FILE

Institute of
Oceanographic Sciences
Deacon Laboratory

INTERNAL DOCUMENT No. 340

Sonic Buoy - Sonic Processor handbook

C H Clayson & R W Pascal

1994

Natural Environment Research Council

INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

INTERNAL DOCUMENT No. 340

Sonic Buoy - Sonic Processor handbook

C H Clayson & R W Pascal

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 & PASCAL, R W	1994

TITLE

Sonic Buoy - Sonar Processor handbook.

REFERENCE

Institute of Oceanographic Sciences Deacon Laboratory, Internal Document, No. 340, 69pp.
(Unpublished manuscript)

ABSTRACT

The Sonic Processor was developed as part of the Sonic Buoy development program; it was based on a similar shipborne system developed for analysis of the wind turbulence spectrum to give the wind stress, using the dissipation technique.

Results from the shipborne system suffered from flow disturbance at the ultrasonic anemometer due to the ship's structure, whereas the Sonic Buoy was designed for optimum exposure of the sensor, resulting in lower scatter of the experimental results.

The Sonic Processor acquires 10 minute records of wind speed data at approximately 21 Hz and spectrally analyses these in near real time. The processed spectra and parameterised data are saved on an EPROM logger; the parameterised data are also sent to the Formatter Processsor for monitoring via satellite telemetry.

This document describes in detail the design and operation of the Sonic Processor and the associated EPROM logger; 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 7

2. FUNCTIONAL DESCRIPTION 7

3. SOFTWARE 8

3.1 Overview 8

3.2 SETTIME - Application for Clock Synchronisation 9

3.3 FFTC2 - Application for Control of Sonic Data Collection/Processing/Logging 10

3.4 RTCN.EXE - Application for updating System Clock from Real Time Clock (RTC) 13

4. HARDWARE 13

4.1 General 13

4.2 Circuit Descriptions 14
4.2.1 DSPCARRY 14
4.2.2 SEROPT Rev. 2 14
4.2.3 EPROM Logger Controller and Memory boards 15

5. WIRING 15

5.1 Lid Connectors 15

5.2 Wiring from Lid to PCBs 15

5.3 Interboard Wiring 17

6. OPERATIONAL 21

6.1 Procedure for Setting the Time/Date 21

6.2 Procedure for Initialisation of the EPROM Logger 23
6.2.1 Getting Started 23
6.2.2 File Initialisation 24

6.3 Procedure for EPROM Logger Replay 24
6.3.1 Requirements 24
6.3.2 To Translate data to the PC 24

7. SPECIFICATION 26

7.1 Supplies 26

7.2 Power Consumption 26

7.3 Data Storage and Output 26

7.4 Sampling 26

7.5 Specification of Sensors 27
7.5.1 Anemometer 27
7.5.2 Compass 27

8. ACKNOWLEDGEMENTS 27

9. REFERENCES 27

APPENDICES 28

Appendix A The EPROM Logger Data Storage Format 28

Appendix B The Sonic Processor Serial Output Message 29

Appendix C Source Code for SETTIME Application 30

Appendix D.1 Source Code for FFTC2 Application 33

Appendix D.2 Source Code RTCN.C 54

Appendix E Hardware 57
General Assembly 57
Parts List 58

Appendix F DSPCARRY - Carrier Board for ECAT/ECAT-X 59
Parts List 59
Circuit Diagram 60
Printed Circuit Board 61

Appendix G SEROPT - Anemometer RS232/422 interface and Opto-isolators 64
Parts List 64
Circuit Diagram 66
Printed Circuit Board 67

1. INTRODUCTION

The Sonic Processor is designed to communicate with a Gill Ultrasonic anemometer, to spectrally process 12,288 samples of anemometer data at quarter-hour intervals, and to output a parameters message to the Formatter. The processor also outputs the spectrum and parameters to the EPROM logger at quarter-hour intervals. The Sonic Processor is a complete PC-based processing system, using DSP Designs Ltd. ECAT^{TM} and ECAT-X^{TM} boards, mounted on a motherboard, DSPCARRY, plugging into an IOSDL 1802 microboard backplane. Also plugged into this backplane are an interface board, SEROPT, and an IOSDL EPROM logger, comprising a processor board and four memory boards. The system is mounted within a tube which also contains the Formatter system.

2. FUNCTIONAL DESCRIPTION

The main functions of the Sonic Processor are as follows:

a) to control the operation of the anemometer and to receive data from it via an RS 422 link; this is done using the ECAT COM1 port with an RS232 to RS422 convertor on the SEROPT board. A modified version of the Gill-supplied software application FASTCOM is used, with data being stored on RAMDisk.

b) to spectrally process 12 sections of 1024 samples of the resultant wind speed. For each section, the following processes are carried out.

the mean value of the resultant wind speed is first calculated and subtracted from the samples.

a partial cosine data window is applied

a 512 point FFT function is used, with the 1024 samples entered as the real and imaginary input values; the output is converted into a 256 line power spectrum which is then corrected for windowing loss.

the spectrum is then multiplied by frequency$^{5/3}$ to give an ideally flat spectrum over the equilibrium region and the mean log(power spectral density * frequency$^{5/3}$) is calculated over the range 2 - 4 Hz; a least squares fit to the (log) spectrum is also computed over this range.

mean values of the resultant wind speed, North, East and Vertical components of the flow and velocity of sound are also computed over the 12288 samples

c) the spectrum is written to the EPROM logger, using the LPT1 parallel printer port with handshake; the processed parameters are also written as detailed in the EPROM data format description (Appendix A)

d) a standard format message (see Appendix B) is sent to the Formatter via the COM2 serial port on the ECAT-X board

The above functions are achieved by the application FFTC2.EXE which is held in ROMDisk drive A. The ROMDisk also holds DOS version 5.0, AUTOEXEC.BAT and CONFIG.SYS files,

the drivers EMS230.SYS, RAMDRIVE.SYS and the applications FASTCOM.EXE, RTCN.EXE, SETCLOCK.EXE and SETTIME.EXE. The latter two applications are run before FFTC2 to allow setting of the hardware Real Time Clock via the COM2 port; this is described in more detail in the Software Section, below.

The EPROM logger consists of a slightly modified IOSDL microboard EPROM controller with four EPROM memory boards. The memory boards are each fitted with 16 x 2 Mbit EPROMs, giving a total capacity of 16 Mbytes. The use of the 2 Mbit EPROMs (instead of the usual 1 Mbit type) required that the most significant Board Select bit is used as the EPROM most significant address bit. Thus the controller fills the lower half of each EPROM in boards 0 to 3 first and then fills the upper halves of the EPROMs by addressing boards 4 to 7. For full details of the EPROM logger, see ref. 1.

3. SOFTWARE

3.1 Overview

The Sonic Processor software is embedded in the ECAT system in two x 256 kbyte EPROMs (IC1 and 8) on the ECAT-X board. These are configured as a ROMDisk drive by use of an appropriate BIOS in EPROM (IC12), combined with a ROMDisk driver ECROM.BIN in EPROM (IC11) on the ECAT board. The ROMDisk (drive A:) EPROMs contain:

DOS version 5.0 - "hidden files" and COMMAND.COM

AUTOEXEC.BAT

CONFIG.SYS

EMS230.SYS

RAMDRIVE.SYS

(the C: drive is RAMDrive, note that this driver must match the DOS version used)

SETCLOCK.EXE

SETTIME.EXE

FFTC2.EXE

FASTCOM.EXE

RTCN.EXE

The process for preparing the EPROMs is described in ref. 2.

The DSP-supplied application SETCLOCK.EXE is run when the system has booted up; this enables setting of the Real Time Clock. The application SETTIME.EXE is then run; this allows synchronisation of the Sonic Processor clock with the clock of an external PC, running the BASIC program SONTIM.BAS and with its COM1 port connected to the Sonic Processor COM2 port. This external PC is normally a battery-powered Husky Hunter 16 (running GWBASIC under DOS).

After the completion of the application SETTIME, the application FFTC2.EXE is run; this is the main data acquisition control program with the functions described above. It "spawns" two

other applications, FASTCOM.EXE and RTCN.EXE; the former is used to control the anemometer and to acquire data from it; the latter is used to update the software clock from the Real Time Clock just before midnight every day. The application FFTC2 remains running continuously until terminated by a manual reset, or by a system failure. A system failure, such as a processor crash or a failure to communicate with the anemometer, will result in the watch dog timer rebooting the system.

3.2 SETTIME - Application for Clock Synchronisation

The application is built from the object file SETTIME.OBJ; this is produced by compiling the 'C' code SETTIME.C. The library SLIBCE.LIB is used when linking. A listing of the source code is given in Appendix C.

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

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

MM is Month (01 - 12)

DD is Day of the Month (01 - 31)

HH is Hour (00 - 23)

mm is Minute (00 - 59)

SS is Second (00 - 59)

and Q is a terminator. The date and time are derived from the system clock.

The application then outputs the Date/Time message via the COM2 port. (on the ECAT-X board); 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 a set interval, the application times out. Otherwise, the external PC's Date/Time message is decoded and used to set the ECAT's Real Time and system clocks, using DOS DATE and TIME calls. The application then outputs a message in the above format (using the received Date/Time) to the external PC via the COM2 port.

Note that the SETTIME application is only effective if the ECAT has been enabled by running SETCLOCK on power up; the latter is supplied by DSP Design Ltd.

This version (1) of the application SETTIME is specific to the ECAT system, although a similar application (but using the COM1 port) has been produced for the DSP GCAT system.

3.3 FFTC2 - Application for Control of Sonic Data Collection/Processing/Logging

The application is built from the object file FFTC4.OBJ, which is produced by compiling the 'C' source code FFTC4.C The library MLIBC7.LIB is used when linking. The QuickC command line for carrying out the above processes is:

qcl /AM /Zr /FPi87 fftc4.c /F 9000 mlibc7.lib

The FFTC4.EXE file is then renamed FFTC2.EXE. A listing of the source code is given in Appendix D.

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

a hardware error handler is set up

the time zone is set to GMT

parameters for defining the FFT process are set up and calculated

housekeeping data for the logged data headers are set up

a continuous loop is then entered, in this loop, which is shown in schematic form in figure 1, below:

the watch dog circuit is triggered

a wait state is entered until it is time for a new "record", i.e. 0, 15, 30 or 45 minutes past the hour; this is effected by the function wait_start

when this occurs,

a string "julian" having the format <jjjhhmm> is created and the sample number "sample" and quarter hour "qtr" variables are updated.

the mean wind speed variable and power spectrum array are reset to zero

the application FASTCOM is then spawned, using the command line:

fastcom testfile 1 1 12288 1

i.e. <application name> <RAMDisk filename for raw data> <mode> <baud rate> <number of samples to be collected> <number of analogue channels>

with RAMDisk filename set to testfile

mode set to 1 for Calibrated UVW and C output with 20.83 Hz sampling rate

baud rate set to 1 for 4800 baud RS422 serial communications

number of samples to be collected set to 12288 (12 sections of 1024)

number of analogue channels set to 1 (for analogue compass input)

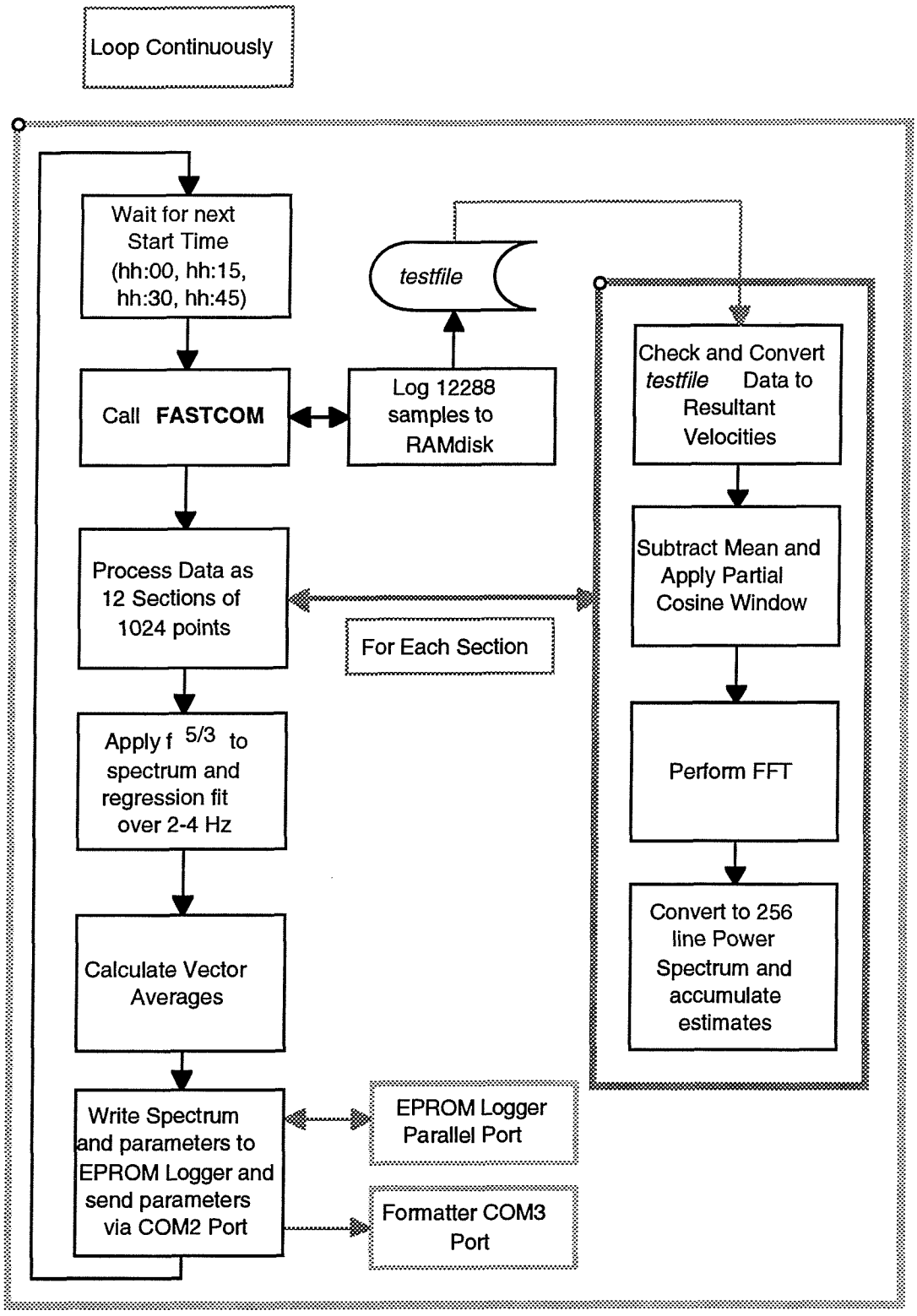


Figure 1 Main Loop in FFTC2 for Data Acquisition/Processing

The application FASTCOM then establishes communications with the anemometer, sets it to the required mode, baud rate and number of analogue channels. The anemometer will initially be in the unprompted mode, whereby it sends blocks of 20 samples of data (10 bytes per sample, 2 bytes each for U, V, W, C and Compass) with a 2 byte record number header, i.e. 202 bytes

total, at intervals of rather less than 1 second. The anemometer is then set into the "prompted" mode, whereby it will send the contents of its data buffer on receipt of a transmit command; the transmit command is sent by FASTCOM at intervals of (nominally) 1 second and, in practice, blocks of length 202, 212 or 222 bytes result, with the average length being 210.3 (10*20.83 + 2). Note that, at changeover from unprompted to prompted mode and vice versa, shorter blocks may result.

FASTCOM repeatedly requests data at 1 second intervals, checks that the record numbers are consecutive, and writes the data to the RAMDisk file until the correct number of samples has been acquired. It then resets the anemometer to unprompted mode.

For further information on FASTCOM, see the Gill handbook, ref. 3.

When the data have been acquired, the other "accumulator" variables are reset to zero and the data are analysed, as 12 sections of 1024 samples.

Each section is first checked and, if error free, converted to a resultant wind speed array a[ ] by the function *getdat*. If an error is encountered, the data are not used and the next section is checked. The first 512 values are entered into the odd members of the array, i.e. a[1], a[3] . . a[1023], the next 512 values are entered into the even members of the array, i.e. a[0], a[2] . . a[1022]. This function also accumulates sums of the East, North and Vertical wind vectors, sin(buoy heading) and cos(buoy heading), derived using the analogue channel compass reading.

The mean value of the a[ ] values for the section is then subtracted from each a[ ] value by the function *dcfilter* and the mean is added to the mean wind speed accumulator variable.

The data a[ ] are then windowed by a partial cosine window function, using the function *window*.

The data are then converted to a spectrum, using the FFT function *four1*. This transforms 512 complex input points to 512 complex output values. The even members of a[ ] are the real components of the input values and the odd members are the imaginary components. After the transform has been executed, the complex output values are "unscrambled" into 256 power estimates which are placed into the array members a[1] to a[256]. These are then added to the p[ ] power accumulator array values.

After this process has been repeated for all good sections of data, the power estimates are corrected for window loss and normalised by dividing by the number of good sections used. Likewise, the mean wind speed and vector accumulator variables are divided by the number of good sections used. The mean buoy heading is calculated from the sin(buoy heading) and cos(buoy heading) accumulator values.

The power estimates are then converted to power spectral densities, which are multiplied by frequency$^{5/3}$, converted to $\log_{10}$ form and placed in the array p[ ]. The 255 values p[2] to p[256] are referred to as the PSD (power

spectral density) values, although they are strictly the values of $\log_{10}$(power spectral density times frequency$^{5/3}$).

These values, together with housekeeping information and computed parameters are then written to the string *eprom*, as described in Appendix A. The parameters include the coefficients of a least squares fit to the PSDs versus $\log_{10}$(frequency) between 2 and 4 Hz, these coefficients are returned by the function *regres*.

The final length of the string *eprom* is 1920 bytes and these bytes are written to the EPROM logger via the parallel LPT1 port as 15 blocks of 128 bytes, using the logger LAV handshake line.

Finally, the parameters are sent to the Formatter via the COM2 port in the message format described in Appendix B.

This concludes the operations within a single pass of the loop; control then returns to the wait_start function which waits for the next record start time.

It is not possible to exit from the application, other than by a hardware reset.

3.4 RTCN.EXE - Application for updating System Clock from Real Time Clock (RTC)

This application is spawned by the main control program FFTC2.EXE during the waiting state just before midnight (between the processing completion time and 23:58:59 hrs). The update is inhibited if the system clock time is 23:59:ss, to prevent any inconsistent date and time values from resulting. The call is as follows:

```
a:\RTCN.EXE 2
```

This calls RTCN in the ROMDISK (a: drive) with an argument of 2, so that the ECAT's system clock is updated using the RTC's time/date.

The application source code is given in Appendix D.2. It uses BIOS calls (using software interrupt 0x1a) to get the RTC time and date; the return values are decoded from BCD into decimal and then used in DOS calls (using software interrupt 0x21) to set the system clock time and date.

4. HARDWARE

4.1 General

The Sonic Processor unit is mounted off the Formatter assembly in the combined Formatter/Sonic Processor housing, using a stage plate and four long aluminium pillars. The backplane (mother) board is mounted on the stage plate using four short pillars. The small watchdog timer board is mounted on the back of the backplane. The backplane has 7 card slots for IOSDL microboards; it is a cut-down version of the FORMBACK design. The boards are:

DSPCARRY - a carrier board for the ECAT and ECAT-X boards

SEROPT - the anemometer interface and opto-isolator circuit board

EPROM CONTROLLER

EPROM CARDS 1 - 4

A general assembly drawing and parts list are given in Appendix E

4.2 Circuit Descriptions

4.2.1 DSPCARRY

This board is required mainly to allow mounting of the ECAT, with its piggy-back ECAT-X, in the microboard slot. The board routes power to the ECAT J2 connector from the backplane. It connects the ECAT COM1 port (on J2) and the ECAT-X COM2 port (on J3) to the backplane. It connects the ECAT LPT1 port (on J2) to a 20 way IDC connector PL5 for connection to the EPROM logger. A reset signal for the EPROM logger is produced by differentiating the INIT-printer control line. Finally it connects the ECAT speaker output to the backplane for triggering the watchdog circuit.

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

4.2.2 SEROPT Rev. 2

This board includes the RS232/RS422 interface for the anemometer, using a MAX232 convertor with a 75176 line driver/receiver. The RS422 lines are protected by transient voltage suppressors and zener diodes, although these would probably not provide protection in the event of a lightning strike.

The board also includes a number of opto-isolators for the RS232 lines between units within the buoy and between the modules and the external monitoring equipment. ICs 3, 4, 5, 7 and part of IC6 isolate the Multimet, Sonic Processor COM2 and Formatter COM1 serial outputs to the monitoring equipment. Externally supplied +5V is required to activate the isolated outputs, so that these are inactive during normal operation (monitoring cable disconnected). IC 6 (7660) is used to generate a -5V supply from the external +5V supply.

ICs 8, 9, 18 and part of 10 isolate the Multimet and Sonic Processor COM2 serial outputs to the Formatter COM4 and COM3 ports; these are powered by the Formatter +5V supply. IC 18 (7660) is used to generate a -5V supply from the Formatter +5V supply.

ICs 11, 12, 17 and part of 15 isolate the anemometer RS232 Tx and Rx lines to the onboard raw data logging system (referred to on the circuit diagram as "disk"); these are powered by the +5V supply from the onboard raw logging system. IC 17 (7660) is used to generate a -5V supply from the onboard raw logging system +5V supply.

ICs 13, 14, 19 and part of 15 isolate the anemometer RS232 Tx and Rx lines to the raw data telemetry system (referred to on the circuit diagram as "radio"); these are powered by the +5V

supply from the onboard raw logging system. IC 19 (7660) is used to generate a -5V supply from the onboard raw logging system +5V supply.

The serial inputs to the board, with the exception of the anemometer signals, are made via Molex connectors. The serial outputs, again with the exception of the anemometer signals, are made via IDC connectors.

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

4.2.3 EPROM Logger Controller and Memory boards

For a complete description of the operation of these units, see the IOSDL EPROM logger handbook, ref. 1.

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.

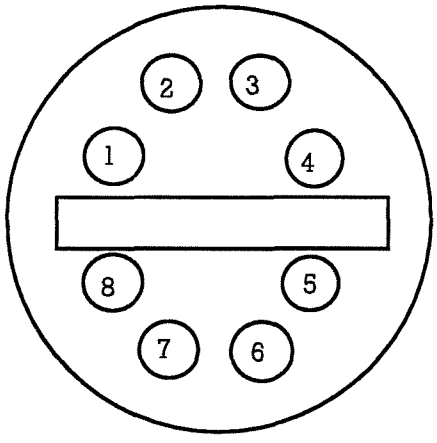


Figure 2 Formatter/Sonic Processor Lid

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) SONIC SENSOR Lemo connector FS2 to Sonic Motherboard SK2 and to Power Lemo Connector FS7

b) RAW DATA O/P Lemo connector FS4 to SEROPT board SK8

c) POWER Lemo connector FS7 to Sonic Motherboard and to Sonic Sensor Lemo Connector FS2

d) MET SERIAL I/P Lemo connector FS5 to SEROPT board SK3

a) Lid Connector FS2 (Lemo Series 3, 6 pin) to Sonic Motherboard SK2 (IDC10 free socket) and to Lid Connector FS7 (Lemo Series 3, 10 pin)

FS2 Pin	Function	SK2 Pin	FS7 Pin
1	Sensor Supply +V		6
2	Sensor Supply 0V		5
3	Serial A	2	
4	Serial B	1	
5	Serial Ground	3	
6	Screen/chassis	4	

b) Lid Connector FS4 (Lemo Series 3, 8 pin) to Sonic Processor SEROPT board SK8 (IDC10 free socket)

FS4 Pin	Function	SK7 Pin
1	+5V GCAT Raw	1
2	Sonic Tx	2
3	Sonic Rx	3
4	0V GCAT Raw	4
5	+5V HF Raw	5
6	Sonic Tx	6
7	Sonic Rx	7
8	+5V External I/P	8

c) Lid Connector FS7 (Lemo Series 3, 10 pin) to Sonic Motherboard SK1 (Weidmuller-Klippon 4 way free socket) and to Lid Connector FS2 (Lemo Series 3, 6 pin)

FS7 Pin	Function	SK1 Pin	FS2 Pin
1	ECAT 0V	1	
2	ECAT +5V	2	
3	EPROM Logger 0V	3	
4	EPROM Logger +15V	4	
5	Sonic Sensor 0V		2
6	Sonic Sensor +15V		1
7-8	n/c		
9-10	(Formatter Supplies)		

d) Lid Connector FS5(Lemo Series 3, 2 pin) to SEROPT board SK3 (Molex 4 pin socket))

FS5 Pin	Function	SK3 Pin
1	Met 0V	1
2	Multimet Serial I/P	4

5.3 Interboard Wiring

This includes:

a) ECAT bus J2 to DSPCARRY board PL4 (power, COM1 and LPT1)

b) ECAT-X J3 to DSPCARRY board PL3 (COM2)

c) DSPCARRY board SK5 to EPROM Controller SK2 (LPT1)

d) EPROM Controller SK2 to EPROM Data Cards 1-4 SK1s(Address and board selects)

e) Sonic Motherboard SK1 to Formatter BMPPROC2 board SK1 (Formatter +5V)

a) ECAT bus connector J2 (100 way IDC) to DSPCARRY board PL4 (50 way header) and PL2 (10 way header)

J2 Pin	Function	PL4 Pin
1 upper	STROBE- (CENTRONICS)	1
2 upper	AUTOFD- (CENTRONICS)	2
3 upper	D0 (CENTRONICS)	3
4 upper	ERROR- (CENTRONICS)	4
5 upper	D1 (CENTRONICS)	5
6 upper	INIT- (CENTRONICS)	6
7 upper	D2 (CENTRONICS)	7
8 upper	SLCTIN- (CENTRONICS)	8
9 upper	D3 (CENTRONICS)	9
10 upper	GND (CENTRONICS)	10
11 upper	D4 (CENTRONICS)	11
12 upper	GND (CENTRONICS)	12
13 upper	D5 (CENTRONICS)	13
14 upper	GND (CENTRONICS)	14
15 upper	D6 (CENTRONICS)	15
16 upper	GND (CENTRONICS)	16
17 upper	D7 (CENTRONICS)	17
18 upper	GND (CENTRONICS)	18

J2 Pin	Function	PL4 Pin
19 upper	ACK- (CENTRONICS)	19
20 upper	GND (CENTRONICS)	20
21 upper	BUSY (CENTRONICS)	21
22 upper	GND (CENTRONICS)	22
23 upper	PE (CENTRONICS)	23
24 upper	GND (CENTRONICS)	24
25 upper	SELECT (CENTRONICS)	25
26 upper	KCLK (KEYBOARD)	26
27 upper	GND (KEYBOARD)	27
28 upper	KDATA (KEYBOARD)	28
29 upper	+5V (KEYBOARD)	29
30 upper	RESET- (KEYBOARD)	30
31 upper	GND (COM1)	31
32 upper	RI (COM1)	32
33 upper	DTR (COM1)	33
34 upper	CTS (COM1)	34
35 upper	TxD (COM1)	35
36 upper	RTS (COM1)	36
37 upper	RxD (COM1)	37
38 upper	DSR (COM1)	38
39 upper	DCD (COM1)	39
40 upper	GND (VIDEO)	40
41 upper	INTENSITY	41
42 upper	GND (VIDEO)	42
43 upper	VID. OUT	43
44 upper	RED	44
45 upper	HSYNC	45
46 upper	GREEN	46
47 upper	VSYNC	47
48 upper	BLUE	48
49 upper	+5V SPKR	49
50 upper	AUDIO SPKR	50

J2 Pin	Function	PL2 Pin
1 lower	GND, 0V(S)	1
2 lower	GND, 0V(S)	2
3 lower	GND, 0V(S)	3
4 lower	GND, 0V(S)	4
5 lower	+5V, +5V(S)	5
6 lower	+5V, +5V(S)	6
7 lower	+5V, +5V(S)	7
8 lower	+5V, +5V(S)	8
9-50	not used	

b) ECAT-X COM2 Connector J3 (IDC10 free socket) to DSPCARRY board PL2 (10 way header)

J3 Pin	Function	PL2 Pin
1	DCD (COM2)	1
2	DSR (COM2)	2
3	RxD (COM2)	3
4	RTS (COM2)	4
5	TxD (COM2)	5
6	CTS (COM2)	6
7	DTR (COM2)	7
8	RI (COM2)	8
9	GND (COM2)	9
10	GND (COM2)	10

c) DSPCARRY board SK5 (IDC20 free socket) to EPROM Controller SK2(IDC50 free socket)

SK5 Pin	Function	SK2 Pin
1	not used	1
2	not used	2
3	BUSY, BUSY-P	3
4	not used, 0DAV-P	4
5	ERROR-, LAV-P	5
6	ACK-, DATOK-P	6
7	not used	7
8	not used	8

SK5 Pin	Function	SK2 Pin
9	D0, DMDI0-P	9
10	D1, DMDI1-P	10
11	D2, DMDI2-P	11
12	D3, DMDI3-P	12
13	D4, DMDI4-P	13
14	D5, DMDI5-P	14
15	D6, DMDI6-P	15
16	D7, DMDI7-P	16
17	STROBE-, DMAP-P	17
18	not used	18
19	not used	19
20	GND, 0V	20

d) EPROM Controller SK2 (IDC50 free socket) to EPROM Data Cards SK1 (IDC34 free sockets)

(chained connections to each SK2 of 4 Data Cards)

SK2 Pin	Function	SK1 Pin
21	MA0-P	1
22	MA1-P	2
23	MA2-P	3
24	MA3-P	4
25	MA4-P	5
26	MA5-P	6
27	MA6-P	7
28	MA7-P	8
29	MA8-P	9
30	MA9-P	10
31	MA10-P	11
32	MA11-P	12
33	MA12-P	13
34	MA13-P	14
35	MA14-P	15
36	MA15-P	16
37	MA16-P	17

SK2 Pin	Function	SK1 Pin
38	MA17-P	18
39	MA18-P	19
40	MA19-P	20
41	MA20-P	21
42	BS1-N	22
43	BS2-N	23
44	BS3-N	24
45	BS4-N	25
46	BS5-N	26
47	BS6-N	27
48	BS7-N	28
49	BS8-N	29
50	MA21-P	30

e) Sonic Motherboard SK1 (Weidmuller-Klippon 4-way free socket) to Formatter BMPPROC2 board SK1 (Weidmuller-Klippon 4-way free socket)

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

6. OPERATIONAL

The Sonic Processor and EPROM logger can be disconnected from the lid power connector by unplugging the orange plug-in terminal block, if required, while powering up the Formatter.

6.1 Procedure for Setting the Time/Date

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 10 way IDC ribbon cable connector from the ECAT-X J3 (COM2) - this connects to the Formatter COM6 port via the SEROPT board

plug the special ribbon cable, labelled "Husky to Sonic", into the Husky or PC 25 way COM1 port (use a 25 to 9 way adaptor if necessary) and into the ECAT-X J3 (COM2) 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

Apply power to the Sonic Processor; this will take about a minute to boot up. When the SETTIME application runs on the ECAT, 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 ECAT Date/Time.

This should be followed shortly by another message of the same format, showing the new time set in to the ECAT from a similar format message sent from the PC/Husky to the ECAT. The ECAT will, after a short pause, run the FFTC2 application.

Remove the ribbon cable from the ECAT COM2 port J3 and reconnect the ribbon cable from the DSPCARRY board.

Note that, in order to monitor the running of the FFTC2 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 ECAT-X COM2 port J3 via a suitable cable, e.g. that labelled "Husky to Sonic", 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. The Sonic messages to the Formatter (format given in Appendix B) can then be monitored; these should occur at about 12, 27, 42 and 57 minutes past the hour,

but will not be produced until the software has gone through a complete acquisition/processing cycle.

6.2 Procedure for Initialisation of the EPROM Logger

In addition to setting up the Sonic Processor, it is necessary to set up the EPROM logger, i.e. to open a new file for the data.

6.2.1 Getting Started

Equipment required :-

1) Dumb terminal or Computer with terminal emulation

- terminal configuration 2400 baud and no parity.

2) Serial cable, with switch to enable interactive mode communications on the EPROM Logger.

Connect the EPROM Logger to the terminal via the serial cable. Set the interactive switch on the serial cable ON and push the reset switch on the EPROM Logger Processor Board. The EPROM Logger's software will then enter into its interactive mode.

A Welcome message should appear on the terminal screen, followed by the current SETUP information.

The SETUP conditions will indicate the following :-

BPR - Bits per record (MultiMet = 68, Sonic Processors = 128)

BOARDS - Number of completely filled memory boards installed (Buoy EL's = 4)

CHIPS - Number of chips on a partially filled board installed (normally = 0)

To modify the SETUP parameters, if required, use the MODIFY command (all commands must be entered in UPPERCASE characters) as follows :-

For 4 full boards

4 BOARDS MODIFY <cr>

If there is a partially populated board (only one partially populated board is permissible and must be the highest board number installed) the number of chips on the board must be declared as follows :-

For 8 memory chips

8 CHIPS MODIFY <cr>

The system will check for any obvious errors in the input of BOARDS or CHIPS i.e. outside technical limitations of the EPROM Logger.

6.2.2 File Initialisation

Before data is collected the EPROM Logger Directory File structure must be initialised. The Directory commands are as follows :-

DIR <cr>	Displays the directory structure.
INIT-AFT <cr>	Removes the directory structure of any previous EPROM Memory cards.
SEARCH <cr>	Locates the Next Free EPROM Address.
OPEN <cr>	Initial command to 'open a new file'.
	Prompts the user for keyboard entry for a filename <cr>.

The switch on the serial Cable should be returned to the non-interactive position, and the cable disconnected from the EPROM Logger.

6.3 Procedure for EPROM Logger Replay

6.3.1 Requirements

Check that the Translation PC is available, and that an Eprom backplane with power supply is connected to the PC. Make sure that there is a controller card which has been programmed for 4800 baud, to go with your memory cards.

6.3.2 To Translate data to the PC

a) Insert the EPROM Memory cards with the data to be translated into backplane, and connect it via the ribbon cable, to the controller card

b) Switch on the Translation PC followed by the power to the EPROM Logger.

c) At the C> prompt change directories on the PC by typing 'CD \ DATATRANS' <cr>. This will result in the prompt DATATRANS> appearing.

d) Type 'ELOG' <cr> to activate the EPROM Logger program.

e) This will ask you to

'press any key' when ready

In response to 'do you have a colour monitor' type Y.

In response to 'No. of bytes per record' enter 68 for Multimet or 128 for the SONIC Processor.

f) You now enter into the Main menu of the program.

Select 1) and enter in a filename for the transferred data.

Select 3) to check Eprom logger communications.

Give a Reset on the controller card and this should generate the Setup information of the logger. If it is not the controller card that recorded the data check that the no. of Boards is correct, no. of extra Chips and that the no. of bytes per Record are correct.

Reset the directory by typing 'INIT-AFT' <cr> and then make the Next Free Address at the end of the data by typing 'SEARCH' <cr>. This may take quite a few minutes to complete .

g) Now press 'Escape' to return to the main menu.

Select 2) to start transfer

In response to 'do you want to start transfer' type Y cr

Now type 'DUMP-IBM' <cr> then answer N cr to the question 'do you want to dump all files'.

Press 'Escape'

h) you should now see the NO. GOOD REC incrementing and the Hard disc on the PC in operation.

i) On completion of the transfer you will be returned back to the main menu.

j) If you chose, you can examine the Binary data file with option 4) but you must go through the whole data set to exit the option , once started.

k) Select option 5) to convert the Binary file to ASCII

Enter filename of binary data

Enter new filename for ASCII data

Now the no. of records processed should be displayed.

l) When conversion is complete select option 6) to EXIT the eprom program and return to DOS.

7. SPECIFICATION

7.1 Supplies

The Sonic Processor requires a +5 Volt supply at ~ 360 mA

The EPROM logger requires a +15 Volt supply at ~ 28 mA

Both of these supplies are normally provided by DC to DC convertors in the DC-DC Convertor Unit.

7.2 Power Consumption

The power consumption of the Sonic Processor, complete with EPROM logger, is typically 2.25 Watts. The consumption including the DC-DC convertors is typically 142 mA at a primary bus supply voltage of +24 Volts, or 3.4 Watts, giving a conversion efficiency of about 66%. Note that this figure does not include the sensor power consumption.

7.3 Data Storage and Output

The spectral data and calculated parameters are stored on the EPROM logger in the format described in Appendix A.

The calculated parameters are output to the Formatter via the COM2 serial RS232 port in the format described in Appendix B.

7.4 Sampling

The anemometer transducers are fired in sequence at intervals of 1 mS; the complete set of firings to give 6 transit times (3 axes * 2 directions) takes 6 mS. Eight sets of transit times are averaged for each reading, giving a total sampling duration of 48 mS, so that the effective sampling rate is 20.83 Hz. The anemometer is operated in mode 1, in which the internally stored calibrations are applied by the anemometer processor giving a set of calibrated 16 bit U, V, W and C values for each sample. The anemometer also acquires the analogue compass reading, H, on its Analogue Input #1; the analogue input is sampled 10 times per second, but a 16 bit value is output with each set of velocity data. The Sonic Processor software acquires a total of 12288 sets of U, V, W, C and H readings over a period of about 590 seconds (nearly 10 minutes) starting at each quarter-hour, i.e. at 00, 15, 30 and 45 minutes past each hour.

The U, V and W values, although transmitted as 16 bit binary (two's complement) have the range -6000 to +6000 , in units of cm/s; a value of -10000 is transmitted if an error (such as blocking of the path) occurs.

The speed of sound, C, value has the range 0 to 18500, in units of 2 cm/s, giving a full scale of 370 m/s; again a value of -10000 is transmitted if an error (such as blocking of a path) occurs.

The heading, H, value has the range 0 to 5000, in units of mV, although permissible values lie within the range 2048 to 4096, as described in 7.5.2, below.

7.5 Specification of Sensors

7.5.1 Anemometer

The anemometer is a Gill Instruments 3 axis asymmetrical research anemometer, mounted on the buoy mast ring with its "North" marking pointing in the direction of the buoy reference North mark,. This results in the wind entering the sensed volume via the clear aperture when the buoy's wind vane is successfully aligning the buoy reference mark into the wind.

The sensor is mounted on a two part base which both waterproofs the base of the anemometer housing and allows two separate Lemo connectors to be used for the Power/Digital Signal and Analogue Input connections. The base is held by a clamp welded to North side of the mast ring.

7.5.2 Compass

The buoy heading sensor (Compass) is a Digicourse gimballed unit with 8 bit Gray coded parallel output; the unit is housed in the main buoy canister, with a key way to ensure correct alignment relative to the buoy North.

The compass is sampled at a rate of 1 Hz and the latched output is converted to binary. The binary output is sent, via opto-isolators, to the Multimet system for logging and is converted to an analogue voltage in the range +2.048 Volts to +4.096 Volts for acquisition by the anemometer. This gives a compass count ranging from 2048 to 4096 digits, which is converted to an 8 bit value by taking (count - 2048) / 8. The compass and the above interface circuits are housed in a cylindrical unit; this unit is fully described in the Compass Unit documentation

8. ACKNOWLEDGEMENTS

The development of this equipment was funded by the MAFF Flood and Coastal Defence Division under commission FD0603.

9. REFERENCES

1. Griffiths, G. and Lewis, A. 1988 IOSDL EPROM Logger Handbook, unpublished manuscript.

2. Pascal, R.W. 1993 Romdisk for SONIC ECAT system, unpublished manuscript.

3. Gill Instruments Ltd. 1992 Solent Research Ultrasonic Anemometer, Product Specification Issue 4.1

APPENDICES

Appendix A The EPROM Logger Data Storage Format

The format consists of a spectrum header, followed by the mean wind speed reading and 255 estimates of the form $\log_{10}(PSD * freq^{5/3})$. These are followed by a parameters header and, finally, the computed parameters, i.e.

Spectrum Header

jjjhhmmFFTSpd<CR> (where jjj is Julian Day, hh is hour, mm is minute of start)

mw.ws<CR> (where mw.ws is mean resultant wind speed)

then 255 lines, each with the format:

+h.est<CR> (where h.est are spectral estimates
for the 2nd to the 256th line)

Parameters Header

jjjhhmmPSDSpd<CR> (where jjj is Julian Day, hh is hour, mm is minute of start)

IDID<CR> (Sonic Sensor ID)

001<CR> (Records per file)

F1.F1<CR> (Lower frequency for averaging range)

F2.F2<CR> (Upper frequency for averaging range)

1<CR> (Sonic Mode)

Computed Parameters

mw.ws<CR> (where mw.ws is mean resultant wind speed)

+nm.ws<CR> (where +nm.ws is mean wind speed from North)

+em.ws<CR> (where +em.ws is mean wind speed from East)

+vm.ws<CR> (where +vm.ws is mean wind speed upwards)

cme.an<CR> (where cme.an is mean speed of sound)

hea.dg<CR> (where hea.dg is mean buoy heading)

+p.sdpsd<CR> (where +p.sdpsd is mean PSD over range F1 to F2)

+a.lalal<CR> (where +a.lalal is least squares fit 'a' coefficient)

+b.bbbbbe+bbb<CR> (where +b.bbbbbe+bbb is least squares fit 'b' coefficient)

END<CR><LF><CR>

making a total of 1920 characters, which are transferred to the EPROM logger in 15 blocks of 128 bytes.

Appendix B The Sonic Processor Serial Output Message

The message is output at approximately 12, 27, 42 and 57 minutes past the hour via the COM2 port at 2400 baud, 8 data bits, 1 stop bit, no parity. The 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 Minutes (00 - 59)

SS is Seconds (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

Appendix C Source Code for SETTIME Application

```
/*****************************SETTIME.C*****************************\

        Execution of this program is included in the autoexec.bat file
        for the sonic buoy sonic processor. It allows the dsp processor
        clock to be optionally reset at boot up time. This is done by
        connecting a PC running the GWBasic program "settime.bas" to
        the COM2 port.
        The DSP time is then set to the PC time.
        If the PC is not connected, this program times out.
        The autoexec then runs the sonic acq/processing prog fftc2.

        Version 1

        CHC September 1991
\*************************************************************************/

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


main()
{
char rsout[45];          /* string sent via serial port */
char stbuf[35];          /* string buffer used for conversions */

long loop_ctr;           /* used for time out */

struct dosdate_t date;
struct dostime_t time;

unsigned status, data;

int ch, ch_hit, port=1;  /* port = 0 for COM1, =1 for COM2 */


/* initialise com2 port, 2400 baud, 8bit data, no parity, 1 stop bit */

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

_bios_serialcom(_COM_INIT, port, data);


/* get calculated date/time and format into string rsout */

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

strcpy(rsout, "Date: ");

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

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

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

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

strcat(rsout, "  Time: ");

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

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

itoa(time.second, stbuf, 10);
strcat(rsout, stbuf);
strcat(rsout, "Q");
/* end string with a "Q" and send it to the external PC */

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

for (ch = 0; ch < strlen(rsout); ch++)
    {
    do
        {
        /* test until transmit buffer is empty, up to 100 tries */
        status = 0x2000 & _bios_serialcom(_COM_STATUS, port, 0);
        loop_ctr++;
        }
    while ( (status != 0x2000) && (loop_ctr < 100) );

    /* send character but abort program if port times out */
    if(_bios_serialcom(_COM_SEND, port, rsout[ch]) > 0x7fff)
        {
        exit(0);
        }
    /* terminate for loop if port times out */
    if ((status & 0x8000) == 0x8000)
        {
        printf("RS232 COM%d timed out\n", port + 1);
        break;
        }
    }

ch=0;
loop_ctr = 0;

/* Now get response date/time string sent by external PC, if connected */
do
    {
    /* get received characters and assemble into string, until LF detected,
    - up to 10000 tries */
    status = 0x100 & _bios_serialcom(_COM_STATUS, port, 0);

    if (status == 0x100)        /* receive buffer contains character(s) */
        {
        /* get and print a received character */
        ch_hit = 0xff & _bios_serialcom(_COM_RECEIVE, port, 0);
        printf("%c", ch_hit);

        /* if character is a "D", reset string buffer pointer */
```

```
            if (ch_hit == 68)
                {
                ch = 0;
                }
            /* put character into string buffer */
            stbuf[ch] = ch_hit;
            ch++;
            }
        loop_ctr++;
        }
    while ( (ch_hit != 10) && (loop_ctr < 10000) );

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

    /* print received string and load the date and time structures with received values */
    printf("\n%s\n", stbuf);
    date.month = 10 * (stbuf[5] - 48) + stbuf[6] - 48;
    date.day = 10 * (stbuf[8] - 48) + stbuf[9] - 48;
    date.year =1900 + 10 * (stbuf[13] - 48) + stbuf[14] - 48;
    time.hour = 10 * (stbuf[21] - 48) + stbuf[22] - 48;
    time.minute = 10 * (stbuf[24] - 48) + stbuf[25] - 48;
    time.second = 10 * (stbuf[27] - 48) + stbuf[28] - 48;

    if (loop_ctr < 10000)
        {
        /* set the Sonic processor date */
        if (_dos_setdate(&date) != 0)
            {
            printf("Error in date set\n");
            }
        /* set the Sonic processor time */
        if (_dos_settime(&time) != 0)
            {
            printf("Error in time set\n");
            }

    /* Now assemble the received date/time into string rsout and send it back to the external PC as
    confirmation */
        strcpy(rsout, "Date: ");

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

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

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

        strcat(rsout, "  Time: ");

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

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

        itoa(time.second, stbuf, 10);
```

```
        strcat(rsout, stbuf);
        strcat(rsout, "Q");

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

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

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

}
```

Appendix D.1 Source Code for FFTC2 Application

```
/*****************************************************************************
                Program FFTC4.C
                Version 1.0  August 1993

                Author CHC
                Compile using command line:
                    qcl /AM /Zr /FPi87 fftc4.c /F 9000 mlibc7.lib
                Rename as FFTC2.EXE for use with standard autoexec

                Sonic processing program: for use with Sonic Buoy, giving
                .PRN files to EPROM logger and COM2 port o/p to formatter.

                Runs on ECAT+ECAT-X or on 286/386+maths co-processor
                Install  as FFTC2.EXE in ROMdisk together with FASTCOM.EXE
                and SETTIME.EXE for ECAT configuration and RTCN.EXE for
                DOS clock updating

                VDU output can be removed by setting DISPLAY to 0
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <conio.h>
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <process.h>
#include <dos.h>
#include<bios.h>
#include<malloc.h>

#define DISPLAY         1
```

```
#define RATE            20.83
#define SECTIONS        12              /* = number of sections to be FFT'd */
#define LINES           256             /* = half the effective no. of samples per section */
#define R_TO_D          57.29578
#define pi              3.141592654
#define FREQ1           2.0
#define FREQ2           4.0

#define RAMFILE         "c:\\testfile"
/* filename for raw data in RAMdisk */

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


typedef enum {FALSE, TRUE} boolean;

/********** Function declarations ***********/
char * aform(int, int);
char * aform1(double, char *, double, double);
double dcfilter(int);
void four1(int);
double fftconsts(int, int, int *, int *, int *, double *, double *);
int getdat(int, int, const char *, char *, double *, double *,
                        double *, double *, double *, double *, double *);
void far harderror_handler(unsigned, unsigned, unsigned far *);
double regres(int, double *, double *, double *, double *, double *, double *);
void send_rs232(char *, double, double, double, double, double,
                                        double, double, double, double);
void wait1(void);
void wait2(void);
char * wait_start(int, int *, int *);
void window(int, double, double);

void show_errors(char *);
void wipe_line(int);

/**********Global Variables**********/
char dum_ch[1024];                      /* added to stop far pointer errors in eprom write */
char eprom[2048];
char buffer[130];
char strval[25];
char message[85];
char julian[10];
char ser_no[12];

chartenv env;
FILE *fh;
double a[4 * LINES + 2], p[LINES + 2];  /* double necessary for precision */
double fu[LINES + 2];                   /* fu[ ] = F() in PKT FSprog */
int rows = 24;
unsigned char error_flag;
unsigned lastcomp = 0;


/******************Start of Main******************/
main()
{
char *ptr;
char **g = &ptr;

char samples[10];
char gilltime[40];
```

```
 char subhead[40] = "Section: ";
 char fft_sec[10];
 char head[40] = "Last Start";
 char oi_comm[20];

 char spec_file[25], spec_ffile[25], spec_flfile[20], spec_fflfile[20];
 char raw_filename[20];
 char baud[5], son_mode[5];
 char back_up[30];
 char sonic_id[6];

 double a1, a2, b;

 double buoy_heading, c_mean, cos_mean, den, dfr, dummy, east_mean;
 double fact, fm, fp, f1, f2, fr1, fr2;
 double inv_freq, mean, north_mean, psd, p1, p2;
 double r, rms1, scale_x, scale_y, sea, seb, sin_mean, sumw, vert_mean;

 double yvalue[LINES + 2];

 double psd_set[101];
 double meanws_set[101];

 int fft, gflag, good_reads, i, j, j2;
 int mode = _VRES16COLOR;
 int n, nans4, nrec = 2 * LINES;
 int nr2, nr23, nr24, nspec = LINES, qtr, re, sample;

 int xp, yp;

 unsigned good_total, status;

 /*************************START OF CODE*********************************/

 #if DISPLAY == 1
     {
     system("cls");
     printf("Buoy Acquisition Program\n");
     }
 #endif

 _harderr(harderror_handler);              /* set up hardware error handling */

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


 /********** Set up parameters for FFT, etc **********/

 sumw = fftconsts(nrec, nspec, &nr2, &nr23, &nr24, &fm, &fp);
 /* nr2 is total number of samples per section */

 nans4 = nspec/4;
 strcpy(baud, "1");                        /* 4800 baud */
```

```
strcpy(sonic_id, "XXXX");
strcpy(son_mode, "1");

inv_freq = (double) log10( (double) nrec / RATE );
/* Calculate frequencies for binning */
fu[1] = 0.0;
for (i = 2; i <= nspec; i++)
    {
    fu[i] = 10.0 * log10( (double) RATE * (i-1)/(2 * nspec) );
    }

ltoa( (long) SECTIONS * (long) nr2, samples, 10 );

#if DISPLAY == 1
    {
    printf("End of setup\n");
    }
#endif

/************ start of continuous loop ************/
do
    {
        printf("\a");                       /* Bell for Watchdog */
        strcpy(julian, wait_start(rows, &sample, &qtr));
        printf("\a");                       /* Bell for Watchdog */

        strcpy(message,"");

        mean = 0.0;
        for (i = 1; i <= nspec; i++)        /* Initialise p[ ] array */
            {
            p[i] = 0.;
            }
        /*>>>>>>>>> call fastcom.exe, datafile c:\testfile, mode 1, 4800 baud,
                        SECTIONS*nr2 samples, 1 analogue input >>>>>>>>>*/

        #if DISPLAY == 1
            {
            wipe_line(rows);
            printf("Record %d - getting %s samples from Sonic      (Wait)",
                        sample, samples);
            }
        #endif

        if ( spawnl(P_WAIT, "fastcom.exe", "fastcom.exe",
                        RAMFILE, son_mode, baud, samples, "1", NULL) == -1)
            {
            #if DISPLAY == 1
                {
                printf("Could not run FASTCOM - Fatal Error\n");
                }
            #endif
            exit(0);
            }

        #if DISPLAY == 1
            {
            system("cls");
            printf("Samples acquired . . . Starting to Process");
            }
        #endif
```

```
wait1();

/******************** Start actual calcs ********************/

good_reads = 0;
north_mean = 0.;
east_mean = 0.;
vert_mean = 0.;
c_mean = 0.;
sin_mean = 0.;
cos_mean = 0.;

for (fft = 1; fft <= SECTIONS; fft++)
    {
    /* Get data section, calc mean, apply window, do fft */
    if (getdat(fft, nr2, RAMFILE, gilltime, &dummy, &north_mean,
            &east_mean, &vert_mean, &c_mean, &sin_mean, &cos_mean) == 0)
        {
        #if DISPLAY == 1
            {
            printf("#");
            }
        #endif
        good_reads++;
        mean += dcfilter(nr2);
        window(nrec, fm, fp);
        four1(nrec);

        /* convert complex estimates to power */
        a[1] = a[1] * a[1] + a[2] * a[2];

        for (j = 2; j <= nspec; j++)
            {
            j2 = j * 2;
            a[j] = a[j2] * a[j2] + a[j2 - 1] * a[j2 - 1]
                                 + a[nr24 - j2] * a[nr24 - j2]
                                 + a[nr23 - j2] * a[nr23 - j2];
            }

        den = sumw * nr2;       /* corrected sumw 11/02/92 */

        /* accumulate power estimates */
        for (i = 1; i <= nspec; i++)
            {
            p[i] += a[i];

            if (a[i] <= 0.)
                {
                #if DISPLAY == 1
                    {
                    printf("Error a[%d] %e\n", i, a[i]);
                    }
                #endif
                a[i] = 0.;
                }
            else
                {
                a[i] = log10(a[i]/den);
                }
            yvalue[i] = (double) a[i];
            }

        /* effectively multiply spectrum by f^5/3 (add log10(freq^5/3))
```

```
            and convert to PSD by adding log10(1/estimate spectral width) */
            for (i = 2; i<=nspec; i++)          /* i.e. i=96->256 */
                {
                a[i] += 1.66666667 * log10((i-1) * RATE / nrec) + inv_freq;
                yvalue[i] = (double) a[i];
                }

            }                                   /* end of if(getdat.....) block */
        else
            {
            #if DISPLAY == 1
                {
                printf("\a");                   /* data faulty */
                }
            #endif
            }
        }                                       /* end of fft loop */

    fclose(fh);                                 /* close testfile after all sections read */
    wait1();

    if (good_reads == 0)
        {
        #if DISPLAY == 1
            {
            _settextposition(rows / 2, 20);
            printf("FATAL ERROR:- BAD DATA FROM SONIC\n");
            }
        #endif
        exit(0);                                /* abort from program - leads to re-boot */
        }

    /* Correct power estimates for windowing, etc */

    den = sumw * nr2 * good_reads;              /* corrected sumw 11/02/92 */

    for (j = 1; j <= nspec; j++)
        {
        p[j] /= den;
        }


    mean /= good_reads;

    good_total = (unsigned) good_reads * nr2;

    north_mean /= good_total;
    east_mean /= good_total;
    vert_mean /= good_total;
    c_mean /= good_total;
    sin_mean /= good_total;
    cos_mean /= good_total;
    buoy_heading = 30 - R_TO_D * atan2(sin_mean, cos_mean);
    if (buoy_heading < 0.)
        {
        buoy_heading += 360.;
        }
    if (buoy_heading > 360.)
        {
        buoy_heading -= 360.;
        }

    #if DISPLAY == 1
```

```
        {
        _settextposition(rows,10);
        printf("                                    ");
        }
#endif
wait1();

for (i = 2; i <= nspec; i++)
        {
        /* convert a[i] to PSD */
        fact = pow((double) (i - 1) * RATE / nrec, 1.66666667) * nrec / RATE;
        p[i] *= fact;
        a[i] = log10(p[i]);
        yvalue[i] = a[i];
        }

#if DISPLAY == 1
        {
        printf("Preparing Spectrum for EPROM\n");
        }
#endif
strcpy(eprom, julian);
strcat(eprom, "FFTSpd\n");
if (sprintf(strval, "%05.2f\n", mean) != 6)
        {
        strcpy(strval, "99.99\n");
        }
strcat(eprom, strval);

for (i = 2; i <= nspec; i++)
        {
        if (sprintf(strval, "%+06.3f\n", yvalue[i]) != 7)
                {
                strcpy(strval, "+9999.\n");
                }
        strcat(eprom, strval);
        }

wait1();

/* Fit regression line */
psd = regres(nrec, &r, &rms1, &sea, &seb, &a1, &b);

psd = log10(psd);
sea = log10(fabs(sea));
seb = log10(fabs(seb));

if (a1 > 0.)
        {
        a1 = log10(a1);
        }
```

```
 else
      {
      a2 = a1;
      a1 = -9.99;
      }

 if (sprintf(strval, "%sPSDSpd\n", julian) != 14)
      {
      strcpy(strval, "jjjhhmmPSDSpd\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%s\n%.3d\n%05.2f\n%05.2f\n%s\n",
                    sonic_id, 1, FREQ1, FREQ2, son_mode) != 23)
      {
      strcpy(strval, "XXXX\nRRR\nf1.f1\nf2.f2\nM\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%05.2f\n", mean) != 6)
      {
      strcpy(strval, "99.99\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%+06.2f\n", north_mean) != 7)
      {
      strcpy(strval, "+99.99\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%+06.2f\n", east_mean) != 7)
      {
      strcpy(strval, "+99.99\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%+06.2f\n", vert_mean) != 7)
      {
      strcpy(strval, "+99.99\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%06.2f\n", c_mean) != 7)
      {
      strcpy(strval, "999.99\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%06.2f\n", buoy_heading) != 7)
      {
      strcpy(strval, "999.99\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%+08.5f\n", psd) != 9)
      {
      strcpy(strval, "+9.99999\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%+08.5f\n", a1) != 9)
      {
      strcpy(strval, "+9.99999\n");
      }
 strcat(eprom, strval);
 if (sprintf(strval, "%+13.5e\n", b) != 14)
      {
      strcpy(strval, "+9.99999e+999\n");
      }
 strcat(eprom, strval);
```

```
strcat(eprom, "END\n\n");
#if DISPLAY == 1
    {
    for (i = 0; i < 1920; i++)
        {
        if (*(eprom + i) == 0)
            {
            printf("Error ");
            }
        }
    }
#endif

if (strlen(eprom) != 1920)                    /* 12.9 Mbytes for 70 days */
    {
    #if DISPLAY == 1
        {
        printf("Formatting error in EPROM data\n");
        }
    #endif
    exit(0);
    }
else
    {
    #if DISPLAY == 1
        {
        printf("Writing data to EPROM\n");
        }
    #endif
    status = _bios_printer(_PRINTER_STATUS, 0, 0);
    printf("Initial status %d\n", status);      /* debug */
    i = 0;
    while ( ( (status & 0x08) > 0 ) && (i < 20) )
    /* bit 3 is inverse of the logger LAV line -
            Reset logger up to 20 times until LAV goes high */
        {
        status = _bios_printer(_PRINTER_INIT, 0, 0);
        wait1();
        i++;
        }
    #if DISPLAY == 1
        {
        printf("Status %d Tries %d\n", status, i);
        }
    #endif

    if (i < 20)
        {
        for (j = 0; j < 1920; j += 128)       /* 15 writes of 128 bytes */
            {
            #if DISPLAY == 1
                {
                printf("%d ", j);
                }
            #endif
            for (n = 0; n < 128; n++)
                {
                status =_bios_printer(_PRINTER_WRITE, 0,
                                        (unsigned) * (eprom + j + n));
                #if DISPLAY == 1
                    {
                    printf("%02x ", status);
                    }
```

```
                    #endif
                    /* for (i = 0; i < 5000; i++); */
                    }
                /* wait2(); */
                /* check status as an alternative to wait2() -
                Wait for up to 2.5 sec for LAV (bit 3) to go high */
                i = 0;
                status = _bios_printer(_PRINTER_STATUS, 0, 0);
                while ( ( (status & 0x08) > 0 ) && (i < 5) )
                    {
                    status = _bios_printer(_PRINTER_STATUS, 0, 0);
                    wait1();
                    i++;
                    }
                #if DISPLAY == 1
                    {
                    printf("Status %d\n", status);
                    }
                #endif
                }
            }
        }

        #if DISPLAY == 1
            {
            /* system("cls"); */
            _settextposition(rows - 3, 0);
            printf("Mean WS=%5.2fm/s, PSD*f^5/3=%+5.2f (%d-%dHz), Fit=%+5.2f\
                        %+10.2e*x    ", mean, psd, (int) FREQ1, (int) FREQ2, a1, b);

            _settextposition(rows - 2, 0);
            printf("(N=%+6.2f:E=%+6.2f:V=%+6.2f:C=%6.2fm/s:Head=%5.1f)\n",
                        north_mean, east_mean, vert_mean, c_mean, buoy_heading);

            printf("\nsending data to formatter\n");
            }
        #endif

        send_rs232(gilltime, psd, mean, north_mean, east_mean, vert_mean,
                        c_mean, buoy_heading, a1, b);
        #if DISPLAY == 1
            {
            system("cls");
            }
        #endif
    }
while (TRUE);                           /* end of do loop */

/* _settextposition(rows - 1,0); */

exit(0);
}                                       /* end of main function */

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

/************ FFTCONSTS sets parameters for fft ************/

double fftconsts(int nrec, int nspec, int *nr2, int *nr23, int *nr24,
            double *fm, double *fp)
{
int j;
double sumw, w;
double alpha = 31.41592654/nrec;
```

```
*nr2 = nrec * 2;
*nr24 = *nr2 + 4;
*nr23 = *nr2 + 3;

*fn = nspec - .5;
*fp = 1/(nspec + .5);

sumw = 0.;

/* Calculate weights for partial cosine taper */

for (j = 1; j <= LINES; j++)
    {
    if (j <= nrec/10)
        {
        sumw += 0.5 * pow(1. + cos(alpha * (LINES - j)), 2.);
        }
    else
        {
        sumw += 2.;
        }
    }

return sumw;
}

/************ GETDAT loads data from diskfile ************/

int getdat(int fft, int nr2, const char *ram_file, char *ch,
           double *addr_dummy, double *addr_north_mean,
           double *addr_east_mean, double *addr_vert_mean,
           double *addr_c_mean, double *addr_ssum, double *addr_csum)
{
double av1, av2, av3, c, cr = cos(.5236), sr = sin(.5236), u, v, w, res;
double lf_ck = 0.5 * 0.149 * 29491200;
/* = half path length ( in m) * counting clock frequency */

int dec, sign;

int i, j, k, theta;
int dbuff[10];
/* In mode1, dbuff holds 10 bytes velocity: 4 off (msbyte,lsbyte) */
/* These are 100*u, 100*v, 100*w, 50*c in m/s, plus comp_rdg (0-5000) */

unsigned comp;

if ( fft == 1 )
    {
    if ( (fh = fopen(ram_file, "rb")) != 0)
        {
        /* read header Mode<sp> */
        for (i = 0; i < 5; i++)
            {
            fgetc(fh);
            }
        if ( fgetc(fh) != '1' )
            {
            #if DISPLAY == 1
                {
                printf("\a");
                }
            #endif
```

```
                return(1);                      /* abort if not Mode 1 */
                }


            for (i = 0; i < 8; i++)             /* <lf>Analog<sp> */
                {
                fgetc(fh);
                }
            if (fgetc(fh) != '1')
                {
                return(1);                      /* abort if Analog Channels not 1 */
                }

            fgetc(fh);                          /* read the <lf> */

            i=0;
            do
                {
                ch[i] = (char) fgetc(fh);
                i++;
                }
            while ( (ch[i - 1] != 0x0a) && (i < 40) );

            ch[i-1] = 0;                        /* replace LF with string terminator */
            /* Resulting string is:
                    "Time hh:mm:ss Date mm/dd/yy" */

            }
        else
            {
            #if DISPLAY == 1
                {
                wipe_line(rows);
                printf("Could not open file\n");
                }
            #endif
            return(1);
            }

    }                                           /* end of block for fft == 1 */

/* for all values of fft */


for (j = -1; j <= 0; j ++)
    {
    for (i = j+2; i <= nr2; i += 2)
        {
        /* get 3 * 2byte vel compts plus 1 * 2byte vel of sound plus 2byte comp */
        for (k = 0; k <= 9; k++)
            {
            dbuff[k] = fgetc(fh);
            if (ferror(fh) != 0)
                {
                return(2);
                }
            }

        /* convert from motorola format to int format */
        u = 0.01 * (int) (dbuff[1] + (dbuff[0] << 8));
        v = 0.01 * (int) (dbuff[3] + (dbuff[2] << 8));
        w = 0.01 * (int) (dbuff[5] + (dbuff[4] << 8));
        c = 0.02 * (int) (dbuff[7] + (dbuff[6] << 8));
```

```
            if ((dbuff[7] == 0xf0) && (dbuff[8] == 0xd8))
                {
                return(2);                          /* path was blocked on 1 or more axes */
                }
            if ((u >60.) || (u < -60.) || (v > 60.) || (v < -60.) || (w > 60.) || (w < -60.))
                {
                return (3);
                }

            /* compass 8 bit value = (sonic count - 2048) / 8
            i.e. (sonic lsbyte / 8) + (sonic msbyte * 256 / 8) - 256 */

            comp = (dbuff[9] >> 3) + (dbuff[8] << 5) - 256;
            if (comp < 0)
                {
                comp = 0;
                }
            if (comp > 255)
                {
                comp = 255;
                }
            if ( (i > 1) && (abs(comp -lastcomp) > 64) && (abs(comp - lastcomp) < 192) )
                {
                comp = 0;
                }
            lastcomp = comp;                        /* NB comp range 0-255 */

            sr = sin(.5236 - 0.0245437 * comp);
            cr = cos(.5236 - 0.0245437 * comp);

            *addr_north_mean += (u * cr + v * sr);
            *addr_east_mean += (v * cr - u * sr);
            *addr_vert_mean += w;
            /* above values are vector averaged north, east and vertical compts */

            *addr_c_mean += c;

            *addr_ssum += sr;
            *addr_csum += cr;

            /* put resultant horiz vel. in array a[ ] (start address a_ptr) */
            /* a[i] = sqrt(u * u + v * v);   removed w^2 term for this vn */
            a[i] = sqrt(u * u + v * v + w * w);          /* added w^2 term for this vn */
            /* printf("%5.3f\n", a[i]); */
            }                                       /* end of i loop */
        }                                           /* end of j loop */

return(0);
}

/* returns    0 if ok
              1 if failure to open file or header incorrect
              2 if error during read
              3 if data out of range
(also returns array of nr2 resultant wind speeds in a[ ]) */
```

```
/************ DCFILTER removes mean from data ************/

double dcfilter(int nr2)
{

int i;
double tot = 0.;

for (i = 1; i <= nr2; i ++)
     {
     tot += a[i];
     }
tot = tot/nr2;
for (i = 1; i <= nr2; i ++)
     {
     a[i] -= tot;
     }

return tot;
}


/************ WINDOW applies partial cosine data window ************/

void window(int nrec, double fm, double fp)
{
int j, j2, nr2 = 2 * nrec;
double alpha = 31.41592654/nrec, w;

for (j = 1; j <= nrec/2; j++)
     {
     if (j <= nrec/10)
          {
          j2 = 2 * j;
          w = 0.5 * (1 + cos(alpha * (LINES - j)));
          a[j2] *= w;
          a[j2 - 1] *= w;
          a[nr2 - j2 + 1] *= w;
          a[nr2 - j2 + 2] *= w;
          }
     }

}


/************ FOUR1 does fft ************/

void four1(int nrec)
{
int i, j = 1, l, m, n = 2 * nrec, s;
double tr, ti, te, t, wpr, wpi, wr, wi, wt;

for (i = 1; i <= n; i += 2)
     {
     if (j > i)
          {
          tr = a[j];
          ti = a[j + 1];
          a[j] = a[i];
          a[j + 1] = a[i + 1];
          a[i] = tr;
          a[i + 1] = ti;
          }
```

```
        m = (int) n/2;

        while ((m >= 2) && (j > m))
            {
            j -= m;
            m /= 2;
            }
        j += m;
        }

    l = 2;
    while (n > l)
        {
        s = 2 * l;
        t = 2 * pi/l;
        te = sin(.5*t);
        wpr = -2 * te * te;
        wpi = sin(t);
        wr = 1.;
        wi = 0.;

        for (m = 1; m <= l; m += 2)
            {
            for (i = m; i <= n; i += s)
                {
                j = i + l;
                tr = wr * a[j] - wi * a[j + 1];
                ti = wr * a[j + 1] + wi * a[j];
                a[j] = a[i] - tr;
                a[j + 1] = a[i + 1] - ti;
                a[i] += tr;
                a[i + 1] += ti;
                }
            wt = wr;
            wr += wr * wpr - wi * wpi;
            wi += wi * wpr + wt * wpi;
            }
        l = s;

        }
}


/************ REGRES fits regression line ************/

double regres(int nrec, double *r, double *rms1, double *sea, double *seb,
                        double *a1, double *b)
{
int i, i1, i2, n;
double psd, xm, xn, ym, yn1;
/* have to use yn1 as yn appears to be in the include files */

double sx = 0., sy = 0., sxx = 0., sxy = 0., syy = 0., ssa, ssb, ssr;

i1 = (int) (1 + (float) (FREQ1 * nrec/RATE));
i2 = (int) (1 + (float) (FREQ2 * nrec/RATE));
psd = 0.;
n = 0;
for (i = i1; i <= i2; i++)
    {
    psd += p[i];
    n++;
    }
```

```
 psd /= n;                                   /* mean PSD over range FREQ1 to FREQ2 */

 xm = fu[1];
 ym = p[1];
 n--;

 for (i = i1 + 1; i <= i2; i++)
      {
      xn = fu[i] - xm;
      yn1 = p[i] - ym;
      sx = sx + xn;
      sy = sy + yn1;
      sxx = sxx + xn * xn;
      sxy = sxy + xn * yn1;
      syy = syy + yn1 * yn1;
      }

 sxx = sxx - (sx * sx) / (double) n;
 sxy = sxy - (sx * sy) / (double) n;
 xm = xm + sx / (double) n;
 syy = syy - (sy * sy) / (double) n;



 *a1 = ym + sy / (double) n;
 *b = sxy / sxx;
 ssa = *a1 * *a1 * (double) n;
 ssb = *b * sxy;
 ssr = syy - ssb;
 *a1 = *a1 - *b * xm;
 *rms1 = ssr / (double) (n - 2);


 if (*rms1 < 0.)
      {
      #if DISPLAY == 1
           {
           printf("RMS negative (%e)- is data OK?\n", *rms1);
           }
      #endif
      *rms1 = 0.;
      *r = 0.;
      *sea = 10000.;
      *seb = 10000.;
      }
 else
      {
      *r = (double) ( sxy / sqrt( sxx * syy ) );
      *sea = (double) sqrt( (double) *rms1 / (double) n );
      *seb = (double) sqrt( (double) *rms1 / sxx );
      *rms1 = (double) sqrt( (double) *rms1 );
      }
 return psd;
 }


 /************ WAIT_START waits for start of next process ************/
 char * wait_start(int rows, int * sample_no, int * qtr)
 {
 char cur_time[10], julian[10], last_time[10];
 div_t quarters;
 int sample = 0;
 time_t tnow;
```

```
struct tm *gmt;

#if DISPLAY == 1
    {
    _settextposition(rows,0);
    printf("Waiting for next Record Start . . . \n");
    }
#endif
do
    {
    time(&tnow);
    gmt = gmtime(&tnow);
    quarters = div(gmt->tm_min, 15);
    _strtime(cur_time);
    if (cur_time[7] != last_time[7])
        {
        #if DISPLAY == 1
            {
            _settextposition(rows, 46);
            printf("Day %d: %s              ", 1 + gmt->tm_yday, cur_time);
            }
        #endif
        }
    strcpy(last_time, cur_time);

    if ((quarters.quot == 3) && (gmt->tm_hour == 23)
                    && (gmt->tm_min < 59) && (sample == 0))
        {
        /* reset the DOS clock just before midnight
            (don't risk it if too close to midnight) */
        if ( spawnl(P_WAIT, "a:\\rtcn.exe", "a:\\rtcn.exe", "2", NULL) == -1)
            {
            #if DISPLAY == 1
                {
                printf("Could not run RTCN\n");
                }
            #endif
            }
        sample = 1;                         /* to prevent multiple setting */
        }
    }
while ( ( quarters.rem != 0 ) || ( gmt->tm_sec != 0 ) );

strcpy( julian, aform((1 + gmt->tm_yday), 3) );
strcat( julian, aform((gmt->tm_hour), 2) );
strcat( julian, aform((gmt->tm_min), 2) );

tnow /= 900;
/* current time in 1/4  hrs since 00:00:00 Jan 1, 1970 */
sample = (int) (tnow % (long) 100);
/* sample runs from 0 to 499 (cyclically) 1/4 hrly */

*sample_no = sample;
*qtr = quarters.quot;

return julian;
}

/************ WAIT1 waits for 1/2 second ************/

void wait1()
{
clock_t tnow, tnext;
```

```
tnow = clock();

do
     {
      tnext = clock();
     }
while ( (double)(tnext - tnow)/CLK_TCK <= 0.5 );
}


/************ AFORM formats a number in specified format ************/

char * aform(int i_var, int n_char)
{
char asc_var[4] = "000", temp[3];
int l_var;
if (i_var <= 0 )
     {
     asc_var[n_char] = '\0';
     return asc_var;
     }
else
     {
     l_var = (int) ( 1 + log10((double) i_var) );
     if ( ((n_char - l_var) < 4) && ((n_char - l_var) > -1) )
          {
          itoa(i_var, temp, 10);
          strcpy(asc_var + n_char - l_var, temp );
          }
     }
return asc_var;
}

/************ HARDERROR_HANDLER handles hardware errors *************/

void far harderror_handler(unsigned deverror, unsigned errcode, unsigned far *devhdr)
{
char dletter, num[5];
error_flag = 1;

if (strlen(message) > 40)
     {
     strcpy(message, "");
     }

if ( (deverror & 0x8000) == 0)
     {
     switch(deverror & 0xff)
          {
          case 0:
               strcat(message, "Drive A ");
               break;
          case 1:
               strcat(message, "Drive B ");
               break;
          case 2:
               strcat(message, "Drive C ");
               break;
          }

     strcat(message, " ERROR:-");
```

```
        itoa(errcode & 0xff, num, 10);
        switch(errcode & 0xff)
            {
            case 0:
                strcat(message, " Write Prot'd");
                break;
            case 2:
                strcat(message, " Not Ready");
                break;
            case 9:
                strcat(message, " No Paper");
                break;
            case 10:
                strcat(message, " Write Fault");
                break;
            case 12:
                strcat(message, " Gen Failure");
                break;
            default:
                strcat(message, " Code ");
                strcat(message, num);
                break;
            }
        switch(deverror & 0x0600)
            {
            case 0:
                strcat(message, "-MSDOS: ");
                break;
            case 0x0200:
                strcat(message, "-FAT: ");
                break;
            case 0x0400:
                strcat(message, "-Directory: ");
                break;
            case 0x0600:
                strcat(message, "-Data Area: ");
                break;
            }
    }
else
    {
    strcpy(message, "Non Disk I/O Error: ");
    if( ( *(devhdr + 4) & 0x8000 ) == 0)
        {
        strcat(message, "Bad Image of FAT: ");
        }
    else
        {
        strcat(message, "Character Device: ");
        }
    }
/* printf("%s\n", message); */

_hardretn(_HARDERR_IGNORE);
}


/************ CHECK_CACHE opens raw data copy file ************/
int check_cache(char *julian, char *raw_filename, FILE *f_cache)
{

strcpy(raw_filename, "f:F");
strcat(raw_filename, julian);
```

```
 strcat(raw_filename, ".raw");

 if ( (f_cache = fopen(raw_filename, "w+"))  == NULL )
      {
      #if DISPLAY == 1
           {
           _settextposition(0, 0);
           printf("*****COULD NOT OPEN RAW DATA FILE IN CACHE*****\n");
           }
      #endif
      return 0;
      }
 else
      {
      fclose(f_cache);
      return 1;
      }
 }

 void wipe_line( int row)
 {
 #if DISPLAY == 1
      {
      _settextposition(row,1);
      printf("                                        ");
      _settextposition(row,10);
      }
 #endif
 }



 /************ AFORM1 formats with specified %spec and range ************/
 char * aform1(double param, char * f_str, double max, double min)
 {
 char buffer[15];
 int ch;

 if ( (param > min) && (param < max) )
      {
      sprintf(buffer, f_str, param);
      }
 else
      {
      sprintf(buffer, f_str, 0);
      }

 return buffer;
 }


 /************  SEND_RS232 sends data to formatter via COM2 ************/

 void send_rs232(char *gilltime, double psd, double mean,
                          double north_mean, double east_mean, double vert_mean,
                          double c_mean, double buoy_heading, double a1, double b)
 {
 char rsout[75];
 char buffer[6];
 unsigned status, data;
 int ch;
 /* initialise com2 port, 2400 baud, 8bit data, no parity, 1 stop bit */
 data = (_COM_CHR8 | _COM_STOP1 | _COM_NOPARITY | _COM_2400);
```

```
_bios_serialcom(_COM_INIT, 1, data);

/* assemble message for formatter */
strcpy(rsout, "S00");
if ( (strcspn(gilltime, "T") == 0) && (strcspn(gilltime, "D") == 14)
     && (strcspn(gilltime, "/") == 21) )
     {
     strcpy(rsout + 3, gilltime + 25);
     strcpy(rsout + 5, gilltime + 19);
     strcpy(rsout + 7, gilltime + 22);
     strcpy(rsout + 9, gilltime + 5);
     strcpy(rsout + 11, gilltime + 8);
     strcpy(rsout + 13, gilltime + 11);

     strcpy(rsout + 15, "00");

     strcpy(rsout + 17, aform1(psd, "%+05.2f", 10., -10.));

     strcpy(rsout + 22, aform1(mean, "%05.2f", 100., 0.));

     strcpy(rsout + 27, aform1(north_mean, "%+06.2f", 100., -100.));

     strcpy(rsout + 33, aform1(east_mean, "%+06.2f", 100., -100.));

     strcpy(rsout + 39, aform1(vert_mean, "%+06.2f", 100., -100.));

     strcpy(rsout + 45, aform1(c_mean, "%06.2f", 1000., 0.));

     strcpy(rsout + 51, aform1(buoy_heading, "%03.0f", 360., 0.));

     strcpy(rsout + 54, aform1(a1, "%+05.2f", 10., -10.));

     strcpy(rsout + 59, aform1(b, "%+10.2E", 1., -1.));
     /* must be capital E for messagecheck to accept it in newform */

     strcat(rsout + 69, "T");
     }
else
     {
     strcat(rsout, "90010100000000+0.0000.00+00.00+00.00T");
     /* strcat(rsout, "90010100000000+0.0000.00+00.00+00.00+00.00000.00000\
                         +0.00+0.00E+000T"); */
     }

for (ch = 0; ch < strlen(rsout); ch++)
     {
     do
          {
          status = 0x2000 & _bios_serialcom(_COM_STATUS, 1, 0);
          }
     while (status != 0x2000);

     status = _bios_serialcom(_COM_SEND, 1, rsout[ch]);
     if ((status & 0x8000) == 0x8000)
          {
          #if DISPLAY == 1
               {
               printf("RS232 COM2 timed out\n");
               }
          #endif

          break;
          }
```

```
    }

}

/************ WAIT2 waits for 2 seconds ************/

void wait2(void)
    {
    clock_t tnow, tnext;

    tnow = clock();
    do
        {
        tnext = clock();
        }
    while ( (tnext - tnow) / CLK_TCK < 2);
    }

/******************** END OF FUNCTION DEFINITIONS ********************/
```

Appendix D.2 Source Code RTCN.C

```
/**********************RTCN.C**********************\
        Program to read Real-Time-Clock time/date
                (call with argv[1] = 1, e.g. rtcn 1)
        or to update the DOS time with the RTC time
                (call with argv[1] = 2, e.g. rtcn 2)
        (useful to keep DOS clock drift low)*
        or to update the RTC time with the DOS time
                (call with argv[1] = 3, e.g. rtcn 3)
        or to set in a fixed DOS time of 1000 ticks
                (call with argv[1] = 4, e.g. rtcn 4)
        (useful for DSP processor after CLOCKSET)

        NB the DOS TIME command reads the DOS time,
        i.e. the system time, but if you enter a
        new time, instead of <return>, it sets both
        RTC and DOS time to the entered time.

        CHC     12th August 1993
\************************************************/

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

#define BIOS_INT 0x1a
#define GET_RTC_TIME 0x02
#define SET_RTC_TIME 0x03
#define GET_RTC_DATE 0x04
#define SET_RTC_DATE 0x05

#define GET_DOS_CLOCK 0x00
#define SET_DOS_CLOCK 0x01

#define DOS_INT 0x21
#define GET_DOS_DATE 0x2a
#define SET_DOS_DATE 0x2b
```

```
#define GET_DOS_TIME 0x2c
#define SET_DOS_TIME 0x2d

main(int argc, char *argv[ ])
{
union REGS xr;
struct SREGS sr;
int hrs, mins, secs, century, year, month, day;
long clockcount = 1000L;

if (argc != 2)
    {
    printf("Argument missing: rtc <n>\n(n = 1 for read RTC time/date)\n");
    printf("(n = 2 for updating DOS time/date with RTC time/date)\n");
    printf("(n = 3 for updating RTC time/date with DOS time/date)\n");
    printf("(n = 4 for setting DOS time/date to 00:00 01/01/1993)\n");
    exit(0);
    }
switch( *argv[1] )
    {
    case '1':                       /* Get and Display RTC time and date */
        {
        xr.h.ah = GET_RTC_TIME;
        int86x(BIOS_INT, &xr, &xr, &sr);
        hrs = 10 * ((xr.h.ch & 0xf0) / 16) + (xr.h.ch & 0x0f);
        mins = 10 * ((xr.h.cl & 0xf0) / 16) + (xr.h.cl & 0x0f);
        secs = 10 * ((xr.h.dh & 0xf0) / 16) + (xr.h.dh & 0x0f);

        printf("Time %.2d:%.2d:%.2d, Date ", hrs, mins, secs);

        xr.h.ah = GET_RTC_DATE;
        int86x(BIOS_INT, &xr, &xr, &sr);
        century = 10 * ((xr.h.ch & 0xf0) / 16) + (xr.h.ch & 0x0f);
        year = 10 * ((xr.h.cl & 0xf0) / 16) + (xr.h.cl & 0x0f);
        month = 10 * ((xr.h.dh & 0xf0) / 16) + (xr.h.dh & 0x0f);
        day = 10 * ((xr.h.dl & 0xf0) / 16) + (xr.h.dl & 0x0f);
        printf("%.2d/%.2d/%.2d%.2d\n", day, month, century, year);

        break;
        }

    case '2':                       /* Update System Clock with RTC time and date */
        {
        do
            {
            xr.h.ah = GET_RTC_TIME;
            int86x(BIOS_INT, &xr, &xr, &sr);
            }
        while ((10 * ((xr.h.dh & 0xf0) / 16) + (xr.h.dh & 0x0f)) == 59);
        hrs = 10 * ((xr.h.ch & 0xf0) / 16) + (xr.h.ch & 0x0f);
        mins = 10 * ((xr.h.cl & 0xf0) / 16) + (xr.h.cl & 0x0f);
        secs = 10 * ((xr.h.dh & 0xf0) / 16) + (xr.h.dh & 0x0f);

        xr.h.ah = SET_DOS_TIME;
        xr.h.ch = hrs;
        xr.h.cl = mins;
        xr.h.dh = secs + 1;
        xr.h.dl = 0;
        int86x(DOS_INT, &xr, &xr, &sr);
        if (xr.h.al != 0)
            {
            printf("%.2d %.2d %.2d\n", hrs, mins, secs);
            }
```

```
            xr.h.ah = GET_RTC_DATE;
            int86x(BIOS_INT, &xr, &xr, &sr);
            century = 10 * ((xr.h.ch & 0xf0) / 16) + (xr.h.ch & 0x0f);
            year = 10 * ((xr.h.cl & 0xf0) / 16) + (xr.h.cl & 0x0f);
            month = 10 * ((xr.h.dh & 0xf0) / 16) + (xr.h.dh & 0x0f);
            day = 10 * ((xr.h.dl & 0xf0) / 16) + (xr.h.dl & 0x0f);

            xr.h.ah = SET_DOS_DATE;
            year += 100 * century;
            xr.h.ch = year / 256;
            xr.h.cl = year - 256 * (year / 256);
            xr.h.dh = month;
            xr.h.dl = day;
            int86x(DOS_INT, &xr, &xr, &sr);

            break;
            }

        case '3':                       /* Update RTC with System Clock time and date */
            {
            xr.h.ah = GET_DOS_TIME;
            int86x(DOS_INT, &xr, &xr, &sr);
            hrs = xr.h.ch;
            mins = xr.h.cl;
            secs = xr.h.dh;

            xr.h.ch = 16 * (hrs / 10) + hrs - 10 * (hrs / 10);
            xr.h.cl = 16 * (mins / 10) + mins - 10 * (mins / 10);
            xr.h.dh = 16 * (secs / 10) + secs - 10 * (secs / 10);
            xr.h.ah = SET_RTC_TIME;
            int86x(BIOS_INT, &xr, &xr, &sr);

            xr.h.ah = GET_DOS_DATE;
            int86x(DOS_INT, &xr, &xr, &sr);

            year = 256 * xr.h.ch + xr.h.cl;
            century = year / 100;
            year -= (100 * century);
            month = xr.h.dh;
            day = xr.h.dl;

            xr.h.ah = SET_RTC_DATE;
            xr.h.ch = 16 * (century / 10) + century - 10 * (century / 10);
            xr.h.cl = 16 * (year / 10) + year - 10 * (year / 10);
            xr.h.dh = 16 * (month / 10) + month - 10 * (month / 10);
            xr.h.dl = 16 * (day / 10) + day - 10 * (day / 10);
            int86x(BIOS_INT, &xr, &xr, &sr);

            break;
            }

        case '4':                       /* set DOS time/date to 00:00 01/01/1993 for test purposes */
            {
            _bios_timeofday(_TIME_SETCLOCK, &clockcount);
            _bios_timeofday(_TIME_GETCLOCK, &clockcount);
            printf("Ticks %d\n", clockcount);
            break;
            }
        }
return 0;
}
```

Appendix E Hardware

General Assembly

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

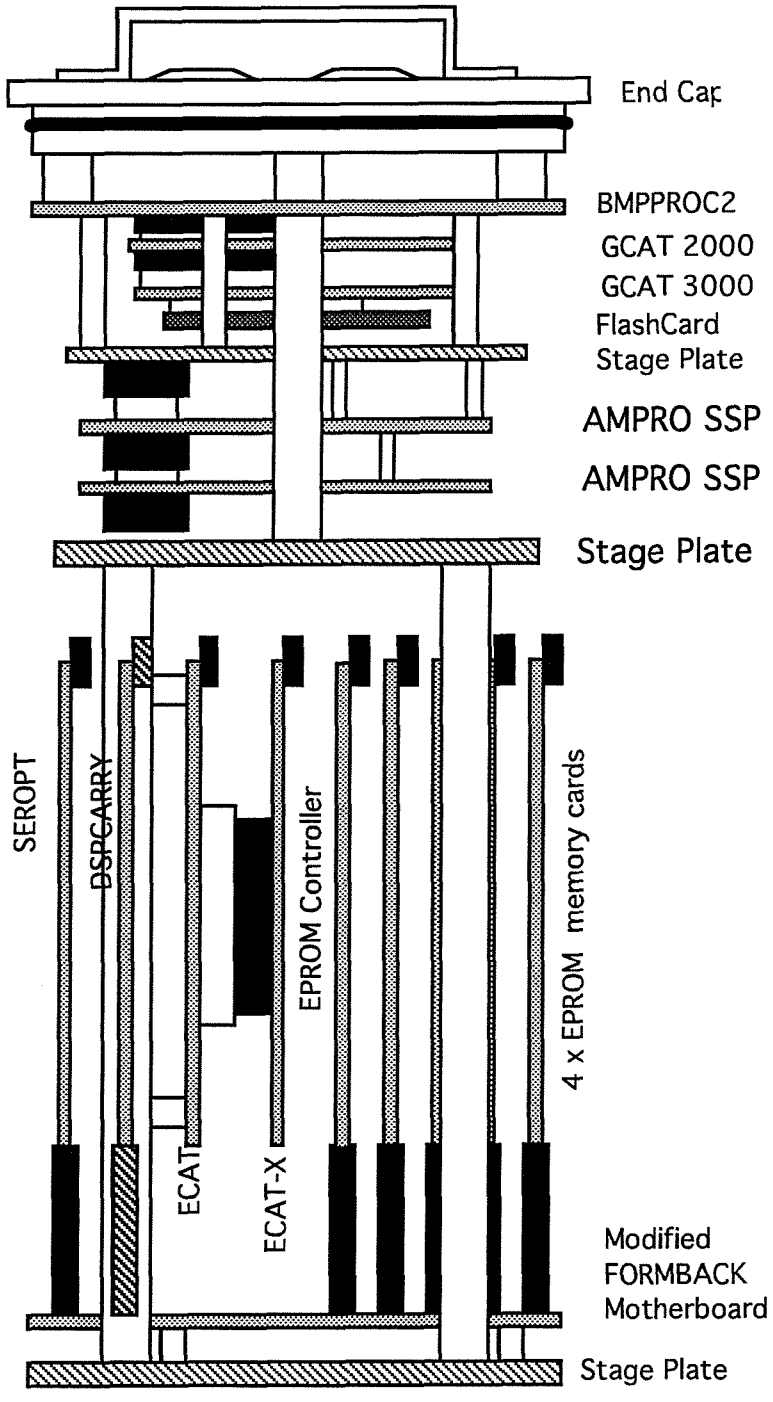


Figure E.1 Schematic of Formatter/Sonic Processor

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 Formatter Handbook)

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

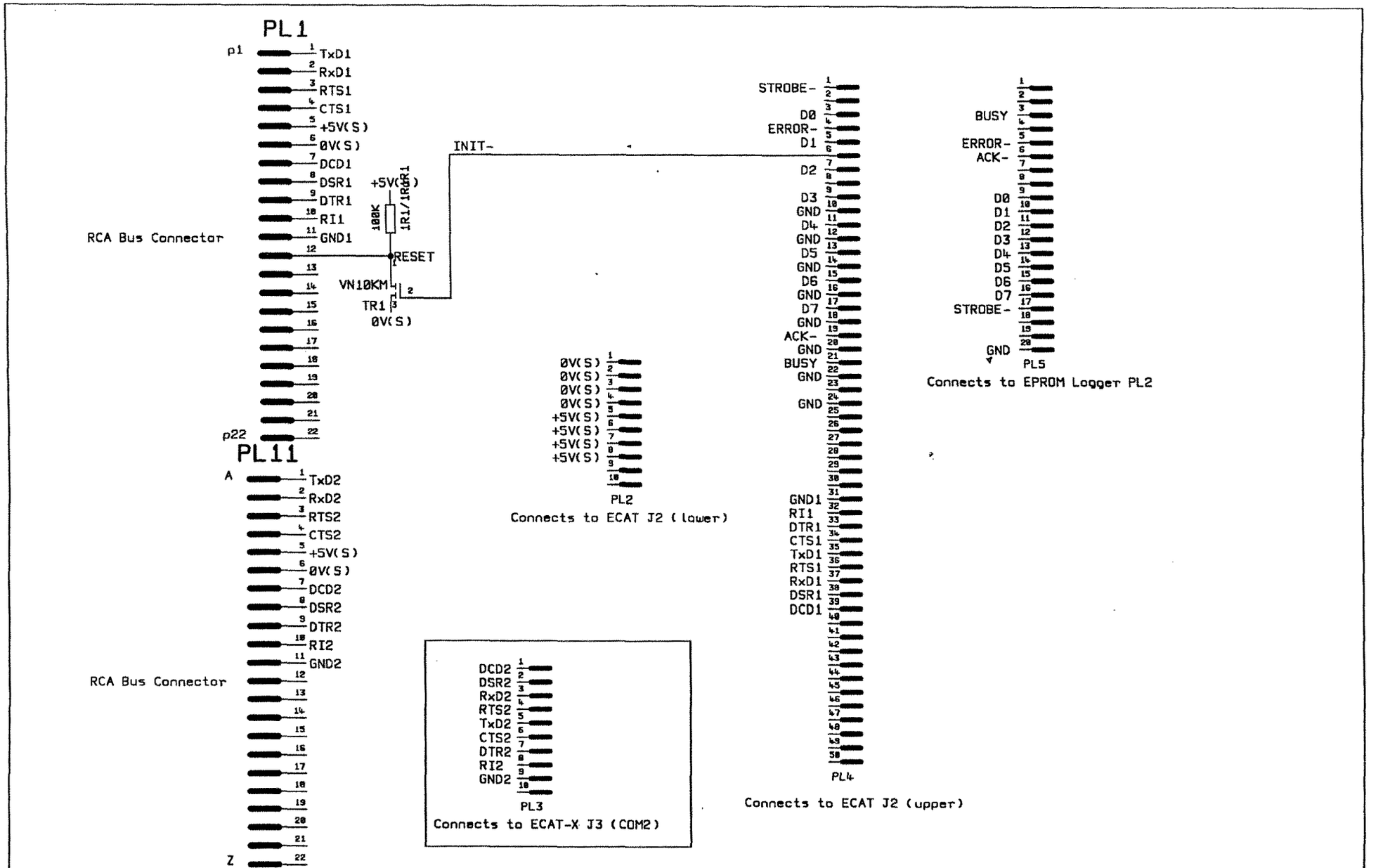
Appendix F DSPCARRY - Carrier Board for ECAT/ECAT-X

Parts List

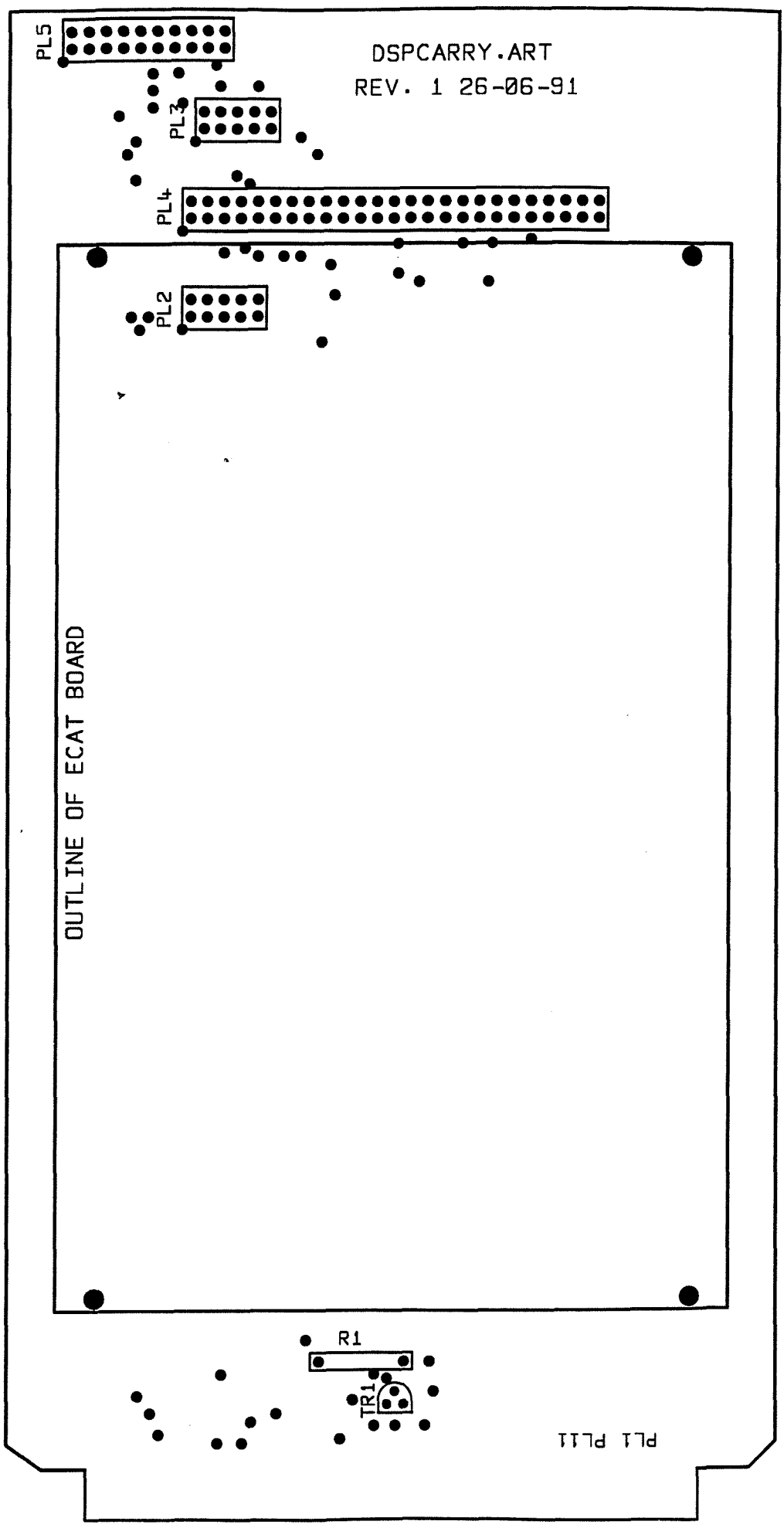
ALPHABETICALLY ORDERED LIST OF PARTS WITH SILK REFERENCES AND DESCRIPTIONS

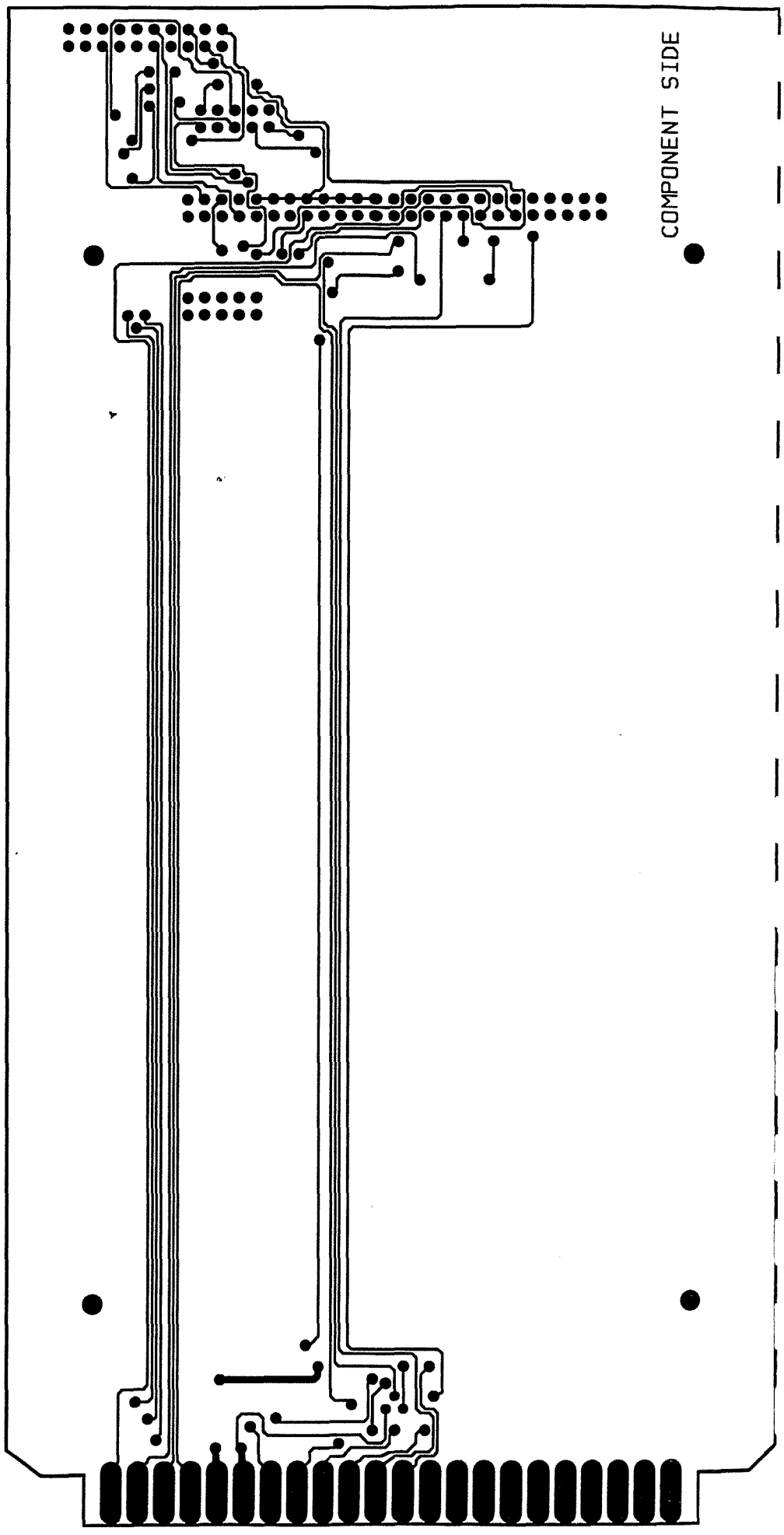
---

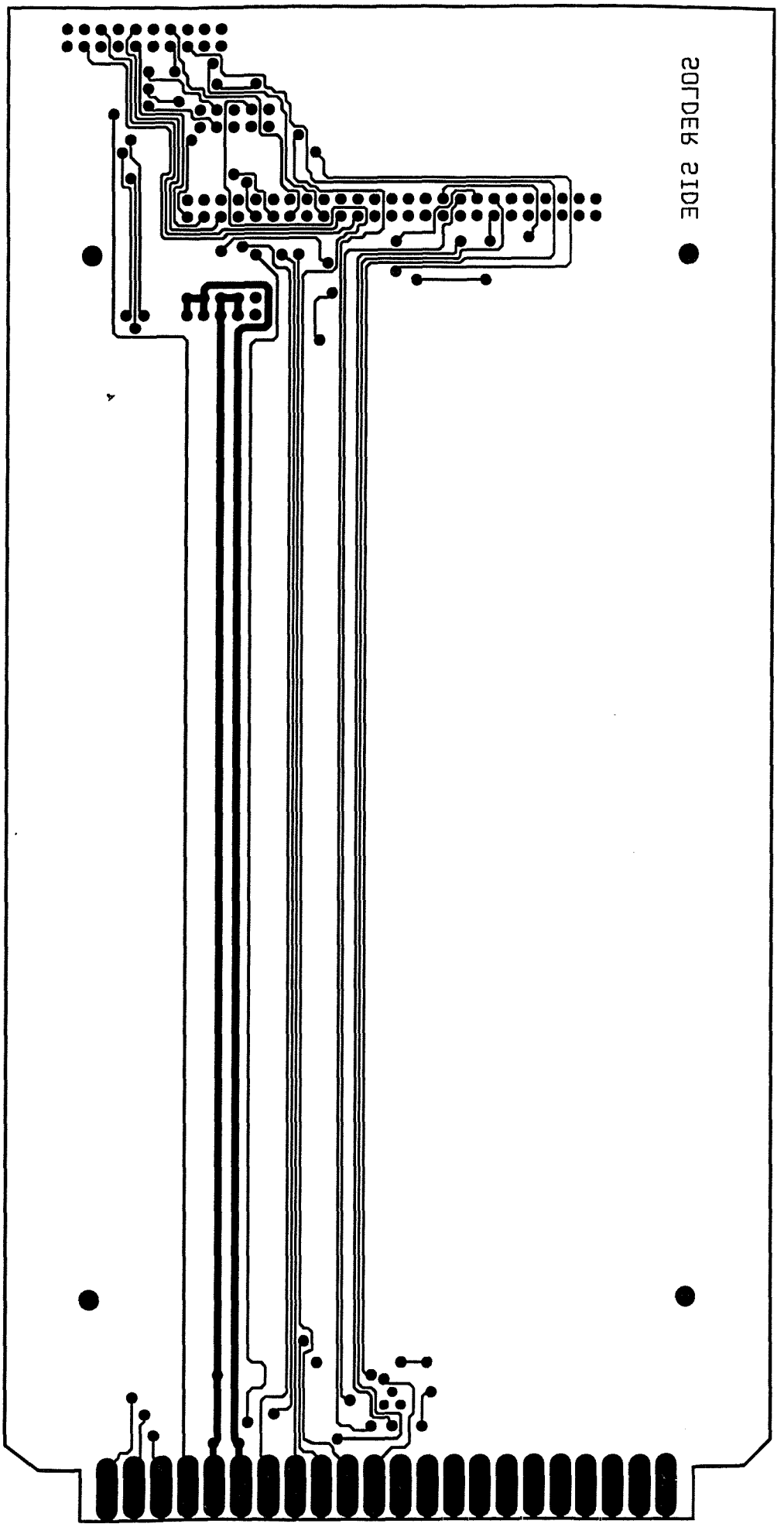
1 off PCB	DSPCARRY	Motherboard manufactured to IOSDL artwork DSPCARRY.ART
2 off IDC10	PL2, PL3	Mini DIP PCB Solder Transition Farnell 145-065
related parts *		
* ~100mm length		10 way IDC cable (PL3 to SK3) Farnell 171-10 1 foot
* 1 off	SK3	Female 10 way Socket Bump/Clip Pol'n Farnell .152-718
* 1 off	SK2	Female 100 way Socket to mate with ECAT J2
	Supplied complete with 2 off 50 way IDC cables by DSP Design Ltd.	
1 off IDC20	PL5	Right angle Male PCB mounting IDC header Farnell .152-021
related parts *		
* 1 off	SK5	Female 20 way Socket Bump/Clip Pol'n Farnell .152-721
* ~100mm length		20 way IDC cable (SK5 to EPROM Controller) Farnell 171-20 1 foot
1 off IDC50	PL4	Mini DIP PCB Solder Transition (connects to SK2 upper, above) Farnell 145-071
1 off RCACON1	PL11	gold plated edge connections
1 off RCACON2	PL1	
1 off RMFW25#100K	R1	Resistor 1/2 W metal film Farnell MFR4 100K
1 off VN10KM	TR1	Low power MOSFET Farnell VN10KM



INSTITUTE OF OCEANOGRAPHIC SCIENCES, DEACON LABORATORY, BROOK ROAD, WORMLEY, GODALMING, SURREY, GU8 5UB, ENGLAND.	FILENAME: DSPCARRY.DGM	CARRIER BOARD FOR DSP SBC TO FIT RCA CARD FRAME	SHEET: 1 OF 1 ISSUE: REV. 1 DATE: 18-06-91


PL5
PL3
PL4
PL2
DSPCARRY.ART
REV. 1 26-06-91
OUTLINE OF ECAT BOARD
R1
TR1
PLL PLL1


COMPONENT SIDE


SOLDER SIDE

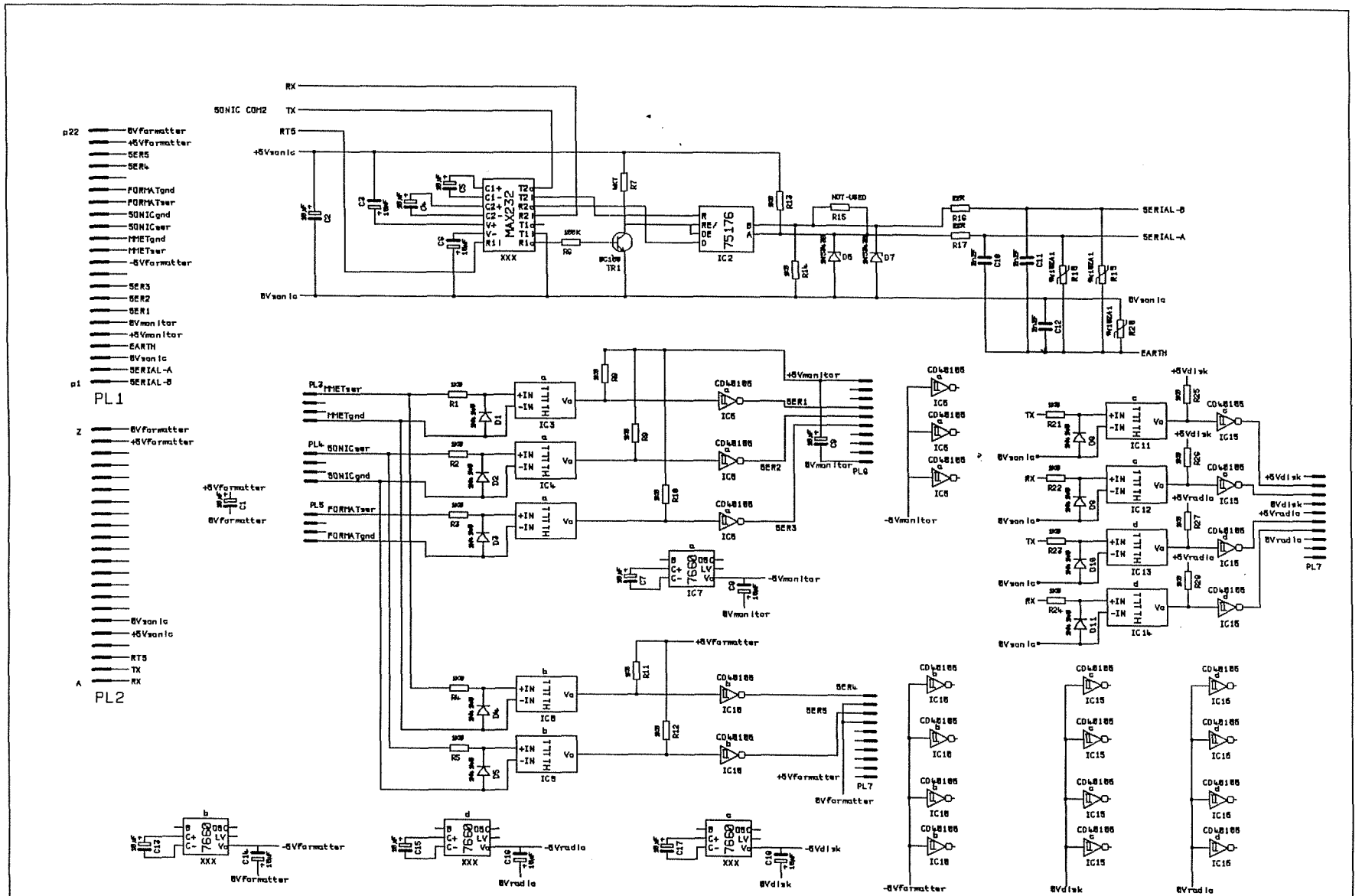
Appendix G SEROPT - Anemometer RS232/422 interface and Opto-isolators

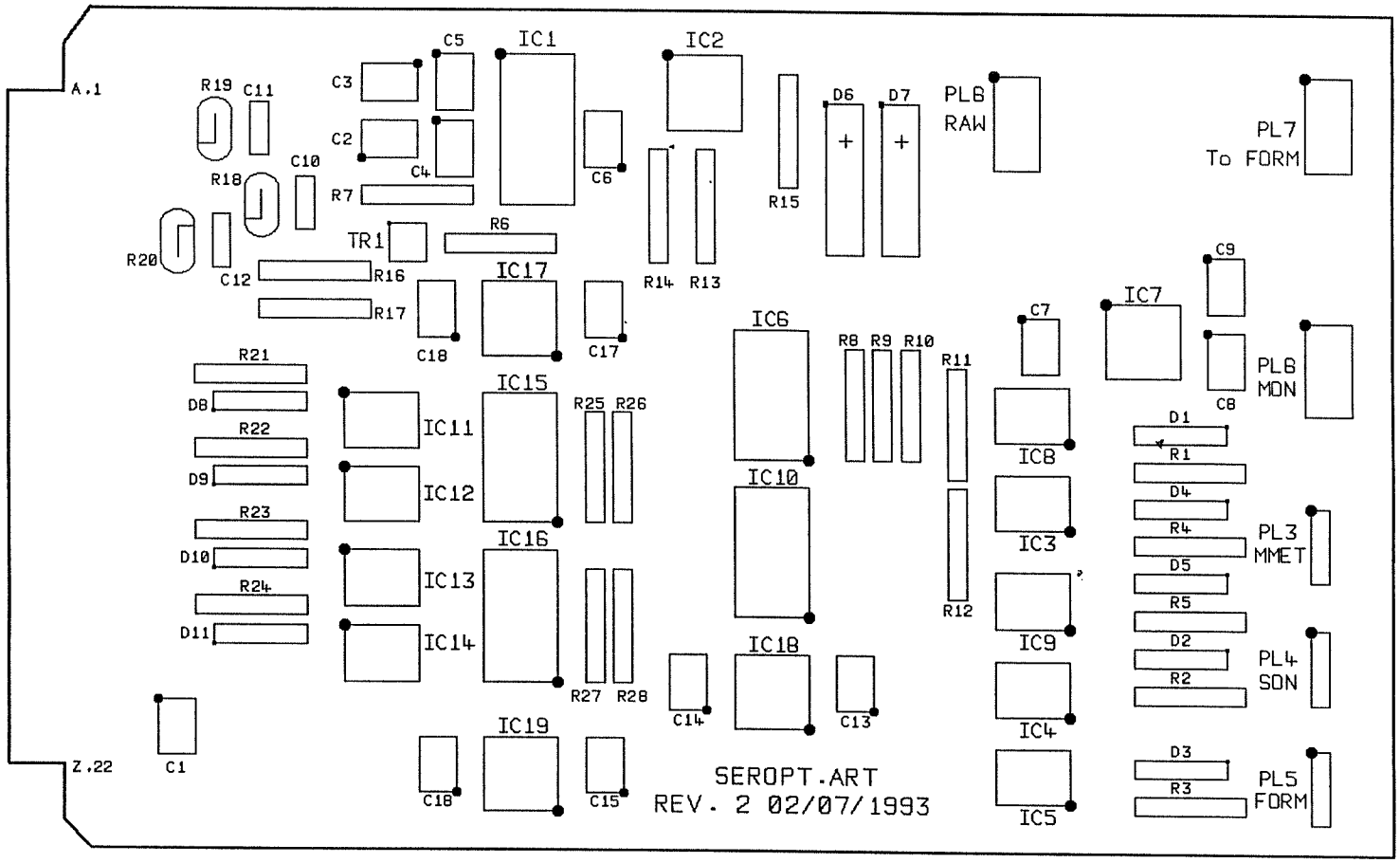
Parts List

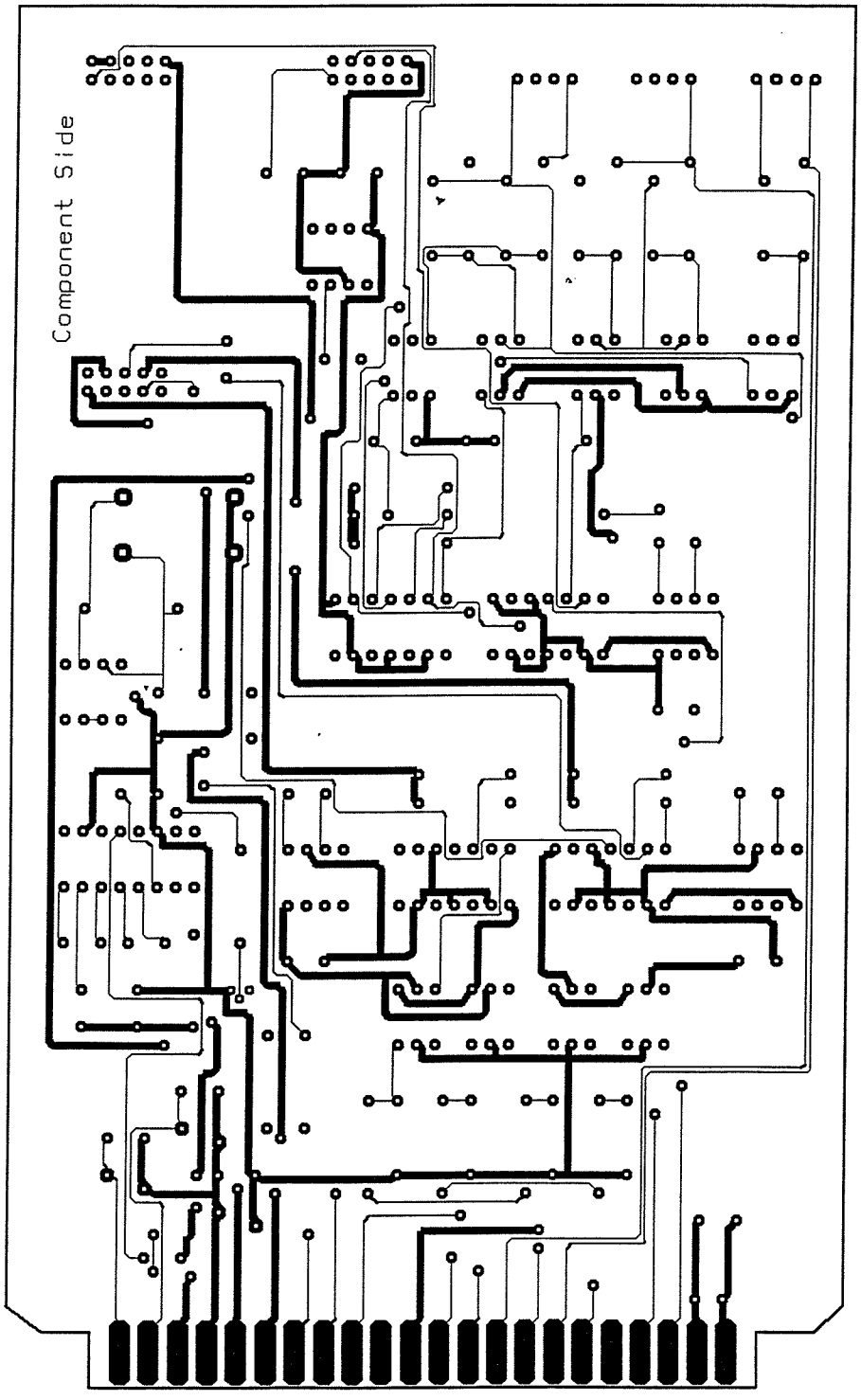
ALPHABETICALLY ORDERED LIST OF PARTS WITH SILK REFERENCES AND DESCRIPTIONS

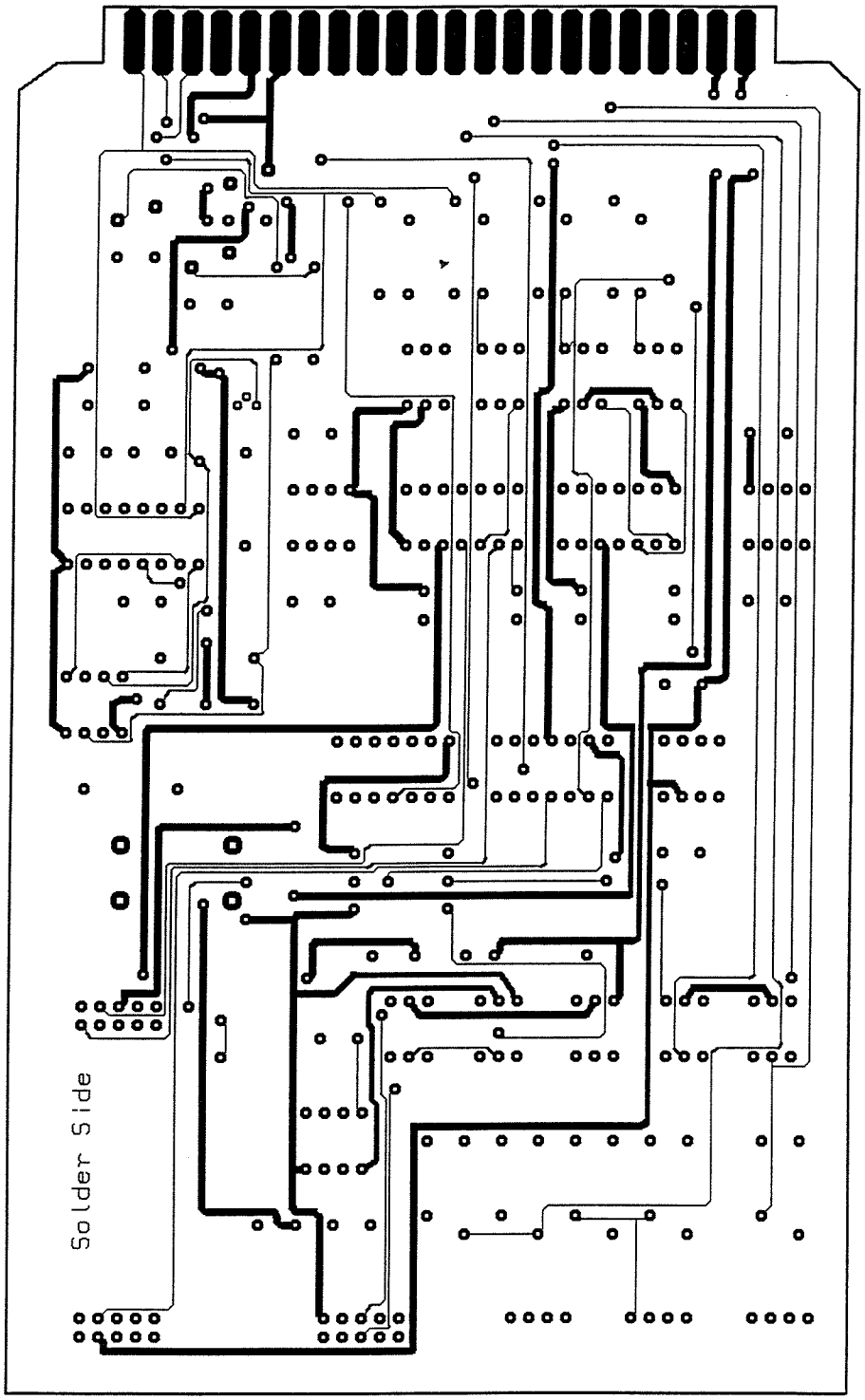
9 off 1N4148	D1, D10, D11, D2, D3, D4, D5, D8, D9	Small Signal Diode Farnell 1N4148
2 off 1N53#1N5343B	D6, D7	Zener Diode Farnell 1N5343B
1 off 75176	IC2	Line Driver/Receiver Farnell SN75LBC176P.
4 off 7660	IC7, 17, 18, 19	CMOS Voltage Convertor Farnell ICL7660CPA-MAX
4 off CD40106	IC6, 10, 15, 16	Schmitt Buffer Farnell CD40106BCN
3 off CFKC2#2N2F	C10, C11, C12	Capacitor Polycarbonate Farnell 147-667
15 off CTANT#10µF	C1, C13, C14, C15, C16, C17, C18, C2, C3, C4, C5, C6, C7, C8, C9	Capacitor Tantalum Farnell 100-906
9 off H11L1	IC3, IC4, IC5, IC8, IC9, IC11, IC12, IC13, IC14	Optocoupler, Schmitt Trigger output Farnell H11L1
3 off IDC10	PL6, PL7, PL8	Right angle Male PCB mounting IDC header Farnell .152-018
related parts *		
* 3 off	SK6, SK7, SK8	Female 10 way Socket Bump/Clip Pol'n Farnell .152-718
1 off MAX232	IC1	CMOS Dual Transmitter/Receiver Farnell MAX232
3 off MOLEX4	PL3, PL4, PL5	90° Square Pin Header with friction lock

		Farnell 146-693
related parts *		
* 3 off	SK3, SK4, SK5	Crimp Terminal Housing (Polarised)
		Farnell 143-094 + inserts 143-116
1 off NPN-BC109	TR1	Low Power Bipolar Transistor
		Farnell BC109-SGS
1 off RCACON1	PL11	(gold plated edge connections)
1 off RCACON2	PL1	(gold plated edge connections)
1 off RMFW25#100K	R6	Resistor 1/2 W metal film
		Farnell MFR4 100K
11 off RMFW25#1K0	R10, R11, R12, R13,	Resistor 1/2 W metal film
	R14, R25, R26, R27,	Farnell MFR4 1K0
	R28, R8, R9	
9 off RMFW25#1K8	R1, R2, R21, R22,	Resistor 1/2 W metal film
	R23, R24, R3, R4, R5	Farnell MFR4 1K8
2 off RMFW25#22R	R16, R17	Resistor 1/2 W metal film
		Farnell MFR4 22R
1 off RMFW25#4K7	R7	Resistor 1/2 W metal film
		Farnell MFR4 4K7
1 off RMFW25#NOT-U	R15	not used
3 off V18ZAA1	R18, R19, R20	Metal Oxide Varistor
		Farnell V18ZA1


PL1
PL2
MAX232
75176
CD4066
7660
SONIC COM2
SERIAL-B
SERIAL-A
EARTH
-8Vformatter
+8Vformatter
-8Vmonitor
+8Vmonitor
-8Vradio
+8Vradio
-8Vdisk
+8Vdisk
-8Vsonic
+8Vsonic
INSTITUTE OF OCEANOGRAPHIC SCIENCES, DEACON LABORATORY,
BROOK ROAD, WORMLEY, GODALMING, SURREY GU8 5UB, ENGLAND
FILENAME
SEROPT.DGM
SONIC BUOY - OPTOISOLATORS AND SONIC RS232/422 INTERFACE
SHEET
1 OF 1
ISSUE
REV 2
DATE
25-06-93


A.1
Z.22
C1
R19
C11
C3
C5
IC1
IC2
C2
C4
C6
R18
C10
R7
R15
D6
D7
PL8
RAW
PL7
To FORM
R20
C12
TR1
R6
R16
R14
R13
IC17
R17
C18
C17
C9
IC7
IC6
R8
R9
R10
R11
C7
C8
PL8
MON
R21
D8
IC15
R25
R26
IC11
IC8
D1
R22
D9
IC12
IC10
IC3
R1
D4
R23
D10
IC16
R4
PL3
MMET
IC13
IC9
D5
R24
D11
IC14
R5
R12
IC18
D2
PL4
SDN
R27
R28
C14
C13
IC4
R2
IC19
D3
PL5
FORM
C18
C15
IC5
R3
SEROPT.ART
REV. 2 02/07/1993


Component Side


Solder Side