The integrity of serial data highway systems

Cowan, D.

The Integrity of Serial Data Highway Systems

by

D.Cowan, B.Sc.

A thesis submitted for the degree of Doctor of Philosophy
in the University of Durham, 1983.

The Integrity of Serial Data Highway Systems

D.Cowan

Abstract

The Admiralty Surface Weapons Establishment (ASWE) have developed a Local Area Network System. This thesis describes the development of a replacement for this LAN system, based around 16 bit microprocessor hosts, as opposed to the minicomputers currently used. This change gave a substantial reduction in size, and allowed the new system to be installed on a ship and tested under operational conditions. Analysis of the data collected during the tests gave performance information on the ASWE system. The performance of this LAN is compared to that of other leading types of LAN. The design of a portable network controller/ monitor unit is presented, which may be manufactured as a standard controller for the ASWE Serial Highway.

Acknowledgements

I wish to express my thanks primarily to Dr. C.T. Spracklen for his assistance during the course of this project. In addition, the advice of Mr. D.J. Dwyer and Dr. D.R. Isaac was much appreciated. Thanks are also due to the departmental technicians who provided valuable technical help on many occasions. The staff of XCC Division, ASWE, assisted in many ways during the course of my research, and they continually tolerated the disruption to their normal work which Dr. Spracklen and myself seemed to bring to them.

I am particularly grateful to the Ministry of Defence who funded this project.

Contents

Glossary of Terms

ACIA Asynchronous Communications Interface Adaptor

ALU Arithmetic Logic Unit

ASCII American Standard Code for Information Interchange

ASH ASWE Serial Highway

ASWE Admiralty Surface Weapons Establishment

CMOS Complimentary Metal-on-Silicon

CPU Central Processor Unit

CSMA Carrier Sense Multiple Access

DMA Direct Memory Access

DS DO Stack

EPROM Erasable Programable Read-only-Memory

FEP Front End Processor

FIFO First In First Out

I/O Input/ Output

LAN Local Area Network

LCD Liquid Crystal Display

LED Light Emitting Diode

MS Machine Stack

OS Operand Stack

PIA Parallel Interface Adaptor

PROM Programable Read-only-Memory

RAM Random Access Memory

VDU Visual Display Unit

Chapter 1

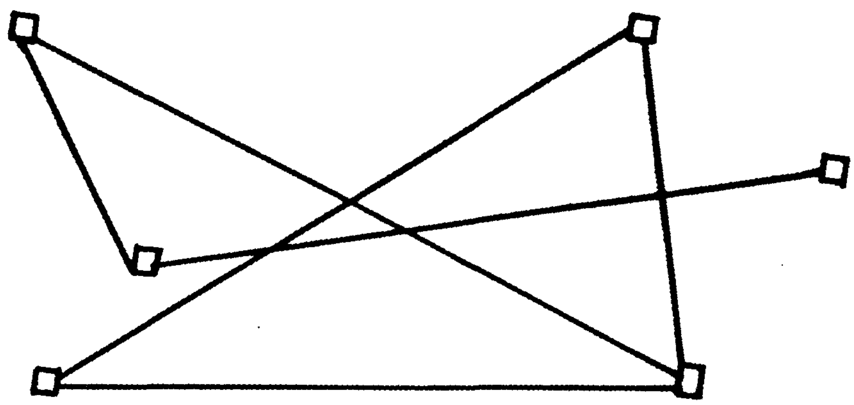
Introduction

1.1 Local Area Networks

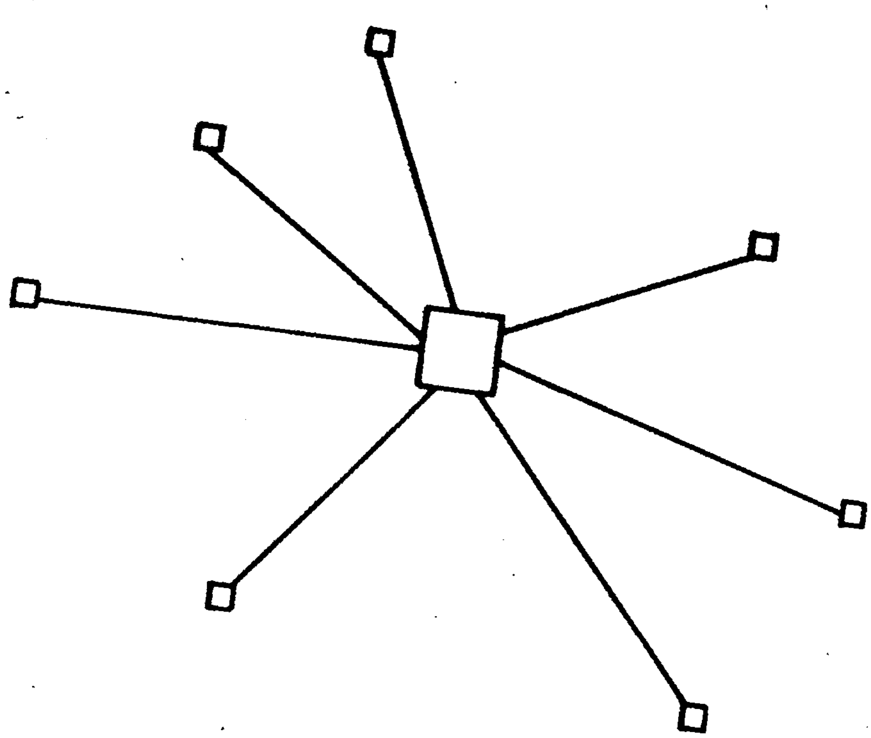
In recent years the trend towards mainframe computers of ever increasing complexity has been overtaken by the use of distributed computer systems, in an attempt to provide greater speed and flexibility of computing power. This has been aided by the greatly reduced costs of computer hardware, and by the ease of application of modern block structured programming languages to multiprocessor systems. These systems normally fall into one of two categories, loosely coupled and tightly coupled systems. In the former, communications between the elements of the system take place at a very much higher rate than in the latter. Tightly coupled systems can have a communication rate of up to 200Mbits/ sec, whilst most loosely coupled systems have a maximum transmission rate of approximately 20Mbits/ sec. The difference in transmission speeds is due to the differing demands placed on the communication system by the elements in the network. Normally, the elements in a tightly coupled system are interdependant and would be unable to function satisfactorily if one element was malfunctioning. Array processor systems and multi-ALU systems fall into this category. Loosely coupled systems normally consist of units which are able to function satisfactorily by themselves, and communication between the elements is normally via data messages rather than machine level instructions as in the tightly coupled systems. The decrease in the transmission rate allows different transmission media to be used, and many loosely coupled systems use serial transmission lines. There is a large range of possible interconnection systems for these two types of distributed computing systems. However, they fall roughly into three categories; point to point, interconnecting bus, and network. In addition, combinations of the three types also occur.

In the past, point to point systems have been used to a great extent because they were the easiest and least expensive to implement. In a point to point system, a dedicated communication path exists between every element in the system. This necessitates a large amount of wiring between elements, but has the great advantage that the receiving unit has an inherent physical address; no additional software is required to generate the address or, in the receiver, to decode the address. However, greater importance is now placed upon the ability to reconfigure a communication system to incorporate new elements, and in a point to point communications system this necessitates expensive and complex rewiring. Thus point to point systems have largely been replaced by some form of shared communications system.

A communication network is a collection of shared communications paths and devices interconnected so at least one pair of devices has more than two simultaneous path possibilities. The most common of the network systems have computers as devices and telephone landlines as communications paths. Examples of these are ARPANET [1,2] and OCTOPUS [3]. Networks are characterised by their topology, protocols, communications disciplines and geographical extent. Networks which communicate over longer distances than 1km, such as ARPANET, are known as long-haul networks, whilst those which work over shorter distances, such as the Xerox FIBERNET [4], are known as short-haul networks. The most common topologies used are the mesh and star, shown in Figure 1.1a. The protocols implemented depend upon both the topology and communication discipline used, however they can usually be subdivided into transport protocols, routing and flow protocols and user level protocols. The communications disciplines used are circuit switching, message switching and packet switching. Our telephone



Mesh



Star

Figure 1.1a Common Networks

system employs circuit switching; a complete circuit must be established between caller and the listening station otherwise the caller hears an engaged signal. Packet switching is employed by ARPANET and most short-haul networks. Messages are segmented into fixed length packets which are only reconstructed at the destination node. Intermediate nodes retransmit packets as received, but certain systems may perform error detection and correction. The distinction between packet switching and message switching is less obvious in certain configurations of local networks.

The pure star topology employed by FIBERNET type systems does not strictly fit the category of a network because of the lack of simultaneous paths.

A data bus is a shared communication path joining many devices with only one path between any two devices. Examples of such systems include MIL-STD 1553B [5], the Cambridge Data Ring [6], and ETHERNET [7,8]. One of the advantages of a data bus system over a point to point system is the ease of reconfiguration to support additional devices. However the data bus system has the disadvantage of the requirement for software addresses for every device, and the complicated protocols and decoding necessary to support this type of addressing.

The term 'Local Area Network' (LAN) is now used to describe short haul networks and data bus systems. The systems upon which most work is currently being undertaken are data bus systems. The most used topologies are ring, redundant ring, linear and redundant linear. The addition of redundancy gives protection against the failure of the transmission media.

Regardless of its topology, a data bus can be active or passive; an active bus is one with signal regeneration at each node, whilst a passive one has no regeneration in the system.

1.1:1 Ring Topology

The topology of a typical ring network is detailed in Figure 1.1:1a. Each unit acts as a repeater on the ring and the LAN normally uses some form of token passing message handling system. A ring network utilises an active data bus. In such a system, single or multiple tokens are continually circulating round the ring. If a unit wishes to transmit a message it waits until it receives this token. It then transmits its message and appends the token to the end of the message. The unit to which the message is addressed takes a copy of the message and also regenerates it and the token in the same manner as the intermediate units. The removal of the message from the ring is left to the unit which originated the message. Although the ring type network theoretically has the advantage of completely decentralised control, in practice there must be a master station which inserts the tokens onto the ring and monitors its activity to ensure that there is always a token present.

The 'Cambridge Ring' LAN is a variation of the ring network. It uses token passing in the form of a 'Message Slot' format. The master unit initially sets up a message structure on the ring consisting of a number of message slots each preceded by a header to indicate whether the slot is empty or full. A unit wishing to transmit a message merely waits until an empty slot arrives, and it then fills the slot and alters the header accordingly. Again, removal of the message is left to the transmitting unit. Unfortunately, this means that the master unit in the ring must set up the message slots, and must then maintain them against the possibility of corruption by noise. Thus the

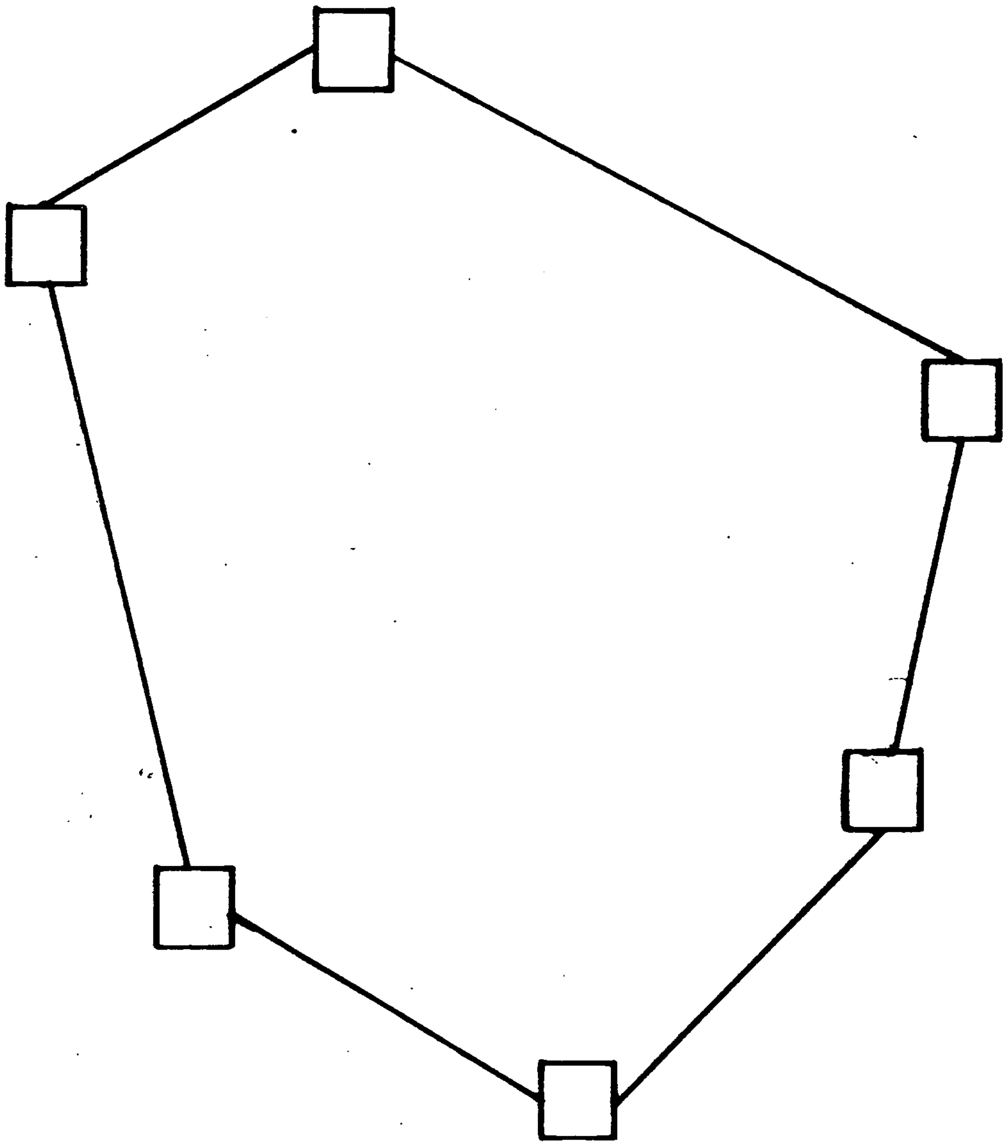


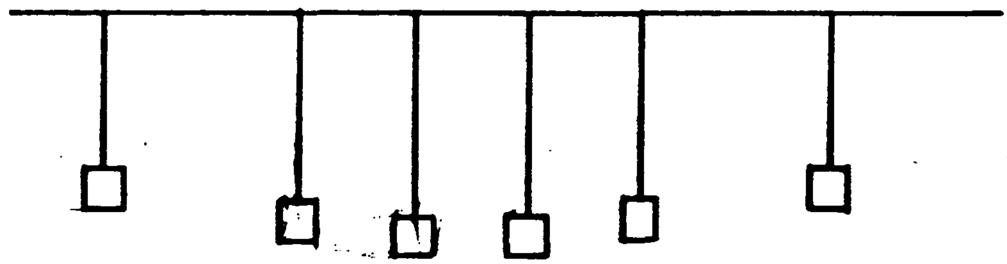
Figure 1.1:1a Ring Topology

advantages of decentralised control presented by the ring concept have been lost. In addition, since each unit on the ring is an active repeater, the failure of any one of these units can cause one section of the LAN to be isolated. This can be overcome by the use of more than one cable, (redundant ring topology) and by transmitting messages around each of the cables in different directions.

1.1:2 Linear Topology

The final class of LAN architecure is the one which is currently the most used. The topology of a linear bus LAN can be seen in Figure 1.1:2a. There are several message handling systems for such an architecture, but they fall into one of two classes; asynchronous and synchronous.

The ALOHA [9,10] system is an example of an early asynchronous LAN. If a unit wished to transmit a message it transmitted it immediately. It then waited for an acknowledgement of reception from the unit to which it had addressed the message. If the acknowledgement was not received, it assumed that the message had not been received, possibly due to the simultaneous transmission of a message by another unit. It then retransmitted the message after a random time interval, to ensure that the same clash did not occur again. However, as the load increased, the number of clashes also increased, and this meant that the maximum channel utilisation of a pure Aloha system was approximately 18 percent [8]. The addition of rudimentary coordination between units increased this utilisation. A series of synchronisation pulses were transmitted on the bus and units were only allowed to transmit messages immediately after the pulse. This increased the possible channel utilisation to 36 percent. An extension to the pure Aloha technique is the Carrier Sense Multiple Access with Collision



Passive
Linear Bus

Active
Linear Bus

Figure 1.1:2a Linear Bus Topology

Detect (CSMA/CD) system used in such LANs as Ethernet. In this system, all units monitor the activity on the bus before transmitting their messages. If the channel is free, a unit may transmit its message. If two (or more) units start transmission at the same time they can detect this clash by monitoring the activity on the channel. They immediately cease transmitting their message. The units then retransmit after a random time, to protect against repeated clashes. This system is efficient under conditions of low loading, however as the loading increases so too does the message delivery time. Channel utilisation in the Ethernet system can reach 98 percent [8].

Synchronous bus LANs have a controller which supervises the transmission of messages by the units on the bus. However, the throughput of the master unit does not determine the throughput of the network, as in the star LAN, because the master does not perform a 'receive and repeat' function. There are several type of sychronous bus control, but two of the more common types are :- round robin and polled systems. In the round robin system, each unit has a list of the order in which the units are to transmit held in its memory. The bus master transmits a message which says 'Next please' and the units consult their list to determine whether or not they are the next one on the list. The relevant unit then transmits a message if it wishes. Then the cycle is repeated. Synchronisation between the units must be achieved initially, to ensure that they are all at the same positions in their lists.

The second type of synchronous bus control commonly used is a polled system. In this system, the bus master explicitly polls each unit in turn. The unit may respond either with a message, or with an acknowledgement to show that it is still connected and functioning. This has the advantage of rigid control over message transmission by the master. It allows a priority system to be implemented by polling a particular unit more often than the others.

Unfortunately, the synchronous bus LANs must have a control unit, and to protect against failure of this unit multiple controllers are normally used. Synchronous LANs do have the advantage of a lower message delivery time than the asynchronous systems when under high loading. Additionally, by using the bus controllers to maintain an error recovery scheme, they can offer guaranteed error free delivery of messages with fewer overheads than in the asynchronous systems.

However, this type of system does include certain overheads due to the poll messages which are not present in an asynchronous system. Careful design is needed to keep these overheads as low as possible. The major advantage of linear LANs over ring LANs, is that by careful choice of the signalling scheme and media used for the bus, it is possible to make it passive.

1.2 The Choice of LAN

Whilst there are many different LANs, there is no one type which is 'the best'. Each type has different characteristics which make it suitable for certain applications but completely unsuitable for others. In general, a careful assessment of the situation must be made before the choice of LAN for a particular application can be made. After such an assessment, the Admiralty Surface Weapons Establishment (ASWE) set down the design for their LAN, which is known as the ASH (ASWE Serial Highway) [11].

For many years, the Royal Navy have used large mainframe computers in their ships. These computers monitor the ships' surroundings with the aid of the radar systems, and provide target extraction and identification facilities to the officers in charge of the ships' operation. In addition, the computer provides automatic control over the various weapons systems used on board ship. As the years have advanced, the complexity and number of the radar systems, the displays, and the weaponry, have increased dramatically. This has led to two distinct problems in the modern naval ship. Firstly, the performance of the ship is very seriously affected if the computer ceases to function. Secondly, the amount of cabling necessary to route all of the control and monitoring functions to the master computer is immense. The adoption of distributed control using a local area network system will solve both of these problems. Every sub-system on board the ship, such as the radar, the gun-turrets, the missile launchers, the status displays etc; will contain a mini or micro computer. They will all be linked together via a local area network. The only cabling necessary for such a system will be the local wiring from the distributed processors to their subsystems, and the LAN cabling, which will consist of several redundant highway cables. Each sub-system could then be factory tested with its local processor in full control.

The LAN used in such a system would have to have the following characteristics.

1) A very high resilience to individual element failure.
2) Guaranteed error free message delivery.
3) A simple, easily available highway cable/ connector to allow simple maintenance and repair.
4) A maximum cable length of 300 metres (due to length of ship).
5) Ease of alteration and reconfiguration.

It was decided that it would be very difficult to attain the necessary message throughput and overall system reliability needed by using a ring bus or star network. This meant the choice of a linear bus architecture. The need for guaranteed error free message delivery, and the knowledge that the bus was to be operated under a fairly high loading at all times dictated the choice of a poll-response system. Unfortunately, this type of system suffers from the obvious setback of centralised control, however the ASH was designed include multiple redundant controllers to alleviate this problem. In addition, the use of a passive highway, implemented using a screened twisted pair, allows the maximum cable length of 300 metres to be achieved without the use of repeaters. The signalling system chosen, a variant of Manchester coding known as Bifrequency Code (section 2.2) allows the cables to be connected without the need for any checks on polarity. The choice of NATO standard cable and connectors allowed the LAN to be simply and easily installed.

Unfortunately for ASWE, such an LAN was not available, so it was necessary to design their own. Great emphasis was placed on the need for simplicity of the host computer to LAN sub-system software interface. This led to the adoption (section 2.1) of a table driven interface between the LAN and its host computer, using an area of shared memory.

1.3 Conclusion

Local area network technology has advanced as the need has arisen for efficient communications between units in a loosely coupled distributed computing system. In such systems, a serial cable is normally used as the transmission media, and by a careful choice of signalling conventions, data rates up to 20Mbits/ sec can be achieved. Several possible topologies exist for LANs, each one suited to a different application. In many instances, distributed computing systems communicating via LANs have replaced single mainframes. This replacement was prompted by the need for greater overall system reliability, and the greater availability of mini and micro computers in recent years. In addition, the introduction of the software addressing used in LANs, in preference to the implicit hardware addressing used in earlier point to point systems, has greatly reduced the hardware changes necessary to reconfigure the system. In order to increase the reliability of the LANs, multiple redundancy is used for critical components and cabling. Careful choice of highway architecture allows a guaranteed error free message delivery, which may be critical in certain applications. ASWE have designed an LAN system for use in a future generation of naval ships and their choice of architecture gives the best performance in the naval environment with which they are concerned.

Chapter 2

The A.S.W.E Serial Highway

2.1 Introduction

The ASH was designed as a response to the needs of ASWE for a high speed local area network with guaranteed error free message delivery and a very high system reliability. Its specifications are laid out in Defence Standard 00/19 [11]. The network is of the poll-response linear bus type. In order to increase system reliability, there is a possibility of a multiple redundant highway cable and/or highway controller configuration. The line signalling and message protocols are handled by dedicated bit-slice front-end processors (FEPs), using AMD 2901 four bit wide devices. These processors are connected to the serial highway cable via transceivers, and to the host processors by specialised Direct Memory Access (DMA) interfaces. The host processor controls the communications processor by means of a set of tables in an area of shared memory. All messages sent on the highway may be put into one of two categories:- control and information messages. The information messages may be further divided into two types; short messages and block messages.

2.2 Signalling Conventions

The line signalling is performed using a variant of Manchester Code known as Bifrequency Code. The signalling rate is 3Mbits/sec. The valid signals are shown in Figure 2.2a. There are also two signalling violations defined as part of the specification, one to signal End of Message (EOM) and the other to signal End of Invalid Message (EOIM). These violations are shown in Figure 2.2b. The signal levels are detailed in Figure 2.2c. Since the highway is a passive linear bus,

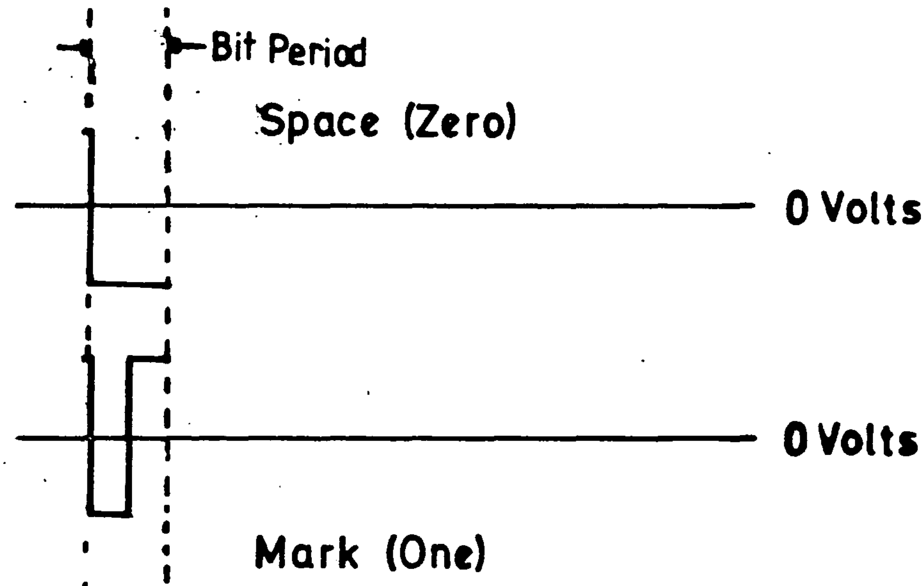


Figure 2.2a

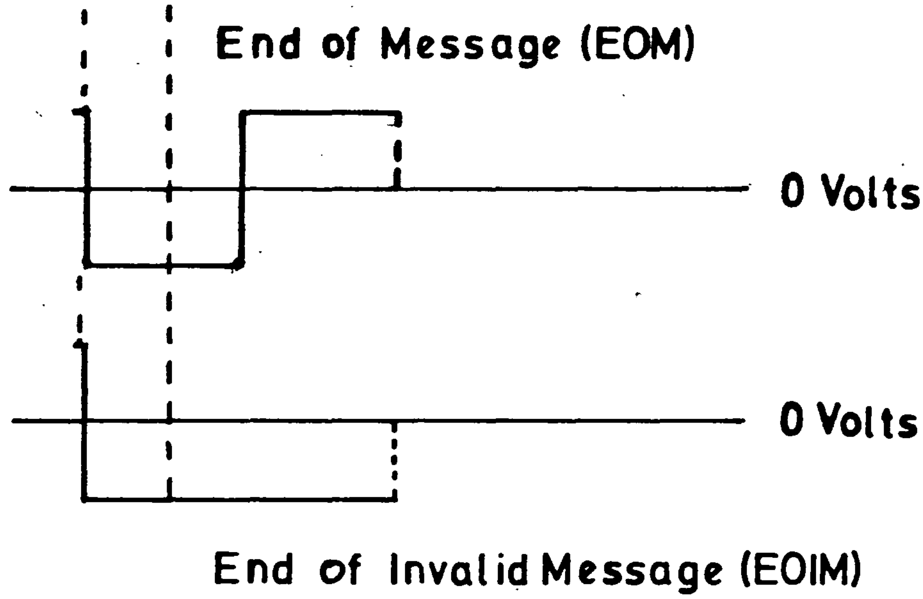


Figure 2.2b

these signal levels are subject to considerable degradation when a large number of units are connected, and/ or a long cable is used. The maximum level of degradation permitted is shown in Figure 2.2c. Error recovery is performed by the retransmission of incorrect messages.

2.3 Message Protocols

2.3:1 Control Messages

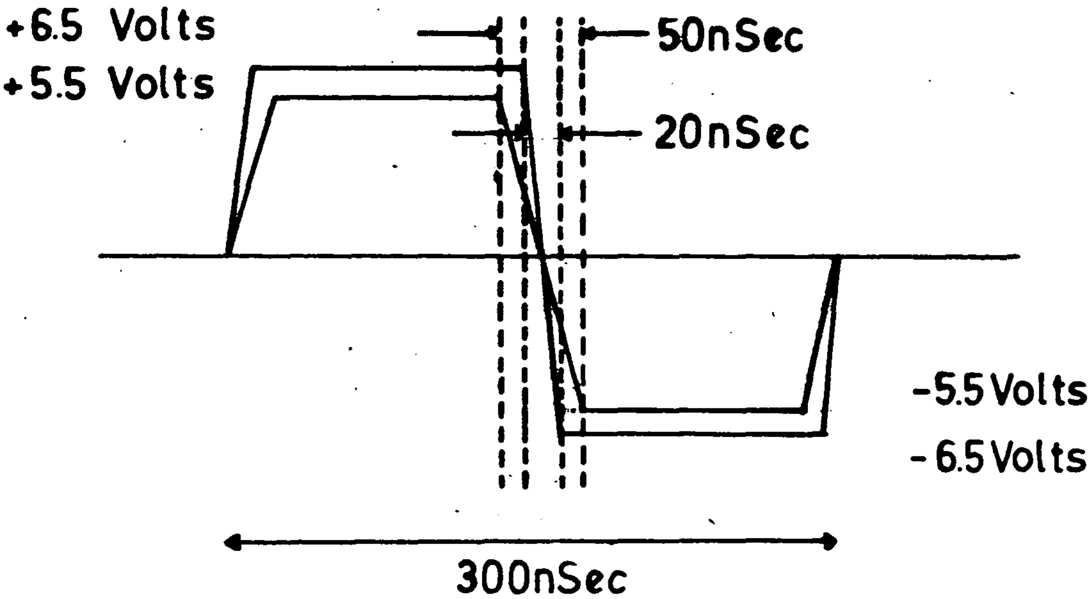
There are four types of control messages whose function is to maintain the poll and response scheme and to manage the error recovery system. The format of these messages may be seen in Figure 2.3:1a.

1) Permission to transmit (PTT): This message is issued by the highway controller and it gives a terminal specified by the DST field permission to use the highway.

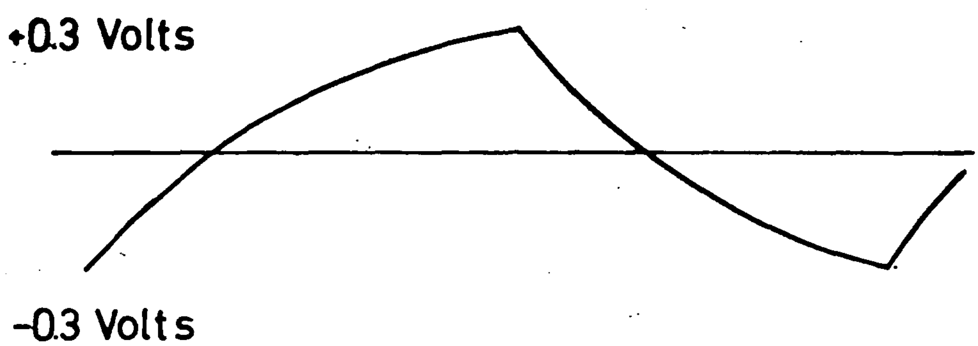
2) Nothing to Transmit (NTT): This message is issued by a terminal in response to a PTT. The NAK field is used by the highway controller in the error recovery system.

3) Repeat Message (RM): This message is issued by the highway controller when a terminal unit indicates that a message has been missed. It takes the format of the class of information message of which it is a repeat, except that byte 2 is equal to byte 4.

4) Null Repeat Message (NRM): This message is issued by the highway controller when it is unable to obtain a valid response from a terminal, or when the controller does not have an error free copy of the message to retransmit.

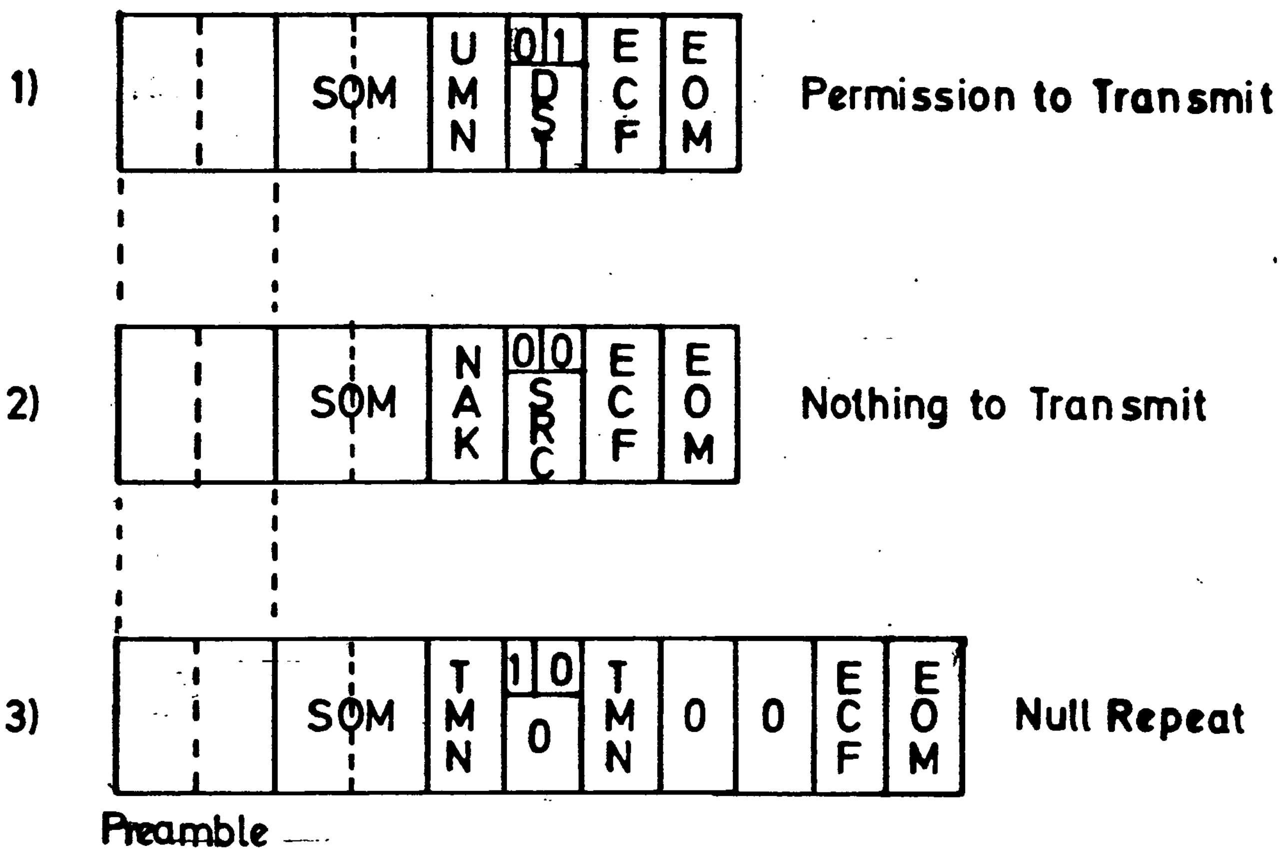


Signal Levels at Transmitter



Maximum Signal Degradation

Figure 2.2c



Control Message Formats

Figure 2.3.1a

2.3:2 Information Messages

These messages are used to pass information between the highway units. There are two classes of message, a data message and a block message.

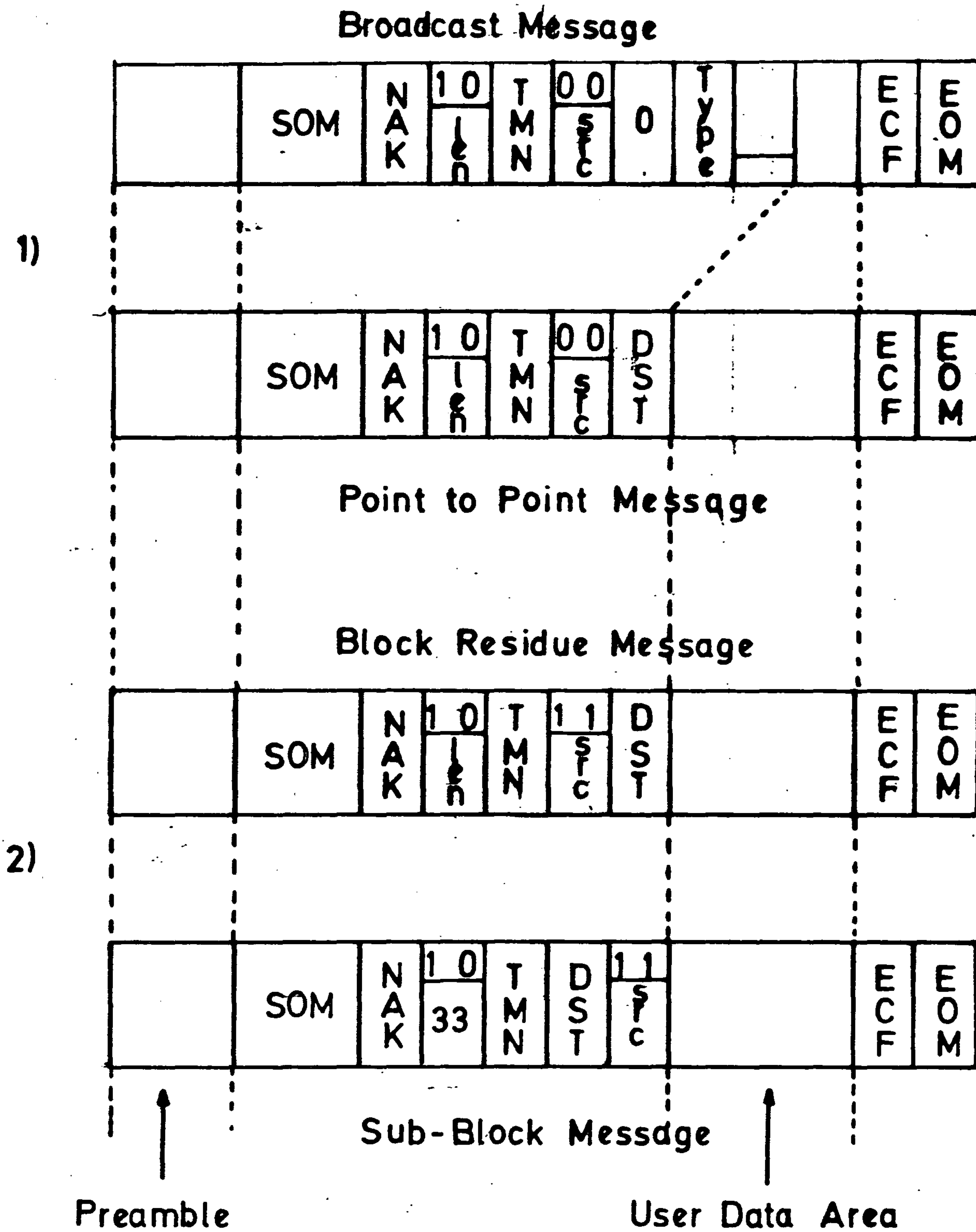
1) Data Message: This may either be directed to a particular terminal unit, in which case it is termed a Point to Point Data Message, or it may be a Broadcast Data Message. The format of each type of message is shown in Figure 2.3:2a. In each case the length of the message may be in the range 2-31 words inclusive.

2) Block Message: A Block message transmission is made up of two types of messages, a Sub-Block Message and a Block Residue Message. The length of the former is 33 words, whilst the latter may be in the range 2-33 words inclusive.

2.3:3 Message Fields

The first byte of a message to be transmitted is defined to be byte 0 and subsequent bytes as byte 1, byte 2 etc. The first bit of each byte is defined to be bit 0. As can be seen in Figures 2.3:1a and 2.3:2a, there are several different fields within the messages. The function of each is as follows

1) Preamble: This is an optional series of ones which is used to obtain hardware synchronisation between transmitter and receiver.



Information Message Formats

Figure 2.3.2a

2) Start of Message: This field consists of two bytes (0 & 1) which consist of fourteen ones and two zeros.

3) Error Check Field: The error check field occupies the last byte of every message and in a message of length 'n' is the modulo 256 sum of bytes 2 to n inclusive.

4) Use Message Number (UMN)

Transmit Message Number (TMN)

Not Acknowledge (NAK)

These fields are used in the error recovery scheme. The UMN is issued by the highway controller as part of any message transmitted by it. The TMN and NAK fields are issued by the terminal units.

5) Type Field (MTB): The type field is set by a transmitting unit, and allows selection by the receiving unit of the types of messages to be received. Messages with an unwanted type are discarded by the receiving unit.

6) Source Field (SRC): This field is set by the transmitting unit to the units highway number (in the range 0-63).

7) Destination Field (DST): This is set by the transmitting unit in a point to point transmission and causes the message to be discarded by all units apart from the one whose highway number matches the DST field.

8) Length Field : This field corresponds to the number of sixteen bit words in the message between byte 4 and the Error Check Field (exclusive).

9) End of Message (EOM), End of Invalid Message (EOIM) : These fields occur after the Error Check Field. The EOM is used after an otherwise valid message, whilst the EOIM is used after a message during the transmission of which some error occurred (such as buffer overrun/ underrun).

2.3:4 The Error Recovery System

Error recovery is performed by a system of error detection and retransmission of messages. All data messages are assigned a 'message number'. This number is assigned by the highway controller when it polls the terminals, and is held in the UMN field of the poll message. When the terminal responds to a poll from the controller with a data message, it inserts this number into the TMN field of the message. As terminals receive messages, they maintain a count of the highest TMN received in contiguous sequence. Any break in the sequence indicates the loss of a message. In this case, when the terminal responds to a controller poll, it responds with a 'nothing to transmit'. The controller will then be able to determine that the TMN sent by the terminal (contained within the NAK field) does not match its UMN, and the terminal therefore needs to have some messages repeated to it. The controller maintains a buffer of the 256 most recent messages it has received and is able to retransmit the message to the terminal from this store. The terminal should then respond with the correct TMN. A terminal unit which cannot be correctly updated will have the 'NAK stuck' status flag set (section 2.9:2). Should the controller receive a message in error, it will repoll the terminal up to four times before it is locked out of the polling sequence with the 'NR' status bit set (section 2.9:2).

2.4 Front-End Processors

The block diagram of this circuit may be seen in Figure 2.4a. The processor is based on two four bit wide microprocessors (AMD2901s) [12,13]. These are very high speed bipolar microprocessors, of a type known as 'bit-slice'. They are designed in such a manner that they may be cascaded together in parallel to obtain the desired word length. In this system, the use of two of these parts gives a word length of eight bits. The block diagram of an AM2901 is shown in Figure 2.4b. It consists of a two port RAM, a high-speed ALU and associated shifting, decoding and multiplexing circuitry. It is controlled by means of an externally generated instruction word, which is nine bits wide. Three of these bits select the ALU source operands, three the desired ALU function and the remaining three the destination register.

This instruction field of nine bits is obtained from a microcode store, the full size of which is 512 words by 32 bits. The fields in a single microcode word are shown in Figure 2.4c. Additional fields are used to select external registers which may be either read (Field A) or write (Field B) registers, and to supply a constant input to the 2901s when selected (Literal Field). The microcode store is addressed by a simple program counter which itself may be selected as an external write register by the 2901s, allowing unconditional branching. Conditional instruction skipping is implemented by using four of the microcode bits which select the desired 'skip flag', the state of which determines the subsequent state of the least significant microcode address bit.

There is also a FIFO buffer on the processor card. This is seen as an external register pair by the 2901s, a write only and a read only

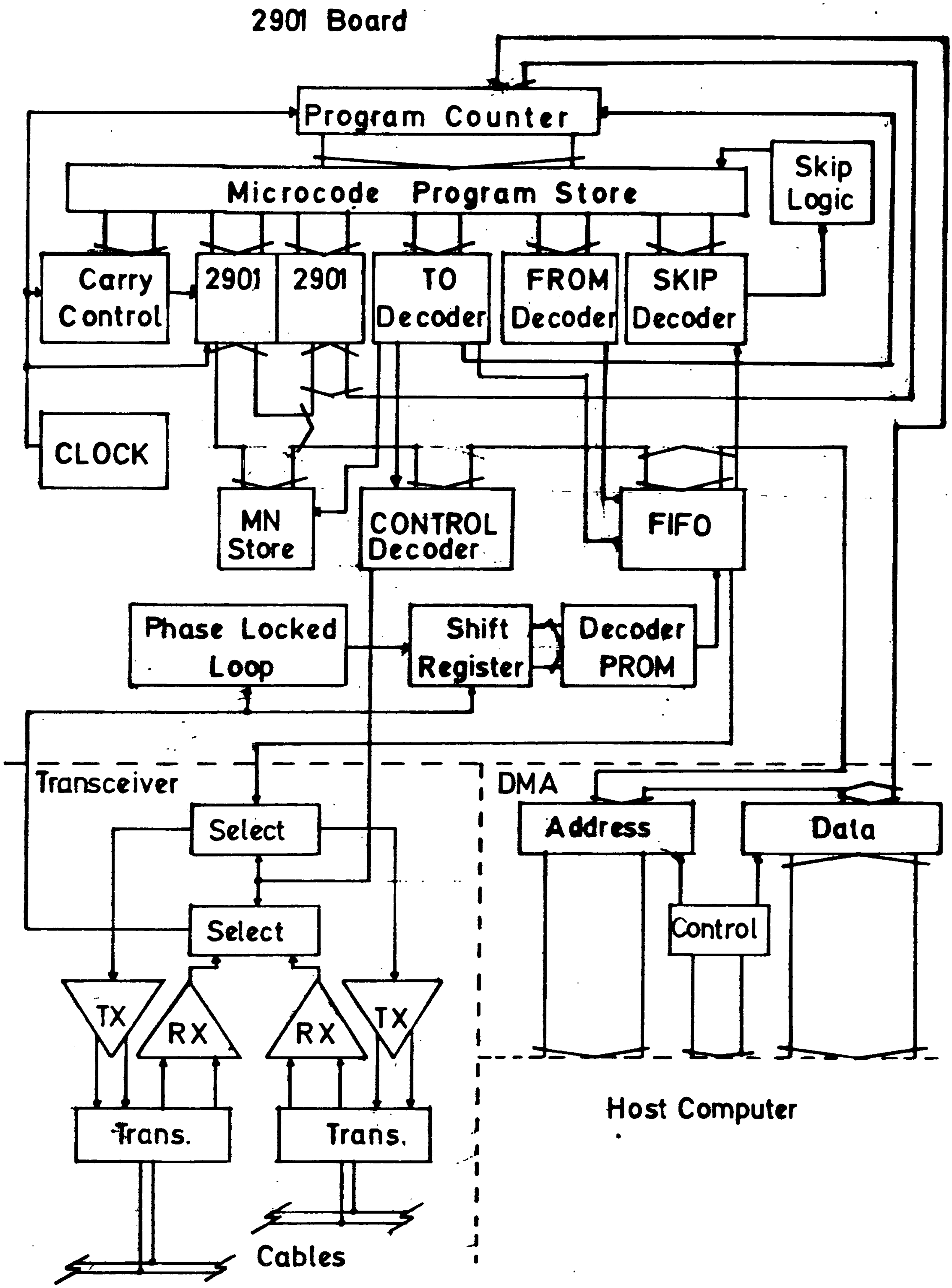


Figure 2.4a ASH Communications Processor

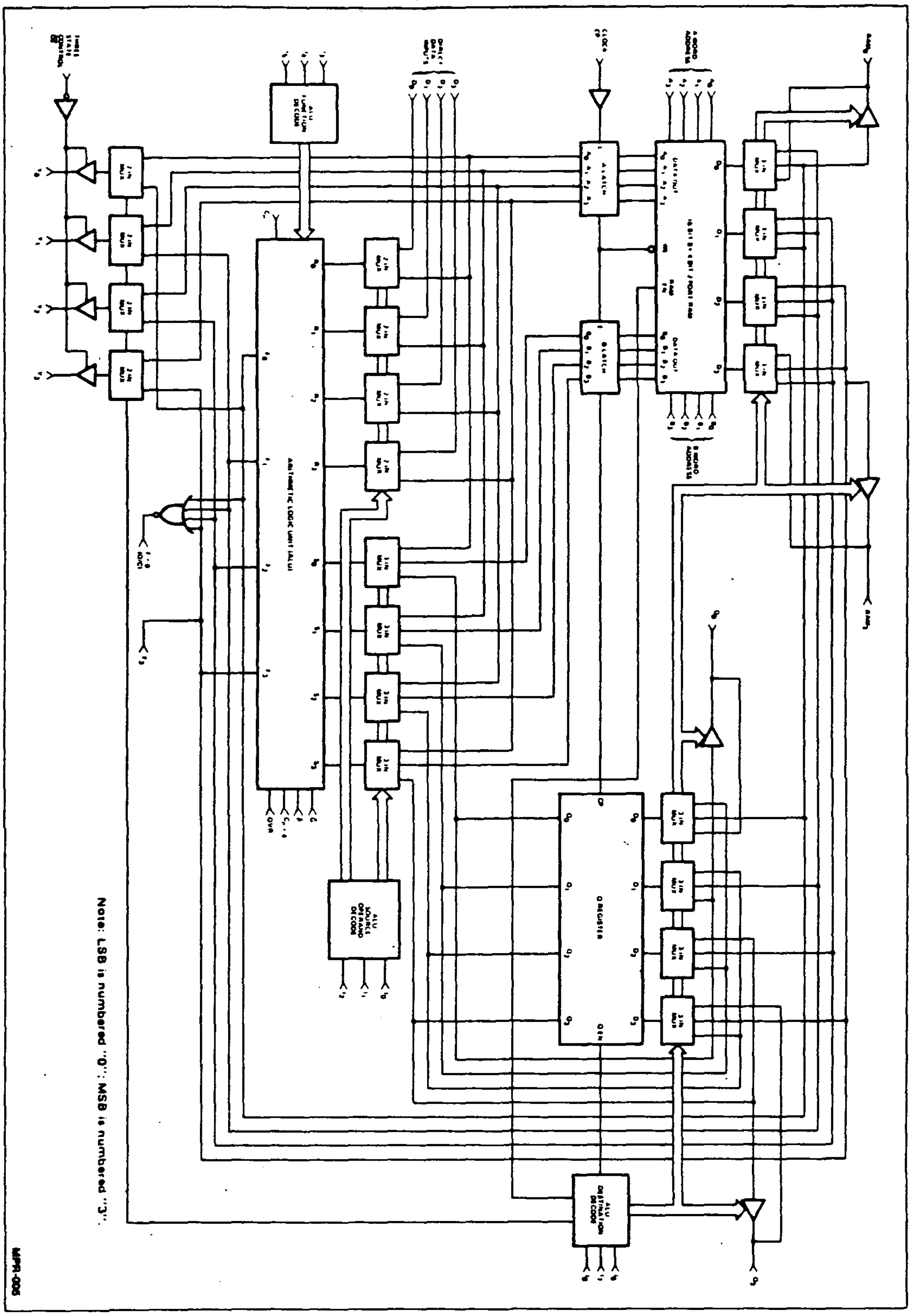


Figure 2.4b.

Block Diagram of 2901

2901 Microcode Format

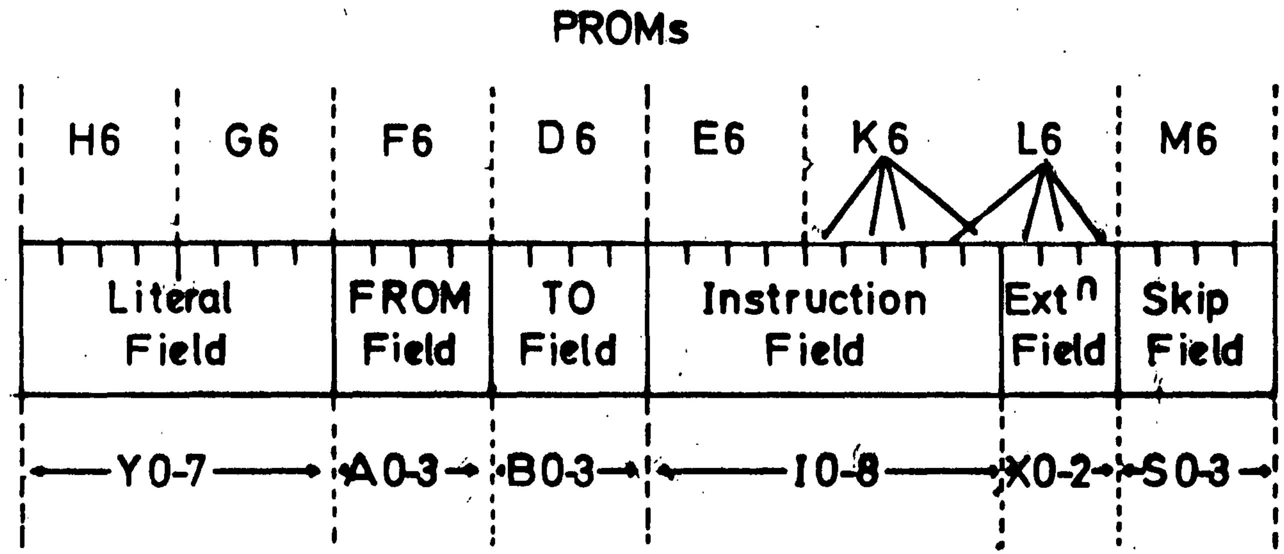


Figure 2.4c

register. The FIFO status flags are connected as skip flags to the skip logic. Also present on the processor card is a control signal decoder, accessed as an external write only register, which is used to supply control signals to various parts of the system.

The input to the FIFO buffer from the transceiver card is decoded from a bifrequency signal to a serial TTL compatible bit stream with the use of a decoding PROM. Initial synchronisation between the decoder and the received signal is achieved with a tracking phase locked loop. The output from the FIFO to the transciever is a bit stream which is encoded to a bifrequency signal on the transceiver board.

Also included in the encoding/ decoding/ buffering section of the processor board is a hardware interlock which restricts the maximum length of continous transmission to approximately 220us.

2.5 Transceiver

The transciever board contains two sets of transmitters and receivers to support a dual redundant highway system. It can be expanded to a triple redundant system by adding a third transciever on the board. Outgoing messages are transmitted on all cables, but messages are received on only one cable at a time. The cable to be used is selected from the processor board by the use of control signals and external registers.

The receiver is of the zero crossing detection type, and was designed to be tolerant of the type of signal degredation previously mentioned (Section 2.2). The transmitter encodes the serial bit stream from the FIFO buffers on the processor board into bifrequency code. It should be noted that one of the control signals (C5) is used in order to send an EOM since this is a coding violation and would therefore be unobtainable by merely sending data to the FIFOs, as would happen during a normal transmission.

2.6 Host Interface

This board allows the 2901 processor card to read and write to the host computers memory. A write transfer is initiated by the 2901 writing two bytes of data, the most significant byte and then the least significant byte, onto latches on the interface board. These latches appear as external write only registers to the 2901s. A read transfer is initiated by the selection of a control flag (C1) by the 2901 board. In each case, the address has been previously set up by the 2901 board. The address is written into counters on the interface board (which auto-increment after each transfer). The successful conclusion of a transfer can be detected by the 2901 board by monitoring the relevant 'skip' flags (S3 & S5).

Additional control over the highway sub-system by the host processor is obtained either by writing to a memory-mapped control register on the interface board, or by using bus control lines (depending upon the method most suited to the host computer's architecture). The host computer controls the FEP with the aid of a set of latches in the interface. The first latch either enables or disables the FEPs ability to interrupt the host computer when it strobes its 't6' line. The second latch sets the start/stop skip flag to the FEP (S4). The third control from the host performs a direct reset of the FEP by pulsing its 'RESET' line.

2.7 Microcode Cross Assembler and ASH Simulator

In order to program the front-end processors, a custom cross-assembler was written [14]. This allowed the microcode instructions to be written in terms of user selectable register names (allowing greater program readability) and register operations. The output of this cross-assembler was a file of microcode suitable for programming the microcode store (in PROM). Samples of this microcode may be seen in Figure 2.7a.

Also, in order to test the correct operation of the microcode without resorting to repeated programming of PROMs, an ASH simulator was written [15]. This took as its input the file of assembled microcode previously mentioned, and by use of a comprehensive set of monitoring instructions, either a single or multi-step simulation of the entire ASH could be achieved. Whilst this did not allow any measure of the real time performance of the system, it did allow substantial debugging of the microcode at a lower level.

2.8 ASH Terminal Unit

2.8:1 Software Tables

Communication between the host computer and the front-end processor is maintained via a set of software tables. A PROM is included as a set of read only registers on the 2901 board pre-programmed with a set of system constants. In the case of the terminal units, only two constants are used, the terminals' Highway Number (in the range 1-63) and the address of the start of the primary table in the area of shared memory. The host computer must be pre-programmed with the address of this table. The format of the primary table may be

```
B61   330C337110              posn=[tptta
B70   0809037010              a,lsa=#08  / receive error
B71   BA00137010              branch cntrl
B80   0A09037010   nullr:     a,lsa=#0a  / null repeat
B81   BA00137010              branch cntrl
B90   0709037011   messg:     a,lsa=#07 , skip  / data message
B91   0909037010   rptmg:     a,lsa=#09  / repeat
BA0   006A107030   cntrl:     msa=start+0
BA1   000F137010              contrl=#00
                              #even
BB0   BB00137015   cntr2:     branch cntr2 , sdnr
BB1   0009101030              lsa=0+a
BC0   002B307130              temp5=msdo+0
BC1   0010007100              a=lsdo+0+1
BD0   00BC104020              msdi=0+temp5+cp
BD1   000B101030              lsdi=0+a
                              #even
BE0   BE00137013   cntr3:     branch cntr3,sdnac
BE1   00C0104030              t0=0+posn
                               /
                               /            wait
                               /!************************************
                               /this allows time after a repeat for the terminal tables to be
                               /handled and for necessary message number updates
                               /
BF0   0032022130   wait1:     a=umn-nackn-1  / number of updates
BF1   2000005130              a=#20+a
C00   000001113F   wait2:     a=0_a-1 , szero
C01   C000137010              branch wait2
C10   C30C337110   wait3:     posn=[seq1
C11   040014611F              0=#04 and flag1 , szero  / no skip if null rpt
C20   BB00137011              branch nullr
C21   B900137011              branch rptmg
                               /
                               /
                               /repeat sequence  (seq)
                               /*************************
                               /
                               /
                               / entered after a repeat from wait via cntr
                               / repeat sequence must not be entered after no response.
                               / the value of 'tries' should allow time for software to clear a
                               / buffer if thats why nak is stuck. if the nm store is
                               / broken the number of tries will be used up and the terminal

                               / will be locked out.
                               /
                              #even
C30   200014611F   seq1:      0=#20 and flag1 , szero / no skip if no response
C31   0A00137011              branch ptab1
C40   010014611F              0=#01 and flag1 , szero / no skip if in seq
C41   2100137011              branch tptt1
C50   0100336110              flag1=#01 ior flag1  / set in seq
C51   0029314130              onak=0_nackn-1
C60   00E7307130              rpt=tries+0
C61   2100137011              branch tptt1
                               /
                               /
                               /timtx message output (timtx)
                               /******************************
                               /
                               /this is entered from tptt when the timtx available bit is set.
                               /
C70   C500037110   timtx1:    a=#c5 / length
C71   000F137010              contrl=#00
C80   003F304130              temp9=0+umn  / retain umn
C81   0003337110              umn=#00
C90   CA0C337110              posn=[timtx2
C91   AF00137010              branch star1
                              #even
CA0   00F3304130   timtx2:    umn=0+temp9 / reset umn
CA1   050D337110              word=#05
CB0   0400037110              a=#04
CB1   00EE330110              flag3=flag3 ior a
CC0   A800137011              branch chff3
                               /
                               /after time is output
                               /
CC1   0B09137010   timtx3:    lsa=#0b
CD0   000C137010              msdi=#00
CD1   000B137010              lsdi=#00
CE0   EB00037110              a=#eb
CE1   00EF340110              flag3=flag3 and a / clear time output flag
                              #even
CF0   CF00137013   timtx4:    branch timtx4 , sdnac
                               /  delay to allow terminal to handle time msg
CF1   000000110F              a=0+a+1 , szero
D00   CF00137011              branch timtx4
D01   FF00137010              branch patch?
```

Figure 2.7a

Samples of 2901 Microcode

Field	Field	Field	Address
In Interrupt Mask	No. of In Buffers		0
Input Position			1
Input Table Location			2
Out Interrupt Mask	No. of Out Buffers		3
Output Position			4
Output Table Location			5
Message Type Table			6 ↓ 37
Highway Number			38
Receive Error Counter			39
Data Starvation Counter			40
Retransmission Counter			41
Buffer Overflow Counter			42
In Data Available	In Transfer Fail	In Res. Length	43
In Block Source			44
In Sub-Block Total	In Sub-Blocks Recvd.		45
In Block Start Address			46
Out Data Available	Out Block Error	Out Res. Length	47
Out Block Destination			48
Out Sub-Block Total	Out Sub-Blocks Tx'd		49
Out Block Start Address			50

Figure 2.8:1a Terminal Primary Table

seen in Figure 2.8:1a. As can be seen, the locations and characteristics of all the other tables and terminal unit functions are held in the primary table. They are as follows:-

1) In Interrupt Mask: This mask is set by the host and is used by the front-end processor to determine whether to interrupt the host at input buffer wrap-around.

2) Number of In Buffers: This field is preset by the host (in the range 1-64) and sets the number of buffers in the input queue.

3) Input Position: This field is used by the front-end processor to indicate the location of the next free Input Buffer. It must not be altered by the host during normal operation.

4) Input Table Location: This field is preset to the word address of the start of the first Input Buffer by the host computer.

5) Out Interrupt Mask: This field is set by the host and is used by the FEP to determine whether to interrupt the host on output buffer wrap-around.

6) Number of Out Buffers: This field is preset by the host (in range 1-64)

7) Output Position: This field is maintained by the FEP and contains the index number of the next free output buffer. It must not be altered by the host.

8) Output Table Location: This is preset by the host to point to the start of the first output buffer.

9) Message Type Table: This field is set by the host to indicate to the FEP which message types it wishes to accept and which to reject. The field is 512 bits long (64 bytes), each bit corresponding to a particular message type.

10) Highway Number: This field is set by the FEP and corresponds to the highway number contained in its PROM.

11) Receive Error Counter: This field is maintained by the FEP and corresponds to the number of errors detected in incoming messages.

12) Data Starvation Counter: This field is maintained by the FEP and is incremented every time the FEP is unable to obtain data from/ transfer data to its host sufficiently rapidly to maintain input/output of a message.

13) Retransmission Counter: This field is maintained by the FEP and is incremented every time the highway controller requests a message repeat.

14) Buffer Overflow Counter: This field is maintained by the FEP and is incremented by one every time an overflow of input buffers occurs.

Fields Relating to Block Transfer

15) In/Out Block Start Address: Preset by the host.

16) In/Out Sub-Block Total: Preset by the host to the number of 32-word sub-blocks expected to be transferred.

17) In/Out Residue Length: Preset by the host to indicate the expected number of words in the block residue message.

18) In Block Source: Preset by the host to indicate the terminal node from which the transfer is expected.

19) Out Block Destination: Preset by the host to indicate the destination of the block transfer.

20) In Sub Blocks Received: Set by the FEP to the number of Sub Blocks actually received.

21) In Transfer Fail: Set by the FEP to 127 if the transfer fails for any reason.

22) In Data Available: This field is set to zero by the host when it has preset all of the other fields relating to the block transfer to indicate that the transfer may go ahead. It may not subsequently be updated by the host until it has been set to one by the FEP to indicate that the transfer has been completed.

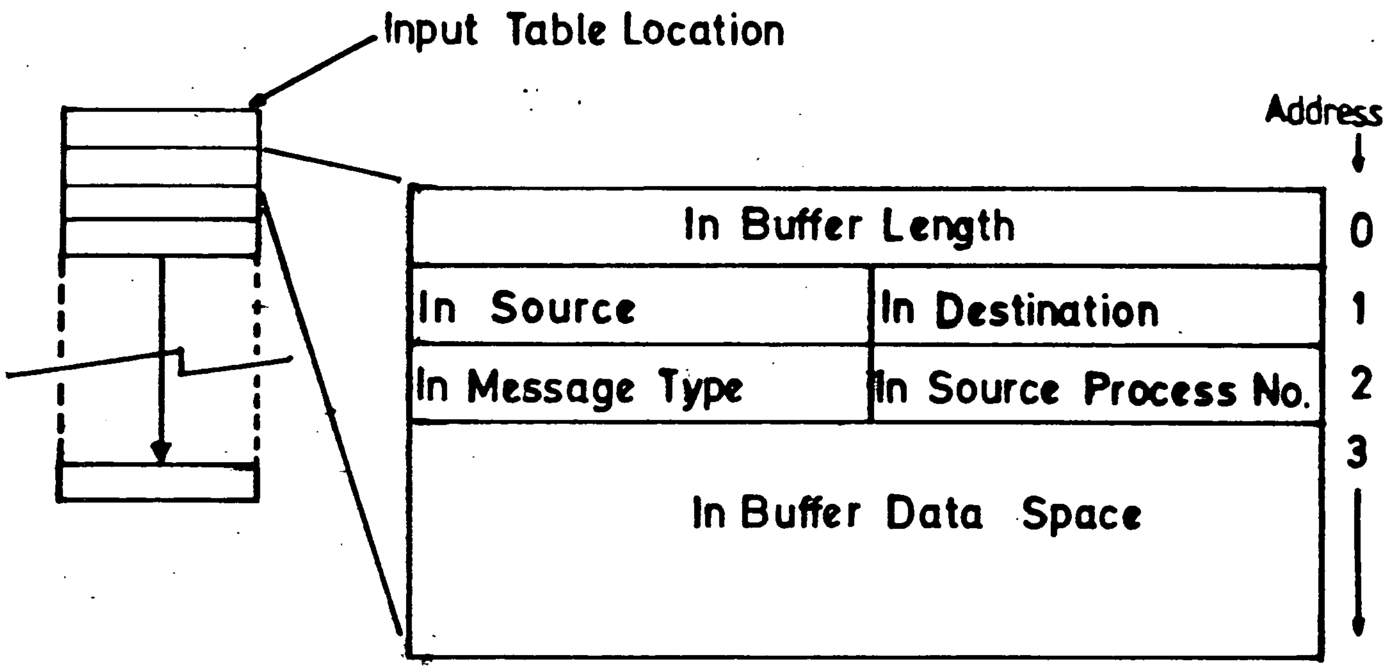
23) Out Sub Blocks Transmitted: This field is maintained by the FEP to indicate the number of sub blocks actually transmitted.

24) Out Block Error: This field is set to 127 by the FEP if the Out Residue Length is greater than 32 (i.e. an error has occurred).

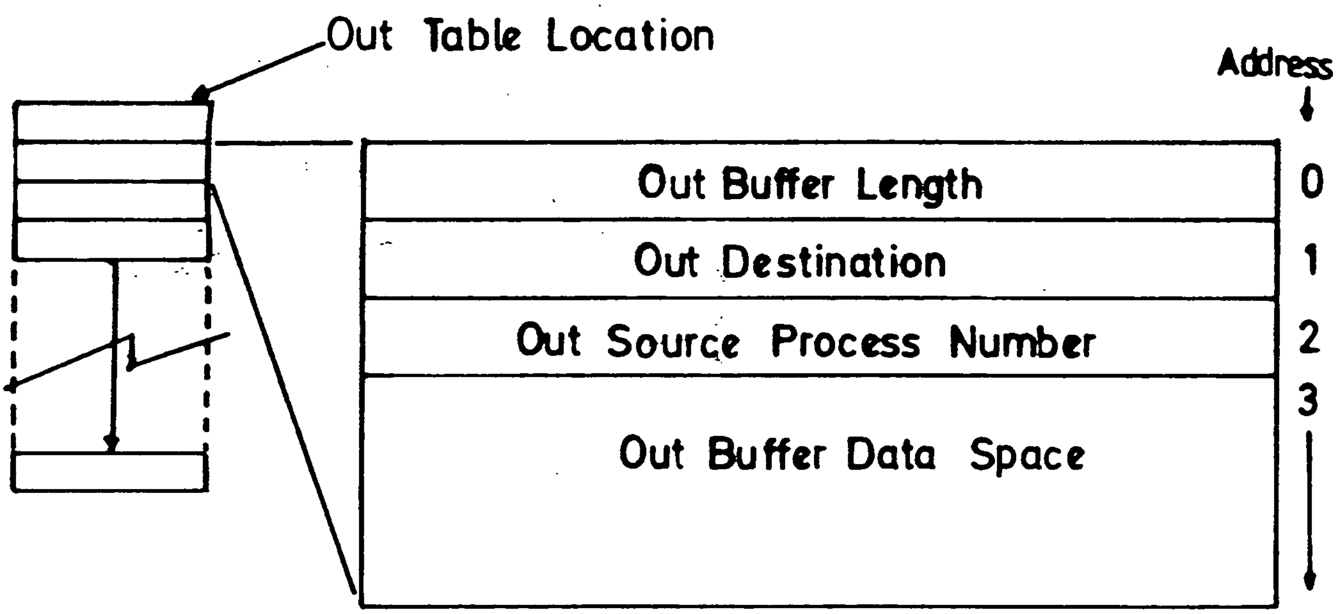
25) Out Block Available: This field is set to one by the host when it has preset all of the other relevant fields. It may not susbsequently by altered by the host until it has been set to zero by the FEP to indicate the conclusion of the transfer.

In and Out Table

These two tables have a similar structure which can be seen in Figure 2.8:1b. The 'Source' field has the same use and meaning as byte 5 (SRC) of an information message. The 'Destination' field has the same meaning as byte 6 of an information message. The 'Message Type' field is equivalent to the MTB (byte 7) together with the MTB extension bit (byte 8 bit 0) of a broadcast message. 'Buffer Length' when non-zero, indicates that the buffer contains valid information. When the information is either sent by the FEP (output buffer) or processed by the host (input buffer) this field should be set to zero to indicate that the buffer is free. The data area takes up the remainder of the buffer as determined from the Buffer Length field.



In Table Format



Out Table Format

Figure 2.8.1b

2.8:2 Host Control of ASH Terminal Unit

There are three additional controls from the host computer to the host processor not included in the tables. They are provided by using some form of programmed output instruction. These are reset, start/ stop and interrupt enable/disable. To perform an orderly starup of the ASH terminal unit, the host must first reset the unit. The ASH will now be awaiting commands. Next the host must tell the ASH to 'stop'. The host computer should then set up the primary and secondary tables, and then start the ASH unit. Subsequently all communication is via the software tables. To send a message, the host computer must determine the location of the next free output buffer. It then sets up all of the fields in this buffer, with the buffer length field being set up last, as this is the indication to the FEP that the buffer is complete and ready to be sent. When it has been sent the FEP will clear the buffer length field.

Message reception is transparent to the host computer, all it must do is to check the input buffers for a buffer with a non-zero buffer length field, indicating that a valid message has been received. It must then clear the message length field (after having copied the message elsewhere) to indicate that the buffer is again available.

Block message transmission is more complex, and requires a higher level of intervention by the host computers. The Block Receive fields in the destination unit must also be correctly set up before the transfer can go ahead. Therefore information about the impending block transfer must be exchanged between the transmitting and receiving units before the transfer can go ahead. This exchange is user dependant, the only constraints being that the number of Sub Blocks and Block Residue Fields set up in the tables of both the transmitting and the receiving units are identical.

2.9 Highway Controller Unit

2.9:1 Software Tables

In the highway controller communication between the front-end processor and its host is via a set of software tables [16]. The address of the 'primary table' is known to both, it being pre-programmed into the FEPs on-board PROM. In addition, there are four secondary tables, whose addresses must be set up by the host computer. These tables are as follows:-

1) Polling Table. This table, of length 64 bytes, is the list of terminals which the highway controller is to poll. The 'Pointer to the Polling Table' (primary table address two) may only be altered by the host when the FEP is halted.

2) Buffer Store. The pointer to this table is held in primary table address three, and is set up by the host computer prior to activation of the FEP. Any subsequent alteration will be ignored by the FEP. The 'Buffer Store' consists of 256 contiguous buffers each of length 34 words. It is used as a circular buffer which contains the last 256 transmitted information messages.

3) Size Store. The pointer to this table is held in primary table address four. Again, it is set up by the host computer prior to initial activation of the FEP and any subsequent alteration will be ignored. The store consists of 256 words, and is used by the FEP as a record of the length of the messages held in the buffer store.

4) Status Table. The pointer to this table is held in primary table

address five, and is set up by the host computer prior to activation of the FEP. Any subsequent alteration will be ignored. The status table is maintained by the FEP as a record of the status of each terminal which is being polled. The table is 64 words in length.

The format of the primary table can be seen in Figure 2.9:1a. In addition to the four pointers detailed above, there are several other fields in the primary table, whose function is as follows:-

1) Controller Terminal Unit Status Word (CTUSW). Primary table address zero. This field contains bits which are set to indicate the current status of the highway controller.

a) Bit zero is set to one when the FEP has been stopped by a channel control command (Section 2.9:2), and is cleared to zero when the controller is restarted.

b) Bit one is set to one when the controller is active and to zero when it is passive (Section 2.10).

c) Bit two is set to one when the controller has overridden a 'Go Passive' command (Section 2.9:2), and cleared within 20 milliseconds of the 'Go Passive' command being cleared, or when the unit does go passive.

d) Bit three is set to one if the controller is active and detects contention for control of the highway (Section 2.10). It is cleared when the controller next assumes active status.

Field	Word
Controller Terminal Unit Status Word	0
Controller Terminal Unit Control Word	1
Pointer to Polling Table	2
Pointer to Buffer Store	3
Pointer to Size Store	4
Pointer to Status Table	5
Self Test Scratchpad	6
	7
Receive Error Counter	8
Repeat Counter	9
Null Repeat Counter	10
Out Time Available	11
Out Time	12 16
In Time Available	17
In Time	18 22

Figure 2.9:1a Controller Primary Table

e) Bits eight to fifteen inclusive are set if the FEP detects a failure of the interface to the host computer.

2) Controller Terminal Unit Control Word (CTUCW). Primary table address one. This field is used by the host computer to control the activities of the FEP, in addition to the channel control commands normally used (Section 2.9).

a) When bit zero is set to one an active controller will assume the passive state (A Go Passive command).

b) Bits two and three are used by the host to select the highway cables on which the controller transmits. If the field is set to zero the controller will transmit on all cables. If the field is set to one, two or three, the controller shall transmit on only the selected cable.(n.b in the current implementation, only cables one and two are fitted, selection of cable three causes the controller to cease transmission on either cable).

3) Monitoring Counters. These are contained in primary table addresses eight to ten.

a) Receive Error Counter. Primary table address eight. This counter is incremented when an error is detected in a received message.

b)Repeat Counter. Primary table address nine. This field is incremented when the controller sends a repeat message.

c)Null Repeat Counter. Primary table address ten. This field is incremented when the controller sends a Null Repeat Message.

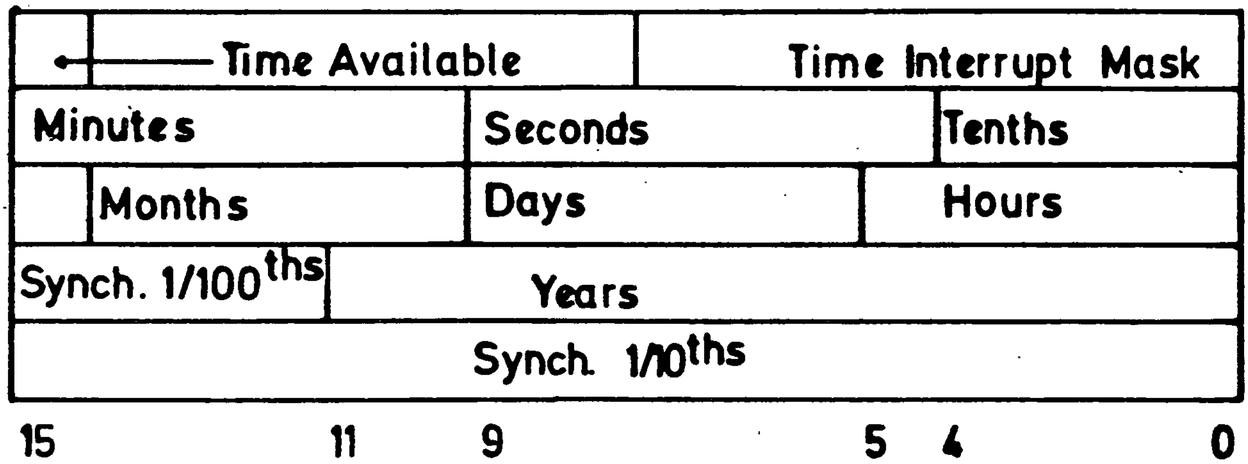
4) Time Fields. These fields are held in primary table addresses eleven to twenty-two. The time fields are sent by the currently active controller, and provide the complete time including year, month, day, hour, minute, second, tenths of seconds and one hundredths of seconds fields. The 'In Time' fields are used when the controller is passive to store any time message received from the active highway controller, while the 'Out Time' fields are used by the host processor to send a time message onto the highway when the controller is active.

a)Out Time Available. Bit fifteen, primary table address eleven. This field is set to one by the host computer of an active controller to indicate that the contents of the Out Time Fields are ready to be transmitted. It is cleared to zero by the FEP after the time message has been sent.

b)Out Time. Primary table addresses eleven to sixteen. This field is set by the host computer, the format of the complete field is shown in Figure 2.9:1b.

c)In Time Available.Bit fifteen, primary table address seventeen. This field is set by a passive controller after it has received a time message, and has updated the 'In Time' fields. The host may clear 'In Time Available' if it wishes to receive another 'In Time'.

d) In Time. Primary table address eighteen to twenty-two. The format of this field is also shown in Figure 2.9:1b.



Format of In/Out Time Fields

Figure 2.9.1b

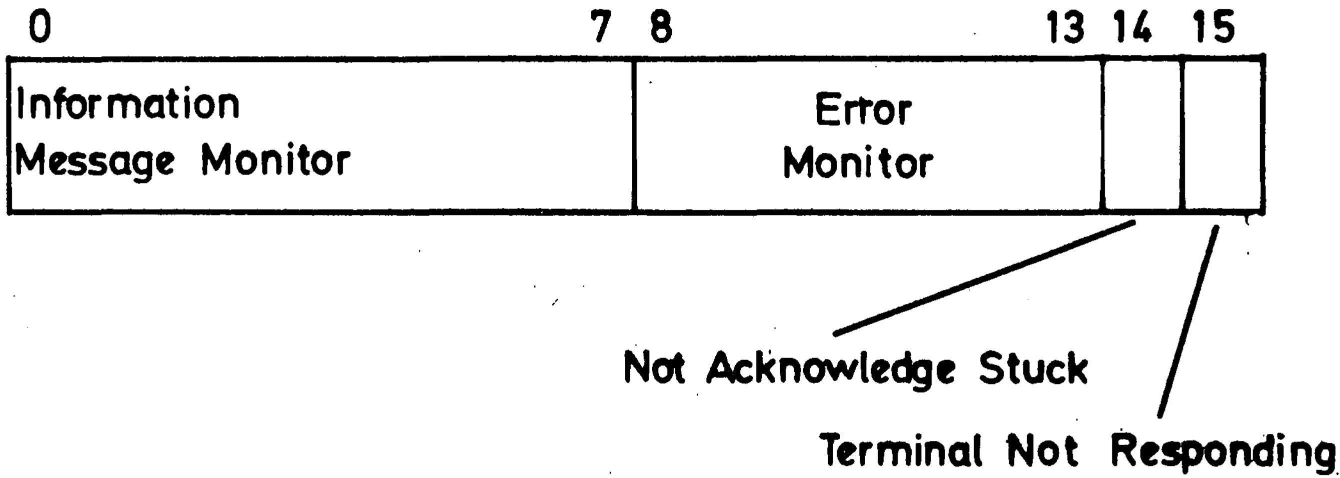
2.9:2 Host Control of Highway Controller Unit

The host computer has control of the controller unit (and therefore the operation of the highway) at three distinct levels. Firstly, it may use the channel control system to start, stop and reset the controller. Secondly, it may use the CTUCW to issue a 'Go Passive' command, or to change the cable used. Lastly, it may alter the tables, but this last control method must be used with caution, as an incorrect alteration can bring the controller, and possibly the entire highway, to a standstill.

The first type of control is used when the FEP is originally switched on. The hardware of the interface card assures that the FEP will be in its halted state immediately after the power is switched on. The tables must be set up by the host computer, and then the FEP may be started using the channel control system. The second control method may be used by the host at any time, and affects the operation of the highway as a whole. The last method of control may be used to add terminals to the polling table, or take them out of the polling table or reset terminals which have been locked out of the polling sequence. This lockout occurs under conditions detailed in section 2.3:3. The status of each terminal may be determined from the relevant status table entry. Each status table entry has several fields (Figure 2.9:2a) they are as follows:-

a) Information Message Monitor. Bits 0 to 7. This field is incremented by one for every information message sent by the terminal, and decremented by sixteen for each NTT message sent.

b) Error Monitor. Bits 8 to 13. This field is incremented by sixteen for each message received which included an error and decremented by



Controller Status Table Entry

Figure 2.9:2a

one for each error free message received.

c) Not Acknowledge Stuck. Bit 14. This bit is set if after four attempts a terminal still does not acknowledge receipt of the repeated message.

d) Terminal Not Responding. Bit 15. This bit is set if after four attempts, a reponse cannot be obtained.

If either bits fourteen or fifteen of the status word for a particular terminal are set, the FEP will stop polling the terminal unit. In this circumstance the terminal unit in question is said to be 'locked-out'. If a terminal has been locked out of the polling sequence, or is to be started up initially, it will be out of synchronisation with the rest of the highway. In order to reset a terminal in these circumstances, the controller must send it a reset message, and then recommence normal polling. The decision to send a reset message must be made by the host computer, based on either the status of a terminal as determined from the status table (i.e. locked out, NAK stuck or running) or by external operator intervention (i.e. adding terminals to the polling table). In order to send a reset message, bit fifteen of the relevant polling table entry must be set, and then cleared to return the polling sequence to normal. This obviously involves changes to the polling table. Changes may not be made to a polling table which is currently being used by the FEP, so they must be made as follows:-

1) Set up a new polling table at a different address to the current one, with the necessary alterations (additional terminals, reset bits set, etc.).

2) Set the channel control to 'Stop'.

3) Wait for the CTUSW stop bit to be set, indicating that the controller has finished a pass of the current polling table.

4) Change the polling table pointer (primary table address two) to point to the new polling table.

5) Set the channel control to 'Go'.

In order to satisfy the timing restrictions concerned with the takeover of an inactive highway by a previously passive controller (Section 2.10:2) the time between the controller setting the CTUSW stop bit, and the host setting the channel control to 'go' must be no more than fifty microseconds.

Thus to reset a terminal, this procedure must be performed twice, once to cause the controller to issue the reset message, and again to return polling to normal.

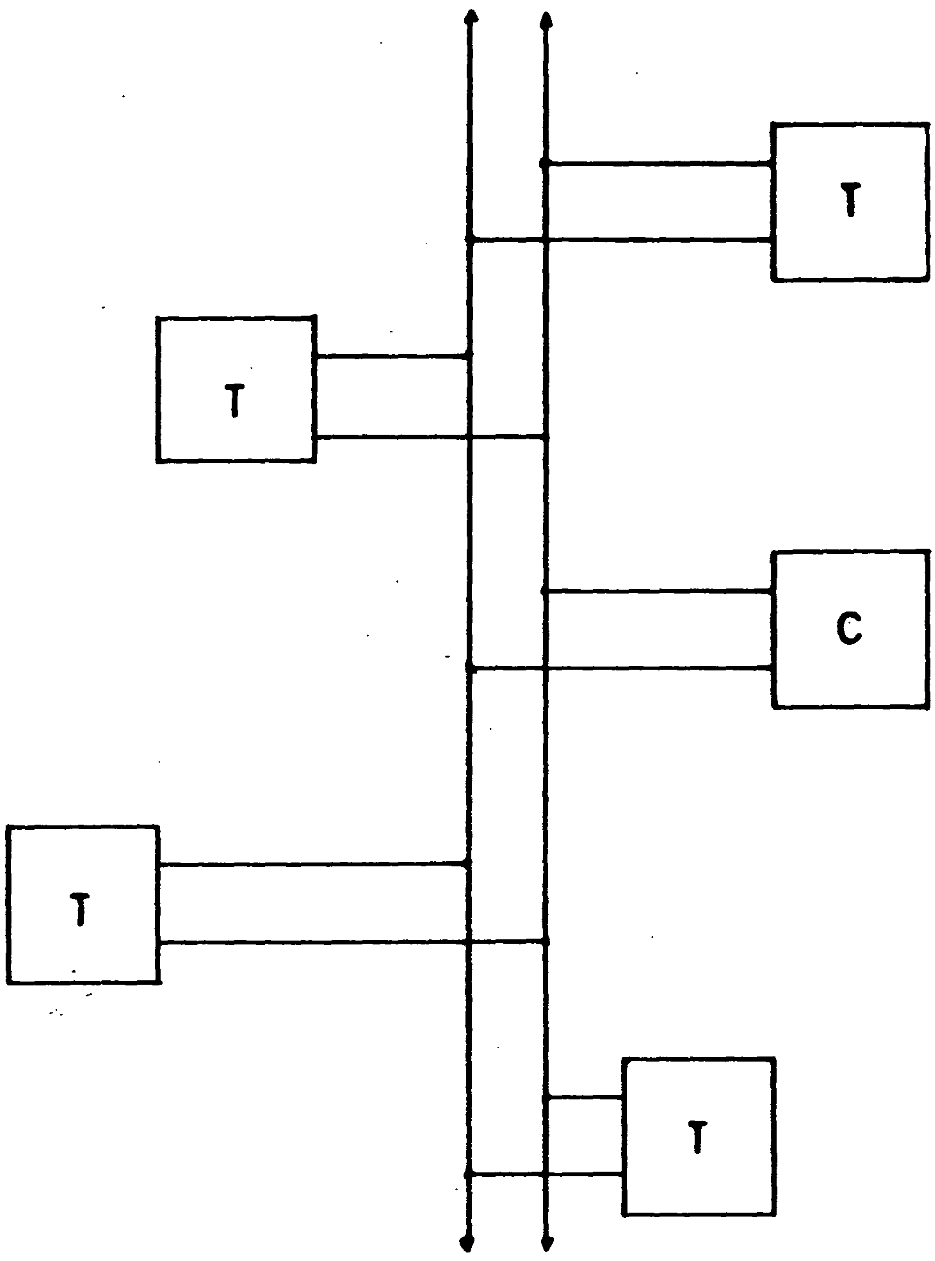
2.10 ASH Configuration

2.10:1 Single Controller/ Twin Highway Cables

The configuration shown in Figure 2.10:1a can include up to sixty-four terminals and one highway controller. These units are connected to a pair of twin screened cables which may each be up to three hundred metres in length. The pair of cables provide dual redundancy of the cabling where all units (apart from the controller, see Section 2.9:1) transmit on both cables and receive on one. The cable which is used for reception is decided by the FEP in each unit, on the basis of the cable with the highest number of error free messages received. The ASH was designed in an attempt to maximise the reliability of a network system and to minimise the possibility of overall system failure due to the failure of a single unit. Thus this particular configuration is not a good design since the failure of the highway controller can cause total system failure. The reliability of the system may be significantly increased by the inclusion of multiple controllers. In principle the scheme used for twin cable redundancy can easily be extended to include a greater number of redundant cables.

2.10:2 Twin Controller/ Twin Highway Cables

The configuration shown in Figure 2.10:2a is the one normally used at A.S.W.E. It is similar to that detailed in section 2.10:1, however there may only be up to sixty-three terminals and there are two controllers. In normal highway operation there will be one active and one passive controller. An active controller is one which is



T Terminal Unit

C Controller

Single Controller - Multiple Cable

Figure 2.10.1a

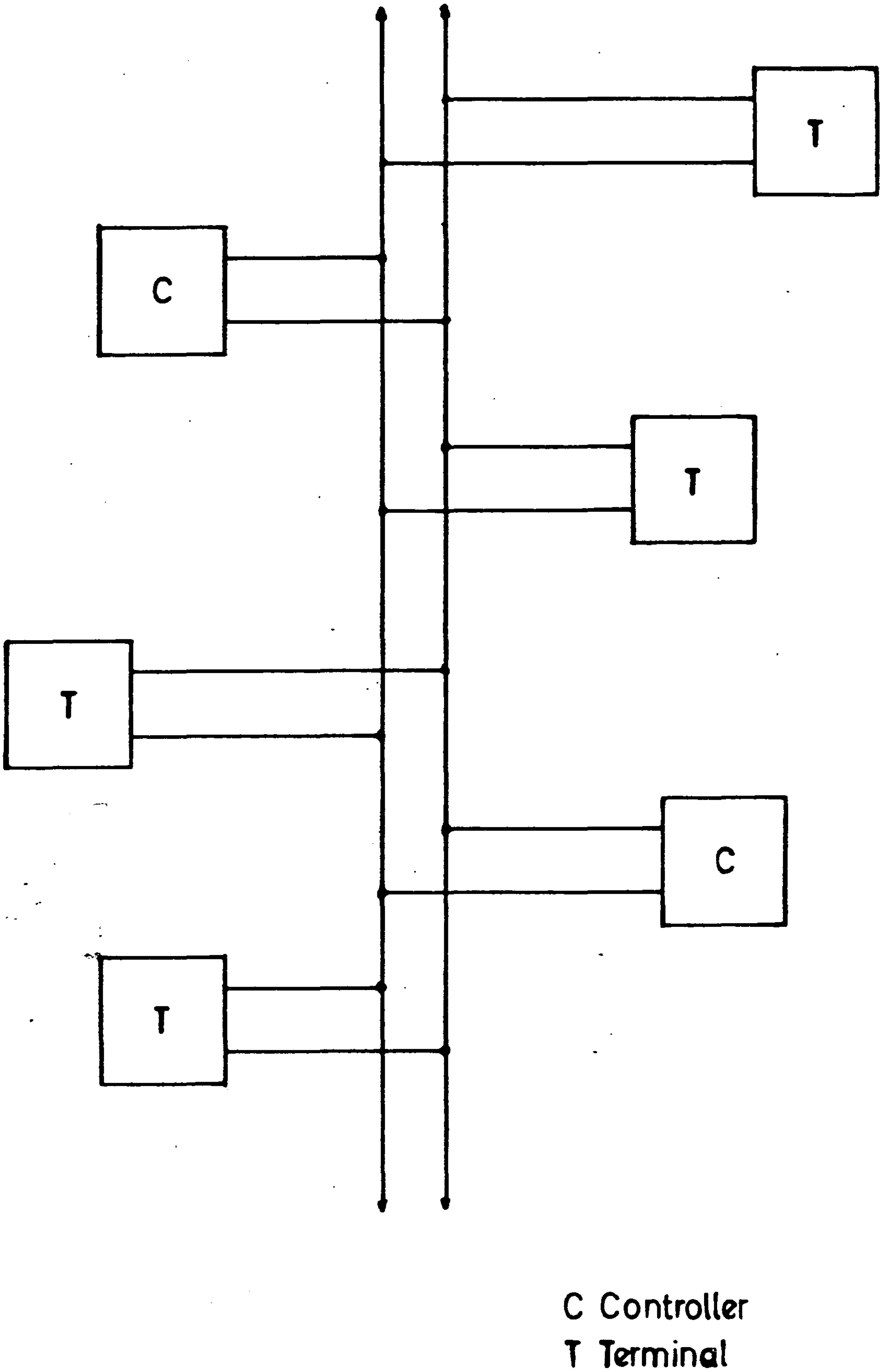


Figure 2.10:2a Twin Controller- Twin Cable

running as normal, while a passive controller is one which is running but is merely performing checks on the operation of the highway so that it is able to take over control should it become necessary.

The two controllers operate in what may be likened to a Carrier Sense system. If the passive controller detects a lack of any activity on either highway for 3 milliseconds it attempts to take over control of the highway. If an active controller detects any activity on the highway within the five milliseconds before the start of transmission it will assume a passive status. The active controller polls the passive one (Highway Number zero) and the passive terminal responds with an NTT message. The active highway controller cannot lock the passive one out of the polling sequence for any reason. If at any time an active controller detects contention for the highway (as a result of inspecting the highway immediately prior to outputting a message) it will change cables and wait for approximately ten to fifteen microseconds. If this second highway is still active, the controller will record a contention by setting the CTUSW contention bit (Bit 3) and assume a passive state. When a previously passive controller assumes control of the highway system, it must first reset all of the terminal units. It does this because it is almost impossible to maintain a complete duplicate of the status and message information held within the previously active contoller, and without this information it is impossible to restart the polling and error recovery system where the other left off. Using this method, the only major loss is of the message backup store which implies that should one of the terminals have 'lost' a message and have been awaiting a retransmission, this message will be permananently lost.

2.10:3 Cable Configuration

Due to the inherent problems of obtaining a reliable 'T' connection to a screened twisted pair, the actual conifuration of the ASH is as detailed in Figure 2.10:3a. The highway cabling is split at each terminal unit, or highway controller, and the 'T' connection is formed internally.

In addition to the 'T' configurations previously described, the specifications include details for a 'Spur Connection' which is as yet not implemented. This will consist of an active repeater which will act as a gateway between two serial highways, each up to 300 metres in length. This configuration is illustrated in Figure 2.10:3b.

2.11 Conclusion

The ASH has been designed with two principle aims, firstly that the system should be as reliable as possible and secondly that the host processor should have as little processing to do as possible in order to send messages on the highway. These have been satisfied by the provision of multiply redundant system elements such as cables and controllers, and by the choice of a memory table driven FEP/ host software interface. In addition, by careful choice of FEP hardware interface design it was possible to allow the units to be interfaced to a wide variety of computers with a minimum of hardware changes.

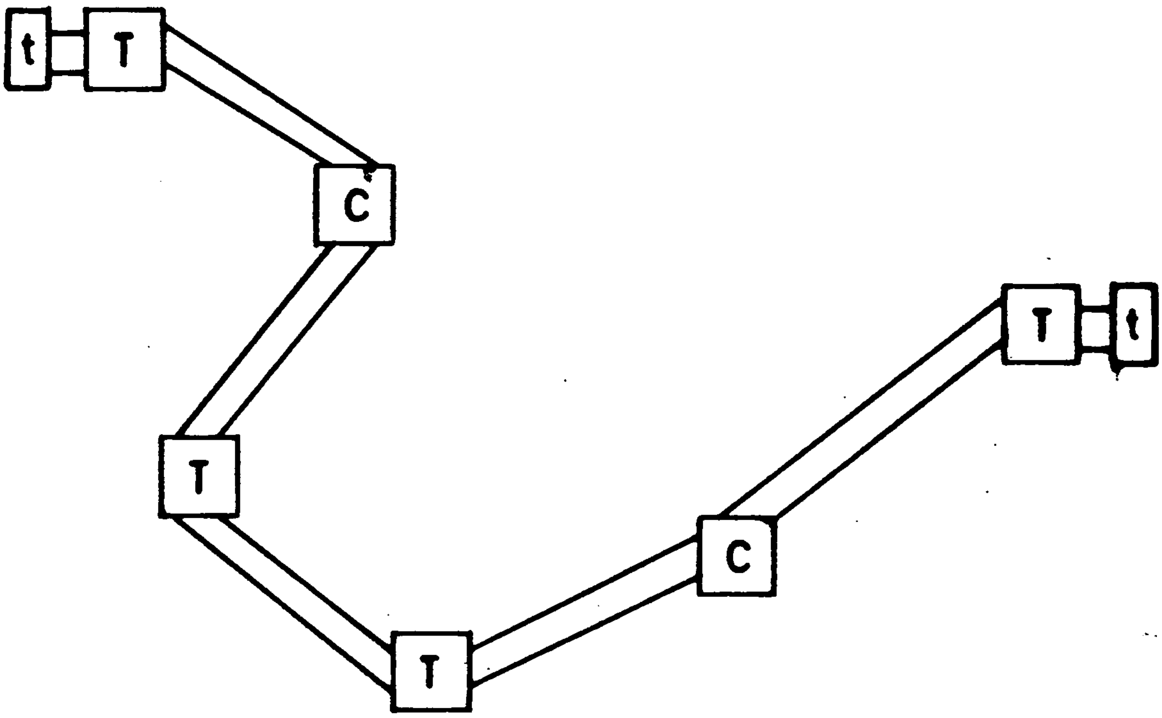


Figure 2.10:3a Actual ASH Configuration

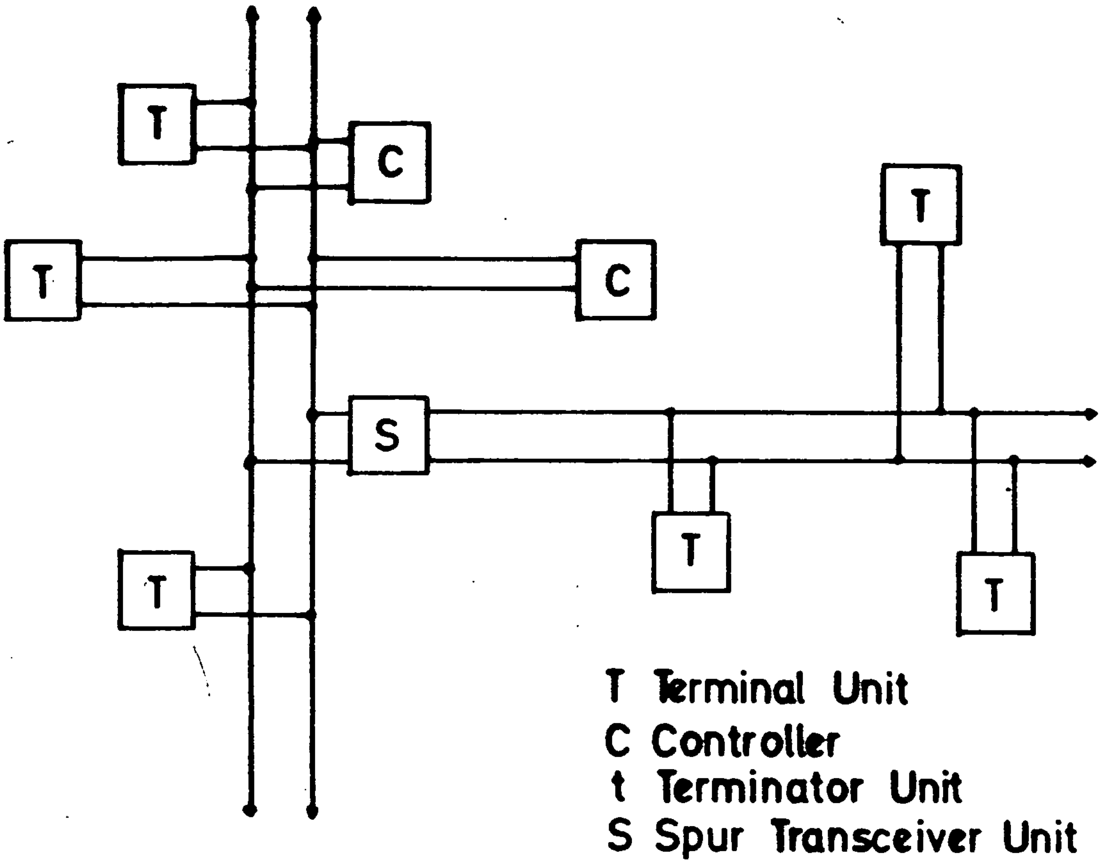


Figure 2.10:3b Spur ASH Configuration

Chapter 3

Computer Systems

3.1 Introduction

In the course of this research it was necessary to use a selection of different computer systems. Initially, a DEC PDP11/34 minicomputer was used to implement some high level support software. The machine was a Durham Computer Department general purpose machine, and operated with the UNIX operating system, which was written by Bell Laboratories in the U.S.A. It was necessary to implement certain programs on this machine which had previously been in use on a Ferranti Argus 700S minicomputer at A.S.W.E. In order to install this software it was necessary to first implement a Coral compiler on the DEC machine, since the great majority of Military software is written in Coral, and these packages were no exception.

In the course of writing software support for the Motorola MC68000 microprocessor, which was the principle microprocessor used in this project, a cross-assembler was produced on a Data General NOVA-3 which operated under RDOS. This machine was chosen for the task, in preference to the DEC, because it was more readily accessible and was used less. Also used in the development of the MC68000 software was a Motorola MC6809 development system using the SSB (Smoke Signal Broadcasting) DOS69 operating system. This was used to a greater extent as the project progressed as all of the software written in SIXTH for the MC68000s (Chapter 4) was written and edited on this system.

3.2 The DEC PDP11/34 and UNIX

Durham University Computing Department owns a DEC PDP11/34 which it maintains as a general service machine. It comprises a fully expanded system with 256kbytes of memory, two 5Mbyte front loading disk units (RK05), a single 10Mbyte top-loader, a dual eight inch floppy disk unit, a lineprinter, and approximately ten VDUs. It runs under UNIX version six [17], which was until recently the newest of Bell Laboratories UNIX operating systems (version seven is now available). UNIX is a user friendly operating system which has been long favoured in universities and has recently been more widely accepted. It offers programmers very easy software accessibility to system devices and files, and since most of the system software is written in a high level language known as 'c', it is considerably easier to understand than most other operating systems (which are normally written in assembler) making it ideal for teaching. It was decided that this machine should be used, firstly due to the ease of access, and secondly due to the ease of programming for file and device input/ output, since it was known from an early stage that it would be necessary to write such software.

3.2.1 The Implementation of BCPL and CORAL

As has been explained, due to the fact that the majority of Military software is written in Coral, it was necessary to install a Coral compiler at Durham. It was not possible to obtain a Coral compiler to run under the Unix operating system, so a more indirect method had to be followed.

A version of Coral [18,19], written in BCPL, and a BCPL compiler, were available. The BCPL compiler and the Coral compiler were designed to run under RT11 (DECs standard single user operating system for the PDP11 series), but a careful comparison of UNIX and RT11 showed that the main differences between them lay in the input/ output (I/O) system and in the assembler language used. However, UNIX already had an RT11 MACRO assembler [20] installed (MACRO is the standard RT11 assembler language) so it was only necessary to alter the I/O routines in order to transport the BCPL compiler to UNIX. This was because an assembler version of the BCPL compiler had been obtained, comprising the sections detailed in Figure 3.2:1a. The only operating system dependant section was the I/O section which was the compiler to operating system interface. It handled file and device I/O and memory allocation, and was identical to the one used in the Coral compiler. This assembler version of the BCPL compiler was a simple version, with just enough complexity to allow compilation of the version of BCPL written in BCPL. When this second, more complex version had been compiled, it was then possible to use this new compiler to compile the Coral compiler. This process is known as bootstrapping.

The difference between the two operating systems is the level at which the user interfaces to devices. In the case of UNIX, a device is handled in a similar manner to a file, whilst in the case of RT11, a device must be handled in a completely different manner, leading to a much more complex I/O system for the BCPL and CORAL compilers under RT11.

In the absence of any formalised test suite for the BCPL compiler, it was felt that a program of the complexity of the CORAL compiler would be a sufficient operational test. Similarly, the software packages written in CORAL and transported from the A.S.W.E. Ferranti

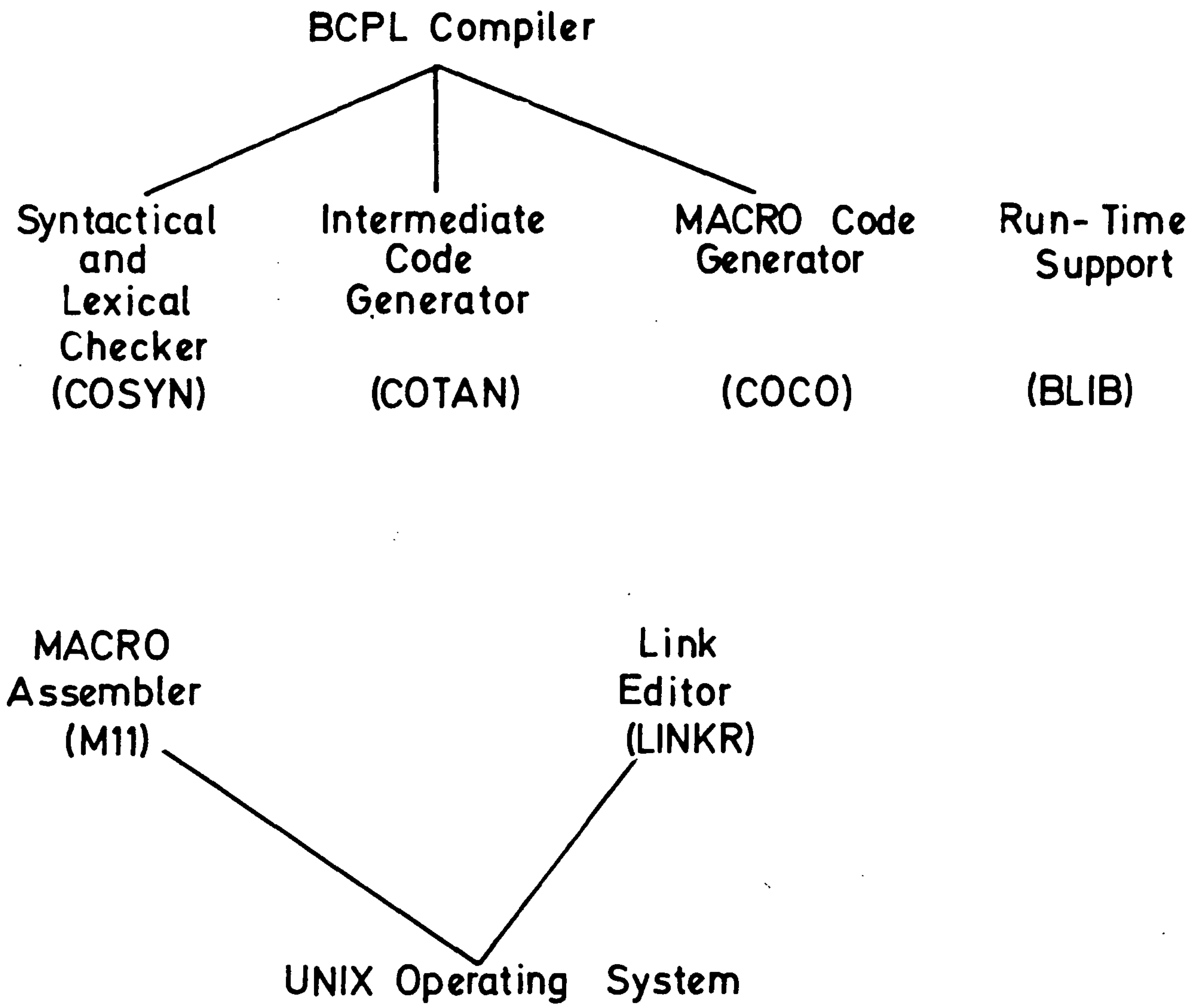


Figure 3.2:1a BCPL Compiler

Argus running CORAL were decided to be a sufficient operational test of the CORAL compiler.

3.2:2 ASH Software Packages

A.S.W.E. had commissioned the writing of two extensive software support packages for the development of the ASH software. The first was a cross-assembler for the microcode for the FEPs [14], and the second was a simulator [15] for the entire highway which was used to test the microcode before its installation in the actual system.

The cross-assembler was a two pass assembler which accepted as input a file of text and produced as output a binary file in a format suitable to be used to program a PROM programmer for the microcode PROMs, and a text file which contained a full assembler listing. The input file contained lines of instructions, each one of which corresponded to a word of microcode. Thus the maximum program length was 512 instructions, corresponding to the full depth of the microcode store. The hardware configuration of the FEPs, as detailed in section 2.4, includes sixteen internal ALU registers (named r0-F), sixteen external read-only registers (named f0-F), and sixteen external write-only registers (named t0-F). These registers may be assigned names using the assembler. This greatly increases the source program readability. The possible combinations of source, destination and internal registers with ALU instructions, are defined by the hardware design, and the program is checked against the allowable combinations by the cross-assembler. The combinations are detailed in Cambridge Consultants cross-assembler manual [14].

At ASWE this program was compiled in one section.

Unfortunately, owing to the inherent limitations of a DEC PDP11/34 and UNIX, the maximum size of a user program is limited to approximately 48kbytes of machine code. Due to the design of the CORAL compiler, the maximum size of a CORAL program which could be compiled in a single segment was approximately four hundred lines. This meant that the cross-assembler had to be segmented into three parts to allow it to be compiled. After this stage had been succesfully accomplished, the compiled program was linked with its run-time package (the same one as was used by the compilers) to produce the complete assembler.

The ASH simulator is a very much more complex program. It allows complete simulation of the operation of the highway system. As an input it accepted the file of microcode produced by the cross-assembler, and a list of the allocation of external ALU registers (i.e. 'FIFO read' register equals f6, 'skip if FIFO full' equals sD etc) representing the actual hardware configuration. Then, the configuration of the highway, i.e the number of terminals and controllers was input. Lastly, a set of monitor points were entered, which governed which simulated registers were printed out during the monitoring. Now the simulation could be started. The program simulated the exact function of the bit slice processors and all of the hardware, with the proviso that messages transmitted from a terminal or controller were merely 'injected' into the simulated FIFOs at all of the other units, i.e. the physical medium of the highway cabling and its transceivers was not simulated. It was possible either to simulate the highway operation in single step mode- corresponding to the execution of a single microcode word in each unit, or in run mode, where execution continued for a preset maximum number of microcode instructions. Alternatively, by using a comprehensive break point monitor, it was possible to cause a break from the run mode into normal monitoring mode at the occurrence of any condition which had previously been selected as a break point (or any combination of multiple conditions). A full macro-executive allowed complicated break, monitor and restart functions to be established.

Due to the extreme complexity of this program, and the fact that it was simulating several highway units at the same time, it was extremely slow when in operation (a simulation of approximately 100 microseconds of highway activity could take as much as half an hour). Also, the program was very large, and in order to compile it under the CORAL on the PDP11/34 it was necessary to segment it into eleven parts. In addition, much of the monitoring section of the program had to be rewritten to compensate for the difference between the Ferranti single user operating system, and UNIX. This was due to the fact that on the Ferranti machine the monitoring was performed on a printer which was used only for that purpose, whilst this was not possible under UNIX because the printer was a shared resource. To compensate for this, a spooling program was included which saved the output for later printing, and the monitoring could also be performed on the VDU.

3.3 The Data General Nova 3 and RDOS

The Department of Applied Physics owns a Nova-3 which runs RDOS- the standard Data General single user operating system [21,22]. Its configuration is as follows:- Nova-3 CPU, 64kbytes memory, 10Mbyte top loading disk unit, VDU, printer and twin eight inch floppy disk unit. RDOS supports a BASIC interpreter, PASCAL compiler and a Nova-3 assembler, as well as a SIXTH interpreter/compiler which was written at Durham.

This machine was used to write a cross-assembler for the Motorola MC68000, because it was initially impossible to obtain one from Motorola which would be suitable for use at Durham. The cross-assembler was written in Nova assembler, and did not include all of the assembler language options possible with the MC68000 due to the extreme complexity of the assembler program that would be necessary. Instead, a simple assembler was produced, to which additional assembler instructions were added when needed.

3.4 The Motorola MC6809 Development System

The Motorola MC6809 [23] is one of the most advanced of the eight bit microprocessors currently available. It is similar in architecture to the earlier MC6800, however it offers the major advantage of a much enlarged instruction set and an additional stack pointer. The additions to the instruction set consist of several new addressing modes, including indirect addressing, which greatly increases the programming flexibility available to the user.

The development system used at Durham University is based around a SWTPc (South-west Technical Products Corp.) CPU board which is housed in an MSI (Midwest Scientific Instruments) chassis [24]. It includes 48kbytes of RAM, a debug monitor (MON09), three serial ports and a triple five inch floppy disk drive unit with disk controller. The system runs under the operating system DOS69 [25], which is produced by SSB (Smoke Signal Broadcasting). This operating system offers all of the basic disk file utilities and a resident assembler and BASIC interpreter. It is an update of the earlier DOS68, written by SSB for the MC6800, and most of it was merely reassembled into MC6809 machine code from the original MC6800 assembler language source, since the MC6800 assembler mnemonics are a subset of those used on the MC6809. This shortcut taken by SSB to obtain an operating system for the MC6809 means that it is no more efficient than the earlier MC6800 version, since it does not take advantage of the additional facilities offered by the MC6809.

The development system has many additional unused connections, both to the main synchronous bus (SS-50C), and to a memory mapped I/O section (SS-30 bus). This allows the speedy addition of user built boards to the system.

3.5 The Motorola MC68000 Single Board Computer

Motorola produces a single board computer, the MEX68KDM [26] a diagram of which can be seen in Figure 3.5a. This microcomputer has an MC68000 as the processor, and many additional components enabling the board to be used in a wide variety of applications.

The MC68000 is Motorola's sixteen bit microprocessor [27,28]. Externally, it has twenty four address lines, sixteen data lines, and control lines for asynchronous and synchronous bus interfaces. Internally, it has a thirty-two bit architecture, in that it has seven

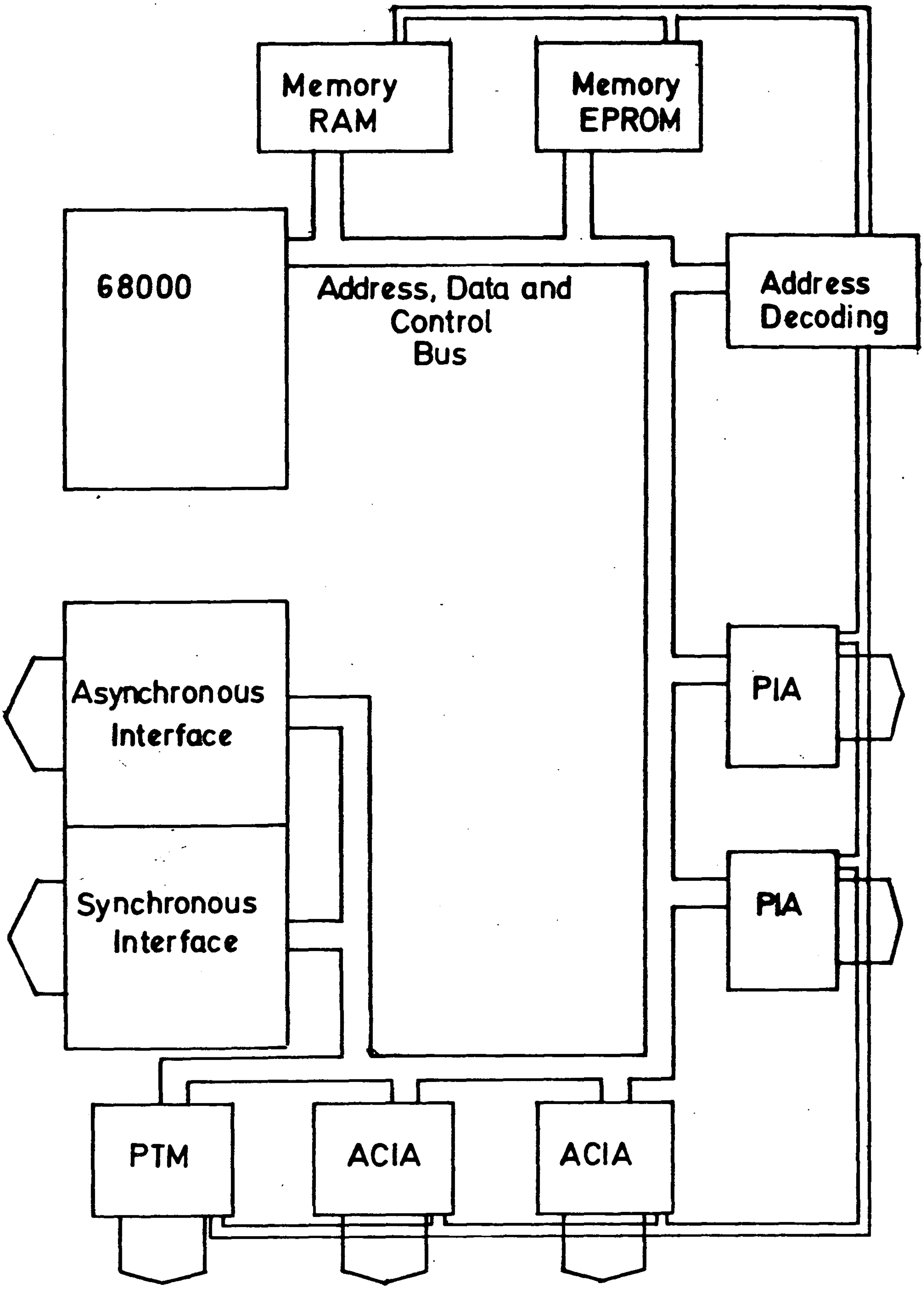


Figure 3.5a MEX68KDM Block Diagram

data registers, seven address registers and two stack pointers, all of which are thirty-two bits in length. The interrupt system is a multi-level one, having seven different priorities, selected by the use of three interrupt line connections to the processor. All but the highest priority interrupt may be masked out by the use of the appropriate instruction. The MC68000 also includes a bus arbitration section, which allows it to be used in multiprocessor systems with a shared bus. The design of the component allows it to be easily interfaced to all of the MC6800 family of peripheral chips [29], which only have an eight bit data bus, as well as the MC68000 family of peripherals which have a sixteen bit data bus (very few of these new peripheral chips are yet available).

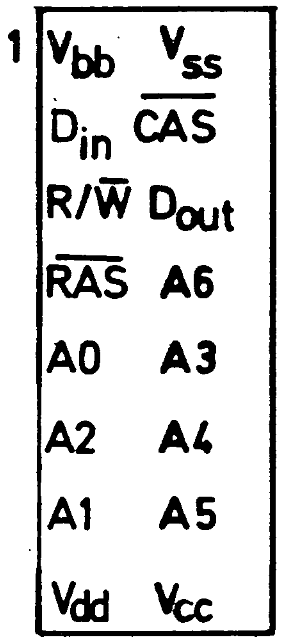
The single board computer also includes 32Kbytes of dynamic RAM, two parallel ports (MC6821), two serial ports (MC6850), a programmable timer (MC6840), a very powerful debug monitor called MACSBUG (held in four 16kbit EPROMS) and additional sockets for a further four EPROMS (which may be 16,32 or 64kbit devices). The monitor provides a full trace/ debug facility for user programs, using the 'trace' mode which is designed into the MC68000 chip. In addition it allows programs to be loaded into RAM using 'S-record' format from an external device via one of the serial ports. Motorola have defined the format of 'S-records' and these are used extensively to allow the serial transfer of data. Initially they were used in paper tape systems as they incorporate parity checks and record length checks.

In addition, the board is provided with external connections (via edge connectors) for the two parallel ports, the programmable timer inputs, two sets of RS-232 serial connections, and a full set of synchronous bus signals designed to be compatible with the EXORCISER development system (Motorola's MC6800 development system). Lastly, the board also has external asynchronous bus connections, allowing an external device to gain access to the on-board memory/ peripherals, or for the MC68000 to gain a similar access to an external devices memory/ peripherals.

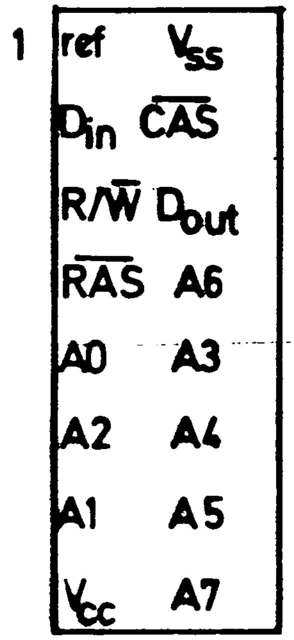
3.5:1 An Upgrade of the On-Board Memory

Although the MC68000 single board computer already included a RAM area of 32kbytes, it was found (section 5.2) to be necessary to expand this RAM area for certain applications. The normal method advocated by Motorola was to plug the MC68000 board into an EXORCISER chassis and plug in standard Motorola RAM modules as needed. However, a closer examination of the circuitry of the MC68000 board showed that with a very few hardware alterations it was possible to upgrade the board to 128kbytes of RAM (Appendix B). This was possible due to the similarity between 64Kbit RAM chips and 16Kbit RAM chips. The pinout of these chips can be seen in Figure 3.5:1a. The 64Kbit component has an additional address line, and has only a single supply rail, in contrast with the twin supply rails used on the 16Kbit device. This meant that the memory could be upgraded by merely adding a two-into-one multiplexor, and rewiring the supplies. Extensive testing of the expanded memory, using a psuedo-random memory test routine, showed it to function correctly.

In addition to the RAM expansion, by reprogramming the MACSBUG monitor into two 32Kbit EPROMs, space was made available for up to 24Kbytes of user program in EPROM. This was necessary due to the requirement (sections 5.1,6.1) that the user programs should be held in non-volatile storage to allow an ordered program restart after a power failure.



16Kbit RAM
(MCM4116)



64Kbit RAM
(HM4864)

Figure 3.5.1a

3.6 Additional Peripherals and Software

In addition to the computer systems already mentioned there were many additional pieces of equipment and software used, the most important of which are detailed below.

3.6:1 Pro-Log PROM Programmer

Many different types of programmable memories were used throughout the project. Firstly, in the FEP there were three different types of fusible link PROMs. In addition, on the MC68000 board, three different types of EPROM, and another type of fusible link PROM were used. This meant that it was necessary to use a 'multi-function' PROM programmer. This type of programmer is able to program a wide variety of different devices either by the use of multiple driving programs stored internally, or by the selection by the user of the correct 'pinout module' and 'configuration module' (this last was merely a PROM which held the correct driving program to be able to program the desired type of PROM). The first type of programmer is very expensive, and on this basis it was decided to purchase the second type. A Pro-Log M920 programmer was purchased, along with two 'Generic Family Modules', four 'pinout modules' and five 'configuration modules'. This enabled any one of the seven PROMs/ EPROMs to be programmed, and many others in addition. The programmer could either be connected to a teletype/ VDU via a serial link, or to a computer system via a parallel link.

For reasons of speed, it was decided to connect the programmer to the MC6809 development system via the parallel link and a parallel port on the development system. This port was in the form of a MC6821 PIA (Parallel Interface Adaptor) plugged into the I/O area mentioned in section 3.4.

A program was then written in assembler to control the programmer. A listing can be seen in Appendix A. This could be used to program the PROM with a microcode program already loaded into memory (by the operating system) and was also able to load the microcode programs into memory itself and subsequently to program the PROMs. Lastly, it was possible to use a 'modify' mode to perform a single location read or write into the PROM.

3.6.2 Computer Communications Software

To allow microcode which had been assembled on the DEC PDP11/34 to be programmed into the fusible link PROMs on the MC6809 system, it was necessary to write programs to allow communication between the two computers. They were separated by some considerable distance and the connection used between them was a 25mA current loop. The communications software composed the data to be transmitted into blocks of a preset length and included an error check field as part of the message to be transmitted. On reception at the MC6809 system, the message was checked for errors (by using the error check field) and if any error was detected the message was retransmitted by the DEC system.

A similar communications package was written for the transfer of data between the NOVA-3 and the MC6809. This allowed MC68000 programs which had been written and assembled on the NOVA-3 to be transferred to floppy disk on the MC6809, and subsequently either to be programmed into EPROM or down line loaded into the MC68000 boards. The program used on the MC6809 system was nearly identical in each case, the only difference being in the commands which had to be sent to the 'other end' of the communication link. The function of the programs on the PDP and NOVA was identical, however the DEC program was written in 'c' and the NOVA program in assembler.

3.7 DMA Interface

As detailed in section 2.6 communication between the host computer and the FEP was by means of a specialised interface. This provided for communication on two levels. Firstly a high speed DMA interface, and secondly an interface by which the host could control the FEP using some form of programmed output. At the start of this project, a version of the interface had been designed for a Ferranti Argus, a Locus 16 and a Konsberg S500. It was necessary to design a new interface for any other computer used at Durham. The first design produced was for a Motorola MC68000 interface. The circuit diagram can be seen in Appendix C. The new interface design differed from the original ASWE designs because it used more LSI parts, as these were not available at the time the ASWE interfaces were designed. This resulted in a decreased chip count and therefore a smaller overall size. The DMA section of this interface was designed to satisfy the timing requirements of both the MC68000 processor and of the FEP, as set down in their relevant specifications. Unfortunately, these specifications proved to differ from the actual physical attributes of the processors, and considerable time was taken in attempting to debug this interface. It makes use of the bus arbitration section of the MC68000 processor by requesting access to the asynchronous bus on the MC68000 board when the FEP indicates that it wishes to perform a memory transfer. This causes the MC68000 to halt at the end of the bus cycle currently being executed, and to pass control to the interface. The interface then completes the FEP's transfer and returns control to the MC68000.

The program control section of the interface is accomplished by partially decoding the address bus of the MC68000. If a write operation by the MC68000 is detected to an address preset by a set of switches on the interface, then the data being written to the address is decoded and latched to obtain the control signals for the FEP board. The signals are specified in section 2.6.

3.8 Conclusion

Several different computer systems were used, each for the task to which it was best suited, or most readily available. The DEC PDP11/34 was used primarily for 'high-level' program development, whilst the Nova-3 was used because access to it was virtually without restriction. The Nova-3 was used only to write assembler programs for the MC68000 system. The MC6809 development system was used for a wide variety of purposes:- to write SIXTH programs for the MC68000, to drive the PROM programmer, and as a monitor station and bulk storage unit (section 6.2:1). Lastly, the MC68000 was the workhorse of the project, being used in all of the ASH units built at Durham. In certain applications, its normal quota of 32Kbytes of on-board RAM was expanded to 128Kbytes. The design of the board gave great ease of interfacing to external sytems, either under programmed I/O via parallel or serial ports, or under DMA control via a synchronous or asynchronous bus. An asynchronous interface was designed to connect the MC68000 boards to the ASH front end processors.

Chapter 4
SIXTH

4.1 Introduction

FORTH was written by C.H. Moore at Palo Alto Laboratory [30,31] to run on a DEC PDP 8. It was designed to allow a large number of tasks to be simultaneously memory resident. At the time at which it was written, memory was very expensive and the PDP 8 had only 12k words of RAM. FORTH optimised its use of memory at the expense of execution speed, to gain maximum possible resource utilisation. The PDP 8 was used to control and monitor a radio telescope. FORTH creates an environment in which complex interface driving software can be easily debugged and tested and so was well suited to that application.

SIXTH is a second generation FORTH which was designed at Durham University specifically for use with modern microcomputer systems. In these systems, memory is readily and cheaply available, as are a wide variety of microcomputer systems, and SIXTH has therefore been optimised for ease of implementation and transportation, rather than for super-efficient memory use. The SIXTH used on the MC68000 systems was specifically designed and written for this research project. Similar SIXTH systems have been written for a number of different systems, both minicomputer and microcomputer based.

4.2 SIXTH Design Philosophy

SIXTH is a stack based language. This gives rise to a notation throughout the system which is primarily 'backwards'. This includes reverse Polish notation for arithmetic functions. In addition, language constructs which are used in other languages (such as IF... THEN... ELSE....) appear in slightly strange format (i.e IF... ELSE.... THEN...).

The system uses SIXTH language words, which are known as definitions. These are held as a 'linked list' in memory. This list is known as the dictionary. Each definition is preceded by a header whose format is shown in Figure 4.2a. It consists; the name of the definition as a string truncated to the first four characters, the total length of the name string and a link address to the previous definition heading. The operating system maintains a pointer to the last definition in the dictionary. To search for a definition in the list, it is searched backwards by starting at the last definition (to which the system holds a pointer). If this is not the desired definition, the pointer to the next one is extracted from its header and the previous definition is then checked. This will continue until the last definition in the linked list (which is the first routine defined in the kernel) is reached. This last definition has its link address pointer set to zero. SIXTH recognises this as being the last definition, and if this stage is reached it implies that the definition was not in the dictionary. SIXTH normally operates from a VDU in interpretative mode. Alternatively it may operate from a file held on some bulk storage medium to 'RELOAD' large sections of program which would be too laborious to retype from the VDU.

In its interpretive mode, character entry is handled on a line-by-line basis by a buffer routine which also provides keyboard handling (i.e. backspace, prompts etc.). The line is parsed into character strings separated by spaces and terminated by a carriage return. SIXTH then attempts to interpret these strings as either numbers or previously defined SIXTH routines. If the string is a number, it is placed on the operand stack, otherwise the string is checked with the dictionary to see if it has been previously defined. If it matches a previous definition, this definition is executed. If it does not, an error is flagged. From this interpretive mode, new definitions may be

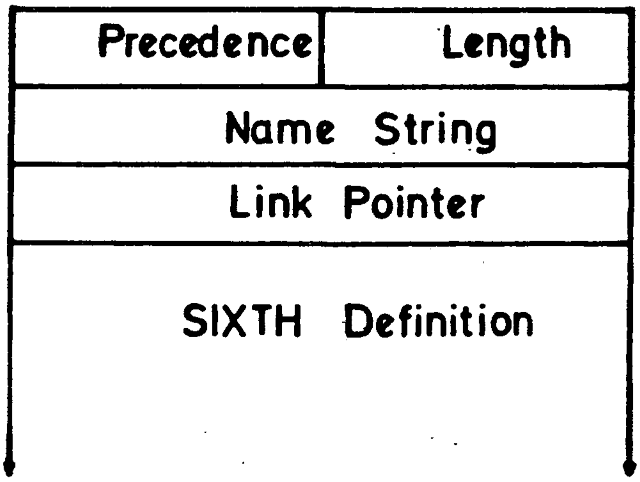


Figure 4.2a SIXTH Header Format

added to the dictionary by placing SIXTH into compile mode (using the definition ':' to start compilation and ';' to terminate it). In compile mode, any valid keyboard entry is compiled onto the end of the dictionary in the same format as previous definitions. After exit from the compile mode, this new definition may be executed from the keyboard in the same way as the old definitions. Thus any definition may either be executed from the keyboard or from within a definition, or may be compiled into a new definition. Additional SIXTH keywords cause the system to accept input from an alternative source to the keyboard. In some systems this would be a resident disk unit, but in the MC68000 system it was the serial link with the MC6809 development system. This allows the loading of the linked dictionary from the bulk storage device.

A normal SIXTH system comprises three parts; firstly the system kernel which is written in assembler, and incorporates all of the basic system routines such as I/O routines, terminal handler, interpreter/ compiler and number handler. The second part is the system dictionary which includes all of the high level routines such as conditional statements, loops, string handling, system utilities, and assembler if included in the implementation. The assembler was not included in the MC68000 implementation of SIXTH because of its great complexity. The final part of SIXTH is the user program section. The second and third sections are written in SIXTH.

The system as implemented on the MC68000 includes three stacks, the operand stack, the 'DO' stack and the machine stack. The first is the primary SIXTH stack which is used for parameter passing between routines and for keyboard interpreting. The second is only used in the 'DO..... LOOP' construct, and is used to stack loop parameters. The final stack is used to retain the return addresses from subroutine calls. The MC68000 processor has seven data registers and seven address registers. The stacks are implemented using three of the address registers.

4.3 System Kernel

As explained, the system kernel was written in assembler and performed all of the basic SIXTH functions. A listing can be found in Appendix A. In all of the routines apart from the first two (CHIN, CHOUT) all parameters are passed on the operand stack, allowing all of these routines to be used by any SIXTH programs.

In the explanations of the routines that follow, the abbreviations 'D0-7' are used to represent data registers zero to seven, 'A0-7' for address registers zero to seven, 'MS' for machine stack, 'OS' for operand stack and 'DS' for the 'DO' stack. The current system I/O port may be either of the two ACIAs on the MC68000 board, and is selected by the value of the address in the location named 'PORT'.

CHIN Reads one character from the system port into D0.

CHOUT Writes one character from D0 to the system port.

BUFFER Used when A VDU is connected to the system port. It prompts the operator for characters and then buffers a line which is terminated by a carriage return. It also sets up the parameters for **WORD.**

CRLF Sends a carriage return/ linefeed pair to the system port.

WORD Parses the line buffer produced by **BUFFER** to set pointers to the beginning and end of the next word in the buffer. Sets the **LAST** flag if the end of the line has been reached.

FIND This is the dictionary search routine. It takes the word parsed by **WORD** and searches the dictionary for it. The address of the definition is placed on the OS if it was found, or zero is placed on the stack if it was not found.

PUSH Pushes the contents of D0 onto the OS.

POP Pops the top word of the OS into D0.

STK Pushes the contents of D0 onto the DS.

UNST Pops the top word of the DS into D0.

NUMBER This is the number crunching routine. It attempts to assemble the ascii word parsed by **WORD** into a binary number, in the base indicated by **RDX**. It then places this number on the OS.

RESTART This routine performs a system restart by resetting all of the SIXTH system variables.

: This routine puts SIXTH into compile mode. It also puts the new header onto the end of the linked list.

EXECUTE This routine is used after an attempt has been made to **FIND** the word. If the attempt succeeded, then the word is either executed or compiled, depending upon whether the system is in compile or execute mode. If the attempt failed, **EXECUTE** calls **NUMBER** in case the word is a number. If it is a number it is either left on the OS or compiled into the dictionary depending upon the machine state. Finally, if the word is neither, **EXECUTE** flags an error condition.

TYPE The length and address of a string to be output are taken from the OS and used to output the string to the system port.

TITLE Displays the SIXTH banner.

; This routine ends compile mode and finishes the new dictionary entry. Anything entered between : and ; is compiled into the dictionary.

CONSTANT Assembles a number into the dictionary from the OS as a complete new dictionary entry. When the new definition is executed, it will put this number onto the OS.

INTEGER Assembles a number from the OS into the definition currently being compiled. When this definition is subsequently executed, this number will be placed on to the OS.

VARIABLE When executed it allocates space for a variable in the new definition currently being compiled, in addition it inserts machine code into the current definition. When the new definition is subsequently executed, this machine code causes the address of the variable space to be placed on the OS.

LOAD Resets the current system port to be the second ACIA connected to the MC6809 system and reads in one line of text into the line buffer.

OPEN This routine sends a command to the MC6809 system via the second ACIA which causes the MC6809 to open the SIXTH dictionary file.

RELOAD This routine sets the reload flag, thus putting SIXTH into reload mode.

IMMEDIATE If this word is compiled into a new definition, it causes the precedence bit (which is part of the header of each definition) to be set to one. This means that when this definition is subsequently included in a new definition, it will be executed rather than compiled, as is normal.

TO Sends the character on the OS to the system port using **CHOUT.**

DISSECT Dissects a binary number on the OS into ASCII characters, and a which it replaces on the stack.

. Uses **DISSECT** to output a number in ASCII to the system port which corresponds to the number which was on the OS, expressed to the current base.

ASMB Puts a number from the OS into the dictionary.

@B, @W, @L These read a byte, a word or a long word from the address held on the OS and place the result on the OS.

!B, !W,!L These write a byte, word or a long word held on the OS into the address also held on the OS.

+, -, *, / These perform the relevant arithmetic operations on numbers already on the OS and place the result back on the OS.

LEFT This routine shifts the number on the OS left by the number of places held on the OS.

SWAP This swaps the top two numbers on the stack.

The interpret loop is the master routine for the SIXTH interpreter/ compiler. Its operation is illustrated in Figure 4.3a, and is fairly straightforward. It calls either BUFFER or **LOAD** depending upon the state of the reload flag. It then calls **WORD, FIND, EXECUTE** until such time as the end of the line is reached (which is indicated by the LAST flag, set by WORD), when it loops back again.

The use of SIXTH is best illustrated by example.

At the simplest level, the line:-

```
4 2 *
```

places two numbers on the OS, takes them both off the stack, multiplies them together and places the result back on the OS. The line:-

```
: SUM 4 2 * ;
```

compiles a new definition named 'SUM' onto the end of the dictionary, which may then be executed, by using the line:-

```
SUM
```

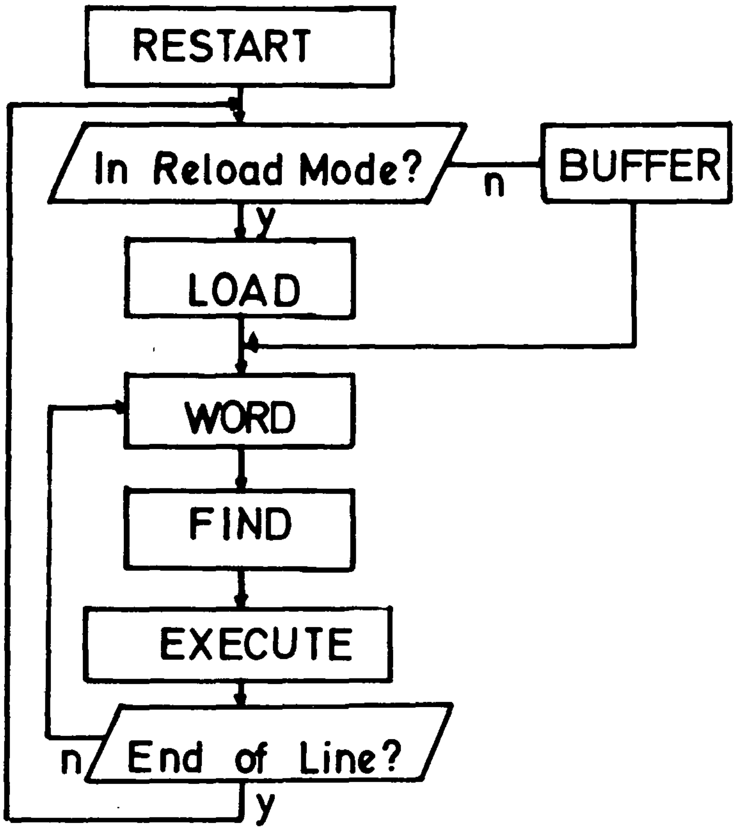


Figure 4.3a SIXTH Interpret Loop

This will perform the same operations as above. The line:-

10 CONSTANT FRED

creates a new definition which when executed places 10 on the OS. The line:-

10 : FRED INTEGER ;

has a similar effect. The memory manipulation commands are straightforward, the line:-

2 100 !W 300 @W

will store the number 2 at location 100, and will then read a word from address 300. The three operation types, byte, word and long word correspond to the modes for memory manipulation available on the MC68000 processor, which are 8,16 or 32 bits in length respectively. The lines:-

```
: THING VARIABLE ;
12 THING !W
THING @W
```

create a new definition called THING, stores the number twelve in the variable space thus allocated, and reads the contents of this variable space onto the OS.

Finally, the SIXTH dictionary may be reloaded by using the commands:-

```
OPEN
RELOAD
```

which open the SIXTH dictionary file on the MC6809 and reload the dictionary, by placing the system into the reload mode (the reload flag is set). The system is returned to normal operation by the inclusion of a 'RESTART' at the end of the dictionary file on the MC6809, which causes all of the SIXTH system variables to be reset, including the reload flag. SIXTH may be operated in any number base, as defined by the contents of the location 'RDX'. The base used only affects the ASCII representation of numbers at the VDU. The internal representation of numbers is always in binary form.

4.4 SIXTH Dictionary

As described, this is held on the MC6809 system in the form of an ASCII file which was written and edited using the standard DOS69 text editing system. A listing of the dictionary may be found in Appendix A. Although it was impossible to include a full 'in-line' MC68000 assembler in the dictionary, several assembler instructions were needed to complete the dictionary. These may be found near the beginning of the dictionary. It should also be noted that although a comment construct is defined using '(' and ')' as delimiters, this new definition occurs some way down the dictionary, so the initial section is uncommented. Several of the main higher level SIXTH constructs are detailed below.

.... DO .... LOOP

This is used in the normal manner, apart from the fact that it may used only when SIXTH is in compile mode, and the loop limits must be entered in reverse, due to the stack orientation of SIXTH. i.e. a valid statement would be:-

: TEST 10 0 DO I . LOOP ;

the 'I' routine places the current value of the loop counter on the OS. **STOP** may be used to abort the loop.

BEGIN .... END

This is the infinite loop construct and may be aborted by the use of **QUIT**.

... IF ... ELSE ... THEN ...

This is the conditional branch construct and may be used with the conditions =<, >=, =, >, <. A valid use would be as follows:-

: TEST < IF . DROP ELSE DROP . THEN ;

4 2 TEST

the definition TEST would then print out the lesser of two numbers on the stack, in this case it would print out '2'. Note that the construct may be used without ELSE, but IF and THEN must always be used together.

STRING

ARRAY allocates an array space within the current definition and also compiles machine code into the current deinition to handle this array space a run time. **STRING** merely fills this array with the specified string. e.g. : SAYING STRING "The quick brown fox" ;
creates an array filled with the specified string, whilst the command:-

SAYING TYPE

causes the size and length of this array to be placed on the OS, and TYPE then uses these parameters to type out the string. Several utilities are included, the more often used are:- **KEEP,** which is used to protect the dictionary against inadvertant deletion, **FORGET,** which is used to delete unwanted dictionary definitions and **WHAT,** which is used to list out the dictionary contents.

Finally, **COMPILE** is used to compile user programs from a file on the MC6809 which is specified by the operator thus:-

COMPILE FRED

This user file, which is held on the MC6809, must be terminated with a **%ENDFILE.**

4.5 Conclusion

A SIXTH interpreter/ compiler was written at Durham for the MC68000 single board computers which was capable of handling a VDU and accepting input, via a serial link, from the MC6809 development system which was used as a program development station. This allowed programs to be tested interactively from the VDU and then stored on floppy disk on the MC6809 system for subsequent recompilation. The SIXTH system was written in three parts, a kernel in assembler, and the dictionary and user programs in SIXTH. The design concepts differed from the FORTH language upon which SIXTH was based, in that memory use is no longer minimised at the expense of program execution speed. This was possible due to the current low cost and easy availability of semiconductor RAM as compared to the time at which FORTH was written.

Chapter 5

The Portable Highway Controller

5.1 Introduction

The portable highway controller was developed at Durham as part of the Durham University version of the ASH system [32]. As explained previously, the Ferranti Argus minicomputers used as hosts to the ASH system at ASWE could not be used at Durham, and instead Motorola MC68000 single board computers were used. Initially all software for both the highway controllers and the terminal units was stored in volatile memory on the MC68000 boards, and a system restart involved reloading all of the programs from the MC6809 development system. Subsequently, the programs were loaded into EPROM and a restart could take place without any intervention from the MC6809.

When the complete system was demonstrated to ASWE, it became apparent that an MC68000 hosted highway controller could perform all of the tasks which had been previously performed by a Ferranti Argus 700F minicomputer, at a much lower cost and in a much more compact form.

On this basis, a draft specification for a portable highway controller was set out whose main requirements were as follows:-

1) The unit should be entirely self-contained, should include some form of highway status display, and a keyboard of some sort to allow the operator to manually alter the highway controller operation.

2) The unit should be completely failsafe i.e. it should be able to restart automatically after a power failure, and important system parameters should be battery backed up, such as the system clock.

3) The operator should have as much (or more) control over highway controller operation as in the Ferranti Argus 700F system, and the display contained in the unit should provide all of the status information necessary for full monitoring of the network.

Several units have been developed at Durham which satisfy these requirements, and after extensive evaluation and testing of these units at ASWE, it has been decided to commission a commercial version for future use.

5.2 Portable Controller Hardware

The hardware used in the portable highway controller is an upgrade of that used in the standard Durham University terminal unit. The basic ASH hardware is as detailed in section 2.4-2.6, with the addition of an extra page (512 by 32) of microcode store. This is a standard ASWE alteration for the ASH systems which include dual controllers, as it was found to be impossible to include all of the necessary software in only one page of microcode store. The desired page is selected with the aid of a 'page register' which appears as a write only register to the 2901. In order to change page, the 2901 must write the desired page number into the register. Program execution will then continue at the same instruction address on the newly selected page.

The other hardware included in the unit was the Motorola MC68000 board with its on-board RAM expanded to 128kbytes (section 3.5:1). A CMOS clock chip (National Semiconductor MM58174) and 3V rechargeable battery were included to provide a battery backed-up system clock. The requirement for full monitoring capability was more difficult to satisfy in the limited space available. Several different types of display were looked into including plasma panels, LED displays and LCD displays. The LCD display was finally chosen on the grounds of availability, compactness, low cost and ease of connection. In addition, LCD displays need only a single 5V power supply, and the unit chosen, a FELTEC 128 character display (4 rows of 32 characters) was available in a low power version which had a power consumption of only 25mW [33]. The keyboard used in the Durham versions was a hexadecimal keypad (Radiospares) however ASWE intend to have a custom keypad designed for them.

Finally, the units contained a fully stabilised four rail power supply, which could run from 240,220,120,110 volts at 50/60Hz. The five volt rail was protected with a crowbar unit to provide the unit with some protection against power supply failure. The power supply design included mains filters and sufficient 'backing voltage' margins, to ensure that the power rails remained stable even when operating with ship-borne power supplies and their occasional lack of regulation. The external case was designed to enable the unit to be either free standing or to be mounted in a nineteen inch rack. A cooling fan and vents were provided at the rear so that the unit could be mounted in the middle of a stack of equipment.

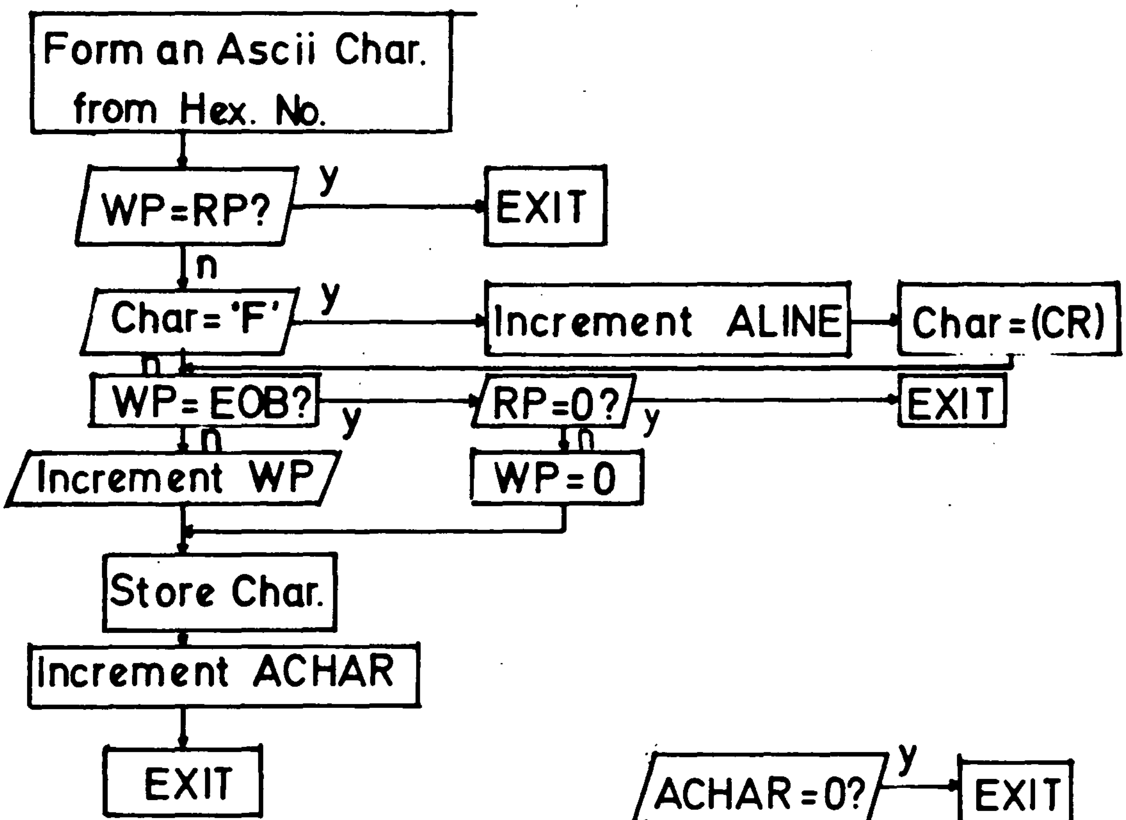
5.3 Controller Software

5.3:1 Design of SIXTH Programs

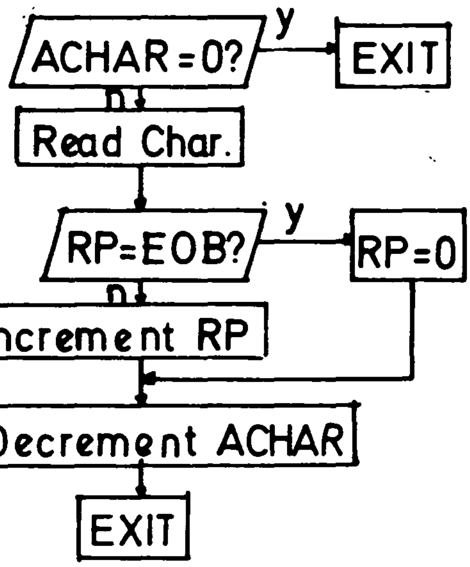
The highway controller software had to perform several separate types of functions which were as follows; firstly to allow the operator to control the function of the highway controller (and therefore of the highway system) via the keypad. Secondly, to provide continuous monitoring of any system feature which the operator wished to inspect (see section 5.3:2). Thirdly, the software had to provide a 'highway maintenance' function to automatically send out time messages, attempt to restart locked-out terminals, etc. Lastly, routines had to be included in the software to provide for orderly power-up restart of the controller and the highway system. The complete program listing may be seen in Appendix A. This SIXTH program is compiled onto the end of a standard SIXTH kernel and dictionary.

The first function is accomplished with an interrupt driven circular buffer routine to allow 'type-ahead' on the keypad (Figure 5.3:1a). Each operator command is a two character code, using the 'F' key as the equivalent of a 'carriage return' key. These commands fall into three groups (section 5.3:2), those that alter the status information which is being displayed, those which alter the operation of the controller, and those which alter the operation of the highway system. The codes generated by the hex keypad and associated circuitry are processed by the software to produce two character ascii commands which may be handled by the SIXTH buffer routine as though they had been originally entered in ascii. As mentioned 'F' is equivalent to 'carriage return' which is the SIXTH line terminator (section 4.3).

Interrupt Handler



WP - Write Pointer
RP - Read Pointer
EOB - End of Buffer



Programmed Character Read Routine

Figure 5.3:1a Keypad Service Routines

The first two types of commands may be carried out without any interaction with the highway controller FEP, however as explained previously (section 2.9:2), in order to alter highway operation, the controller must first be halted, and then restarted. An optimised routine was written for this function whose overall function is to swap the current polling table with an alternative one. The section of the routine executed with the controller halted has been minimised to keep within the time constraints mentioned in section 2.9:2.

The monitoring task is performed continuously, and is halted only when the operator is using the keypad (Figure 5.3:1b). At such times, the current monitoring cycle is completed, and then the display is cleared and used to echo characters to the operator and to prompt for the necessary entries. At the conclusion of that particular command entry, normal monitoring is resumed. The monitoring is normally in the form of a menu of controller status information taken from the various tables (section 2.9:1). This will only change should any of the various counters be updated by the FEP. The menu system is also used to display the terminal status information as extracted from the status tables, however the status information on multiple terminal units is displayed in rotation.

The highway maintenance function is invisible to the operator, and occurs at regular predefined intervals. The real-time clock chip is used to provide an interrupt once every five seconds. This prompts the MC68000 to set up a new time message in the 'out-time' space. The status of each terminal in the polling table is also checked, and should there be any terminals locked out, the MC68000 attempts to have them restarted, using the procedure detailed in section 2.9:2. Additionally, when the controller is passive instead of active, the interrupt routine will update the real-time clock, should an 'In-time'

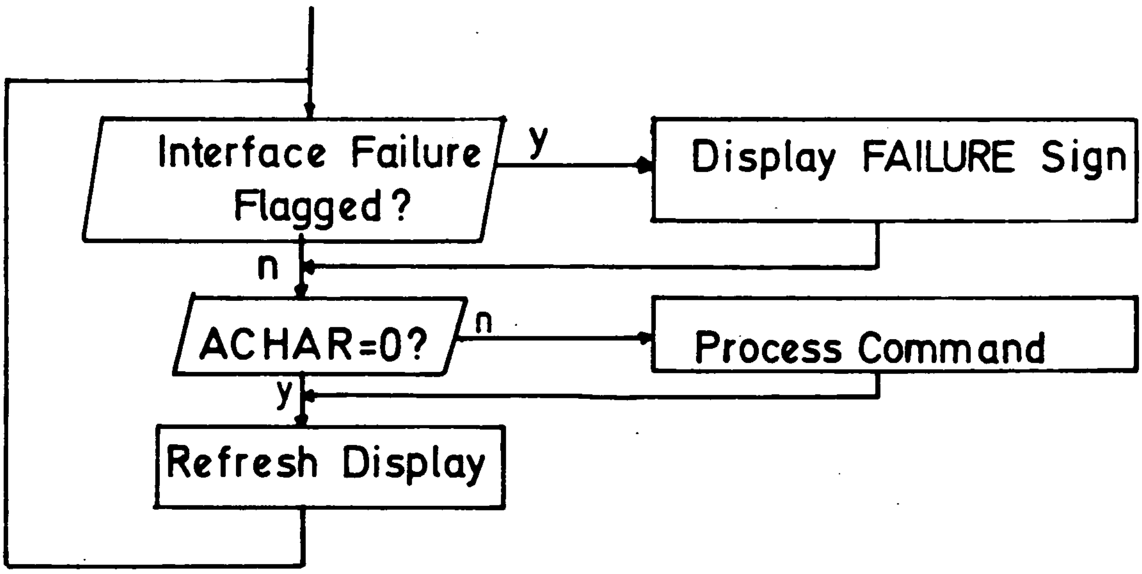


Figure 5.3:2b HWC Monitoring Routine

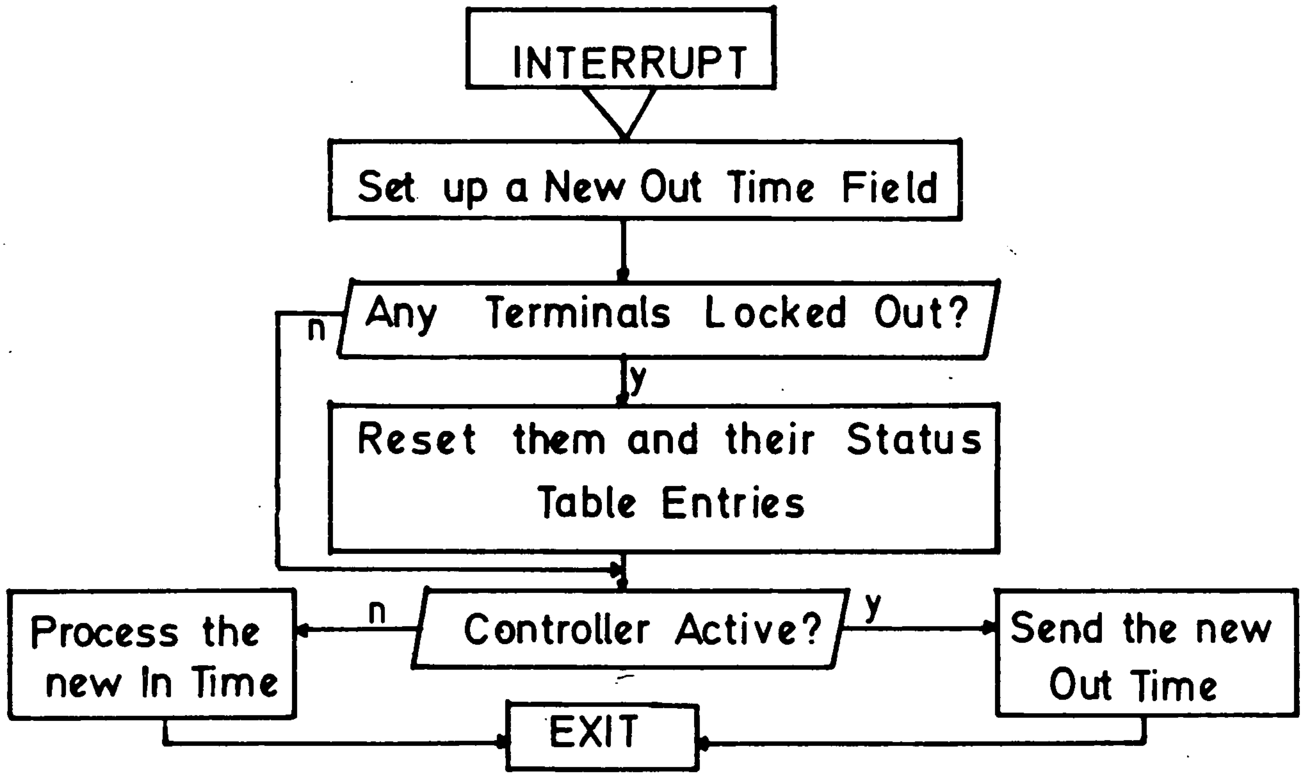


Figure 5.3:1b HWC Maintenance Routine

message have been recieved from the active controller.

The system chosen for the power-up software was as detailed in Figure 5.3:1c. A 'map' of the contents of the RAM immediately after loading SIXTH and compiling all of the controller programs was taken using the MACSBUG monitor. This map was transferred onto disk on the MC6809 system. In addition, a map of the SIXTH variable space was made, and a small supervisory program was assembled on the NOVA-3, and loaded onto disk on the MC6809. These various sections were programmed into the EPROMs as detailed in Figure 5.3:1d. At power-up, the MC68000 board generates a reset pulse. At reset, the MC68000 picks up the reset address from memory locations \$00000-\$00004, which are decoded into the EPROM address space. This address was set to point to the supervisory program, which then proceeded to copy the EPROM contents into the RAM space. After this copy was completed, control was passed to the SIXTH reset routine. This routine reset and halted the controller and set up the software tables to a predetermined default (currently to poll terminal numbers 0-16). The controller was then started, and control passed to the normal SIXTH program loop.

This method of restarting is very wasteful of memory space, because there are two copies of the entire SIXTH program when the unit is running correctly, and one copy is only ever used at reset time. An alternative would be to run the MC68000 from SIXTH programs stored in EPROM. This would give a great saving in memory, but would give rise to two other problems. Firstly, the access time for the EPROMs was considerably longer than for the RAM, thus programs would run slower than from RAM. Secondly, the SIXTH used on the MC68000 was designed to be run from RAM, and would have had to be considerably rewritten to allow it to be run from EPROM. Also, it was not relocatable and this would have led to addressing problems. The restart method chosen had

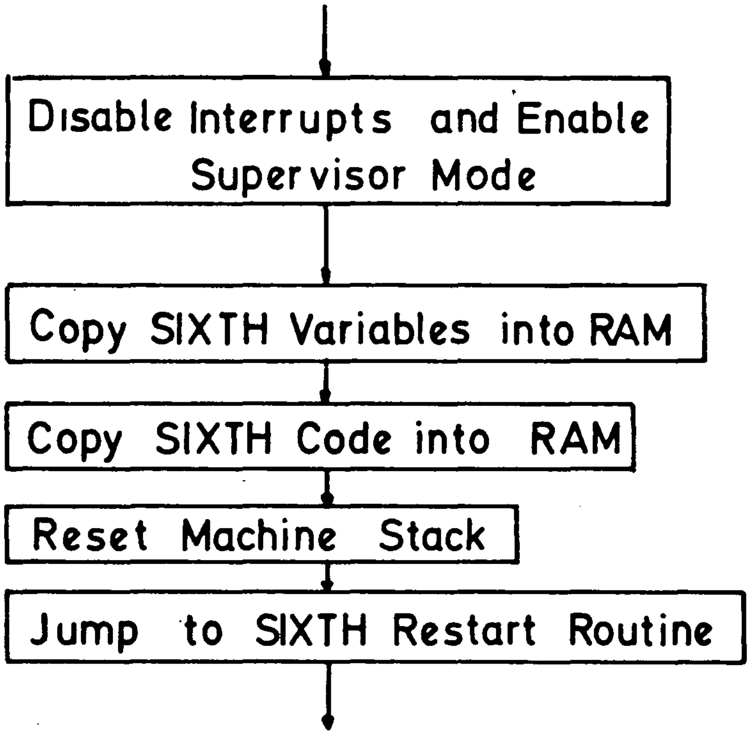


Figure 5.3:1c Power-up Software

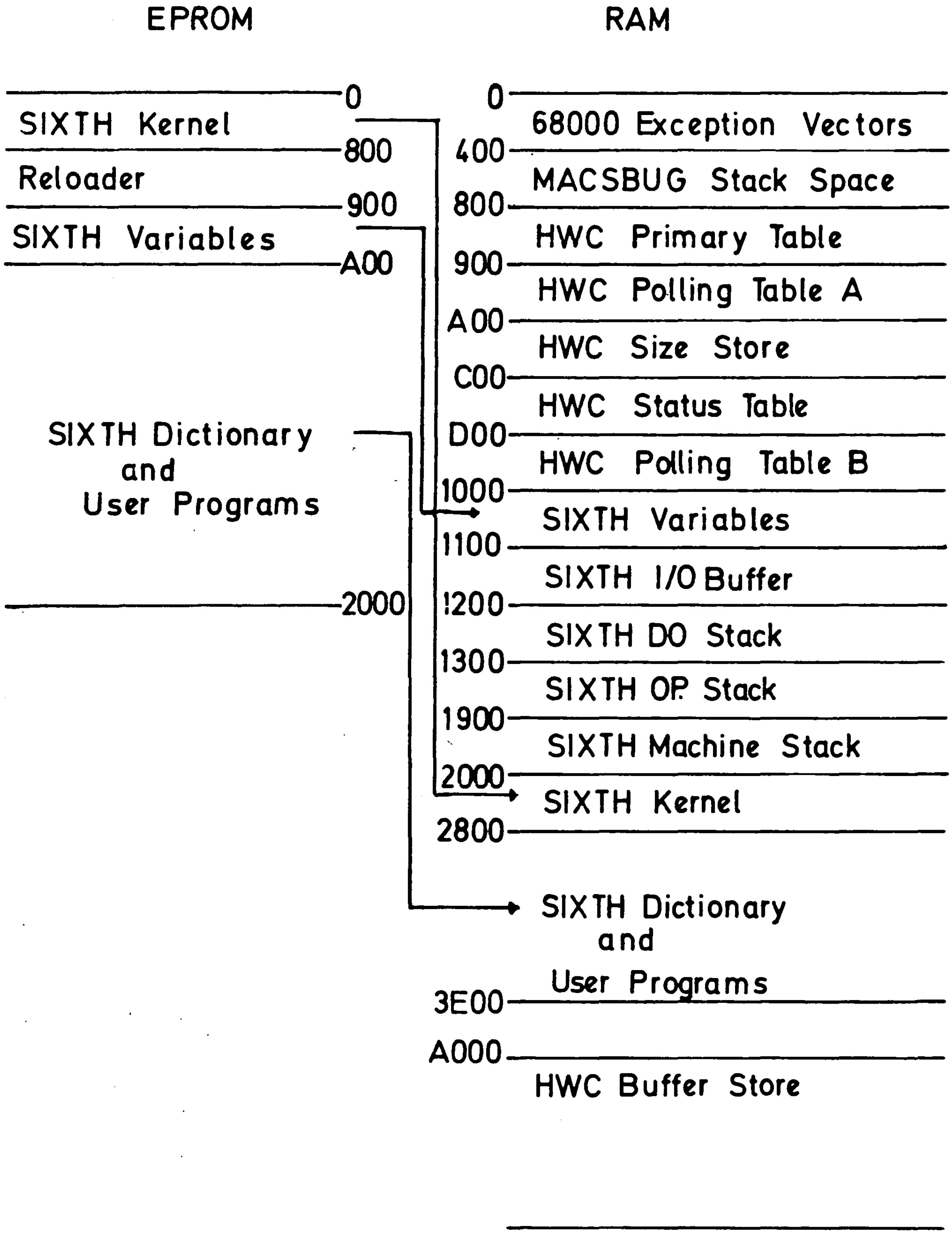


Figure 5.3:1d EPROM to RAM Map

the considerable advantage that to implement it, the standard MC68000 board needed only to have the restart vector changed (necessitating the changing of one EPROM) and to have the SIXTH EPROMs plugged in. In all other respects is was identical to the MC68000s in the terminal units, allowing complete interchangeability of hardware, and the great majority of software.

5.3:2 The Use and Upgrading of the Controller Software

The user commands are detailed in Appendix D. Unless the operator explicitly commands the controller's FEP to start, stop or reset (commands C1, C2, C3) it will continue to maintain normal highway signalling. In addition, the command 'AAA' causes termination of the normal mode of operation of the MC68000, and returns program control to the SIXTH interpret loop (section 4.3). This allows a VDU to be used as the I/O device. By this means, the operator may debug the software whilst the controller FEP is still in operation. Additional SIXTH routines may be added to those already present in RAM by compiling them from the MC6809 system or from the VDU. When these routines have been debugged and tested they may then be added to the normal controller program by reprogramming the EPROMs with the additional routines.

5.4 Testing the Portable Highway Controller

After the software for the controller had been written and debugged to the stage at which it would compile, it was necessary to devise some method of testing the many routines and the interactions between them. The routines concerned with monitoring the software tables and with altering the tables were tested interactively by causing SIXTH to respond to both the keypad and the VDU simultaneously. Thus the VDU could be used to check that the flat-panel display was correctly displaying the contents of the software tables, and that the keypad was performing the correct changes.

The sections of the user program which proved most difficult to debug were those which had to interact with the FEP and those which performed the restart function. The former routines had to be tested with the aid of a Logic Analyser (Hewlett-Packard 1615A) while the controller was connected to the complete ASH system, to ensure that none of the rigid time constraints were violated. The latter routines were difficult to debug due to the fact that in their final form they 'boostrap' a copy of the user program into RAM, and program control is passed to routines within this boostrapped program. Thus if there is any mistake in the copy section of the routines, a complete processor crash could occur. The debug monitor, MACSBUG, was used to debug these routines as far as possible, however when the stage was reached at which the user routines were performing (or attempting to perform) the entire restart, this was no longer possible because MACSBUG was not initialised after the power-up, and could not function correctly. The final debugging had to be carried out with the aid of the logic analyser.

After performing all the tests possible on the unit's stand-alone function, it was necessary to test it while connected to the highway system. These tests were first carried out at Durham, with the controller connected to a system comprising six terminal units, and then at ASWE, with the controller connected to a system comprising seven terminal units and an additional controller. The controller was tested whilst the highway systems were performing soak tests (see section 6.1). Terminal units were stopped and then started again, to ensure that the controller software was capable of automatically reseting terminal units. In the ASWE system, a second controller was used to check the operation of the portable controller in both active and passive modes. Extensive testing, over weeks of continuous use, necessitated minor alterations to the software which were mainly concerned with the MC68000/ operator software interface rather than the MC68000/ FEP software interface.

The final test of the portable controller was the ship trial (section 7) during which its operation in a hostile environment was fully tested.

5.6 Conclusion

A highway controller was designed and constructed at Durham as part of the ASH system built there. Interest was expressed by ASWE in the concept of a self-contained portable replacement for their Argus 700F based highway controllers. A set of requirements for such a unit was laid out, and a portable highway controller was designed at Durham to satisfy their requirements. The unit included a keypad and flat-panel liquid crystal display to allow operator control and monitoring. The software, which was written in SIXTH, performed all the functions necessary to supervise the FEP and to maintain correct highway operation. In addition, the unit was 'plug-in and go', in that it held the software in EPROM and could perform an auto-restart of itself and the highway system after a power failure. Extensive testing of the unit both at Durham and at ASWE has produced a proven design which may be manufactured in quantity.

Chapter 6

ASH Ship Trials

6.1 Introduction

The ASH is a highway system which is primarily intended for use on board ships in the late 1980s and 1990s. In addition it may be used as a high speed office LAN within certain MOD establishments. Although it had been extensively tested in a screened computer room environment within ASWE, it had never been tested on board a ship. This was due to the physical size of the ASH when based on a 'commercial' Ferranti Argus. Later systems will be based on the military Argus, which is a much smaller unit.

ASWE realised that it would be possible to test their LAN system using the Durham version, based around the small MC68000 single board computers. A test system using only the ASH for inter-system communication was proposed, because of the restrictions on access to several of the compartments in which the terminal units were placed. This system was designed and tested at Durham, and a monitoring unit based around the MC6809 development system was also included to provide a performance record on floppy disk, rather than on lineprinter paper, as was normal practice at ASWE.

The complete system was installed on board H.M.S. Londonderry, and ran continuously for six days, collecting some 3Mbytes of data concerning the operation of the highway. After the trials, extensive data analysis allowed a comparison between the operation of the highway as observed over the week on board the ship, with the operation of the highway as observed over similar periods at Durham and ASWE.

6.2 Test Hardware

6.2:1 MC6809 Monitoring Unit

Due to the inaccessibility of several of the highway terminal units and the difficulty in handling large amounts of computer printout in the small available space on board ship, it was decided at an early stage that the performance data collected from the test system should be in the form of records on floppy disk which could be printed out or analysed at a later date. A suite of monitoring programs (section 6.3:5) was incorporated into the software in each terminal unit. This software caused the reports from every terminal unit to be sent via the highway to one particular terminal unit, the master unit. This unit was situated in the Fixed Trials Office (F.T.O), and was accessible to the operators. It was connected to the MC6809 development system via a 9600 baud RS232 serial link, and software in the master unit and the MC6809 periodically updated the MC6809 disks. The hardware involved in the MC6809 system included the standard development system already described, with the addition of a serial port for connection to the master terminal unit. The MC6809 system was connected to a dedicated ships supply for the F.T.O. whose regulation was considerably better than that of the normal ships supply. This alternative supply was chosen because of the difficulty of providing adequate protection against disk corruption in the event of severe (but normal on standard ships supply) voltage fluctuations.

6.2:2 MC68000 Highway Terminal Units

An important consideration in the design and implementation of the hardware and software for the ASH ship trials, was that the software used in the terminal units which were not in the F.T.O. (slave units) should be held in non-volatile storage, and should have no need for local operator intervention. Thus the slave unit's hardware differed from that already described (section 3.5) only in the addition of four EPROMs which held the test software. The units were mounted securely to some part of the ships fittings to avoid damage in rough weather. The only connections necessary were to the ships standard 110volt/ 60Hz supply, and to the highway cabling. The layout of the units and cables in the test is shown diagramatically in Figure 6.2:2a. In addition, a small hand held battery V.D.U. (G.R. Electronics) was used during the initial installation, to check on the correct local operation of each unit, before they were tested using the ASH.

6.3 Ship Trial Software

6.3:1 Design Concept

The terminal unit software necessary for the ship trials had to perform three specific functions. Firstly, the operator had to be able to control the actions of the remote (slave) terminal units from the master terminal unit. This involved the remote starting and stopping of test software, and the resetting of tables etc. Secondly, the master unit had to perform as a monitor/ information gatherer for status and performance data being sent from all of the slave units,

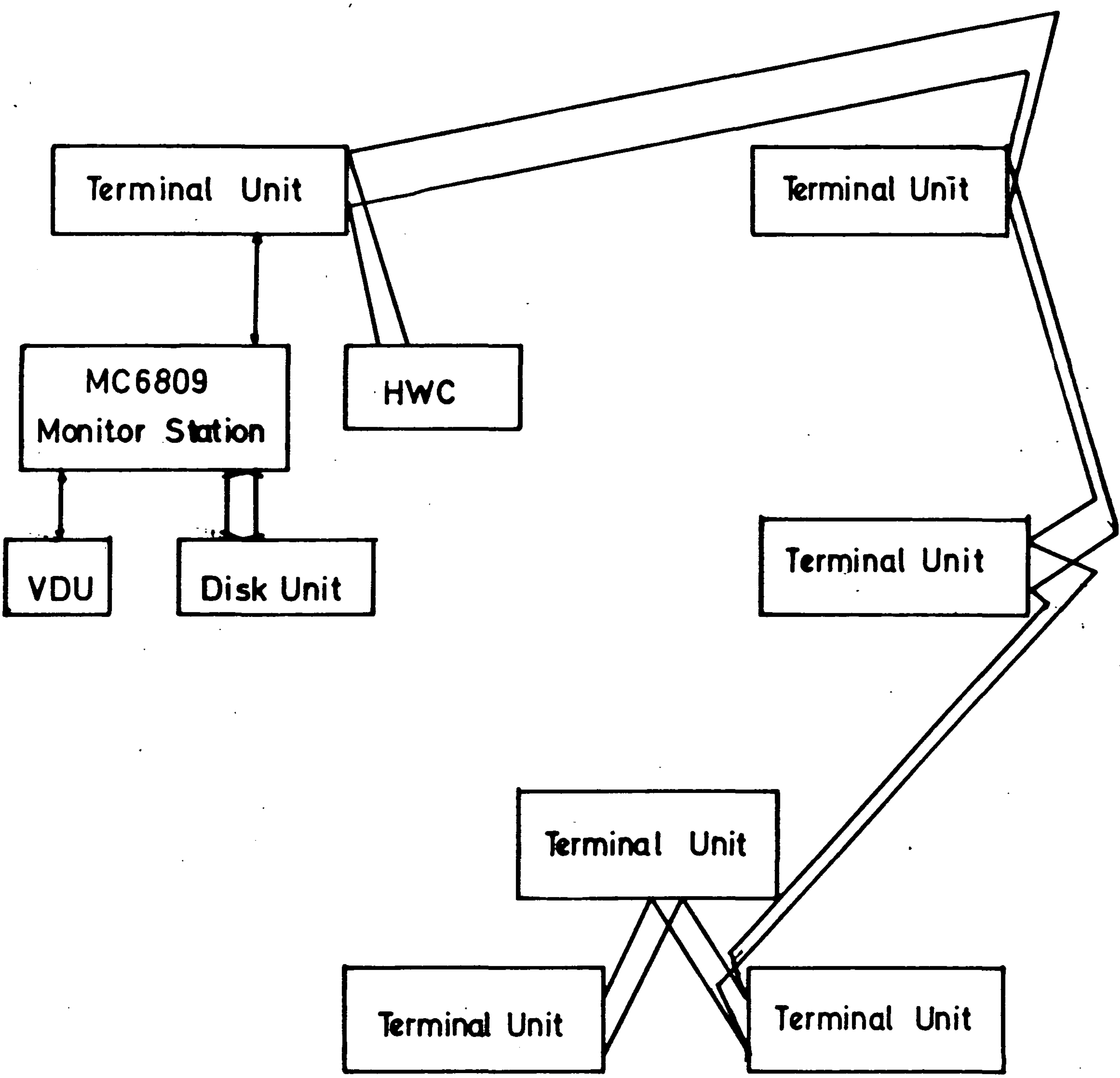


Figure 6.2:2a Schematic of ASH Ship Trial

and subsequently pass this data on to the MC6809 system which was acting as a bulk storage unit. Lastly, all of the units had to participate in soak tests of the highway, at the same time as the other two functions were being performed.

In addition to these functional requirements, the software suite had to be capable of restarting after a power failure in any of the slave units, and in the event of a highway signalling failure each slave unit had to be capable of returning the status of itself and its FEP to a level at which it could again receive messages from the highway. Full listings for the soak test software can be found in Appendix A.

This set of requirements necessitated some fairly complex programming, and meant that it was necessary to construct an operating environment in the MC68000 systems which was akin to that in a multi-tasking system. Indeed, at one point the design of such a system was considered as a possible solution to the programming problem, however time restraints and a long term hardware failure in the NOVA-3 (which would have had to be used to produce the new multi-tasking kernel) caused this approach to be abandoned. Instead, the multi-tasking environment was emulated with the aid of the multi-level interrupt system which is a feature of the MC68000 micro-processor.

6.3:2 Block Message Soak Test

The principle behind the Block Message Soak Test (BMST) was as follows. One terminal unit (in the case of these trials, the master unit) transmitted a broadcast block message of a predetermined length onto the highway. This message was composed of words of data which were cyclically generated from a stored generator word (Figure

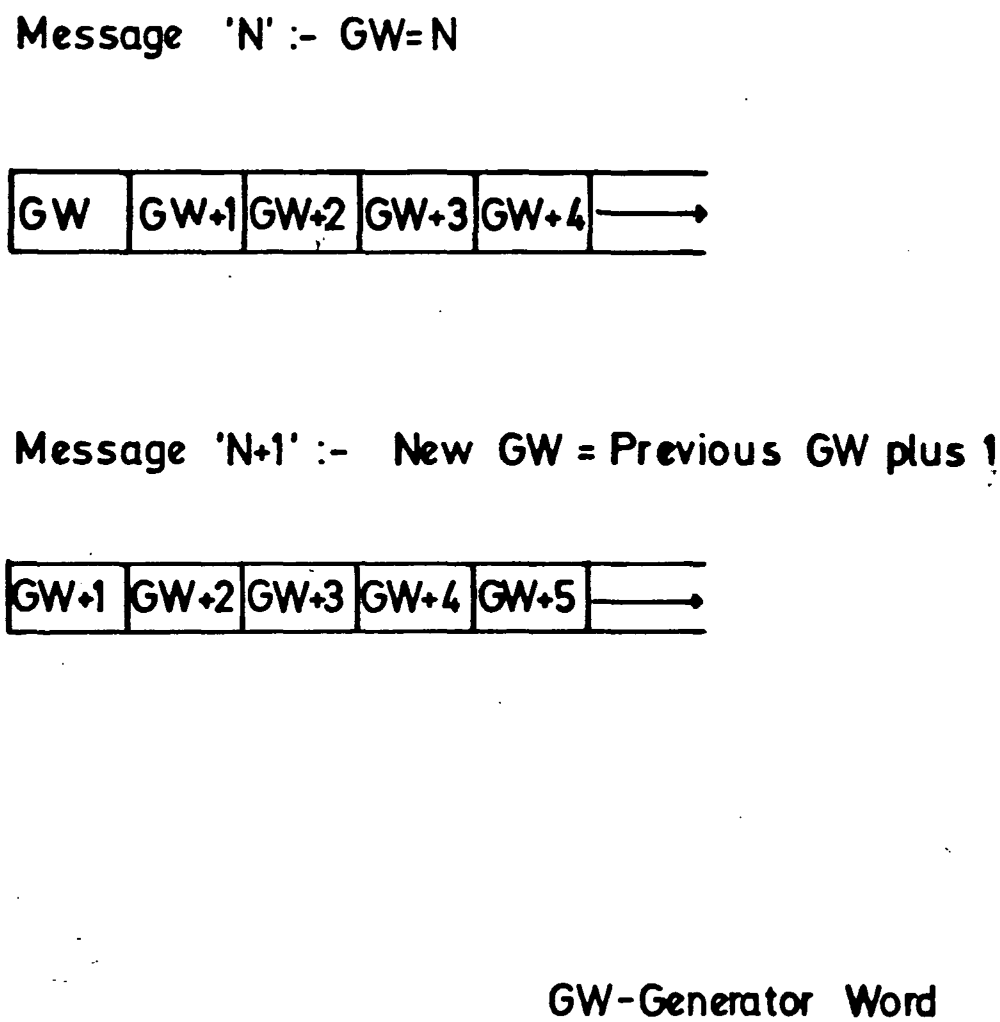


Figure 6.3.2a Block Message Generation

6.3:2a). Thus the first word in any transmitted test block was the generator word. This generator word was incremented by one after each block was transmitted. At the receiving units, in this case the slave units, the content of each received block was compared with the expected content, again by the use of a generator word. This allowed the receivers to check each word of each block for correct content. The generator word was initialised to zero at the start of the test, in both the receivers and the transmitter. Thereafter, the generator was updated only after reception of a message(at the receivers), or after transmission (at the transmitter). If a receiver detected a message out of sequence (e.g every data word was a constant value greater or less than expected) it would make a record of the fact, and store the first word of data in the out of sequence block as its new generator word, to get into synchronisation with the transmitter again. Also, if any error was detected, this was noted and a report transmitted (section 6.3:4). This type of test had been in common use at ASWE for a considerable length of time. However, at ASWE, the system used to determine block message transmission frequency is purely empirical, a transmission rate is chosen by the operator based on past experience of rates which are suitable. The slowest piece of processing in the test is that which occurs in the receiver when it analyses the received block of data. Thus if a transmission rate is chosen which is slightly too fast, an overrun will occur at the receiver. On occasions this overrun may take several hours to occur, and cause a BMST to be aborted after several hours of results have been collected.

As an alternative to this scheme, a system of handshaking between the slave units and the master unit was adopted for the sea trials. This system involved the use of control messages (section

6.3:4) issued by each slave unit after a block had been analysed, and the unit had set up the 'In Block' (section 2.3) fields ready to receive the next block. This system allowed the highest possible data throughput, with no risk of overrun at the receiving units. However, it did mean that the failure of one slave unit would cause the test to stop because it would no longer be transmitting its handshaking messages. The master unit waited for such a message from every slave before transmitting the next test block of data. Unfortunately, although this could be overcome by operator intervention at the master unit, the slave units would still be in the middle of a BMST and normal control messages sent via the highway would be ignored. To overcome this problem a timed restart sequence was implemented in each of the slave units to cause the BMST to be abandoned if there was no highway soak test activity for more than five minutes at a time. A flow diagram of the BMST can be seen in Figure 6.3:2b.

After extensive testing of the software, firstly in a single MC68000 system, and then on the complete highway system, it was decided that the sections of SIXTH program which generated and checked the test blocks of data could be usefully replaced by assembler routines, in order to speed the throughput of the test. Unfortunately, owing to the extreme complexity of the MC68000 assembler language, it was not possible to include an in-line assembler in the SIXTH system, (section 4.4) as is possible in other SIXTH systems. Instead, the assembler routines were written and assembled on the NOVA-3, and included in the SIXTH program as machine code. This alteration to the soak test improved the test throughput by an order of magnitude. The improvement was due to careful design of the assembler routines to avoid the inefficiency inherent in the use of subroutine threaded code.

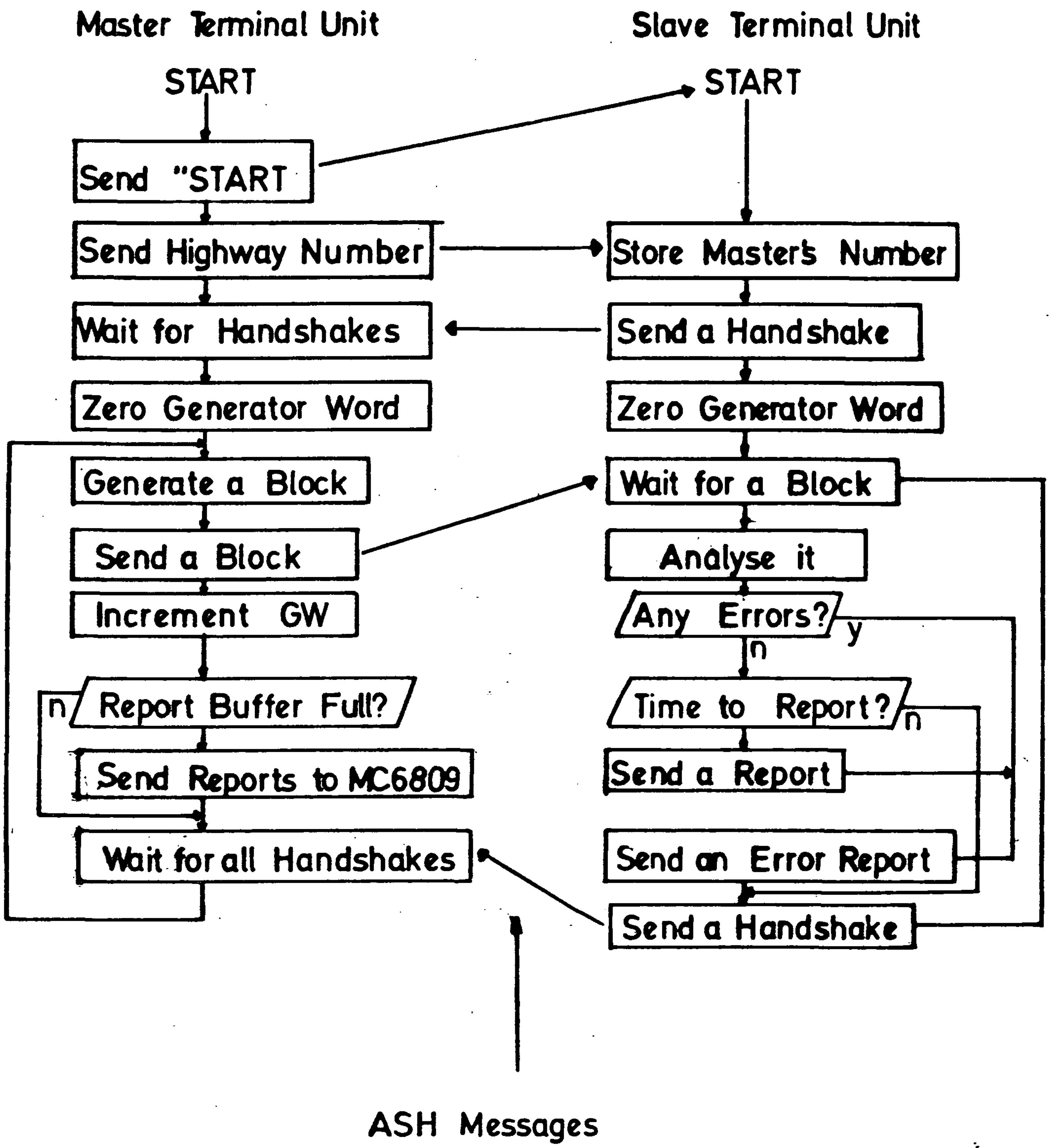


Figure 6.3:2b BMST Flow Diagram

6.3:3 Short Message Soak Test

The mechanism used to govern the frequency of the short message soak tests (SMST) at ASWE is again largely empirical. The operator specifies the transmission rate of test messages at each terminal in turn, and then instructs each unit to start the test. The latter operation is particularly ad hoc, since it is impossible to start all units simultaneously because the operator has to press a key on a VDU to start each unit and normally is unable to perform this operation on more than two units at a time. The problems encountered in the BMST concerning overrun also occur in the SMST. It was thus decided to use an entirely different system in the Durham SMST.

The requirements for a SMST are that every terminal in the test should transmit and receive messages to/ from every other terminal in the test. There should be no 'transmitter' as in the BMST, rather every unit should generate its own test messages. Two schemes are possible to perform this test. In the first, each unit transmits and receives broadcast block messages, and in the second each terminal transmits and receives point-to-point messages. In the first scheme handshaking would have to be performed in much the same way as for the BMST; i.e. a test message could not be transmitted unless a handshake message had been received from all of the units expected to receive the message. This could cause the same lockout problems as described in the BMST. Alternatively, the second method allows a considerably more elegant solution. If test message transmission is restricted to a point-to-point exchange with the unit from which a message has just been received then this overcomes the lockout problem. If a unit ceases to run the test then all that will happen is that no futher messages will be received from it by any other unit, and thus no further messages will be transitted to it by any of the units. In addition this solution makes more efficient use of the ASH since there is no necessity to transmit handshake messages.

Unfortunately, as with all elegant solutions, several difficulties were encountered with the second scheme, which was the one used in the Durham SMST. Firstly, the scheme used to maintain the generator word in the BMST would be very difficult to use because messages transmitted from a particular unit are no longer received by all other units, but by one unit only. Thus if there were five units in the test, unit 'A' would receive approximately one in four of the messages transmitted by unit 'B', and these messages need not necessarily be spaced apart by regular intervals of four messages. This meant that a different scheme was needed to inform the recipient of a test message of the value of the generator word. Fortunately, the ASH protocols include provision for a 'message type' word of length nine bits, allowing up to 512 different message types to be specified. The test control messages were using several of the message types between 0 and 255 (section 6.3:4) and the message types 256-511 were set aside to specify generator bytes, as opposed to generator words. This meant that the least significant eight bits of the MTB in a short message test message were initialised by the transmitter to the generator byte used, and were used by the receiver to check the content of the recieved message.

Another problem of the chosen SMST handshaking system was the increased complexity of the initial stages of the test. The complete block diagram of the test is illustrated in Figure 6.3:3a. Since each unit will only transmit a message to a terminal it has first received a message from, the startup section of test must perform two functions. Initially, every unit in the test must broadcast a message

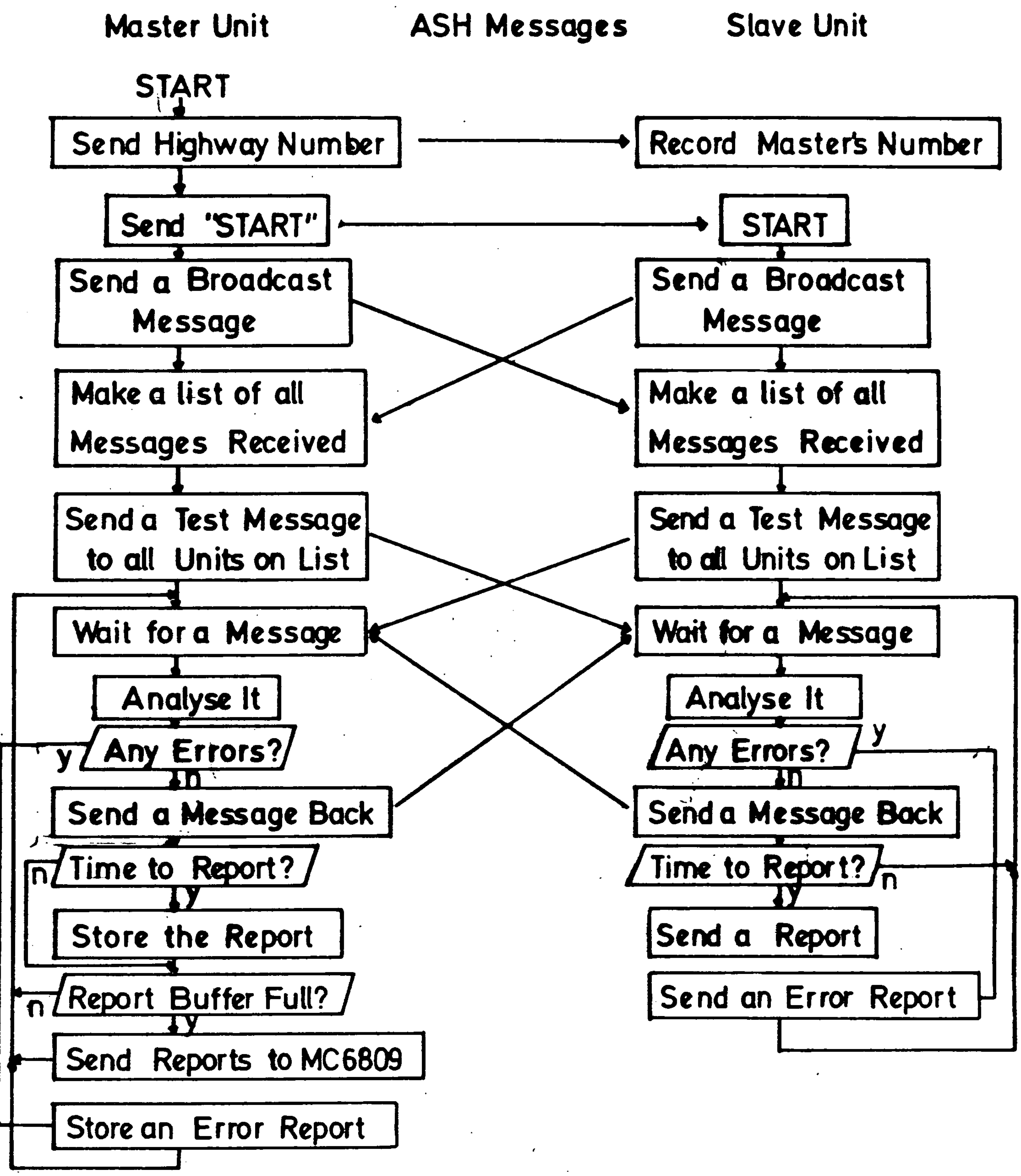


Figure 6.3:3a SMST Flow Diagram

to inform all of the other units that it is present and ready to participate in the test. Secondly, each unit must store a list of the terminals which broadcast to it in this manner, and subsequently issue one test message to each of the units in this list. After this has been performed the test may proceed as previously described, since all the terminals in the test should now have received one message from every other unit in the test.

A final problem with this test is that the timeout system used in the BMST will not function correctly, since if one unit stops the others will continue to operate. To overcome this problem an additional control message was added (section 6.3:4) which when transmitted by any unit on the highway caused all the other units to abort the SMST.

6.3:4 Test Control Software

As previously described, it was necessary to provide some means whereby a master unit could maintain control over the other slave units in the tests via the ASH. Several methods were tested, but the method finally chosen had the advantage of simplicity of programming over the other possibilities.

As described in section 4.3, SIXTH makes use of a line buffer which is normally updated from the VDU or from the routine used to perform a RELOAD. It was decided that the simplest possible method of 'remote' control by a master unit over the slave units would be to provide some mechanism in the slave's SIXTH program which would allow the master unit to send a SIXTH command line via the ASH which would then subsequently be interpreted in the normal (section 4.3) way by the SIXTH kernel. This routine consists of two parts, the routines which allow a command to be sent from the master unit and the routines which process the command in the slave unit. The former routine is very simple and merely sets up an output buffer in the FEP buffer space which contains a SIXTH command line. The routine at the receiver is much more complex, and uses an interrupt service routine, driven by the Programmeable Timer Unit (PTM) at intervals of one second. This interrupt service routine checks the state of the receiver's input buffers. Should a message have been received in the previous interval of one second the service routine determines whether or not it is a control message. This is determined by examination of the message type. Types 0-255 were defined to be available as control messages. Currently there are only three types defined. One of these types is used in the test handshaking scheme, the second in the status and error report scheme, and the third is used to pass control messages to the SIXTH interpreter. The reception of any one of these three valid control messages causes SIXTH to stack the current machine state and process the command message. Upon completion of this processing the machine's previous state is unstacked and execution continues from the point at which it was halted. Thus, as long as a user program does not mask out the PTM interrupts, this scheme will operate invisibly during execution of any program, or whilst SIXTH is awaiting commands from the VDU. Alternatively, the section of the routine which performs the checking and processing of the command messages may be explicitly executed by the user at any time.

Thus to start a test running in a slave unit, the master merely has to send a command via the ASH which is identical to the command that an operator would use to start the test (were there a VDU connected to the slave unit). Thus sending the command 'SSRUN' via the ASH would start the SMST, as would typing the command 'SSRUN' onto a

VDU connected to the slave unit.

Once started, the SMST disables interrupts and executes the command message processing routine periodically to check whether a relevant command has been sent. The abort command, which may be issued by the master unit (SABORT), sets a 'halt' flag in the slave's memory and this is also checked periodically. Should the flag be set, the test is abandoned.

6.3:5 Test Report Software

In order to monitor the activity of the highway during both the BMST and the SMST, and to gather any information concerning errors occurring during these tests, reports were issued by each terminal. These reports were received and buffered by the master terminal. A report buffer was maintained in the master unit's memory for each of the slave units. When any one of the buffers was more than 75% full, a message was sent by the master unit to the MC6809 development system, via an RS232 link, requesting the use of floppy disk storage. When a response was obtained from the MC6809 system the relevant buffer was transmitted down the RS232 link. Then the MC6809 system wrote it onto floppy disk. During the BMST, the master unit was not participating in the soak test, thus test reports were only issued from the slave units. However, during the SMST all of the units, including the master, were participating in the test and test reports were issued by all of the units. The report software was in three distinct sections; the issueing section (in all units), the receiving section (in the master unit) and the storage section (in the MC6809 system). Flow diagrams for each software section may be seen in Figure 6.3:5a.

The issueing section could issue two types of reports. The standard type, whose format may be seen in Figure 6.3:5b, and a special error report, whose format may be seen in Figure 6.3:5c. The standard report was issued periodically after a preset number of soak test cycles. It included information on the total number of errors detected by the FEP, the total number undetected by the FEP, the total number of messages received, and in the BMST, the number of messages receieved out of sequence. During the SMST separate counters were maintained for the number of messages received from each unit in the test, whereas only one such counter was used during the BMST because only one unit was transmitting.

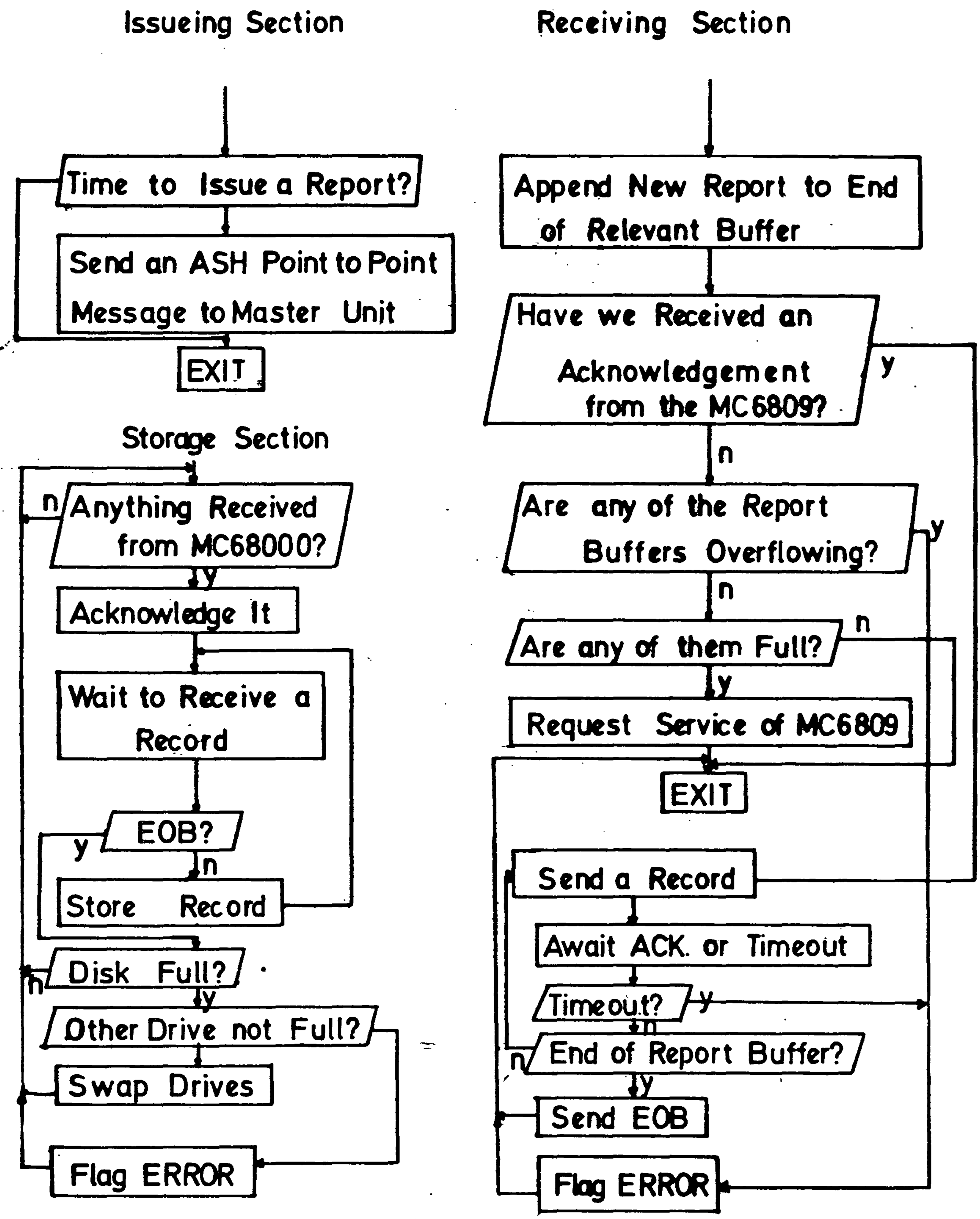
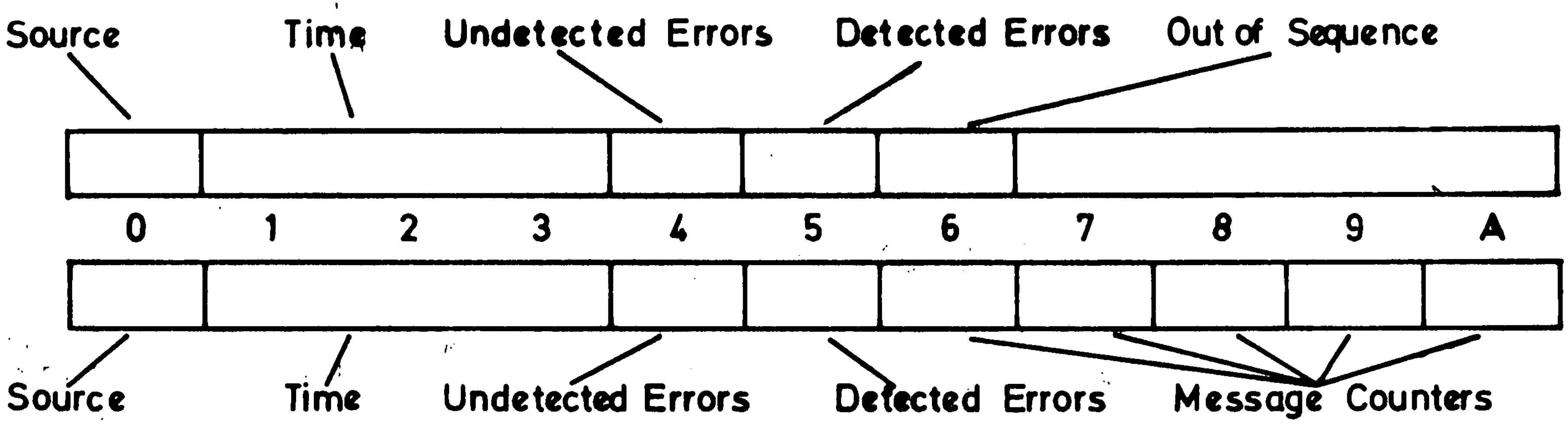


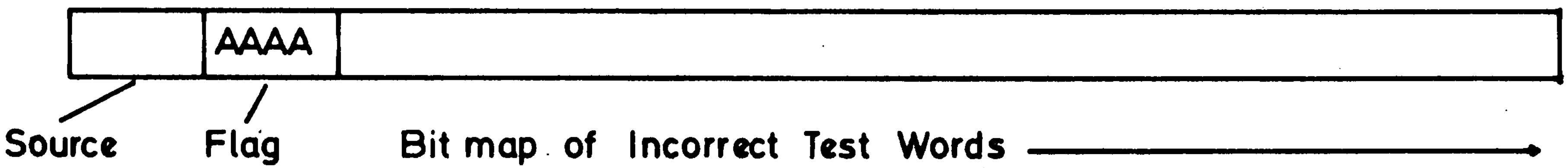
Figure 6.3:5a ASH Test Monitoring and Report Software

BMST Report Format



SMST Report Format

Figure 6.3.5b



Error Report Format

Figure 6.3:5c

The second report type was issued immediately after an error was detected by the soak test software which had not been detected by the FEP. The ASH should be a guaranteed error free message delivery system, thus if the soak test software found an undetected error it implied that there had been a breakdown in the error detection system. The error report consisted of a count of the number of errors which were detected by the software, and a map of the bits which had been in error in each of the incorrect bytes of received data. This bit map could be analysed at some other time to discover in what way the error detection scheme had broken down, and how it could be improved upon to eliminate such errors.

The second section of the report software, the receiving section, was part of the interrupt service routine described in section 6.3:4. Report messages had a 'message type' of 1. If the interrupt routine in the master unit detected a 'type 1' message it would then check the message to determine the source, and store the message at the end of the relevant buffer. A further routine, which was executed periodically during the test, checked the buffers, and if any of them were more than 75% full, initiated the section of the program which transferred the buffer of reports to the MC6809 and reset the buffer pointers. This section of program was very simple, it merely sent a request to the MC6809 system to be serviced. When this request was acknowledged, the reports in the relevant buffer were sent to the MC6809 one report at a time. After the MC6809 had processed each report it issued an acknowledgement which caused the master unit to transmit the next report. If the MC6809 hung up for any reason the master unit would eventually time-out and signal a buffer overrun error to the operator.

During the SMST, when the master unit was also issueing reports, these were entered directly into the relevant buffer in the master unit, rather than being sent on the ASH. That buffer was then treated in a similar manner to the slaves' buffers by the sections of program which checked for 'buffer full' and sent the reports to the MC6809 system.

The final piece of the report software was the section running on the MC6809 system. This program was written in assembler and performed three functions. Firstly it maintained the link with the master unit, waiting for any requests for communication to be issued. Secondly, when one of these requests was received it acknowledged it, and then proceeded to receive the report buffer as detailed above. During the reception of the buffers, they were stored in memory and after the entire buffer had been received they were written to disk, in order to keep the time which the master unit was 'communicating' with the MC6809 down to a minimum. This was necessary because during that time the master unit was no longer participating in the soak test. Finally, the MC6809 program performed monitoring and maintenance functions. The program checked the disks, and was able to swap to another disk unit when the previous one was full. If there was not an empty disk available, the program would flag the operator to change the disk. A small degree of monitoring of the reports being stored was also possible, in the form of a display of the most recently received reports from each of the units in the test. This could be called up by the operator from a VDU connected to the MC6809 system.

6.4 Test Results

6.4:1 Analysis Techniques

As previously described, the results from the sea trials were collected onto floppy disks at the MC6809 monitoring station. This process continued for the almost the entire week of the sea trials. The tests were only stopped in order to change between the block message and short message tests. This resulted in the collection of some 3Mbytes of data which had to be processed and analysed. In order to provide some control data for the experiment, the tests were also run in the laboratory at Durham University. Also, in order to have some data on the conditions in which the units were operating, the technical staff on board ship filled in detailed logs if there was any change in the status of electrical equipment, e.g. convertors or generators switched on or off. It had been suggested that the units which were operating in the more electrically 'noisy' environments, would be subject to a greater number of receive errors. For the purpose of the trials, the units were numbered as follows:-

0) FTO- Fixed Trials Office-Deck 1

1) CCR(H.P)- H.F. Transmitter- Deck 2

2) CRO- Radar- Deck 2

3) OPS- Deck 1

4) CMR- Conversion Machinery Room- Deck 3

The data which was collected during the sea trials was processed in two different ways. Firstly it was checked for the occurrence of undetected errors, and secondly for the occurrence of detected errors. Then graphs were plotted of the log error rate against the time for each terminal.

The task of analysis was performed by an MC6800 system which was running BASIC. The trial records were read in off the floppy disks with the aid of a small section of assembler code. Then the error rate was calculated over a certain integration period, which could be preset by the user. Finally, the MC6800 plotted the results on an HP flat-bed plotter.

6.4:2 Discussion of Results

With such an enormous amount of data to be analysed, it became immediately obvious that it would be impossible to plot graphs which covered the reports from all of terminals for the complete trial. Instead, graphs were plotted for a certain time period for all of the terminals in an attempt to relate their physical environment to the error rate which occurred at that terminal. Then it was hoped that some of the data collected on the ship machinery logs could be used to explain any fluctuations in error rates.

The first thing which was discovered was that no errors occurred which were undetected by the ASH hardware during the entire length of the trials. This meant that no further analysis of that particular type of error was necessary.

Next the detected error rate was analysed. A selection of graphs can be seen in Appendix E. Graphs 1-5 show an analysis of the log error rate for the first six hours of the trials. During this period the ship was preparing to leave port. Each point plotted on these graphs represents one minute of data. It can be seen at this point that there is a very close correlation between the error rates in graphs 1,3 & 4, whereas the graph for the terminal in the CMR room (graph 5, number (4)) appears quite different. This difference implies that the errors were induced directly into this terminal unit rather than onto the highway cable itself, otherwise the error rates would be identical at all of the terminal units. The physical positioning of this unit would support this theory, since the CMR was the only compartment on Deck 3 which had a terminal unit in it. It was definitely the most severe environment since it contained approximately eight high powered rotary convertors. The results detailed in graph 6 also support this theory. These are the error rates for the unit in the F.T.O. which was a shielded test office, with its own stabilised A.C. supply. As can be seen, the error rates for this unit are lower by more than an order of magnitude.

Additional series of results are shown in graphs 7-11, 12-16, and 17-21. These graphs all show a consistancy of error rates for the remote units of approximately 1 part in 10^5, and for unit 0 of between 1 part in 10^6 and 1 part in 10^7.

The conclusion which must be drawn from these results is that the highway cabling is virtually unaffected by the environment in which it is placed. Any fluctuation in the error rate between different terminal units is caused by the environment in which that particular unit is situated. This change may either be due to the quality of the supply to the unit, or to direct electromagnetic pickup within the unit. Also, after a comparison with the machinery logs, there appeared to be no direct correlation between changes in the state of the machinery and the error rates. The machinery in the C.M.R. was running continuously thus there were no changes in that compartment which would affect the error rate of that terminal unit.

As addditional evidence to support this conclusion, graph 22 presents remote terminal tests carried out in a control experiment at Durham. In this environment, it can be seen that the error rate is very similar to that measured in the F.T.O. on board the ship.

6.5 Conclusion

A software environment suitable for running tests on board a ship was designed and implemented. Hardware was constructed and installed aboard the ship in four remote compartments, and a test office. The highway was tested continously for a week, and a large volume of data was collected. After detailed analysis of the test results, two major conclusions were reached. Firstly, the protocols implemented in the ASH were capable of preventing any undetected errors being passed on to the computer system to which the terminal units were connected. Secondly, there was a level of background noise causing an error rate of approximately 1 part in 10^6, but depending upon the environment in which the terminal unit was placed, the error rate could increase by a factor of ten.

Based on these conclusions, it can be recommended that the exact source of this increased error rate is determined. Since great care had been taken in the design of the power supplies for the terminal units, and they had been tested in the laboratory under severe conditions of simulated supply fluctuation, it can be reasonably assumed that the increase in error rates was due to interference with the internal circuitry of the terminal units. If this could be proved to be the case, possible greater attention to screening of the unit as a whole, or certain sections of the circuitry in particular, might alleviate the problem.

Chapter 7

LAN Technology

7.1 Introduction

Many research centres are currently attempting to increase the performance of the basic types of LAN by the introduction of new techniques and the mingling of different LAN technologies. Each basic type of LAN has its advantages and disadvantages, and by careful redesign it is possible to reduce the disadvantages of each type to a minimum. The ASWE Serial Highway was designed after careful consideration of the network technologies available at that time. It has now reached a stage of development at which any advance in its performance may have to be achieved by a radical change in its design. It is possible that several of its most serious limitations may have been overcome elsewhere in the research being performed into LANs. Specifically, the areas which are of most interest are the necessity for centralised control, the survivability of the ASH after damage, and the system throughput under normal and abnormal loads and constraints.

However, in the case of the ASH a necessary constraint on any system modifications is that they should still conform as closely as possible to the specifications [11]. For example, although system throughput could be increased dramatically by a change in transmission media from a twisted pair to fibre optic cables, this would mean a radical and undesirable change to the specifications. Alternatively, the provision of a more flexible system of redundant controllers, or possibly the use of decentralised control, need not involve a radical change of specification.

A review of much of the work which has been performed on improving LAN performance has been carried out, and an attempt has been made to relate this to the current ASH. Suggestions are made for system redesign which attempt to conform as much as possible to the current specifications.

7.2 Review of Basic LAN Characteristics

The basic operation and characteristics of the common LAN architectures was discussed in section 1. The architectures fall approximately into two classes, ring and linear bus. The ring systems theoretically have the advantage of completely decentralised control, however their system of signal regeneration at every node, and the single ring cable normally used, mean that the system is vulnerable to the failure of single nodes or cables. The ring systems may be categorised into three types; the Pierce Loop, the Newhall Loop, and the Delay Insertion Loop.

A Pierce Loop consists of fixed length message time slots circulating around a loop, which fill the loop length. Examples of this type are the original Pierce Loop [34], and the Cambridge Ring [6]. A ring monitor/ control node must be included in this system to maintain the message slots. This type of system can accomodate multiple simultaneous users.

A Newhall Loop serves only one user at a time, who passes a 'bus available' token when its message transfer is complete. Examples of this are the original Newhall Loop [35], and the NPGS ring. Once agian, a master node must monitor the ring to ensure that a token is circulating.

Each node in a delay-insertion ring system contains two shift registers. One is permanently connected to the incoming signal, and the other is used to accomodate user messages. When a message has been placed in the second register by the user, the node awaits a clear space on the ring. When this occurs, the user message is clocked out onto the ring. If an incoming message should be received during the time the message is being clocked onto the ring, it is shifted into the first register, and clocked out onto the ring at the end of the user message. The nodes are responsible for the removal of their messages when they have circulated around the ring. A monitor/ control unit is not necessary in this type of system. An example of this is the DLCN ring [37].

A comparison of the three types of basic ring system [38] shows that although a Pierce loop allows simultaneous users, a small ring size restricts the number of time slots available, thus restricting the number of simultaneous users. A Newhall loop is superior to a Pierce loop at high mean message arrival rates on small rings. A delay insertion loop allows simultaneous users. However an elaborate protocol may be needed to handle real-time data due to the unpredictable message delays caused by intervening nodes transmitting to the bus. Of the three, only the delay insertion loop has no requirement for a master node at some point on the ring.

Simulations of the performance of the Cambridge ring system [39,40] have shown it to perform well under conditions of low load. However, an increase in the number of nodes on the ring can seriously degrade its handling of real time messages due to the time taken for the signal regeneration at each node. Under conditions of heavy loading the message transmission delay increased towards a guaranteed maximum value. The standard Cambridge ring system uses 38 bit packets, of which a maximum of 16 may be used for data. Thus there is a mimimum inherent overhead of 58 percent in the system.

Linear Bus LANs may be divided into two more general classes, synchronous and asynchronous. The former requires some form centralised control function, whilst the latter uses completely decentralised control. A system which uses decentralised control has a very high reliability, however at high bus loading the mean message arrival times will be significantly higher than in the centralised control system, due to the bus arbitration techniques used. As already described, the ETHERNET [8] system is an example of a CSMA-CD LAN (Carrier Sense Multiple Access with Collision Detect) in which bus control is achieved by a system of collision detection and random retransmission. In such a system bus utilisation can reach 98 percent under heavy loading [8] using data packets of length 512 bytes or longer. Approximately 21 percent of this traffic is ETHERNET overheads such as packet headers, implying that under these conditions of very high load, useful bus utilisation can exceed 75 percent. However if the size of the packets is reduced whilst maintaining the bus loading, channel utilisation drops dramatically due to the increased number of collisions. If a packet length of 64 bytes is used, utilisation drops to approximately 80 percent, giving a useful bus utilisation of approximately 63 percent. Simulation suggests [39,40] that for the long packets, message transmission delay times can increase by a factor of ten (as compared to low loading), whilst for short packets, the delay can increase by factor of 50. Additionally, there is no error recovery scheme inherent in the design of ETHERNET, thus any additional error recovery messages included in the basic protocol would reduce the utilisation still further.

The ASWE Serial Highway uses a centralised controller. The polling scheme, which is in operation at all times, polls every terminal in turn and represents a constant overhead. A controller poll consists of 7 bytes, as does a Nothing to Transmit response from a terminal. A typical message from a terminal with some data content has a length in the range 12-72 inclusive, and includes 12 bytes of control information. Under conditions of maximum loading, where every transmission from a terminal is a maximum length information message, the effective channel utilisation is approximately 76 percent. This decreases as the load decreases to 50 percent useful utilisation at 31 percent loading. Their are two major advantages of this system; firstly, the message transmission delay time at high loading is increased by only a factor of six as compared to the low loading situation (for a maximal system consisting 64 terminal units). Secondly, an error recovery scheme is included in the message protocols, and this scheme only necessitates additional bus traffic if an error is detected. In this system, the controller maintains the recovery scheme, and a message backup store is not needed in every transmitting node, as would be the case if a standard ETHERNET system was to include error recovery.

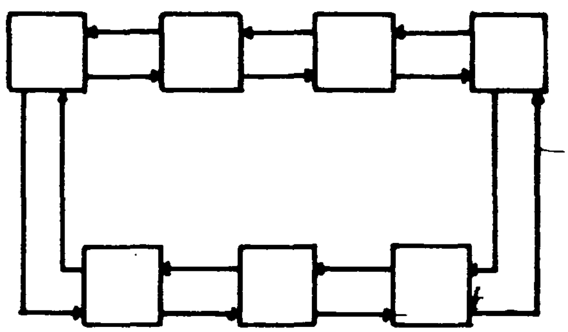
7.3 Improvements to the Basic LAN Technologies

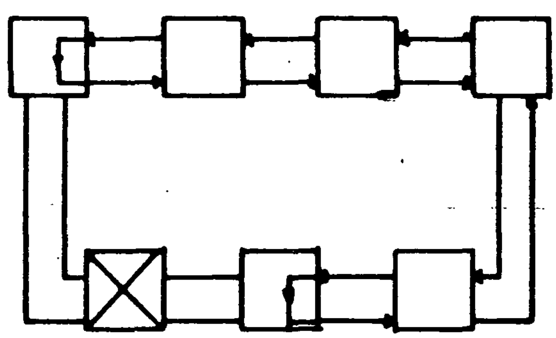
7.3:1 Ring LANs

A great deal of research has been carried out on several areas of the ring LAN technology in an attempt to eradicate some of the more obvious disadvantages. The first of these areas concerns the problems of ring failure due to the malfunction of a ring node or interelement cable. The Litton-DPS system [41] is designed for military applications and incorporates dual redundancy of the ring cabling to decrease the systems' vulnerability. Two cables connect every node on the ring. The primary loop is used for data, whilst the second is used for backup. The bus controller, which may be any unit on the bus, continually monitors bus operation for abnormal conditions. A backup bus controller is also assigned, whose task is to monitor both the bus and the bus controller, and to assume control when it determines that the normal controller has failed. An idle pattern is continually transmitted on the backup ring to enable its status to be monitored. The failure of any node is easily detected, and those nodes adjacent to the failed node can automatically switch that node out of the ring (Figure 7.3:1a). It is based on a Newhall ring system. The bus controller provides clock synchronisation for the ring, and maintains the 'Go Ahead' token. If more than one node or cable failure occurs, the ring can still function as two or more separate smaller rings, since the bus controller function may be dynamically reassigned. The system is implemented using advanced high speed processors and the current transmission rate of 20mbits/ sec can be increased by the replacement of the coaxial cable bus with a fibre-optic bus, with no change to the ring protocols.

The Litton-DPS system has approached the problems of ring vulnerability with the addition of a more complex communications processor at every node. As a possible alternative, work performed at MIT [42] suggests a much simpler alternative using a 'Star-Shaped Ring Network'. In a normal Cambridge ring system, the electronic failure of a node is protected against by providing a bypass relay which will connect the input cable to the output cable should the node fail to maintain a signal 'I am functioning correctly'. Unfortunately, should this signal be maintained if the node is not functioning correctly, the bypass relay cannot be activated, and the ring will be rendered unusable. This will then necessitate the local testing of every node to attempt to discover the unit which is malfunctioning. The work performed at MIT recommends the inclusion of a 'wire centre' to which all of the cables are routed, as shown in Figure 7.3:1b. The bypass relays are re-sited at the wire centre and the cabling to every node consists of two ring cables (input and output) and the 'I am functioning' signal. This signal is monitored by the wire centre, as

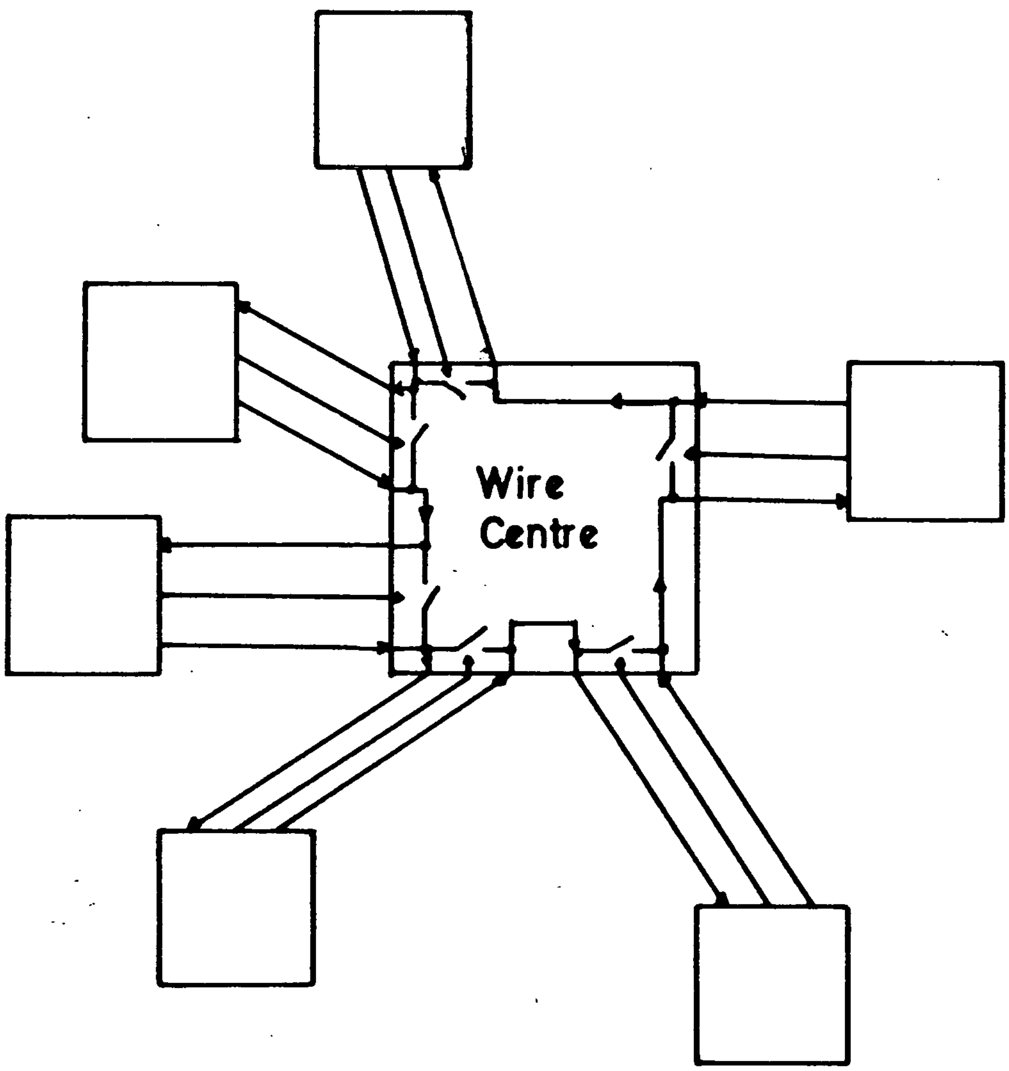
LITTON-DPS System





Dual Ring allows automatic reconfiguration in the event of the failure of a node.

Figure 7.3.1a



Star Shaped Ring LAN System

Figure 7.3.1b

is the activity on the ring cables. A failure of the signal, or abnormal activity (or lack of activity) on the cables from any node result in the bypass relay being activated. This scheme also allows greater ease of reconfiguration, since there is no need to break the ring to add further nodes. An additional cable need only be connected into the wire centre, and when the node is operational, the relevant bypass relay will be deactivated.

As mentioned, the first approach involves a far more complex communications processor in every node. The second approach is simpler, but more vulnerable since damage or malfunction in the wire centre could cause complete ring failure.

An additional problem in any ring system is that it is impossible to incorporate any type of priority access scheme. This is due to the round robin token passing system which is inherent in a ring network, and in certain applications is a serious drawback.

7.3.2 Decentralised Control Linear Bus LANs

ETHERNET has many advantages over a ring system because of its passive bus construction. Its overall bus utilisation and message transmission delay degradation at high loading cannot be significantly improved while still using the original CSMA-CD principles. However, priority access can be included into the ETHERNET system [43]. This allows important information to be transmitted with less delay at times of high bus loading. This priority system functions as follows; each packet is preceded by a preamble signal of length corresponding to its priority. A packet of the lowest priority has no preamble signal. When the channel is busy, a station wanting to transmit a packet waits until the channel becomes idle. When a collision is detected during transmission, the station does not stop transmitting the packet if the collision is within its preamble period. When the collision becomes undetected during its preamble, the station continues to transmit the packet. This case means that the other packets had priority levels lower than that of its packet. When the station detects collision during the transmission of its packet, it aborts the packet and retransmits it after some random delay. This corresponds to the case when the other packets' priority was higher than that of its packet. In a system using two priority levels, when the ratio of the traffic of the higher level packet to the total traffic is small (less than 20 percent) the higher level packet is nearly always successfully transmitted after only one trial, even under heavy loading [43].

Motorola have devised a system [44] in which the round-robin polling scheme described in section 1.1:2 has been implemented using completely decentralised control. In normal operation, each node sounds-off in sequence by sending a packet which identifies it as the current user of the channel. All other nodes hear these sound-off packets and synchronise to them. Each node finds its place in the sequence when it is time to sieze the channel. If a node has information to transmit, it sends the data immediately after its sound-off packet, up to a predefined time limit. All other nodes monitor the channel, and can determine when it has finished occupying the channel so that the following user may proceed.

When a user node fails, the other users detect the failure by sensing that the channel has been idle longer than the prescribed waiting time. When this happens, all nodes know who is the next expected user, and update their 'next expected user' counters accordingly. Although the sound-off packets contribute to the system overheads, they do contain message source information which may therefore be omitted from the information packet. New nodes may be added by updating the user lists at each node.

This system does not need a centralised controller, however one or more of the nodes must have the ability to cause the other nodes to alter their user lists. Since this system is essentially a message slot system, the choice of maximum information packet length will dictate the message transmission delay. A priority scheme cannot be implemented in this system due to the round-robin nature of the access scheme.

In conclusion, in ETHERNET systems, although overall message transmission delay times may be seriously degraded by high bus loading, a great improvement may be achieved for a small percentage of the traffic by the inclusion of a system of message priorities. A sound-off scheme can succesfully be used to decrease this delay time under high loading, however a priority system cannot be implemented. ETHERNET is most efficient under light loading, when very few collisions occur, whilst the sound-off scheme, which is similar in effect to an LAN system with a polling mechanism, is more efficient at higher loadings.

7.3:2 Centralised Control Linear Bus LANs

A system designed by Sperry Univac for the Canadian Government utilises multiple bus cabling and reassignable centralised bus control [45]. This system is part of SHINPADS (SHipboard INtegrated Processing And Display System). The key areas of interest in the development of this bus system were bus access time, and transmission system reconfiguration time. There are several bus cables, of which two are used at any time. One is the control channel, the other is the data channel. The former is used solely for the purpose of system control and reconfiguration, whilst the latter is reserved entirely for message traffic. Bus arbitration is carried out on the control channel, with the net result being a controlled allocation of the other channel for the purpose of sending data. This allows 'pipeline' levels of performance to be achieved on the data channel. Any of the available channels may be used as a control or data channel. The arbitration is carried out by a reassignable bus controller. Each node includes a control processor which can function either as a normal bus node, or as a bus master and bus node. The node assigned to be bus master polls the other nodes and determines their data channel usage requirements. It then dispatches the authority to transmit on the data channel to the node with the highest priority. The node priorities and the frequency of polling of nodes relative to others are under user control. Requests for the use of the data channel fall into one of two categories; immediate and normal. In the immediate mode, the relevant node is given immediate access to the data channel at the end of the current transmission, providing there are no other immediate requests in the controllers queues. In this case the new request is added to the end of the queue. Normal requests are queued according to the priority of the requesting node. The terminal nodes continually monitor the control channel for activity. If no activity is detected for more than a certain period, the activity on all other channels is monitored for normal control activity. Should this be detected, a systematic change of active control channels will take place in the terminal node. If no activity is detected on any channels, then the bus controller function must be reassigned to one of the other nodes. Currently, this reassignment is directed by the user, who may either direct the node to which he is connected to assume bus control, or may direct another node to assume bus control.

In a polled linear bus system, the overheads due to the poll-response system cause great inefficiency under conditions of light loading. As a possible solution to this problem, an adaptive polling technique has recently been proposed [46,47[. The essence of this technique, which has been designated probing, is to poll groups of terminals rather than individual units. If a member of a group of terminals being probed has a message to transmit, it responds in the affirmative by transmitting on the bus. Upon receiving a positive response to a probe, the controller splits the group into two sub-groups and probes each in turn. This process continues until the relevant terminal is isolated. This type of polling system is essentially a tree search. The best system performance may be obtained by dynamically varying the size of the group being polled, according to the probability of a terminal having a message. Thus at times of high loading, the polling system would be similar to that in a pure polling system, whilst in times of light loading, large groups would be polled. Compared to the conventional pollng system, this system will offer substantially improved message transmission delay times at light loadings, and similar delays at heavy loadings.

In conclusion, it can be said that in conditions of high loading, the centralised control bus systems are superior in performance to the decentralised systems. It is very easy to add a priority polling scheme because the controller has complete control over the allocation of bus access. New terminals may be added to the polling system by merely causing the controller to add another terminal to its polling scheme. Unfortunately, the pure polling scheme becomes inefficient when used on a bus with a large number of terminals, and the probing technique described improves the performance of a polled system when there is a large probability that few of the units will have a message to transmit. Due to the fact that a central controller is used, care must be taken in providing a mechanism for this function to be reassigned after equipment failure. Complete network failure will result should this function not be reassigned.

7.4 A Second Generation ASWE Serial Highway

It has become obvious during this review of different systems that when faced with similar criteria for the choice of LAN technology, different research groups have made different choices of LAN technology. In general it would appear that when a decision is made to seek an LAN with better characteristics than available from the one currently in use, most groups chose to upgrade their current system, rather than to switch technologies.

In the case of the ASH, it would appear that the original aims of the system designers cannot be fulfilled by a radical technology change. A ring system could not offer the system survivability offered by the passive linear bus. It is interesting to note that the Litton-DPS system [41[is being developed for the same type of military applications as the ASH, however its designers consider that it is suitable for this environment. The addition of the second ring cable allows single node or cable failure, however if more failures occur the ring will be segmented into several sections. This is clearly undesirable, when in a linear bus system it would be possible to include a higher level of cable redundancy to protect against a greater number of cable failures.

The CSMA-CD systems offer an attractive alternative to the polled system currently used. However, the uncertainty in message transmission delay times would be a serious problem in a LAN system primarily concerned with real time data. Additionally, a priority system is essential for the transmission of critical data in military applications. The priority ETHERNET system described would be a possible alternative to the polled system currently being used. It offers the advantage of completely decentralised control and is the best alternative of the asynchronous linear systems.

It is, however, a requirement that the original ASH specification be conformed to as much as is possible. The areas of interest in a second generation ASH are; decentralisation of control function, and decreasing the polling scheme overheads on the bus. A possible solution to the latter is the probing scheme described. The addition of grouping protocols related to the terminal units 'Highway Number' would allow this system to function and necessitate very little change to the basic specification. However, the combination of the probing technique and the SHINPADS serial data bus techniques would provide a very powerful second generation technique. If a probing scheme was used on the control channel it would reduce the data channel access time due to the normal polling system overheads. Also, the since the control and data channels are being operated in parallel, a great increase in message throughput could be achieved.

In the current ASH system, the highway controller is entirely separate, in both hardware and software, from the computers to which the terminal units are connected. Any change in the polling list or polling priorities must be originated by the computer which is host to the highway controller. Adoption of the system suggested above would allow such alterations to be originated from elsewhere in the system, because the controller function would be incorporated into the terminal nodes and its tables could be altered by appropriate instructions to its co-resident terminal node. The controller function would be duplicated at all the nodes, however only one controller would be active at any time. If that controller should fail, its function could be taken over by another node, possibly on the basis of 'highway number' or possibly by contention access. The present system of error recovery could still be maintained, as could the present protocol system. The control messages would be transmitted on the control bus and the information messages on the data bus. Without the inclusion of the probing scheme or any protocol changes, this would mean that the throughput could be increased dramatically for low loads. By the addition to the protocols of a control message from the terminal units saying 'Yes I have something to transmit' and a message from the controller saying 'Proceed', the throughput under all loading conditions could be improved, due to the fact that while the data bus was in use, the controller could continue its polling cycle until it found a terminal with something to transmit. It would then wait until the current data bus user had completed its transmission, and signal to the relevant terminal node that it could proceed.

This design change would necessitate a large change in the hardware of the interface. However, several of the inherent problems in the current ASH would be removed, and the survivability of the LAN system would be greatly increased.

7.5 Conclusion

This section has described in detail various alternative approaches to improving the characteristics of the basic LANs; the ring systems, the decentralised control linear bus systems, and the centralised control linear bus systems. Several of the techniques used are now implemented in working systems, whilst others are still at the simulation stage. It has become obvious that the mingling of different LAN technologies gives a significant improvement as compared to the original systems. A second generation ASWE Serial Highway system has been considered, in which the original specifications are conformed to as closely as possible, and the original design criteria are used in the selection of new techniques. A system based upon a twin channel centralised control system was chosen as the most attractive possibility. This type of system offers a greater system throughput under all loading conditions by operating the control and data channels in parallel. The addition of a completely reassignable controller function, by incorporating the controller function into each of the terminal units, would give a great increase in survivability. An adaptive polling technique, known as probing, is discussed, and it is suggested that its inclusion in the second generation ASH, while necessitating some message format changes, would give an even greater improvement in the low loading message transmission delay time.

Chapter 8

Conclusion

Distributed computing systems fall into two categories, loosely and tightly coupled. The loosely coupled systems normally communicate via a serial cable, and are known as Local Area Networks. These distributed systems are used as replacements for large single mainframes, as the distribution of hardware and software greatly improves the systems' survivability and eases the initial testing. Most LAN systems currently under development fall into one of two categories, ring or linear bus.

ASWE have developed their own LAN for naval applications in the late 1980s and 1990s. It is based on a linear bus LAN with a central controller. The addition of redundancy of the controller function and the cabling gives greater system survivability. Ths system has been used as a laboratory test bed for some time, and the basic principles of operation have been well proved. It is implemented using high speed bipolar bit-slice microprocessors in dedicated front-end processors. These FEPs communicate with their minicomputer hosts via an area of shared memory.

There are several alternative LAN technologies available. Ring LANs are most suited to office applications using short rings, where the delays introduced by signal regeneration at every node are not significant, owing to the non real-time nature of the messages. Also, the ring LANs are very vulnerable to cable or node failure, and if they are to be used in military applications great care must be taken over the provision of redundant signal paths to protect the integrity of the system. Although this type of LAN theoretically has the advantage of completely distributed control, in practice most of the common systems have a monitor/ control station to supervise the LANs activity.

Linear bus LANs are more suited to military applications than ring LANs due to the possibility of using a passive bus (no signal regeneration at nodes). There are two possible types of linear bus LANs, those with decentralised control, such as ETHERNET, and those with centralised control, such as the ASH. The former has greater survivability, whilst the latter performs better in conditions of high bus loading.

This thesis has described the replacement of the minicomputer hosts normally used with the ASH by microcomputer hosts based around Motorola's MC68000 16bit microprocessor. This replacement gave an enormous reduction in size, allowing the new system to be installed on board a ship. Tests were performed on the integrity of this system whilst the ship was performing normal manoevres. Analysis of these results has given the first performance data on the ASH system when used in the environment for which it was designed. It performed perfectly at all times, and there was no indication that any of the error protection systems currently employed would need to be changed.

As part of this system, a portable highway controller was developed. This aroused considerable interest within ASWE, as it was able to perform all of the functions which are currently performed by a highway controller FEP with a Ferranti Argus host, at a fraction of the current cost and size. Separate trials of this unit have been performed at ASWE, over long periods of time (months). These have indicated that the unit performs to its specifications, and a new controller unit may be manufactured based upon the highway controller designed and built at Durham.

A review of work currently being performed in the LAN field has been carried out. The shortcomings of each type of LAN system are being reduced by mingling the different technologies. It is suggested that a combination of two of the 'new generation' LAN systems with the ASH, would give substantial performance increases, with the need for minimal specification changes. This new system would have multiple redundant linear buses, and would use two simultaneously. One channel would be used for control information and the other for data. This would allow 'pipeline' levels of performance to be achieved on the data channel.

To conclude, LAN technology has advanced alongside the ever increasing demands for greater speed, reconfigurability and survivability of distributed computing systems. However, as these demands grow ever greater and more difficult to realise, it is necessary to make modifications to the basic LAN technology. In order to further improve the ASWE Serial Highway system, it will be necessary to perform substantial changes in the basic system. The ASH is now ten years old, and very few of the other LAN systems have survived that length of time without major alterations.

Notes on publications by the author:-

"Boost µP-board memory capacity with simple hardware changes"
D. Cowan, EDI, 29th October 1981, pp 197-198.

Bibliography

[1] F.G. Heart et al "The Interface Message Processor for the ARPA Computer Network" AFIPS Conf. Proc. SJCC 36, June 1970.

[2] H. Frank, I.T. Frisch, W.S. Chou "Topological Considerations in the Design of the ARPA Computer Network" AFIPS Conf. Proc. SJCC 36, June 1970.

[3] D.J. Farber "Networks: An Introduction" Datamation, April 1972, pp 36-39.

[4] E.G. Rawson, R.M. Metcalfe "FIBERNET: Multimode Optical Fibers for Local Computer Networks" IEEE Trans. Comm. 26, 7 (July 1978) pp 983-990.

[5] MIL-STD-1553B, 21 September 1978 (Aircraft Internal Time Division Command/ Response Multiplex Data Bus, DoD, Washington, D.C.)

[6] K. Lunn, K.H. Bennett "Message Transport on the Cambridge Ring- a Simulation", Software-practical and experimental (GB), Vol 11, part 7, 1981, pp 711-716.

[7] R.M. Metcalfe, D.R. Boggs "Ethernet: Distributed Packet Switching for Local Computer Networks", Comm. ACM. 19, 7 July 1979.

[8] J.F. Shoch, J.A. Hupp "Measured Performance of an Ethernet Local Area Network" Comm. ACM 23, December 1980.

[9] N.Abramson "The ALOHA System- another alternative for computer communications" Proc. 1977 Fall Joint Comp. Conf. 37, AFIPS Press pp 281-285.

[10] N.Abramson "The Throughput of Packet Switching Braodcasting Channels" IEEE Trans. Comm. 25, Jan 1977, pp 117-128.

[11] Defence Standard 00-19/ Issue 1, Ministry of Defence, 19th January 1981.

[12] "The AMD2900 Family Data Book" AMD, 1979.

[13] J. Mick, J. Brick "Bit-Slice Microprocessor Design", McGraw-Hill, 1980.

[14] R.D. Weatherby "Mini Link X2901 Cross Assembler" Cambridge Consultants Ltd. report, December 1976.

[15] C.T. Spracklen "Durham University-ASWE Minilink Simulator" Durham University report, August 1978.

[16] M.Stainsby "Specification of the Software Interface to the Highway Controller (Version 6), Draft 3, 2nd May 1980, ASWE report.

[17] D.M. Ritchie, K.Thompson "The Unix Time Sharing System" Comm. ACM 17, July 1974, pp 365-375.

[18] K.L. Hunt, R.J. Firth "Guide to Coral-66 on the PDP11-45" RMCS, February 1976.

[19] "Official Definition of CORAL 66" Ministry of Technology, HMSO, 1970.

[20] "RT-11 System Reference Manual" Digital Equipment Corp. 1976.

[21] "NOVA Line Computers" Data General, 1979.

[22] "Real Time Disk Operating System (RDOS) Reference Manual" Data General, 1979.

[23] "MC6809 Preliminary Programming Manual" Motorola Inc. 1979.

[24] "The MSI 6800 Computer System" Midwest Scientific Instruments Inc. 1977.

[25] "DOS-69" Smoke Signal Broadcasting, 1980.

[26] "MC68000 Design Module, Users Guide" Motorola Inc. 1979.

[27] "MC68000, 16-Bit Microprocessor, Users Manual" Motorola Inc. 1979

[28] G. Kane, D. Hawkins, L. Leventhal "68000 Assembly Language Programming" OSBORNE/ McGraw-Hill, 1981.

[29] "Microcomputer Components" Motorola Inc. 1979.

[30] C.H. Moore "Forth: A New Way to Program a Minicomputer" Astron. Astrophys. Supp. 15, pp497-511 1974.

[31] E.D. Rather "Forth: A Fresh Approach to Programming" Forth Inc. 1977.

[32] D. Cowan, C.T. Spracklen "Annual Report, 1981" Durham University, 1981.

[33] "Feltec LCD Display" Feltec, 1981.

[34] J.R. Pierce "Network for Block Switching of Data" BSTJ 51, 6, July-August 1972.

[35] W.D. Farmer, E.E. Newhall "An Experimental Distributed Switching System to Handle Bursty Computer Traffic" Proc. ACM Symp. on Problems in the Organisation of Data Comms. October 1969, pp 1-34.

[36] E.R. Hafner et al "A Digital Loop Communications System" IEEE Trans. Comm. 23 June 1974, pp 877-881.

[37] A.B. Gojko et al "A Performance Study of the Distributed Loop Computer Network (DLCN)" Proc. Comp. Networking Symposium, NBS 15 December 1977.

[38] P.A. Willis "Data Buses and Distributed Data Processing in U.S. Navy Ships" Naval Surface Weapons Center/ Dahlgren, Spetember 1981.

[39] G.S. Blair "A Performance Study of the Cambridge Ring" Computer Networks, 6 (1982) pp 13-20.

[40] M.V. Wilkes, D.J. Wheeler "The Cambridge Digital Communications Ring" Local Area Comms. Network Symp. Boston May 1979.

[41] R. Mauriello "A Distributed Processing System for Military Applications" Computer Design, Sept-Nov 1980.

[42] J.H. Slatzer, K.T. Pogan "A Star-Shaped Ring Network with High Maintainability" Comp. Networks 4, 1980, pp 239-244.

[43] I. Iida et al "Random Access Packet Switched Local Computer Network with Priority Function" IEEE Telecomms. Conf. 30 Nov 1980, pp 37.41-6.

[44] D.Scavezze "Nodes Sound-Off to Control Access to Local Network" Electronics (USA) 1981, 54, 12, pp 176-181.

[45] S.C. Andersen "A Serial Data Bus Control Method" Comp. Networks 3, 1979, pp 361-372.

[46] J.F. Hayes "An Adaptive Technique for Local Distribution" IEEE Trans. Comm. 26, Aug 1978, pp 1178-1186.

[47] J.F. Hayes "Local Distribution in Computer Communications" IEEE Comms.

[48] A.K. Agrawal, V.V. Vadakan "Jet Propulsion Local Area Network (JPLAN)" 2nd Conf. on Dist. Comp. Systems. Paris, 8th April 1981, pp 360-368.

[49] M.A. Dineson "Broadband Local Area Networks Enhance Communication Design" EDN, 1981, 26, 5, pp 77-85.

[50] D.D. Clark et al "An Introduction to Local Area Networks" Proc. IEEE, 66, 11th November 1978, pp 1497-1517.

[51] A.K. Mok, S.A. Ward "Distributed Broadcast Channel Access" Comp. Networks, 3, 1979, pp 327-335.

[52] W.B. Watson "Simulation Study of the Traffic Dependant Performance of a Prioritised CSMA Braodcast Network" Comp. Networks, 3, 1979, pp 427-434.

[53] S.S. Lam "A Carrier Sense Multiple Access Protocol for Local Area Networks" Comp. Networks, 4, 1980, pp 21-32.

[54] F.A. Tobagi, V.B. Hunt "Performance Analysis of Carrier Sense Multiple Access with Collision Detect" Comp. Networks, 4, 1980, pp 245-259.

[55] J.F. Shoch "Carrying Voice Traffic Through an ETHERNET Local Network- A General Overview" Int. Workshop on Local Area Comp. Networks, Zurich, Aug 1980.

[56] J.F. Shoch "An Annotated Bibliography on Local Computer Networks" XEROX, Palo Alto, 1980.

[57] T.D. Wells, M.G. Stainsby "ADNET: An Experimental Information Distribution System" ASWE, XCC, October 1978.

APPENDIX A

Program Listings

```
                        *CHEKL*
                        *CHECK FOR FLAG IN A REGISTER LOW*

015E 34    12           CHEKL  PSHS   A,X
0160 8E    F538                LDX    #PIA
0163 A5    02                  BITA   DRB,X
0165 27    04    016B          BEQ    CHKLR
0167 1A    01                  SEC
0169 20    02    016D          BRA    CHKLR2
016B 1C    FE           CHKLR  ANDCC  #$FE      CLC
016D 35    92           CHKLR2 PULS   A,X,PC

                        *CHEKH*
                        * CHECKS FOR FLAG HIGH *

016F 34    12           CHEKH  PSHS   A,X
0171 8E    F538                LDX    #PIA
0174 A5    02                  BITA   DRB,X
0176 26    04    017C          BNE    CHKHR
0178 1A    01                  SEC
017A 20    02    017E          BRA    CHKHR2
017C 1C    FE           CHKHR  ANDCC  #$FE
017E 35    92           CHKHR2 PULS   A,X,PC

                        *LSET*
                        * SETS THE APPROPRIATE FLAG LOW*

0180 34    10           LSET   PSHS   X
0182 8E    F538                LDX    #PIA
0185 43                        COMA
0186 A4    02                  ANDA   DRB,X
0188 A7    02                  STA    DRB,X
018A 35    90                  PULS   X,PC

                        *HSET*
                        *SETS APPROPRIATE FLAG HIGH*

018C 34    10           HSET   PSHS   X
018E 8E    F538                LDX    #PIA
0191 AA    02                  ORA    DRB,X
0193 A7    02                  STA    DRB,X
0195 35    90                  PULS   X,PC

                        *AWRITE*
                        *WRITE TO PROM PROGRAMMER *


0197 9D    B2           AWRITE JSR    ADDRS
0199 25    20    01BB          BCS    WERR1
019B BD    0102                JSR    GETTB
019E BC    05F7         AWRTE1 CMPX   DTBEND
01A1 2E    17    01BA          BGT    AWRTE2
01A3 A6    80                  LDA    ,X+
01A5 B4    06A5                ANDA   MASK
```



```
01A8 BD    0265                JSR    DWRITE
01AB 24    F1    019E          BCC    AWRTE1
01AD 34    10                  PSHS   X
01AF 30    8D 0312             LEAX   STR9,PCR
01B3 BD    D2A6                JSR    ZOUTST
01B6 35    10                  PULS   X
01B8 20    E4    019E          BRA    AWRTE1
01BA 39                 AWRTE2 RTS
01BB 30    8D 02B8      WERR1  LEAX   STR6,PCR
01BF BD    D2A6                JSR    ZOUTST
01C2 39                        RTS

                        *ALOAD*
                        *LOAD A BINARY DISK FILE INTO THE DATA TABLE*


01C3 30    8D 0312      ALOAD  LEAX   STR10,PCR FILE NAME QUIZ
01C7 BD    D2A6                JSR    ZOUTST
01CA BD    D2B5                JSR    ZLINEI
01CD BD    D2BB                JSR    ZPEEK     LOOK AT NEXT CHAR
01D0 81    0D                  CMPA   #CR       CR?
01D2 27    1A    01EE          BEQ    LOAD1     IF YES JUST CONTINUE FROM OL
01D4 BD    D783                JSR    CDFM      JUST TO BE SAFE
01D7 30    8D 041E             LEAX   RFCB,PCR
01DB BD    D291                JSR    ZFLSPC
01DE 86    04                  LDA    #QSO4R    SET FOR READ
01E0 A7    84                  STA    XFC,X
01E2 BD    D786                JSR    DFM       GO OPEN IT
01E5 27    07    01EE          BEQ    LOAD1     OK SO BRA
01E7 BD    D2A9                JSR    ZTYPDE
01EA BD    D783                JSR    CDFM
01ED 39                        RTS
01EE 86    05           LOAD1  LDA    #QSREAD
01F0 A7    84                  STA    XFC,X
01F2 17    FF0D 0102           LBSR   GETTB
01F5 1F    12                  TFR    X,Y
01F7 30    8D 0317             LEAX   STR12,PCR SIZE QUIZ
01FB BD    D2A6                JSR    ZOUTST    PUT IT
01FE BD    D2B5                JSR    ZLINEI    REPLY
0201 BD    D2A0                JSR    ZGETHN    TRANSLATE
0204 5D                        TSTB
0205 27    0B    0212          BEQ    LOAD7     NO REPLY SO CONTINUE
0207 1F    20                  TFR    Y,D       DATA TABLE BASE
0209 BF    05F7                STX    DTBEND    OOPS
020C F3    05F7                ADDD   DTBEND    NEW END
020F FD    05F7                STD    DTBEND    SAVE IT
0212 8E    05F9         LOAD7  LDX    #RFCB
0215 BD    D786         LOAD6  JSR    DFM
0218 26    13    022D          BNE    LOAD2
021A 81    FF                  CMPA   #$FF
021C 26    F7    0215          BNE    LOAD6
021E 10BC  05F7         LOAD4  CMPY   DTBEND    END OF TABLE?
0222 2E    13    0237          BGT    LOAD3
0224 BD    D786                JSR    DFM       READ ANOTHER CHAR
0227 26    04    022D          BNE    LOAD2
0229 A7    A0                  STA    0,Y+      AND STORE IT
022B 20    F1    021E          BRA    LOAD4
```

```
00333 0092 26   04   0098          BNE    COM4
00334 0094 8E   01C3               LDX    #ALOAD
00335 0097 39                      RTS
00336 0098 81   4D        COM4     CMPA   #MODIFY
00337 009A 26   04   00A0          BNE    COM5
00338 009C 8E   0360               LDX    #AMOD
00339 009F 39                      RTS
00340 00A0 81   53        COM5     CMPA   #SIZE
00341 00A2 26   04   00A8          BNE    COM6
00342 00A4 8E   02F7               LDX    #ASIZE
00343 00A7 39                      RTS
00344 00A8 81   45        COM6     CMPA   #MONIT
00345 00AA 26   A9   0055          BNE    GETCM
00346 00AC BD   D783               JSR    CDFM
00347 00AF 7E   FC57               JMP    AMONIT
00348
00349
00350
00351                     *ADDRS*
00352                     *INTERPRETS START AND END ADDRESS*
00353                     *AND SETS UP PROGRAMMER ACCORDINGLY*
00354                     *CARRY SET IF ERRO OCCURS*
00355
00356
00357
00358 00B2 17   0183 0238 ADDRS    LBSR   PRINIT    INITIALISE THE PROGRAMMER
00359 00B5 86   01                 LDA    #ADD
00360 00B7 17   00A4 015E ADDRS1   LBSR   CHEKL     CHECK FOR ADDRESS LOW
00361 00BA 25   FB   00B7          BCS    ADDRS1    IF CARRY SET THEN ADDRESS HI
00362 00BC 86   10                 LDA    #MODE
00363 00BE 17   00BF 0180          LBSR   LSET      SET MODE LOW
00364 00C1 30   8D 036B            LEAX   STR1,PCR
00365 00C5 BD   D2A6               JSR    ZOUTST    SEND LOW ADDRESS QUIZ
00366 00C8 BD   D2B5               JSR    ZLINEI    AND RECEIVE IT
00367 00CB BD   D2A0               JSR    ZGETHN    MAKE IT INTO HEX ADDRESS IN
00368 00CE 5D                      TSTB             IF ZERO NO HEX ADDRESS
00369 00CF 26   03   00D4          BNE    ADDRS2
00370 00D1 8E   0000               LDX    #$0000    SO LOAD DEFAULT
00371 00D4 17   01DE 02B5 ADDRS2   LBSR   WRITEA    WRITE BOTTOM ADDRESS TO PROG
00372 00D7 AF   8D 05C6            STX    BOT,PCR
00373 00DB 30   8D 0368            LEAX   STR2,PCR
00374 00DF BD   D2A6               JSR    ZOUTST
00375 00E2 BD   D2B5               JSR    ZLINEI
00376 00E5 BD   D2A0               JSR    ZGETHN    TOP ADDRESS
00377 00E8 5D                      TSTB
00378 00E9 26   03   00EE          BNE    ADDRS3
00379 00EB BE   06A3               LDX    HIGH
00380 00EE 17   01C4 02B5 ADDRS3   LBSR   WRITEA
00381 00F1 25   0C   00FF          BCS    ADDER1
00382 00F3 AF   8D 05A8            STX    TOP,PCR
00383 00F7 86   01        ADDRS4   LDA    #ADD
***WARNING   001--00000
00384 00F9 17   0073 016F          LBSR   CHEKH
00385 00FC 25   F9   00F7          BCS    ADDRS4    CHECK FOR ADDRESS HIGH
00386 00FE 39                      RTS
00387 00FF 1A   01        ADDER1   SEC
00388 0101 39                      RTS
```



```
00389
00390
00391
00392                     *GETTB*
00393                     *GET THE DATA TABLE BASE*
00394                     *ALSO SETS TABLE TOP TO BASE ADDRESS PLUS 512 *
00395
00396
00397 0102 30   8D 0355   GETTB    LEAX   STR3,PCR TABLE BSE QUIZ
00398 0106 BD   D2A6               JSR    ZOUTST
00399 0109 BD   D2B5               JSR    ZLINEI
00400 010C BD   D2A0               JSR    ZGETHN
00401 010F 5D                      TSTB
00402 0110 26   03   0115          BNE    GETTB1
00403 0112 8E   06A9               LDX    #DTABLE
00404 0115 FC   069F      GETTB1   LDD    TOP
00405 0118 B3   06A1               SUBD   BOT
00406 011B FD   05F7               STD    DTBEND
00407 011E 1F   10                 TFR    X,D
00408 0120 F3   05F7               ADDD   DTBEND
00409 0123 FD   05F7               STD    DTBEND    NOW WE HAVE THE ABSOLUTE EN
00410 0126 39                      RTS
00411
00412
00413
00414
00415                     *AREAD*
00416                     *READS IN A TABLE FULL OF DATA*
00417                     *OR UNTIL THE PROGRAMMER CLAMS UP*
00418
00419
00420
****WARNING   001--00384
00421 0127 17   FF88 00B2 AREAD    LBSR   ADDRS
00422 012A 25   29   0155          BCS    AERR1
00423 012C 17   0114 0243          LBSR   SREAD
00424 012F 86   40                 LDA    #READW
****WARNING   001--00421
00425 0131 17   0058 018C          LBSR   HSET      SELECT READ
00426 0134 86   10                 LDA    #MODE
00427 0136 8D   54   018C          BSR    HSET      SET MODE HIGH FOR READ
00428 0138 8D   C8   0102          BSR    GETTB     GO GET TABLE ADDRESS
00429 013A BC   05F7      AREAD1   CPX    DTBEND
00430 013D 2E   0A   0149          BGT    AREAD2
00431 013F 17   0196 02D8          LBSR   DREAD     READ A BYTE
00432 0142 F4   06A5               ANDB   MASK
00433 0145 E7   80                 STB    ,X+
00434 0147 20   F1   013A          BRA    AREAD1
00435 0149 17   0108 0254 AREAD2   LBSR   SWRITE
00436 014C 86   40                 LDA    #READW
****WARNING   001--00425
00437 014E 17   002F 0180          LBSR   LSET
00438 0151 17   0100 0254          LBSR   SWRITE    SELECT WRITE
00439 0154 39                      RTS
00440 0155 30   8D 031E   AERR1    LEAX   STR4,PCR
00441 0159 BD   D2A6               JSR    ZOUTST
00442 015C 20   EB   0149          BRA    AREAD2
```

```
                         * DRIVER FOR PROM PROGRAMMER *
              F538       PIA    EQU   $F538
              0001       CRA    EQU   1
              0003       CRB    EQU   3
              0000       DRA    EQU   0
              0002       DRB    EQU   2
              00FB       NDDR   EQU   %11111011
              0004       DATA   EQU   %00000100
              00FF       AWRT2  EQU   $FF
              0000       ARD    EQU   $00
              0078       BREG   EQU   %01111000
              0001       ADD    EQU   $01
              0002       ERROR  EQU   $02
              0004       RES    EQU   $04
              0008       TRANS  EQU   $08
              0010       MODE   EQU   $10
              0020       INTER  EQU   $20
              0040       READW  EQU   $40
              0057       WRITE  EQU   'W
              0051       QUIT   EQU   'Q
              0052       READ   EQU   'R
              004C       LOAD   EQU   'L
              004D       MODIFY EQU   'M
              0053       SIZE   EQU   'S
              0045       MONIT  EQU   'E
              FC57       AMONIT EQU   $FC57


0000                            ORG   $0000
0000 10CE D200           MAIN   LDS   #$D200
0004 BD   0033                  JSR   PIAIN
0007 8E   0000                  LDX   #$0000
000A BF   06A1                  STX   BOT
000D 8E   01FF                  LDX   #$01FF
0010 BF   069F                  STX   TOP
0013 BF   06A3                  STX   HIGH
0016 BF   05F7                  STX   DTBEND
0019 86   03                    LDA   #$03
001B B7   06A6                  STA   AFIELD
001E 86   0F                    LDA   #$0F
0020 B7   06A5                  STA   MASK      DEFAULTS
0023 BD   0055           MAIN1  JSR   GETCM
0026 108E 002E                  LDY   #MAIN2
002A 34   20                    PSHS  Y
002C 1F   15                    TFR   X,PC
002E 17   0324 0355      MAIN2  LBSR  DELAY
0031 20   F0   0023             BRA   MAIN1




                         *PIAIN *
                         *INITIALISES THE PIA AS FOLLOWS *
                         *A SIDE ALL OUTPUTS *
                         *B SIDE 0:-ADDRESS  INPUT *
                         *       1:-ERROR    INPUT *
                         *       2:-RESPONSE INPUT *
                         *       3:-TRANSFER OUPUT *
                         *       4:-MODE     OUTPUT *
                         *       5 INTERLOCK OUTPUT *
                         *       6 BUFFER CONTROL OUTPUT *
                         * CA2 BUFFER CONTROL OUTPUT *



0033 8E   F538           PIAIN  LDX   #PIA      PIA ADDRESS
0036 86   00                    LDA   #$00
0038 A7   01                    STA   CRA,X     SET FOR WRITE TO DDR
003A A7   03                    STA   CRB,X     DITTO
003C C6   FF                    LDB   #AWRT2
003E E7   84                    STB   DRA,X     INITIALLY A FOR WRITE
0040 C6   78                    LDB   #BREG
0042 E7   02                    STB   DRB,X
0044 86   04                    LDA   #DATA
0046 A7   01                    STA   CRA,X
0048 A7   03                    STA   CRB,X     BOTH NOW DATA REGS
004A 86   40                    LDA   #READW
004C 17   0131 0180             LBSR  LSET      SET FOR WRITE
004F 86   18                    LDA   #%00011000 TRAN,MODE HIGH
0051 17   0138 018C             LBSR  HSET
0054 39                         RTS




                         *GETCM *
                         * COMMAND LINE INTERPRETER *


0055 30   8D 0557        GETCM  LEAX  STR15,PCR
0059 BD   D2A6                  JSR   ZOUTST
005C 30   8D 0534               LEAX  STR14,PCR
0060 BD   D2A6                  JSR   ZOUTST
0063 30   8D 03C5               LEAX  COMST,PCR
0067 BD   D2A6                  JSR   ZOUTST
006A BD   D2B5                  JSR   ZLINEI
006D BD   D297                  JSR   ZGNCHR
0070 84   7F                    ANDA  #$7F
0072 81   0D                    CMPA  #CR
0074 27   DF   0055             BEQ   GETCM
0076 81   52                    CMPA  #READ
0078 26   04   007E             BNE   COM1
007A 8E   0127                  LDX   #AREAD
007D 39                         RTS
007E 81   57             COM1   CMPA  #WRITE
0080 26   04   0086             BNE   COM2
0082 8E   0197                  LDX   #AWRITE
0085 39                         RTS
0086 81   51             COM2   CMPA  #QUIT
0088 26   06   0090             BNE   COM3
008A BD   D783                  JSR   CDFM
008D 7E   D283                  JMP   ZWARMS
0090 81   4C             COM3   CMPA  #LOAD
```

```
022D 30    8D 02BB   LOAD2  LEAX   STR11,PCR
0231 BD    D2A6             JSR    ZOUTST
0234 BD    D783             JSR    CDFM
0237 39             LOAD3  RTS
                    *PRINIT*
                    *INITIALISES THE PROM PROGRAMMER*
                    *BY PULSING INTERLOCK HIGH*


0238 86    20        PRINIT LDA    #INTER
023A 17    FF4F 018C        LBSR   HSET
023D 86    20               LDA    #INTER
023F 17    FF3E 0180        LBSR   LSET
0242 39                     RTS

                    *SREAD*
                    *SET UP THE A SIDE OF THE PIA FOR READ*


0243 108E F538       SREAD  LDY    #PIA
0247 86    00               LDA    #$00      FOR DDRS
0249 A7    21               STA    CRA,Y
024B 86    00               LDA    #ARD      SELECT A FOR READ
024D A7    A4               STA    DRA,Y
024F 86    04               LDA    #DATA
0251 A7    21               STA    CRA,Y
0253 39                     RTS


                    *SWRITE*
                    *SET UP THE A SIDE FOR WRITE*


0254 108E F538       SWRITE LDY    #PIA
0258 86    00               LDA    #$00
025A A7    21               STA    CRA,Y
025C 86    FF               LDA    #AWRT2    SELECT A FOR WRITE
025E A7    A4               STA    DRA,Y
0260 86    04               LDA    #DATA
0262 A7    21               STA    CRA,Y
0264 39                     RTS
                    *DWRITE*
                    *WRITE ONE BYTE OF DATA TO PROGRAMMER*
                    *CARRY SET ON RETURN INDICATES ERROR*
           FFFF      ESTORE EQU    -1


0265 34    56        DWRITE PSHS   A,B,X,U
0267 1F    43               TFR    S,U
0269 32    7E               LEAS   -2,S
026B 86    00               LDA    #$00
026D A7    5F               STA    ESTORE,U
026F 8E    F538             LDX    #PIA
0272 86    10               LDA    #MODE
0274 17    FEE7 015E        LBSR   CHEKL     CHECK FOR CORRECT MODE
0277 24    05   027E        BCC    DWR1      YES OK
0279 86    10               LDA    #MODE
```



```
027B 17    FF02 0180        LBSR   LSET      NO SO SET RIGHT ONE
027E E6    C4        DWR1   LDB    ,U        UNSTACK DATA, IT WAS IN A WH
0280 E7    84               STB    DRA,X     AND WRITE IT
0282 86    08               LDA    #TRANS
0284 17    FEF9 0180        LBSR   LSET      AND INITIATE TRANSFER
0287 86    04        DWR2   LDA    #RES
0289 17    FED2 015E        LBSR   CHEKL     WAIT FRO RES
028C 25    F9   0287        BCS    DWR2
028E 86    02               LDA    #ERROR
0290 17    FEDC 016F        LBSR   CHEKH     MAKE SURE NO EROR
0293 25    1A   02AF        BCS    DWERR2
0295 86    08        DWR4   LDA    #TRANS
0297 17    FEF2 018C        LBSR   HSET
029A 86    04        DWR3   LDA    #RES
029C 17    FED0 016F        LBSR   CHEKH
029F 25    F9   029A        BCS    DWR3
02A1 A6    5F               LDA    ESTORE,U
02A3 26    04   02A9        BNE    DWR5      ERROR
02A5 1C    FE               ANDCC  #$FE
02A7 20    02   02AB        BRA    DWR6
02A9 1A    01        DWR5   SEC
02AB 32    62        DWR6   LEAS   2,S
02AD 35    D6               PULS   A,B,X,U,PC

02AF 86    FF        DWERR2 LDA    #$FF
02B1 A7    5F               STA    ESTORE,U
02B3 20    E0   0295        BRA    DWR4


                     *WRITEA*
                     *WRITE THE NUMBER IN X AS AFIELD*
                     *HEX DIGITS MOST SIGNIFICANT*
                     *FIRST*
                     *NO ERROR RETURNS FROM HERE*
02B5 34    10        WRITEA PSHS   X         SAVE FOR RETURN
02B7 B6    06A6             LDA    AFIELD
02BA 81    02               CMPA   #$02
02BC 27    0A   02C8        BEQ    WRIT1
****WARNING   001--00437
02BE 17    FF93 0254        LBSR   SWRITE    SET UP FOR WRITE
02C1 A6    E4               LDA    ,S        MOST SIG BYTE
02C3 84    0F               ANDA   #$F       MASK FOR RH NIBBLE
****WARNING   001--00652
02C5 17    FF9D 0265        LBSR   DWRITE    AND WRITE IT
02C8 A6    61        WRIT1  LDA    1,S
02CA 44                     LSRA
02CB 44                     LSRA
02CC 44                     LSRA
02CD 44                     LSRA             GET LH NIBBLE TO RIGHT
****WARNING   001--00655
02CE 17    FF94 0265        LBSR   DWRITE    AND DISPOSE OF IT
02D1 A6    61               LDA    1,S       RH NIBBLE, LS BYTE
****WARNING   001--00661
02D3 17    FF8F 0265        LBSR   DWRITE    AND DISPOSE OF THAT
02D6 35    90               PULS   X,PC      HURRAY
                     *DREAD*
                     *READ ONE BYTE FROM PROGRAMMER*
```

```
                                 *IN LIST MODE*


02D8 108E F538          DREAD   LDY     #PIA
02DC 86   08                    LDA     #TRANS
02DE 17   FE9F 0180             LBSR    LSET       INITIATE THE READ
02E1 86   04            DREAD1  LDA     #RES
02E3 17   FE78 015E             LBSR    CHEKL
02E6 25   F9   02E1             BCS     DREAD1
02E8 E6   A4                    LDB     DRA,Y
02EA 86   08                    LDA     #TRANS
02EC 17   FE9D 018C             LBSR    HSET
02EF 86   04            DREAD2  LDA     #RES
02F1 17   FE7B 016F             LBSR    CHEKH
02F4 25   F9   02EF             BCS     DREAD2
02F6 39                         RTS


                        *ASIZE*
                        *ALTER DEFAULT SIZE ATTRIBUTES*
                        *SETS UP THE BYTE MASK,TOP ADDRESS AND NUMBER OF CH
02F7 30   8D 0229       ASIZE   LEAX    STR13,PCR
02FB BD   D2A6                  JSR     ZOUTST
02FE BD   D2B5                  JSR     ZLINEI
0301 BD   D2A0                  JSR     ZGETHN
0304 5D                         TSTB
0305 26   09   0310             BNE     ASIZE1     WAS CR SO DEFAULTS
0307 86   03            ASIZE2  LDA     #$03
0309 8E   01FF                  LDX     #$01FF
030C C6   0F                    LDB     #$0F
030E 20   1F   032F             BRA     ASIZE3
0310 8C   0003          ASIZE1  CMPX    #$0003
0313 27   F2   0307             BEQ     ASIZE2
0315 8C   0002                  CMPX    #$0002
0318 26   09   0323             BNE     ASIZE4
031A 86   02                    LDA     #$02
031C 8E   00FF                  LDX     #$00FF
031F C6   0F                    LDB     #$0F
0321 20   0C   032F             BRA     ASIZE3
0323 8C   0001          ASIZE4  CMPX    #$0001
0326 26   11   0339             BNE     ASIZE5
0328 86   02                    LDA     #$02
032A 8E   001F                  LDX     #$001F
032D C6   FF                    LDB     #$FF
032F B7   06A6          ASIZE3  STA     AFIELD
0332 BF   06A3                  STX     HIGH
0335 F7   06A5                  STB     MASK
0338 39                         RTS
0339 8C   0004          ASIZE5  CMPX    #$0004
033C 26   09   0347             BNE     ASIZE6
033E 86   03                    LDA     #$03
0340 8E   07FF                  LDX     #$07FF
0343 C6   FF                    LDB     #$FF
0345 20   E8   032F             BRA     ASIZE3
0347 8C   0005          ASIZE6  CMPX    #$0005
034A 26   BB   0307             BNE     ASIZE2     USE DEFAULTS
034C 86   03                    LDA     #$03
```



```
034E 8E   0FFF                  LDX     #$0FFF
0351 C6   FF                    LDB     #$FF
0353 20   DA   032F             BRA     ASIZE3

                        *DELAY*
                        *DELAYS FOR A LONG TIME*
0355 8E   FFFF          DELAY   LDX     #$FFFF
0358 30   1F            DELAY1  LEAX    -1,X
035A 8C   0000                  CMPX    #$0000
035D 26   F9   0358             BNE     DELAY1
035F 39                         RTS





                        *AMOD*
                        *EXAMINE AND PERHAPS MODIFY A PROM ADDRESS*

          FFFE          PADDS   EQU     -2
          FFFC          TMP1    EQU     -4

0360 1F   43            AMOD    TFR     S,U
0362 32   7A                    LEAS    -6,S       GET VARIABLE SPACE
0364 8E   0000                  LDX     #$0000
0367 AF   5E                    STX     PADDS,U
0369 AF   5C                    STX     TMP1,U
036B 86   2A            AMOD2   LDA     #'*
036D BD   D2BE                  JSR     ZOUTCH
0370 BD   D2B5                  JSR     ZLINEI
0373 BD   D2A0                  JSR     ZGETHN
0376 5D                         TSTB               DID WE GET A HEX NUMBER
0377 26   07   0380             BNE     AMOD1      YES
0379 81   0D                    CMPA    #CR
037B 26   EE   036B             BNE     AMOD2      IF NOT CARRIAGE RETURN REPEA
037D 32   66                    LEAS    6,S
037F 39                         RTS                OTHERWISE RETURN

0380 BC   06A3          AMOD1   CMPX    HIGH
0383 25   03   0388             BLO     AMODA
0385 BE   06A3                  LDX     HIGH
0388 AF   5E            AMODA   STX     PADDS,U    STORE ADDRESS
038A 30   5E                    LEAX    PADDS,U
038C BD   D2AF                  JSR     ZOUTHA     PUT 2 BYTES AS HEX
038F AE   5E                    LDX     PADDS,U
0391 8D   7C   040F             BSR     SETUP
0393 17   FEAD 0243             LBSR    SREAD
0396 86   40                    LDA     #READW
0398 17   FDF1 018C             LBSR    HSET
039B 86   10                    LDA     #MODE
039D 17   FDEC 018C             LBSR    HSET       NOW ALL OK FOR A READ
03A0 17   FF35 02D8             LBSR    DREAD      SO DO IT
03A3 F4   06A5                  ANDB    MASK       PROPER MASK
03A6 E7   5C                    STB     TMP1,U
03A8 86   40                    LDA     #READW
03AA 17   FDD3 0180             LBSR    LSET
03AD 17   FEA4 0254             LBSR    SWRITE     SET UP FOR A WRITE
```

```
05DB        2A                    FCC      '******************'
05EE        0D                    FCB      $0D,$0A
05F0        00                    FCB      $00
05F1        20           STR16    FCC      / 77/
05F4        0D                    FCB      $0D
05F5        0A                    FCB      $0A
05F6        00                    FCB      00
05F7        0000         DTBEND   FDB      $0000
05F9        00A6         RFCB     RMB      166
069F        0000         TOP      FDB      $0000
06A1        0002         BOT      RMB      2
06A3        0002         HIGH     RMB      2
06A5        0001         MASK     RMB      1
06A6        0001         AFIELD   RMB      1
06A7        0002         OPBYTE   RMB      2
06A9        0200         DTABLE   RMB      $200
                                  END

TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00008--00663
```

2) Computer Communications Programs.

```
03B0 86   20                 LDA   #$20       SPACE
03B2 BD   D2BE               JSR   ZOUTCH     PUT IT
03B5 1F   31                 TFR   U,X
03B7 30   1C                 LEAX  TMP1,X     GET ADDRESS OF BYTE
03B9 BD   D2AC               JSR   ZOUTHX     PUT AS TWO HEX DIGITS
03BC 86   20                 LDA   #$20
03BE BD   D2BE               JSR   ZOUTCH
03C1 BD   D2B5               JSR   ZLINEI
03C4 BD   D2A0               JSR   ZGETHN
03C7 5D                      TSTB             ANY HEX DIGITS?
03C8 26   1E   03E8          BNE   AMOD5      YES SO GO WRITE
03CA 81   2E                 CMPA  #'.        NO
03CC 27   9D   036B          BEQ   AMOD2      START AGAIN
03CE 81   5E                 CMPA  #'^
03D0 26   0B   03DD          BNE   AMOD6
03D2 AE   5E                 LDX   PADDS,U
03D4 8C   0000               CMPX  #$0000
03D7 27   A7   0380          BEQ   AMOD1
03D9 30   1F                 LEAX  -1,X       DECREMENT LOCATION COUNTER
03DB 20   A3   0380          BRA   AMOD1
03DD AE   5E         AMOD6   LDX   PADDS,U
03DF BC   06A3               CMPX  HIGH
03E2 27   9C   0380          BEQ   AMOD1
03E4 30   01                 LEAX  1,X        INCREMENT X
03E6 20   98   0380          BRA   AMOD1
03E8 AF   5C         AMOD5   STX   TMP1,U
03EA AE   5E                 LDX   PADDS,U
03EC 8D   21   040F          BSR   SETUP
03EE A6   5D                 LDA   TMP1+1,U GET LSB OF DATA TO BE WRITTE
03F0 17   FE72 0265          LBSR  DWRITE     WRITE IT
03F3 24   0C   0401          BCC   AMOD7
03F5 30   8D 01F8            LEAX  STR16,PCR
03F9 BD   D2A6               JSR   ZOUTST
03FC AE   5E                 LDX   PADDS,U
03FE 16   FF7F 0380          LBRA  AMOD1
0401 AE   5E         AMOD7   LDX   PADDS,U
0403 BC   06A3               CMPX  HIGH
0406 1027 FF76 0380          LBEQ  AMOD1
040A 30   01                 LEAX  1,X        OTHERWISE INCREMENT IT
040C 16   FF71 0380          LBRA  AMOD1

                     *SETUP*
                     *INITS PROGRAMMER*
                     *SENDS ADDRESS OF BYTE TO BE READ*
                     *LEAVES PROGRAMMER IN A WRITE STATE*

040F 17   FE26 0238  SETUP   LBSR  PRINIT
0412 86   01                 LDA   #ADD
0414 17   FD47 015E  SETUP1  LBSR  CHEKL
0417 25   FB   0414          BCS   SETUP1
0419 86   10                 LDA   #MODE
041B 17   FD62 0180          LBSR  LSET
041E 17   FE94 02B5          LBSR  WRITEA     WRITES OUT THE ADDRESS IN X
0421 17   FE91 02B5          LBSR  WRITEA     TWICE FOR A SINGLE BYTE
0424 86   01                 LDA   #ADD
0426 17   FD46 016F  SETUP2  LBSR  CHEKH
0429 25   FB   0426          BCS   SETUP2
```



```
042B 39                      RTS


                     *STORAGE*
042C      2D         CONST   FCC   /-> /
042F      00                 FCB   00
0430      50         STR1    FCC   /PROM START ADDRESS  ? /
0446      00                 FCB   00
0447      50         STR2    FCC   /PROM END ADDRESS ? /
045A      00                 FCB   00
045B      44         STR3    FCC   /DATA TABLE START ADDRESS ? /
0476      00                 FCB   00
0477      41         STR4    FCC   /ADDRESS FIELD ERROR/
048A      0D                 FCB   $0D
048B      0A                 FCB   $0A
048C      00                 FCB   00
048D      44         STR5    FCC   /DATA ERROR /
0498      0D                 FCB   $0D,$0A
049A      00                 FCB   00
          0477       STR6    EQU   STR4
049B      57         STR7    FCC   /WRITE DATA ERROR/
04AB      0D                 FCB   $0D,$0A
04AD      00                 FCB   00
04AE      4E         STR8    FCC   /NON ZERO FIELD ERROR/
04C2      0D                 FCB   $0D,$0A
04C4      00                 FCB   00
04C5      50         STR9    FCC   /PROGRAMMING ERROR/
04D6      0D                 FCB   $0D,$0A
04D8      00                 FCB   00
04D9      42         STR10   FCC   /BINARY FILE NAME? /
04EB      00                 FCB   00
04EC      45         STR11   FCC   /EOF FOUND BEFORE END OF DATA BUFFER/
050F      0D                 FCB   $0D,$0A
0511      00                 FCB   00
0512      44         STR12   FCC   /DATA TABLE SIZE? /
0523      00                 FCB   00
0524      53         STR13   FCC   /SELECT NEW SIZE ATTRIBUTES/
053E      0D                 FCB   $0D,$0A
0540      30                 FCC   /01 = 32 * 8/
054B      0D                 FCB   $0D,$0A
054D      30                 FCC   /02 = 256 * 4/
0559      0D                 FCB   $0D,$0A
055B      30                 FCC   /03 = 512 * 4 (DEFAULT)/
0571      0D                 FCB   $0D,$0A
0573      30                 FCC   /04 = 2056 * 8 /
0581      0D                 FCB   $0D,$0A
0583      30                 FCC   /05 = 4096 * 8 /
0591      0D                 FCB   $0D,$0A
0593      00                 FCB   00
0594      53         STR14   FCC   /SIZE DEFAULTS TO 512 * 4/
05AC      0D                 FCB   $0D,$0A
05AE      0000               FDB   00
05B0      1A         STR15   FCB   $1A
05B1      2A                 FCC   '*******************'
05C4      0D                 FCB   $0D,$0A
05C6      2A                 FCC   /* PROM PROGRAMMER */
05D9      0D                 FCB   $0D,$0A
```

```
                         *
                         *  SIXTH OPERATING SYSTEM FOR 68000
                         *  AUGUST 1981
                         *  REVISION 2.1
                         * DAVE COWAN
                         *
00200                         ORG 1000
01000   264C             REST     DC.W INTE1     RESTART VECTOR
01002   0003FF01         ACIA1     DC.L $3FF01     PORT1 ( TERMINAL )
01006   0003FF21         ACIA2     DC.L $3FF21     PORT2 ( HOST )
0100A   00021C2E         OUT    DC.L $21C2E
0100E   00021DB4         IN     DC.L $21DB4
01012   00020008         MACS      DC.L $20008
                         *
                         * VARIABLES USED
                         *
01016   00002000         MSTCK DC.L $2000
0101A   00001300         DOSTK DC.L $1300
0101E   0040             PREF      DC.B $40      PREFIX CHARACTER
01020   1100             STBUF       DC.W $1100      BUFFER START
01022   0000             BWORD       DC.W $0          START WORD POINTER
01024   0000             EWORD       DC.W $0          END WORD POINTER
01026   263A             DLAST       DC.W XSWAP     LAST DEFINITION POINTER
01028   0000             LAST      DC.W $0          LAST WORD FLAG
0102A   0000             RDX      DC.W $0          CURRENT RADIX
0102C   0000             NFLAG       DC.W $0          NUMBER FLAG
0102E   2800             DP      DC.W $2800      NEXT FREE LOCATION
001030  0000             STATE       DC.W $0          SYSTEM STATE
001032  00001900         OPSTK       DC.L $1900      OP STACK
001036  0000             RELO      DC.W $0          RELOAD FLAG
001038  0003FF01         PORT      DC.L       $3FF01
00103C  00000000         CTIME       DC.L $0000      SPACE FOR PTM
001040  0000                   DC.W $0000
                         *
                         *     ACTUAL PROGRAM START
                         *
001042                          ORG 2000
                         *
                         *
002000  0004             XCHIN       DC.B $4
002002  4348494E               DC.L 'CHIN'
002006  0000                   DC.W 0
                         *
                         * CHARACTER INPUT ROUTINE
                         *
002008  2278100E         SIN      MOVE.L IN,A1
00200C  20781038               MOVE.L PORT,A0
002010  4E91                   JSR (A1)
002012  4E75                   RTS
                         *
                         *
002014  0005             XOUT       DC.B $5
002016  43484F55               DC.L 'CHOU'
00201A  2000                   DC.W XCHIN
                         *
                         * CHARACTER OUTPUT ROUTINE
                         *
00201C  2278100A         SOUT       MOVE.L OUT,A1
002020  20781038               MOVE.L PORT,A0
002024  4E91                   JSR (A1)
002026  4E75                   RTS
```

```
002028  0004             XMAC       DC.B $4
00202A  4D414353               DC.L 'MACS'
00202E  2014                   DC.W XOUT
                         *
                         * MACSBUG REENTRY POINT
                         *
002030  22781012         MAC      MOVE.L MACS,A1
002034  4ED1                   JMP (A1)
                         *
                         *
002036  0006             XBUF       DC.B $6
002038  42554646               DC.L 'BUFF'
00203C  2028                   DC.W XMAC
                         *
                         * SYSTEM-USER IO ROUTINE
                         *
00203E  4EB8209A         BUFF       JSR CRLF
002042  31F810201024           MOVE.W STBUF,EWORD      SET UP PARAMS FOR WORD
002048  11FC00001028           MOVE $0,LAST      CLEAR END OF BUFFER FLAG
00204E  3038101E               MOVE.W PREF,D0      PREFIX CHARACTER
002052  4EB8201C               JSR SOUT      SEND PREFIX CHAR
002056  34781020               MOVE.W STBUF,A2      START OF BUFFER POINTER
00205A  4EB82008         NEXT       JSR SIN            GET A CHARACTER
00205E  0C00000D               CMPI $$0D,D0      CR ?
002062  66000006               BNE BSP            NO
002066  14C0                   MOVE D0,(A2)+      STORE IT
002068  4E75                   RTS             KILL IT
00206A  0C000008         BSP      CMPI $$08,D0      BACKSPACE
00206E  66000010               BNE NPT            NO
002072  4EB8201C               JSR SOUT      ECHOE IT
002076  1222                   MOVE -(A2),D1      BACK OFF
002078  B4F81020               CMPA.W STBUF,A2      BACK AT START ?
00207C  6F00FFC0               BLE BUFF      RESTART
002080  0C000020         NPT      CMPI $' ',D0      LOOK AT NON PRINTING
002084  6D00FFD4               BLT NEXT      IF SO IGNORE
002088  4EB8201C               JSR SOUT      ECHOE THE CHARACTER
00208C  14C0                   MOVE D0,(A2)+      STORE IT
00208E  4EF8205A               JMP NEXT      AND AGAIN PLEASE !!!!
                         *
                         *
002092  0004             XCRLF       DC.B $4
002094  43524C46               DC.L 'CRLF'
002098  2036                   DC.W XBUF
                         *
                         * SIMPLE CRLF ROUTINE
                         *
00209A  303C000D         CRLF       MOVE.W $$0D,D0
00209E  4EB8201C               JSR SOUT
0020A2  303C000A               MOVE.W $$0A,D0
0020A6  4EB8201C               JSR SOUT
0020AA  4E75                   RTS
                         *
                         *
0020AC  0004             XWORD       DC.B $4
0020AE  574F5244               DC.L 'WORD'
0020B2  2092                   DC.W XCRLF
                         *
                         * SETS WORD POINTERS FOR FIND
                         *
0020B4  31F810241022     WORD       MOVE.W EWORD,BWORD      SET END = BEGINNING
0020BA  34781022               MOVE.W BWORD,A2            TO SCAN
0020BE  0C1A0020         SPACE       CMPI $' ',(A2)+            SPACE PERHAPS
0020C2  6700FFFA               BEQ SPACE
```

2) SIXTH dictionary.

```
1000 CONSTANT EQUATES
EQUATES 2 + CONSTANT ACI1
ACI1 4 + CONSTANT ACI2
ACI2 10 + CONSTANT MSTCK
MSTCK 4 + CONSTANT DOSTK
DOSTK 4 + CONSTANT PREFIX
PREFIX 2 + CONSTANT ST
ST 2 + CONSTANT WB
WB 2 + CONSTANT WE
WE 2 + CONSTANT DL
DL 2 + CONSTANT LAST
LAST 2 + CONSTANT RDX
RDX 2 + CONSTANT NFLAG
NFLAG 2 + CONSTANT DP
DP 2 + CONSTANT STATE
STATE 2 + CONSTANT OPSTK
OPSTK 4 + CONSTANT FRELO
FRELO 2 + CONSTANT PORT
PORT 4 + CONSTANT CTIME
0 CONSTANT 0
1 CONSTANT 1
2 CONSTANT 2
3 CONSTANT 3

: OCT 8 RDX !W ;
: HEX 10 RDX !W ;
: DEC A RDX !W ;
: DUP POP PUSH PUSH ;
: DROP POP ;
: OVER SWAP DUP POP STK SWAP UNST PUSH ;
: ROT SWAP POP STK SWAP UNST PUSH ;
: HERE DP @W ;
: 1- 1 - ;
: 1+ 1 + ;
: ! !W ;
: @ @W ;
: '1 DUP @ 1+ SWAP ! ;
: DP! DP @ ! DP @ 2 + DP ! ;
: WNUM WORD NUMBER ;
: L9 7 LEFT 2 LEFT ;
: WNUM2 WNUM + WNUM L9 + DP! ;


: MOVE IMMEDIATE 2000 WNUM2 ;
: SUB IMMEDIATE 9080 WNUM2 ;
: SUBW IMMEDIATE 9040 WNUM2 ;
: AND IMMEDIATE C080 WNUM2 ;
: PSHS IMMEDIATE 2F00 DP! ;
: PULS IMMEDIATE 201F DP! ;
: MASK POP MOVE 0 1 POP AND 1 0 PUSH ;
: B2 6000 + DP! HERE DUP 0 DP! ROT POP MOVE 0 1 POP SUBW 1 0 PUSH SWAP ! ;
: BRA IMMEDIATE 0000 B2 ;
: BEQ IMMEDIATE 0700 B2 ;
: BNE IMMEDIATE 0600 B2 ;
: BLE IMMEDIATE 0F00 B2 ;
: BGT IMMEDIATE 0E00 B2 ;
: BGE IMMEDIATE 0C00 B2 ;
: BPL IMMEDIATE 0A00 B2 ;
: BLT IMMEDIATE 0D00 B2 ;
: RTE IMMEDIATE 4E73 DP! ;
: FRAME IMMEDIATE 48E7 DP! FFFE DP! ;
: SYRES IMMEDIATE 4E70 DP! ;
: ENINT IMMEDIATE 027C DP! F8FF DP! ;
: DISINT IMMEDIATE 007C DP! 0700 DP! ;


: #INT IMMEDIATE INTEGER ;
: CALL 4EB8 DP! ;
: # IMMEDIATE 4EB8 DP! WORD FIND 8 + DP! ;
: ## IMMEDIATE WORD FIND 8 + ;
: => IMMEDIATE CALL ## CALL #INT DP! WORD FIND 8 + ;
: <= IMMEDIATE INTEGER => DP! #INT DP! ;
: <DO POP STK POP STK POP STK ;
: 'LOOP UNST UNST UNST ;
: DO IMMEDIATE 2D3C DP! 0 DP! HERE 0 DP! => <DO <= HERE ;
: <LOOP UNST PUSH MOVE 0 5 UNST PUSH SUB 0 5 1 + POP STK POP STK MOVE 5 0
: LOOP IMMEDIATE => <LOOP <= # BGT HERE SWAP ! => 'LOOP <= ;
: ABORT 'LOOP PULS ;
: NEXT PULS UNST PSHS UNST PSHS UNST PUSH STK PULS STK PULS STK 8 - POP PS
: STOP PULS UNST PUSH UNST PUSH UNST PSHS STK POP STK POP STK ;
: QUIT PULS ;
: I UNST PSHS UNST PUSH STK PULS STK ;
: SK IMMEDIATE WNUM HERE + ;
: = POP MOVE 0 1 POP SUB 1 0 SK C BEQ 0 SK 8 BRA 1 POP ;
: > POP MOVE 0 1 POP SUB 1 0 SK C BGT 0 SK 8 BRA 1 POP ;
: < POP MOVE 0 1 POP SUB 1 0 SK C BLT 0 SK 8 BRA 1 POP ;
: , CRLF . ;
: BASE RDX @ DUP DEC , RDX ! ;
: =0SKIP MOVE 0 1 HERE 8 + # BEQ 4EF8 DP! 0 DP! ;
: >0SKIP MOVE 0 1 HERE 8 + # BGT 4EF8 DP! 0 DP! ;
: IF IMMEDIATE >0SKIP HERE 2 - ;
: THEN IMMEDIATE HERE SWAP ! ;
: ELSE IMMEDIATE 4EF8 DP! 0 DP! # THEN HERE 2 - ;
: =< OVER OVER < IF DROP DROP ELSE = THEN ;
: >= OVER OVER > IF DROP DROP ELSE = THEN ;
: <> OVER OVER < IF SWAP THEN ;
: BYTE DUP @B SWAP 1 + SWAP ;
: ( IMMEDIATE WB @ DUP 100 1 DO BYTE 29 = IF DROP I + WE ! ABORT
                                            THEN LOOP DROP DROP ;
: ARRAY IMMEDIATE 2 * DUP DUP 2D3C DP! 0 DP! DP!
              2D3C DP! 0 DP! HERE 6 + DP! HERE 4 + +
                # BRA DP @ + DP ! ; ( PUTS SIZE,ADDX ON STACK )
CTIME 6 + CONSTANT GT
   GT 4 + CONSTANT ST2
ST2 4 + CONSTANT WIDTH
: DELIM WORD WB @ DUP GT ! GT '1 @B ;
: FILL 0 WIDTH ! DP @ 10 + ST2 ! DELIM 100 1 DO GT @ @B GT '1
 OVER OVER = IF DROP DROP WIDTH @ GT @ WE ! 0 ST2 @ !B ABORT
 THEN WIDTH '1 ST2 @ !B ST2 '1 LOOP ;    ( FILLS AFTER THE RTS )
: STRING IMMEDIATE FILL 1 + 2 / # ARRAY ;
: LOC WORD FIND ;      ( LOCATES A WORD IN THE DICTIONARY )
: SPACE 20 TO ;       ( TYPE 1 SPACES )
: LIST DO I @B . SPACE LOOP ;    ( MEMORY DUMP )
: SAY TYPE ;
: STR1 STRING %THERE ARE % ;
: STR2 STRING % DEFINITIONS % ;
: DEFN DUP @B . SPACE 1 + DUP @B . 1 + SPACE ;
: NAME 3 0 DO DUP I + @B TO LOOP ;
: STR3 STRING %NOT FOUND % ;
: STR4 STRING %DEFINITION KEPT% ;
: BEGIN IMMEDIATE HERE ;
: END IMMEDIATE 4EF8 DP! DP! ;

: LOCATE IMMEDIATE LOC ;
```

```
: INTER  PULS EQUATES @ POP PSHS ;    ( JMP TO INTERPRET LOOP )
: MACSBUG PULS 2013A POP PSHS ; ( JMPS STRAIGHT TO MACS CLI )
: SERROR CRLF STRING %ERROR% SAY CRLF  INTER ; ( TRAP ERROR )
: SABORT CRLF STRING %ABORT% SAY
 CRLF INTER ;     ( ABORT BUTTON HANDLER )
: .K VARIABLE ;
: KEEP WORD FIND .K ! ;     ( WRITE PROTECTS DICTIONARY SPACE )
: WHAT CRLF DL @ 1000 1 DO DUP DUP . SPACE DEFN NAME SWAP
    .K @ = IF DROP ABORT THEN 4 + @ CRLF DUP 0 = IF STR1 SAY
      RDX @ DEC I . RDX ! STR2 SAY
        DROP ABORT THEN LOOP DROP ;
: FORGET WORD FIND DUP 0 = IF
      CRLF DROP STR3 SAY ELSE DUP .K @ =< IF
        CRLF DROP STR4 SAY ELSE DUP DP ! 6 + @ DL !
          THEN THEN ;   ( FORGETS ALL DEFNS UP TO THE SELECTED ONE )

( INTERRUPT STUFF )
: INSTALL 4 * 60 + LOC B + SWAP !L ;

: STORE VARIABLE ;
: SETUP SYRES
 DISINT 0 STORE ! LOCATE SERROR #INT B +
     100 1 DO DUP STORE @ 4 + DUP STORE !
        !L LOOP DROP LOCATE SABORT
           #INT B + 7C !L       ( LOADS UP VECTOR AREA )
 ;


( ACIA SWAPPING STUFF )

: CHECK BEGIN CHIN PUSH 02 = IF ABORT THEN END ;
: ACL 3 OVER !B 15 SWAP !B ;
: A1 ACI1 @L PORT !L ;
: A2 ACI2 @L PORT !L ;
: STR WORD LENGTH PUSH WB @ ;
: T1 ACI1 @L ACL ;
: T2 ACI2 @L ACL ;
: TRANS ACI1 @L 55 OVER !B FD SWAP 2 + !B ;
: P2 TRANS CHECK T1 T2 ;


( NEW OPEN COMMAND STUFF FOR ANY FILE )

: COM_LINE STR STRING &1:LD  & A2 TYPE TYPE OD TO ;
: WAIT BEGIN CHIN PUSH OD = IF ABORT THEN END ;
: REP_TEST BEGIN CHIN PUSH DUP 41 = IF POP 1 ABORT ELSE 58 = IF 0 ABORT THEN THE
: OPEN COM_LINE WAIT REP_TEST A1 DUP 0 = IF STRING & FAILED & SAY THEN ;

: █ A1 STRING %EOF FOUND% SAY RESTART ;    ( RUBOUT COMMAND FOR EOF )
: %ENDFILE 0 FRELO ! A2 1B TO A1 ;
: COMPILE OPEN 0 = IF %ENDFILE ABORT THEN RELOAD ;
SETUP

RESTART
```

Chapter 5 Listings

1) Portable Controller Program.

```
( HIGHWAY CONTROLLER TABLES )

 ( PRIMARY TABLE )

800 CONSTANT PT                  ( PRIMARY TABLE POINTER )
PT CONSTANT CTUSW                ( CONTROLLER TERMINAL UNIT STATUS WORD )
CTUSW 2 + CONSTANT CTUCW         ( CONTROLLER TERMINAL UNIT CONTROL WORD )
CTUCW 2 + CONSTANT PTPT          ( POINTER TO POLLING TABLE )
PTPT 2 +  CONSTANT PTBS          ( POINTER TO BUFFER STORE )
PTBS 2 +  CONSTANT PTSS          ( POINTER TO STATUS STORE )
PTSS 2 +  CONSTANT PTST          ( POINTER TO STATUS STORE )
PTST 2 +  CONSTANT STSP          ( SELF TEST SCRATCHPAD )
STSP 4 +  CONSTANT REC           ( RECEIVE ERROR COUNTER )
REC 2 +   CONSTANT RC            ( REPEAT COUNTER )
RC 2 +    CONSTANT NRC           ( NULL REPEAT CONUNTER )
NRC 2 +    CONSTANT OTA          ( OUT TIME AVAILABLE )
OTA 2 +   CONSTANT OT            ( OUT TIME )
PT 66 +    CONSTANT ITA          ( IN TIME AVAILABLE )
ITA 2 +   CONSTANT IT            ( IN TIME )


 ( STATUS TABLE )
C00      CONSTANT STT

 ( SIZE STORE )

A00       CONSTANT MSS


 ( BUFFER STORE )

A000          CONSTANT BS

( POLLING TABLES )
900       CONSTANT PTA           ( POLLING TABLE A )

D00       CONSTANT PTB           ( POLLING TABLE B )


( CHANNEL CONTROL WORD )

3C000 CONSTANT CCW

( CONTROL WORDS )

2 CONSTANT GO                    ( START TERMINAL UNIT )
3 CONSTANT CSTOP                 ( STOP TERMINAL UNIT )
4 CONSTANT OFFINT                ( SWITCH OFF INTERRUPTS )
5 CONSTANT ONINT                 ( SWITCH ON AND CLEAR INTERRUPTS )
7 CONSTANT RESET                 ( RESET AND HALT TERMINAL UNIT )



( FELTEC LCD DRIVERS )

3FF40 CONSTANT PIA     ( FIRST THE RELEVANT PIA REGISTERS )
PIA CONSTANT DRA2
PIA 2 + CONSTANT DRB2
PIA 4 + CONSTANT CRA2
PIA 6 + CONSTANT CRB2
```

```
: OBYTE DUP 3FF42 !B 3FF42 @B OVER = IF DROP ELSE
3FF42 !B 3C 3FF46 !B 7 3FF44 !B THEN 34 CRB2 !B 3C CRB2 !B ;
: CL'FEL 7F 0 DO 20 OBYTE LOOP ;               ( CLEAR ONE ENTIRE BUFFER )
: INIT'FEL 00 CRB2 !B FF DRB2 !B 0C CRB2 !B    ( ALL OUTPUTS )
           88 OBYTE 82 OBYTE CL'FEL 81 OBYTE   ( CLEAR BUFF1 )
           88 OBYTE A2 OBYTE CL'FEL 81 OBYTE   ( CLEAR BUFF2 )
           90 OBYTE 00 OBYTE ;                 ( RESET CURSOR )
: PT' VARIABLE ;              ( BUFFER POINTER FOR FELTEC )
: SCROLL 84 OBYTE OBYTE 82 OBYTE 1F 0 DO 20 OBYTE LOOP 81 OBYTE ;
: PT'FEL PT' DUP @ 7F MASK DUP 1F MASK 0 = IF DUP SCROLL THEN SWAP '1 ;
: T_DIS VARIABLE ;    ( CURRENT DISPLAY TYPE )
: FEL'CL 88 OBYTE 82 OBYTE CL'FEL 81 OBYTE 90 OBYTE 0 OBYTE 0 PT' ! 0 T_
: OUT'FEL PUSH DUP DUP A = IF DROP       ( IGNORE LF )
                         ELSE DUP D = IF DROP PT' @ 20 + 60 MASK PT'
                         ELSE DUP 08 = IF DROP PT' DUP @ 1- SWAP !
                                  90 OBYTE PT' @ OBYTE
                                 ELSE
         PT'FEL 84 OBYTE OBYTE 82 OBYTE OBYTE 81 OBYTE
                  PT' @ 90 OBYTE OBYTE
                                 THEN
                  THEN THEN POP ;    ( LEAVE CHAR IN DO FOR BUFF )
( REAL TIME CLOCK )

3C020 CONSTANT CLOCK      ( ADDRESS 0 OF REAL TIME CLOCK )
: INV -1 SWAP - ;              ( DAT FROM CLOCK IS COMPLIMENTED )
: .. INV F MASK . ;            ( AND ONLY LOWER THREE BITS IS VALID )
: 4INV INV F MASK ;
: CCLEAR 2 * CLOCK + 0 INV SWAP ! ;  ( CLEARS A CLOCK LOCATION )
: CREAD 2 * CLOCK + @ ;              ( READS FROM A CLOCK LOCATION )
: STOPCLOCK 0 INV CLOCK E 2 * + ! ;
: GOCLOCK 0 INV CLOCK ! 1 INV CLOCK E 2 * + ! ;
: CSTORE 2 * CLOCK + SWAP INV SWAP ! ; ( STORES DATA AT CLOCK LOCN )
: STOD STOPCLOCK 1 0 DO BUFFER WNUM C I - CSTORE LOOP
      5 0 DO BUFFER WNUM 9 I - CSTORE LOOP GOCLOCK ; ( SET A NEW TIME )
: 4INV INV F MASK ;
: _CREAD 10 1 DO DUP CREAD DUP 4INV F MASK F < IF SWAP DROP ABORT
     THEN DROP LOOP SWAP DROP ;
: GTOD 9 1 DO I _CREAD 4INV LOOP C B DO I _CREAD 4INV LOOP ;
: TIME GTOD 5 1 DO . . 3A TO LOOP . ;




( HIGHWAY CONTROLLER DRIVERS )

(      STATUS TABLE FIELDS     )
: ●NAK 2 * STT + @ 4000 MASK 4000  / ;     ( STACKS NAK BIT FOR A TERMINA
: ●NR 2 * STT + @ 8000 MASK 8000  / ;     ( STACKS NR BIT FOR A TERMINAL
: ●IMM 2 * STT + @ FF MASK ;        ( STACKS INFORMATION MONITOR )
: ●EM 2 * STT + @ 3F00 MASK 100 / ; ( STACKS ERROR MONITOR )

( GET ALL STATUS ON A PARTICULAR TERMINAL )
: ●GTSTATS DUP ●IMM SWAP DUP ●EM SWAP DUP ●NR SWAP ●NAK ;
                         ( ORDER OFF STACK -> NAK,NR,EM,IMM )


( CONTROLLER STATUS )

: ●STOP CTUSW @ 1 MASK  ;  ( STACKS START/STOP BIT )
: ●ACTIVE CTUSW @ 2 MASK 2 / ; ( STACKS ACTIVE/PASSIVE BIT )
: ●OGP CTUSW @ 4 MASK 4 / ; ( STACKS OVERRIDE GO PASSIVE BIT )
```

Chapter 6 Listings

1) Master terminal Unit Program.

```
( NEW MASTER UNIT SOFTWARE )
( DESTINED FOR THE SHIP TRIALS )
( JANUARY 1982 )
( INCLUDES REVISION TO ALLOW USE OF MTB EXTENSION BIT )
( INCLUDES BOTH BLOCK AND SHORT MESSAGE TESTS )
( REVISION 2.1 10/1/82 )
( LONGER REPORTS TO ALLOW DECENT STATUS )
( REPORTS IN SMST )

( STOREIT AND DOIT INCLUDED IN THIS ONE )

( TERMINAL UNIT PRIMARY TABLE )


A00 CONSTANT PR_TAB
PR_TAB CONSTANT IN_INT
IN_INT 1 + CONSTANT IN_NO
IN_NO  2 + CONSTANT IN_POS
IN_POS 1 + CONSTANT IN_TAB
IN_TAB 2 + CONSTANT O_INT
O_INT 1 + CONSTANT O_NO
O_NO  2 + CONSTANT O_POS
O_POS 1 + CONSTANT O_TAB
O_TAB 2 + CONSTANT MES_TAB
MES_TAB 40 + CONSTANT HW_NO
HW_NO   2 + CONSTANT RE_COUNT
RE_COUNT 2 + CONSTANT DAT_STARV
DAT_STARV 2 + CONSTANT RETR_COUNT
RETR_COUNT 2 + CONSTANT BUF_OVER
BUF_OVER   2 + CONSTANT IN_BLK_STAT
IN_BLK_STAT     2 + CONSTANT IN_BLK_SOURCE
IN_BLK_SOURCE  2 + CONSTANT IN_BLK_TOTAL
IN_BLK_TOTAL 1 + CONSTANT IN_BLK_TOT_RECVD
IN_BLK_TOT_RECVD 1 + CONSTANT IN_BLK_ADDRESS
IN_BLK_ADDRESS 2 + CONSTANT O_BLK_STAT
O_BLK_STAT      3 + CONSTANT O_BLK_DESTIN
O_BLK_DESTIN      1 + CONSTANT O_BLK_TOT
O_BLK_TOT 1 + CONSTANT O_BLK_TOT_TXD
O_BLK_TOT_TXD 1 + CONSTANT O_BLK_START

( END OF PRIMARY TABLE )

( IN TABLE )
( MASK FOR A SINGLE BUFFER AREA )

    ( RELATIVE TO START OF BUFFER )
1  CONSTANT IN_BUF_LEN
IN_BUF_LEN 1 + CONSTANT IN_DEST
IN_DEST   1 + CONSTANT IN_SOURCE
IN_SOURCE 1 + CONSTANT IN_TYPE
IN_TYPE 2 + CONSTANT IN_DAT_BUF



( OUT TABLE )
( MASK FOR A SINGLE BUFFER AREA )

    ( RELATIVE TO START OF BUFFER )
1 CONSTANT O_BUF_LEN
O_BUF_LEN 1 + CONSTANT O_DEST
O_DEST     2 + CONSTANT O_TYPE
O_TYPE     2 + CONSTANT O_DAT_BUF
```

```
: TABLE IMMEDIATE 2D3C DP! 0 DP! DP! 2D3C DP! 0 DP! DP! ;
: ( ;
: ) ROT * + + ;

( CHANNEL CONTROL WORD )

3C000 CONSTANT CCW

( CONTROL WORDS )

2 CONSTANT GO               ( START TERMINAL UNIT )
3 CONSTANT CSTOP            ( STOP TERMINAL UNIT )
4 CONSTANT OFFINT           ( SWITCH OFF INTERRUPTS )
5 CONSTANT ONINT            ( SWITCH ON AND CLEAR INTERRUPTS )
7 CONSTANT RESET            ( RESET AND HALT TERMINAL UNIT )


( BITS TO CONTROL TERMINAL UNIT )


40 5900 : IN_TABLE TABLE ;
40 5D00 : OUT_TABLE TABLE ;
: OUT_BUF_NO VARIABLE ; ( 68K'S RECORD OF NEXT FREE BUFFER )
: GET_BUF OUT_BUF_NO DUP @ 1 + F MASK SWAP ! ;
: CL2 DO 0 I !B LOOP ;

( MESSAGE SEND ROUTINE )
: SEND OUT_TABLE ( OUT_BUF_NO @ O_BUF_LEN ) @B
      0 = IF        ( BUFFER IS FREE SO CARRY ON )
          OUT_TABLE ( OUT_BUF_NO @ 5 ) DUP 5 - CL2
          OUT_TABLE ( OUT_BUF_NO @ O_TYPE ) ! ( MESSAGE TYPE )
          OUT_TABLE ( OUT_BUF_NO @ O_DEST ) !B ( DESTINATION )
          OVER 0 DO DUP I + @B
                    OUT_TABLE ( OUT_BUF_NO @ O_DAT_BUF I + ) !B LOOP DROP
          1 + 2 / 3 + 3F MASK OUT_TABLE ( OUT_BUF_NO @ O_BUF_LEN ) !B
          GET_BUF
          ELSE POP POP POP POP
          THEN
;

( BLOCK RECEIVE AND TRANSMIT ROUTINES )

: TOT&REM                     ( GET SUB-BLOCK TOTAL AND REMAINDER )
        1 + 2 / FFFF MASK           ( WORD COUNT )
        20 / DUP SWAP    ( SUB-BLOCK TOTAL )
        3F 10 LEFT MASK
        80 / 200 /              ( REMAINDER )
;

: BLK_SEND O_BLK_STAT @ 8000 MASK 0 = IF
        O_BLK_START O_BLK_STAT CL2          ( CLEAR UP TABLE )
        O_BLK_DESTIN !B                              ( DESTINATION )
        2 / O_BLK_START !                           ( START ADDRESS )
        TOT&REM
        O_BLK_STAT !
        O_BLK_TOT !B
        80 O_BLK_STAT !B                     ( GO GO GO )
        ELSE DROP DROP DROP ( GET RID OF PARAMS )
        THEN
;

: BLK_REC IN_BLK_STAT @ 8000 MASK 0 = IF
        DROP DROP DROP      ( GET RID OF UNWANTED PARAMS )
        ELSE 0 IN_BLK_STAT 2 + !L           ( CLEAR UP )
```

```
( RUN A TEST )
( ASSUMES ALL UNITS ALREADY SETUP CORRECTLY )

: RUN NSET SMY              ( SEND THE BK.SRC NUMBER & RESET VARIABLES )
     SRUN                   ( TELL THE RECEIVERS TO START )
     RESET'REP              ( SET ALL POINTERS TO ZERO )
     BEGIN                  ( THIS IS WHAT WE CAME FOR )
         8000 0 DO BK.TX LOOP      ( DO A FEW TRANSMISSIONS )
OD TO TEST_BYTE @ .
         BUF?
         END
;

: 52 SETUP ;
: SETUP 52 0 WAITING ! RESET'REP ACI2 @L 2 + @B ;



( SHORT MESSAGE SOAK TESTS )
( ASWE SHIP TRIALS )
( REVISION 1.0 JANUARY 1982 )


: ANALYSE CODE
        9482     SUB.L D2,D2
        2602     MOVE.L D2,D3
        2802     MOVE.L D2,D4
        265E     MOVE.L (A6)+,A3
        245E     MOVE.L (A6)+,A2
        221E     MOVE.L (A6)+,D1
        3601 A   MOVE.W D1,D3
        381B     MOVE.W (A3)+,D4
        B943     EOR.W D4,D3
        670A     BEQ.S B
        5242     ADDQ #1,D2
        B47C     CMP ##9,D2
        0009
        6E02     BGE.S B
        2D03     MOVE.L D3,-(A6)
        5241 B   ADDQ #1,D1
        B5CB     CMP.L A3,A2
        6EE8     BGE.S A
        2D02     MOVE.L D2,-(A6)
        4E75     RTS
        0000
;
: M_BYTE VARIABLE ;            ( MTB, USED TO GENERATE MESSAGES )
BK.START CONSTANT M_START         ( USE SAME SPACE AS BLOCK TEST )
39   CONSTANT M_LENGTH         ( MAXIMUM DATA MESSAGE LENGTH )
M_START M_LENGTH + CONSTANT M_END
: UCOUNT VARIABLE ;        ( COUNT OF UNITS IN THIS TEST )
: FLAG VARIABLE ;

: SME' X_COM %   UNITS! % ;
: SME SME' HW_NO @ DISSECT
     DUP 1 = IF DROP OVER !B
    ELSE 2 = IF ROT OVER 1 + !B SWAP OVER !B THEN THEN
   0 0 SEND
;

: WAIT_A_WHILE 4000 1 DO LOOP ;
```

```
: UNITS! U_POINT [ UCOUNT @ ] !
         UCOUNT '1 ;           ( STORE ANOTHER TERMINAL NUMBER )


: M.GEN M_BYTE @ FF MASK DUP 8 LEFT +   ( FORM GENERATOR )
       M_END M_START BK.GEN ;

: M.SEND DUP M.GEN DROP M_LENGTH M_START
         ROT ROT M_BYTE @ FF MASK
         100 + SEND                  ( SET TYPE EXTENSION BIT )
         M_BYTE '1 ;


: REP'NO VARIABLE ;
B : REP ARRAY ;

: M.SAVE E - 2 * C + REP SWAP DROP + '1 ; ( INC MESSAGE COUNT )


: RX_WAIT 0 FLAG ! 4000 0 DO RECEIVE 0 = IF STOP
                                 ELSE I FLAG !
                                 THEN LOOP ;

: ERRORTOT VARIABLE ;

: CL_REP REP DUP 15 + SWAP DO 0 I !B LOOP DROP ;



: MSTRIP CL_REP AAAA REP [ 1 ] !   ( FLAG ERROR RECORD )
          REP'NO @ REP [ 0 ] !
          A 2 DO REP [ I ] ! LOOP
           REP STOREIT ;




: M.ERRS ERRORTOT @ + ERRORTOT ! MSTRIP ;

: M.ANALYSE FF MASK DUP 8 LEFT +  SWAP
            POP STK ROT SWAP
            OVER + SWAP ANALYSE
            DUP 0 > IF M.ERRS
                   ELSE DROP THEN UNST PUSH ;


: GTIME1 0 PR_TAB 66 + ! 4000 1 DO PR_TAB 66 + @ 8000 = IF STOP THEN LOOP
        2 0 DO PR_TAB 68 + I2 @ OVER I2 ! LOOP DROP ;


: MREPORT                         ( REPORT TO ITSELF )
        REP 2 + GTIME1 DROP
        ERRORTOT @ REP [ 4 ] !    ( UNDETECTED ERRORS )
        RE_COUNT @ REP [ 5 ] !    ( DETECTED ERRORS )
        REP'NO   @ REP [ 0 ] !    ( REPORT NUMBER )
M_BYTE @ REP [ 6 REP'NO @ + ] !    ( TOTAL MESSAGES SENT )
  REP STOREIT                     ( DO THINGS THE EASY WAY )
;

: SETBITS                         ( SET A FEW URGENTLY NEEDED BITS )
HW_NO @ 3F MASK E - REP'NO !
CL_REP 0 ERRORTOT ! 0 M_BYTE ! ;
```

```
: USAY CRLF STRING %TERMINALS IN TEST:- % SAY
      SPACE HW_NO @ . ;

: SMRUN                         ( SAY HELLO TO EVERYONE )
       DISINT
       RESET'REP
       SETBITS
       UCOUNT @ 1 -             ( GET TOTAL NUMBER OF RESPONSES )
       USAY
       0 DO U_POINT [ I ] @ DUP M.SEND
            SPACE . LOOP CRLF
  BEGIN
      LSIZE 1 DO
            RX_WAIT             ( WAIT TO RECEIVE ANYTHING )
            FLAG @ 4000 = IF STOP THEN
       DUP 100 MASK 0 > IF
            M.ANALYSE           ( ANALYSE RECEIVED MESSAGE )
            M.SEND              ( REPLY TO THE SRC TERMINAL )
            M.SAVE              ( SAVE COUNT FOR STATUS REPORTS )
       ELSE PROCESS
            THEN
            LOOP
            MREPORT
            BUF?
      FLAG @ 4000 = IF ENINT QUIT THEN
  END ;

: S2 SETUP ;
: SETUP S2 1FF 100 DO I S_TYP LOOP
SETBITS ;

: SSRUN' X_COM %SMRUN % ;
: SRUN  0 ERRORTOT ! 0 M_BYTE ! 0 UCOUNT ! 0 FLAG ! ;
: SRUN1 SRUN SME ;
: SSCL' X_COM %SRUN1 % ;
: SSCL SRUN SSCL' 0 0 SEND WAIT_A_WHILE SME ;

: SSRUN SMY SSCL WAIT_A_WHILE
       SSRUN' 0 0 SEND
       SMRUN ;

: SSTOP' X_COM %SSTOP %  ;     ( TELL OTHER UNITS TTO STOP SMST )
: SSTOP SSTOP' 0 0 SEND ;

: 1RESET LOCATE INTER $INT 8 + EQUATES ! ;
: _RES RESTART SETUP 1RESET INTER ;
: 2RESET LOCATE _RES $INT 8 + EQUATES ! ;

;ENDFILE
```

2) Slave Terminal Unit Program.

```
            COUNT2 @ 10 > IF QUIT THEN
 END

TOP VARIABLE ;        ( USED TO FLAG A STOP )
 SETUP ;
TBITS HW_NO @ 3F MASK E - REP'NO !
     CL_REP 0 FSTOP !

TUP 52 SETBITS ;


ORT MESSAGE SOAK TESTS )
WE SHIP TRIALS )
VISION 1.0 JANUARY 1982 )

.GEN CODE
    265E     MOVE.L (A6)+,A3
    245E     MOVE.L (A6)+,A2
    2216     MOVE.L (A6),D1
    36C1 ONE MOVE.W D1,(A3)+
    5241     ADDQ.W #1,D1
    B5CB     CMPA.L A3,A2
    6CF8     BGE ONE
    4E75     RTS
    0000

BYTE VARIABLE ;            ( MTB, USED TO GENERATE MESSAGES )
TART CONSTANT M_START          ( USE SAME SPACE AS BLOCK TEST )
 CONSTANT M_LENGTH         ( MAXIMUM DATA MESSAGE LENGTH )
ART M_LENGTH + CONSTANT M_END
OUNT VARIABLE ;       ( COUNT OF UNITS IN THIS TEST )
AG VARIABLE ;

E' X_COM %   UNITS! % ;
E SME' HW_NO @ DISSECT
 DUP 1 = IF DROP OVER !B
 ELSE 2 = IF ROT OVER 1 + !B SWAP OVER !B THEN THEN
) 0 SEND


IT_A_WHILE 4000 1 DO LOOP ;


I_POINT ARRAY ;      ( ARRAY OF TERMINAL NUMBERS OF UNITS IN TEST )

TS! U_POINT [ UCOUNT @ ] !
    UCOUNT '1 ;          ( STORE ANOTHER TERMINAL NUMBER )

EN M_BYTE @ FF MASK DUP 8 LEFT +   ( FORM GENERATOR )
   M_END M_START BK.GEN ;

END DUP M.GEN DROP M_LENGTH M_START
    ROT ROT M_BYTE @ FF MASK
    100 + SEND                   ( SET TYPE EXTENSION BIT )
    M_BYTE '1 ;

AVE E - 2 * C + REP SWAP DROP + '1 ; ( INC RECEIVE COUNT )
```

```
: RX_WAIT 0 FLAG ! 4000 0 DO RECEIVE 0 = IF STOP
                                   ELSE I FLAG !
                                   THEN LOOP ;

: M.ERRS ERRORTOT @ + ERRORTOT ! MSTRIP ;

: M.ANALYSE FF MASK DUP 8 LEFT + SWAP      ( USES MTB EXTRACTED  FROM )
            POP STK ROT SWAP
            OVER + SWAP ANALYSE
            DUP 0 > IF M.ERRS
                    ELSE DROP THEN UNST PUSH ;

: SREPORT                         ( REPORT TO MASTER TERMINAL )
        REP 2 + GTIME1 DROP        ( TIME WORDS )
        ERRORTOT @ REP [ 4 ] !      ( UNDETECTED ERRORS )
        RE_COUNT @ REP [ 5 ] !      ( DETECTED ERRORS )
        REP'NO   @ REP [ 0 ] !      ( REPORT NUMBER )
M_BYTE @ REP [ 6 REP'NO @ + ] !      ( TOTAL TXD MESSAGE COUNT )
  REP BK.SRC @ 3F MASK 1 SEND ;      ( SEND TO MASTER )


200 CONSTANT LSIZE


: SSTOP 1 FSTOP ! ;                 ( FLAG A STOP TO SSRUN LOOP )


: SMRUN                            ( SAY HELLO TO EVERYONE )
       DISINT
       SETBITS
       UCOUNT @ 1 -             ( GET TOTAL NUMBER OF RESPONSES )
       0 DO U_POINT [ I ] @ M.SEND LOOP   ( TRANSMIT MESSAGE TO EACH )
  BEGIN
     LSIZE 1 DO
            RX_WAIT              ( WAIT TO RECEIVE ANYTHING )
            FLAG @ 4000 = IF STOP THEN
       DUP 100 MASK 0 > IF
            M.ANALYSE            ( ANALYSE RECEIVED MESSAGE )
            M.SEND               ( REPLY TO THE SRC TERMINAL )
            M.SAVE
     ELSE PROCESS
     THEN
         FSTOP @ 1 = IF 4000 FLAG ! STOP THEN   ( STOP IF FLAGGED )
            LOOP
     SREPORT
     FLAG @ 4000 = IF ENINT QUIT THEN
  END ;

: 52 SETUP ;
: SETUP 52 1FF 100 DO I S_TYP LOOP
;

: SSRUN' X_COM %SMRUN % ;
: SRUN  0 SEQU ! 0 ERRORTOT ! 0 M_BYTE ! 0 UCOUNT ! 0 FLAG ! ;
: SRUN1 SRUN SME ;

: 1RESET LOCATE INTER #INT 8 + EQUATES ! ;
: _RES RESTART SETUP 1RESET INTER ;
: 2RESET LOCATE _RES #INT 8 + EQUATES ! ;

%ENDFILE
```

3) MC6809 Monitor Unit Program.

```
* PROGRAM TO STORE THE INFORMATION
* PRESENTED AT THE SECOND PORT
* BY THE 68000
* ASSUMES SECOND PORT IS AT F518
* ASSUMES PRIMARY ACIA AT F500
* COMMANDS AVAILABLE ARE :-
* D DISPLAY DISK STATUS
* Q QUIT LOGGING ACTIVITY
* A ENGAGE THE AUTO PILOT
* N DISENGAGE THE AUTO-PILOT
* C CHANGE DRIVES, USE INSTEAD OF GEORGE
* R REPORTS LAST MESSAGE FROM EACH TERMINAL
```



```

                F500        ACIA1   EQU     $F500
                F518        ACIA2   EQU     $F518

0000 8E         0276        START   LDX     #TMPS
0003 BD         D2A6                JSR     ZOUTST
0006 BD         00AA        NINE    JSR     INIT
0009 8E         F518                LDX     #ACIA2
000C 86         03                  LDA     #$3
000E A7         84                  STA     X
0010 86         15                  LDA     #$15
0012 A7         84                  STA     X
0014 8D         53   0069           BSR     DKINIT    OPEN THE DISK FILE
0016 8E         F500        FIVE    LDX     #ACIA1
0019 A6         84                  LDA     ,X
001B 85         01                  BITA    #1
001D 27         03   0022           BEQ     SEVEN
001F BD         0146                JSR     SERVE
0022 8E         F518        SEVEN   LDX     #ACIA2
0025 A6         84          ONE     LDA     ,X        ANYTHING THERE??
0027 85         01                  BITA    #1
0029 27         EB   0016           BEQ     FIVE      NO SO TRY AGAIN
002B E6         01                  LDB     1,X       CLEAR HANDSHAKE
002D 108E       071D                LDY     #BUFF
0031 C6         40                  LDB     #$40
0033 E7         01                  STB     1,X       TELL 68K WE ARE READY
0035 C6         00                  LDB     #0
0037 CE         0000        FOUR    LDU     #0
003A A6         84          THREE   LDA     ,X
003C 85         01                  BITA    #1        RX'D ANYTHING??
003E 26         22   0062           BNE     TWO       YES SO GO
0040 33         41                  LEAU    1,U       COUNT OF LAPSED TIME
0042 1183       FFFF                CMPU    #$FFFF    HAVE I WAITED LONG ENOUGH??
0046 26         F2   003A           BNE     THREE     NOT YET I HAVEN'T!
0048 C1         00                  CMPB    #0
004A 27         EE   003A           BEQ     THREE
004C B6         0554                LDA     ERROR     DO WE HAVE A PROBLEM
004F 4D                             TSTA
0050 26         05   0057           BNE     TEN
0052 8D         37   008B           BSR     OUTPUT    YES I HAVE, SO GO DUMP DATA
0054 BD         01FE                JSR     SAVE
0057 B6         0556        TEN     LDA     AUTO
005A 4D                             TSTA
005B 27         B9   0016           BEQ     FIVE
005D BD         00E2                JSR     GEORGE
0060 20         B4   0016           BRA     FIVE      AND RETURN TO LIVE ANOTHER D
0062 A6         01          TWO     LDA     1,X       GET THE RELEVANT CHARACTER
0064 A7         A0                  STA     Y+        BUFFER IT
0066 5C                             INCB
0067 20         CE   0037           BRA     FOUR      AND GO TRY FOR ANOTHER

0069 BE         0557        DKINIT  LDX     CFCB
006C 86         01                  LDA     #Q5O4W
006E A7         84                  STA     XFC,X     OPEN FILE FOR WRITE
0070 BD         D7B6                JSR     DFM       WELL, LET THE DFM DO IT FOR
0073 27         0E   0083           BEQ     DKONE     ALL IS WELL SO GO
0075 BD         D2A9        SIX     JSR     ZTYPDE    ALL IS NOT WELL SO TELL THE
```

```
0078 BD   D783              JSR   CDFM     CLOSE ALL FILES
007B 8E   051B              LDX   #TMP13
007E BD   D2A6              JSR   ZOUTST
0081 0E   06                JMP   NINE
0083 4F              DKONE  CLRA
0084 B7   0552              STA   DONE     FLAG FOR GEORGE
0087 B7   0554              STA   ERROR
008A 39                     RTS            HURRAYYYYYYYYYYYY

008B 34   20         OUTPUT PSHS  Y        SAVE LAST BUFFER ADDRESS
008D 108E 071D              LDY   #BUFF    GET THE FIRST
0091 BE   0557              LDX   CFCB
0094 86   02                LDA   #QSWRIT
0096 A7   84                STA   XFC,X    WRITE TO THE FILE
0098 A6   A0         OUTONE LDA   Y+       GET A CHAR
009A BD   D786              JSR   DFM      PUT IT ON DISK
009D 27   03   00A2         BEQ   OUTTWO
009F BD   D2A9              JSR   ZTYPDE
00A2 10AC E4         OUTTWO CMPY  S        HAVE WE FINISHED YET
00A5 2D   F1   0098         BLT   OUTONE   NO SO GO DO ANOTHER
00A7 35   20                PULS  Y        CLEAR UP STACK
00A9 39                     RTS

00AA 8E   02F3       INIT   LDX   #TMP6
00AD BD   D2A6              JSR   ZOUTST   DRIVE NUMBER QUIZ
00B0 BD   D289              JSR   ZINCH    GET IT
00B3 84   7F                ANDA  #$7F     STRIP PARITY
00B5 1F   89                TFR   A,B
00B7 BD   D2BE              JSR   ZOUTCH
00BA BD   D2DC              JSR   ZCRLF
00BD C1   31                CMPB  #'1
00BF 26   06   00C7         BNE   IN1
00C1 86   01                LDA   #1
00C3 C6   02                LDB   #2
00C5 20   0A   00D1         BRA   IN2
00C7 C1   51         IN1    CMPB  #'Q
00C9 1027 00B1 017E         LBEQ  QUIT     ALLOW A QUIT IF NEEDED
00CD 86   02                LDA   #2
00CF C6   01                LDB   #1
00D1 8E   05D3       IN2    LDX   #FCB1
00D4 A7   02                STA   XUN,X
00D6 BF   0557              STX   CFCB     FCB1 IS FIRST DRIVE
00D9 8E   0678              LDX   #FCB2
00DC E7   02                STB   XUN,X    AND FCB2 IS THE SECOND ONE
00DE BF   0559              STX   AFCB     ALTERNATE FCB
00E1 39                     RTS
                     * GEORGE, THE AUTO-PILOT

00E2 BE   0557       GEORGE LDX   CFCB
00E5 86   00                LDA   #QFREE
00E7 A7   84                STA   XFC,X
00E9 BD   D786              JSR   DFM      DETERMINE FREE ON CURRENT DR
00EC 81   01                CMPA  #$01     UPPER BYTE
00EE 2D   01   00F1         BLT   GRG1
00F0 39                     RTS            ENOUGH IS LEFT SO WE HAVE NO
00F1 BD   0137       GRG1   JSR   ALTF     FIND THE FREE ON THE ALTERNA
00F4 C1   10                CMPB  #$10     LOWER BYTE ON CURRENT DRIVE
00F6 23   17   010F         BLS   GRG2
00F8 81   01                CMPA  #$01     UPPER BYTE ON ALTERNATE
00FA 2D   01   00FD         BLT   GRG3
00FC 39                     RTS            SUFFICIENT ON BOTH DRIVES FO
00FD B6   0552       GRG3   LDA   DONE
0100 4D                     TSTA
0101 26   0B   010E         BNE   GRG5     ONLY ISSUE ONE WARNING
0103 8E   0440              LDX   #TMP10
0106 BD   D2A6              JSR   ZOUTST
0109 86   01                LDA   #1
010B B7   0552              STA   DONE     THIS ENSURES WE ONLY DO IT O
010E 39              GRG5   RTS
010F 81   01         GRG2   CMPA  #$01     UPPER BYTE OF ALTERNATE DRIV
0111 2D   18   012B         BLT   GRG4
0113 BE   0557              LDX   CFCB     ALTERNATE DRIVE IS OK
0116 86   03                LDA   #QSWC    SO CLOSE CURRENT DRIVE
0118 A7   84                STA   XFC,X
011A BD   D786              JSR   DFM
011D 10BE 0559              LDY   AFCB
0121 BF   0559              STX   AFCB     SO SWAP DRIVES
0124 10BF 0557              STY   CFCB
0128 9D   69                JSR   DKINIT   AND OPEN A NEW LOGGING FILE
012A 39                     RTS            AND ALL IS OK
012B 8E   04C1       GRG4   LDX   #TMP11
012E BD   D2A6              JSR   ZOUTST   PANIC LADS!!!!!!!!
0131 86   01                LDA   #1
0133 B7   0554              STA   ERROR
0136 39                     RTS

0137 34   04         ALTF   PSHS  B
0139 BE   0559              LDX   AFCB
013C 86   00                LDA   #QFREE
013E A7   84                STA   XFC,X
0140 BD   D786              JSR   DFM      GET FREE ON ALTERNATE DRIVE
0143 35   04                PULS  B
0145 39                     RTS

                     * COMMAND SERVICE ROUTINE

0146 A6   01         SERVE  LDA   1,X      GET THE COMMAND CHARACTER
0148 81   44                CMPA  #'D
014A 26   2E   017A         BNE   S1       DISPLAY COMMAND??
014C BE   0557              LDX   CFCB     YUP
014F 86   00                LDA   #QFREE   FIND OUT THE FREE SPACE
0151 A7   84                STA   XFC,X
0153 BD   D786              JSR   DFM
0156 8E   021D              LDX   #TMP1
0159 A7   84                STA   ,X       HIGH ORDER COUNT
015B E7   01                STB   1,X      LOW ORDER COUNT
015D BD   D2AF              JSR   ZOUTHA   OUTPUT FOUR HEX CHARS
0160 8E   021F              LDX   #TMP2
0163 BD   D2A6              JSR   ZOUTST
0166 BE   0557              LDX   CFCB
0169 A6   02                LDA   XUN,X    CURRENT DRIVE NUMBER
016B 8E   021D              LDX   #TMP1
016E A7   84                STA   ,X
0170 BD   D2AC              JSR   ZOUTHX   OUTPUT TWO HEX CHARS
```

```
0173 8E    023A            LDX   #TMP3
0176 BD    D2A6            JSR   ZOUTST
0179 39                    RTS
017A 81    51         S1   CMPA  #'Q
017C 26    0C   018A       BNE   S2        QUIT COMMAND ??
017E 8E    0253       QUIT LDX   #TMP4
0181 BD    D2A6            JSR   ZOUTST    YES SO ACKNOWLEDGE
0184 BD    D7B3            JSR   CDFM      CLOSE ALL FILES
0187 7E    D283            JMP   ZWARMS    RETURN TO FACE THE MUSIC!!
018A 81    41         S2   CMPA  #'A
018C 26    0A   0198       BNE   S3
018E B7    0556            STA   AUTO      ENGAGE GEORGE, THE AUTOPILOT
0191 8E    0318            LDX   #TMP7
0194 BD    D2A6            JSR   ZOUTST
0197 39                    RTS
0198 81    4E         S3   CMPA  #'N
019A 26    0B   01A7       BNE   S4
019C 4F                    CLRA
019D B7    0556            STA   AUTO      DISENGAGE GEORGE
01A0 8E    0343            LDX   #TMP8
01A3 BD    D2A6            JSR   ZOUTST
01A6 39                    RTS
01A7 81    48         S4   CMPA  #'H
01A9 26    07   01B2       BNE   S5
01AB 8E    0366            LDX   #TMP9
01AE BD    D2A6            JSR   ZOUTST
01B1 39                    RTS
01B2 81    43         S5   CMPA  #'C
01B4 26    1E   01D4       BNE   S6
01B6 8E    04D7            LDX   #TMP12
01B9 BD    D2A6            JSR   ZOUTST
01BC BE    0557            LDX   CFCB
01BF 86    03              LDA   #QSWC
01C1 A7    84              STA   XFC,X     CLOSE CURRENT ONE
01C3 BD    D7B6            JSR   DFM
01C6 10BE  0559            LDY   AFCB
01CA 10BF  0557            STY   CFCB
01CE BF    0559            STX   AFCB      SWAP FCB'S
01D1 9D    69              JSR   DKINIT    OPEN OTHER DRIVE
01D3 39                    RTS
01D4 81    52         S6   CMPA  #'R
01D6 26    25   01FD       BNE   S7
01D8 108E  055B            LDY   #SBUF     SAVE BUFF START
01DC 5F                    CLRB
01DD A6    A0         S6B  LDA   Y+
01DF 5C                    INCB
01E0 BD    D2AC            JSR   ZOUTHX
01E3 86    20              LDA   #$20
01E5 BD    D2BE            JSR   ZOUTCH
01E8 C1    18              CMPB  #$18
01EA 26    0B   01F7       BNE   S6A
01EC 86    0D              LDA   #$D
01EE BD    D2BE            JSR   ZOUTCH
01F1 86    0A              LDA   #$A
01F3 BD    D2BE            JSR   ZOUTCH
01F6 5F                    CLRB
01F7 108C  05D3       S6A  CMPY  #SEND
```



```
01FB 26    E0   01DD          BNE   S6B
01FD 39               S7      RTS

01FE 31    A8 E8      SAVE    LEAY  -$18,Y
0201 EC    22                 LDD   2,Y
0203 1083  AAAA               CMPD  #$AAAA   ERROR REPORT???
0207 26    01   020A          BNE   SAV1
0209 39                       RTS            YES SO RETURN
020A A6    21         SAV1    LDA   1,Y      GET REP'NO
020C C6    18                 LDB   #$18     RECORD LENGTH
020E 3D                       MUL            OFFSET INTO BUFFER
020F 8E    055B               LDX   #SBUF    BUFFER START
0212 3A                       ABX            POINTER TO RECORD BUFF
0213 C6    18                 LDB   #$18
0215 A6    A0         SAV2    LDA   Y+       GET A CHAR FROM INCOMING REC
0217 A7    80                 STA   X+       STORE IN SBUF
0219 5A                       DECB
021A 2E    F9   0215          BGT   SAV2
021C 39                       RTS

021D       0000       TMP1    FDB   0        SPACE FOR D COMMAND
021F       20         TMP2    FCC   / FREE SECTORS ON DRIVE :- /
0239       00                 FCB   0
023A       20         TMP3    FCC   / CURRENT LOGGING DRIVE/
0250       0D                 FCB   $0D,$0A,00
0253       43         TMP4    FCC   /CLOSE ALL FILES AND QUIT LOGGING/
0273       0D                 FCB   $0D,$0A,00
0276       2A         TMP5    FCC   /*****************************/
0293       0D                 FCB   $0D,$0A
0295       2A                 FCC   /* DATA LOGGING PROGRAM V1.1 */
02B2       0D                 FCB   $0D,$0A
02B4       2A                 FCC   /*       DECEMBER 1981       */
02D1       0D                 FCB   $0D,$0A
02D3       2A                 FCC   /*****************************/
02F0       0D                 FCB   $0D,$0A,0
02F3       45         TMP6    FCC   /ENTER FIRST LOGGING DRIVE NUMBER :-
0317       00                 FCB   0
0318       45         TMP7    FCC   /ENGAGE GEORGE, THE FAITHFUL AUTO-PIL
0340       0D                 FCB   $0D,$0A,0
0343       44         TMP8    FCC   /DISENGAGE GEORGE, THE POOR BEAST/
0363       0D                 FCB   $0D,$0A,0
0366       48         TMP9    FCC   /HELP/
036A       0D                 FCB   $0D,$0A
036C       41                 FCC   /A :- ENGAGE THE AUTO-PILOT/
0386       0D                 FCB   $0D,$0A
0388       43                 FCC   /C :- CHANGE DRIVES, USE INSTEAD OF O
03B1       0D                 FCB   $0D,$0A
03B3       44                 FCC   /D :- DISPLAY SYSTEM STATUS/
03CD       0D                 FCB   $0D,$0A
03CF       4E                 FCC   /N :- DISENGAGE THE AUTO-PILOT/
03EC       0D                 FCB   $0D,$0A
03EE       51                 FCC   /Q :- CLOSE FILES AND QUIT LOGGING/
040F       0D                 FCB   $0D,$0A
0411       52                 FCC   /R :- DISPLAYS LAST REPORT FROM EACH
043D       0D                 FCB   $0D,$0A,0
0440       2A         TMP10   FCC   /**********WARNING***********/
```

```
045E   0D                  FCB   $0D,$0A
0460   20                  FCC   / CURRENT AND SECONDARY DRIVES /
047E   0D                  FCB   $0D,$0A
0480   20                  FCC   /       BOTH NEARLY FULL         /
049E   0D                  FCB   $0D,$0A
04A0   2A                  FCC   /******************************/
04BE   0D                  FCB   $0D,$0A,0
04C1   2A          TMP11   FCC   /*****ATTENTION*****/
04D4   0D                  FCB   $0D,$0A,$07
04D7   2A          TMP12   FCC   /****CHANGE DRIVES****/
04EC   0D                  FCB   $0D,$0A,00
04EF   2A                  FCC   /**SECONDARY FULL***/
0502   0D                  FCB   $0D,$0A,$07
0505   43                  FCC   /CHANGE IMMEDIATELY/
0517   0D                  FCB   $0D,$0A,$07,0
051B   45          TMP13   FCC   /ERROR WITH LOGGING FILE/
0532   0D                  FCB   $0D,$0A
0534   52                  FCC   /REPLACE A DISK AND RE-ENTER/
054F   0D                  FCB   $0D,$0A,00
0552   0002        DONE    RMB   2          A FLAG FOR GEORGE
0554   0002        ERROR   RMB   2          FLAG FOR BOTH DISKS FULL
0556   00          AUTO    FCB   0
0557   0002        CFCB    RMB   2
0559   0002        AFCB    RMB   2
055B   0078        SBUF    RMB   120
       05D3        SEND    EQU   *
05D3   0003        FCB1    BSZ   3
05D6   36                  FCC   /68KDAT/
05DC   009C                BSZ   156
0678   0003        FCB2    BSZ   3
067B   36                  FCC   /68KDAT/
0681   009C                BSZ   156
       071D        BUFF    EQU   *
                           END

AL ERRORS 00000
AL WARNINGS 00000--00000
```

```
BLOCK MESSAGE SOAK TEST )
RECEIVE ROUTINE )
ASWE TRIALS )
DECEMBER 1981 )

00 CONSTANT BK.START                    ( START OF TEST BLOCK )
) CONSTANT BK.LENGTH
.START BK.LENGTH + CONSTANT BK.END                ( END OF TEST BLOCK )
TEST_BYTE VARIABLE ;                    ( START OF MESSAGE TEST_BYTE )
ERRORTOT VARIABLE ;                     ( ERROR TOTAL )
BK.SRC VARIABLE ;
STARTING VARIABLE ;
COUNT VARIABLE ;                        ( COUNT OF BK.RX TRIES )
SEQU VARIABLE ;                         ( OUT OF SEQUENCE COUNT )
COUNT2 VARIABLE ;
EACH RX HAS A UNIQUE REP'NO )
AND THE TX USES IT TO REFERENCE ITS ARRAY OF POINTERS )
REP'NO VARIABLE ;


: REP ARRAY ;

CL_REP REP DUP 15 + SWAP DO 0 I !B LOOP DROP ;
MSTRIP CL_REP AAAA REP [ 1 ] !   ( FLAG ERROR RECORD )
        REP'NO @ REP [ 0 ] !
        A 2 DO REP [ I ] ! LOOP
        REP BK.SRC @ 3F MASK 1 SEND ;

ANALYSE CODE
      9482    SUB.L D2,D2
      2602    MOVE.L D2,D3
      2802    MOVE.L D2,D4
      265E    MOVE.L (A6)+,A3
      245E    MOVE.L (A6)+,A2
      221E    MOVE.L (A6)+,D1
      3601 A  MOVE.W D1,D3
      381B    MOVE.W (A3)+,D4
      B943    EOR.W D4,D3
      670A    BEQ.S B
      5242    ADDQ #1,D2
      B47C    CMP #09,D2
      0009
      6E02    BGE.S B
      2D03    MOVE.L D3,-(A6)
      5241 B  ADDQ #1,D1
      B5CB    CMP.L A3,A2
      6EE8    BGE.S A
      2D02    MOVE.L D2,-(A6)
      4E75    RTS
      0000


SETS UP ERRORTOT OR SEQU DEPENDING ON THE NUMBER OF ERRORS )

BK.ERRS DUP 20 > IF SEQU '1 BK.START @ 1 + TEST_BYTE !
                        DROP MSTRIP
                      ELSE ERRORTOT @ + ERRORTOT ! MSTRIP
                      THEN ;

BK.REC BK.LENGTH BK.START BK.SRC @
          BLK_REC                  ( SET UP TO RECEIVE THE BLOCK )
```

```
: BKR.STAT IN_BLK_STAT @B 80 MASK
;

( CHECK THAT THE RECEIVED BLOCK AGREES WITH THE ONE WE ARE EXPECTING )

: BK.ANALYSE TEST_BYTE @
          BK.END BK.START
                          ANALYSE
                          TEST_BYTE '1
;
( USE THIS COMMAND TO SET UP A COMMAND LINE )
( WHICH WILL BE SENT ELSEWHERE )
: X_COM IMMEDIATE # STRING 0D DP @ 1 - !B ;


: OK? 02 2000 BK.SRC @ 3F MASK 2 SEND ; ( ACKNOWLEDGE BLOCK RX )


( RECEIVE A BLOCK IF POSSIBLE )
( EVERY UNSUCCESSFUL ATTEMPT IS COUNTED AND ANOTHER )
( HANDSHAKE IS SENT FOR EVERY 300 TIMES IT FAILS )

: BK.RX BKR.STAT 00 > IF
                           BK.ANALYSE
                       DUP 0 > IF BK.ERRS ELSE DROP
                               THEN
                           BK.REC OK? COUNT '1
                       THEN  ( AVOID A LOCK OUT )
;

( SET UP A FEW VARIABLES )
( THIS COMMAND IS SENT BY THE TRANSMITTER )
( AND INITIALISES BK.SRC ACCORDINGLY )

: SET.TX BK.SRC ! 0 STARTING  ! 0 ERRORTOT ! 0 TEST_BYTE !
0 SEQU ! 0 COUNT2 ! ;



( GET THE FIRST THREE TIME WORDS INTO A THREE CONSECUTIVE LOCATIONS )

: GTIME1 0 PR_TAB 66 + ! 4000 1 DO PR_TAB 66 + @ 8000 = IF STOP THEN LOOP
        2 0 DO PR_TAB 68 + I2 @ OVER I2 ! LOOP DROP ;


( SEND A REPORT TYPE ONE 'STOREIT' TO TX )
( REPORT THE TIME, ERRORTOT, RE_COUNT AND REP'NO )

: REPORT REP 2 + GTIME1 DROP        ( GET SOME TIME INTO THE SCENE )
        ERRORTOT @ REP [ 4 ] !   ( THE TOTAL NUMBER OF ERRORS )
        RE_COUNT @ REP [ 5 ] !   ( RECEIVE ERROR TOT )
        REP'NO   @ REP [ 0 ] !   ( THE TERMINALS REPORT NUMBER )
        SEQU     @ REP [ 6 ] !    ( MESSAGE OUT OF SEQUENCE )
     TEST_BYTE   @ REP [ 7 ] !   ( NUMBER OF MESSAGES, SORT OF )
     REP BK.SRC @ 3F MASK 1 SEND ;

( TEST LOOP )
: RUN 0 COUNT ! BK.REC OK?     ( FIRST HANDSHAKE TRANSMISSION )
     BEGIN
          8000 0 DO BK.RX LOOP
                 COUNT @ 0 = IF OK? COUNT2 '1 ELSE 0 COUNT2 !
                             THEN 0 COUNT !
          REPORT
```

```
        IN_BLK_STAT 1 + !B                    ( REMAINDER )
        IN_BLK_TOTAL !B                        ( SUB-BLOCK TOTAL )
        0 IN_BLK_STAT !B                      ( GO GO GO )
        THEN
;

( MESSAGE RECEIVE ROUTINES )

: INV -1 SWAP - ;
: S_TYP              ( SELECT A CERTAIN MESAGE TYPE )
      DUP F MASK 1 SWAP LEFT SWAP 10 / FFFF MASK 2 *
      MES_TAB + DUP @ INV ROT ROT MASK OVER @ + SWAP ! ;
            ( SETS THE DESIRED BIT IN THE MTT ARRAY )

: CL_TYP             ( DESELECT A CERTAIN MESSAGE TYPE )
       DUP F MASK 1 SWAP LEFT INV SWAP 10 / FFFF MASK 2 *
       MES_TAB + DUP @ ROT ROT MASK SWAP ! ;
            ( CLEARS THE DESIRED BIT IN THE MTT ARRAY )

: IN_BUF_NO VARIABLE ;

: REL_BUF IN_BUF_NO DUP @ 1 + F MASK SWAP ! THEN ;
            ( MOVE TO NEXT INPUT BUFFER, WRAPAROUND AT 'F' )

: RECEIVE                  ( CHECKS TO SEE IF A MESSAGE IS AVAILABLE )
         IN_TABLE ( IN_BUF_NO @ IN_BUF_LEN ) @B DUP
         0 > IF 3 - 2 *                  ( SIZE IN BYTES )
              IN_TABLE ( IN_BUF_NO @ IN_DAT_BUF )   ( ADDRESS )
              IN_TABLE ( IN_BUF_NO @ IN_SOURCE ) @B  ( SOURCE )
              IN_TABLE ( IN_BUF_NO @ IN_TYPE ) @     ( TYPE )
           0 IN_TABLE ( IN_BUF_NO @ IN_BUF_LEN ) !B ( CLEAR IT )
              REL_BUF          ( READY FOR NEXT TIME )
              0                ( EVERYTHING WAS OK )
              ELSE DROP -1     ( NOTHING WENT RIGHT!! )
              THEN
;

: TAB_SET RESET CCW ! B00 A00 CL2 6800 5700 CL2
 80 IN_INT !B 10 IN_NO !B
 2B80 IN_TAB !
 80 O_INT !B  10 O_NO !B
 2D80 O_TAB !
 0 OUT_BUF_NO !
 0 IN_BUF_NO !
 8000 IN_BLK_STAT !  ;

 SETUP THE TERMINAL UNIT )
 S2 SETUP ;      ( THE SETUP IN THE DICTIONARY MUST BE USED TOO )
 SETUP S2 TAB_SET FFFF MES_TAB ! GO CCW ! ;

 THIS ROUTINE ALLOWS THE INSERTION OF HAND ASSEMBLED )
 MACHINE CODE TO SPEED THINGS UP A BIT )

 CODE IMMEDIATE 100 1 DO FRELO @ 0 = IF BUFFER ELSE LOAD THEN
                    WORD NUMBER DUP 0 = IF
             STOP THEN DP! LOOP ;
 PTM REGISTERS )

FF61 CONSTANT PTM      ( BASE )
TM      CONSTANT WR3
TM      CONSTANT WR1
TM 2 + CONSTANT WR2
TM 4 + CONSTANT T#1
TM 6 + CONSTANT WT#1
```

```
( USE PTM TO PROVIDE INTERRUPTS FOR 2901 SERVICE ROUTINE )

( SET IT TO INTERRUPT APPROXIMATELY ONCE A SECOND )
: SETPTM 0 WR2 !B 1 WR3 !B 1 WR2 !B    ( DIVIDE BY 8 )
       42 WR1 !B                       ( CONTINUOUS OPERATION )
       7F T#1 !B FF WT#1 !B            ( AS SLOW AS POSSIBLE )
;

( THIS ROUTINE WILL PERFORM A NORMAL SIXTH )
( WORD FIND EXECUTE ON THE CONTENTS OF )
( A TYPE ZERO MESSAGE WHICH HAS BEEN RECEIVED )
( FROM THE OUTSIDE WORLD BY THE 2901 SUBSYSTEM )

: DOIT WE !                            ( START ADDRESS )
     0 LAST !
     30 0 DO WORD FIND EXECUTE         ( NEW INTERPRET LOOP )
             0 LAST @ = IF             ( OK IF = 0 )
                        ELSE STOP      ( OTHERWISE GIVE UP )
                        THEN LOOP
;

: [ IMMEDIATE ;       ( I.E THROW IT AWAY )
: ] 2 * + SWAP DROP ;
: I2 I 2 * + ;

: TX.OK VARIABLE ;

: PROCESS
            DUP 2 = IF TX.OK ^1 POP POP POP POP
        ELSE DUP 0 = IF           ( TYPE ZERO?? )
                   DROP DROP DOIT POP    ( DO AS THE MAN SAYS )
      ELSE POP POP POP POP
           THEN THEN
;

( IRQ5 HANDLES THE PTM INTERRUPT )
( IT FIRST CHECKS FOR ANY RECEIVED MESSAGES )
( IF ANY HAVE BEEN RECEIVED, IT TRIES FOR EITHER )
( A TYPE ZERO, OR A TYPE ONE, OR IGNORES IT )
( FINALLY IT CLEARS THE PTM INTERRUPT )

: IRQ5 FRAME WE @ WB @ LAST @ FRAME        ( SAVE FOR POSTERITY )
         RECEIVE 0 = IF
         PROCESS
         THEN
         UNFRAME LAST ! WB ! WE !
         WR2 @B T#1 @B  ( CLEAR PTM INTERRUPT )
         UNFRAME RTE
;

: S2 SETUP ;
 : SETUP S2 LOCATE IRQ5 #INT 8 + 74 !L
SETPTM
ENINT
;
```

```
NEW SLAVE UNIT SOFTWARE )
DESTINED FOR THE SHIP TRIALS )
JANUARY 1982 )
REVISION 3.0 JANUARY 1982 )
INCLUDES BLOCK AND SHORT MESSAGE TESTS )
INCLUDES REVISION TO ALLOW USE OF MTB EXTENSION BIT )

STOREIT AND DOIT INCLUDED IN THIS ONE )

TERMINAL UNIT PRIMARY TABLE )


00 CONSTANT PR_TAB
R_TAB CONSTANT IN_INT
N_INT 1 + CONSTANT IN_NO
N_NO  2 + CONSTANT IN_POS
N_POS 1 + CONSTANT IN_TAB
N_TAB 2 + CONSTANT O_INT
_INT 1 + CONSTANT O_NO
_NO  2 + CONSTANT O_POS
_POS 1 + CONSTANT O_TAB
_TAB 2 + CONSTANT MES_TAB
ES_TAB 40 + CONSTANT HW_NO
J_NO   2 + CONSTANT RE_COUNT
E_COUNT 2 + CONSTANT DAT_STARV
AT_STARV 2 + CONSTANT RETR_COUNT
ETR_COUNT 2 + CONSTANT BUF_OVER
UF_OVER   2 + CONSTANT IN_BLK_STAT
I_BLK_STAT    2 + CONSTANT IN_BLK_SOURCE
I_BLK_SOURCE  2 + CONSTANT IN_BLK_TOTAL
I_BLK_TOTAL 1 + CONSTANT IN_BLK_TOT_RECVD
I_BLK_TOT_RECVD 1 + CONSTANT IN_BLK_ADDRESS
I_BLK_ADDRESS 2 + CONSTANT O_BLK_STAT
BLK_STAT      3 + CONSTANT O_BLK_DESTIN
BLK_DESTIN      1 + CONSTANT O_BLK_TOT
BLK_TOT 1 + CONSTANT O_BLK_TOT_TXD
BLK_TOT_TXD 1 + CONSTANT O_BLK_START

END OF PRIMARY TABLE )

IN TABLE )
MASK FOR A SINGLE BUFFER AREA )

  ( RELATIVE TO START OF BUFFER )
 CONSTANT IN_BUF_LEN
_BUF_LEN 1 + CONSTANT IN_DEST
_DEST  1 + CONSTANT IN_SOURCE
_SOURCE 1 + CONSTANT IN_TYPE
_TYPE 2 + CONSTANT IN_DAT_BUF



OUT TABLE )
MASK FOR A SINGLE BUFFER AREA )

  ( RELATIVE TO START OF BUFFER )
CONSTANT O_BUF_LEN
BUF_LEN 1 + CONSTANT O_DEST
DEST     2 + CONSTANT O_TYPE
TYPE     2 + CONSTANT O_DAT_BUF


TABLE IMMEDIATE 2D3C DP! 0 DP! DP! 2D3C DP! 0 DP! DP! ;
( ,
```

```
: ) ROT * + + ;

( CHANNEL CONTROL WORD )

3C000 CONSTANT CCW

( CONTROL WORDS )

2 CONSTANT GO               ( START TERMINAL UNIT )
3 CONSTANT CSTOP            ( STOP TERMINAL UNIT )
4 CONSTANT OFFINT           ( SWITCH OFF INTERRUPTS )
5 CONSTANT ONINT            ( SWITCH ON AND CLEAR INTERRUPTS )
7 CONSTANT RESET            ( RESET AND HALT TERMINAL UNIT )


( BITS TO CONTROL TERMINAL UNIT )


40 5700 : IN_TABLE TABLE ;
40 5800 : OUT_TABLE TABLE ;
: OUT_BUF_NO VARIABLE ; ( 68K'S RECORD OF NEXT FREE BUFFER )
: GET_BUF OUT_BUF_NO DUP @ 1 + F MASK SWAP ! ;
: CL2 DO 0 I !B LOOP ;

( MESSAGE SEND ROUTINE )
: SEND OUT_TABLE ( OUT_BUF_NO @ O_BUF_LEN ) @B
      0 = IF        ( BUFFER IS FREE SO CARRY ON )
          OUT_TABLE ( OUT_BUF_NO @ 5 ) DUP 5 - CL2
          OUT_TABLE ( OUT_BUF_NO @ O_TYPE ) ! ( MESSAGE TYPE )
          OUT_TABLE ( OUT_BUF_NO @ O_DEST ) !B ( DESTINATION )
          OVER 0 DO DUP I + @B
                   OUT_TABLE ( OUT_BUF_NO @ O_DAT_BUF I + ) !B LOOP DROP
          1 + 2 / 3 + 3F MASK OUT_TABLE ( OUT_BUF_NO @ O_BUF_LEN ) !B
          GET_BUF
          ELSE POP POP POP POP
          THEN
;

( BLOCK RECEIVE AND TRANSMIT ROUTINES )

: TOT&REM                 ( GET SUB-BLOCK TOTAL AND REMAINDER )
        1 + 2 / FFFF MASK          ( WORD COUNT )
        20 / DUP SWAP   ( SUB-BLOCK TOTAL )
        3F 10 LEFT MASK
        80 / 200 /          ( REMAINDER )
;

: BLK_SEND O_BLK_STAT @ 8000 MASK 0 = IF
          O_BLK_START O_BLK_STAT CL2       ( CLEAR UP TABLE )
          O_BLK_DESTIN !B                       ( DESTINATION )
          2 / O_BLK_START !                  ( START ADDRESS )
          TOT&REM
          O_BLK_STAT !
          O_BLK_TOT !B
          80 O_BLK_STAT !B                 ( GO GO GO )
          ELSE DROP DROP DROP ( GET RID OF PARAMS )
          THEN
;

: BLK_REC IN_BLK_STAT @ 8000 MASK 0 = IF
        DROP DROP DROP       ( GET RID OF UNWANTED PARAMS )
        ELSE 0 IN_BLK_STAT 2 + !L        ( CLEAR UP )
        IN_BLK_SOURCE !
        2 / IN_BLK_ADDRESS !
        TOT&REM
```

```
( A TYPE ZERO, OR A TYPE ONE, OR IGNORES IT )
( FINALLY IT CLEARS THE PTM INTERRUPT )

: IRQ5 FRAME WE @ WB @ LAST @ FRAME          ( SAVE FOR POSTERITY )
            RECEIVE 0 = IF
            PROCESS
            THEN
            UNFRAME LAST ! WB ! WE !
            WR2 @B T#1 @B  ( CLEAR PTM INTERRUPT )
            UNFRAME RTE
;

: 52 SETUP ;
 : SETUP 52 LOCATE IRQ5 #INT 8 + 74 !L
SETPTM
ENINT
;


( PERIODICAL BLOCK TRANSMIT ROUTINE )
( BLOCK MESSAGE SOAK TEST FOR ASWE TRIALS )
( DECEMBER 1981 )
( REVISION 2.0 28/12/81 )

7400 CONSTANT BK.START                 ( START OF TEST BLOCK )
800 CONSTANT BK.LENGTH
BK.START BK.LENGTH + CONSTANT BK.END       ( END OF TEST BLOCK )

: TEST_BYTE VARIABLE ;              ( USED FOR CYCLIC MESSAGE GENERATION )
: RX_COUNT VARIABLE ;     ( TOTAL NUMBER OF TERMINALS )


( GENERATE A NEW BLOCK OF DATA )
( FROM THE TEST BYTE )
( SP => START,END,TEST_BYTE )

: BK.GEN CODE
        265E      MOVE.L (A6)+,A3
        245E      MOVE.L (A6)+,A2
        2216      MOVE.L (A6),D1
        36C1 ONE  MOVE.W D1,(A3)+
        5241      ADDQ.W #1,D1
        B5CB      CMPA.L A3,A2
        6CF8      BGE ONE
        4E75      RTS
        0000
;

: BKS.STAT O_BLK_STAT @B           ( TX BLOCK STATUS BYTE )
        80 MASK 0 = IF 0 ELSE -1
        THEN
                         ( RETURNS 0 IF FINISHED, -1 IF NOT )
;

( BK.TX CHECKS THAT A HANDSHAKE HAS BEEN RECEIVED )
( IF SO, AND THE LAST BLOCK HAS BEEN TRANSMITTED )
( IT WILL GENERATE A NEW BLOCK, TRANSMIT IT )
( AND UPDATE THE TEST_BYTE )

: BK.TX
      TX.OK @ RX_COUNT @ >= IF
      BKS.STAT 0 = IF     ( WAIT FOR THE OK FROM ABOVE )
                      TEST_BYTE @ BK.END BK.START BK.GEN
                      DROP BK.LENGTH BK.START O_BLK_SEND
                      TEST_BYTE '1 0 TX.OK !
                      THEN
                      THEN
;


( X_COM MAKES A CR TERMINATED STRING )

: X_COM IMMEDIATE # STRING 0D DP @ 1 - !B ;

( THESE TWO ROUTINES FORM THE '800A SET.TX' COMMAND SENT )
( TO ALL RECEIVERS )

: SCOM X_COM &8000 SET.TX & ;
: SMY SCOM HW_NO @ DISSECT DROP OVER 3 + !B
        0 0 SEND 0 TEST_BYTE ! 0 TX.OK ! ;

( THESE ARE USED TO START THE TEST OFF )
: SRUN' X_COM %RUN % ;
: SRUN SRUN' 0 0 SEND ;

( REC.OK IS EXECUTED BY THE TRANSMITTER AS A HANDSHAKE )
:  REC.OK TX.OK '1 ;

( RESET ALL REP POINTERS TO ZERO )

: RESET'REP 4 0 DO 0 REP'POINT [ I ] ! LOOP ;

( DISPENSE WITH REPORTS WHICH ARE CLOGGING UP THE BUFFER )
( PRINTS THEM UP AND RESETS THE POINTER )

: WAITING VARIABLE ;       ( COUNT OF DUMP TRIES )
: BUF'SEND A2 DUP 2 * REP'POINT SWAP DROP + DUP @ 1 - 0 DO
                                          OVER 300 * I 16 * +
                                          REP'BASE + DUP
                  15 + SWAP DO I @B TO LOOP
                                          LOOP
          0 SWAP ! DROP 0 WAITING !
ACI1 @L PORT !L
;


; ACHECK ACI2 @L @B 1 MASK 0 > IF ACI2 @L 2 + @B DROP 0
                               ELSE -1
                               THEN
;


: OVERRUN STRING %SLIGHT PROBLEM CHAPS, OVERRUN!!!! % SAY
0 WAITING ! RESET'REP CRLF
;
: DISPENSE WAITING @ 0 = IF 40 ACI2 @L 2 + !B
          THEN ACHECK 0 = IF BUF'SEND QUIT
          ELSE WAITING '1 DROP THEN
          WAITING @ 10 > IF OVERRUN QUIT THEN
;

: BUF? CHECK              ( SEE IF ANYTHING NEEDS TO BE DONE )
     DUP -1 = IF DROP
              ELSE DISPENSE
              THEN ;

: NSET STRING %NUMBER OF RECEIVERS IN TEST?? % SAY
     CRLF BUFFER WORD NUMBER RX_COUNT ! CRLF ;
```

```
        2 / IN_BLK_ADDRESS !
        TOT&REM
        IN_BLK_STAT 1 + !B                      ( REMAINDER )
        IN_BLK_TOTAL !B                          ( SUB-BLOCK TOTAL )
        0 IN_BLK_STAT !B                        ( GO GO GO )
        THEN
;

( MESSAGE RECEIVE ROUTINES )

: INV -1 SWAP - ;
: S_TYP             ( SELECT A CERTAIN MESAGE TYPE )
     DUP F MASK 1 SWAP LEFT SWAP 10 / FFFF MASK 2 *
     MES_TAB + DUP @ INV ROT ROT MASK OVER @ + SWAP ! ;
          ( SETS THE DESIRED BIT IN THE MTT ARRAY )

: CL_TYP              ( DESELECT A CERTAIN MESSAGE TYPE )
      DUP F MASK 1 SWAP LEFT INV SWAP 10 / FFFF MASK 2 *
      MES_TAB + DUP @ ROT ROT MASK SWAP ! ;
          ( CLEARS THE DESIRED BIT IN THE MTT ARRAY )

: IN_BUF_NO VARIABLE ;

: REL_BUF IN_BUF_NO DUP @ 1 + F MASK SWAP ! ;
          ( MOVE TO NEXT INPUT BUFFER, WRAPAROUND AT 'F' )

: RECEIVE             ( CHECKS TO SEE IF A MESSAGE IS AVAILABLE )
        IN_TABLE [ IN_BUF_NO @ IN_BUF_LEN ] @B DUP
        0 > IF 3 - 2 *                     ( SIZE IN BYTES )
             IN_TABLE [ IN_BUF_NO @ IN_DAT_BUF ]    ( ADDRESS )
             IN_TABLE [ IN_BUF_NO @ IN_SOURCE ] @B  ( SOURCE )
             IN_TABLE [ IN_BUF_NO @ IN_TYPE ] @     ( TYPE )
           0 IN_TABLE [ IN_BUF_NO @ IN_BUF_LEN ] !B ( CLEAR IT )
             REL_BUF          ( READY FOR NEXT TIME )
             0                ( EVERYTHING WAS OK )
             ELSE DROP -1     ( NOTHING WENT RIGHT!! )
             THEN
;

: TAB_SET RESET CCW ! B00 A00 CL2 6800 5900 CL2
 80 IN_INT !B 10 IN_NO !B
 2C80 IN_TAB !
 80 O_INT !B  10 O_NO !B
 2E80 O_TAB !
 0 OUT_BUF_NO !
 0 IN_BUF_NO !
 8000 IN_BLK_STAT !  ;

: SETUP THE TERMINAL UNIT )
 52 SETUP ;     ( THE SETUP IN THE DICTIONARY MUST BE USED TOO )
 SETUP 52 TAB_SET FFFF MES_TAB ! GO CCW ! ;

 THIS ROUTINE ALLOWS THE INSERTION OF HAND ASSEMBLED )
 MACHINE CODE TO SPEED THINGS UP A BIT )

 CODE IMMEDIATE 100 1 DO FRELO @ 0 = IF BUFFER ELSE LOAD THEN
                     WORD NUMBER DUP 0 = IF
            STOP THEN DP! LOOP ;
 PTM REGISTERS )

FF61 CONSTANT PTM     ( BASE )
TM     CONSTANT WR3
TM     CONSTANT WR1
TM 2 + CONSTANT WR2
```

```
PTM 4 + CONSTANT T#1
PTM 6 + CONSTANT WT#1

( USE PTM TO PROVIDE INTERRUPTS FOR 2901 SERVICE ROUTINE )

( SET IT TO INTERRUPT APPROXIMATELY ONCE A SECOND )
: SETPTM 0 WR2 !B 1 WR3 !B 1 WR2 !B    ( DIVIDE BY 8 )
         42 WR1 !B                     ( CONTINUOUS OPERATION )
         7F T#1 !B FF WT#1 !B          ( AS SLOW AS POSSIBLE )
;

( THIS ROUTINE WILL PERFORM A NORMAL SIXTH )
( WORD FIND EXECUTE ON THE CONTENTS OF )
( A TYPE ZERO MESSAGE WHICH HAS BEEN RECEIVED )
( FROM THE OUTSIDE WORLD BY THE 2901 SUBSYSTEM )

: DOIT WE !                             ( START ADDRESS )
      0 LAST !
      30 0 DO WORD FIND EXECUTE         ( NEW INTERPRET LOOP )
              0 LAST @ = IF             ( OK IF = 0 )
                   ELSE STOP            ( OTHERWISE GIVE UP )
                   THEN LOOP
;

( BITS TO STORE DATA RECEIVED FROM THE OTHER TERMINALS )
( STOREIT WILL STORE TEN BYTES OF DATA FROM A TYPE ONE )
( MESSAGE IN THE REP BUFFER, AT A LOCATION AS POINTED )
( TO BY REP'POINT, WHICH IS ALSO UPDATED BY THIS ROUTINE )

: [ IMMEDIATE ;        ( I.E THROW IT AWAY )
: ] 2 * + SWAP DROP ;
10 : REP'POINT ARRAY ;
: I2 I 2 * + ;
6400 CONSTANT REP'BASE


: STOREIT DUP @ DUP            ( GET THE REP'NO FROM MESSAGE )
        2 * REP'POINT SWAP DROP + DUP @ OVER OVER 1 + SWAP !
                         SWAP DROP     ( UP RELEVANT STORE POINTER )
                         16 * SWAP 300 * + REP'BASE +
        15 0 DO OVER I2 @ OVER I2 ! LOOP    ( COPY OUT MESSAGE )
            DROP DROP DROP
;

( CHECK THAT NONE OF THE REPORT BUFFERS IS TOO FULL )
( RETURNS EITHER THE REP'NO OF A FULL ONE OR -1 )

: CHECK 4 0 DO REP'POINT [ I ] @ 12 > IF I ABORT THEN LOOP -1 ;

: TX.OK VARIABLE ;


: PROCESS
              DUP 2 = IF TX.OK "1 POP POP POP POP
         ELSE DUP 0 = IF           ( TYPE ZERO77 )
                     DROP DROP DOIT POP    ( DO AS THE MAN SAYS )
        ELSE DUP 1 = IF DROP DROP STOREIT
        ELSE
             THEN THEN THEN
;


( IRQ5 HANDLES THE PTM INTERRUPT )
( IT FIRST CHECKS FOR ANY RECEIVED MESSAGES )
( IF ANY HAVE BEEN RECEIVED, IT TRIES FOR EITHER )
```

```
: C2 CSTOP CCW ! FEL'CL \STOPPING SAY DEL 20 PT' ! \OK SAY DEL ;
: C3 RESET CCW ! 0 FAILURE ! FEL'CL \RESETTING SAY DEL 20 PT' !
    \OK SAY DEL 1 CTUSW ! ;
: CA CTUCW @ FFFE MASK DUP 1 + CTUCW !
    FEL'CL \PASSIVE SAY DEL CRLF \OK SAY DEL CTUCW ! ;
: CC FEL'CL \SELECT SAY BUFFER WORD NUMBER 1 LEFT
    CTUCW @ FFF9 MASK + CTUCW ! CRLF \OK SAY DEL ;

( CHANGE SYSTEM OPERATION COMMANDS )
: BD FEL'CL \NEWSYNCH SAY 0 CTIME !L 0 CTIME 4 + ! DEL 20 PT' ! \OK SAY ;

: BA 0 WHERE 0 7F 0 DO #RES LOOP ;
: BB PTPT @ 1 LEFT DUP PTA = IF PTB ELSE PTA THEN
    OVER @ OVER ! 7F 1 DO OVER I 2 * + @ DUP
    0 = IF DROP FEL'CL \NEWONE SAY BUFFER WORD NUMBER
        8000 + OVER I 2 * + ! DUP 0 SWAP I 2 * + 2 + ! STOP
        ELSE OVER I 2 * + ! THEN LOOP DROP DROP
        _CPOL
        DEL CRLF \OK SAY DEL ;  ( ADD A SINGLE TERMINAL )

: BC PTPT @ 1 LEFT PTA = IF PTB ELSE PTA  THEN
    7F 0 DO FEL'CL \TERMINAL? SAY BUFFER WORD NUMBER DUP
        0 = IF I 0 = IF OVER I 2 * + ! NEXT
            ELSE OVER I 2 * + ! STOP
            THEN
        THEN
    8000 + OVER I 2 * + ! LOOP DROP
    _CPOL DEL
    CRLF \OK SAY ;

: BE FEL'CL \NEWTIME SAY CRLF STOPCLOCK STOD ZTIME DEL DEL ;
( COMMAND LINE INTERPRETER )

: CLI FRAME FEL'CL 0 DIS! BUFFER WORD FIND EXECUTE FEL'CL UNFRAME ;

( RUN TIME SYSTEM )

: RUN BEGIN FAILURE @ 0 > IF 4 DIS! THEN
          ACHAR @ 0 > IF CLI THEN
 DISPLAY @ DUP 4 = IF ZFAIL ELSE
          DUP 3 = IF ZTIME ELSE
          DUP 2 = IF ZTERM ELSE
          DUP 1 = IF ZCONT ELSE
          FEL'CL \HEAD SAY 0 DIS! DEL
          THEN THEN THEN THEN DROP 38 CRB2 !B FF DRB2 !B 3C CRB2 !B END ;


( THE FIDDLY BITS !!! )

  ( THE TRAP ERROR AND ABORT BUTTON HANDLER GUTS )
: ALT LOCATE OUT'FEL #INT @ + 100A !L  ( ALTERNATIVE O/P ROUTINE )
     LOCATE R2       #INT @ + 100E !L  ( ALTERNATIVE I/P ROUTINE )
;
: TRAP INIT'FEL FEL'CL SET'PAD ALT 0 DUP ALINE ! DUP ACHAR ! DUP RP !
 DUP WP ! F CSTORE F CREAD A F CSTORE F CREAD DROP DROP
 OFFINT CCW ! ONINT CCW ! 0 FAILURE ! RESTART DEC ;

: SABORT DISINT CRLF STRING %SABORT% SAY TRAP ENINT INTER ;
: SERROR DISINT CRLF STRING %ERROR % SAY TRAP ENINT INTER ;

: SETUP DISINT 0 STORE ! LOCATE SERROR #INT @ +
       100 1 DO DUP STORE @ 4 + DUP STORE !
             !L LOOP DROP                  ( SET UP FOR TRAP ERROR )
        LOCATE SABORT #INT @ + 7C !L      ( ABORT BUTTON HANDLER )
        LOCATE IRQ4 #INT @ + 70 !L    ( KEYPAD HANDLER )
        LOCATE IRQ3 #INT @ + 6C !L    ( CLOCK HANDLER )
        LOCATE IRQ2 #INT @ + 68 !L     ( 2901 HANDLER )
        SETPER
        ALT
        PTB 10 0 DO DUP I 2 * + I SWAP ! LOOP DROP
0 FAILURE !
0 CONTENTIONS !
0 ALINE !
0 ACHAR !
0 RP !
0 WP !
0 WHERE !
A F CSTORE
F CREAD F CREAD F CREAD
DROP DROP DROP
_CPOL
       ENINT ;

: 1RESET LOCATE RUN #INT @ + EQUATES ! ;
: 2RESET 264C EQUATES ! ;
: _RES RESTART SETUP DEC 1RESET RUN ;     ( RESTART VECTOR HANDLER )
: 3RESET LOCATE _RES #INT @ + EQUATES ! ;  ( USED ONLY FOR RESTARTING )

: AAA 2RESET 21C2E 100A !L    ( MACSBUG O/P ROUTINE )
            21DB4 100E !L    ( MACSBUG I/P ROUTINE )
            INTER ;
%ENDFILE
```

```
: CONTENTIONS VARIABLE ;
: FAILURE VARIABLE ;

: IRQ2 FRAME #CONT 1 = IF CONTENTIONS ^1 ( CONTENTION HAS OCCURRED )
                     THEN #FAIL 0 > IF FF FAILURE ! ( STORE FAILURE )
                                    THEN
       UNFRAME RTE ;

: WHERE VARIABLE ;
: WHERE? WHERE @ 3F MASK ;

: BCD DUP 9 > IF A / DUP F 10 LEFT MASK 80 / 200 / ( GET DEC NUMBERS )
             SWAP F MASK ELSE 0 THEN SWAP ;

  ( GET TOD FROM IN-TIME FIELD, ONLY USED WHEN NOT ACTIVE !! )
: #STOD IT 2 + @ 400 / F MASK BCD    ( MONTH )
        IT 2 + @ 20 / 1F MASK BCD    ( DAY )
        IT 2 + @ 1F MASK BCD         ( HOUR )
        IT @ 400 / 3F MASK BCD       ( MINUTES )
    STOPCLOCK
    9 4 DO I CSTORE LOOP
    C 8 DO I CSTORE LOOP
    GOCLOCK  ;
: #RES #ACTIVE 1 = IF #STOP 0 = IF WHERE? 2 * STT + @ C000 MASK
      0 = IF
          ELSE PTPT @ DUP 1 LEFT PTA = IF PTB ELSE PTA THEN
      7F 0 DO DUP I 2 * + DUP @ 3F MASK SWAP OVER SWAP !
           WHERE? = IF DUP I 2 * + WHERE? 8000 + SWAP !
                   THEN LOOP
      2 /
      CSTOP CCW ! WSTOP
      PTPT !
      GO CCW !
      0 STT WHERE? 2 * + !
      CSTOP CCW ! WSTOP
      PTPT !
      GO CCW !
      THEN WHERE ^1 THEN THEN ;

( INTERRUPT HANDLER FOR CLOCK )
( WILL UPDATE OUT-TIME IF OTA IS CLEAR )

WIDTH 4 + CONSTANT TOT
: IRQ3 FRAME 0 T_DIS ! F CREAD  DROP
       CTUSW @ 3 MASK 2 = IF OTA @ 8000 MASK 0 = IF
              0 TOT ! GTOD          ( SET UP FOR START )
              A * + 5 LEFT TOT !
              A * + TOT @ + 5 LEFT TOT !
              A * + TOT @ + OT 2 + !  ( MONTH, DAY & HOURS )
              0 TOT !
              A * + 6 LEFT TOT ! A * + TOT @ + 4 LEFT + OT !
                                    ( MINUTES SECONDS TENTHS )
              CTIME 2 + @ CTIME 4 + @ CTIME @ ( GET SYNCH TIME )
                1000 * 7BD + OT 4 + ! ( 1/10THS AND YEAR )
              OT 6 + !
              OT 8 + !                  ( MULTIPLES OF TENTHS )
              8000 OTA !
              THEN
              20 0 DO #RES LOOP
     ELSE ITA @ DUP 8000 MASK 0 = IF
          ELSE #STOD 7FFF MASK ITA !
          THEN
      THEN
      UNFRAME RTE ;
```

```
( IRQ3 HANDLER AND CIRCULAR BUFFER DRIVING ROUTINES )
( FOR HEX KEYPAD )

10 : ST2 ARRAY ;  ( CIRCULAR BUFFER )
: ALINE VARIABLE ;
: WP VARIABLE ;
: RP VARIABLE ;
: ACHAR VARIABLE ;


: IRQ4 FRAME DRA2 @B DUP 9 > IF 37 + ELSE 30 +    ( FORM ASCII CHAR )
           THEN WP @ 1 + RP @ = IF DROP UNFRAME RTE ( BUFF FULL )
           THEN DUP 46 = IF DROP OD ALINE ^1 ( USE 'F' AS CR )
           THEN WP @ DUP 1F < IF WP ^1         ( INC WRITE POINTER )
                              ELSE RP @ 0 > IF 0 WP !  ( WRAPAROUND )
                                   ELSE DROP DROP UNFRAME RTE
                                   THEN
                              THEN ST2 SWAP DROP + !B ( STORE IT )
           ACHAR ^1             ( WE HAVE CHARS SO FLAG IT )
           UNFRAME RTE ;

( CIRCULAR BUFFER READ ROUTINE )
( WILL READ FROM HEX KEYPAD'S CIRCULAR BUFFER )

: PREAD ACHAR @ 0 = IF FF QUIT     ( NO CHARS SO QUIT !! )
       ELSE
        RP @ ST2 SWAP DROP + @B   ( GET CHAR FROM CIRC BUFFER )
        RP @ 1F = IF 0 RP ! ELSE RP ^1    ( FORM THE NEW POINTER )
                THEN
        ACHAR DUP @ 1- SWAP !       ( DECREMENT FLAG )
       THEN
 ;
( A READ ROUTINE SUITABLE TO BE PATCHED IN TO THE KERNEL )
: PAD_IN BEGIN PREAD DUP FF < IF QUIT ELSE DROP THEN END ;
: R2 PAD_IN POP ;            ( LEAVE CHAR I DO FOR BUFF ROUTINE )

( DISPLAY COMMANDS )

: ZNUMBER VARIABLE ;
: DISPLAY VARIABLE ;
: DIS! T_DIS ! ;
: ZTEST T_DIS @ OVER = IF DROP DROP DROP
                       ELSE T_DIS ! FEL^CL
                       SAY
                       THEN ;

: ZFAIL \ITFAILED 4 ZTEST DEL ;
: ZTIME \TIM 3 ZTEST TIME DEL ;
: ZTERM ZNUMBER @ DUP 0 = IF DROP ATDIS
                        ELSE 1- TDIS DEL
                        THEN ;
: ZCONT CDIS DEL ;

: DISP! DISPLAY ! ;
: DB 3 DISP! ;
: DC 1 DISP! ;
: DD 2 DISP! FEL^CL 0 DIS! \TERMINAL? SAY BUFFER WORD NUMBER 1 +
      ZNUMBER ! ;
: DE 2 DISP! 0 ZNUMBER ! ;

( CHANGE CONTROLLER FUNCTION COMMANDS )
: C1 GO CCW ! FEL^CL 0 DIS! \STARTING SAY DEL 20 PT^ ! \OK SAY DEL ;
```

```
: @CONT CTUSW @ 8 MASK 8 / ; ( STACKS CONTENTION BIT )
: @FAIL CTUSW @ FF00 MASK 100 / ; ( STACKS STORE FAIL BITS )
: @CABLE CTUCW @ 6 MASK 2 / ;  ( STACKS SELECTED CABLE )

( GET CONTROLLER STATISTICS )
: @GCSTATS RC @ NRC @ @CABLE @ACTIVE @STOP REC @ ;


( SOME USEFUL STRINGS )

: \NAK STRING %NAK STUCK % ;
: \BLANK STRING %              % ;
: \NR STRING %   NR    % ;
: \RESP STRING %RESPONDING% ;
: \ERR STRING %ERRORS  % ;
: \INFO STRING %INFORMATION MONITOR % ;
: \TERM STRING %TERMINAL NUMBER % ;
: \TIM STRING %SYSTEM TIME   % ;
: \HEAD STRING %***MICROLINK*** % ;
: \CONT STRING %HIGHWAY CONTROLLER% ;
: \RC STRING %REPEATS % ;
: \NRC STRING %NULL REPEATS% ;
: \CABLE STRING %CABLE % ;
: \ACT STRING %ACTIVE  % ;
: \PASS STRING %PASSIVE % ;
: \RUNNING STRING %RUNNING % ;
: \STOPPED STRING %STOPPED % ;
: \REC STRING %RECEIVE ERRORS% ;
: \STARTING STRING %START CONTROLLER% ;
: \STOPPING STRING %STOP CONTROLLER % ;
: \NEWTIME STRING %SET NEW TIME% ;
: \NEWSYNCH STRING %NEW SYNCH TIME  % ;
: \SELECT STRING %CABLE NO. ? % ;
: \PASSIVE STRING %GO PASSIVE% ;
: \OK STRING %OK% ;
: \NEWONE STRING %ADD TERMINAL NO.% ;
: \TERMINAL? STRING %TERMINAL NO.? % ;
: \RESTERM STRING %RESET TERMINAL ?% ;
: \RESETTING STRING %RESET CONTROLLER% ;
: \ITFAILED STRING %***INTERFACE FAILURE*** % ;

: SPACES 4 1 DO SPACE LOOP ;

( TERMINAL STATISTICS DISPLAY LOOP )
: TDIS T_DIS DUP @ 2 = IF DROP ELSE FEL'CL
                        \TERM SAY
                        40 PT' ! \ERR SAY
                        60 PT' ! \INFO SAY
                        2 SWAP !
                        THEN
  DUP
  11 PT' ! . SPACE            ( TERMINAL NUMBER )
  @GTSTATS
  20 PT' ! 1 = IF \NAK ELSE \BLANK THEN SAY
  34 PT' ! 1 = IF \NR ELSE \RESP THEN SAY
  54 PT' ! . SPACES   ( ERRORS )
  74 PT' ! . SPACES   ( IMM )
 ;

: DEL 1000 1 DO LOOP ;

( DISPLAY STATS FOR ALL OF THE TERMINALS )

: ATDIS PTPT @ 1 LEFT 40 0 DO DUP I 2 * + @ 3F MASK DUP
```

```
                  0 > IF TDIS ELSE I 0 = IF TDIS
                                         ELSE DROP STOP
                                         THEN
                             THEN
  DEL LOOP DROP ;

( CONTROLLER STATISTICS DISPLAY LOOP )

: CDIS T_DIS DUP @ 1 = IF DROP  ( OK CONT ALREADY DISPLAYED )
      ELSE FEL'CL \CONT SAY
           20 PT' ! \REC SAY
           40 PT' ! \NRC SAY 57 PT' ! \CABLE SAY
           60 PT' ! \RC SAY
           1 SWAP !             ( NOW CONTROLLER STATS. BEING DISPLAYED
      THEN
  @GCSTATS
  30 PT' ! .                    ( RECEIVE ERRORS )
  37 PT' ! 1 = IF \STOPPED ELSE \RUNNING THEN SAY
  17 PT' ! 1 = IF \ACT ELSE \PASS THEN SAY
  5E PT' ! .                    ( CABLE NUMBER )
  50 PT' ! .                    ( NULL REPEATS )
  70 PT' ! .                    ( REPEAT COUNT )
 ;
: SET'PAD 00 CRA2 !B F0 DRA2 !B 7 CRA2 !B ; ( 4 I/P'S 4 O/P'S )
: SHUTDOWN SYRES 0 F CSTORE DISINT ;    (  ROUTINE TO STOP THE INTS )
: CLEARTAB DUP FF + SWAP DO 0 I !B LOOP ;  ( CLEAR CONT. TABLE )
: SETPER               ( SETUP PERIPHERALS AND CLEAR TABLES )
  RESET CCW !           ( NOW FOR THE CONTROLLER )
  PT CLEARTAB
  PTA CLEARTAB
  PTB CLEARTAB
  MSS CLEARTAB
  BS CLEARTAB
  STT CLEARTAB         ( CLEAR A BIT OF ALL THE TABLES )
  PTA 2 / PTPT !       ( GIVE IT SOMETHING TO CHEW ON )
  1 CTUSW !            ( SET STOPPED TO AVOID CONFUSION )
  0 0 CSTORE
  0 F CSTORE F CREAD  F CREAD F CREAD DROP DROP DROP
  A F CSTORE F CREAD F CREAD F CREAD DROP DROP DROP
      ( SET UP THE REAL TIME CLOCK )
  INIT'FEL FEL'CL     ( SET UP FELTEC DISPLAY )
  SET'PAD             ( SET UP HEX KEYPAD )
  ;


: WSTOP BEGIN @STOP 1 = IF QUIT THEN END ;
: _CPOL PTPT @ 1 LEFT DUP PTA = IF PTB ELSE PTA THEN DUP
                                       ( GET SECONDARY TABLE )
      2 /
      CSTOP CCW ! WSTOP        ( FLAG STOP AND WAIT )
      PTPT !                   ( STORE SECONDARY )
      GO CCW !                 ( RESTART CONTROLLER )
      STT CLEARTAB
      7F 0 DO OVER OVER I 2 * + @  ( READ FROM NEW TABLE )
      7FFF MASK SWAP I 2 * + !     ( PUT IN OLD TABLE )
      LOOP
      DROP 2 /
      CSTOP CCW ! WSTOP        ( FLAG STOP & WAIT )
       PTPT !                  ( PUT BACK MODIFIED NEW TABLE )
      GO CCW ! ONINT CCW ! ;             ( RESTART CONTROLLER )


( INTERRUPT HANDLER FOR 2901 )
```

```
0025E0  0001    XADD    DC.B #1
0025E2  2B202020        DC.L '+   '
0025E6  25D2            DC.W X1L
                *
                * BASIC ARITHMETIC OPERATIONS
                *
0025E8  201E    ADD     MOVE.L (A6)+,D0
0025EA  221E            MOVE.L (A6)+,D1
0025EC  D280            ADD.L D0,D1
0025EE  2D01            MOVE.L D1,-(A6)
0025F0  4E75            RTS
                *
                *
0025F2  0001    XSUB    DC.B #1
0025F4  2D202020        DC.L '-   '
0025F8  25E0            DC.W XADD
                *
                * SUBTRACTS 32 BITS
                *
0025FA  221E    SUB     MOVE.L (A6)+,D1
0025FC  201E            MOVE.L (A6)+,D0
0025FE  9081            SUB.L D1,D0
002600  2D00            MOVE.L D0,-(A6)
002602  4E75            RTS
                *
                *
                * MULTIPLY 16 BY 16 TO 32
                *
002604  0001    XMUL    DC.B #1
002606  2A202020        DC.L '*   '
00260A  25F2            DC.W XSUB
00260C  201E    MUL     MOVE.L (A6)+,D0
00260E  221E            MOVE.L (A6)+,D1
002610  C0C1            MULU D1,D0
002612  2D00            MOVE.L D0,-(A6)
002614  4E75            RTS
                *
                * DIVIDE 32 BY 16 EQUALS 16, REM 16
                *
002616  0001    XDIV    DC.B #1
002618  2F202020        DC.L '/   '
00261C  2604            DC.W XMUL
00261E  221E    DIV     MOVE.L (A6)+,D1
002620  201E            MOVE.L (A6)+,D0
002622  80C1            DIVU D1,D0
002624  2D00            MOVE.L D0,-(A6)
002626  4E75            RTS
                *
002628  0004    XLEFT   DC.B #4
00262A  4C454654        DC.L 'LEFT'
00262E  2616            DC.W XDIV
                *
                * MOVE LEFT BY N PLACES
                *
002630  201E    LEFT    MOVE.L (A6)+,D0
002632  221E            MOVE.L (A6)+,D1
002634  E1A9            LSL.L D0,D1
002636  2D01            MOVE.L D1,-(A6)
002638  4E75            RTS
                *
00263A  0004    XSWAP   DC.B #4
00263C  53574150        DC.L 'SWAP'
002640  2628            DC.W XLEFT
                *
                * WILL SWAP THE TOP TWO ON THE OLD STACK
                *
002642  201E    SWAP    MOVE.L (A6)+,D0
002644  221E            MOVE.L (A6)+,D1
002646  2D00            MOVE.L D0,-(A6)
002648  2D01            MOVE.L D1,-(A6)
00264A  4E75            RTS
                *
                *
                * THE INTERPRET LOOP
                *
                *
00264C  2E781016    INTE1   MOVE.L MSTCK,A7
002650  4EB8222E            JSR SREST
002654  0C3800001036 INTE2  CMPI #0,RELO
00265A  6700000A            BEQ INTE0
00265E  4EB8240E            JSR LOAD
002662  6000000C            BRA INTE3
002666  21F810021038 INTE0  MOVE.L    ACIA1,PORT
00266C  4EB8203E            JSR BUFF
002670  4EB820B4    INTE3   JSR WORD
002674  4EB82104            JSR FIND
002678  4EB822B0            JSR EXEC
00267C  0C3800011028        CMPI #1,LAST
002682  6700FFD0            BEQ INTE2
002686  6000FFE8            BRA INTE3
 Rdos Error code -6

[SYS 8.6]
```

```
0024BE  4E75                RTS
                    *
                    *
0024C0  0007        XDISS      DC.B #7
0024C2  44495353          DC.L 'DISS'
0024C6  24B0              DC.W XTO
0024C8  241E        DISS       MOVE.L (A6)+,D2
0024CA  4283              CLR.L D3
0024CC  4281              CLR.L D1
0024CE  3202              MOVE.W D2,D1
0024D0  82F8102A    XPRN3      DIVU RDX,D1
0024D4  3801              MOVE.W D1,D4              DO THIS INSTEAD OF SWAP
0024D6  E089              LSR.L #8,D1       WHICH DOESNT WORK
0024D8  E089              LSR.L #8,D1
0024DA  4280              CLR.L D0
0024DC  1001              MOVE.B D1,D0
0024DE  223C00000000      MOVE.L #00,D1
0024E4  3204              MOVE.W D4,D1              DITTO HERE
0024E6  0C000009          CMPI #9,D0
0024EA  6E00000A          BGT XPRN2
0024EE  06000030          ADDI #$30,D0
0024F2  60000006          BRA XPRN4
0024F6  06000037    XPRN2      ADDI #$37,D0
0024FA  4EB82166    XPRN4      JSR PUSH
0024FE  5203              ADDQ #1,D3              CHAR COUNT,ALWAYS >=1
002500  0C440000          CMPI.W #00,D4              ANYTHING LEFT??
002504  6600FFCA          BNE XPRN3              YES SO TRY AGAIN
002508  E08A              LSR.L      #8,D2              NOW TRY TO SEE FOR 32 BI
00250A  E08A              LSR.L      #8,D2
00250C  6F000028          BLE      XPRN6
002510  0C030004          CMPI #4,D3
002514  67000016          BEQ XPRN7
002518  103C0030    XPRN8      MOVE #$30,D0
00251C  4EB82166          JSR PUSH
002520  06030001          ADDI #1,D3
002524  0C030004          CMPI #4,D3
002528  6D00FFEE          BLT XPRN8
00252C  4281        XPRN7      CLR.L      D1
00252E  3202              MOVE.W     D2,D1
002530  4282              CLR.L      D2
002532  6000FF9C          BRA      XPRN3
002536  1003        XPRN6      MOVE D3,D0
002538  4EB82166          JSR PUSH
00253C  4E75              RTS
                    * THE REAL PRINT ROUTINE
                    *
00253E  0001        X.      DC.B #1
002540  2E202020          DC.L '.   '
002544  24C0              DC.W XDISS
002546  4EB824C8    PRN     JSR DISS
00254A  4EB82172          JSR POP
00254E  4283              CLR.L D3
002550  1600              MOVE D0,D3
002552  4EB824B8    XPRN5      JSR TO
002556  04030001          SUBI #1,D3
00255A  6600FFF6          BNE XPRN5
00255E  4E75              RTS
                    *
                    *
002560  0004        XASMB      DC.B #4
002562  41534D42          DC.L 'ASMB'
002566  253E              DC.W     X.
                    *
                    * WILL PUT A NUMBER ON THE STACK AS IS INTO THE DIC
```

```
002568  3478102E    ASMB     MOVE.W DP,A2
00256C  343C0004         MOVE.W #$4,D2
002570  14DE        BK3      MOVE (A6)+,(A2)+
002572  5302             SUBQ #1,D2
002574  6600FFFA         BNE BK3
002578  31CA102E         MOVE.W A2,DP
00257C  4E75             RTS
00257E  4E75             RTS
                    *
                    *
002580  0002        X@B      DC.B #2
002582  40422020         DC.L '@B  '
002586  2560             DC.W XASMB
002588  225E        @B       MOVE.L (A6)+,A1
00258A  4280             CLR.L D0
00258C  1011             MOVE.B (A1),D0
00258E  2D00             MOVE.L D0,-(A6)
002590  4E75             RTS
                    *WORD GET
002592  0002        X@W      DC.B #2
002594  40572020         DC.L '@W  '
002598  2580             DC.W X@B
00259A  225E        @W       MOVE.L (A6)+,A1
00259C  4280             CLR.L D0
00259E  3011             MOVE.W (A1),D0
0025A0  2D00             MOVE.L D0,-(A6)
0025A2  4E75             RTS
                    *LONG WORD
0025A4  0002        X@L      DC.B #2
0025A6  404C2020         DC.L '@L  '
0025AA  2592             DC.W X@W
0025AC  225E             MOVE.L (A6)+,A1
0025AE  2D11             MOVE.L (A1),-(A6)
0025B0  4E75             RTS
                    *
0025B2  0002        X!B      DC.B #2
0025B4  21422020         DC.L '!B  '
0025B8  25A4             DC.W X@L
0025BA  225E        !B       MOVE.L (A6)+,A1
0025BC  201E             MOVE.L (A6)+,D0
0025BE  1280             MOVE.B D0,(A1)
0025C0  4E75             RTS
                    *
                    *
                    * STORE A WORD
0025C2  0002        X!W      DC.B #2
0025C4  21572020         DC.L '!W  '
0025C8  25B2             DC.W X!B
0025CA  225E        !W       MOVE.L (A6)+,A1
0025CC  201E             MOVE.L (A6)+,D0
0025CE  3280             MOVE.W D0,(A1)
0025D0  4E75             RTS
                    *
                    *
                    * STORE A LONG WORD
0025D2  0002        X!L      DC.B #2
0025D4  214C2020         DC.L '!L  '
0025D8  25C2             DC.W X!W
0025DA  225E        !L       MOVE.L (A6)+,A1
0025DC  229E             MOVE.L (A6)+,(A1)
0025DE  4E75             RTS
                    *
                    *
                    *
```

```
002372  2A4D4143            DC.L '*MAC'
002376  53204940            DC.L 'S IM'
00237A  554D2056            DC.L 'UM V'
00237E  322E312A            DC.L '2.1*'
                    *
                    *
002382  0101        XSEMI   DC.B $101
002384  3B202020            DC.L ';   '
002388  2352                DC.W XTITL
                    *
                    * ENDS COMPILE MODE
                    *
00238A  043800011030 SEMI   SUBI #1,STATE
002390  6C000006            BGE AUTO
002394  4EB8222E            JSR SREST     FORCE A RESTART SINCE STATE IS NOW
002398  3478102E    AUTO    MOVE.W DP,A2
00239C  34FC4E75            MOVE.W #4E75,(A2)+
0023A0  31CA102E            MOVE.W A2,DP
0023A4  31FC0040101E        MOVE.W #$40,PREF
0023AA  4E75                RTS
                    *
                    *
0023AC  0008        XCONS   DC.B #8
0023AE  434F4E53            DC.L 'CONS'
0023B2  2382                DC.W XSEMI
                    *
                    * WILL INTERPRET COMPILATION NUMBERS
                    *
0023B4  4EB82272    CONS    JSR COLON+6
0023B8  3478102E            MOVE.W DP,A2
0023BC  34FC2D3C            MOVE.W #$2D3C,(A2)+
0023C0  4EB8256C            JSR ASMB+4
0023C4  4EB82398            JSR SEMI+0E
0023C8  4E75                RTS
                    *
                    *
                    *INTEGER ROUTINE
0023CA  0007        XINTE   DC.B #07
0023CC  494E5445            DC.L 'INTE'
0023D0  23AC                DC.W XCONS
0023D2  3478102E    INTE    MOVE.W DP,A2
0023D6  34FC2D3C            MOVE.W #$2D3C,(A2)+     EQU MOVE.L #XXXX,-(A6)
0023DA  4EB8256C            JSR ASMB+4
0023DE  4E75                RTS
                    *VARIABLE
0023E0  0108        XVAR    DC.B #108
0023E2  56415249            DC.L 'VARI'
0023E6  23CA                DC.W XINTE
0023E8  3478102E    VAR     MOVE.W DP,A2
0023EC  34FC4EB8            MOVE.W #$4EB8,(A2)+           JSR #XXXX
0023F0  34FC2400            MOVE.W #VAR2,(A2)+
0023F4  24FC00000000        MOVE.L #00,(A2)+           SPACE FOR VAR
0023FA  31CA102E            MOVE.W A2,DP                GET DP BACK
0023FE  4E75                RTS
002400  2D17        VAR2    MOVE.L (A7),-(A6)               GETS ADDRES OF VAR
002402  5897                ADDQ.L #4,(A7)
002404  4E75                RTS
002406  0004        XLOAD   DC.B #4
002408  4C4F4144            DC.L 'LOAD'
00240C  23E0                DC.W XVAR
                    *
                    * WILL LOAD SOURCE TEXT FROM HOST
                    *  PROGRAM 68KLOAD MUST BE RUNNING
                    *  AND 68KDIC PRESENT IN THE SYSTEM
                    *
```

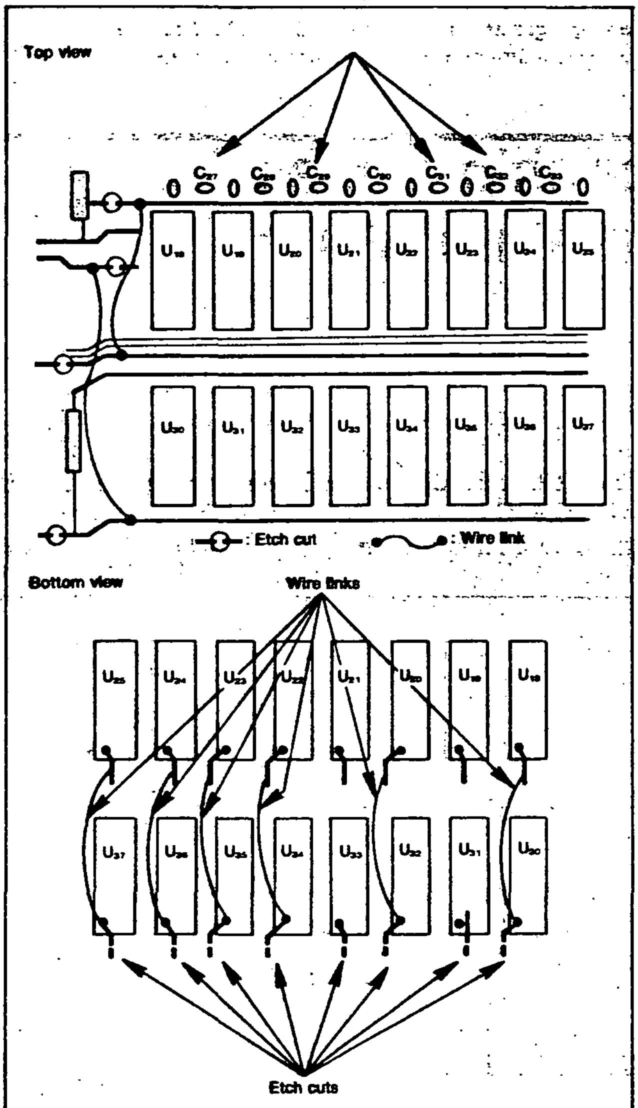
```
00240E  34781020    LOAD    MOVE.W STBUF,A2
002412  21F810061038        MOVE.L ACIA2,PORT
002418  103C001B            MOVE #$1B,D0
00241C  4EB8201C            JSR SOUT
002420  4EB82008    GETCH   JSR SIN
002424  0C00000D            CMPI #$0D,D0
002428  66000008            BNE LOAD2
00242C  14C0                MOVE D0,(A2)+
00242E  60000010            BRA LOAD3
002432  0C000020    LOAD2   CMPI #' ',D0
002436  6D00FFE8            BLT GETCH
00243A  14C0                MOVE D0,(A2)+
00243C  6000FFE2            BRA GETCH
002440  31F810201024 LOAD3  MOVE.W STBUF,EWORD     FOR WORD ROUTINES
002446  31FC00001028        MOVE.W #0,LAST
00244C  4E75                RTS
                    *
                    *STARTS THE DOS TRANSFER PROGRAM RUNNING
                    *THIS PROGRAM TERMINATES ITSELF ON
                    *EOF AND RESPONDS TO ESC PROMPTS
00244E  0004        XOPEN   DC.B #4
002450  4F50454E            DC.L 'OPEN'
002454  2406                DC.W XLOAD
002456  21F810061038 OPEN   MOVE.L ACIA2,PORT
00245C  267C00002482        MOVE.L #COMM,A3
002462  263C0000000D        MOVE.L #$0D,D3
002468  101B        OPEN1   MOVE.B (A3)+,D0
00246A  4EB8201C            JSR SOUT
00246E  5303                SUBQ #1,D3
002470  6600FFF6            BNE OPEN1
002474  4EB82008    OPEN2   JSR SIN
002478  0C00000D            CMPI #$0D,D0
00247C  6600FFF6            BNE OPEN2
002480  4E75                RTS
                    *
                    * COMMAND LINE FOR DOS
002482  52554E2C    COMM    DC.L 'RUN,'
002486  313A4C44            DC.L '1:LD'
00248A  2E36384B            DC.L '.68K'
00248E  0D00                DC.W $0D00
                    *
                    *
002490  0006        XRELD   DC.B #6
002492  52454C4F            DC.L 'RELO'
002496  244E                DC.W XOPEN
                    *
                    * ALL THIS DOES IS SET THE RELOAD FLAG
                    *
002498  11FC00011036 RELD   MOVE #1,RELO
00249E  4E75                RTS           IS IT WORTH IT !
                    *
                    *IMMEDIATE MODE
0024A0  0109        XIMM    DC.B #109
0024A2  494D4D45            DC.L 'IMME'
0024A6  2490                DC.W XRELD
0024A8  34781026            MOVE.W DLAST,A2
0024AC  5212                ADDQ.B #1,(A2)
0024AE  4E75                RTS
                    *
                    *
0024B0  0002        XTO     DC.B #2
0024B2  544F2020            DC.L 'TO  '
0024B6  24A0                DC.W XIMM
0024B8  201E        TO      MOVE.L (A6)+,D0
0024BA  4EB8201C            JSR SOUT
```

APPENDIX B

An Upgrade of On-Board Memory

Boost μP-board memory capacity with simple hardware changes

Some minor rewiring and the addition of an inexpensive data selector chip tailors Motorola's MC68000 evaluation board, the MEX68KDM, for a fourfold increase in on-board memory capacity. The board's complement of sixteen MCM4116 RAMs (16-kbit devices) can be replaced with 64-kbit types, such as the 4164. The procedure costs less than one tenth the expense of an EXORciser chassis and additional memory modules.



1. Several etch cuts and wire jumpers help reconfigure the MEX68KDM evaluation board for operation with 64-kbyte RAMs. The greater-capacity memory chips boost on-board storage from 32 to 128 kbytes.

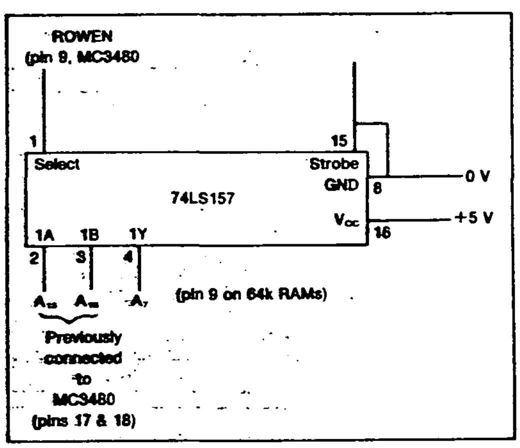
Key to the modification is in the design of the 64-kbit RAMs. The pinout of these devices is almost identical to that of the 16-kbit devices used in the board. In addition, such 64-kbit devices as the MCM4664 or HM4864 have the same refresh requirements as the MCM4116 used on the evaluation board.

Both sizes of RAM chip require a 128 row-address count with a 2-ms time interval. The board's existing multiplexer/refresh counter (MC3242) and memory controller (MC3480) can be used for the 64-kbit devices since the additional address line to the chip (pin 9 on the 64-k RAMs) is not used during the refresh cycle. Only an SN74LS157 two-to-one data selector is needed to multiplex the extra two address lines onto the 64-k chips.

To accomplish the changeover, the 5-V supply line to pin 1 of the RAMs must be disconnected. All 64-k parts are single-supply types. Etch cuts can be made between the memory chips and the controller so that only one wire must be added to connect together each RAM's pin 1. To allow pin-1 refresh, this rail must be tied high via a 1-kΩ resistor.

The +12-V connection to pin 8 must be disconnected and replaced by a connection to +5 V. A single etch cut and the addition of a single wire accomplishes this change. As shown in Fig. 1, the +5-V connection to pin 9 must be disconnected to allow connection of an additional address line to this pin on the 64-k devices. Four decoupling capacitors must be removed, nine tracks cut, and six wires added.

The address lines to pins 17 and 18 of the MC3480



2. An inexpensive data selector chip, the 74LS157, is the only additional hardware required for modification of the evaluation board.

must be disconnected and reconnected to the data selector (Fig. 2). The controller's pins 17 and 18 must be grounded so that RAS_1 and CAS are selected. The remainder of the data selector's pins are connected as shown. The +12 and −5-V connections to the ROM jumper area must be reconnected directly to the supply rails. The board's PROM, an N82S129, must be reprogrammed to the pattern shown in Fig. 3. A 74LS287 may be substituted.

In operation, line A_{15} or A_{16} of the board is multiplexed onto pin 9 of the 64-k RAMs, depending on the state of ROWEN (the line used by the MC3242 for multiplexing of the other 14 address lines). During a refresh cycle, the state of the additional address line is not important.

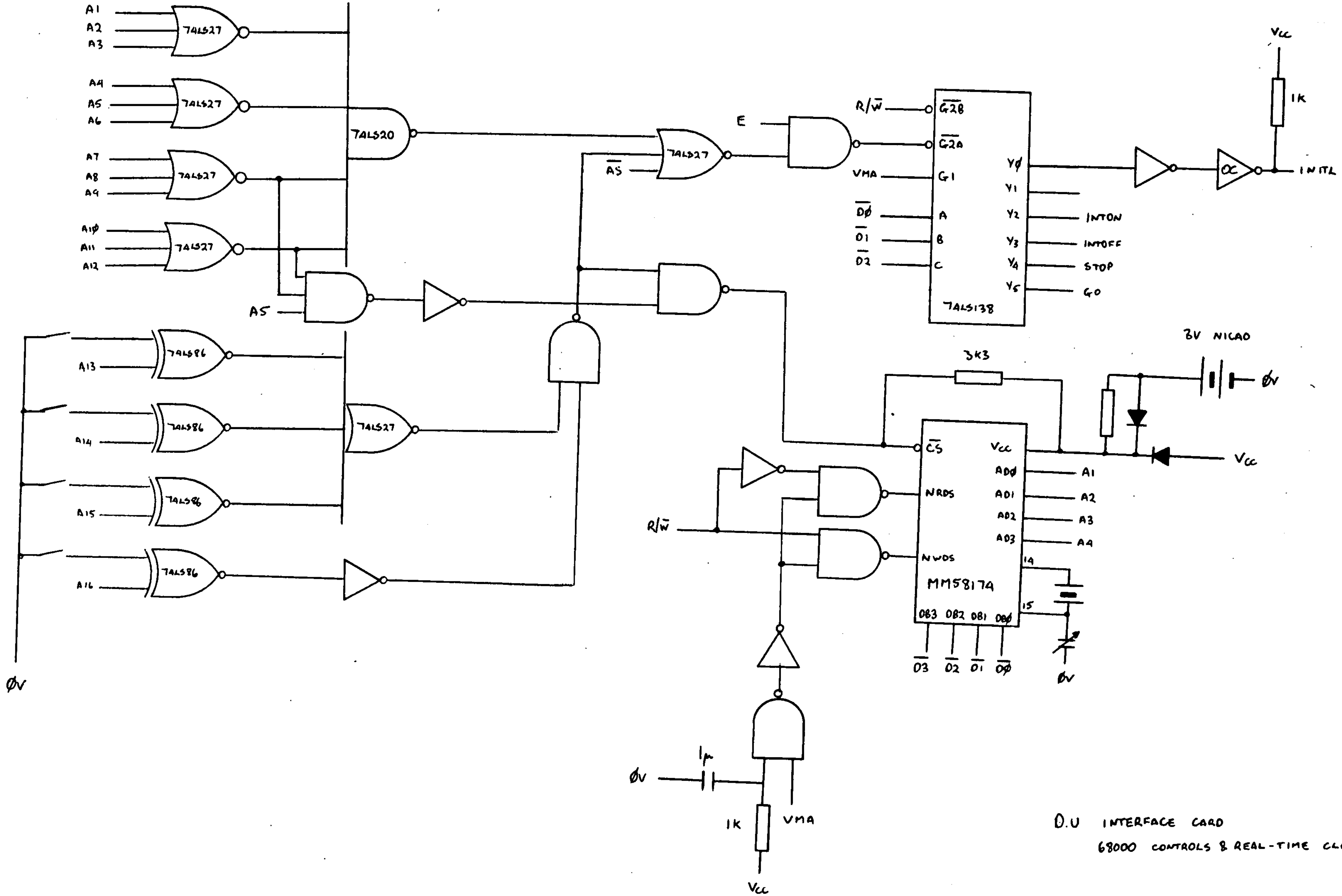
David Cowan, Research Assistant, Department of Applied Physics & Electronics, Durham University, South Rd., Durham DH1 3LE, United Kingdom.

00	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
10	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
20	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
30	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
40	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
50	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
60	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
70	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C
80	00	08	00	08	00	08	00	08	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	02	02

3. The evaluation board's bipolar PROM, an N82S129, must be reprogrammed as above. An SN74LS287 can be used if a Texas Instruments' programmer is more readily available.

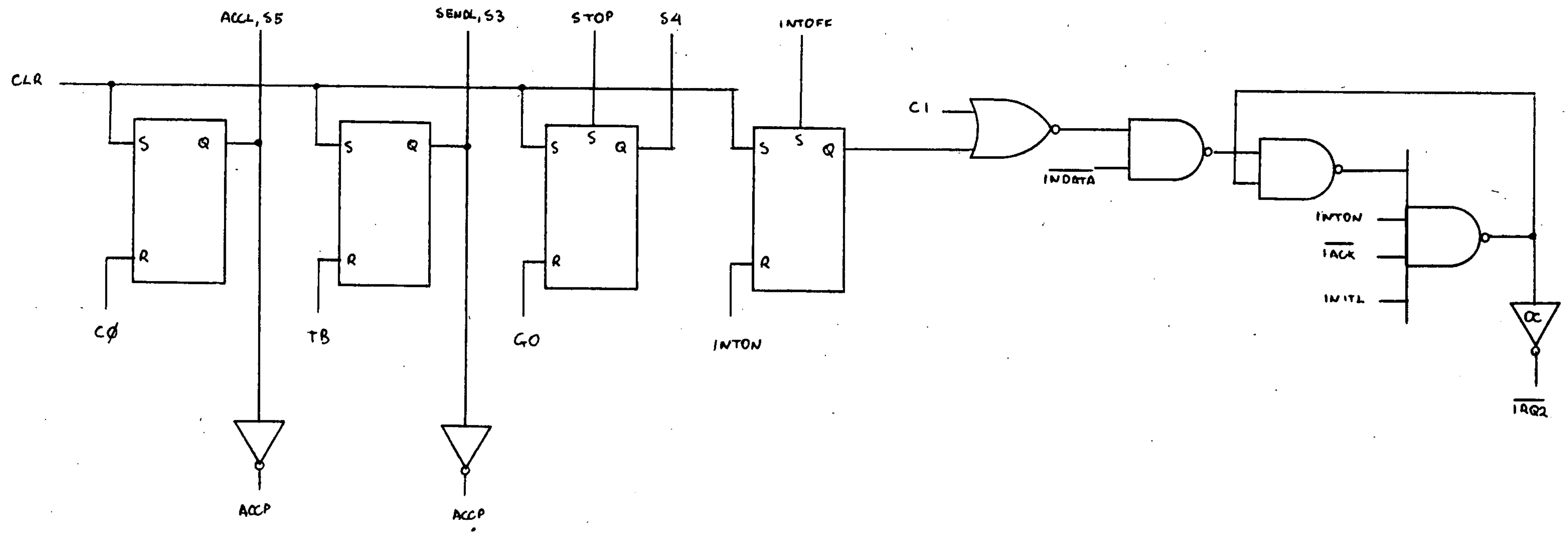
APPENDIX C

DMA Interface Circuit Diagrams

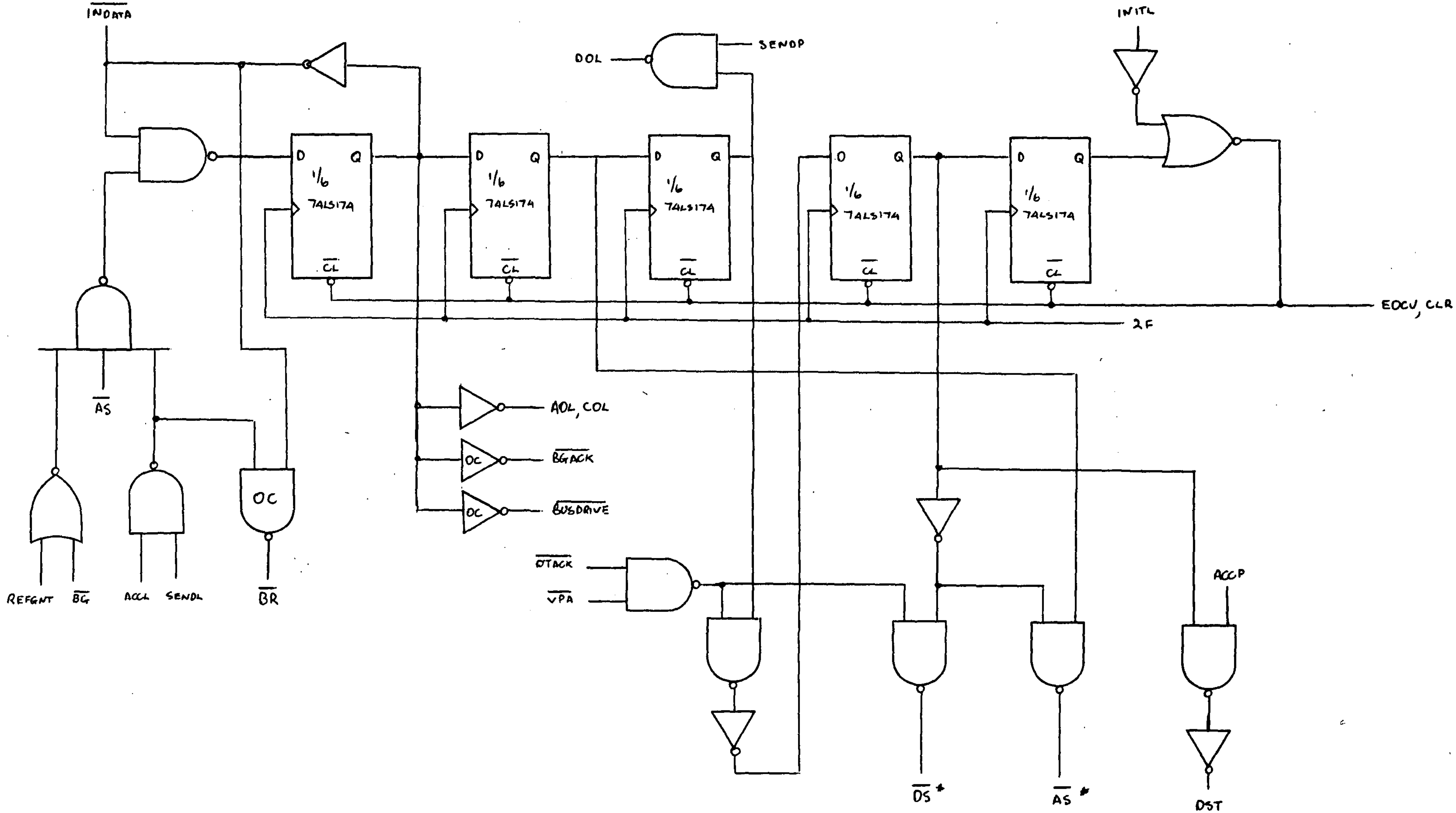


D.U INTERFACE CARD

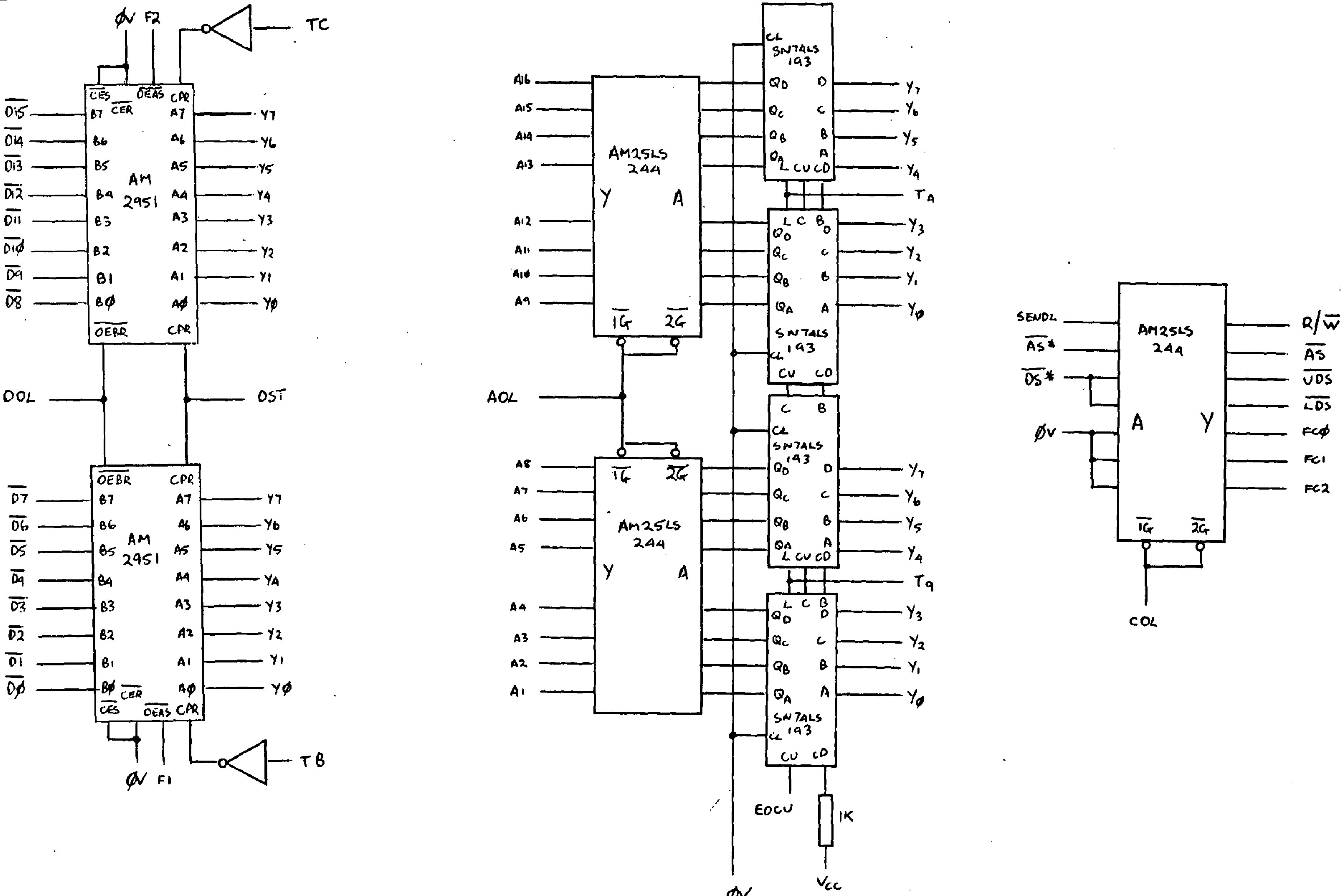
68000 CONTROLS & REAL-TIME CLOCK



D.U. INTERFACE CARD
2801 CONTROLS

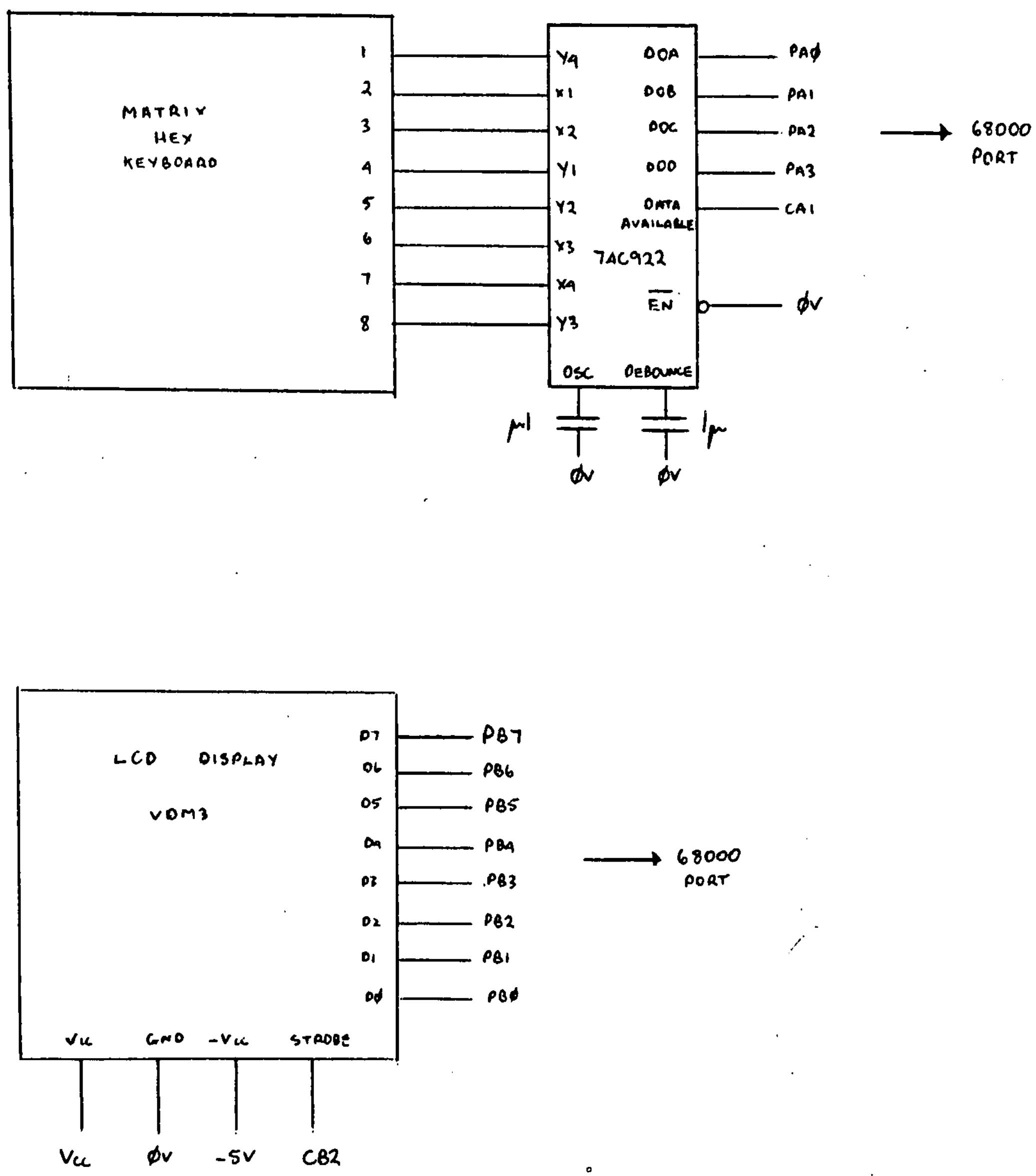


D.U INTERFACE CARD
MASTER CONTROL SECTION



D.U INTERFACE CARD

2901 → 68000 BUFFERS/ADDRESS COUNTERS



D.U. INTERFACE CARD
KEYPAD & LCD DISPLAY

Appendix D

Portable Highway Controller Commands

Portable Highway Controller Guide

Portable ASH Controller

User Instructions

The instructions which may be used by the operator are normally two character commands, terminated by an 'F'. The commands fall into three groups, as follows:-

Controller Operation

C1 Start Controller, clear and enable interrupts to the MC68000 host.

C2 Stop Controller.

C3 Reset Controller and disable interrupts.

CA Direct Controller to 'Go Passive'.

CC Change cable, prompts for desired cable number.

General Highway Operation

BA Reset a single terminal unit. Prompts for relevant terminal unit.

BB Add a single terminal to the polling scheme.

BC Set up a complete new polling table.

BD Reset the system sync. time to zero.

BE Set up a new time of day. Enter a single character per line, as follows:- months*10, months, days*10, days, hours*10, hours, minutes*10, minutes.

Monitoring Display

DB Display system time.

DC Display controller status.

DD Display statistics for a particular terminal unit.

DE Display the statistsics of all terminal units.

Executive Systems

AAA Return executive control to the VDU.

On power-up, the portable highway controller unit performs a reboot of the operating system for the MC68000. It then resets the ASH controller tables to their normal default conditions, including a polling table which consists of terminal units 0-16 inclusive. The MC68000 will then start the controller, which can then assume control of the ASH system, should it be the only active controller. The polling table may then be reset by the use of the **BB** or **BC** commands. After using the **AAA** command to redirect executive control to the VDU, control may be returned to the keypad/ front-panel display, by typing **_RES.**

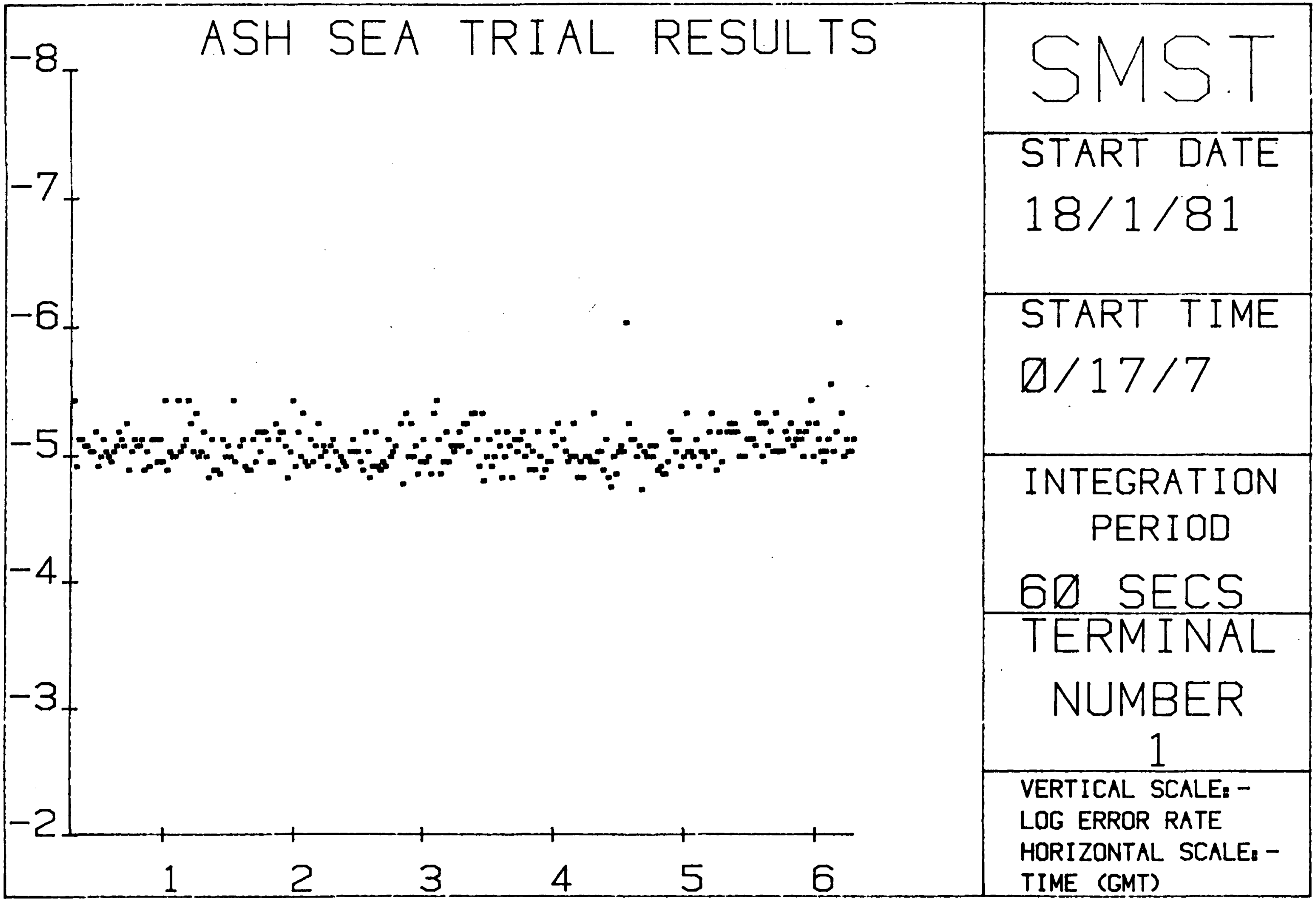
Documented Bugs

1) Display Corruption. Due to a fault in the address decoding for the PIAs on the MEX68KDM board, occasional corruption of their control and data register contents can occur. To protect against this, the registers of the PIA which drive the keypad and display are checked and updated more often than necessary. This gives rise to the periodically flashing display unit. It will be possible to remove these additional checks in the next controller, which will not use the MEX68KDM boards.

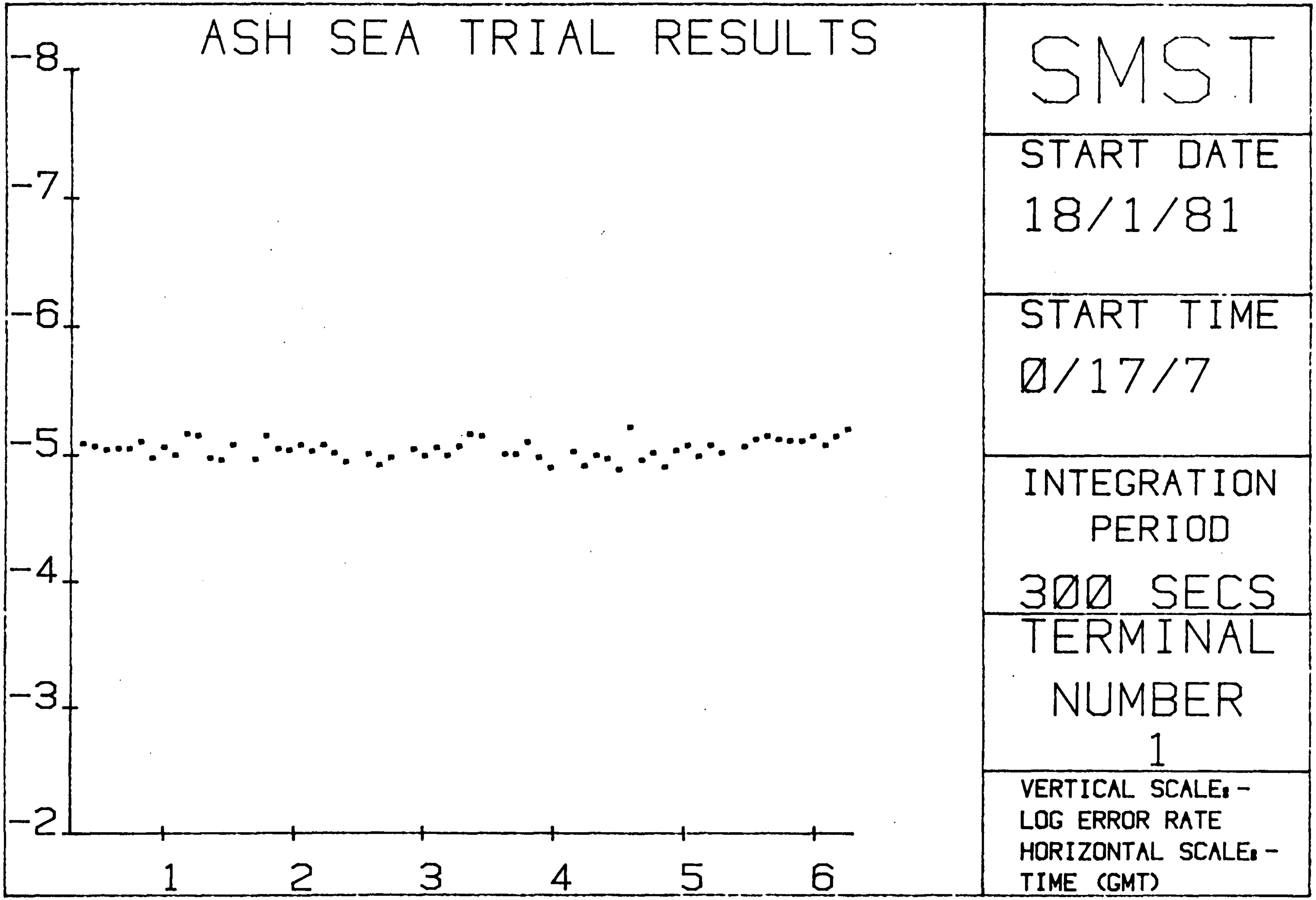
2) Time of Day. A software fault in the version 1.2 controller software causes the time of day to be updated incorrectly in passive highway controller units. This fault is characterised by a time of day which appears to be stationary.

Appendix E

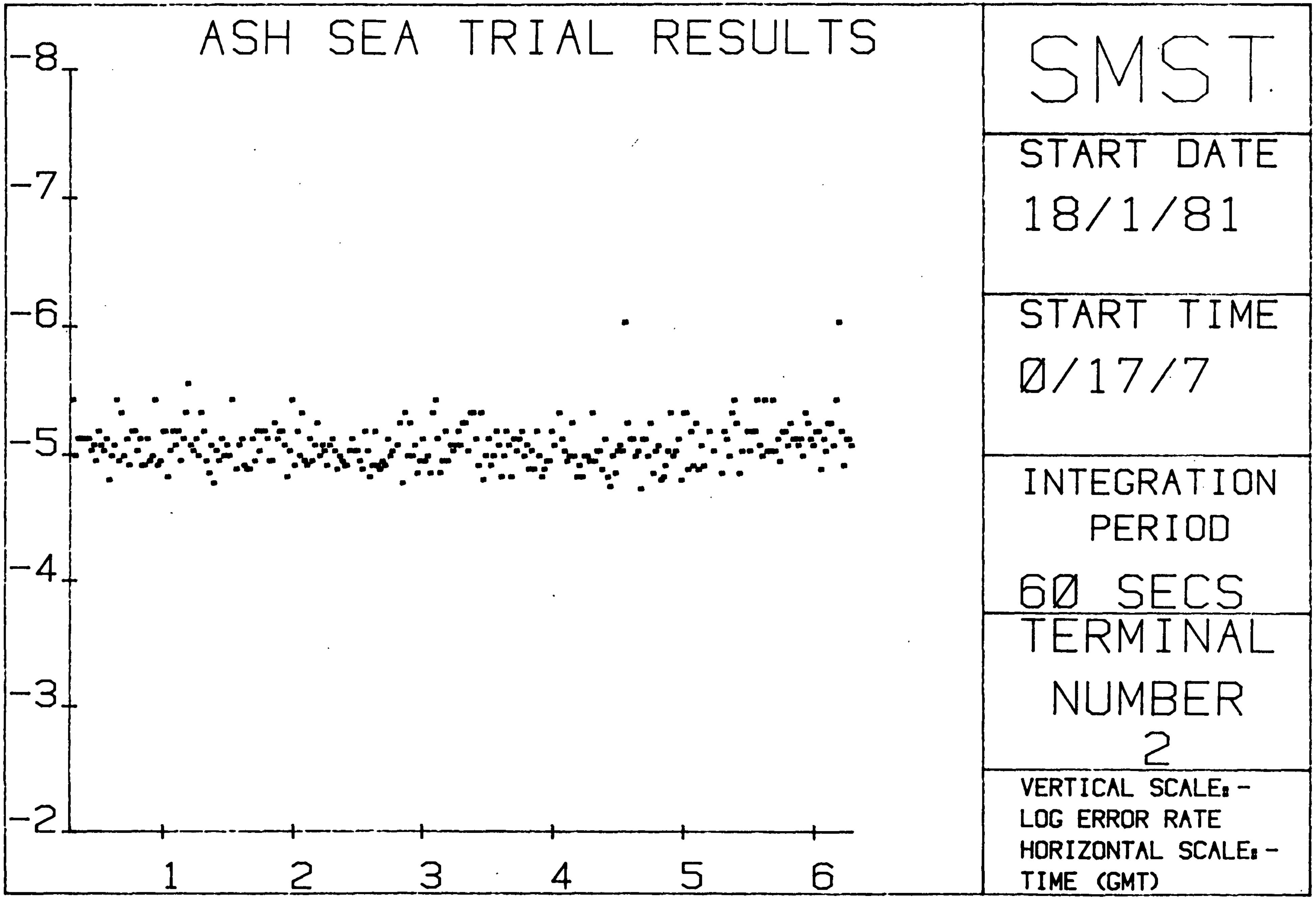
Graphs of ASH Test Results


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
1
2
3
4
5
6
SMST
START DATE
18/1/81
START TIME
Ø/17/7
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
1
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

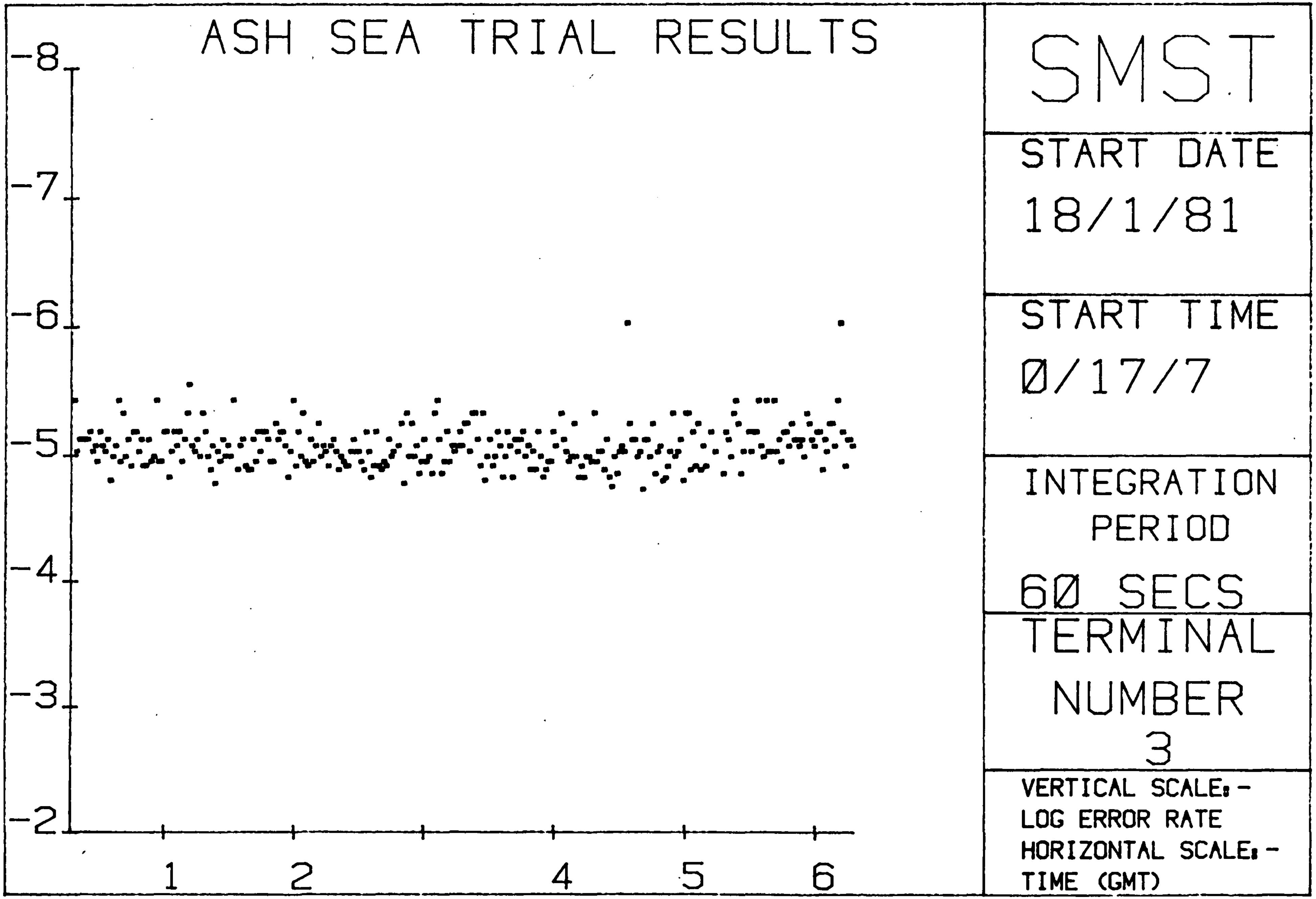
-Graph 1-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
1
2
3
4
5
6
SMST
START DATE
18/1/81
START TIME
Ø/17/7
INTEGRATION PERIOD
300 SECS
TERMINAL NUMBER
1
VERTICAL SCALE: -
LOG ERROR RATE
HORIZONTAL SCALE: -
TIME (GMT)

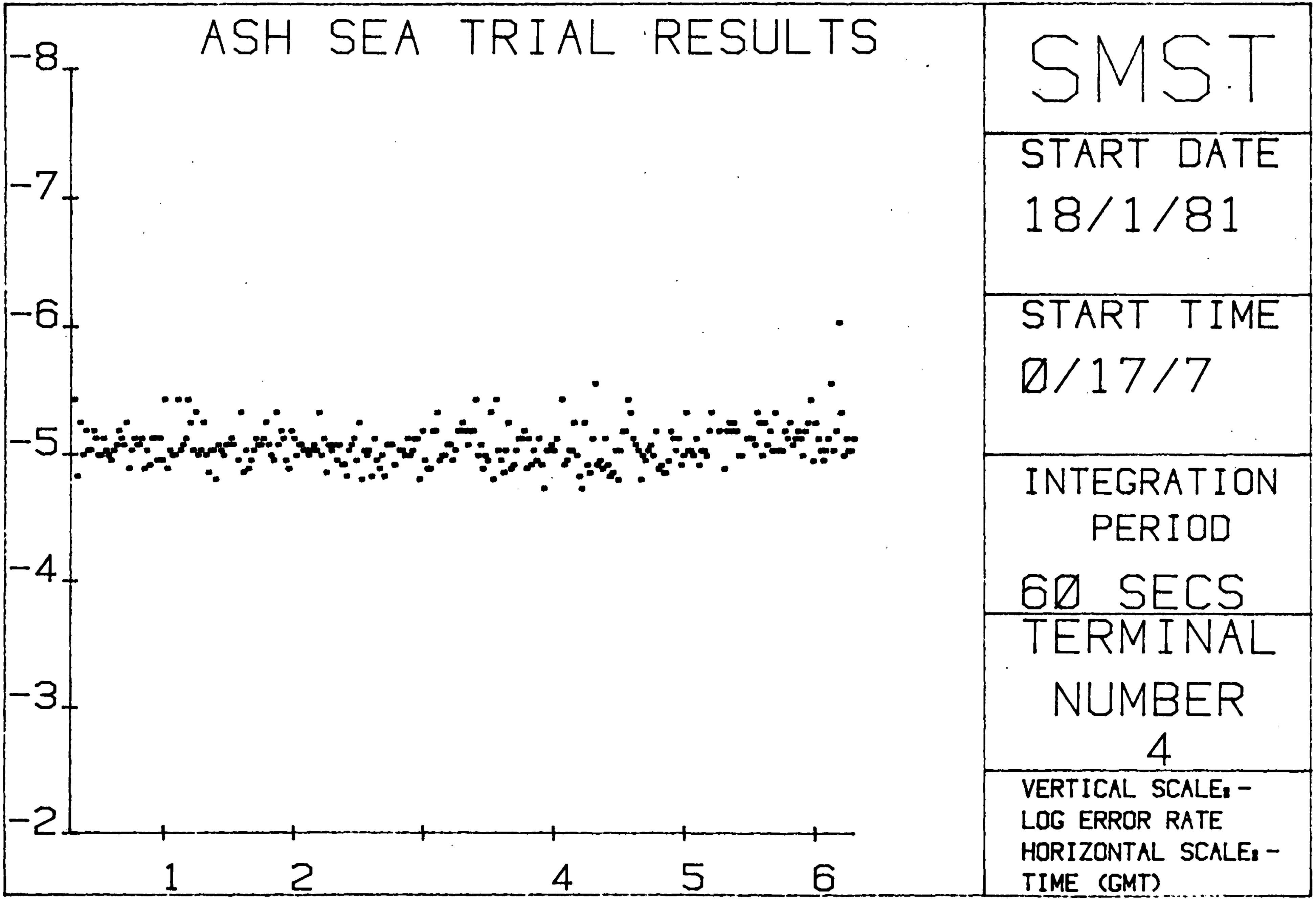
-Graph 2-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
1
2
3
4
5
6
SMST
START DATE
18/1/81
START TIME
Ø/17/7
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
2
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

-Graph 3-

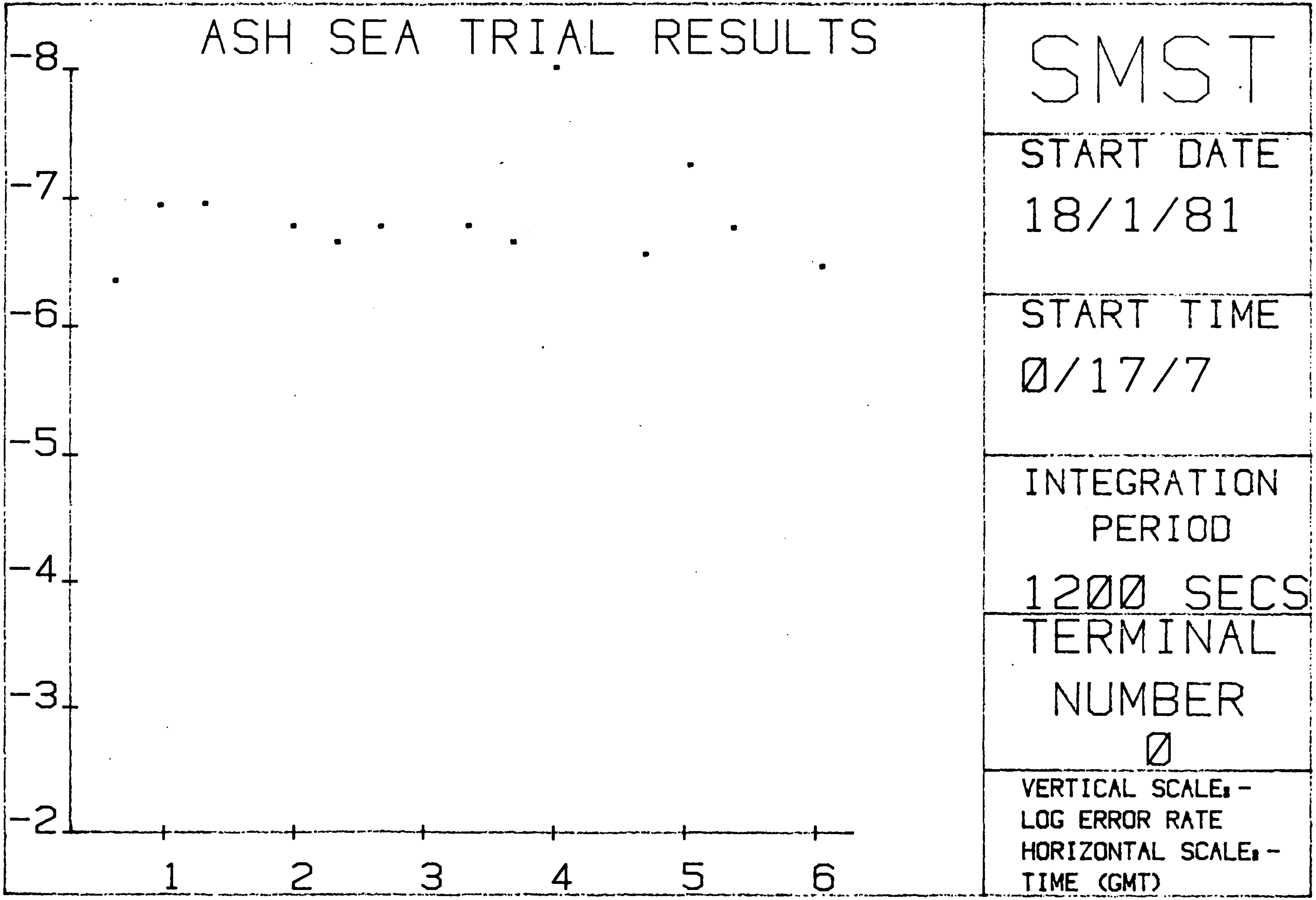

ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
1
2
4
5
6
SMST
START DATE
18/1/81
START TIME
Ø/17/7
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
3
VERTICAL SCALE:-
LOG ERROR RATE
HORIZONTAL SCALE:-
TIME (GMT)

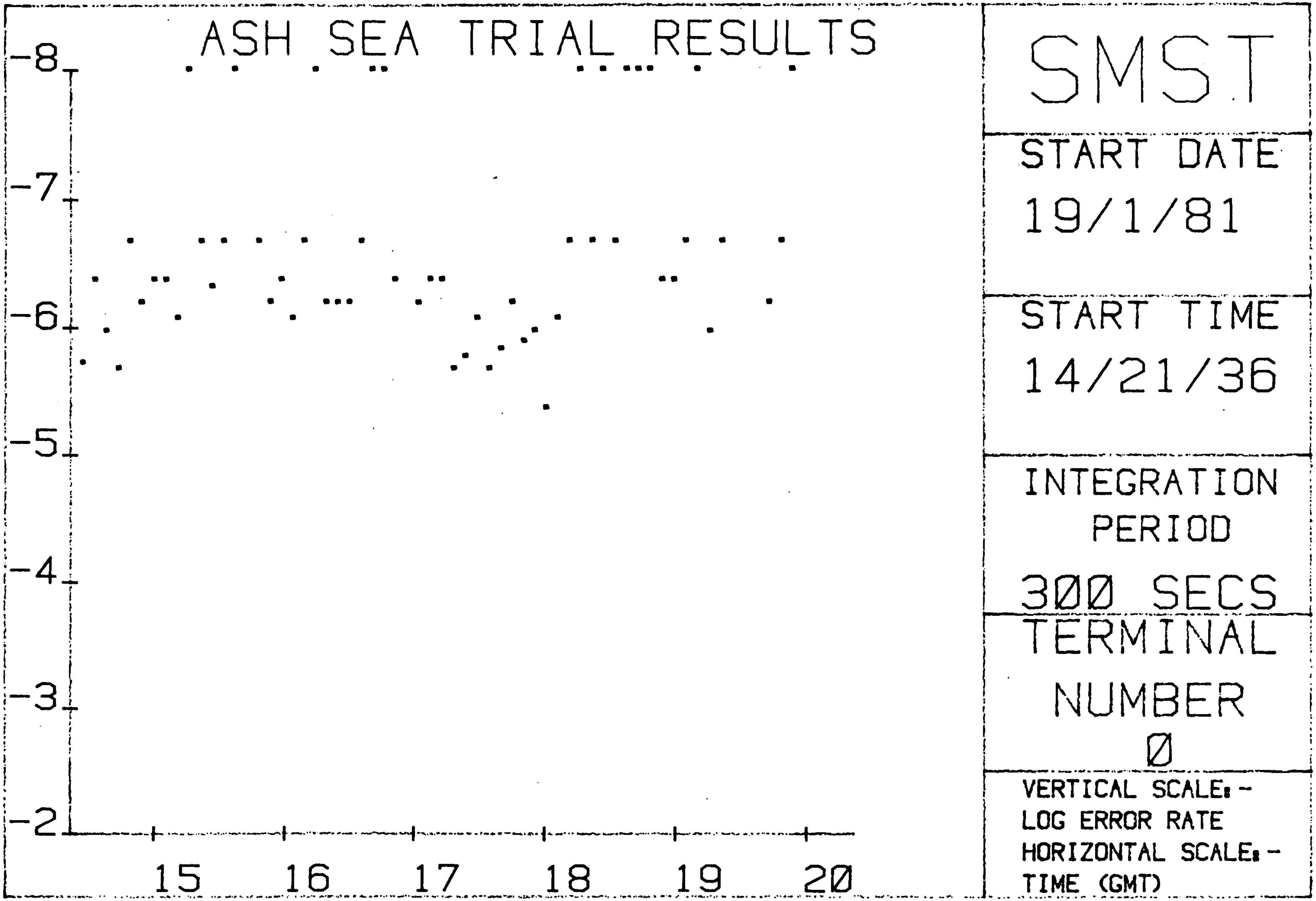
-Graph 4-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
1
2
4
5
6
SMST
START DATE
18/1/81
START TIME
Ø/17/7
INTEGRATION
PERIOD
6Ø SECS
TERMINAL
NUMBER
4
VERTICAL SCALE:-
LOG ERROR RATE
HORIZONTAL SCALE:-
TIME (GMT)

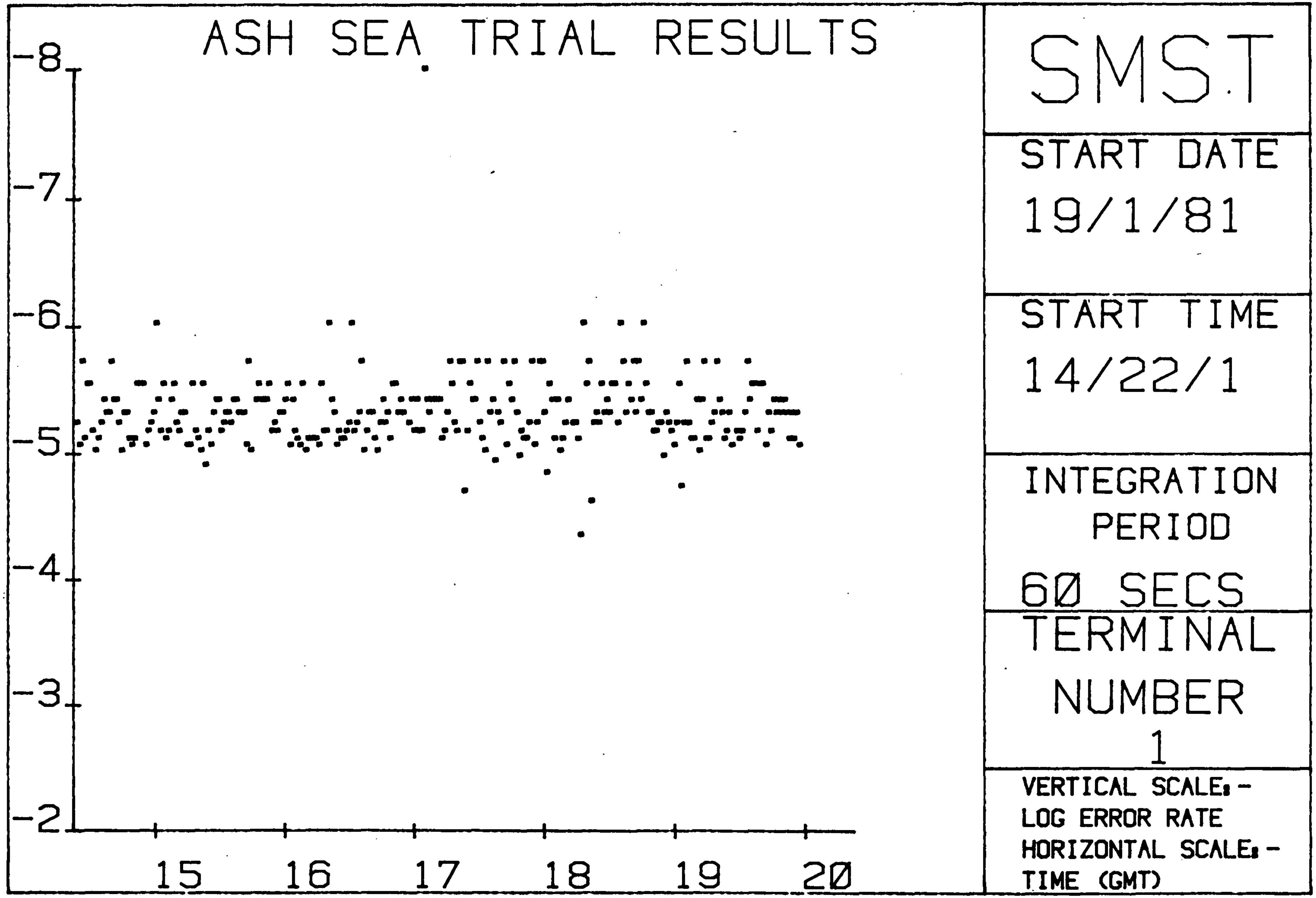
-Graph 5-

-Graph 6-

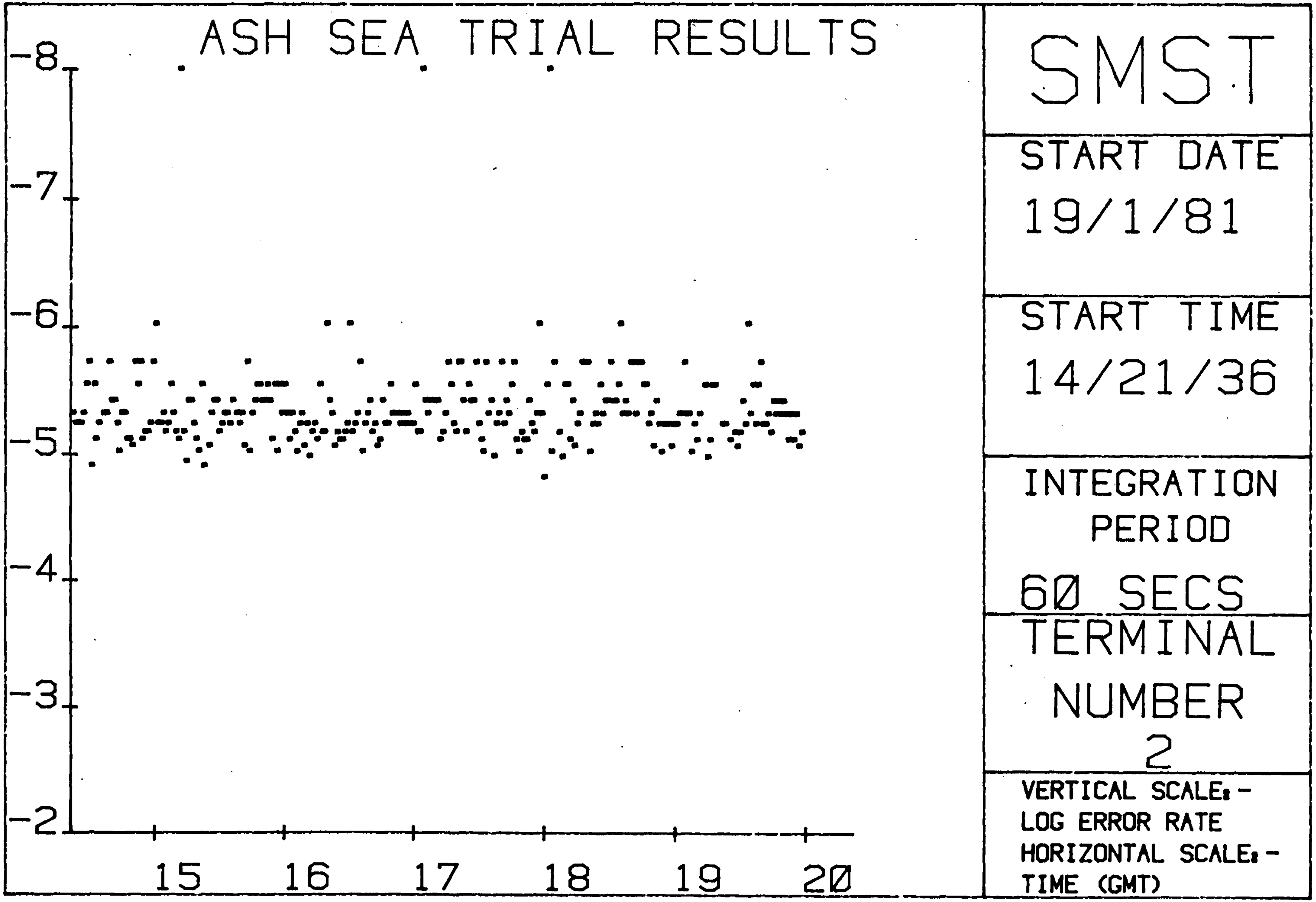

ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
1
2
3
4
5
6
SMST
START DATE
18/1/81
START TIME
Ø/17/7
INTEGRATION PERIOD
12ØØ SECS
TERMINAL NUMBER
Ø
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
15
16
17
18
19
20
SMST
START DATE
19/1/81
START TIME
14/21/36
INTEGRATION PERIOD
300 SECS
TERMINAL NUMBER
0
VERTICAL SCALE: -
LOG ERROR RATE
HORIZONTAL SCALE: -
TIME (GMT)

-Graph 7-

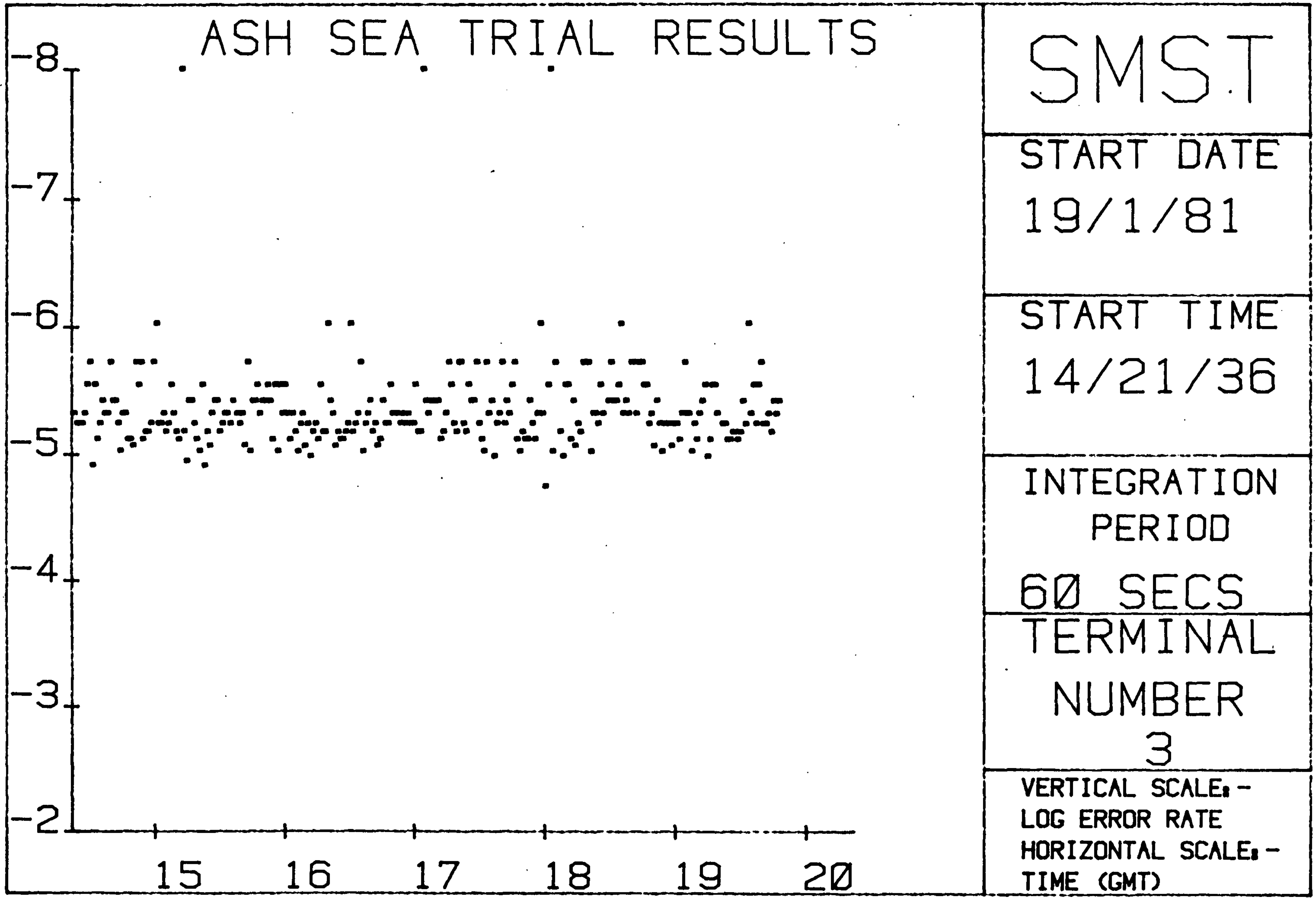

ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
15
16
17
18
19
20
SMST
START DATE
19/1/81
START TIME
14/22/1
INTEGRATION PERIOD
60 SECS
TERMINAL NUMBER
1
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

-Graph 8-

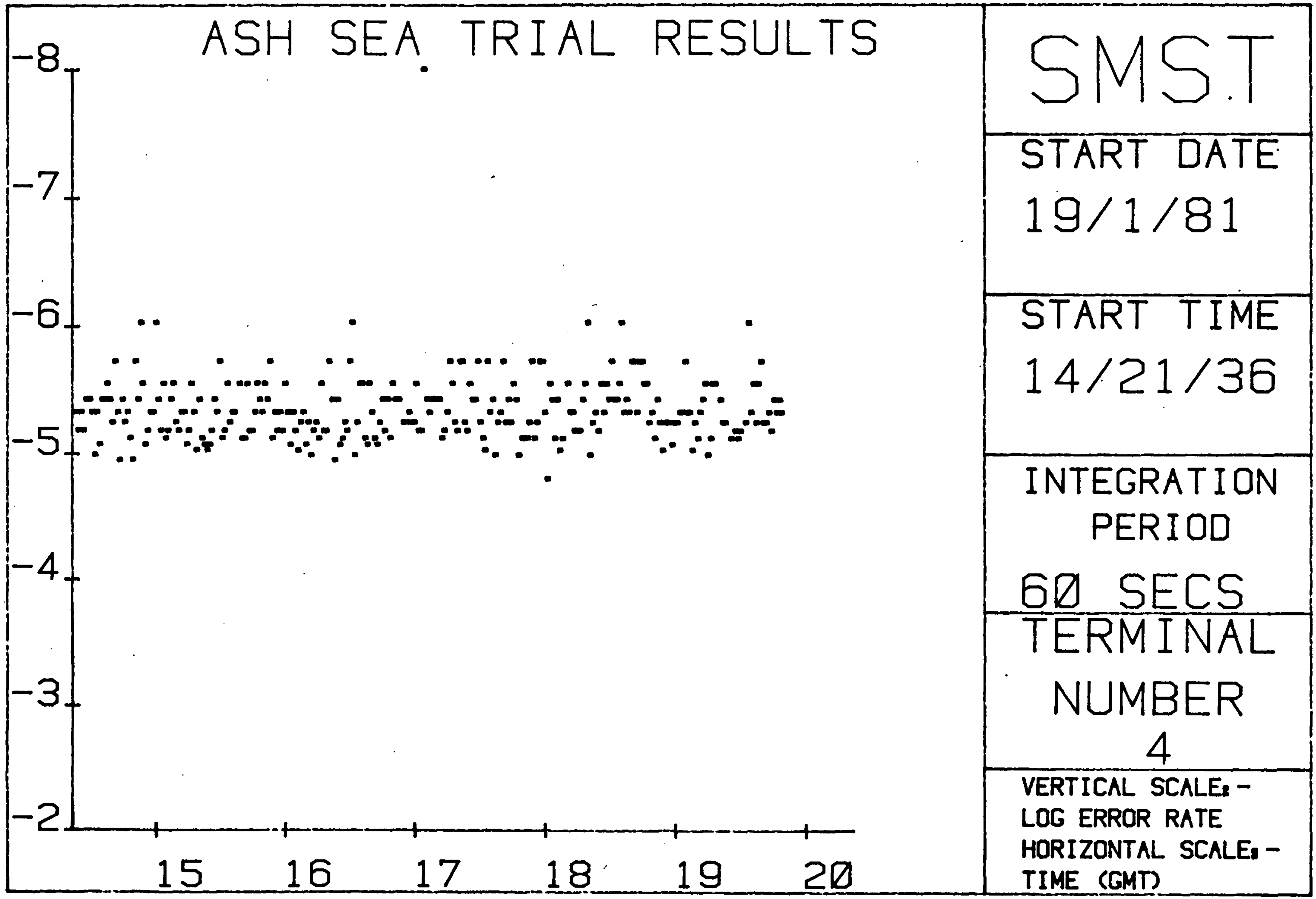


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
15
16
17
18
19
20
SMST
START DATE
19/1/81
START TIME
14/21/36
INTEGRATION PERIOD
60 SECS
TERMINAL NUMBER
2
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

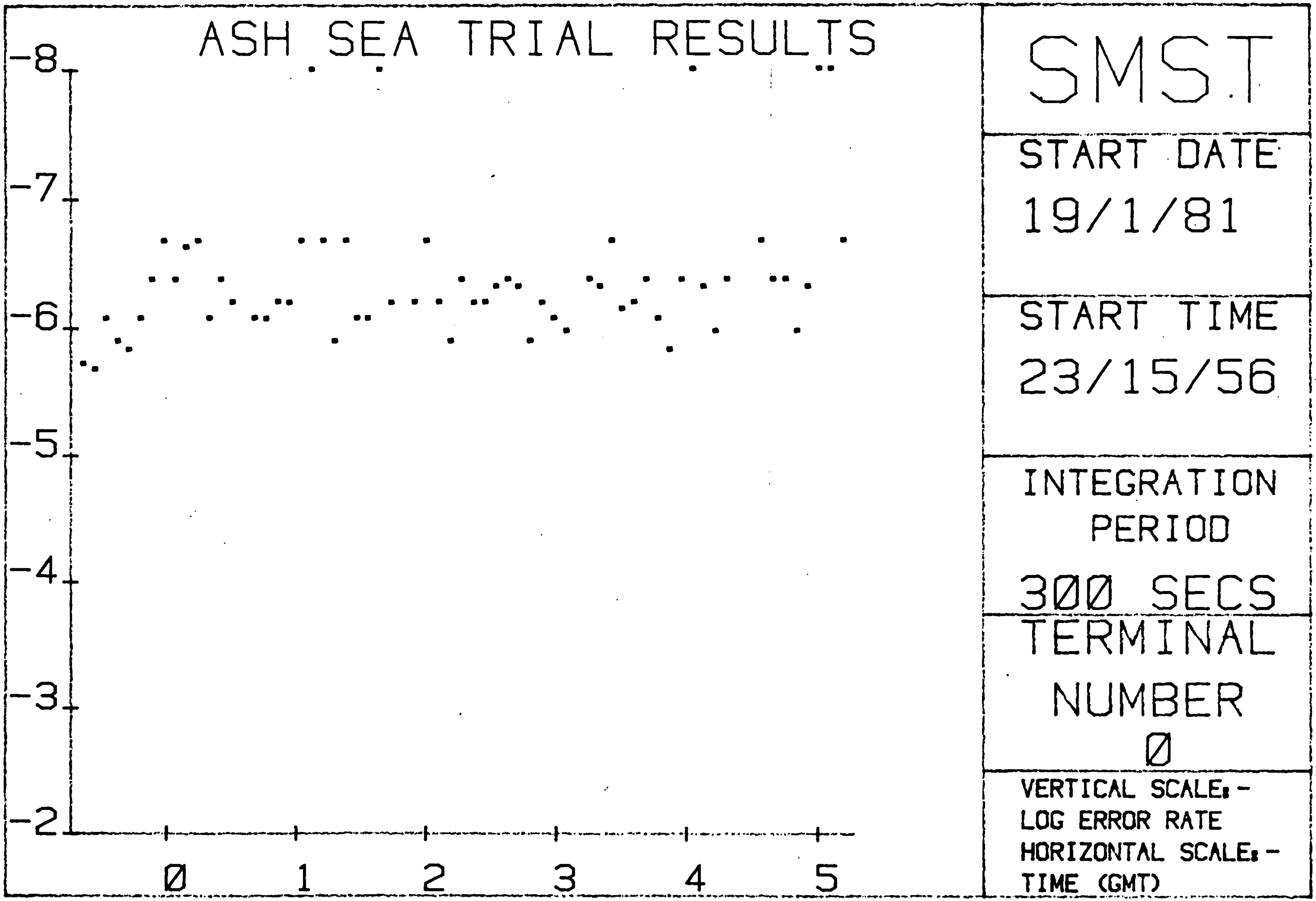
-Graph 9-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
15
16
17
18
19
20
SMST
START DATE
19/1/81
START TIME
14/21/36
INTEGRATION PERIOD
60 SECS
TERMINAL NUMBER
3
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

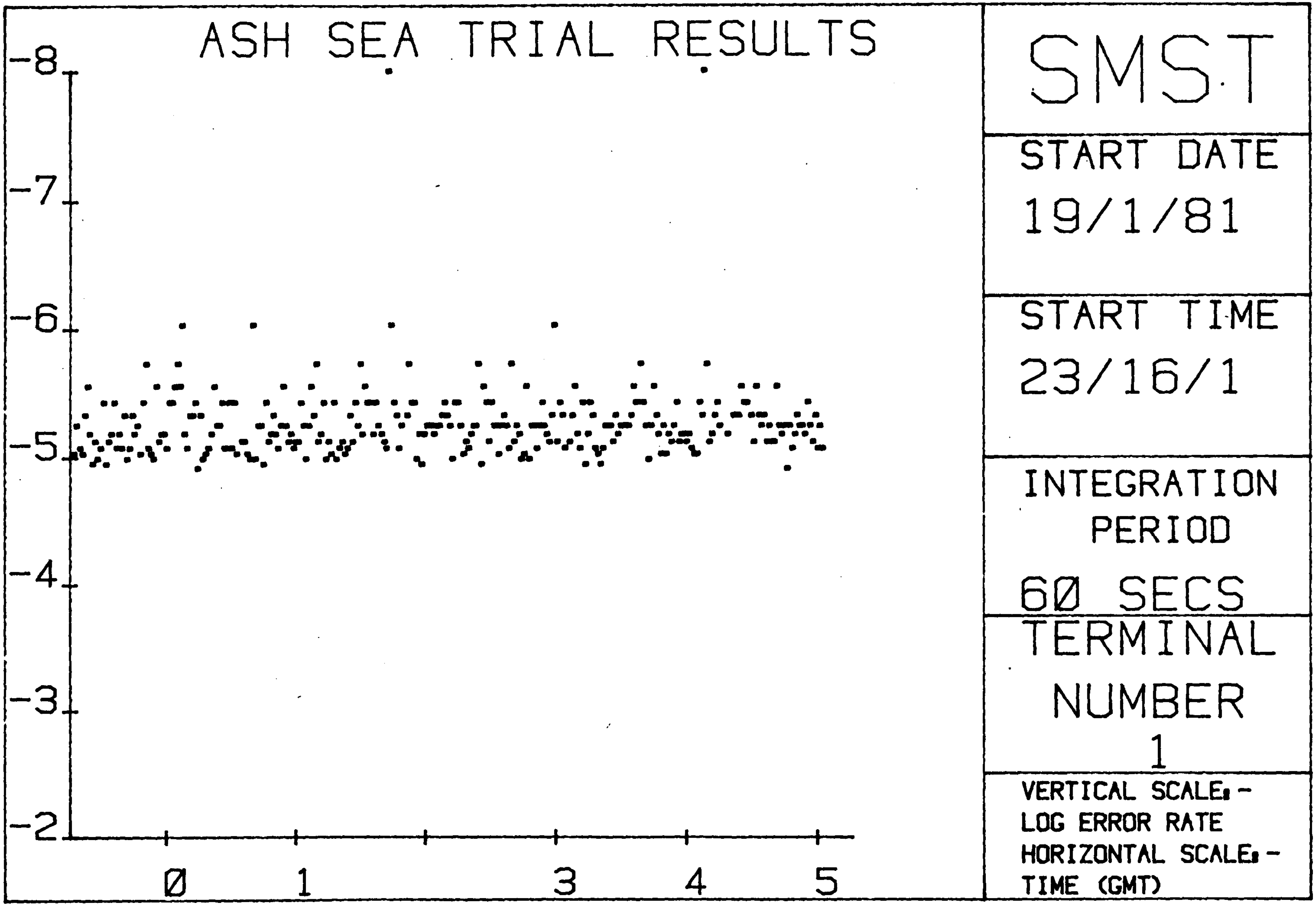
-Graph 10-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
15
16
17
18
19
20
SMST
START DATE
19/1/81
START TIME
14/21/36
INTEGRATION PERIOD
60 SECS
TERMINAL NUMBER
4
VERTICAL SCALE: -
LOG ERROR RATE
HORIZONTAL SCALE: -
TIME (GMT)

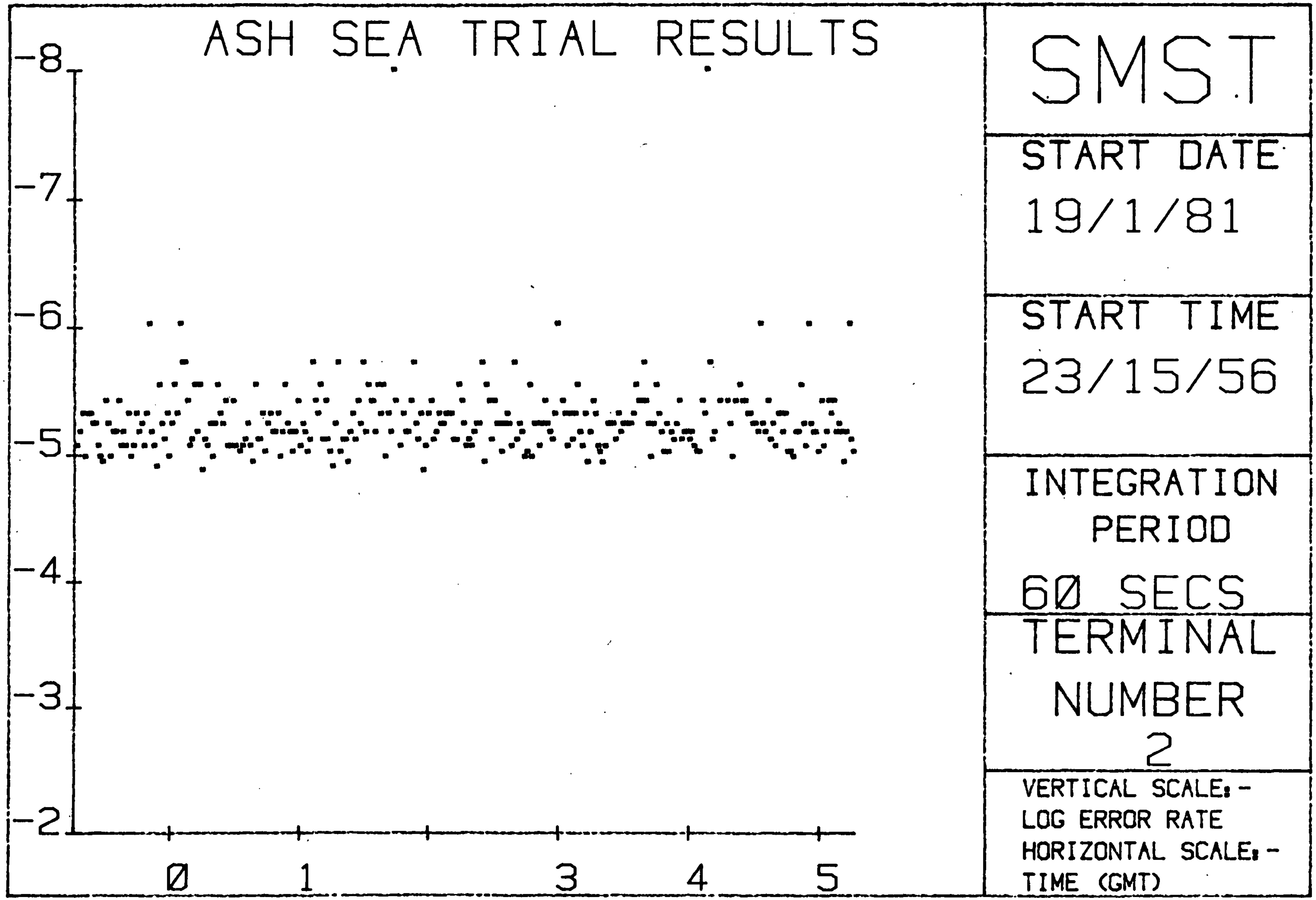
-Graph 11-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
Ø
1
2
3
4
5
SMST
START DATE
19/1/81
START TIME
23/15/56
INTEGRATION PERIOD
300 SECS
TERMINAL NUMBER
Ø
VERTICAL SCALE:-
LOG ERROR RATE
HORIZONTAL SCALE:-
TIME (GMT)

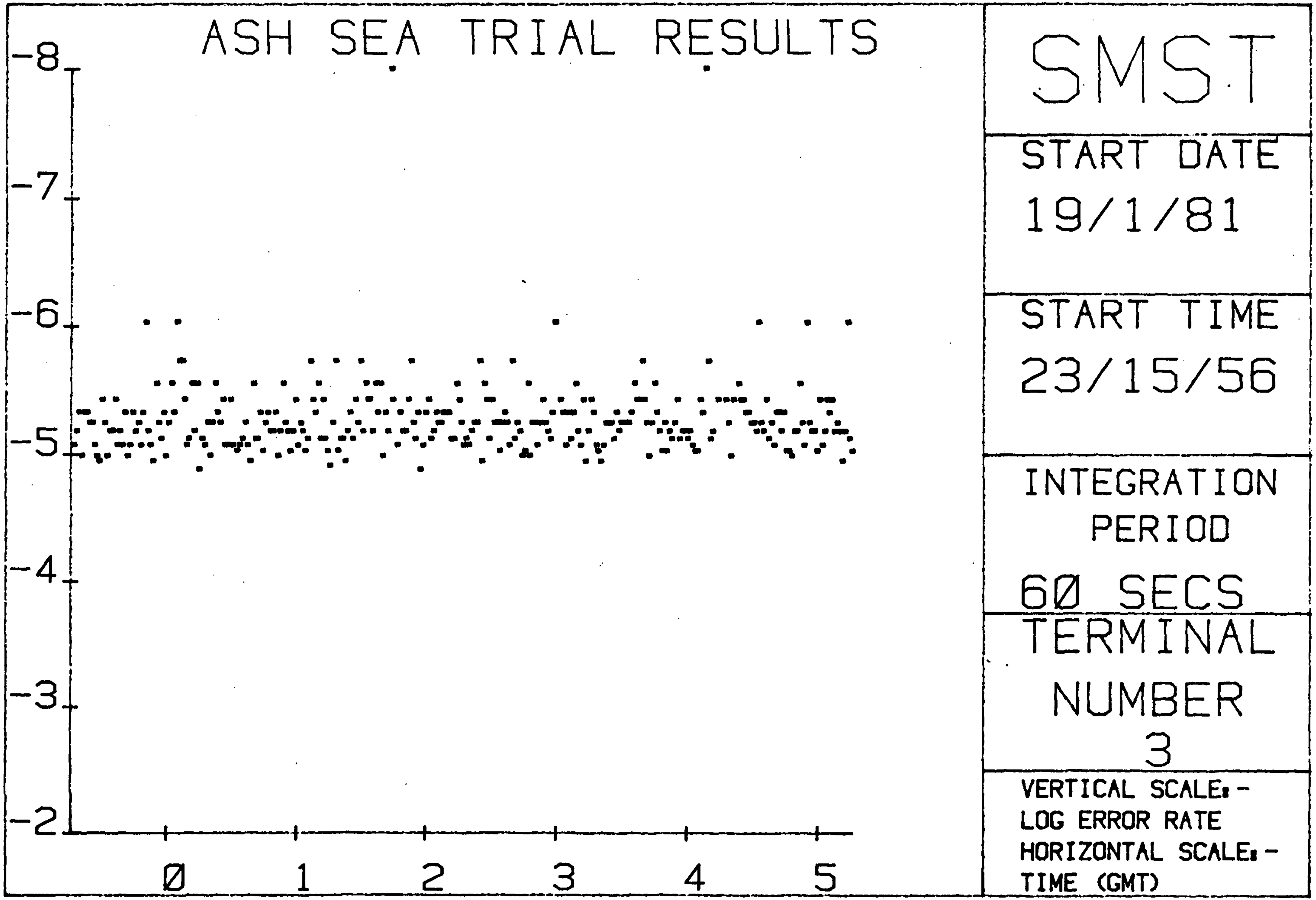
-Graph 12-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
Ø
1
3
4
5
SMST
START DATE
19/1/81
START TIME
23/16/1
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
1
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

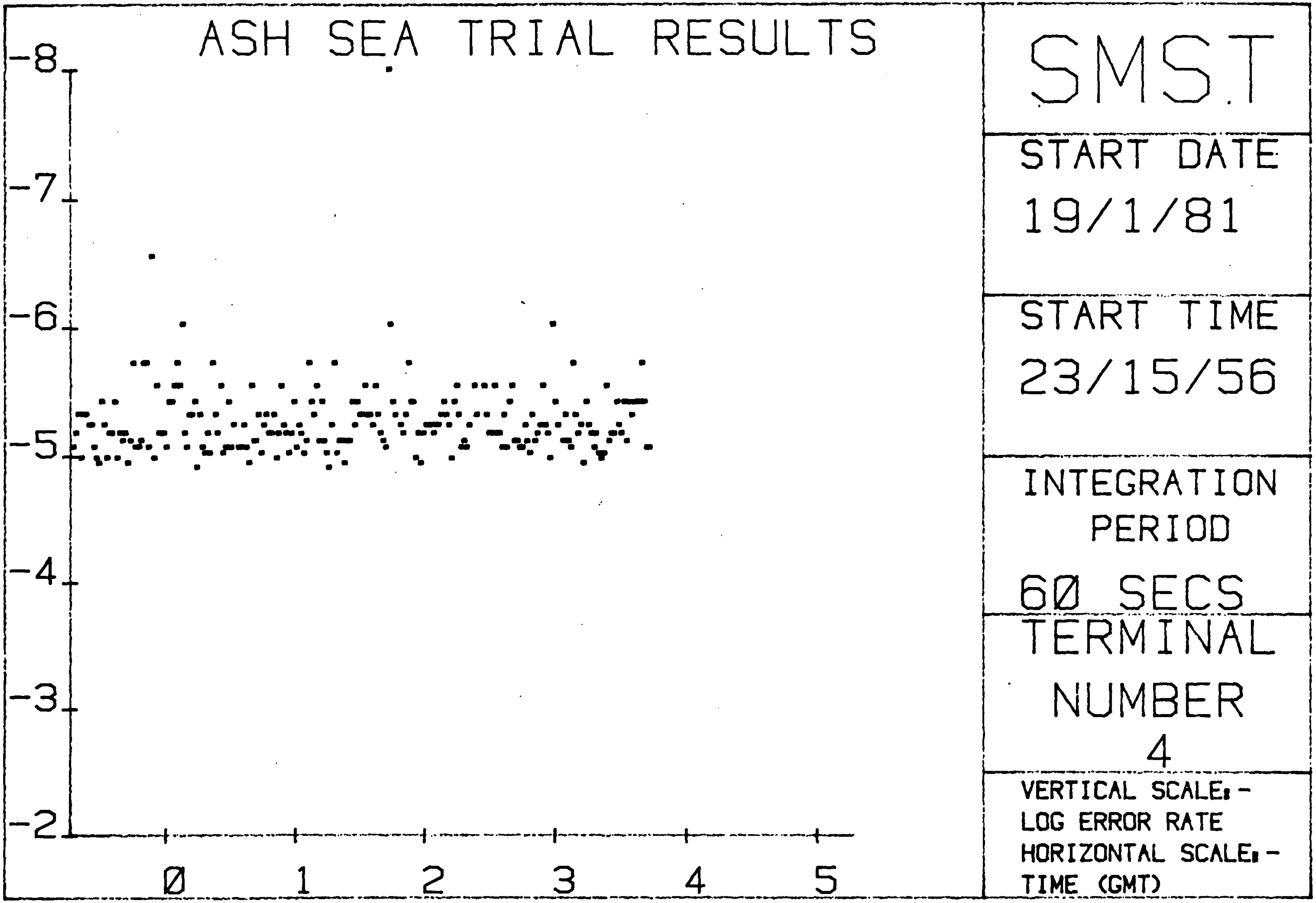
-Graph 13-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
Ø
1
3
4
5
SMST
START DATE
19/1/81
START TIME
23/15/56
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
2
VERTICAL SCALE:-
LOG ERROR RATE
HORIZONTAL SCALE:-
TIME (GMT)

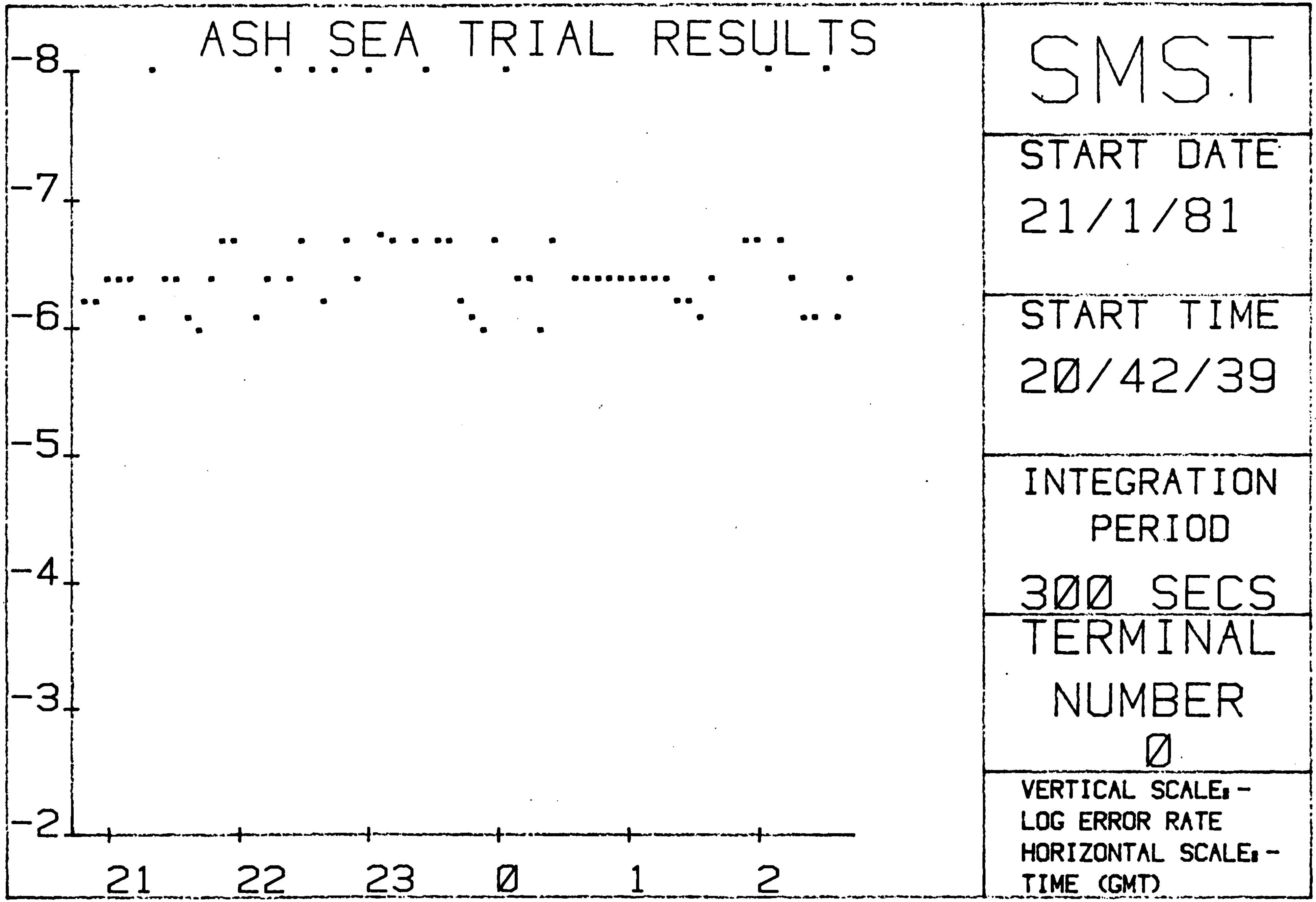
-Graph 14-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
Ø
1
2
3
4
5
SMST
START DATE
19/1/81
START TIME
23/15/56
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
3
VERTICAL SCALE: -
LOG ERROR RATE
HORIZONTAL SCALE: -
TIME (GMT)

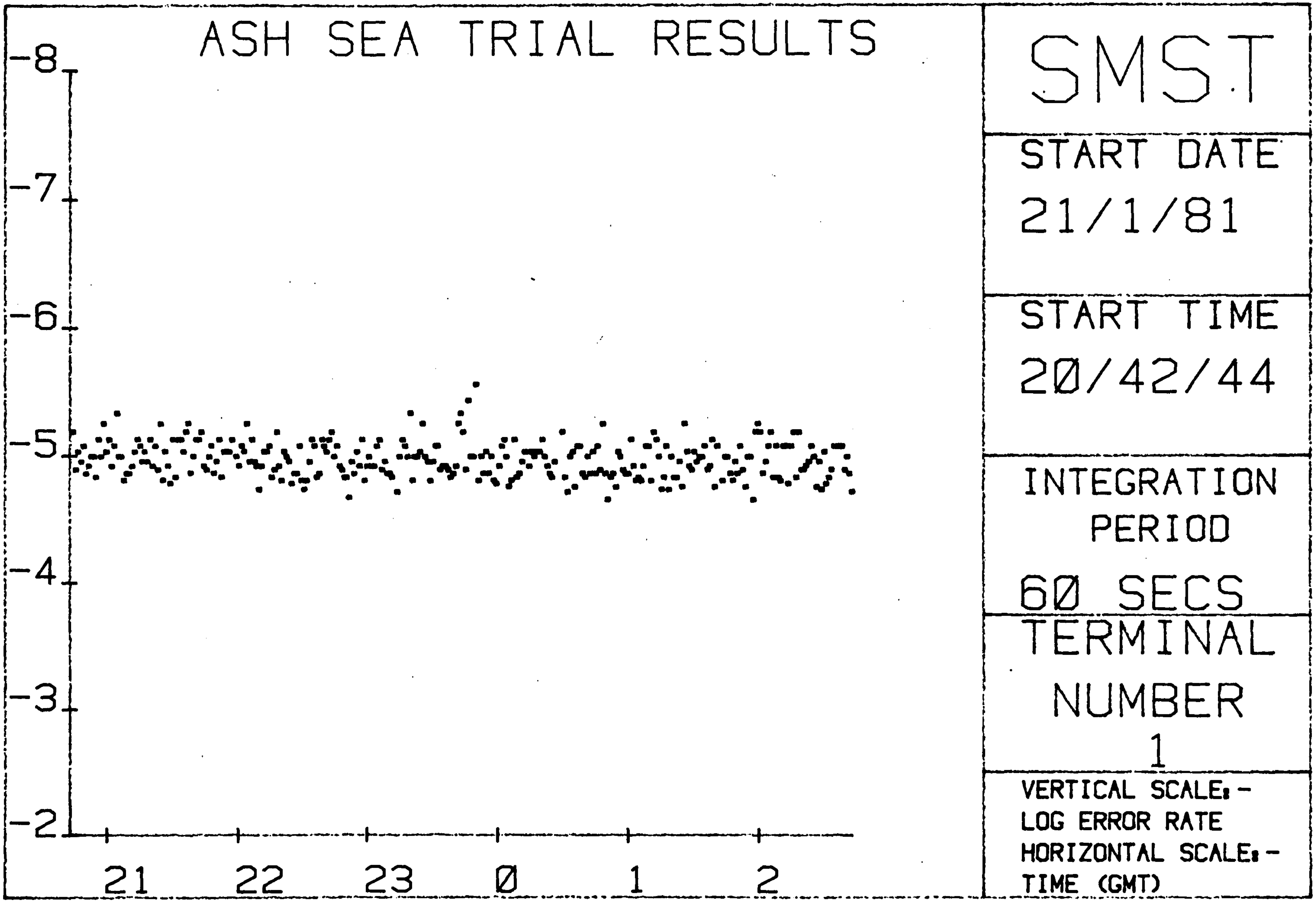
-Graph 15-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
Ø
1
2
3
4
5
SMS.T
START DATE
19/1/81
START TIME
23/15/56
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
4
VERTICAL SCALE: -
LOG ERROR RATE
HORIZONTAL SCALE: -
TIME (GMT)

-Graph 16-

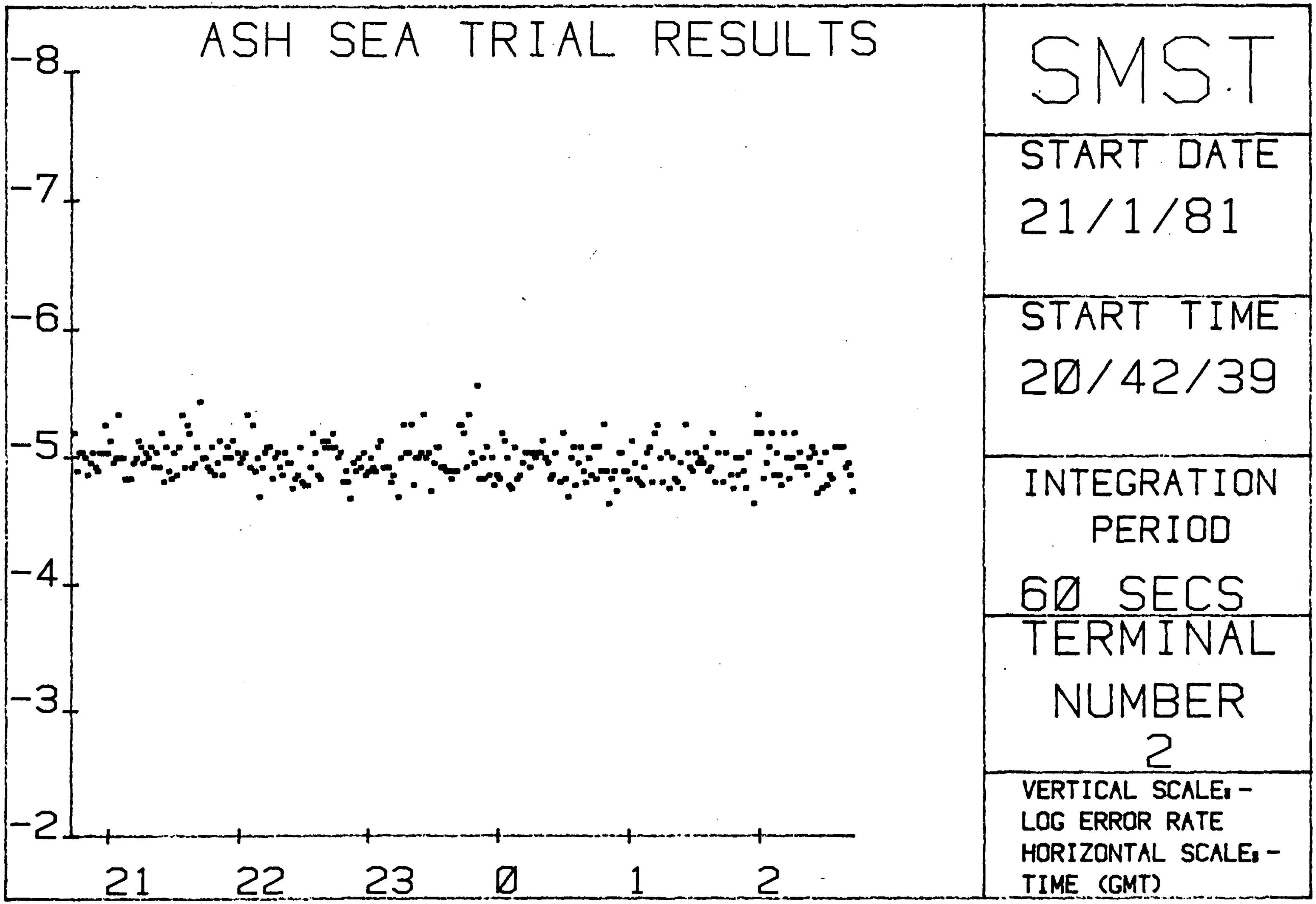

ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
21
22
23
Ø
1
2
SMST
START DATE
21/1/81
START TIME
2Ø/42/39
INTEGRATION PERIOD
3ØØ SECS
TERMINAL NUMBER
Ø
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

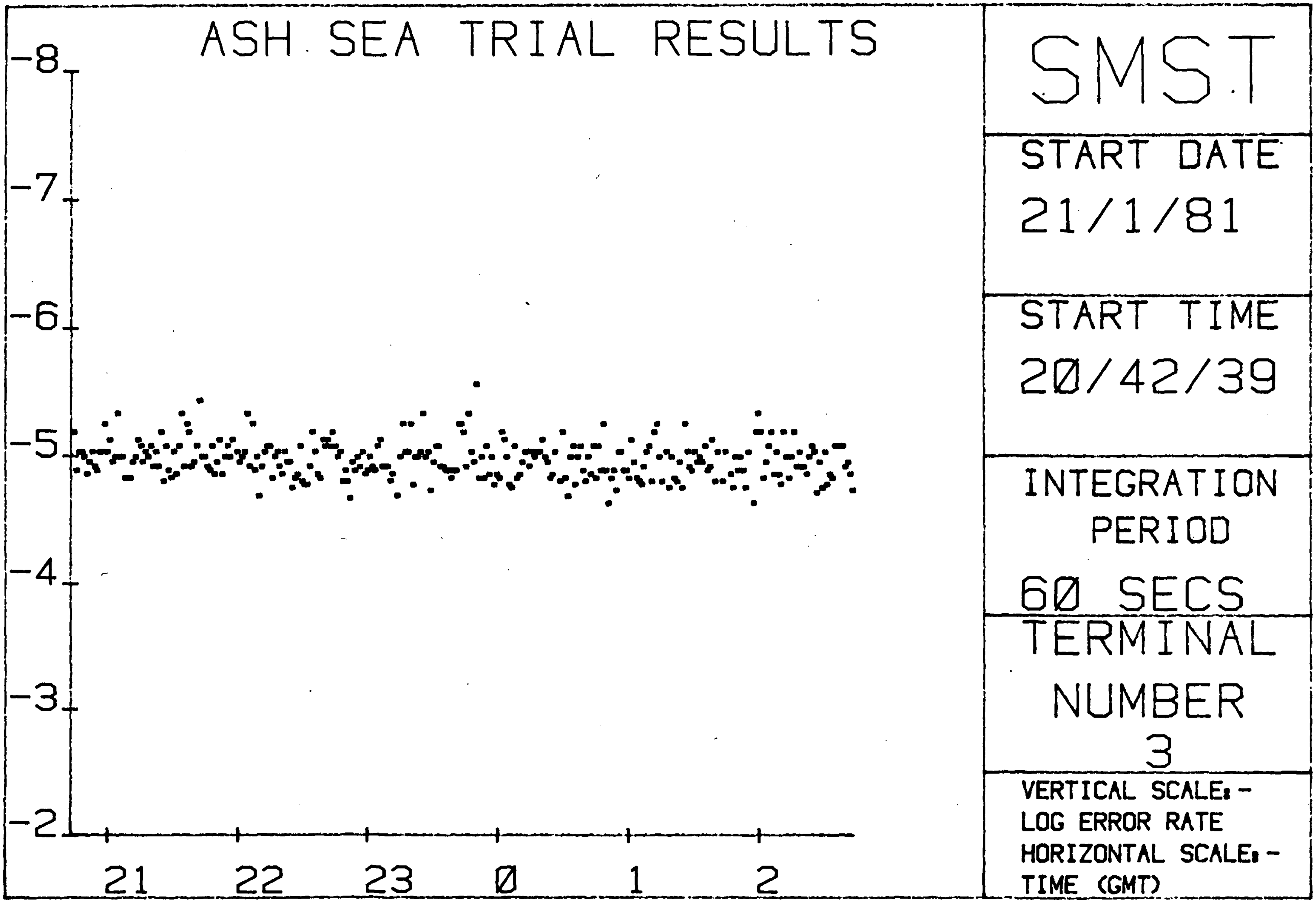
-Graph 17-

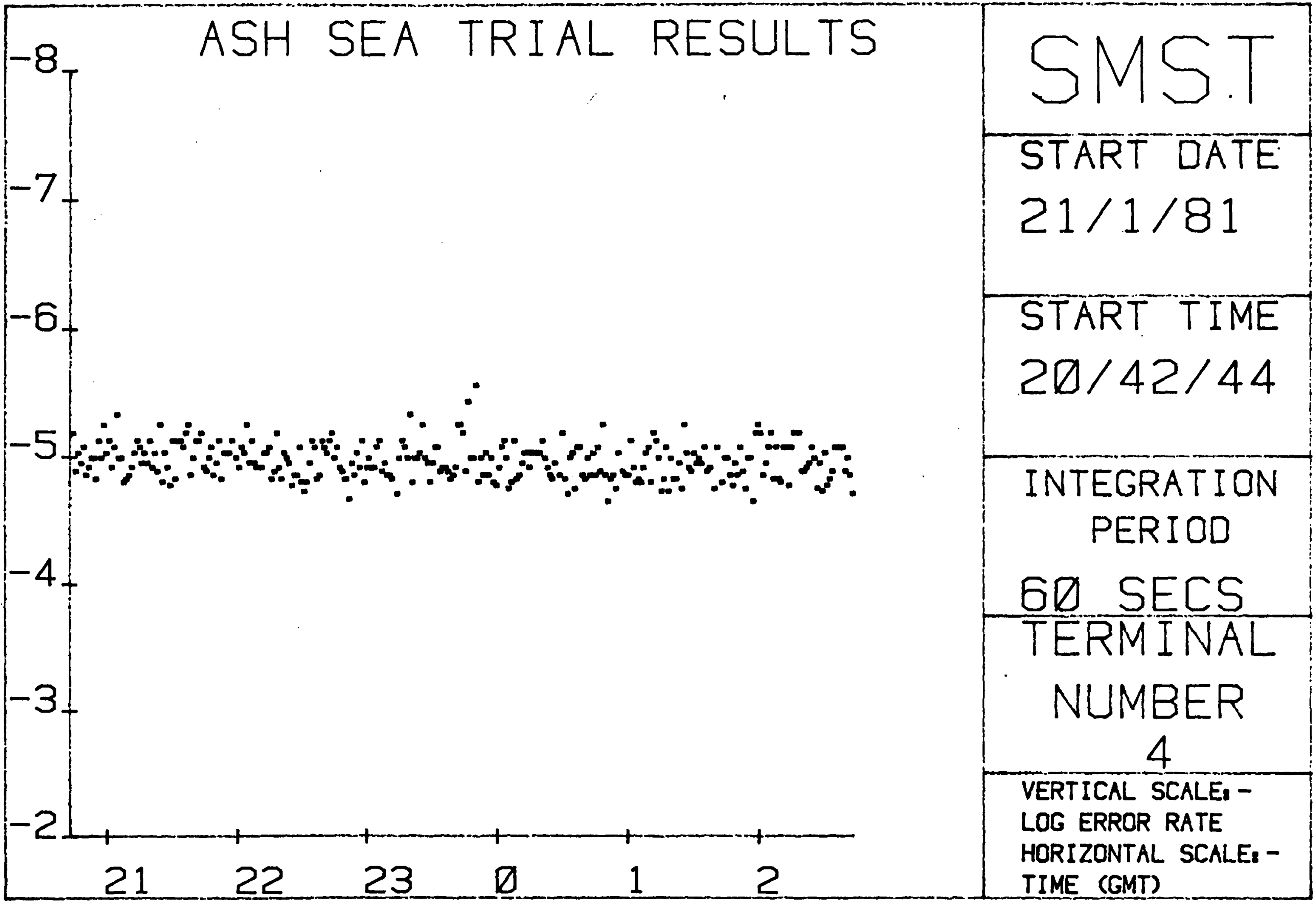


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
21
22
23
Ø
1
2
SMST
START DATE
21/1/81
START TIME
2Ø/42/44
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
1
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

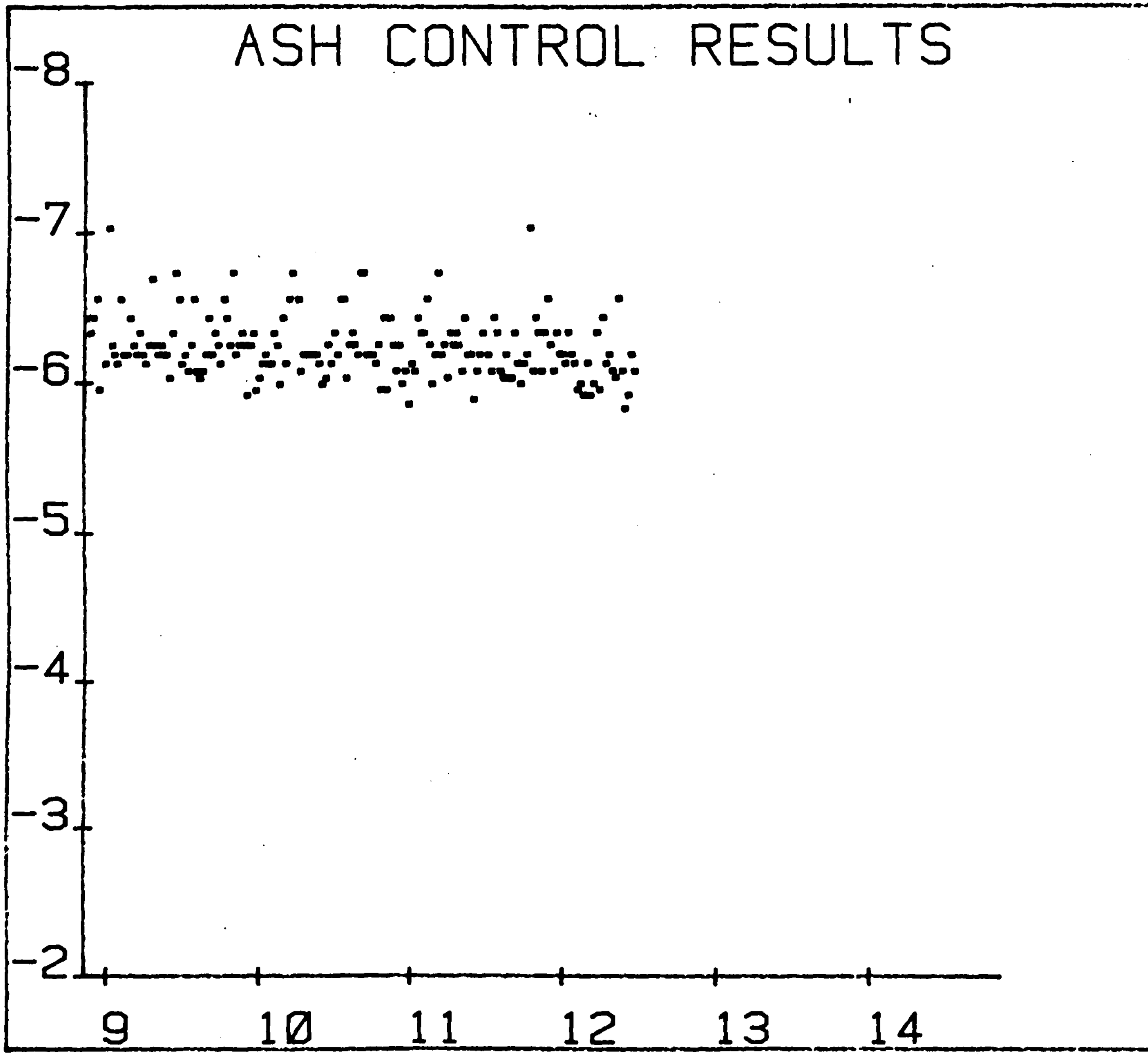
-Graph 18-


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
21
22
23
Ø
1
2
SMST
START DATE
21/1/81
START TIME
2Ø/42/39
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
2
VERTICAL SCALE: -
LOG ERROR RATE
HORIZONTAL SCALE: -
TIME (GMT)


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
21
22
23
Ø
1
2
SMST
START DATE
21/1/81
START TIME
2Ø/42/39
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
3
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)


ASH SEA TRIAL RESULTS
-8
-7
-6
-5
-4
-3
-2
21
22
23
Ø
1
2
SMST
START DATE
21/1/81
START TIME
2Ø/42/44
INTEGRATION PERIOD
6Ø SECS
TERMINAL NUMBER
4
VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)

-Graph 21-



SMST

START DATE
1/2/81

START TIME
8/50/56

INTEGRATION PERIOD
60 SECS

TERMINAL NUMBER
1

VERTICAL SCALE: - LOG ERROR RATE
HORIZONTAL SCALE: - TIME (GMT)