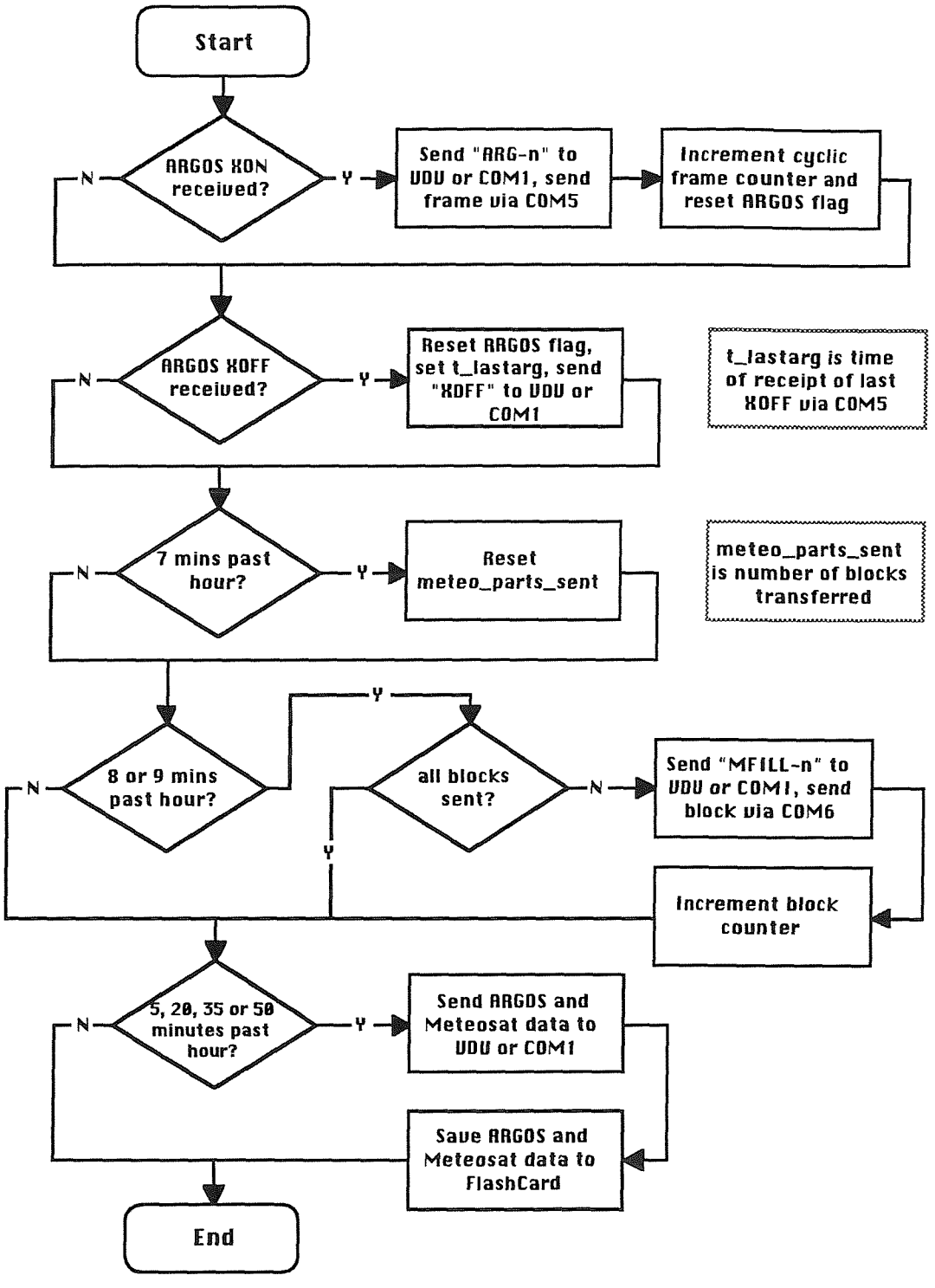


Figure D.6
NEWFORM Flow Chart - Message Check

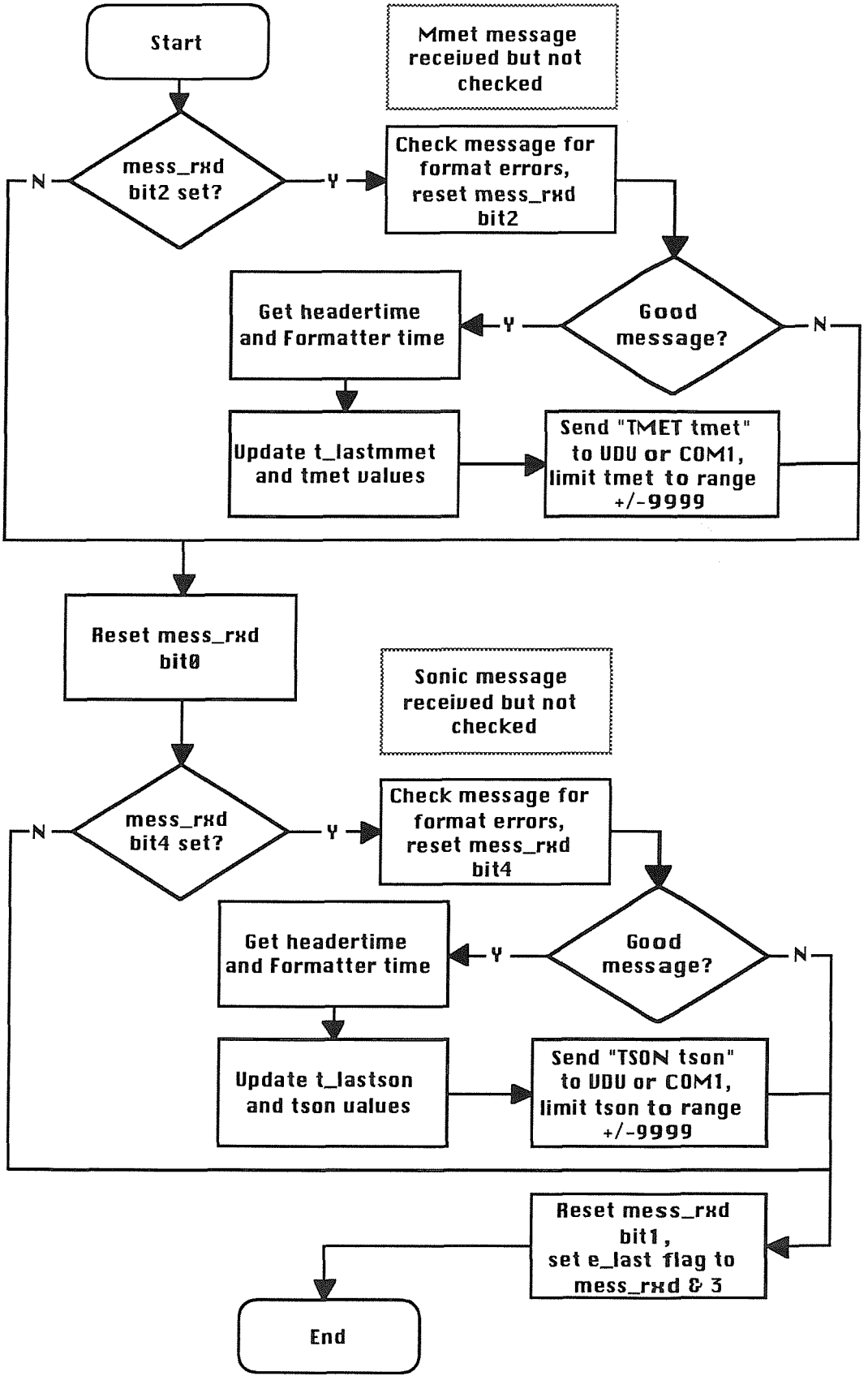


Figure D.7
NEWFORM Flow Chart - Conditional Convert & Update

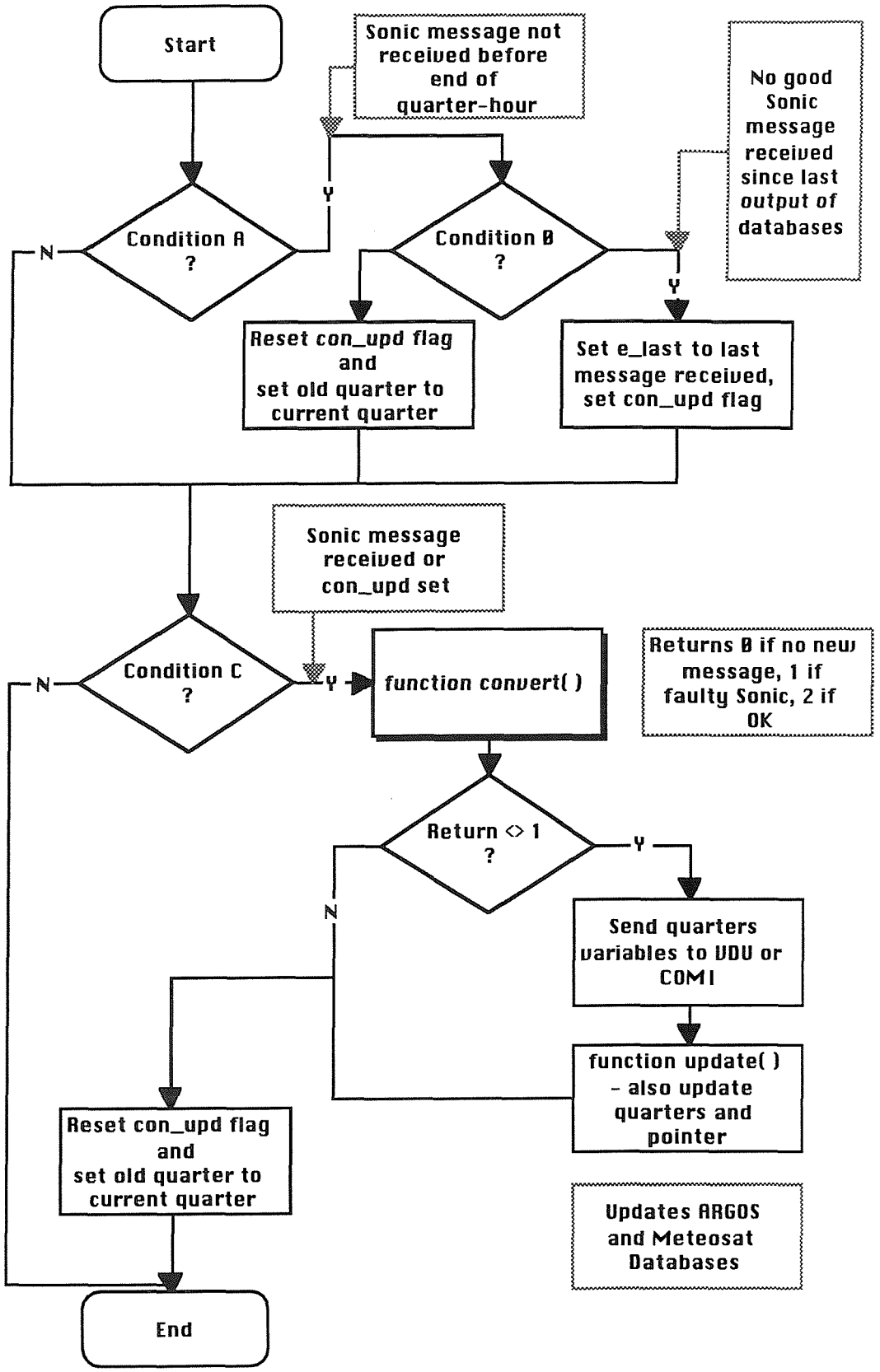


Figure D.8
NEWFORM Flow Chart - Conditional Convert & Update/function convert( )

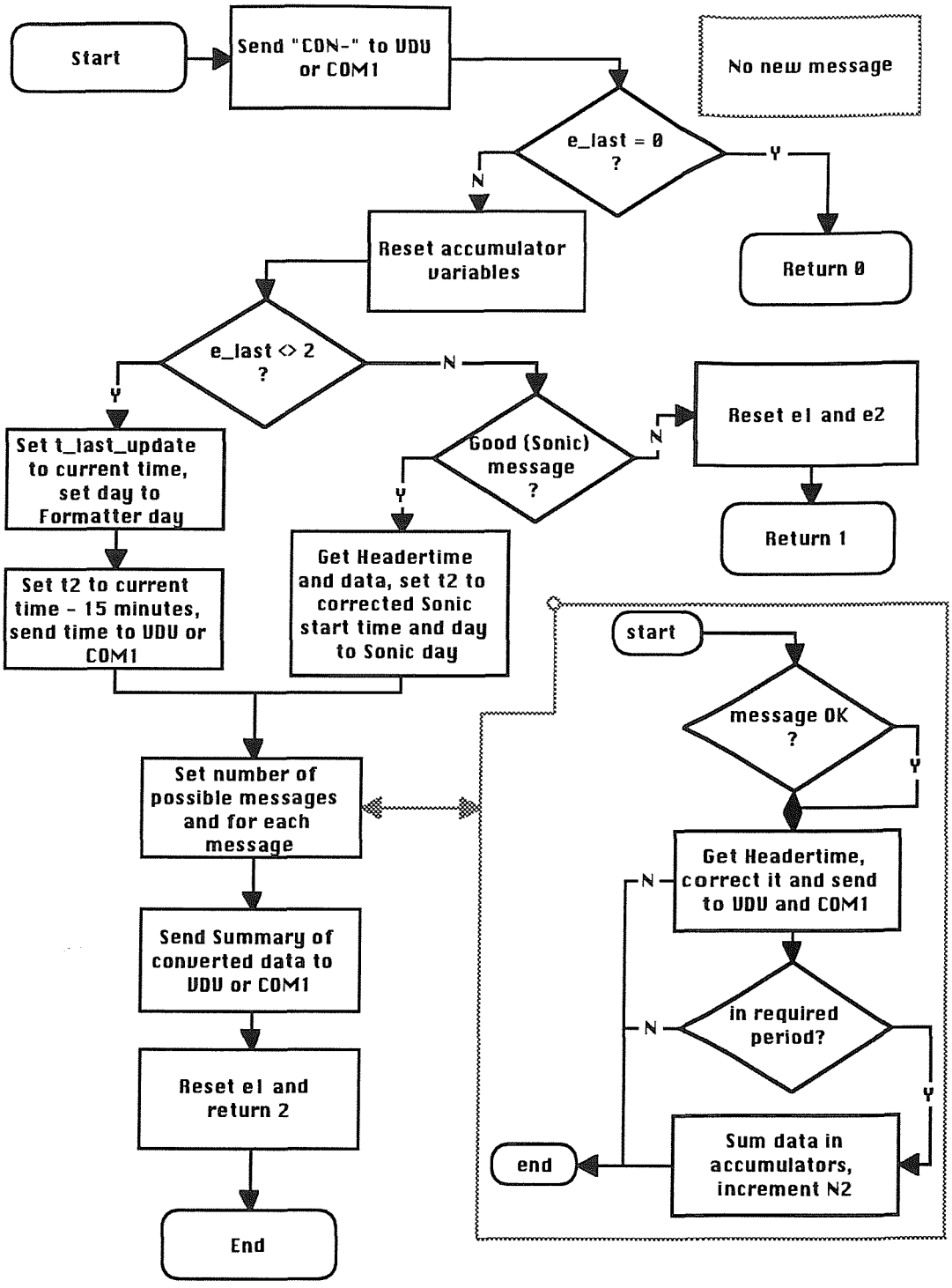
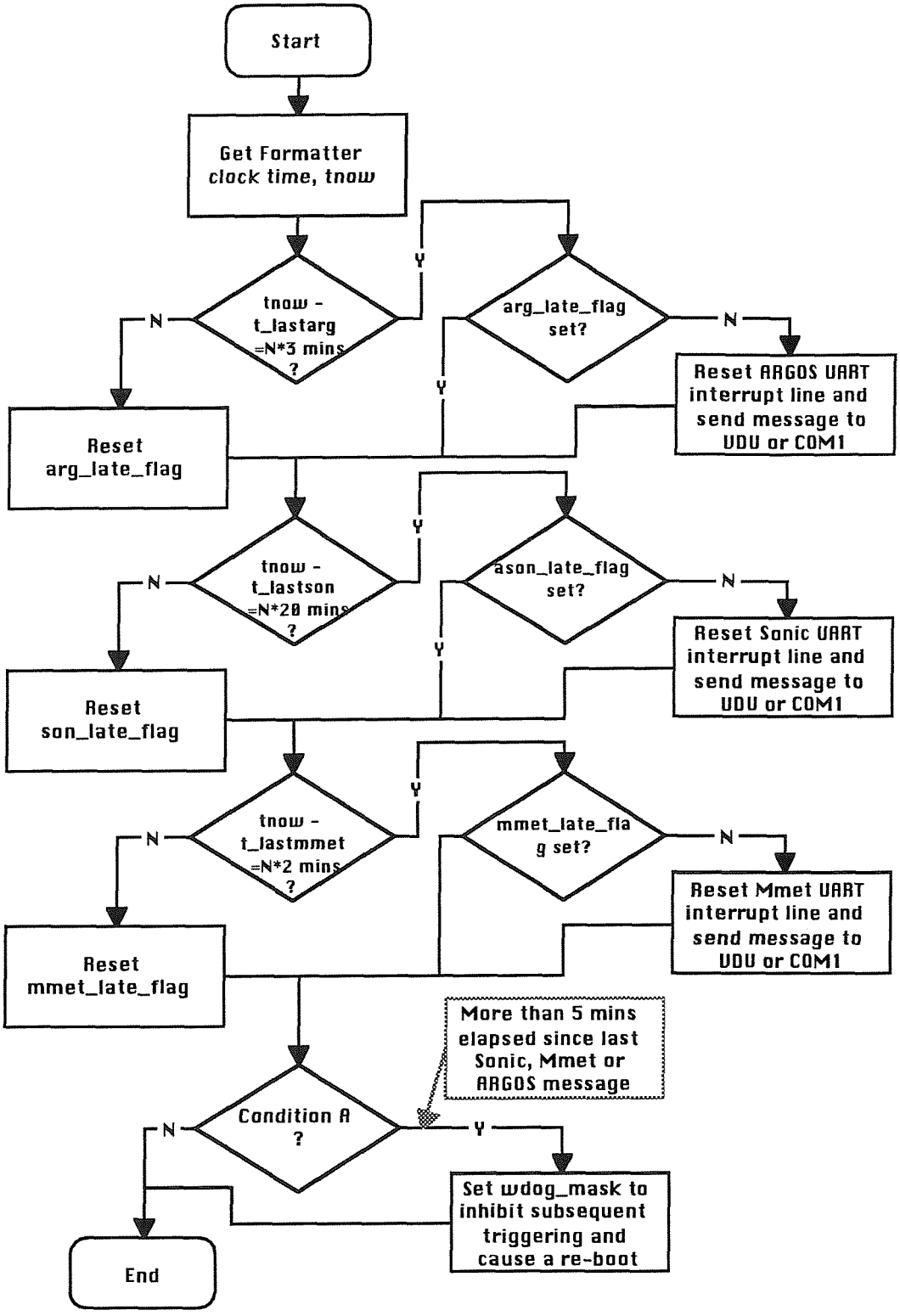


Figure D.9
NEWFORM Flow Chart - Timeout Check



C Source Code

```
/*******************************NEWFORM1.C*******************************
* Program for new GCAT formatter to replace Sonic Buoy 1802 formatter
*
* (uses 2 x Ampro Minimodules SSP (dual port serial + 1 port parallel)
* for COM3 - COM6 ports (and GCAT2000 82C710 UART for COM1 port)
* (could not get GCAT3000 COM2 port to work with external ports)
*
* This vn of NEWFORM.C includes save to flashcard and must be made using
* the batch file C1.BAT which executes NMAKE F1 to link the flash code
*
* Author CHC   21/9/1993
*
* Working version used on SWALES October 1993
*
* other versions of newform for temporary test only, e.g. newform3.c
*************************************************************************/

#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"form_var.c"          /* declares/initialises arrays */
#include"coms.c"              /* Port & UART register definitions */

#define MON_PORT     1        /* COM1 o/p only for monitoring via SEROPT */
#define SONIC_PORT   3        /* COM3 interrupt IRQ4 driven */
#define MMET_PORT    4        /* COM4 interrupt IRQ7 driven */
#define ARGOS_PORT   5        /* COM5 interrupt IRQ3 driven */
#define METEO_PORT   6        /* COM6 o/p only with h/w handshake */

#define TRUE  1
#define FALSE 0
#define DISPLAY FALSE
/* set to TRUE to enable output to VDU */
/* set to FALSE for output via COM1 port */

#define BIN_TO_RAD 2. * 3.14159 / 256.      * converts 8 bit rdg to radians */
#define RAD_TO_DEG 360. / (2. * 3.14159)    /* converts radians to degrees */
#define BIN_TO_DEG 360. / 256.              * converts 8 bit rdg to degrees */

#define ARGOS_LENGTH  128                   /* 4 x 32 byte messages */
#define METEO_LENGTH  288                   /* 17 + 4*61 + 27 bytes */
#define REC_LENGTH    416L                  /* ARGOS + METEO */
#define SONIC_REC_LENGTH 10                 /* 10 minutes for 12+1024 samples */

/* 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 */

#pragma check_stack(on)
#pragma check_pointer(on)
```

```
/************************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 send_rs232(unsigned, char *, int, int *);
void wdog(void);
void meteofill(void);
void rs232op(void);
void messagecheck(int);
int convert(void);
void update(void);
void headertime(char **);
int ival(char *, int);
double fval(char *, int, int);
void sdat(char **);
void sum(char **);
void pack(void);
int parity(int);
void n2bits(char *, char *);
void meteo(void);
void format(char **, unsigned *, double, char *);
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(void);
double comp_mean(double *);
void send1(char *);


/* 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)();
void interrupt far our_irq7_handler(void);
void (interrupt far *old_irq7_handler)();
```

```
/*****************************GLOBAL VARIABLES*************************/
char data_buffer[ARGOS_LENGTH + METEO_LENGTH + 10];
char display_buffer[144];
char header_contents[40];
char julian[12];

double psd, sonic_mws, north_mean, east_mean;
double vert_mean, c_mean, heading, heading_scatter, fit_a, fit_b;
double comp_rad, sin_total, cos_total;
int argos_parts_sent = 0, meteo_parts_sent = 0, argos_send_flag = 0;
int s1, m1, h1, d1, n1, n2;
int s10, s11, m10, m11, h10, h11, h21, q10, q11, q20, q21, jt1;
int e_last, e1, e2 = 0;
int v10, v20, v30, v40;
int jd1, jd2;
int flash_full, full_flag = 0, logging;
int mess_rxd = 0, wdog_mask = 0;

ldiv_t t_check;
long t_last_update;
time_t t_lastarg = 0L, t_lastson = 0L, t_lastmmet = 0L, tnow, tmet, tson;

unsigned compass[16];
unsigned new_ints, old_ints;
unsigned header_block, header_startptr;
unsigned reclen, record_no, segment, seg_ptr, start_block, start_offset;

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


/*****************************START OF MAIN*****************************/
main()
{
char *ptr;
char **g1 = &ptr;

int con_upd, n;
int arg_late_flag = 0, son_late_flag = 0, mmet_late_flag = 0;

union REGS regs;

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

/* the following CREG gets/writes are for test purposes mainly */
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);
sprintf(display_buffer, "CREG 0Fh before init: %x\n\r", regs.h.al);

#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#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);
sprintf(display_buffer, "PC/CHIP Options: %x\n\r", regs.h.al);
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

/* normally returns 0x02, i.e. drive B is PCMCIA */
sprintf(display_buffer, "82C710 Options: %x\n\r", regs.h.ah);
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

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

/* disable 82C710 serial (COM1) and parallel ports
regs.h.ch = regs.h.ah & 0xf3;
regs.h.cl = regs.h.al;
regs.h.ah = 0x08;
regs.h.bl = 1;
int86(0x1f, &regs, &regs); */



/* this disables the COM2 (IRQ3) UART in the 8680 */
regs.h.ah = 0x14;
regs.h.bh = 0x0f;
regs.h.al = 0x0e;
regs.h.bl = 1;
int86(0x1f, &regs, &regs);


/* set up the COM ports */
if (init_coms() == 0)
    {
    sprintf(display_buffer, "Exiting program, COMS error\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    exit(0);
    }
```

```
/* need to set timezone to GMT */
if (putenv("TZ=GMT") == -1)
    {
    sprintf(display_buffer, "Error in setting TZ\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #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;
sprintf(display_buffer, "SDATA 0A: %02x\n\r", n);
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif


if (n & 0x0c)                          /* Card Detect lines bits 2&3 should be low */
    {
    sprintf(display_buffer,
            "***************Flash Card not inserted***************\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    logging = 0;
    flash_full = 1;
    }
else
    {
    logging = 1;
    flash_full = 0;
    /* Find last directory entry */
    locn = seek_end(DIR_CHIP);
    sprintf(display_buffer, "Flash dir ptr:%lx\n\r", locn);
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
```

```
    {
    send1(display_buffer);
    }
#endif

segment = (unsigned) (locn >> 16);
seg_ptr = (unsigned) (locn & 0xffff);
dir_ptr = locn;
if (locn == DIRECTORY_START)
    {
    sprintf(display_buffer, "Virgin FlashCard\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif


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

    sprintf(display_buffer, "Last Directory Entry:- Segment %x, Offset %x\n\r",
                                segment, seg_ptr);
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #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);
            }
        #else
            {
            send1(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;
        sprintf(display_buffer,
        "Last Record:- Block %x, Offset %x, Length %lx\n\rFlash data ptr %lx\n\r",
                header_block, header_startptr, header_reclength, fl_ptr);
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif

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

/* initialise some variables */
readclock(1);
h10 = h1;
q10 = m1 / 15;
m10 = m1;
e_last = 0;
e1 = 0;
tnow =  ((24L * jt1) + (long) h1) * 60L + (long) m1 + (30L + s1) / 60L;
t_lastson = tnow;
t_lastmmet = tnow;
t_lastarg = tnow;

outp(0x43, 0xb6);
outp(0x42, 0);
outp(0x42, 40);

/* trigger watchdog circuit */
wdog();
```

```
/*************************************************************************
****************************START OF CONTINUOUS LOOP*********************
*************************************************************************/
while (kbhit() == 0)
    {
    read_hhmm();
    if (s11 != s10)
        {
        wdog();
        s10 = s11;
        }
    if (argos_send_flag == 0x11)
        {
        sprintf(display_buffer, "ARG-%d\n\r", argos_parts_sent);
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        send_rs232(ARGOS_PORT, b3, ARGOS_LENGTH, &argos_parts_sent);
        argos_parts_sent &= 3;                  /* cyclic txn of 4 frames of 32 bytes */
        argos_send_flag = 0;
        }
    if (argos_send_flag == 0x13)
        {
        argos_send_flag = 0;
        readclock(1);
        t_lastarg =  ((24L * jt1) + (long) h1) * 60L + (long) m1 + (30L + s1) / 60L;

        sprintf(display_buffer, "XOFF\n\r");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        }

    meteofill();                                /* tests for appropriate time to txfr data */

    /* send monitor data at 5, 20, 35 and 50 minutes past the hour */

    if ( (div(m11, 15).rem == 5) && (m11 != m10) )
        {
        rs232op();
        data_save();
        m10 = m11;
        e2 = 0;
        }

    if (mess_rxd & 0x04)                        /* complete Mmet message received,
                                                   but not yet checked */
        {
        messagecheck(0);
        mess_rxd &= 0x0b;                       /* mask out bit 2 */
        *g1 = b1 + l[0] * g[0];                 /* pointer to start of message */
```

```
    if ( *(*g1 + l[0] -1) == 0 )                    /* message terminator OK */
        {
        headertime(g1);                             /* tnow = minutes since new year */
        tmet = tnow + 1L;
        readclock(1);
        tnow =  ((24L * jt1) + (long) h1) * 60L + (long) m1 + (30L + s1) / 60L;
        t_lastmmet = tnow;
        tmet -= tnow;
        sprintf(display_buffer, "TMET %ld\n\r", tmet);
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        if (tmet > 9999L)
            {
            tmet = 9999L;
            }
        if (tmet < -9999L)
            {
            tmet = -9999L;
            }
        }
    }
else
    {
    mess_rxd &= 0x0e;                               /* mask out bit 0 */
    }

if (mess_rxd & 0x08)                                /* complete Sonic message received,
                                                       but not yet checked */
    {
    messagecheck(1);
    mess_rxd &= 0x07;                               /* mask out bit 4 */
    *g1 = b2 + l[1] * g[1];                         /* pointer to start of message */

    if ( *(*g1 + l[1] -1) == 0 )                    /* message terminator OK */
        {
        headertime(g1);                             /* tnow = minutes since new year */
        tson = tnow + 12L;                          /* estimated message txn 12 mins
                                                       after acq start */
        readclock(1);
        tnow =  ((24L * jt1) + (long) h1) * 60L + (long) m1 + (30L + s1) / 60L;
        t_lastson = tnow;
        tson -= tnow;
        sprintf(display_buffer, "TSON %ld\n\r", tson);
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
```

```
            if (tson > 9999L)
                {
                tson = 9999L;
                }
            if (tson < -9999L)
                {
                tson = -9999L;
                }
            }
        }
    else
        {
        mess_rxd &= 0x0d;                   /* mask out bit 1 */
        }
    e_last = mess_rxd & 0x03;

    if ( !(e_last & 2) && (q11 != q10) )    /* sonic not recvd before new quarter */
            {
            if (e2 == 0)
            /* no good Sonic message processed since databases output */
                {
                e_last = e1;                /* set e_last to source of last message
                                               received */
                con_upd = 1;                /* set flag to init a convert/update process */
                }
            else
                {
                con_upd = 0;
                q10 = q11;
                }
            }

    if ( (e_last & 2) || (con_upd == 1) )   /* future logic change possible */
        {
        if (convert() != 1)                 /* not a faulty Sonic message */
            {
            /* temp addition of o/p for debug */
            sprintf(display_buffer,
                "Q20 %d Q21 %d Q10 %d Q11 %d\n\r", q20, q21, q10, q11);
            #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
            #else
                {
                send1(display_buffer);
                }
            #endif
            update();

            q10 = q11;
            q20 = q21;
            c[2]++;
            }
        q10 = q11;
        con_upd = 0;
        }
        readclock(0);
        tnow =  ((24L * jt1) + (long) h1) * 60L + (long) m1 + (30L + s1) / 60L;
```

```
t_check = ldiv(tnow - t_lastarg, 3L);
if ( (t_check.rem == 0L) && (t_check.quot != 0) )
    {
    if (!arg_late_flag)
        {
        /* reset UART interrupt line */
        inp(COM5_BASE + RX_BUFF_REG);
        inp(COM5_BASE + LINE_STATUS_REG);
        outp(0x20, 0x67);
        sprintf(display_buffer, "Argos Late\r\n");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        arg_late_flag = 1;
        }
    }
else
    {
    arg_late_flag = 0;
    }
t_check = ldiv(tnow - t_lastson, 20L);
if ( (t_check.rem == 0L) && (t_check.quot != 0) )
    {
    if (!son_late_flag)
        {
        /* reset UART interrupt line */
        inp(COM3_BASE + RX_BUFF_REG);
        inp(COM3_BASE + LINE_STATUS_REG);
        outp(0x20, 0x64);                    /* clear in-service register bit */
        sprintf(display_buffer, "Sonic Late\r\n");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        son_late_flag = 1;
        }
    }
else
    {
    son_late_flag = 0;
    }

t_check = ldiv(tnow - t_lastmmet, 2L);
if ( (t_check.rem == 0L) && (t_check.quot != 0) )
    {
    if (!mmet_late_flag)
        {
        /* reset UART interrupt line */
        inp(COM4_BASE + RX_BUFF_REG);
        inp(COM4_BASE + LINE_STATUS_REG);
        outp(0x20, 0x63);
        sprintf(display_buffer, "Mmet Late\r\n");
```

```
                #if DISPLAY == TRUE
                    {
                    printf(display_buffer);
                    }
                #else
                    {
                    send1(display_buffer);
                    }
                #endif
                mmet_late_flag = 1;
                }
            }
        else
            {
            mmet_late_flag = 0;
            }

        if ( ((tnow - t_lastson) > 300L) || ((tnow - t_lastmmet) > 300L)
                || ((tnow - t_lastarg) > 300L) )
            {
            wdog_mask = 1;
            }
    }
/*********************************************************************
************************END OF CONTINUOUS LOOP*********************
*********************************************************************/

clean_up();
return 0;
}
/*************************END OF MAIN*****************************/


/*********************START OF FUNCTIONS**********************/

/******** READCLOCK gets time/date from system clock ********/
void readclock(int show_time)
    {
    struct tm *tmnow;

    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;
    jt1 = tmnow->tm_yday;
    sprintf(display_buffer, "Day %d: date %02d/%02d: time %02d:%02d:%02d\n\r",
                jt1 + 1, d1, n1, h1, m1, s1);
    if (show_time == 1)
        {
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        }
    }
```

```
/*************** READ_HHMM gets time from system clock ***************/
void read_hhmm(void)
    {
    struct tm *tmnow;

    time(&tnow);
    tmnow = gmtime(&tnow);
    hll = tmnow->tm_hour;
    mll = tmnow->tm_min;
    sll = tmnow->tm_sec;
    qll = mll / 15;
    }


/********************* SEND_RS232 sends a message ************************/
void send_rs232(unsigned port, char * message_buffer,
            int mess_length, int *num_sent)
{
unsigned status;
int ch, del_ctr, part_length, tries = 0;

switch(port)
    {
    case ARGOS_PORT:
        part_length = mess_length / 4;          /* send ARGOS in 8 frames */
        break;
    case METEO_PORT:
        part_length = mess_length / 72;         /* send METEO in 72 chunks */
        break;
    case SONIC_PORT:
        part_length = mess_length;              /* send each monitor message in 1 chunk */
        break;
    default:
        part_length = mess_length;
        break;
    }

for (ch = *num_sent * part_length; ch < (*num_sent + 1) * part_length; ch++)
    {
    if (ser_putc(port, message_buffer + ch) != 1)
        {
        tries++;
        }
    for (del_ctr = 0; del_ctr < 1000; del_ctr++);
    /* added delay because of occasional errors in data transfer to DCP */
    }
sprintf(display_buffer, "%d RS232 errors-port %d\n\r", tries, port);
#if DISPLAY == TRUE
    {
    if (tries > 0)
        {
        printf(display_buffer);
        }
    }
#else
    {
    if (tries > 0)
        {
        send1(display_buffer);
        }
    }
#endif
```

```
(*num_sent)++;
}

/************ 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) */
        }
    }

/***************** METEOFILL sends data to DCP ***************/
void meteofill(void)
    {
    /* check time is suitable and send METEO_LENGTH bytes from b4[] to DCP */

    if ( (m11 == 7) && (meteo_parts_sent > 0) )
        {
        meteo_parts_sent = 0;
        return;
        }

    if ( (m11 > 7) && (m11 < 10) && (meteo_parts_sent < (METEO_LENGTH / 4)) )
        {
        sprintf(display_buffer, "MFILL-%d\n\r", meteo_parts_sent);
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        send_rs232(METEO_PORT, b4, METEO_LENGTH, &meteo_parts_sent);
        }

    }

/************ RS232OP send ARGOS and METEO to screen and MON_PORT ***********/
/************ also copies ARGOS and METEO to data_buffer for flash **********/
void rs232op(void)
    {
    char mess_hdr[10];

    int ch, mess_line, mon_messages_sent = 0;

    /* write databases to screen */
    sprintf(display_buffer, "\n\rARGOS%.02d%.02d\n\r", h11, m11);
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
```

```
#else
    {
    send1(display_buffer);
    }
#endif

for (mess_line = 0; mess_line < 4; mess_line++)
    {
    strcpy(display_buffer,"");                    /* reset string */
    for (ch = 0; ch < 32; ch++)
        {
        sprintf(mess_hdr, "%02X", b3[32 * mess_line + ch] & 0xff);
        strcat(display_buffer, mess_hdr);
        }
    strcat(display_buffer, "\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #endif
    send1(display_buffer);                        /* send whether DISPLAY or not */
    }
sprintf(display_buffer, "\n\rMETEO%.02d%.02d\n\r", h11, m11);
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

for (mess_line = 0; mess_line < METEO_LENGTH; mess_line += 32)
    {
    for (ch = 0; ch < 32; ch++)
        {
        display_buffer[ch] = b4[mess_line + ch];
        }
    display_buffer[32] = 0;
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #endif
    send1(display_buffer);                        /* send whether DISPLAY or not */
    }
}

/********** MESSAGECHECK checks MMET or SONIC message for errors **********/
void messagecheck(int cf)
    {
    char ch, f_err;
    char *ptr;
    char **g1 = &ptr;

    int n;

    cf &= 1;                                      /* 0 for Mmet, 1 for Sonic */

    sprintf(display_buffer, "MCHK-%d\n\r\"", cf);
```

```
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif
if (cf == 0)                                    /* Multimet */
    {
    *g1 = &b1[0];
    }
if (cf == 1)                                    /* Sonic */
    {
    *g1 = &b2[0];
    }
*g1 += ( l[cf] * g[cf] );

for (n = 0; n < l[cf]; n++)
    {
    display_buffer[n] = *(*g1 + n);
    }
display_buffer[l[cf]] = 0;
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

ch = *(*g1 + l[cf] - 1);

if ( (ch != 84) && (ch != 212) )
    {
    sprintf(display_buffer, " %d ", ch);
    #if DISPLAY == 1
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    f_err = 128;                                /* faulty terminator */
    }
else
    {
    f_err = 0;
    }
for (n = 1; n < l[cf] - 2; n++)
    {
    ch = *(*g1 + n);

    if ( (ch < 43) || (ch > 70) )
        {
        f_err++;                                /* parity error */
        sprintf(display_buffer, "!%d!", ch);
```

```
            #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
            #else
                {
                send1(display_buffer);
                }
            #endif
            }
        }
    if (f_err == 0)
        {
        sprintf(display_buffer, "\"\n\rOK\n\r");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        }

    else
        {
        sprintf(display_buffer, "\":Error:%d\n\r", f_err);
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        }
    if (f_err > 255)
        {
        f_err = 255;
        }
    *(*g1 + l[cf] - 1) = f_err;                    /* poke f_err into last char of message */
    }


/**************** CONVERT puts data in data array ***************/
int convert(void)
/* Returns 0 if no new message, 1 if faulty Sonic message, 2 if OK */
    {
    int hn, mn, m3, n;
    long t2, t3;

    char *ptr;
    char **g1 = &ptr;

    sprintf(display_buffer, "CON-");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
```

```
#else
    {
    send1(display_buffer);
    }
#endif
if (e_last == 0)
    {
    u[30] = 0.;
    e1 = 0;
    return 0;                          /* exit due to lack of new message */
    }

for (n = 0; n < 30; n++)
    {
    u[n] = 0.;
    }
sin_total = 0.;
cos_total = 0.;

if (e_last != 2)                       /* last message Mmet */
    {
    readclock(1);
    tnow = (long)( ((jt1 * 24L) + h1) * 60L + m1 + (s1 + 30L) / 60L);
    t_last_update = tnow;
    /* no. of minutes since new year start */
    jd2 = jt1 + 1;
    sprintf(display_buffer, "System tnow:%ld\n\r", tnow);
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    t2 = tnow - 15;                    /* NB t2 is FORMATTER clock time for start
                                          of averaging period */

    }
else                                   /* last message Sonic */
    {
    e2 = 1;
    *g1 = b2 + l[1] * g[1];

    if ( *(*g1 + l[1] -1) != 0 )       /* terminator faulty */
        {
        e1 = 0;
        e2 = 0;
        return 1;
        }
    else                               /* terminator good */
        {
        headertime(g1);
        sdat(g1);
        t2 = tnow - tson;              /* tnow from sonic headertime
                                          (t2 = sonic acq start time corrected to
                                          FORMATTER clocktime */

        jd2 = jd1;
```

```
            if ( labs(tnow - t_last_update) < 5 )
                {
                c[2] += 19;                         /* results in decrement of c[2]
                                                           when div(c[2], 20) taken */
                }
            }

        }                                           /* end of part specific to Sonic message */
    m3 = 14;
    if (f[0] != 1)
        {
        m3 = g[0];                                  /* was g[0] only */
        }
    n2 = 0;
    for (mn = 0; mn <= m3; mn++)
        {
        *g1 = b1 + l[0] * mn;
        if ( *(*g1 + l[0] - 1) == 0 )
            {
            headertime(g1);                         /* tnow is MMET Headertime */
            tnow -= tmet;                           /* corrected to FORMATTER clocktime */
            sprintf(display_buffer, "SHdrSt %ld:MMHdr %ld\n\r", t2, tnow);

            #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
            #else
                {
                send1(display_buffer);
                }
            #endif
            if ( (tnow >= t2) && (tnow < (t2 + SONIC_REC_LENGTH) ) )
            /* Mmet sample lies within nominal Sonic acq period */
                {
                sum(g1);
                }
            }
        }

sprintf(display_buffer, "N2 %d\n\r", n2);
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

if (n2 >= 1)
    {
    sprintf(display_buffer, "Chan  z[0]  z[1]  z[2]  z[3] u[n]\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
```

```
  for (hn = 0; hn < 9; hn++)
       {
       n = h[hn];
       u[n] /= n2;
       if (n < 8)
            {
            u[n] = (8192 - u[n]) / 819.2;
            }
       if ( (n > 7) && (n < 24) )
            {
            u[n] /= 50;
            }
       sprintf(display_buffer, "%2d %+6.3f %+8.4f %+10.2e %+10.2e %+8.3f\n\r",
                                 n, z[n][0], z[n][1], z[n][2], z[n][3], u[n]);
       #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
       #else
            {
            send1(display_buffer);
            }
       #endif
       u[n] = z[n][0] + z[n][1] * u[n] + z[n][2] * u[n] * u[n] + z[n][3] * u[n] * u[n] * u[n];
       v[hn] = u[n];
       }
  v[9] = comp_mean(&heading_scatter);
  u[h[9]] = v[9];
  sprintf(display_buffer, "\n\r%d %+9.3f\n\r", h[9], v[9]);
  #if DISPLAY == TRUE
       {
       printf(display_buffer);
       }
  #else
       {
       send1(display_buffer);
       }
  #endif

  for (n = 0; n < 10; n++)
       {
       sprintf(display_buffer, "%+6.1f ", v[n]);
       #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
       #else
            {
            send1(display_buffer);
            }
       #endif
       }
  sprintf(display_buffer, "\n\r");
  #if DISPLAY == TRUE
       {
       printf(display_buffer);
       }
  #else
       {
       send1(display_buffer);
       }
  #endif
  }                                        /* end of if n2 >= 1 */
```

```
    u[30] = (float) n2 - 3.;
    if (u[30] < 0.)
        {
        u[30] = 0.;
        }
    if (u[30] > 7.)
        {
        u[30] = 7.;
        }
    e1 = 0;
    return 2;
    }


/*************** UPDATE updates METEO database **************/
void update(void)
    {
    sprintf(display_buffer, "UPD:");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    c[2] = div(c[2], 20).rem;                    /* c[2] range is 0 to 19 */
    if (e_last == 0)
        {
        psd = -5.;
        sonic_mws = 0.;
        fit_a = -5.;
        vert_mean = 0.;
        }

    v10 = div((int) (100 * (6 + psd) - 0.5), 512).rem;
    if (v10 < 0)
        {
        v10 = 0;
        }

    v20 = div((int) (10 * sonic_mws + 0.5), 512).rem;
    if (v20 < 0)
        {
        v20 = 0;
        }

    v30 = div((int) (100 * (6 + fit_a) - 0.5), 512).rem;
    if (v30 < 0)
        {
        v30 = 0;
        }

    v40 = div((int) (50 * vert_mean + 256.5), 512).rem;
    if (v40 < 0)
        {
        v40 = 0;
        }
    pack();
```

```
    if ( (q11 != q10) || (q21 != q20) )
        {
        meteo();
        }
    }


/**************** HEADERTIME gets message header date/time ***************/
void headertime(char **g1)                          /* tested 10/5/93 */
    {
    char b$[15], n;
    int k, d2, m2, m4;

    sprintf(display_buffer, "HT-");

    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    for (k = 0; k < 12; k++)
        {
        n = *(*g1 + 3 + k);
        if ( ((int) n > 47) && ((int) n < 58) )
            {
            *(b$ + k) = n;
            }
        else
            {
            *(b$ + k) = 48;
            }
        }
    *(b$ + 12) = '\0';

    m2 = ival(b$,2);
    sprintf(display_buffer, "%.2d/", m2);
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    d2 = 0;
    if (m2 != 1)
        {
        for (m4 = 1; m4 < m2; m4++)
            {
            d2 += d[m4 - 1];

            if ( (m2 > 2) && (m4 == 2) && (div(ival(b$,0), 4).rem == 0) )
                {
                d2++;                               /* leapyear */
                }
            }
        }
```

```
tnow = (long) ival(b$, 4) + d2;
sprintf(display_buffer, "%.2d ",ival(b$, 4));
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

jd1 = (int) tnow;

tnow = 24L * (tnow - 1) + ival(b$, 6);
sprintf(display_buffer, "%.2d:", ival(b$,6));
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif


tnow = 60L * tnow + ival(b$, 8);
sprintf(display_buffer, "%.2d:", ival(b$, 8));
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

tnow += (long) (0.5 + ival(b$, 10) / 60.);
/* number of minutes since new year */
sprintf(display_buffer, "%.2d\n\r", ival(b$, 10));
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

if (*g1 >= b2)
    {
    h21 = ival(b$, 6);
    q21 = ival(b$, 8) / 15;
    }
```

```
    if (*g1 >= b2)                          /* if a sonic header */
        {
        for (k = 0; k < 10; k++)            /* copy date/time into julian */
            {
            *(julian + k) = *(b$ + k);
            }
        *(julian + 10) = 0;
        }
    }

/************* IVAL converts 2 characters from string to int ************/
int ival(char *b$, int p)                   /* tested 10/5/93 */
    {
    char t$[3];
    int n;

    *t$ = *(b$ + p);
    *(t$ + 1) = *(b$ + p + 1);
    *(t$ + 2) = '\0';
    n = atoi(t$);
    return n;
    }

/************** FVAL converts string chars m -> m+p to double *************/
double fval(char * b$, int m, int p)
    {
    char t$[15];

    int n;

    m--;
    for (n = 0; n < p; n++)
        {
        *(t$ + n) = *(b$ + m + n);
        }
    *(t$ + p) = '\0';
    return atof(t$);
    }

/***************** SDAT converts a good Sonic message ****************/
void sdat(char **g1)
    {
    char b$[60];

    int k;

    sprintf(display_buffer, "SDAT:-");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    if ( (*(*g1 + l[1] - 1) && 0x7f) != 0 )
        {
        sprintf(display_buffer, "Faulty\n\r");
```

```
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        psd = -9.99;
        sonic_mws = 0.;
        north_mean = 0.;
        east_mean = 0.;
        vert_mean = 0.;
        c_mean = 0.;
        heading = 0.;
        fit_a = 0.;
        fit_b = 0.;
        }

    else
        {
        sprintf(display_buffer, "OK\n\r");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        for (k = 16; k <= 67; k++)
            {
            *(b$ + k - 16) = *(*g1 + 1 + k);
            }
        psd = fval(b$, 1, 5);
        if (psd == +9.99)
            {
            psd = -9.99;
            }
        sonic_mws = fval(b$, 6, 5);
        north_mean = fval(b$, 11, 6);
        east_mean = fval(b$, 17, 6);
        vert_mean = fval(b$, 23, 6);          /* added v5-v9 for Vmean, Cmean, etc. */
        c_mean = fval(b$, 29, 6);
        heading = fval(b$, 35, 3);
        fit_a = fval(b$, 38, 5);
        if (fit_a == +9.99)
            {
            fit_a = -9.99;
            }
        fit_b = fval(b$, 43, 10);
        }
*(*g1 + l[1] - 1) = (char) 128;

}
```

```
/******************** SUM sums good MMet messages ********************/
void sum(char **g1)
    {
    char b$[7];
    char *ptr;
    char **h2 = &ptr;
    char *stopat;
    char ch;
    int hn, k;

    sprintf(display_buffer, "S-\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    b$[0] = '0';
    b$[1] = 'x';

    for (hn = 0; hn < 10; hn++)
    /* order WS1, WS2, WD1, WD2, AT1, AT2, ST1, ST2, BAT, HDG */
        {
        *h2 = *g1 + 17 + 4 * h[hn];              /* hex data starts at 18th character */
        /* printf("%u ", *h2); */
        for (k = 0; k < 4; k++)
            {
            ch = *(*h2 + k);
            if (isxdigit(ch) != 0)
                {
                b$[k + 2] = ch;
                }
            else
                {
                b$[k + 2] = (char) 48;
                }
            }
        *(b$ + 6) = '\0';
        if (hn != 9)                             /* not the compass channel */
            {
            u[h[hn]] += (float) strtoul(b$, &stopat, 16);
            sprintf(display_buffer, "%6.0f ", u[h[hn]]);
            #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
            #else
                {
                send1(display_buffer);
                }
            #endif
            }
        else
            {
            compass[n2] = (unsigned) (strtoul(b$, &stopat, 16) & 0xff);
            sprintf(display_buffer, "%.3d\n\r", compass[n2]);
            #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
```

```
            #else
                {
                send1(display_buffer);
                }
            #endif
            }
        }

    n2++;
    n2 &= 0x0f;
    }


/*************** PACK packs data into ARGOS database **************/
void pack(void)
    {
    char b31, b32;
    char *ptr;
    char **o1 = &ptr;

    div_t result;

    int p, q;
    unsigned offset;

    sprintf(display_buffer, "PACK-");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    q = ( (h11 << 2) + q11) & 0x7f;
    p = parity(q);
    result = div(c[2], 5);
    offset = (result.rem << 2) + (result.rem << 1) + (result.quot << 5);
    *(b3 + offset) = (char) ((p << 7) + q);
    p = parity(v10);
    *(b3 + offset + 1) = (char) ( (v10 >> 2) + (p << 7) );
    p = parity(v20);
    *(b3 + offset + 2) = (char) (( (v10 & 0x03) << 6) + (p << 5) + (v20 >> 4));
    p = parity(v30);
    *(b3 + offset + 3) = (char) (( (v20 & 0x0f) << 4) + (p << 3) + (v30 >> 6));
    p = parity(v40);
    *(b3 + offset + 4) = (char) (( (v30 & 0x3f) << 2) + (p << 1) + (v40 >> 8));
    *(b3 + offset + 5) = (char) (v40 & 0xff);

    *o1 = b3 + ((1 + result.quot) << 5);
    /* b3 + 32, + 64, + 96, + 128 for c[2]=0->4, 5->9, 10->14, 15->19 */
    b31 = *(*o1 - 2);
    b32 = *(*o1 - 1);
    n2bits(&b31, &b32);
    *(*o1 - 2) = b31;
    *(*o1 - 1) = b32;
    sprintf(display_buffer, "\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
```

```
    #else
        {
        send1(display_buffer);
        }
    #endif
    }


/***************** PARITY gets parity bit for 9 bit word ***************/
int parity(int valu)
    {
    int p, q;
    sprintf(display_buffer, "P");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    p = 0;                                  /* p = 0 for even parity,
                                               p = 1 for odd parity */
    for (q = 1; q < 512; q *= 2)
        {
        if ((valu & q) > 0)
            {
            p++;
            }
        }
    p &= 0x01;
    return p;
    }

/***************** N2BITS puts N2 into database B31, B32 ***************/
void n2bits(char *b31, char *b32)
    {
    int n_val, p;

    n_val = (int) u[30];

    switch(div(c[2], 5).rem)
        {
        case 0:
            *b31 &= 0x0f;
            *b31 |= (n_val << 4);
            break;
        case 1:
            *b31 &= 0x71;
            *b31 |= (n_val << 1);
            break;
        case 2:
            *b31 &= 0x7e;
            *b31 |= (n_val >> 2);
            *b32 &= 0x3f;
            *b32 |= ( (n_val & 0x03) << 6);
            break;
        case 3:
            *b31 &= 0xc7;
            *b32 |= (n_val << 3);
            break;
```

```
            case 4:
                *b32 &= 0xf8;
                *b32 |= n_val;
                break;
            }

        p = parity((int) *b32 & 0xff);
        p = parity( (p << 7) + (int) (*b31 & 0x7f) );
        *b31 |= (p << 7);
        }


/***************** METEO loads data into METEO database ****************/
void meteo(void)
    {
    char *ptr;
    char **gm = &ptr;

    div_t result;

    int q30, h30, q;

    unsigned offset;

    sprintf(display_buffer, "MET-");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
    switch(e_last)
        {
        case 2:
            q30 = q21;
            h30 = h21;
            break;
        case 1:
            q30 = q11;
            h30 = h11;
            break;
        case 0:
            q30 = 3;
            h30 = 24;
            break;
        }
    *gm = b4 + 12;
    offset = 0;
    format(gm, &offset, (double) jd2, "XXX");
    *(*gm + offset - 1) = 13;

    *gm += (5 + 61 * q30);
    q = (4 * h30 + q30) & 0xff;
    result = div(q, 10);
    **gm = (char) (48 + result.quot);
    *(*gm + 1) = (char) (48 + result.rem);
    *(*gm + 2) = ',';

    offset = 3;
    format(gm, &offset, 100 * psd, "+XXX");
```

```
        format(gm, &offset, 10 * sonic_mws, "XXX");
        format(gm, &offset, 10 * north_mean, "+XXX");
        format(gm, &offset, 10 * east_mean, "+XXX");
        format(gm, &offset, 10 * vert_mean, "+XXX");
        format(gm, &offset, 100 * fit_a, "+XXX");
        format(gm, &offset, 10 * v[4], "+XXX");
        format(gm, &offset, 10 * v[5], "+XXX");
        format(gm, &offset, 10 * v[6], "+XXX");
        format(gm, &offset, 10 * v[7], "+XXX");
        format(gm, &offset, 10 * v[0], "XXX");
        format(gm, &offset, v[2], "XXX");
        *(*gm + offset - 1) = 13;
        *(*gm + offset) = 10;

        *gm = b4 + 17 + 61 * 4;
        offset = 0;
        format(gm, &offset, 10 * v[8], "XXX");
        format(gm, &offset, v[9], "XXX");
        format(gm, &offset, heading_scatter, "XXX");
        format(gm, &offset, (double) tmet, "+XXXX");
        format(gm, &offset, (double) tson, "+XXXX");

        *(*gm + offset - 1) = 13;
        *(*gm + offset) = 10;

        sprintf(display_buffer, "\n\r");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif

        }


/*************** FORMAT formats data into METEO database **************/
void format(char **gm, unsigned *offset, double vs, char *f$)
        {
        char *result;

        double v5;

        int f1, k, llen, m, n, p, r, v6;

        sprintf(display_buffer, "FMT");
        #if DISPLAY == TRUE
            {
            printf(display_buffer);
            }
        #else
            {
            send1(display_buffer);
            }
        #endif
        f1 = 1;
        if (*f$ != '+')
            {
            f1 = 0;
            }
```

```
 else
      {
      if (fabs(vs) < 0.001)
           {
           vs = 0.;
           }
      }
 if (vs > 0.)
      {
      *(*gm + *offset) = '+';
      }
 else
      {
      *(*gm + *offset) = '-';
      }
 llen = (int) strlen(f$);
 /* find 1st occurence of DP, strchr returns a pointer to DP, if found */

 if ( (result = strchr(f$, 46)) == NULL)          /* no DP found */
      {
      p = llen + 1;
      }
 else
      {
      p = 1 + result - f$;
      }

 m = llen - p;
 if (m < 0)
      {
      m = 0;
      }
 n = p - fl - 1;
 v5 = fabs(vs);
 switch(m)
      {
      case 0:
           r = 1;
           break;
      case 1:
           r = 10;
           break;
      case 2:
           r = 100;
           break;
      default:
           r = 1;
           break;
      }
 v5 = (double) ( (int) (v5 * r + 0.5) ) / r;
 vs = v5 - floor(v5);                             /* vs is mantissa */
 if (n >= 1)
      {
      for (k = 1; k <= n; k++)
           {
           v6 = 10 * (int) (0.1 * v5);
           *(*gm + *offset + fl + n - k) = (char) (48 + (int) (v5 - v6));
           v5 *= 0.1;
           }
      }
```

```
    if (p <= llen)
        {
        *(*gm + *offset + p - 1) = '.';
        }
    if (m >= 1)
        {
        for (k = 1; k <= m; k++)
            {
            v6 = (int) (10 * vs);
            if (k == m)
                {
                v6 = (int) (10 * vs + 0.5);
                }
            *(*gm + *offset + p + k - 1) = (char) (48 + v6);
            vs = 10 * vs - v6;
            }
        }
    *(*gm + *offset + llen) = (char) 44;
    *offset += (llen + 1);
    }
```

```
/***************** 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);
n = inp(COM3_BASE + MODEM_CONTR_REG);
outp(COM3_BASE + MODEM_CONTR_REG, n & 0xf7);
n = inp(COM4_BASE + MODEM_CONTR_REG);
outp(COM4_BASE + MODEM_CONTR_REG, n & 0xf7);
n = inp(COM5_BASE + MODEM_CONTR_REG);
outp(COM5_BASE + MODEM_CONTR_REG, n & 0xf7);
n = inp(COM6_BASE + MODEM_CONTR_REG);
outp(COM6_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);
outp(COM3_BASE + INT_ENABLE_REG, 0);
outp(COM4_BASE + INT_ENABLE_REG, 0);
outp(COM5_BASE + INT_ENABLE_REG, 0);
outp(COM6_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);
    inp(COM3_BASE + n);
    inp(COM4_BASE + n);
    inp(COM5_BASE + n);
    inp(COM6_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);
_dos_setvect(INT_NO7, old_irq7_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 & IRQ7;

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

/* COM5 */
if (com_init(ARGOS_PORT, BAUD_1200, 0, CHRS_8 | STOP_1 | NOPARITY) == NULL)
    {
    sprintf(display_buffer, "Initialised COM5 Port\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    }
else
    {
    sprintf(display_buffer, "Failed to initialise COM5 Port\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    return 0;
    }
```

```
/* COM6 */
if (com_init(METEO_PORT, BAUD_300, 0, CHRS_7 | STOP_2 | EVENPARITY) == NULL)
    {
    sprintf(display_buffer, "Initialised COM6 Port\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    }
else
    {
    sprintf(display_buffer, "Failed to initialise COM6 Port\n\r");

    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    return 0;
    }

/* COM3 */
if (com_init(SONIC_PORT, BAUD_2400, 0, CHRS_8 | STOP_1 | NOPARITY) == NULL)
    {
    sprintf(display_buffer, "Initialised COM3 Port\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    }
else
    {
    sprintf(display_buffer, "Failed to initialise COM3 Port\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    return 0;
    }
```

```
 /* COM4 */
 if (com_init(MMET_PORT, BAUD_2400, 0, CHRS_8 | STOP_1 | NOPARITY) == NULL)
      {
      sprintf(display_buffer, "Initialised COM4 Port\n\r");
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #endif

      }
 else
      {
      sprintf(display_buffer, "Failed to initialise COM4 Port\n\r");

      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #endif

      return 0;
      }

/* COM1 or COM2 */
/* NB    if MON_PORT==2, set up for 1200 baud rate as xtal is 3.6864 MHz
         if MON_PORT==1, set up for 2400 baud rate as xtal is 1.8432 MHz */
if (com_init(MON_PORT, BAUD_2400, 0, CHRS_8 | STOP_1 | NOPARITY) == NULL)
      {
      sprintf(display_buffer, "Initialised COM%d Port\n\r", MON_PORT);
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #endif

      }
else
      {
      sprintf(display_buffer, "Failed to initialise COM%d Port\n\r", MON_PORT);
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #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;

sprintf(display_buffer, "Old Int Mask register Contents: %x\n\r", old_ints);

#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

new_ints = old_ints & imask;                    /* enables IRQ 3, 4 & 7  (ints 11, 12 & 15) */
outp(0x21, new_ints);
n = inp(0x21);
sprintf(display_buffer, "New Int Mask register Contents: %x\n\r", n);
#if DISPLAY == TRUE
    {
    printf(display_buffer);
    }
#else
    {
    send1(display_buffer);
    }
#endif

/* save existing int handlers */
old_irq3_handler = _dos_getvect(INT_NO3);
old_irq4_handler = _dos_getvect(INT_NO4);
old_irq7_handler = _dos_getvect(INT_NO7);


/* load new int handlers */
_disable();
_dos_setvect(INT_NO3, our_irq3_handler);
_dos_setvect(INT_NO4, our_irq4_handler);
_dos_setvect(INT_NO7, our_irq7_handler);
_enable();

/* enable interrupts for Rx (not Tx or Modem) in UARTs */
/* NB include COM5 for ARGOS XON detection */
outp(COM1_BASE + INT_ENABLE_REG, 0);
outp(COM2_BASE + INT_ENABLE_REG, 0);
outp(COM3_BASE + INT_ENABLE_REG, RX_DATA_AVAIL_EN | RX_ERR_EN);
outp(COM4_BASE + INT_ENABLE_REG, RX_DATA_AVAIL_EN | RX_ERR_EN);
outp(COM5_BASE + INT_ENABLE_REG, RX_DATA_AVAIL_EN | RX_ERR_EN);
outp(COM6_BASE + INT_ENABLE_REG, 0);
```

```
/* read UART registers to clear any interrupts pending */
for (n = 0; n < 7; n++)
    {
    inp(COM1_BASE + n);
    inp(COM2_BASE + n);
    inp(COM3_BASE + n);
    inp(COM4_BASE + n);
    inp(COM5_BASE + n);
    inp(COM6_BASE + n);
    }

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

outp(COM1_BASE + MODEM_CONTR_REG, 0);
outp(COM2_BASE + MODEM_CONTR_REG, 0);
outp(COM3_BASE + MODEM_CONTR_REG, 0x08);
outp(COM4_BASE + MODEM_CONTR_REG, 0x08);
outp(COM5_BASE + MODEM_CONTR_REG, 0x08);
outp(COM6_BASE + MODEM_CONTR_REG, 0);

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;
        case 3:
            base_address = COM3_BASE;
            break;
        case 4:
            base_address = COM4_BASE;
            break;
        case 5:
            base_address = COM5_BASE;
            break;
        case 6:
            base_address = COM6_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;
    }
```

```
/******************** SER_PUTC writes a character to port ******************/
int ser_putc(int port, char *ch)
/* returns  0 if time out, 1 if OK, -1 if port other than 3 - 6 incl. */
    {
    unsigned base_address, cts_flag, n;
    switch(port)
        {
        case 1:
            base_address = COM1_BASE;
            cts_flag = 0;
            break;
        case 2:
            base_address = COM2_BASE;
            cts_flag = 0;
            break;
        case 3:
            base_address = COM3_BASE;
            cts_flag = 0;
            break;
        case 4:
            base_address = COM4_BASE;
            cts_flag = 0;
            break;
        case 5:
            base_address = COM5_BASE;
            cts_flag = 0;
            break;
        case 6:
            base_address = COM6_BASE;
            cts_flag = CLEAR_TO_SEND;
            break;
        default:
            return -1;
            break;
        }

    n = 0;
    while ( ( ((inp(base_address + LINE_STATUS_REG) & TX_HR_EMPTY) == NULL)
            || ((inp(base_address + MODEM_STAT_REG) & CLEAR_TO_SEND) != cts_flag) )
                    && (n < 10000) )
        {
        n++;
        }
    if (n >= 10000)
        {
        return 0;
        }
    outp(base_address + TX_HOLDING_REG, *ch);
    return 1;
    }

/******************** SER_GETC gets a character from port ****************/
int ser_getc(int port)
/* returns -1 if time out or port not in range 3 - 6, character if OK */
    {
    unsigned base_address, n;
    switch(port)
        {
        case 3:
            base_address = COM3_BASE;
            break;
```

```
        case 4:
            base_address = COM4_BASE;
            break;
        case 5:
            base_address = COM5_BASE;
            break;
        case 6:
            base_address = COM6_BASE;
            break;
        default:
            return -1;
            break;
        }

    n = 0;
    while ( ((inp(base_address + LINE_STATUS_REG) & DATA_READY) == NULL)
                    && (n < 1000) )
        {
        n++;
        }
    if (n >= 1000)
        {
        return -1;
        }
    return inp(base_address + RX_BUFF_REG);
    }


/**********INTERRUPT HANDLER FOR COM3 (SONIC) INTERRUPT HANDLING**********/
void interrupt far our_irq4_handler()
    {
    int m, n = 0;
    _enable();
    m = inp(COM3_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(COM3_BASE + RX_BUFF_REG);
                break;
            case RX_ERR:
                inp(COM3_BASE + LINE_STATUS_REG);
                n = 253;
                break;
            case MODEM_STATUS:
                inp(COM3_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 < 254)
            {
```

```
            switch(a[1])
                {
                case 0:
                    if ((n == 83) || (n == 211))
                        {
                        a[1] = 1;
                        c[1] = 0;
                        g[1]++;
                        if (g[1] == 4)
                            {
                            g[1] = 0;
                            f[1] = 1;
                            }
                        *(b2 + l[1] * g[1] + c[1]) = (char) n;
                        c[1]++;
                        }
                    break;
                case 1:
                    *(b2 + l[1] * g[1] + c[1]) = (char) n;
                    c[1]++;
                    if ( (c[1] >= l[1]) || (n == 84) || (n == 212) )
                        {
                        mess_rxd |= 0x0a;       /* 2, but set bit 3 (8) to activate
                                                   messagecheck */
                        e1 = 2;
                        a[1] = 0;
                        c[1] = 0;
                        }
                    break;
                default:
                    break;
                }
            }
        }
    while ( (m = (inp(COM3_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 COM3 INTERRUPT HANDLING*******/

/****INTERRUPT HANDLER FOR COM5 (ARGOS) INTERRUPT HANDLING****/
/* returns received character in argos_send_flag */
void interrupt far our_irq7_handler()
    {
    int m, n = 0;
    _enable();

    m = inp(COM5_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(COM5_BASE + RX_BUFF_REG);
                break;
            case RX_ERR:
                inp(COM5_BASE + LINE_STATUS_REG);
                n = 254;
                break;
```

```
                case MODEM_STATUS:
                    inp(COM5_BASE + MODEM_STAT_REG);
                    n = 254;
                    break;
                case TXHR_EMPTY:
                    n = 254;
                    break;
                case INT_PENDING:
                    n = 255;
                    break;
                default:
                    n = 255;
                    break;
                }
            }
        while ( (m = (inp(COM5_BASE + INT_IDENT_REG) & 0x07)) != INT_PENDING);

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

        _chain_intr(old_irq7_handler);          /* other sources of int handled */
        }
/*********END OF INTERRUPT HANDLER FOR COM5 INTERRUPT HANDLING*******/


/************INTERRUPT HANDLER FOR COM4 (MMET) INTERRUPT HANDLING*********/
void interrupt far our_irq3_handler()
    {
    int m, n = 0;
    _enable();

    m = inp(COM4_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(COM4_BASE + RX_BUFF_REG);
                break;
            case RX_ERR:
                inp(COM4_BASE + LINE_STATUS_REG);
                n = 254;
                break;
            case MODEM_STATUS:          /* should never occur */
                inp(COM4_BASE + MODEM_STAT_REG);
                n = 254;
                break;
            case TXHR_EMPTY:            /* should never occur */
                n = 254;
                break;
            case INT_PENDING:
                n = 255;
                break;
            default:
                n = 255;
                break;
            }
        if (n < 254)
            {
            switch(a[0])
                {
```

```
                case 0:
                    if ((n == 83) || (n == 211))
                        {
                        a[0] = 1;
                        c[0] = 0;
                        g[0]++;
                        if (g[0] == 15)
                            {
                            g[0] = 0;
                            f[0] = 1;
                            }
                        *(b1 + l[0] * g[0] + c[0]) = (char) n;
                        c[0]++;
                        }
                    break;
                case 1:
                    *(b1 + l[0] * g[0] + c[0]) = (char) n;
                    c[0]++;

                    if ( (c[0] >= l[0]) || (n == 84) || (n == 212) )
                        {
                        mess_rxd |= 0x05;       /* 1, but set bit 2 (4) to activate
                                                   messagecheck */
                        e1 = 1;
                        a[0] = 0;
                        c[0] = 0;
                        }
                    break;
                default:
                    break;
                }
            }
        } while ( (m = (inp(COM4_BASE + INT_IDENT_REG) & 0x07)) != INT_PENDING);

    outp(0x20, 0x20);                       /* non-specific EOI ? in do-while */
    _chain_intr(old_irq3_handler);          /* other sources of int handled */
    }

/*************** 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) );
    sprintf(display_buffer, "Block %u Ptr %u\n\r", block, b_ptr);
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif
```

```
 if (block > 63)
      {
      sprintf(display_buffer, "Out of Storage Space\n\r");
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #endif

      /* exit(0); */
      full_flag = 1;
      return(0);
      }
 if( ( (unsigned long) b_ptr + nbytes) > 65536 )
      {
      sprintf(display_buffer, "Saving %u bytes, block %u, pointer %u\n\r",
                        (unsigned) (65535 - b_ptr), block, b_ptr);
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #endif

      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
           {
           sprintf(display_buffer, "Failed\n\r");
           #if DISPLAY == TRUE
                {
                printf(display_buffer);
                }
           #else
                {
                send1(display_buffer);
                }
           #endif

           return(0);
           }
      }
 else
      {
      sprintf(display_buffer, "Final Saving %u bytes, block %u, pointer %u\n\r",
                        (unsigned) nbytes, block, b_ptr);
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
```

```
            #else
                {
                send1(display_buffer);
                }
            #endif

            if (pcmcia_save((unsigned) nbytes - 1, block, b_ptr, s_buffer) == 0)
                {
                flash_pointr += nbytes;
                nbytes = 0;
                }
            else
                {
                sprintf(display_buffer, "Failed\n\r");
                #if DISPLAY == TRUE
                    {
                    printf(display_buffer);
                    }
                #else
                    {
                    send1(display_buffer);
                    }
                #endif

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

    sprintf(display_buffer, "OK\n\r");
    #if DISPLAY == TRUE
        {
        printf(display_buffer);
        }
    #else
        {
        send1(display_buffer);
        }
    #endif

    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)

    {
    char dir_entry[35];
    char *ptr;
    char dummy[10];

    time_t t_rec;
    struct tm *gmt;


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

    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;
 ptr = (char *) &v20;
 dir_entry[15] = *ptr++;
 dir_entry[16] = *ptr;
 strcpy(dir_entry + 17, jul_start + 3);                    /* hhmmss (ss are overwritten) */
 ptr = (char *) &v10;
 dir_entry[21] = *ptr++;
 dir_entry[22] = *ptr;
 ptr = (char *) &v30;
 dir_entry[23] = *ptr++;
 dir_entry[24] = *ptr;
 ptr = (char *) &v40;
 dir_entry[25] = *ptr++;
 dir_entry[26] = *ptr;

 dir_entry[27] = 0;
 dir_entry[28] = 0;

 dir_entry[29] = 0;
 dir_entry[30] = 0;

 dir_entry[31] = 0;

 /* strcpy(dir_entry + 21, jul_start);
 strcpy(dir_entry + 30, "dm"); */

 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(void)
 {
 int ch;

 sprintf(display_buffer, "FLASH\n\r");
 #if DISPLAY == TRUE
      {
      printf(display_buffer);
      }
```

```
#else
    {
    send1(display_buffer);
    }
#endif


/* load the buffer with the ARGOS and METEO messages */
for (ch = 0; ch < ARGOS_LENGTH; ch++)
    {
    *(data_buffer + ch) = *(b3 + ch);
    }
for (ch = 0; ch < METEO_LENGTH; ch++)
    {
    *(data_buffer + ARGOS_LENGTH + ch) = *(b4 + ch);
    }


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

start_block = (unsigned) (fl_ptr >> 16);
start_offset = (unsigned) (fl_ptr - (start_block << 16) );
reclen = (unsigned) REC_LENGTH;


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

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)
    {
    logging = 0;
    flash_full = 1;
    }

}


/*****************COMP_MEAN calculates mean compass and std devn**********/
double comp_mean(double *comp_var)
{
double mean = 0., a_sum[4] = {0., 0., 0., 0.};

int mn, c_sector, tot[4] = {0, 0, 0, 0};

unsigned long c_sum[4] = {0L, 0L, 0L, 0L};
```

```
 for (mn = 0; mn < n2; mn++)
      {
      c_sector = div(compass[mn], 90).quot;
      if (c_sector > 3)
           {
           c_sector = 0;
           }
      tot[c_sector]++;
      c_sum[c_sector] += compass[mn];
      }

 for (mn =0; mn < 4; mn++)
      {
      if (tot[mn] > 0)
           {
           a_sum[mn] = (double) c_sum[mn] / (double) tot[mn];
           }
      }
 if ( (tot[0] > 0) && (tot[3] > 0) )
      {
      a_sum[0] += 360.;
      a_sum[1] += 360.;
      }
 for (mn = 0; mn < 4; mn++)
      {
      mean += a_sum[mn] * tot[mn];
      }
 mean /= (tot[0] + tot[1] + tot[2] + tot[3]);
 mean = fmod(mean, 360.);

 a_sum[0] = 0.;
 for (mn = 0; mn < n2; mn++)
      {
      a_sum[1] = fabs((double) compass[mn] - mean);
      if (a_sum[1] > 180.)
           {
           a_sum[1] = 360. - a_sum[1];
           }
      a_sum[0] += (a_sum[1] * a_sum[1]);
      sprintf(display_buffer, "%6.0f ", a_sum[0]);
      #if DISPLAY == TRUE
           {
           printf(display_buffer);
           }
      #else
           {
           send1(display_buffer);
           }
      #endif

      }
 *comp_var = sqrt(a_sum[0] / (double) n2);

 return mean;
 }


 /*************************SEND1 send a string to COM1**********************/
 void send1(char * message)
      {
      int ch = 0;
```

```
    while (message[ch] != '\0')
        {
        ser_putc(MON_PORT, message + ch);
        ch++;
        }
    }
```

Include Files for NEWFORM1

```
/*************************COMS.C*************************
*
* definitions of interrupt masks, port addresses and
* control/output register bits for 8250/16450 type UARTs
*
* CHC 26th May 1993
*
********************************************************/

/* Masks for enabling interrupt controller
        (AND these to enable combinations) */
#define IRQ0                    0xfe
#define IRQ1                    0xfd
#define IRQ2                    0xfb
#define IRQ3                    0xf7
#define IRQ4                    0xef
#define IRQ5                    0xdf
#define IRQ6                    0xbf
#define IRQ7                    0x7f

/* interrupt numbers */
#define INT_NO3                 0x0b        /* int 11 or IRQ3 (COM 5) */
#define INT_NO4                 0x0c        /* int 12 or IRQ4 (COM 3) */
#define INT_NO7                 0x0f        /* int 15 or IRQ7 (COM 4) */

/* standard base addresses for COM port UART registers */
#define COM1_BASE               0x03f8
#define COM2_BASE               0x02f8
#define COM3_BASE               0x03e8
#define COM4_BASE               0x02e8
#define COM5_BASE               0x0280
#define COM6_BASE               0x0288
#define COM7_BASE               0x0290
#define COM8_BASE               0x0298

/* offsets for 8250/16450 type UART registers */
#define RX_BUFF_REG             0
#define TX_HOLDING_REG          0
#define INT_ENABLE_REG          1
#define INT_IDENT_REG           2
#define LINE_CONTROL_REG        3
#define MODEM_CONTR_REG         4
#define LINE_STATUS_REG         5
#define MODEM_STAT_REG          6
#define DIV_LATCH_LSREG         0
#define DIV_LATCH_MSREG         1
```

```
/* divisors for a 1.8432 MHz clock rate
(set into Divisor Latches with DLAB set) */
#define BAUD_300            0x0180
#define BAUD_600            0x00c0
#define BAUD_1200           0x0060
#define BAUD_2400           0x0030
#define BAUD_4800           0x0018
#define BAUD_9600           0x000c

/* Line Control Register settings */
#define CHRS_7              0x02
#define CHRS_8              0x03

#define STOP_1              0
#define STOP_2              0x04

#define NOPARITY            0
#define EVENPARITY          0x18
#define ODDPARITY           0x08

#define DLAB                0x80

/* Line Status Register mask bits */
#define DATA_READY          0x01
#define TX_HR_EMPTY         0x20
#define TX_EMPTY            0x40
#define RX_ERROR            0x0e

/* Modem Status Register mask bits */
#define CLEAR_TO_SEND       0x10

/* Interrupt Enable Register settings */
#define RX_DATA_AVAIL_EN    0x01
#define TXHR_EMPTY_EN       0x02
#define RX_ERR_EN           0x04
#define MODEM_STATUS_EN     0x08

/* Interrupt Ident. Register mask bits */
#define INT_PENDING         0x01
#define RX_ERR              0x06
#define RX_DATA_AVAIL       0x04
#define TXHR_EMPTY          0x02
#define MODEM_STATUS        0



/**************************FORM_VAR.C*****************************
*
* array declarations/initialisations for NEWFORM.C
* (new formatter program
*
* Author CHC  20th September 1993
*
*****************************************************************/

/* byte arrays:-
b1 for Mmet data (15 messages of 138 characters + a bit to spare)
b2 for Sonic data (4 messages of 70 characters + a bit to spare)
b3 for ARGOS message (128 bytes at present + 128 to spare)
b4 for Meteosat message (288 bytes at present)
*/
char b1[2304], b2[512];
char b3[256];
```

```
char b4[288] = "B01\r\n51005\r\n]]]\r\n\
Q1,+PPP,WWW,+NNN,+EEE,+VVV,+AAA,+AT1,+AT2,+ST1,+ST2,YW1,YD1\r\n\
Q2,+PPP,WWW,+NNN,+EEE,+VVV,+AAA,+AT1,+AT2,+ST1,+ST2,YW1,YD1\r\n\
Q3,+PPP,WWW,+NNN,+EEE,+VVV,+AAA,+AT1,+AT2,+ST1,+ST2,YW1,YD1\r\n\
Q4,+PPP,WWW,+NNN,+EEE,+VVV,+AAA,+AT1,+AT2,+ST1,+ST2,YW1,YD1\r\n\
BBB,HHH,SCA,+TMTM,+TSTS\r\n\r\n";
/* 1 line of 17 chars + 4 lines of 61 chars + 1 line of 27 chars
including the \r\ns - total 288 chars = 4*72 or 16*18 */

/* integer arrays, mainly flags and pointers, e.g.
a[2] are start of message detected flags for Mmet[0] and Sonic[1]
c[4] are pointers for messages in byte arrays b1[] - b4[], respectively
d[12] are days of months
e[2] are flags for Mmet[0] and Sonic[1] message acquired
f[2] are flags for complete set of Mmet[0] and Sonic[1] messages acquired
g[2] are message numbers, 0 -> 14 for Mmet, 0 -> 3 for Sonic
h[10] is set of Mmet channels to be processed
l[2] are lengths of Mmet[0] and Sonic[1] messages, in bytes

*/

int a[2] = {0, 0};
int c[4] = {0, 0, 0, 0};
int d[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int e[2] = {0, 0};
int f[2] = {0, 0};
int g[2] = {-1, -1};
int h[10] = {11, 10, 0, 1, 15, 14, 13, 12, 2, 28};      /* numb from 0-> */
/* WS1, WS2, WD1, WD2, AT1, AT2, ST1, ST2, BAT, HDG */
int l[2] = {138, 70};                                   /* was 40 before Vmean, Cmean, etc., added */
/* double arrays:-
u[31] for Mmet data channels converted to engineering units
(u[30] used to hold number of messages summed, less 3)
v[10] for selected Mmet data
z[30][4] for Mmet 3rd order calibration coefficients
*/

double u[31] = {0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.};
double v[10] = {0., 0., 0., 0., 0., 0., 0., 0., 0., 0.};

double z[30][4]=
    {
        {0.0, 72.0, 0.0, 0.0},
        {0.0, 72.0, 0.0, 0.0},
        {0.0, 4.1, 0.0, 0.0},
        {0.0, 5.2, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {.1722464, .101078, 0.0, 0.0},                  /* YG8930 WS2 */
        {.1621761, .1014255, 0.0, 0.0},                 /* YG6991 WS1 */
        {-1.077974, 7.742313e-2, 7.021846e-6, 3.138185e-10},        /* SST2 io0011 */
        {-1.069409, 7.699735e-2, 7.625109e-6, 1.082553e-10},        /* SST1 io0010 */
        {-1.181543, 7.740265e-2, 5.518508e-6, 2.807694e-9},         /* AT2 io0002 */
        {-.9671096, 7.760471e-2, 4.618097e-6, 4.173004e-9},         /* AT1 io0001 */
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
```

```
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0},
        {0.0, 1.0, 0.0, 0.0},
        {0.0, 0.0, 0.0, 0.0}
    };
```

Assembly Code

```
; Name   FLASH5.ASM
;
; Function: FOR GCAT - drivers for PCMCIA Flash EEPROM Card
; NB test vn of flash4.asm for LFEAT macro
;
; assemble using masm /MX flash4;
;
; developed from DSP code, but uses LFEAT AX instructions which
; are not recognised by MASM.
; Macro 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          ; 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          ; 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     x
                    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 E General Assembly and Parts List

The assembly of the combined Sonic Processor/Formatter unit is shown in Figure E.1.

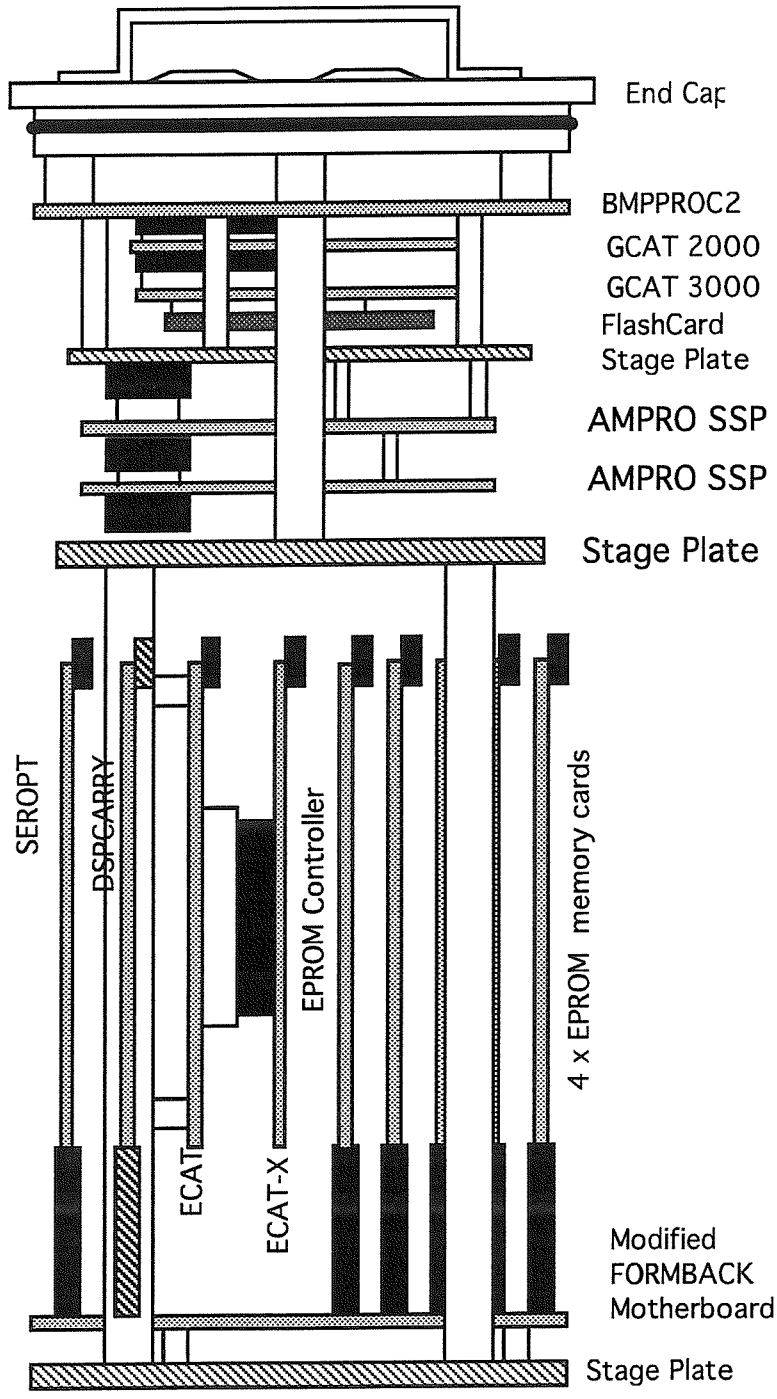


Figure E.1 Schematic of Formatter/Sonic Processor

Parts List

1 off C.5597-131 Formatter Tube Lid assembled with LEMO connectors and internal interconnecting IDC cables as per wiring specification, section 5.2

1 off C.5597-19 Formatter Tube (as for Battery Housing)

3 off C.5597-135 Formatter Spacer-1 for mounting BMPPROC2 off Lid

3 off C.5597-137 Formatter Spacer-3 for mounting stage plate

1 off C.5597-143 Formatter Disc-1 (stage plate)

3 off 6 mm Nuts and Locking Washers for stage plate

1 off BMPPROC2 Board (assembled with components as per Figure E.2

1 off GCAT 3000 unit

1 off GCAT 2000 unit

4 off C.5597-136 Formatter Spacer-2 for mounting I/O board stage plate

1 off I/O board Mounting Plate to sketch "dsp serial chassis"

Assorted Fasteners for I/O board stage plate

2 off AMPRO Minimodule^{TM} /SSP

1 off 64 way Bus Connecting Cable (IDC) with DIN41612 connectors

Assorted Spacers and Fasteners for mounting AMPRO boards (from AMPRO kit)

4 off C.5597-138 Formatter Spacer-4 for mounting Sonic Processor stage plate

1 off C.5597-144 Formatter Disc-2 (sonic processor stage plate)

Assorted Fasteners for above

4 off C.5597-139 Formatter Spacer-5 for mounting Sonic Processor backplane off stage plate

1 off Sonic Processor Backplane (modified FORMBACK), fitted with

7 off Edge Connectors/Card Guides

1 off Watchdog board

1 off SEROPT board

1 off DSPCARRY board

1 off ECAT board

1 off ECAT-X board

1 off EPROM Controller board

4 off EPROM memory boards fitted with 2 Mbit EPROMs

Appendix F BMPPROC2 - Motherboard for GCAT and AMPRO Minimodules™

Parts List

---

1 lot of parts from BMPPROC2.KIT

ALPHABETICALLY ORDERED LIST OF PARTS WITH SILK REFERENCES AND DESCRIPTIONS

---

1 off PCB	BMPPROC2	Motherboard manufactured to IOSDL artwork BMPPROC2.ART
1 off 1N4148	D4	Small Signal Diode Farnell 1N4148
2 off 1N5400	D1, D2	Power Diode Farnell MBR1060
1 off CD4025	IC7	Triple 3 i/p NOR gate Farnell CD4025BCN
1 off CD4060	IC8	Oscillator/Divider Farnell CD4060BCN
2 off CFKC2#220P	C21, C22	Capacitor Polycarbonate Farnell 147-661
2 off CMKS2#0U1	C24, C25	Capacitor Polycarbonate Farnell 143-680
4 off CMKS2#0U47	C7, C23	Capacitor Polycarbonate Farnell 143-684
2 off CTANT#15U	C1, C2,	Capacitor Tantalum 35V Farnell 100-907
1 off CTANT#22U	C8	Capacitor Tantalum 25V Farnell 100-892
1 off D9SKT-RT	H4	90° PCB-mounting 9 way D Socket Farnell 150-738
1 off DC24-5S	CONV1	24V i/p to 5V @ 2A o/p DC-DC Convertor

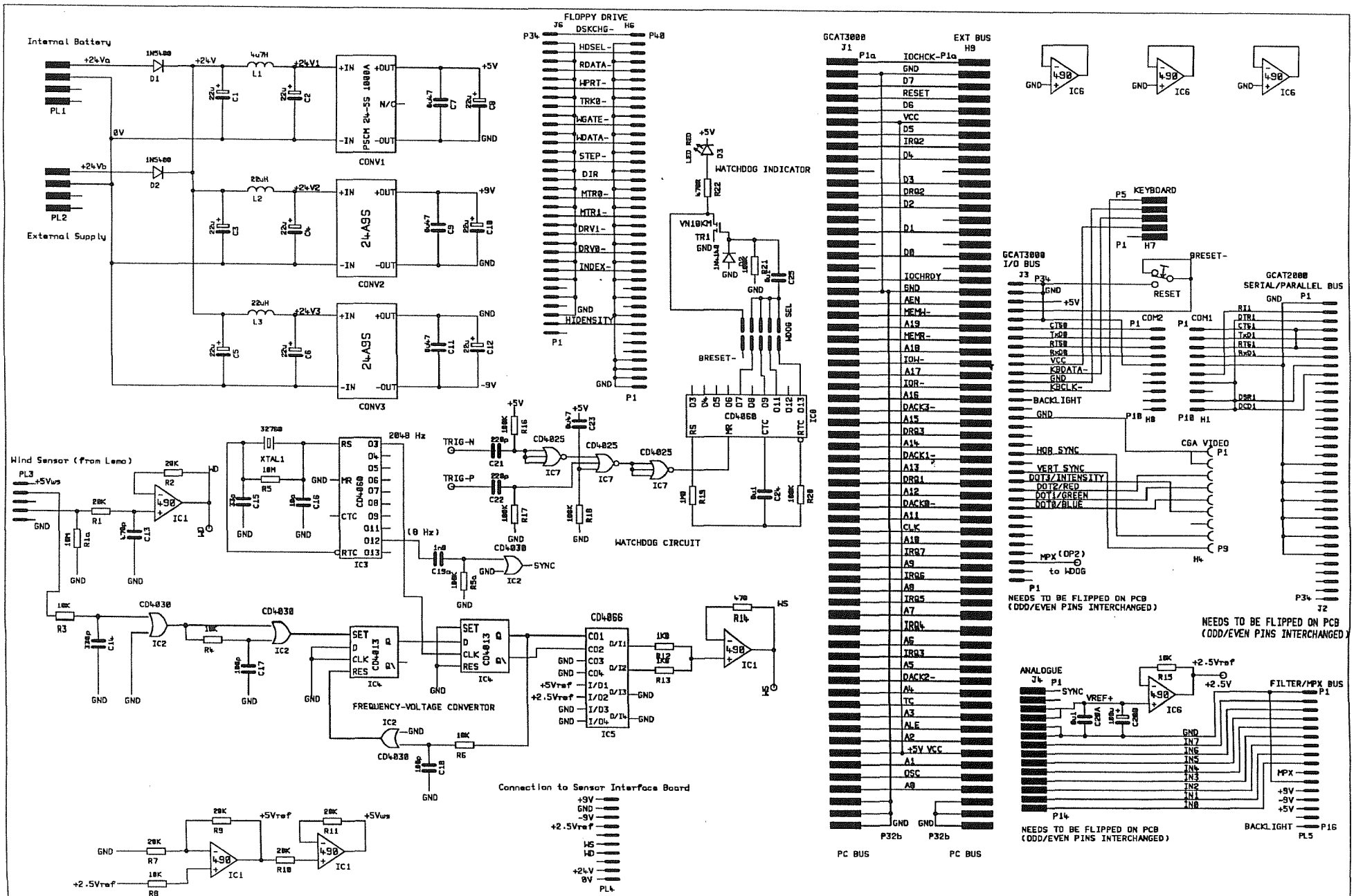
		KRP Power Source UK LPD 10/33 - 5S2000A
1 off DIN5PIN-RT	H7	PCB-mounting DIN 5 way Socket Farnell 148-505
1 off GCAT64CON	GCAT64CON/2	Pin Header Straight Double Row 64 way part of Farnell 148-195
related parts *		
* 2 off		64 way 'B' Body IDC DIN41612 Socket Farnell 224-327
* ~ 200 mm		64 way IDC cable Farnell 148-301 1 foot
1 off GCAT64CON	J1	Socket Double Row 64 way 0.1" M20-9833206
1 off IDC10	H1	Male PCB mounting IDC header Farnell 145-057
related parts *		
* 1 off	SKH1	Female 10 way Socket Bump/Clip Pol'n Farnell .152-718
* ~200mm		10 way IDC cable (SK1 to LEMO Monitor) Farnell 171-10 1 foot
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 IDC40	H6	Socket Double Row 40 way 0.1" part of an M20-9833206
2 off IRF#22UH	L2, L3	R.F.Choke Farnell 177-510
1 off IRF#4U7H	L1	R.F.Choke Farnell 177-508
1 off LED-0.2-RED	D3	Red LED Farnell 213-664
1 off PATCH5	WDOGSEL	Pin Header Straight Double Row 10 way part of Farnell 148-195
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
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

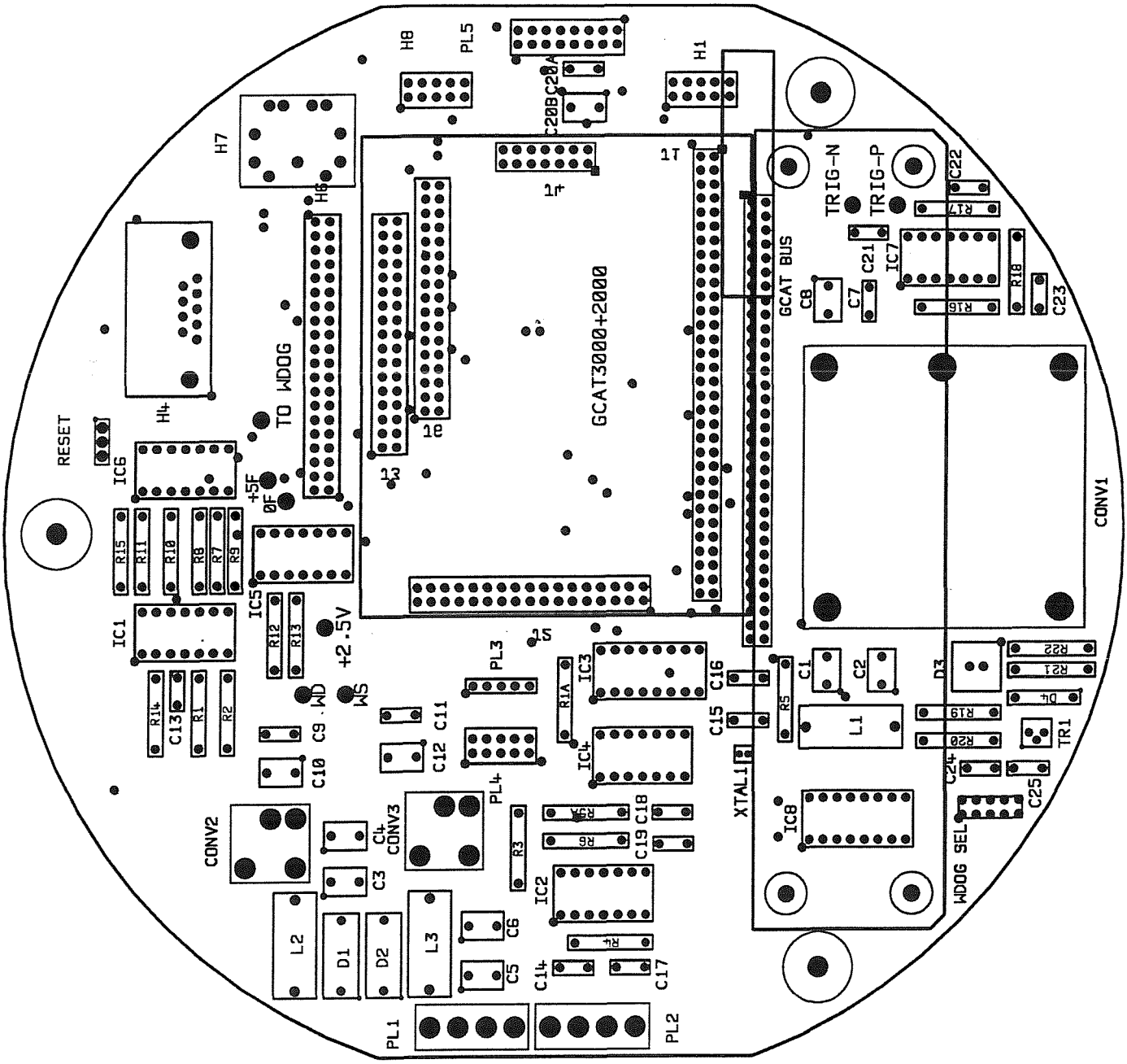
Additional Parts Required for AMPRO Minimodules™

4 off	Female 10 way Socket Bump/Clip Pol'n
	Farnell .152-718
* ~800mm	10 way IDC cable (SK1 to LEMO Monitor)
	Farnell 148-288 3 foot length
3 off	Transipillars 25mm x M3
	Farnell 147-958

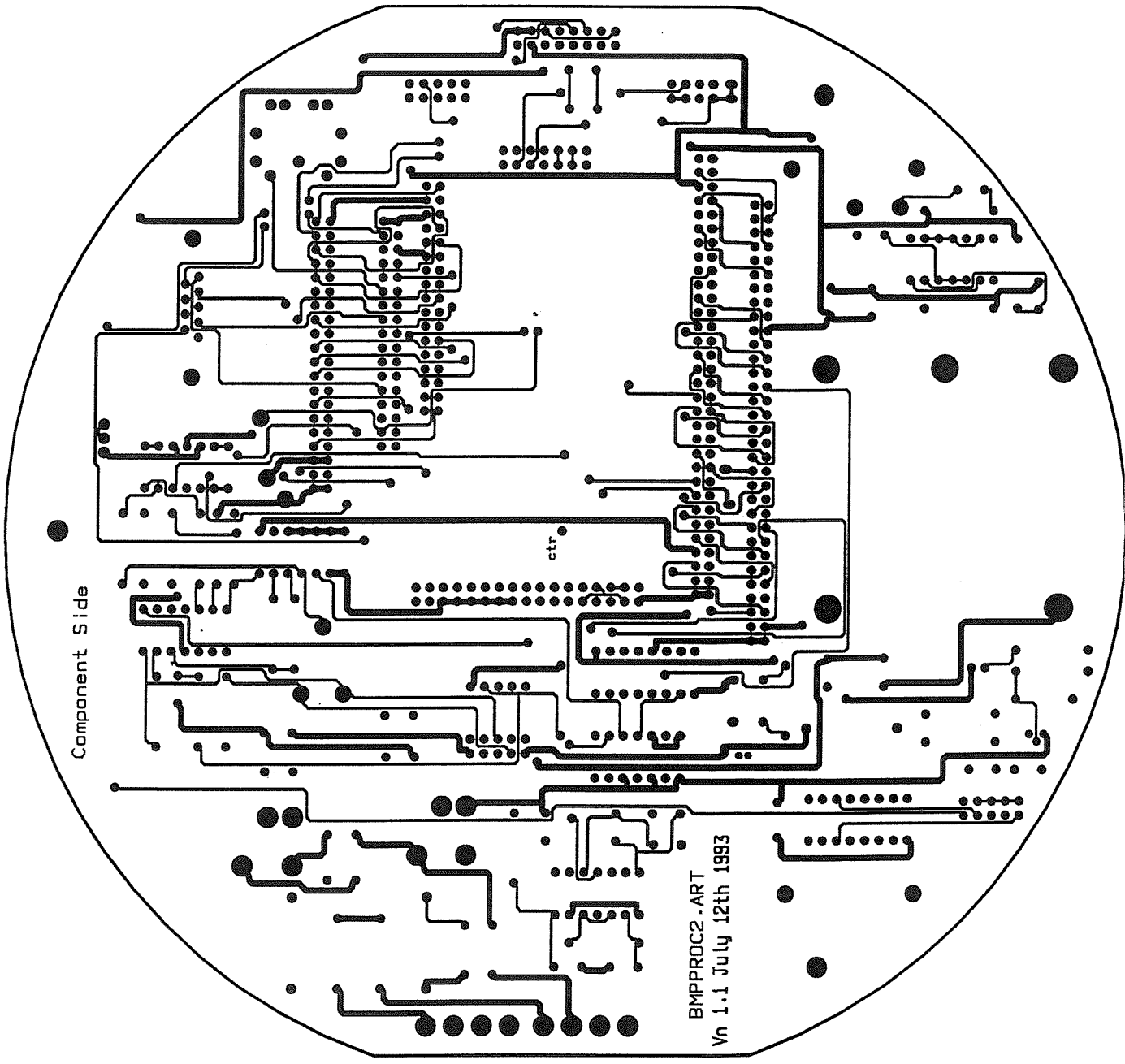


INSTITUTE OF OCEANOGRAPHIC SCIENCES, DEACON LABORATORY, BROOK ROAD, WORMLEY, GODALMING, SURREY GU8 5UB, ENGLAND	FILENAME	Sonic Buoy Motion Package Processor Board	SHEET 1 OF 1
	BMPPROC2.DGM		ISSUE 2
			DATE 02/06/1993

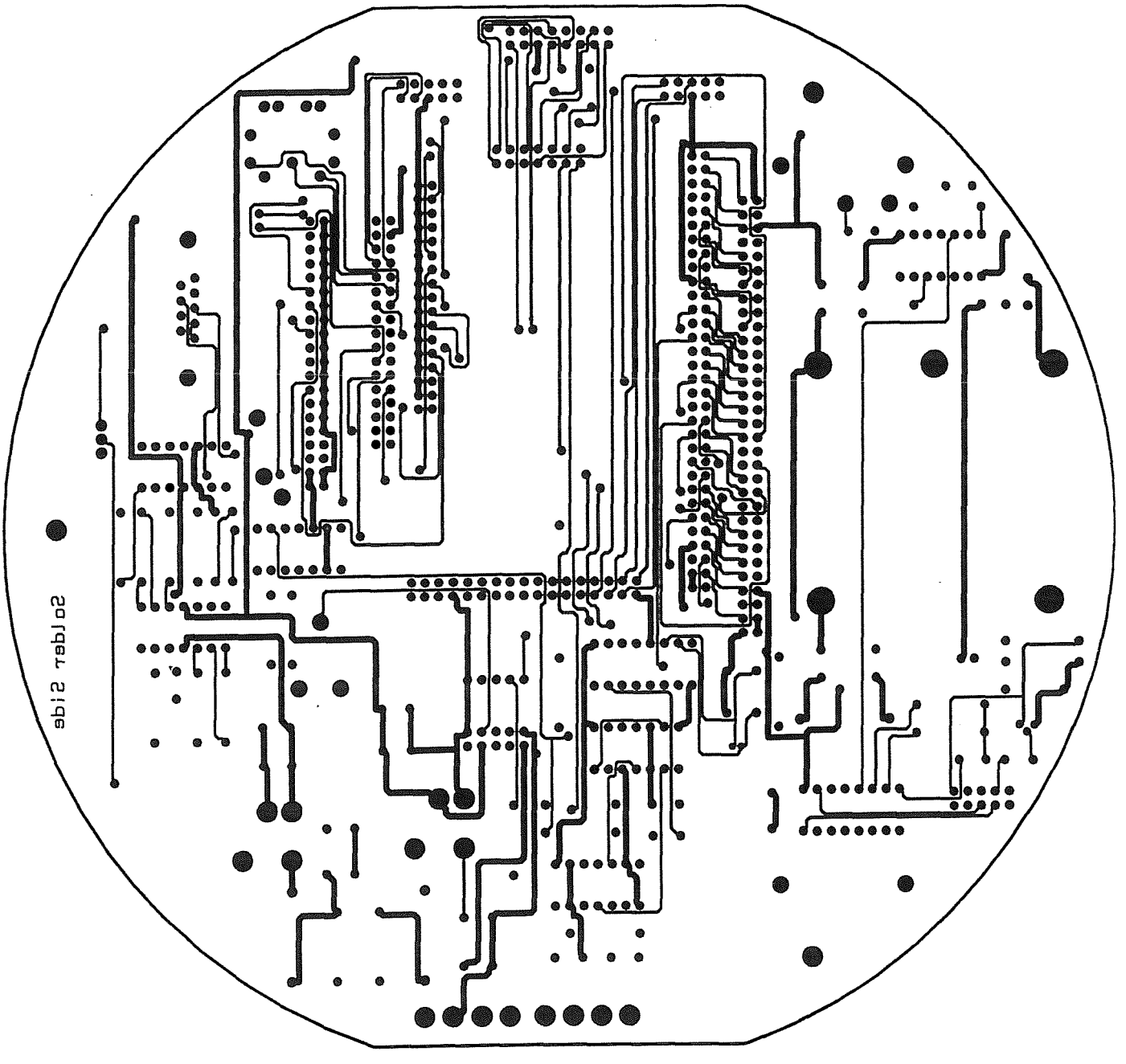
silk screen


GCAT3000+2000
TO WDOG
RESET
+5F
0F
+2.5V
TRIG-N
TRIG-P
GCAT BUS
WDOG SEL
XTAL1
CONV1
CONV2
CONV3
H1
H4
H6
H7
H8
PL1
PL2
PL3
PL4
PL5
IC1
IC2
IC3
IC4
IC5
IC6
IC7
IC8
L1
L2
L3
D1
D2
D3
TR1

compt side tracking



solder side tracking



Brook Road, Wormley, Godalming
Surrey, GU8 5UB,
United Kingdom
Telephone +44 (0) 428-684141
Facsimile +44 (0) 428-683066
Telex 858833 OCEANS G

Natural
Environment
Research
Council